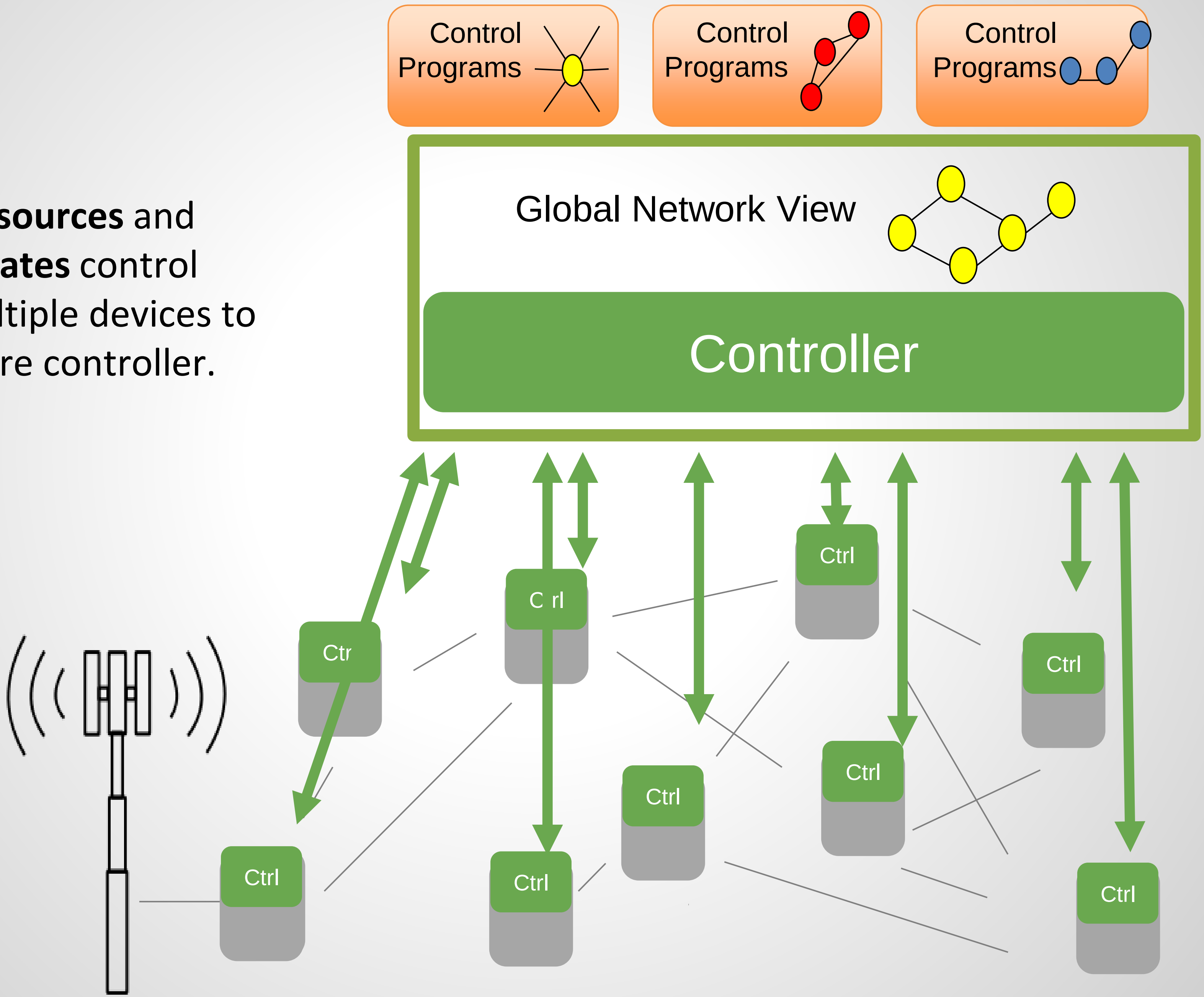
The Zen of Consistent Distributed Network Updates

Stefan Schmid

TU Berlin & Telekom Innovation Labs (T-Labs)

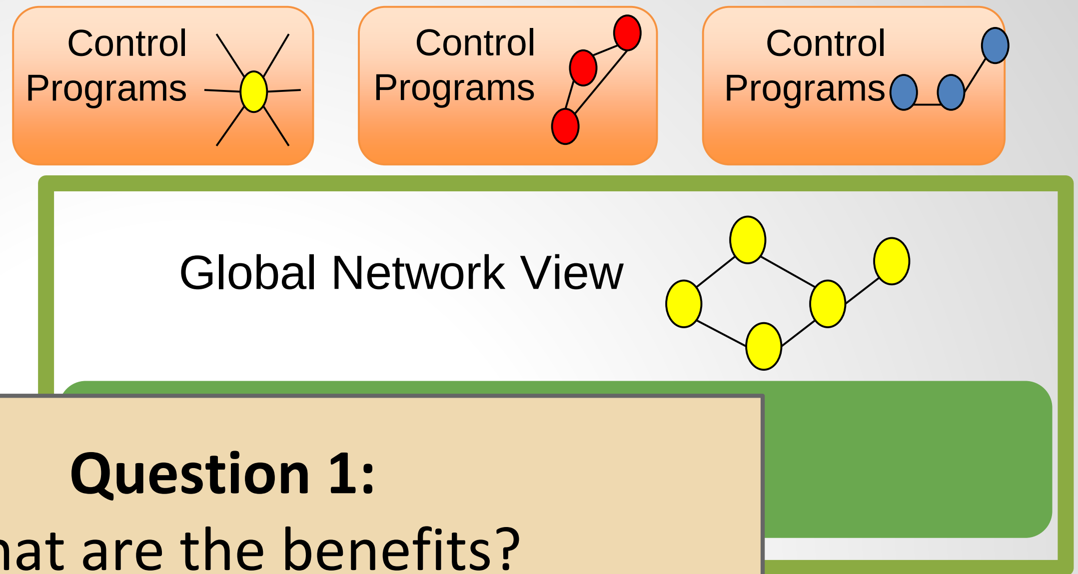
Just a little bit of background: SDN in a Nutshell

SDN **outsources** and **consolidates** control over multiple devices to a software controller.

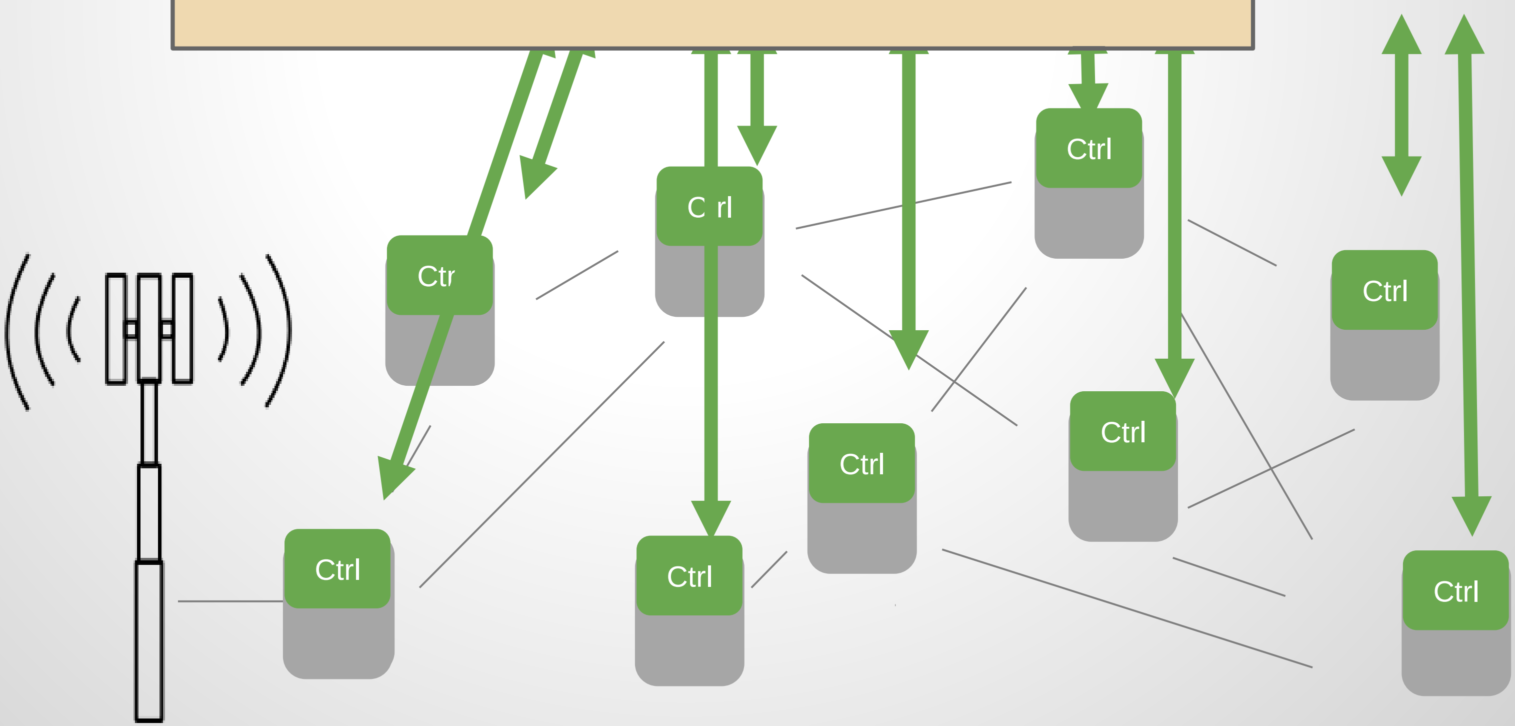


Just a little bit of background: SDN in a Nutshell

SDN **outsources** and **consolidates** control over multiple de
a software contr



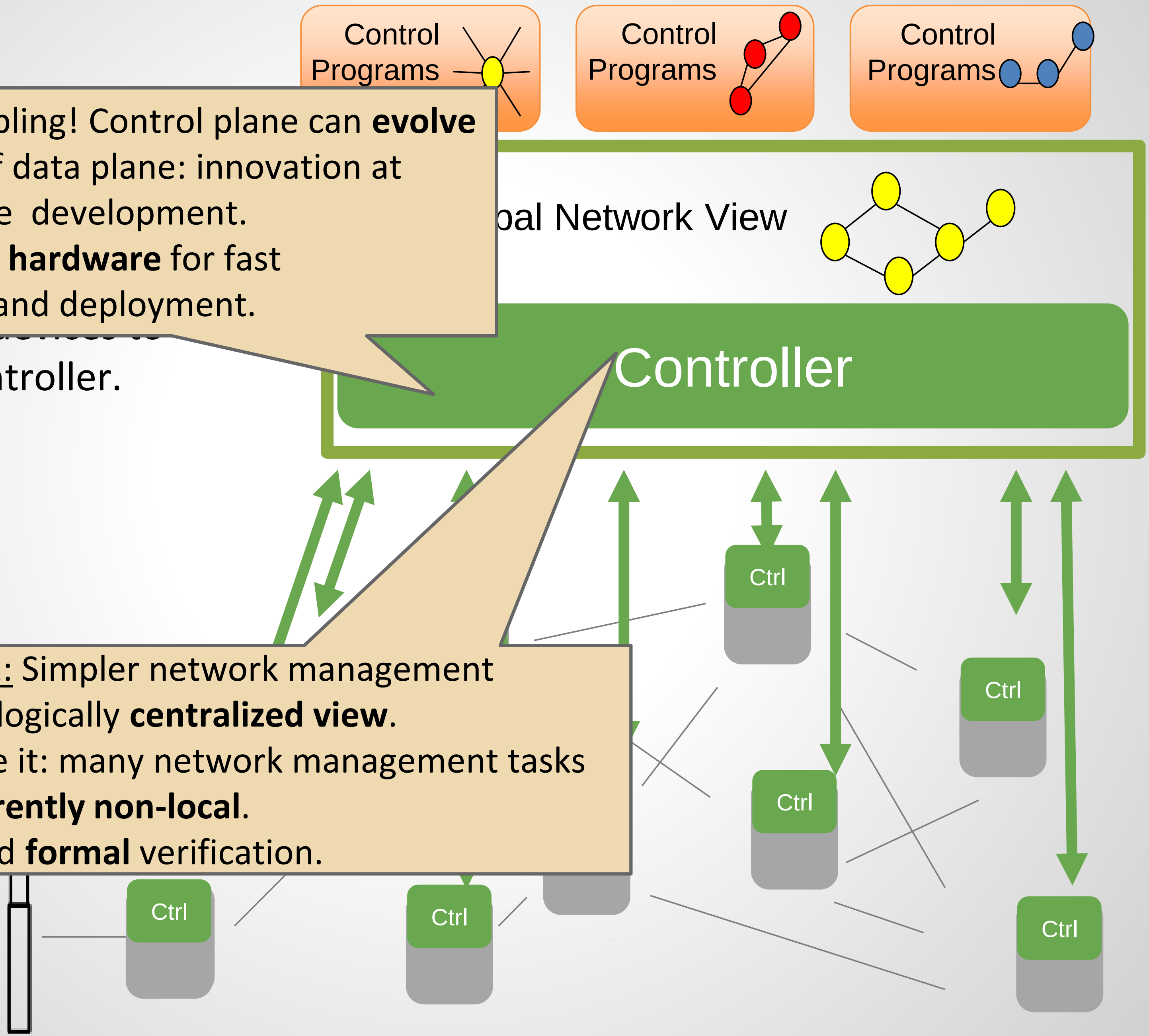
Question 1:
What are the benefits?



SDN in a Nutshell

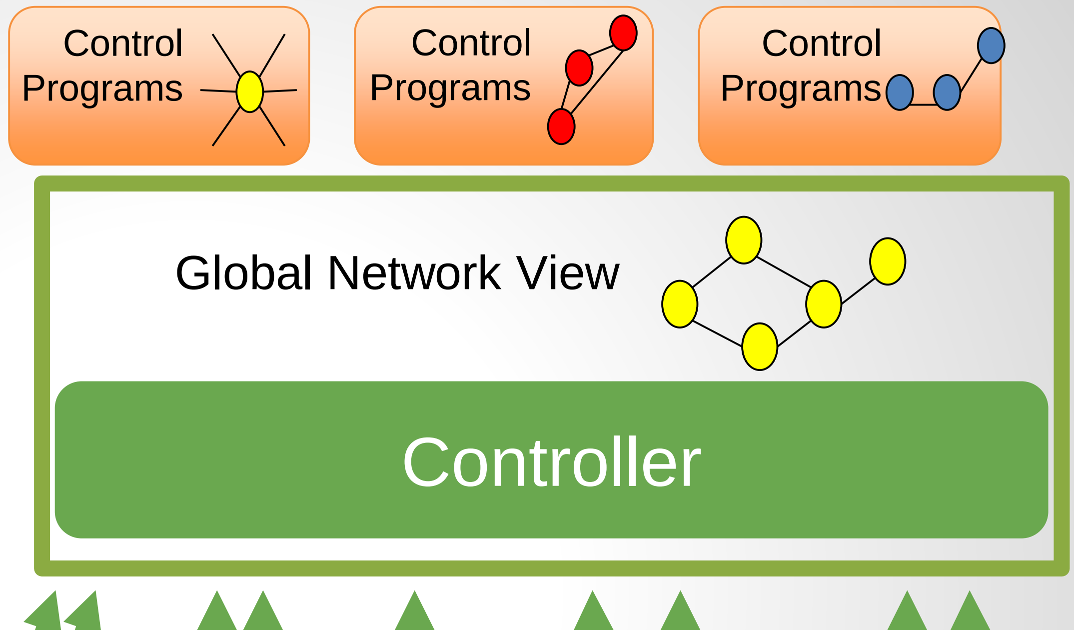
Benefit 1: Decoupling! Control plane can **evolve independently** of data plane: innovation at speed of software development.
Software trumps hardware for fast implementation and deployment.

a software controller.



Benefit 2: Simpler network management through logically **centralized view**.
Let's face it: many network management tasks are **inherently non-local**.
Simplified **formal** verification.

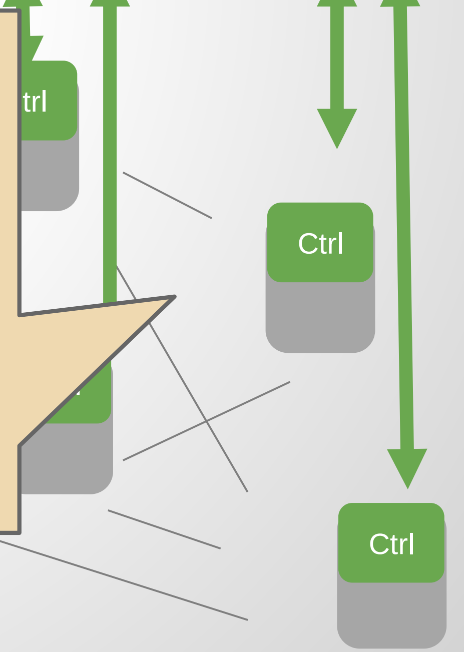
SDN in a Nutshell



SDN **outsources** and **consolidates** control over multiple devices to a software controller.

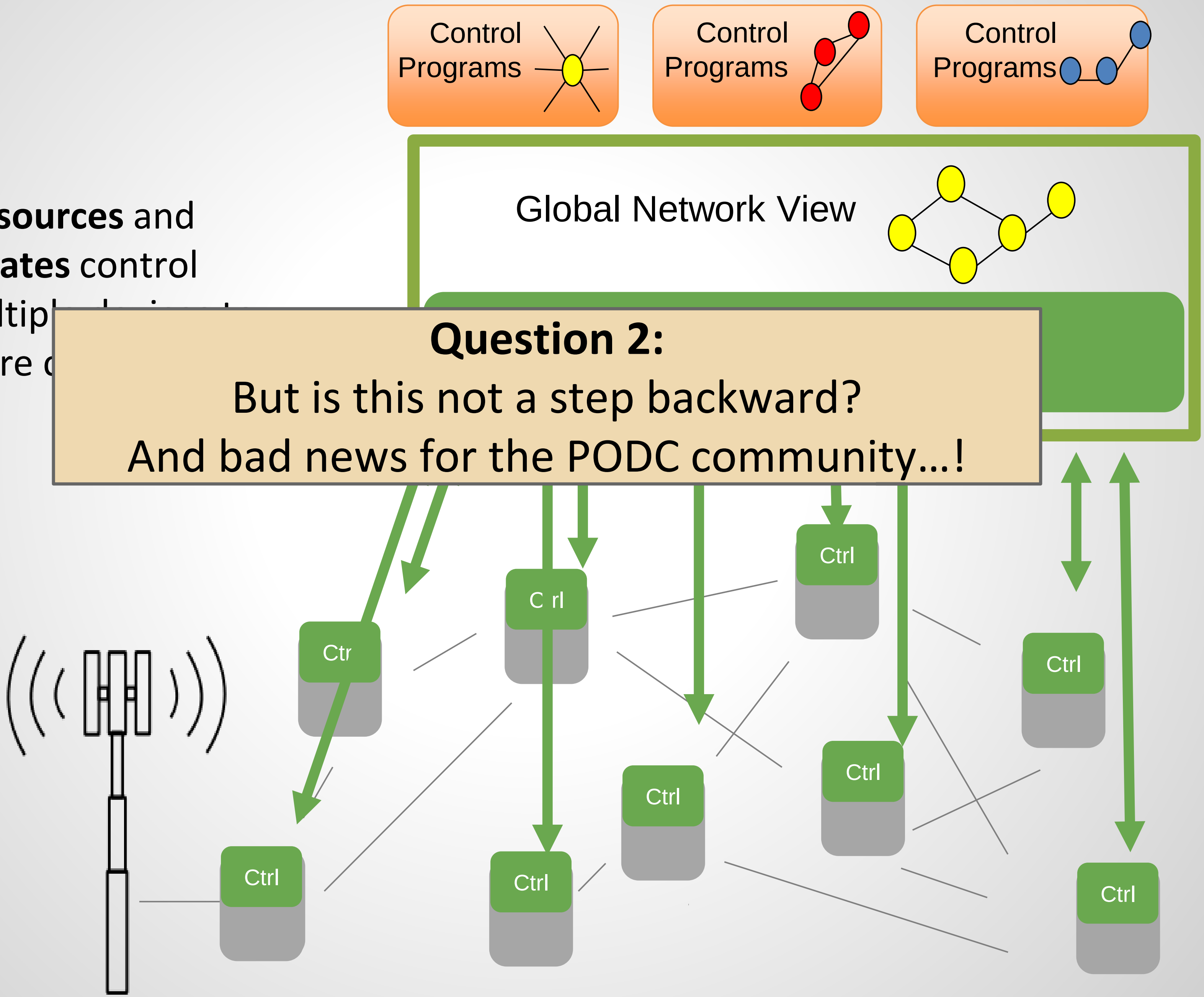
<u>Benefit 3:</u> Standard API OpenFlow is about **generalization**!

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



SDN in a Nutshell

SDN **outsources** and **consolidates** control over multipl... a software c...



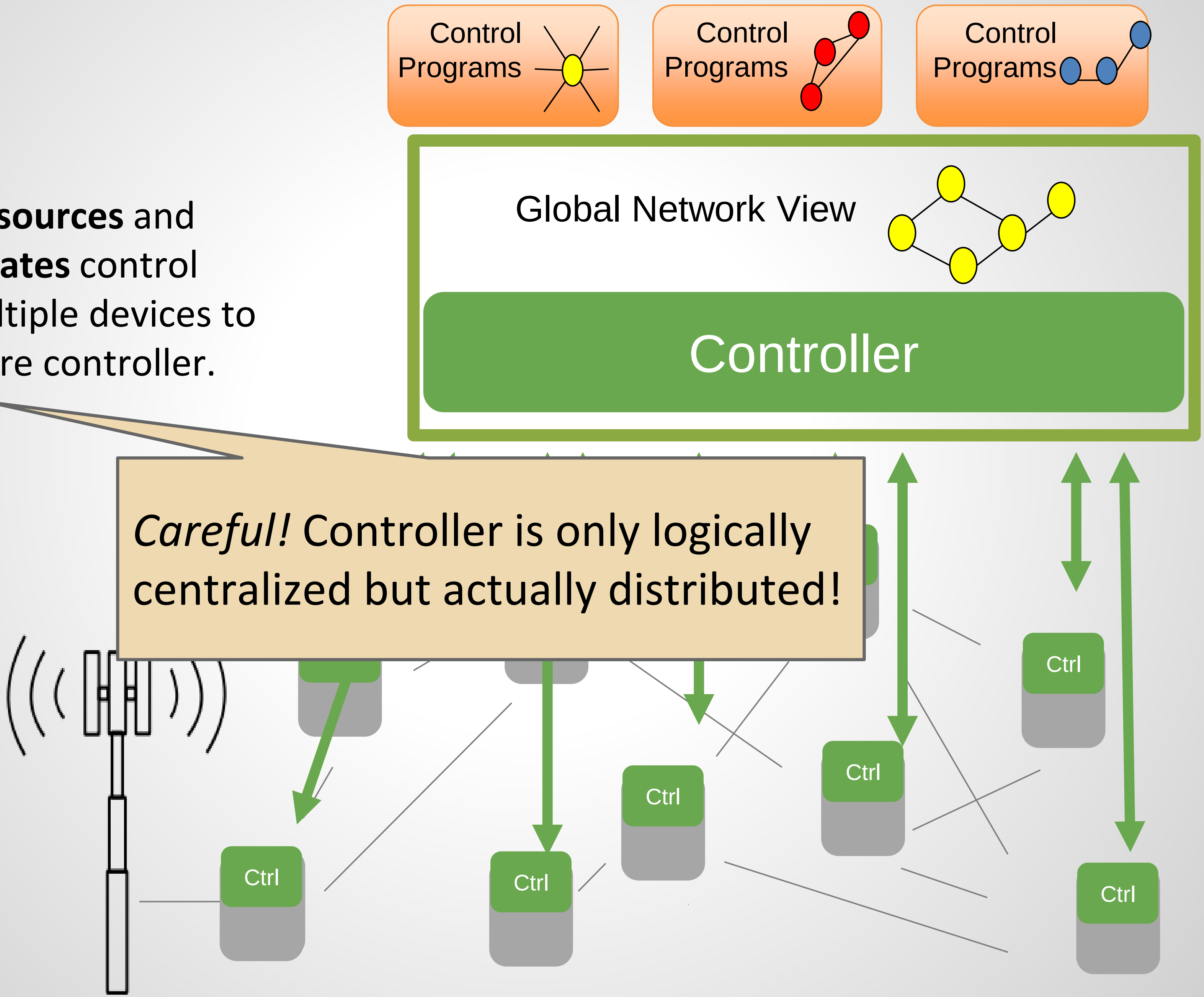
Question 2:

But is this not a step backward?

And bad news for the PODC community…!

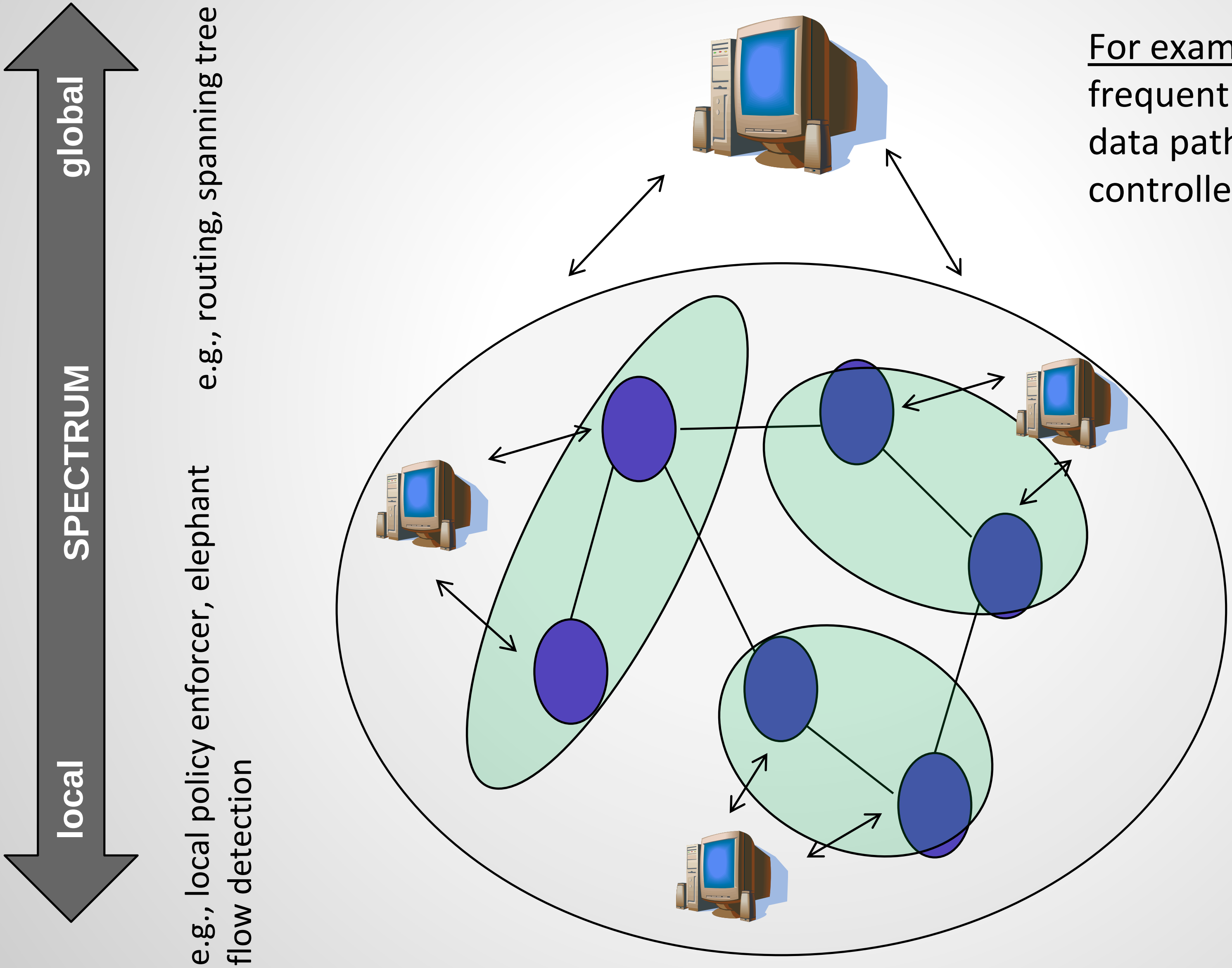
SDN in a Nutshell

SDN **outsources** and **consolidates** control over multiple devices to a software controller.



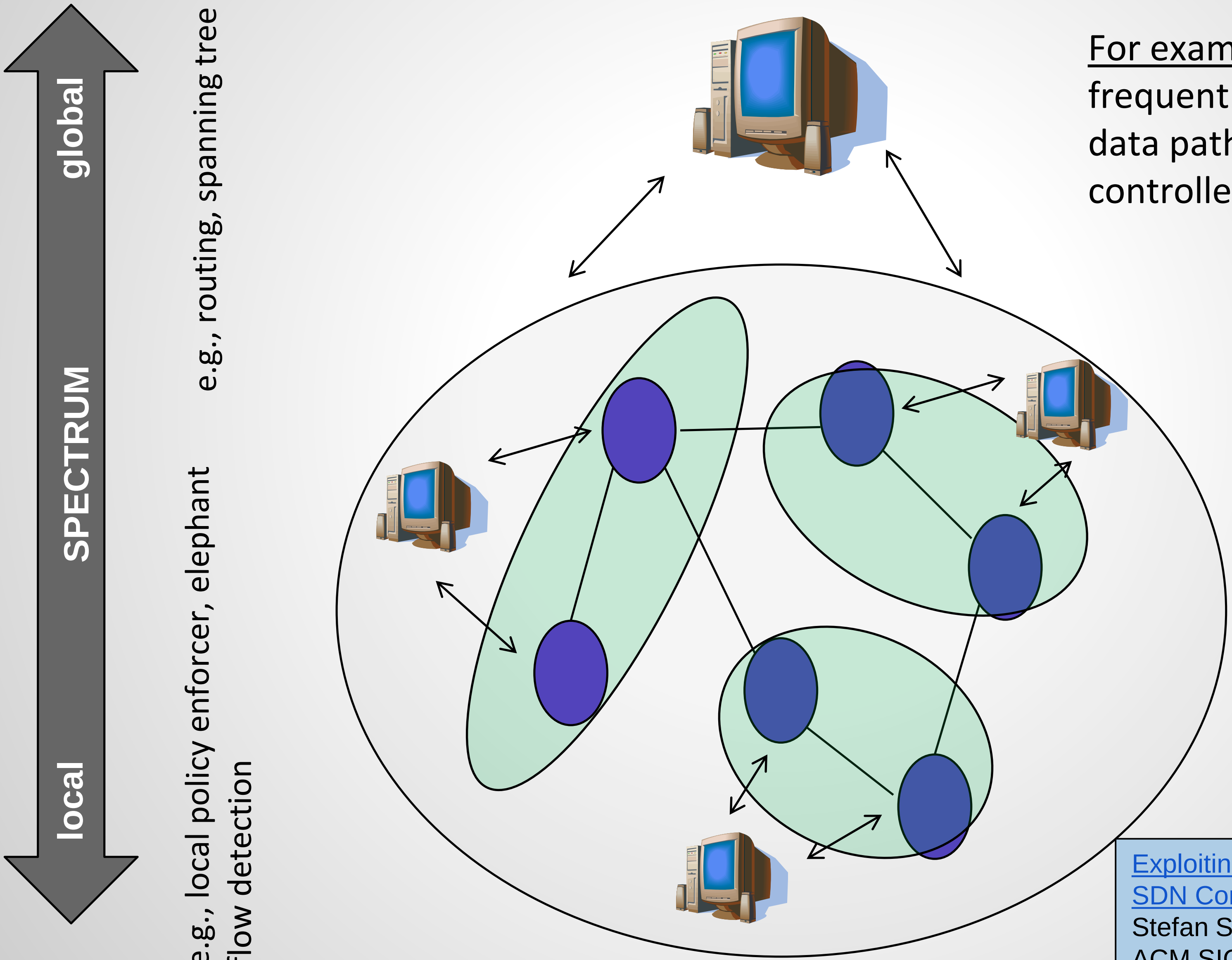
Careful! Controller is only logically centralized but actually distributed!

Distributed Challenge 1:
What can and should be controlled locally?



For example: Handle frequent events close to data path, shield global controllers.

Distributed Challenge 1:
What can and should be controlled locally?



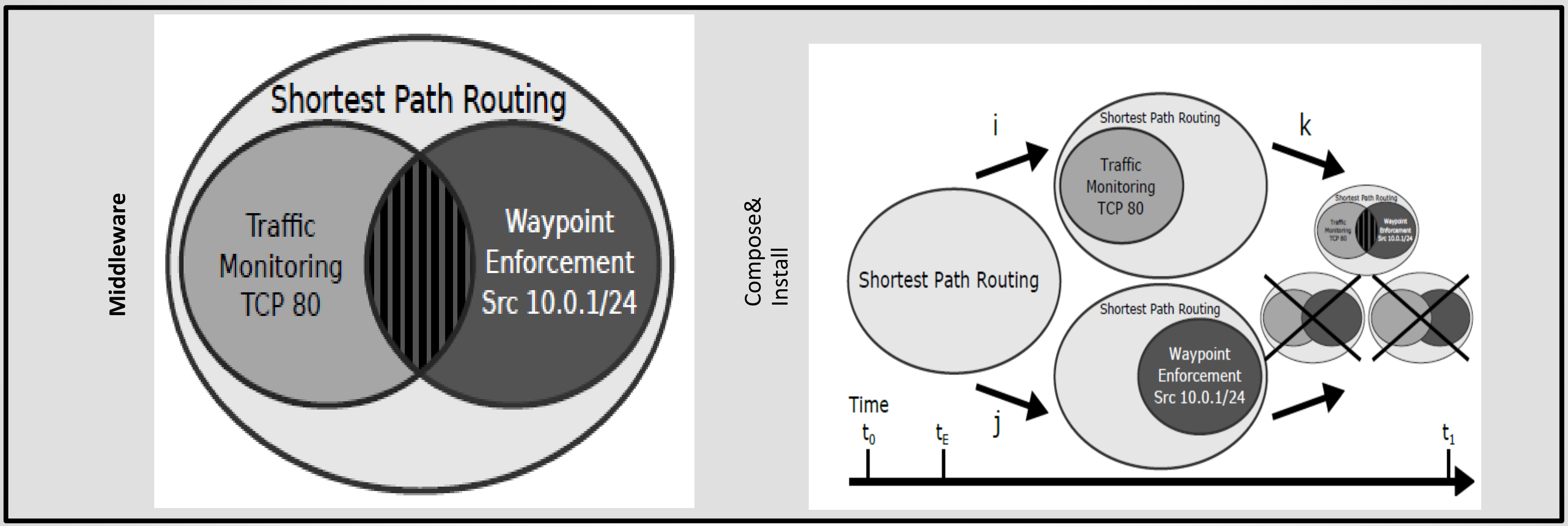
<u>For example:</u> Handle frequent events close to data path, shield global controllers.

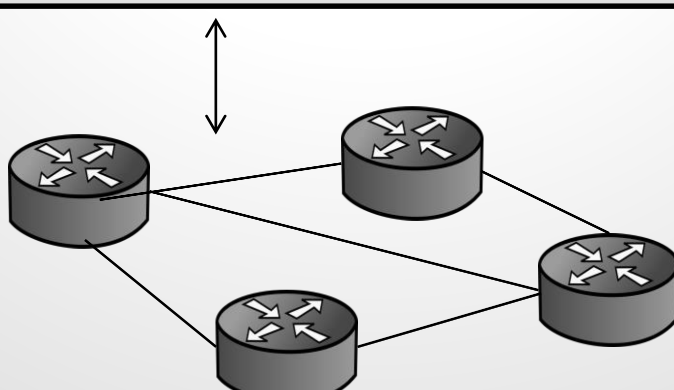
[Exploiting Locality in Distributed SDN Control](#)
Stefan Schmid and Jukka Suomela. ACM SIGCOMM **HotSDN** 2013.

Distributed Challenge 2:
How to deal with concurrency?

In charge
of ACLs

In charge
of tunnels

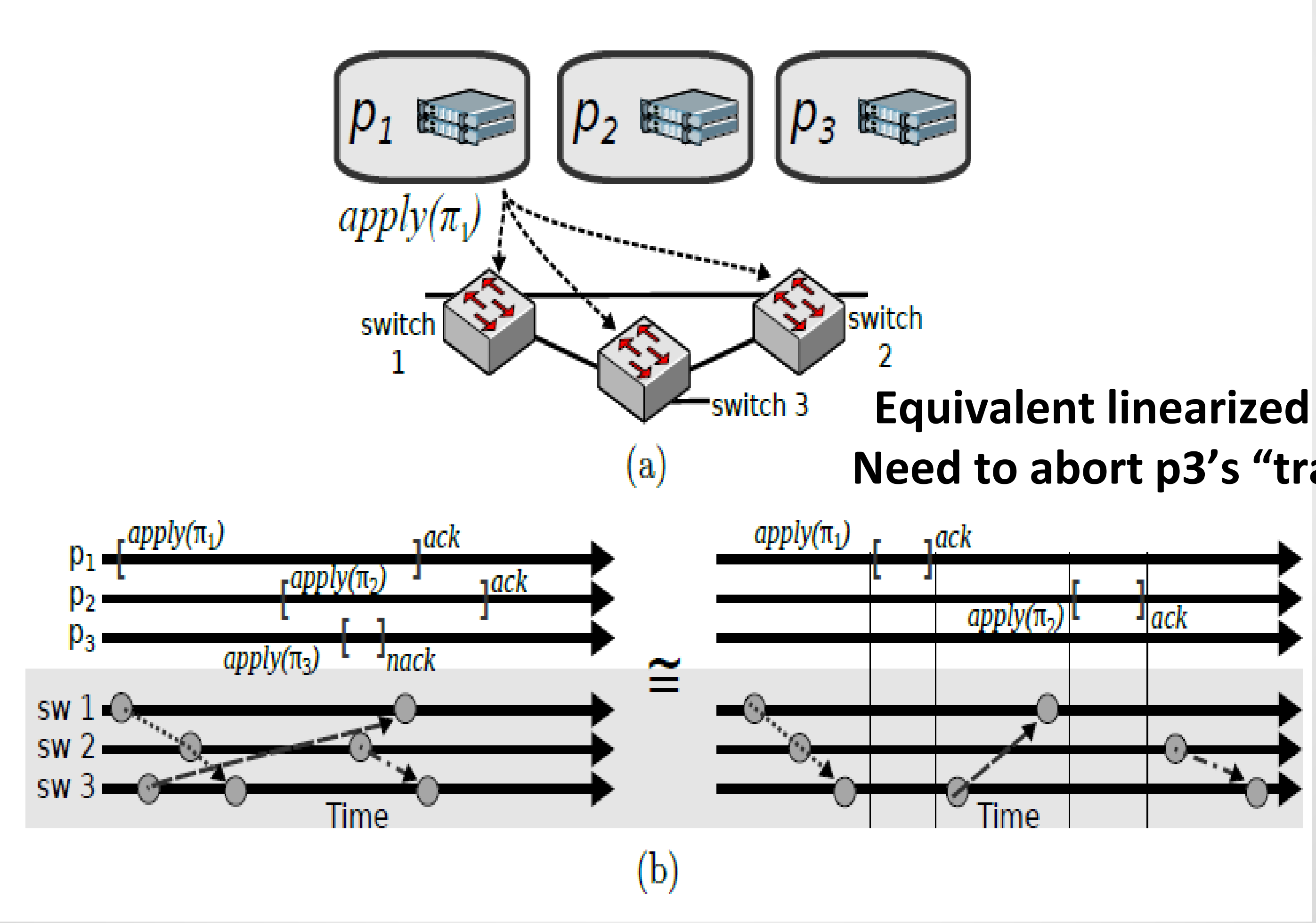




Distributed Challenge 2:
How to deal with concurrency?

Problem: Conflict free, per-packet consistent policy composition and installation

Holy Grails: Linearizability (Safety), Wait-freedom (Liveness)

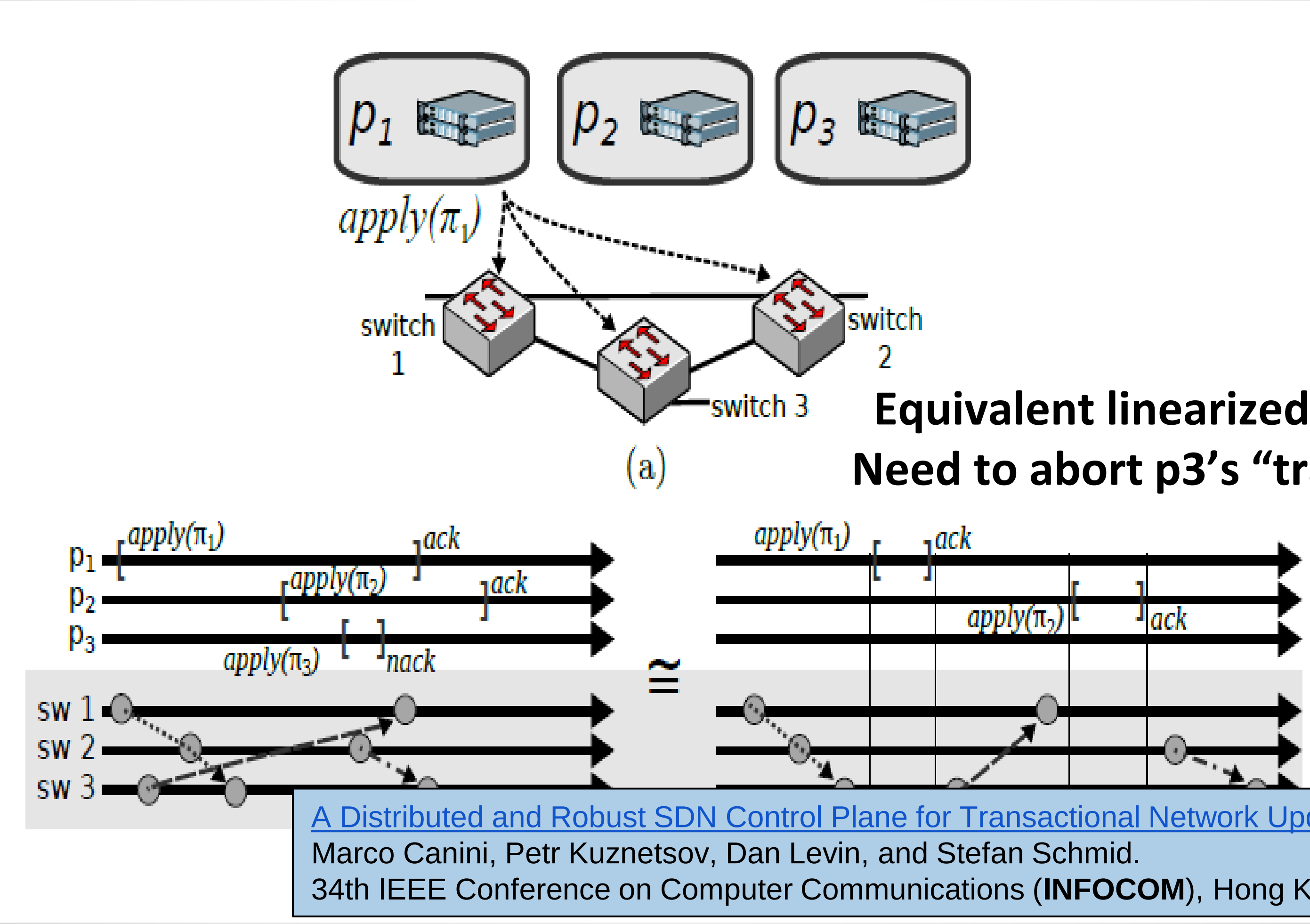


Equivalent linearized schedule!
Need to abort p3's "transaction".

Distributed Challenge 2:
How to deal with concurrency?

Problem: Conflict free, per-packet consistent policy composition and installation

Holy Grails: Linearizability (Safety), Wait-freedom (Liveness)



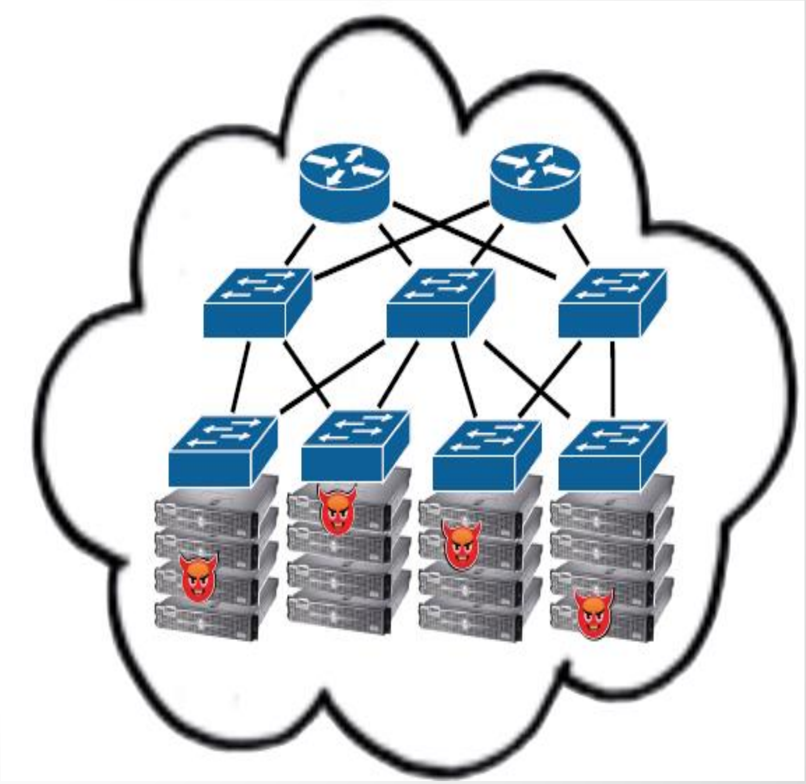
Equivalent linearized schedule!
Need to abort p3's "transaction".

[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..

Focus of this talk: Consistent Network Updates

Important, e.g., in Cloud

What if your traffic was *not* isolated from other tenants during periods of routine maintenance?



Example: Outages

Even technically sophisticated companies are struggling to build networks that provide reliable performance.

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

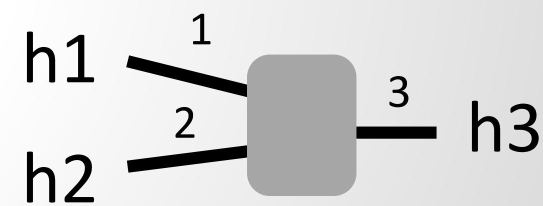
Thanks to Nate Foster for examples (at PODC 2014)!

The SDN *Hello World*: MAC Learning (Distributed Challenge 3 resp. *Fail*)

Distributed Computing Fail: Updating a Single Switch

Already updating a single switch from a single controller is non-trivial!

- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets
- Example
  - h1 sends to h2:
  - h3 sends to h1:
  - h1 sends to h3:



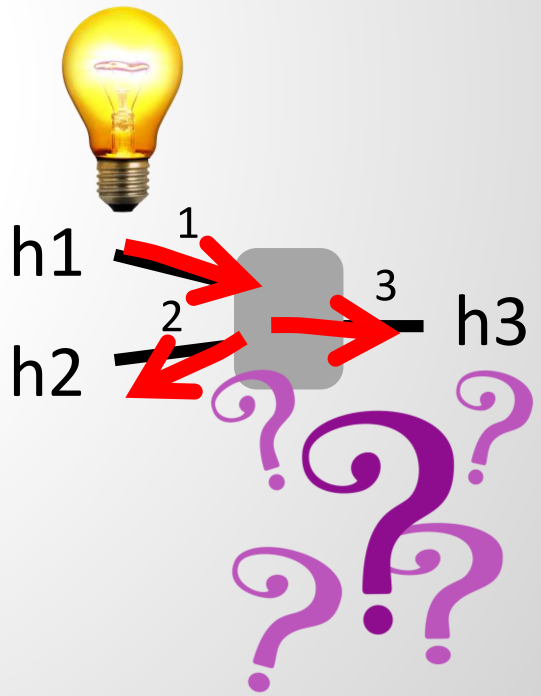
Thanks to Jennifer Rexford for example!

Distributed Computing Fail: Updating a Single Switch

Already updating a single switch from a single controller is non-trivial!

- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets
- Example
  - h1 sends to h2:

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



Thanks to Jennifer Rexford for example!

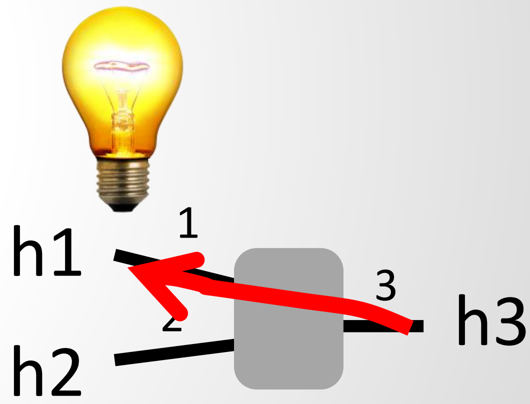
Distributed Computing Fail: Updating a Single Switch

Already updating a single switch from a single controller is non-trivial!

- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets
- Example
  - h1 sends to h2:

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

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



Thanks to Jennifer Rexford for example!

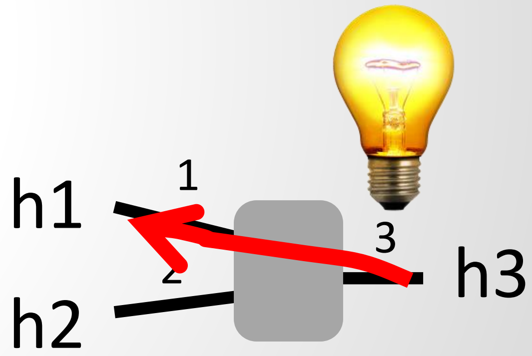
Distributed Computing Fail: Updating a Single Switch

Already updating a single switch from a single controller is non-trivial!

- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets
- Example
  - h1 sends to h2:

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

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



Thanks to Jennifer Rexford for example!

Distributed Computing Fail: Updating a Single Switch

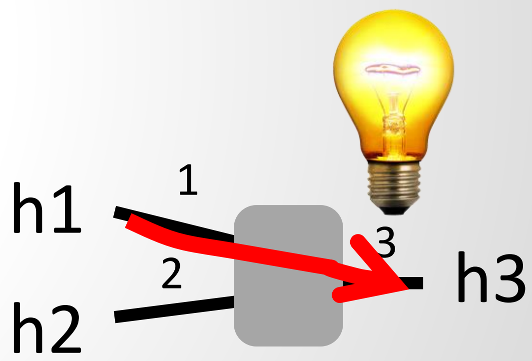
Already updating a single switch from a single controller is non-trivial!

- Fundamental networking task: MAC learning
  - Flood packets sent to unknown destinations
  - Learn host's location when it sends packets
- Example
  - h1 sends to h2:

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

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

    forward to p3



Thanks to Jennifer Rexford for example!

Distributed Computing Fail: Updating a Single Switch

Already updating a single switch from a single controller is non-trivial!

- Fundamental task: MAC learning
  - Flood packets sent to unknown destinations

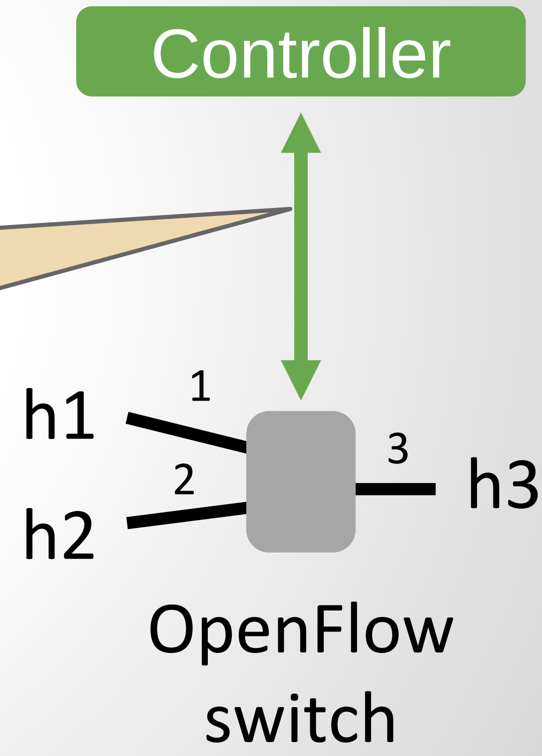
Now: how to do via controller?
Install rules as you learn!
And match on host address and port.

  - h3 sends to h1:

    forward to p1, learn (h3,p3)

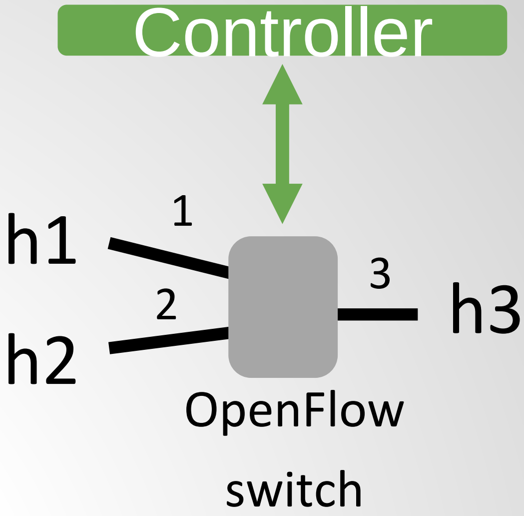
  - h1 sends to h3:

    forward to p3



Thanks to Jennifer Rexford for example!

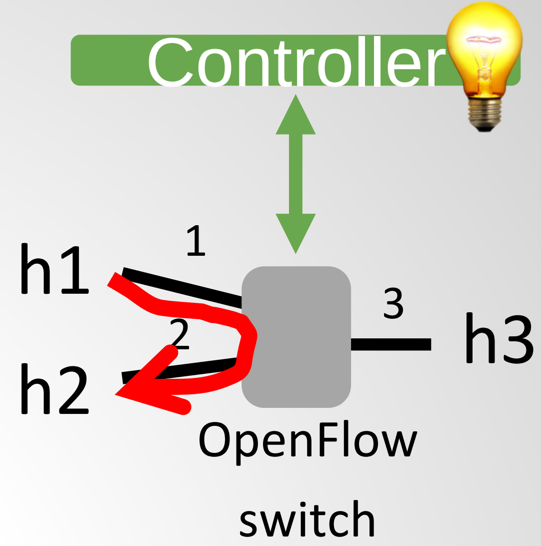
Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

- What happens when h1 sends to h2?

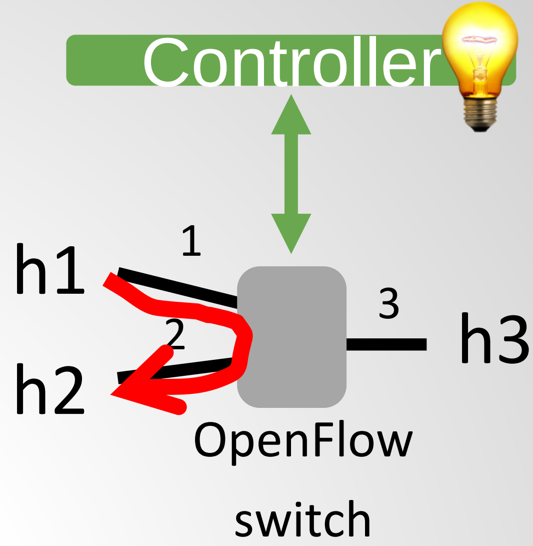
Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

- What happens when h1 sends to h2?
  - Controller learns that h1@p1 and installs rule on switch!

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

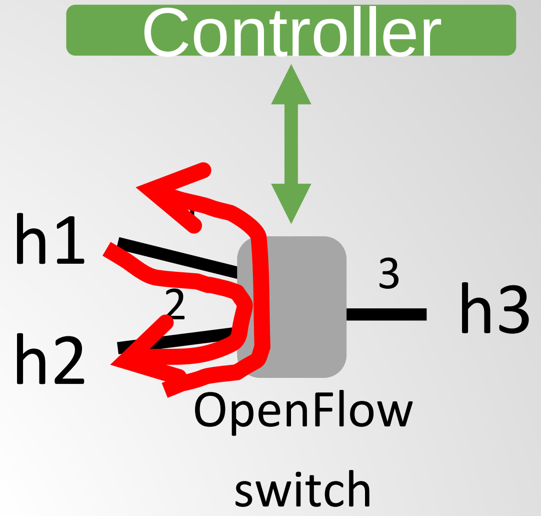
Pattern	Action
*	Send to controller

h1 sends to h2

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

- What happens when h1 sends to h2?
  - Controller learns that h1@p1 and installs rule on switch!

Example: SDN MAC Learning Done Wrong



- ❑ Initial rule *: Send everything to controller

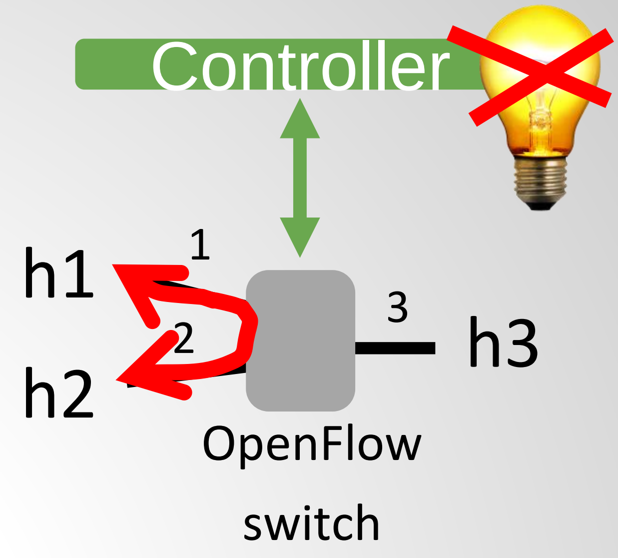
Pattern	Action
*	Send to controller

h1 sends to h2

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

- ❑ What happens when h2 sends to h1?

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

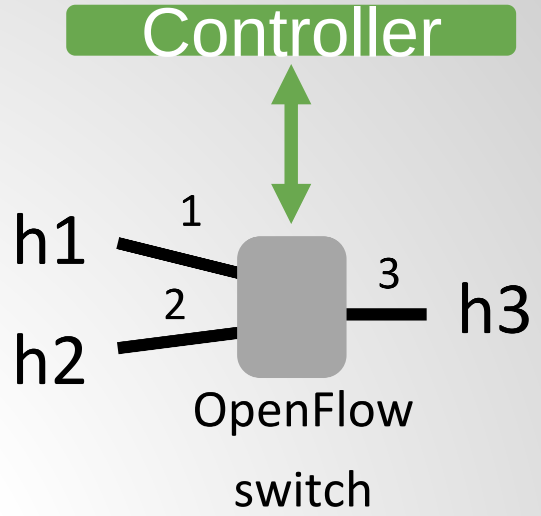
Pattern	Action
*	Send to controller

h1 sends to h2

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

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

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

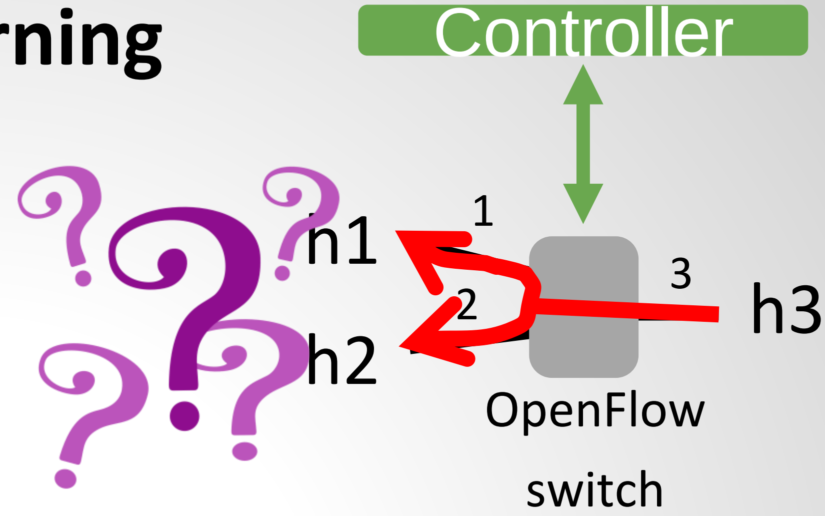
Pattern	Action
*	Send to controller

h1 sends to h2

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

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

Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

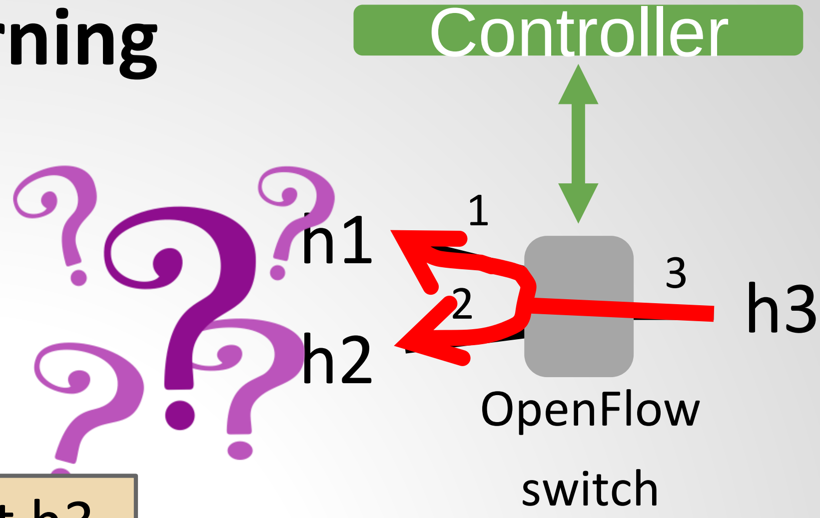
Pattern	Action
*	Send to controller

h1 sends to h2

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

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

Example: SDN MAC Learning Done Wrong



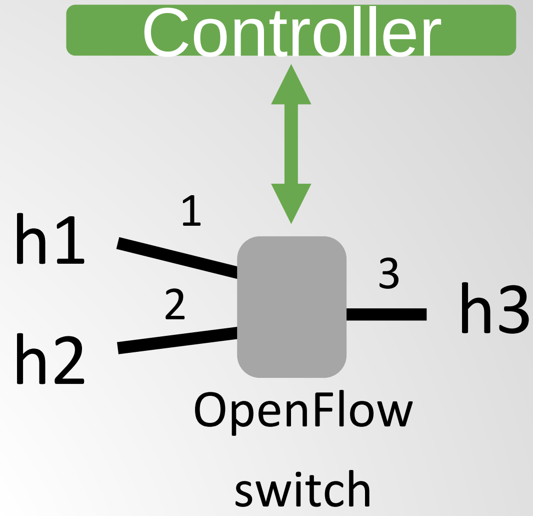
- Initial rule *: Send everything to controller

> Controller however does learn about h3. Then answer from h2 missed by controller too: all future requests to h2 flooded?!?

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

- Wha… pens when h2 sends to h1?
  - Switc… ows destination: message forwarded to h1
  - No cont… oller interaction, no new rule for h2
- What happens when h3 sends to h2?
  - Flooded! Controller did not put the rule to h2!

Example: SDN MAC Learning Done Wrong

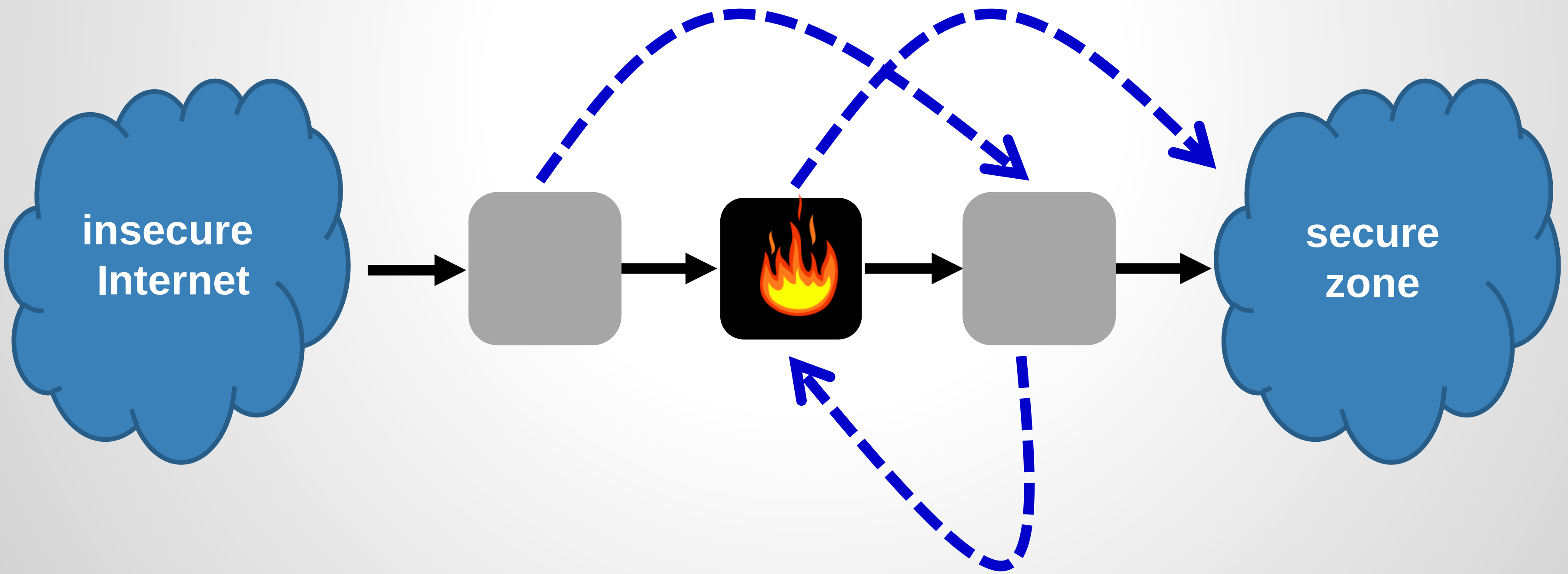


- Initial rule *: Send everything to controller

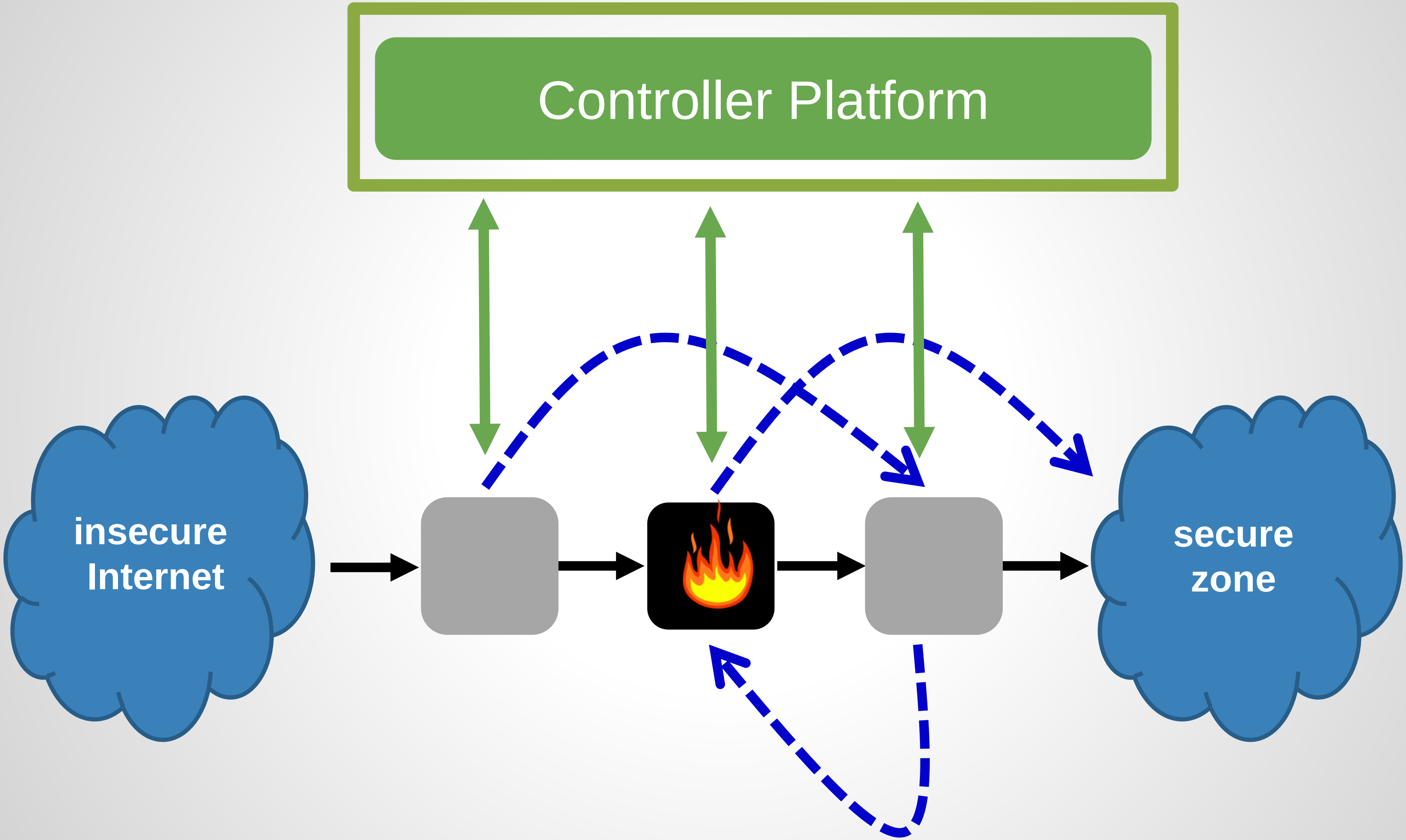
A bug in early controller software.
Hard to catch! A performance issue, not a consistency one (arguably a key strength of SDN?).

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

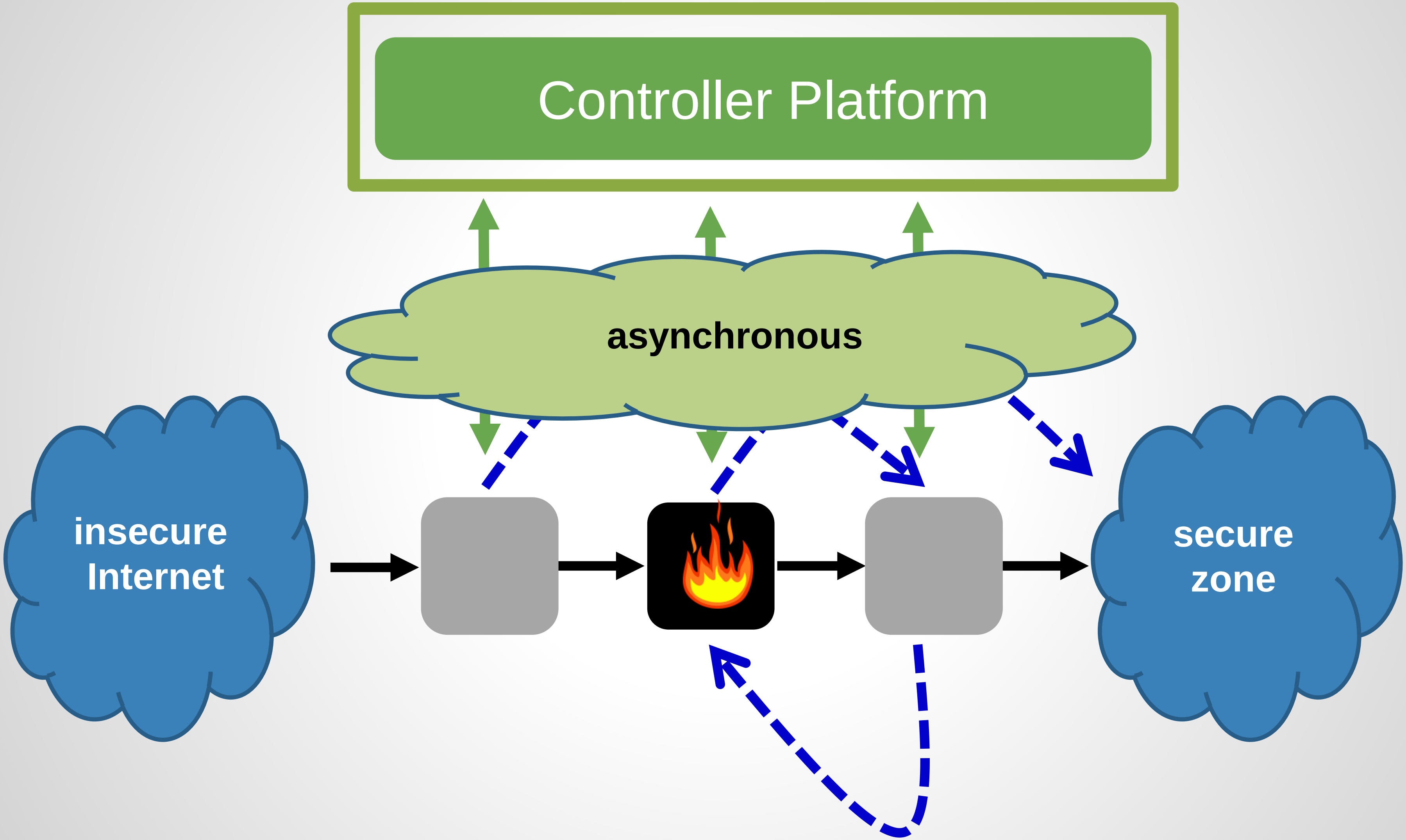
Distributed Challenge 4: Multi-Switch Updates



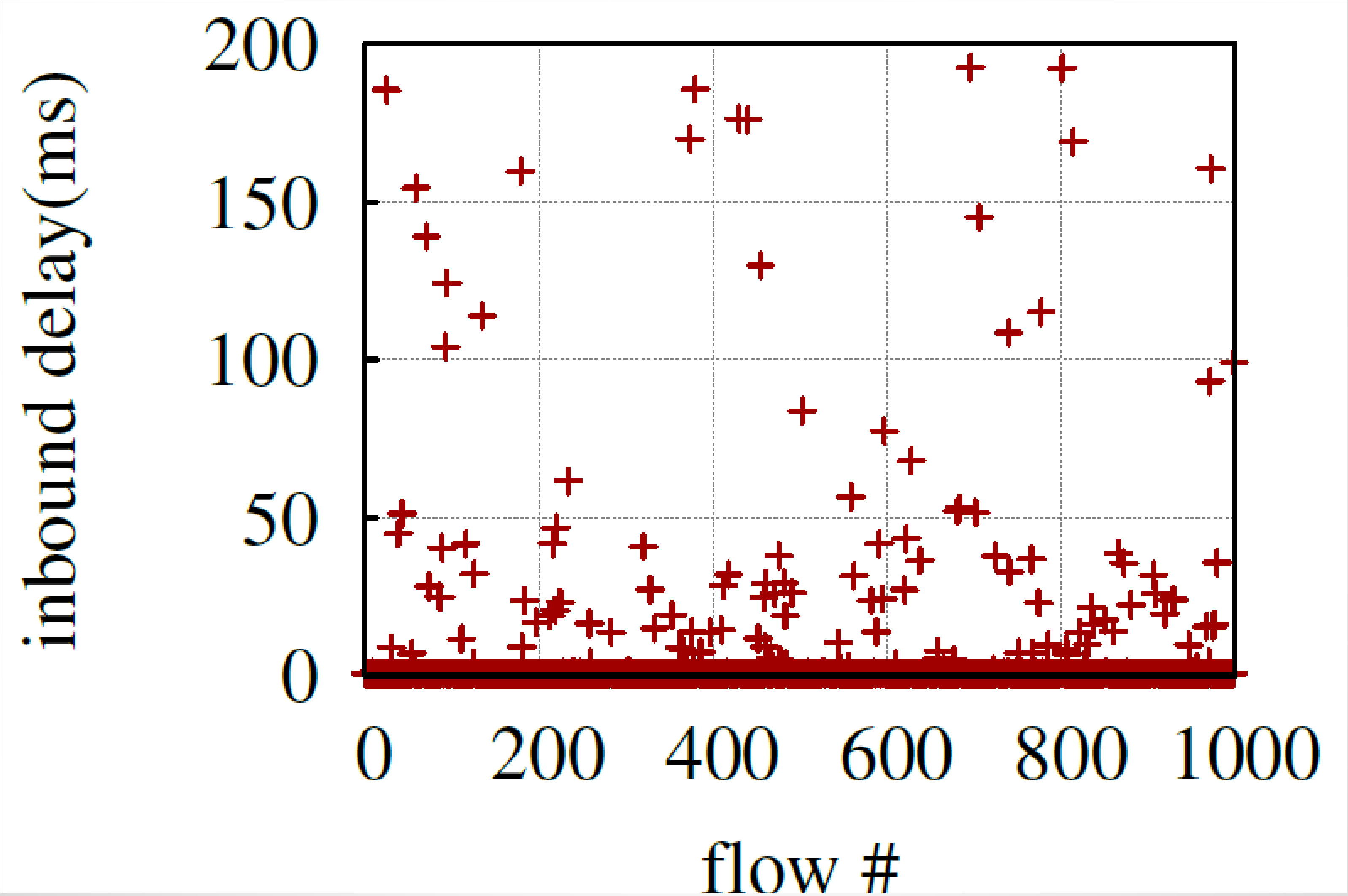
Distributed Challenge 4: Multi-Switch Updates



Distributed Challenge 4: Multi-Switch Updates



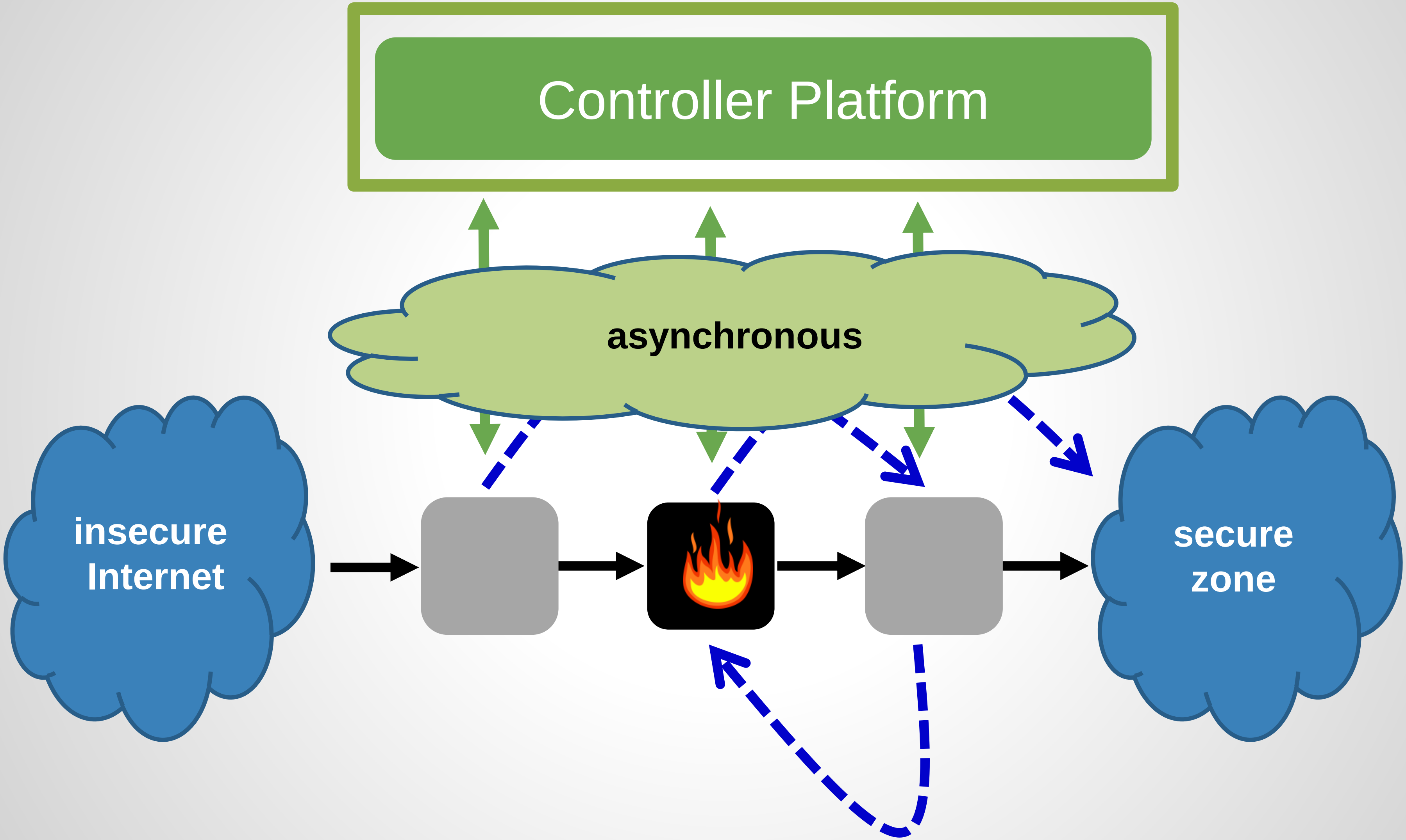
An Asynchronous Distributed System!



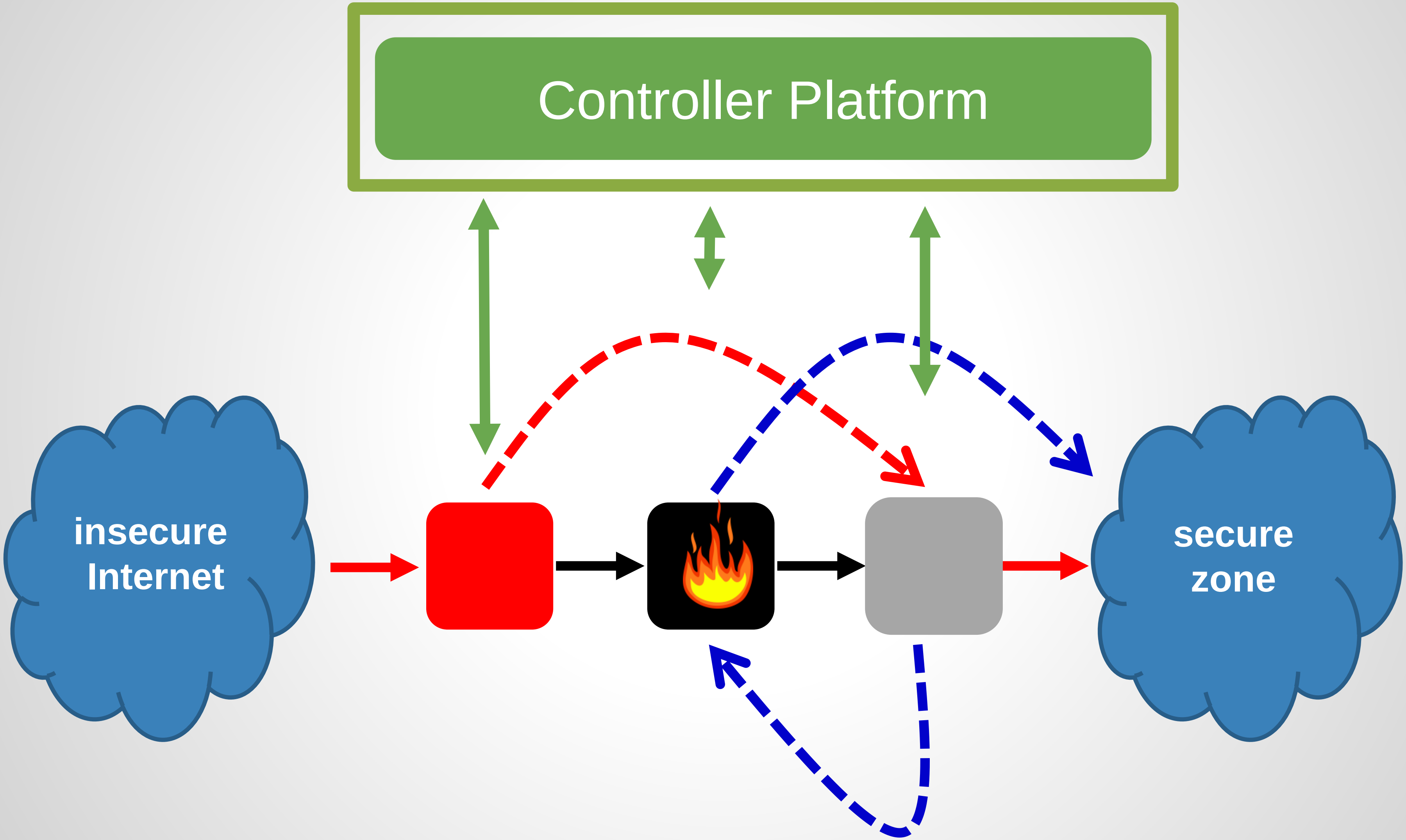
He et al., ACM SOSR 2015:

without network latency

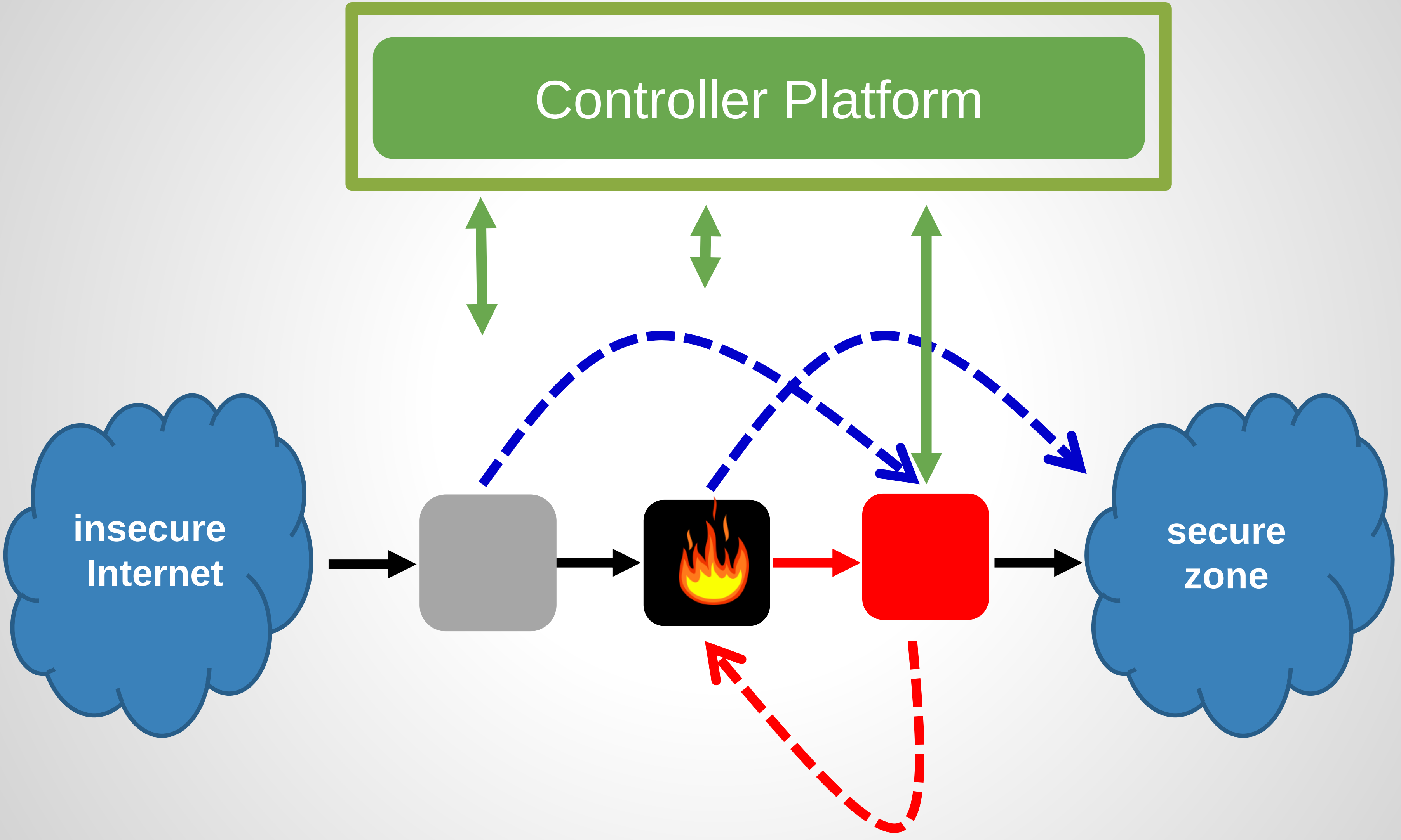
What Can Go Wrong?



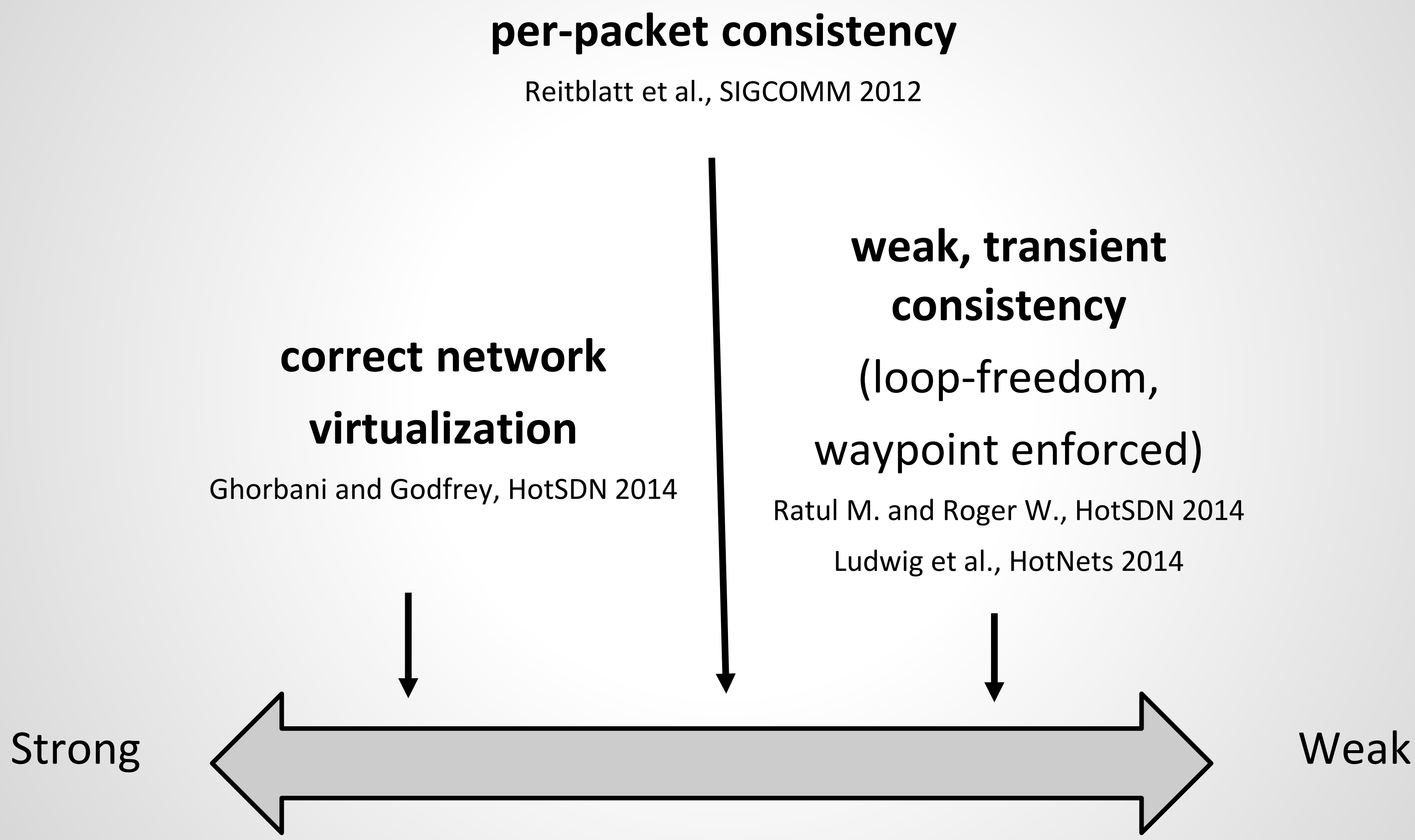
Example 2.1: Bypassed Waypoint



Example 2.2: Loop



The Spectrum of Consistency

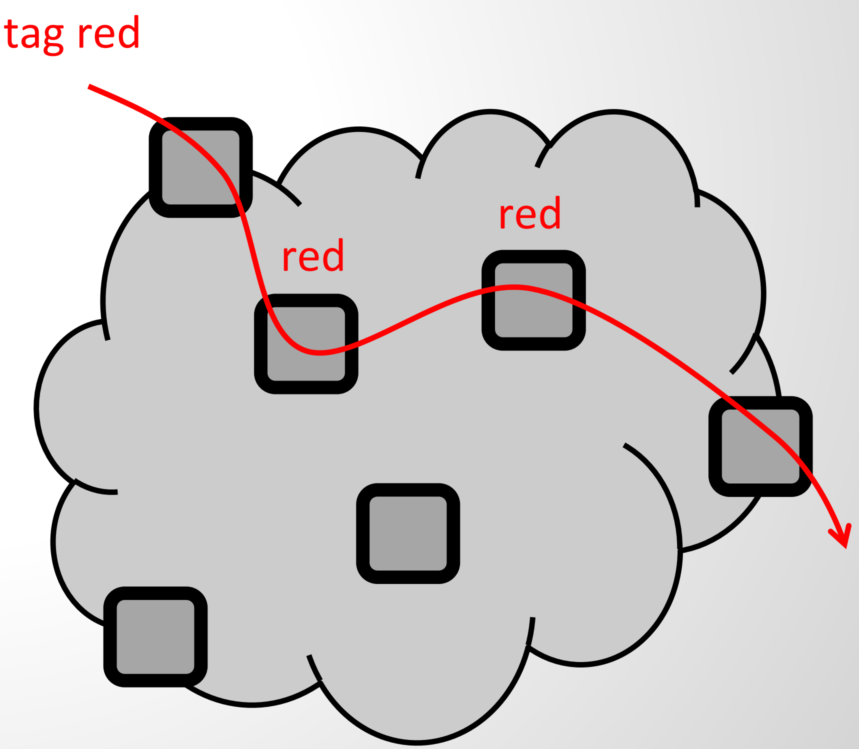


Example: Per-Packet Consistency

<u>Definition:</u> Any packet should either traverse the old route, or the new route, *but not a mixture*

<u>Implementation:</u>

- ❑ 2-Phase installation
- ❑ Tagging at ingress port

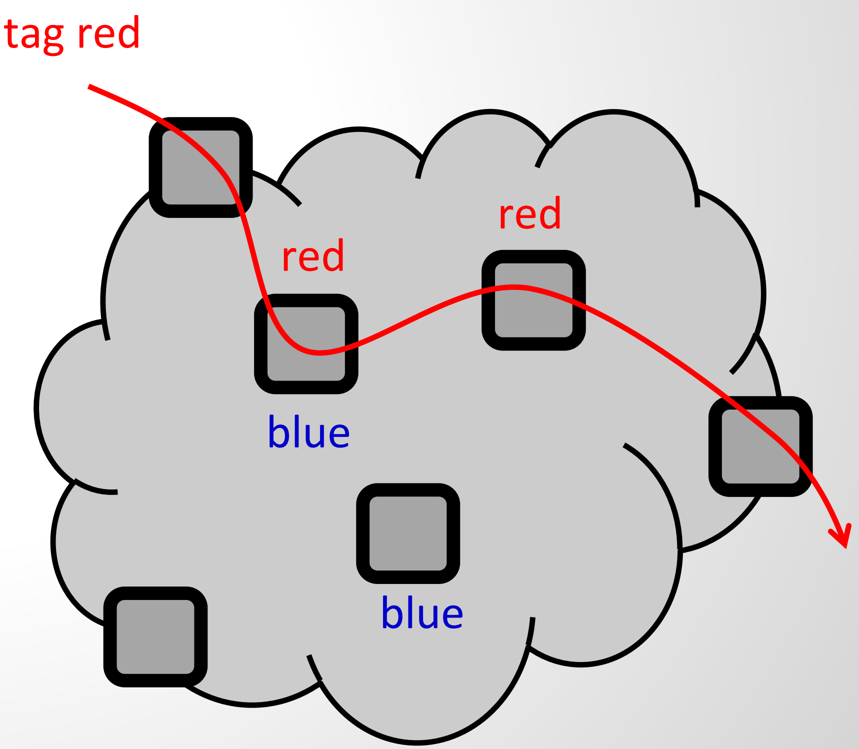


Example: Per-Packet Consistency

Definition: Any packet should either traverse the old route, or the new route, *but not a mixture*

Implementation:

- 2-Phase installation
- Tagging at ingress port



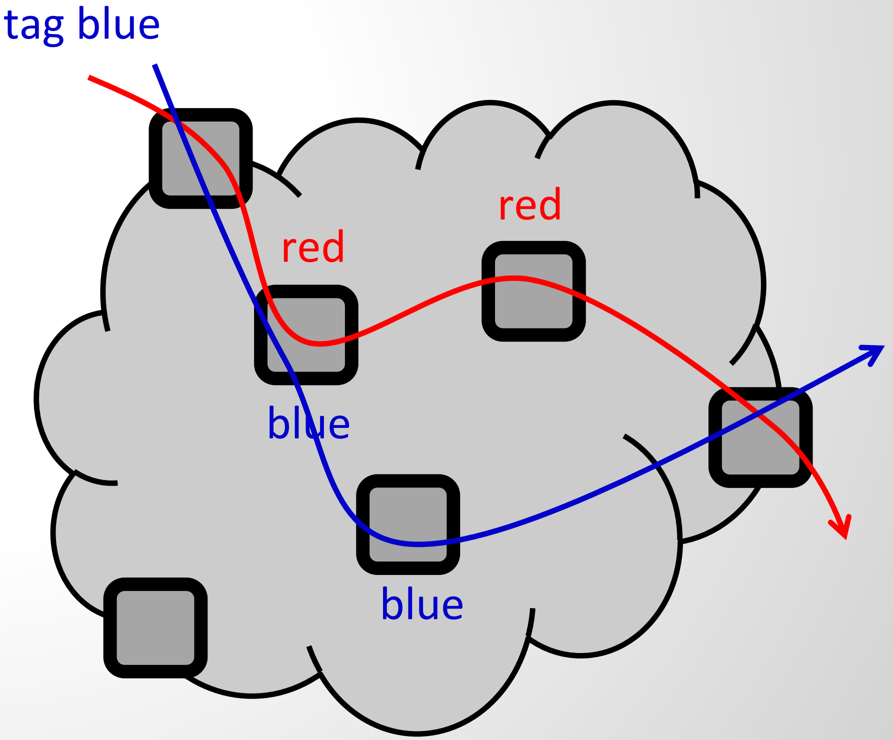
Start preparing new route!

