Automated Black-box Verification of Networking Systems

UCL ENGINEERING
Change the world

UCL

Collaborators

Nate Foster

Matteo Sammartino

Stefan Zetzsche

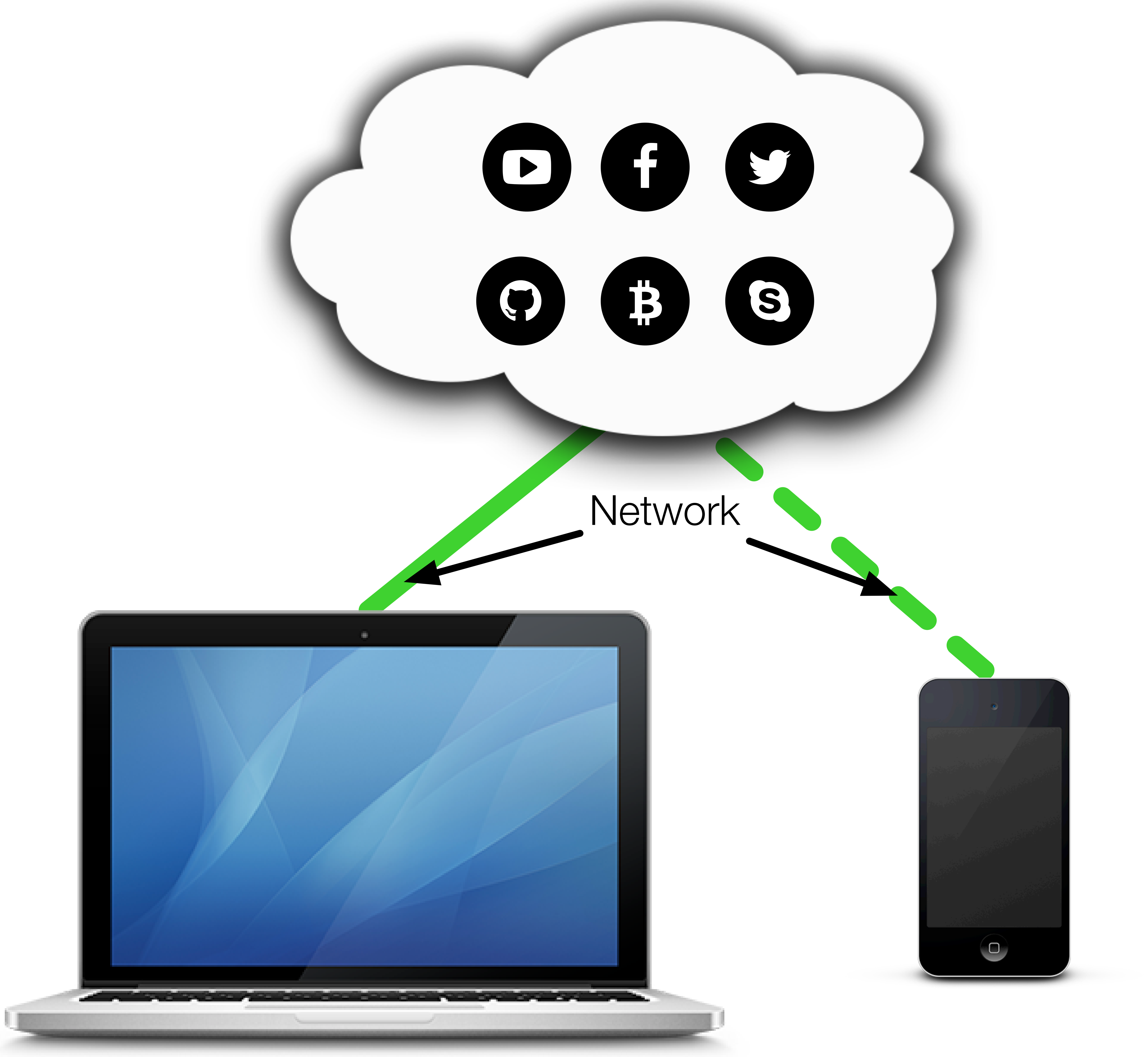
Dexter Kozen

Steffen Smolka

Many of today's high-level languages were designed in an era when computers looked like this...


But nowadays, computers look like this...

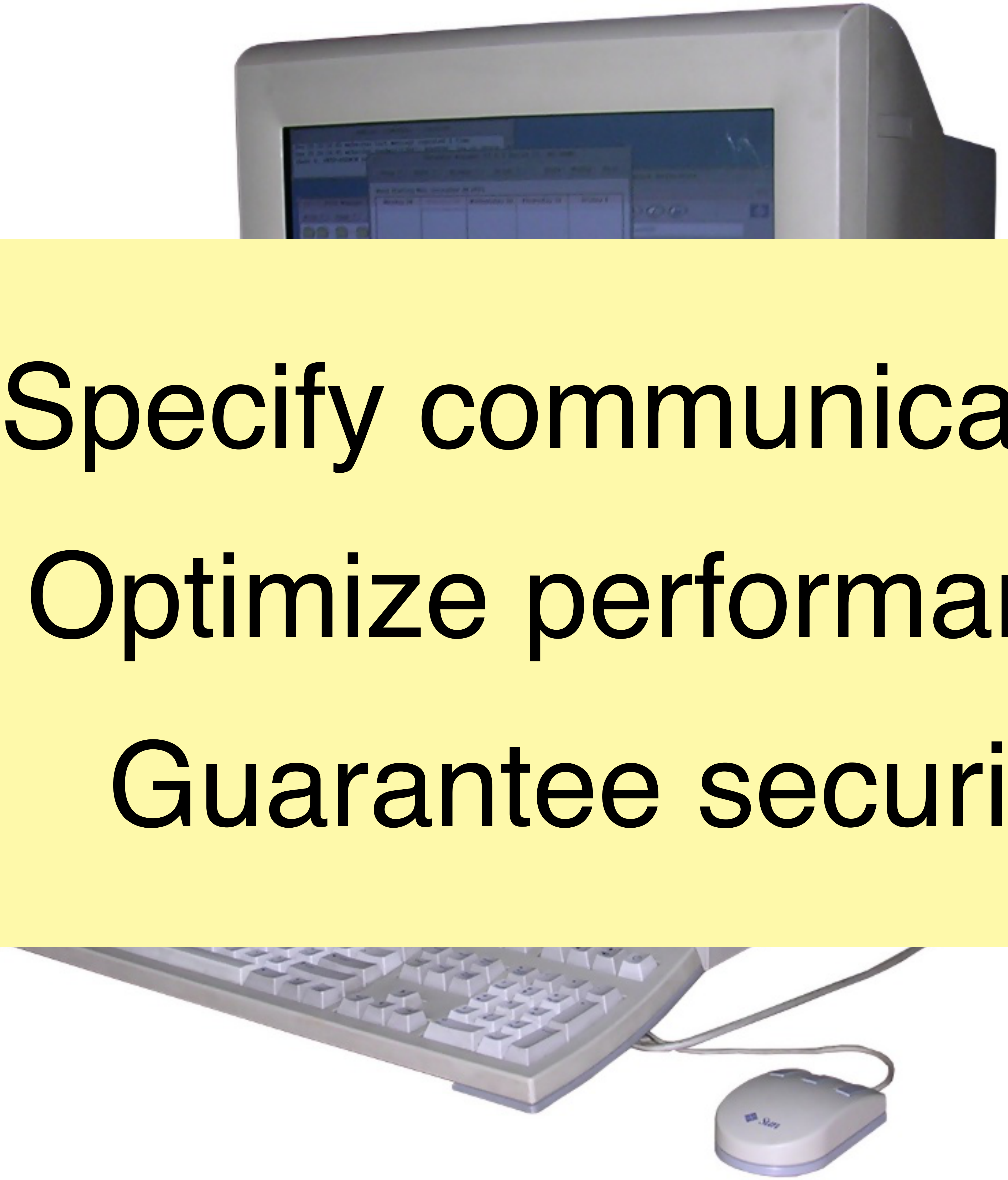
And applications are structured like this...



We need new kinds of abstractions and tools for programming these networked systems!

