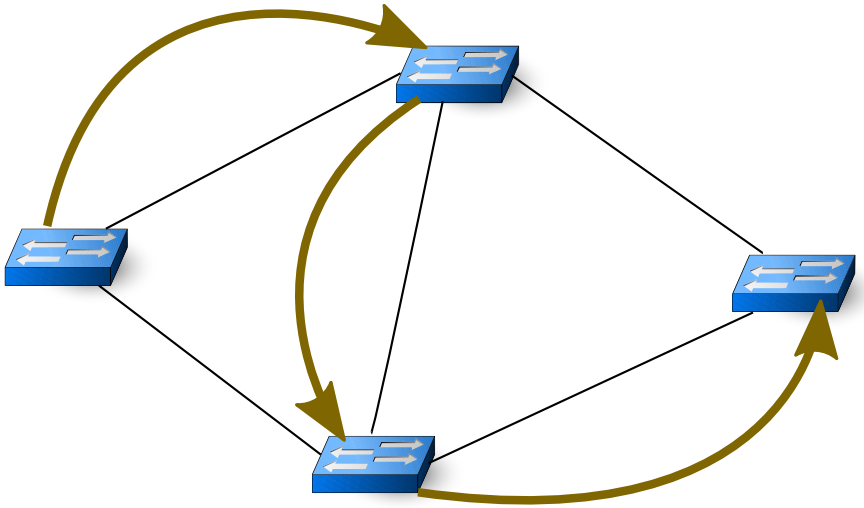
Scheduling Loop-free Network Updates: It's Good to Relax!

Arne Ludwig, Jan Marcinkowski and Stefan Schmid

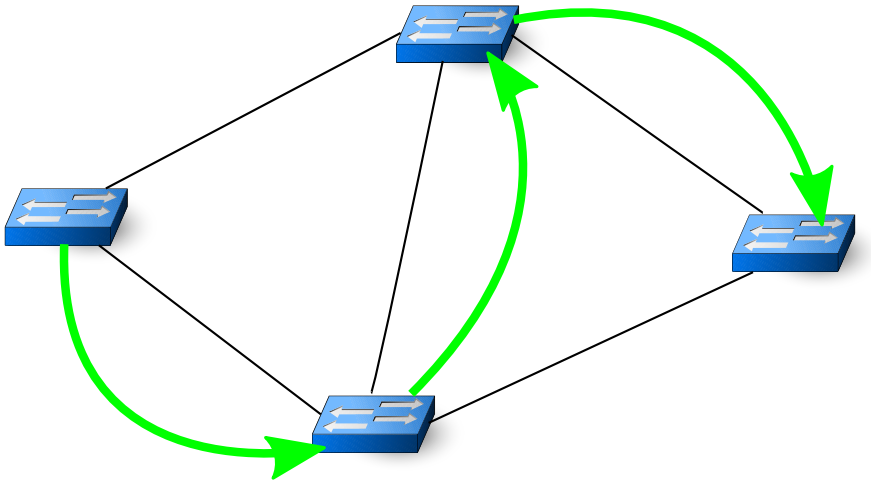
Uniwersytet Wrocławski

Update happens

- Network updates happen
  - Changing security policies
  - Traffic Engineering



- Potential high damage if inconsistent
  - Security policy violation
  - Shared cloud resources



Strong vs weak consistency

Strong consistency[1]	Weak consistency[2]
Alg: - Two phase Commit → Either old or new route Cons: - Needs more switch memory - Problematic with middleboxes (changed headers) - Very late first effects	

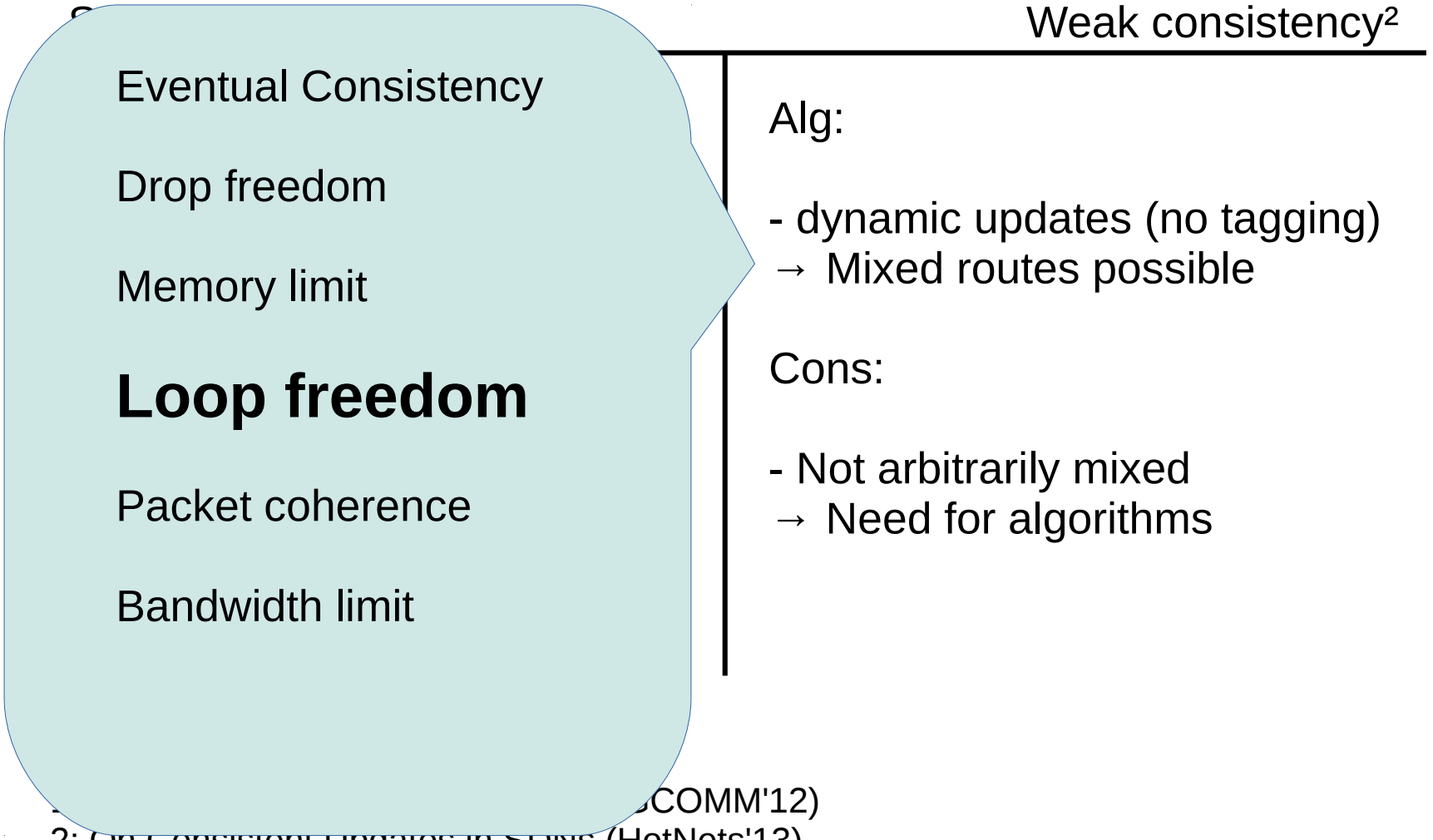
1: Abstraction for network update (SIGCOMM'12)
2: On Consistent Updates in SDNs (HotNets'13)

Strong vs weak consistency

Strong consistency[1]	Weak consistency[2]
Alg: - Two phase Commit → Either old or new route Cons: - Needs more switch memory - Problematic with middleboxes (changed headers) - Very late first effects	Alg: - dynamic updates (no tagging) → Mixed routes possible Cons: - Not arbitrarily mixed → Need for algorithms

1: Abstraction for network update (SIGCOMM'12)
2: On Consistent Updates in SDNs (HotNets'13)

Strong vs weak consistency



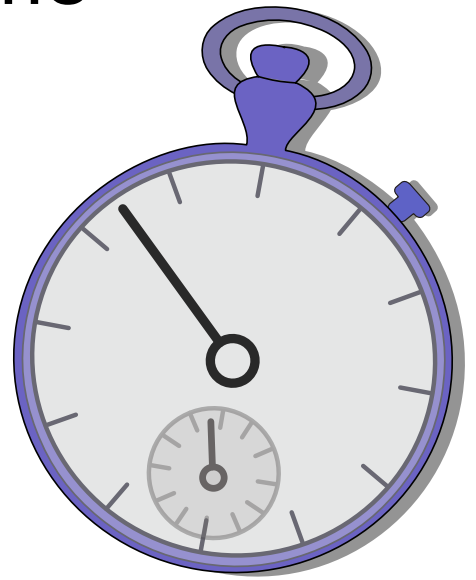
COMM'12)

2: On Consistent Updates in SDNs (HotNets'13)

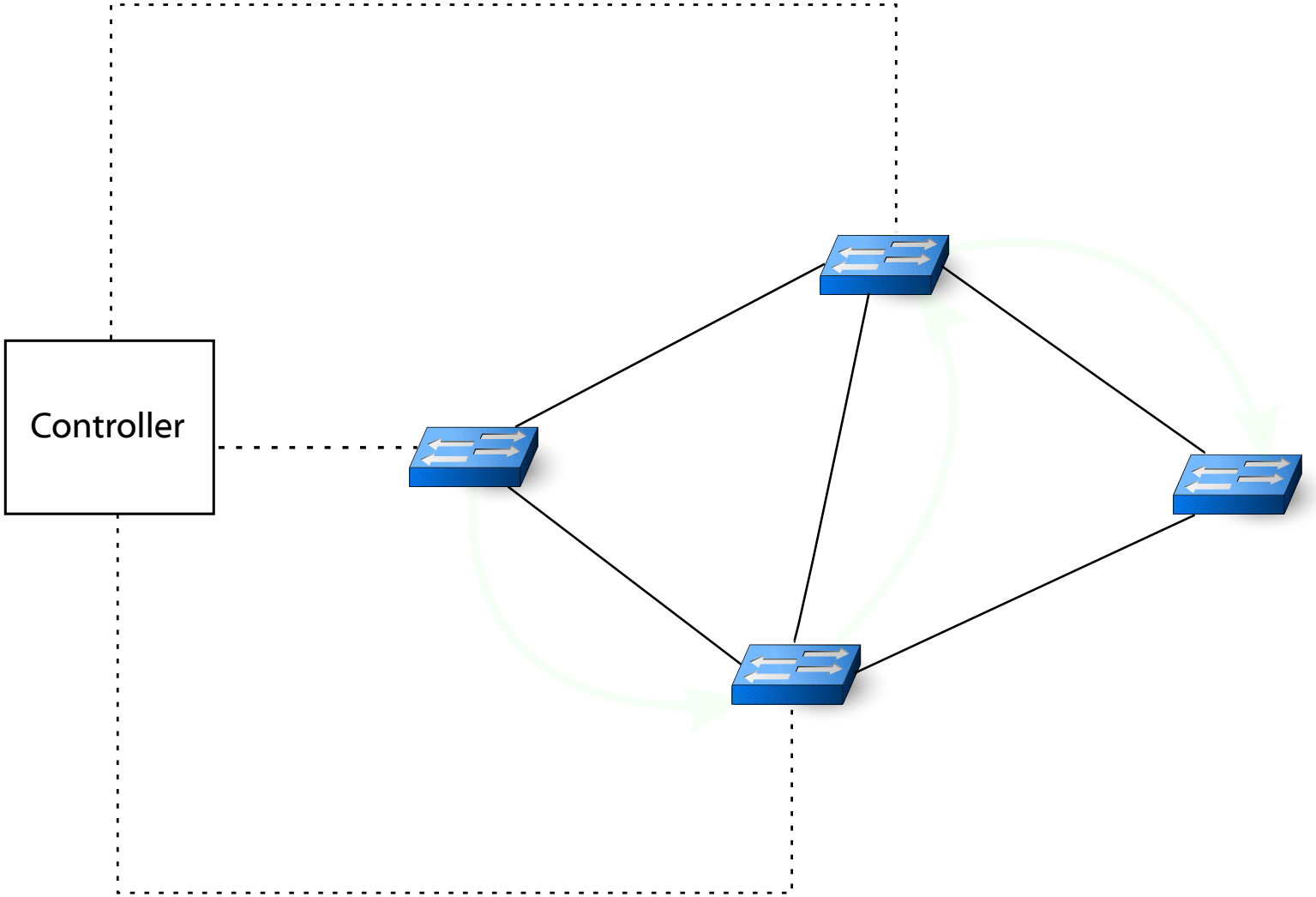
The challenge: Fast and asynchronous updates

- Interactions take time (Kuźniar PAM'15, Dionysus SIGCOMM'14)
- Reach consistent state as soon as possible

