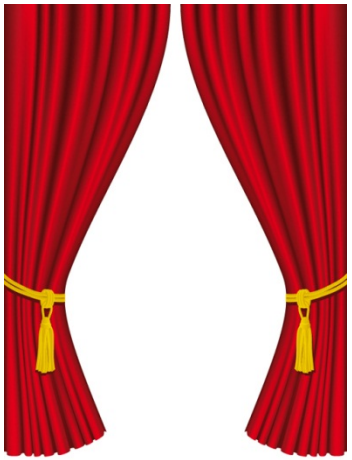
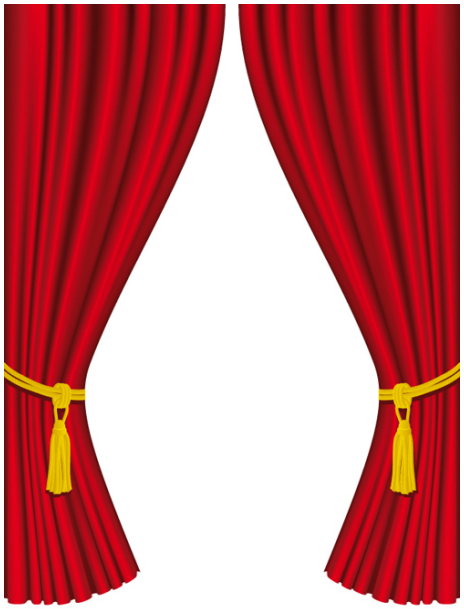


Automatic Verification of P4 Networks

Nuno Lopes, Nikolaj Bjorner, Andrey Rybalchenko, Nick McKeown, Dan Talayco, George Varghese

