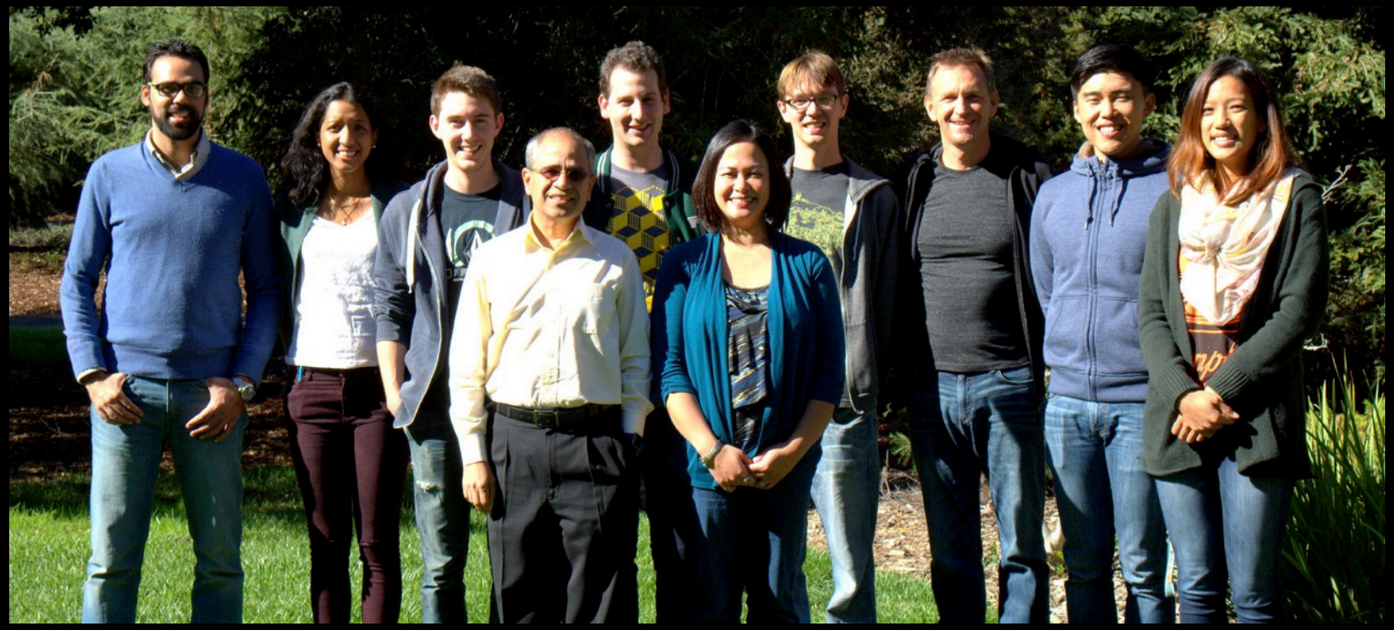
# Welcome!

## 2nd P4 Workshop, Nov 2015

Nick McKeown



# Public Domain Statement

http://www.stanford.edu/~nickm/public_domain.html

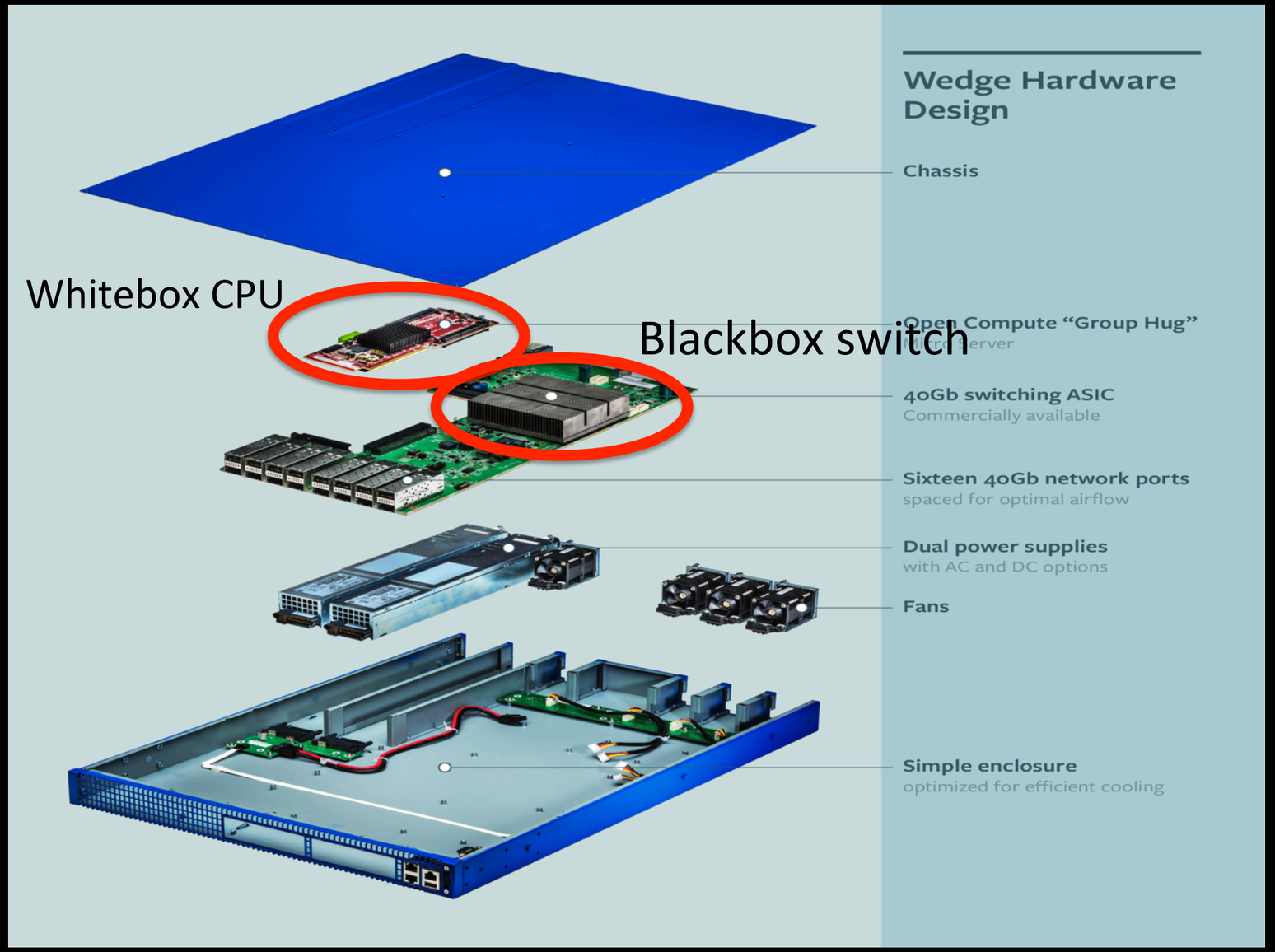
I place all of my research work in the public domain. I believe university research should serve society at large, and should be freely available for anyone to use and learn from. I have nothing against patents in principle: Industry invests huge amounts of time and money in R&D and needs to protect new ideas to give them the confidence to continue to invest. As academic researchers, I favor staying ahead by running faster, rather than protecting our backs. It keeps us on our toes and makes us more innovative. And it makes it much easier to work with our colleagues in industry.