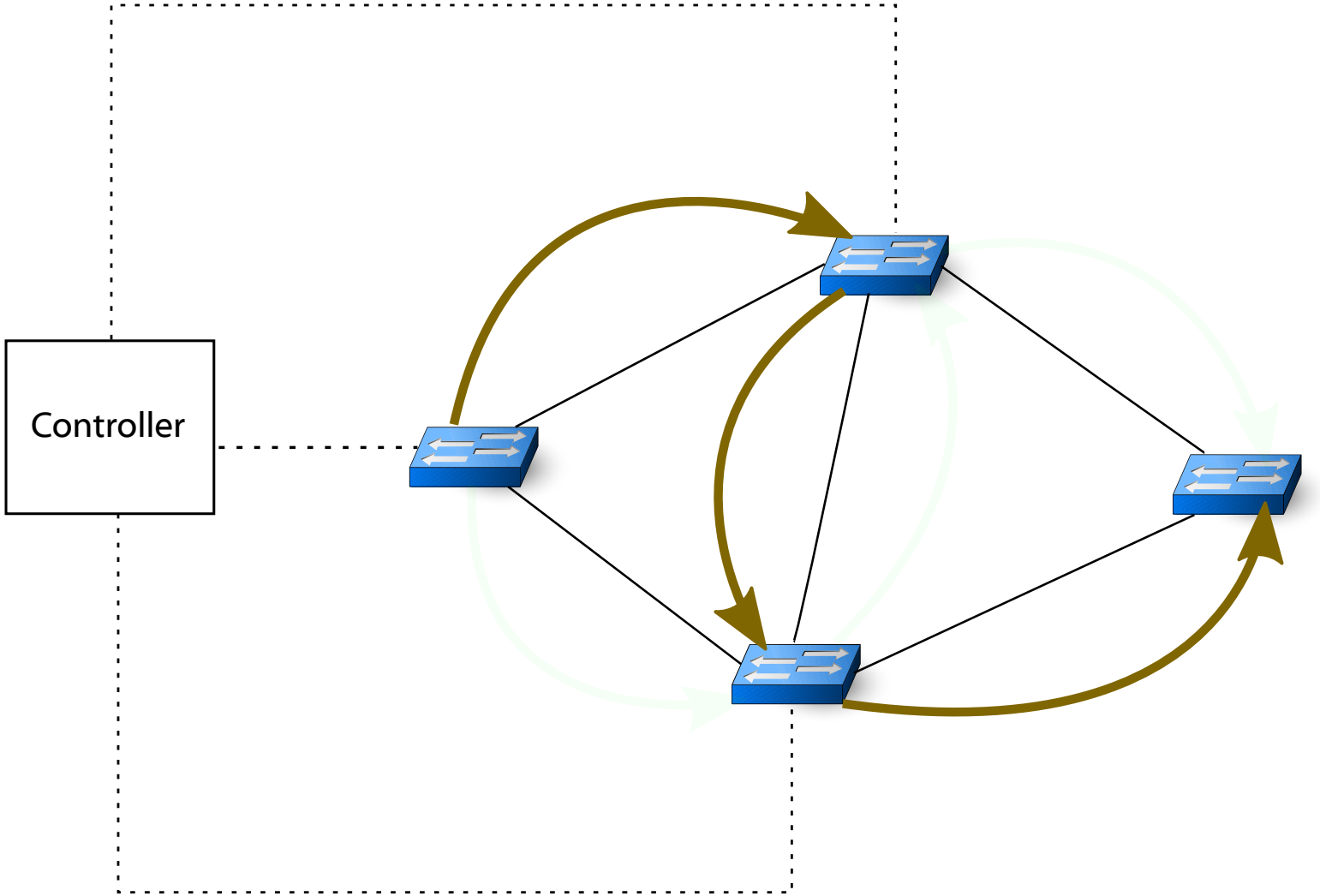
  → Minimize the overall update time



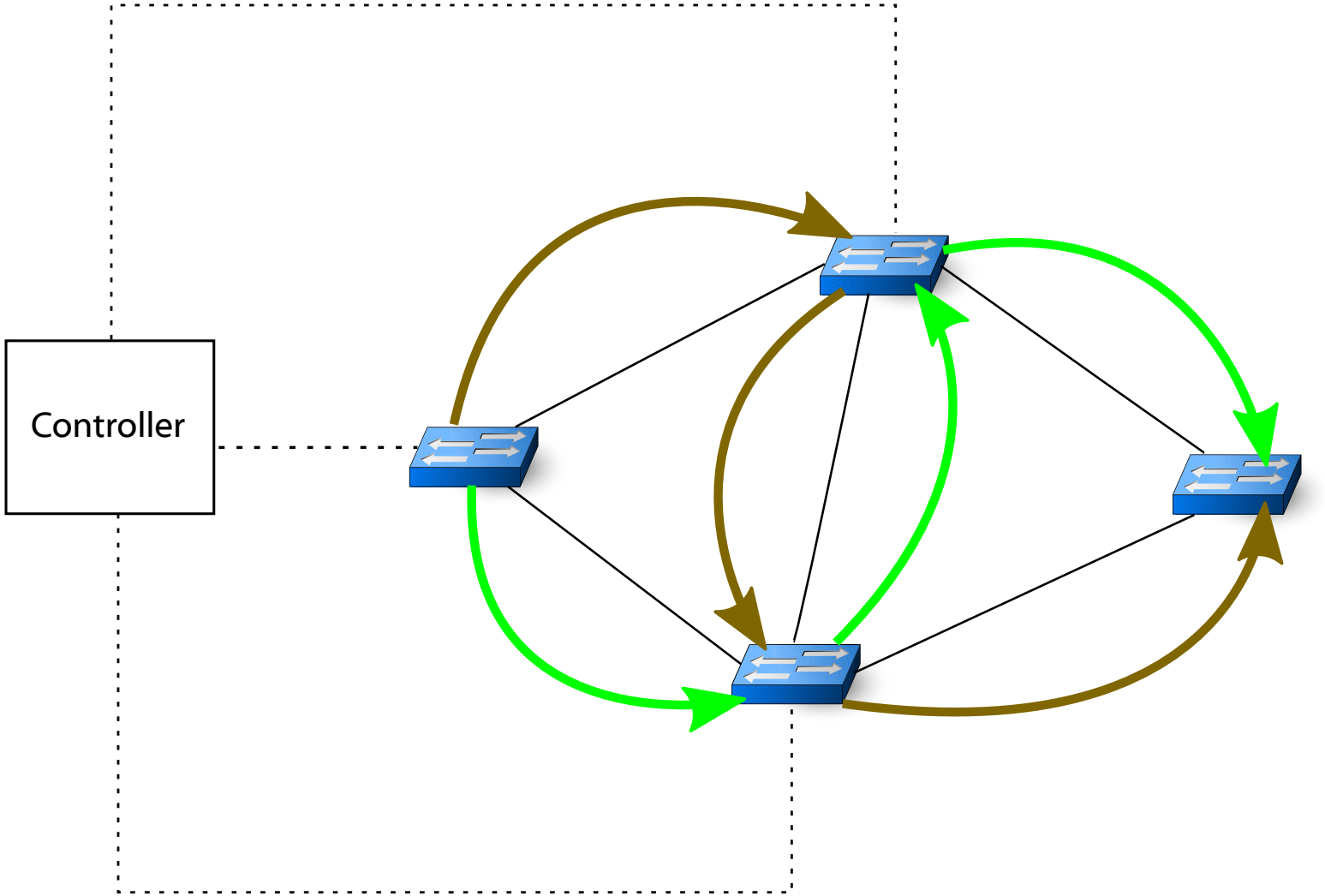
Asynchronous updates



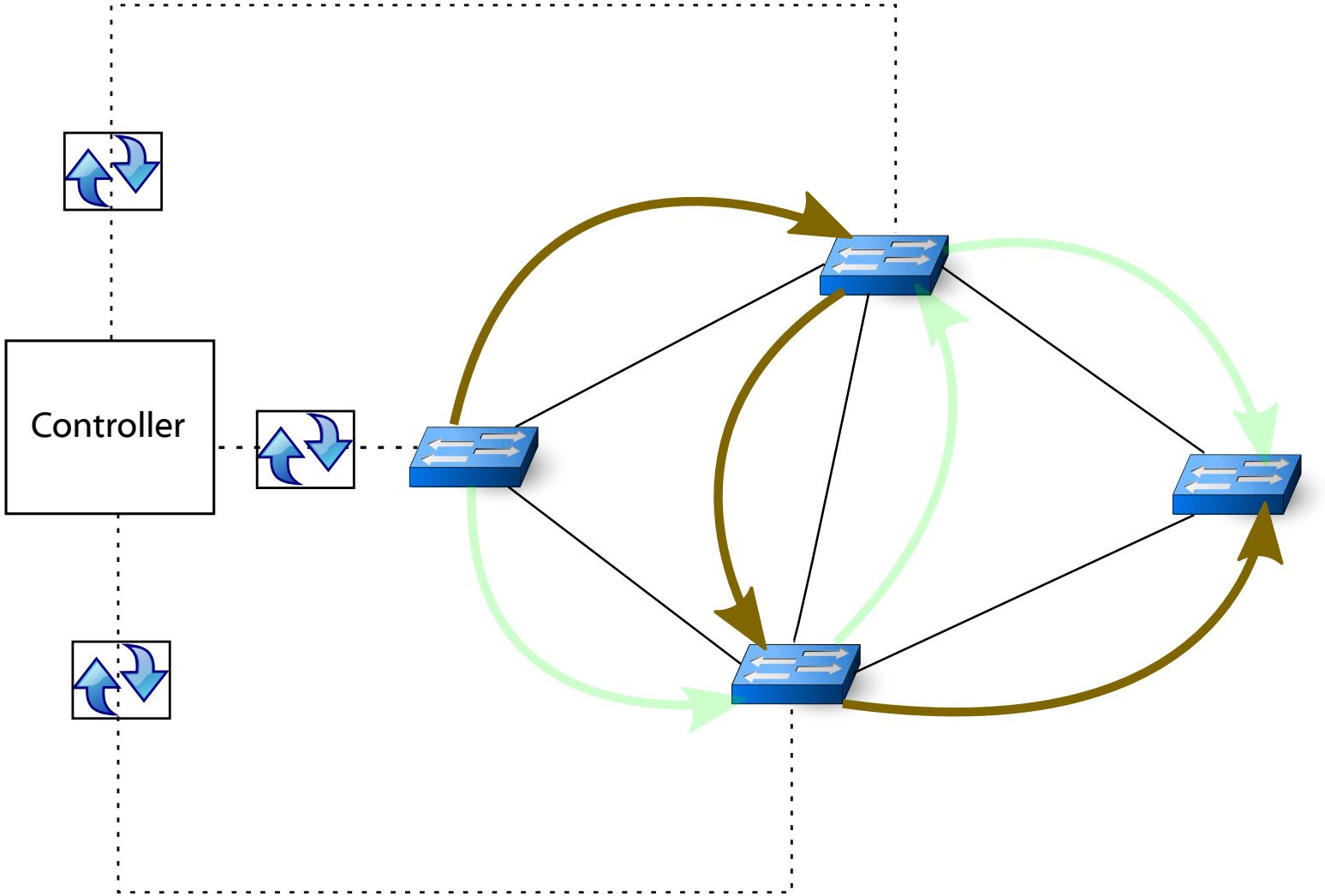
Asynchronous updates



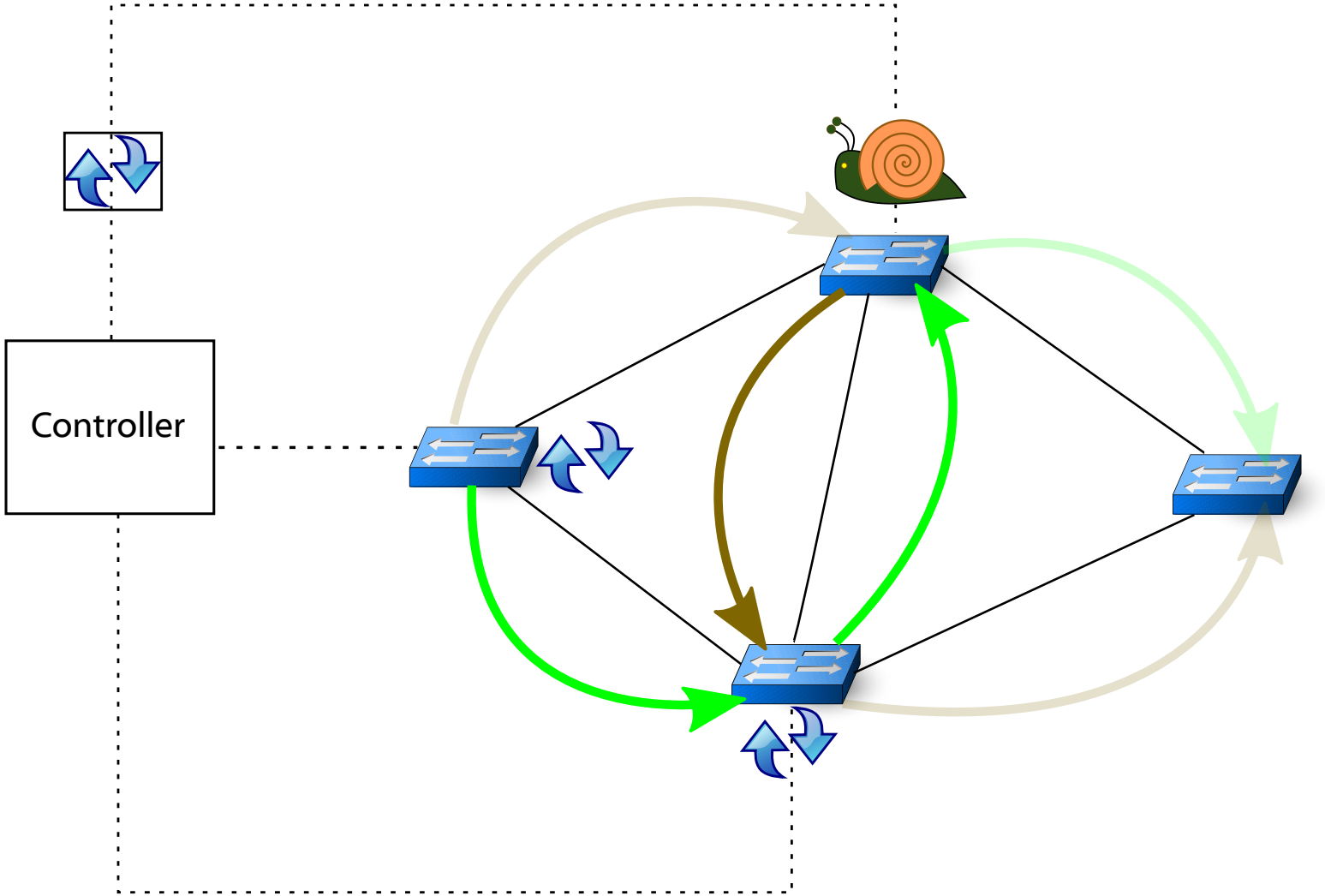
Asynchronous updates



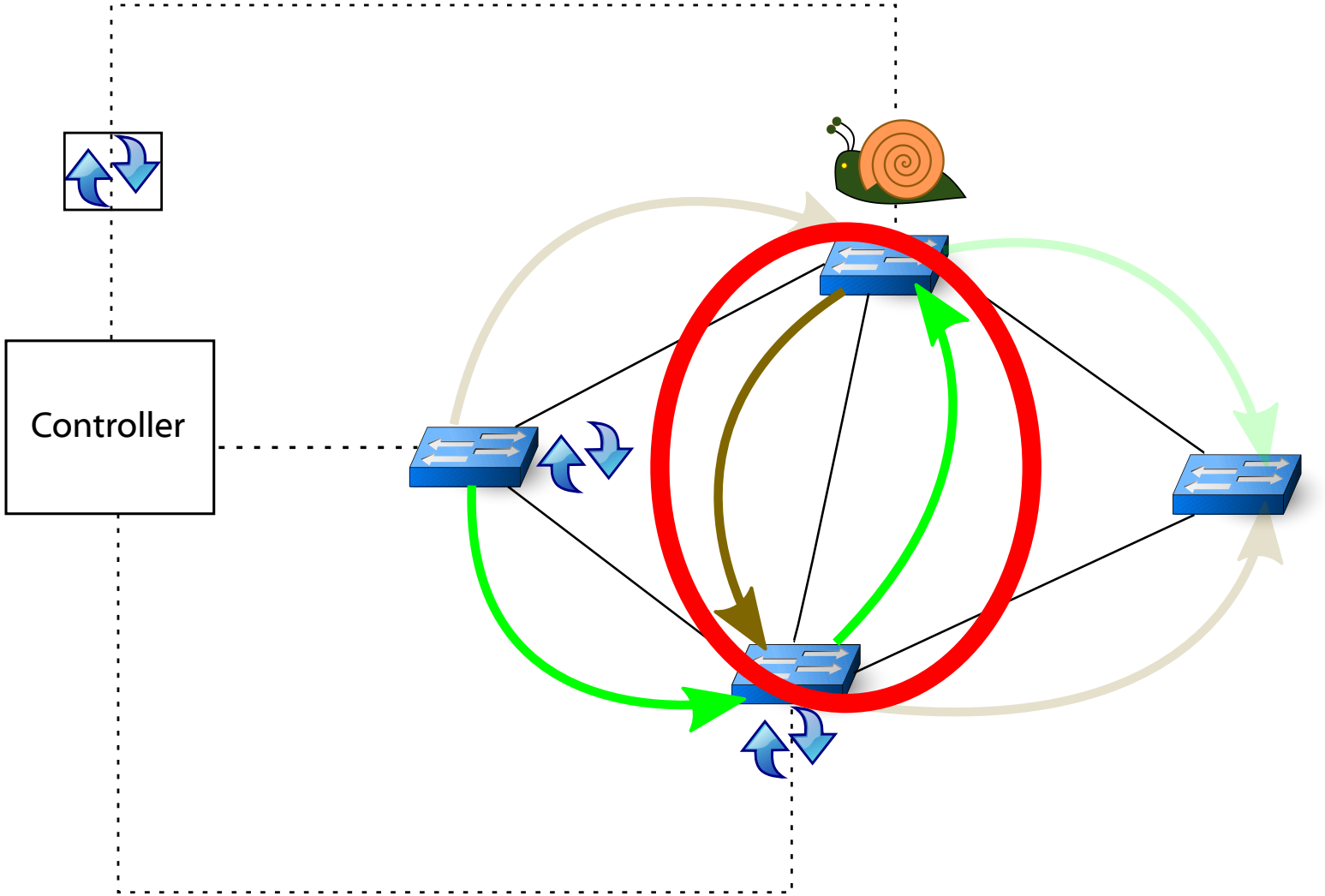
Asynchronous updates



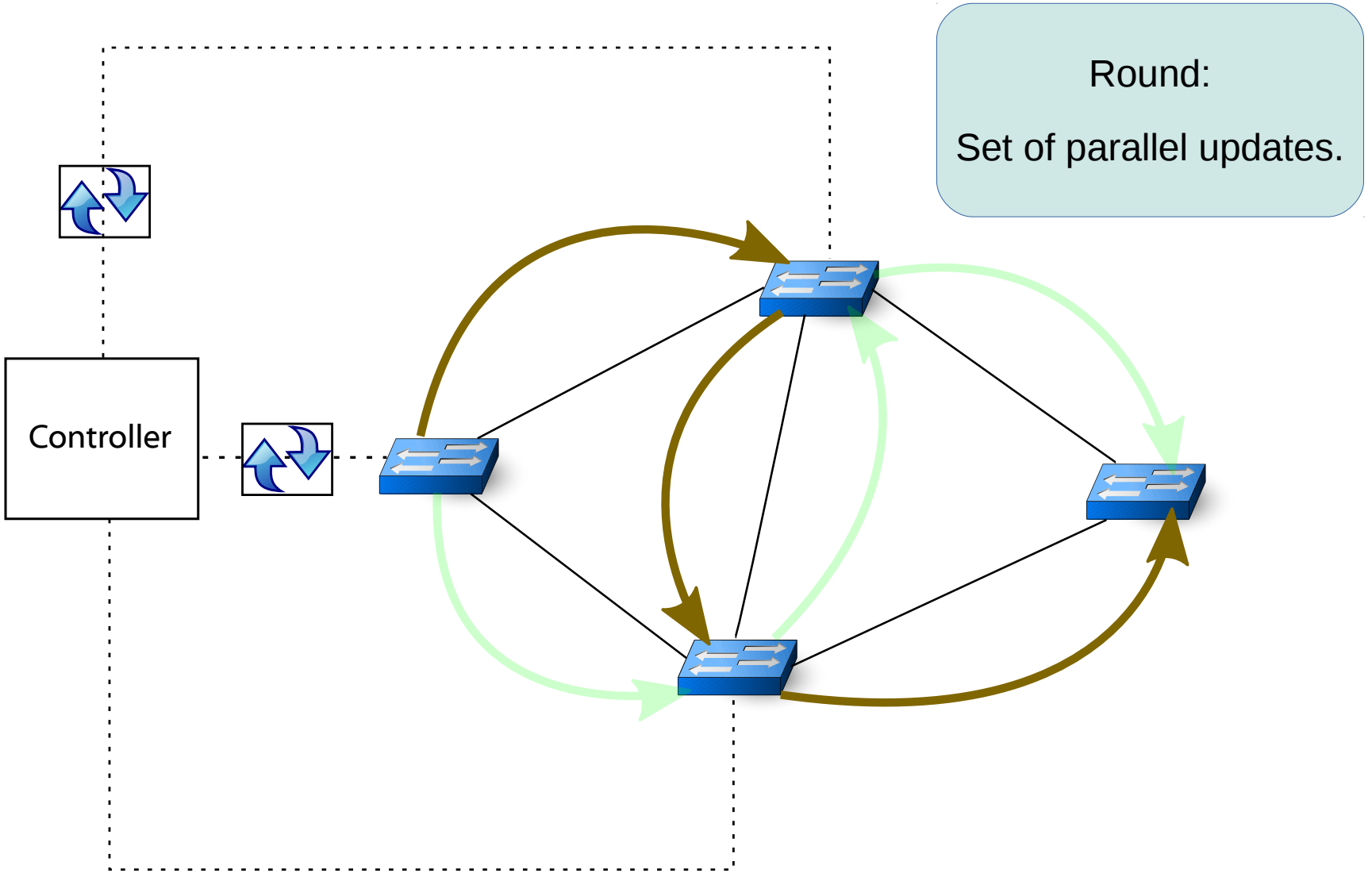
Asynchronous updates



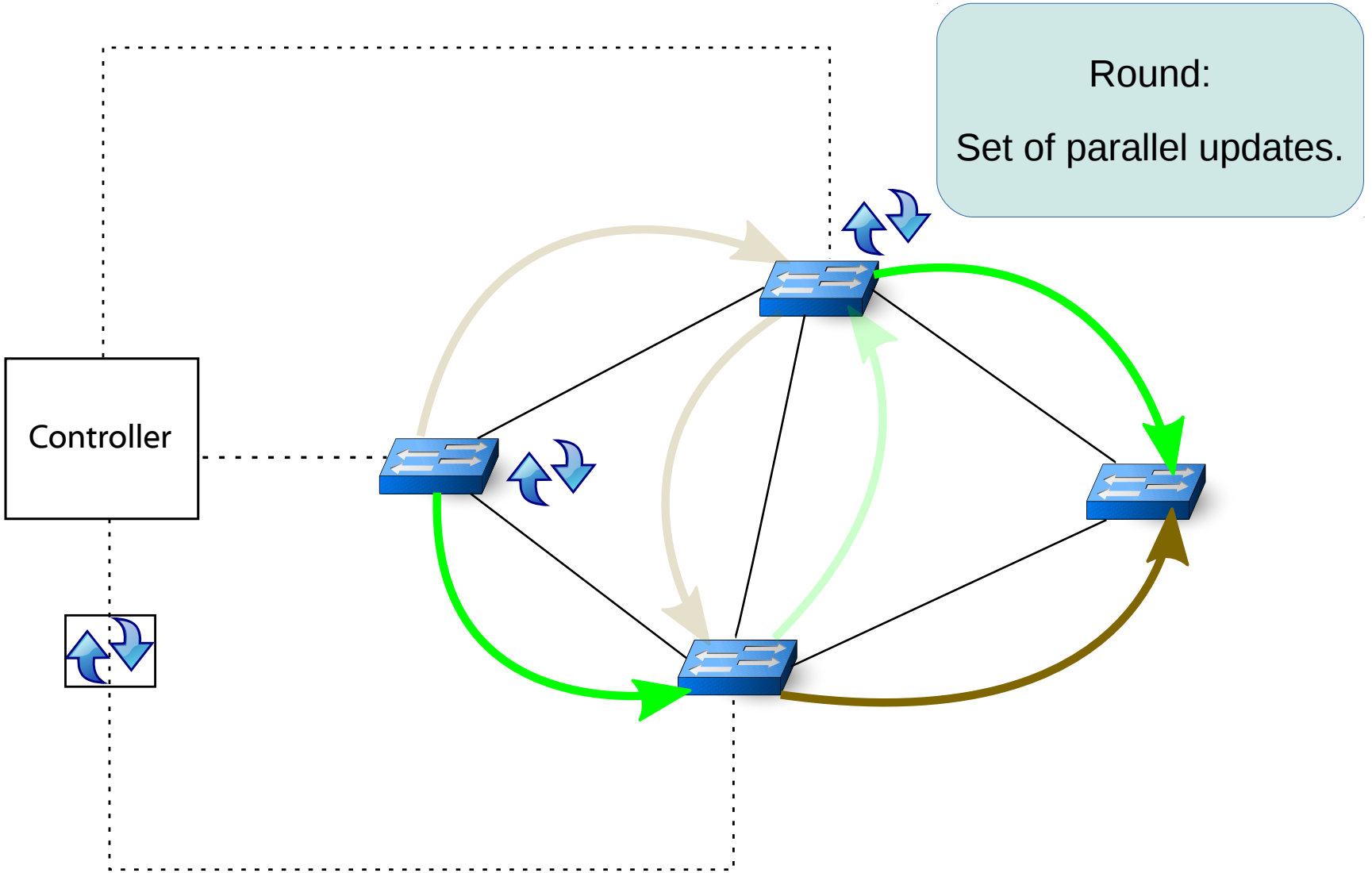
Asynchronous updates



