Expanding the P4 universe

Gordon Brebner

Senior Fellow

Adaptive & Embedded Computing Group

Happy birthday to P4$_{16}$

- P4$_{16}$ specification version 1.1.0, May 2017

Brebner’s experience-based ‘law’ of domain-specific language evolution

	Proprietary	Standard
Purified	PX (2009 – 2014)	$P4_{16}$ (2017 – 2022)
Ad hoc	G (2006 – 2009)	$P4_{14}$ (2014 – 2017)

3-year itch

5-year itch

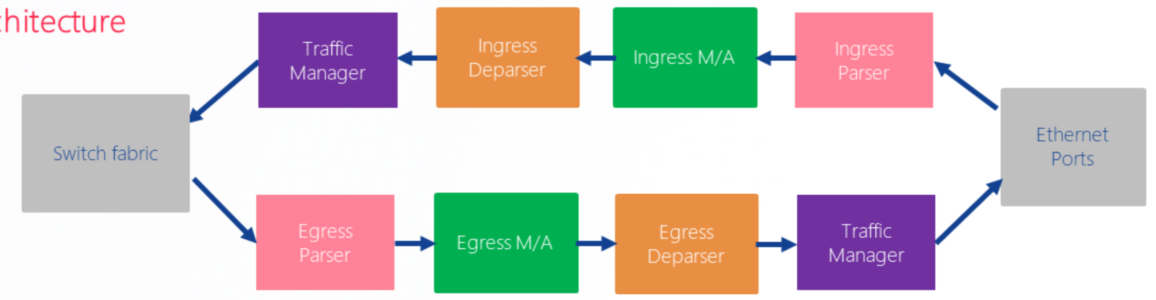
3-year itch

5-year itch ?

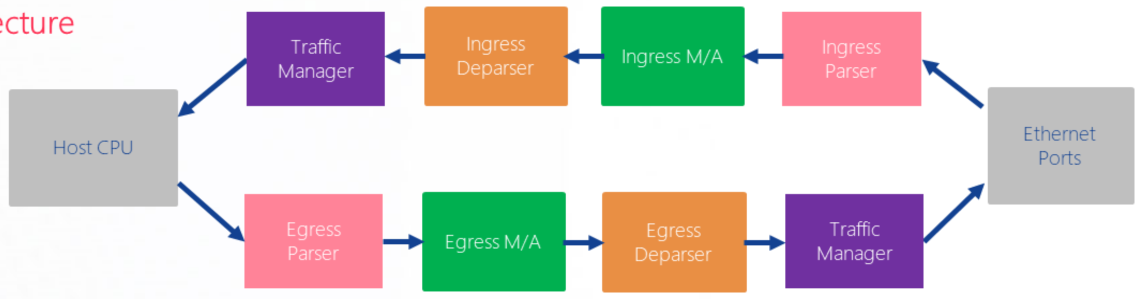
?

