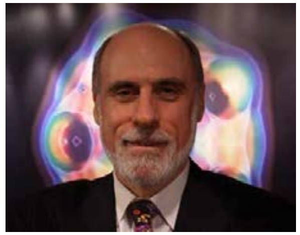



FOR PUBLIC CLOUDS, PRIVATE CLOUDS, ENTERPRISE NETWORKS, ISPS, . . .

# NETWORK DESIGN AUTOMATION: WHEN CLARKE MEETS CERF

George Varghese
UCLA
(with collaborators from MSR, Stanford, UCLA)

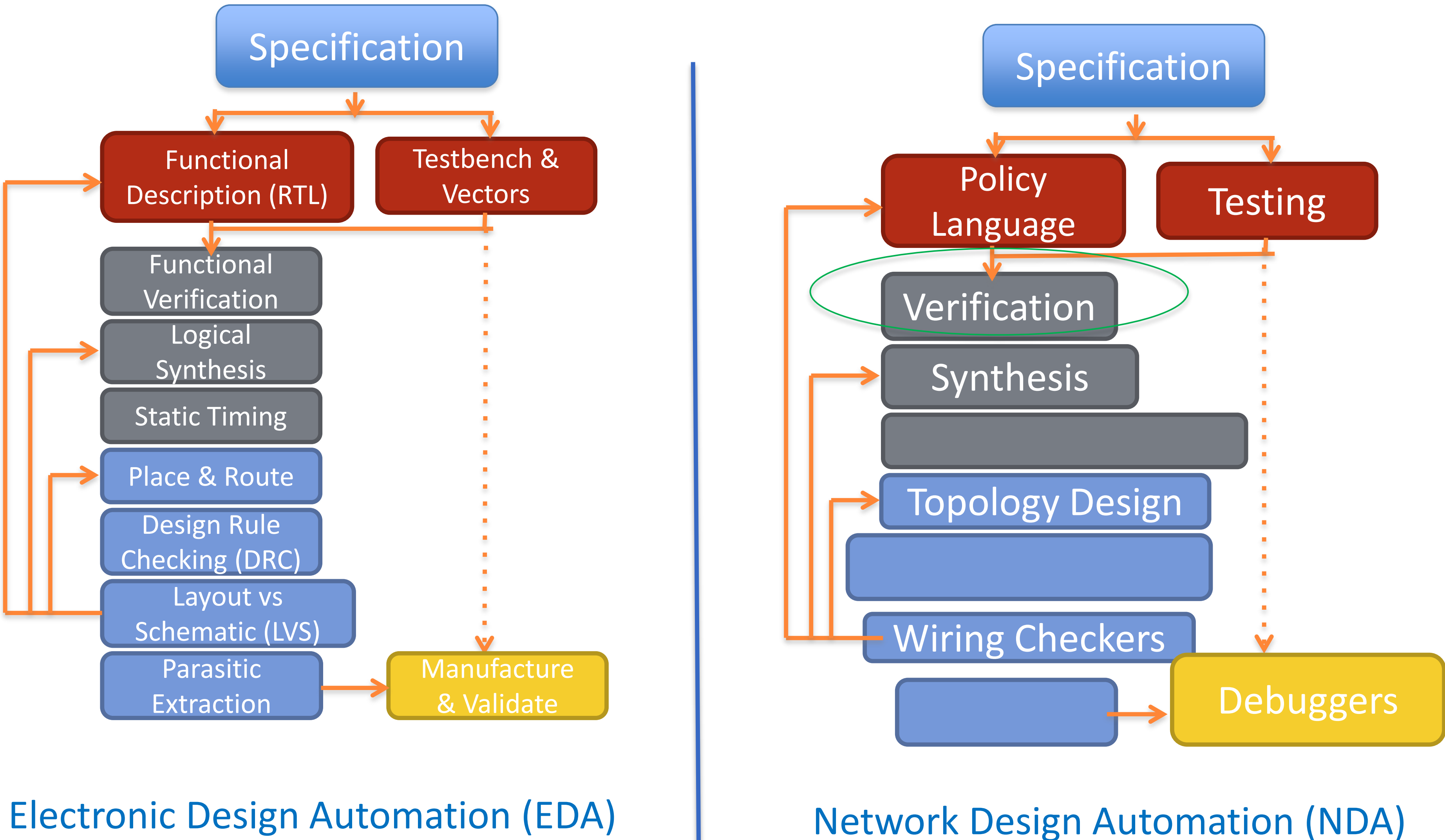
IFIF Keynote & NSF Workshop, 2020

NDA

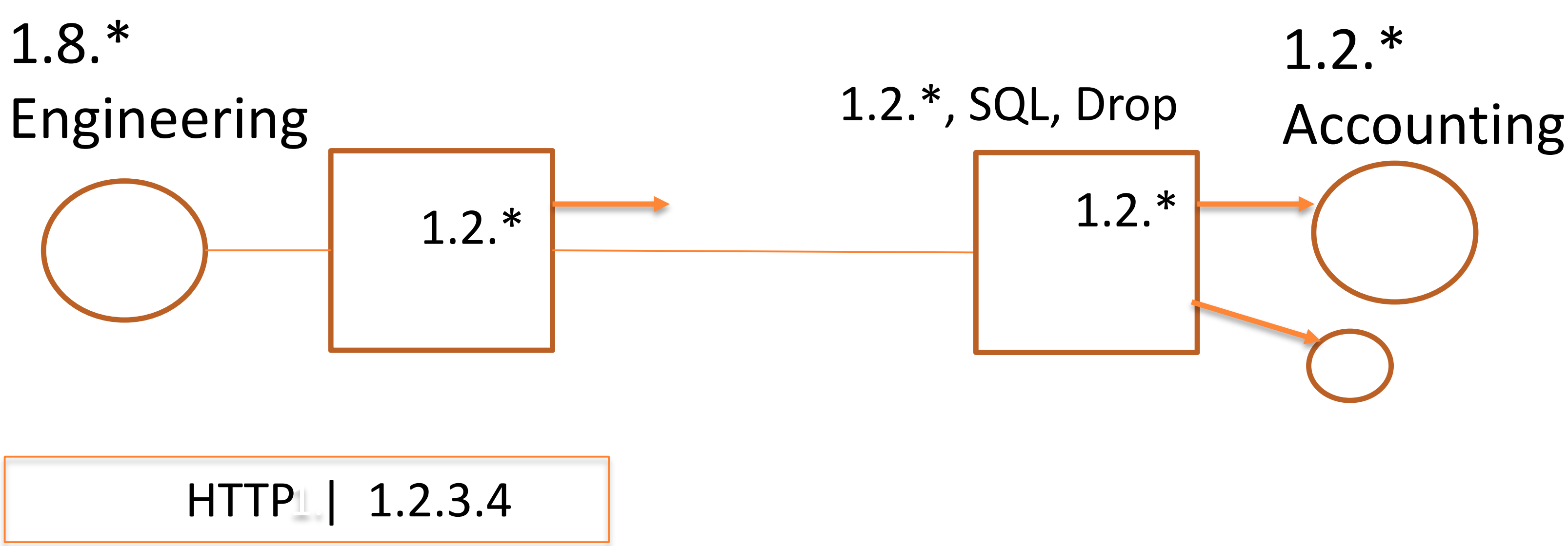
# The Tides of EDA

**Alberto Sangiovanni-Vincentelli**
University of California at Berkeley

# This talk: from EDA to NDA



# Model and Terminology




- Routers, links, interfaces
- Packets, headers
- Prefix match rules, manually placed Access Control (ACL) rules in router configuration files. Easy to make errors

