Brief overview of P4

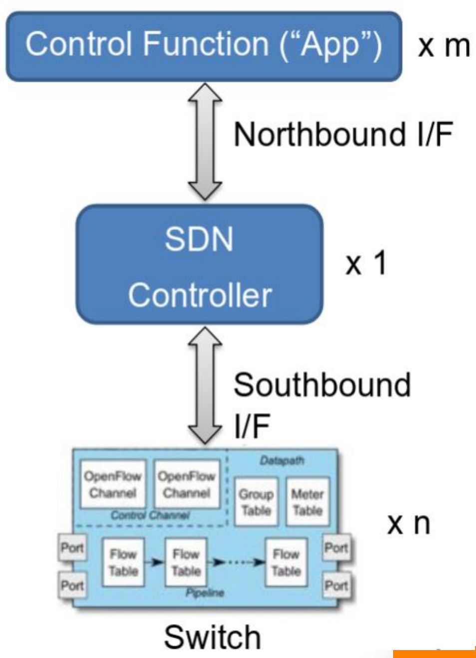
Levente Csikor (BME-TMIT)

SINCE 1992 HSNLab

ERICSSON

What is Data Plane Programming and WHY?

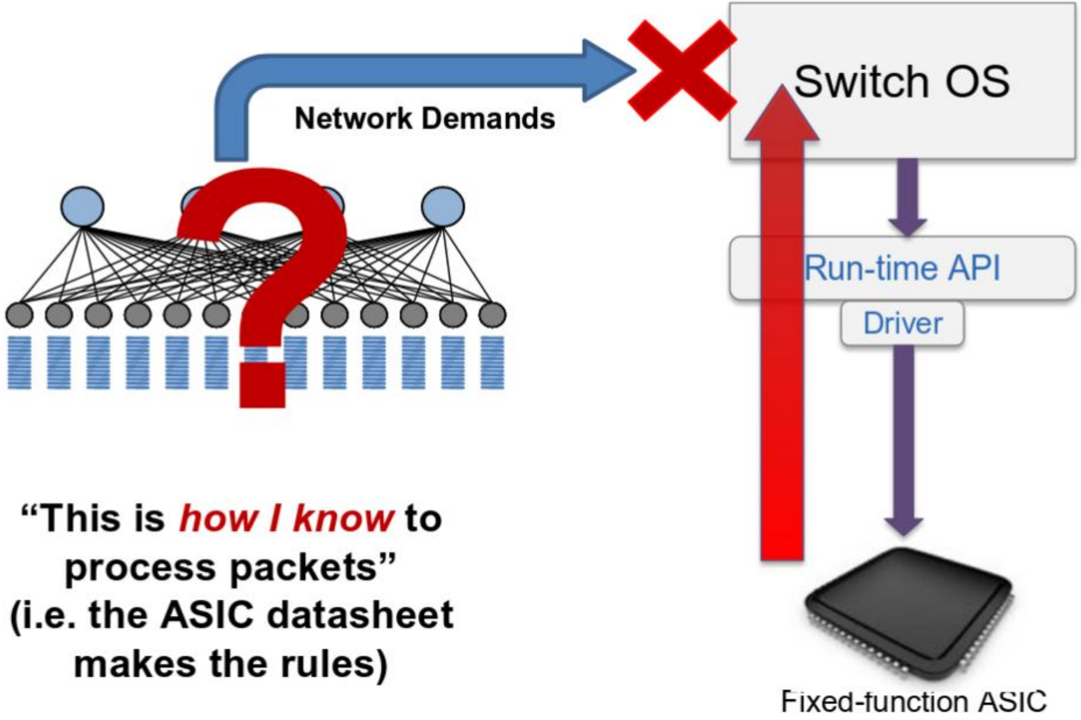
- SDN – Logically Centralized Control
- OpenFlow
  - Standardized model
    - match and action abstraction
  - Standardized protocol to interact with switch
    - Flow table population, query statistics, etc.
- Single Pane of Glass for Control



