

High-Speed Forwarding: A P4 Compiler with a Hardware Abstraction Library for Intel DPDK

Sándor Laki

Eötvös Loránd University

Budapest, Hungary

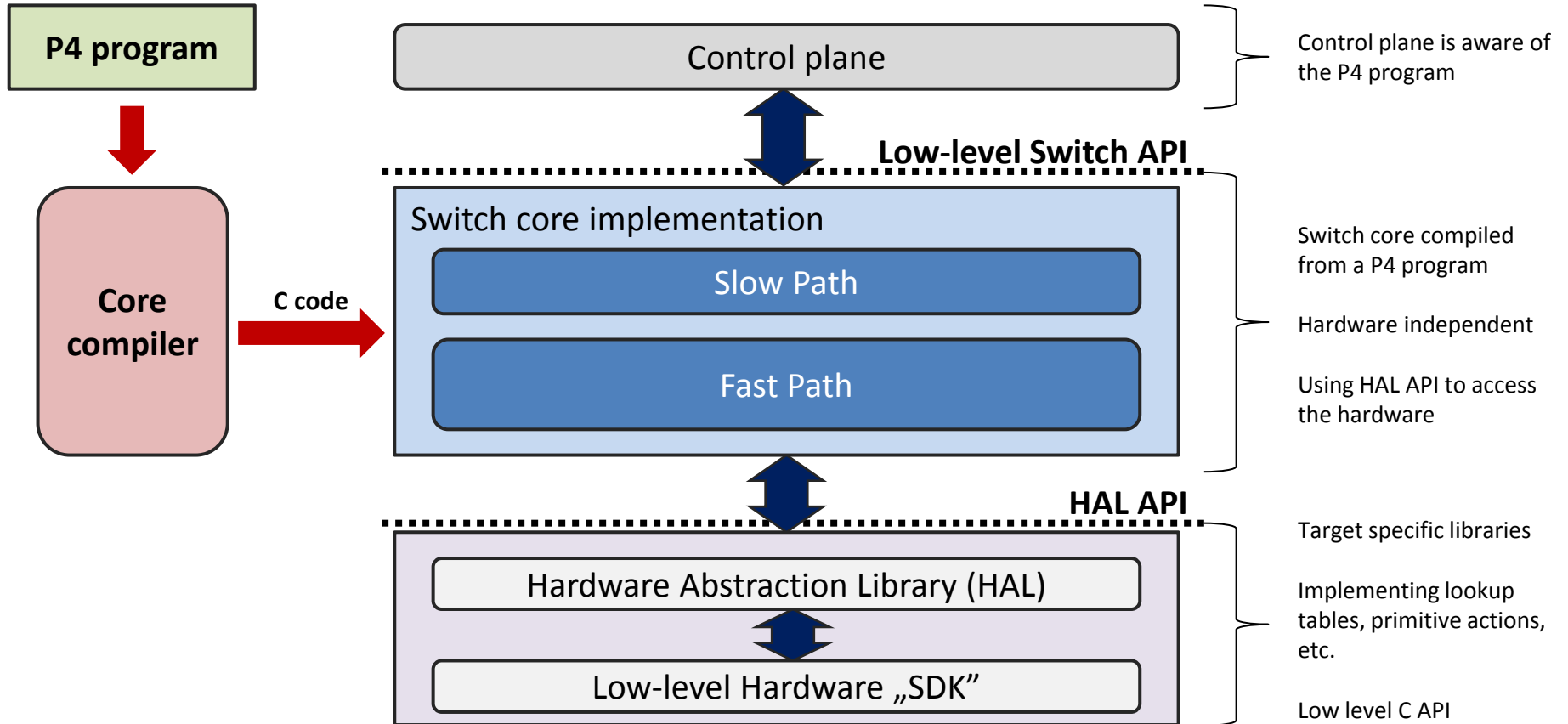
lakis@elte.hu



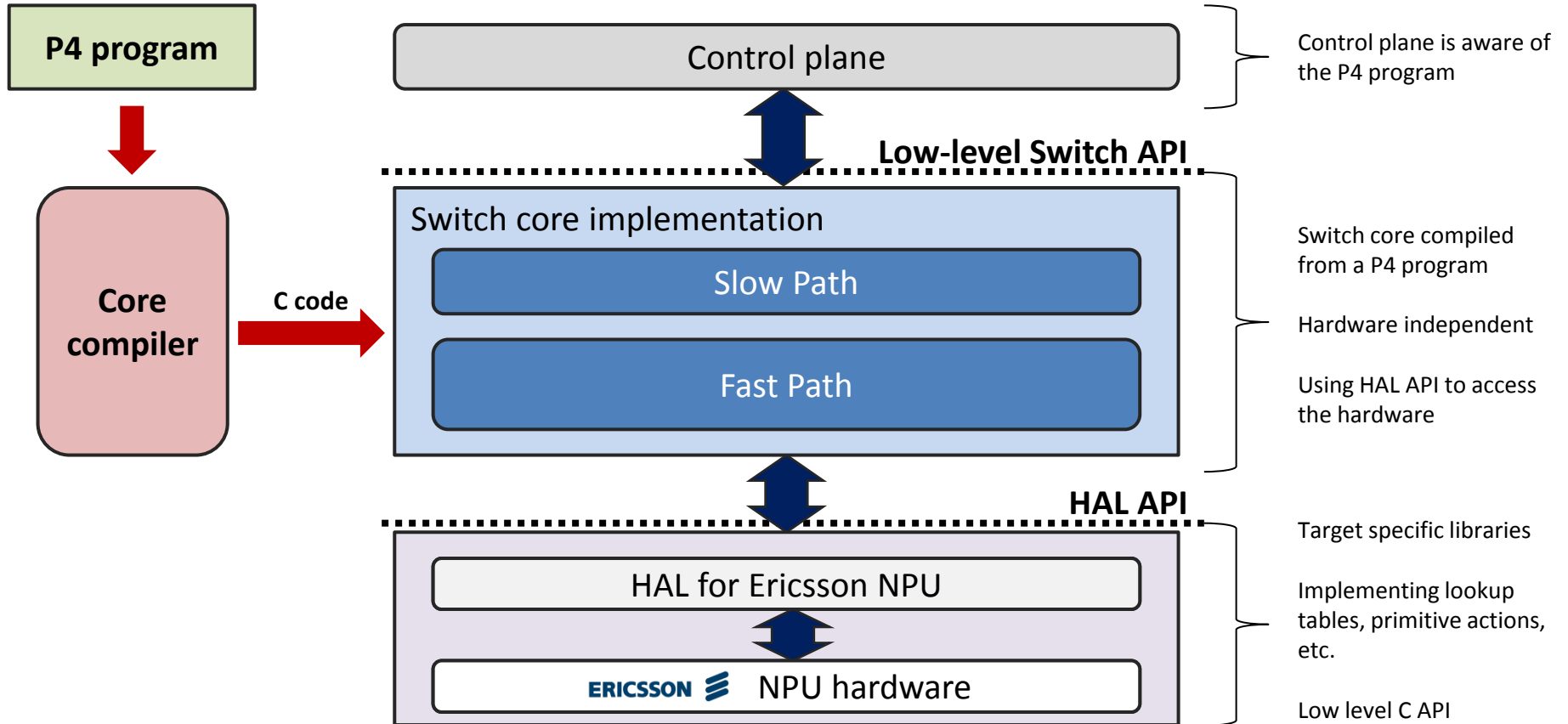
Motivation

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

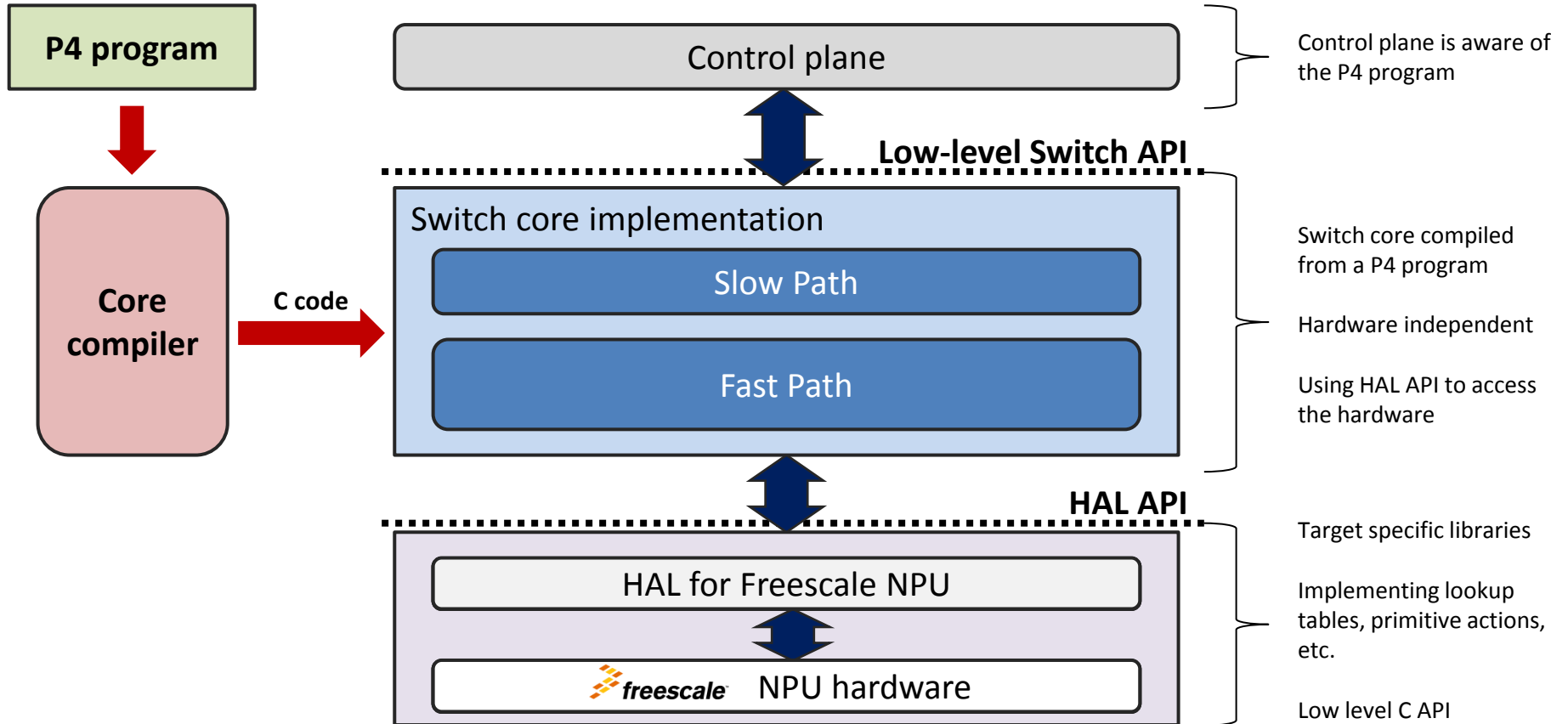
