Eötvös Loránd University

Processing sensor data streams and real-time event detection with programmable network devices

* joint work with Karlstad Univeristy and Ericsson Research

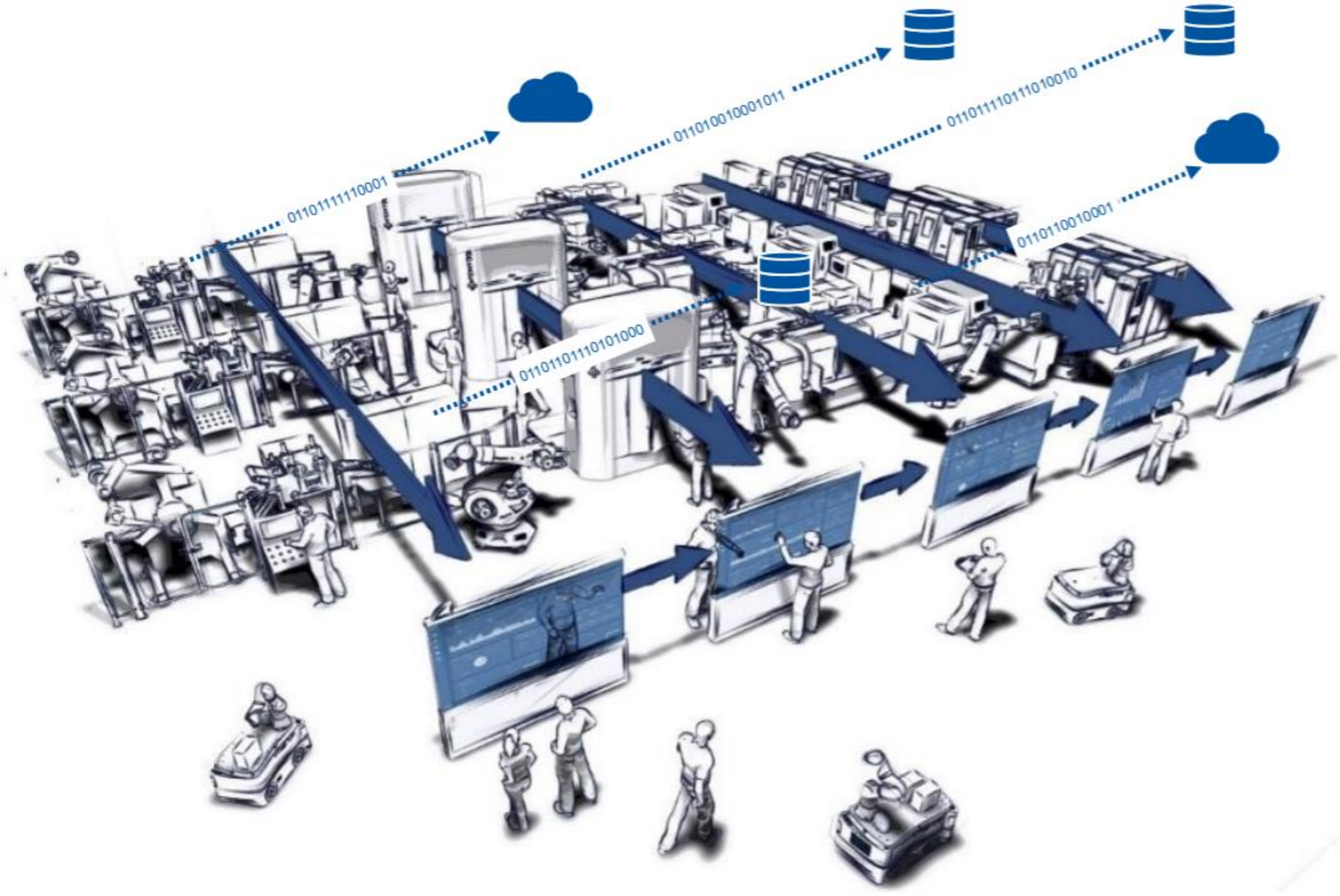
Sándor Laki, PhD

Department of Information Systems
ELTE Eötvös Loránd University
Budapest, Hungary

Contact: lakis@inf.elte.hu
Web: http://lakis.web.elte.hu

** J. Vestin, A. Kassler, S. Laki, G. Pongrácz: Towards In-Network Event detection and Filtering for Publish/Subscribe Communication using Programmable Data Planes, In IEEE Transactions on Network and Service Management (IEEE TNSM), accepted*

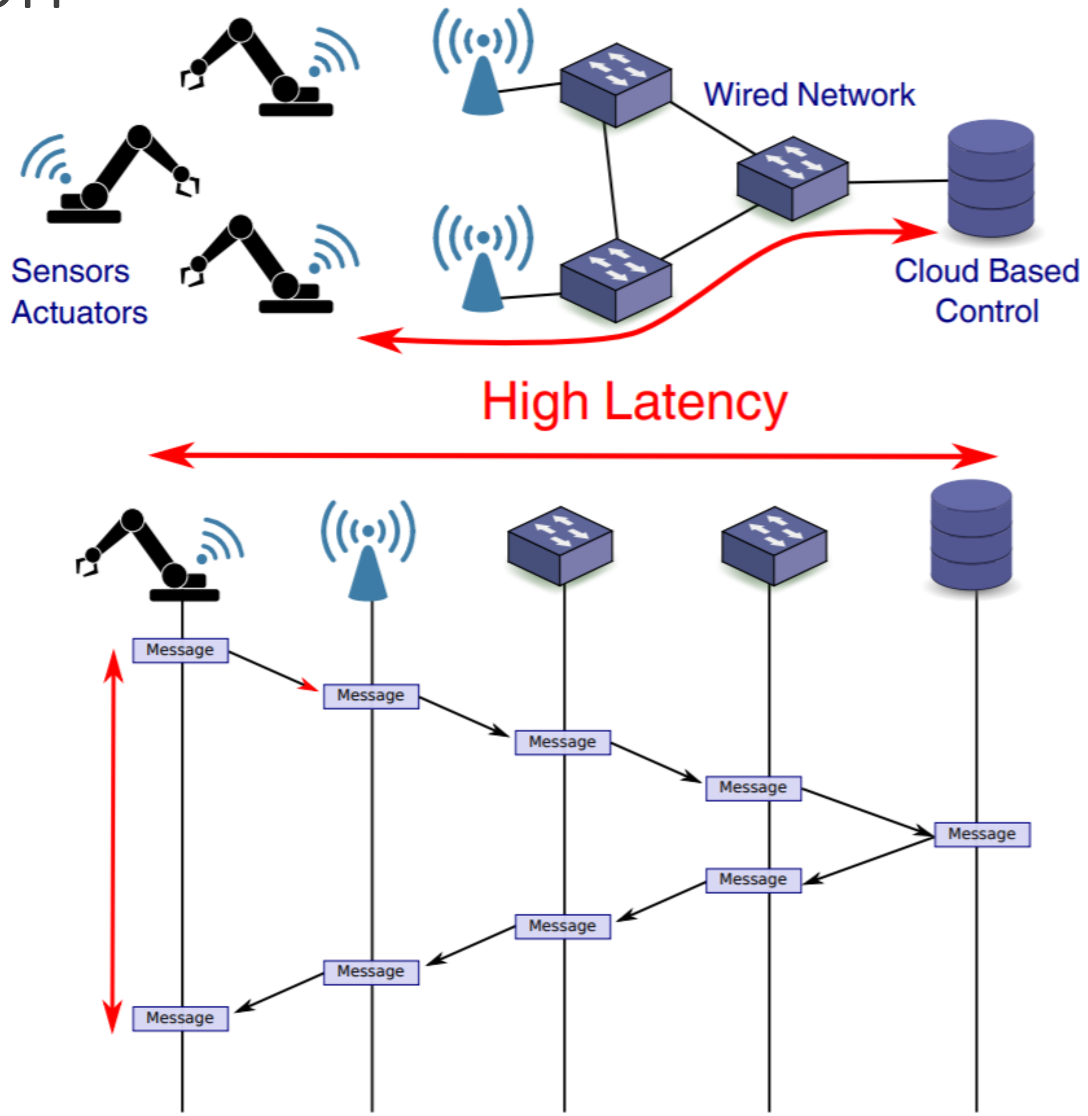
Industry 4.0
Highly integrated smart production



