Modular Control Plane Verification via Temporal Invariants

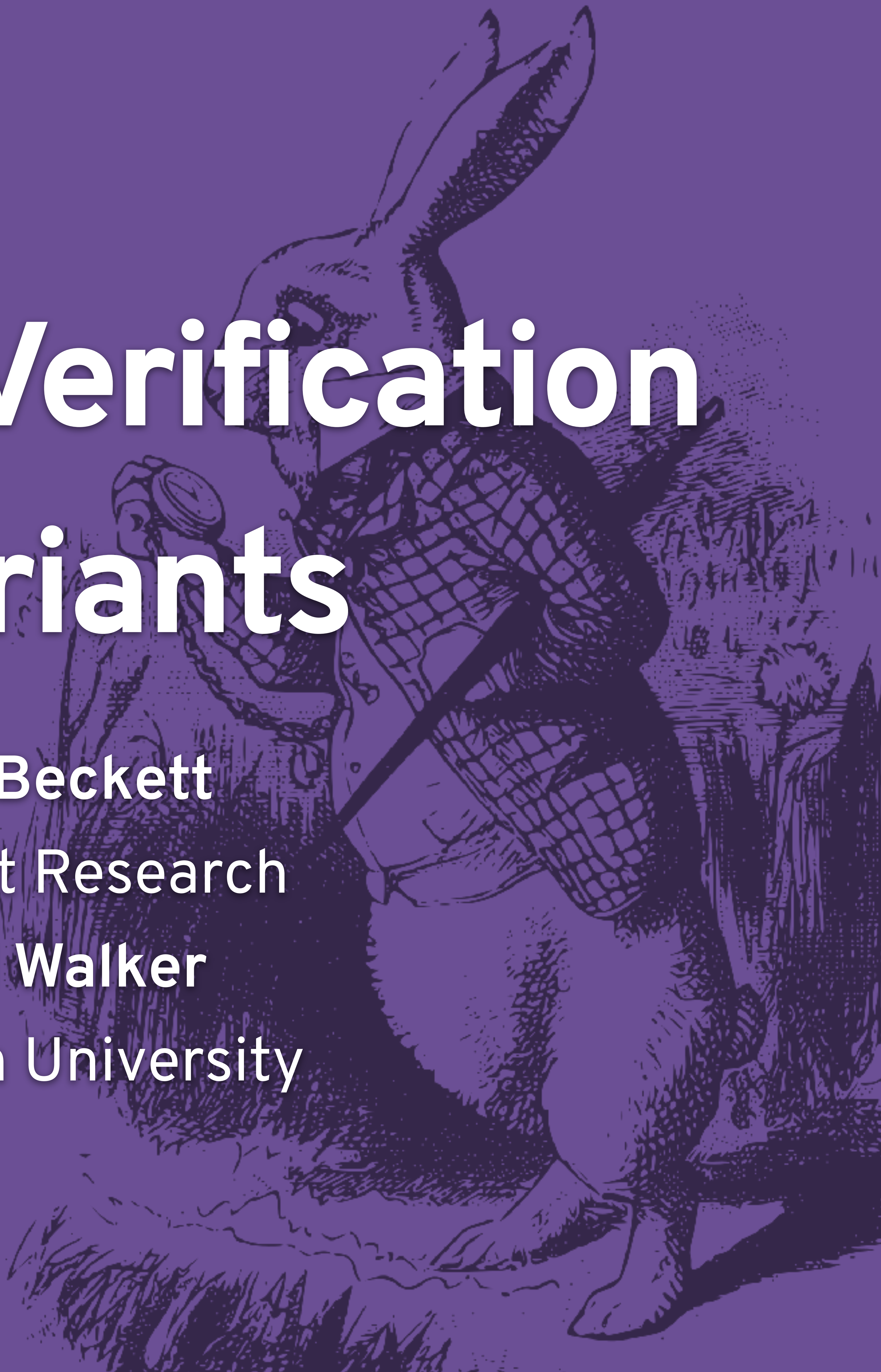
Timothy Alberdingk Thijm
Princeton University

Ryan Beckett
Microsoft Research

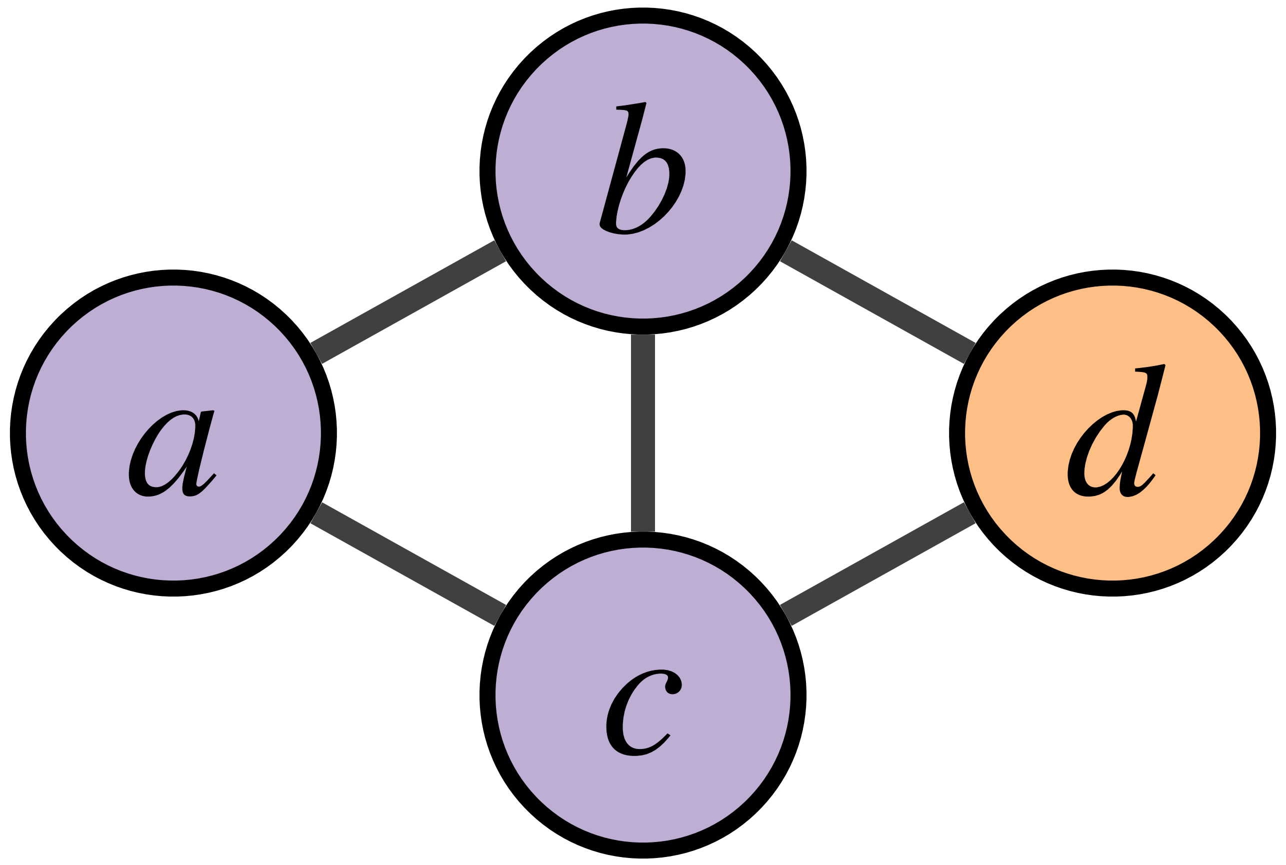
Aarti Gupta
Princeton University

David Walker
Princeton University

PLDI 2023

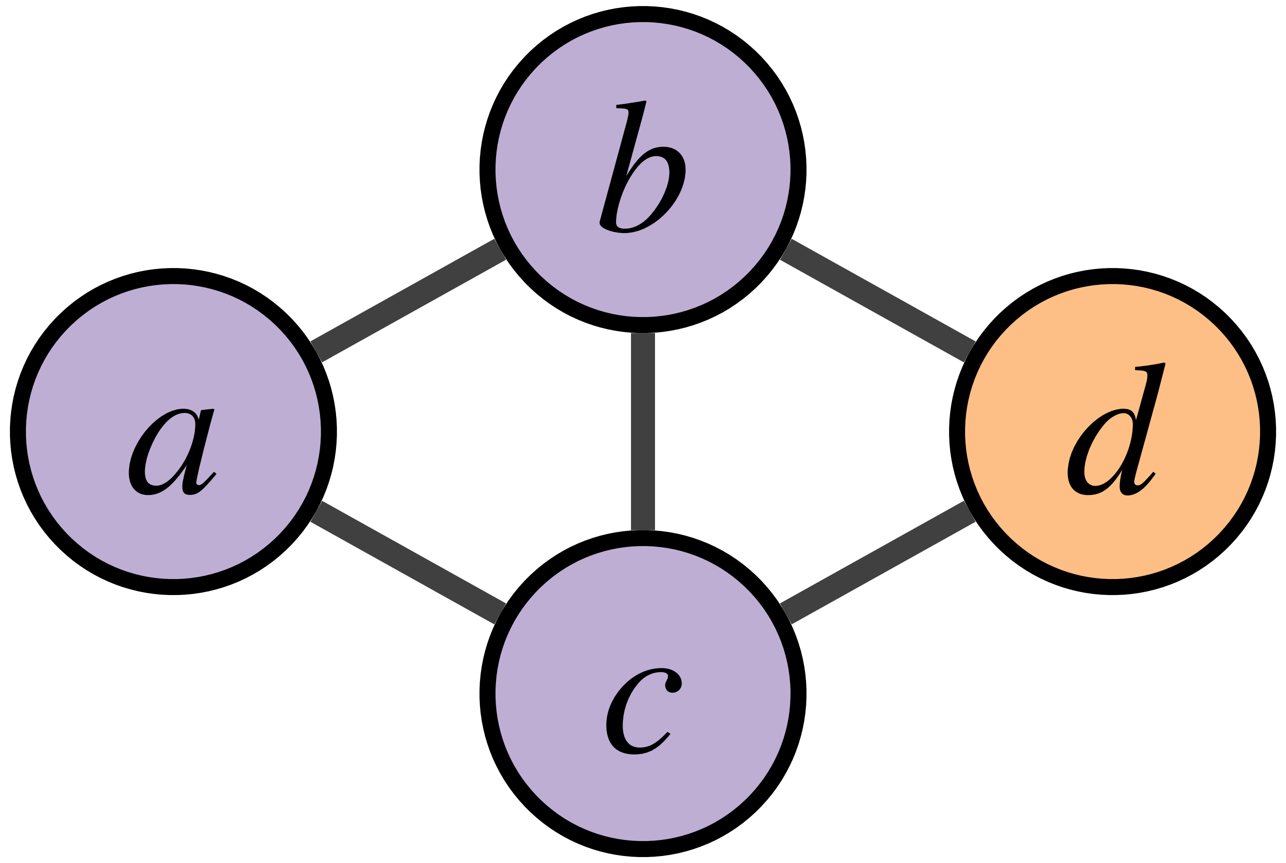


What is the Control Plane?



What is the Control Plane?

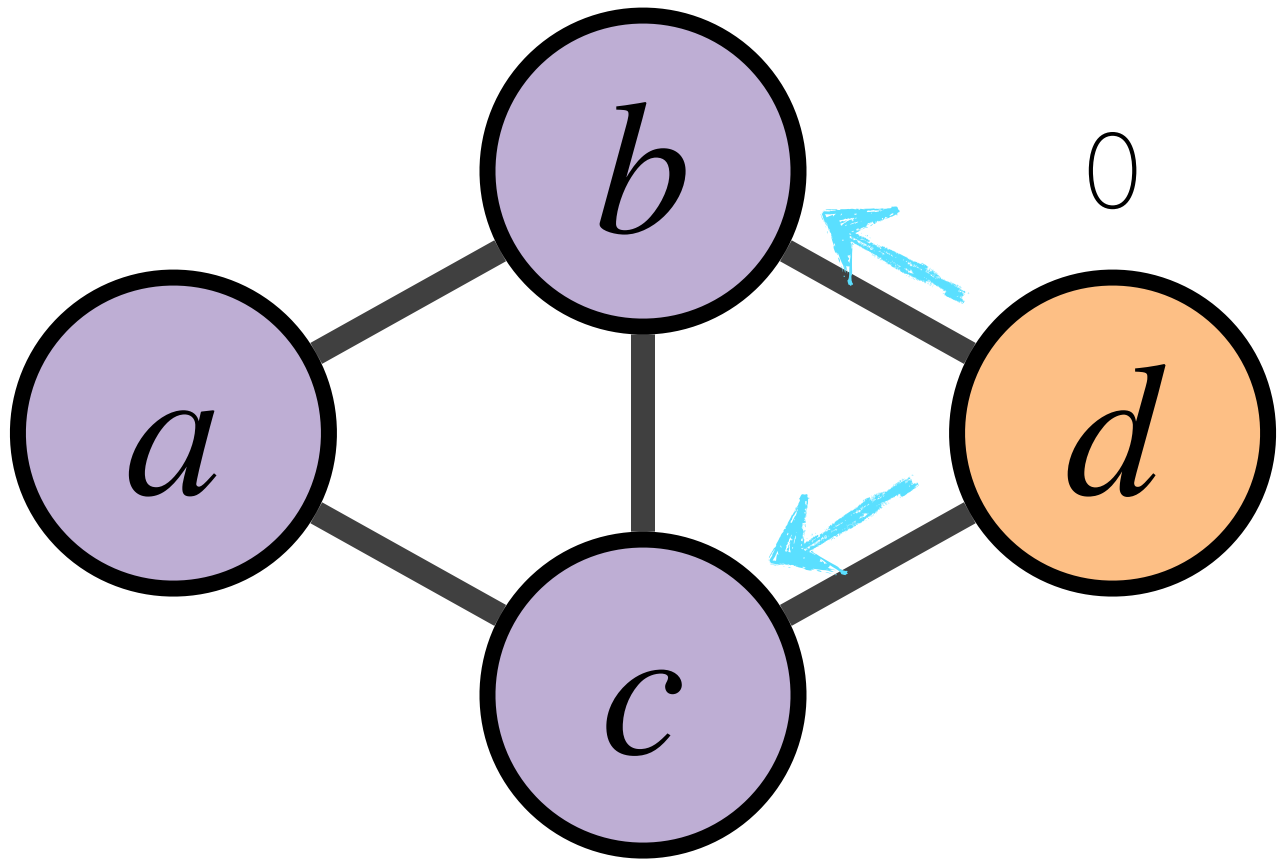
Goal: determine routes to use to forward traffic



What is the Control Plane?

Goal: determine routes to use to forward traffic

Send initial route announcements

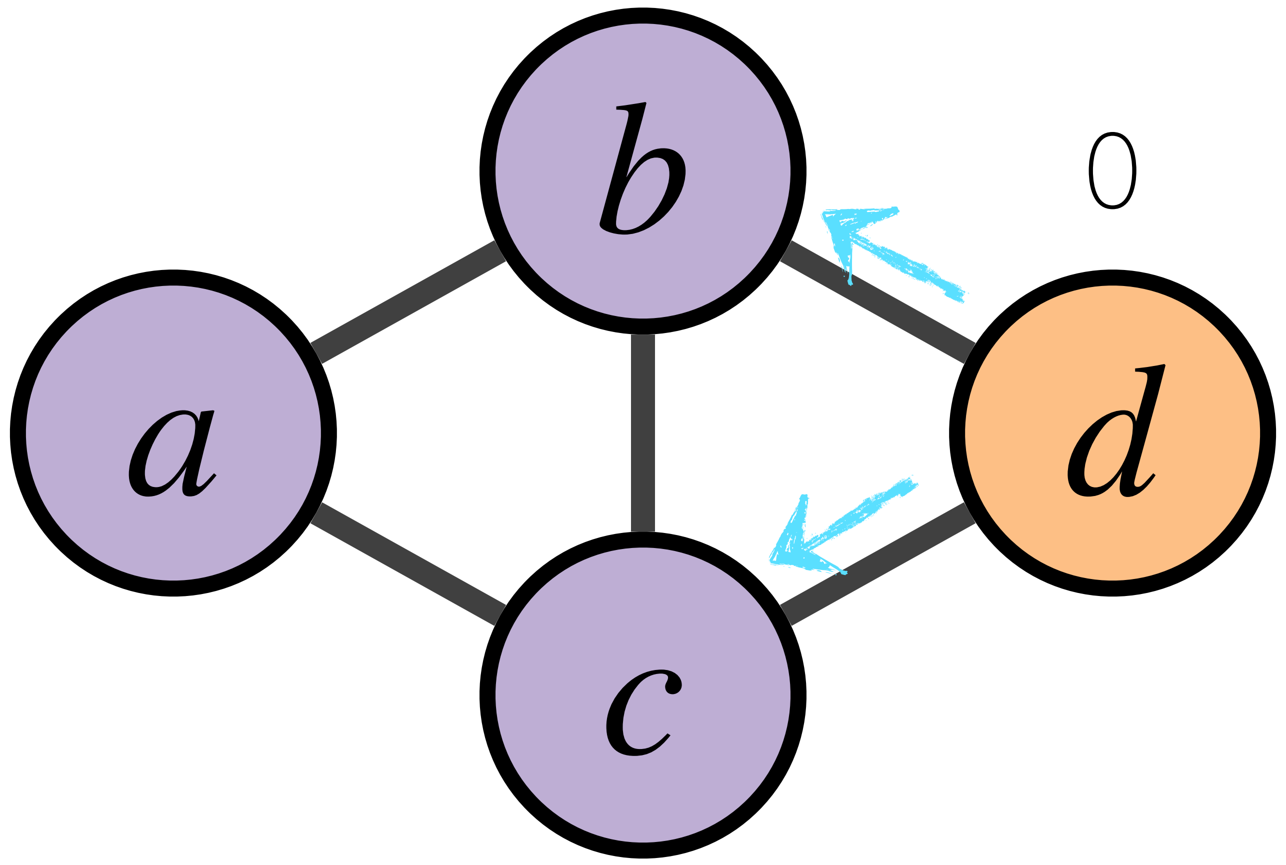


What is the Control Plane?

Goal: determine routes to use to forward traffic

Send initial route announcements

Receive announcements, process according to **configs**

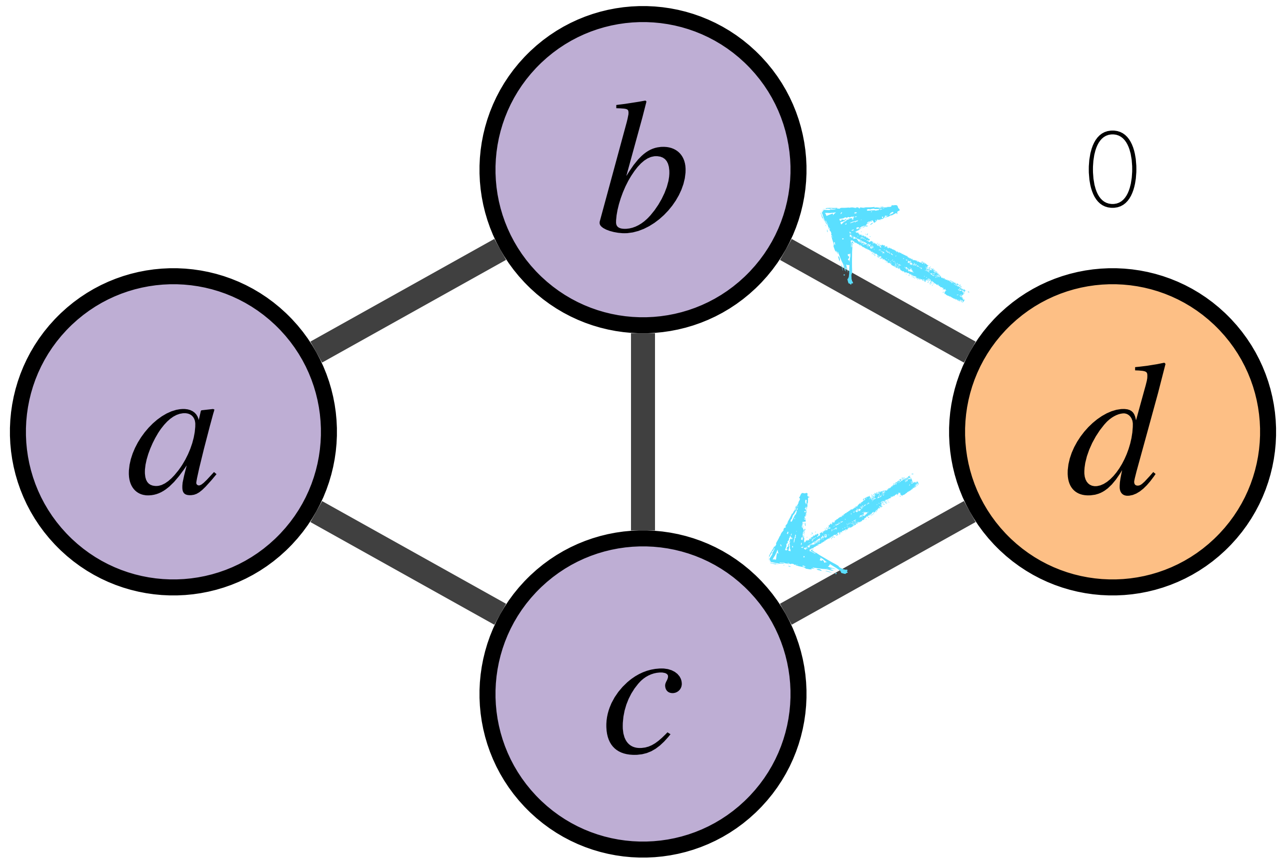


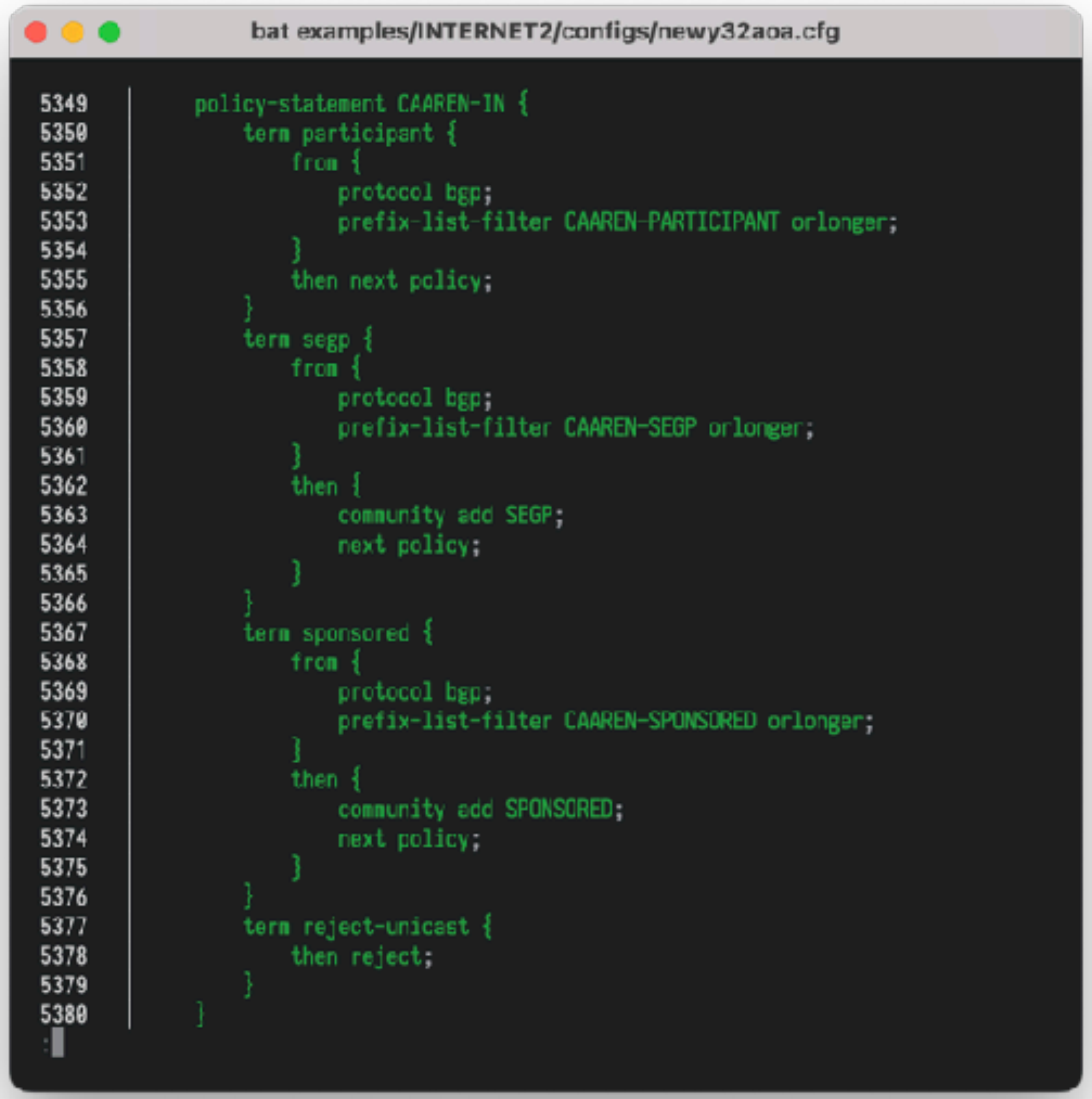
What is the Control Plane?

Goal: determine routes to use to forward traffic

Send initial route announcements

Receive announcements, process according to **configs**





Distributed, written in vendor-specific, low-level language

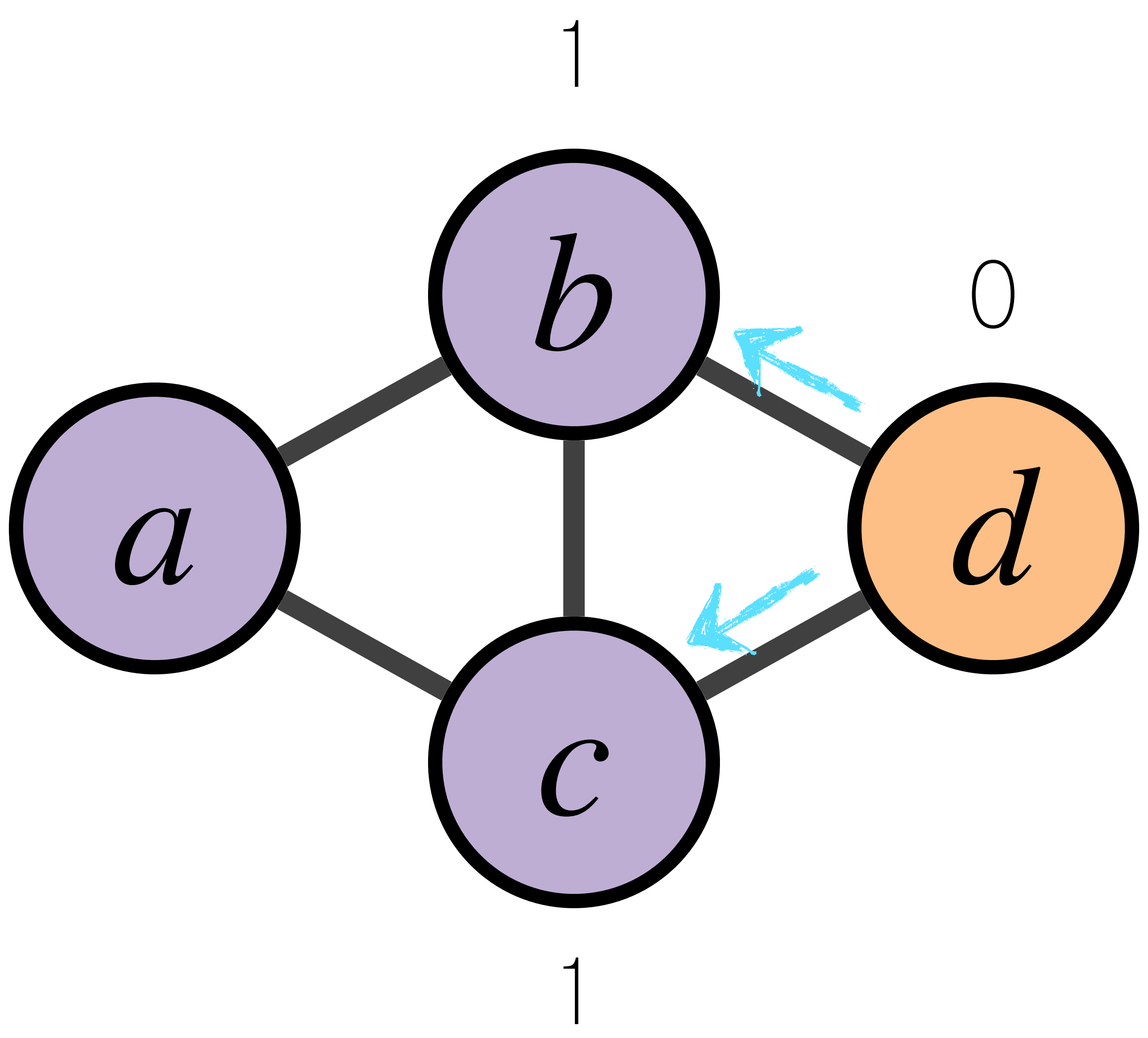
What is the Control Plane?

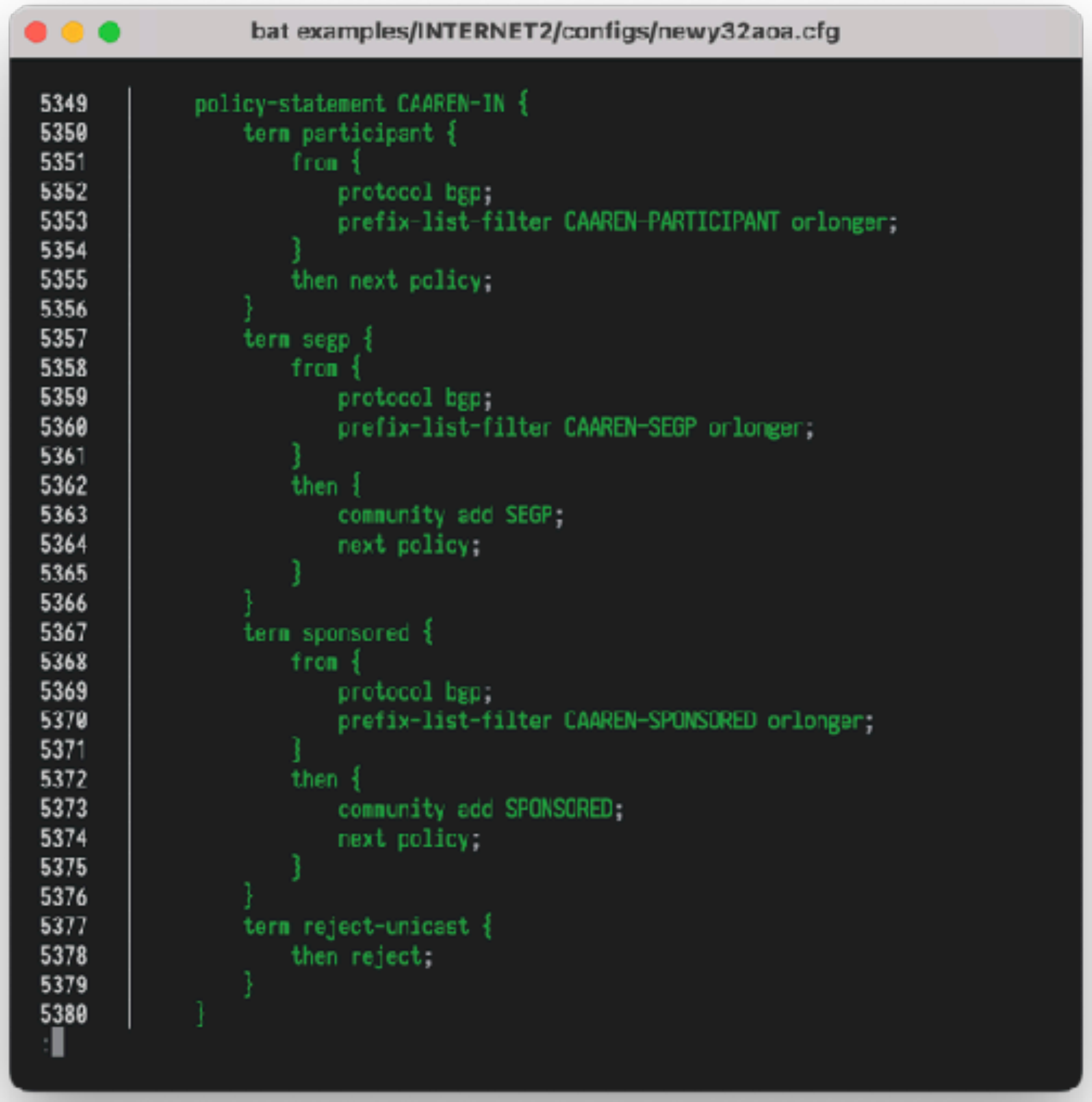
Goal: determine routes to use to forward traffic

Send initial route announcements

Receive announcements, process according to **configs**

Select best announcement





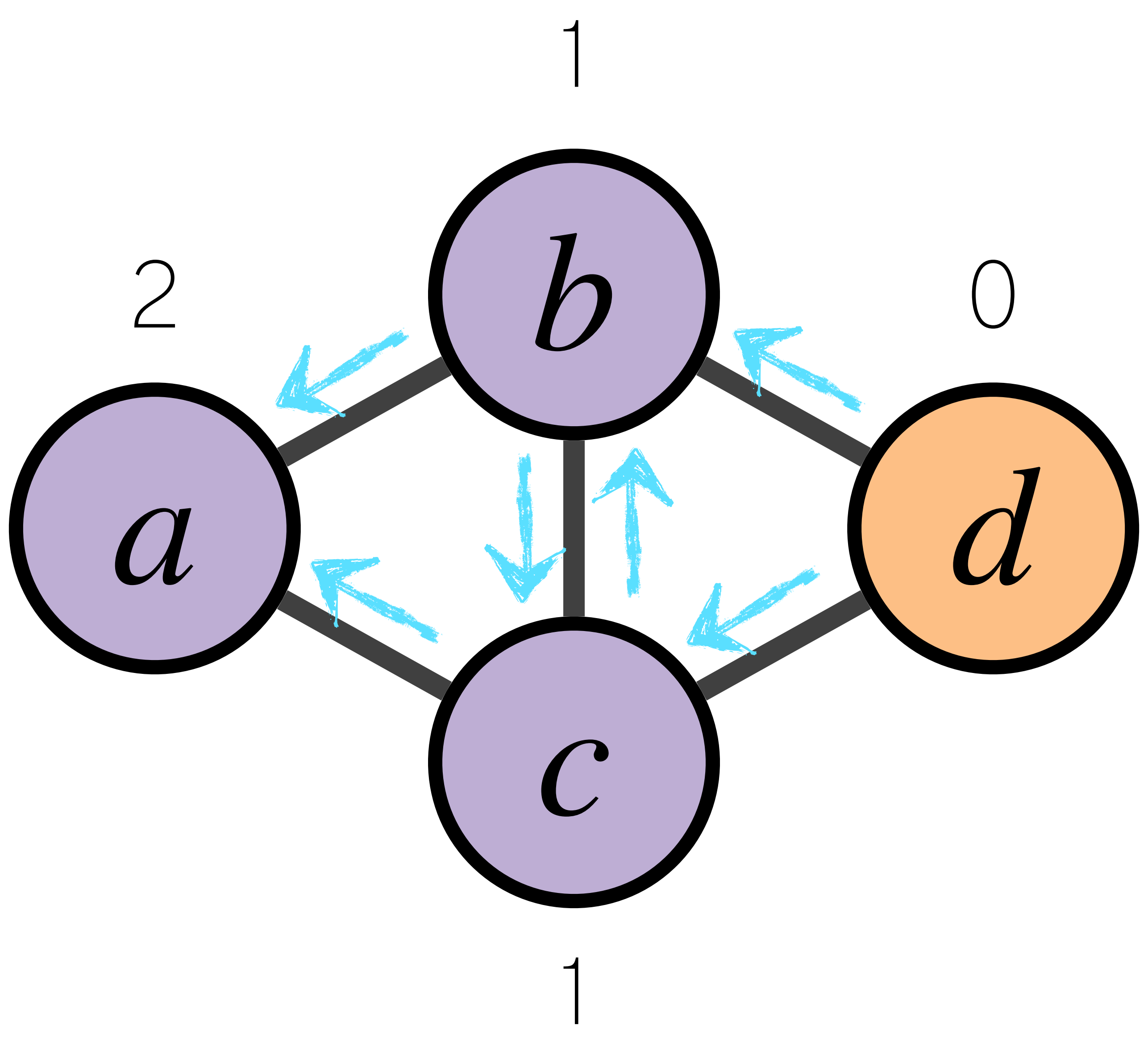

```
policy-statement CAAREN-IN {
    term participant {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-PARTICIPANT orlonger;
        }
        then next policy;
    }
    term segp {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-SEGP orlonger;
        }
        then {
            community add SEGP;
            next policy;
        }
    }
    term sponsored {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-SPONSORED orlonger;
        }
        then {
            community add SPONSORED;
            next policy;
        }
    }
    term reject-unicast {
        then reject;
    }
}
```

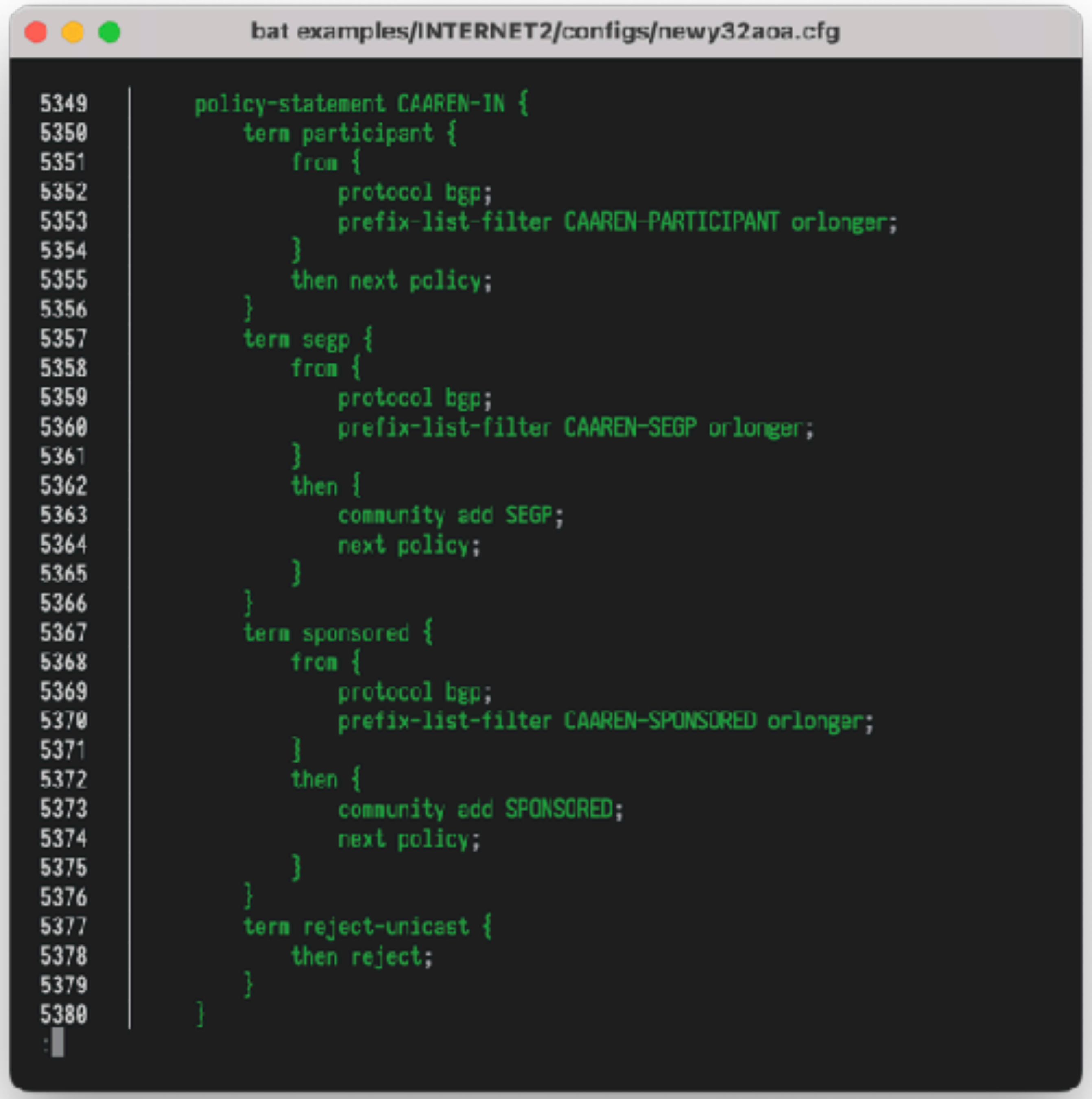
Distributed, written in vendor-specific, low-level language

What is the Control Plane?

