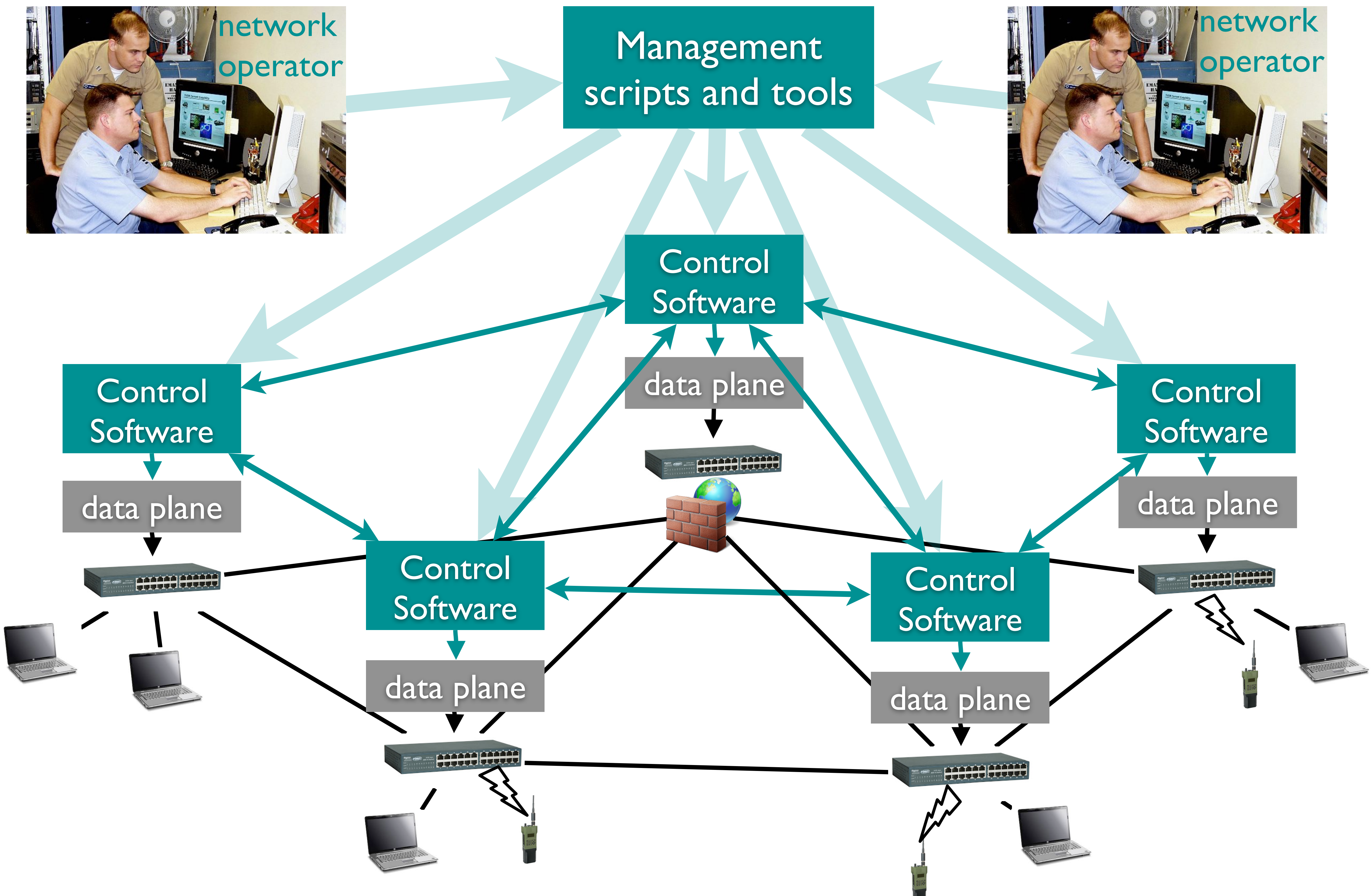
Automated Synthesis of Reactive Controller for Software-defined Networks

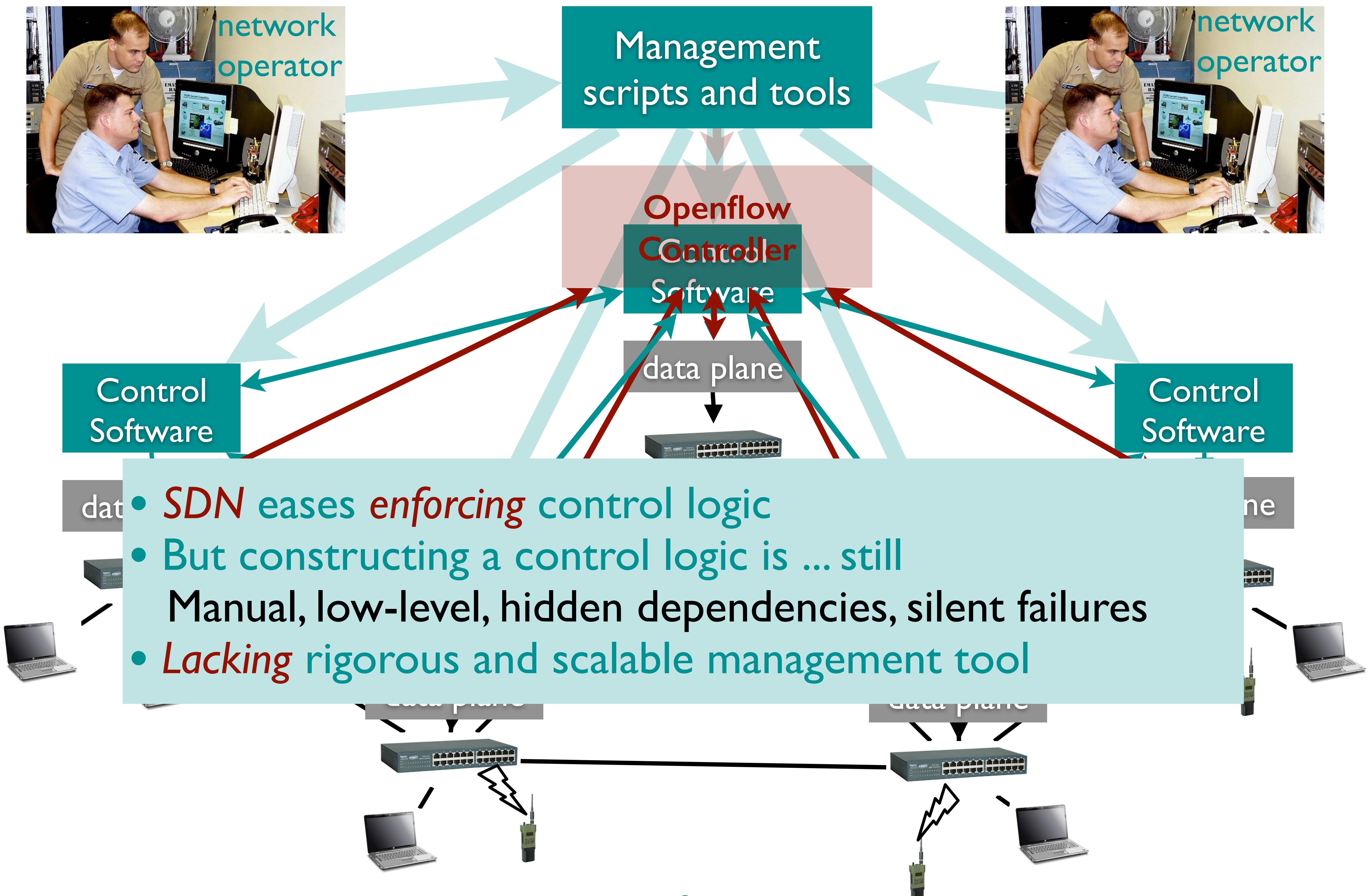
Anduo Wang Salar Moarref
Ufuk Topcu Boon Thau Loo Andre Scedrov