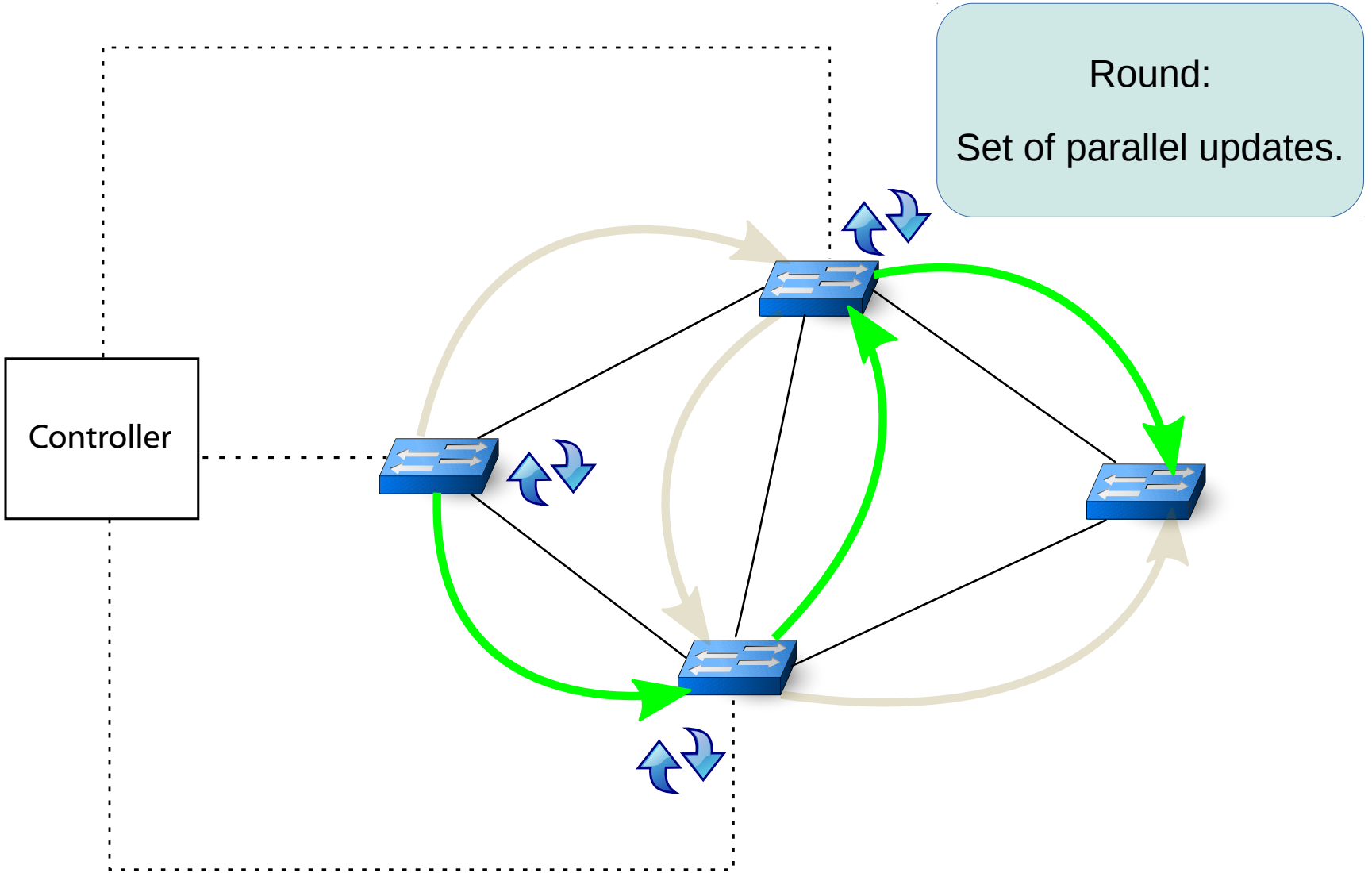
Asynchronous updates - solution



Asynchronous updates - solution

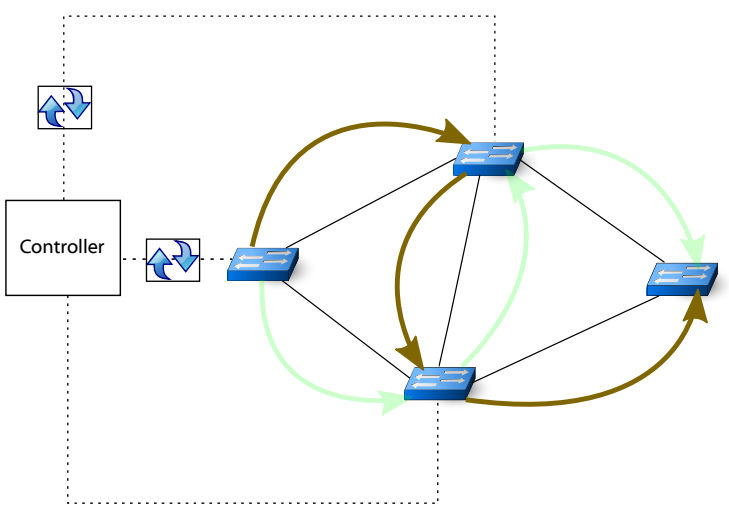


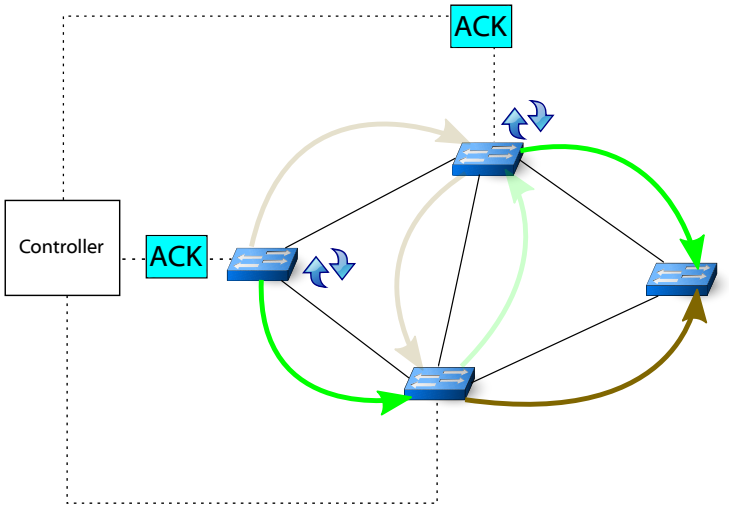
Asynchronous updates - solution



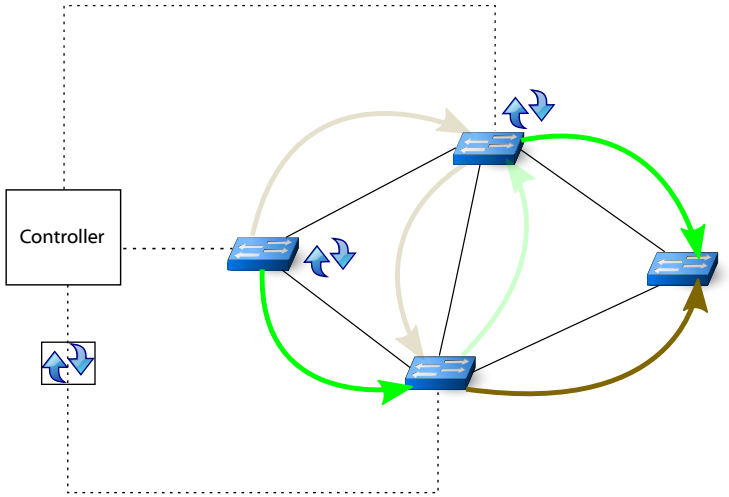
Asynchronous updates - solution

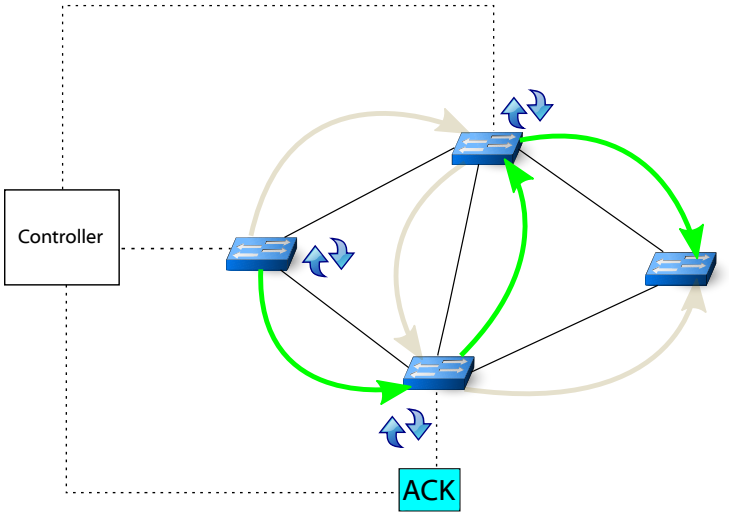
Round: 1





Round: 2





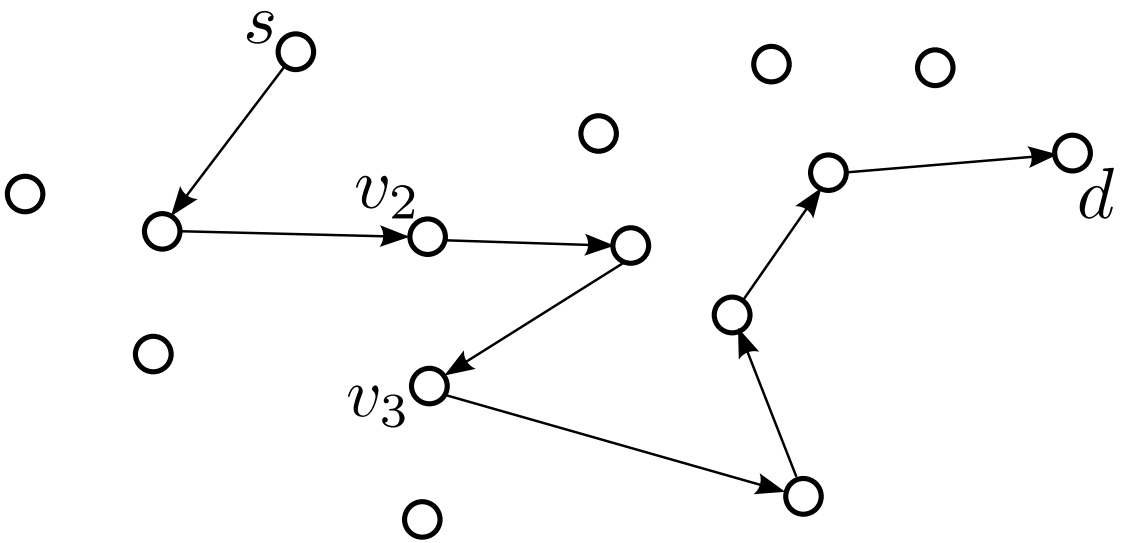
Minimal number of rounds for loop-free updates?