Multi-target Compiler Architecture



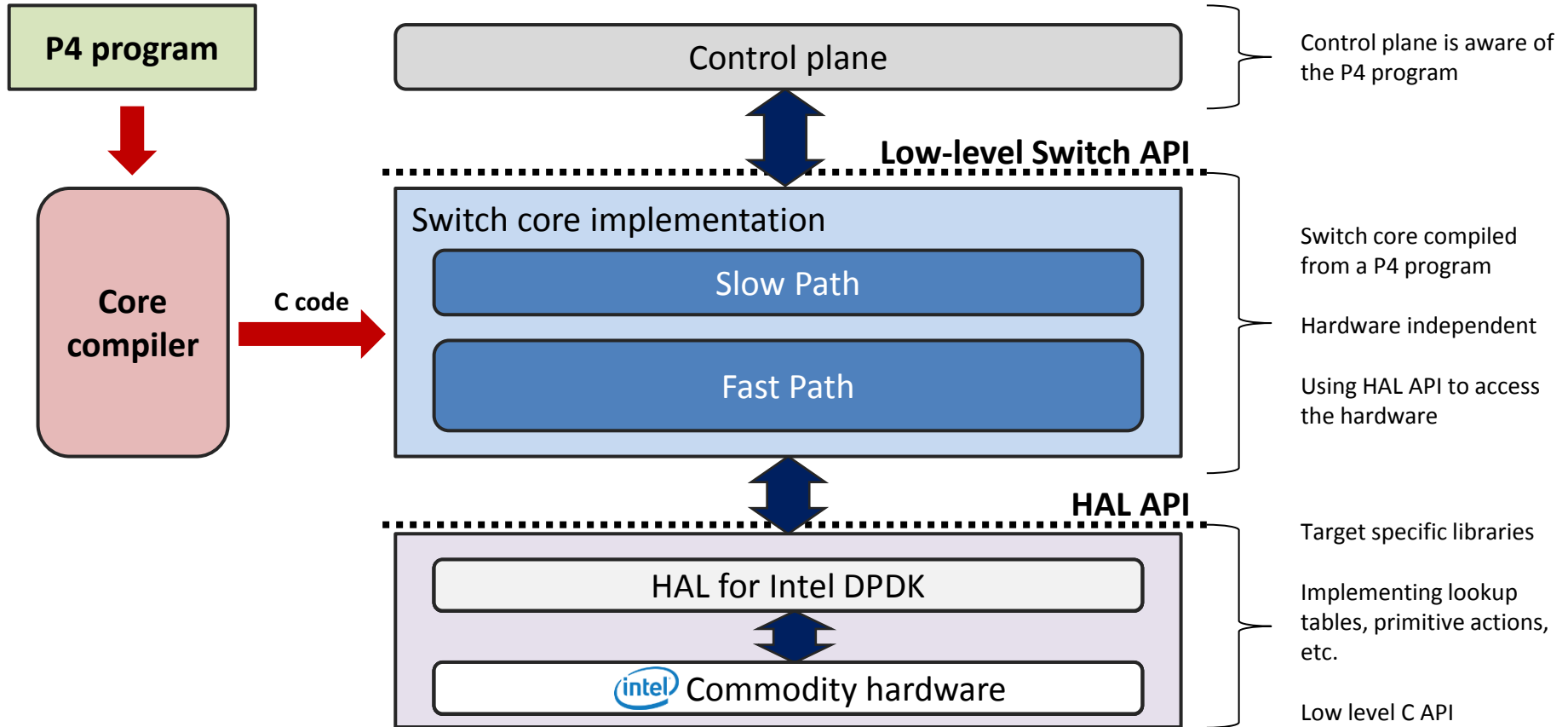
Multi-target Compiler Architecture



Multi-target Compiler Architecture



Multi-target Compiler Architecture



Multi-target Compiler Architecture

1. Hardware-independent „Core”

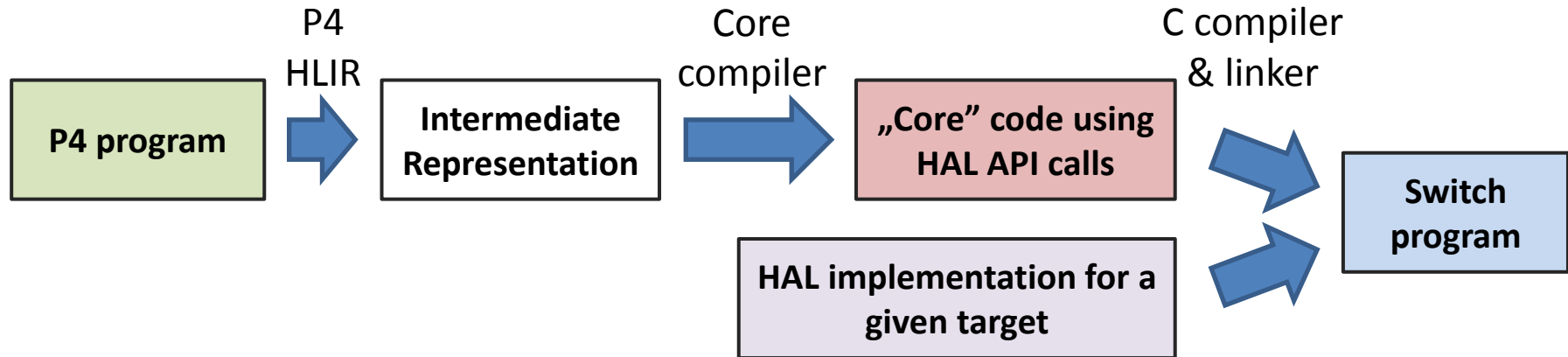
- P4 HLR is used to generate the Intermediate Representation (IR)