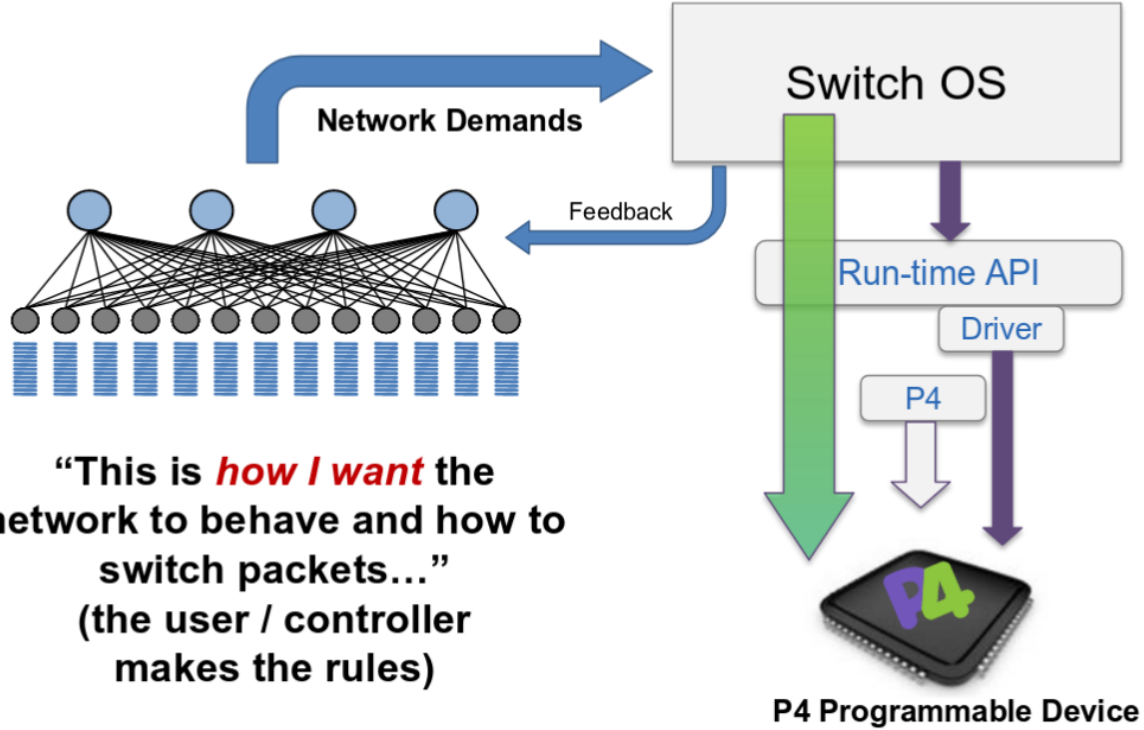
ISSUES with OpenFlow

- Data-plane protocol evolution needs continuous update to latest standards (12 -> 40 header in OF)
- Limited interoperability between vendors
  - Supported OF version (v1.1 vs. v1.3)
  - Missing features (incomplete implementation)
  - Performance issues (HW vs. SW implementation)
  - ASICs requires certain order of pipeline elements to be efficient

Current Networking: Bottom-up Design



P4: Top-down Design



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

So why should I program my data plane?

