Intent Based Networking - the technology

Jeff Tantsura
Head of Networking Strategy @Apstra

რასაცა გასცემ შენია, რას არა
დაკარგულია

That which we give makes us richer,
that which is hoarded is lost

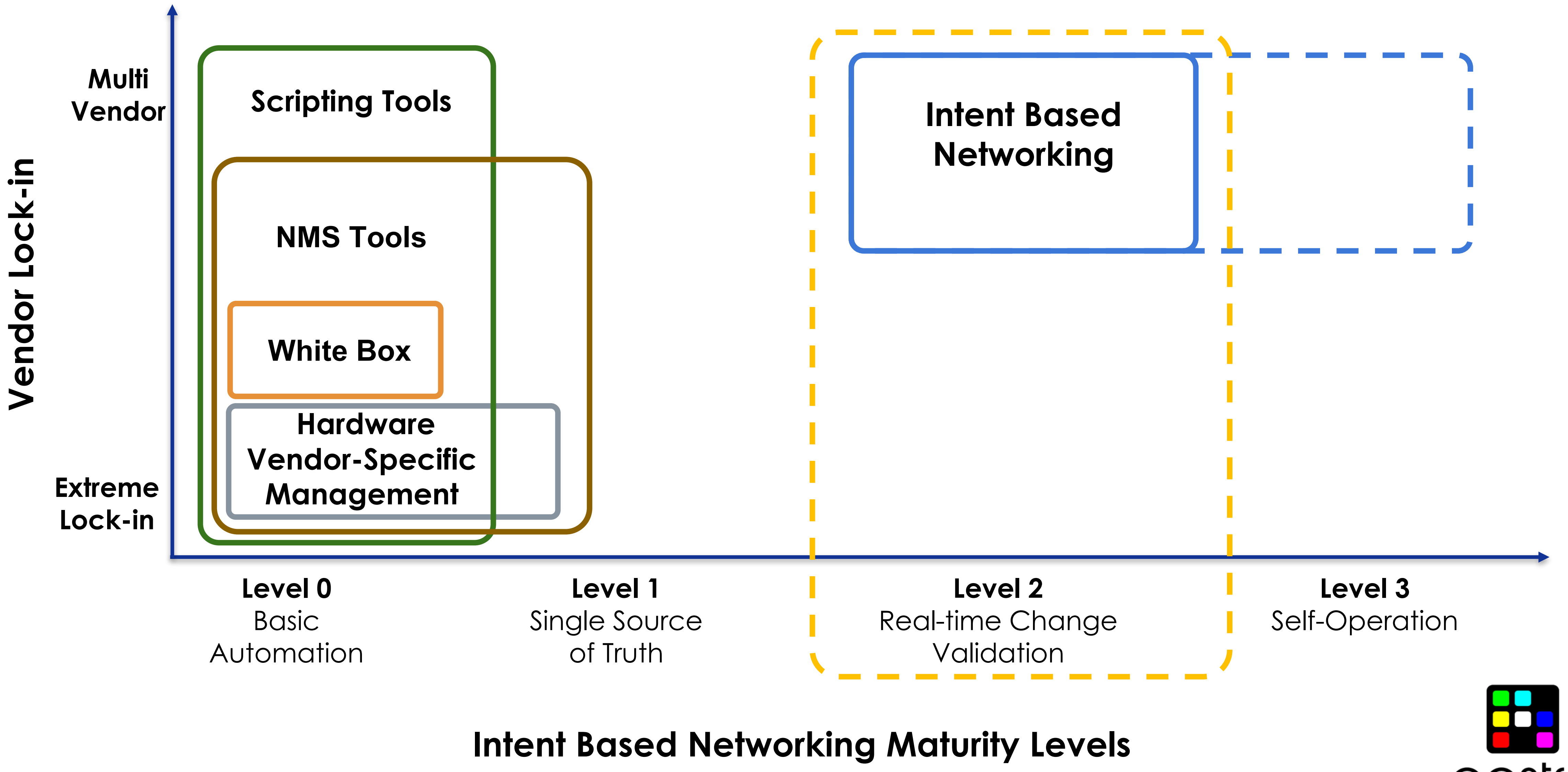
apstra

IBN Landscape

It's a Fan!
It's a Wall!
It's a Spear!
It's a Rope!
It's a Snake!
It's a Tree!

apstra

IBN Landscape



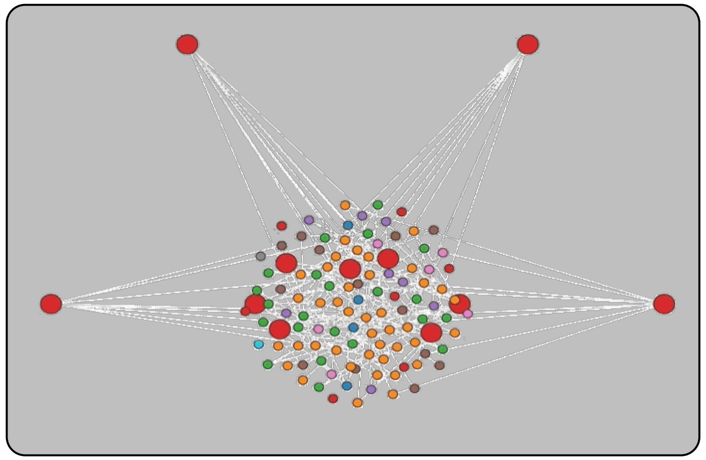
apstra

IBN Design Philosophy

Networks managed as a whole system, not individual components

Successful networks are defined by the outcomes produced by the whole system

Intent Based Networking is about "*what*" not "*how*"



apstra

IBN life cycle

Design	Build	Deploy	Validate
Intent consumption	Intent modeling	Intent instantiation	Intent validation (continuous)
Tell me what you want (your intent)	**Let me model/build the logical intent model**	**Let me instantiate your intent (networking)**	**Let me validate that the network still does it as intended**

Architectural Goals of IBN

Problems to be solved:

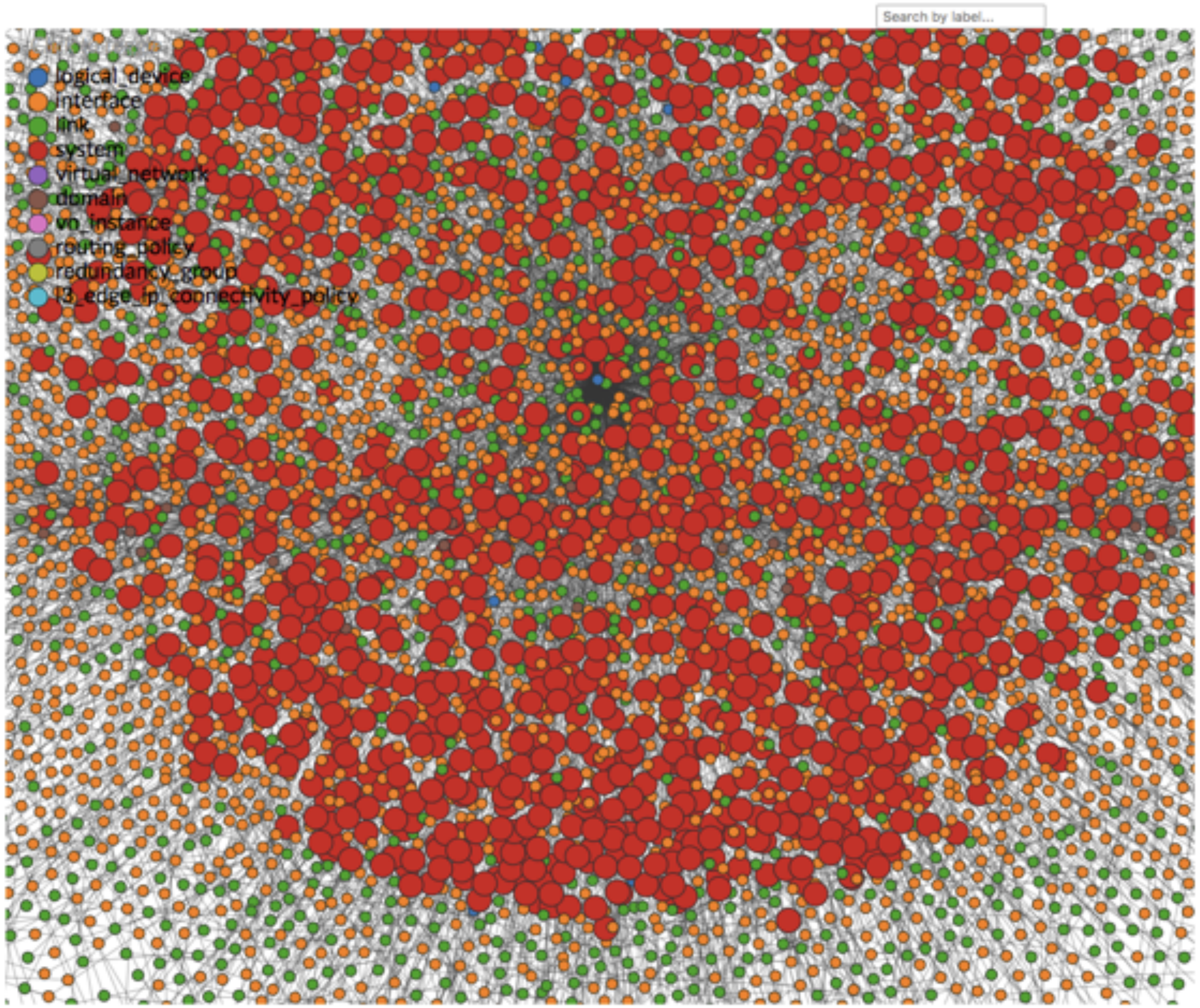
- Composition/decomposition @scale
- Dealing with changes:
  - Planned change – can I achieve desired (future) state while preserving original intent
  - Unplanned change – impact of the change, difference between intended and operational states, how to get to intended state

Architectural Goals of IBN

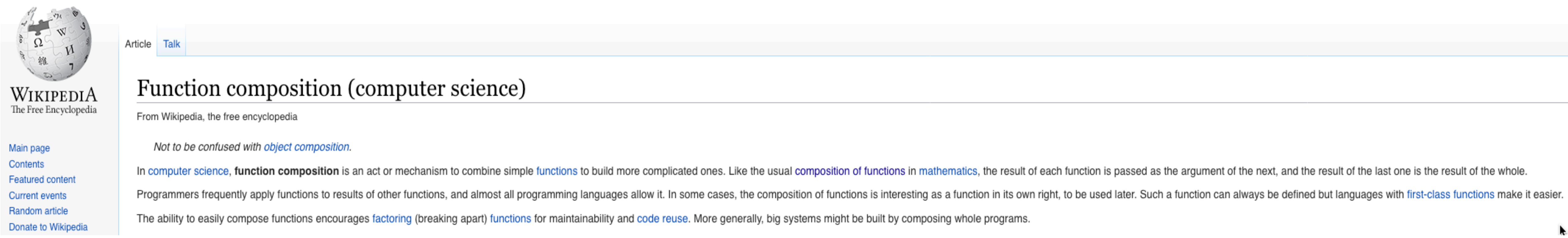
Problems to be solved:

- Closed loop validation:
  - continuously validate *outcomes* against the *intent* to ensure that the *composition* is working as intended
  - extract more knowledge by collecting less data (IBA)
  - highly optimized SNR (signal to noise ratio) in analytics

Dealing With Scale?



Composition



Article Talk

Function composition (computer science)

From Wikipedia, the free encyclopedia

Not to be confused with object composition.

In computer science, **function composition** is an act or mechanism to combine simple functions to build more complicated ones. Like the usual composition of functions in mathematics, the result of each function is passed as the argument of the next, and the result of the last one is the result of the whole.

Programmers frequently apply functions to results of other functions, and almost all programming languages allow it. In some cases, the composition of functions is interesting as a function in its own right, to be used later. Such a function can always be defined but languages with first-class functions make it easier.

The ability to easily compose functions encourages factoring (breaking apart) functions for maintainability and code reuse. More generally, big systems might be built by composing whole programs.

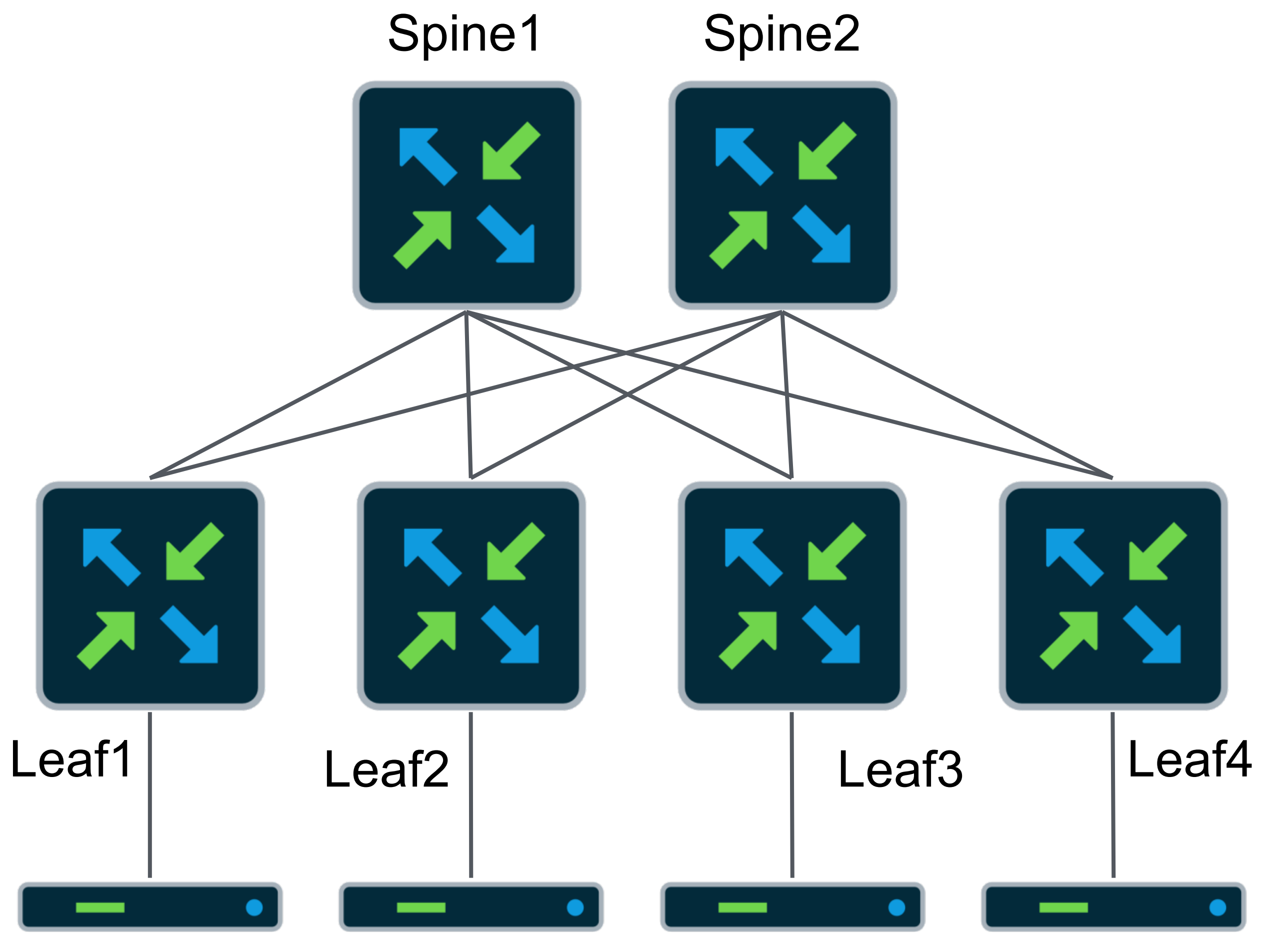
Why model a graph?

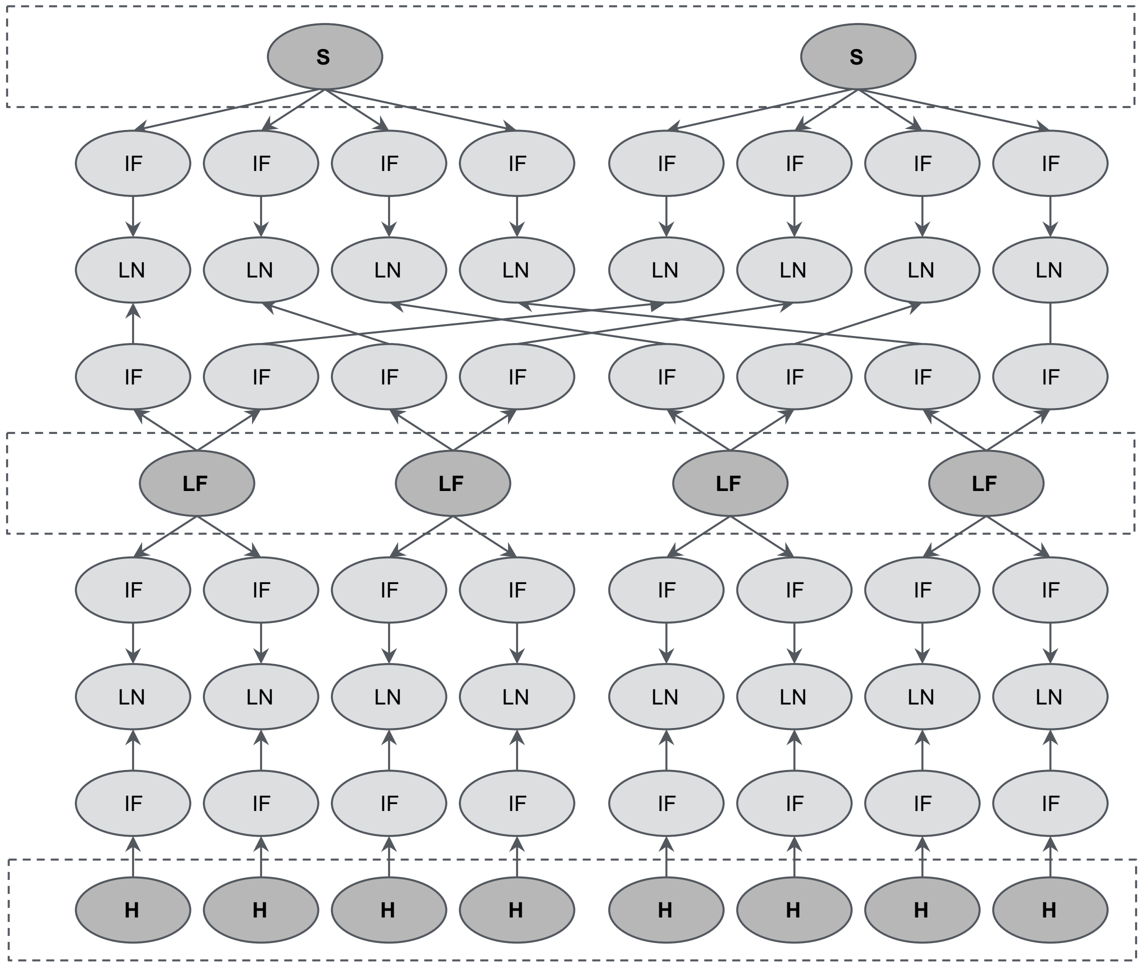
- Networks are intuitively the connected set of **nodes** and **relationships**
- As network requirements **change** the model can be easily **extended**
- Efficiently run **queries** that were **not anticipated** at model design time