- Our core compiler compiles the IR to a hardware independent C code with HAL API calls

2. Hardware-dependent „HAL”

- Implementing a well defined API that fulfills the requirements of most hardware
- A static and thin library implementing networking primitives for a given target
- Written by a hardware expert

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



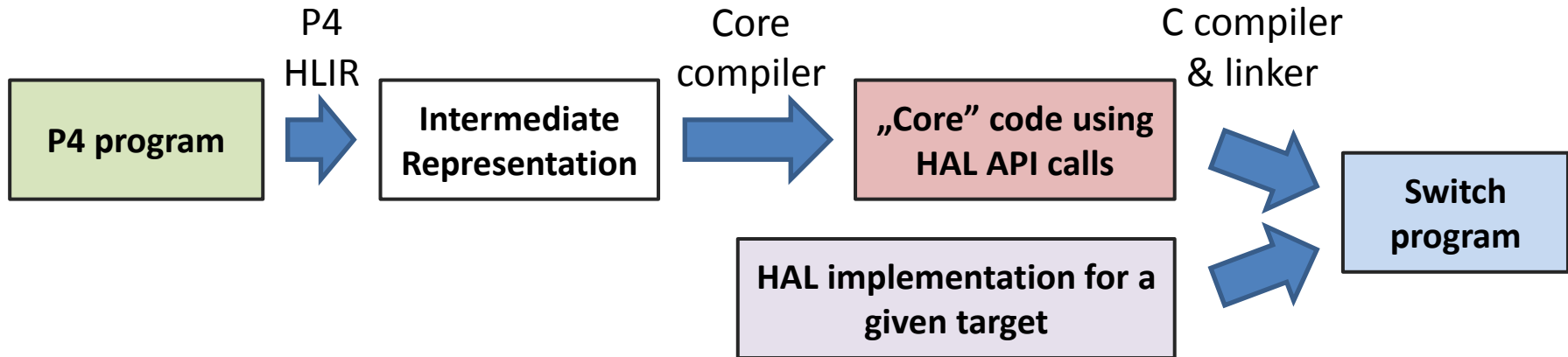
Multi-target Compiler Architecture

- **PROs**

- much simpler compiler
- modularity = better maintainability
- exchangeable HAL = retargetable switch (without rewriting a single line of code)
- HAL 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 HAL API