Outline

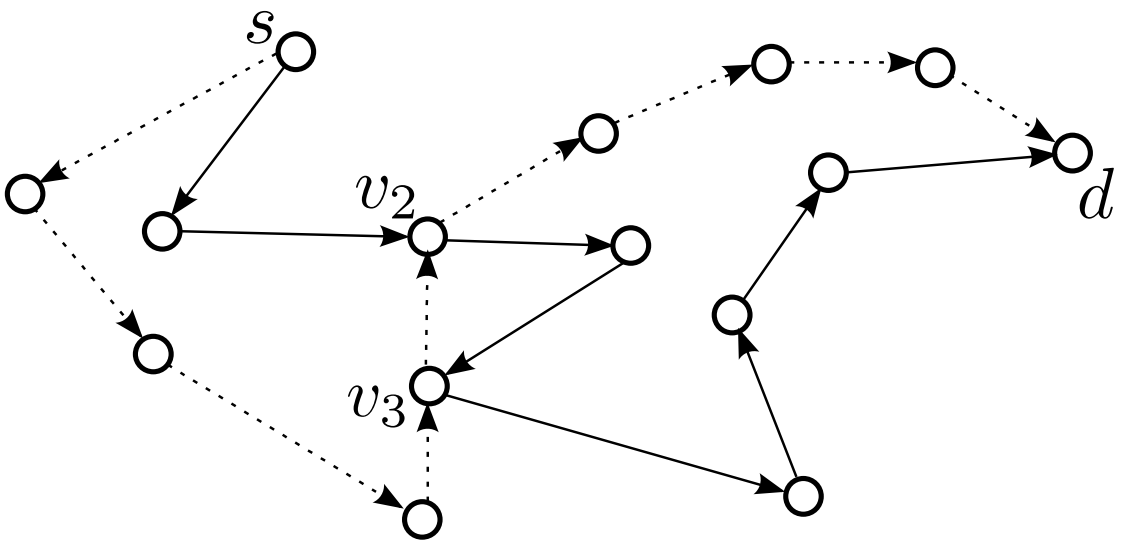
- Model
- 2-rounds is easy
- 3-rounds is hard
- Takes up to n rounds
- It's good to relax

Trivial compression

- Solid lines = current path



Trivial compression



- Solid lines = current path
- Dashed lines = new path

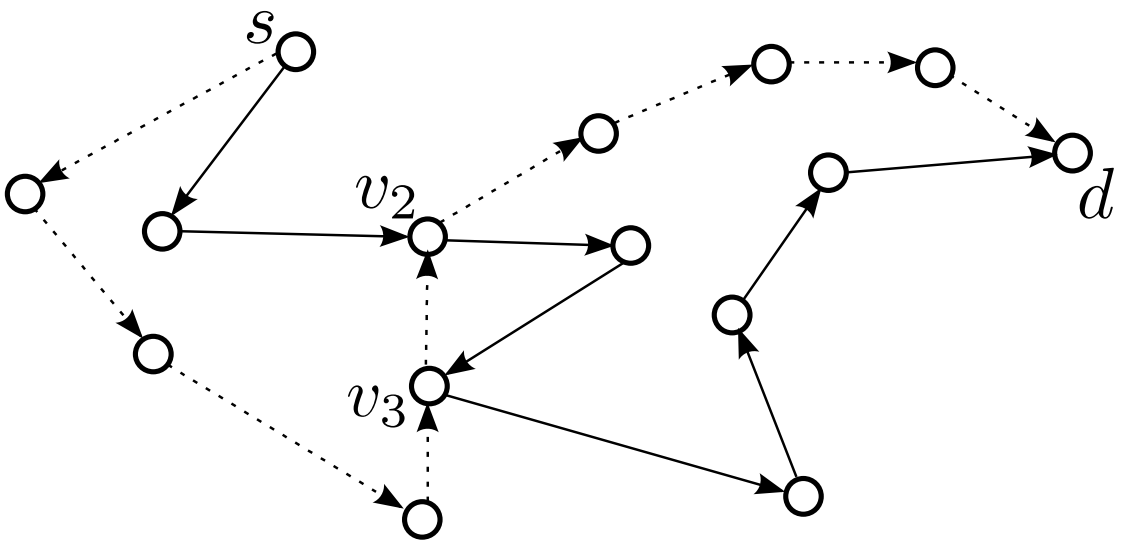
Flow-specific path

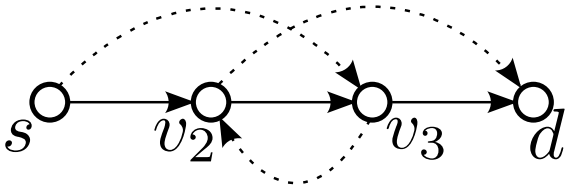
Trivial compression

Solid lines = current path

Dashed lines = new path

Flow-specific path



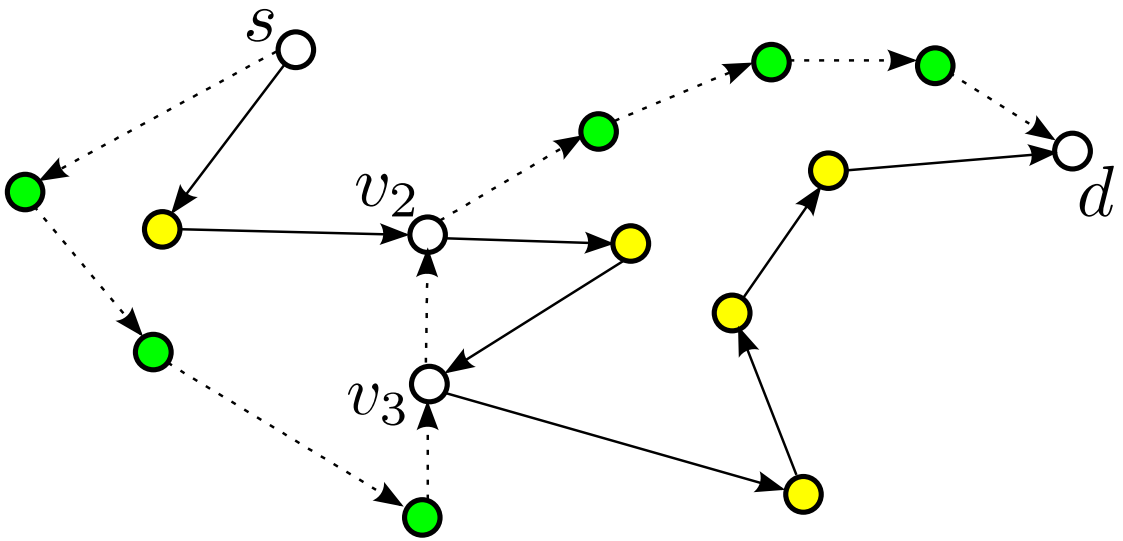


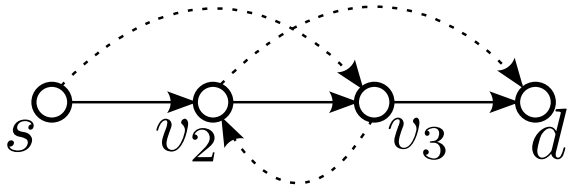
Trivial compression

Solid lines = current path

Dashed lines = new path

Flow-specific path

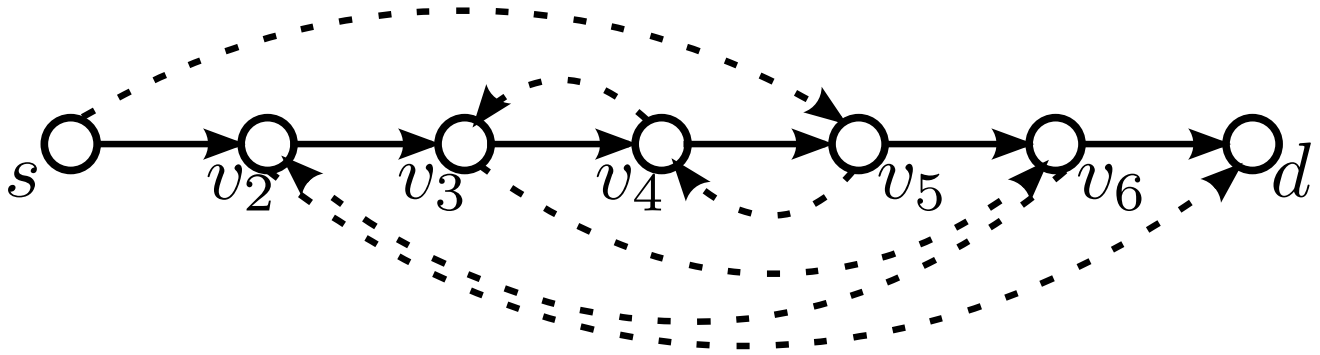




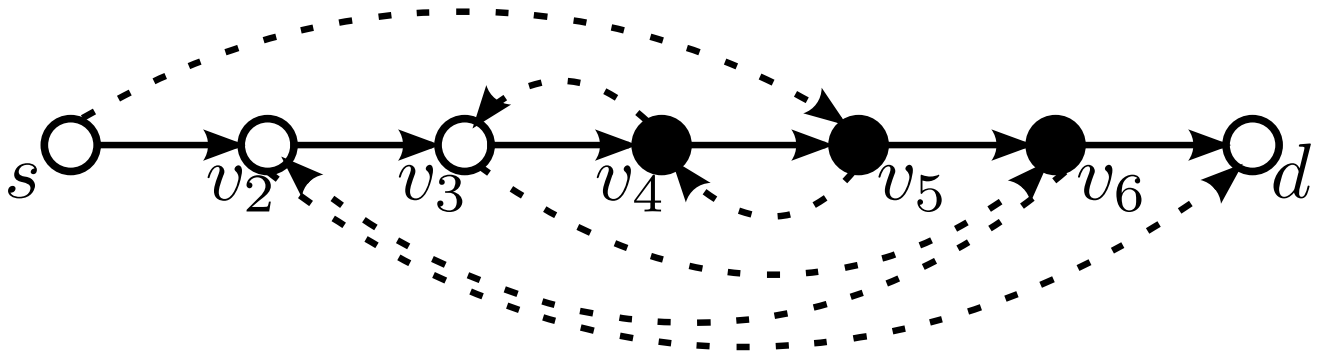
Safe to be updated

Safe to be left untouched

Basic example



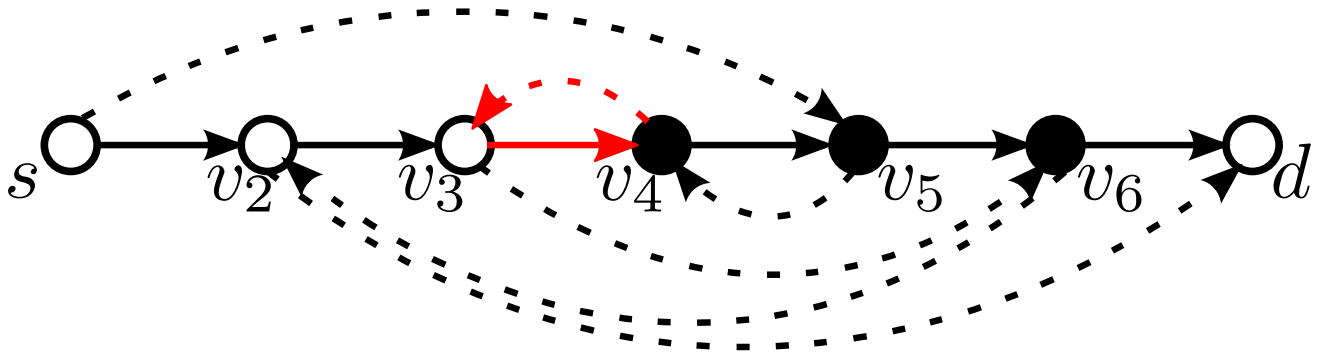
Basic example



F (white)
B (black)

- Forward (F) nodes → updateable
- Backward (B) nodes → not updateable

Basic example

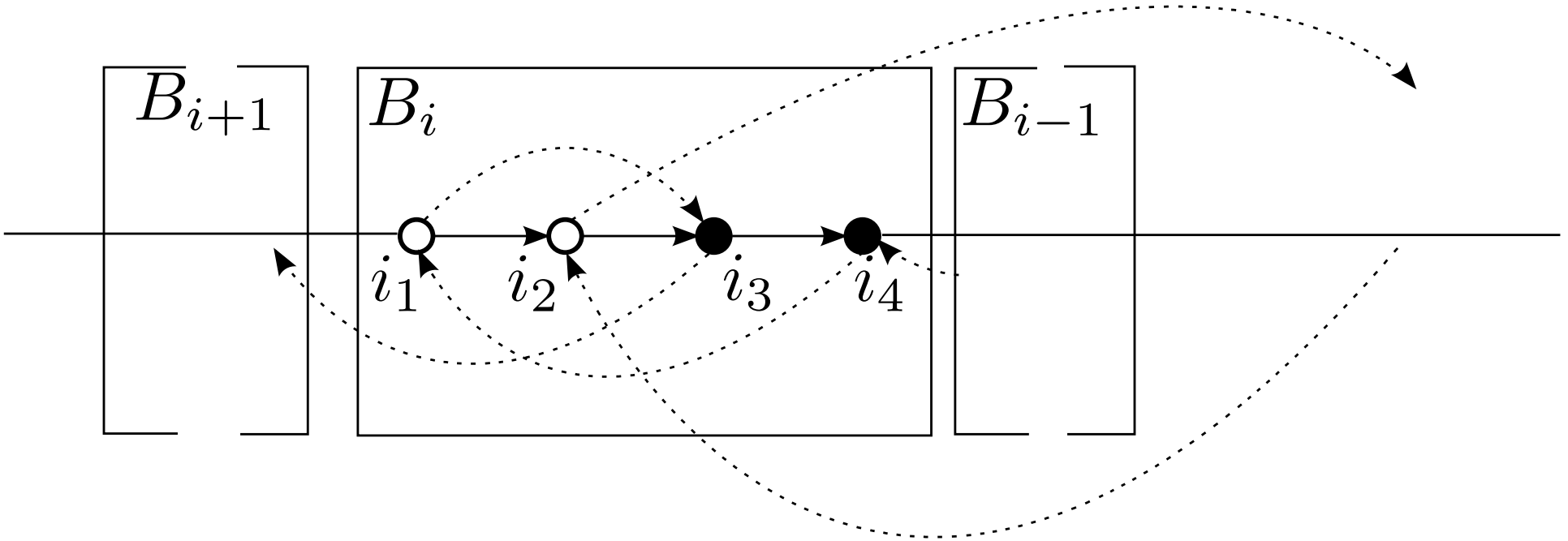


F (white)
B (black)

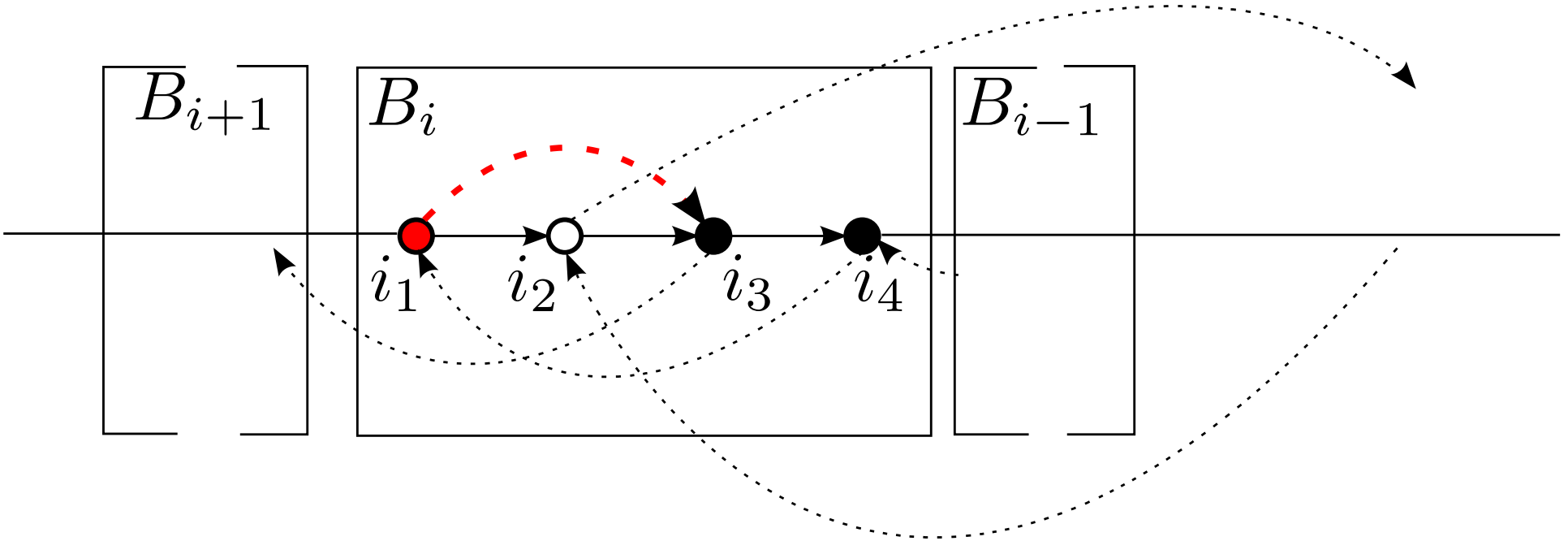
- Forward (F) nodes → updateable
- Backward (B) nodes → not updateable

How to update a network in a (transiently) loop-free manner?

