A Language-Based Approach to Network Verification and Synthesis

Nate Foster
Cornell University

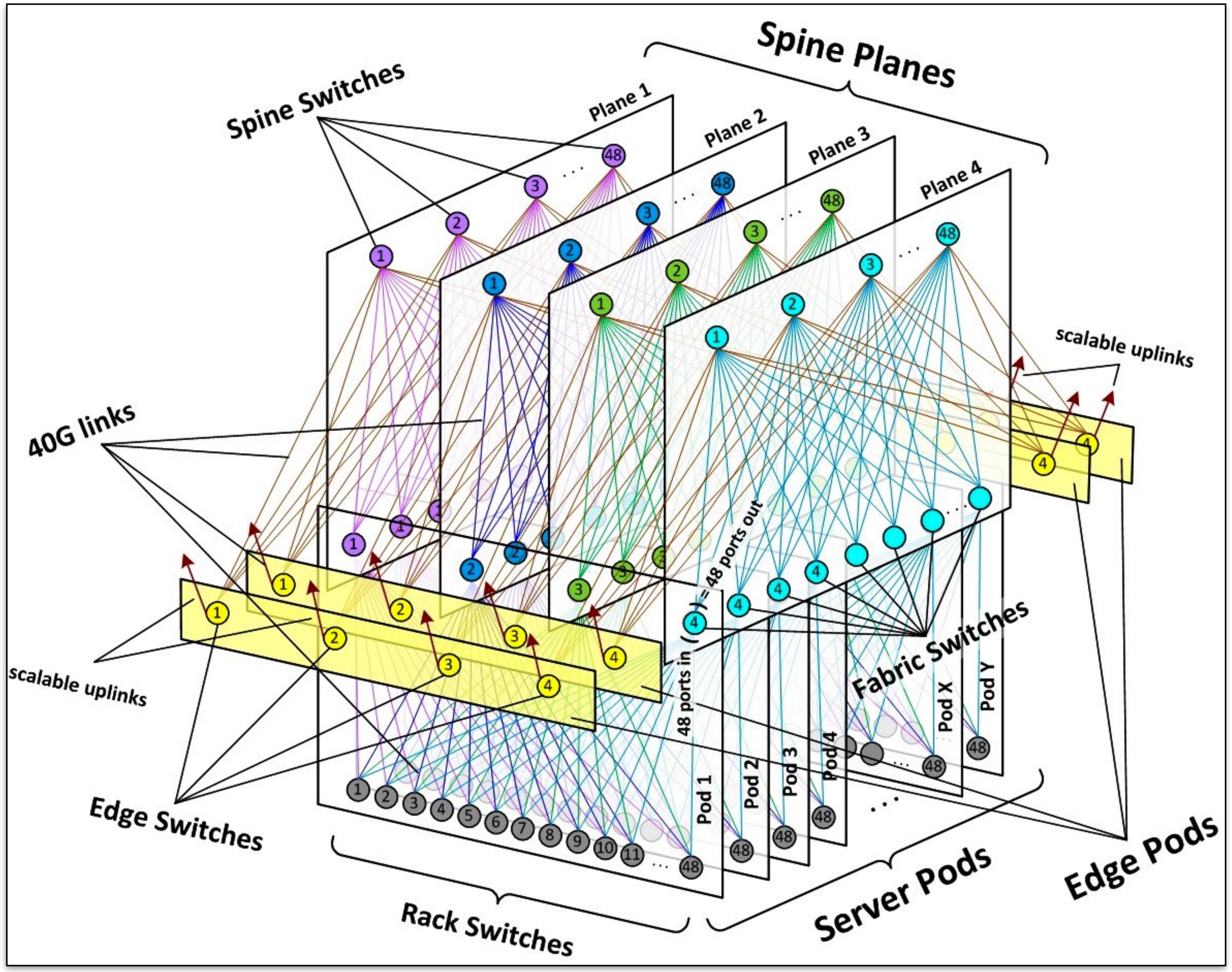
Microsoft Research
Faculty Summit 2015

Challenges

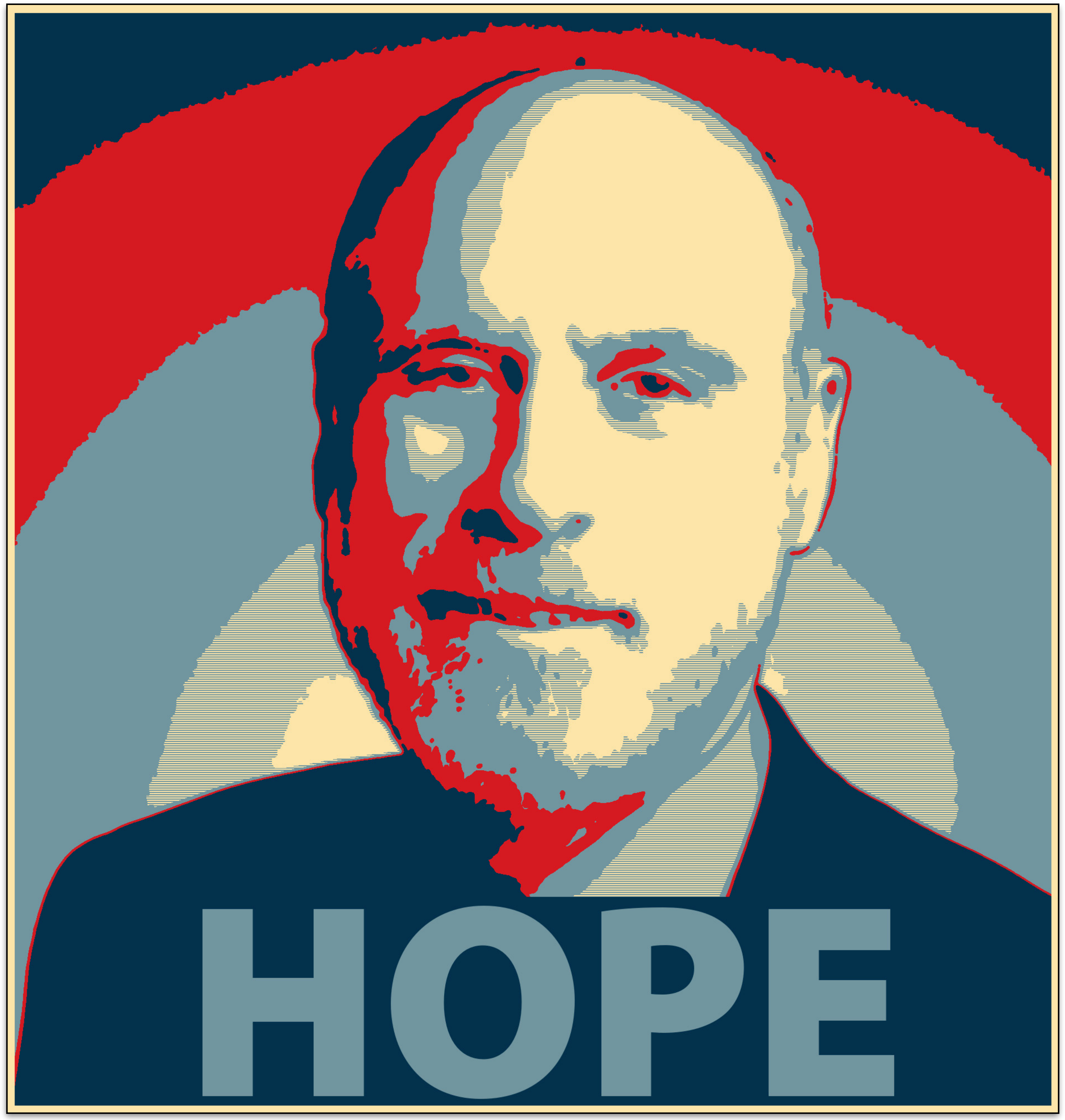
Networks are a critical part of our computing infrastructure...

...they have grown dramatically in size and complexity...

... and are quickly becoming unwieldy for operators to manage!



Network Management



Operators use a variety of techniques to keep networks running such as:

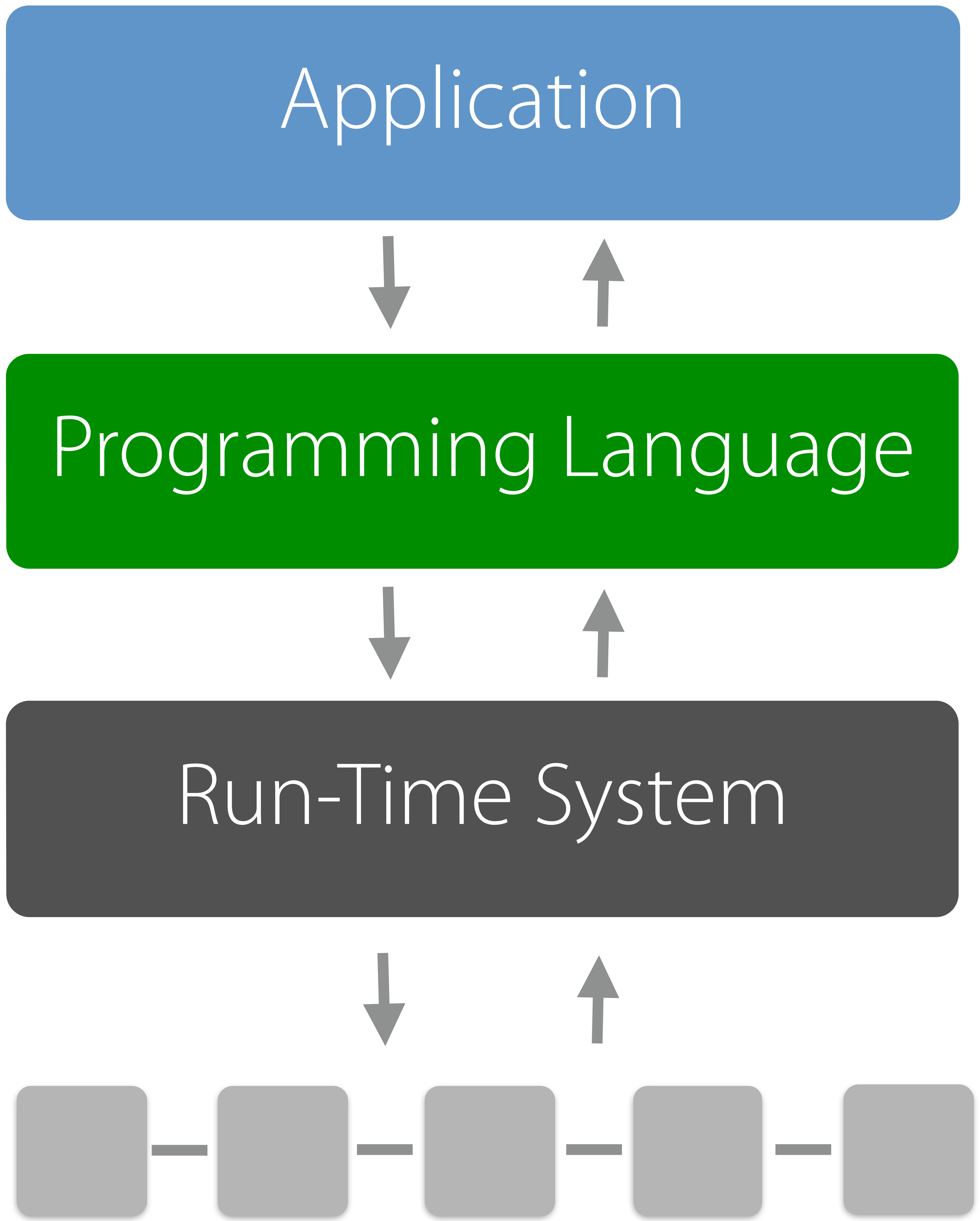
- Generating low-level configurations from high-level policies
- Scraping configurations using command-line interfaces
- Diagnosing errors using **`ping`** and **`traceroute`**

Toward Design Automation

1. Design high-level languages that model essential network features

2. Develop semantics that enables reasoning precisely about behavior

3. Build tools to synthesize low-level implementations automatically



Focus on *reachability properties* that capture the essential function of a network: moving data from one location to another

Machines

A *machine model* describes behavior in terms of concepts like pipelines of hardware lookup tables

Match	Actions
ethTyp=0x800, ipProto=0x06, tcpDstPort=22, ethSrc=00:00:00:00:00:01	Drop
ethTyp=0x800, ipProto=0x06, tcpDstPort=22, ethSrc=00:00:00:00:00:02	Drop
ethTyp=0x800, ipProto=0x06, tcpDstPort=22,	Inport
ethTyp=0x800, ipProto=0x06	Inport
ethType=0x800	Inport
*	Inport