The „core“

Run to completion model

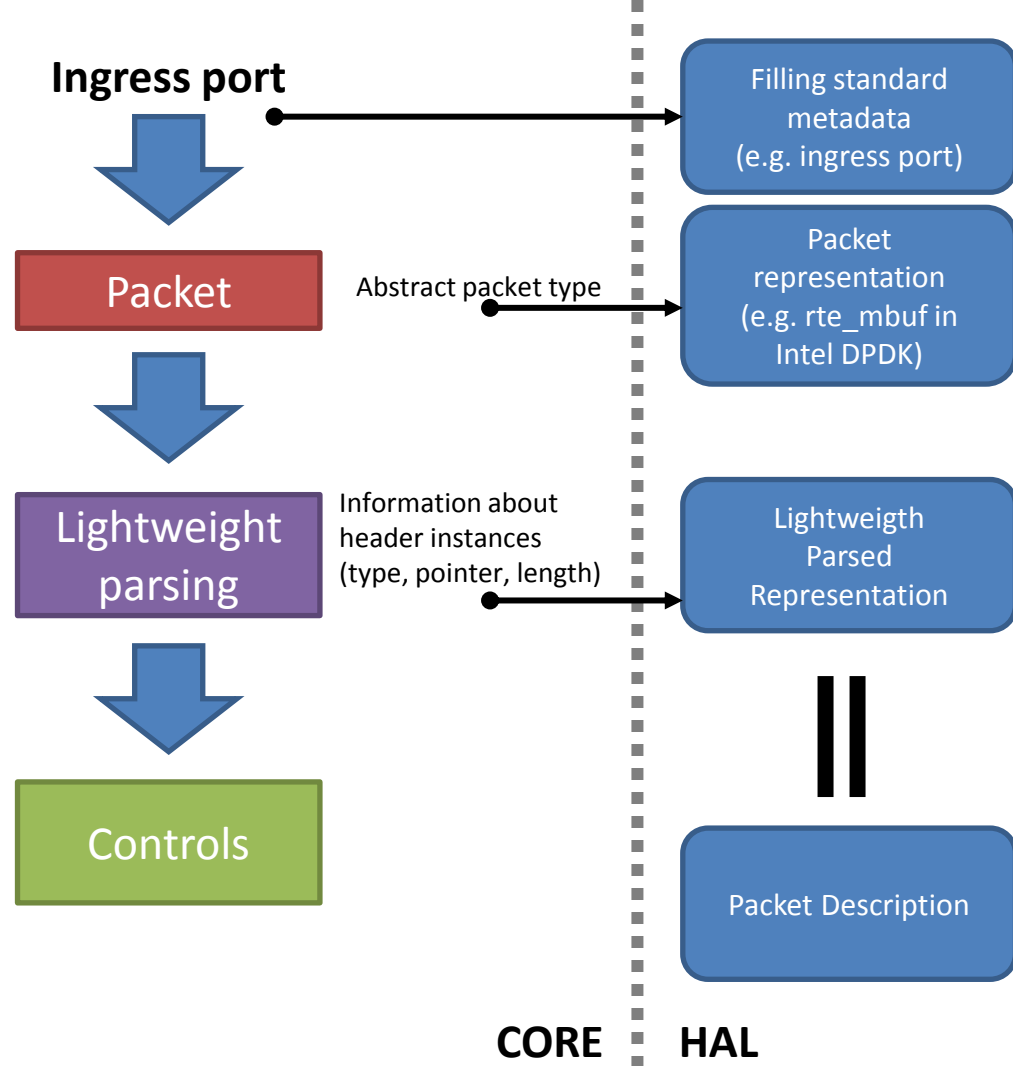
- May change in the future

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

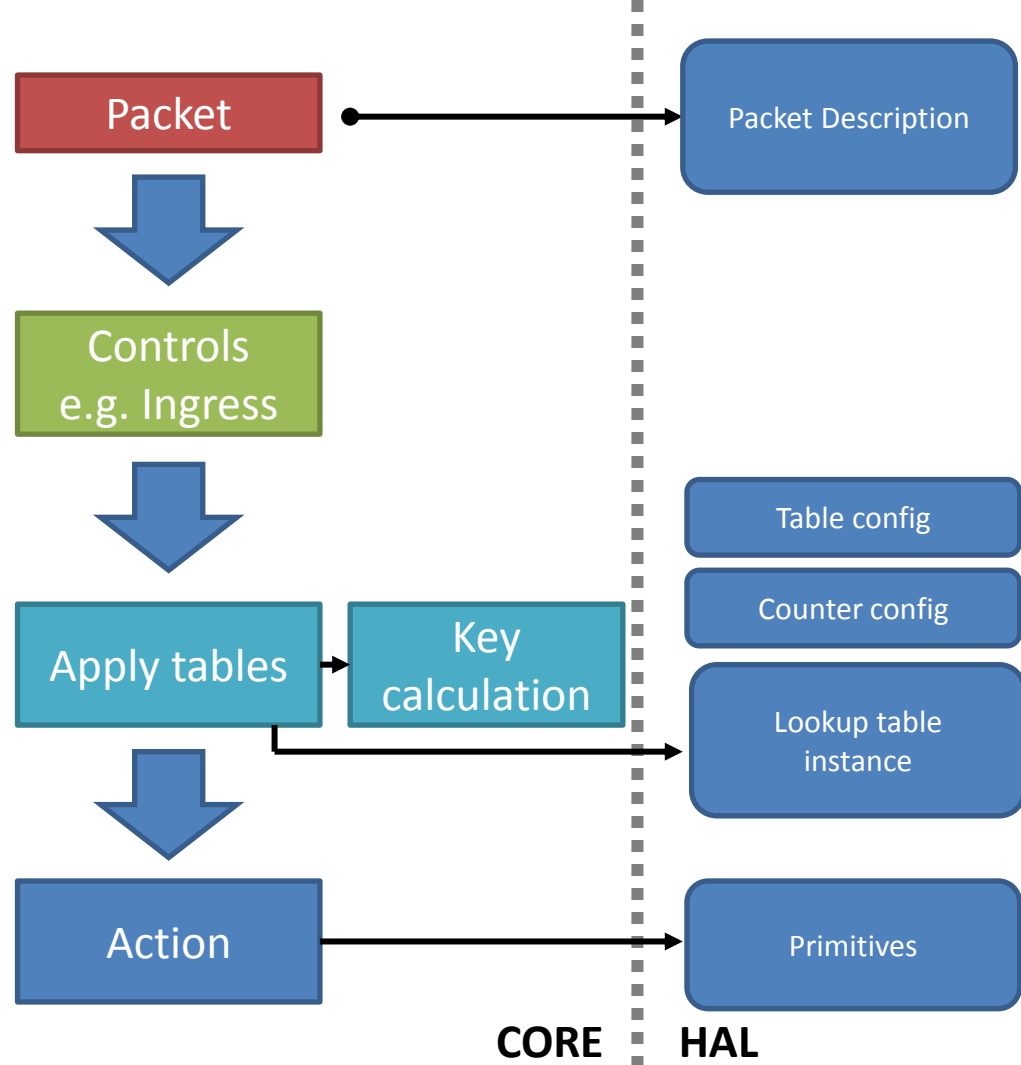
- May change in the future

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“

Run to completion model

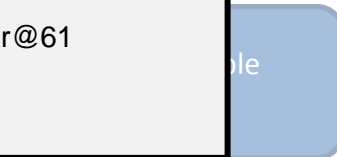
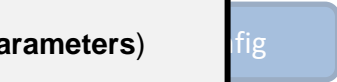
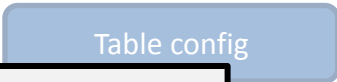
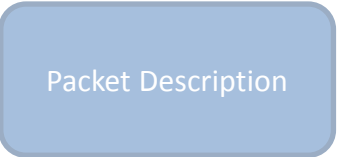
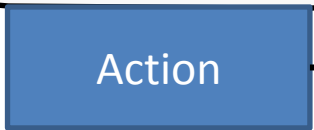
- May change in the future

Core implements

- Packet „parsing“
- Control programs

```
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



CORE | HAL



Hardware Abstraction Library

Examples for HAL primitives

Low-level generic C API

- For networking hardwares
- DPDK and NPUs

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

```
MODIFY_BYTEBUF_BYTEBUF(pd, dstfield, src, srclen)
MODIFY_INT32_BYTEBUF(pd, dstfield, src, srclen)
MODIFY_INT32_INT32(pd, dstfield, value32)
```

Field extraction

```
EXTRACT_INT32(pd, field, dst)
```

Counter operations

```
increase_counter(int counterid, int index)
read_counter(int counterid, int index)
```

Table 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)
```

What is Intel DPDK?