I am very lucky that my employer gives me the discretion - as a Principal Investigator - to decide which of my ideas are placed in the public domain. Since 1999, I have placed all my research in the public domain.

# Conventional wisdom

*“Programmable forwarding planes run 10-100 times slower than fixed forwarding planes.”*

# Wedge


Wedge Hardware Design
Chassis
Whitebox CPU
Blackbox switch
Open Compute "Group Hug"
40Gb switching ASIC
Commercially available
Sixteen 40Gb network ports
spaced for optimal airflow
Dual power supplies
with AC and DC options
Fans
Simple enclosure
optimized for efficient cooling

# Why does my whitebox switch have a blackbox inside?

# 2011: Why can’t we do the same in networking?

The “match + action pipeline” was emerging

1. Suggests a good model for data parallelism
2. Suggests a set of basic *plumbing primitives*

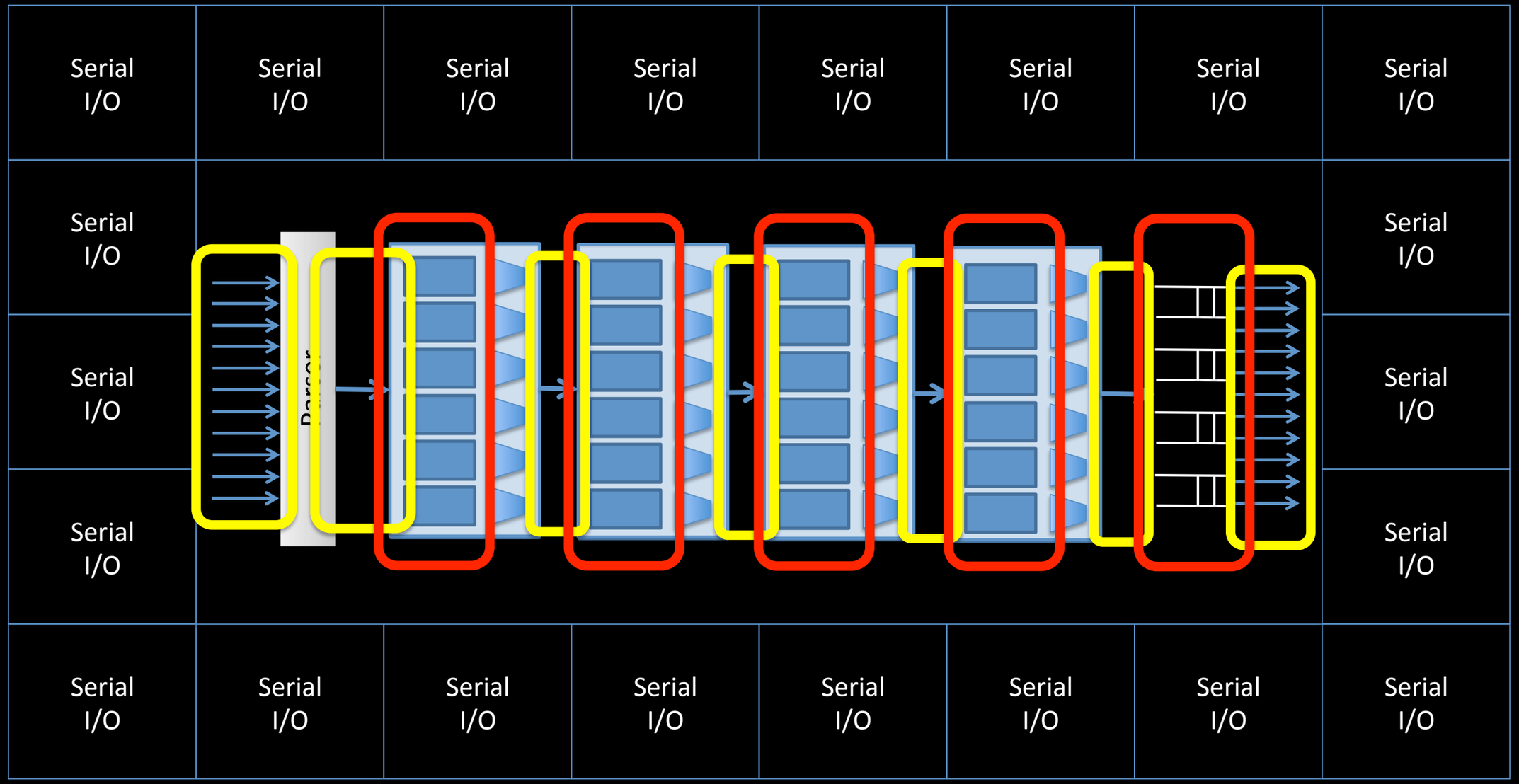
*We would need a domain-specific chip*

*We would need a domain-specific language, compilers and tool chains*

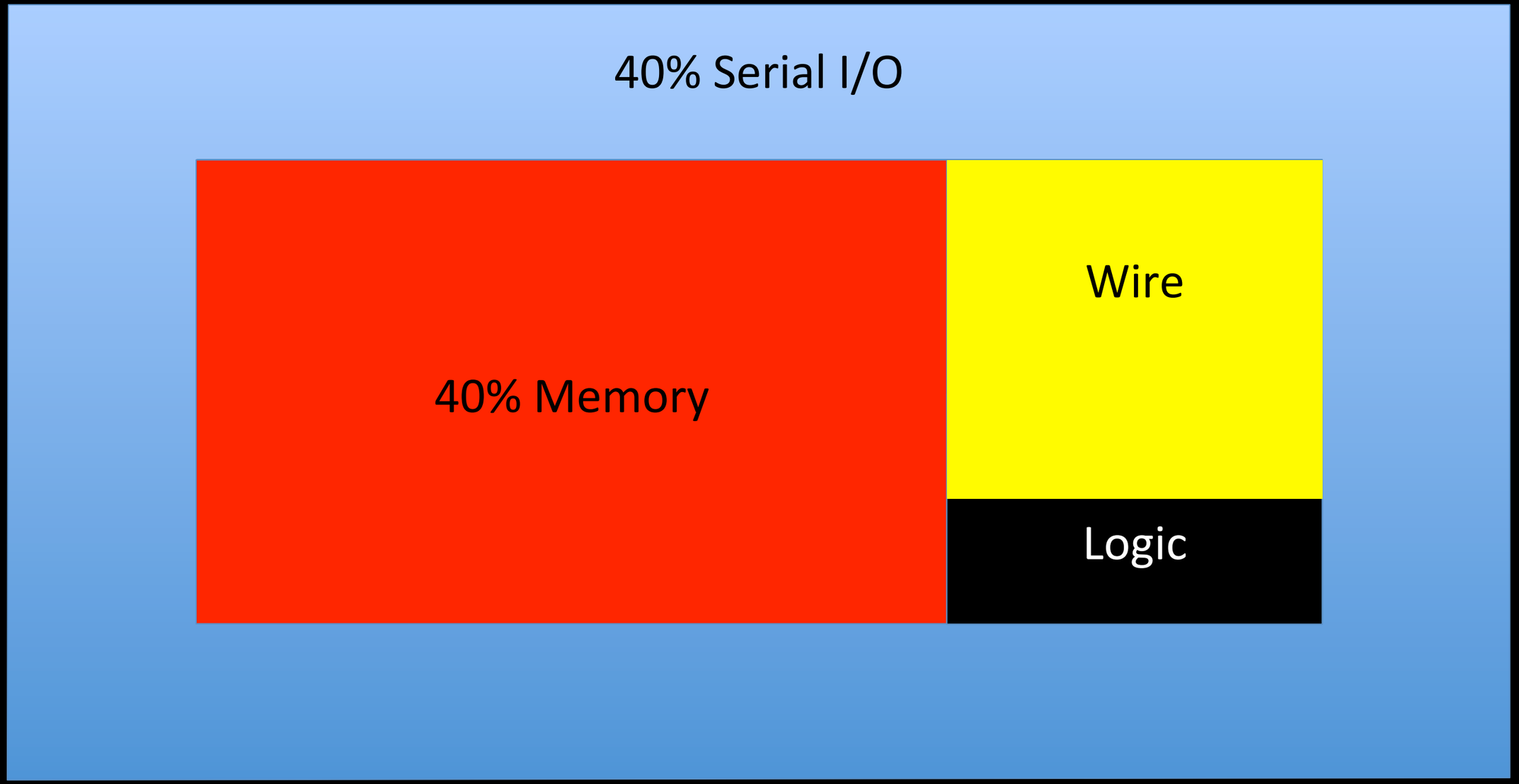
# First attempt: Stanford + Texas Instruments
[Sigcomm 2013]