Goal: determine routes to use to forward traffic

- Send initial route announcements
- Receive announcements, process according to **configs**
- Select best announcement
- Broadcast selected route to neighbors





bat examples/INTERNET2/configs/newy32aoa.cfg

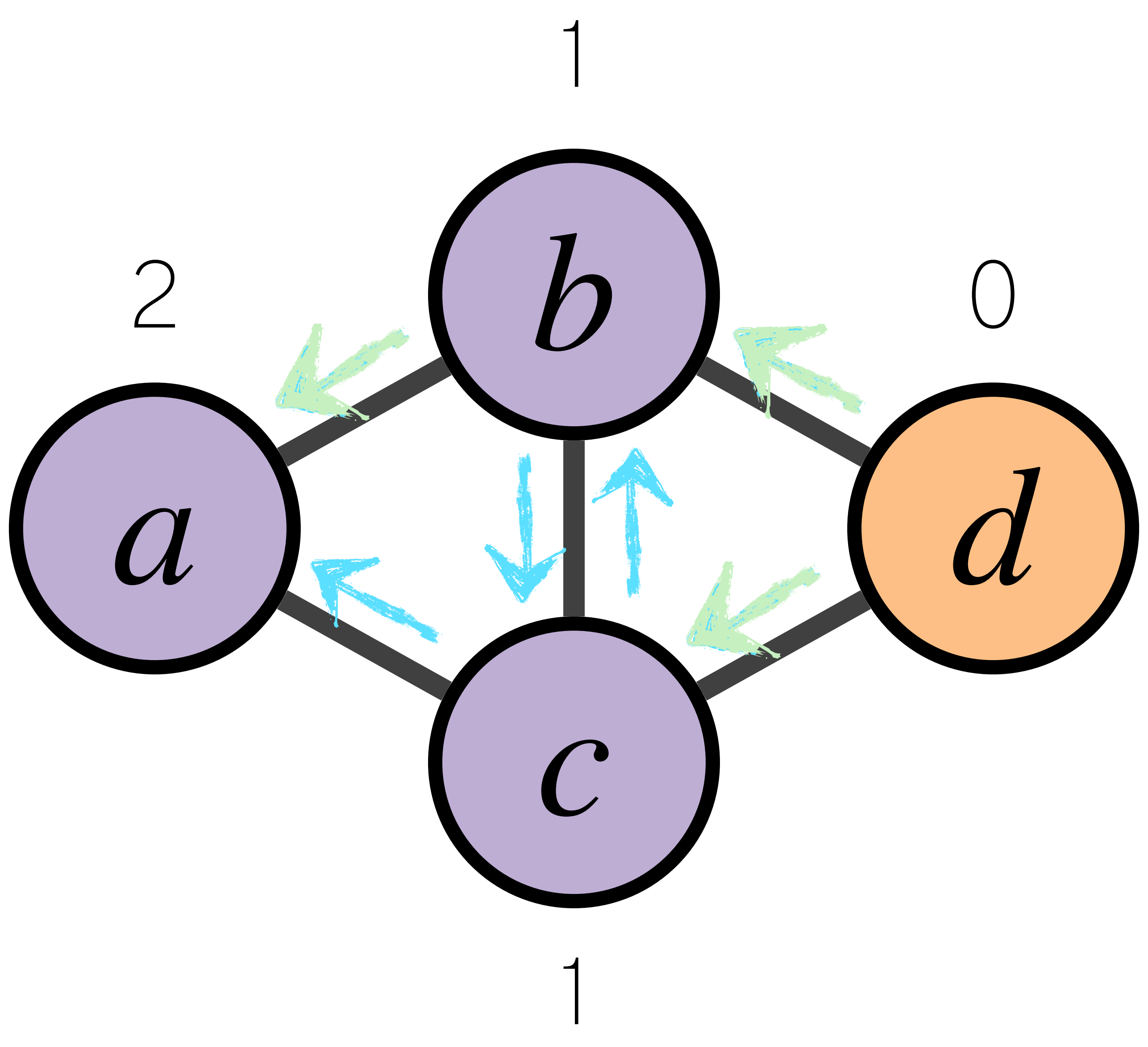
```
policy-statement CAAREN-IN {
    term participant {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-PARTICIPANT orlonger;
        }
        then next policy;
    }
    term segp {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-SEGP orlonger;
        }
        then {
            community add SEGP;
            next policy;
        }
    }
    term sponsored {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-SPONSORED orlonger;
        }
        then {
            community add SPONSORED;
            next policy;
        }
    }
    term reject-unicast {
        then reject;
    }
}
```

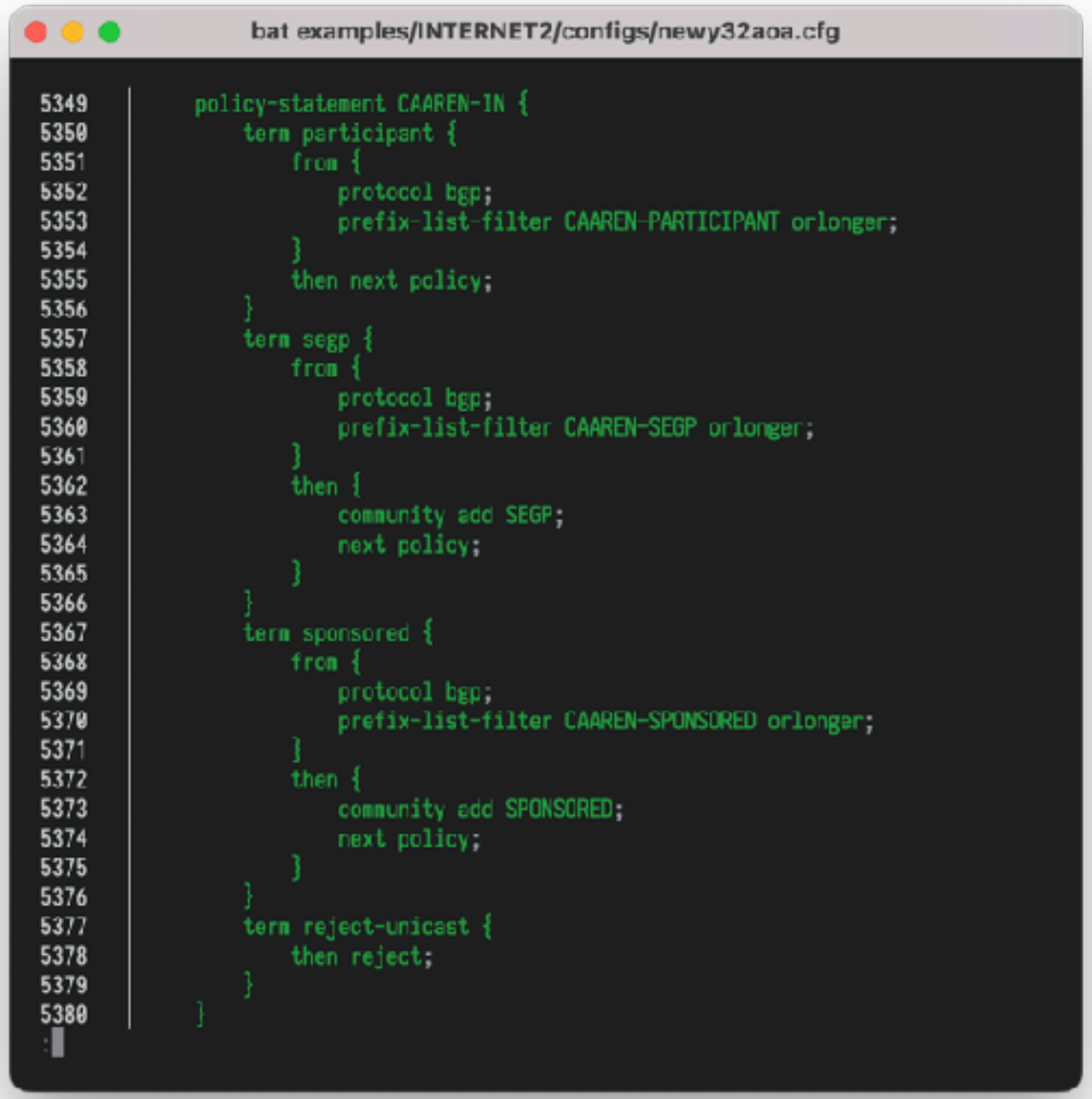
Distributed, written in vendor-specific, low-level language

What is the Control Plane?

Goal: determine routes to use to forward traffic

- Send initial route announcements
- Receive announcements, process according to **configs**
- Select best announcement
- Broadcast selected route to neighbors
- Converge to a **stable state**




bat examples/INTERNET2/configs/newy32aoa.cfg

```
policy-statement CAAREN-IN {
    term participant {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-PARTICIPANT orlonger;
        }
        then next policy;
    }
    term segp {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-SEGP orlonger;
        }
        then {
            community add SEGP;
            next policy;
        }
    }
    term sponsored {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-SPONSORED orlonger;
        }
        then {
            community add SPONSORED;
            next policy;
        }
    }
    term reject-unicast {
        then reject;
    }
}
```

Distributed, written in vendor-specific, low-level language

What is the Control Plane?

Goal: determine routes to use to forward traffic

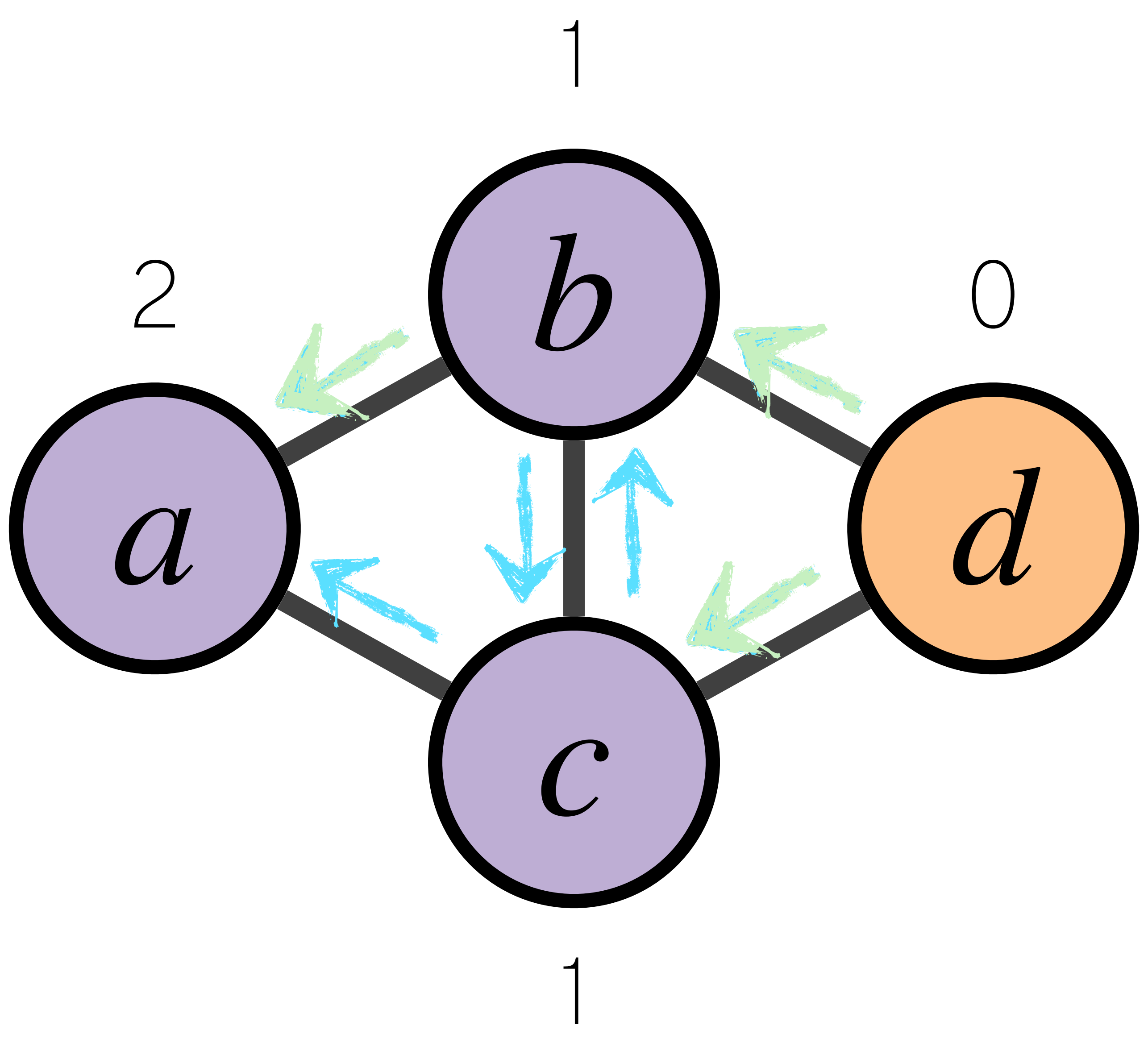
Send initial route announcements

Receive announcements, process according to **configs**

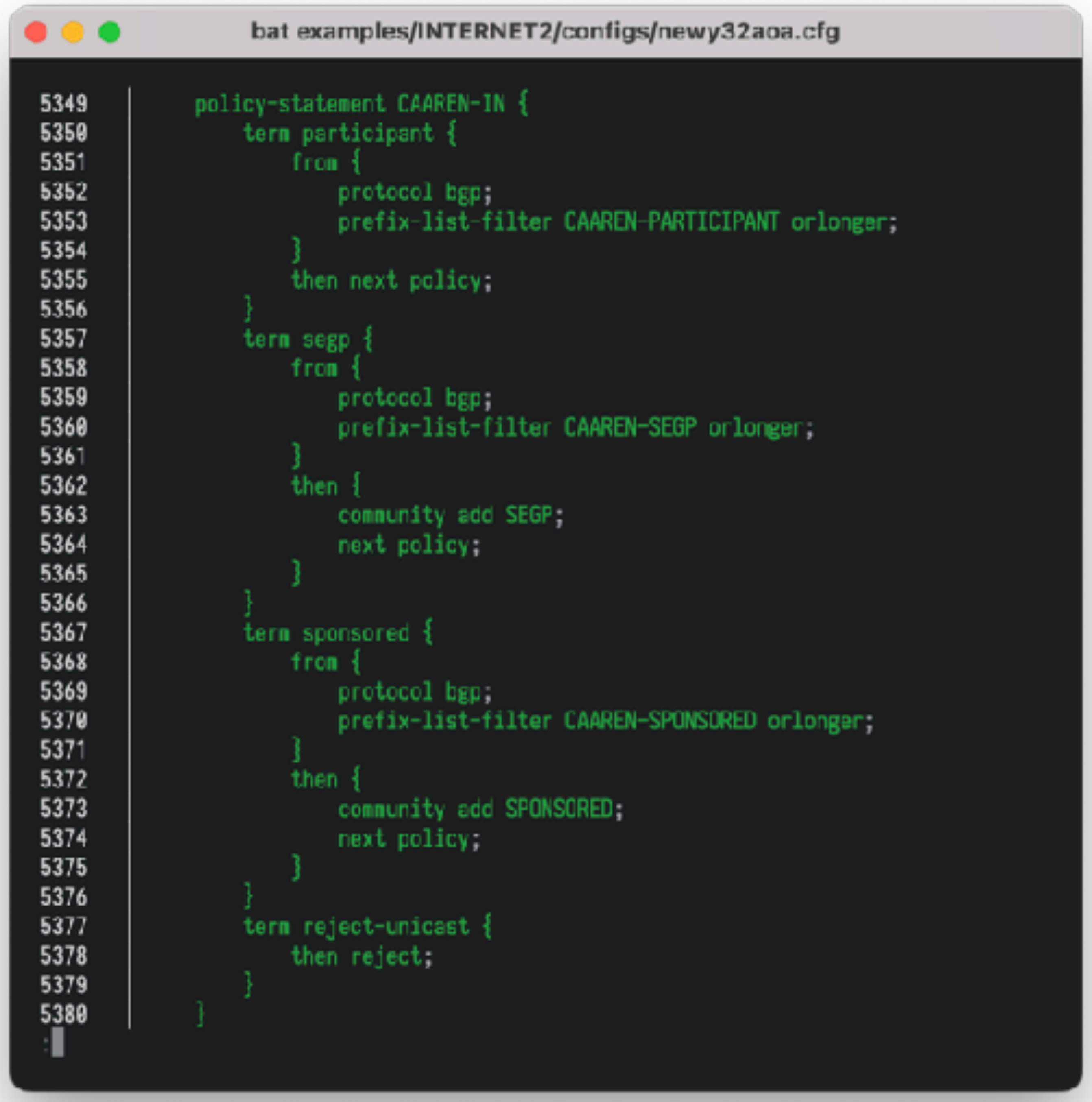
Select best announcement

Broadcast selected route to neighbors

Converge to a **stable state**

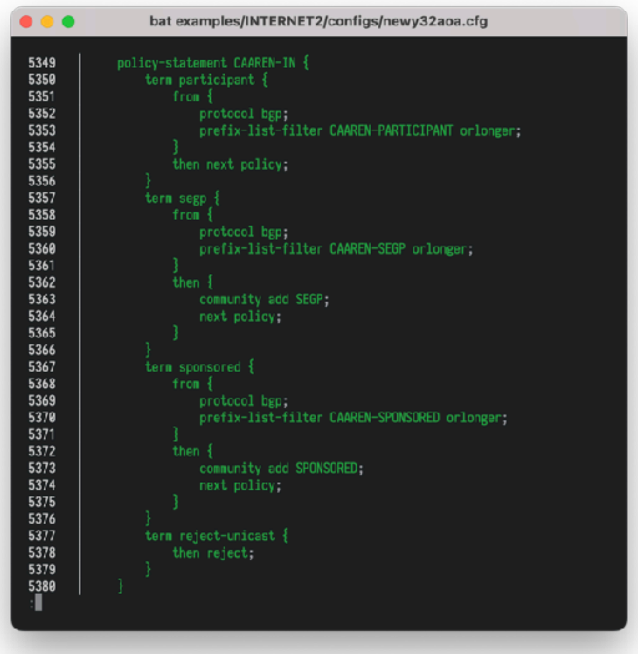


Policies for security, traffic engineering, fault tolerance, cost, *etc.*



Distributed, written in vendor-specific, low-level language

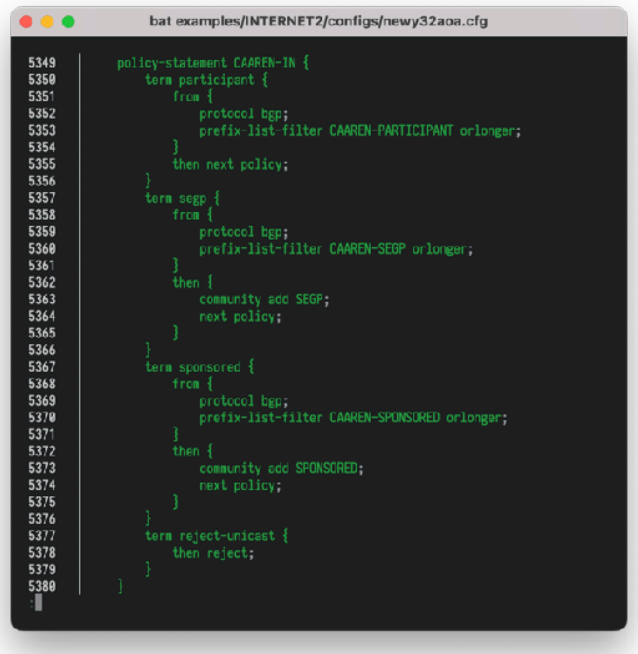
How Do We Verify Control Planes?



Network configuration files

Policies for security, traffic engineering, fault tolerance, cost, *etc.*

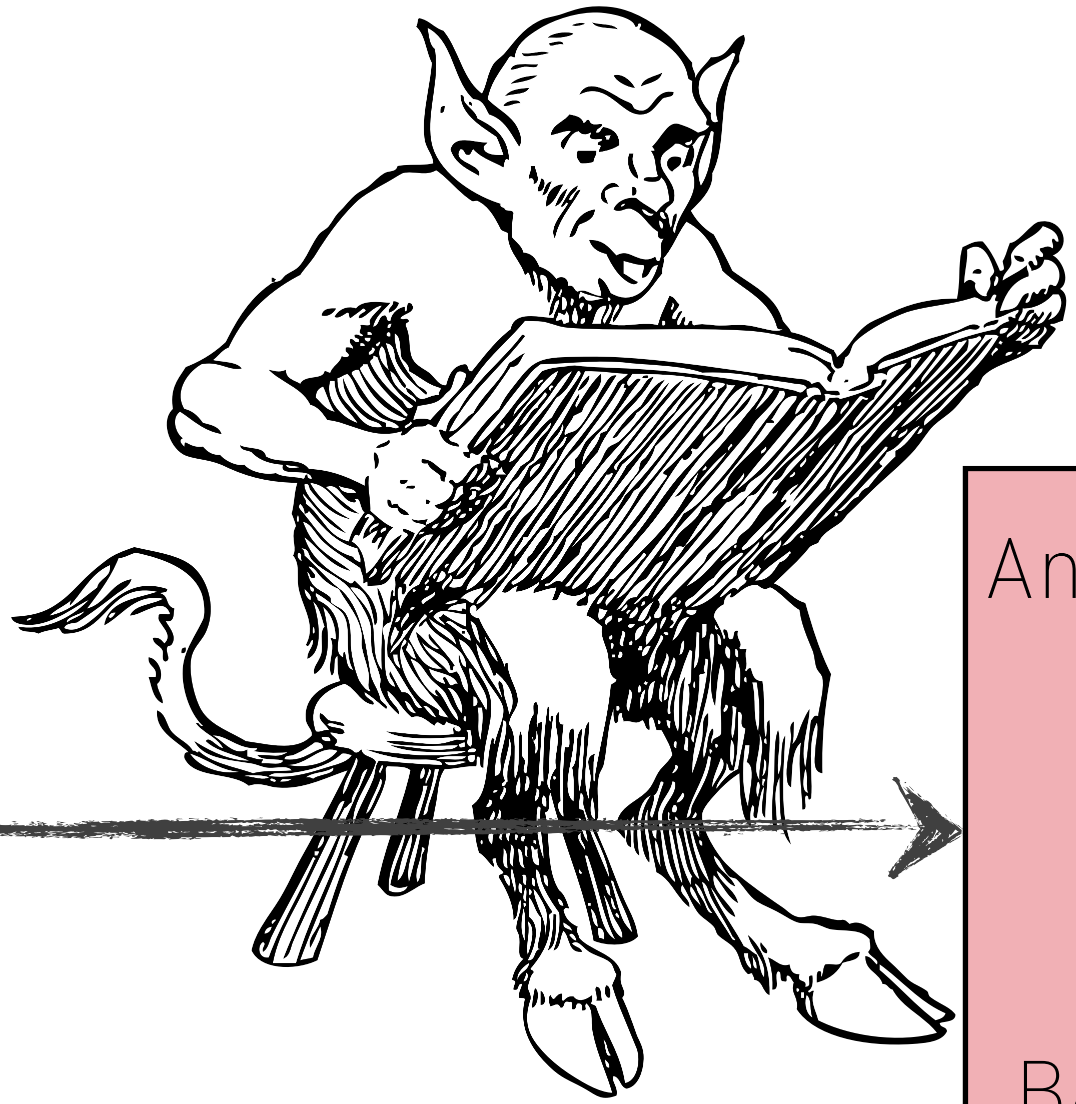
How Do We Verify Control Planes?




```
policy-statement CAAREN-IN {
    term participant {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-PARTICIPANT orlonger;
        }
        then next policy;
    }
    term segp {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-SEGP orlonger;
        }
        then {
            community add SEGP;
            next policy;
        }
    }
    term sponsored {
        from {
            protocol bgp;
            prefix-list-filter CAAREN-SPONSORED orlonger;
        }
        then {
            community add SPONSORED;
            next policy;
        }
    }
    term reject-unicast {
        then reject;
    }
}
```

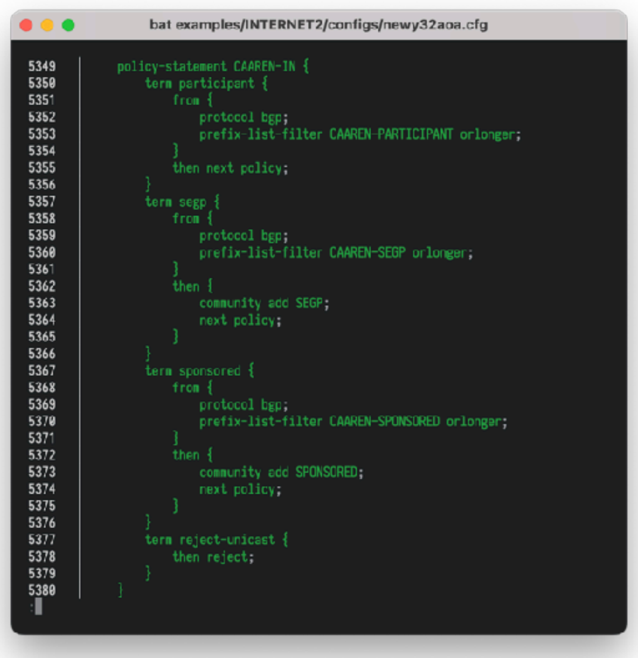
Network configuration files

Policies for security, traffic engineering, fault tolerance, cost, *etc.*



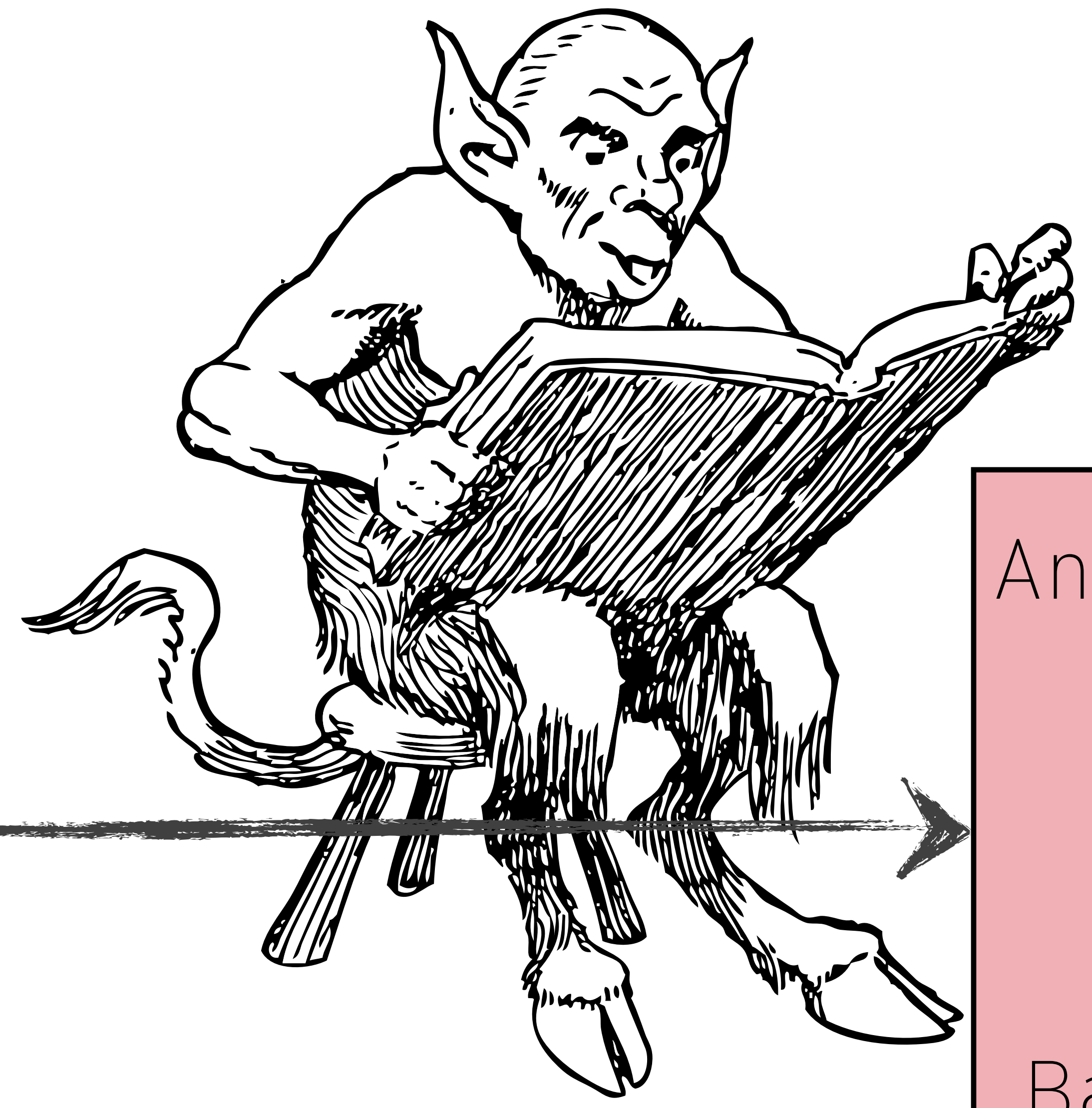
Analyze <u>all configurations together</u> to find property violations using a control plane verifier (e.g., Batfish, ARC, Minesweeper, Bagpipe, Tiramisu, Plankton, Hoyan)

How Do We Verify Control Planes?



Network configuration files

Policies for security, traffic engineering, fault tolerance, cost, *etc.*



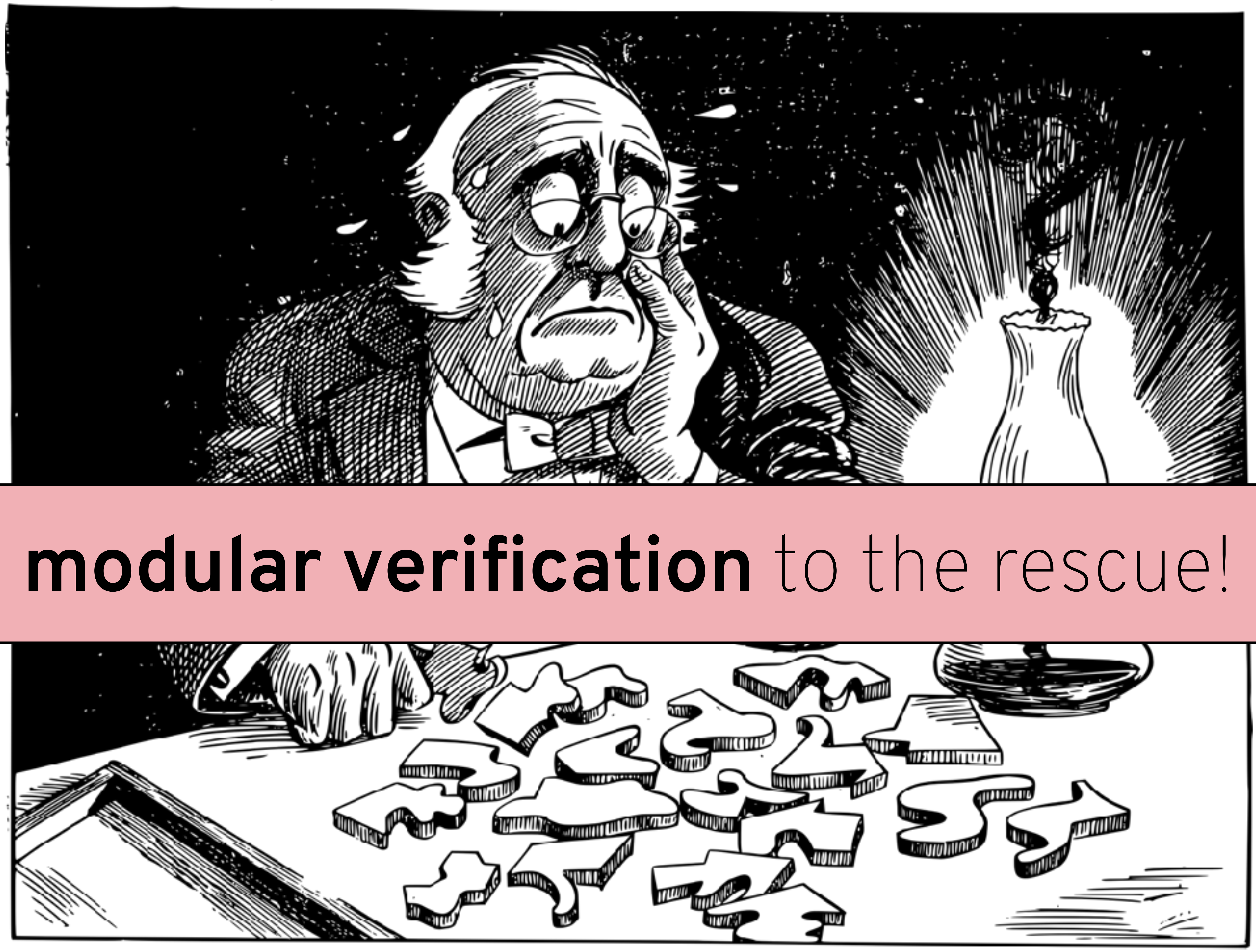
Analyze all configurations together to find property violations using a control plane verifier (e.g., Batfish, ARC, Minesweeper, Bagpipe, Tiramisu, Plankton, Hoyan)

Repeat when configurations change

Scaling Control Plane Verification

Many networks are **too big and too complex** to verify monolithically!

Scaling Control Plane Verification

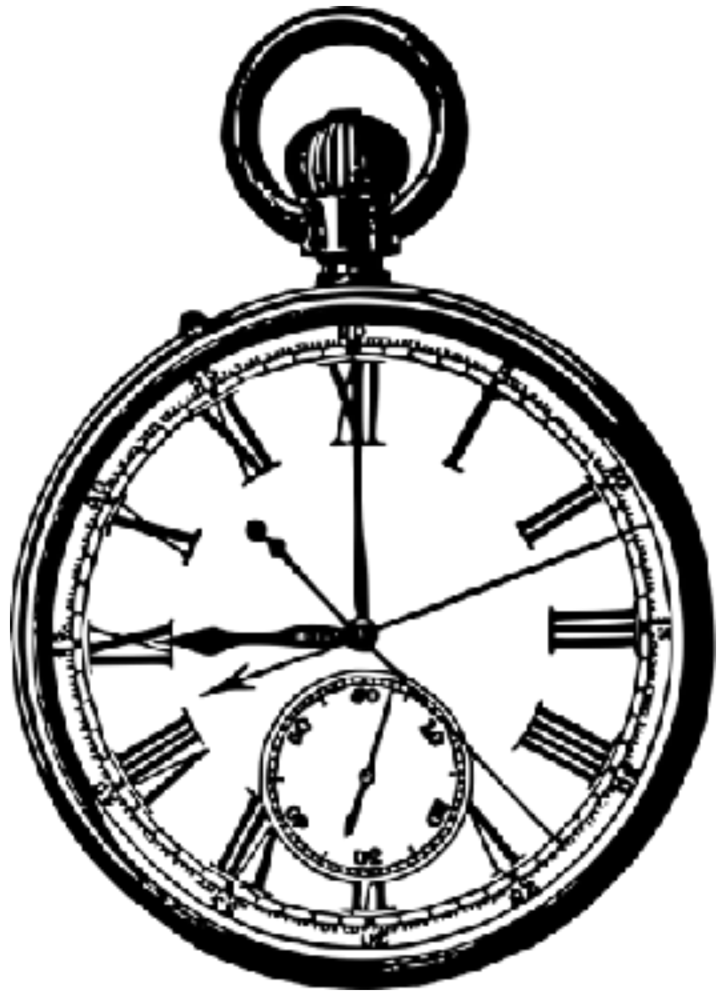


modular verification to the rescue!

Many networks are **too big and too complex** to verify monolithically!