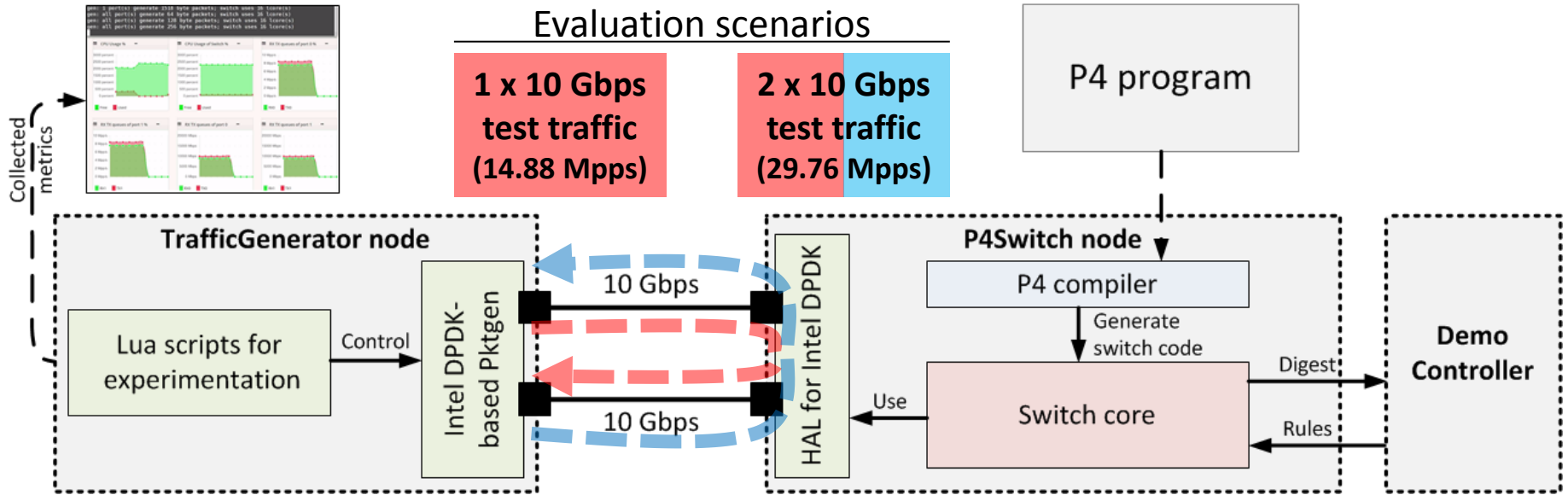
DATA PLANE DEVELOPMENT KIT

- User-space libraries to improve packet processing performance on x86
 - Bypassing Linux kernel
- Techniques to minimize the time needed for packet processing
 - DMA, Poll mode, SSE, Hugepages, Cache utilization, Lock-free multi-core sync, NUMA awareness
- Wide range of supported NICs
 - Intel cards, Mellanox, etc.
- Multi Architecture Support
 - IBM Power 8, Tile-GX, ARM v7/v8

HAL for Intel DPDK

- Our current HAL implementation is based on DPDK 2.2.0
- Reuses the 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
- NUMA support
 - Lookup tables
 - Two instances of each table on each socket - lock-free solution
 - active/passive instances
 - lcore always turns to its socket's instance
 - Counter instances for each lcore on the corresponding socket
 - To avoid overhead by cache coherency

Evaluation setup

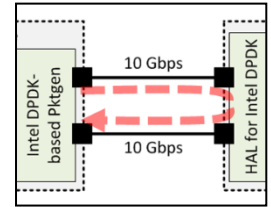
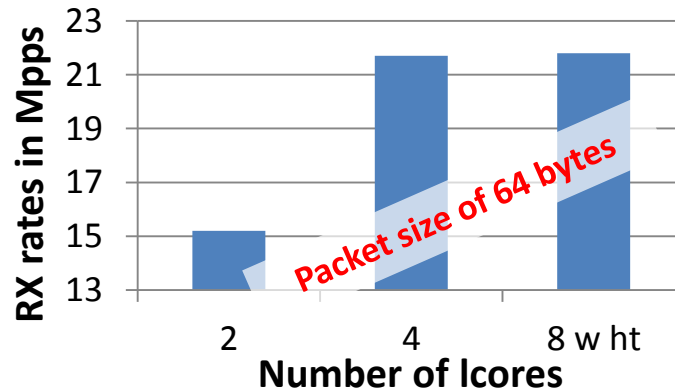
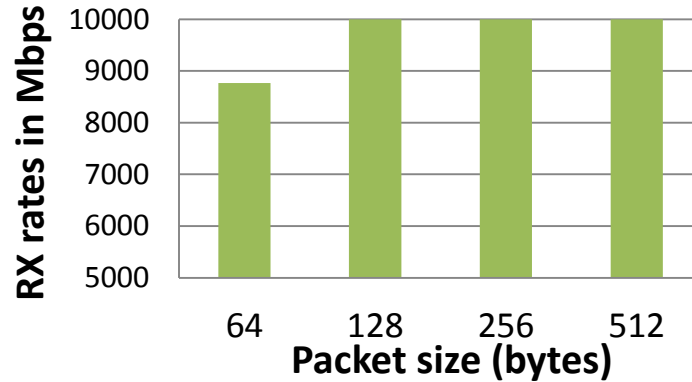


TrafficGenerator and P4Switch nodes

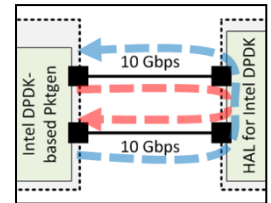
- Intel XEON E5-2630 6 cores, 12 threads, @ 2.3GHz, 2x8 GB DDR3 SDRAM
- Dual 10 Gbps NIC (Intel 82599ES)
- 1 Gbps management interface

L2 forwarding

- Simple L2 forwarding with mac learning
- Two lookup tables
 - smac & dmac
 - Exact matches only
- Generating digests
 - For unseen src-MACs
- Demo controller fills tables smac and dmac according to the digest received



