The Art of Consistent SDN Updates

Stefan Schmid

Aalborg University

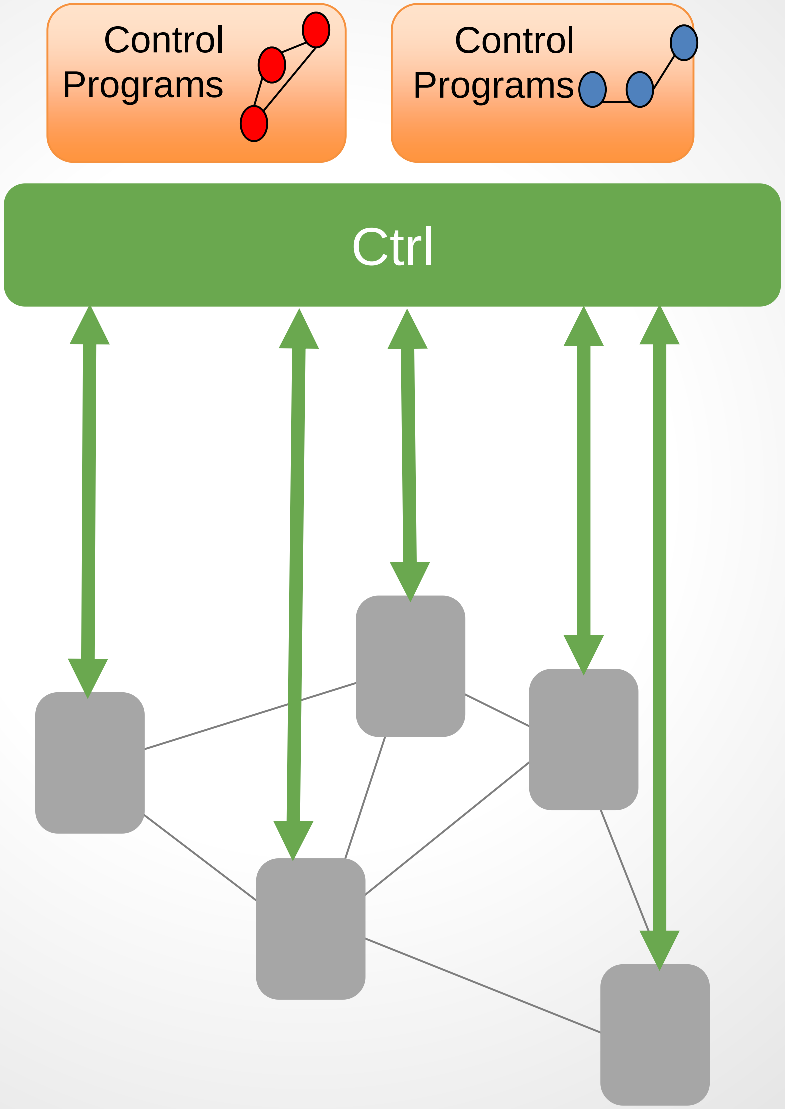
The Art of Consistent SDN Updates

Stefan Schmid

Aalborg University

Smart students in Berlin & Wroclaw:
Arne Ludwig, Jan Marcinkowski,
Szymon Dudycz, Matthias Rost,
Damien Foucard, Saeed Amiri

SDN: Algorithms *with a fundamental twist!*

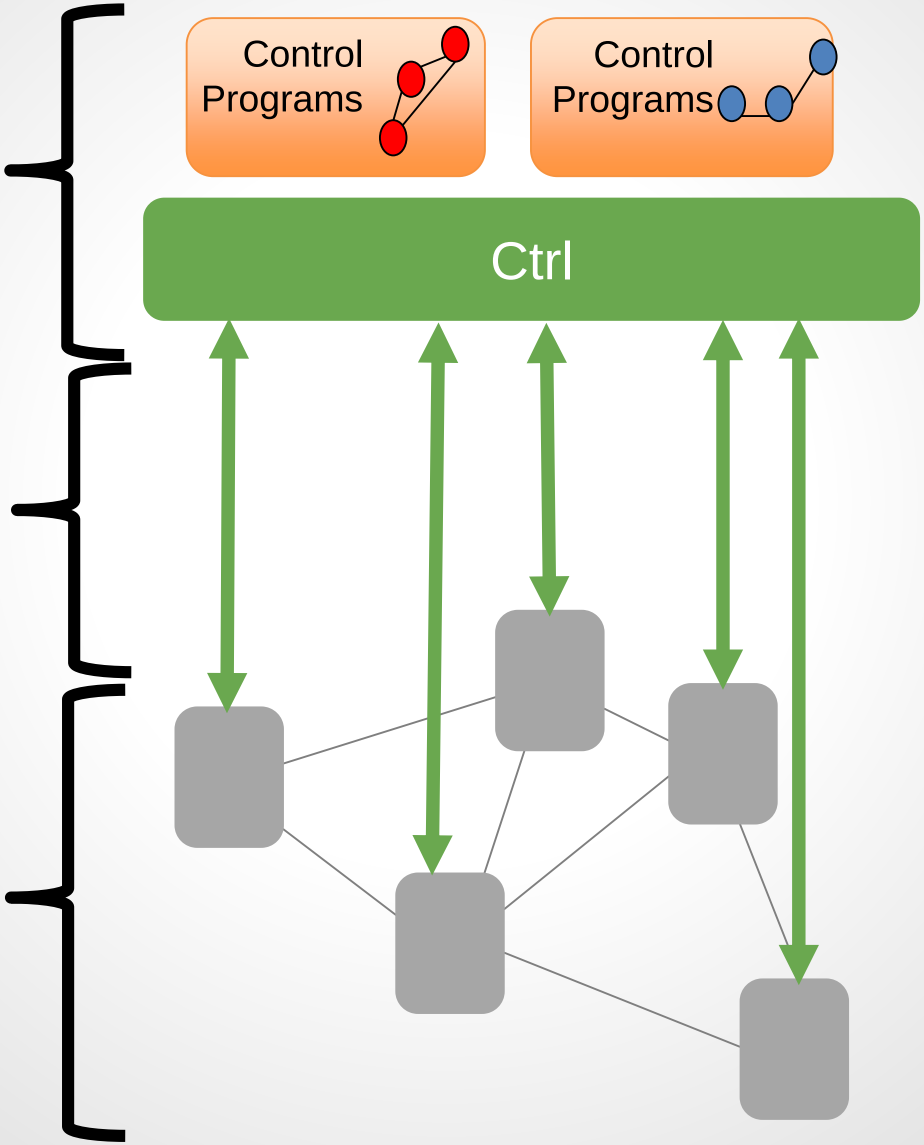


SDN: Algorithms *with a fundamental twist!*

Applications and Control Plane

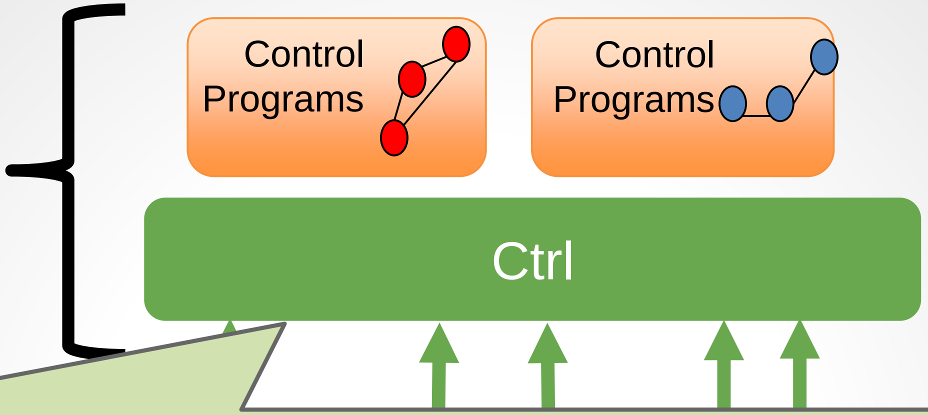
… and regarding decoupling / inter-connect!

Data Plane



SDN: Flexiblities and Constraints

Applications and Control Plane



SDN/OpenFlow is about **generality and flexibility**: in terms of **how packets are matched** (L2-L4 header fields and beyond), how **flows are defined** (fine vs coarse granular, proactive vs reactive), events can be handled **centrally vs in a distributed manner**, etc.

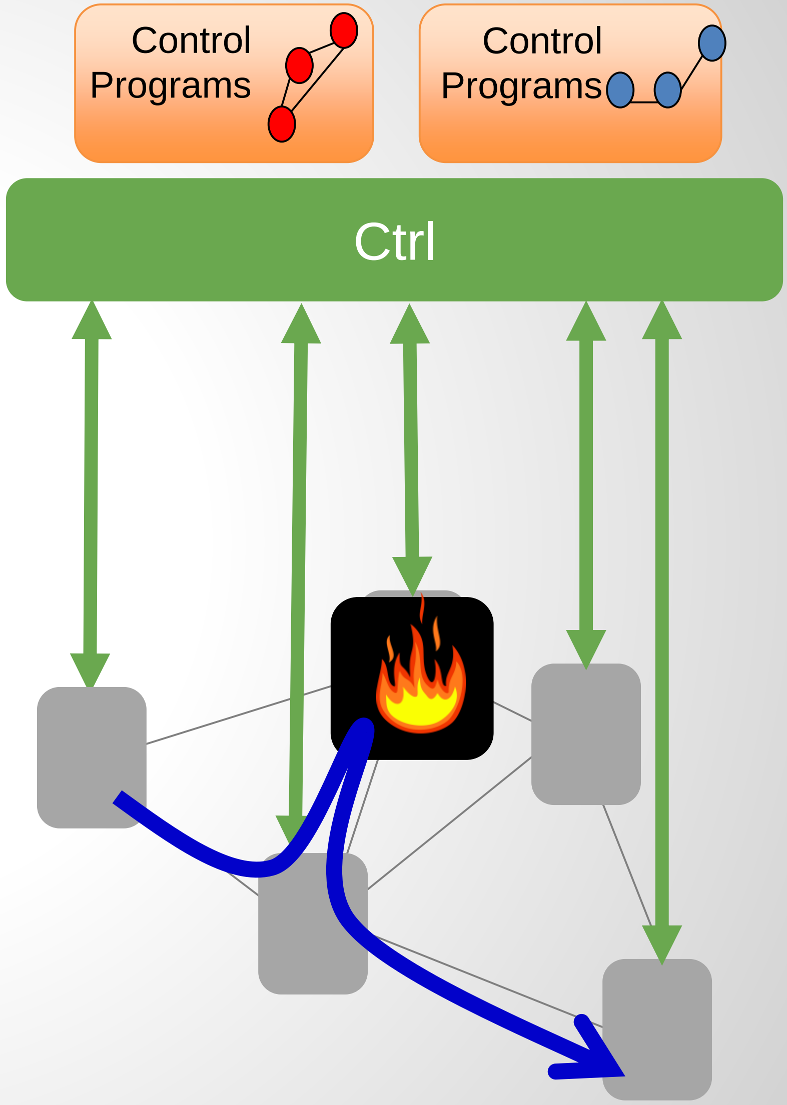
Dat

But there are also constraints and challenges: SDN is an inherently **asynchronous distributed system** (controller decoupled), switches are **simple devices** (not a Turing or even state machine!), IP-routing is prefix based, careful use of dynamic flexibilities: **don’t shoot in your foot!**

Applications: Algorithms *with a twist!*

- ❑ Let’s consider: Traffic Engineering
  - ❑ Circuit routing, **call admission**
  - ❑ Raghavan, Wolsey, Awerbuch, etc.

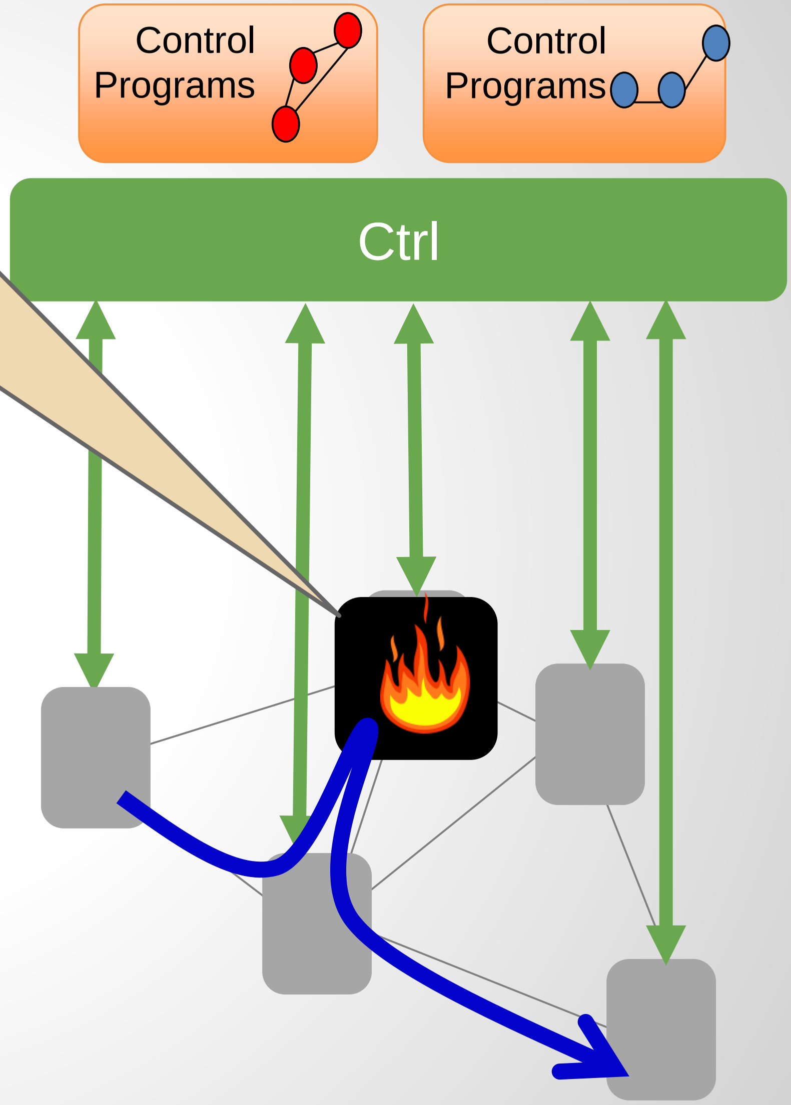
- ❑ *SDN twist*: more general/flexible!
  - ❑ **Non-shortest** paths and more
  - ❑ Enables **complex network services:** steer traffic through middleboxes i.e. **waypoints** (firewall, proxy etc.): paths may contain **loops**!
  - ❑ More than independent routing per segment: **none-or-all segment admission control, joint optimization**
  - ❑ E.g., LP relaxation (Raghavan et al.): how to **randomly round and decompose complex requests**?



Applications: Algorithms *with a twist!*

Optionally *NFV twist*: where to place NFV (or hybrid SDN)? Facility location / capacitated dominating set, *but*: not distance to but distance *via function(s)* matters!

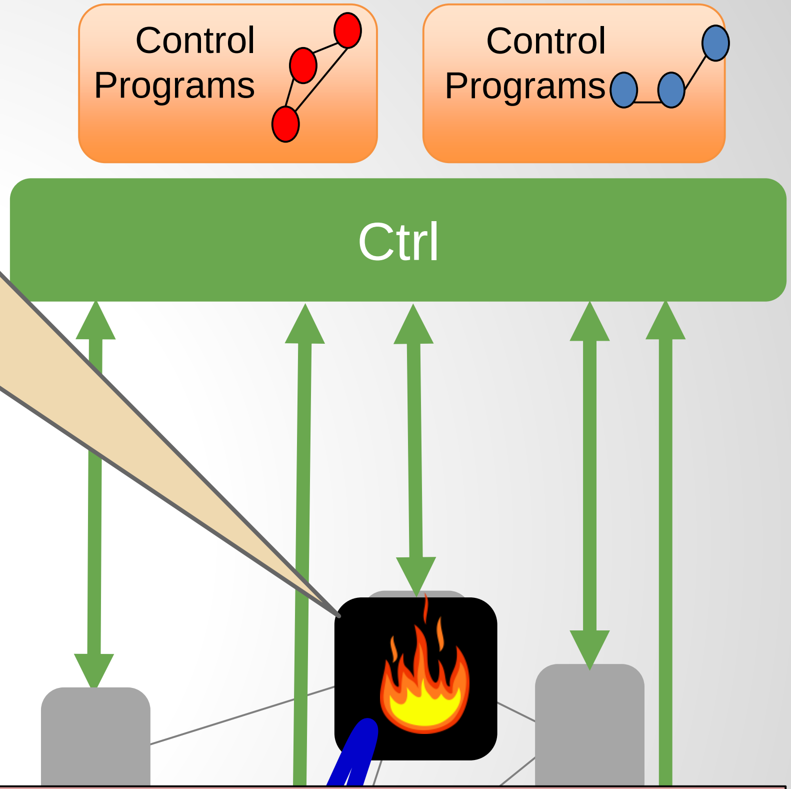
- ❑ Le
  - ❑
  - ❑
- ❑ SD
  - ❑ **Non-shortest** paths
  - ❑ Enables **complex network services:** steer traffic through middleboxes i.e. **waypoints** (firewall, proxy etc.): paths may contain **loops**!
  - ❑ More than independent routing per segment: **none-or-all segment admission control, joint optimization**
  - ❑ E.g., LP relaxation (Raghavan et al.): how to **randomly round and decompose complex requests**?



Applications: Algorithms *with a twist!*

- Le
  -
  -
- SD
  - **Non-shortest** paths
  - Enables **complex network services:** steer traffic through middleboxes i.e. **waypoints** (firewall, proxy etc.): paths may contain **loops**!
  - More than independen segment: **none-or-all se** **admission control, join**
  - E.g., LP relaxation (Raghavan et al.): how to **randomly roun** **decompose complex re**

Optionally *NFV twist*: where to place NFV (or hybrid SDN)? Facility location / capacitated dominating set, *but*: not distance to but distance *via function(s)* matters!



[Online Admission Control and Embedding of Service Chains](#)
Tamás Lukovszki and Stefan Schmid.
22nd International Colloquium on Structural Information and Communication Complexity (**SIROCCO**), Montserrat, Spain, July 2015.

[Service Chain and Virtual Network Embeddings: Approximations using Randomized Rounding](#)
Matthias Rost and Stefan Schmid.
ArXiv Technical Report, April 2016.

Applications: Algorithms *with a twist!*

❑ Le

Optionally *NFV twist*: where to place NFV (or hybrid SDN)? Facility location / capacitated dominating set, *but*: not distance to but distance *via function(s)* matters!

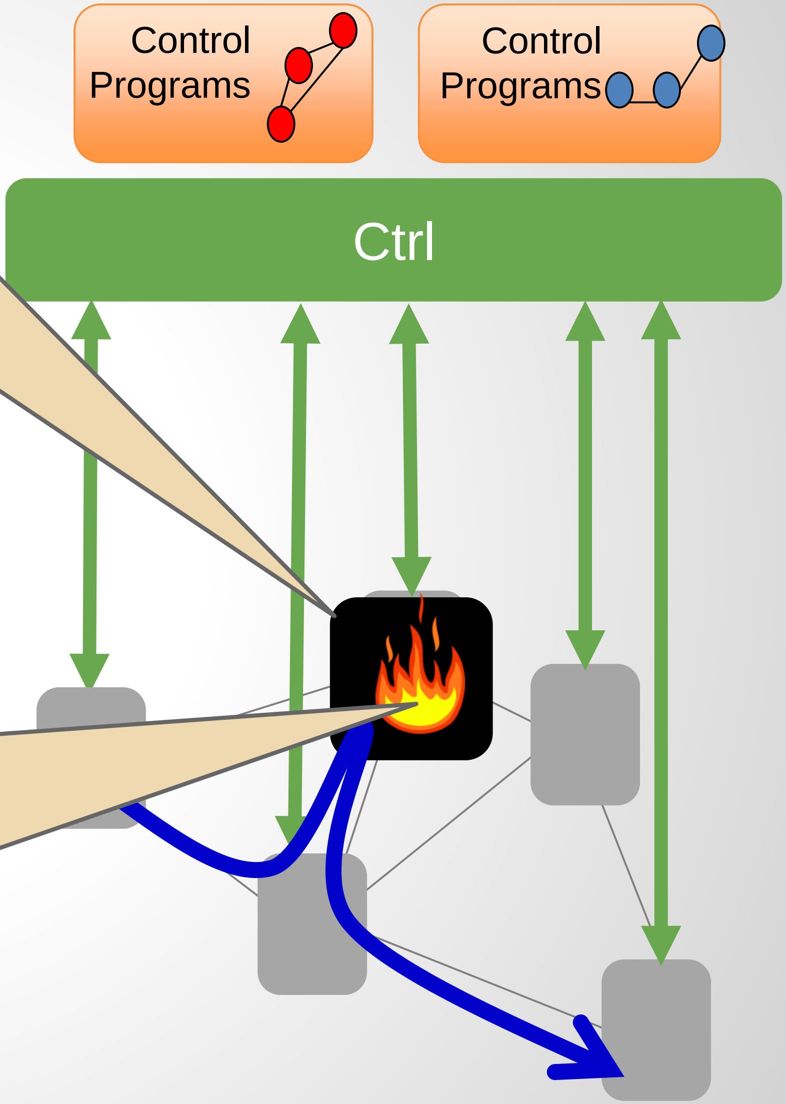
❑ SD

❑ **Non-shortest** paths

❑ Enables **complex network services:** steer traffic through middleboxes i.e.

Migration upon each new request undesirable: want **incremental deployment**! Related to submodular capacitated set cover and scheduling (Fleischer, Khuller), *but* **end-to-end**.

decompose complex requests?



Applications: Algorithms *with a twist!*

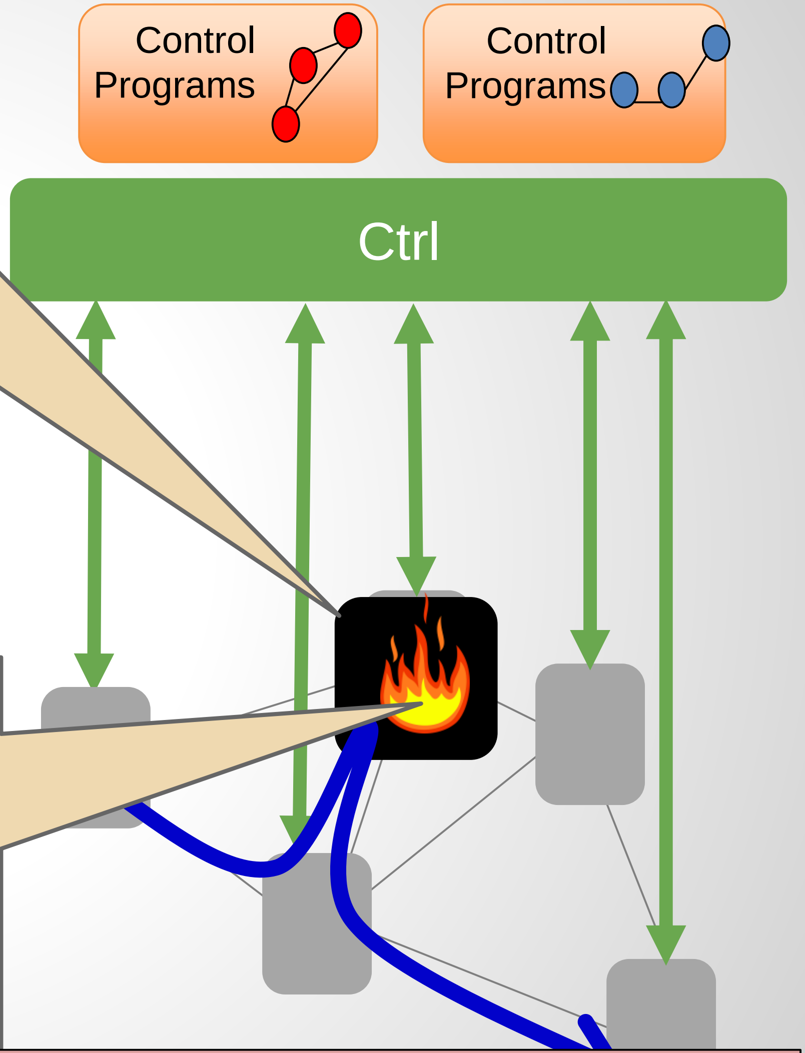
- Le
  -
  -

Optionally *NFV twist*: where to place NFV (or hybrid SDN)? Facility location / capacitated dominating set, *but*: not distance to but distance *via function(s)* matters!

- SD
  - **Non-shortest** paths
  - Enables **complex network services:** steer traffic through middleboxes i.e.

Migration upon each new request undesirable: want **incremental deployment**! Related to submodular capacitated set cover and scheduling (Fleischer, Khuller), *but* **end-to-end**

decompose complex re

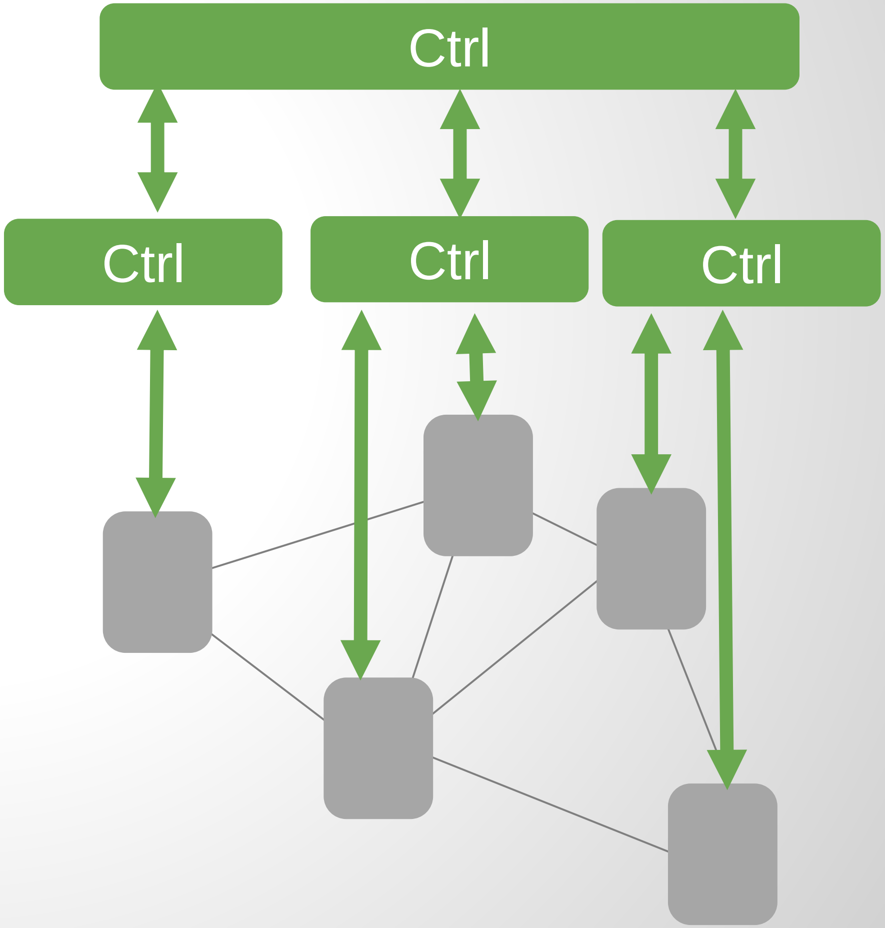


It's a Match! Near-Optimal and Incremental Middlebox Deployment
Tamás Lukovszki, Matthias Rost, and Stefan Schmid.
ACM SIGCOMM Computer Communication Review (**CCR**), January 2016.

Control Plane: Algorithms *with a twist!*

- ❑ Reduce **latency and overhead**: What can be computed locally?
  - ❑ Routing vs heavy-hitter detection?
  - ❑ LOCAL model! Insights apply: **verification vs optimization**

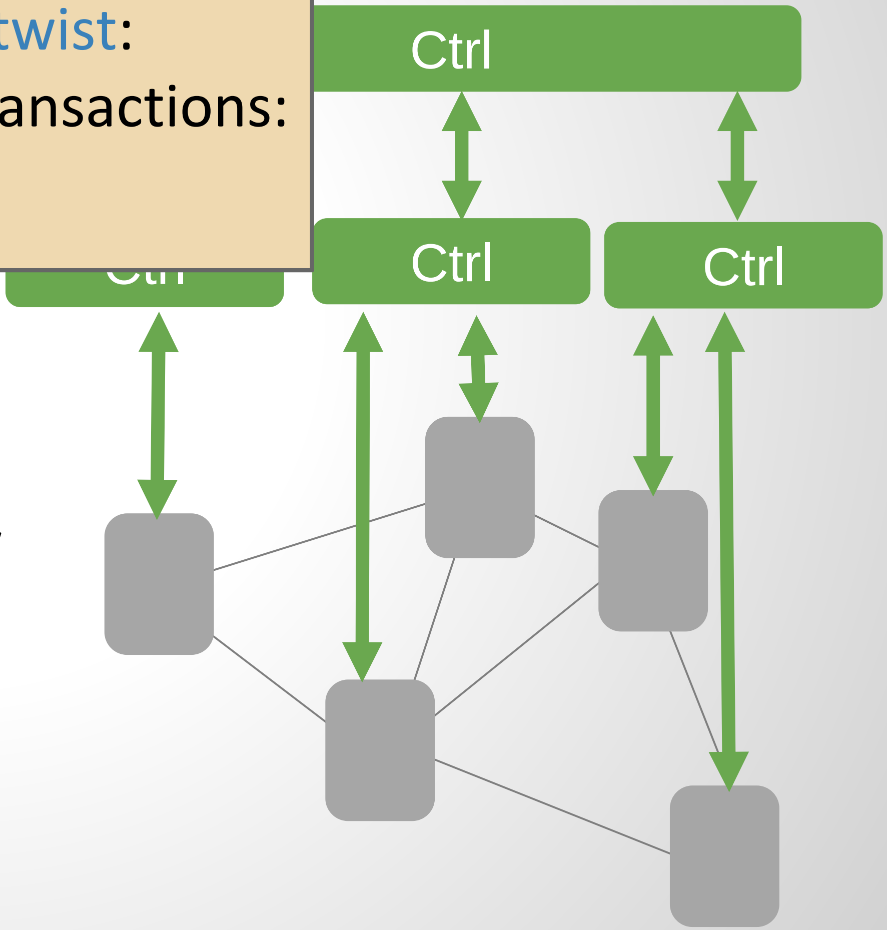
- ❑ *SDN twist: pre-processing!*
  - ❑ Hard in LOCAL: **symmetry breaking**! But unlike **ad-hoc networks**: no need to discover network from scratch
  - ❑ Topology events **less frequent** than flow related events
  - ❑ If **links fail**: **subgraph**! Find recomputed structures that are still useful in subgraph (e.g., **proof labelings**)
  - ❑ Precomputation known to help for relevant problems: **load-balancing / matching**



Control Plane: Algorithms *with a twist!*

How to make control plane robust? **Software transactional memory** problem:
network configuration = shared memory,
updates = **transactions**, but with a twist:
flows are uncontrolled, real-time transactions:
do not abort! (And not only read!)

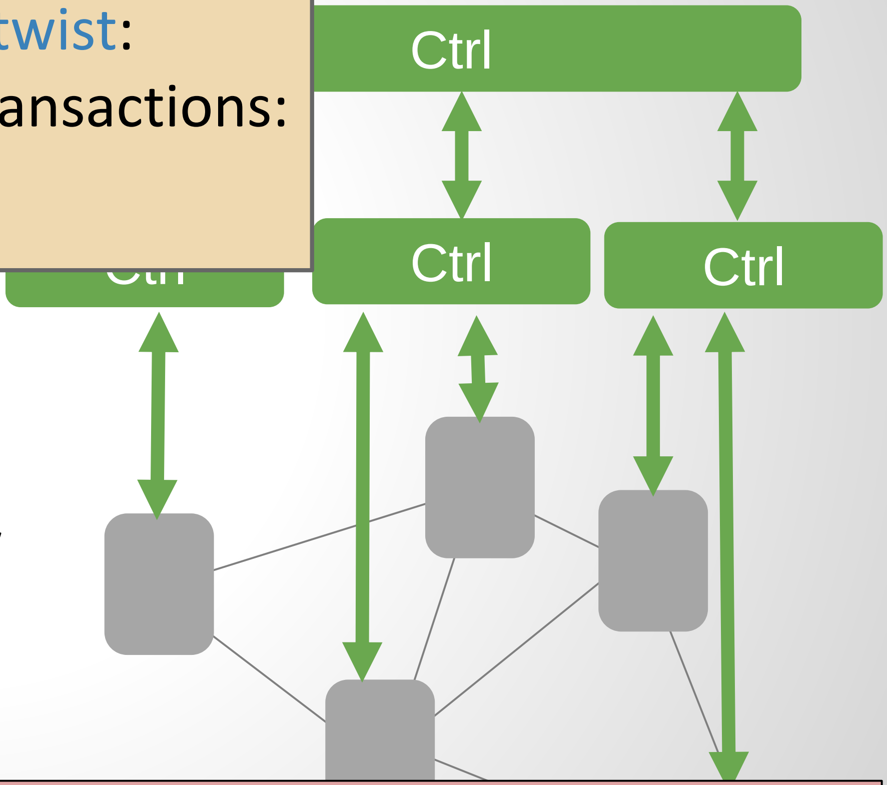
- *SDN twist: pre-processing!*
  - Hard in LOCAL: **symmetry breaking**! But unlike **ad-hoc networks**: no need to discover network from scratch
  - Topology events **less frequent** than flow related events
  - If **links fail**: **subgraph**! Find recomputed structures that are still useful in subgraph (e.g., **proof labelings**)
  - Precomputation known to help for relevant problems: **load-balancing / matching**



Control Plane: Algorithms *with a twist!*

How to make control plane robust? **Software transactional memory** problem:
network configuration = shared memory,
updates = **transactions**, but with a twist:
flows are uncontrolled, real-time transactions:
do not abort! (And not only read!)

- *SDN twist: pre-processing!*
  - Hard in LOCAL: **symmetry breaking**! But unlike **ad-hoc networks**: no need to discover network from scratch
  - Topology events **less frequent** than flow related events
  - If **links fail**: **subgraph**! Find recomputed structures that are still useful in subgraph (e.g., **proof labe**
  - Precomputation known t
    relevant problems: **load-l**
    matching

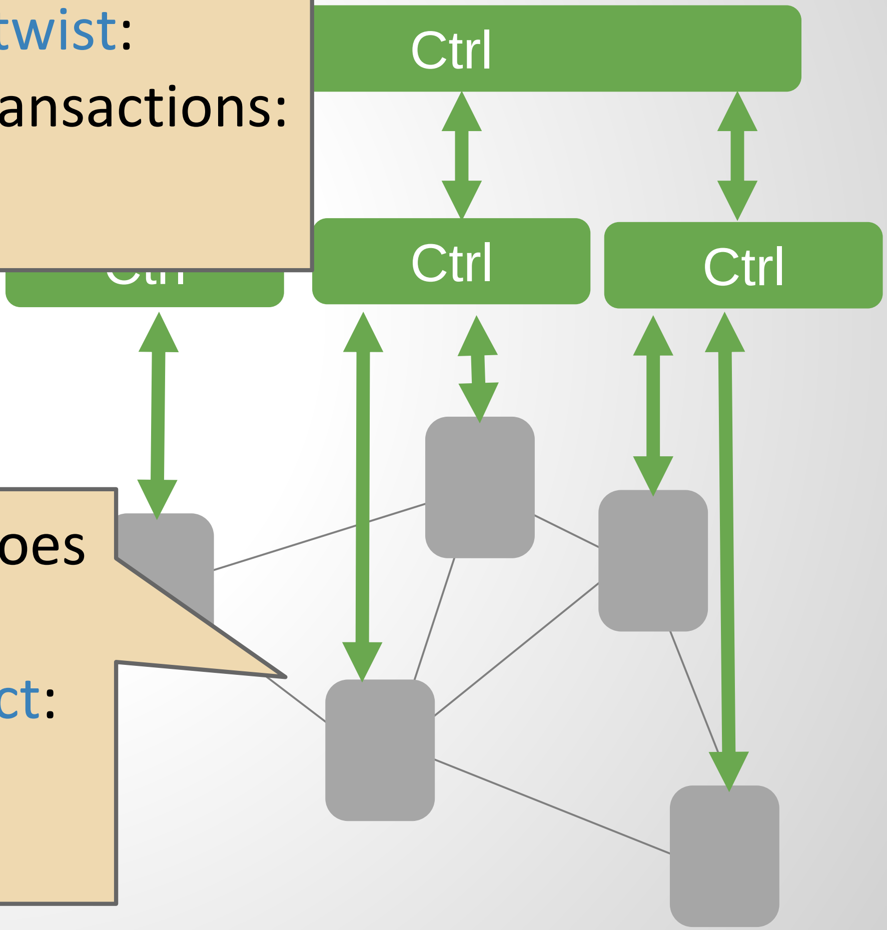


[A Distributed and Robust SDN Control Plane for Transactional Network Updates](#)
Marco Canini, Petr Kuznetsov, Dan Levin, and Stefan Schmid.
34th IEEE Conference on Computer Communications (**INFOCOM**), Hong Kong, April 2015.