- Centralized
- Sequential
- Functional

- Distributed
- Concurrent
- Interactive



Specify communication

Optimize performance

Guarantee security

Software-Defined Networking

Networking

"The last bastion of mainframe computing" [Hamilton 2009]

- Modern computers
  - implemented with commodity hardware
  - programmed using general-purpose languages, standard interfaces
- Networks
  - built and programmed the same way since the 1970s
  - low-level, special-purpose devices implemented on custom hardware
  - routers and switches that do little besides maintaining routing tables and forwarding packets
  - configured locally using proprietary interfaces
  - network configuration ("tuning") largely a black art

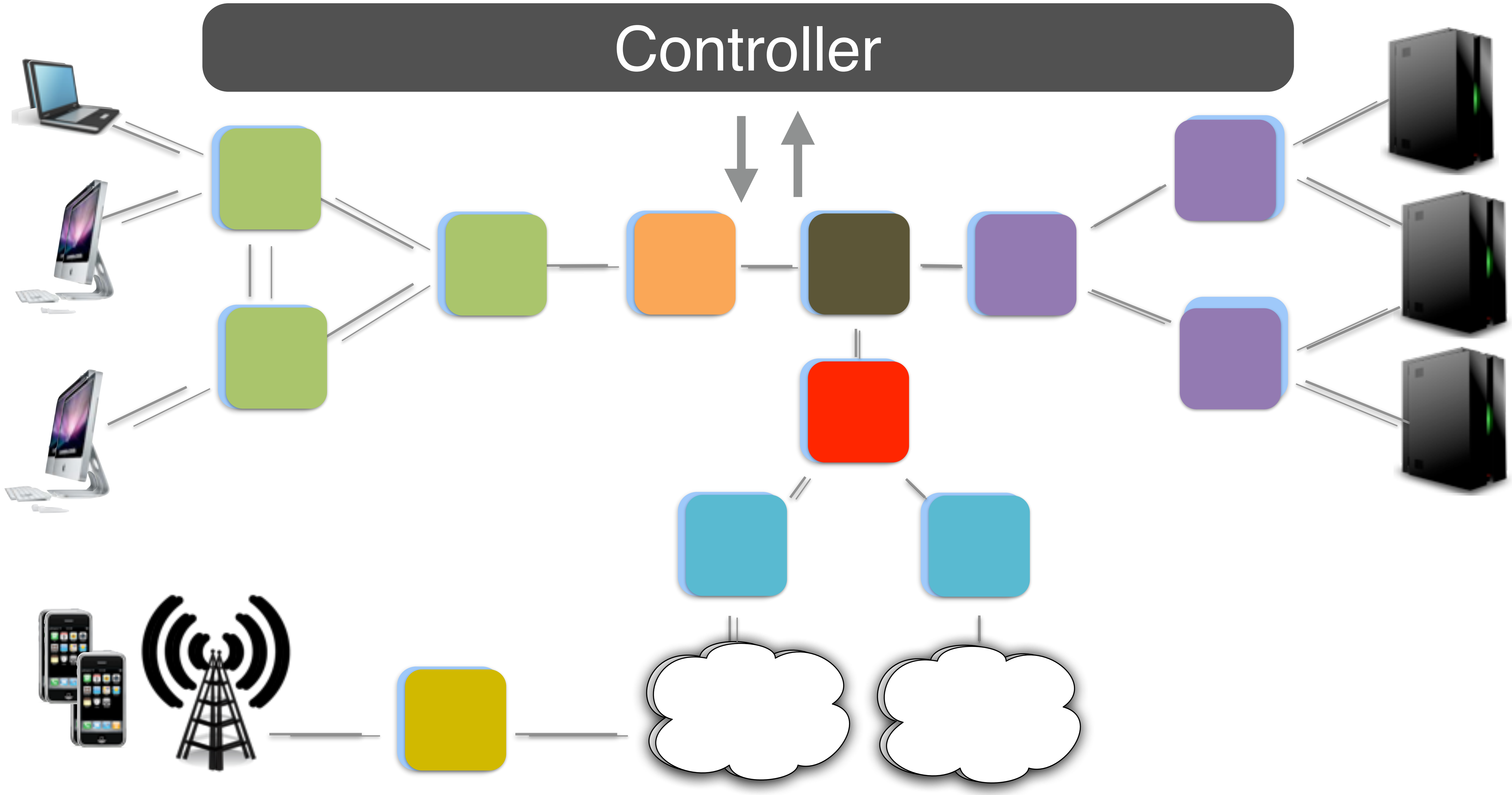
Networking

- Difficult to implement end-to-end routing policies and optimizations that require a global perspective
- Difficult to extend with new functionality
- Effectively impossible to reason precisely about behavior

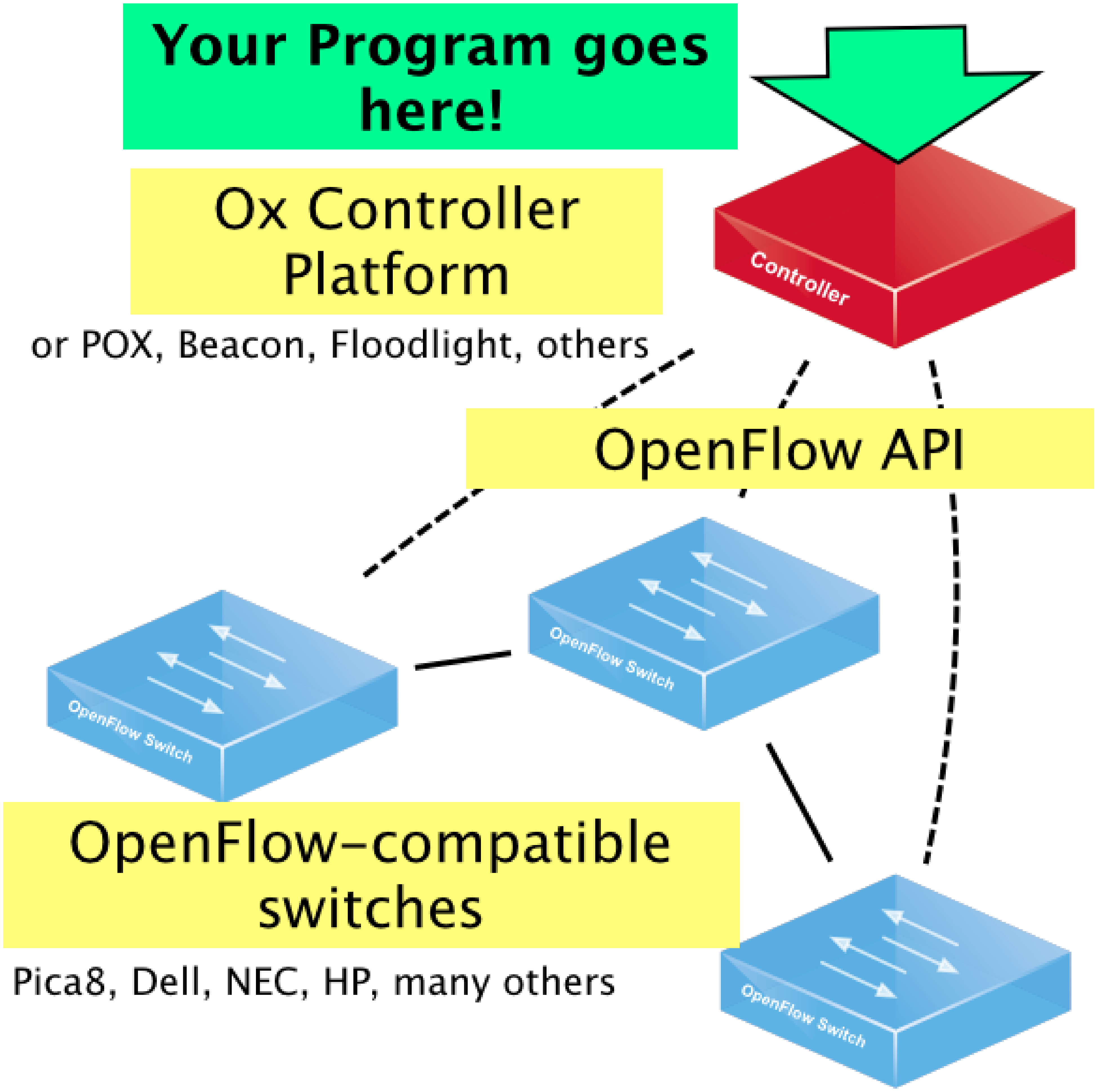
Software-Defined Networking

A clean-slate architecture based on two key ideas:

- Generalize network devices
- Separate control and forwarding



Software-Defined Networks



OpenFlow

A first step: the OpenFlow API [McKeown & al., SIGCOMM 08]

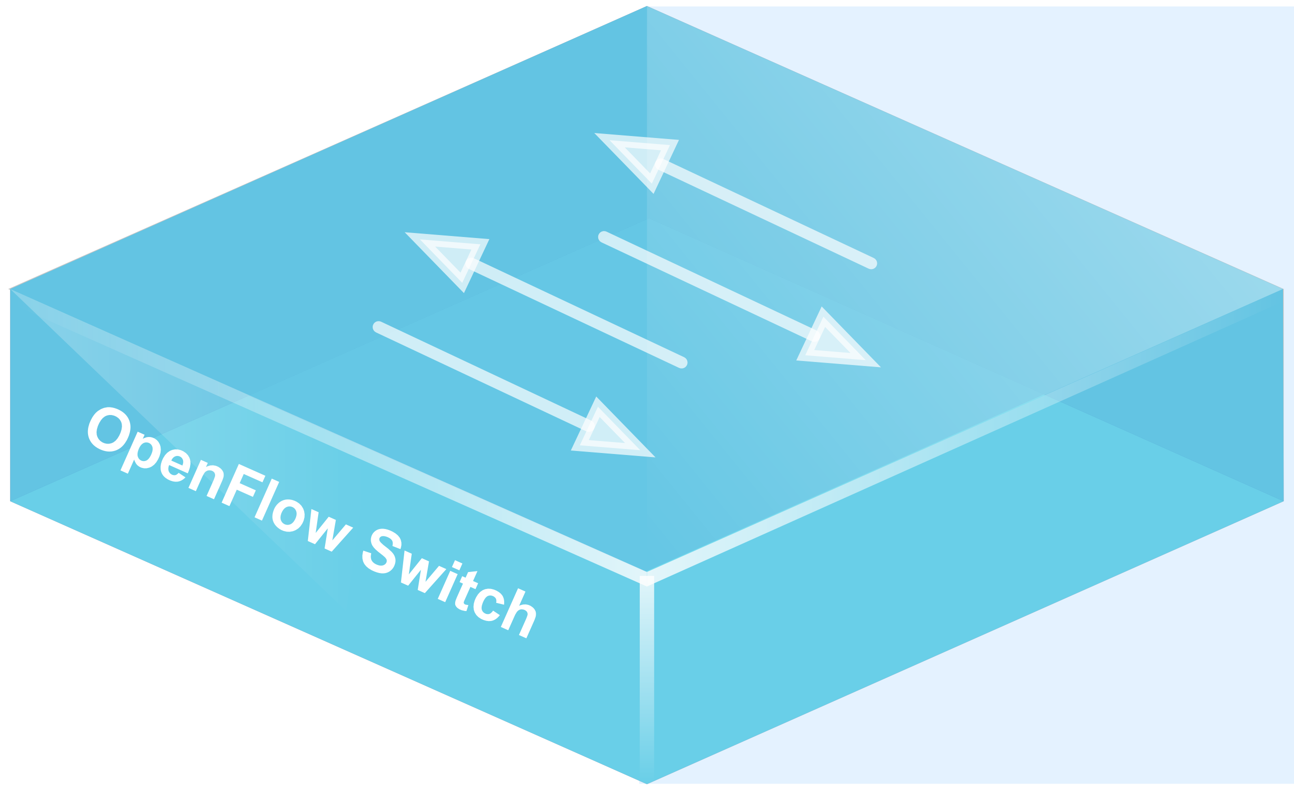
- specifies capabilities and behavior of switch hardware
- a language for manipulating network configurations
- very low-level: easy for hardware to implement, difficult for humans to write and reason about

But...

- is platform independent
- provides an open standard that any vendor can implement

OpenFlow Switch

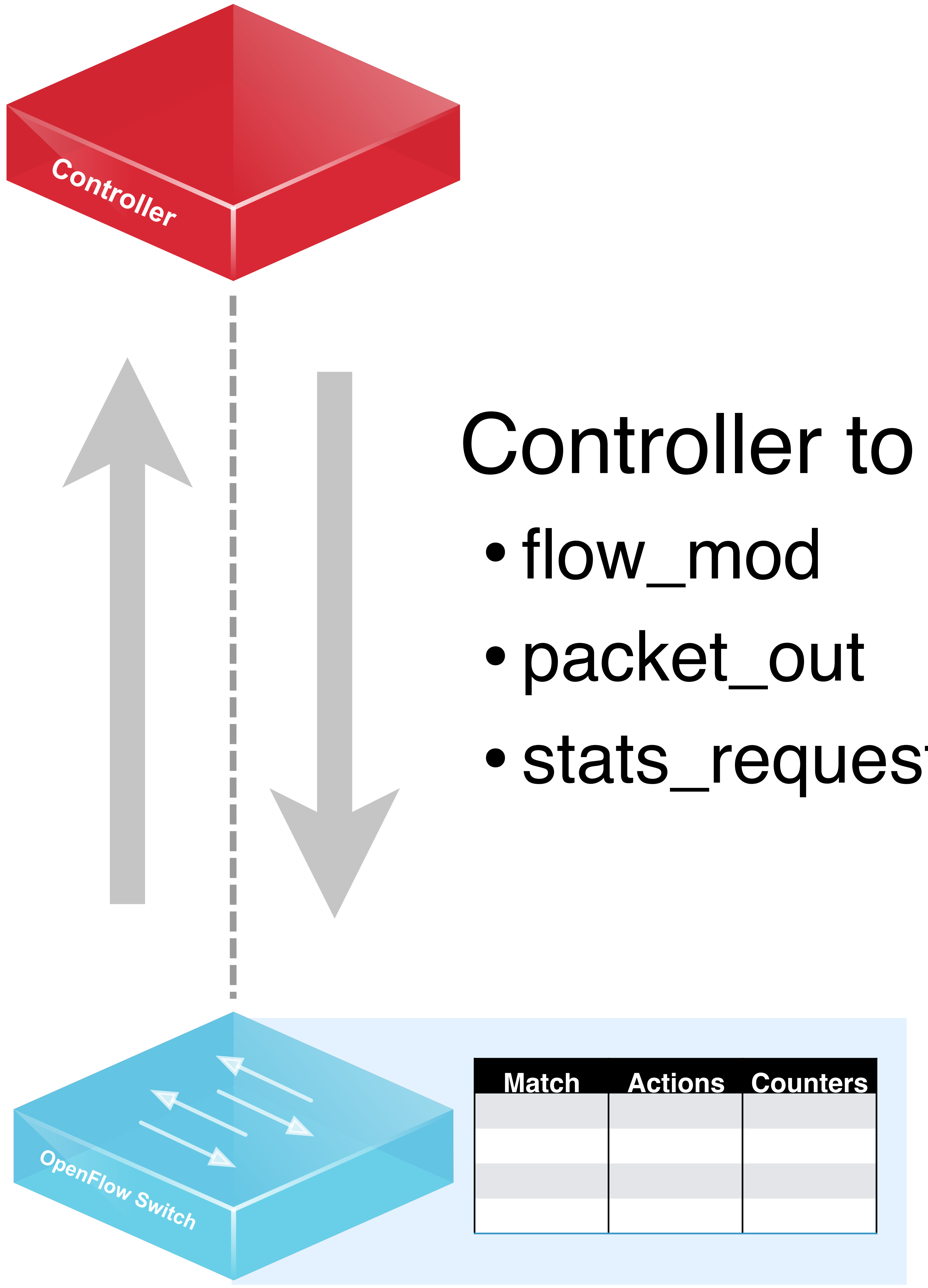
General-purpose packet-processing device that can be used to implement switches, routers, firewalls, etc.



