Extending the range of P4 programmability

Gordon Brebner
Xilinx Labs, San Jose, USA

P4EU Keynote, Cambridge, UK
24 September 2018

XILINX

What this talk is about

- P4 history and status
- Portable NIC Architecture (PNA)
- Programmable Target Architecture (PTA)
- Programmable Traffic Manager (PTM)
- Towards open reference platforms

P4 history and status

P4

Programming Protocol-independent Packet Processors

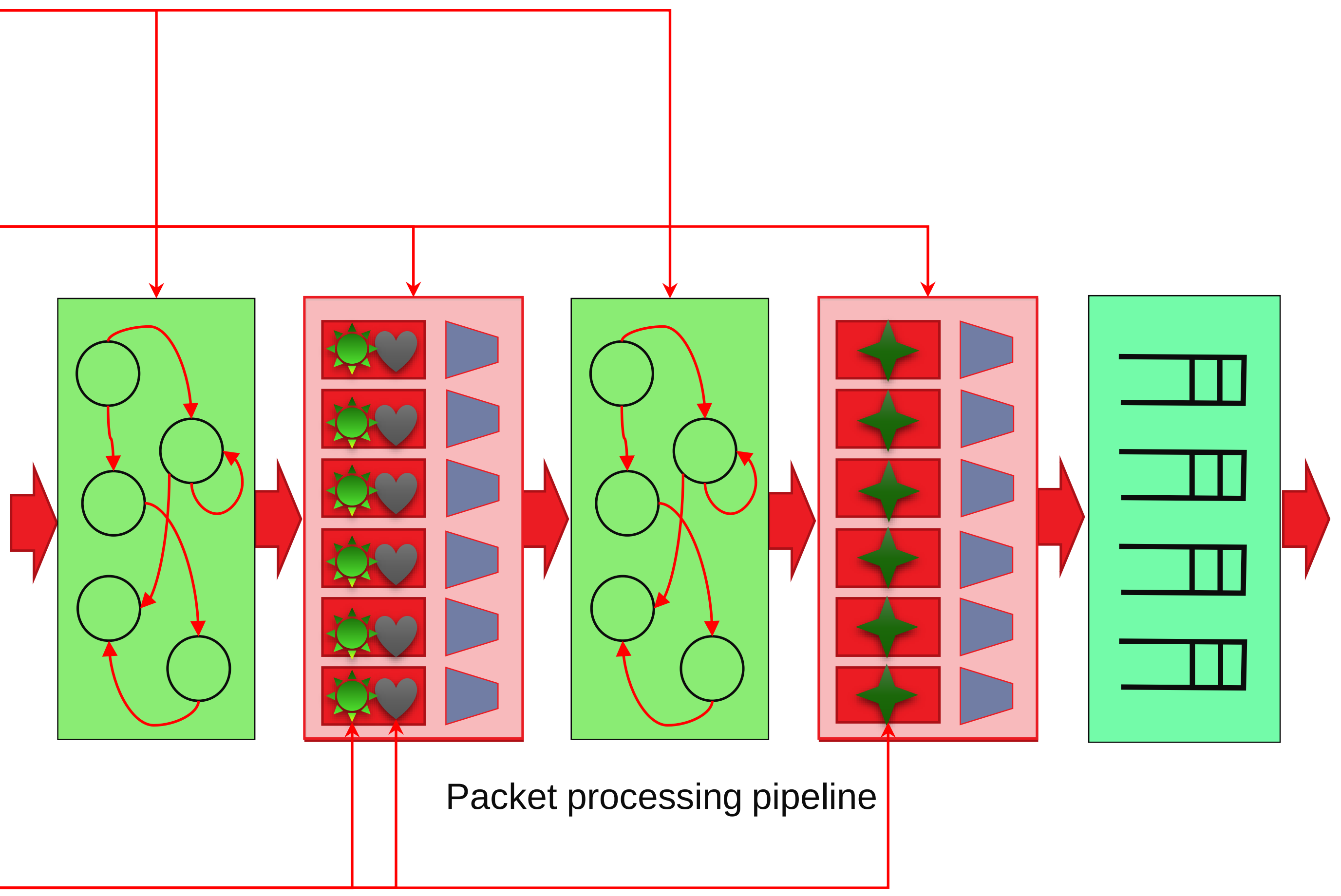
- **Language first appeared in paper published in July 2014**
  - Original version and early evolution now known as $P4_{14}$
  - Revised version known as $P4_{16}$ released in May 2017

- **Three goals:**
  - Reconfigurability in the field – reprogramming of networking equipment
  - Protocol independence – not tied to any specific networking protocols
  - Target independence – not tied to any specific networking hardware

- **P4 Language Consortium (P4.org) set up in 2015**
  - Xilinx was a founding member of P4.org
  - Now has >100 members

P4 language features … in one slide

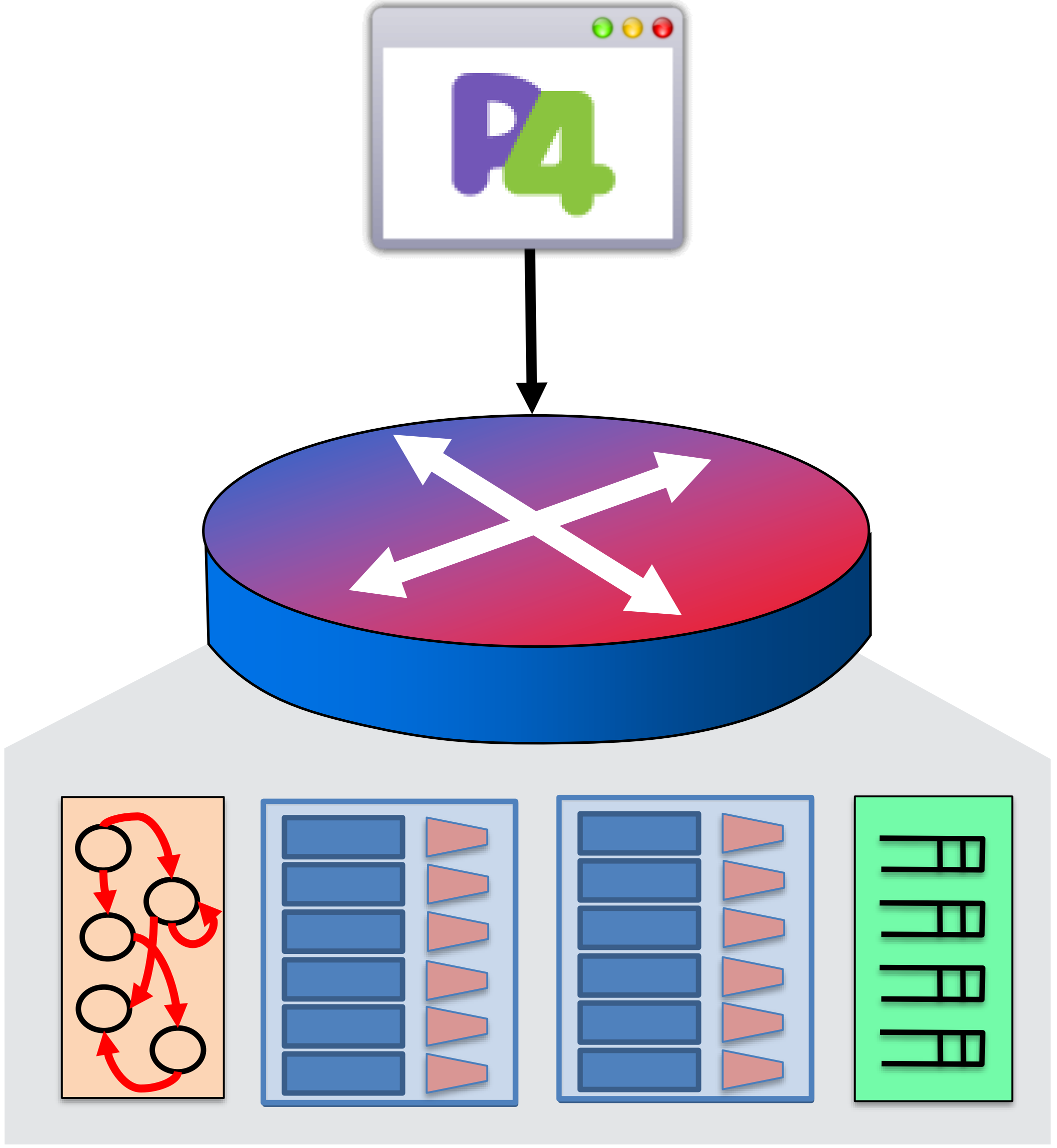
Parsers	State Machines, bit-field extraction
Controls	Match-Action Tables, control flow statements
Expressions	Basic operations and operators
Data Types	Bit-strings, headers, structures, arrays
Architecture	Programmable blocks and their interfaces
Extern Libraries	Support for specialized components



XILINX

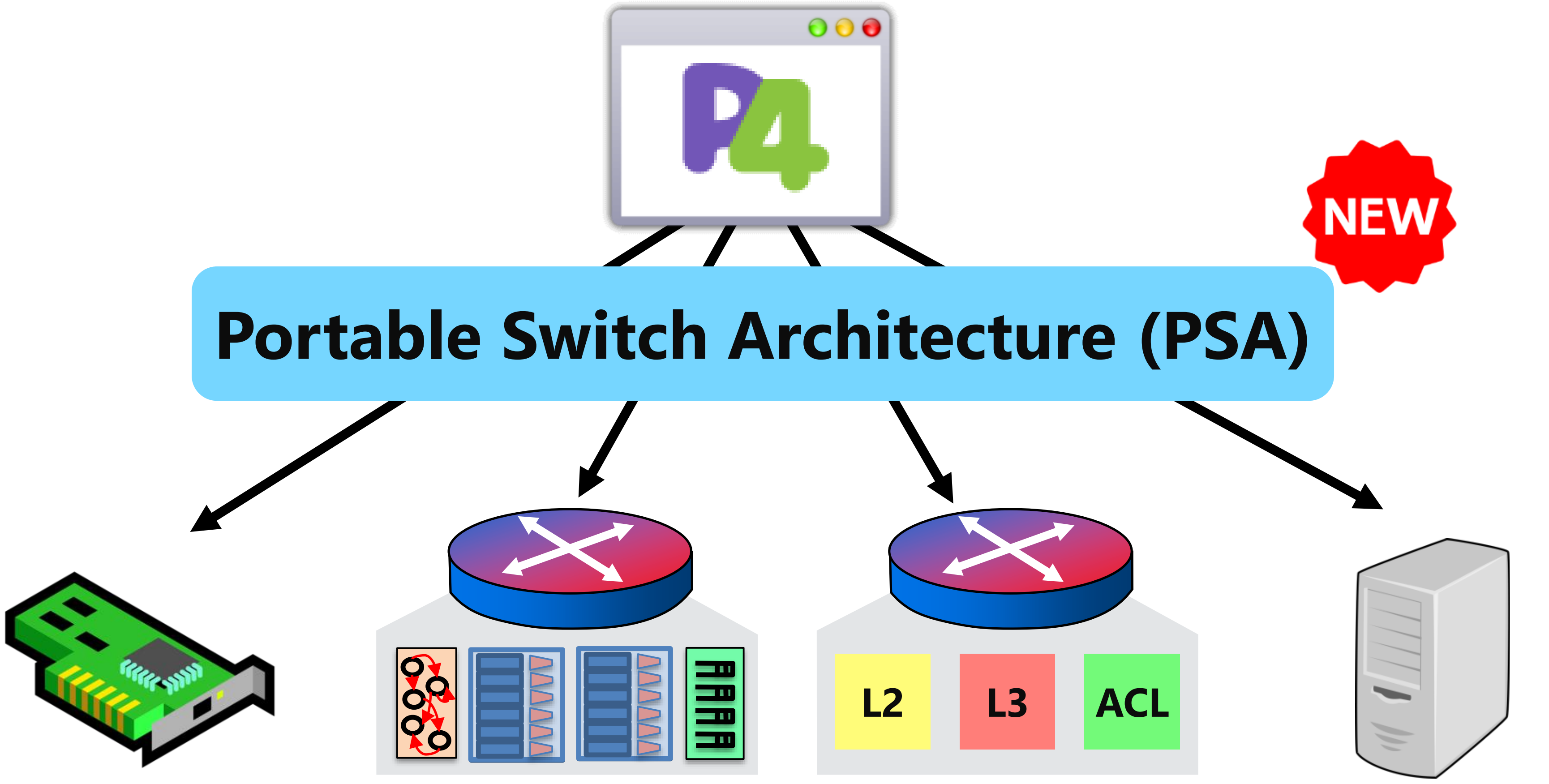
Original perspective (P4$_{14}$)

- Language Design WG



Diverse targets (P4$_{16}$)

- Language Design WG
- Architecture WG



Portable Switch Architecture (PSA)