Control Plane: Algorithms *with a twist!*

How to make control plane robust? **Software transactional memory** problem: **network configuration = shared memory,** updates = **transactions**, but with a twist: flows are uncontrolled, real-time transactions: do not abort! (And not only read!)

- *SDN twist: pre-processing!*
  - Hard in LOCAL: **symmetry breaking**! But unlike **ad-hoc networks**: no need to discover network from scratch



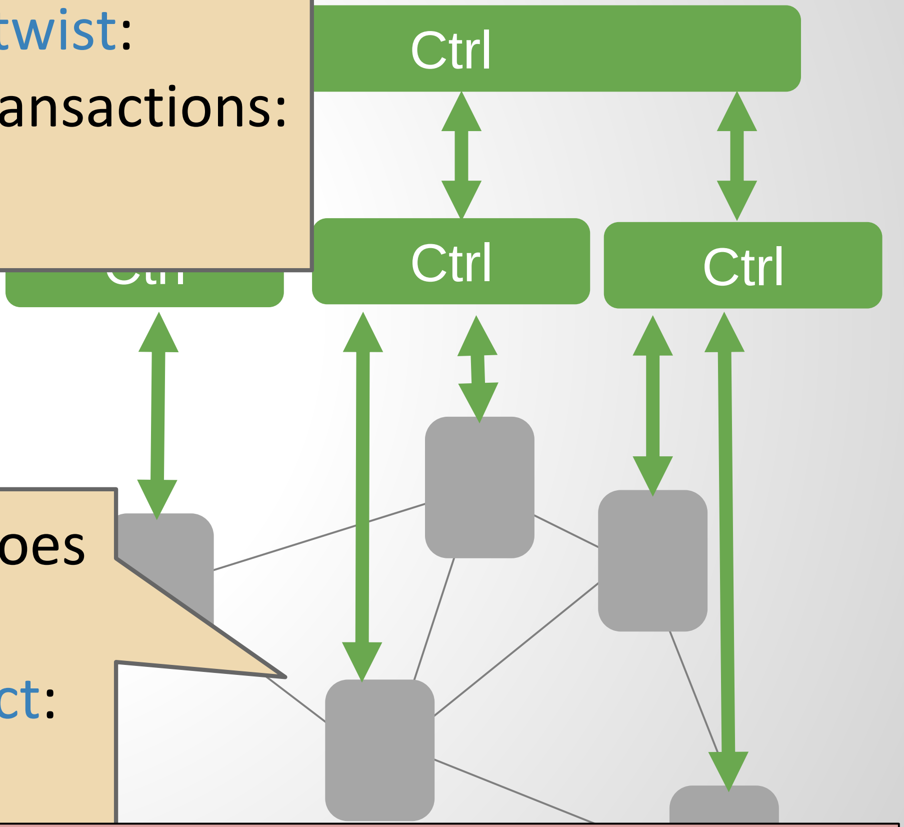
Careful: independent flow spaces does not imply that controllers can **concurrently** update without conflict: e.g., due to **shared embedding**! Atomic read-modify-write?

relevant problems: **load-balancing / matching**

Control Plane: Algorithms *with a twist!*

How to make control plane robust? **Software transactional memory** problem: **network configuration = shared memory,** updates = **transactions**, but with a twist: flows are uncontrolled, real-time transactions: do not abort! (And not only read!)

- *SDN twist: pre-processing!*
  - Hard in LOCAL: **symmetry breaking**! But unlike **ad-hoc networks**: no need to discover network from scratch



Careful: independent flow spaces does not imply that controllers can **concurrently** update without conflict: e.g., due to **shared embedding**! Atomic read-modify-writ

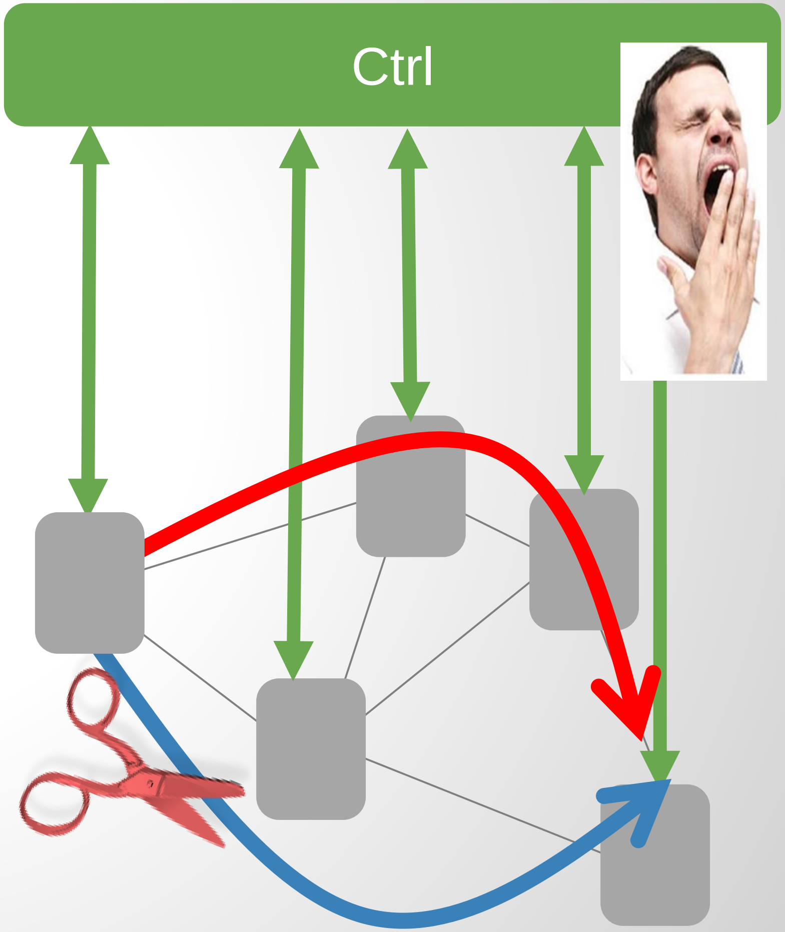
In-Band Synchronization for Distributed SDN Control Planes
Liron Schiff, Petr Kuznetsov, and Stefan Schmid.
ACM SIGCOMM Computer Communication Review (**CCR**), January 2016.

relevant problems: **load-b**
matching

Data Plane: Algorithms *with a twist!*

- ❑ Even in SDN: Keep some functionality in the data plane!
  - ❑ E.g., for **performance**: OpenFlow local fast failover: 1st line of defense

- ❑ SDN twist: data plane algorithms operate under **simple conditions**
  - ❑ Failover tables are **statically** (proactively) **preconfigured**, **w/o multiple faiures knowledge**
  - ❑ At runtime: **local view only** and **header space is scarce resource**
  - ❑ W/ tagging: **graph exploration**
  - ❑ W/o tagging: **combinatorial problem**
  - ❑ Later: **consolidate this with controller**!

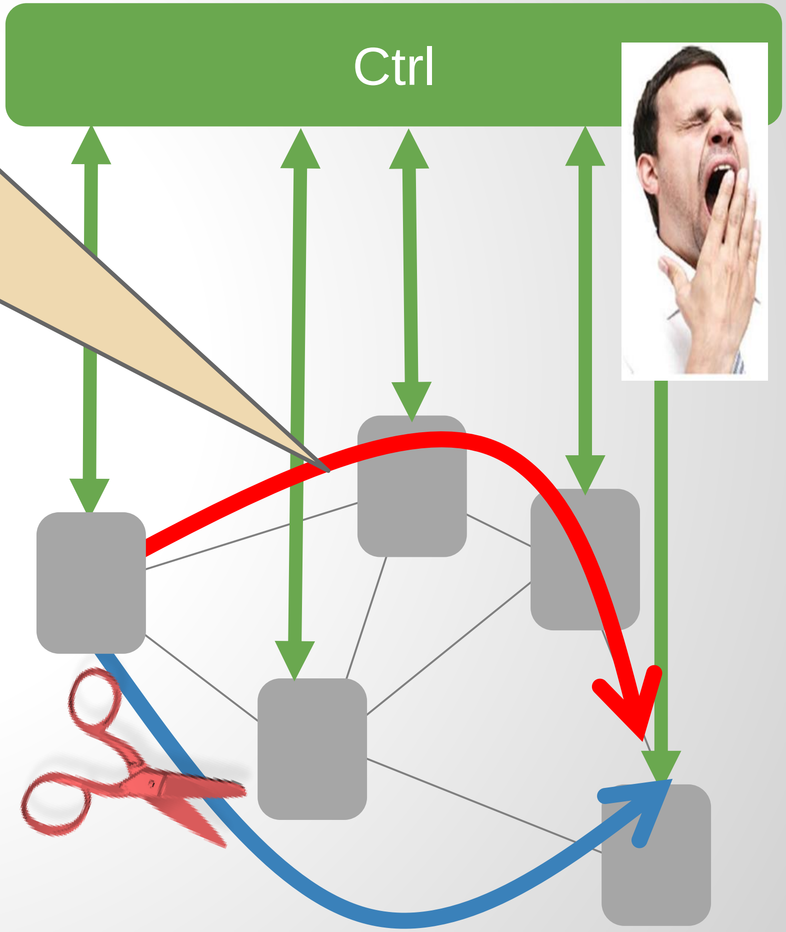


Data Plane: Algorithms *with a twist!*

With **infinite header space** ideal robustness possible. But what about bounded header space? And resulting route lengths?
Without good algorithms, routing may disconnect way before physical network does!

operate under **simple conditions**

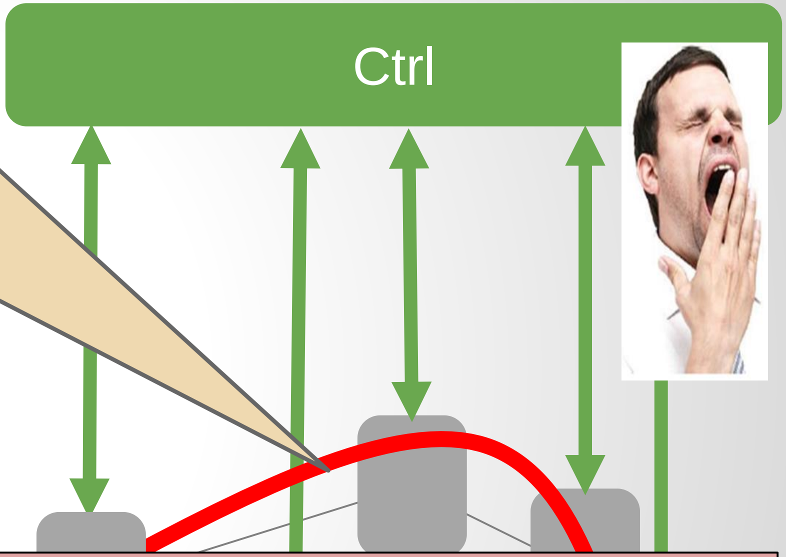
- ❑ Failover tables are **statically** (proactively) **preconfigured**, **w/o multiple faiures knowledge**
- ❑ At runtime: **local view only** and **header space is scarce resource**
- ❑ W/ tagging: **graph exploration**
- ❑ W/o tagging: **combinatorial problem**
- ❑ Later: **consolidate this with controller**!



Data Plane: Algorithms *with a twist!*

With **infinite header space** ideal robustness possible. But what about bounded header space? And resulting route lengths?
Without good algorithms, routing may disconnect way before physical network does!

- operate under **simple conditions**
  - Failover tables are **statically** (proactively) **preconfigured**, **w/o multiple faiures knowle**
  - At runtime: **local view** c **space is scarce resourc**
  - W/ tagging: **graph exploration**
  - W/o tagging: **combinat**
  - Later: **consolidate this**



[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.

[Provable Data Plane Connectivity with Local Fast Failover: Introducing OpenFlow Graph Algorithms]
Michael Borokhovich, Liron Schiff, and Stefan Schmid.
ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (**HotSDN**), Chicago, Illinois, USA, August 2014.

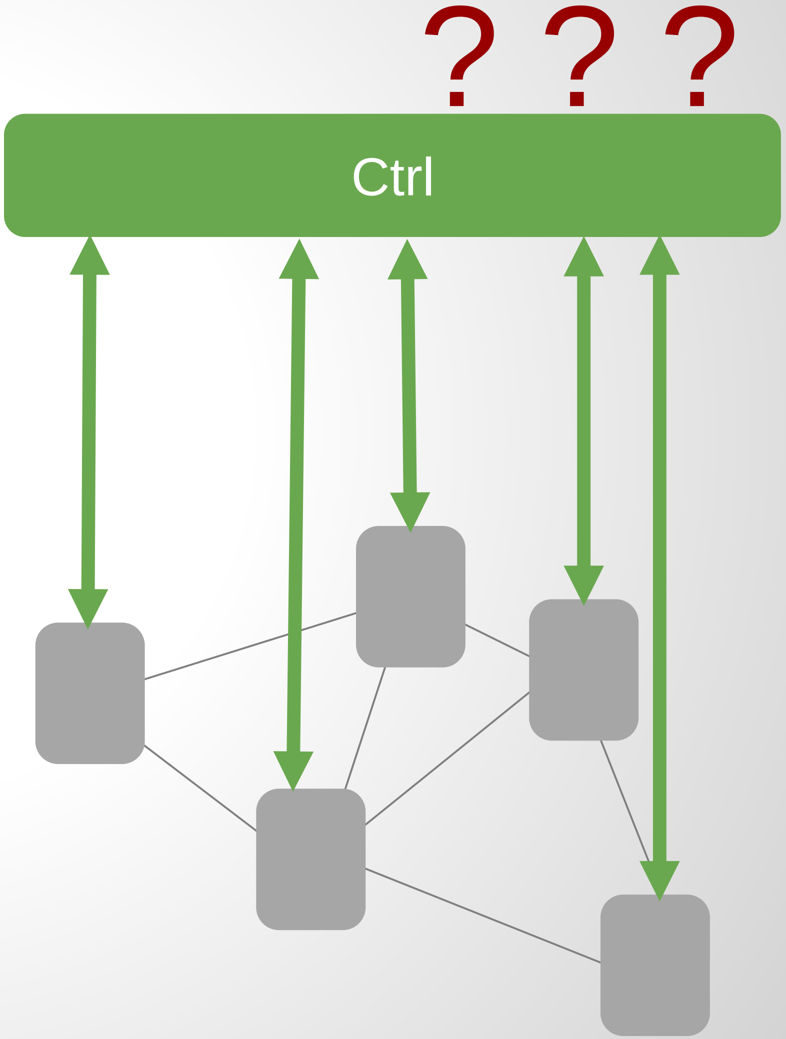
Decoupling: Algorithms *with a twist!*

❑ Decoupling already challenging for a single switch!

❑ Network *Hello World* application: MAC learning

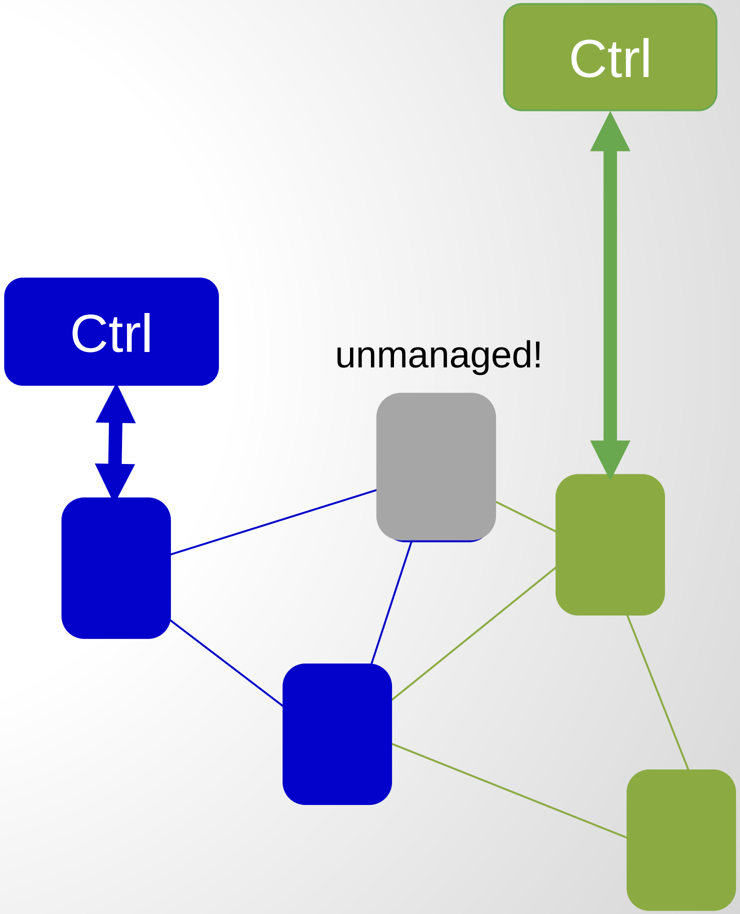
❑ MAC learning has *SDN twist*: MAC learning SDN controller is decoupled: may miss response and keep flooding!

❑ Need to **configure rules** s.t. controller stays informed when necessary!



Decoupling: Algorithms *with a twist!*

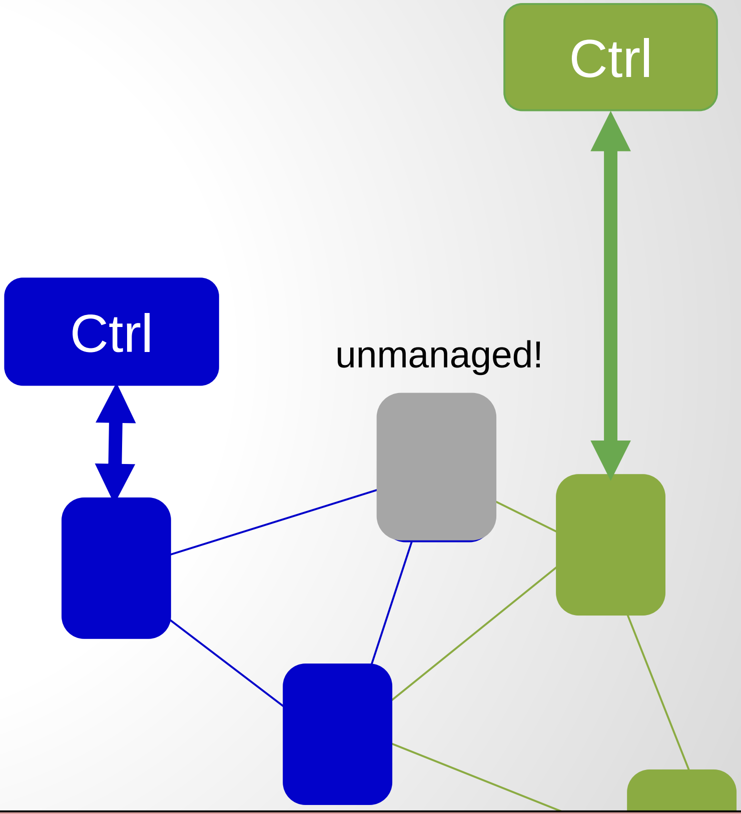
- In-band control: cheap but algorithmically challenging!
  - Distributed coordination algorithms to manage switches?
  - Powerful fault-tolerance concept: **self-stabilization**
- *SDN twist*: switches **are simple**!
  - Cannot actively participate in **arbitrary self-stab spanning tree protocols**
  - Controller needs to install tree rules



Decoupling: Algorithms *with a twist!*

- ❑ In-band control: cheap but algorithmically challenging!
  - ❑ Distributed coordination algorithms to manage switches?
  - ❑ Powerful fault-tolerance concept: **self-stabilization**

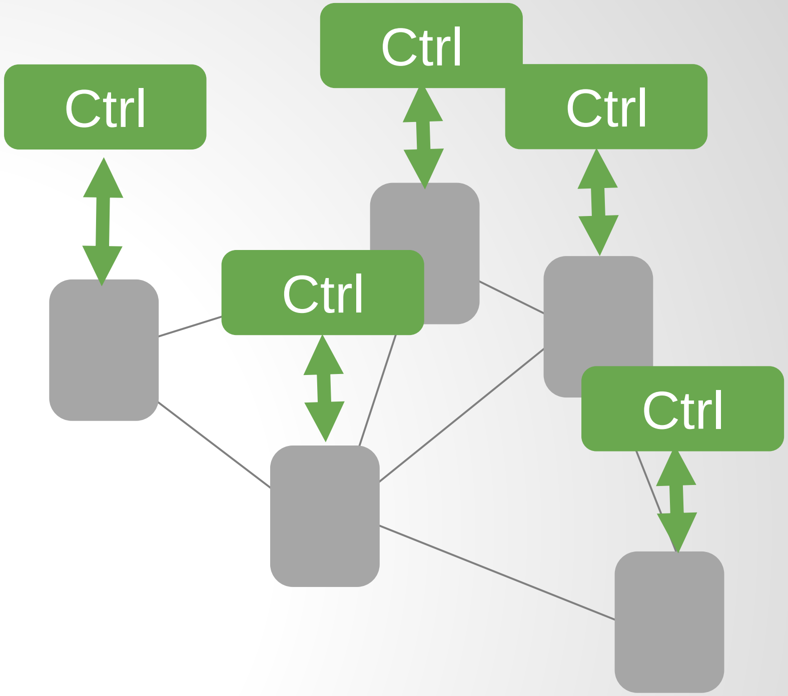
- ❑ *SDN twist*: switches **are simple**!
  - ❑ Cannot actively participate in **arbitrary self-stab spanning tree protocols**
  - ❑ Controller needs to install tree rules



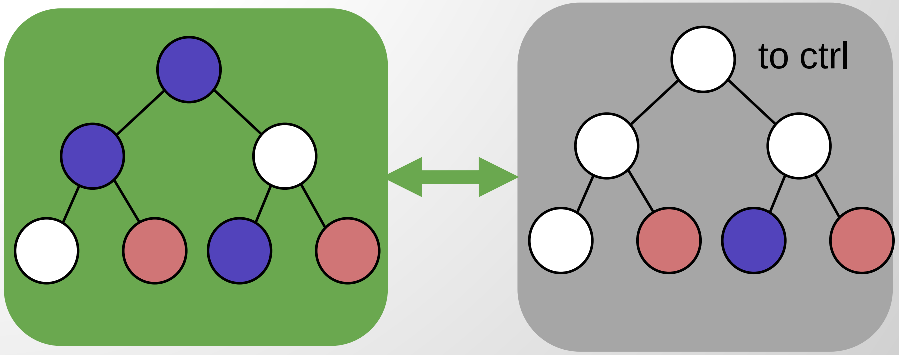
Ground Control to Major Faults: Towards a Fault Tolerant and Adaptive SDN Control Network
Liron Schiff, Stefan Schmid, and Marco Canini.
IEEE/IFIP DSN Workshop on Dependability Issues on SDN and NFV (**DISN**), Toulouse, France, June 2016.

Decoupling: Algorithms *with a twist!*

- Researchers proposed to *exploit SDN* rule definition flexiblities to solve growing FIB size problem
  - OpenFlow-based IP router: **caching and aggregation**
  - **Zipf law**: many infrequent prefixes at controller
  - Extremely distributed control ☺



- Online paging *with SDN twist*
  - Forwarding semantic: **largest common prefix forwarding**, i.e., dependencies: only offload **root-contiguous** set in trie
  - Can do **bypassing**



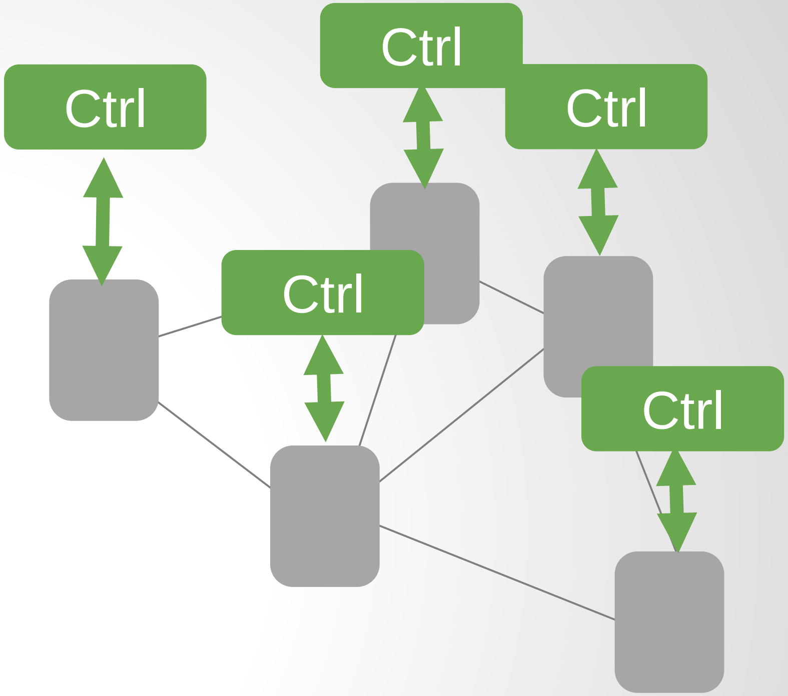
Decoupling: Algorithms *with a twist!*

❑ Researchers proposed to *exploit SDN* rule definition flexiblities to solve growing FIB size problem

  ❑ OpenFlow-based IP router: **caching and aggregation**

  ❑ **Zipf law**: many infrequent prefixes at controller

  ❑ Extremely distributed control ☺



❑ Online paging *with SDN twist*

  ❑ Forwarding semantic: **la**
  common prefix forwar
  dependencies: only offl
  contiguous set in trie

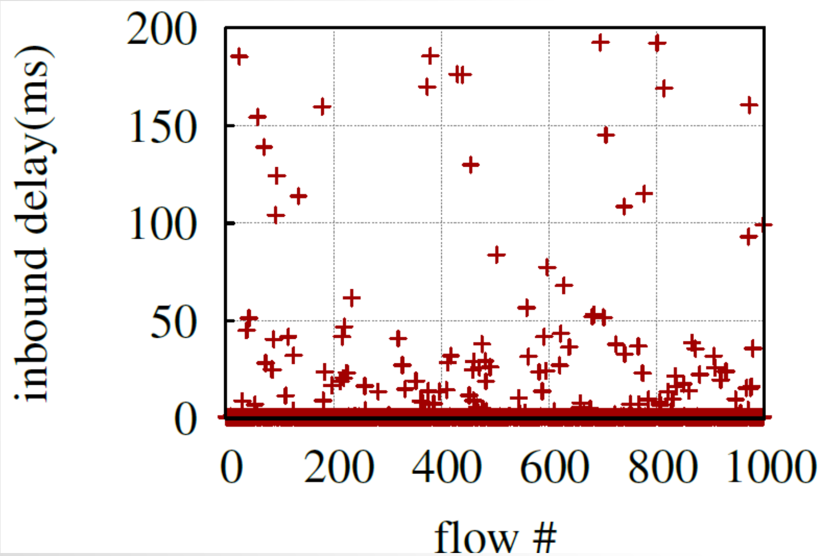
  ❑ Can do **bypassing**

Competitive FIB Aggregation without Update Churn
Marcin Bienkowski, Nadi Sarrar, Stefan Schmid, and Steve Uhlig.
34th International Conference on Distributed Computing Systems (**ICDCS**), Madrid, Spain, June 2014.

Online Tree Caching
Marcin Bienkowski, Jan Marcinkowski, Maciej Pacut, Stefan Schmid, and Aleksandra Spyra.
ArXiv Technical Report, February 2016.

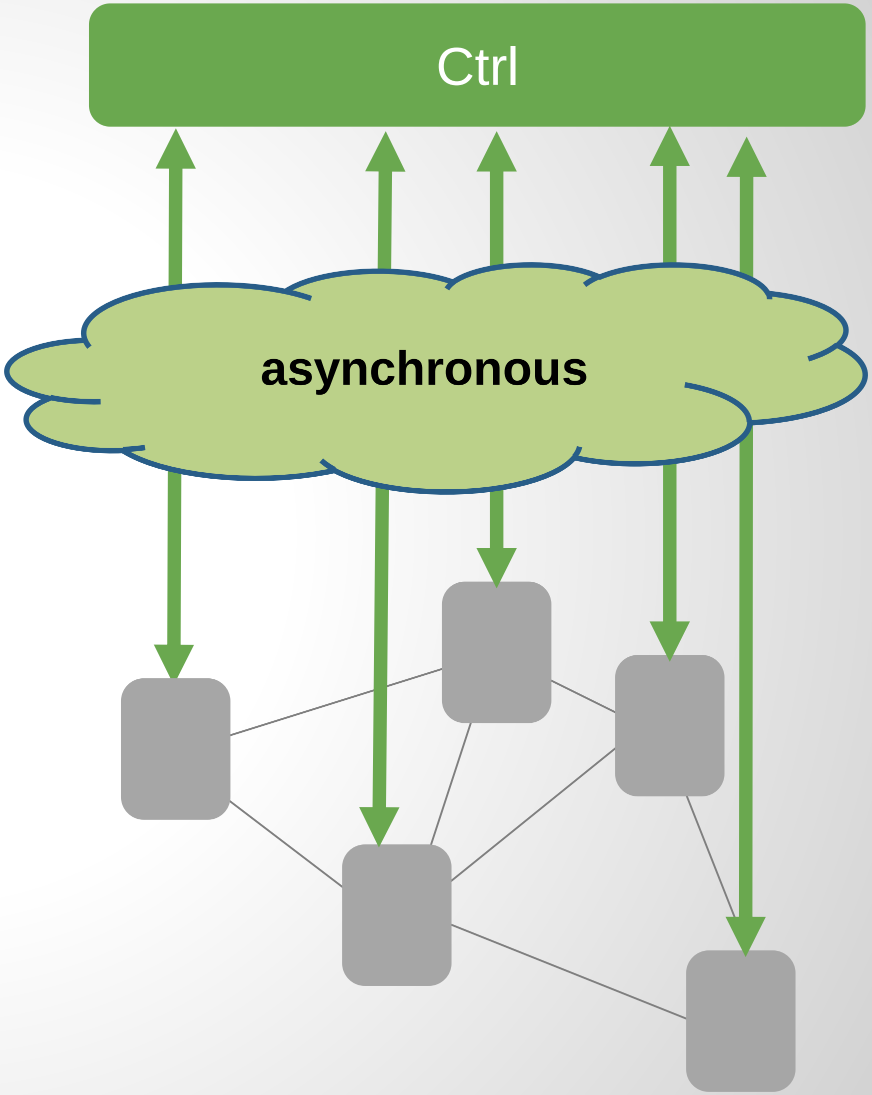
Interconnect: Algorithms *with a twist!*

- Another challenge: asynchronous communication channel



He et al., ACM SOSR 2015:

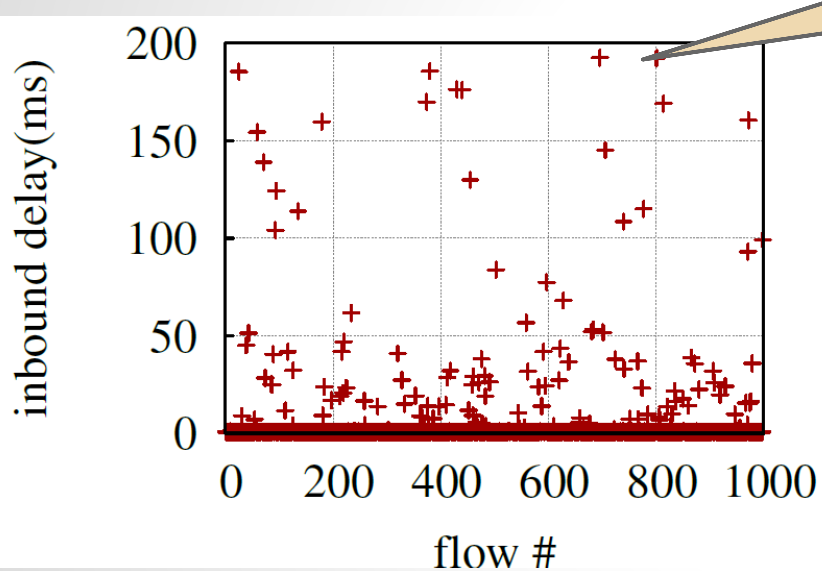
without network latency



Interconnect: Algorithms *with a twist!*

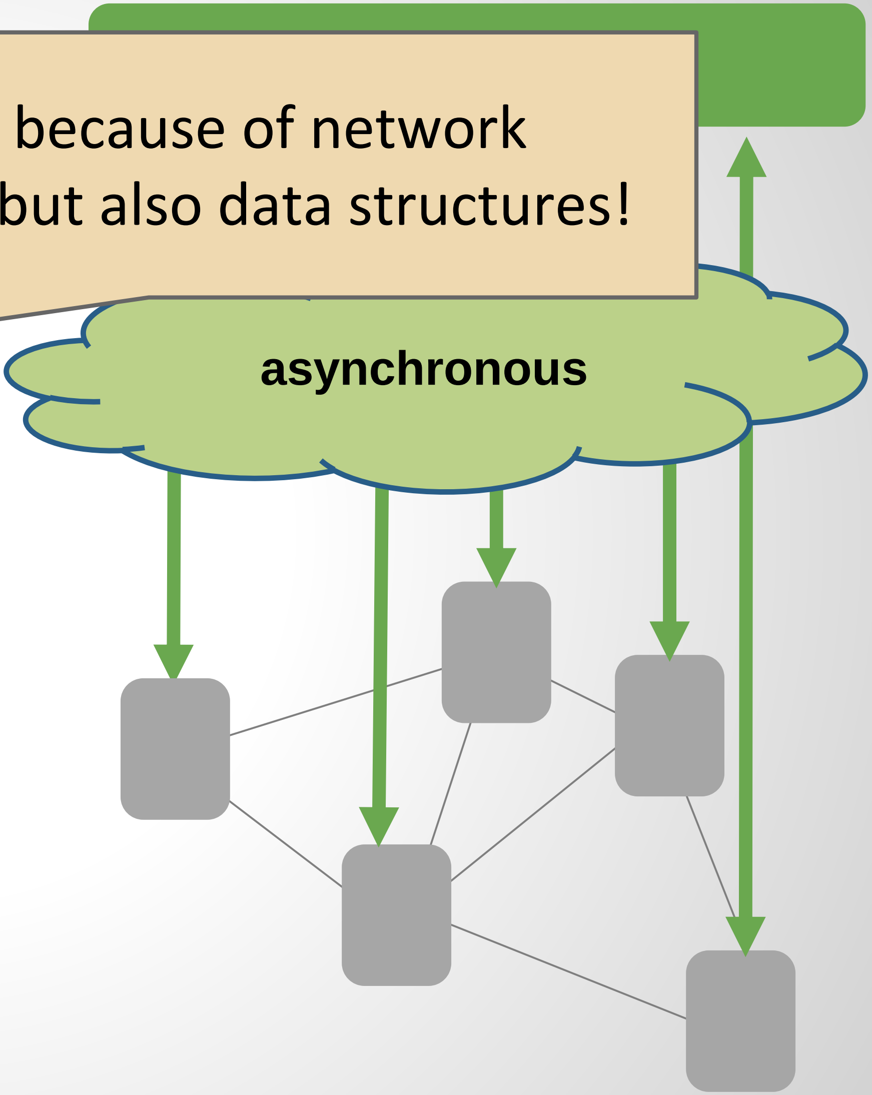
- ❑ Another challenge: async communication channel

Not only because of network latency, but also data structures!