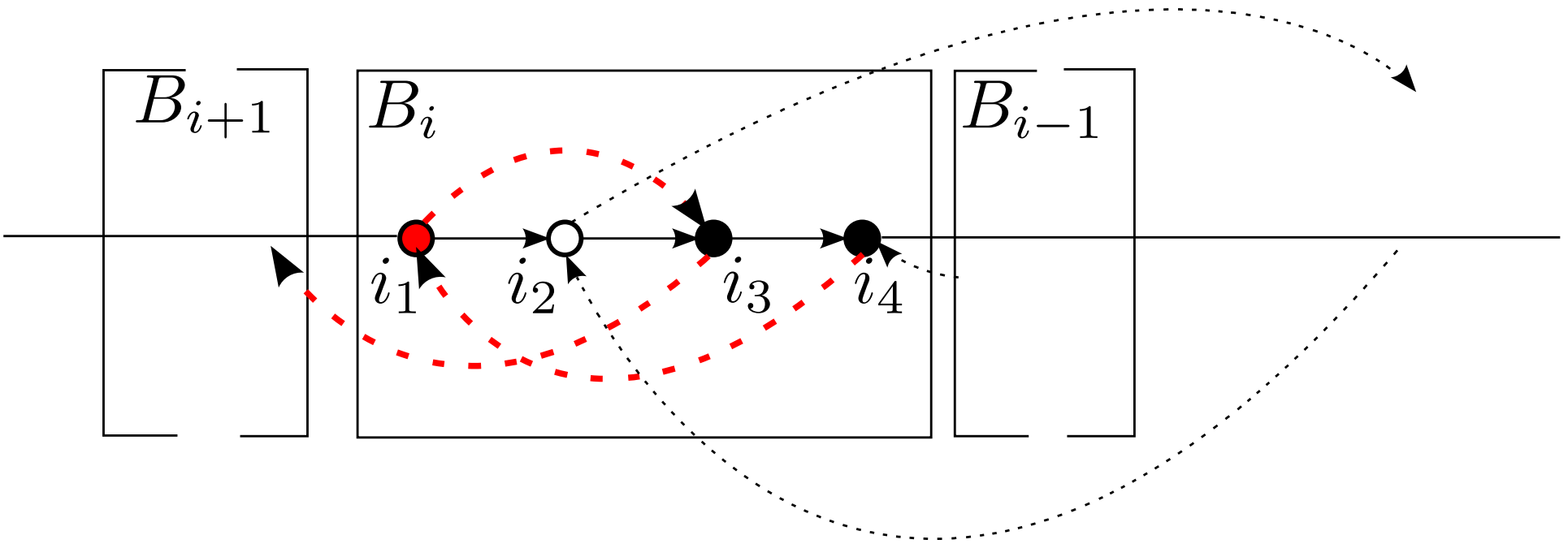
What about greedy?



What about greedy?



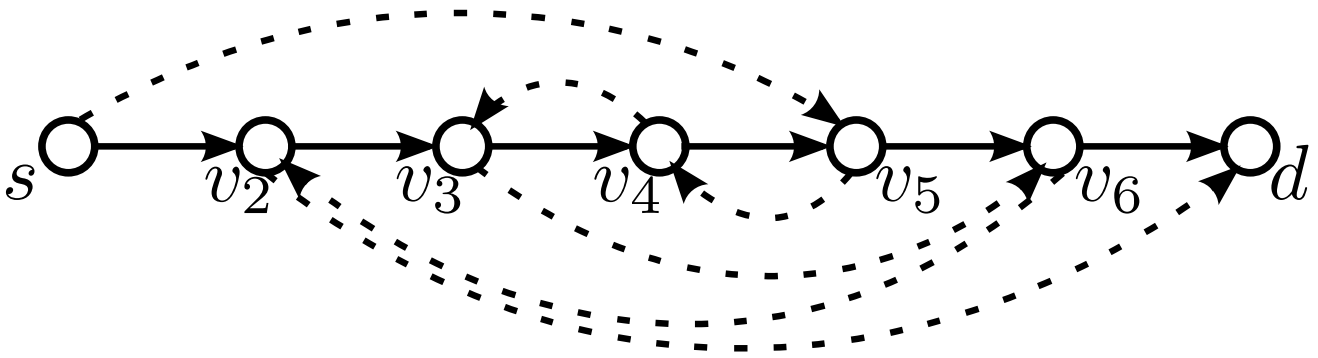
What about greedy?



- From $O(1)$ to $\Omega(n)$ rounds

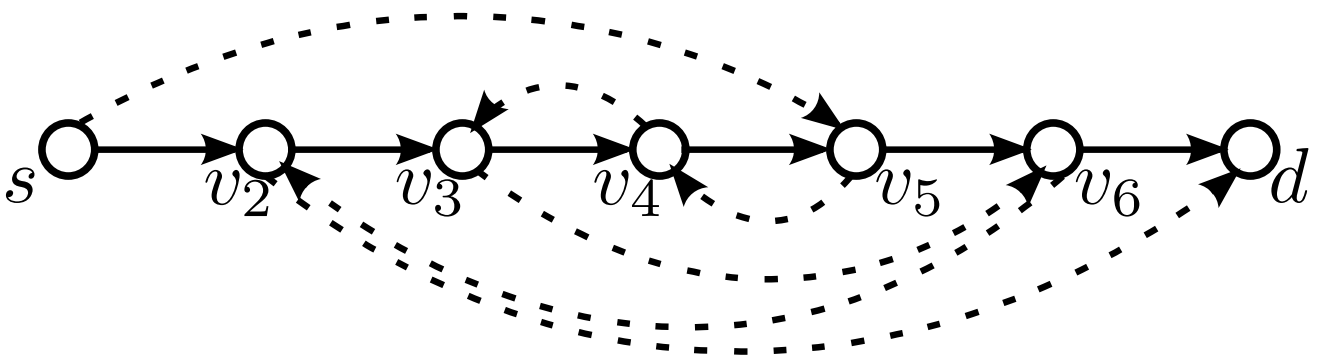
Reversed update pattern

- Standard:

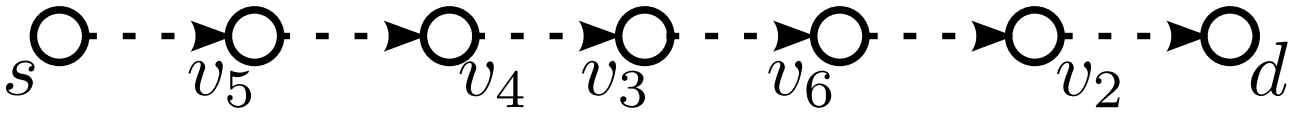


Reversed update pattern

- Standard:

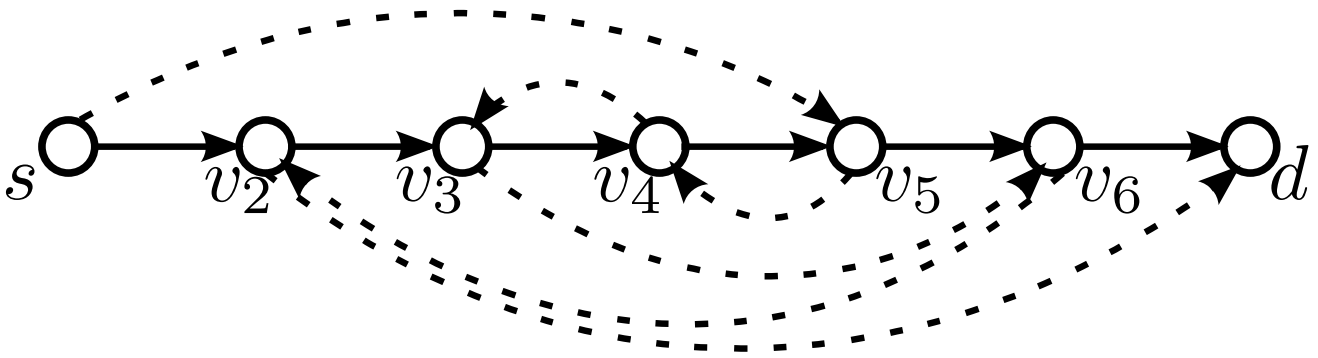


- Reverse:

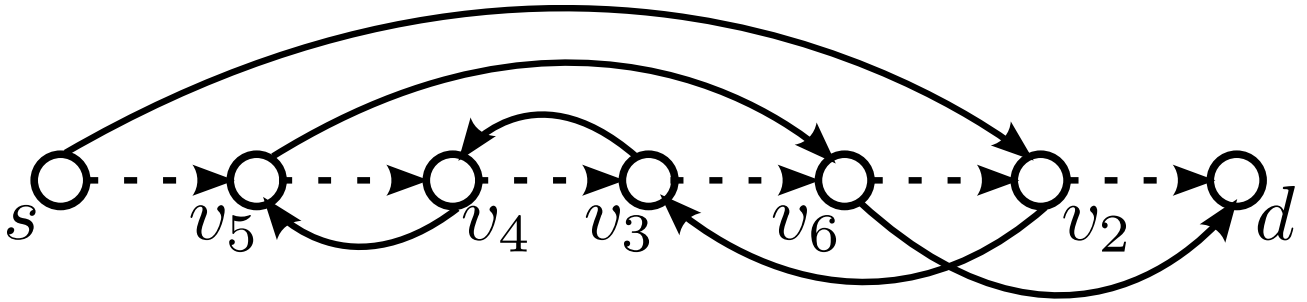


Reversed update pattern

- Standard:

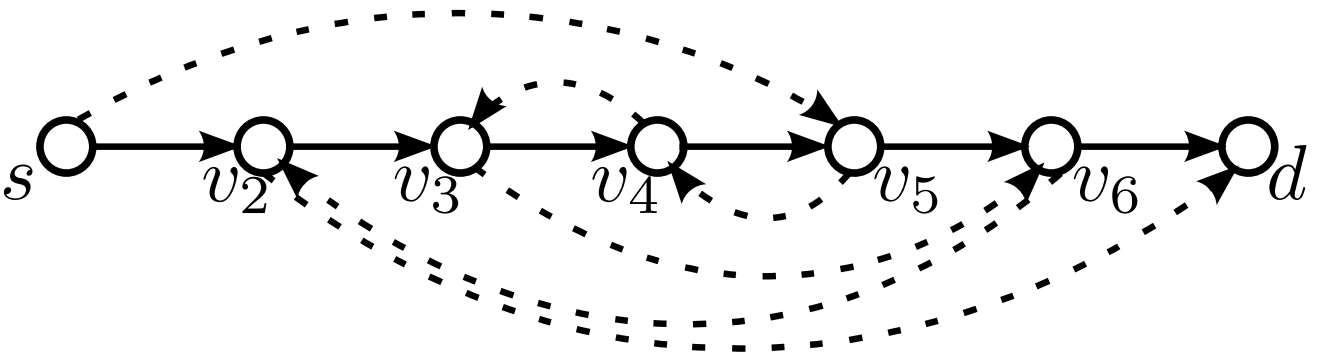


- Reverse:

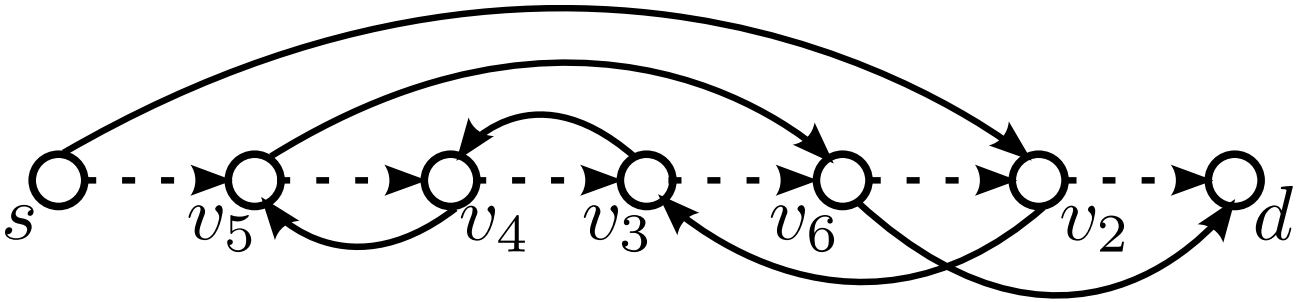


Reversed update pattern

- Standard:



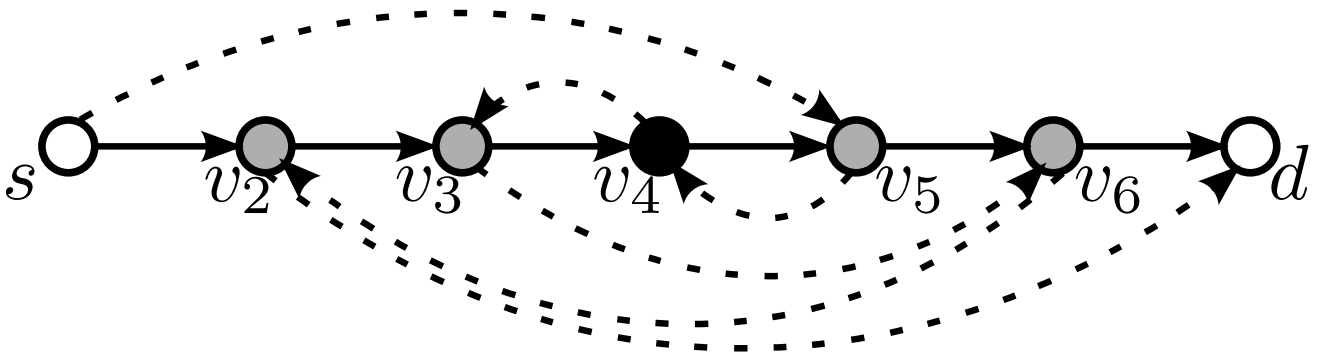
- Reverse:



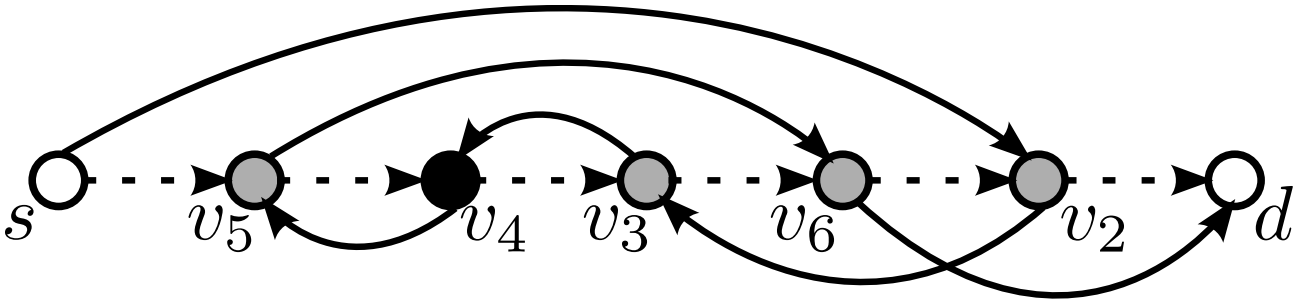
A valid update schedule for standard is a flipped valid update schedule for reverse!