# Problem with Networks today


- Manual Configurations: Managers override shortest paths for security, load balancing, and economics
- Problem: Manually programming *individual* routers to implement *global* policy leads to cloud failures

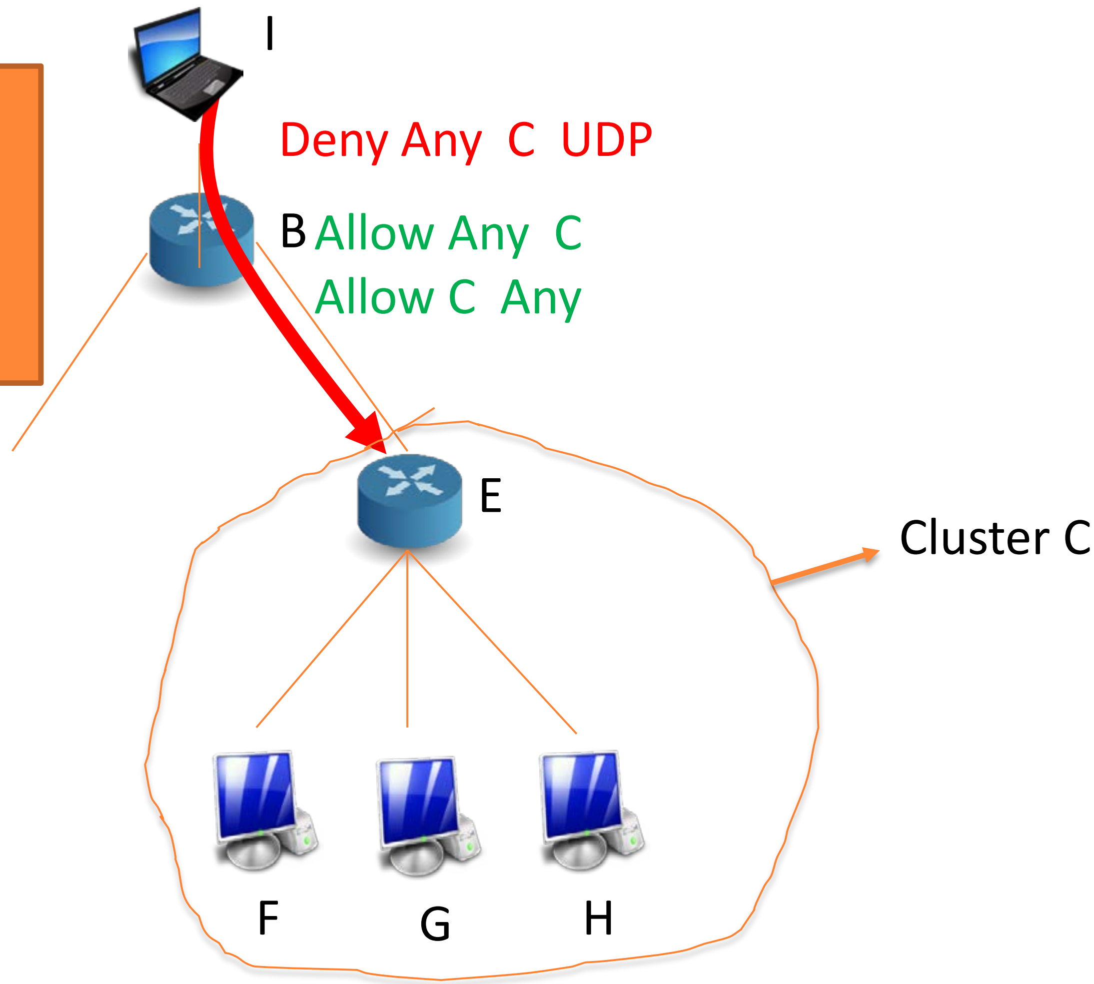
# Manual Traffic “steering knobs”

- Data forwarding/Data Plane:
  - Access Control Lists (predicates on headers)
  - VLANs (a way to virtualize networks)
- Routing/ Control Plane:
  - Communities: equivalence classes on routes via a tag
  - Static routes: a manager supplied route

Managers use many more knobs for isolation, economics

# Why manual reasoning is hard

POLICY
- Internet and Compute can communicate
- Internet cannot send to controllers




Name Service Queries are now blocked!

# Why automated reasoning is imperative

- Challenges: 2^{100} possible headers to test!
  - Scale: devices (1000s), rules (millions), ACL limits (< 700)
  - Diversity: 10 different vendors, > 10 types of headers
  - Rapid changes (new clusters, policies, attacks)
- Severity: (2012 NANOG Network Operator Survey):
  - 35% have 25 tickets per month, take > 1 hour to resolve
  - Welsh: vast majority of Google "production failures" due to "bugs in configuration settings"
  - Amazon, GoDaddy, United Airlines: high profile failures

As we migrate to services ($100B public cloud market), network failure a debilitating cost.

# Simple questions hard to answer today

- Which packets from A can reach B?
- Is Group X provably isolated from Group Y?
- Why is my backbone utilization poor?