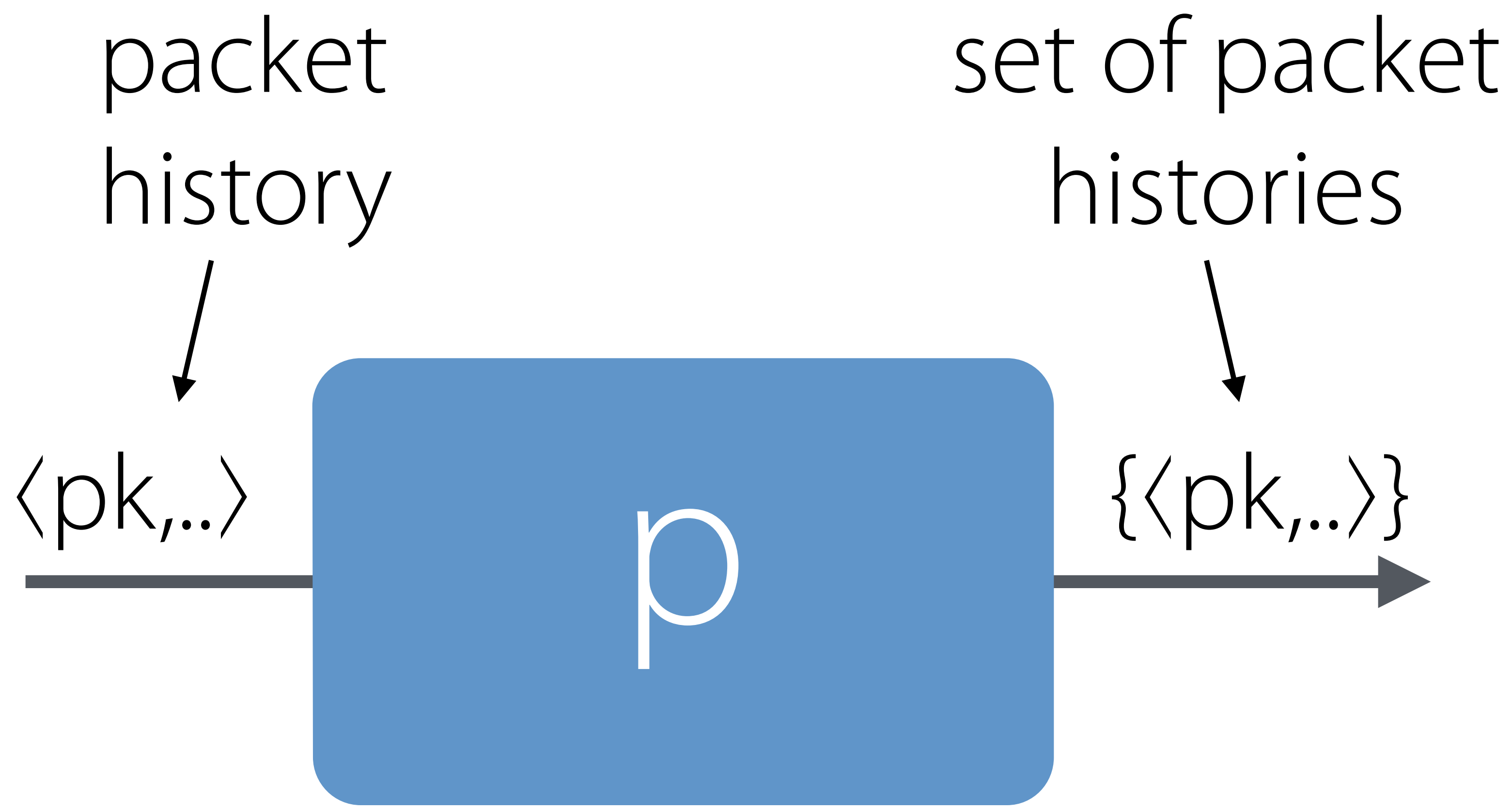
Machines

A *machine model* describes behavior in terms of concepts like pipelines of hardware lookup tables

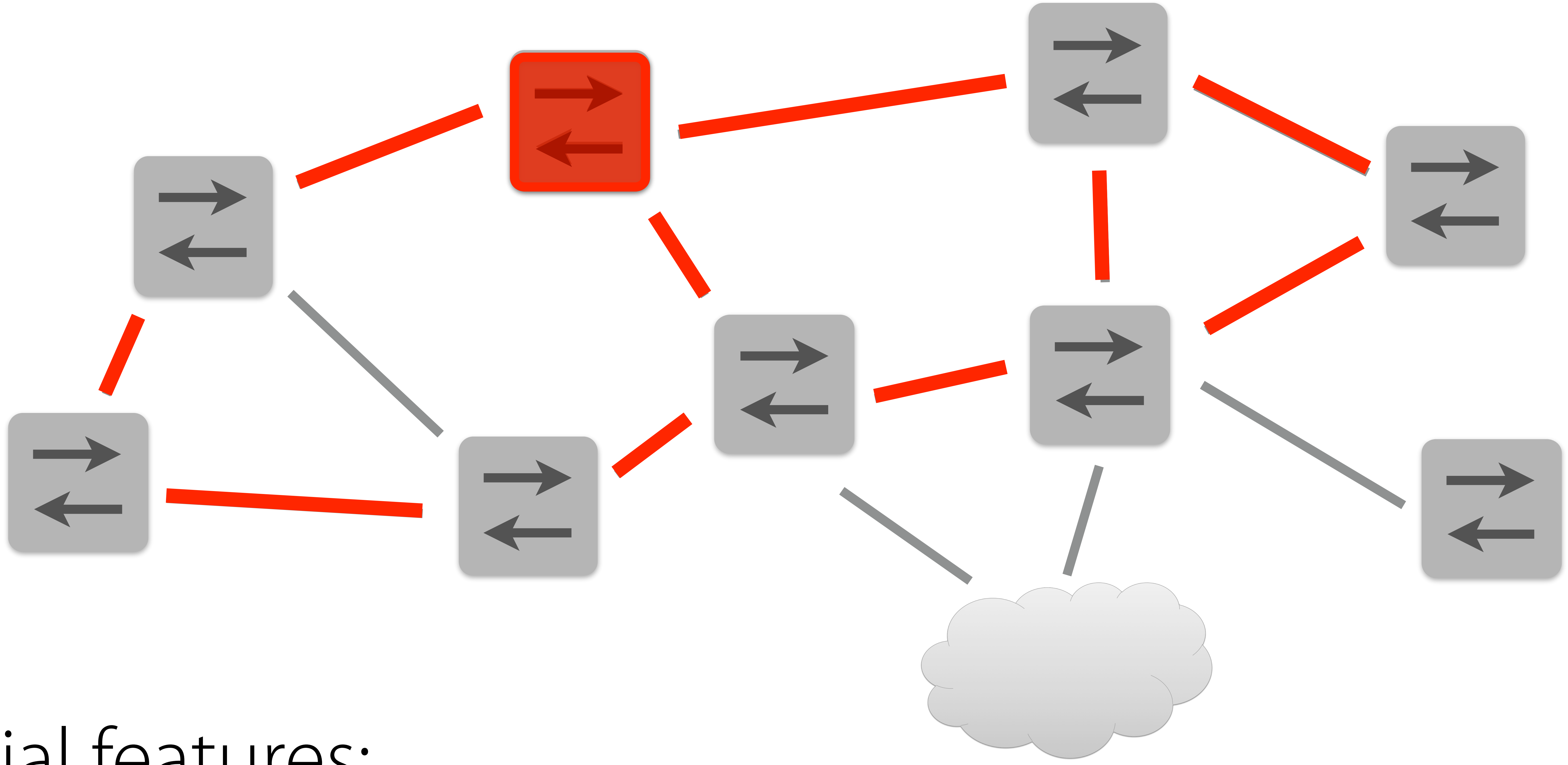
Match	Actions
ethTyp=0x800, ipProto=0x06, tcpDstPort=22, ethSrc=00:00:00:00:00:01	Drop
ethTyp=0x800, ipProto=0x06, tcpDstPort=22, ethSrc=00:00:00:00:00:02	Drop
ethTyp=0x800, ipProto=0x06, tcpDstPort=22,	Inport
ethTyp=0x800, ipProto=0x06	Inport
ethType=0x800	Inport
*	Inport

Languages

A *programming model* describes behavior in terms of concepts like mathematical functions on packets



What should a network programming language provide?



Two essential features:

- Packet classifiers
- Forwarding paths

NetKAT Language

[POPL '14]

$$
\begin{aligned}
\text{pol} ::= \; & \textbf{false} \\
\mid \; & \textbf{true} \\
\mid \; & \text{field} \;\mathbf{=}\; \text{val} \\
\mid \; & \text{pol}_1 \;\mathbf{+}\; \text{pol}_2 \\
\mid \; & \text{pol}_1 \;\mathbf{;}\; \text{pol}_2 \\
\mid \; & \mathbf{!}\text{pol} \\
\mid \; & \text{pol}^{*} \\
\mid \; & \text{field} \;\mathbf{:=}\; \text{val} \\
\mid \; & S \Rightarrow T
\end{aligned}
$$

NetKAT Language

[POPL '14]

pol ::= **false**
| **true**
| field **=** val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| **!**pol
| pol*
| field **:=** val
| S⇒T

Boolean
Algebra

NetKAT Language

[POPL '14]

pol ::= **false**
| **true**
| field = val
| pol_1 + pol_2
| pol_1 ; pol_2
| !pol
| pol*
| field := val
| S⇒T

Boolean
Algebra

+

Kleene
Algebra

NetKAT Language

[POPL '14]

pol ::= **false**
| **true**
| field **=** val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| **!**pol
| pol*
| field **:=** val
| S⇒T

Boolean Algebra

+

Kleene Algebra

+

Packet Primitives

NetKAT Language

[POPL '14]

pol ::= **false**
| **true**
| field **=** val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| **!**pol
| pol*
| field **:=** val
| S⇒T

Boolean Algebra
+
Kleene Algebra
} KAT [Kozen '96]

+

Packet Primitives

NetKAT Language

[POPL '14]

pol ::= **false**
| **true**
| field **=** val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| **!**pol
| pol*
| field **:=** val
| S⇒T

Boolean Algebra
+
Kleene Algebra

} KAT [Kozen '96]

+
Packet Primitives

} NetKAT [Anderson et al. '14]

NetKAT Language

[POPL '14]

pol ::= **false**
| **true**
| field = val
| $pol_1 + pol_2$
| !pol
| pol*
| field := val
| S⇒T

Boolean Algebra

+

+

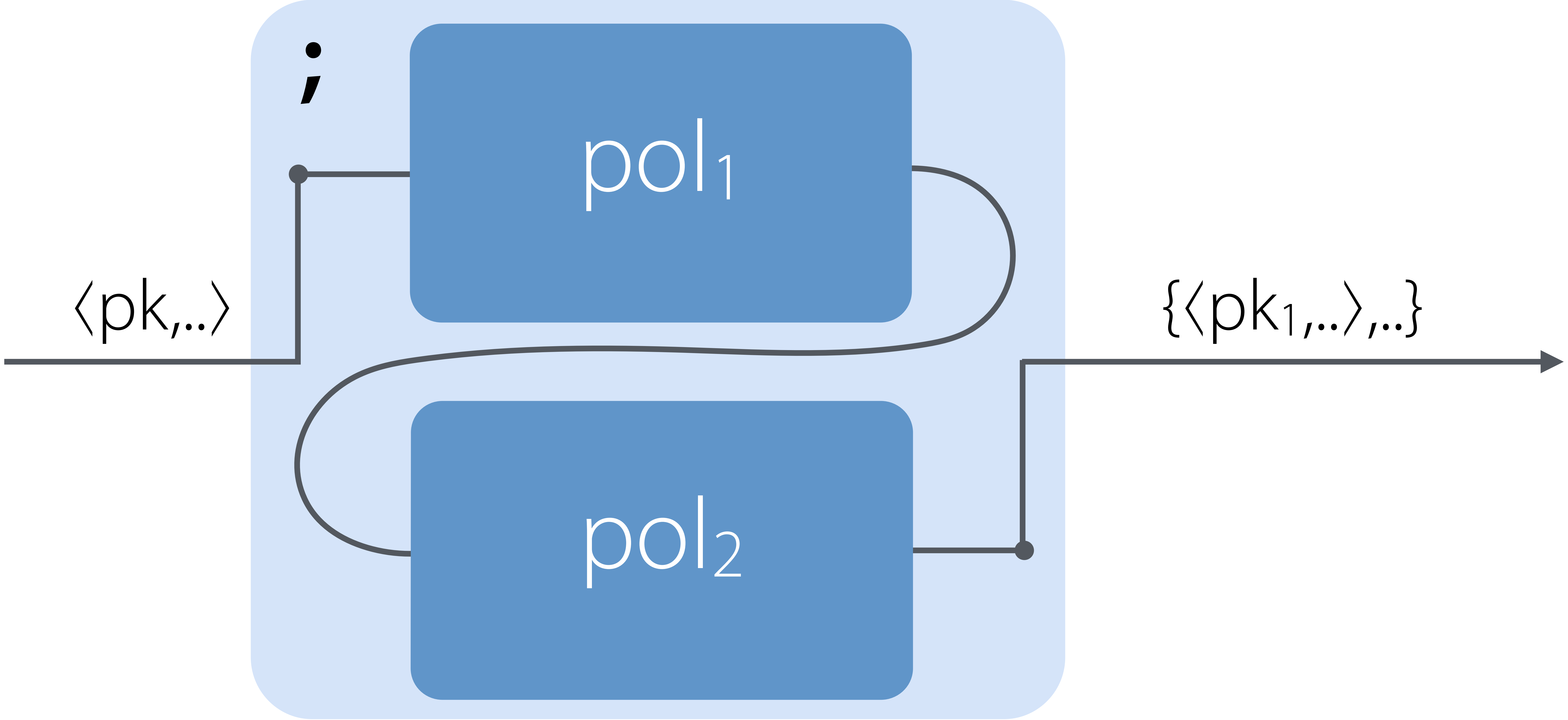
Packet Primitives

KAT

[Kozen '96]

$$\textbf{if } p_1 \textbf{ then } p_2 \textbf{ else } p_3 \triangleq (p_1 ; p_2) + (!p_1 ; p_3)$$

pol ::= **false**
| **true**
| field = val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| **!**pol
| pol*
| field **:=** val
| S⇒T