FPGAs, NPUs

Programmable switch ASICs

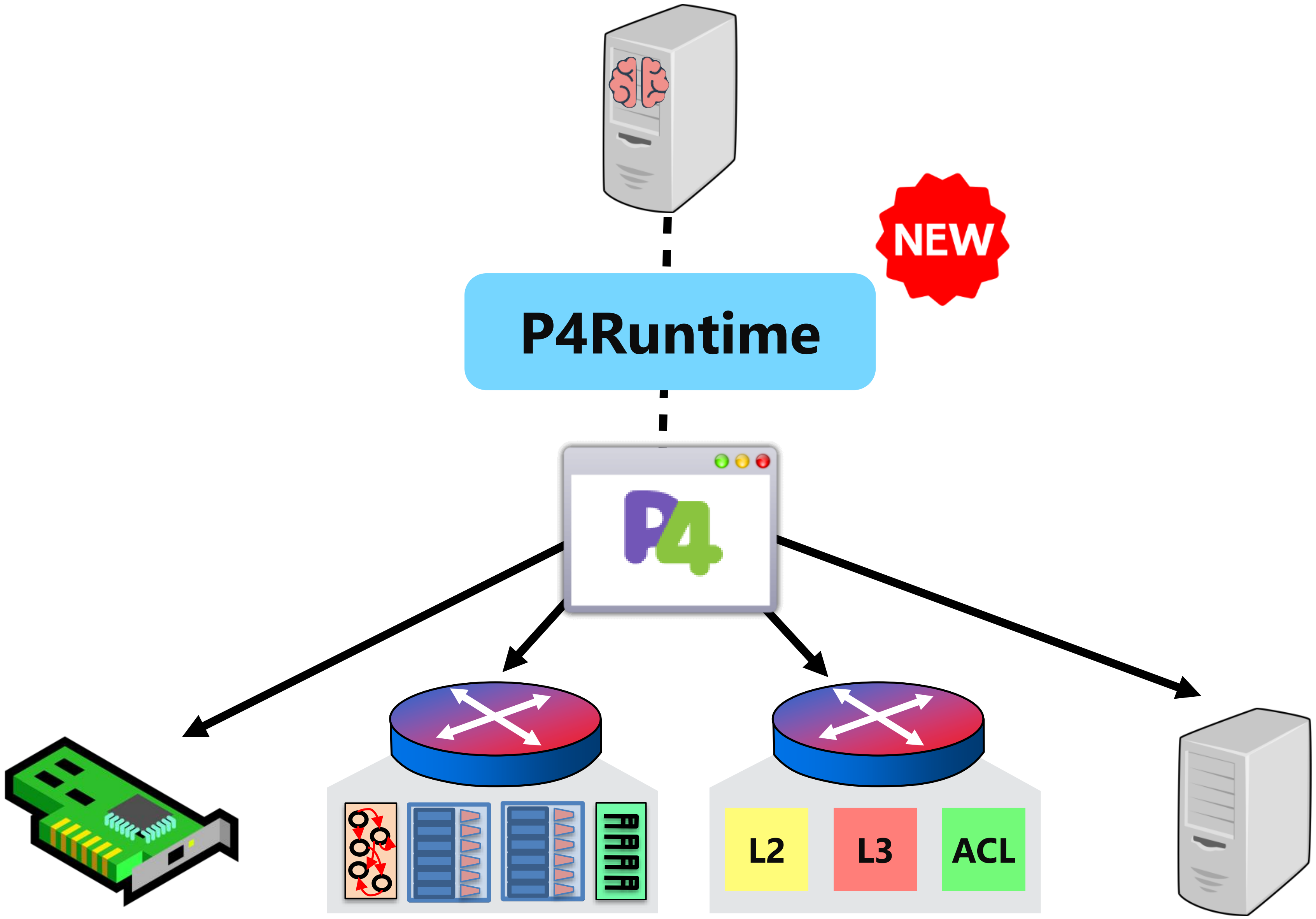
Fixed-function switch ASICs

Software switches

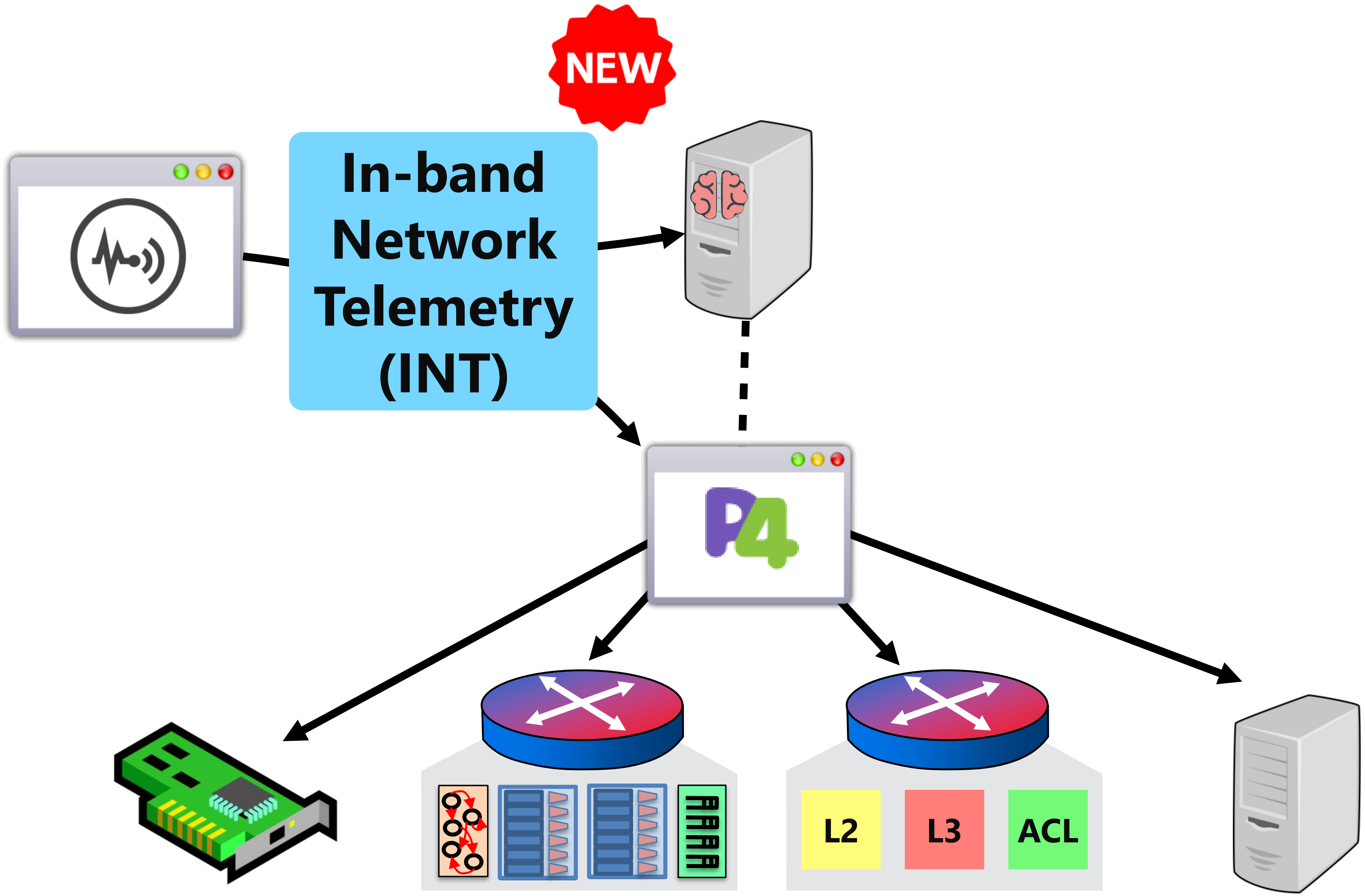
XILINX.

Complex control planes

- Language Design WG
- Architecture WG
- API WG



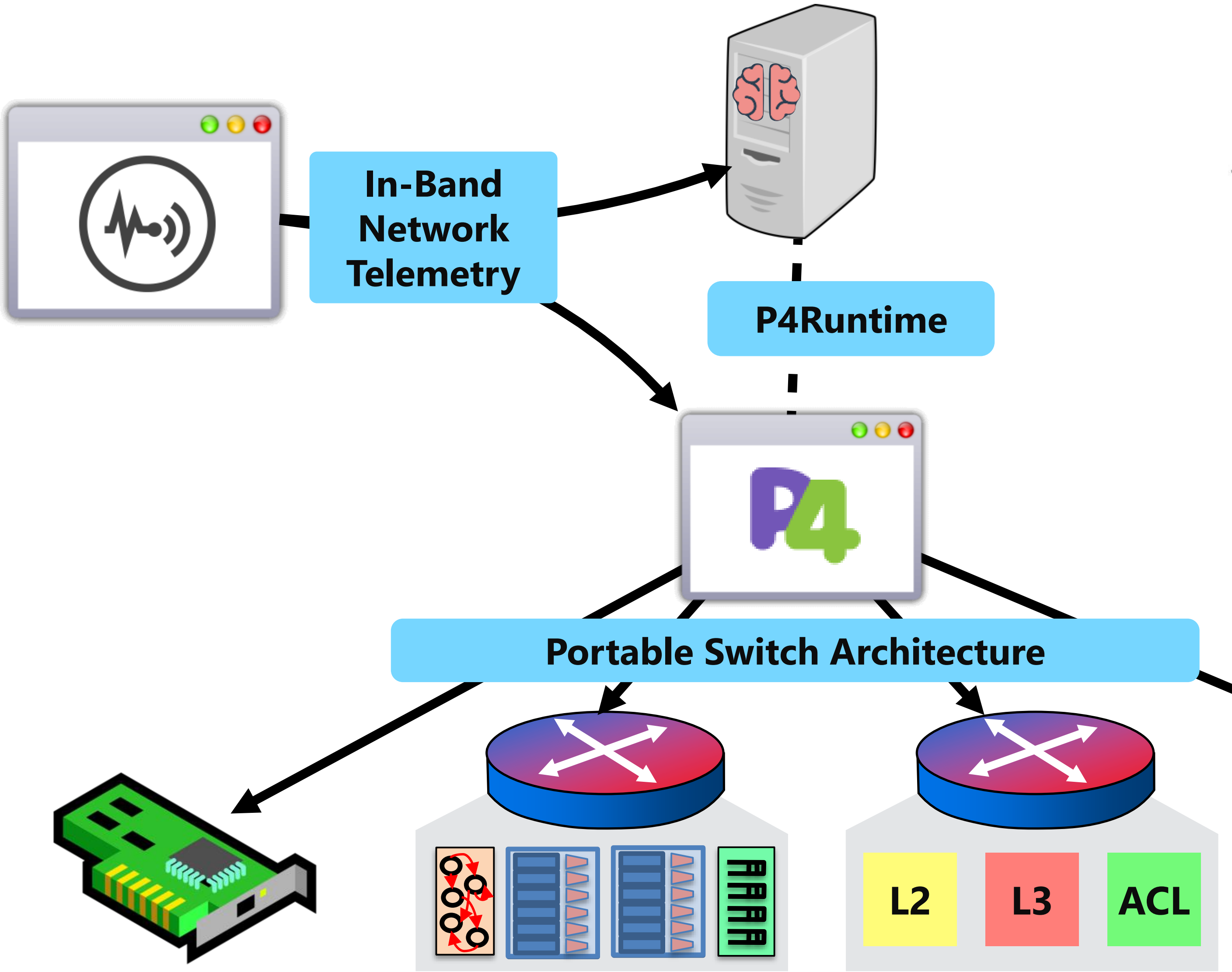
Rich applications



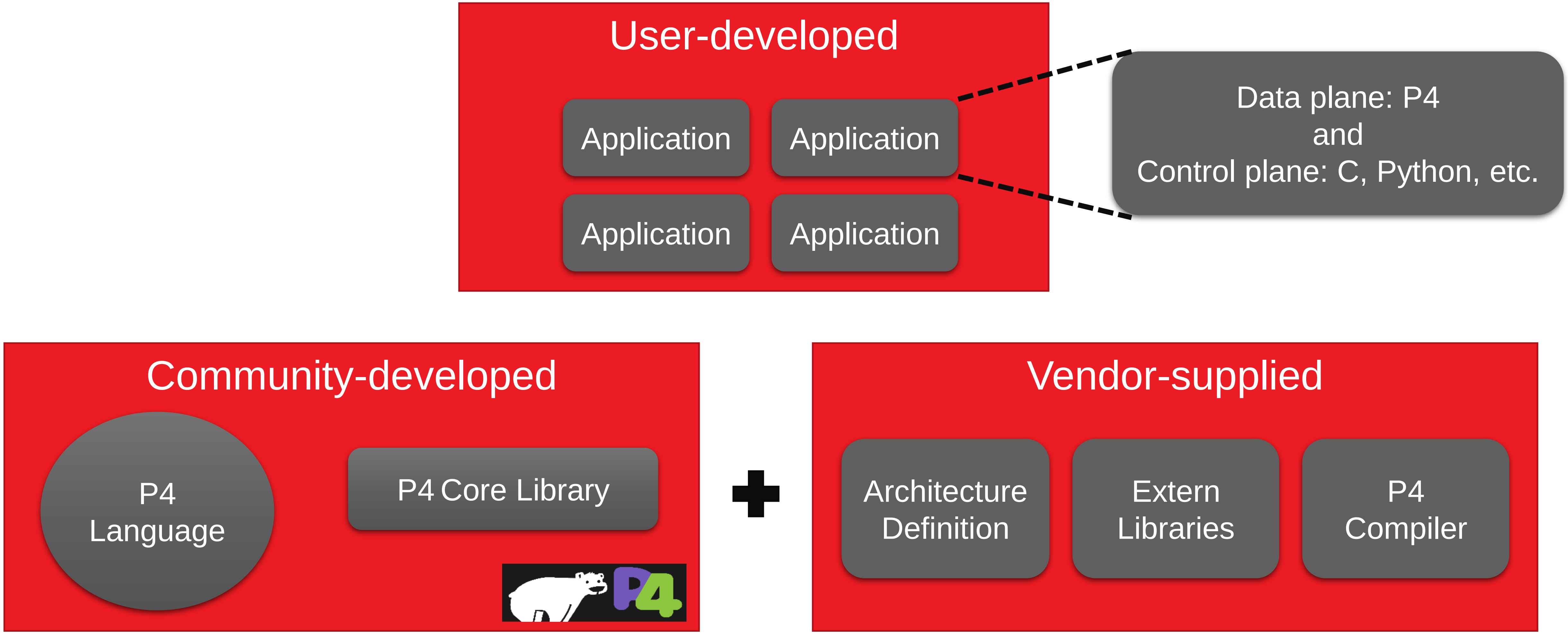
- Language Design WG
- Architecture WG
- API WG
- Applications WG