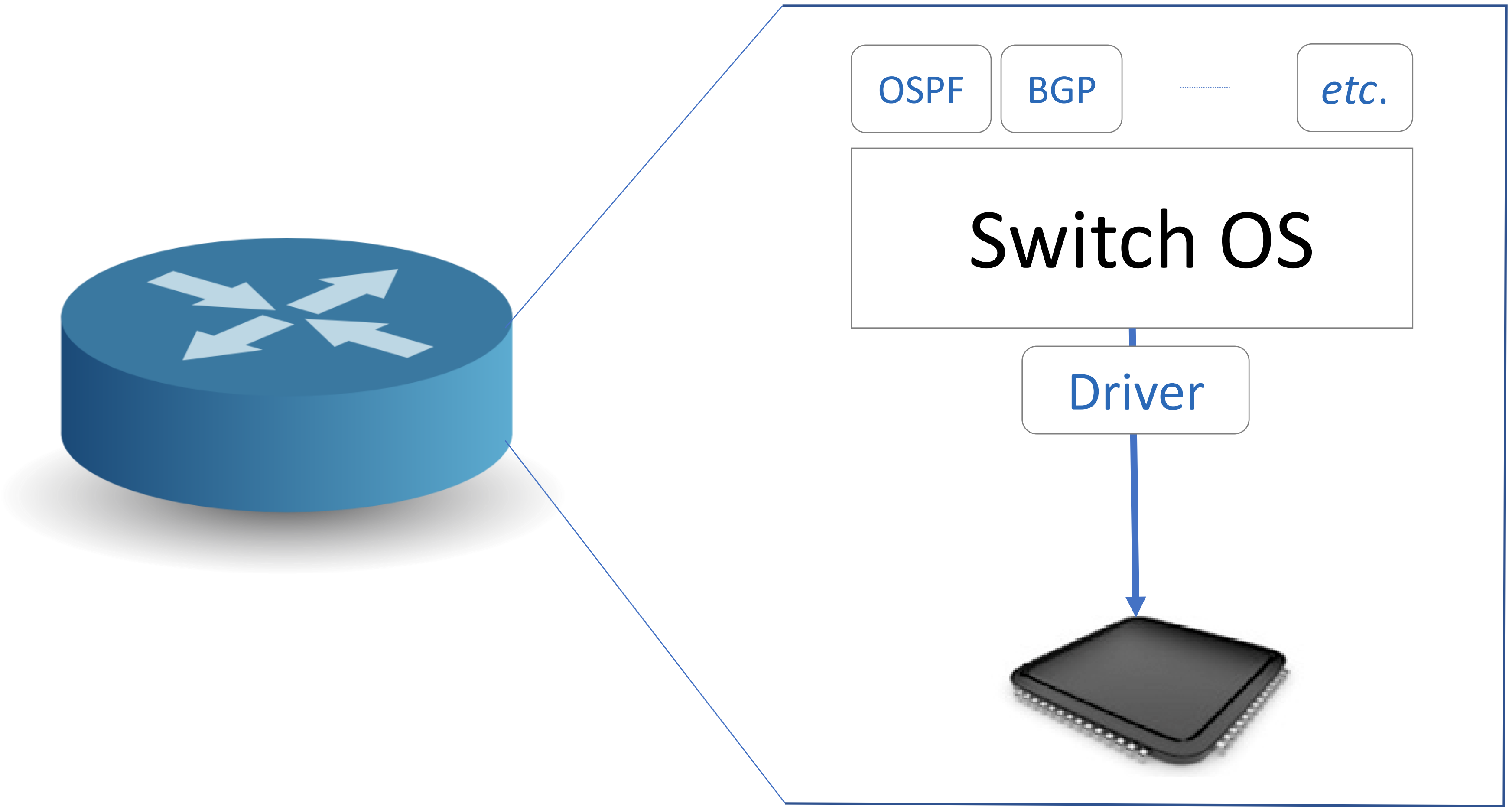
Industry 4.0
Highly integrated smart production

- **Smart Production**

  Multiple physical processes
  - In parallel, but highly dependent
  - Precision sensing provides massive amounts of data.
  - Control algorithms run in local cloud.
- **Challenges**
  - High control requires stable and ultra-low latency.
  - Raw sensor data requires huge data rates. (In particular imaging and AR)