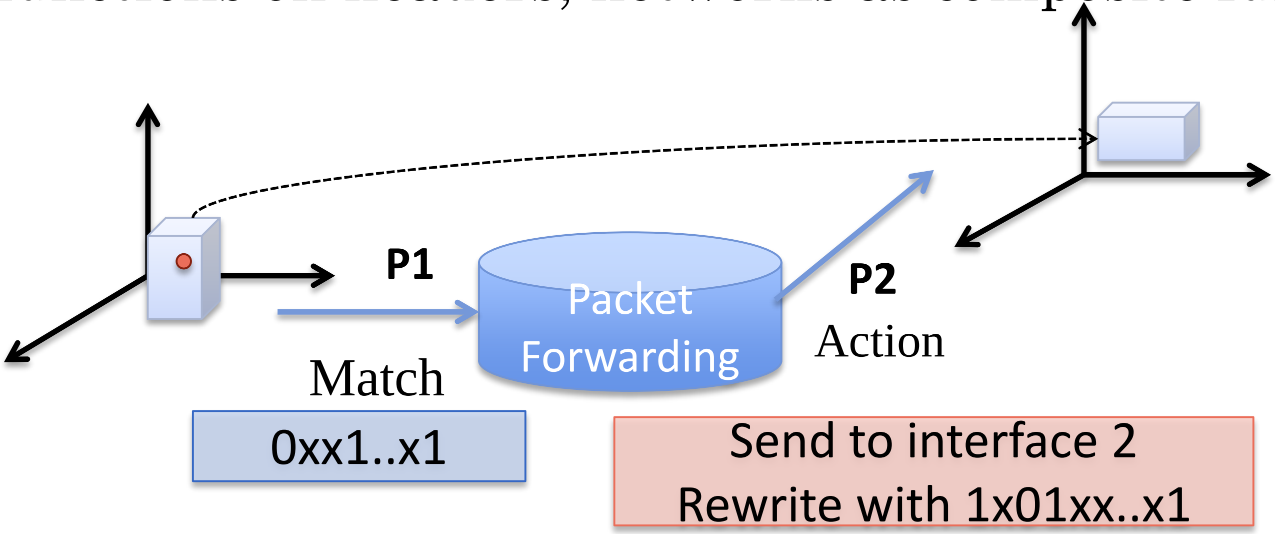
NEED BOTTOM UP ANALYSIS OF EXISTING SYSTEMS

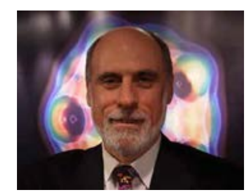
Formal methods have been used to verify (check *all* cases) large chip designs and programs.

This talk: can we use formal methods across *all* headers, & inputs for large clouds?

# Approach: Treat Networks as Programs

- Model header as point in header space, routers as functions on headers, networks as composite functions



CAN NOW ASK WHAT THE EQUIVALENT OF *ANY* PROGRAM ANALYSIS TOOL IS FOR NETWORKS

# Problems addressed/Outline

- Part 1: Classical verification tools do not scale
    - Scaling via Network Specific Symmetries (POPL 16)
- Part 2: Lack of specifications
    - Finding Bugs without Specifications (NSDI 2020)
- Part 3: A vision for Network Design Automation (NDA)

# Scaling Network Verification

(Plotkin, Bjorner, Lopes, Rybalchenko, Varghese, POPL 2016)

exploiting network specific *symmetries*

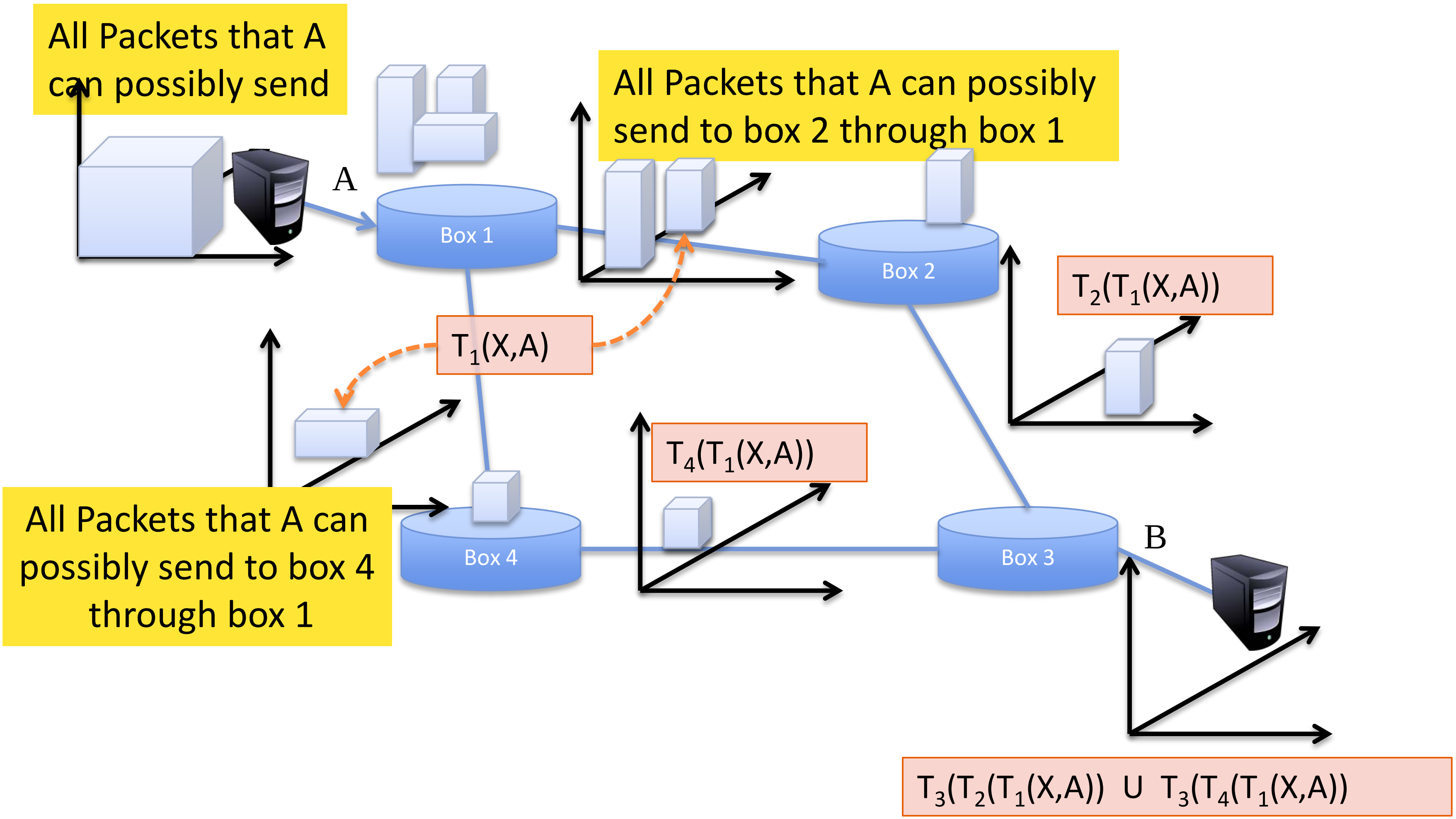
# Formal Network Model [HSA 12]

- 1 - Model sets of packets based on relevant header bits, as subsets of a $\{0,1,*\}^L$ space – the Header Space
- 2 – Define union, intersection on Header Spaces
- 3 – Abstract networking boxes (Cisco routers, Juniper Firewalls) as transfer functions on sets of headers
- 4– Compute packets that can reach across a path as composition of Transfer Functions of routers on path
- 5. Find all packets that reach between every pair of nodes and check against reachability specification

