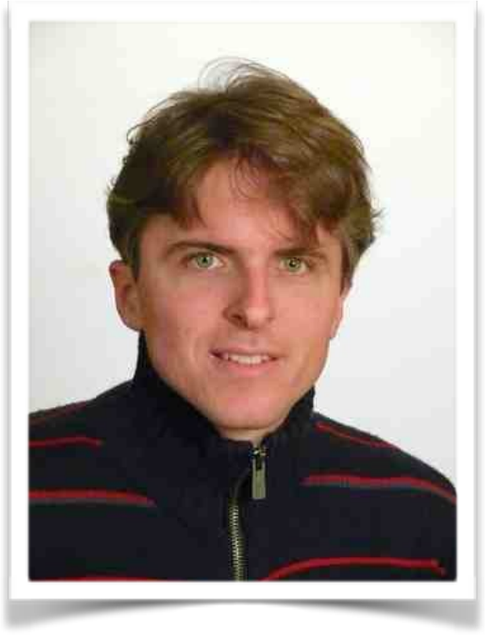
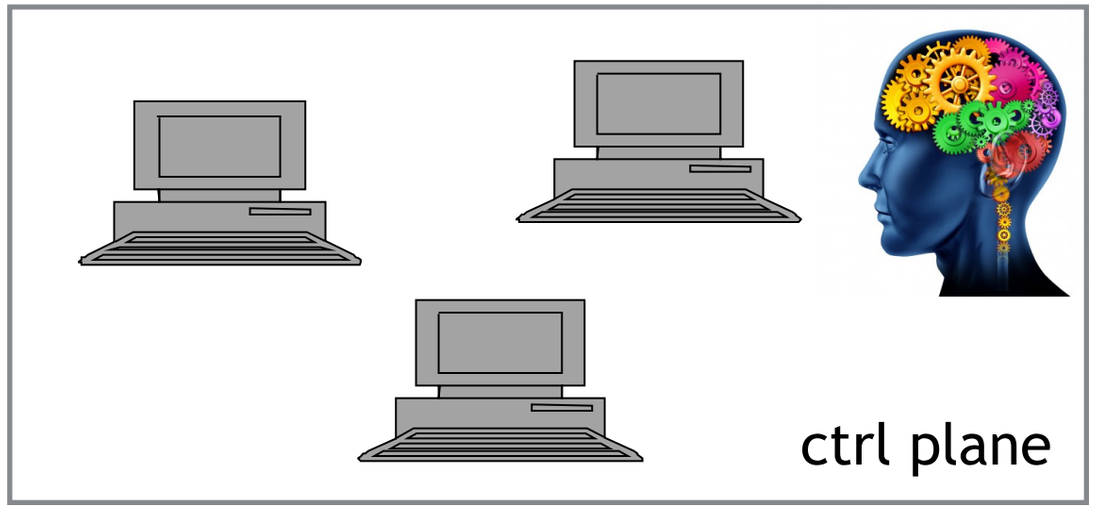
Reclaiming the Brain:
Useful OpenFlow Functions in the Data Plane

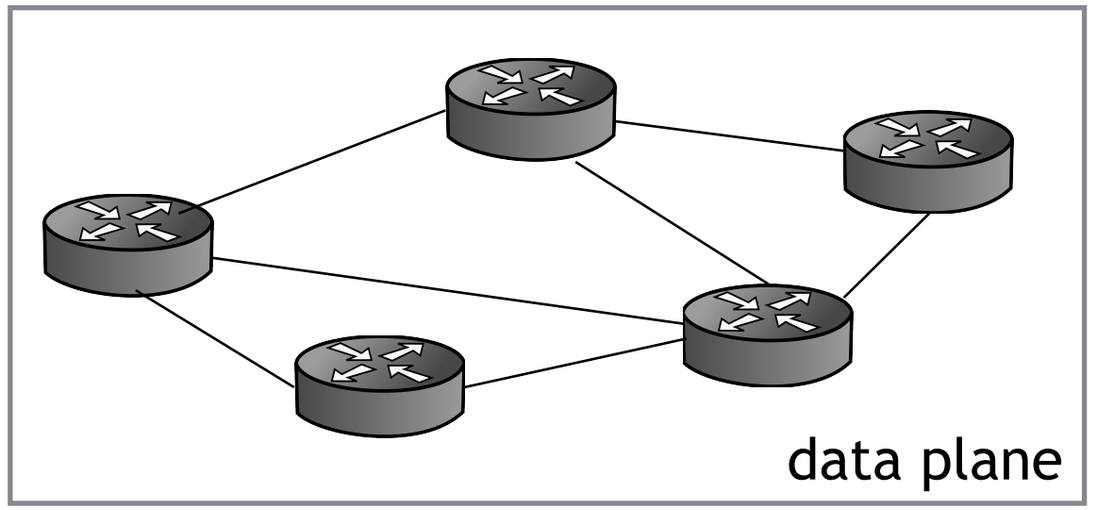
Liron Schiff (Tel Aviv Uni, Israel)
Michael Borokhovich (UT Austin, United States)
Stefan Schmid (TU Berlin & T-Labs, Germany)



My Talk in One Slide

* Separation of the planes enables:
  * simplified network management and operation
  * faster innovation



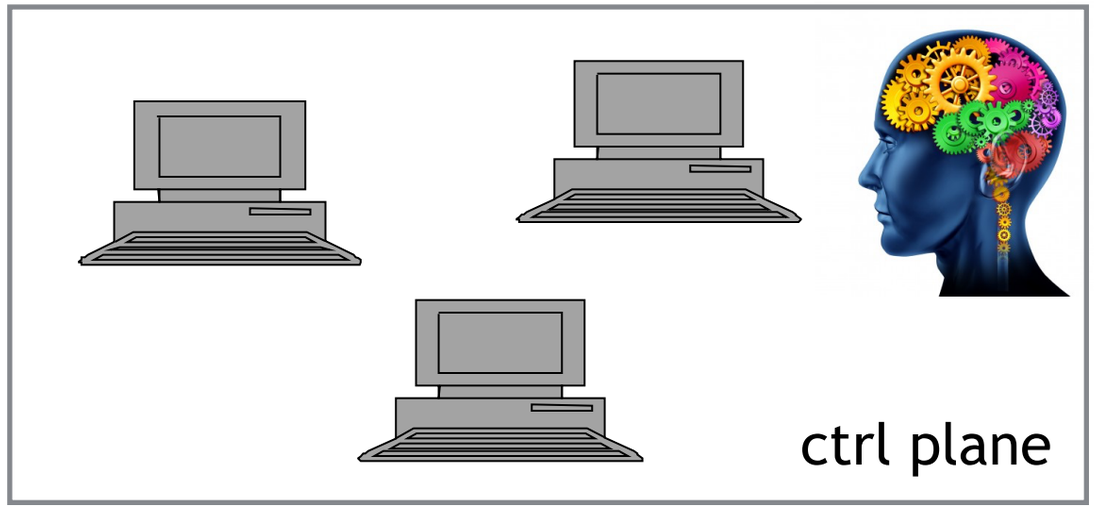


N

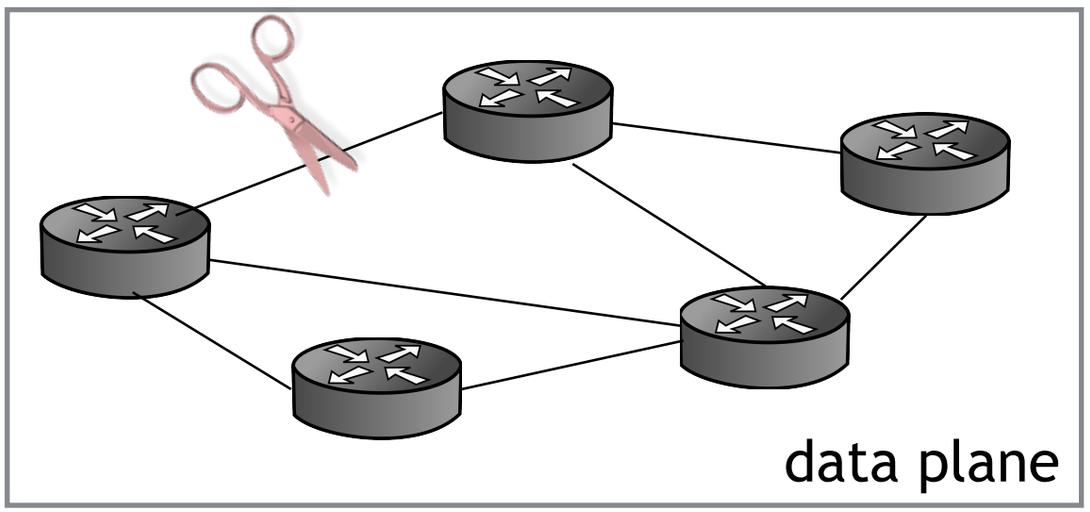
S

My Talk in One Slide

* Separation of the planes enables:
  * simplified network management and operation
  * faster innovation

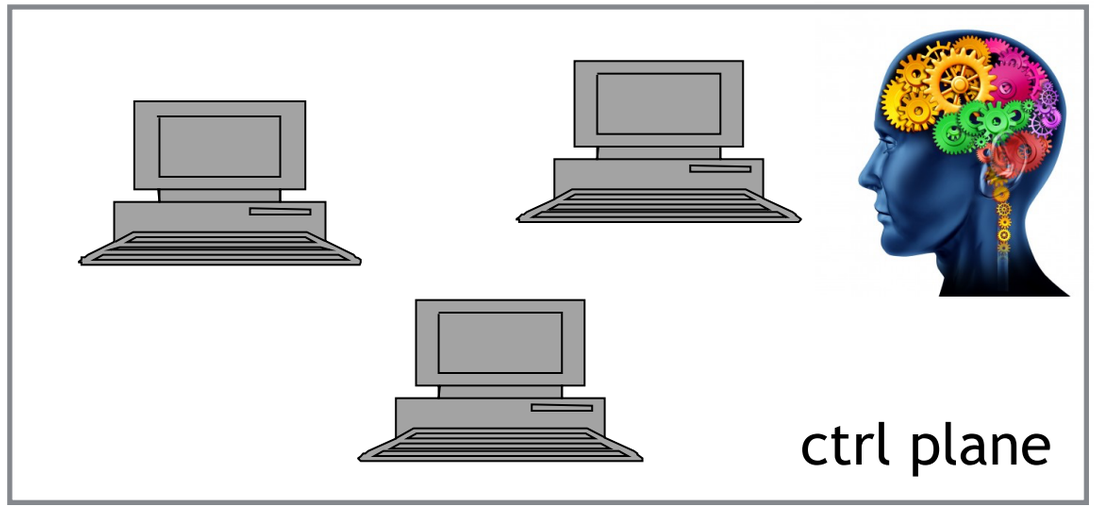


* **However:**
  * controller may miss certain data plane events
  * indirection => latency

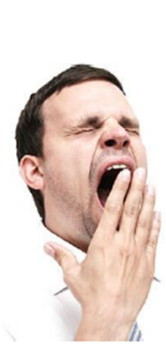


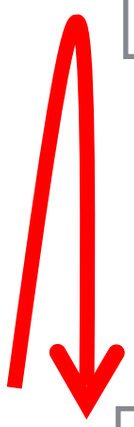
My Talk in One Slide

* Separation of the planes enables:
  * simplified network management and operation
  * faster innovation

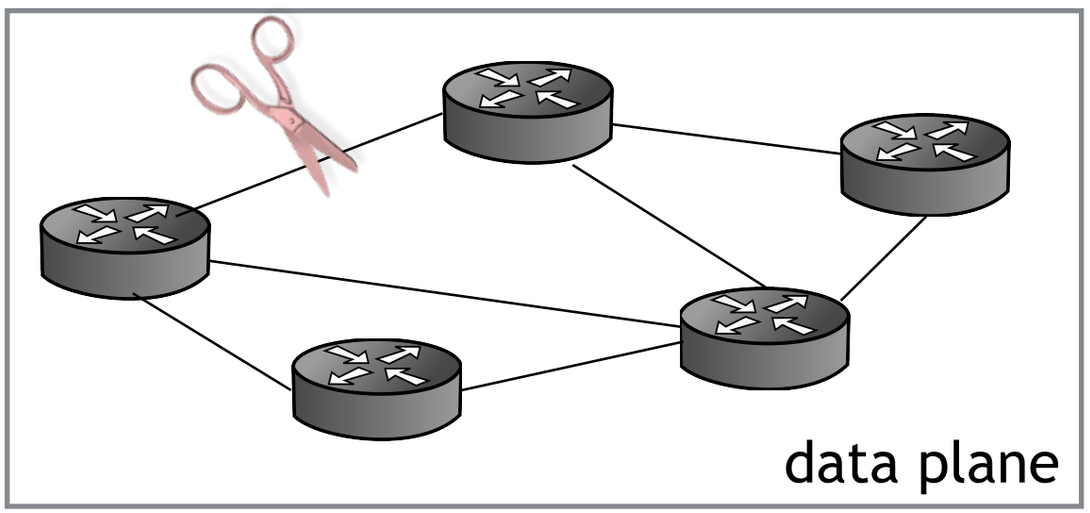


* **However:**
  * controller may miss certain data plane events
  * indirection => latency



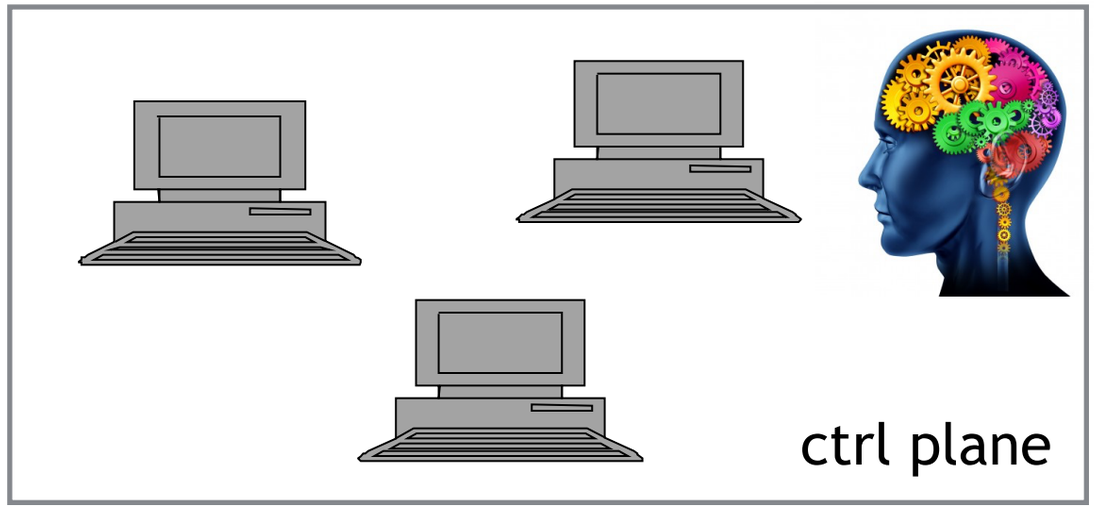


* **What functionality should be kept in data plane? — A big question!**

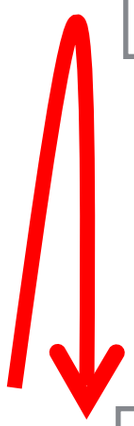


My Talk in One Slide

* Separation of the planes enables:
  * simplified network management and operation
  * faster innovation

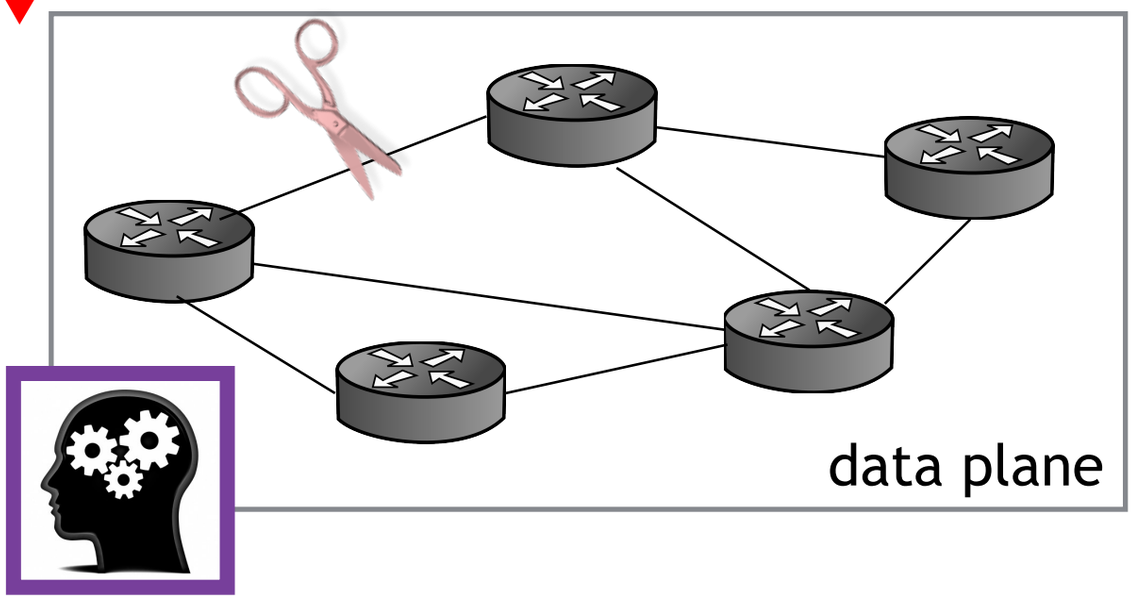


* **However:**
  * controller may miss certain data plane events
  * indirection => latency

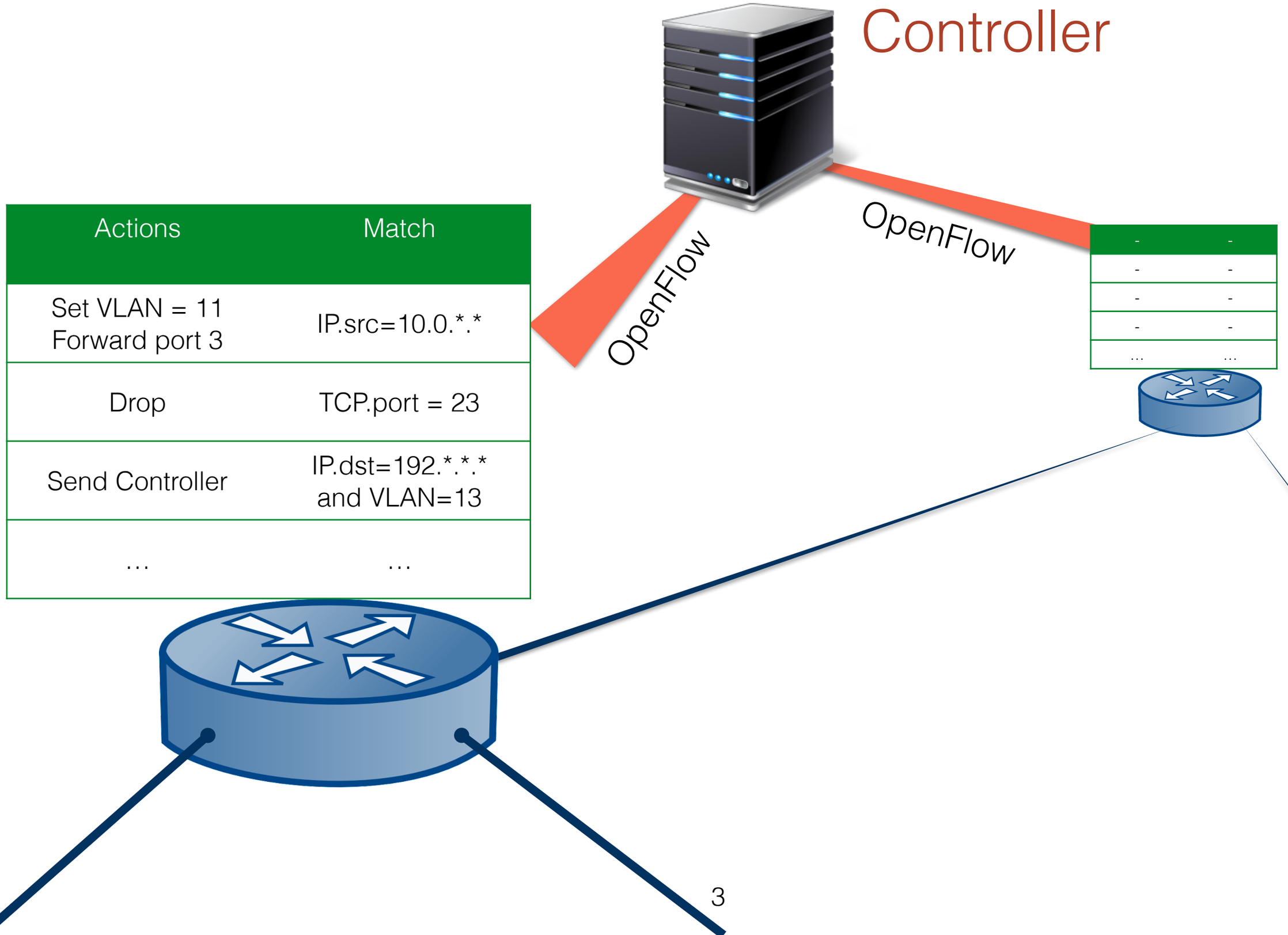


* **What functionality should be kept in data plane? — A big question!**

In this talk: example of functions that can be kept in the data plane.

