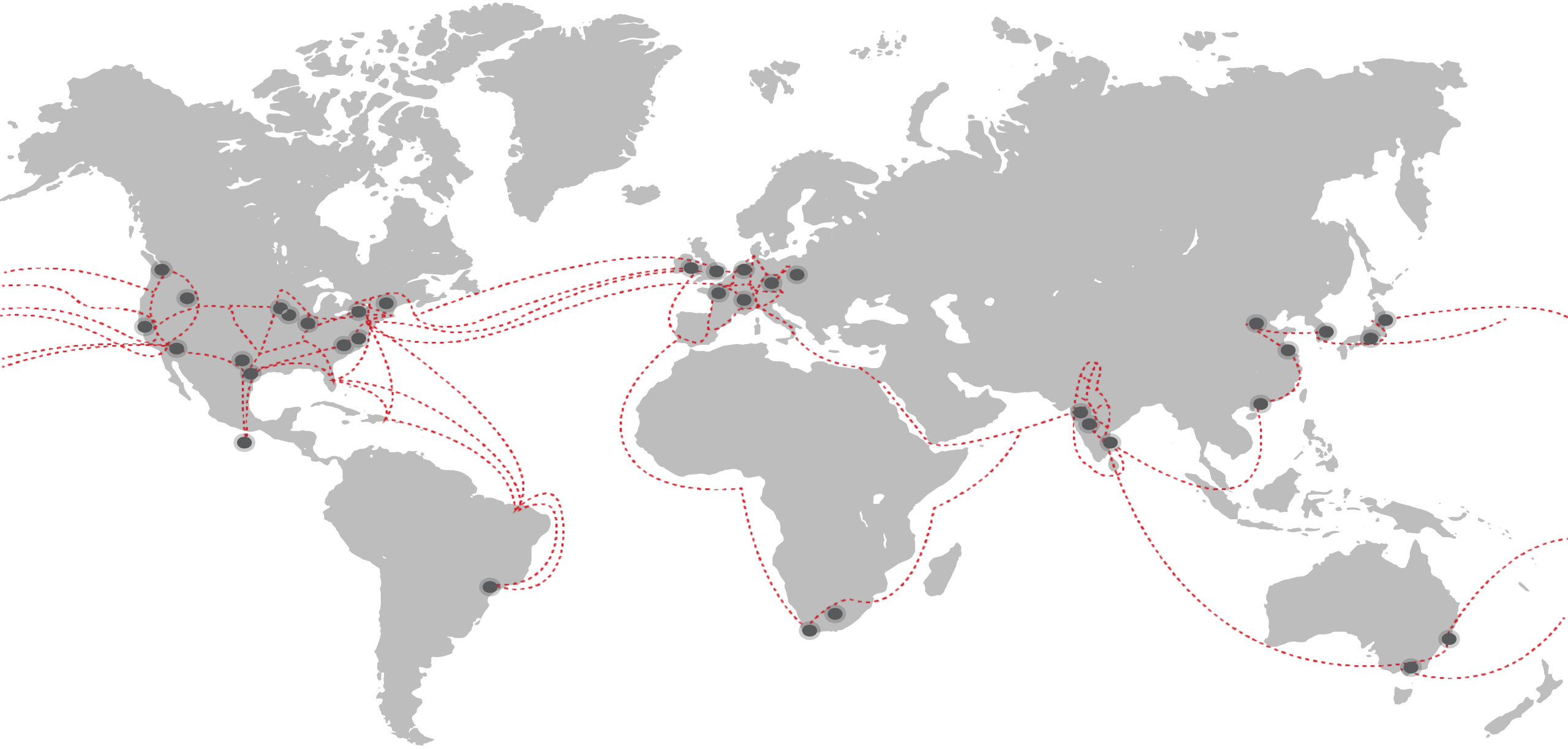
NV: A Framework for Modelling Network Protocols

Nick Giannarakis, **Devon Loehr**, Ryan Beckett, David Walker [pldi 2020]

Ryan Beckett, Aarti Gupta, Ratul Mahajan, David Walker [popl 2020]

Networks connect us to our online services



Network errors have a large blast radius

Microsoft Says Config. Change Caused Azure Outage

Microsoft: Misconfig Network Device Caus Outage

A misconfigured network device caused Thursd

Google cloud is down, affecting numerous applications and

Chad Fullerton
@chad_fullerton

With Confidence In AWS Shaken, Who Could Benefit?

Amazon.com, Inc. (NASDAQ: AMZN) faced a setback Tuesday due to an outage at its cloud computing platform — Amazon Web Services, or AWS....

benzinga.com

Google details 'catastrophic' cloud outage events: Promises to do better next time

Data-center automation software was behind what Google describes as a "catastrophic failure" last Sunday.

By Liam Tung | June 7, 2019 -- 12:39 GMT (05:39 PDT) | Topic: Cloud

Amazon's massive AWS outage was caused by human error

One incorrect command and the whole internet suffers.

By Jason Del Rey | @DelRey | Mar 2, 2017, 2:20pm EST

The Problem

Configs are:

Low-level

Large

Ad hoc

Complicated

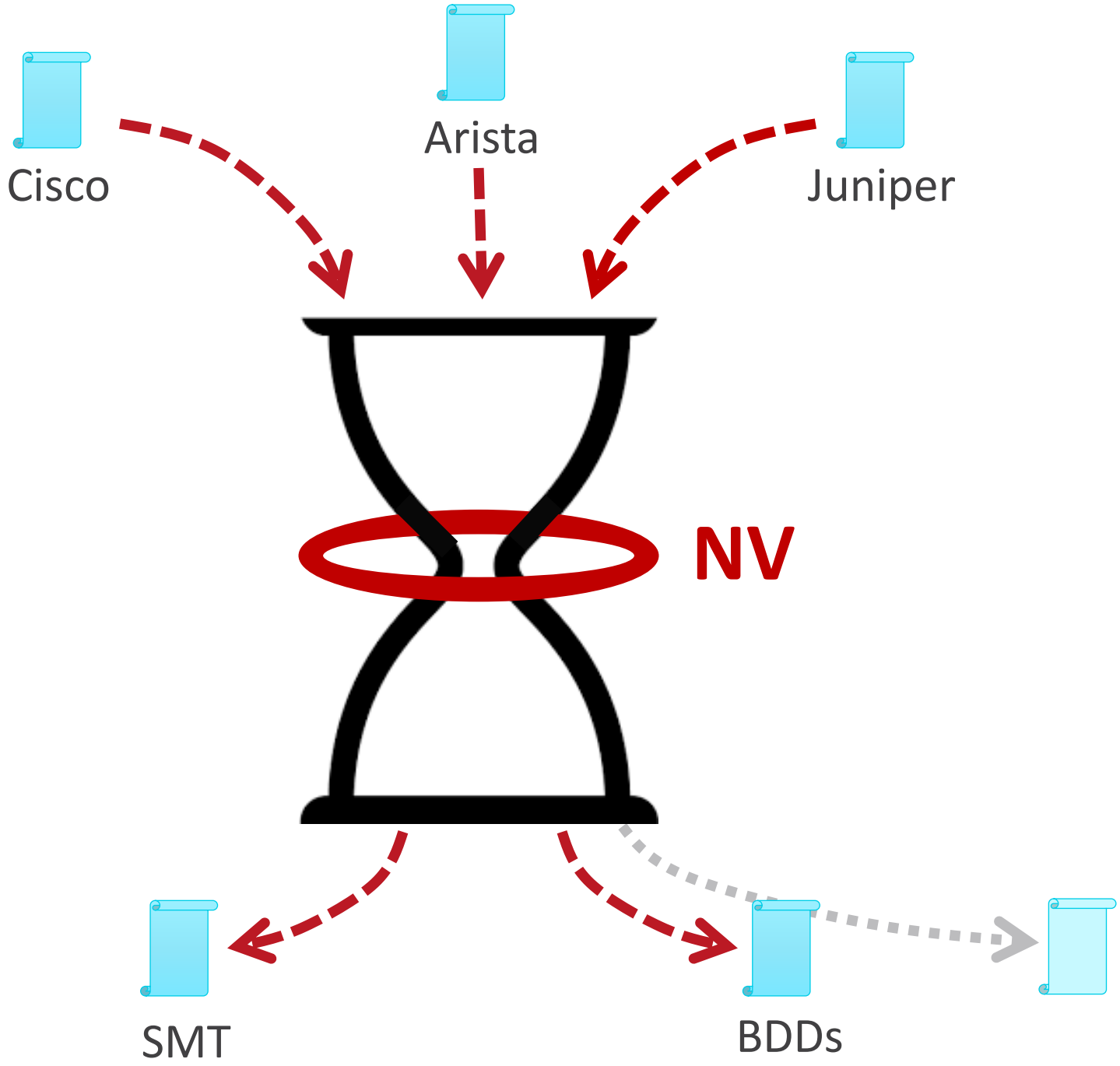
Non-compositional

Distributed

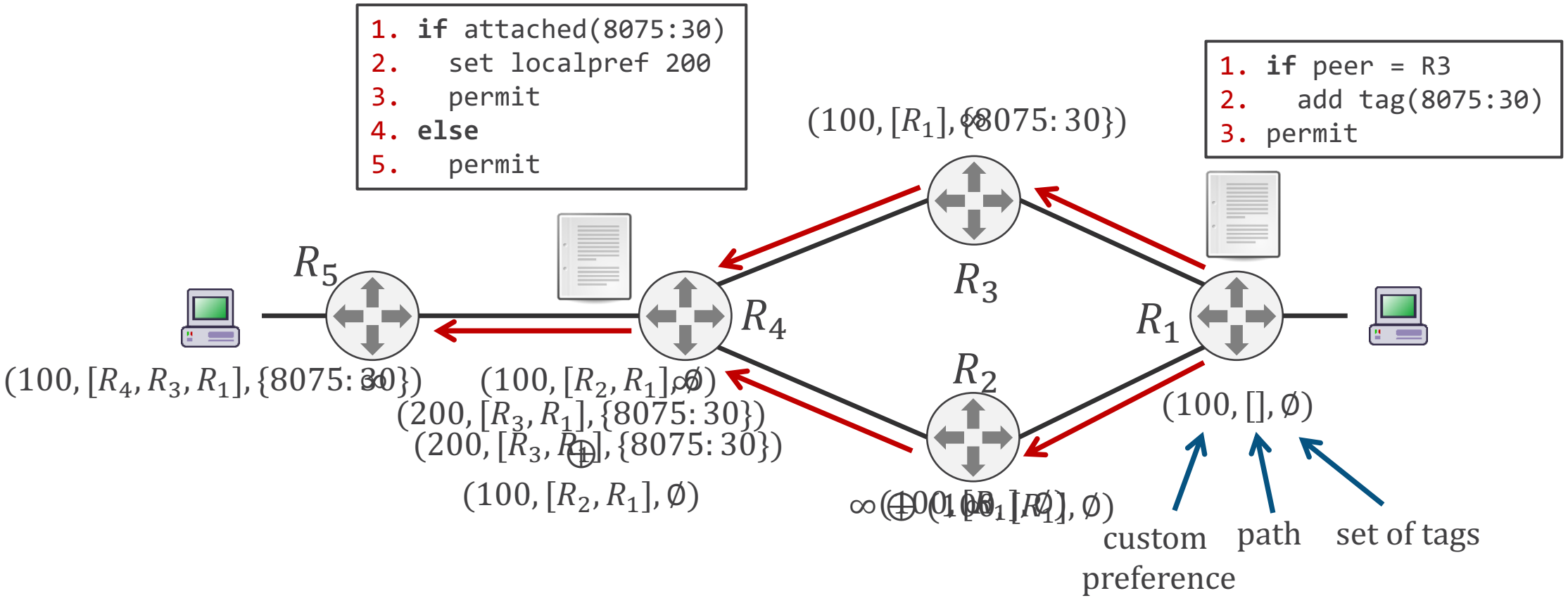
Produced by many vendors

```
1    interface Ethernet0
2      ip address 172.16.0.0/31
       ... configuring topology

500  ip route 192.168.1.0 255.255.255.0 192.168.2.0
     ... static routes

800  bgp router 1
801    redistribute static
802    neighbor 172.16.0.1 remote-as 2
803    neighbor 172.16.0.1 route-map RMO out
       ... Configuring BGP connections

2000 router ospf 1
2001   redistribute static metric 20 subnets
2002   distance 70
2003   network 192.168.42.0 0.0.0.255 area 0
       ... Configuring OSPF connections

3000 ip community-list standard comm1 permit 1:2 1:3
3001 ip prefix-list pfx permit 192.168.2.0/24
3002 route-map RMO permit 10
3003   match community comm1
3004   match ip address prefix-list pfx
3005   set local-preference 200
3006 route-map RMO permit 20
3007   set metric 90
       ... Configuring routing policies
```