Education

- Language Design WG
- Architecture WG
- API WG
- Applications WG
- Education WG



P4 ecosystem



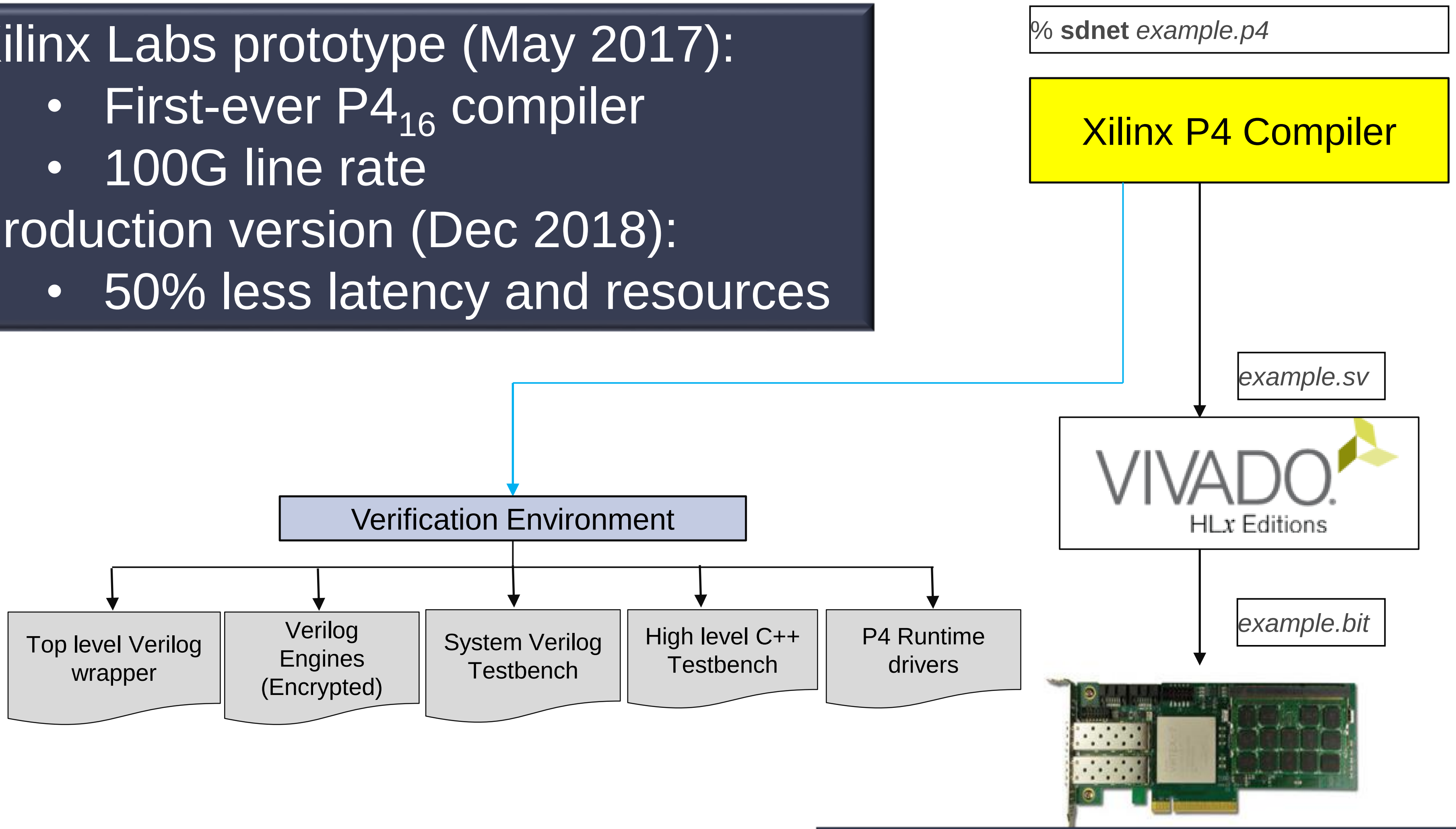
Xilinx (P4) SDNet product (www.xilinx.com/sdnet)

Xilinx Labs prototype (May 2017):

- First-ever $P4_{16}$ compiler
- 100G line rate

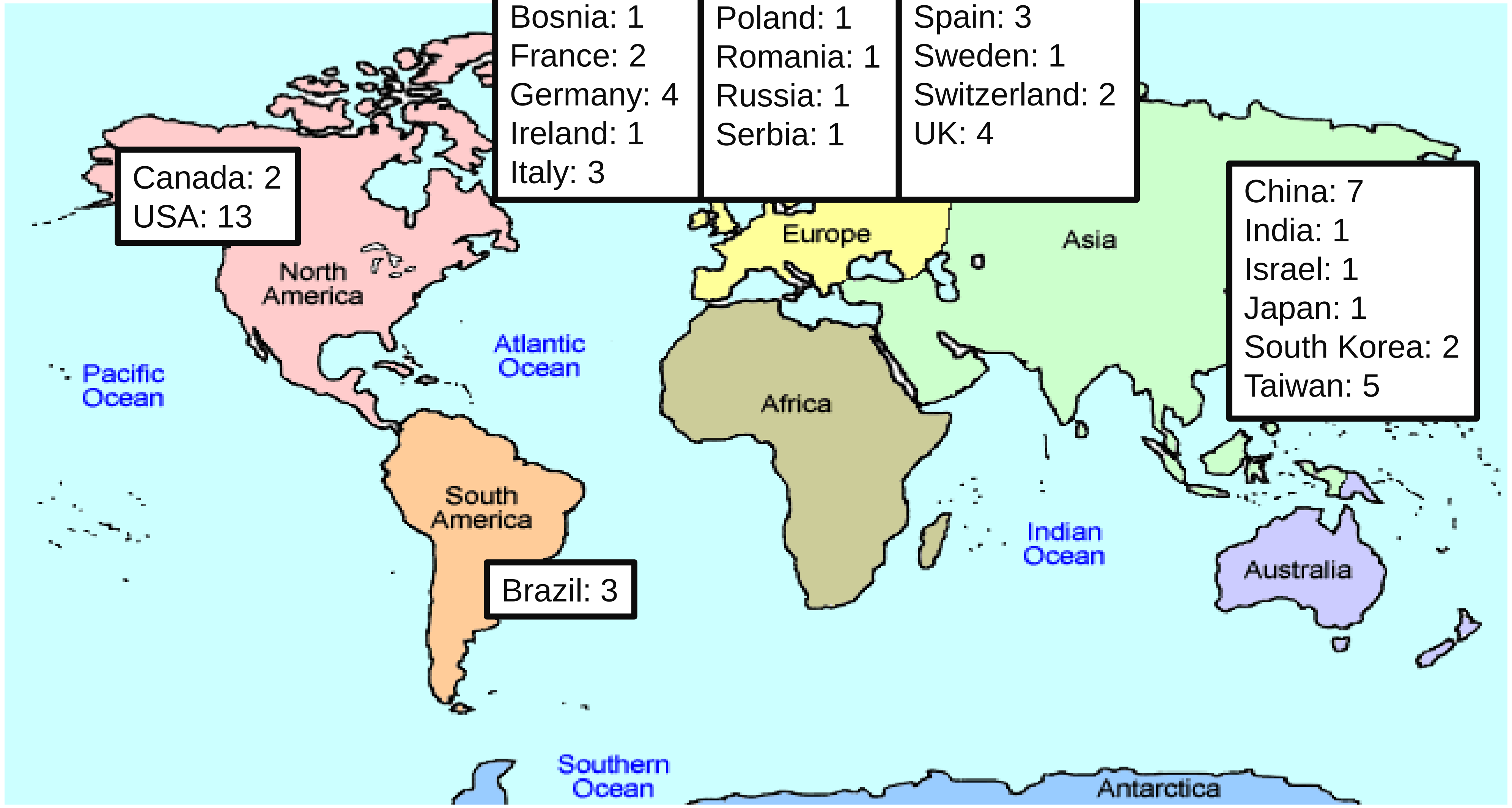
Production version (Dec 2018):

- 50% less latency and resources



Example target: Xilinx P4-Smart NIC card

SDNet-supported research community today:
60 institutions in 22 countries



Canada: 2
USA: 13

Brazil: 3

Bosnia: 1
France: 2
Germany: 4
Ireland: 1
Italy: 3

Poland: 1
Romania: 1
Russia: 1
Serbia: 1

Spain: 3
Sweden: 1
Switzerland: 2
UK: 4

China: 7
India: 1
Israel: 1
Japan: 1
South Korea: 2
Taiwan: 5

Status of P4

- **Industry Momentum**
  - Diverse collection of P4-enabled targets
  - Growing number of P4-based products
  - Real-world deployments
- **Academic Interest**
  - Research papers at top conferences
  - New courses at leading universities
- **Open Source Community**
  - Vibrant technical working groups
  - Powerful set of P4 tools
  - P4.org joined Linux Foundation this year

"Our whole networking industry stands to benefit from a language like P4 that unambiguously specifies forwarding behavior, with dividends paid in software developer productivity, hardware interoperability, and furthering of open systems and customer choice."