Switches and NICs: from a P4 viewpoint

- Switch-style architecture



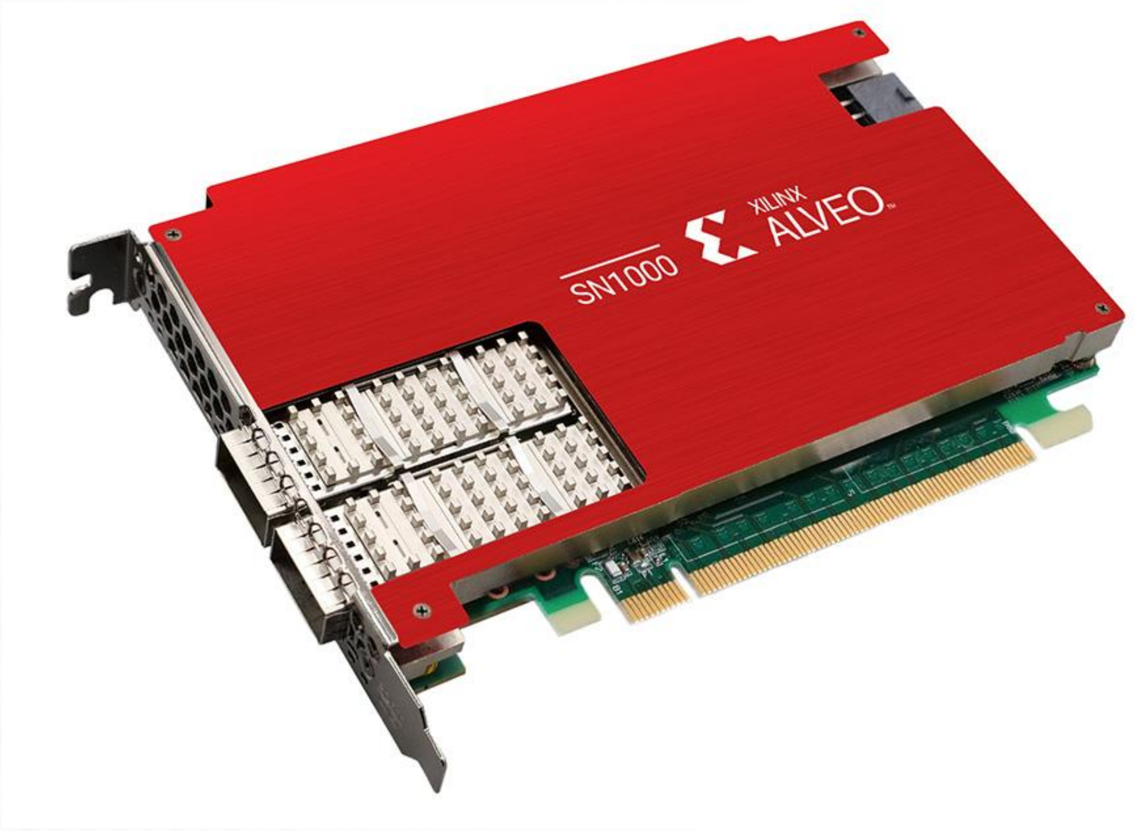
- NIC-style architecture



- “NIC”: Choose your preferred name/flavor
  - SmartNIC
  - DSC
  - IPU
  - DPU
  - SmartNAC
  - NAS
  - …
- Key thing for P4: ***Network-Attached Hub***

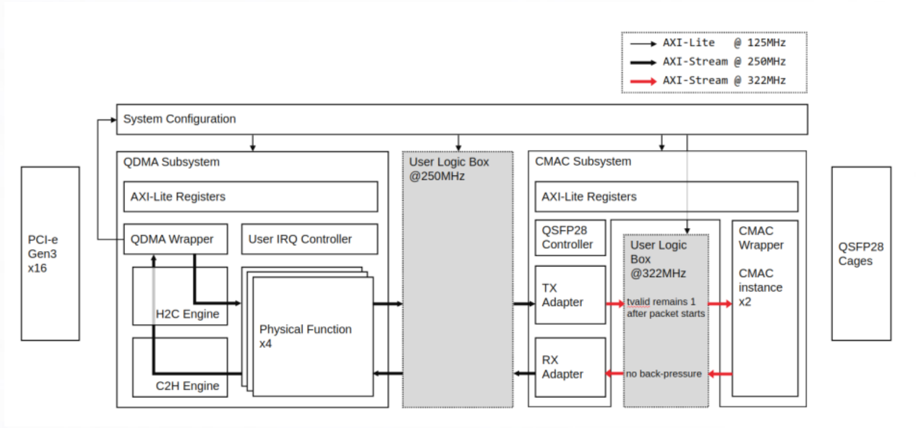
All look rather similar in P4 world: it’s the programmable data plane that matters

AMD and P4: SNxxxx SmartNIC product family



- Two-port SmartNIC: SN1022 is 2 x 100G
- Line-rate shrink-wrapped data plane implemented on FPGA
  - Standard protocol processing
  - Encapsulation / decapsulation
  - OVS offload
- **FPGA programmed by AMD-Xilinx using P4**
- Data paths have multiple points for user-developed plug-ins
  - Open-ended extensions
- **FPGA programmed by user using P4**

AMD and P4: Open-source OpenNIC project



Community is porting OpenNIC to increasing number of Alveo platforms

- https://github.com/Xilinx/open-nic
- Bare-metal NIC for networking researchers
  - One- or two-port 100G configurations
  - Design your own line-rate processing
  - Standard network and DPDK drivers
- **Data paths can be implemented in P4**
  - Using P4-to-FPGA compilation flow
- Growing international community
  - Piloted with ~20 research groups

Democratization of P4

- Original switch focus
  - Not many P4-programmable switches
  - Not many people allowed to program switches
  - Switches are not (re-)programmed very often
  - Hence research tends to be more theoretical

- New NIC focus
  - Multiple examples of P4-programmable NICs
  - Wider access to systems housing NICs
  - NICs can be frequently reprogrammed
  - Blossoming of P4-related research anticipated