Hint: you **will not** know all the queries at model definition time

apstra

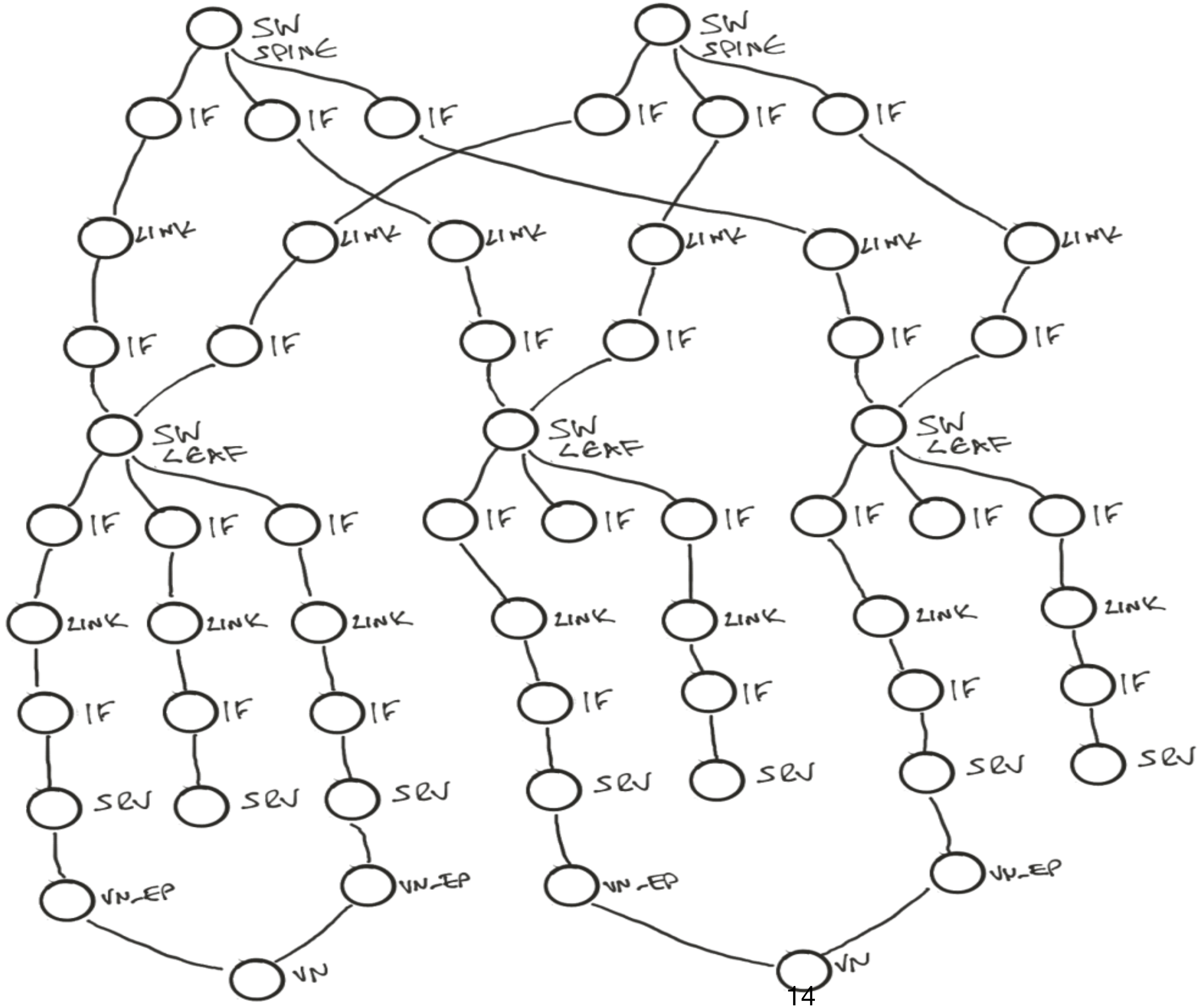
Intent-> Graph composition

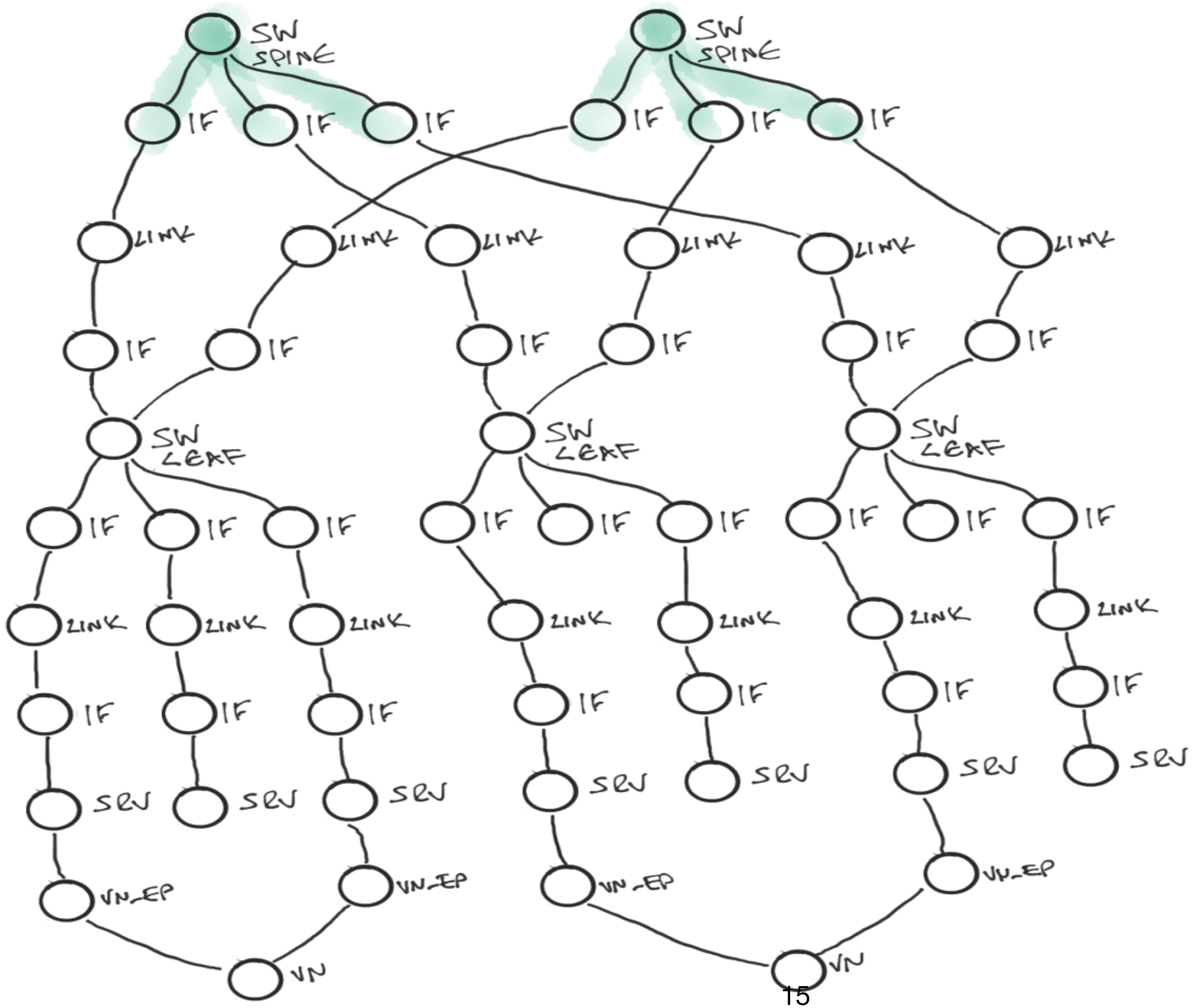




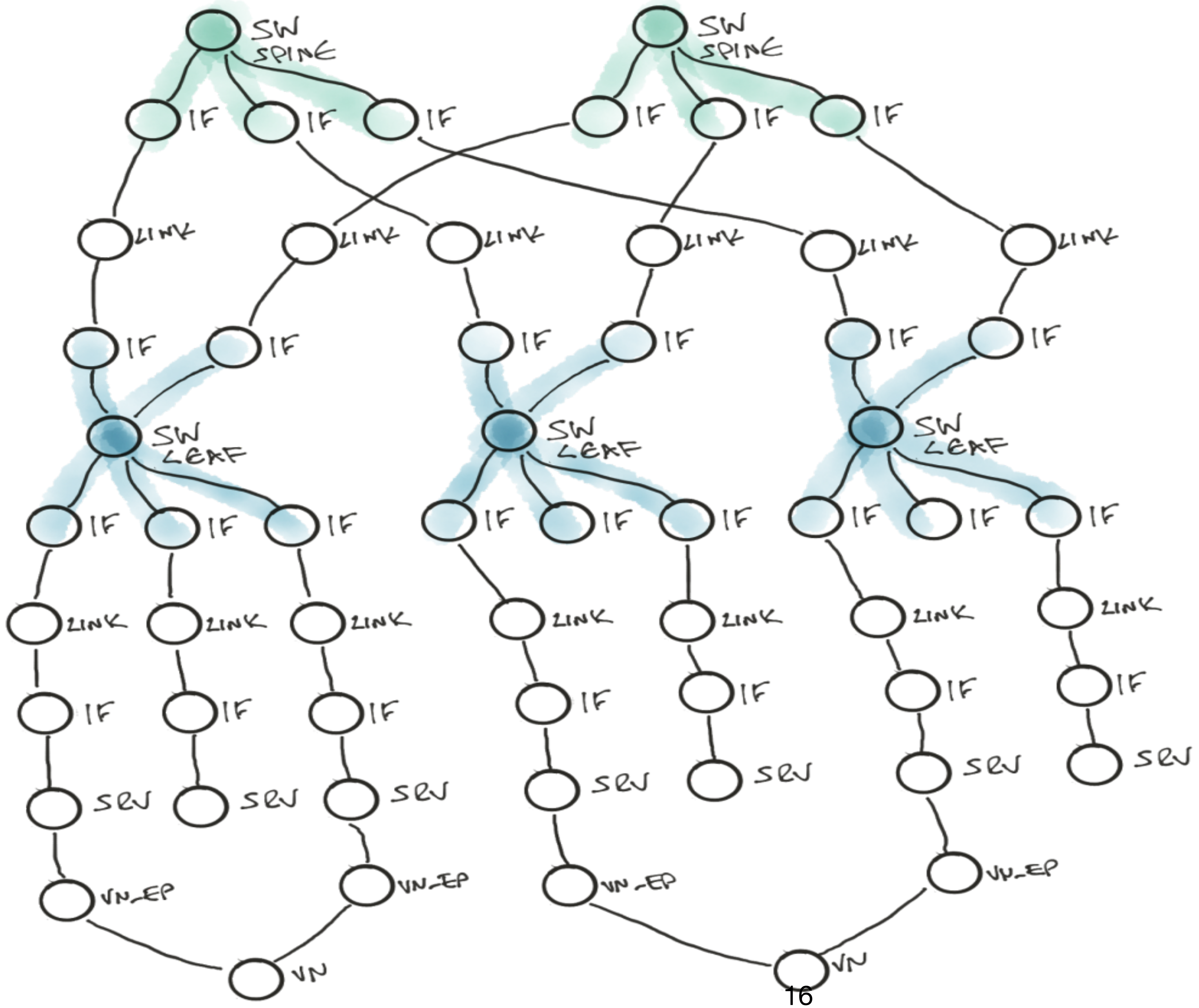
apstra

Function composition


SW SPINE
SW SPINE
IF
LINK
SW LEAF
SW LEAF
SW LEAF
SRV
VN_EP
VN
VN

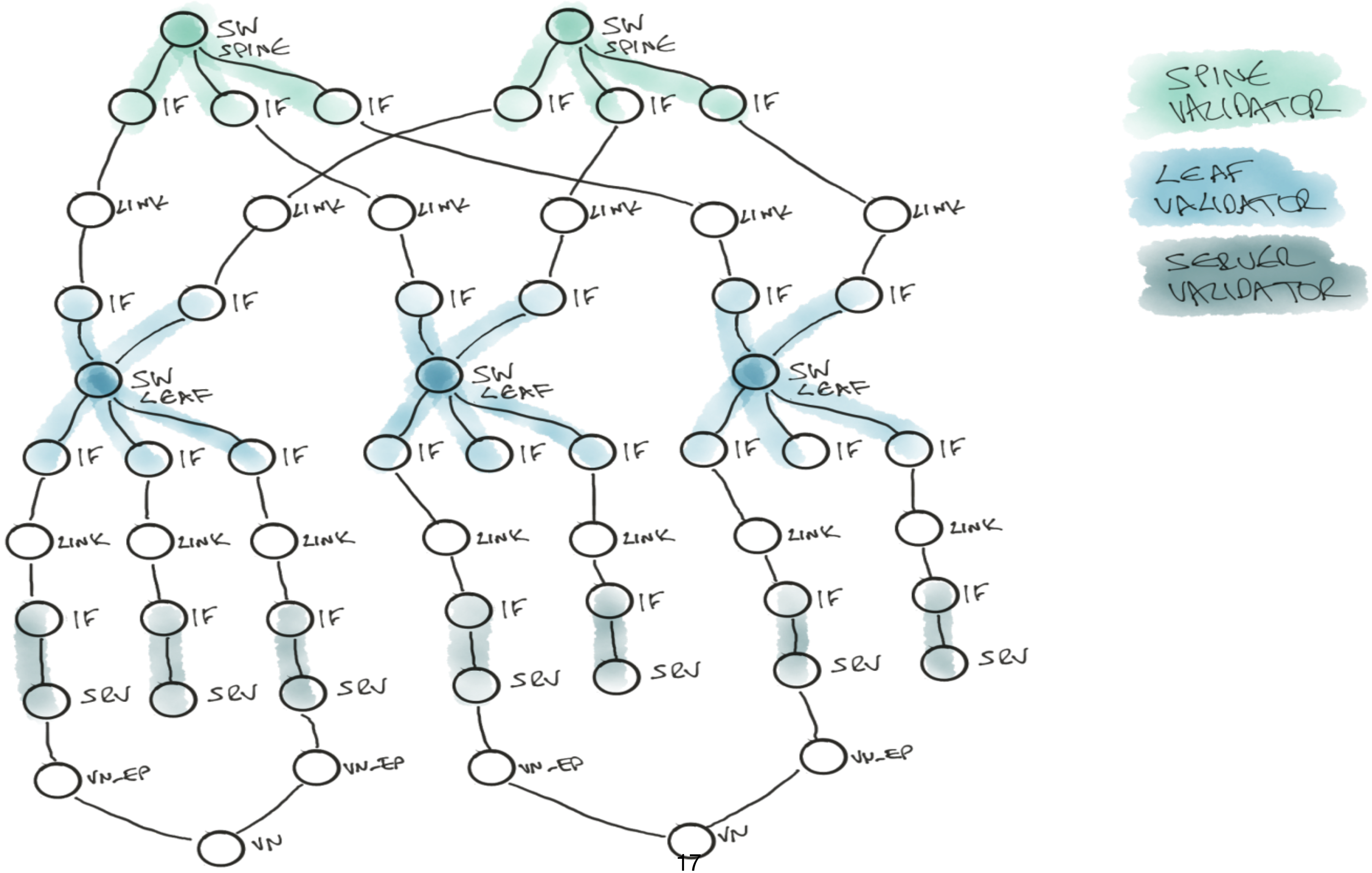


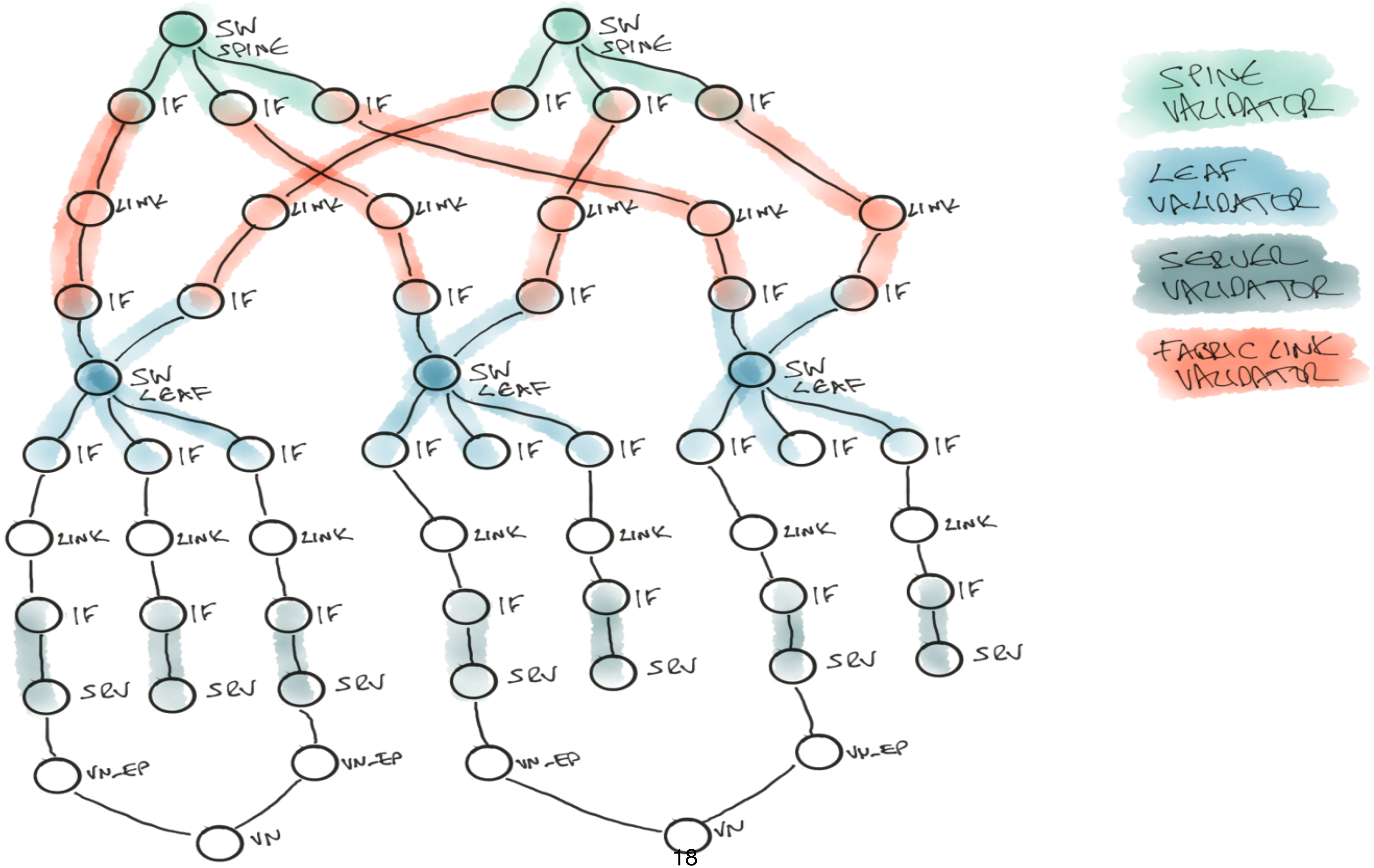
SPINE VALIDATOR

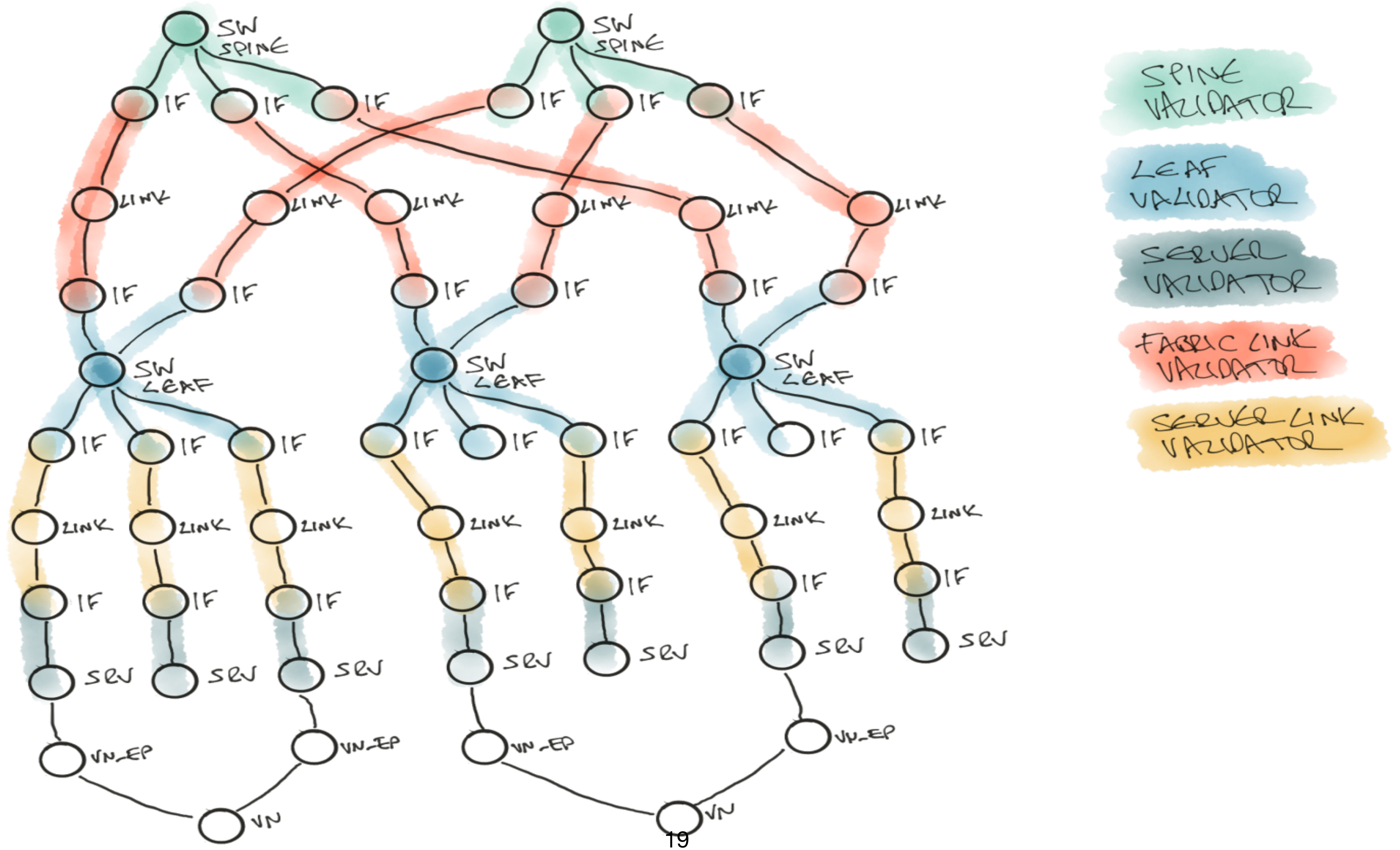


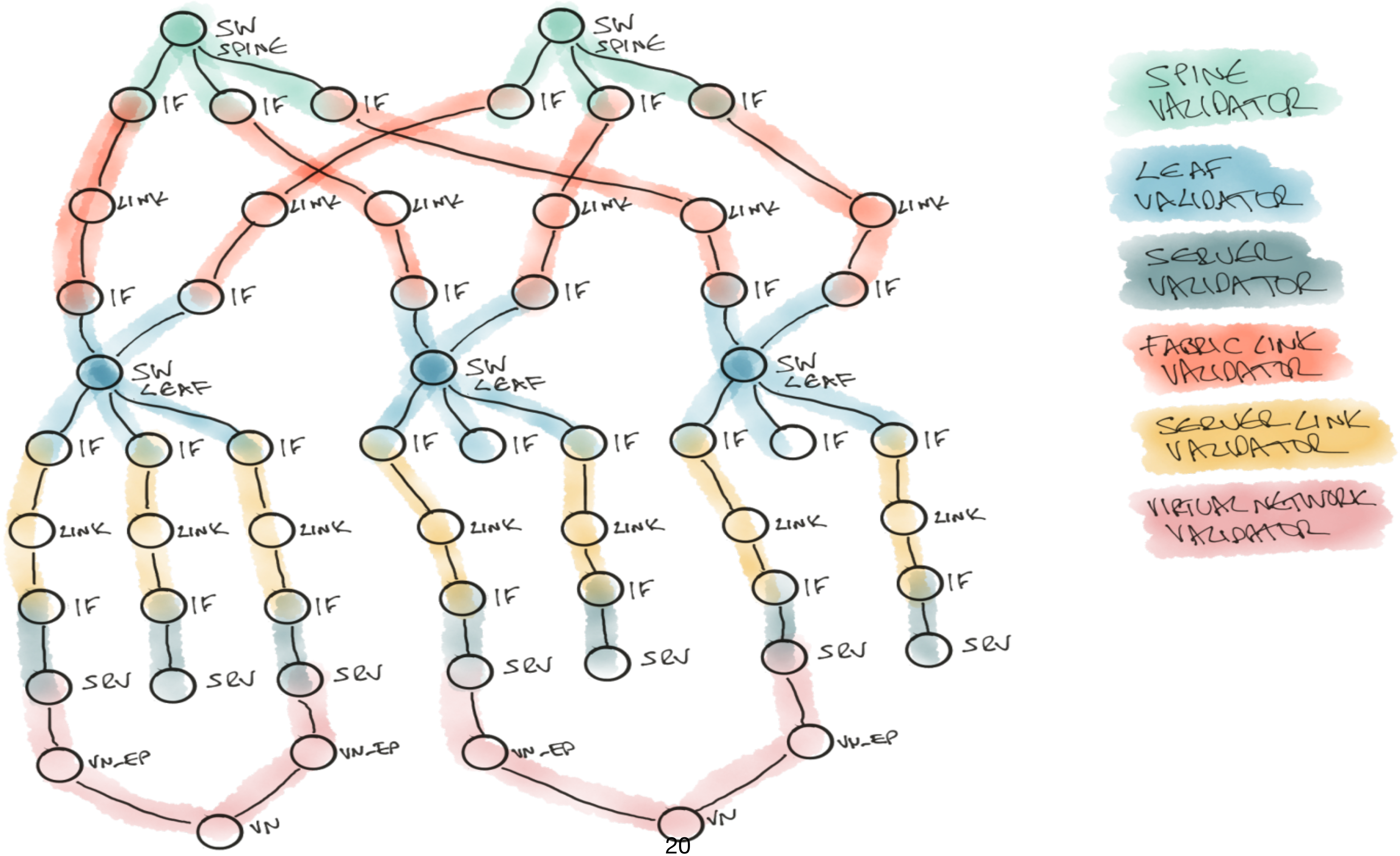
🟩 SPINE VALIDATOR

🟦 LEAF VALIDATOR

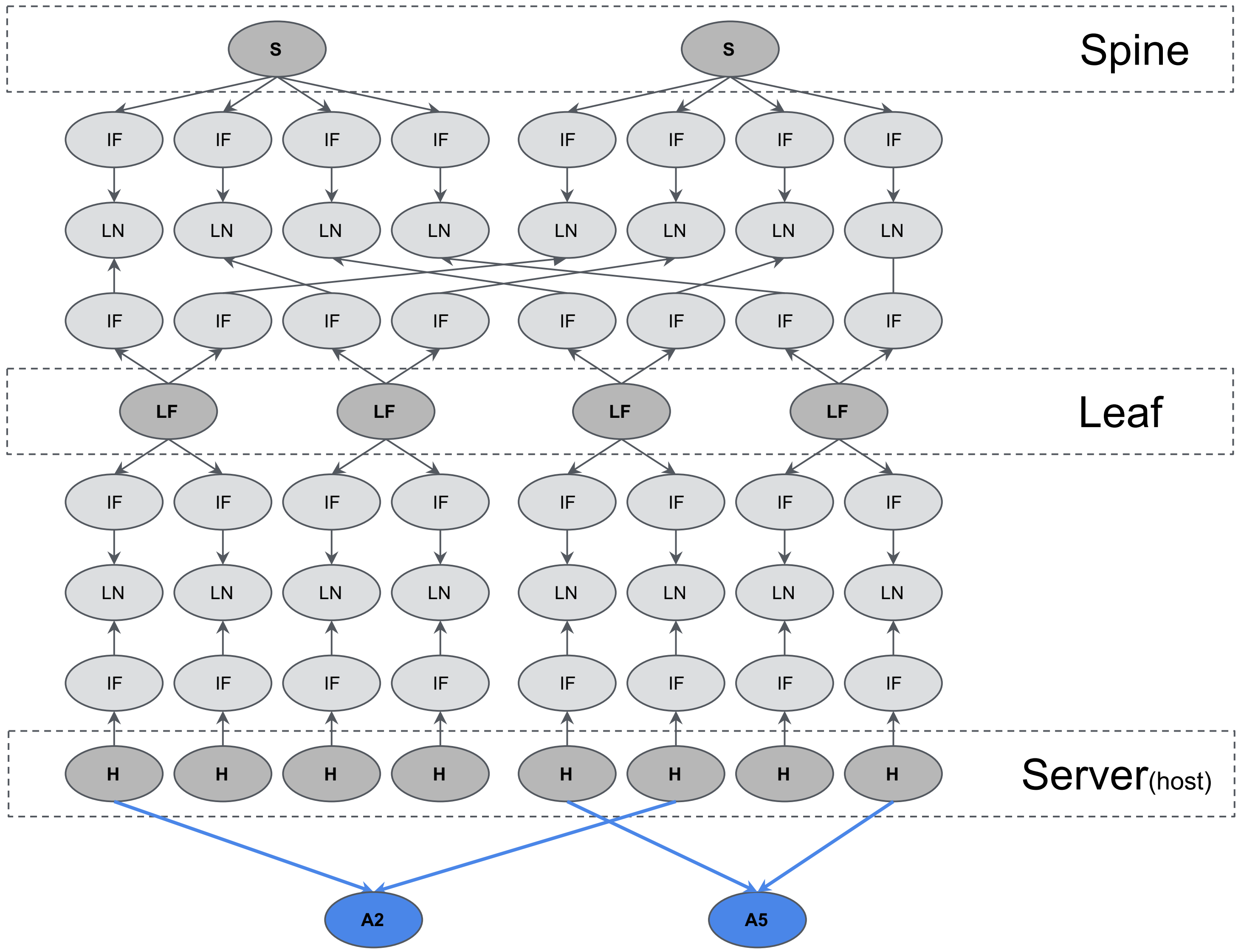

SW SPINE
SW SPINE
IF
IF
IF
IF
IF
IF
LINK
LINK
LINK
LINK
LINK
LINK
IF
IF
IF
IF
IF
IF
SW LEAF
SW LEAF
SW LEAF
IF
IF
IF
IF
IF
IF
IF
IF
IF
LINK
LINK
LINK
LINK
LINK
LINK
LINK
IF
IF
IF
IF
IF
IF
IF