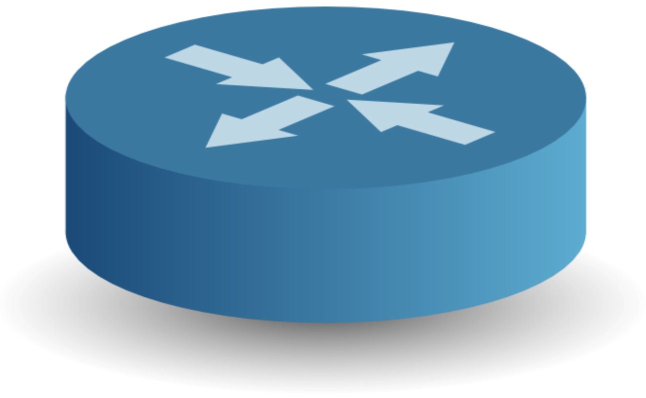


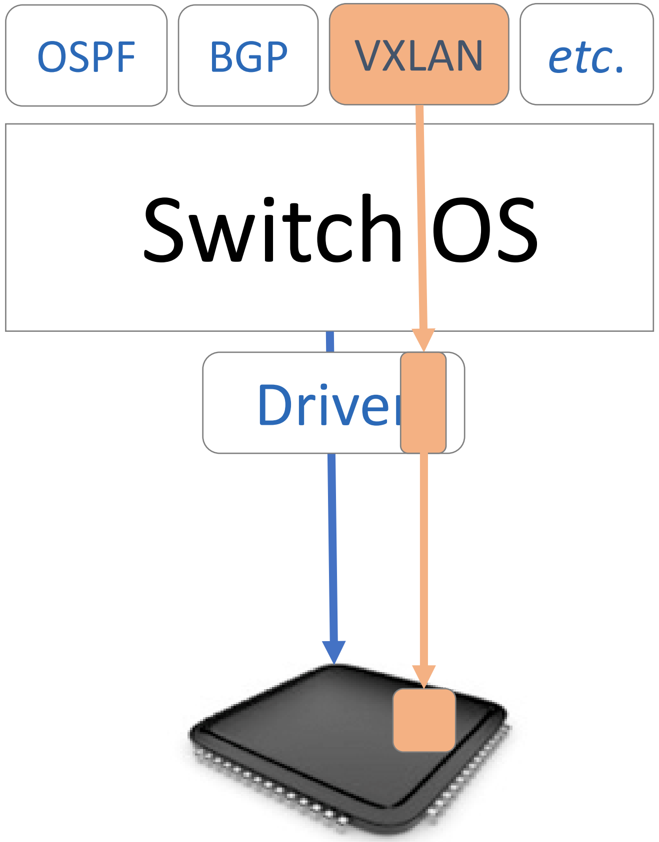
Problem with fixed function ASICs



Source: Slides of Nick McKeown

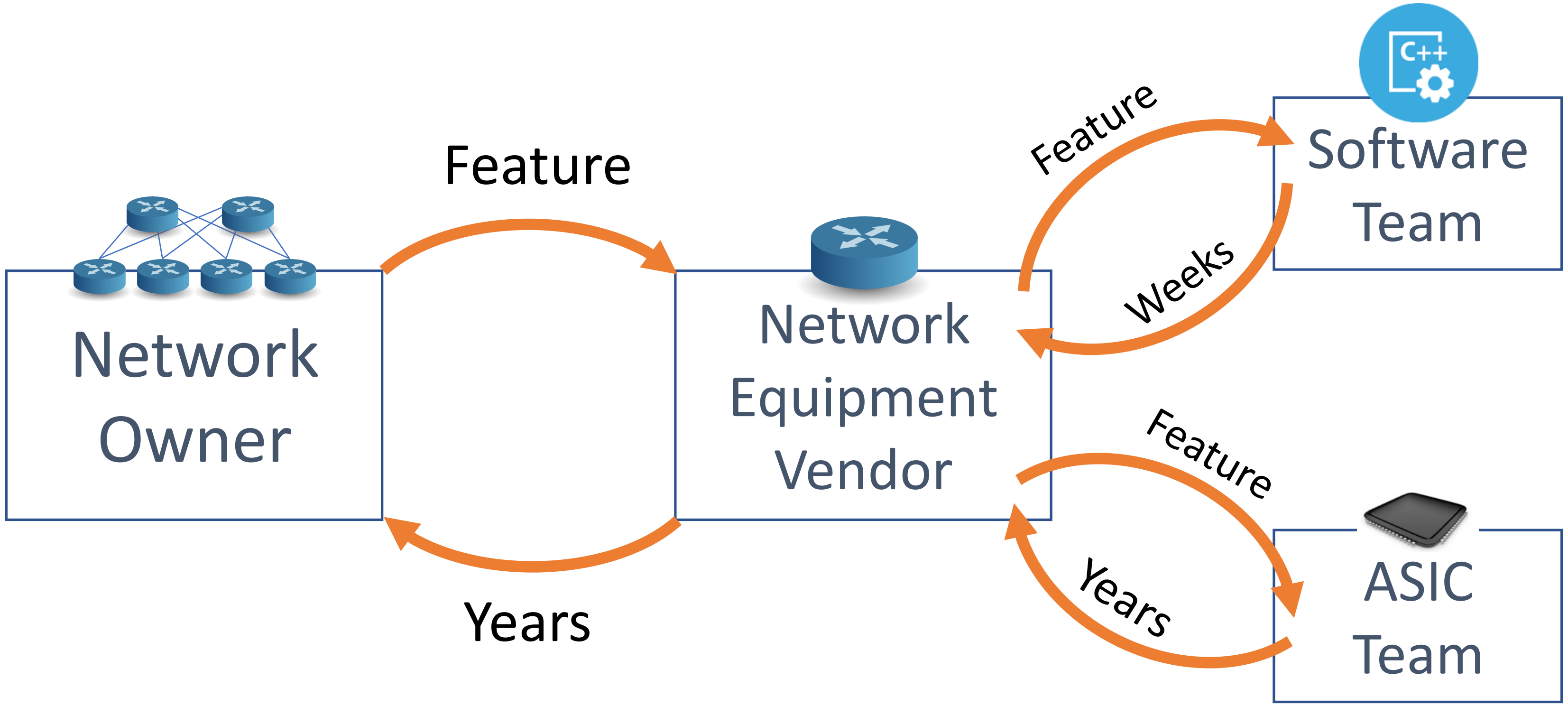
Problem with fixed function ASICs





Source: Slides of Nick McKeown

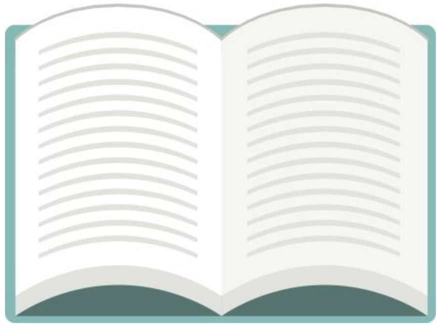
Development cycle of a new network feature

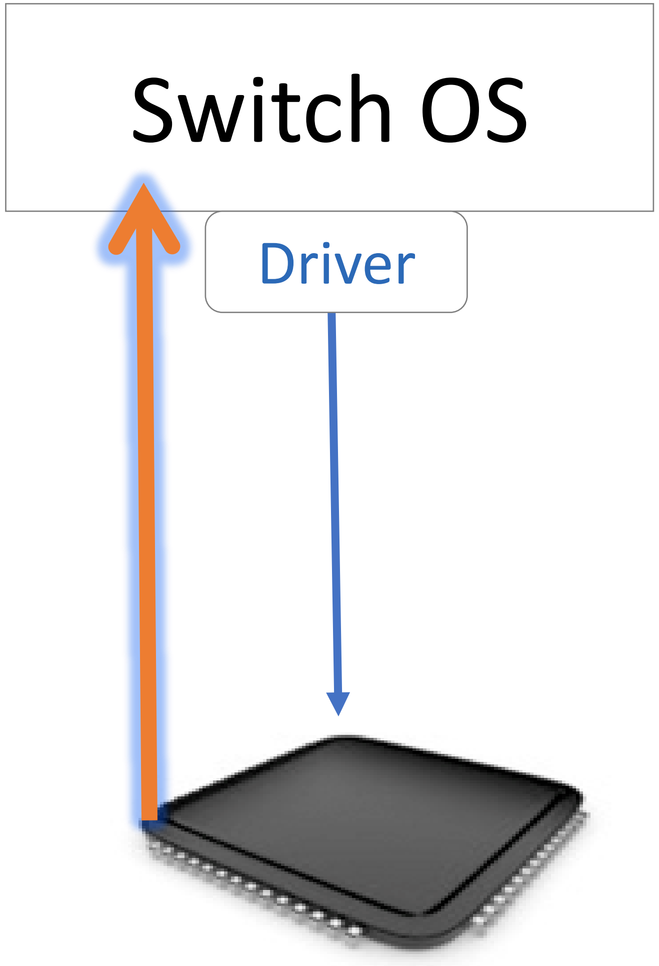


Source: Slides of Nick McKeown

Network systems are built "bottoms-up"

"This is how I process packets …"





Fixed-function switch

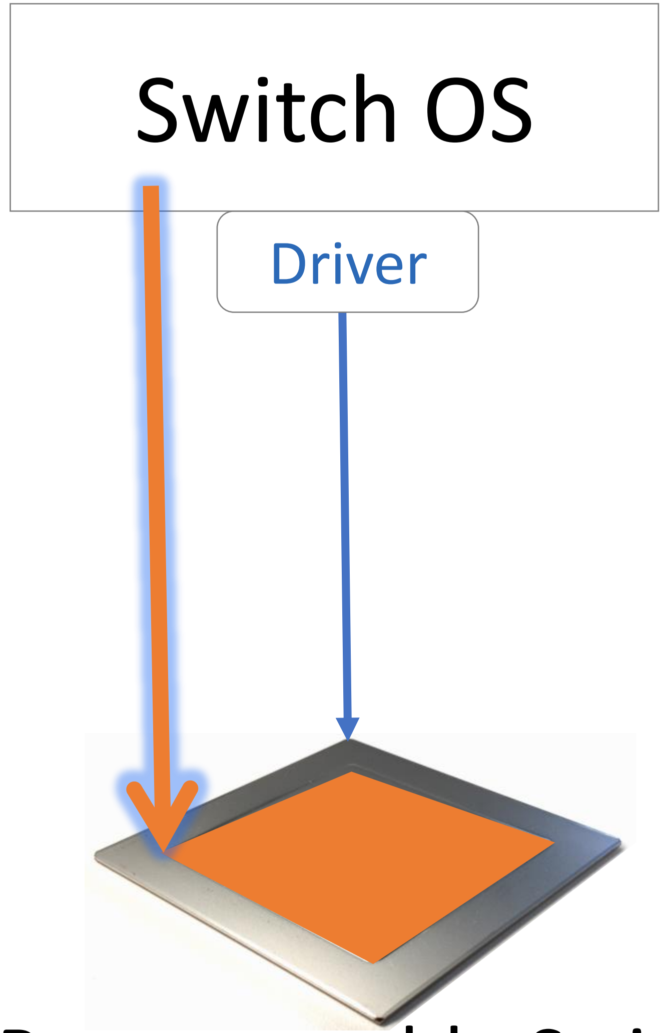
Source: Slides of Nick McKeown

Network systems are starting to be programmed “top-down”