All Network boxes modelled as a Transfer Function:

$$T : (h, p) \rightarrow \{(h_1, p_1), \ldots, (h_n, p_n)\}$$

# Computing Reachability [HSA 12]

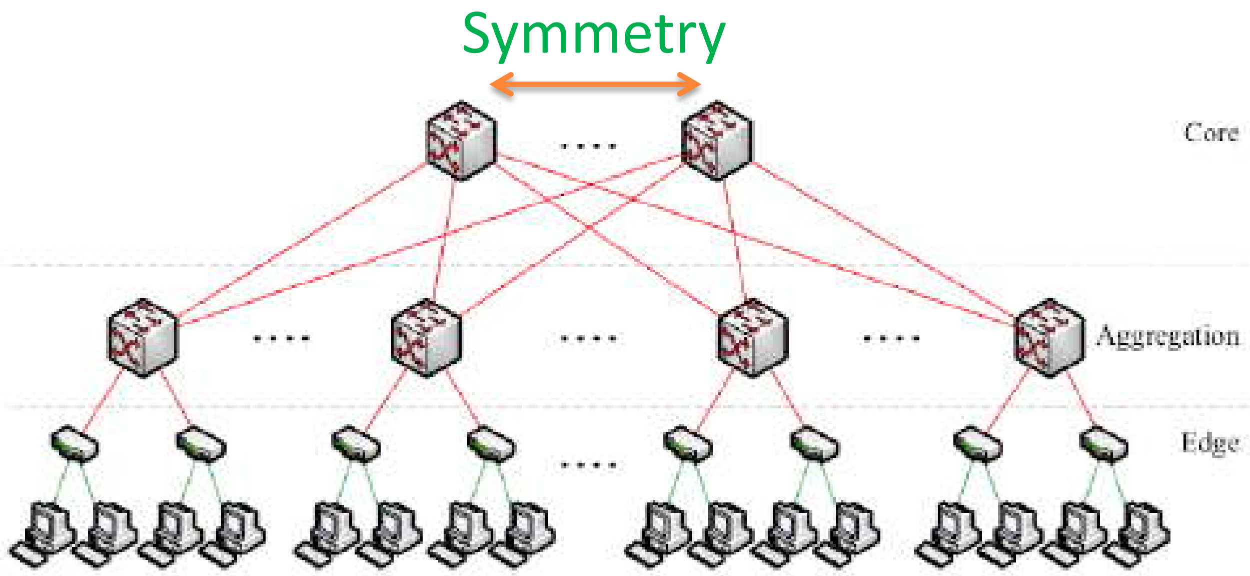



COMPLEXITY DEPENDS ON HEADERS, PATHS, NUMBER OF RULES

# Unfortunately, in practice . . .

- Header space equivalencing: 1 query in < 1 sec. Uses ternary simulation! Major improvement over SAT solvers and model checkers.
- But real data centers: 100,000 hosts, 1 million rules, 1000s of routers, 100 bits of header
- So N^2 pairs takes 5 days to verify all specs.

# Exploit Design Regularities to scale?




Common data center interconnect topology. Host to switch links are GigE and links between switches are 10 GigE.

Can exploit regularities in rules and topology (not headers):

- Reduce fat tree to "*thin tree*"; verify reachability cheaply in thin tree.
- How can we make this idea precise?

# *Factored* symmetries

- (Emerson-Sistla): Symmetry on *state* space

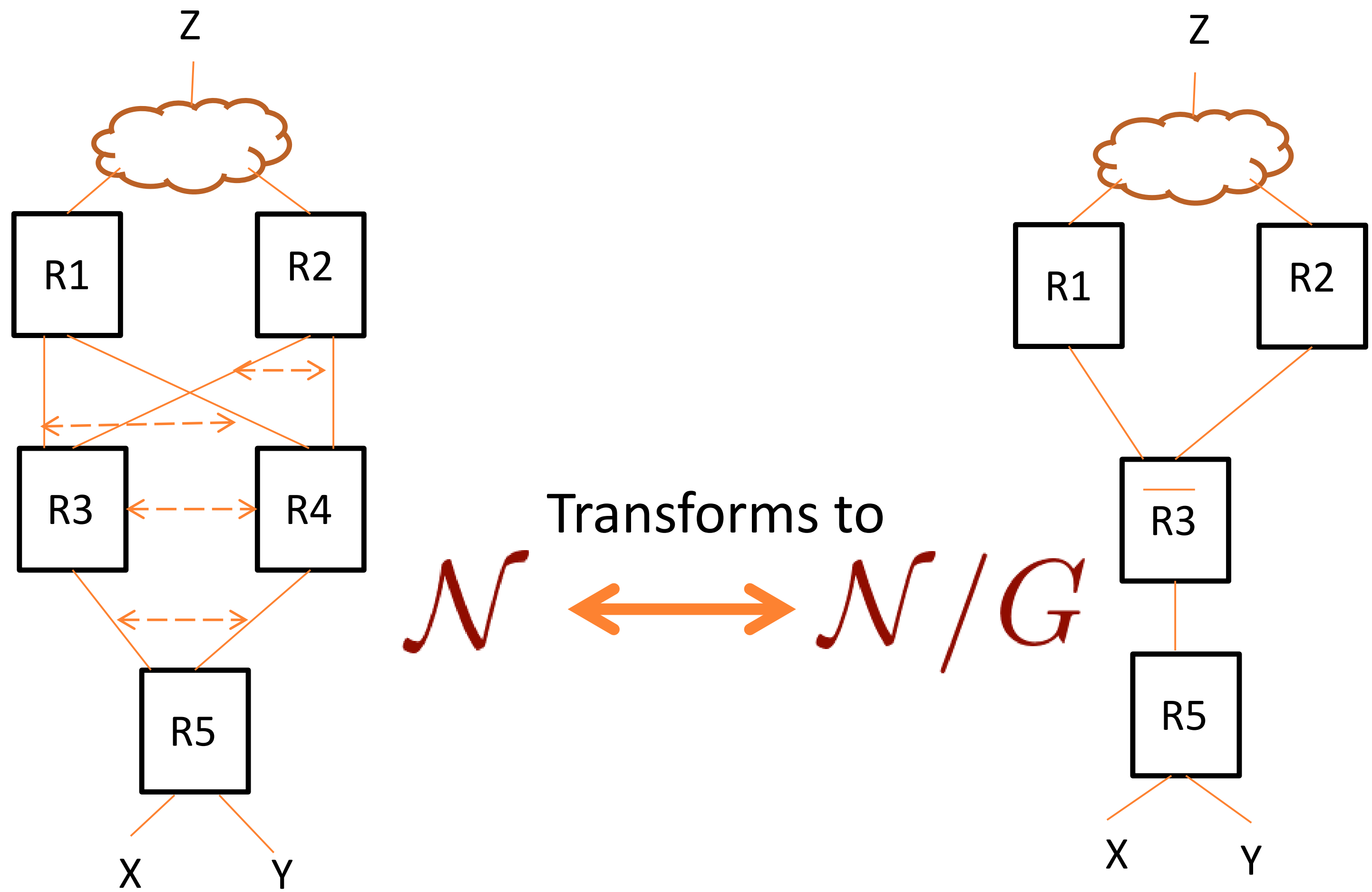
$$h@p \rightarrow^*_{\mathcal{N}} h'@p' \iff \pi_{\mathcal{N}}(h@p) \rightarrow^*_{\mathcal{N}} \pi_{\mathcal{N}}(h'@p')$$