University of Pennsylvania

Networks are complicated

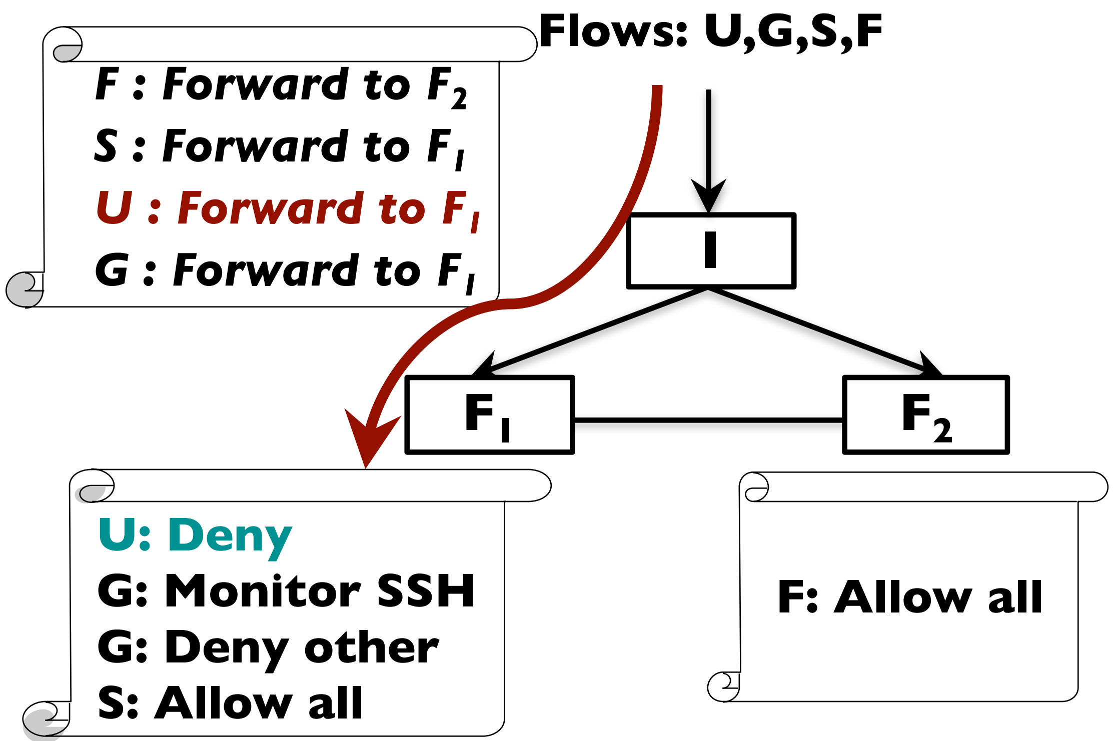


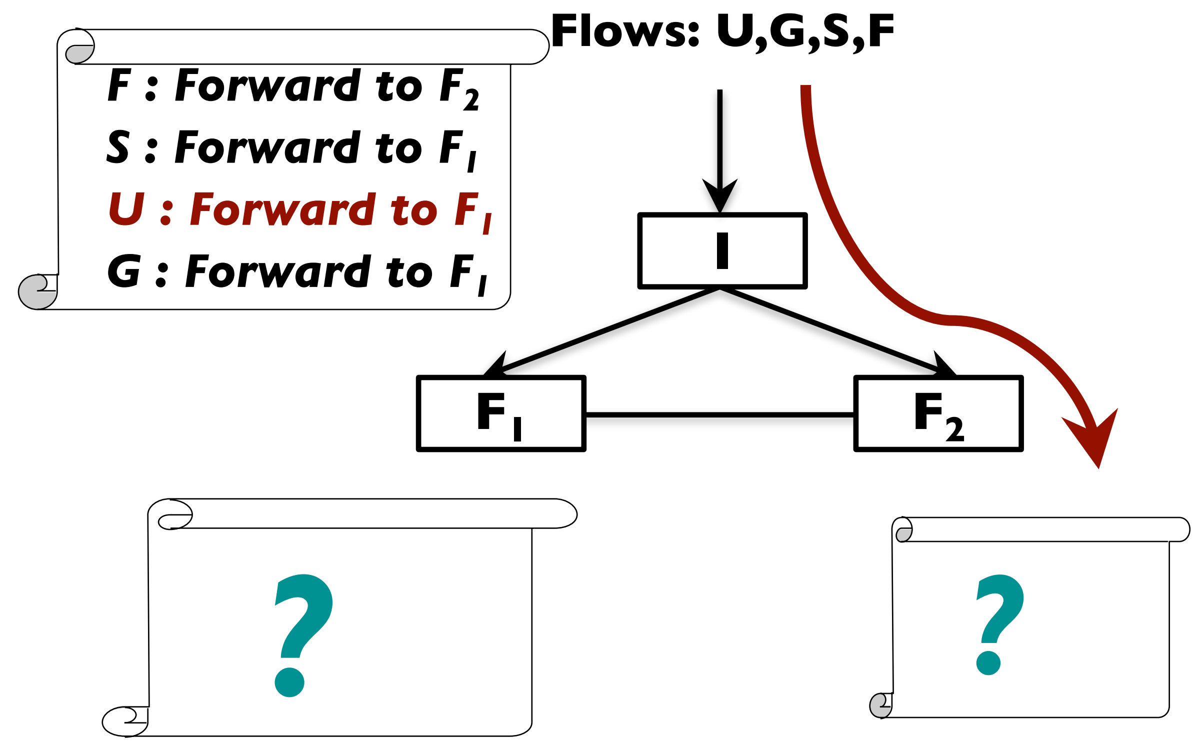
Management in Software-Defined Network



- *SDN* eases *enforcing* control logic
- But constructing a control logic is ... still
  Manual, low-level, hidden dependencies, silent failures
- *Lacking* rigorous and scalable management tool

Example problem: constructing access control





Switches: I (ingress), $F_{1,2}$ (for two servers)
Flows: U (untrusted), G(guest), S(student), F(faculty)
Security policy: do not allow U flows to transit