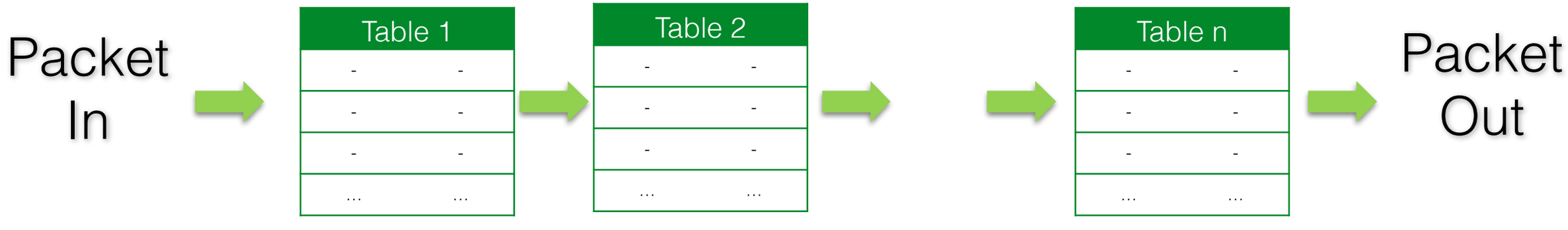


SDN and OpenFlow



OpenFlow in a Nutshell

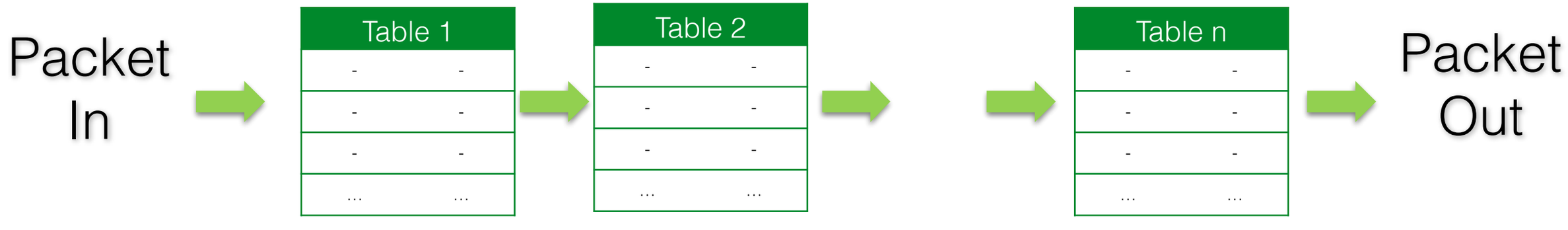
* Switch pipeline



* Basic Actions
  * Set a field
  * Append a label
  * Forward to a port/controller/flood
  * Goto Table x

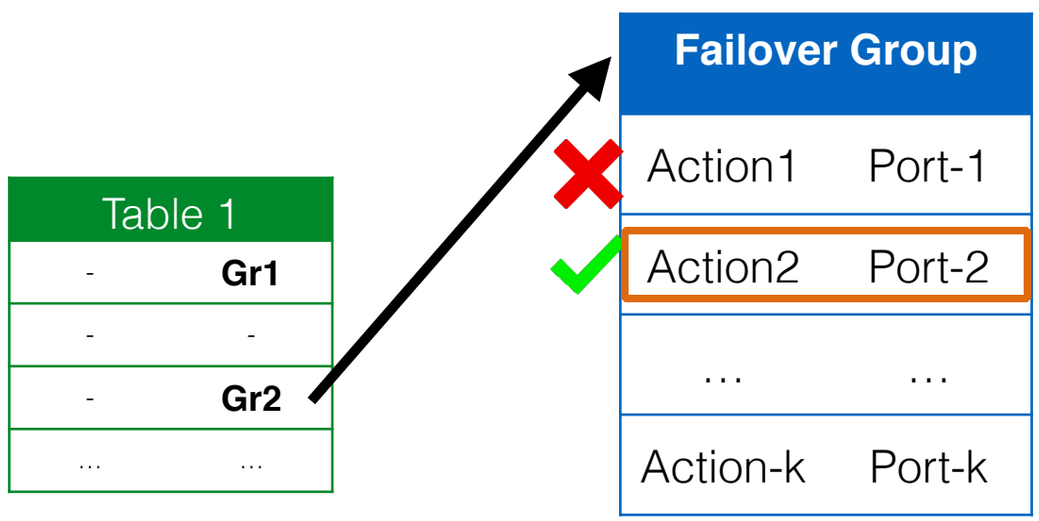
OpenFlow in a Nutshell

* Switch pipeline



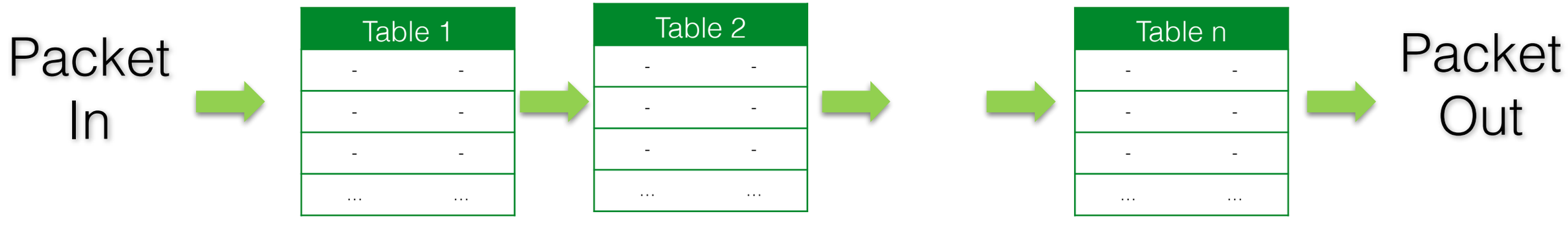
* Basic Actions
  * Set a field
  * Append a label
  * Forward to a port/controller/flood
  * Goto Table x

* Advanced/optional Actions
  * Link state based
  * Round-robin selection



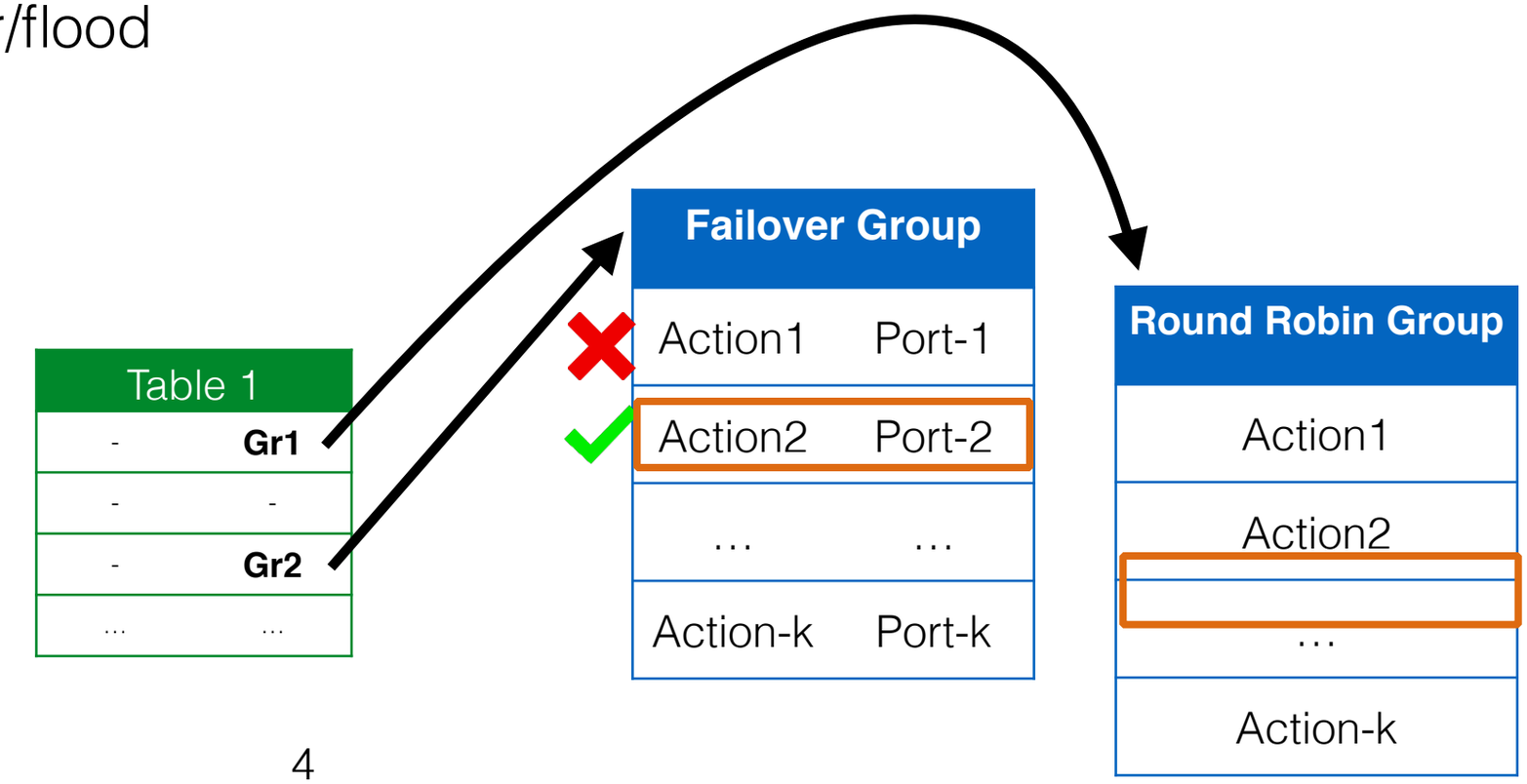
OpenFlow in a Nutshell

* Switch pipeline



* Basic Actions
  * Set a field
  * Append a label
  * Forward to a port/controller/flood
  * Goto Table x

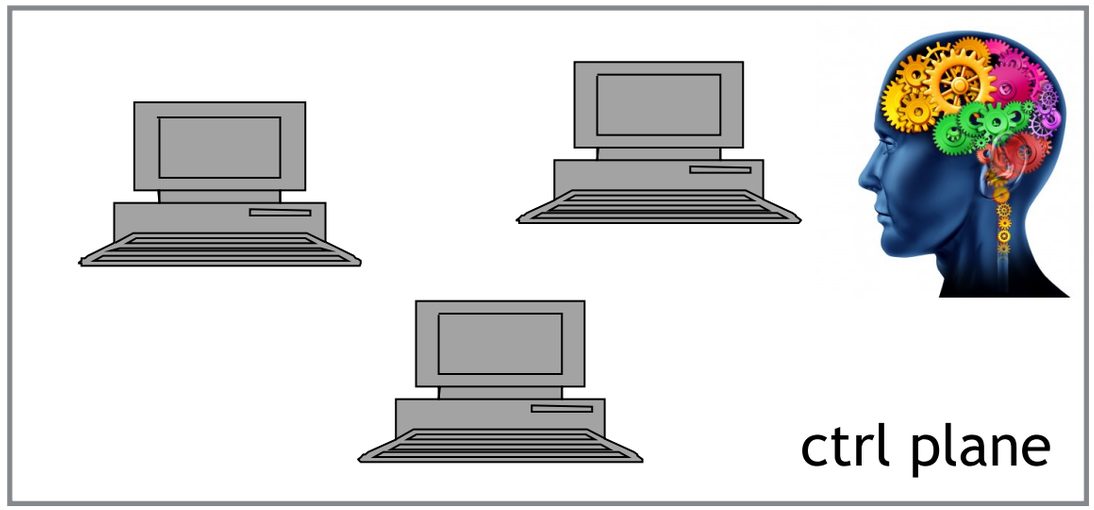
* Advanced/optional Actions
  * Link state based
  * Round-robin selection

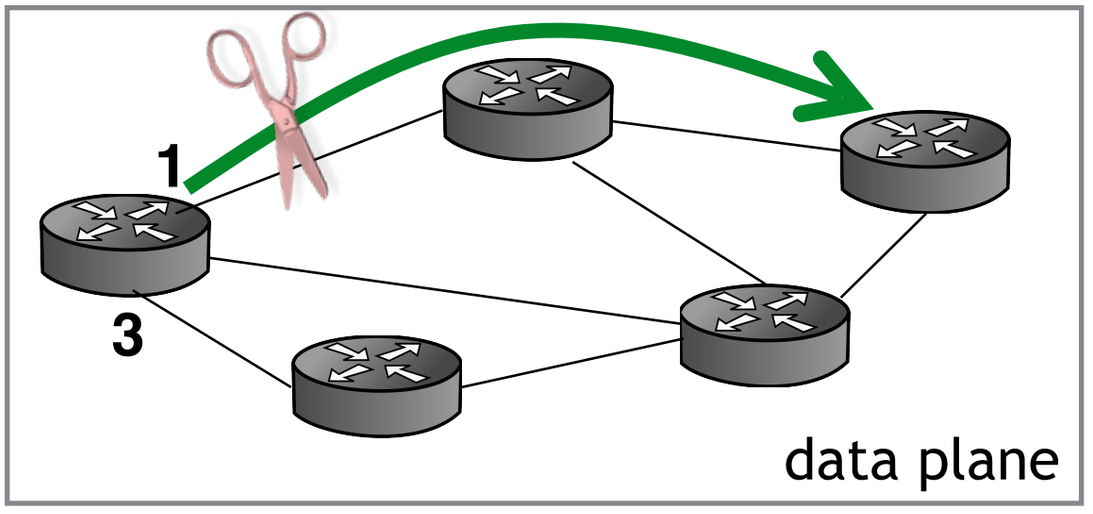


Fast Failover - Adding Brains to the South

* Proactive reaction to link failures

Failover Group	
Fwd	Port **1**
Fwd	Port **3**
…	…
Action-k	Port-k

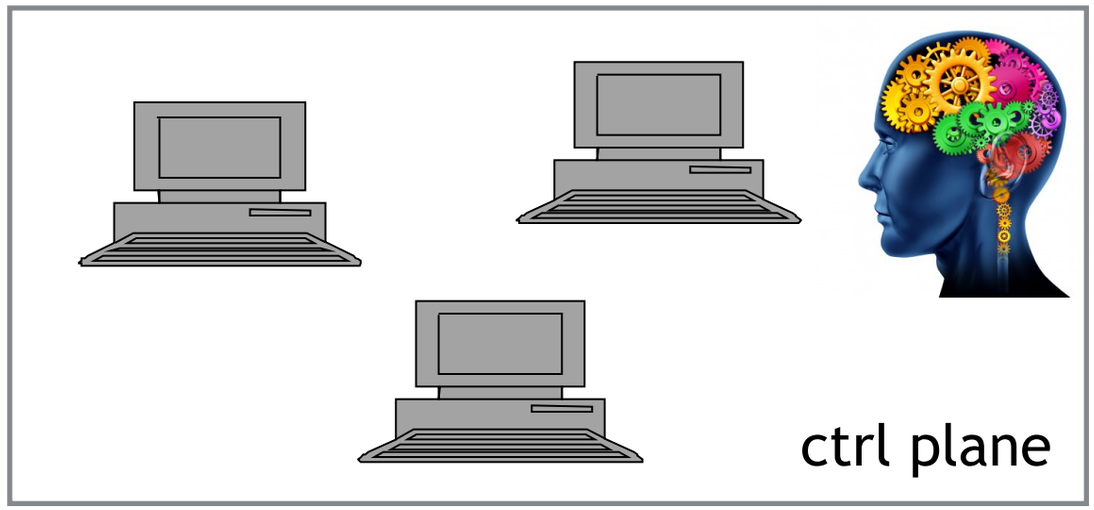


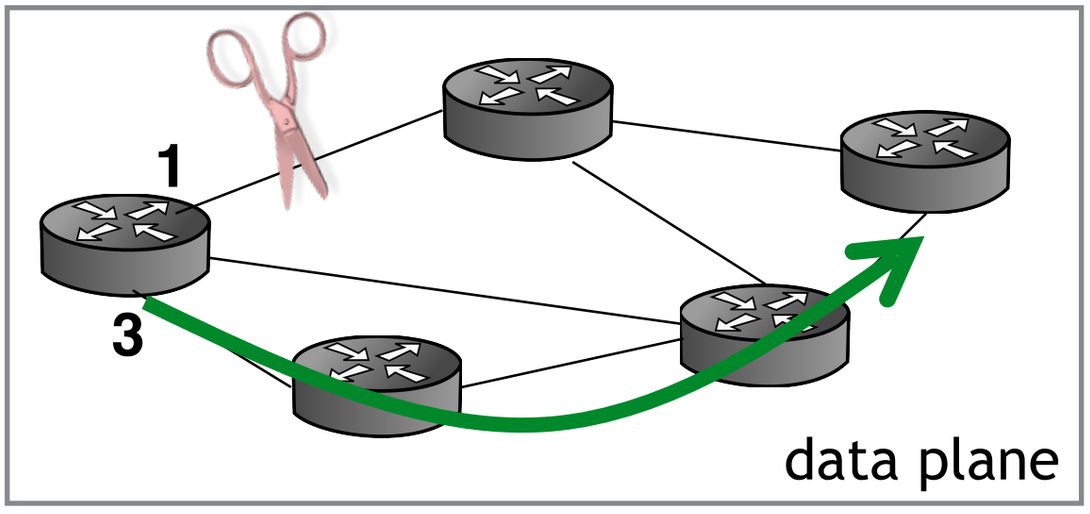


Fast Failover - Adding Brains to the South

* Proactive reaction to link failures

Failover Group	
Fwd	Port **1**
Fwd	Port **3**
…	…
Action-k	Port-k

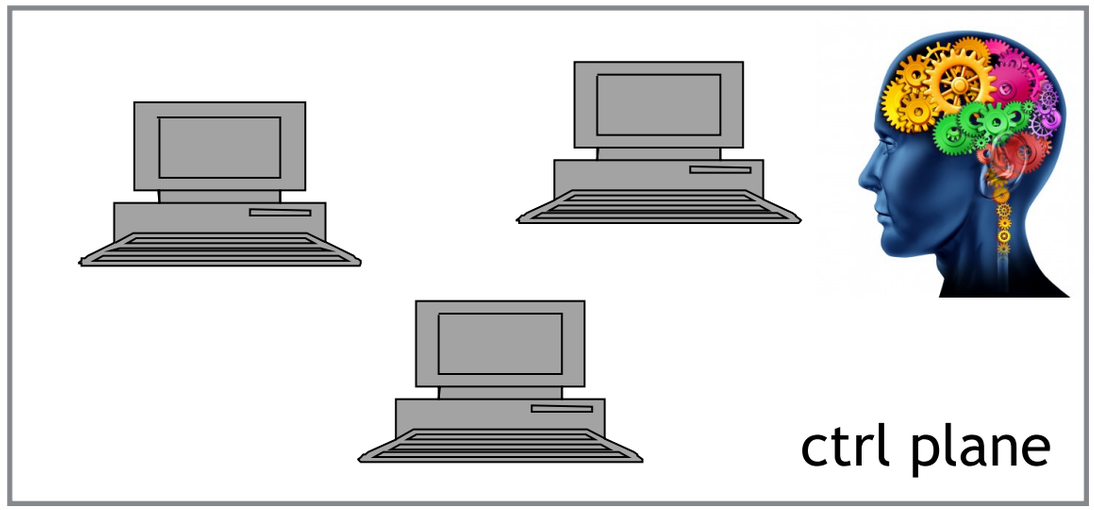


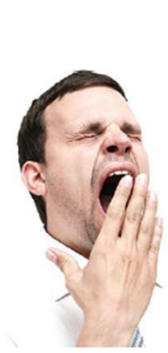


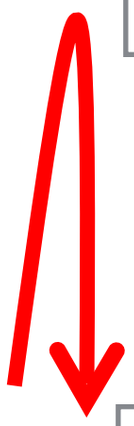
Fast Failover - Adding Brains to the South

* Proactive reaction to link failures

Failover Group	
Fwd	Port **1**
Fwd	Port **3**
…	…
Action-k	Port-k



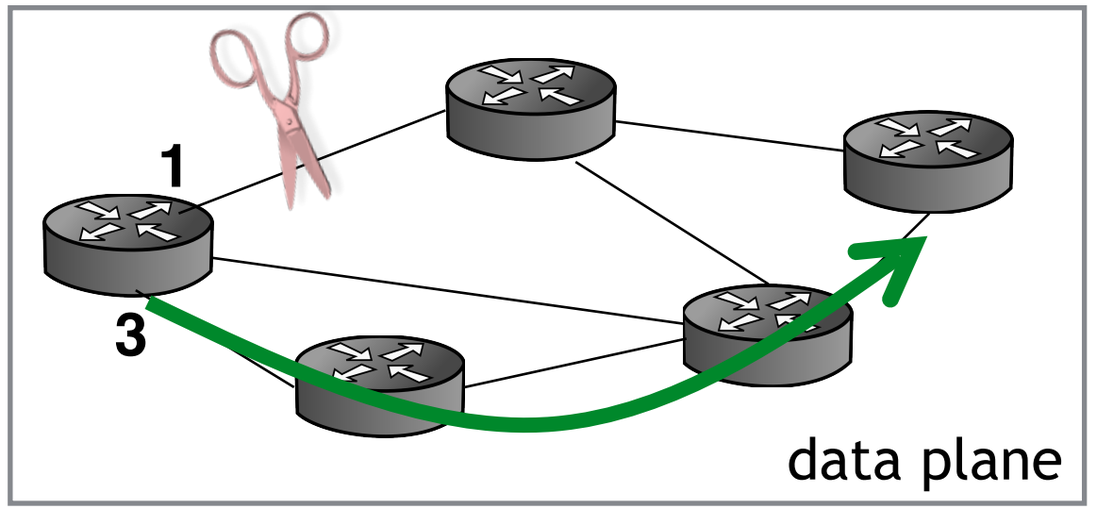




N

S

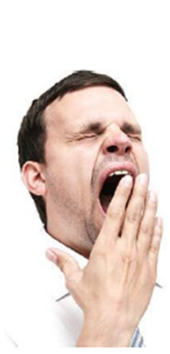
* Non-trivial to use
  * May quickly introduce loops
  * May introduce high load
* Much better with Tags

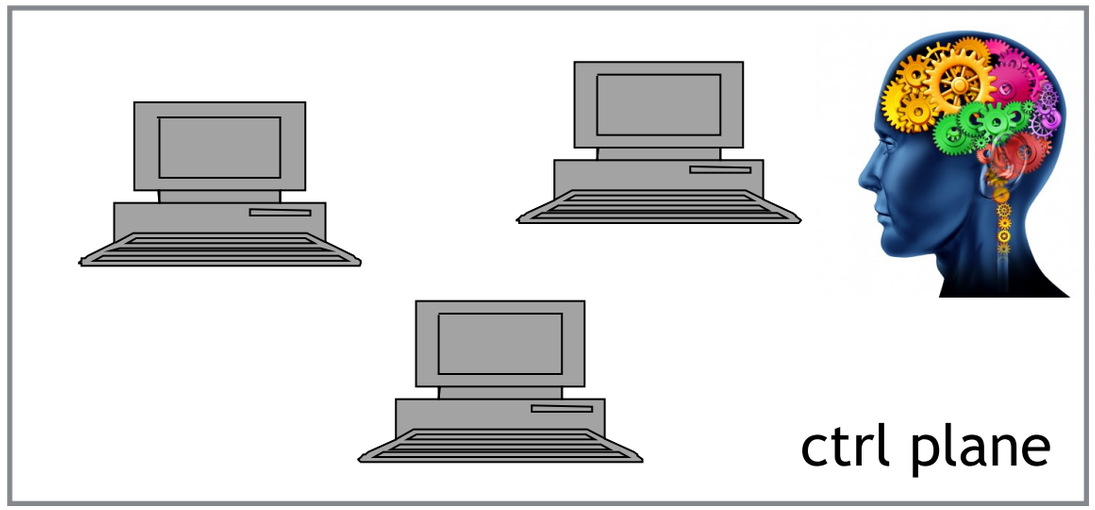


Fast Failover - Adding Brains to the South

* Proactive reaction to link failures

Failover Group	
Fwd	Port **1**
Fwd	Port **3**
…	…
Action-k	Port-k

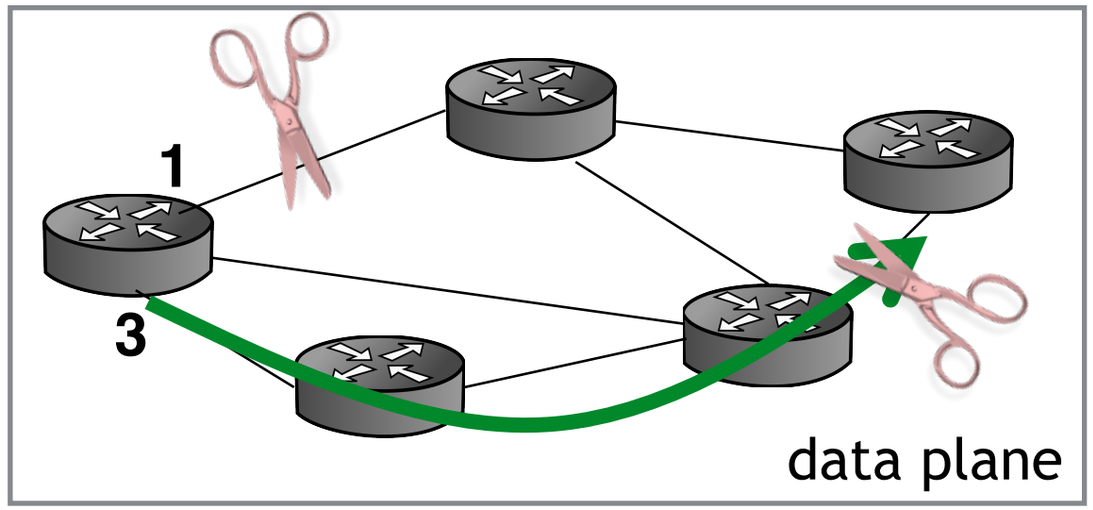




N

S

* Non-trivial to use
  * May quickly introduce loops
  * May introduce high load
* Much better with Tags