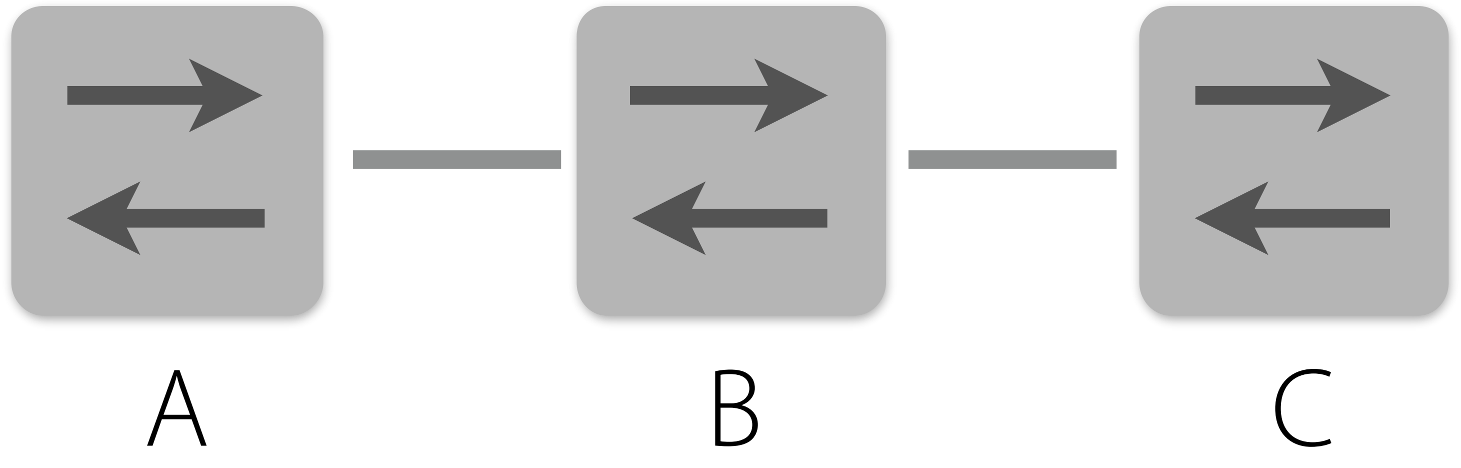
Sequential composition pol_1 **;** pol_2 runs the input through pol_1 and then runs every output through pol_2

Encodings

Switch forwarding tables and network topologies can be represented in NetKAT using simple encodings

Pattern	Actions
dstport=22	Drop
srcip=10.0.0.1	Forward 1
*	Forward 2

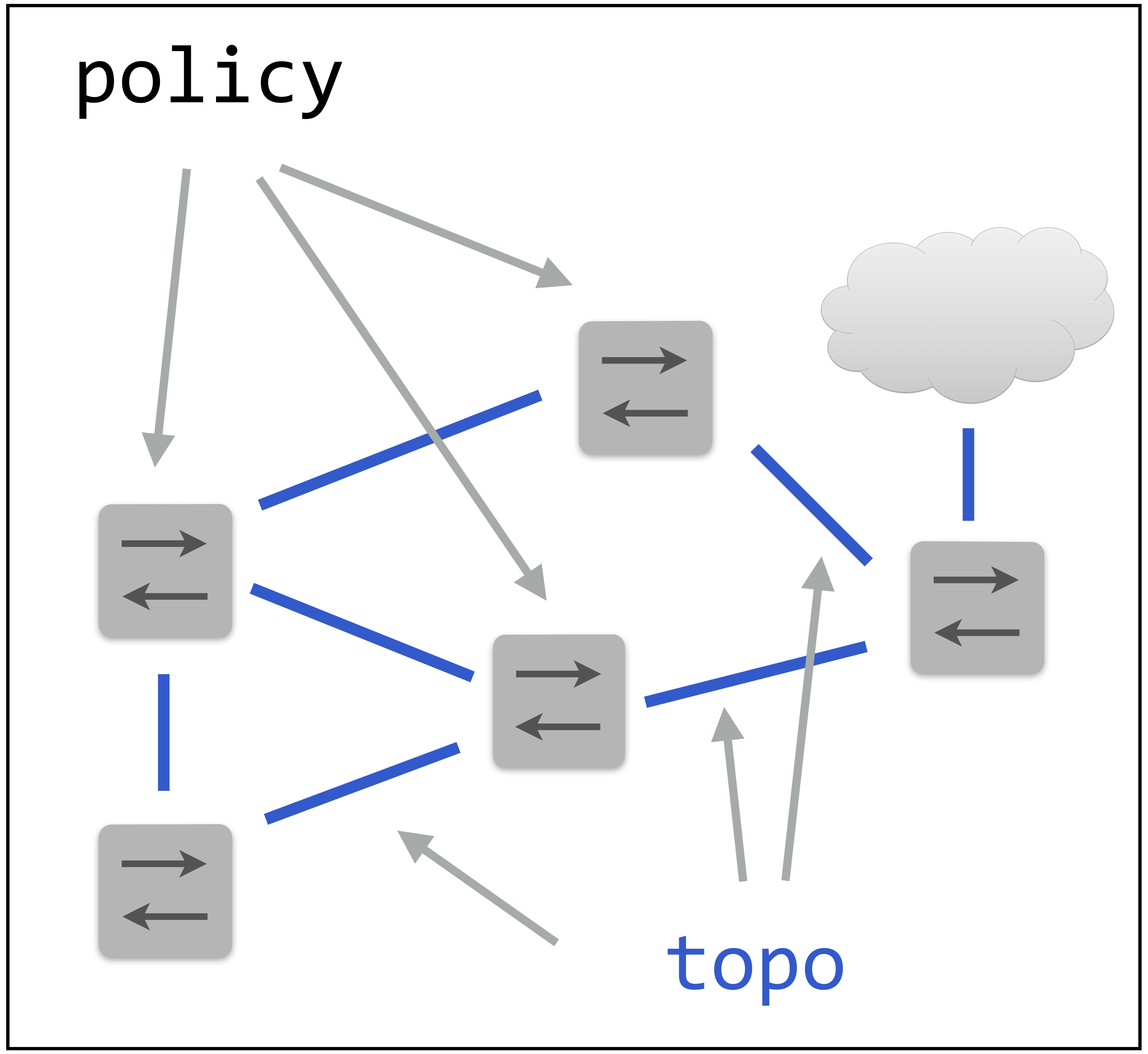
if dstport=22 **then false**
else if srcip=10.0.0.1 **then** port **:=** 1
else port **:=** 2



$$A \Rightarrow B + B \Rightarrow A + B \Rightarrow C + C \Rightarrow B$$

Networks

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



policy

+

(policy; topo); policy

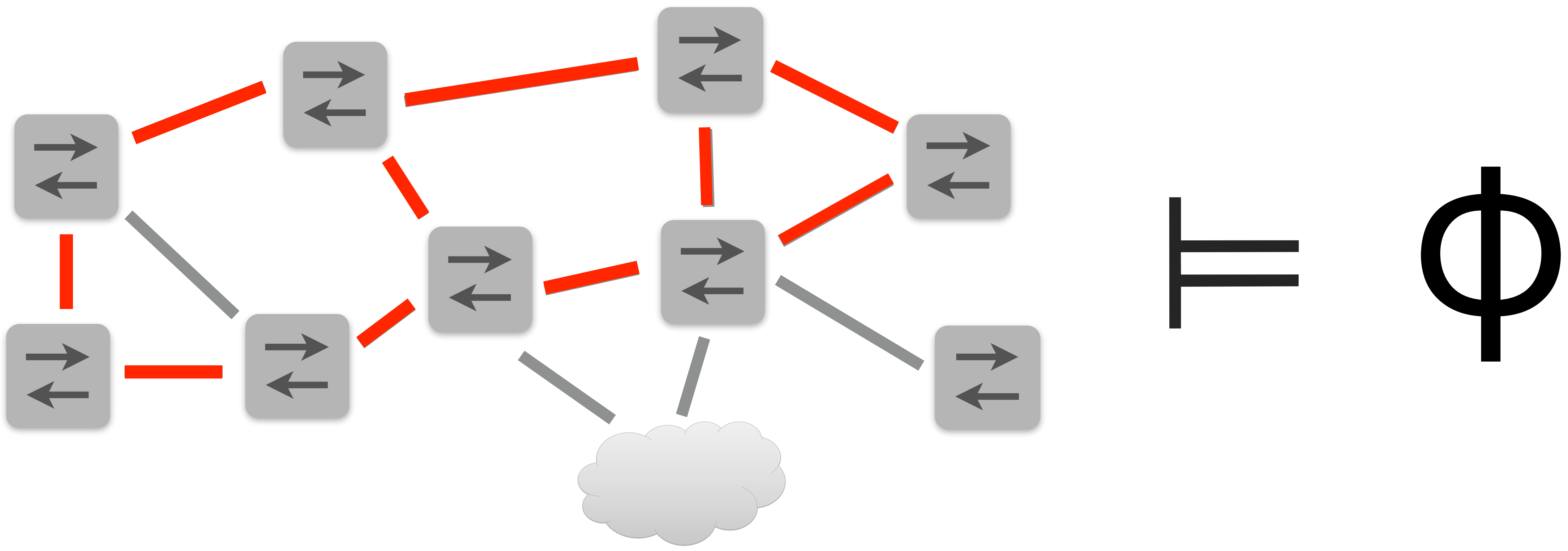
+

(policy; topo; policy; topo); policy

⋮

(policy; topo)*; policy

Reachability



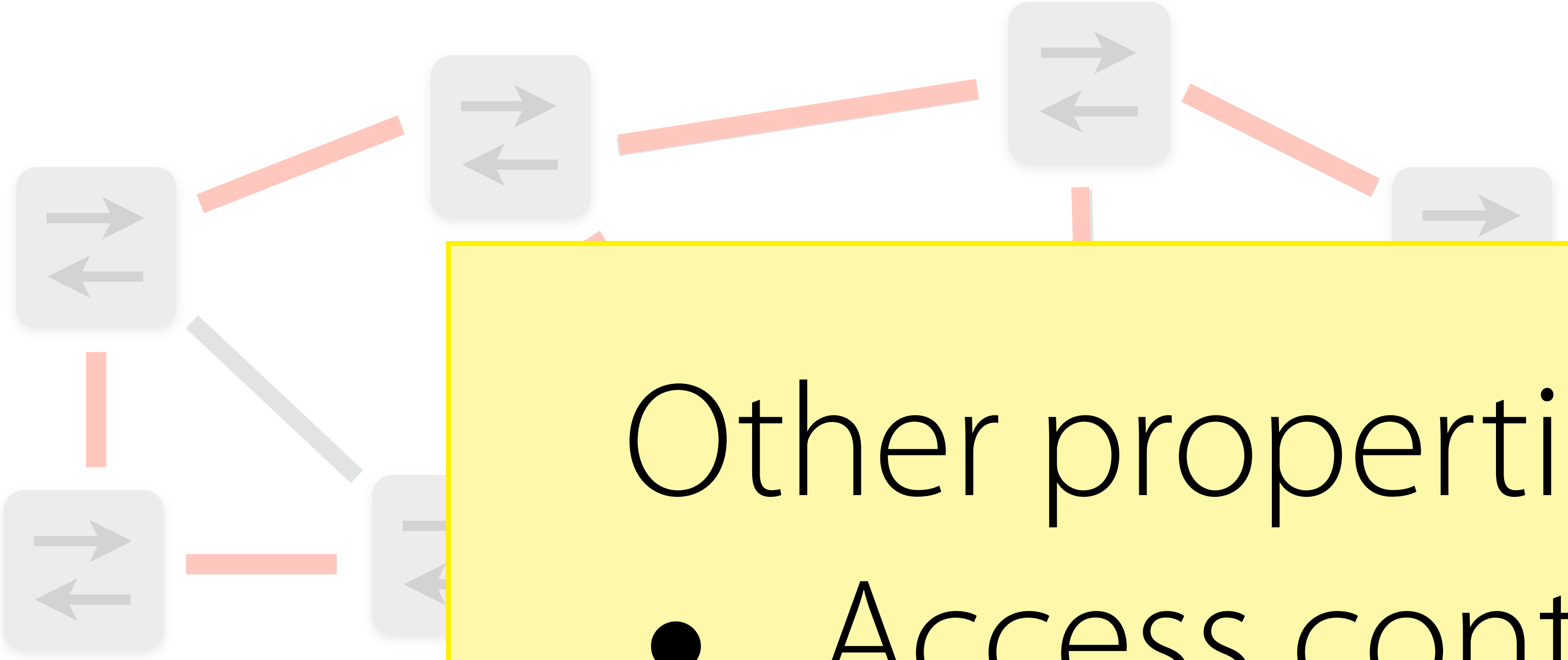
Given a network, want to be able to answer questions like:

"Does the network forward from ingress to egress?