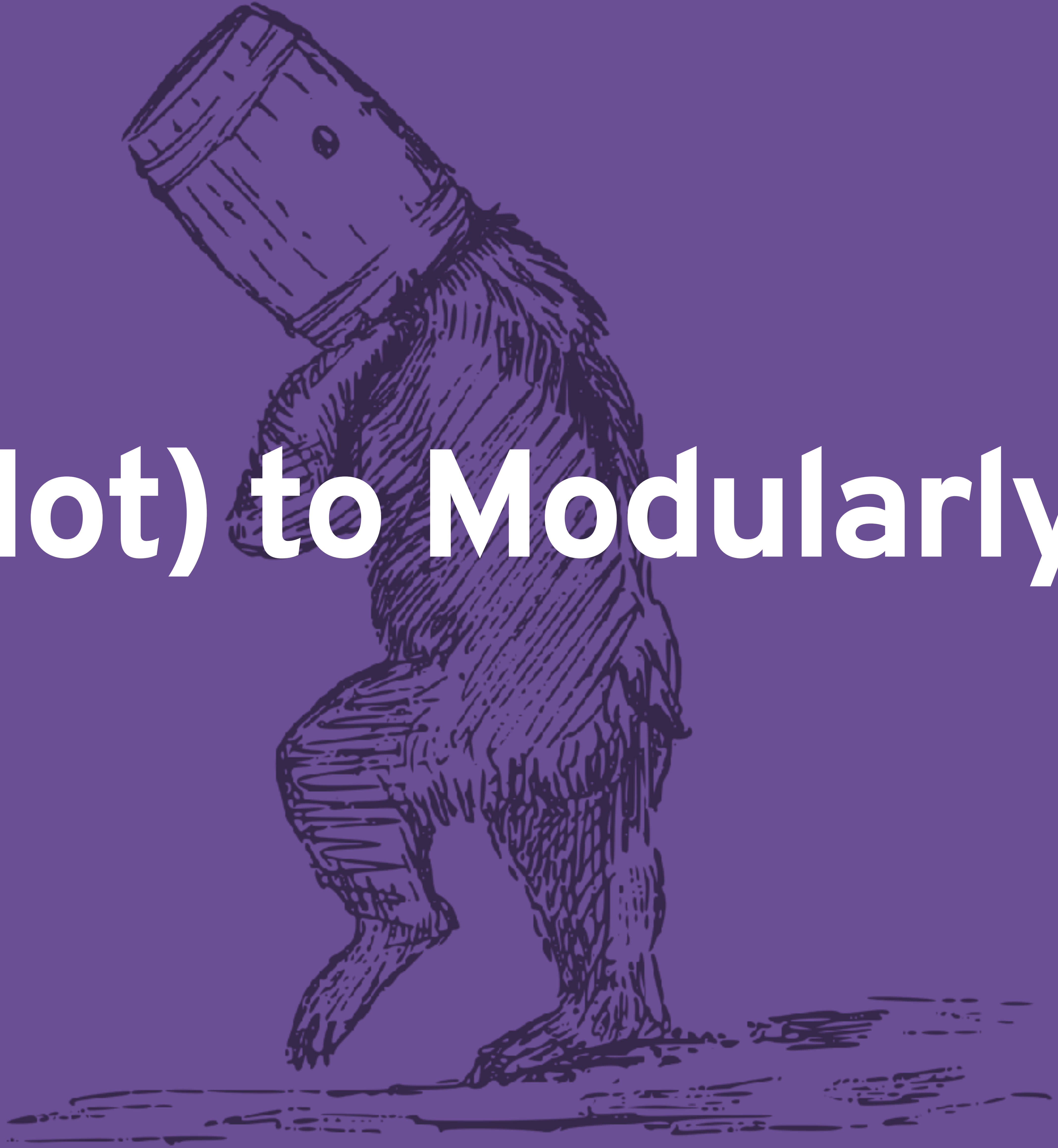
Our Contributions

demonstrate why naive stable states analysis is unsuitable for modular verification



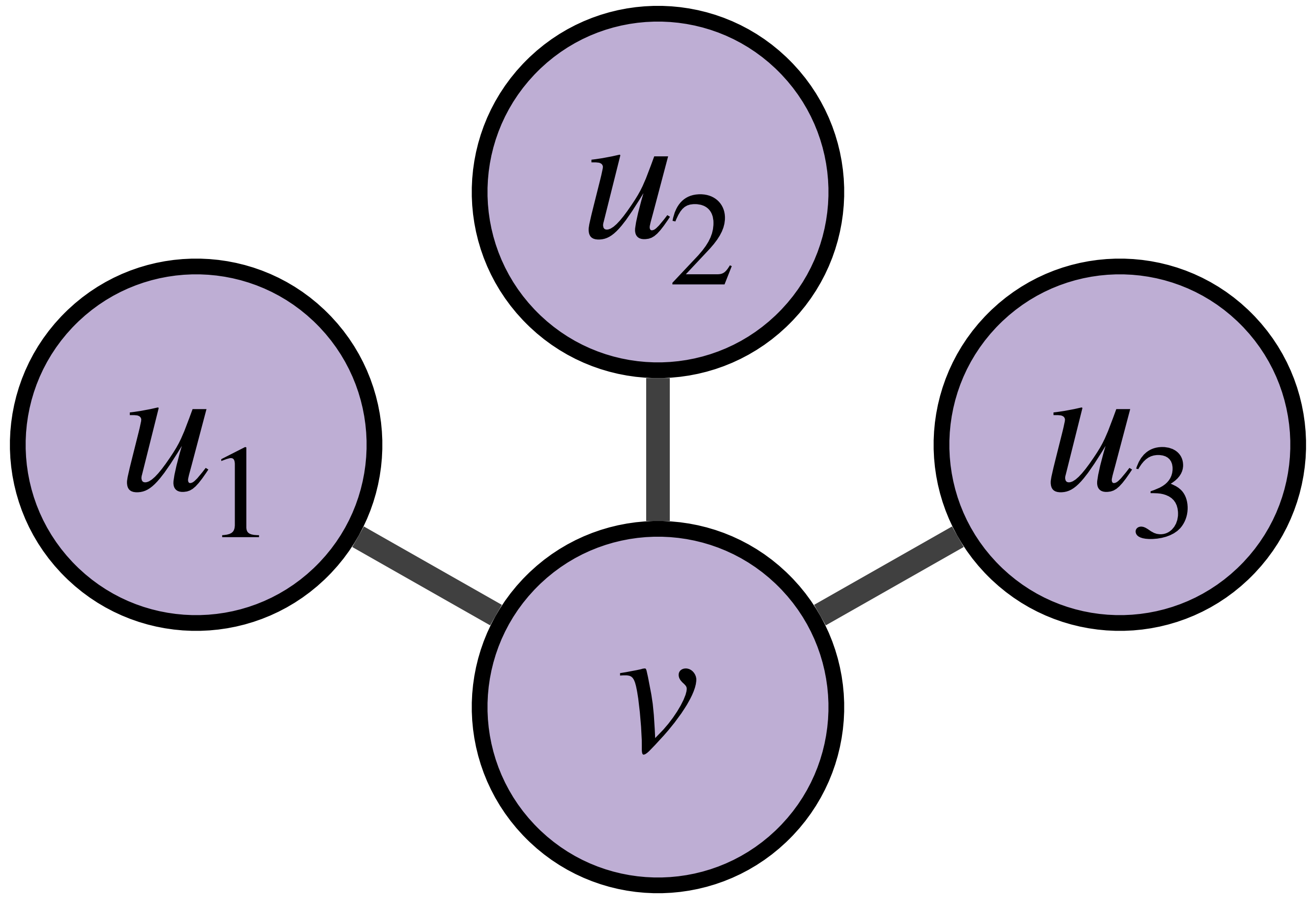
present time-based theory for modular control plane analysis, with SMT-based verification procedure

verify properties of 2000-node data centers and complex wide-area networks in seconds!



How (Not) to Modularly Verify

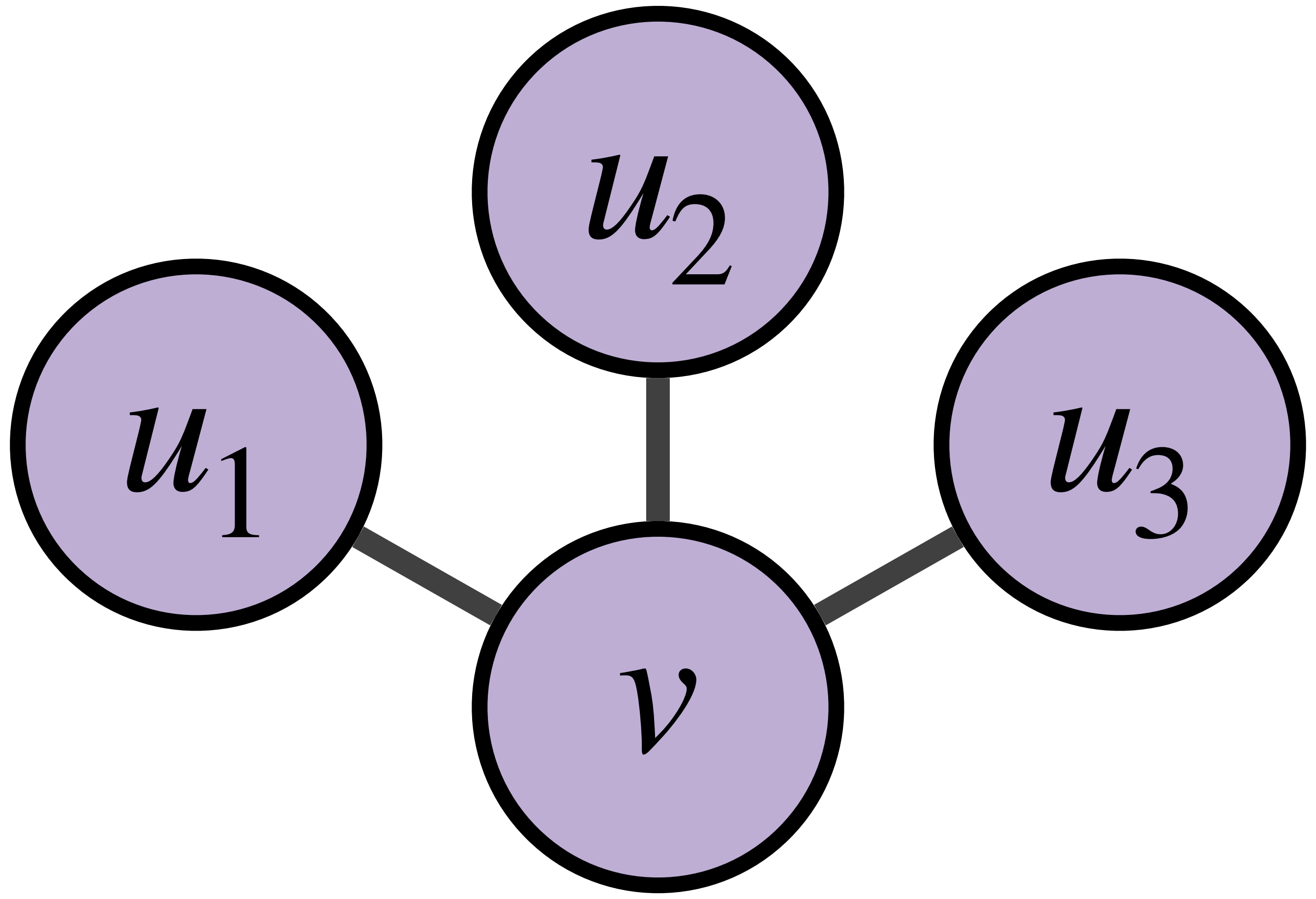
Modular Network Verification



Modular Network Verification

sound modular analysis:

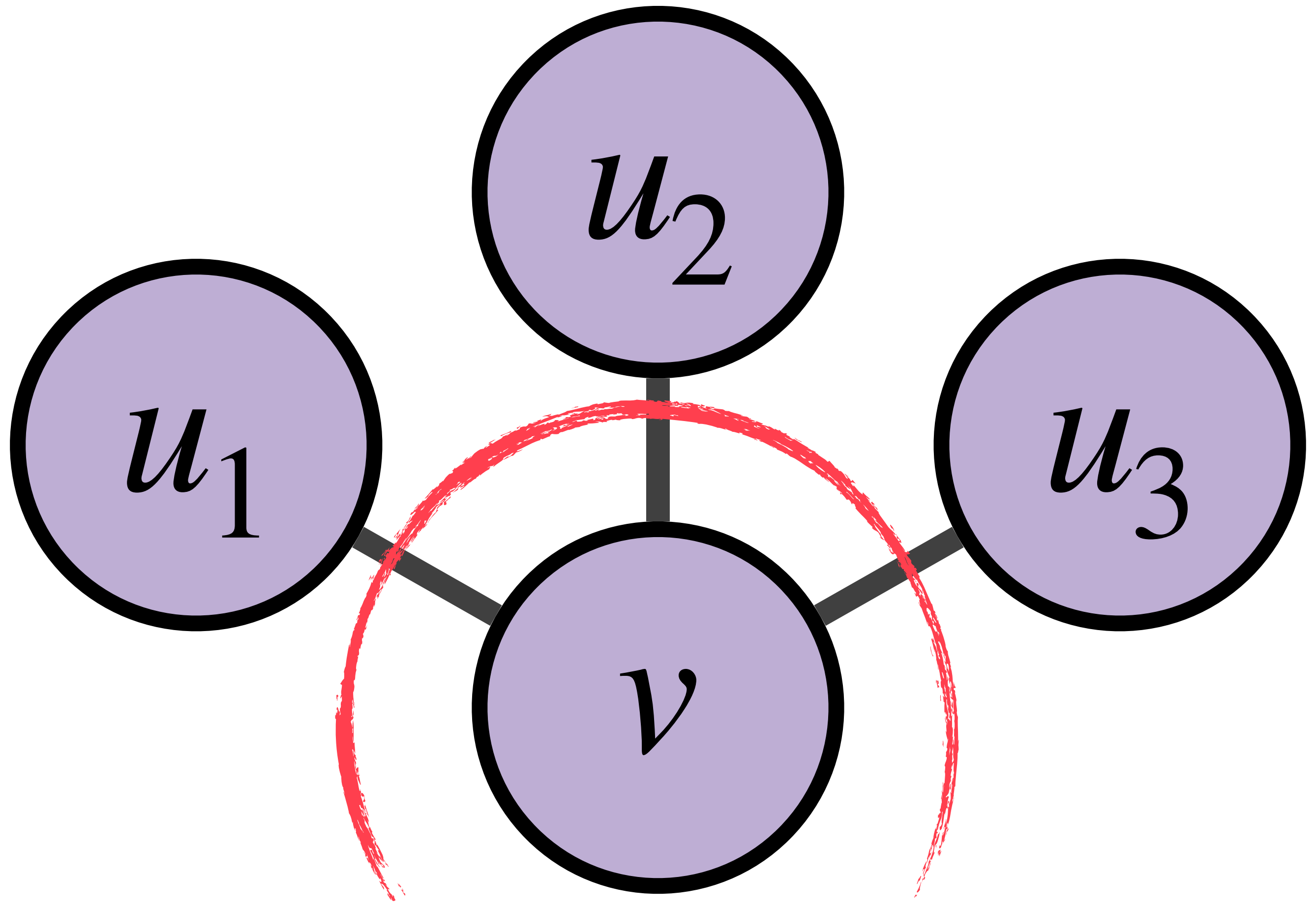
captures all monolithic routing behavior



Modular Network Verification

sound modular analysis:
captures all monolithic routing behavior

split the network up into **node-local components** to verify independently

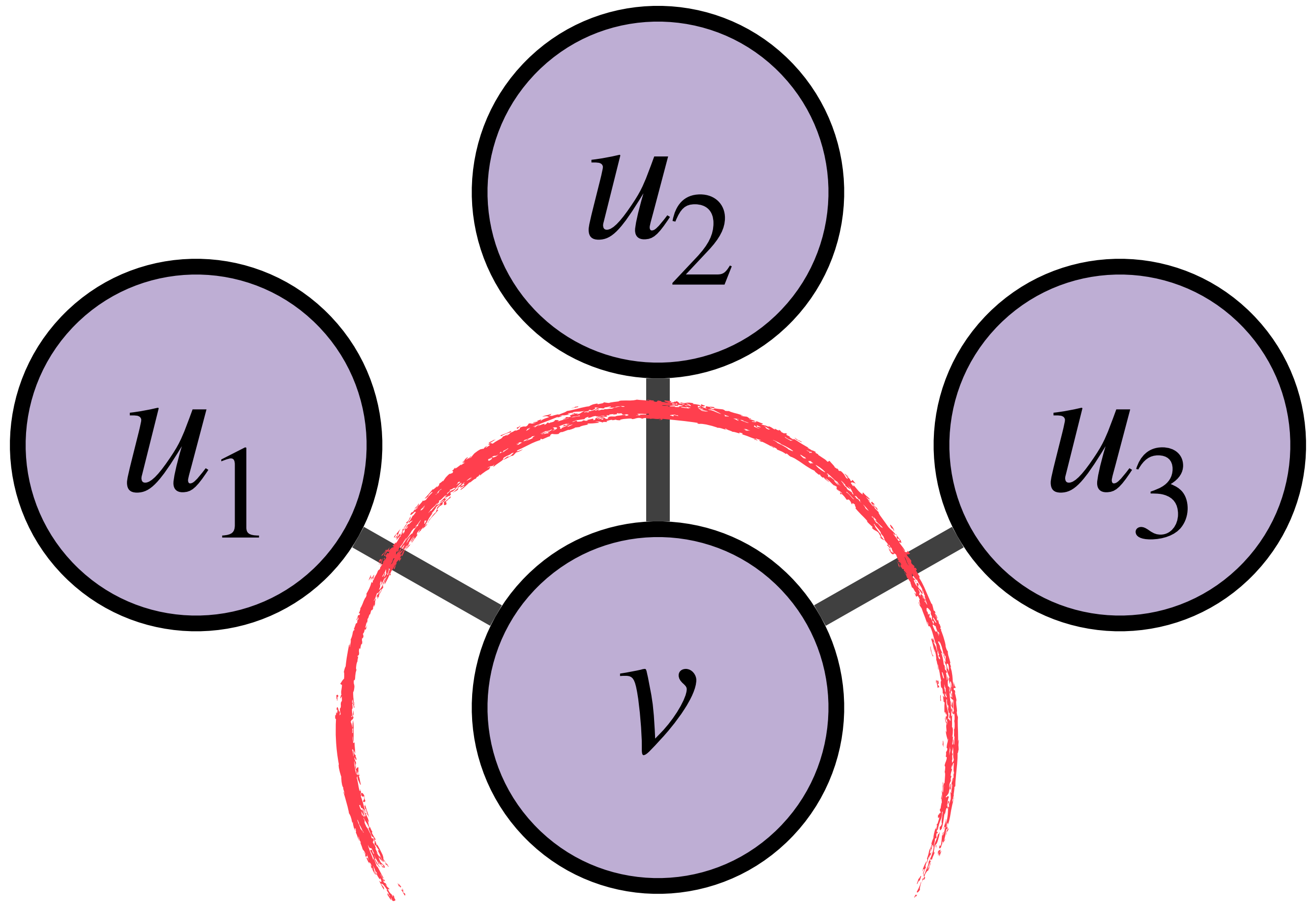


Modular Network Verification

sound modular analysis: captures all monolithic routing behavior

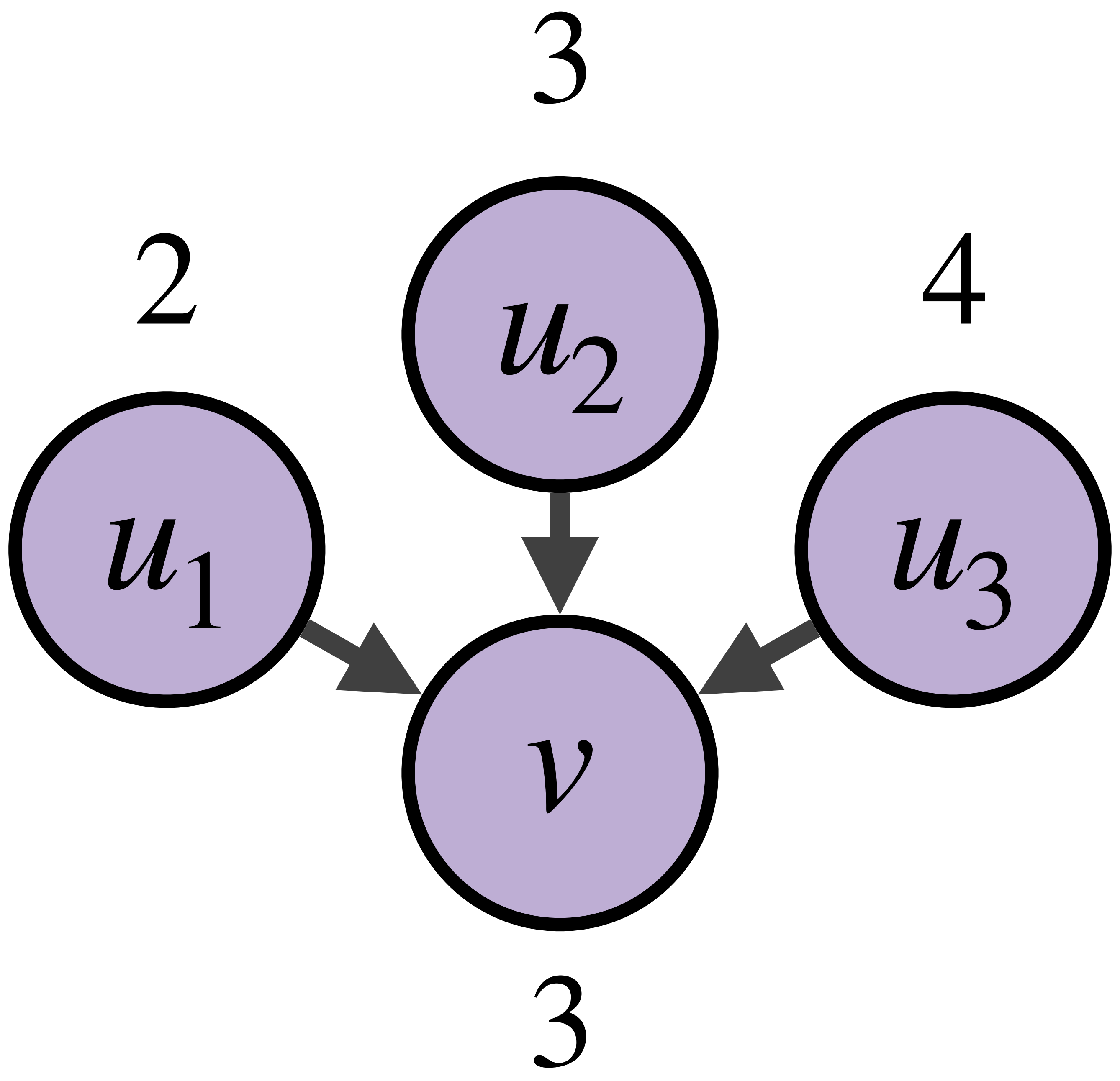
split the network up into **node-local components** to verify independently

represent cross-component dependencies using **interfaces**



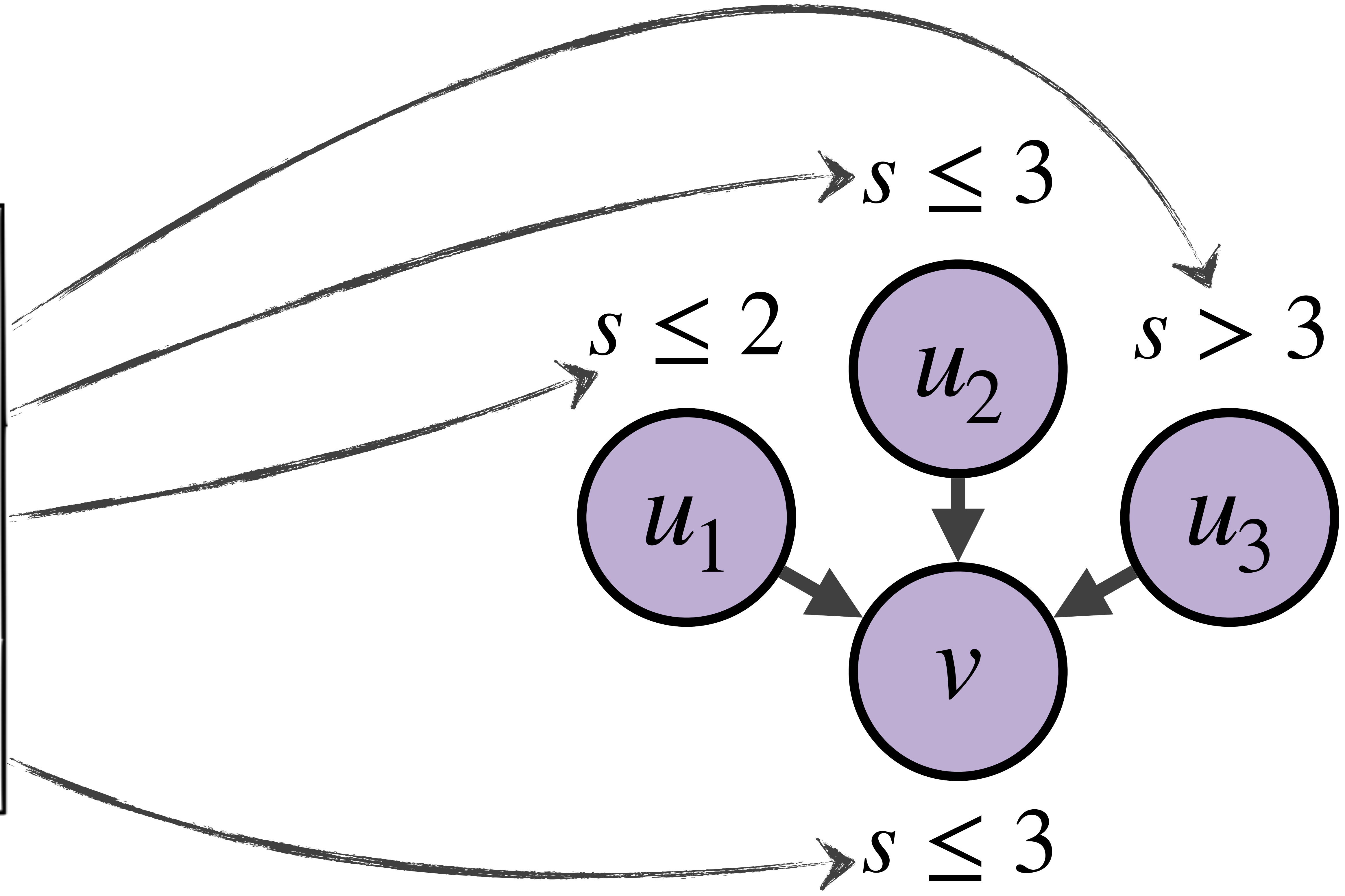
Interfaces

User



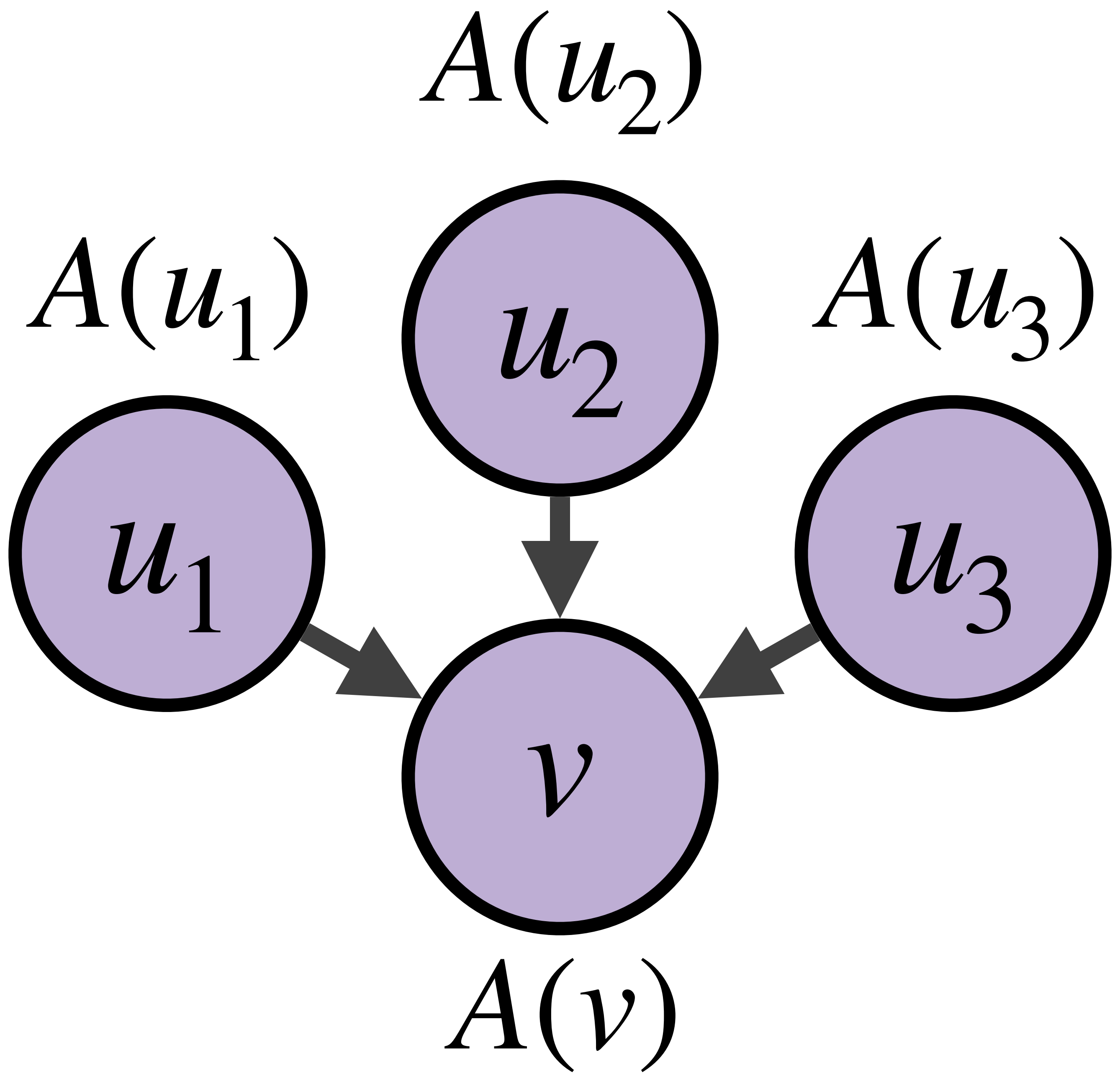
Interfaces

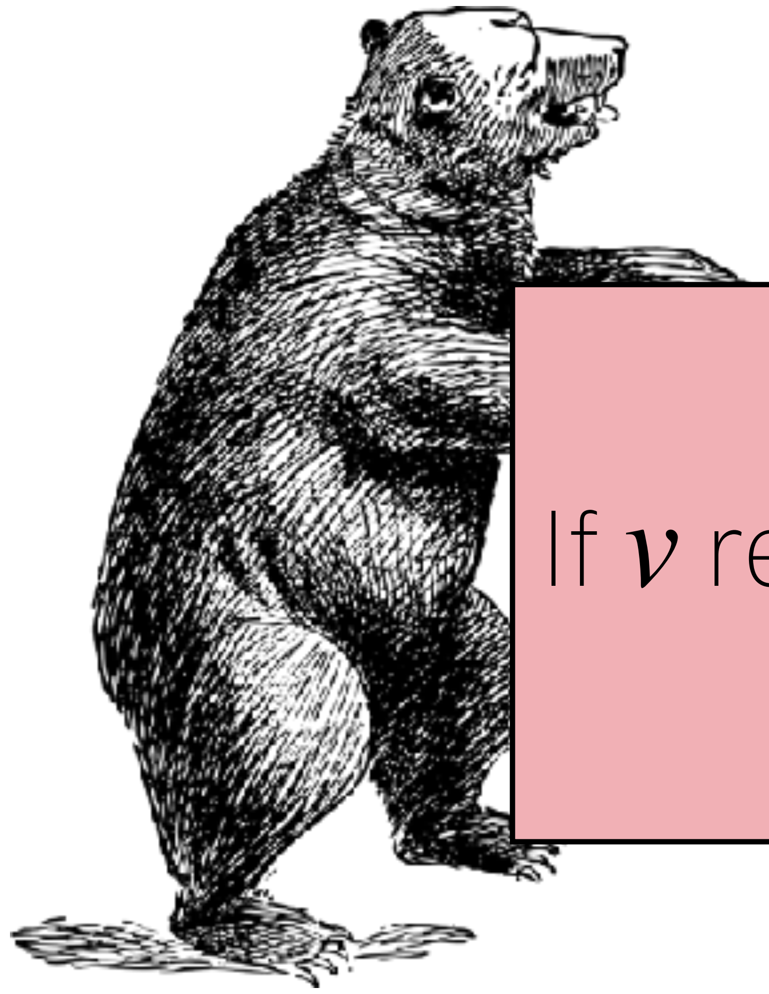
User



interface A over-approximates the converged states of the node v with a set of states $A(v)$

Bear's Modular Verification Procedure



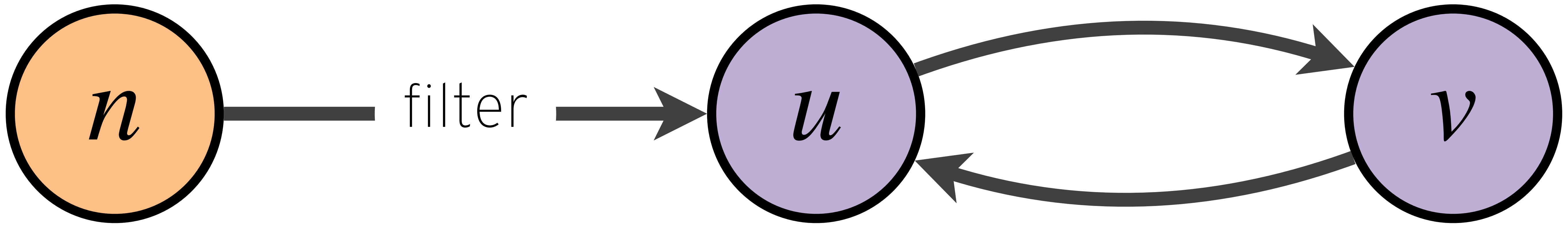


Verification condition:

If v receives any routes satisfying $A(u_1), A(u_2), \ldots, A(u_m)$,

does its selected route satisfy $A(v)$?