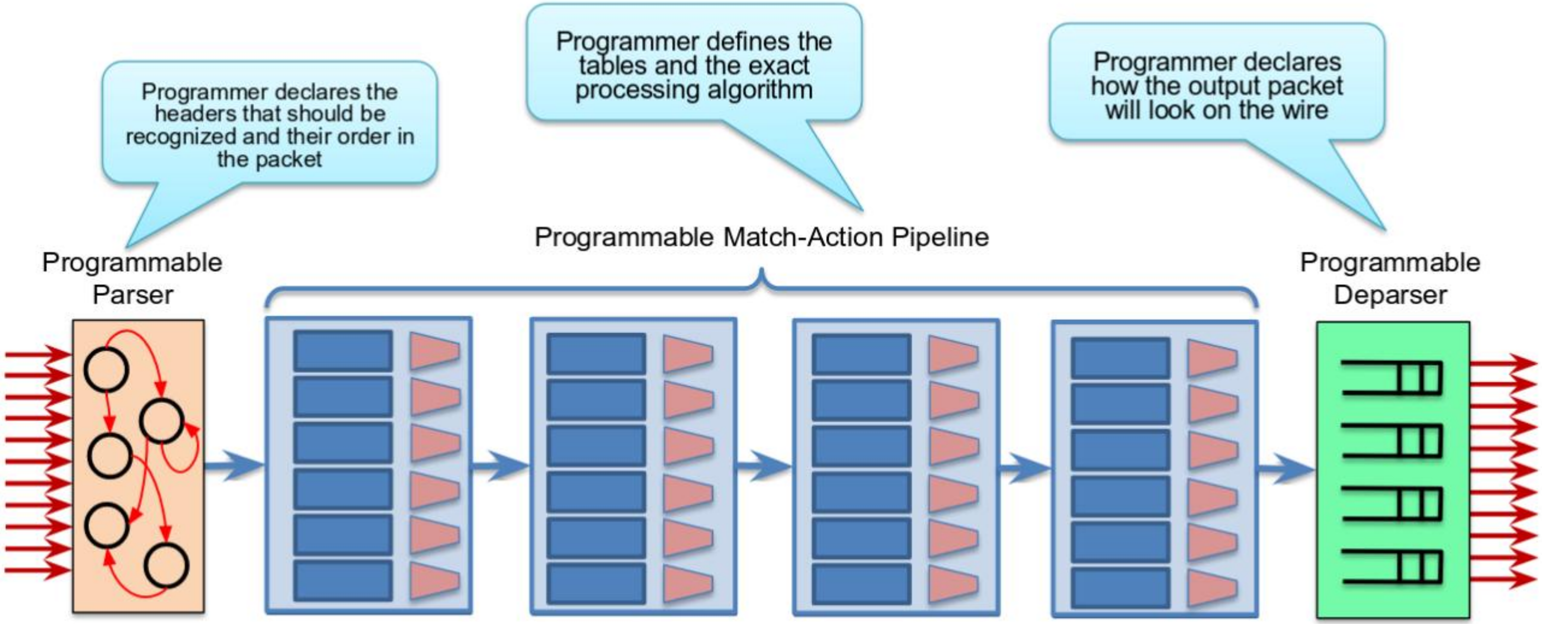
- If I (can re-) program the “ASIC”, I can easily:
  - Add new features
    - Implement my own protocol (based on my own new headers/labels)
    - Keep your ideas in-house!
  - Reduce complexity and dead code elimination
    - My program only contains the necessary elements
    - My program has no unused components
  - Efficient use of resources
    - If I will never need VLAN, why should the switch know it?
  - Greater visibility
    - New diagnostic techniques (Let’s make switches smart again!)
  - SW style development for the data plane
    - Faster innovation than in OF

What can you do with P4 (in Academia) ?

- 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]

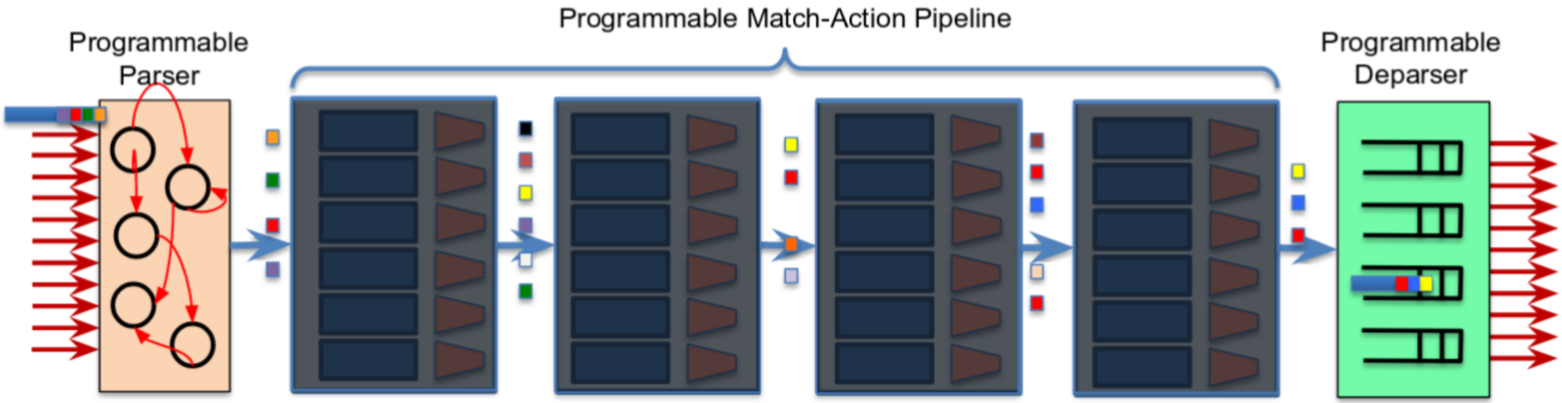
[1] Miao, Rui, et al. *"SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs*." SIGCOMM, 2017.
[2] Handley, Mark, et al. *"Re-architecting datacenter networks and stacks for low latency and high performance."* SIGCOMM, 2017.
[4] Kim, Changhoon, et al. *"In-band network telemetry via programmable dataplanes."* SIGCOMM. 2015.
[3] Xin Jin et al. *"NetCache: Balancing Key-Value Stores with Fast In-Network Caching."* , 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, 2017.

