Synthesis of software-based protocols for dynamically reconfigurable networks

Ufuk Topcu (Univ of Pennsylvania)

www.seas.upenn.edu/~UTopcu

Outline:

- Reconfiguration in electric power networks on aircraft
- An academic testbed
- Newer thoughts: reconfiguration in software-based networks

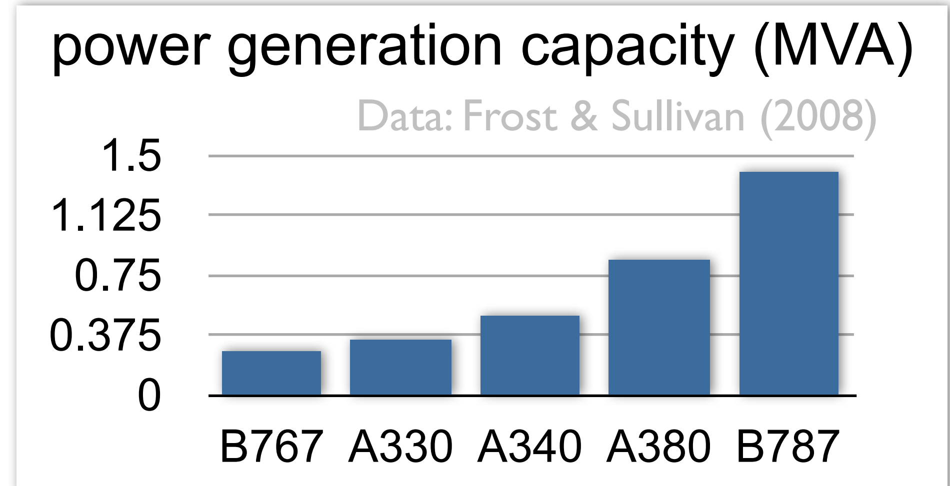
Collaboration with Murray, Ozay, Xu (Caltech), Rogersten (KTH) and Wang (UPenn)

References: CDC ’12, IEEE TII ’13 (review), HSCC ’13, CDC ’11

More-electric aircraft

Electrically powered systems (rather than pneumatically or hydraulically)

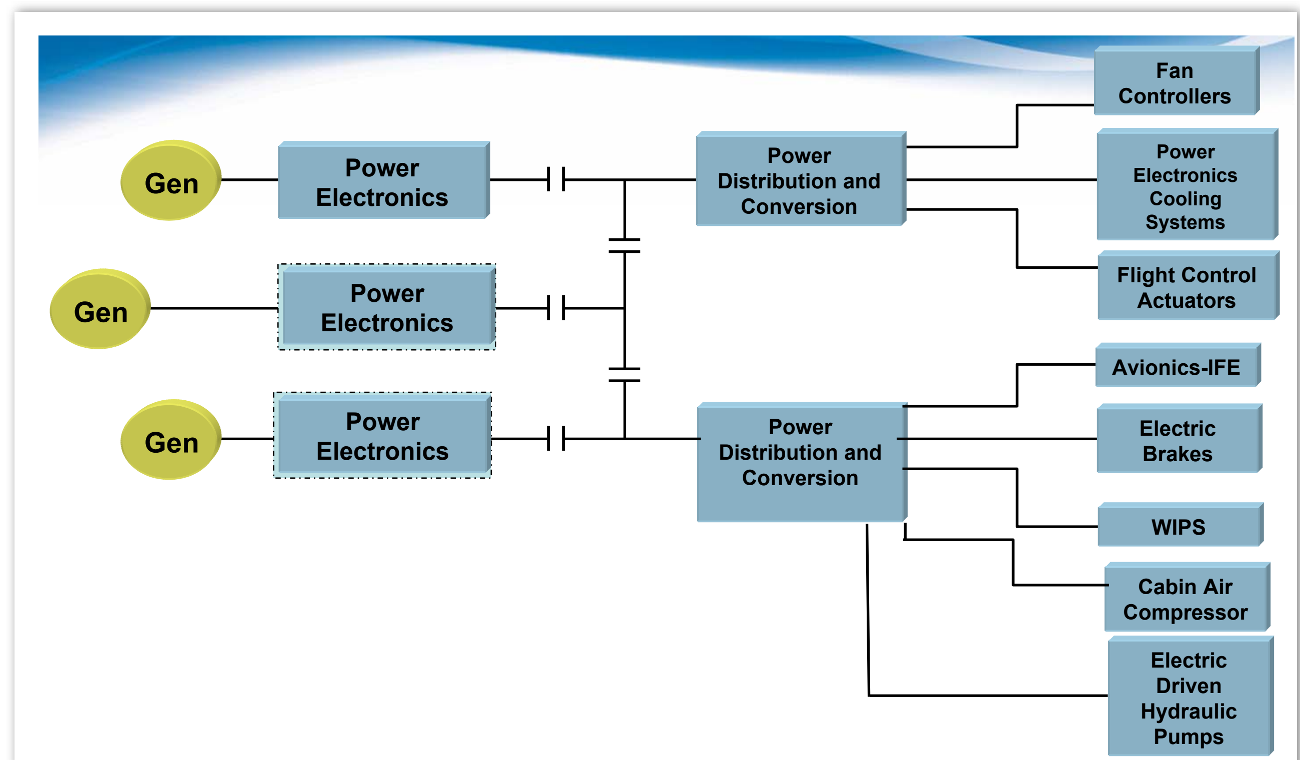
- air conditioning & cabin pressurization
- brakes & landing gear
- wing ice protection



Opportunities:

- More efficient
  - Less power off-takes from engines
  - Lower losses due to transfer
- Right function at the right time

- Weight reductions
  - Electrical systems heavier than conventional counterparts
  - System-level power and energy optimization is key.



Picture from: www.ece.cmu.edu/~electriconf/2008/PDFs/Karimi.pdf

Challenges:

- Safety-critical electric power system
- Distributed architectures
- Increased complexity

Single-line diagram -- electric power distribution

- Generation
  - Engines
  - APUs
  - External power
- Buses
  - AC vs DC
  - Essential vs non-essential
  - High vs low voltage
- Loads
- Transformers
- Rectifier units
- Contactors

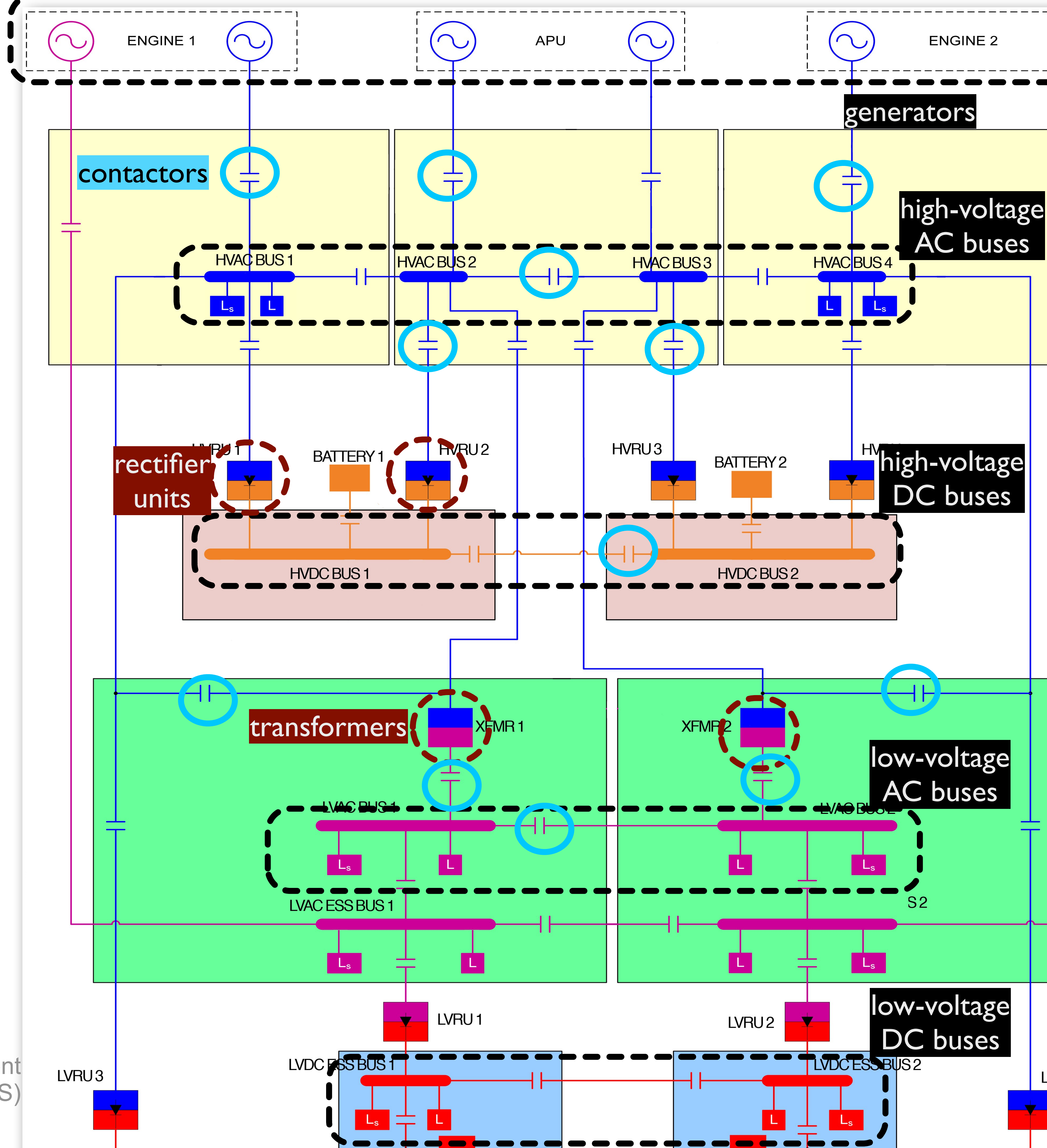


Figure adapted from US Patent 7439634 B2 by Rich Poisson (UTAS)

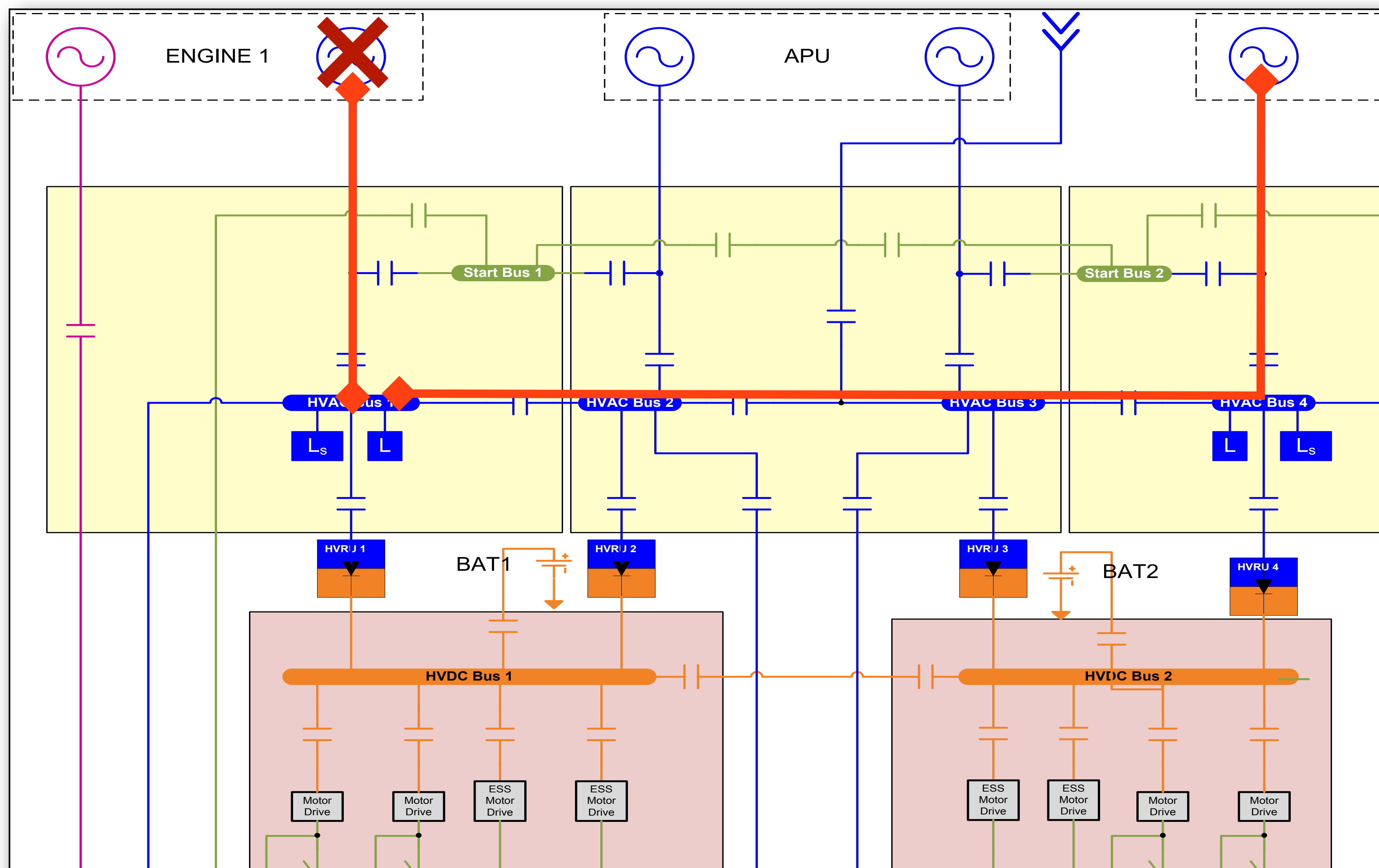
Dynamic reconfiguration

Reconfigure the network
by opening and/or close the contactors

in reaction to the changes in the environment

- health status of the components
- flight phase
- pilot requests

to satisfy safety and performance specifications.