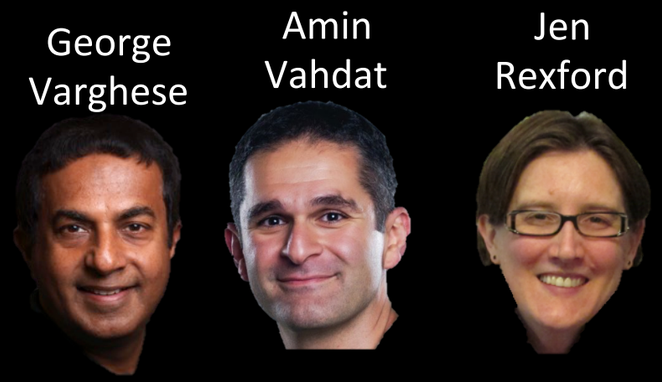
- Protocol independent switch chip
- Small primitive action set – about 6 primitives
- Huge parallelism possible
- 10-15% chip area and power penalty

# Switching Chip Area



# Switching Chip Area





George Varghese



Amin Vahdat



Jen Rexford

Dan Daly

Pat Bosshart

Glen Gibb

Martin Izzard

Dave Walker

Cole Schlesinger

Dan Talayco

Nick McKeown

ACM Sigcomm Computer Communications Review

July 2014

# P4: Programming Protocol-Independent Packet Processors

Pat Bosshart†, Dan Daly*, Glen Gibb†, Martin Izzard‡, Nick McKeown‡, Jennifer Rexford**, Cole Schlesinger**, Dan Talayco†, Amin Vahdat¶, George Varghese§, David Walker**
†Barefoot Networks *Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research

**ABSTRACT**

P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able to change the way switches process packets once they are deployed. (2) Protocol independence: Switches should not be tied to any specific network protocols. (3) Target independence: Programmers should be able to describe packet-processing functionality independently of the specifics of the underlying hardware. As an example, we describe how to use P4 to configure a switch to add a new hierarchical label.

## 1. INTRODUCTION

Software-Defined Networking (SDN) gives operators programmatic control over their networks. In SDN, the control plane is physically separate from the forwarding plane, and one control plane controls multiple forwarding devices. While forwarding devices could be programmed in many ways, having a common, open, vendor-agnostic interface (like OpenFlow) enables a control plane to control forwarding devices from different hardware and software vendors.

| Version | Date | Header Fields |
|---|---|---|
| OF 1.0 | Dec 2009 | 12 fields (Ethernet, TCP/IPv4) |
| OF 1.1 | Feb 2011 | 15 fields (MPLS, inter-table metadata) |
| OF 1.2 | Dec 2011 | 36 fields (ARP, ICMP, IPv6, etc.) |
| OF 1.3 | Jun 2012 | 40 fields |
| OF 1.4 | Oct 2013 | 41 fields |

Table 1: Fields recognized by the OpenFlow standard

The OpenFlow interface started simple, with the abstraction of a single table of rules that could match packets on a dozen header fields (e.g., MAC addresses, IP addresses, protocol, TCP/UDP port numbers, etc.). Over the past five years, the specification has grown increasingly complicated (see Table 1), with many multiple stages of rule tables, to allow switches to expose more of their capabilities to the controller.

The proliferation of new header fields shows no signs of stopping. For example, data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality. Rather than repeatedly extending the OpenFlow specification, we argue that future switches should support flexible mechanisms for parsing packets and matching header fields, allowing controller applications to leverage these capabilities through a common, open interface (i.e., a new "OpenFlow 2.0" API). Such a general, extensible approach would be simpler, more elegant, and more future-proof than today's OpenFlow 1.x standard.

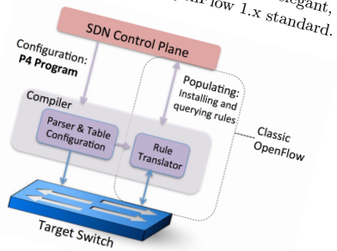



**Figure 1: P4 is a language to configure switches.**

Recent chip designs demonstrate that such flexibility can be achieved in custom ASICs at terabit speeds [1, 2, 3]. Programming this new generation of switch chips is far from easy. Each chip has its own low-level interface, akin to microcode programming. In this paper, we sketch the design of a higher-level language for Programming Protocol-independent Packet Processors (P4). [illegible] relationship between P4 [illegible] it how packets [illegible] as O