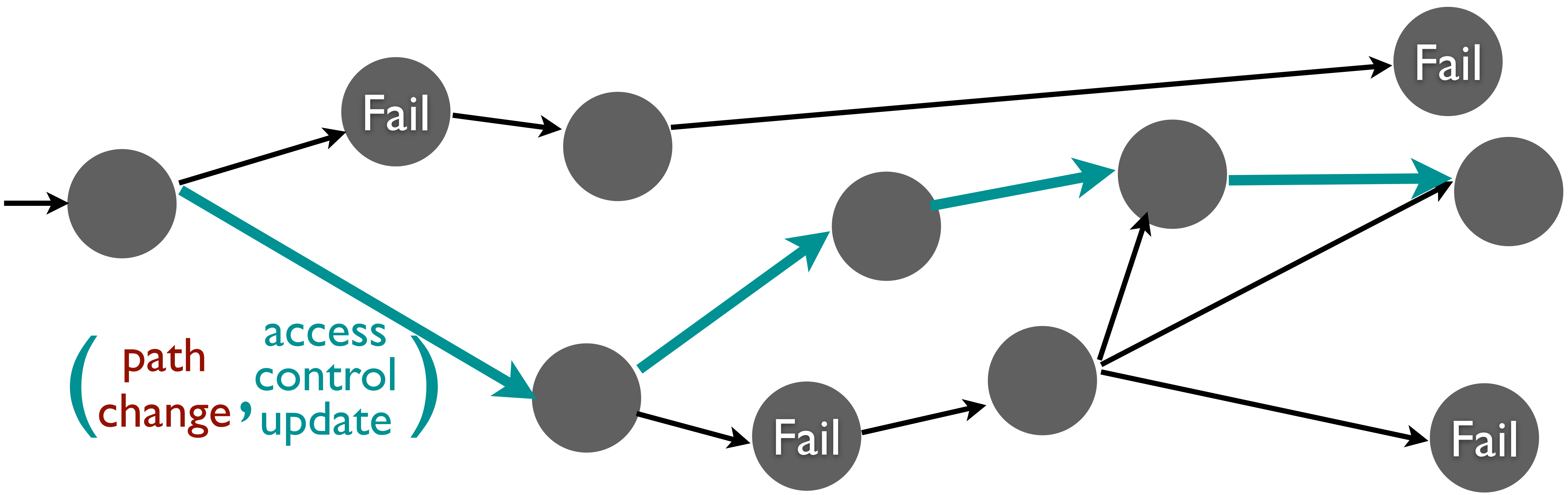
Routing path changes
How to update access control

- Find a strategy for updating access control rules
  - Enforce security policy for all path changes
- Given a strategy, find an ordering of rule updates
  - Enforce security policy for all transient states

Outline

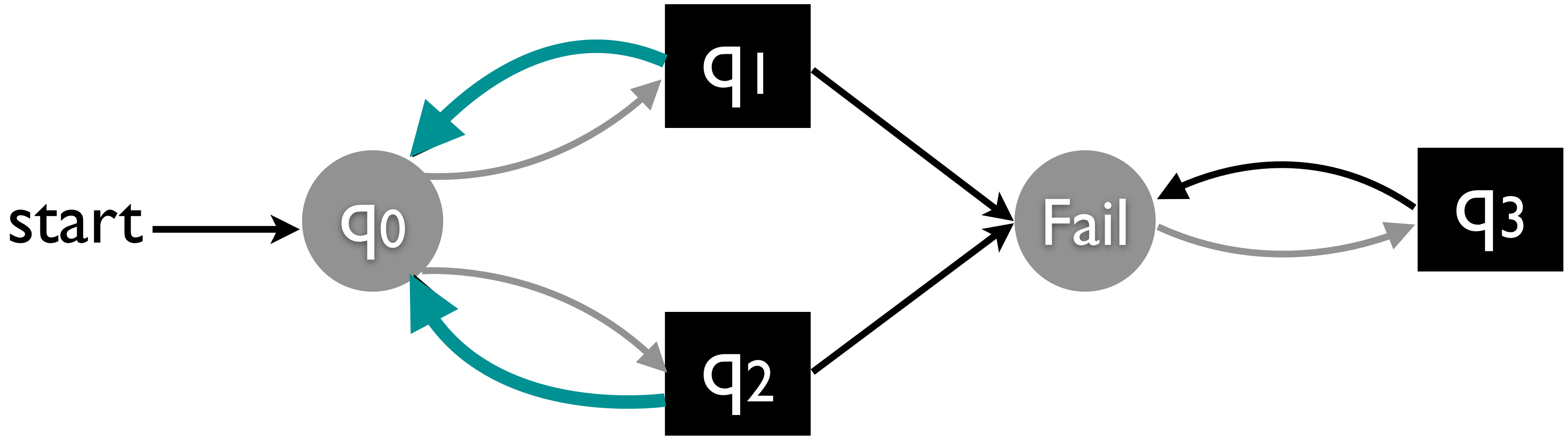
- Synthesize provably correct control logic
  - Formulate and solve as a reactive synthesis problem
- Scale by network abstraction
  - Introduce network abstraction as simulation relation

Synthesize access-control for example problem



- Formulate as reactive synthesis -- a two-player, temporal logic game
  - Routing path rule (player 1) triggers a change, access-control (player 2) makes an update in response
  - Temporal property specifies security policy
    - A winning strategy for access-control enforces security policy against any path change

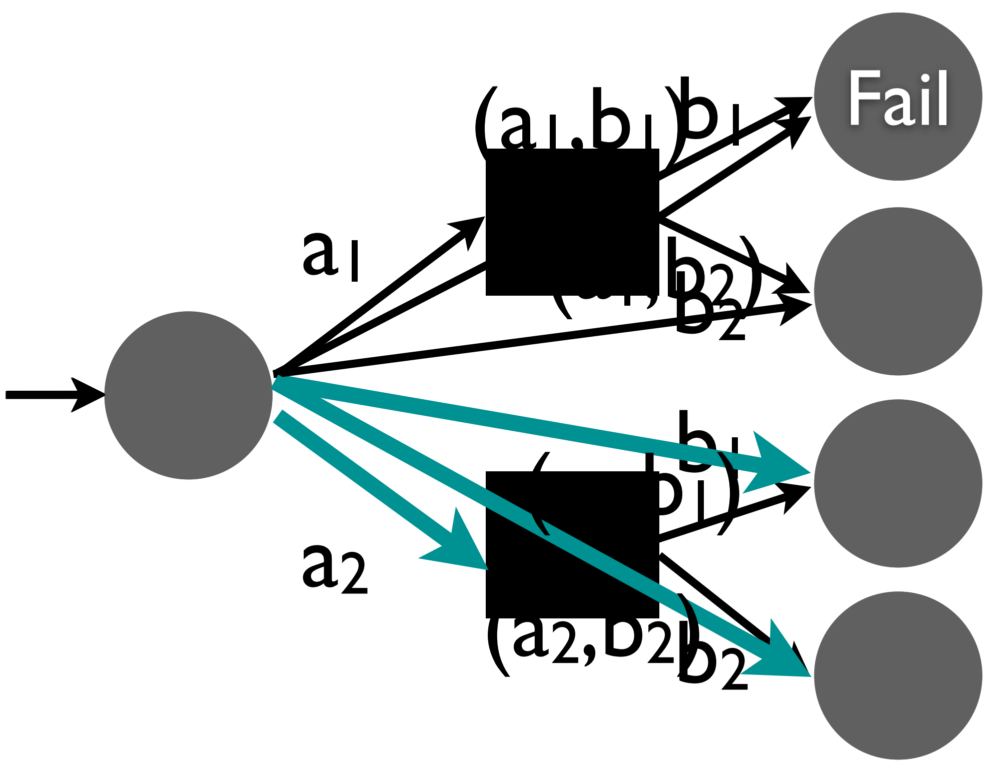
Background: two player, temporal logic game