Sample specifications

Requirements:

No AC bus shall be simultaneously powered by more than one AC source

Essential AC buses shall never be unpowered more than 50 msec

Priority	Bus 1	Bus 2	Bus 3	Bus 4
1	G_L	A_L	A_R	G_R
2	G_R	G_L	G_R	G_L
3	A_L	G_R	G_L	A_R
4	A_R	A_R	A_L	A_L

Do not exceed the capacity of the generator

Do not lose more than one bus for single failure

Buses shall be powered according to their priority tables

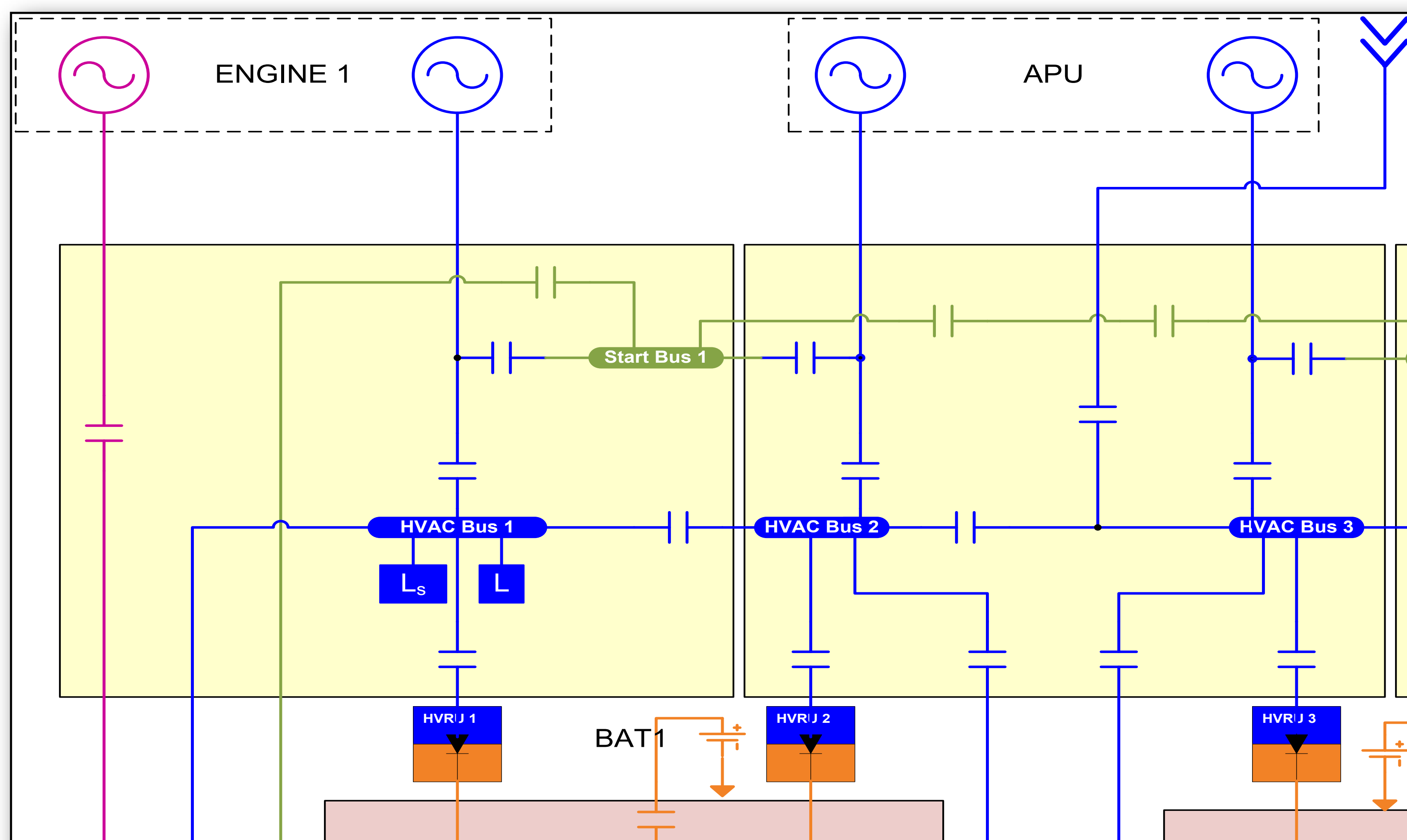
Bounds on the number and sequence of contactor switchings

Assumptions:

Known reliability of components

Worst-case bounds on contactor switching times

Typical failure modes to react to



Workflow

Abstract Model
Single Line Diagram
+
Formal Specifications
Temporal Logic

↓

TuLiP
Python-based toolbox for automatic controller synthesis

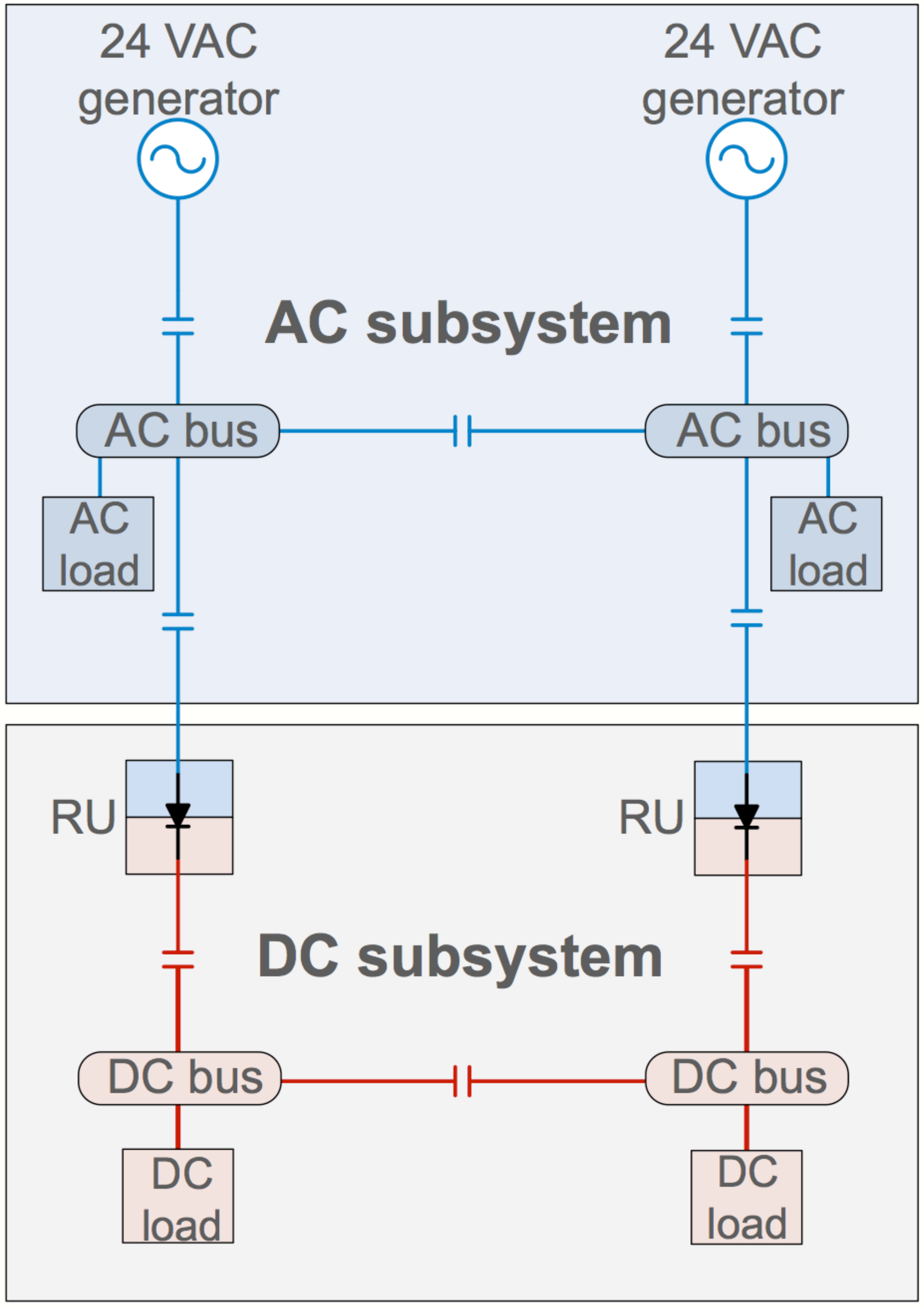
↓

Software Models
Simulink + Power Systems Toolbox

↓

Hardware Implementation

Single-line diagram for the testbed

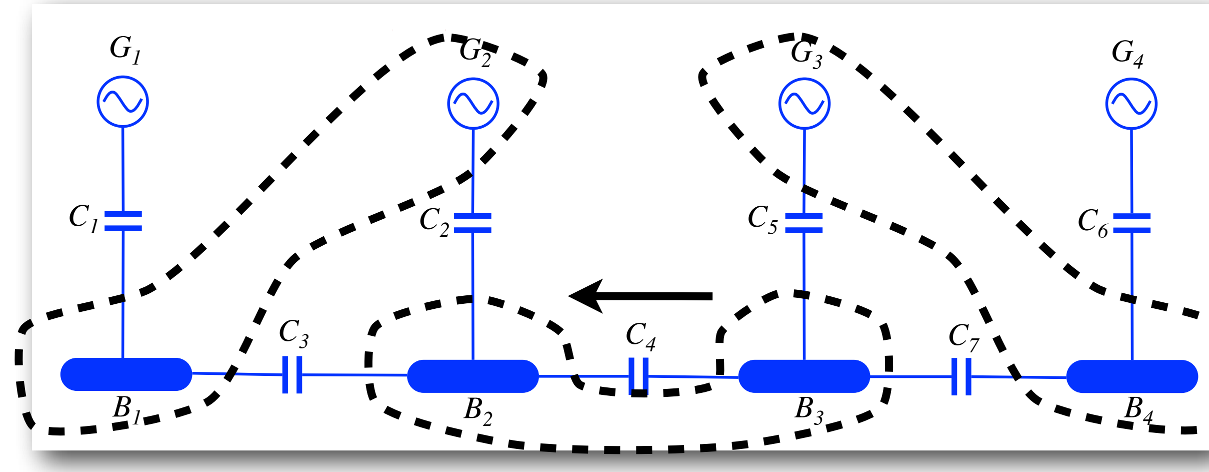


Formal specifications (a subset of them)

Requirements:

Always open contactors neighboring an unhealthy generator

$$\bigwedge_{G\in\mathcal{G}} \Box \left\{ (g=0) \rightarrow \bigwedge_{C\in\mathcal{C}_G} (\tilde{c}=0) \right\}$$



No paralleling

$$\bigwedge_{B\in\mathcal{B}_{AC}} \Box\neg \bigvee_{G\in\mathcal{N}(B), C\in\mathcal{C}_G} [(c=1)\wedge(c_B^1=1)]$$

$$\bigwedge_{C\in\mathcal{C}_b} \Box \left[\neg \left((b_C^1=1) \wedge \bigvee_{X\in\mathcal{N}(B_C^1)} (X=1) \right) \rightarrow \neg(\tilde{c}=-1) \right]$$

"Flow direction" through contactor

Buses are powered only if connected to a healthy generator or a powered bus

$$\bigwedge_{B\in\mathcal{B}} \Box \left\{ \left[\bigvee_{C\in\mathcal{C}_G, G\in\mathcal{N}(B)} ((c=1)\ \wedge\ (g=1)) \right] \rightarrow (b=1) \right\}$$

Bounded duration of "unpoweredness" of essential buses -- introduce a clock

$$\Box\left\{ \theta_B \leq \frac{T}{\delta t} \right\}$$

Assumptions:

At least one of the generators is always healthy

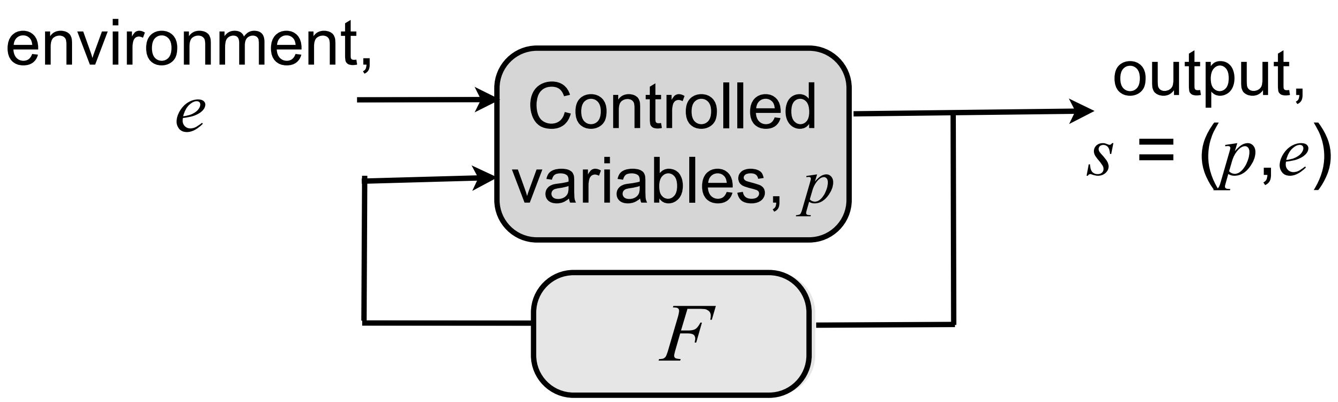
$$\Box\left\{ \bigvee_{G\in\mathcal{G}} (g=1) \right\}$$

Reactive synthesis as a two-player temporal logic game

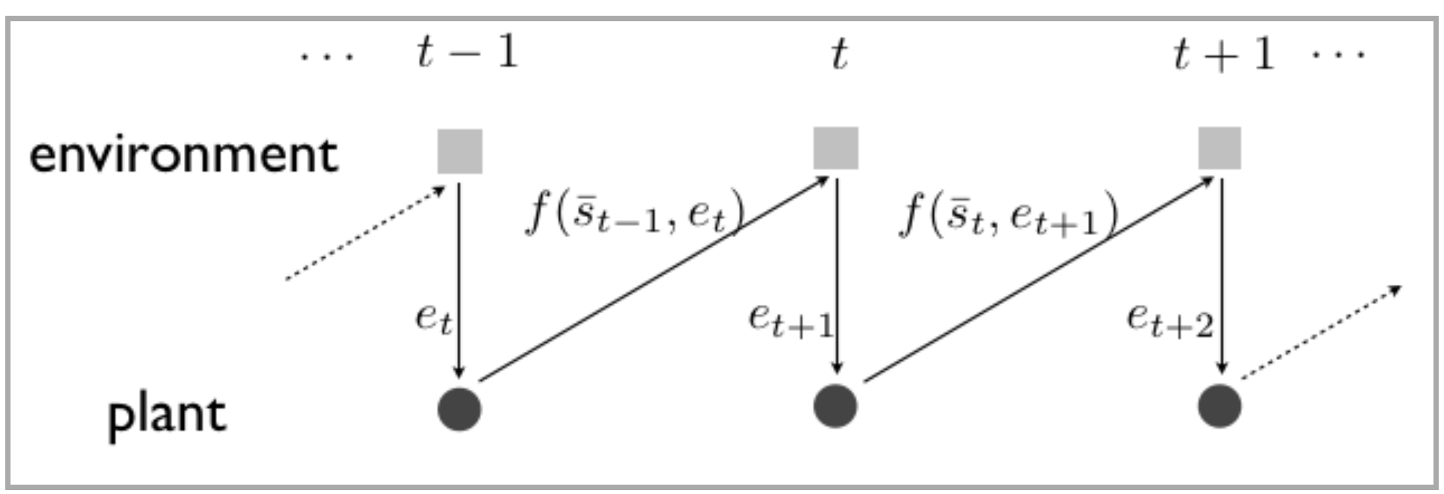
Given:

- Environment (e) and controlled (p) variables over finite domains
- Temporal logic specification $\varphi(e,p)$ = $\varphi_e \rightarrow \varphi_s$

Find: A map F such that realizes the specification.



Can be formulated as a game between the environment and the system.