“This is precisely how you must process packets”

```
table int_table {
  reads {
    ip.protocol;
  }
  actions {
    export_queue_latency;
  }
}
```

```
action export_queue_latency (sw_id) {
  add_header(int_header);
  modify_field(int_header.kind, TCP_OPTION_INT);
  modify_field(int_header.len, TCP_OPTION_INT_LEN);
  modify_field(int_header.sw_id, sw_id);
  modify_field(int_header.q_latency,
               intrinsic_metadata.deq_timedelta);
  add_to_field(tcp.dataOffset, 2);
  add_to_field(ipv4.totalLen, 8);
  subtract_from_field(ingress_metadata.tcpLength,
                      12);
}
```



Programmable Switch

Source: Slides of Nick McKeown

https://www.sigcomm.org/sites/default/files/ccr/papers/2014/July/0000000-0000004.pdf

00004.pdf 1 / 8

P4: Programming Protocol-Independent Packet Processors

Pat Bosshart[†], Dan Daly[*], Glen Gibb[†], Martin Izzard[†], Nick McKeown[‡], Jennifer Rexford[**], Cole Schlesinger[**], Dan Talayco[†], Amin Vahdat[¶], George Varghese[§], David Walker[**]
[†]Barefoot Networks [*]Intel [‡]Stanford University [**]Princeton University [¶]Google [§]Microsoft Research

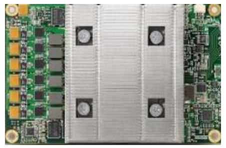
ABSTRACT

P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able to change the way switches process packets once they are

multiple stages of rule tables, to allow switches to expose more of their capabilities to the controller.

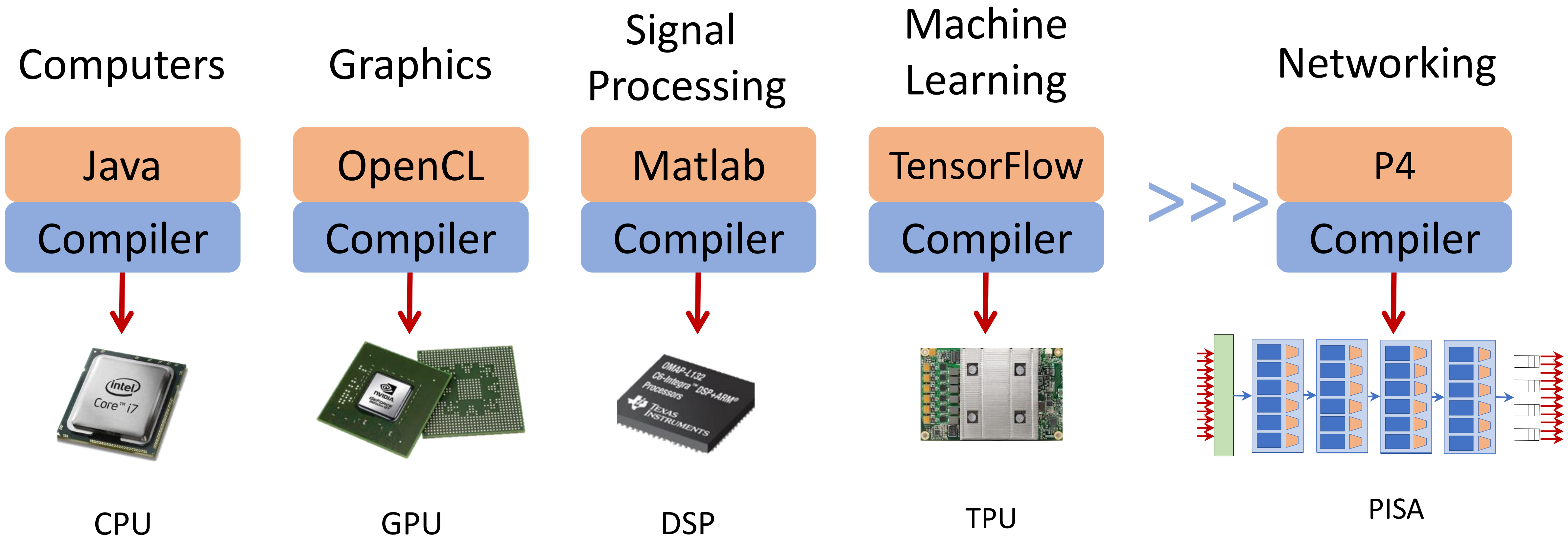
The proliferation of new header fields shows no signs of stopping. For example, data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality. Rather than repeatedly extending the OpenFlow specification, we argue that future switches should support flexible mechanisms for parsing packets and matching header fields, allowing controller applications to leverage these capabilities through a common, open inter-

Domain Specific Processors

Computers	Graphics	Signal Processing	Machine Learning		Networking
Java	OpenCL	Matlab	TensorFlow	>>>	Language
Compiler	Compiler	Compiler	Compiler		Compiler
↓	↓	↓	↓		↓
					?
CPU	GPU	DSP	TPU		