The Solution

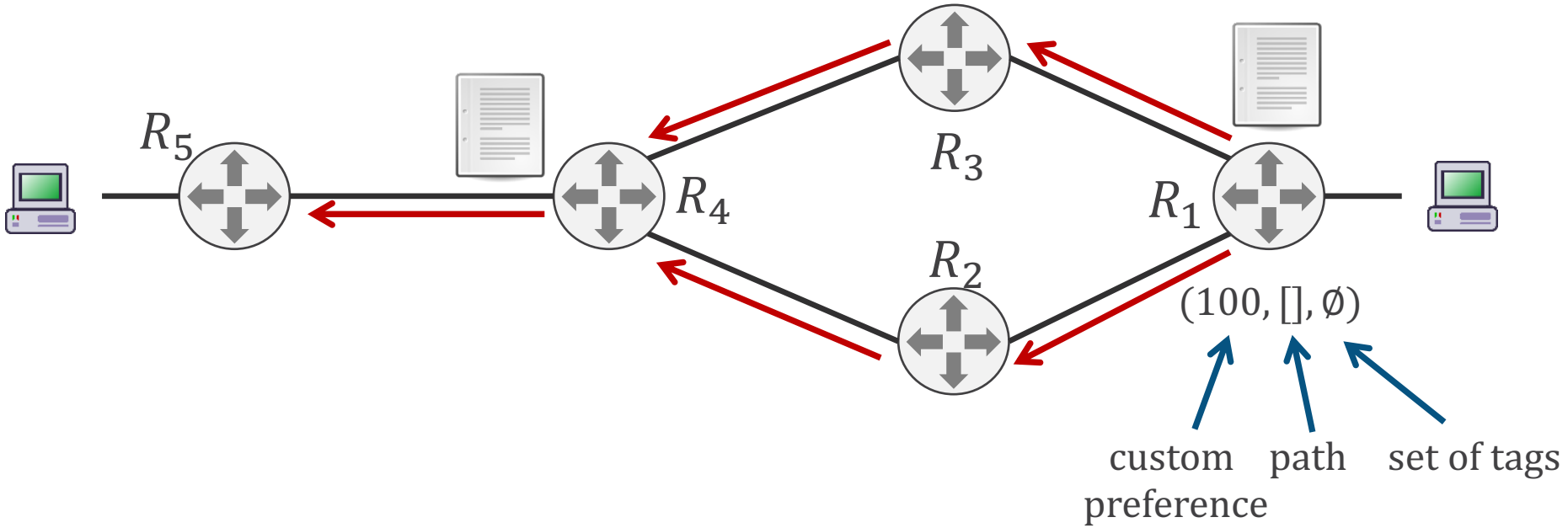


Example 1: Idealized BGP



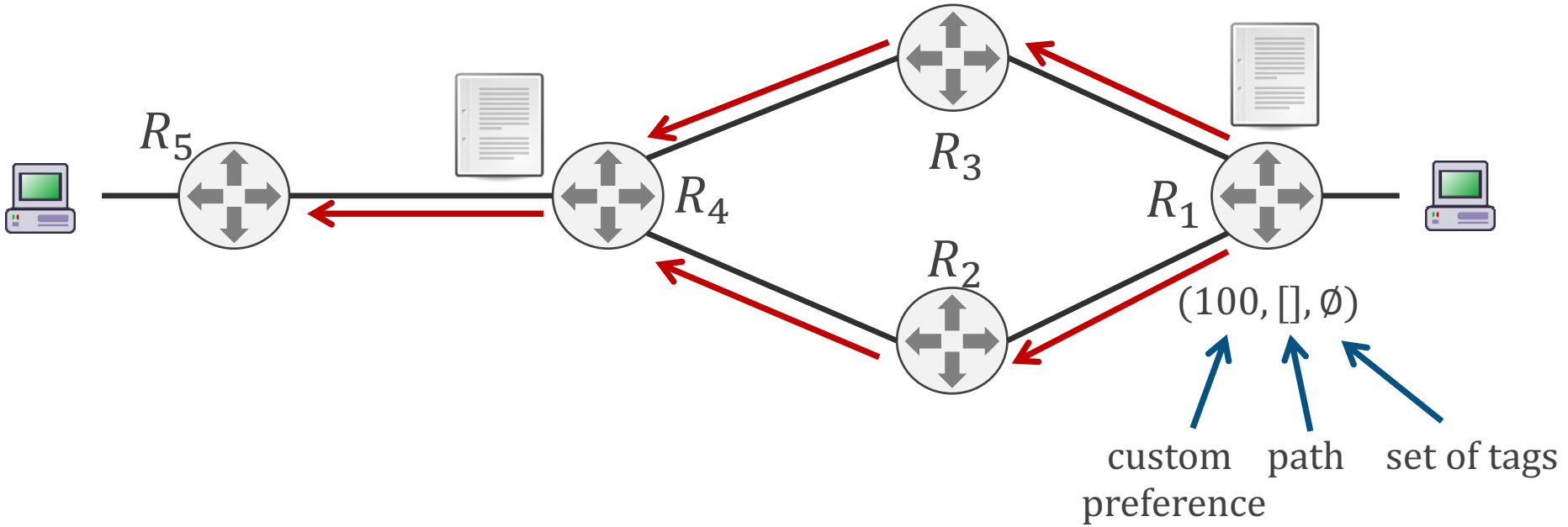
So, to model routing, we need:

- a network topology
- a type for routes
- an initial state
- route transfer and merge operations



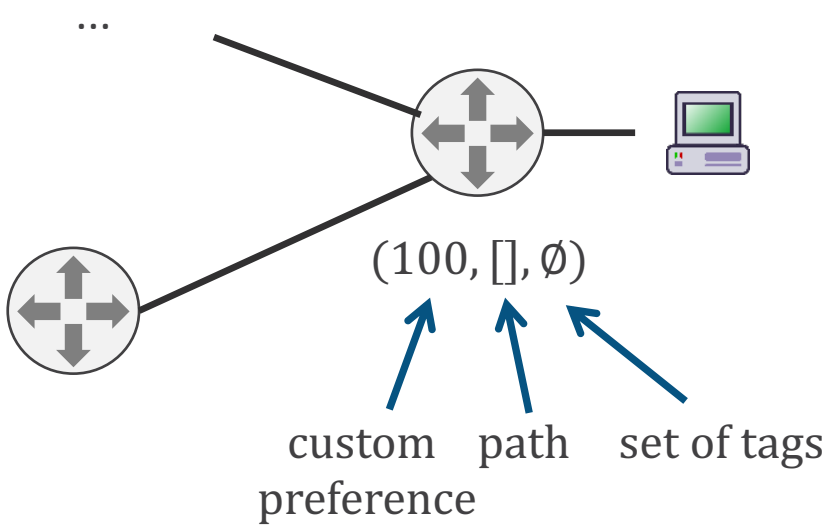
So, to model routing, we need:

- a network topology ..... a graph
- a type for routes ..... types for finite functional data
- an initial state ..... initial value
- route transfer and merge operations ..... non-recursive functions