Match	Actions	Counters
10.0.0.1	Drop	(73,2458)
10.0.0.2	Forward 2	(16,846)
10.0.0.3	Forward 3	(23,5729)
*	Controller	(5,472)

Key data structure is a *flow table* containing a prioritized list of match-action *rules* and *counters*

OpenFlow Controller



Switch to controller:

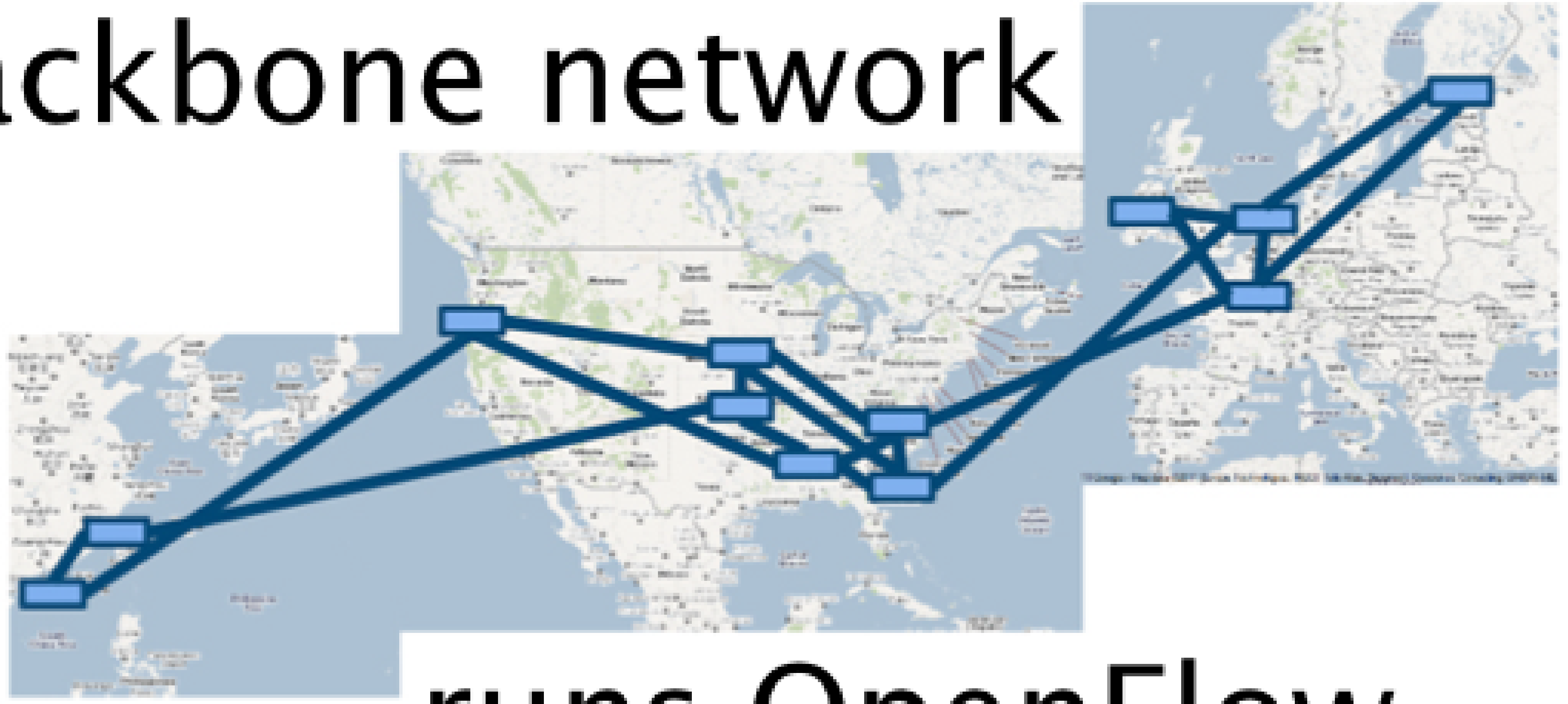
- switch_connected
- switch_disconnected
- port_status
- packet_in
- stats_reply

Controller to switch:

- flow_mod
- packet_out
- stats_request

A Major Trend in Industry

Backbone network



runs OpenFlow

Bought by VMware for $1.2B

Verification of networks

Trend in PL&Verification after Software-Defined Networks

- Design *high-level languages* that model essential network features
- Develop *semantics* that enables reasoning precisely about behaviour
- Build *tools* to synthesise low-level implementations automatically

- Frenetic [Foster & al., ICFP 11]
- Pyretic [Monsanto & al., NSDI 13]
- Maple [Voellmy & al., SIGCOMM 13]
- FlowLog [Nelson & al., NSDI 14]
- Header Space Analysis [Kazemian & al., NSDI 12]
- VeriFlow [Khurshid & al., NSDI 13]
- NetKAT [Anderson & al., POPL 14]
- and many others . . .

But…

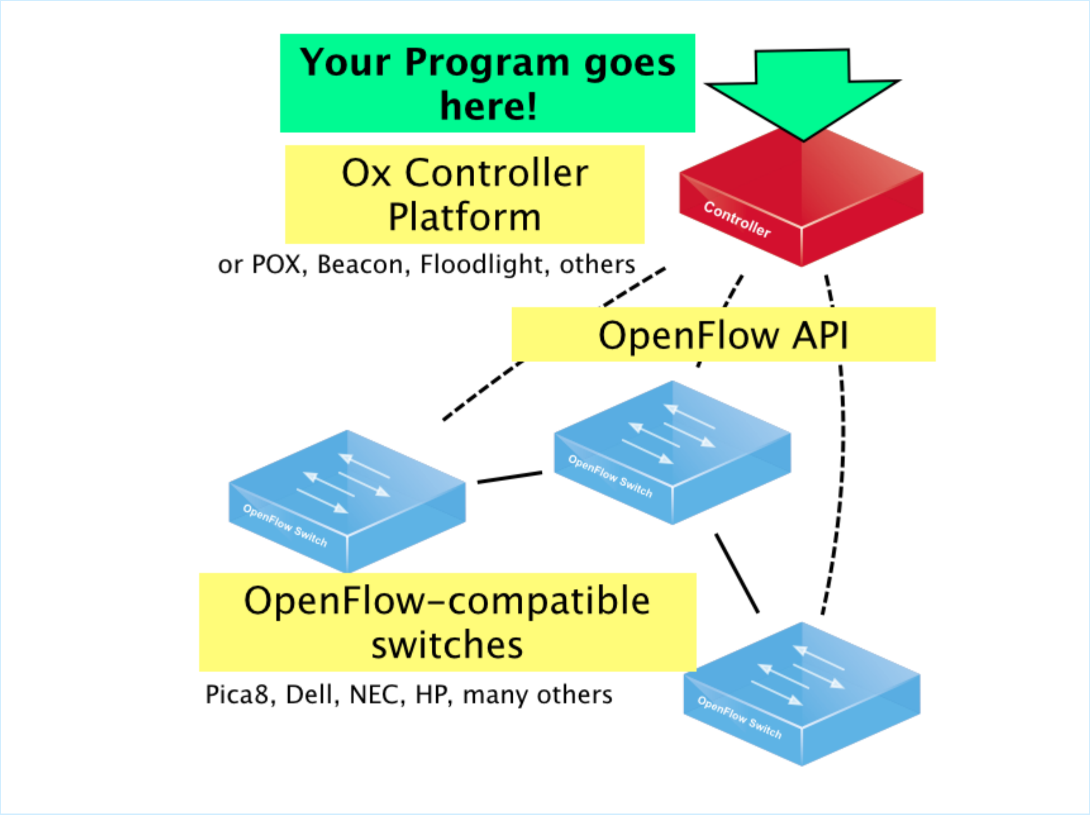
What if there is no formal model?

Does the low-level implementation really do what it is supposed to do?