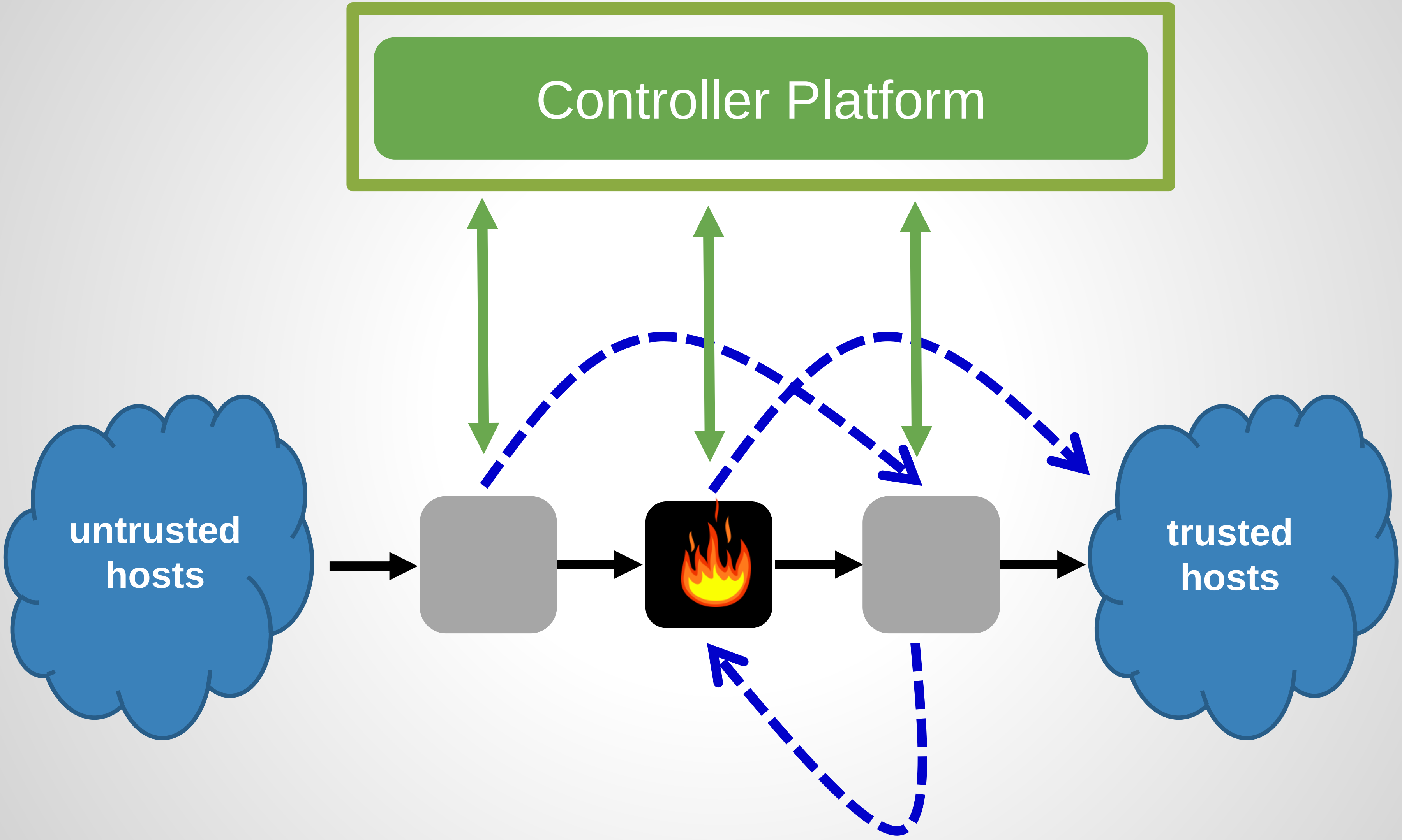


He et al., ACM SOSR 2015:

without network latency

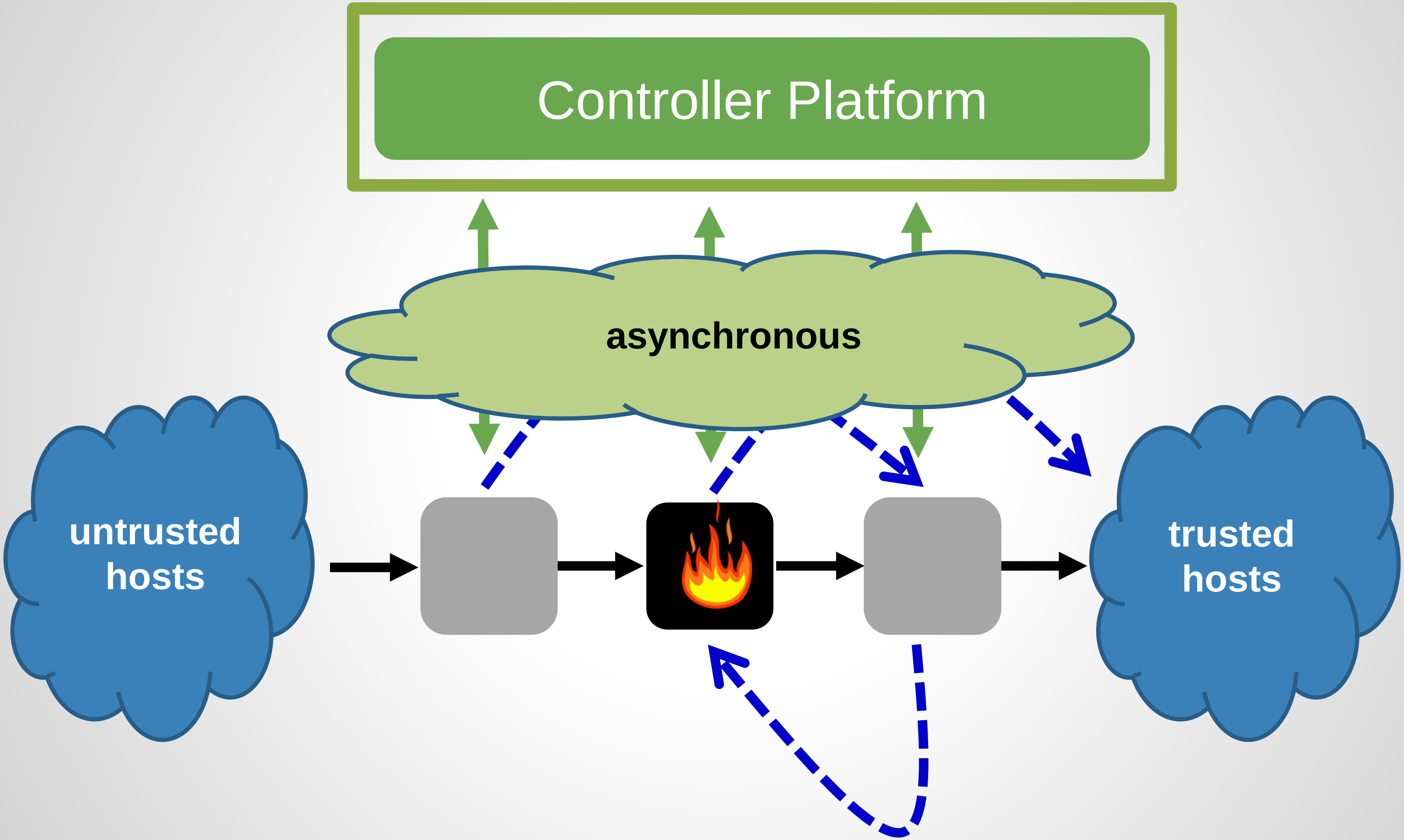


What can possibly go wrong?



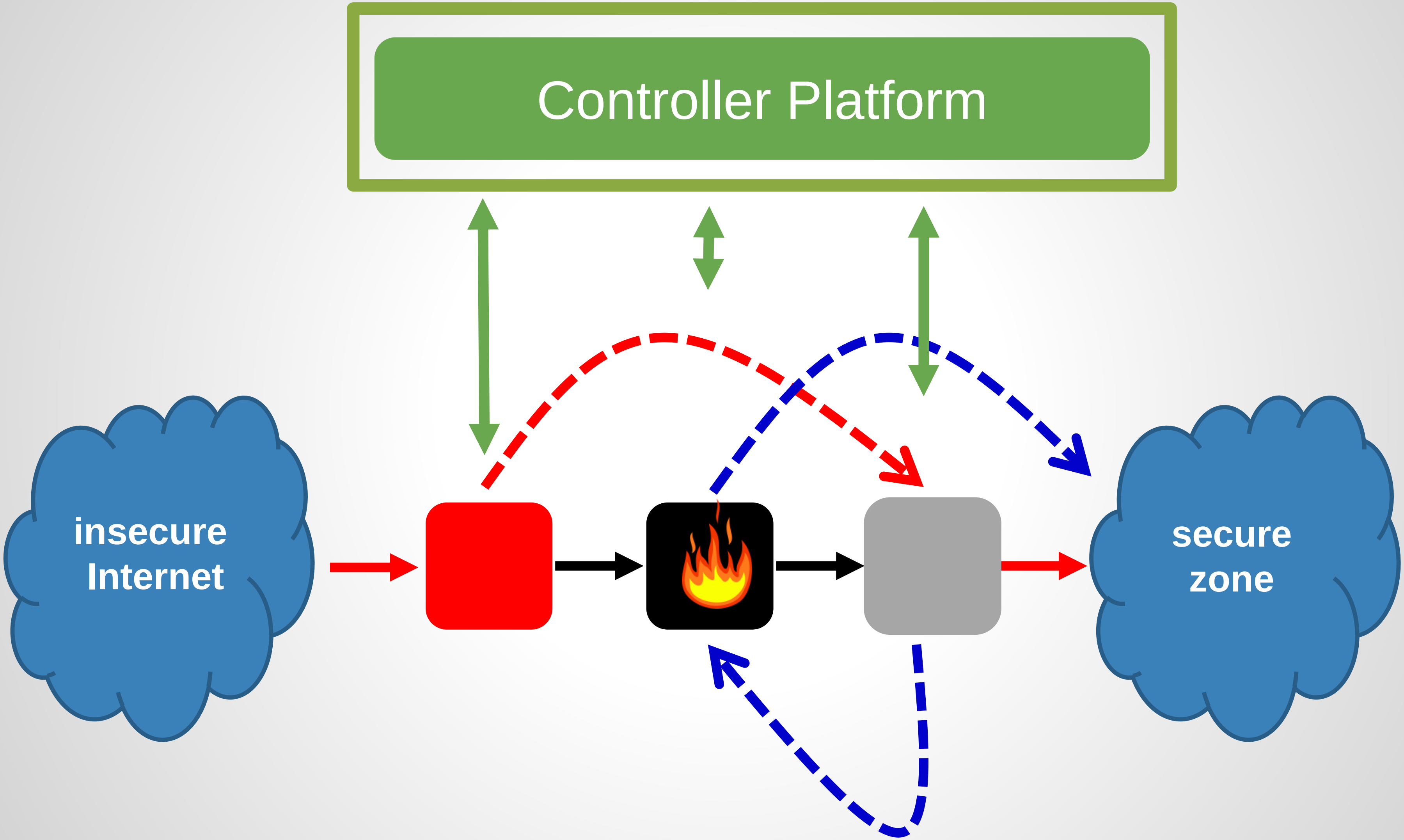
Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

What can possibly go wrong?

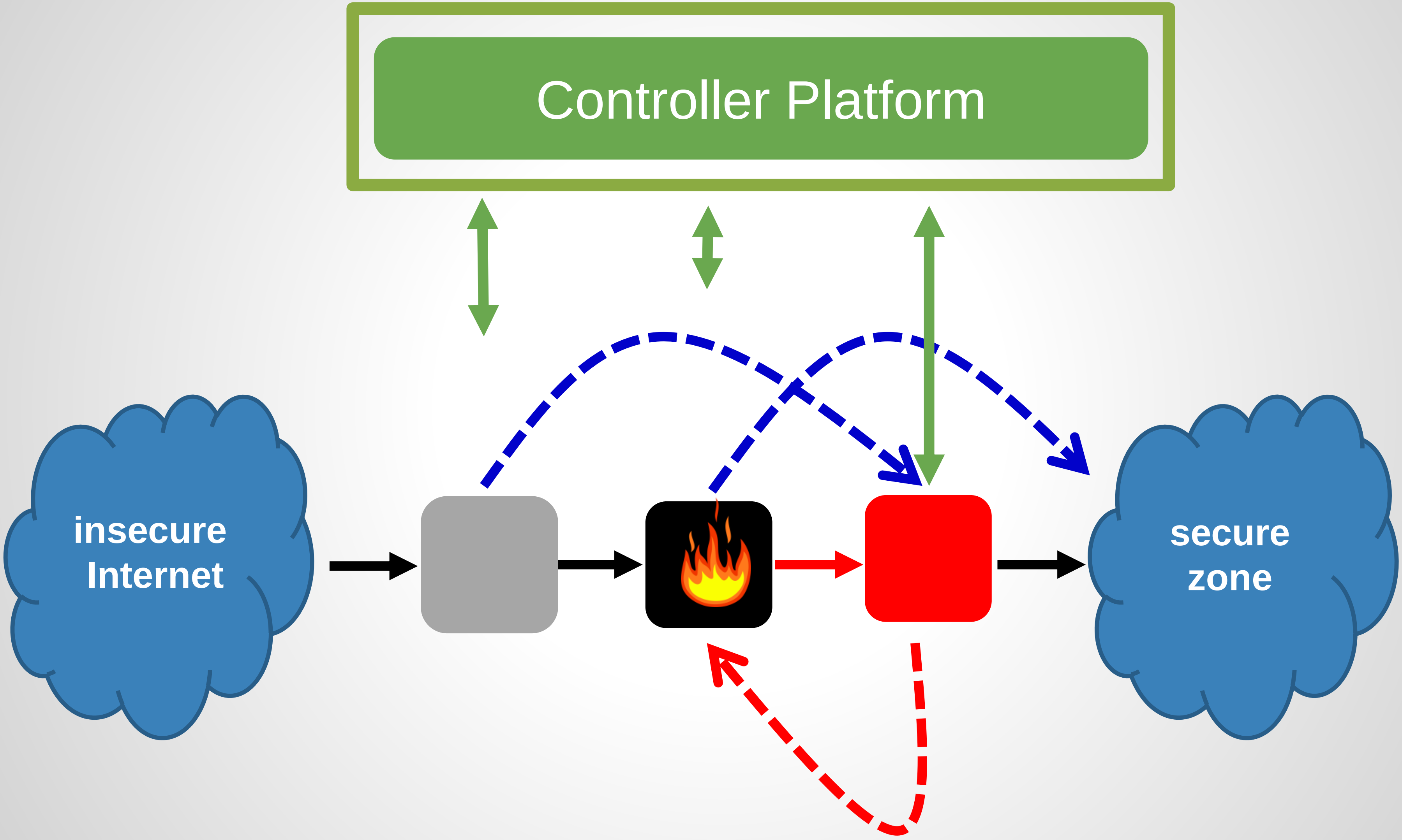


Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

Example 1: Bypassed Waypoint

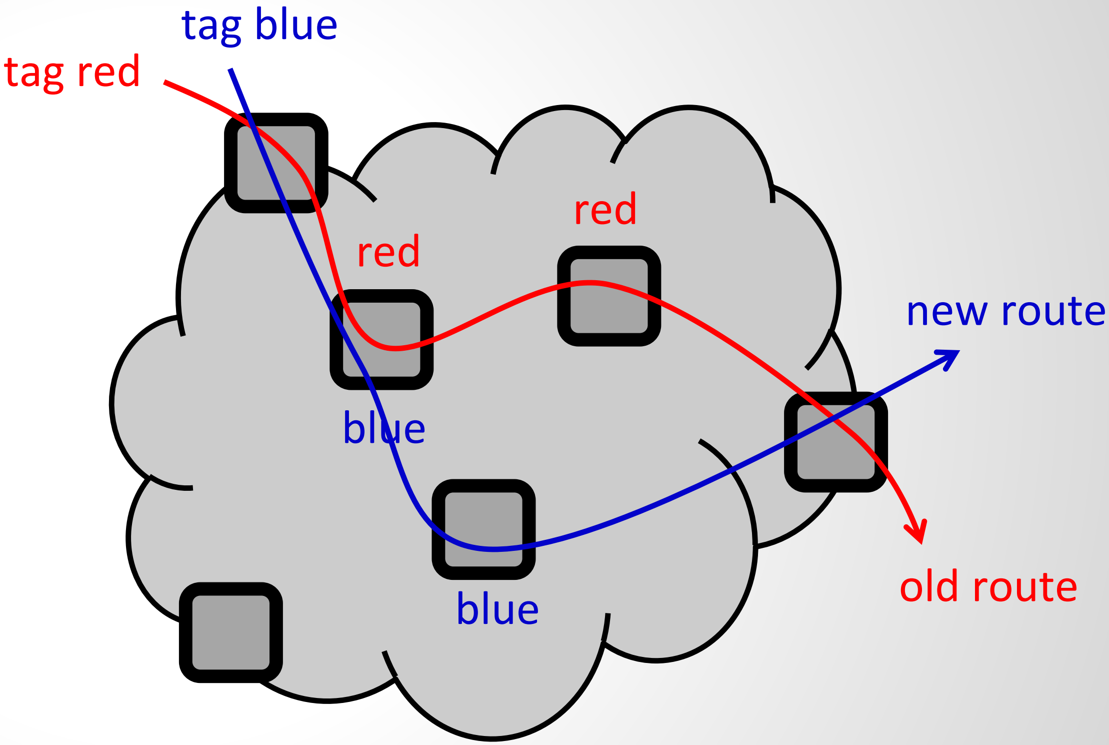


Example 2: *Transient* Loop



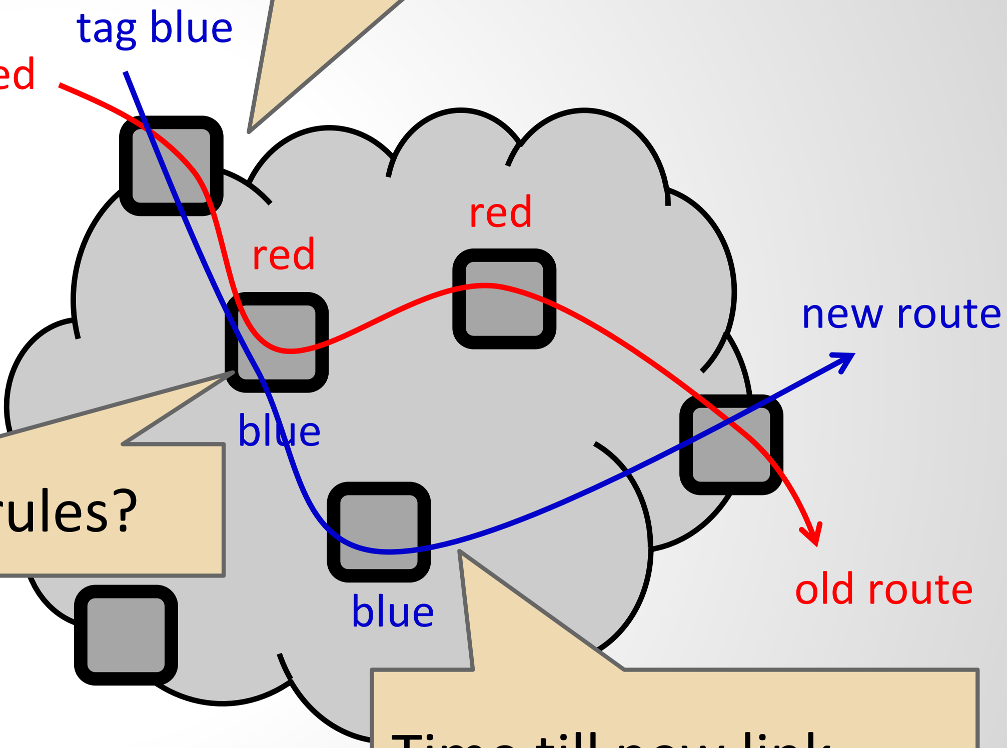
Tagging: A Universal Solution?

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



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?

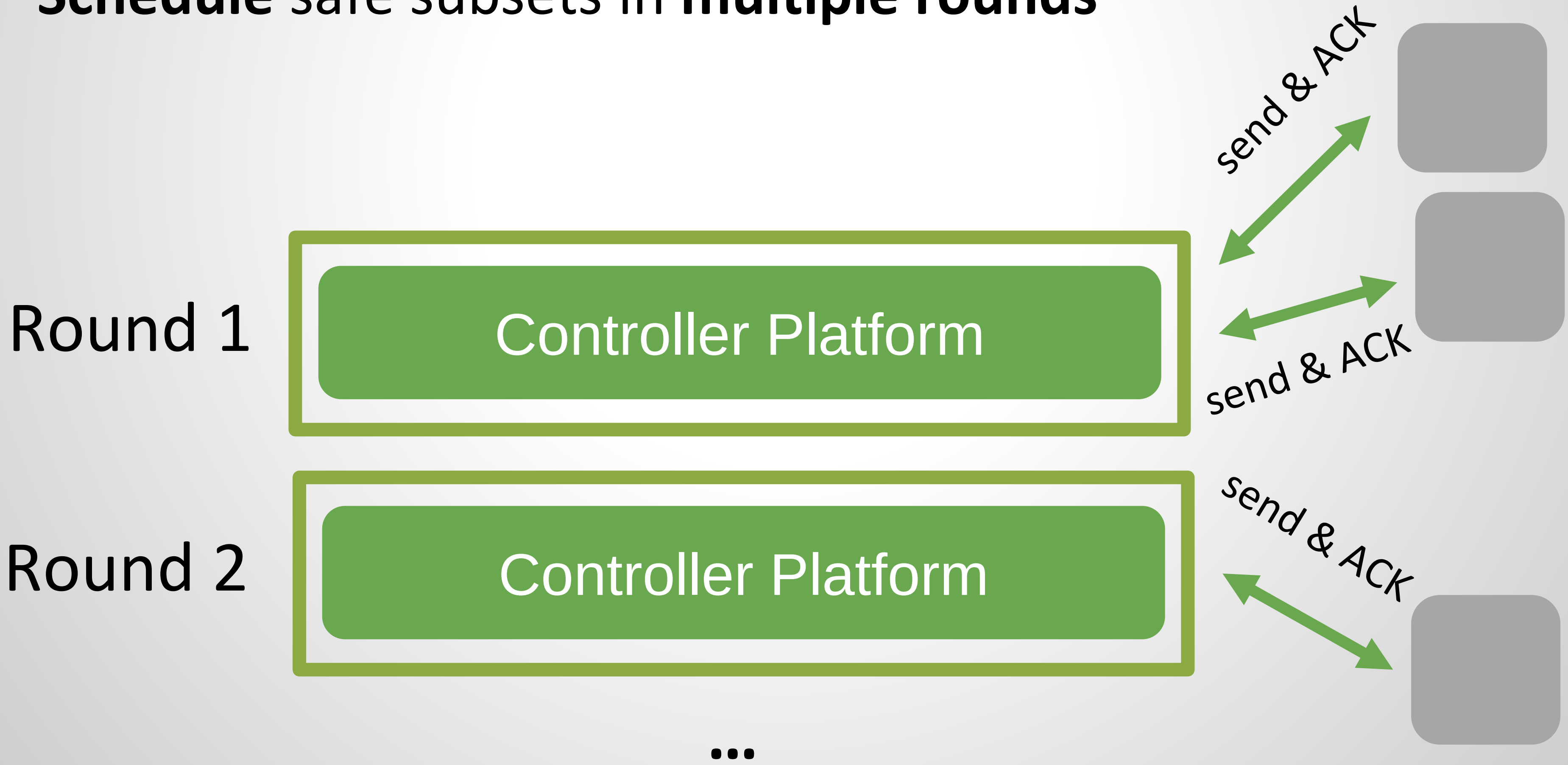
Cost of extra rules?

Time till new link becomes available?

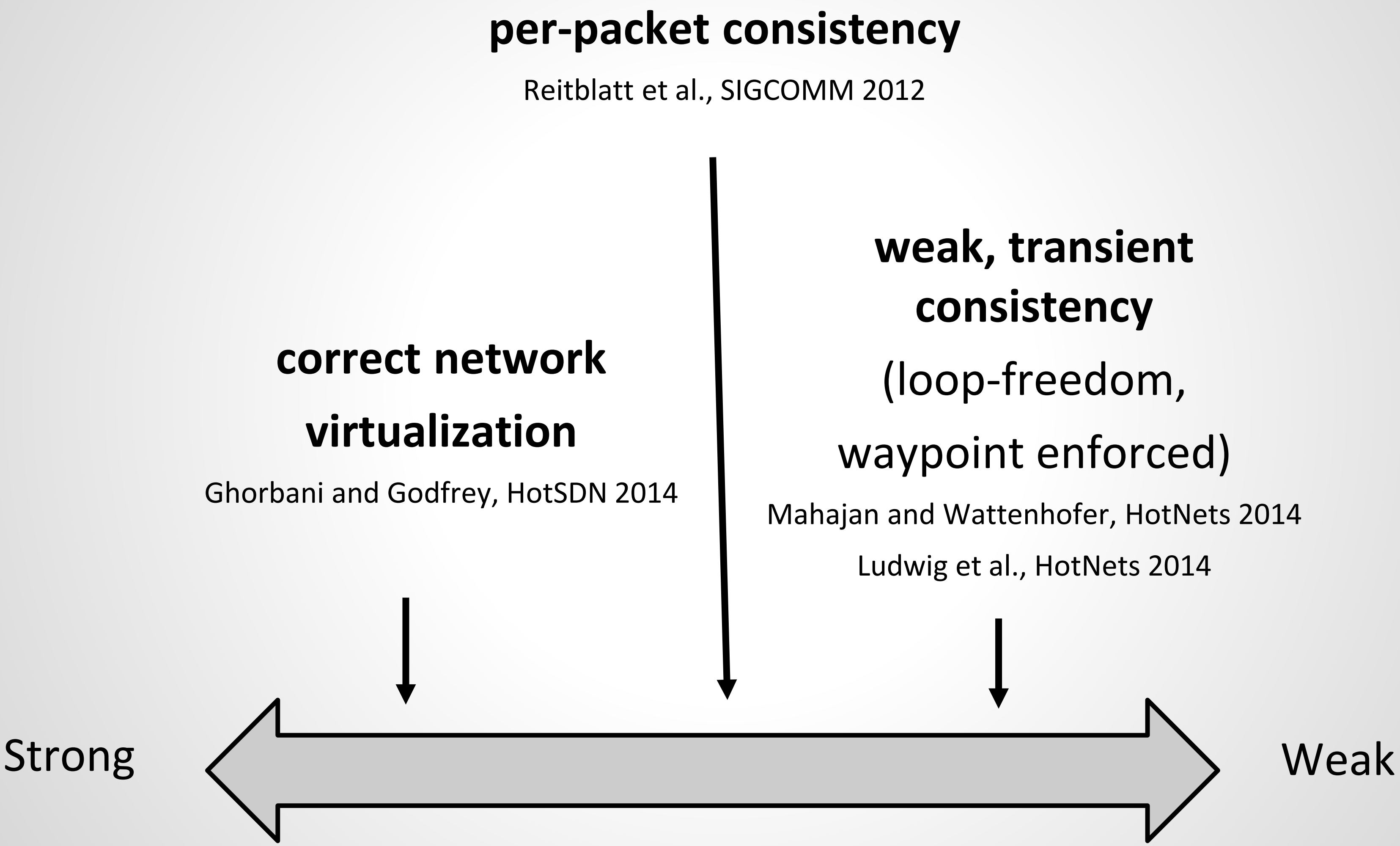
Alternative: Weaker Transient Consistency

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

Schedule safe subsets in **multiple rounds**



The Spectrum of Consistency



per-packet consistency

Reitblatt et al., SIGCOMM 2012

correct network virtualization

Ghorbani and Godfrey, HotSDN 2014

weak, transient consistency

(loop-freedom, waypoint enforced)

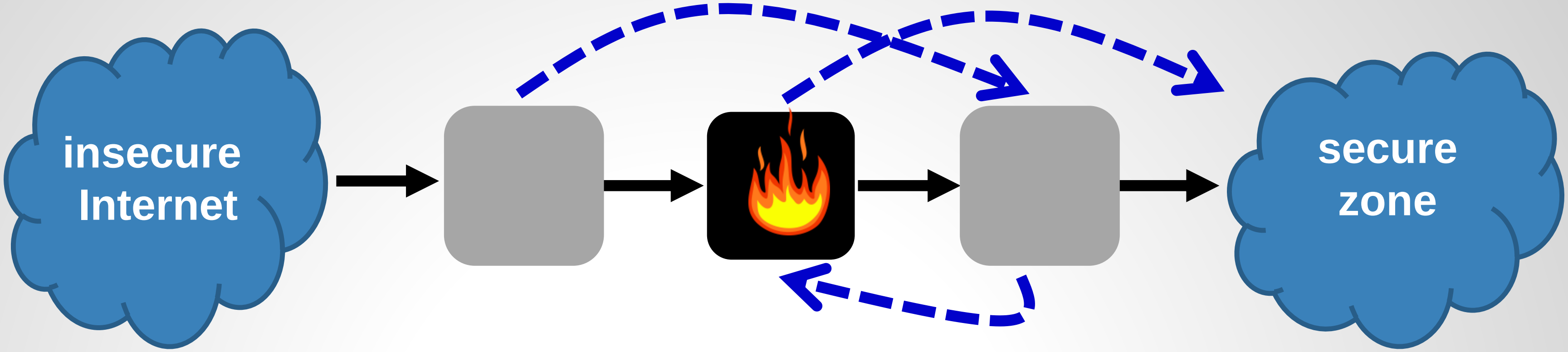
Mahajan and Wattenhofer, HotNets 2014

Ludwig et al., HotNets 2014

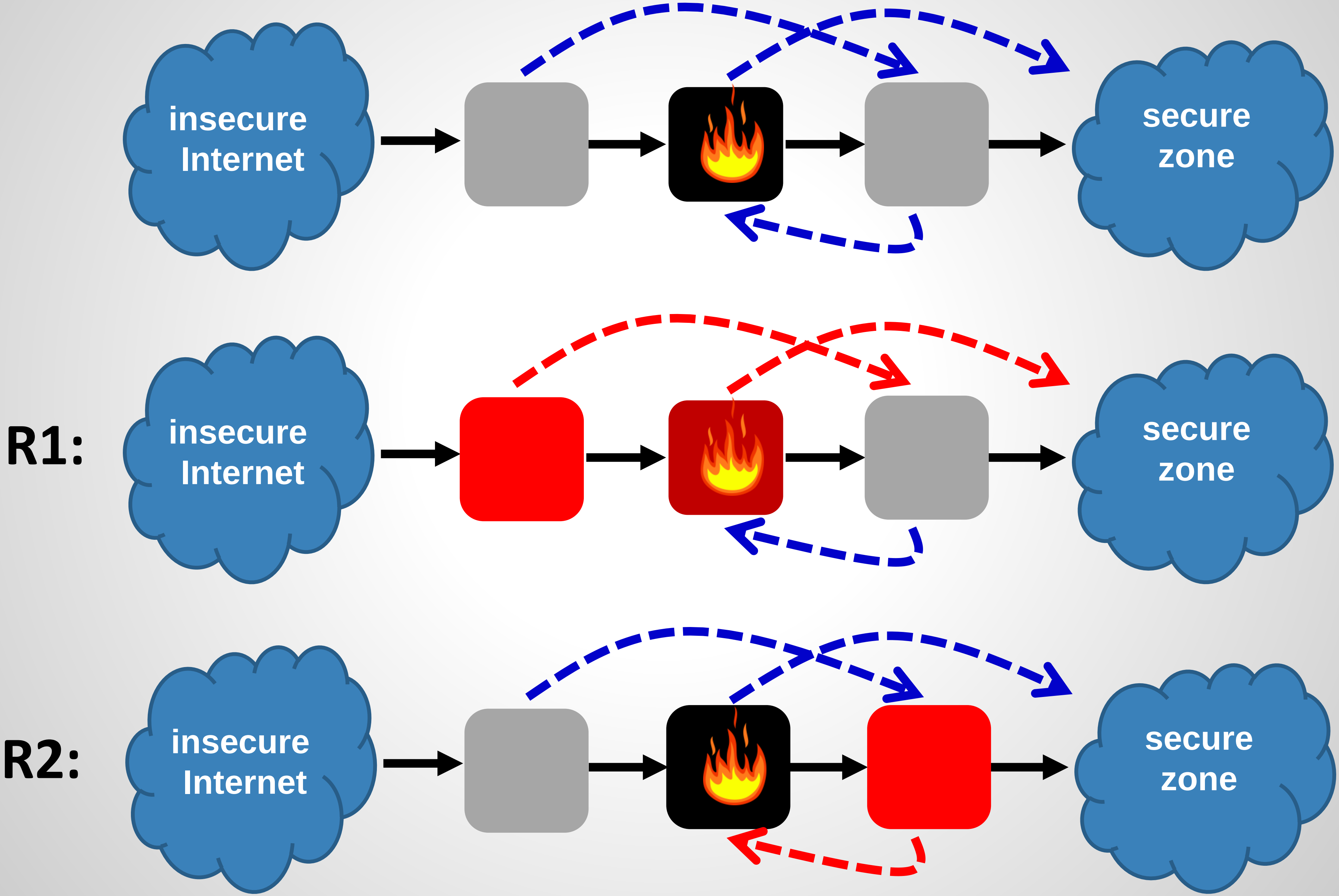
Strong

Weak

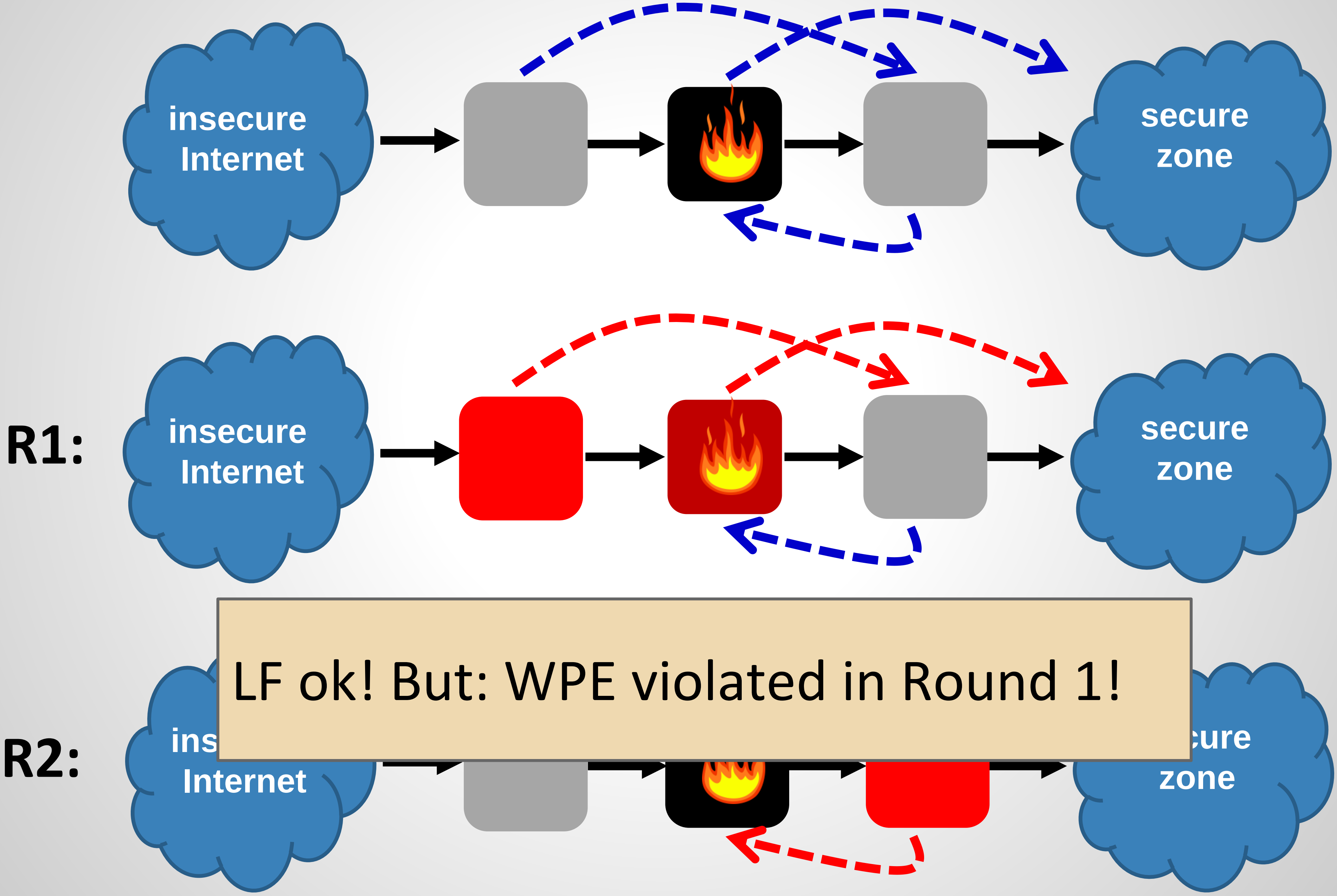
Going Back to Our Examples: LF Update?



