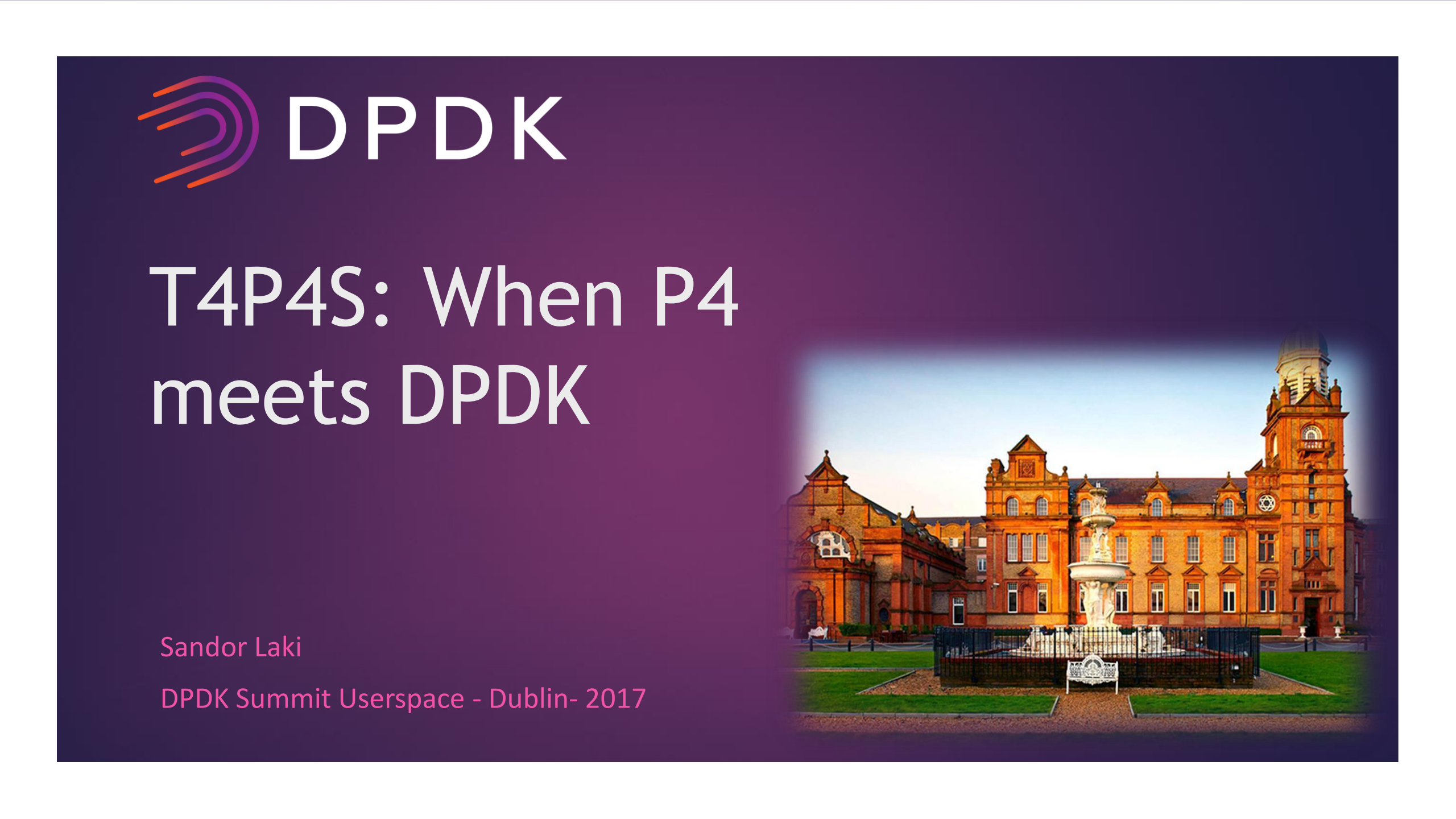


T4P4S: When P4 meets DPDK

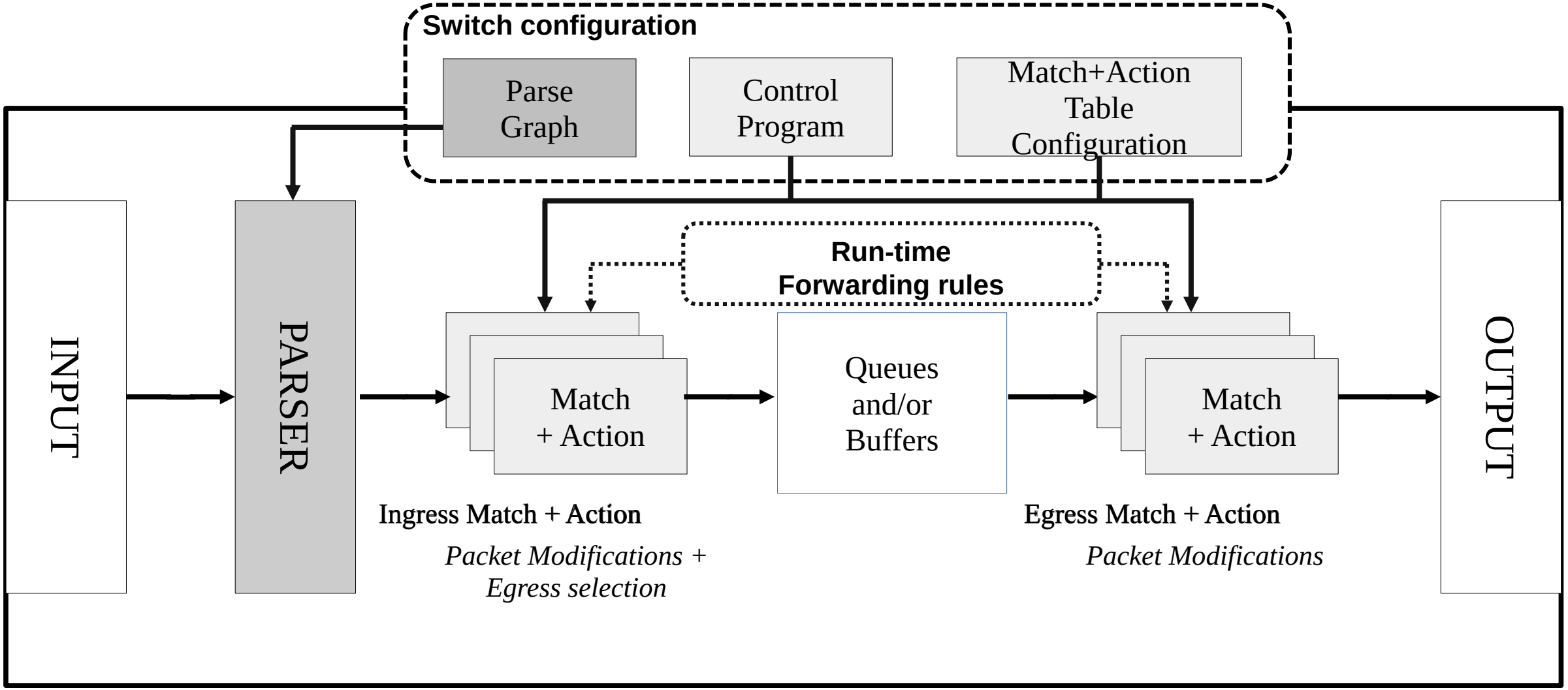
Sandor Laki

DPDK Summit Userspace - Dublin- 2017

What is P4?

- Domain specific language for programming any kind of data planes
  - Flexible, protocol and target independent
  - Re-configurable
  - Intuitive abstractions
- Open source project
  - Free membership
  - Strong industrial interest (56 industrial partners + 15 universities)
- Intensive activity
  - Workshops, extended language specification, network function examples in P4
  - Evolving language: P4-14 and P4-16 variants
  - Open source tools: behavioral compiler, reusable frontend, JSON representation, switch API

P4 model (P4-14)



A P4-14 Example


```
header_type ethernet_t {
        fields {
                dstAddr : 48;
                srcAddr : 48;
                etherType : 16;
        }
}
```

```
parser parse_ethernet {
        extract(ethernet);
        return select(latest.etherType) {
                0x8100 : parse_vlan;
                0x800 : parse_ipv4;
                0x86DD : parse_ipv6;
        }
}
```

```
table ipv4_lpm {
        reads {
                ipv4.dstAddr : lpm;
        }
        actions {
                set_next_hop;
                drop;
        }
}
```

```
action set_next_hop(nhop_ipv4_addr, port)  {
        modify_field(metadata.nhop_ipv4_addr, nhop_ipv4_addr);
        modify_field(standard_metadata.egress_port, port);
        add_to_field(ipv4.ttl, -1);
}
```

A P4-14 Example


```
header_type ethernet_t {
        fields {
                dstAddr : 48;
                srcAddr : 48;
                etherType : 16;
        }
}
```

```
table ipv4_lpm {
        reads {
                ipv4.dstAddr : lpm;
        }
        actions {
                set_next_hop;
                drop;
        }
}
```