# Domain Specific Processors

Signal Processing

My codec

Compiler


DSP

Graphics

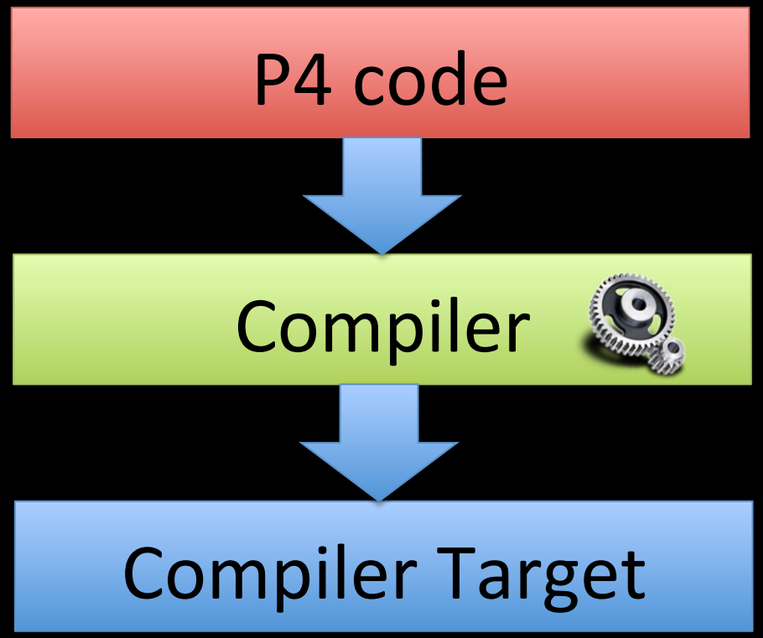
My renderer

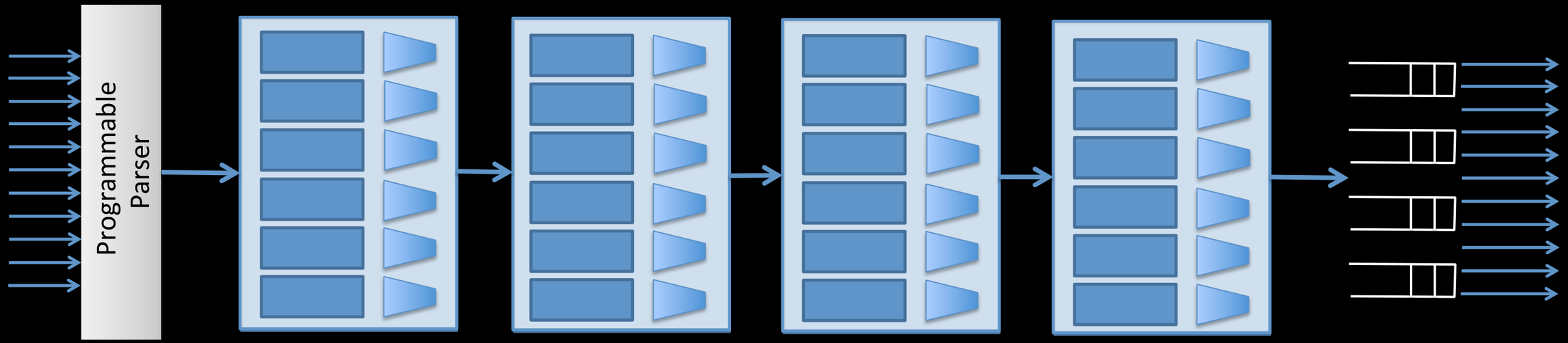
Compiler


GPU

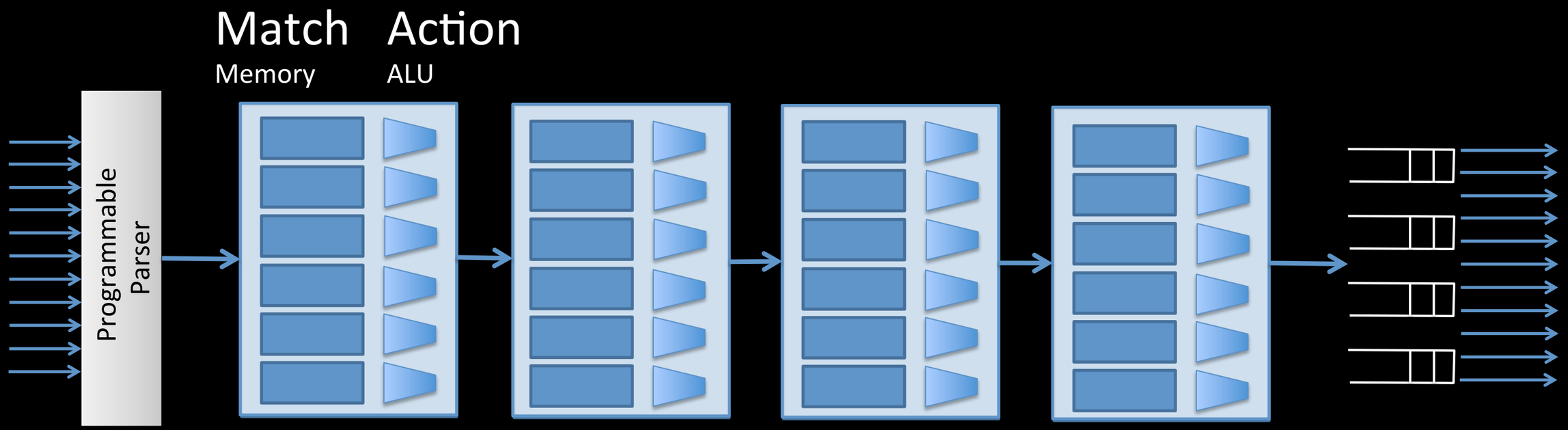
Let’s take stock of where we are today

# P4 and PISA

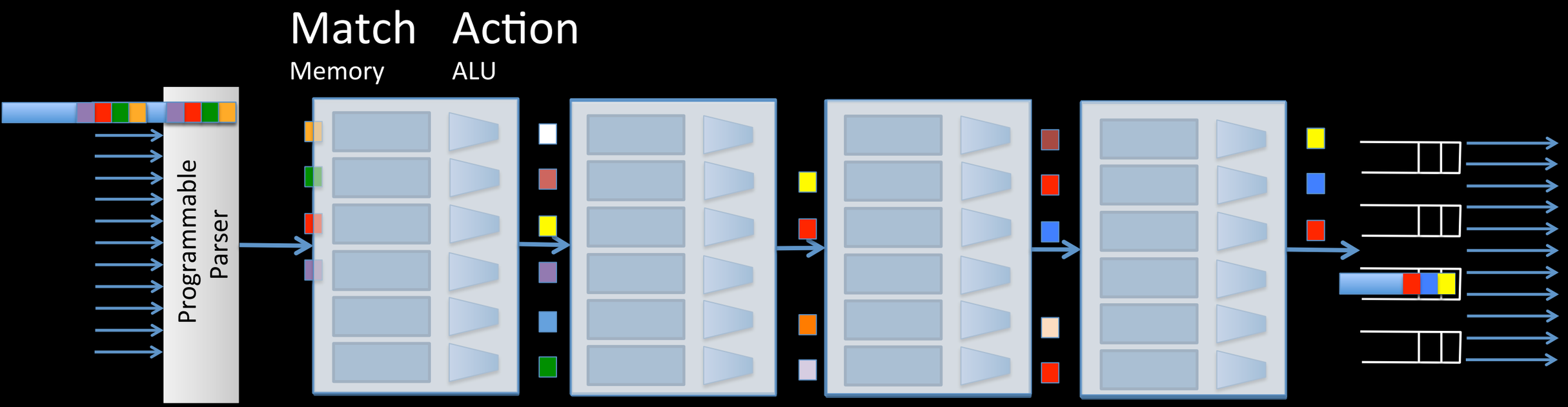


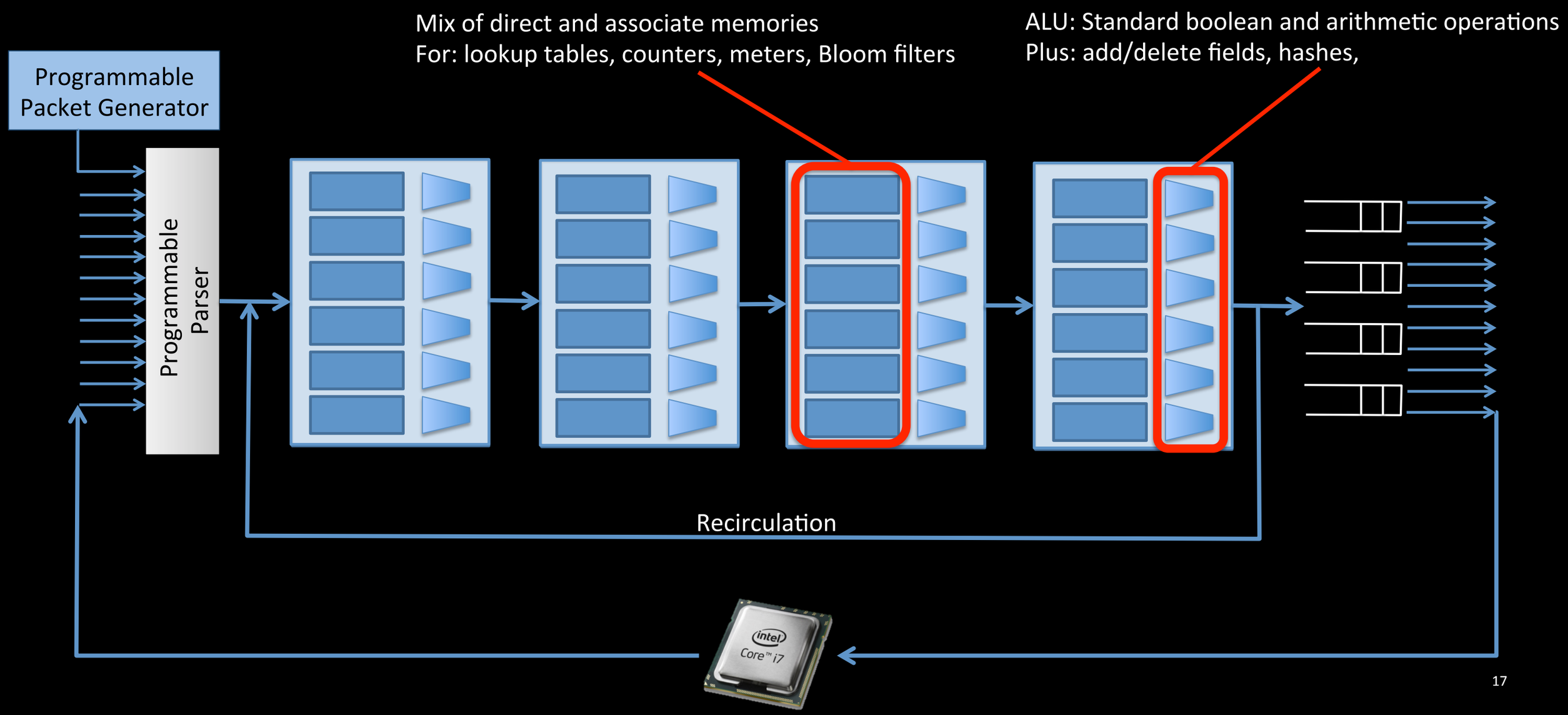
# PISA: Protocol Independent Switch Architecture



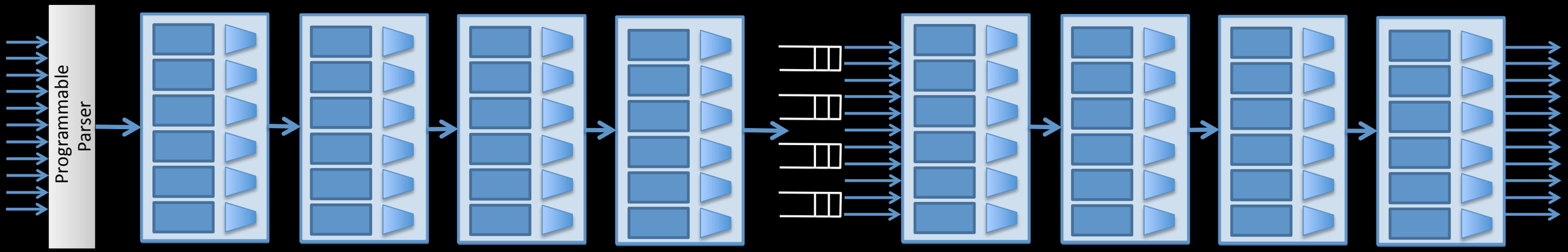
# PISA: Protocol Independent Switch Architecture



# PISA: Protocol Independent Switch Architecture



# PISA: Protocol Independent Switch Architecture



There is lots more work to do

# Work ahead of us

1. Packet scheduling: PIFO is a good start. Is it enough?