— Tom Edsall, Cisco

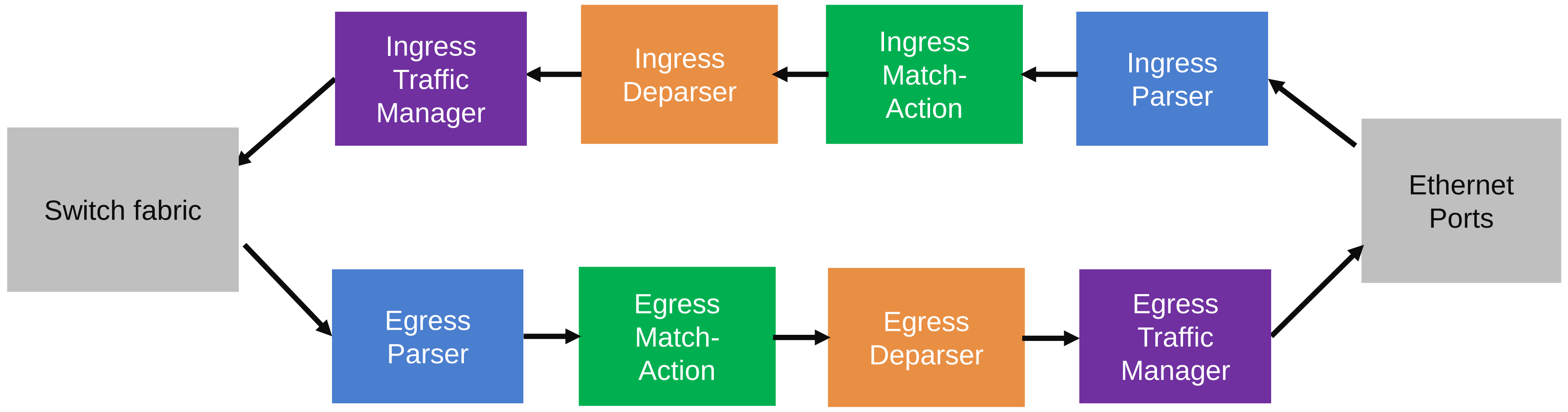
Portable NIC Architecture

P4 community desire
New P4.org Architecture sub-group

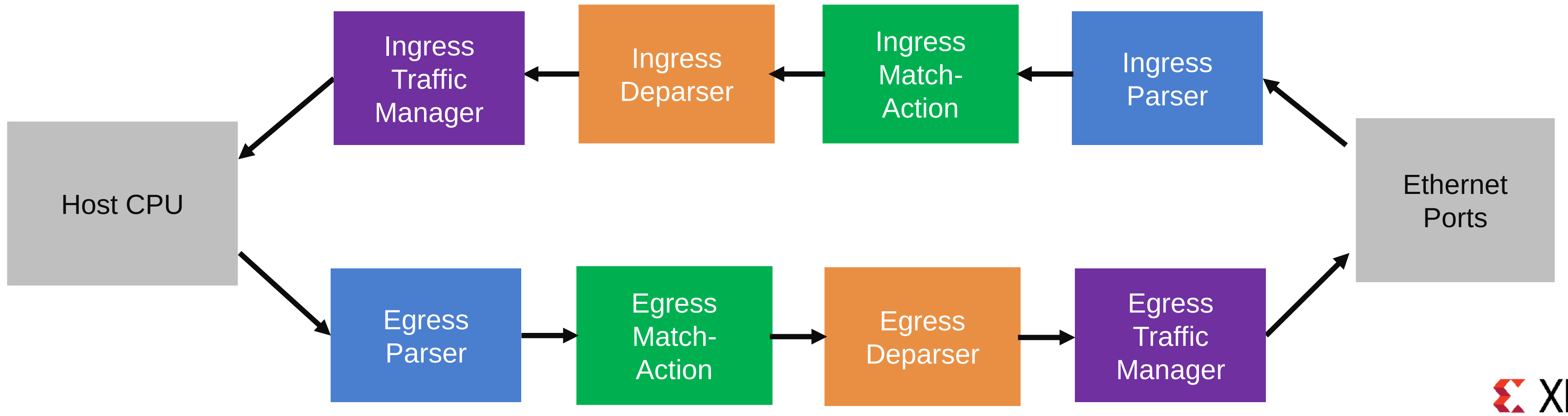
XILINX

Switch vs. NIC: Superficially similar …

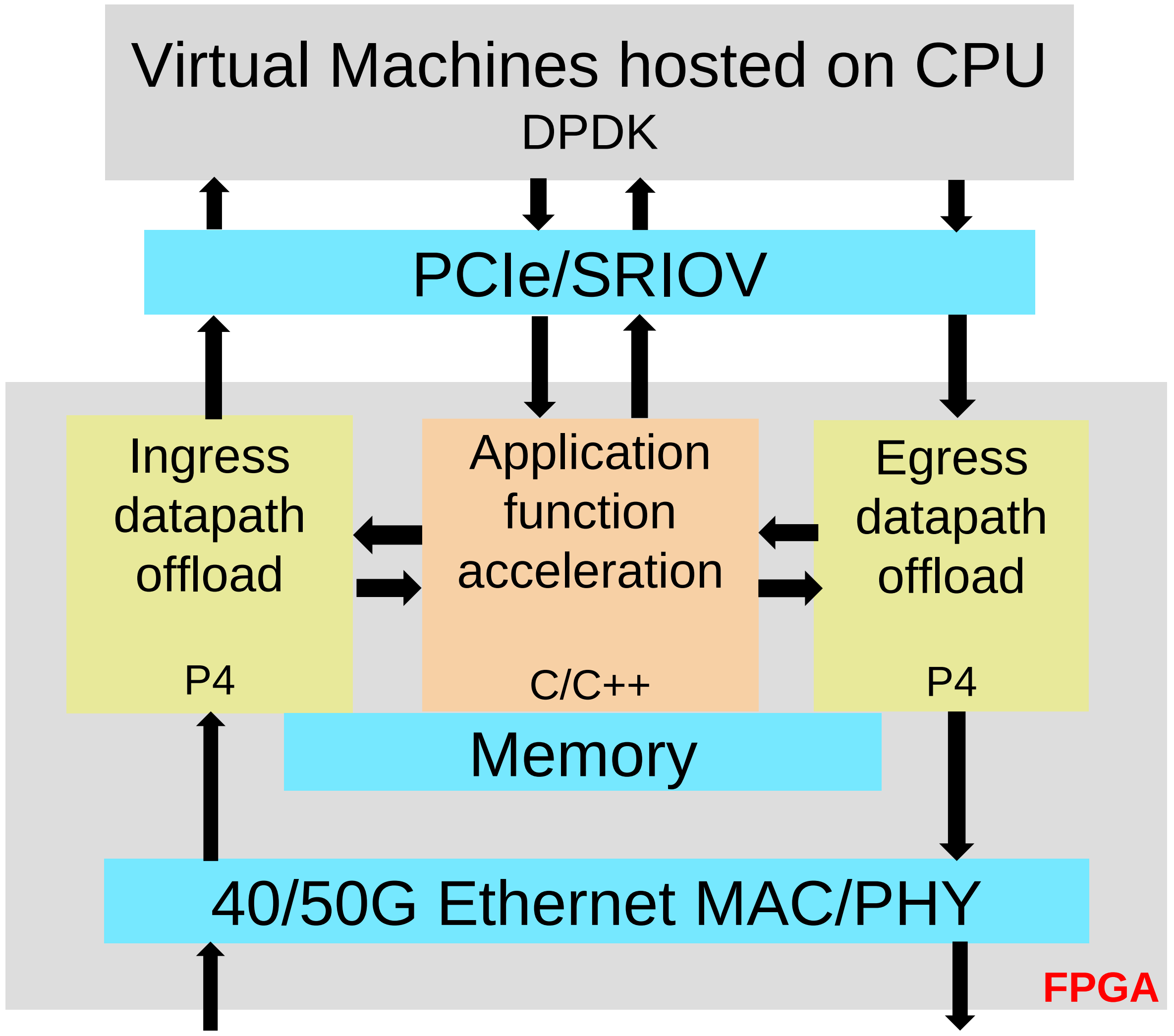
> **Switch-style architecture**



> **NIC-style architecture**

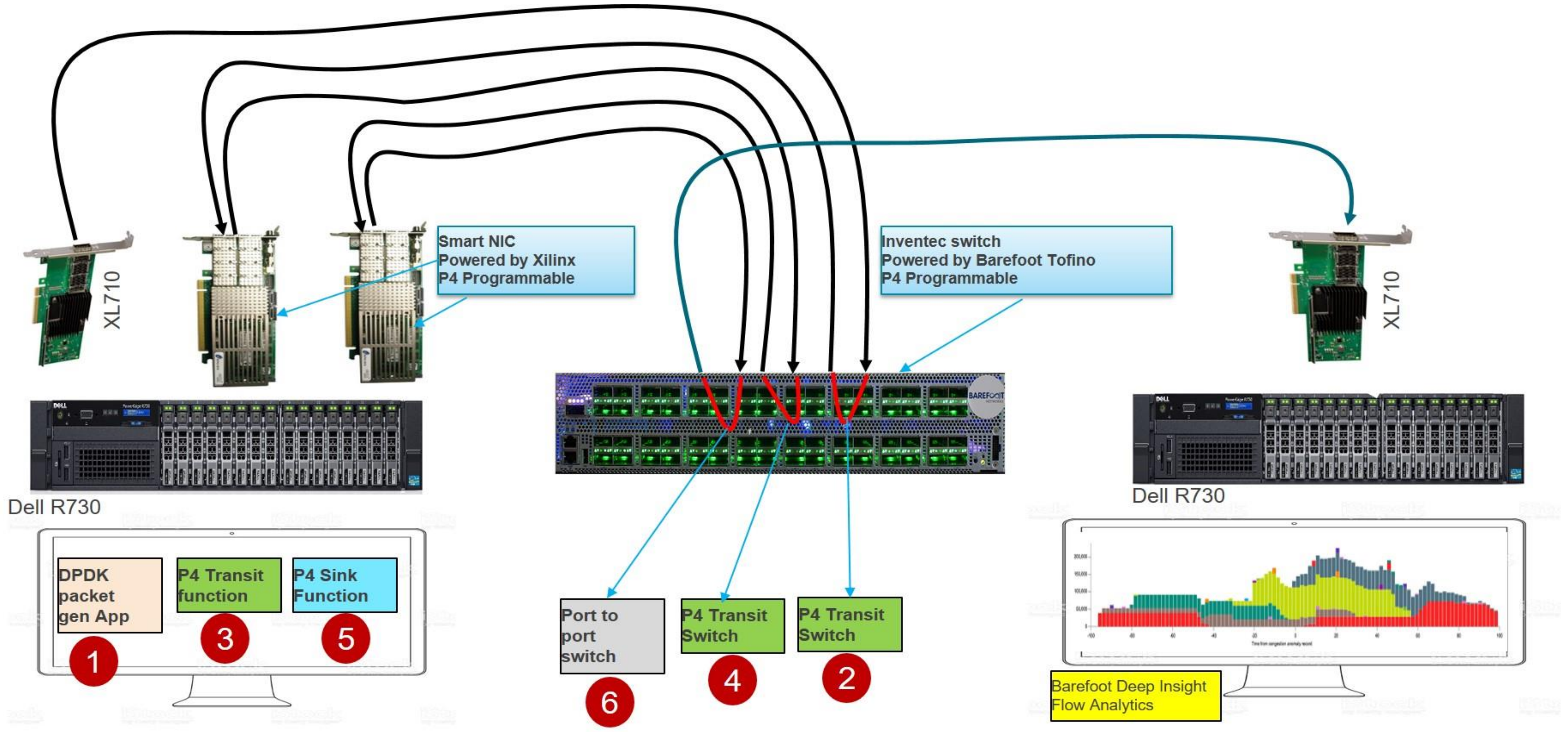


Xilinx Labs Smart NIC prototype (evolved 2015-2018)



Xilinx NICs and Barefoot switch:
In-band Network Telemetry (INT) inter-operability

Demonstrated at MWC 2018 and OFC 2018

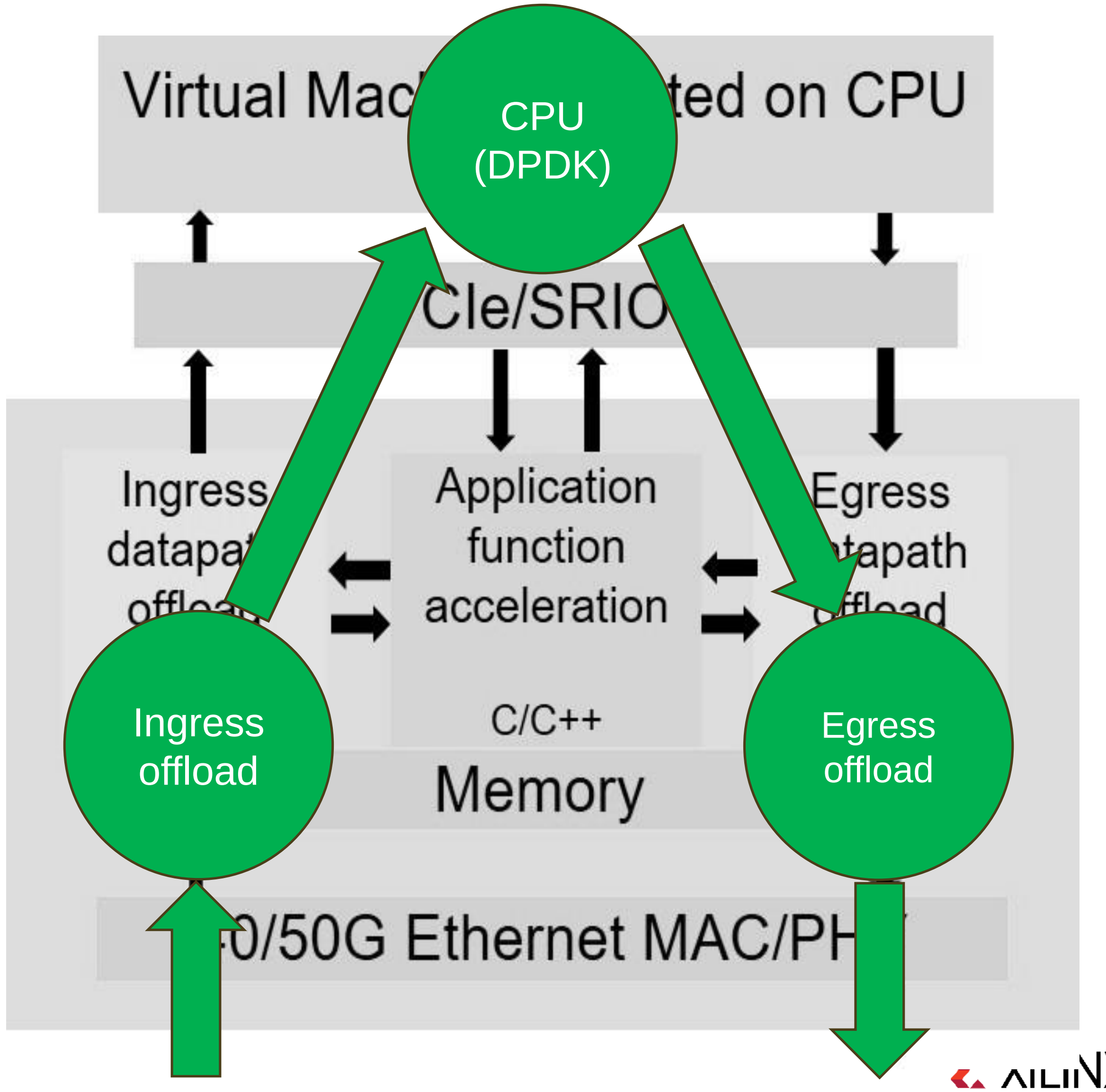


XILINX.