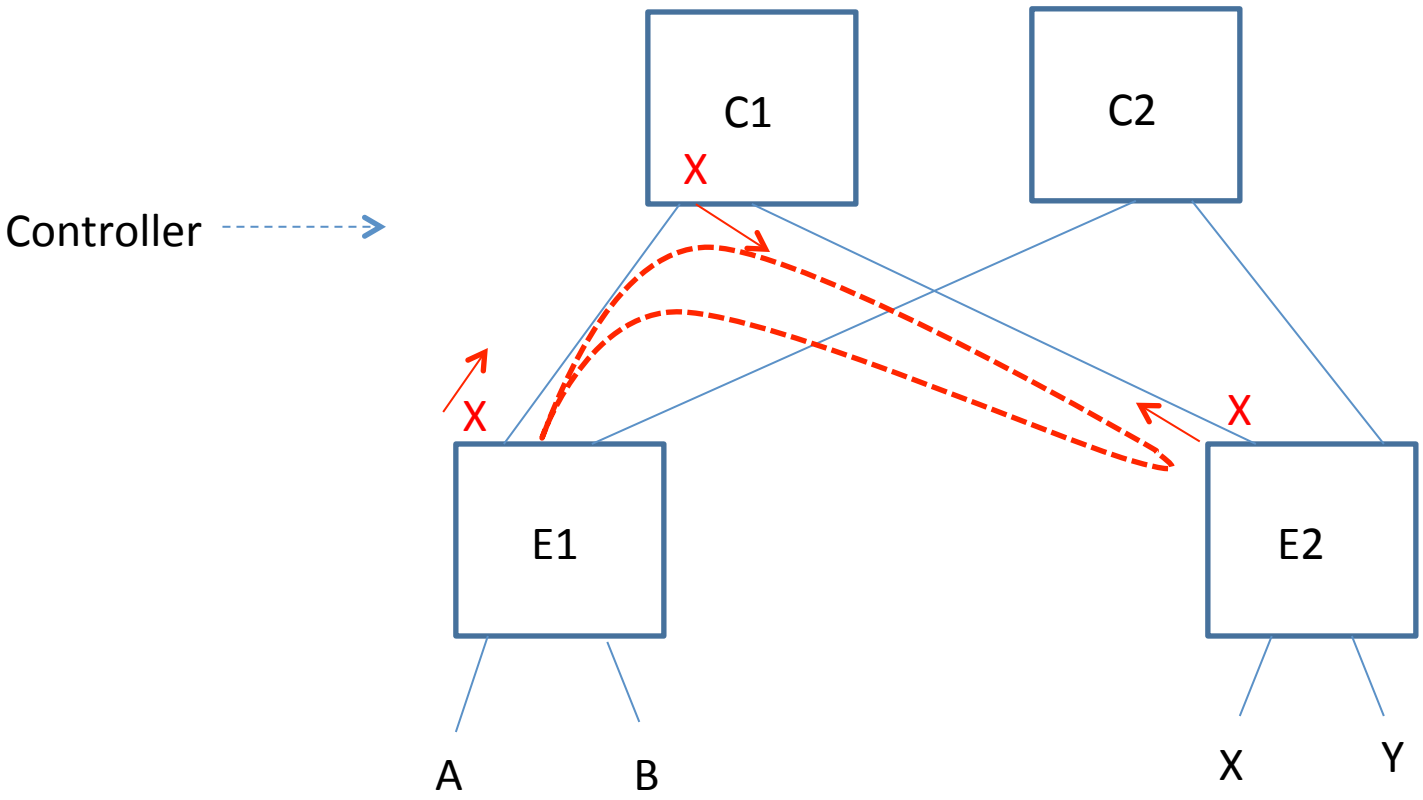
Microsoft, Intel, Stanford



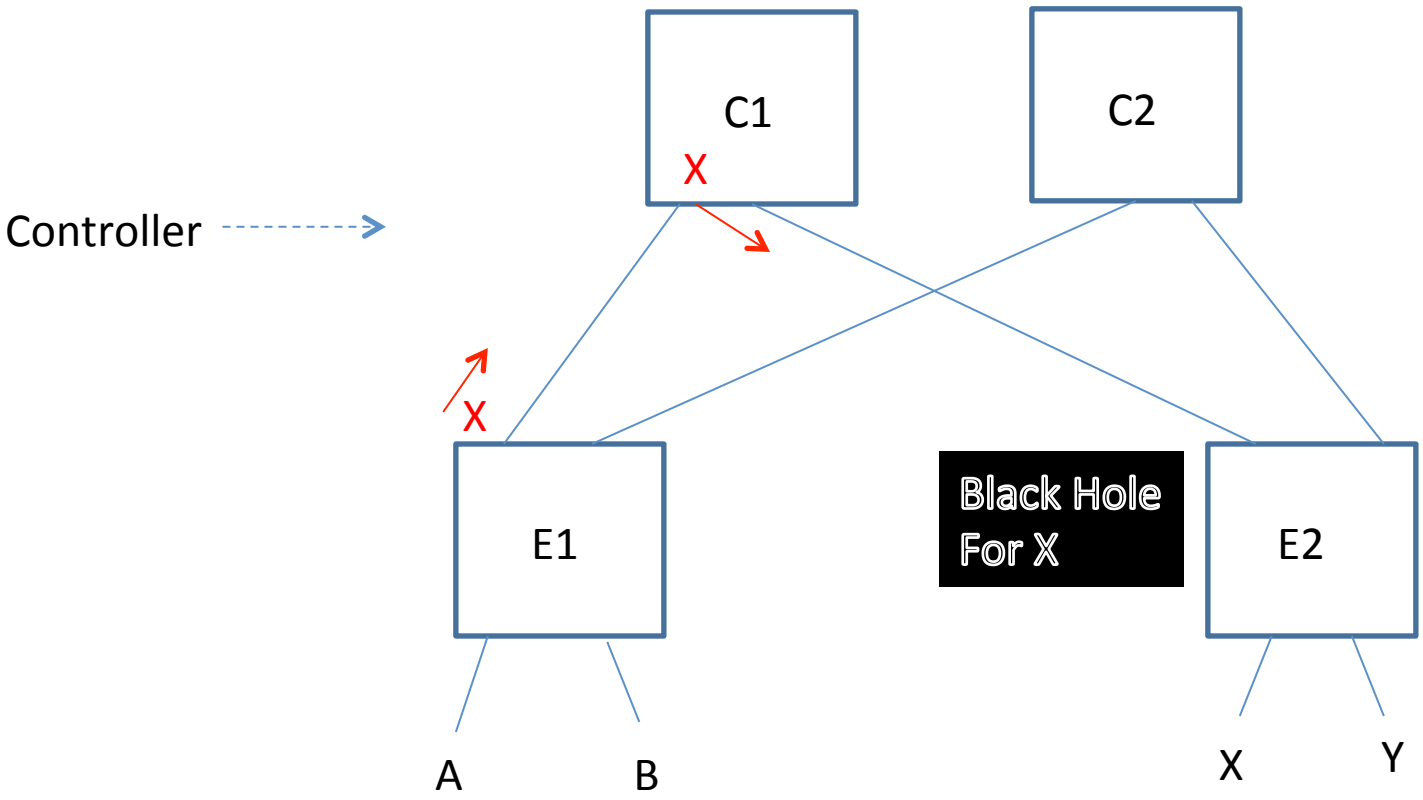
Scene 1: SDN cries out for verification

The *flexibility* provided by software defined networks also allows new software bugs that can compromise network operations