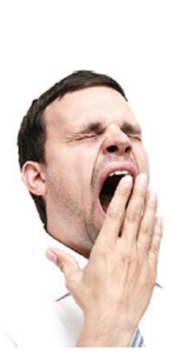


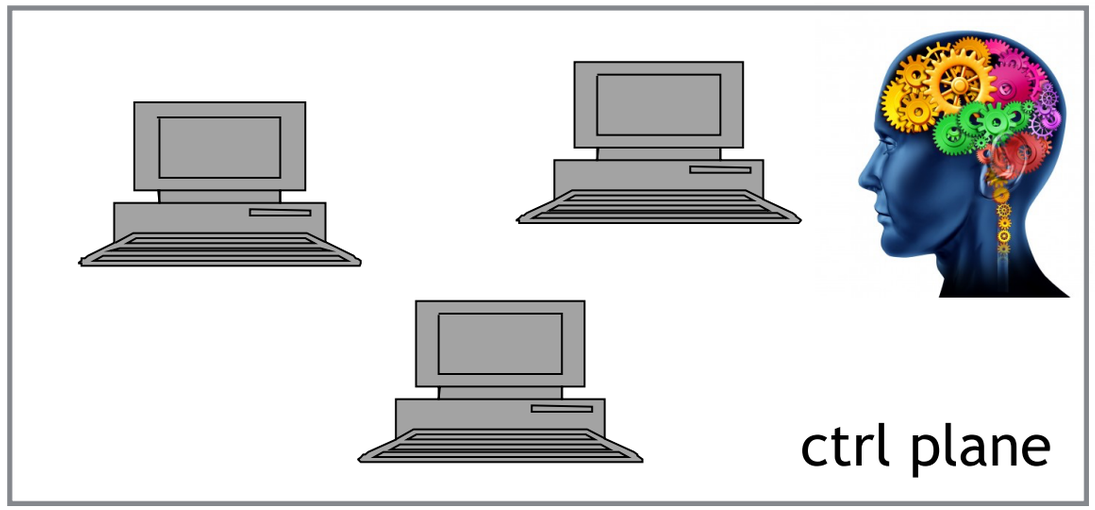
Tag	

Fast Failover - Adding Brains to the South

* Proactive reaction to link failures

Failover Group	
Fwd	Port **1**
Fwd	Port **3**
…	…
Action-k	Port-k

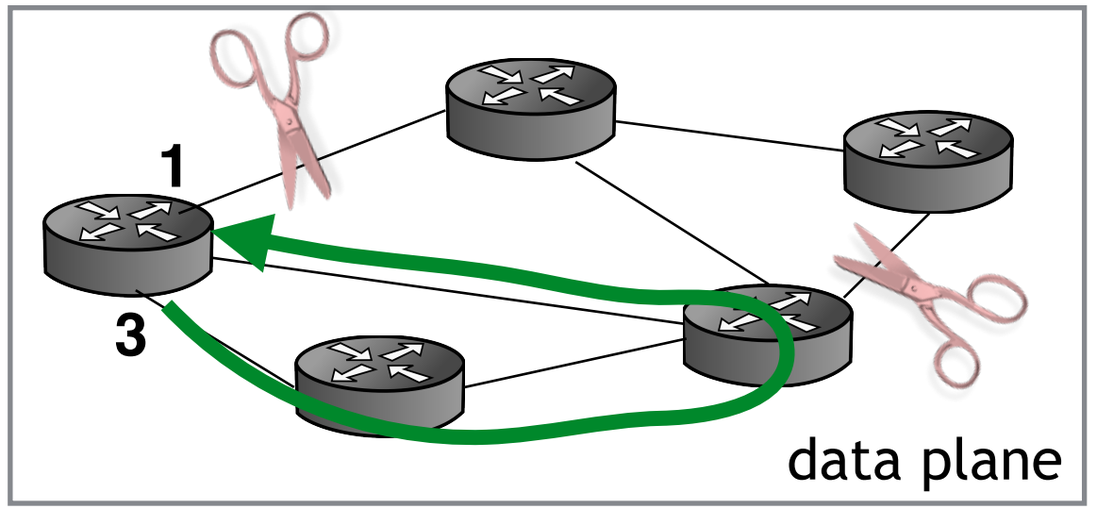




N

S

* Non-trivial to use
  * May quickly introduce loops
  * May introduce high load
* Much better with Tags



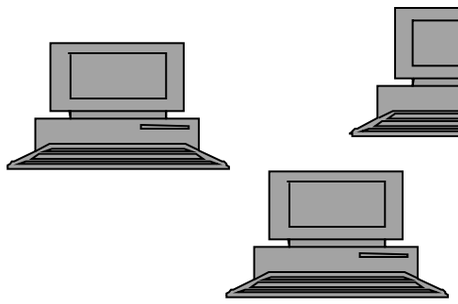
Tag	

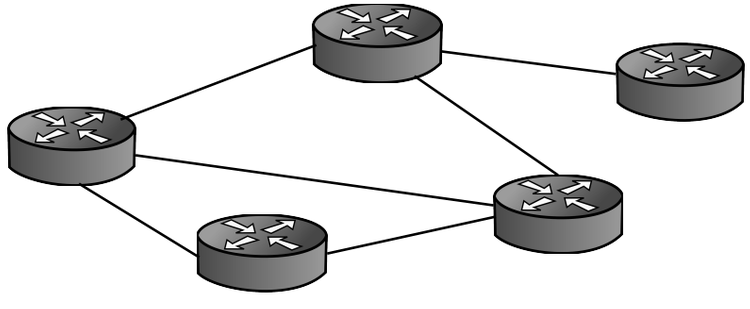
Functions in the ***South***

- Reduce interactions with the control plane
- Make data plane more robust

- Monitoring functions:
  - *Topology snapshot*
  - *Blackhole detection*
  - *Critical node detection*

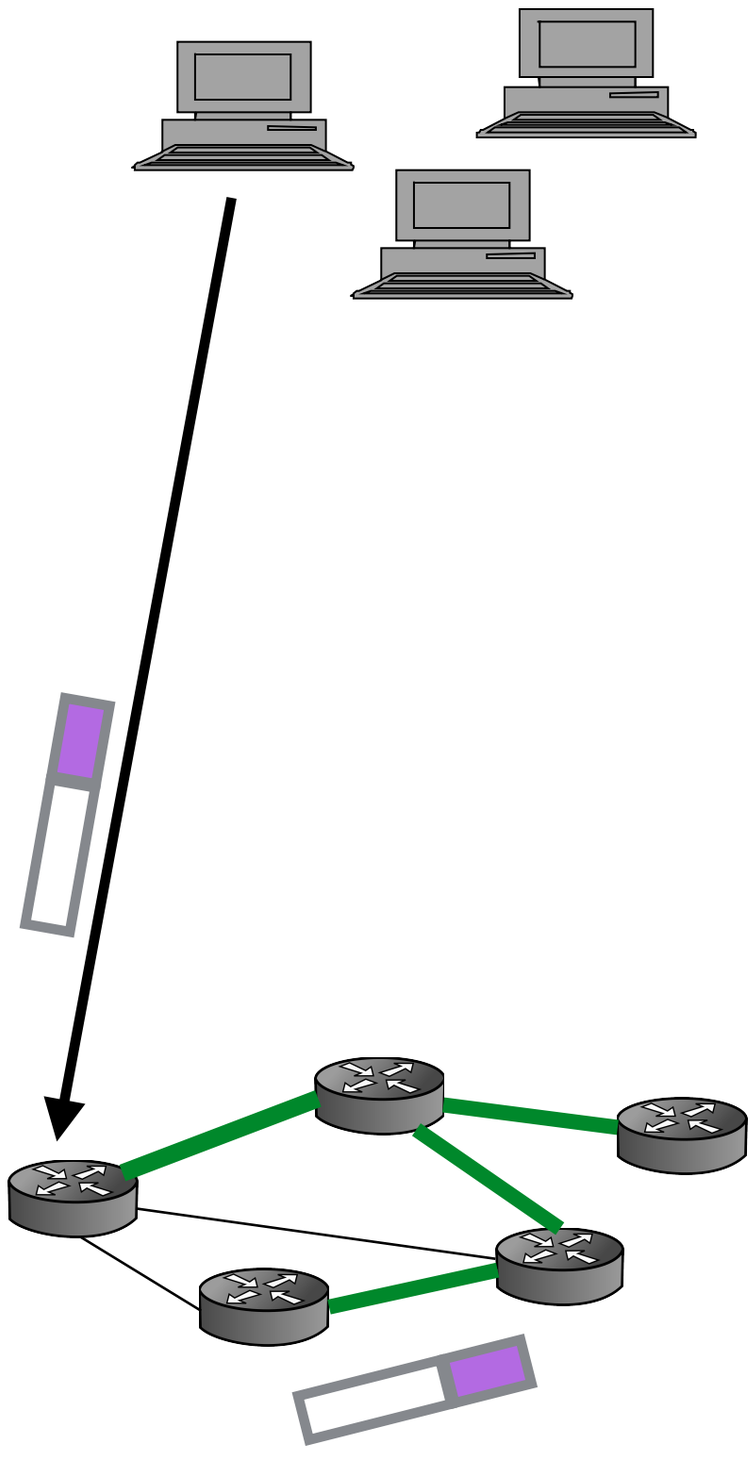
- Communication functions:
  - *Anycast*





How it is possible? ***SmartSouth*** template.

- SmartSouth — in-band graph DFS traversal
- State of each node stored in the packet:
  - parent
  - current neighbor the node traverses
- Implemented using a simple match-action paradigm
- **Uses Fast Failover technique.**



How it is possible? ***SmartSouth*** template.

* Pseudocode —> Match&Action tables

Algorithm 1 Algorithm ***SmartSouth*** – Template

Input: current node: v_i, input port: in, packet global params: $pkt.start$, packet tag array: $\{pkt.v_j\}_{j\in[n]}$
Output: output port: out

1: **if** $pkt.start = 0$ **then**
2: $pkt.start \leftarrow 1$
3: $out \leftarrow 1$
4: **else**
5: **if** $pkt.v_i.cur = 0$ **then**
6: $pkt.v_i.par \leftarrow in$; $out \leftarrow 1$; $First\_visit()$
7: **else if** $in = pkt.v_i.cur$ **then**
8: $out \leftarrow pkt.v_i.cur + 1$; $Visit\_from\_cur()$
9: **else**
10: $out \leftarrow in$; $Visit\_not\_from\_cur()$
11: **goto** 26
12: **if** $out = \Delta_i + 1$ **then**
13: $out \leftarrow pkt.v_i.par$
14: **goto** 22

15: **while** out failed **or** $out = pkt.v_i.par$ **do**
16: $out \leftarrow out + 1$
17: **if** $out = \Delta_i + 1$ **then**
18: $out \leftarrow pkt.v_i.par$
19: **goto** 22

20: $Send\_next\_neighbor()$
21: **goto** 23
22: $Send\_parent()$
23: $pkt.v_i.cur \leftarrow out$
24: **if** $out = 0$ **then**
25: $Finish()$
26: **return** out

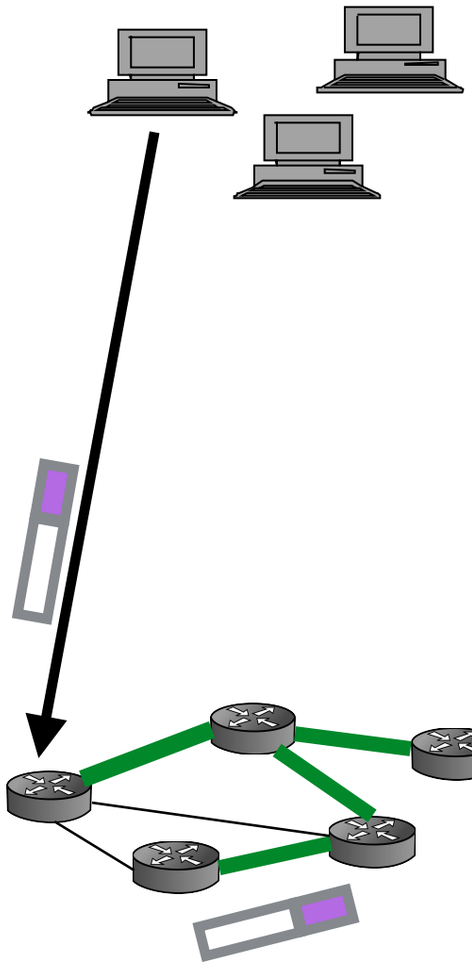
How it is possible? ***SmartSouth*** template.

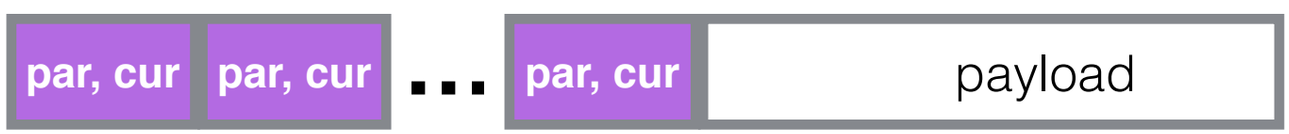
* Pseudocode —> Match&Action tables

if $pkt.v_i.cur = 0$ **then**
  $pkt.v_i.par \leftarrow in$; $out \leftarrow 1$;
else if $in = pkt.v_i.cur$ **then**
  $out \leftarrow pkt.v_i.cur + 1$;

Flow Table B

Match			Instructions
in	$pkt.v_i.cur$	$pkt.v_i.par$	
*	0	*	$pkt.v_i.par \leftarrow in$, Table 1
1	1	*	Table 2
2	2	*	Table 3
3	3	*	Table 4
...	...	...	...





How it is possible? ***SmartSouth*** template.

* Pseudocode —> Match&Action tables

if $pkt.v_i.cur = 0$ **then**
  $pkt.v_i.par \leftarrow in$; $out \leftarrow 1$;
else if $in = pkt.v_i.cur$ **then**
  $out \leftarrow pkt.v_i.cur + 1$;

while *out* failed
  $out \leftarrow out + 1$

Flow Table B

Match			Instructions
in	$pkt.v_i.cur$	$pkt.v_i.par$	
*	0	*	$pkt.v_i.par \leftarrow in$, Table 1
1	1	*	Table 2
2	2	*	Table 3
3	3	*	Table 4
...	...	...	...

Flow Table 1

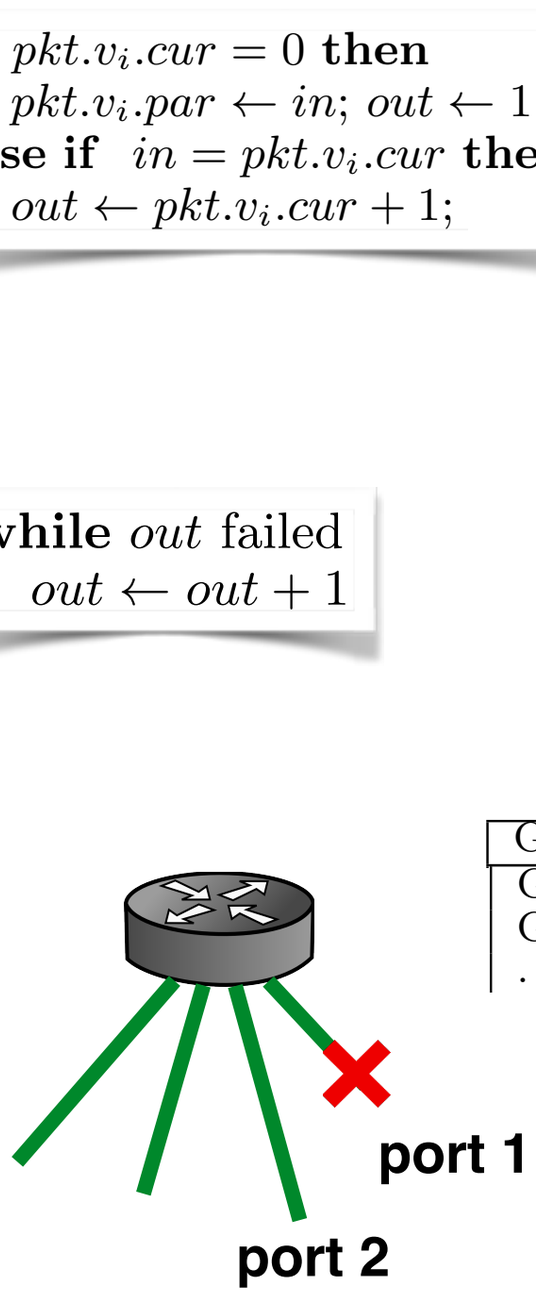
Match	Instructions
sb	
0	Gr 1, Table 2
1	Drop

Flow Table 2

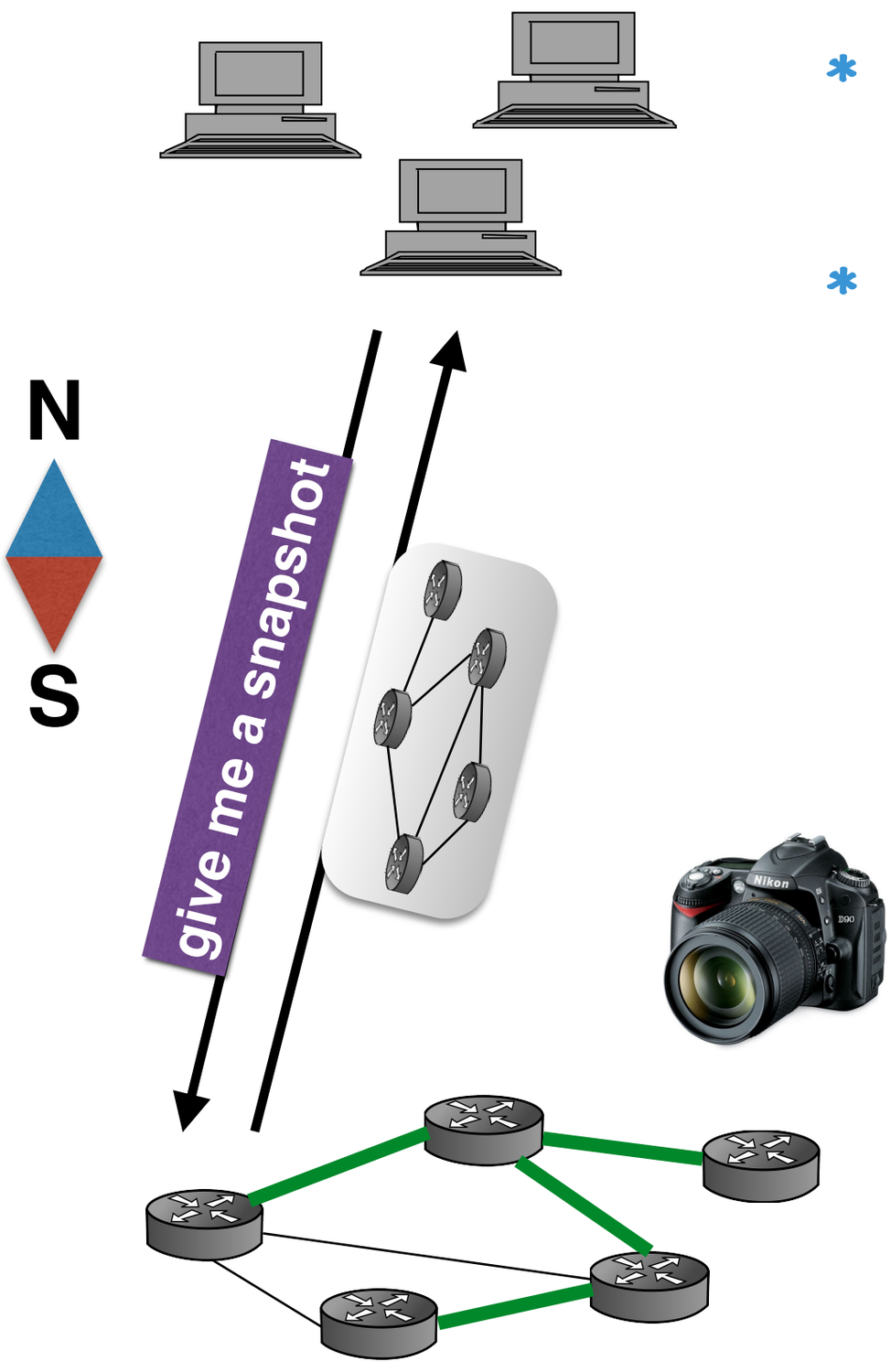
Match	Instructions
sb	
0	Gr 2, Table 3
1	Drop

...

Group	Actions
Gr 1	$\langle sb \leftarrow 1, pkt.v_i.cur \leftarrow 1, pkt.start \leftarrow 1, \text{Fwd } 1\rangle$
Gr 2	$\langle sb \leftarrow 1, pkt.v_i.cur \leftarrow 2, pkt.start \leftarrow 1, \text{Fwd } 2\rangle$
...	...