Going Back to Our Examples: LF Update!

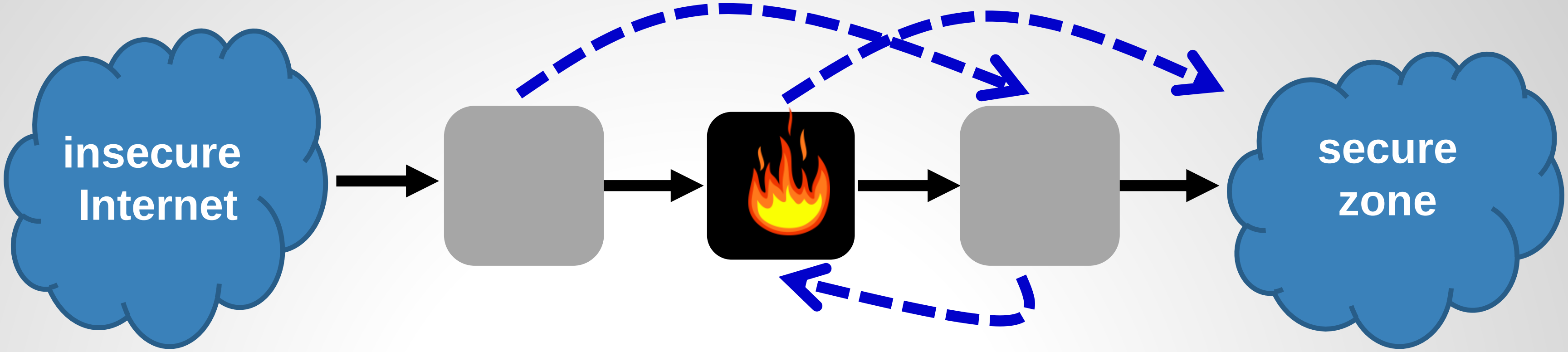


Going Back to Our Examples: LF Update!

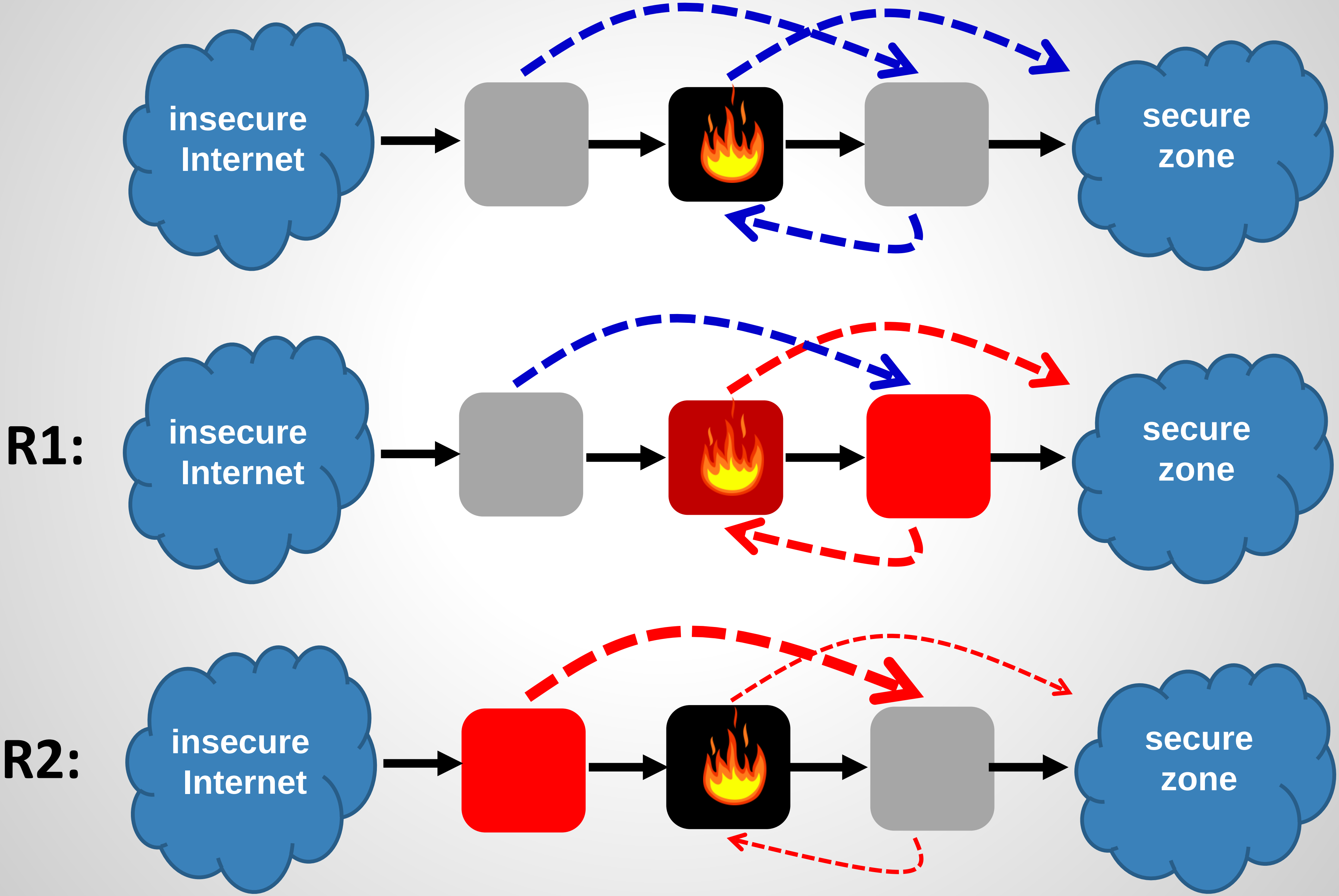


LF ok! But: WPE violated in Round 1!

Going Back to Our Examples: WPE Update?



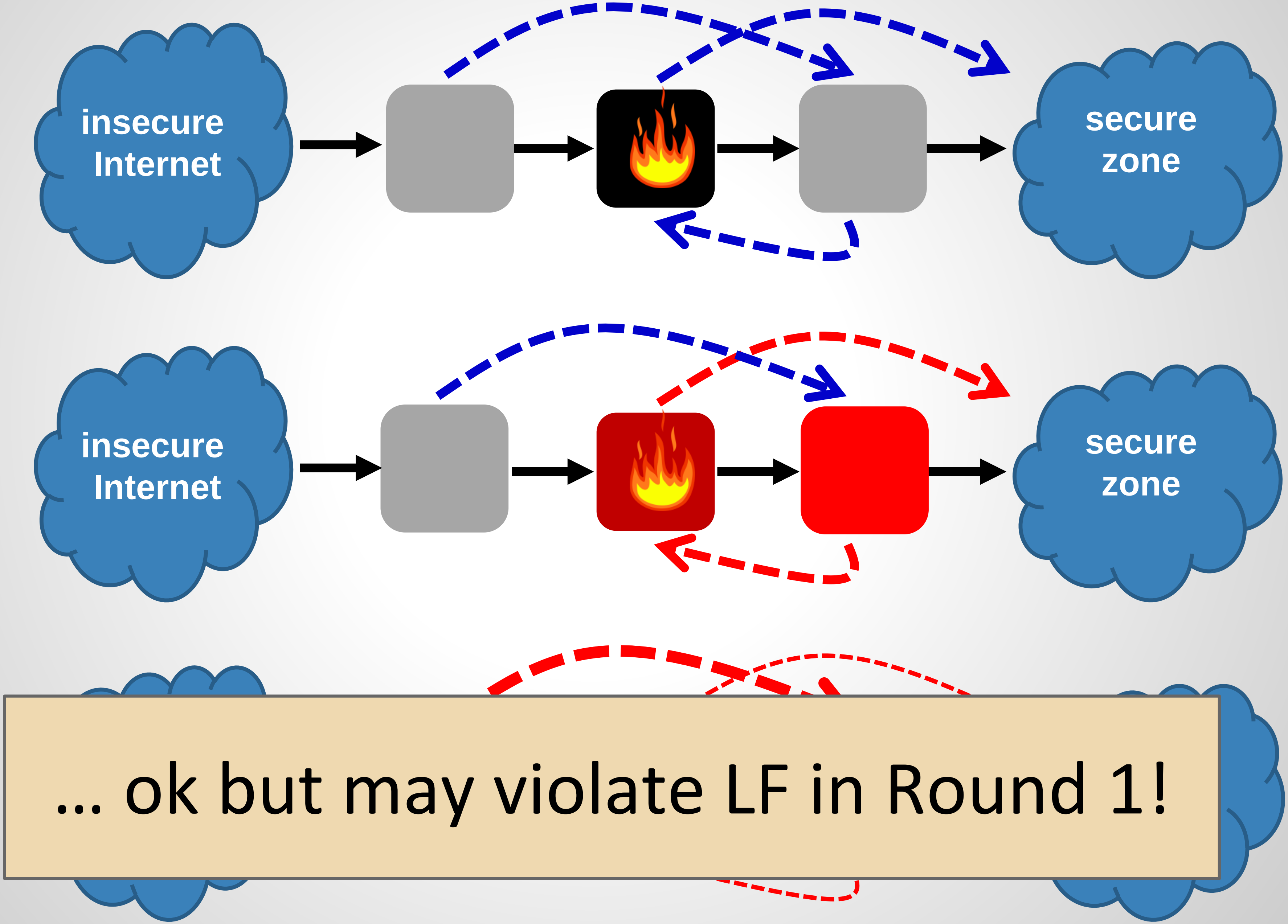
Going Back to Our Examples: WPE Update!



Going Back to Our Examples: WPE Update!

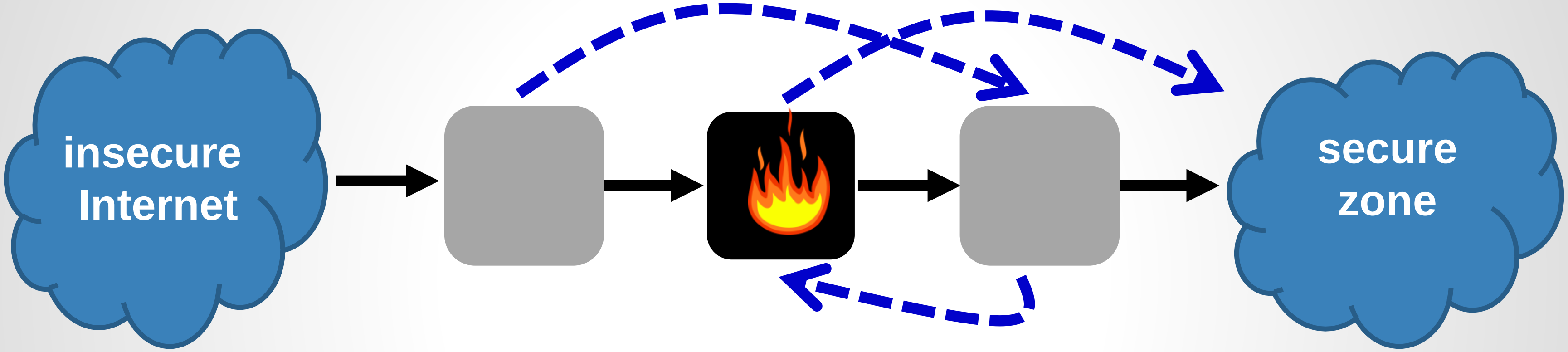
R1:

R2:

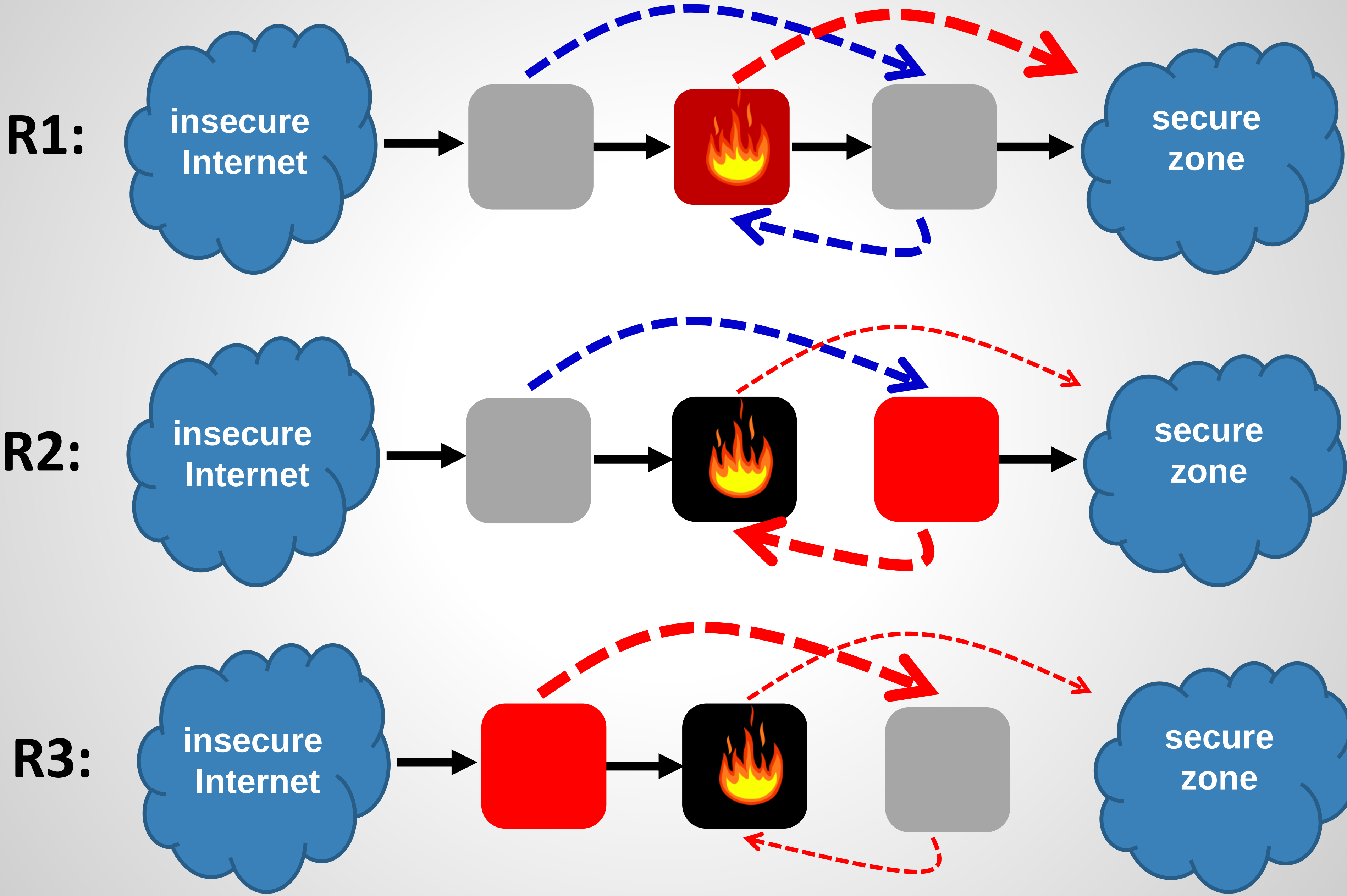


… ok but may violate LF in Round 1!

Going Back to Our Examples: Both WPE+LF?



Going Back to Our Examples: WPE+LF!



R1:

insecure Internet

secure zone

R2:

insecure Internet

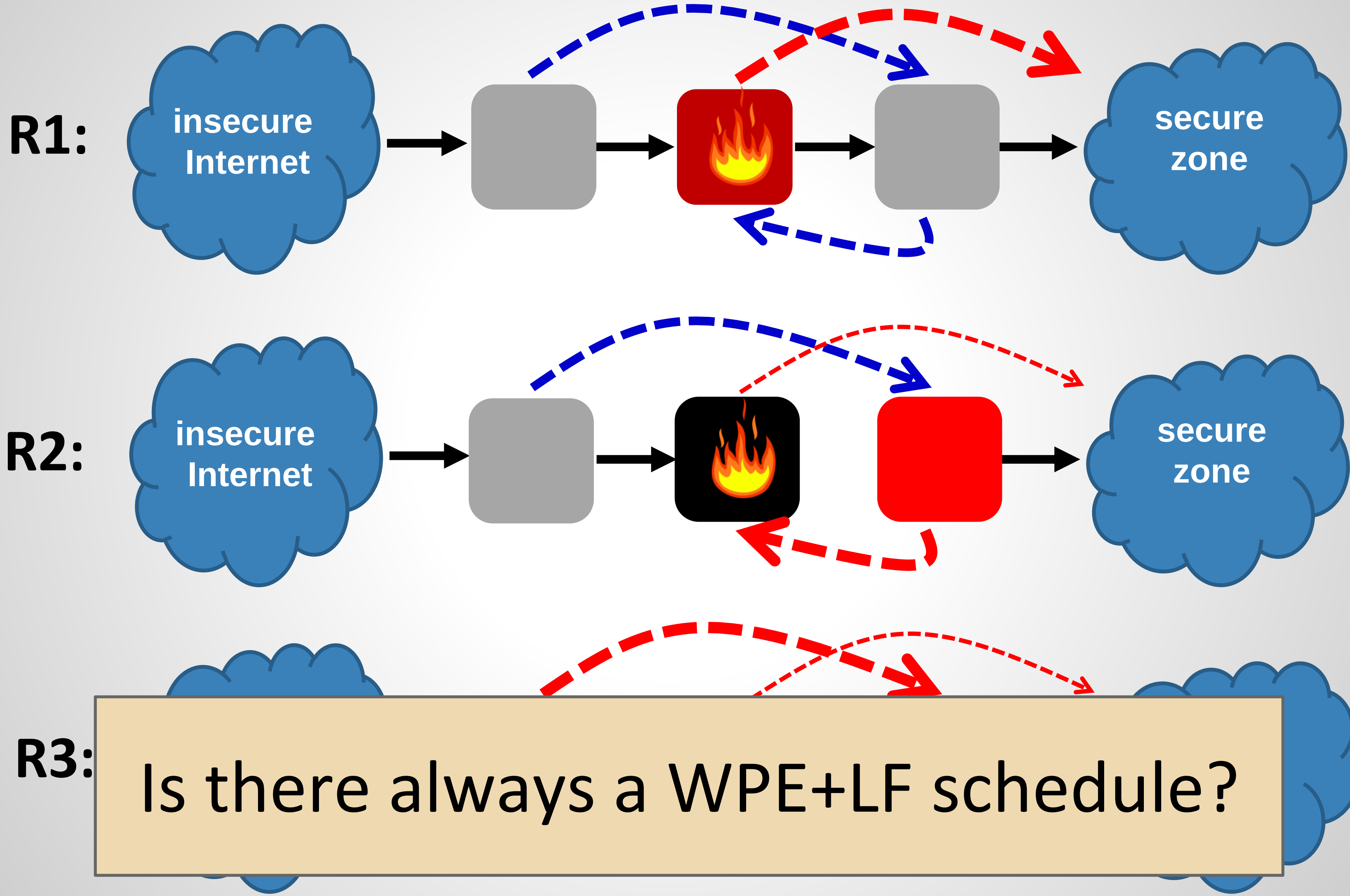
secure zone

R3:

insecure Internet

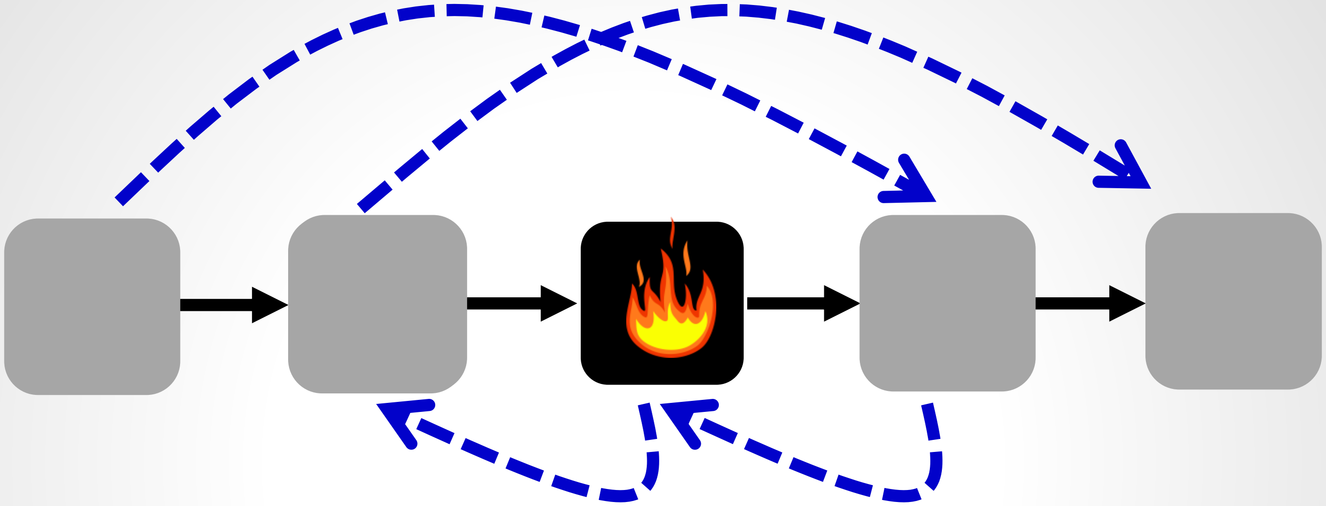
secure zone

Going Back to Our Examples: WPE+LF!

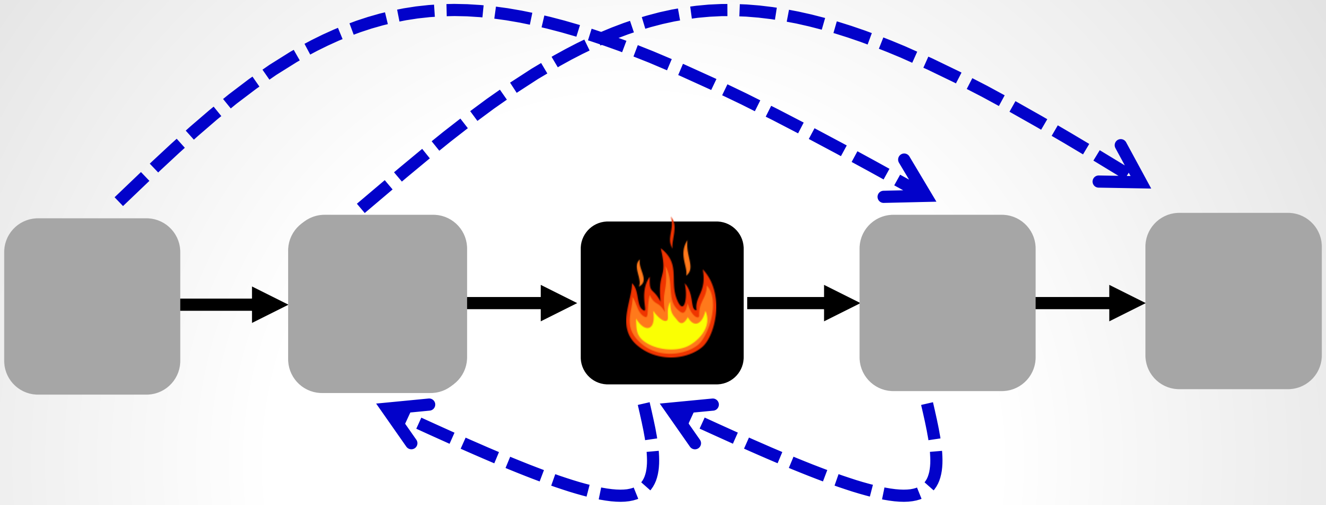


Is there always a WPE+LF schedule?

What about this one?



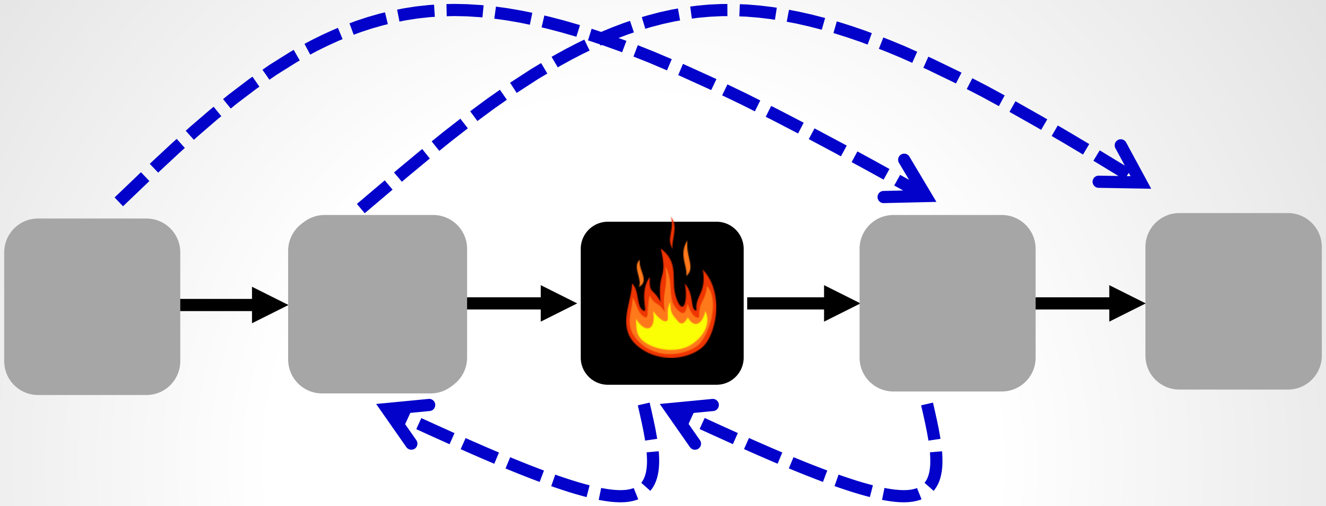
LF and WPE may conflict!



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

No schedule exists!

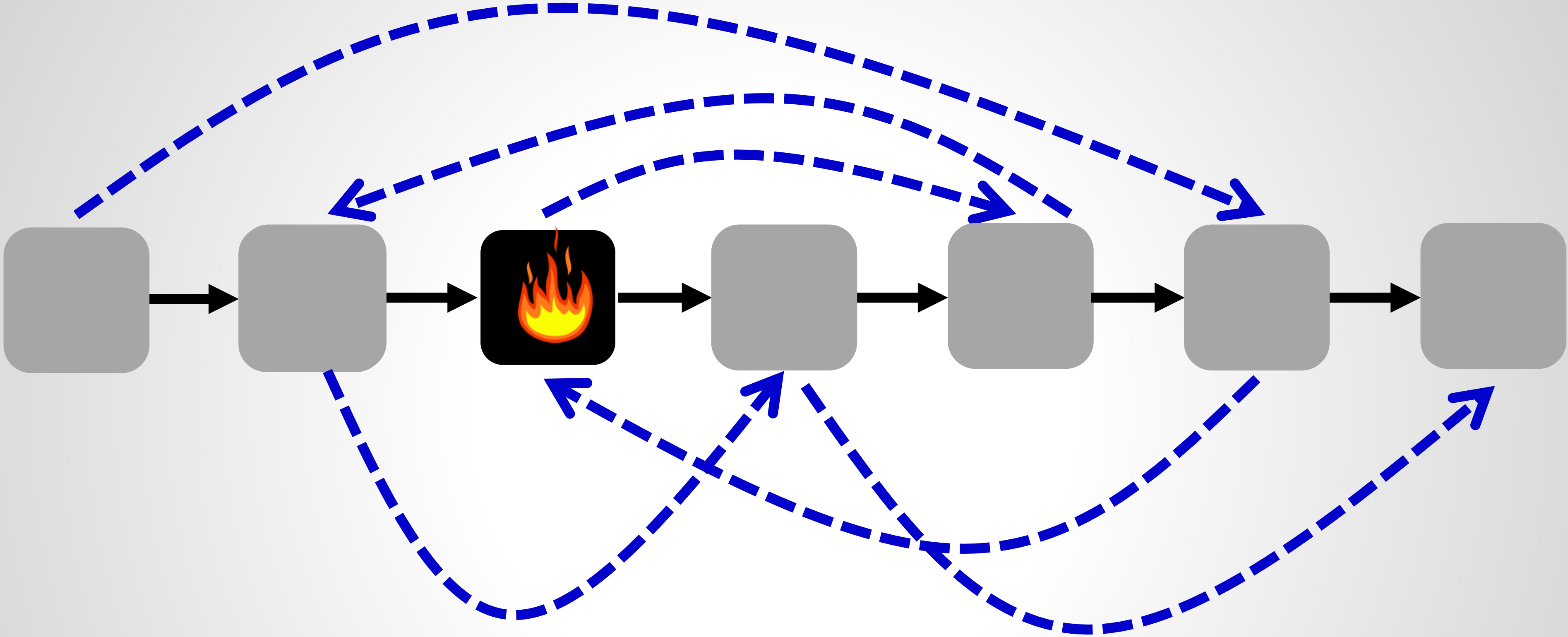
LF and WPE may conflict!



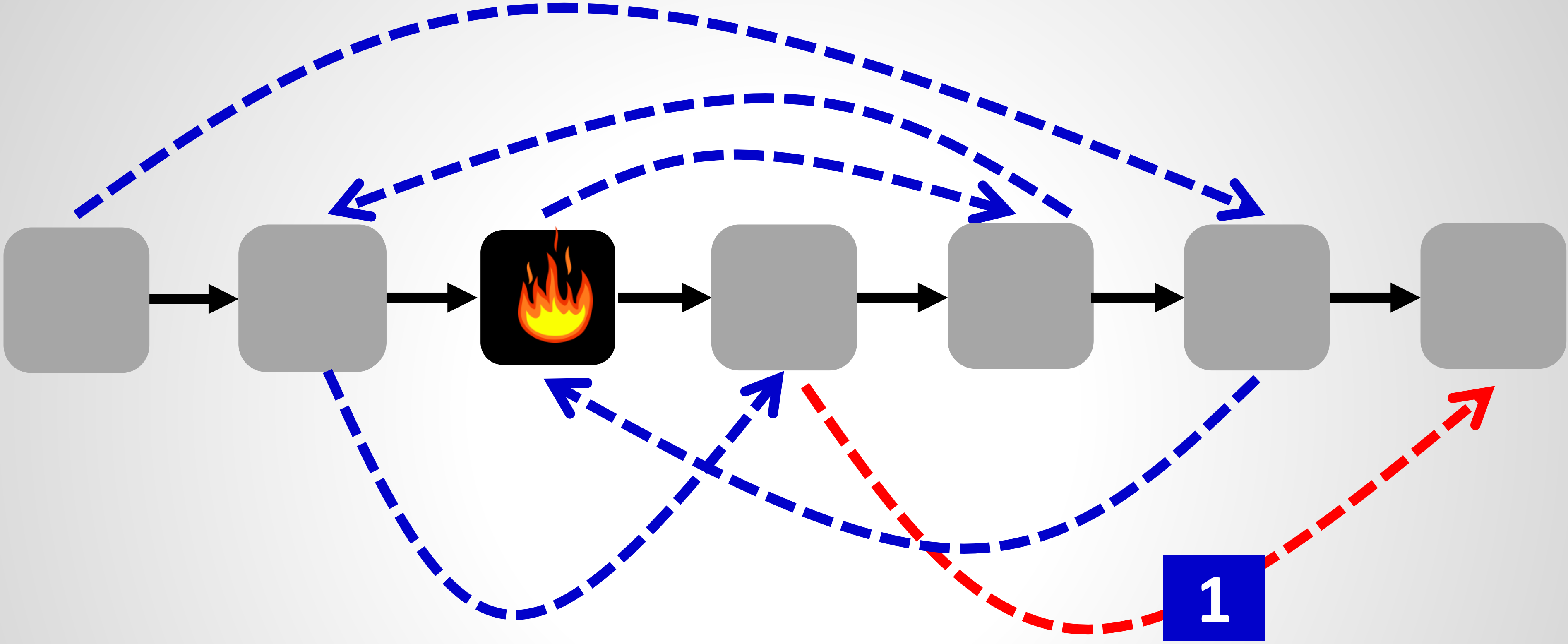
- ❑ Cannot update any **forward edge** in R1: WP
- ❑ Cannot update any **backward edge** in R1: LF

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.

What about this one?

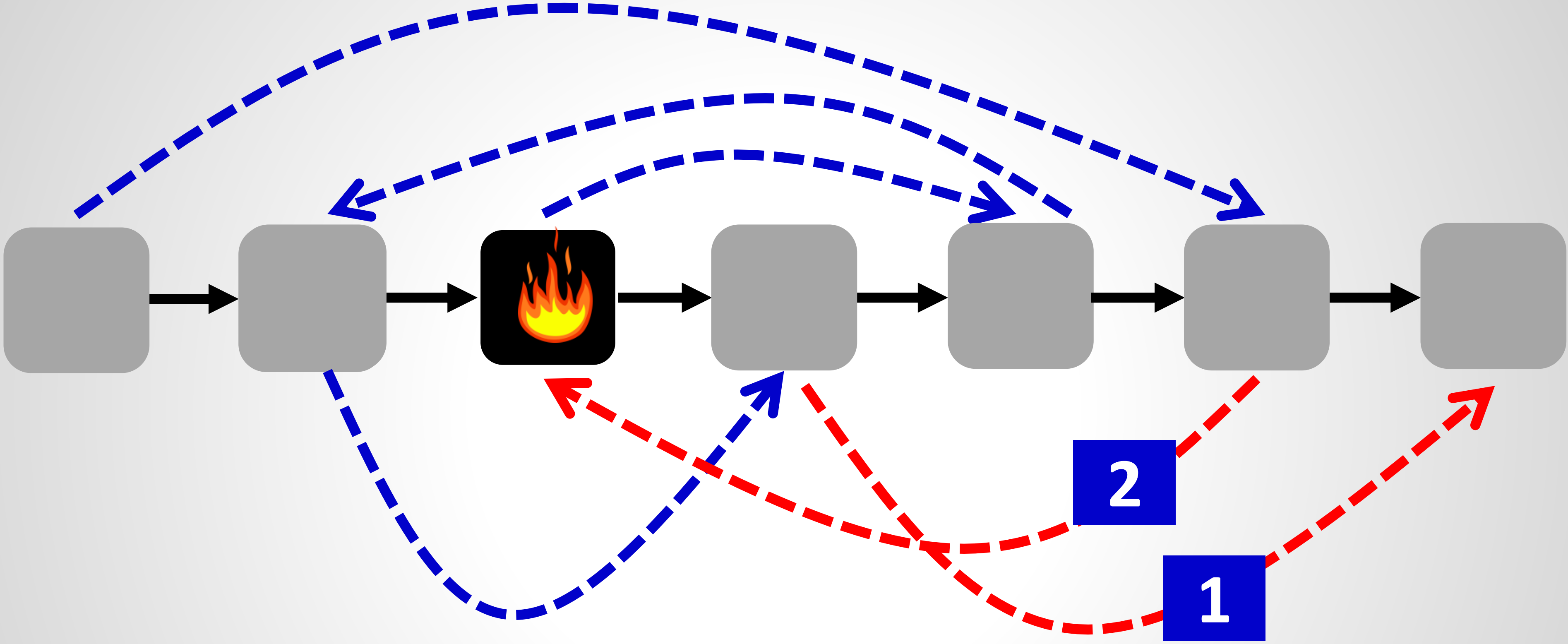


What about this one?