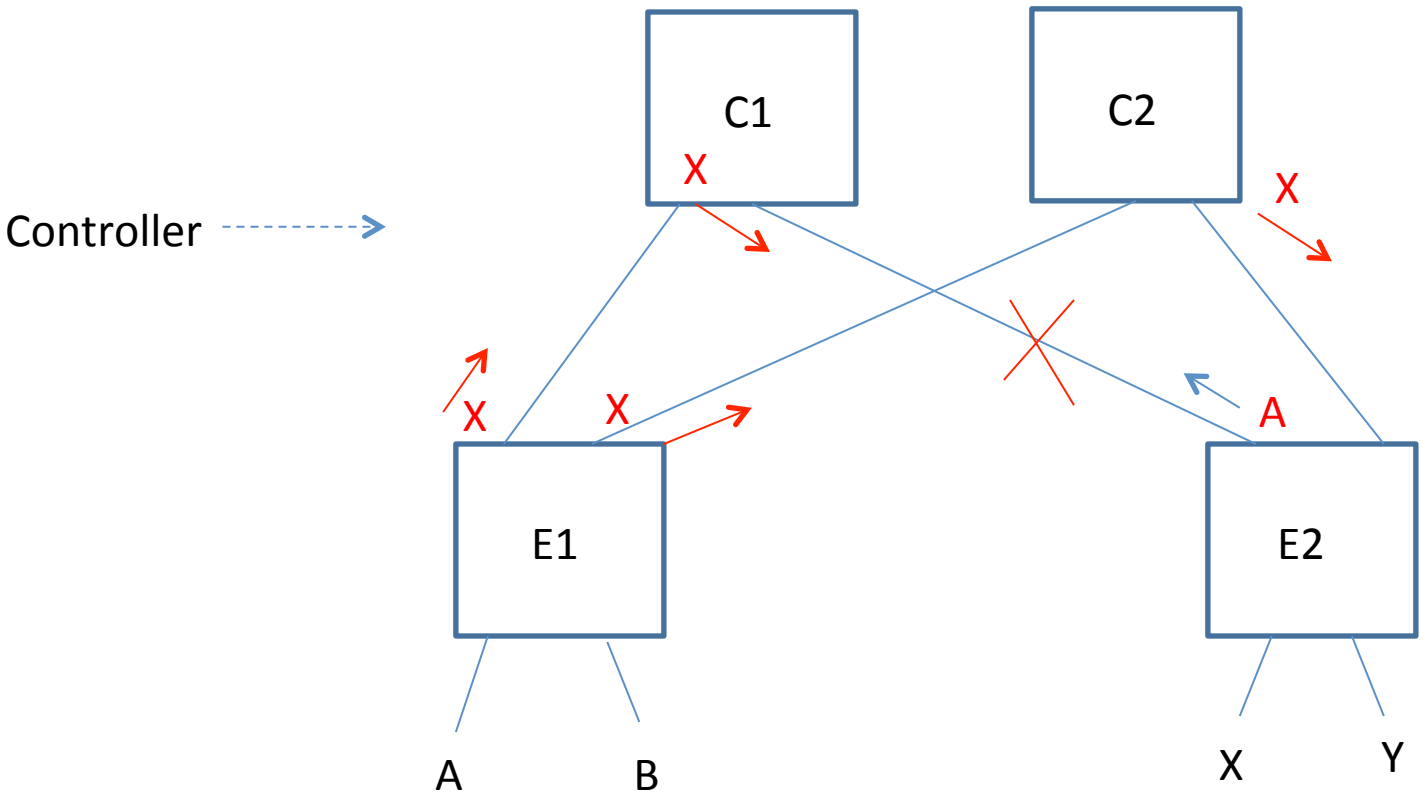
MISCONFIGURED TABLE LOOPS



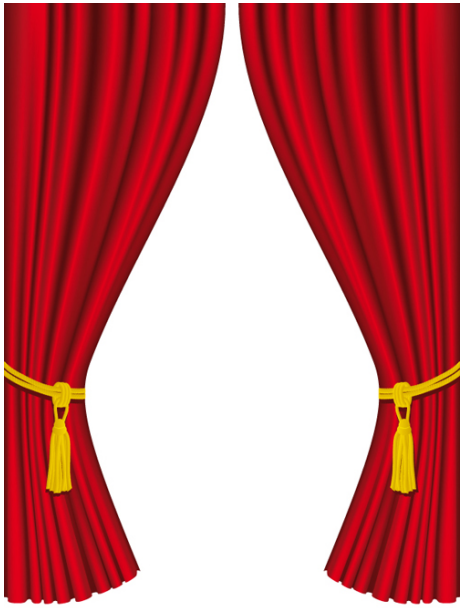
BLACK HOLES



INSUFFICIENT RESILIENCY



After failure of link from C1 to E2, A can talk to X but X cannot talk to A



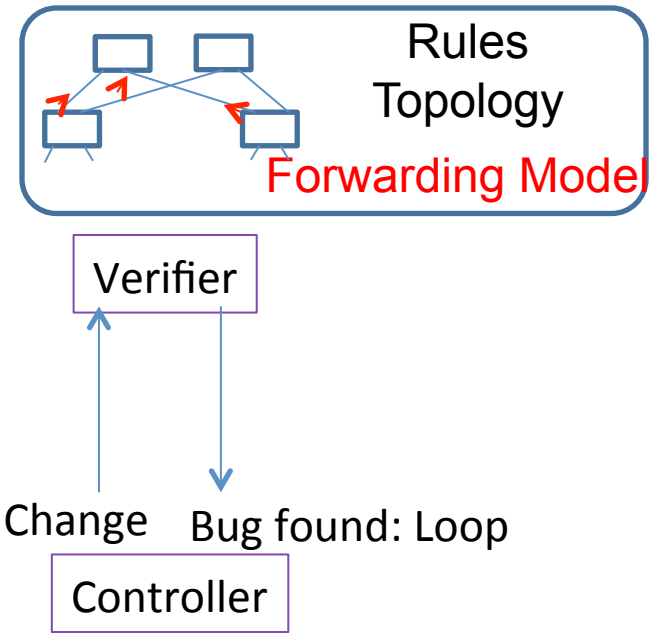
Scene 2: How verification can help

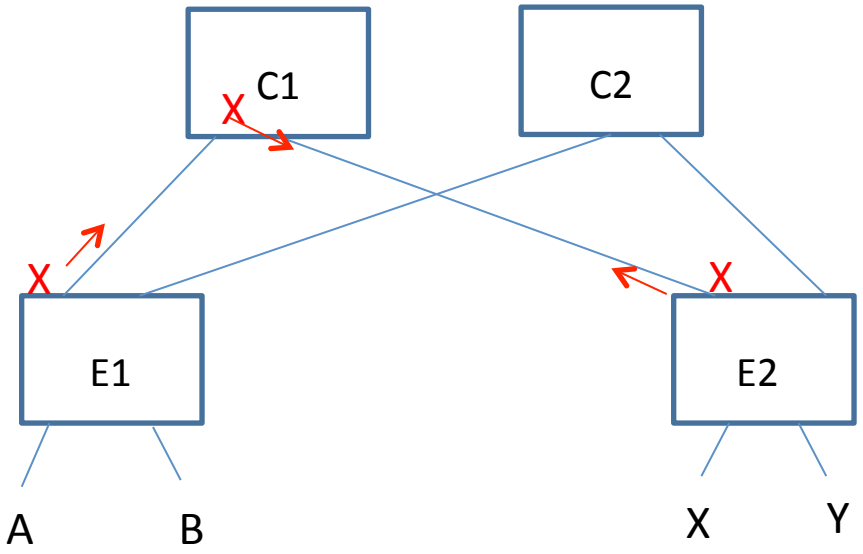