- (Us): Factor symmetries on *topology, headers.* Define symmetry group *G* on *topology*

$$\text{Then } \mathcal{N} \sim \mathcal{N}/G \text{ (via bisimulation)}$$

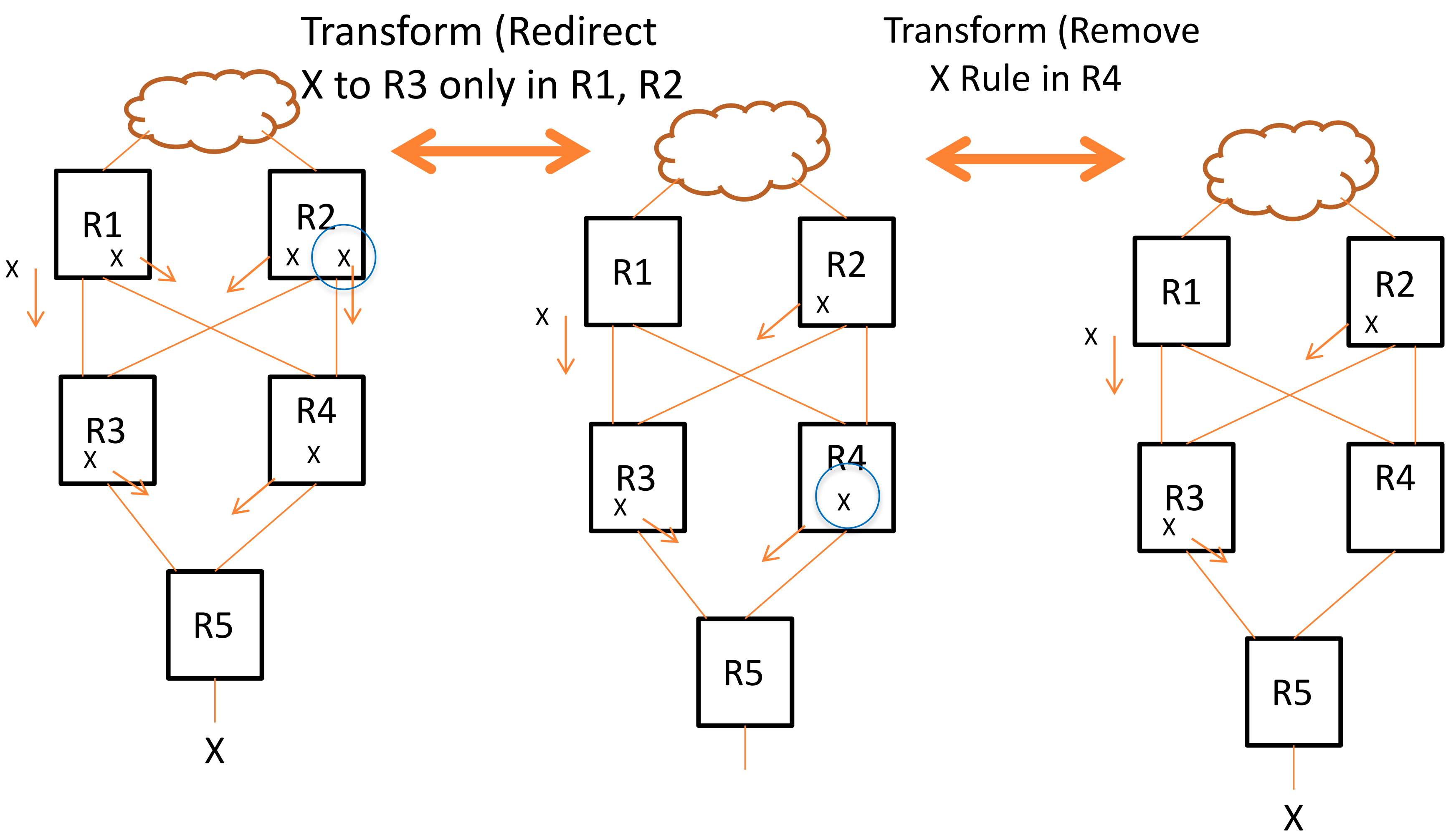
- Theorem: Any reachability formula *R* for original holds iff *R'* holds for reduced network.

# Topological Group Symmetry




REQUIRES *PERFECTLY* SYMMETRICAL RULES AT R3 & R4.
IN PRACTICE, A FEW RULES ARE DIFFERENT.

# Near-symmetry → rule (not box) surgery




Instead of removing boxes, "squeeze" out redundant rules iteratively by redirection and removal. Automate using Union-Find

# Exhaustive verification solutions

- Header equivalence classes: $2^{100}$ → 4000
- Rule surgery: 820,000 rules → 10K rules
- Rule surgery time → few seconds
- Verify all pairs: 131→ 2 hours
- 65 x improvement with simplest ideas. With 32-core machine & other surgeries → 1 minute goal

→ Can do periodic rapid checking of network invariants. Simple version in operational practice

# Ongoing work

| Limitation | Research Project |
|---|---|
| *Booleans* only (Reachability) | *Quantitative* Verification (QNA) |
| No *incremental* way to compute header equivalence classes | New data structure (ddNFs) Venn diagram intersection |
| *Data plane* only: no verification of routing computation | *Control Space Analysis* (second part of talk) |
| *Correctness* faults only (no *performance* faults) | Data-plane tester ATPG (aspects in Microsoft clouds) |
| Stateless Forwarding Only | Work at Berkeley, CMU |


# Finding Misconfigurations without Specs

(Kakarla, Beckett, Jayaram, Millstein, Tamir, Varghese, NSDI 2020)