2. More stateful processing: We have counters and meters already, but we need more.
3. Conventions for generating APIs: For example, how do I make sure my API is compliant with SAI, or my existing API?
4. Composing code: A mix of modularity and good coding practices.

# Programmable Forwarding

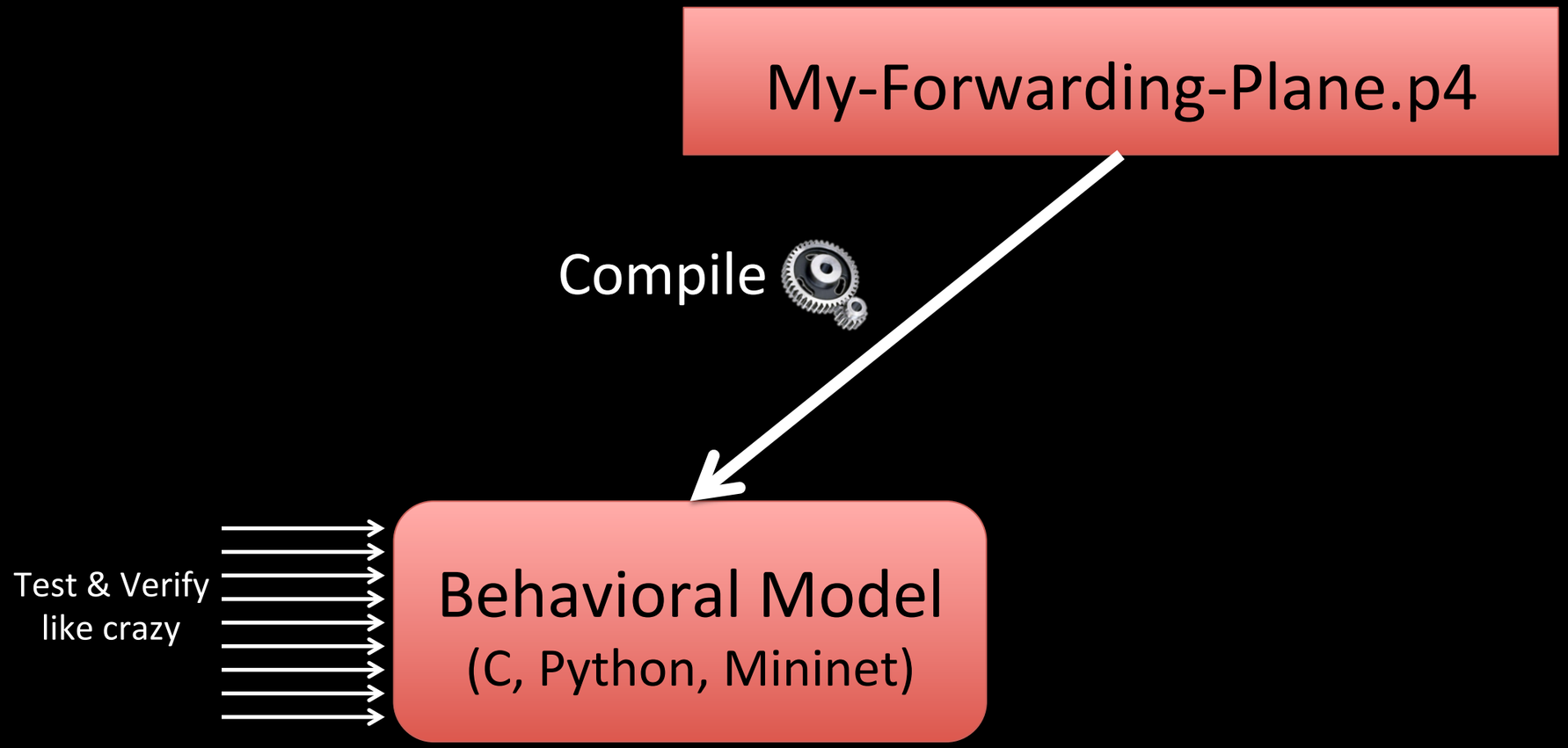
1. Add/delete features and protocols.
2. Add greater visibility into the forwarding plane.
3. Programmable in the field: Continuous updates to dataplane
4. One box, many uses: Switch, router, firewall, load-balancer, …
5. Ideas belong to programmer:
   - You no longer need to tell your secrets to switch & box vendors
   - You write and own your own code

