Disclaimer

- These are not slides from DTU (refer to p4.org)
- You can find the original slides at:
  - https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf
- We removed some of the slides and organized in several groups
- You are welcome to check the original slides for a wider perspective
- If you have doubts about:
  - What P4 is, why it is beneficial or the core concepts of P4: Refer to file "**1 - P4 Information.pdf**"
  - What the core elements of P4 programming are (controls, externs, tables, data types), how to program a target, how the architecture refers to a P4 program and then: Refer to file "**2 - Basics on P4 programming.pdf**"
  - What P4 runtime is, how it related to P4 overall and what you can do with it: Refer to file "**3 - P4 Runtime.pdf**"

P4 Language Tutorial

Goals

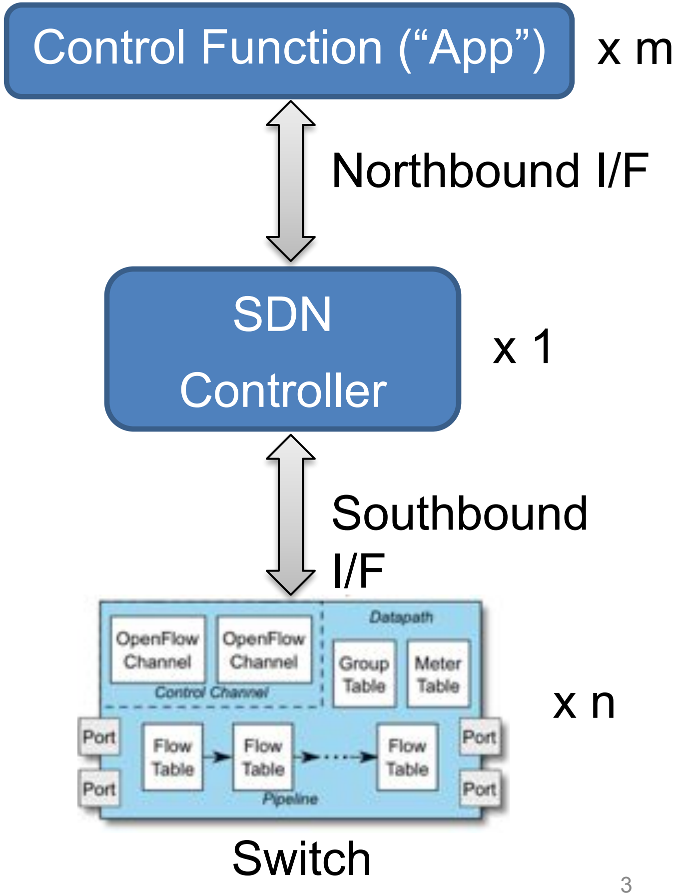
- **Learn P4 Language**
  - Traditional applications
  - Novel applications
- **Learn P4 software tools**
  - P4 Compiler
  - BMv2
  - P4Runtime
- **Learn about P4 hardware targets**
  - mini-workshop featuring solutions by Barefoot, Netronome, Netcope and NetFPGA.
- **Networking (the other kind)**
- **Have fun!**

What is Data Plane Programming?