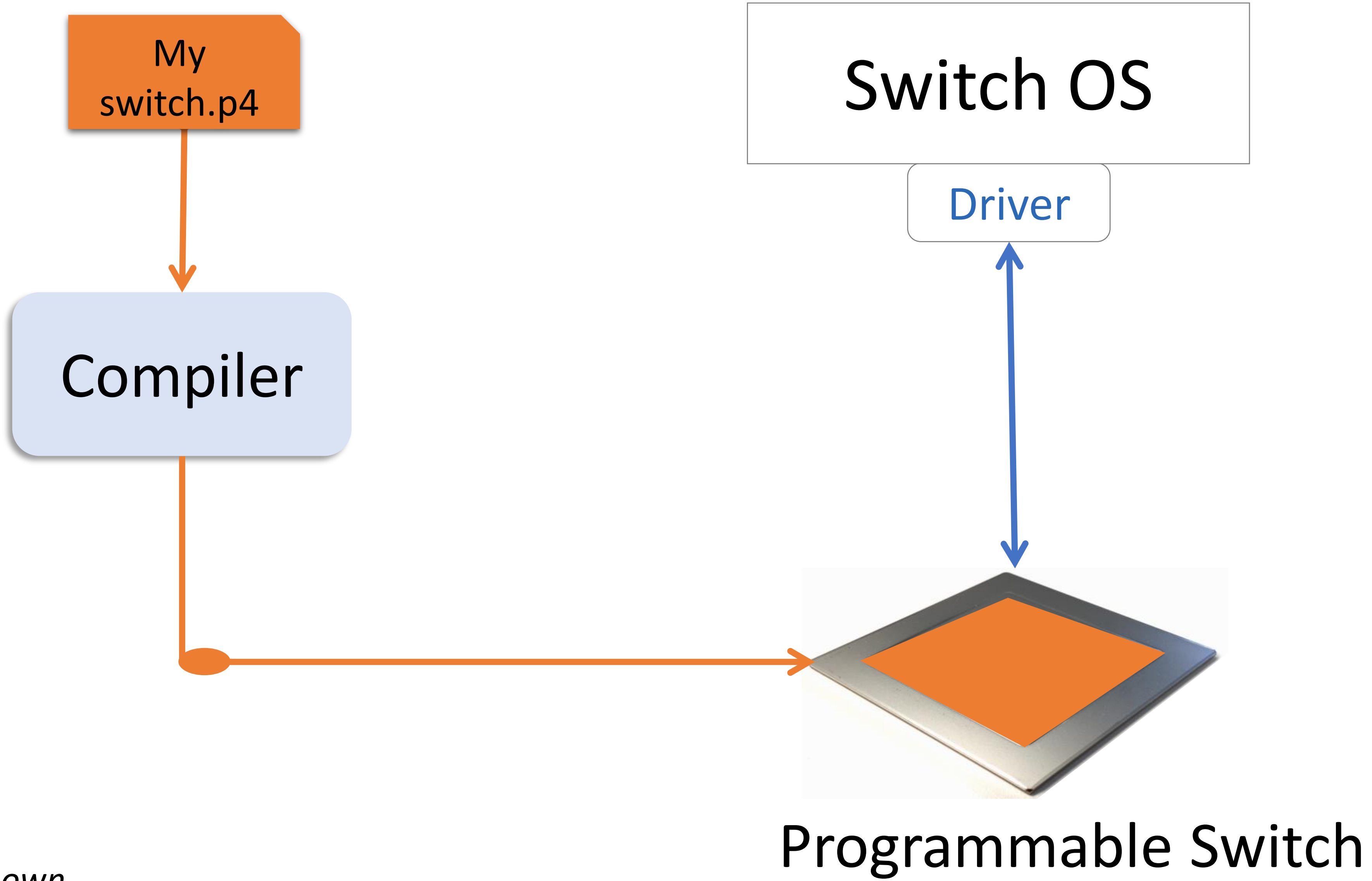
Source: Slides of Nick McKeown

Domain Specific Processors



Source: Slides of Nick McKeown

My own data plane program

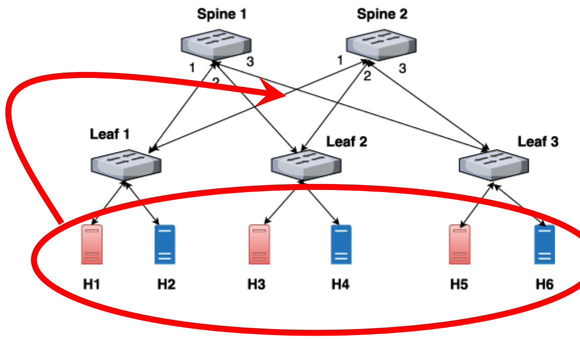


Source: Slides of Nick McKeown

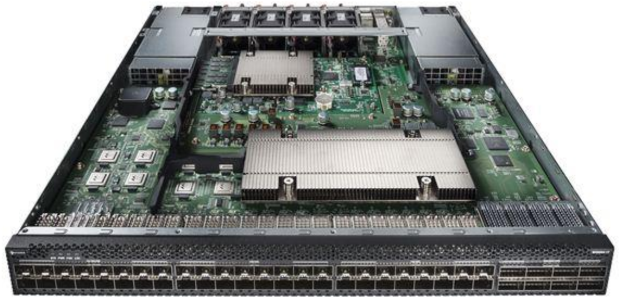
In-network computing

- **Emerging field** of networking
  - With the advent of programmable switches (BF/Intel Tofino) and P4 language

- Idea of **moving computations** from servers **to the network**
  - Enabling novel applications: caching, stream processing, query processing, load balancing, real-time control, in-network consensus, etc.



- Programmable switches are **not only** packet forwarding elements
  - Unexploited computational capacities
  - High throughput, ultra-low latency
  - Limitations
    - pipeline computing model, limited number of stages, limited memory, …