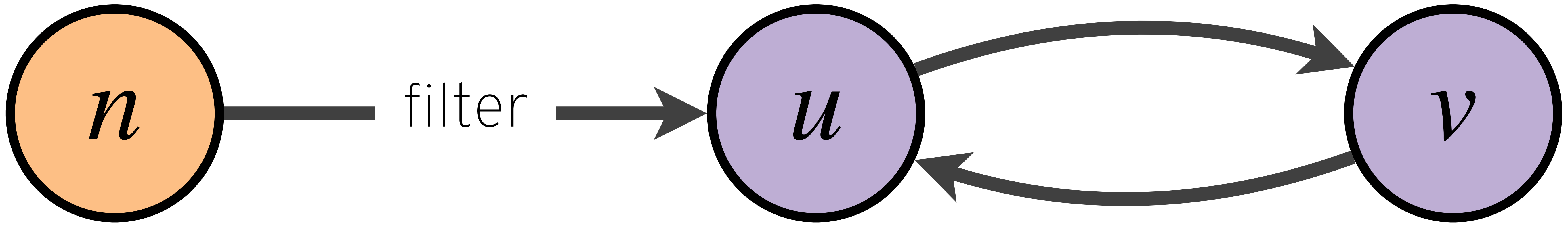
An Example Network



An Example Network

routes: ⟨**lp** : $\mathbb{N}$, **len** : $\mathbb{N}$⟩ or ∞

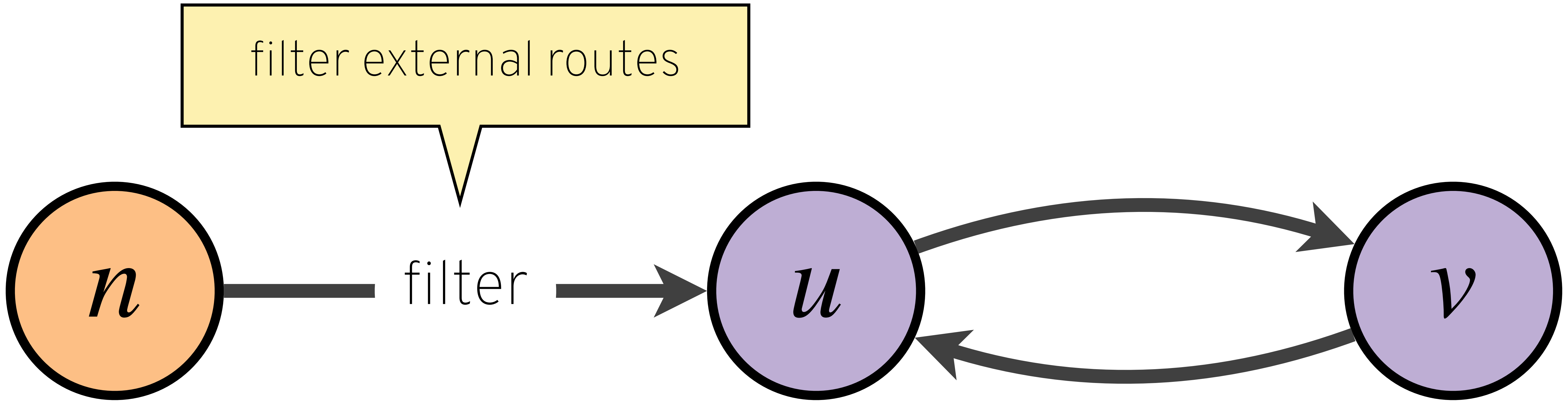
select by highest **lp**, lowest **len**



An Example Network

routes: $\langle \mathbf{lp} : \mathbb{N}, \mathbf{len} : \mathbb{N} \rangle$ or ∞

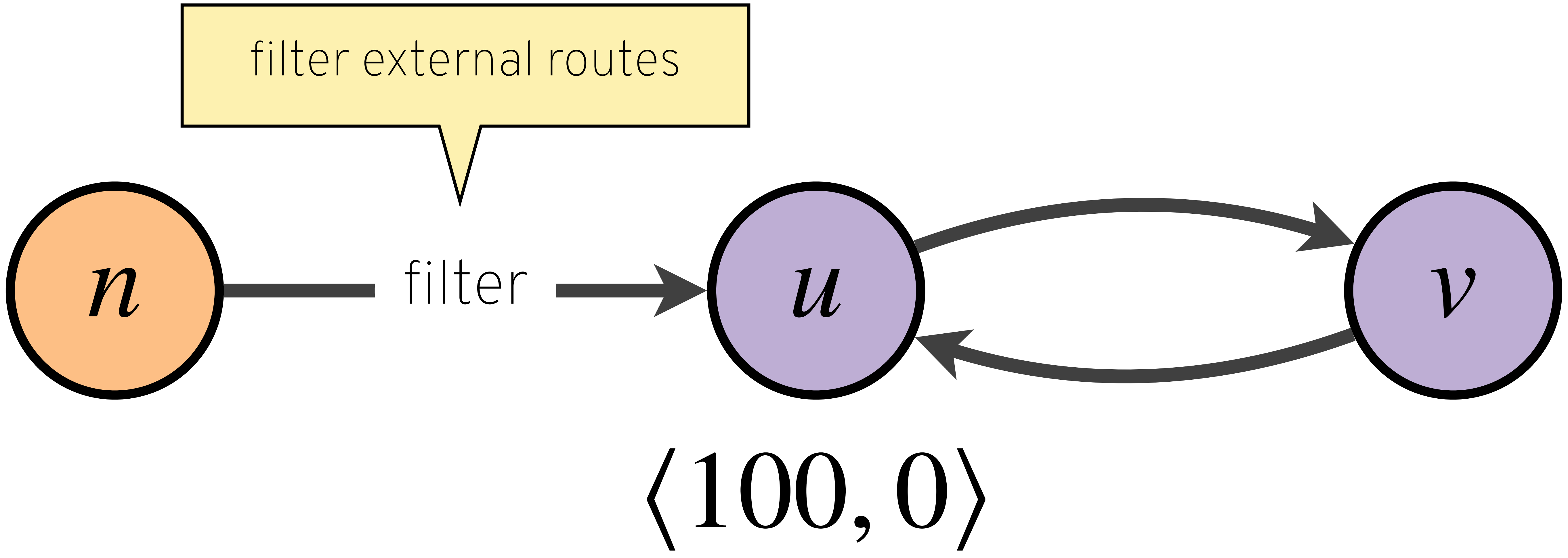
select by highest **lp**, lowest **len**



An Example Network

routes: $\langle \textbf{lp} : \mathbb{N}, \textbf{len} : \mathbb{N} \rangle$ or ∞

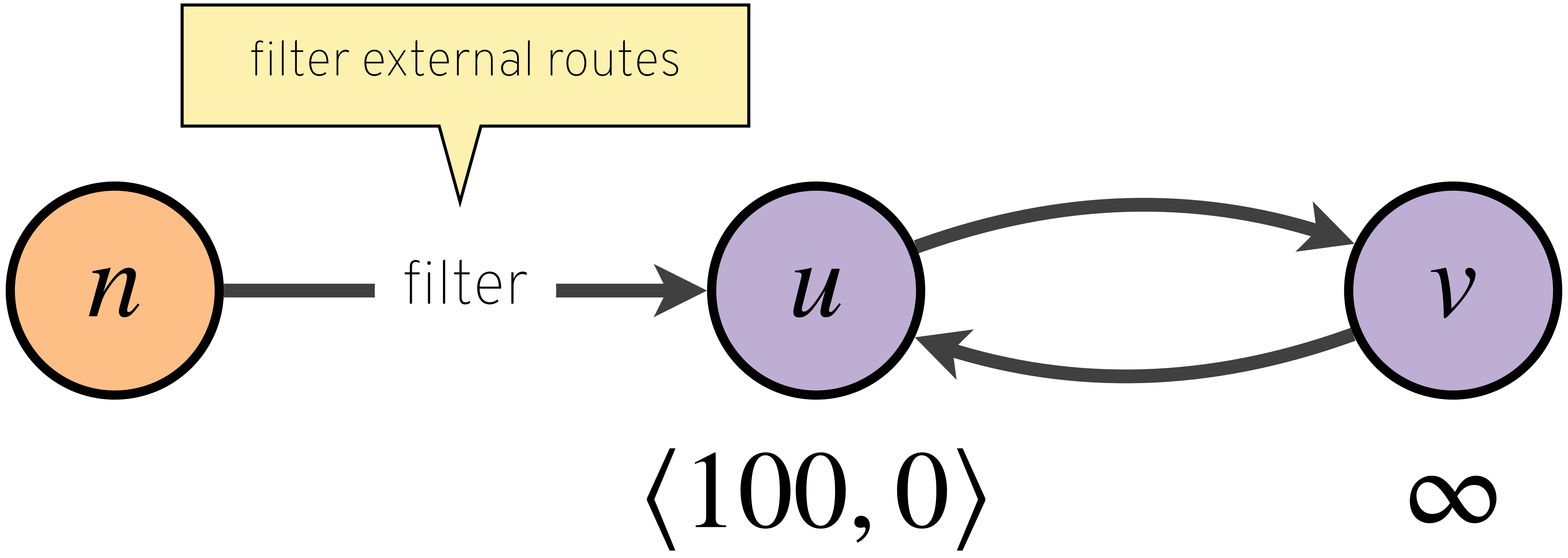
select by highest **lp**, lowest **len**



An Example Network

routes: $\langle \mathbf{lp} : \mathbb{N}, \mathbf{len} : \mathbb{N} \rangle$ or ∞

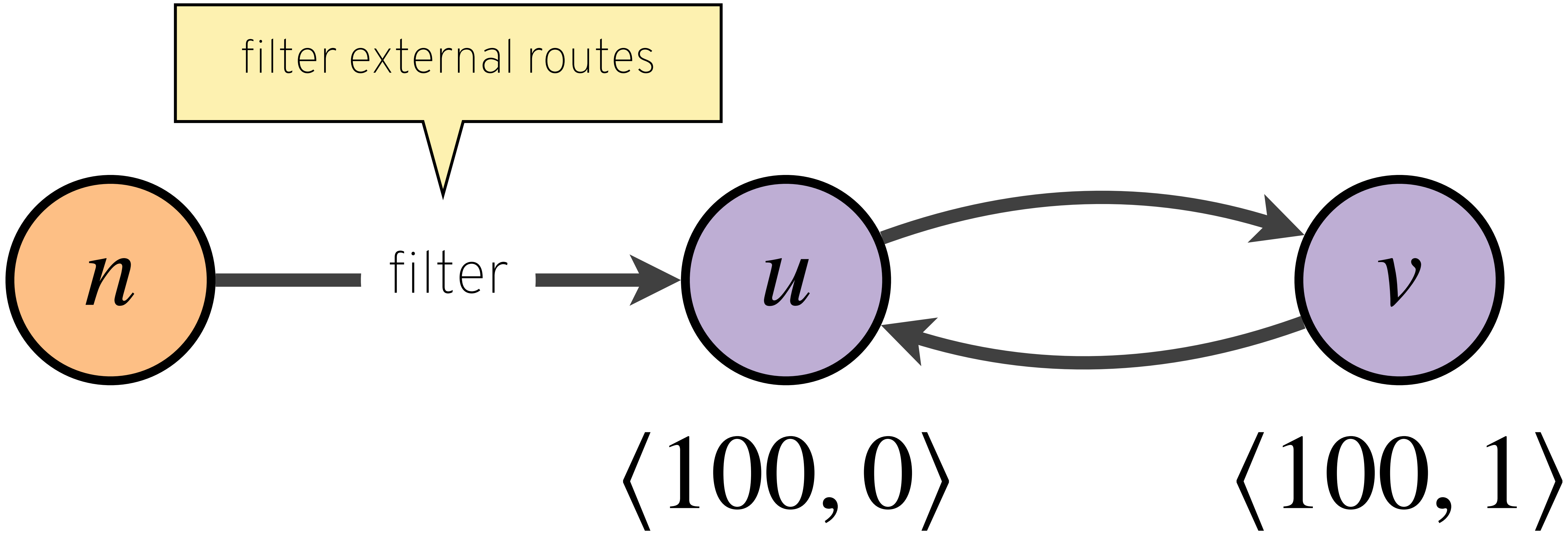
select by highest **lp**, lowest **len**



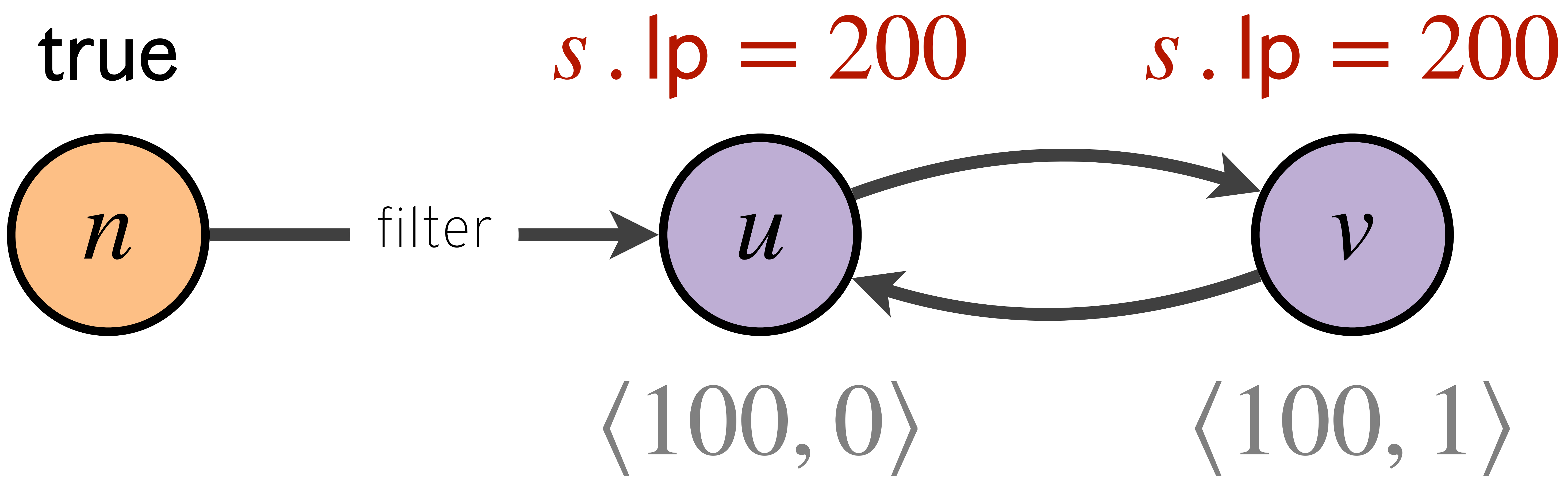
An Example Network

routes: $\langle \mathbf{lp} : \mathbb{N}, \mathbf{len} : \mathbb{N} \rangle$ or ∞

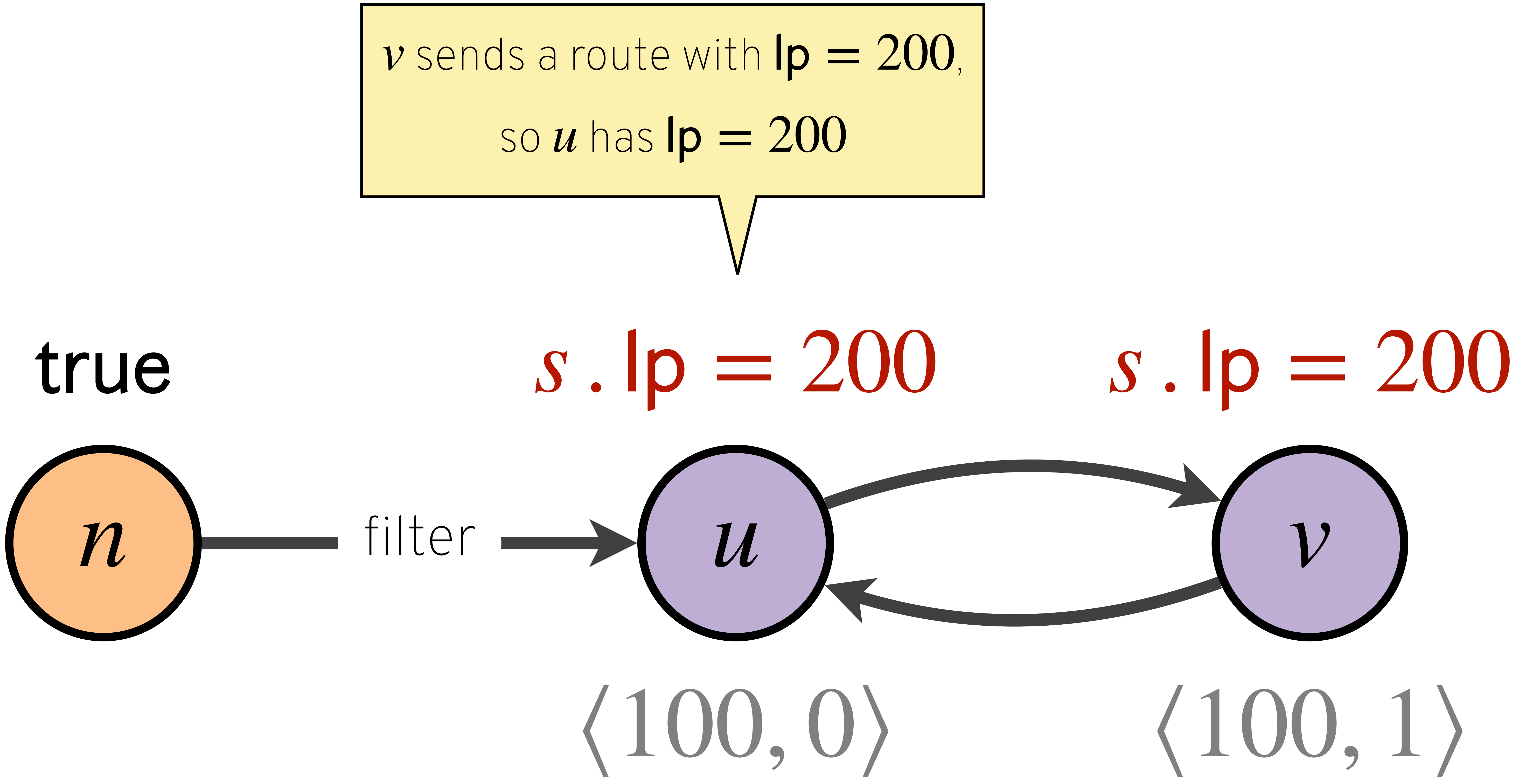
select by highest **lp**, lowest **len**



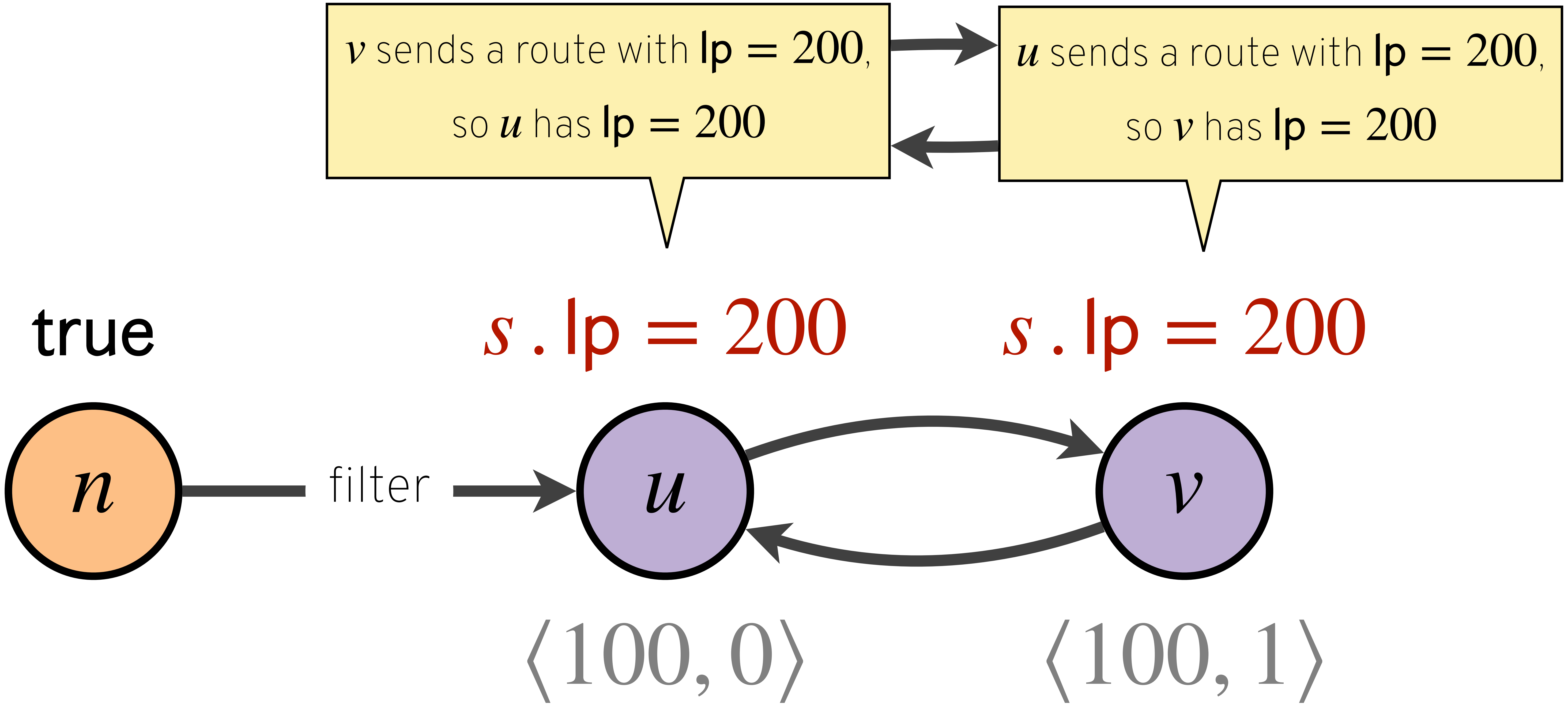
Execution Interference



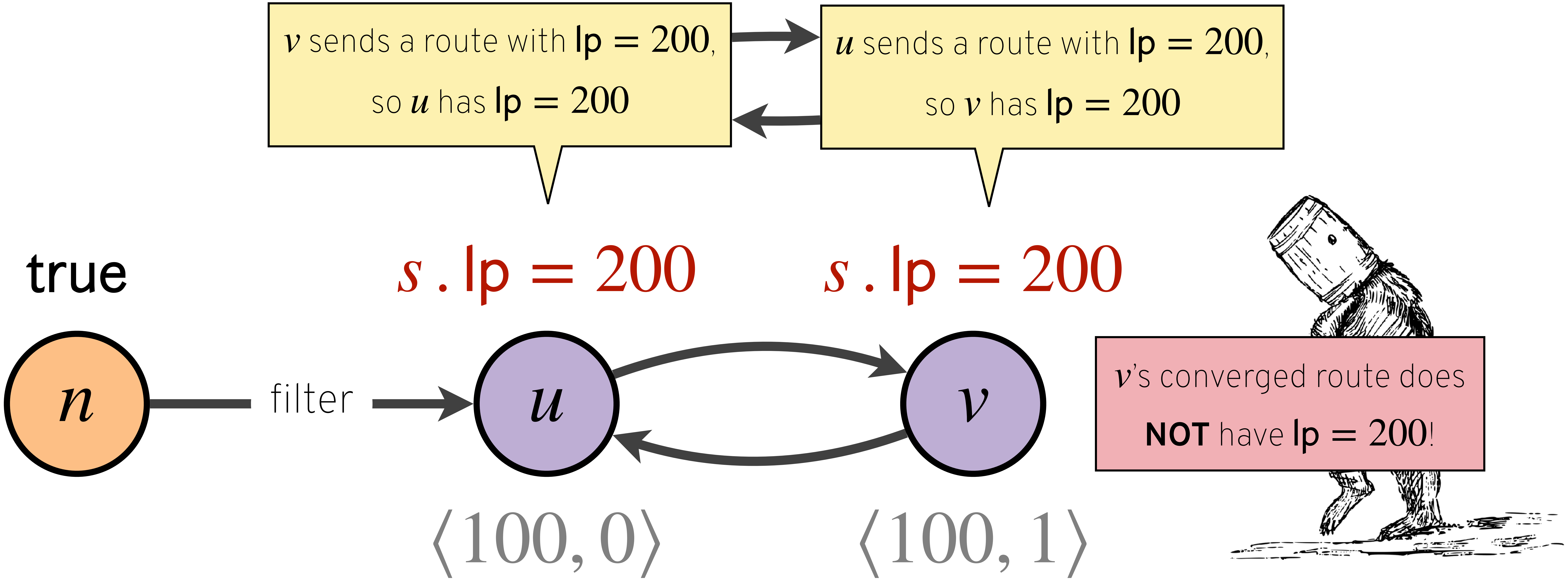
Execution Interference



Execution Interference



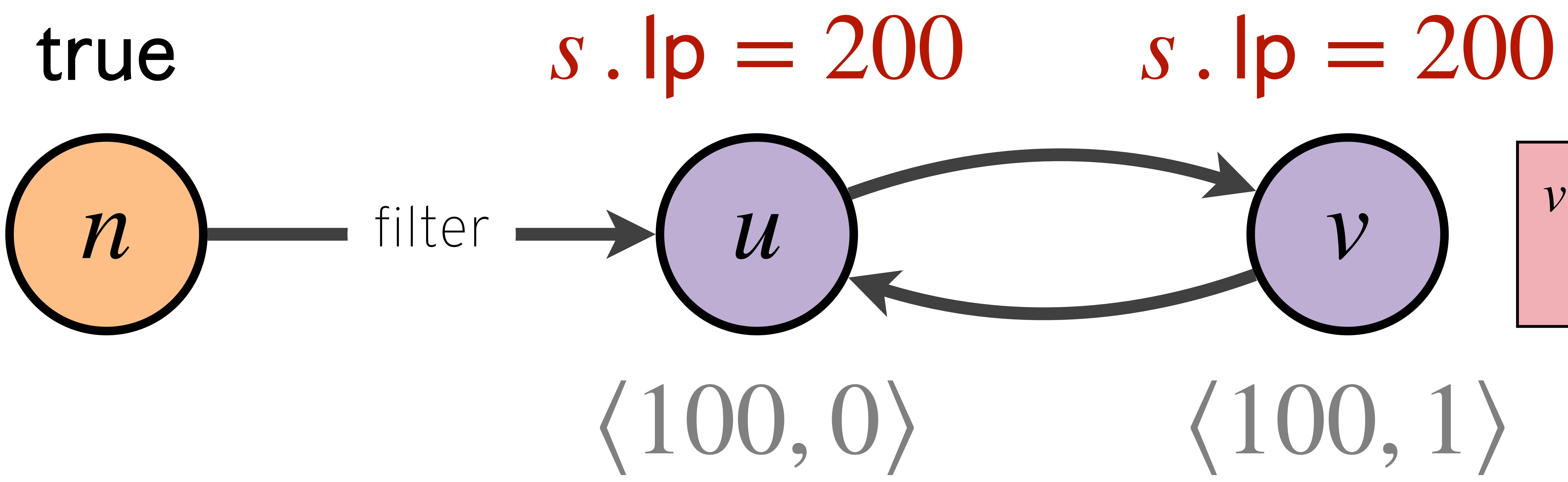
Execution Interference



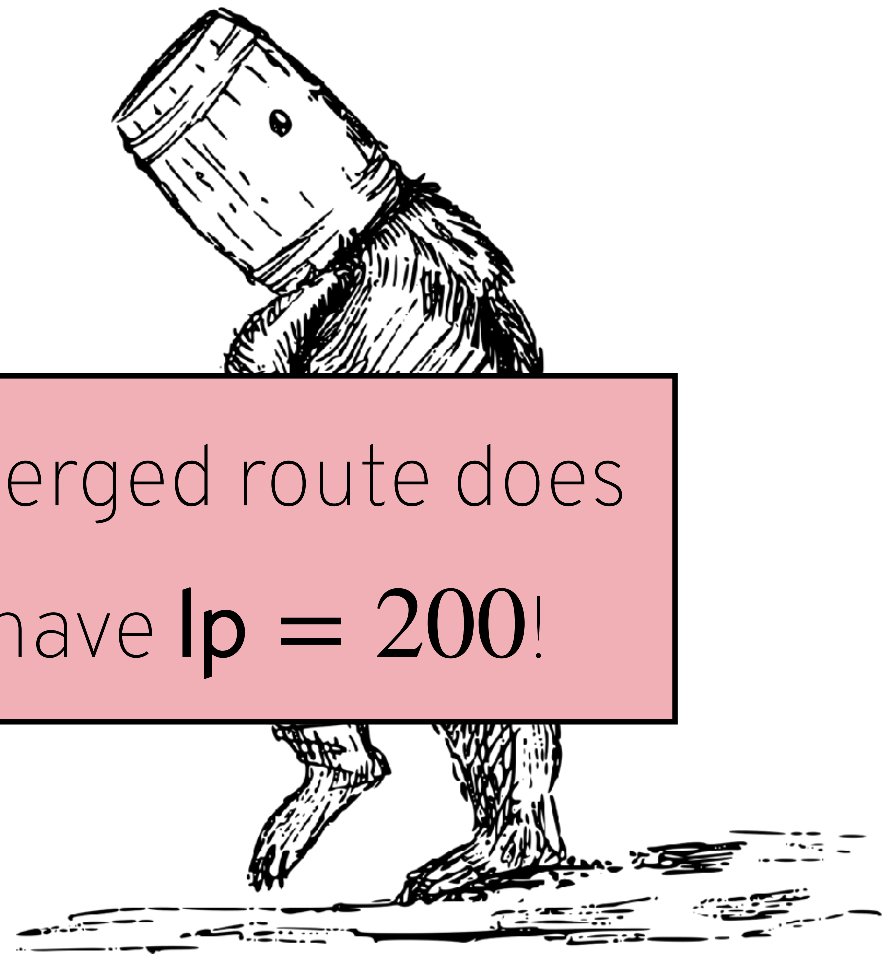
Execution Interference

v sends a route with $\mathsf{lp} = 200$, so u has $\mathsf{lp} = 200$

u sends a route with $\mathsf{lp} = 200$, so v has $\mathsf{lp} = 200$

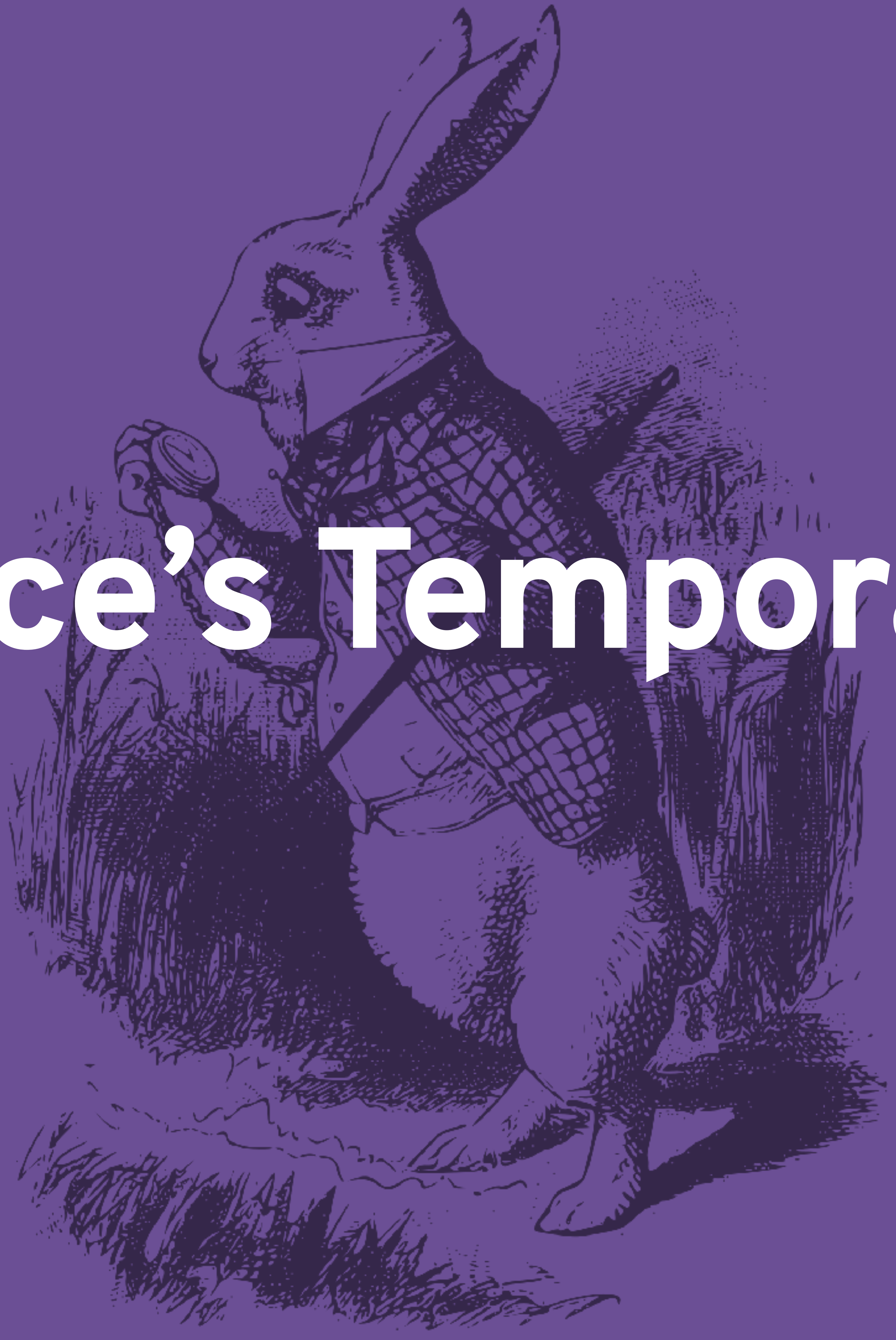


v's converged route does **NOT** have $\mathsf{lp} = 200$!

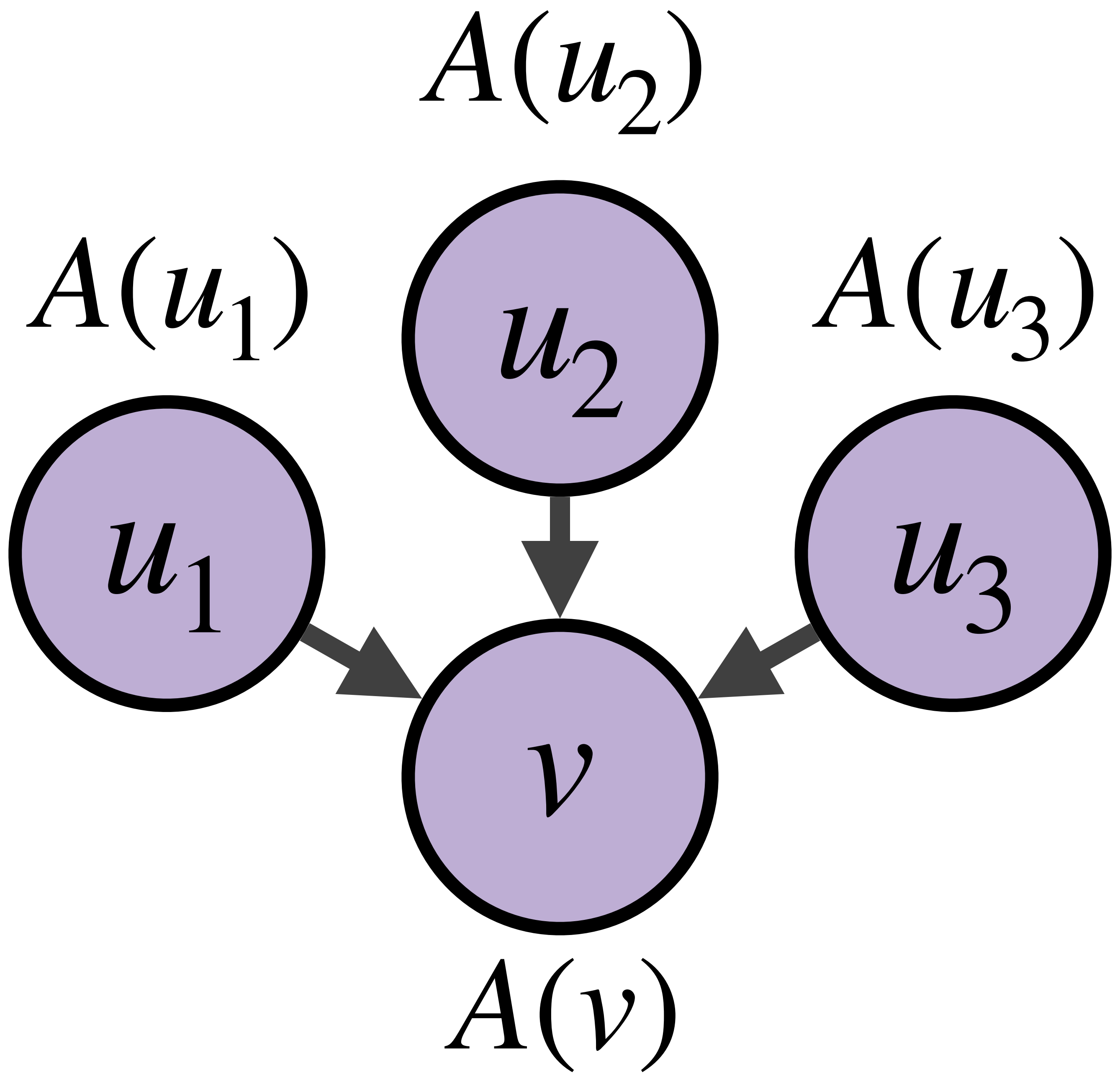


interfaces are **unsound**: exclude the legitimate converged routes, but the checks pass!

Timepiece’s Temporal Model

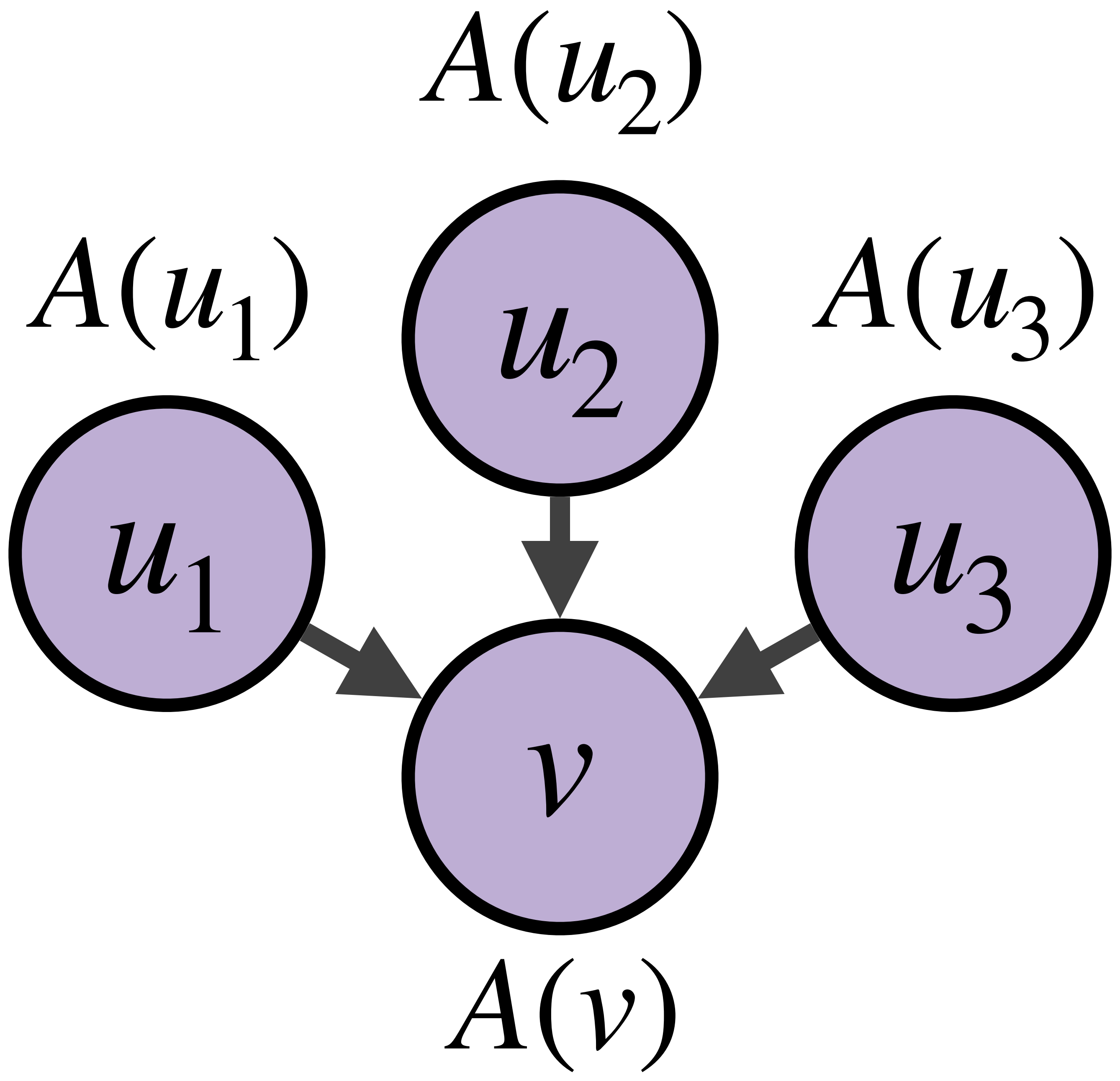


Temporal Interfaces



interface A over-approximates the converged states of the node v with a set of states $A(v)$

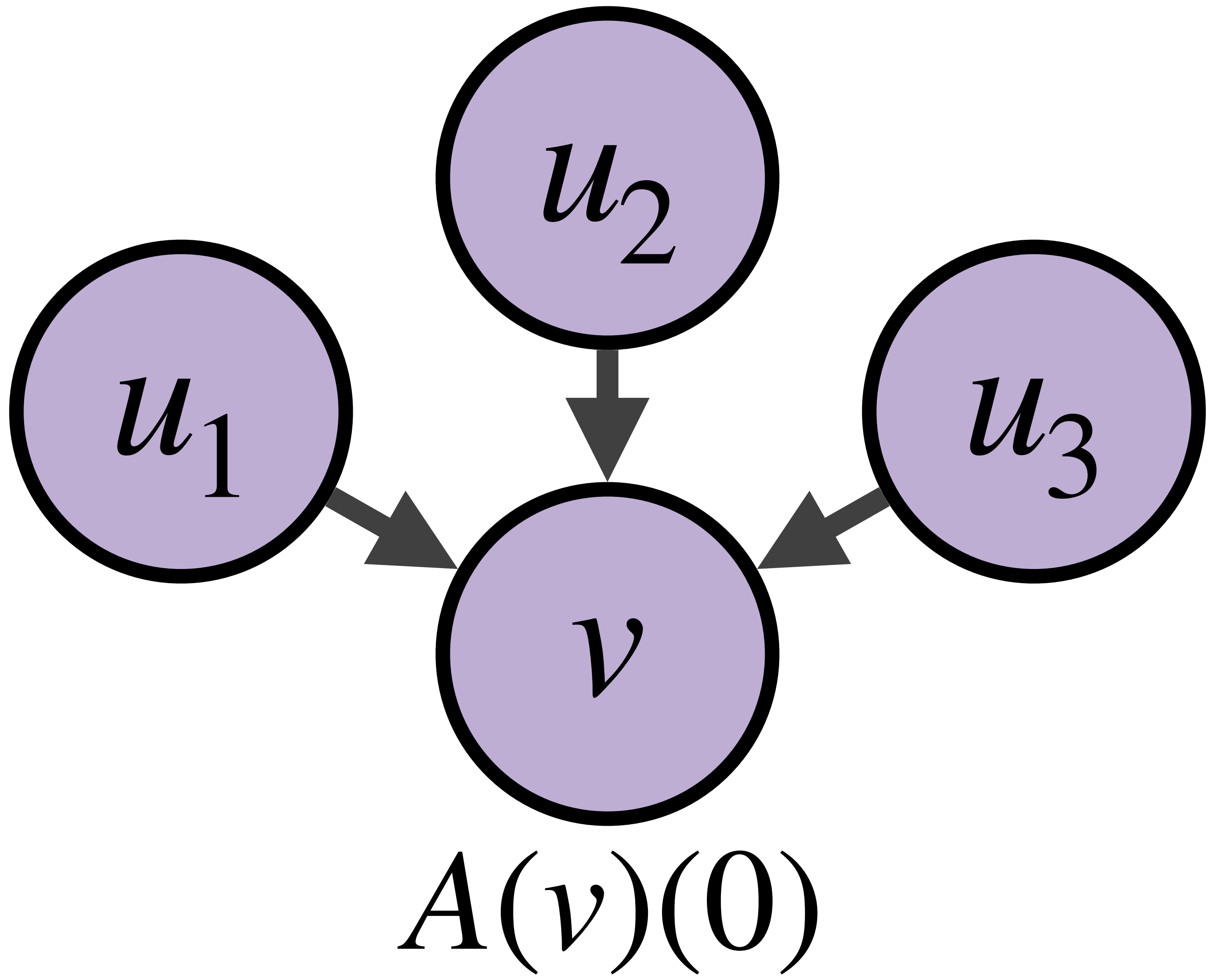
Temporal Interfaces



temporal interface A over-approximates the states of the node v **at time** t with a set of states $A(v)(t)$