Simulators can catch many bugs but cannot *check* reachability and loop freedom across *all* packet headers

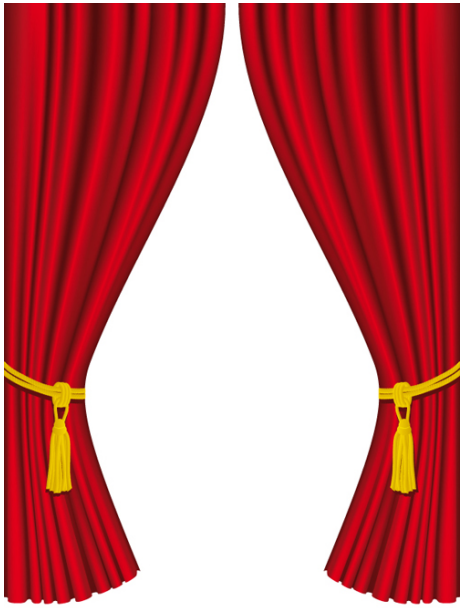
Network Verification

- Model checking needed: efficient search across all possible packets
- Recent tools: Veriflow, Hassel, NICE, Flowlog, NetCore, NoD
- Challenge: 100 bit headers → 2^{100} headers, thousands of routers, millions of rules
- Approach: Exploit network structure: prefixes, symmetries