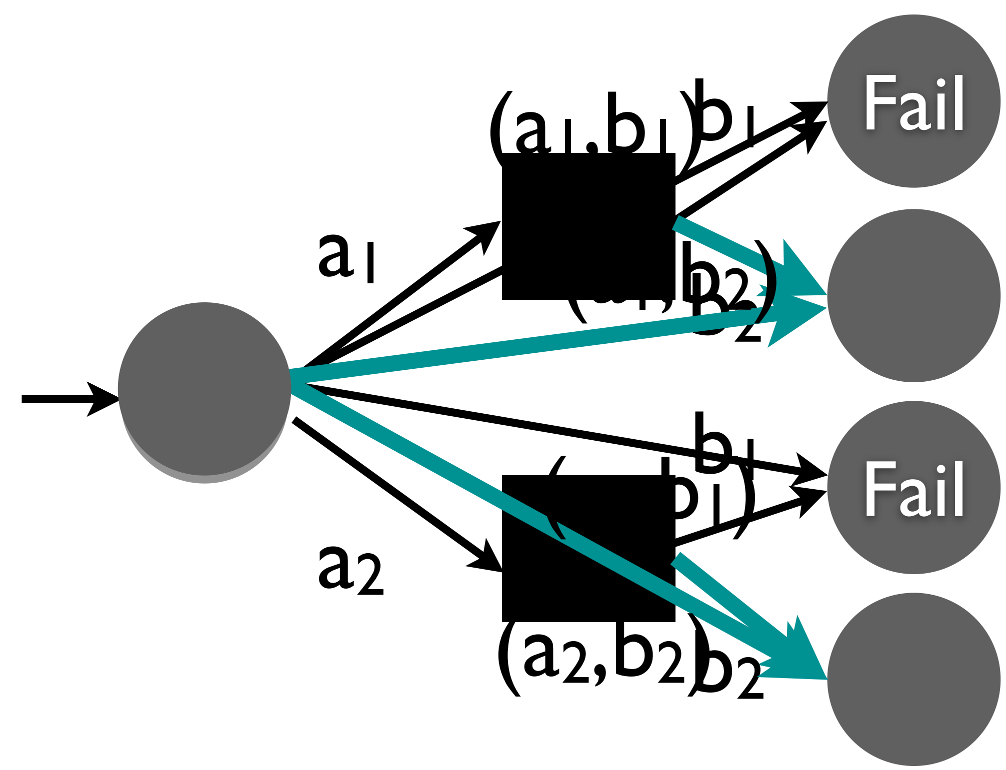


- Two players make alternating moves
  - Circle (Square) represents two player states $Q_1(Q_2)$
- Temporal logic specifies player's goal
  - Never enter Fail state: $\Box(\neg \text{Fail})$
- Synthesize a winning strategy
  - Q_2 avoids Fail state regardless how Q_1 moves