PISA: Protocol-Independent Switch Architecure



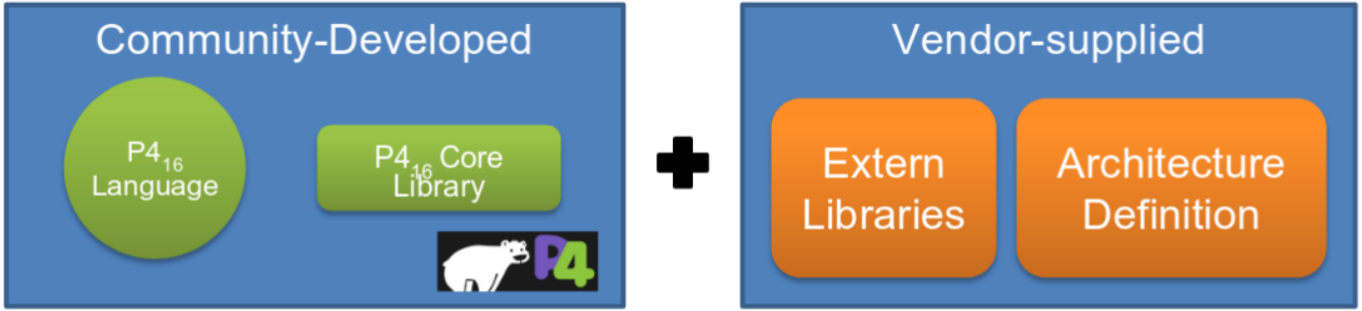
PISA: Protocol-Independent Switch Architecture

- Packet is parsed into individual headers
- Headers (and intermediate results) can be used for match/action
- Headers can be modified, removed, added (even completely new)
- Packet is de-parsed (serialized)