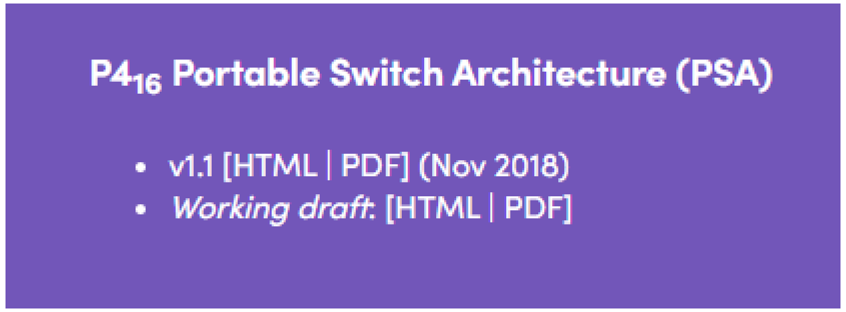
- **This could be the ‘five-year refresh’ for P4**

Expanding P4 coverage

- Switch-oriented functional coverage
  - Basic switching functions – PISA model
  - Notable unprogrammable component: traffic manager
  - In-switch compute examples often involve ‘P4 abuse’
- NIC-oriented functional opportunities and needs are greater
  - Still have standard packet processing functions
  - Customized transport and higher functions
  - Termination (to host, storage) functions
  - Other offloaded infrastructural functions
  - Network-attached compute (inc. ML)
- **How far might the domain of domain-specific P4 be broadened?**

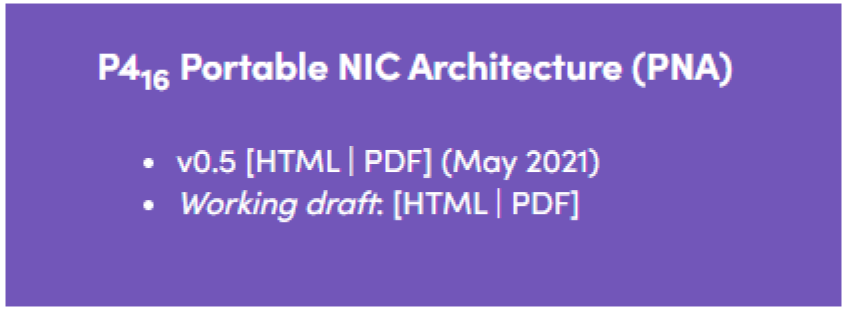
P4 standard architectures: good, bad, or irrelevant?

- The key innovation in $P4_{16}$ was language-architecture separation
  - Something not the case in $P4_{14}$
  - This notion has stood the test of time
- Associated with this was the idea of standard architectures
  - To encourage portability of P4 code
- The notion of externs, and standard libraries, has stood the test of time
- But the notion of standard architecture models maybe has not
  - Portable Switch Architecture (PSA): more-or-less PISA revisited
  - Portable NIC Architecture (PNA): less clear what's portable
- Might stifle innovation?



$P4_{16}$ Portable Switch Architecture (PSA)

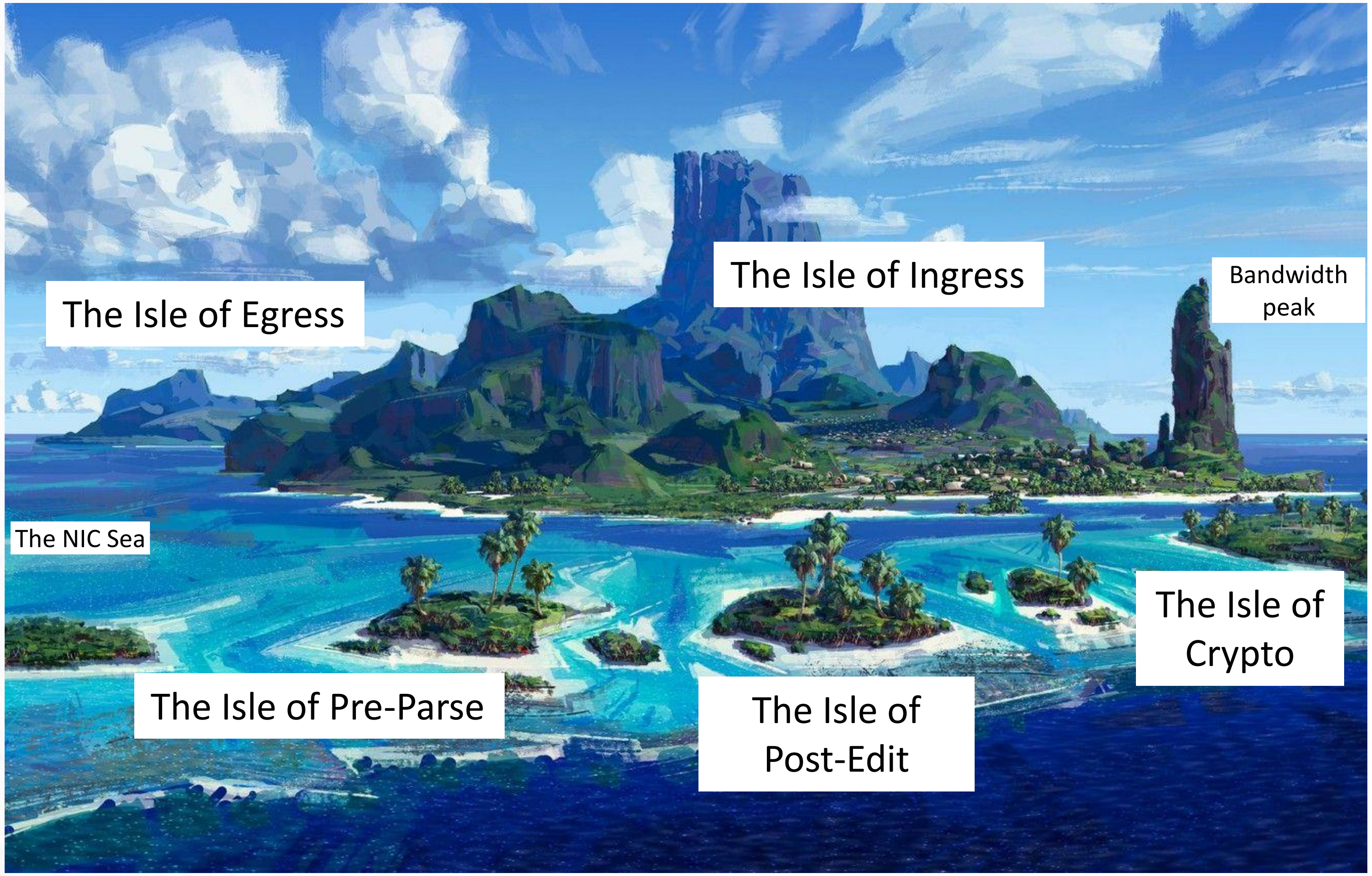
- v1.1 [HTML | PDF] (Nov 2018)
- *Working draft:* [HTML | PDF]