Use Case 1/3: Basic NIC ingress and egress

> **Example:**

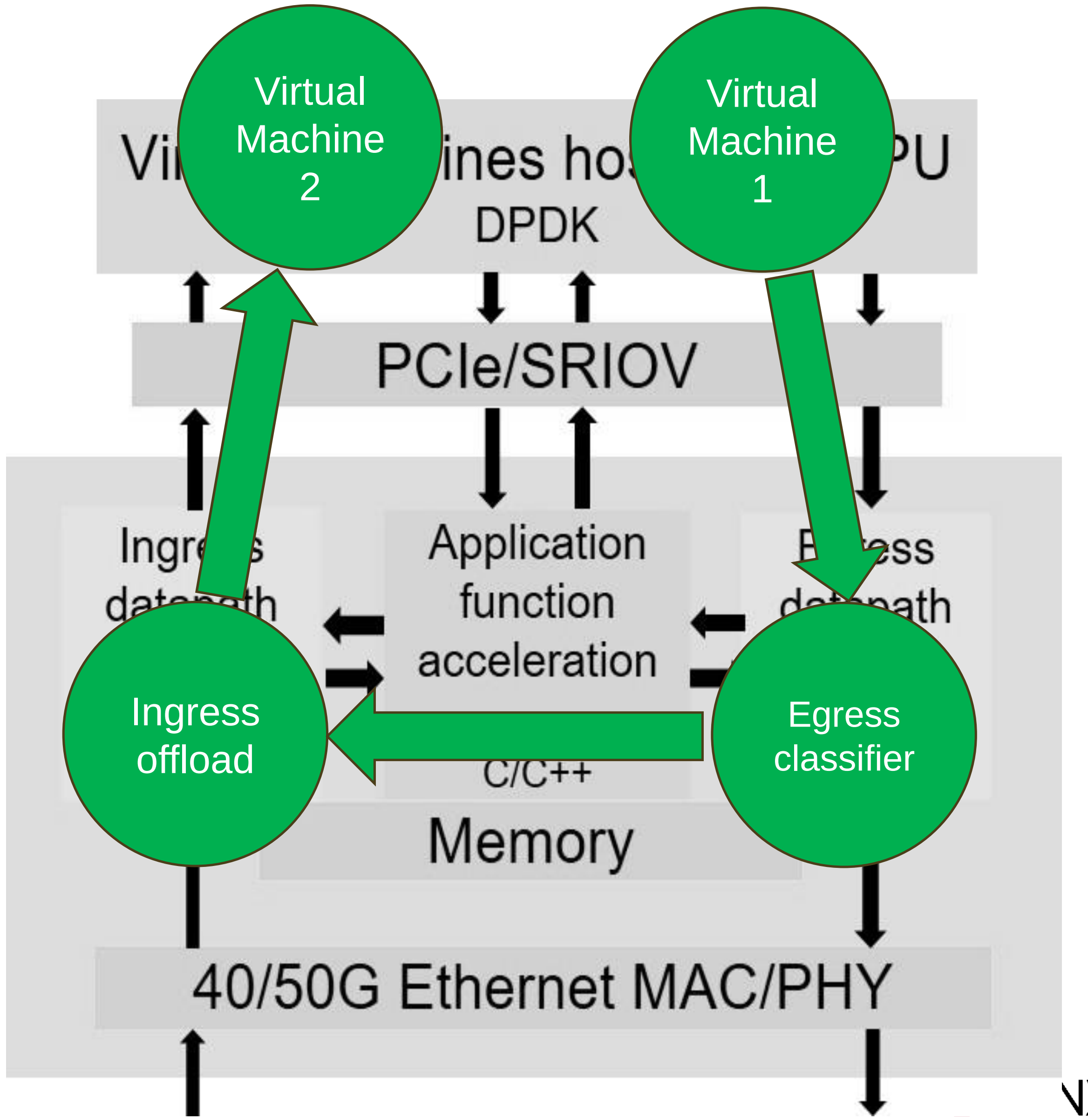
- 40Gb/s IP packet forwarding
- 1 CPU core needed instead of 6 CPU cores
- Full line rate with 64-byte packets



Use Case 2/3: Direct egress to ingress bridging

> **Example:**

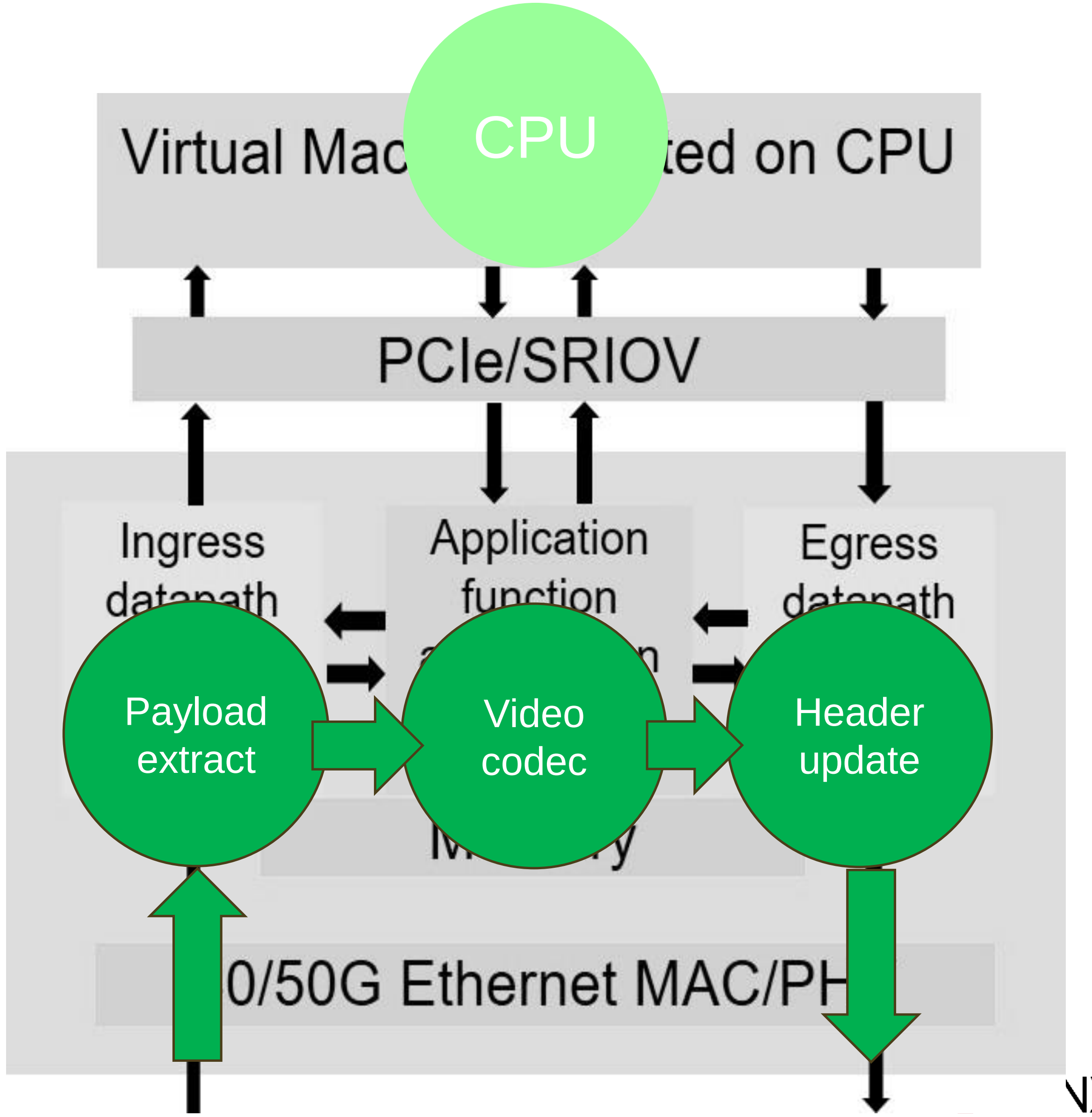
- NFV Service Function Chaining (SFC)
  - Offload of NSH protocol used for SFC
- 5x reduction in VM-to-VM latency
- Throughput matches the PCIe bandwidth



Use Case 3/3: Bump-in-wire acceleration

> **CPU out of main processing loop**
>> Just used for configuration and exceptions

> **Example:**
>> Video Transcoding appliance
>> Accelerate video coding
>> 25x better frames/second per Watt



Some Portable NIC Architecture (PNA) discussions

- **Expect there to be separate ingress and egress pipelines**
  - What are the standard components of each pipeline? Are there pipeline variants?
  - Which components are P4-programmable?
  - Is direct interaction between ingress and egress, and egress and ingress, allowed?

- **How is host CPU interface modelled?**
  - Differentiate data plane CPU roles, and control plane CPU roles
  - Impact on P4Runtime

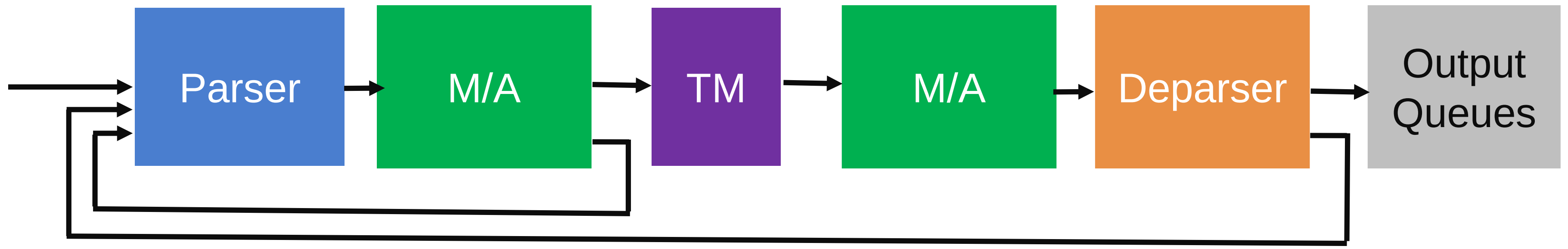
- **Beyond packet forwarding (future steps – of general P4 interest)**
  - Is protocol (e.g., TCP) termination covered?
  - Is ‘Type 3’ NIC covered – payload processing as well?