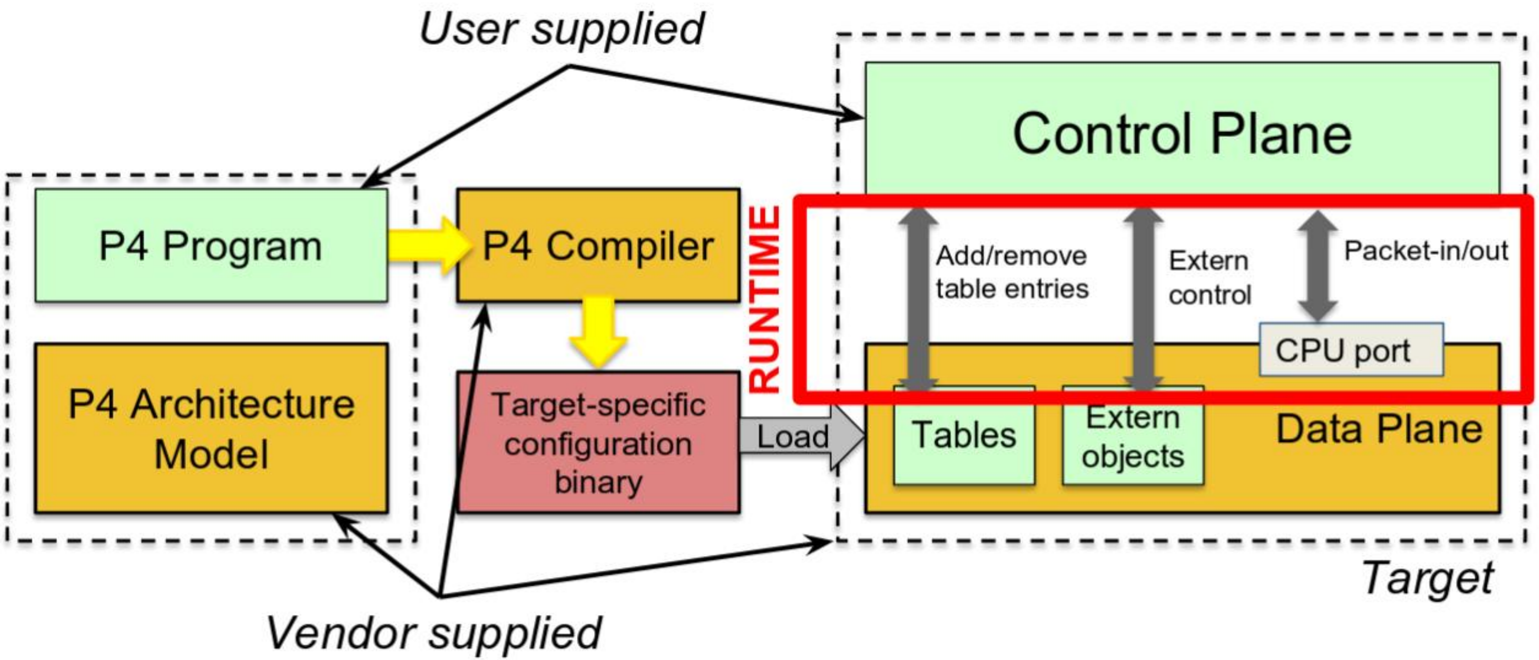


The new $P4_{16}$ approach

- $P4_{14}$ was the predecessor (kind of a PoC implementation)
- $P4_{16}$:
  - P4 Core library (supported by all P4-compatible appliance)
  - P4 Architecture (additional features, extern functions – vendor supplied)



Birds-eye view of P4 programming

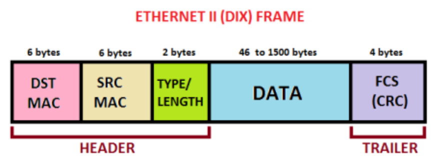


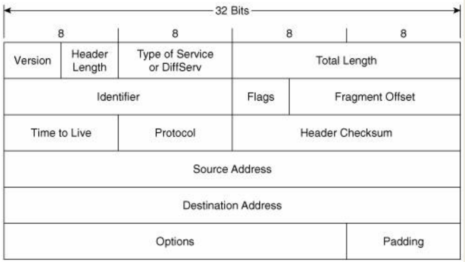
P4$_{16}$ program template

```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
  ethernet_t    ethernet;
  ipv4_t        ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
         out headers hdr,
         inout metadata meta,
         inout standard_metadata_t smeta) {
  ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {
  ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  ...
}
```

```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
  ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
  ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                   inout metadata meta) {
  ...
}
/* SWITCH */
V1Switch(
  MyParser(),
  MyVerifyChecksum(),
  MyIngress(),
  MyEgress(),
  MyComputeChecksum(),
  MyDeparser()
) main;
```

P4₁₆: High-level abstraction with low-level types






```
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;
header ethernet_t {
  macAddr_t dstAddr;
  macAddr_t srcAddr;
  bit<16>   etherType;
}
header ipv4_t {
  bit<4>    version;
  bit<4>    ihl;
  bit<8>    diffserv;
  bit<16>   totalLen;
  bit<16>   identification;
  bit<3>    flags;
  bit<13>   fragOffset;
  bit<8>    ttl;
  bit<8>    protocol;
  bit<16>   hdrChecksum;
  ip4Addr_t srcAddr;
  ip4Addr_t dstAddr;
}
```

Basic types:

- **bit<n>:** unsigned integer (bitstring) of size *n*
- **bit**: is the same as bit<1>
- **int<n>**: signed integer of size n (>=2)
- **varbit<n>**: variable-length bitstring: special type with limited functions (no matching, no comparing to other types, etc.)