Temporal Interfaces

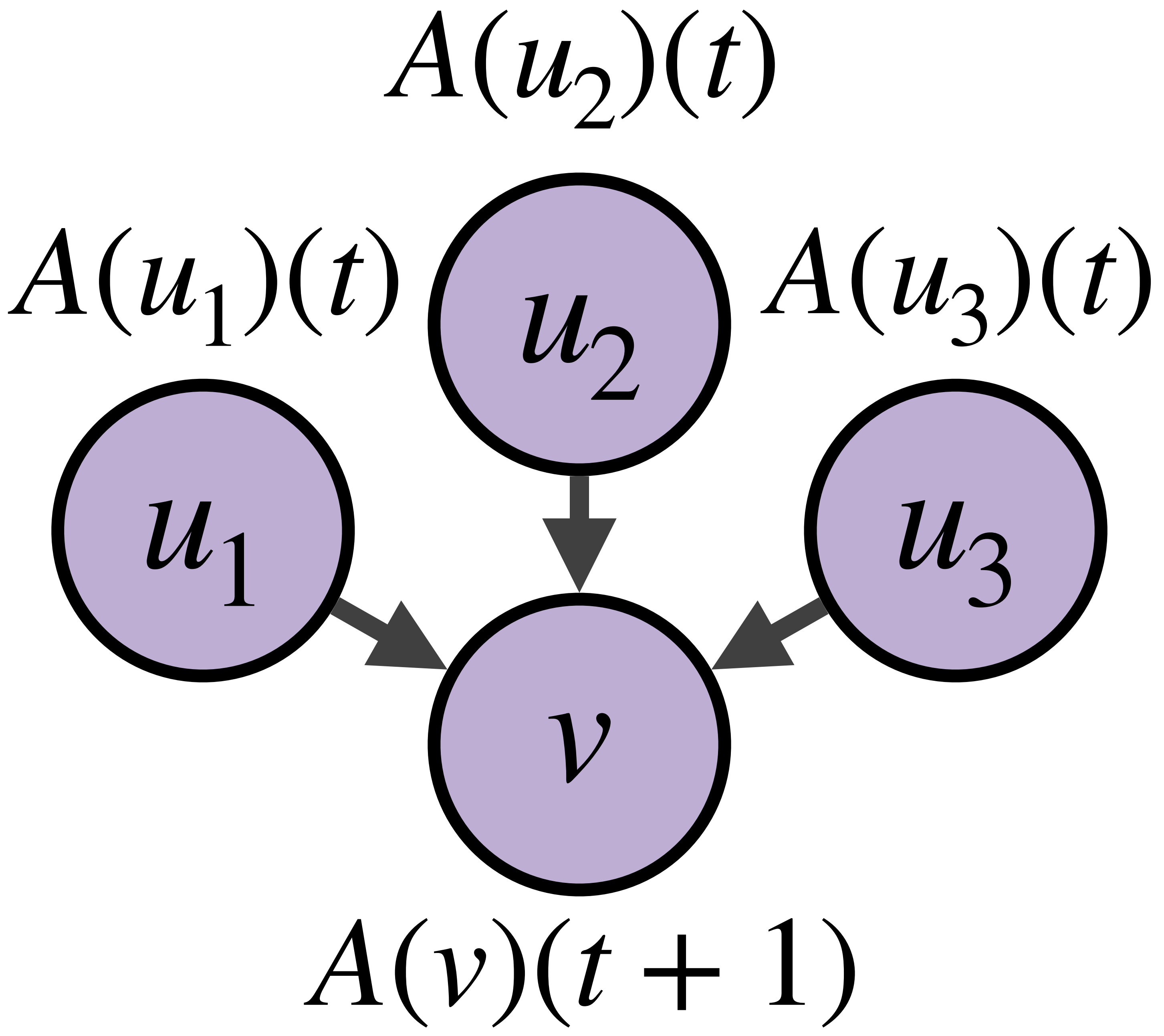
base check: interfaces at time 0 holds on initial routes



temporal interface A over-approximates the states of the node v **at time** t with a set of states $A(v)(t)$

Temporal Interfaces

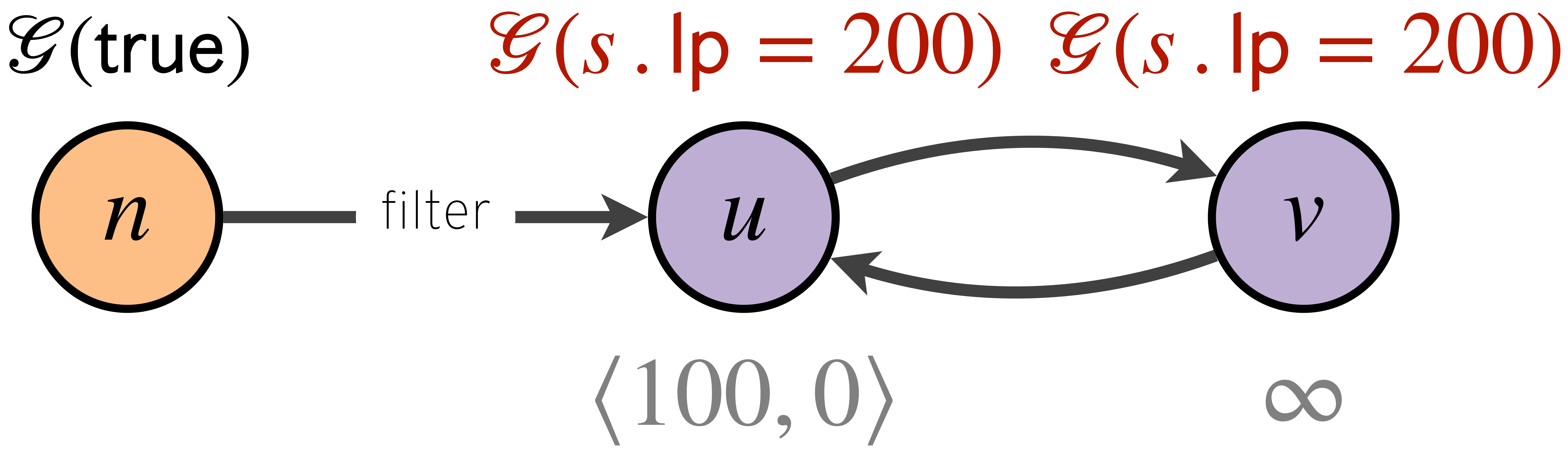
base check: interfaces at time 0 holds on initial routes



inductive check: for all times t, interfaces at time t+1 holds given interfaces from time t

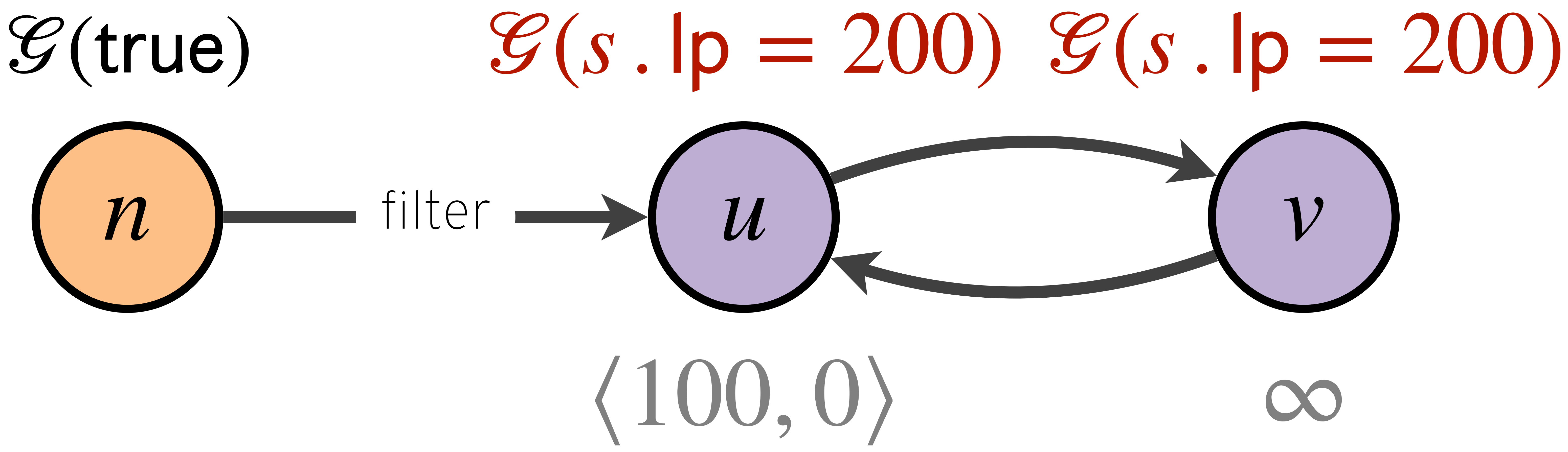
temporal interface A over-approximates the states of the node v **at time** t with a set of states $A(v)(t)$

Preventing Interference



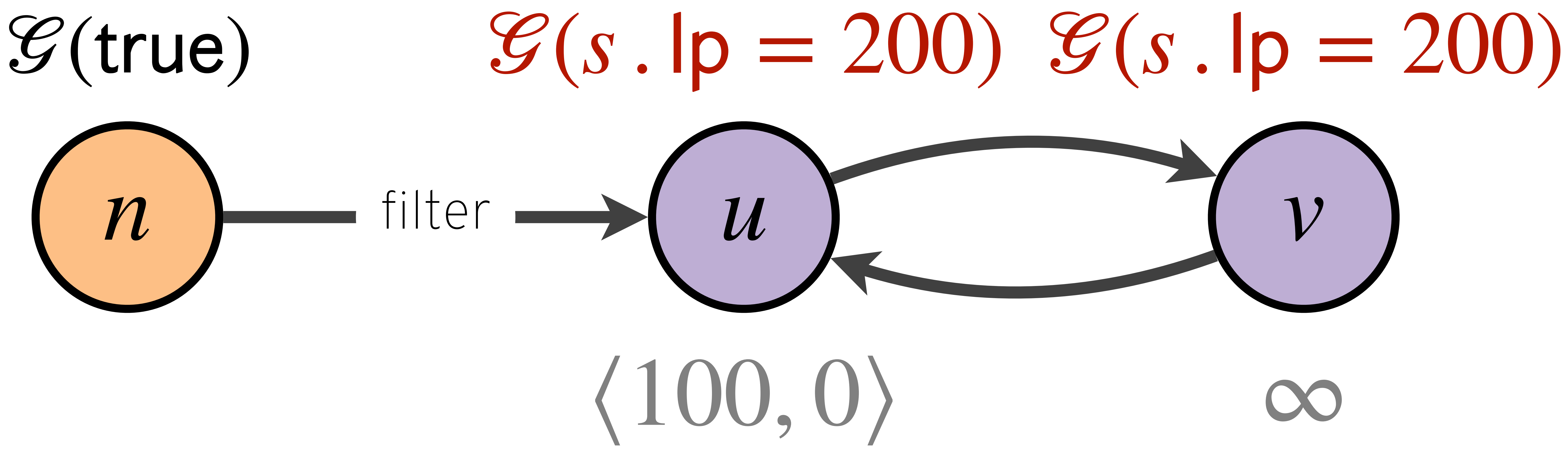
Preventing Interference

$\mathcal{G}P$ (**globally** P): at every point in time, the predicate P holds



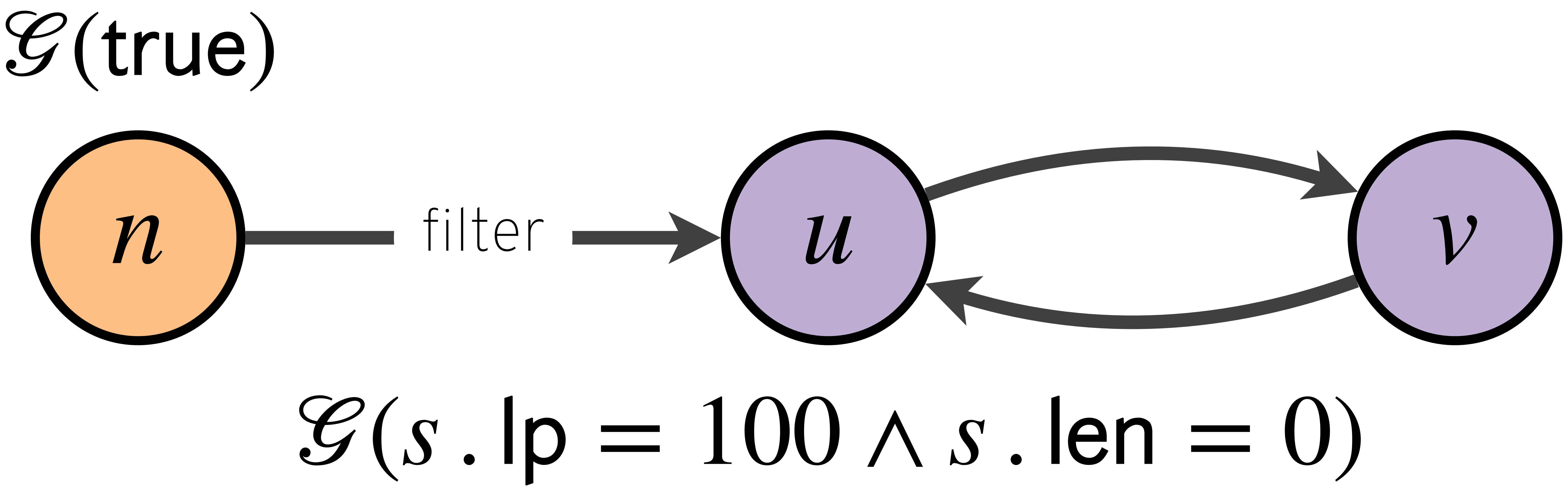
Preventing Interference

$\mathcal{G}P$ (**globally** P): at every point in time, the predicate P holds

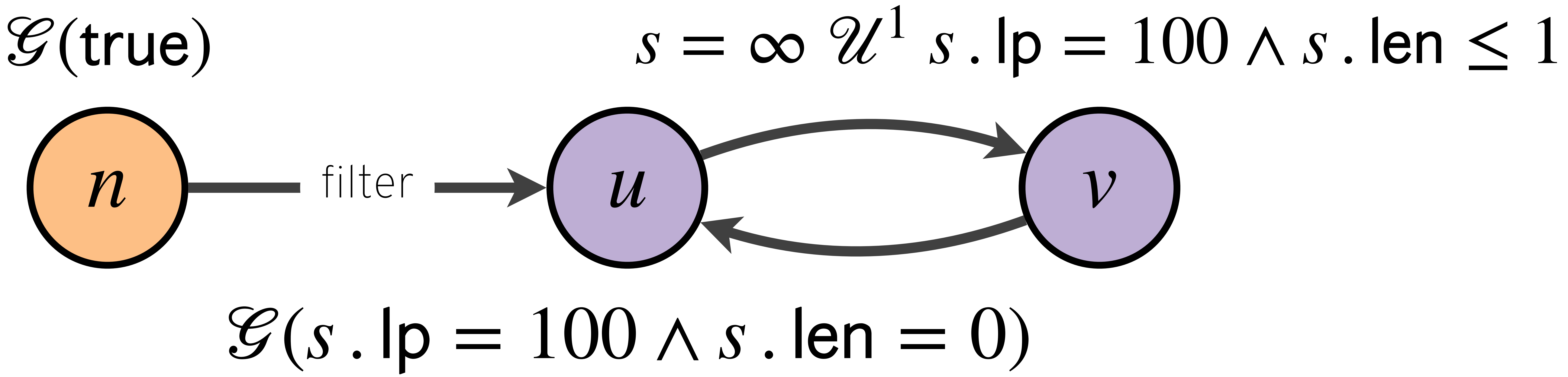


Base checks fail: interfaces $A(u)$ and $A(v)$ do not hold for initial routes at time 0

Proving Path Length



Proving Path Length

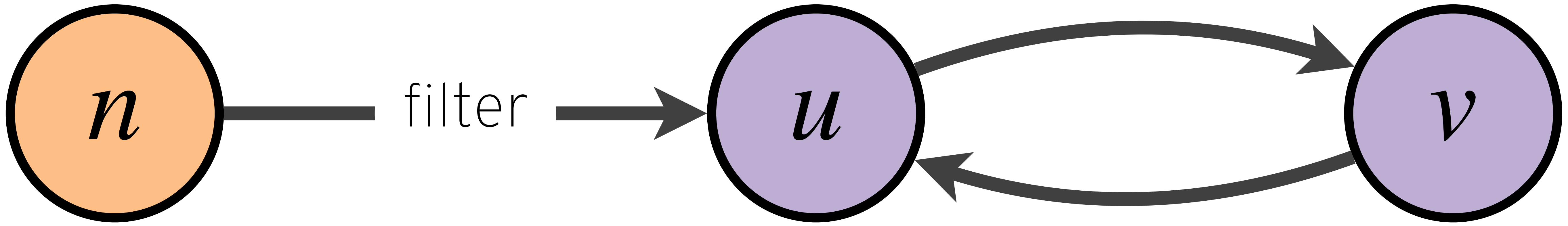


Proving Path Length

$P\ \mathcal{U}^t\ Q$ (P **until** Q **at** t): until time t, P holds; at and after t, Q holds

$\mathcal{G}(\text{true})$

$s = \infty\ \mathcal{U}^1\ s\,.\,\text{lp} = 100 \wedge s\,.\,\text{len} \leq 1$



$\mathcal{G}(s\,.\,\text{lp} = 100 \wedge s\,.\,\text{len} = 0)$

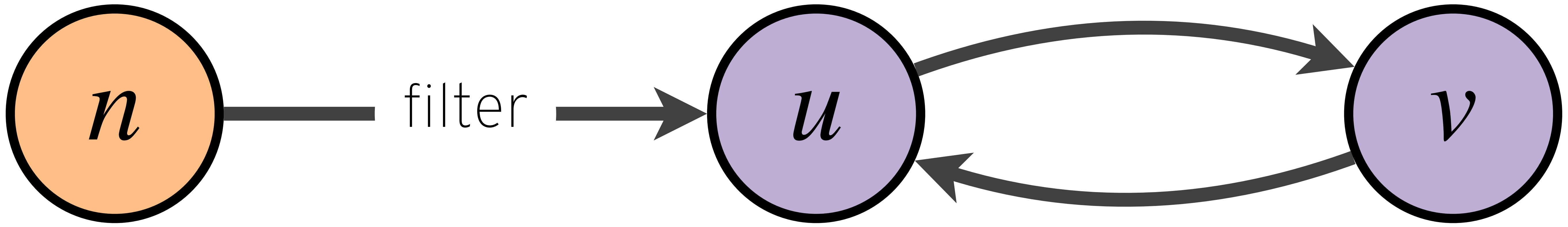
Proving Path Length

$P\ \mathcal{U}^t\ Q$ (P **until** Q **at** t): until time t, P holds; at and after t, Q holds

v has no route until time 1

$\mathcal{G}(\text{true})$

$s = \infty\ \mathcal{U}^1\ s.\text{lp} = 100 \wedge s.\text{len} \leq 1$



$\mathcal{G}(s.\text{lp} = 100 \wedge s.\text{len} = 0)$

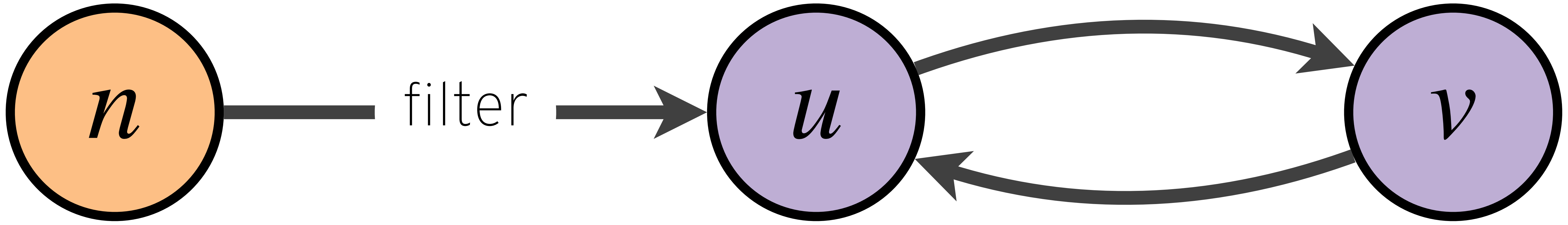
Proving Path Length

$P\ \mathcal{U}^t\ Q$ (P **until** Q **at** t): until time t, P holds; at and after t, Q holds

v has no route until time 1

$\mathcal{G}(\text{true})$

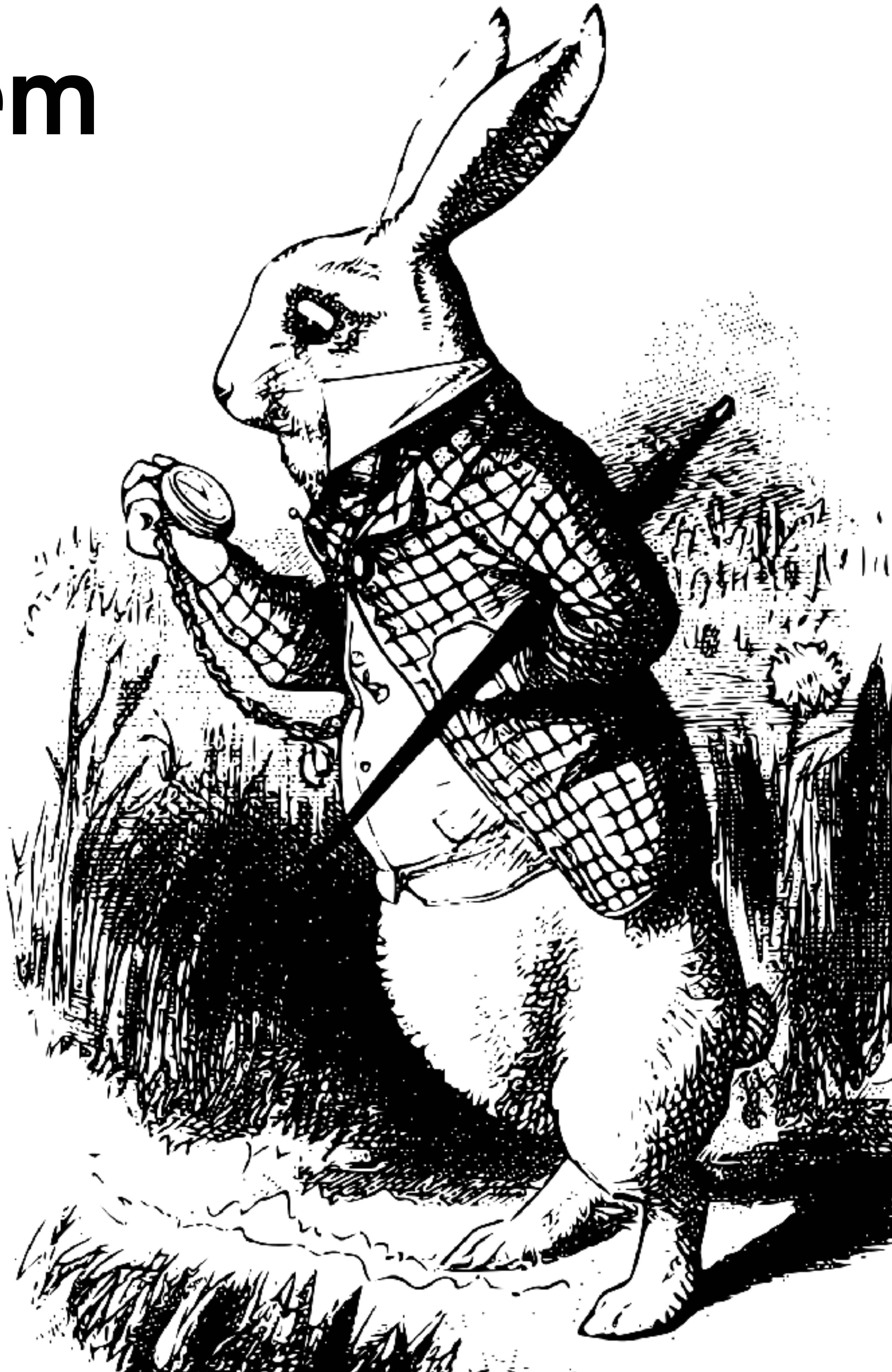
$s = \infty\ \mathcal{U}^1\ s\,.\,\text{lp} = 100 \wedge s\,.\,\text{len} \leq 1$



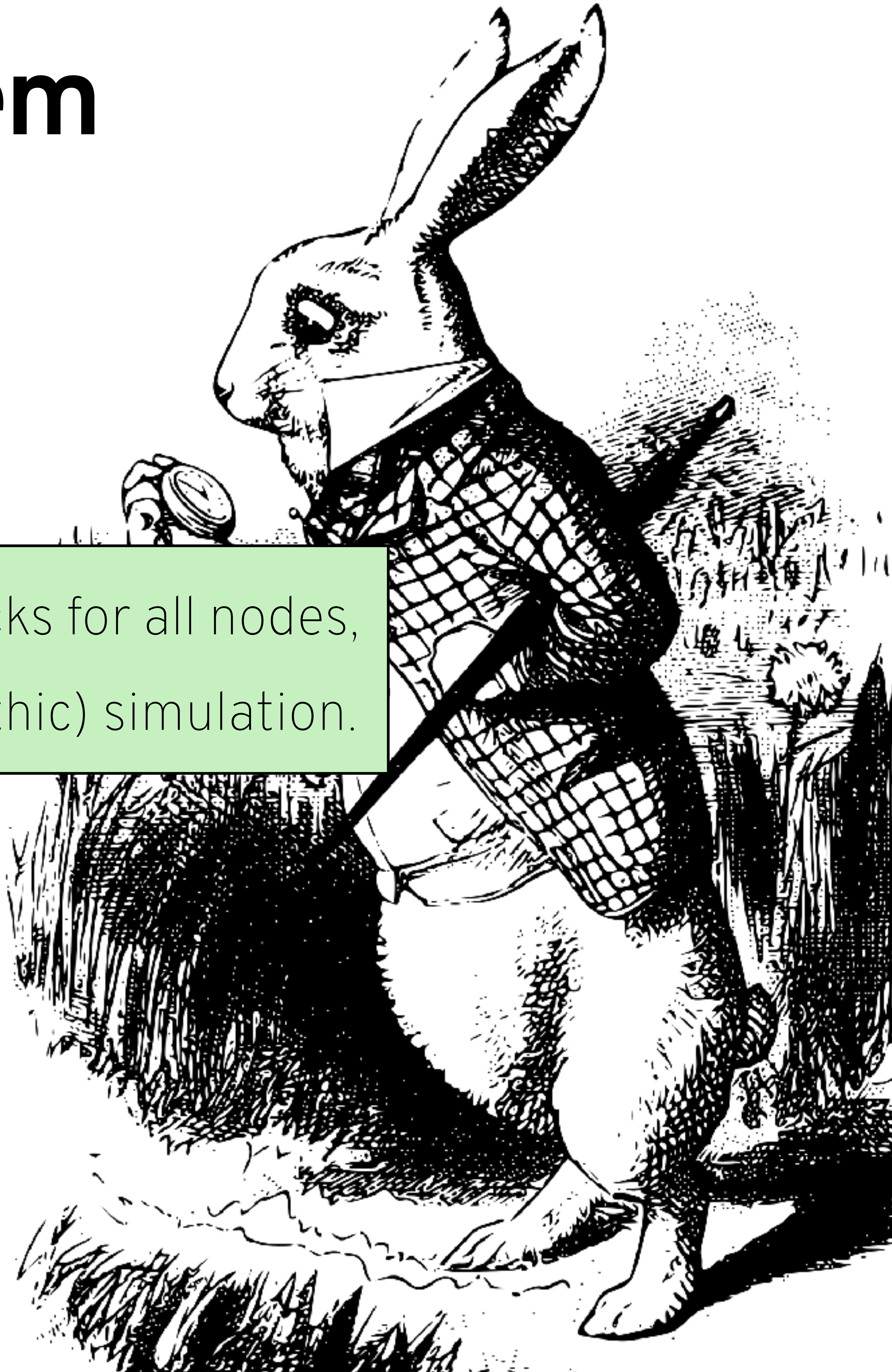
at time 1, v has a route with lp $= 100$ and len ≤ 1

$\mathcal{G}(s\,.\,\text{lp} = 100 \wedge s\,.\,\text{len} = 0)$

Soundness Theorem

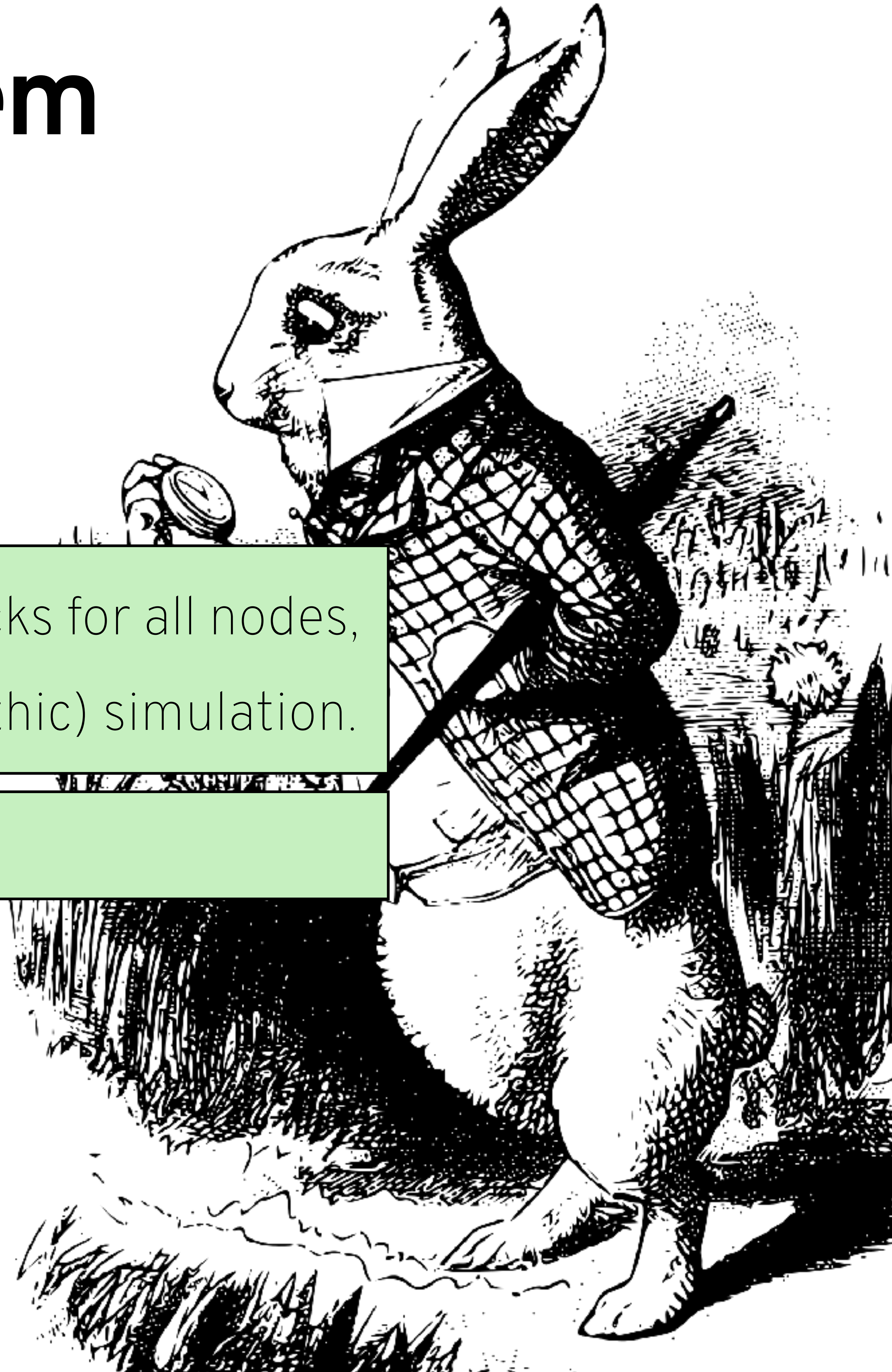


Soundness Theorem



If interface A satisfies the base and inductive checks for all nodes, then A includes all states computable via (monolithic) simulation.

Soundness Theorem

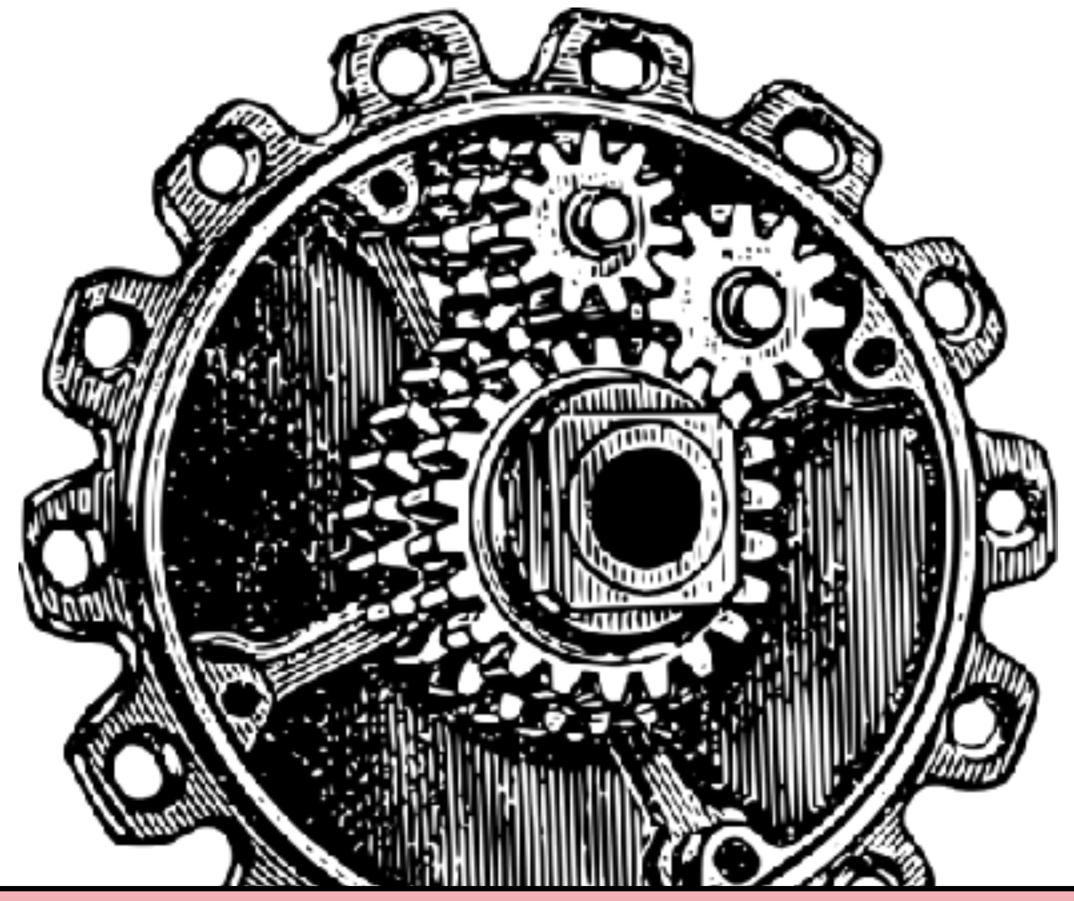


If interface A satisfies the base and inductive checks for all nodes, then A includes all states computable via (monolithic) simulation.

Proof by induction on time.