$P4_{16}$ Portable NIC Architecture (PNA)

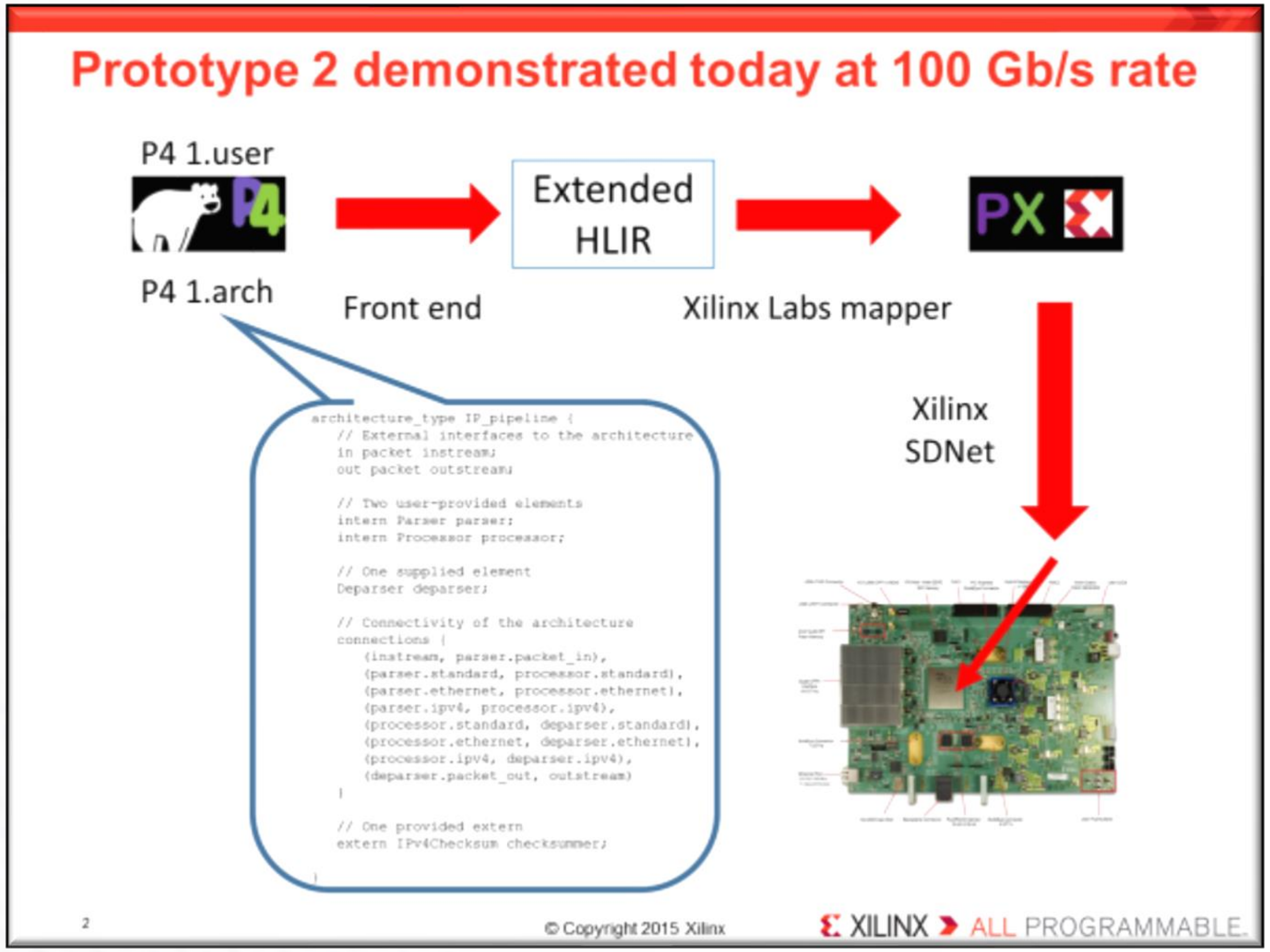
- v0.5 [HTML | PDF] (May 2021)
- *Working draft:* [HTML | PDF]