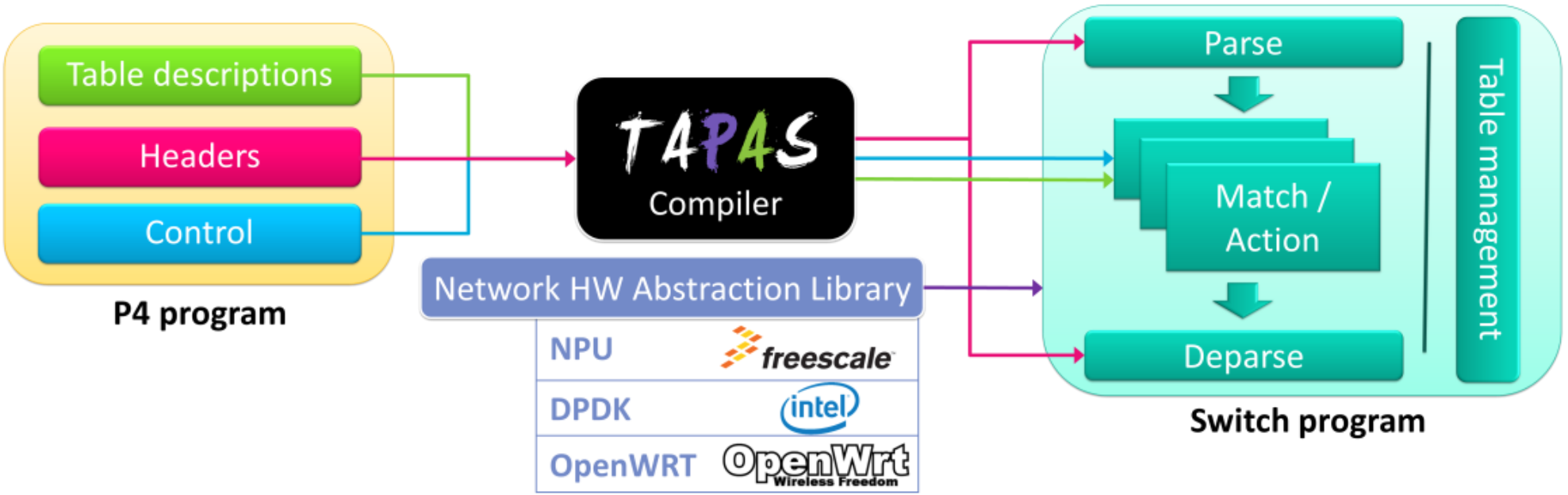
```
control ingress {
    apply(l2_table);
    if (valid(ipv4)) {
        apply(port_mapping);

        apply(bd);

        apply(ipv4_fib) {
            on_miss {
                apply(ipv4_fib_lpm);
            }
        }

        apply(nexthop);
    }
}
```

```
_ethernet {
thernet);
ect(latest.etherType) {
00 : parse_vlan;
0 : parse_ipv4;
DD : parse_ipv6;
```

```
_addr, port)  {
op_ipv4_addr, nhop_ipv4_addr);
adata.egress_port, port);
```

Goals of T4P4S

- Extended data plane programmability
  - P4 code as a high level abstraction
- Support of different hardware targets
  - CPUs, NPUs, FPGA, etc.
- Create a compiler that separates hardware
  dependent and independent parts
  - Easily retargetable P4 compiler



Multi-target Compiler Architecture for P4

DPDK

1. Hardware-independent „Core”

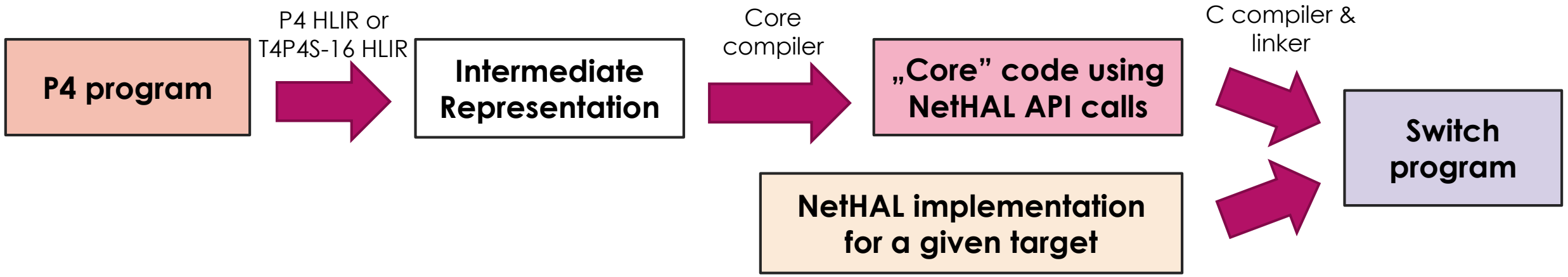
- Using an Intermediate Representation (IR)
- Compiling IR to a hardware independent C code with NetHAL calls

2. Hardware-dependent „Network Hardware Abstraction Layer” (NetHAL)

- Implementing well primitives that fulfill the requirements of most hardware
- A static and thin library
- Written by a hardware expert (currently available for DPDK and ODP)

3. Switch program

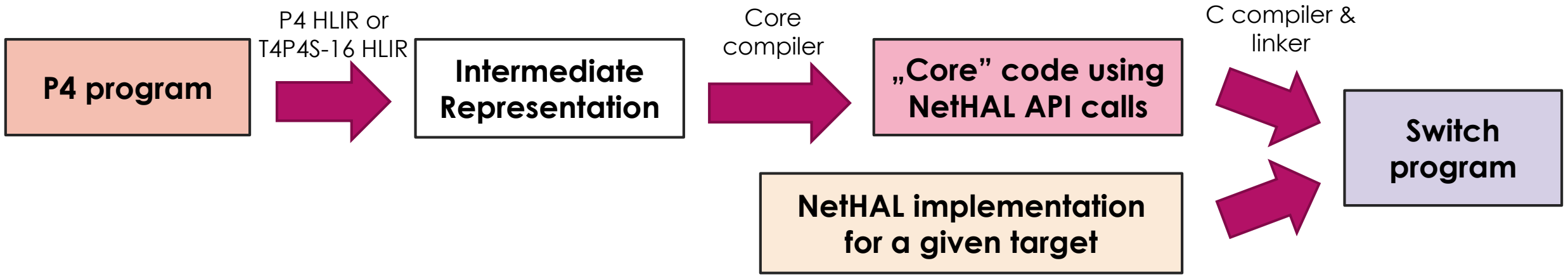
- Compiled from the hardware-independent C code of the „Core” and the target-specific HAL
- Resulting in a hardware dependent switch program