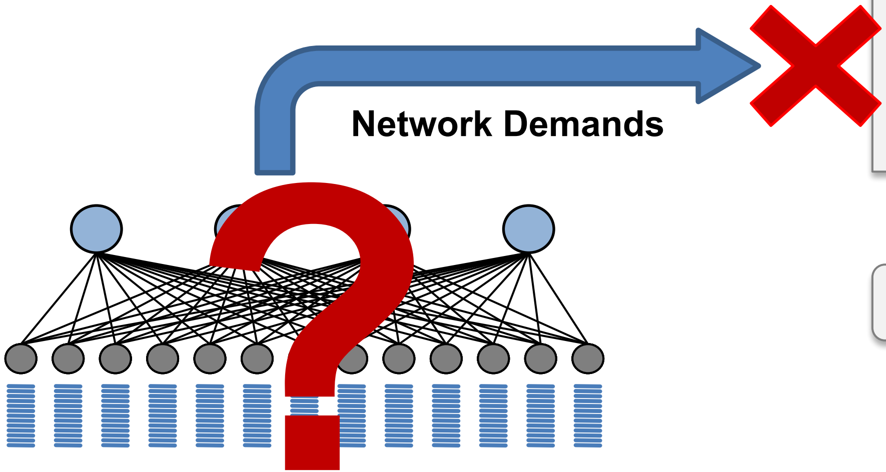
- **Why program the Data Plane?**

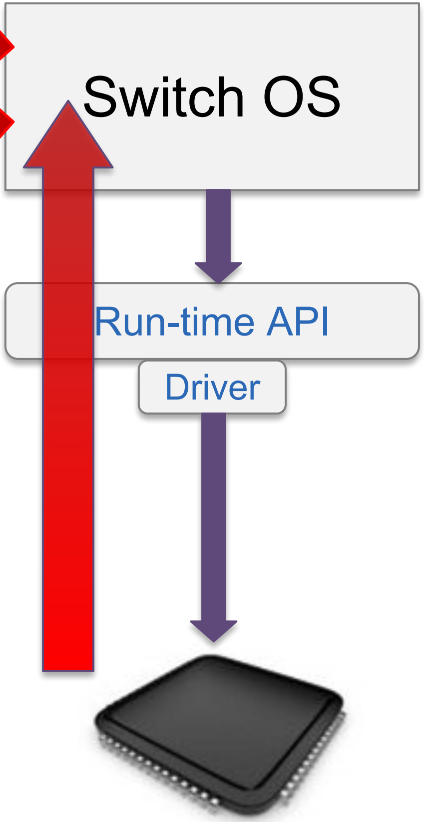
Software Defined Networking: Logically Centralized Control

- **Main contributions**
  - OpenFlow = standardized *model*
    - match/action abstraction
  - OpenFlow = standardized *protocol* to interact with switch
    - download flow table entries, query statistics, etc.
  - *Concept* of *logically* centralized control via a single entity ("SDN controller")
    - Simplifies control plane – e.g. compute optimal paths at one location (controller), vs. waiting for distributed routing algorithms to converge
- **Issues**
  - Data-plane protocol evolution requires changes to standards (12 → 40 OpenFlow match types)
  - Limited interoperability between vendors => southbound I/F differences handled at controller (OpenFlow / netconf / JSON / XML variants)



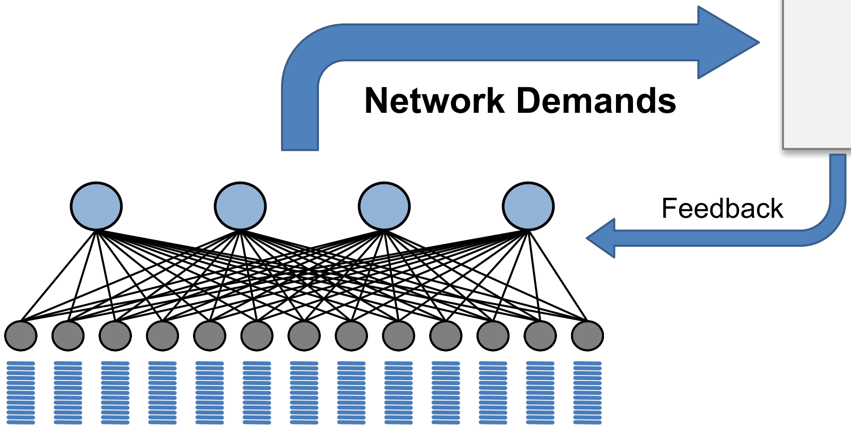
Status Quo: Bottom-up design

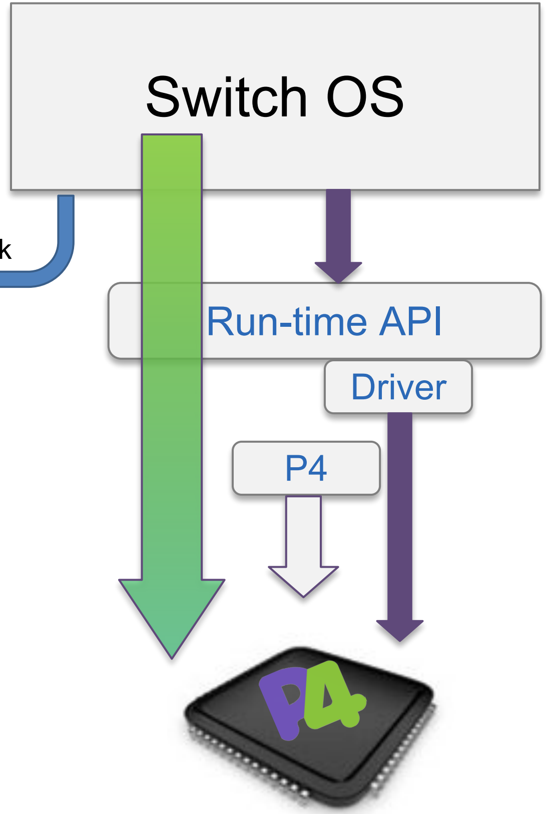




“This is *how I know* to process packets”
(i.e. the ASIC datasheet makes the rules)