SRV
SRV
SRV
SRV
SRV
SRV
SRV
VN_EP
VN_EP
VN_EP
VN_EP
VN
VN
SPINE VALIDATOR
LEAF VALIDATOR
SERVER VALIDATOR


SW SPINE
SW SPINE
IF
LINK
SW LEAF
SRV
VN_EP
VN
SPINE VALIDATOR
LEAF VALIDATOR
SERVER VALIDATOR
FABRIC LINK VALIDATOR

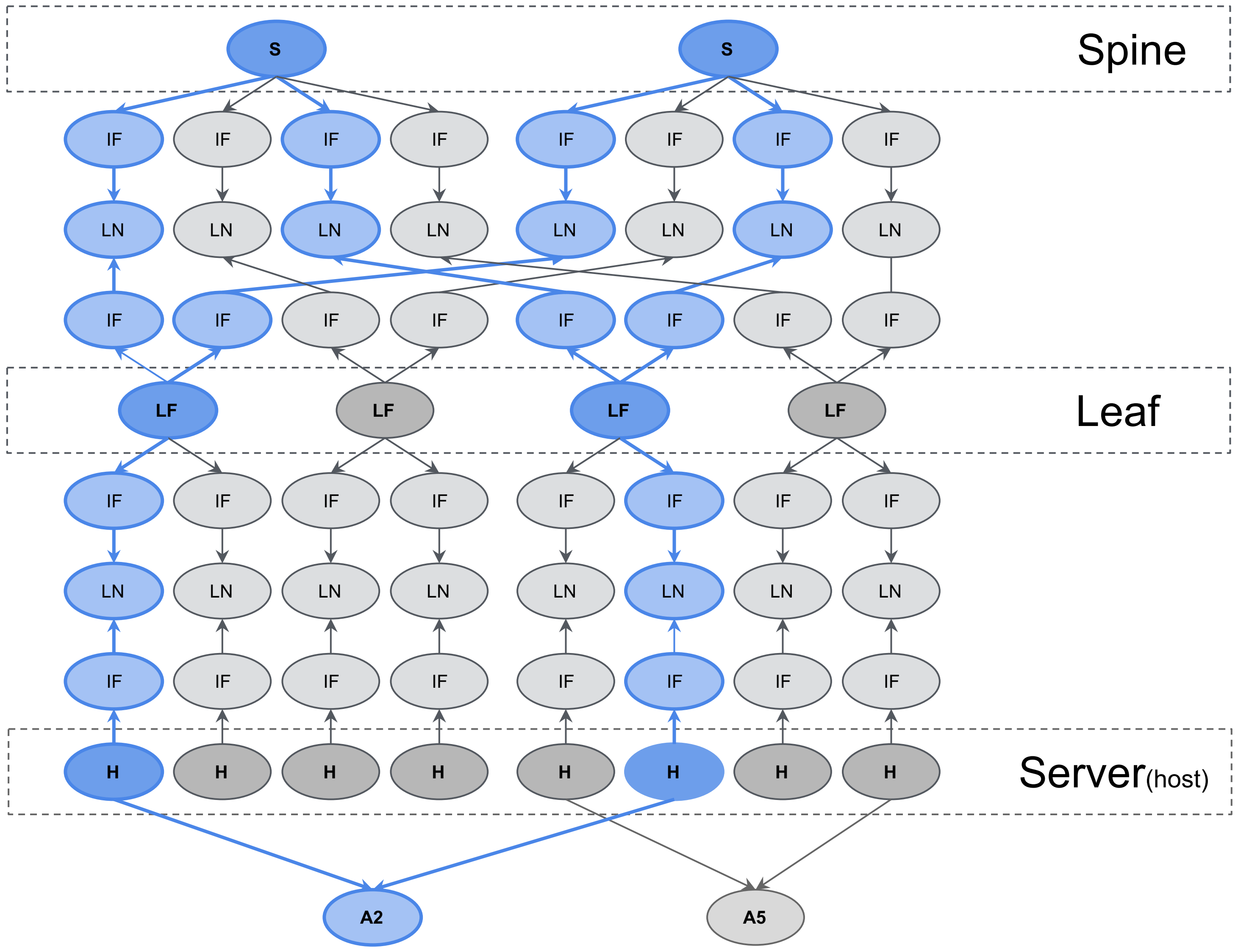

SW SPINE
SW SPINE
IF
LINK
SW LEAF
SRV
VN_EP
VN
SPINE VALIDATOR
LEAF VALIDATOR
SERVER VALIDATOR
FABRIC LINK VALIDATOR
SERVER LINK VALIDATOR


SW SPINE
SW SPINE
IF
LINK
SW LEAF
SW LEAF
SW LEAF
SRV
VN_EP
VN
VN
SPINE VALIDATOR
LEAF VALIDATOR
SERVER VALIDATOR
FABRIC LINK VALIDATOR
SERVER LINK VALIDATOR
VIRTUAL NETWORK VALIDATOR

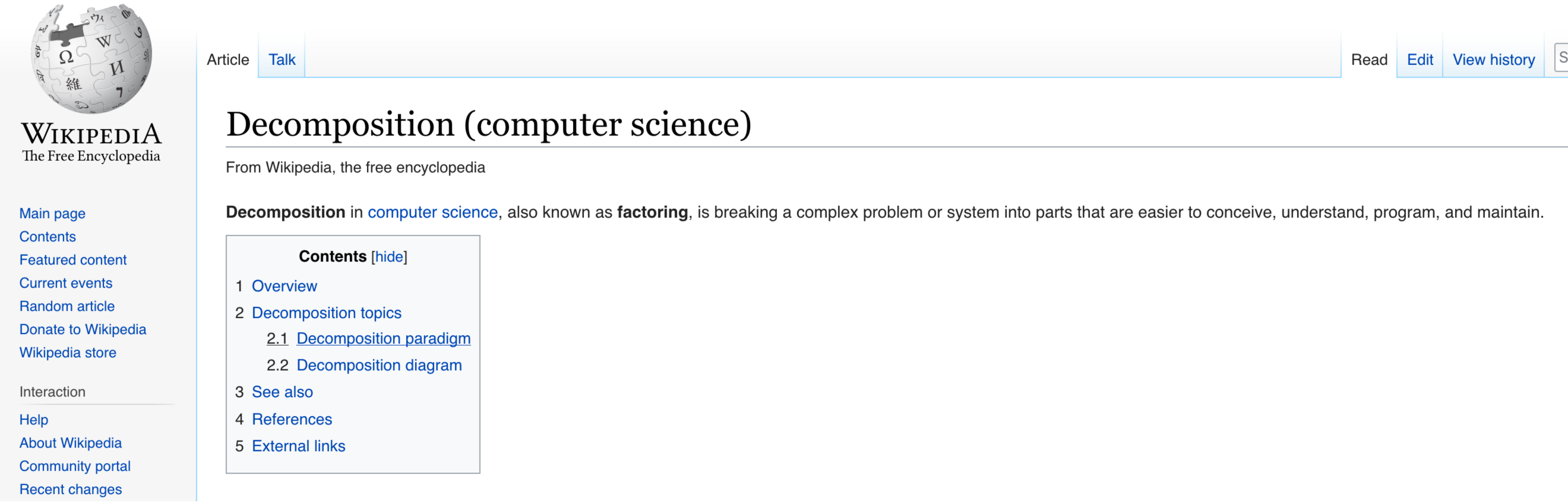
Resulting Model



Query: Links that carry app “A2” traffic



Decomposition



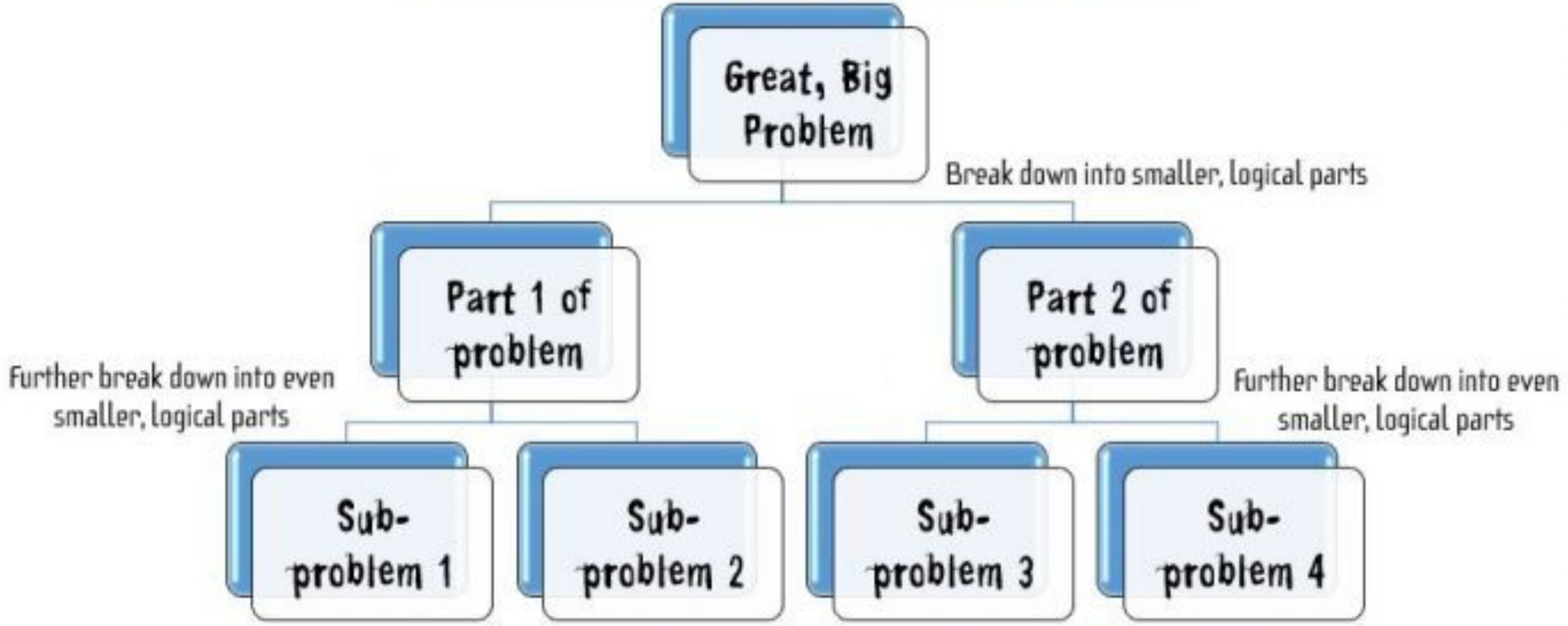
Decomposition (computer science)