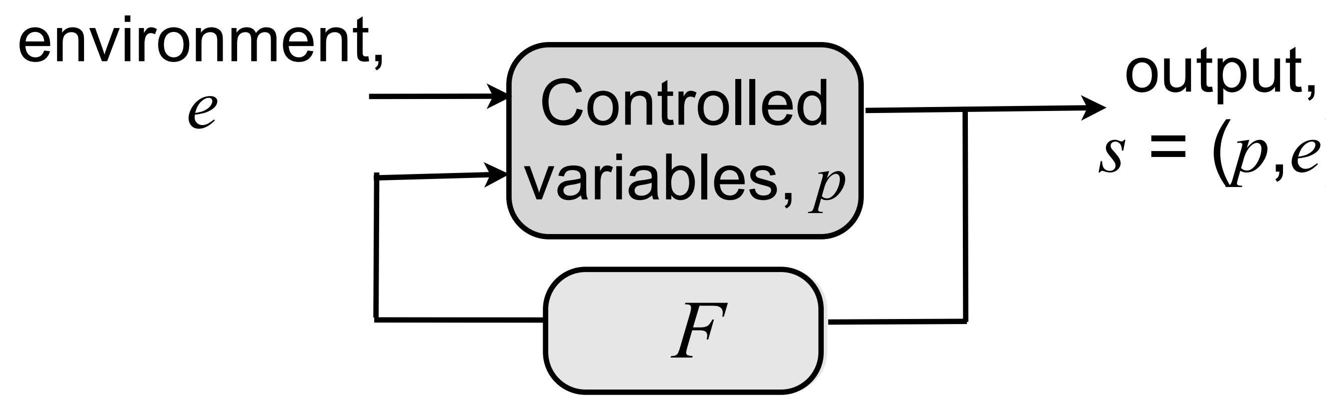


Reactive synthesis as a two-player temporal logic game

Given:

- Environment (e) and controlled (p) variables over finite domains
- Temporal logic specification $\varphi(e,p) = \varphi_e \rightarrow \varphi_s$

Find: A map F such that realizes the specification.



Solving the game:

- Intractable for general LTL
- Polynomial complexity for GR[1] specifications
  [Piterman et al., 2007&2011]

Both sides are of the form ($\alpha \in \{e, s\}$):

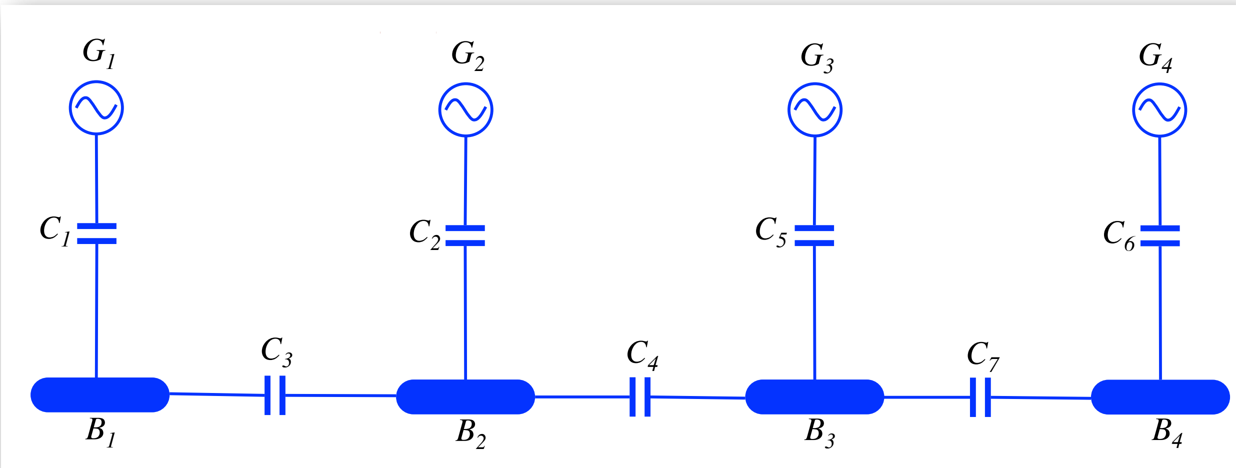
$$\varphi_\alpha = \underbrace{\theta^\alpha_{init}}_{\text{initial conditions}} \wedge \underbrace{\bigwedge_{i \in K_\alpha} \Box \psi^\alpha_i}_{\text{safety + transitions}} \wedge \underbrace{\bigwedge_{i \in L_\alpha} \Box \Diamond J^\alpha_i}_{\text{fairness + goals}}$$

initial conditions

safety + transitions (always)

fairness + goals (always eventually)

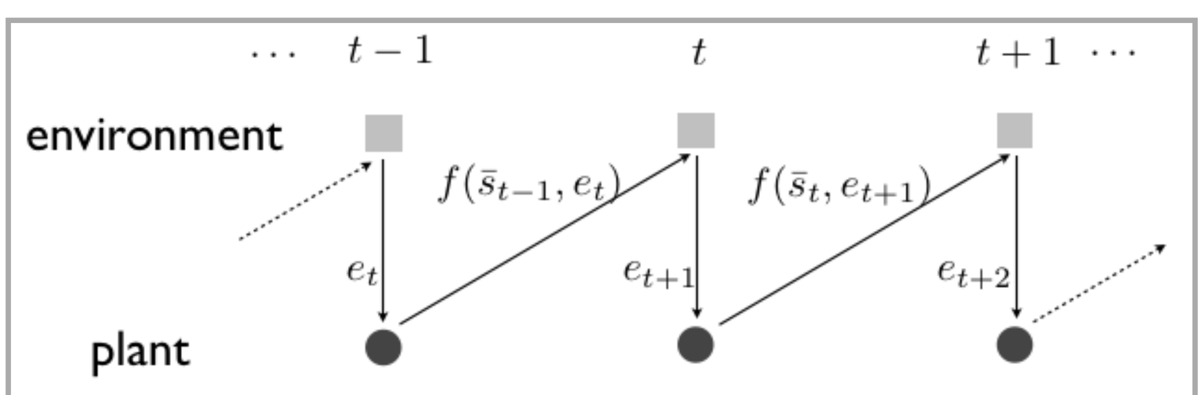
Structure of the controller



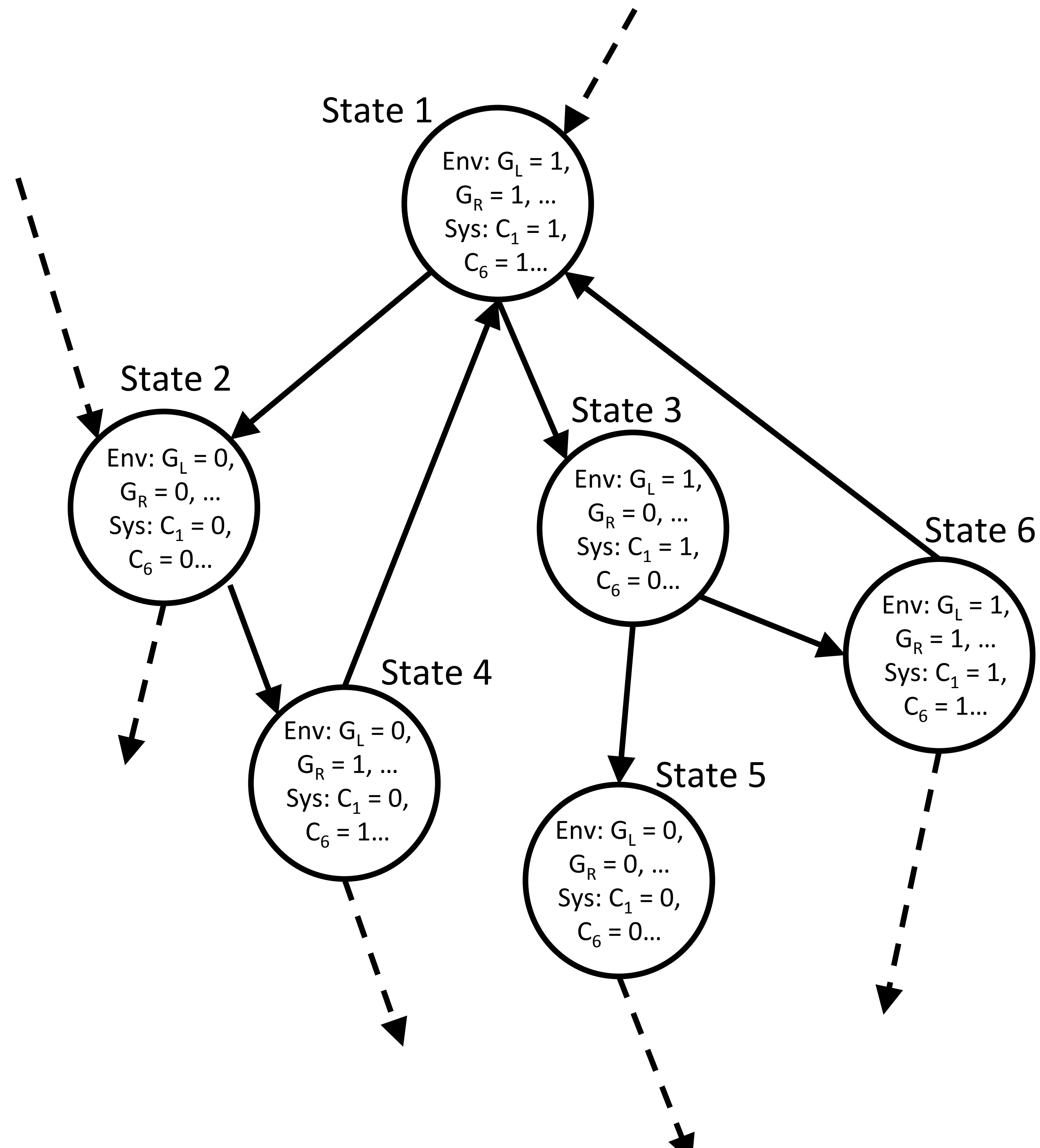
e: health status of the generators
p: contactor status & bus powered
$s = (e, p)$

Strategy:

$$f : (s_0 s_1 \dots s_t, e_{t+1}) \mapsto p_{t+1}$$

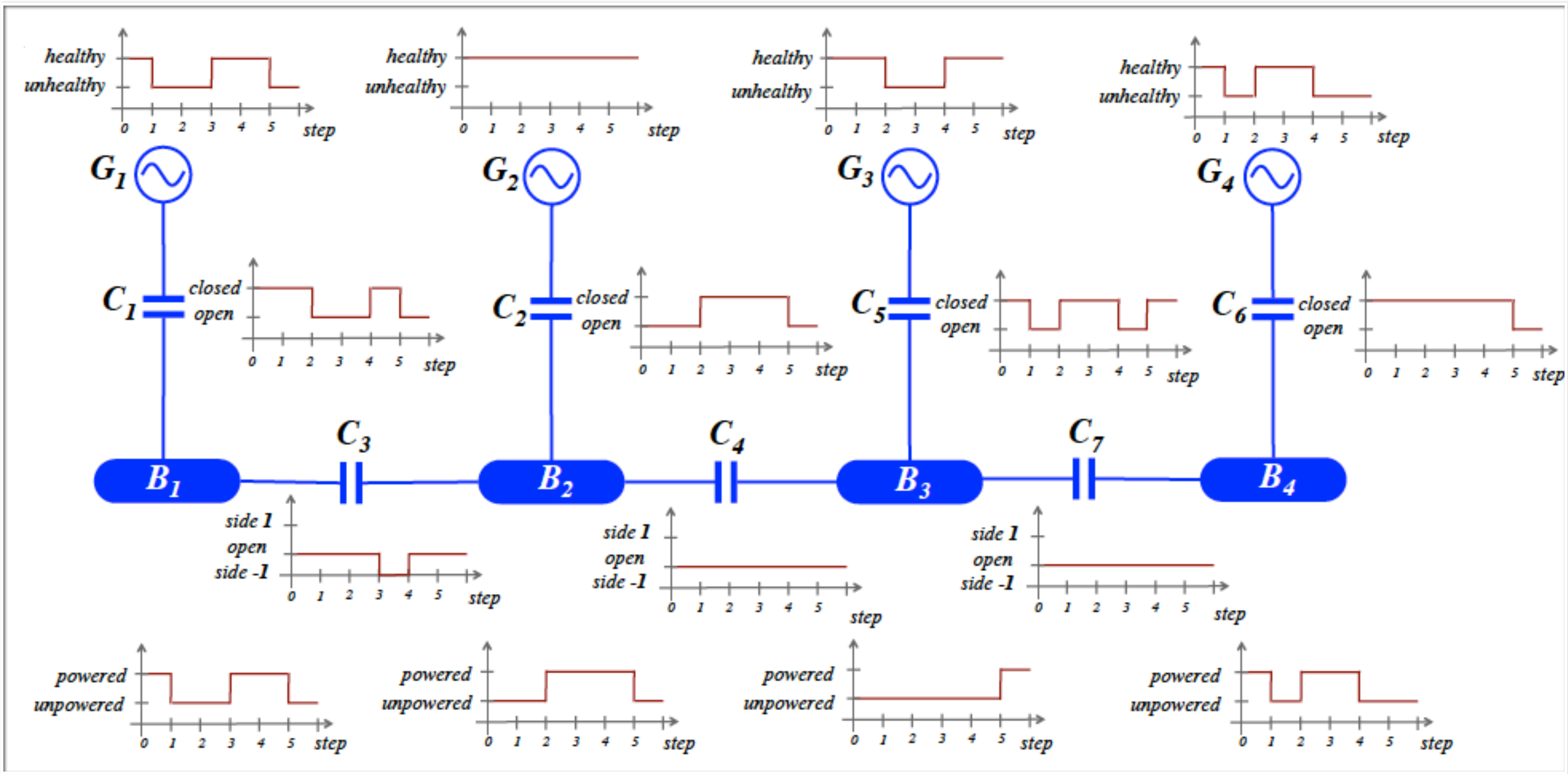


Automaton representation:

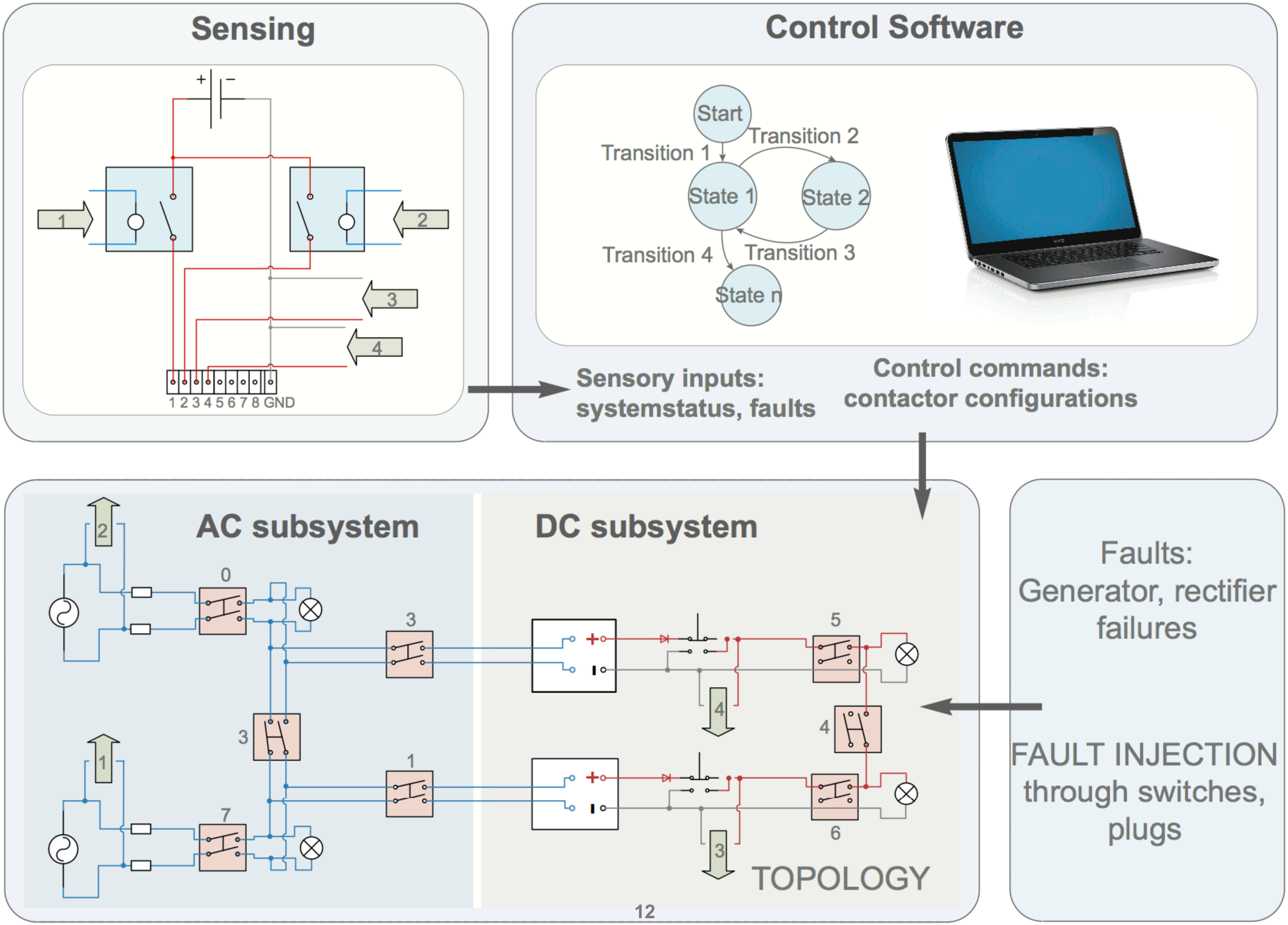


A sample simulation -- sanity check

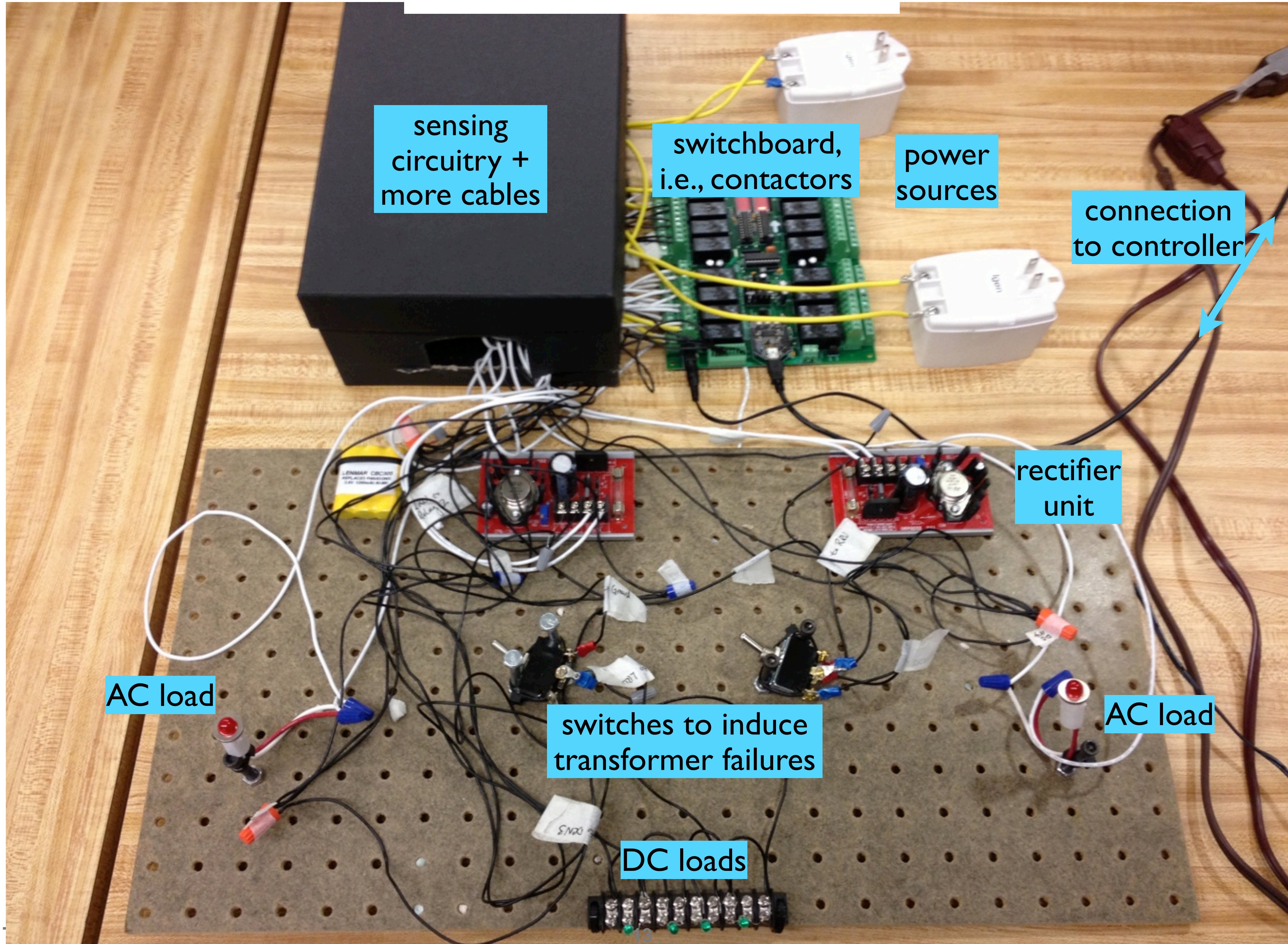
Given arbitrary, admissible environment signals, read the system outputs from the automaton