A Better Approach: Top-down design





“This is *how I want* the network to behave and how to switch packets…”
(the user / controller makes the rules)

P4 Programmable Device

Benefits of Data Plane Programmability

- **New Features** – Add new protocols
- **Reduce complexity** – Remove unused protocols
- **Efficient use of resources** – flexible use of tables
- **Greater visibility** – New diagnostic techniques, telemetry, etc.
- **SW style development** – rapid design cycle, fast innovation, fix data plane bugs in the field
- **You keep your own ideas**

Think programming rather than protocols…

Programmable Network Devices

- **PISA: Flexible Match+Action ASICs**
  - Intel Flexpipe, Cisco Doppler, Cavium (Xpliant), Barefoot Tofino, …
- **NPU**
  - EZchip, Netronome, …
- **CPU**
  - Open Vswitch, eBPF, DPDK, VPP…
- **FPGA**
  - Xilinx, Altera, …

These devices let us tell them how to process packets.

What can you do with P4?

- **Layer 4 Load Balancer – SilkRoad[1]**
- **Low Latency Congestion Control – NDP[2]**
- **In-band Network Telemetry – INT[3]**
- **Fast In-Network cache for key-value stores – NetCache[4]**
- **Consensus at network speed – NetPaxos[5]**
- **Aggregation for MapReduce Applications [6]**
- **… and much more**

[1] Miao, Rui, et al. "SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs." SIGCOMM, 2017.
[2] Handley, Mark, et al. "Re-architecting datacenter networks and stacks for low latency and high performance." SIGCOMM, 2017.
[4] Kim, Changhoon, et al. "In-band network telemetry via programmable dataplanes." SIGCOMM. 2015.
[3] Xin Jin et al. "NetCache: Balancing Key-Value Stores with Fast In-Network Caching." To appear at SOSP 2017
[5] Dang, Huynh Tu, et al. "NetPaxos: Consensus at network speed." SIGCOMM, 2015.
[6] Sapio, Amedeo, et al. "In-Network Computation is a Dumb Idea Whose Time Has Come." *Hot Topics in Networks*. ACM, 2017.

Brief History and Trivia

- May 2013: Initial idea and the name “P4”
- July 2014: First paper (SIGCOMM CCR)
- Aug 2014: First $P4_{14}$ Draft Specification (v0.9.8)
- Sep 2014: $P4_{14}$ Specification released (v1.0.0)
- Jan 2015: $P4_{14}$ v1.0.1
- Mar 2015: $P4_{14}$ v1.0.2
- Nov 2016: $P4_{14}$ v1.0.3
- May 2017: $P4_{14}$ v1.0.4

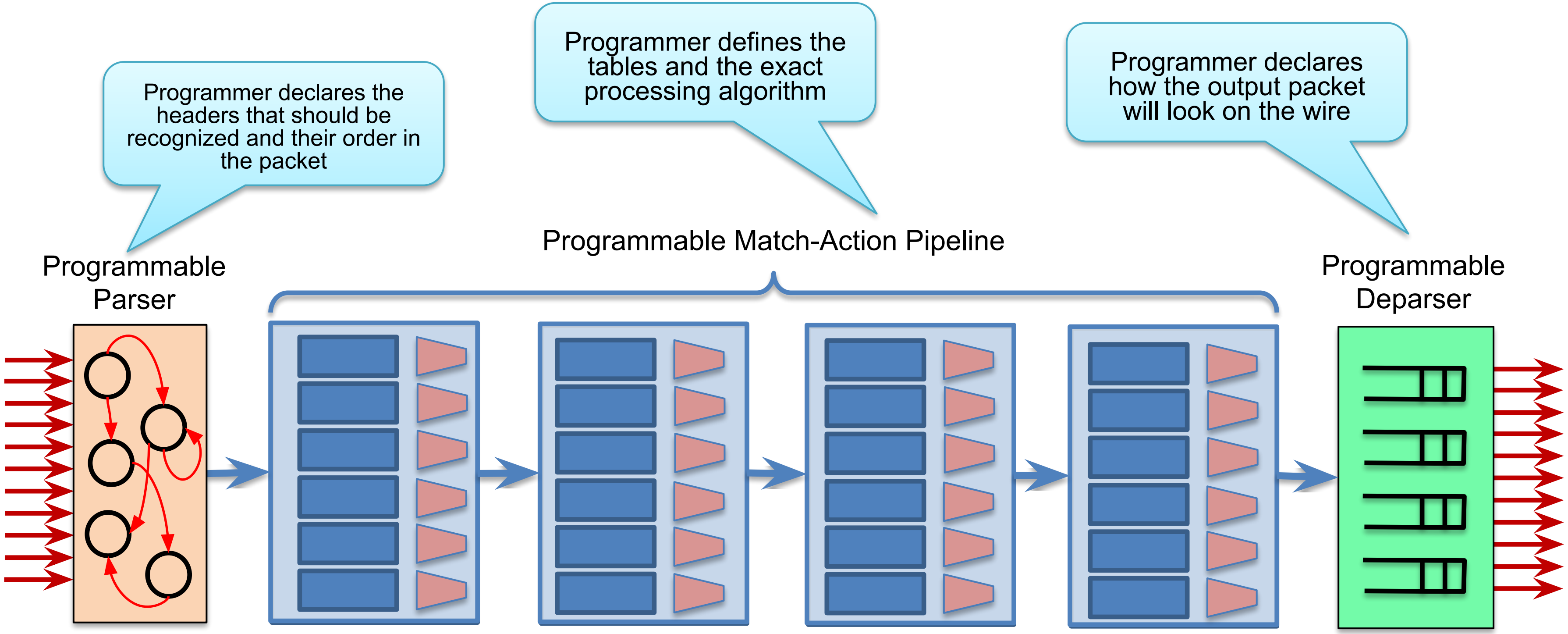
- Apr 2016: $P4_{16}$ – first commits
- Dec 2016: First $P4_{16}$ Draft Specification
- May 2017: $P4_{16}$ Specification released