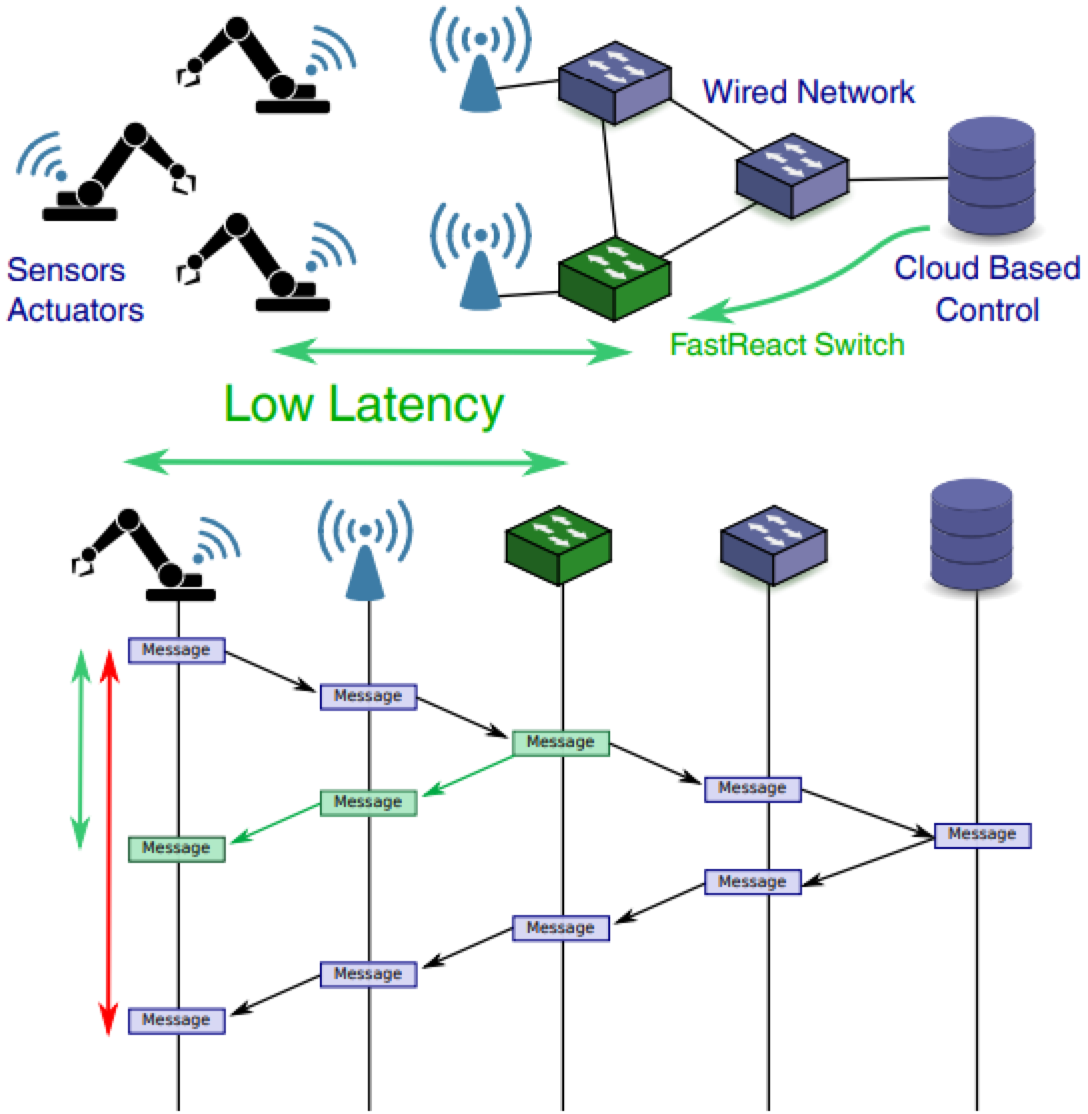


A good summary: https://www.sigarch.org/in-network-computing-draft/

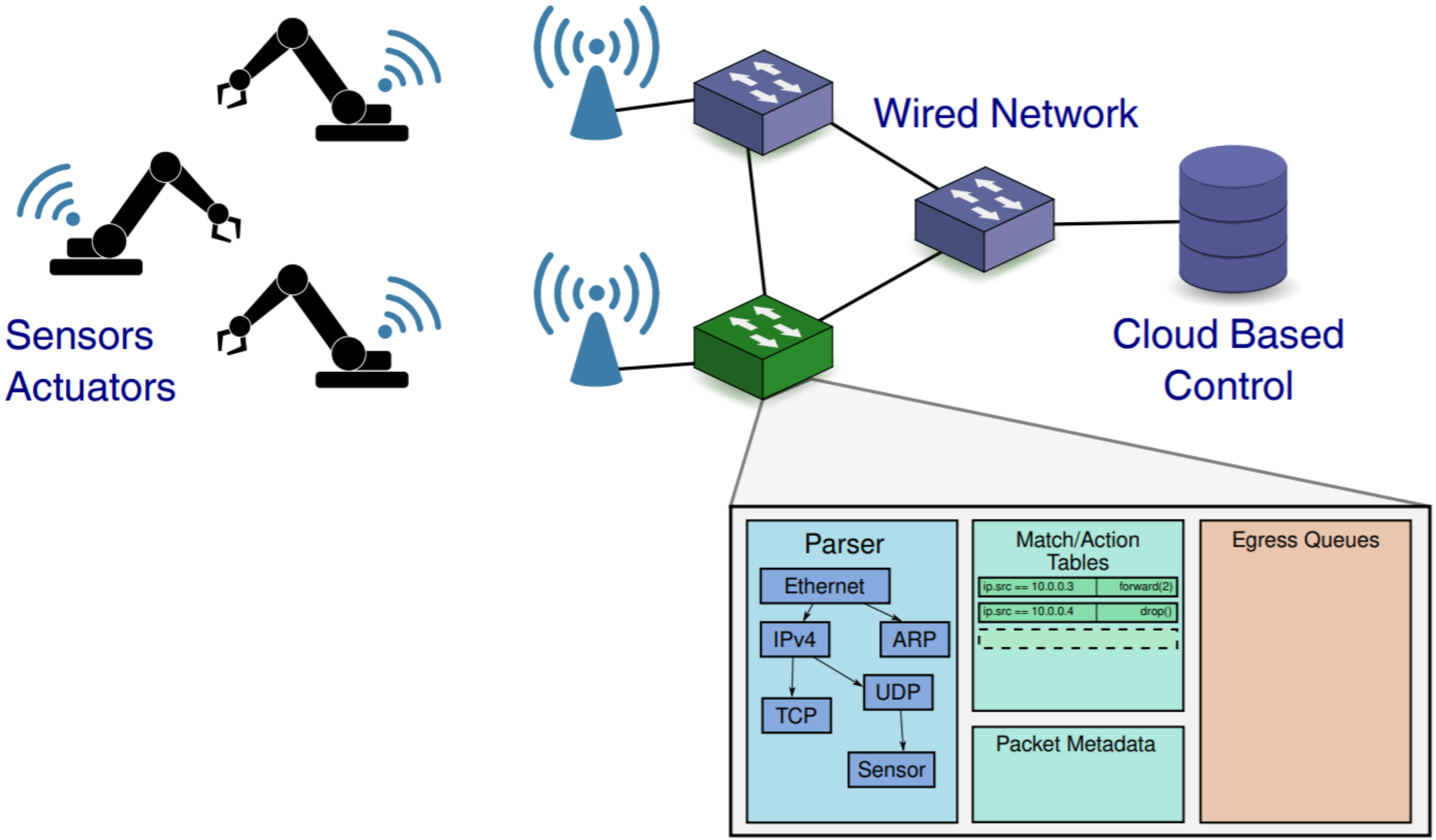
In-Network Event Detection and Filtering for Publish/Subscribe Communication

- **Local Decision Making** instead of centralized control
  - Early reaction reduces time required for processing
  - Reduces network data rate
  - Fewer devices that can fail
- **FastReact**
  - Implemented in P4 data plane programming language
  - Sensor value history, moving averages, etc.
  - Trigger local actions based on locally stored data

```
if (sensor1 > 50) && (sensor2 < 25):
    trigger_actuator(<portno>)
```

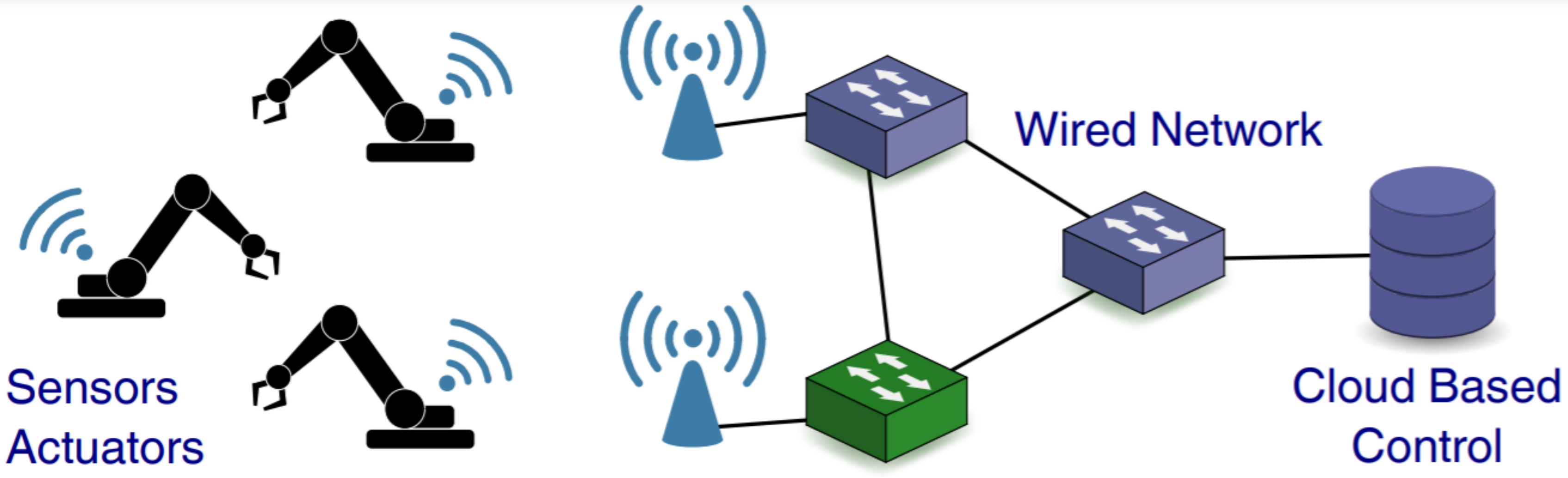


In-Network Event Detection and Filtering for Publish/Subscribe Communication



FastReact P4 Program

In-Network Event Detection and Filtering for Publish/Subscribe Communication



Collect from multiple sensors

ID: Temp4C
Val: 67

ID: Temp4D
Val: 33

ID: Temp4E
Val: 32

Also keep historical values

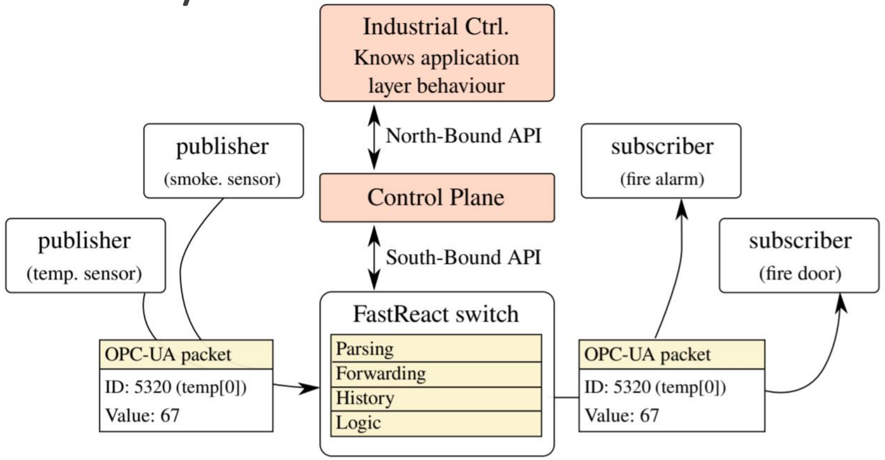
Switch Local Memory
Temp4C: 67
Temp4D: 33
Temp4E: 32