What we propose

Automaton

Build black-box model via interactions with the system



Automated Modelling

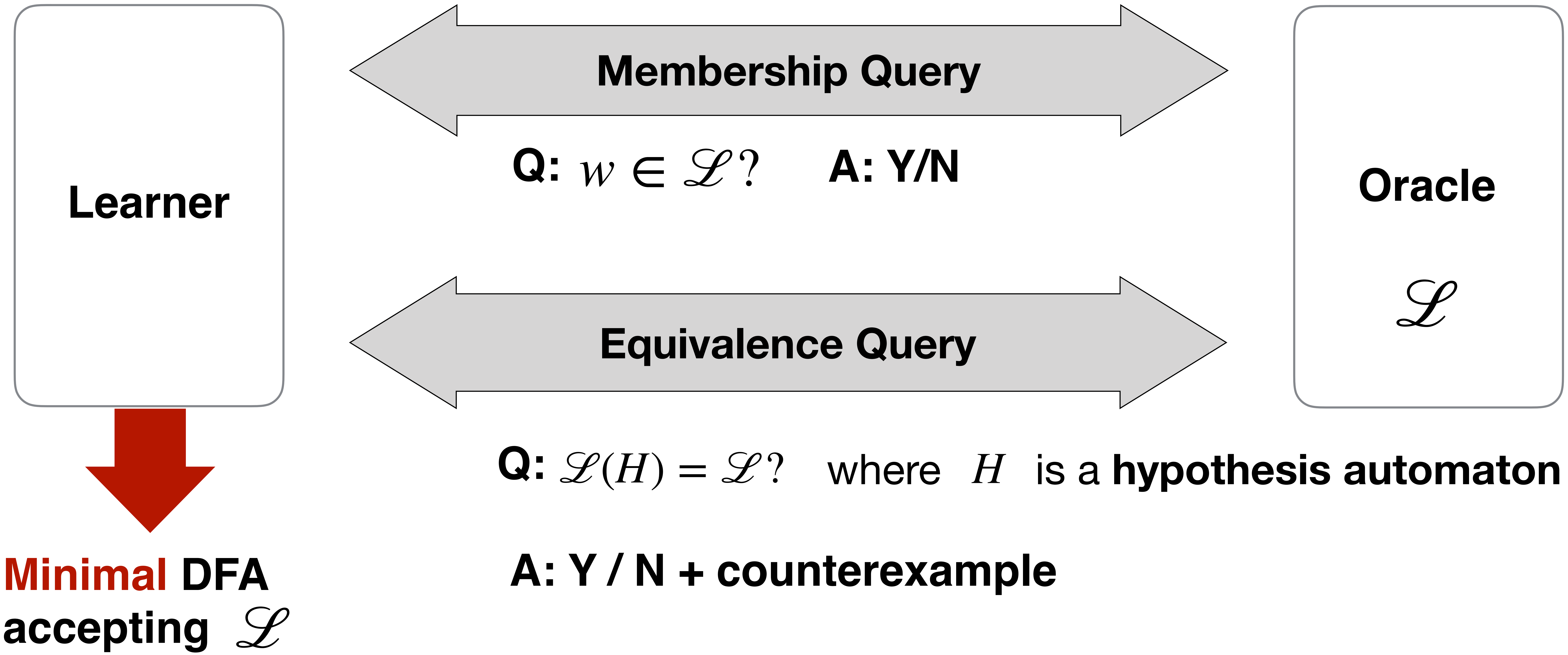
Automated Verification

Properties

Automata learning (Angluin '87)

Finite alphabet of system's actions A

Set of system behaviours is a **regular language** $\mathscr{L} \subseteq A^{\star}$



In practice…

"Lazy" testing and model-checking

Good for scalability!

Many interesting applications

- Detect TLS implementations flaws [USENIX Sec. Sym. ’15]
- TCP implementations [CAV ’16]
- Analysis of botnet protocols [CCS ’10]
- Bank cards …

To each application domain its model…

Mealy Machines

Probabilistic

Alternating

Weighted

Register

Non-deterministic

Universal

Buchi

Do I need to write my automata learning algorithm from scratch?

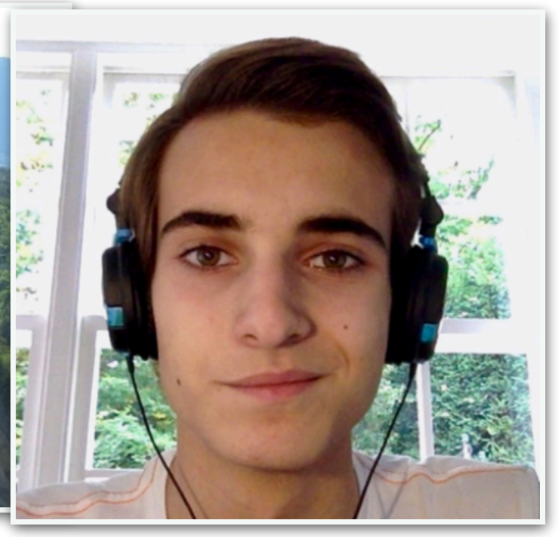
NO! Category Theory can help!

Categorical **A**utomata **L**earning **F**ramework

calf-project.org

Gerco van Heerdt
UCL



Joshua Moerman
Radboud University

Bartek Klin
Warsaw University

Michal Szynwelski
Warsaw University

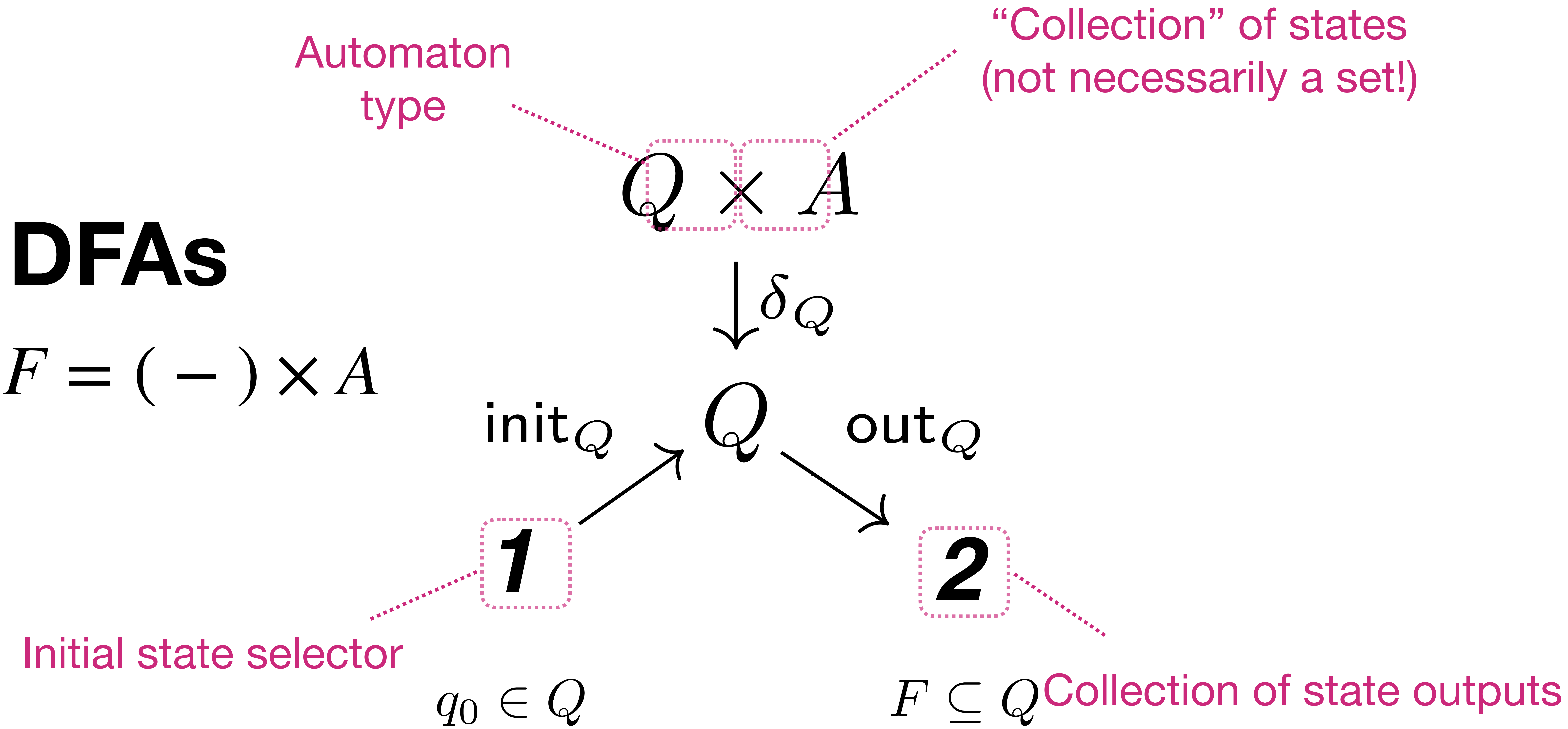
Maverick Chardet
ENS Lyon

Tiago Ferreira
UCL Intern

Different automata, same structure

DFAs

$$F = (\,-\,) \times A$$



A general framework

Abstract observation data structure

approximates

Target minimal automaton

$$FQ$$
$$\downarrow \delta_Q$$
$$I \xrightarrow{\mathsf{init}_Q} Q \xrightarrow{\mathsf{out}_Q} Y$$

abstract closedness and consistency

Hypothesis automaton

$$FH$$
$$\downarrow \delta_H$$
$$I \xrightarrow{\mathsf{init}_H} H \xrightarrow{\mathsf{out}_H}$$

General correctness theorem

Guidelines for implementation

ns!

