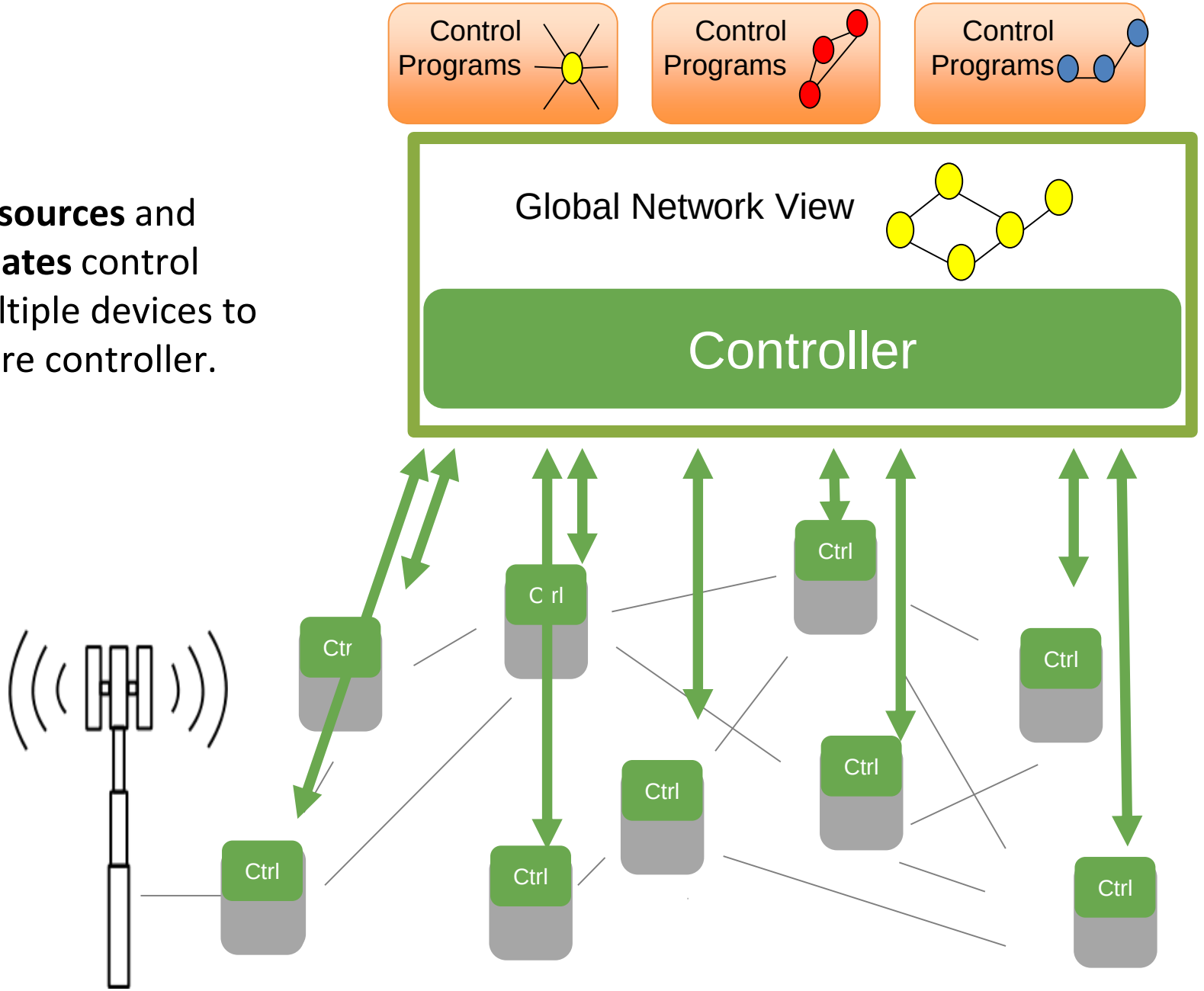
GIAN Course on Distributed Network Algorithms

Software-Defined and Virtualized Networks

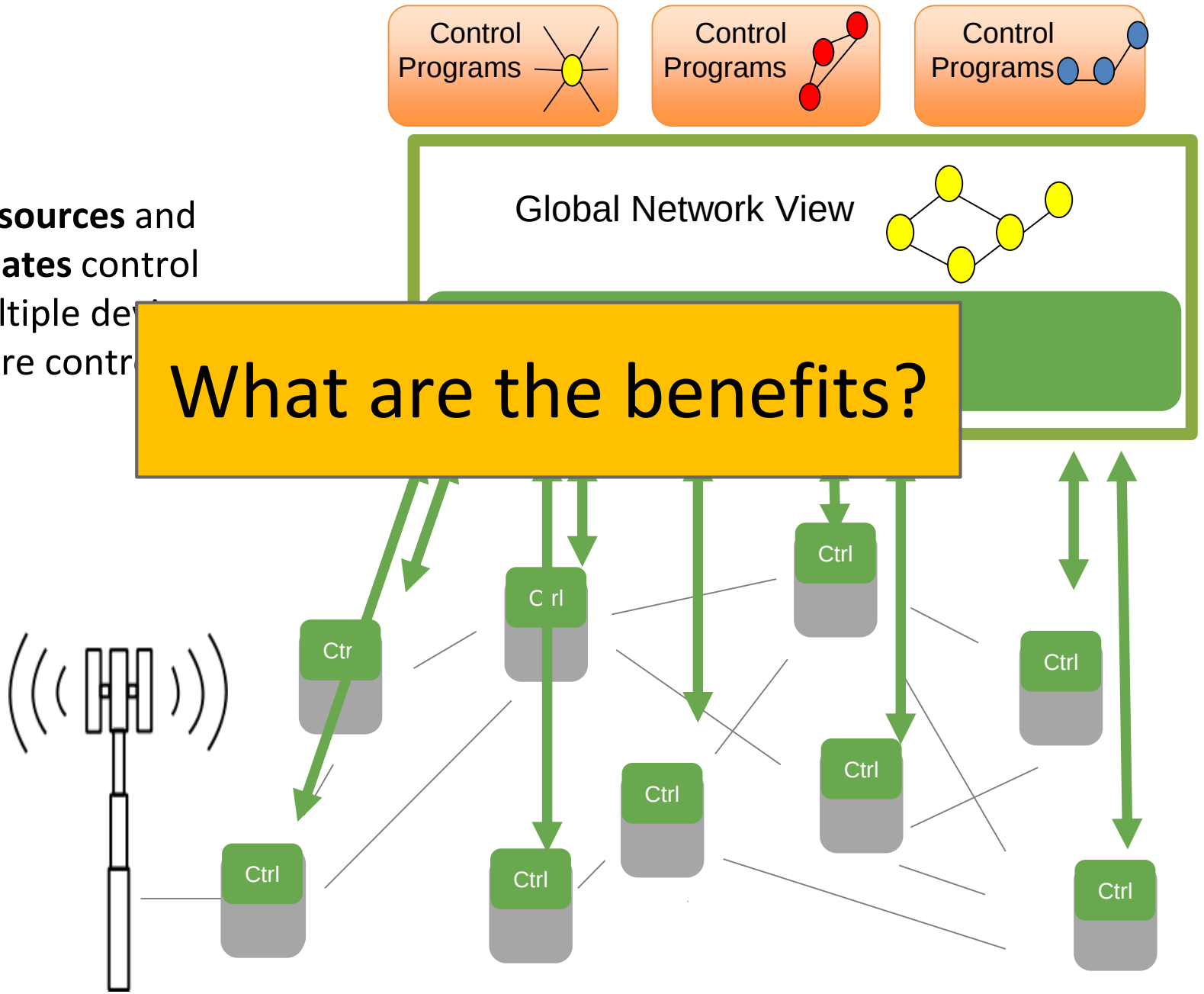
Software-Defined Networking

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

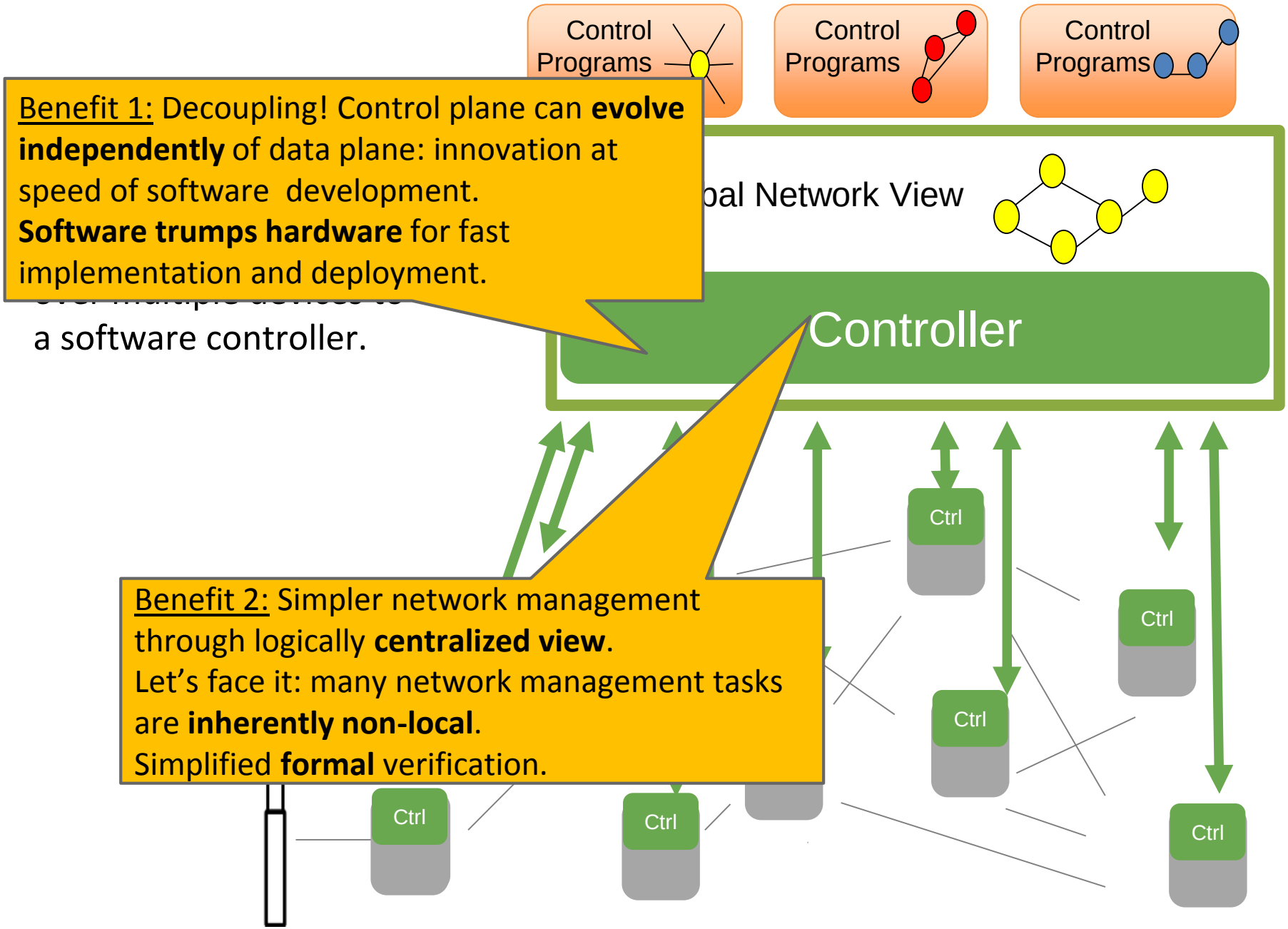


SDN in a Nutshell

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



SDN in a Nutshell



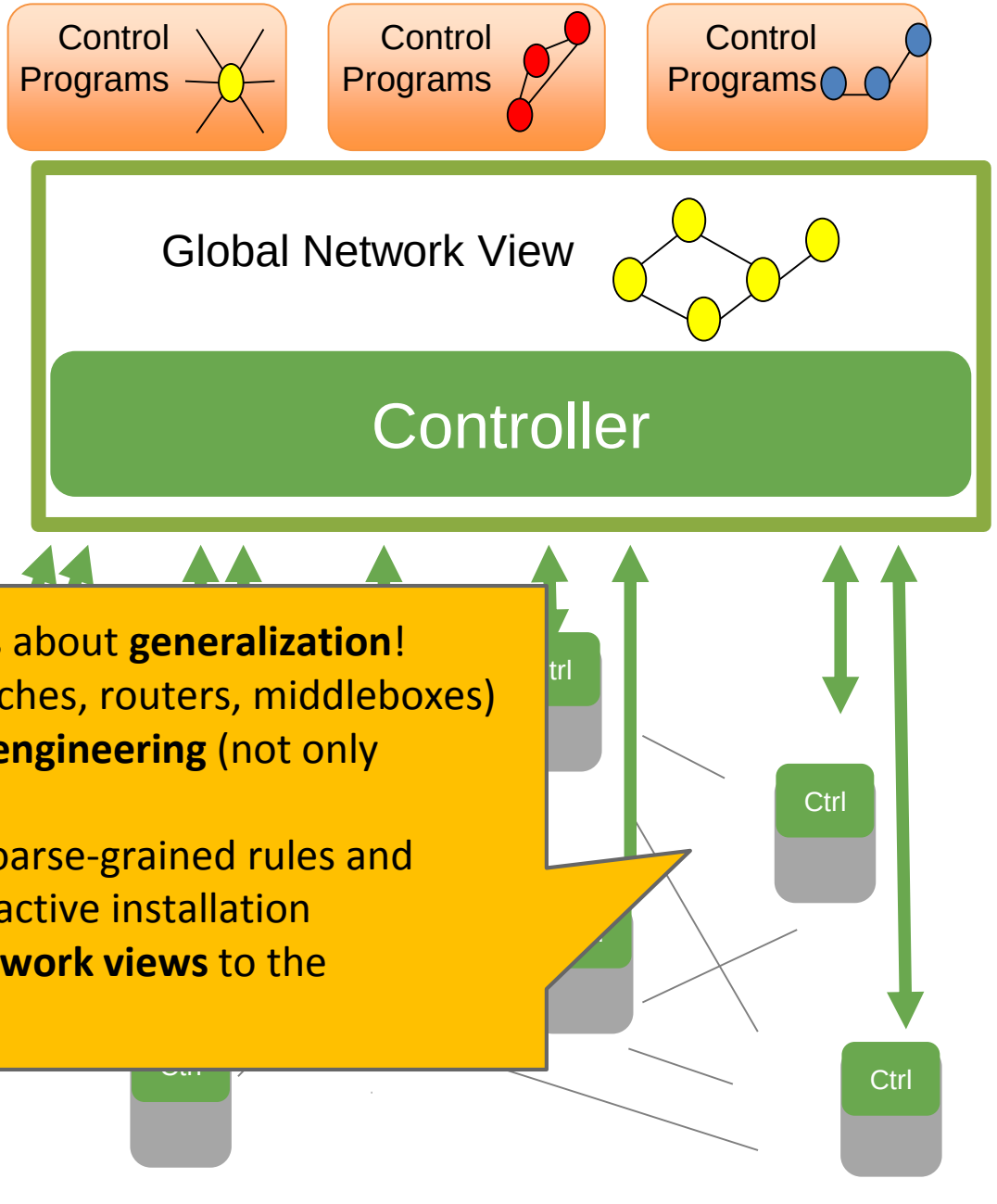
Benefit 1: Decoupling! Control plane can **evolve independently** of data plane: innovation at speed of software development.
Software trumps hardware for fast implementation and deployment.

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.

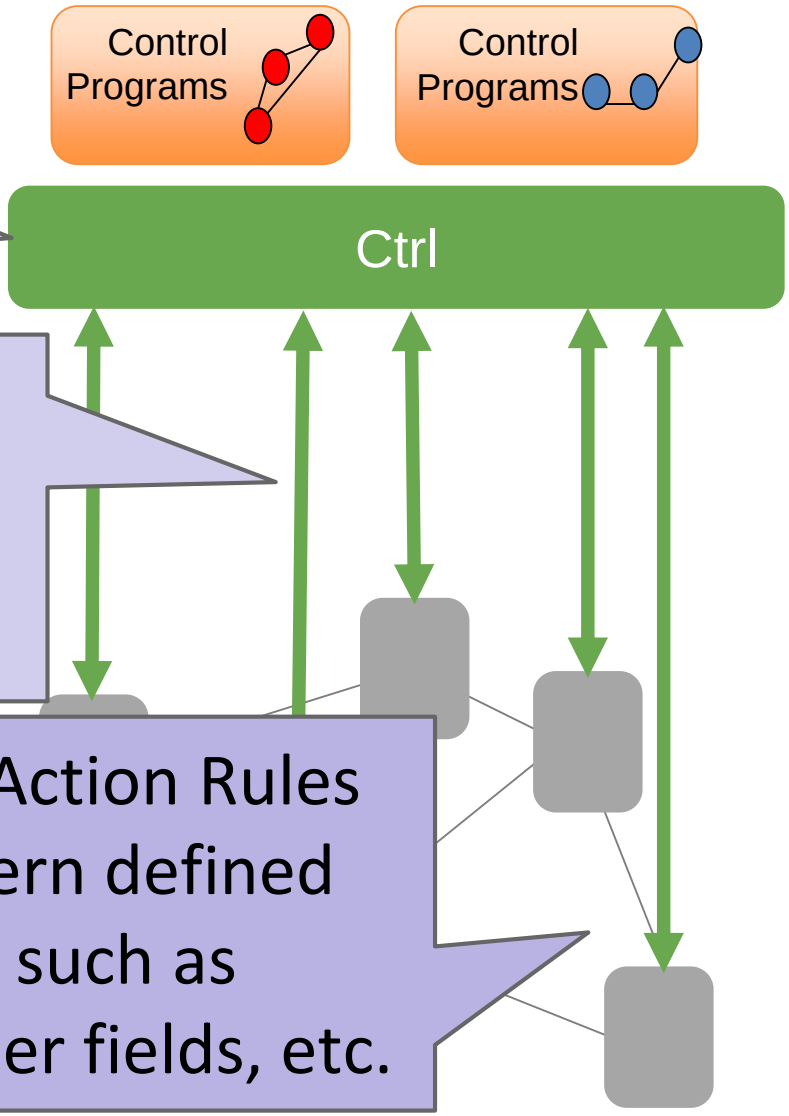


Benefit 3: Standard API OpenFlow is about **generalization**!

- 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

OpenFlow 101

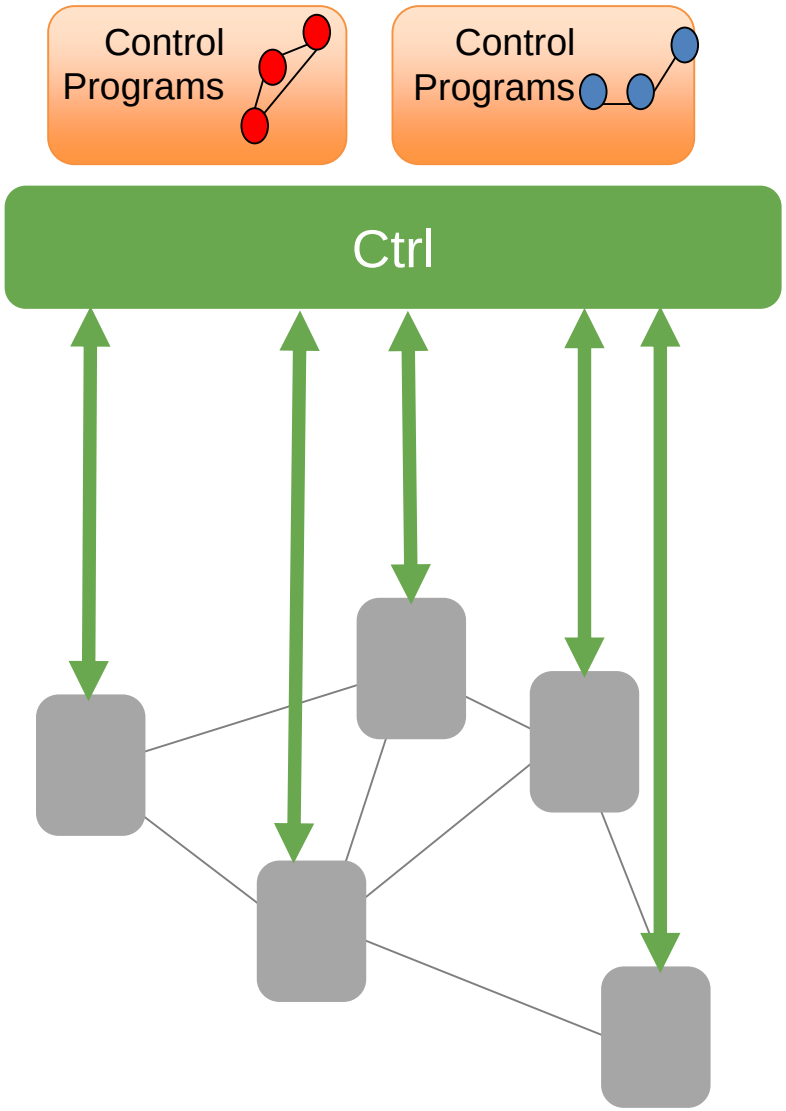
Controller installs and modified match action rules as needed (FLOW-MOD).



If no rule matches packet, it is forwarded to controller: decides what to do (PACKET-IN).

Switch stores set of Match-Action Rules (in flow tables): Match pattern defined over packet header, actions such as forward, drop, change header fields, etc.

SDN: Algorithms *with a fundamental twist!*

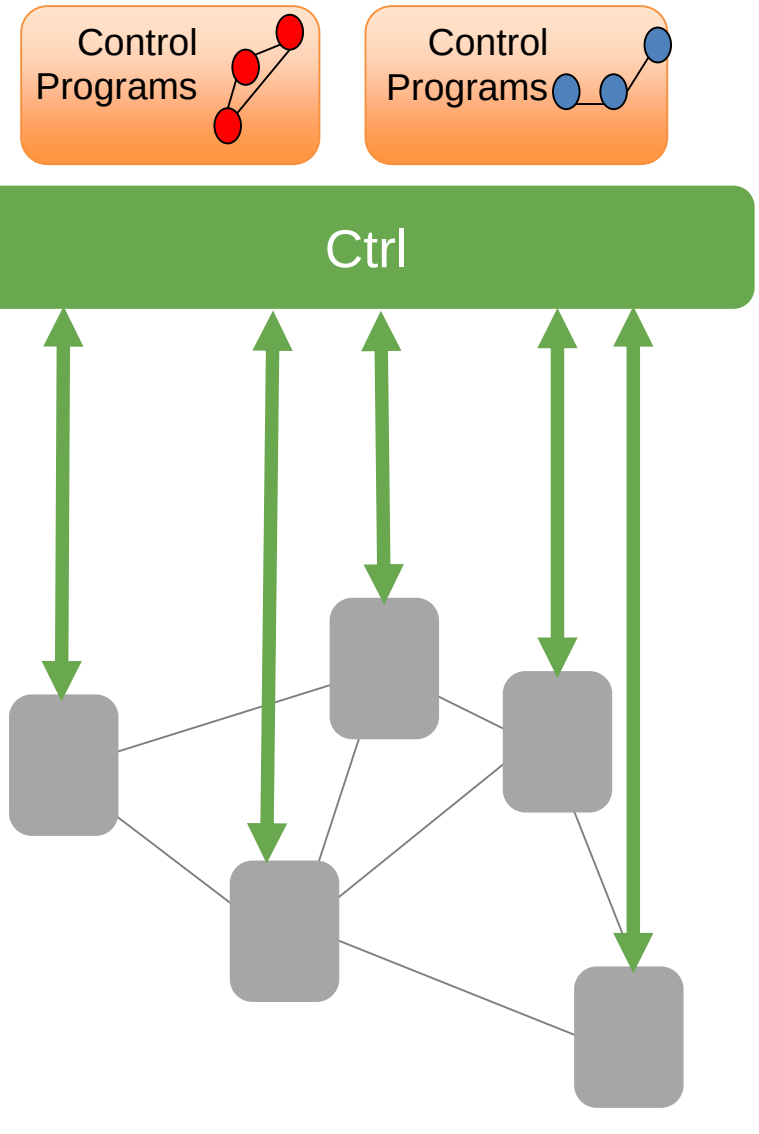


SDN: Algorithms *with a fundamental twist!*

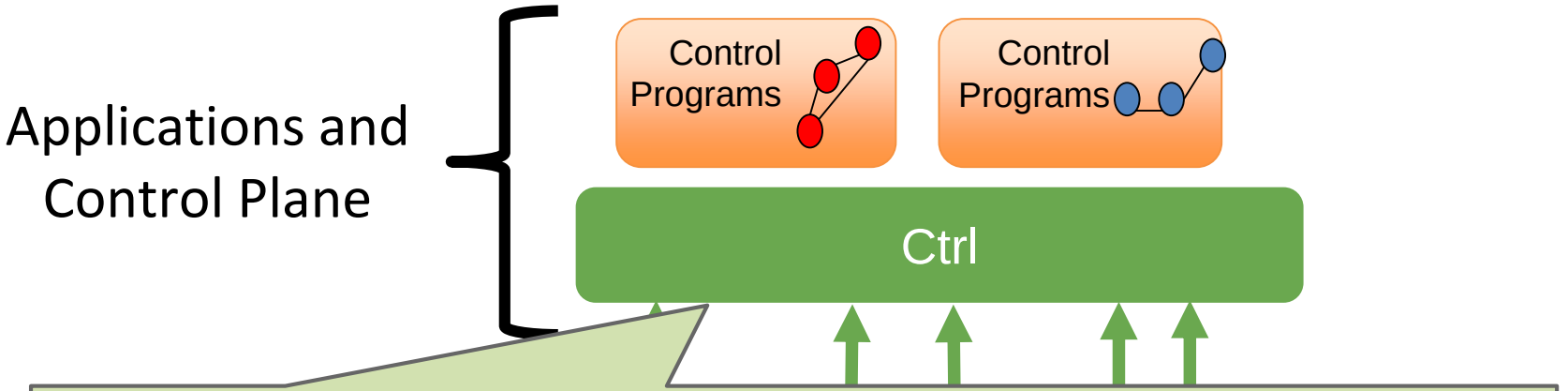
Applications and Control Plane

… and regarding decoupling / inter-connect!

Data Plane



SDN: Flexiblities and Constraints



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.

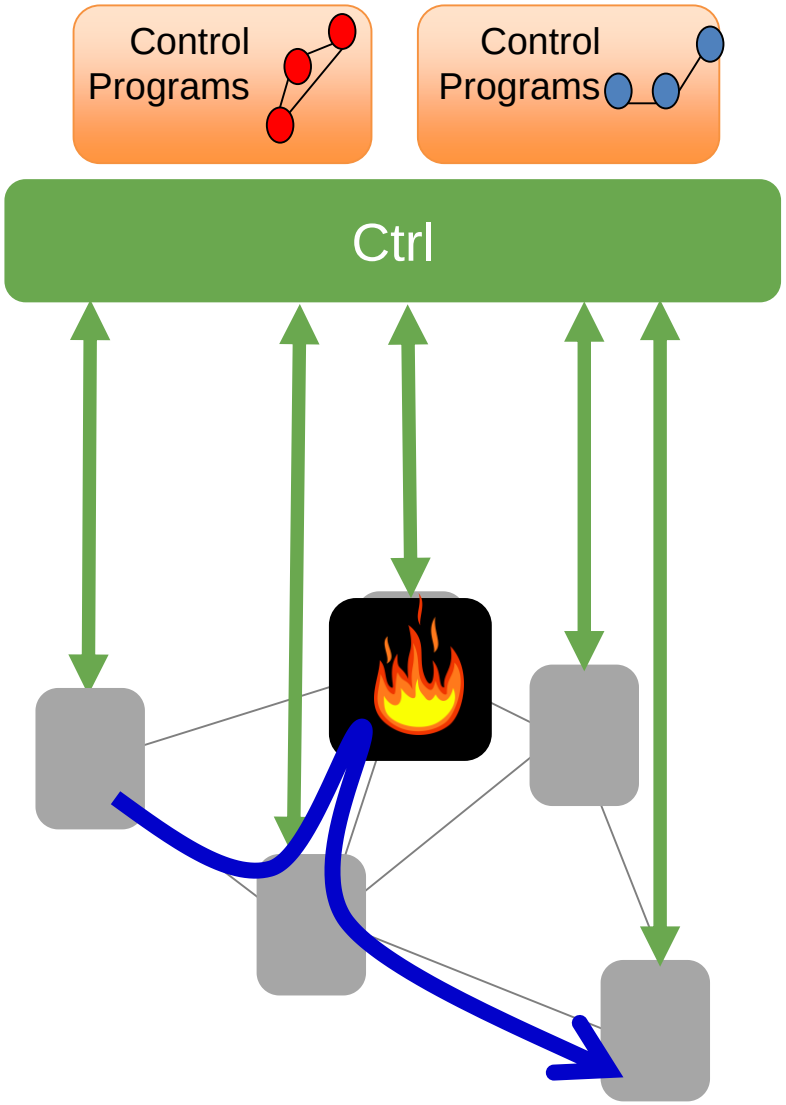
Dat

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.

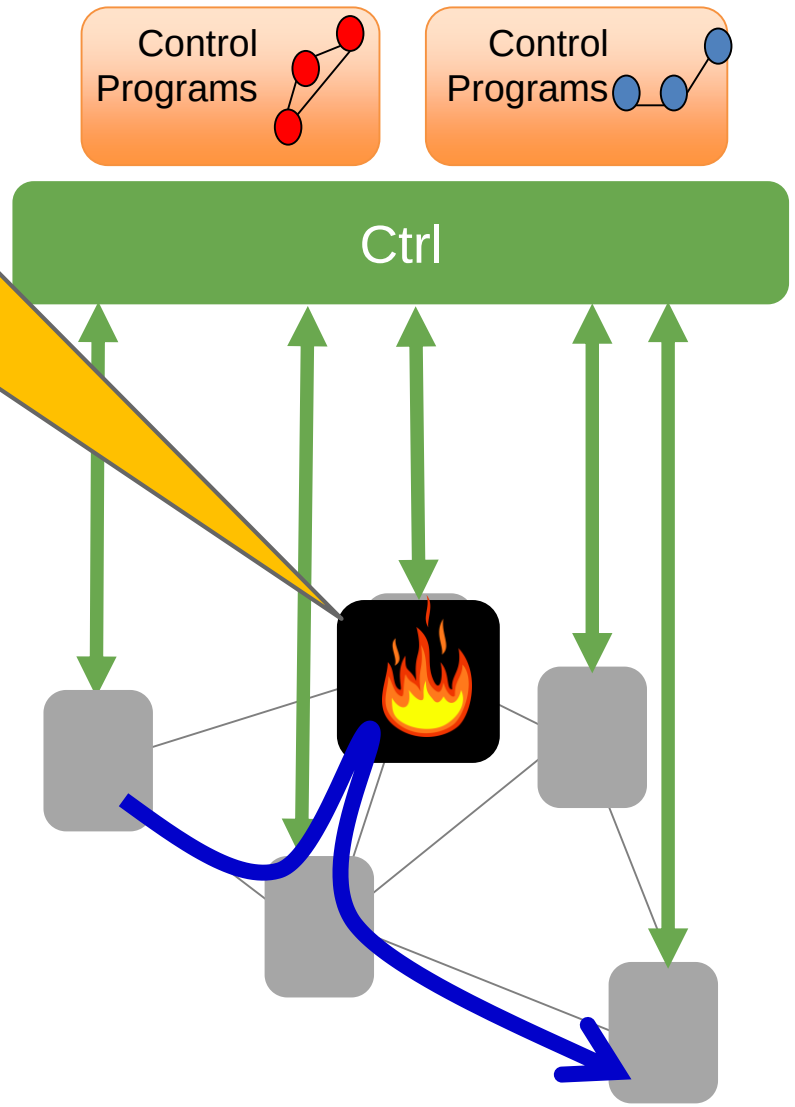
- *SDN twist*: more general/flexible!
  - **Non-shortest** paths and more
  - Enables **complex network services:** steer traffic through middleboxes i.e. **waypoints** (firewall, proxy etc.): paths may contain **loops**!
  - More than independent routing per segment: **none-or-all segment admission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



Applications: Algorithms *with a twist!*

- 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**?

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!



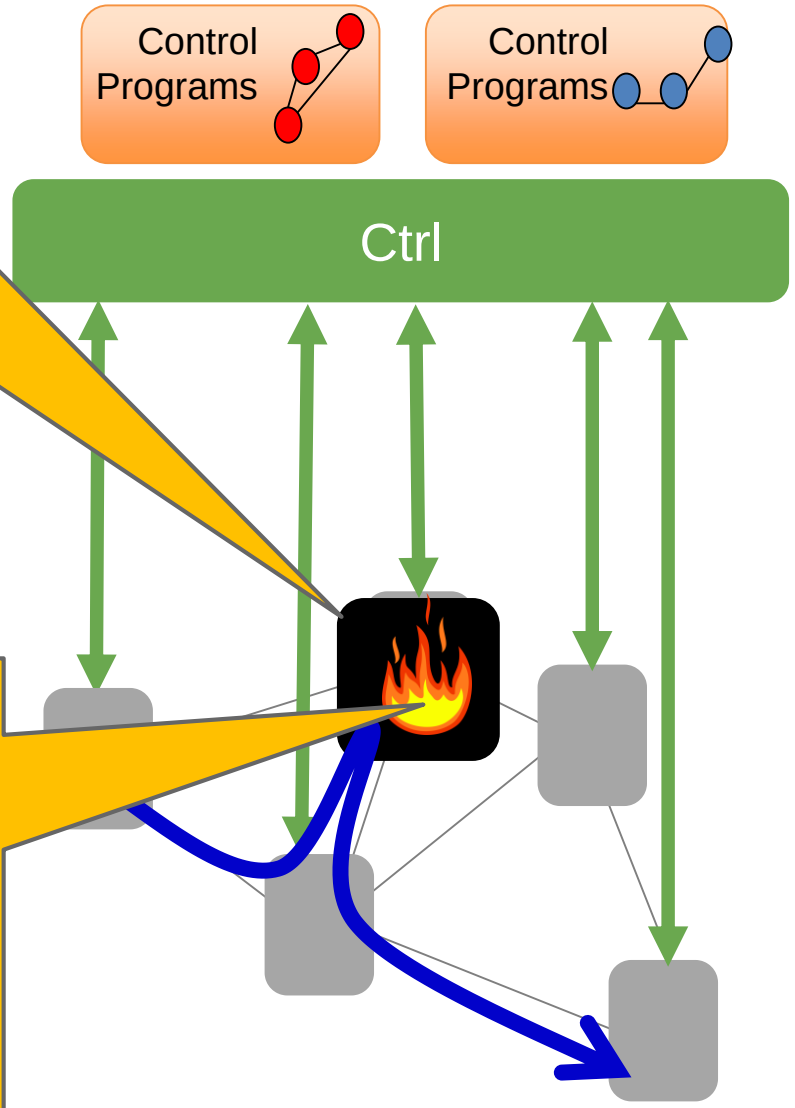
Applications: Algorithms *with a twist!*

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

decompose complex requests?

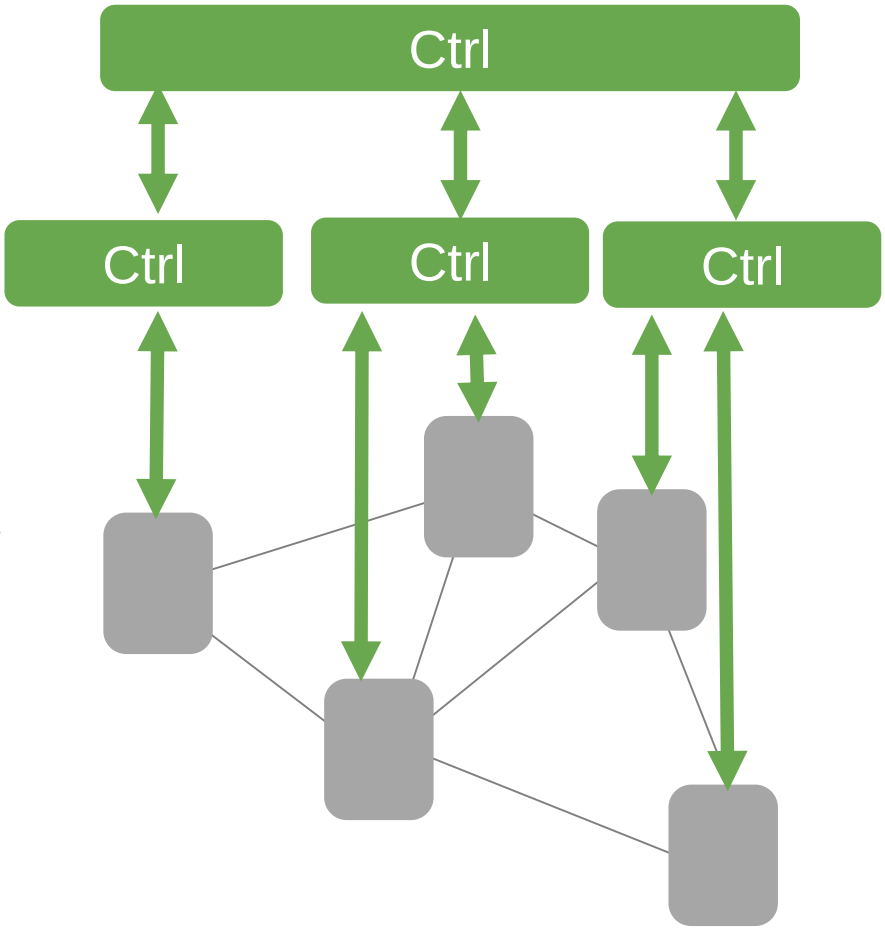
Optionally *NFV twist*: where to place NFV (or hybrid SDN)? Facility location / capacitated dominating set, *but*: not distance to but distance *via function(s)* matters!

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



Control Plane: Algorithms *with a twist!*

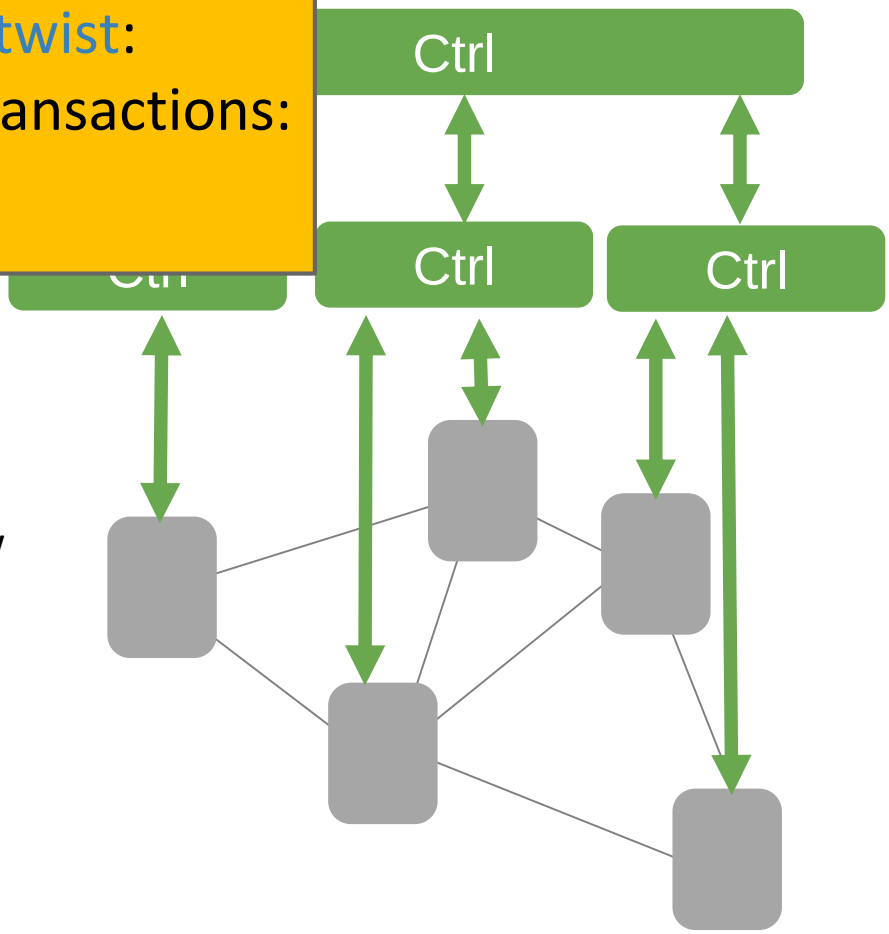
- Reduce **latency and overhead**: What can be computed locally?
  - Routing vs heavy-hitter detection?
  - LOCAL model! Insights apply: **verification vs optimization**



- *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!*

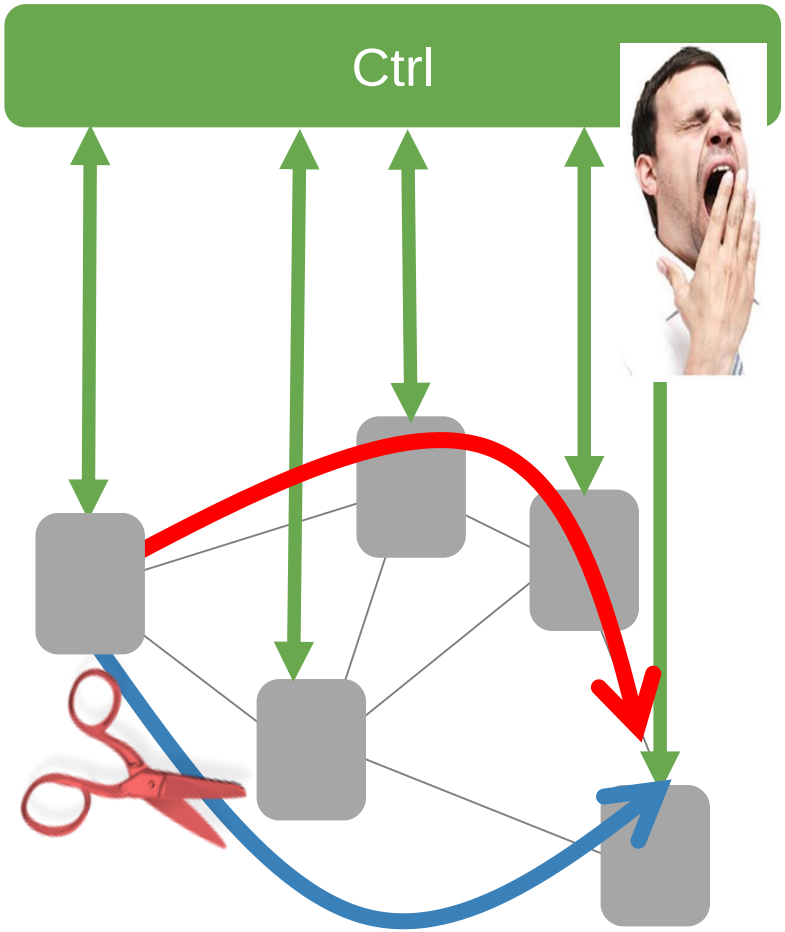
- How to make control plane robust? **Software transactional memory** problem: **network configuration = shared memory,** updates = **transactions**, but with a twist: flows are uncontrolled, real-time transactions: do not abort! (And not only read!)
- *SDN twist: pre-processing!*
  - Hard in LOCAL: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



Data Plane: Algorithms *with a twist!*

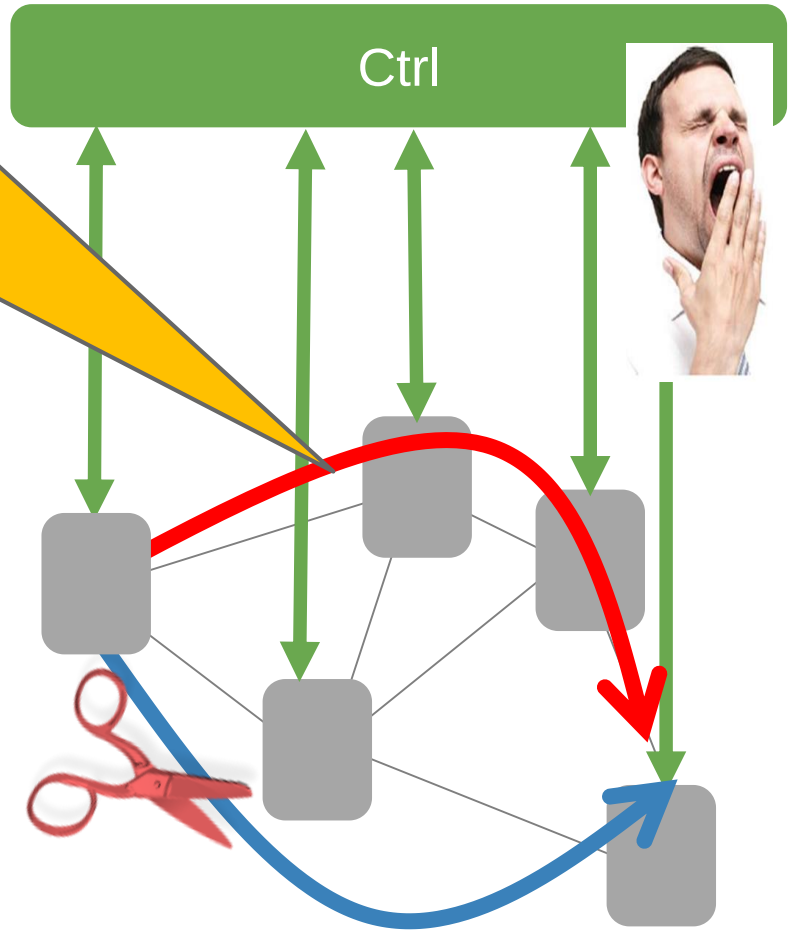
- Even in SDN: Keep some functionality in the data plane!
  - E.g., for **performance**: OpenFlow local fast failover: 1st line of defense

- 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**!



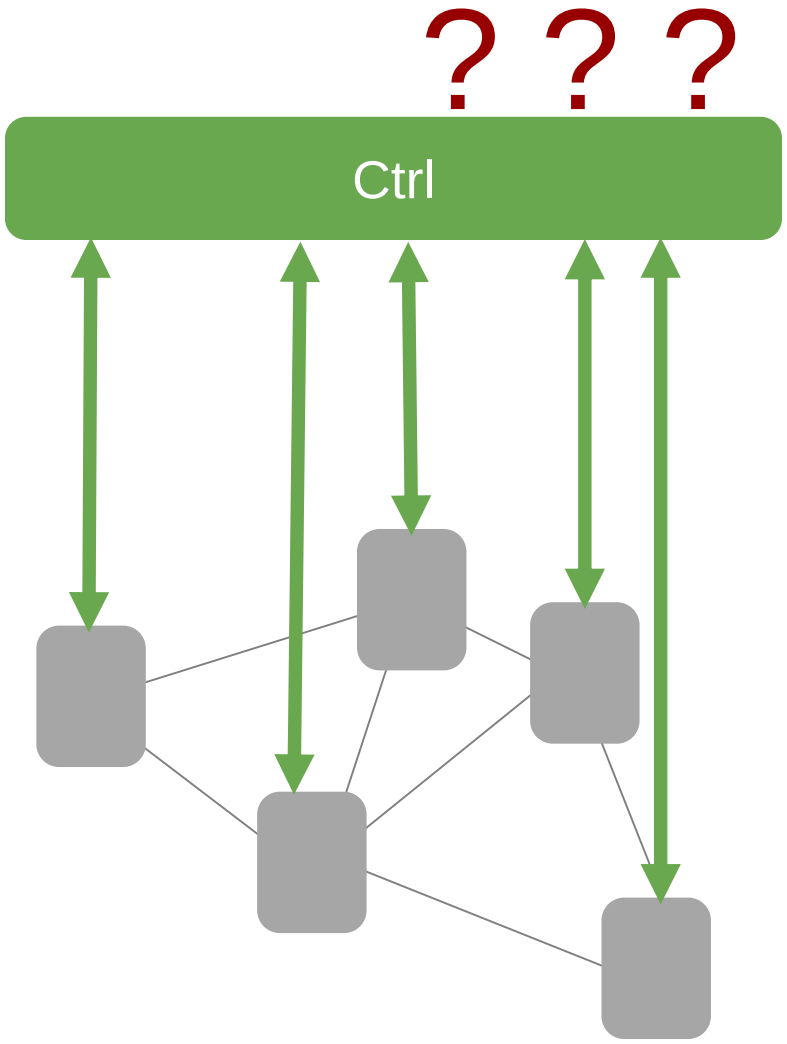
Data Plane: Algorithms *with a twist!*

- 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**!



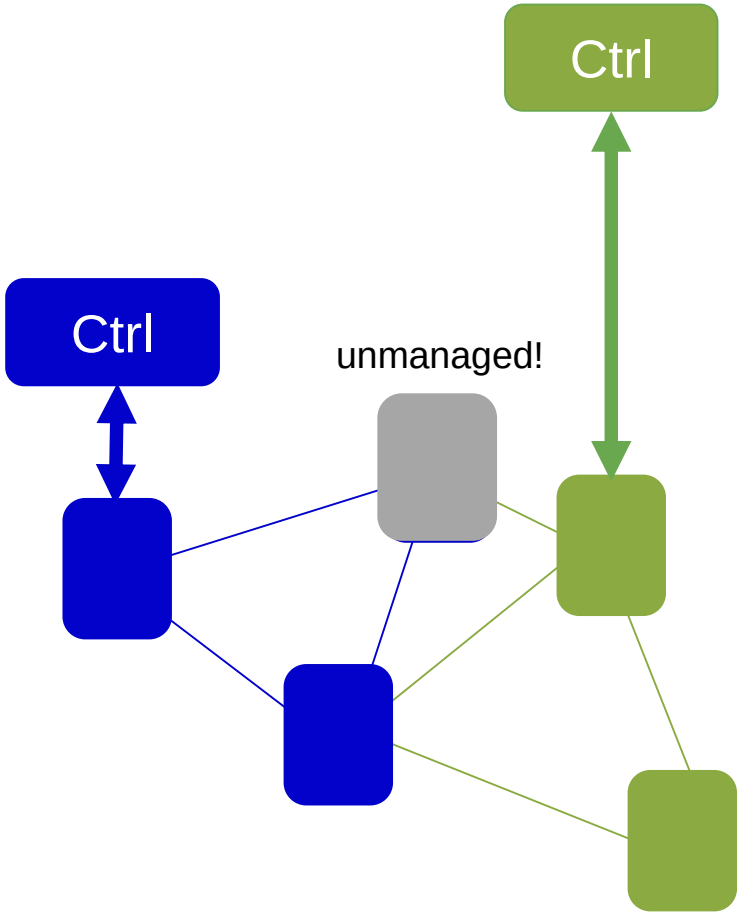
Decoupling: Algorithms *with a twist!*

- 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!