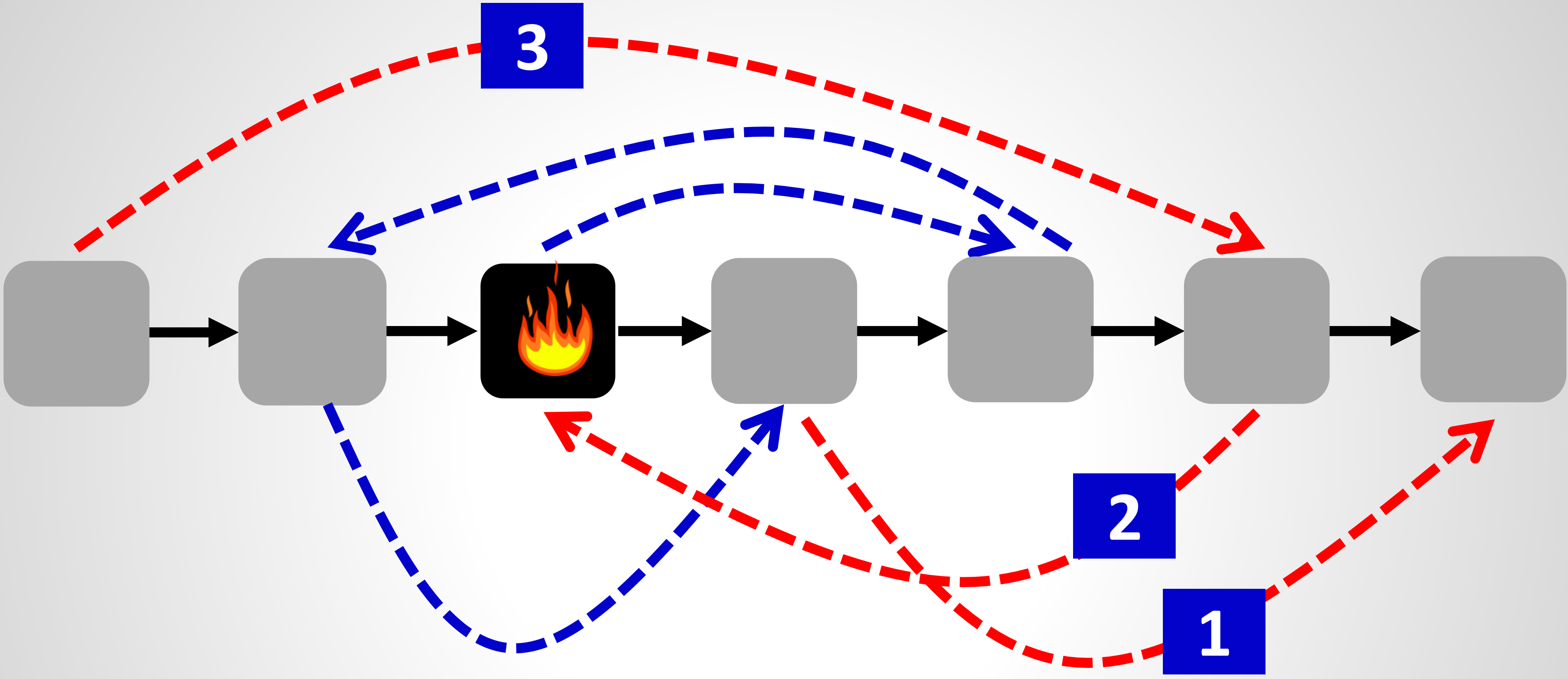
- ❑ Forward edge after the waypoint: safe!
  - ❑ No loop, no WPE violation

What about this one?



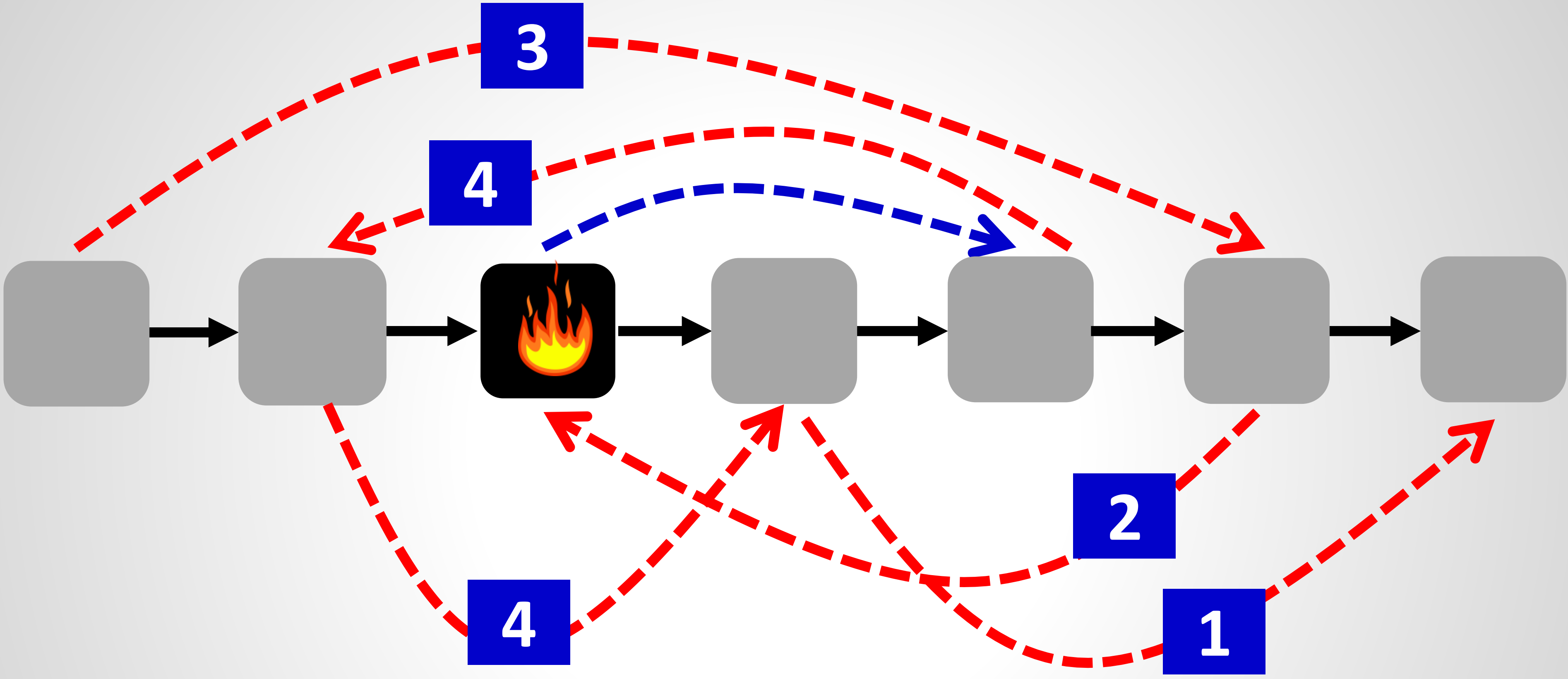
- ❑ Now this backward is safe too!
  - ❑ No loop because exit through 1

What about this one?



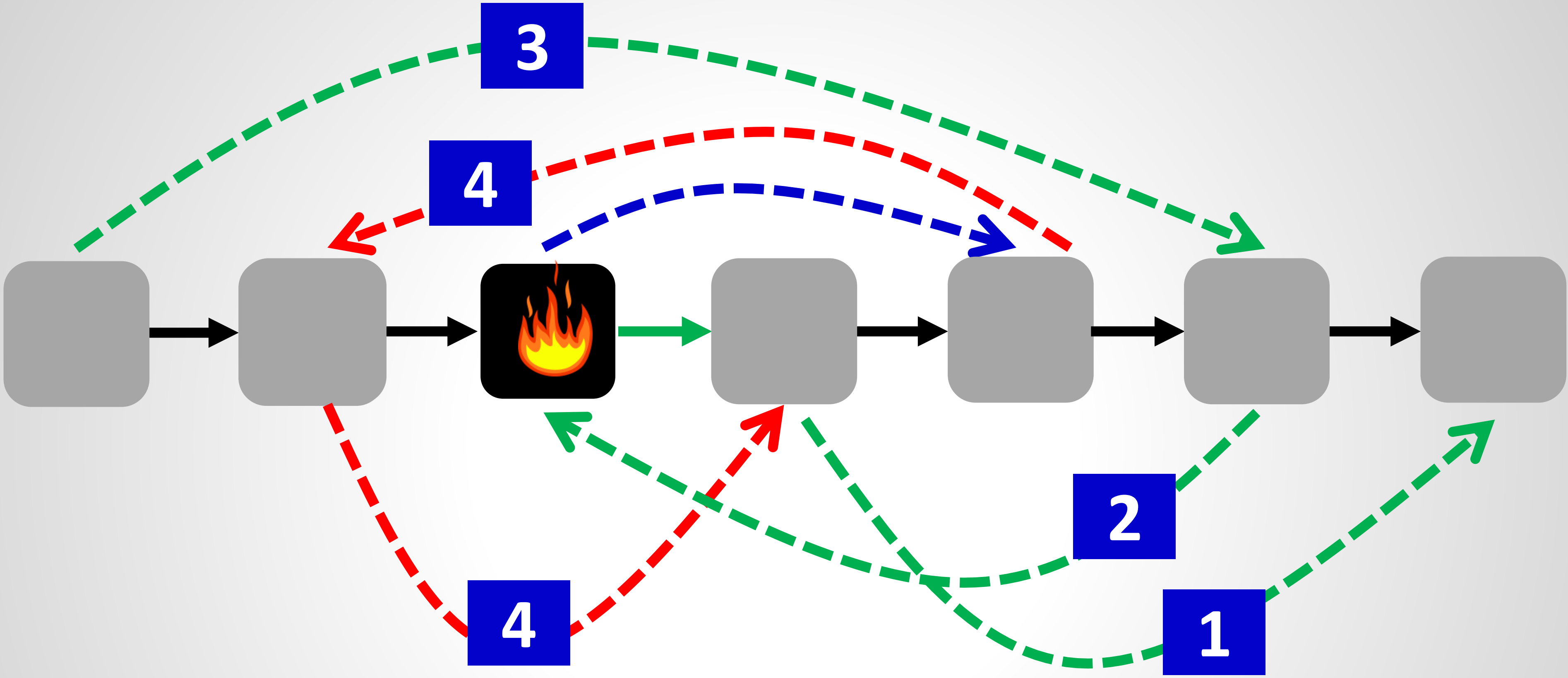
- ❑ Now this is safe: **2** ready back to WP!
  - ❑ No waypoint violation

What about this one?



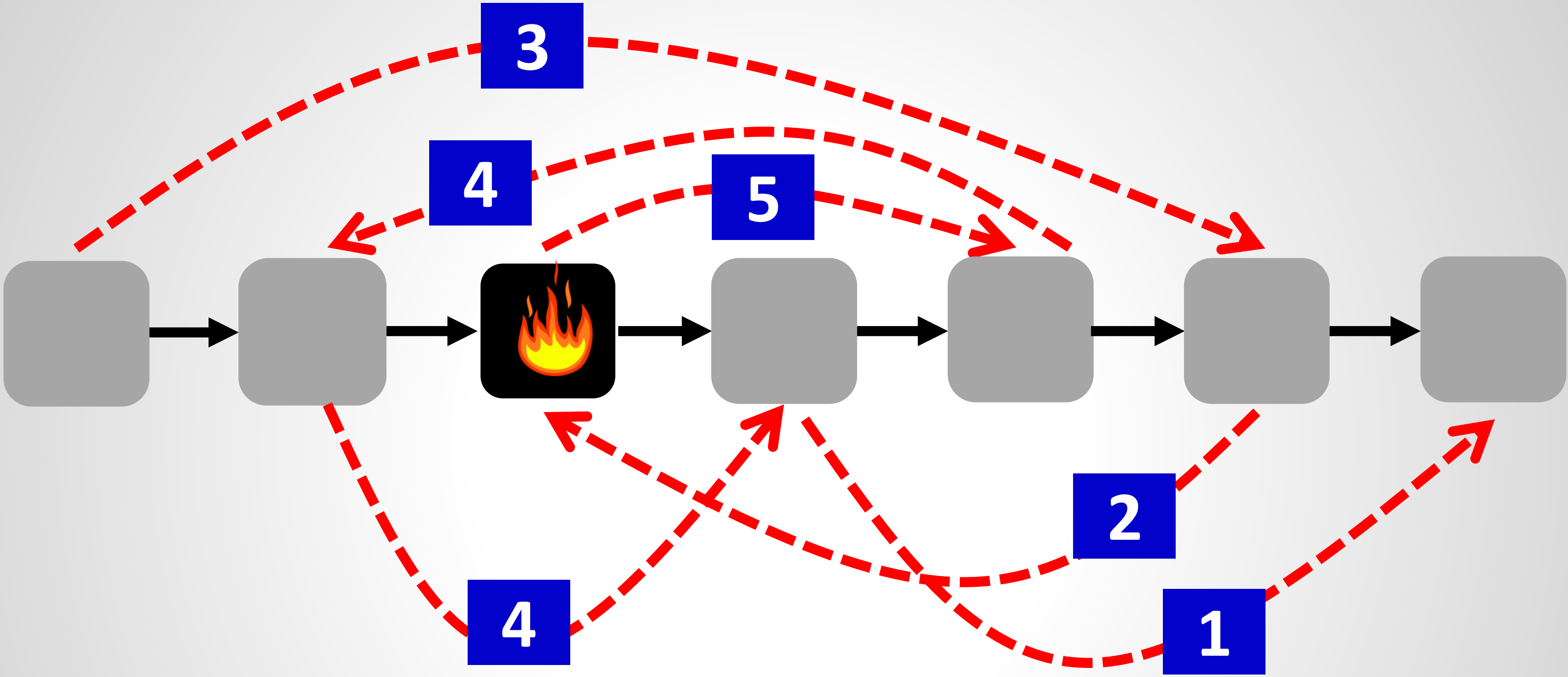
- ❑ Ok: loop-free and also not on the path (exit via 1)

What about this one?

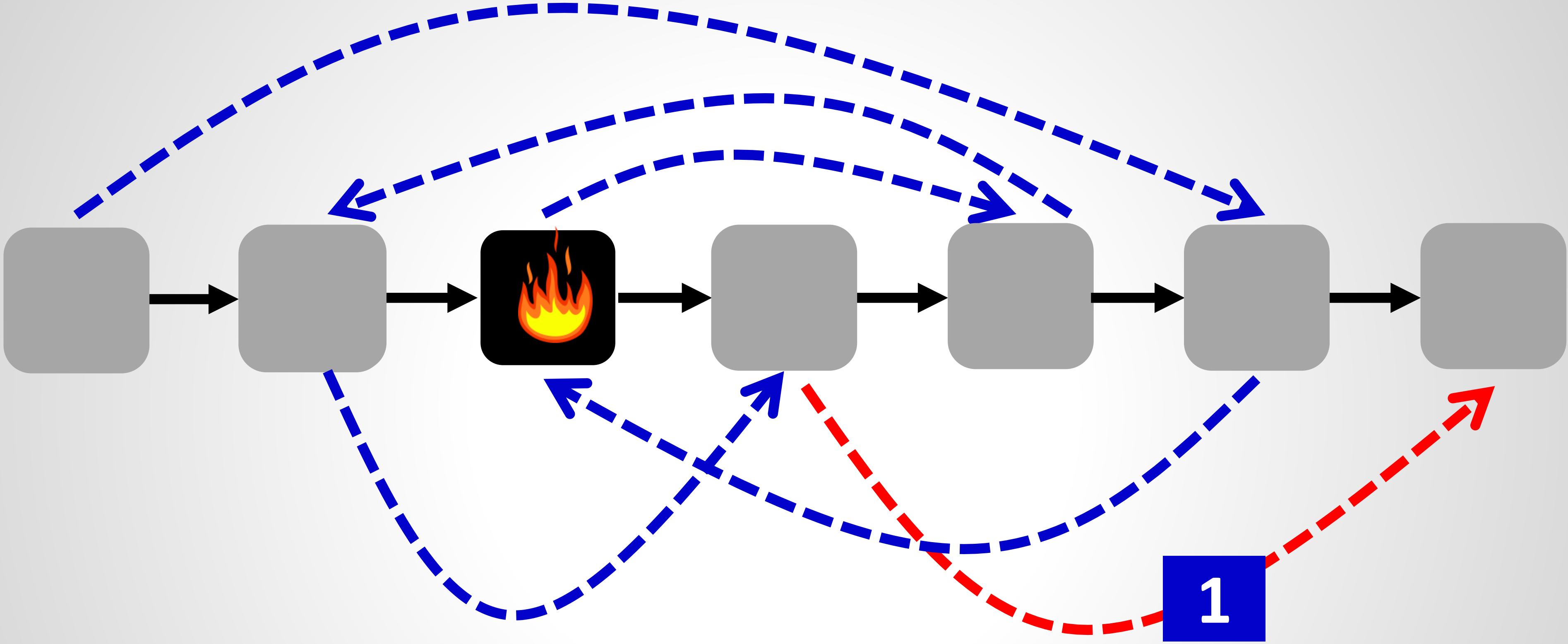


- ❑ Ok: loop-free and also not on the path (exit via 1)

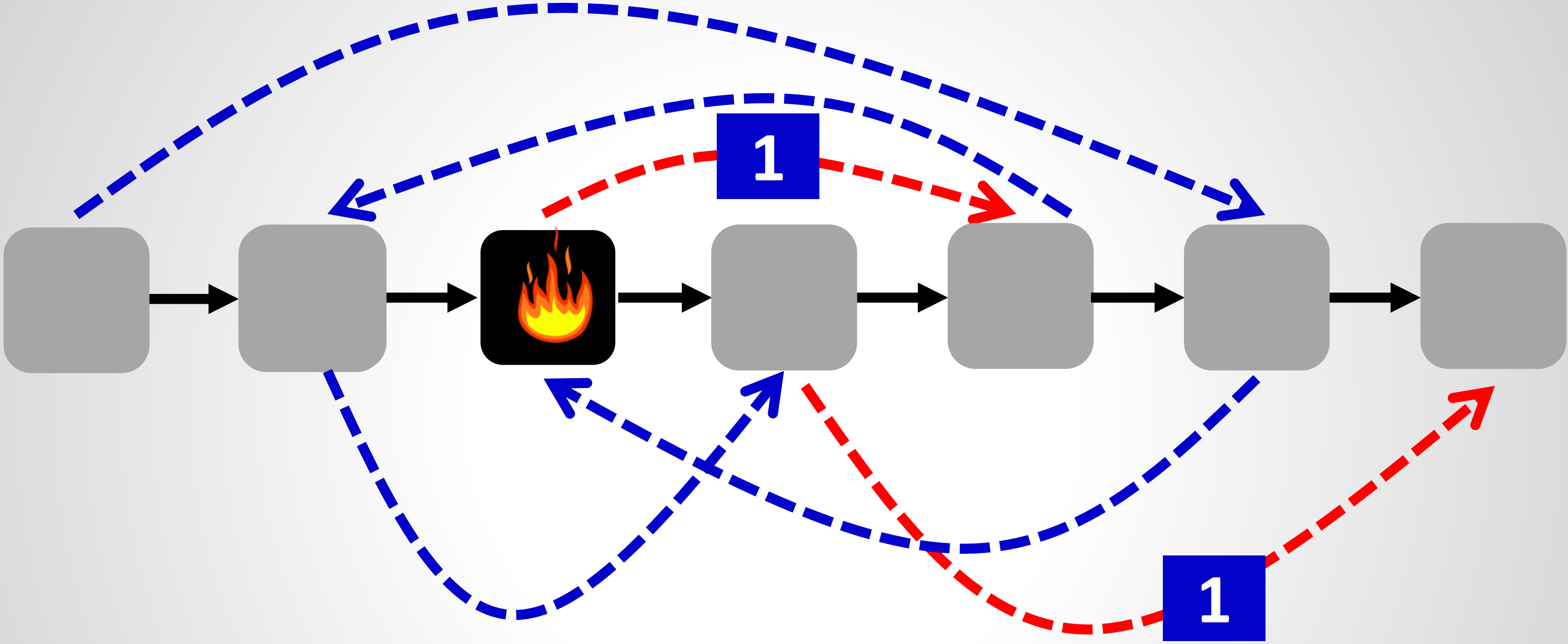
What about this one?



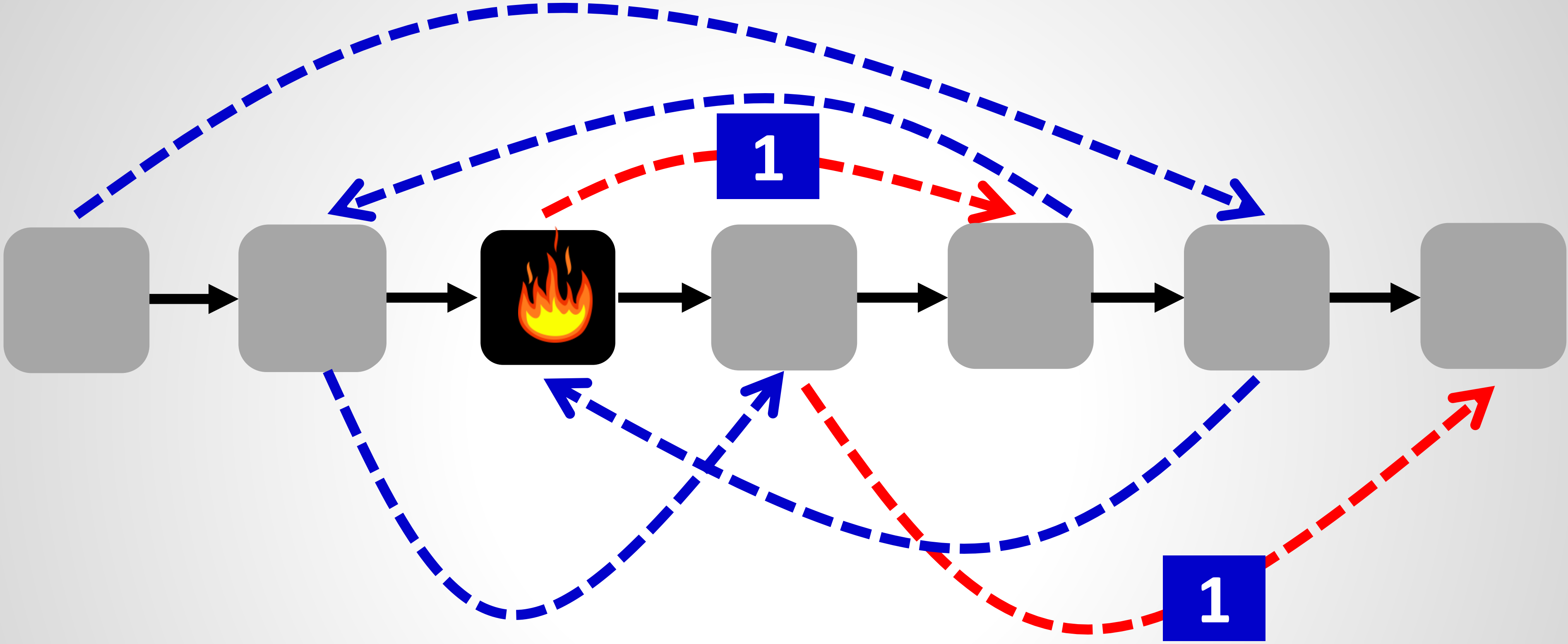
Back to the start: What if....



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

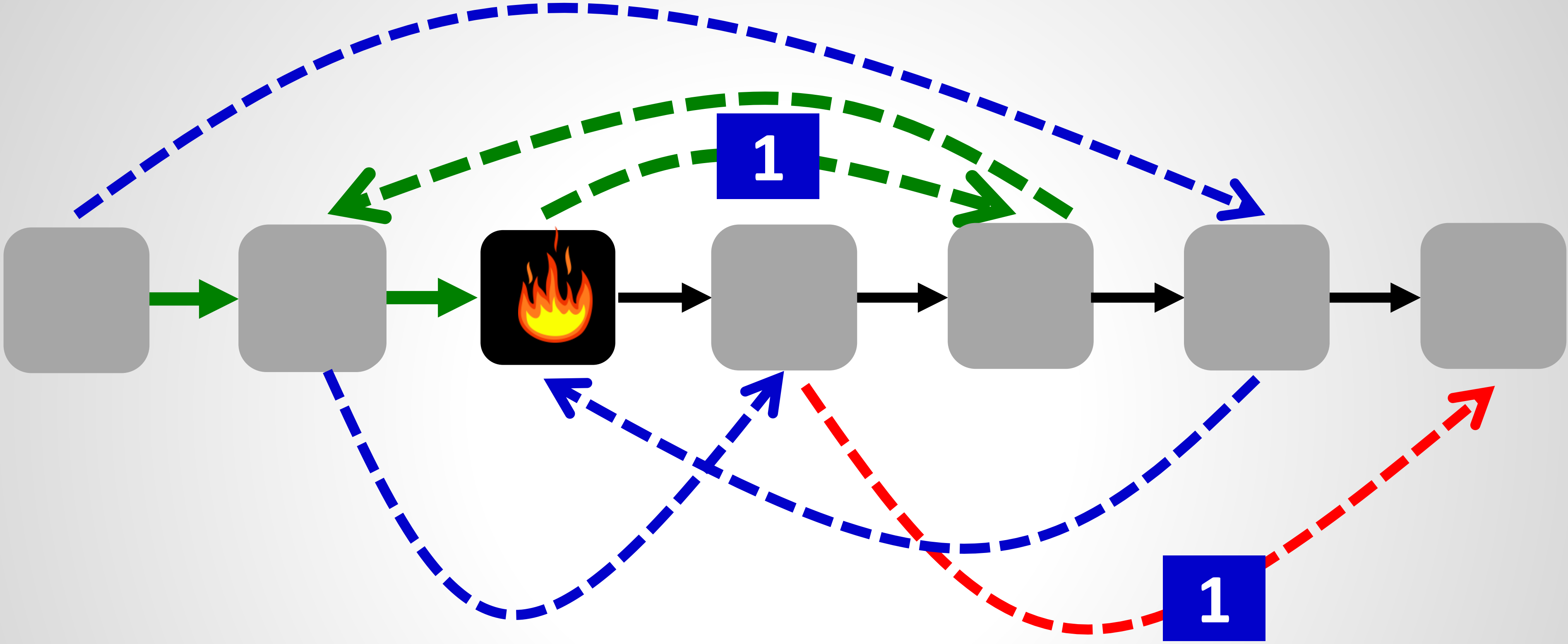


Back to the start: What if…. also this one?!



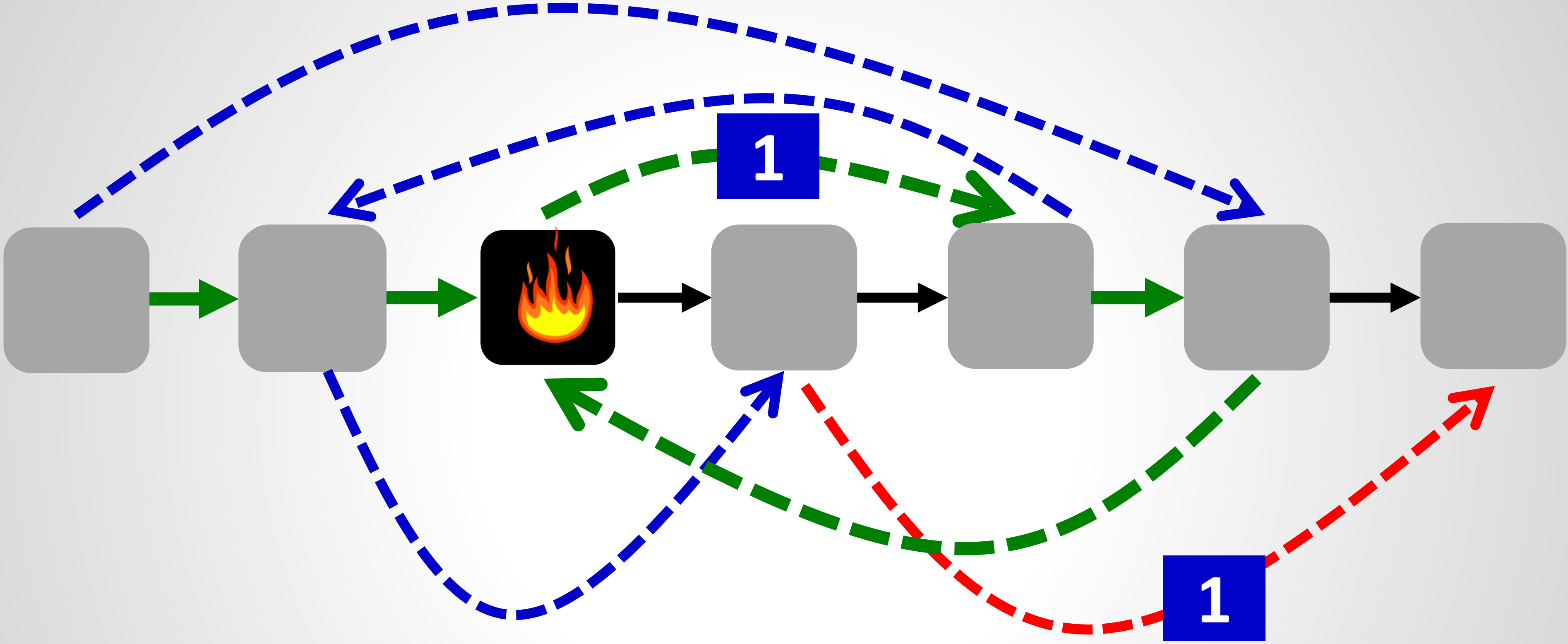
- ❑ Update any of the 2 backward edges? LF ☹

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



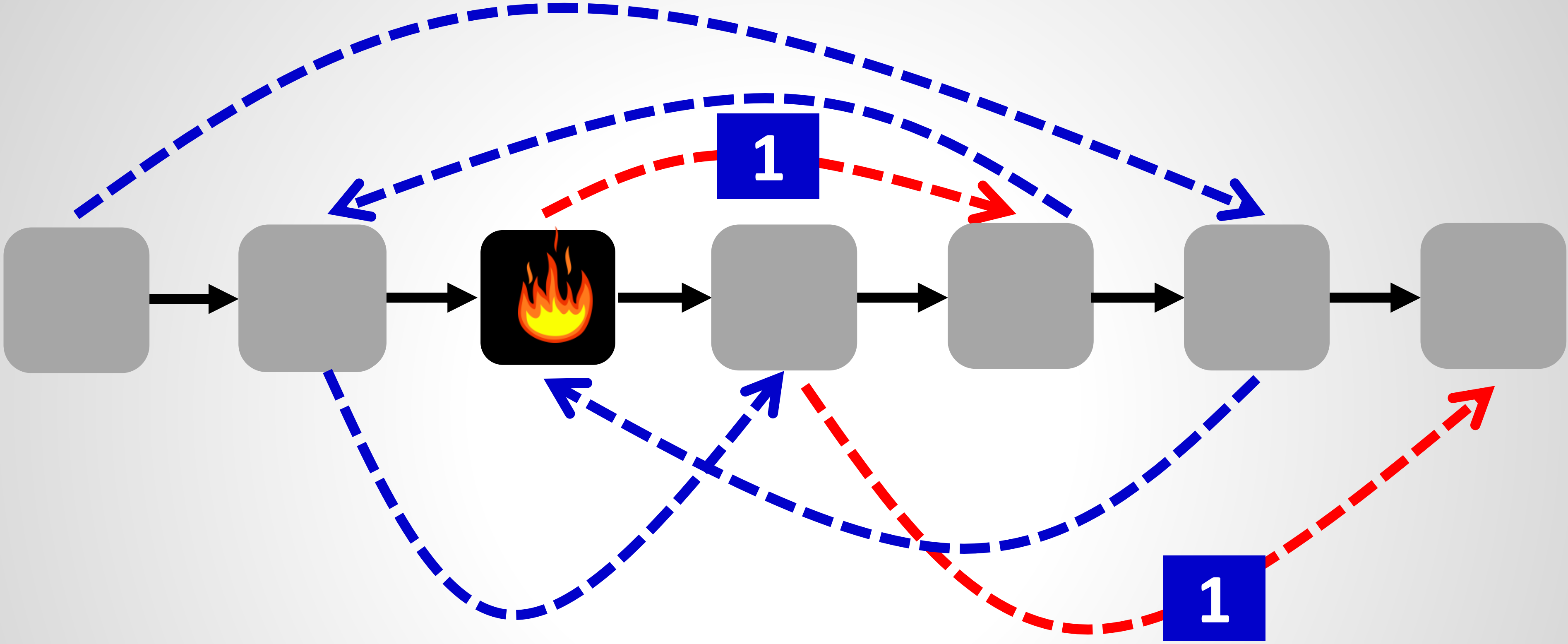
- ❑ Update any of the 2 backward edges? LF ☹

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



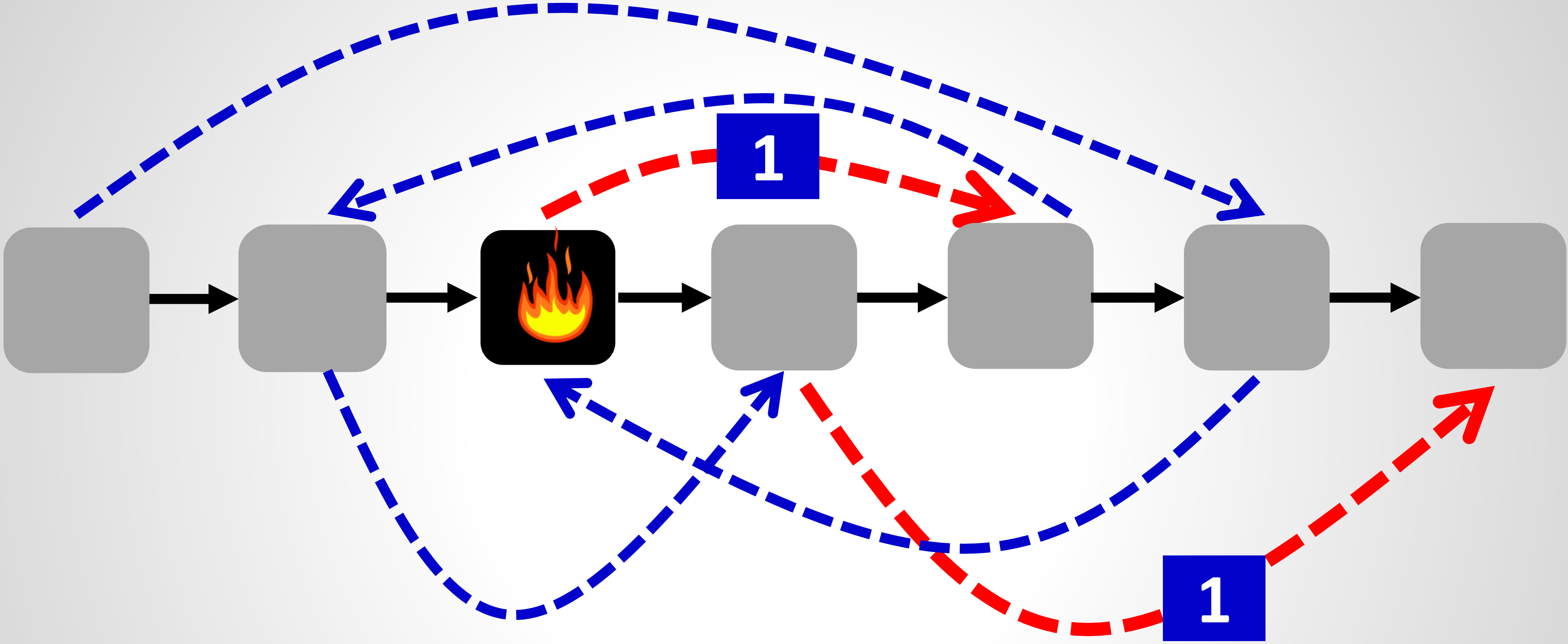
❑ Update any of the 2 backward edges? LF ☹