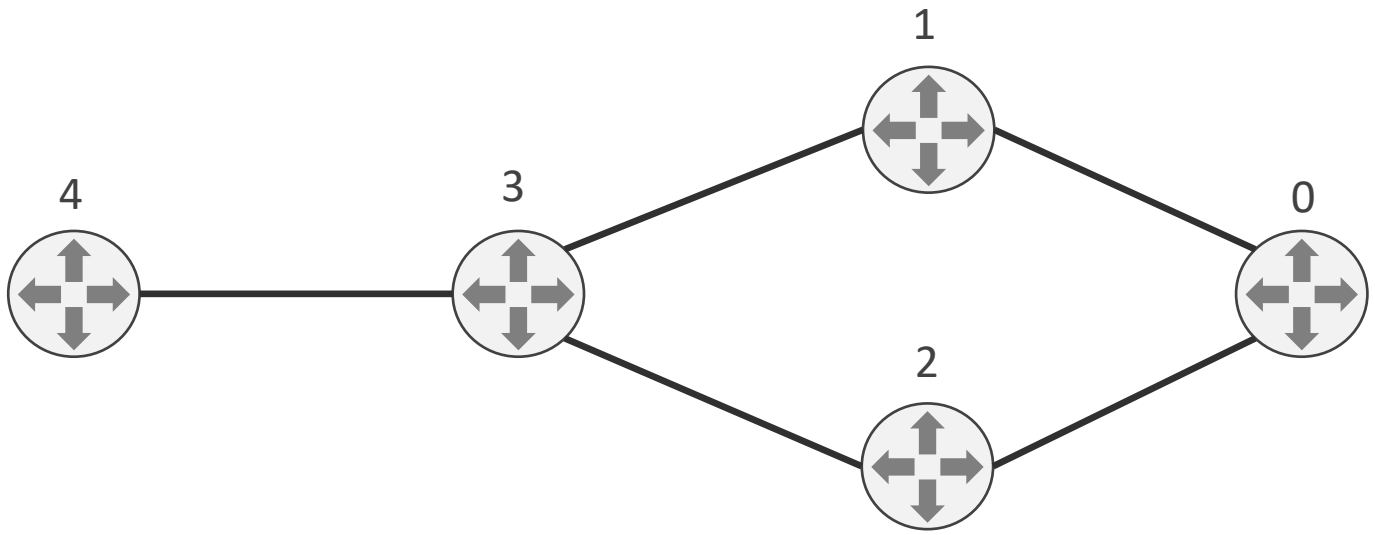
Modelling Routes (v 0.1)

```
type bgp = { lp   : int;          (* local preference *)
             path : int;          (* path length      *)
             comm : bool; }       (* community tag    *)

type attribute = option[bgp]      (* route or not *)
```



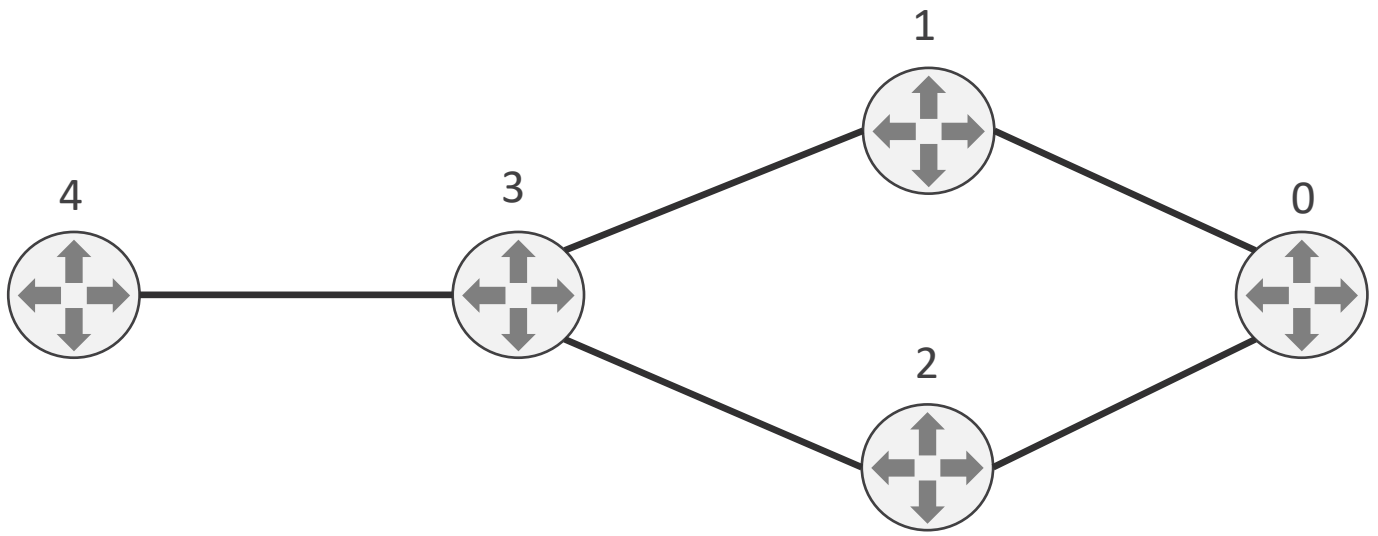
Modelling Topology

```
let nodes = 5
let edges = {0=1; 0=2; 1=3; 2=3; 3=4}
```



Modelling Route Propagation

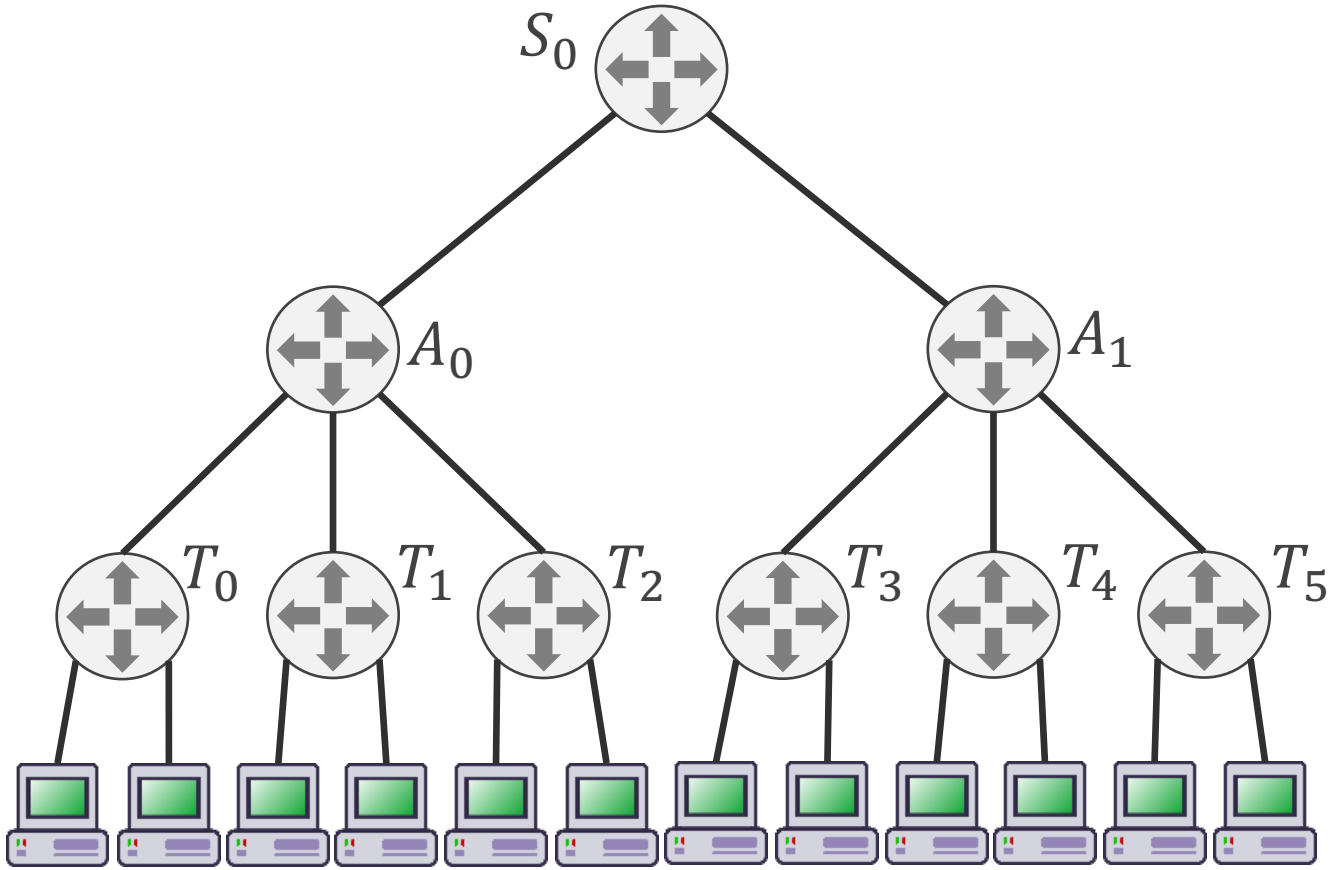
```
(* transfer route along an edge *)
let trans (e:edge) (r:attribute) : attribute = ...

(* merge neighboring attributes with your initial attribute *)
let merge (n:node) (r1:attribute) (r2:attribute) : attribute =
...
```