Synthesis -- two player, temporal logic game

Combine alternating transitions into a joint action

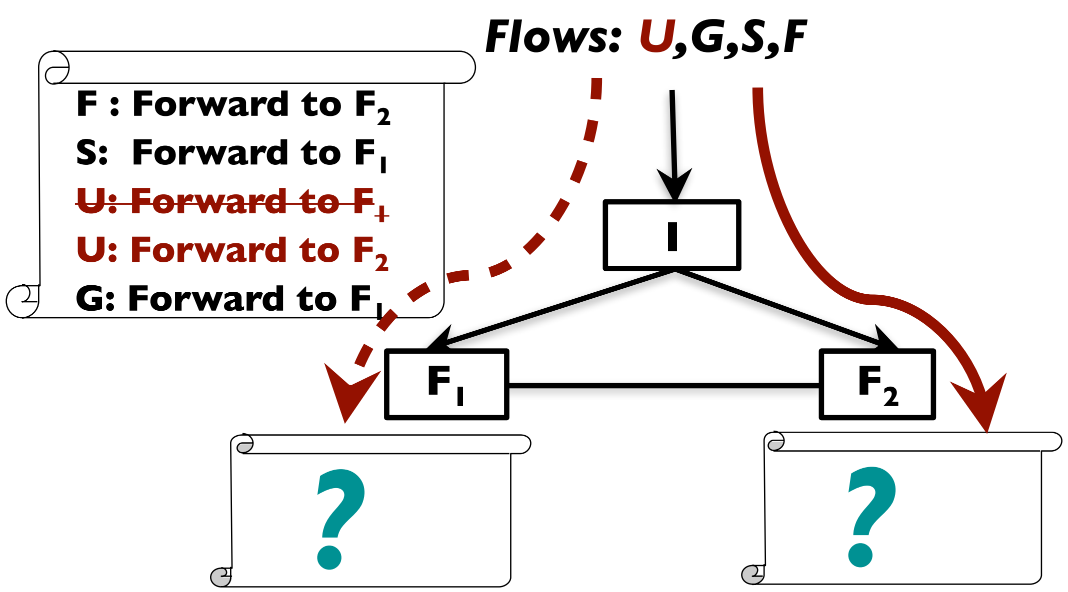




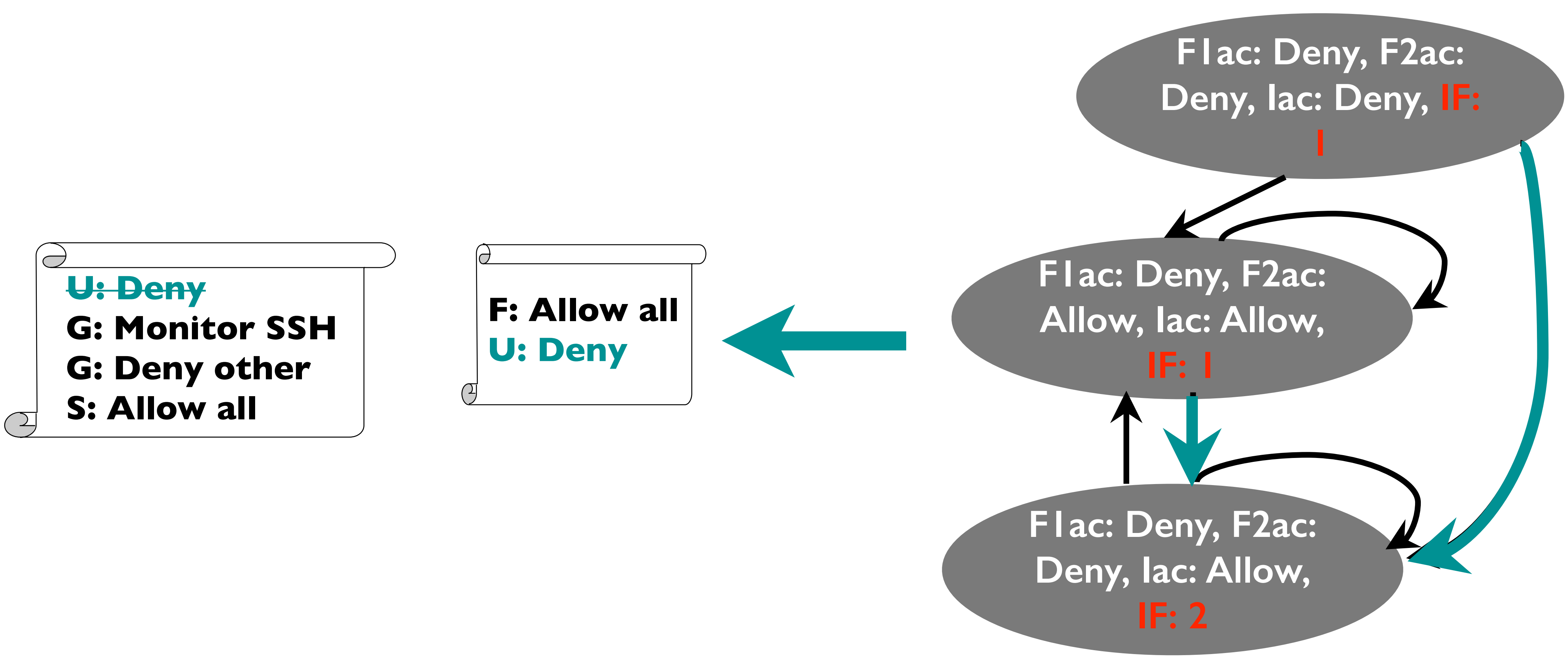
Circle has a winning strategy
Winning strategy (a_2,b_1), (a_2,b_2) a_2
Square has no winning strategy

Square has a winning strategy
Winning strategy (a_1,b_2) (a_2,b_2) b_2
Circle has no winning strategy

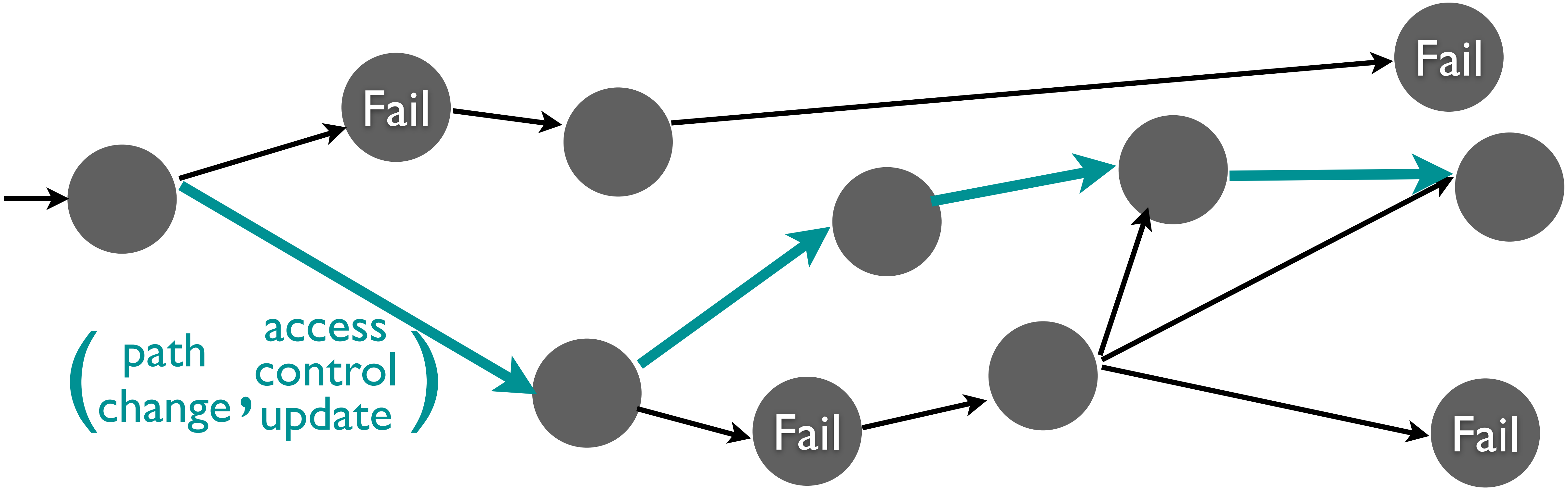
Example: synthesize access-control



- Input
  - Two player variables, system transitions, security invariant
- Output
  - A strategy with finite memory