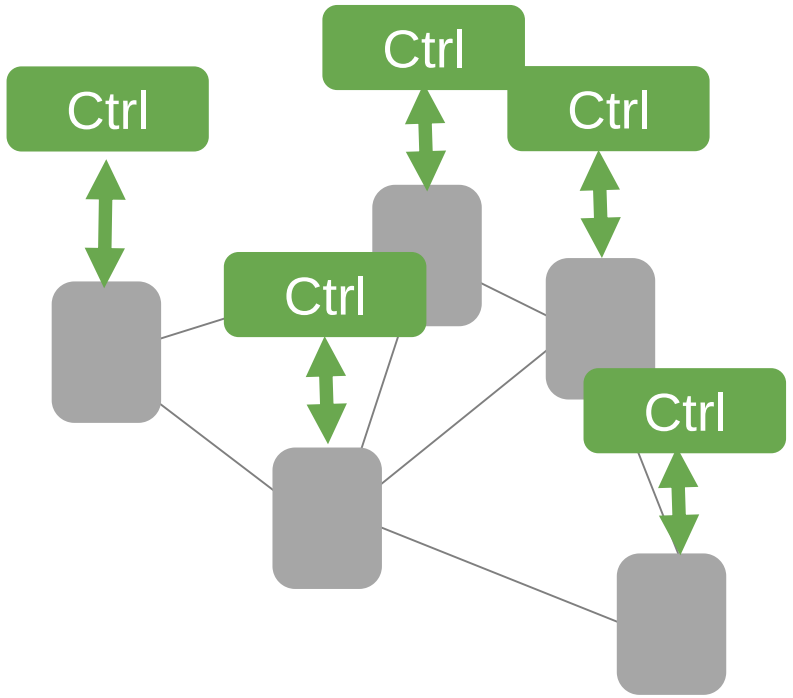
Decoupling: Algorithms *with a twist!*

- 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

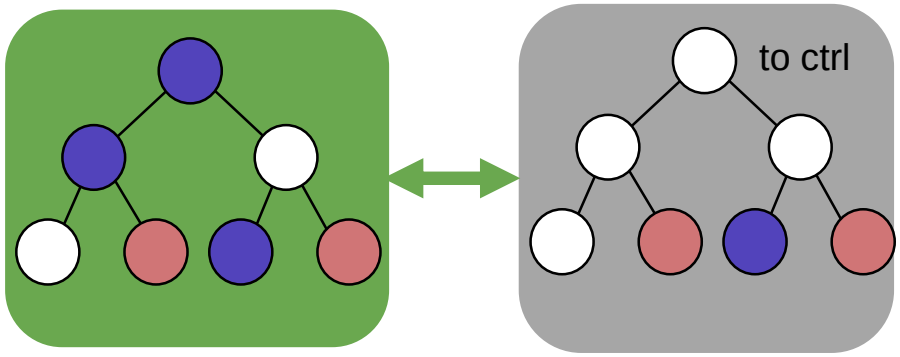


Decoupling: Algorithms *with a twist!*

- 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 ☺

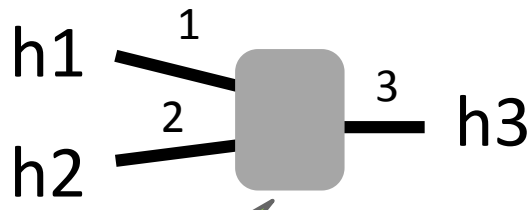


- Online paging *with SDN twist*
  - Forwarding semantic: **largest common prefix forwarding**, i.e., dependencies: only offload **root-contiguous** set in trie
  - Can do **bypassing**



Example: SDN MAC Learning Done Wrong

Hello world in SDN: MAC learning.

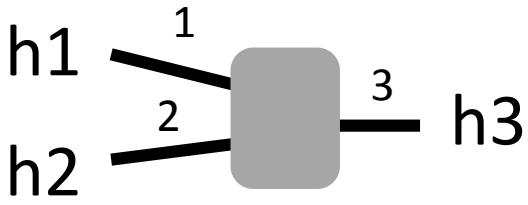


Simple state maching: if destination not known, flood. If known, send directly. And try to learn!

MAC Learning before SDN

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:



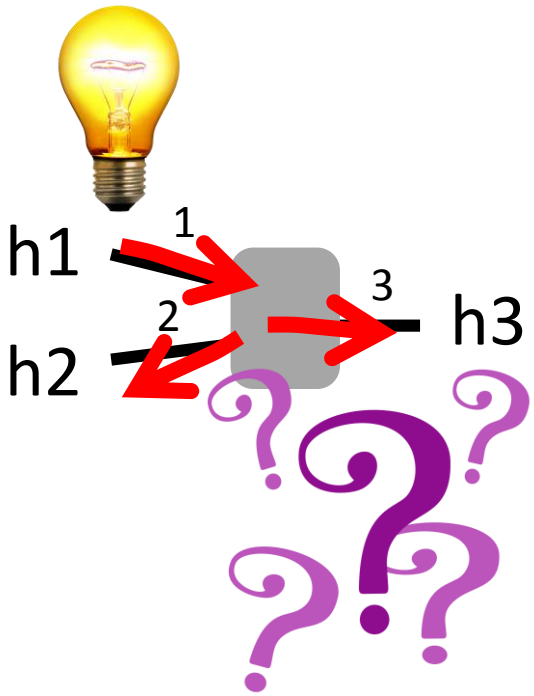
Thanks to Jennifer Rexford for example!

MAC Learning before SDN

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!

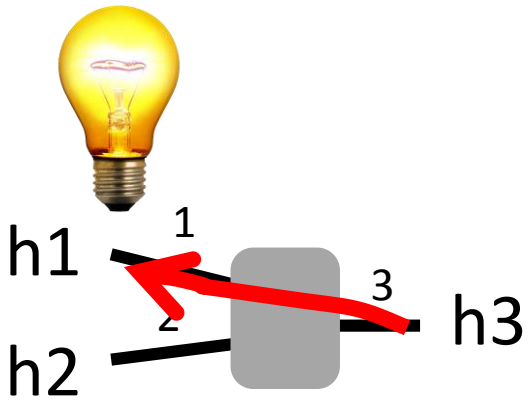
MAC Learning before SDN

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!

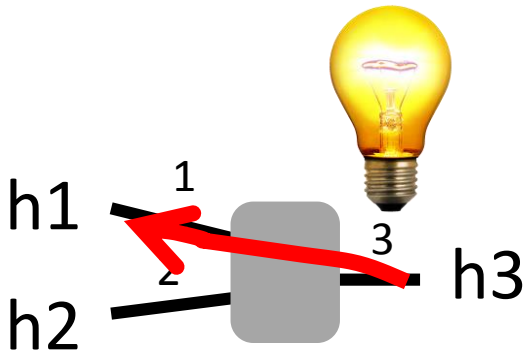
MAC Learning before SDN

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!

MAC Learning before SDN

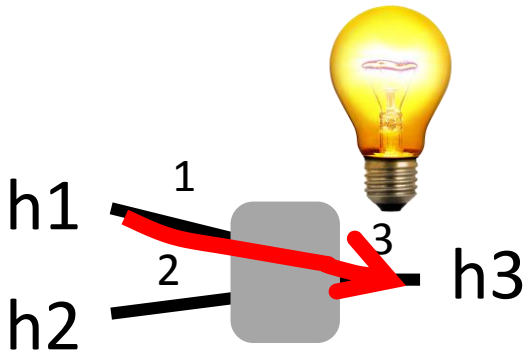
Already updating a single switch from a single controller is non-trivial!

- 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

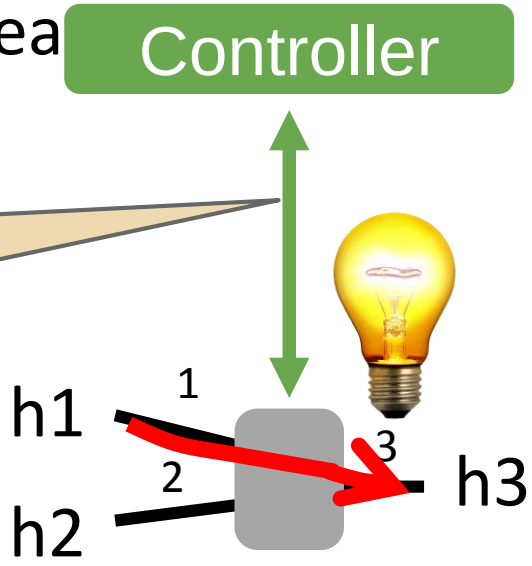


Thanks to Jennifer Rexford for example!

MAC Learning *with SDN*

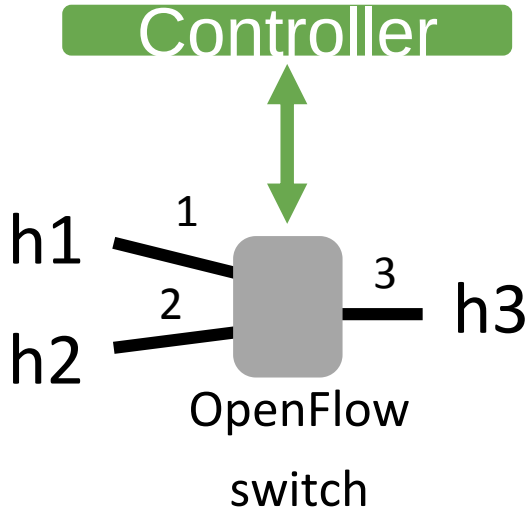
Already updating a single switch from a single controller is non-trivial!

- Fundamental networking task: MAC lea
  - Flood packets sent to unknown destinations
  - Now: how to do via controller?
    Install rules as you learn!
    And match on host address and port.
  - **flood, learn (h1,p1)**
  - Install rules as you learn!
    And match on host address and port.
  - h1 sends to h3:
  - **forward to p3**



Thanks to Jennifer Rexford for example!

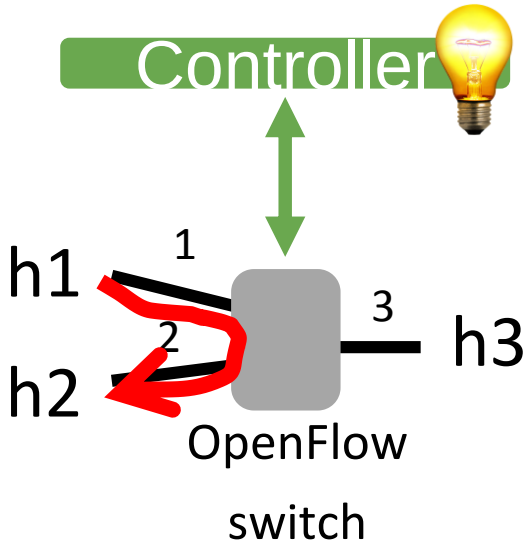
Example: SDN MAC Learning Done Wrong



- Initial rule *: Send everything to controller

- What happens when h1 sends to h2?

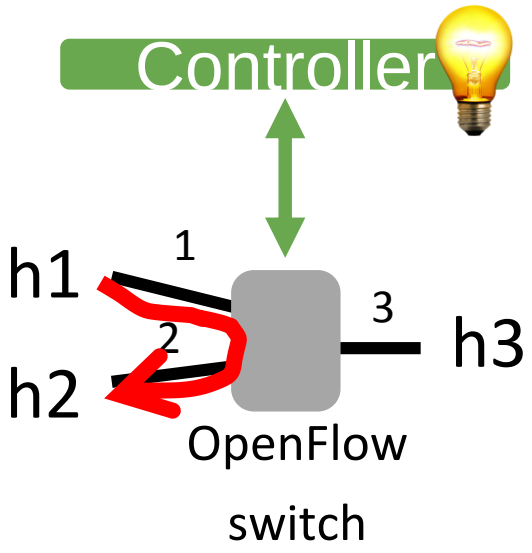
Example: SDN MAC Learning Done Wrong



- 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

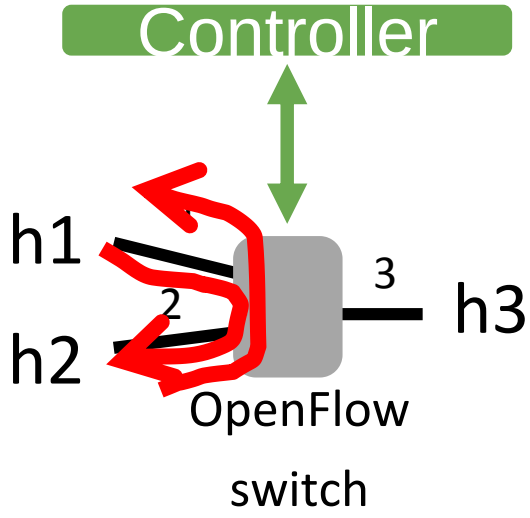
Pattern	Action
*	Send to controller

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

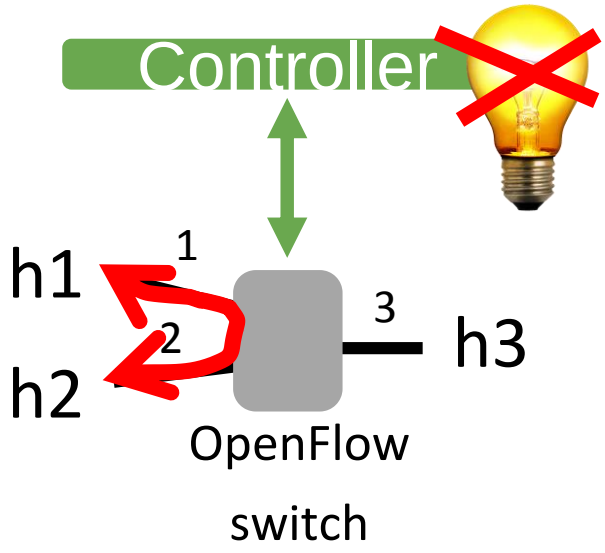
Pattern	Action
*	Send to controller

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

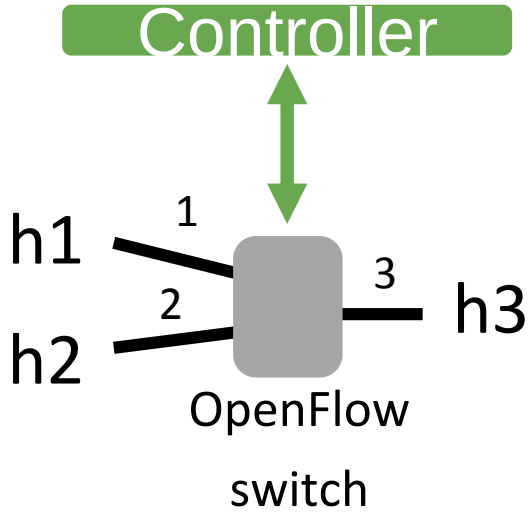
Pattern	Action
*	Send to controller

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

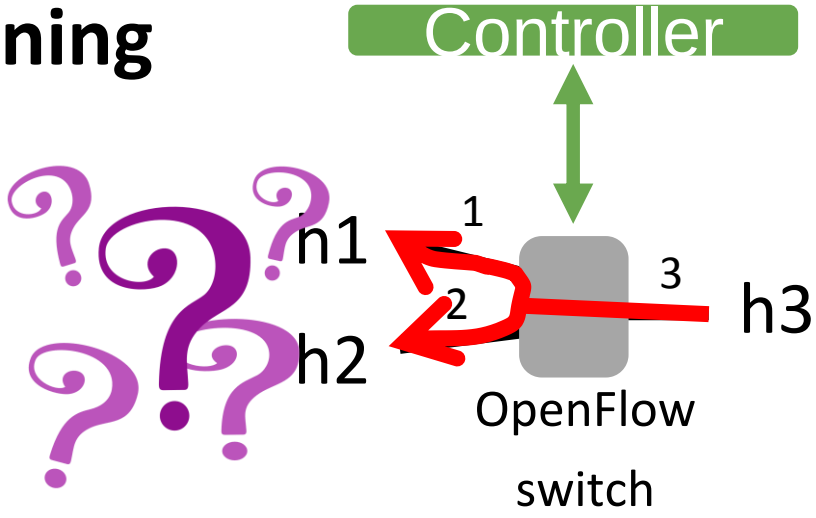
Pattern	Action
*	Send to controller

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

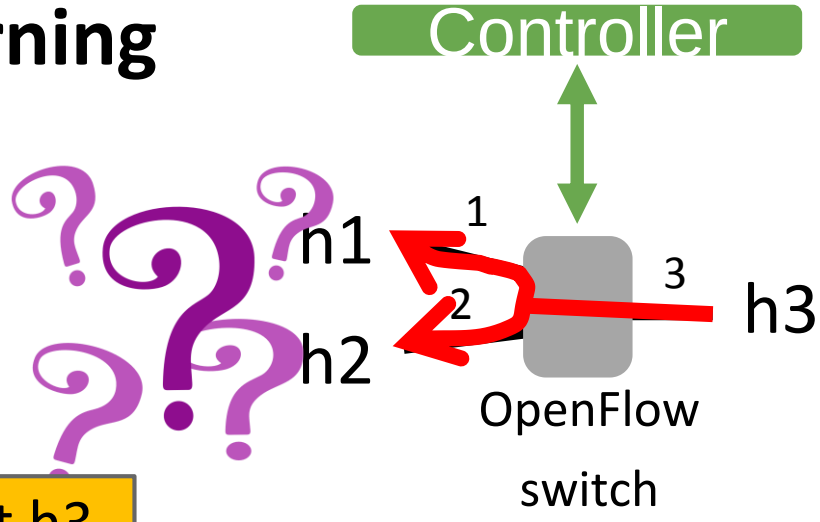
Pattern	Action
*	Send to controller

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 for destination h2.

Example: SDN MAC Learning Done Wrong



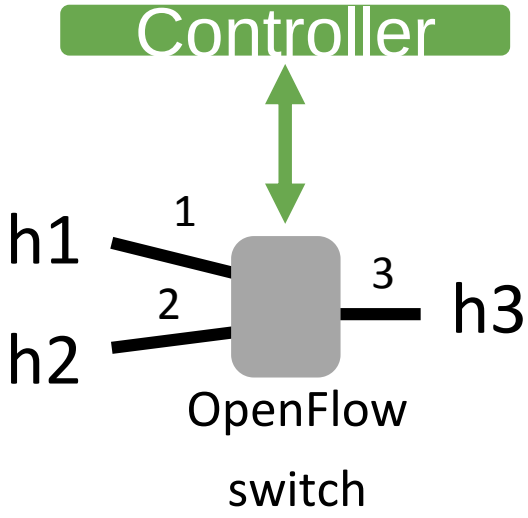
- Initial rule *: Send

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

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 for destination h2.

Example: SDN MAC Learning Done Wrong

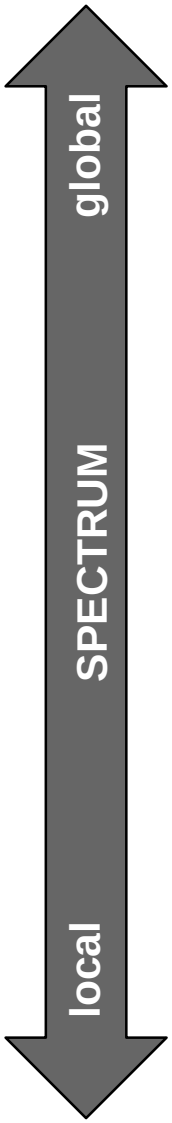


- Initial rule *: Send everything to controller

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

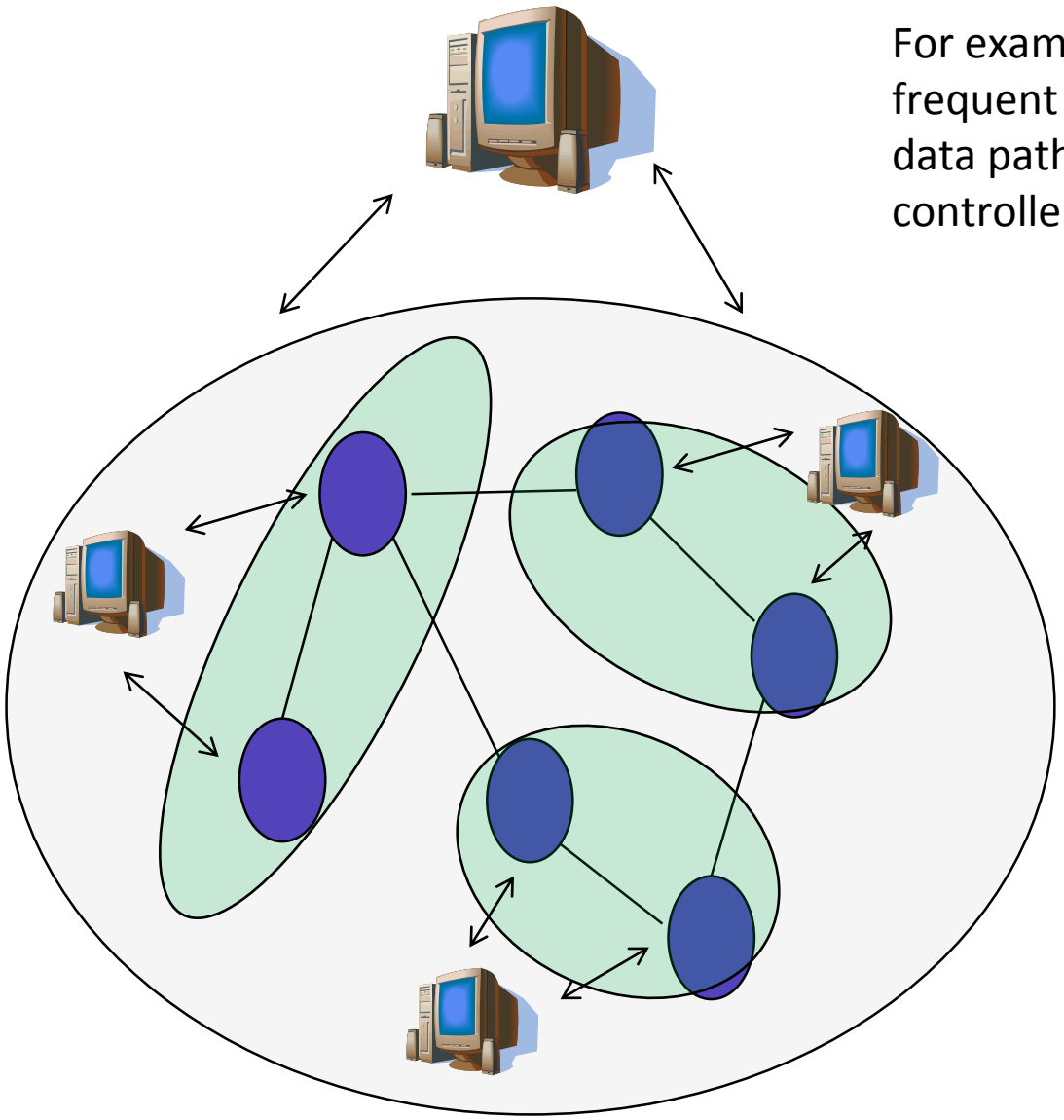
- What happens when h2 sends to h1?
  - Switch knows destination: message forwarded to h1
  - No controller interaction, no new rule for h2
- What happens when h3 sends to h2?
  - Flooded! Controller did not put the rule to h2!

Challenge: Local Control



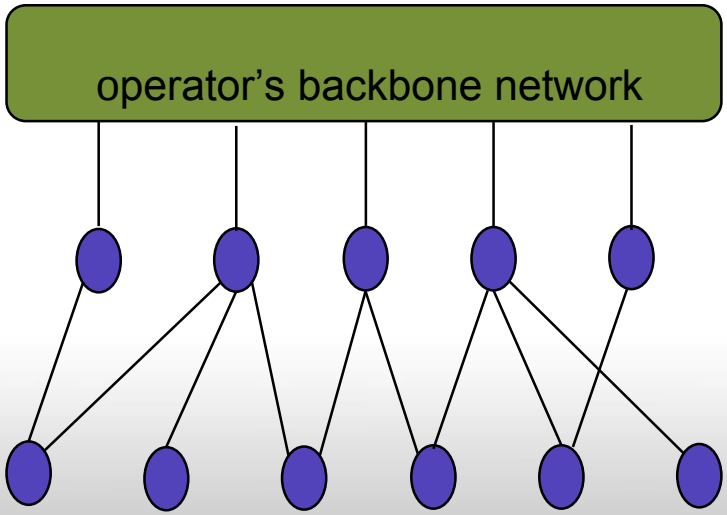
e.g., routing, spanning tree

e.g., local policy enforcer, elephant flow detection



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

SDN Task 1: Link Assignment („Semi-Matching Problem“)

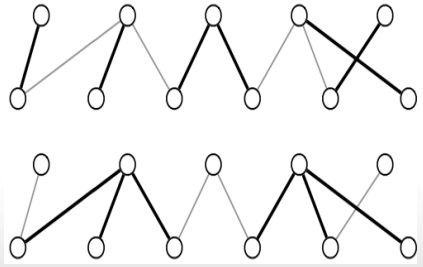


PoPs

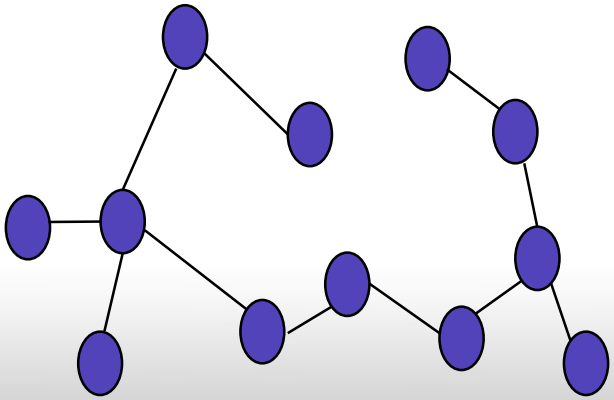
redundant links

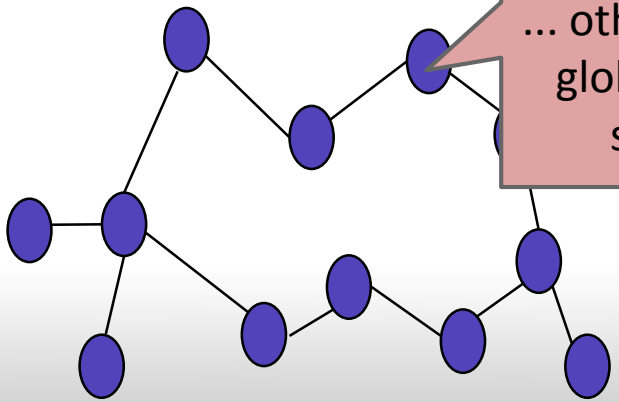
customer sites

Some tasks are local (e.g., heavy hitter detection, load balancing)...



SDN Task 2: Spanning Tree Verification

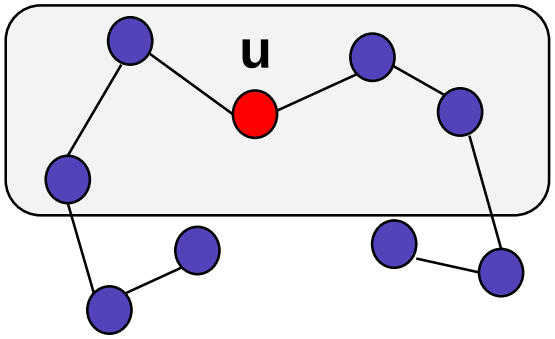




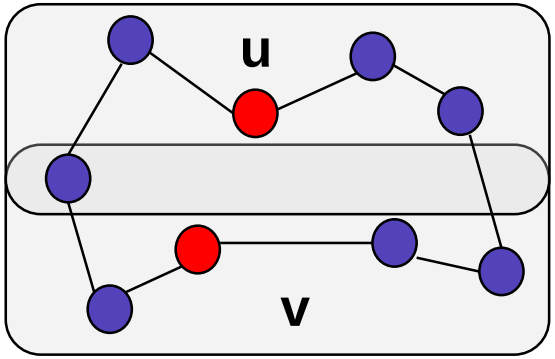
... others are inherently global (e.g., routing, spanning tree)

Limitations of a Local View

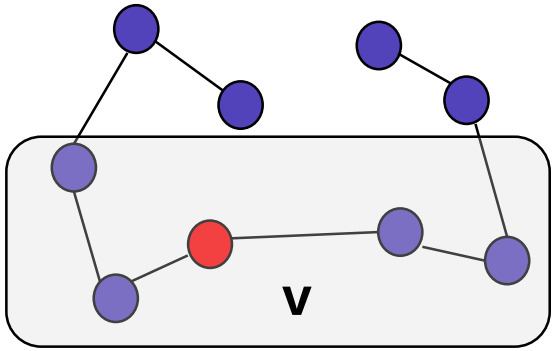
Example: checking loop-freedom



OK



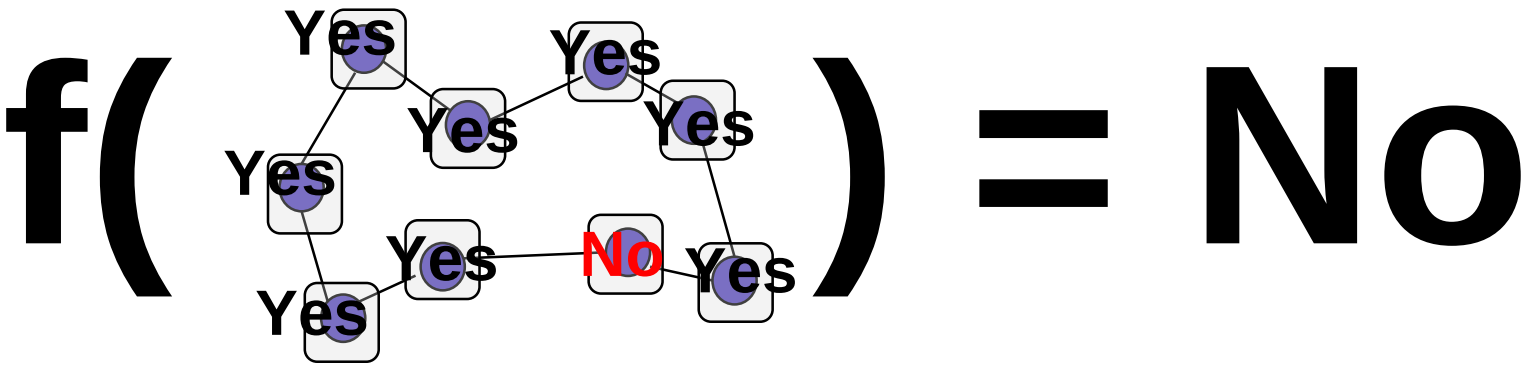
not OK



OK

Verification is easier!

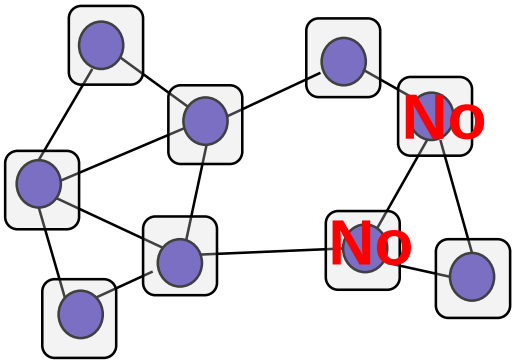
- Verification is often easier than computation
  - Sometimes sufficient if at least one controller notices inconsistency: it can then trigger global re-computation

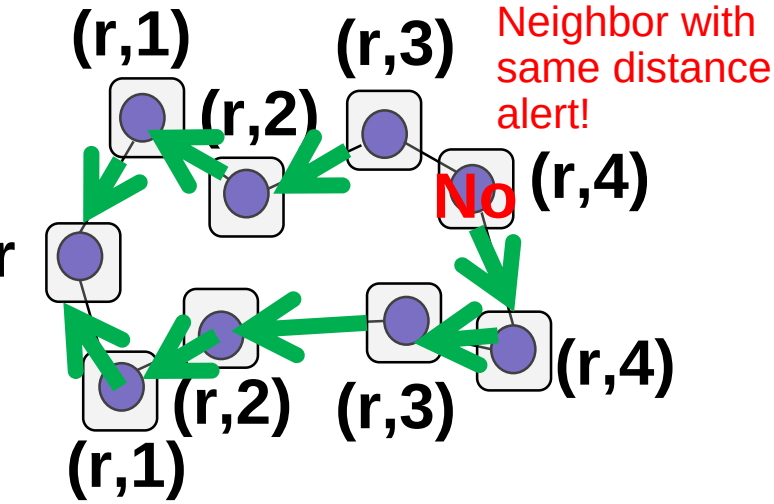


- Similar to classic computability theory
  - NP-complete problem solutions can be verified in polynomial time

Example: Local Checkability

Euler Property: Hard to compute Euler tour ("each edge exactly once"), but easy to verify! 0-bits (= no communication) : output whether degree is even.

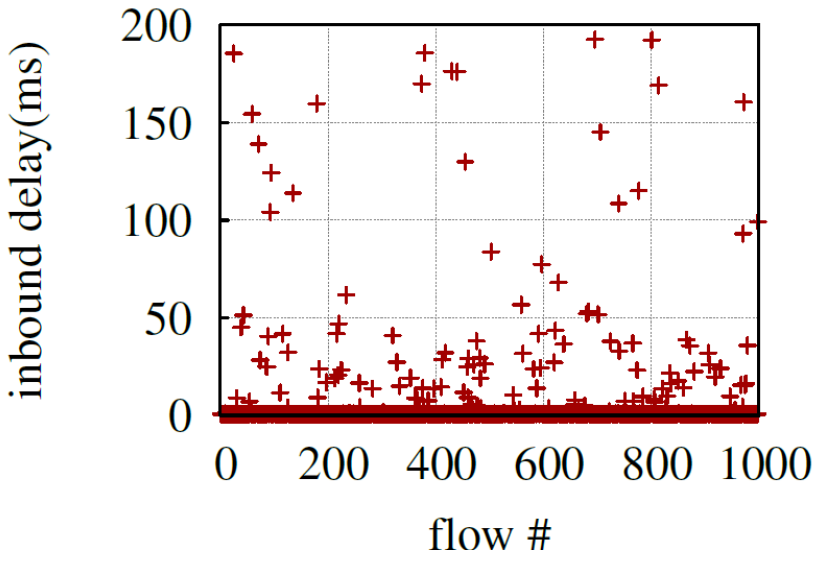




Spanning Tree Property: Label encodes root node plus distance & direction to root. At least one node notices that root/distance not consistent! Requires O(log n) bits.

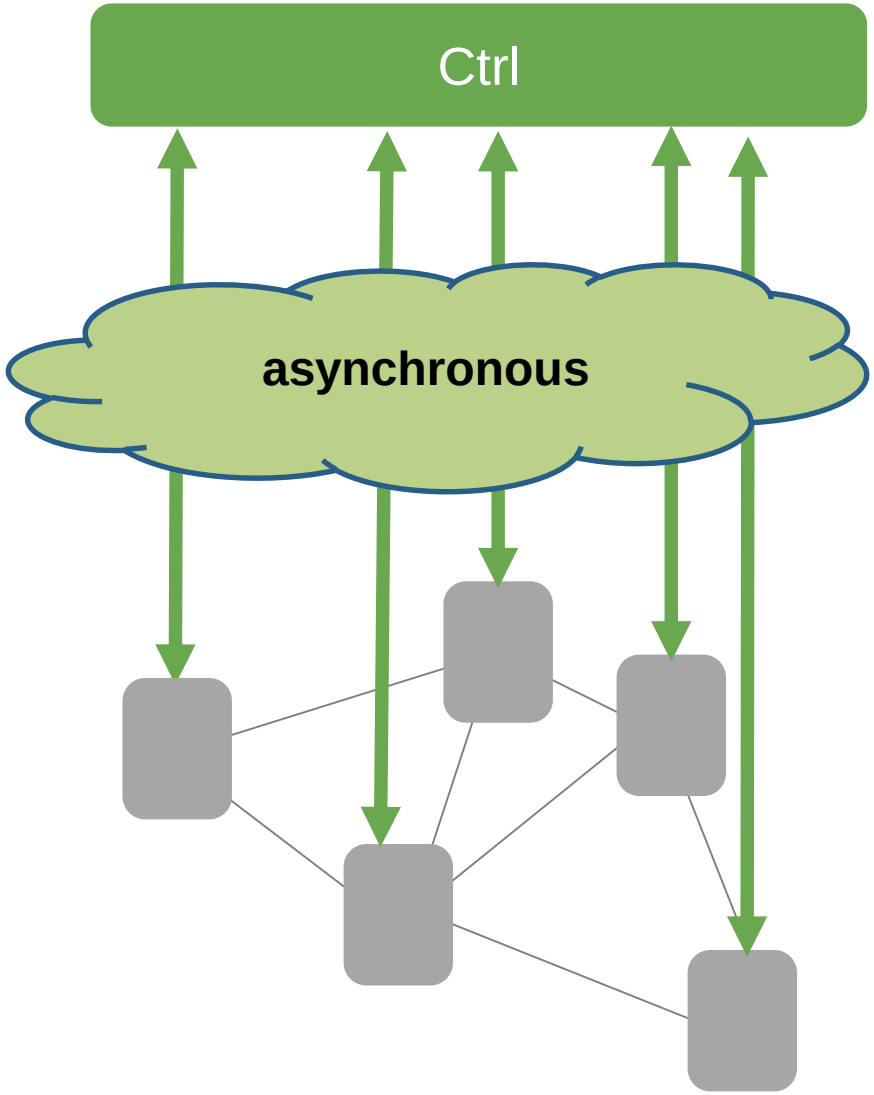
Interconnect: Algorithms *with a twist!*

- Another challenge: asynchronous communication channel



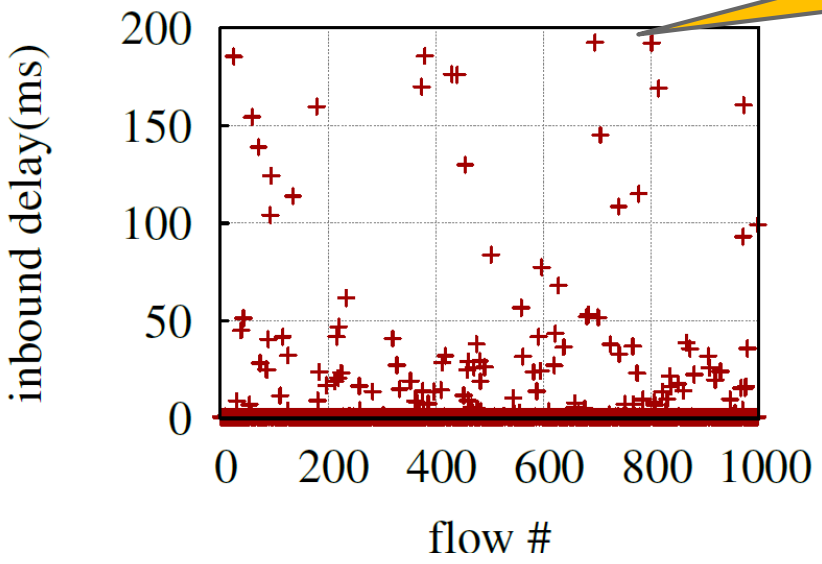
He et al., ACM SOSR 2015:

without network latency



Interconnect: Algorithms *with a twist!*

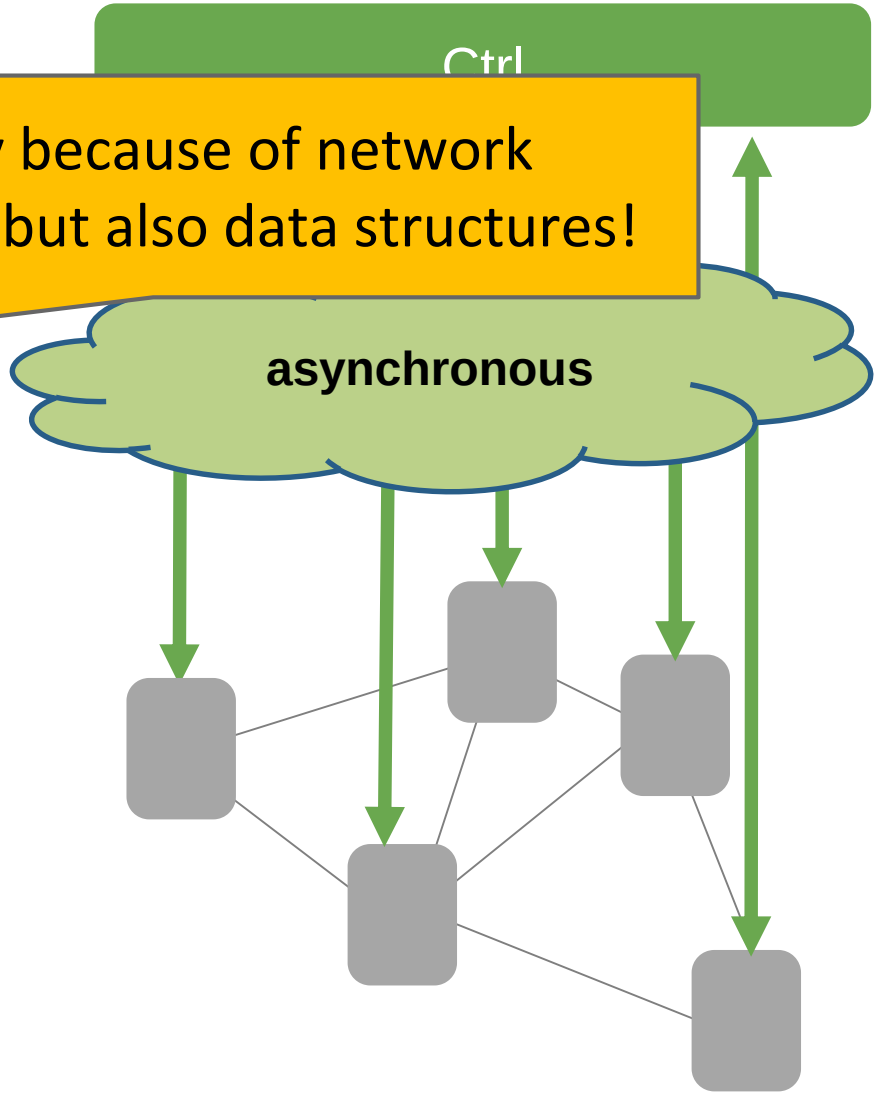
- Another challenge: asynchronous communication channel



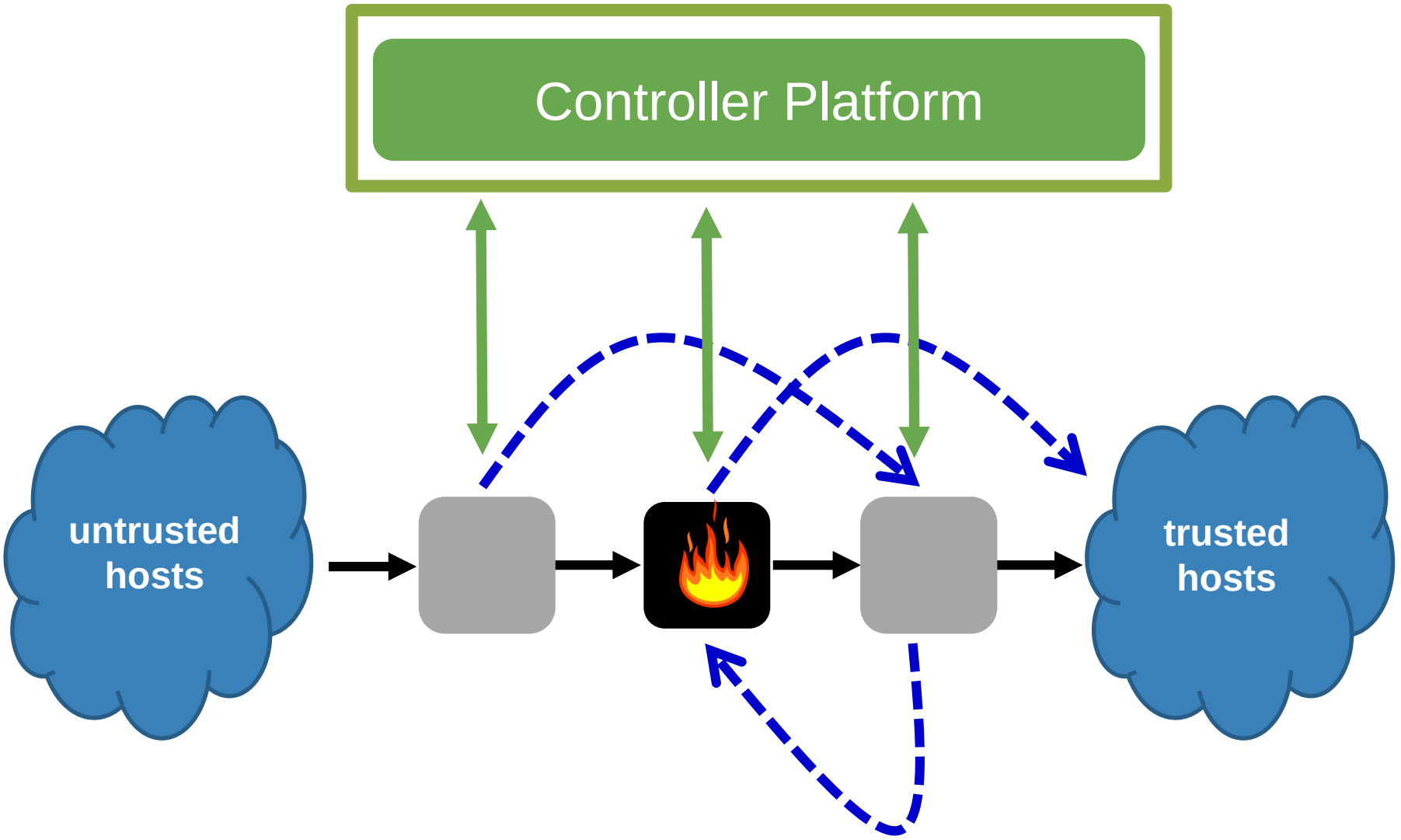
He et al., ACM SOSR 2015:

without network latency

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

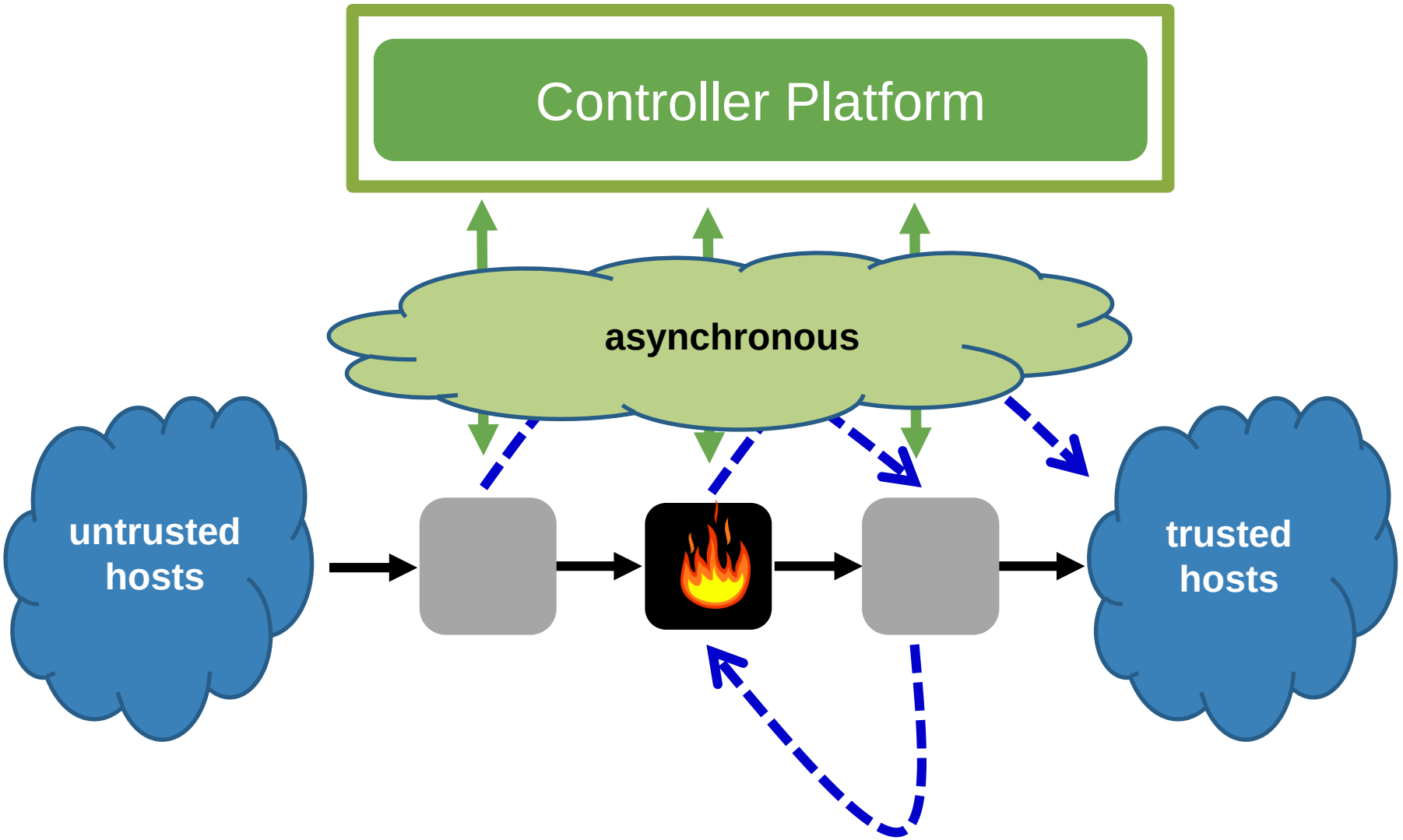


What can possibly go wrong?