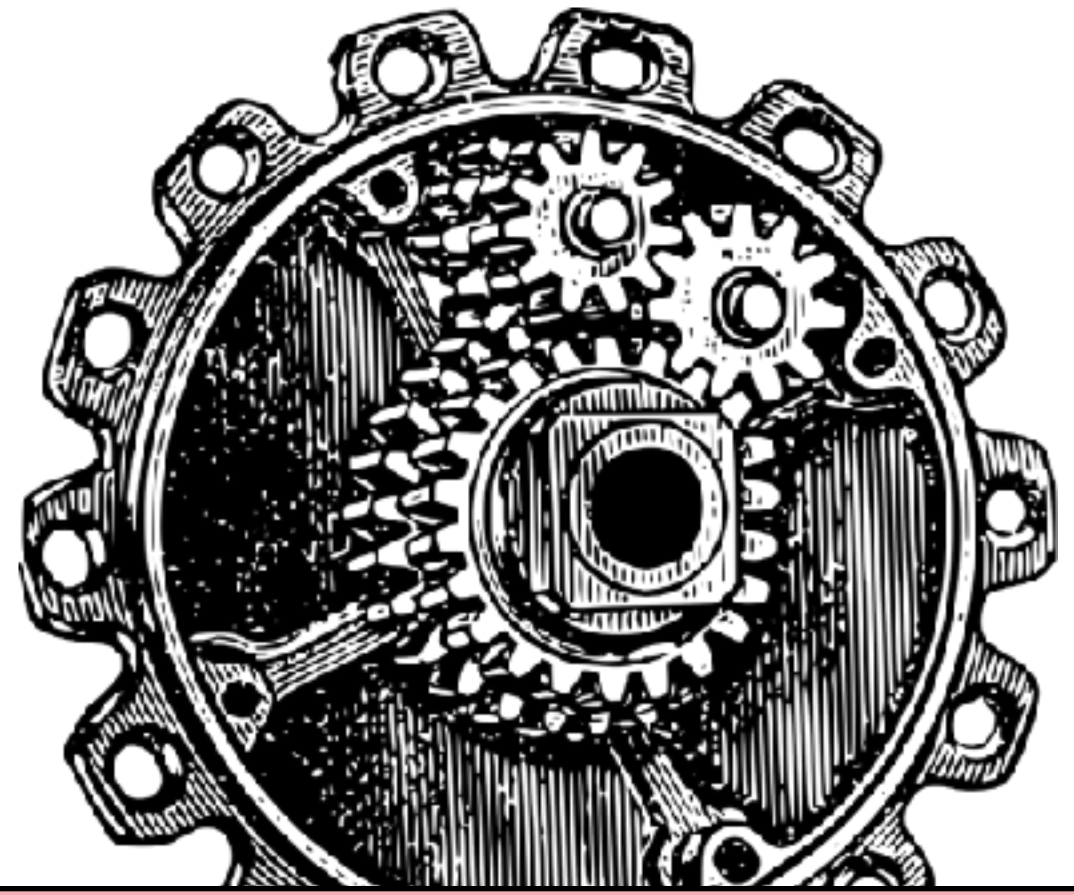
How to Use Timepiece

How to Use Timepiece



define network semantics in C# or via configurations (via Batfish)

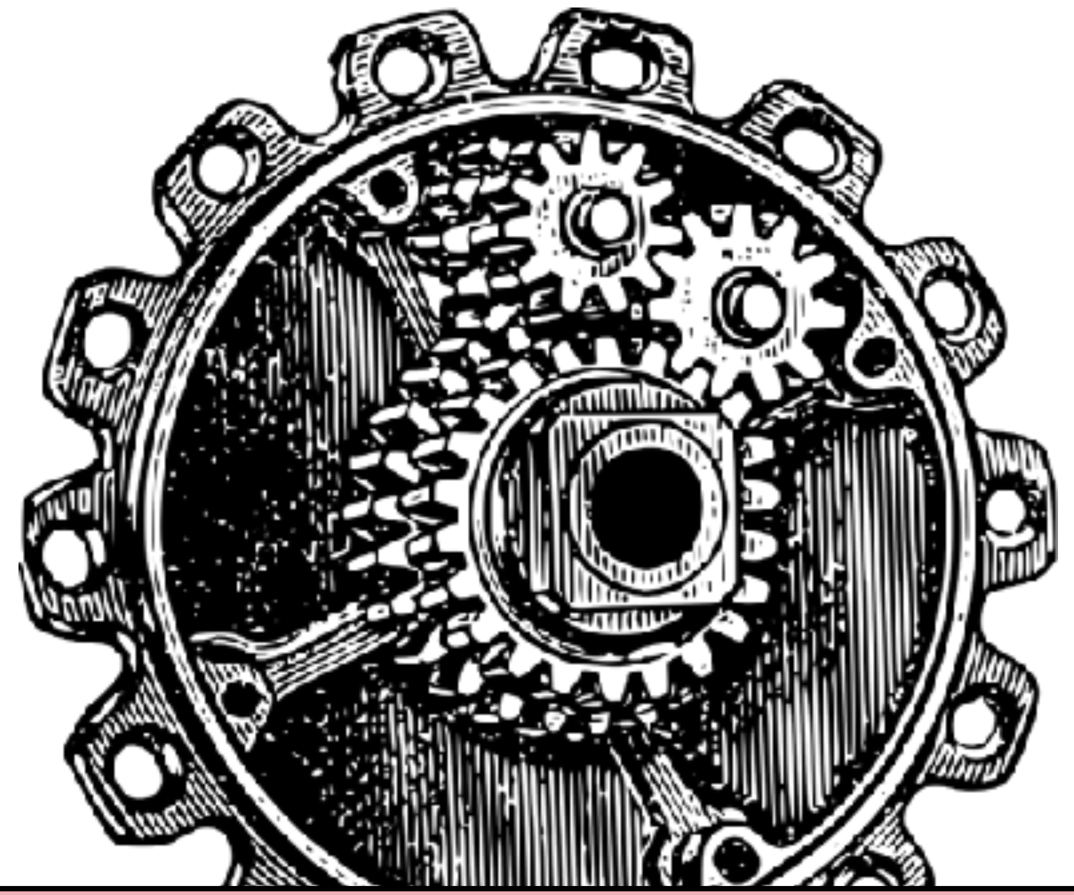
How to Use Timepiece



define network semantics in C# or via configurations (via Batfish)

write interfaces using C# library of temporal operators ($\mathscr{G}, \mathscr{U}^t, \mathscr{F}^t$)

How to Use Timepiece

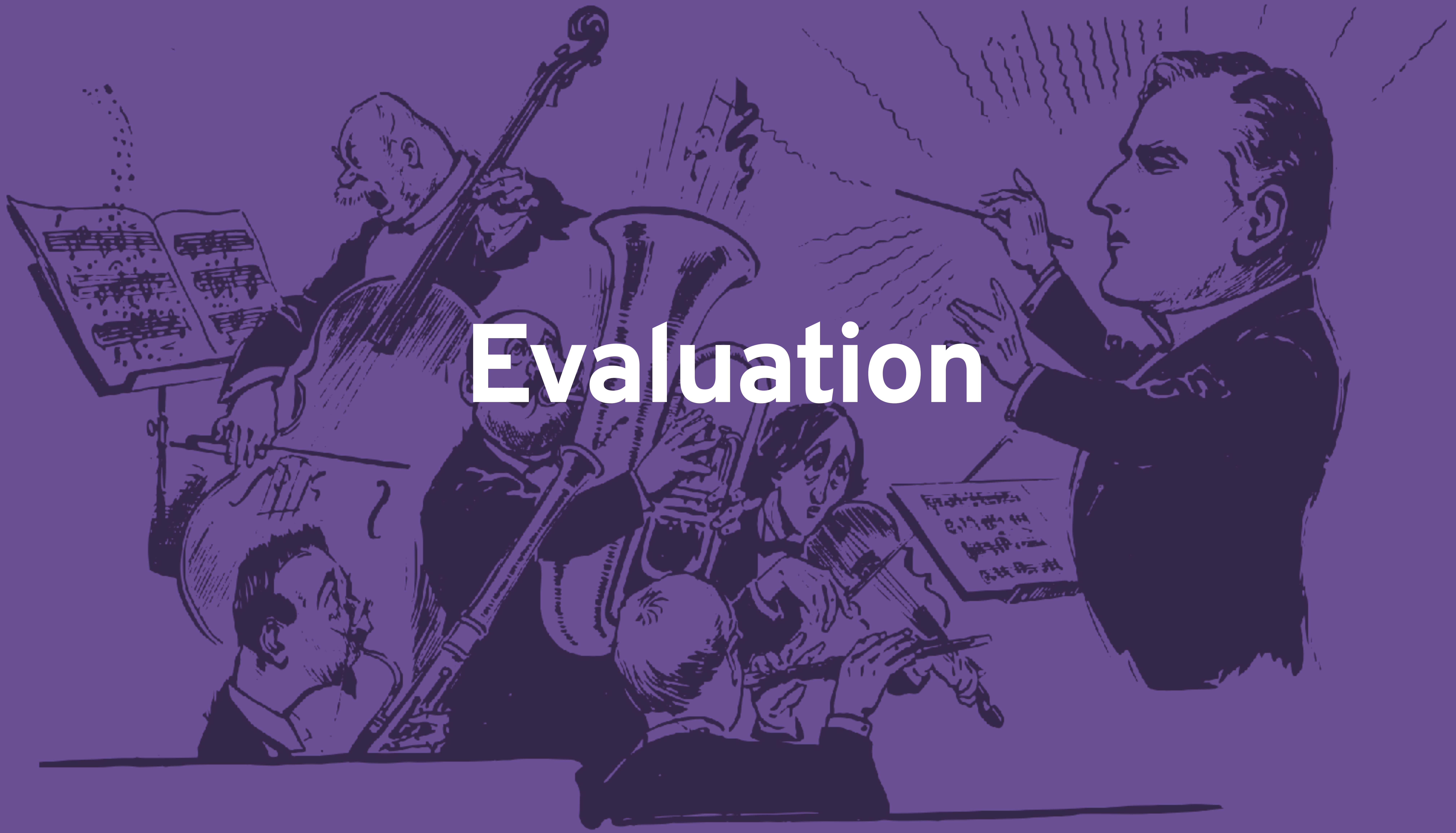


define network semantics in C# or via configurations (via Batfish)

write interfaces using C# library of temporal operators ($\mathscr{G}, \mathscr{U}^t, \mathscr{F}^t$)

check VCs *in parallel* on every node using Satisfiability Modulo Theories (SMT) solver

Evaluation



Evaluation

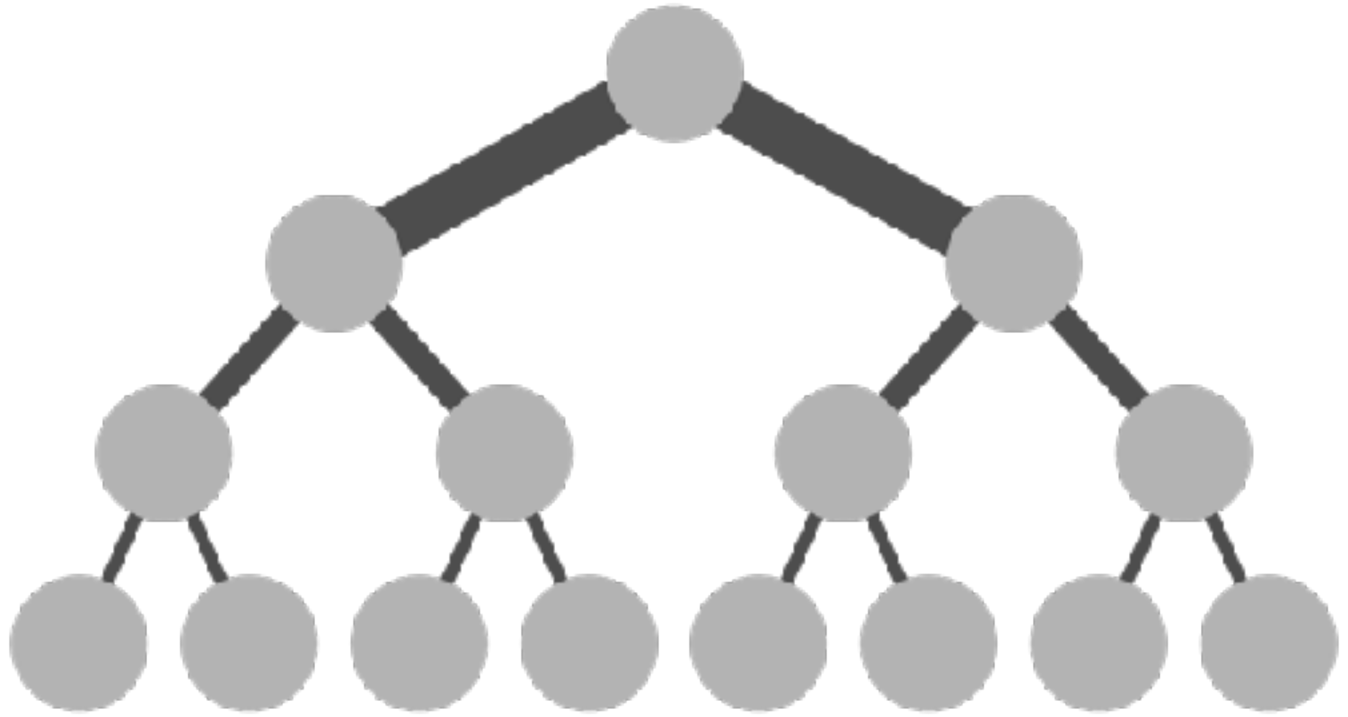
Evaluation

does Timepiece scale to large networks?

does Timepiece handle complex policies?

how easy is it to write invariants for different properties?

Evaluation

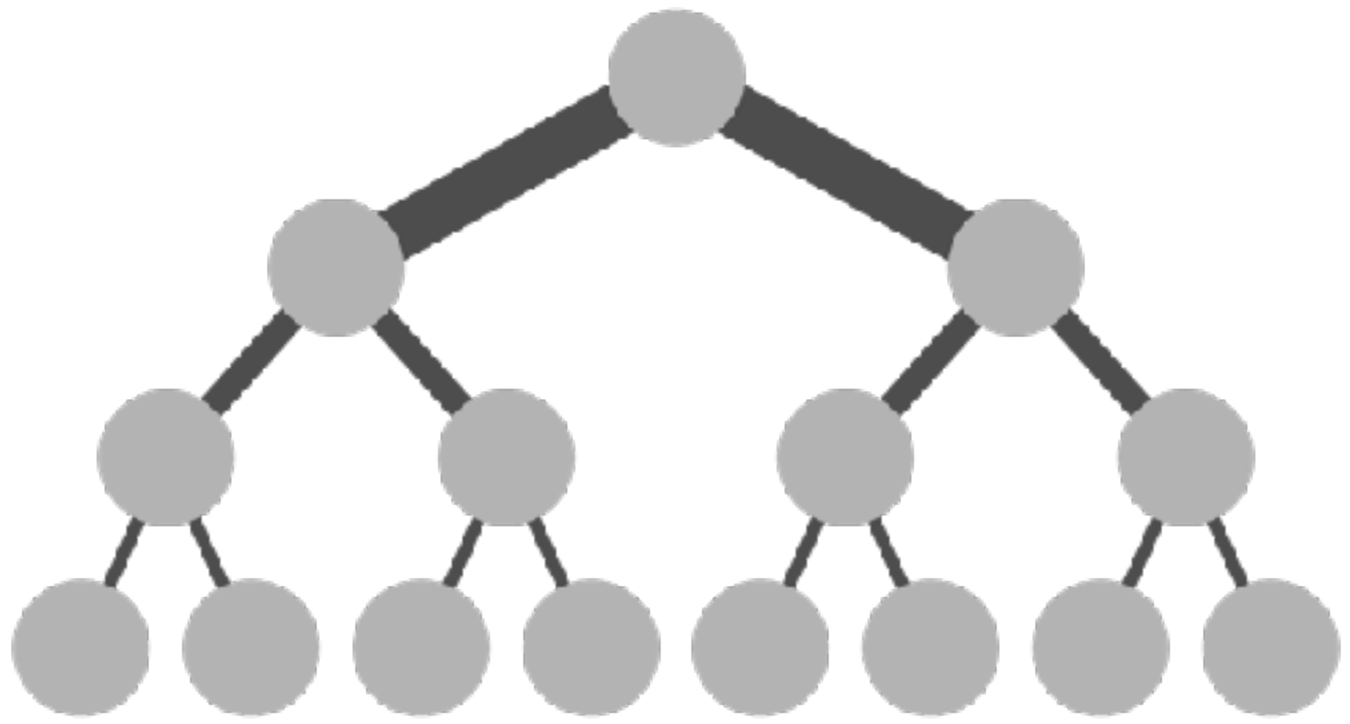


Fat-tree data center networks
C# model of eBGP routing protocol
20–2000 nodes

does Timepiece handle complex policies?

how easy is it to write invariants for different properties?

Evaluation



Fat-tree data center networks

C# model of eBGP routing protocol

20–2000 nodes

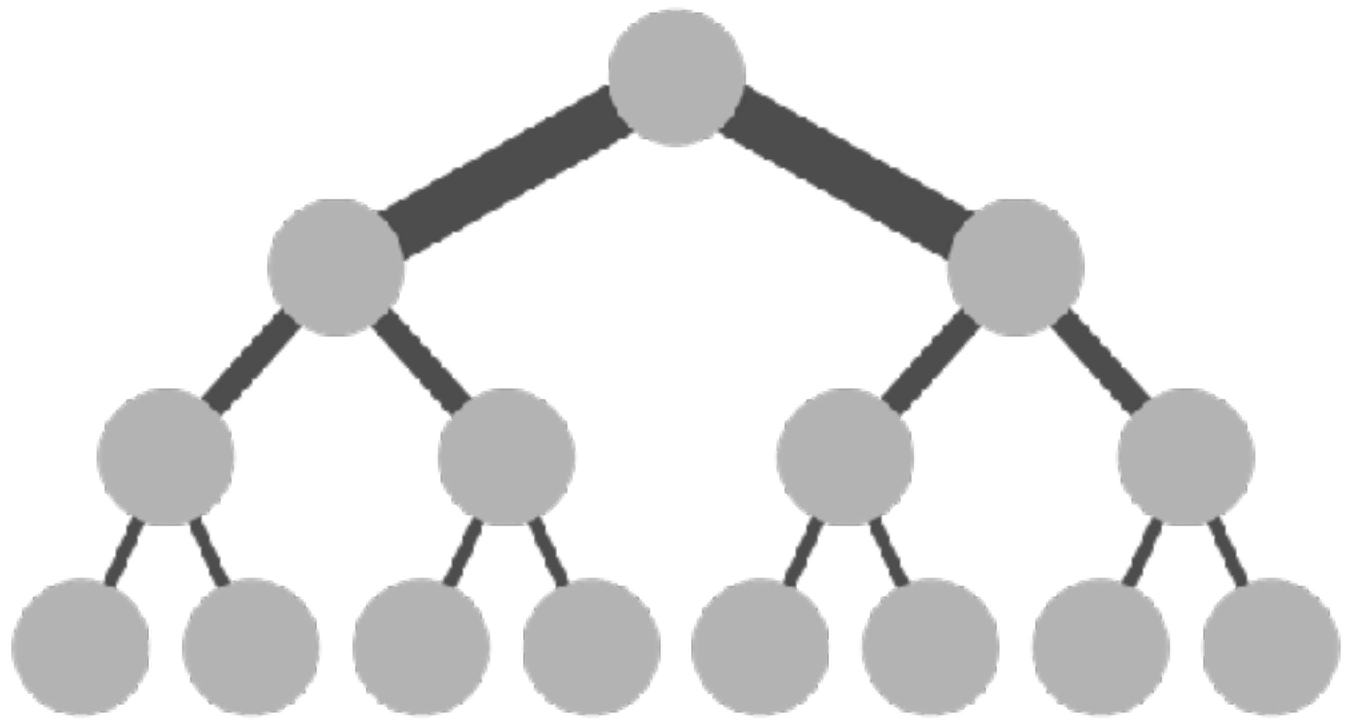
Internet2 wide-area network

102,753 lines of Juniper configuration code

263 nodes (10 internal, 253 external)

how easy is it to write invariants for different properties?

Evaluation



Fat-tree data center networks
C# model of eBGP routing protocol
20–2000 nodes

Reachability | Path length
Valley freedom | Hijack filtering

Internet2 wide-area network
102,753 lines of Juniper configuration code
263 nodes (10 internal, 253 external)

No transit

Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

Benchmark
Reachability
Path length
Valley freedom
Hijack filtering
No transit

Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

Benchmark	Network LoC
Reachability	81
Path length	88
Valley freedom	89
Hijack filtering	146
No transit	88 (+102,753)

Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

Benchmark	Network LoC	Annotation LoC
Reachability	81	3
Path length	88	7
Valley freedom	89	12
Hijack filtering	146	21
No transit	88 (+102,753)	5

Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

Benchmark	Network LoC	Annotation LoC	Nodes
Reachability	81	3	2000
Path length	88	7	2000
Valley freedom	89	12	2000
Hijack filtering	146	21	2000
No transit	88 (+102,753)	5	263

Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

Benchmark	Network LoC	Annotation LoC	Nodes	Monolithic time
Reachability	81	3	2000	14s
Path length	88	7	2000	>2h
Valley freedom	89	12	2000	>2h
Hijack filtering	146	21	2000	>2h
No transit	88 (+102,753)	5	263	>2h

Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

Benchmark	Network LoC	Annotation LoC	Nodes	Monolithic time	Modular time
Reachability	81	3	2000	14s	28s
Path length	88	7	2000	>2h	1204s
Valley freedom	89	12	2000	>2h	398s
Hijack filtering	146	21	2000	>2h	142s
No transit	88 (+102,753)	5	263	>2h	38s

Evaluation

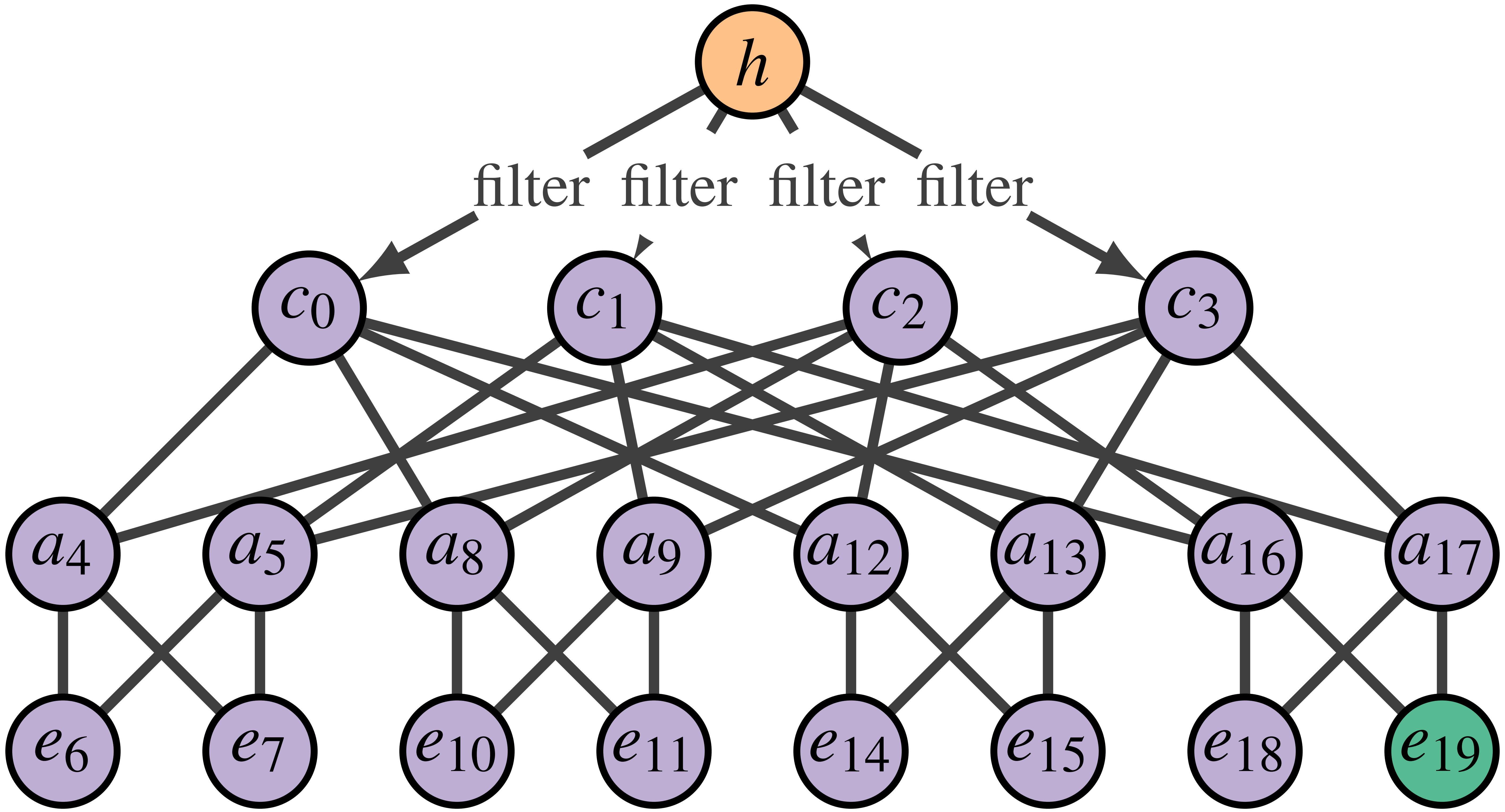
on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

Benchmark	Network LoC	Annotation LoC	Nodes	Monolithic time	Modular time
Reachability	81	3	2000	14s	28s
Path length	88	7	2000	>2h	1204s
Valley freedom	89	12	2000	>2h	398s
Hijack filtering	146	21	2000	>2h	142s
No transit	88 (+102,753)	5	263	>2h	38s

Fat-tree Hijack Filtering

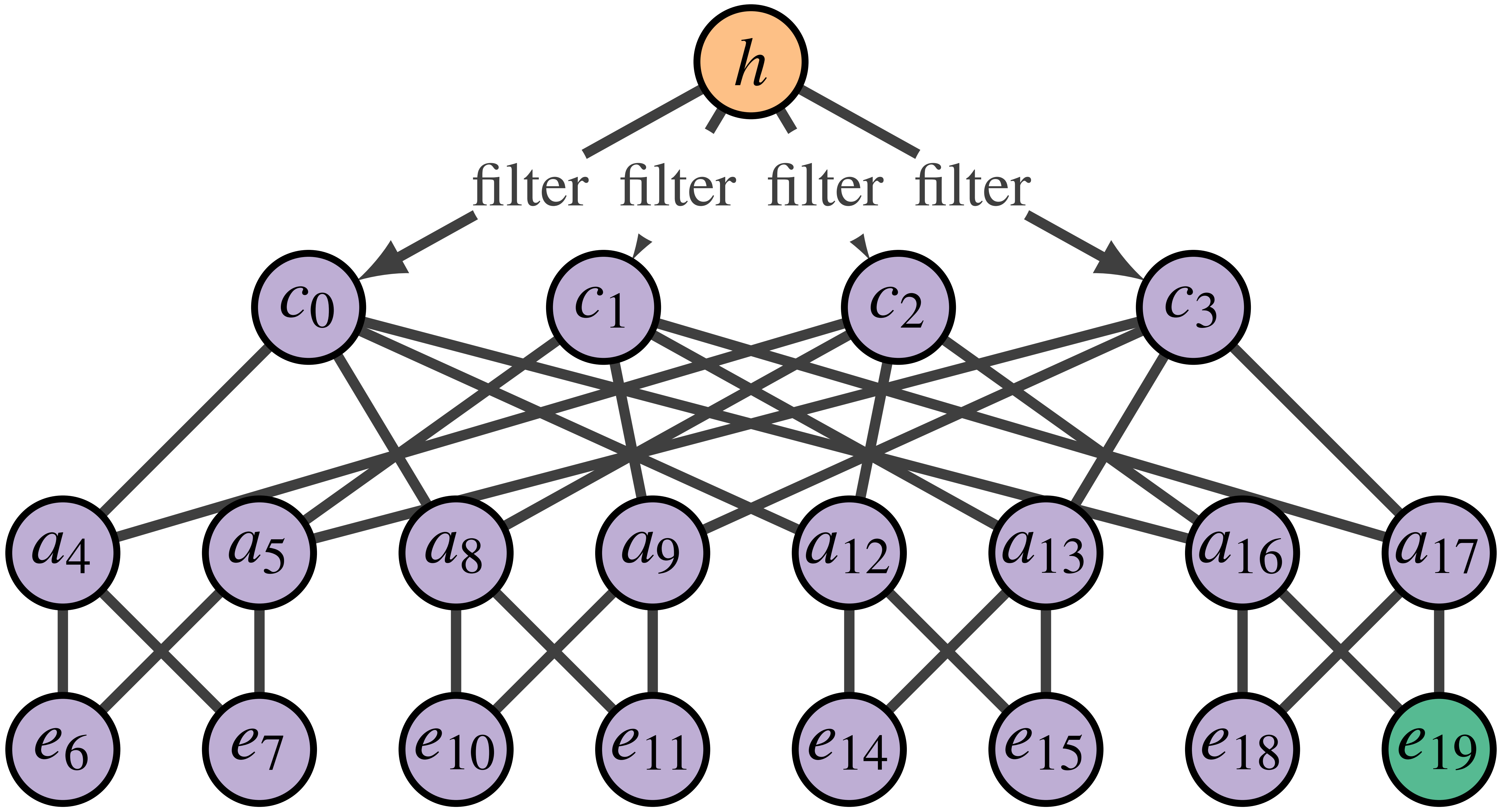
BGP misconfiguration/attack:
a “hijacker” node announces it has a path to a prefix it doesn’t own, misleading others to route through the hijacker

Fat-tree Hijack Filtering



BGP misconfiguration/attack:
a “hijacker” node announces it has a path to a prefix it doesn’t own, misleading others to route through the hijacker

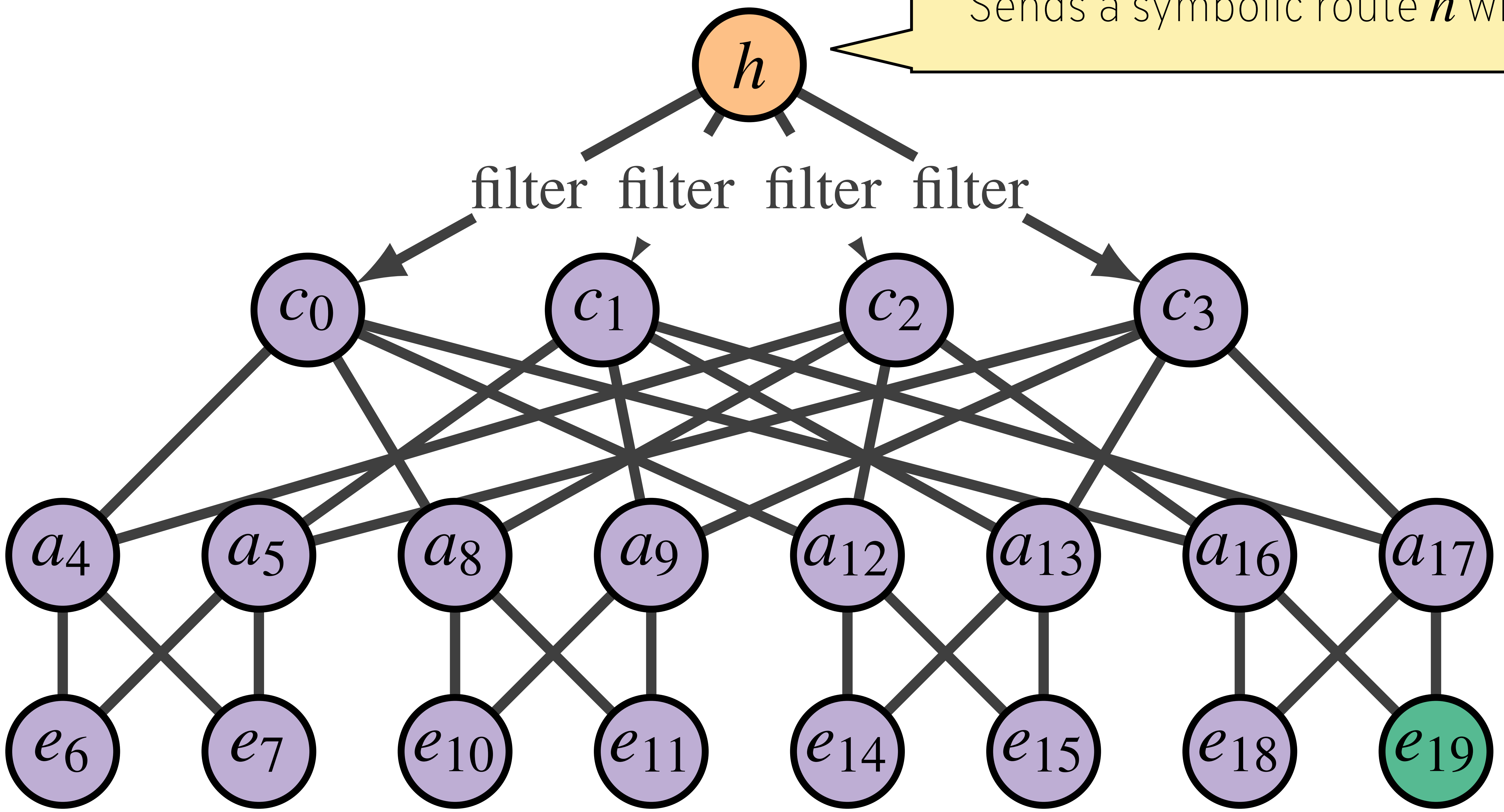
Fat-tree Hijack Filtering



BGP misconfiguration/attack:
a "hijacker" node announces it has a path to a prefix it doesn't own, misleading others to route through the hijacker

Sends a route with symbolic prefix p

Fat-tree Hijack Filtering



Sends a symbolic route h with **tag** true

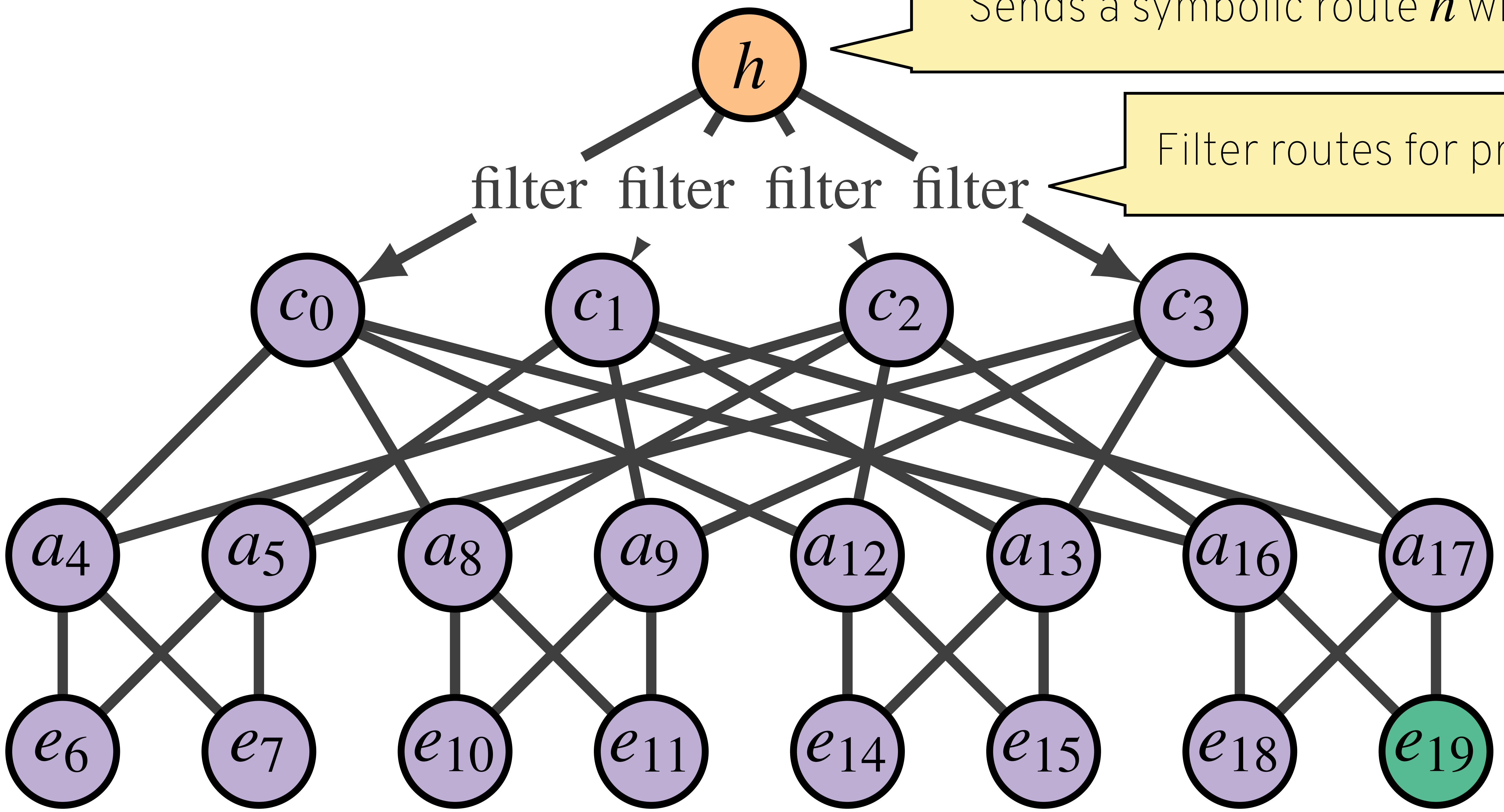
BGP misconfiguration/attack:
a "hijacker" node announces it has a path to a prefix it doesn't own, misleading others to route through the hijacker

Sends a route with symbolic prefix p

Fat-tree Hijack Filtering

Sends a symbolic route h with **tag** true

Filter routes for prefix p



BGP misconfiguration/attack:
a "hijacker" node announces it has a path to a prefix it doesn't own, misleading others to route through the hijacker

Sends a route with symbolic prefix p

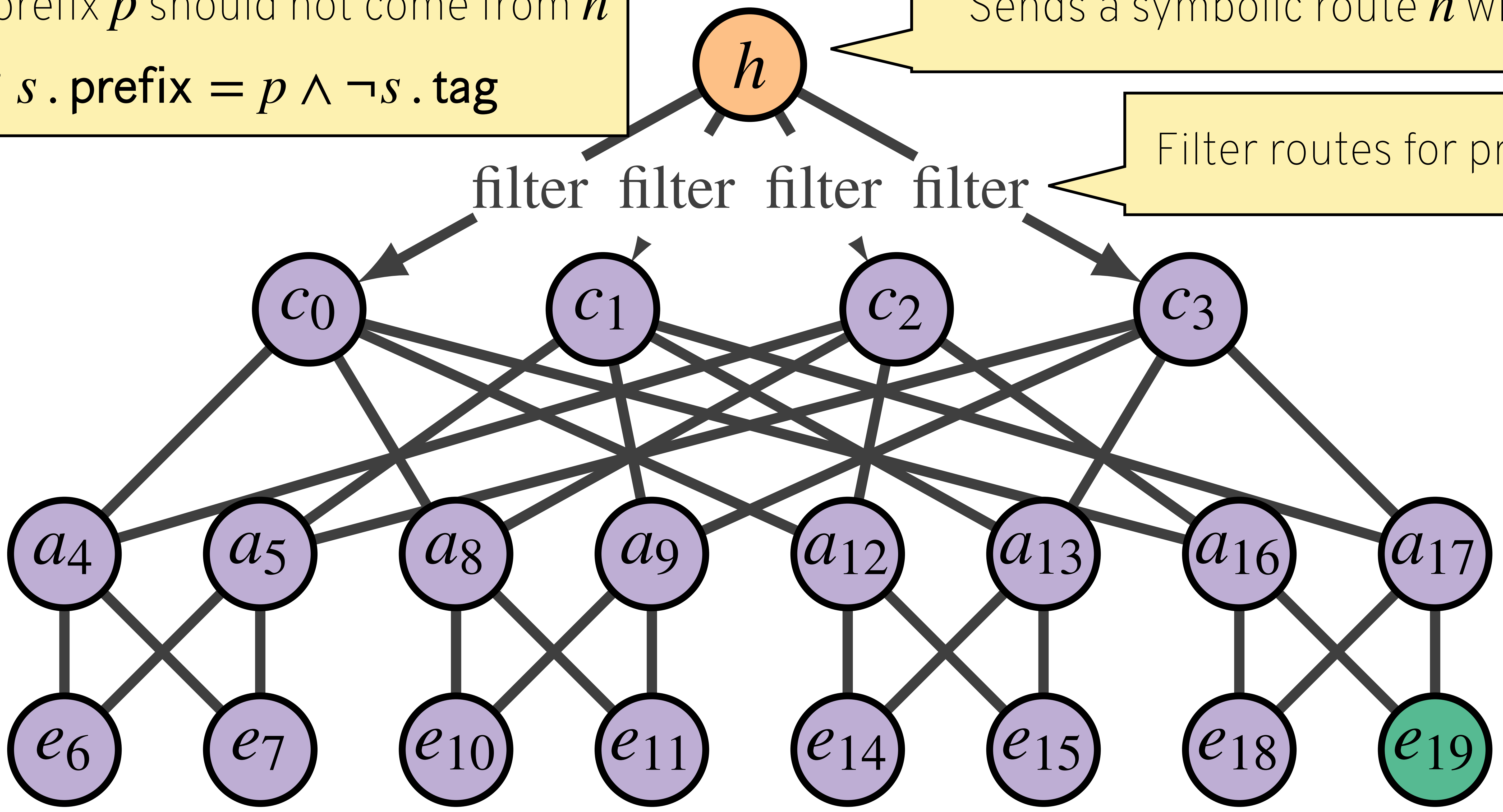
Fat-tree Hijack Filtering

Converged routes for prefix p should not come from h

$P(v) \equiv \text{true}\ \mathcal{U}^4\ s.\text{prefix} = p \wedge \neg s.\text{tag}$