Example: Per-Packet Consistency

Definition: Any packet should either traverse the old route, or the new route, *but not a mixture*

Implementation:

- 2-Phase installation
- Tagging at ingress port



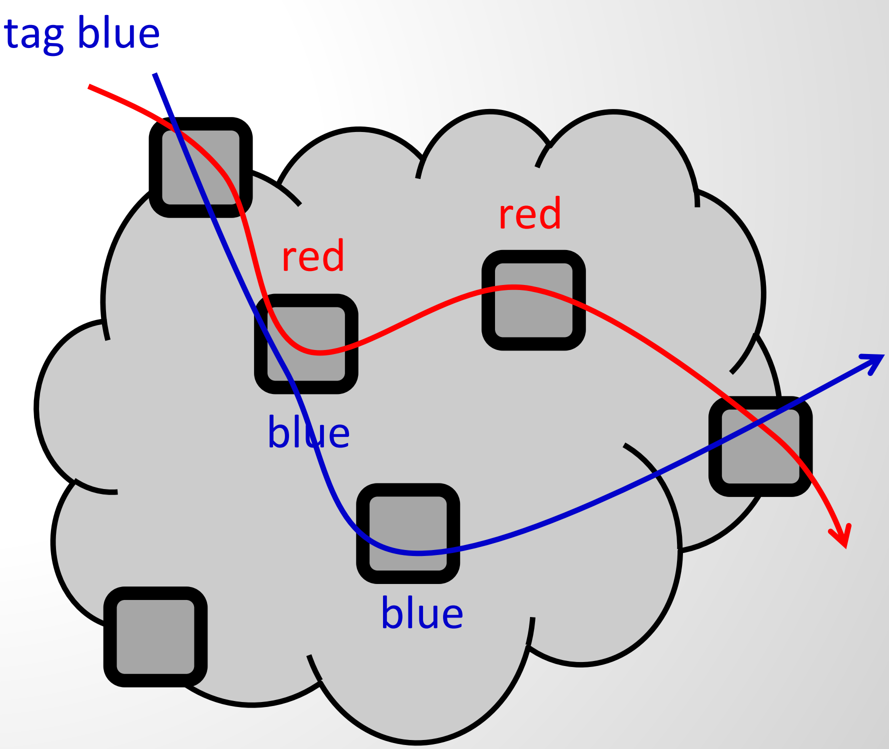
And then tag newly arriving packets!

Example: Per-Packet Consistency

Definition: Any packet should either traverse the old route, or the new route, *but not a mixture*

Implementation:

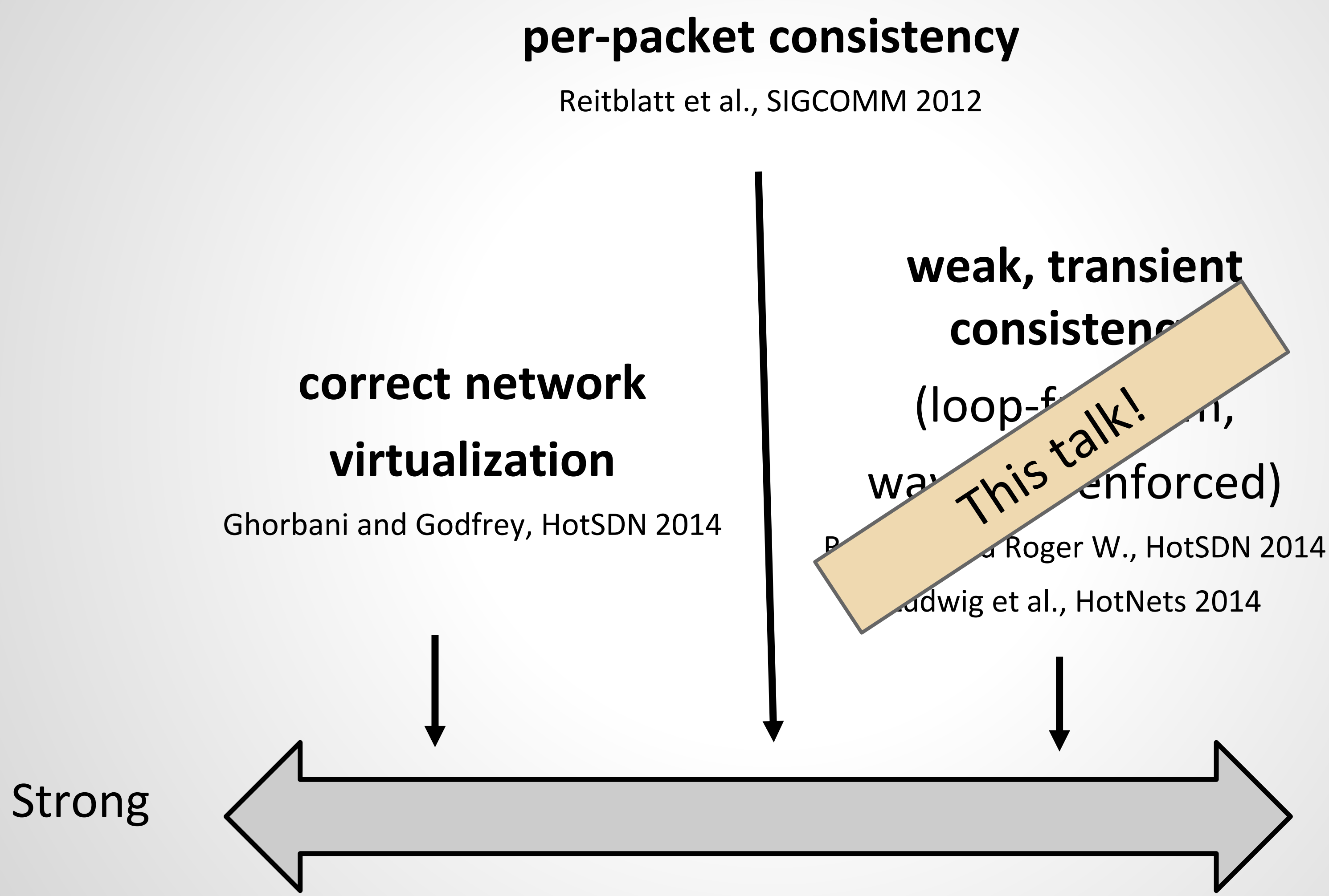
- 2-Phase installation
- Tagging at ingress port



Disadvantages:

- Tagging: memory
- Late effects

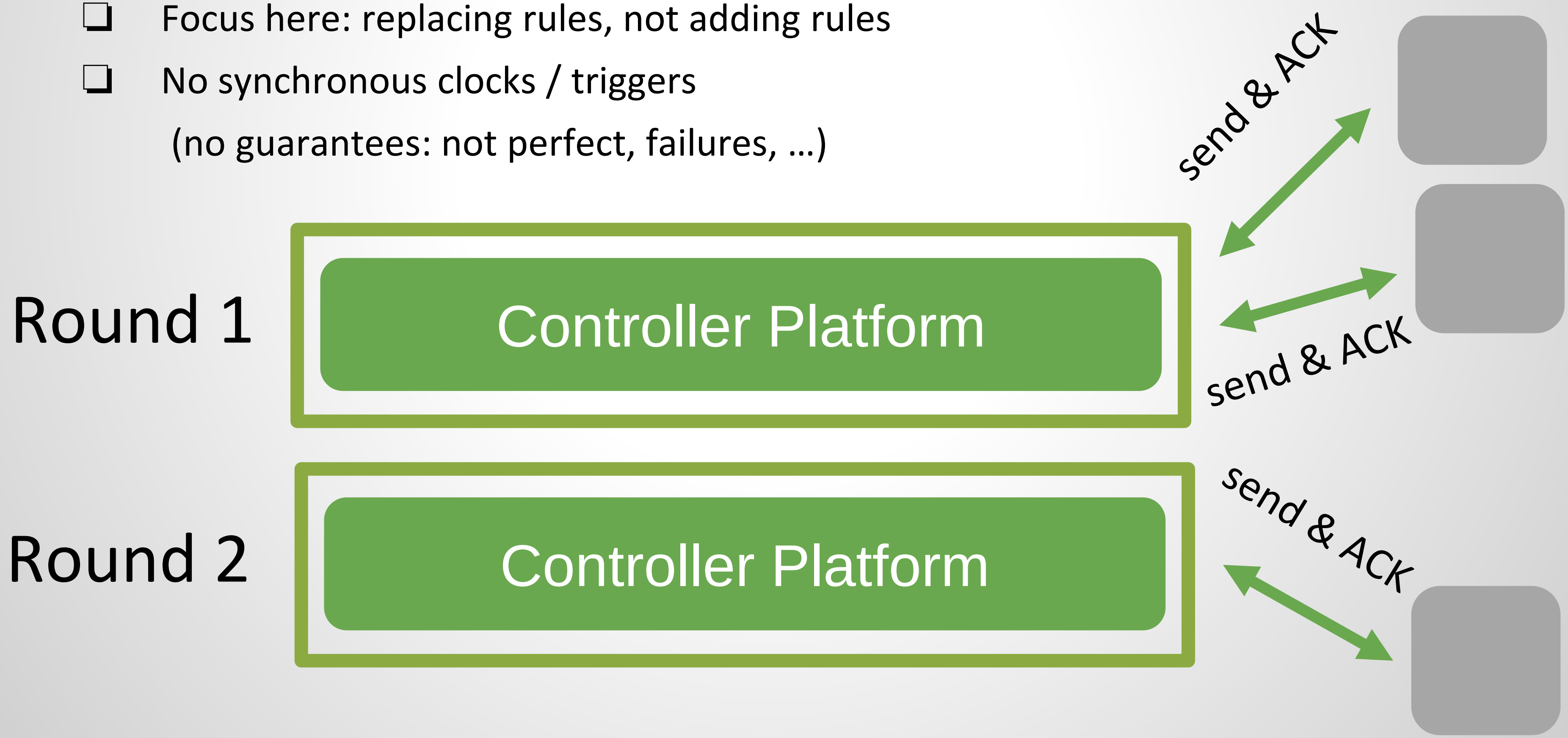
The Spectrum of Consistency



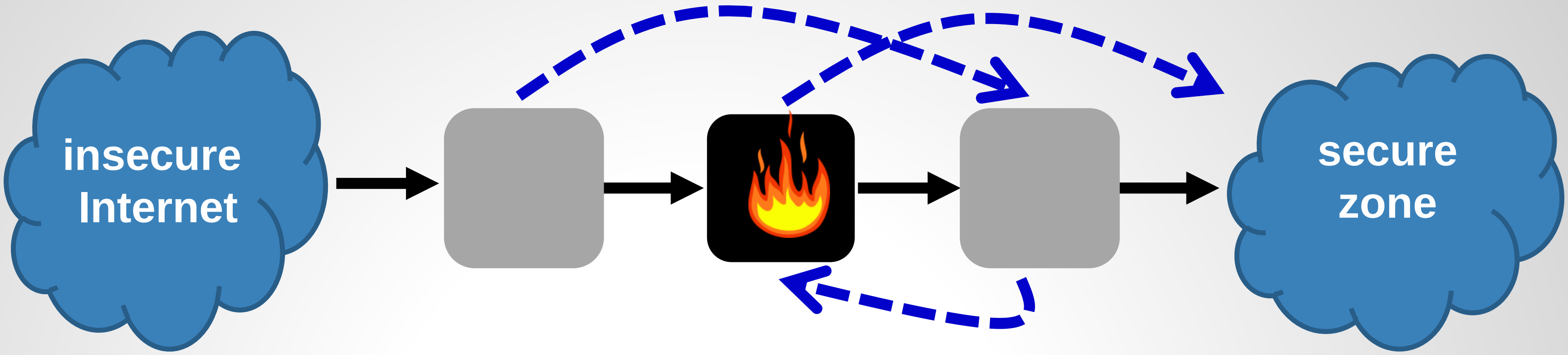
Implementing weaker transient consistency?

- Idea: Avoid tagging and keep consistent by updating in multiple rounds
  - No tagging needed
  - Focus here: replacing rules, not adding rules
  - No synchronous clocks / triggers

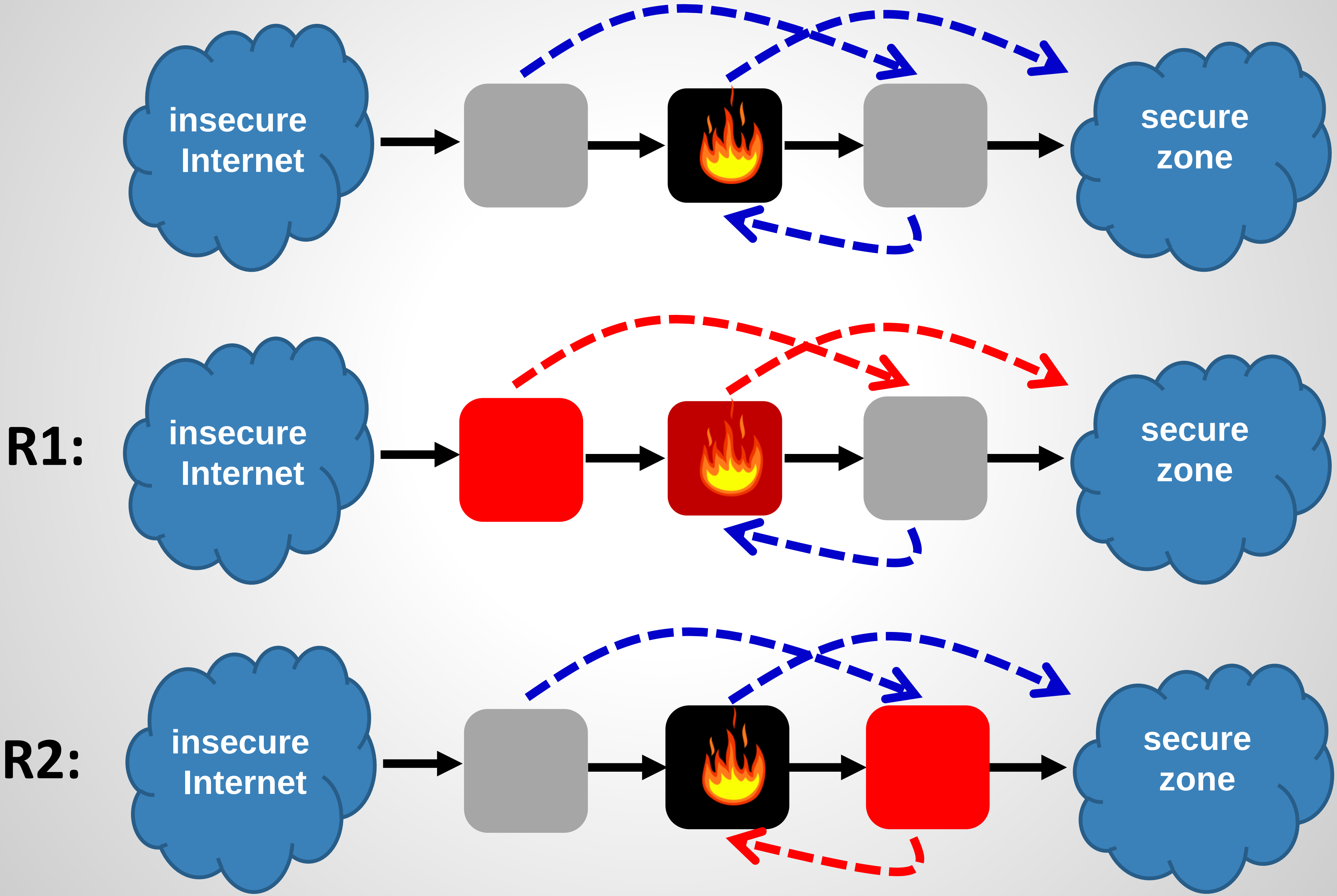
    (no guarantees: not perfect, failures, …)



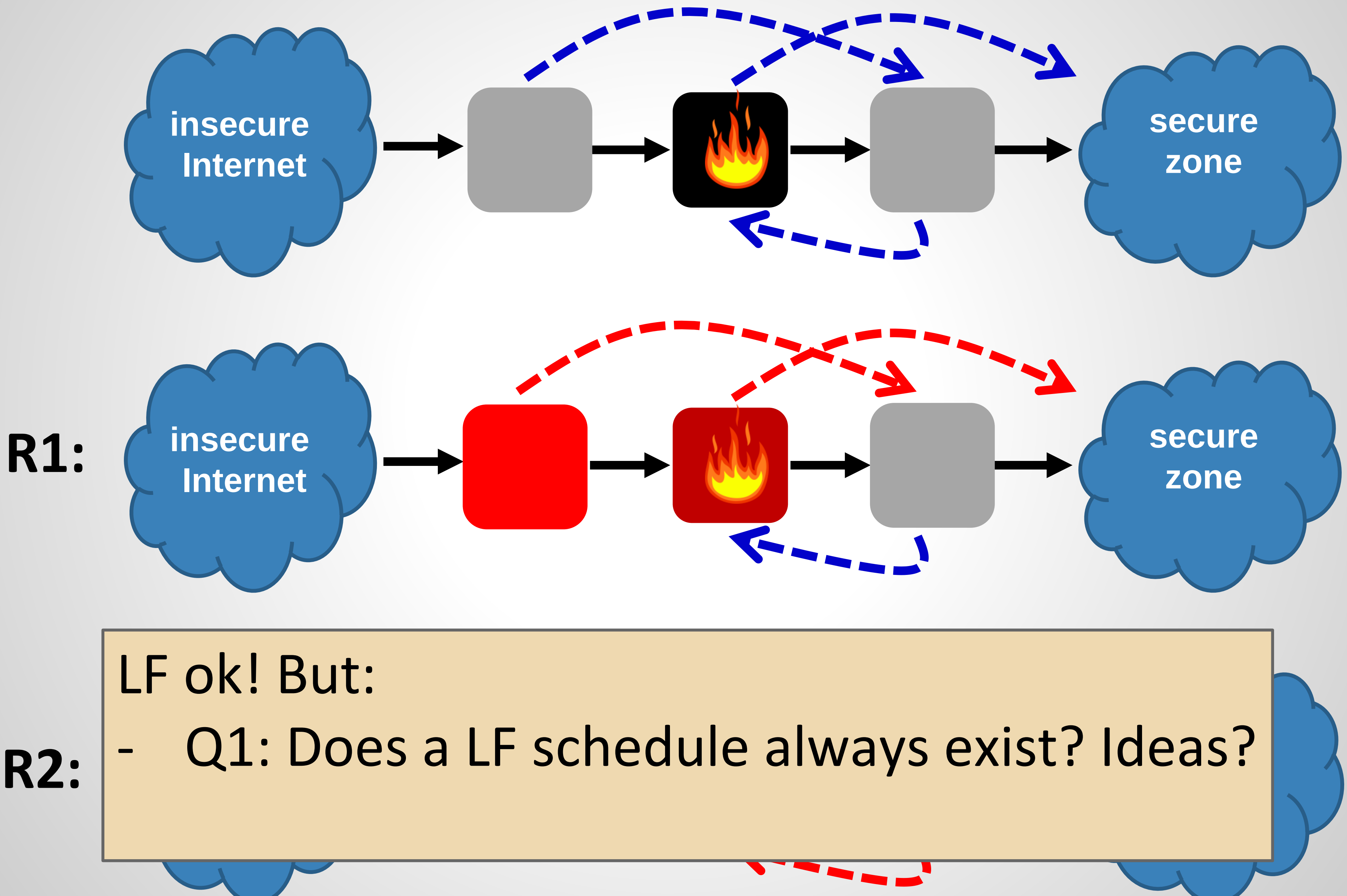
Going Back to Our Examples: LF Update?



Going Back to Our Examples: LF Update!



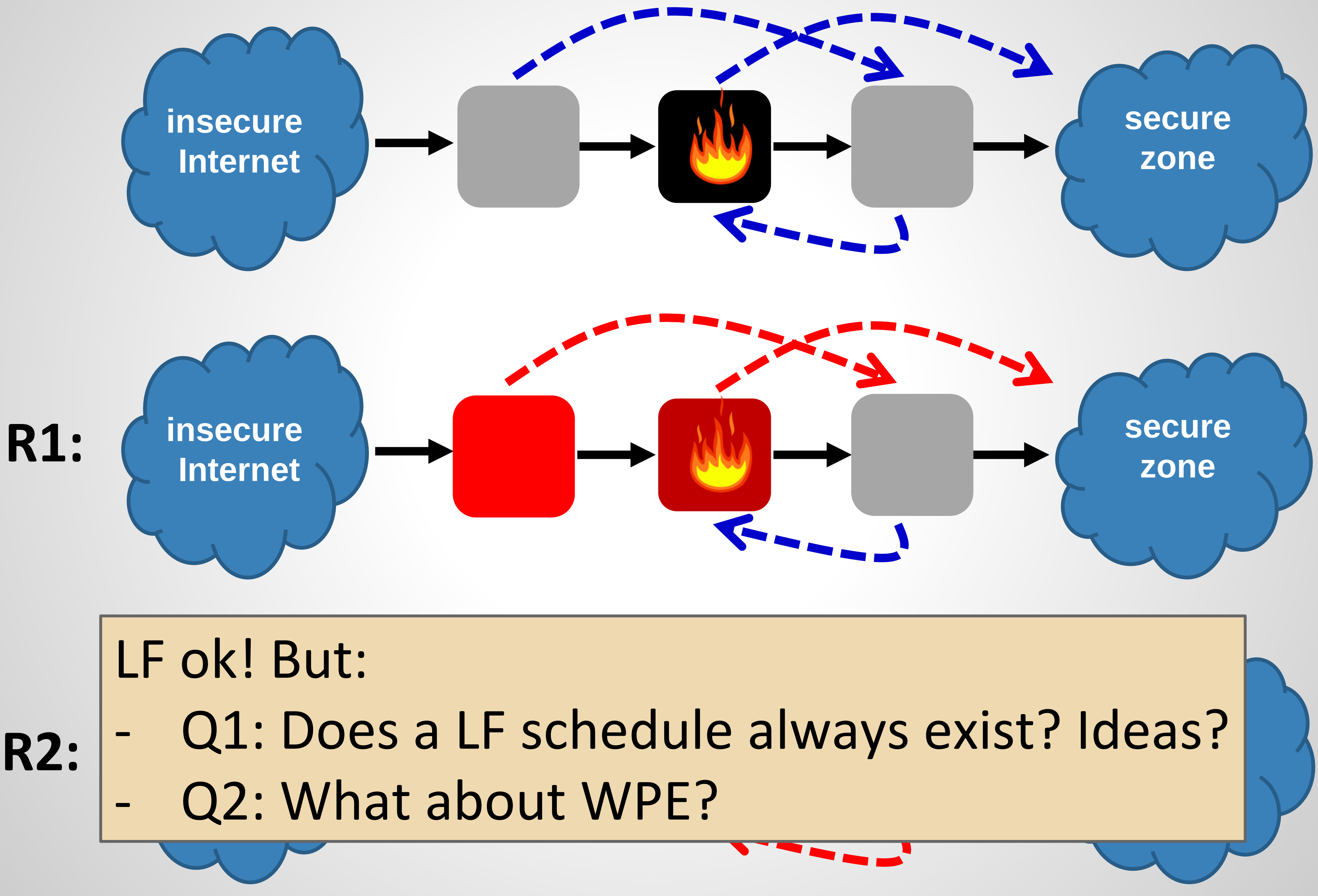
Going Back to Our Examples: LF Update!



LF ok! But:

- Q1: Does a LF schedule always exist? Ideas?

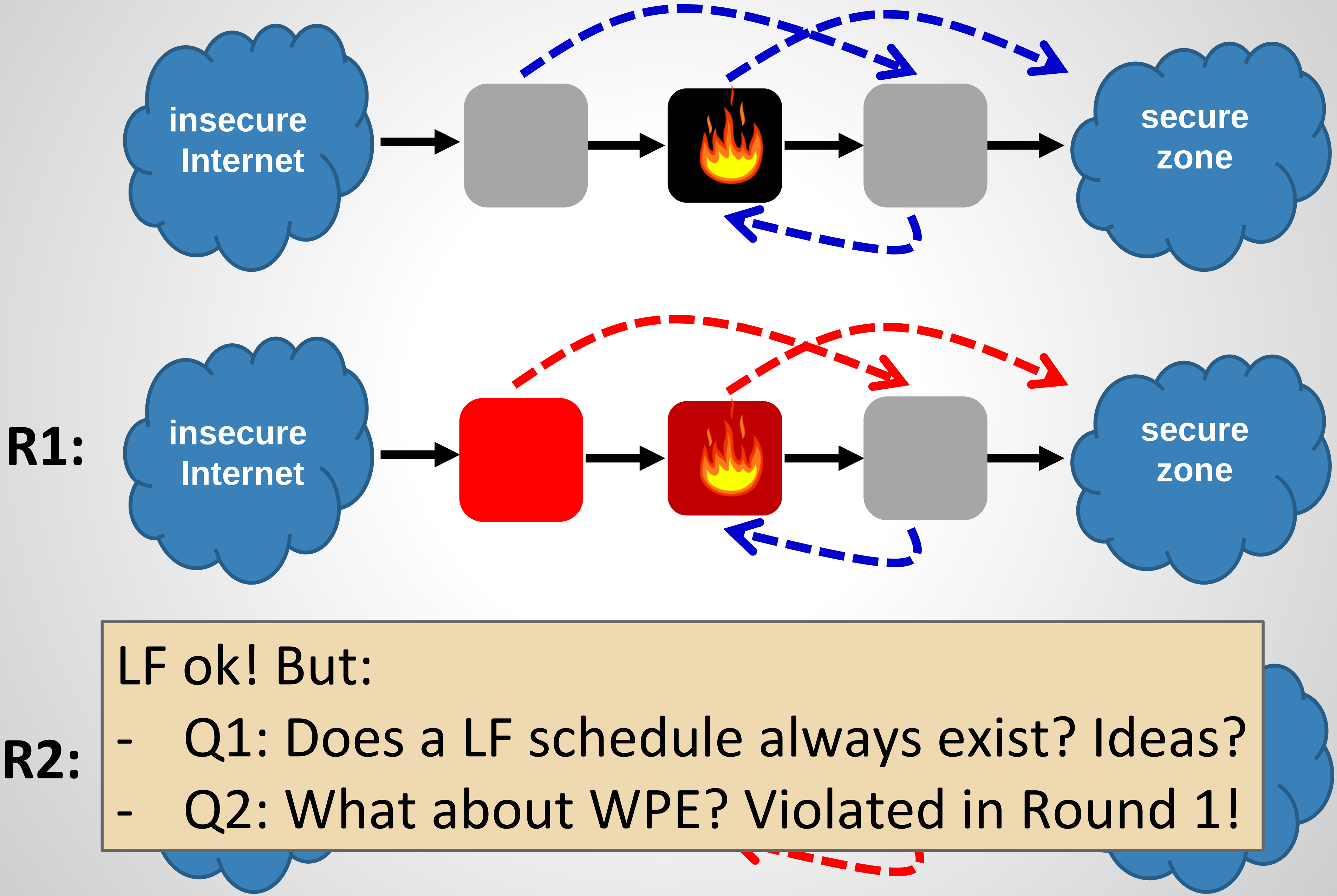
Going Back to Our Examples: LF Update!



LF ok! But:

- Q1: Does a LF schedule always exist? Ideas?
- Q2: What about WPE?

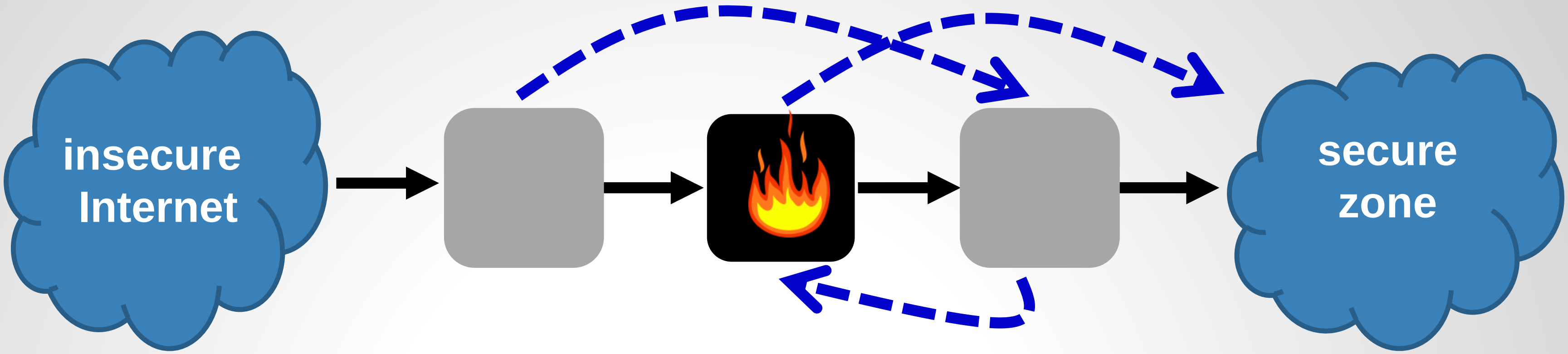
Going Back to Our Examples: LF Update!



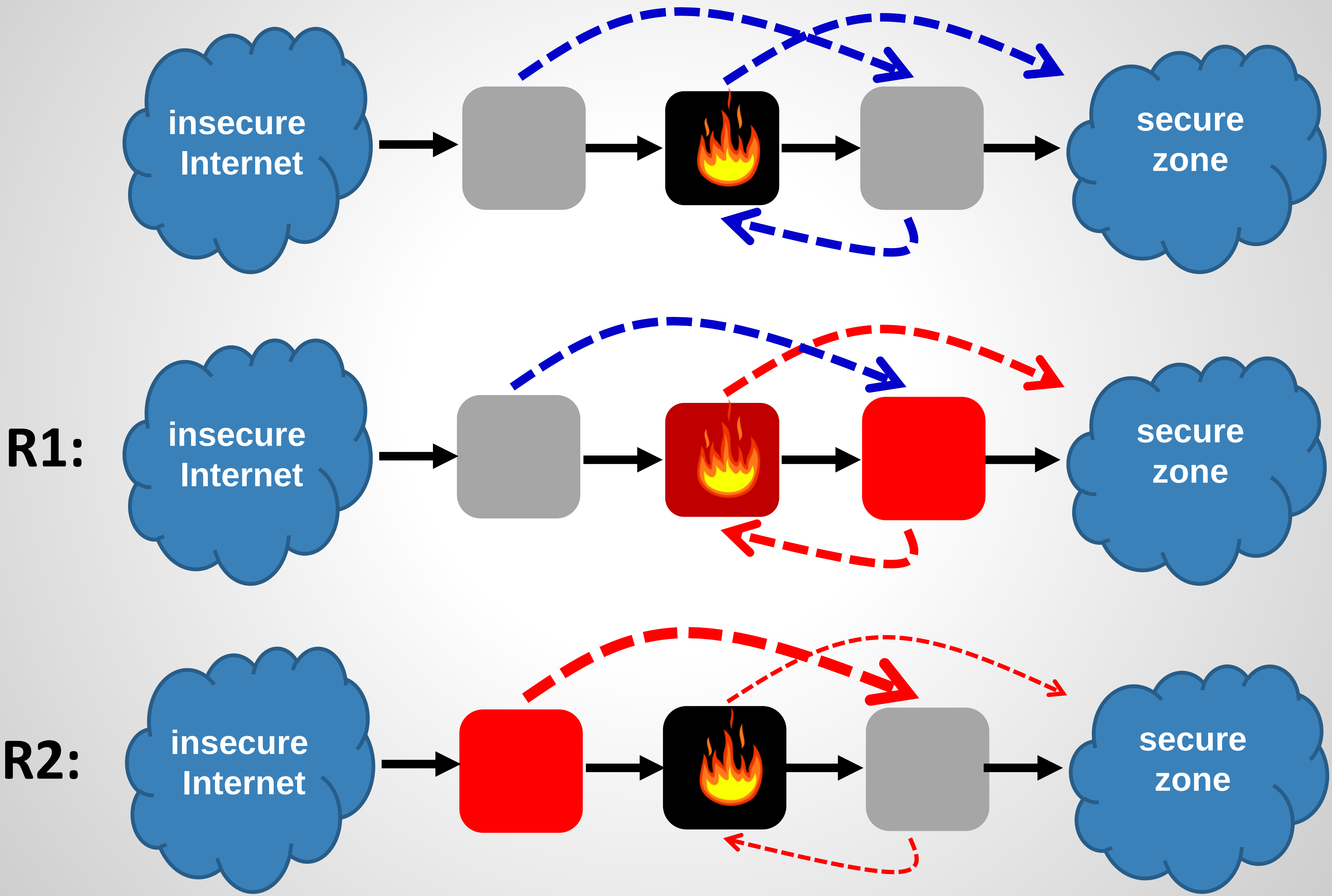
LF ok! But:

- Q1: Does a LF schedule always exist? Ideas?
- Q2: What about WPE? Violated in Round 1!

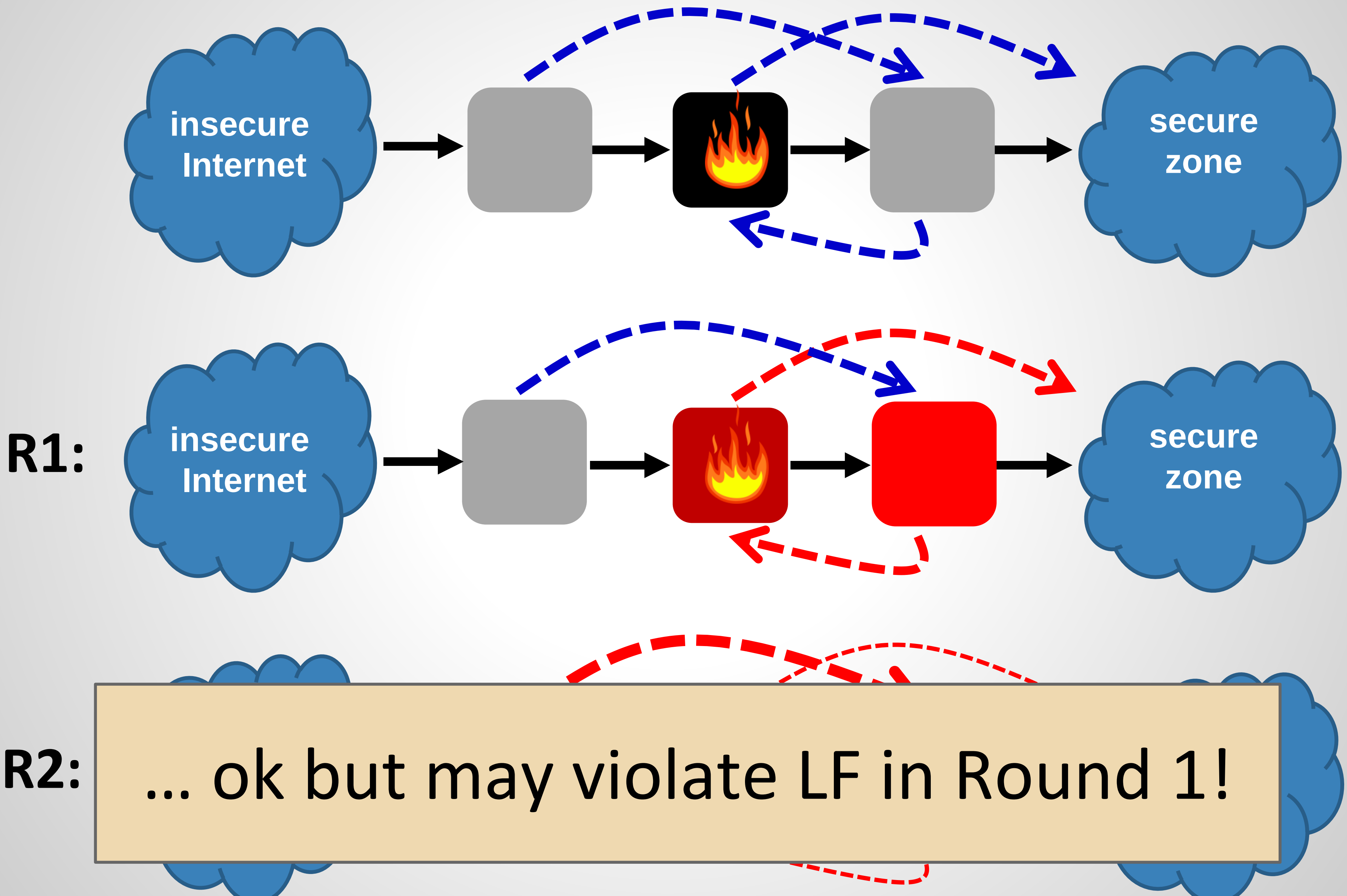
Going Back to Our Examples: WPE Update?



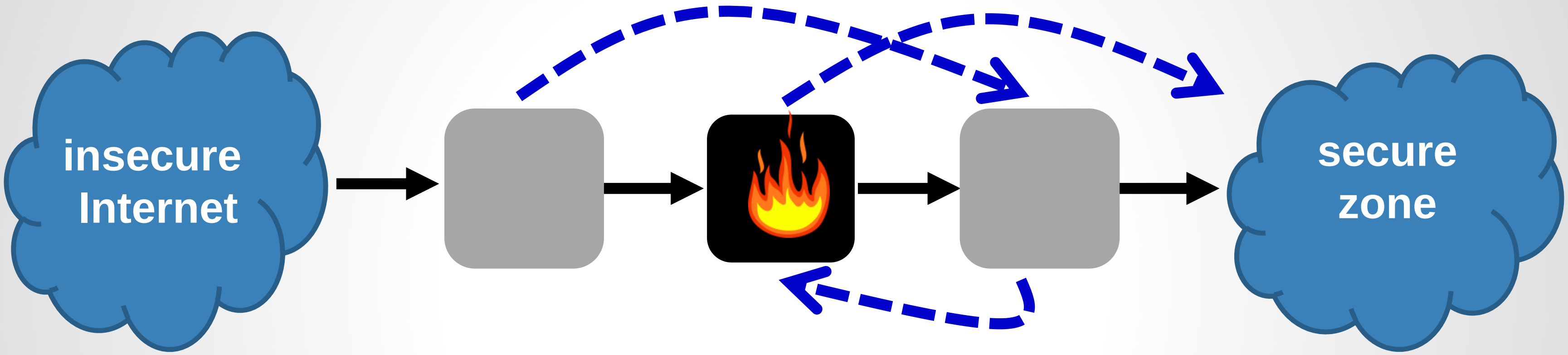
Going Back to Our Examples: WPE Update!



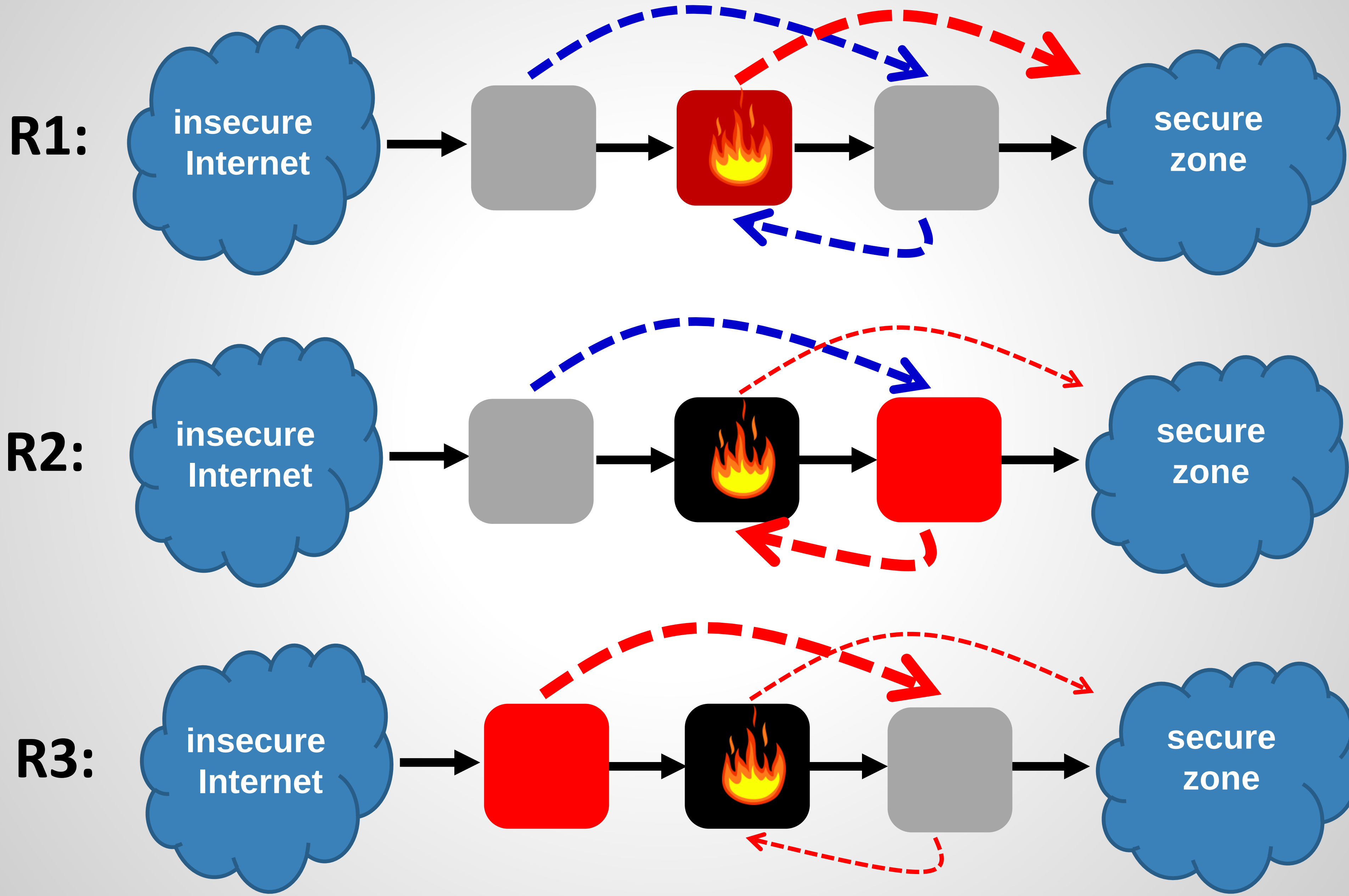
Going Back to Our Examples: WPE Update!