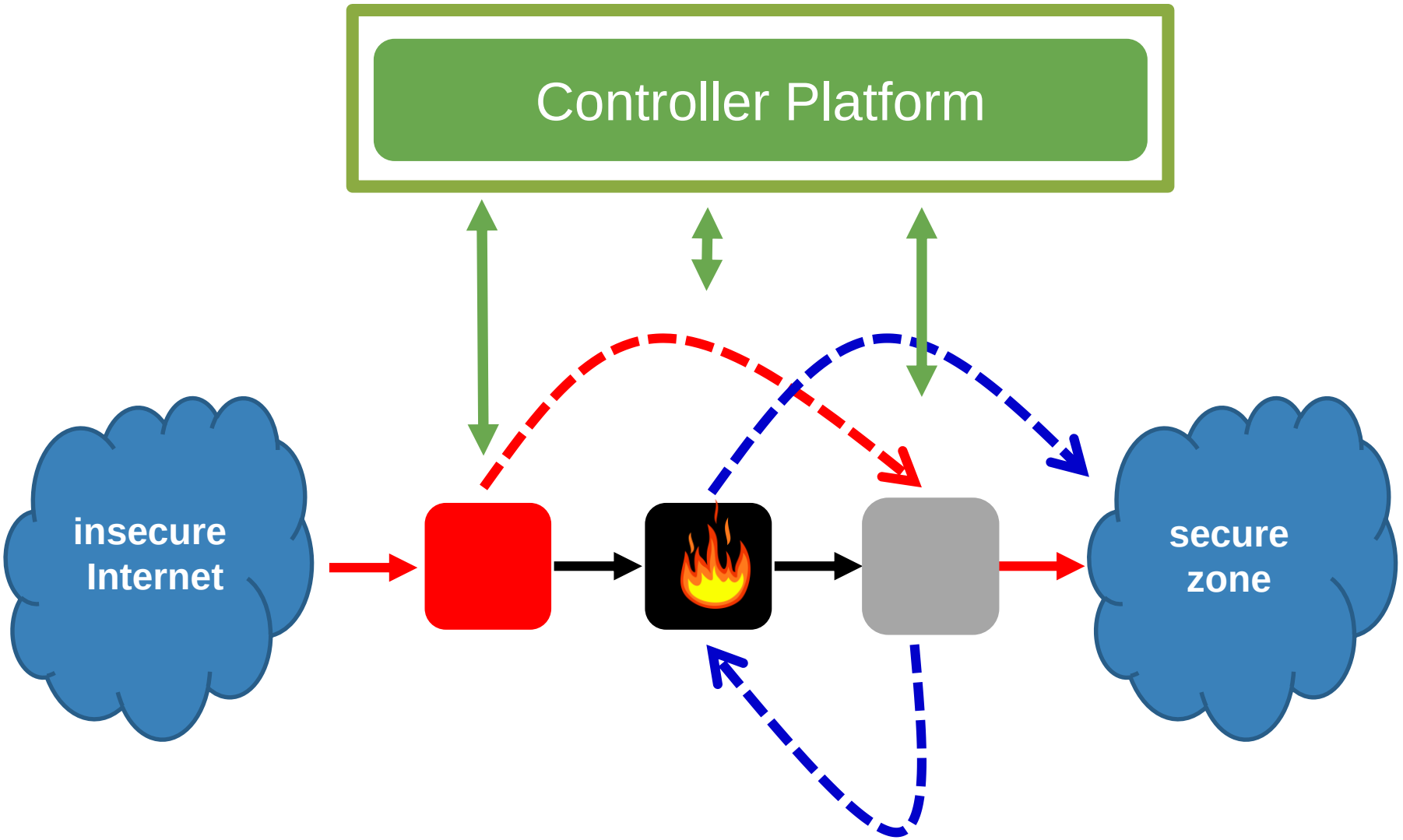
Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

What can possibly go wrong?

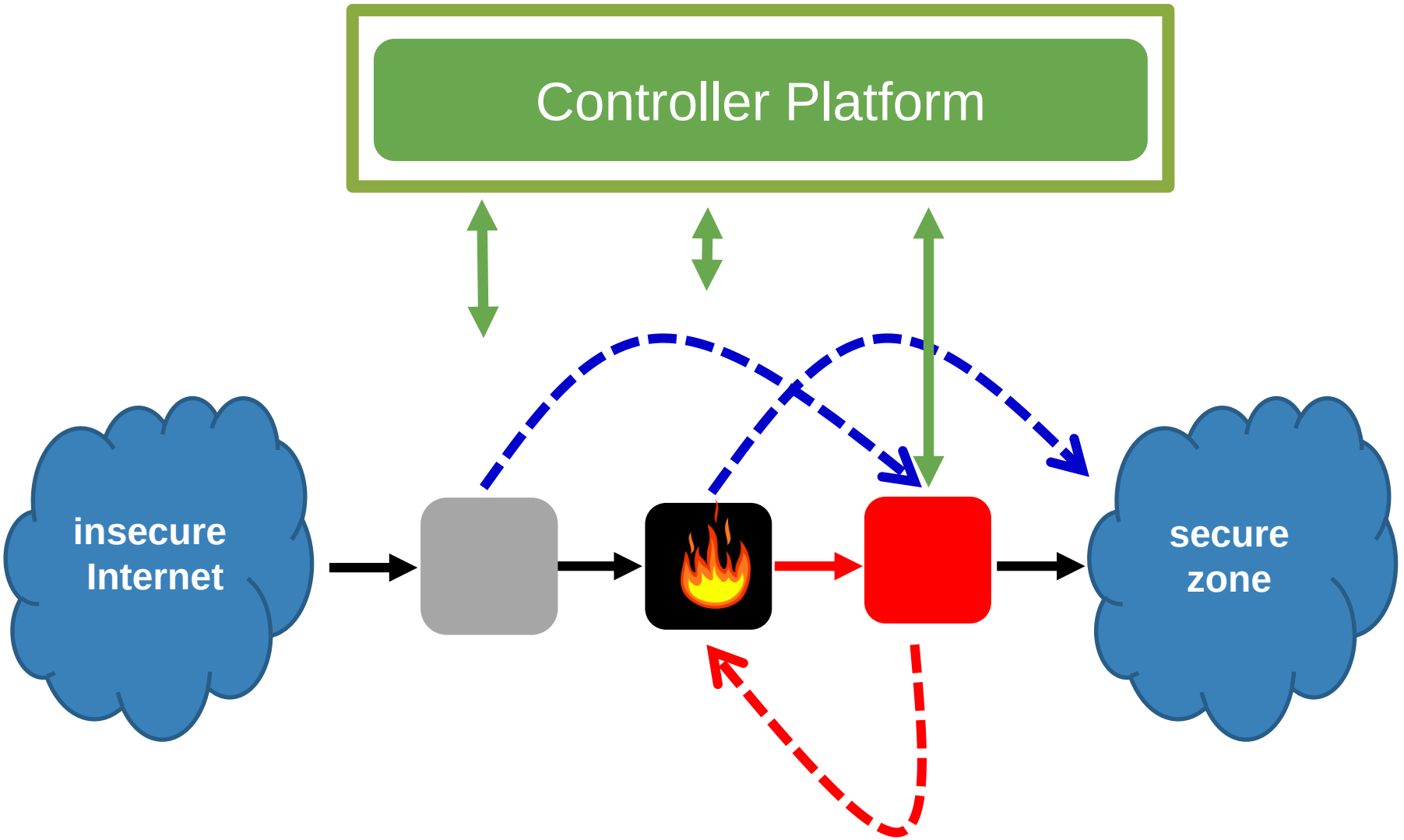


Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

Example 1: Bypassed Waypoint

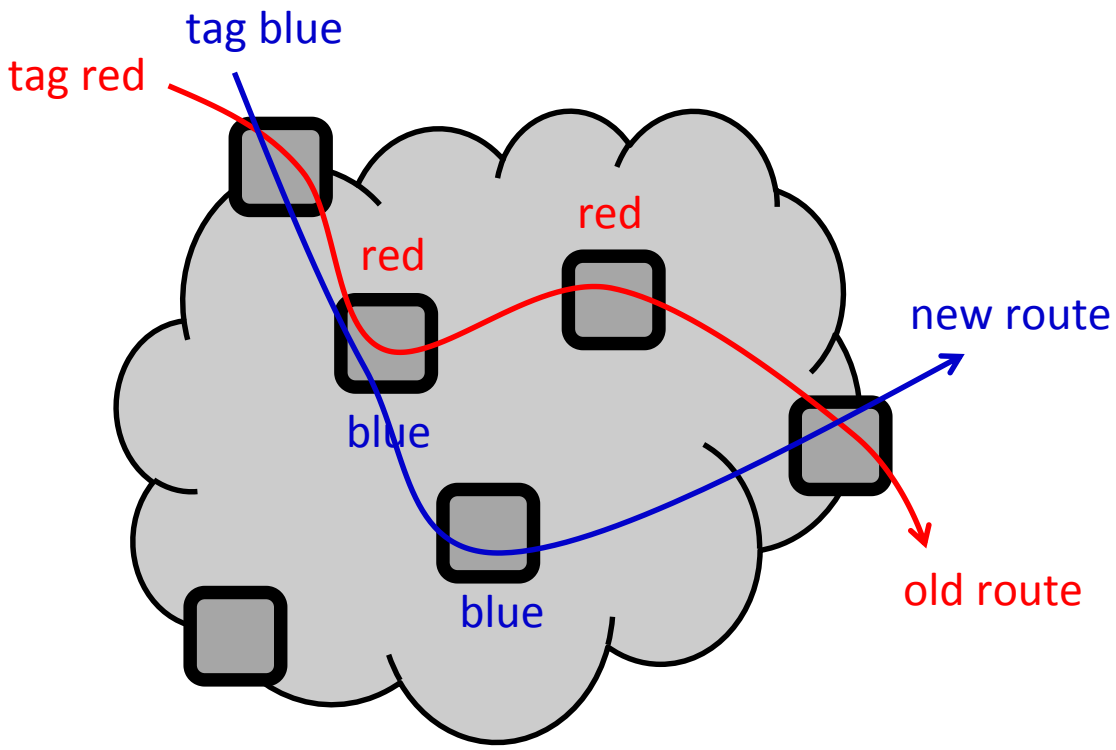


Example 2: *Transient* Loop



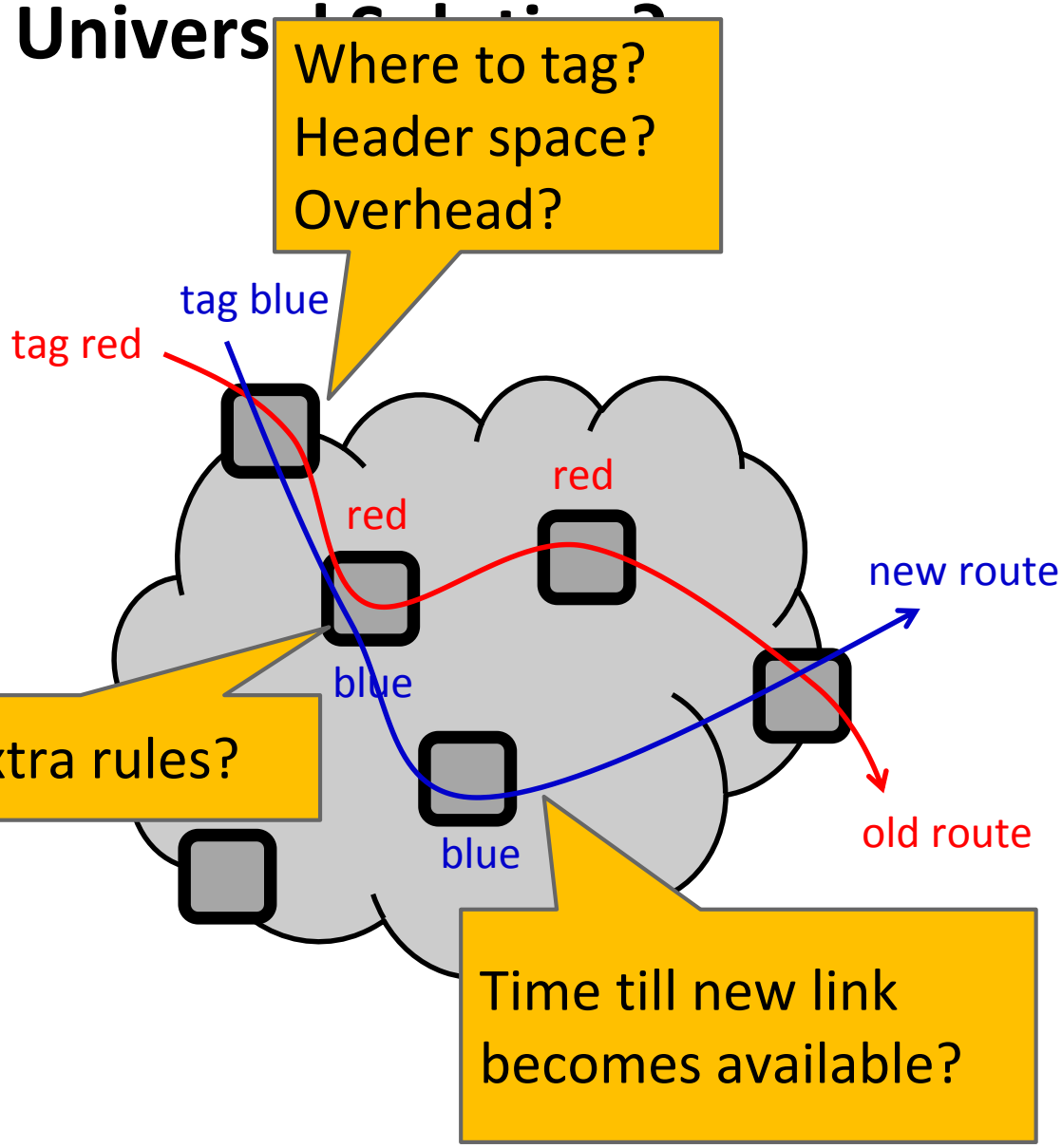
Tagging: A Universal Solution?

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



Tagging: A Universal Solution?

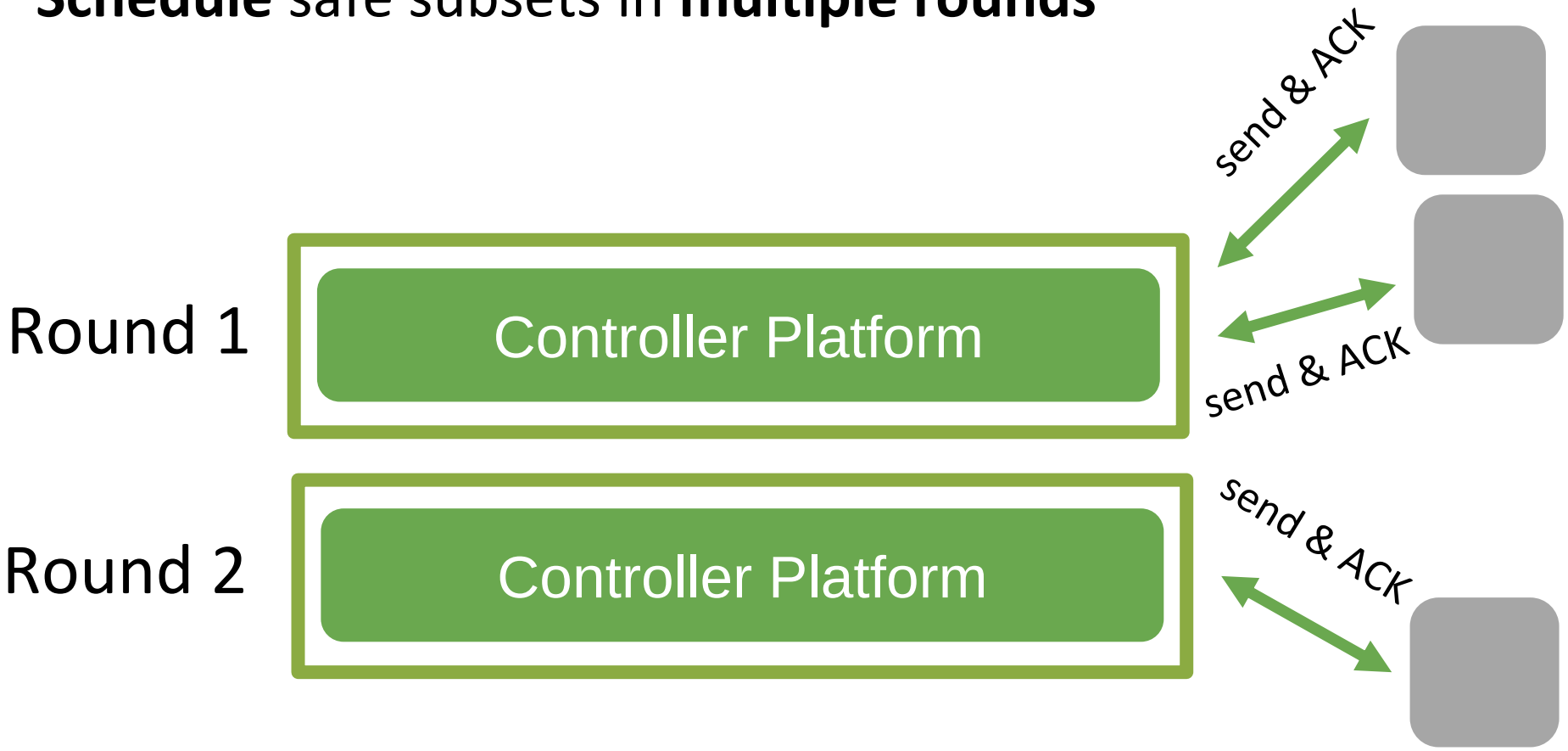
- Old route: red
- New route: blue
- 2-Phase Update:
  - Install blue rules internally
  - Flip tag at ingress ports



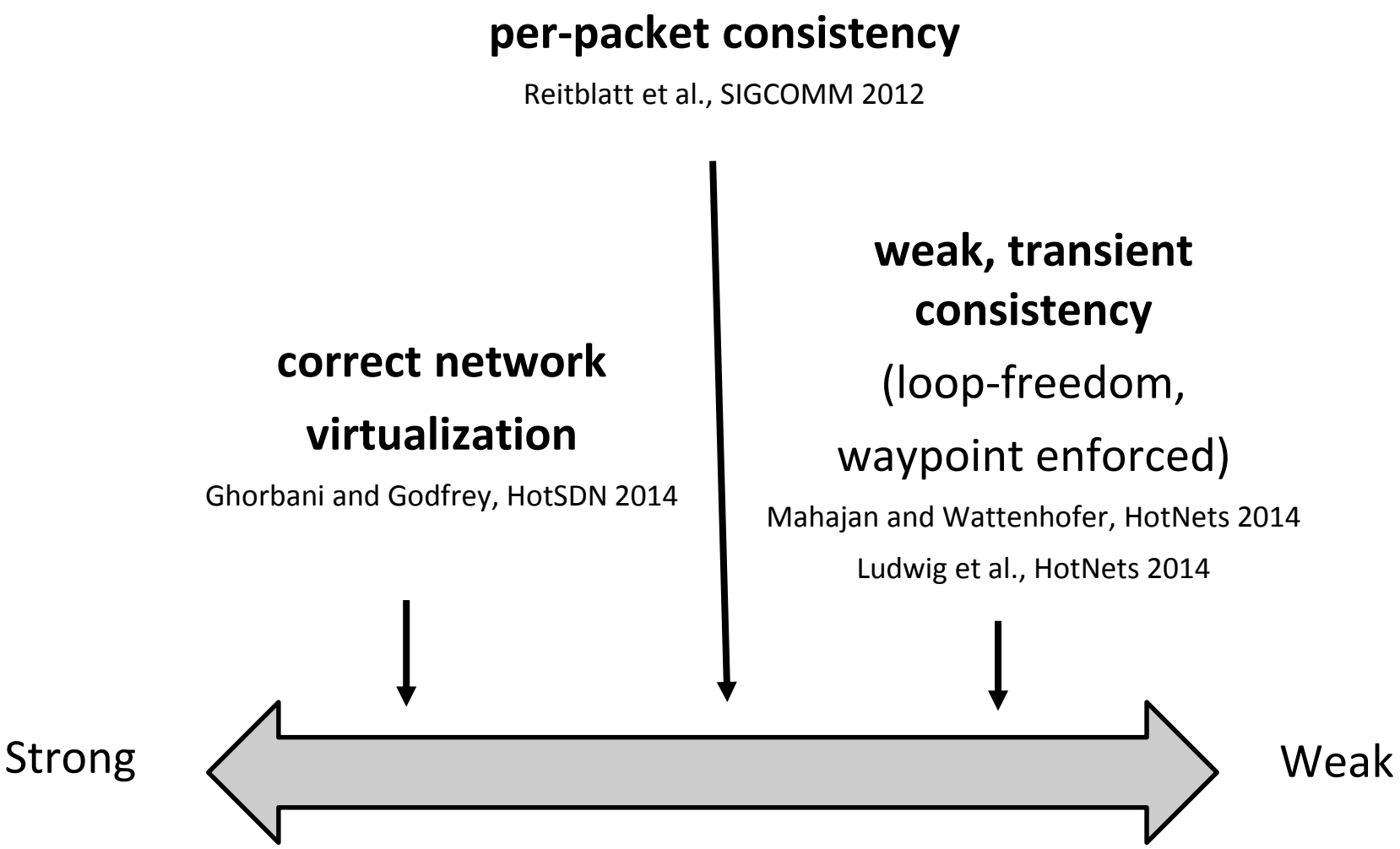
Alternative: Weaker Transient Consistency

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).

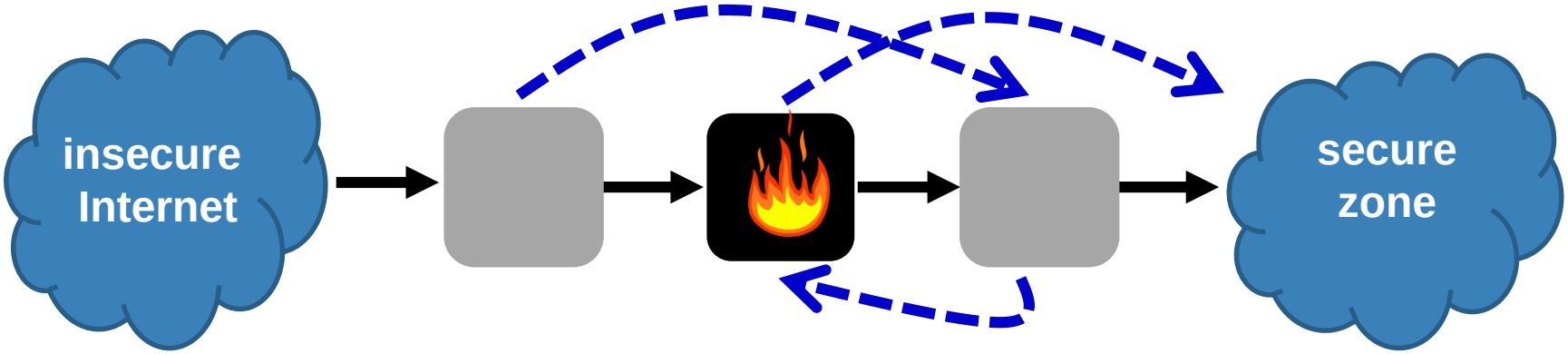
Schedule safe subsets in **multiple rounds**



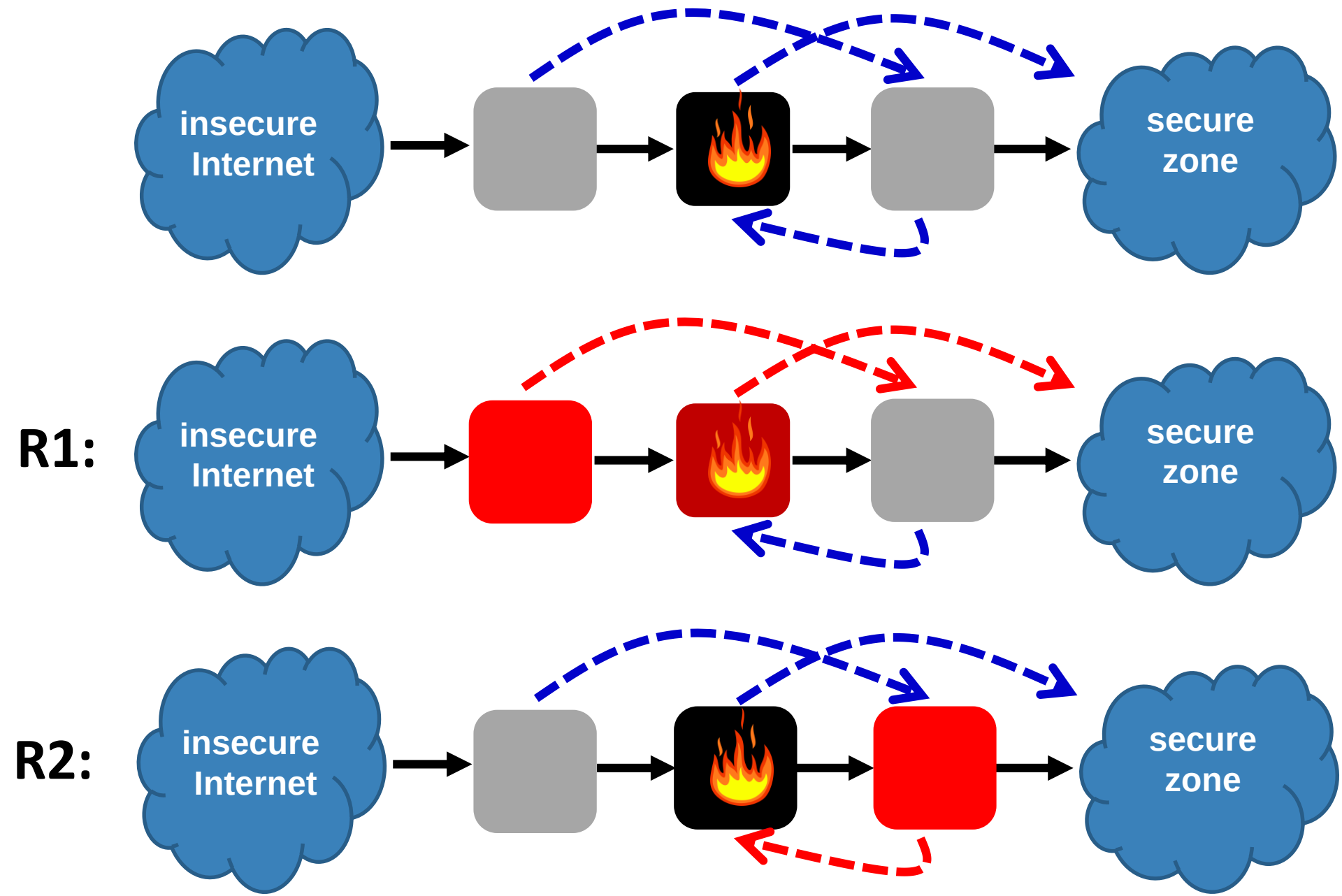
The Spectrum of Consistency



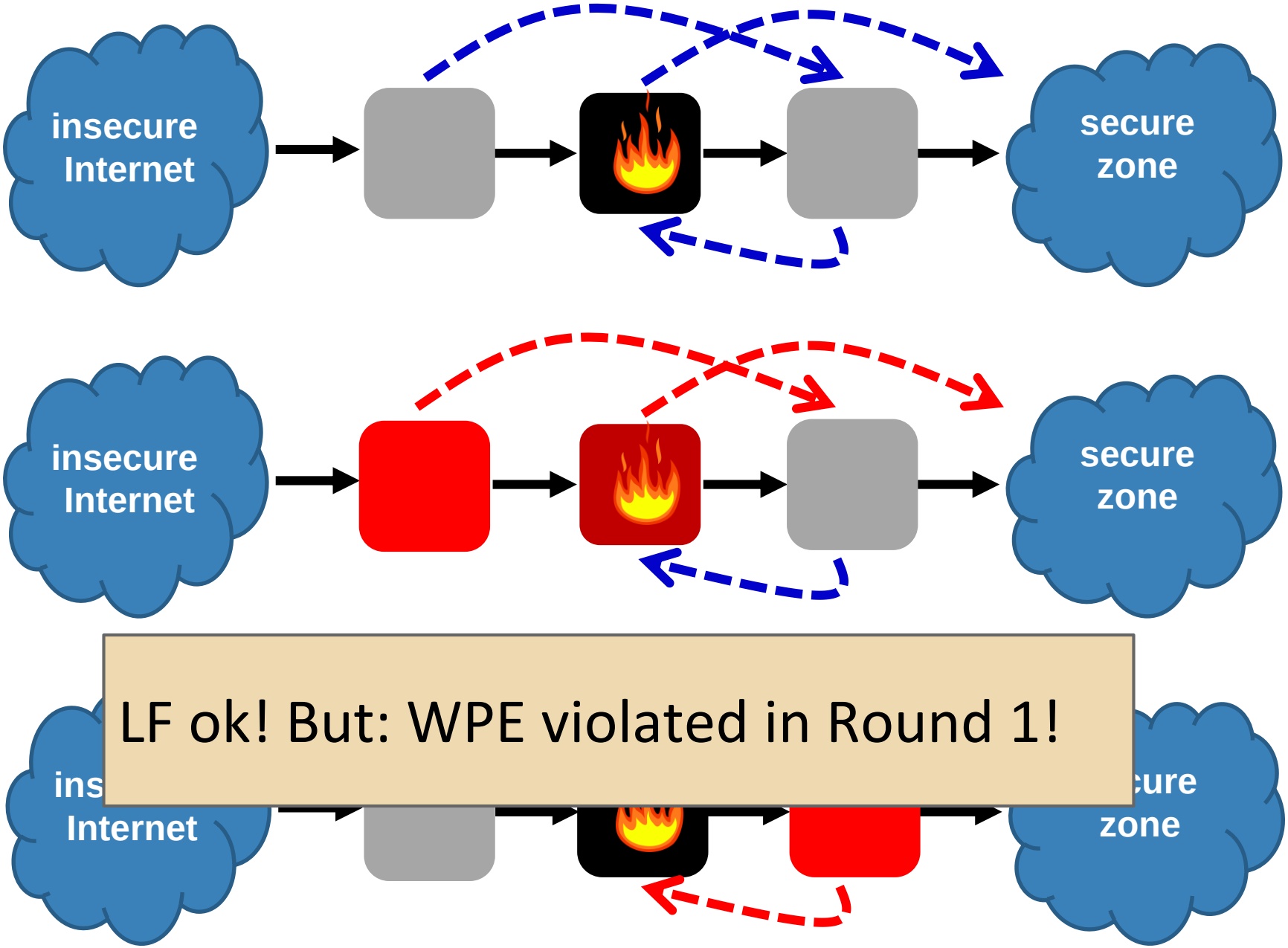
Going Back to Our Examples: LF Update?



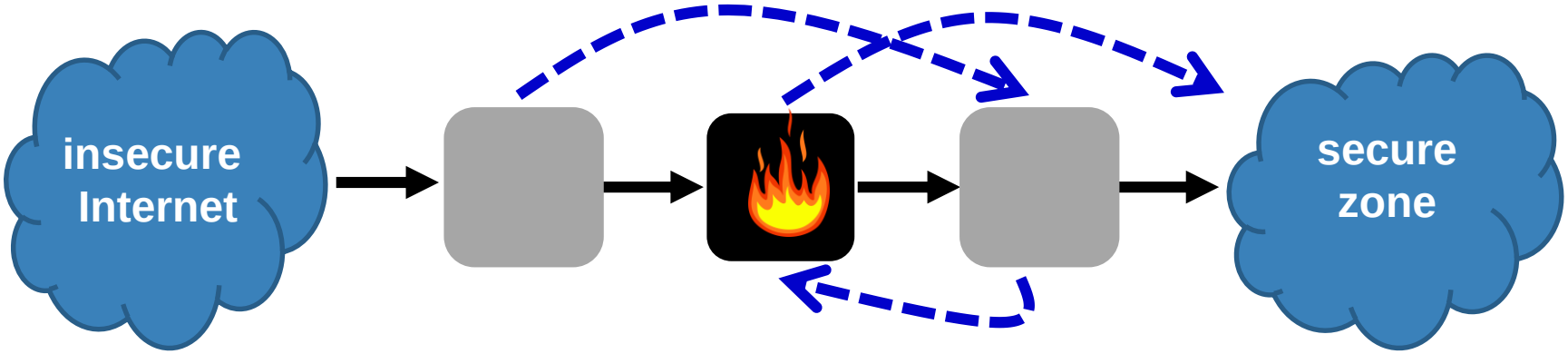
Going Back to Our Examples: LF Update!



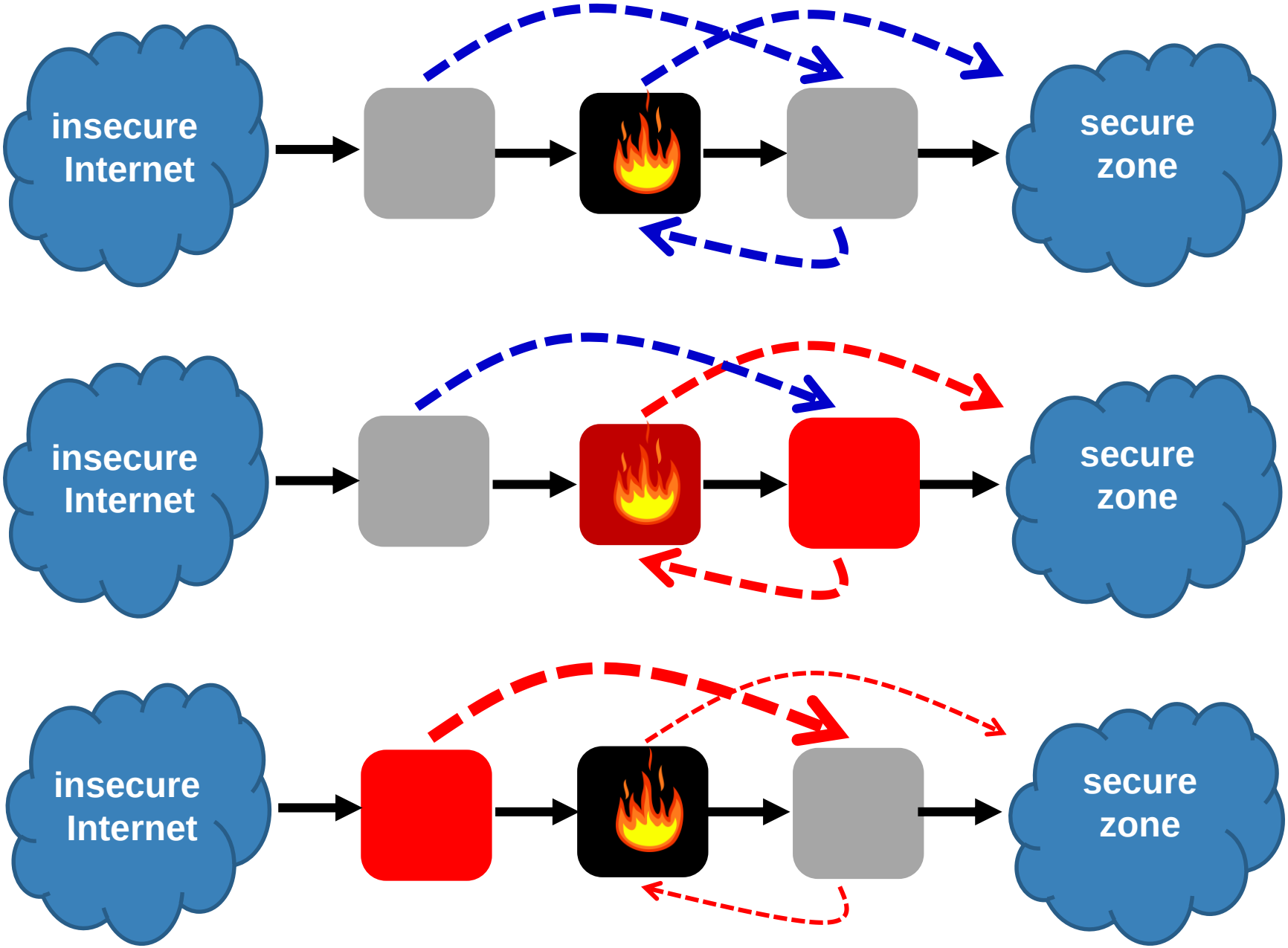
Going Back to Our Examples: LF Update!



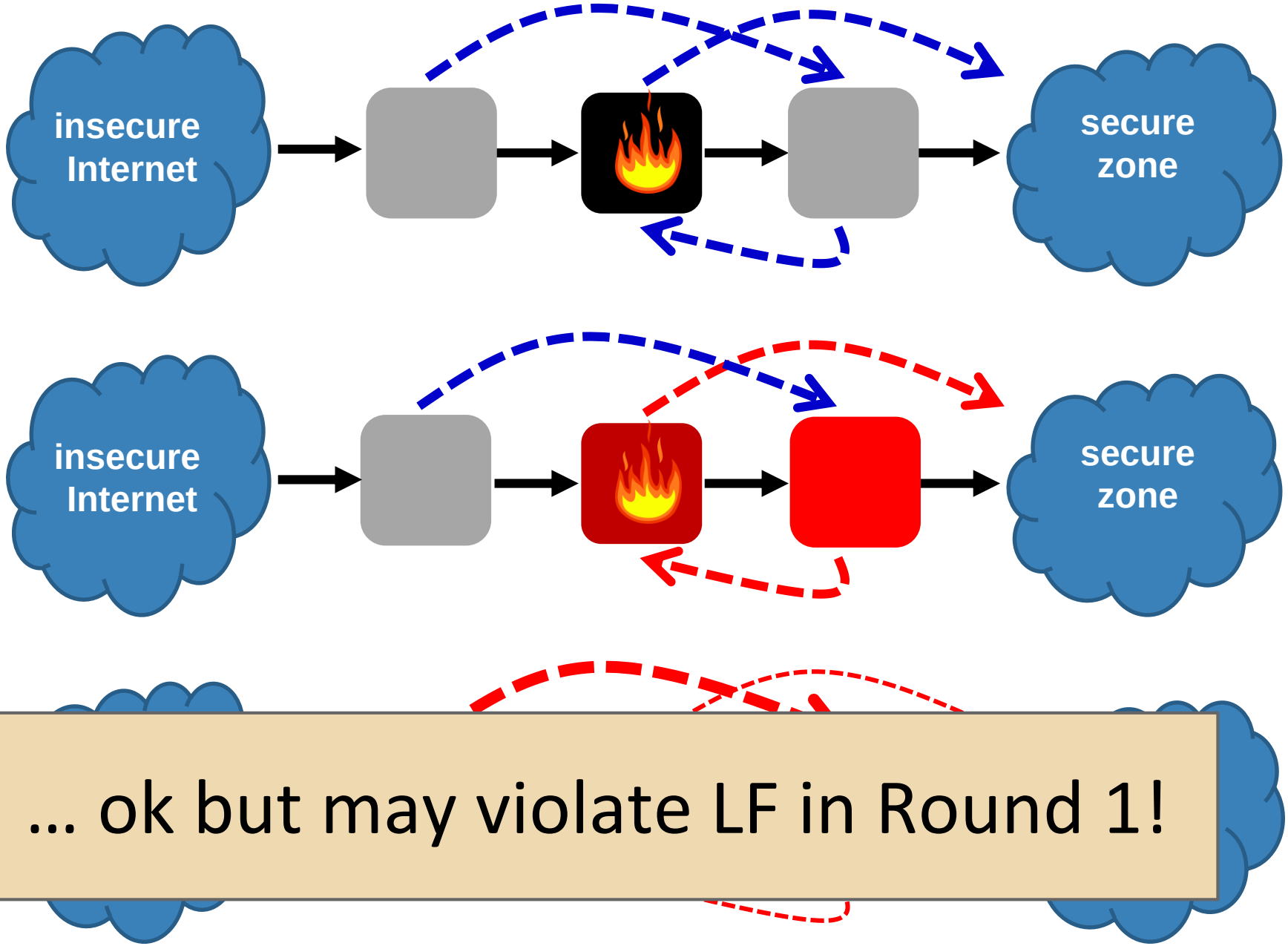
Going Back to Our Examples: WPE Update?



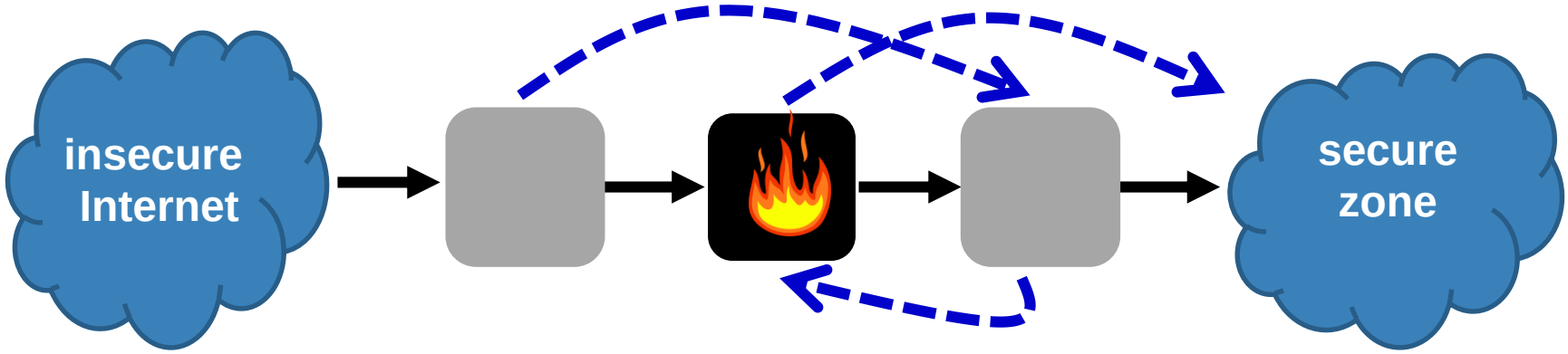
Going Back to Our Examples: WPE Update!



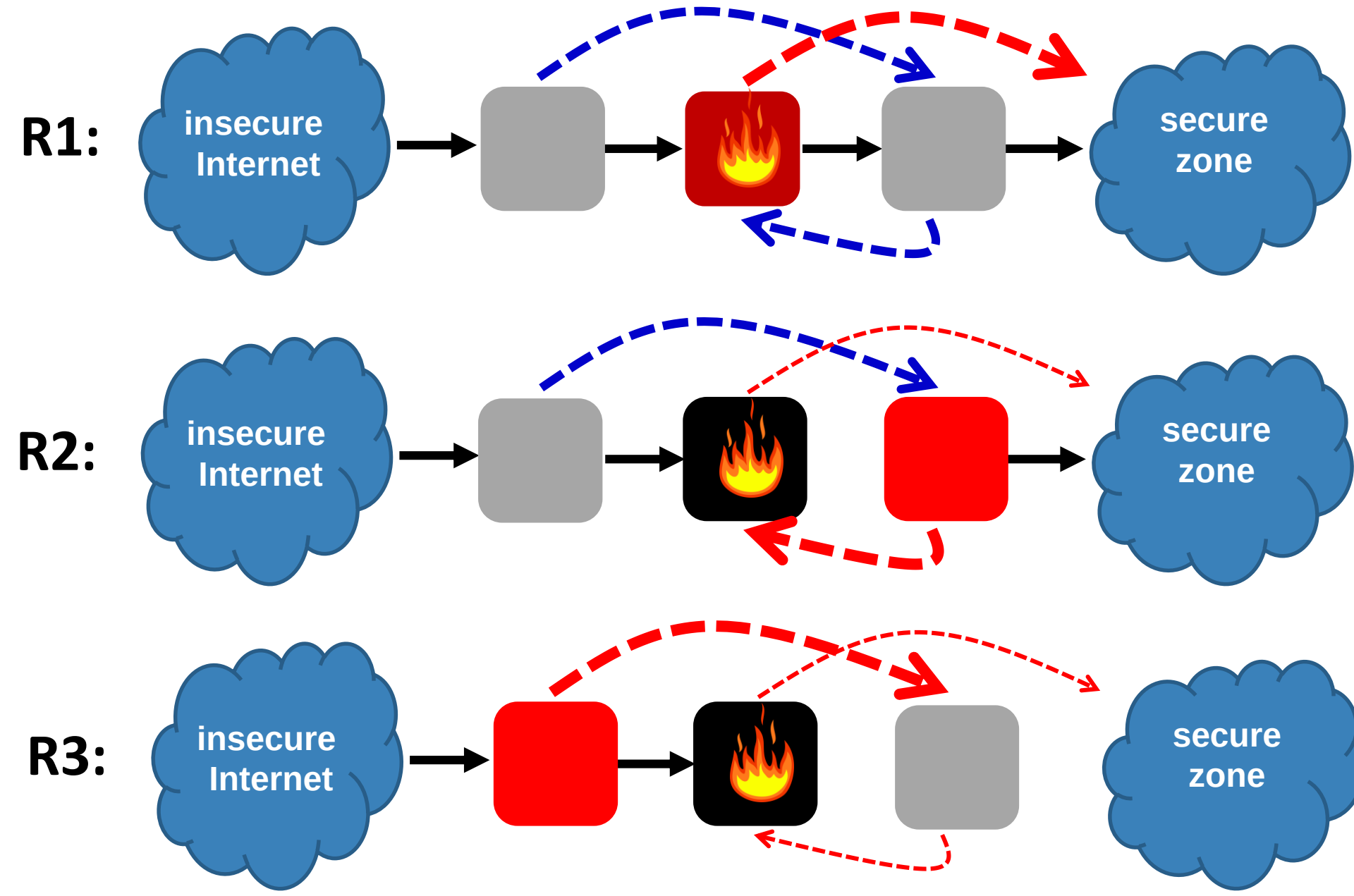
Going Back to Our Examples: WPE Update!



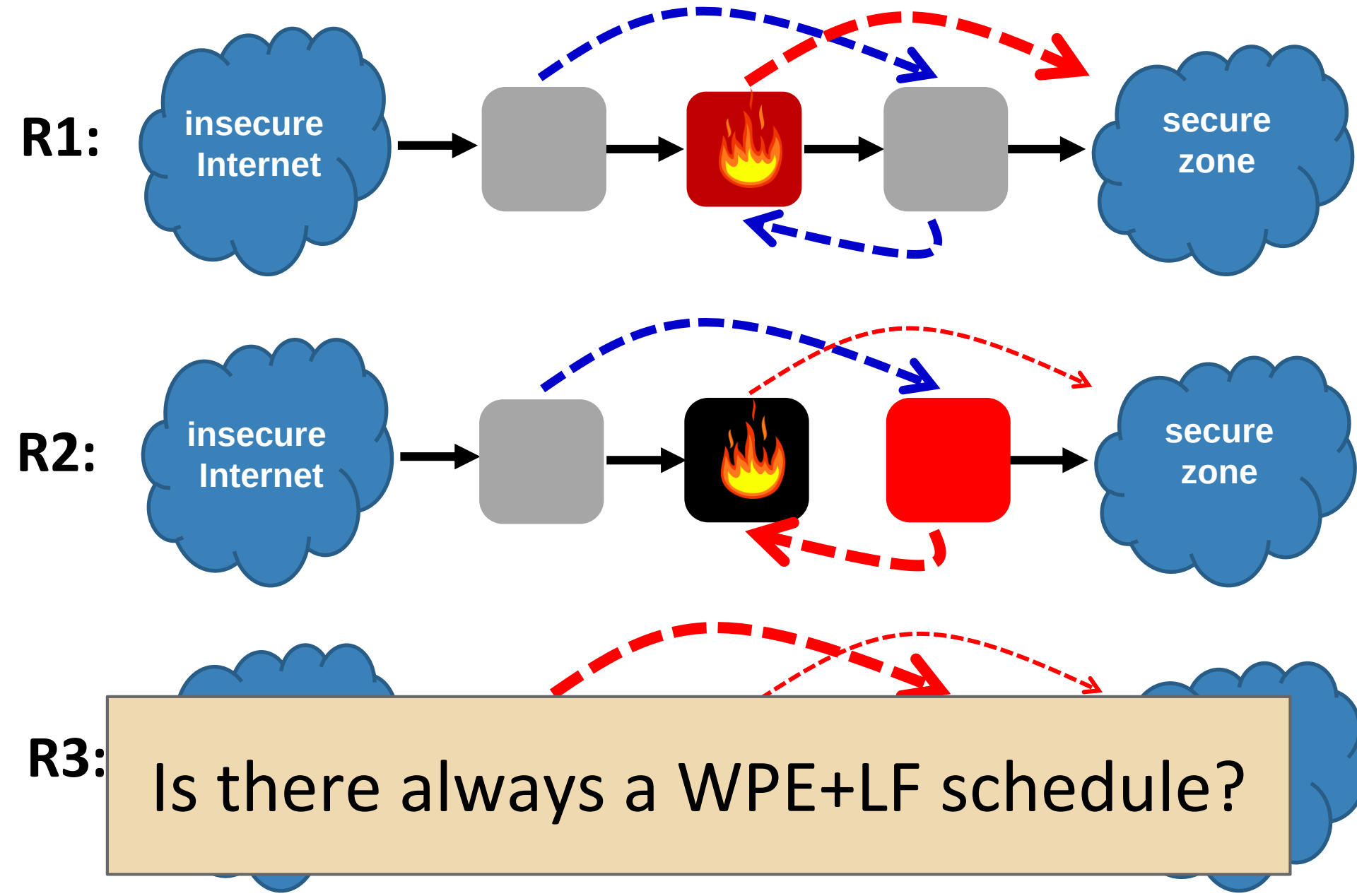
Going Back to Our Examples: Both WPE+LF?



Going Back to Our Examples: WPE+LF!

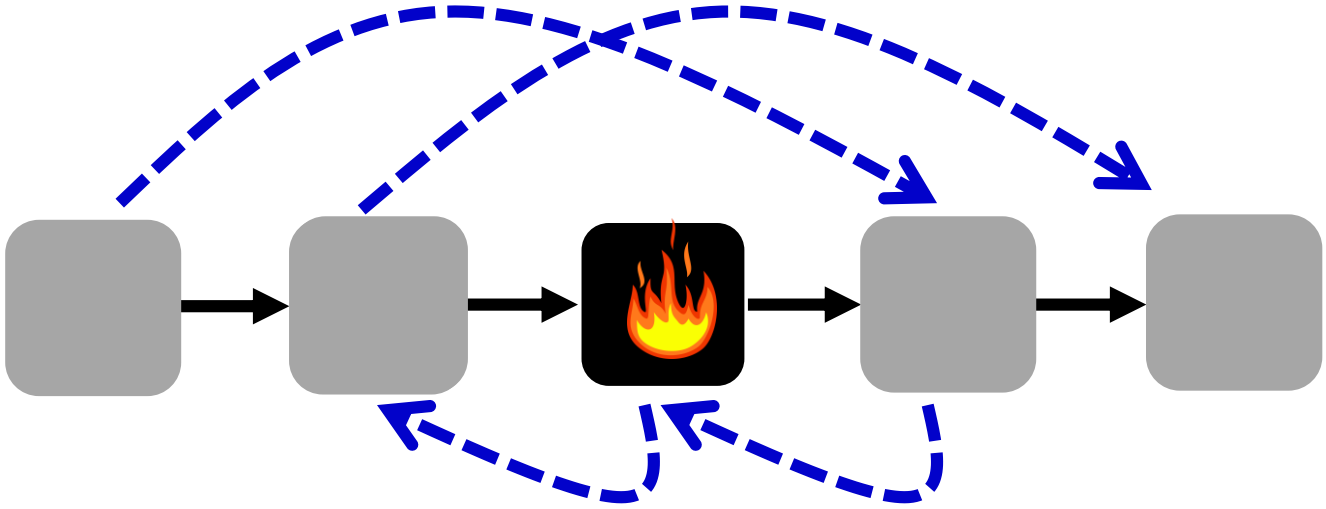


Going Back to Our Examples: WPE+LF!