From Wikipedia, the free encyclopedia

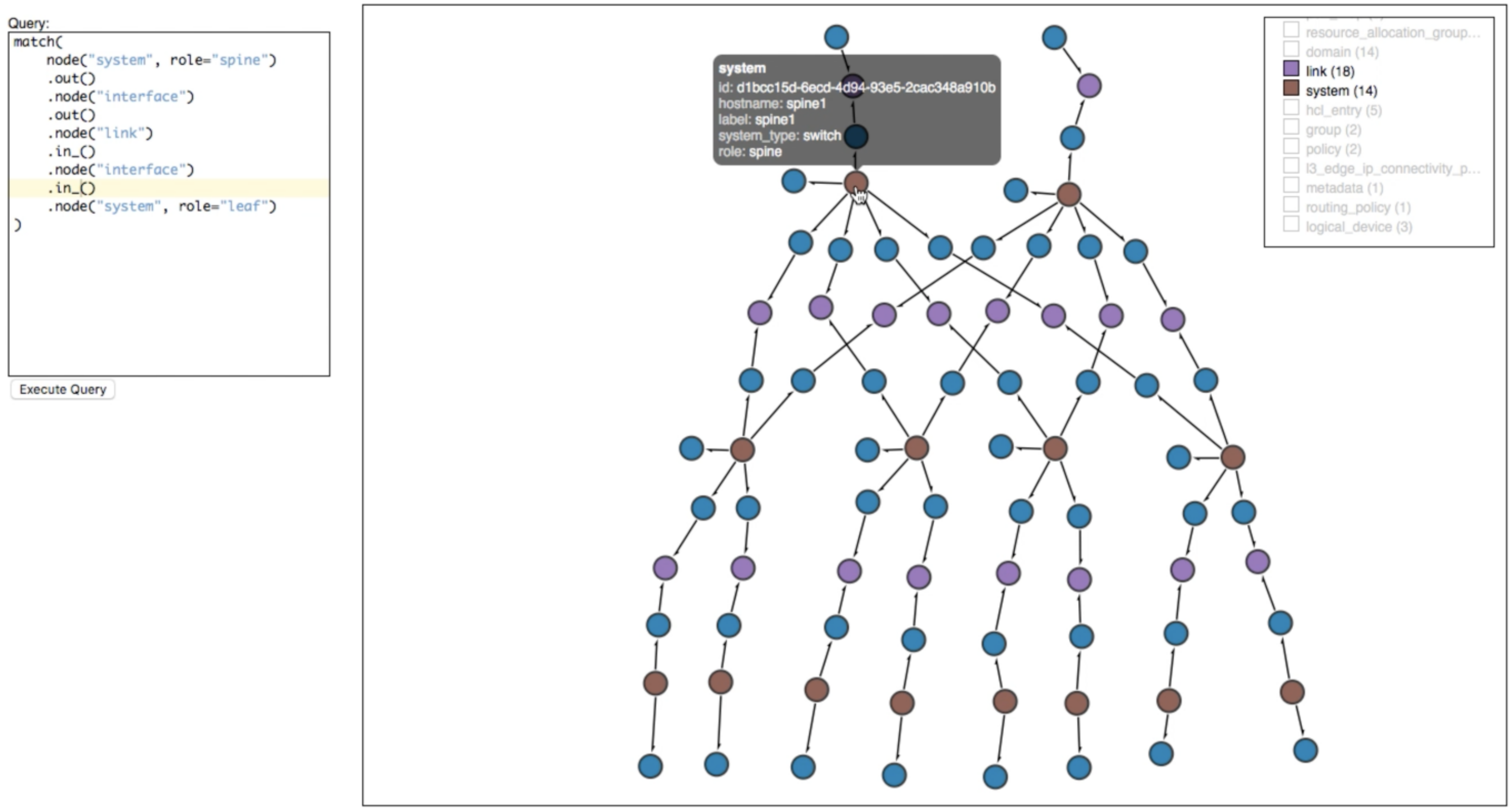
Decomposition in computer science, also known as **factoring**, is breaking a complex problem or system into parts that are easier to conceive, understand, program, and maintain.

Contents [hide]

DECOMPOSITION


Great, Big Problem
Break down into smaller, logical parts
Part 1 of problem
Part 2 of problem
Further break down into even smaller, logical parts
Further break down into even smaller, logical parts
Sub-problem 1
Sub-problem 2
Sub-problem 3
Sub-problem 4

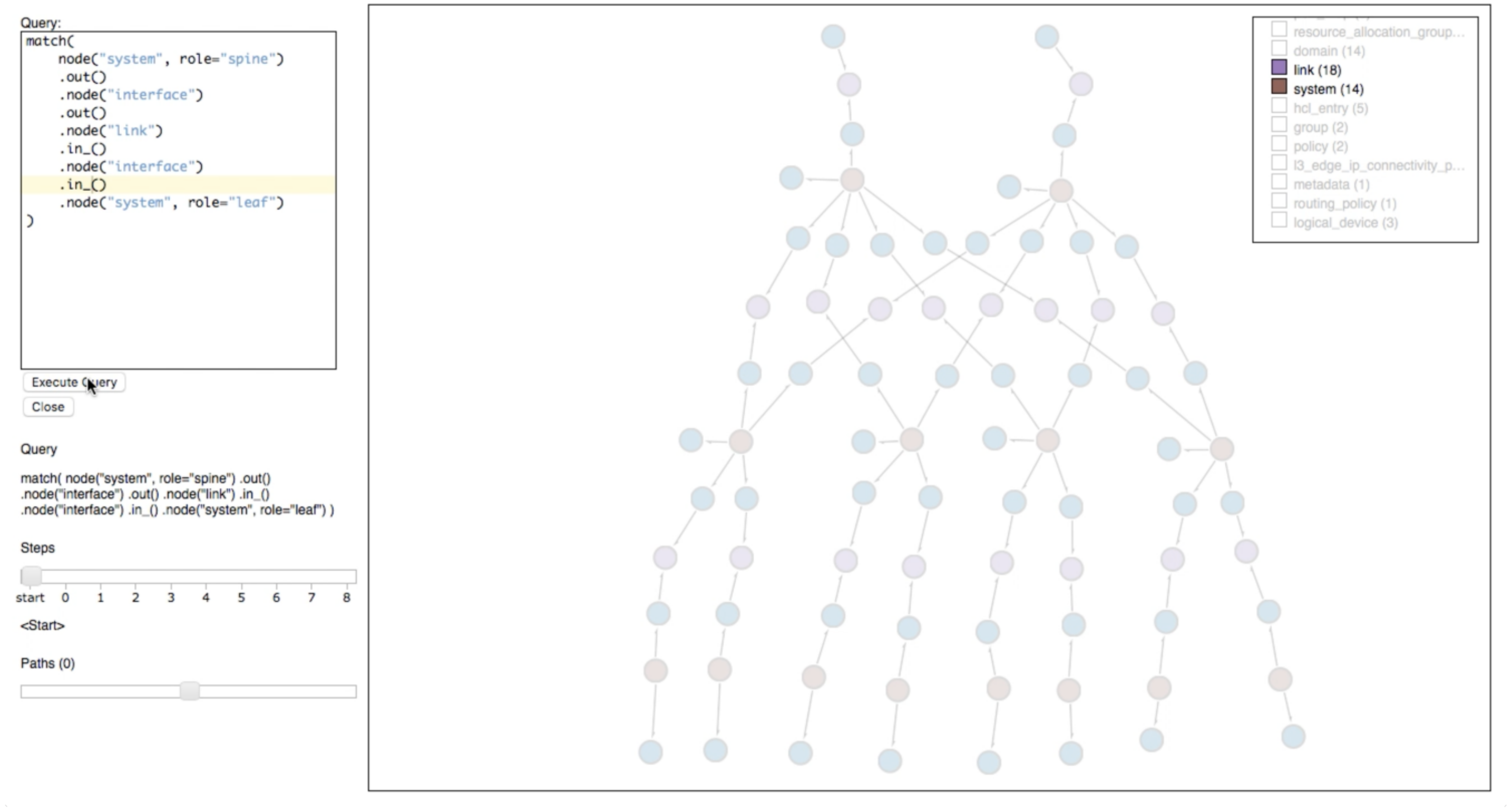
Decomposition: walking the graph



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

system
id: d1bcc15d-6ecd-4d94-93e5-2cac348a910b
hostname: spine1
label: spine1
system_type: switch
role: spine



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

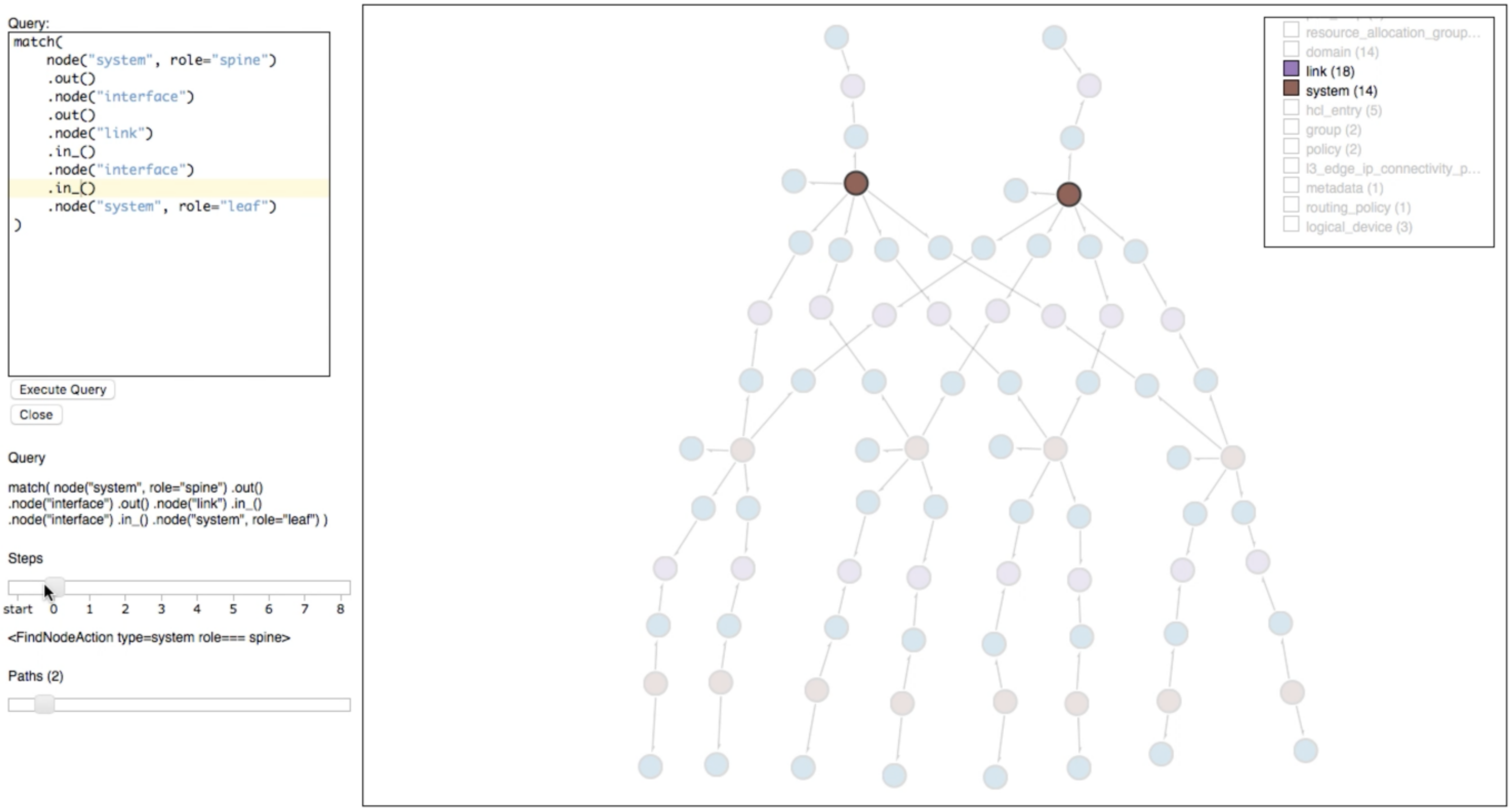
Query

match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

<Start>

Paths (0)



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Execute Query

Close

Query

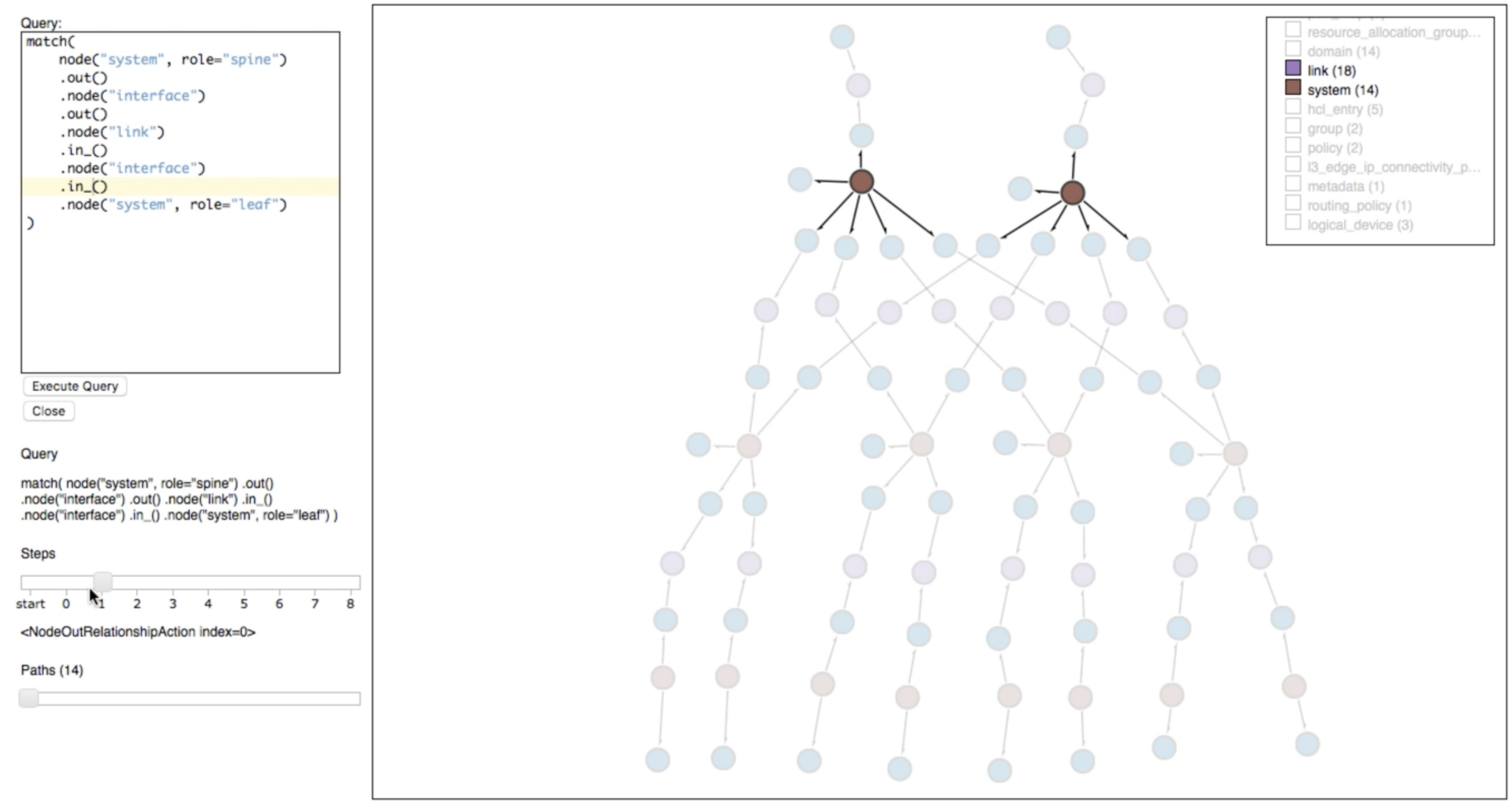
match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

<FindNodeAction type=system role=== spine>

Paths (2)

☐ resource_allocation_group...
☐ domain (14)
🟪 link (18)
🟫 system (14)
☐ hcl_entry (5)
☐ group (2)
☐ policy (2)
☐ l3_edge_ip_connectivity_p...
☐ metadata (1)
☐ routing_policy (1)
☐ logical_device (3)



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Execute Query

Close

Query

match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

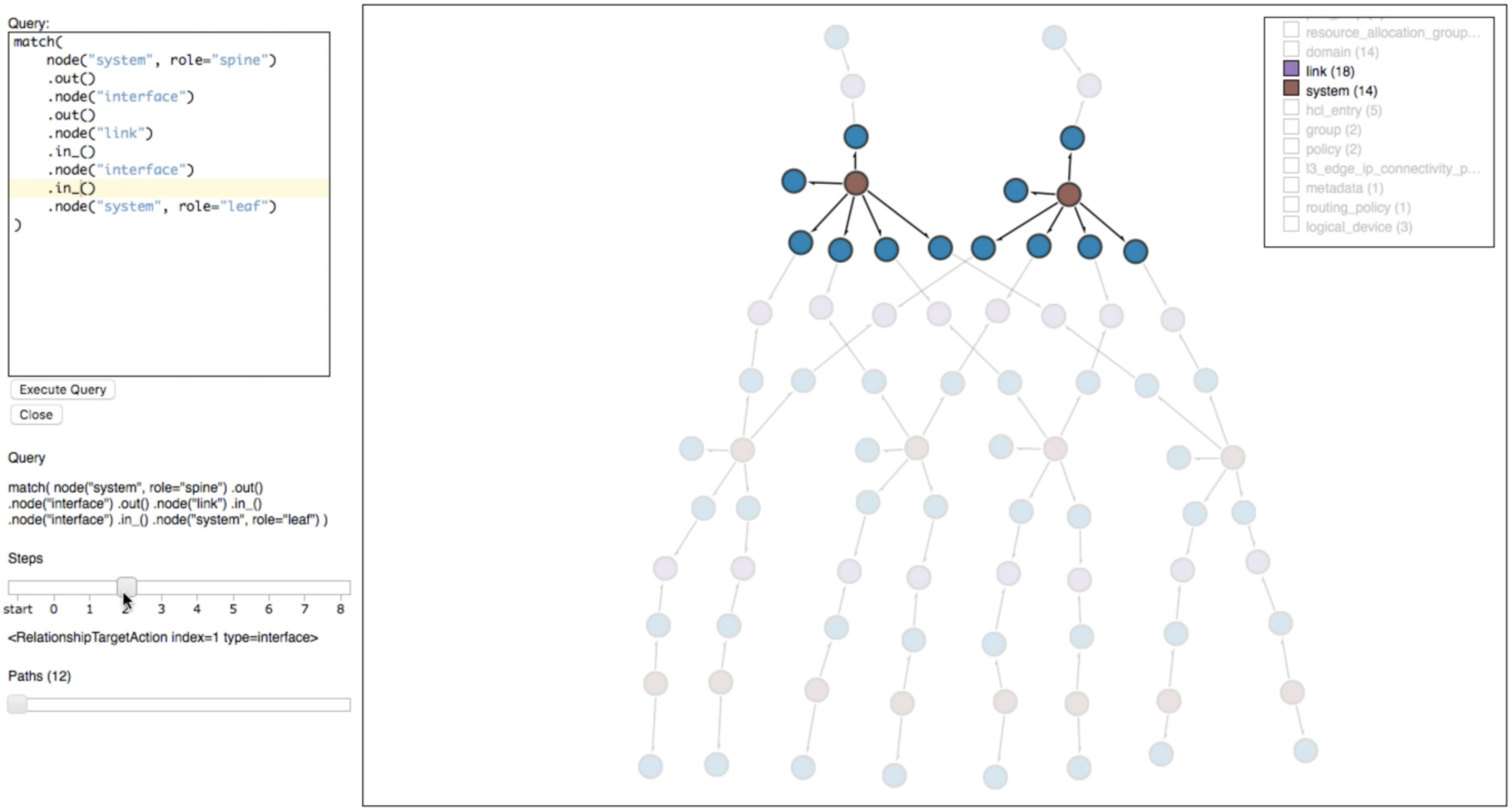
Steps

start 0 1 2 3 4 5 6 7 8

<NodeOutRelationshipAction index=0>

Paths (14)

- ☐ resource_allocation_group...
- ☐ domain (14)
- 🟪 link (18)
- 🟫 system (14)
- ☐ hcl_entry (5)
- ☐ group (2)
- ☐ policy (2)
- ☐ l3_edge_ip_connectivity_p...
- ☐ metadata (1)
- ☐ routing_policy (1)
- ☐ logical_device (3)


Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Execute Query

Close

Query

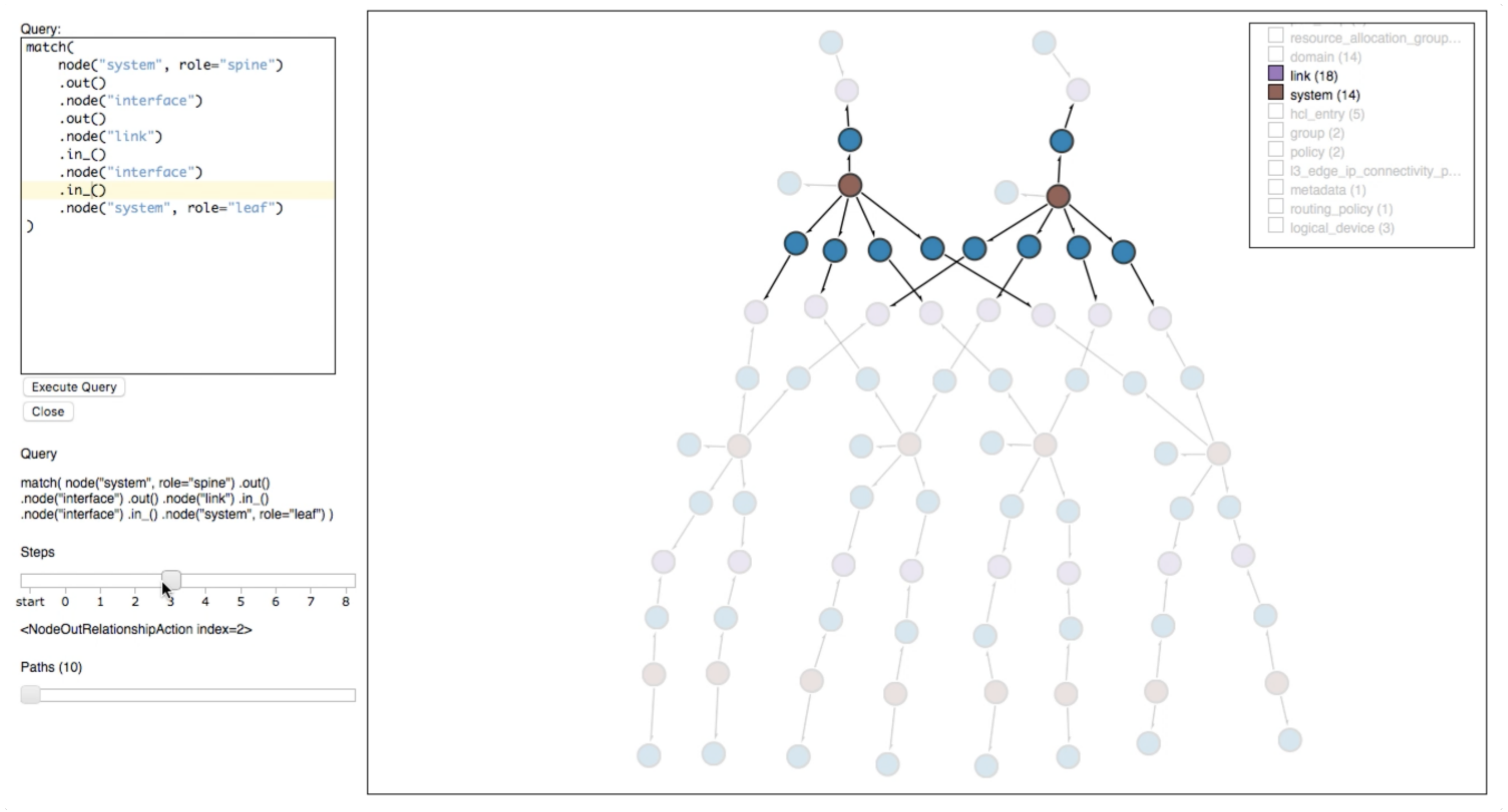
match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

start 0 1 2 3 4 5 6 7 8

<RelationshipTargetAction index=1 type=interface>

Paths (12)

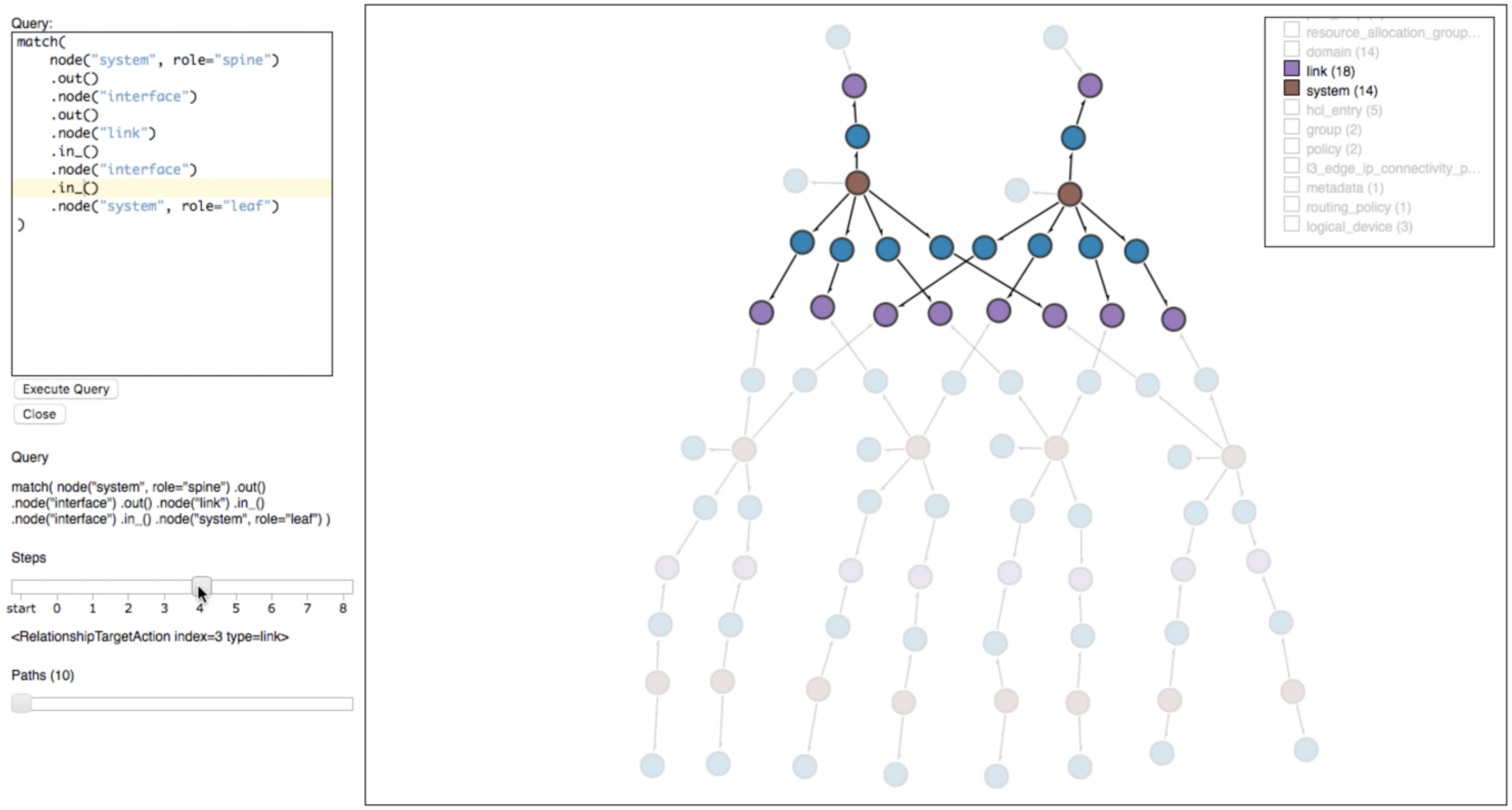


```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Query

match(node("system", role="spine") .out()
.node("interface") .out() .node("link") .in_()
.node("interface") .in_() .node("system", role="leaf"))

<NodeOutRelationshipAction index=2>



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Query

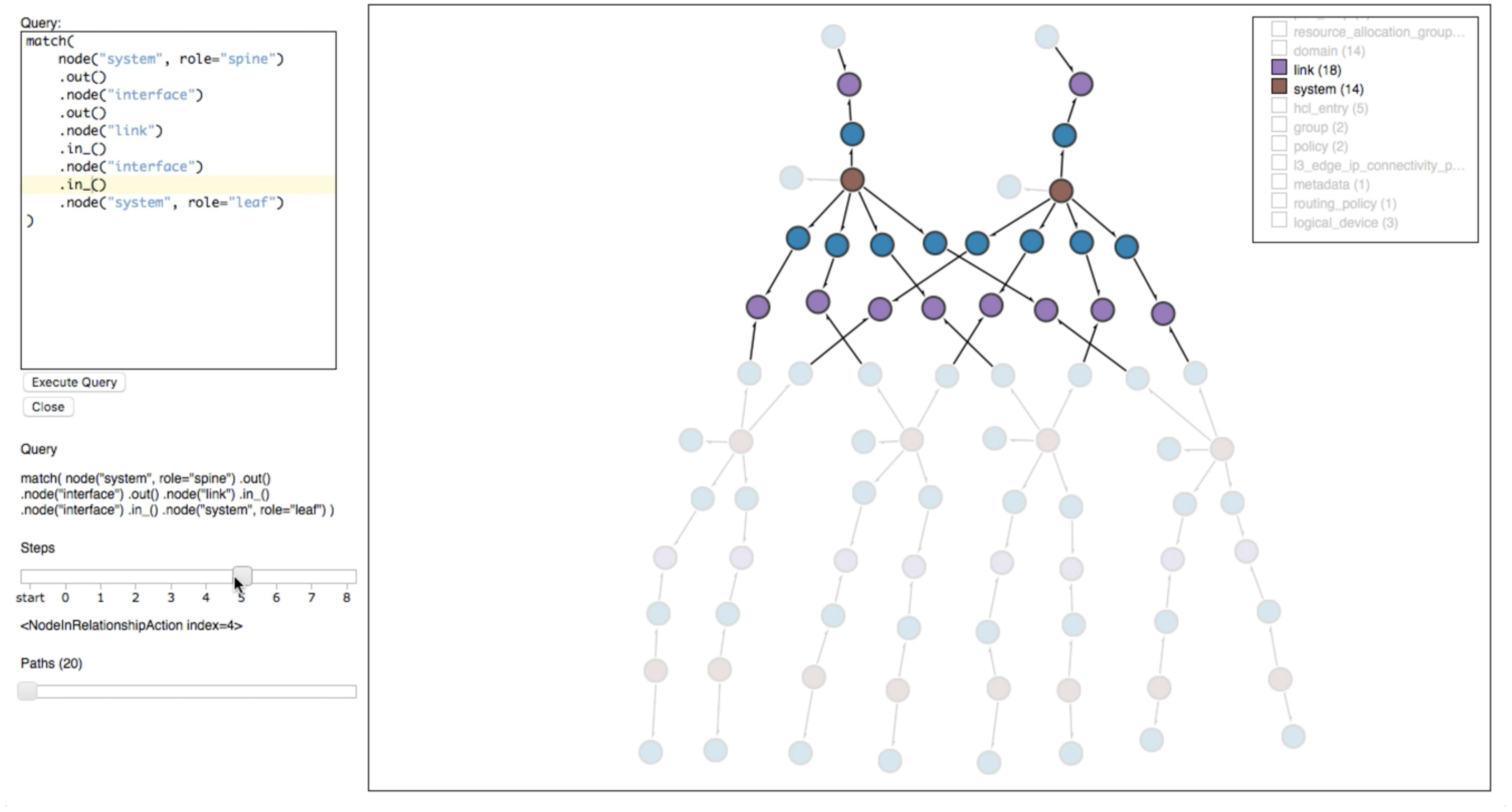
match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

<RelationshipTargetAction index=3 type=link>

Paths (10)

☐ resource_allocation_group...
☐ domain (14)
🟪 link (18)
🟫 system (14)
☐ hcl_entry (5)
☐ group (2)
☐ policy (2)
☐ l3_edge_ip_connectivity_p...
☐ metadata (1)
☐ routing_policy (1)
☐ logical_device (3)



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

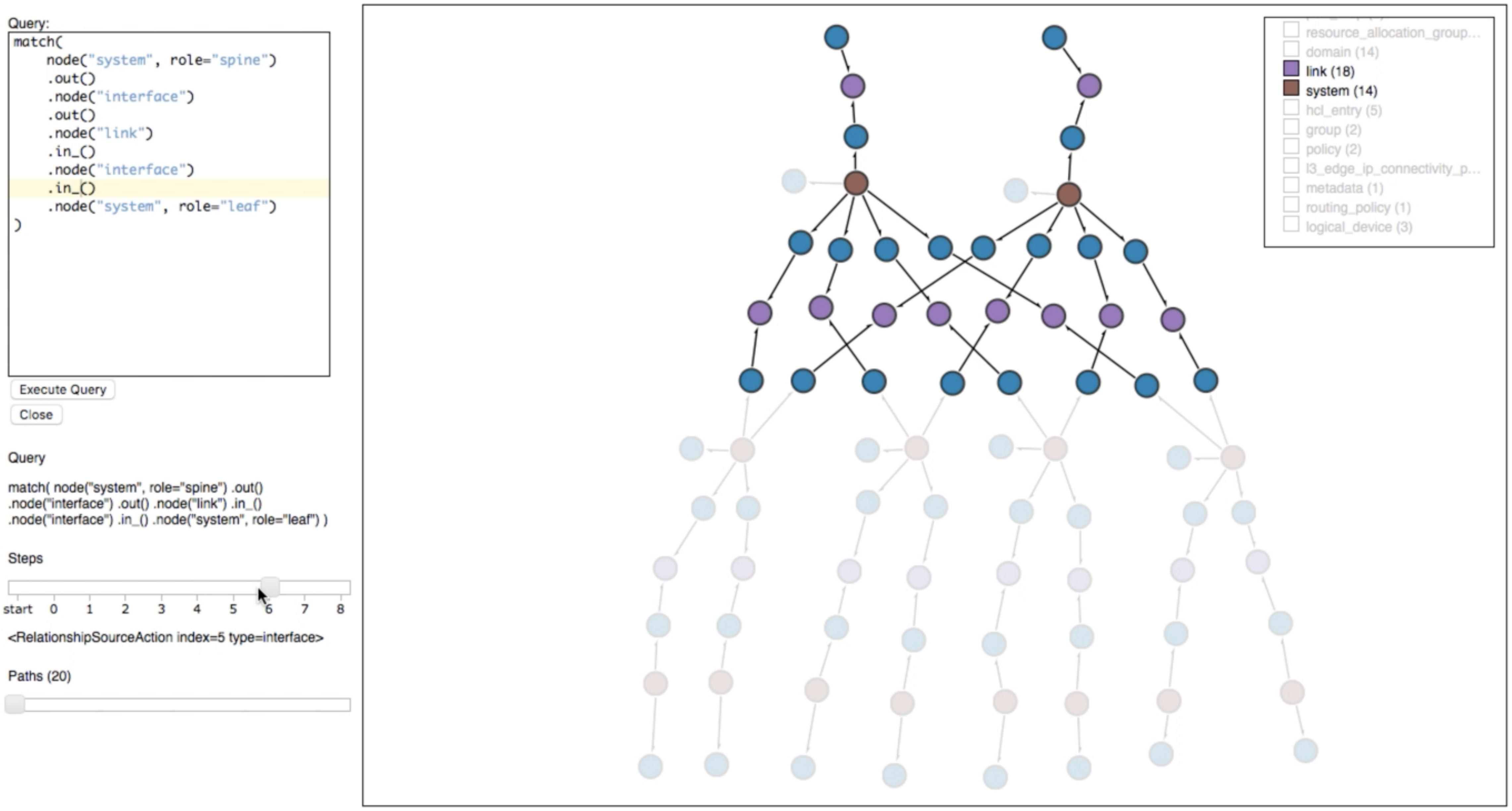
Query

match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

<NodeInRelationshipAction index=4>

Paths (20)



```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Query

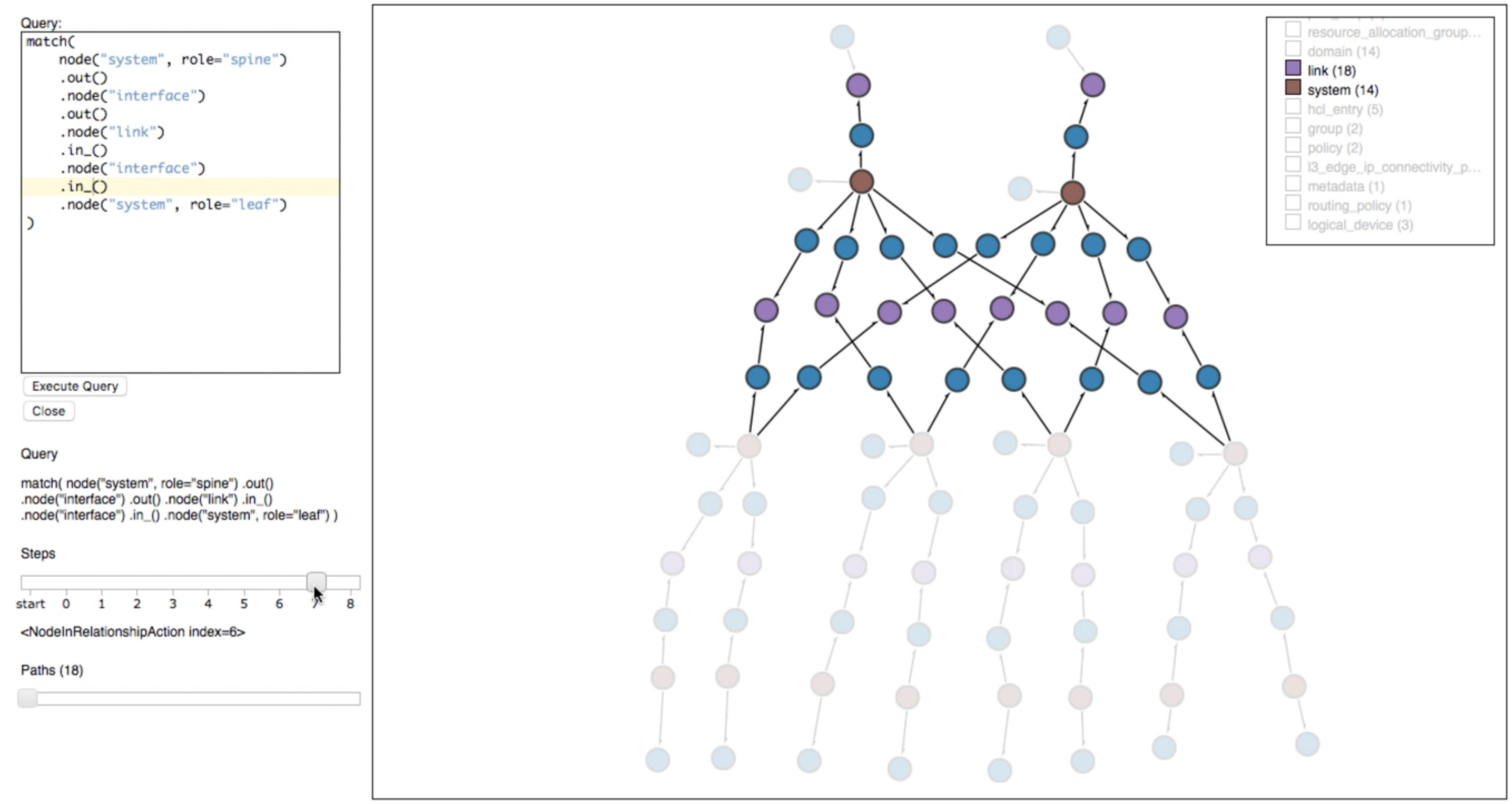
match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

start 0 1 2 3 4 5 6 7 8

<RelationshipSourceAction index=5 type=interface>

Paths (20)