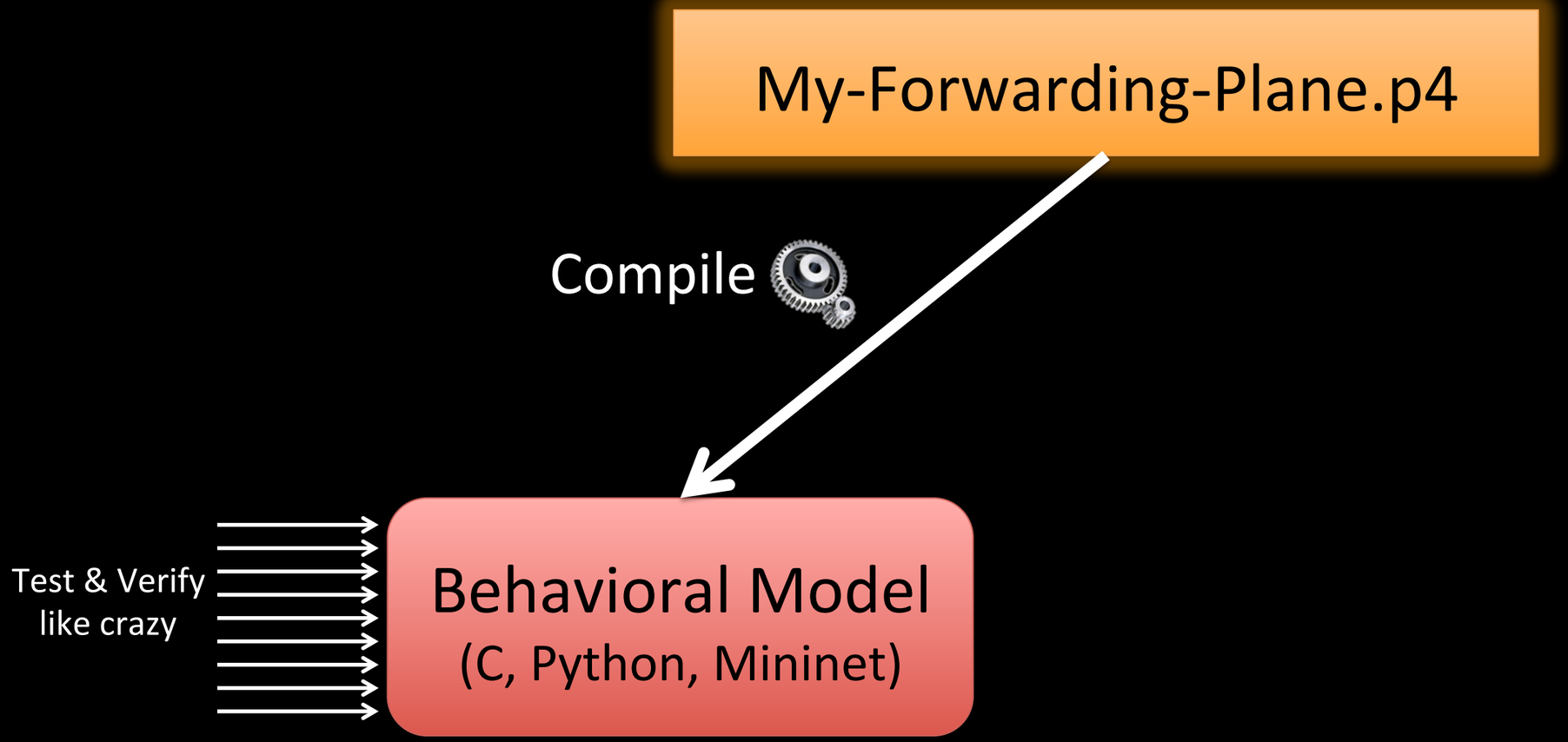
# Portability

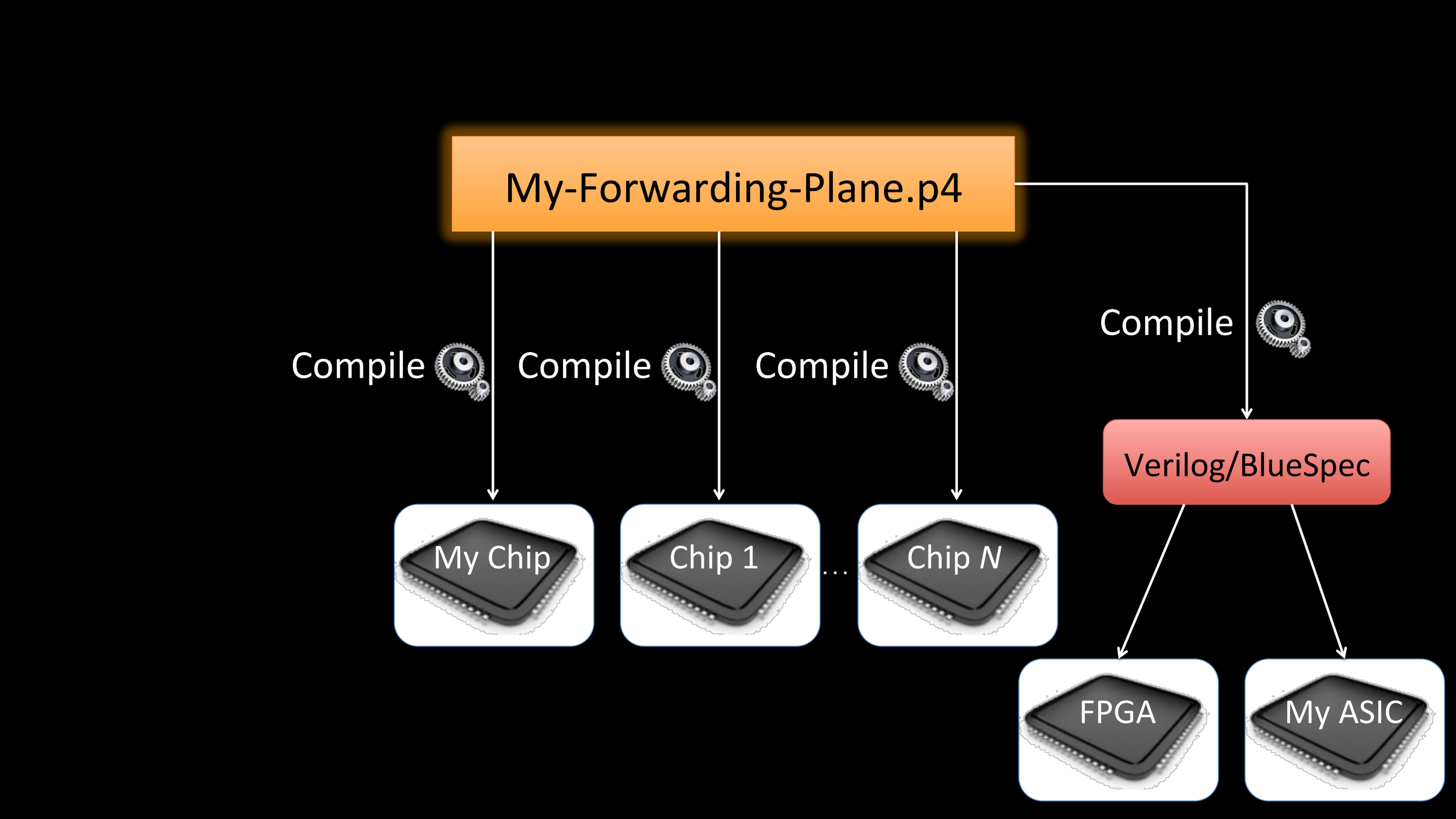
One program, many targets:

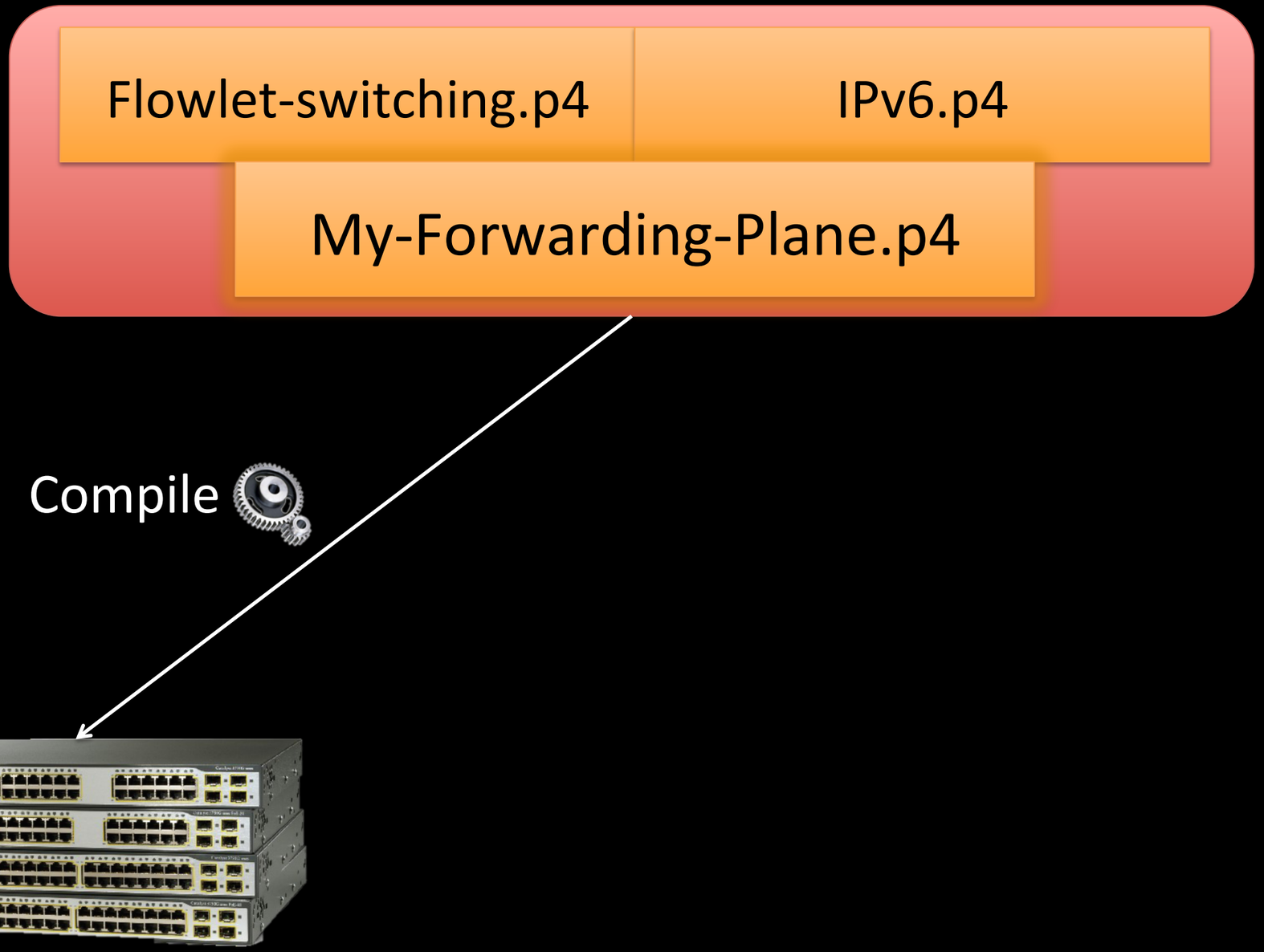
- Competition among chip vendors to be the best target, not best features
- Move seamlessly between chips
- Move seamlessly between software and hardware switches
- A consistent feature set up and down a product line
- Less time developing software for new switch chips