Multi-target Compiler Architecture for P4

DPDK

- **PROs**
  - Much simpler compiler
  - Modularity = better maintainability
  - Exchangeable NetHAL = re-targetable switch (without rewriting a single line of code)
  - NetHAL is not affected by changes in the P4 program
- **CONs**
  - Potentially lower performance
  - Difficulties with protocol/hardware-dependent optimization
  - Communication overhead between the components (C function calls)
  - Too general vs too detailed NetHAL API



The „core”

Run to completion model

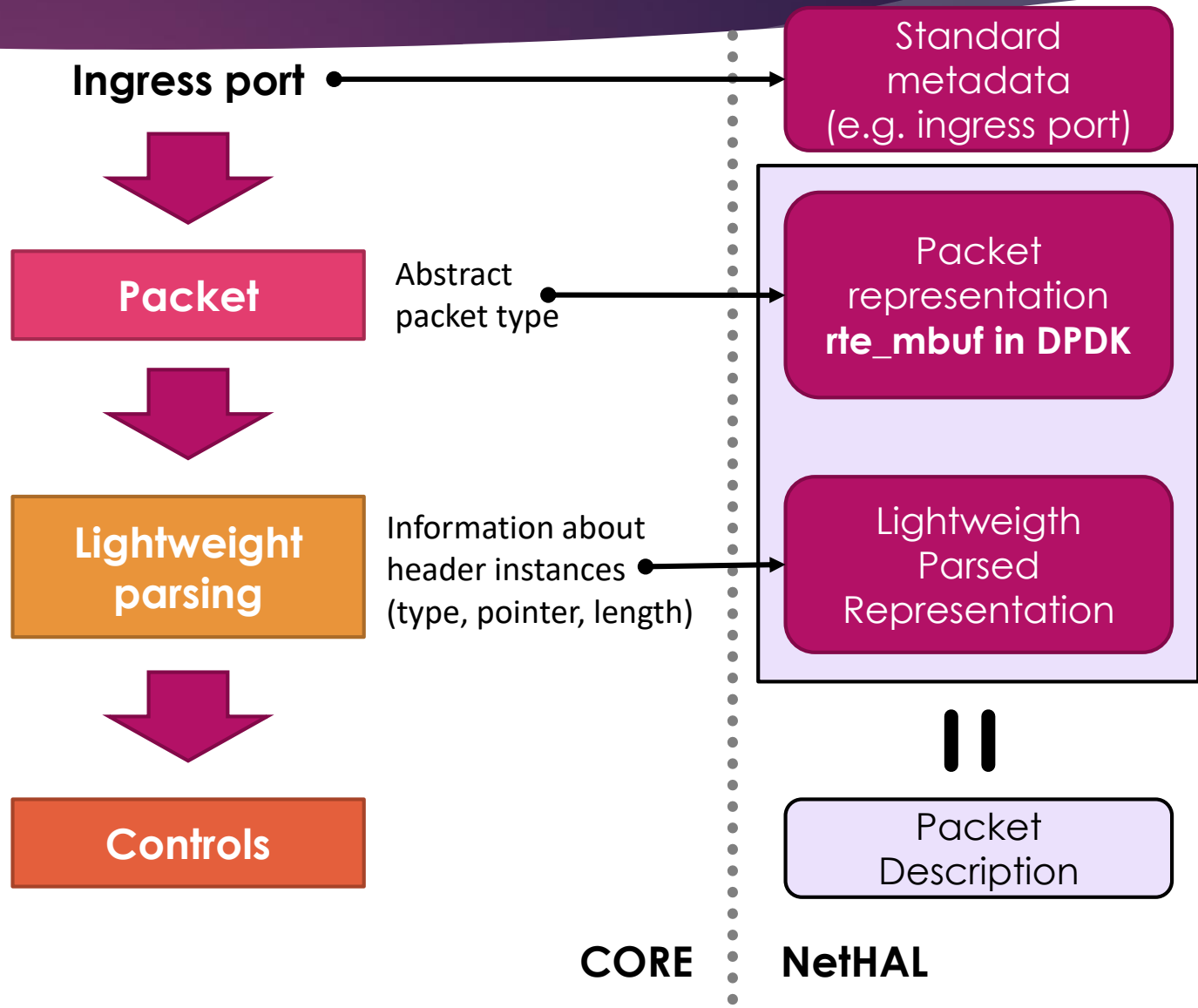
- Plans to move to a pipeline model

The core implements

- Packet „parsing”
- Control programs
- Actions
- Key calculations for lookup tables