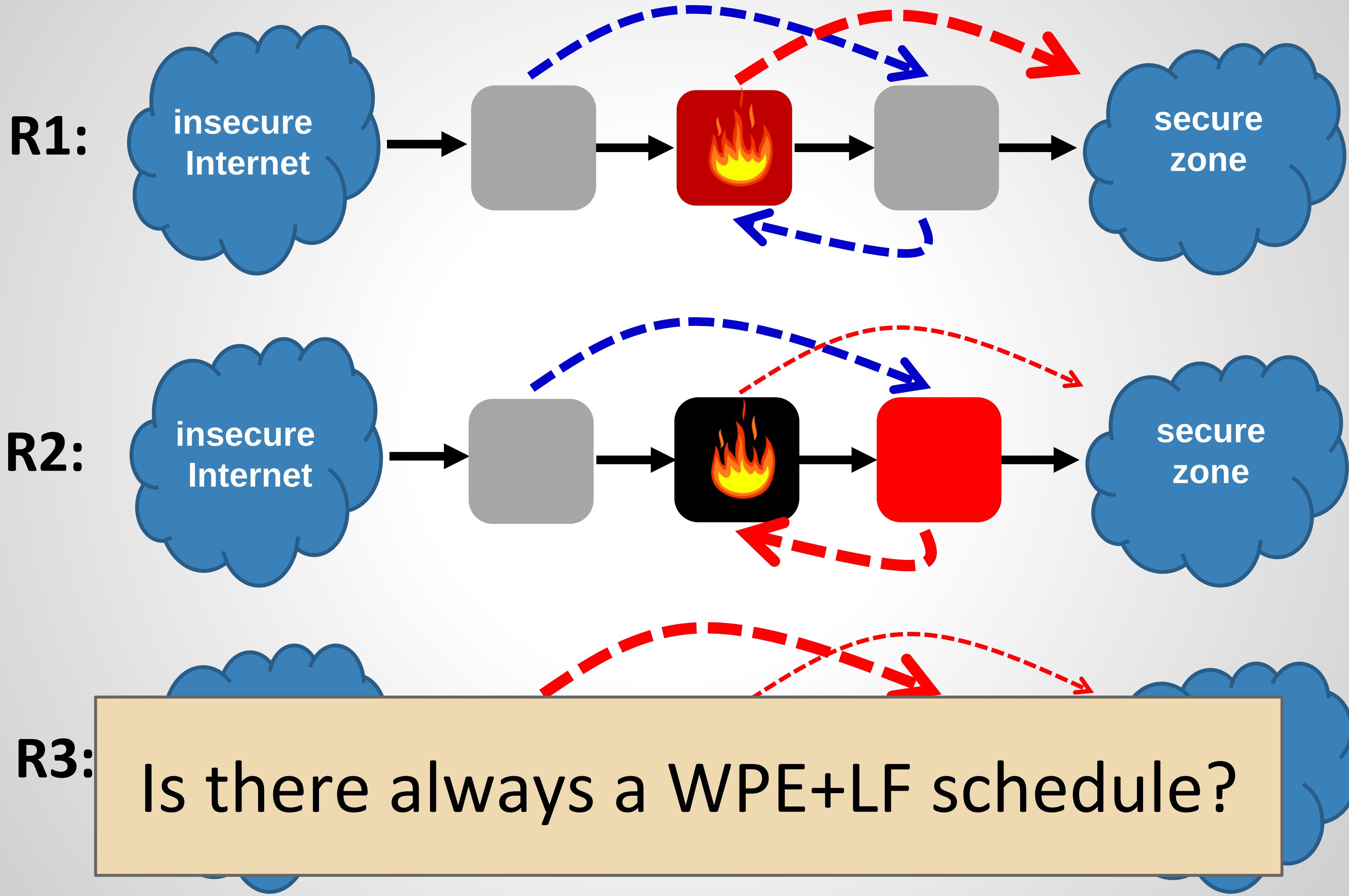
Going Back to Our Examples: Both WPE+LF?



Going Back to Our Examples: WPE+LF!

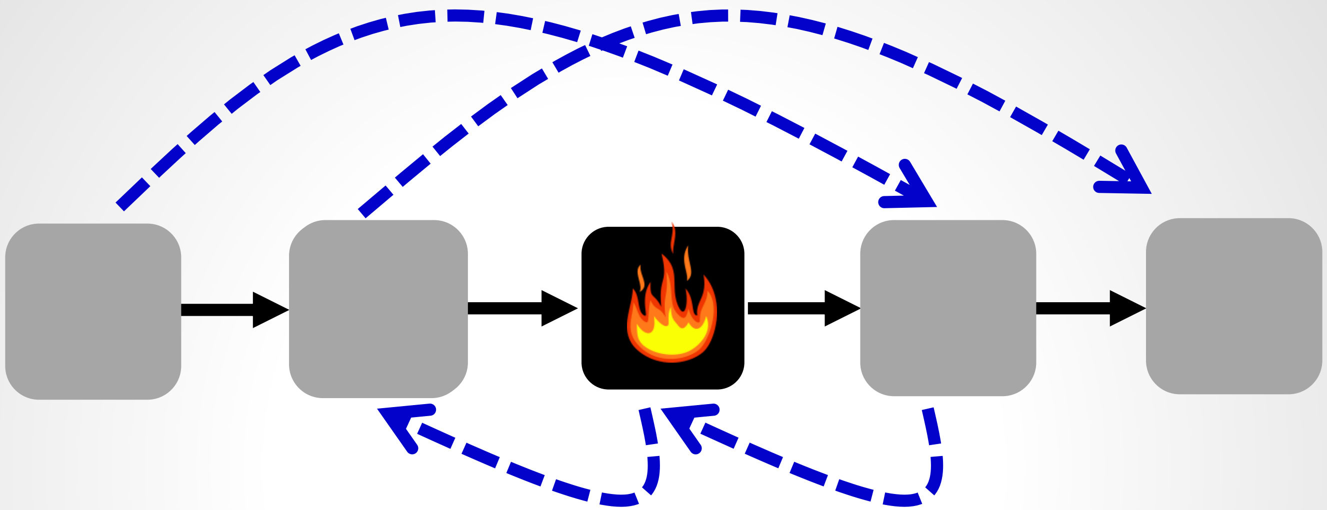


Going Back to Our Examples: WPE+LF!

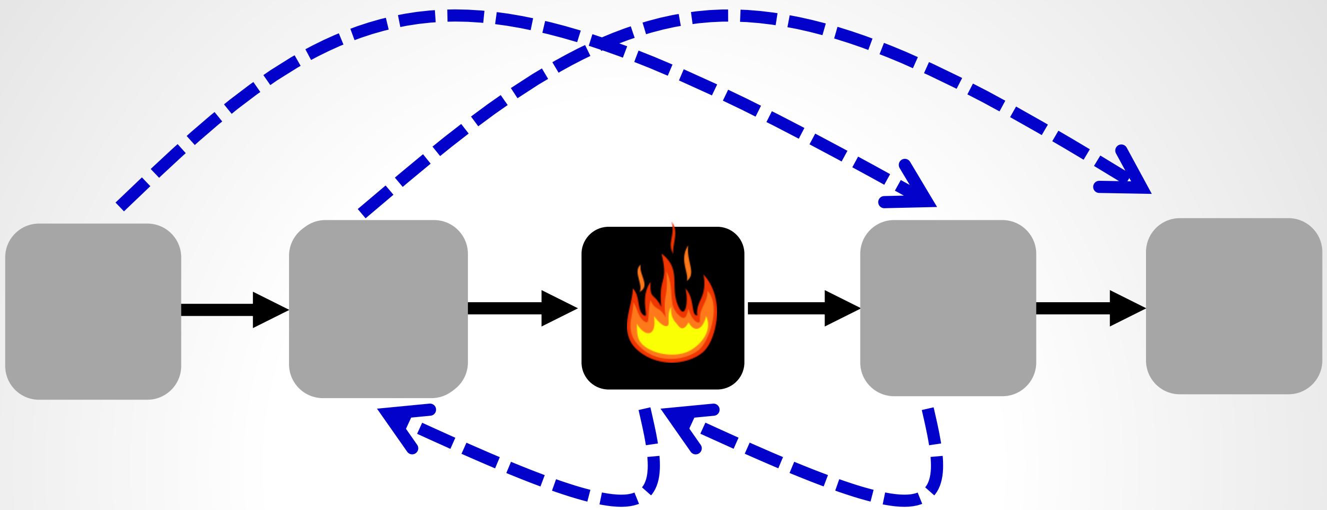


Is there always a WPE+LF schedule?

What about this one?



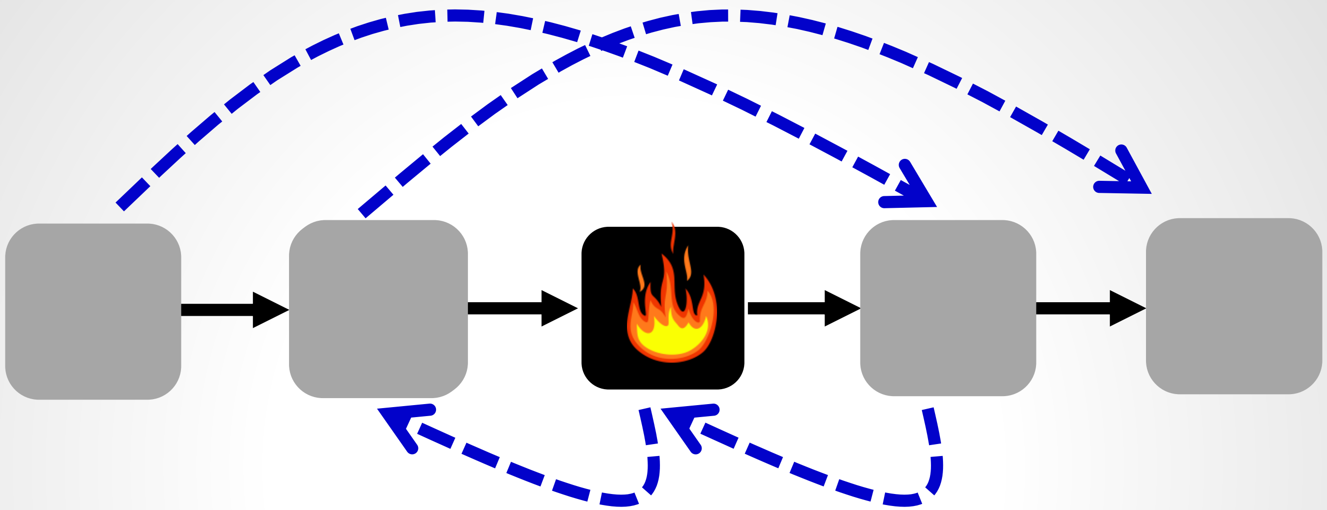
LF and WPE may conflict!



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

No schedule exists!

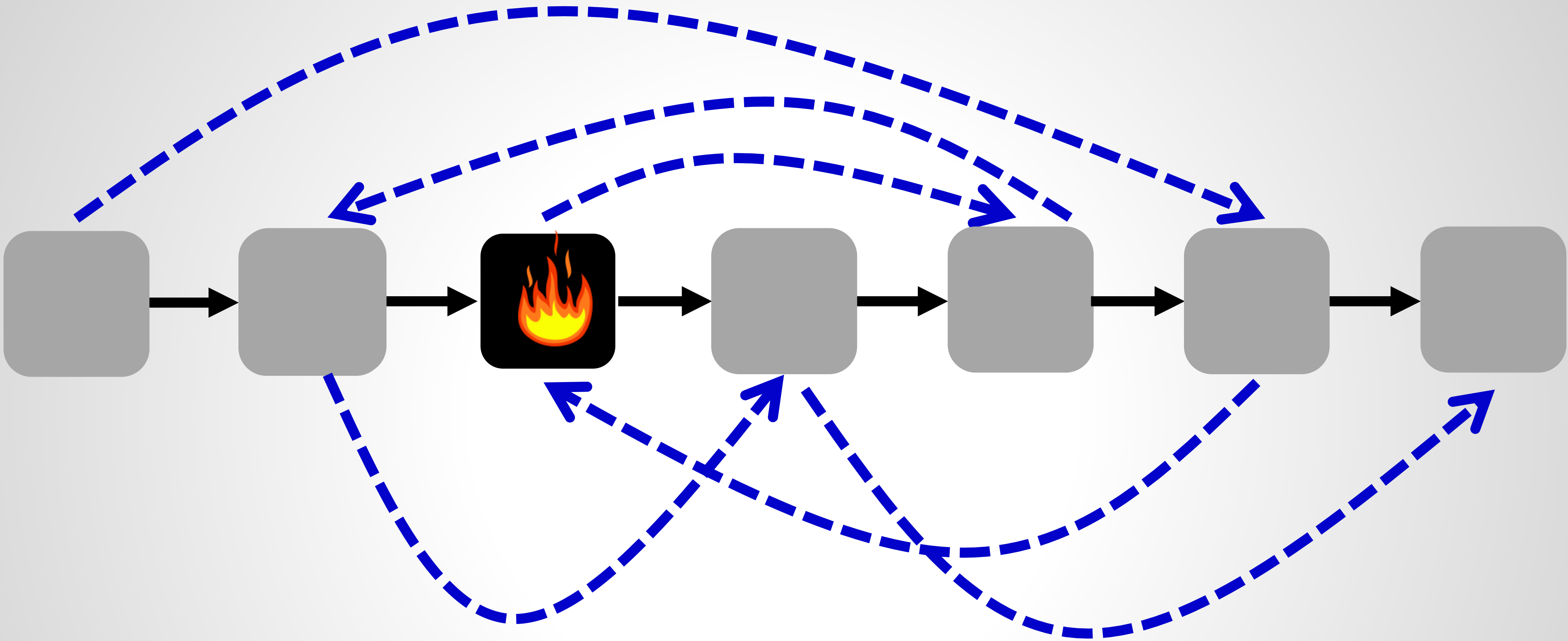
LF and WPE may conflict!



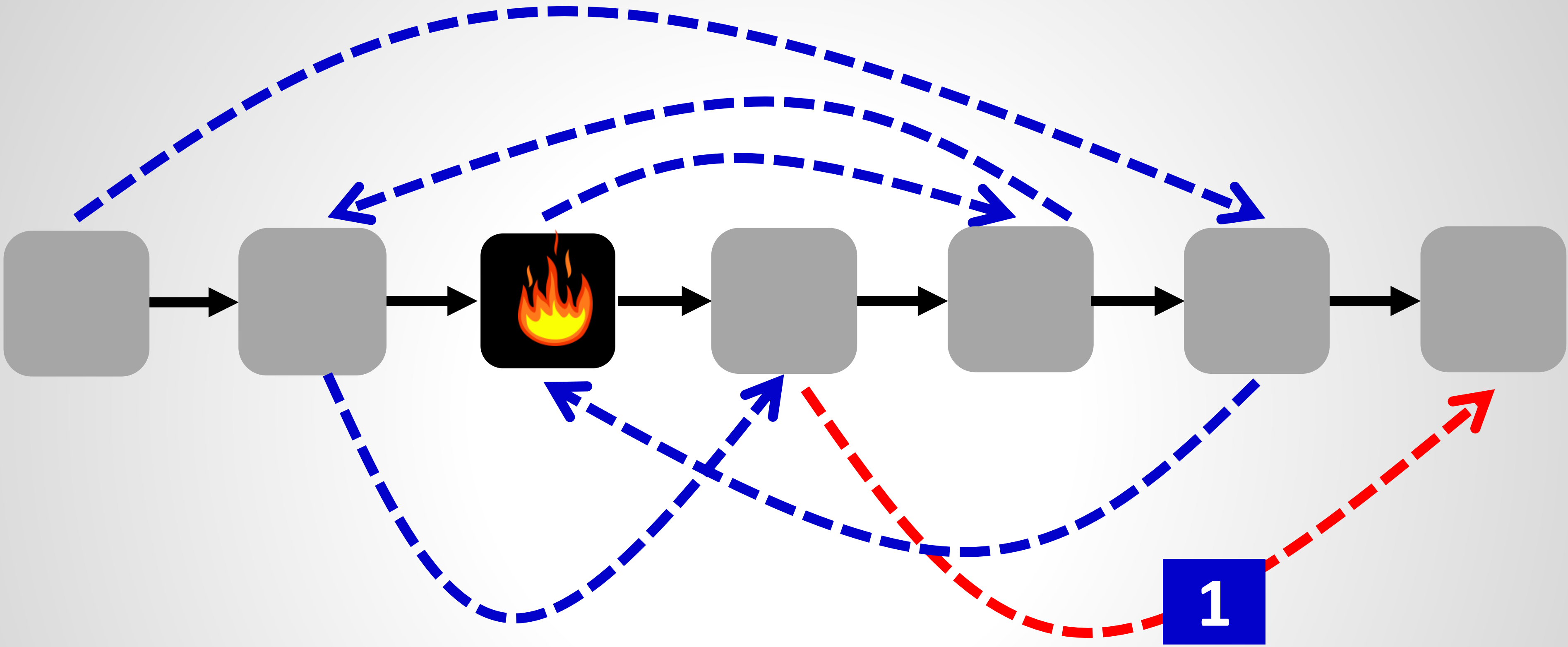
- ❑ Cannot update any forward edge in R1: WP
- ❑ Cannot update any backward edge in R1: LF

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...

How about this one?

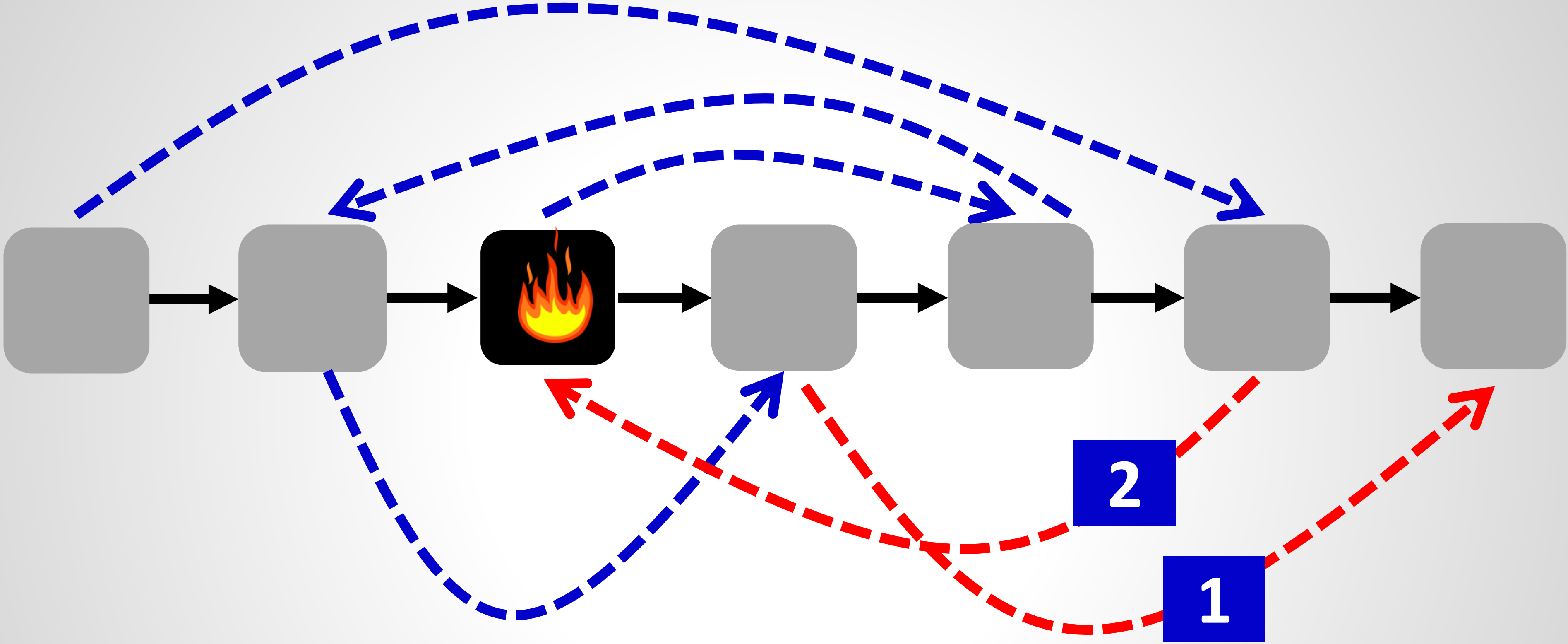


How about this one?



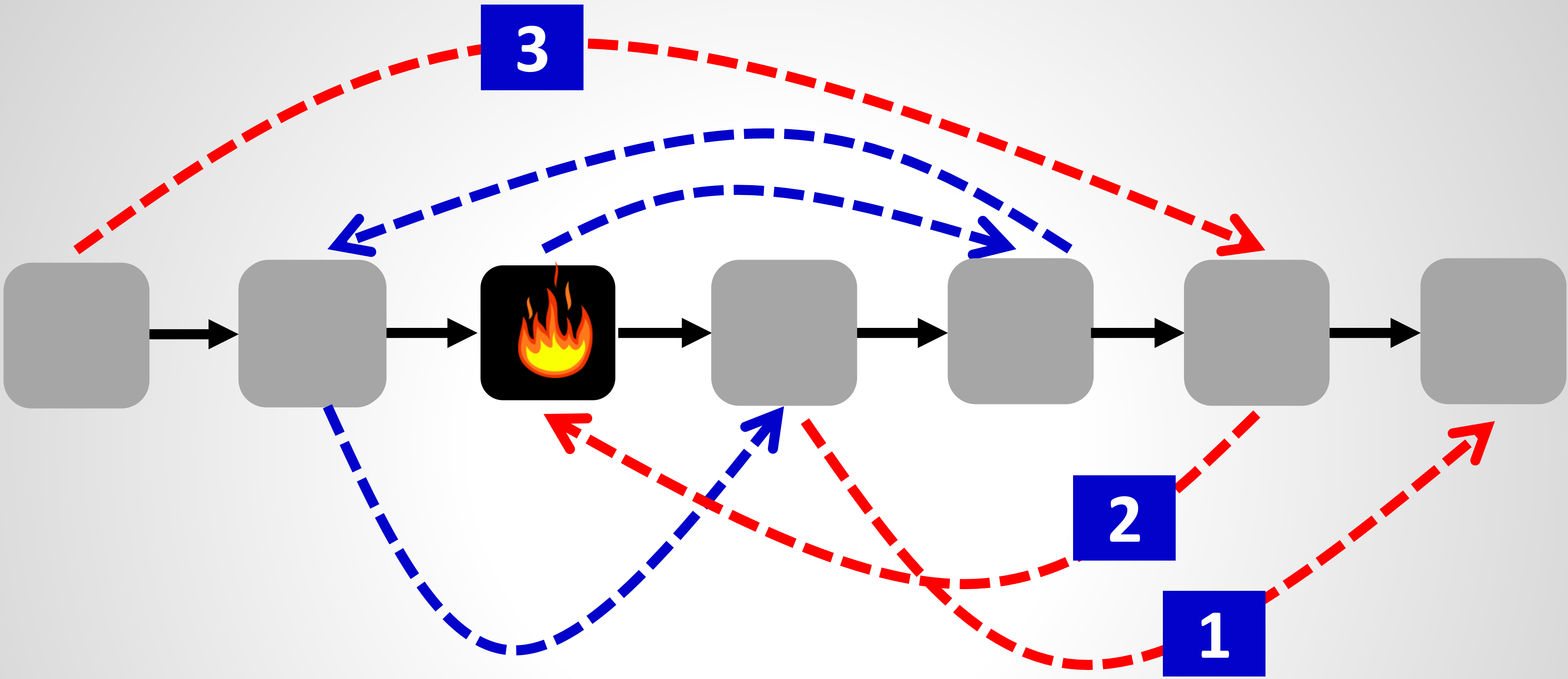
- Forward edge after the waypoint: safe!
  - No loop, no WPE violation

How about this one?



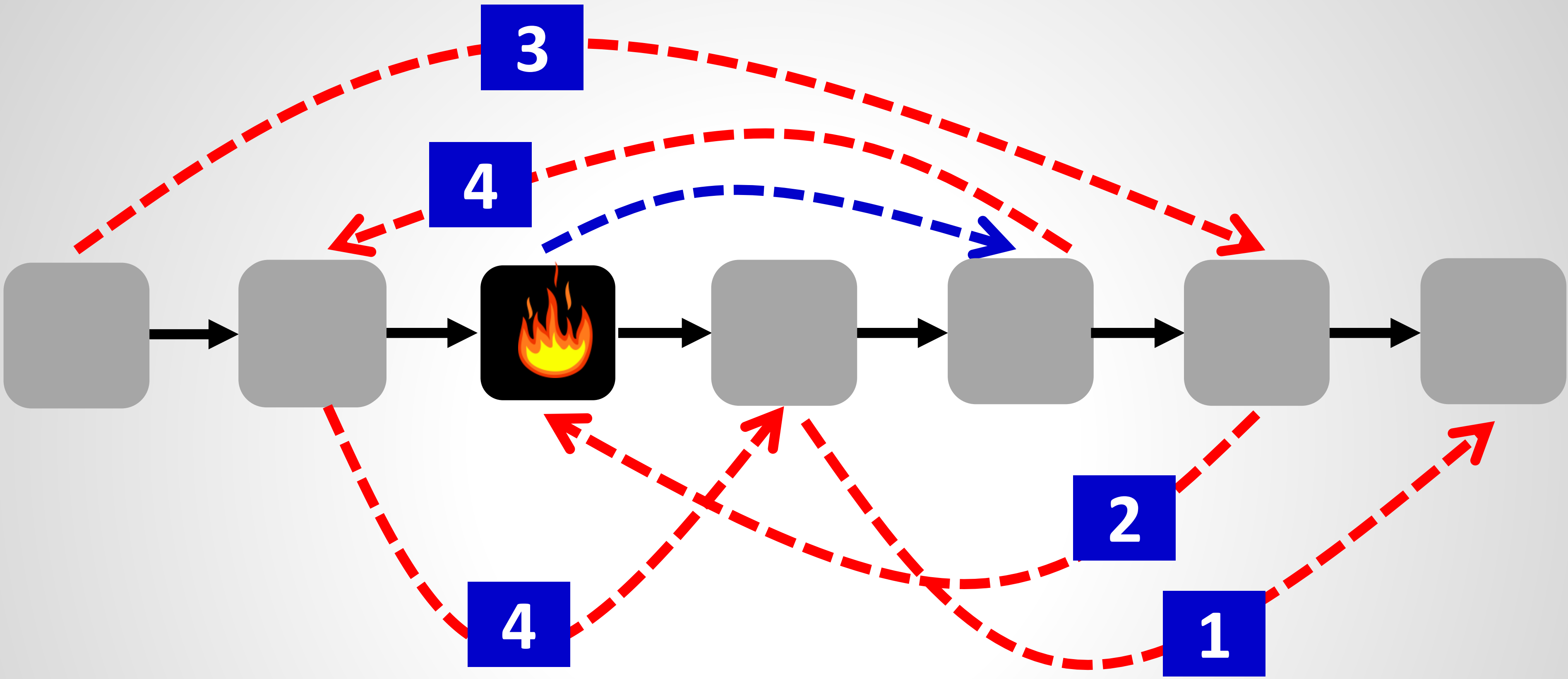
- Now this backward is safe too!
  - No loop because exit through 1

How about this one?



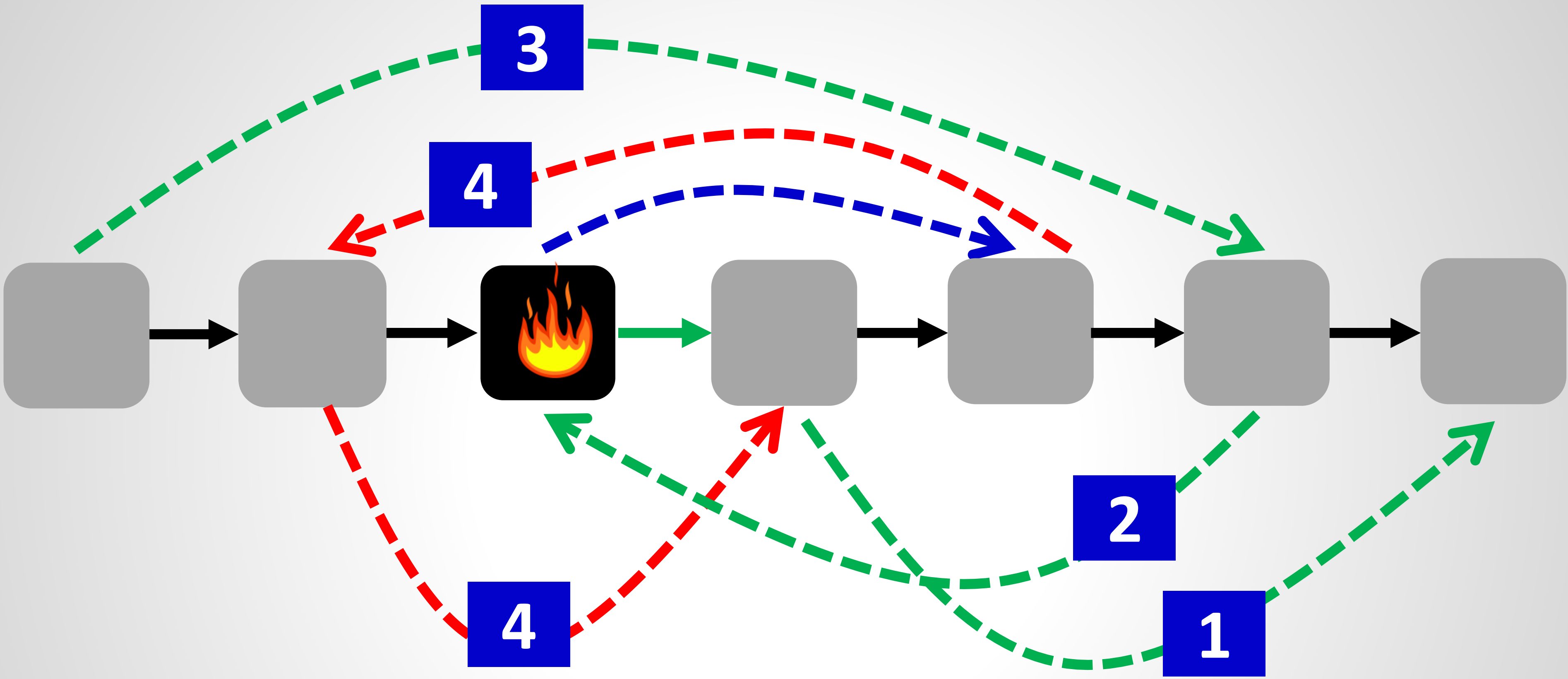
- Now this is safe: $\boxed{2}$ ready back to WP!
  - No waypoint violation

How about this one?



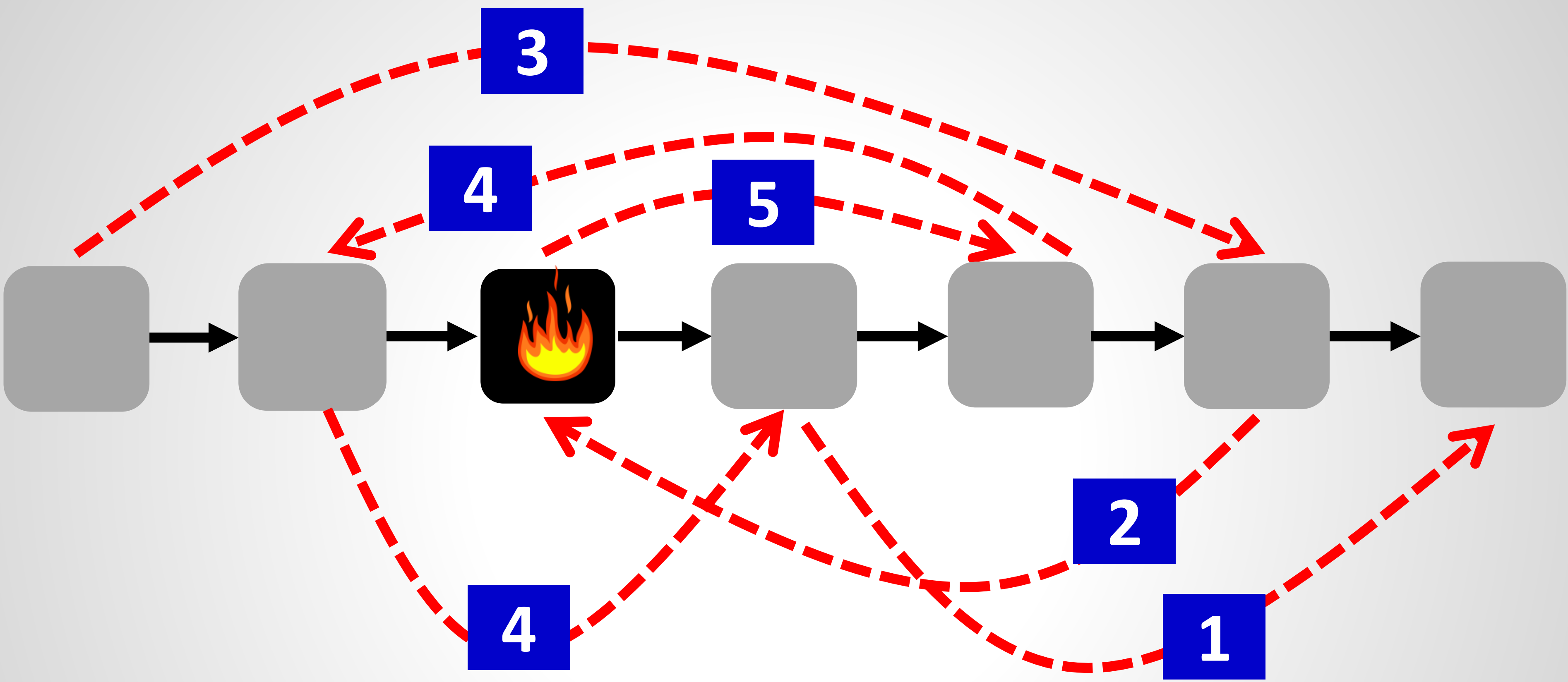
❑ Ok to update as not on the path (goes to d via 1)

How about this one?



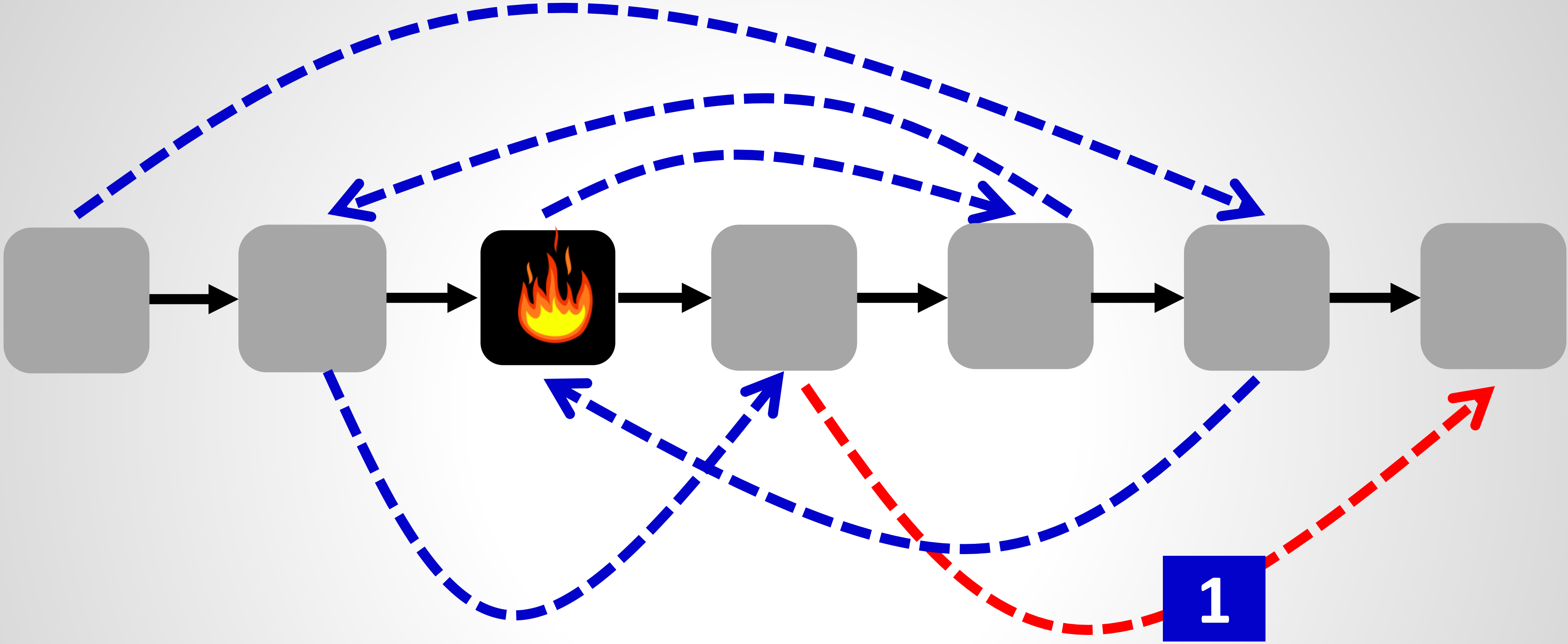
- ❑ Ok to update as not on the path (goes to d via 1)

How about this one?

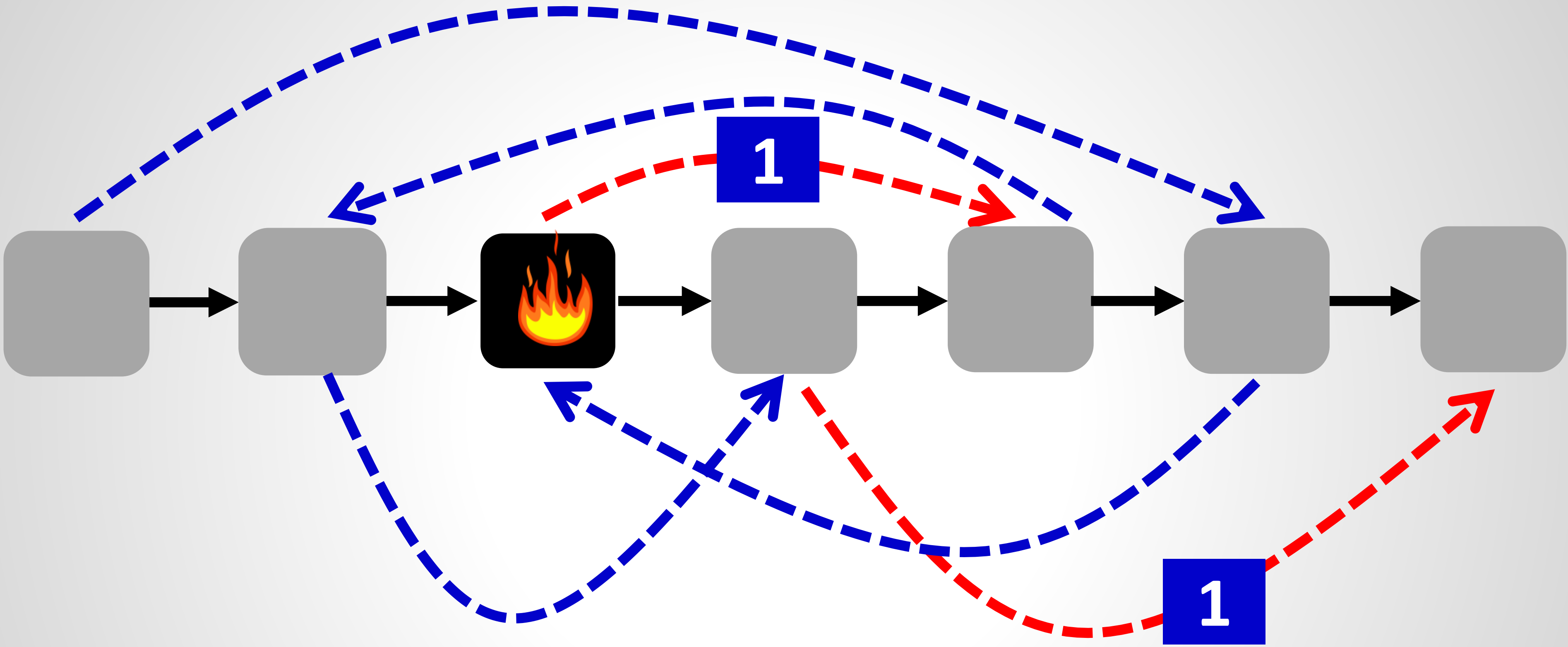


- Ok to update as not on the path (goes to d via 1)

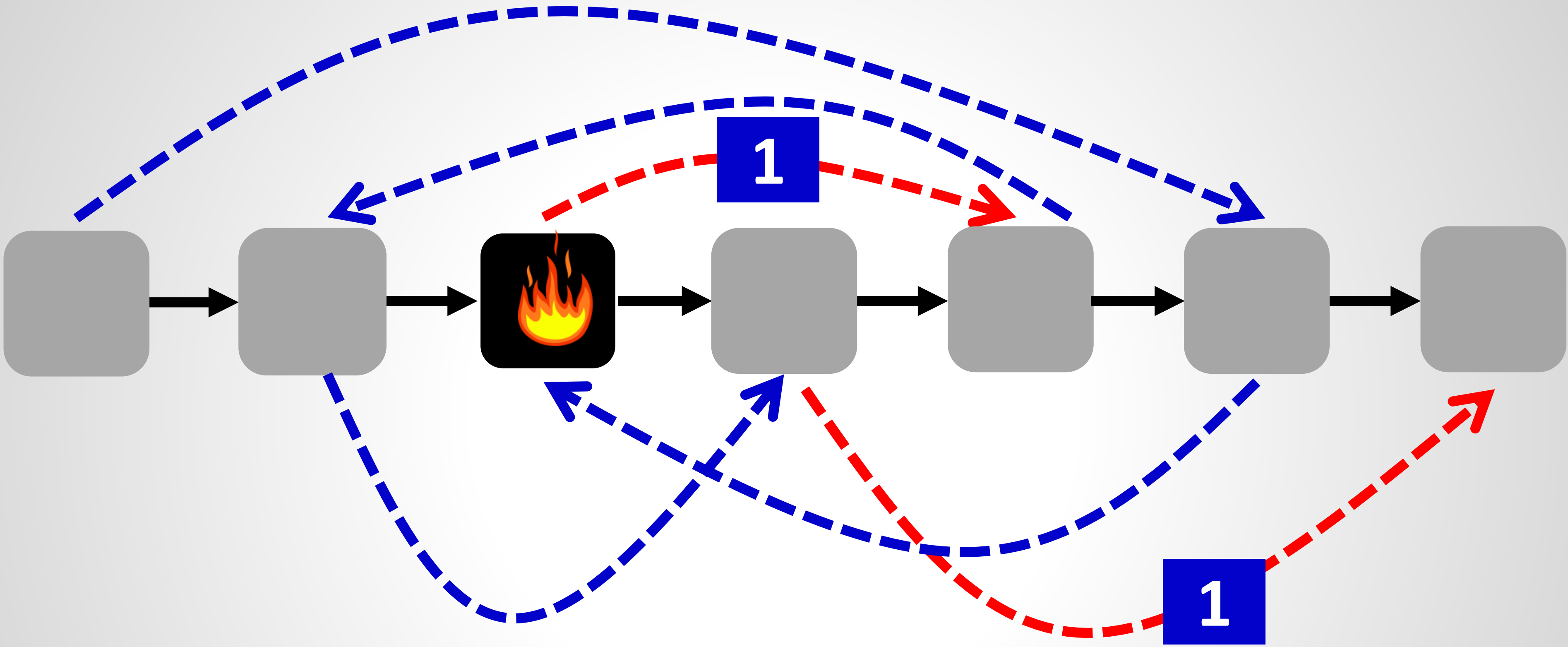
Back to the start: What if….



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

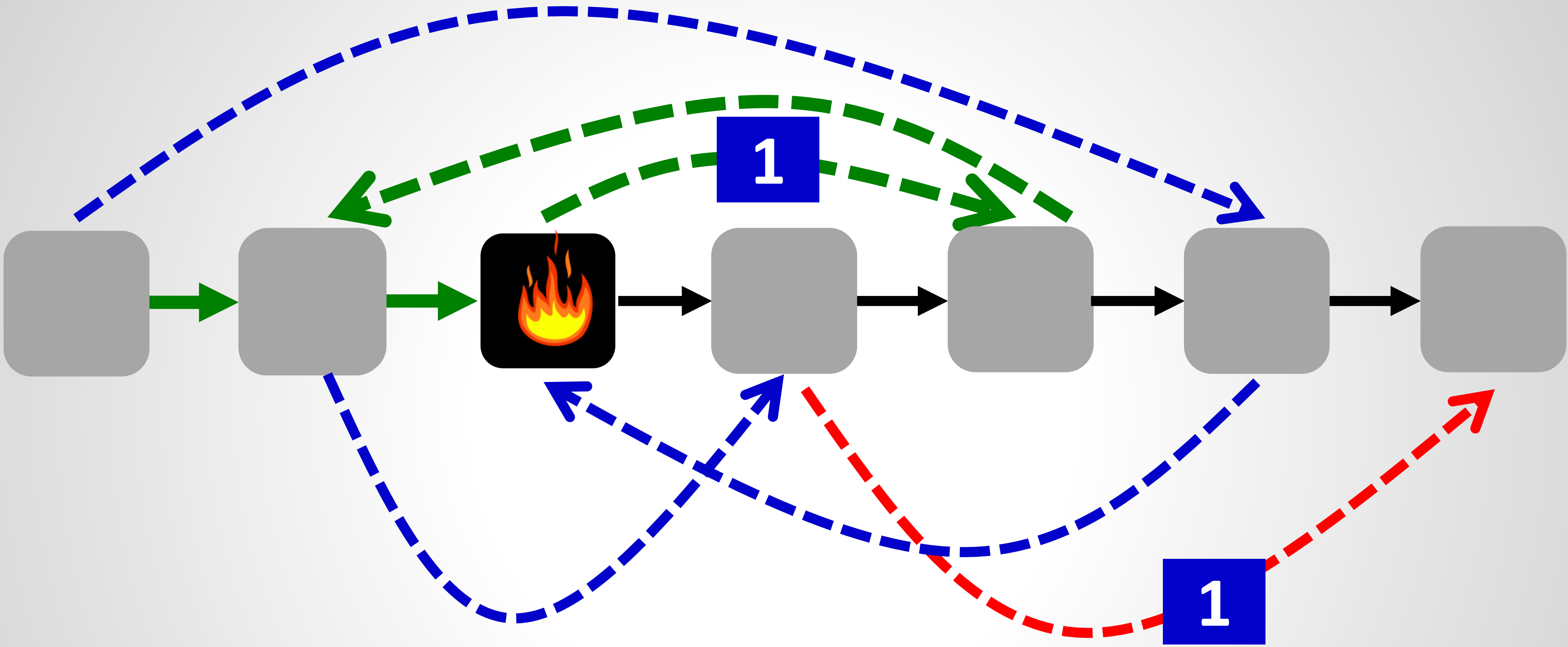


Back to the start: What if…. also this one?!