Can reduce this question (and many others) to equivalence

$$\text{in; (policy; topo)}^*\text{; policy; out} \equiv \text{in; out}$$

Reachability



Other properties:

- Access control
- Traffic Isolation
- Loop freedom
- Blackhole freedom

Given a network, … questions like:

“Does the netw… egress?

Can reduce this question (and many others) to equivalence

$$in;\ (policy;\ topo)^*;\ policy;\ out \equiv in;\ out$$

NetKAT Proof System

Kleene Algebra Axioms

$p + (q + r) \equiv (p + q) + r$

$p + q \equiv q + p$

$p + \textbf{false} \equiv p$

$p + p \equiv p$

$p; (q; r) \equiv (p; q); r$

$p; (q + r) \equiv p; q + p; r$

$(p + q); r \equiv p; r + q; r$

$\textbf{true}; p \equiv p$

$p \equiv p; \textbf{true}$

$\textbf{false}; p \equiv \textbf{false}$

$p; \textbf{false} \equiv \textbf{false}$

$\textbf{true} + p; p^* \equiv p^*$

$\textbf{true} + p^*; p \equiv p^*$

$p + q; r + r \equiv r \Rightarrow p^*; q + r \equiv r$

$p + q; r + q \equiv q \Rightarrow p; r^* + q \equiv q$

Boolean Algebra Axioms

$a + (b ; c) \equiv (a + b) ; (a + c)$

$a + \textbf{true} \equiv \textbf{true}$

$a + !a \equiv \textbf{true}$

$a ; b \equiv b ; a$

$a ; !a \equiv \textbf{false}$

$a ; a \equiv a$

Packet Axioms

$f := n; f' := n' \equiv f' := n'; f := n$ if $f \neq f'$

$f := n; f' = n' \equiv f' = n'; f := n$ if $f \neq f'$

$f := n; f = n \equiv f := n$

$f = n; f := n \equiv f = n$

$f := n; f := n' \equiv f := n'$

$f = n; f = n' \equiv \textbf{false}$ if $n \neq n'$

$A \Rightarrow B; f = n \equiv f = n; A \Rightarrow B$ if $f \neq$ switch

NetKAT Proof System

Kleene Algebra Axioms

$p + (q + r) \equiv (p + q) + r$
$p + q \equiv q + p$
$p + \textbf{false} \equiv p$
$p + p \equiv p$
$p; (q; r) \equiv (p; q); r$
$p; (q + r) \equiv p; q + p; r$
$(p + q); r \equiv p; r + q; r$
$\textbf{true}; p \equiv p$
$p \equiv p; \textbf{true}$
$\textbf{false}; p \equiv \textbf{false}$
$p; \textbf{false} \equiv \textbf{false}$
true
true
p
p

Boolean Algebra Axioms

$a + (b ; c) \equiv (a + b) ; (a + c)$
$a + \textbf{true} \equiv \textbf{true}$
$a + !a \equiv \textbf{true}$
$a ; b \equiv b ; a$
$a ; !a \equiv \textbf{false}$
$a ; a \equiv a$

Packet Axioms

$f := n; f' := n' \equiv f' := n'; f := n$ if $f \neq f'$
$f := n; f' = n' \equiv f' = n'; f := n$ if $f \neq f'$
$f := n; f = n \equiv f := n$
$f = n; f := n \equiv f = n$
$f := n; f := n' \equiv f := n'$
$f = n; f = n' \equiv \textbf{false}$ if $n \neq n'$
$A \Rightarrow B; f = n \equiv f = n; A \Rightarrow B$ if $f \neq$ switch

NetKAT Proof System

Kleene Algebra Axioms

$p + (q + r) \equiv (p + q) + r$

$p + q \equiv q + p$

$p + \textbf{false} \equiv p$

$p + p \equiv p$

$p; (q; r) \equiv (p; q); r$

$p; (q + r) \equiv p; q + p; r$

$(p + q); r \equiv p; r + q; r$

$\textbf{true}; p \equiv p$

$p \equiv p; \textbf{true}$

$\textbf{false}; p \equiv \textbf{false}$

$p; \textbf{false} \equiv \textbf{false}$

true

true

p

p

Boolean Algebra Axioms

$a + (b ; c) \equiv (a + b) ; (a + c)$

$a + \textbf{true} \equiv \textbf{true}$

$a + !a \equiv \textbf{true}$

$a ; b \equiv b ; a$

$a ; !a \equiv \textbf{false}$

$a ; a \equiv a$

Packet Axioms

$f := n; f' := n' \equiv f' := n'; f := n$ if $f \neq f'$

$f := n; f' = n' \equiv f' = n'; f := n$ if $f \neq f'$

$f := n; f = n \equiv f := n$

$f = n; f := n \equiv f = n$

$f := n; f := n' \equiv f := n'$

$f = n; f = n' \equiv \textbf{false}$ if $n \neq n'$

$A \Rightarrow B; f = n \equiv f = n; A \Rightarrow B$ if $f \neq$ switch

NetKAT Proof System

Kleene Algebra Axioms

$p + (q + r) \equiv (p + q) + r$
$p + q \equiv q + p$
$p + \textbf{false} \equiv p$
$p + p \equiv p$
$p; (q; r) \equiv (p; q); r$
$p; (q + r) \equiv p; q + p; r$
$(p + q); r \equiv p; r + q; r$
$\textbf{true}; p \equiv p$
$p \equiv p; \textbf{true}$
$\textbf{false}; p \equiv \textbf{false}$
$p; \textbf{false} \equiv \textbf{false}$
true
true
p
p

Boolean Algebra Axioms

$a + (b ; c) \equiv (a + b) ; (a + c)$
$a + \textbf{true} \equiv \textbf{true}$
$a + !a \equiv \textbf{true}$
$a ; b \equiv b ; a$
$a ; !a \equiv \textbf{false}$
$a ; a \equiv a$

Packet Axioms

$f := n; f' := n' \equiv f' := n'; f := n \quad \text{if } f \neq f'$
$f := n; f' = n' \equiv f' = n'; f := n \quad \text{if } f \neq f'$
$f := n; f = n \equiv f := n$
$f = n; f := n \equiv f = n$
$f := n; f := n' \equiv f := n'$
$f = n; f = n' \equiv \textbf{false} \quad \text{if } n \neq n'$
$A \Rightarrow B; f = n \equiv f = n; A \Rightarrow B \quad \text{if } f \neq \text{switch}$

NetKAT Proof System

Kleene Algebra Axioms

$p + (q + r) \equiv (p + q) + r$
$p + q \equiv q + p$
$p + \textbf{false} \equiv p$
$p + p \equiv p$
$p; (q; r) \equiv (p; q); r$
$p; (q + r) \equiv p; q + p; r$
$(p + q); r \equiv p; r + q; r$
$\textbf{true}; p \equiv p$
$p \equiv p; \textbf{true}$
$\textbf{false}; p \equiv \textbf{false}$
$p; \textbf{false} \equiv \textbf{false}$
true
true
p
p

Boolean Algebra Axioms

$a + (b ; c) \equiv (a + b) ; (a + c)$
$a + \textbf{true} \equiv \textbf{true}$
$a + !a \equiv \textbf{true}$
$a ; b \equiv b ; a$
$a ; !a \equiv \textbf{false}$
$a ; a \equiv a$

Packet Axioms

$f := n; f' := n' \equiv f' := n'; f := n$ if $f \neq f'$
$f := n; f' = n' \equiv f' = n'; f := n$ if $f \neq f'$
$f := n; f = n \equiv f := n$
$f = n; f := n \equiv f = n$
$f := n; f := n' \equiv f := n'$
$f = n; f = n' \equiv \textbf{false}$ if $n \neq n'$
$A \Rightarrow B; f = n \equiv f = n; A \Rightarrow B$ if $f \neq$ switch

NetKAT Proof System

Kleene Algebra Axioms

$p + (q + r) \equiv (p + q) + r$

$p + q \equiv q + p$

$p + \textbf{false} \equiv p$

$p + p \equiv p$

$p; (q; r) \equiv (p; q); r$

$p; (q + r) \equiv p; q + p; r$

$(p + q); r \equiv p; r + q; r$

$\textbf{true}; p \equiv p$

$p \equiv p; \textbf{true}$

$\textbf{false}; p \equiv \textbf{false}$

$p; \textbf{false} \equiv \textbf{false}$

true

true

p

p

Boolean Algebra Axioms

$a + (b ; c) \equiv (a + b) ; (a + c)$

$a + \textbf{true} \equiv \textbf{true}$

$a + !a \equiv \textbf{true}$

$a ; b \equiv b ; a$

$a ; !a \equiv \textbf{false}$

$a ; a \equiv a$

Packet Axioms

$f := n; f' := n' \equiv f' := n'; f := n$ if $f \neq f'$

$f := n; f' = n' \equiv f' = n'; f := n$ if $f \neq f'$

$f := n; f = n \equiv f := n$

$f = n; f := n \equiv f = n$

$f := n; f := n' \equiv f := n'$

$f = n; f = n' \equiv \textbf{false}$ if $n \neq n'$

$A \Rightarrow B; f = n \equiv f = n; A \Rightarrow B$ if $f \neq$ switch

NetKAT Proof System

Soundness: If $\vdash p \equiv q$, then $[\![p]\!] = [\![q]\!]$

Completeness: If $[\![p]\!] = [\![q]\!]$, then $\vdash p \equiv q$

Kleene Algebra Axioms

$p + (q + r) \equiv (p + q) + r$

$p + q \equiv q + p$

p + **fa**

p + p

p; (q;

p; (q

(p + q

true;

p ≡ p; **true**

false; p ≡ **false**

p; **false** ≡ **false**

true

true

p

p

Boolean Algebra Axioms

$a + (b ; c) \equiv (a + b) ; (a + c)$

a + **true** ≡ **true**

$f := n; f' := n' \equiv f' := n'; f := n$ if $f \neq f'$

$f := n; f' = n' \equiv f' = n'; f := n$ if $f \neq f'$

$f := n; f = n \equiv f := n$

$f = n; f := n \equiv f = n$

$f := n; f := n' \equiv f := n'$

$f = n; f = n' \equiv$ **false** if $n \neq n'$

$A \Rightarrow B; f = n \equiv f = n; A \Rightarrow B$ if $f \neq$ switch

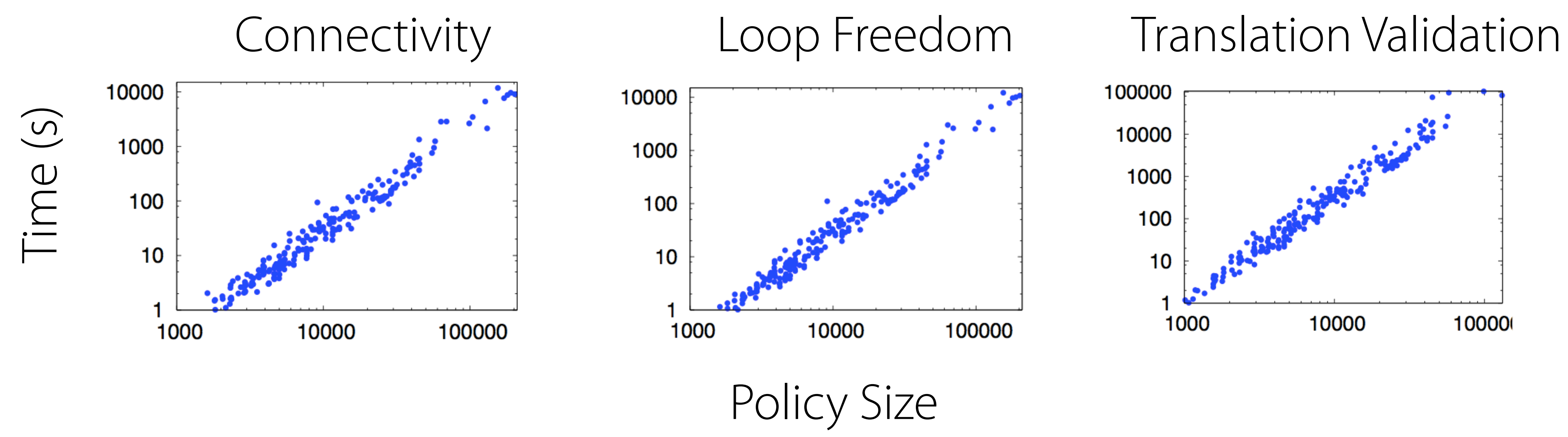
NetKAT Automata

[POPL '15]