P4 archipelagos: collections of P4 islands



- P4 subsystems can be portable
  - Designed using modules
  - Include standard externs
- Used and re-used as components in diverse systems
- Example:
  - AMD SN1000 SmartNIC data path built from nine separate P4 components
  - Then user plug-ins are more separate P4 components
- Moving P4 onwards from a standard-ish switch era to a diversified NIC era

Expanding coverage: Programable Target Architecture (P4 Workshop, Nov 2015)



Prototype 2 demonstrated today at 100 Gb/s rate

P4 1.user
Extended HLIR
P4 1.arch
Front end
Xilinx Labs mapper
PX
Xilinx SDNet

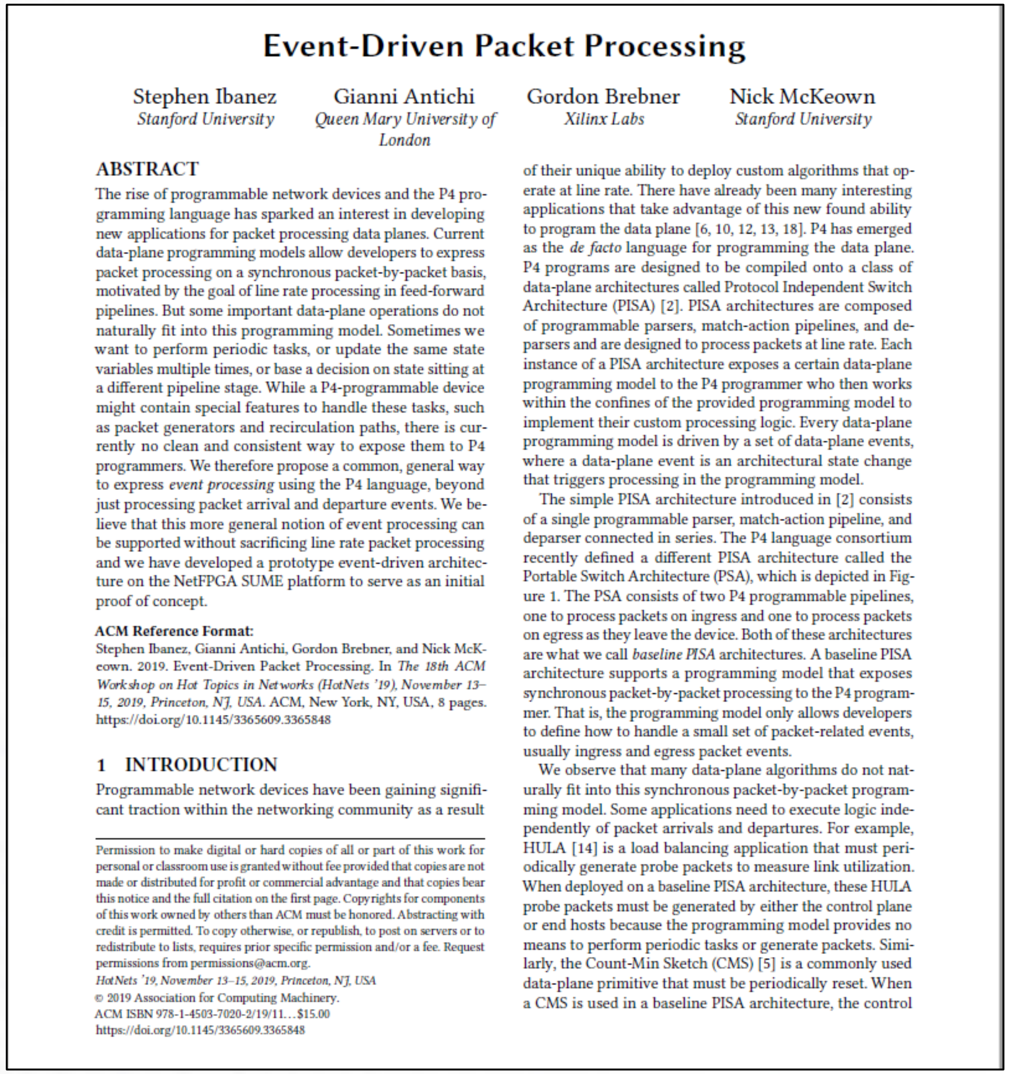
```
architecture_type IP_pipeline {
   // External interfaces to the architecture
   in packet instream;
   out packet outstream;

   // Two user-provided elements
   intern Parser parser;
   intern Processor processor;

   // One supplied element
   Deparser deparser;

   // Connectivity of the architecture
   connections {
      (instream, parser.packet_in),
      (parser.standard, processor.standard),
      (parser.ethernet, processor.ethernet),
      (parser.ipv4, processor.ipv4),
      (processor.standard, deparser.standard),
      (processor.ethernet, deparser.ethernet),
      (processor.ipv4, deparser.ipv4),
      (deparser.packet_out, outstream)
   }

   // One provided extern
   extern IPv4Checksum checksummer;

}
```