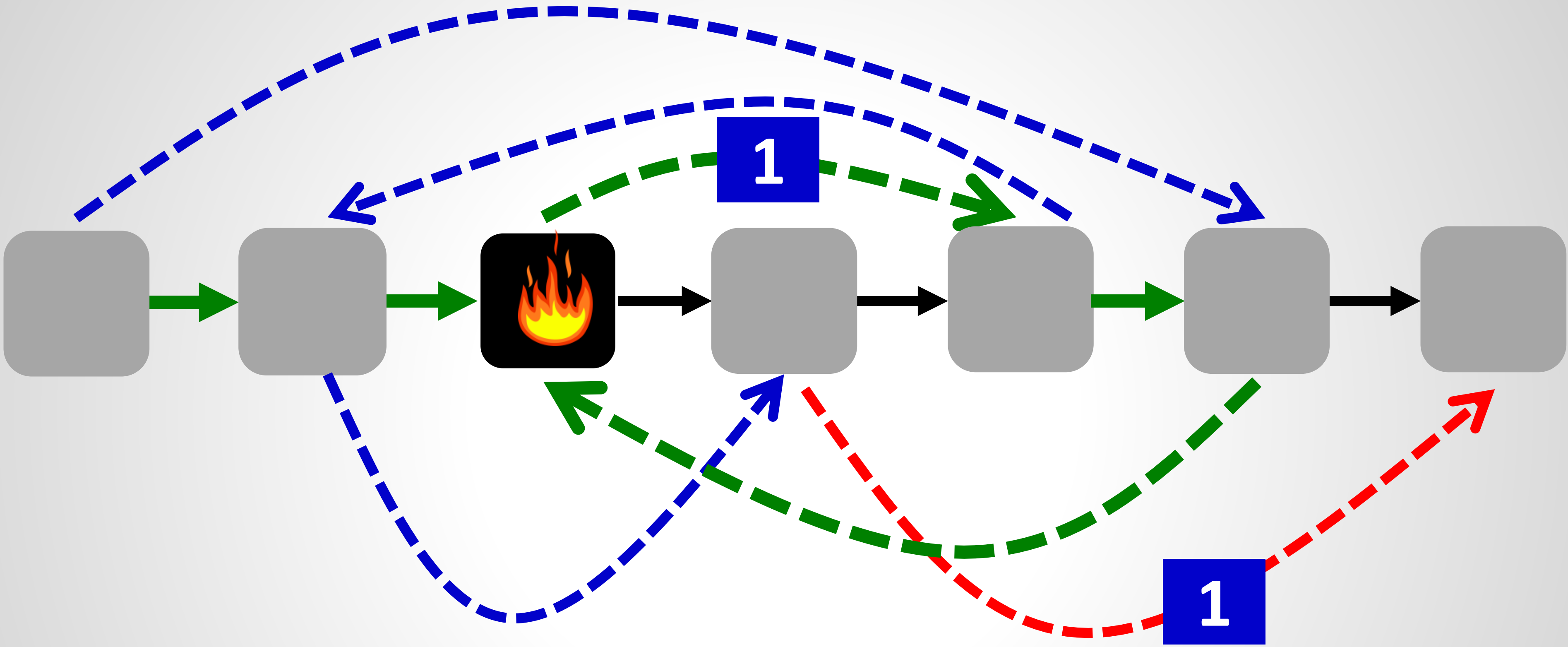
- ❑ Update any of the 2 backward edges? LF ☹

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



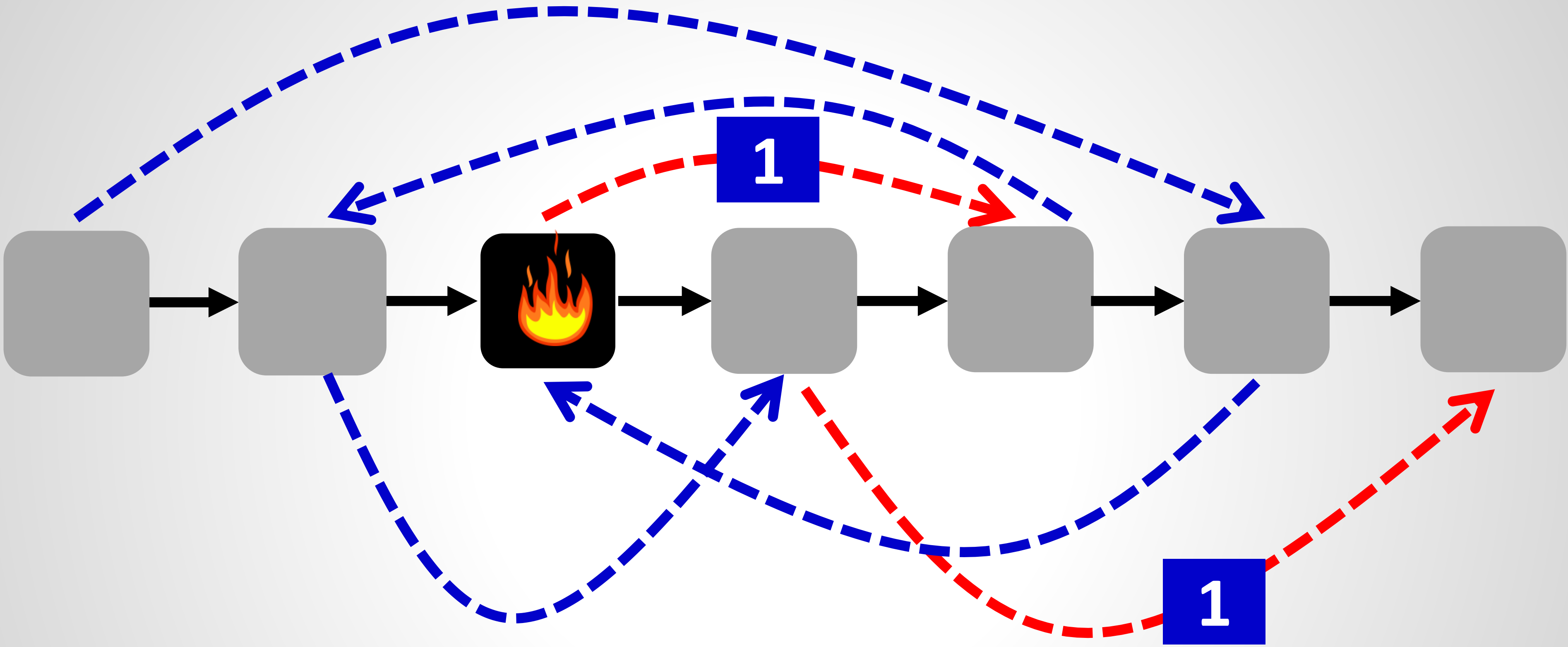
❑ Update any of the 2 backward edges? LF ☹

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



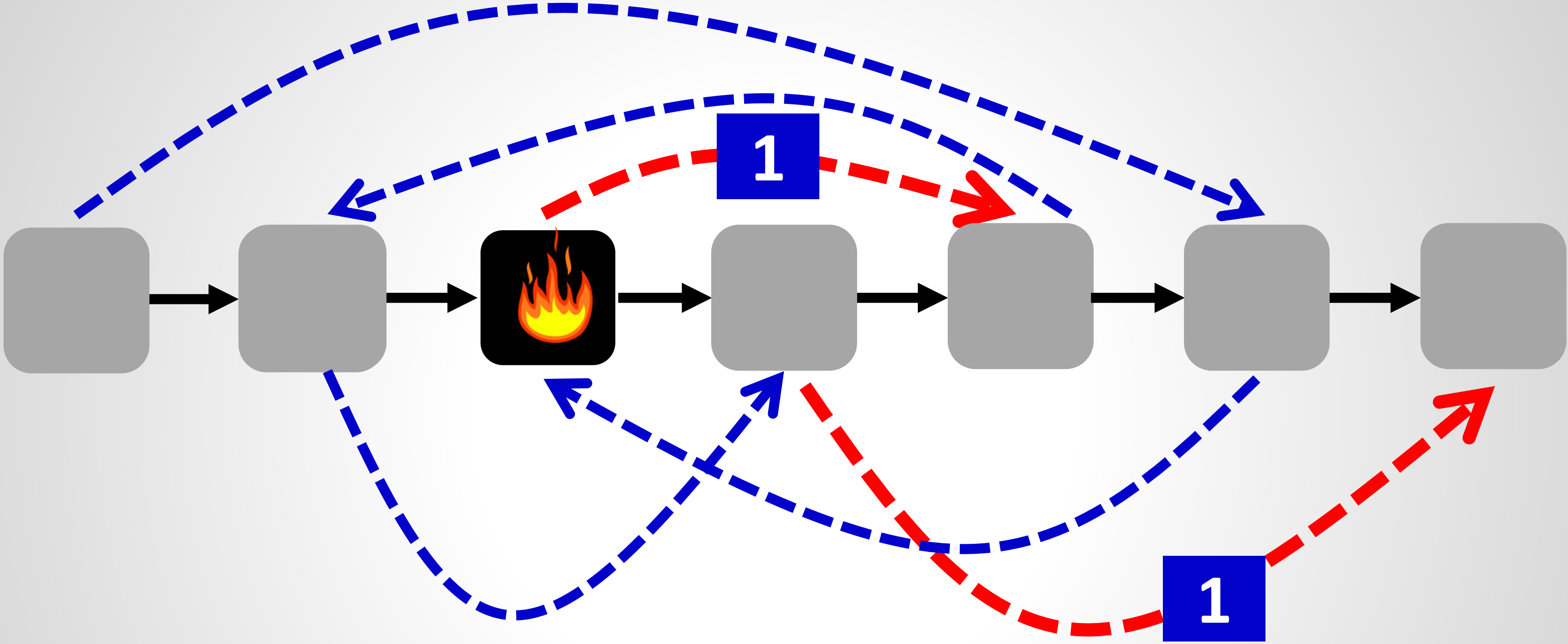
- ❑ Update any of the 2 backward edges? LF ☹

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

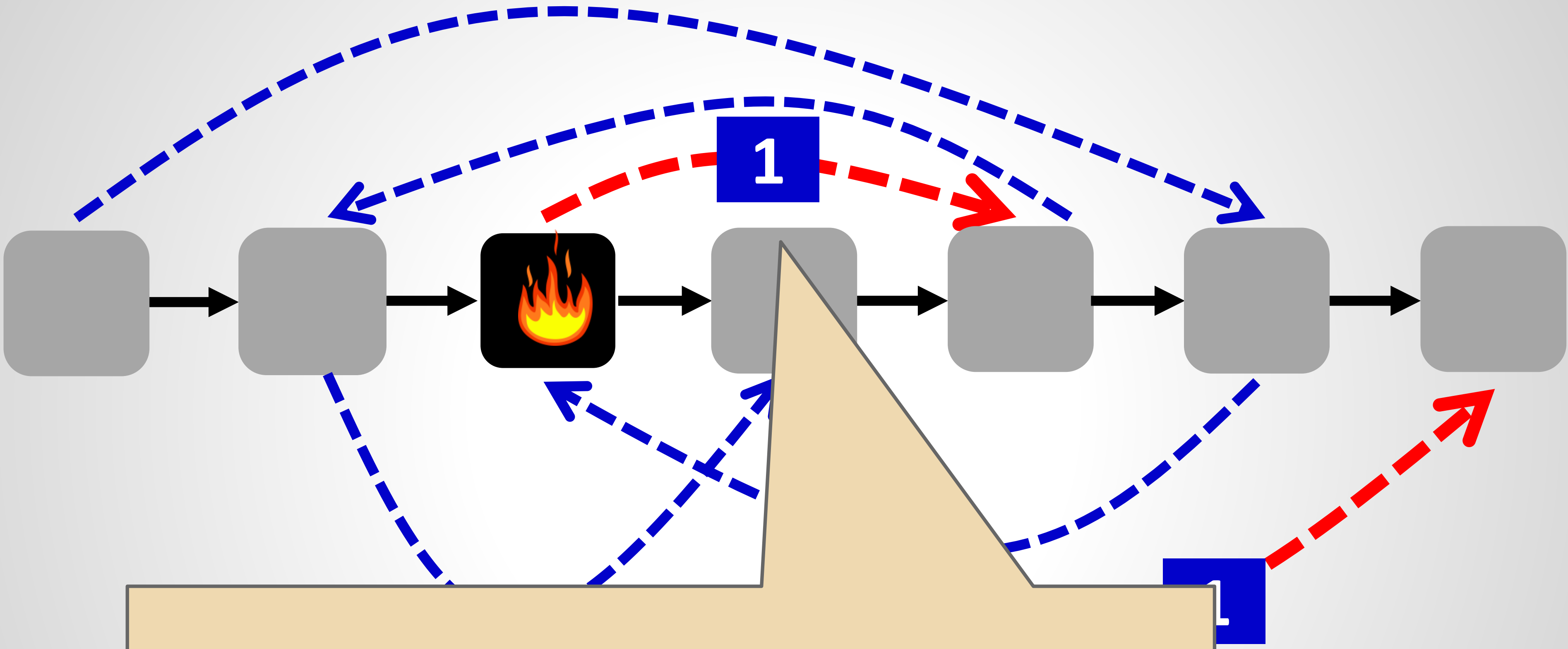


- ❑ Update any of the 2 backward edges? LF ☹
- ❑ Update any of the 2 other forward edges? WPE ☹
- ❑ What about a combination? Nope…

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

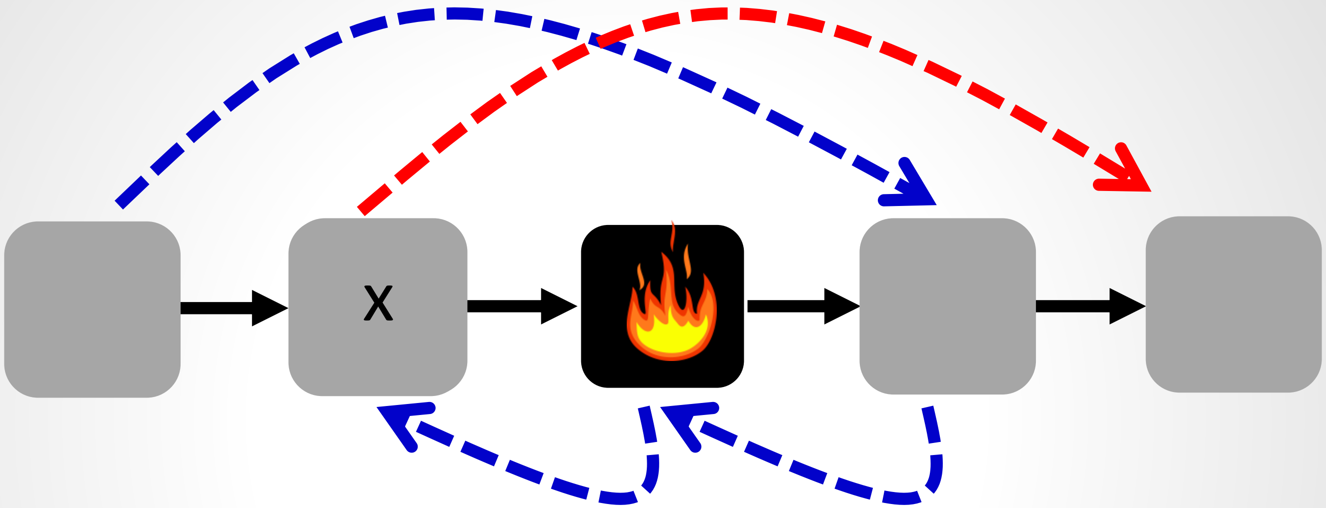


Back to the start: What if…. also this one?!



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

Remark on WPE: ACKs are not enough!



- I may never be able to update this edge!
- Packets may be waiting right before x
- So rounds require waiting (upper bound on latency)

Let‘s forget about Waypoint Enforcement for a moment: Then loop-free update schedules always exist!

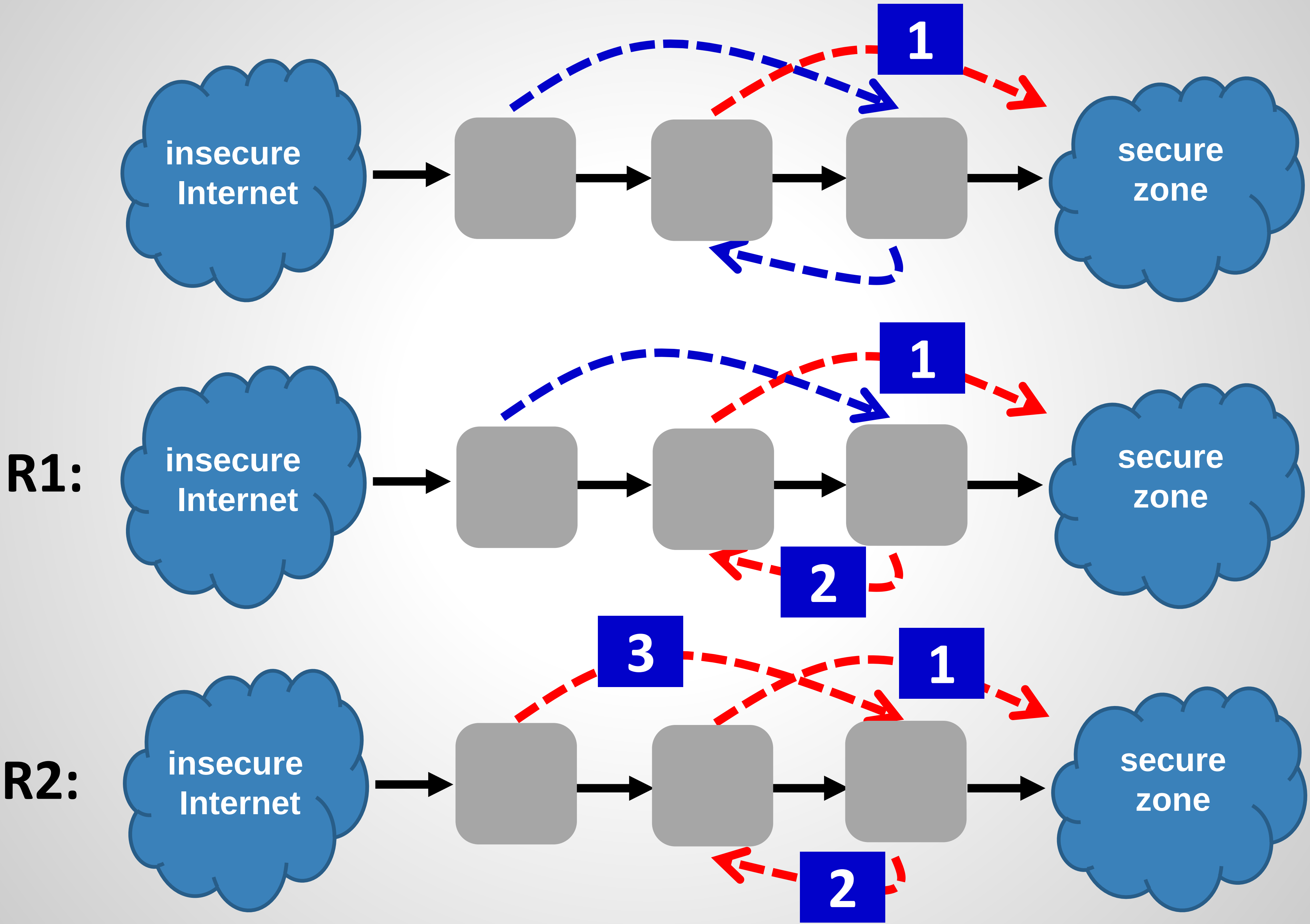
Let‘s forget about Waypoint Enforcement for a moment: Then loop-free update schedules always exist!

Why? Trivial strategy?

Let‘s forget about Waypoint Enforcement for a moment: Then loop-free update schedules always exist!

Why? Trivial strategy? E.g., start from end?

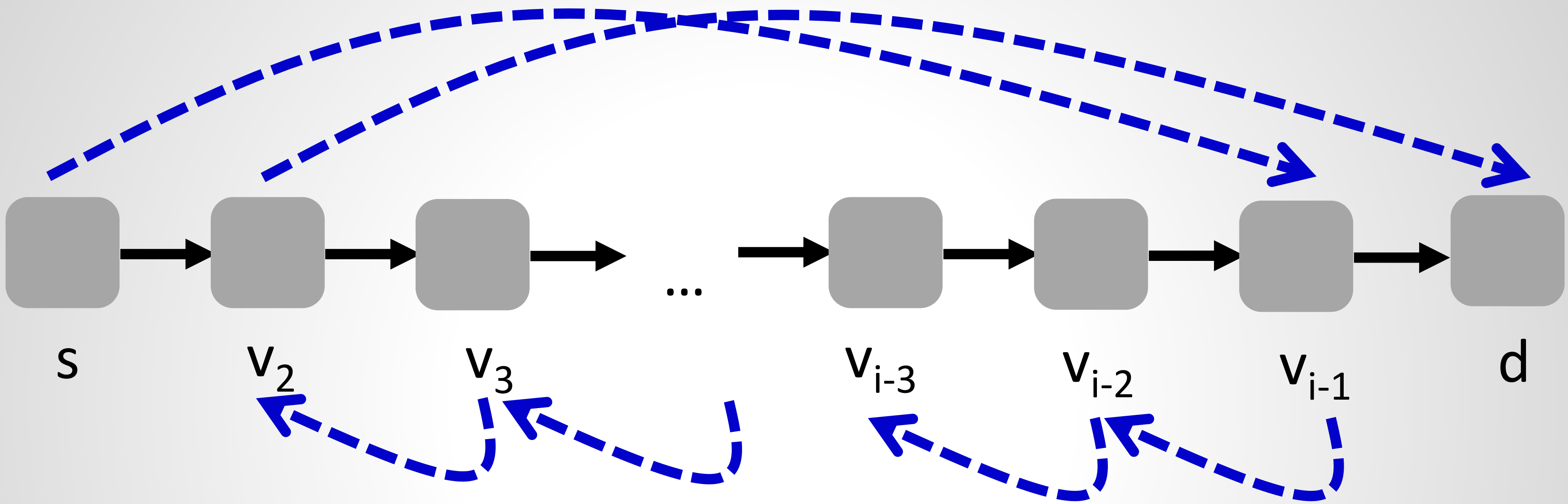
LF Update: Start from end...



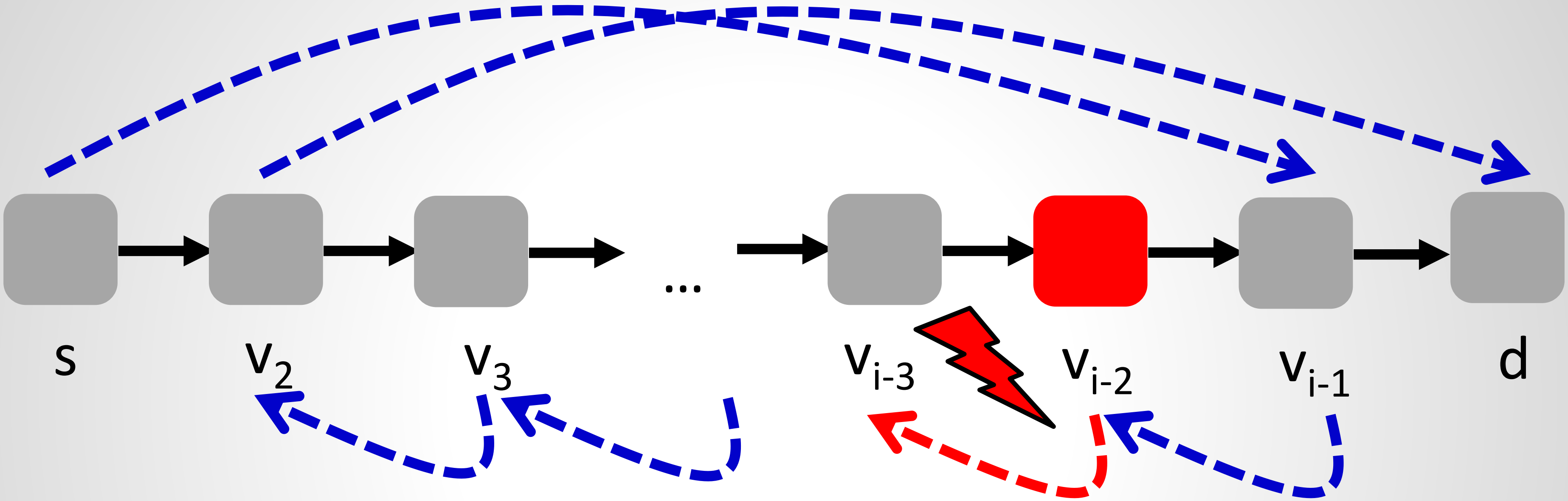
Let‘s forget about Waypoint Enforcement for a moment:
Then loop-free update schedules always exist!

How many rounds are required in the worst case?

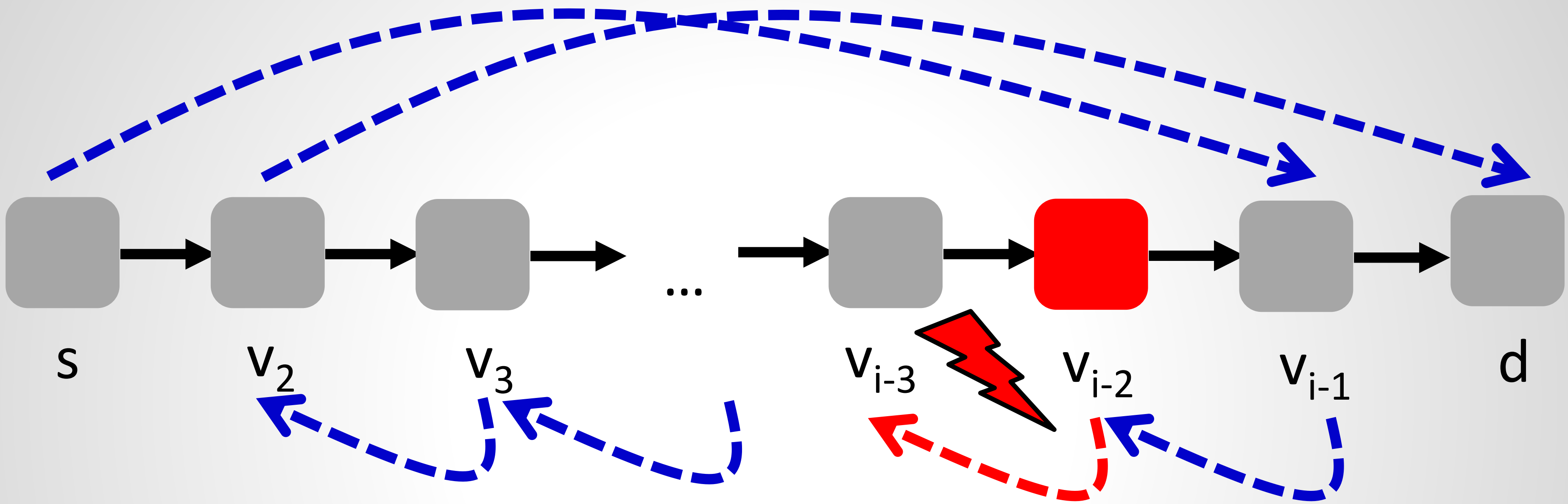
How to update LF?



How to update LF?

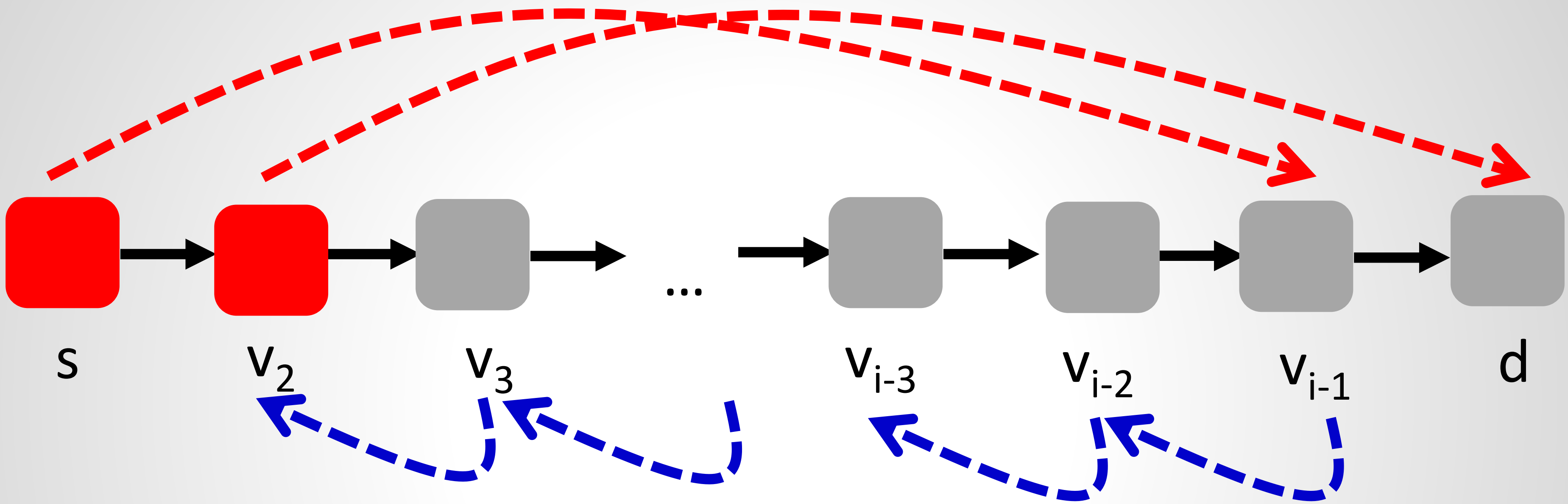


How to update LF?



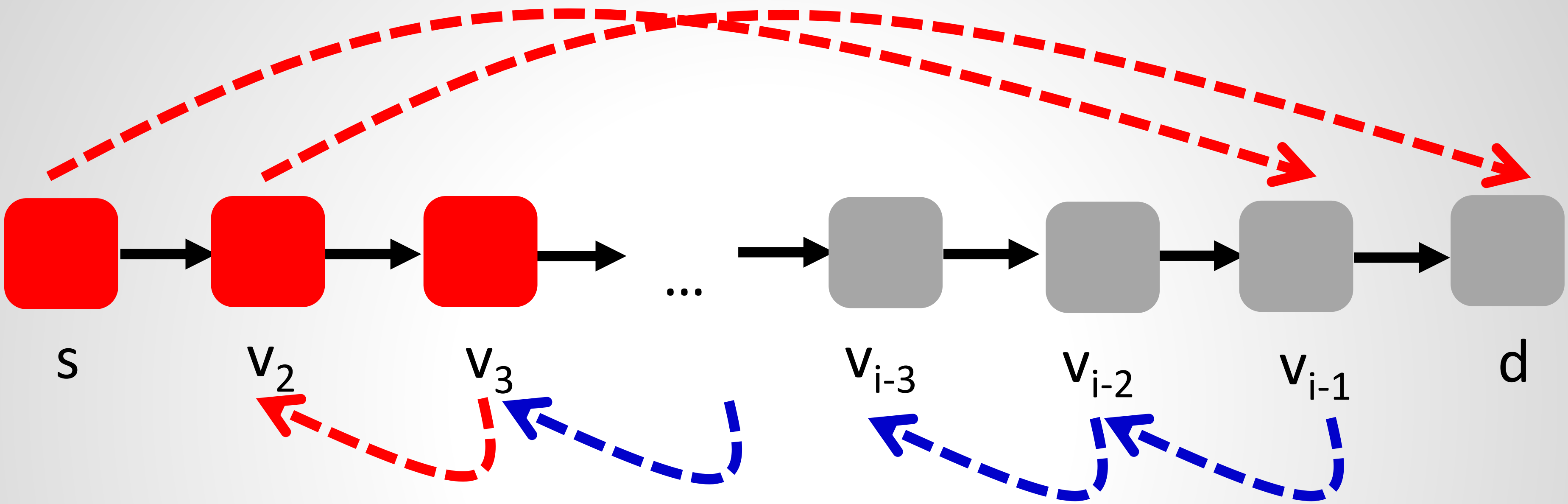
- Must update v_i before v_{i+1}

How to update LF?



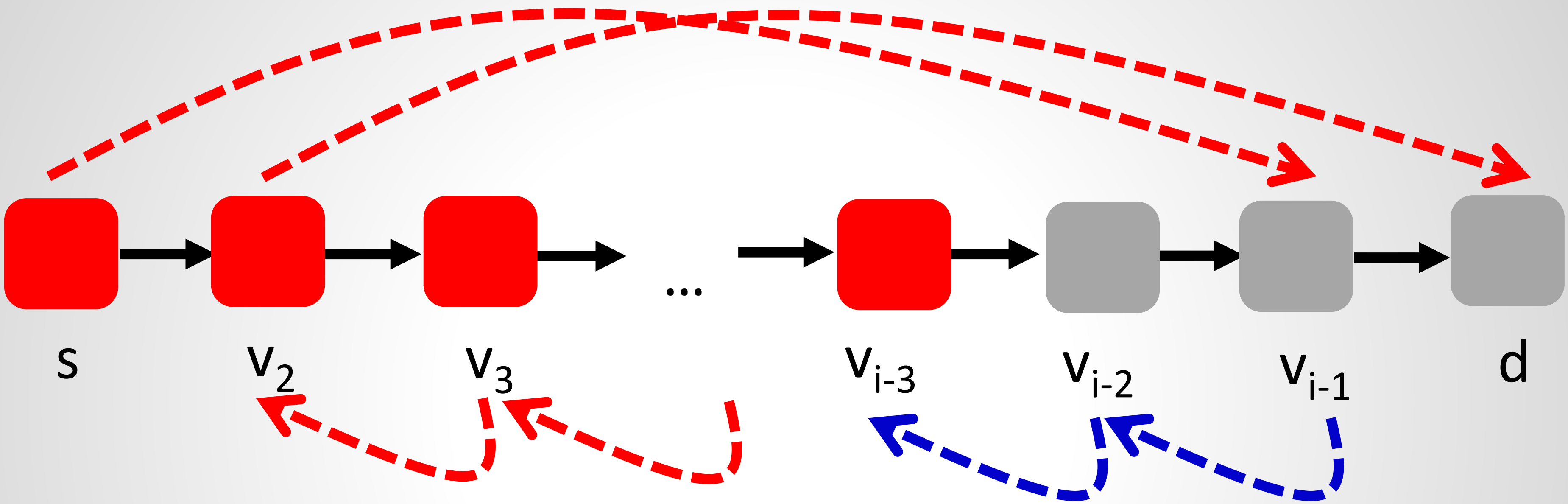
- Must update v_i before v_{i+1}

How to update LF?



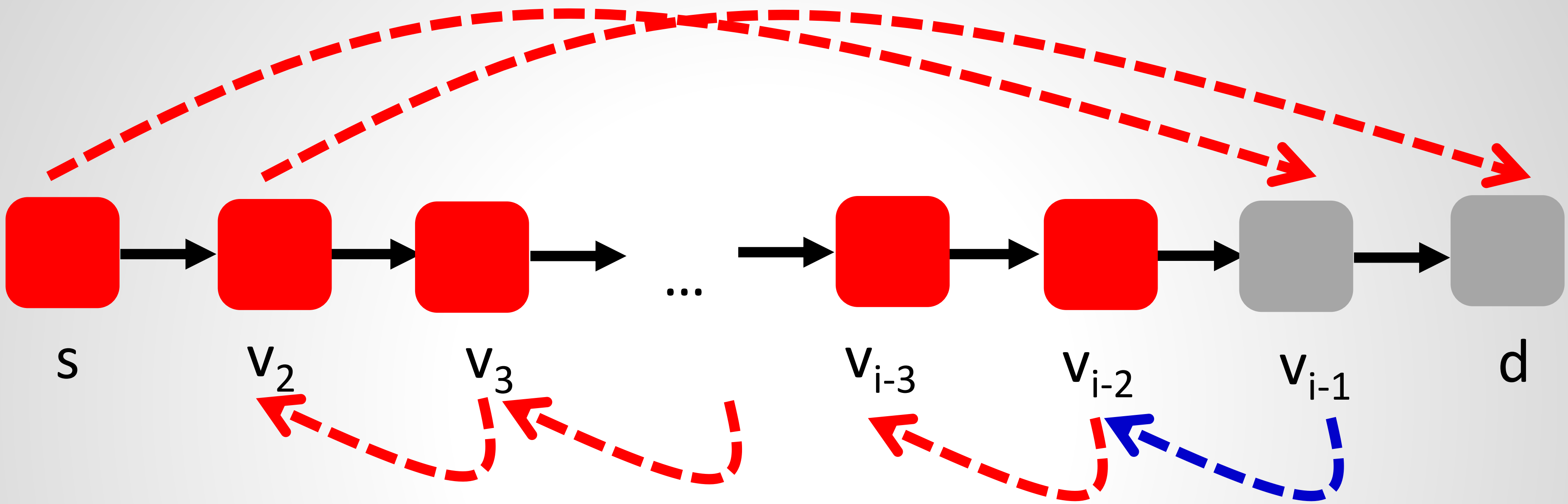
- Must update v_i before v_{i+1}

How to update LF?



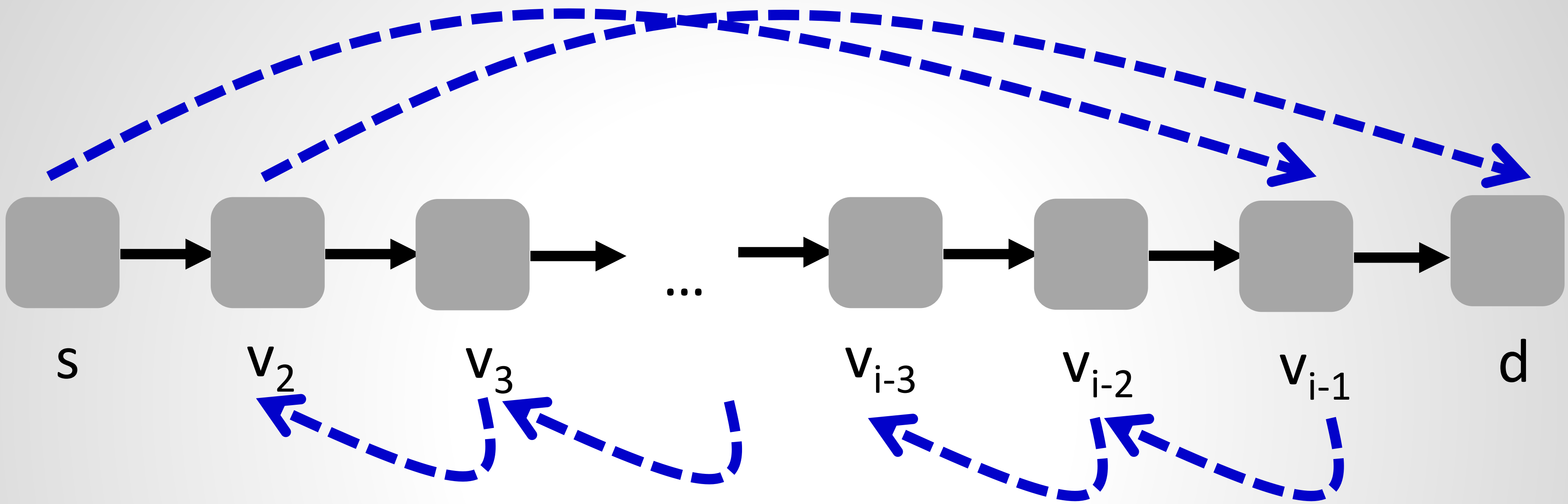
- Must update v_i before v_{i+1}

How to update LF?



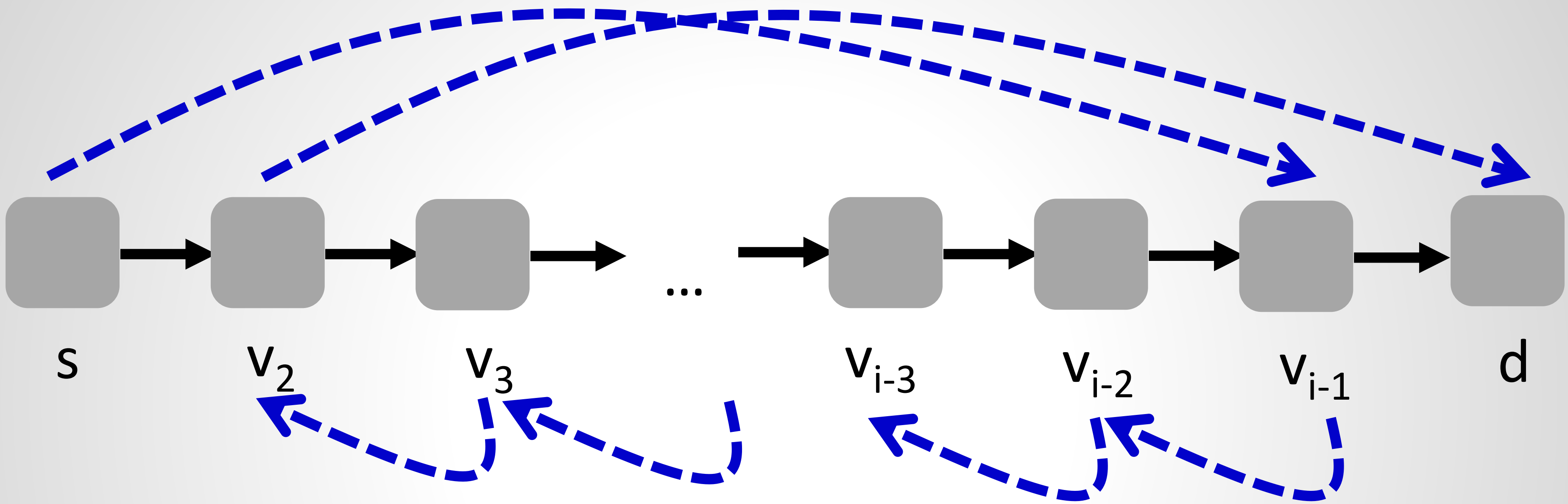
- Must update v_i before v_{i+1}

Ω(n) rounds to be loop-free!



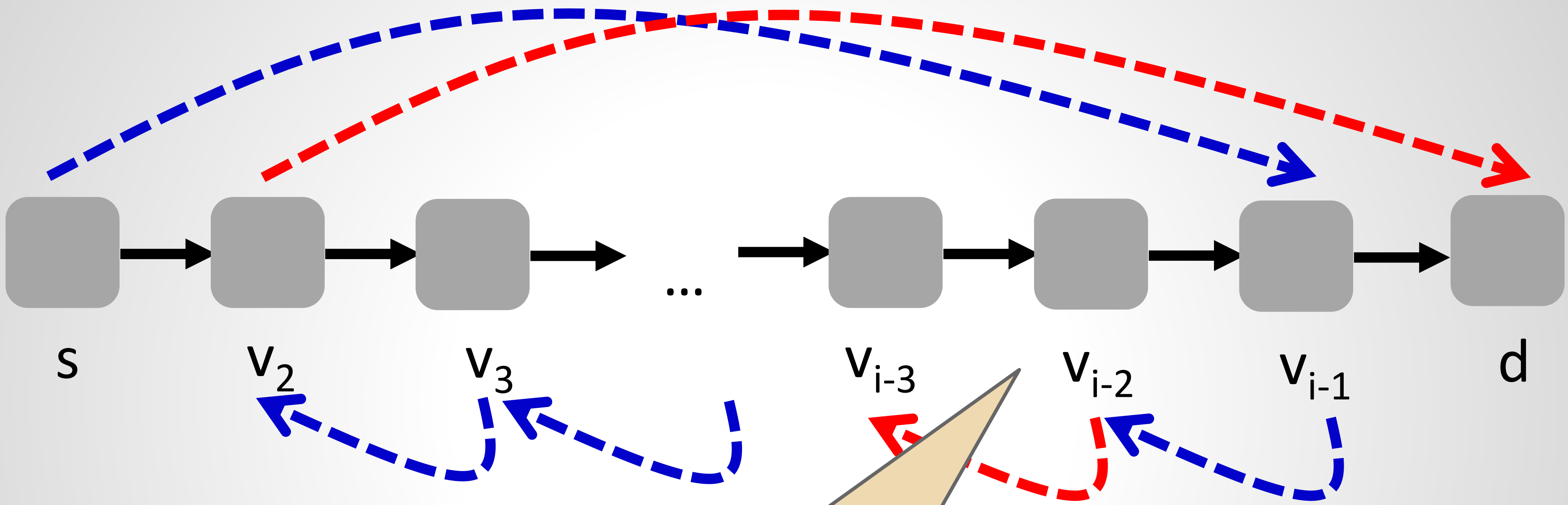
- Must update v_i before v_{i+1}
- Takes Ω(n) rounds: v_3 v_4 v_5 v_6 …

However: It can be good to relax!



- *However:* Topological loops may not be a problem if they do not occur on the active (s,d) path

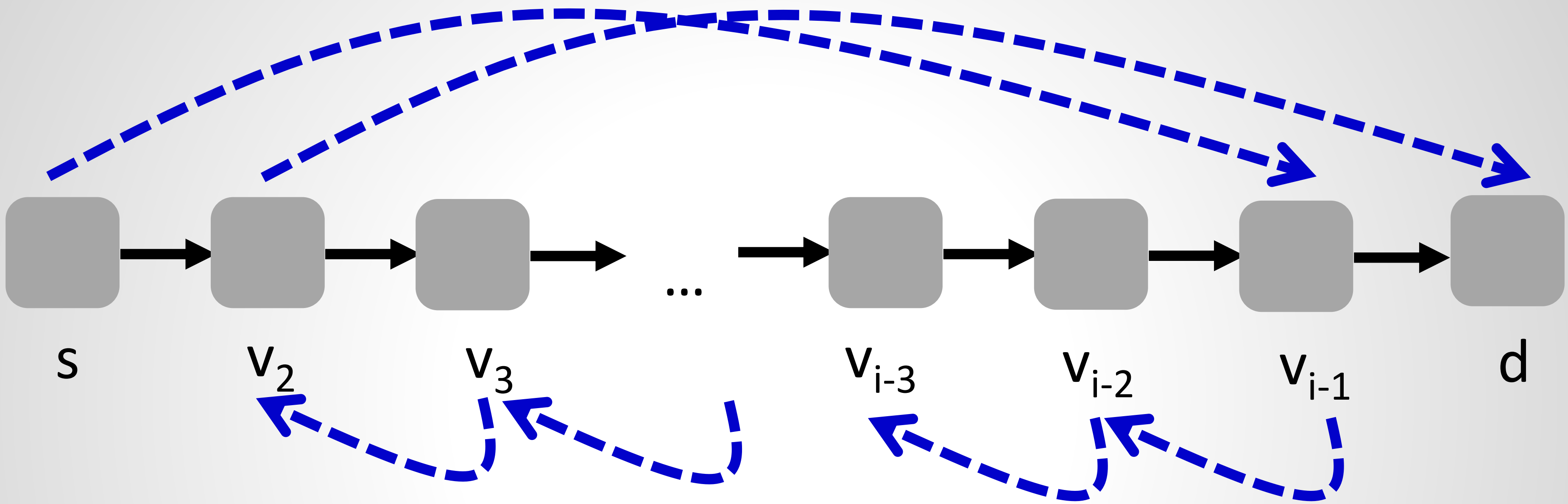
However: It can be good to relax!