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Query

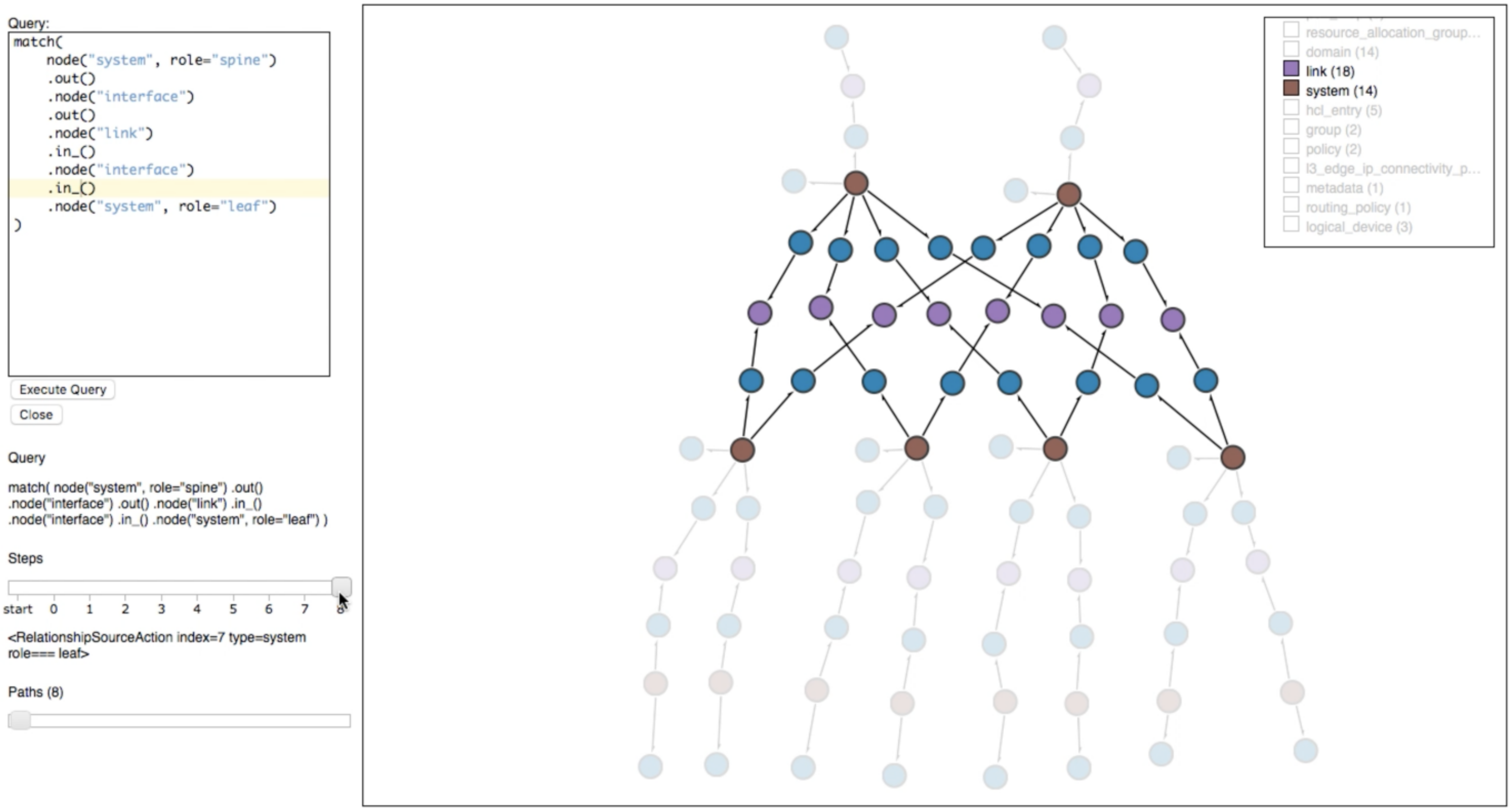
match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

start 0 1 2 3 4 5 6 7 8

<NodeInRelationshipAction index=6>

Paths (18)


Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

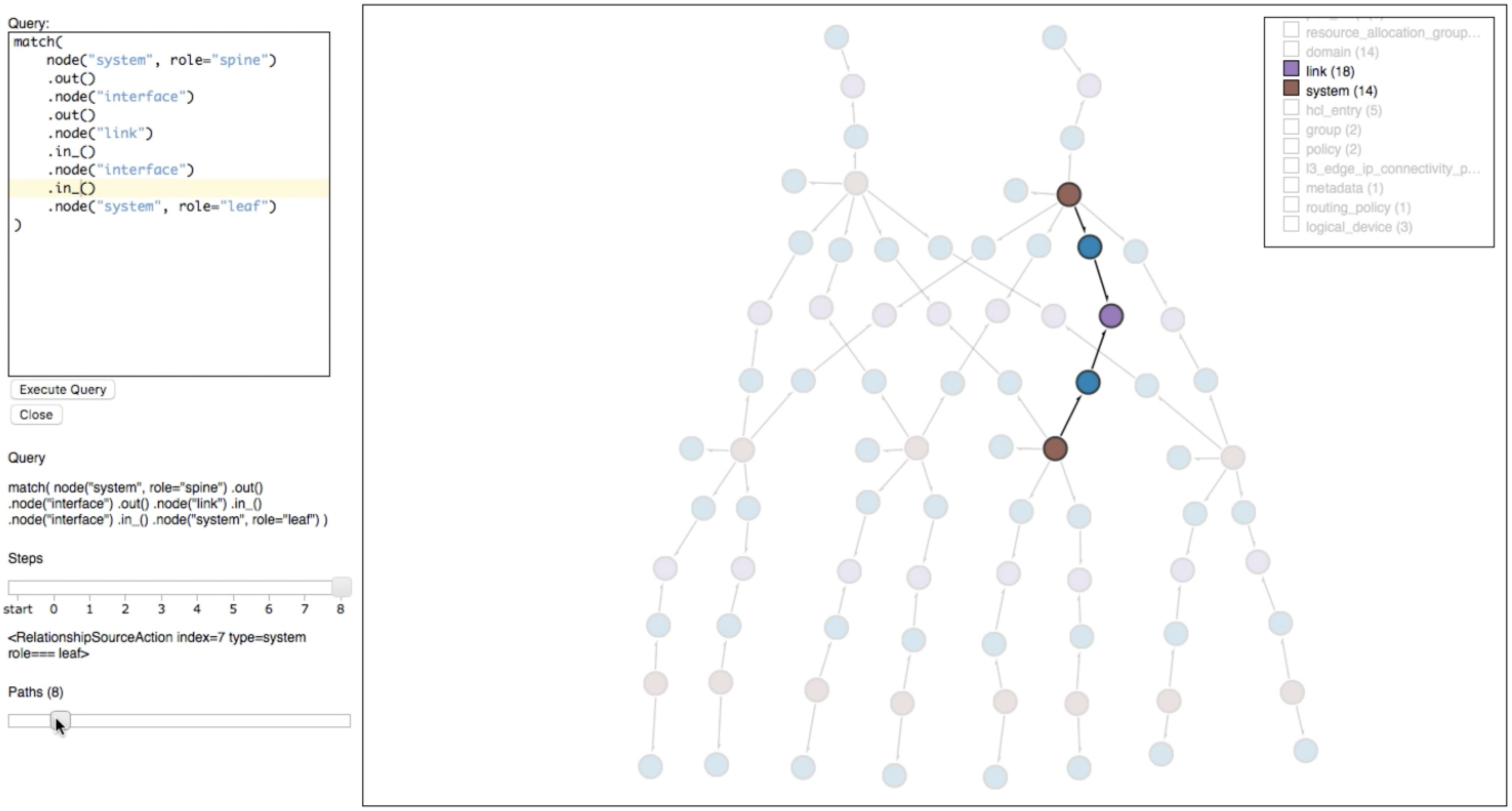
Query

match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

<RelationshipSourceAction index=7 type=system role=== leaf>

Paths (8)

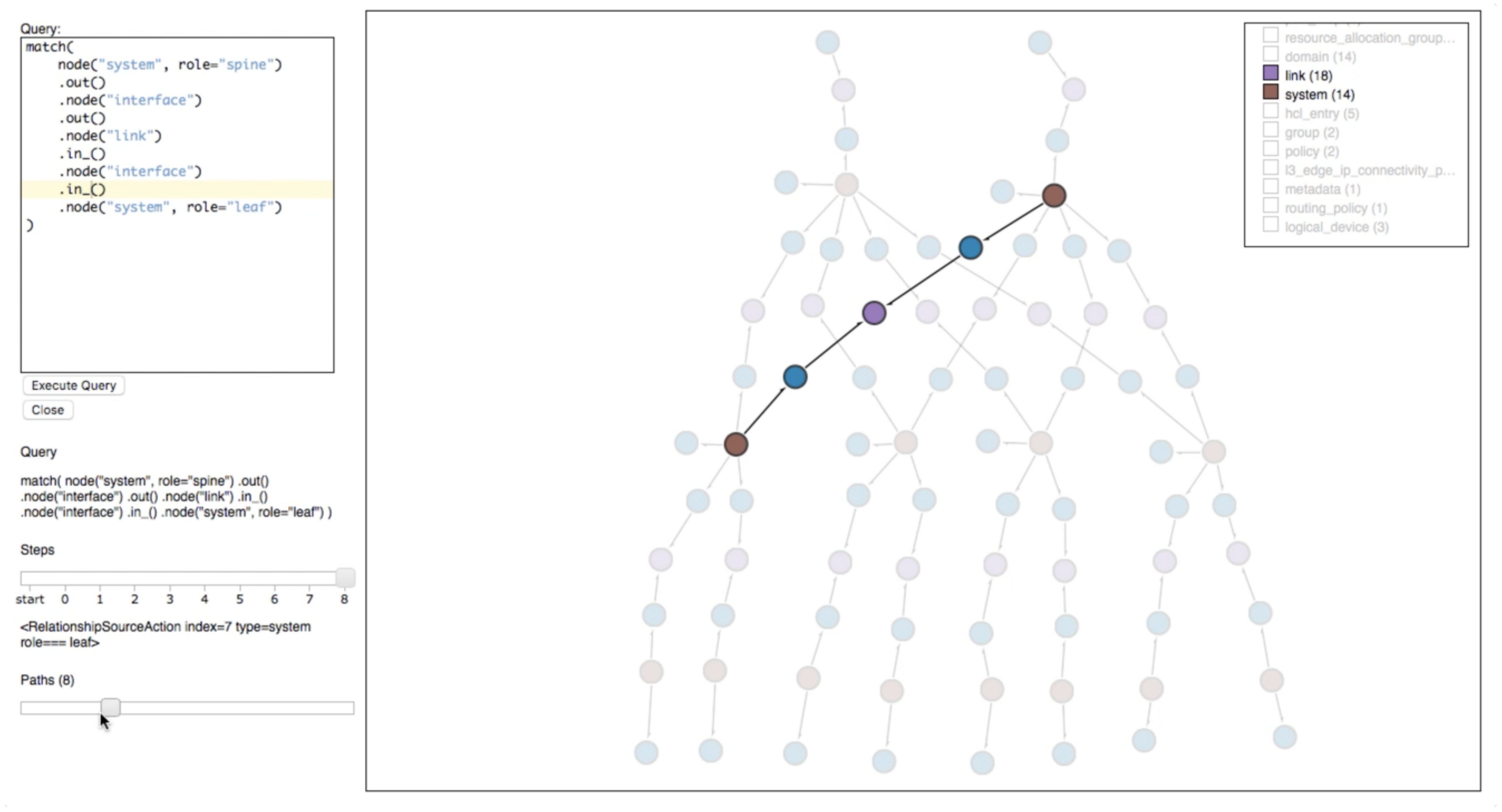


```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Query

match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

<RelationshipSourceAction index=7 type=system role=== leaf>


Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Execute Query

Close

Query

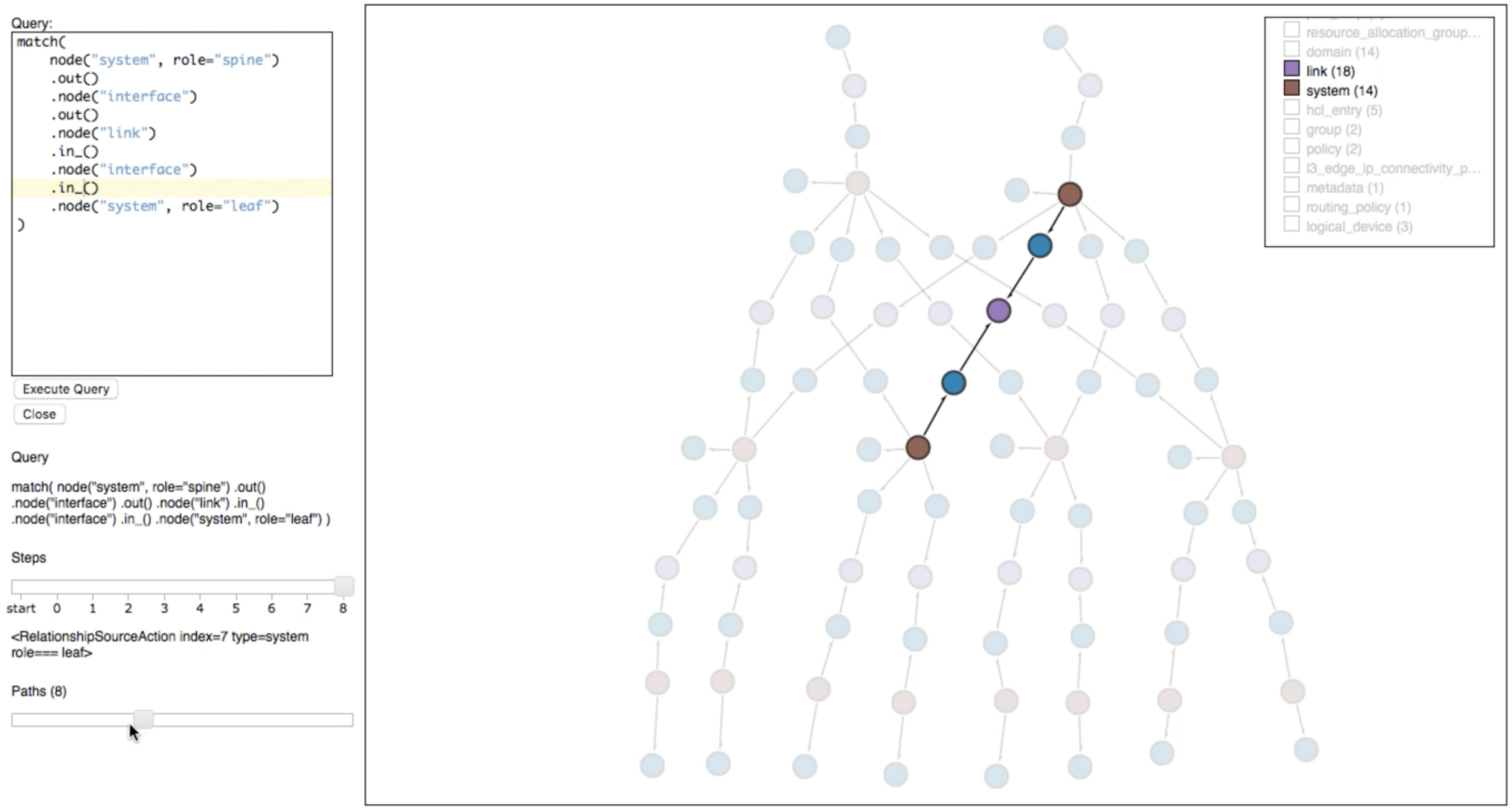
match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

start 0 1 2 3 4 5 6 7 8

<RelationshipSourceAction index=7 type=system role=== leaf>

Paths (8)



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

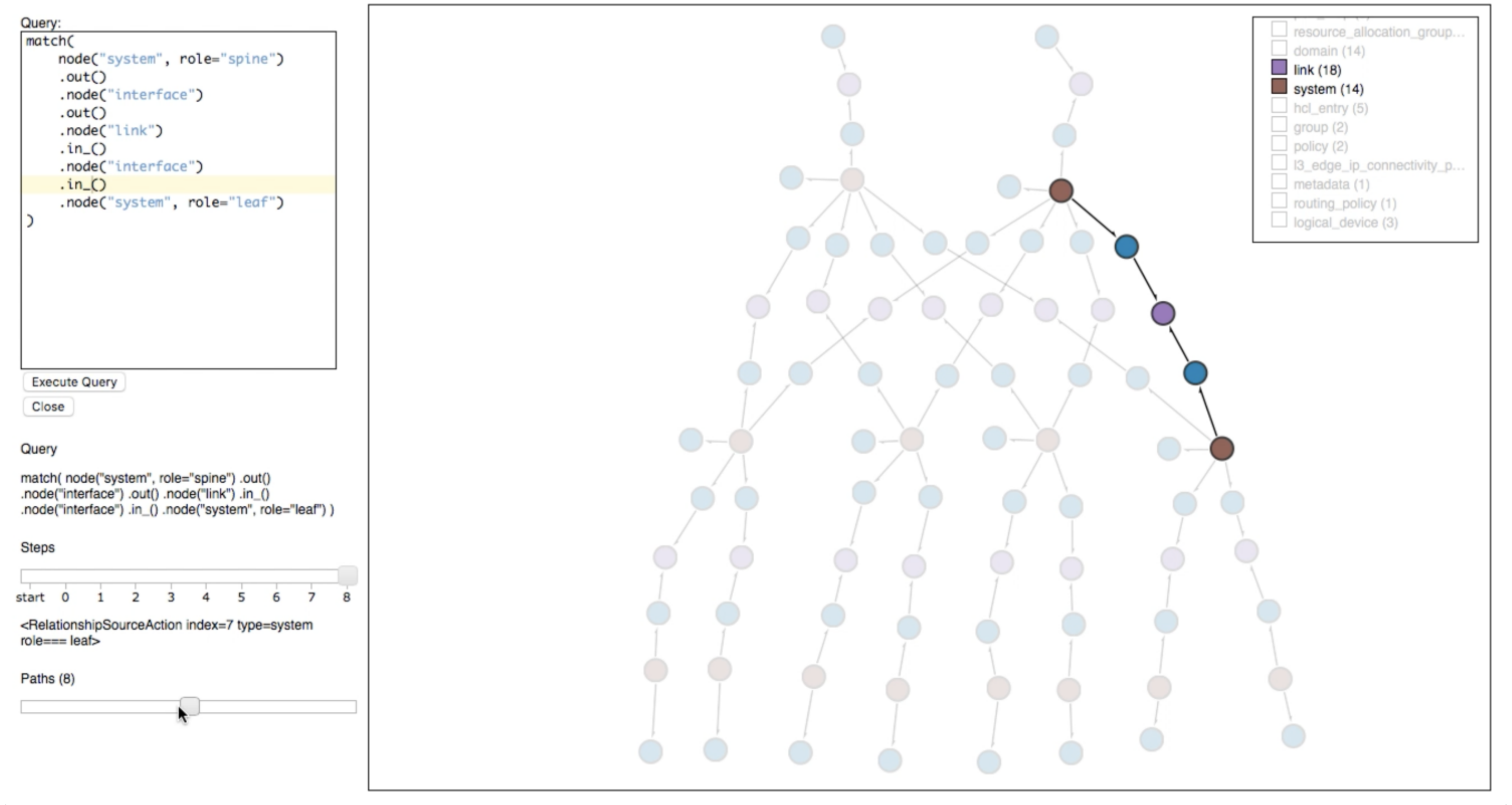
Query

match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

<RelationshipSourceAction index=7 type=system role=== leaf>

Paths (8)



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

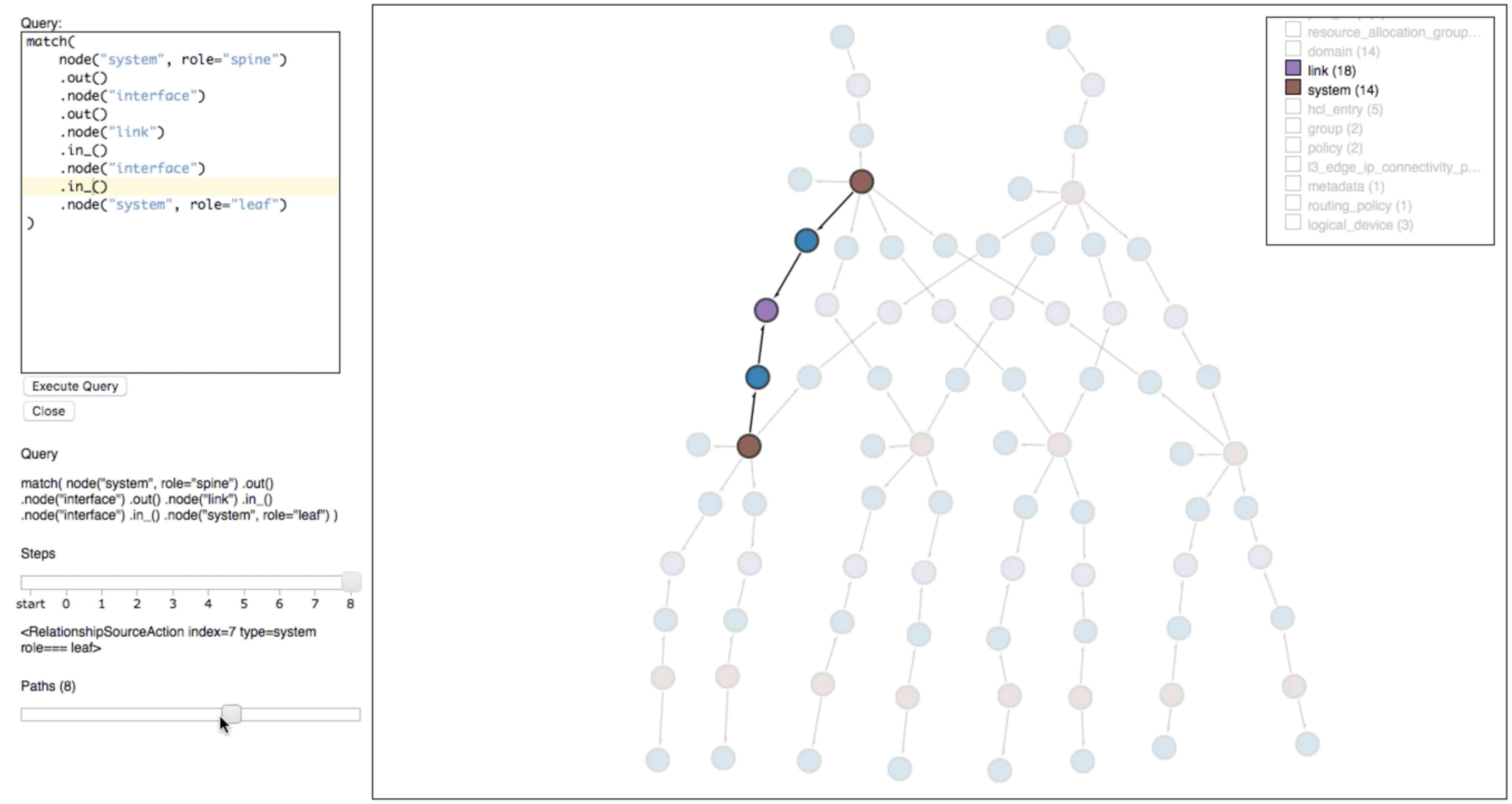
Query

match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

<RelationshipSourceAction index=7 type=system role=== leaf>

Paths (8)



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Execute Query

Close

Query

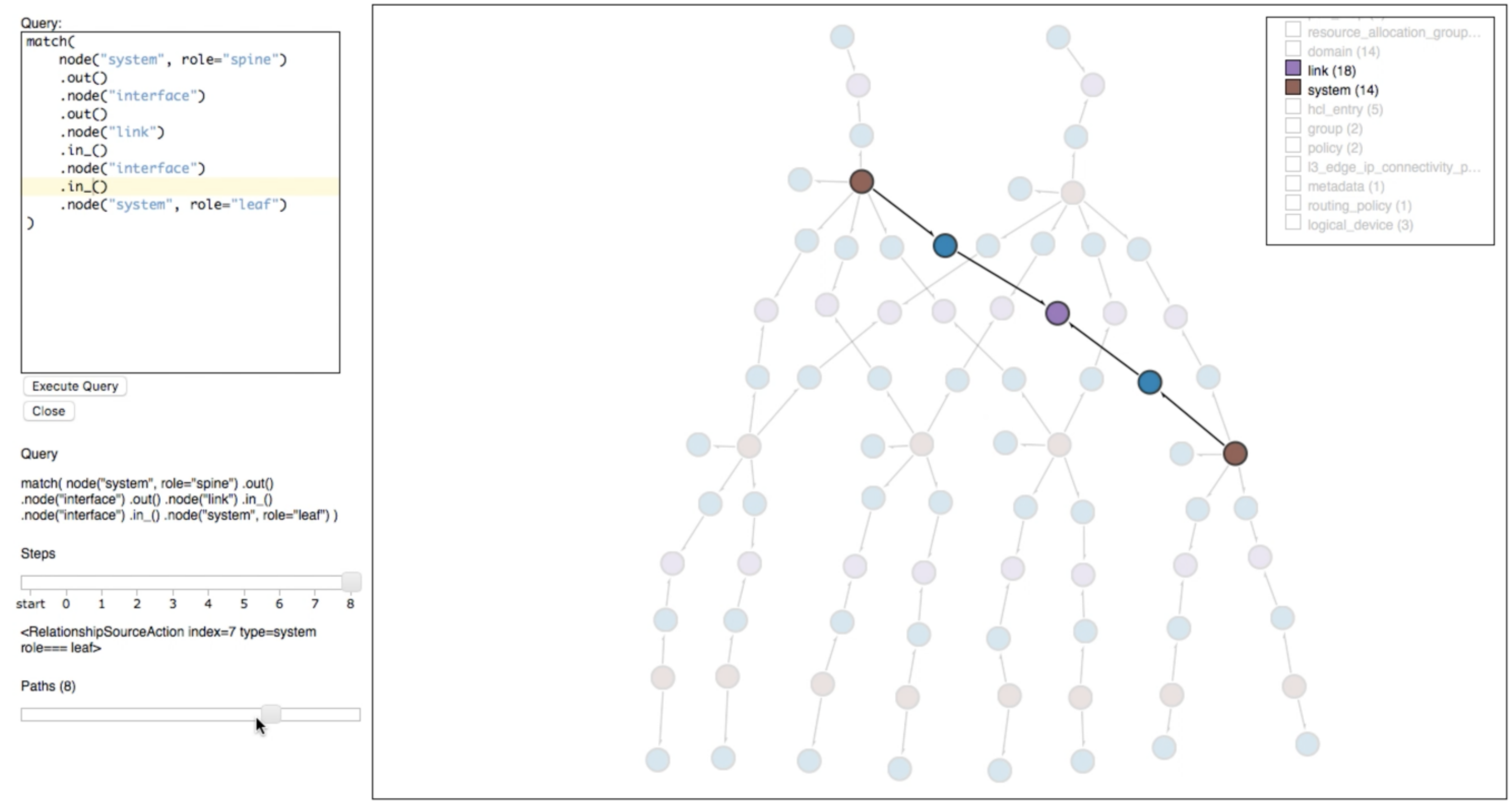
match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

start 0 1 2 3 4 5 6 7 8

<RelationshipSourceAction index=7 type=system role=== leaf>

Paths (8)



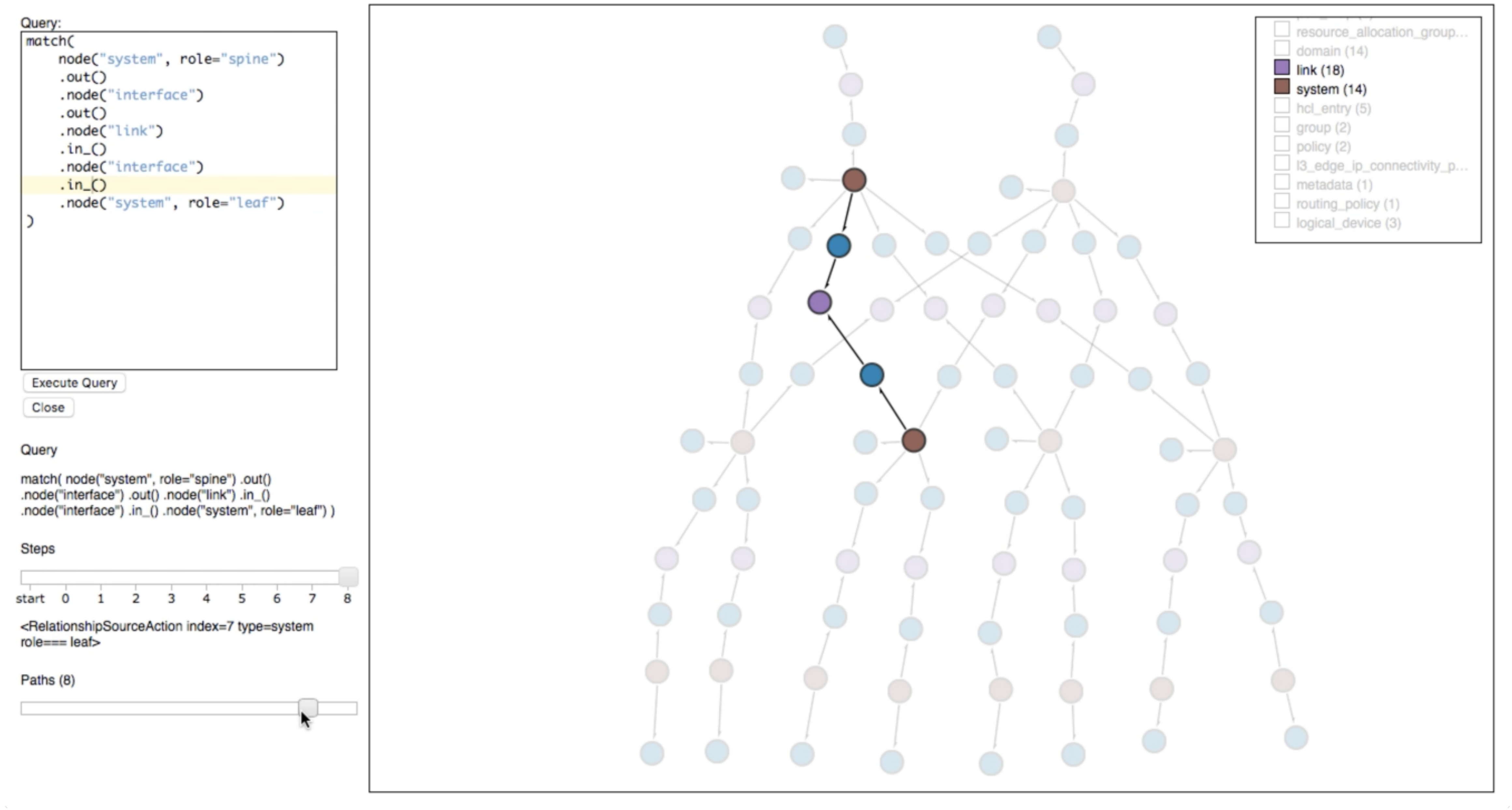
Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Query

match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