Programmable Target Architecture

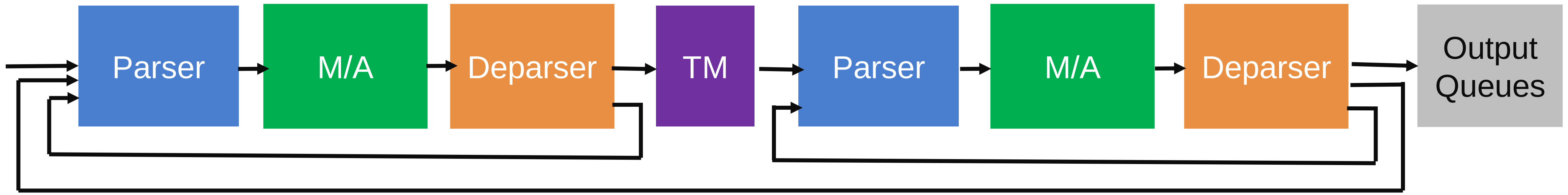
Stanford, Xilinx Labs
Now in discussion with Barefoot, Cornell, VMware Research

Examples of the many possible target architectures

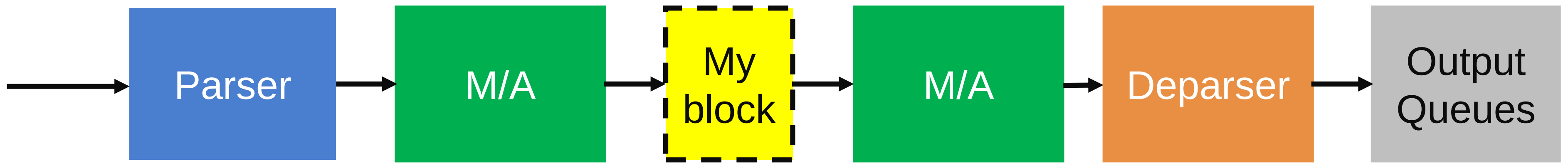
V1 Model



Portable Switch Architecture (PSA)



Custom in-line processing



Programmable Target Architecture (PTA)

> **Motivations**

- Extend P4 (“P4+”) to allow description of target architectures: components and connectivity
- End-to-end P4 program verification relative to particular architectures
- Explore performance tradeoffs of various architectures

> **Three actors**

NEW

(1) Target architecture designer

Implements:
- Externs in target architecture
  - In-line (packet processing)
  - Look-aside (header processing)
- P4Runtime+ API for externs

Provides:
- P4+ architecture description

(2) P4 programmer

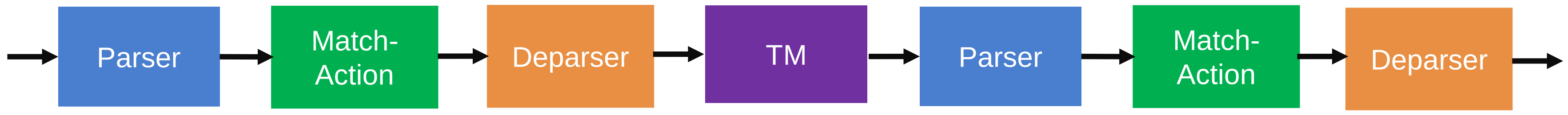
Implements:
- P4-programmable “holes” in the target architecture

(3) Runtime programmer

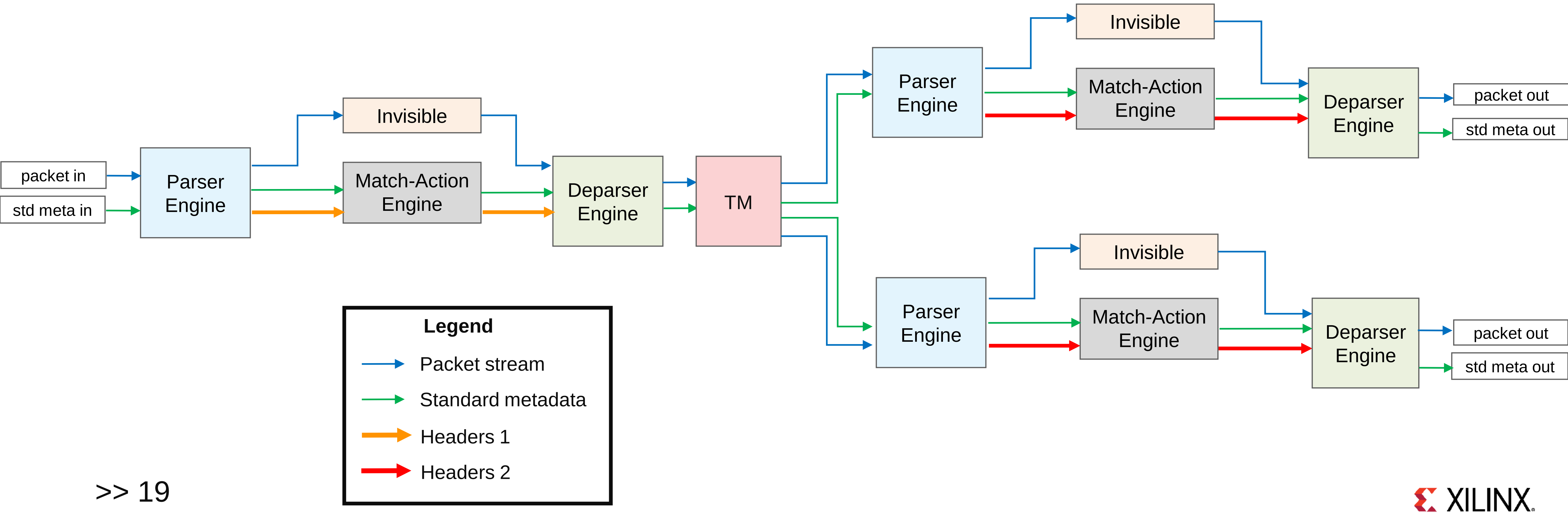
Implements:
- Runtime controller for P4-populated target architecture

Example: Custom target architecture

Logical P4 pipeline view:



Internal design view:



XILINX

Custom architecture description using experimental P4+

```
#define NUM_PORTS 2
struct std_meta_t {...}

// Define (header processing) externs ...

// Define Architectural Elements
parser Parser<H>(packet_in        p_in,
                 out H            *headers, // * distinguishes between headers and metadata
                 inout std_meta_t std_meta,
                 packet_out       .p_out);  // . indicates that port is hidden (i.e. invisible at this pipeline stage)

control Pipe<H>(inout H          *headers,
                inout std_meta_t std_meta,
                packet_inout     .p);

control Deparser<H>(packet_out       p_out,
                    in H             *headers,
                    inout std_meta_t std_meta,
                    packet_in        .p_in);

extern TM(packet_in        p_in,
          in std_meta_t    std_meta_in,
          (
            packet_out     p_out,
            out std_meta_t std_meta
          )*NUM_PORTS); // * operator indicates replicated ports

package Example<H1, H2> (Parser<H1> p1,
                         Pipe<H1> map1,
                         Deparser<H1> d1,
                         TM tm,
                         Parser<H2> p2,
                         Pipe<H2> map2,
                         Deparser<H2> d2) {
    // * operator indicates forked replication
    arch = {p1, map1, d1, tm, (p2, map2, d2)*NUM_PORTS}
}
```

P4+ code:
Written by target architecture designer

XILINX

Custom architecture Interface (auto-generated)

```
struct std_meta_t {...}

// Define (header processing) externs ...

// Define Architectural Elements
parser Parser<H>(packet_in p_in,
                 out H headers,
                 inout std_meta_t std_meta);

control Pipe<H>(inout H headers,
                inout std_meta_t std_meta);

control Deparser<H>(packet_out p_out,
                    in H headers,
                    inout std_meta_t std_meta);

package Example<H1, H2> (Parser<H1> p1,
                         Pipe<H1> map1,
                         Deparser<H1> d1,
                         Parser<H2> p2,
                         Pipe<H2> map2,
                         Deparser<H2> d2);
```

Standard P4 code: Imported by P4 programmer

Prototype P4+ workflow being demonstrated at P4EU today

XILINX.

Programmable Traffic Manager

MIT, NYU, Stanford, Xilinx Labs
New P4.org Architecture sub-group

What is Traffic Management?

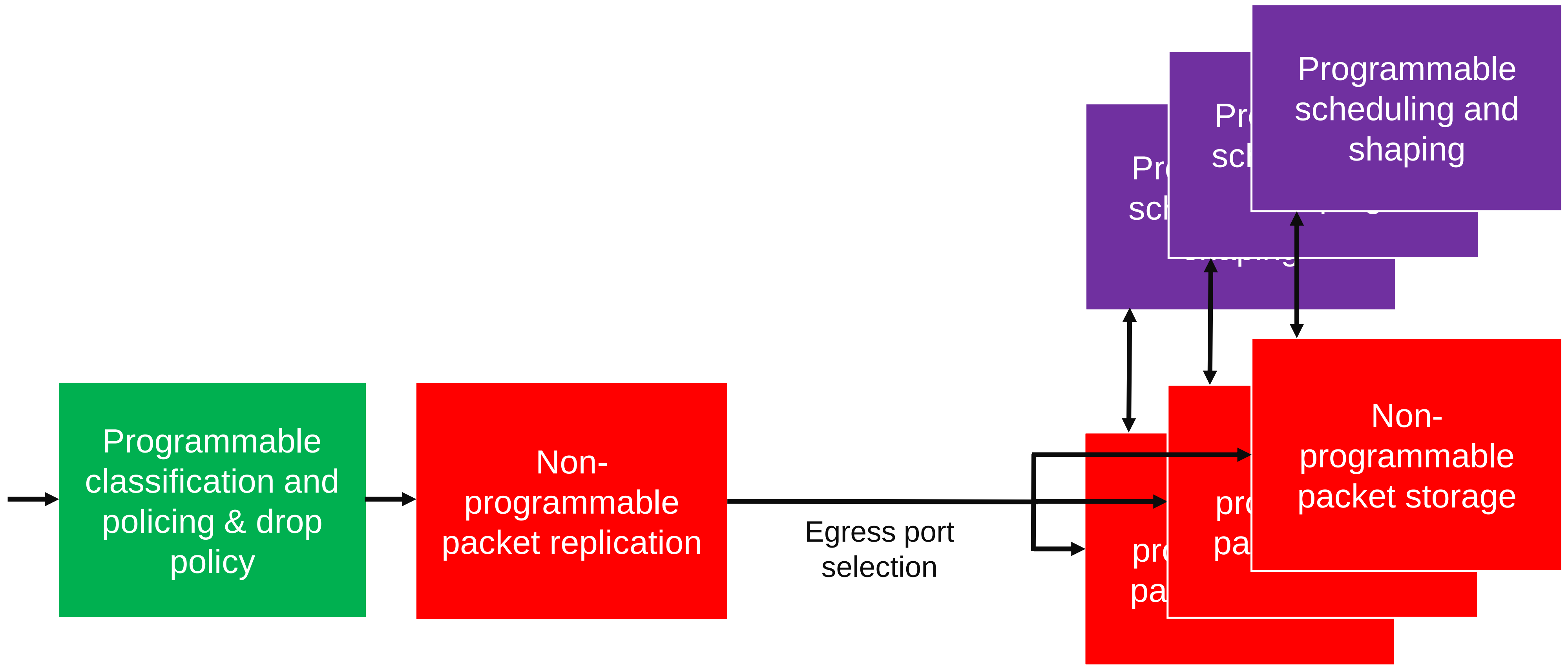
> **Policing: compliance with agreed rate**
> **Drop policy: how to avoid/deal with congestion**
> **Replication: cloning and multicasting packets**
> **Packet buffering: temporary storage of packets**
> **Packet scheduling: determining order of transmission**
> **Traffic shaping: forcing rate and pace**

> **Associated with *Classification* – mapping packet flows to egress ports and queues**

Why should we care about Traffic Management?

- **Lots of different types of traffic with different characteristics and requirements**
  - Characteristics: burstiness, packet sizes, flow sizes, flow rates
  - Requirements: throughput, latency, loss, jitter, reordering, flow completion time, pacing
- **Network operators have a wide range of objectives**
  - Meet all Service Level Agreements
  - Maximize network utilization
  - Achieve fairness, while prioritizing certain traffic
- **Network devices are acquiring more TM functionality**
  - About 50% of a modern programmable switch chip is dedicated to traffic management and buffering – but this part *is currently not programmable*
- **Particular programmability benefits, alongside general P4 benefits**
  - Network operators can fine-tune for performance
  - Small menu of standard algorithms to choose from today
  - … Many possible algorithms that can be expressed

Programmable Traffic Manager (PTM) architecture



Programmable classification and policing & drop policy
Non-programmable packet replication
Egress port selection
Programmable scheduling and shaping
Non-programmable packet storage

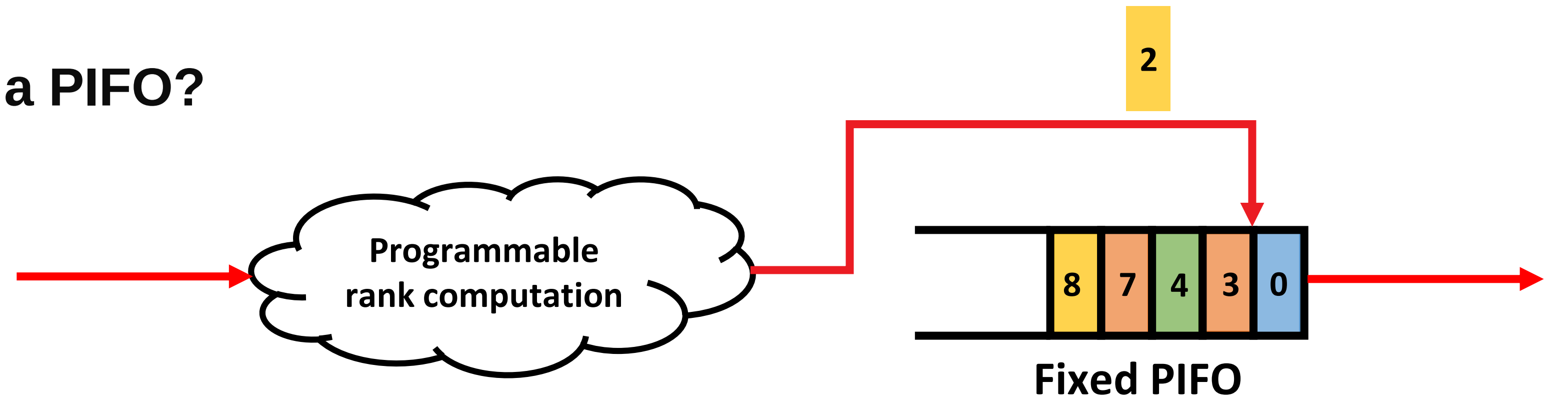
Buffering and queueing for each egress port

May have many associated queues per port

XILINX.