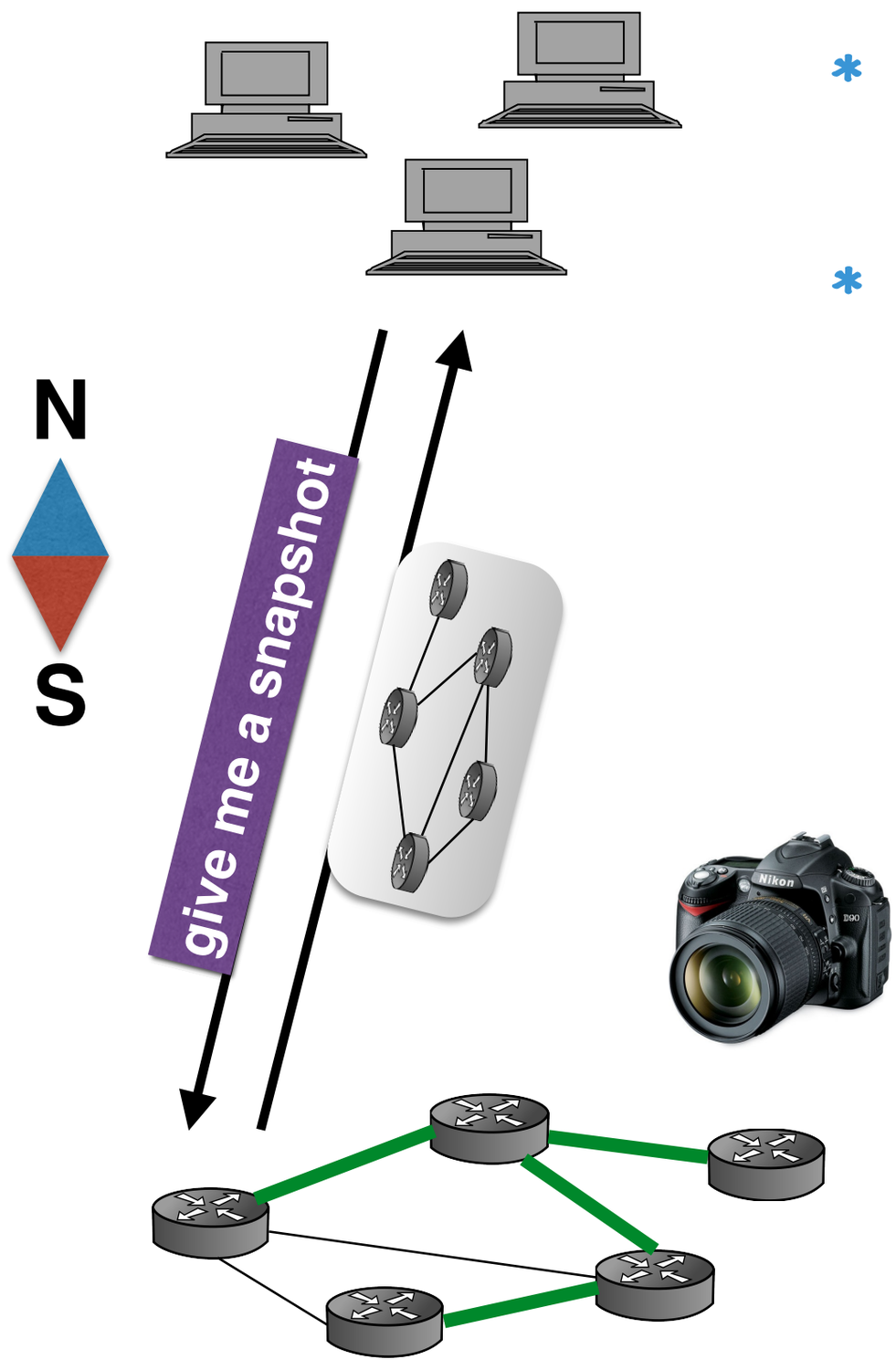


Functions in the *South:* ***Topology Snapshot***



- Fault tolerant
  - No connectivity assumption
- Requires a single connection to controller
  - Unlike built-in “Topology service” in OpenFlow

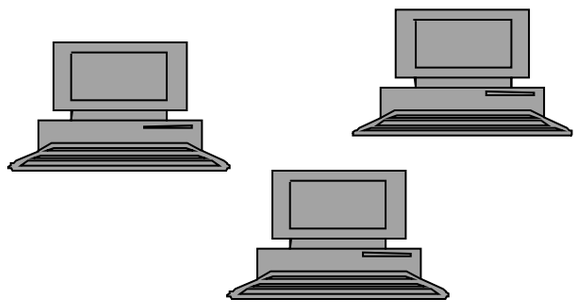
Functions in the *South:* ***Topology Snapshot***

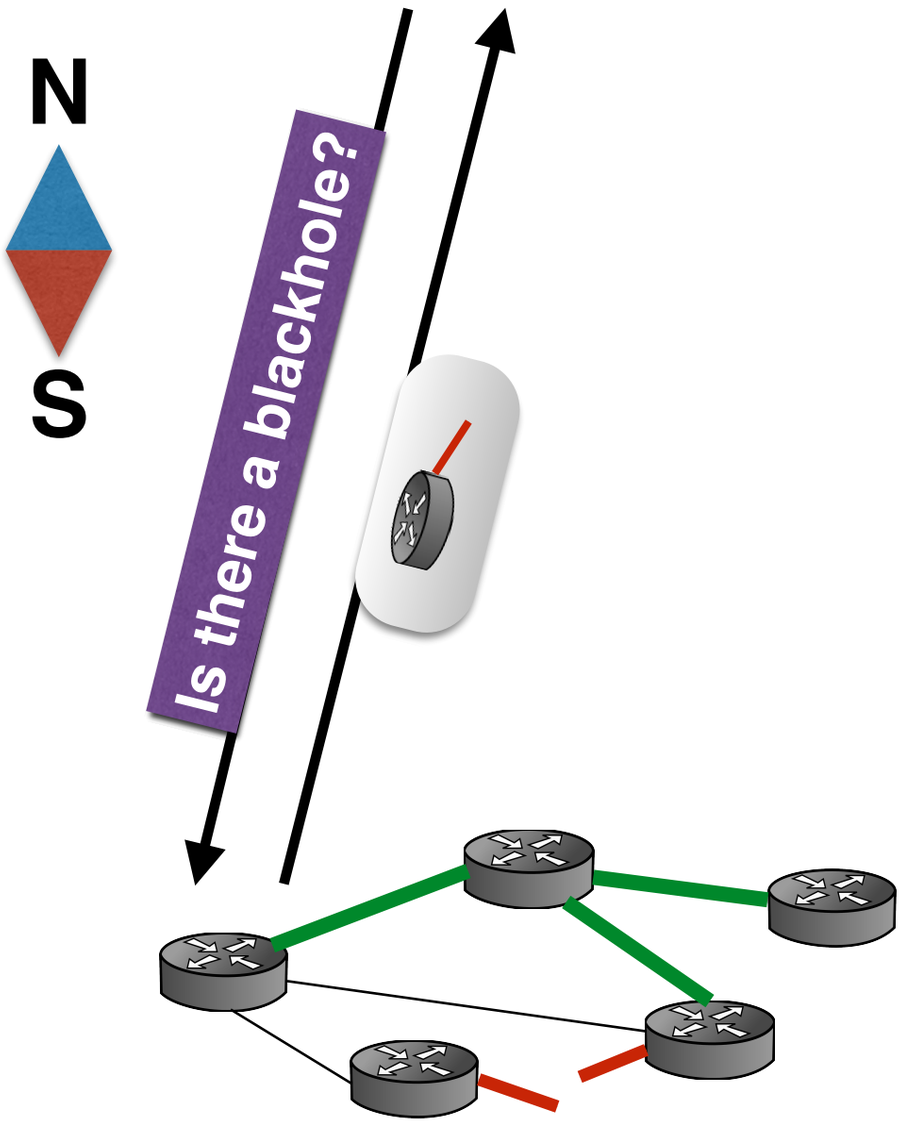


- Fault tolerant
  - No connectivity assumption
- Requires a single connection to controller
  - Unlike built-in “Topology service” in OpenFlow

- During the DFS traversal, topology information is written to the packet header

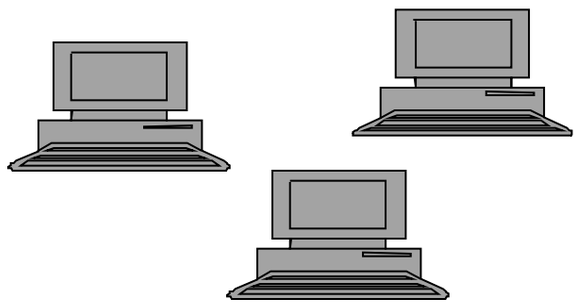
Functions in the *South:* ***Blackhole Detection***

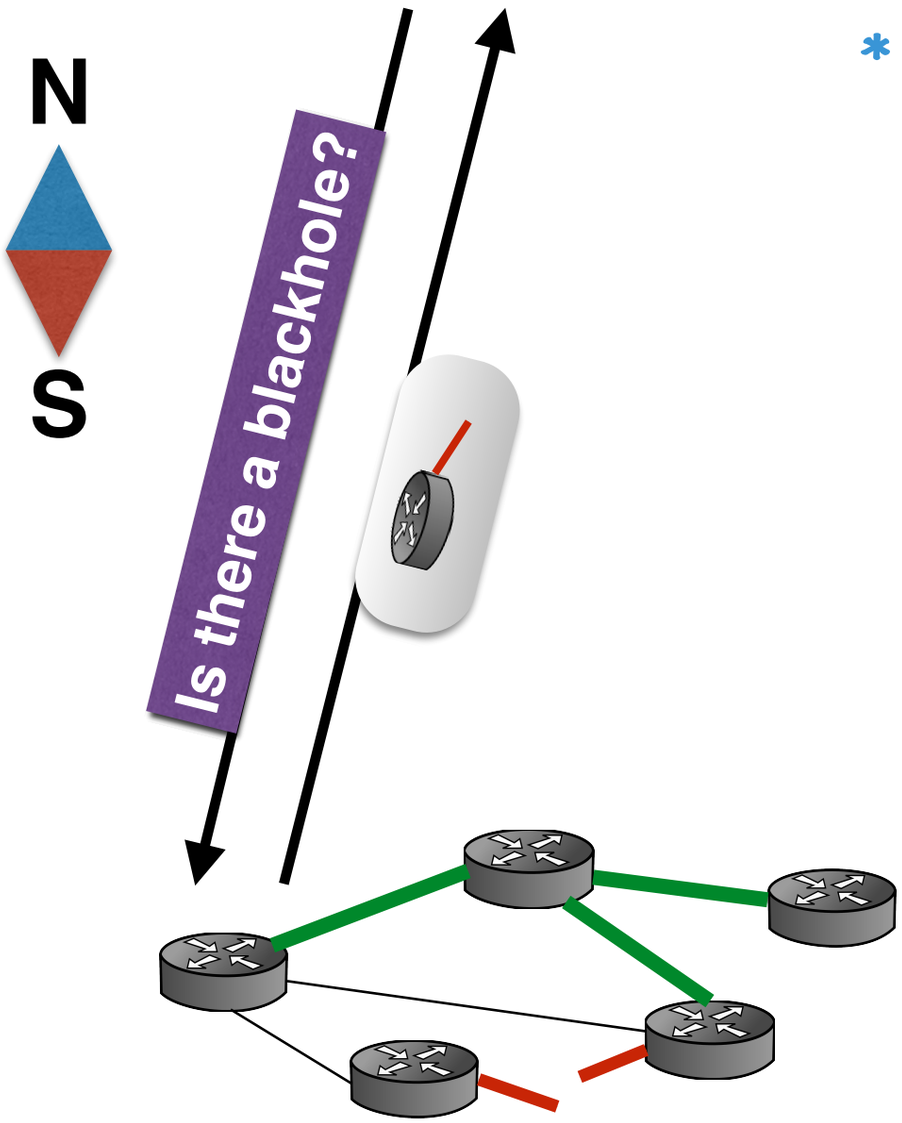




- Detects connectivity loss regardless of the cause
  - physical failure
  - configuration errors
  - unsupervised carrier network errors

Functions in the *South:* ***Blackhole Detection***

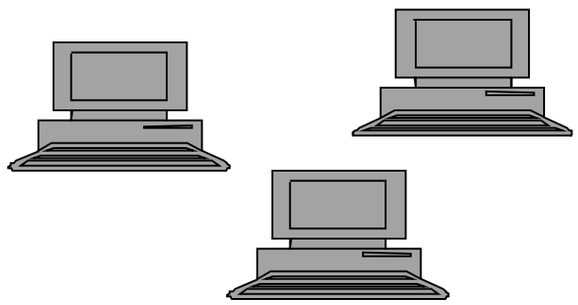


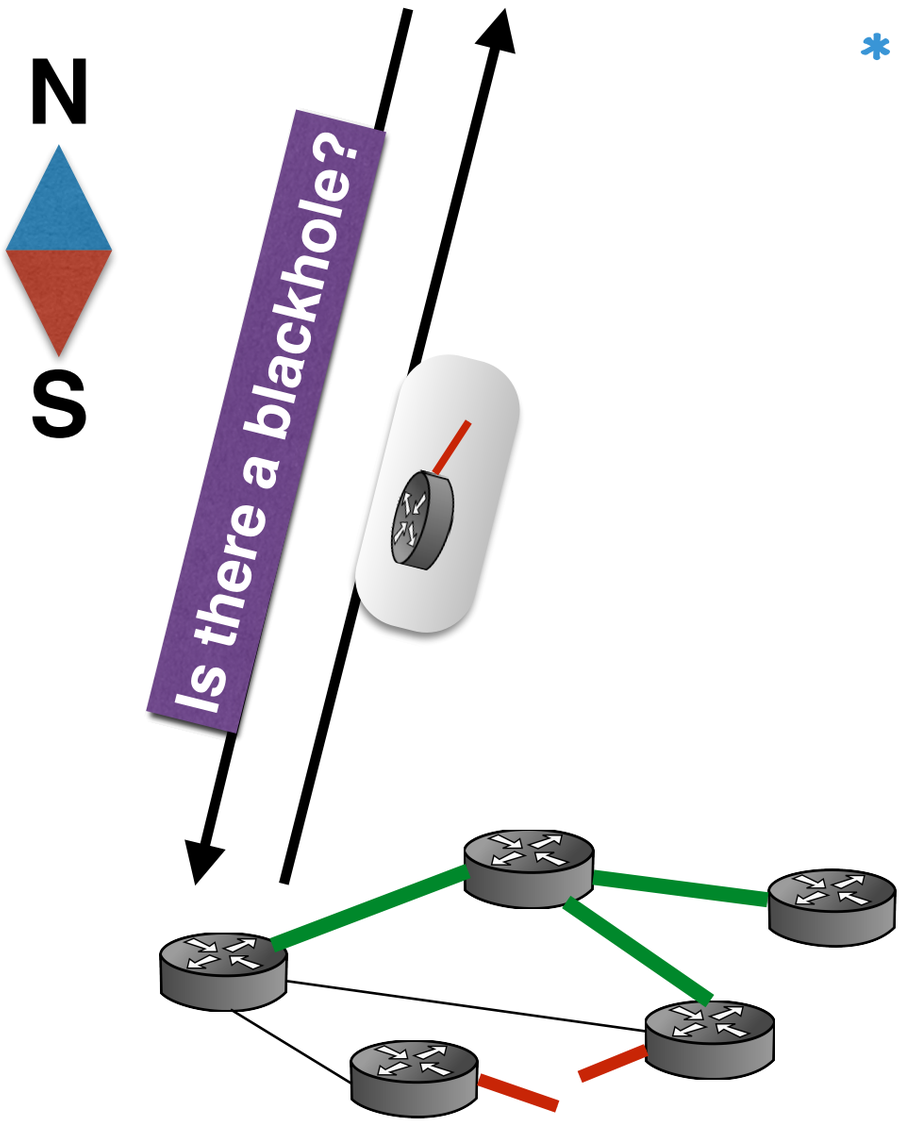


- Detects connectivity loss regardless of the cause
  - physical failure
  - configuration errors
  - unsupervised carrier network errors

- Two possible implementations:

Functions in the *South:* ***Blackhole Detection***

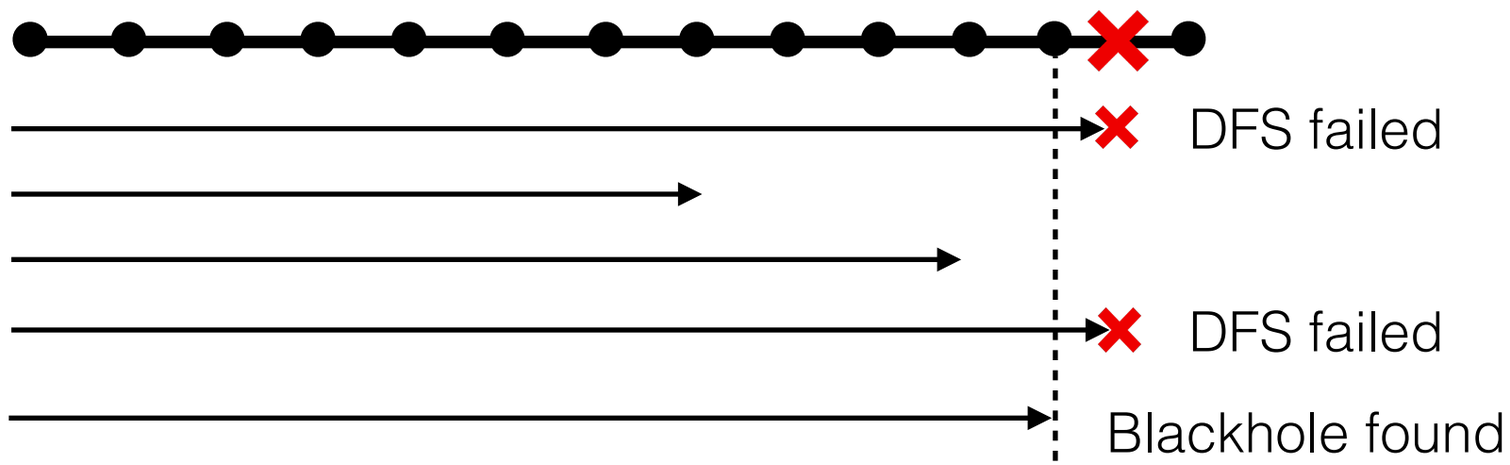




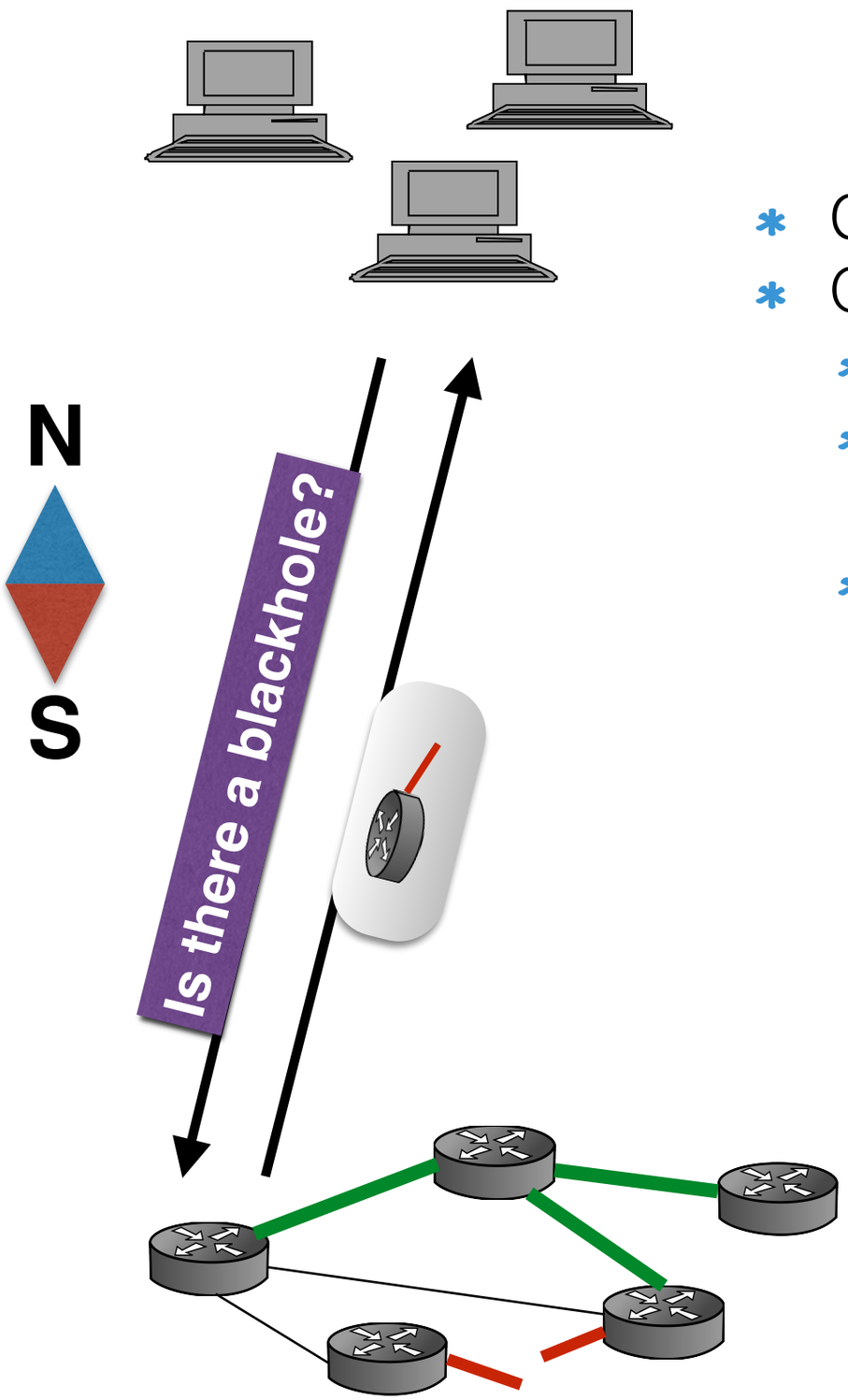
- Detects connectivity loss regardless of the cause
  - physical failure
  - configuration errors
  - unsupervised carrier network errors

- Two possible implementations:

- ***DFS traversal with TTL***
  - (log n) DFS traversals (binary search)