CALF: Categorical Automata Learning Framework (CSL '17)

Gerco van Heerdt, Matteo Sammartino, Alexandra Silva

Other automata & optimizations

Change automaton type

Learning Nominal Automata (POPL ’17)

Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, Michal Szynwelski

Set	**DFAs**
Nom	**Nominal automata**
Vect	**Weighted automata**

Change main data structure

Observation tables

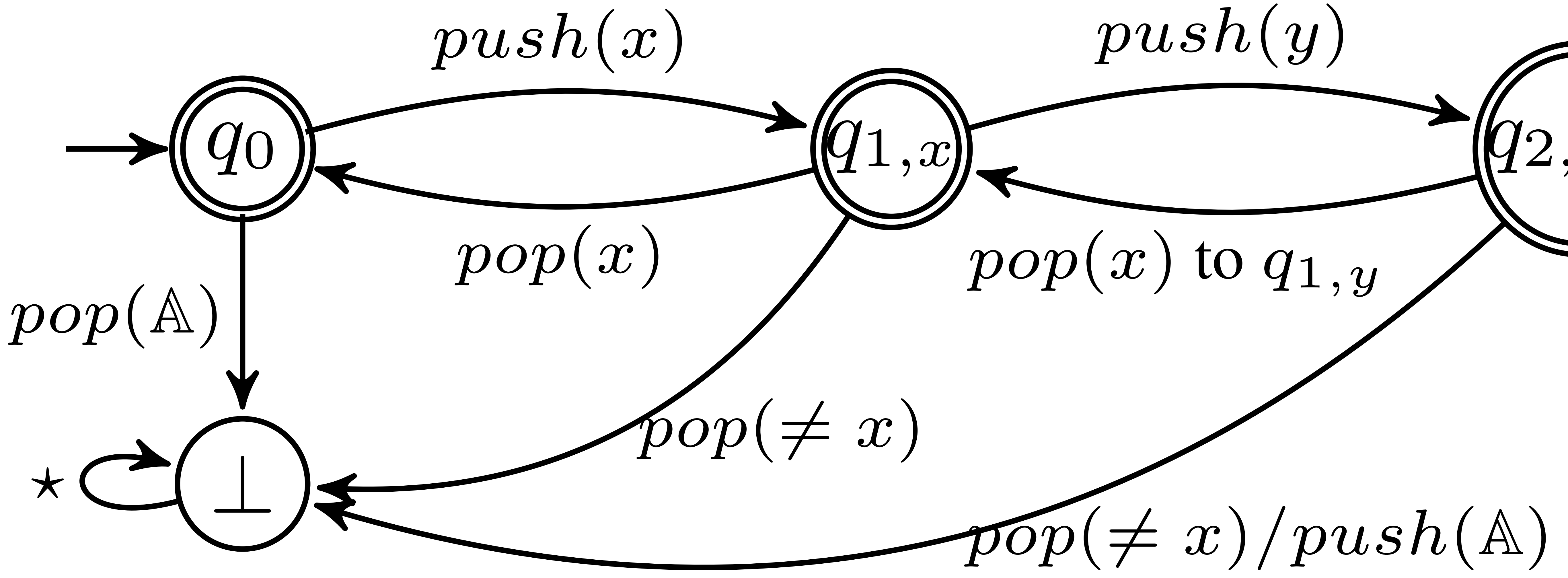
Discrimination trees

Plug monads in

Powerset	**NFAs**
Powerset with intersection	**Universal automata**
Double powerset	**Alternating automata**
Maybe monad	**Partial automata**

Infinite alphabets

infinite-state, but **finitely representable** automata



Other automata & optimizations

Change automaton type

Learning Nominal Automata (POPL ’17)

Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, Michal Szynwelski

Nom	**Nominal automata**
Vect	**Weighted automata**

Change main data structure

Observation tables

Discrimination trees

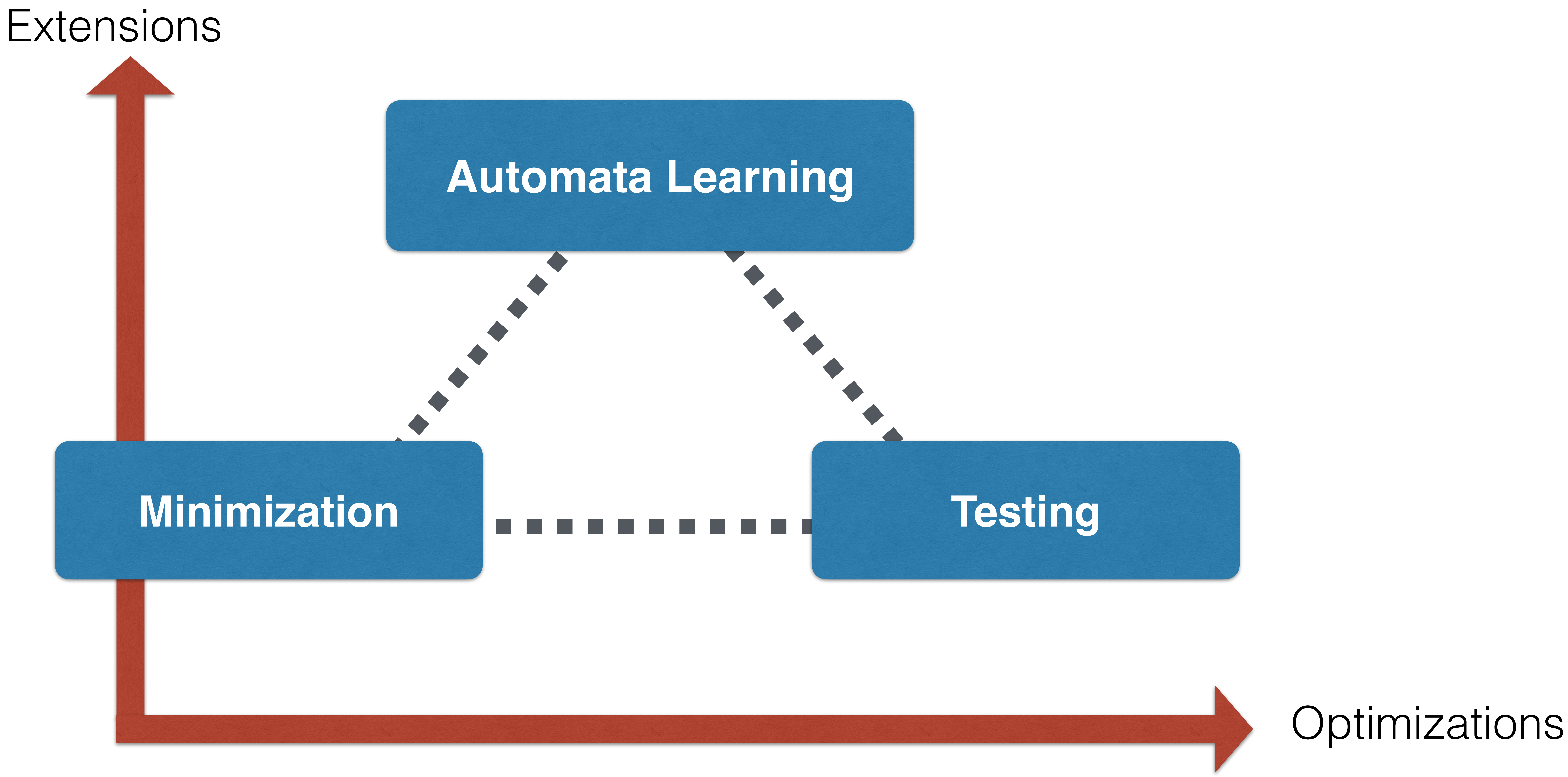
Optimising Automata Learning via Monads

Gerco van Heerdt, Matteo Sammartino, Alexandra Silva (arXiv:1704.08055)