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

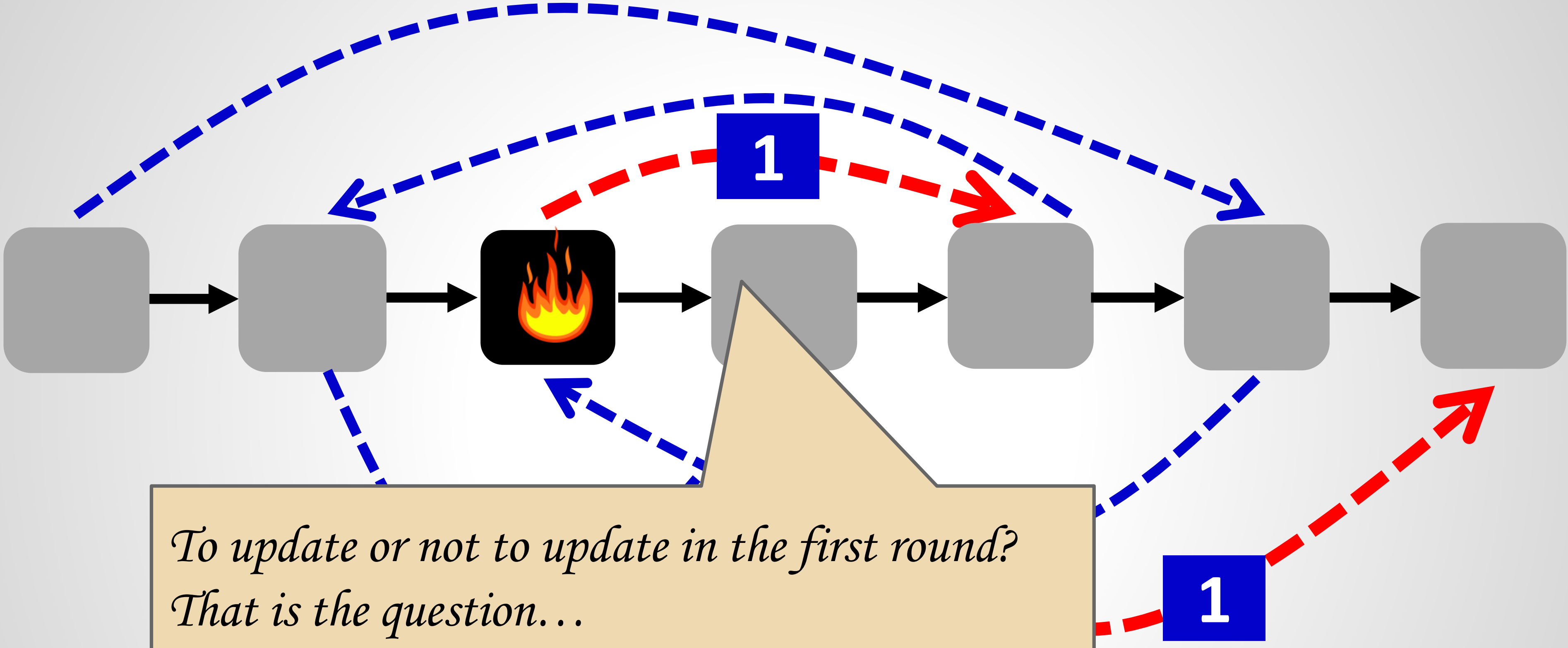


- ❑ Update any of the 2 backward edges? LF ☹
- ❑ Update any of the 2 other forward edges? WPE ☹
- ❑ What about a combination? Nope…

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

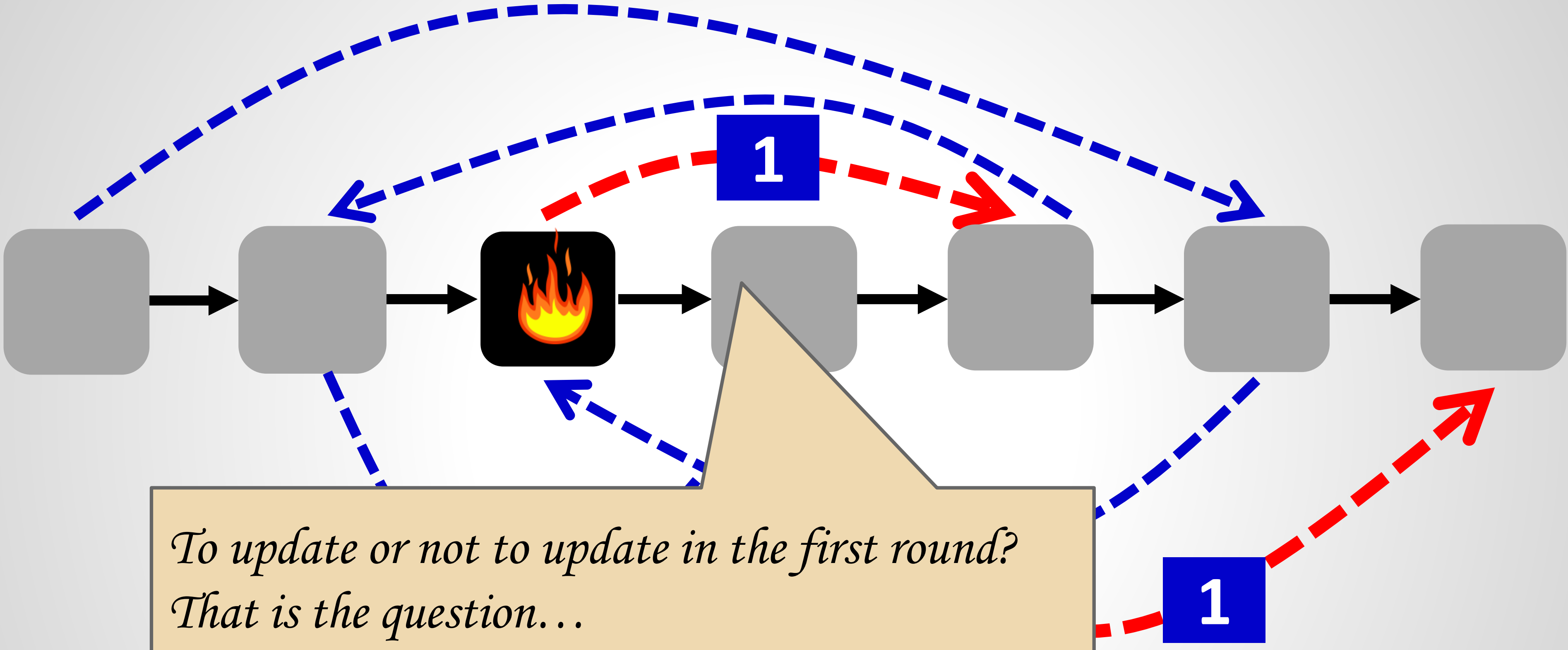


Back to the start: What if…. also this one?!



To update or not to update in the first round?
That is the question…
… which leads to NP-hardness!

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



To update or not to update in the first round?
That is the question…

… which leads to NP-hardness!

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

Let us focus on loop-freedom only: always possible in *n* rounds! (How?) But how to minimize rounds?

Example: Optimal 2-Round Update Schedules

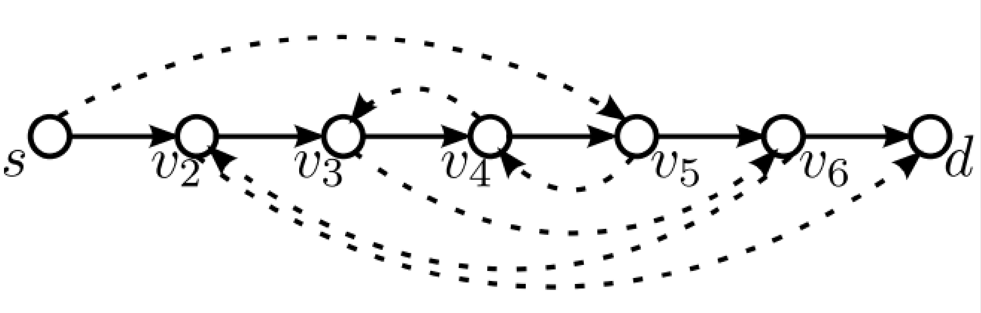
Example: Optimal 2-Round Update Schedules

Clear: in Round 1 (R1), I can only update „forward“ links!

What about last round? Observe: Update schedule read backward (i.e., updating **from new to old policy**), must also be legal! I.e., in last round (R2), I can do all „forward“ edges of old edges wrt to new ones! **Symmetry**!

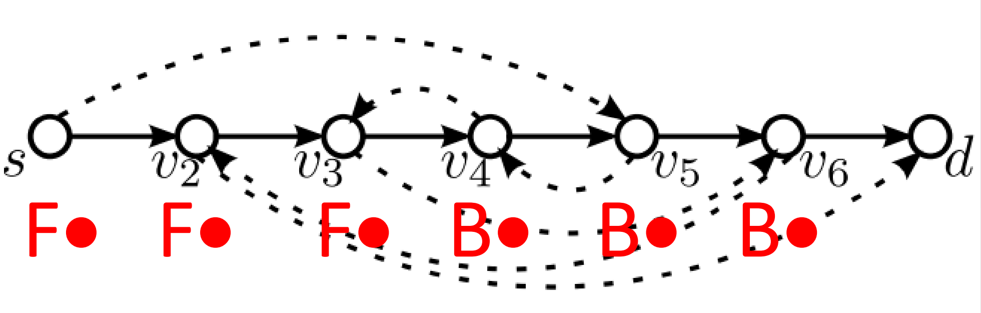
Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

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

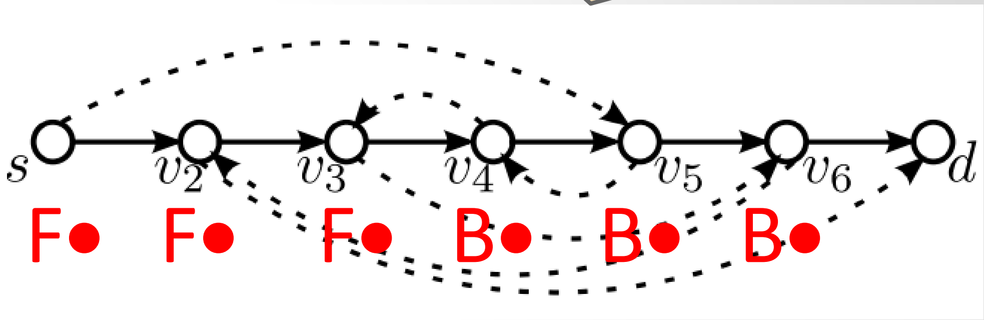


Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

- ❑ Classify nodes/edg

Old policy from left to right!

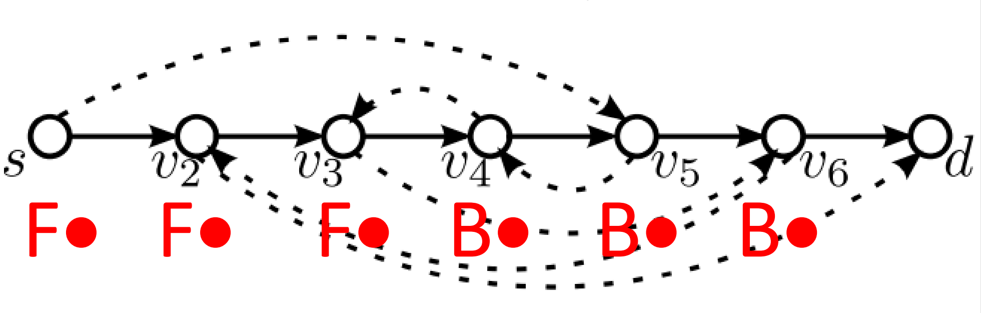
  - ❑ F•, B•: Does (dashed) new edge point forward or backward wrt (solid) old path?

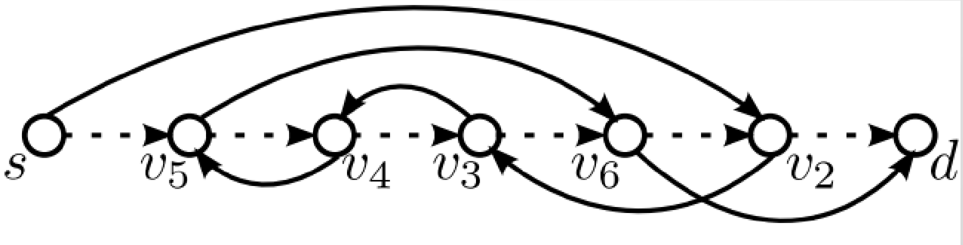


Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

Old policy from left to right!

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



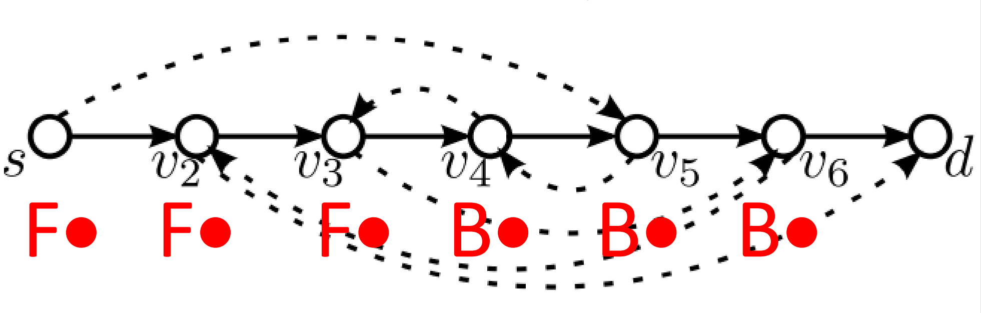


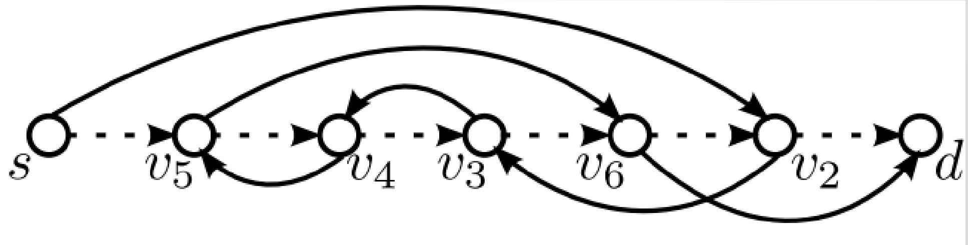
New policy from left to right!

Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

Old policy from left to right!

- ❑ Classify nodes/edg
  - ❑ 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?



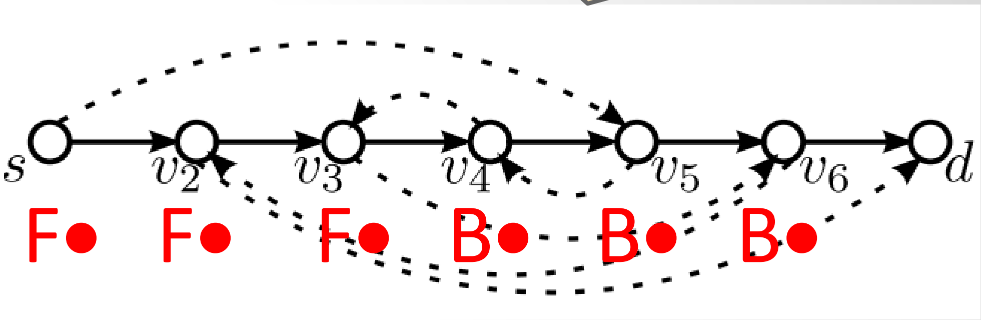


New policy from left to right!

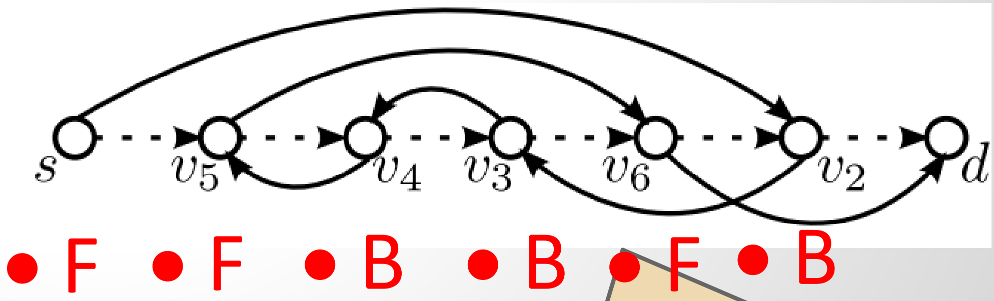
Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

Old policy from left to right!

- ❑ Classify nodes/edg
  - ❑ 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?



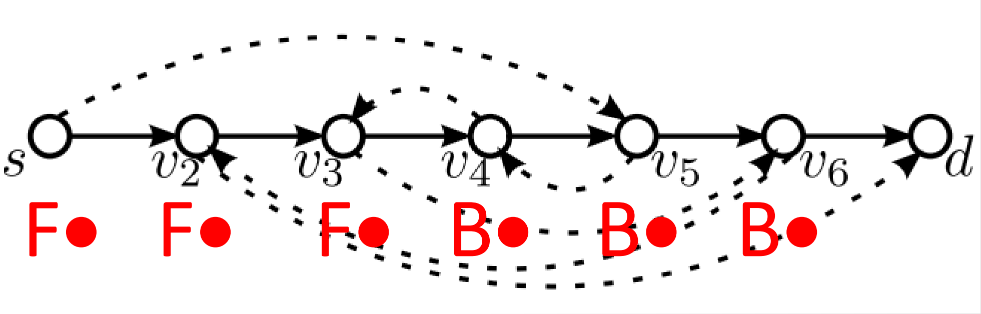
New policy from left to right!

Optimal Algorithm for 2-Round Instances:
ging Symmetry!

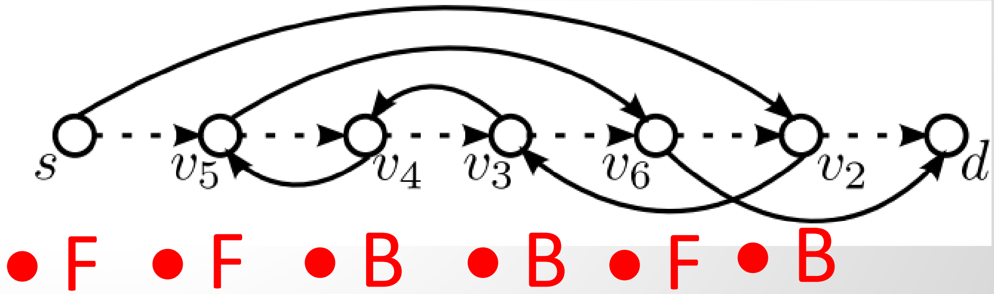
❑ es with 2-letter code:

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

❑ 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?



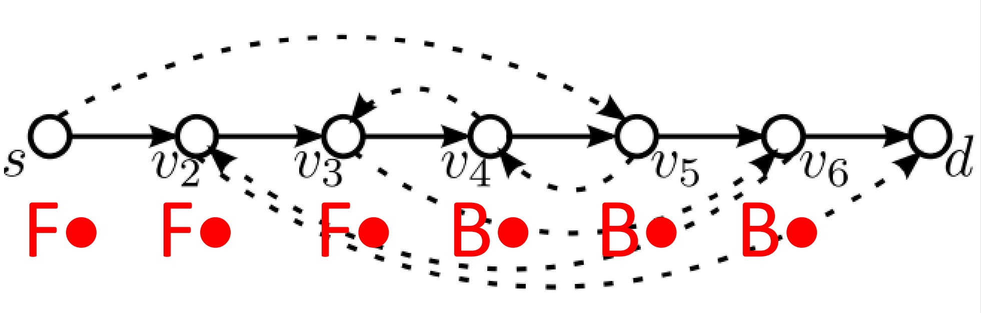
Optimal Algorithm for 2-Round Instances: ging Symmetry!

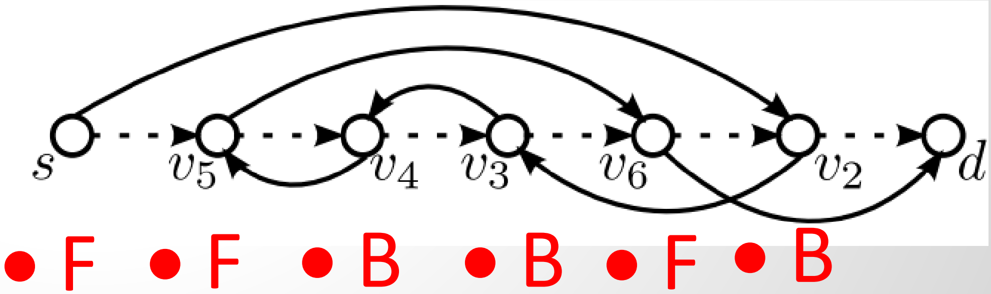
Insight 1: In the 1st round, I can safely update all forwarding (F•) edges! For sure loopfree.

es with 2-letter code:

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

old edge point forward or backward wrt (dashed) new path?





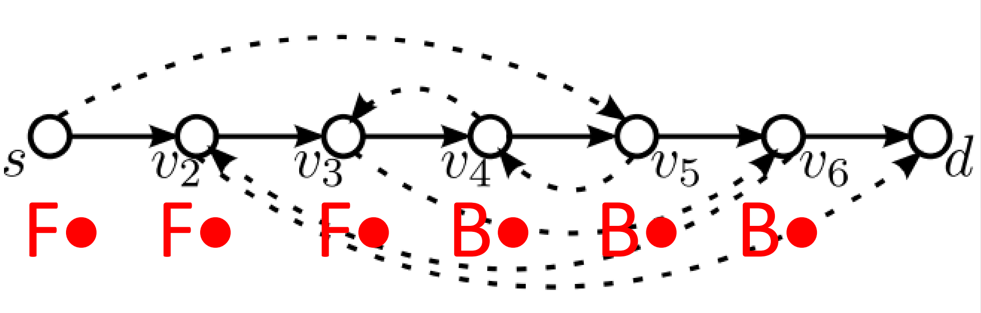
Optimal Algorithm for 2-Round Instances:
ging Symmetry!

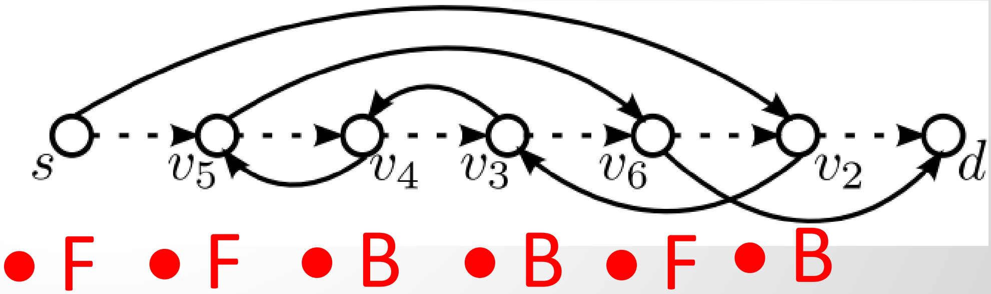
es with 2-letter code:

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!

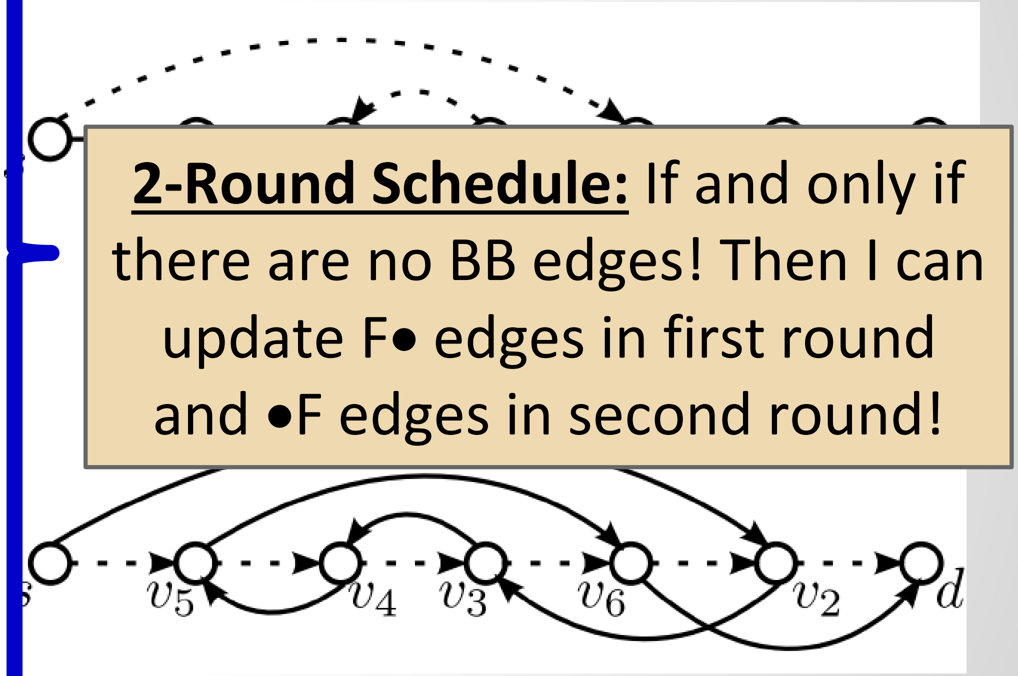
es with 2-letter code:

old edge point forward

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.



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

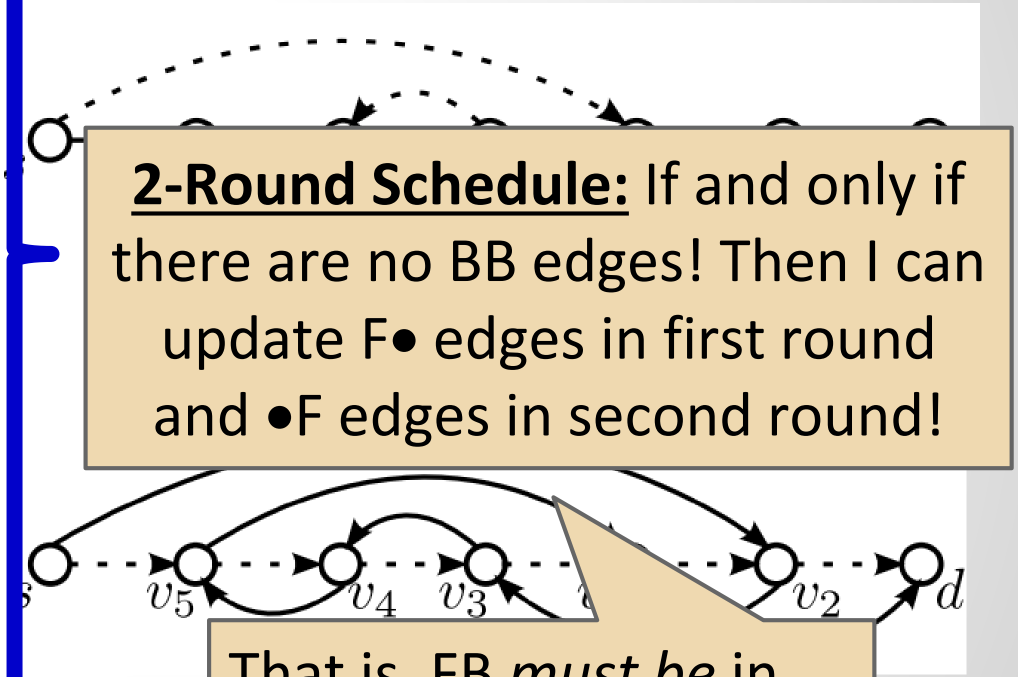
Optimal Algorithm for 2-Round Instances: ging Symmetry!

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

es with 2-letter code:

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.

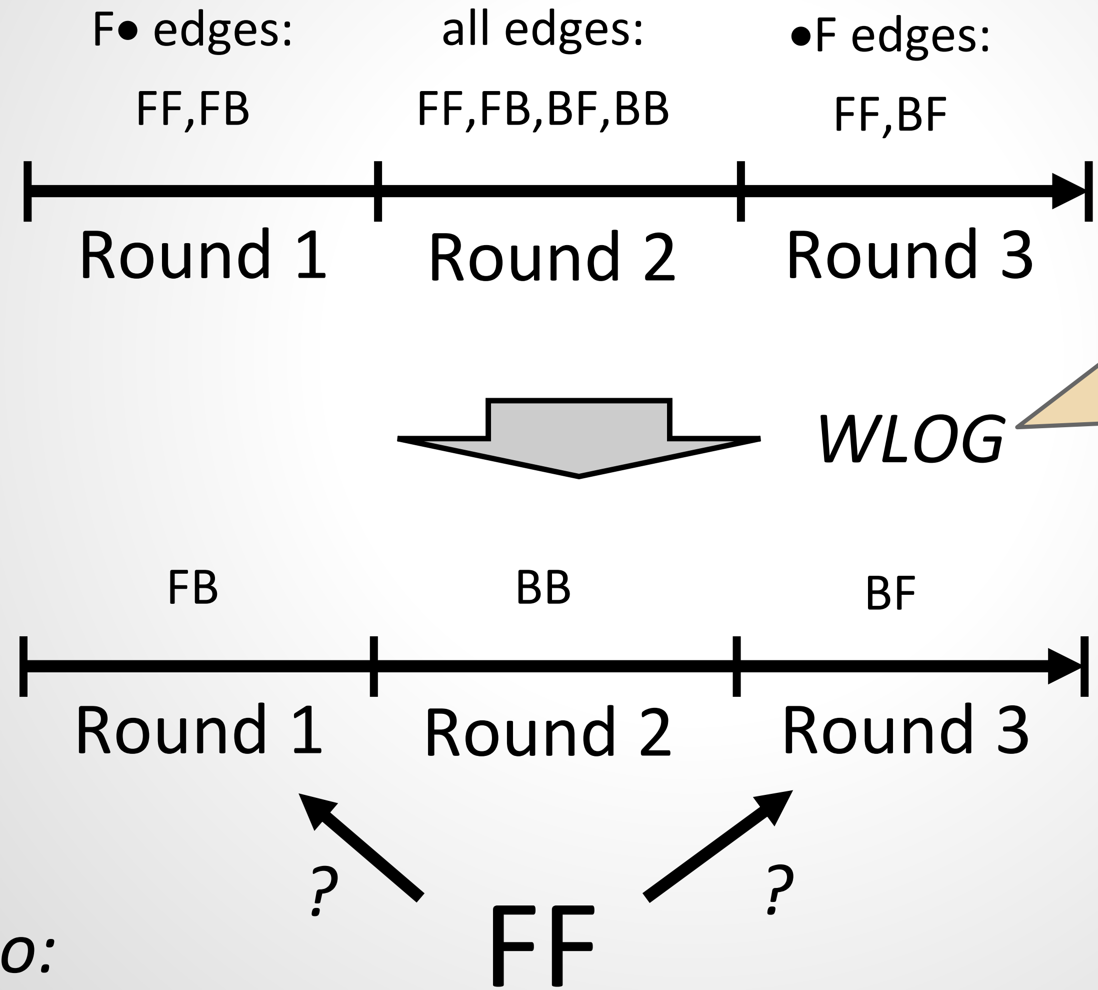


2-Round Schedule: If and only if there are no BB edges! Then I can update F• edges in first round and •F edges in second round!

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

Intuition Why 3 Rounds Are Hard

- ❑ Structure of a 3-round schedule:



F• edges: FF,FB

all edges: FF,FB,BF,BB

•F edges: FF,BF

Round 1 Round 2 Round 3

WLOG

W.l.o.g., can do FB in R1 and BF in R3.

FB BB BF

Round 1 Round 2 Round 3

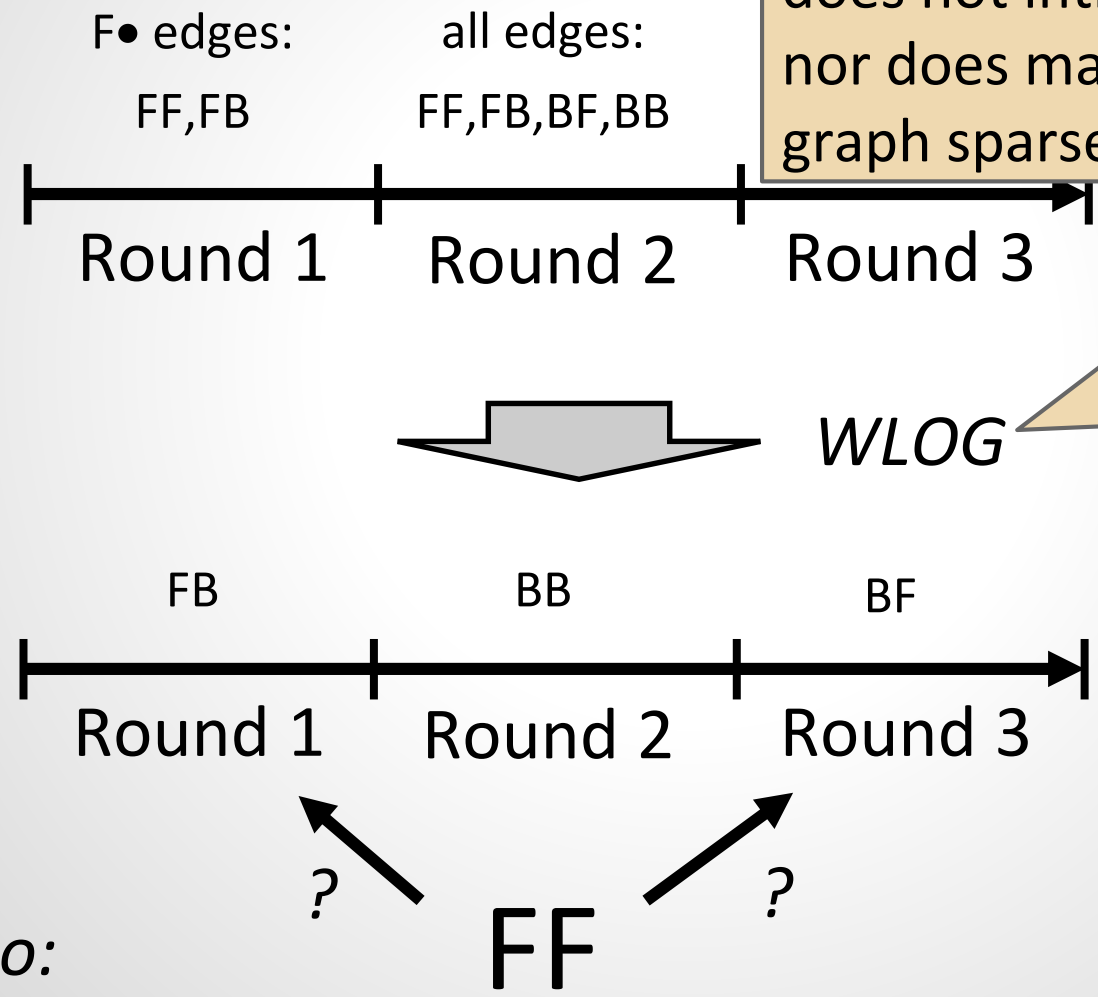
Boils down to:

? FF *?*

Intuition Why 3 Rounds Are Hard

- Structure of a 3-round sc

Moving forward edges does not introduce loops, nor does making the graph sparser.