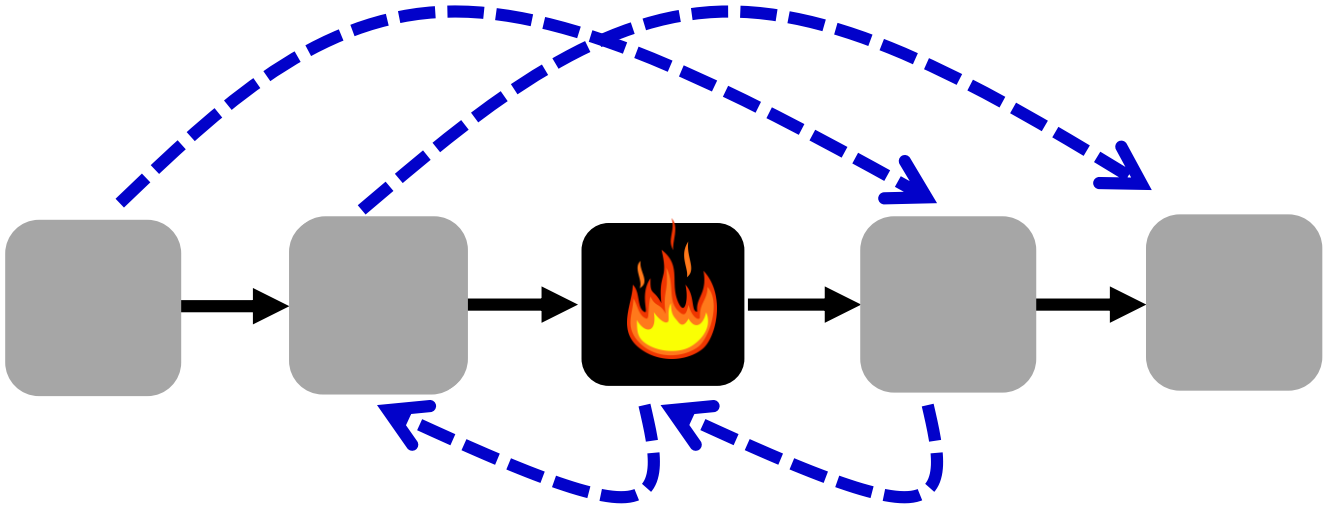


Is there always a WPE+LF schedule?

What about this one?



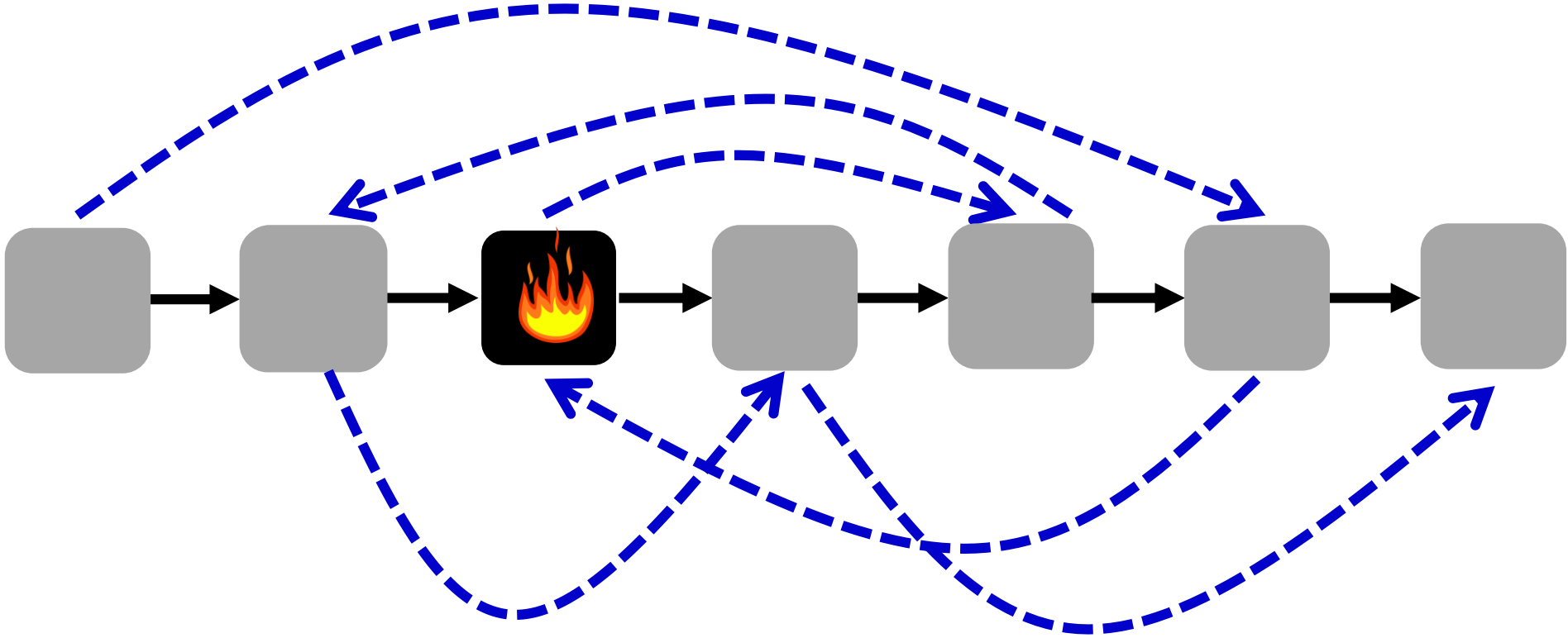
LF and WPE may conflict!



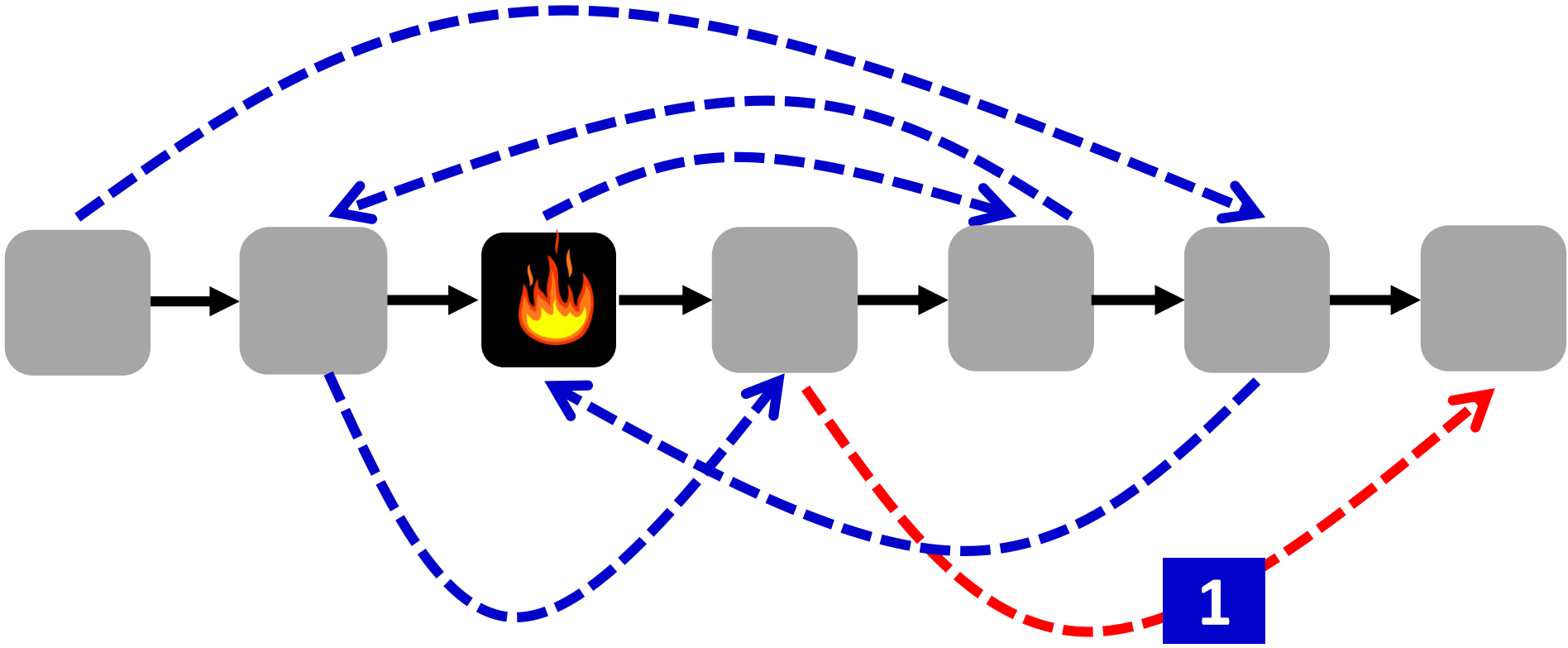
- ❑ Cannot update any **forward edge** in R1: WP
- ❑ Cannot update any **backward edge** in R1: LF

No schedule exists!

What about this one?

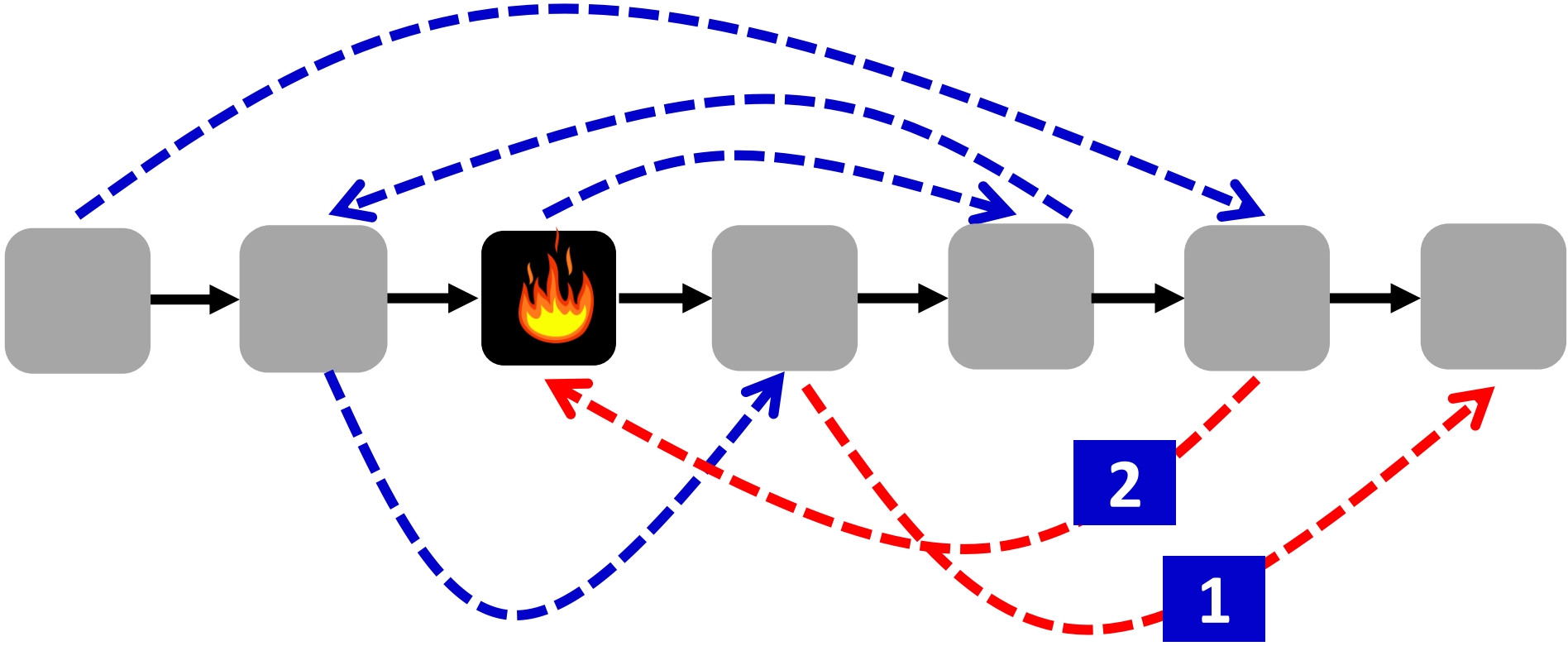


What about this one?



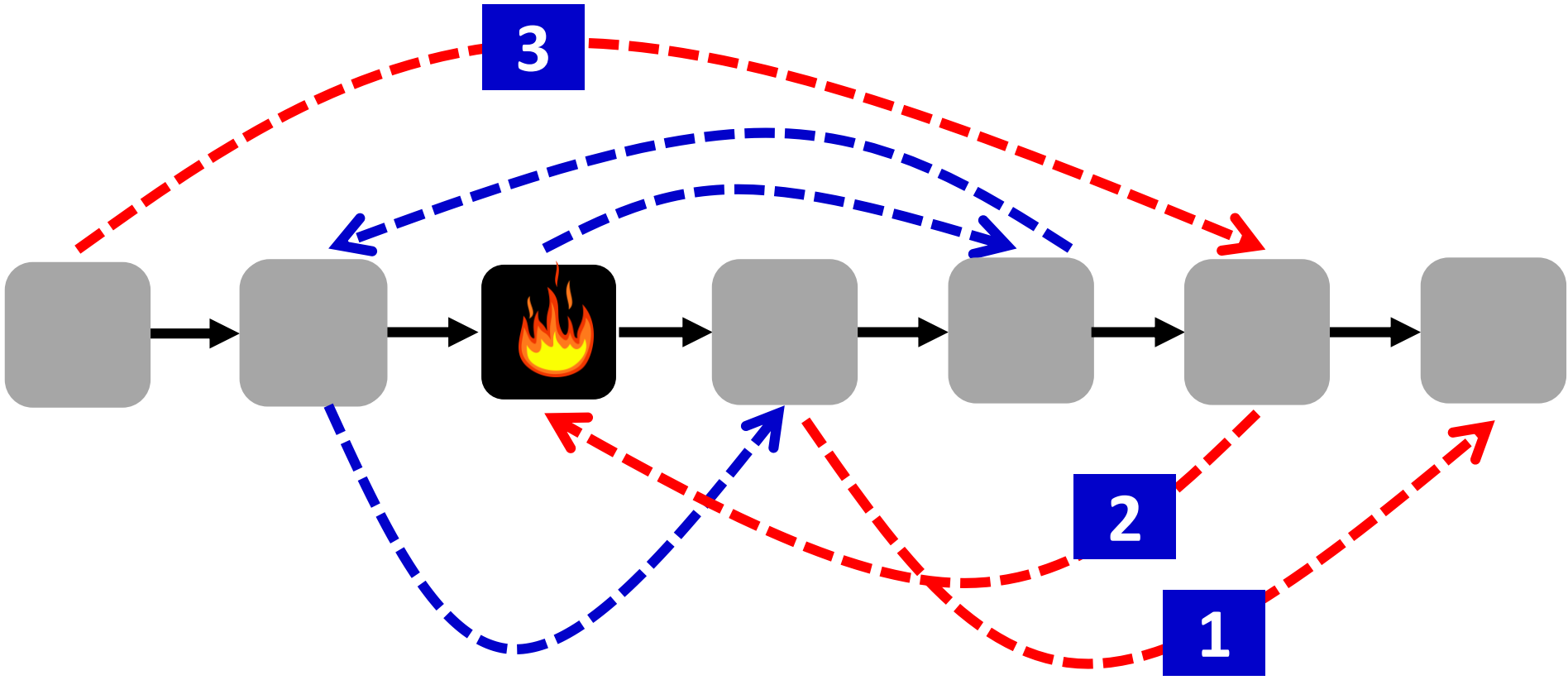
- Forward edge after the waypoint: safe!
  - No loop, no WPE violation

What about this one?



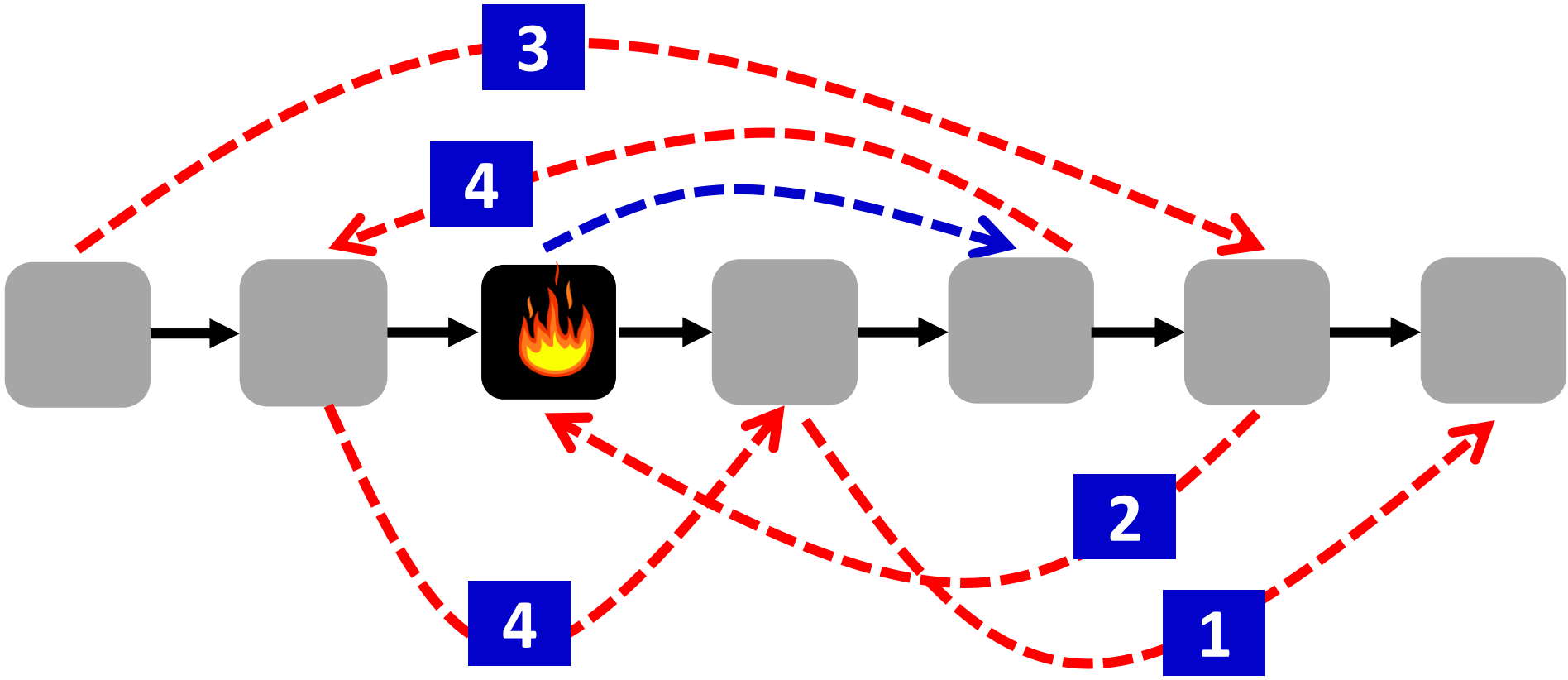
- Now this backward is safe too!
  - No loop because exit through 1

What about this one?



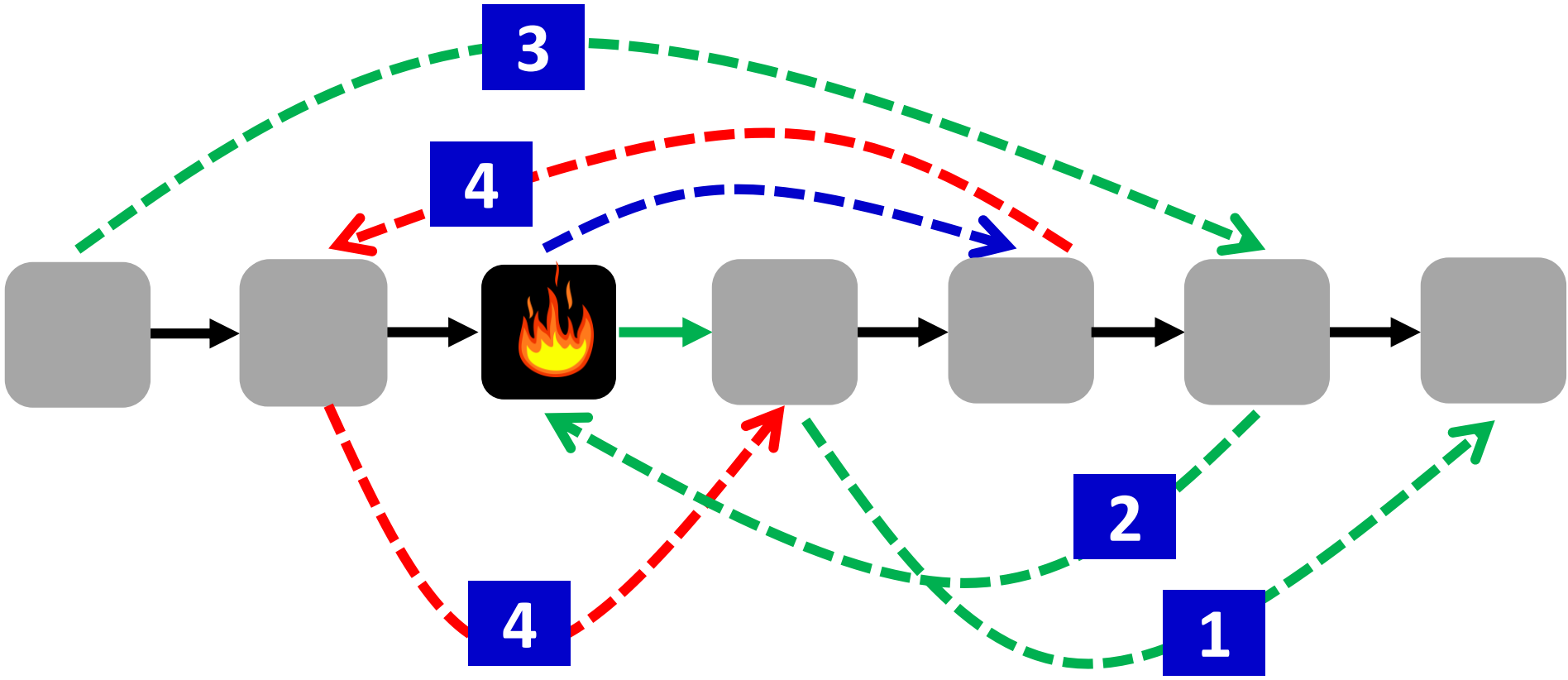
- ❑ Now this is safe: **2** ready back to WP!
  - ❑ No waypoint violation

What about this one?



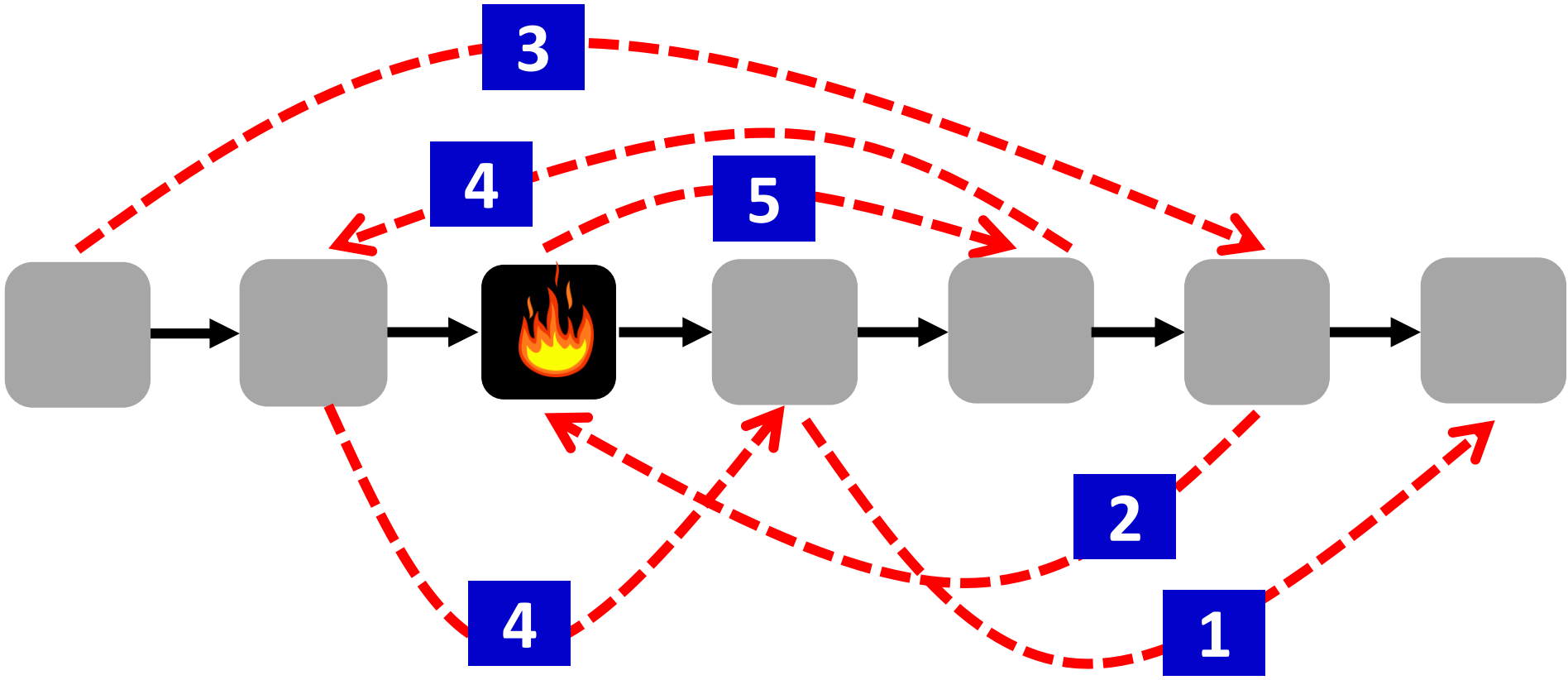
❑ Ok: loop-free and also not on the path (exit via 1)

What about this one?

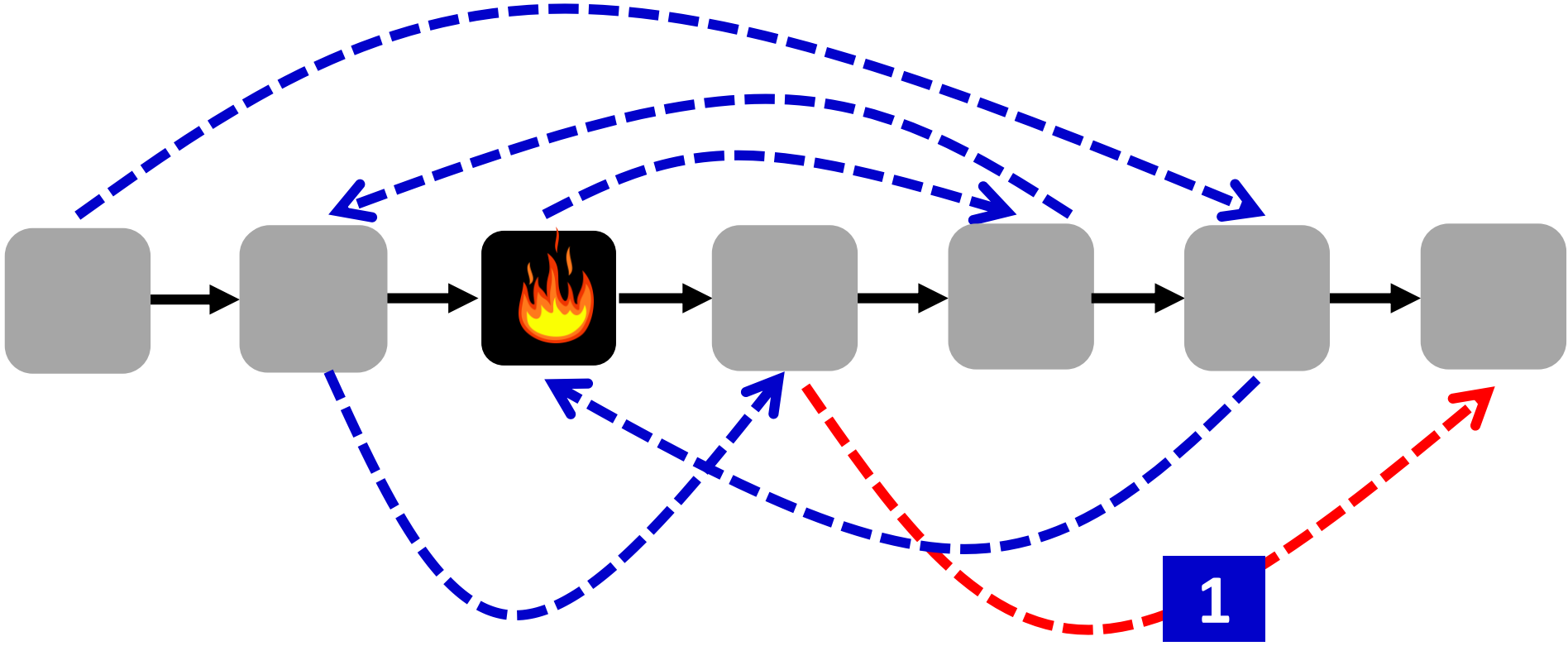


❑ Ok: loop-free and also not on the path (exit via 1)

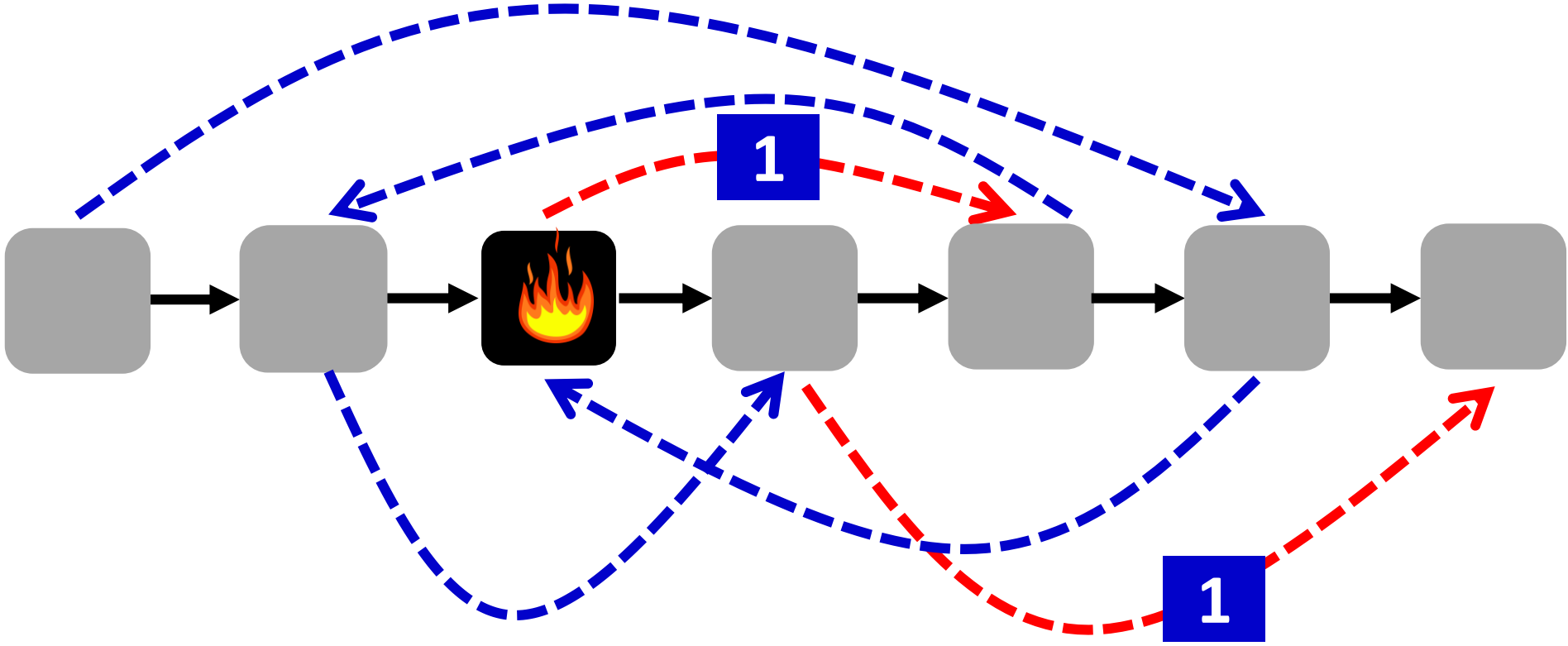
What about this one?



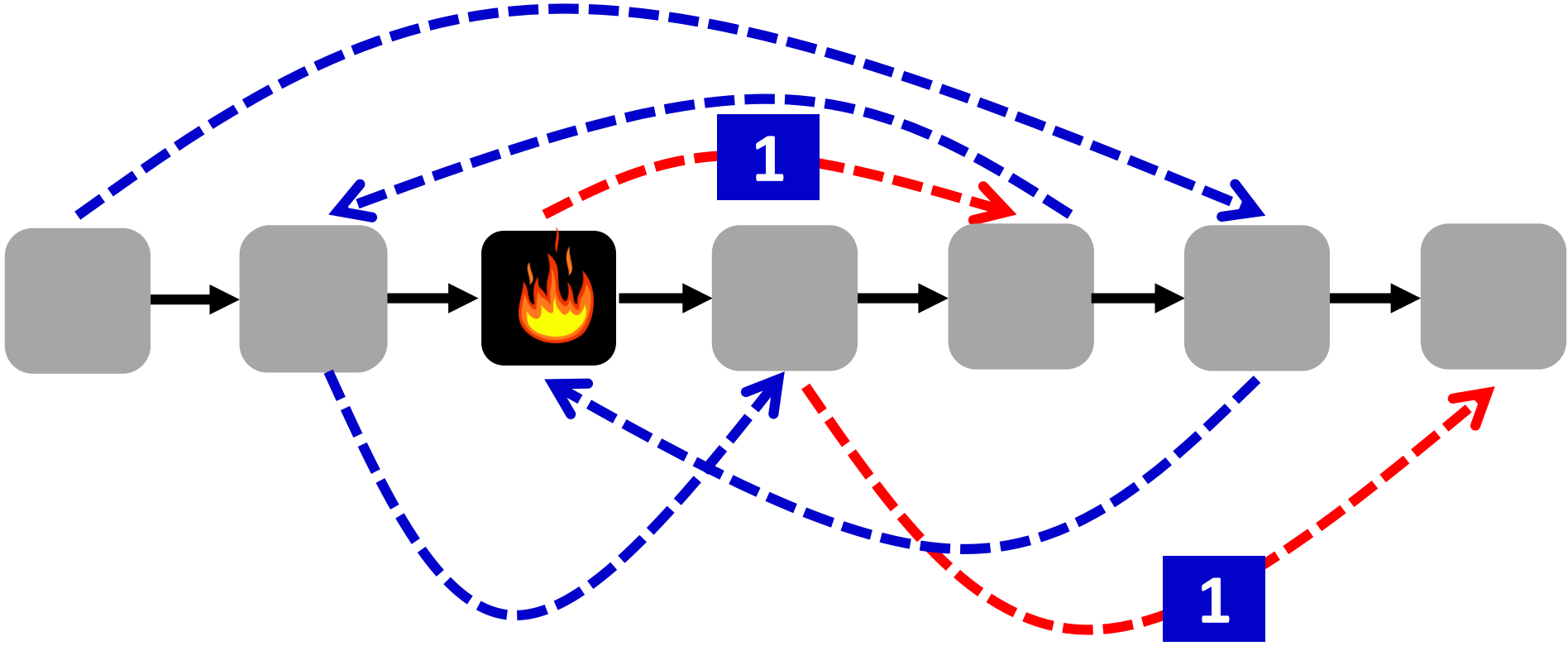
Back to the start: What if….



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

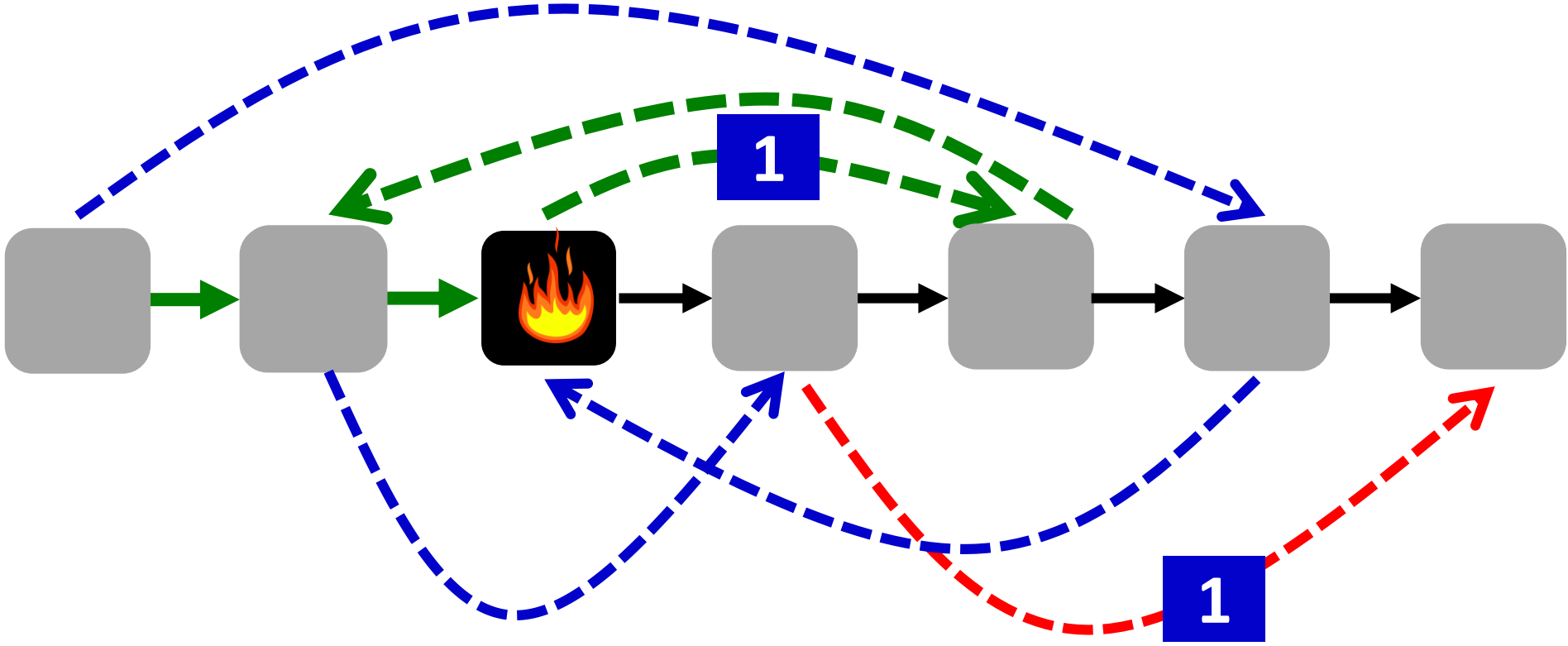


Back to the start: What if…. also this one?!



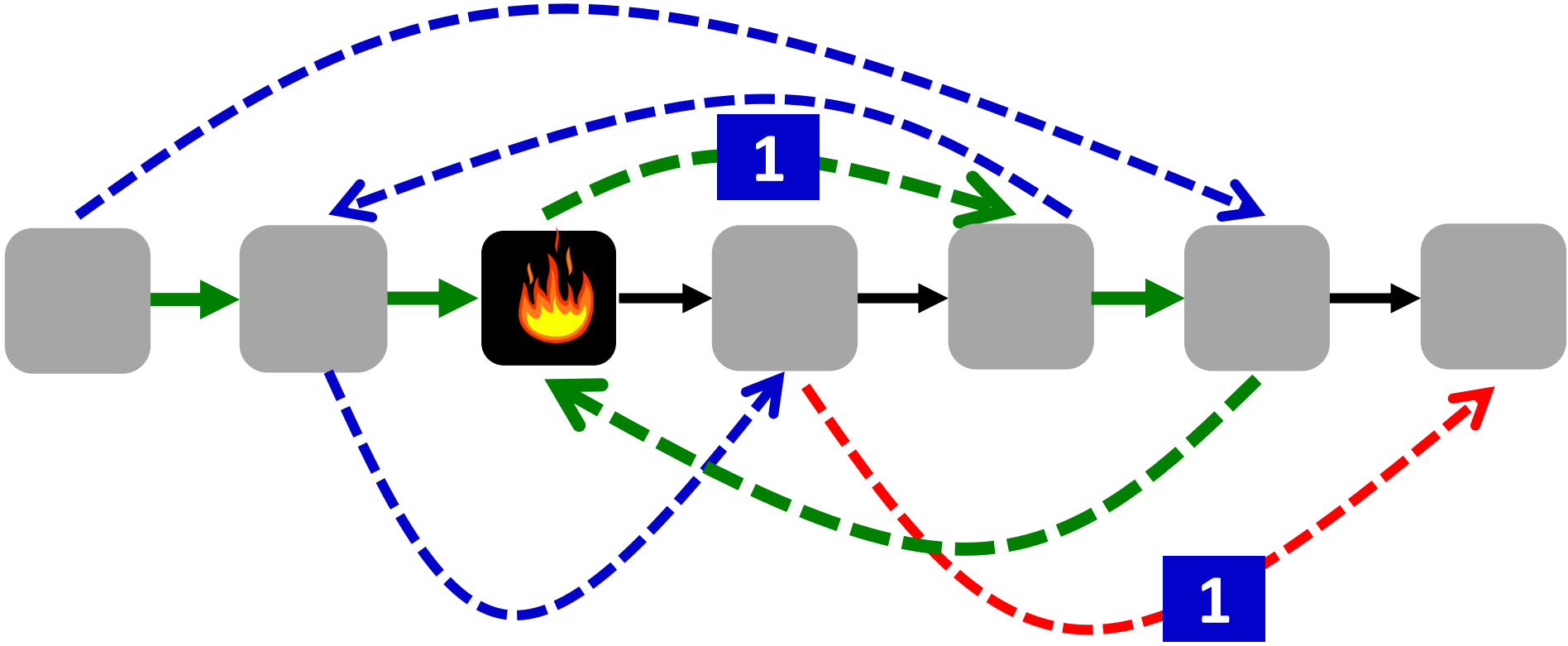
❑ Update any of the 2 backward edges? LF ☹

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



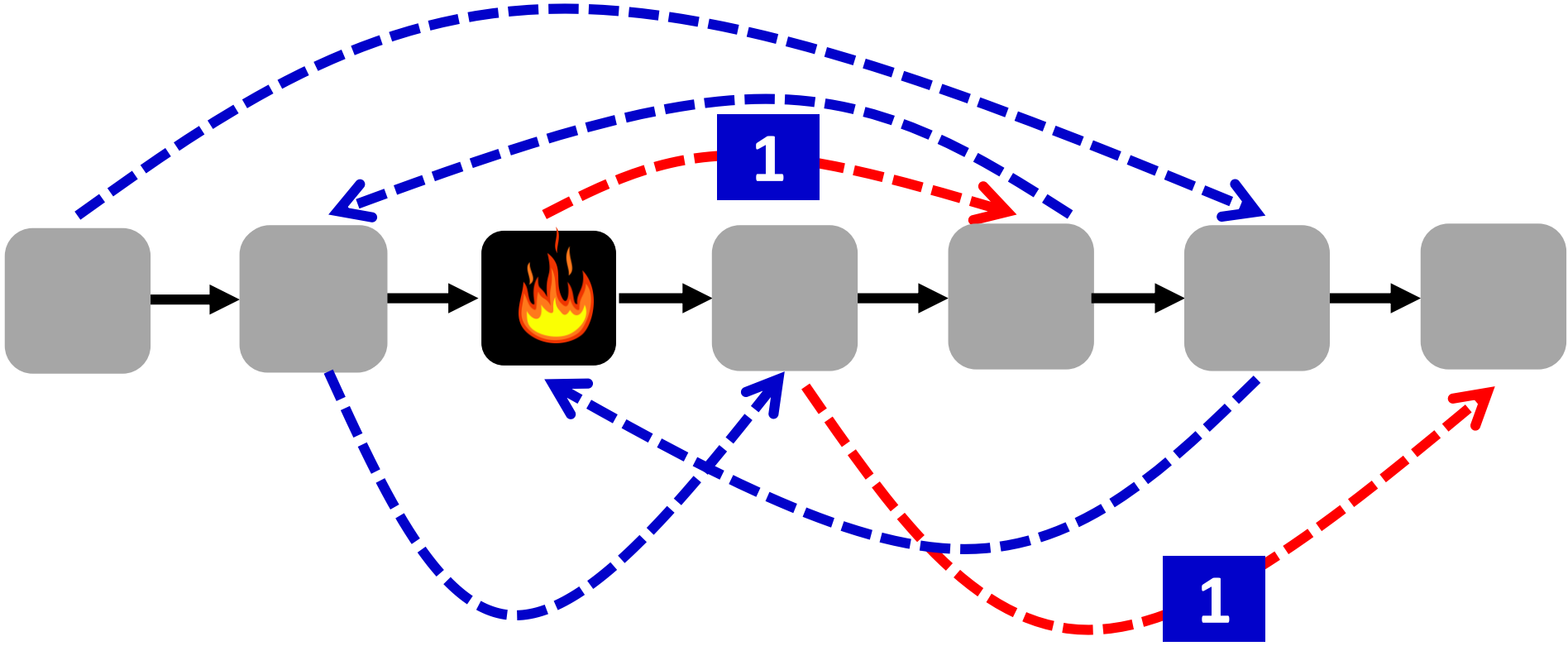
❑ Update any of the 2 backward edges? LF ☹

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



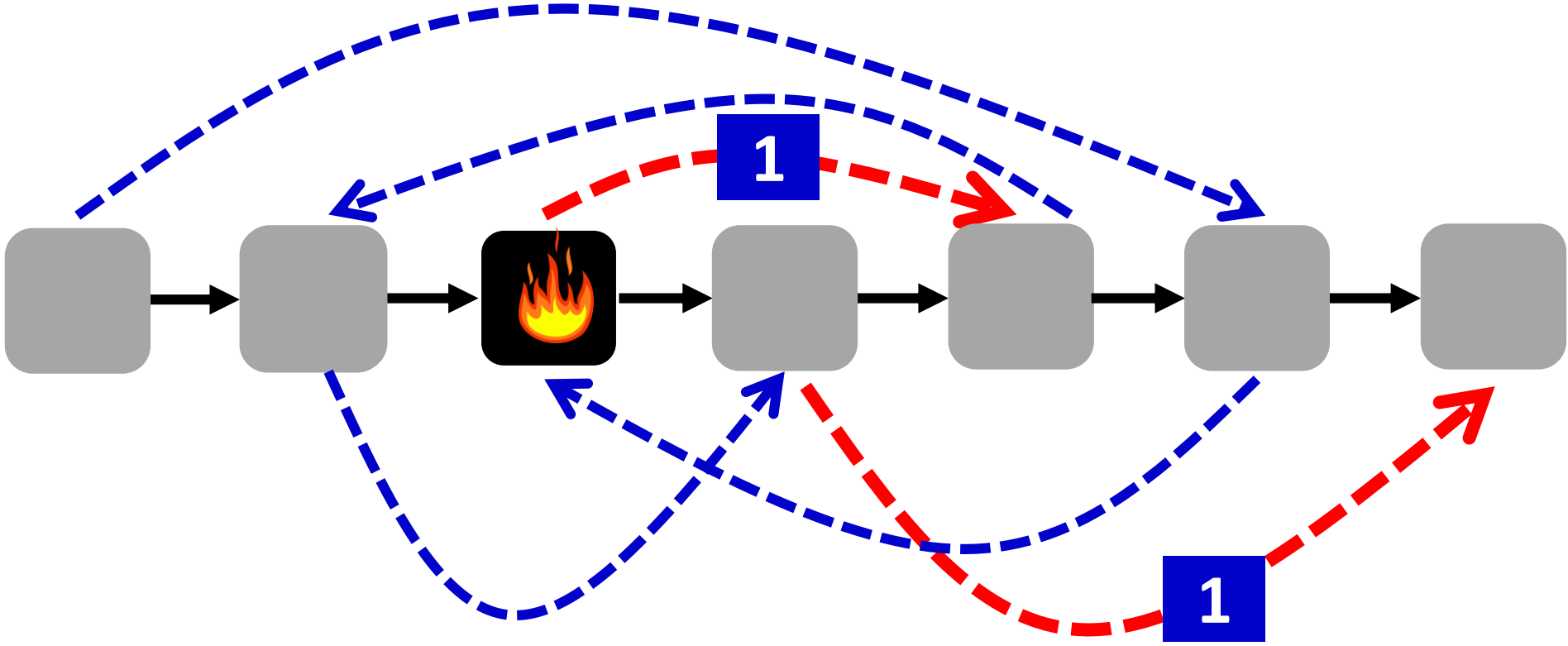
❑ Update any of the 2 backward edges? LF ☹

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

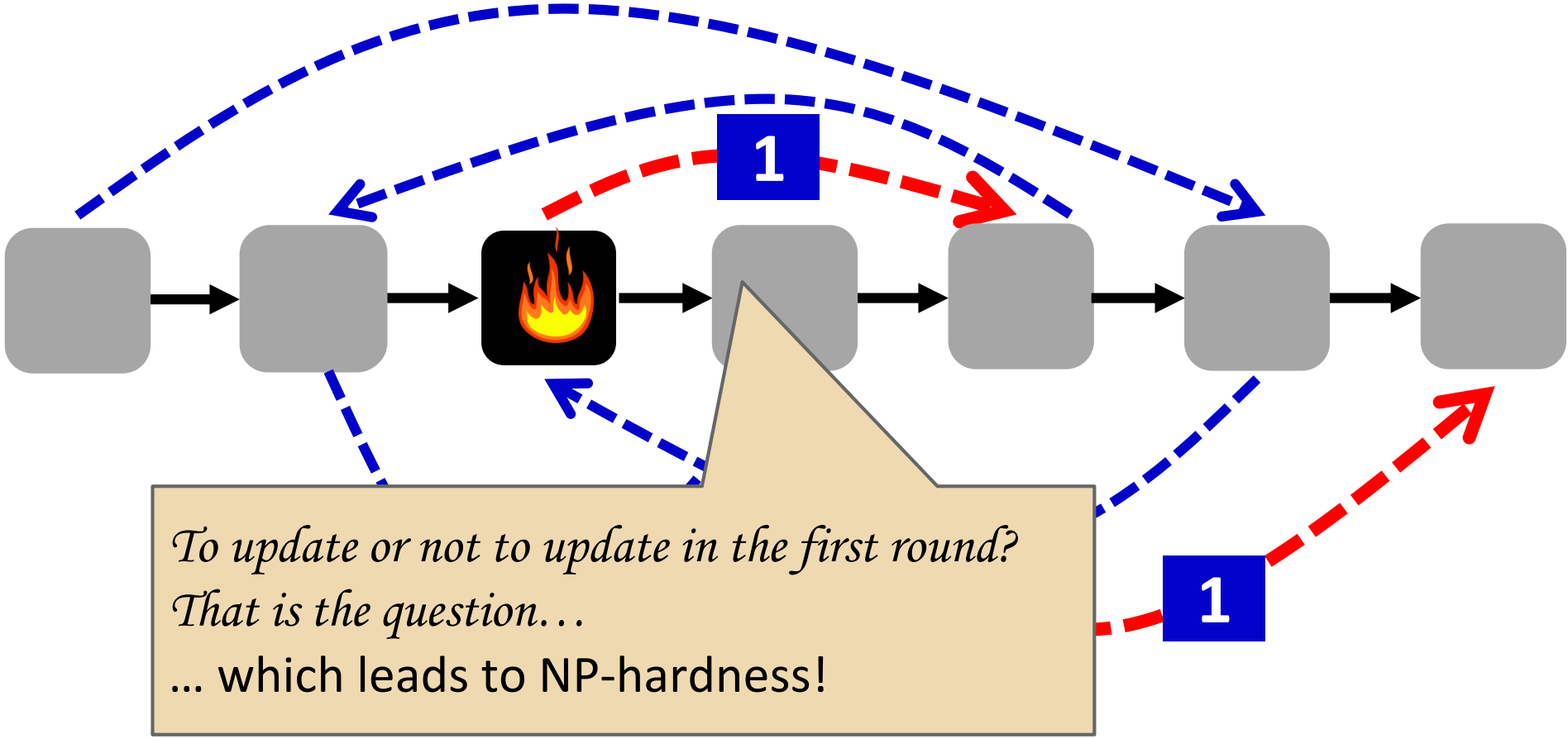


- ❑ Update any of the 2 backward edges? LF ☹
- ❑ Update any of the 2 other forward edges? WPE ☹
- ❑ What about a combination? Nope…

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



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



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

Example: Optimal 2-Round Update Schedules

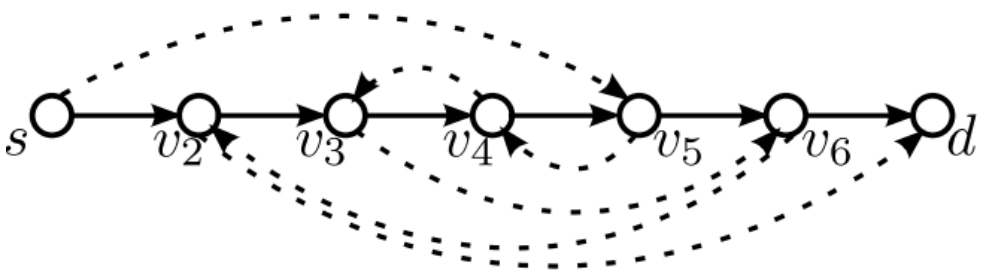
Example: Optimal 2-Round Update Schedules

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**!