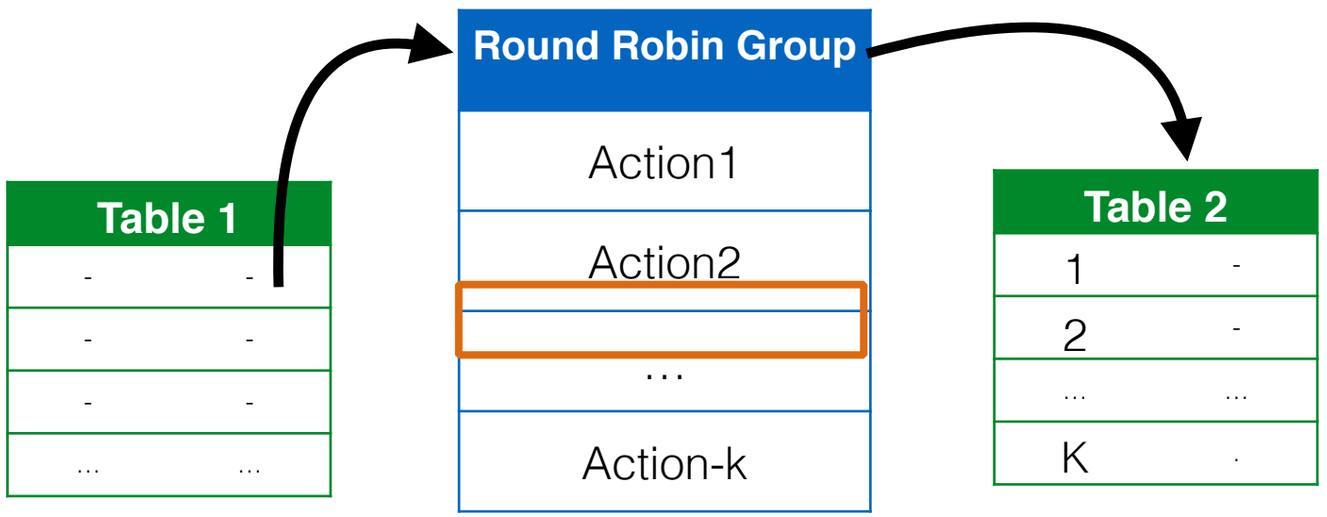


Functions in the *South:* ***Blackhole Detection***

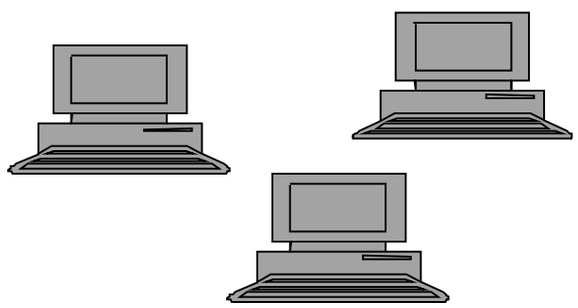


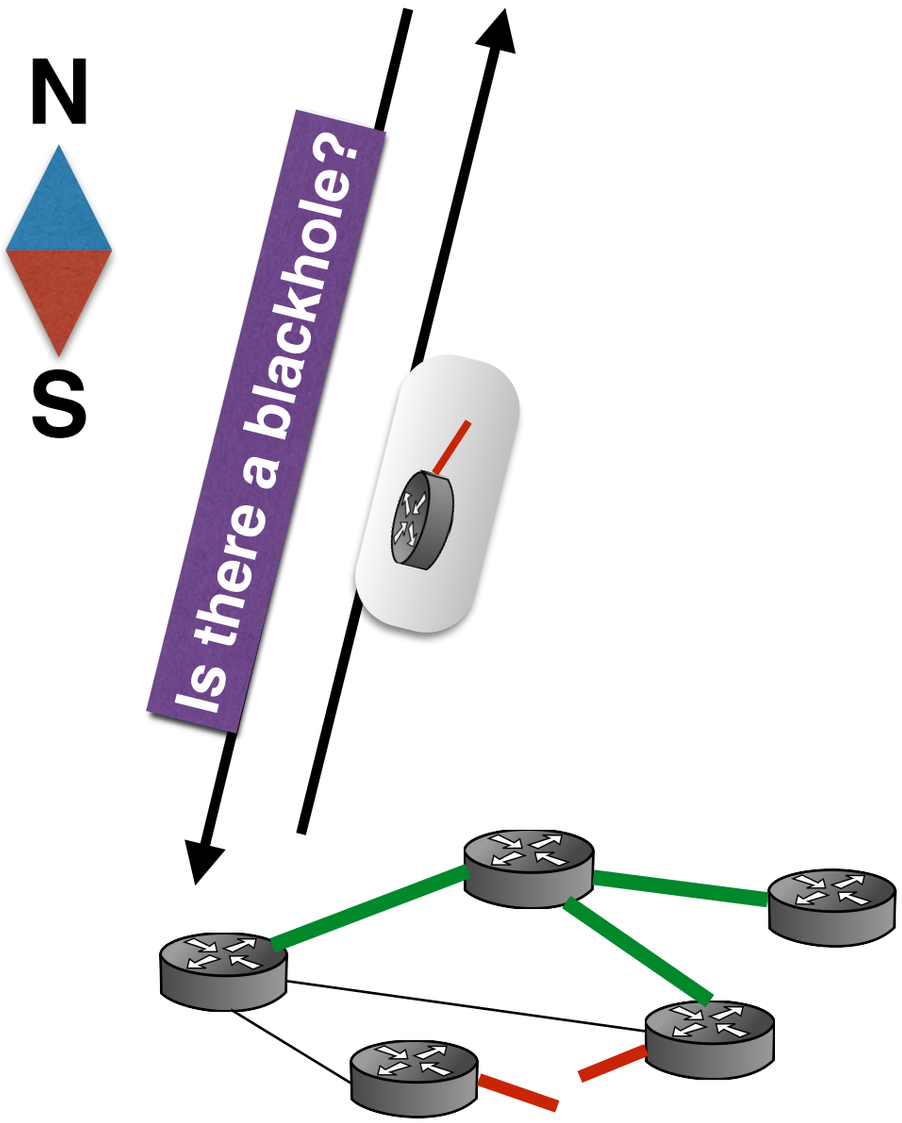
Smart **"In-band"** ***Counters***

- General counters - access only by controller
- Our counters:
  - access during packet processing
  - counter value can be written to packet or metadata
  - implemented using Round-Robin action group



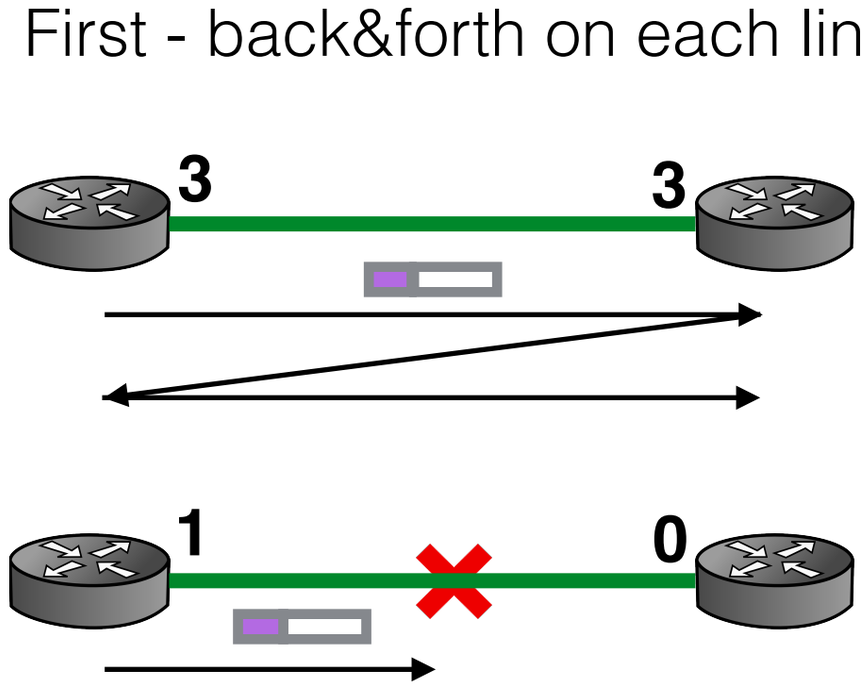
Functions in the *South:* ***Blackhole Detection***



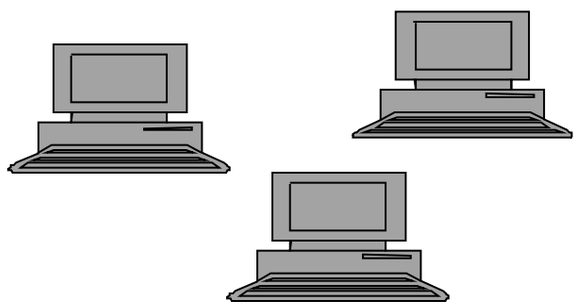


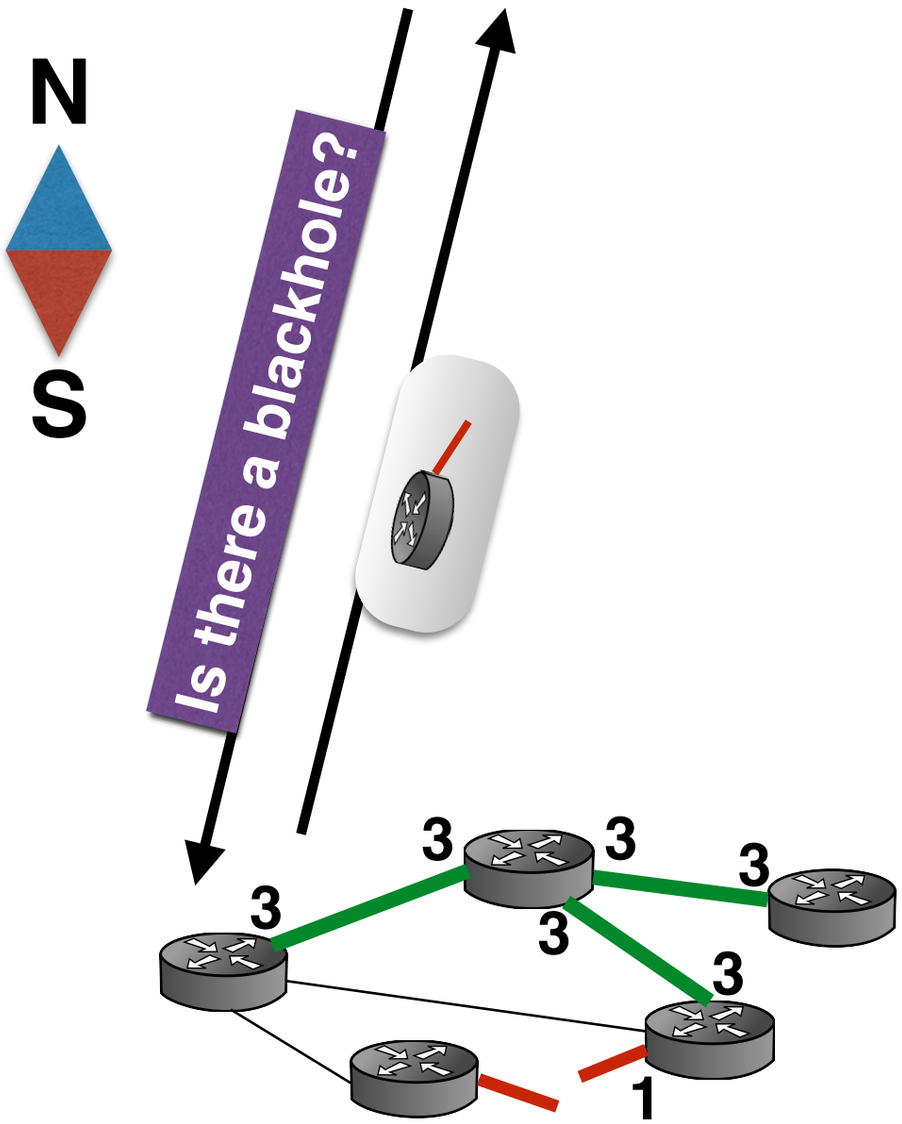
Blackhole detection with *SmartCounters*

- Install *SmartCounter* for each port
- Only two DFS traversals required:
  - First - back&forth on each link



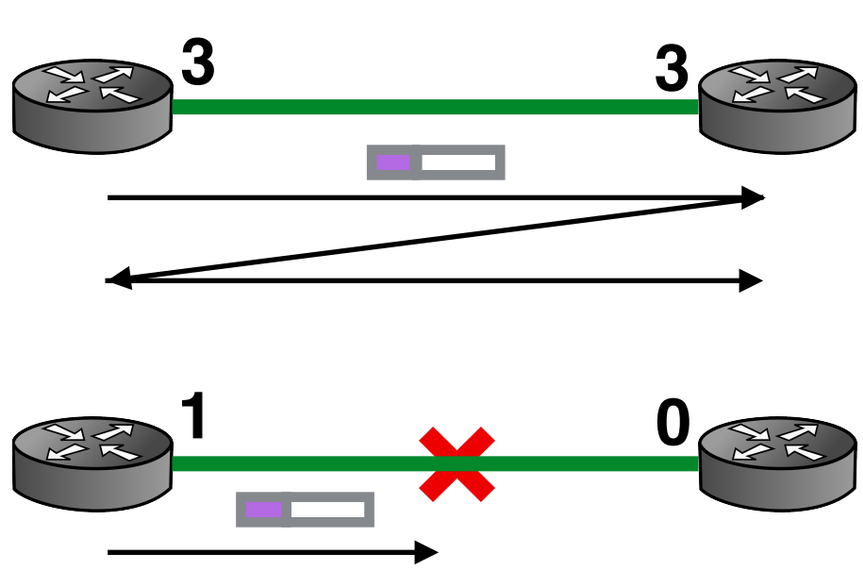
Functions in the *South:* ***Blackhole Detection***





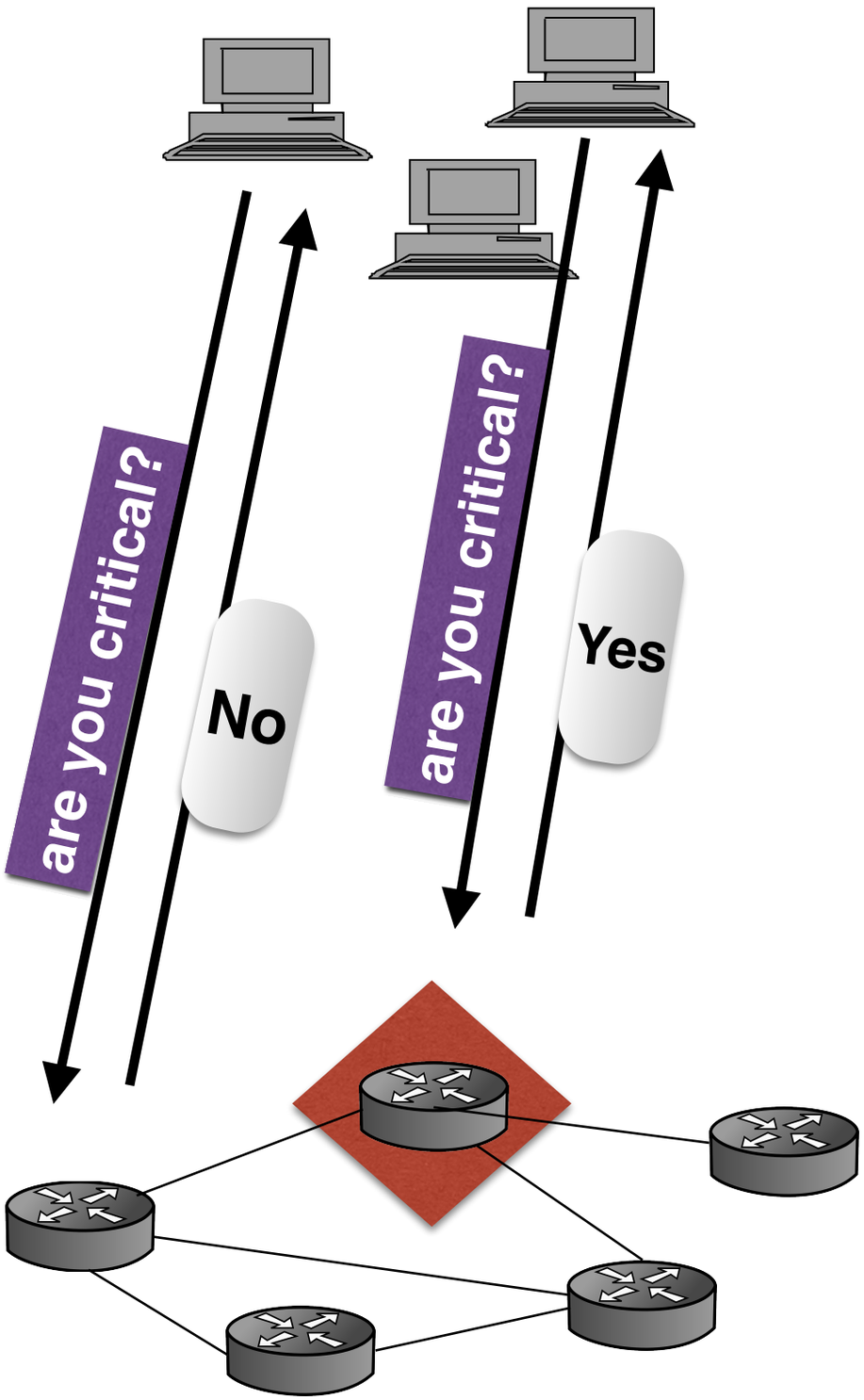
Blackhole detection with *SmartCounters*

- Install *SmartCounter* for each port
- Only two DFS traversals required:
  - First - back&forth on each link



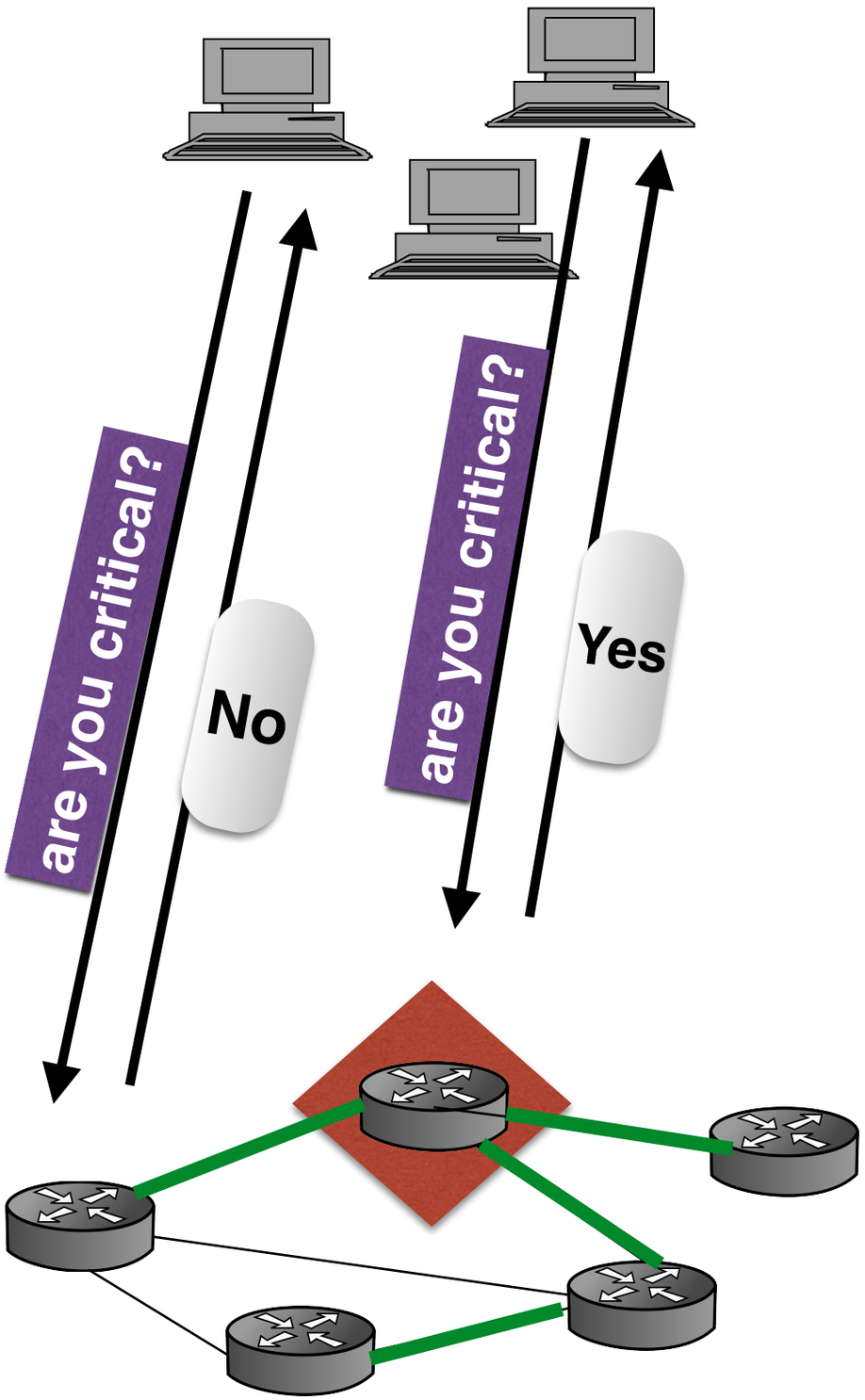
- Second - find port with counter value 1

Functions in the *South:* ***Critical Node Detection***



- Checks if a node is critical for connectivity
- Non-critical node may be removed for
  - maintenance
  - energy conservation
- Cheaper than *Snapshot*

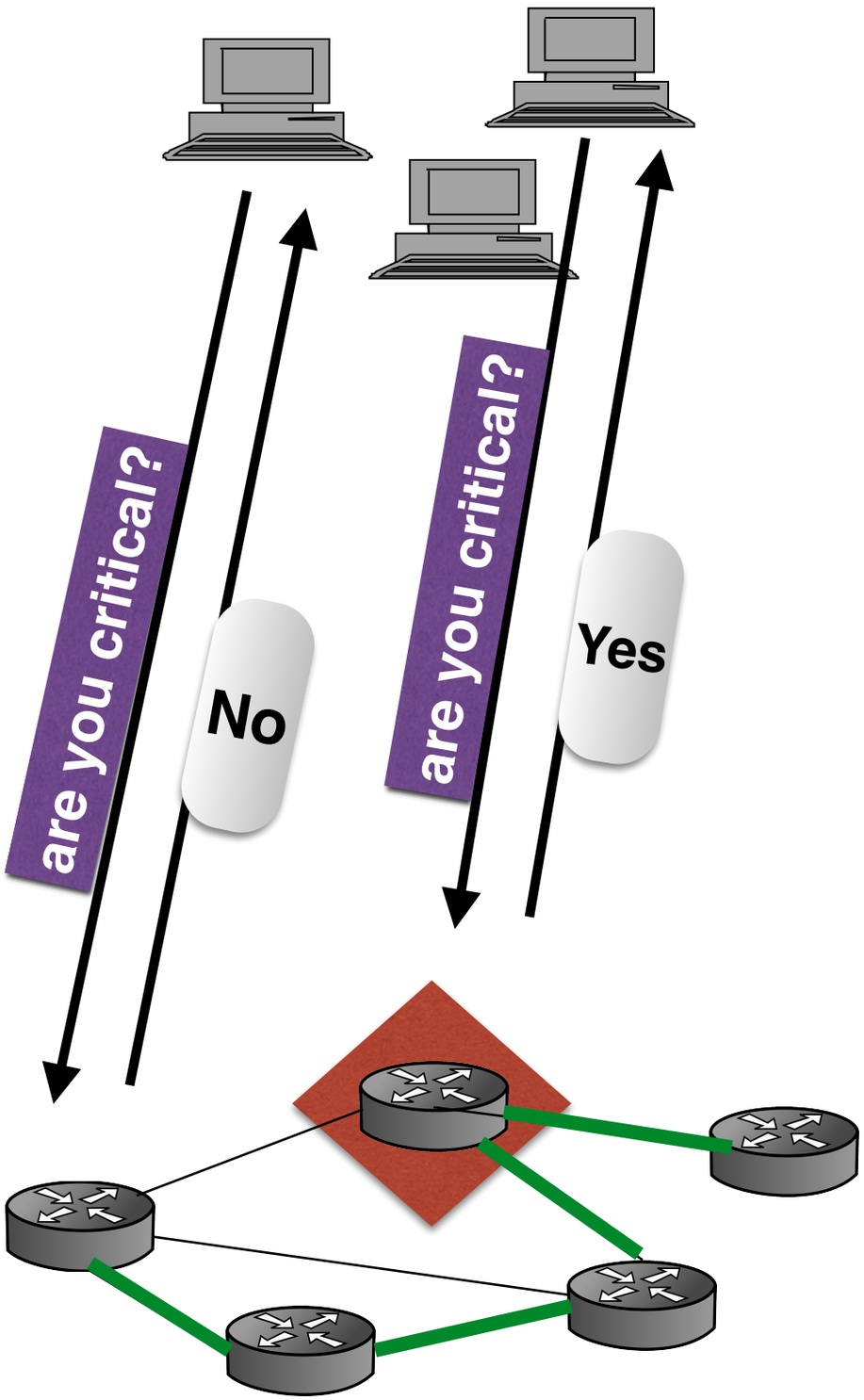
Functions in the *South:* ***Critical Node Detection***



- Checks if a node is critical for connectivity
- Non-critical node may be removed for
  - maintenance
  - energy conservation
- Cheaper than *Snapshot*

- One DFS traversal with *root = v*
  - **If** *v* non-critical: it is parent for exactly one node