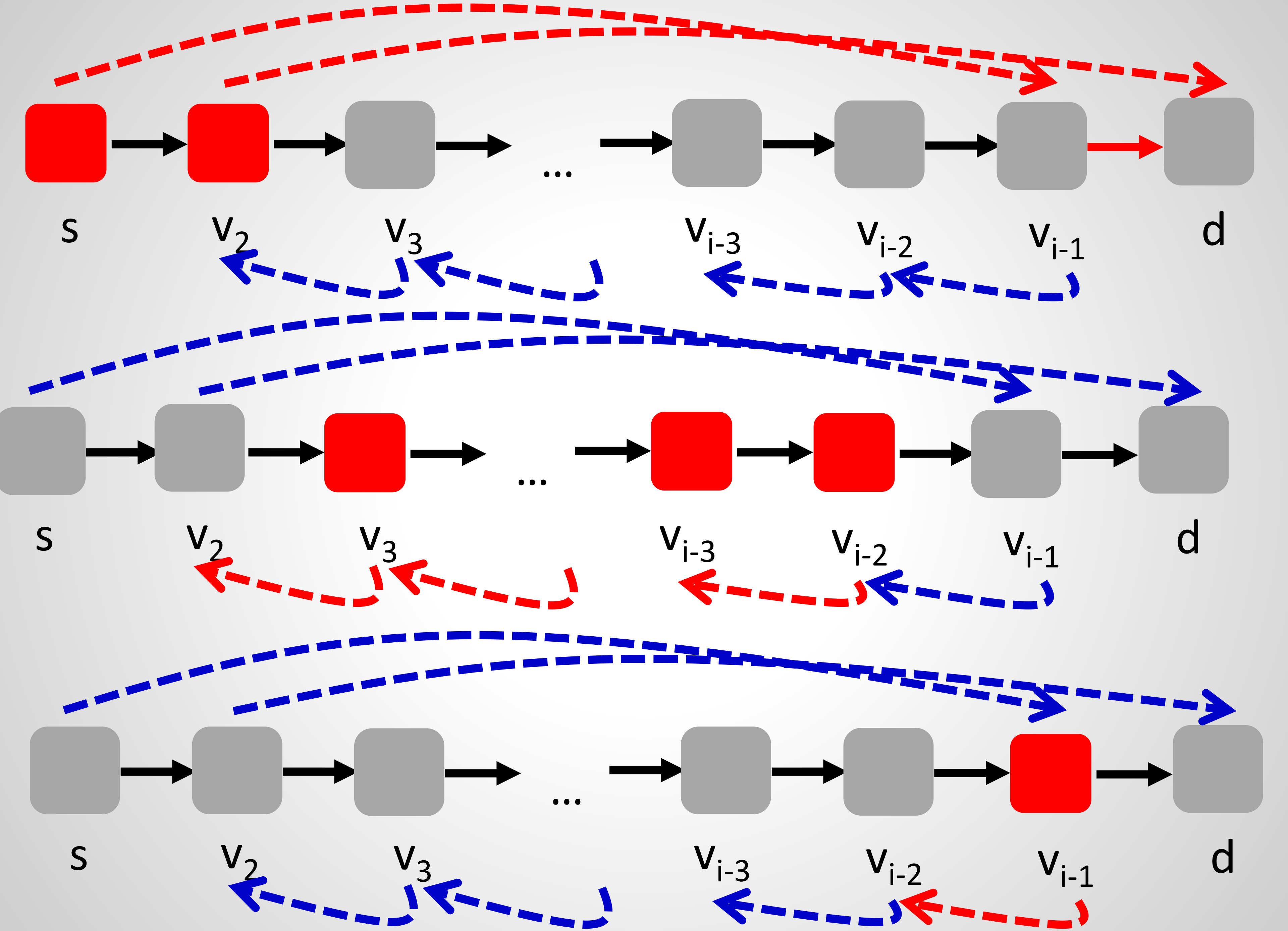
- *However:* Topological l... ay not be a problem if ...) path

For this example: Only old packets may loop here, new packets from s go via v_2 to d.

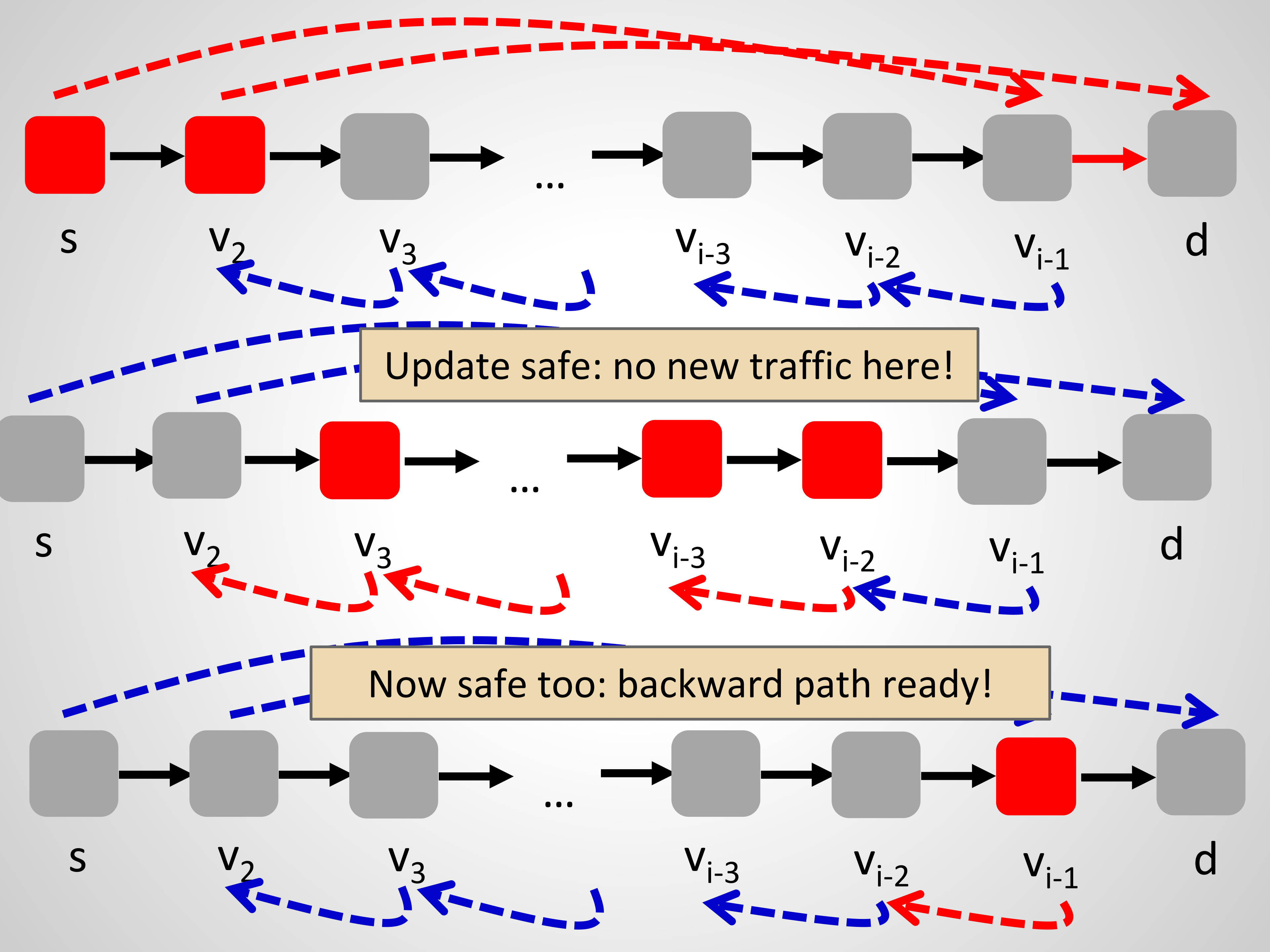
However: It can be good to relax!



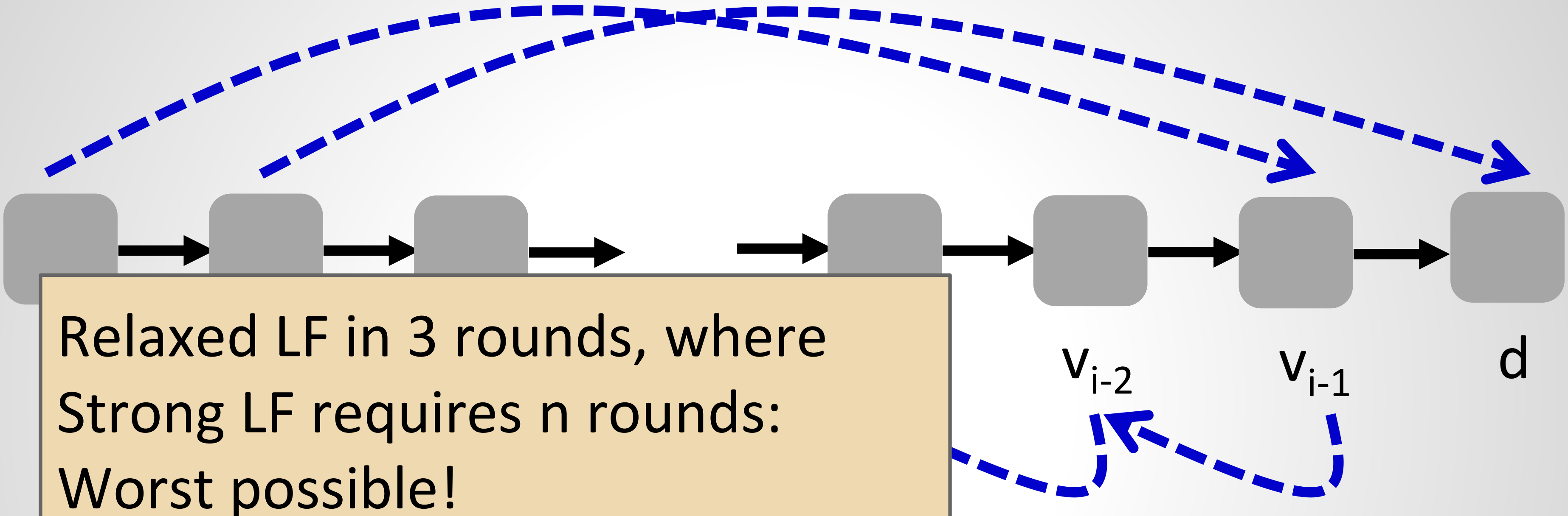
- *However:* Topological loops may not be a problem if they do not occur on the active (s,d) path
- **Schedule: (1) forward edges, (2) backward edges except last one, (3) last backward edge**



...
s
v2
v3
vi-3
vi-2
vi-1
d
...
s
v2
v3
vi-3
vi-2
vi-1
d
...
s
v2
v3
vi-3
vi-2
vi-1
d


s
v2
v3
...
vi-3
vi-2
vi-1
d
Update safe: no new traffic here!
Now safe too: backward path ready!

However: It can be good to relax!

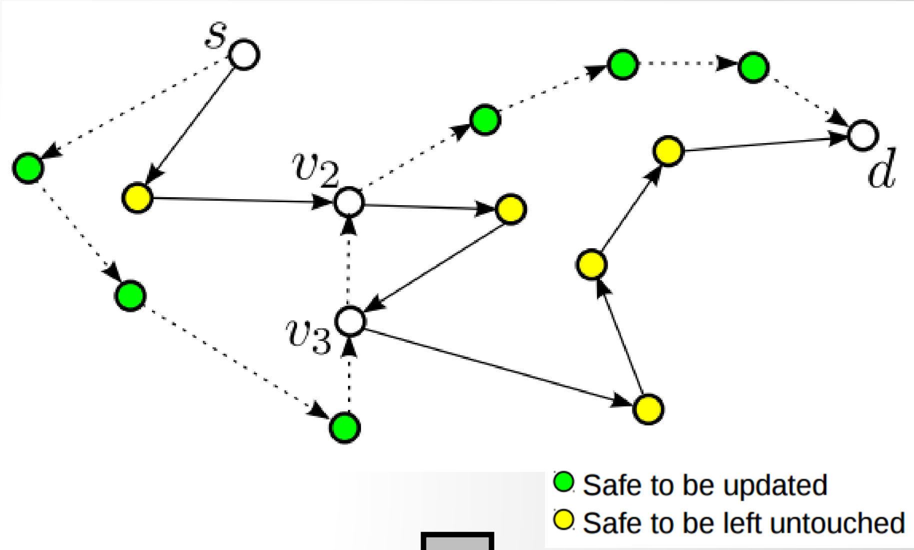


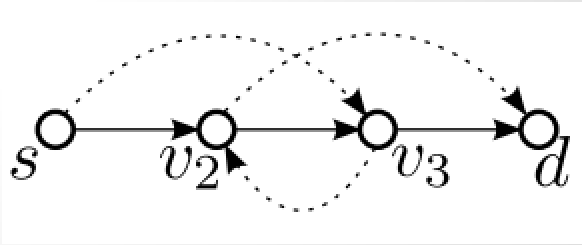
Relaxed LF in 3 rounds, where Strong LF requires n rounds: Worst possible!

- Topological loops n not be a problem if they do not occur on the ac ve (s,d) path
- **Schedule: (1) forward edges, (2) backward edges except last one, (3) last backward edge**

Why did we consider the line only? Model & Simplification

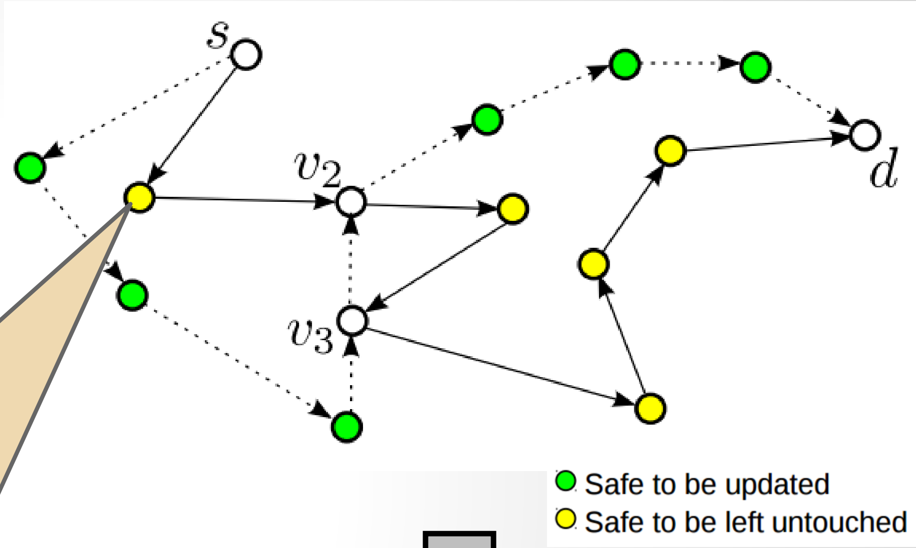
- Given old (solid) and new path (dashed)
- We can focus on nodes which need to be updated and lie on both paths (others trivial)
- Can be represented as a line
- Convention: old path solid from left to right

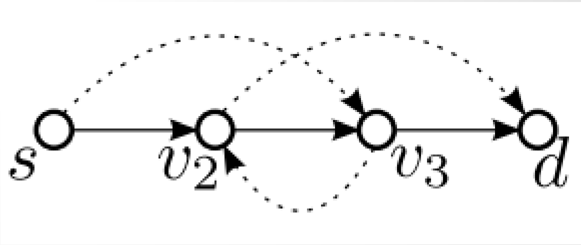




Why did we consider the line only? Model & Simplification

- Given old (solid) and new path (dashed)
- We can focus on nodes which need to be updated and lie on both pa (others trivial)





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

Good Algorithms to Schedule (Strong) LF Updates?

Idea: Greedy

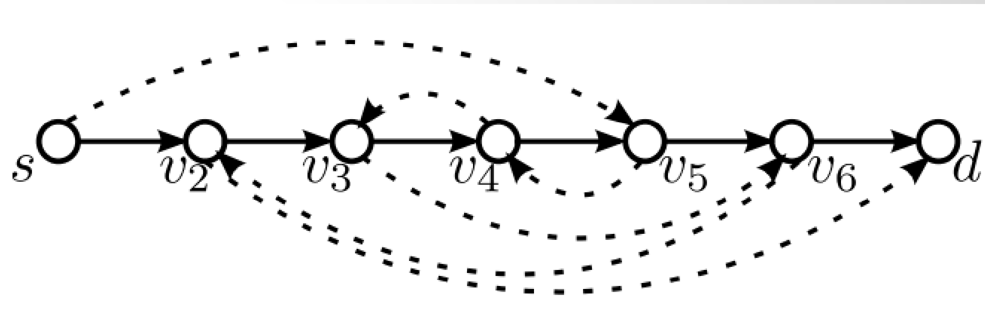
- Greedy: Schedule a maximum number of nodes in each round!

- However, it turns out that this is bad:
  - A single greedy round can force the best possible schedule to go from O(1) to Ω(n) rounds
  - Moreover, being greedy in NP-hard: a (hard) special variant of Feedback Arc Set Problem (out-degree 2, 2 valid paths)

Less Ambitious: Algorithms for 2-Round Instances?

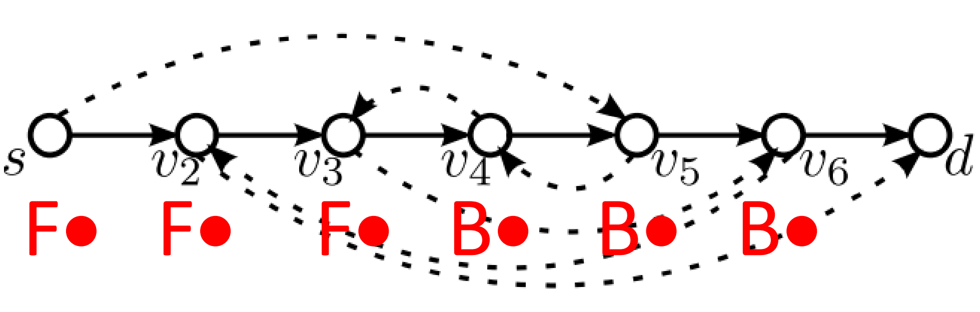
Less Ambitious: Algorithms for 2-Round Instances

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



Less Ambitious: Algorithms for 2-Round Instances

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

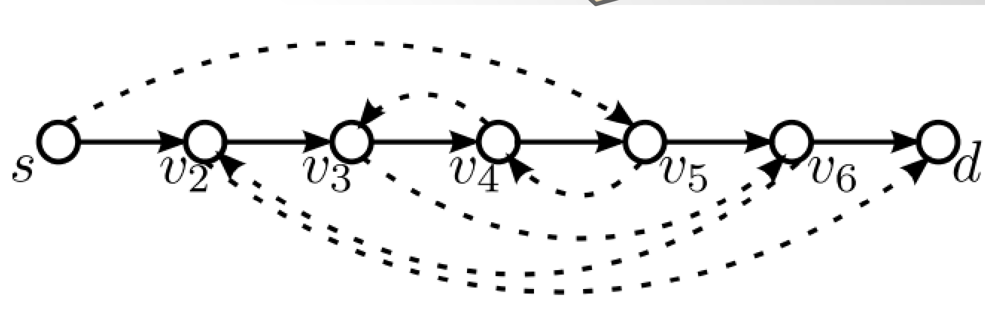


Less Ambitious: Algorithms for 2-Round Instances

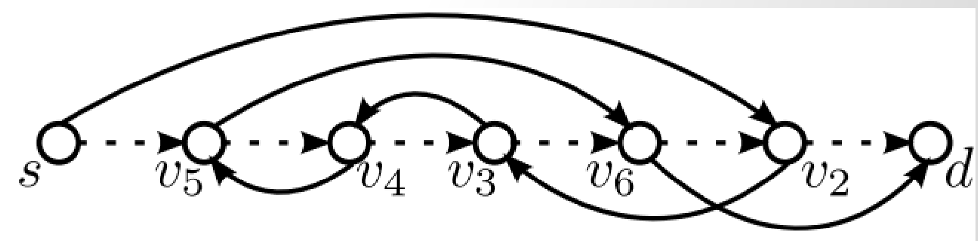
- Classify nodes/edg

Old policy from left to right!

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



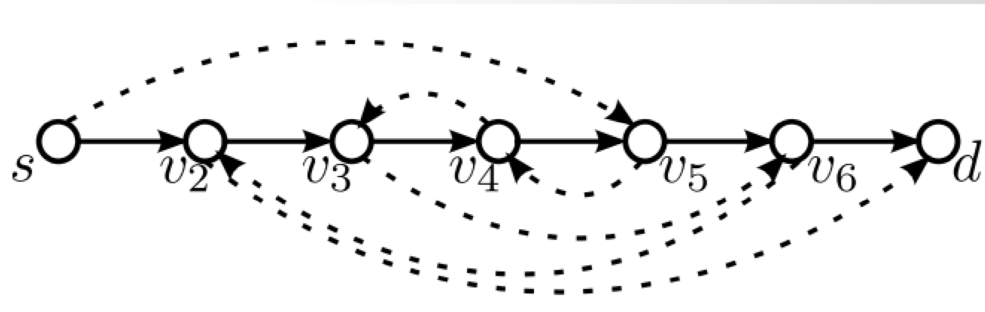
  - •F, •B: Does the (solid) old edge point forward or backwart wrt (dashed) new path?



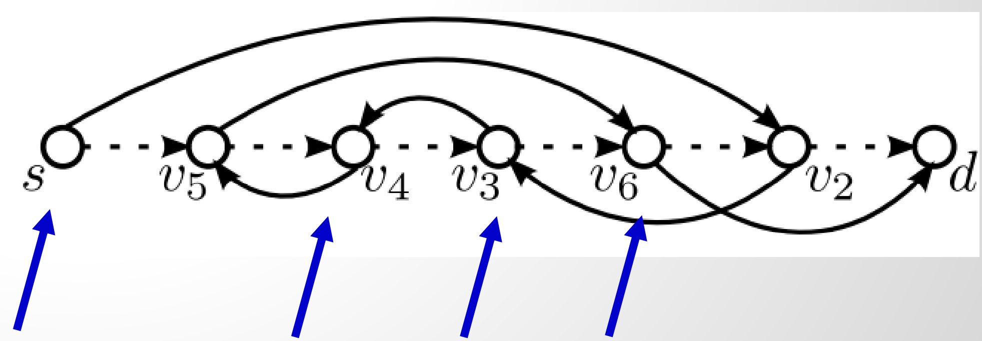
New policy from left to right!

Less Ambitious: Algorithms for 2-Round Instances

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



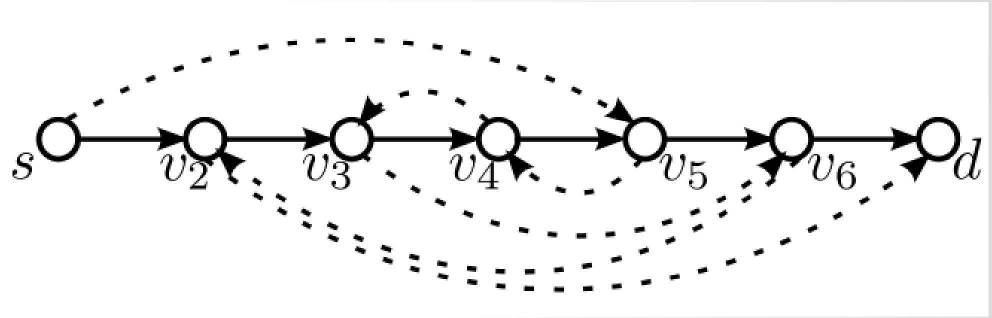
  - •F, •B: Does the (solid) old edge point forward or backwart wrt (dashed) new path?



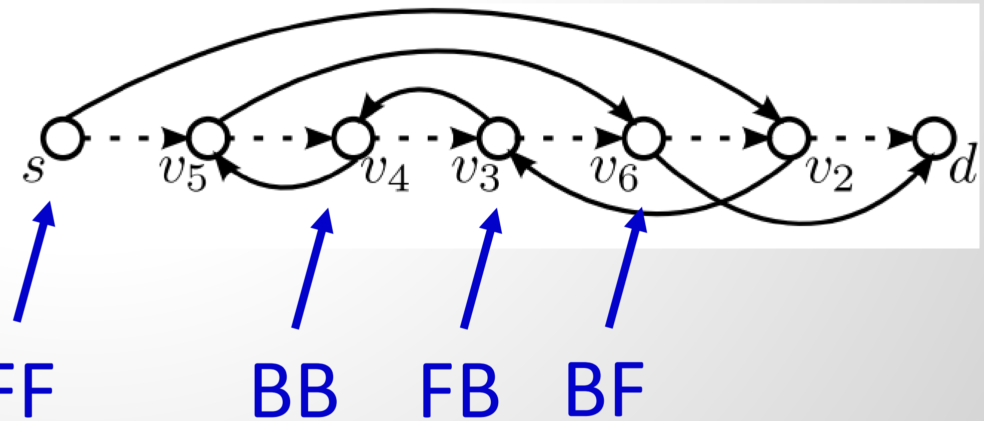
? ? ? ?

Less Ambitious: Algorithms for 2-Round Instances

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



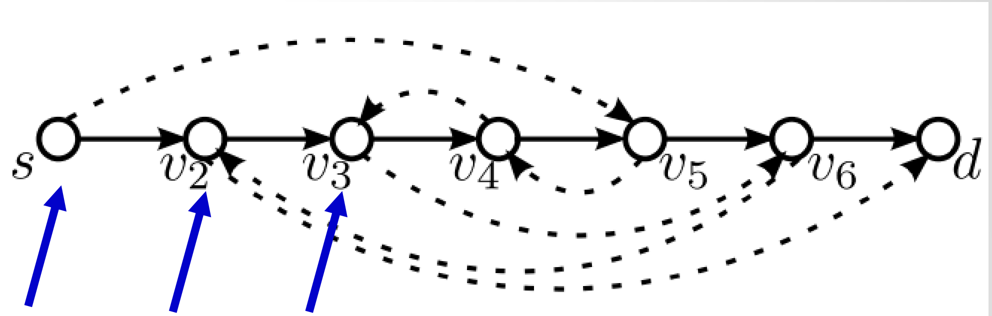
  - •F, •B: Does the (solid) old edge point forward or backwart wrt (dashed) new path?



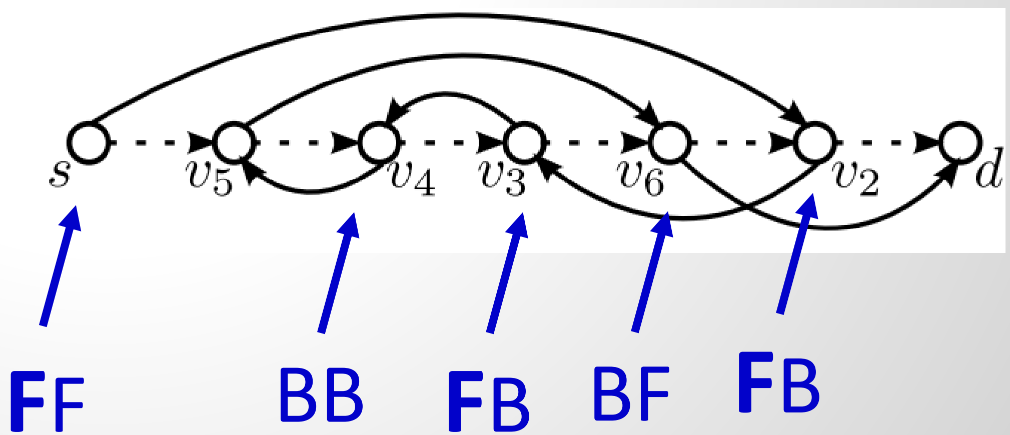
Insight 1: In the 1st round, I can safely update all forwarding (F•) edges! For sure loopfree.

…rithms for 2-Round Instances

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



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



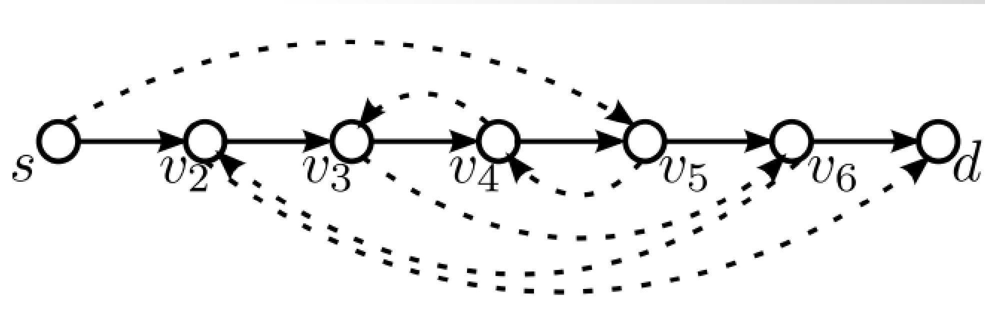
Insight 1: In the 1st round, I can safely update all forwarding (F•) edges! For sure loopfree.

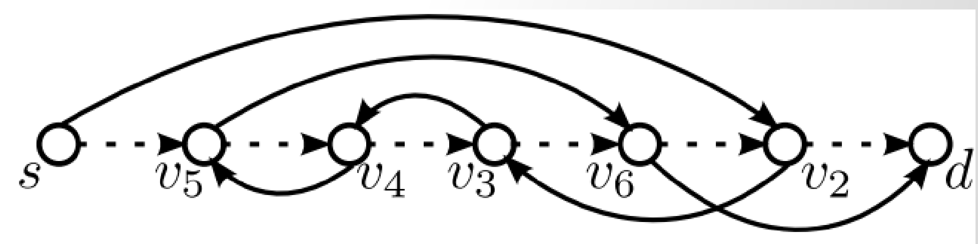
...rithms for 2-Round Instances

- ...s with 2-letter code:

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

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





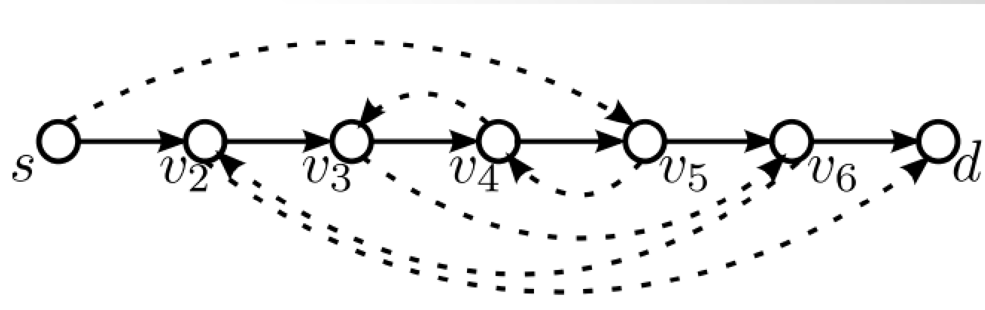
rithms for 2-Round Instances

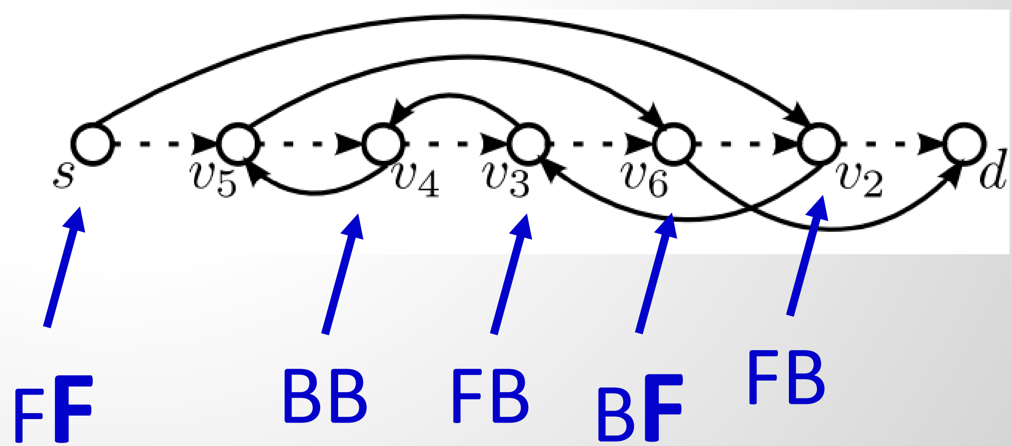
es with 2-letter code:

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 used 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.





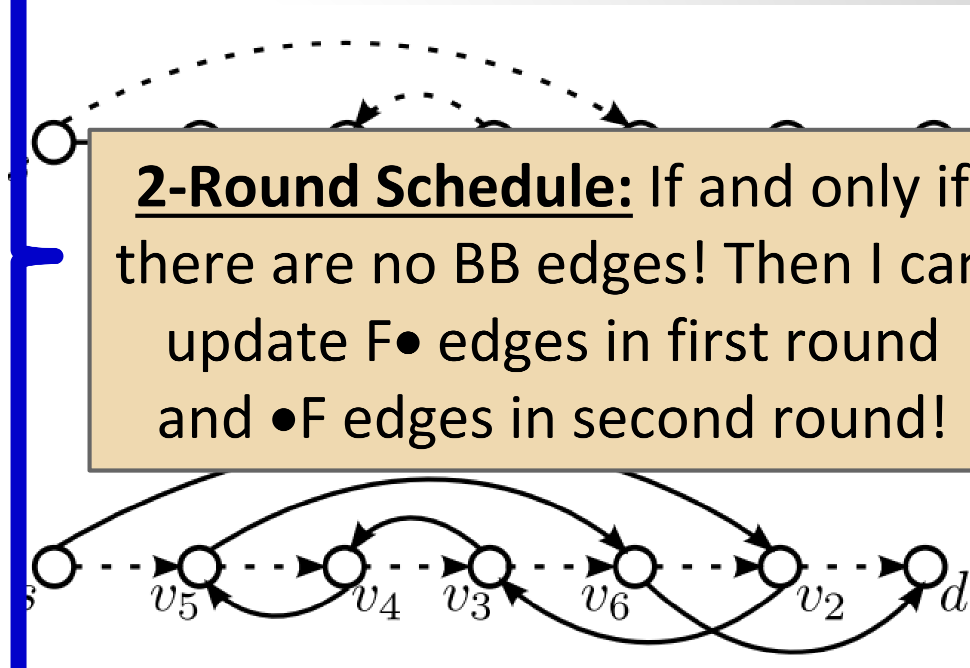
…rithms for 2-Round Instances

…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 used 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!

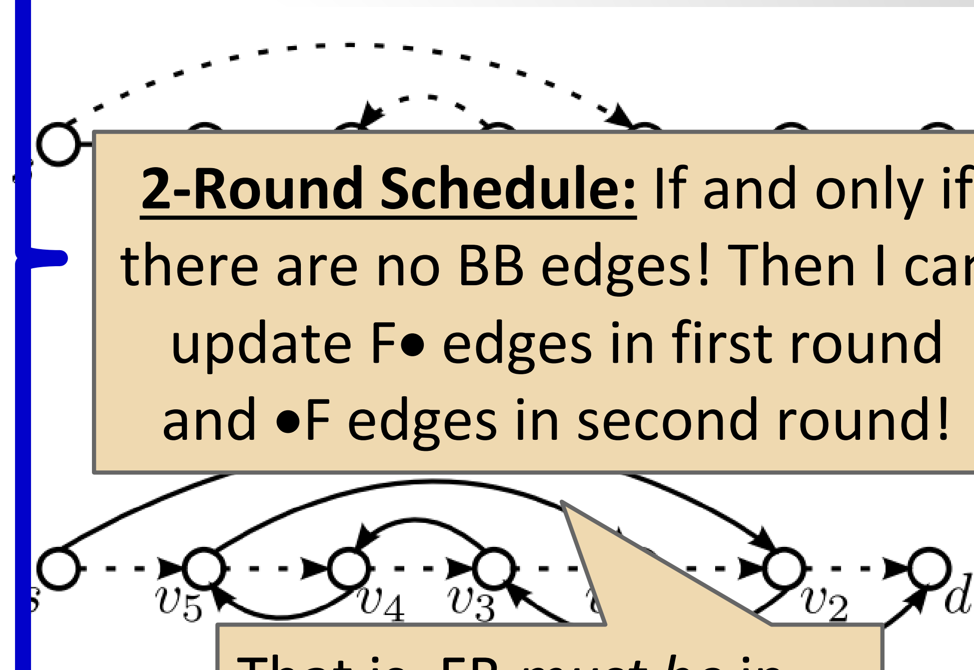
…rithms for 2-Round Instances

…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 used 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*!

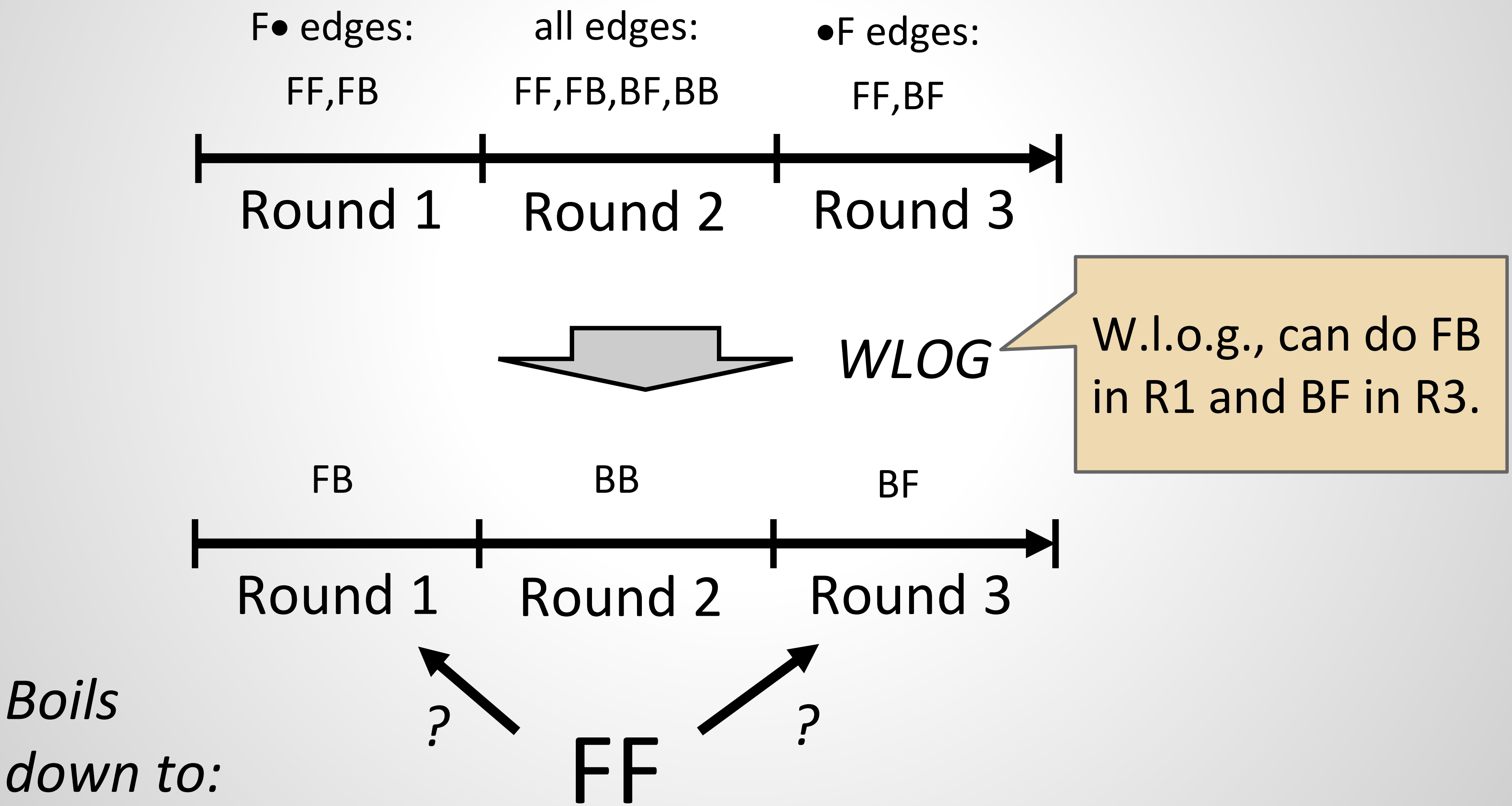
What about 3 rounds?

What about 3 rounds?

- Structure of a 3-round schedule:

What about 3 rounds?

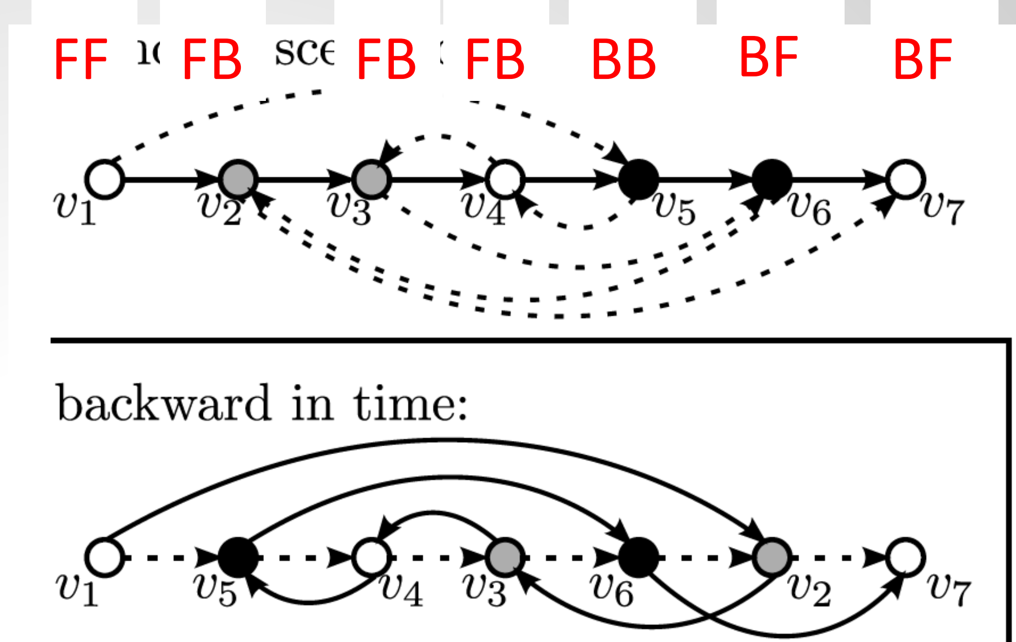
- Structure of a 3-round schedule:



Proof

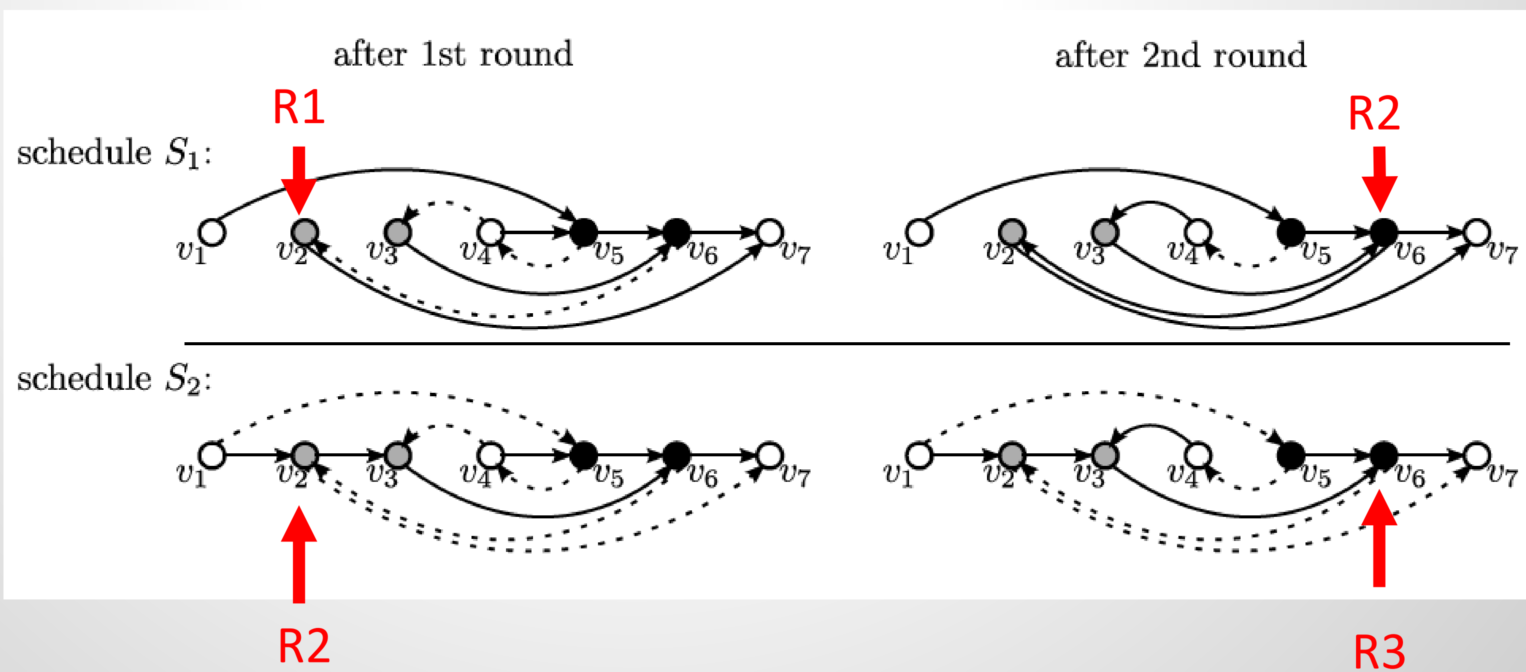
Claim: If there exists 3-round schedule, then also one where FB are only updated in Round 1.

Reason: Can move FB to first round!