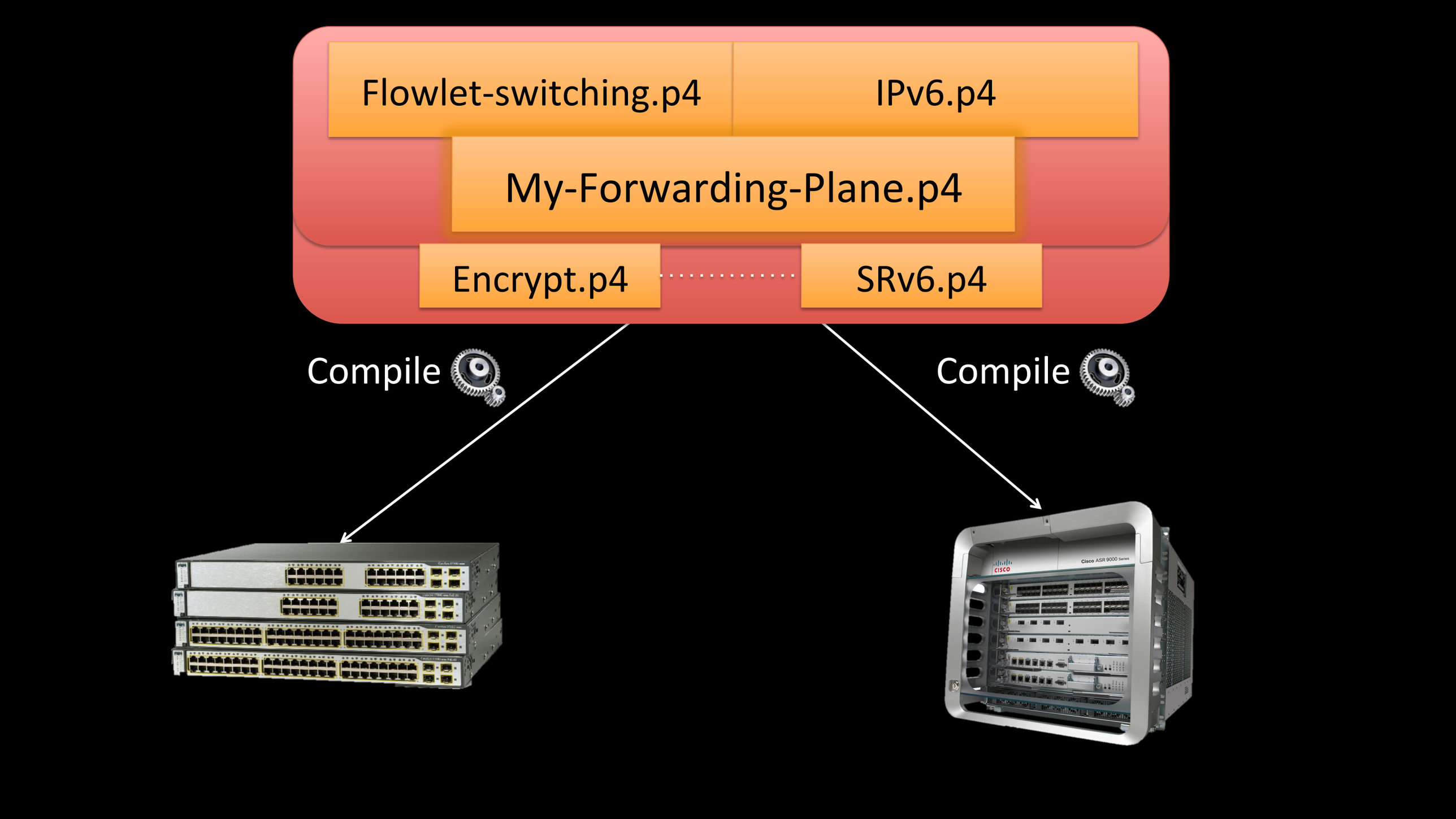
Portability is key to adoption


My-Forwarding-Plane.p4
Compile
Test & Verify
like crazy
Behavioral Model
(C, Python, Mininet)


My-Forwarding-Plane.p4
Compile
Test & Verify
like crazy
Behavioral Model
(C, Python, Mininet)


My-Forwarding-Plane.p4
Compile
Compile
Compile
Compile
Verilog/BlueSpec
My Chip
Chip 1
…
Chip N
FPGA
My ASIC


Flowlet-switching.p4
IPv6.p4
My-Forwarding-Plane.p4
Compile


Flowlet-switching.p4
IPv6.p4
My-Forwarding-Plane.p4
Encrypt.p4
SRv6.p4
Compile
Compile
cisco
Cisco ASR 9000 Series

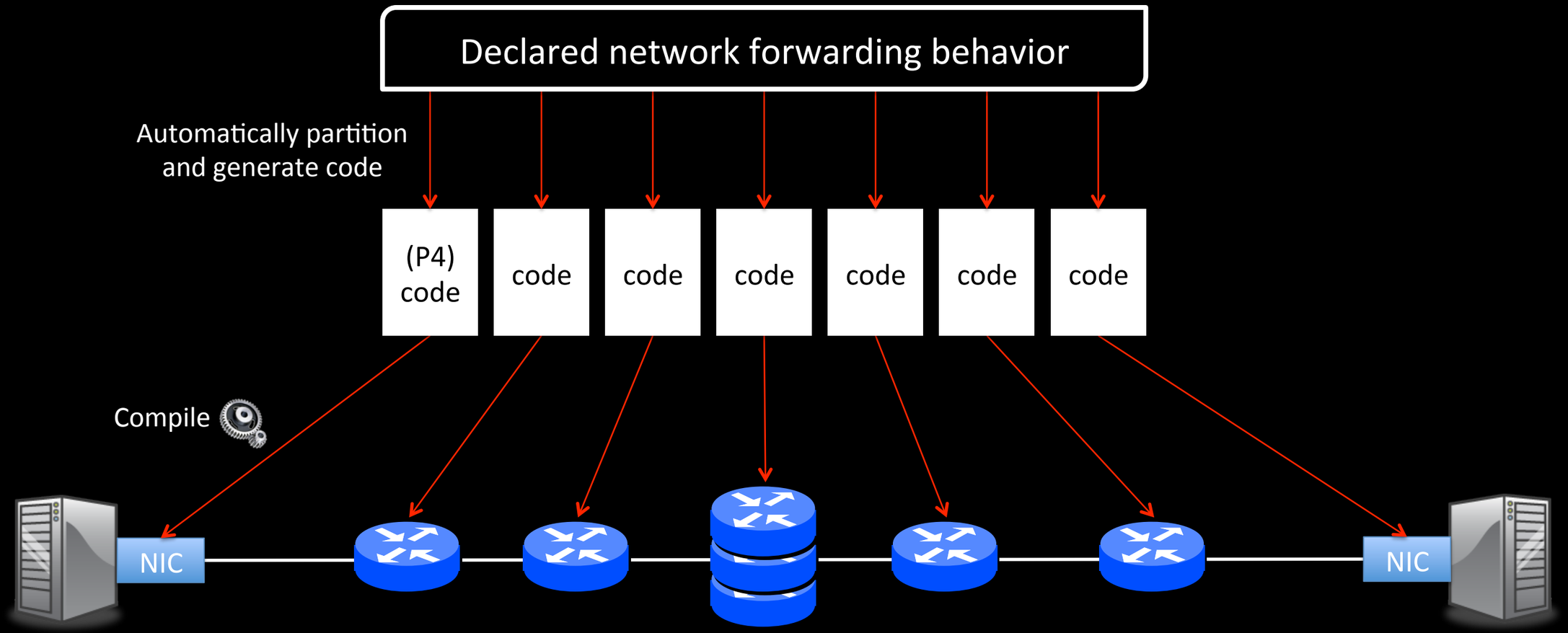
Customer-measure.p4

My-Forwarding-Plane.p4

Compile

Compile

# A long-term aspiration (still)



<The End>