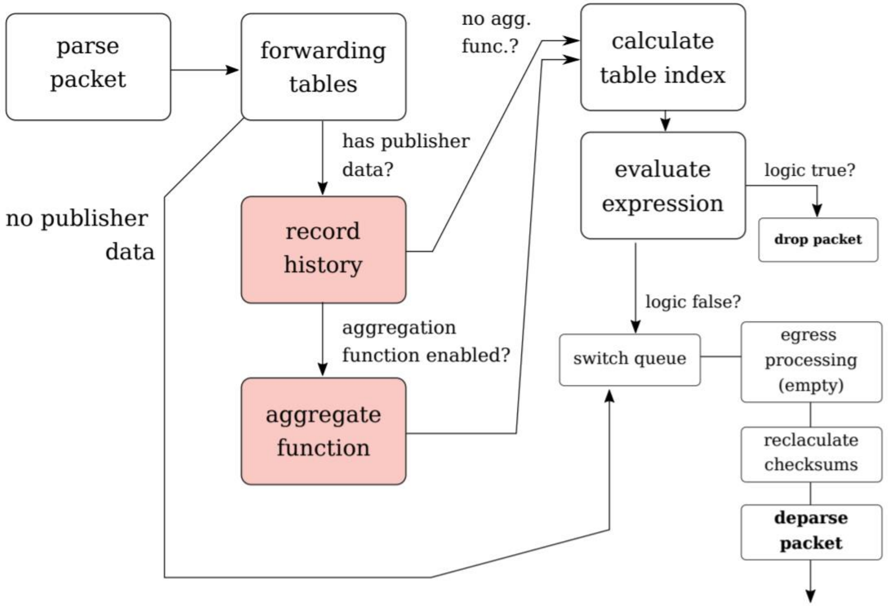
Switch Local Memory
Temp4C: 67, 66, 63, 65...
Temp4D: 33, 33, 33, 32...
Temp4E: 32, 32, 31, 32...

Can perform actions on certain conditions

```
if Temp4C > 70: notify actuator
```

In-Network Event Detection and Filtering for Publish/Subscribe Communication





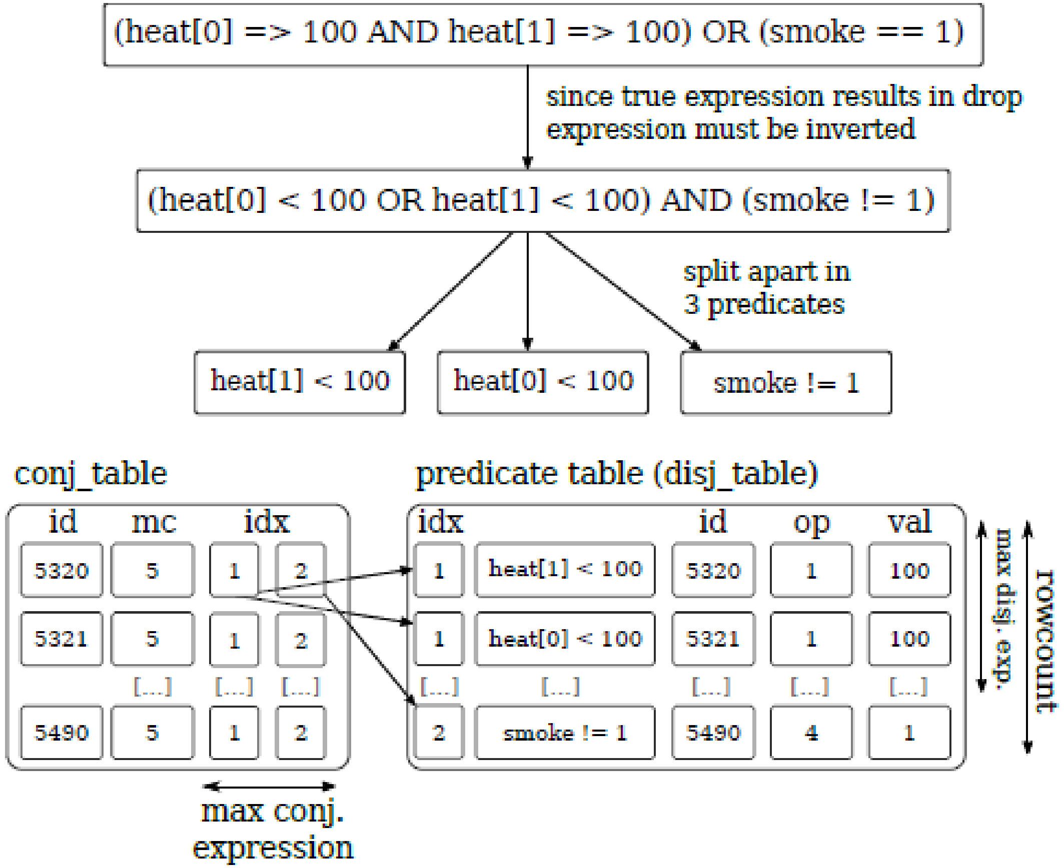


Fig. 3: FastReact table processing for the expression (heat[0] $\geq$ 100 $\wedge$ heat[1] $\geq$ 100) $\vee$ (smoke = 1).