Packet parsing

- Lightweight Parsed Representation
- Determining the positions and types of headers in the packet
- No "real" parsing or field extraction
  - lazy evaluation



The „core”

Run to completion model

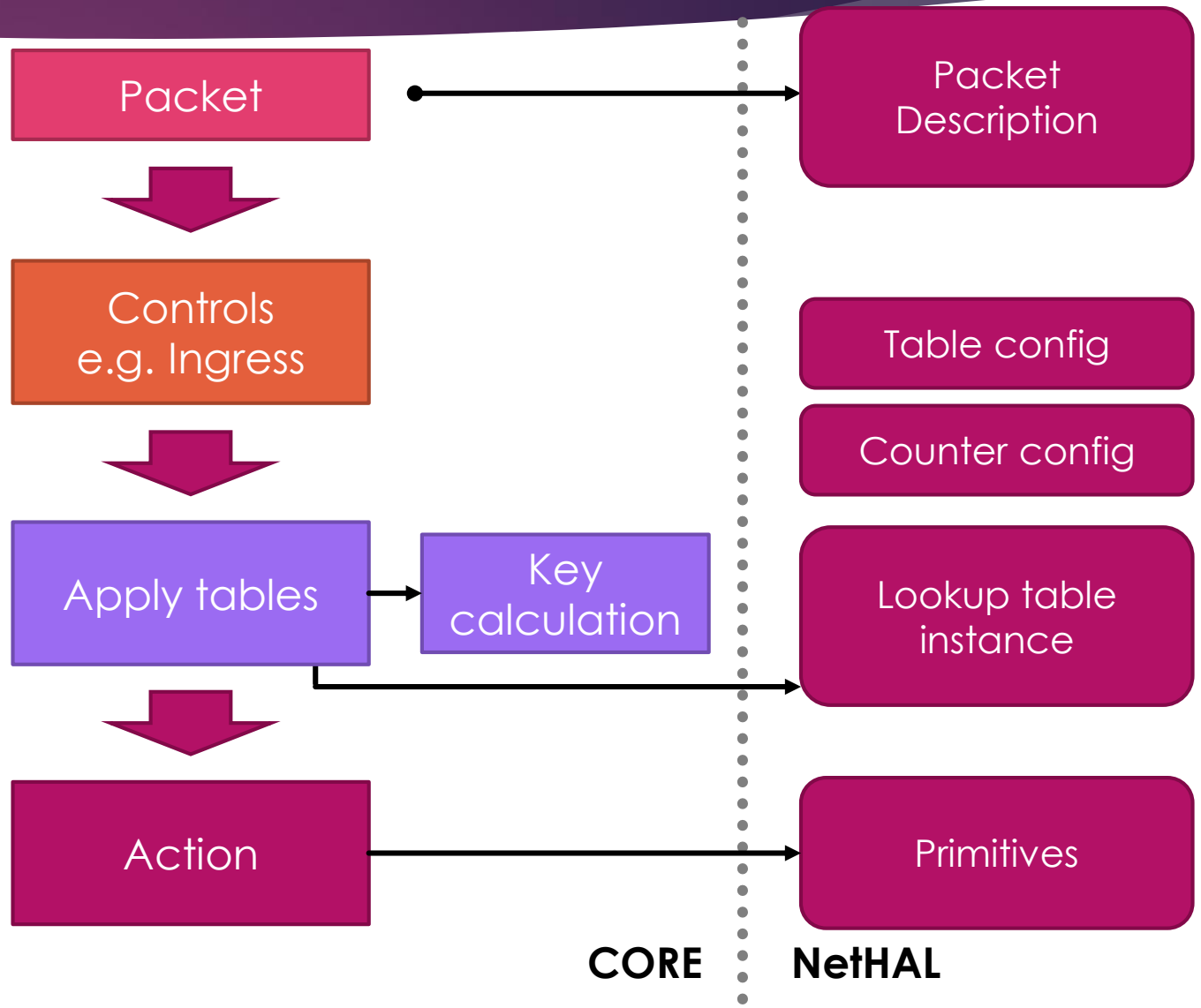
- Plans to move to a pipeline model

The core implements

- Packet „parsing”
- Control programs
- Actions
- Key calculations for lookup tables

Controls and actions

- Controls and actions are translated to C functions
- Key calculation for lookup tables
- Fields are extracted when needed
- In-place field modifications



The „core”

```
void action_code_forward(
                packet_descriptor_t* pkt,
                lookup_table_t** tables ,
                struct action_forward_params parameters)

{// sugar@199

   MODIFY_INT32_BYTEBUF(
                pkt,
                field_instance_standard_metadata_egress_port,
                parameters.port,
                2)// sugar@61

}// sugar@207
```

- Key calculation for lookup tables
- Fields are extracted when needed
- In-place field modifications

Network Hardware Abstraction Library

Low-level generic C API

- For networking hardwares

Hardware specific implementations of

- States/settings (tables, counters, meters etc.)
- Related operations (table insert/delete/lookup, counter increment, etc.)
- Packet RX and TX operations
- Primitive actions (header-related + digests)
- Helpers for primitive actions (field-related)
  - Implemented as macros for performance reasons