Sends a symbolic route h with **tag** true

Filter routes for prefix p



BGP misconfiguration/attack:
a "hijacker" node announces it has a path to a prefix it doesn't own, misleading others to route through the hijacker

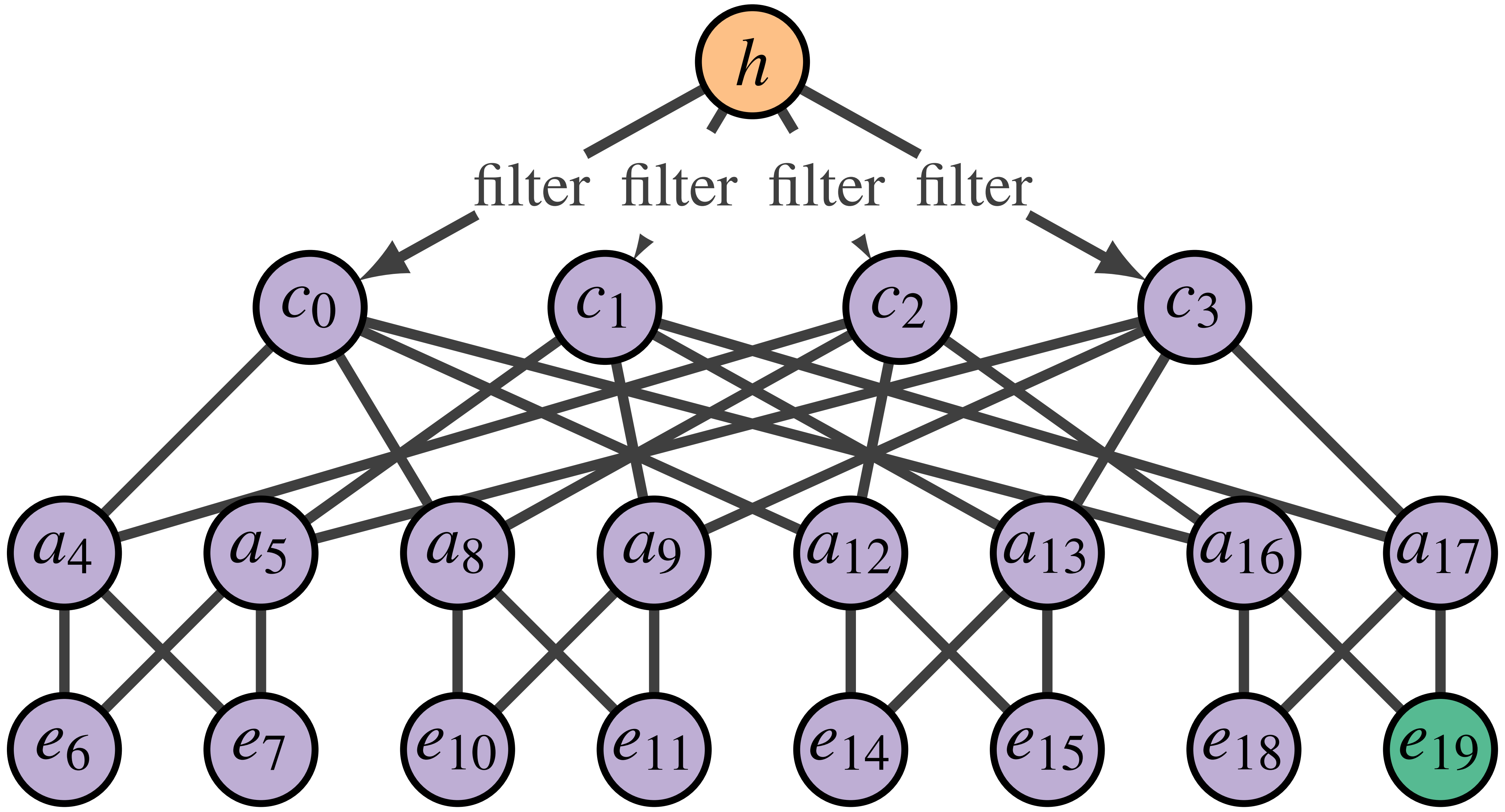
Sends a route with symbolic prefix p

Fat-tree Hijack Filtering

Converged routes for prefix $\boldsymbol{p}$ should not come from $\boldsymbol{h}$

$$P(v) \equiv \text{true}\ \mathcal{U}^4\ s\,.\,\text{prefix} = p \land \neg s\,.\,\text{tag}$$

Interface **composes** an "eventual invariant" with a "safety invariant"



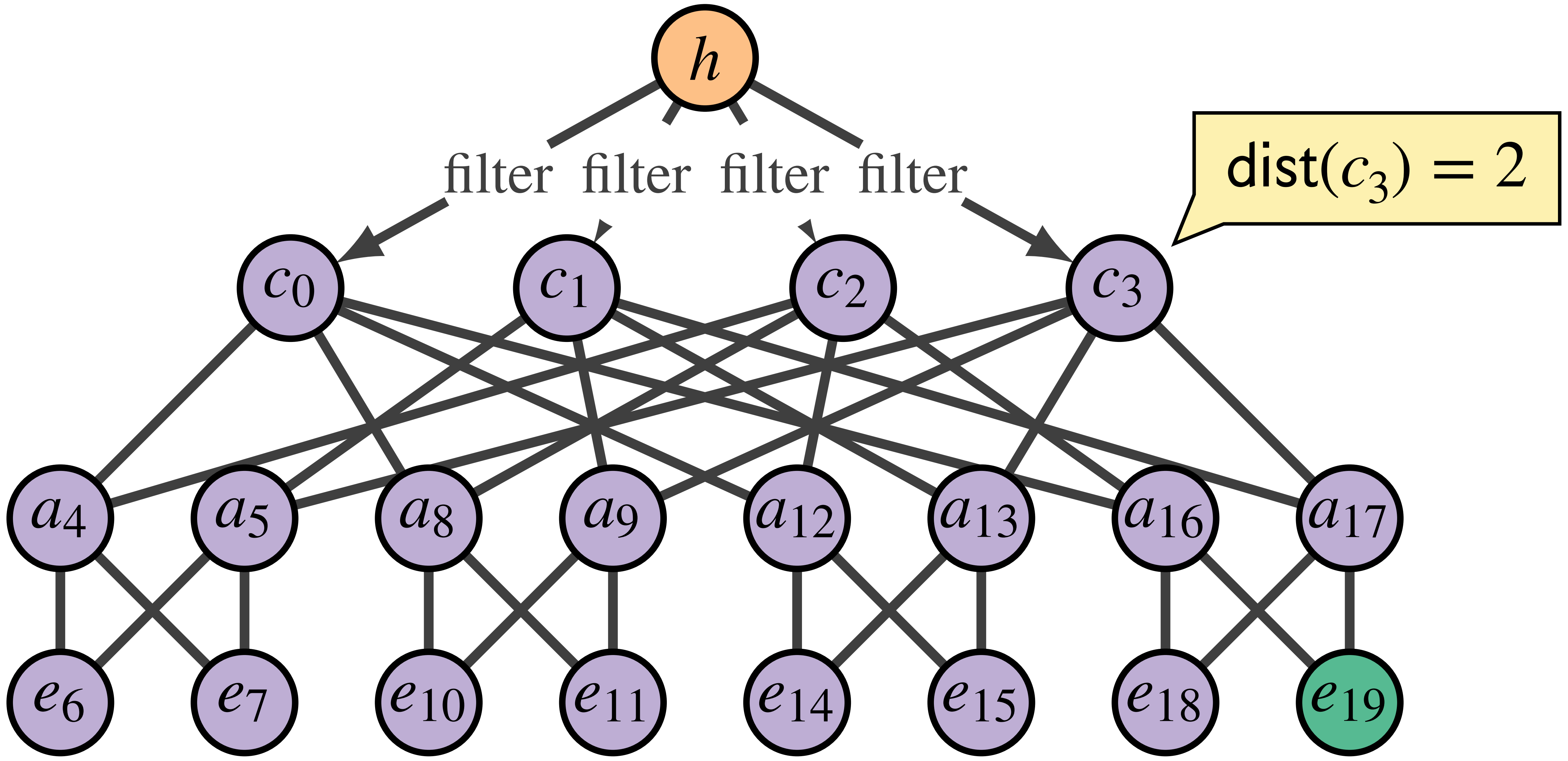
Fat-tree Hijack Filtering

Converged routes for prefix $\boldsymbol{p}$ should not come from $\boldsymbol{h}$

$P(v) \equiv \mathsf{true}\ \mathscr{U}^4\ s\,.\,\mathsf{prefix} = p \land \neg s\,.\,\mathsf{tag}$

Interface **composes** an “eventual invariant” with a “safety invariant”

All nodes’ interfaces are parameterized by their distance $\mathsf{dist}(v)$ from $\boldsymbol{e_{19}}$



Fat-tree Hijack Filtering

Converged routes for prefix p should not come from h

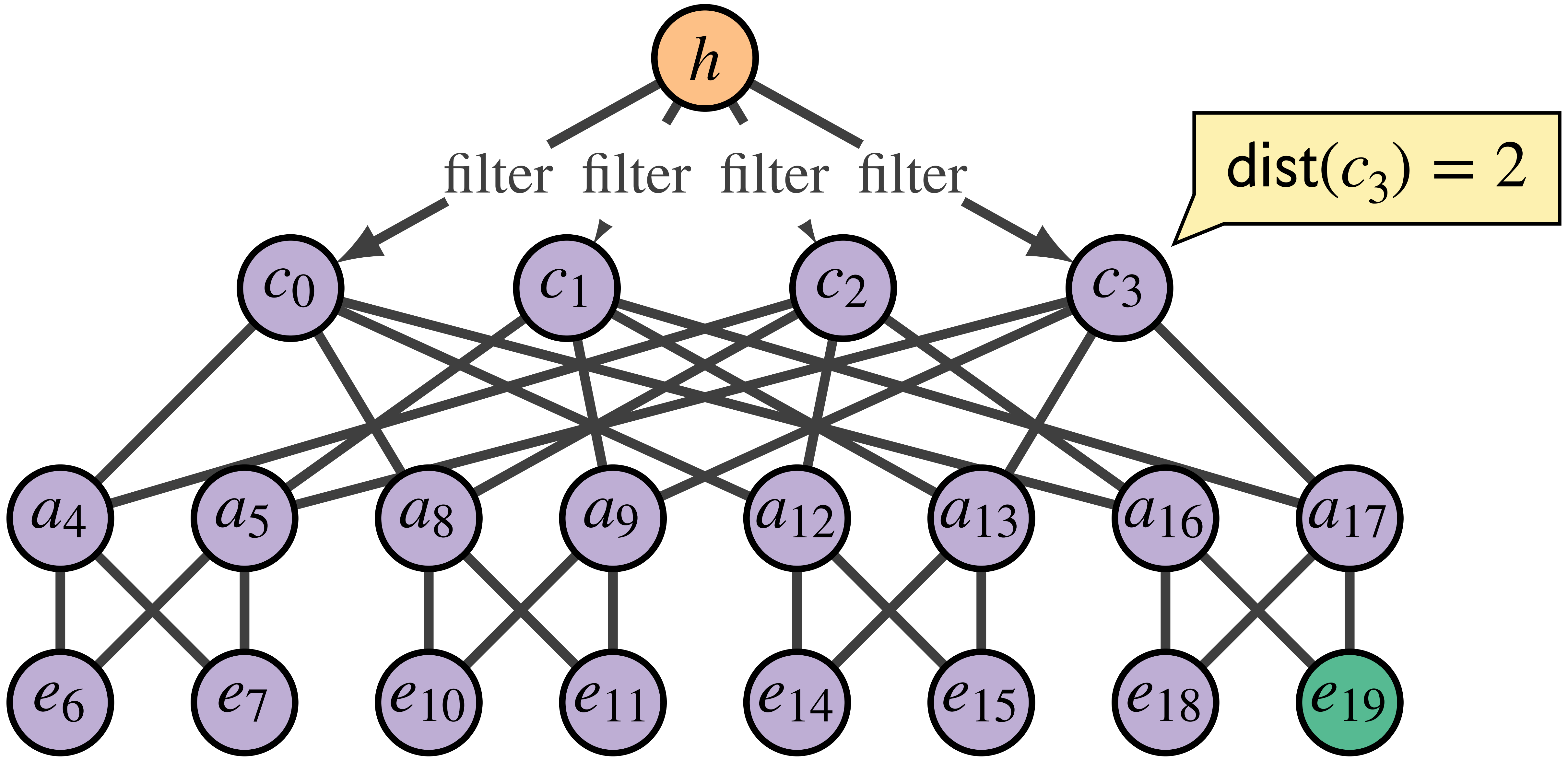
$P(v) \equiv \text{true}\ \mathcal{U}^4\ s\,.\,\text{prefix} = p \land \neg s\,.\,\text{tag}$

Interface **composes** an "eventual invariant" with a "safety invariant"

All nodes' interfaces are parameterized by their distance $\text{dist}(v)$ from e_{19}

Nodes are **eventually** "internally reachable"

$\text{true}\ \mathcal{U}^{\text{dist}(v)}\ s\,.\,\text{prefix} = p \land \neg s\,.\,\text{tag}$



Fat-tree Hijack Filtering

Converged routes for prefix p should not come from h

$P(v) \equiv \mathsf{true}\ \mathcal{U}^4\ s.\mathsf{prefix} = p \land \neg s.\mathsf{tag}$

Interface **composes** an "eventual invariant" with a "safety invariant"

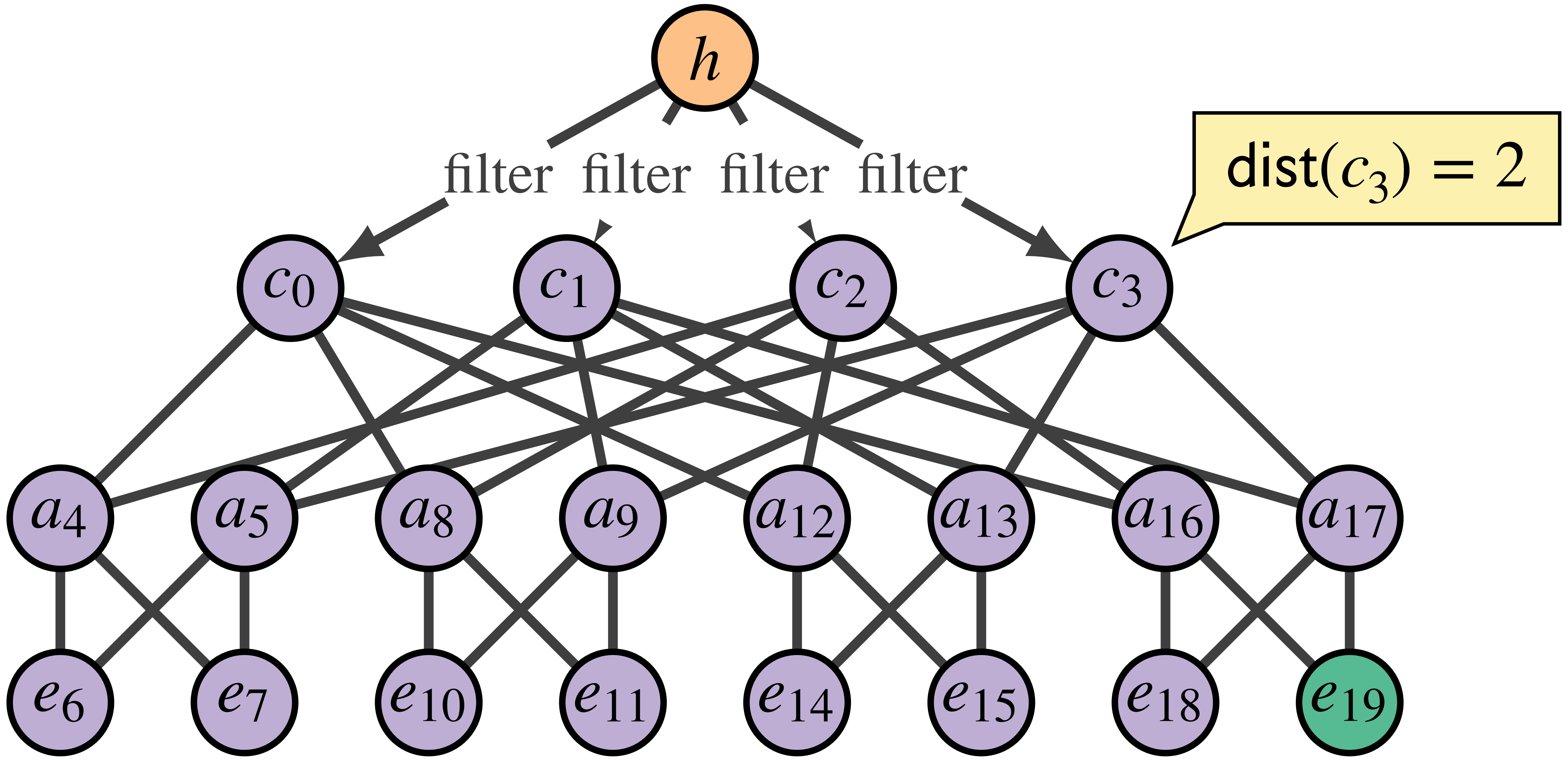
All nodes' interfaces are parameterized by their distance $\mathsf{dist}(v)$ from e_{19}

Nodes are **eventually** "internally reachable"

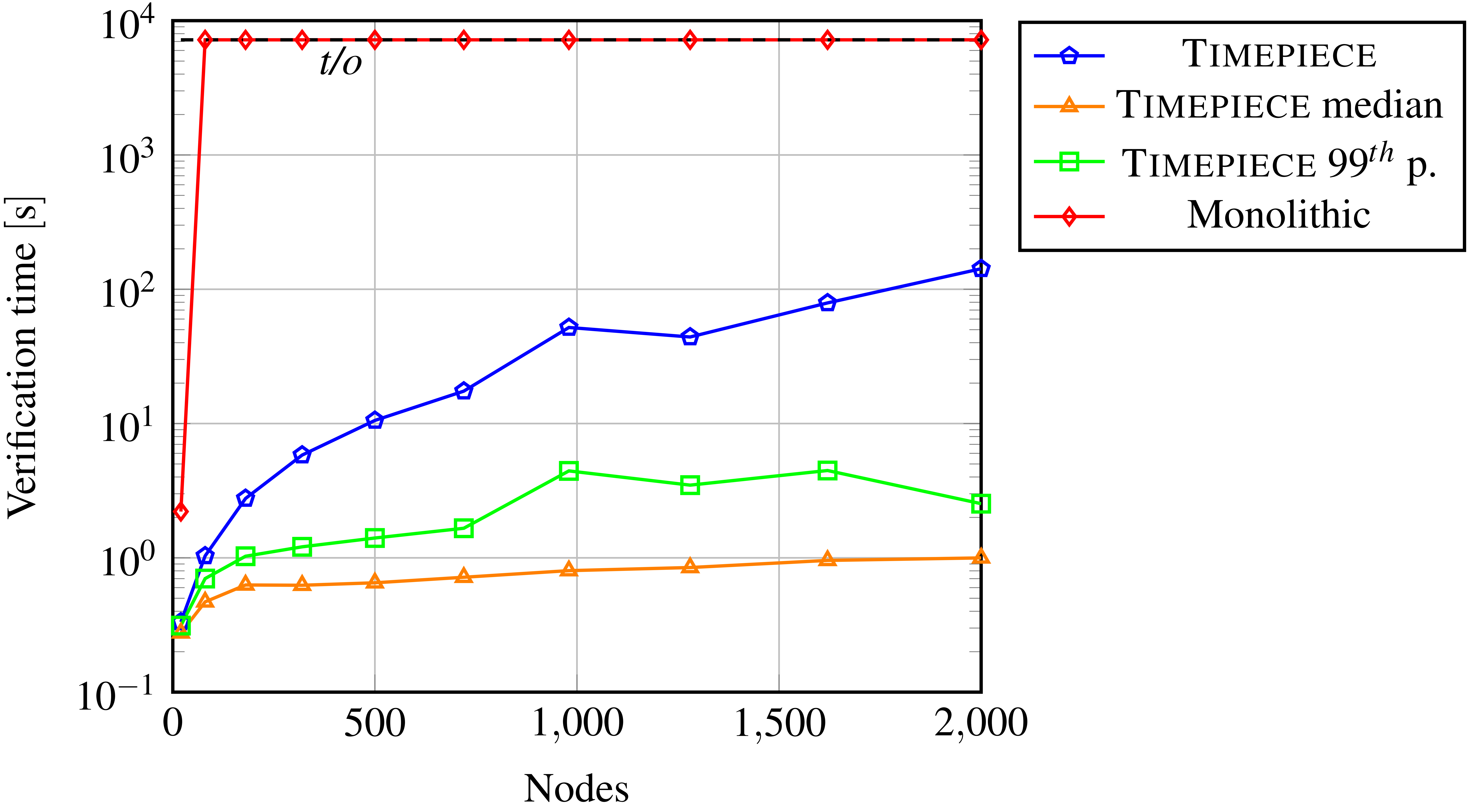
$\mathsf{true}\ \mathcal{U}^{\mathsf{dist}(v)}\ s.\mathsf{prefix} = p \land \neg s.\mathsf{tag}$

Nodes **never** use hijacking routes

$\mathcal{G}(s.\mathsf{prefix} = p \rightarrow \neg s.\mathsf{tag})$

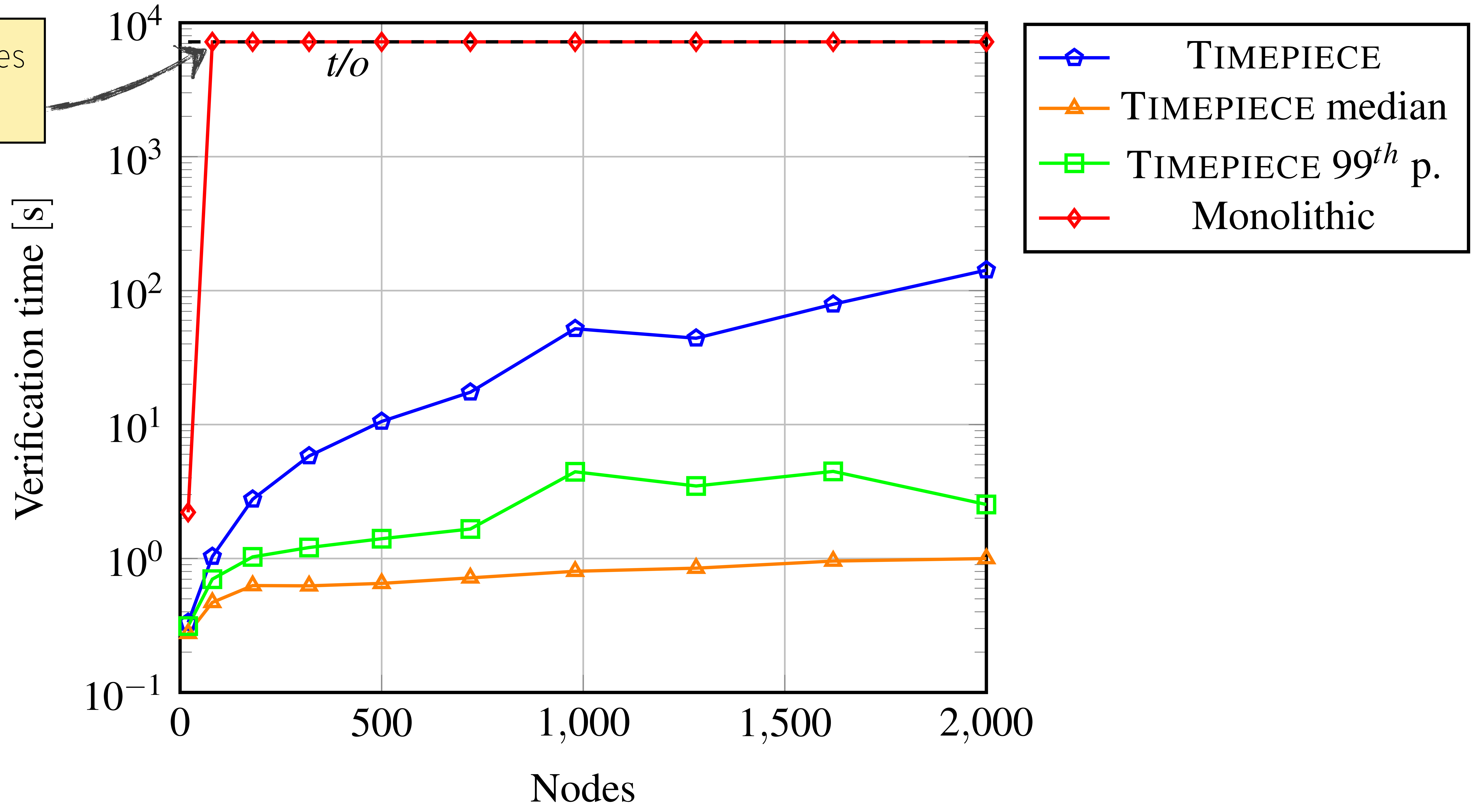


Fat-tree Hijack Filtering



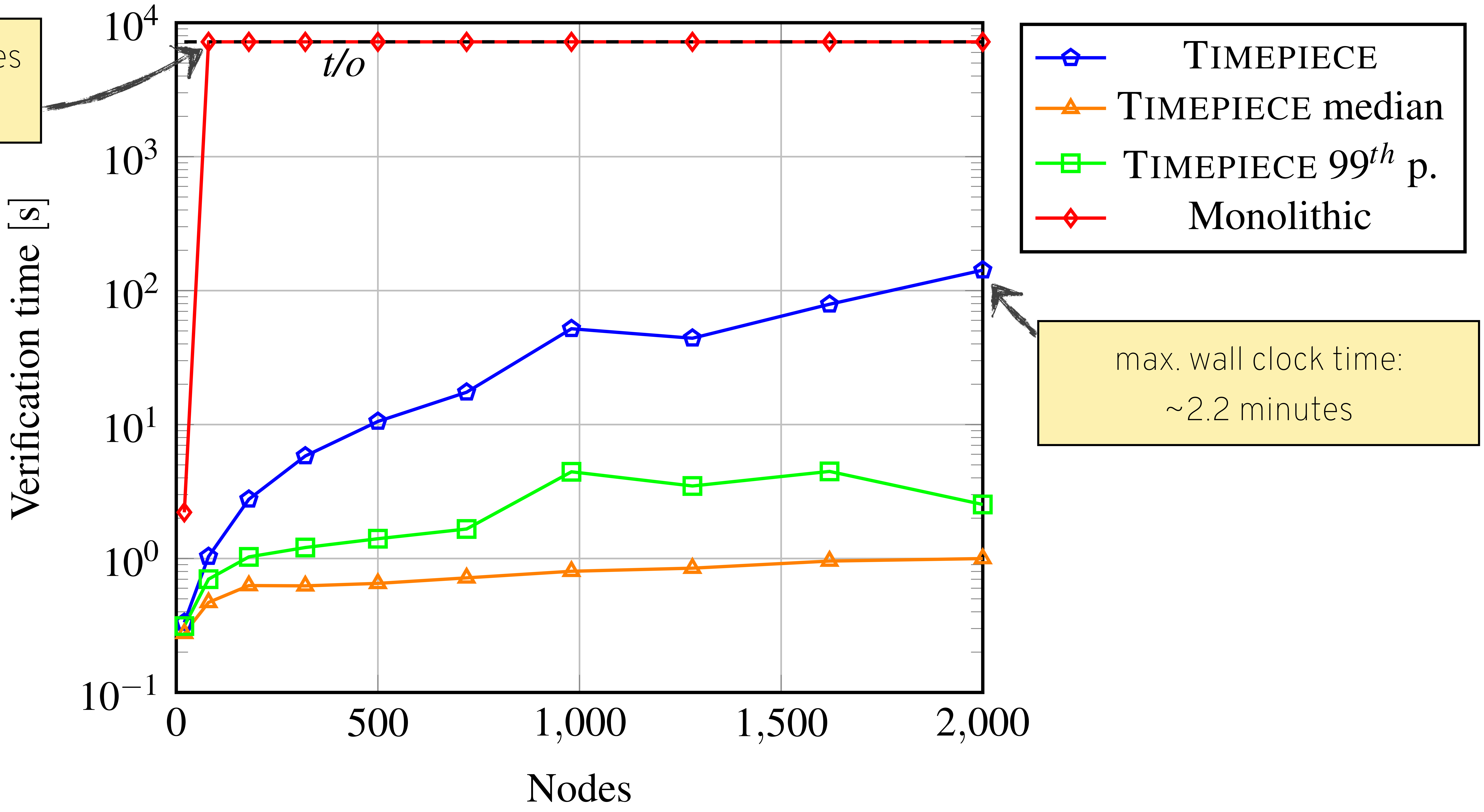
Fat-tree Hijack Filtering

Monolithic verification times out (>2h) at 80 nodes



Fat-tree Hijack Filtering

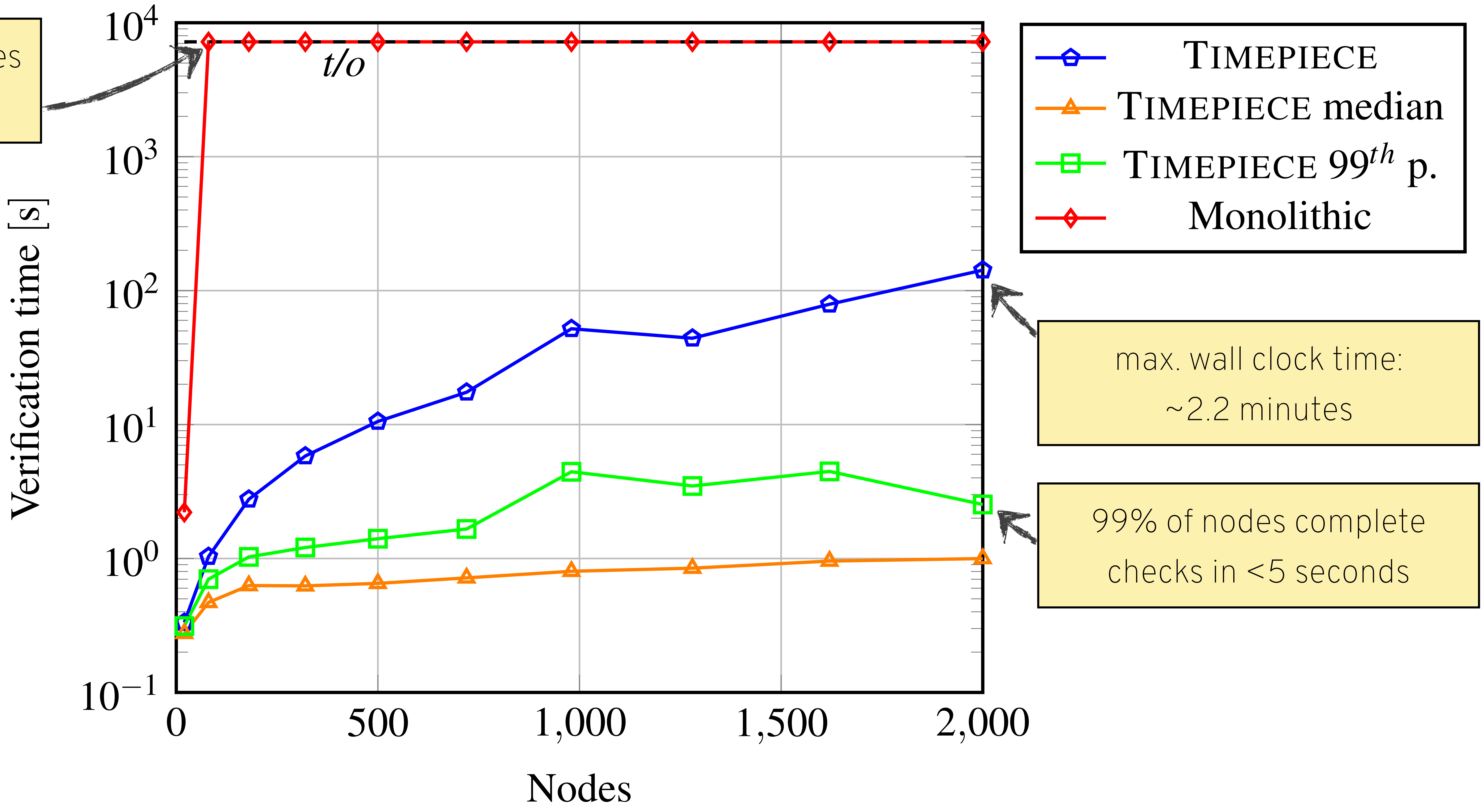
Monolithic verification times out (>2h) at 80 nodes



max. wall clock time: ~2.2 minutes

Fat-tree Hijack Filtering

Monolithic verification times out (>2h) at 80 nodes



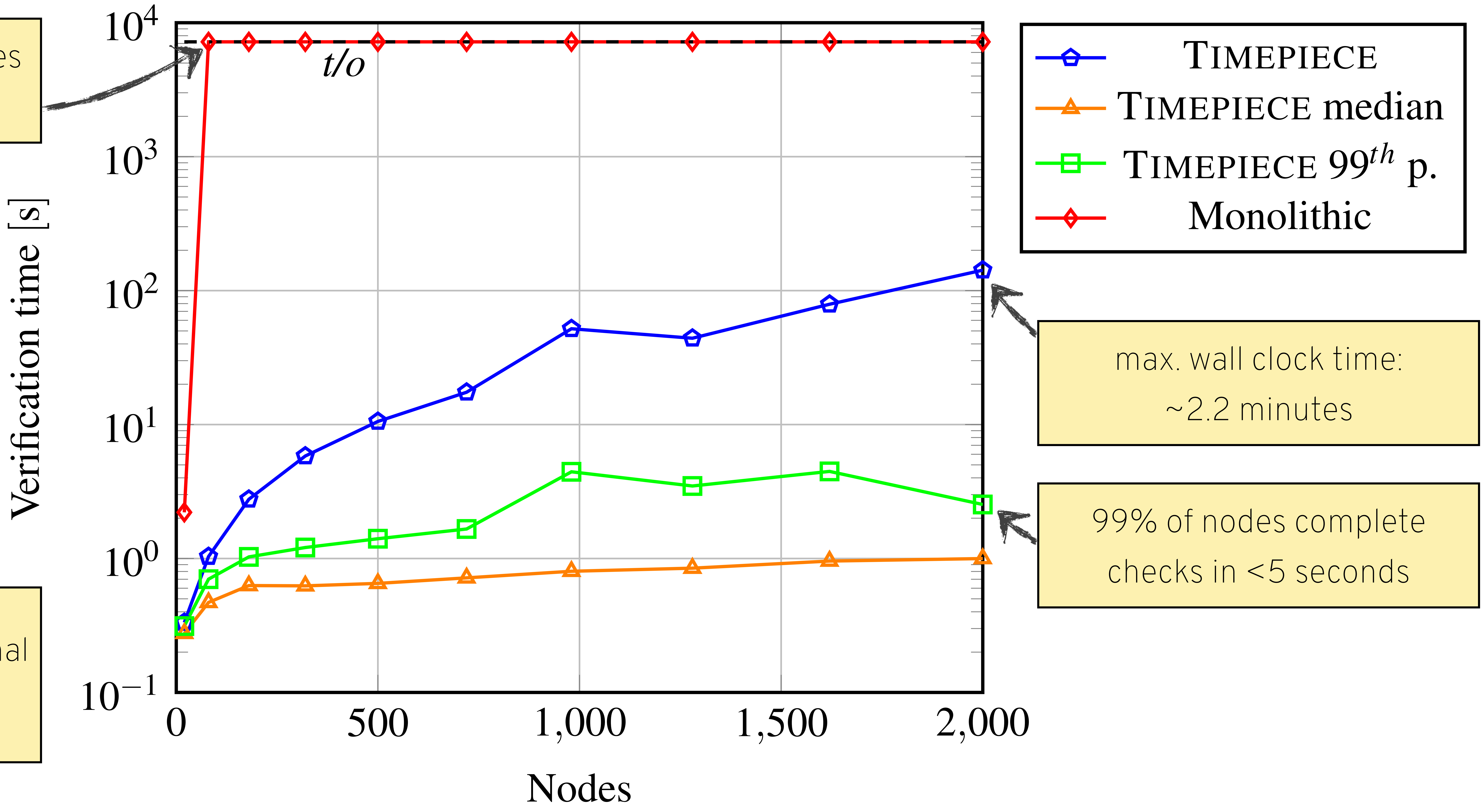
max. wall clock time: ~2.2 minutes

99% of nodes complete checks in <5 seconds

Fat-tree Hijack Filtering

Monolithic verification times out (>2h) at 80 nodes

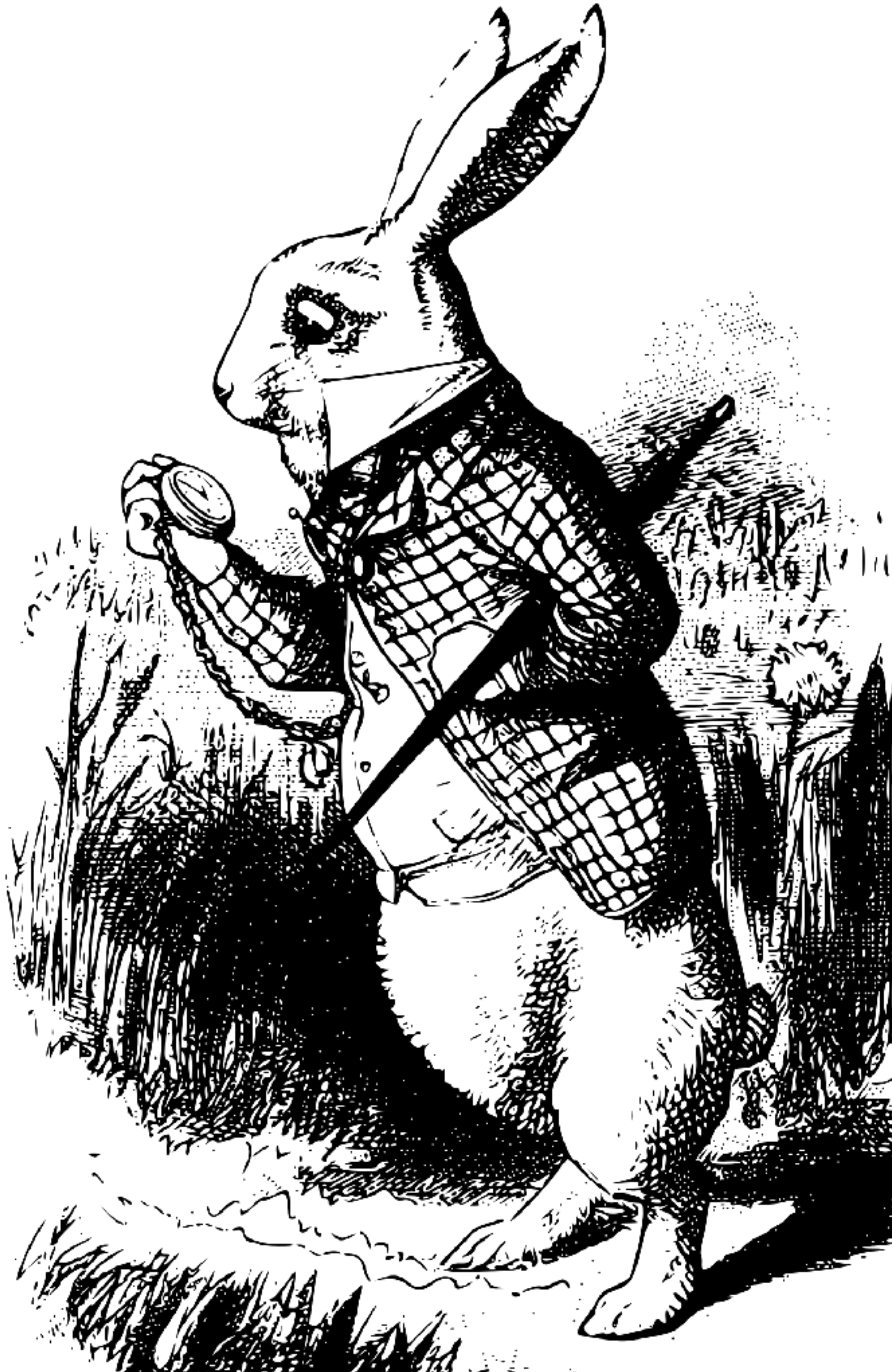
verification time proportional to node's degree