2 © Copyright 2015 Xilinx XILINX ALL PROGRAMMABLE.

Slide from talk at P4 Workshop, November 2015

- This was when the language/architecture notion was just crystalizing
- Describing complete systems with P4 extensions
  - P4 components
  - Other non-P4 components
  - Connectivity between components
- Inspired by Click
- Implemented using FPGA
  - Not so popular for fixed-component targets
- Meant also to be vehicle for formal descriptions of portable standard architectures, like PSA
- **New thought: could describe a P4 archipelago**

Expanding coverage: Generalized event-driven processing model



Event-Driven Packet Processing

Stephen Ibanez, *Stanford University*
Gianni Antichi, *Queen Mary University of London*
Gordon Brebner, *Xilinx Labs*
Nick McKeown, *Stanford University*

ABSTRACT

The rise of programmable network devices and the P4 programming language has sparked an interest in developing new applications for packet processing data planes. Current data-plane programming models allow developers to express packet processing on a synchronous packet-by-packet basis, motivated by the goal of line rate processing in feed-forward pipelines. But some important data-plane operations do not naturally fit into this programming model. Sometimes we want to perform periodic tasks, or update the same state variables multiple times, or base a decision on state sitting at a different pipeline stage. While a P4-programmable device might contain special features to handle these tasks, such as packet generators and recirculation paths, there is currently no clean and consistent way to expose them to P4 programmers. We therefore propose a common, general way to express *event processing* using the P4 language, beyond just processing packet arrival and departure events. We believe that this more general notion of event processing can be supported without sacrificing line rate packet processing and we have developed a prototype event-driven architecture on the NetFPGA SUME platform to serve as an initial proof of concept.

ACM Reference Format:
Stephen Ibanez, Gianni Antichi, Gordon Brebner, and Nick McKeown. 2019. Event-Driven Packet Processing. In *The 18th ACM Workshop on Hot Topics in Networks (HotNets '19), November 13–15, 2019, Princeton, NJ, USA*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3365609.3365848

1 INTRODUCTION

Programmable network devices have been gaining significant traction within the networking community as a result of their unique ability to deploy custom algorithms that operate at line rate. There have already been many interesting applications that take advantage of this new found ability to program the data plane [6, 10, 12, 13, 18]. P4 has emerged as the *de facto* language for programming the data plane. P4 programs are designed to be compiled onto a class of data-plane architectures called Protocol Independent Switch Architecture (PISA) [2]. PISA architectures are composed of programmable parsers, match-action pipelines, and deparsers and are designed to process packets at line rate. Each instance of a PISA architecture exposes a certain data-plane programming model to the P4 programmer who then works within the confines of the provided programming model to implement their custom processing logic. Every data-plane programming model is driven by a set of data-plane events, where a data-plane event is an architectural state change that triggers processing in the programming model.