Plug monads in

Powerset	**NFAs**
Powerset with intersection	**Universal automata**
Double powerset	**Alternating automata**
Maybe monad	**Partial automata**

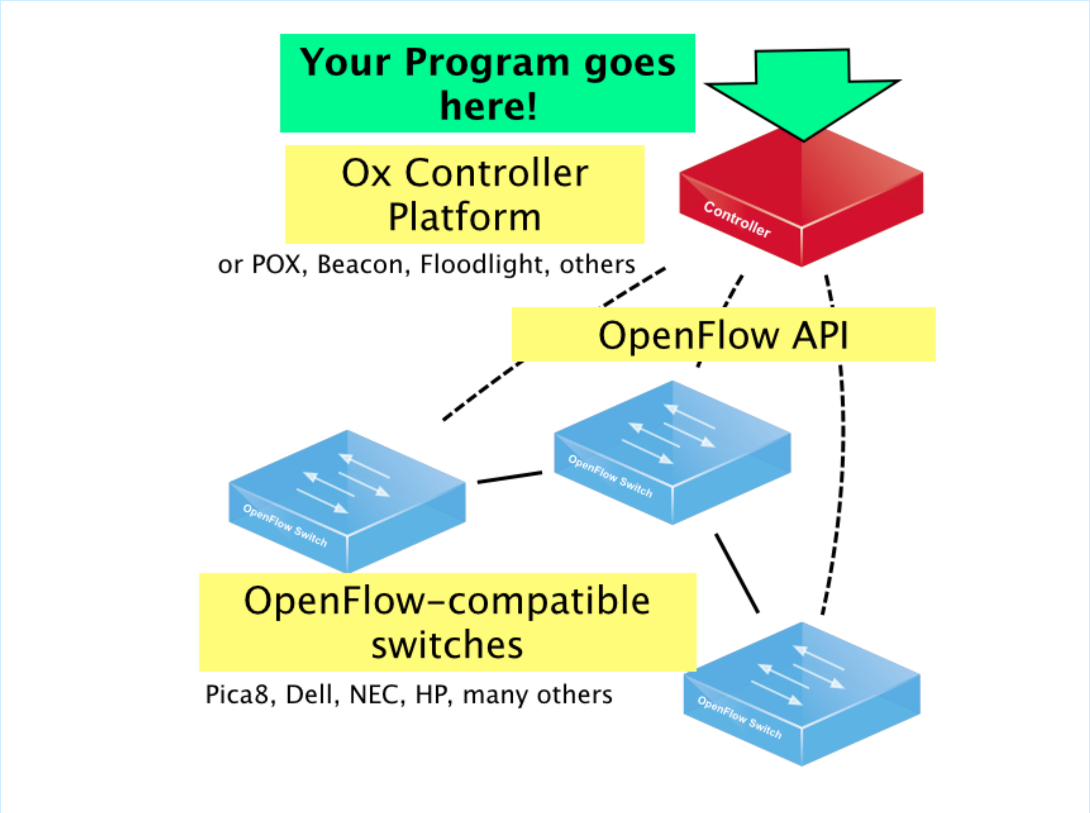
Connections with other techniques



What we propose

Automaton

Build black-box model via interactions with the system



Automated Modelling

Properties

Automated Verification

Language to describe behaviours

NetKAT

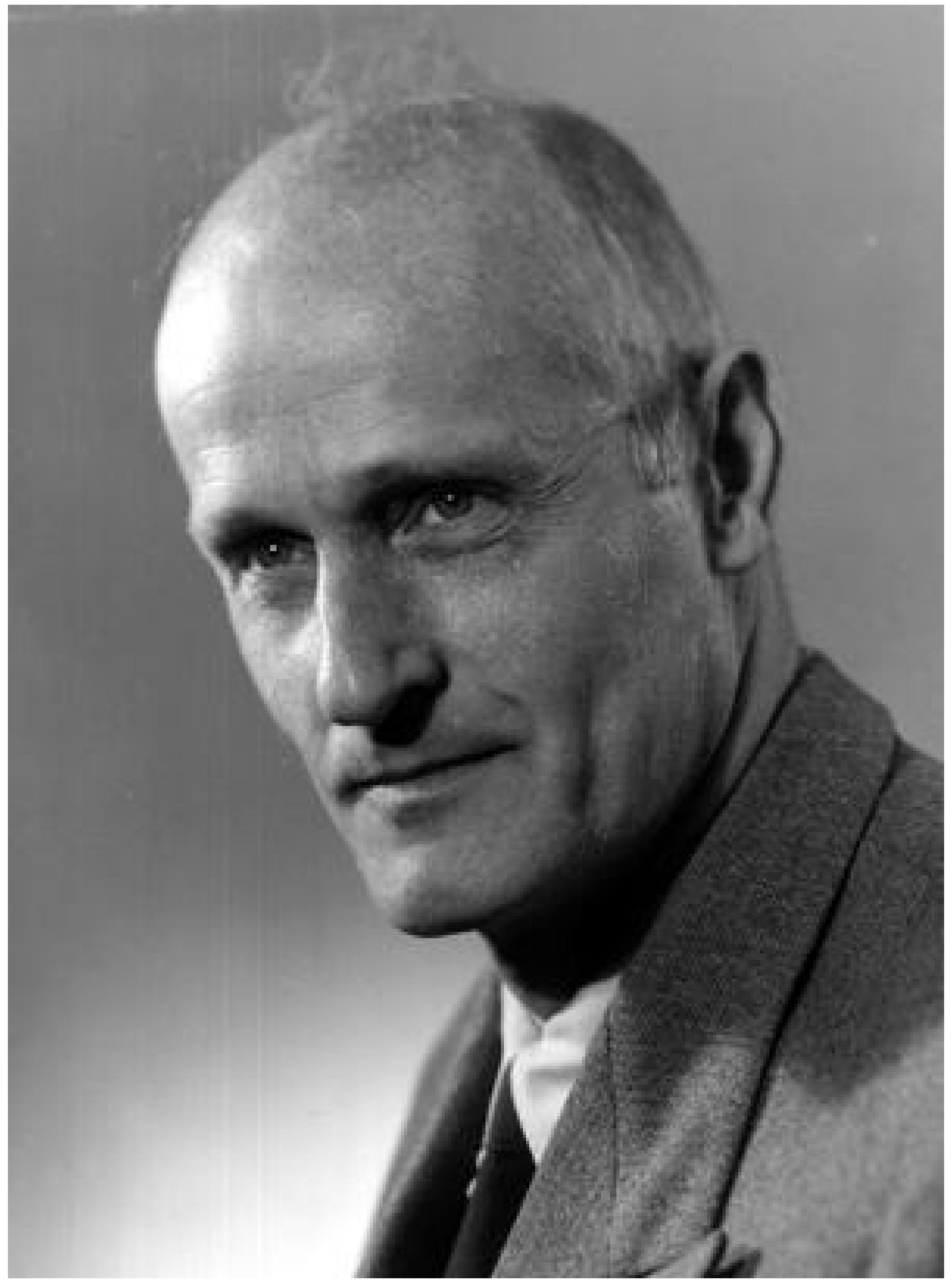
$=$

Kleene algebra with tests (KAT)

$+$

additional specialized constructs particular to network topology and packet switching

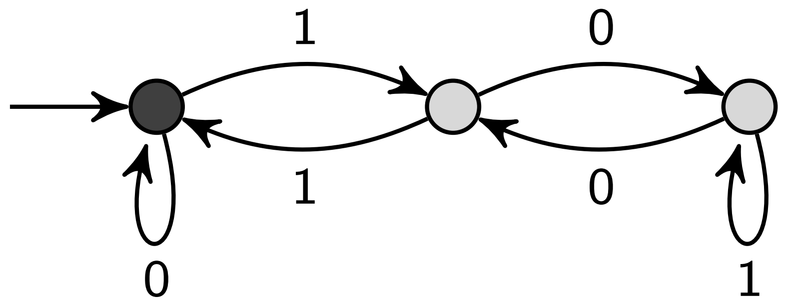
Net**KA**T



Stephen Cole Kleene
(1909–1994)