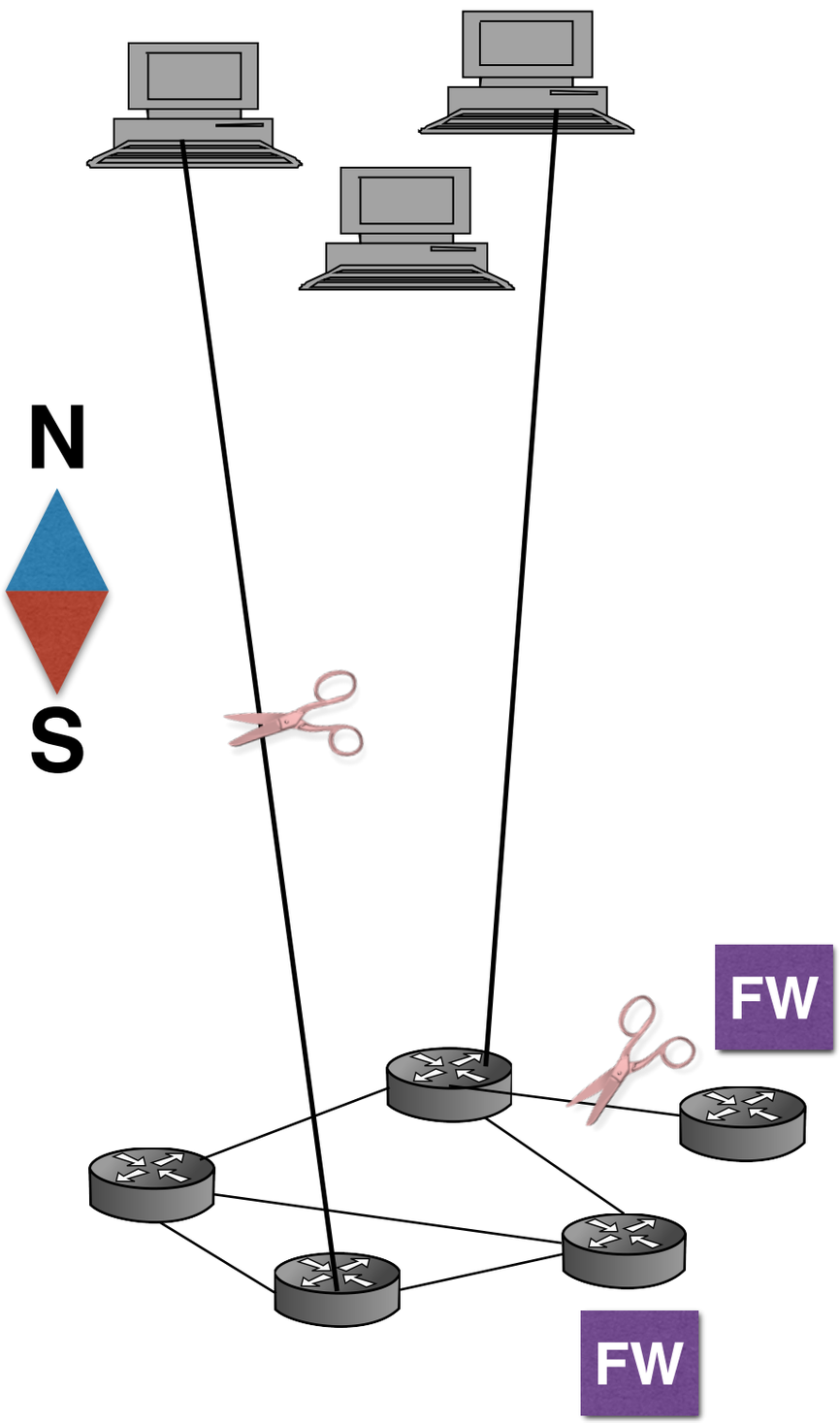
Functions in the *South:* ***Critical Node Detection***



- Checks if a node is critical for connectivity
- Non-critical node may be removed for
  - maintenance
  - energy conservation
- Cheaper than *Snapshot*

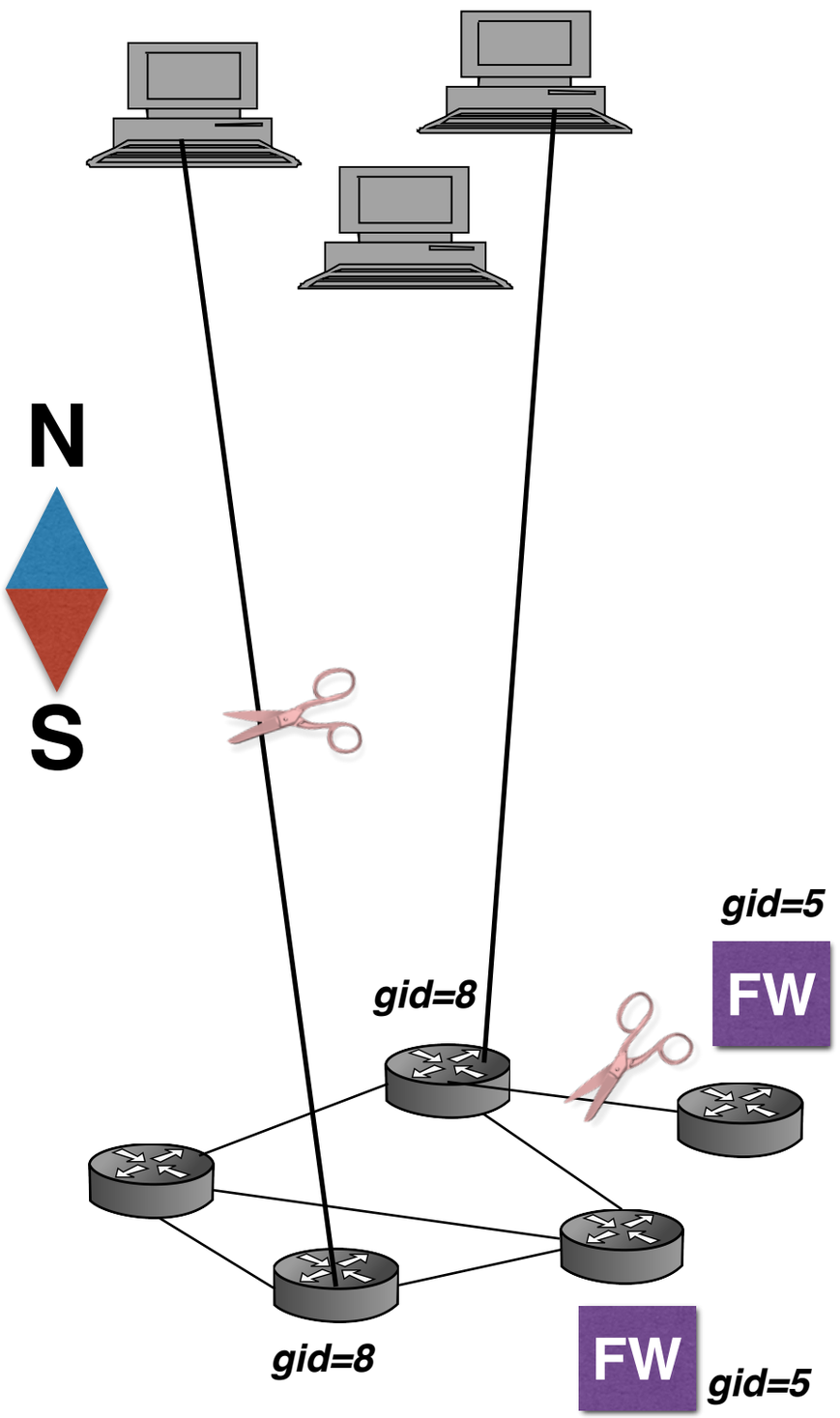
- One DFS traversal with *root = v*
  - **If** *v* non-critical: it is parent for exactly one node
  - **Else**: it is parent for more than 1 node

Functions in the *South:* ***Anycast***



- Multiple, unknown destinations, NFV
- Alternative path to the control plane
- Extendable to service chains

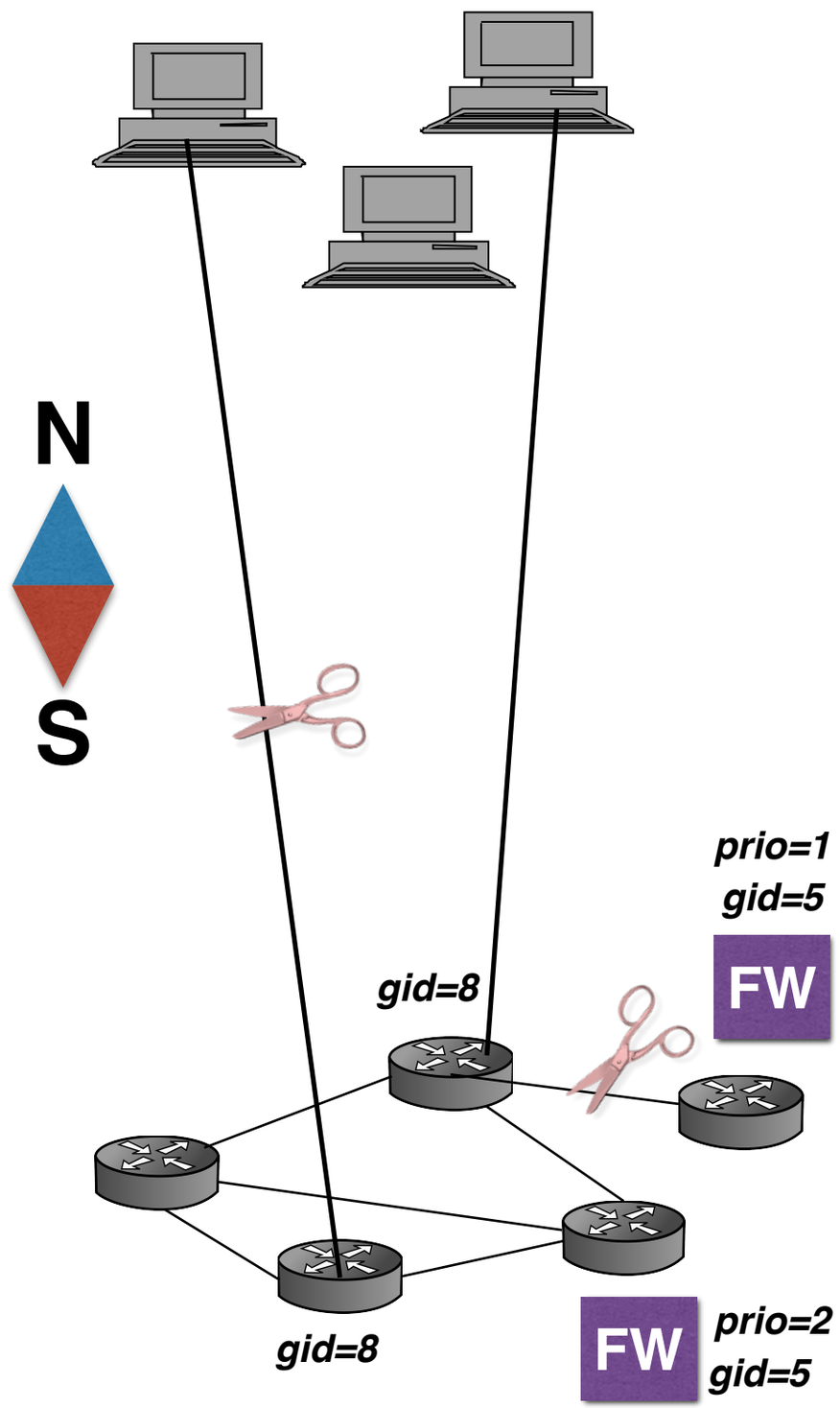
Functions in the *South:* ***Anycast***



- Multiple, unknown destinations, NFV
- Alternative path to the control plane
- Extendable to service chains

- **Anycast** - one DFS traversal
  - If ***gid*** match: forward to “self port”
  - Else: continue DFS traversal

Functions in the *South:* ***Anycast***



- Multiple, unknown destinations, NFV
- Alternative path to the control plane
- Extendable to service chains

- **Anycast** - one DFS traversal
  - If ***gid*** match: forward to “self port”
  - Else: continue DFS traversal

- **Priocast -** two DFS traversals
  - First - find highest priority dest
  - Second - deliver to the best dest

SmartSouth in practice

- Using existing OpenFlow match fields for tagging:
  - IPv6 addresses
  - VLAN tags
  - MPLS tags
  - Using this approach we can support up to few dozens of nodes

- More tag space in the future
  - Future OpenFlow will probably support flexible match and set
  - NoviFlow switches already support UDP payload access

Conclusions

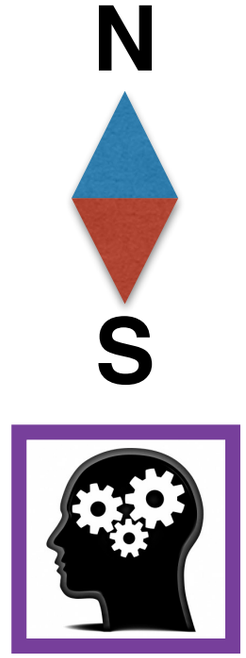
- "Dumb" data plane can implement useful and complex functionality
  - Snapshot
  - Blackhole detection
  - Critical node detection
  - Anycast
  - Smart Counters

- Nourish discussion on what should be implemented and where

Conclusions

- "Dumb" data plane can implement useful and complex functionality
  - Snapshot
  - Blackhole detection
  - Critical node detection
  - Anycast
  - Smart Counters

- Nourish discussion on what should be implemented and where



Thank You!

Related Work

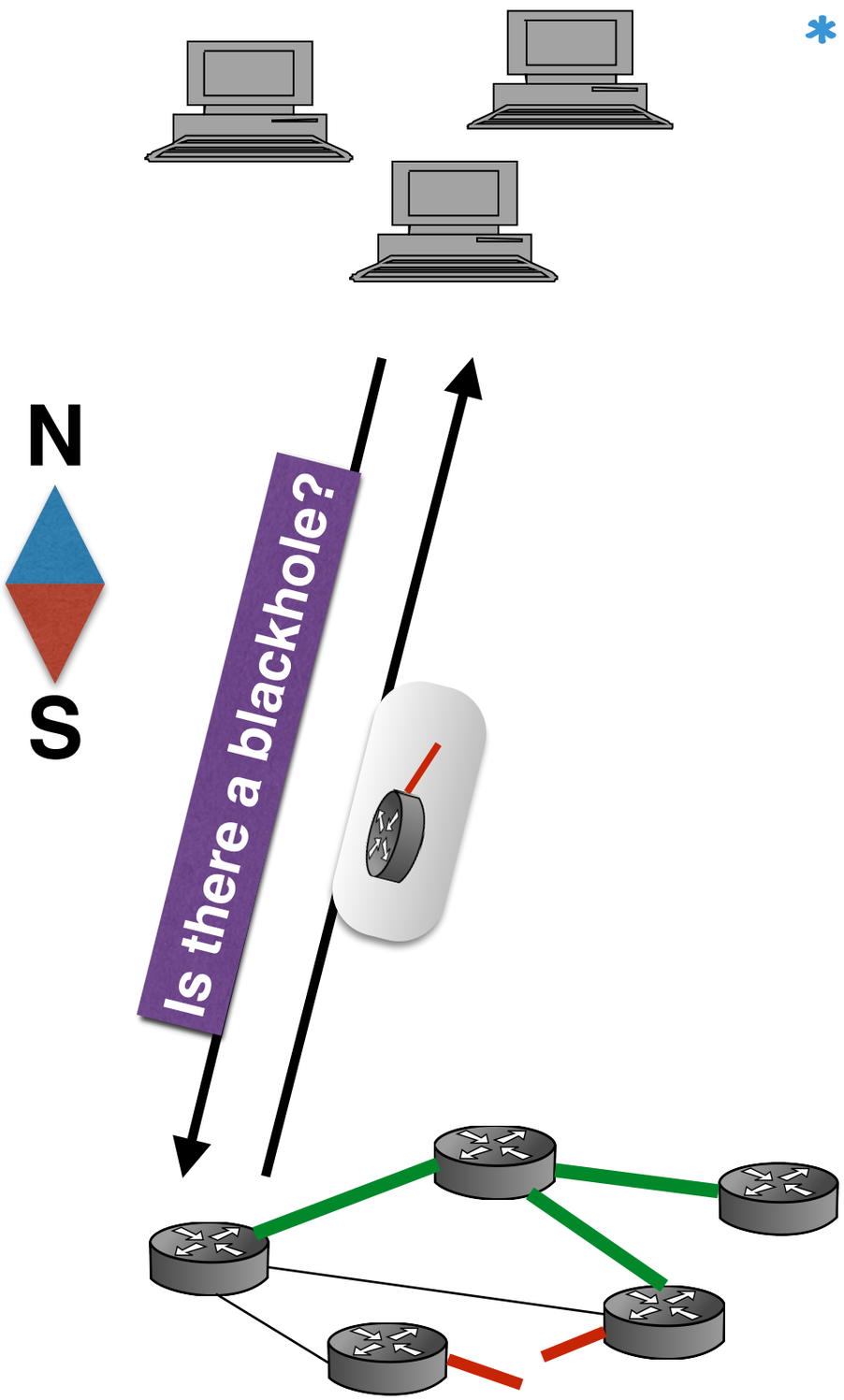
- Our Opodis
- Our hotsdn
- Shapira's link reversal?
- Explain why it cannot be implemented in OpenFlow (since it requires storing state in the switch)

Complexity

- Number of messages used by each function
- Space required in the header (take from the HotNets paper)

Service	Complexity	
	out-band #msgs×size	in-band #msgs×size
Snapshot	$1 \times O(1) + 1 \times O(\|E\|)$	$(4\|E\| - 2n) \times O(\|E\|)$
Anycast	0	$(4\|E\| - 2n) \times \|data\|$
Priocast	0	$(8\|E\| - 4n) \times \|data\|$
Blackhole 1	$2 \log \|E\| \times O(1)$	$(8\|E\| - 4n) \times O(1)$
Blackhole 2	$3 \times O(1)$	$4\|E\| \times O(1)$
Critical	$2 \times O(1)$	$(4\|E\| - 2n) \times O(1)$

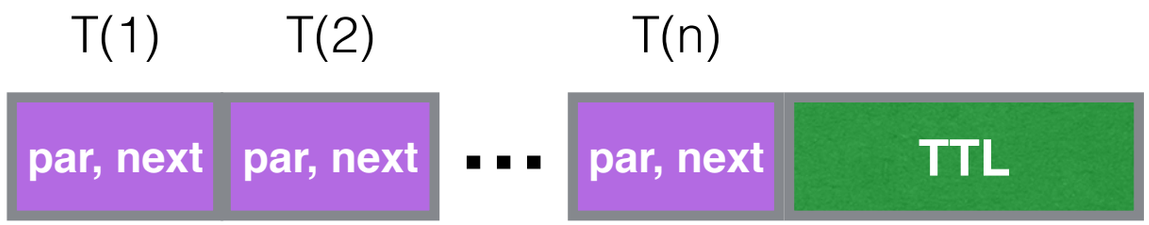
Functions in the *South:* ***Blackhole Detection***



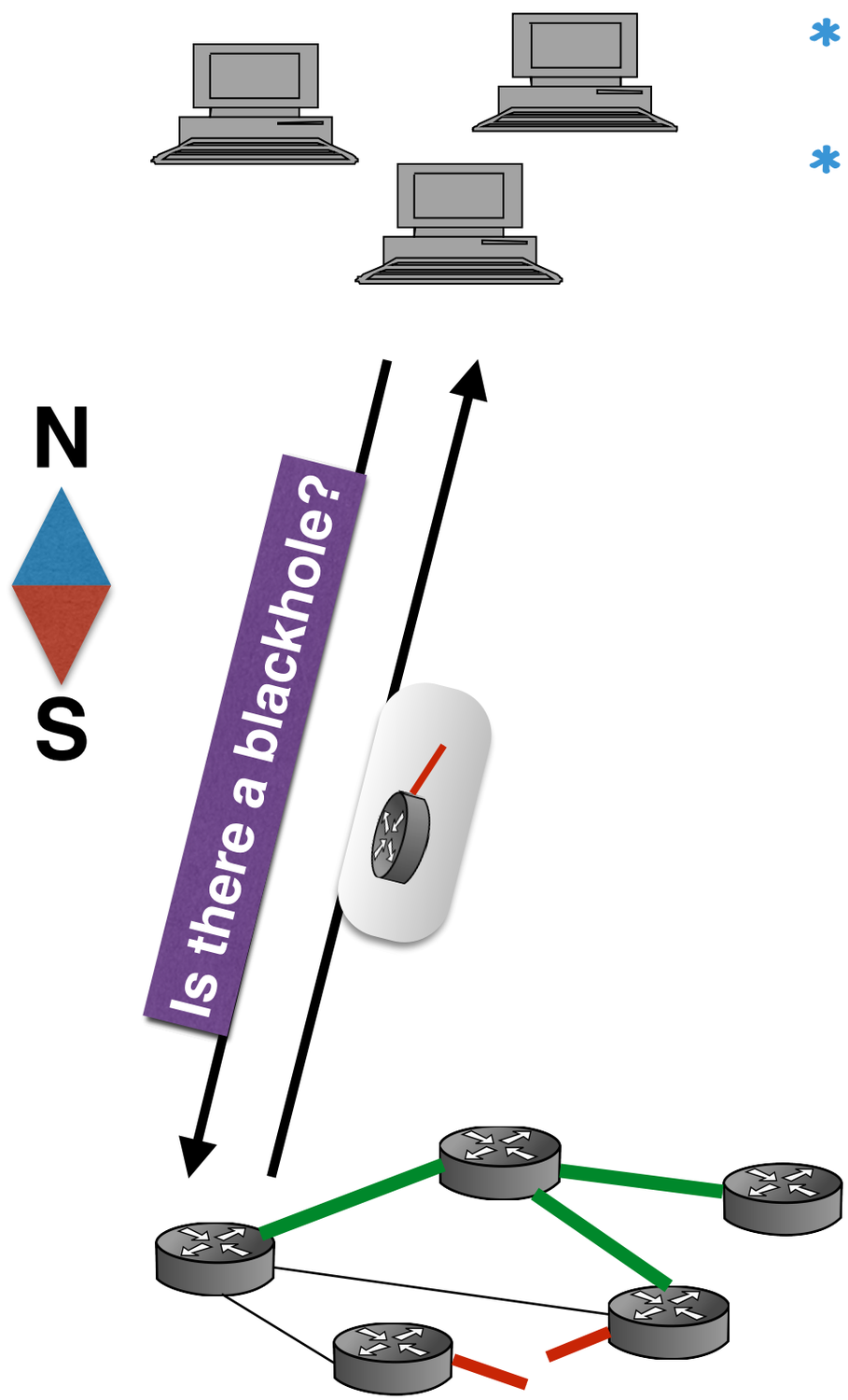
* Detects connectivity loss regardless of the cause

Functions in the *South:* ***Blackhole Detection***

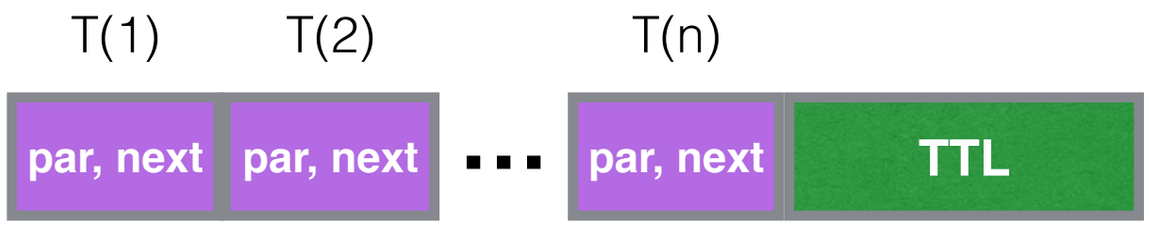
- Detects connectivity loss regardless of the cause
- Two possible implementations:
  - DFS traversal with TTL
    - (log n) DFS traversals (binary search)