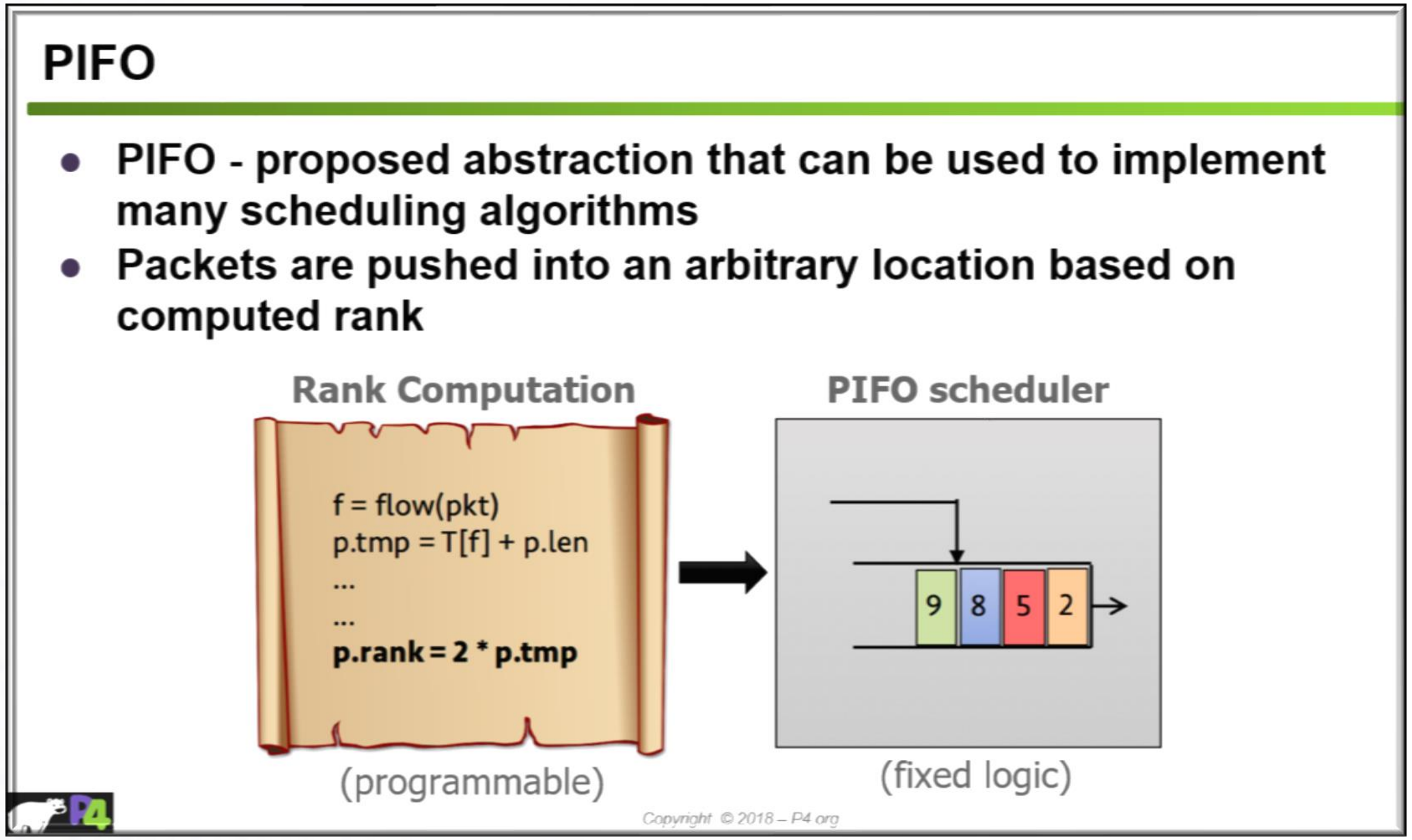
The simple PISA architecture introduced in [2] consists of a single programmable parser, match-action pipeline, and deparser connected in series. The P4 language consortium recently defined a different PISA architecture called the Portable Switch Architecture (PSA), which is depicted in Figure 1. The PSA consists of two P4 programmable pipelines, one to process packets on ingress and one to process packets on egress as they leave the device. Both of these architectures are what we call *baseline PISA* architectures. A baseline PISA architecture supports a programming model that exposes synchronous packet-by-packet processing to the P4 programmer. That is, the programming model only allows developers to define how to handle a small set of packet-related events, usually ingress and egress packet events.