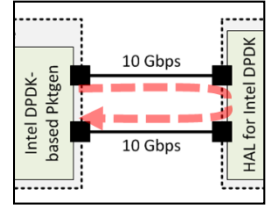
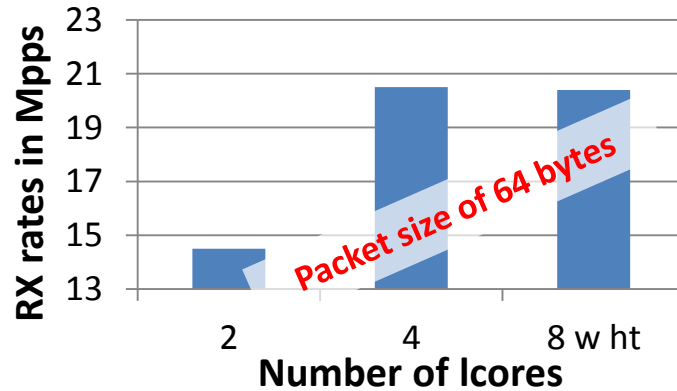
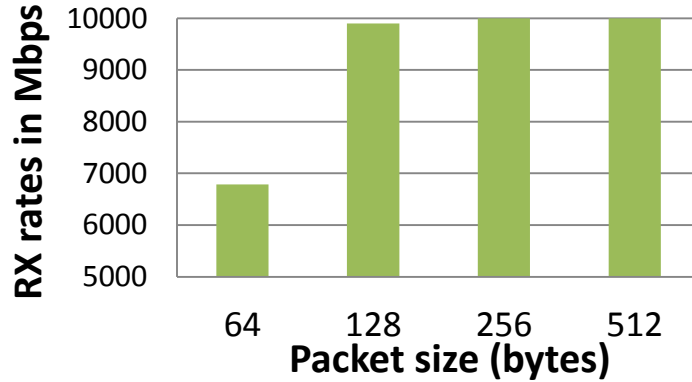
**1 x 10 Gbps
TX rate**



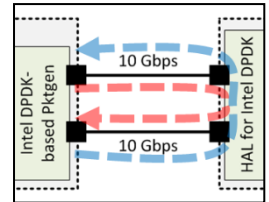
**2 x 10 Gbps
TX rate
(29.76 Mpps)**

Simple routing

- Simple L3 forwarding example
- Three lookup tables
 - ipv4_lpm, nexthops, send_frame
 - Lpm and exact matches
- Demo controller fills tables in advance



**1 x 10 Gbps
TX rate**



**2 x 10 Gbps
TX rate
(29.76 Mpps)**

Conclusion & Future Work

- Lessons learnt
 - Not easy to find the proper abstraction level of the HAL
 - P4 primitive actions are not fully implemented in the HAL
 - in most cases only small hw-dependent helper functions are defined, for flexibility and performance reasons
 - Inspection of the assembly code is needed to optimize the switch program
- Current state
 - Our compiler separates the hw dependent and independent functionalities
 - Supports P4 1.0 specification (almost complete)
 - HAL for Intel DPDK is under testing and performance tuning
- Future work:
 - HAL for Freescale and other NPUs
 - HAL extension for Slow Path development
 - Code optimization to get better performance
 - Performance and scalability tests
 - First public release of the compiler



Thank you for your attention!

```
Link State : <UP-10000-FD> <UP-10000-FD>
Pkts/s Rx : 12033975 0
Tx : 0 14971025
Mbits/s Rx/Tx : 8066/0 0/9393
Broadcast : 0 0
Multicast : 0 0
64 Bytes : 217776846 0
65-127 : 0 0
128-255 : 0 0
256-511 : 0 0
512-1023 : 0 0
1024-1518 : 0 0
Runt/Jumbo : 0/0 0/0
Errors Rx/Tx : 0/0 0/0
Total Rx Pkts : 205605779 0
Tx Pkts : 0 252699961
Rx MBs : 198167 0
Tx MBs : 0 169814
ARP/ICMP Pkts : 0/0 0/0
Tx Count/Rate : Forever/1004 Forever/1004
Pkts/s/Tx Bursts : 64/32 64/32
Dro/Beast Pkts : 1234/5678 1234/5678
Pkt Type:VLAN ID: IPv4/TCP:0001 IPv4/TCP:0001
Src IP Address : 192.168.1.1 192.168.0.1
Dst IP Address : 192.168.0.1/24 192.168.1.1/24
Src MAC Address : 00:23:90:91:1b:2a 00:23:90:91:1b:2a
Dst MAC Address : 00:23:90:91:1b:2a 00:23:90:91:1b:2b
Pktgen Ver:2.7.7(DPDK-1.7.0)
```

The first release of our P4 compiler will soon be available at
<http://p4.elte.hu>

The team: Dániel Horpácsi, Róbert Kitlei, Sándor Laki, Dániel Leskó, Máté Tejfel, Péter Vörös