The Push-In-First-Out (PIFO) model [SIGCOMM 2016]

> **What is a PIFO?**



Programmable scheduling and shaping

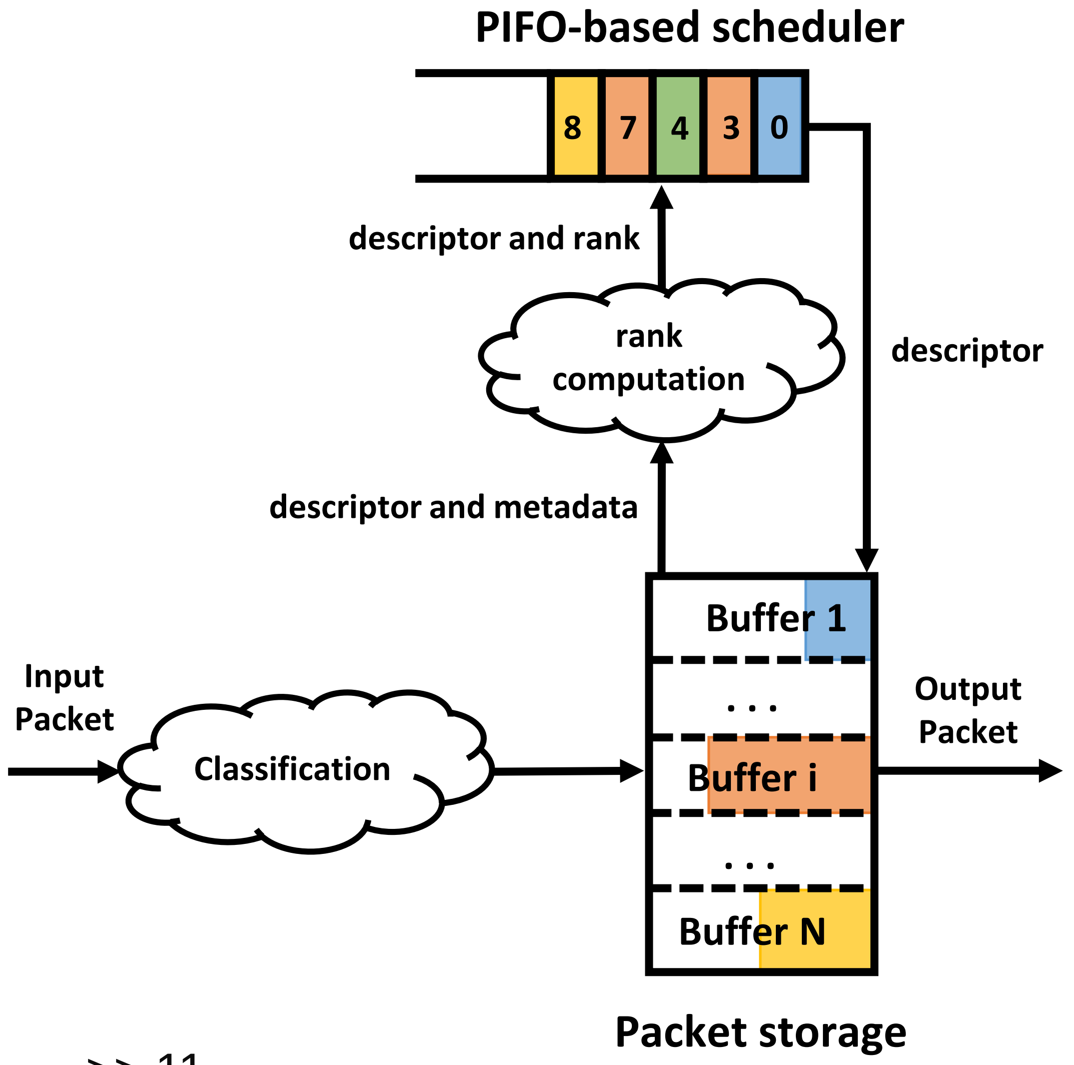
> **Why is the PIFO a good model for scheduling and shaping?**
>> Ordering decision made at time of enqueue → helps relax timing pressure at output ports
>> Clear separation between programmable part and fixed part

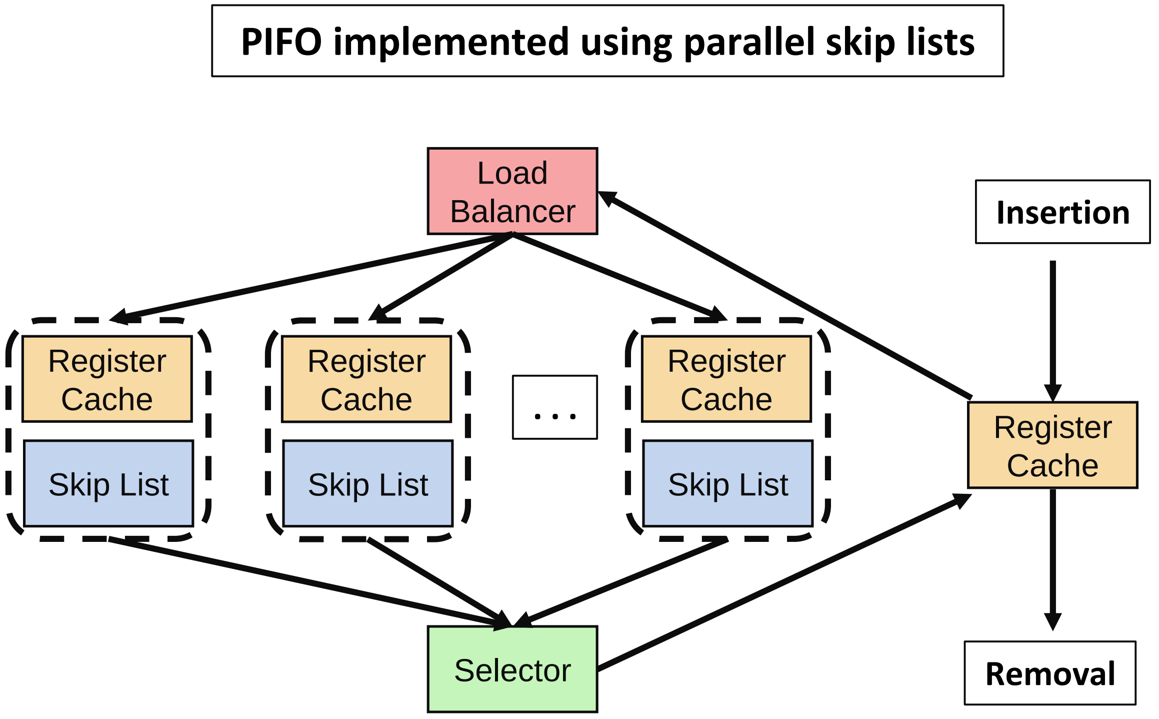
> **Can implement existing algorithms, for example:**
>> Start Time Fair Queueing (STFQ), Least Slack-Time First (LSTF), Stop-and-Go Queueing, Minimum rate guarantees, fine grained priority scheduling, Service-Curved Earliest Deadline First (SC-EDF), Rate-Controlled Service Disciplines (RCSD)
>> Token bucket rate limiting

> **Can implement new algorithms using programmable rank computation**

Prototype implemented on FPGA for 4x10G line rate

NYU+Stanford+Xilinx Labs demonstration at P4 Workshop, June 2018



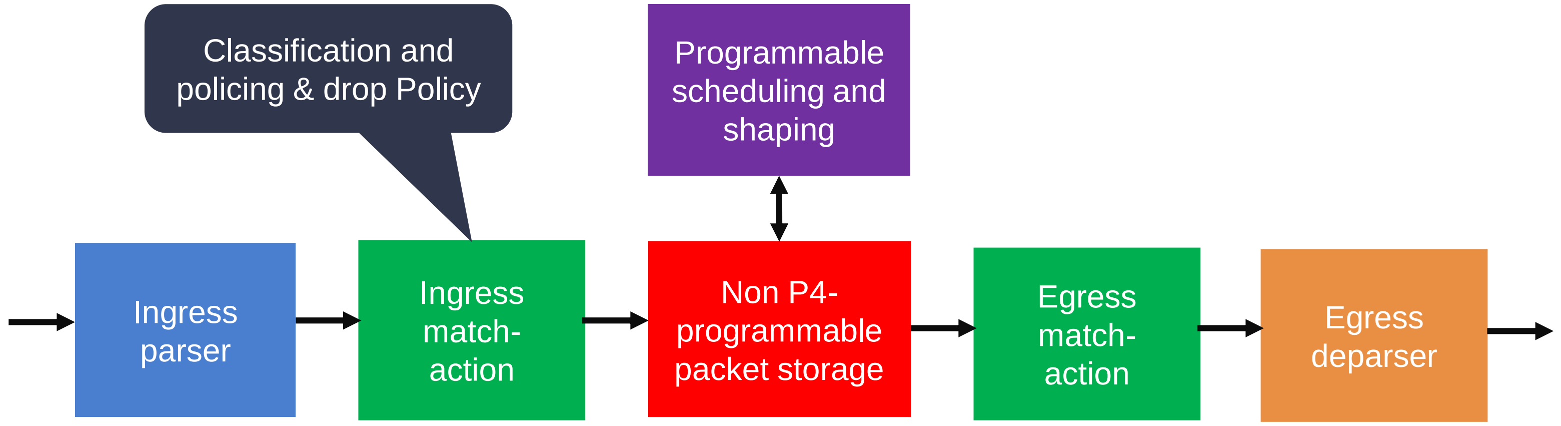


Example: Possible P4 pipeline extension for TM

User defined scheduling metadata

```
parser Parser<H, M>(packet_in b,
                    out H hdr,
                    out M user_meta,
                    inout std_meta_t std_meta);

control Ingress<H, M, D>(inout H hdr,
                         out D sched_meta,
                         inout M user_meta,
                         inout std_meta_t std_meta);
```

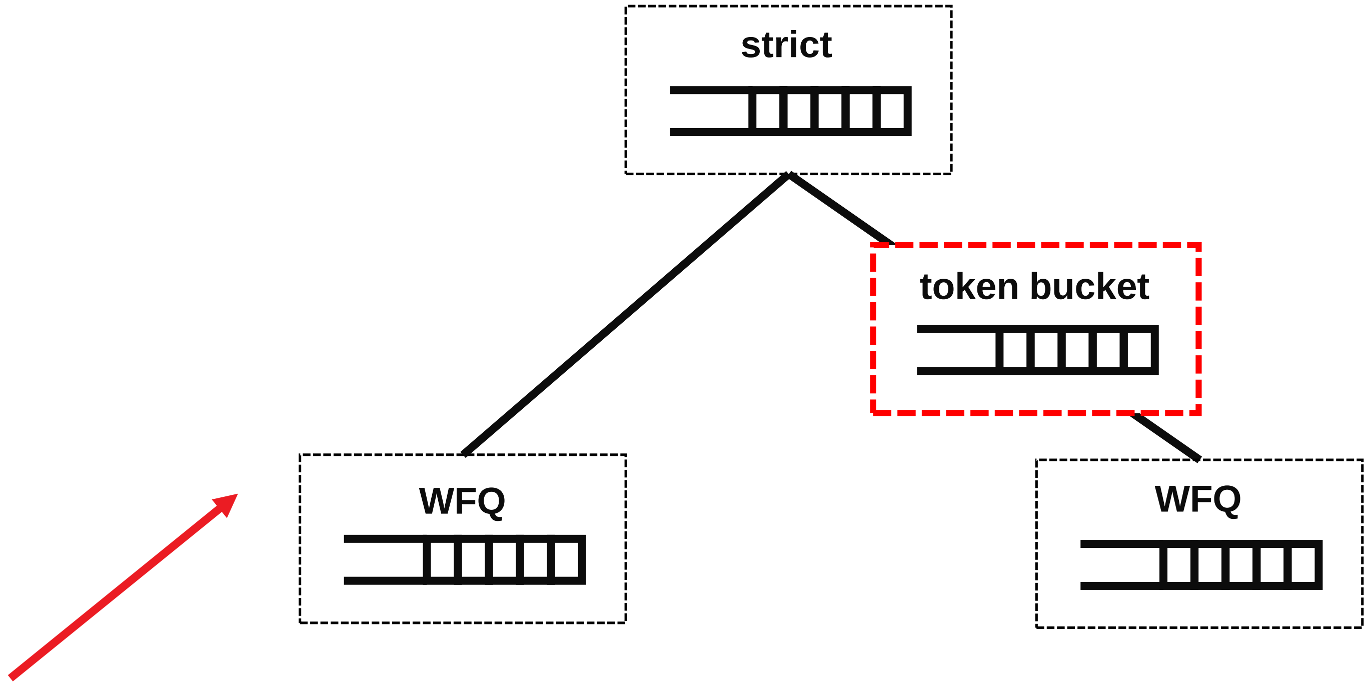
```
scheduler MyScheduler<D>(in D sched_meta);

control Egress<H, M>(inout H hdr,
                     inout M user_meta,
                     inout std_meta_t std_meta);

control Deparser<H, M>(packet_out b,
                       in H hdr,
                       in M user_meta,
                       inout std_meta_t std_meta);
```


Classification and policing & drop Policy
Programmable scheduling and shaping
Ingress parser
Ingress match-action
Non P4-programmable packet storage
Egress match-action
Egress deparser

XILINX.