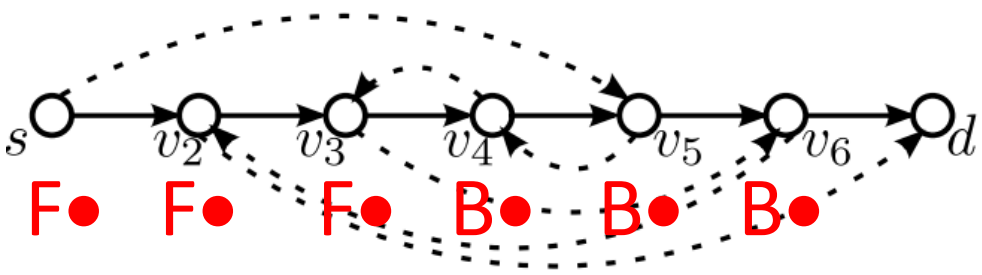
Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

- 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?

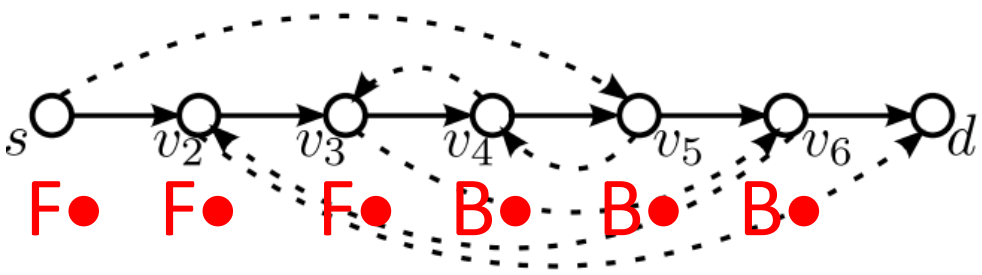


Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

- Classify nodes/edges

Old policy from left to right!

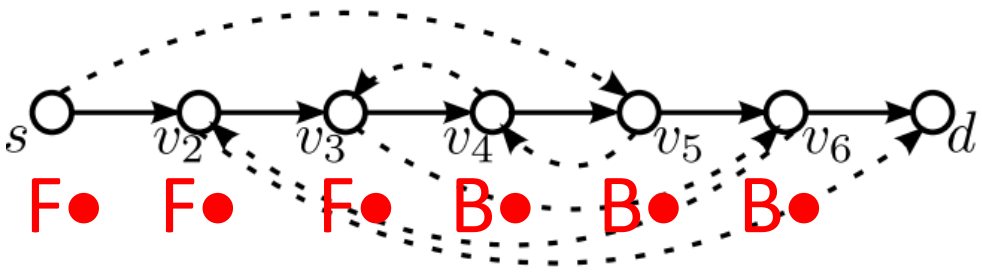
- F●, B●: Does (dashed) new edge point forward or backward wrt (solid) old path?

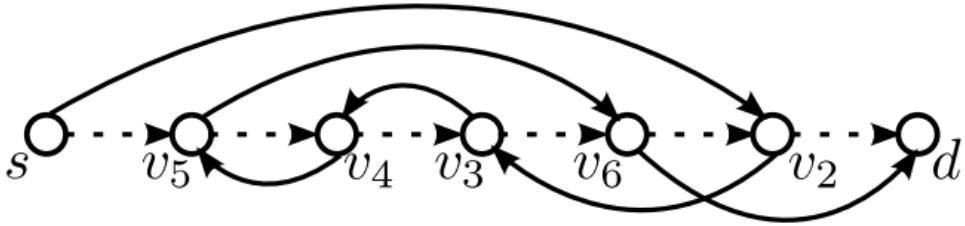


Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

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

Old policy from left to right!





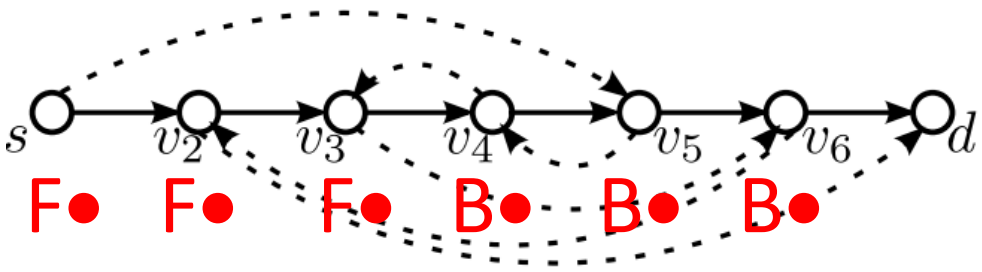
New policy from left to right!

Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

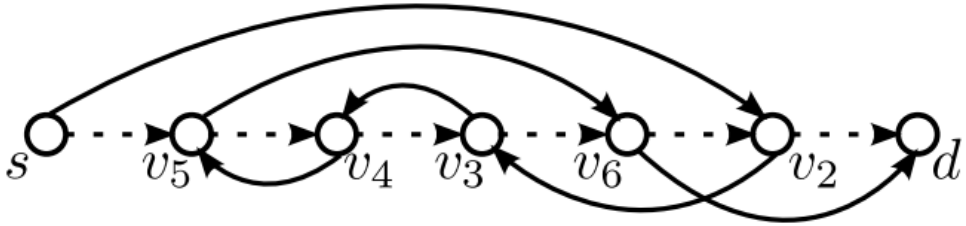
- Classify nodes/edges

Old policy from left to right!

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



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



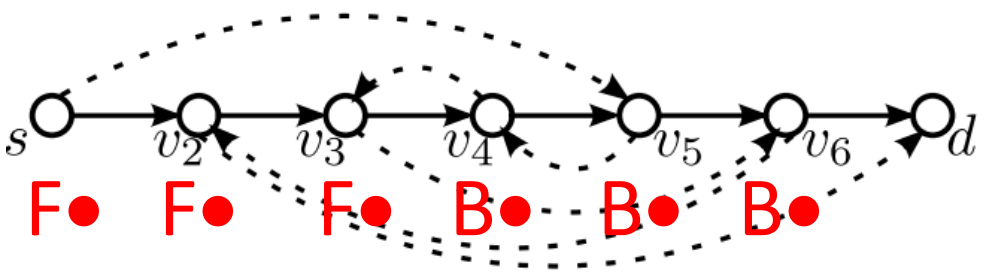
New policy from left to right!

Optimal Algorithm for 2-Round Instances: Leveraging Symmetry!

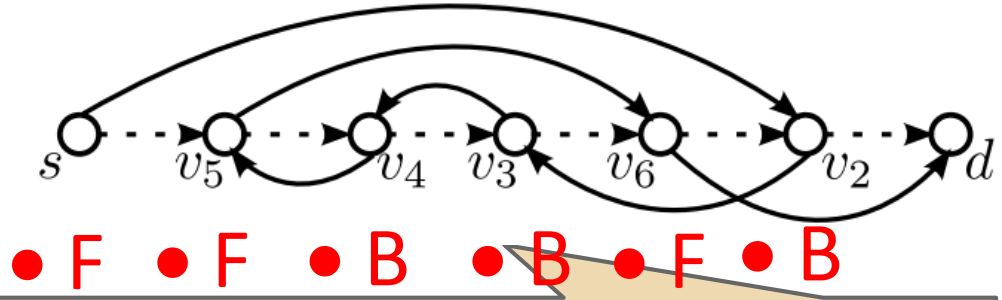
- Classify nodes/edges

Old policy from left to right!

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



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

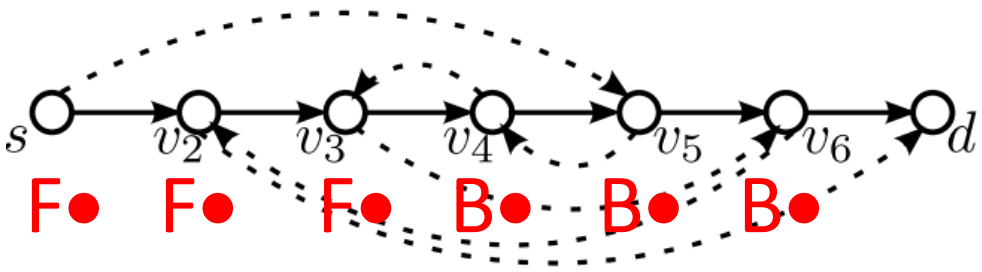


New policy from left to right!

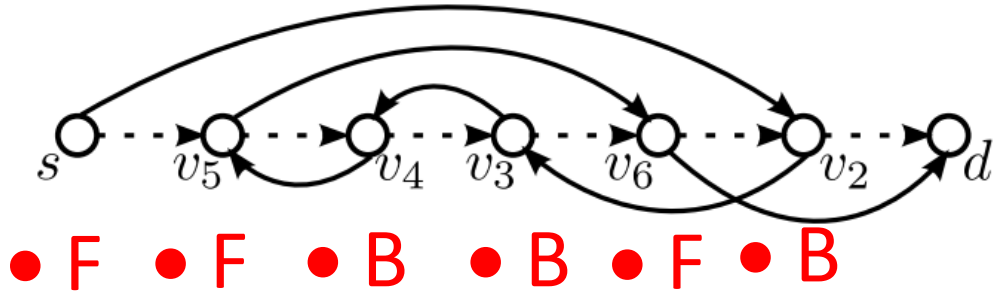
Optimal Algorithm for 2-Round Instances: ging Symmetry!

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

- s 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 backward wrt (dashed) new path?

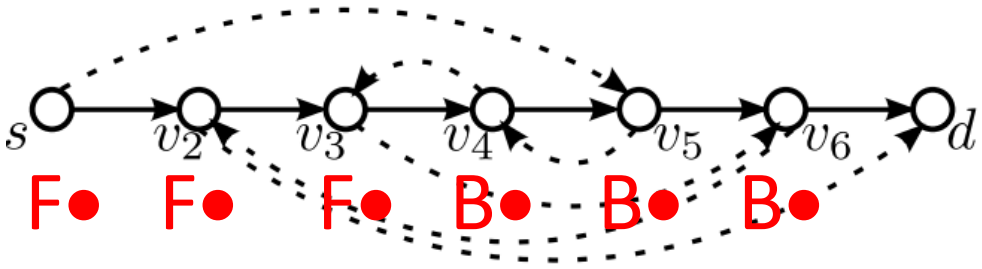


Optimal Algorithm for 2-Round Instances: ging Symmetry!

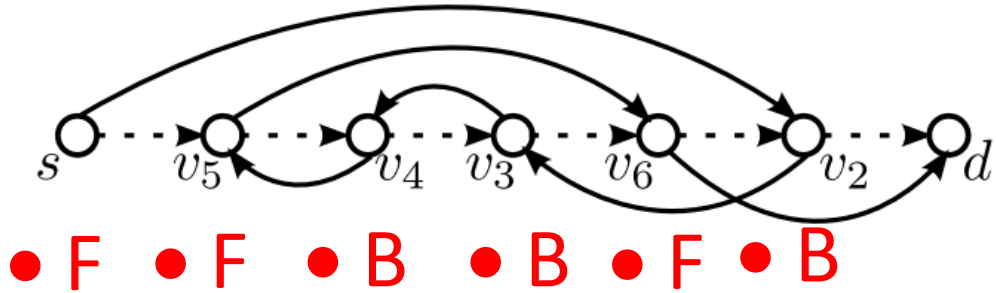
Insight 1: In the 1st round, I can safely update all forwarding (F•) edges! For sure loopfree.

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

s with 2-letter code:



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



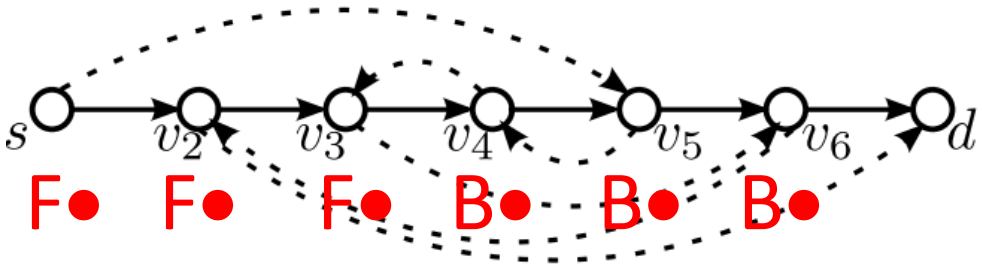
Optimal Algorithm for 2-Round Instances: ging Symmetry!

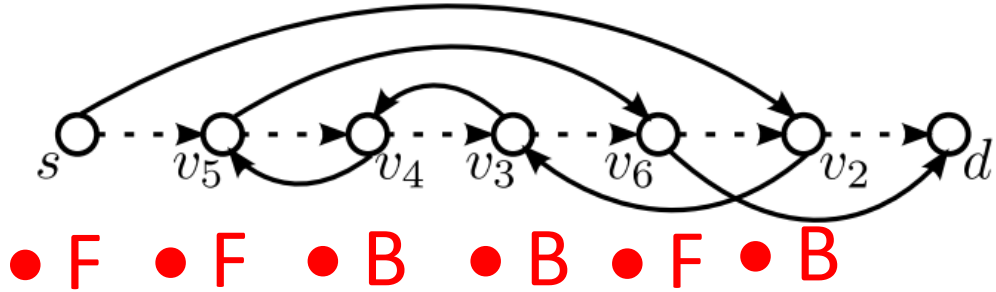
s with 2-letter code:

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!

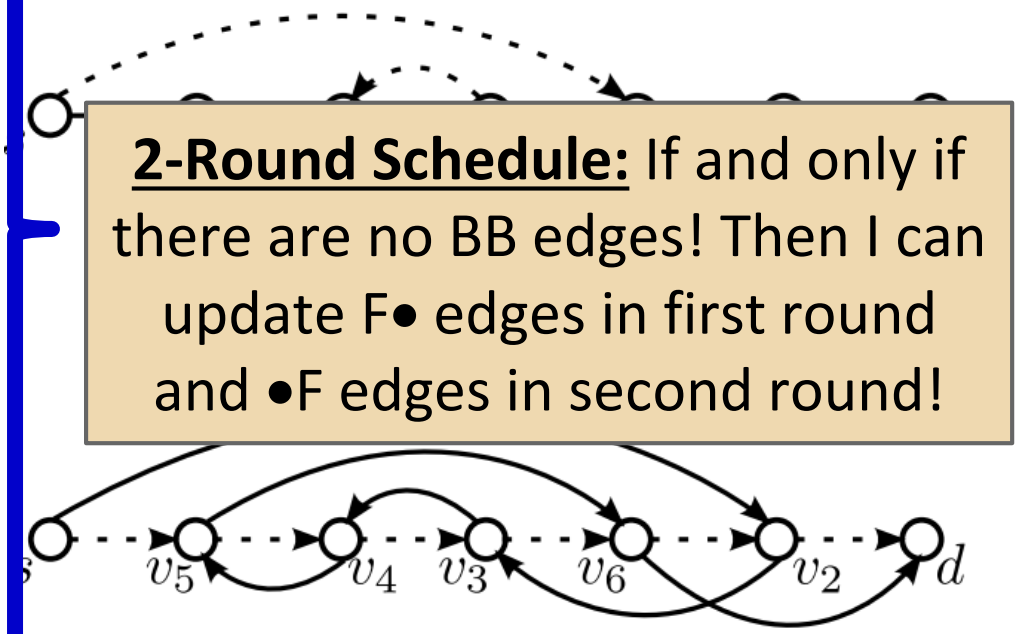
es with 2-letter code:

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!

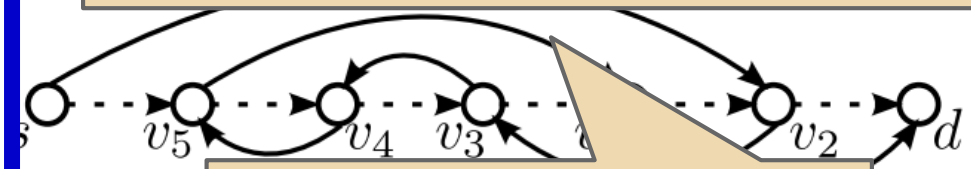
s with 2-letter code:

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.

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*!

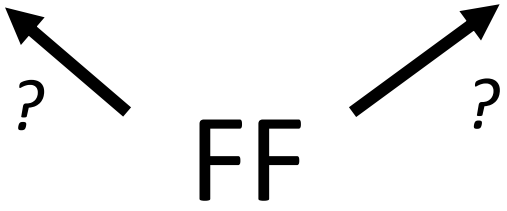
Intuition Why 3 Rounds Are Hard

❑ Structure of a 3-round schedule:

WLOG

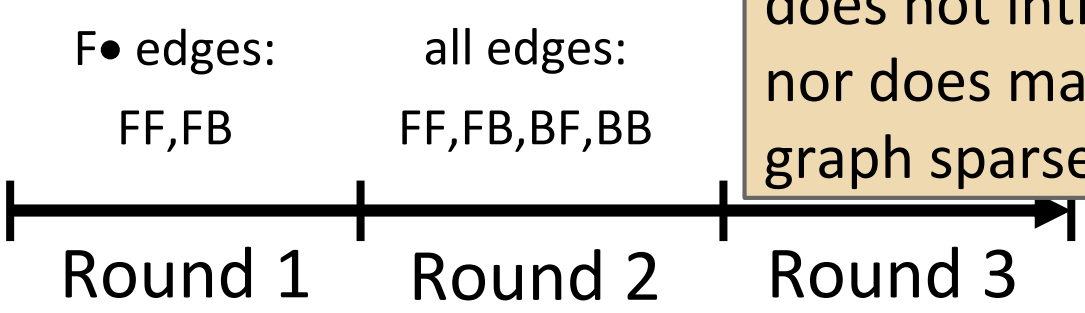
W.l.o.g., can do FB in R1 and BF in R3.

Boils down to:



Intuition Why 3 Rounds Are Hard

❑ Structure of a 3-round sc

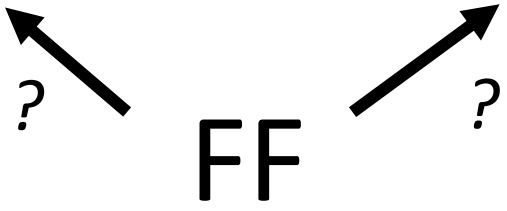


Moving forward edges does not introduce loops, nor does making the graph sparser.

WLOG

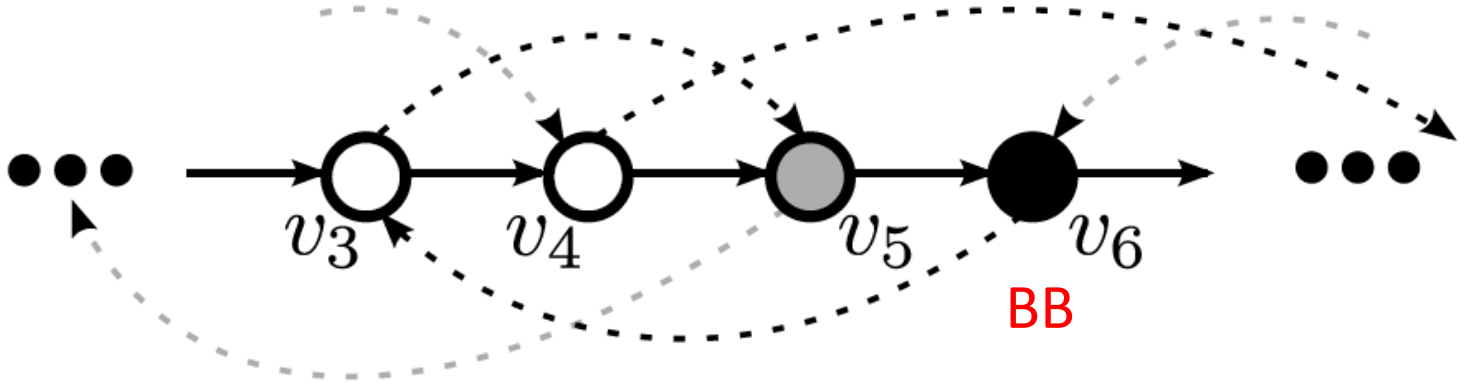
W.l.o.g., can do FB in R1 and BF in R3.

Boils down to:



Intuition Why 3 Rounds Are Hard

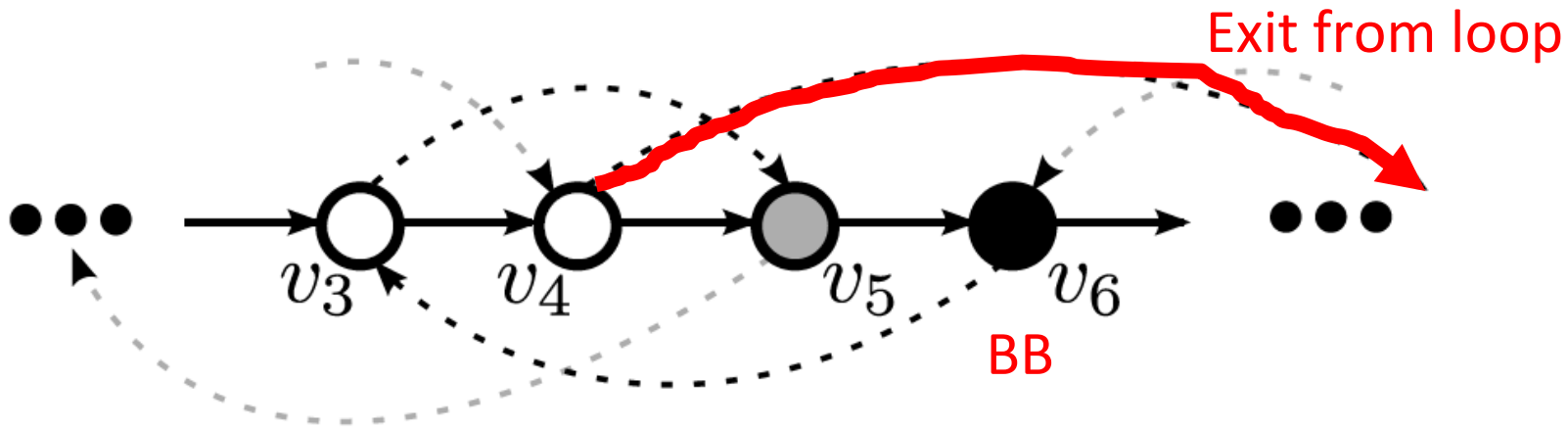
A hard decision problem: when to update FF?



- 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

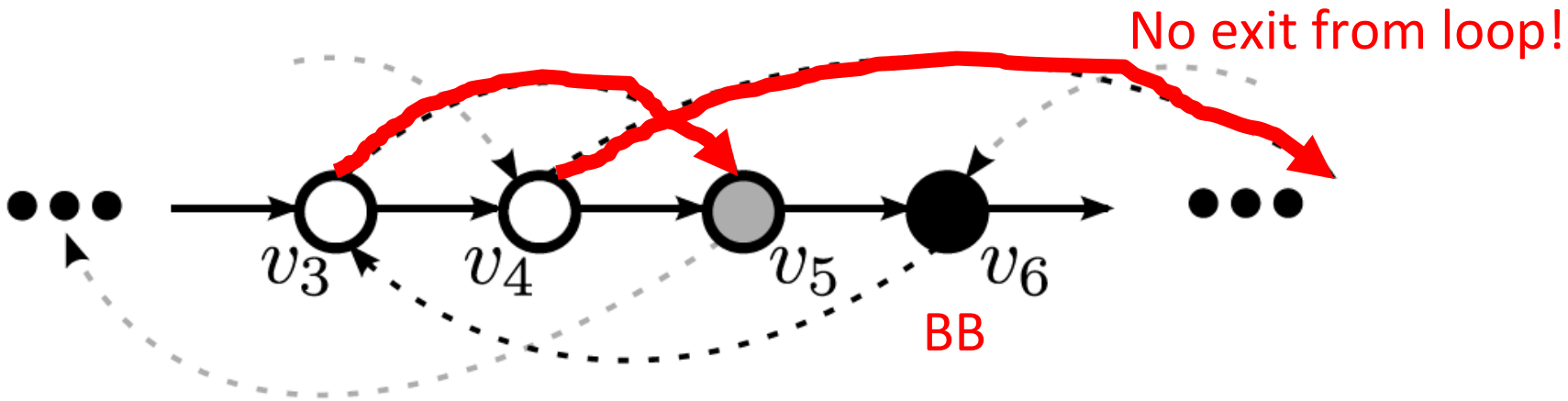
A hard decision problem: when to update FF?



- 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

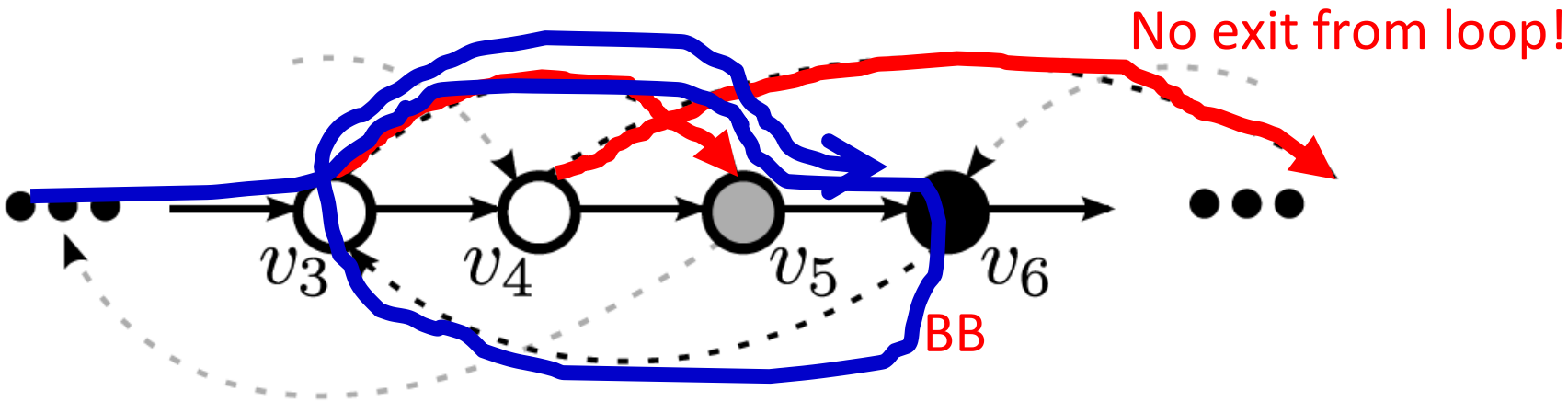
A hard decision problem: when to update FF?



- 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

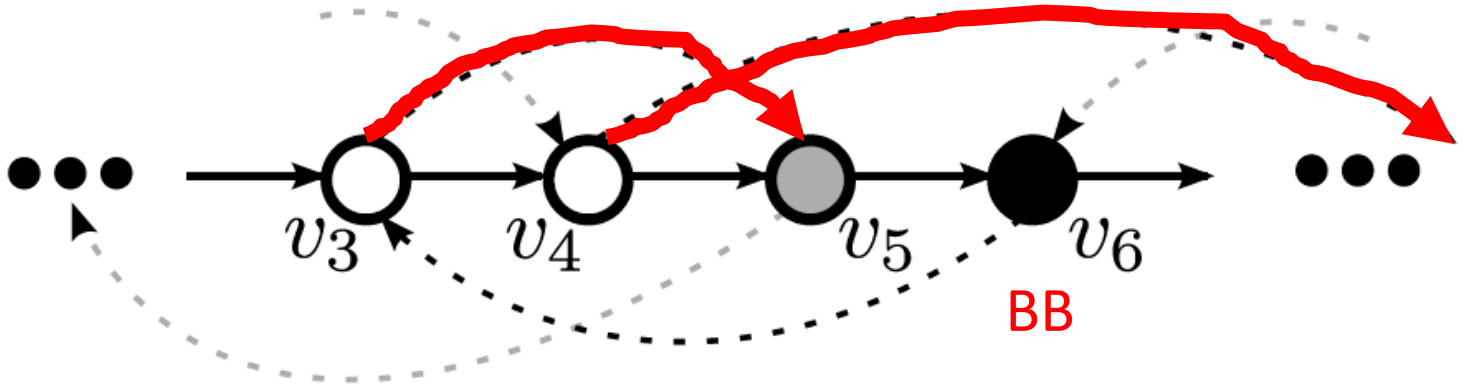
A hard decision problem: when to update FF?



- 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

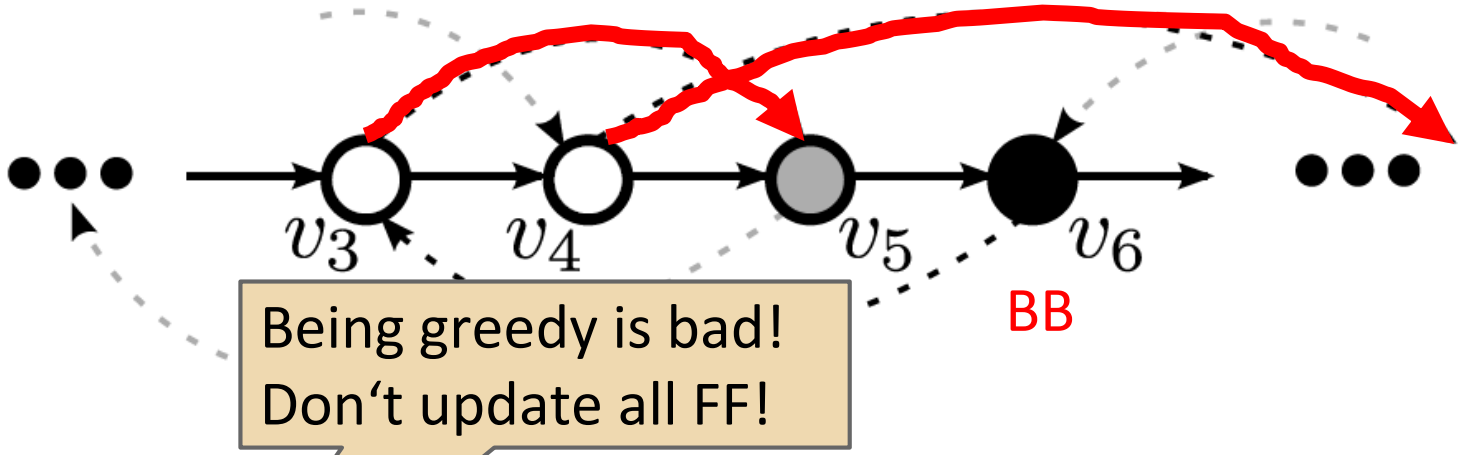
A hard decision problem: when to update FF?



- 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?

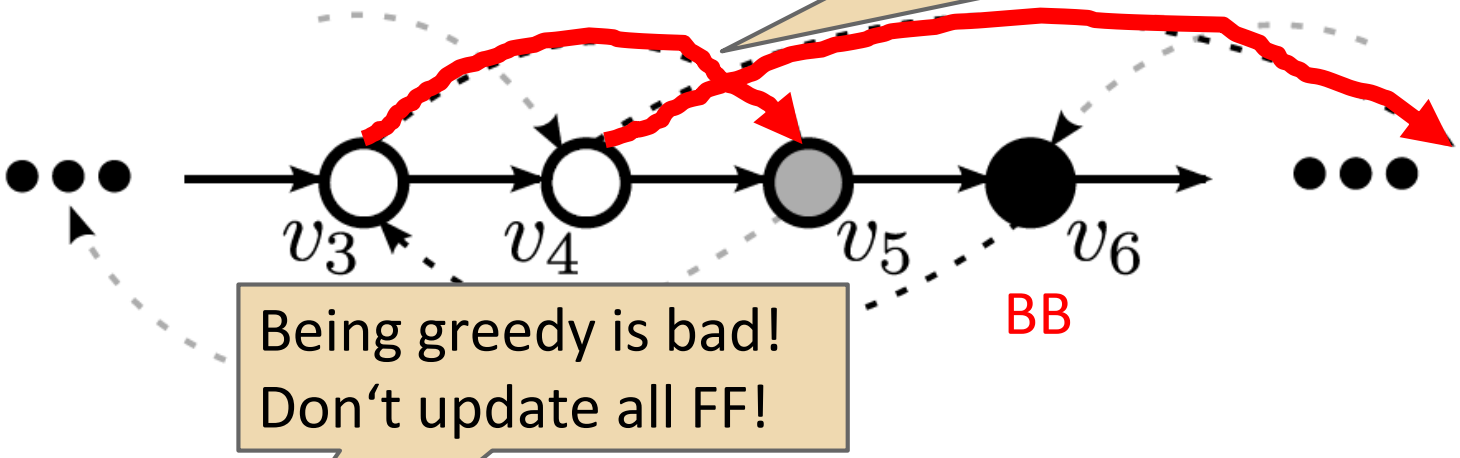


- We know: [illegible]ode v_6 can only be updated in R2
- When to [illegible]pdate FF nodes to make enable update BB in R2
- E.g, updating FF-node v_4 in R1 allows to update BB v_6 in R2
- But only if FF-node v_3 is not updated as well in R1: potential loop
- Smells like a gadget: which FF nodes to update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

Intuition Why 3

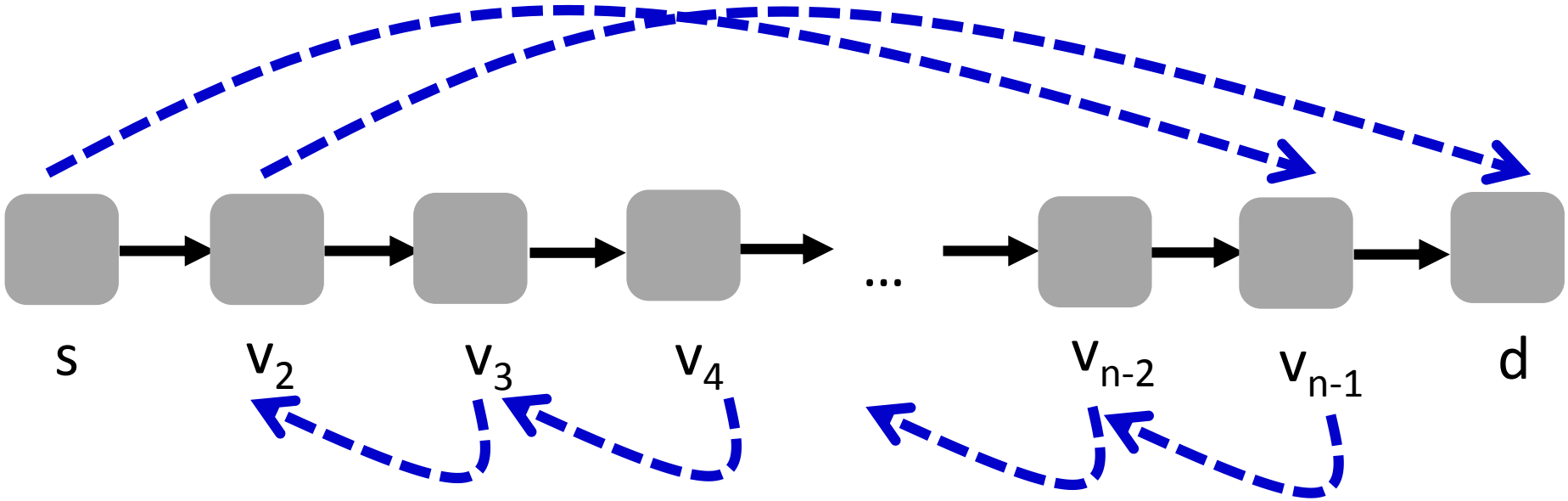
A hard decision problem:

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

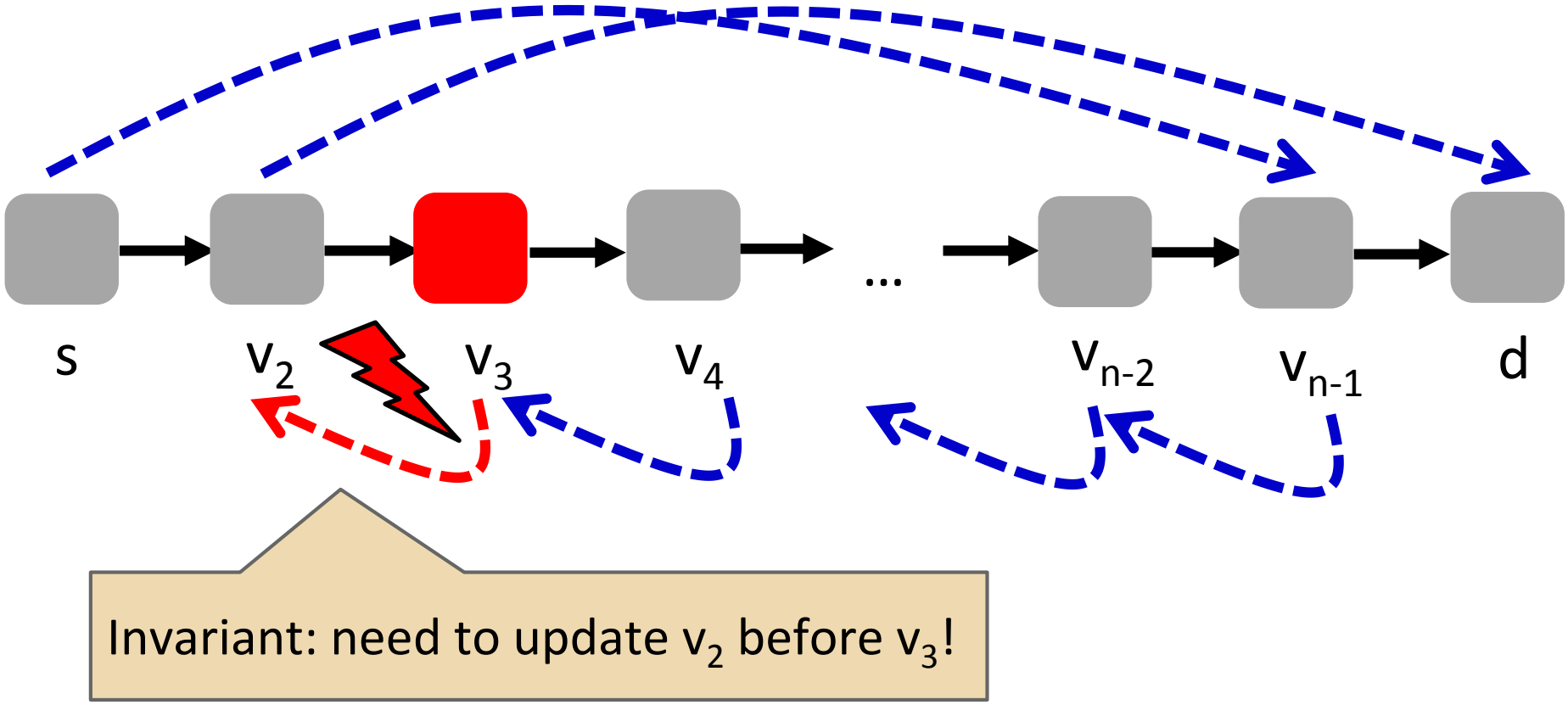


- We know: [illegible]ode v_6 can only be updated in R2
- When to [illegible]pdate FF nodes to make enable update BB in R2
- E.g, updating FF-node v_4 in R1 allows to update BB v_6 in R2
- But only if FF-node v_3 is not updated as well in R1: potential loop
- Smells like a gadget: which FF nodes to update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

It‘s Good to Relax: How to update LF?

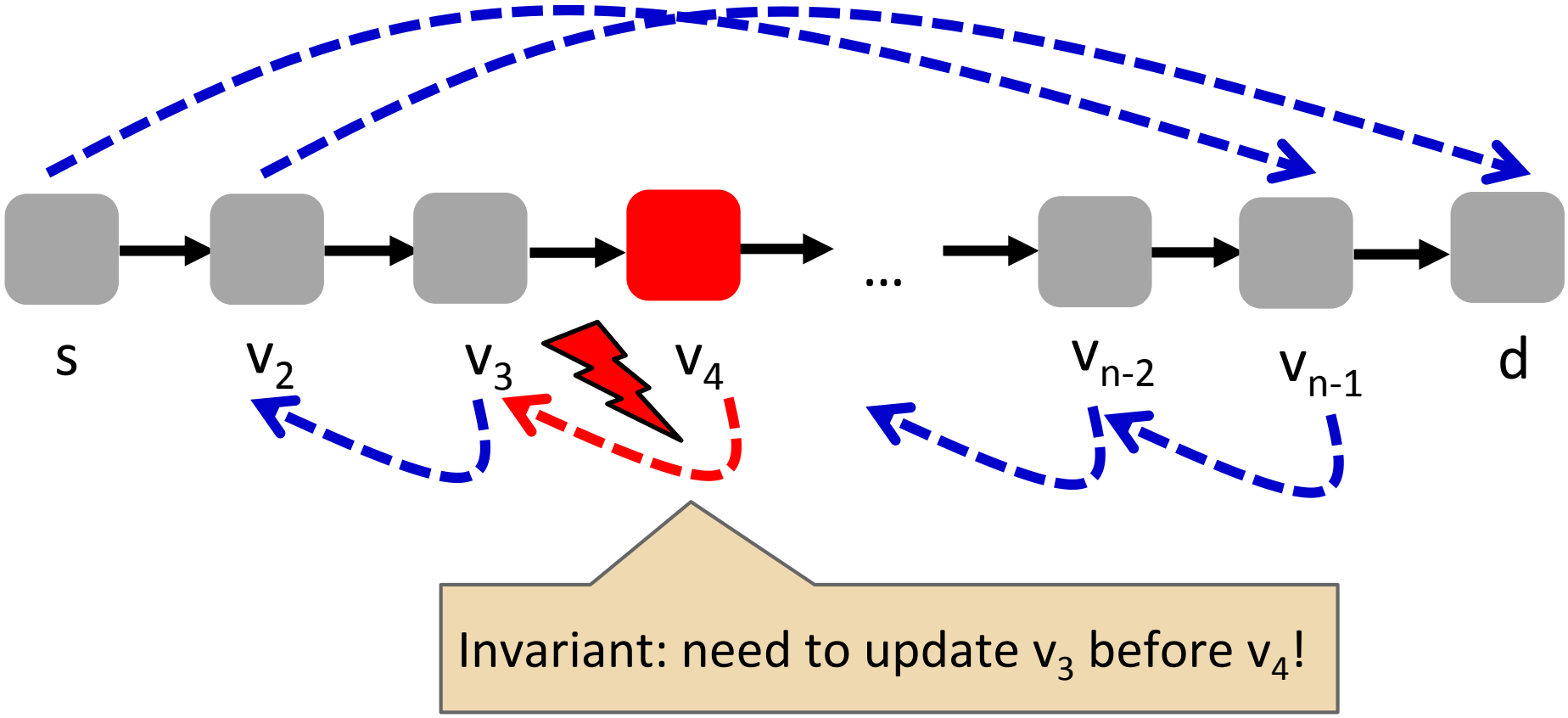


LF Updates Can Take Many Rounds!

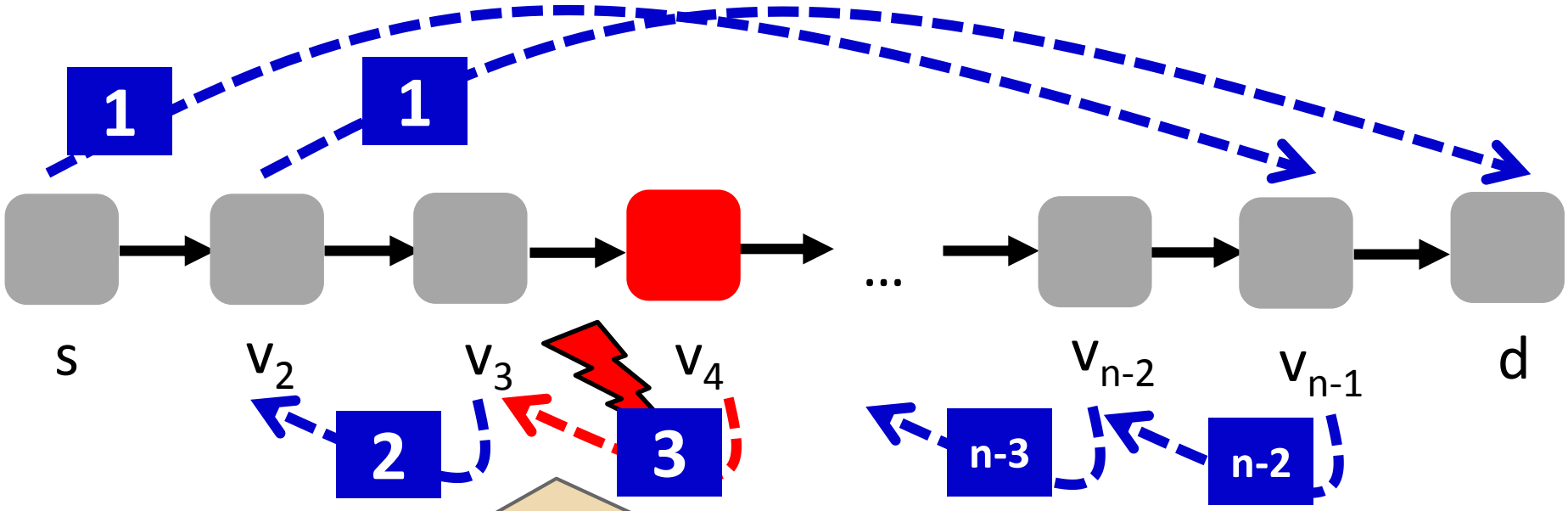


Invariant: need to update v_2 before v_3!

LF Updates Can Take Many Rounds!



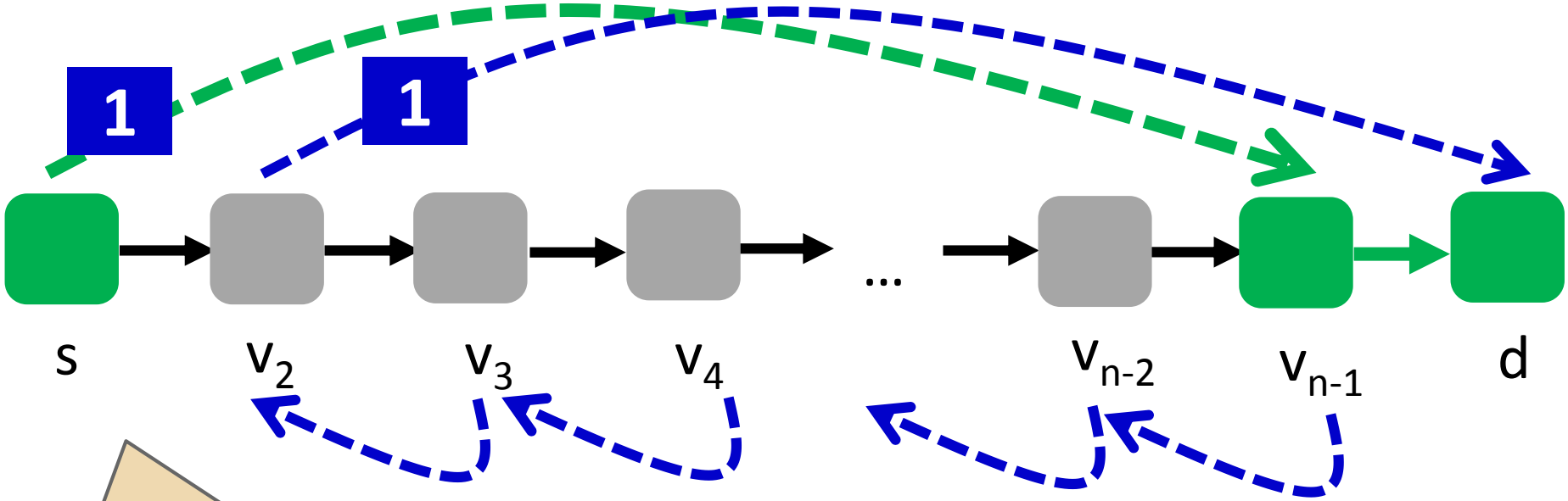
LF Updates Can Take Many Rounds!