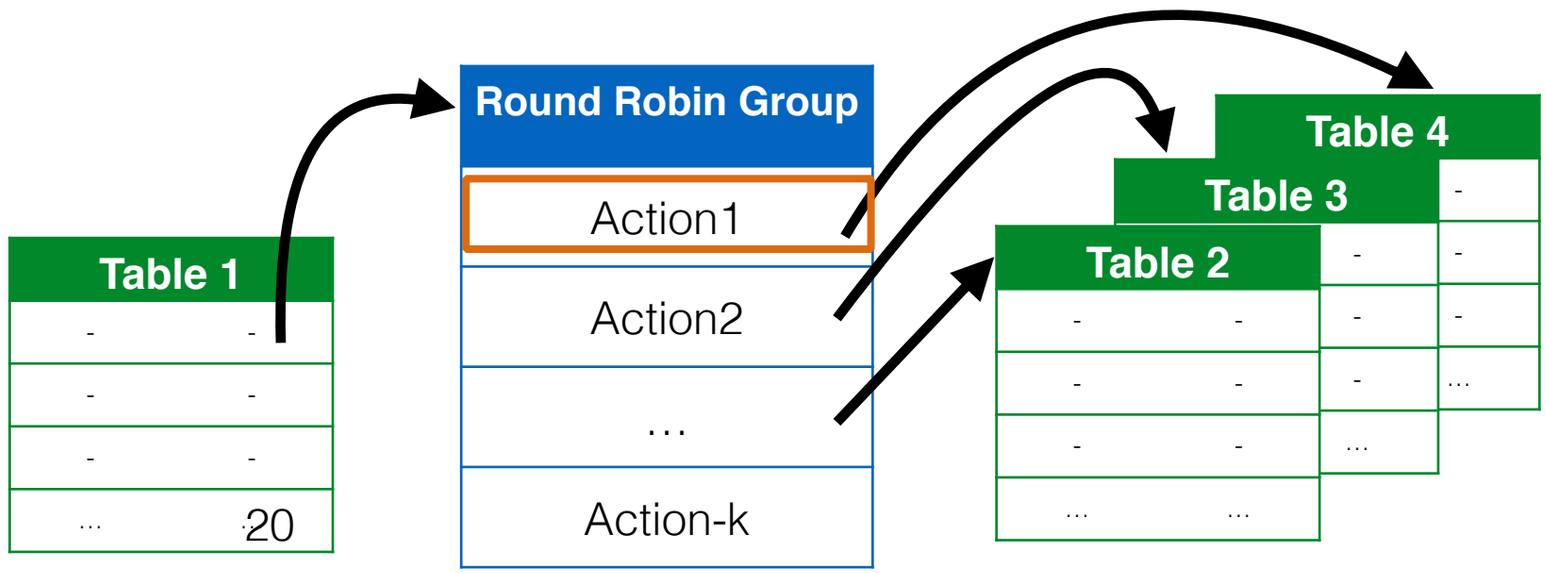
Functions in the *South:* ***Blackhole Detection***



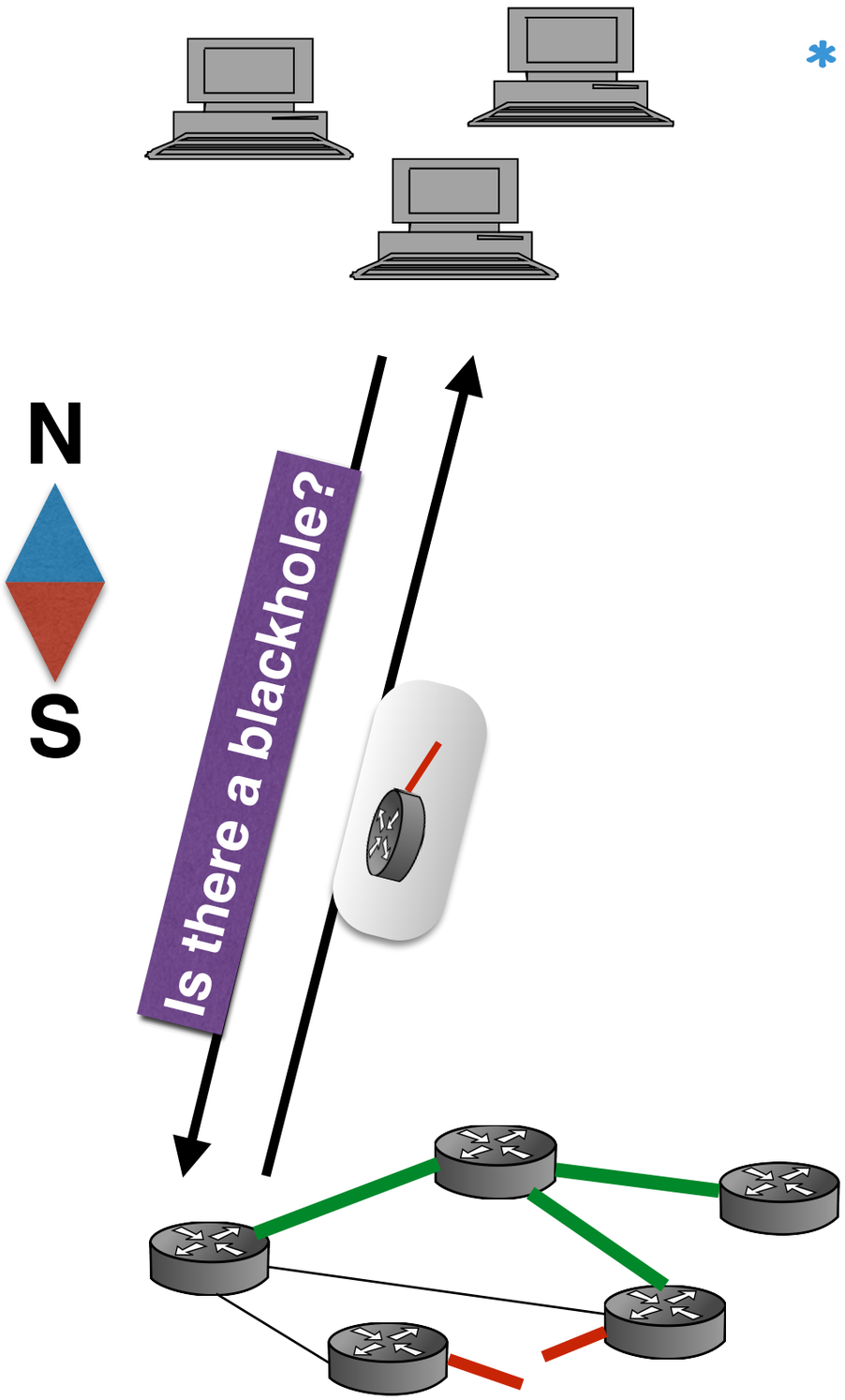
- Detects connectivity loss regardless of the cause
- Two possible implementations:
  - DFS traversal with TTL
    - (log n) DFS traversals (binary search)



- ***Smart Counters:***
  - Adding “state” to the switch
  - Only two DFS traversals required

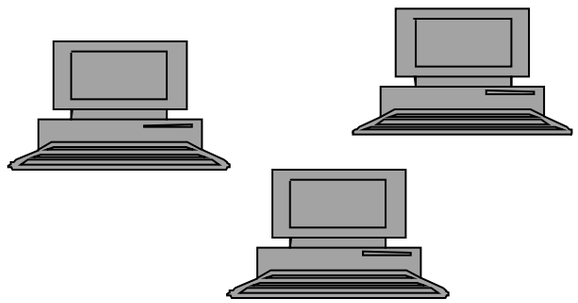


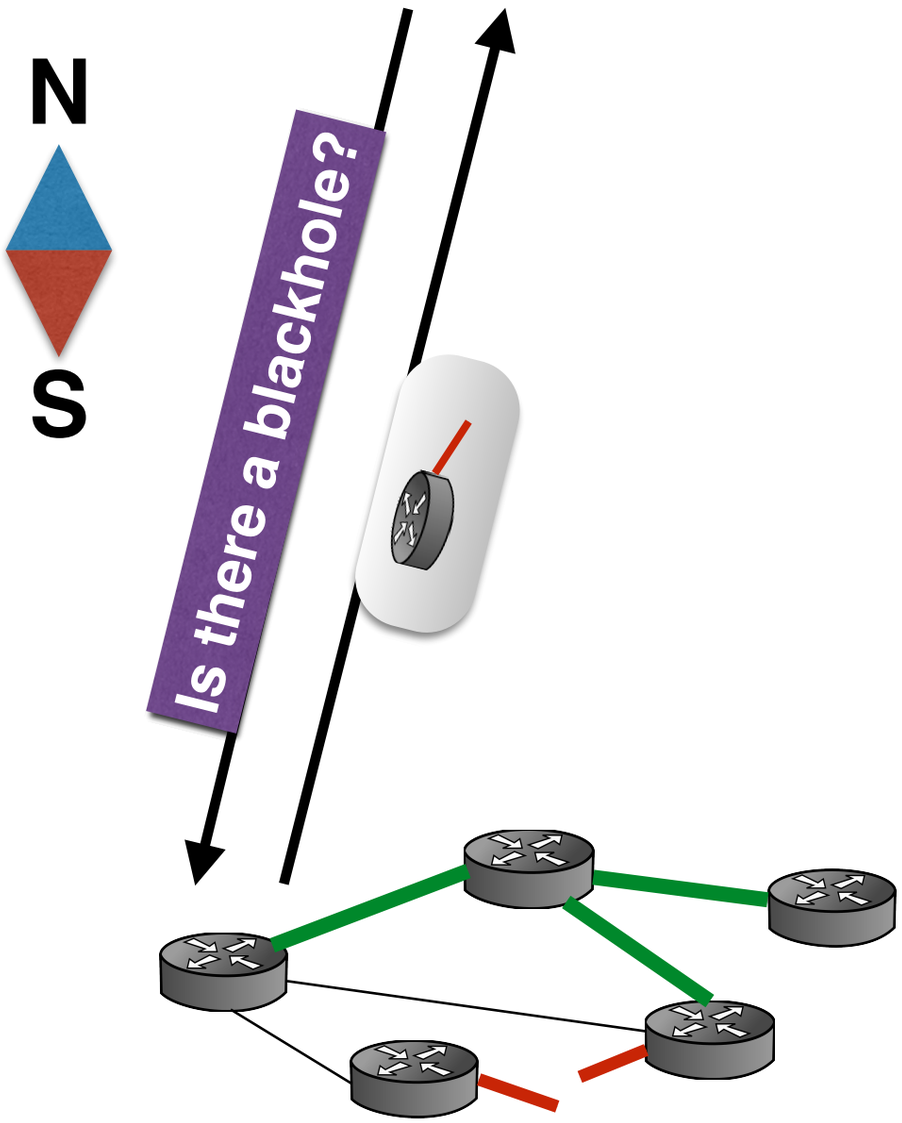
Functions in the *South:* ***Blackhole Detection***



- Detects connectivity loss regardless of the cause

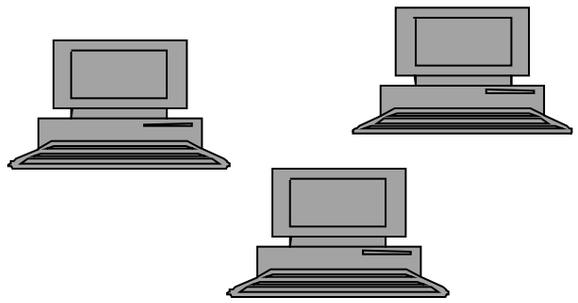
Functions in the *South:* ***Blackhole Detection***

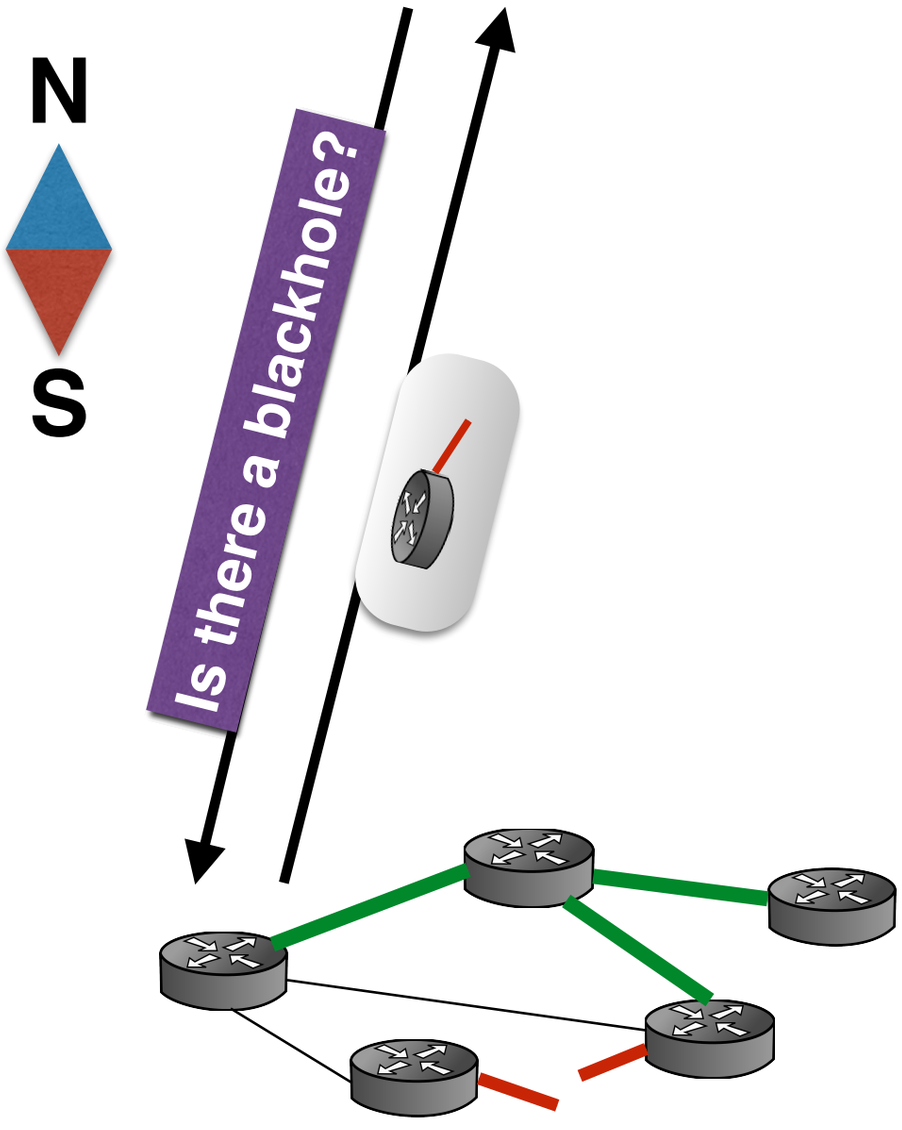




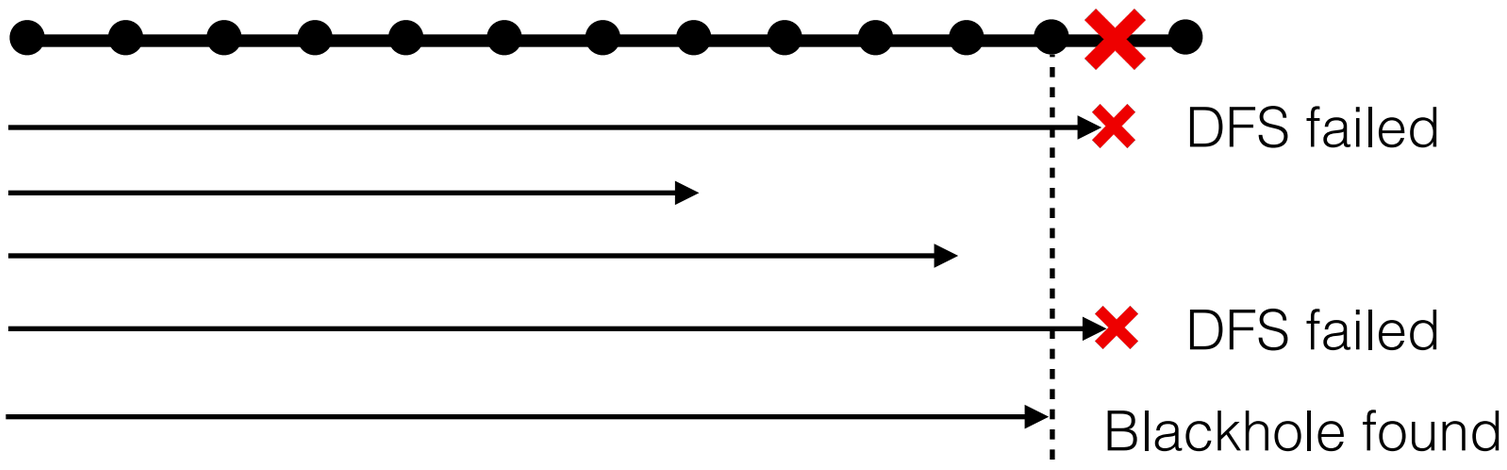
- Detects connectivity loss regardless of the cause
- Two possible implementations:

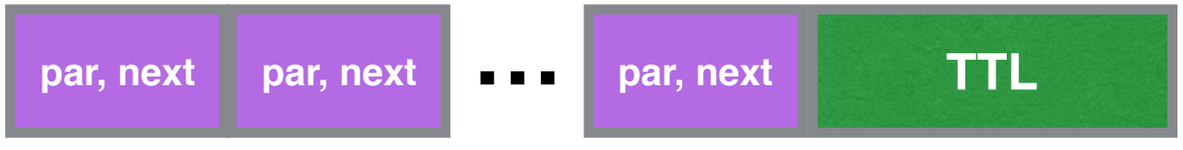
Functions in the *South:* ***Blackhole Detection***





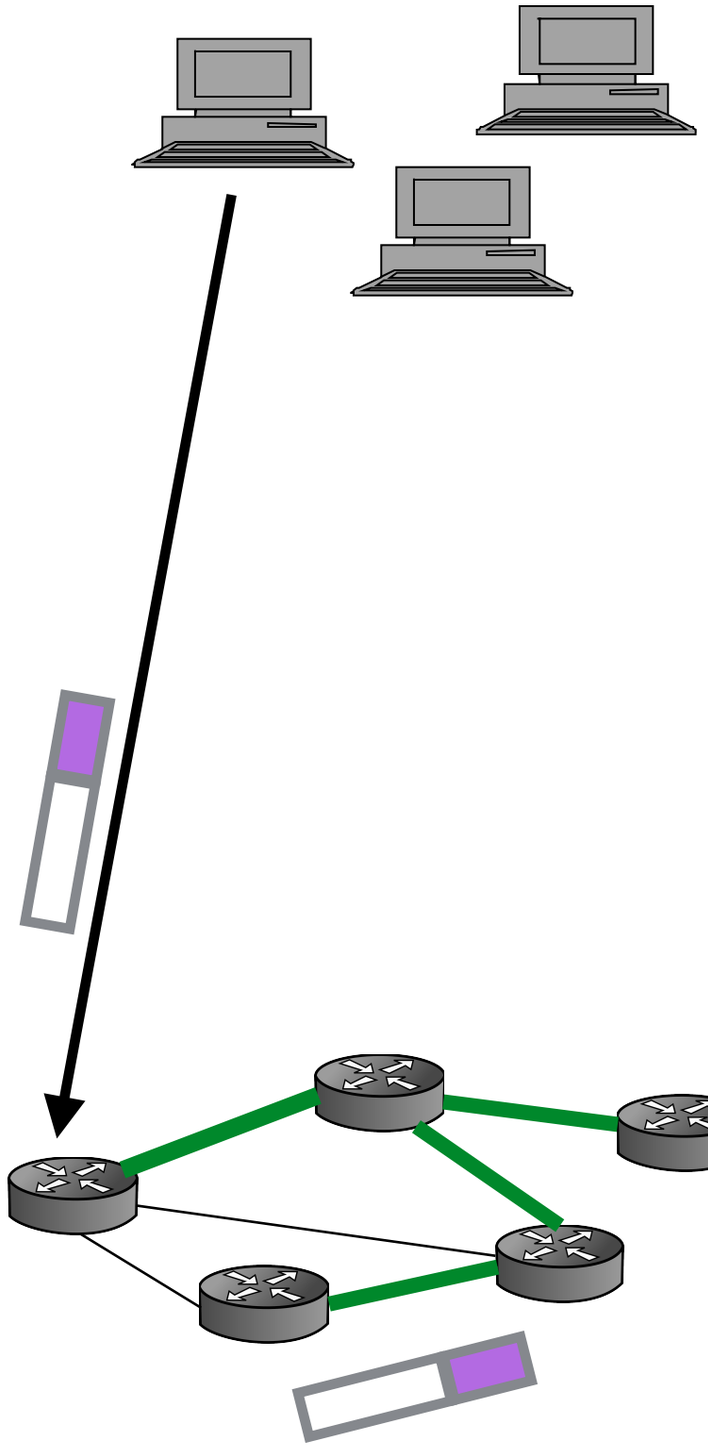
- Detects connectivity loss regardless of the cause
- Two possible implementations:
  - ***DFS traversal with TTL***
    - (log n) DFS traversals (binary search)





How it is possible? ***SmartSouth*** template.

* Based on in-band graph DFS traversal.
* Implemented using a simple match-action paradigm
* **Uses Fast Failover technique.**



How it is possible? ***SmartSouth*** template.