“Full coverage” regression testing via verification







Scene 3: Why P4 breaks things

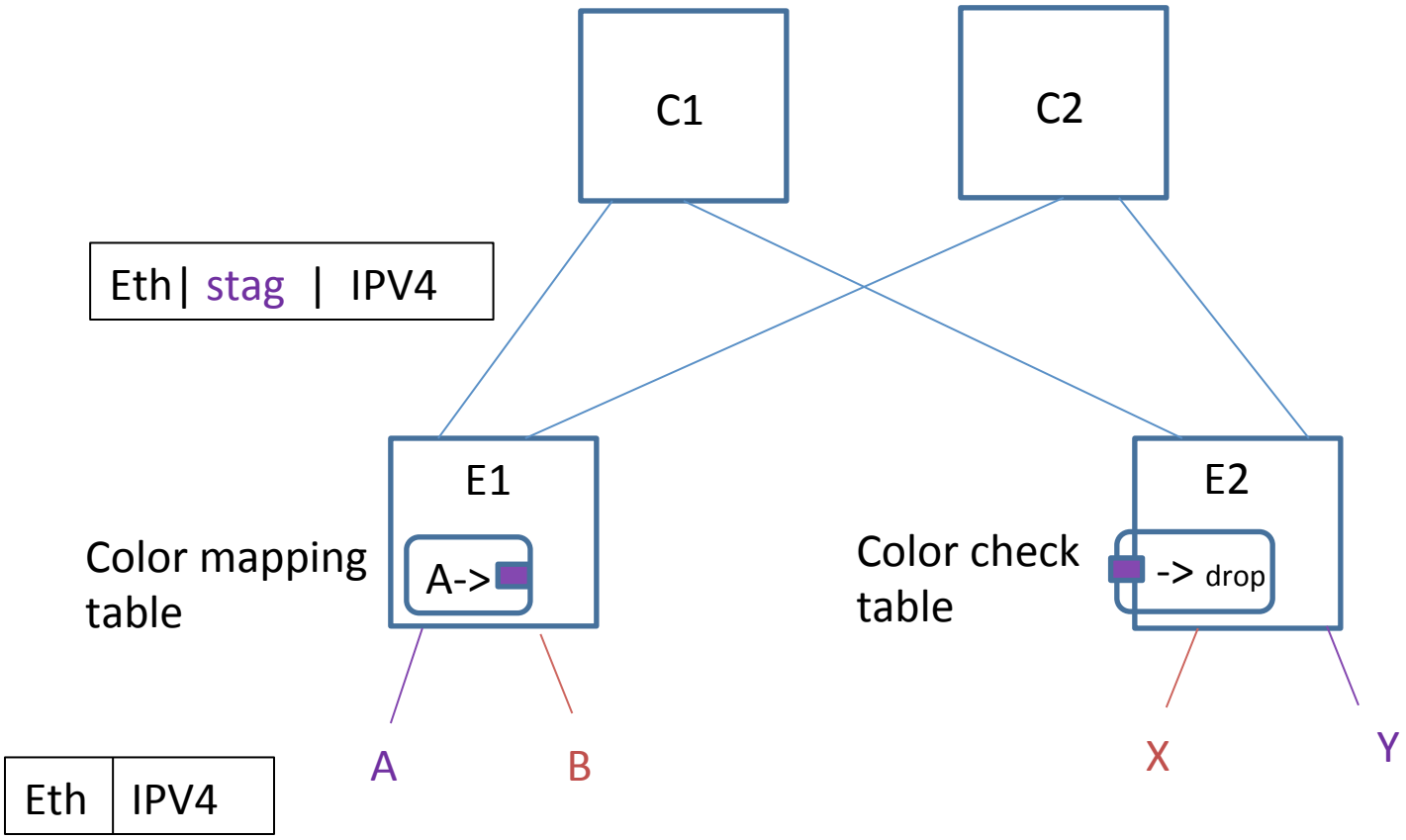
Three common P4 use cases

- Programmable headers and forwarding
- Moving match resources between tables
- Monitoring and debugging

Example Forwarding Functionality

- Goal: Divide hosts into arbitrary equivalence classes that are isolated from each other without using ACLs between every pair.
- Untrusted host software-> Want Network enforced security, do not trust hypervisor