Reversed update pattern

- Standard:



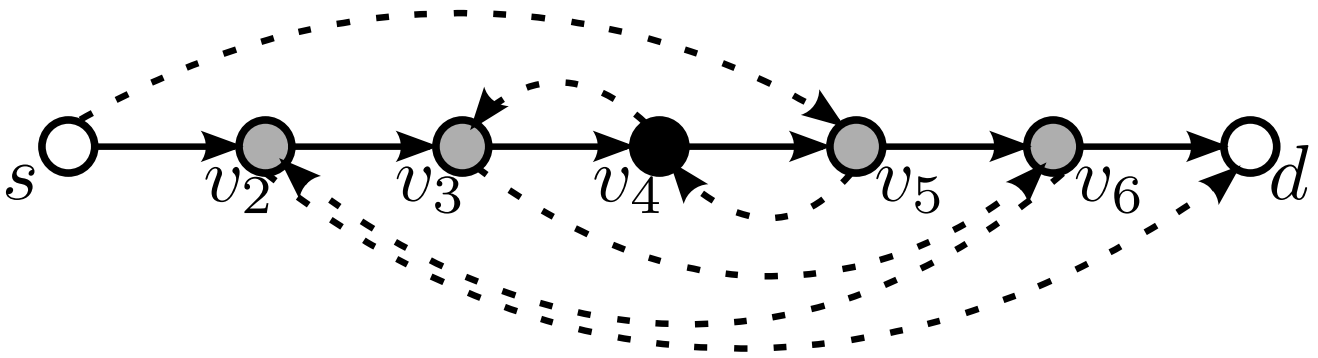
- Reverse:



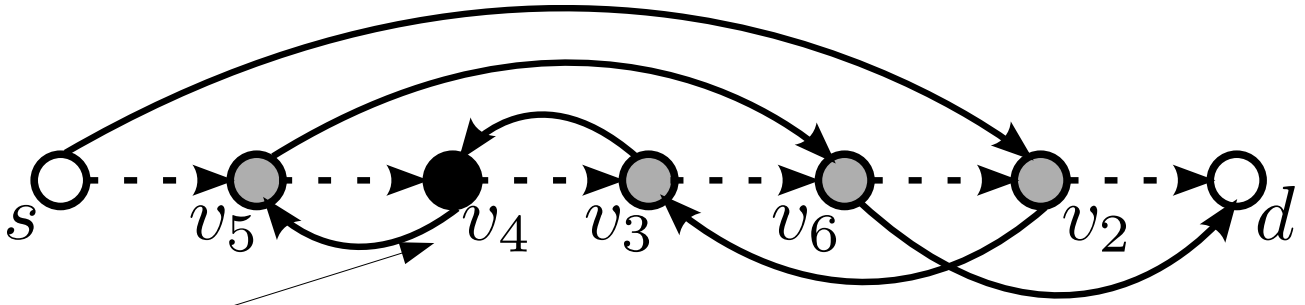
- FF

Reversed update pattern

- Standard:



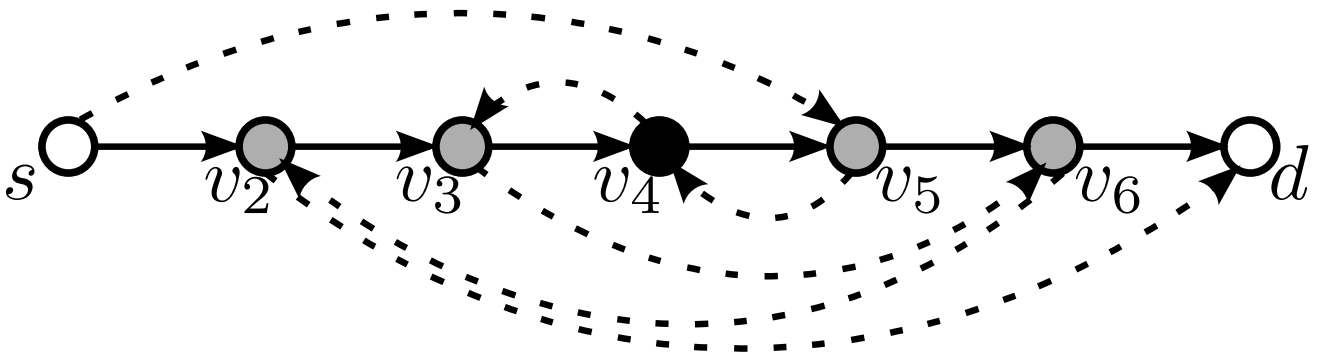
- Reverse:



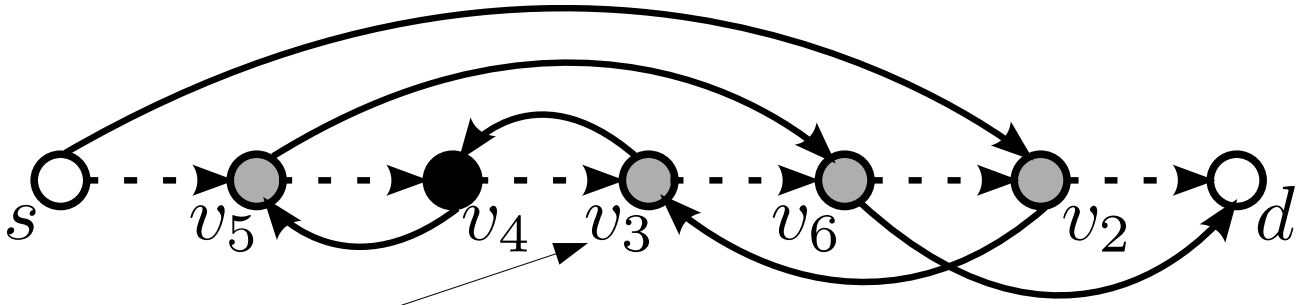
- FF, BB

Reversed update pattern

- Standard:



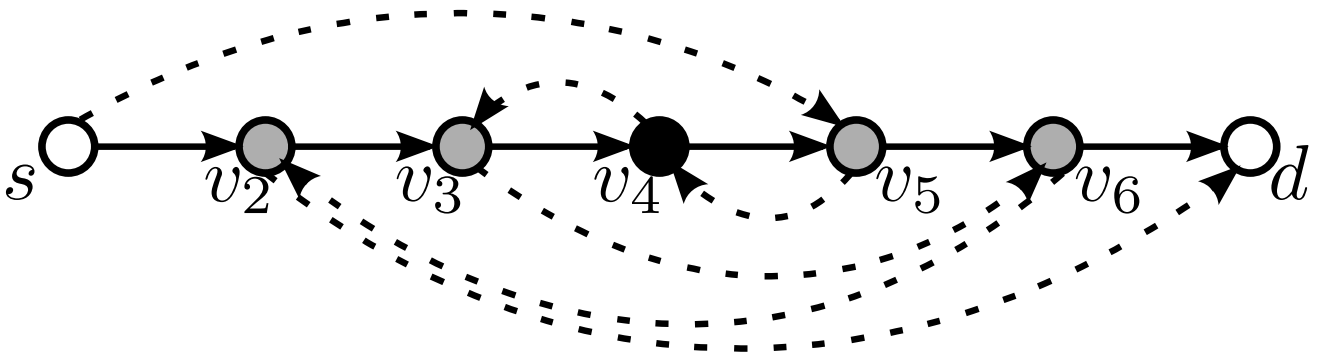
- Reverse:



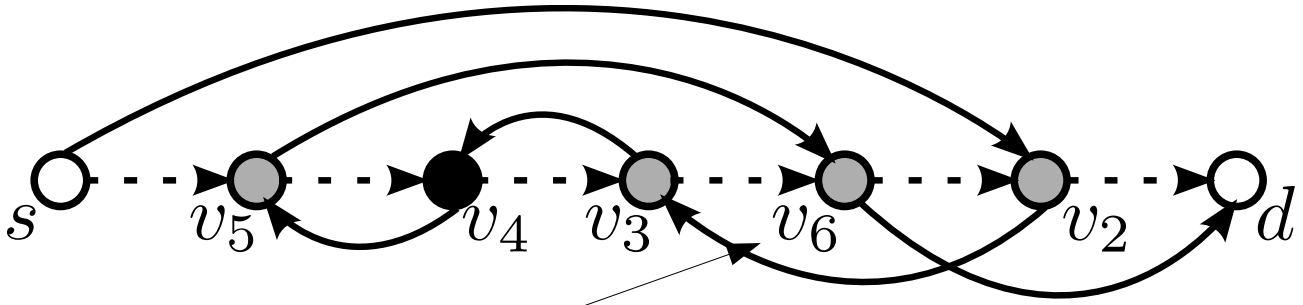
- FF, BB, FB

Reversed update pattern

- Standard:



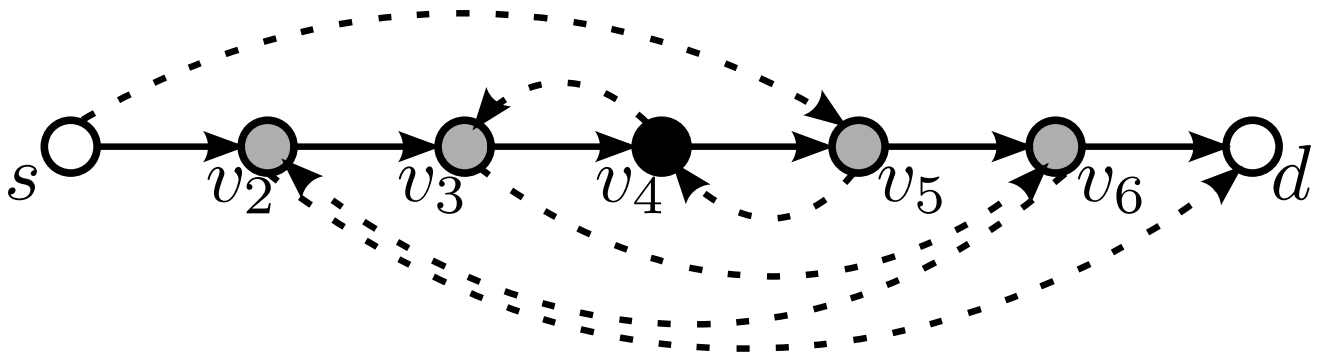
- Reverse:



- FF, BB, FB, BF

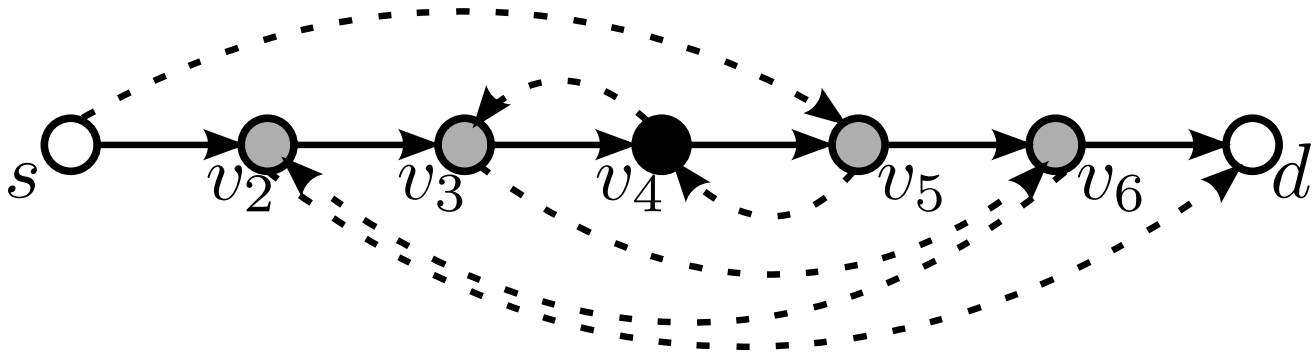
2 rounds is easy

- No BB nodes → 2 round schedule exists



What about 3 rounds?

- 3 rounds is hard!



3 rounds is hard

- Where to update FF nodes?
- 3-SAT reduction
- Creating the gadgets
- Connecting the gadgets

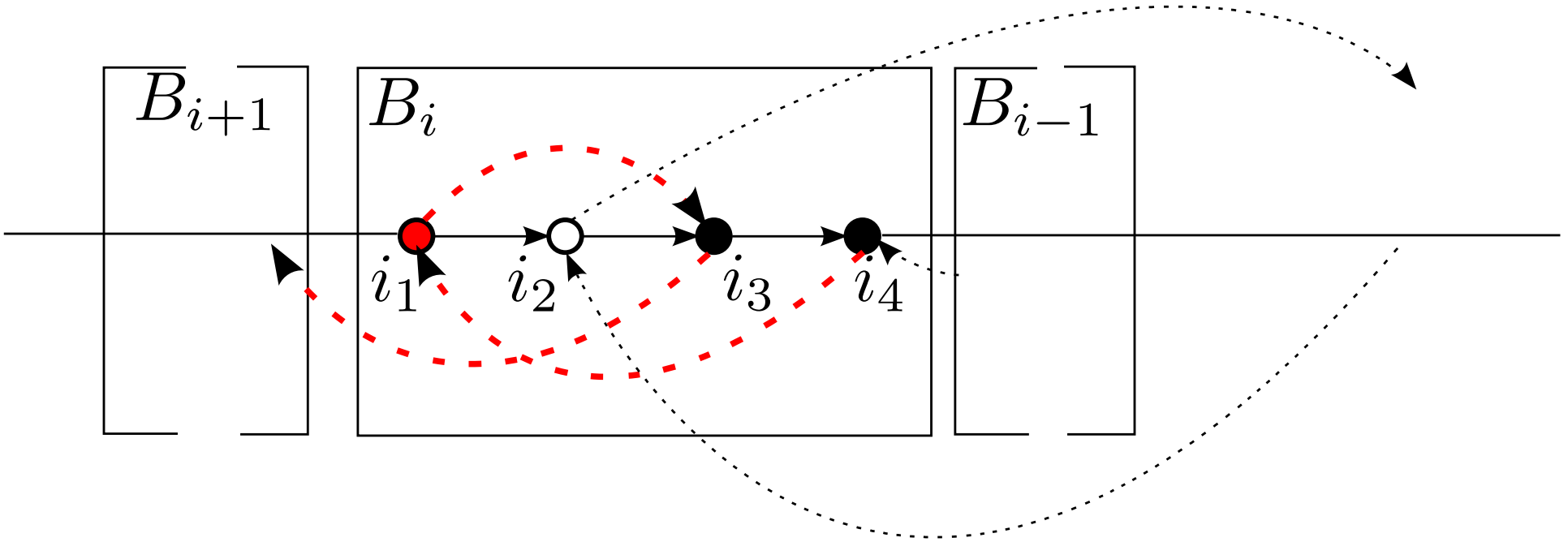
Where to update FF nodes?

- BB nodes updated in 2nd round
- FB nodes can be moved to 1st round
- BF nodes can be moved to 3rd round

- Where to update FF nodes?

Where to update FF nodes?

- Remember greedy!



3-SAT reduction

$$(x \vee \overline{y} \vee z) \wedge (x \vee w \vee \overline{v}) \wedge (\overline{x} \vee v \vee \overline{w})$$

- Create #X variables: $x_0, x_1, \ldots, x_l$
- Assignment clauses: $x_0 \vee \overline{x}_0$
- Implication clauses: $x_0 \rightarrow x_1$
- Exclusive clauses: $(\neg x_l \vee \neg \overline{x}_l)$

3-SAT reduction

$$(x \vee \overline{y} \vee z) \wedge (x \vee w \vee \overline{v}) \wedge (\overline{x} \vee v \vee \overline{w})$$

- Create #X variables: $x_0, x_1, \ldots, x_l$
- Assignment clauses: $x_0 \vee \overline{x}_0$
- Implication clauses: $x_0 \rightarrow x_1$
- Exclusive clauses: $(\neg x_l \vee \neg \overline{x}_l)$

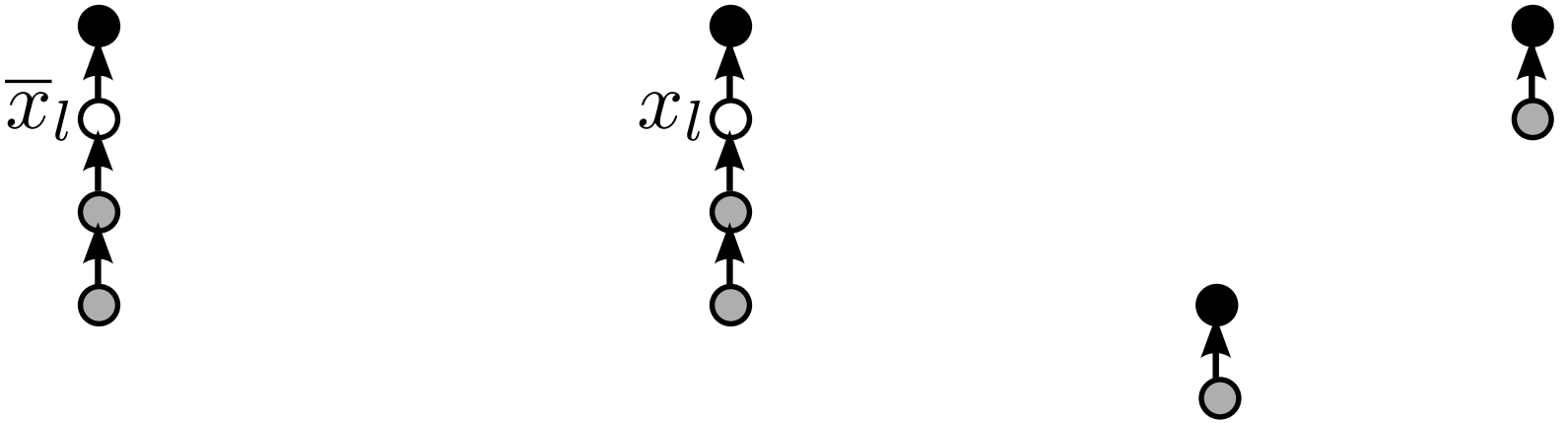
Positive evaluation → update in first round

Negative evaluation → update in third round

Exclusive clause (example)