- Based on in-band graph DFS traversal.
- Implemented using a simple match-action paradigm
- **Uses Fast Failover technique.**

Algorithm 1 Algorithm ***SmartSouth*** – Template

Input: current node: v_i, input port: in, packet global params: $pkt.start$, packet tag array: $\{pkt.v_j\}_{j\in[n]}$
Output: output port: out

1: **if** $pkt.start = 0$ **then**
2: $pkt.start \leftarrow 1$
3: $out \leftarrow 1$
4: **else**
5: **if** $pkt.v_i.cur = 0$ **then**
6: $pkt.v_i.par \leftarrow in$; $out \leftarrow 1$; $First\_visit()$
7: **else if** $in = pkt.v_i.cur$ **then**
8: $out \leftarrow pkt.v_i.cur + 1$; $Visit\_from\_cur()$
9: **else**
10: $out \leftarrow in$; $Visit\_not\_from\_cur()$
11: **goto** 26
12: **if** $out = \Delta_i + 1$ **then**
13: $out \leftarrow pkt.v_i.par$
14: **goto** 22

15: **while** out failed **or** $out = pkt.v_i.par$ **do**
16: $out \leftarrow out + 1$
17: **if** $out = \Delta_i + 1$ **then**
18: $out \leftarrow pkt.v_i.par$
19: **goto** 22

20: $Send\_next\_neighbor()$
21: **goto** 23
22: $Send\_parent()$
23: $pkt.v_i.cur \leftarrow out$
24: **if** $out = 0$ **then**
25: $Finish()$
26: **return** out

Flow Table B

Match			Instructions
in	$pkt.v_i.cur$	$pkt.v_i.par$	
1	0	$*$	$pkt.v_i.par \leftarrow in$, Table 2
1	1	2	Table 3
2	2	3	Table 4
3	3	4	Table 5
...	...	...	...
$\Delta_i - 2$	$\Delta_i - 2$	$\Delta_i - 1$	Table Δ_i
$\Delta_i - 1$	$\Delta_i - 1$	Δ_i	Table C
$*$	0	$*$	$pkt.v_i.par \leftarrow in$, Table 1
1	1	$*$	Table 2
2	2	$*$	Table 3
3	3	$*$	Table 4

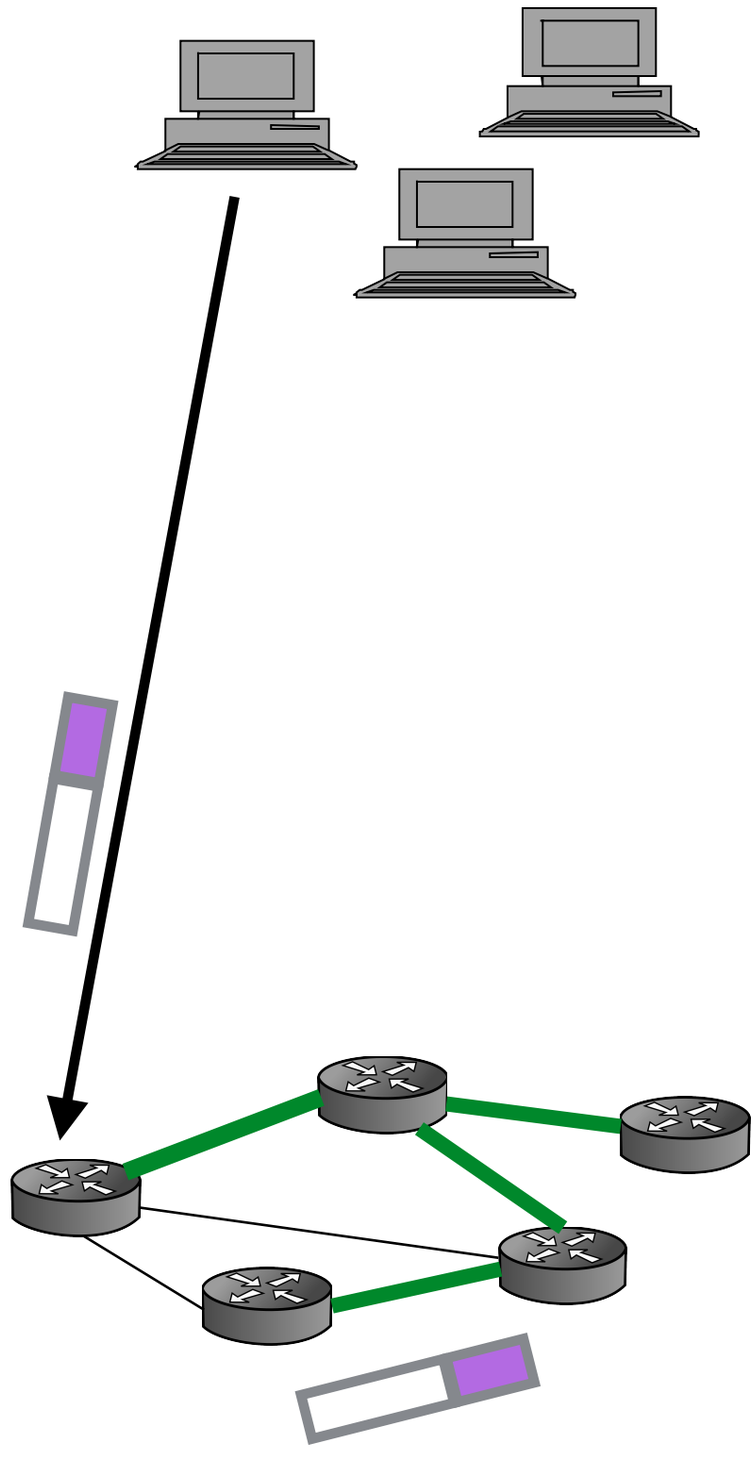
Flow Table A (Start)

Match		Instructions
$pkt.start$	dst	
0	1	Gr 0.1, Table 1
0	2	Gr 0.2, Table 1
...	...	...
0	n	Gr 0.n, Table 1
$*$	$*$	Table B

Flow Table C (Send-Parent)

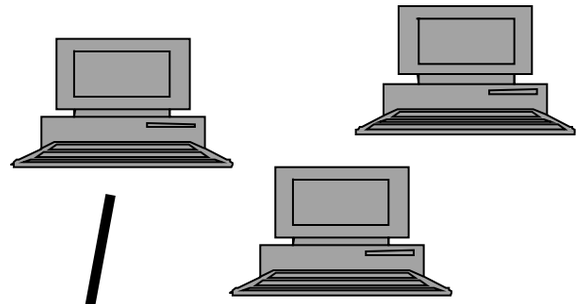
Match	Instructions
$pkt.v_i.par$	
1	Fwd 1
2	Fwd 2
...	...
Δ_i	Fwd Δ_i

Group	Actions
Gr 0.1	$\langle sb \leftarrow 1, \text{Fwd } Route(1)\rangle$
Gr 0.2	$\langle sb \leftarrow 1, \text{Fwd } Route(2)\rangle$
...	...
Gr 0.n	$\langle sb \leftarrow 1, \text{Fwd } Route(n)\rangle$
Gr 1	$\langle sb \leftarrow 1, pkt.v_i.cur \leftarrow 1, pkt.start \leftarrow 1, \text{Fwd } 1\rangle$
Gr 2	$\langle sb \leftarrow 1, pkt.v_i.cur \leftarrow 2, pkt.start \leftarrow 1, \text{Fwd } 2\rangle$
...	...
Gr Δ_i	$\langle sb \leftarrow 1, pkt.v_i.cur \leftarrow \Delta_i, pkt.start \leftarrow 1, \text{Fwd } \Delta_i\rangle$



How it is possible? ***SmartSouth*** template.

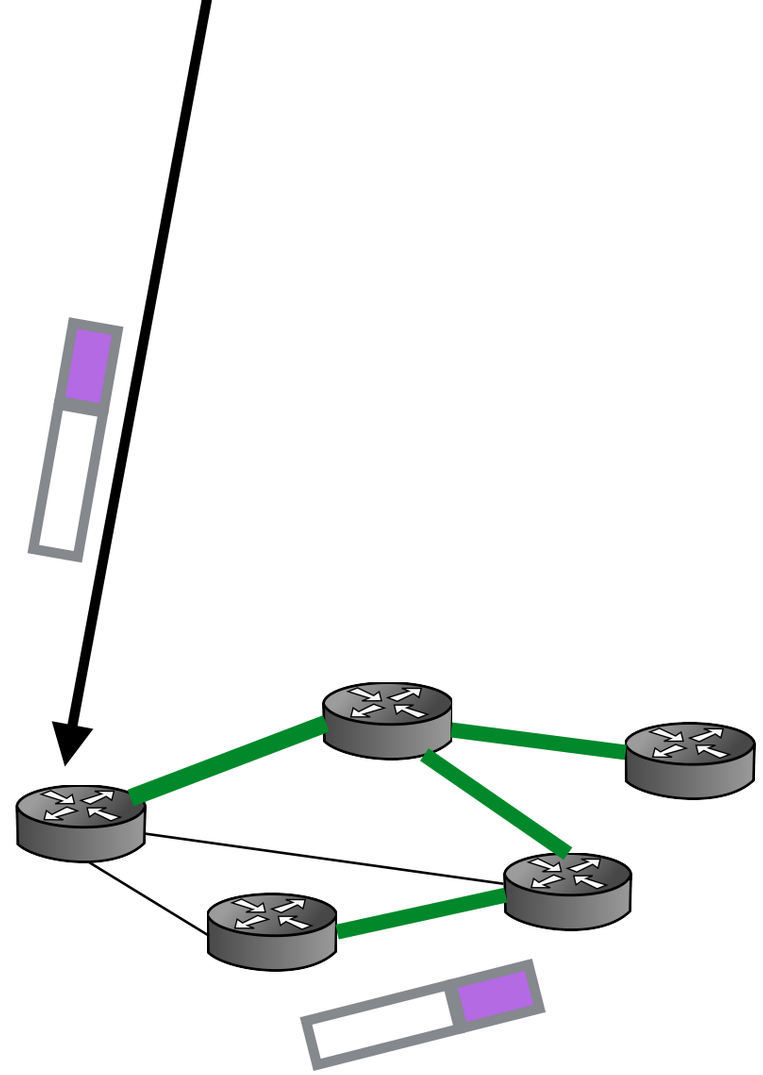
- Based on in-band graph DFS traversal.
- Implemented using a simple match-action paradigm
- **Uses Fast Failover technique.**



$out \leftarrow in$; $Visit\_not\_from\_cur()$
goto 26
if $out = \Delta_i + 1$ **then**
$out \leftarrow pkt.v_i.par$
goto 22

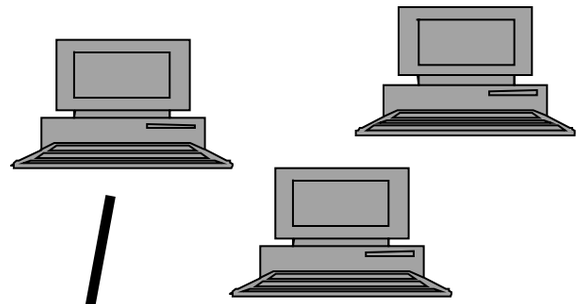
Flow Table A (Start)

Match		Instructions
pkt.start	*dst*	
0	1	Gr 0.1, Table 1
0	2	Gr 0.2, Table 1
...	...	...
0	n	Gr 0.n, Table 1
$*$	$*$	Table B



How it is possible? ***SmartSouth*** template.

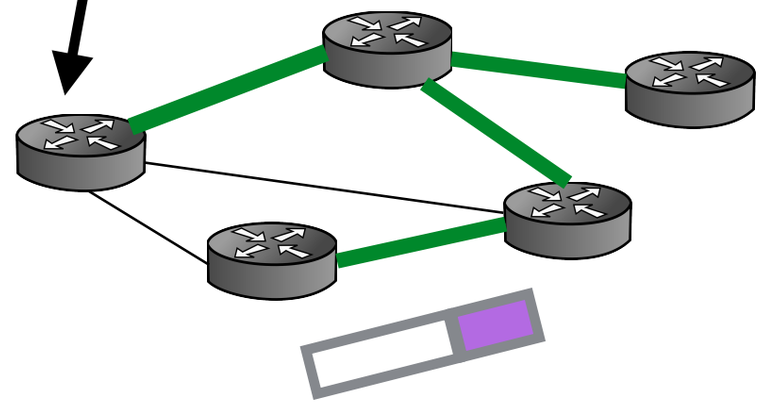
- Based on in-band graph DFS traversal.
- Implemented using a simple match-action paradigm
- **Uses Fast Failover technique.**

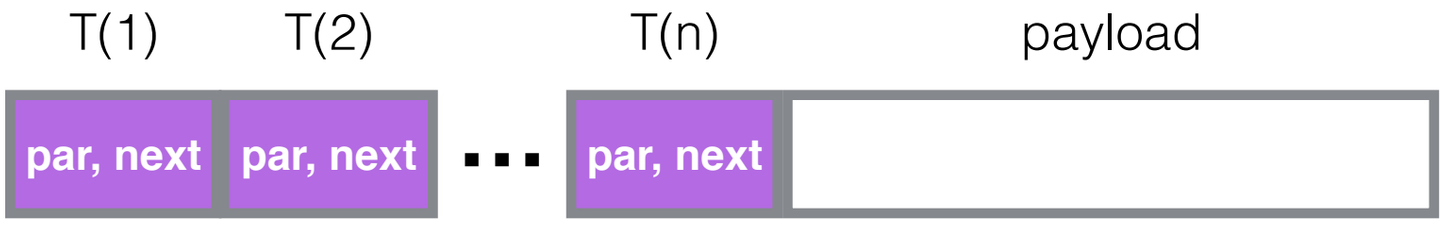


$out \leftarrow in$; $Visit\_not\_from\_cur()$
goto 26
if $out = \Delta_i + 1$ **then**
$out \leftarrow pkt.v_i.par$
goto 22

Flow Table A (Start)

Match		Instructions
pkt.start	*dst*	
0	1	Gr 0.1, Table 1
0	2	Gr 0.2, Table 1
...	...	...
0	*n*	Gr 0.n, Table 1
*	*	Table B

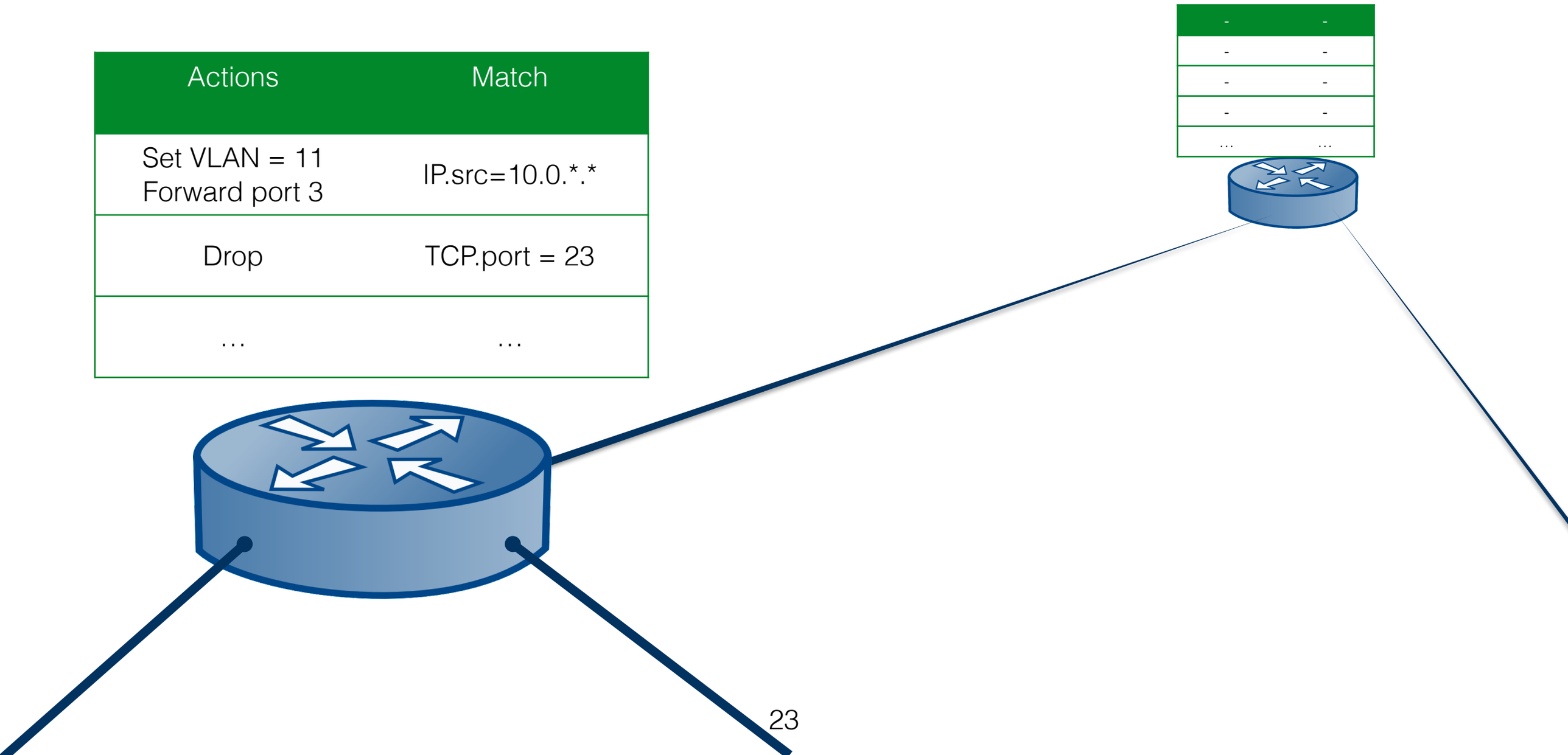




Data Plane - Match & Action

Actions	Match
Set VLAN = 11 Forward port 3	IP.src=10.0.*.*
Drop	TCP.port = 23
…	…

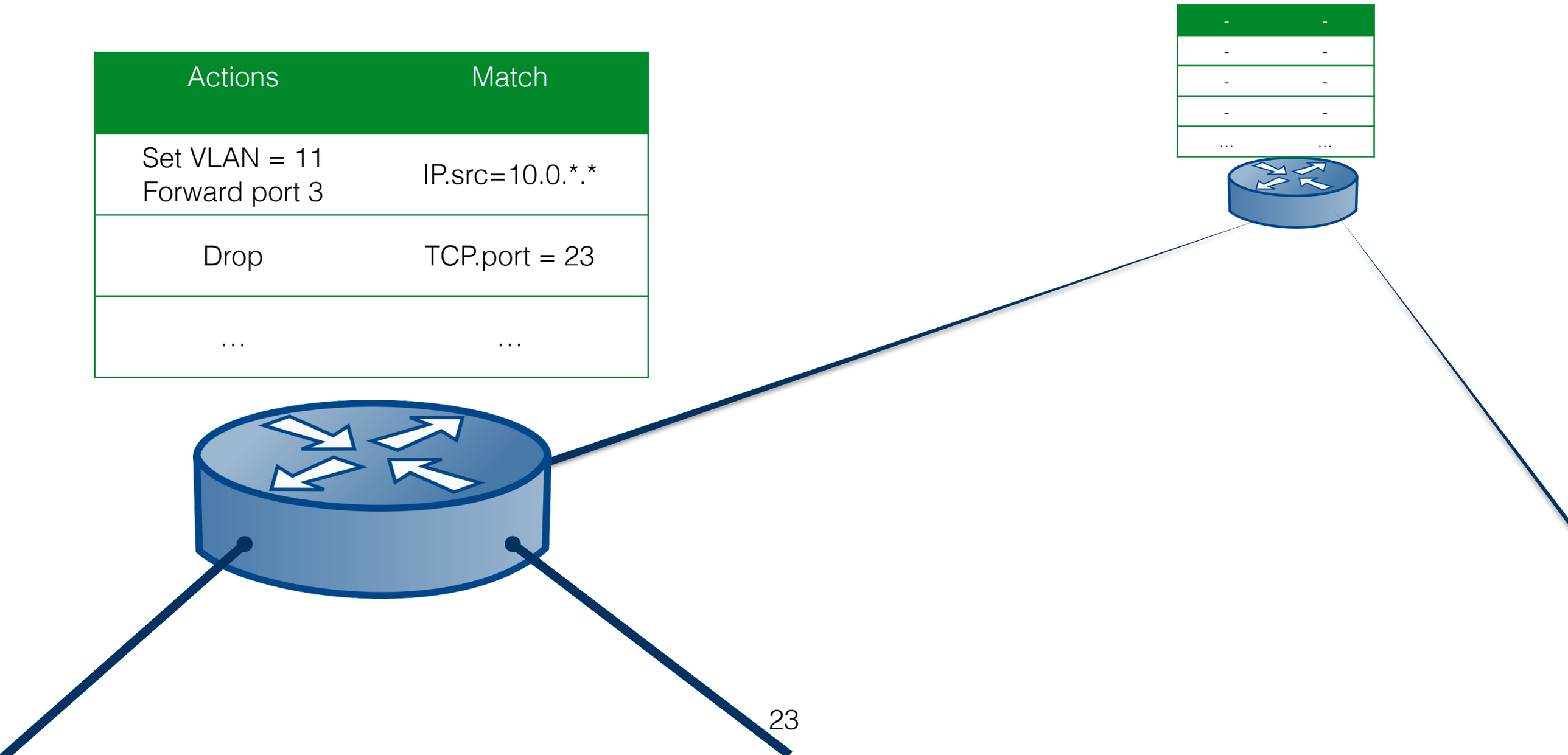
-	-
-	-
-	-
-	-
…	…



Data Plane - Match & Action

Actions	Match
Set VLAN = 11 Forward port 3	IP.src=10.0.*.*
Drop	TCP.port = 23
…	…

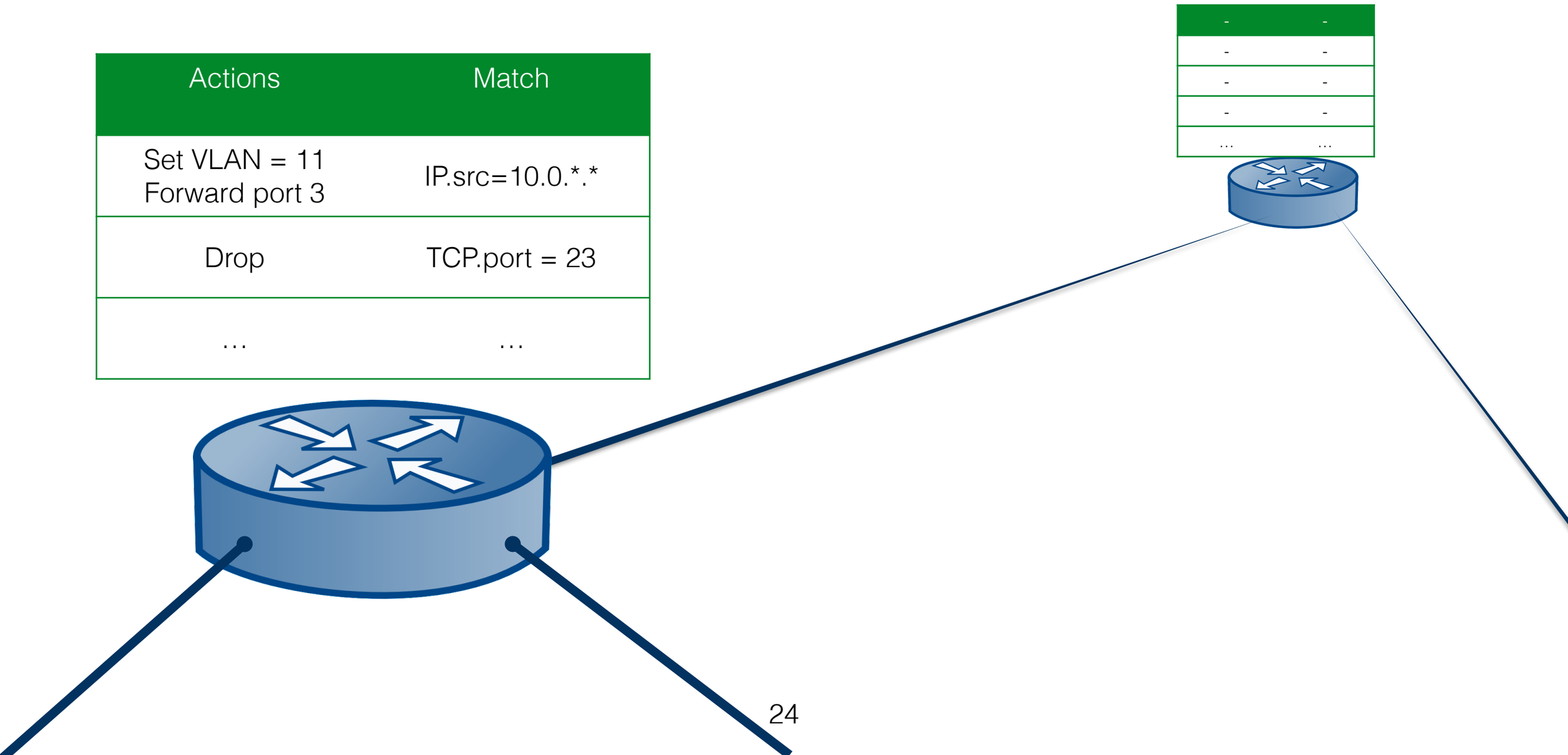
-	-
-	-
-	-
-	-
…	…



Distributed Control Plane

Actions	Match
Set VLAN = 11 Forward port 3	IP.src=10.0.*.*
Drop	TCP.port = 23
…	…

-	-
-	-
-	-
-	-
…	…



Distributed Control Plane

Actions	Match
Set VLAN = 11 Forward port 3	IP.src=10.0.*.*
Drop	TCP.port = 23
…	…

-	-
-	-
-	-
-	-
…	…