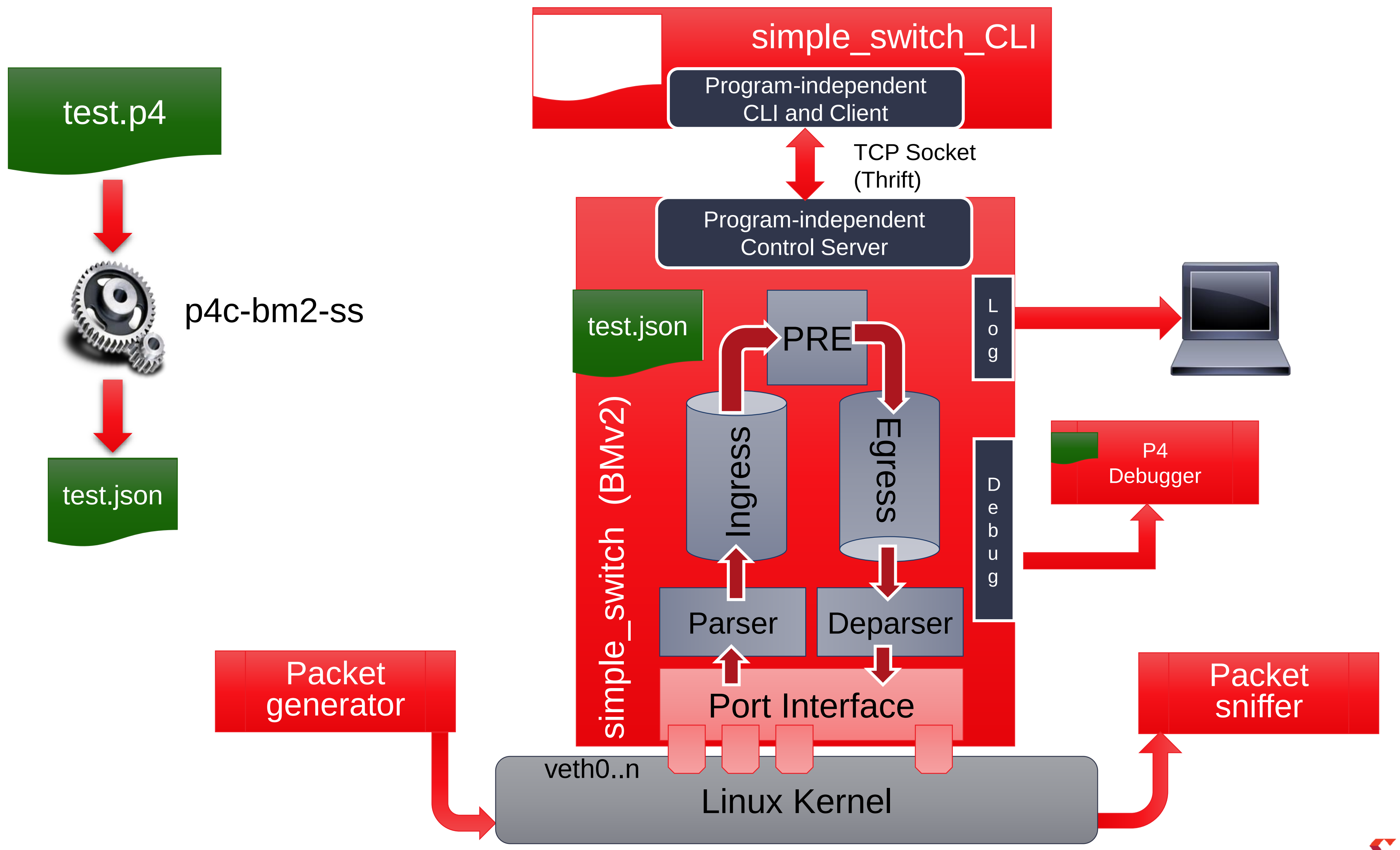
Example: Possible P4 extension for scheduler/shaper

```
scheduler MyScheduler(in sched_meta_t sched_meta)
{
   /* Define PIFO tree nodes */
   /* root scheduling node */
   node strict_priority {
        type = scheduling;
        pifo<rank_t>(2048) p;
        enqueue = { … }
        dequeue = { … }
   }
   /* shaping node */
   node token_bucket {
        type = shaping;
        pifo<rank_t, sched_meta_t>(2048) p;
        enqueue = { … }
        dequeue = { … }
   }
   /* Define the shape of the scheduling/shaping tree */
   tree myTree { strict_priority(), {wfq(), {token_bucket(), {wfq()} } }
   table find_path { … }
   apply {
        find_path.apply();
        // apply the scheduling algorithm defined by the tree
        myTree.apply(leaf_node);
   }
}
```



Towards open reference platforms

Software platform: P4 toolchain for BMv2 simulation



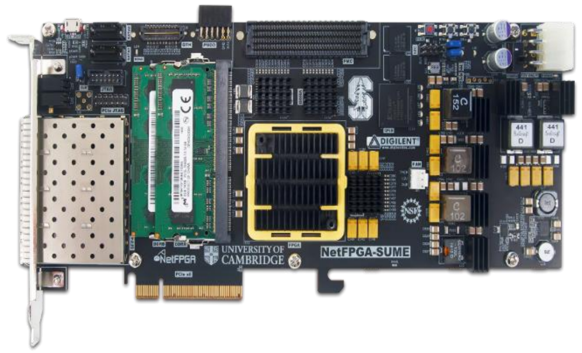
Hardware platform: NetFPGA (= Networked FPGA)

- Line-rate, flexible, open networking hardware for teaching and research
- Begun in 2007 by Stanford and Xilinx Labs, now anchored at Cambridge
- NetFPGA systems deployed at over 150 institutions in over 40 countries

Four elements:

- Community: NetFPGA.org
- Low-cost board family
- Tools and reference designs
- Contributed projects

NetFPGA-1G-CML
4x1G ports

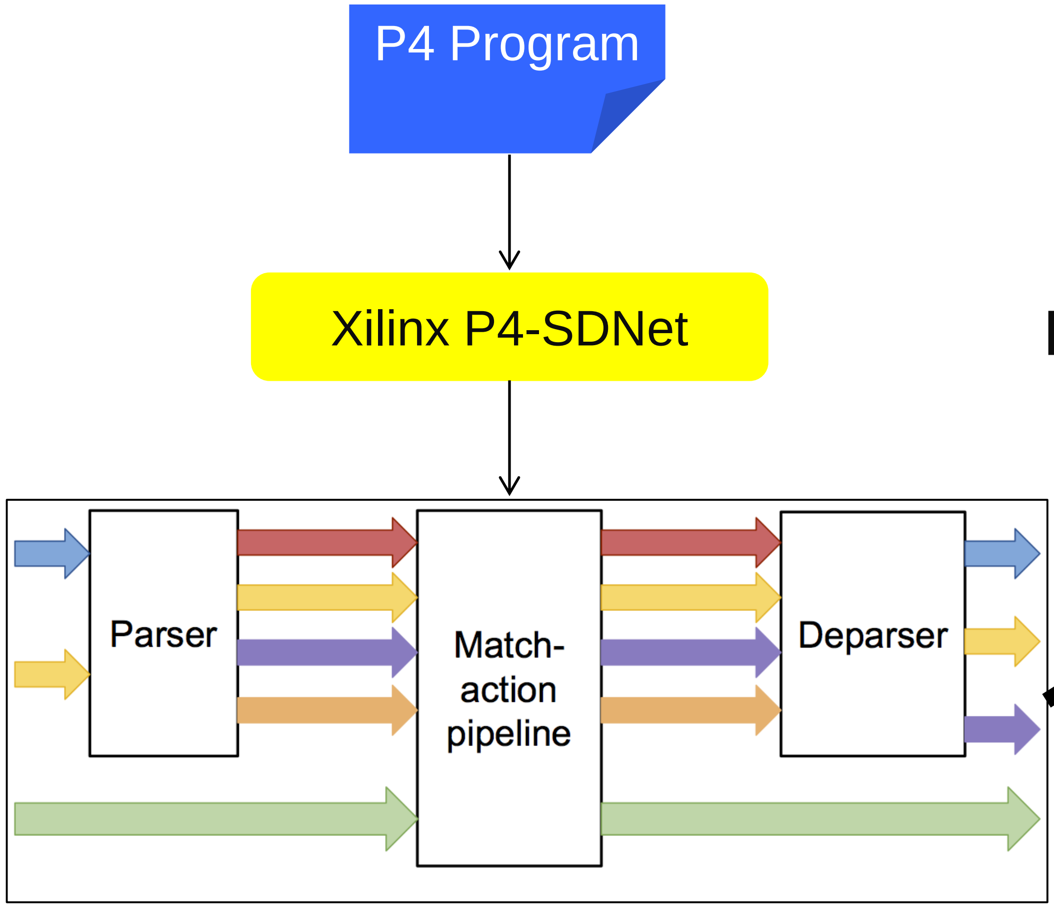


NetFPGA-SUME
4x10G ports

Hardware platform: P4→NetFPGA workflow

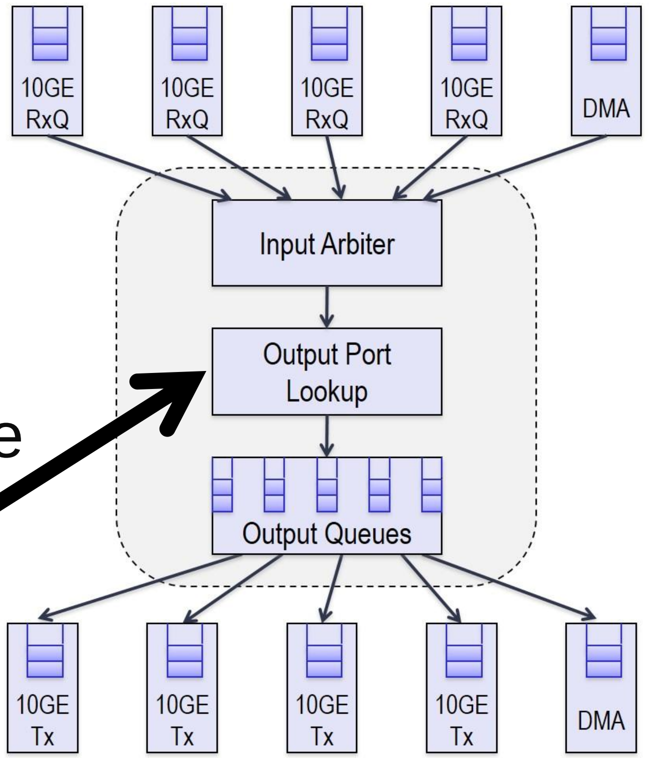
https://github.com/NetFPGA/P4-NetFPGA-public/wiki

See flier in your P4EU registration



Drop-in substitute

NetFPGA SUME reference switch design



4x10G Ethernet switch, with CPU slow path as 5th port

>> 5

XILINX.

Possible future P4 open reference platform collection

Two architecture types	NIC style (PNA)	Switch style (PSA)
with		
Two implementation types	Hardware (FPGA)	Software (simulation)

Conclusion

Research directions

> **Language: Extend coverage of P4**
>
> >> Programmable Traffic Management (MIT + NYU + Stanford + Xilinx Labs + P4.org)
>
> >> Programmable Target Architectures (Cornell + Stanford + VMWare Research + Xilinx Labs)

> **Infrastructure: Open source hardware reference platform for P4**
>
> >> Complement existing software reference platform
>
> >> Cover NIC-style architectures as well as switch-style architectures

> **Applications**
>
> >> Congestion control; In-band network telemetry
>
> >> In-network computing
>
> >> Programmable networking novelty
>
> >> … your ideas here

Call to action

- **Become a member of P4.org**
  - No fee, and simple membership agreement
  - Code and data under Apache 2.0 license
- **Participate in working groups, and their *ad hoc* sub-groups (e.g., PNA, PTM)**
  - Activities are open to all members
  - Anyone with a good idea can help shape the future of P4
- **Contribute to evolving open source provision**
  - Compiler (p4c) – common front end and mid ends, and target-specific backends
  - Software reference switch (bmv2) – and future open platforms
  - Control plane API (P4Runtime)
  - Tutorials
  - Documentation
  - Standard applications
  - New applications

The End