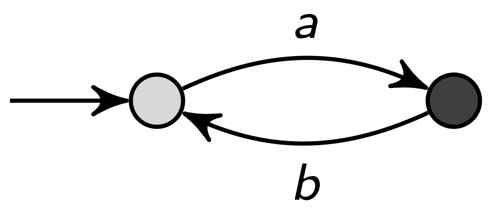
$(0 + 1(01^*0)^*1)^*$

{multiples of 3 in binary}



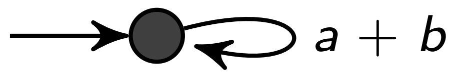
$(ab)^*a = a(ba)^*$

$\{a, aba, ababa, \ldots\}$



$(a + b)^* = a^*(ba^*)^*$

{all strings over $\{a, b\}$}



Net**KAT**

$(K, B, +, \cdot, {}^*, {}^-, 0, 1), \quad B \subseteq K$

- $(K, +, \cdot, {}^*, 0, 1)$ is a Kleene algebra
- $(B, +, \cdot, {}^-, 0, 1)$ is a Boolean algebra
- $(B, +, \cdot,$ …
- $p, q, r, \ldots$
- $a, b, c, \ldots$

KAT = simple imperative language

If b **then** p **else** q = b;p + !b;q

While b **do** p = (bp)*!b

NetKAT

Deductive Completeness and Complexity

- deductively complete over language, relational, and trace models
- subsumes propositional Hoare logic (PHL)
- deductively complete for all relationally valid Hoare-style rules

$$\frac{\{b_1\}\ p_1\ \{c_1\},\ \ldots,\ \{b_n\}\ p_n\ \{c_n\}}{\{b\}\ p\ \{c\}}$$

- decidable in PSPACE

Applications

- protocol verification
- static analysis and abstract interpretation
- verification of compiler optimizations

NetKAT

- a packet π is an assignment of constant values n to fields x
- a packet history is a nonempty sequence of packets
  $\pi_1 :: \pi_2 :: \cdots :: \pi_k$
- the head packet is π_1

NetKAT

- assignments $x \leftarrow n$
  assign constant value n to field x in the head packet
- tests $x = n$
  if value of field x in the head packet is n, then pass, else drop
- dup
  duplicate the head packet

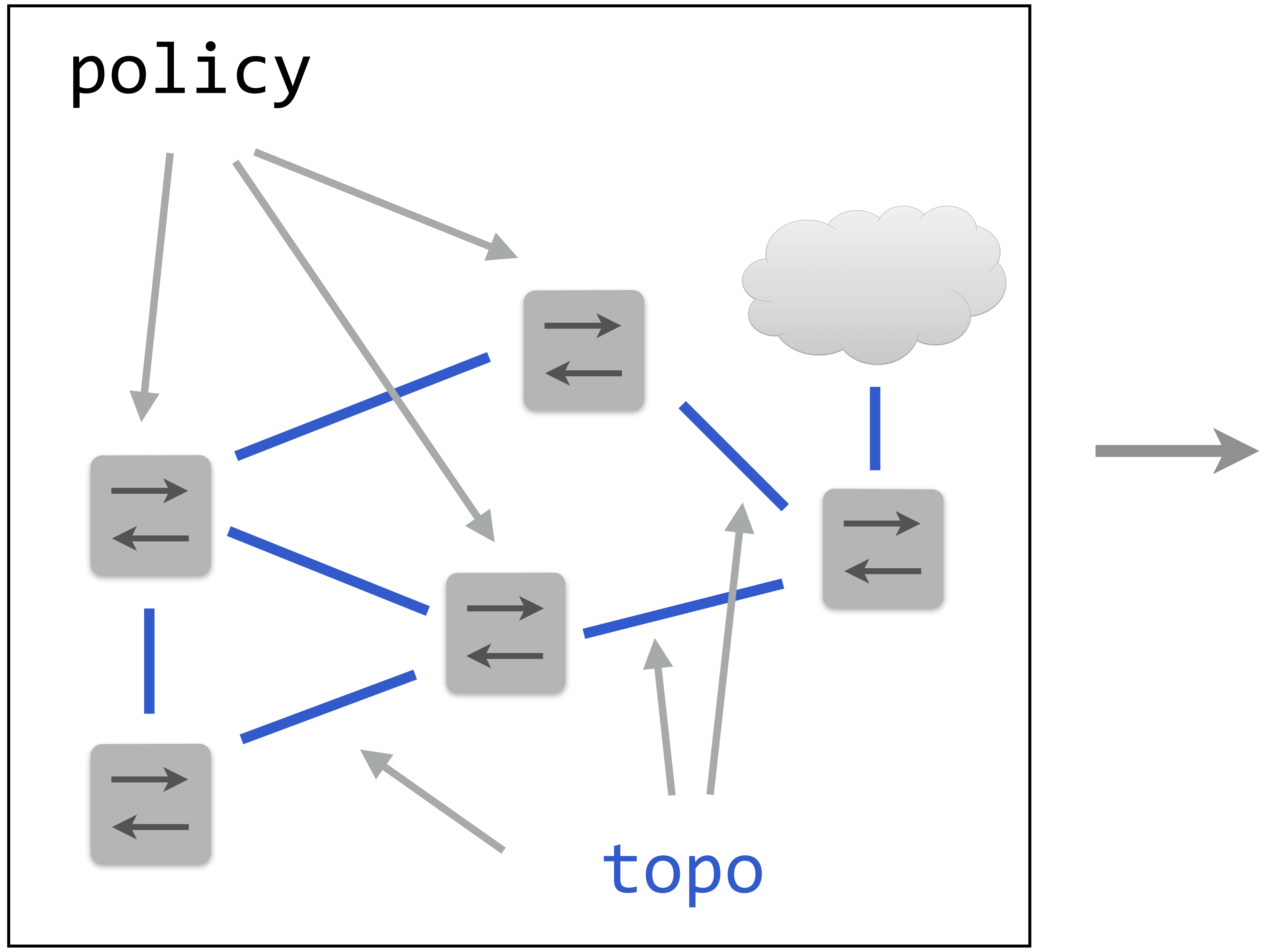
Networks in NetKAT

sw=6;pt=8;dst := 10.0.1.5;pt:=5

For all packets located at port 8 of switch 6, set the destination address to 10.0.1.5 and forward it out on port 5.

Networks in NetKAT

The behaviour of an entire network can be encoded in NetKAT
by interleaving steps of processions by switches and topology



policy
+
(policy; topo); policy
+
(policy; topo; policy; topo); policy
⋮
(policy; topo)*; policy

Semantics

$$[\![e]\!] : H \to 2^H$$

$$[\![x \leftarrow n]\!](\pi_1 :: \sigma) \triangleq \{\pi_1[n/x] :: \sigma\}$$

$$[\![x = n]\!](\pi_1 :: \sigma) \triangleq \begin{cases} \{\pi_1 :: \sigma\} & \text{if } \pi_1(x) = n \\ \varnothing & \text{if } \pi_1(x) \neq n \end{cases}$$

$$[\![\mathsf{dup}]\!](\pi_1 :: \sigma) \triangleq \{\pi_1 :: \pi_1 :: \sigma\}$$

Verification using NetKAT

Reachability

- Can host A communicate with host B? Can every host communicate with every other host?

Security

- Does all untrusted traffic pass through the intrusion detection system located at C?

Loop detection

- Is it possible for a packet to be forwarded around a cycle in the network?

Verification using NetKAT

Soundness and Completeness [Anderson et al. 14]

- $\vdash p = q$ if and only if $[\![p]\!] = [\![q]\!]$

Decision Procedure [Foster et al. 15]

- NetKAT coalgebra
- efficient bisimulation-based decision procedure
- implementation in OCaml
- deployed in the Frenetic suite of network management tools

Missing ingredient

NetKat

$$sw = 6; pt = 8; dst := 10.0.1.5; pt := 5$$

For all packets located at port 8 of switch 6, set the destination address to 10.0.1.5 and forward it out on port 5.

CKA

[Hoare et al., JLAMP '11] [Kappe et al., CONCUR '17, ESOP '18]

$$a; b \parallel c; d$$

*Thread 1: do **a** and then **b*** *Thread 2: do **c** and then **d***

Current explorations

Forwarding/ Filtering behaviour

Concurrency

Large data domains

NetKat **CALF** CKA

in collaboration with amazon

Other research directions

Software Analysis

- Learning the “correct ways” of using undocumented code
- Learning-based automated test generation

Hardware Analysis

- Analysing concurrency in hardware, in collaboration with arm