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

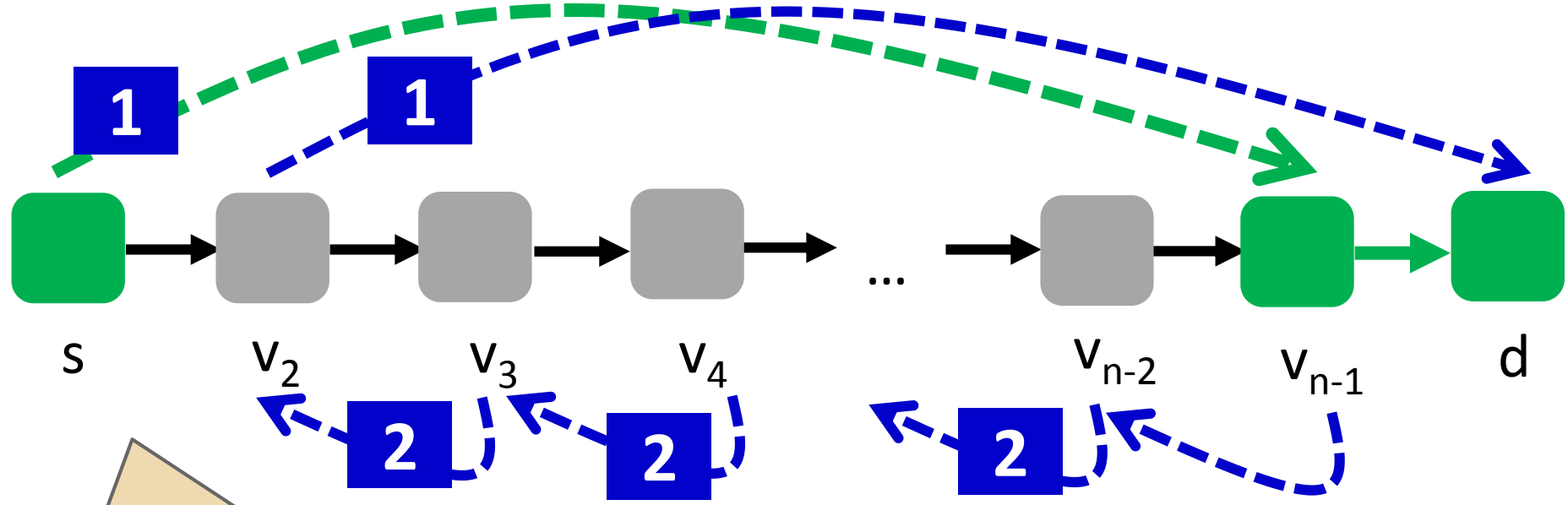
$\Omega(n)$ rounds?! In principle, yes…:
Need a path back out before updating backward edge!

It is good to relax!



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

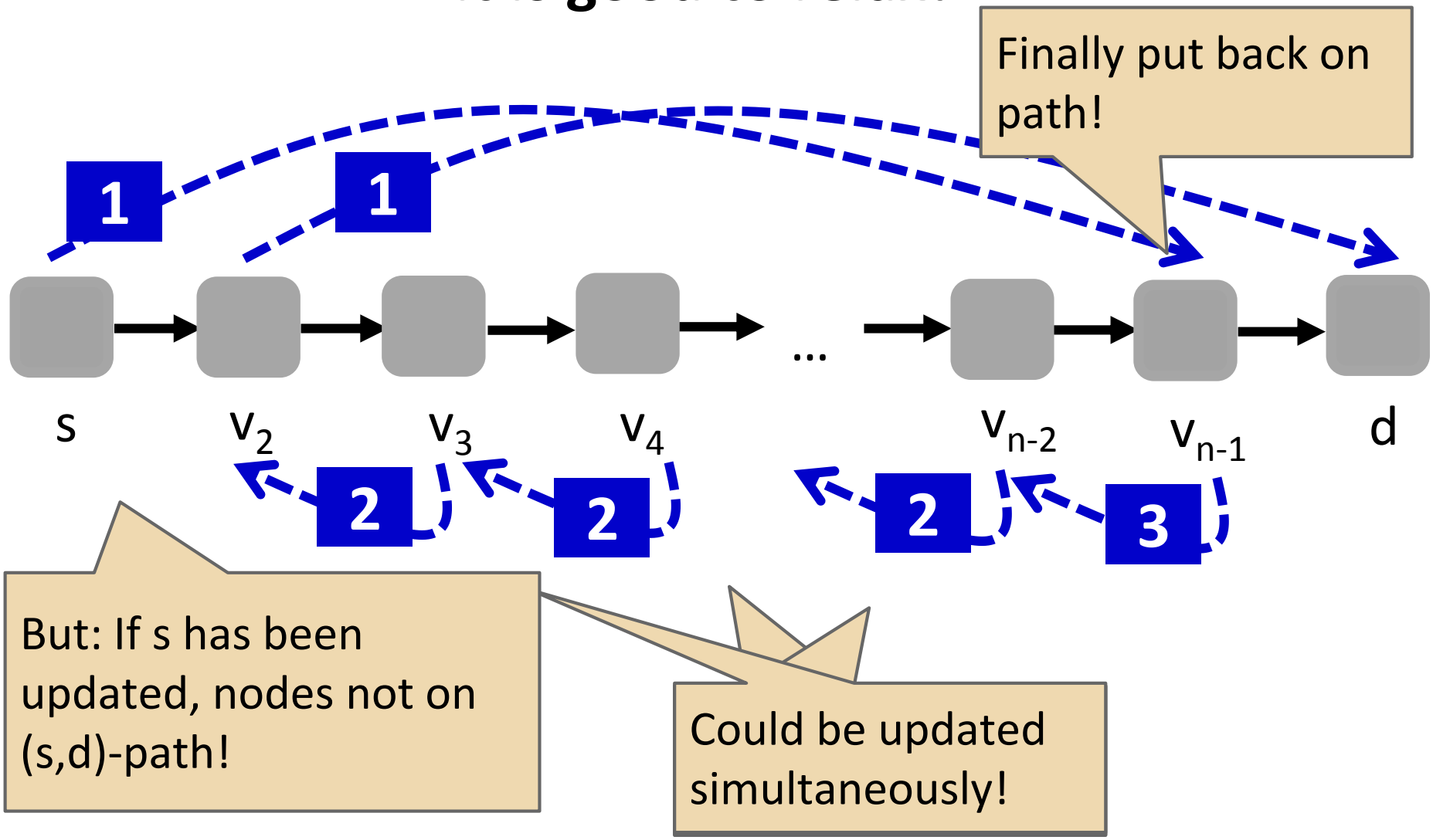
It is good to relax!



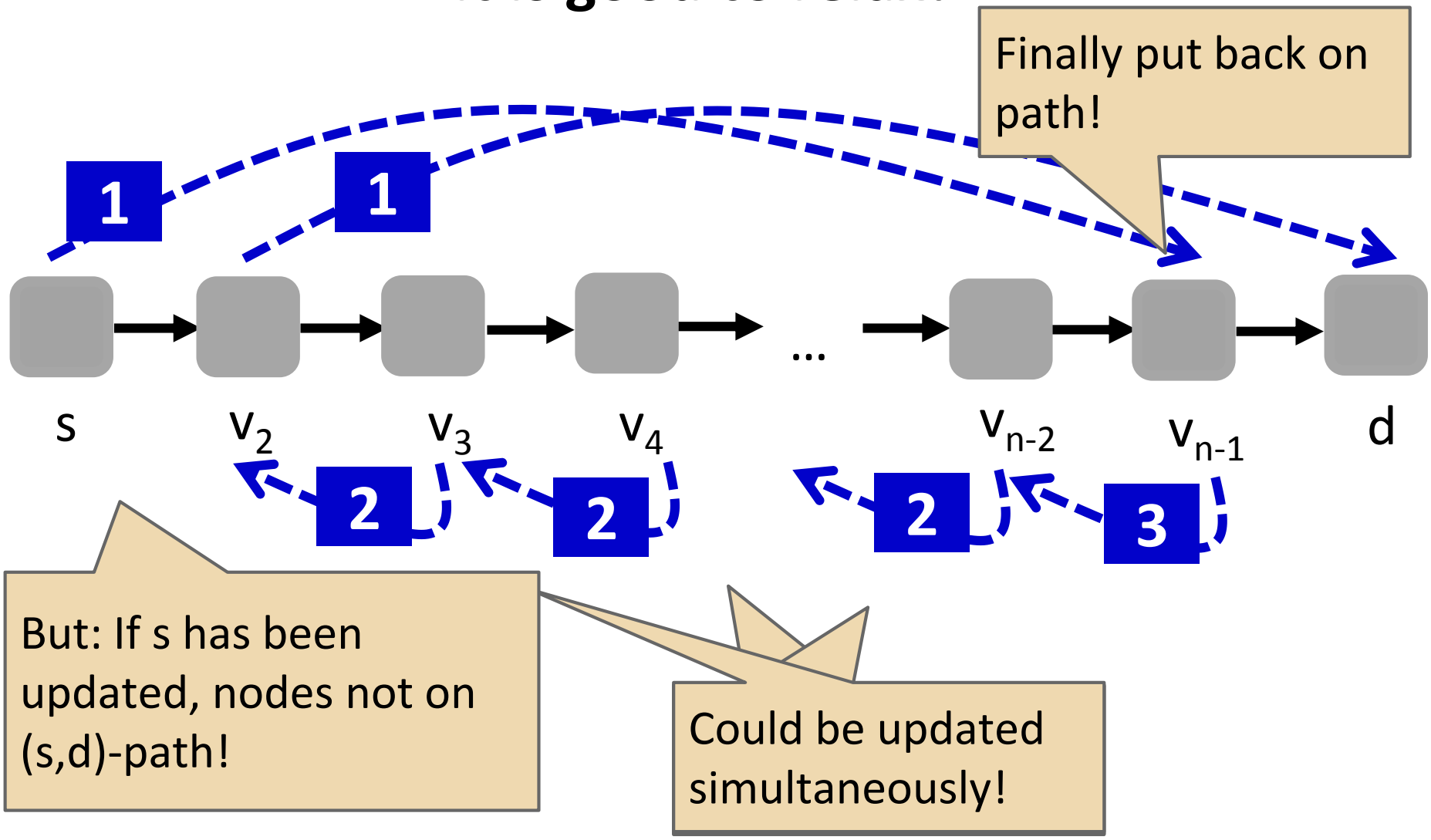
But: If s has been updated, nodes not on (s,d)-path!

Could be updated simultaneously!

It is good to relax!

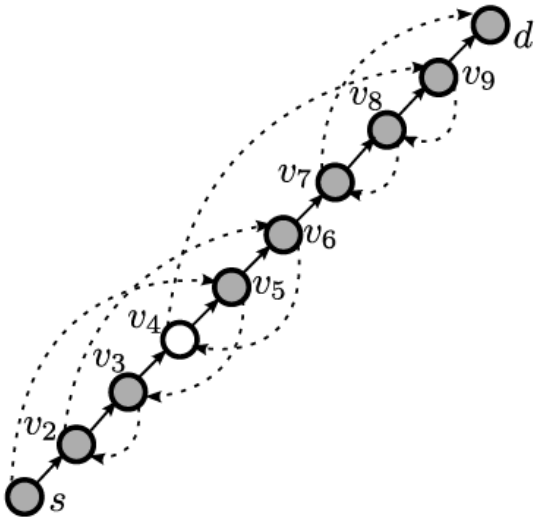


It is good to relax!



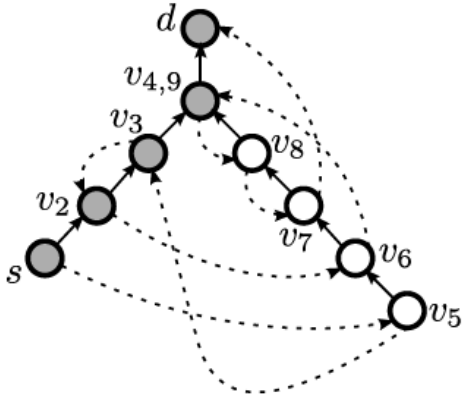
3 rounds only!

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



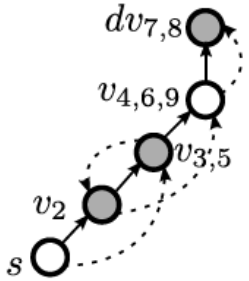
round: 1

Shortcut



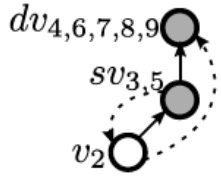
round: 2

Prune



round: 3

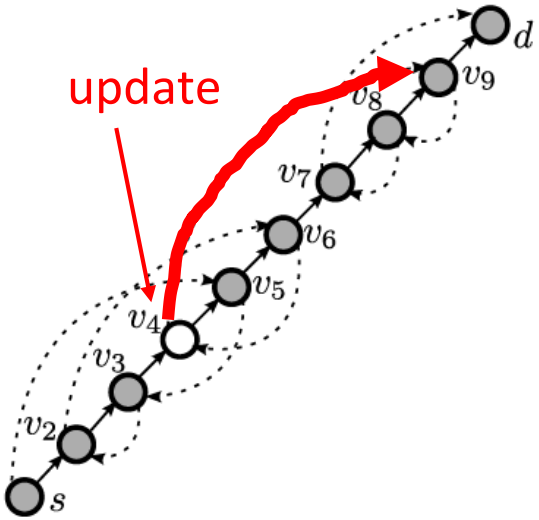
Shortcut



round: 4

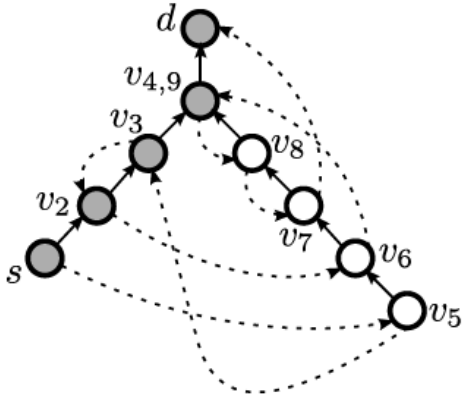
Prune

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



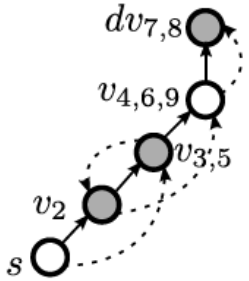
round: 1

Shortcut



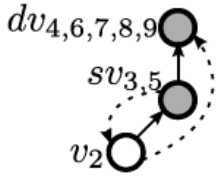
round: 2

Prune



round: 3

Shortcut

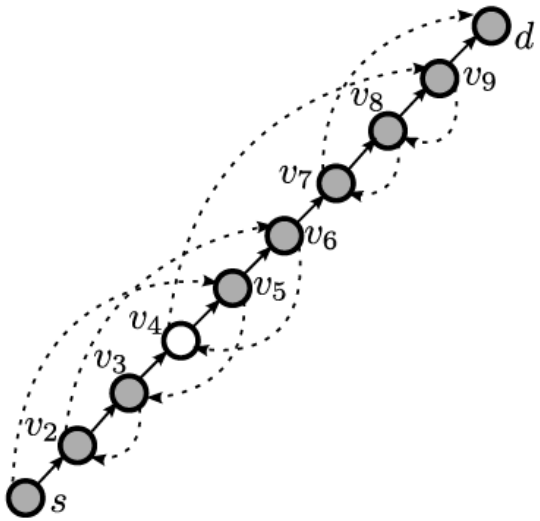


round: 4

Prune

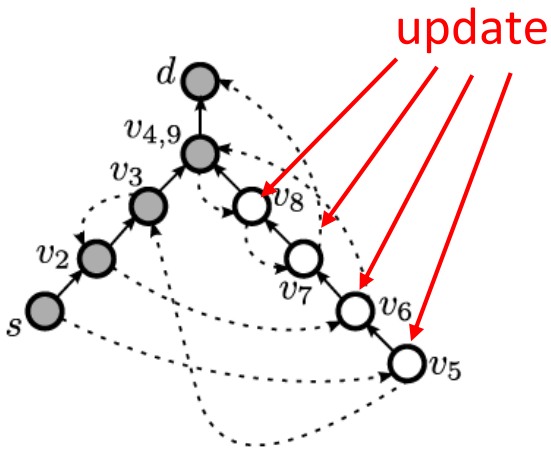
Greedily choose far-reaching (independent) forward edges.

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



round: 1

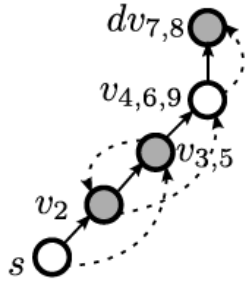
Shortcut



round: 2

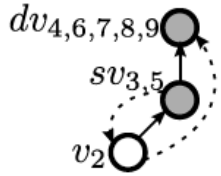
Prune

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



round: 3

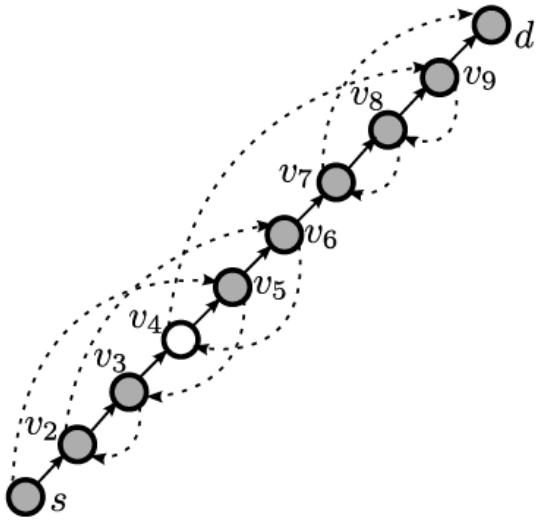
Shortcut



round: 4

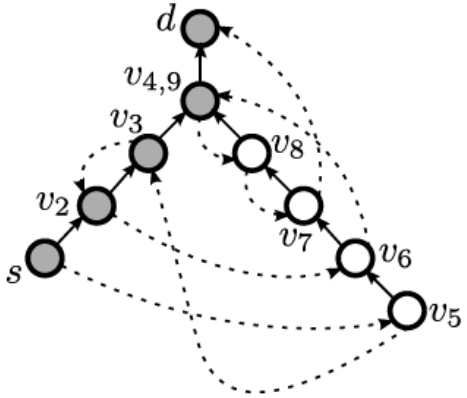
Prune

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



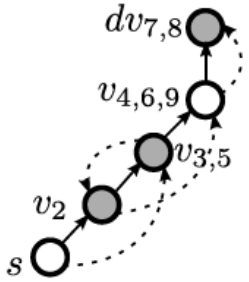
round: 1

Shortcut



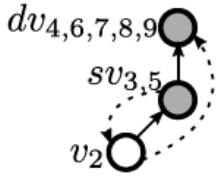
round: 2

Prune



round: 3

Shortcut

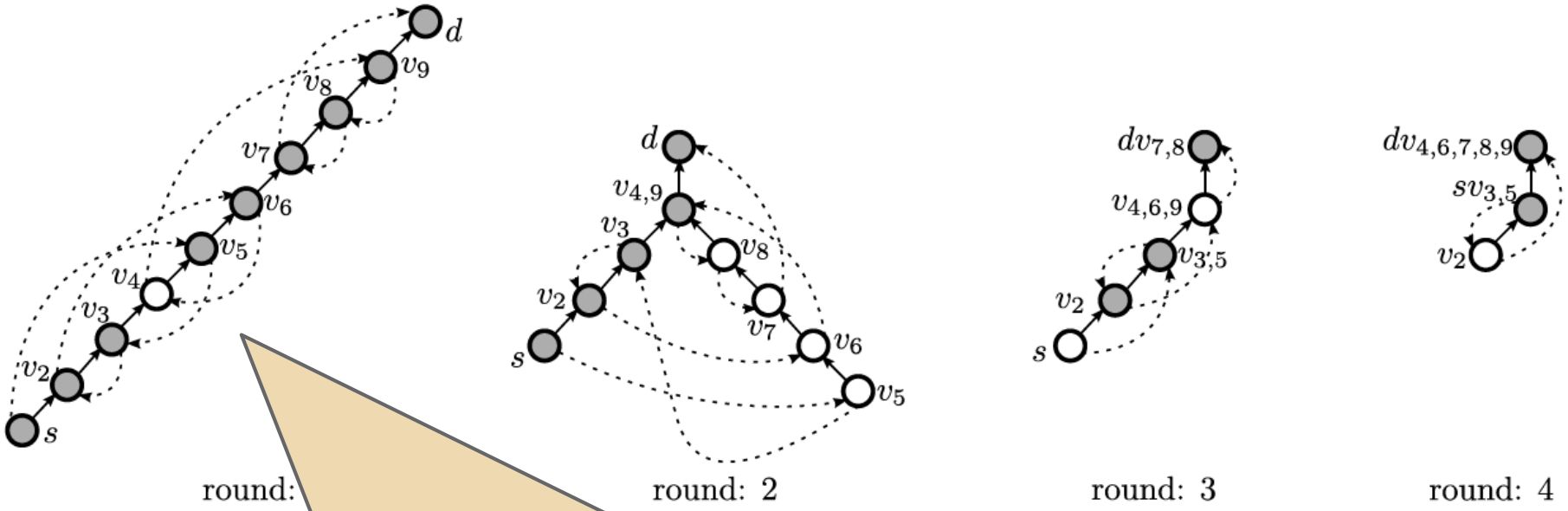


round: 4

Prune

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

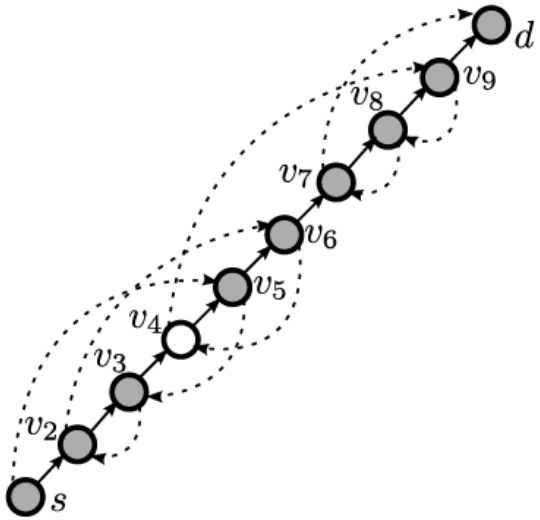
A log(n)-time Algorithm: *Peacock* in Action



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

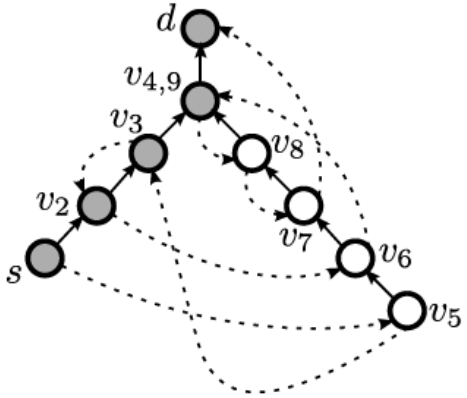
Prune

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



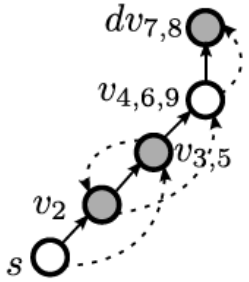
round: 1

Shortcut



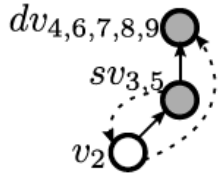
round: 2

Prune



round: 3

Shortcut

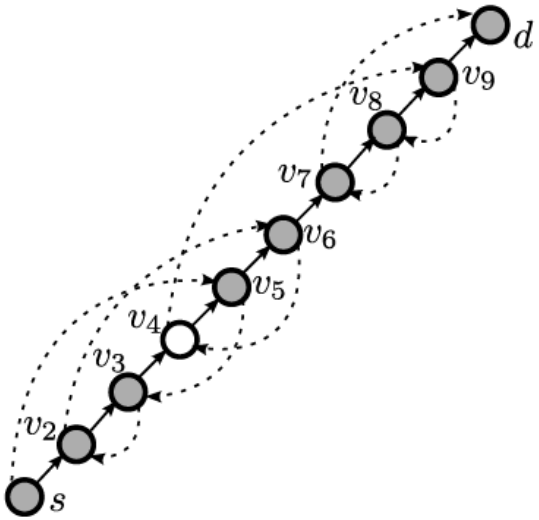


round: 4

Prune

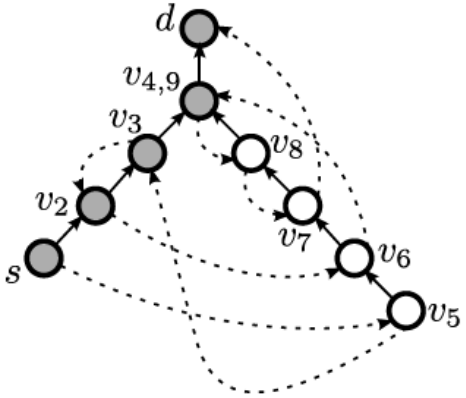
At least 1/3 of nodes merged in each round pair (shorter s-d path): logarithmic runtime!

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



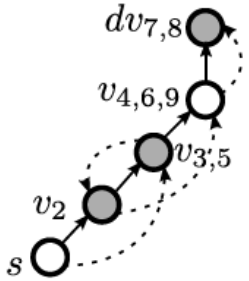
round: 1

Shortcut



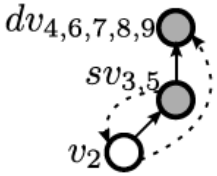
round: 2

Prune



round: 3

Shortcut

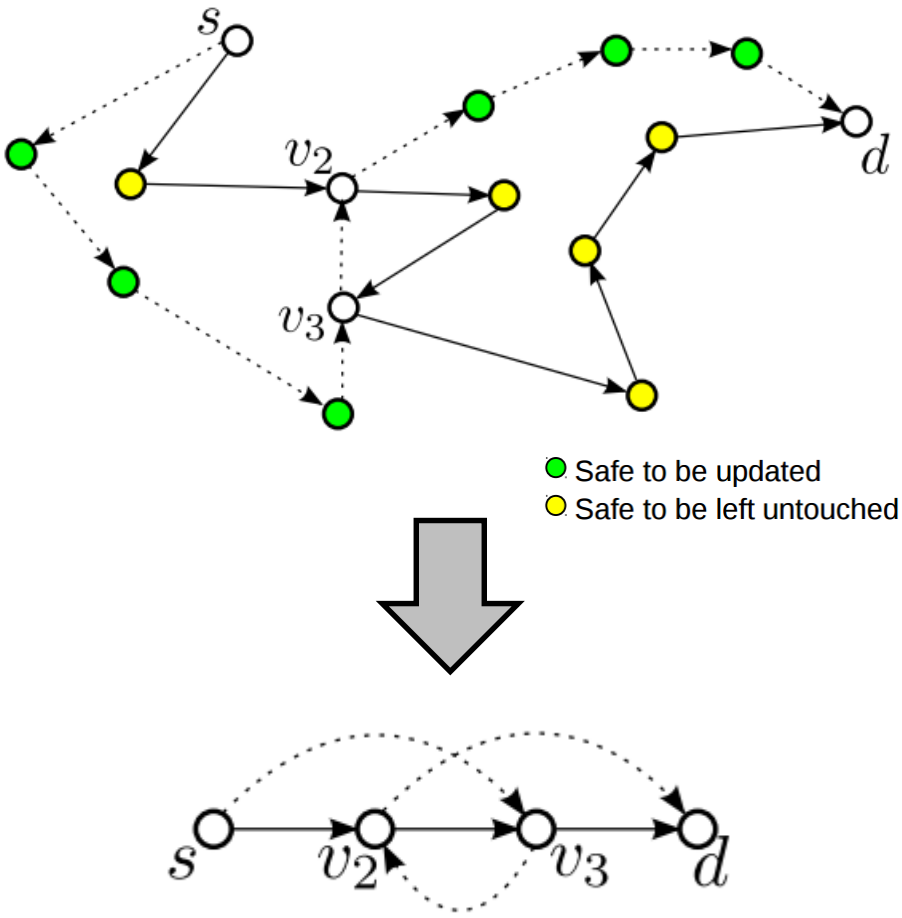


round: 4

Prune

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 $\Omega(n)$ although O(1) possible

Loop-Freedom: Summary of Results

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

Being greedy is bad!
And hard ☺

GIAN Course on Distributed Network Algorithms

Virtual Network Embeddings

Big Data: ~1 Petabyte/day

Science

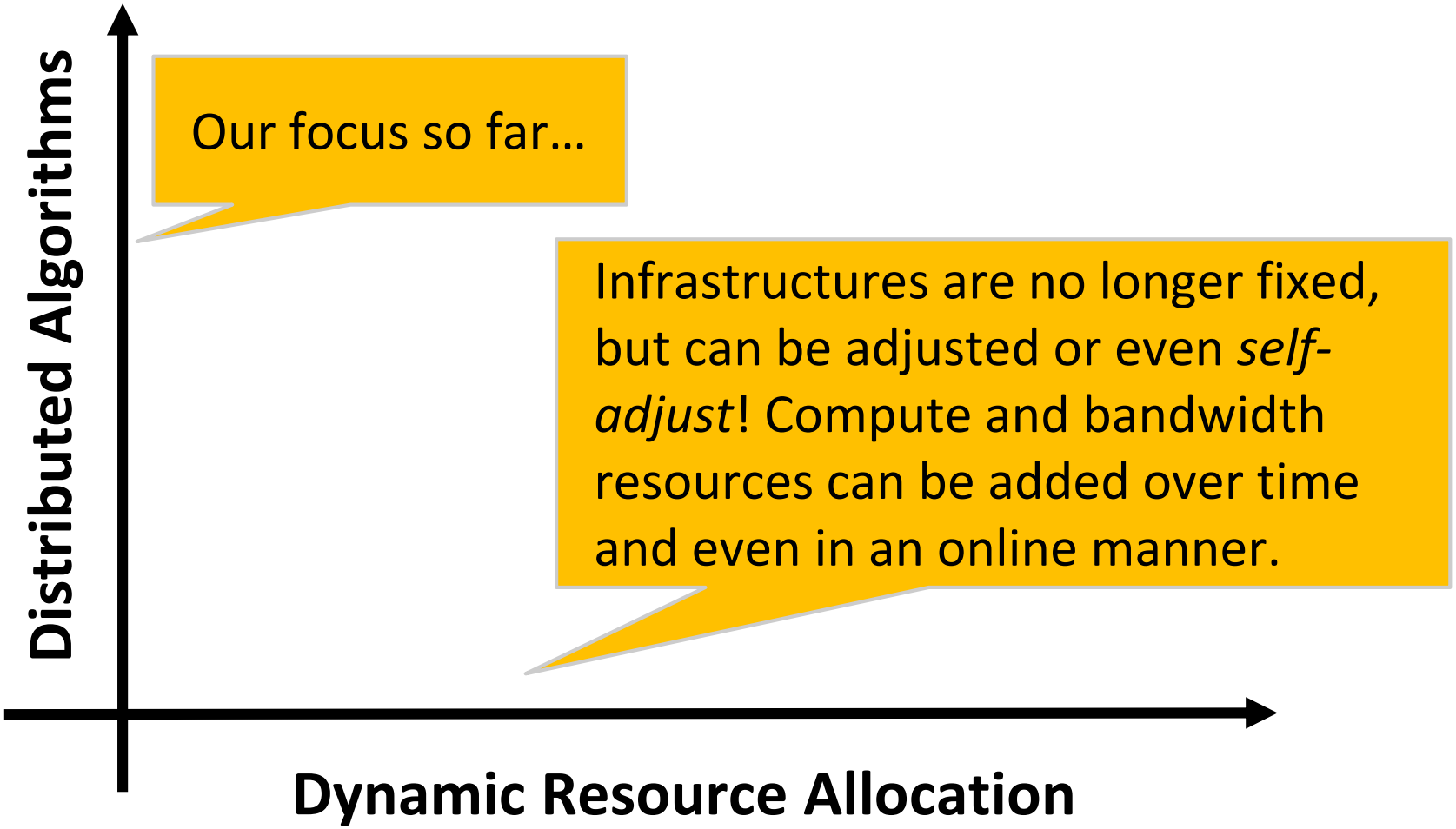
- Traditionally: astronomers and physicists
  - CERN LHC: 15 petabytes / year
- More recently: biologists and life scientists join big data club
  - Everything from '*why coastal algae bloom to what microbed dwell where in the human body*'

Business

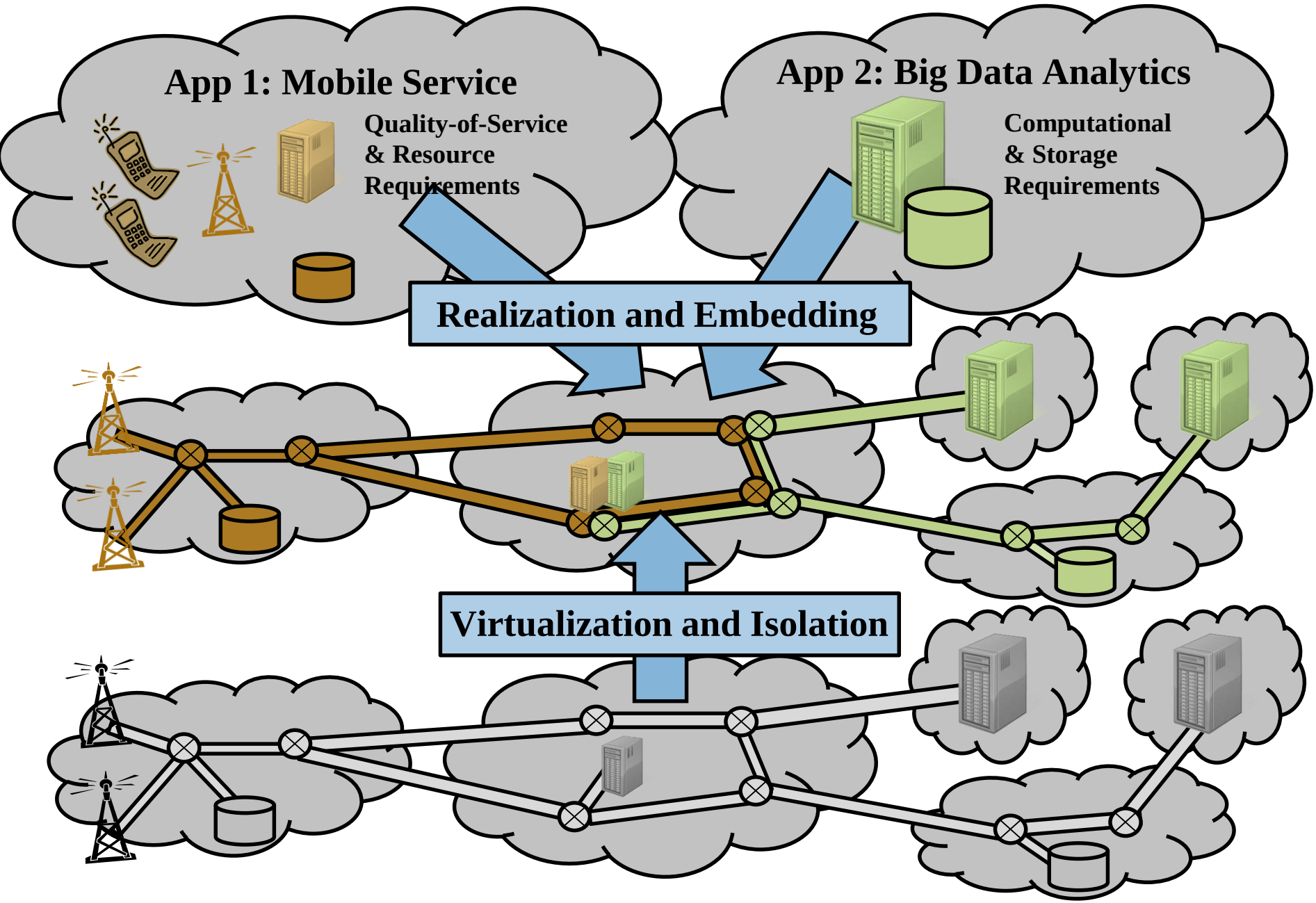
- Facebook, Twitter, LinkedIn
  - Operate pipelines of 100s terabytes / day
  - Logging user interactions, monitoring compute infrastructures, tracking business-critical functions
- More and more machine-driven data collection
- Advent of Internet-of-Things, smart devices, …

* D. Abady et al. The Beckman Report on Database Research. *Communication of the ACM*, February 2016

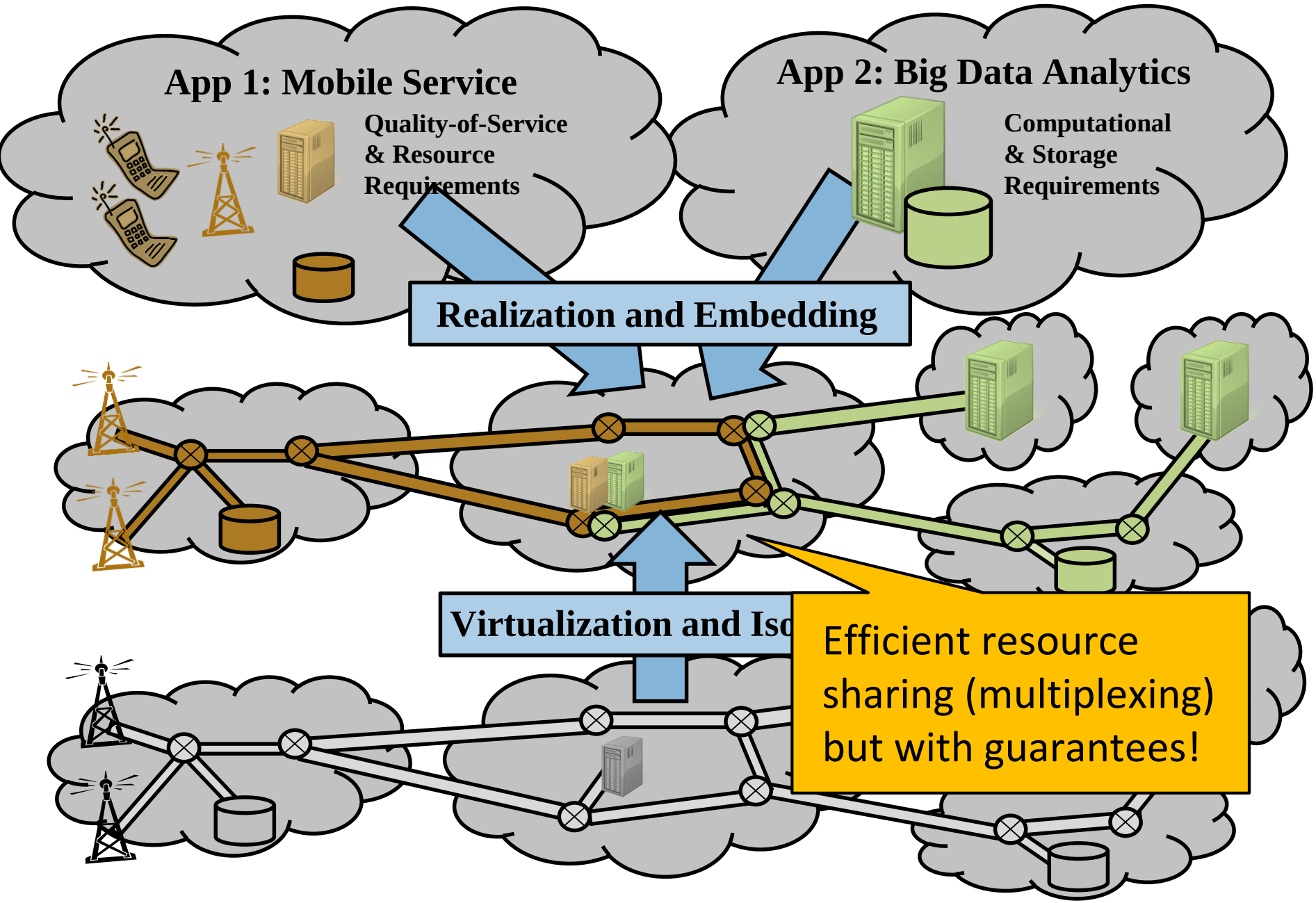
Challenges in Processing Big Data



Opportunity and Challenge: Virtualization



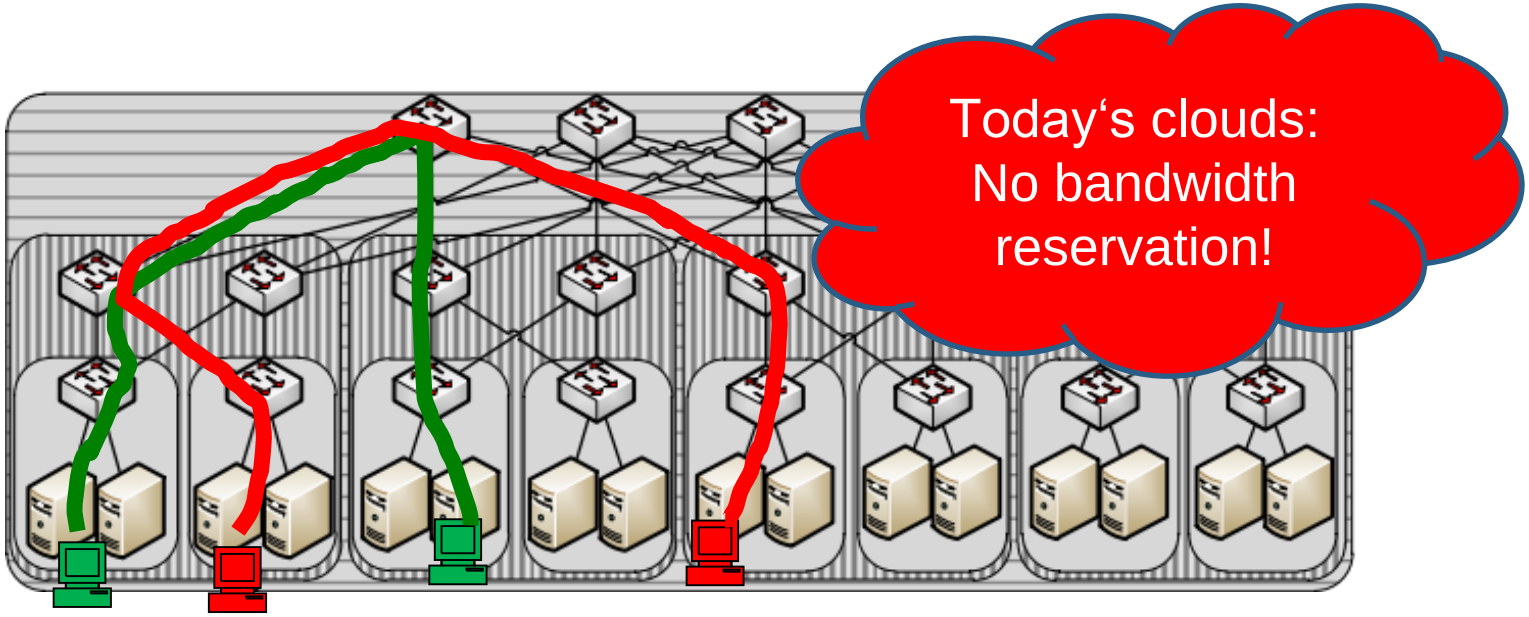
Opportunity and Challenge: Virtualization



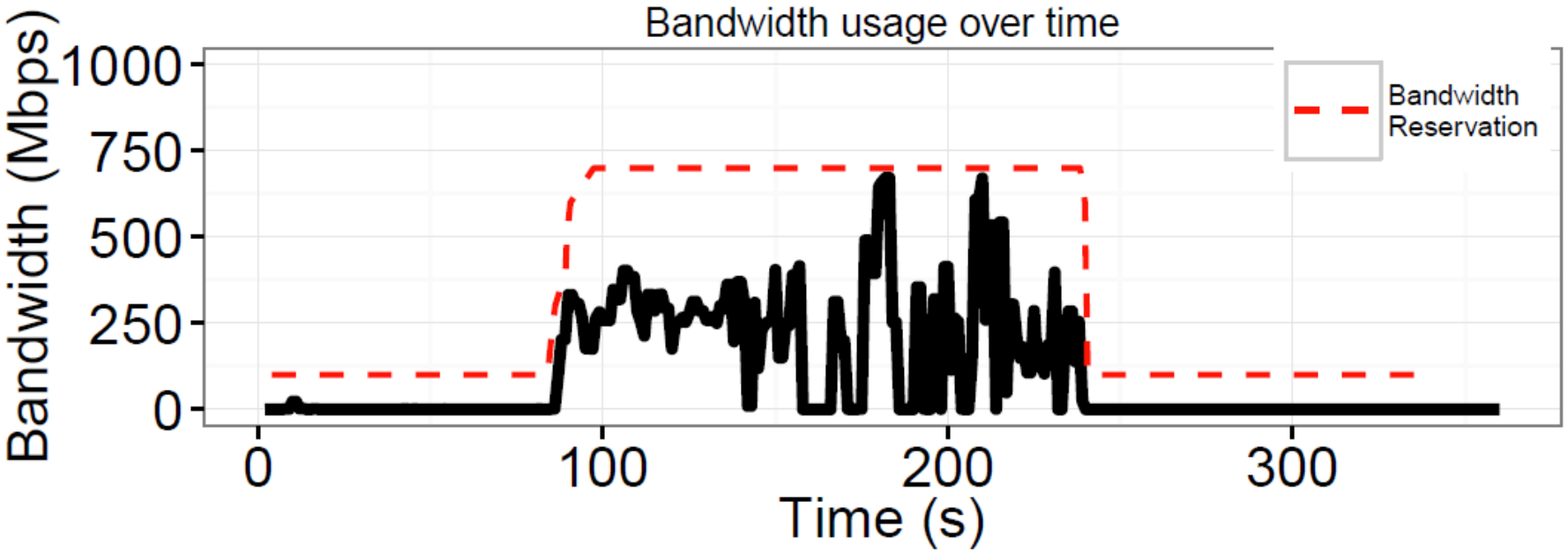
Big Data Analytics + Networking?

- Scale-out databases, batch processing applications etc.: significant network traffic
  - Example Facebook study: 33% of **execution time** due to communication

- Therefore: predictable performance requires performance isolation and bandwidth reservations



Example: Bandwidth Requirements in Hadoop

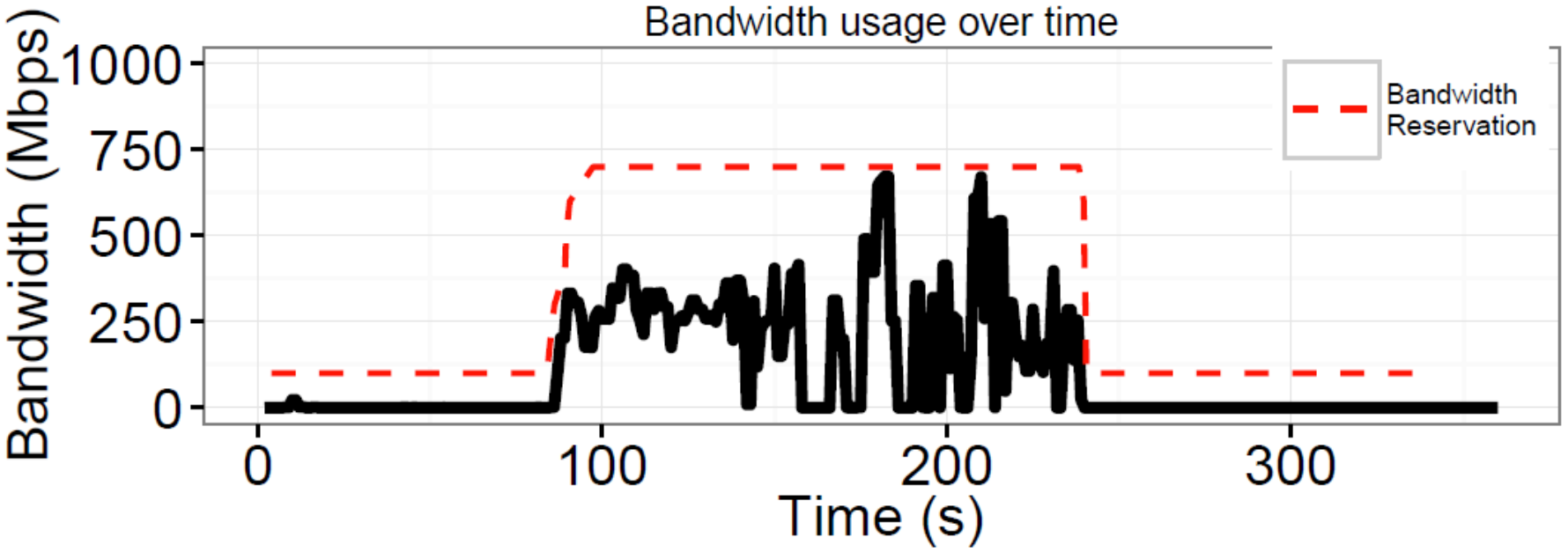


Bandwidth utilization of a TeraSort job over time.

In red: desired bandwidth reservation

- ❑ Predictable performance requires reservations!
- ❑ But how to minimize reservations? And when?

Example: Bandwidth Requirements in Hadoop



A virtual network embedding problem: minimize reserved bandwidth = path lengths.