Checking Properties

```
(* check reachability of route for prefix to all nodes n *)
assert (n:node) (r:attribute) : bool =
  let prefix = (100..., 24) in
  match r[prefix] with
    None -> false
  | Some _ -> true
```

Example 2: Data Center Routing

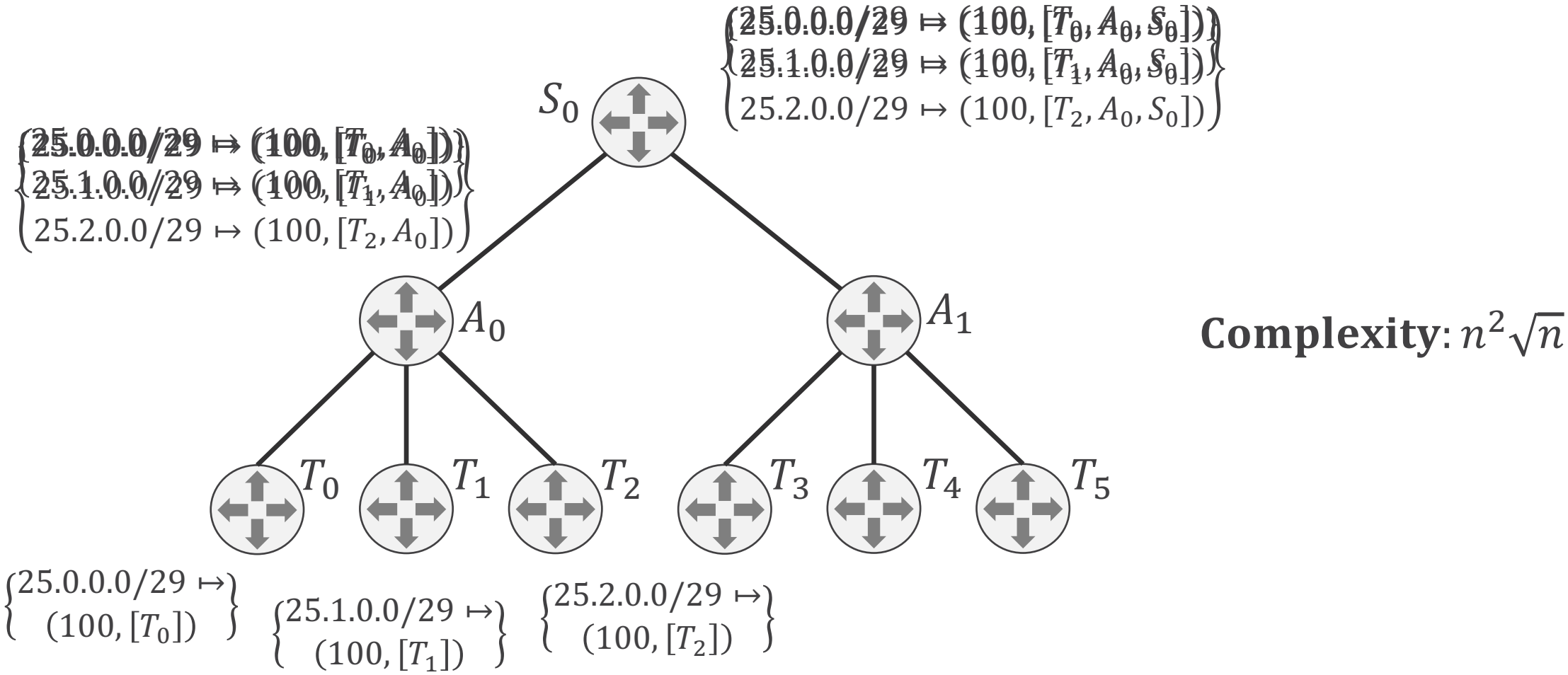


Spine Routers (S)

Aggregation Routers (A)