Add and remove headers

add_header(packet_descriptor_t* p, header_reference_t h)
push(packet_descriptor_t* p, header_stack_t h)
remove_header(packet_descriptor_t* p, header_reference_t h)
pop(packet_descriptor_t* p, header_stack_t h)

Field modification & extraction

MODIFY_BYTEBUF_BYTEBUF(pd, dstfield, src, srclen)
MODIFY_INT32_BYTEBUF(pd, dstfield, src, srclen)
MODIFY_INT32_INT32(pd, dstfield, value32)
EXTRACT_INT32(pd, field, dst)

Table & counter operations

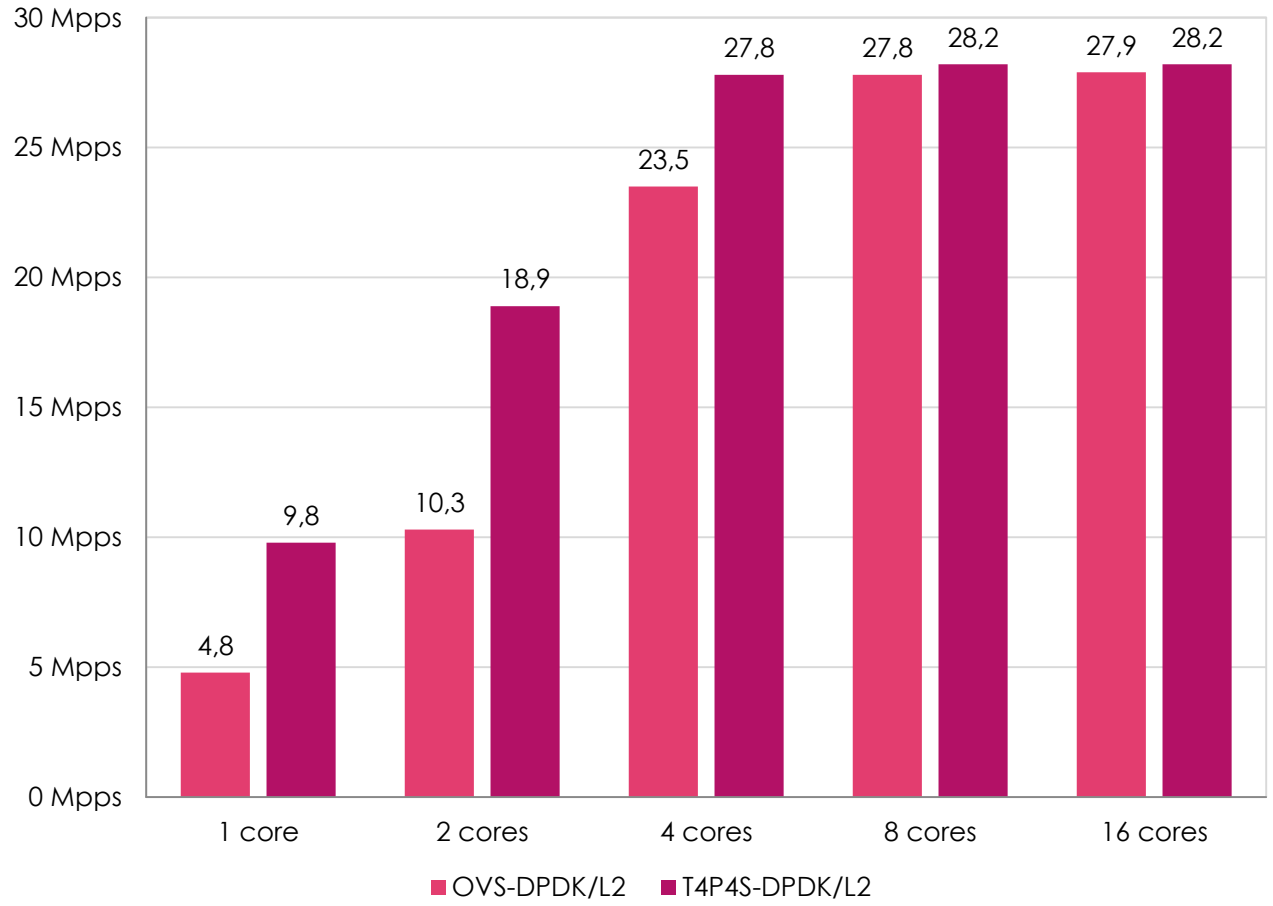
exact_lookup(lookup_table_t* t, uint8_t* key)
lpm_lookup(lookup_table_t* t, uint8_t* key)
ternary_lookup(lookup_table_t* t, uint8_t* key)
exact_add(lookup_table_t* t, uint8_t* key, uint8_t* value)
lpm_add(lookup_table_t* t, uint8_t* key, uint8_t depth, uint8_t* value)
ternary_add(lookup_table_t* t, uint8_t* key, uint8_t* mask, uint8_t* value)
increase_counter(int counterid, int index)
read_counter(int counterid, int index)