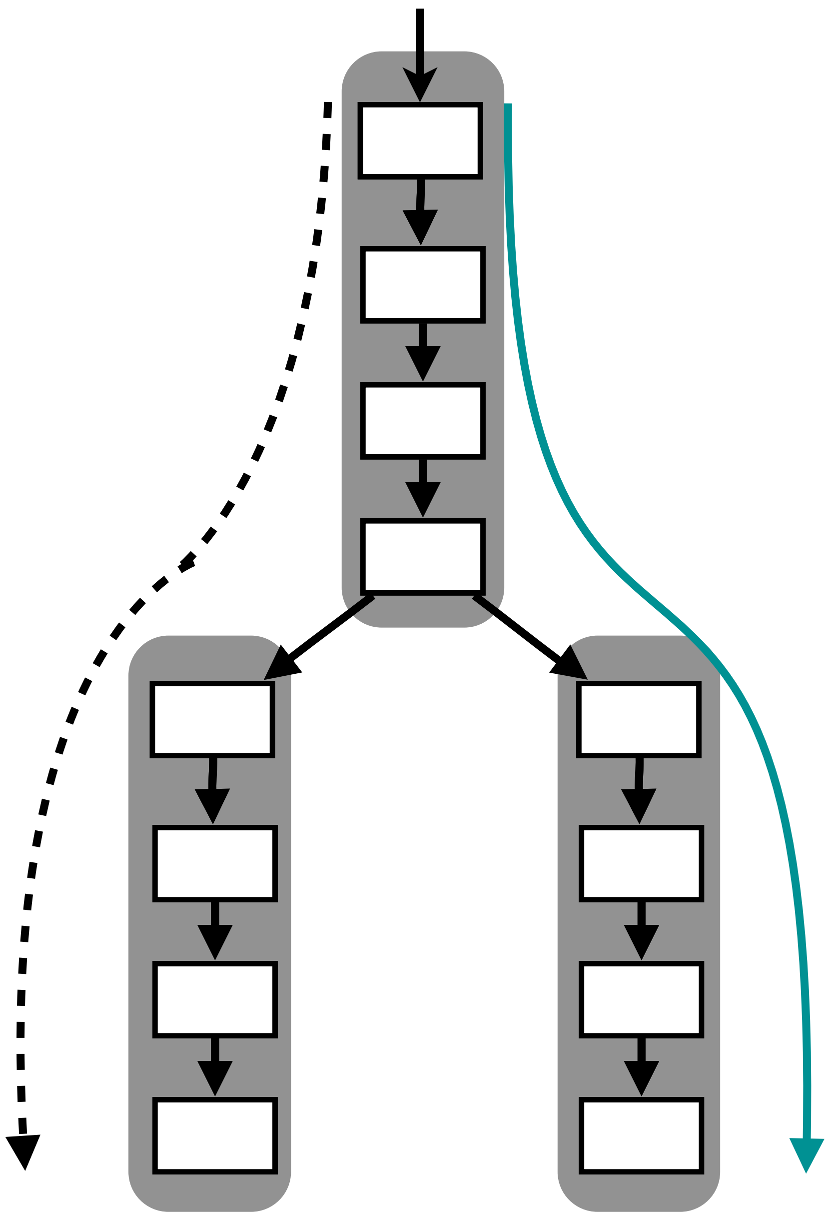
Synthesis is hard



- Synthesis for general temporal property is hard
  - (Relative) efficient for some properties
  - Safety (always avoid P), response (if P_1 then P_2), persistence (eventually stay at P), recurrence (infinitely often P)
- Need scaling technique ...