Why $P4_{16}$?

- **Clearly defined semantics**
  - You can describe what your data plane program is doing
- **Expressive**
  - Supports a wide range of architectures through standard methodology
- **High-level, Target-independent**
  - Uses conventional constructs
  - Compiler manages the resources and deals with the hardware
- **Type-safe**
  - Enforces good software design practices and eliminates “stupid” bugs
- **Agility**
  - High-speed networking devices become as flexible as any software
- **Insight**
  - Freely mixing packet headers and intermediate results

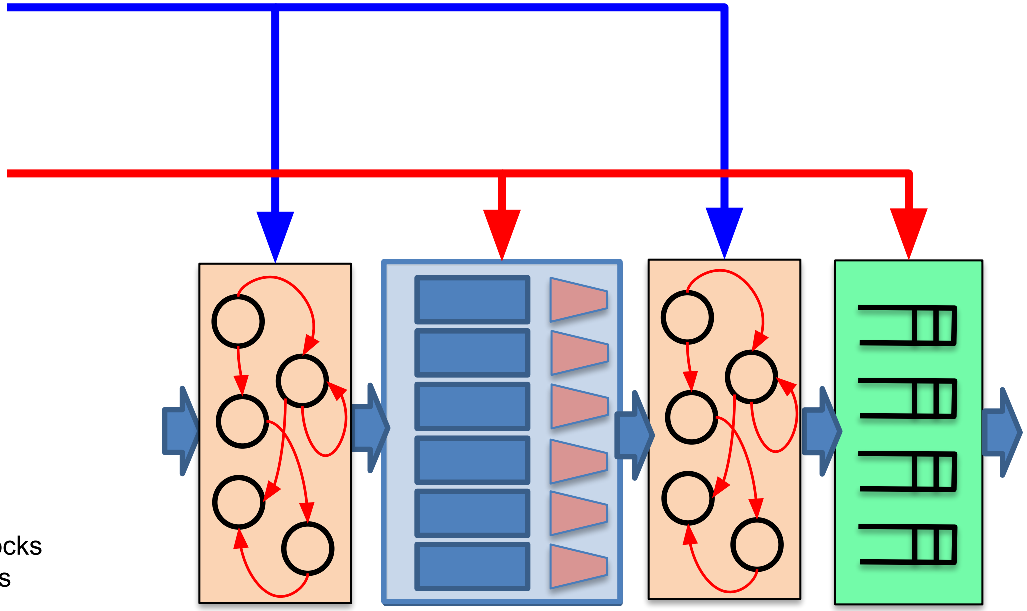
P4_16 Data Plane Model

PISA: Protocol-Independent Switch Architecture



$P4_{16}$ Language Elements

Parsers	State machine, bitfield extraction
Controls	Tables, Actions, control flow statements
Expressions	Basic operations and operators
Data Types	Bistrings, headers, structures, arrays
Architecture Description	Programmable blocks and their interfaces
Extern Libraries	Support for specialized components



P4_16 Approach

Term	Explanation
P4 Target	An embodiment of a specific hardware implementation
P4 Architecture	Provides an interface to program a target via some set of P4-programmable components, externs, fixed components