NetHAL for DPDK

- Our current NetHAL implementation is based on DPDK 17.05
  - Earlier versions are also supported
- Reuses LPM and HASH table implementations of DPDK
- Atomic integers for counters and meters
- Run to completion model
  - Each packet is processed by a dedicated lcore from parsing to egressing [current state]
  - Plans for pipeline models and table segmentation
- NUMA support
  - Lookup tables
    - Two instances of each table on each socket - lock-free solution
      - active/passive instances
  - Counter instances for each lcore on the corresponding socket
    - To avoid overhead by cache coherency

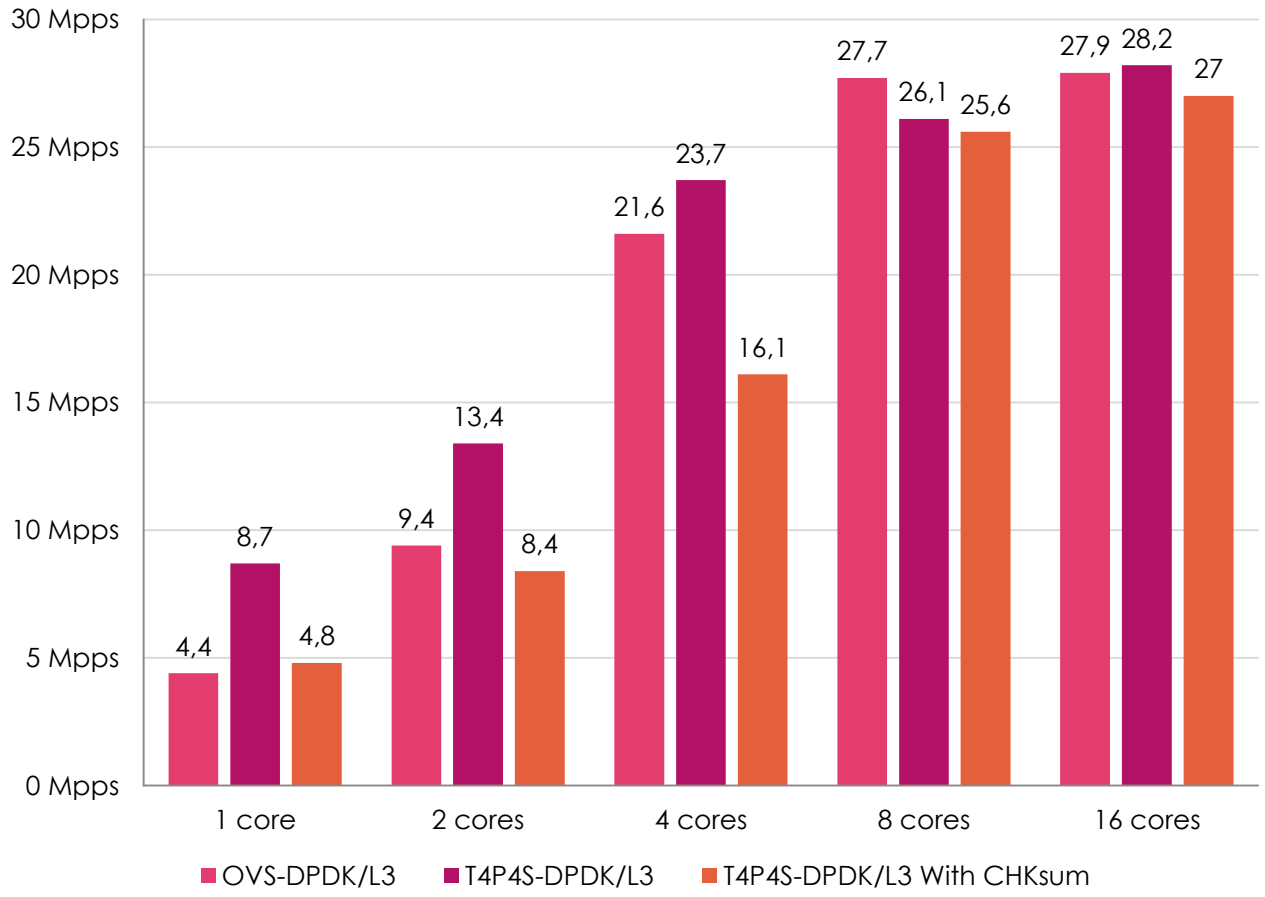
Evaluation - L2 forwarding

- L2 forwarding
  - Source mac learning
    - Two exact match tables: src mac + dst mac
- Testbed setup
  - Intel(R) Xeon(R) CPU E5-1660 v4 @ 8c 16t 3.20GHz, 8x8GB DDR4 SDRAM
  - Dual port 100 Gbps NIC
    - Mellanox MT27700 Family [ConnectX-4]
  - T4P4S performance is compared to OVS
    - Identical implementations in OpenFlow and P4
  - Pseudo random test traffic generated
    - A few hundred flows