We observe that many data-plane algorithms do not naturally fit into this synchronous packet-by-packet programming model. Some applications need to execute logic independently of packet arrivals and departures. For example, HULA [14] is a load balancing application that must periodically generate probe packets to measure link utilization. When deployed on a baseline PISA architecture, these HULA probe packets must be generated by either the control plane or end hosts because the programming model provides no means to perform periodic tasks or generate packets. Similarly, the Count-Min Sketch (CMS) [5] is a commonly used data-plane primitive that must be periodically reset. When a CMS is used in a baseline PISA architecture, the control

Paper presented at Hot Nets ‘19

- Standard P4 programming model involves handling only packet-related events at ingress or egress
- To broaden scope, desirable to generalize event-driven model to encompass other types of triggering events, for example:
  - Packet arrivals and departures
  - Buffer overflow and underflow
  - Link status change
  - Timer expiration
  - Control plane signals
  - User-defined events
- Example applications:
  - Traffic management
  - In-network computing
  - Congestion-aware forwarding
  - Network management and monitoring

Expanding coverage: Programmable Traffic Management



PIFO

- PIFO - proposed abstraction that can be used to implement many scheduling algorithms
- Packets are pushed into an arbitrary location based on computed rank

- Tackle the most famous 'black box' in standard P4 architectures
- Relevant in NIC context, as part of infrastructure offload for hosts with multiple tenants
- Extend P4 to include scheduling and shaping algorithms, to then enable programmable traffic management
- Experiments have compared PIFO style with selection before queue insertion and more conventional style with selection at queue removal

Based on PIFO research (Sivaraman et al., SIGCOMM 2016)
FPGA implementation with Dalleggio (NYU) and Ibanez (Stanford)

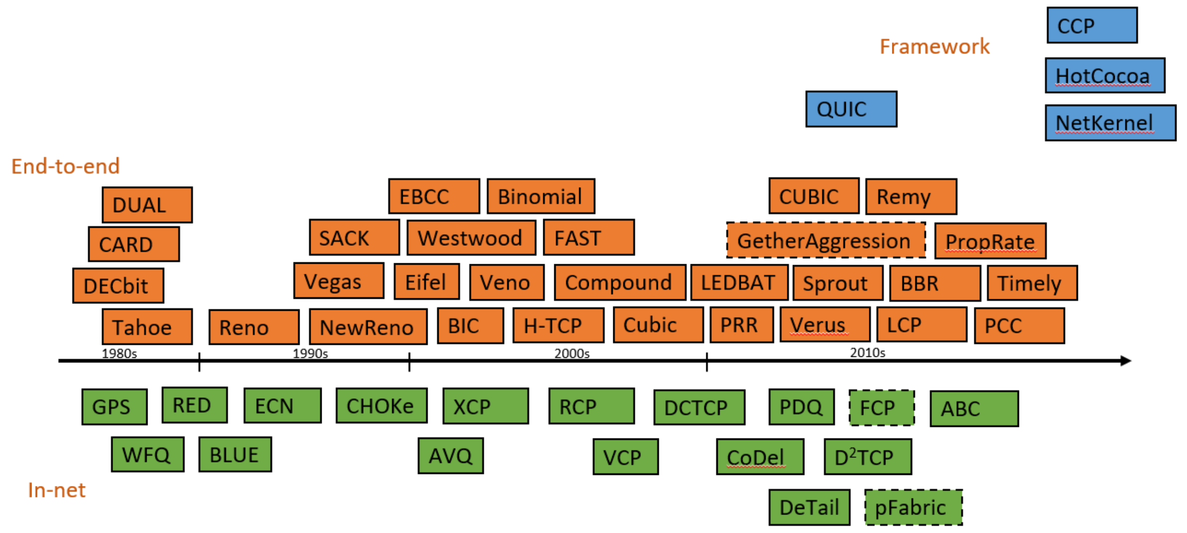
Expanding coverage: Programmable Transport Protocols

- Overall goal is to support customized transport protocols
  - Essential for end systems

- Three main P4 extension areas needed:
  - Congestion control mechanisms
  - Segmentation and reassembly
  - Statefulness

- Notable past work by Mina Arashloo et al.

- Congestion control:
  - Recent work on offload of active and passive measurement aspects to NIC hardware
  - New focus for the P4 Applications WG



Plethora of congestion control approaches:
Opportunity for extended-P4 programmability

Other expansion trends

- **NIC context for P4 has inspired various non-standard extensions, e.g.**
  - Host interface: programmable DMA
  - Data plane writeable match-action tables
  - Stateful segmentation
  - Non-trivial computational externs
  - Cryptographic operations
- **Watch out for other things presented at P4 Workshop**

Takeaways

- P4 is alive and well, and at the lift-off point for expanding its universe
- Change of emphasis from Switch to NIC (or whatever marketing term is preferred) means democratization of P4
- NIC context drives expansion of functional coverage of P4 into natural adjacencies
- NIC diversity suggests move from standard P4 architectures to custom P4 archipelagos
- **Great opportunities for everyone to put effort into developing a pure framework for P4 expansion**

AMD

DISCLAIMER AND ATTRIBUTIONS