exploiting network specific *data mining*

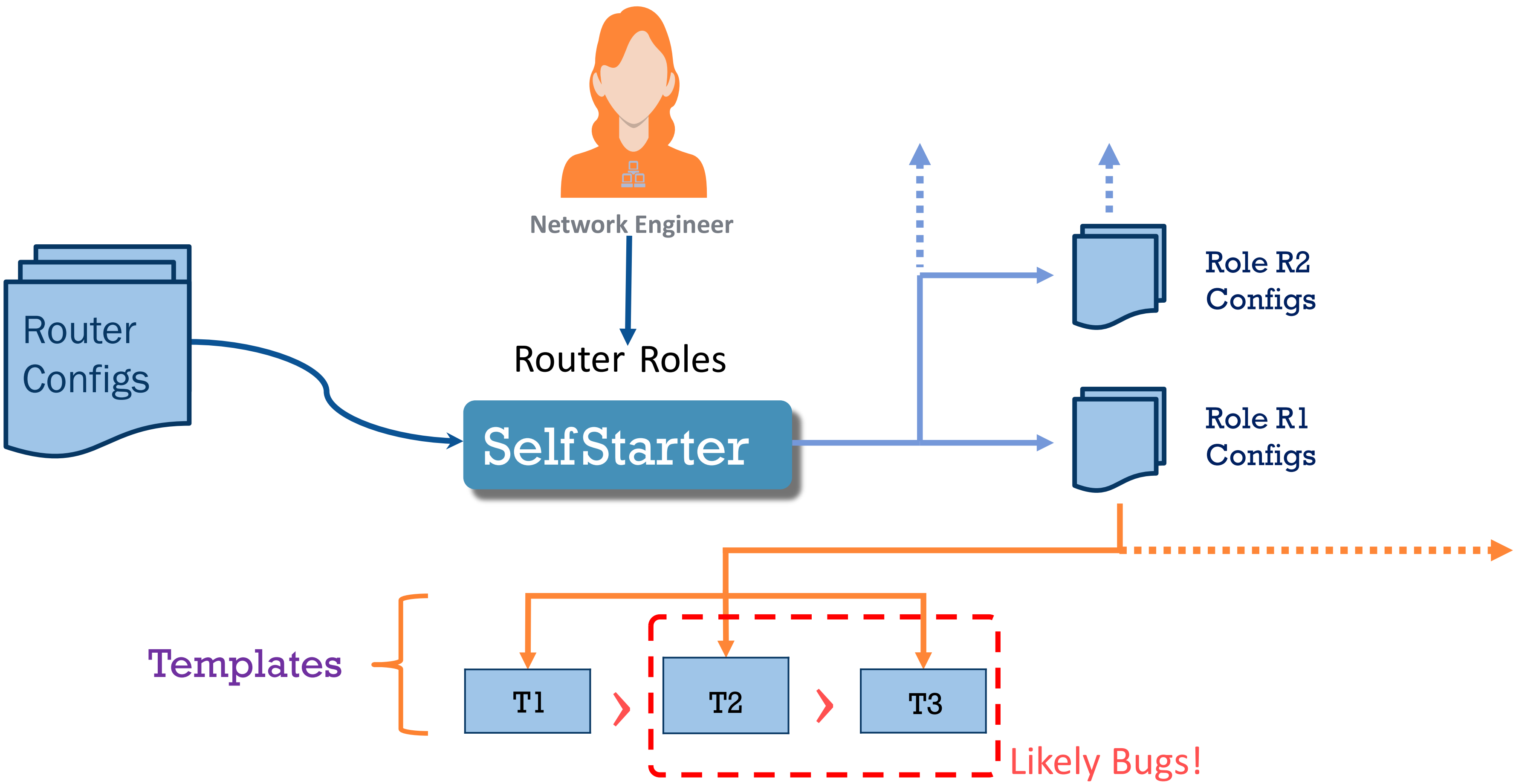
# NETWORK VERIFICATION STATUS

- Scaling: Network specific formal methods that scale to large networks by defining equivalence classes.
- Commercial Entries: Forward Networks, Veriflow Networks, IntentionNet, Amazon, Cisco
- Limited success: can check for certain canned properties (e.g., no loops) but can't verify network specific properties
- Lack of specifications: distributed management, churn, turnover → knowledge, if any, is partial and imprecise

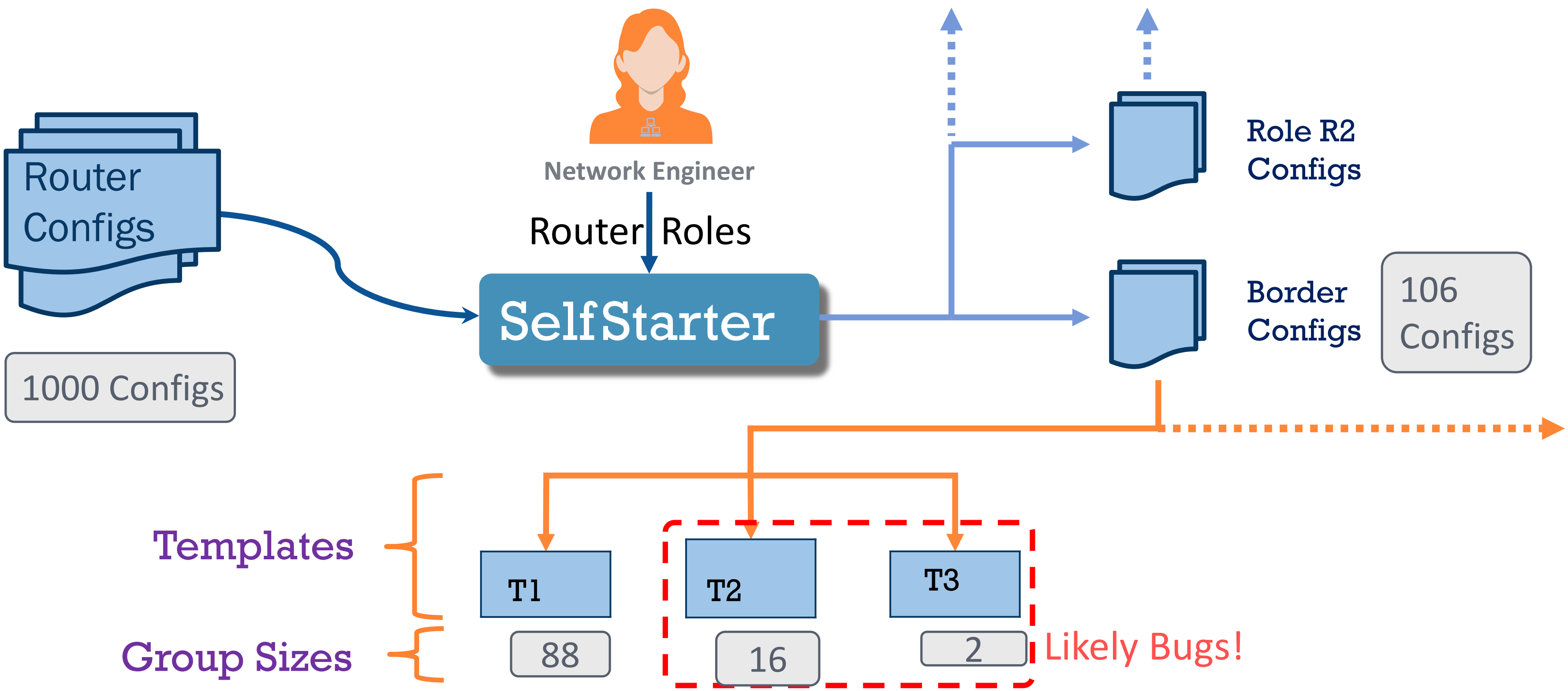
# SELF-STARTER: FINDING BUGS USING NETWORK SPECIFIC DATA MINING

- Bug Finding not Proofs: Limit ourselves to finding bugs not proving correctness
- Deviant behavior (SOSP 01): deviation from majority -> bug. Found many bugs in Linux
- Network Specific Insight: Routers in same role (e.g. core, edge) should be similar; deviations → likely bugs
- Network Specific Data Mining: clustering, k-means works badly, instead cluster based on "similar" templates
- Templating Algorithm: parameter generalization crossed with sequence alignment

# End-to-End Design



# Example run on UCLA



# Analogy of Anomaly Detection for Stories

## Story 1

John met Harry
in the park.
Harry and John
played soccer.
Later, John went
home to supper

## Story 2

Bob met Brad
in the park.
Brad and Bob
played soccer.
Later, Bob went
home to supper

## Story 3

John is a
trumpet player
John plays
Mozart at night.
John won a prize
for music.