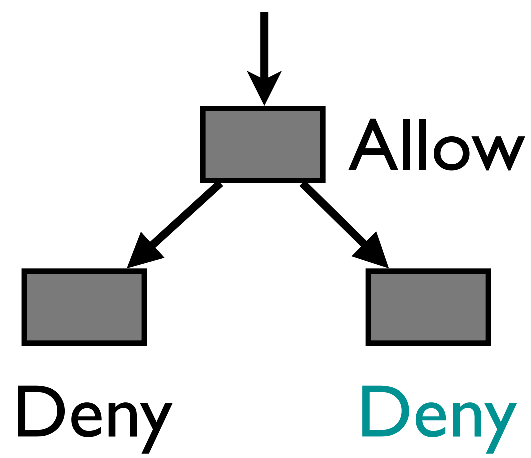
Scaling by abstraction

Large network *A*

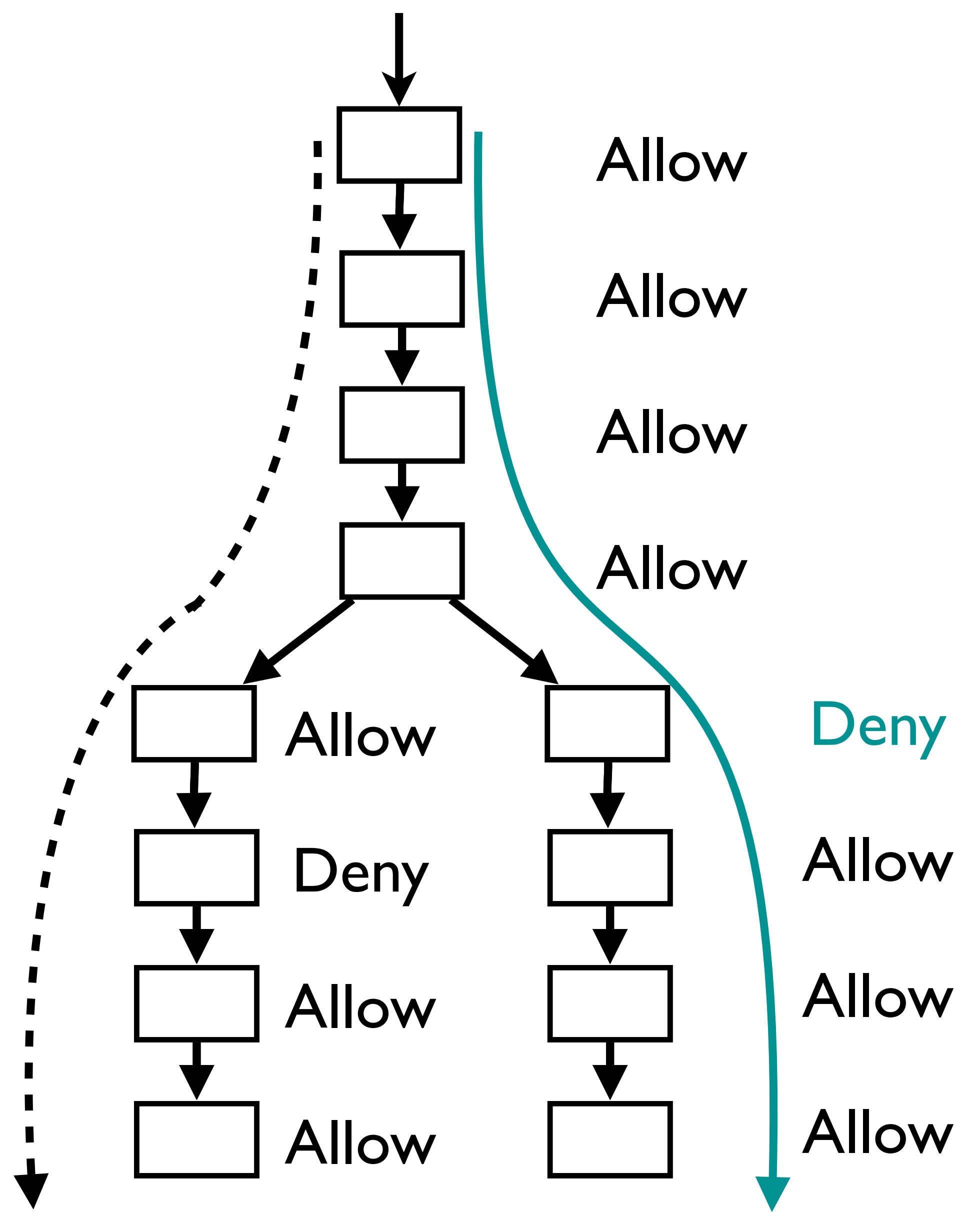


Network abstraction
Smaller network *B* that simulates *A*

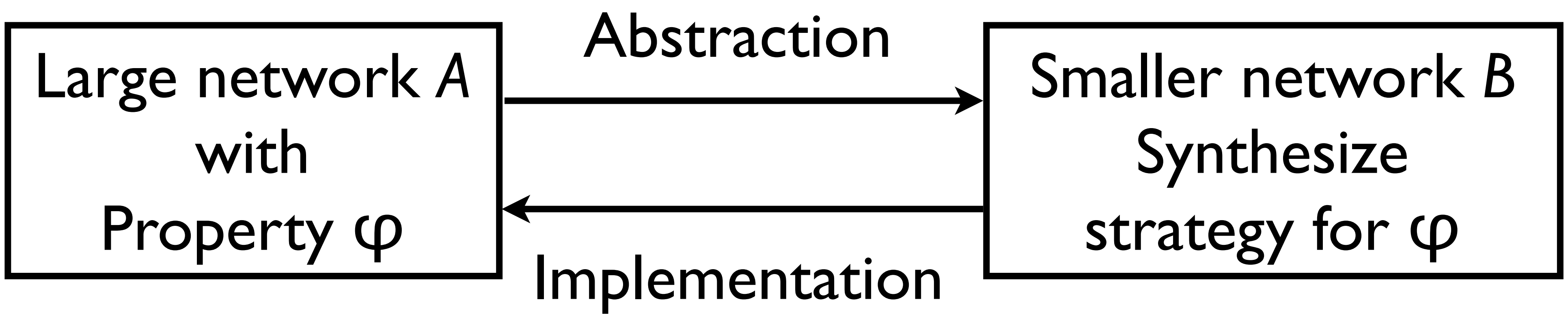
Perform synthesis
on abstract network



Implement synthesized solution on original network



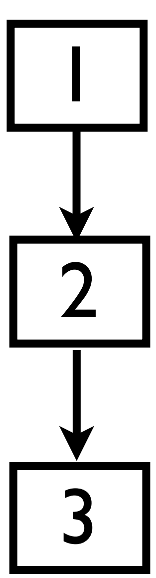
Synthesis by abstraction



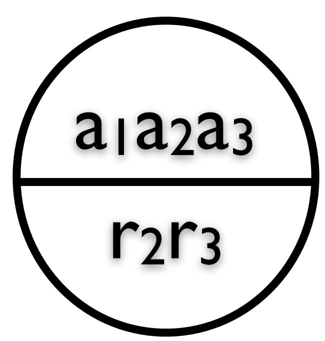
- Introduce network abstraction by simulation relation
- Simulation is a relation $R : A \rightarrow B$
  - Model *A*, *B* by transition system with observation
  - *R* maps states and transition in *A* to that in *B* with same observation
- Simulation *R* ensures φ synthesized for *B* is also preserved in A

Transition system model

- Transition system (V_0, V, T, O, H) for a network
  - Network states V (initial V_0), observable outputs O
  - Network transitions $T \subseteq V \times V$
  - Output function $H: V \rightarrow O$ maps each network state to its observable behavior relevant in synthesis