Overview of testbed functionality

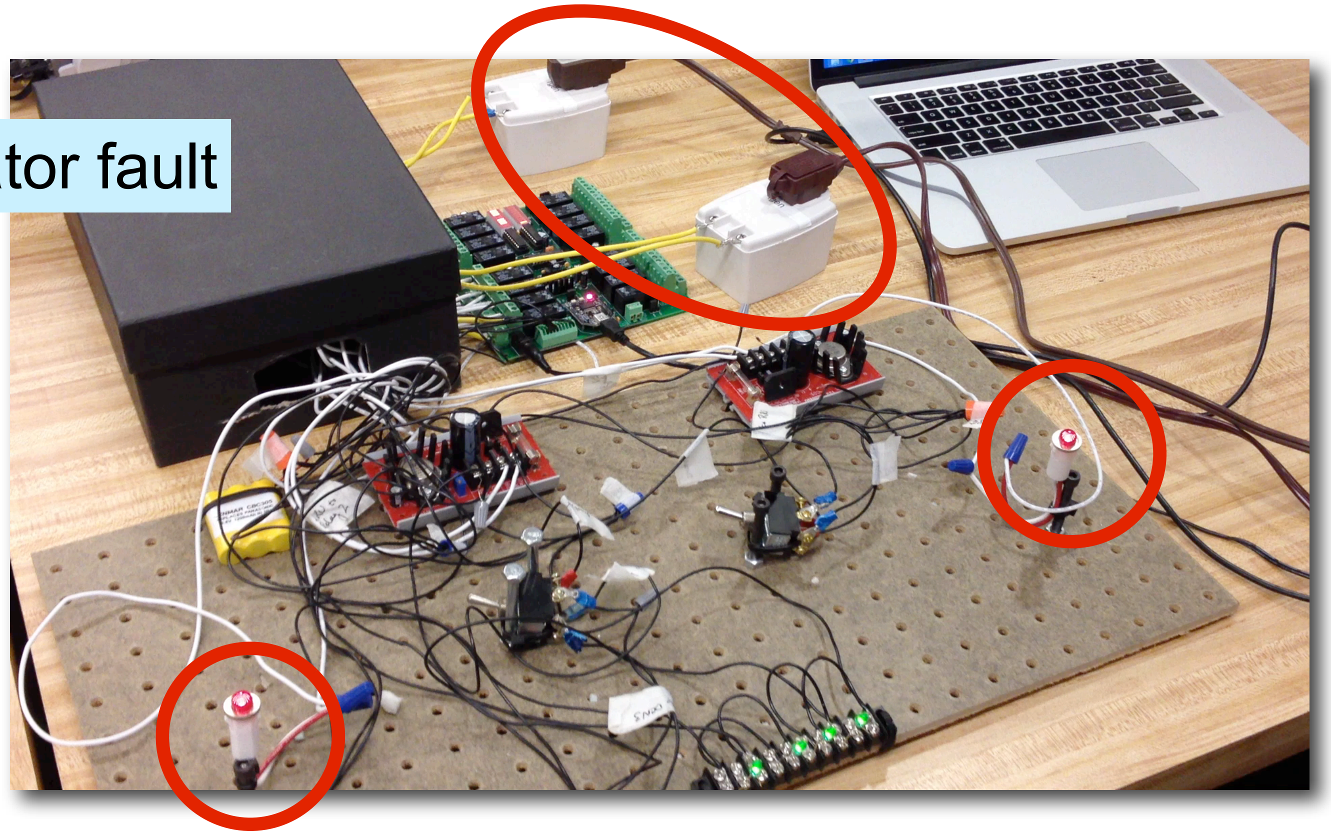


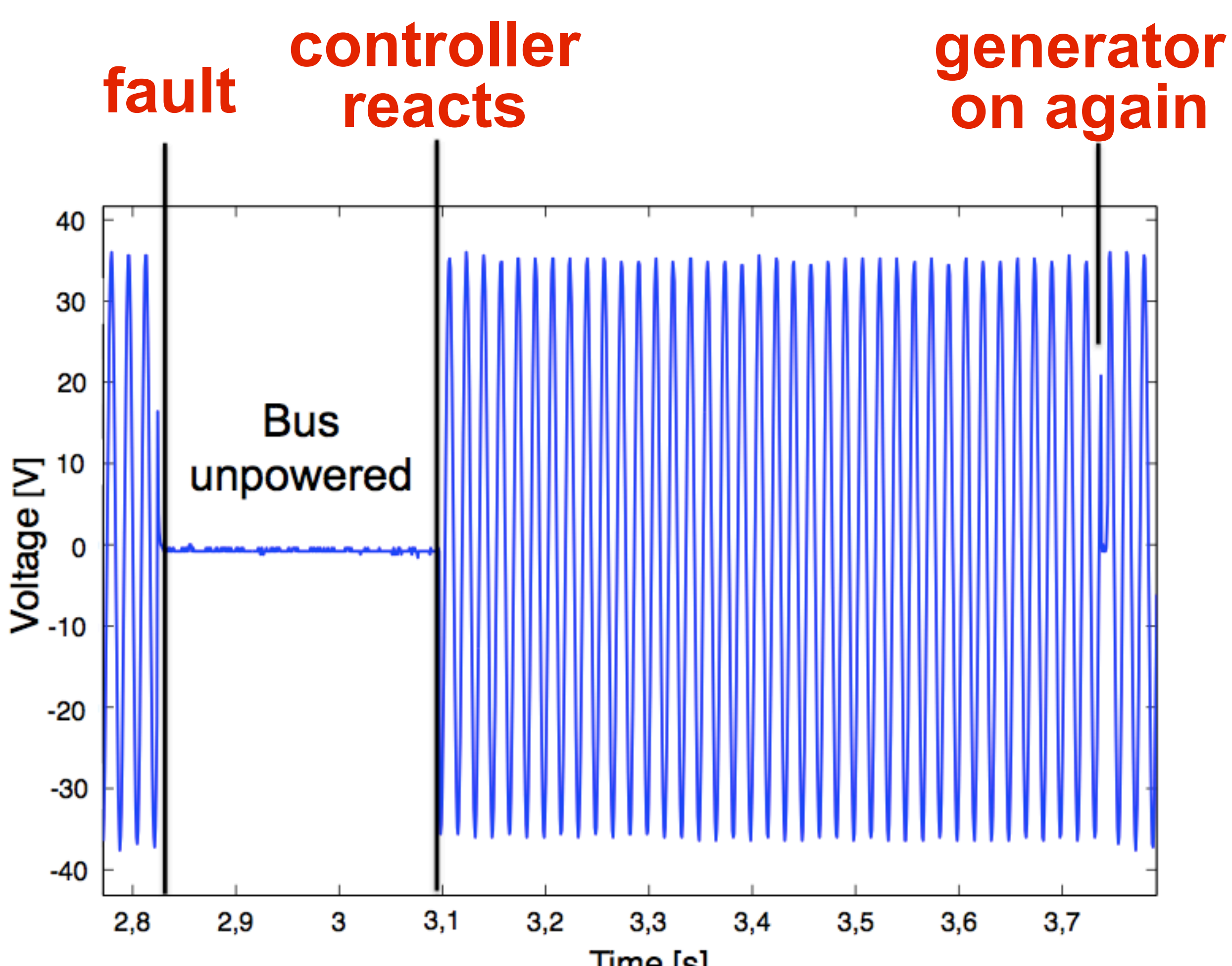
A look of the testbed

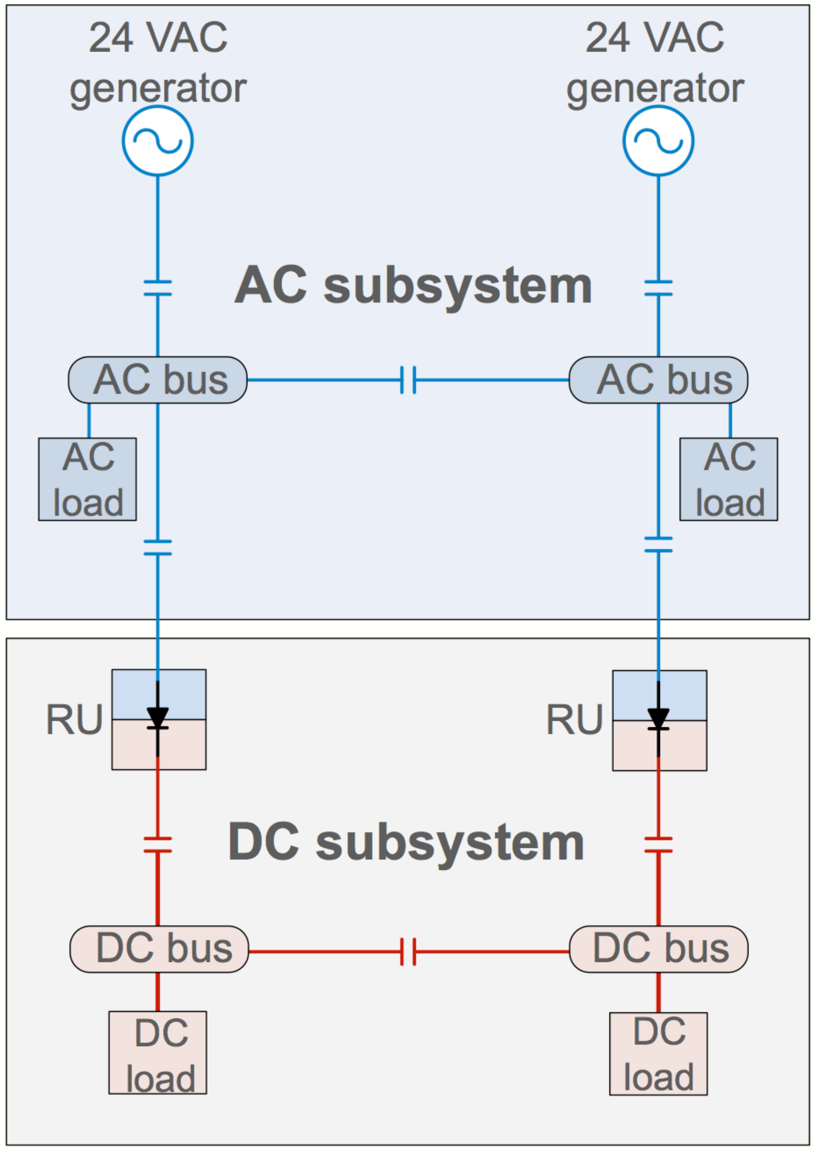


Hardware tests -- normal operation

AC generator fault

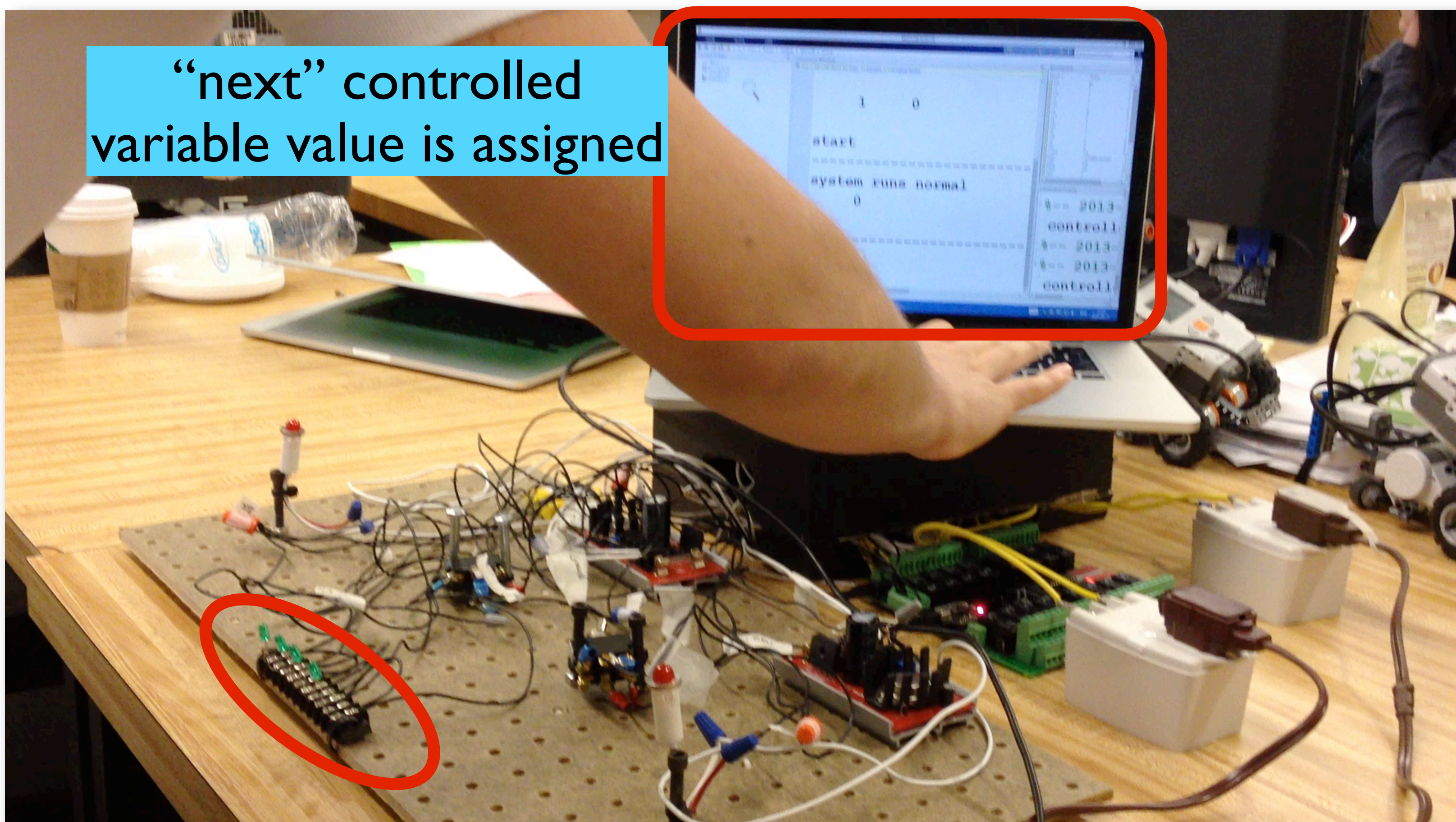


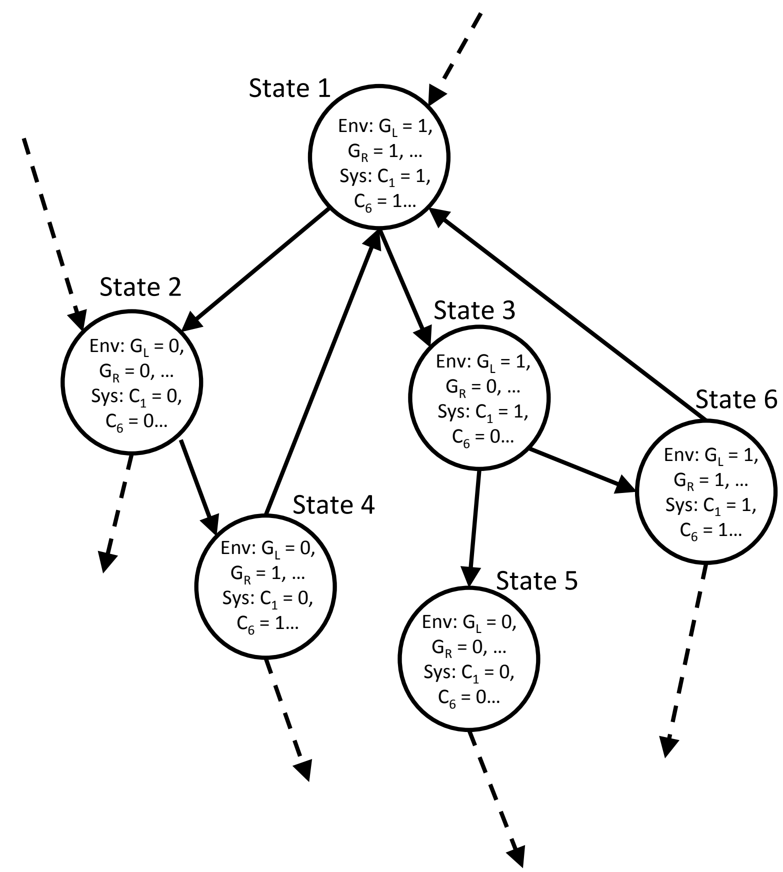




Hardware tests -- environment assumptions violated

Both transformers become unhealthy simultaneously---violating an assumption---and the controller cannot assign a “next” value for the controlled variables.





Limitations and lessons

Sensing and perception are important and often ignored.

- Matching the sensing modalities in theory and practice
- Limitations in sensing
- Uncertainties in perception

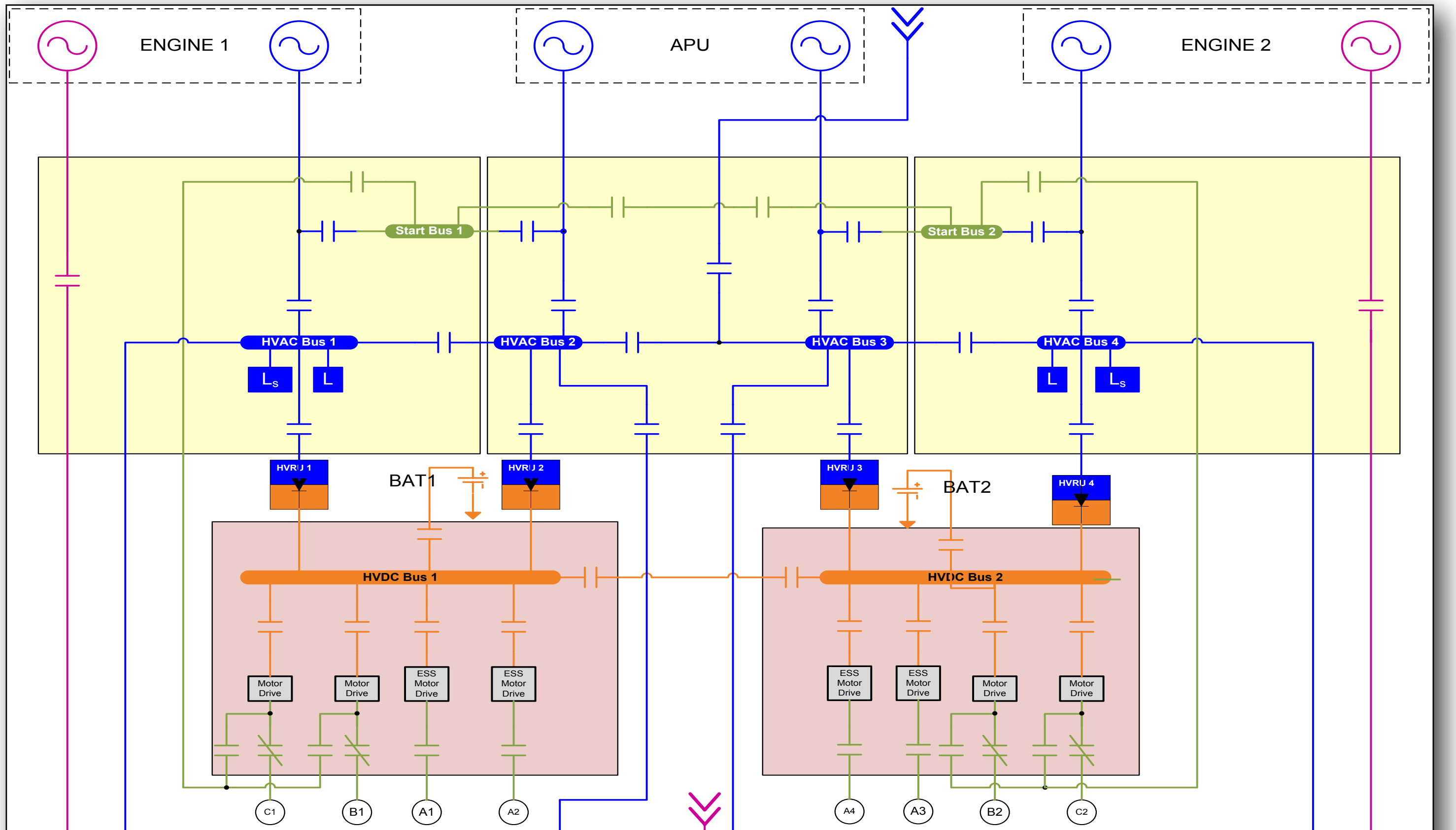
Have ignored most of the hard timing constraints

Have mostly ignored the underlying dynamics

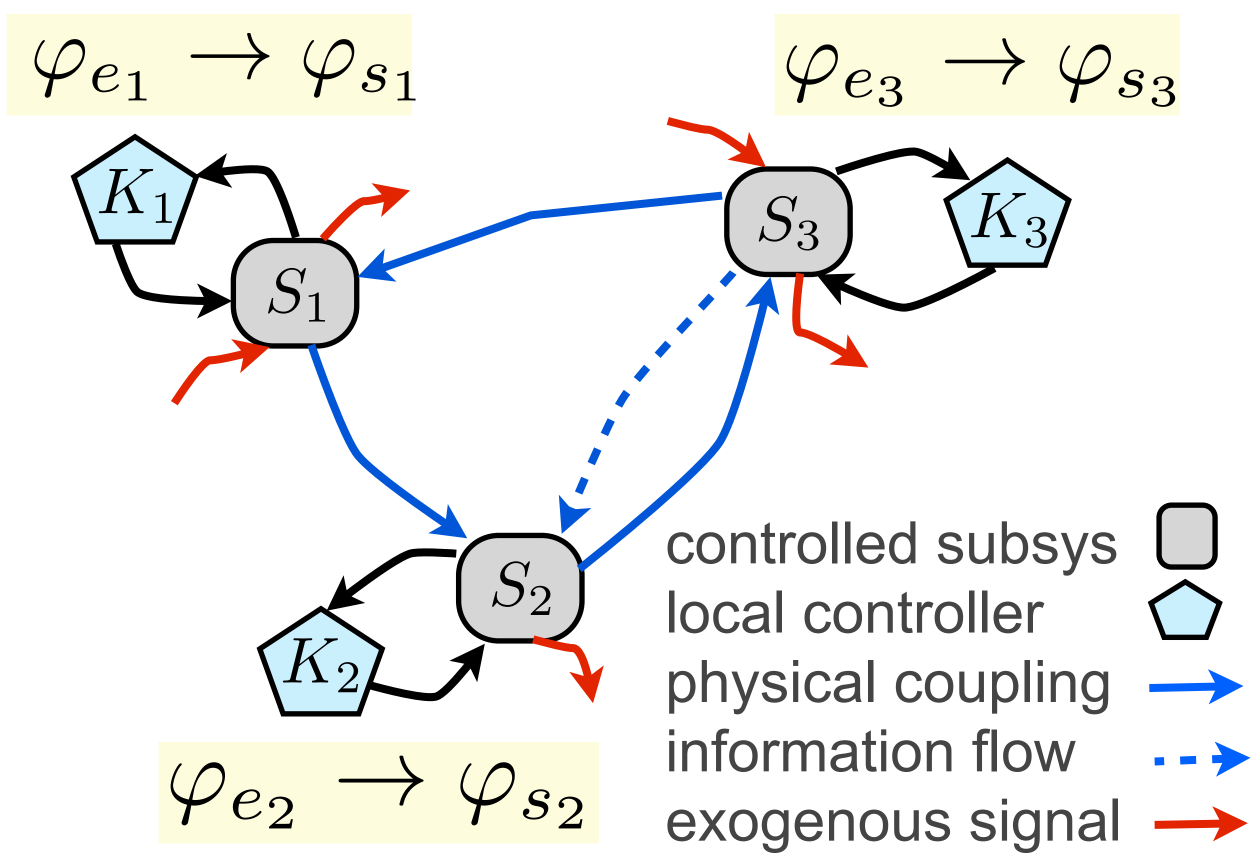
- Seems to work fine at this level of detail
- In general, need for hierarchical control

Controller structure is key for...

- Reliability
- Scalability



Compositional synthesis of distributed protocols



$$\underbrace{\wedge_i \varphi_{e_i} \to \varphi_e}_{\text{"weaker" environment assumptions}} \to \underbrace{\varphi_s \to \wedge_i \varphi_{s_i}}_{\text{"stronger" system requirements}}$$

<u>Extra (mild) technical conditions</u>: No common controlled variables & loops are well-posed.