# Anomaly Detection by clustering templates

A met B in the park. B and A played soccer. Later, A went home to supper

John is a trumpet player.
John plays Mozart at night.
John won a prize for music.

Template 1 (2 instances)

Template 2 (1 instance) (the anomaly)

# Same idea for Network Config "stories" Challenge 1: Benign Differences

```
ip access-list extended ACL1
deny     udp    host          0.0.0.0      any
permit   tcp    17.12.11.0    0.0.0.255    any
deny     icmp   17.12.11.0    0.0.0.255    any
permit   ip     16.21.0.0     0.0.63.255   any
permit   ip     17.12.11.0    0.0.0.255    any
```

```
ip access-list extended ACL2
deny     udp    host          0.0.0.0      any
permit   tcp    17.12.13.0    0.0.0.255    any
deny     icmp   17.12.13.0    0.0.0.255    any
permit   ip     17.12.13.0    0.0.0.255    any
permit   ip     16.23.0.0     0.0.63.255   any
```

Challenge: Non-identical lines

Ingredient 1: Parameterization

# Challenge 2: Missing Lines and Reordering

```
ip access-list extended ACL1
deny     udp    host         0.0.0.0      any
permit   tcp    17.12.11.0   0.0.0.255    any
deny     icmp   17.12.11.0   0.0.0.255    any
permit   ip     16.21.0.0    0.0.63.255   any
permit   ip     17.12.11.0   0.0.0.255    any
```

```
ip access-list extended ACL2
deny     udp    host         0.0.0.0      any
permit   tcp    17.12.13.0   0.0.0.255    any
deny     icmp   17.12.13.0   0.0.0.255    any
permit   ip     17.12.13.0   0.0.0.255    any
permit   ip     16.23.0.0    0.0.63.255   any
```

Challenge: Allow certain reorderings but not arbitrary reorderings

Solution: Two-level abstraction using blocks

A block is a contiguous sequence of lines that can be arbitrarily reordered but the order of blocks is important.

# Ingredient 2: Sequence Alignment +Blocks

```
ip access-list extended ACL1
deny    udp   host        0.0.0.0     any
permit  tcp   17.12.11.0  0.0.0.255   any
deny    icmp  17.12.11.0  0.0.0.255   any
permit  ip    16.21.0.0   0.0.63.255  any
permit  ip    17.12.11.0  0.0.0.255   any
```

```
ip access-list extended ACL2
deny    udp   host        0.0.0.0     any
permit  tcp   17.12.13.0  0.0.0.255   any
deny    icmp  17.12.13.0  0.0.0.255   any
permit  ip    17.12.13.0  0.0.0.255   any
permit  ip    16.23.0.0   0.0.63.255  any
```

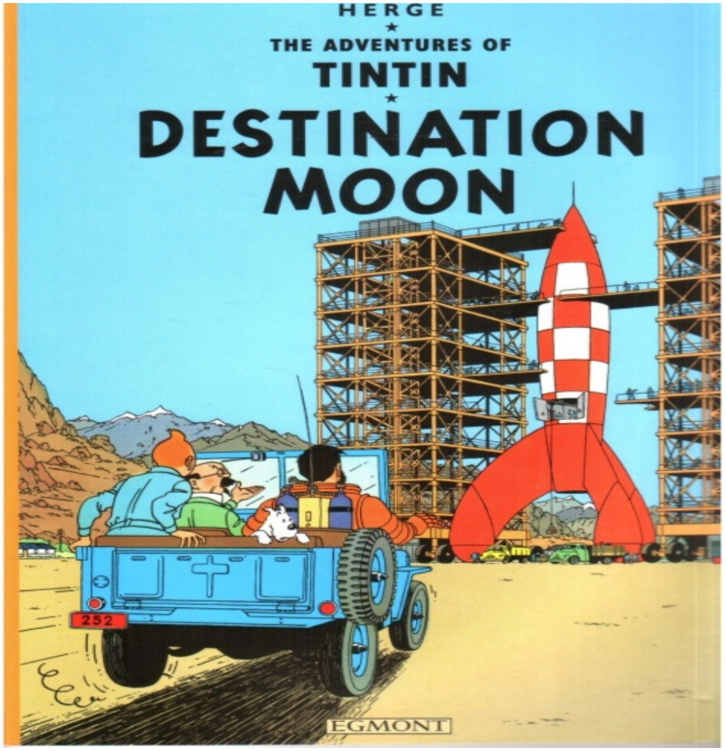
Block Alignment → *Sequence alignment* to prevent cross-block reordering

Line Reorderings → *Minimum-weight bipartite matching* to allow within-block line reorderings

# Results

| Network | Segment Type | Consistent Templates | Inconsistent Templates | | |
|---|---|---|---|---|---|
| | | | Identified | Investigated | True Positives (% of investigated) |
| UCLA | ACLs | 0 | 6 | 3 | 3 (100%) |
| Microsoft WAN | Prefix lists | 10042 | 166 | 138 | 7 (5%) |
| | Route policies | 10969 | 56 | 33 | 33 (100%) |
| Microsoft Data center | ACLs | 9700 | 938 | 400* | 400 (100%)* |
| | Prefix lists | 2954 | 0 | - | - |
| | Route policies | 11653 | 230 | 230* | 230 (100%)* |