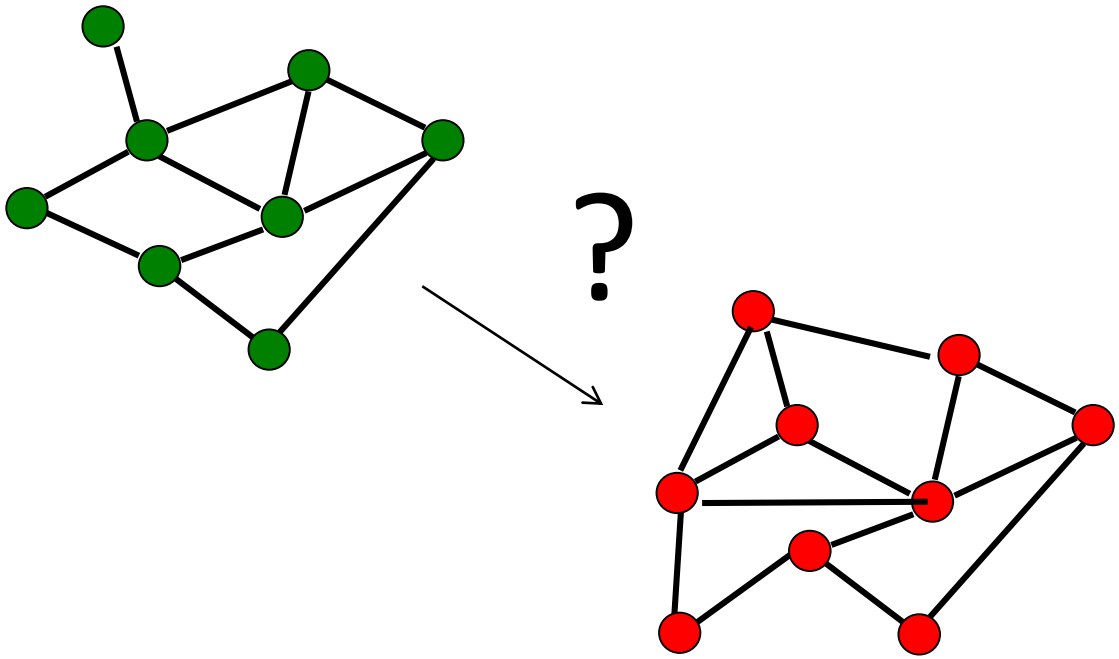
time.

- ❑ Predictable performance requires reservations!
- ❑ But how to minimize reservations? And when?

Predictable Performance: An Algorithmic Challenge

How to exploit VM placement flexibilities?

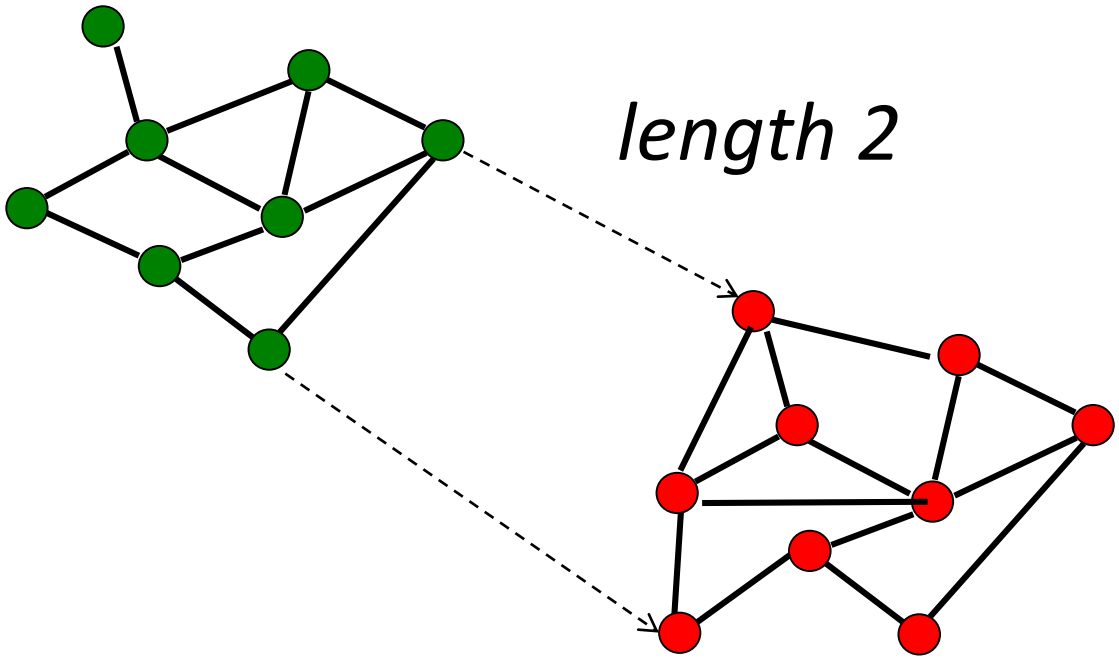
- Goal: Map communicating VMs close to each other
- I.e., minimize the sum of the logical link lengths



Predictable Performance: An Algorithmic Challenge

How to exploit VM placement flexibilities?

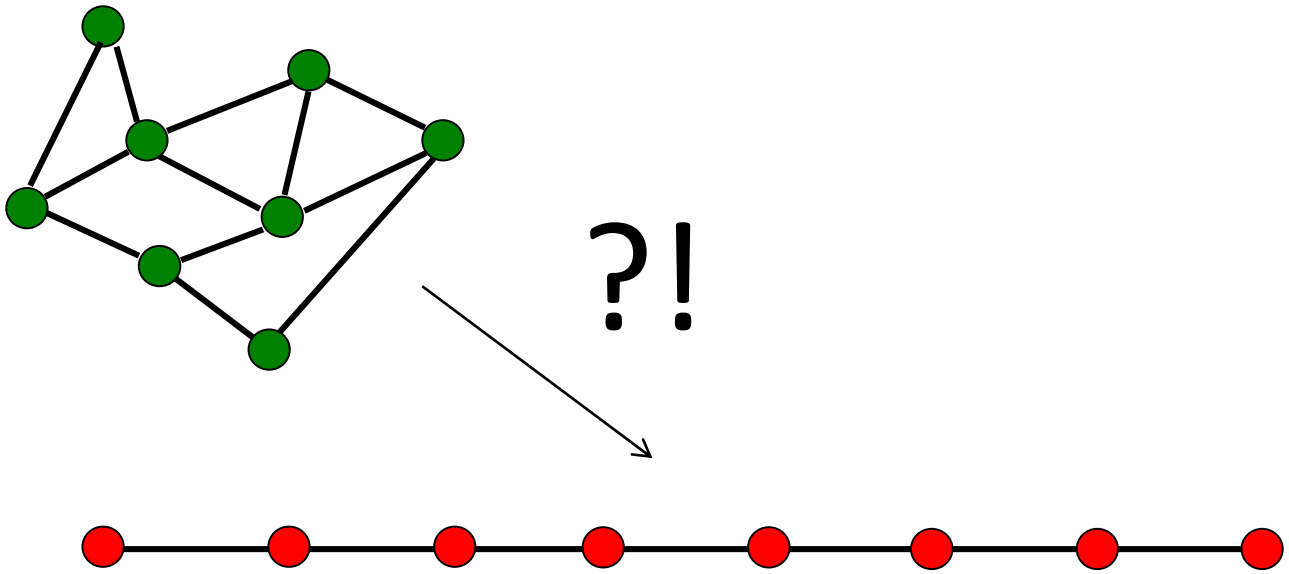
- Goal: Map communicating VMs close to each other
- I.e., minimize the sum of the logical link lengths



Predictable Performance: An Algorithmic Challenge

How to exploit VM placement flexibilities?

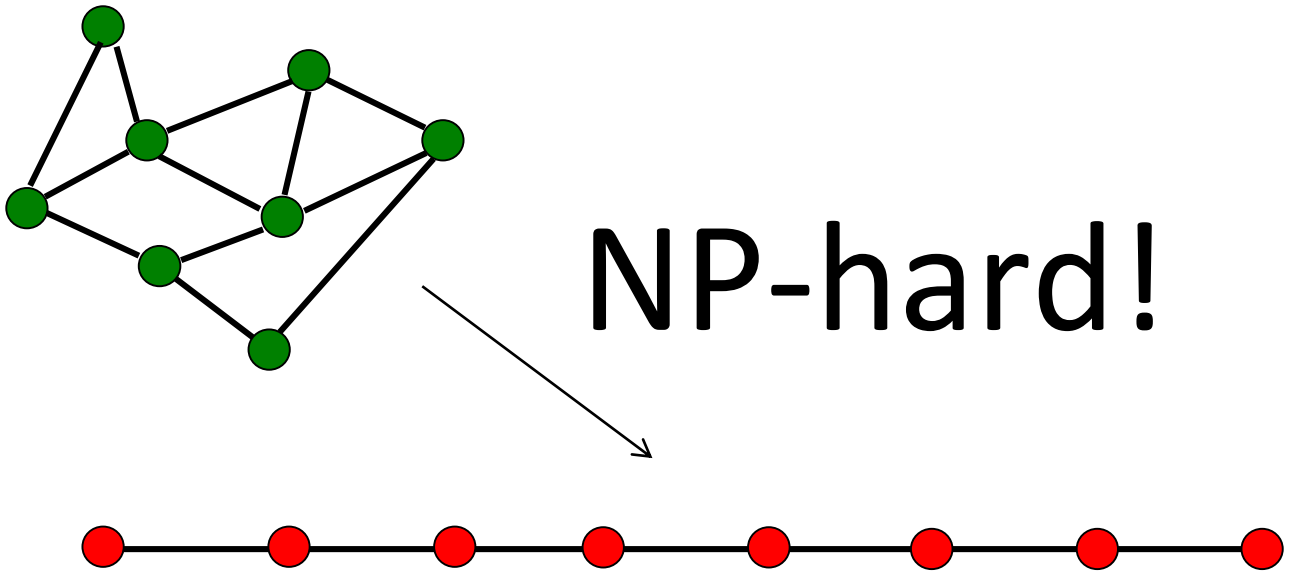
- Goal: Map communicating VMs close to each other
- I.e., minimize the sum of the logical link lengths



Predictable Performance: An Algorithmic Challenge

How to exploit VM placement flexibilities?

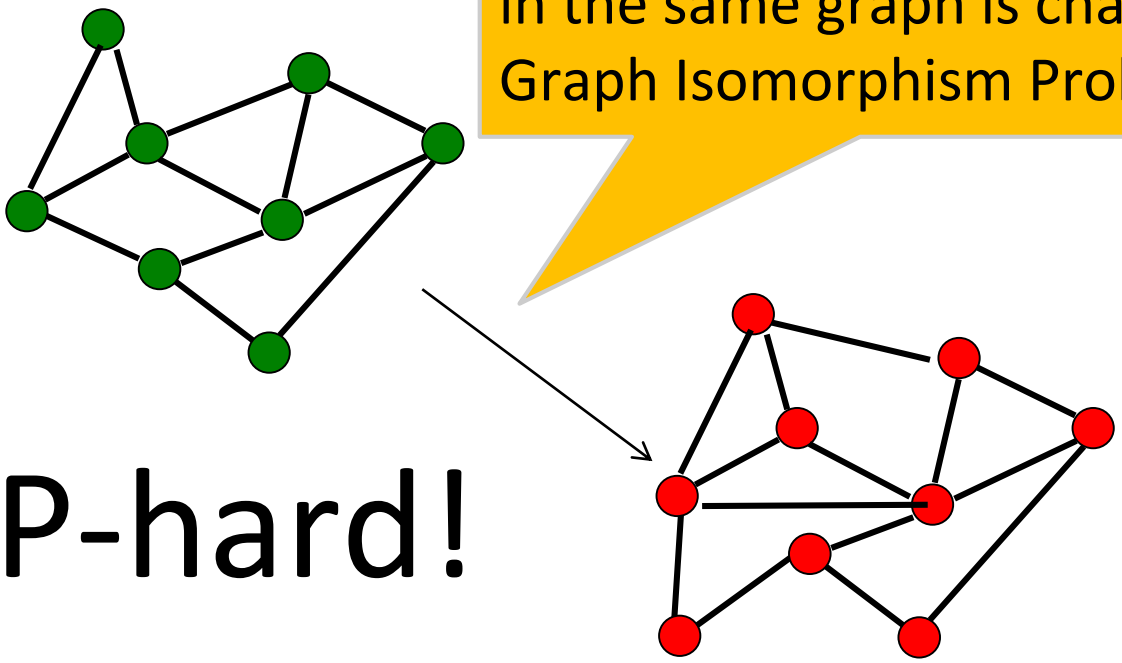
- Goal: Map communicating VMs close to each other
- I.e., minimize the sum of the logical link lengths
- NP-hard (min-max and avg): Minimum Linear Arrangement



Predictable Performance: An Algorithmic Challenge

How to exploit VM placement flexibilities?

- ❏ Goal: Map communicating VMs close to each other
- ❏ I.e., minimize the sum of the logical link lengths



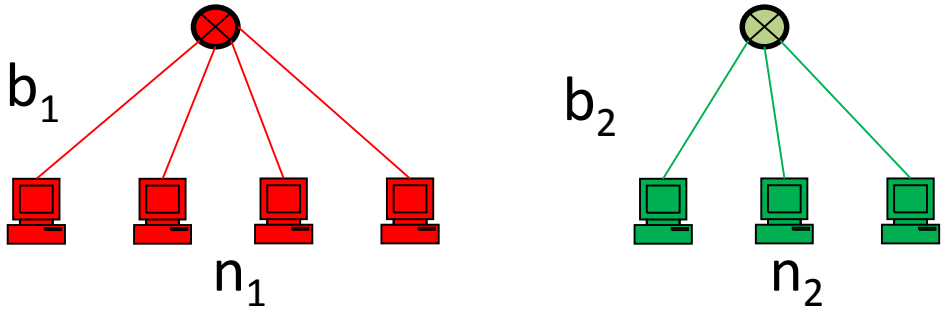

That's all Folks!

In practice, solutions are often possible!

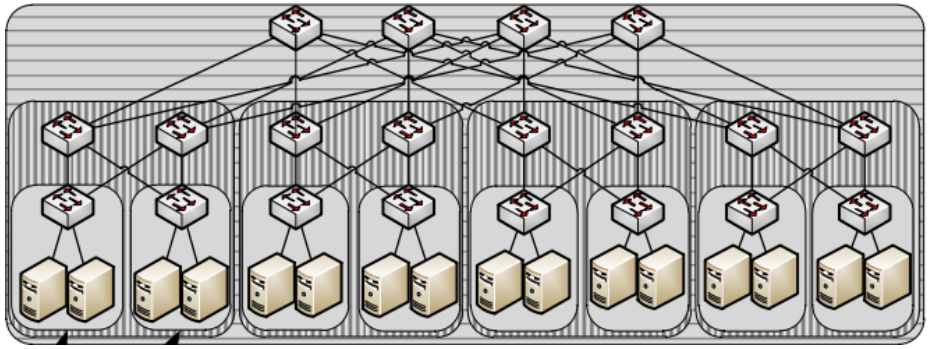
Virtual Clusters: Abstraction for Batch Processing

A frequently studied virtual network abstraction!

- Virtual Cluster *VC(n,b)*
  - Connects *n* virtual machines to a «logical» switch with bandwidth guarantees *b*
  - A simple abstraction

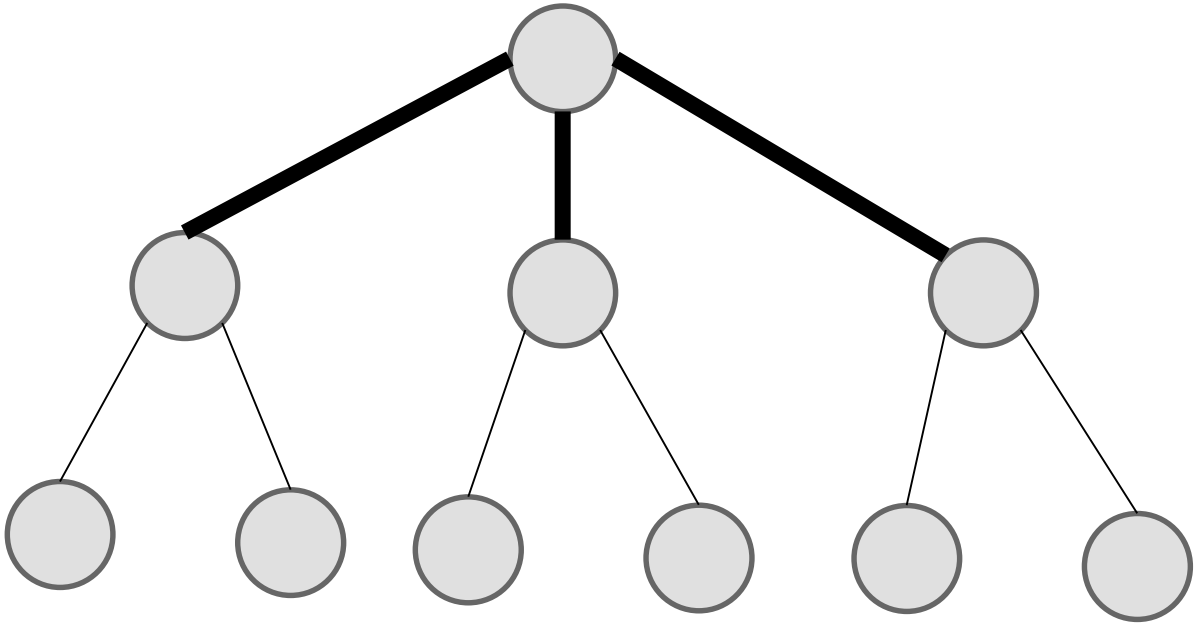


Recall: Clos Topology

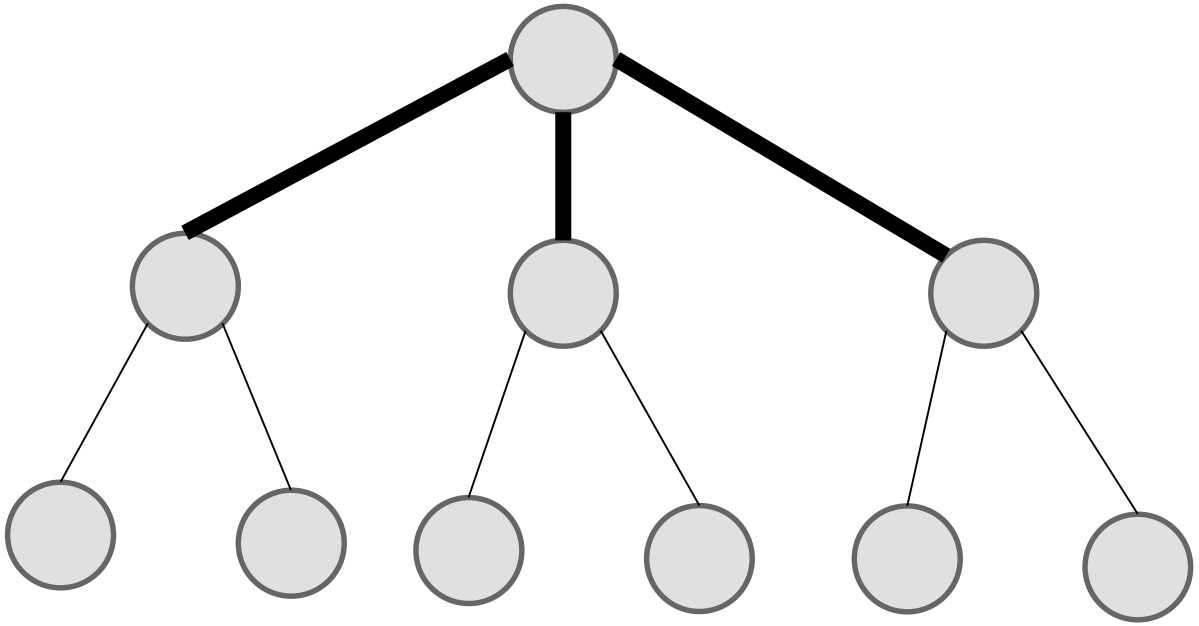


Actual datacenter topologies look like this!

And sometimes can even be abstracted as simple fat-trees (ECMP).

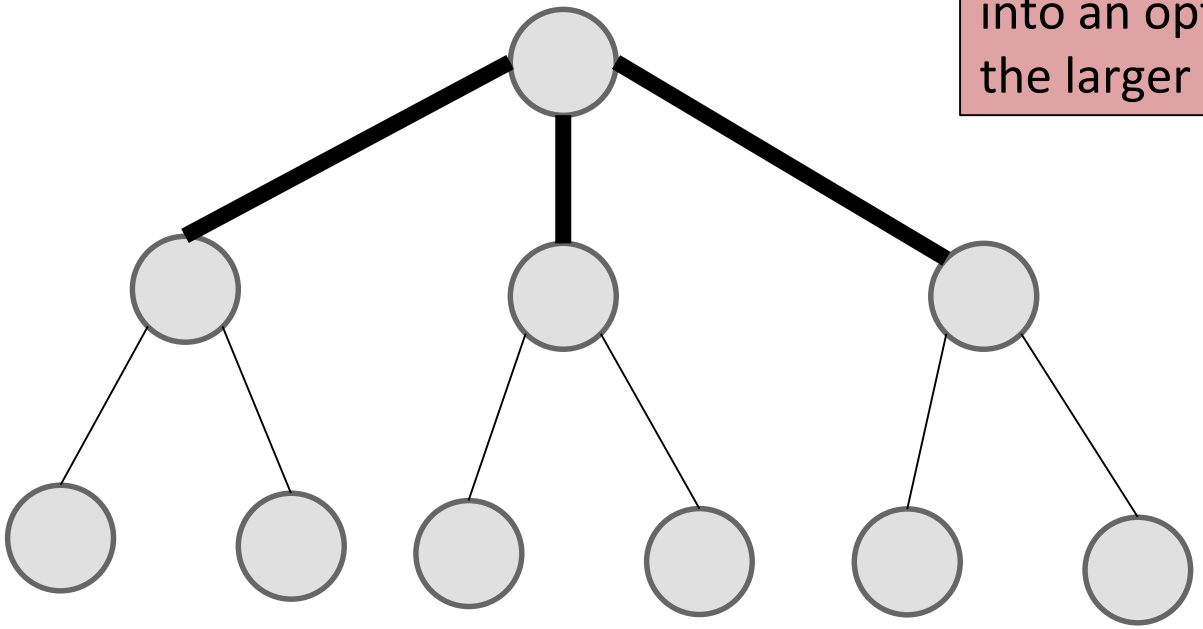


How to embed a Virtual Cluster in a Fat-Tree?



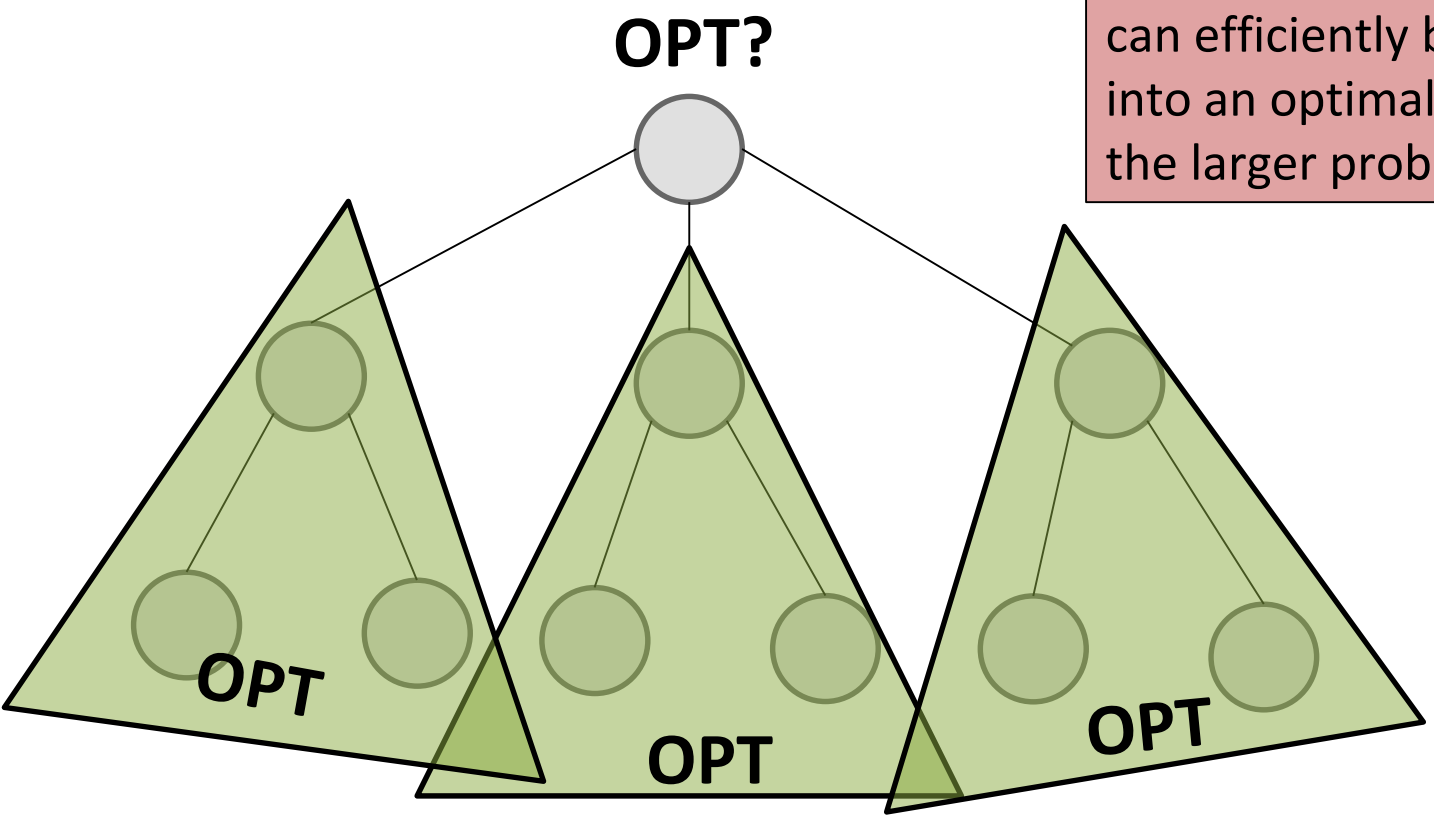
How to embed a Virtual Cluster in a Fat-Tree?

Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!



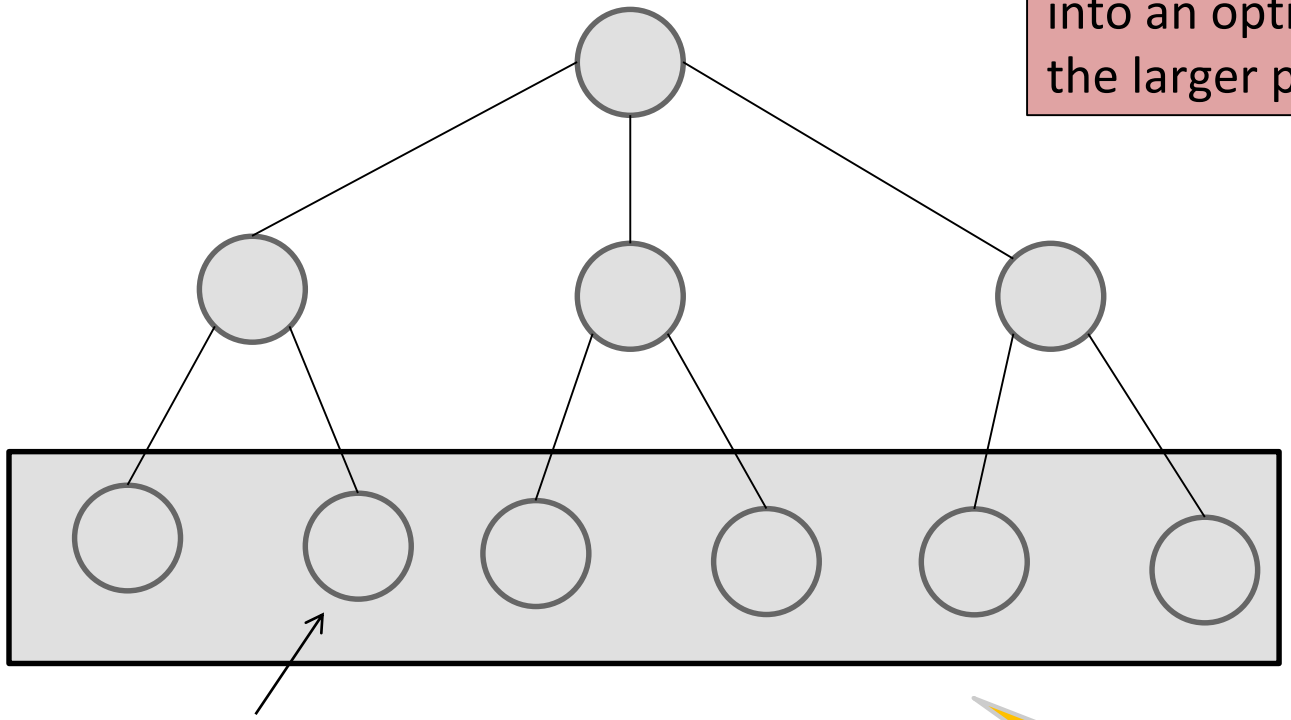
How to embed a Virtual Cluster in a Fat-Tree?

Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!



How to embed a Virtual Cluster in a Fat-Tree?

Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!



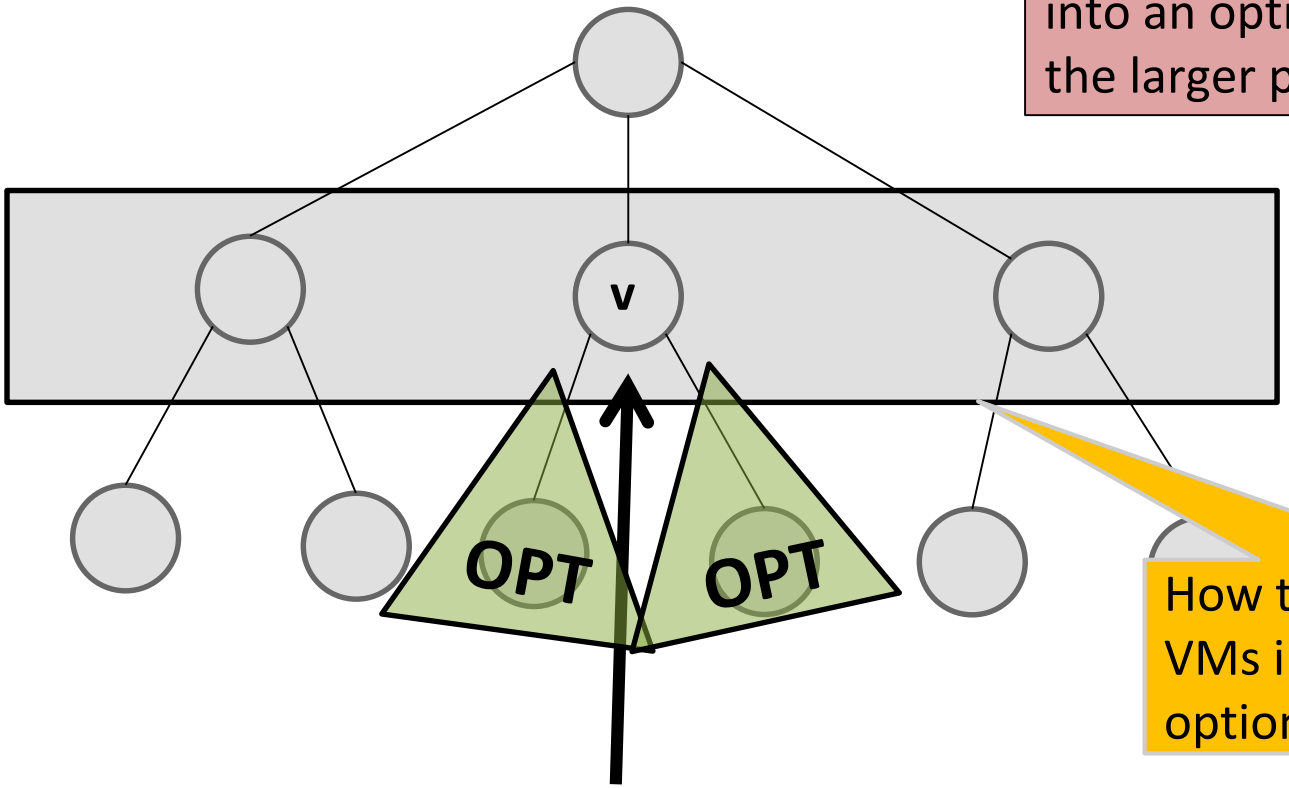
t = 0: solve leaves!

How to optimally embed x VMs here, $x \in \{0, ..., n\}$?
Easy: Cost = 0 or ∞!

Start from leaves: how to optimally embed x VMs in this substree? Trivial.

How to embed a Virtual Cluster in a Fat-Tree?

Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!



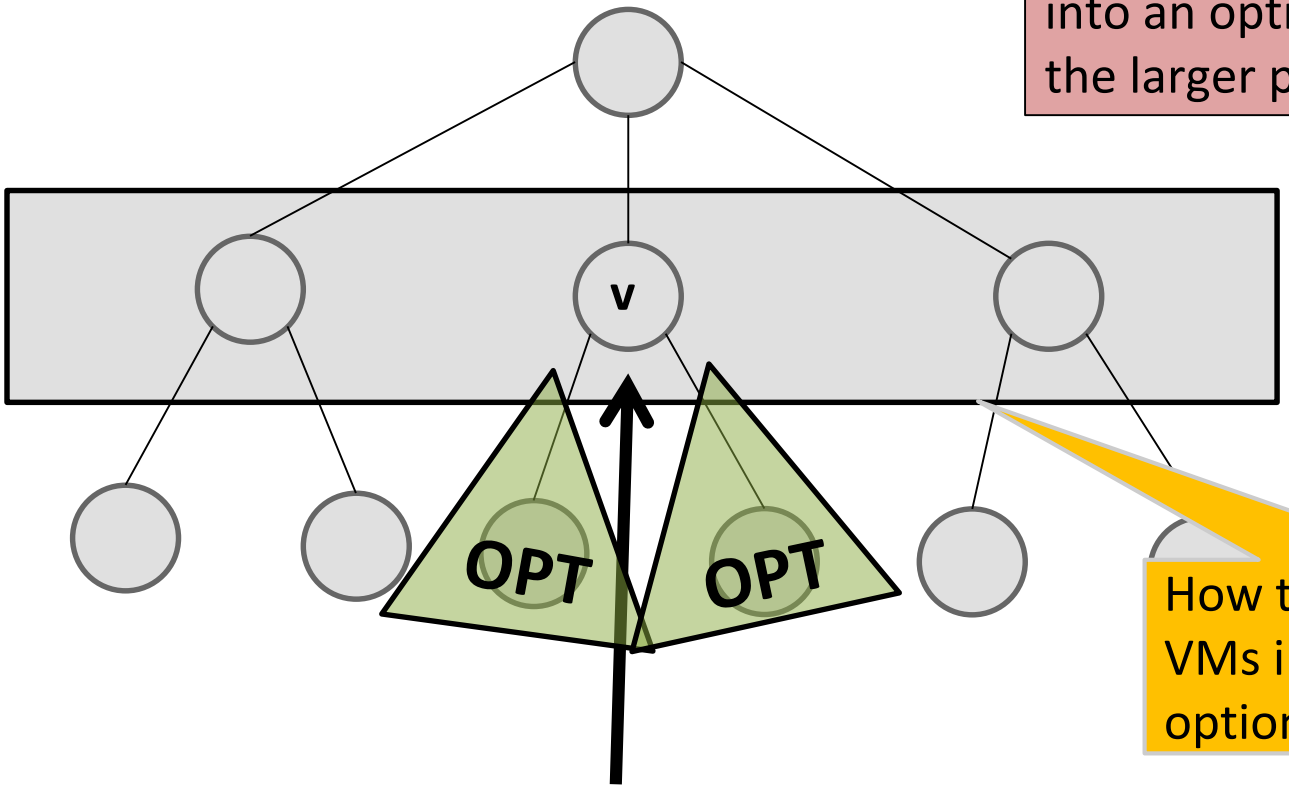
t = 1: solve height 1!

How to optimally embed x VMs in this substree? Try all options: y left, x-y right!

$$Cost[x] = \min_y Cost[y] + Cost[x-y] + \text{connections to } v$$

How to embed a Virtual Cluster in a Fat-Tree?

Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!



t = 1: solve height 1!

How to optimally embed x VMs in this substree? Try all options: y left, x-y right!

$$Cost[x] = \min_y Cost[y] + Cost[x-y] + \text{connections to } v$$

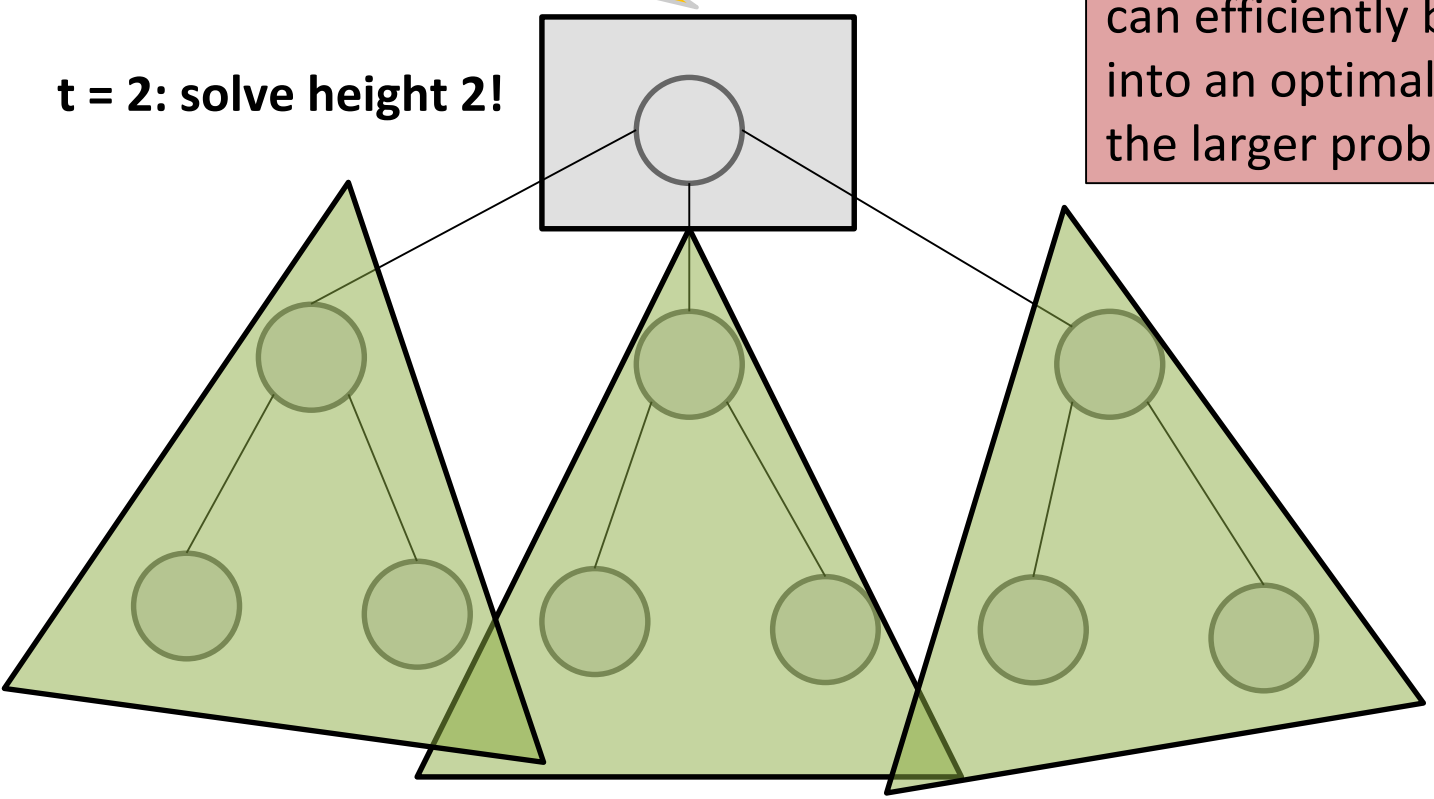
Or just account on upward link (number of leaving links!)

How to embed a Virtual Cluster in a Fat-Tree?

Just need to decide the best split of VMs and sum up optimal costs from subtrees!

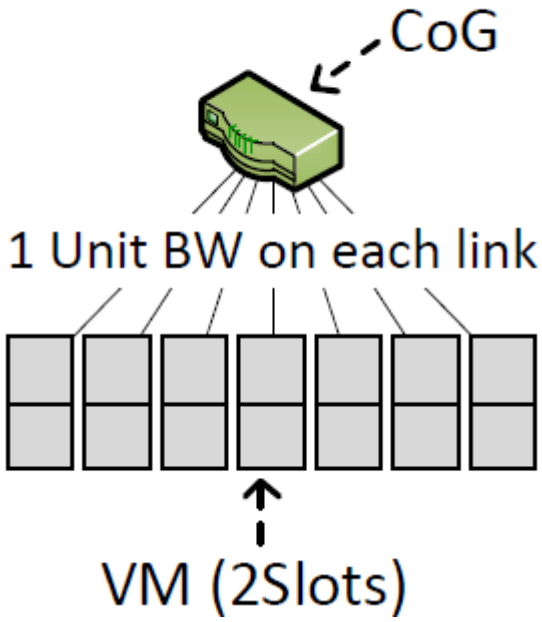
Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!

t = 2: solve height 2!

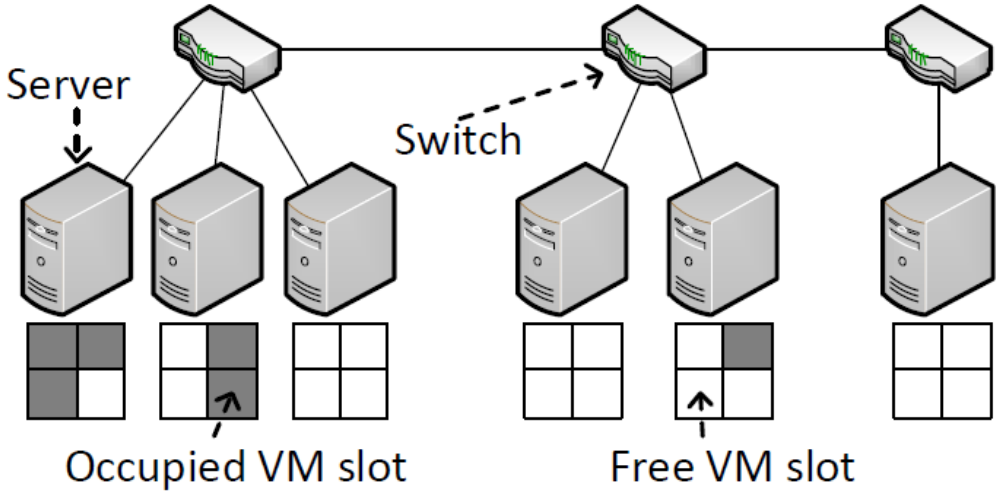


How to embed a Virtual Cluster in a General Graph?

How to embed?



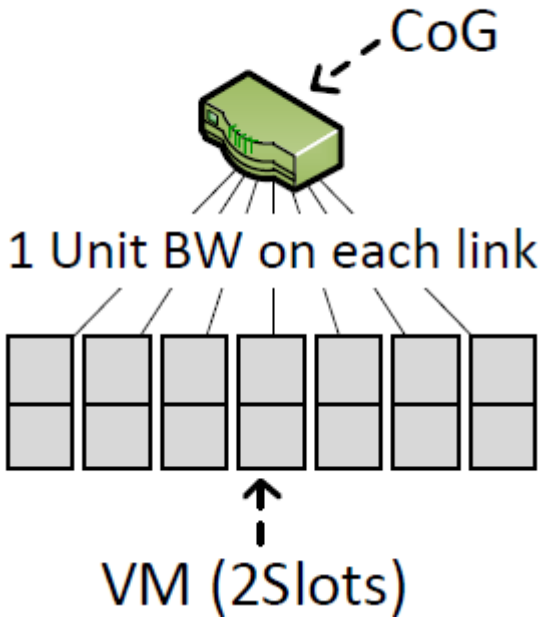
Guest Graph



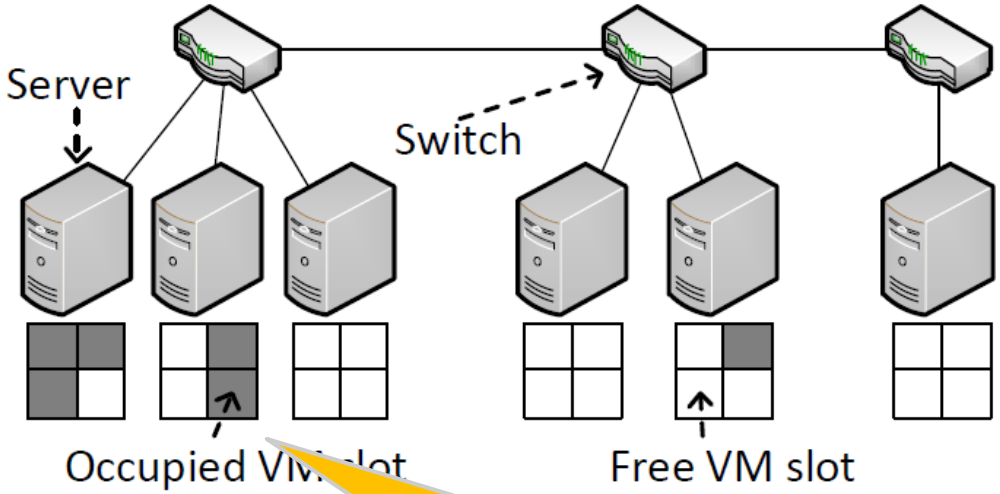
Host Graph

How to embed a Virtual Cluster in a General Graph?

How to embed?



Guest Graph

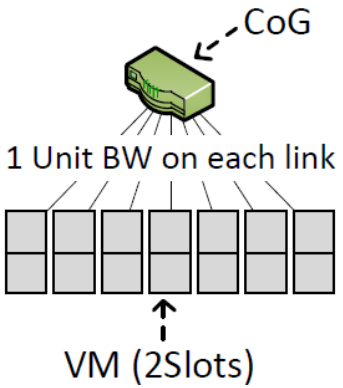


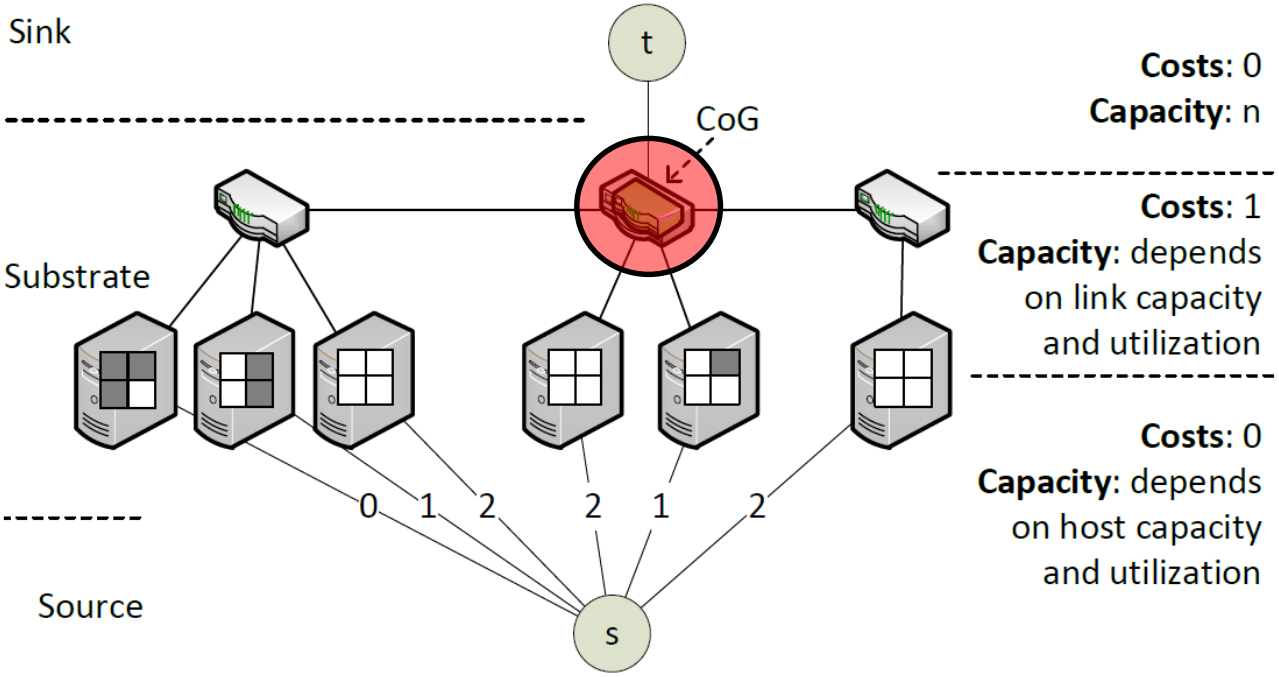
Idea: Can reduce the problem to a flow problem!

How to embed a Virtual Cluster in a General Graph?

Algorithm:

- Try all possible locations for virtual switch (CoG)
- Extend network with artificial source s and sink t
- Add capacities
- Compute min-cost max-flow from s to t
  (or simply: min-cost flow of volume n)

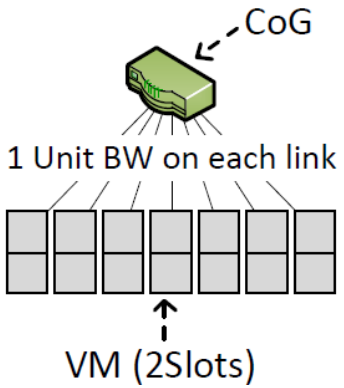


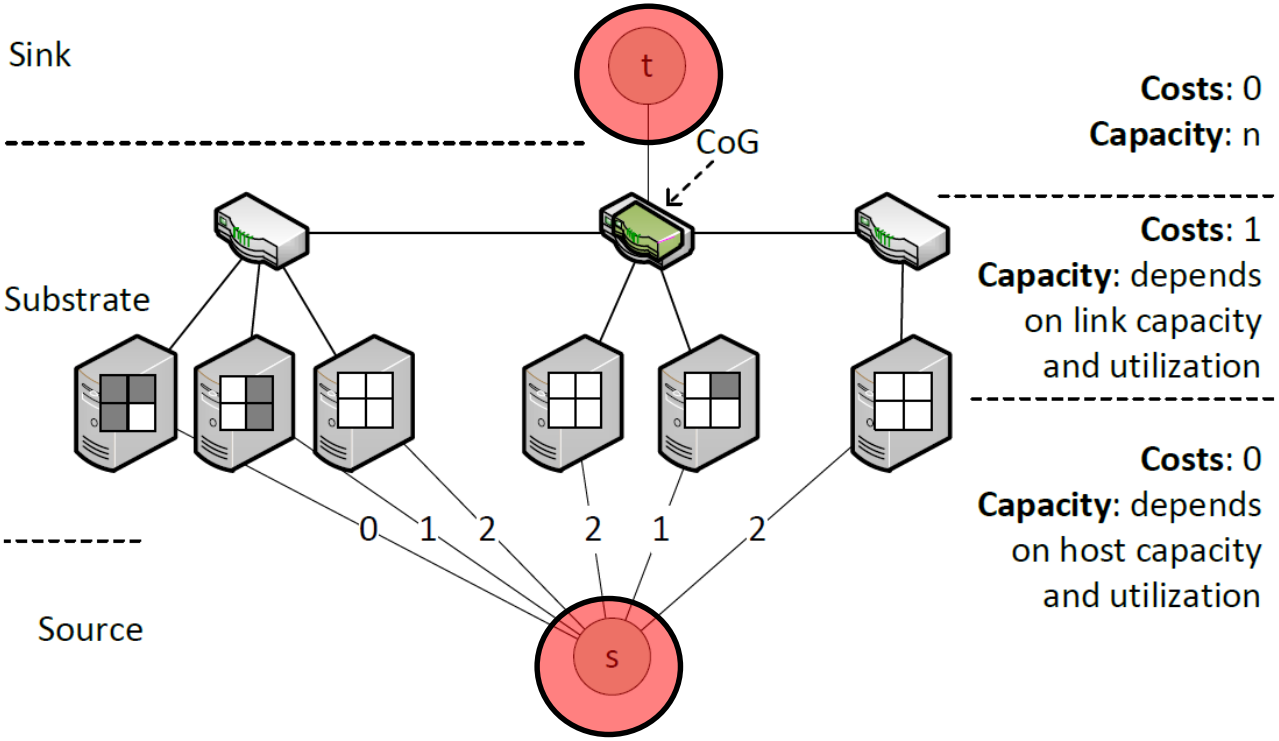


How to embed a Virtual Cluster in a General Graph?