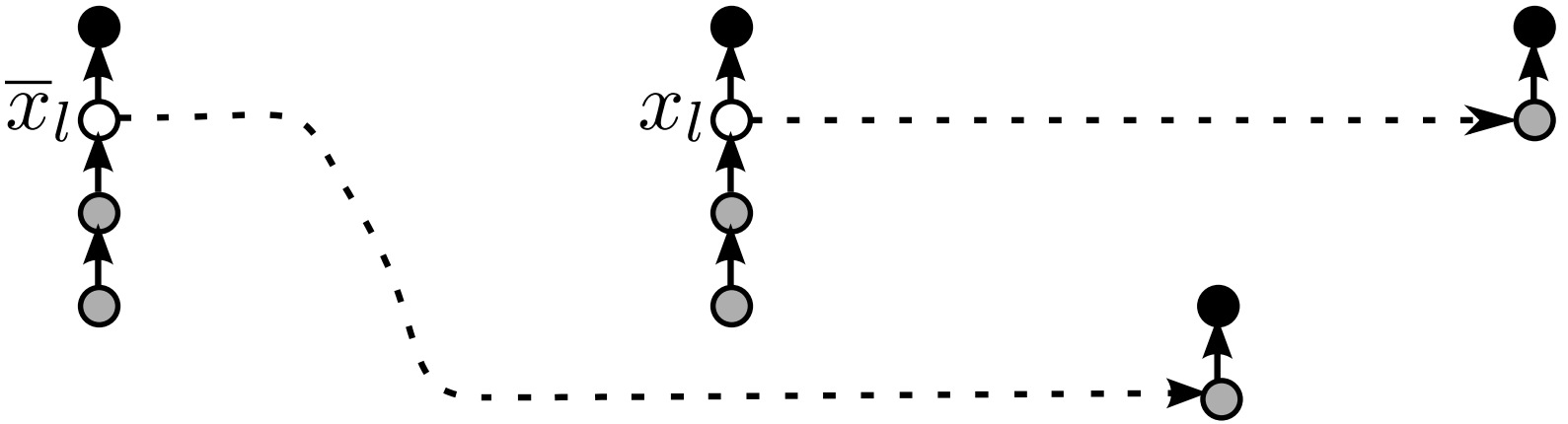
$\overline{x}_{l\bigcirc}$ $x_{l\bigcirc}$

Exclusive clause (example)



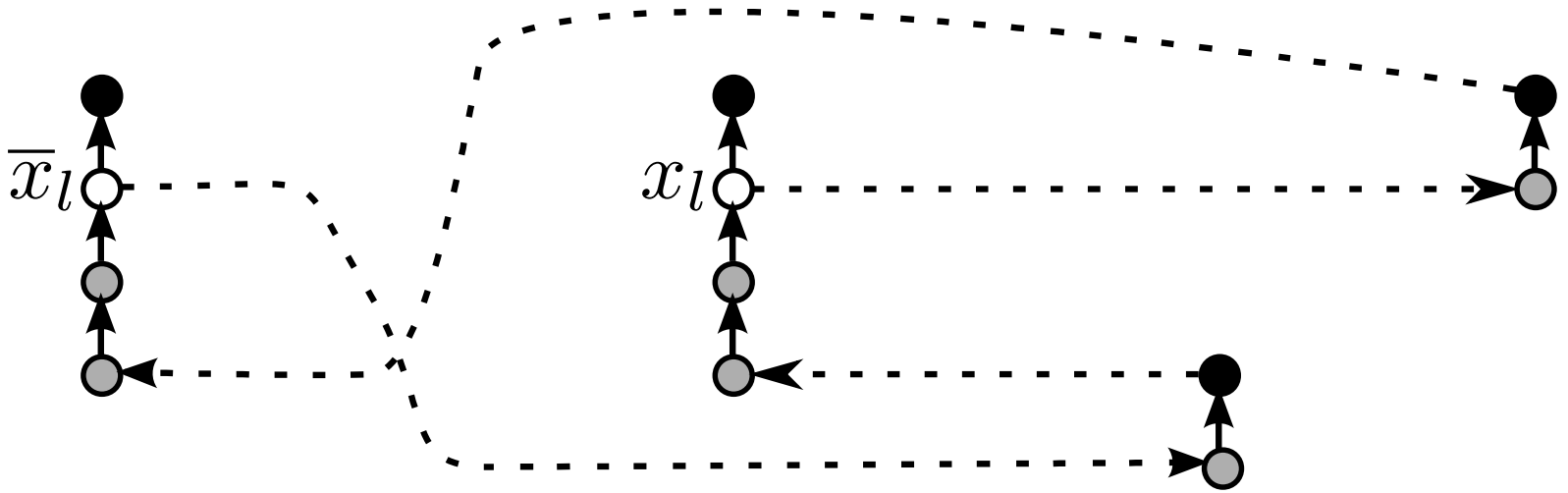
FF (white), BB (black), B* (grey)

Exclusive clause (example)



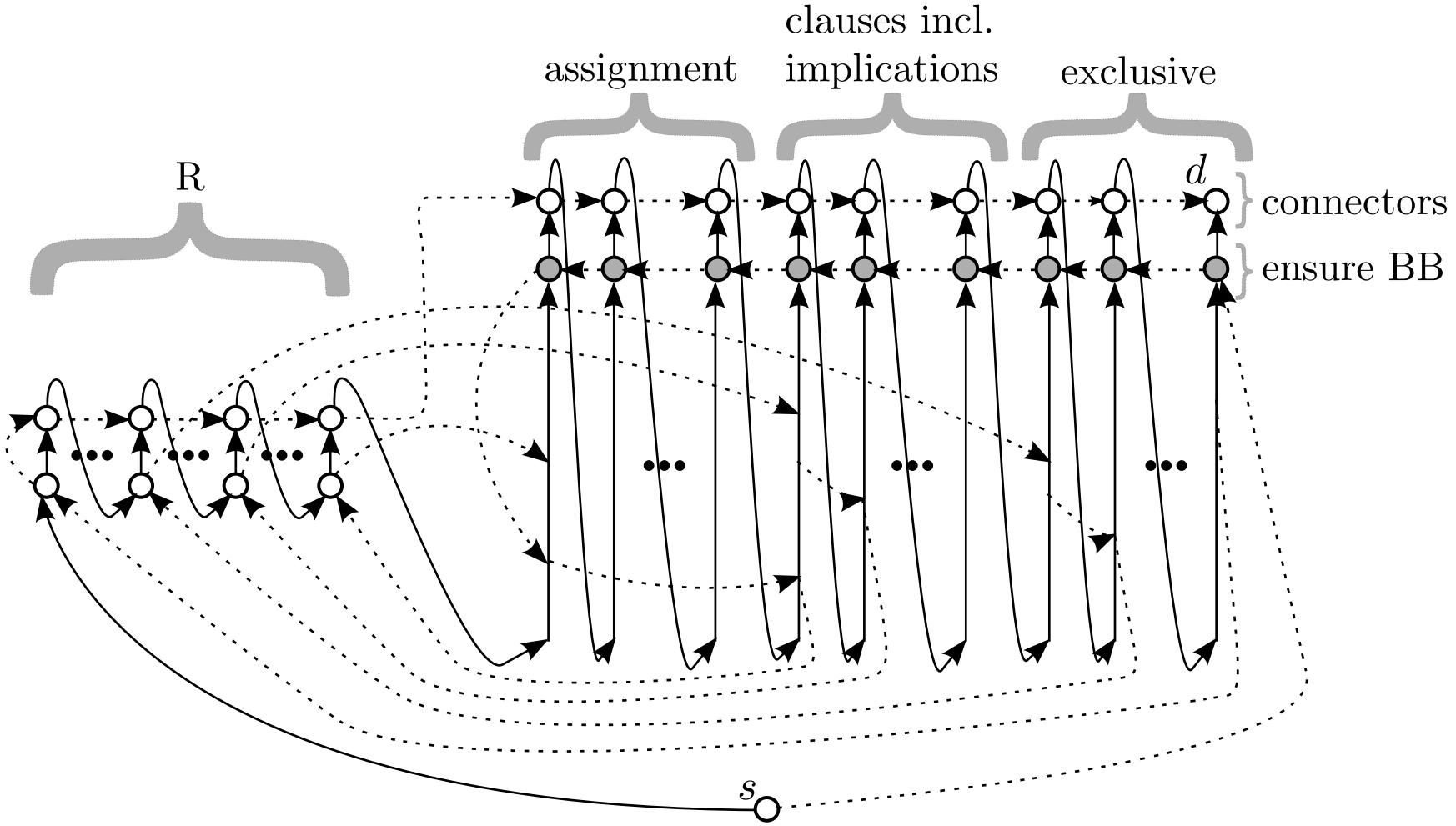
FF (white), BB (black), B* (grey)

Exclusive clause (example)

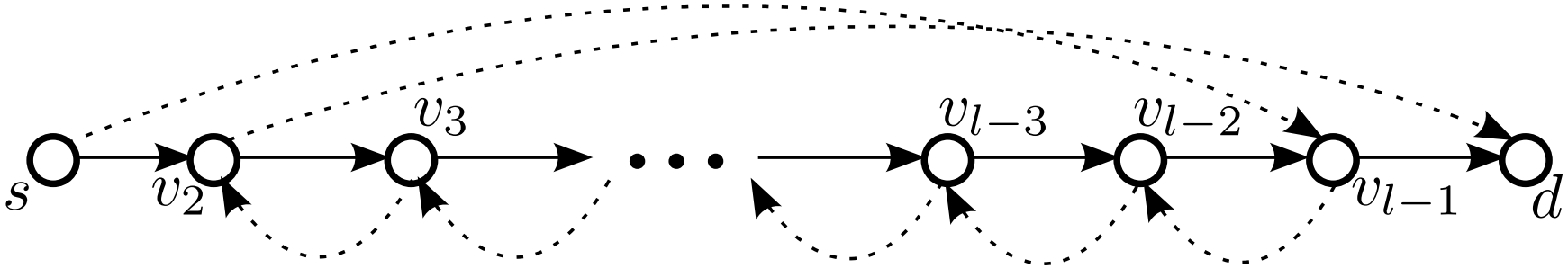


FF (white), BB (black), B* (grey)