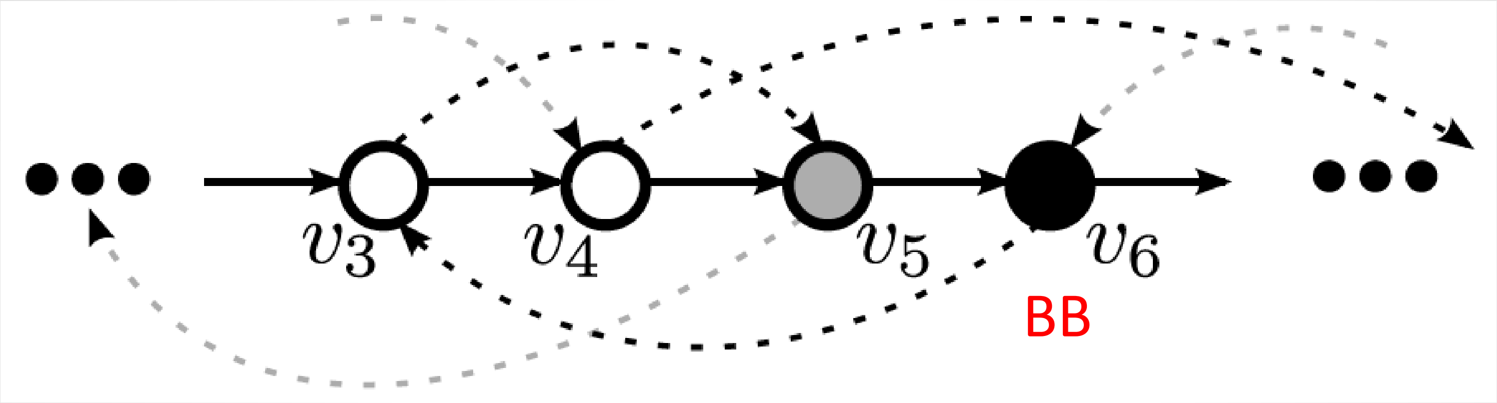


W.l.o.g., can do FB in R1 and BF in R3.

Boils down to:

Intuition Why 3 Rounds Are Hard

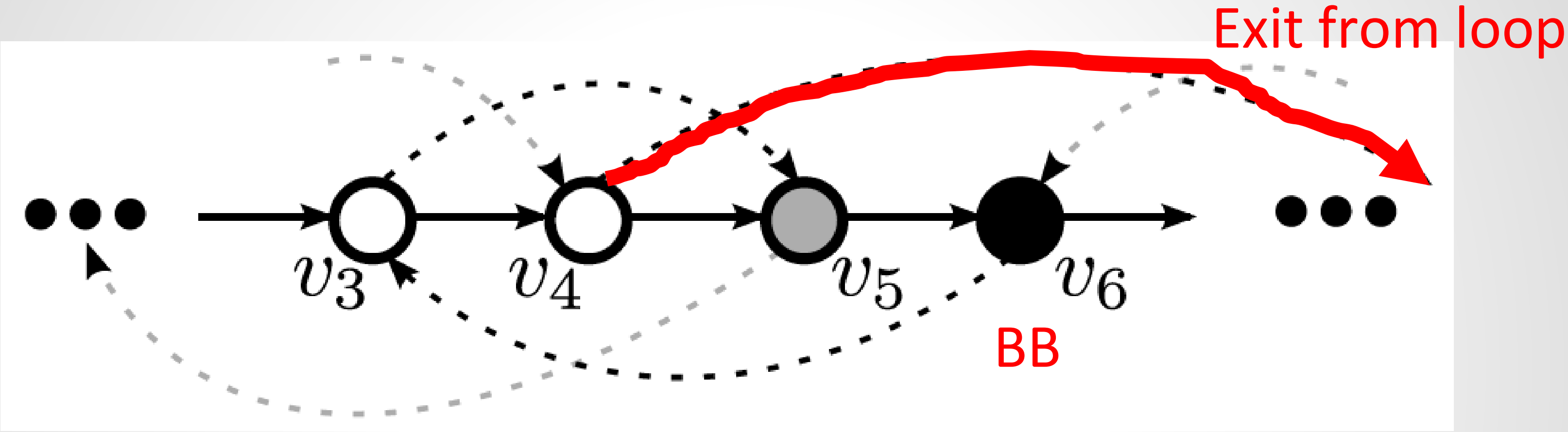
A hard decision problem: when to update FF?



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

Intuition Why 3 Rounds Are Hard

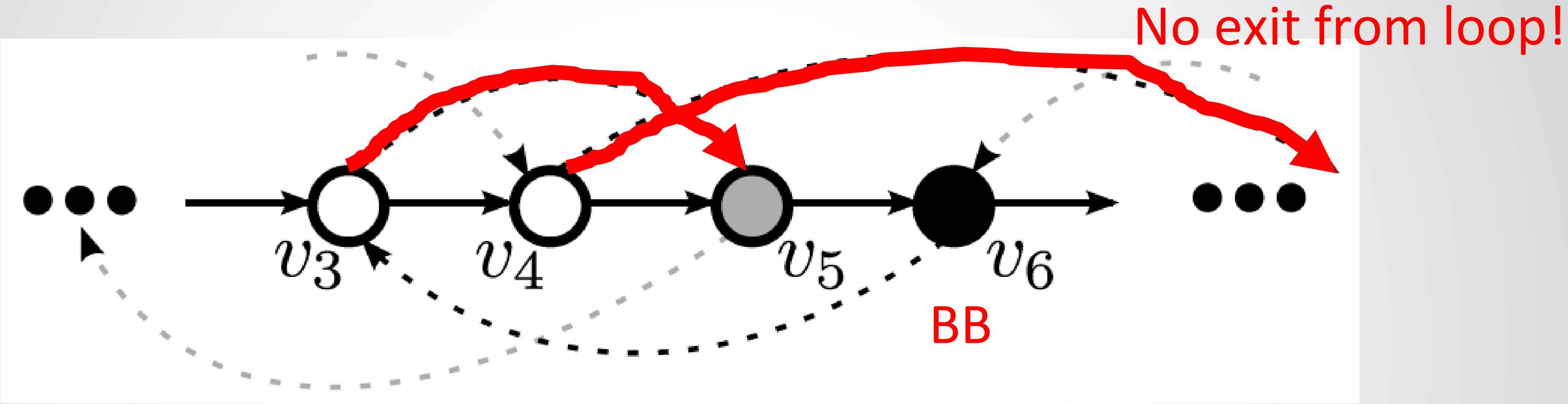
A hard decision problem: when to update FF?



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

Intuition Why 3 Rounds Are Hard

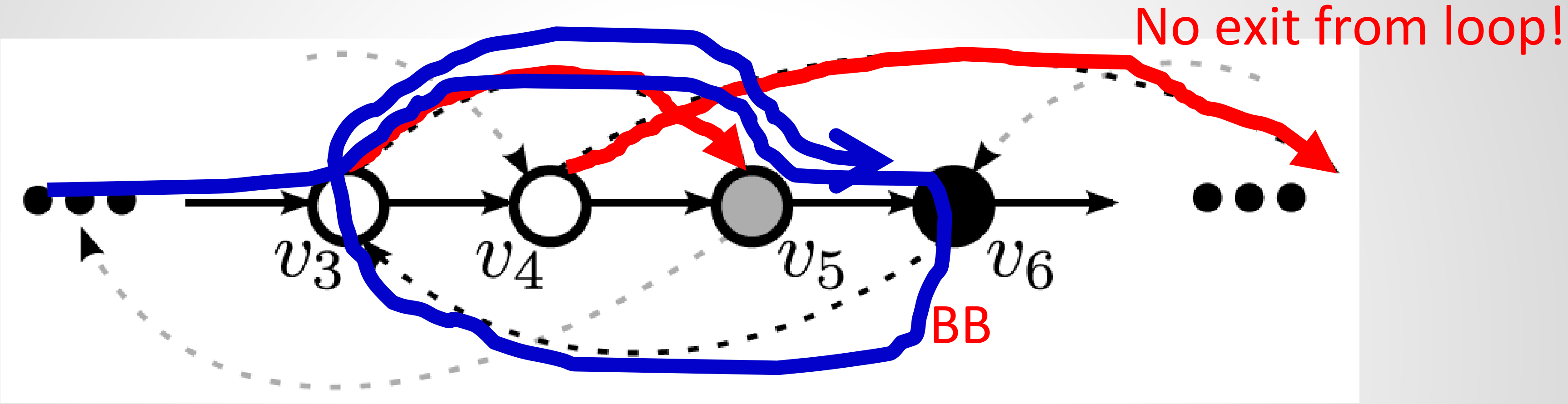
A hard decision problem: when to update FF?



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

Intuition Why 3 Rounds Are Hard

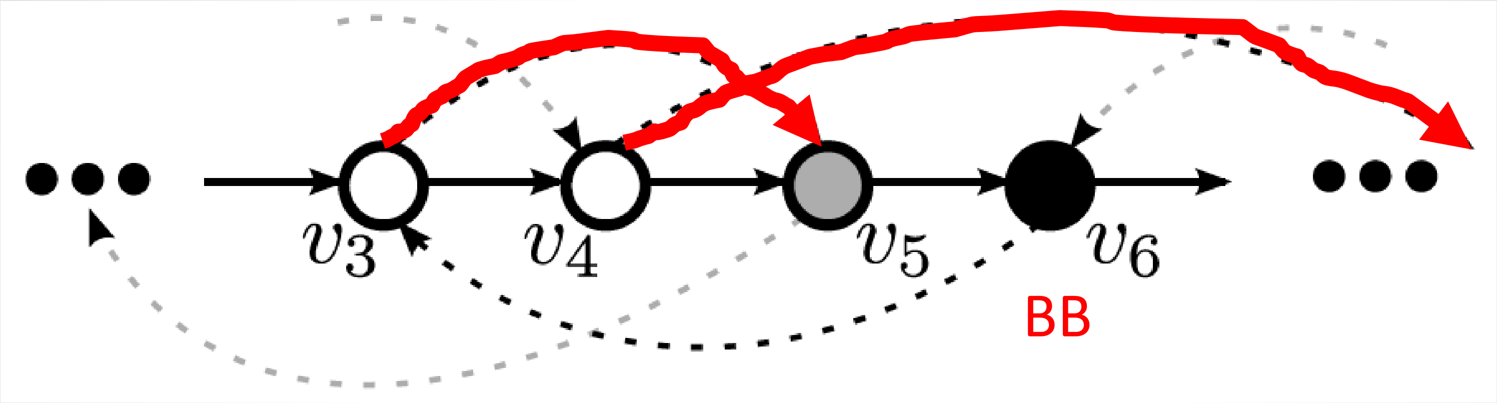
A hard decision problem: when to update FF?



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

Intuition Why 3 Rounds Are Hard

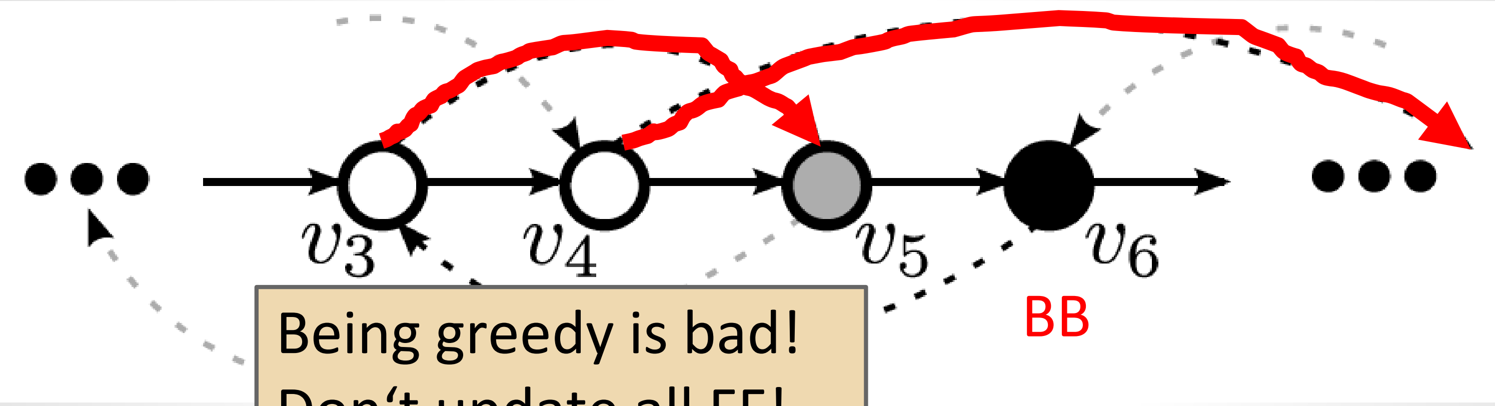
A hard decision problem: when to update FF?



- ❑ We know: BB node v_6 can only be updated in R2
- ❑ When to update FF nodes to make 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
- ❑ Smells like a gadget: which FF nodes to update when is hard!

Intuition Why 3 Rounds Are Hard

A hard decision problem: when to update FF?



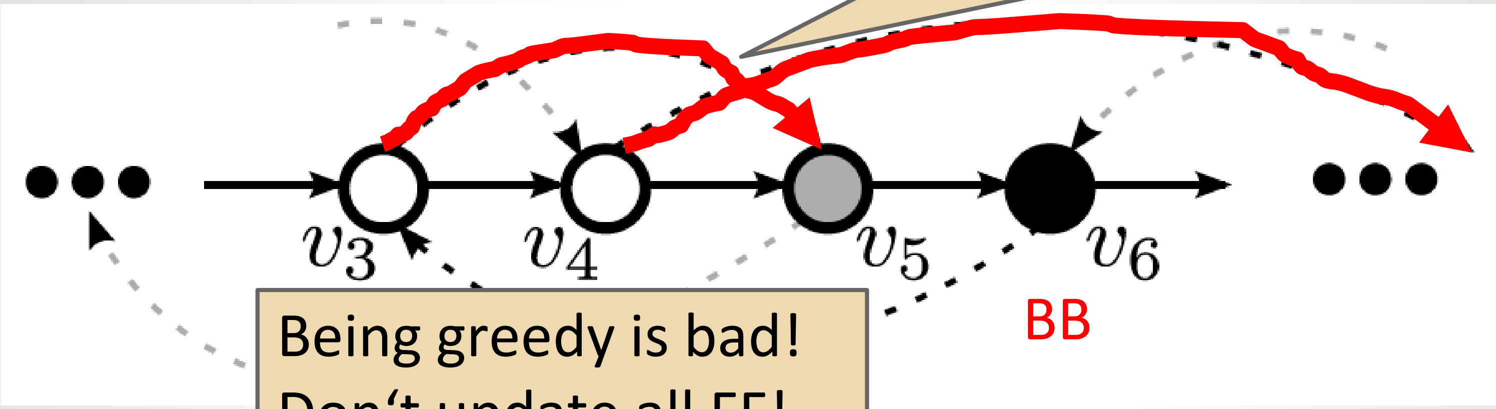
Being greedy is bad!
Don‘t update all FF!

- ❑ We know: BB node v_6 can only be updated in R2
- ❑ When to update FF nodes to make 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
- ❑ Smells like a gadget: which FF nodes to update when is hard!

Intuition Why 3

Devil lies in details: original paths must also be valid! I.e., to prove that such a configuration can be reached.

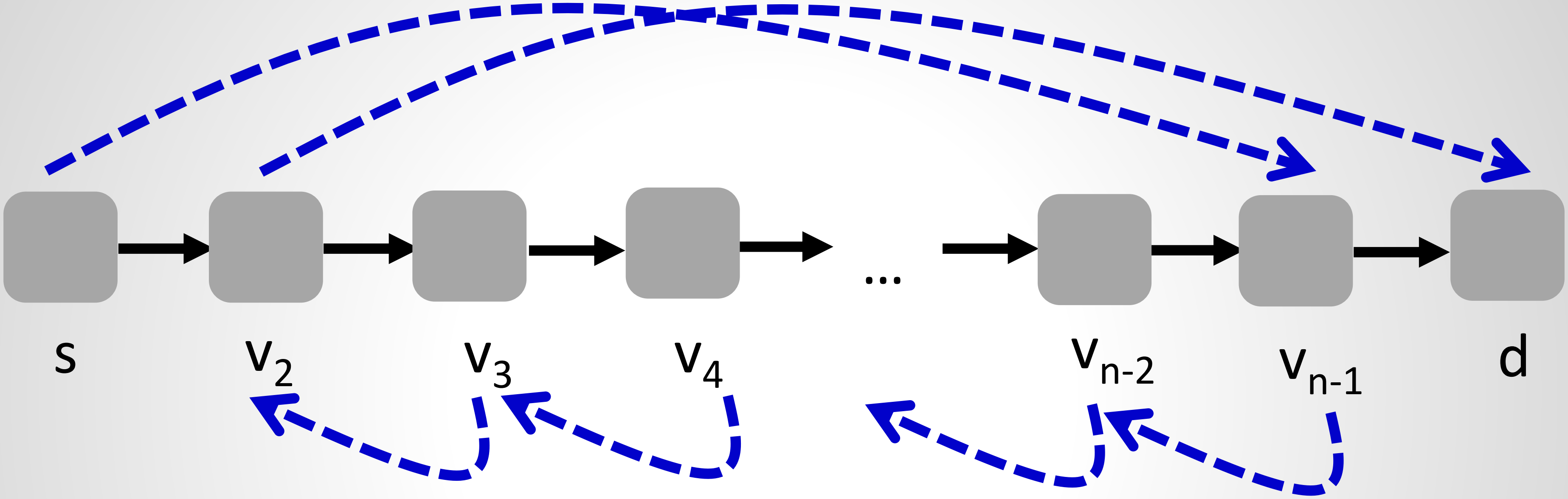
A hard decision problem:



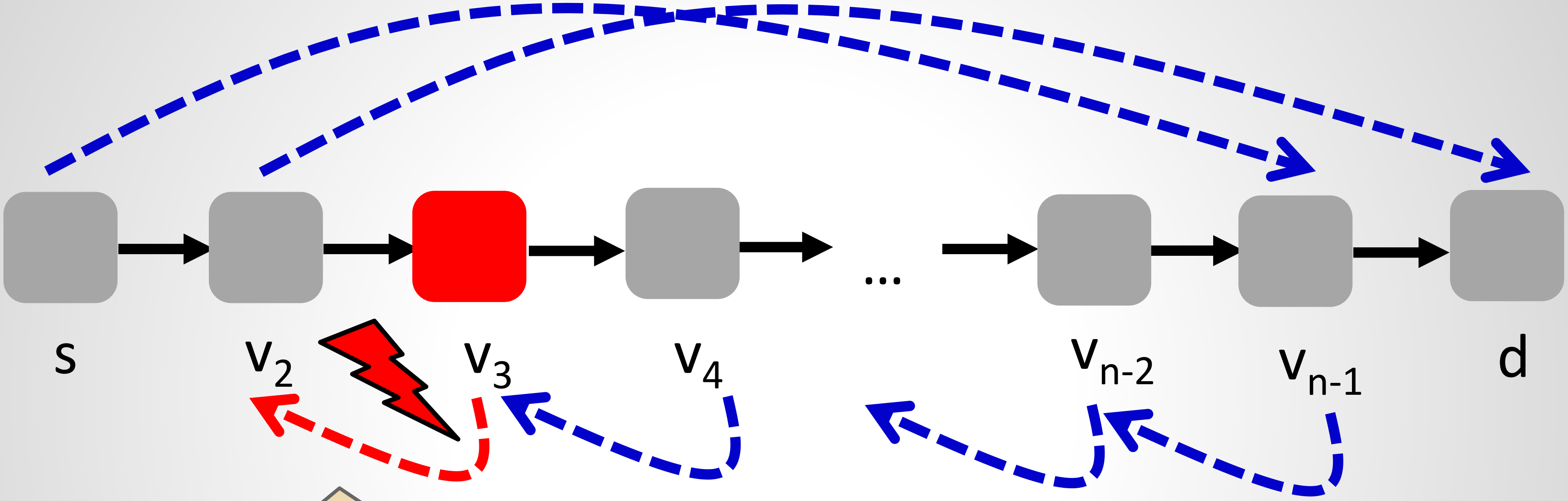
Being greedy is bad! Don't update all FF!

- ❑ We know: BB node v_6 can only be updated in R2
- ❑ When to update FF nodes to make 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
- ❑ Smells like a gadget: which FF nodes to update when is hard!

It‘s Good to Relax: How to update LF?

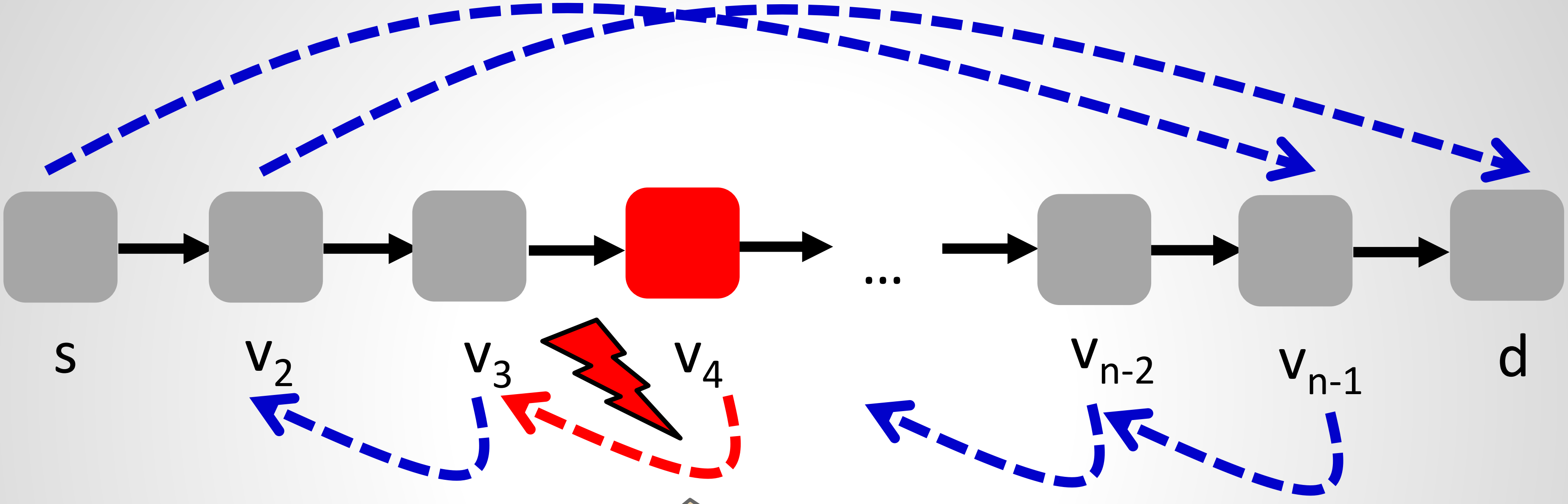


LF Updates Can Take Many Rounds!



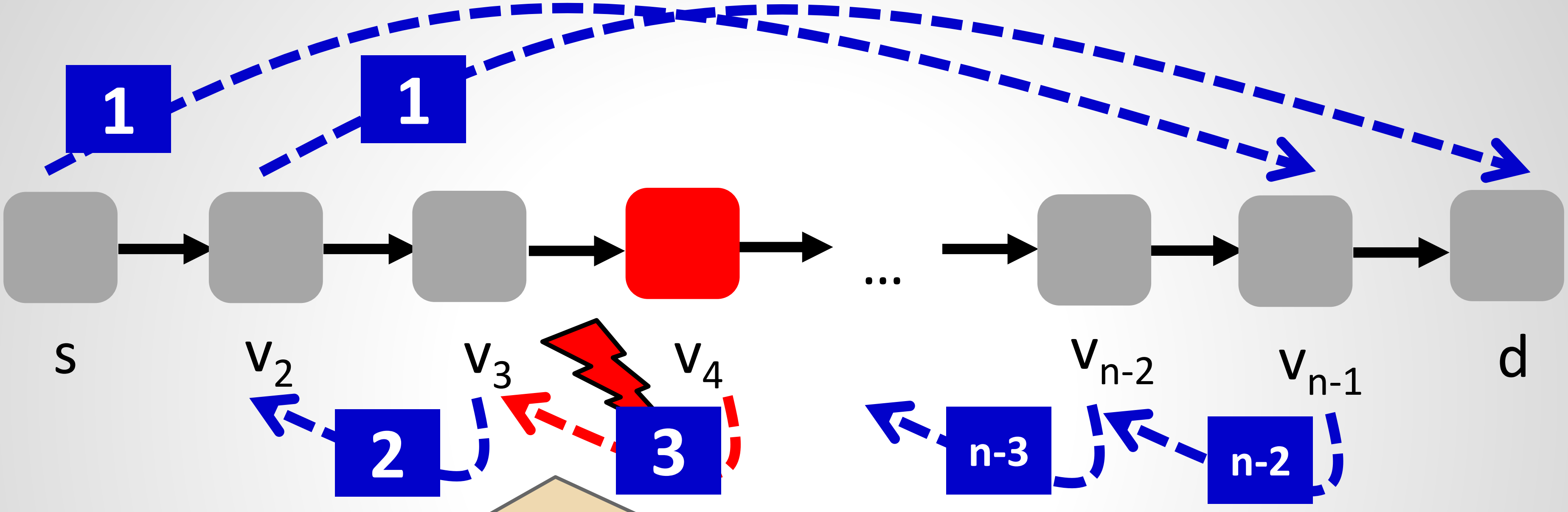
Invariant: need to update v_2 before v_3!

LF Updates Can Take Many Rounds!



Invariant: need to update v_3 before v_4!

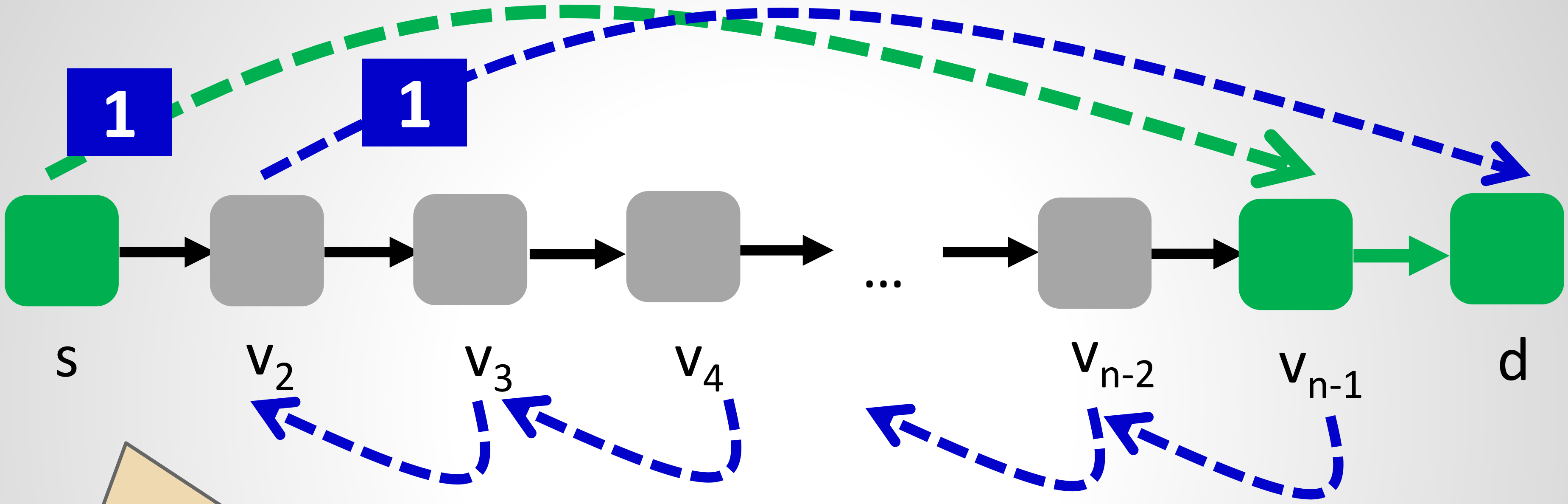
LF Updates Can Take Many Rounds!



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

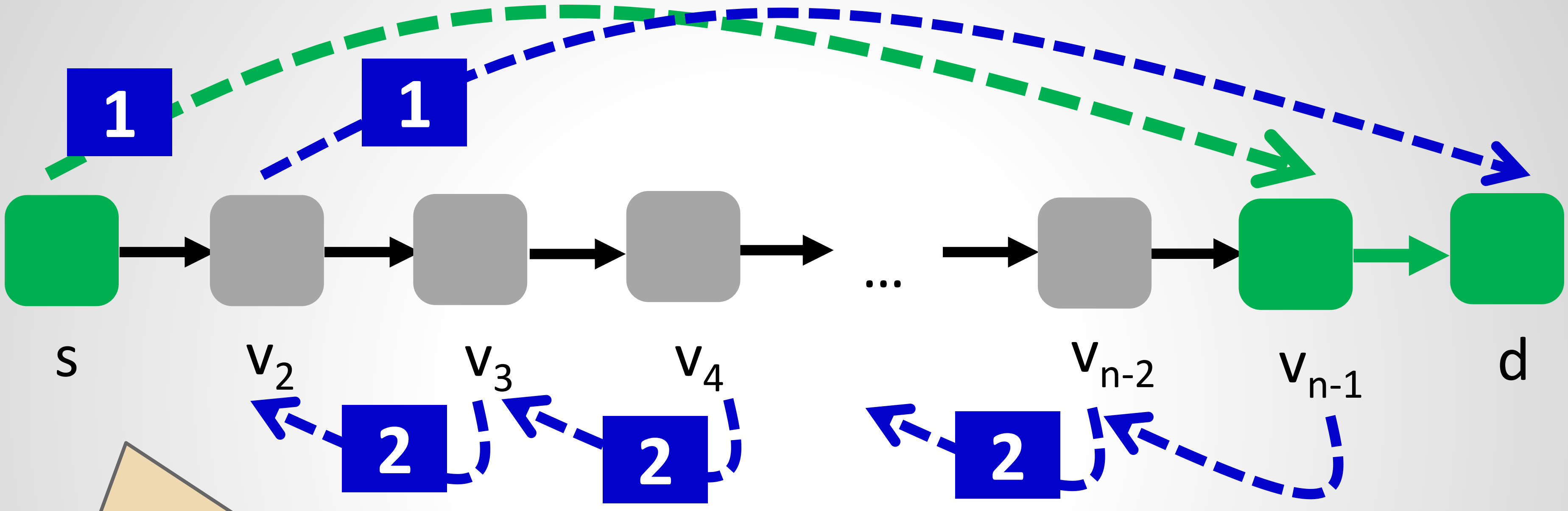
Ω(n) rounds?! In principle, yes…: Need a path back out before updating backward edge!

It is good to relax!



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

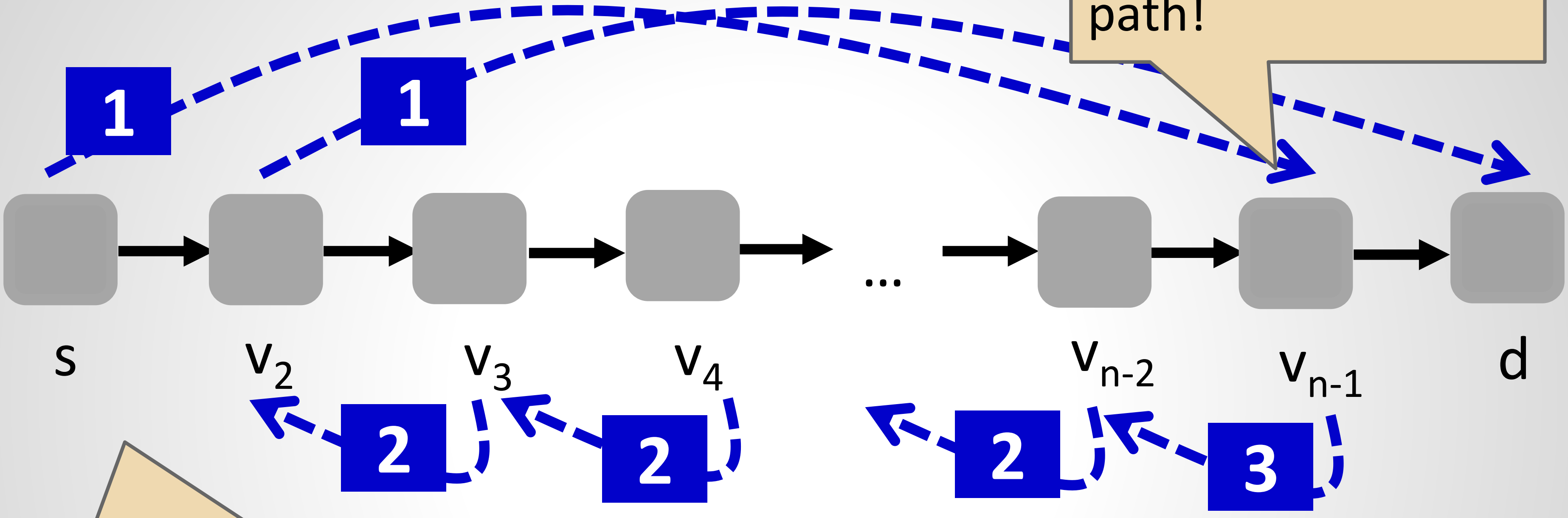
It is good to relax!



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

Could be updated simultaneously!

It is good to relax!

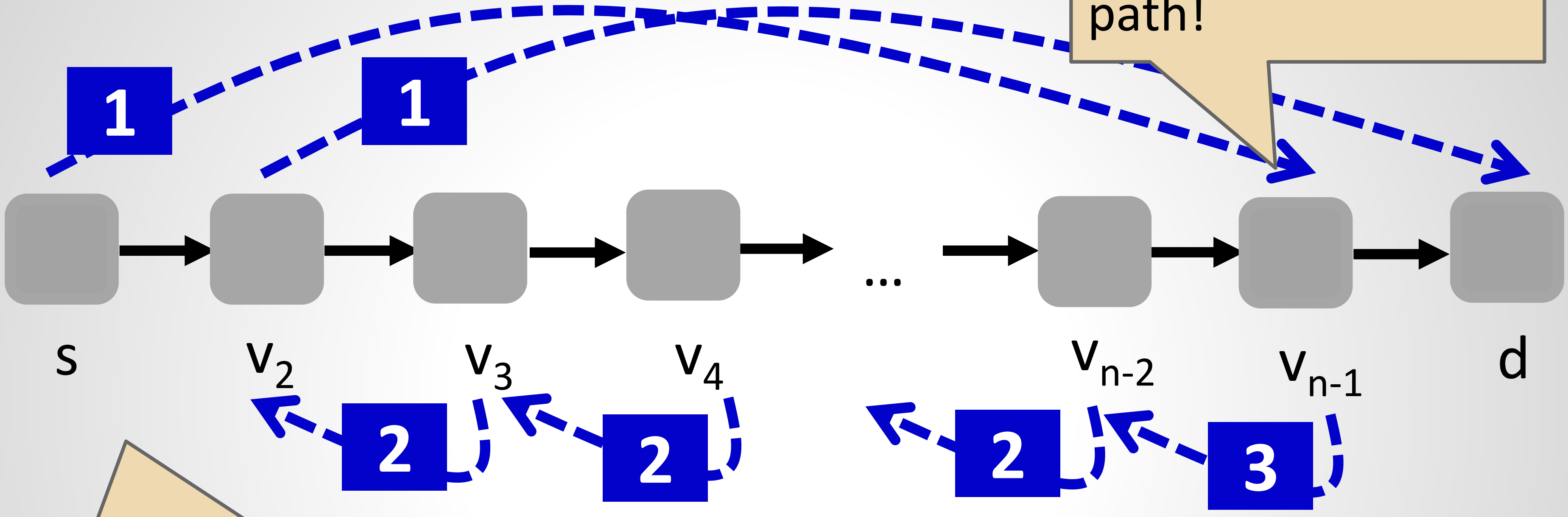


Finally put back on path!

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

Could be updated simultaneously!

It is good to relax!