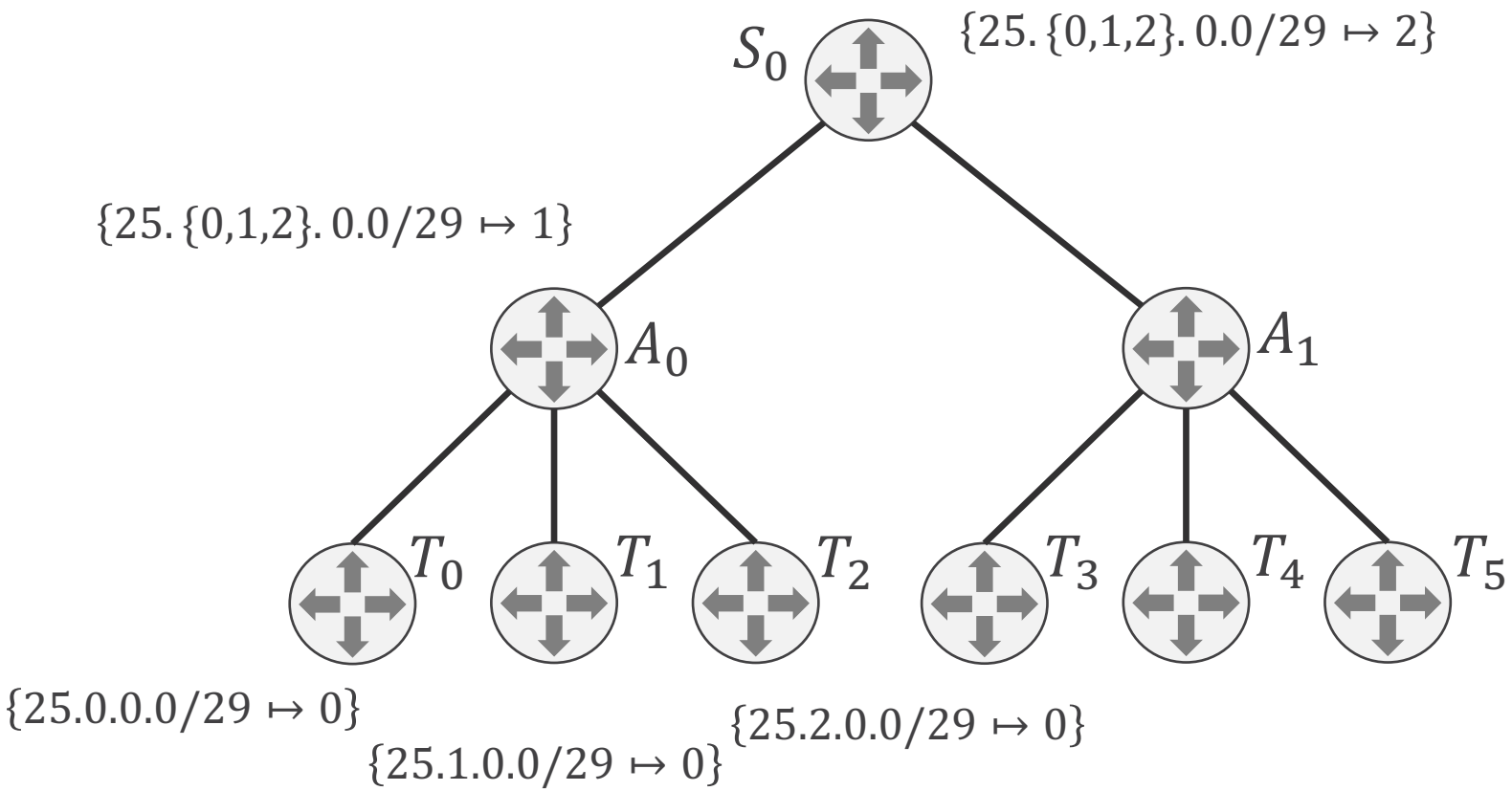
Top-of-Rack Routers (T)

Example 2: Data Center Routing



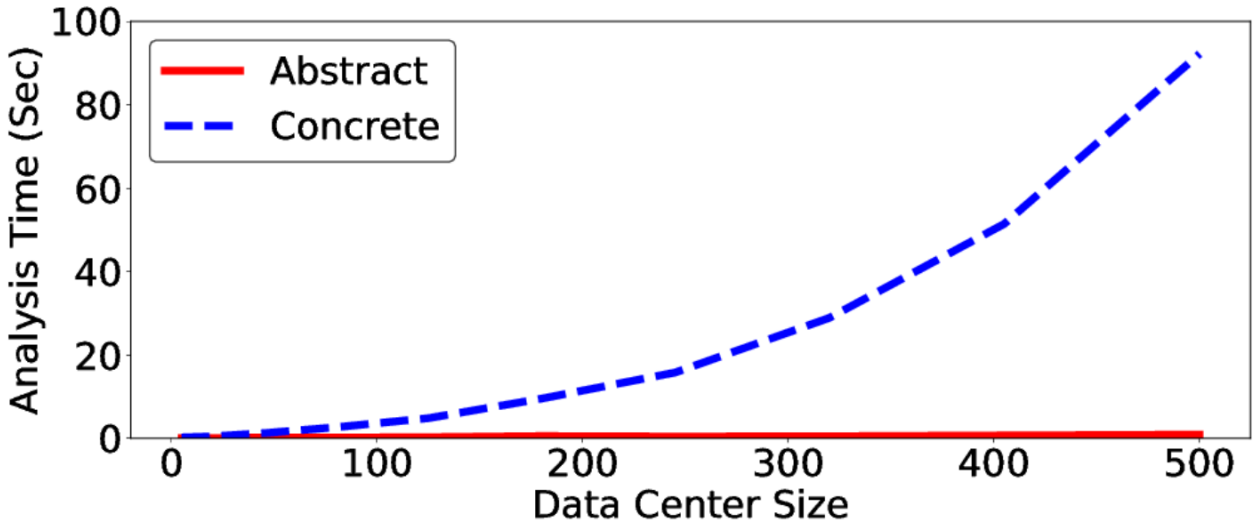
Complexity: $n^2\sqrt{n}$

Example 2: Data Center Routing



Complexity: $n\sqrt{n}$

Empirical Results



Simulation time vs. data center size
for verifying all-pairs connectivity

Two Key Ideas

(1) Choose the right abstractions

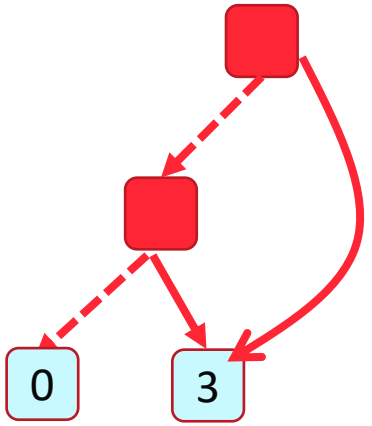
$$\begin{Bmatrix} 25.0.0.0/29 \mapsto (100, [T_0, A_0, S_0]) \\ 25.1.0.0/29 \mapsto (100, [T_1, A_0, S_0]) \\ 25.2.0.0/29 \mapsto (100, [T_2, A_0, S_0]) \end{Bmatrix} \longrightarrow \begin{Bmatrix} 25.0.0.0/29 \mapsto 3 \\ 25.1.0.0/29 \mapsto 3 \\ 25.2.0.0/29 \mapsto 3 \end{Bmatrix}$$

leaves can't be shared/ represented compactly

leaves can be shared

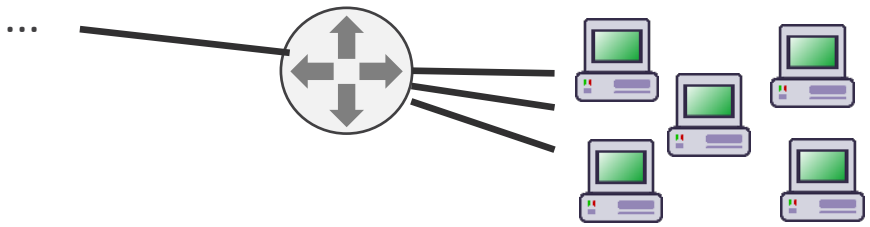
(2) Represent maps compactly

- We use multi-terminal BDDs (mtBDDs)
- Map operations are designed to keep mtBDDs skinny