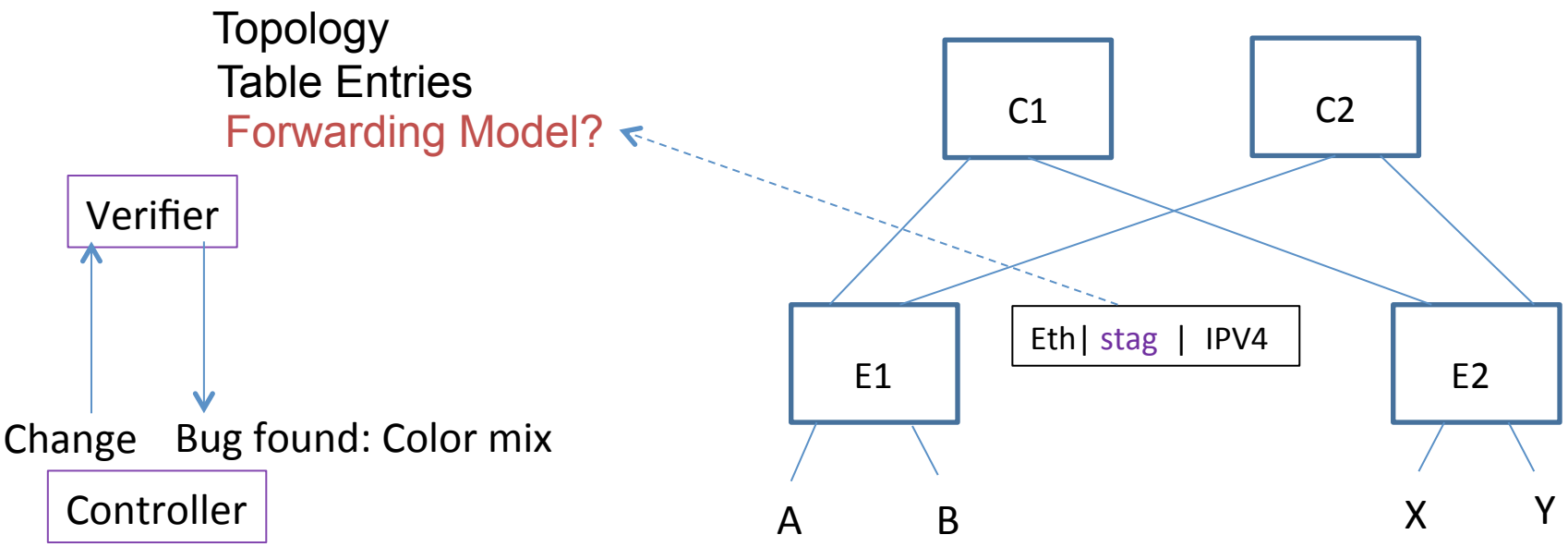
ADDING SECURITY TAGS

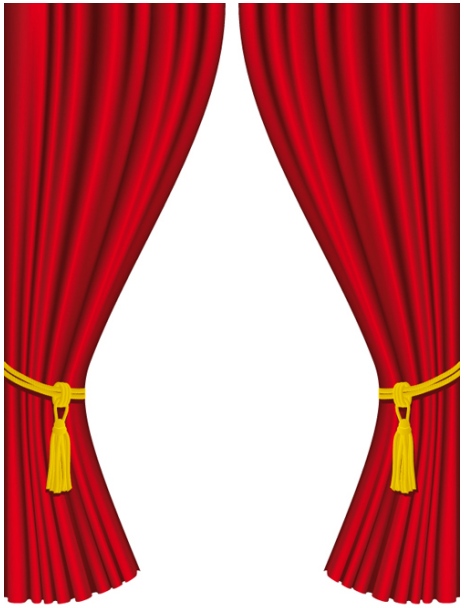


P4 program for Stag

- Header: Add new security tag (color) field
- Tables: Add 2 new tables to map colors at input and check colors at output
- Control flow: Add color map before forward on ingress and after forward on egress