```
<RelationshipSourceAction index=7 type=system
role=== leaf>
```



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

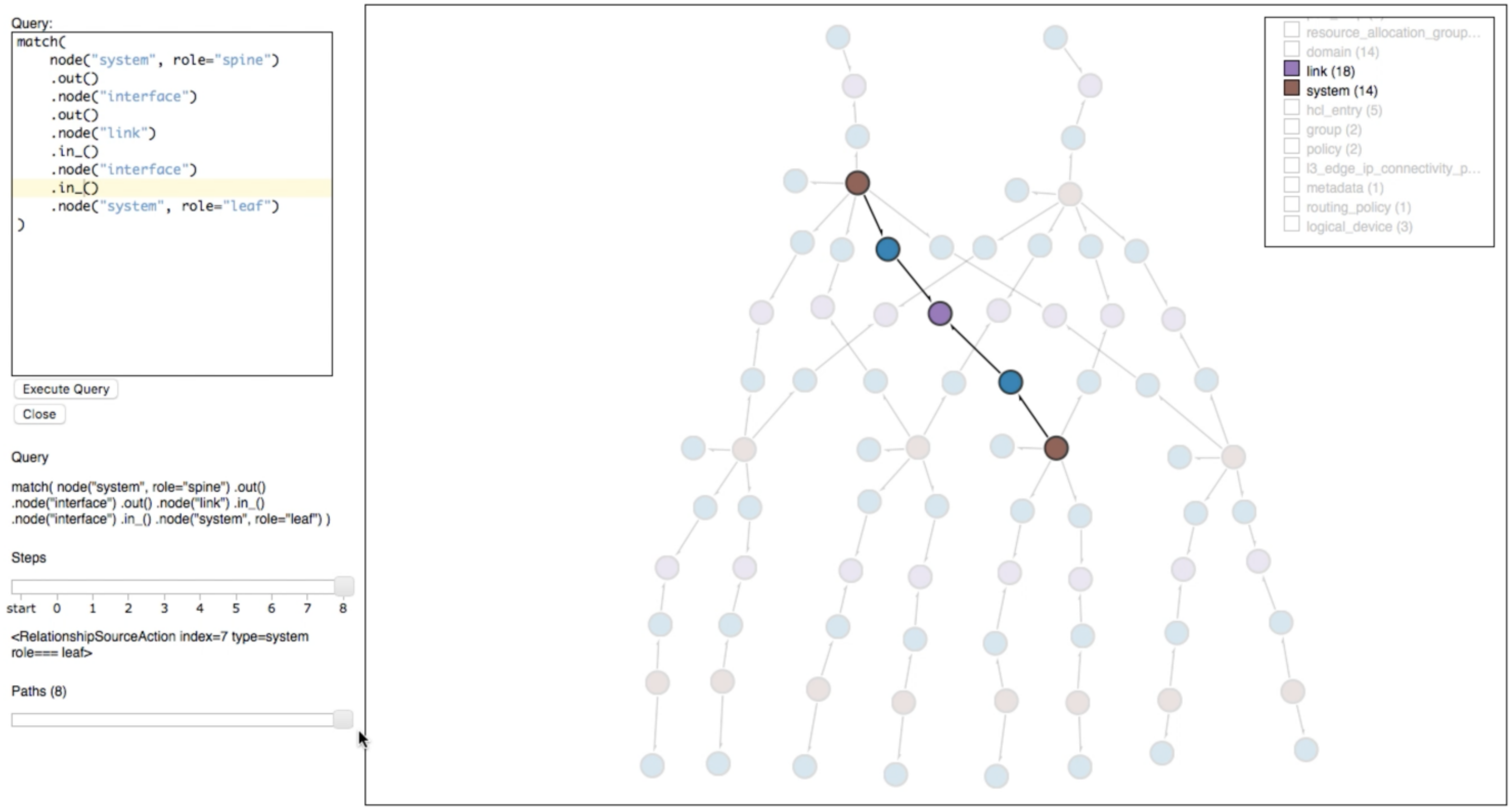
Query

match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

<RelationshipSourceAction index=7 type=system role=== leaf>

Paths (8)



Query:

```
match(
    node("system", role="spine")
    .out()
    .node("interface")
    .out()
    .node("link")
    .in_()
    .node("interface")
    .in_()
    .node("system", role="leaf")
)
```

Execute Query

Close

Query

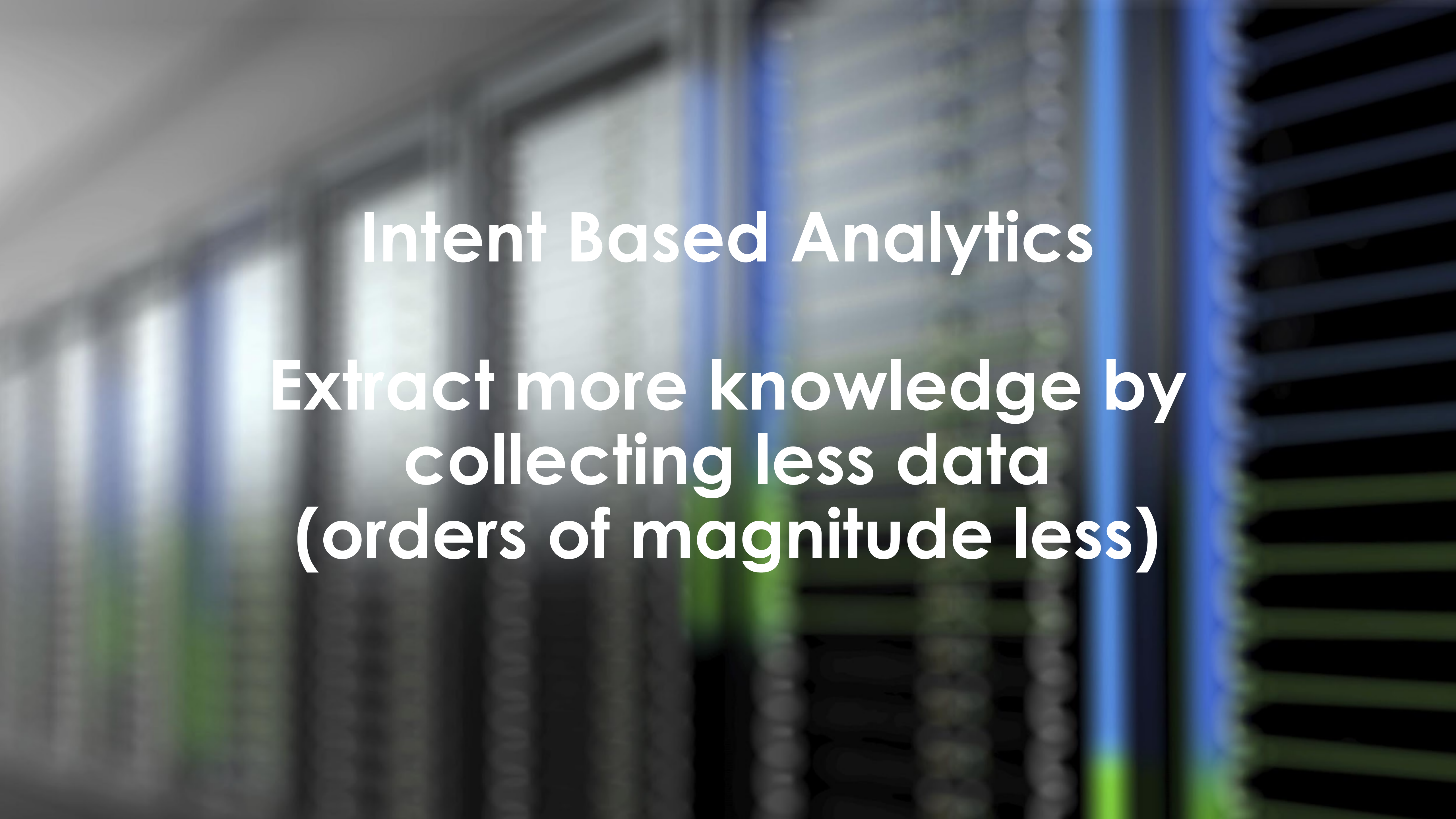
match(node("system", role="spine") .out() .node("interface") .out() .node("link") .in_() .node("interface") .in_() .node("system", role="leaf"))

Steps

start 0 1 2 3 4 5 6 7 8

<RelationshipSourceAction index=7 type=system role=== leaf>

Paths (8)



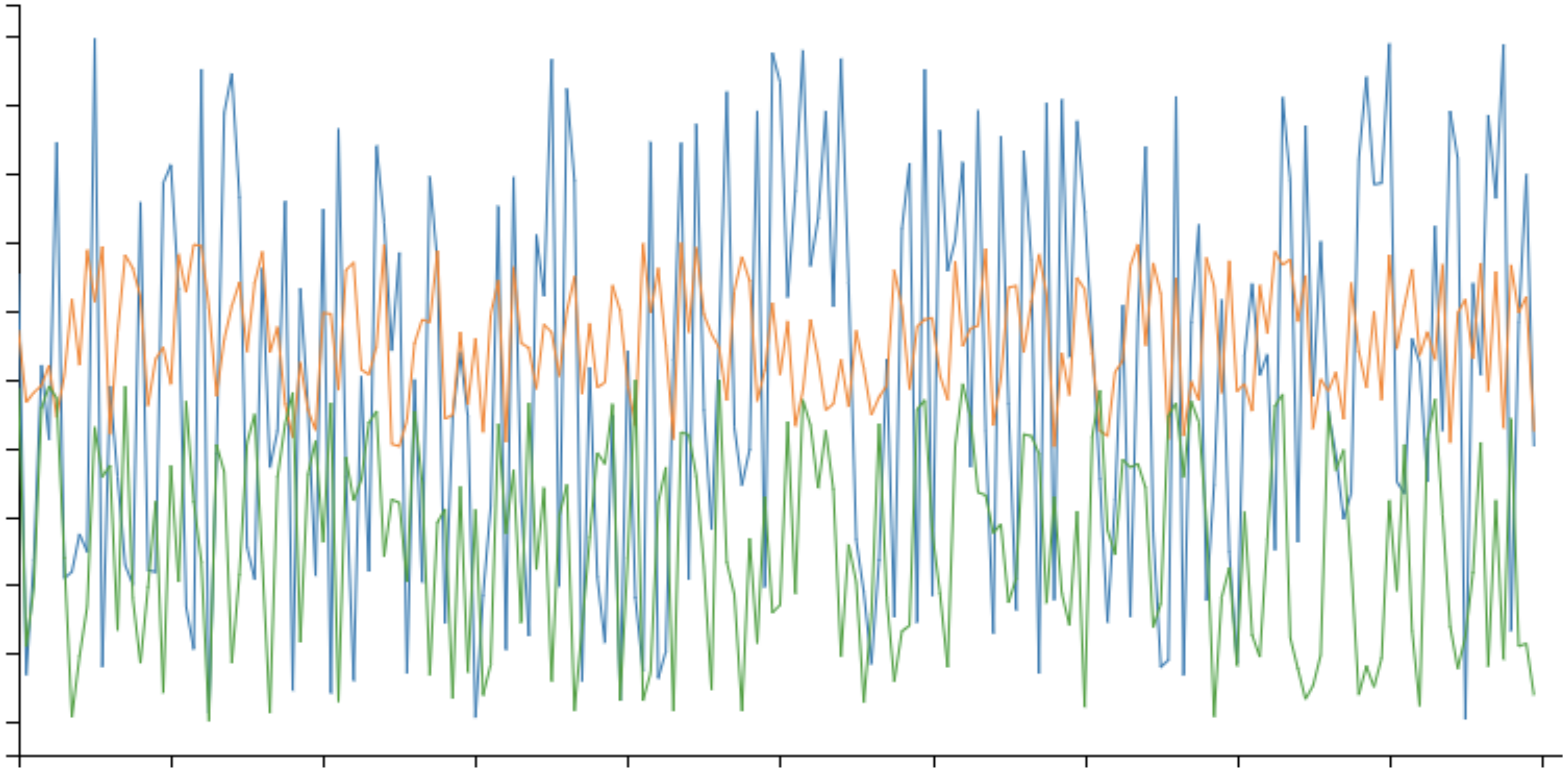
Intent Based Analytics

Extract more knowledge by collecting less data (orders of magnitude less)

Was I looking for something?

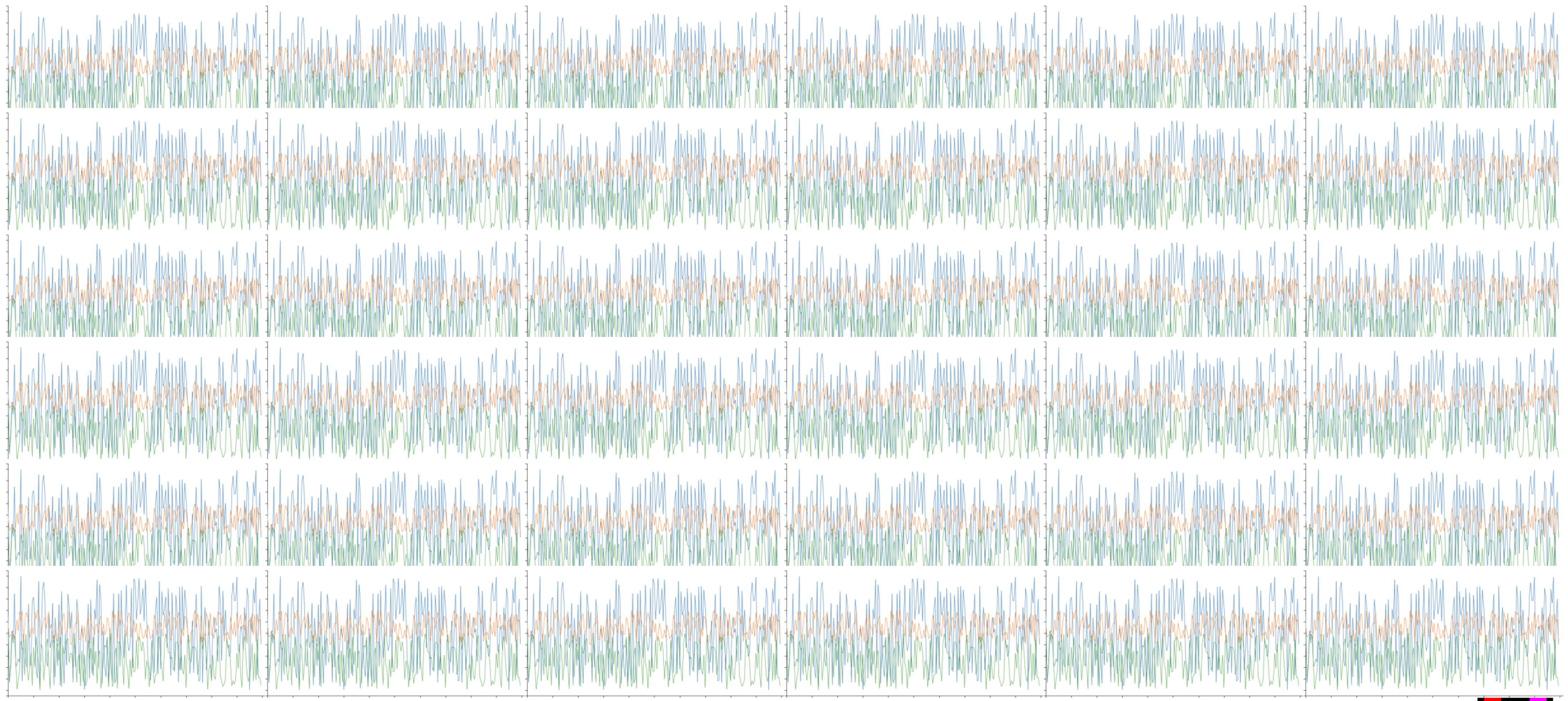
apstra

Gathering high def telemetry

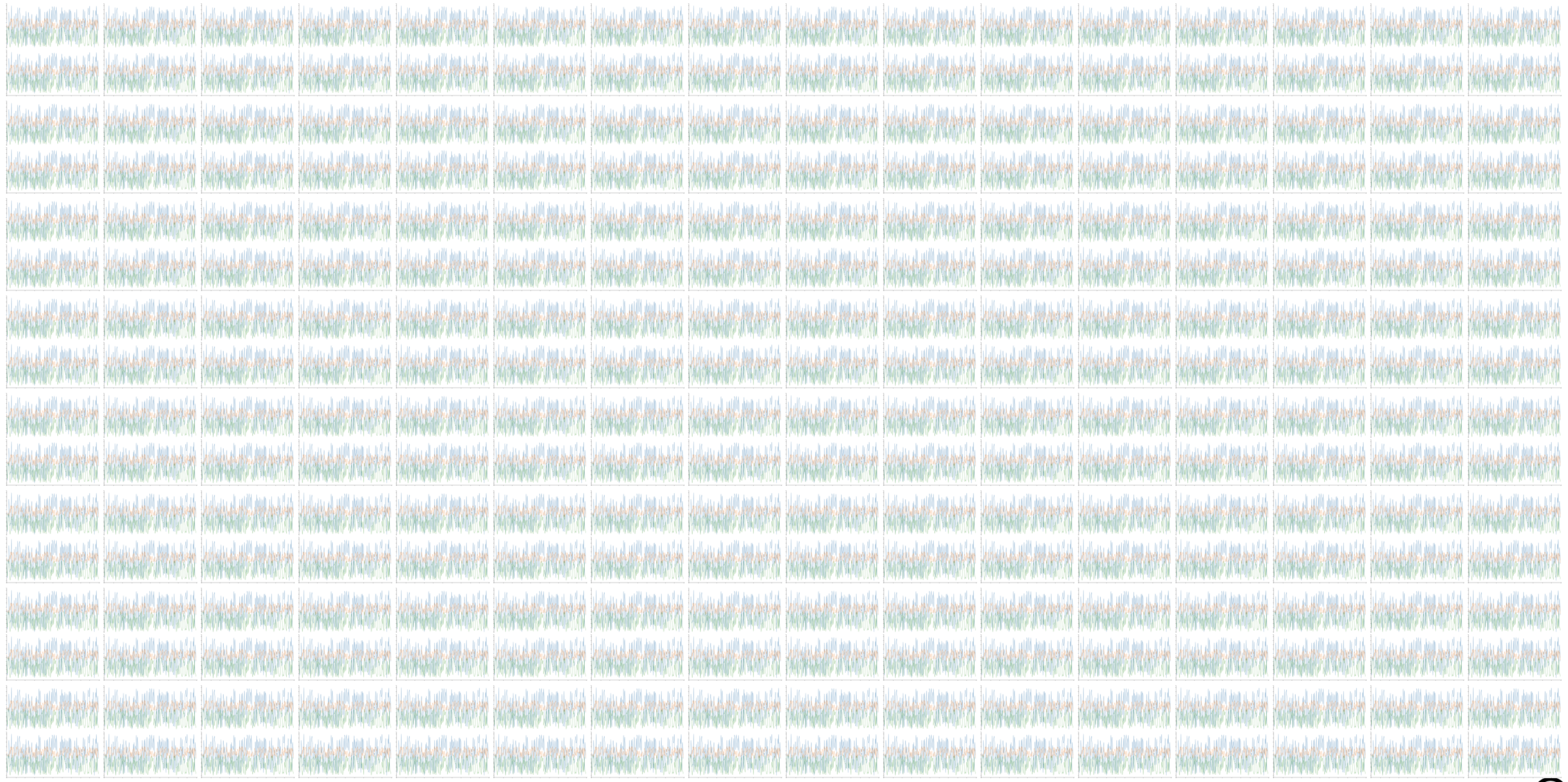


apstra

For all my leaf1 interfaces

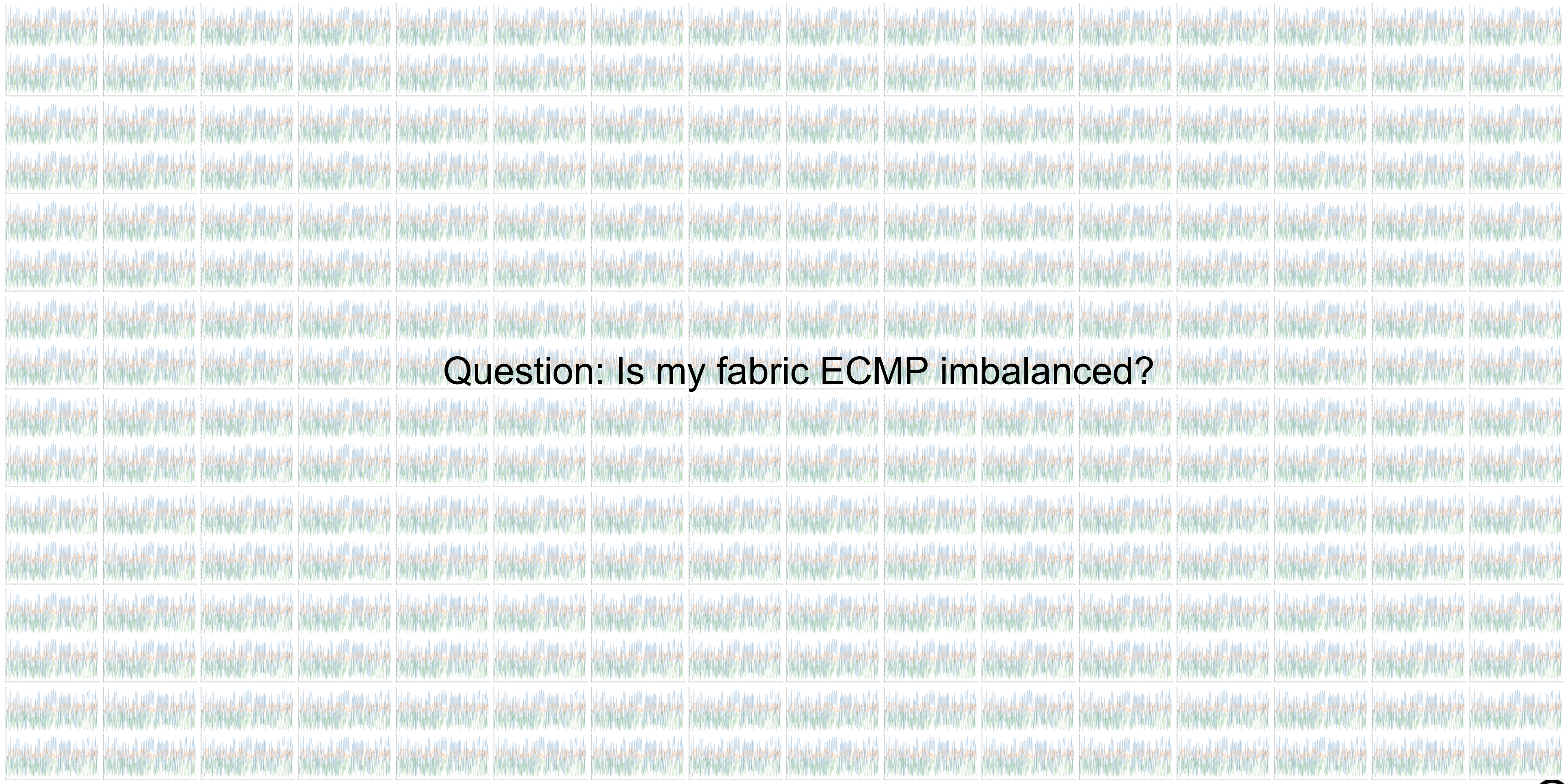


For all my leafs



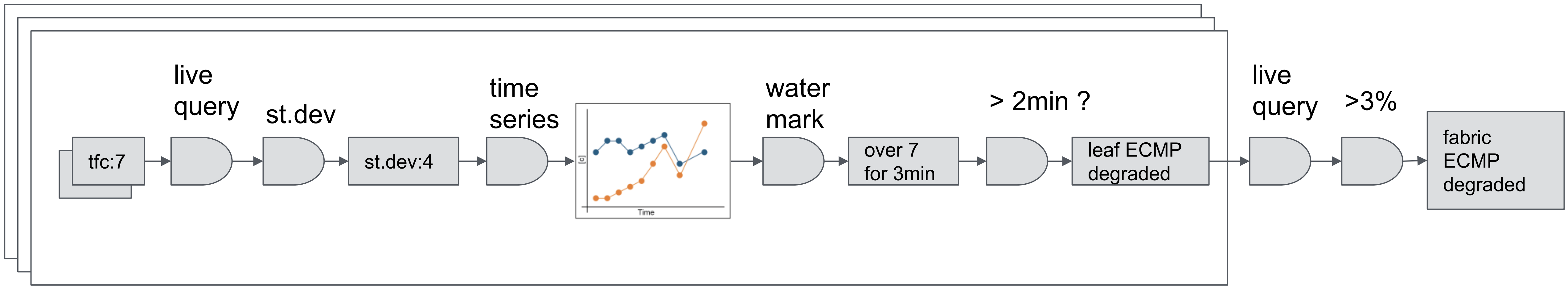
apstra

So that I have insight


Question: Is my fabric ECMP imbalanced?

apstra

IBA : ECMP fabric health



apstra

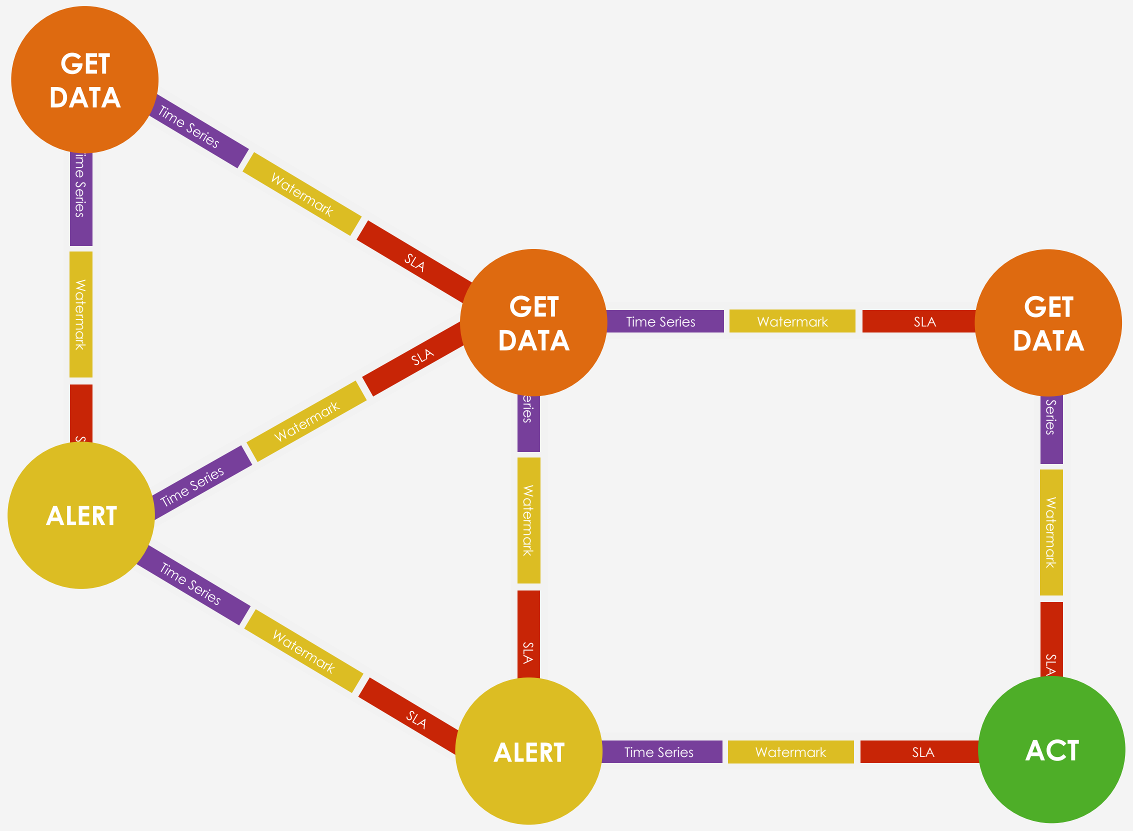
IBA : ECMP fabric health

1 Probes are Directed Graphs

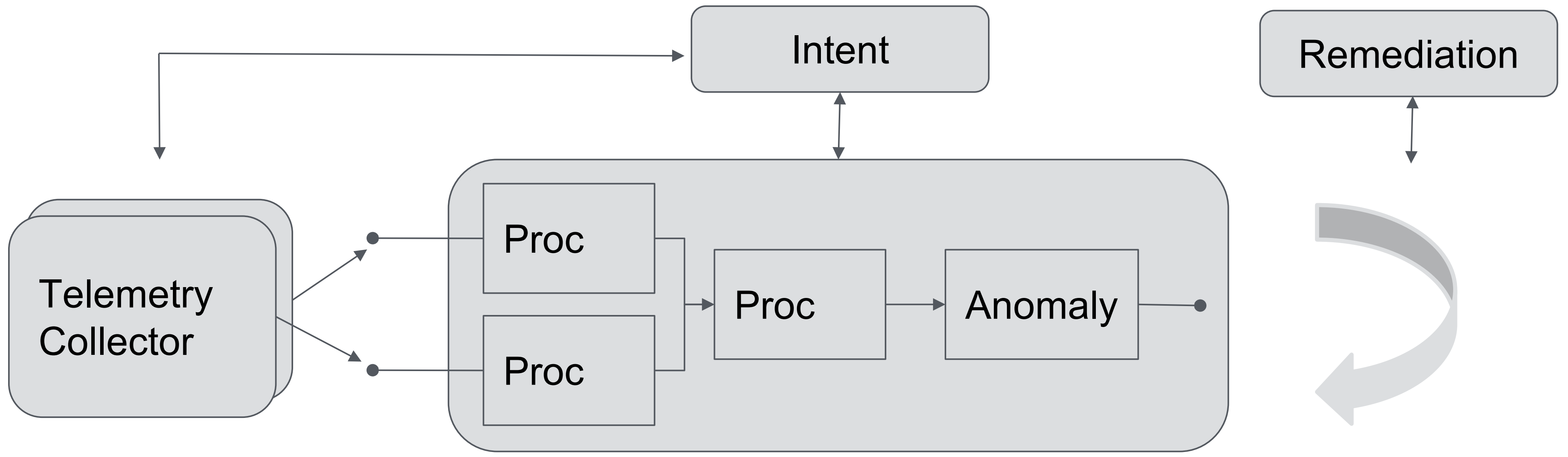
2 Will calculate until a result is found

3 Each IBA Probe is a collection of processors

4 This is one probe



IBA – context aware analytics



Declaratively specified, definition is de-coupled from instantiation

Once specified, is in constant sync with intent

Extracts knowledge out of the raw telemetry – context drives the content

New telemetry is “wired-in”

apstra

Conclusion

- Basic automation, while hot topic - is the first and easiest step in the IBN journey
- Single source of truth is mandatory for an IBN system to be able to reason about any change
- Day 2 operations @scale:
  - context aware continues validation
  - dealing with changes
  - configuration drift
  - remediation

is the most complicated area of technologies to deal with!

apstra

Questions

apstra

Thank You!

www.apstra.com

@ApstraInc

https://www.linkedin.com/company/apstra

https://www.facebook.com/apstrainc/

apstra®