Can exploit NetKAT's regular structure to build equivalent finite automata

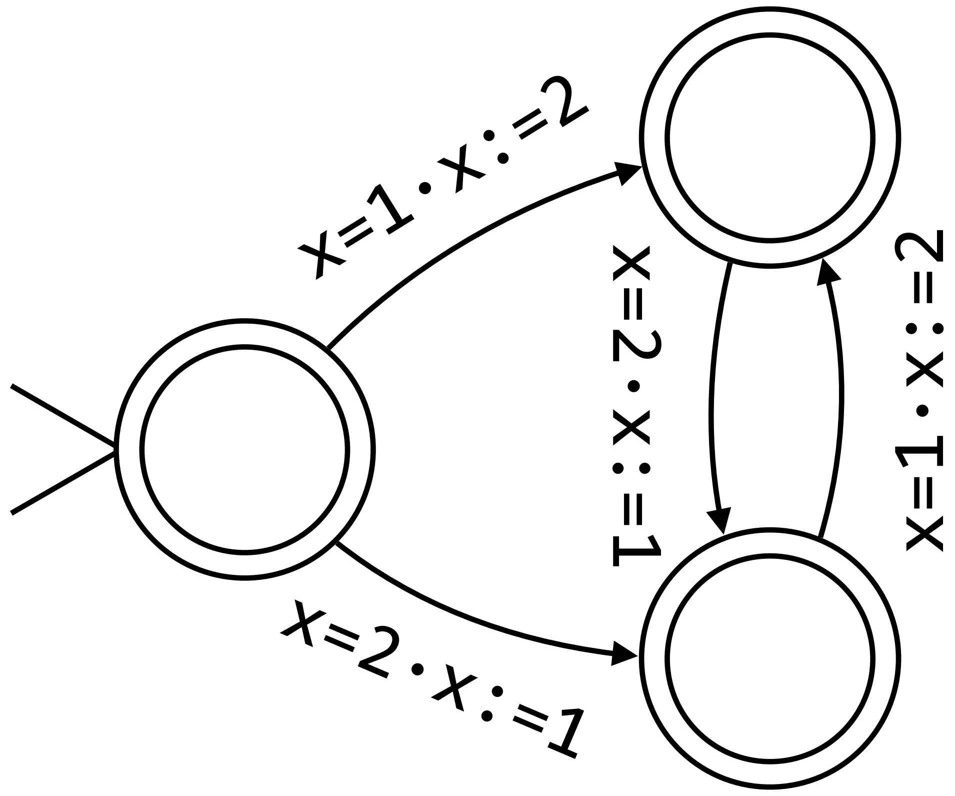
Automata provide a practical way to decide program equivalence

Prototype implementation performs well on Topology Zoo benchmarks



(x=1; x:=2; A⇒B +

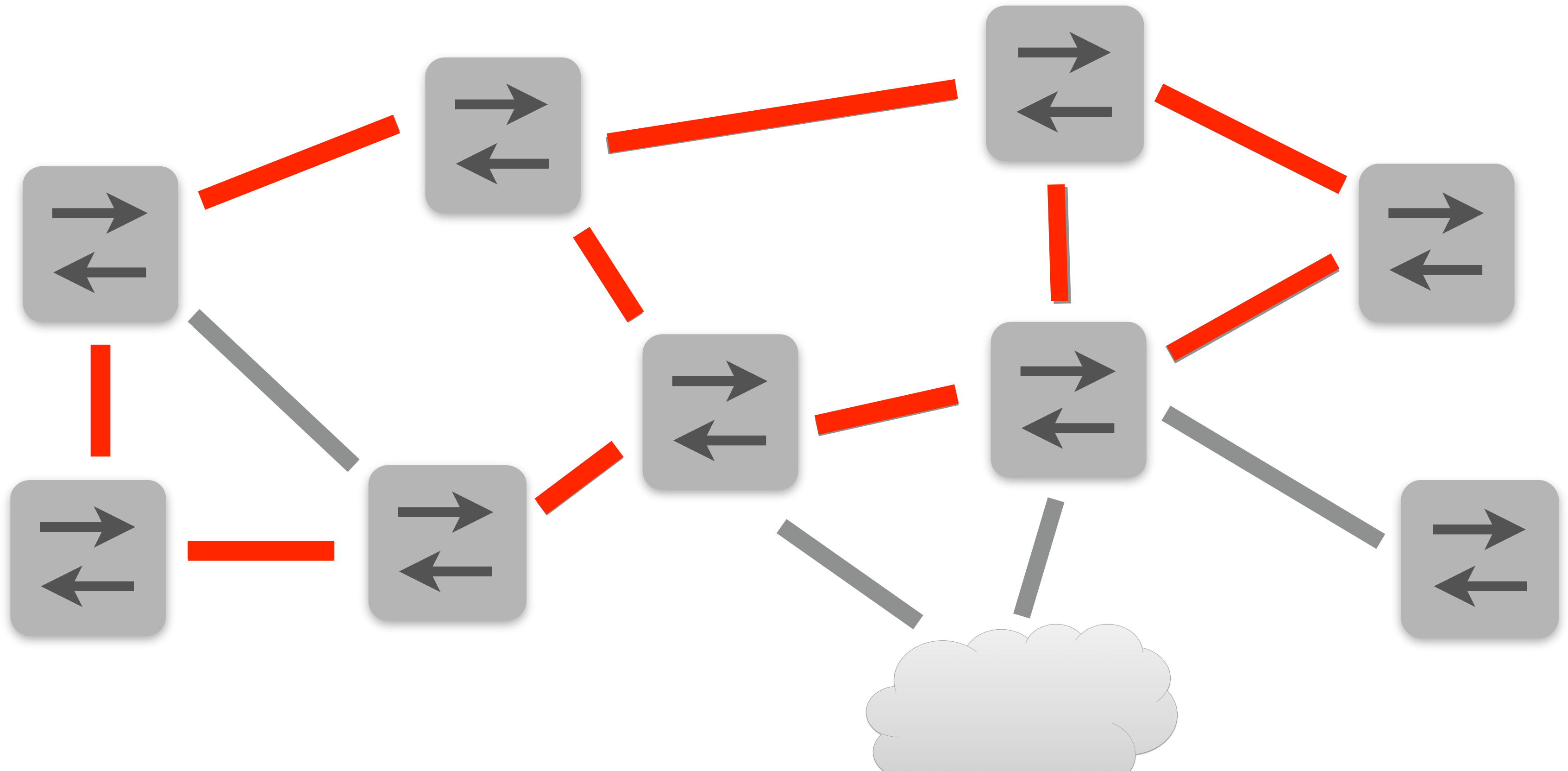
x=2; x:=1; B⇒A)*

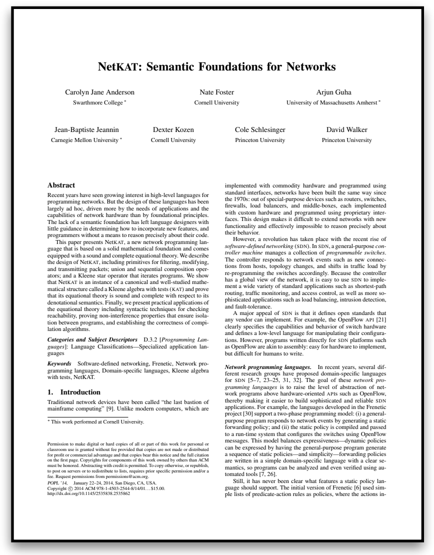


Other Applications

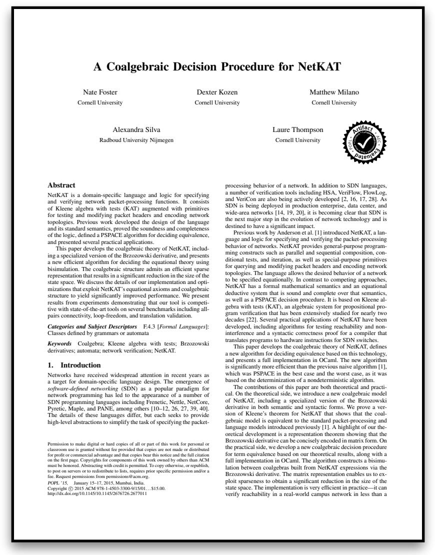
Regular paths have many uses:

- Network Virtualization
- Traffic Engineering
- Fault Tolerance
- Application Intent




NetKAT: Semantic Foundations for Networks

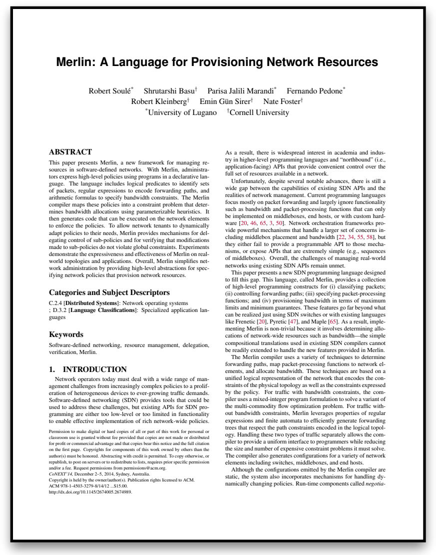
[POPL '14]


A Coalgebraic Decision Procedure for NetKAT

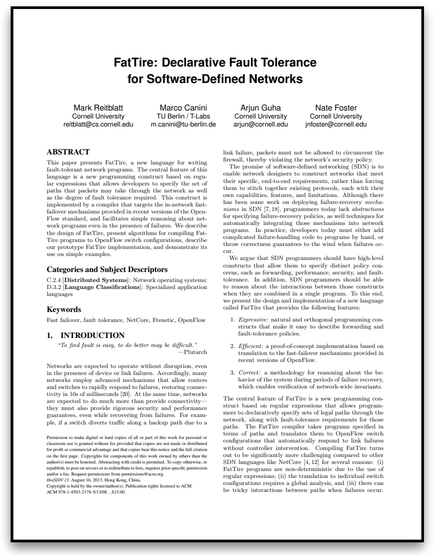
[POPL '15]


A Fast Compiler for NetKAT

[ICFP '15]


Merlin: A Language for Provisioning Network Resources

[CoNext '14]


FatTire: Declarative Fault Tolerance for Software-Defined Networks

[HotSDN '13]

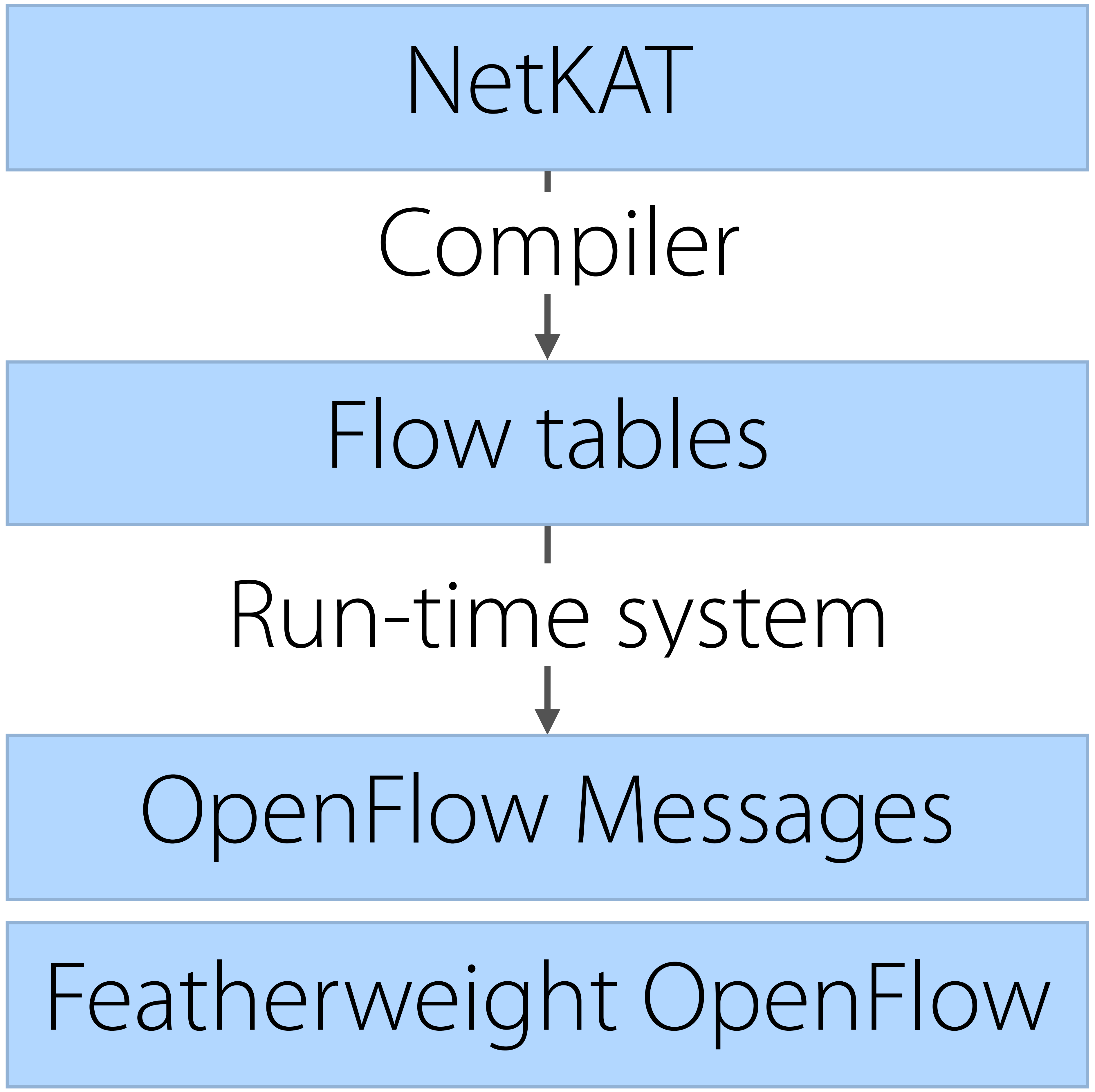
Verified Implementation

[PLDI '13]