S1: as early as possible

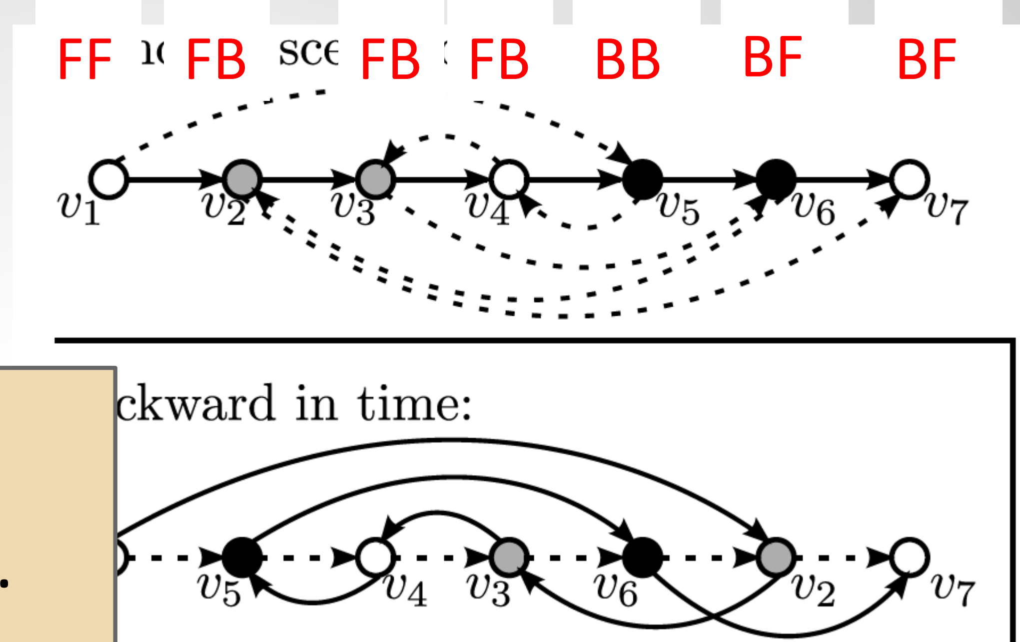
S2: as late as possible



Proof

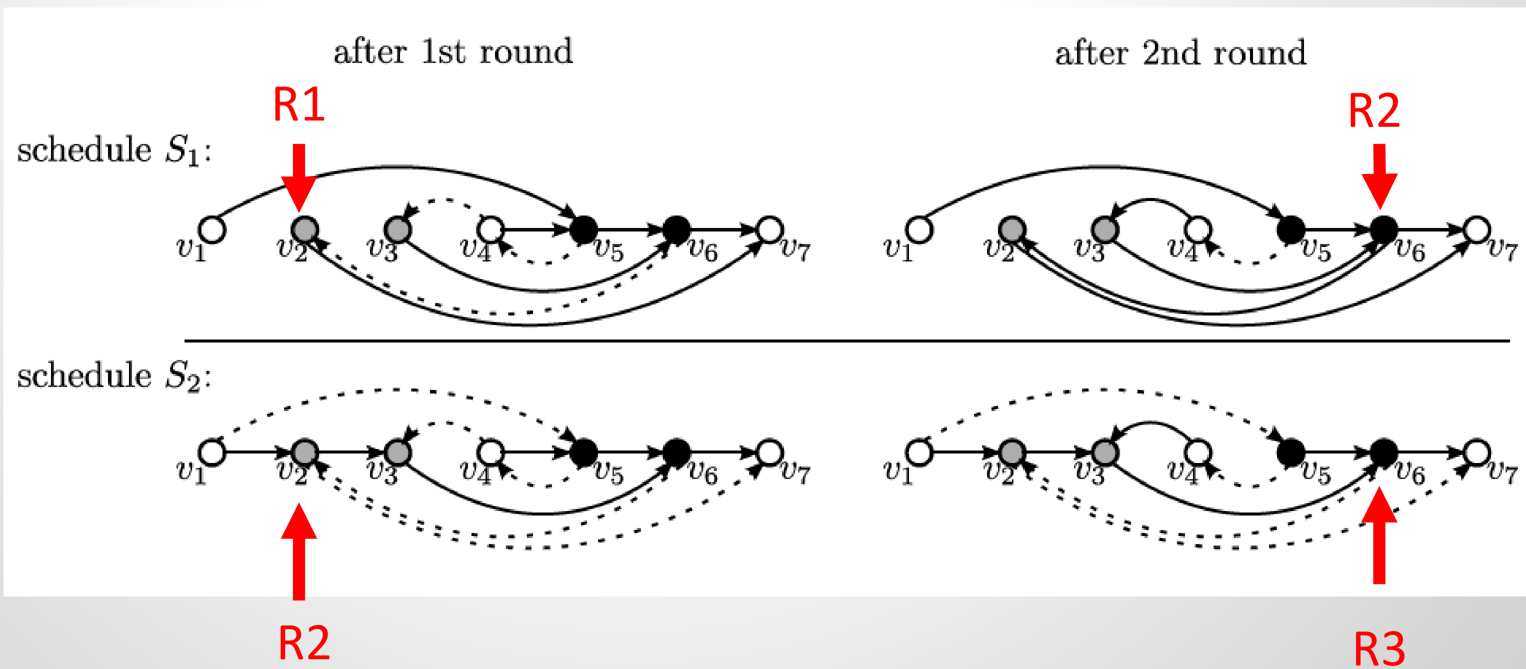
Claim: If there exists 3-round schedule, then also

Fowarding edges do not introduce loops in G(t=1).



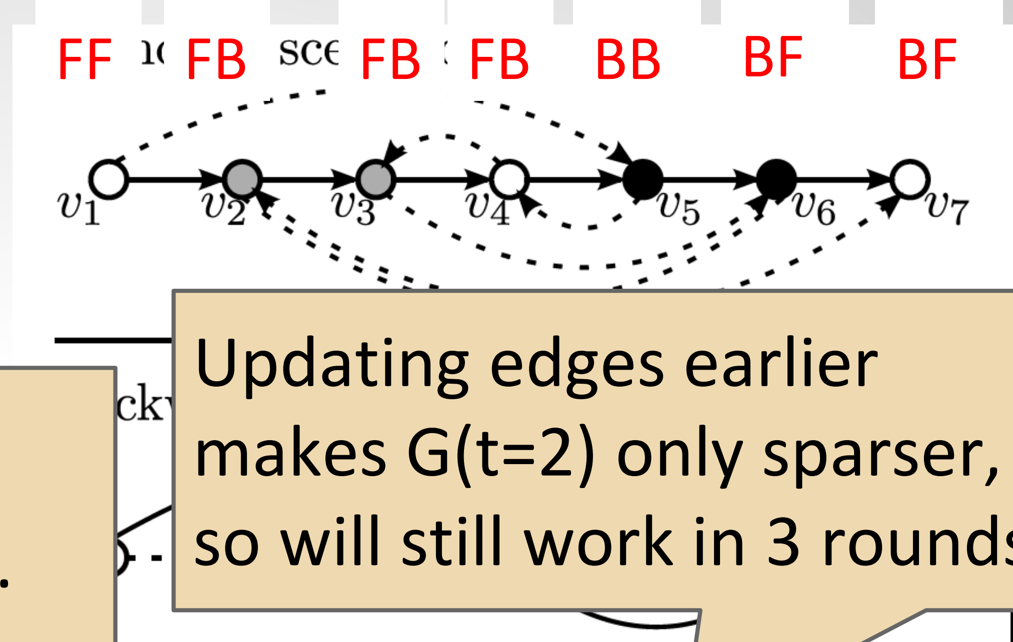
S1: as early as possible

S2: as late as possible



Proof

Claim: If there exists 3-round schedule, then also

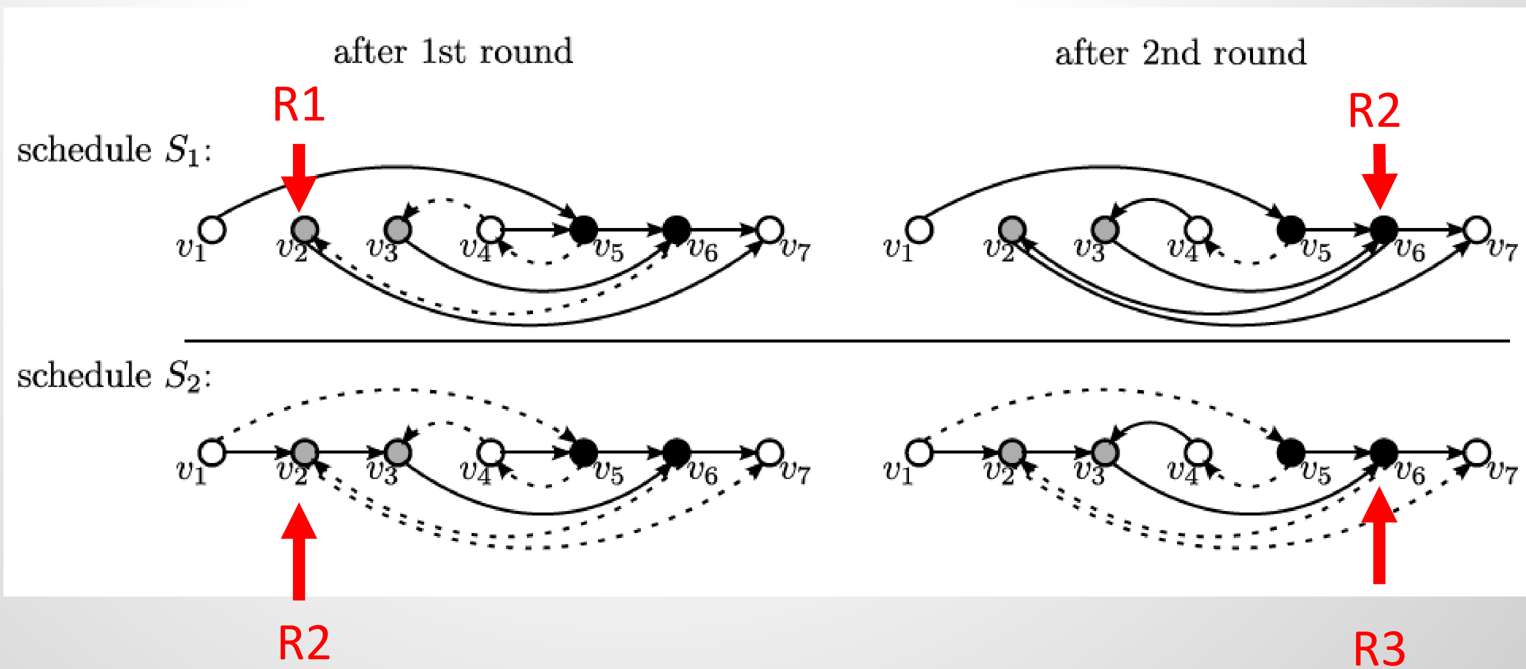


Fowarding edges do not introduce loops in G(t=1).

Updating edges earlier makes G(t=2) only sparser, so will still work in 3 rounds.

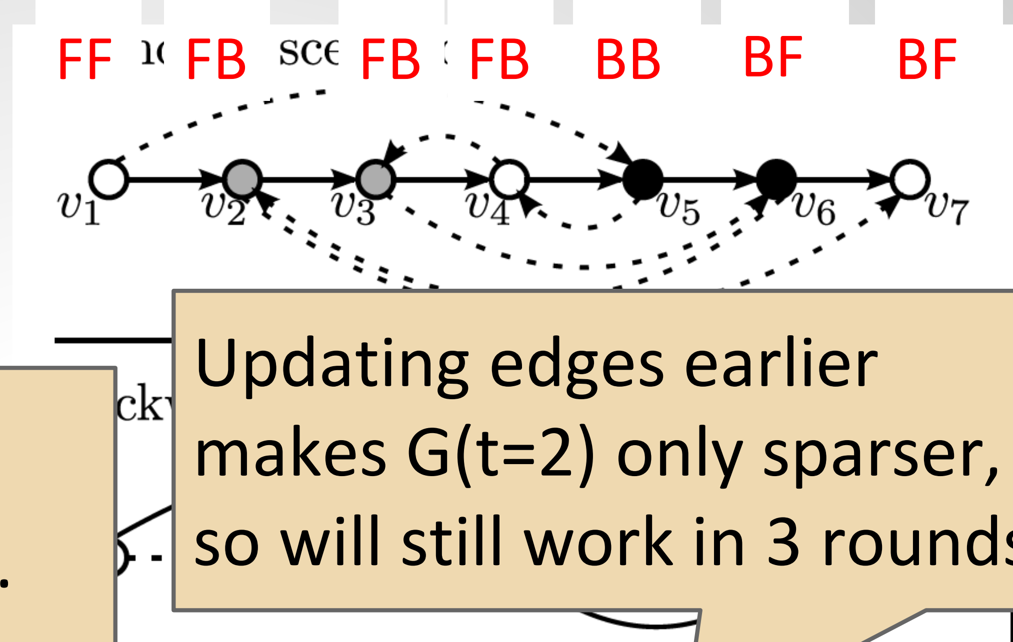
S1: as early as possible

S2: as late as possible



Proof

Claim: If there exists 3-round schedule, then also

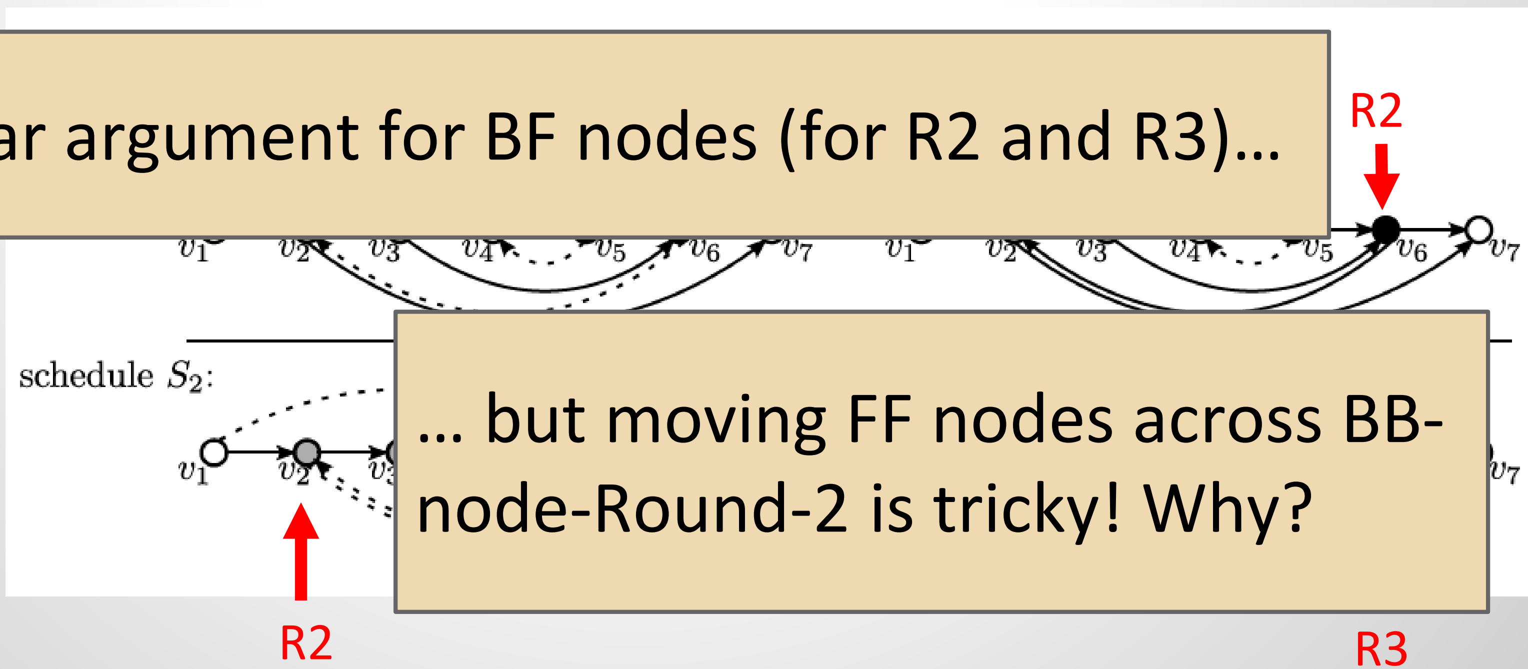


Fowarding edges do not introduce loops in G(t=1).

Updating edges earlier makes G(t=2) only sparser, so will still work in 3 rounds.

S1: as ... as possible

S2: as late as possible

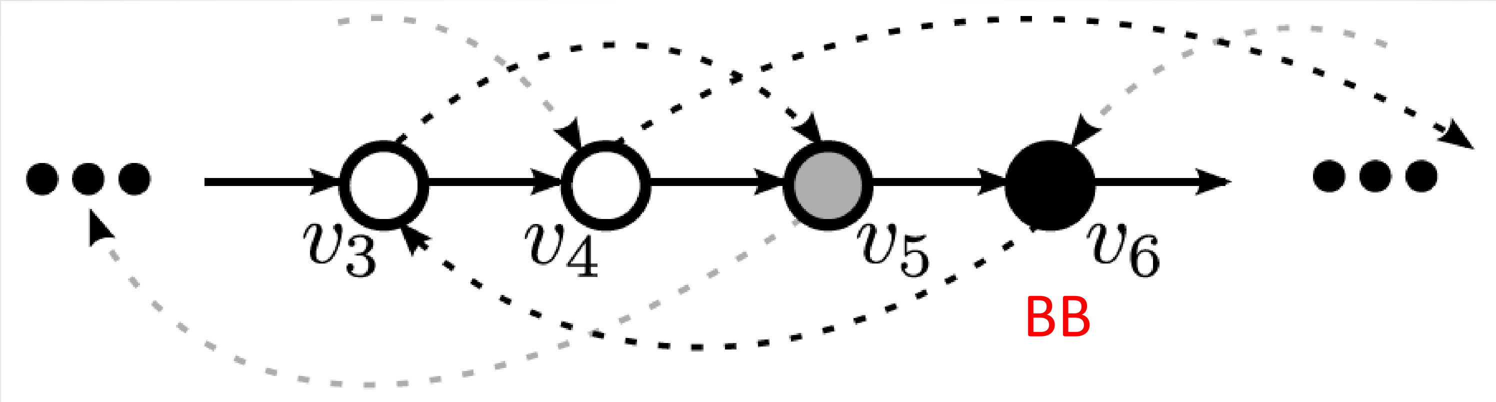


Similar argument for BF nodes (for R2 and R3)…

… but moving FF nodes across BB-node-Round-2 is tricky! Why?

NP-hardness

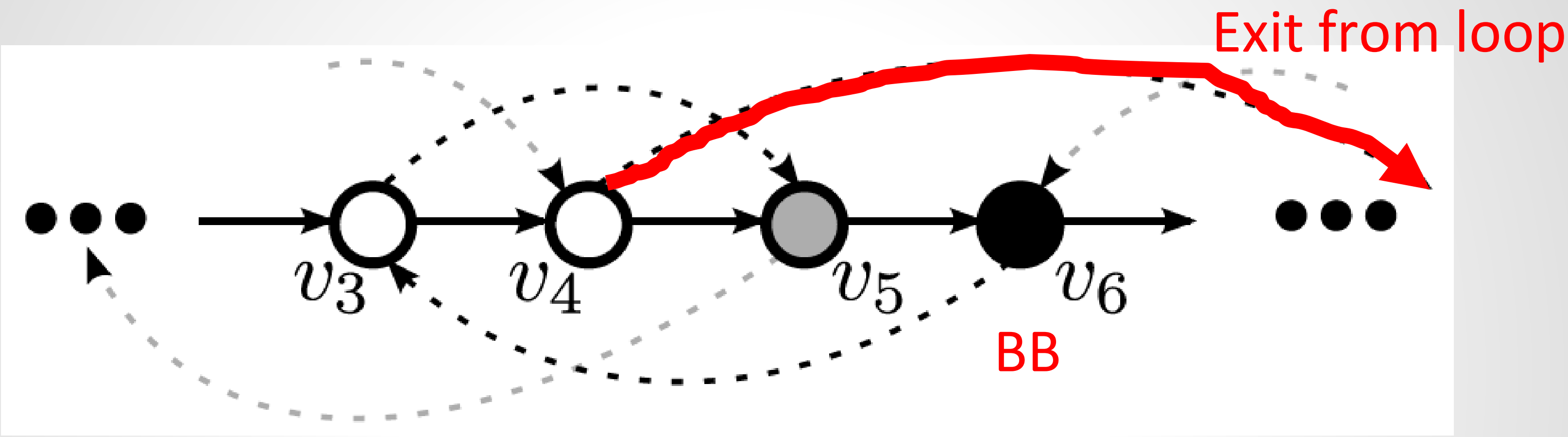
A hard decision problem: when to update FF?



- ❑ We know: BB node v_6 can only be updated in R2

NP-hardness

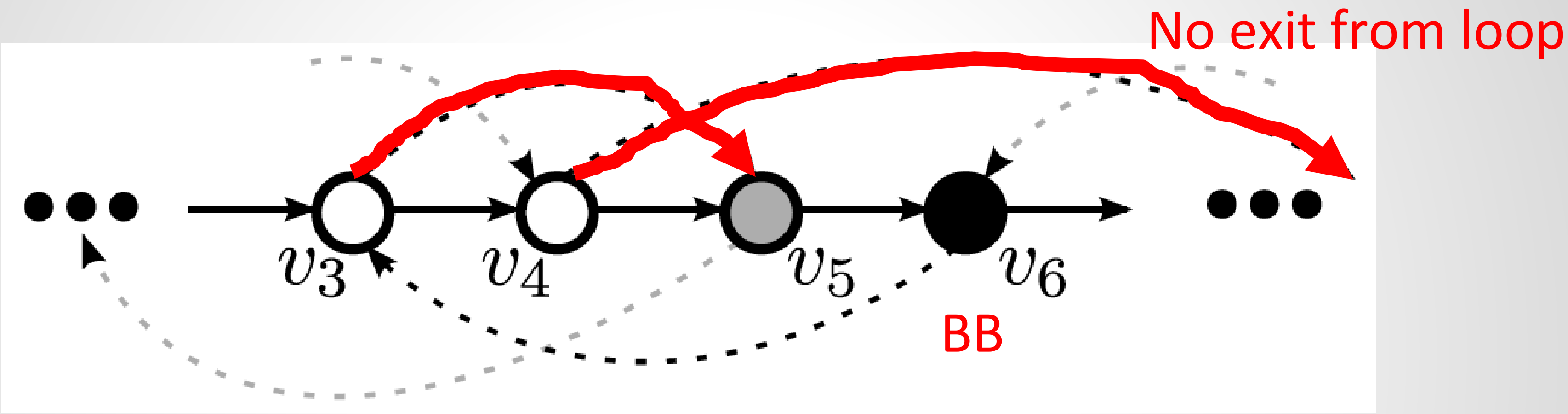
A hard decision problem: when to update FF?



- We know: BB node v_6 can only be updated in R2
- Updating FF-node v_4 in R1 allows to update BB node v_6 in R2

NP-hardness

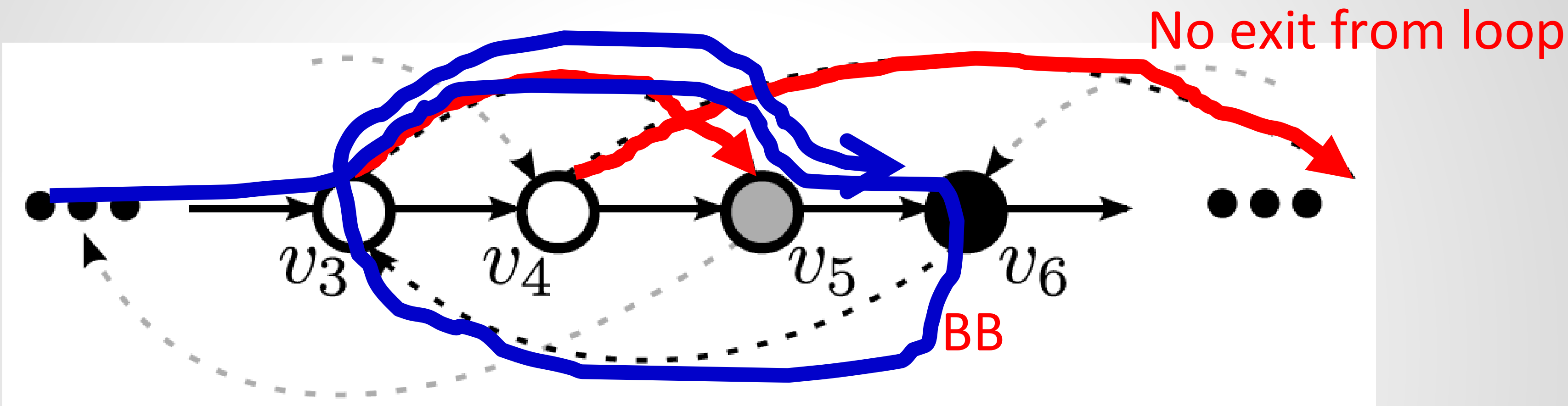
A hard decision problem: when to update FF?



- We know: BB node v_6 can only be updated in R2
- Updating FF-node v_4 in R1 allows to update BB node v_6 in R2
- Updating FF-node v_3 as well in R1 would be bad: cannot update v_6 in next round: potential loop

NP-hardness

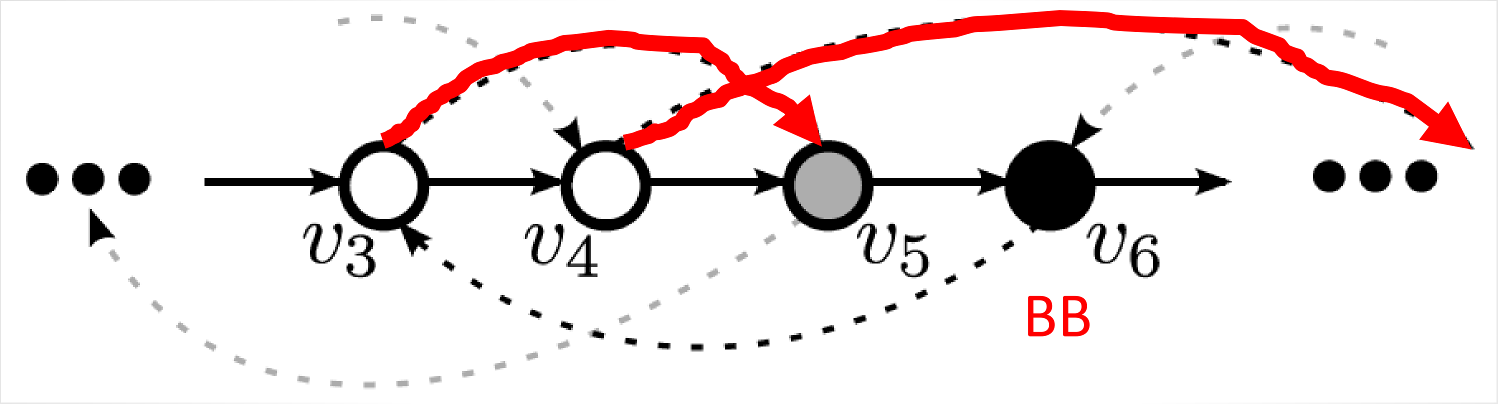
A hard decision problem: when to update FF?



- We know: BB node v_6 can only be updated in R2
- Updating FF-node v_4 in R1 allows to update BB node v_6 in R2
- Updating FF-node v_3 as well in R1 would be bad: cannot update v_6 in next round: potential loop

NP-hardness

A hard decision problem: when to update FF?



- We know: BB node v_6 can only be updated in R2
- Updating FF-node v_4 in R1 allows to update BB node v6 in R2
- Updating FF-node v_3 as well in R1 would be bad: cannot update v_6 in next round: potential loop
- Node v_5 is B• and cannot be updated in R1

NP-hardness

- Reduction from a 3-SAT version where variables appear only a small number of times
  - Variable x appearing p_x times positively and n_x times negatively is replaced by:

    $$x_0, x_1, \ldots, x_{p_x}, x_l, \overline{x}_0, \overline{x}_1, \ldots, \overline{x}_{n_x}$$

  - Gives low-degree requirements!

- Types of clauses
  - Assignment clause: $(x_0 \vee \overline{x}_0)$
  - Implication clause: $(x_i \rightarrow x_{i+1})$
  - Exclusive Clause: $(\neg x_l \vee \neg \overline{x}_l)$

NP-hardness

We need a low degree…

- Reduction from … where variables appear only a small number of times
  - Variable x appearing p_x times positively and n_x times negatively is replaced by:

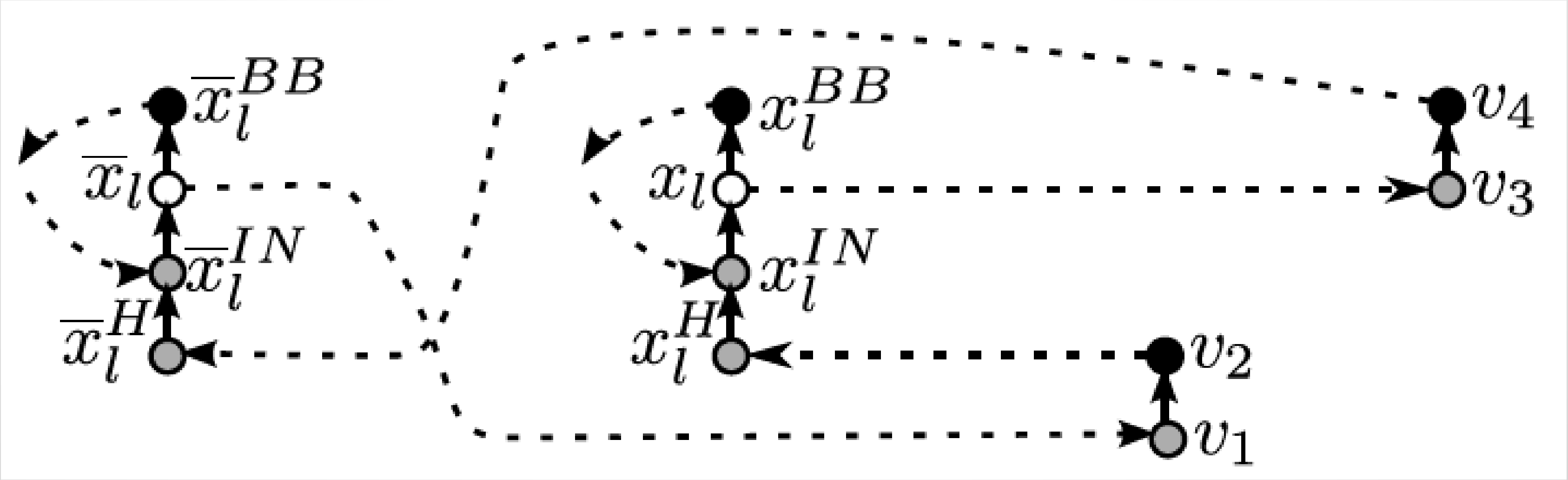
    $$x_0, x_1, \ldots, x_{p_x}, x_l, \overline{x}_0, \overline{x}_1, \ldots, \overline{x}_{n_x}$$
  - Gives low-degree requirements!

- Types of clauses
  - Assignment clause: $(x_0 \vee \overline{x}_0)$
  - Implication clause: $(x_i \rightarrow x_{i+1})$
  - Exclusive Clause: $(\neg x_l \vee \neg \overline{x}_l)$

Connecting clones: consistent value for original variable.

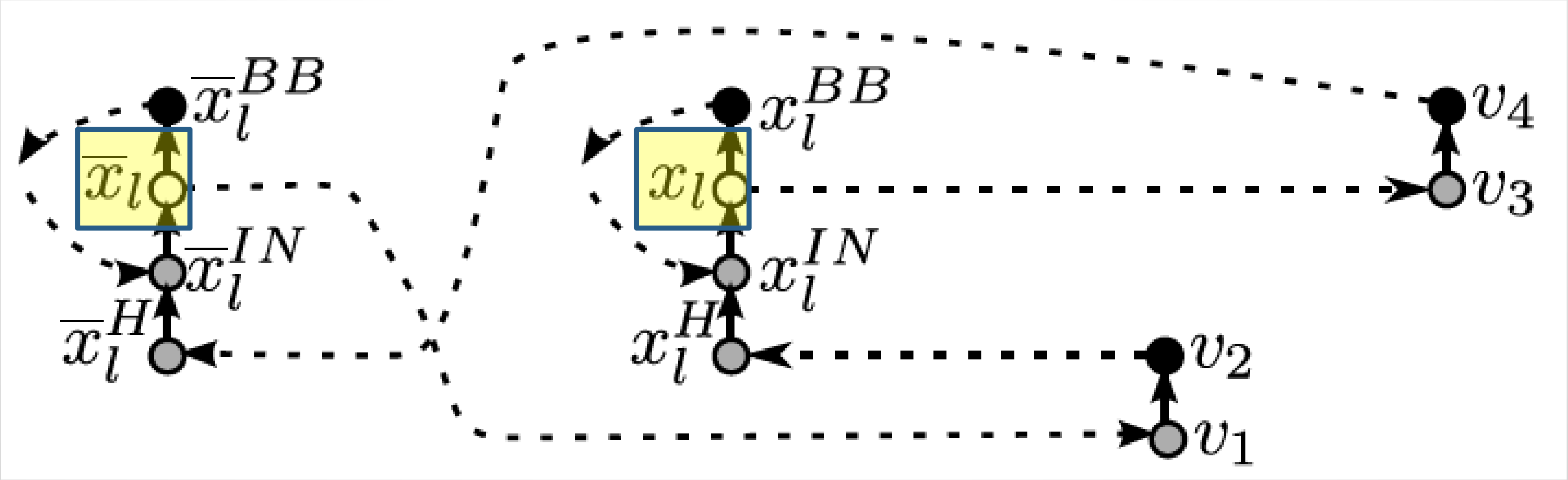
Example: Gadget for Exclusive Clause $(\neg x_l \vee \neg \overline{x}_l)$

- Updating x_l prevents $\overline{x}_l$ update and vice versa
- BB nodes v_2 and v_4 need to be updated in R2 and will introduce a cycle otherwise
- So only one of the two can be updated in R1



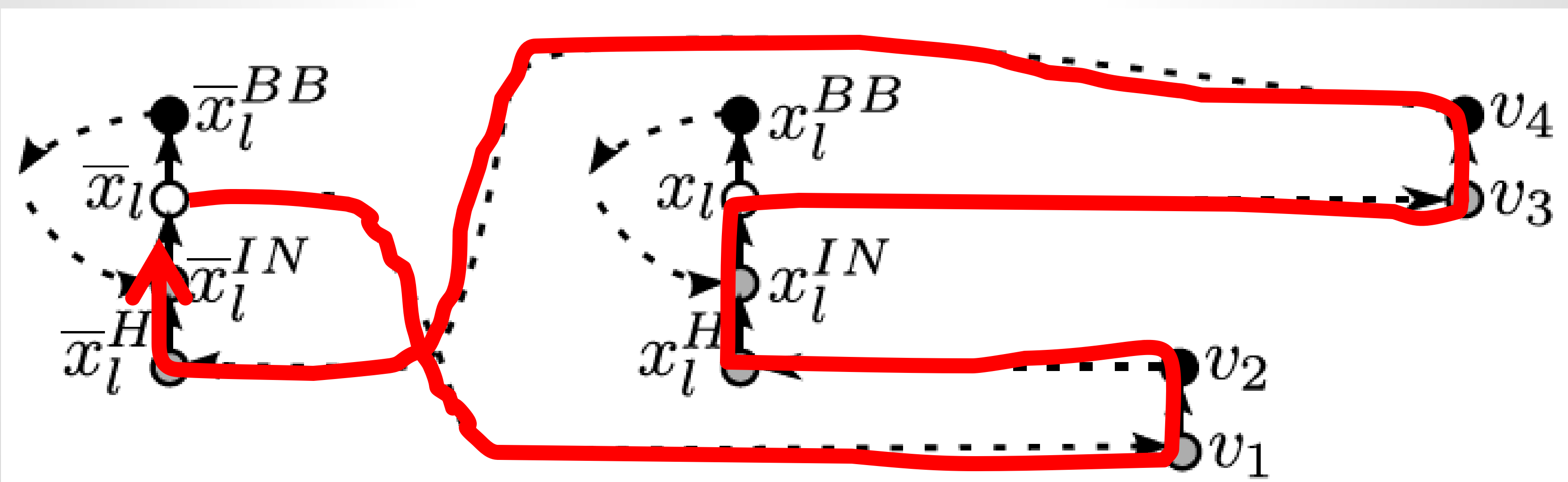
Example: Gadget for Exclusive Clause $(\neg x_l \vee \neg \overline{x}_l)$

- Updating x_l prevents $\overline{x}_l$ update and vice versa
- BB nodes v_2 and v_4 need to be updated in R2 and will introduce a cycle otherwise
- So only one of the two can be updated in R1

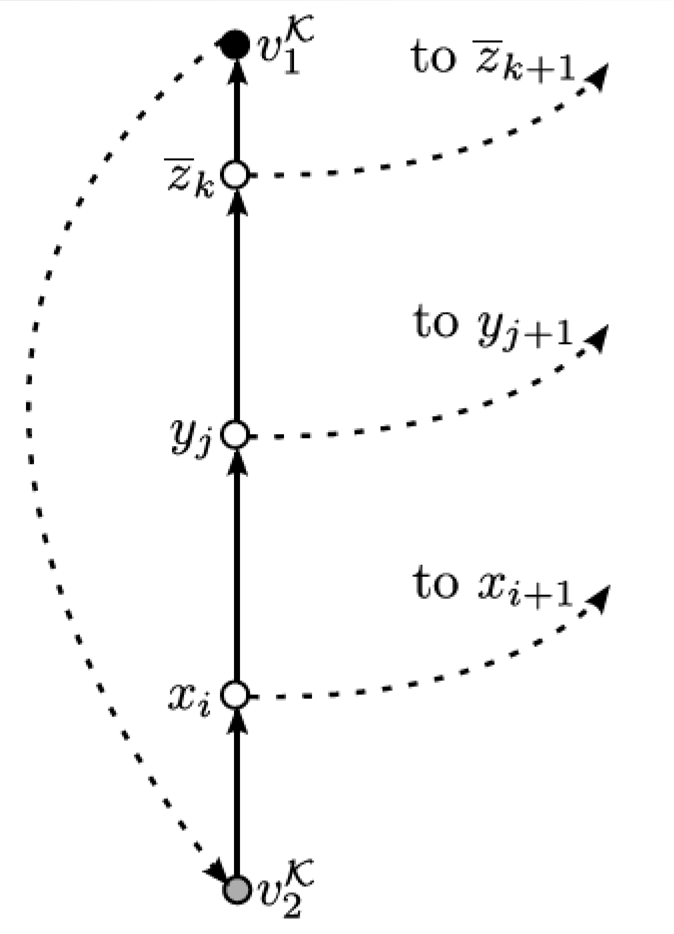


Example: Gadget for Exclusive Clause $(\neg x_l \vee \neg \overline{x}_l)$

- Updating x_l prevents x_l update and vice versa
- BB nodes v_2 and v_4 need to be updated in R2 and will introduce a cycle otherwise
- So only one of the two can be updated in R1

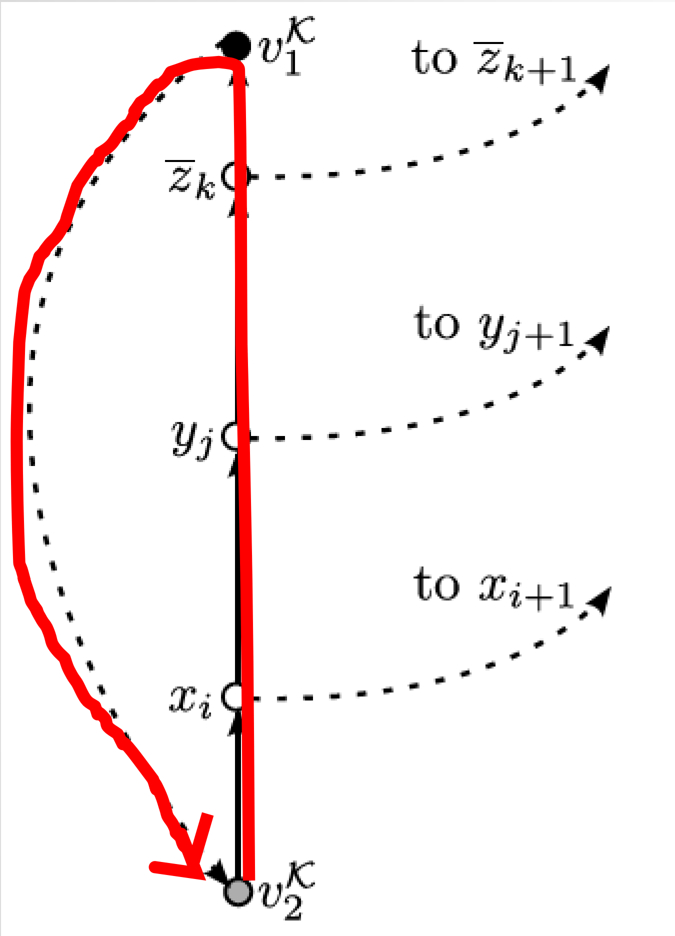


Example: Gadget for Clause $x_i \vee y_j \vee \overline{z}_k$



- ❑ Need to update (satisfy) at least one of the literals in the clause…
- ❑ … so to escape the potential loop

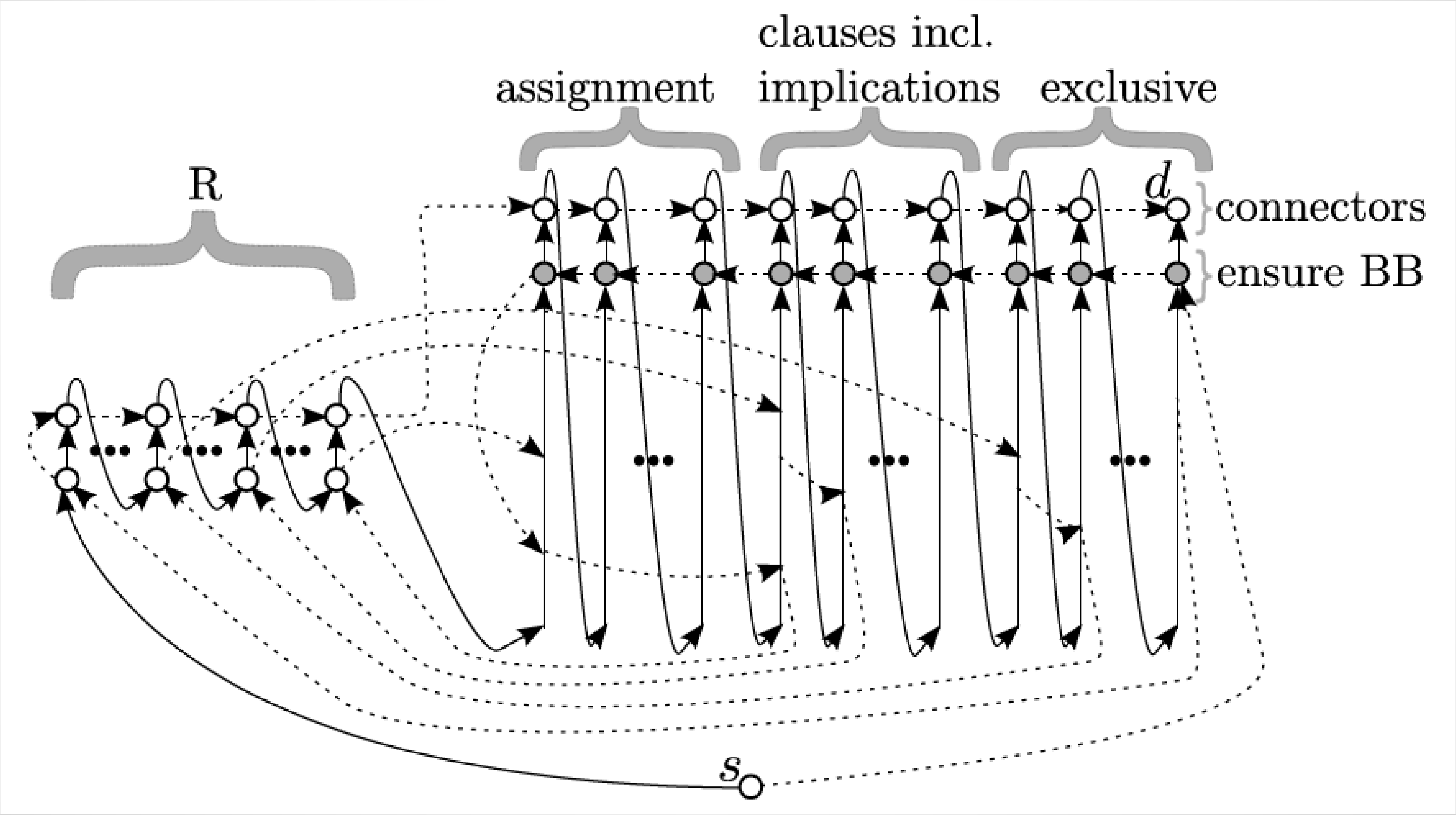
Example: Gadget for Clause $x_i \vee y_j \vee \overline{z}_k$



- Need to update (satisfy) at least one of the literals in the clause…
- … so to escape the potential loop

NP-hardness

- Eventually everything has to be connected…
- … to form a valid path



Relaxed Loopfreedom

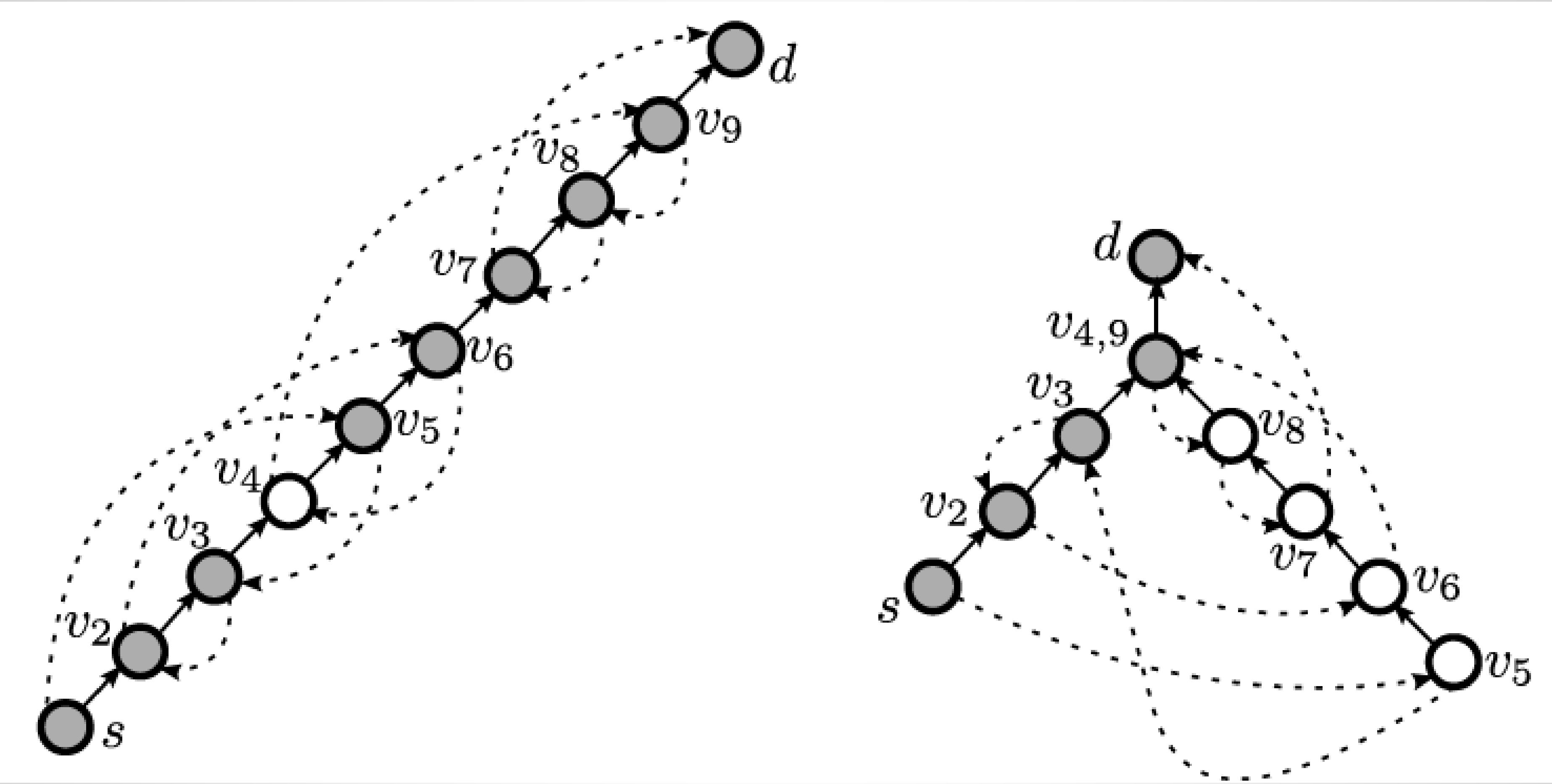
- Recall: relaxed loop-freedom can reduce number of rounds by a factor O(n)
- But how many rounds are needed for relaxed loop-free update in the worst case?
- We don't know…
- … what we do know: next slide ☺

Peacock: Relaxed Updates in O(log n) Rounds

First some concepts:

- **Node merging:** a node which is updated is irrelevant for the future, so merge it with subsequent one
- **Directed tree:** while initial network consists of two directed paths (in-degree=out-degree=2), during update rounds, situation can become a directed tree
  - in-degree can increase due to merging
  - dashed in- and out-degree however stays one

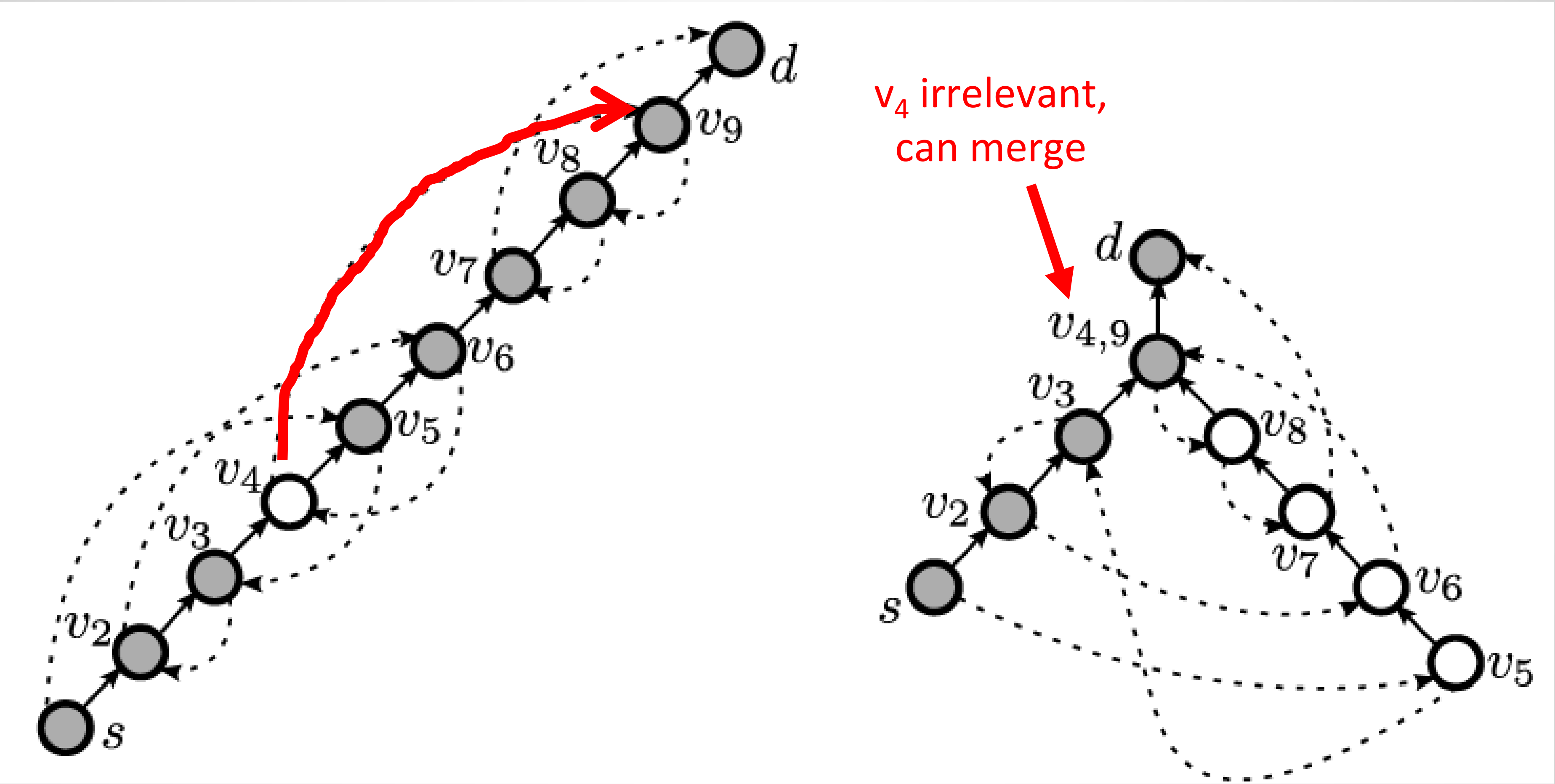
Example



Initially: Two valid paths!