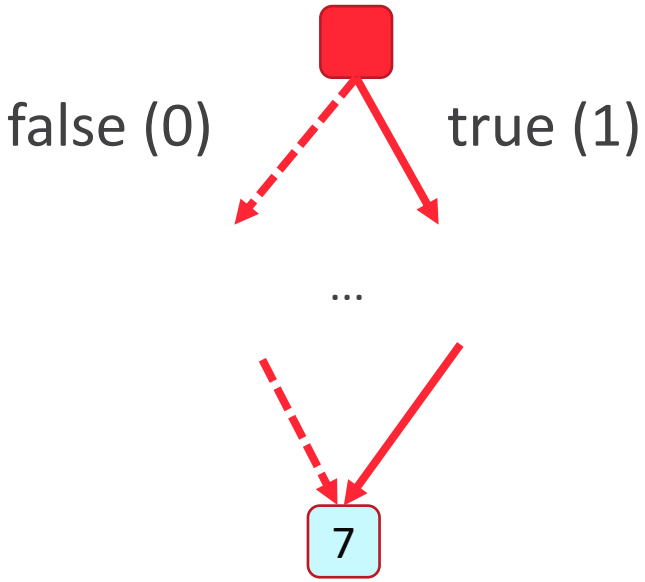
Modelling Routes (v 2.0)

```
type bgp = { path : int; }      (* path length          *)

type prefix = int * int5        (* (dest IP, length)  *)

type attribute = dict[prefix, option[bgp]]
```



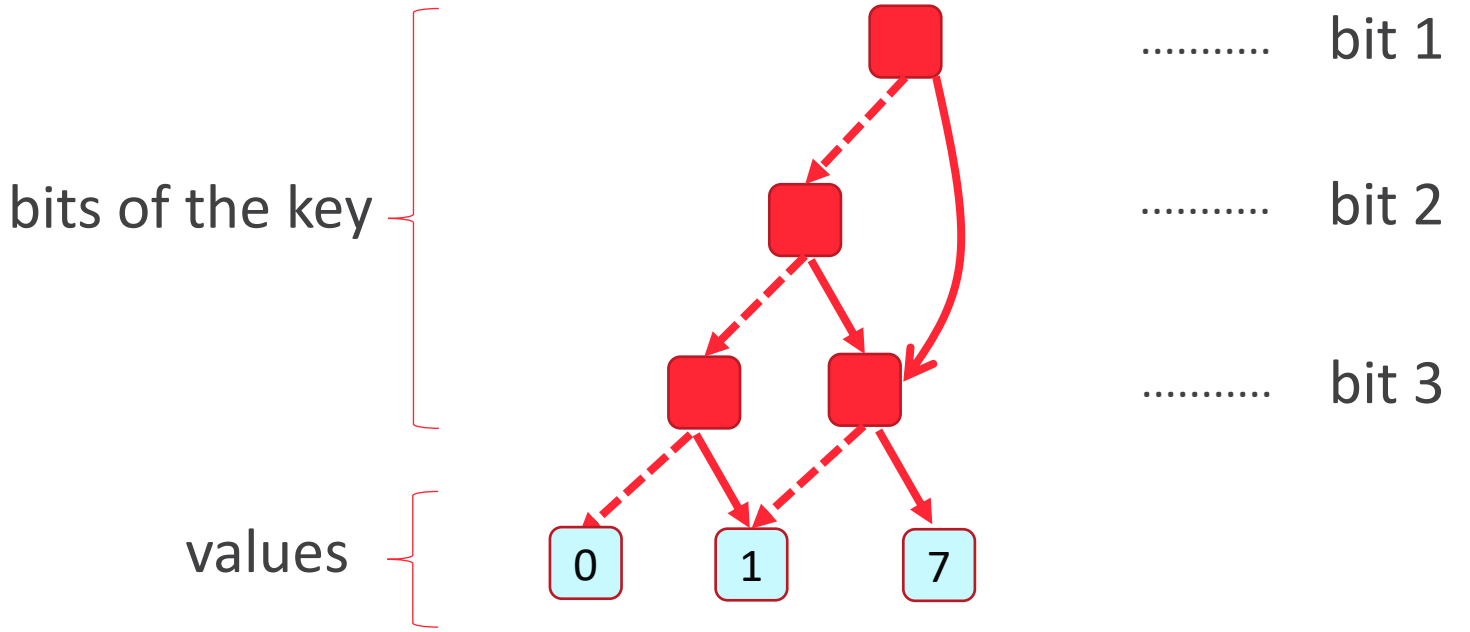
Maps as Multi-terminal BDDs

internal nodes are bits of the key



terminal nodes are values

Maps as Multi-terminal BDDs



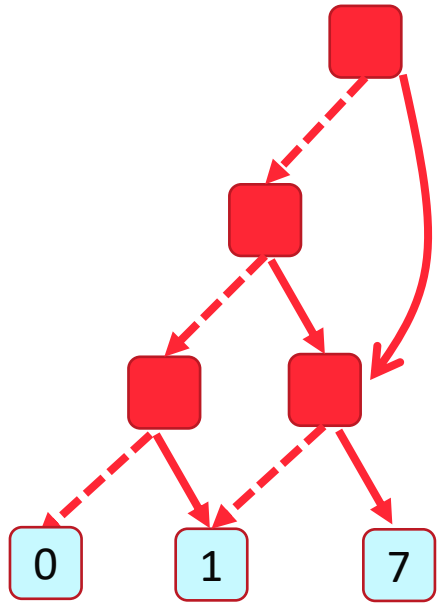
represents:

$$\begin{Bmatrix} 000 \mapsto 0 \\ 001 \mapsto 1 \\ 010 \mapsto 1 \\ 011 \mapsto 7 \\ 100 \mapsto 1 \\ \ldots \end{Bmatrix}$$

Maps as Multi-terminal BDDs

MtBDDs are good representations when skinny

- map over values keeps them skinny
- map over keys + values can blow up
- mapife p f1 f2: splits at most by a factor of two