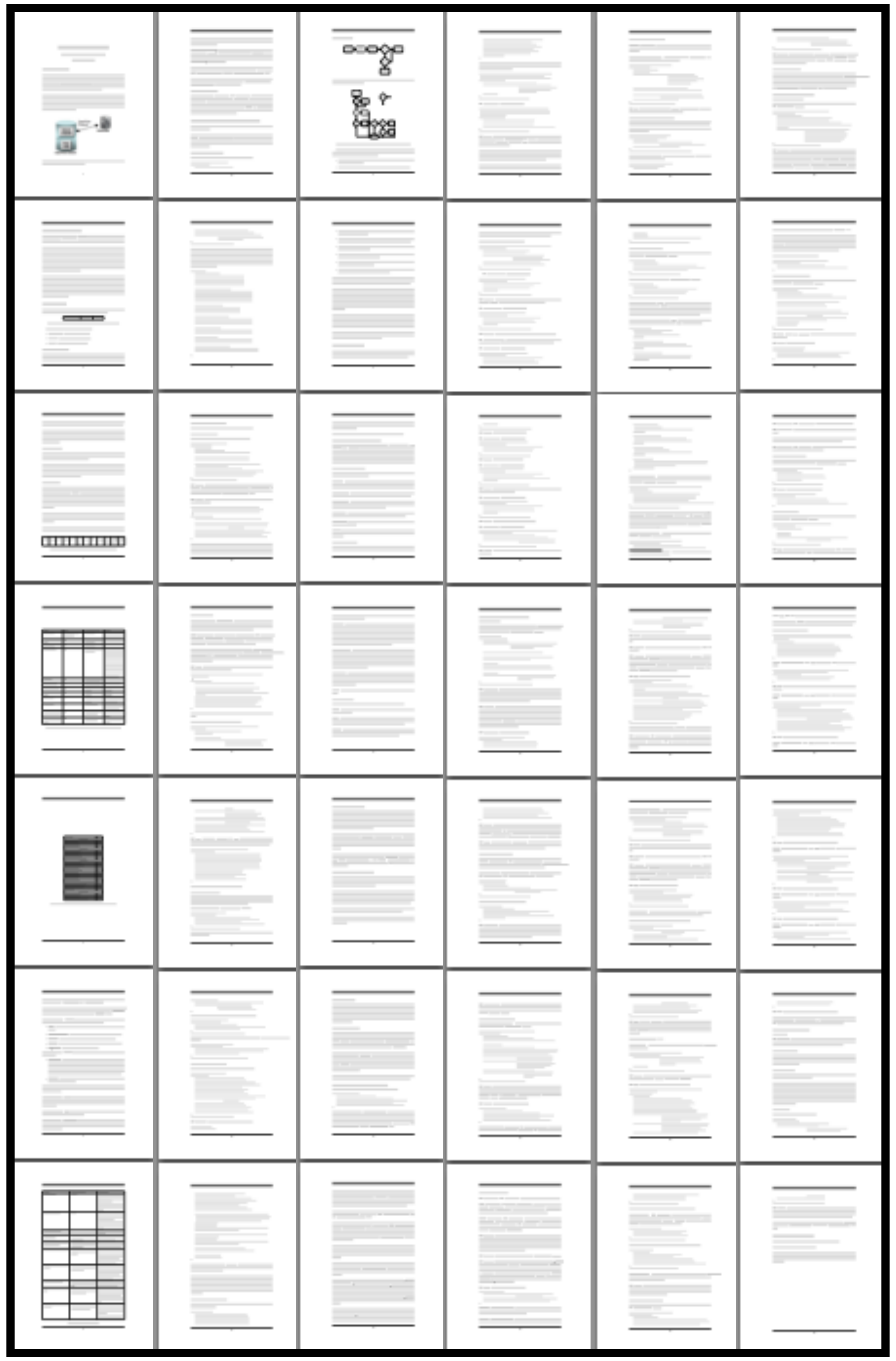
Question: How can we know the NetKAT compiler is correct?

Answer: implement it in a proof assistant!

- Formalize source and target languages in Coq
- Prove that transformations preserve semantics
- Extract code to OCaml and execute on real hardware



OpenFlow Specification



42 pages...

...of informal prose

...diagrams and flow charts

...and C struct definitions

Featherweight OpenFlow

Syntax

Devices	Switch	S	$::= \mathbb{S}(sw, pts, RT, inp.outp, inm, out$
	Controller	C	$::= \mathbb{C}(\sigma, f_{in}, f_{out})$
	Link	L	$::= \mathbb{L}(loc_{src}, pks, loc_{dst})$
	OpenFlow Link to Controller	M	$::= \mathbb{M}(sw, SMS, CMS)$
Packets and Locations	Packet	pk	$::= abstract$
	Switch ID	sw	$\in \mathbb{N}$
	Port ID	pt	$\in \mathbb{N}$
	Location	loc	$\in sw \times pt$
	Located Packet	lp	$\in loc \times pk$
Controller Components	Controller state	σ	$::= abstract$
	Controller input relation	f_{in}	$\in sw \times CM \times \sigma \rightsquigarrow \sigma$
	Controller output relation	f_{out}	$\in \sigma \rightsquigarrow sw \times SM \times \sigma$
Switch Components	Rule table	RT	$::= abstract$
	Rule table Interpretation	$[\![RT]\!]$	$\in lp \to \{\!\| lp_1 \cdots lp_n \|\!\} \times \{\!\| CM_1 \cdots C$
	Rule table modifier	ΔRT	$::= abstract$
	Rule table modifier interpretation	apply	$\in \Delta RT \to RT \to \Delta RT$
	Ports on switch	pts	$\in \{pt_1 \cdots pt_n\}$
	Consumed packets	inp	$\in \{\!\| lp_1 \cdots lp_n \|\!\}$
	Produced packets	$outp$	$\in \{\!\| lp_1 \cdots lp_n \|\!\}$
	Messages from controller	inm	$\in \{\!\| SM_1 \cdots SM_n \|\!\}$
	Messages to controller	$outm$	$\in \{\!\| CM_1 \cdots CM_n \|\!\}$
Link Components	Endpoints	loc_{src}, loc_{dst}	$\in loc$ where $loc_{src} \neq loc_{dst}$
	Packets from loc_{src} to loc_{dst}	pks	$\in [pk_1 \cdots pk_n]$
Controller Link	Message queue from controller	SMS	$\in [SM_1 \cdots SM_n]$
	Message queue to controller	CMS	$\in [CM_1 \cdots CM_n]$
Abstract OpenFlow Protocol	Message from controller	SM	$::= \textbf{FlowMod}\ \Delta RT \mid \textbf{PktOut}\ pt\ p$
	Message to controller	CM	$::= \textbf{PktIn}\ pt\ pk \mid \textbf{BarrierReply}\ n$

- Models all features related to packet forwarding and *all* essential asynchrony
- Supports arbitrary controllers

Semantics

$$\frac{(outp', outm') = [\![RT]\!](lp)}{\mathbb{S}(sw, pts, RT, \{\!\| lp \|\!\} \uplus inp, outp, inm, outm) \xrightarrow{lp} \mathbb{S}(sw, pts, RT, inp, outp' \uplus outp, inm, outm' \uplus outm)} \text{(Pkt-Process)}$$

$$\frac{}{\begin{array}{ll} & \mathbb{S}(sw, pts, RT, inp, \{\!\| (sw, pt, pk) \|\!\} \uplus outp, inm, outm) \mid \mathbb{L}((sw, pt), pks, loc') \\ \longrightarrow & \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \mid \mathbb{L}((sw, pt), [pk] +\!\!+ pks, loc') \end{array}} \text{(Send-Wire)}$$

$$\frac{}{\begin{array}{ll} & \mathbb{L}(loc, pks +\!\!+ [pk], (sw, pt)) \mid \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \\ \xrightarrow{(sw,pt,pk)} & \mathbb{L}(loc, pks, (sw, pt)) \mid \mathbb{S}(sw, pts, RT, \{\!\| (sw, pt, pk) \|\!\} \uplus inp, outp, inm, outm) \end{array}} \text{(Recv-Wire)}$$

$$\frac{RT' = \text{apply}(\Delta RT, RT)}{\mathbb{S}(sw, pts, RT, inp, outp, \{\!\| \textbf{FlowMod}\ \Delta RT \|\!\} \uplus inm, outm) \longrightarrow \mathbb{S}(sw, pts, RT', inp, outp, inm, outm)} \text{(Switch-FlowMod)}$$

$$\frac{pt \in pts}{\mathbb{S}(sw, pts, RT, inp, outp, \{\!\| \textbf{PktOut}\ pt\ pk \|\!\} \uplus inm, outm) \longrightarrow \mathbb{S}(sw, pts, RT, inp, \{\!\| (sw, pt, pk) \|\!\} \uplus outp, inm, outm)} \text{(Switch-PktOut)}$$

$$\frac{f_{out}(\sigma) \rightsquigarrow (sw, SM, \sigma')}{\mathbb{C}(\sigma, f_{in}, f_{out}) \mid \mathbb{M}(sw, SMS, CMS) \longrightarrow \mathbb{C}(\sigma', f_{in}, f_{out}) \mid \mathbb{M}(sw, [SM] +\!\!+ SMS, CMS)} \text{(Ctrl-Send)}$$

$$\frac{f_{in}(sw, \sigma, CM) \rightsquigarrow \sigma'}{\mathbb{C}(\sigma, f_{in}, f_{out}) \mid \mathbb{M}(sw, SMS, CMS +\!\!+ [CM]) \longrightarrow \mathbb{C}(\sigma', f_{in}, f_{out}) \mid \mathbb{M}(sw, SMS, CMS)} \text{(Ctrl-Recv)}$$

$$\frac{SM \neq \textbf{BarrierRequest}\ n}{\begin{array}{ll} & \mathbb{M}(sw, SMS +\!\!+ [SM], CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \\ \longrightarrow & \mathbb{M}(sw, SMS, CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, \{\!\| SM \|\!\} \uplus inm, outm) \end{array}} \text{(Switch-Recv-Ctrl)}$$

$$\frac{}{\begin{array}{ll} & \mathbb{M}(sw, SMS +\!\!+ [\textbf{BarrierRequest}\ n], CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, \emptyset, outm) \\ \longrightarrow & \mathbb{M}(sw, SMS, CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, \emptyset, \{\!\| \textbf{BarrierReply}\ n \|\!\} \uplus outm) \end{array}} \text{(Switch-Recv-Barrier)}$$

$$\frac{}{\begin{array}{ll} & \mathbb{S}(sw, pts, RT, inp, outp, inm, \{\!\| CM \|\!\} \uplus outm) \mid \mathbb{M}(sw, SMS, CMS) \\ \longrightarrow & \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \mid \mathbb{M}(sw, SMS, [CM] +\!\!+ CMS) \end{array}} \text{(Switch-Send-Ctrl)}$$

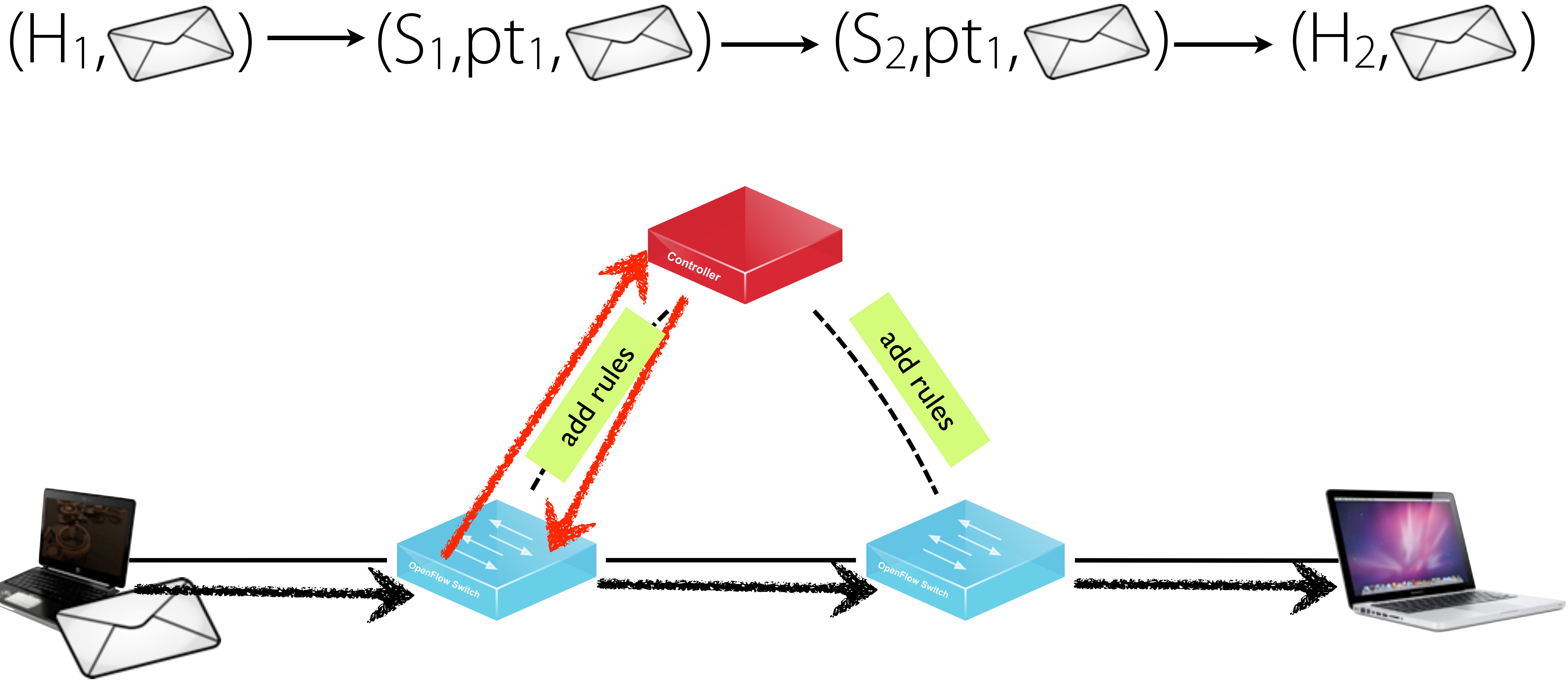
Weak Bisimulation

(H_1, )