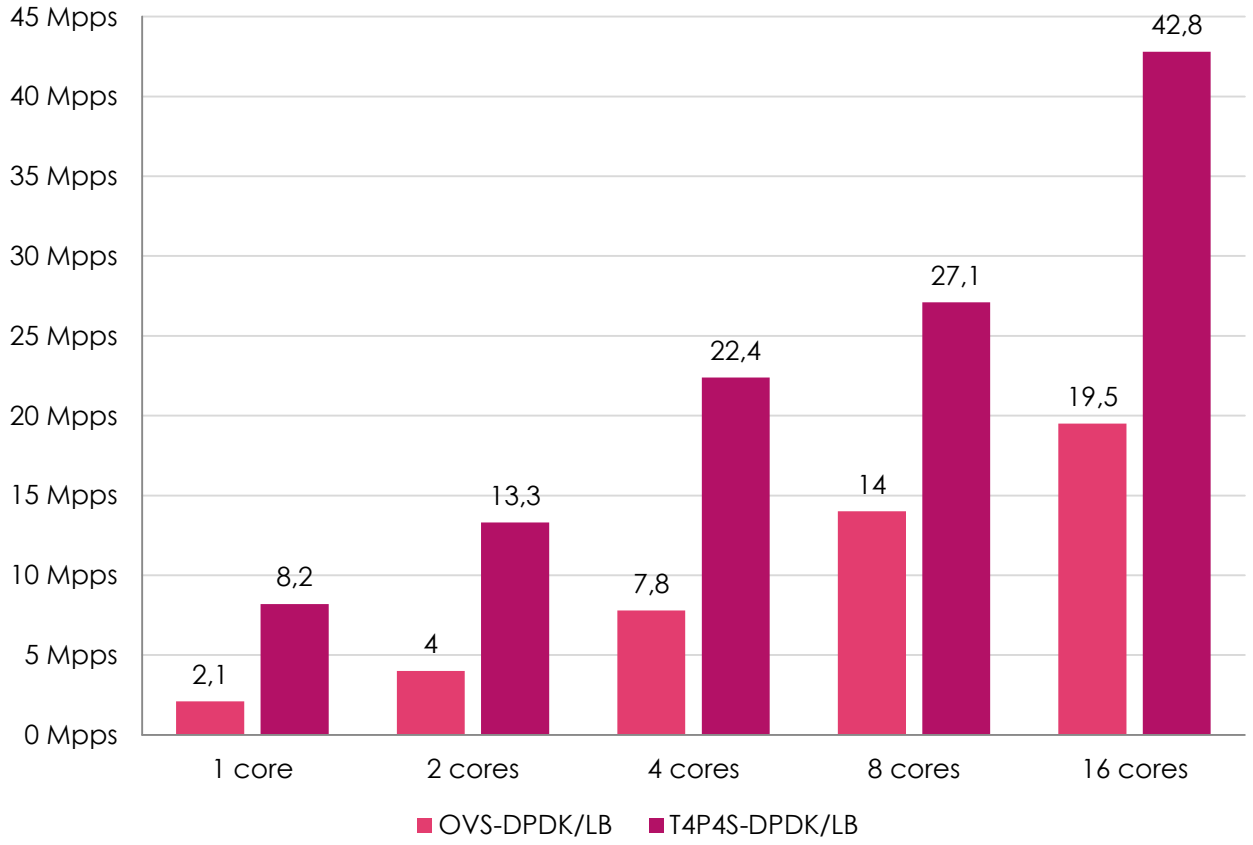
Evaluation - L3 forwarding

- L3 forwarding
  - LPM and exact match tables
- Testbed setup
  - Intel(R) Xeon(R) CPU E5-1660 v4 @ 8c 16t 3.20GHz, 8x8GB DDR4 SDRAM
  - Dual port 100 Gbps NIC
    - Mellanox MT27700 Family [ConnectX-4]
  - T4P4S performance is compared to OVS
    - Identical implementations in OpenFlow and P4
  - Pseudo random test traffic generated
    - A few hundred flows



Evaluation - Simple load balancer

- Load balancing
  - Based on the first bit of the IP address
    - LPM table with a few entries only (number of servers)
- Testbed setup
  - Intel(R) Xeon(R) CPU E5-1660 v4 @ 8c 16t 3.20GHz, 8x8GB DDR4 SDRAM
  - Dual port 100 Gbps NIC
    - Mellanox MT27700 Family [ConnectX-4]
  - T4P4S performance is compared to OVS
    - Identical implementations in OpenFlow and P4
  - Pseudo random test traffic generated
    - 10K flows



T4P4S

- A translator for P4 Switches
  - **Open source** (on GitHub)
    - Visit our site: http://p4.elte.hu
    - Or the GitHub repository: https://github.com/P4ELTE/t4p4s
  - **P4-14** language support (support of **P4-16** is under development)
  - Support of multiple targets
    - by the **Hardware Independent Core** and **Network Hardware Abstraction Libraries**
    - NetHALs for **Intel** (DPDK), **Freescale** (ODP SDK), **OpenWRT** (Native Linux) platforms

Open source available on GitHub!

TAPAS

http://p4.elte.hu

Questions?

Sandor Laki, PhD

ELTE Eötvös Loránd University

Budapest, Hungary

lakis@elte.hu