max. wall clock time: ~2.2 minutes

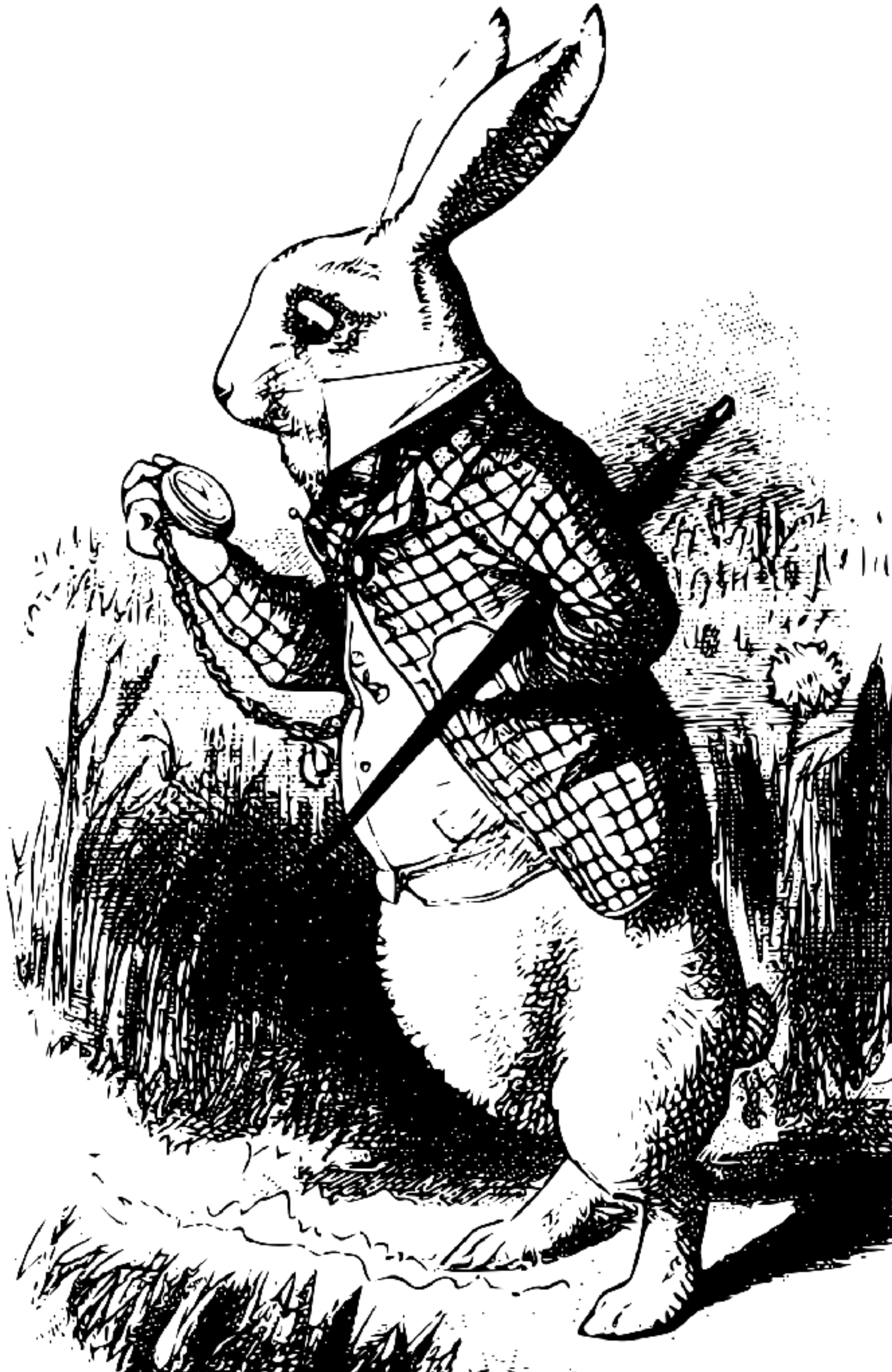
99% of nodes complete checks in <5 seconds

Takeaways



Takeaways

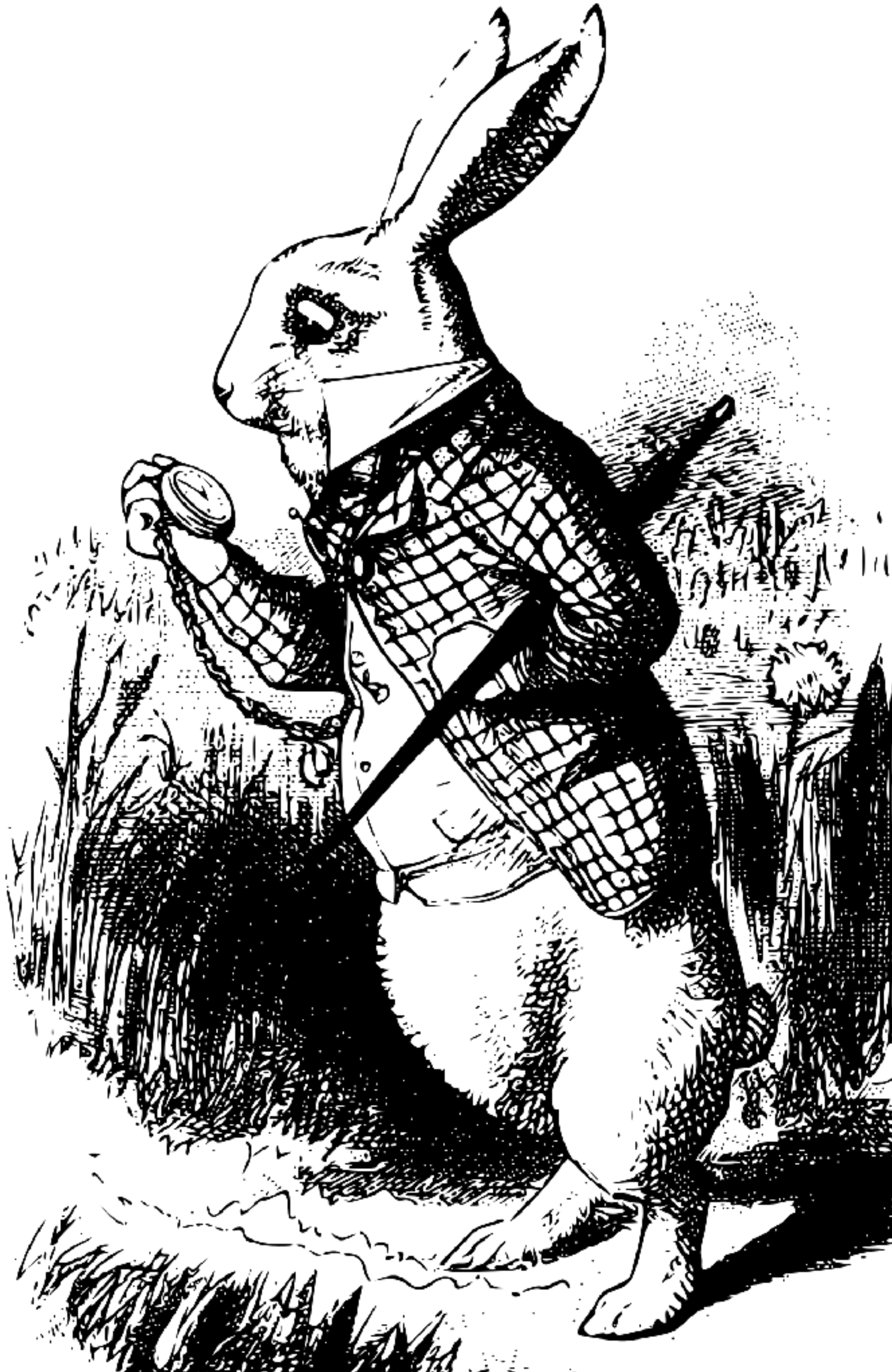
Big, complex control planes need modular tools



Takeaways

Big, complex control planes need modular tools

Temporal invariants provide a **correct basis for modular verification**

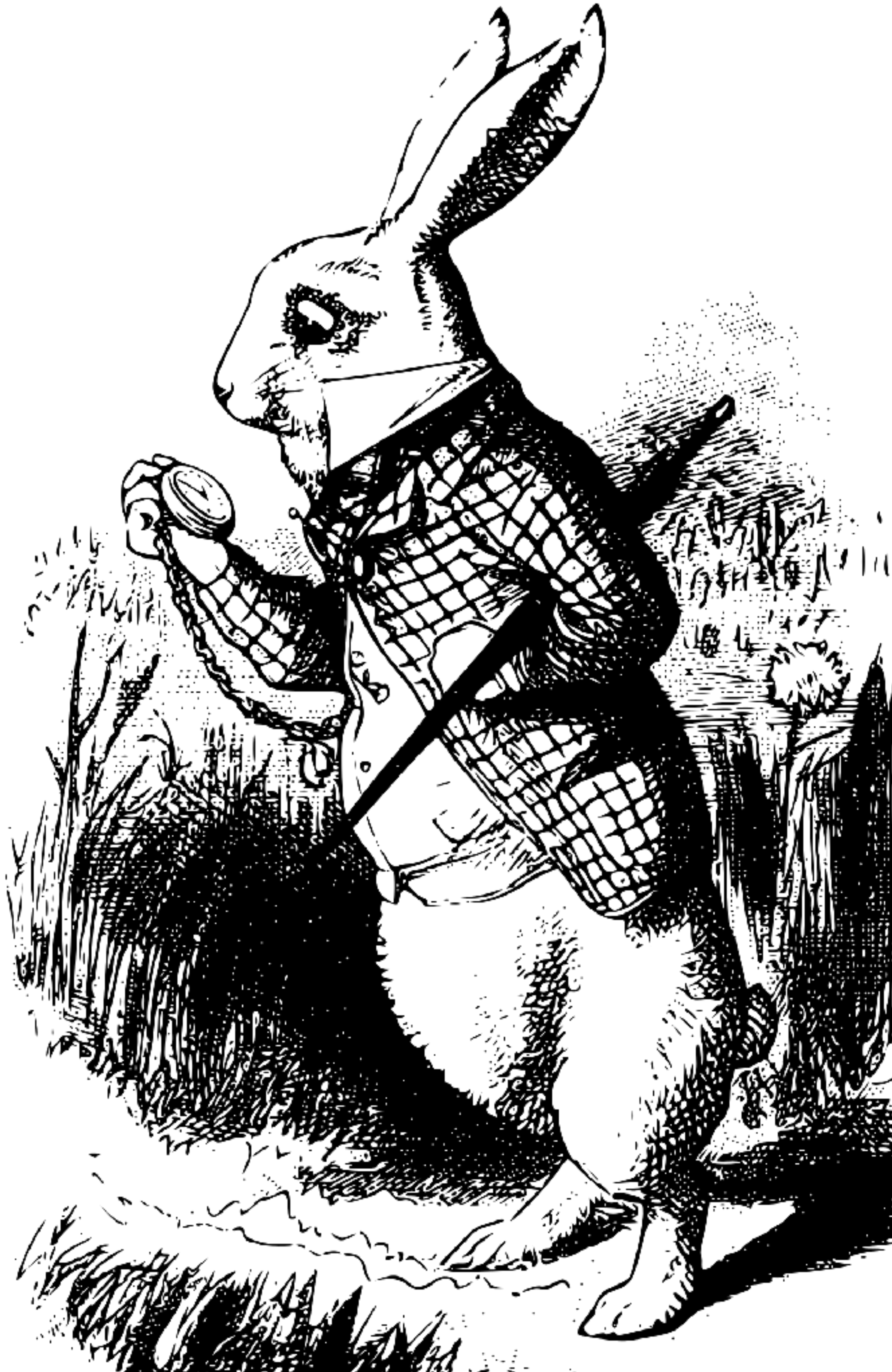


Takeaways

Big, complex control planes need modular tools

Temporal invariants provide a **correct basis for modular verification**

Scale to thousands of nodes & complex policies



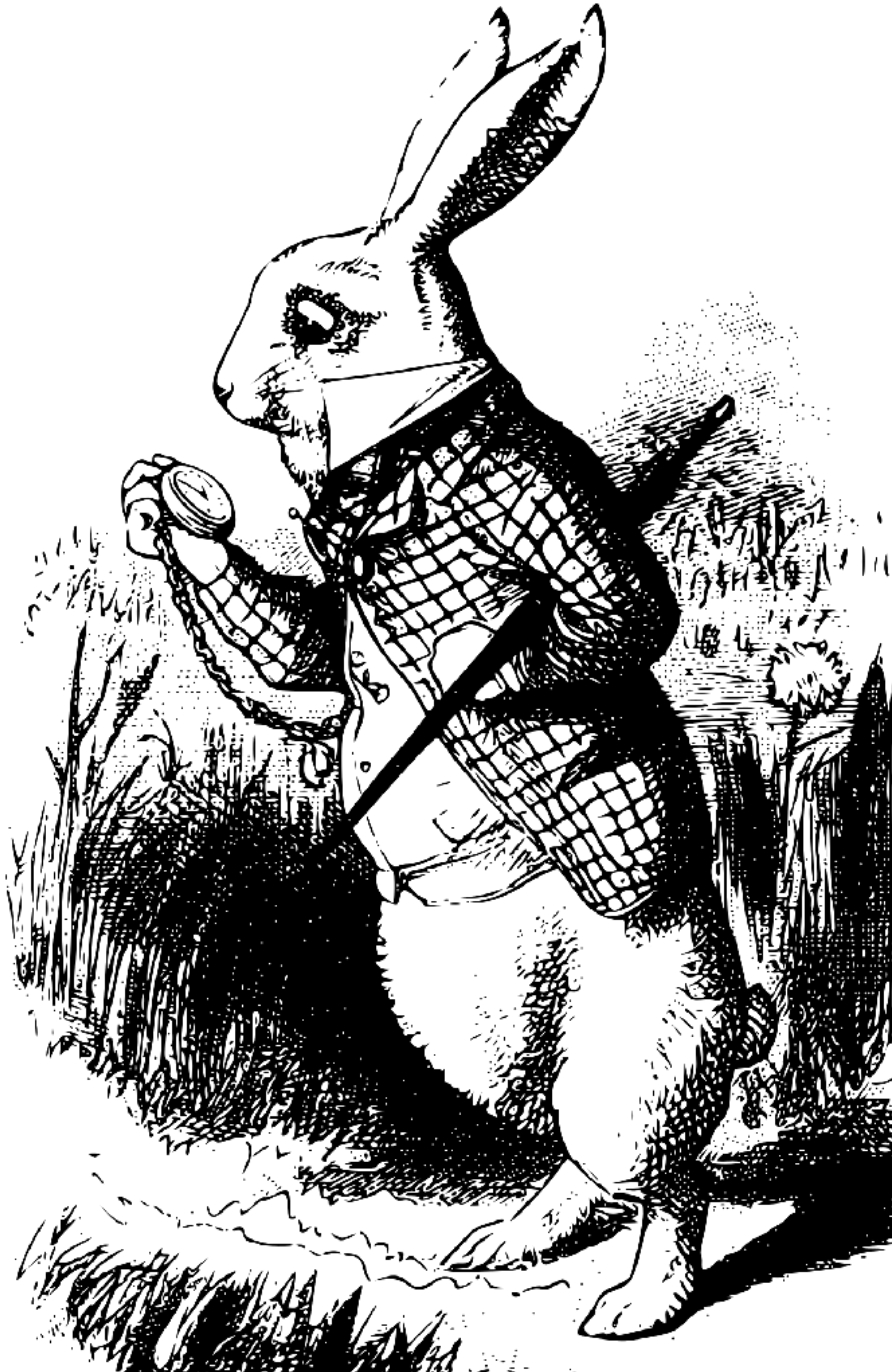
Takeaways

Big, complex control planes need modular tools

Temporal invariants provide a **correct basis for modular verification**

Scale to thousands of nodes & complex policies

Read the paper to learn more!



Thank You!

I'm looking for a job!

cs.princeton.edu/~tthijm

Our paper

Tim Alberdingk Thijm
Princeton

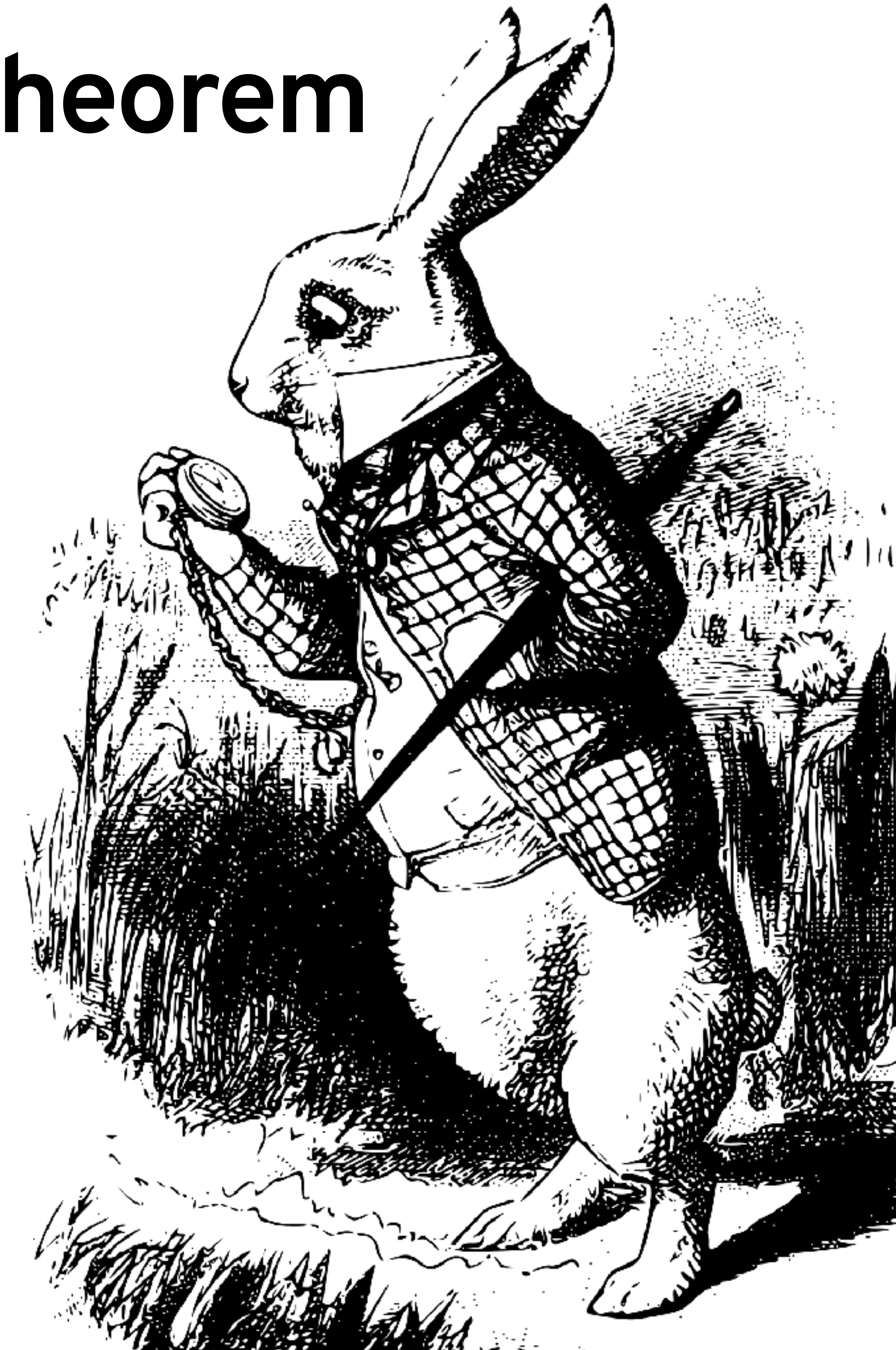
Ryan Beckett
Microsoft Research

Aarti Gupta
Princeton

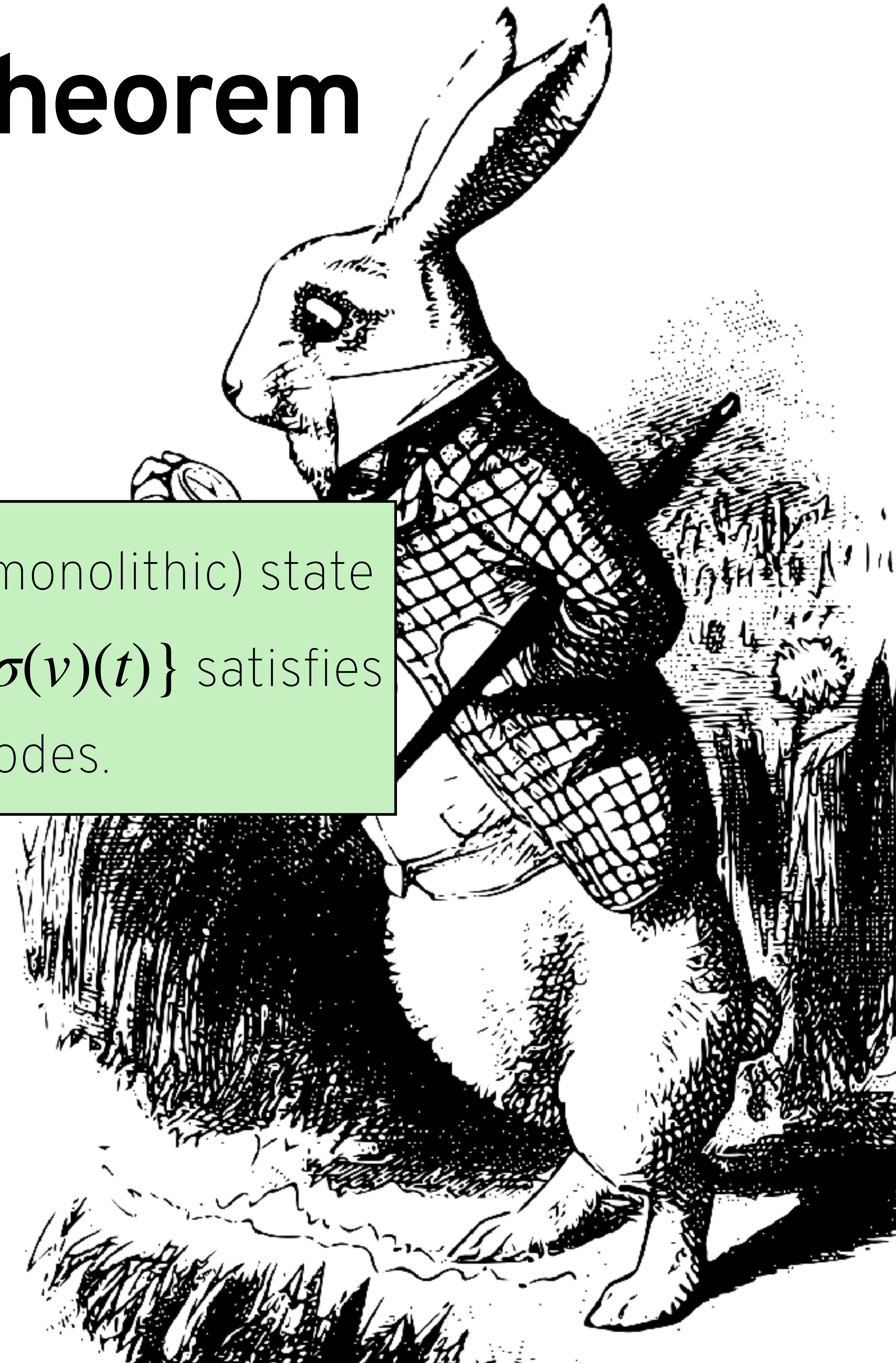
Dave Walker
Princeton

Extra slides

Closed Completeness Theorem

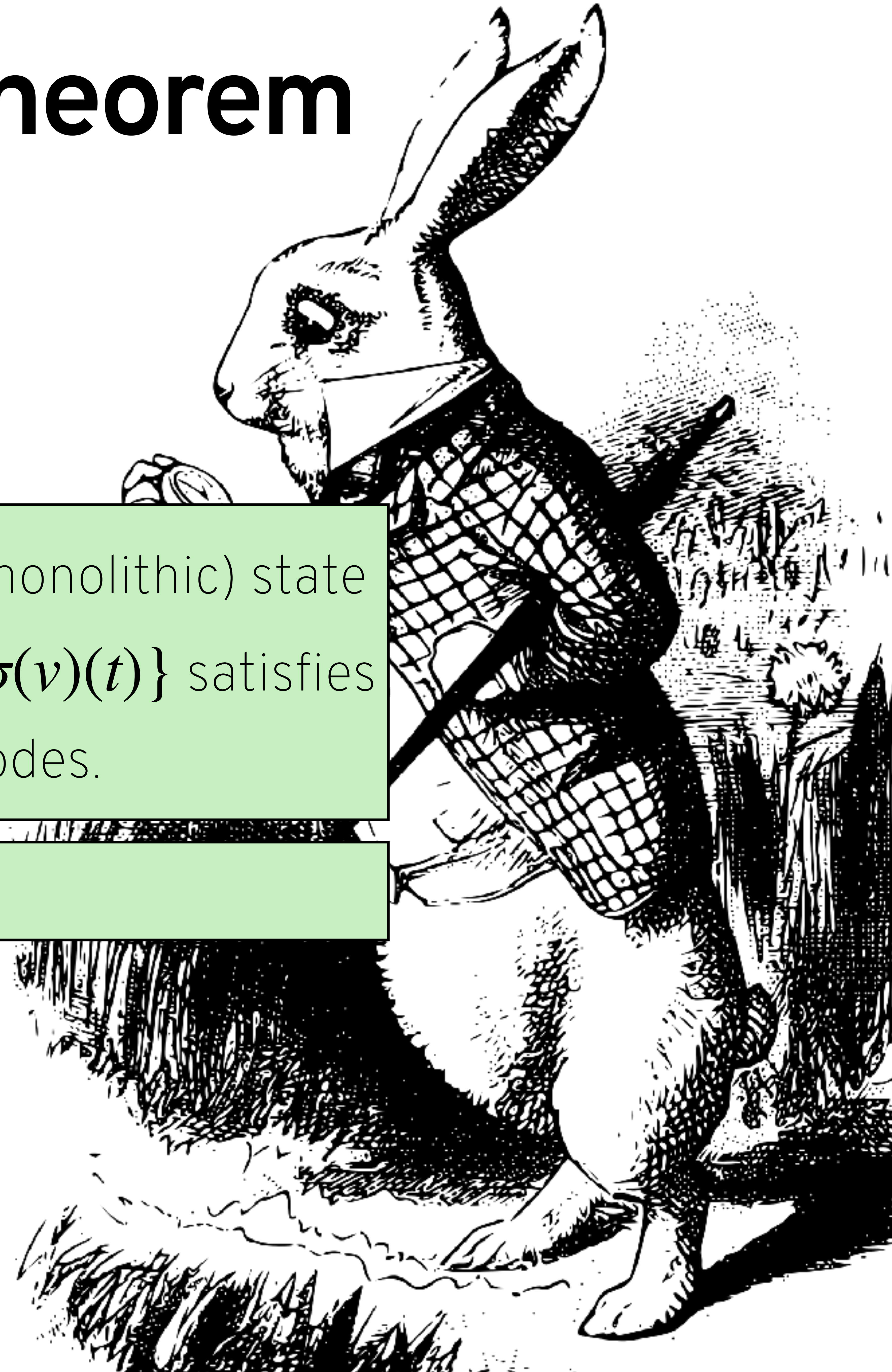


Closed Completeness Theorem



Starting from fixed initial routes, if $\sigma(v)(t)$ is the (monolithic) state of node v at time t, then the interface $A(v)(t) = \{\sigma(v)(t)\}$ satisfies the base and inductive checks for all nodes.

Closed Completeness Theorem



Starting from fixed initial routes, if $\sigma(v)(t)$ is the (monolithic) state of node v at time t, then the interface $A(v)(t) = \{\sigma(v)(t)\}$ satisfies the base and inductive checks for all nodes.

Proof by induction on time.

Evaluation

Evaluation

Benchmark

Reachability

Path length

Valley freedom

Hijack filtering

No transit

Evaluation

Benchmark	Nodes
Reachability	20-2000
Path length	20-2000
Valley freedom	20-2000
Hijack filtering	20-2000
No transit	263

Evaluation

Benchmark	Nodes	Network LoC
Reachability	20-2000	81
Path length	20-2000	88
Valley freedom	20-2000	89
Hijack filtering	20-2000	146
No transit	263	88 (+102,753)

Evaluation

Benchmark	Nodes	Network LoC	Annotation LoC
Reachability	20-2000	81	3
Path length	20-2000	88	7
Valley freedom	20-2000	89	12
Hijack filtering	20-2000	146	21
No transit	263	88 (+102,753)	5

Evaluation

Benchmark	Nodes	Network LoC	Annotation LoC	Monolithic hits 2h timeout?
Reachability	20-2000	81	3	No (fixed dest.) 80 nodes (symbolic dest.)
Path length	20-2000	88	7	80 nodes (fixed dest.) 20 nodes (symbolic dest.)
Valley freedom	20-2000	89	12	180 nodes
Hijack filtering	20-2000	146	21	80 nodes (fixed dest.) 20 nodes (symbolic dest.)
No transit	263	88 (+102,753)	5	Yes

Evaluation

Benchmark	Nodes	Network LoC	Annotation LoC	Monolithic hits 2h timeout?	Modular verification time
Reachability	20-2000	81	3	No (fixed dest.) 80 nodes (symbolic dest.)	28s (fixed 2000 nodes) 336s (symbolic 2000 nodes)
Path length	20-2000	88	7	80 nodes (fixed dest.) 20 nodes (symbolic dest.)	1204s (fixed 2000 nodes) 3953s (symbolic 2000 nodes)
Valley freedom	20-2000	89	12	180 nodes	398s (fixed 2000 nodes) 3506s (symbolic 1280 nodes)
Hijack filtering	20-2000	146	21	80 nodes (fixed dest.) 20 nodes (symbolic dest.)	142s (2000 nodes) 2196s (symbolic 2000 nodes)
No transit	263	88 (+102,753)	5	Yes	38s

Related Work

Related Work

Satisfiability Modulo Theories (SMT)-based verification

Scalable Verification of Border Gateway Protocol Configurations with an SMT Solver

Konstantin Weitz Doug Woos Emina Torlak
Michael D. Ernst Arvind Krishnamurthy Zachary Tatlock
University of Washington, USA
{weitzkon, dwoos, emina, mernst, arvind, ztatlock}@cs.washington.edu

A General Approach to Network Configuration Verification

Ryan Beckett, Princeton University
Aarti Gupta, Princeton University
Ratul Mahajan, Microsoft Research & Intentionet
David Walker, Princeton University

Related Work

Satisfiability Modulo Theories (SMT)-based verification

Scalable Verification of Border Gateway Protocol Configurations with an SMT Solver

Konstantin Weitz Doug Woos Emina Torlak
Michael D. Ernst Arvind Krishnamurthy Zachary Tatlock
University of Washington, USA
{weitzkon, dwoos, emina, mernst, arvind, ztatlock}@cs.washington.edu

A General Approach to Network Configuration Verification

Ryan Beckett, Princeton University
Aarti Gupta, Princeton University
Ratul Mahajan, Microsoft Research & Intentionet
David Walker, Princeton University

simulation-based verification

Plankton: Scalable network configuration verification through model checking

Santhosh Prabhu, Kuan-Yen Chou, Ali Kheradmand, P. Brighten Godfrey, Matthew Caesar
University of Illinois at Urbana-Champaign

Tiramisu: Fast Multilayer Network Verification

Anubhavnidhi Abhashkumar*, Aaron Gember-Jacobson†, Aditya Akella*
University of Wisconsin - Madison, Colgate University†*

Related Work

Satisfiability Modulo Theories (SMT)-based verification

Scalable Verification of Border Gateway Protocol Configurations with an SMT Solver

Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, Zachary Tatlock
University of Washington, USA
{weitzkon, dwoos, emina, mernst, arvind, ztatlock}@cs.washington.edu

A General Approach to Network Configuration Verification

Ryan Beckett, Princeton University
Aarti Gupta, Princeton University
Ratul Mahajan, Microsoft Research & Intentionet
David Walker, Princeton University

simulation-based verification

Plankton: Scalable network configuration verification through model checking

Santhosh Prabhu, Kuan-Yen Chou, Ali Kheradmand, P. Brighten Godfrey, Matthew Caesar
University of Illinois at Urbana-Champaign

Tiramisu: Fast Multilayer Network Verification

Anubhavnidhi Abhashkumar*, Aaron Gember-Jacobson†, Aditya Akella*
University of Wisconsin - Madison, Colgate University†*

modular SMT-based verification

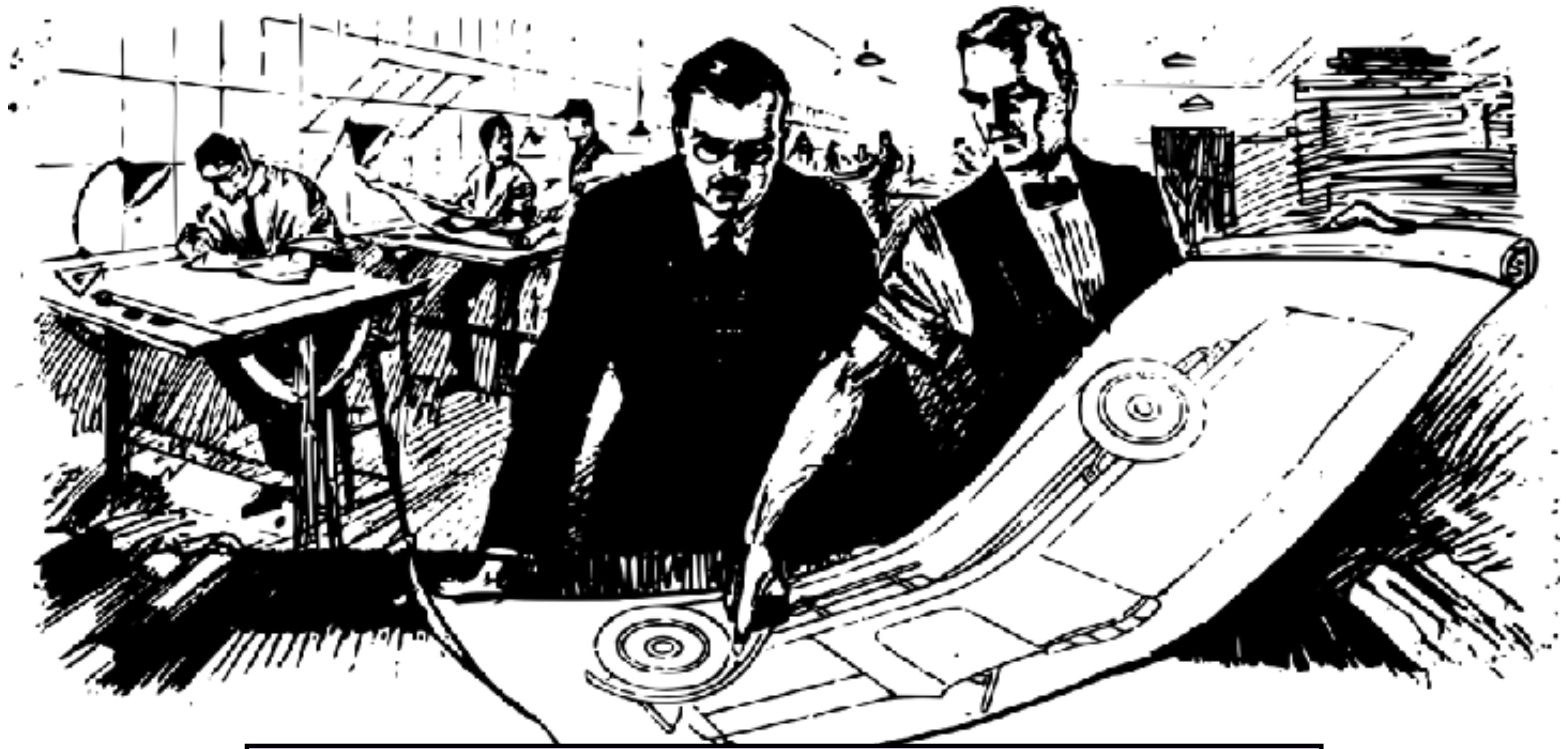
LIGHTYEAR: Using Modularity to Scale BGP Control Plane Verification

Alan Tang, *University of California, Los Angeles*
Ryan Beckett, *Microsoft*
Karthick Jayaraman, *Microsoft*
Todd Millstein, *UCLA / Intentionet*
George Varghese, *UCLA*

Kirigami, the Verifiable Art of Network Cutting

Timothy Alberdingk Thijm, *Princeton University*, Princeton, USA, tthijm@cs.princeton.edu
Ryan Beckett, *Microsoft Research*, Redmond, USA, ryan.beckett@microsoft.com
Aarti Gupta, *Princeton University*, Princeton, USA, aartig@cs.princeton.edu
David Walker, *Princeton University*, Princeton, USA, dpw@cs.princeton.edu

Challenges



finding the correct invariants

synchronous network semantics