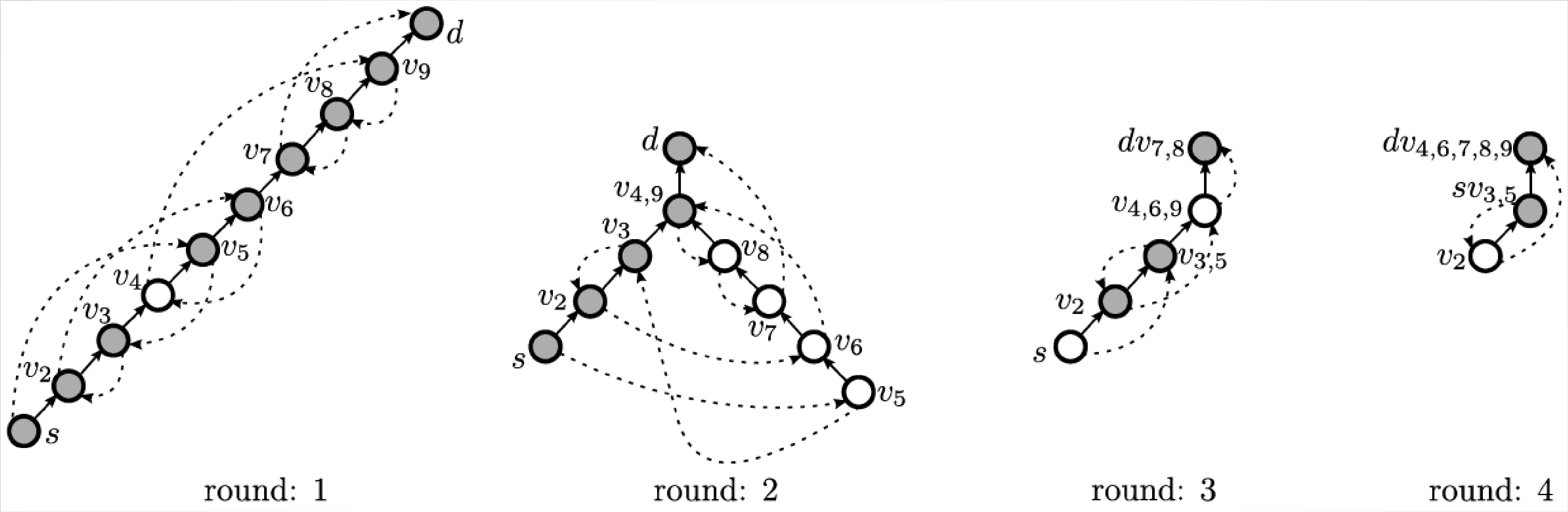
Finally put back on path!

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

Could be updated simultaneously!

3 rounds only!

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



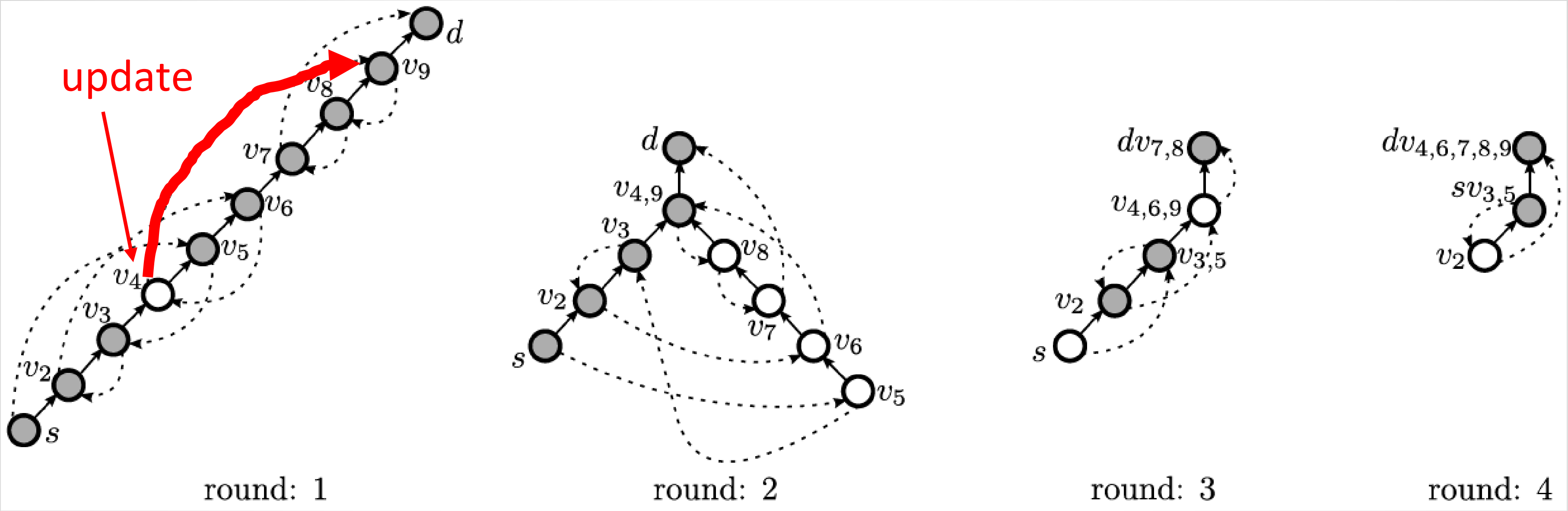
Shortcut

Prune

Shortcut

Prune

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



Shortcut

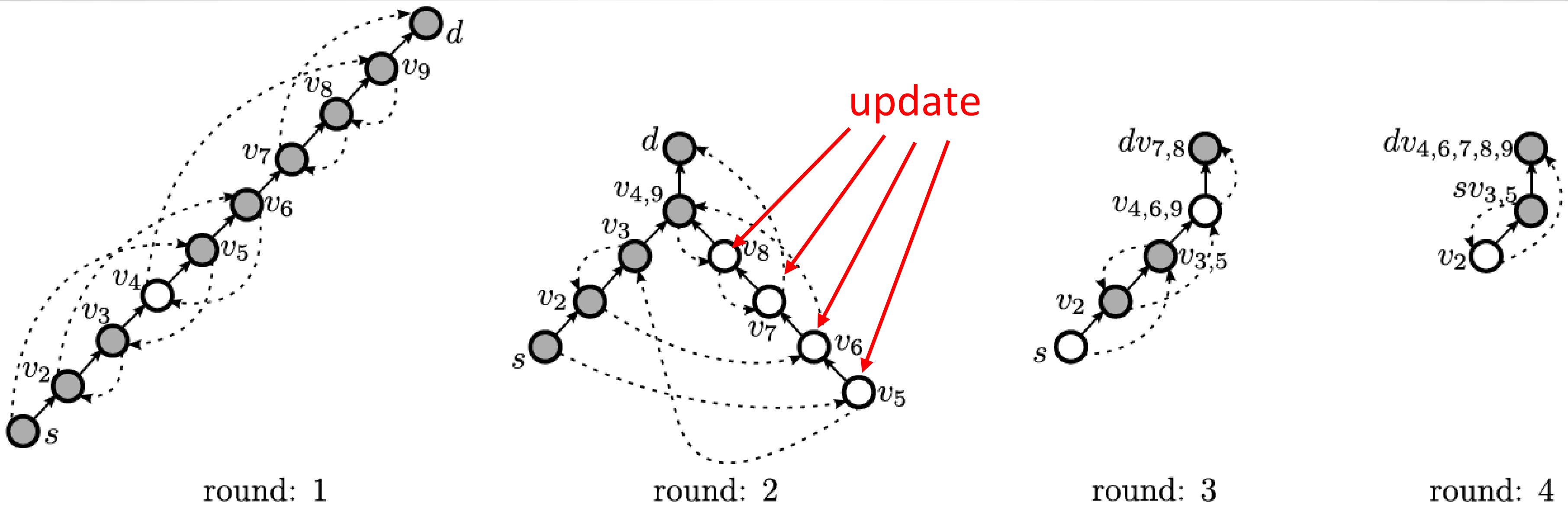
Prune

Shortcut

Prune

Greedily choose far-reaching (independent) forward edges.

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



Shortcut

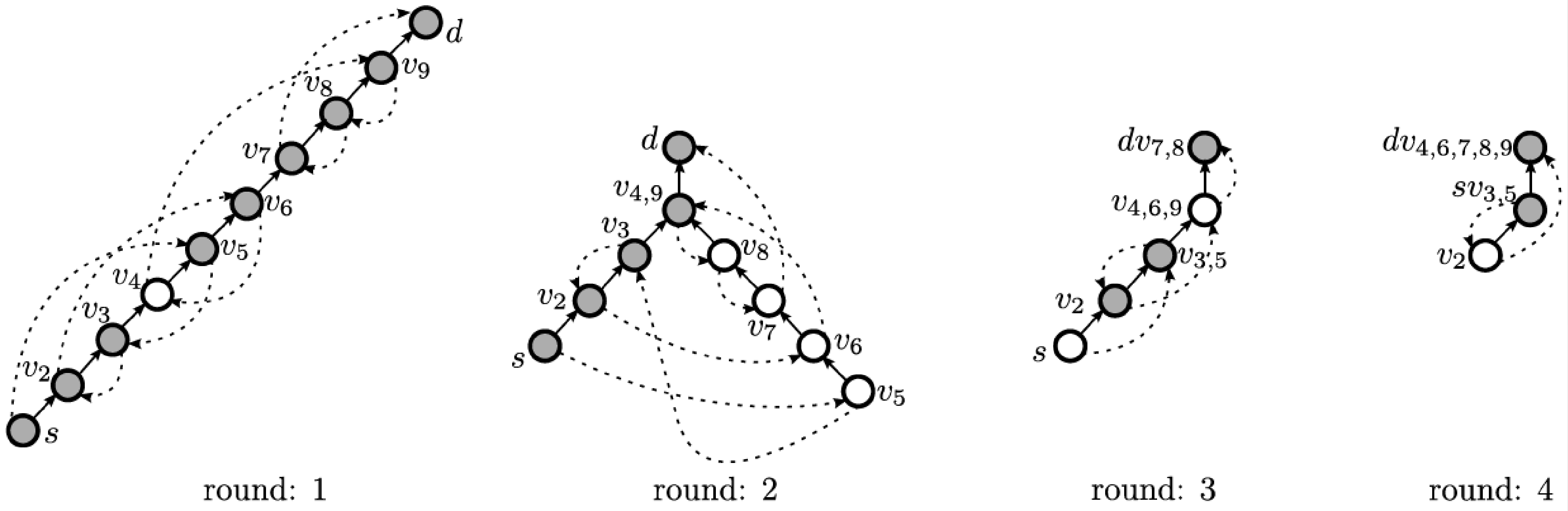
Prune

Shortcut

Prune

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

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



Shortcut

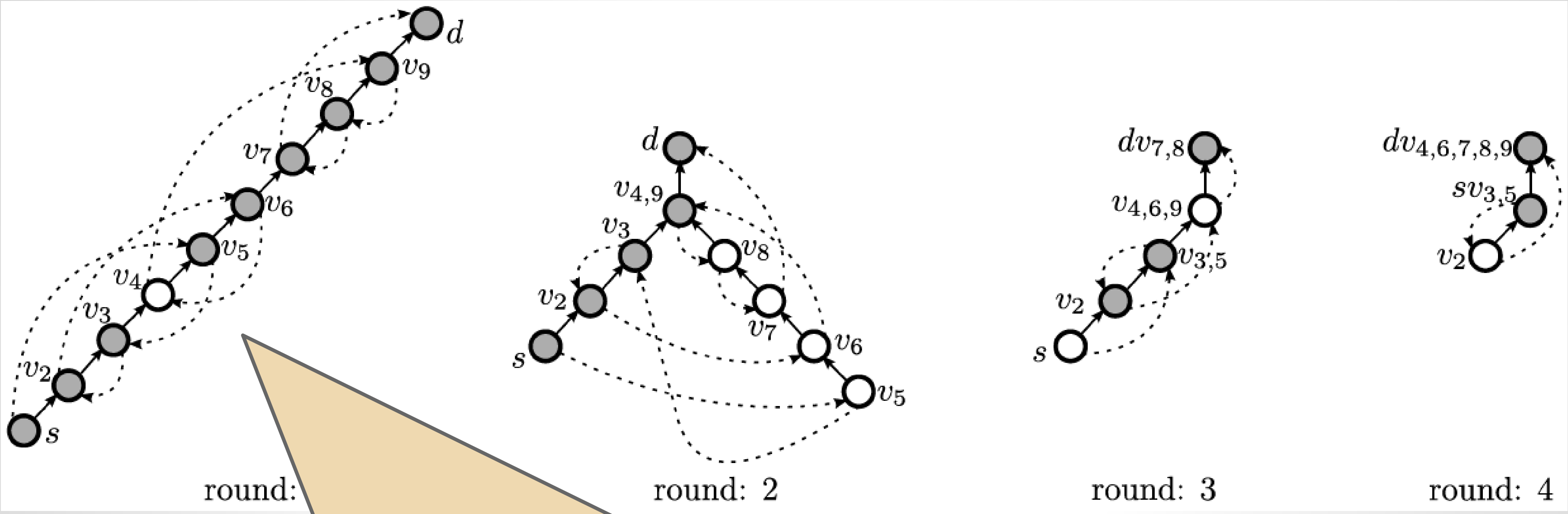
Prune

Shortcut

Prune

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

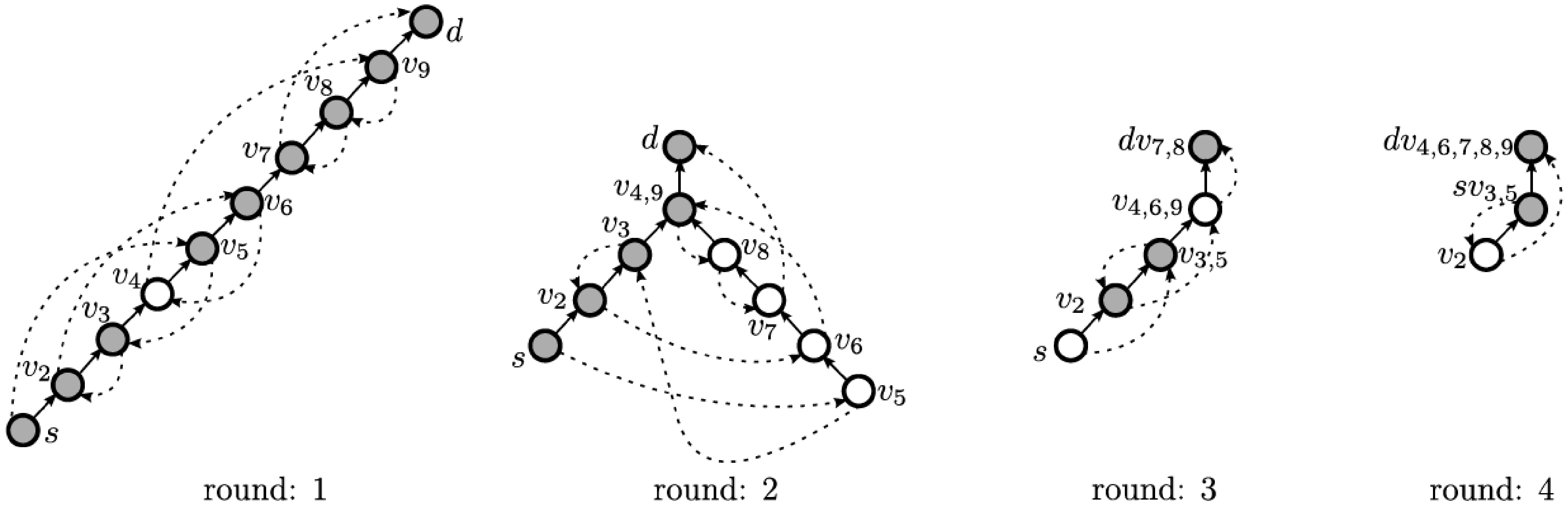
A log(n)-time Algorithm: *Peacock* in Action



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

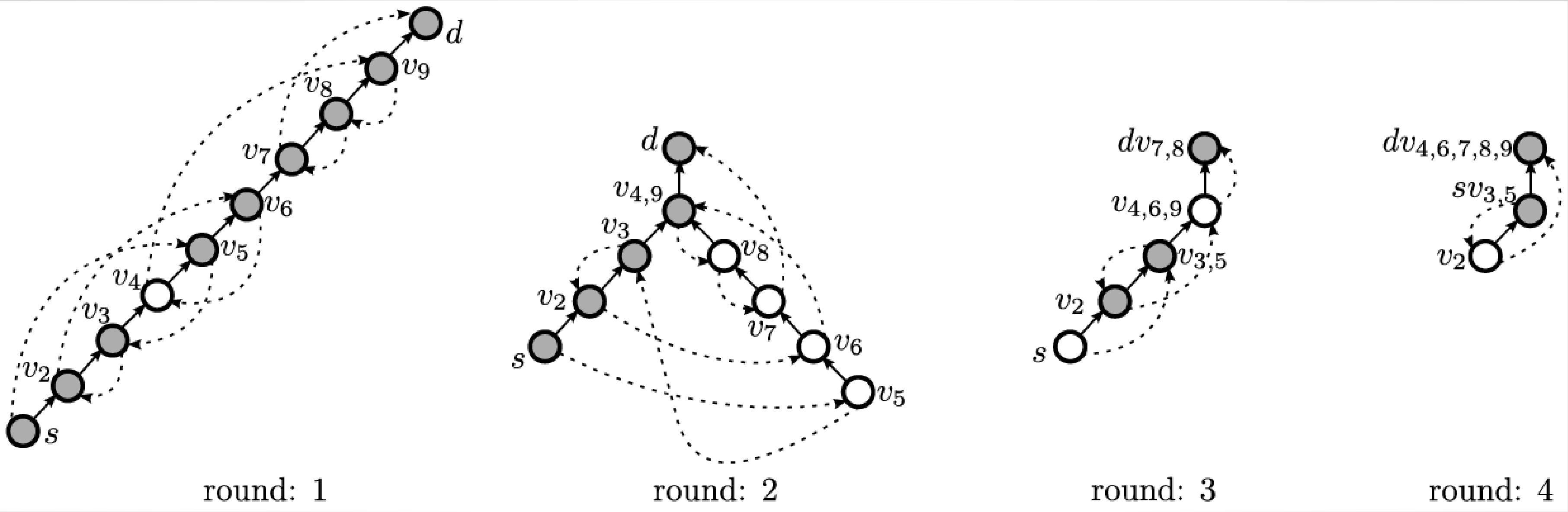
Prune

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

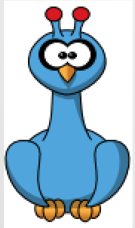


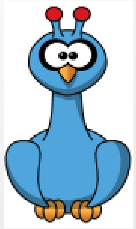
Shortcut

Prune

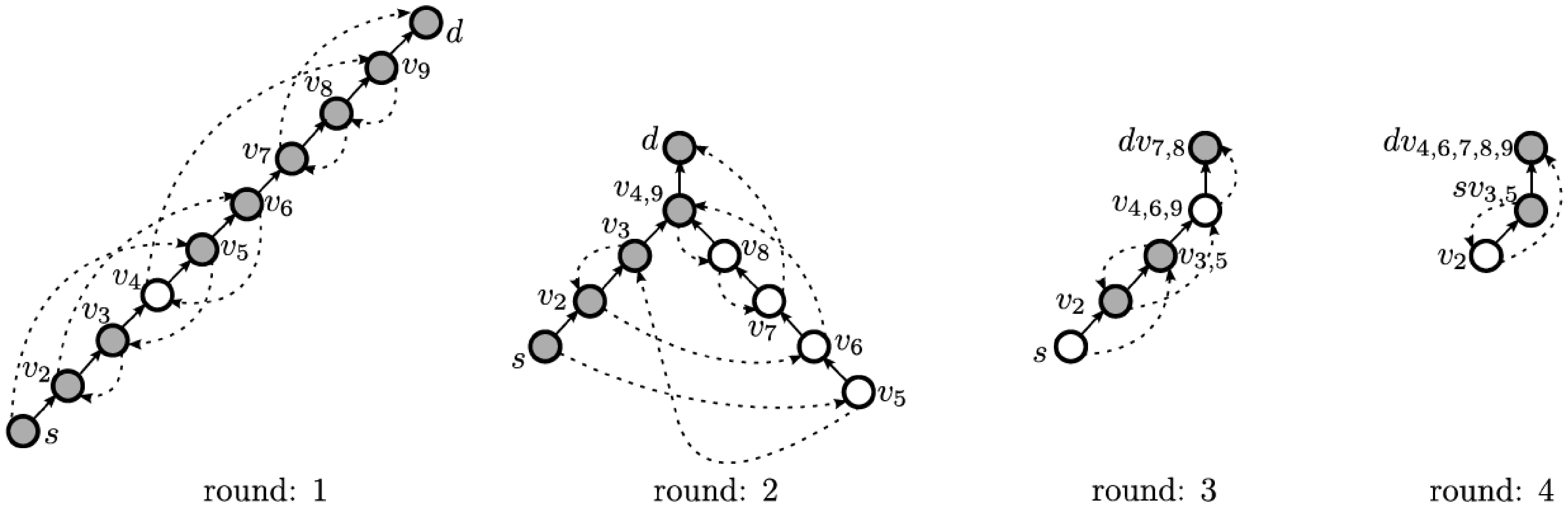
Shortcut

Prune





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

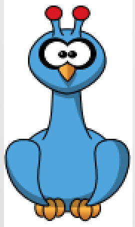


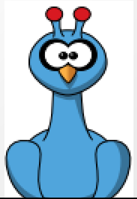
Shortcut

Prune

Shortcut

Prune

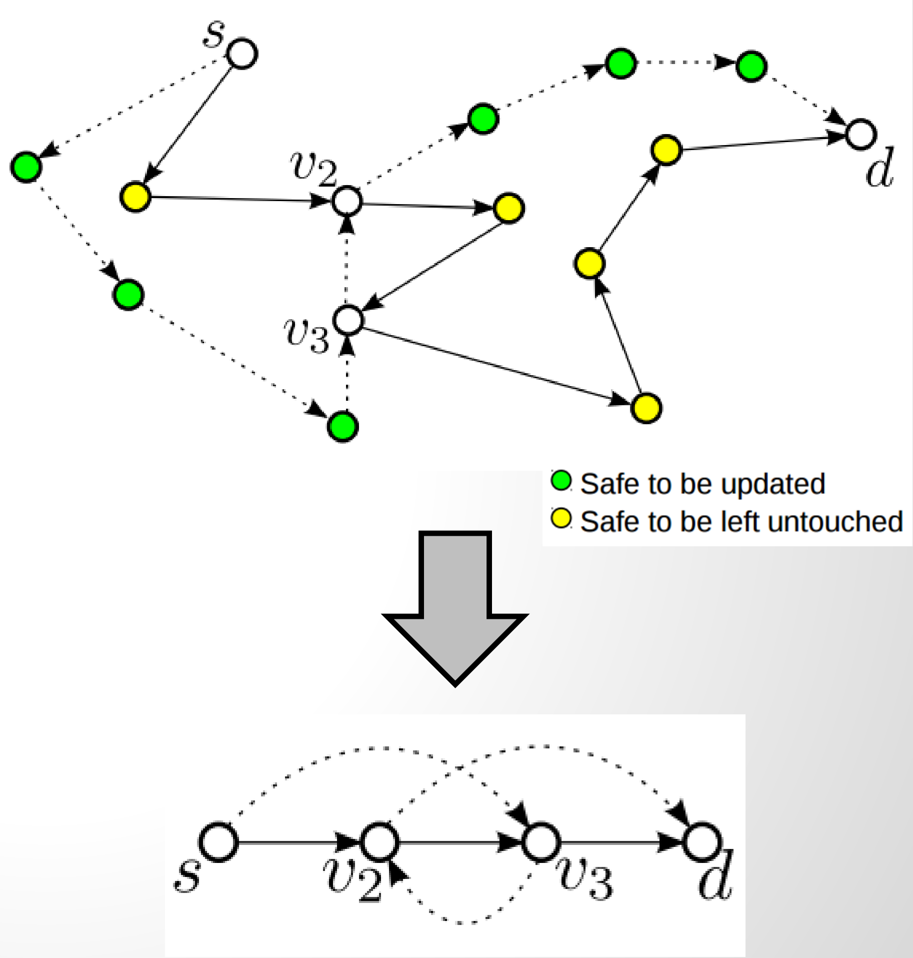




Scheduling Loop-free Network Updates: It's Good to Relax!
Arne Ludwig, Jan Marcinkowski, and Stefan Schmid.
ACM Symposium on Principles of Distributed Computing (**PODC**),
Donostia-San Sebastian, Spain, July 2015.

Remark on the Model

Easy to update new nodes which do not appear in old policy. And just keep nodes which are not on new path!



Loop-Freedom: Summary of Results

- ❑ 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 Ω(n) although O(1) possible
- ❑ Multiple policies: aggregate updates to given switch!
  - ❑ Related to Shortest Common Supersequence Problem

Loop-Freedom: Summary of Results

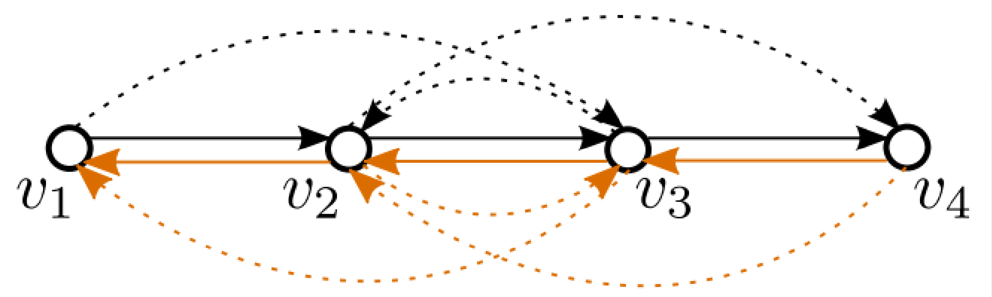
- Minimizing the **number of rounds**
  - For 2-round instances: polynomial time
  - For 3-round instances: NP-hard, no approximation known
- Relaxed notion of loo nds
  - No approximation kno

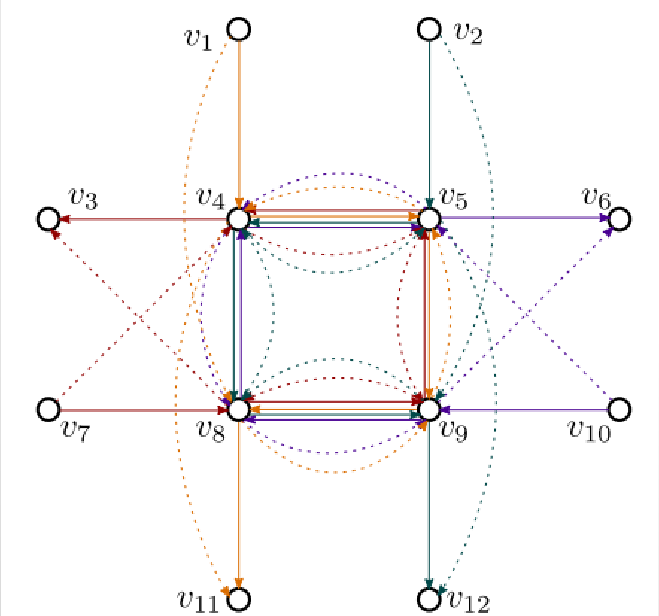
Being greedy is bad!
And hard ☺

- 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 Ω(n) although O(1) possible
- Multiple policies: aggregate updates to given switch!
  - Related to Shortest Common Supersequence Problem

Extension: Multiple Policies

At least one node needs to be **touched** twice: otherwise at least one flow will have a temporary loop:

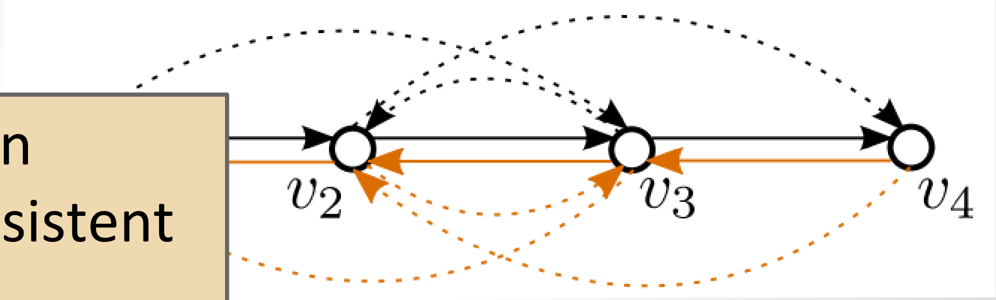




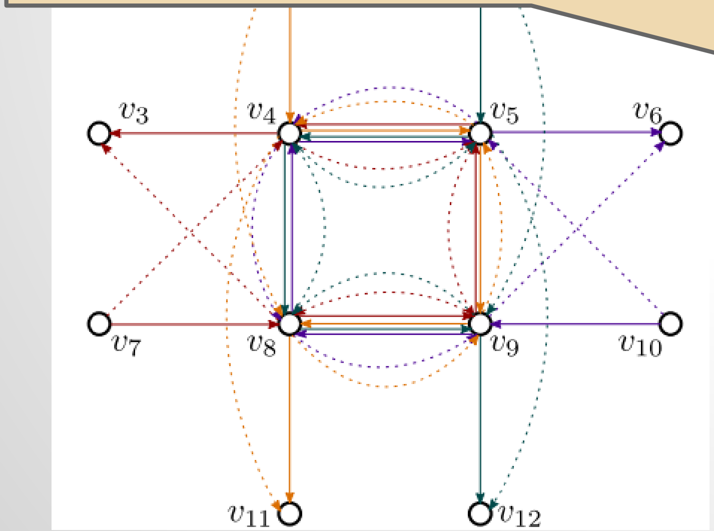
Worst case: k policies require k touches!

Extension: Multiple Policies

At least one node needs to be **touched** twice: oth flov tem



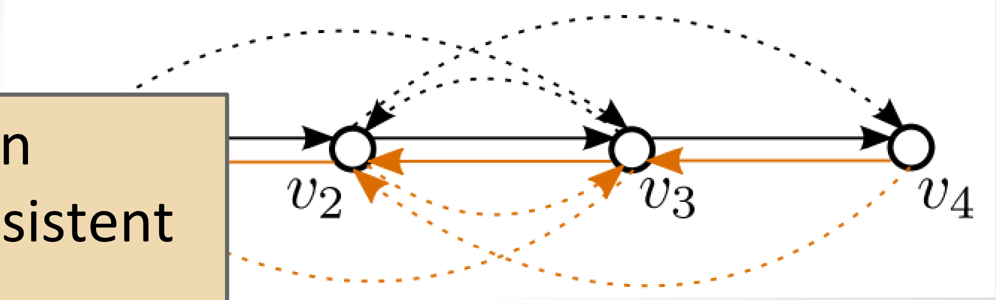
On the positive side: given individual transiently consistent schedules, can optimally combine them using dynamic programming! Independently of the consistency property.



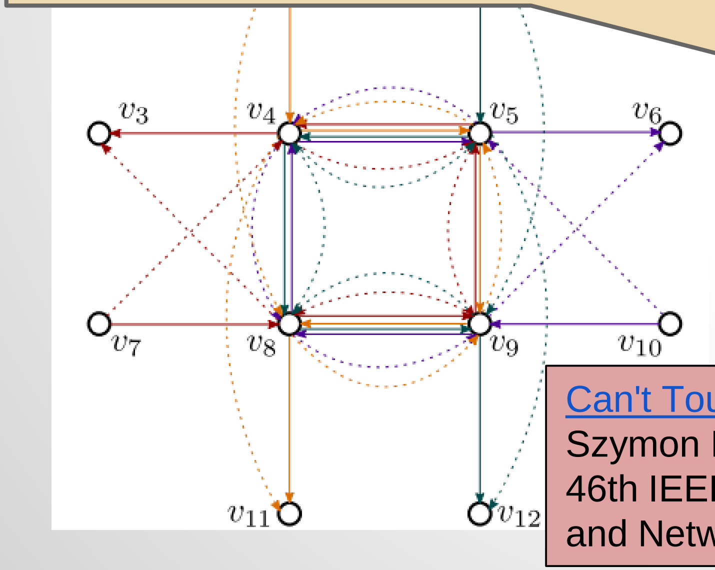
Worst case: k policies require k touches!

Extension: Multiple Policies

At least one node needs
to be **touched** twice:
oth
flow
tem



On the positive side: given individual transiently consistent schedules, can optimally combine them using dynamic programming! Independently of the consistency property.



Worst case: k policies require k touches!

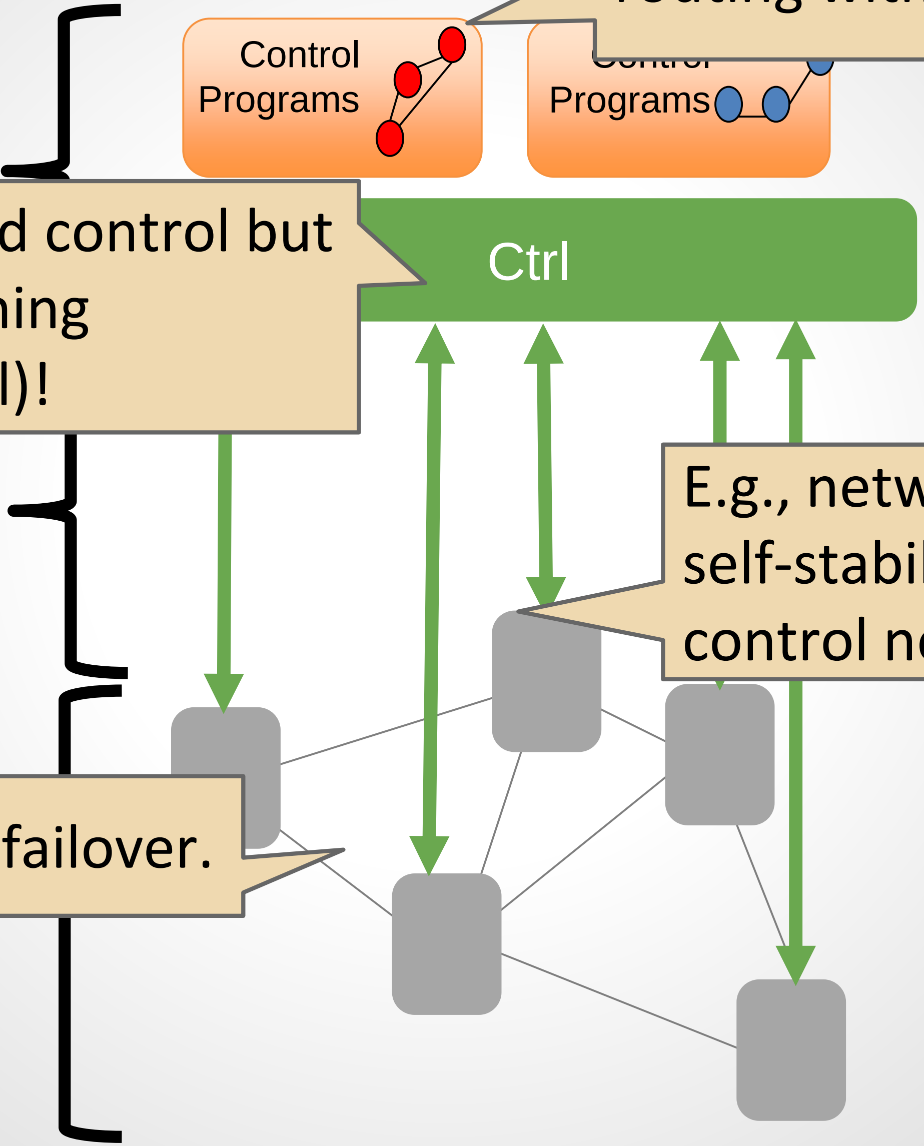
Can't Touch This: Consistent Network Updates for Multiple Policies
Szymon Dudycz, Arne Ludwig, and Stefan Schmid.
46th IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**), Toulouse, France, June 2016.

Conclusic

Applications and C

... and regarding inter-connect!

Da



E.g., admission control and routing with waypoints.

E.g., distributed control but also MAC learning (Jen@Dagstuhl)!

E.g., network updates or self-stabilizing in-band control network.

E.g., robust failover.

Own References

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.

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.

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.

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.

Provable Data Plane Connectivity with Local Fast Failover: Introducing OpenFlow Graph Algorithms
Michael Borokhovich, Liron Schiff, and Stefan Schmid.
ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (**HotSDN**), Chicago, Illinois, USA, August 2014.