After updating v_4.

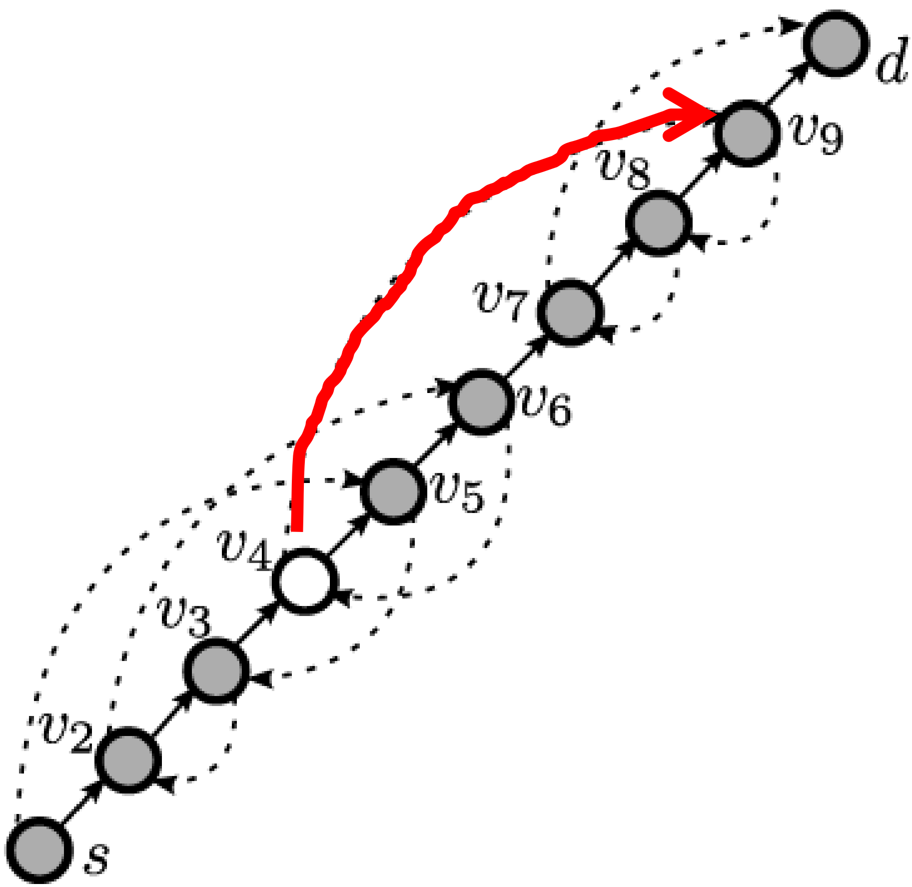
Example



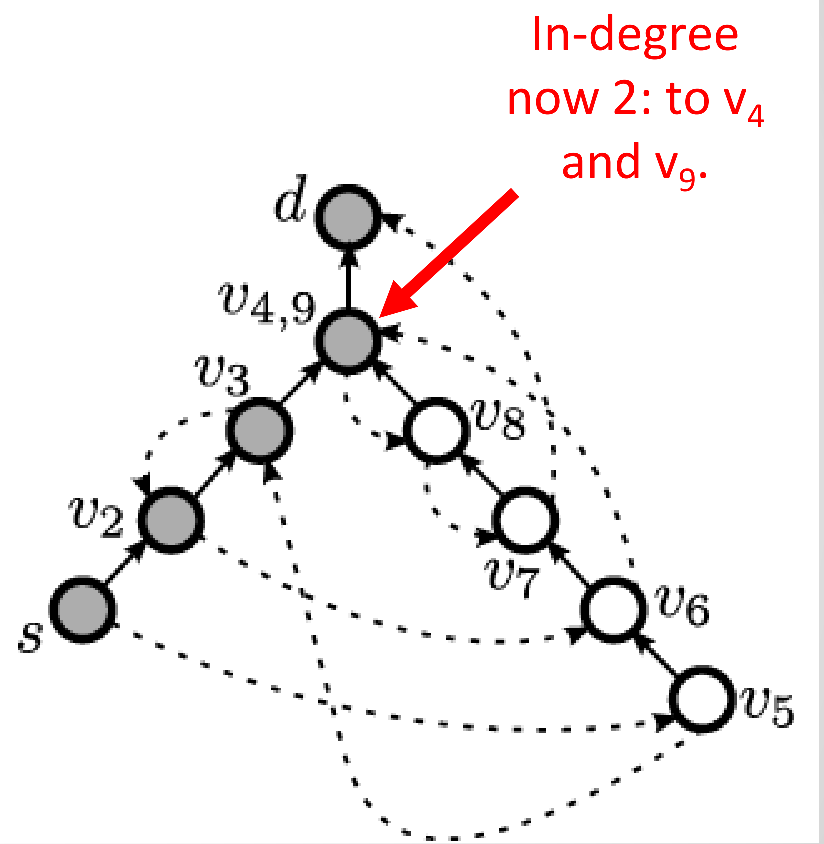
Initially: Two valid paths!

After updating v_4.

Example

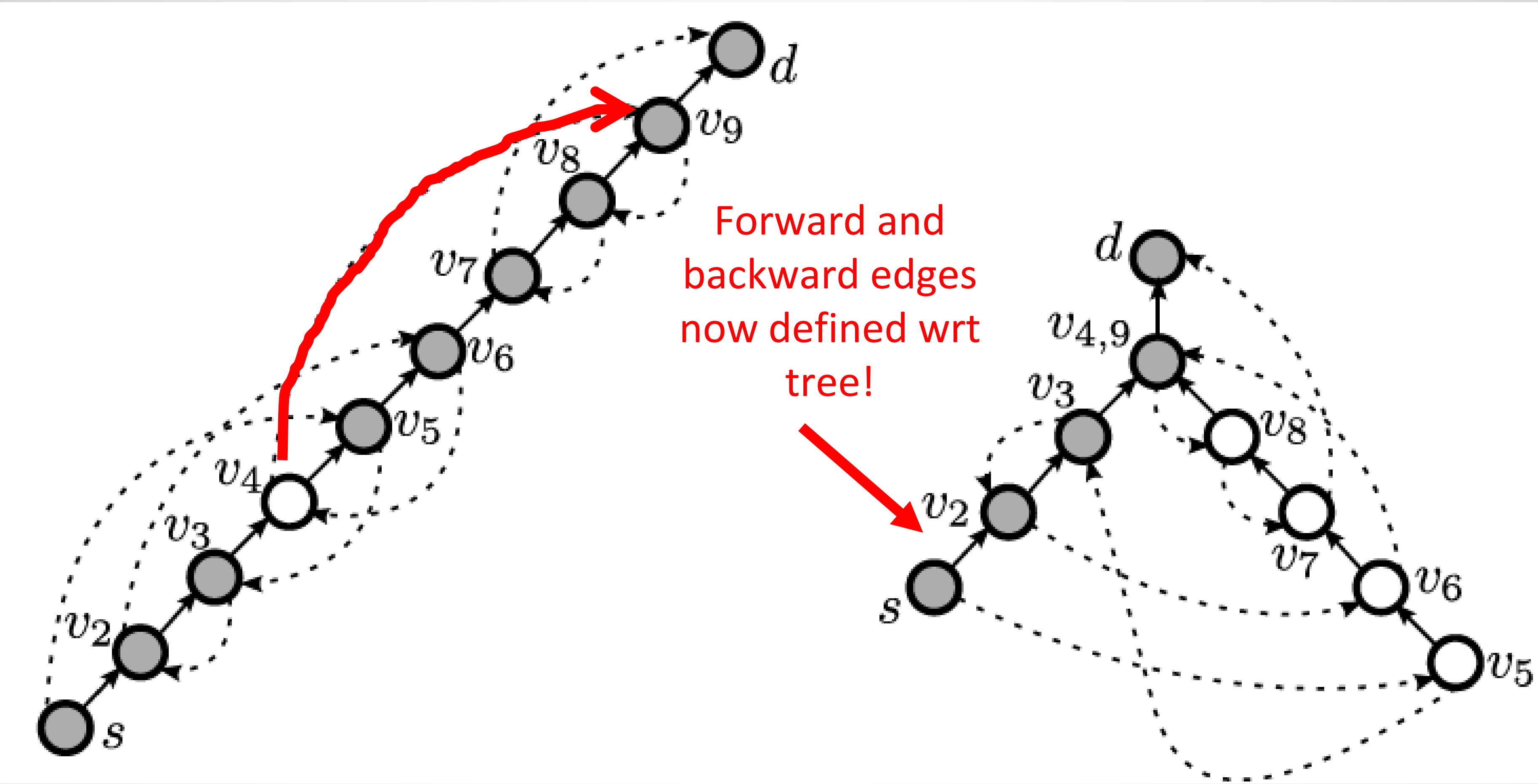


Initially: Two valid paths!



After updating v_4.

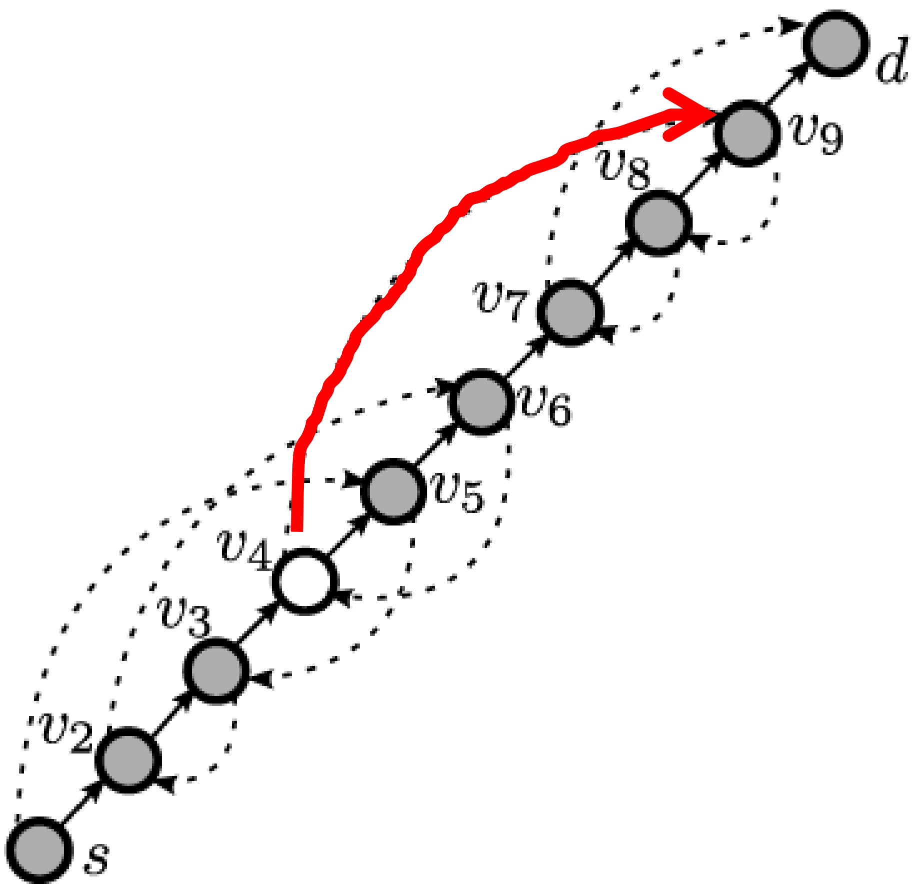
Example



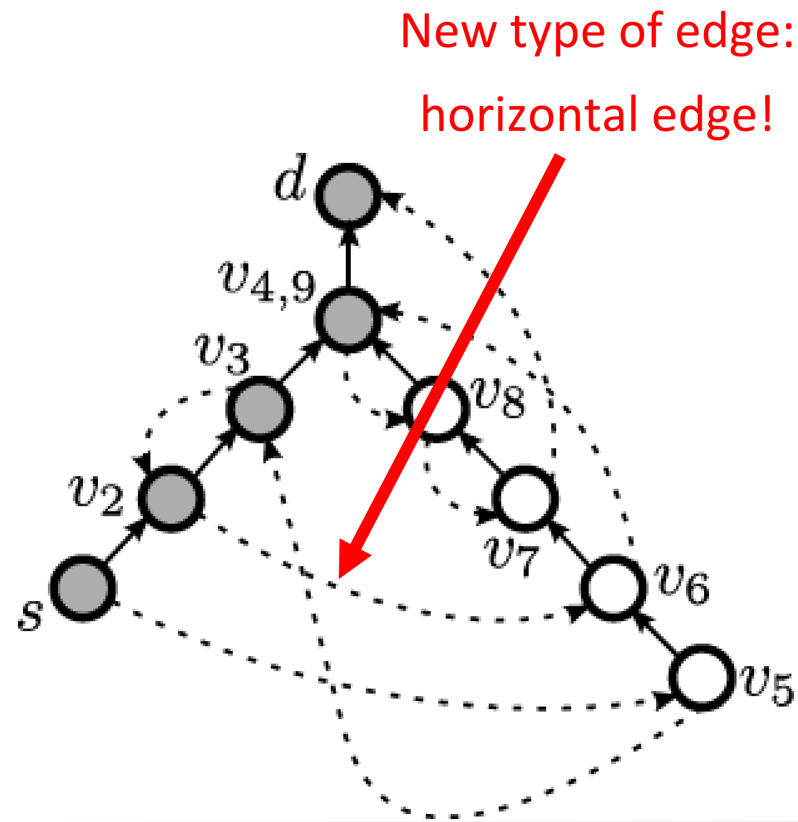
Initially: Two valid paths!

After updating v_4.

Example



Initially: Two valid paths!



After updating v_4.

Ideas of Peacock Algorithm

- **Rounds come in pairs:** Try to update (and hence merge) as much as possible in every other round

- **Round 1 (odd rounds): Shortcut**
  - Move source close to destination
  - Generate many «independent subtrees» which are easy to update!

- **Round 2 (even rounds): Prune**
  - Update independent subtrees
  - Brings us back to a chain!

Ideas of Peacock Algorithm

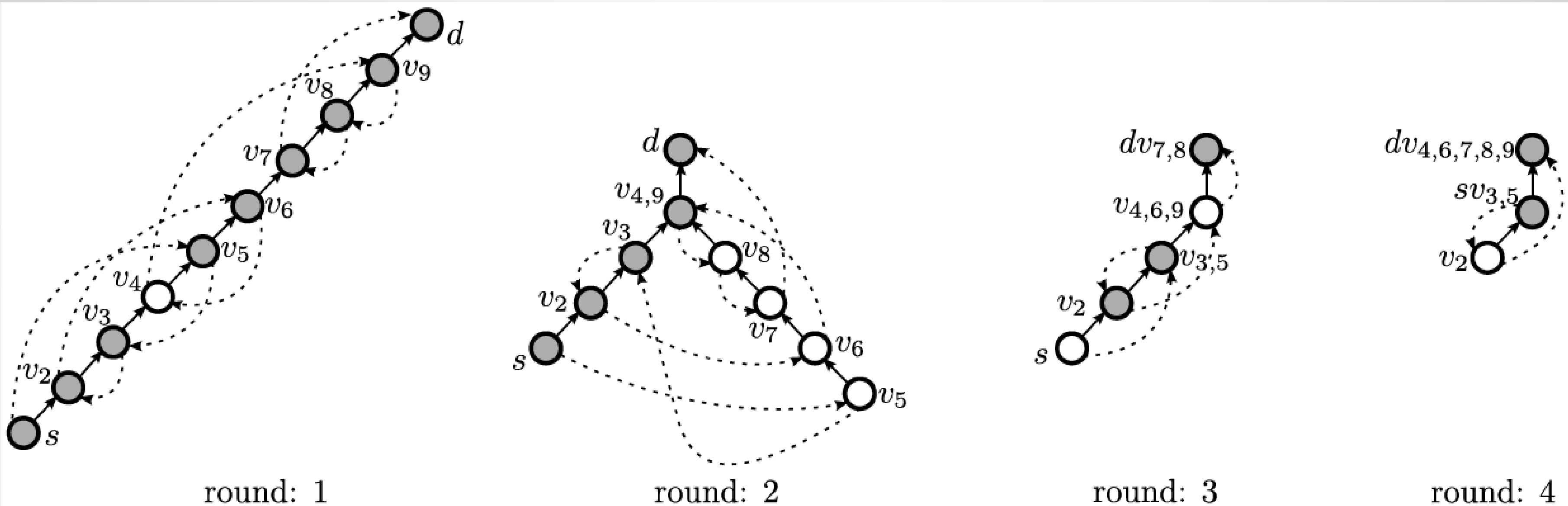
- **Rounds come in pairs:** Try to update (and hence merge) as much as possible in every other round

- **Round 1 (odd rounds): Shortcut**
  - Move source close to destination
  - Generate many «independent subtrees» which are easy to update!

Don‘t update all FF edges!

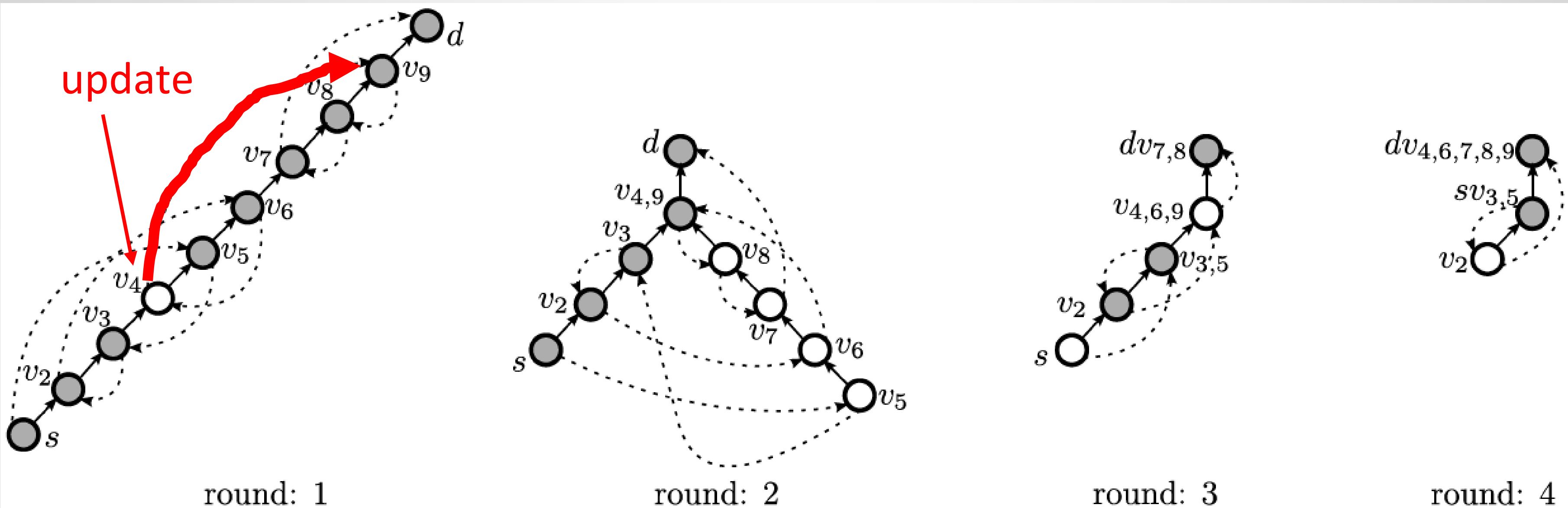
- **Round 2 (even rounds): Prune**
  - Update independent subtrees
  - Brings us back to a chain!

Peacock in Action



Shortcut　　Prune　　Shortcut　　Prune

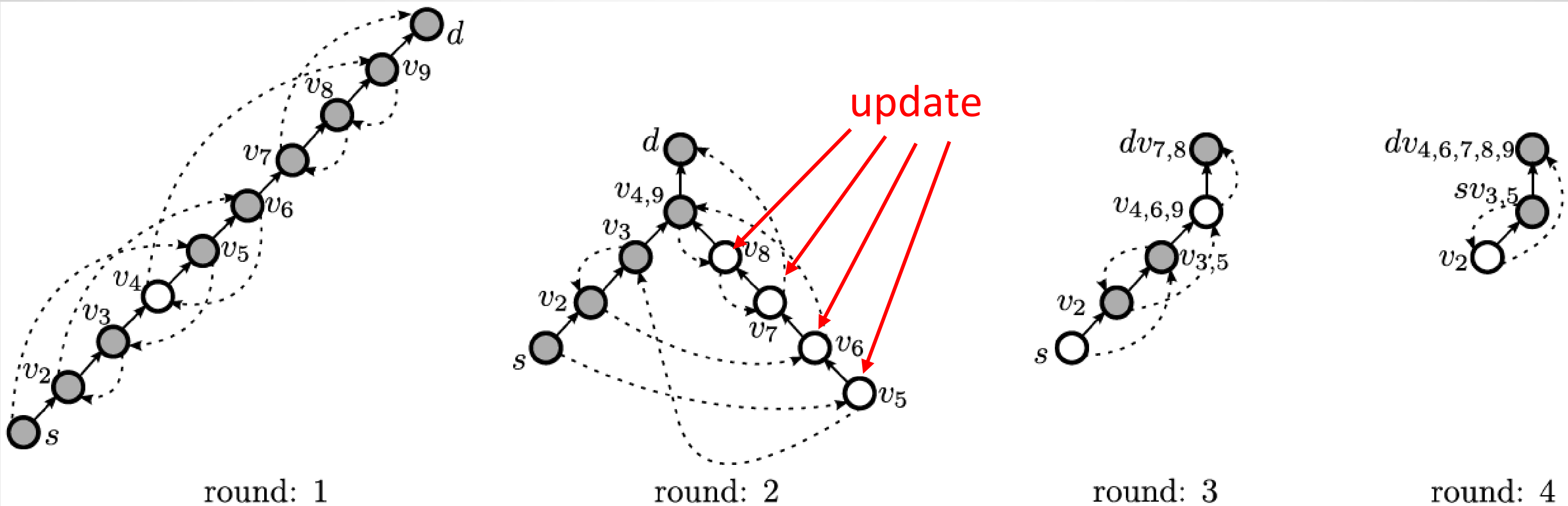
Peacock in Action



Shortcut Prune Shortcut Prune

Greedily choose far-reaching (independent) forward edges.

Peacock in Action



Shortcut

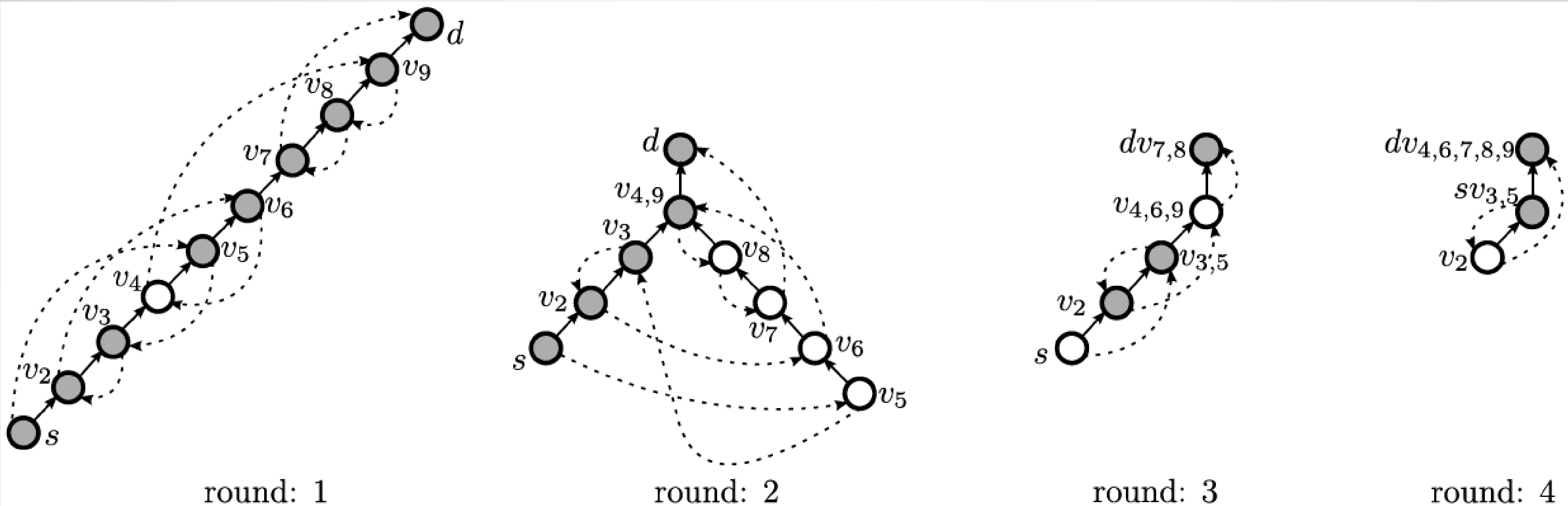
Prune

Shortcut

Prune

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

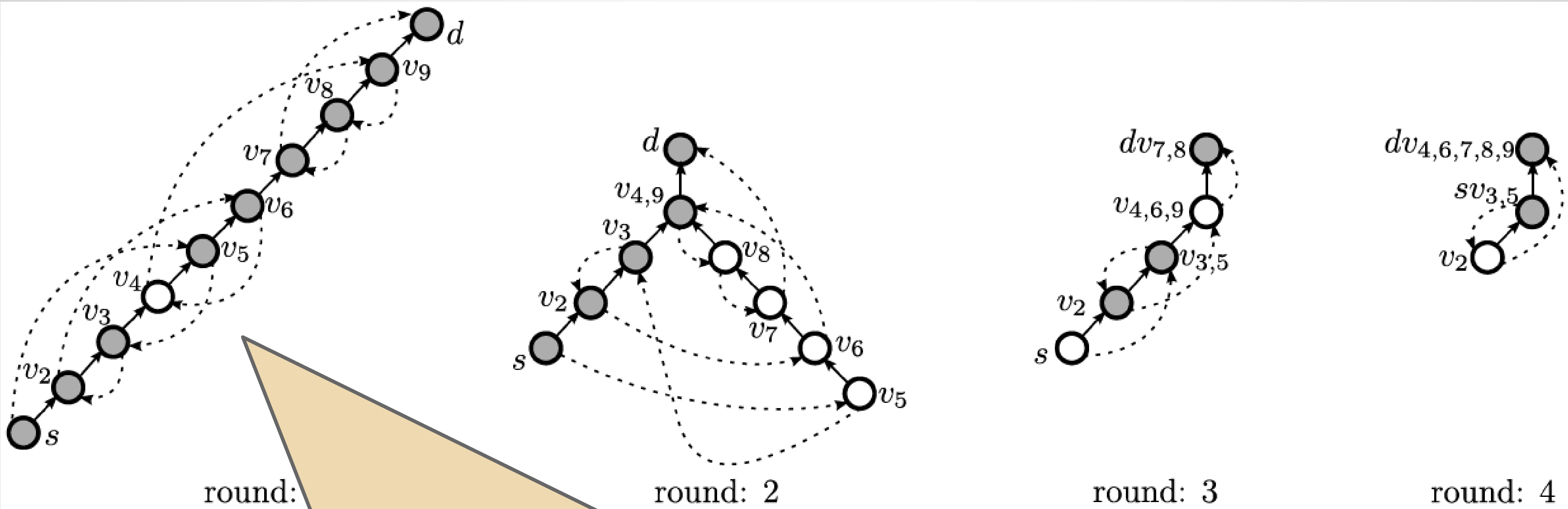
Peacock in Action



Shortcut Prune Shortcut Prune

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

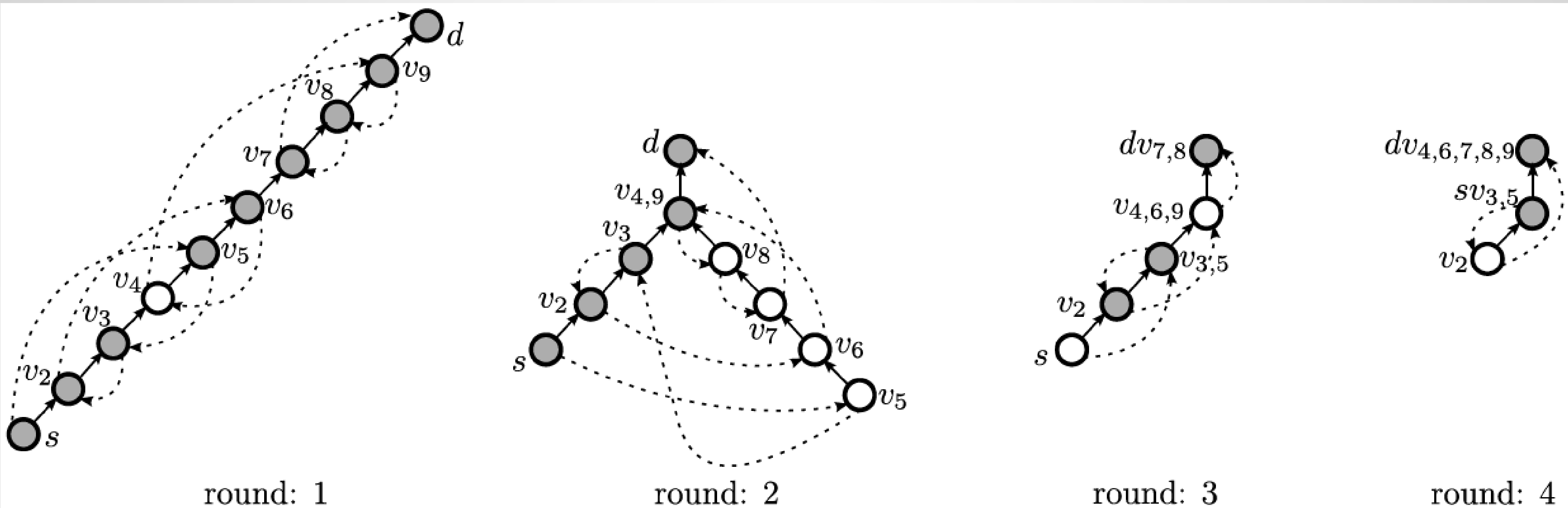
Peacock in Action



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

Prune

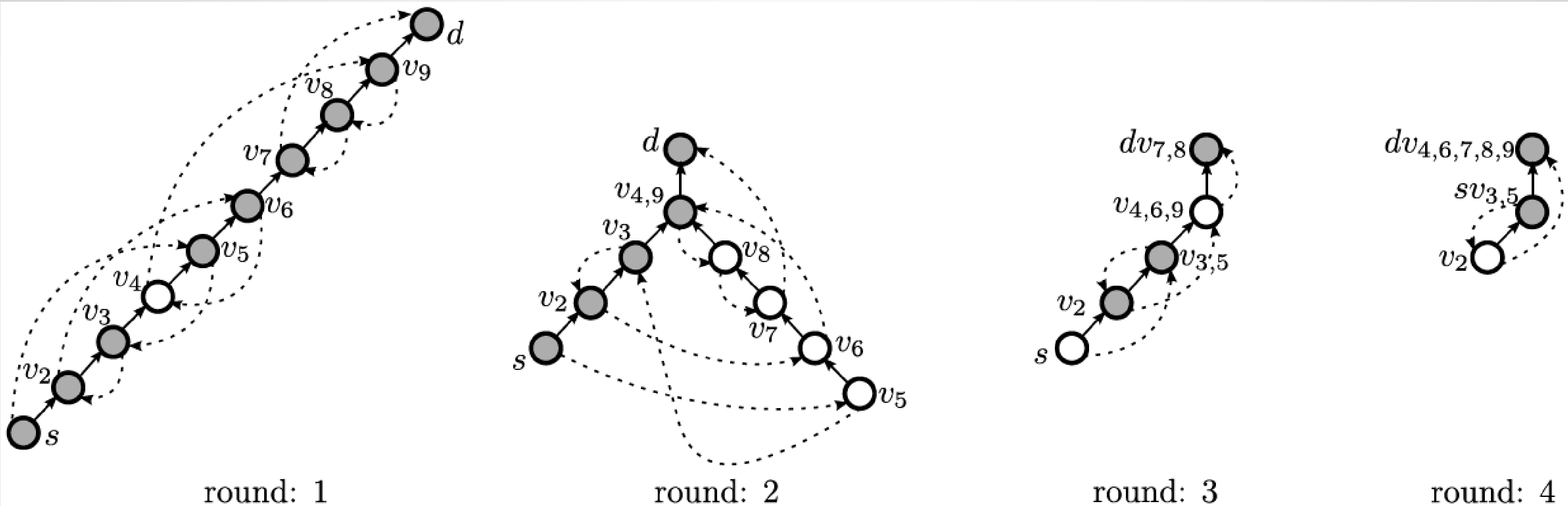
Peacock in Action



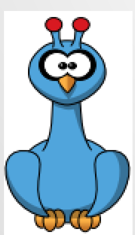
Shortcut | Prune | Shortcut | Prune

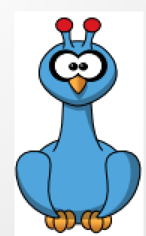
At least 1/3 of nodes merged in each round pair (shorter s-d path): logarithmic runtime!

Peacock in Action

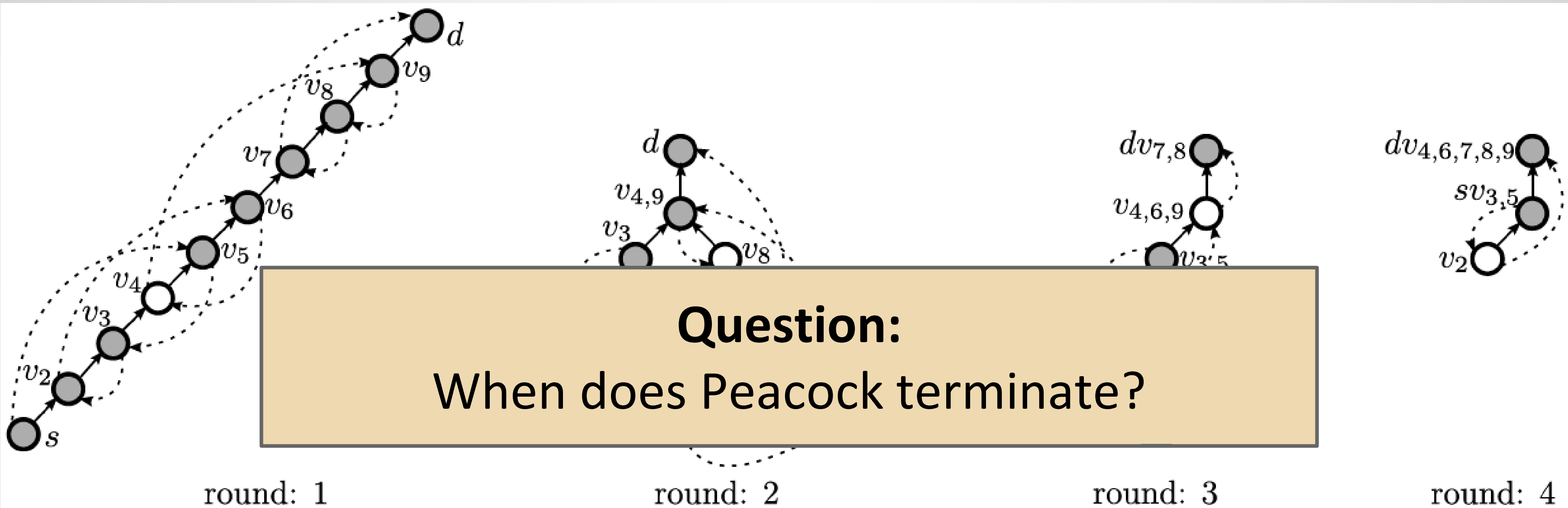


Shortcut Prune Shortcut Prune





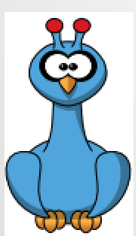

Peacock in Action



Question:

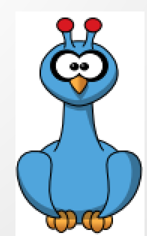
When does Peacock terminate?

Shortcut



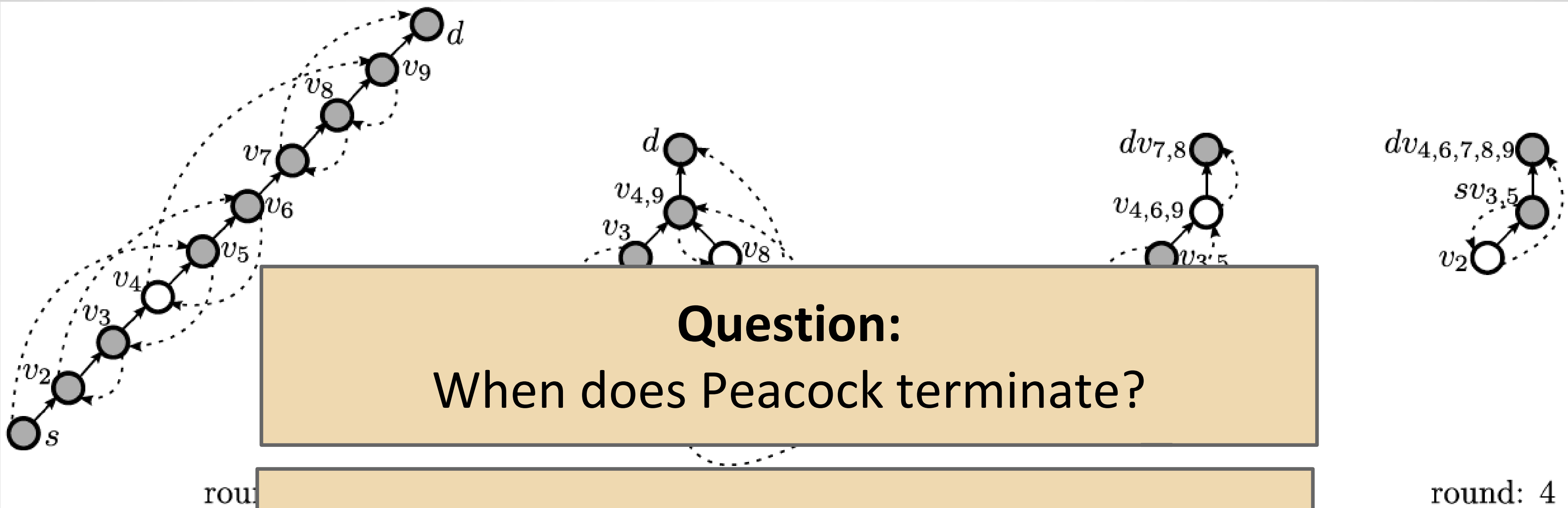
Prune

Shortcut



Prune

Peacock in Action

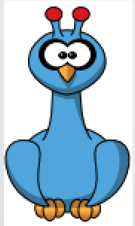


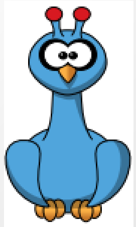
Question:
When does Peacock terminate?

Answer:
Only in odd rounds: then s-d merged

Shor

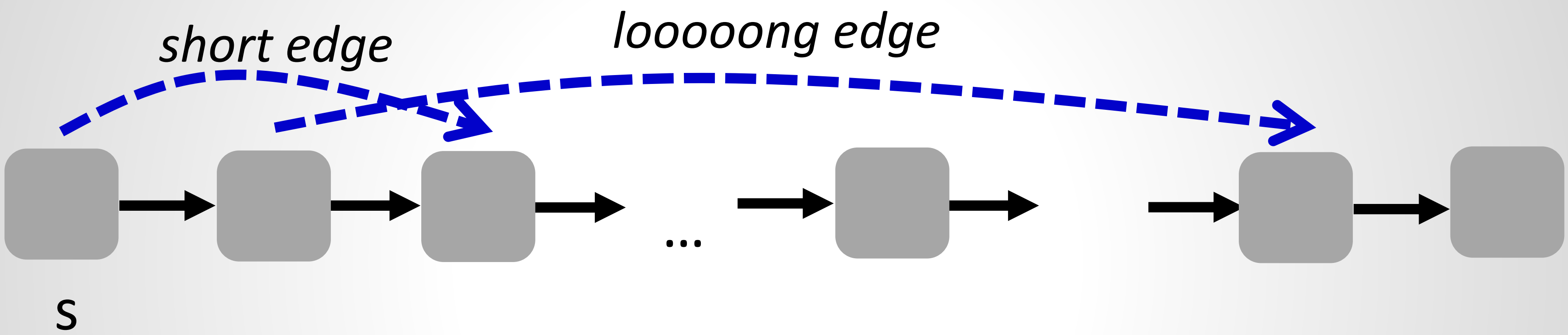
Prune



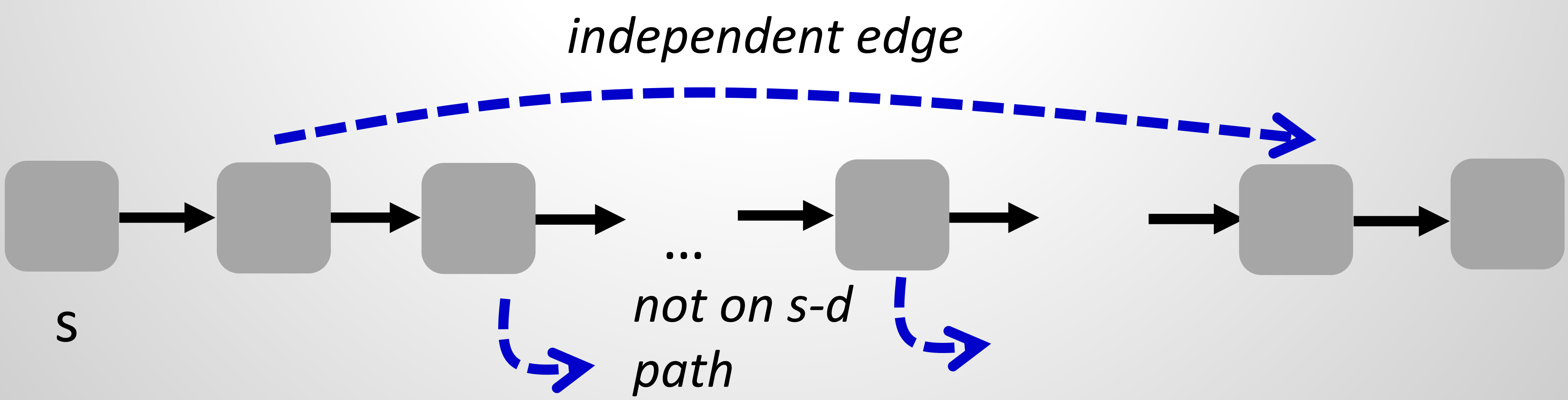


Why not update two non-independent edges?

- Don’t update all FF edges: A short edge may not reduce distance to source if it jumps over a long edge



- Can update all fwd edges starting in interval



Conclusion

- SDN offers fundamental distributed problems

- So far we know:
  - Strong LF:
    - Greedy arbitrarily bad (up to n rounds) and NP-hard
    - 2 rounds easy
    - 3 rounds hard
  - Relaxed LF:
    - Peacock solves any scenario in O(log n) rounds
    - Computational results indicate that # rounds grows
  - LF and WPE may conflict

Thank you!

And thanks to co-authors: Arne Ludwig, Jan Marcinkowski

as well as Marco Canini, Damien Foucard, Petr Kuznetsov, Dan Levin, Matthias Rost, Jukka Suomela

and more recently Saeed Amiri, Szymon Dudycz, Felix Widmaier

Own References

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.

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.