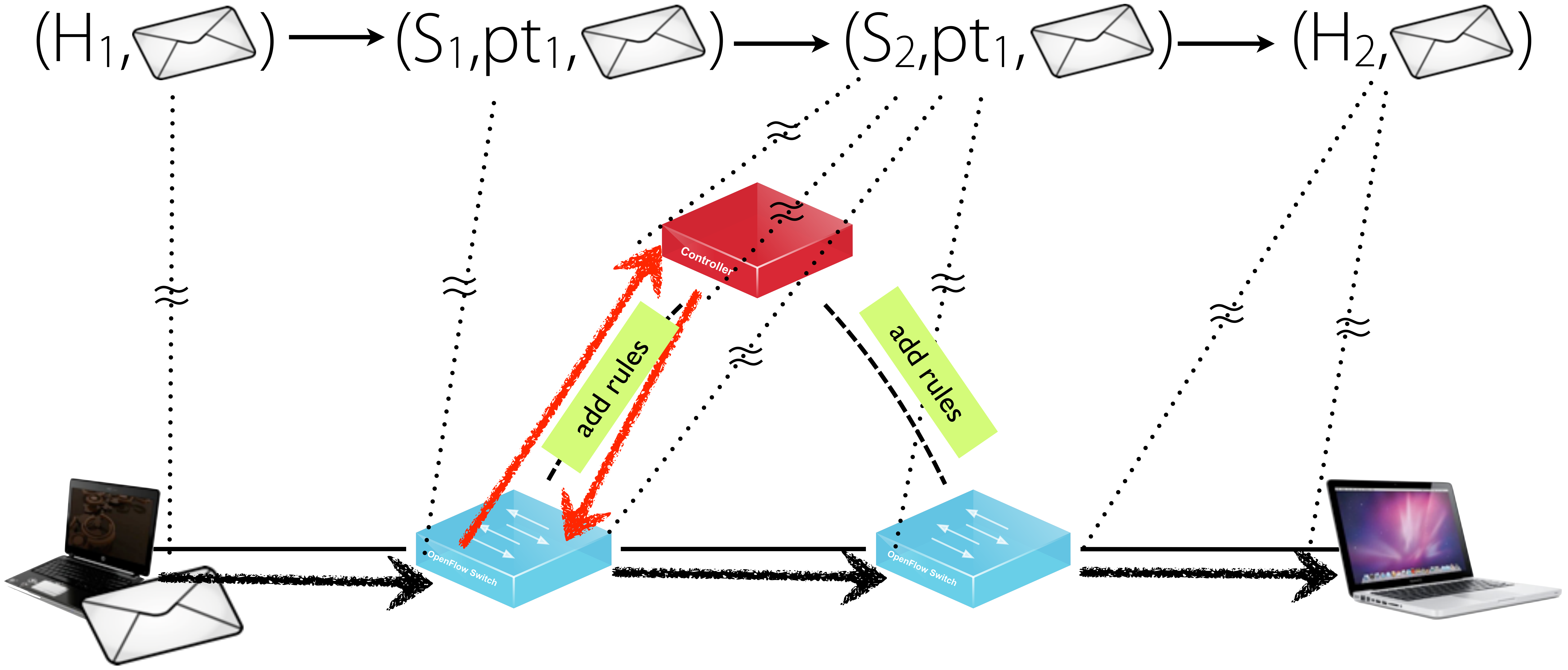
Weak Bisimulation

$(H_1,$  $) \longrightarrow (S_1, pt_1,$  $) \longrightarrow (S_2, pt_1,$  $) \longrightarrow (H_2,$  $)$

Weak Bisimulation



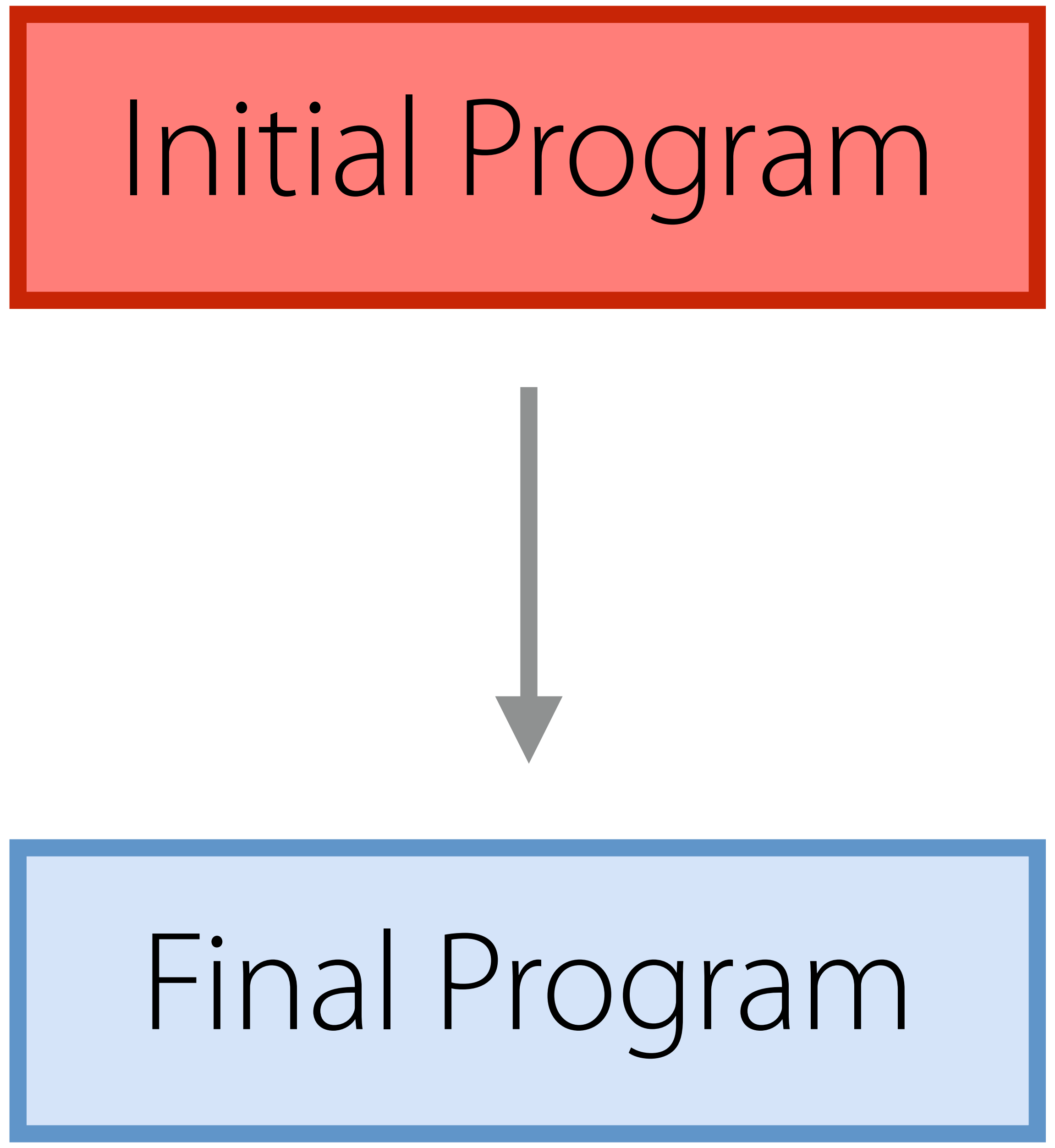
Weak Bisimulation

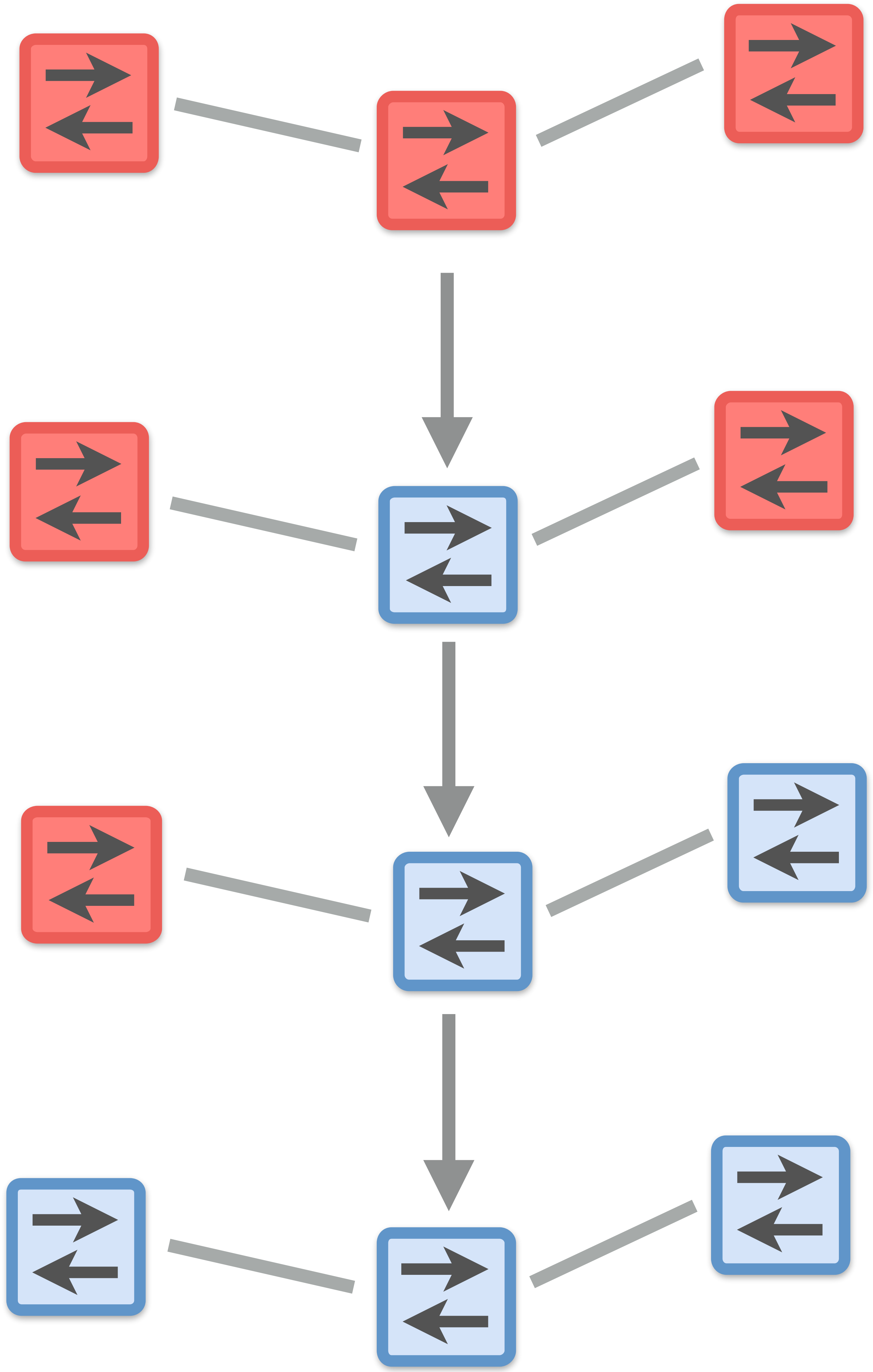


Theorem: NetKAT semantics is weakly bisimilar to Featherweight OpenFlow + run-time system

Network Updates

Question: how can we gracefully transition the network from one program to another?