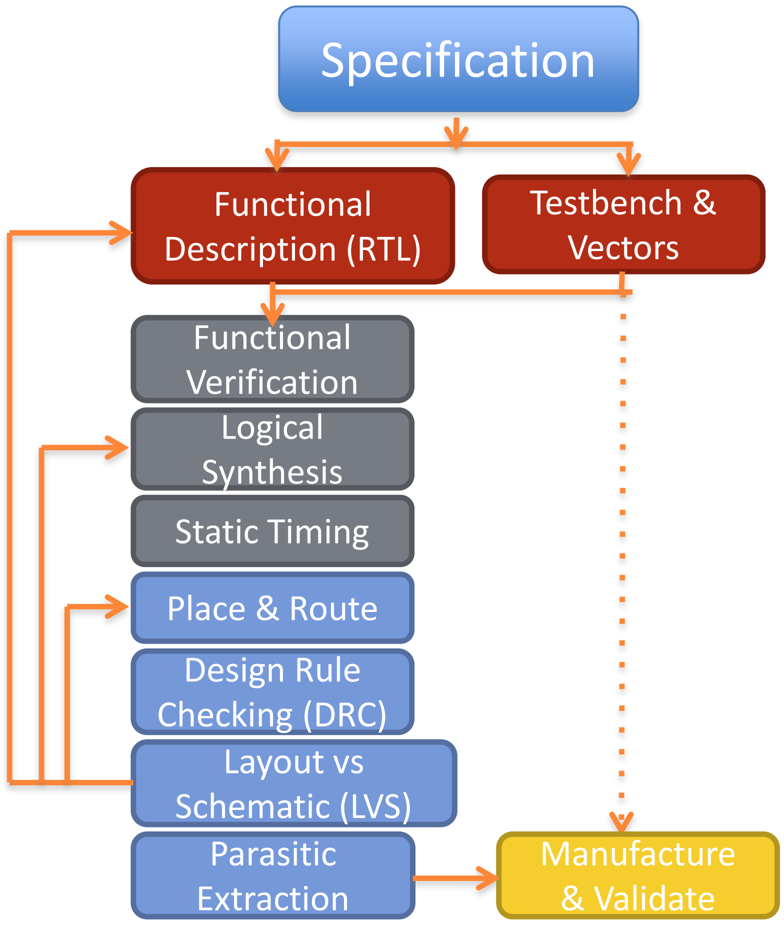
90 min



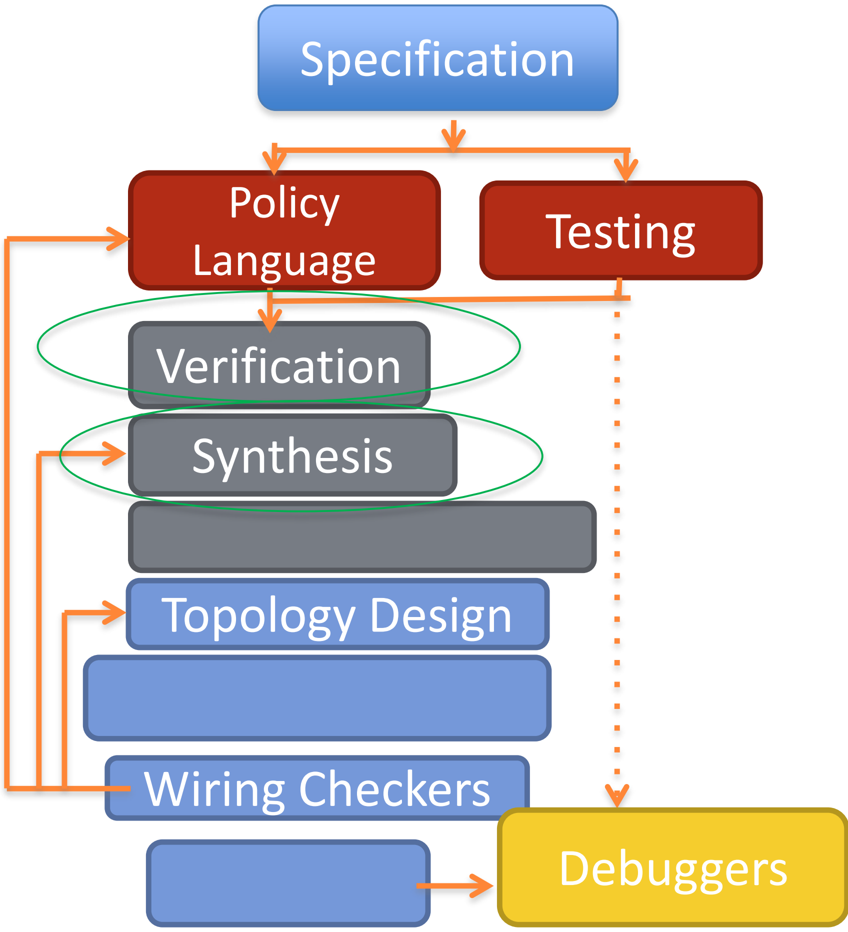

# 3.0 NETWORK DESIGN AUTOMATION

## NSF LARGE GRANT 1901510, UCLA, USC

# Digital Hardware Design as Inspiration




Electronic Design Automation
(McKeown SIGCOMM 2012)



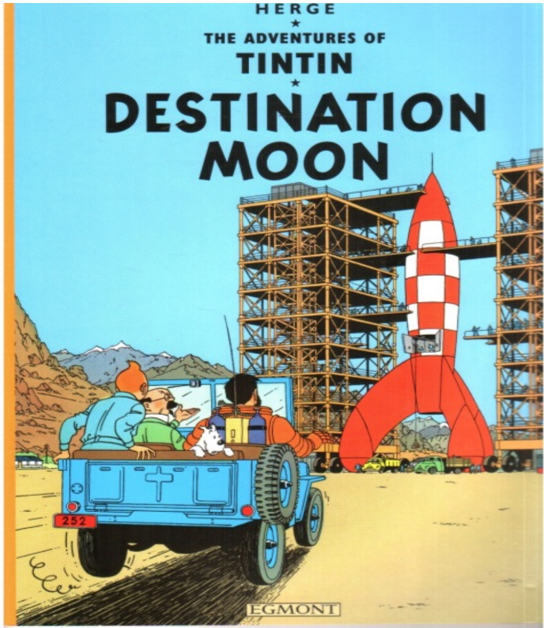

Network Design Automation
(NDA): NSF Large Grant

# EDA design tool wish list

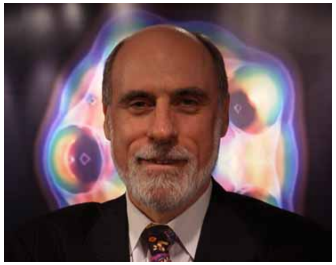
- Analysis:
  - Automatic test packets (“Post-silicon” debug)
  - Debuggers (how to “step” through network?)
  - Timing Verification for real time traffic
- Synthesis:
  - A Verilog for network configurations?
  - Scalable specifications (network types?)

# Conclusion

- Inflection Point: Rise of services, SDNs
- Intellectual Opportunity: New techniques, *network specific symmetries*, *network specific data mining*.
- Working chips with billion transistors. Large networks next? Need help from EDA!



# Thanks




- (MSR): N. Bjorner, R. Beckett, K. Jayaraman, N. Lopes, G. Plotkin, A. Rybalchenko
- (Stanford): P. Kazemian, N. McKeown
- (UCLA): T. Millstein, Y. Tamir, S. Kesava, A. Tang