Fact: $\varphi_e \to \varphi_s$ is realizable if every $\varphi_{e_i} \to \varphi_{s_i}$ is realizable.

Contracts formalize information exchange,...

- design-time---between the design teams---and
- run-time---between the subsystems.

Distributed controllers for the power network

Master (SYS2) / **Slave** (SYS1):

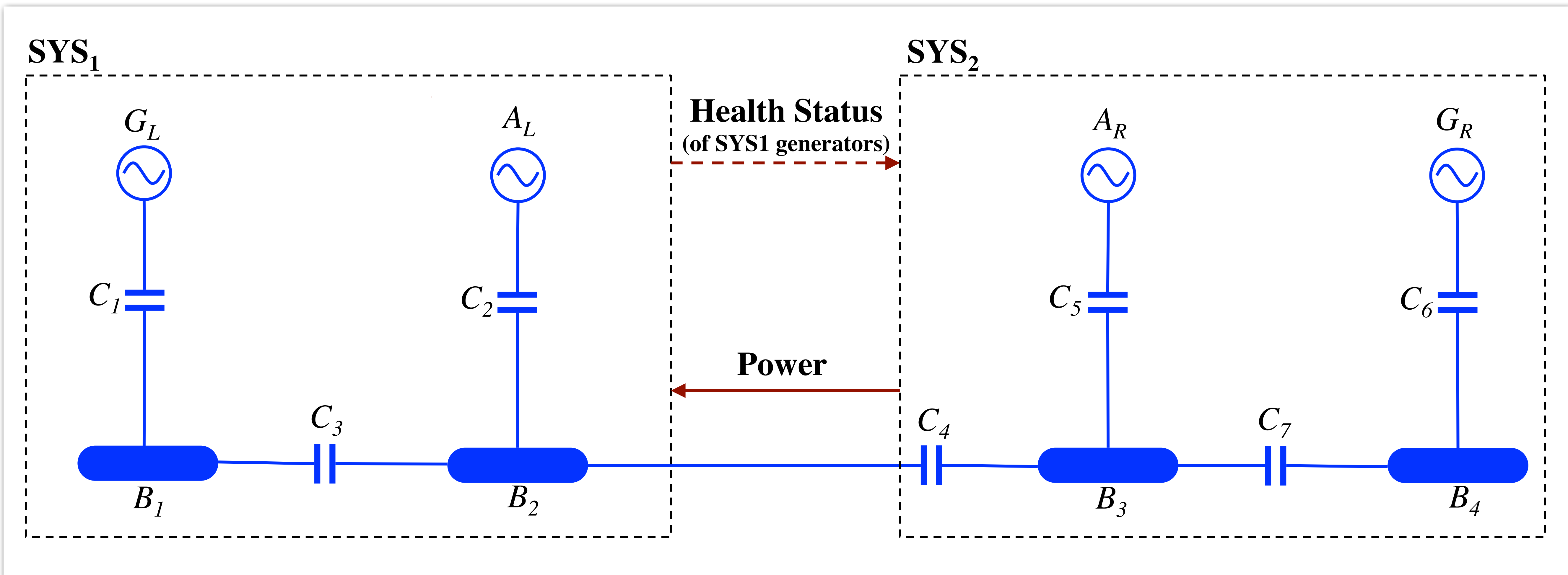
- Uni-directional power flow (SYS2 → SYS1)
- Assume always A_R or G_R healthy
- SYS2 sees the health status of SYS1
- Make B_3 an essential bus

Decentralized:

- Bi-directional power flow
- Restrictions to avoid deadlock
- Make B_2 and B_3 an essential bus
- Additional assumptions on both sides

$$\square(G_L = 0 \wedge A_L = 0 \rightarrow C_4 = -1)$$

$$\square(G_R = 0 \vee G_L = 0 \vee B_2 = 1)$$



Application to the testbed

Specifications naturally decompose into AC and DC parts.

assumptions:

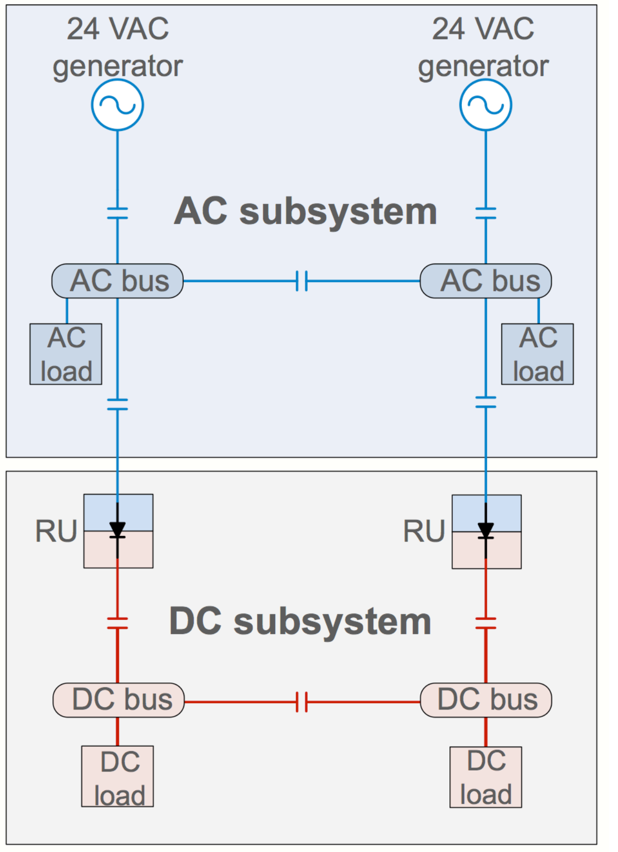
$$\square(((gen_1 = healthy) \vee (gen_2 = healthy)) \wedge ((ru_1 = healthy) \vee (ru_2 = healthy))),$$

non-paralleling of AC buses:

$$\square\neg((c_1 = closed) \wedge (c_2 = closed) \wedge (c_3 = closed))$$

strict timing constraints:

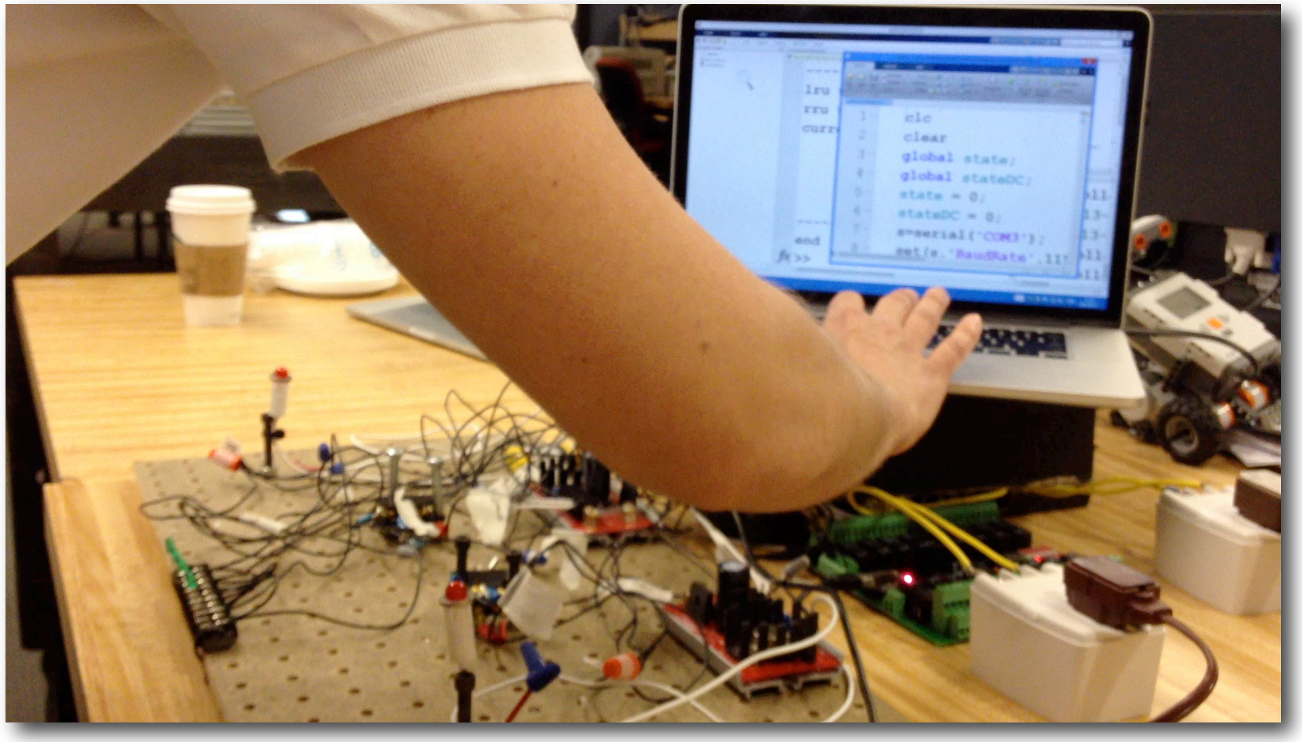
$$\square(b_i = powered), \text{ for } i \in \{1, 2, 3, 4\}$$



Assume: Rectifier units have capacitors that can power the DC buses for some time T>0 and generators stay healthy (once they become) for longer than T.

Impose extra assumption on DC side:

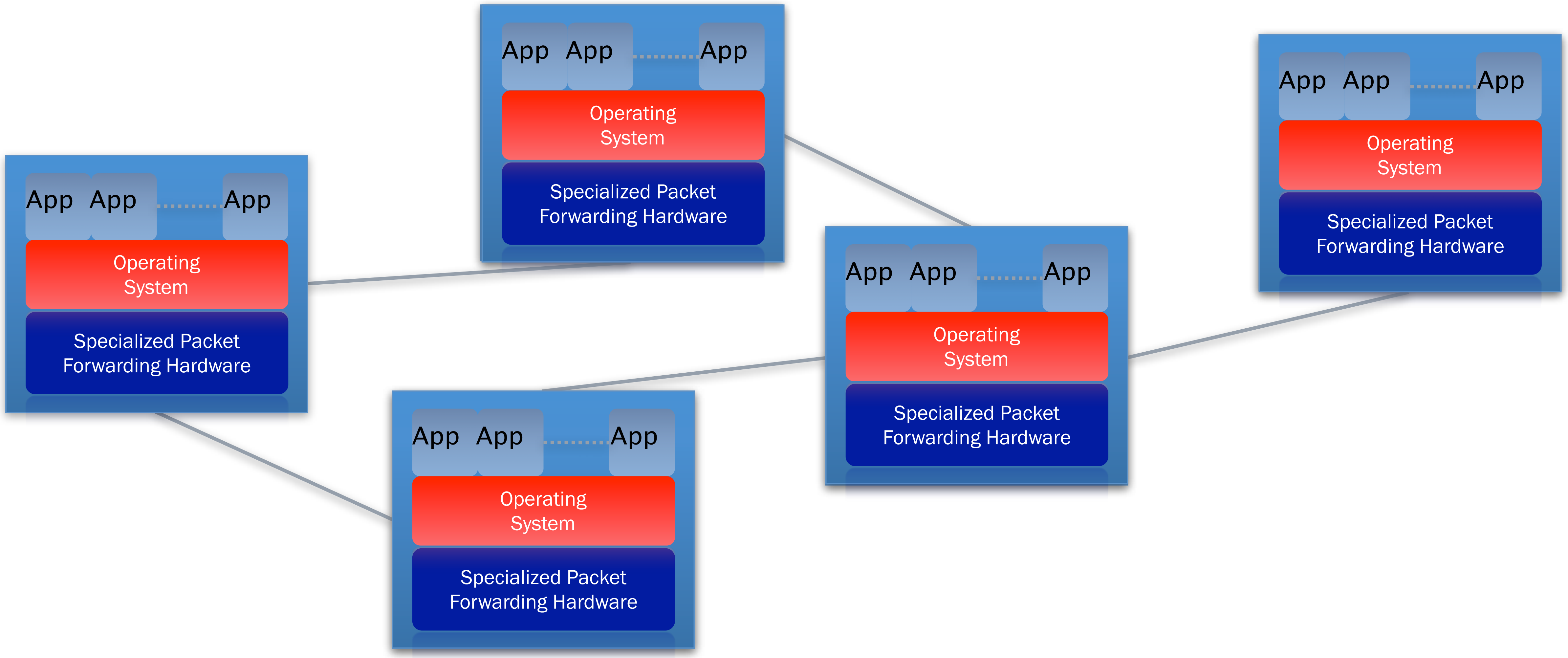
$$\square(ru_1 = healthy \wedge ru_2 = healthy) \rightarrow \square(bDC_1 = powered \wedge bDC_2 = powered)$$



Software-defined networking (SDN)

Traditional networks

- Limited intelligence, implemented as routing protocols
- Integrated control plane and data plane
- Hard to manage or change



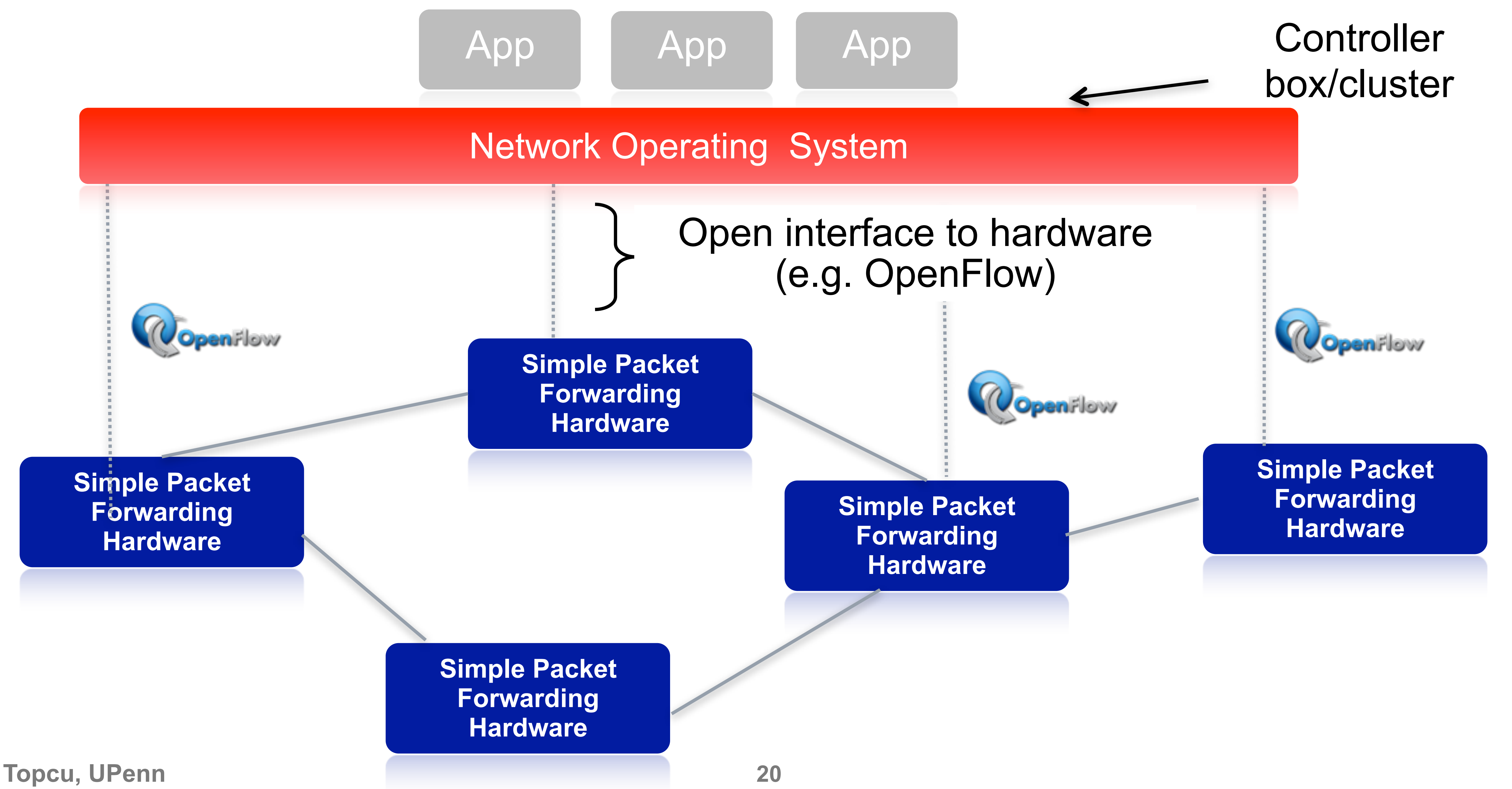
Software-defined networking (SDN)