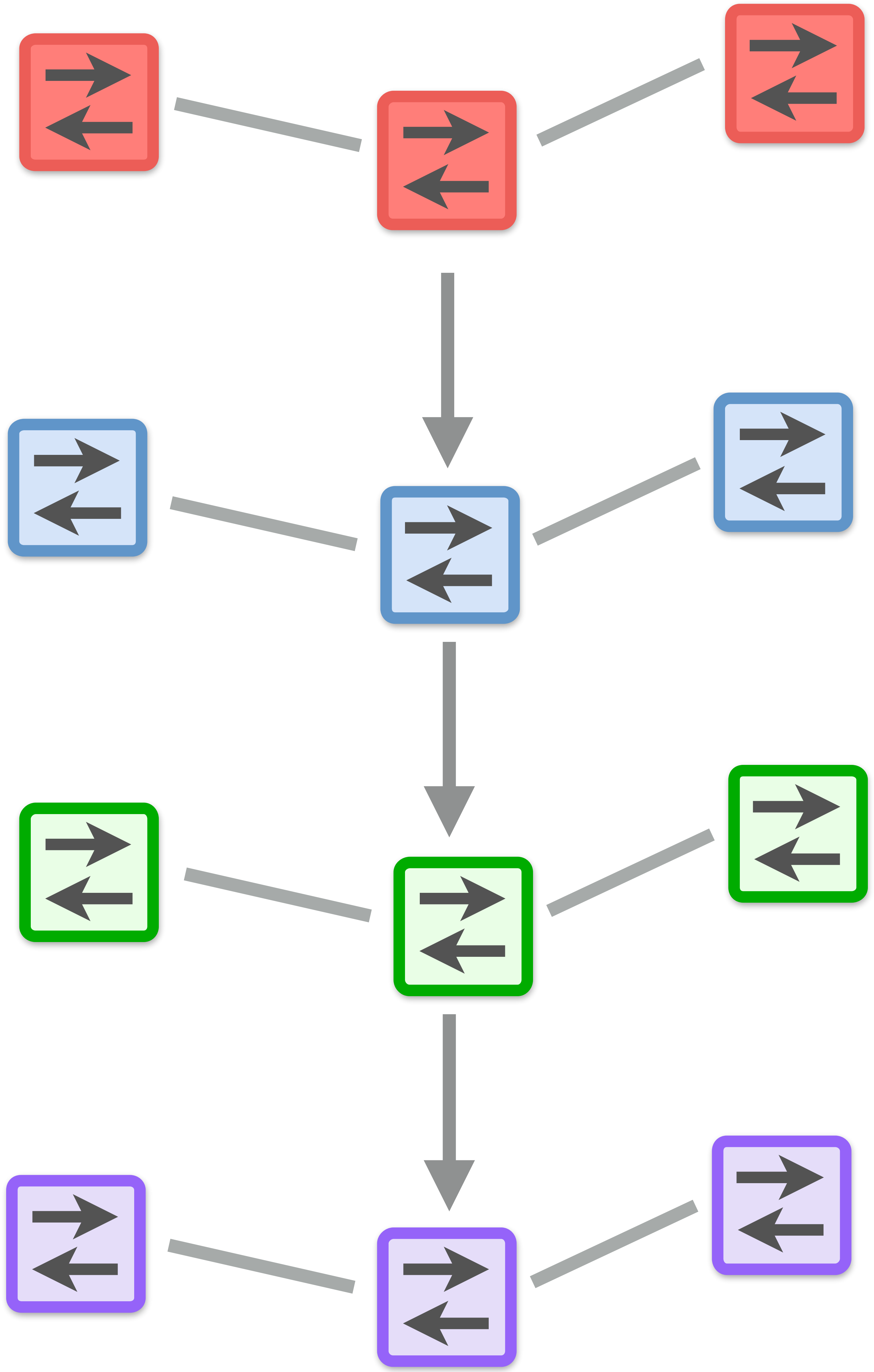
Consistent Updates

[SIGCOMM '12]

Operationally: every packet (or flow) processed using a consistent version of the network-wide configuration

Semantically: guarantee preserves all safety properties

Implementations: many different possibilities–e.g., one option is to use a two-phase distributed protocol

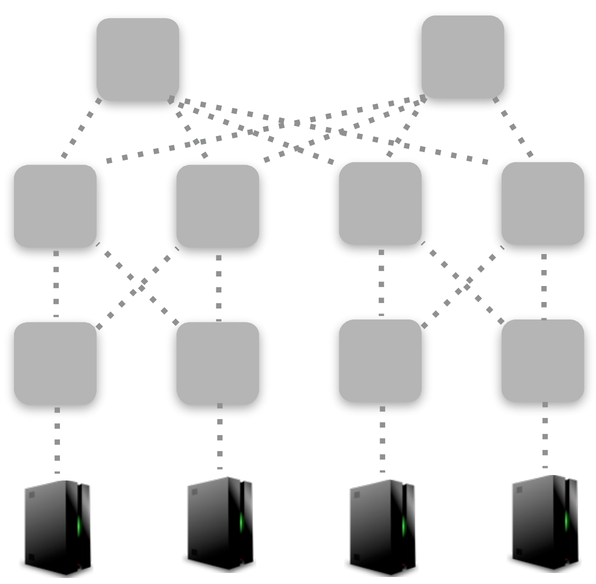


Update Synthesis

[PLDI '15]

ϕ

Logical property

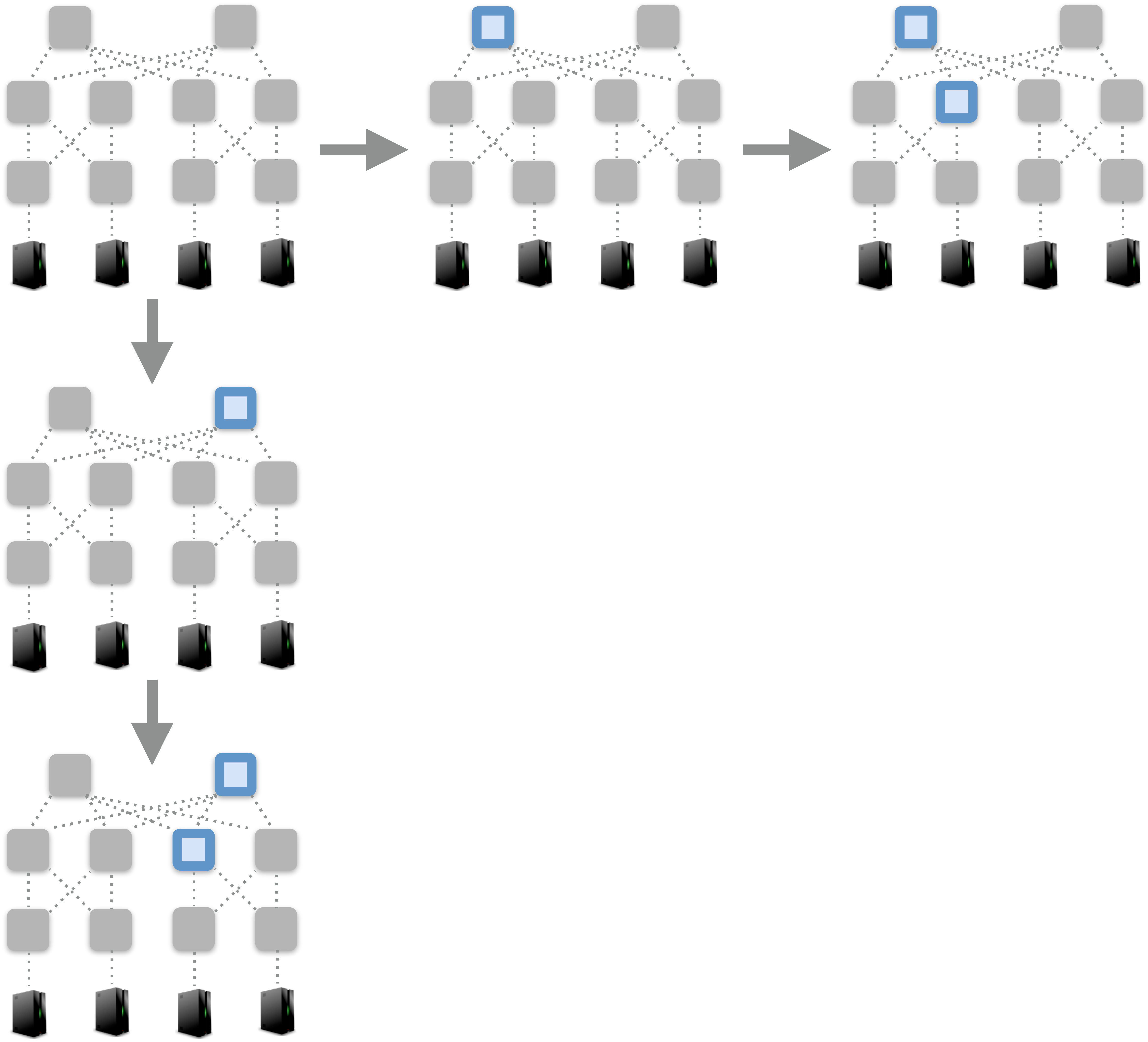


topology + configurations

Update Synthesis

[PLDI '15]

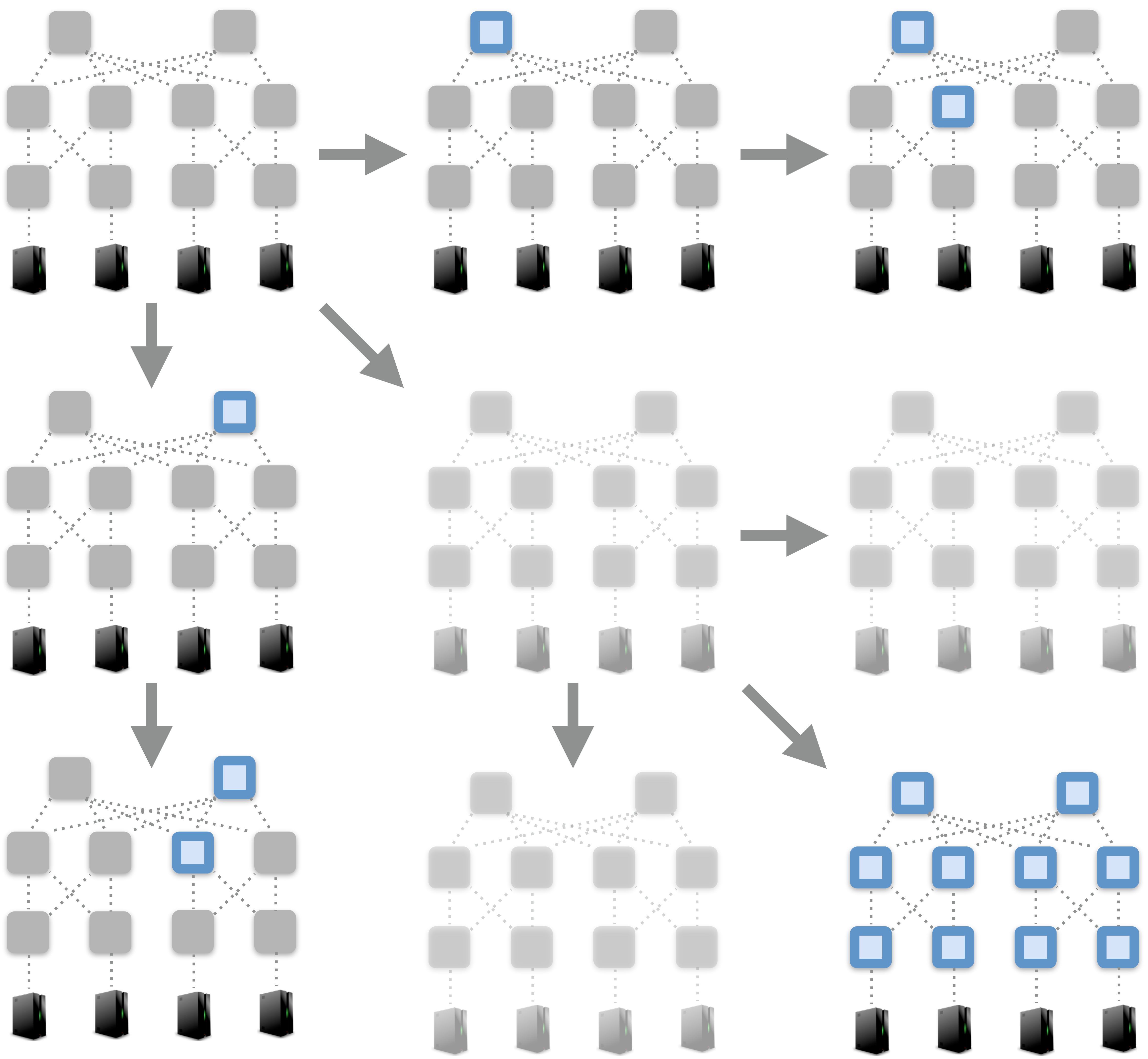
ϕ



Update Synthesis

[PLDI '15]

Φ



Conclusion

- Programming languages and formal methods have a key role to play in next-generation networking platforms
- The NetKAT language offers expressive constructs for specifying and verifying network functionality
- Formal methods are ready for prime time!

Ongoing Work

- Probabilistic semantics
- Stateful functions
- Multi-packet properties

Thank you!

- Carolyn Anderson (UMass)
- Pavol Cerny (Colorado)
- Arjun Guha (UMass)
- Jean-Baptiste Jeannin (CMU)
- Dexter Kozen (Cornell)
- Jedidiah McClurg (Colorado)
- Matthew Milano (Cornell)
- Mark Reitblatt (Cornell)
- Jennifer Rexford (Princeton)
- Cole Schlesinger (Princeton)
- Alexandra Silva (Nijmegen/UCL)
- Steffen Smolka (Cornell)
- Laure Thompson (Cornell)
- Dave Walker (Princeton)

frenetic

http://frenetic-lang.org/