Algorithm:

- Try all possible locations for virtual switch
- Extend physical network with artificial source s and sink t
- Add capacities
- Compute min-cost max-flow from s to t
  (or simply: min-cost flow of volume n)

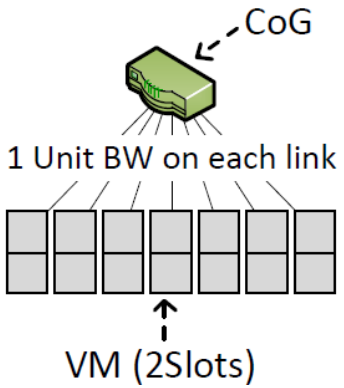


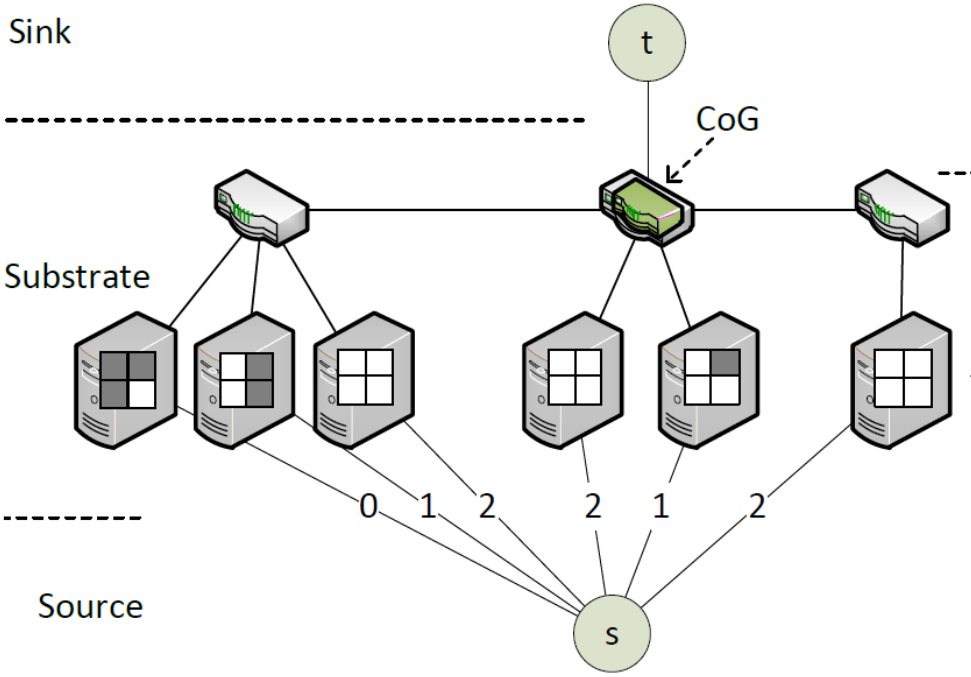


How to embed a Virtual Cluster in a General Graph?

Algorithm:

- Try all possible locations for virtual switch
- Extend physical network with artificial source s and sink t
- Add capacities
- Compute min-cost max-flow from s to t
  (or simply: min-cost flow of volume n)





Costs: 0
Capacity: n

enough to embed n VMs

Costs: 1
Capacity: depends on link capacity and utilization

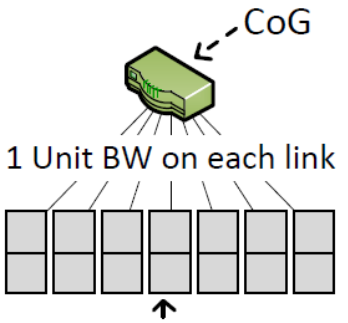
Costs: 0
Capacity: depends on host capacity and utilization

capacity = floor(available resources / unit demand)

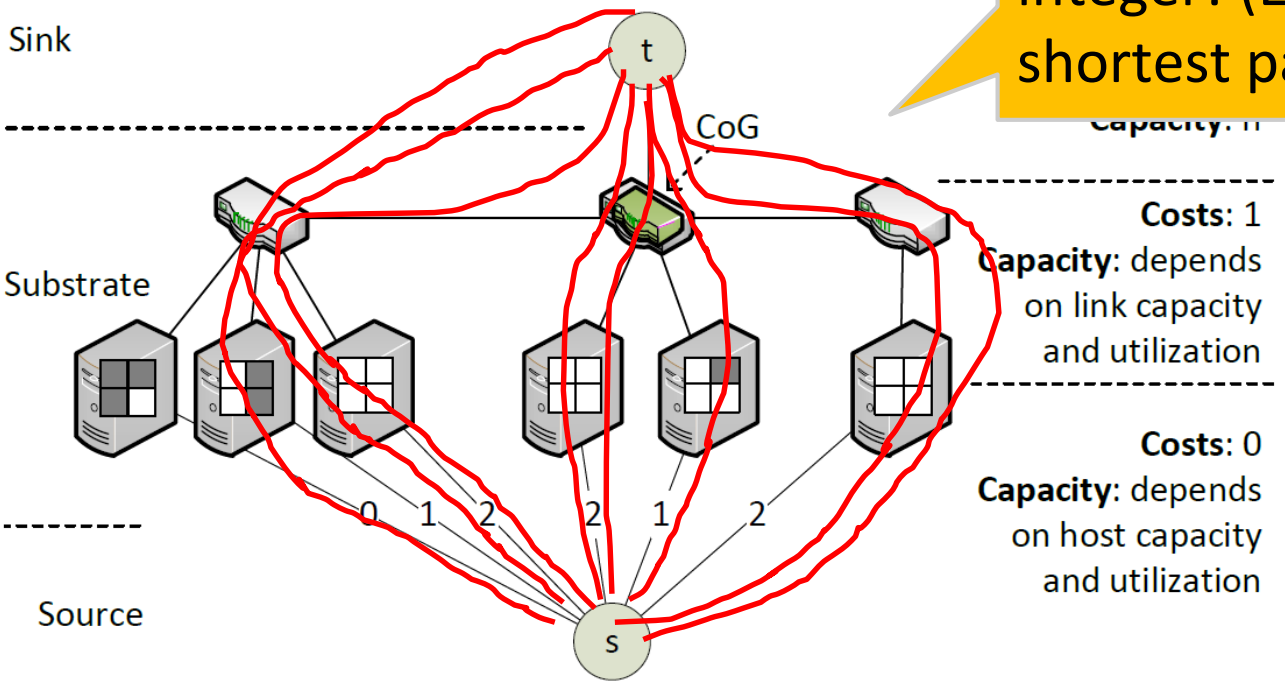
How to embed a Virtual Cluster in a General Graph?

Algorithm:

- Try all possible locations for virtual switch
- Extend network with artificial source s and sink t
- Add capacities
- Compute min-cost max-flow from s to t
  (or simply: min-cost flow of volume n)



Flows integer if links are integer! (E.g., successive shortest paths algorithm.)



Rigorous Solutions for the General Embedding Problem: MIP

Recipe for VNEP formulation :

- ❑ A (linear) objective function (e.g., load or footprint)
- ❑ A set of (linear) constraints
- ❑ Feed it to your favorite solver (CPLEX, Gurobi, etc.)

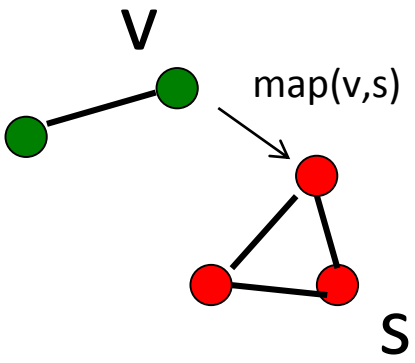
Rigorous Solutions for the General Embedding Problem: MIP

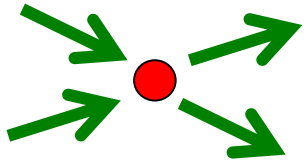
Recipe for VNEP formulation :

- ❑ A (linear) objective function (e.g., load or footprint)
- ❑ A set of (linear) constraints
- ❑ Feed it to your favorite solver (CPLEX, Gurobi, etc.)

Details:

- ❑ Introduce binary variables ***map(v,s)*** to map virtual nodes *v* on substrate node *s*
- ❑ Introduce flow variables for paths (splittable or not?)
- ❑ Ensure **flow conservation**: all flow entering a node must leave the node, unless it is the source or the destination





Mixed Integer Programs (MIPs)

- MIPs can be quite fast

- However, that's not the end of the story:

$$\text{MIP} \neq \text{MIP}$$

Mixed Integer Programs (MIPs)

- MIPs can be quite fast

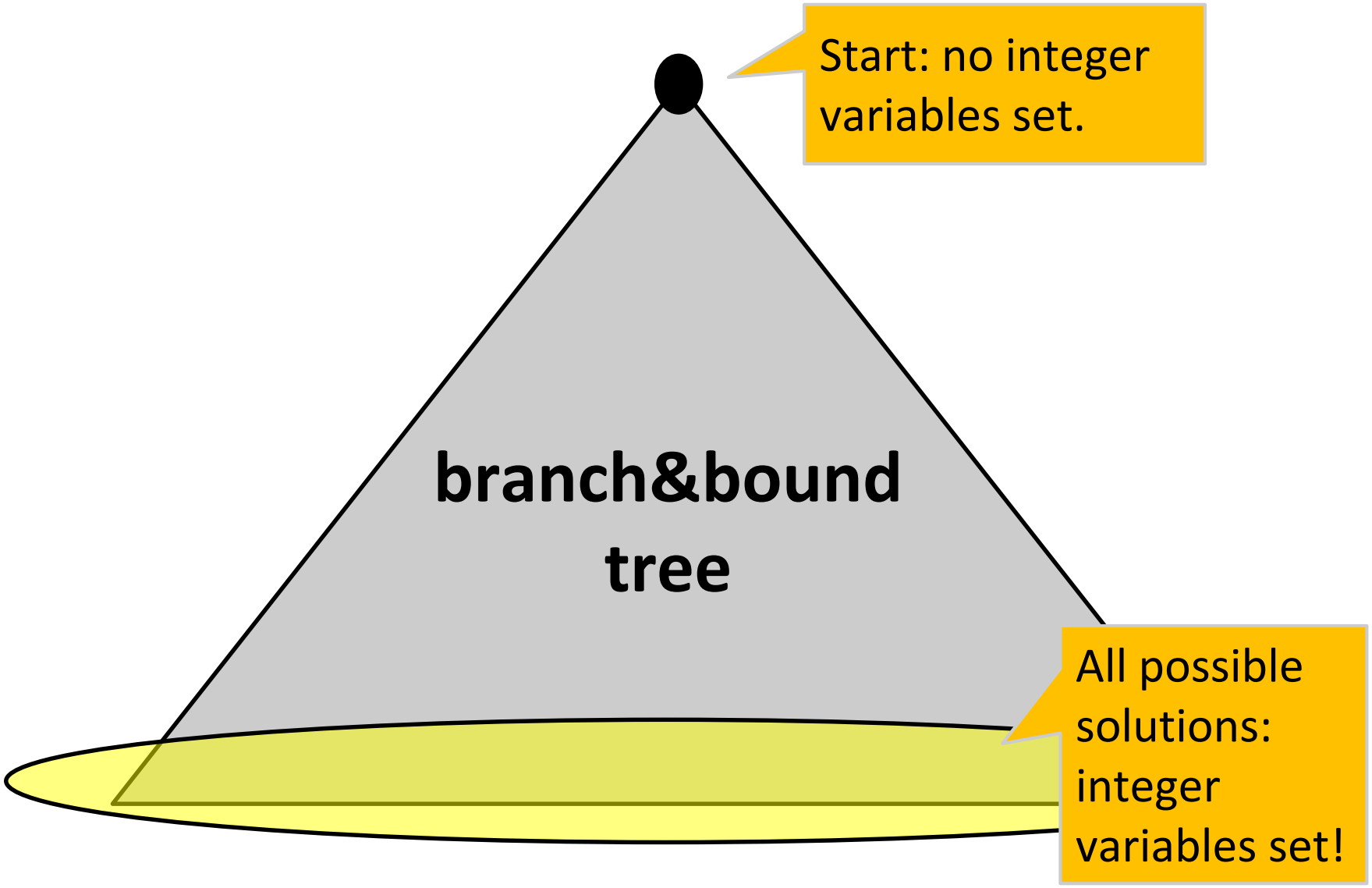
Maybe for pure integer programs, SAT solvers faster?

- However, that's not the end of the story:

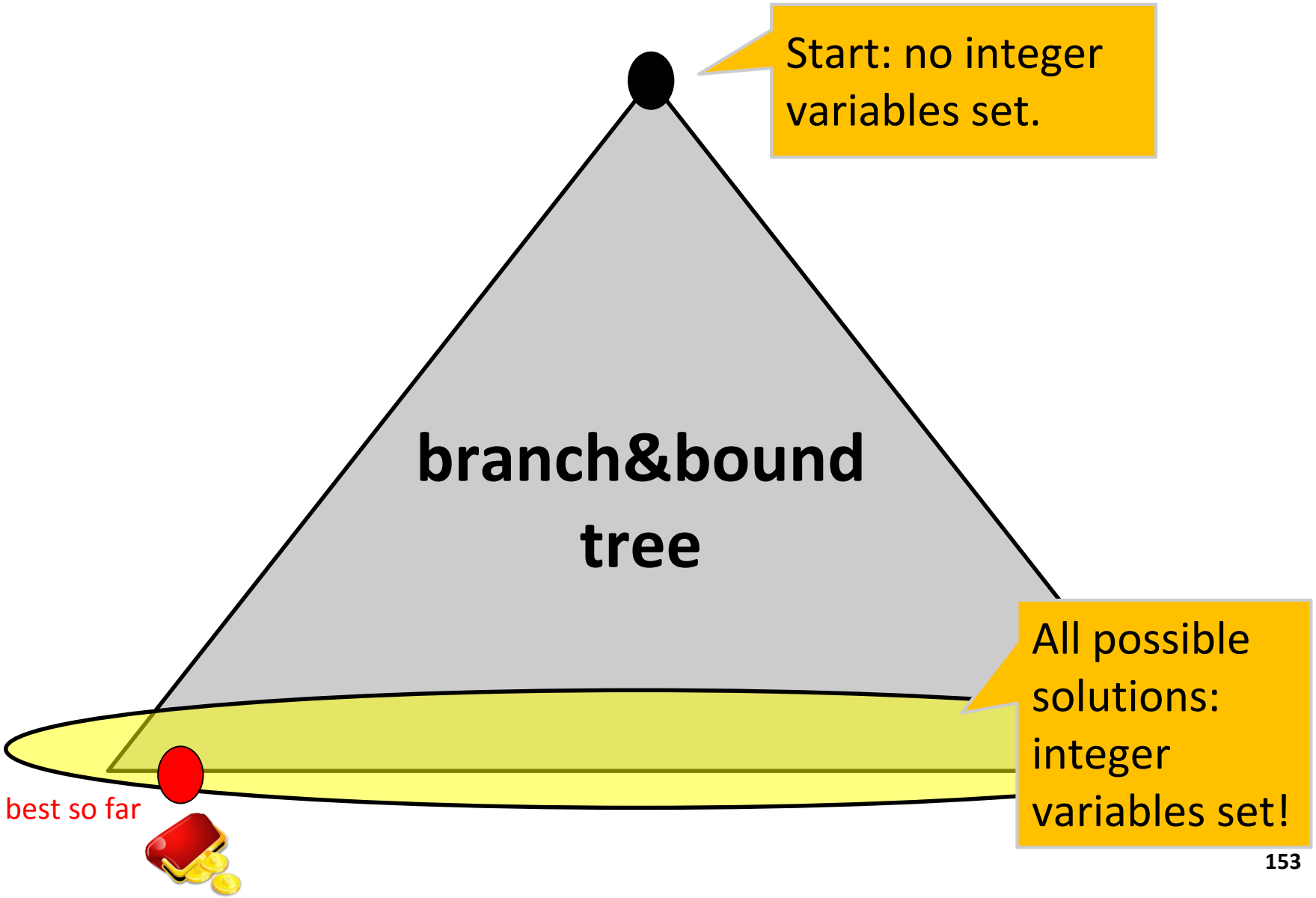
$$\text{MIP} \neq \text{MIP}$$

Many solvers (Gurobi, CPLEX) relay on branch-and-bound framework: relaxations!

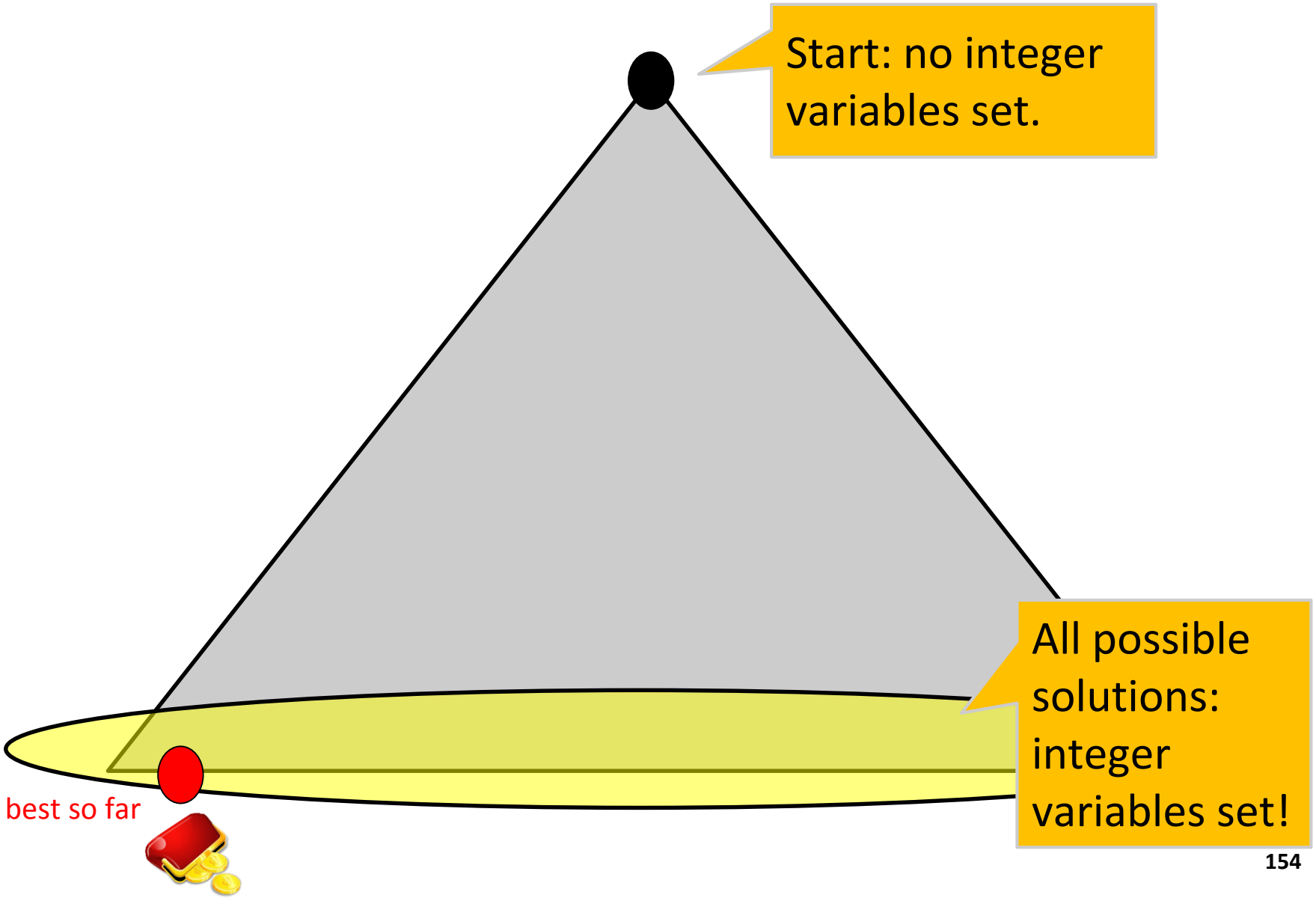
Branch-and-Bound Framework



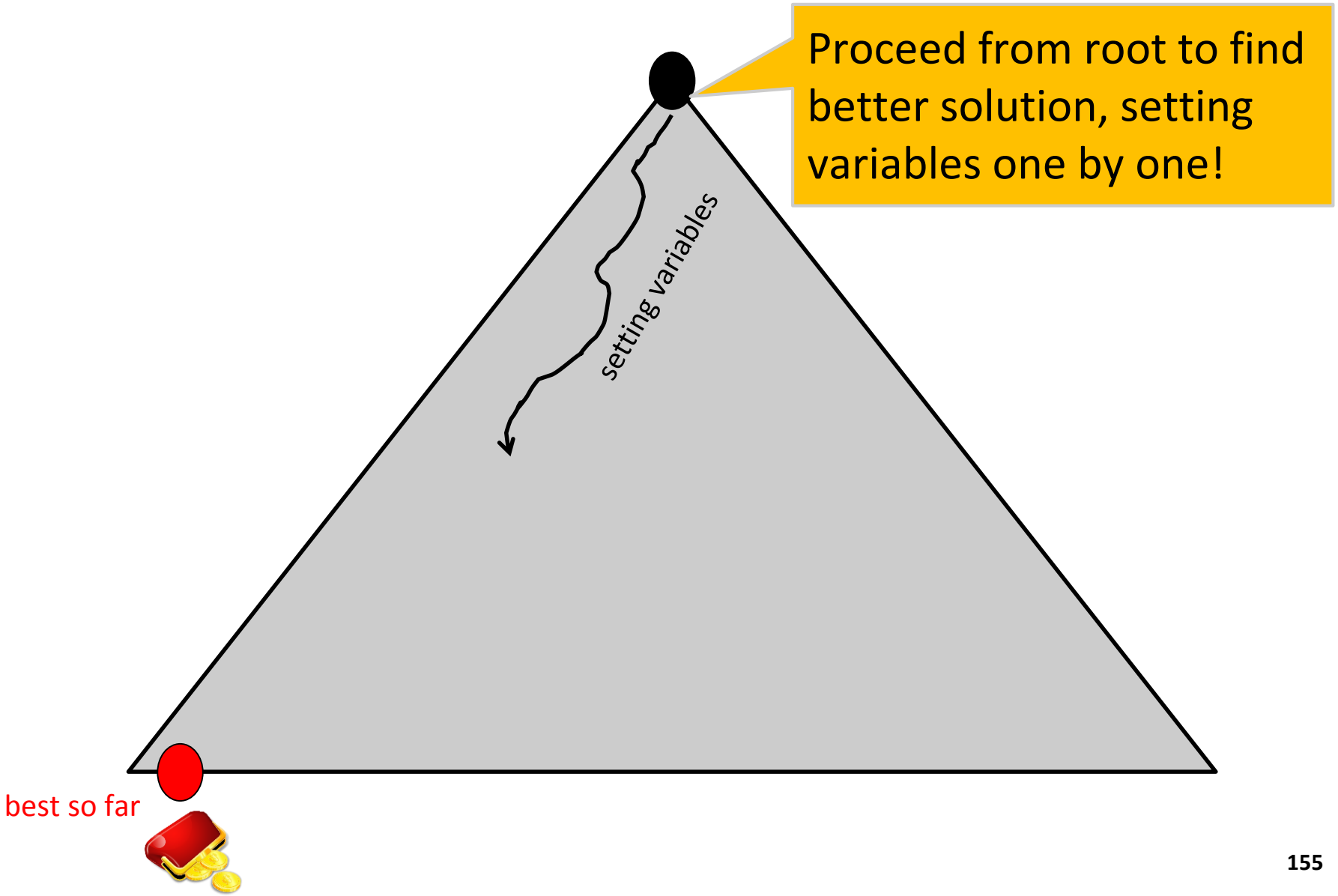
Branch-and-Bound Framework



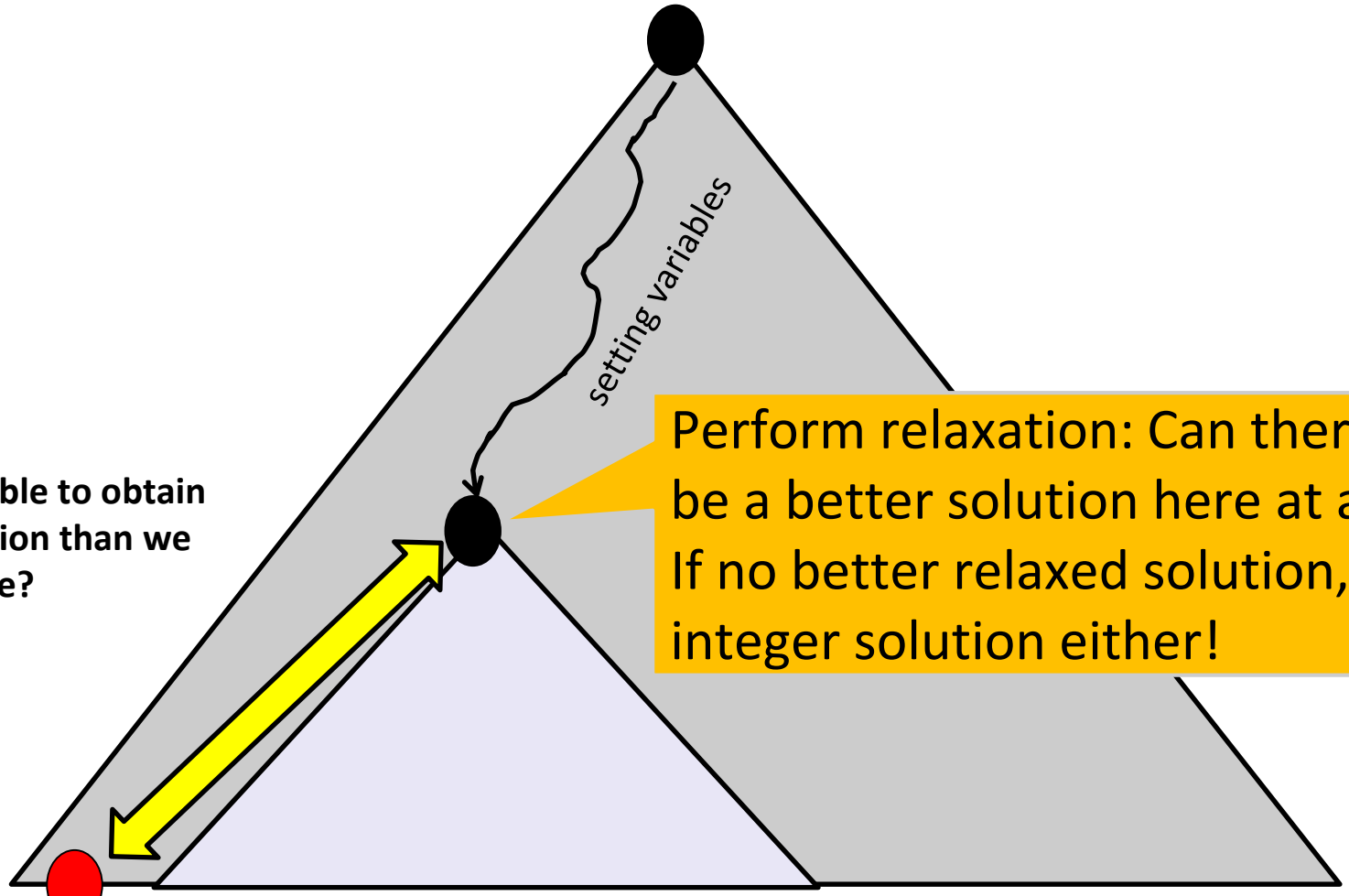
Branch-and-Bound Framework



Branch-and-Bound Framework



Branch-and-Bound Framework

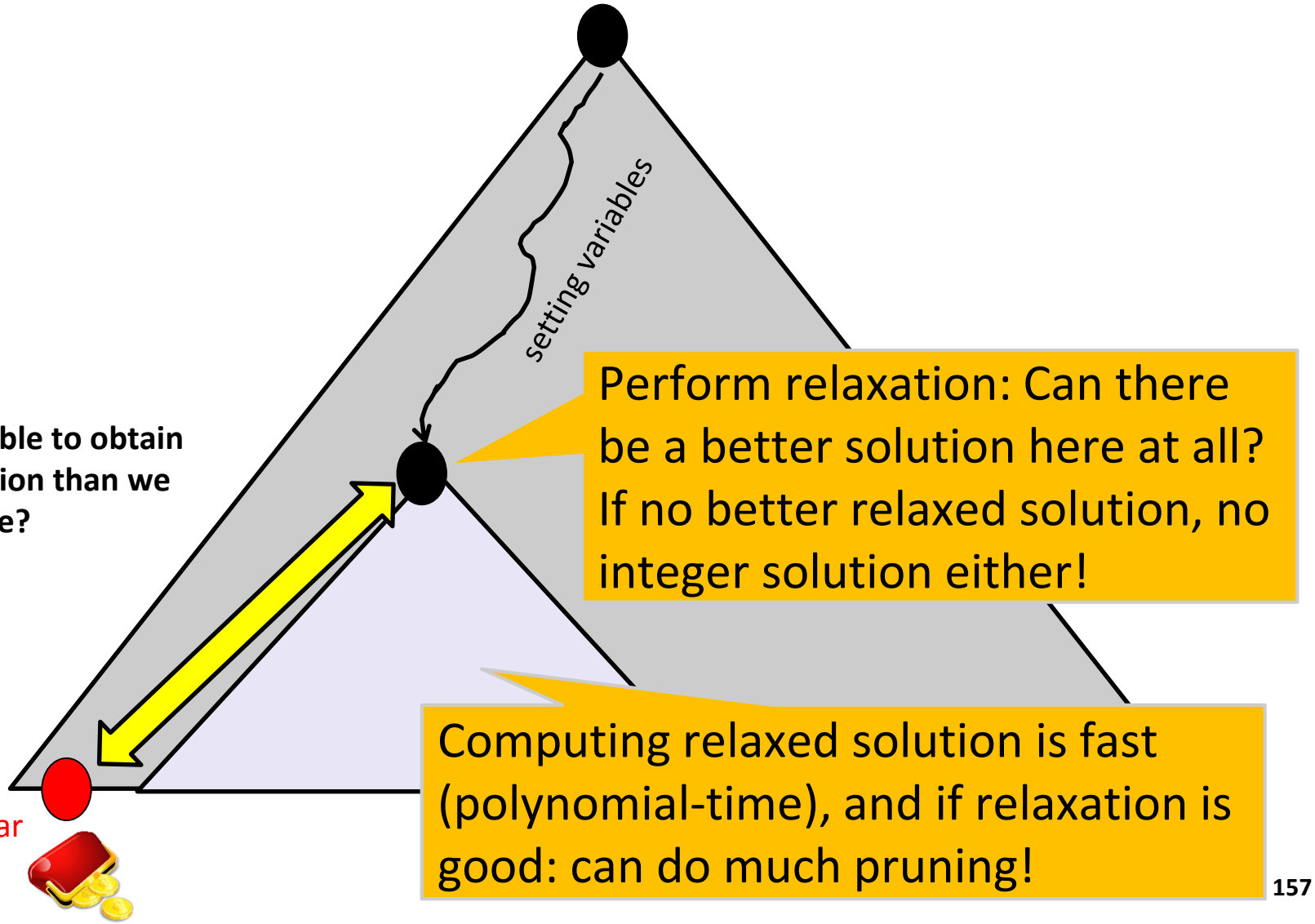


Relax: possible to obtain better solution than we already have?

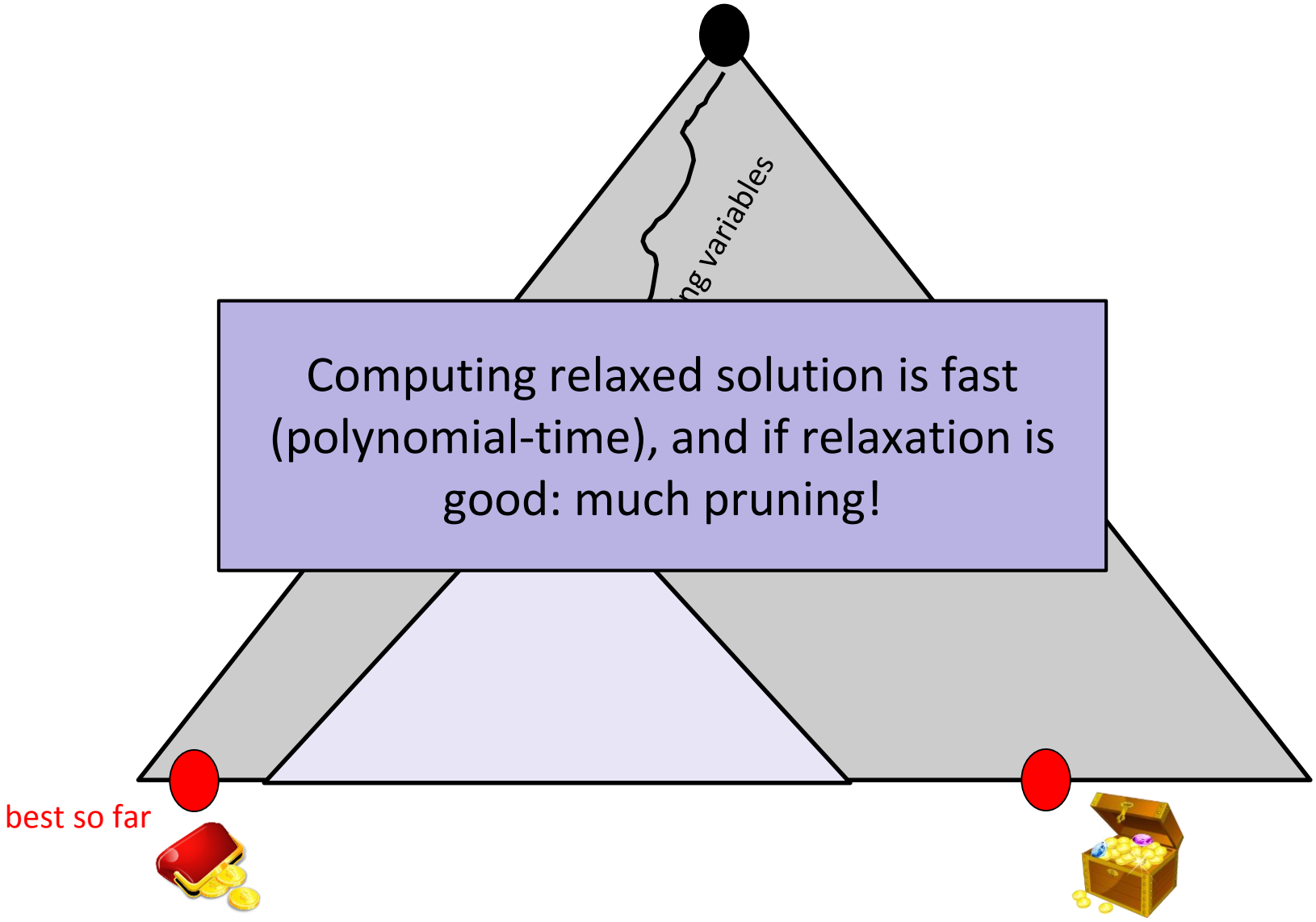
Perform relaxation: Can there be a better solution here at all? If no better relaxed solution, no integer solution either!

best so far

Branch-and-Bound Framework



Branch-and-Bound Framework

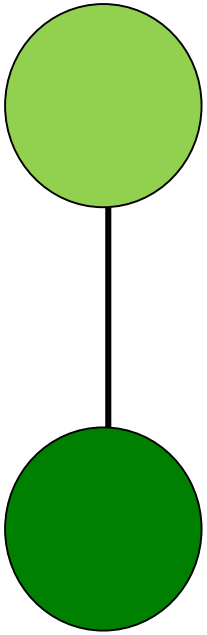


Good Relaxations?

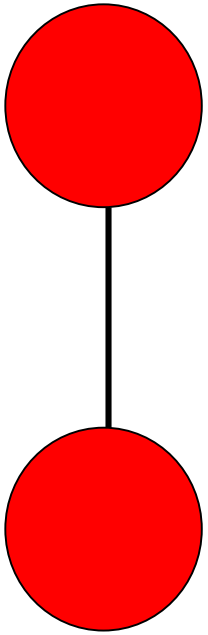
- Recall: Relaxations useful if they give good bounds

- What happens here?

VNet:



Physical Network:

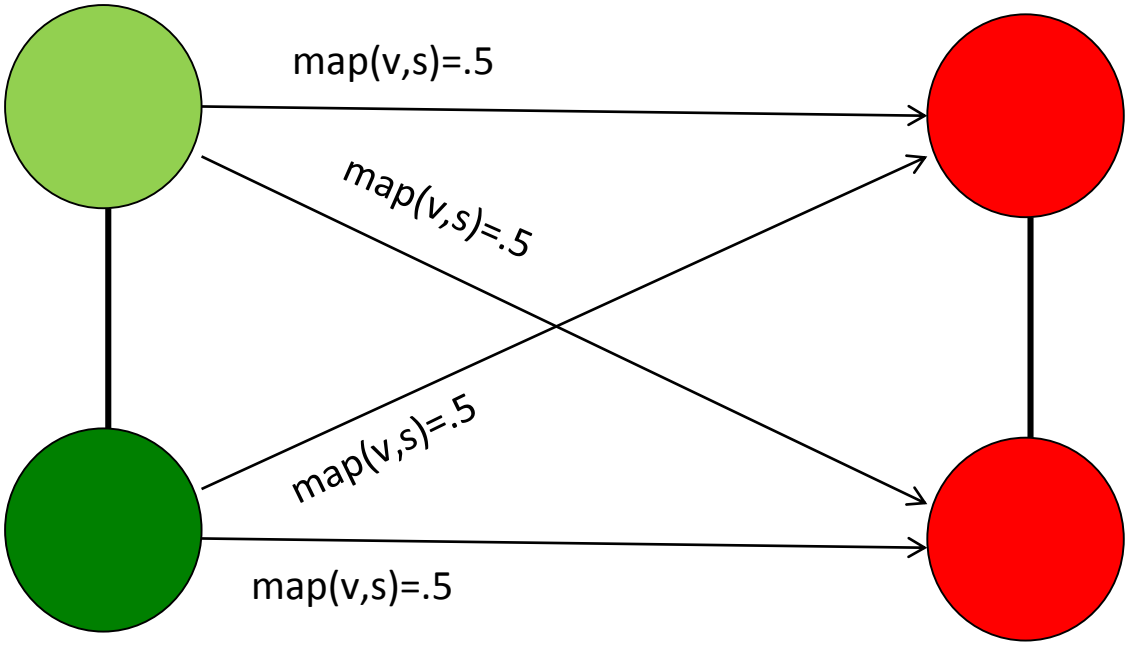


Good Relaxations?

- Recall: Relaxations useful if they give good bounds

- What happens here?

VNet: Physical Network:

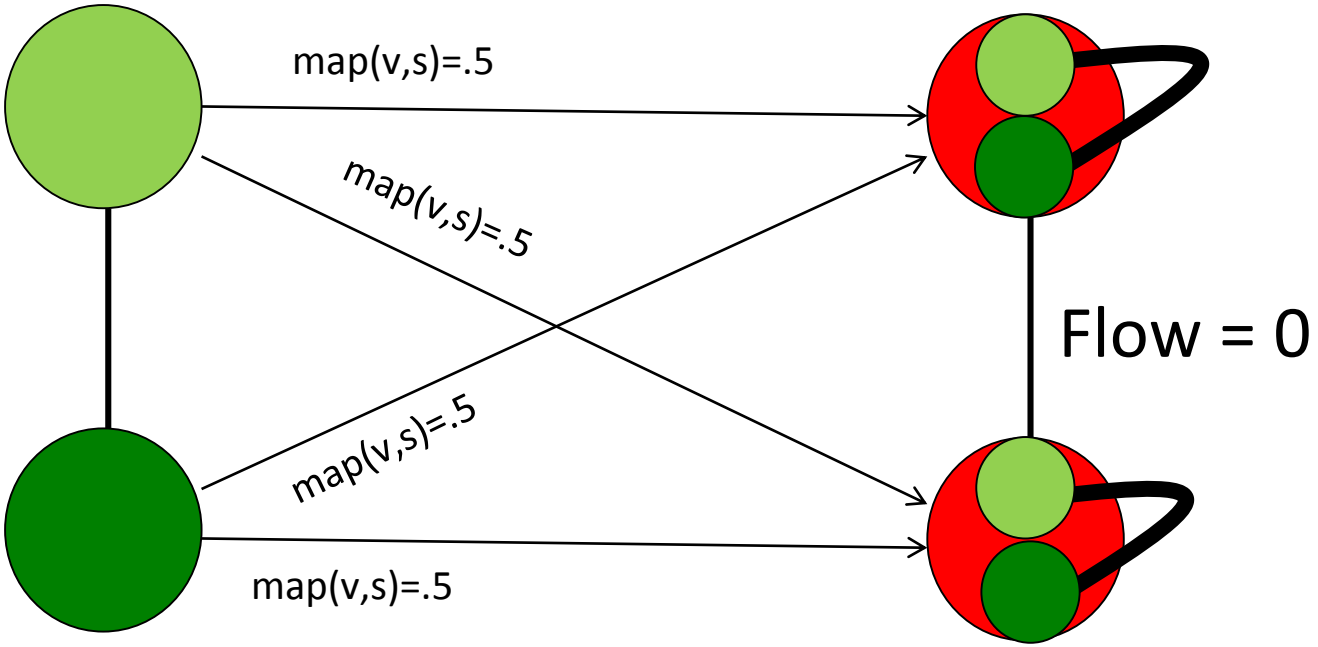


Good Relaxations?

- Recall: Relaxations useful if they give good bounds

- What happens here?

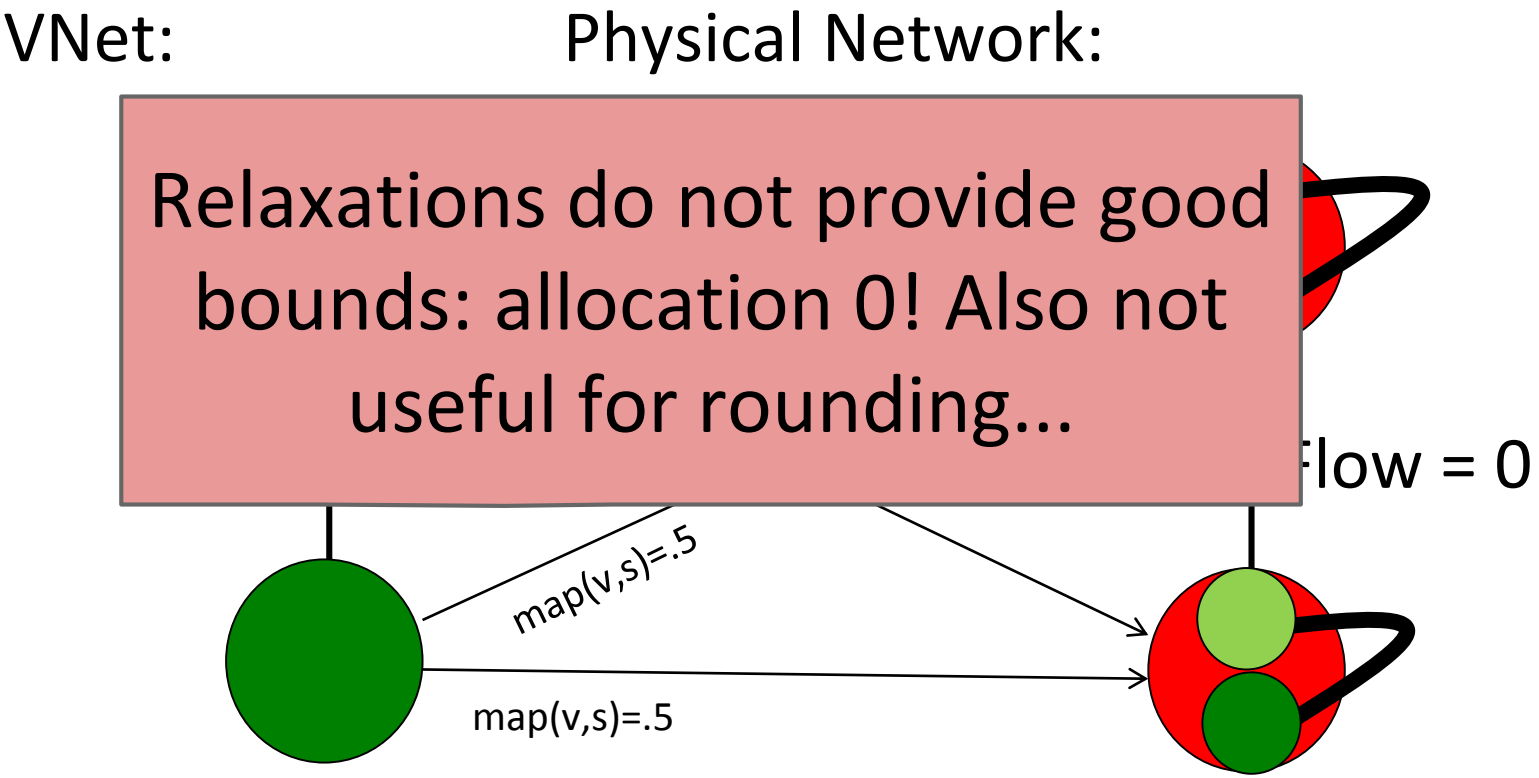
VNet: Physical Network:



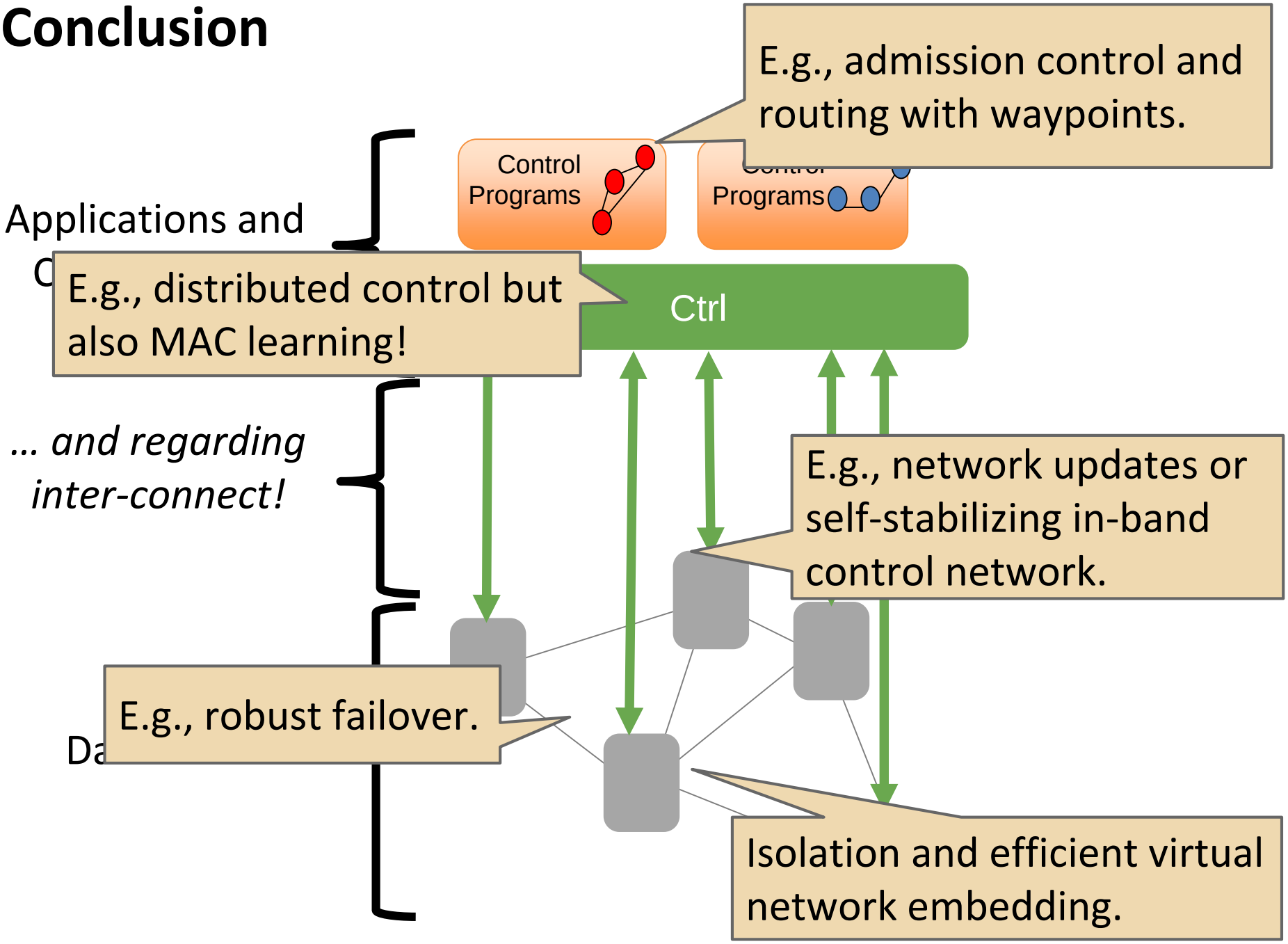
Good Relaxations?

- Recall: Relaxations useful if they give good bounds

- What happens here?



Conclusion


E.g., admission control and routing with waypoints.
Control Programs
Control Programs
Applications and
E.g., distributed control but also MAC learning!
Ctrl
… and regarding inter-connect!
E.g., network updates or self-stabilizing in-band control network.
E.g., robust failover.
Isolation and efficient virtual network embedding.

End of Lecture

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.