Why Regression testing can Break





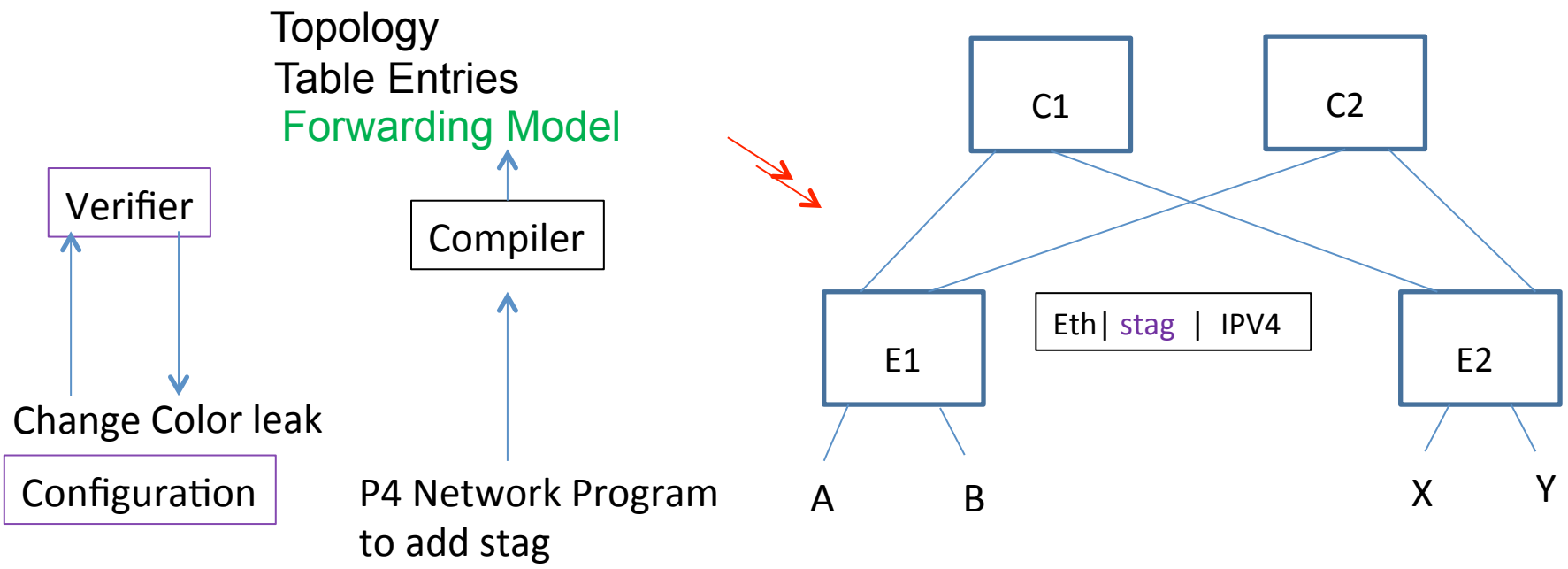
Scene 4: Automatically compiling P4 to a verification language

What is the problem?

- The forwarding model of a router is hardcoded within Hassel and Veriflow
- NoD allows users to add new rules but hard to keep up with change
- Better solution: *automatically* compile from P4 to a forwarding model accepted by a tool.
- We chose: P4 → Datalog -> NoD

SDNs require verifiers but *Programmable* networks need *automatic* verifiers

REGRESSION TESTING RESTORED



Compiler Details

- Semantics: First gave an operational semantics for P4 statements related to forwarding
- Datalog conversion: Straightforward conversion from semantics to Datalog
- Optimization: Lots of inlining done, more optimizations needed.
- Implementation: 2900 Lines of OCaml code

Semantics may be useful independent of verification

What Properties are "automatic"?

- Unchanged: Reachability, loops, resilience
- New class of bugs: P4 programs can produce "ill formed" packets (e.g., 2 stags in a packet)
- Automating well-formedness checks: Prove packets output at egress are parseable at ingress
- Better still: Prove well-formedness regardless of contents of tables ("symbolic" tables)

Conclusion

- Verification provides “full coverage” regression test suite essential for SDN networks
- Automatically compile P4 to verification model to keep pace with change
- Consider using NoD/P4 compiler under MIT license to build tools for your customers
- Check out Andrey’s demo