Header types:

- Ordered collection of Basic types
- Byte-aligned
- Can be valid on invalid

Typedef:

- Alternative name for a frequently used type

P4₁₆: High-level abstraction with low-level types

```
/* Architecture */
struct standard_metadata_t {
  bit<9>  ingress_port;
  bit<9>  egress_spec;
  bit<9>  egress_port;
  bit<32> clone_spec;
  bit<32> instance_type;
  bit<1>  drop;
  bit<16> recirculate_port;
  bit<32> packet_length;
  ...
}

/* User program */
struct metadata {
  ...
}
struct headers {
  ethernet_t    ethernet;
  ipv4_t        ipv4;
}
```

Other useful types:

- **struct:** unordered collection of basic types or structs
- **Header stack**: Array of headers
- **Header union**: one of several headers

$P4_{16}$: Parsers

```
state start {
  transition parse_ethernet;
}

state parse_ethernet {
  packet.extract(hdr.ethernet);
  transition select(hdr.ethernet.etherType) {
    0x800: parse_ipv4;
    default: accept;
  }
}
```

States like function definitions in any programming language

packet.extract(header type): extract bits from the packet according to our defined header

Select like if-else/case statements in any programming language

Accept: “end” of parsing -> important header fields are extracted for further processing

P4₁₆: Tables

```
table ipv4_lpm {
    key = {
      hdr.ipv4.dstAddr: lpm;
    }
    actions = {
      ipv4_forward;
      drop;
      NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

- Defines the format of the table
  - Key fields (+ match kind: exact, ternary, lpm)
  - Actions
  - Action data
- Performs lookup
- Executes the chosen action
- Note: Control plane populates the table

Key	Action	Action Data
10.0.1.1/32	ipv4_forward	dstAddr=00:00:00:00:01:01 port=1
10.0.1.2/32	drop	
*`	NoAction	

P4$_{16}$: Tables

```
/* core.p4 */
action NoAction() {
}

/* basic.p4 */
action drop() {
  mark_to_drop();
}

/* basic.p4 */
action ipv4_forward(macAddr_t dstAddr,
                    bit<9> port) {
  ...
}
```

- Actions can have two different parameters
  - Directional (from the data plane)
  - Directionless (from the control plane)
- Actions used in tables:
  - Typically use directionless parameters
  - May sometimes use directional parameters too

```
action ipv4_forward(macAddr_t dstAddr, bit<9> port)
{
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl – 1;
    standard_metadata.egress_spec = port;  //this is how define ouput port
}
```

$P4_{16}$: Applying tables and de-parsing

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
  table ipv4_lpm {
    ...
  }
  apply {
     ...
     ipv4_lpm.apply();
     ...
  }
}
```

```
control DeparserImpl(packet_out packet,
                     in headers hdr) {
  apply {
    ...
    packet.emit(hdr.ethernet);
    ...
  }
}
```

$P4_{16}$ Summary

- Switches become smart again (OF made them dumb)
- Additional (custom) features can be implemented and deployed
- No need for firmware upgrade (for in-house development)
- Limited instruction set (e.g., no *loops*, no *double/float numbers*, etc.)
  - But everything else is done in **wirespeed** (tens or hundred Gbps)
- Supported devices
  - Barefoot Tofino chip (P4 ASIC)
  - Edge core Wedge 100-32X
  - Costs tens of thousands of dollars
  - Not really Out-of-the-Box deployment (in its infancy now)
    - Licences, simulators, special IDEs, months of legal agreement processes

Thank you

Levente Csikor (csikor@tmit.bme.hu)

2018.11.05.

Credits:
Most of the content is adapted from P4 language consortium (p4.org)

SINCE 1992 HSNLab ERICSSON