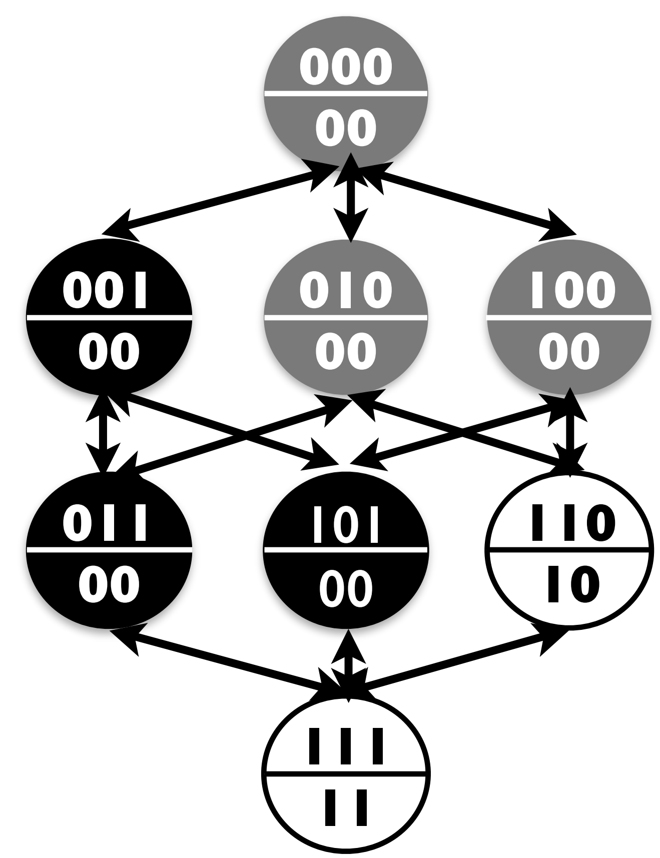
Network



V: a_1, a_2, a_3 are access control for 1,2,3

O: r_2, r_3 are reachability for 2,3

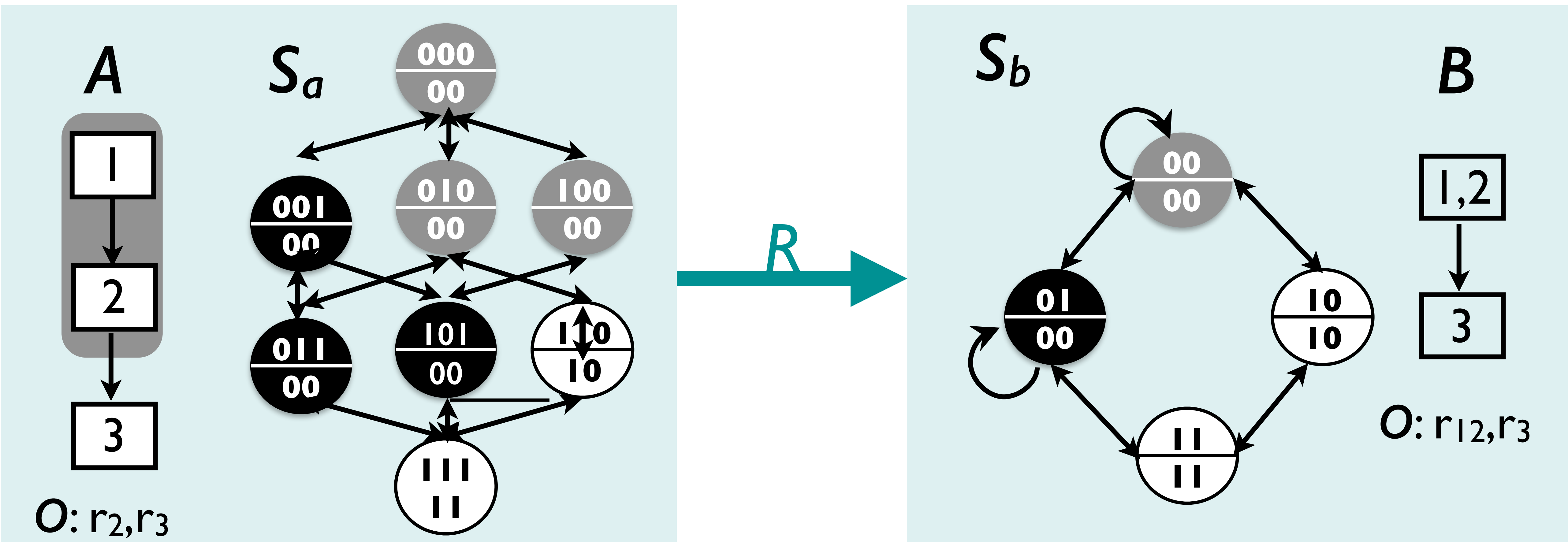
States



State transition

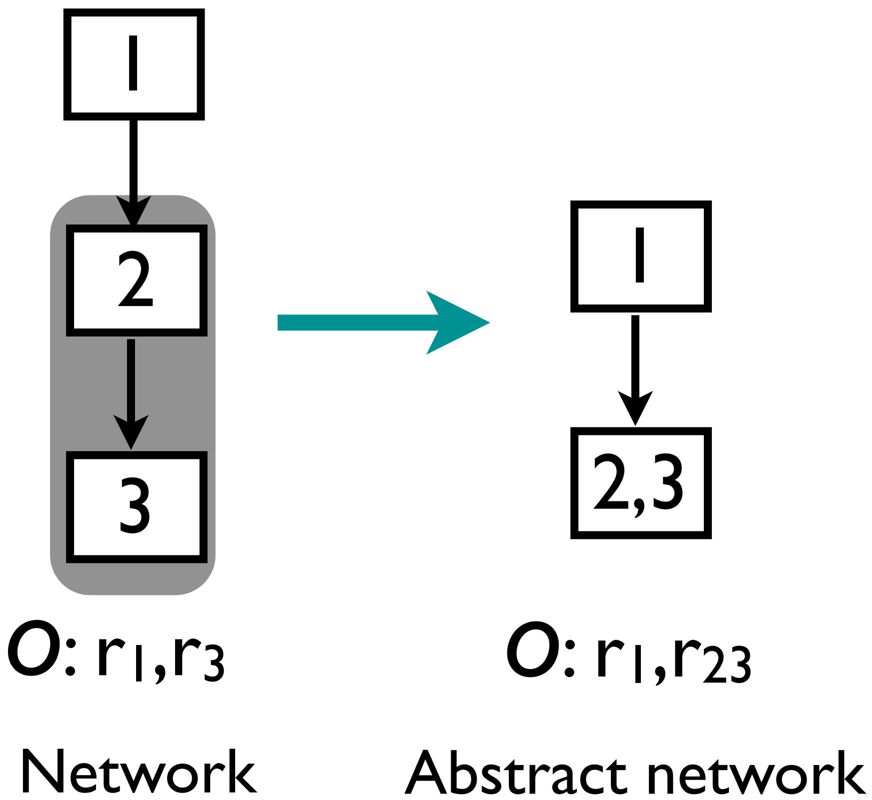
Simulation preserves synthesis property

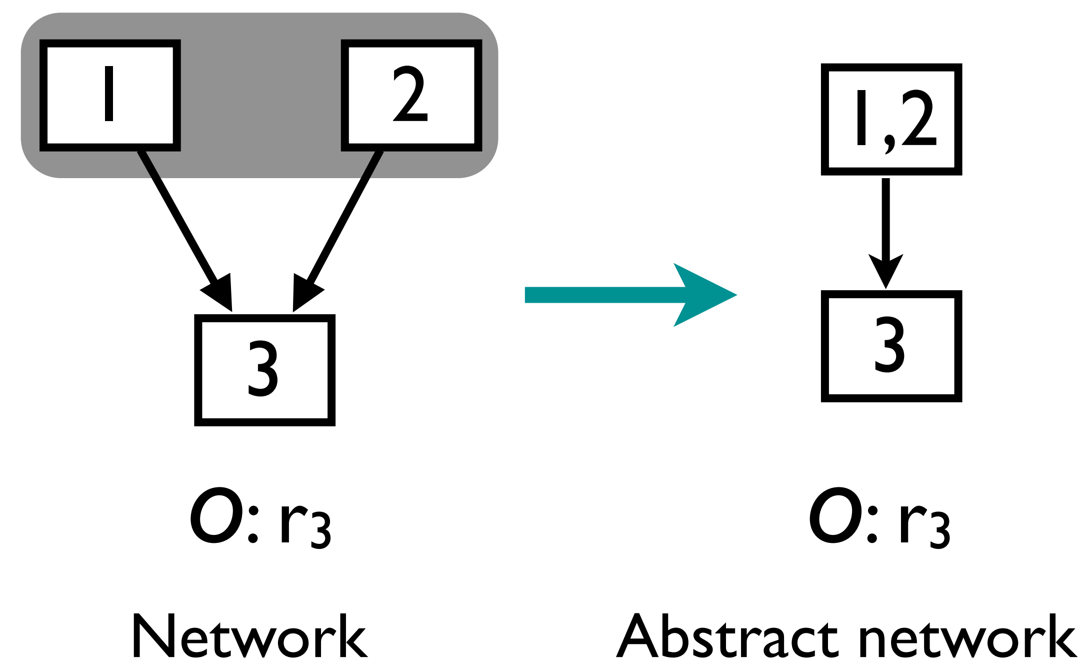
- Simulation from S_a to S_b is a relation *R that:*
  - *maps each $v_a \in V_a$ to some $v_b \in V_b$ with same output value*
  - *maps each transition $(v_a, v_a') \in T_a$ in S_a to some transition $(v_b, v_b') \in T_b$ in S_b*

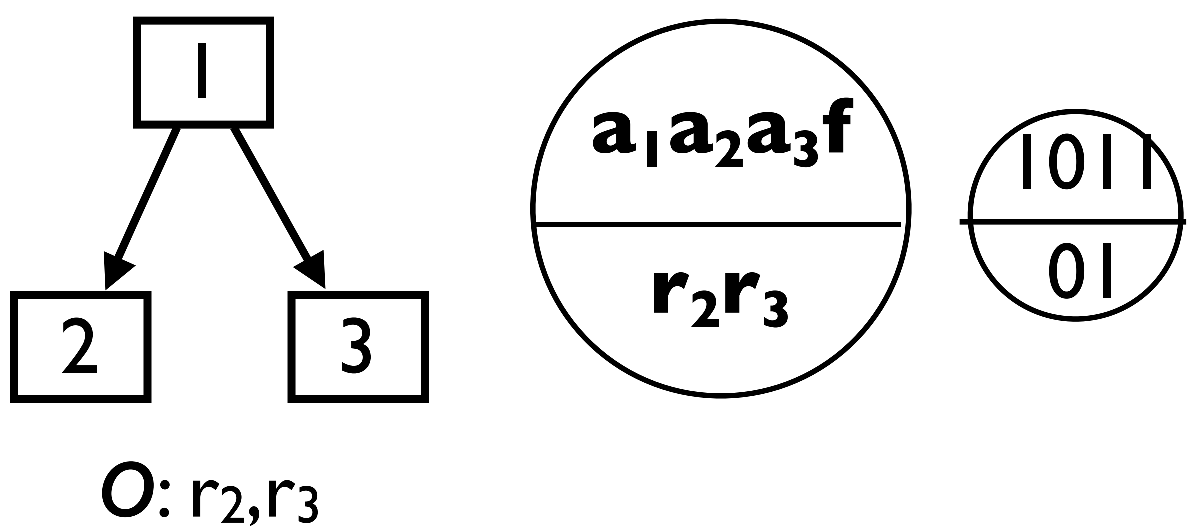


Theorem If S_a is simulated by S_b. Let φ be a LTL property over the output variables $O_a(O_b)$. Then, we have S_b satisfies φ implies S_a satisfies φ

More abstraction examples







No abstraction exists

Conclusion

- Construct provably correct configuration
  - Automate critical part of network management
- Formulate and solve reactive synthesis problem
  - Leverage off-the-shelf tools
- Scale by network abstraction
  - Propose network abstraction as simulation relation

Discussion

- Need killer app for synthesis
  - Look for complex network elements and properties
    - Middlebox, racing conditions, failure recovery
    - Extract properties from examples
  - Combine logical constraints and optimization goal
- Network abstraction
  - General patterns in edge configuration
  - Abstraction for composing distributed control