Traditional networks

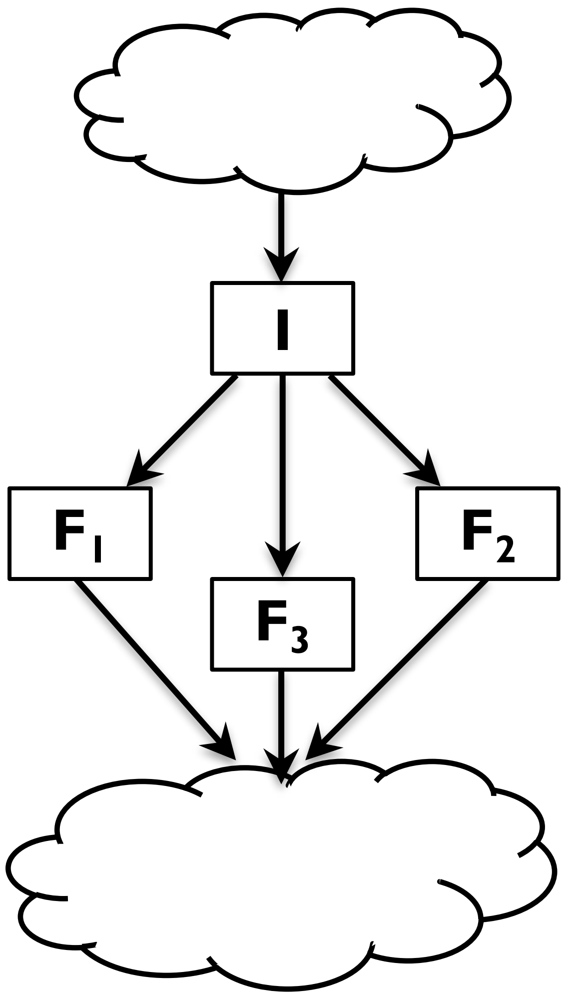
- Limited intelligence, implemented as routing protocols
- Integrated control plane and data plane
- Hard to manage or change

Software-defined networks

- Control plane is separated from data plane
- Centralized (to certain extent)
- Hard to scale & reason about



A synthesis problem in SDN



Configuration migration: Update the forwarding rules

	Type	Action
I	U, G S F	Forward F1 Forward F2 Forward F3
F1	SSH *	Deny Allow
F2	*	Allow
F3	*	Allow

→

	Type	Action
I	U G S, F	Forward F1 Forward F2 Forward F3
F1	SSH *	Deny Allow
F2	SSH *	Deny Allow
F3	*	Allow

Constraints: Enforce a security policy that denies SSH traffic from untrustworthy hosts, but allows all other traffic to pass through the network

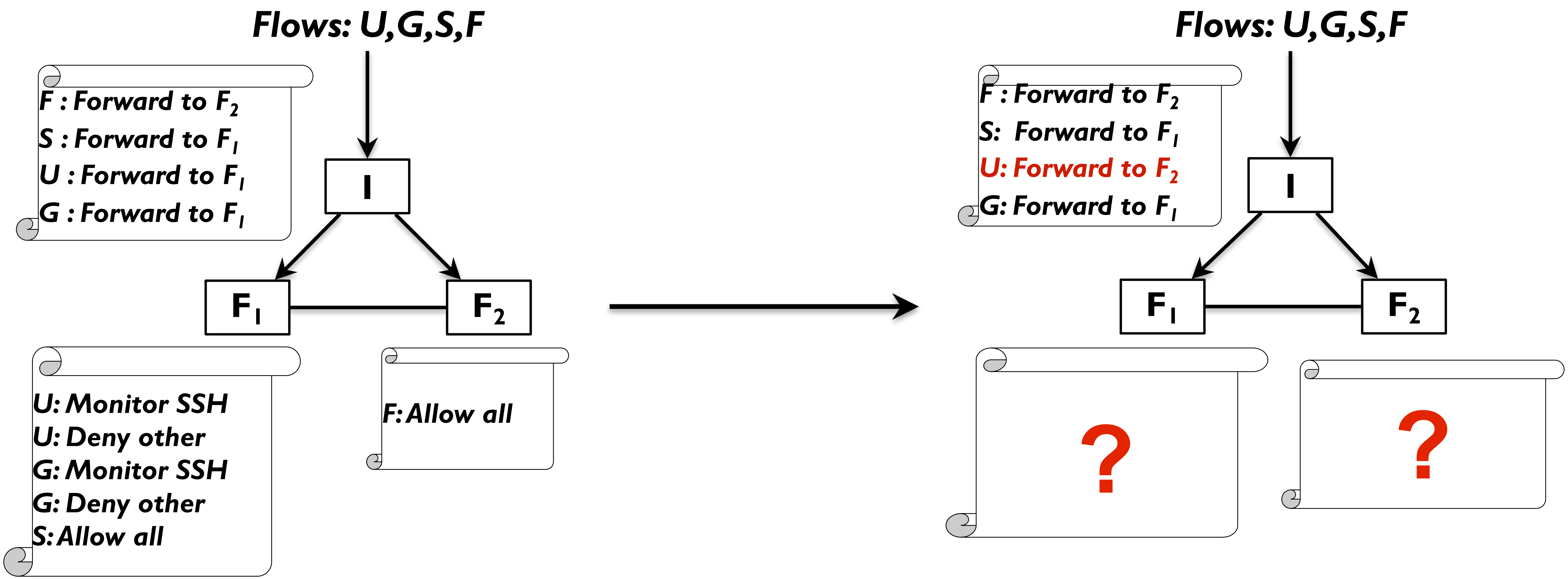
Synthesize ordering of rule updates

Simply a closed-system synthesis question -- use nuSMV to obtain:

- Update I to forward S traffic to F3
- Update F2 to deny SSH packets
- Update I to forward G traffic to F2

Access control reconfiguration

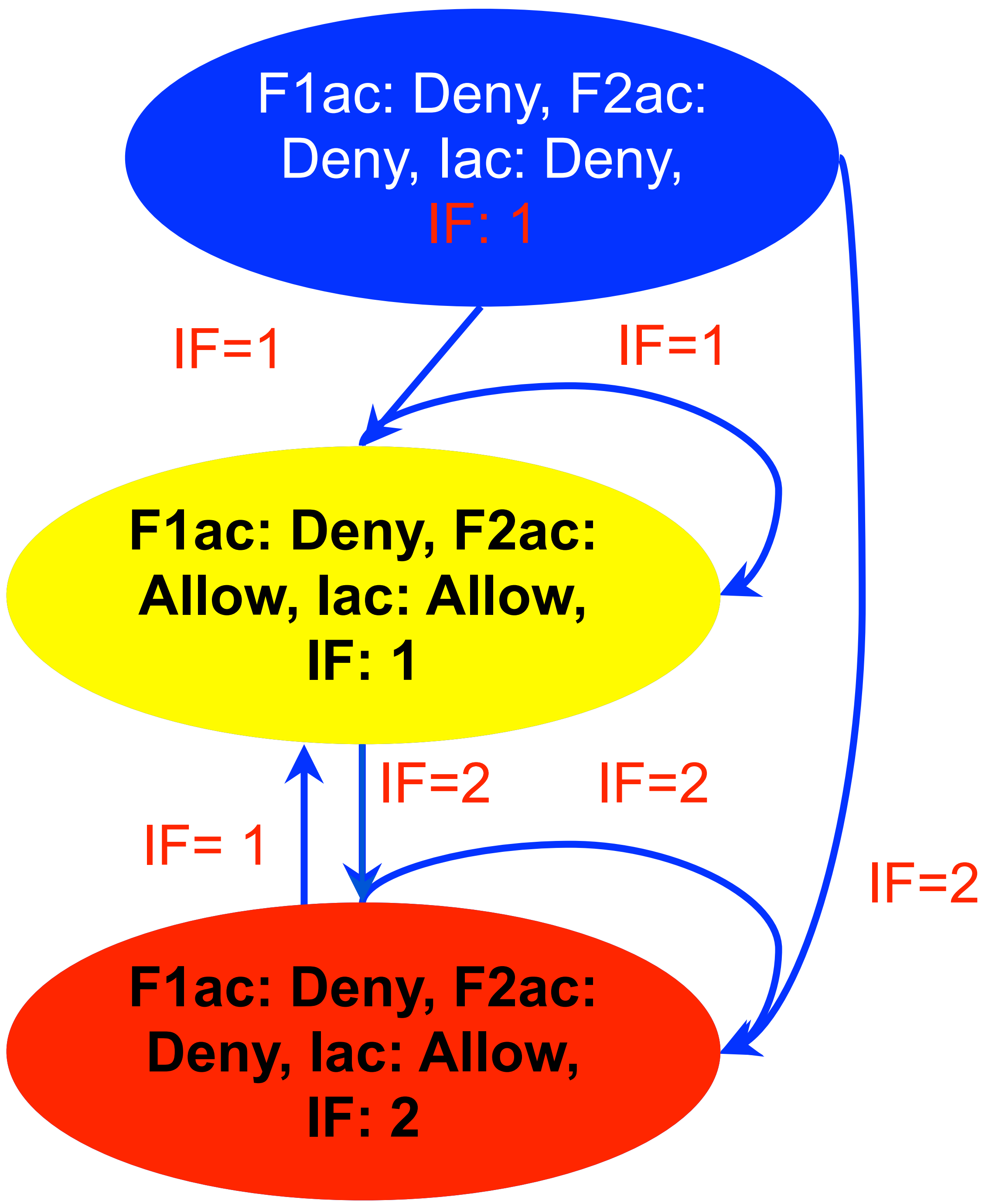
Same as configuration migration with updates chosen by users, e.g., to balance loads
Hence, need to treat the updates as adversarial inputs from environment



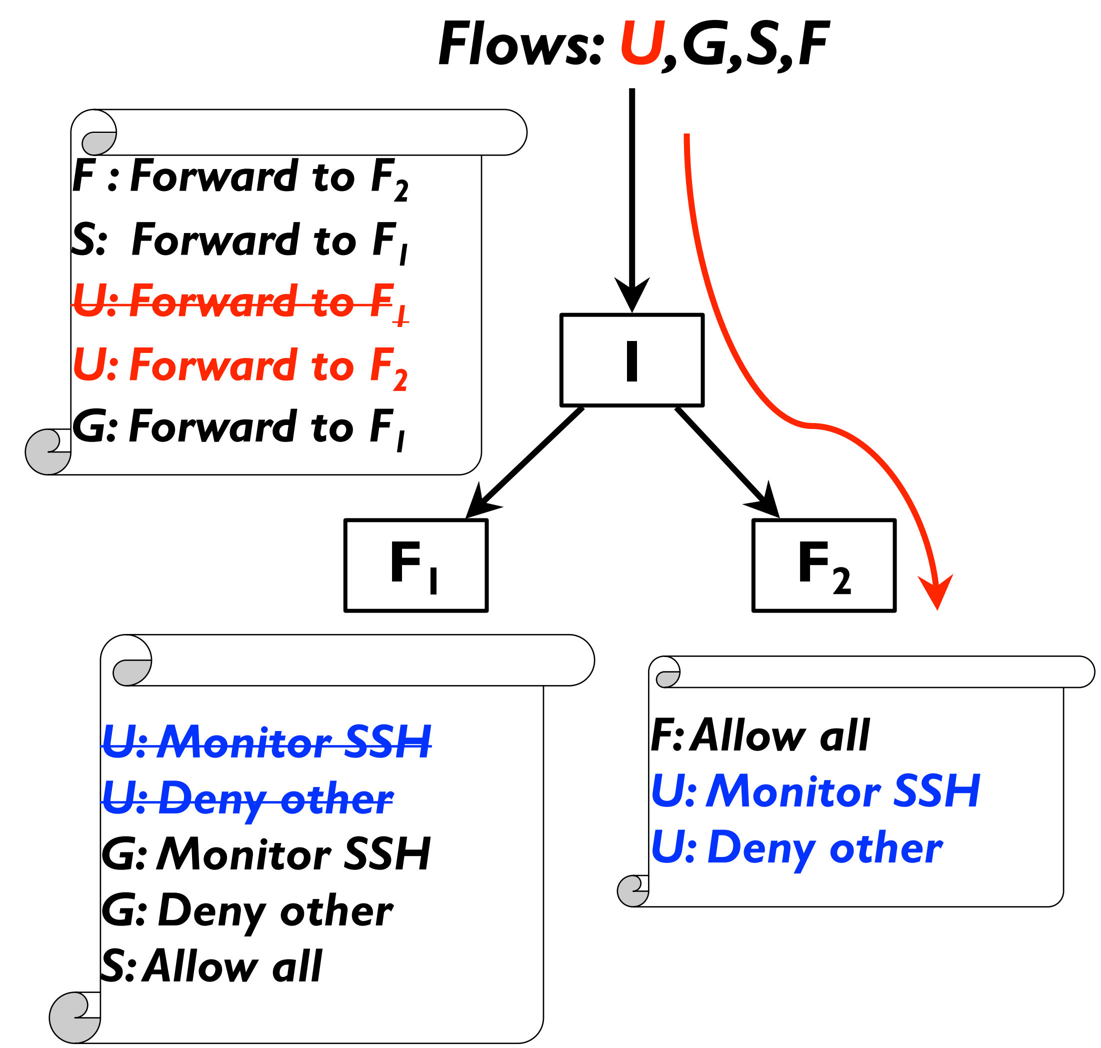
- Switches: I(ingress), $F_{1,2}$ (switches for two servers)
- Flows: U(untrusted), G(guest), S(student), F(faculty)
- High-level policy for U flows: monitor SSH packets, deny all other