Tends to work well in networks, which have a lot of symmetry

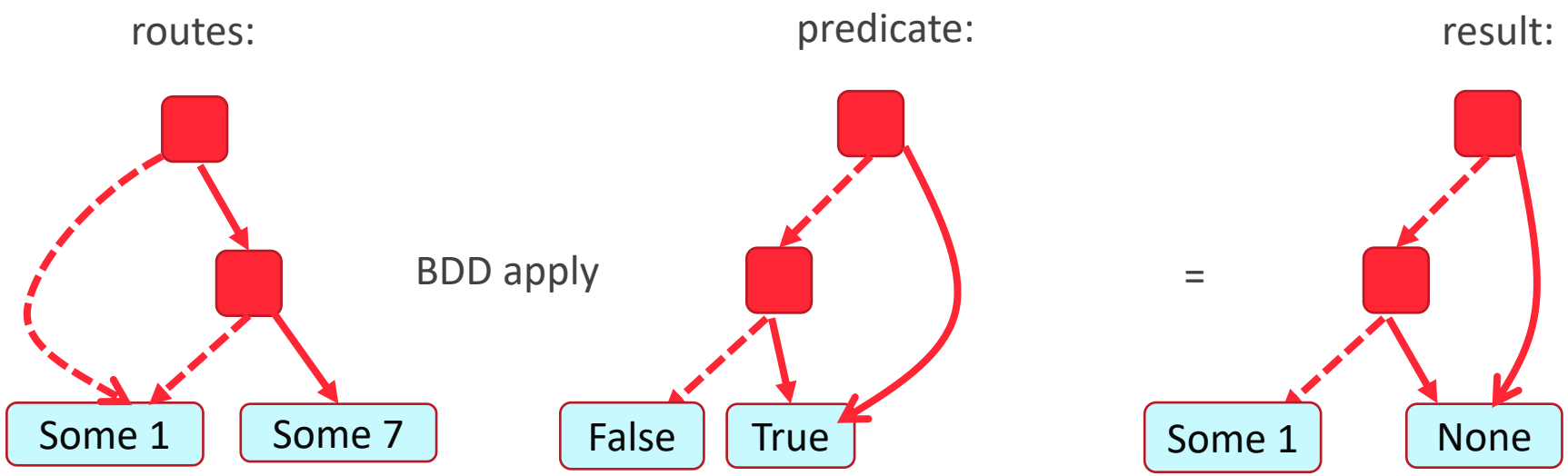
Modelling Route Propagation

applied to leaves of the BDD

```
...
type attribute = dict[prefix, option[bgp]]

(* propagation of single BGP route *)
let trans_bgp (bgp:bgp) =
  match bgp with
     None -> None
   | Some {lp=lp; len=l; comm=c} ->
       Some {lp=lp; len=l+1; comm=c}

(* propagation of all BGP routes *)
let trans_bgp_all (r:attribute) =
  map trans_bgp r
```

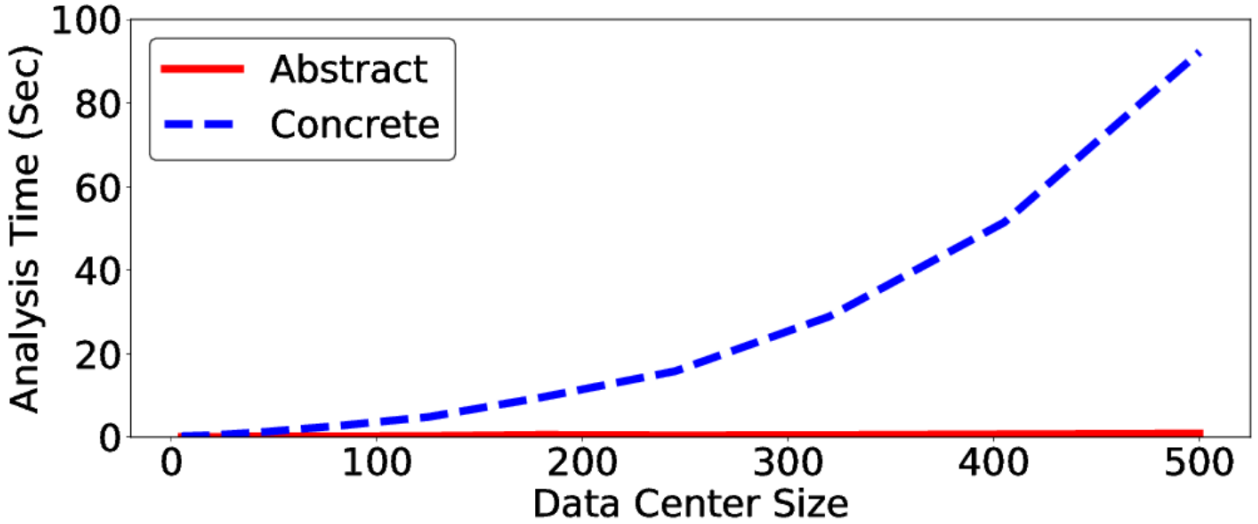
Modelling Route Propagation

```
(* define predicate p over keys *)
let p (ip, length) = set.member bogons ip

(* define transformer f over values *)
let f v = None

(* custom filtering of routes *)
let permit (r:attribute) = mapif p f r
```



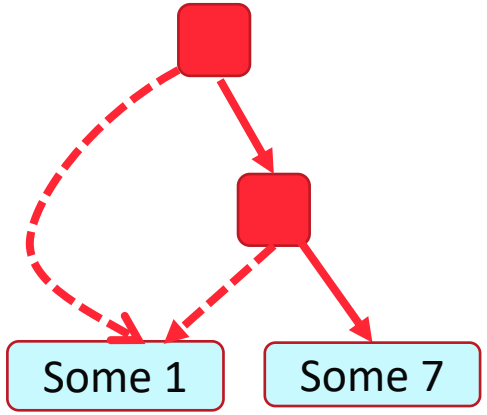
The Bottom Line

POPL 2020 (Hand-crafted analyses):



PLDI 2020: Design of NV. Programming new models.
New fault tolerance analyses.

More Generally

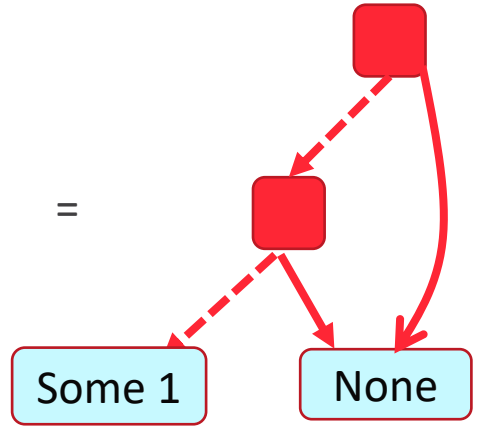


Treat collections as BDDs

```
filter f dict
```

Filter data/
Apply tests

=



Examine test results

More Generally

Can we design a sublanguage for manipulating BDDs?

Convert collections to BDDs?

Convert user functions to BDD-processing ones?

Deploy in QuickCheck/SmallCheck?

Kinda like a BDD-based version of Rosette?

What other application areas will benefit?

Summary and Conclusions

Network correctness is more important than ever

NV is a new functional language for modelling protocols and defining custom abstractions

It’s got an efficient BDD-based simulator (and an SMT solver) back end.