Big picture

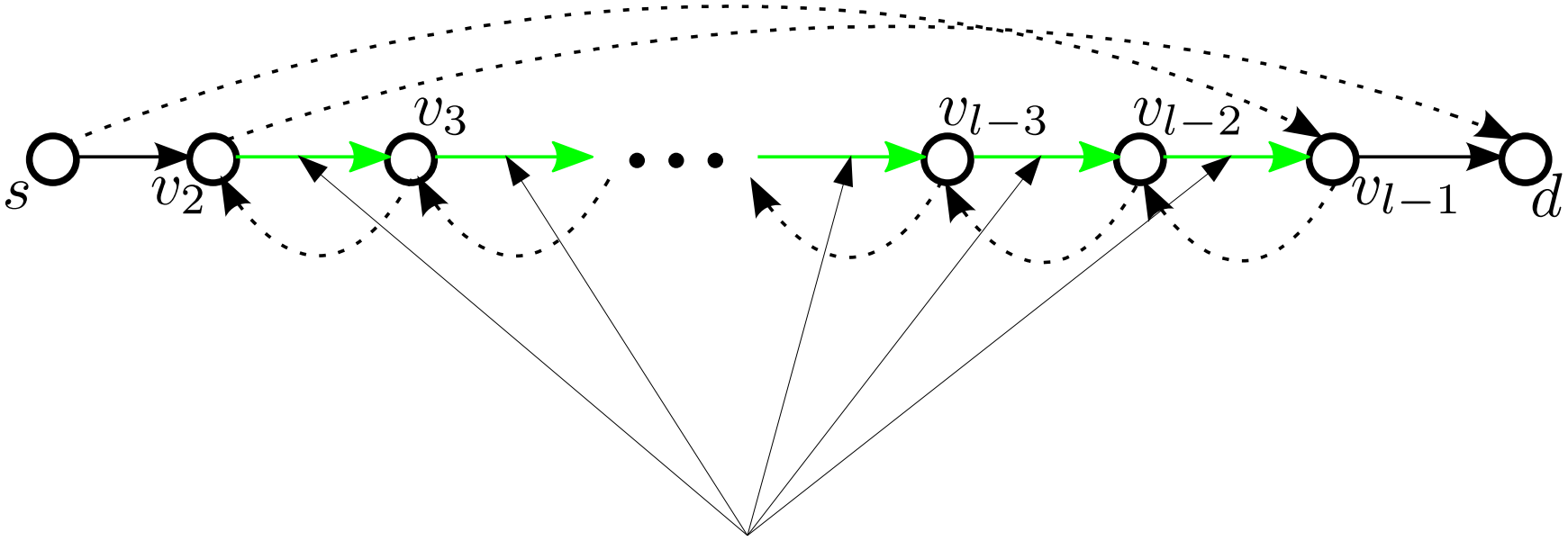

assignment
clauses incl.
implications
exclusive
R
d
connectors
ensure BB
s

Strong loop-freedom: worst case



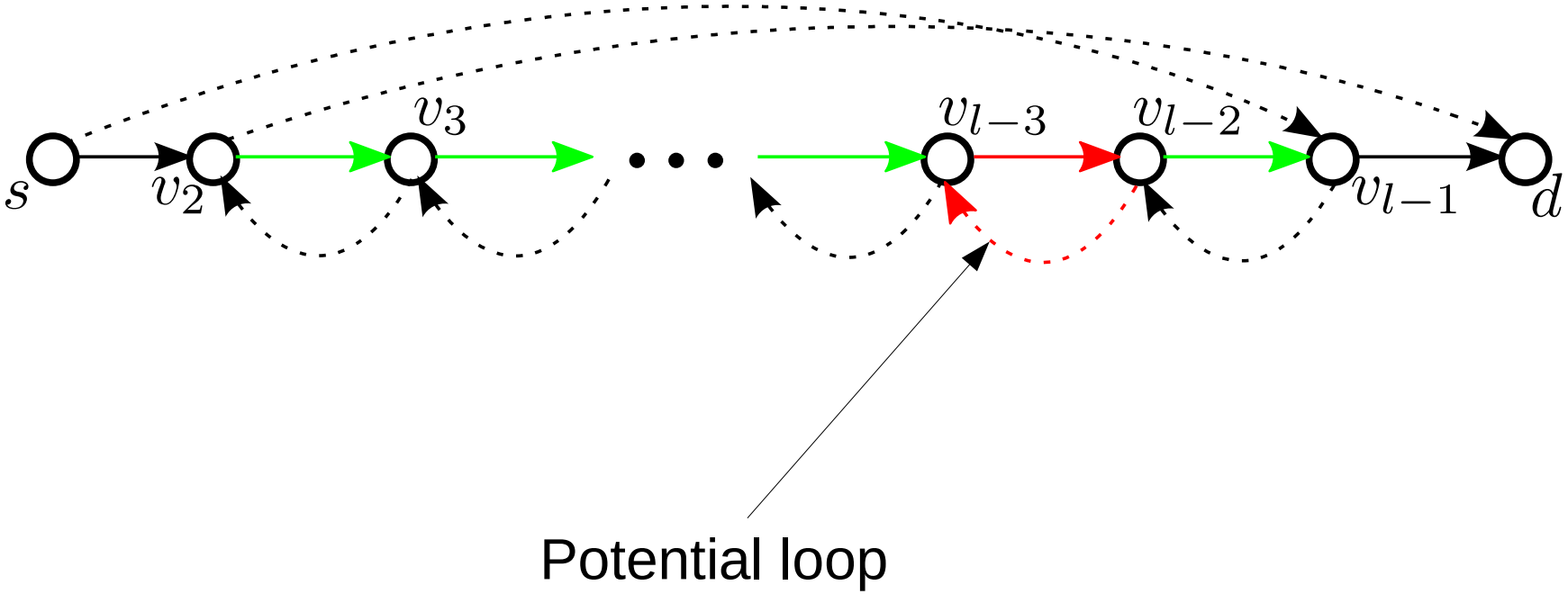
Strong loop-freedom: worst case

- Update s, v_2 in first round



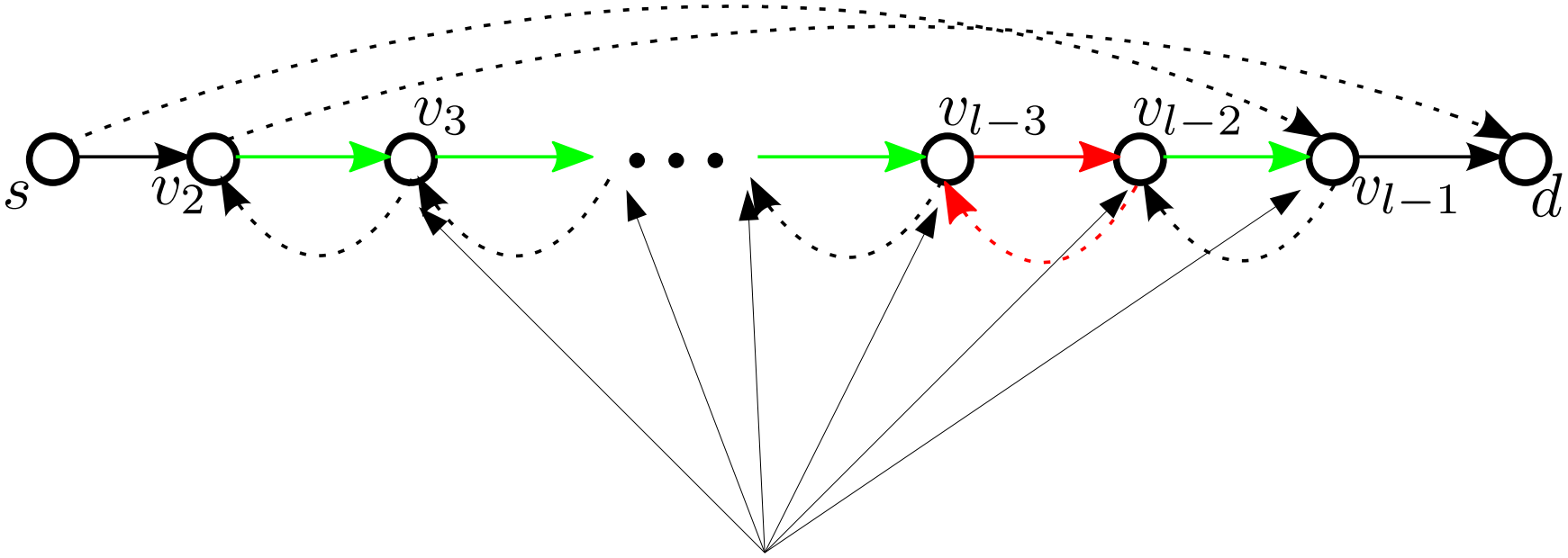
Strong loop-freedom: worst case

- Update s, v_2 in first round



Strong loop-freedom: worst case

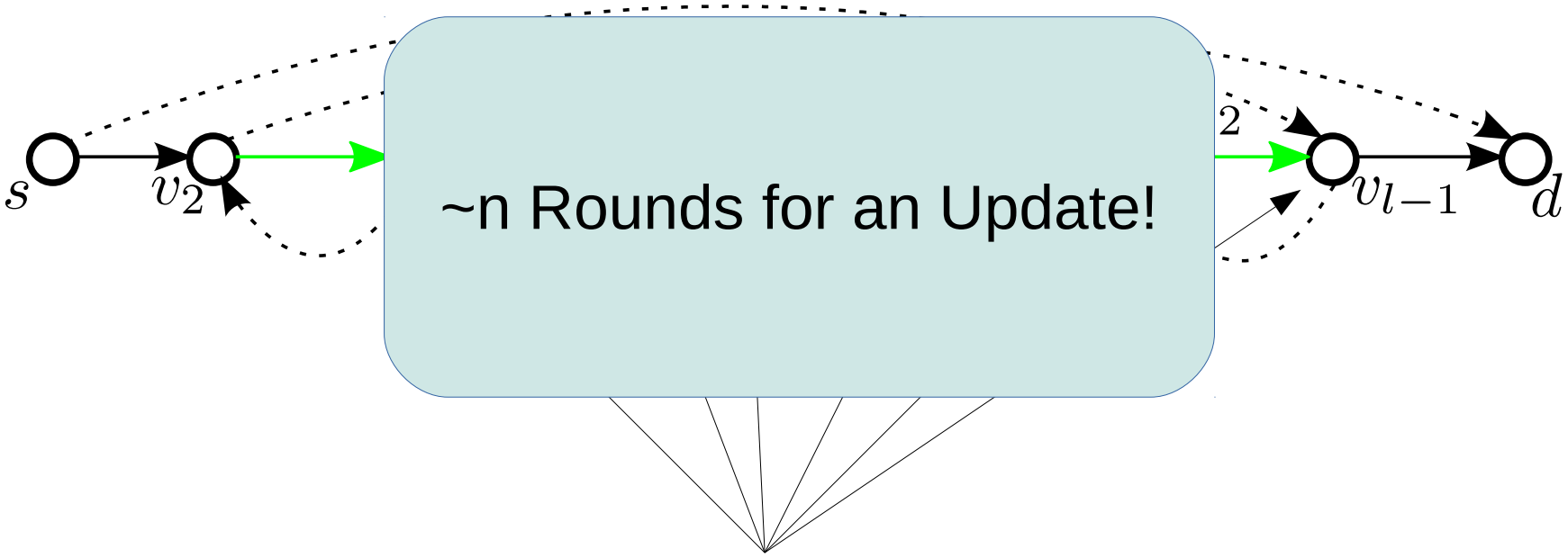
- Update s, v_2 in first round



Update all nodes
one after the other

Strong loop-freedom: worst case

- Update s, v_2 in first round



Bad news so far! Time to relax.

- Prevent topological loops
  - NP hard for 3 rounds
  - Some instances need O(n) rounds

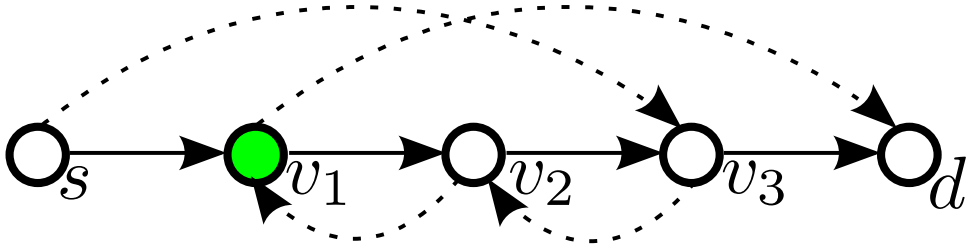
Bad news so far! Time to relax.

- Prevent topological loops
  - NP hard for 3 rounds
  - Some instances need O(n) rounds
- Relaxed loop freedom
  - Practical relevance only on path between s and d
  - Fast to compute
  - O(log n) rounds for every instance

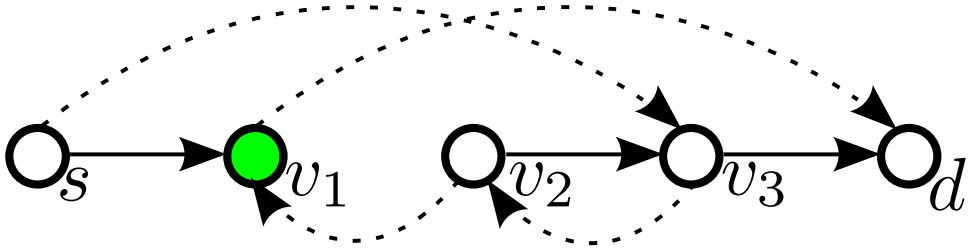
Peacock

- Shortcut
  - Reduce the distance between s and d
    - → Pick longest ranging forward edges
- Prune
  - Reduce the number of remaining nodes
    - → Update every node which is not on the s-d path

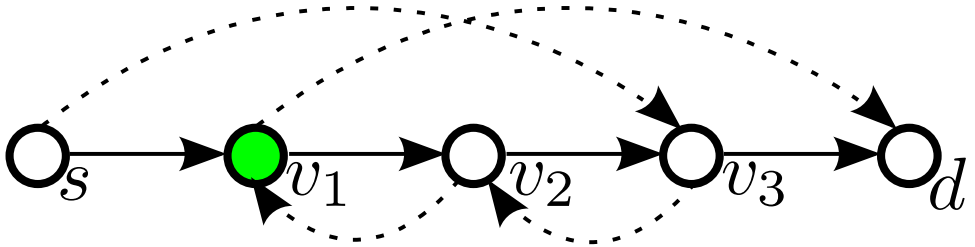
Logical reduction



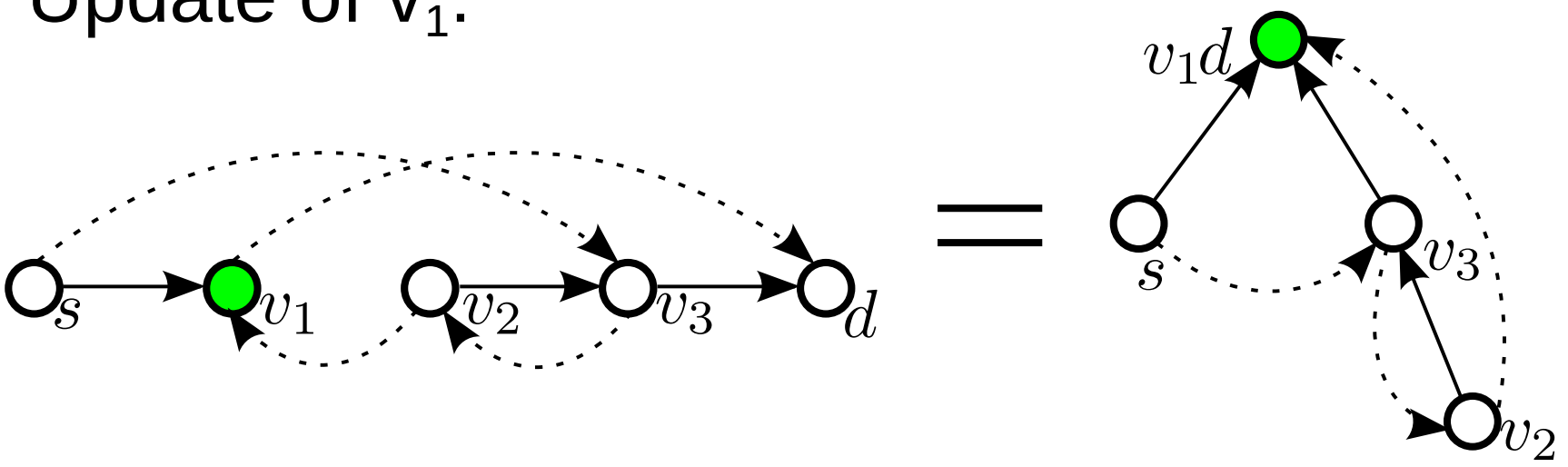
- Update of v_1:



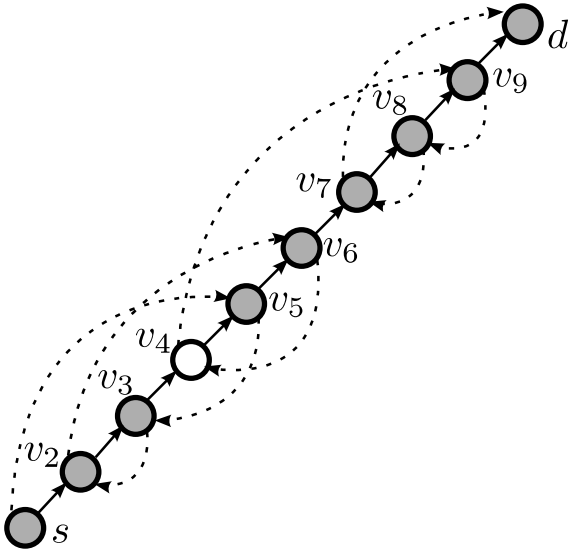
Logical reduction



- Update of v_1:



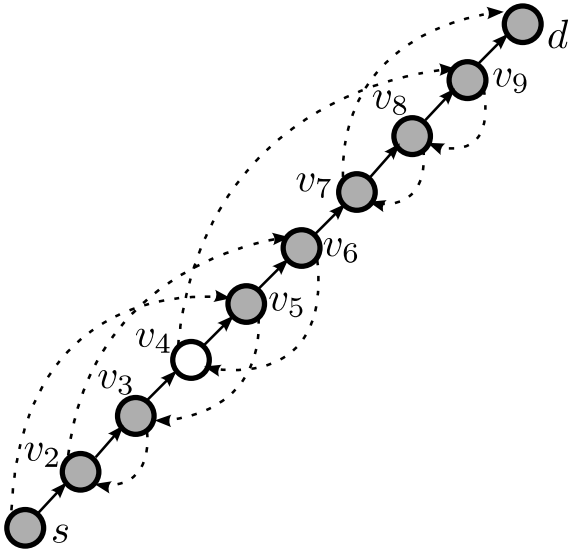
Peacock



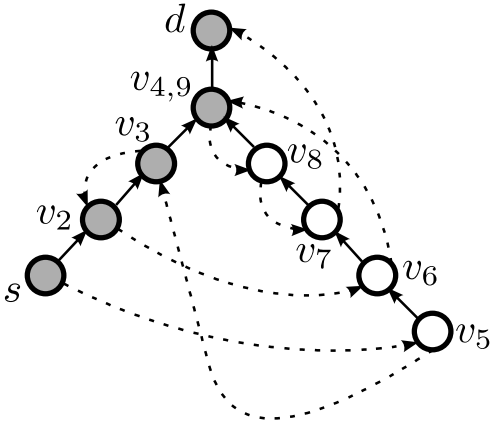
round: 1

Shortcut

Peacock



round: 1

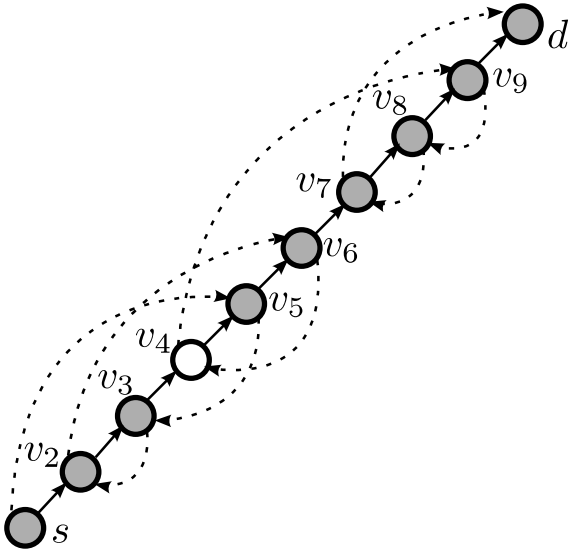


round: 2

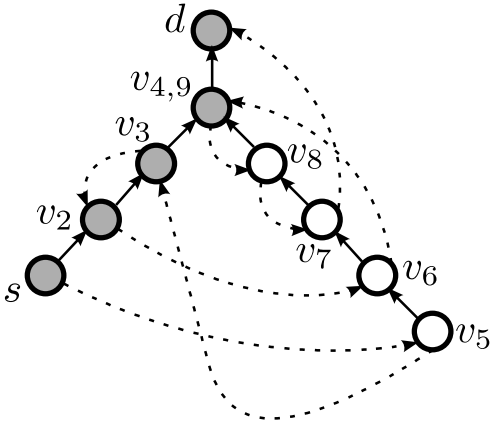
Shortcut

Prune

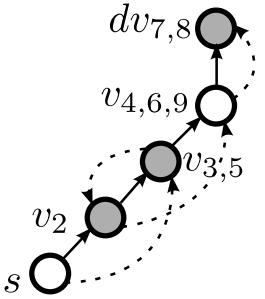
Peacock



round: 1



round: 2



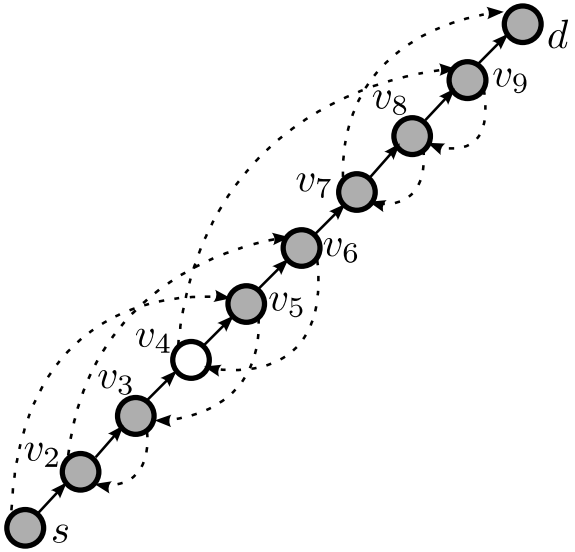
round: 3

Shortcut

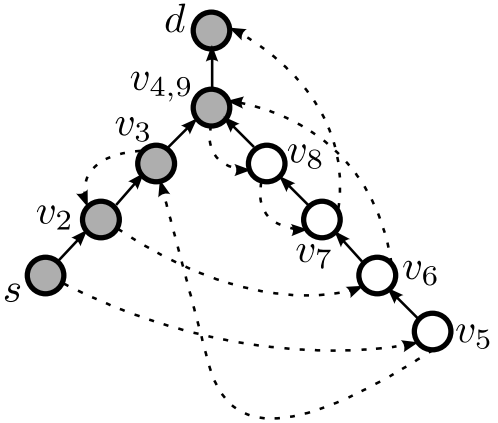
Prune

Shortcut

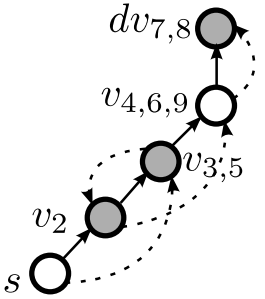
Peacock



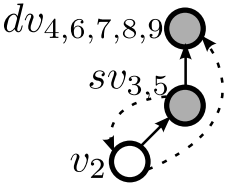
round: 1



round: 2



round: 3



round: 4

Shortcut

Prune

Shortcut

Prune

Conclusion

- SDN introduces interesting algorithmic questions
- Strong LF:
  - Greedy arbitrarily bad (up to n rounds)
  - 2 rounds easy
  - 3 rounds hard
- Introduction of Relaxed LF:
  - Peacock solves any scenario in O(log n) rounds
  - Computational results indicate the #rounds grow
- Most related work: (Ludwig et al. HotNets'14, Mahajan et al. HotNets'13)