A toy example for access control reconfiguration

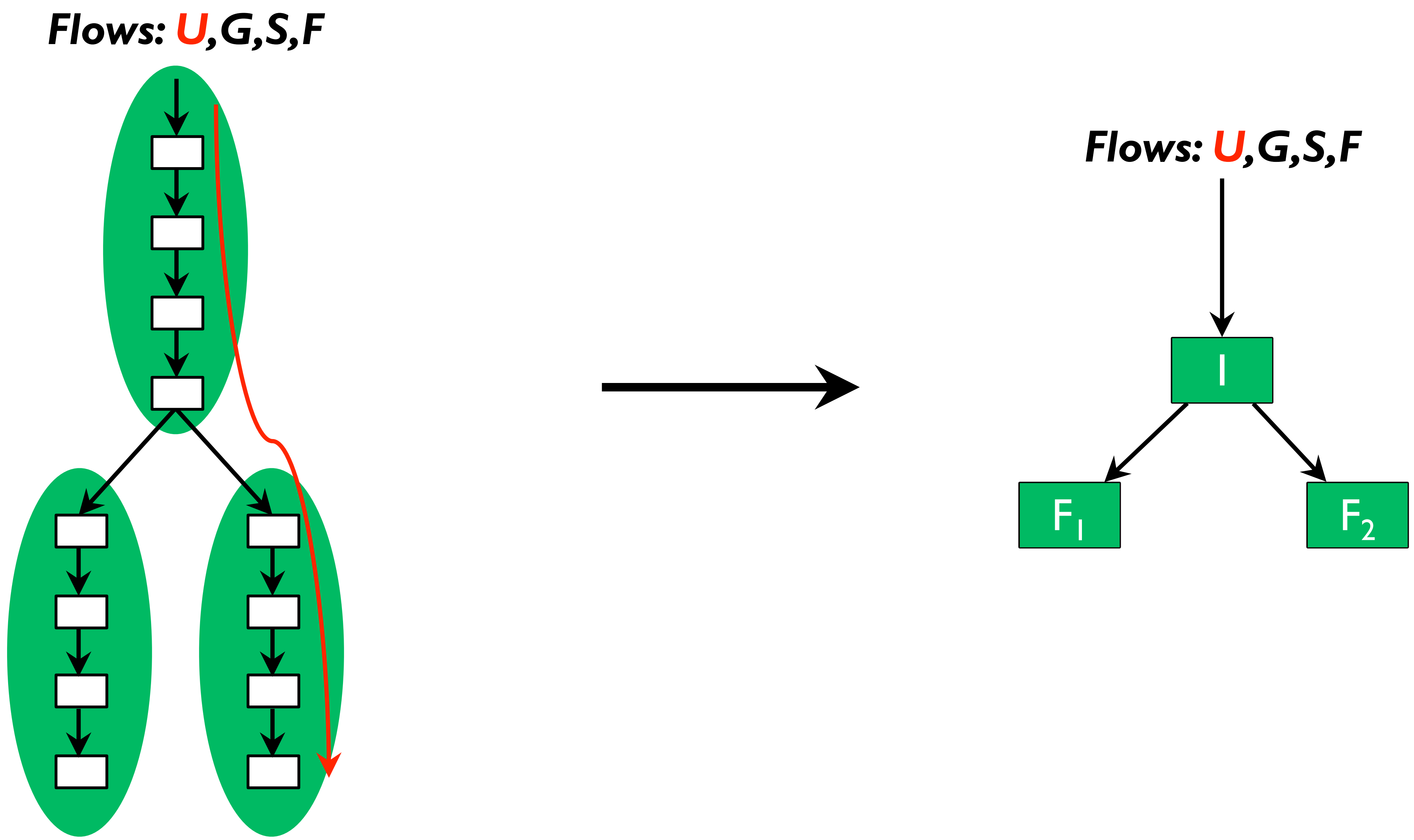
Part of the resulting automaton



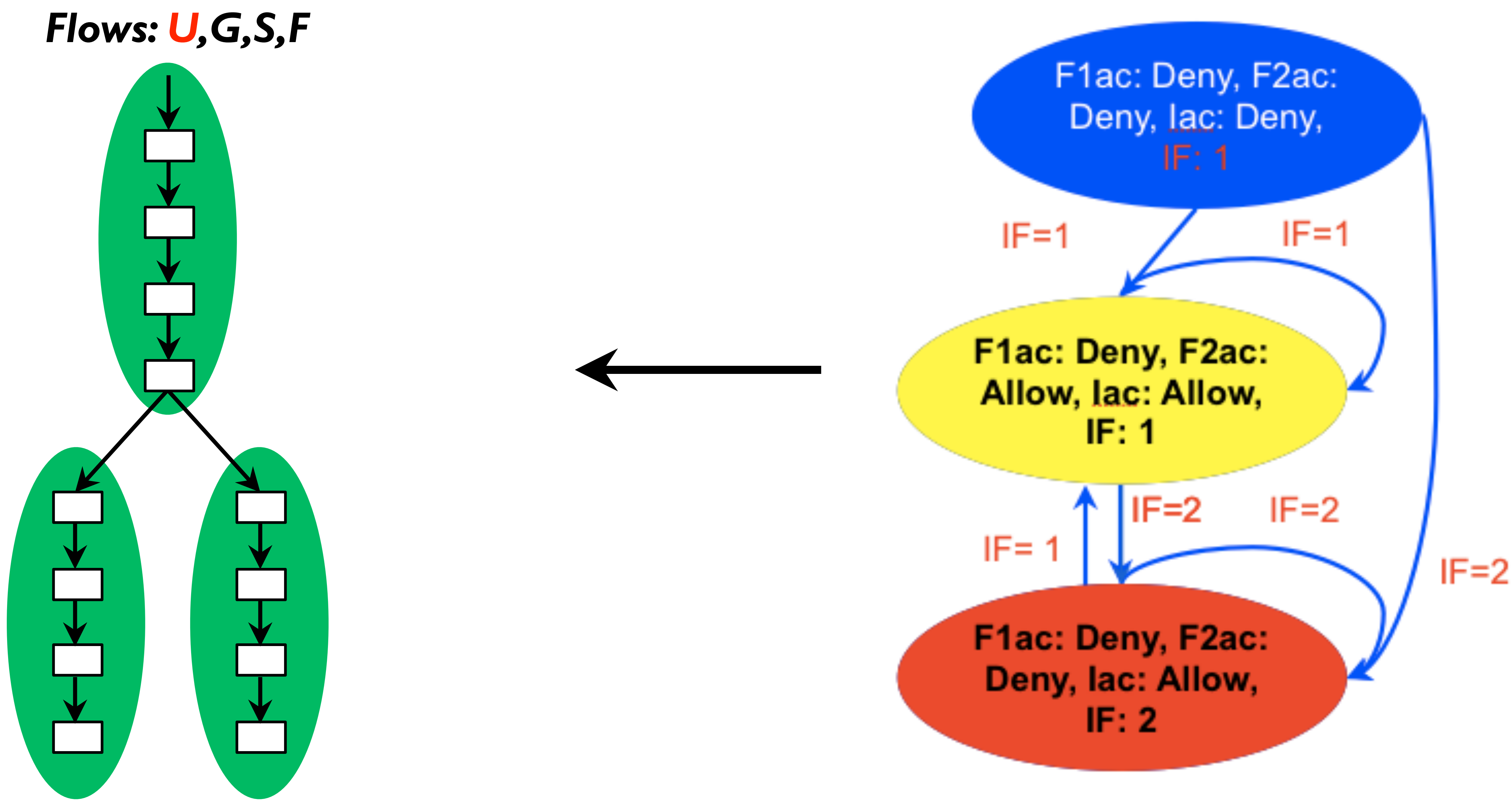
Updates in access control



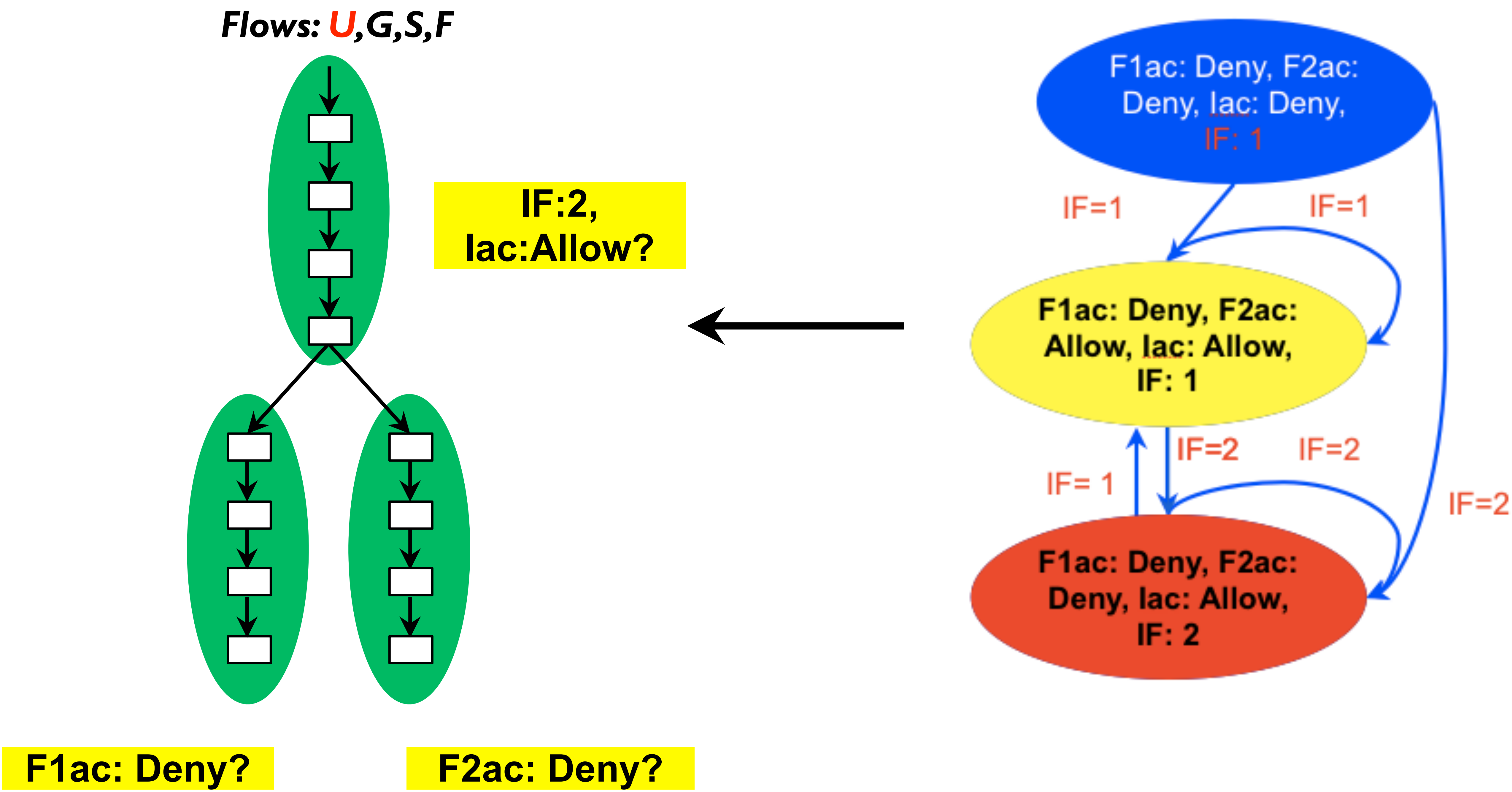
Access control reconfiguration using a topology-based abstraction



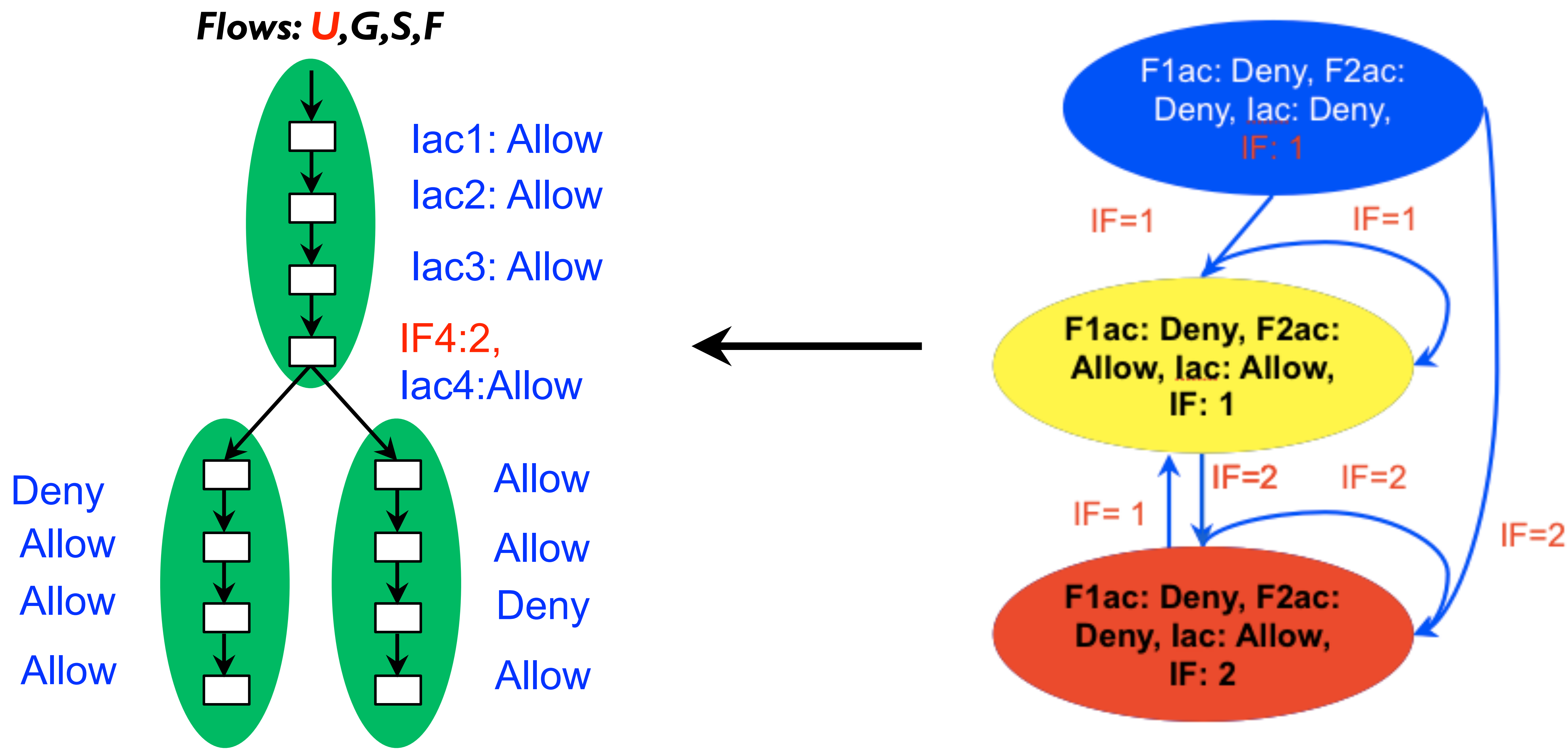
Access control reconfiguration using a topology-based abstraction



Access control reconfiguration using a topology-based abstraction



Access control reconfiguration using a topology-based abstraction



Interactions with software-defined networking

Lessons from networking on making progress in scalability?

Relatively well-established notions of network abstractions

- based on topology
- based on flow properties/predicates, e.g., bandwidth

Property-preserving decompositions

- virtual networks
- slide isolation