Evaluation - Latency

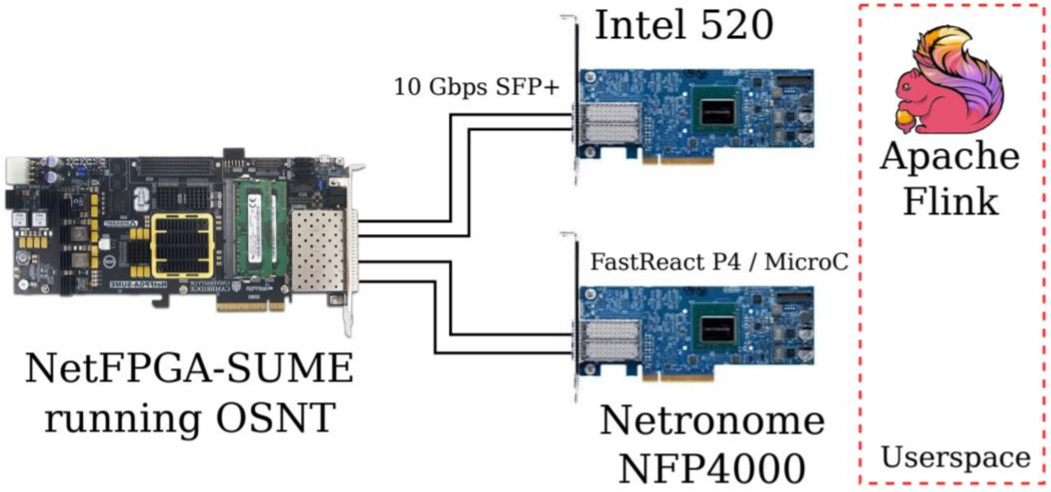


Fig. 5: Testbed setup used for evaluation

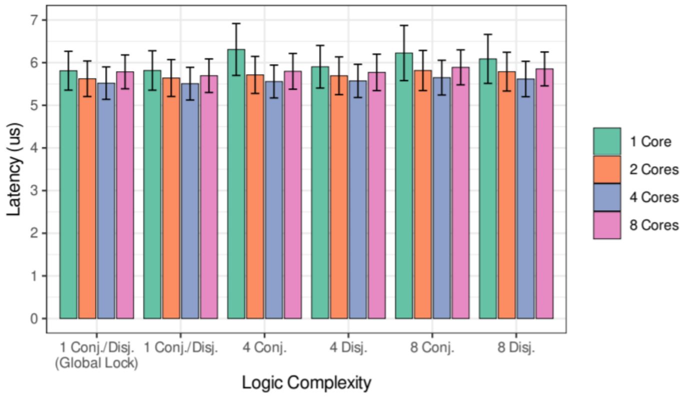
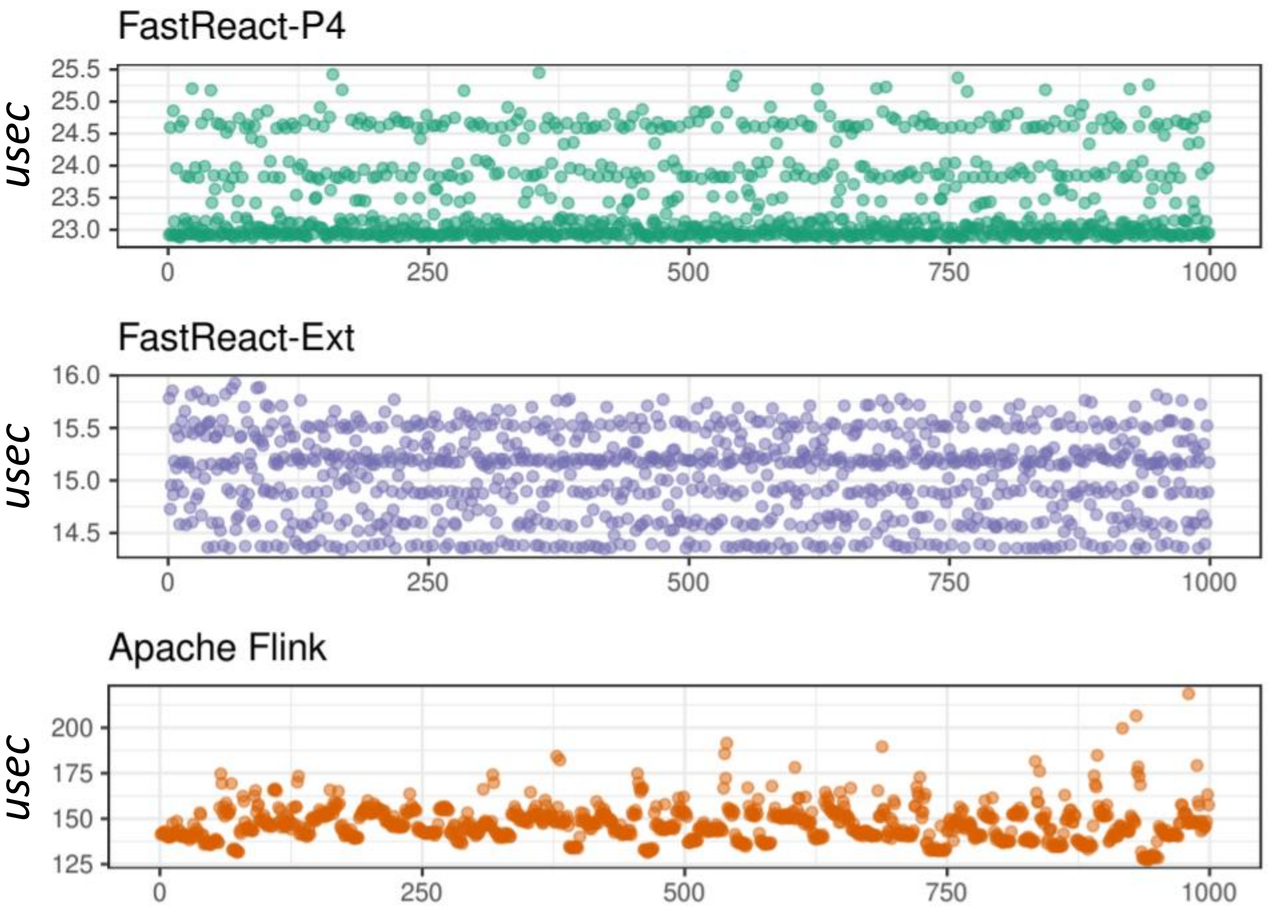


Fig. 14: Latency measurements of FastReact-PS-P4 running on the T4P4S switch, varying the disjunctive and conjunctive logic complexity. Low throughput case, preventing queue buildup.



Device	Min	Mean	Max	sd	99%
FastReact-PS-P4	22.84	23.47	25.97	0.68	25.21
FastReact-PS-Ext	14.33	15.01	16.53	0.41	15.78
Apache Flink	123.88	146.69	6270.64	67.94	171.83

Latency comparison. Values are in usec.

Evaluation - Throughput

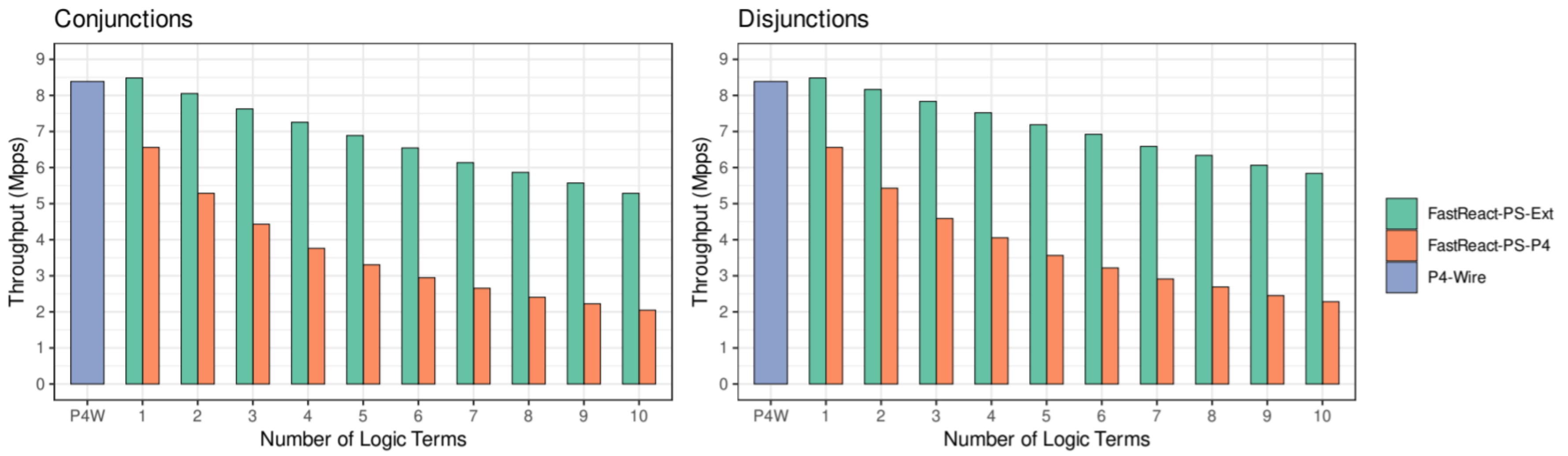


Fig. 12: Throughput measurements of FastReact-PS-P4 and FastReact-PS-Ext, varying disjunctive and conjunctive logic complexity with filled logic tables.

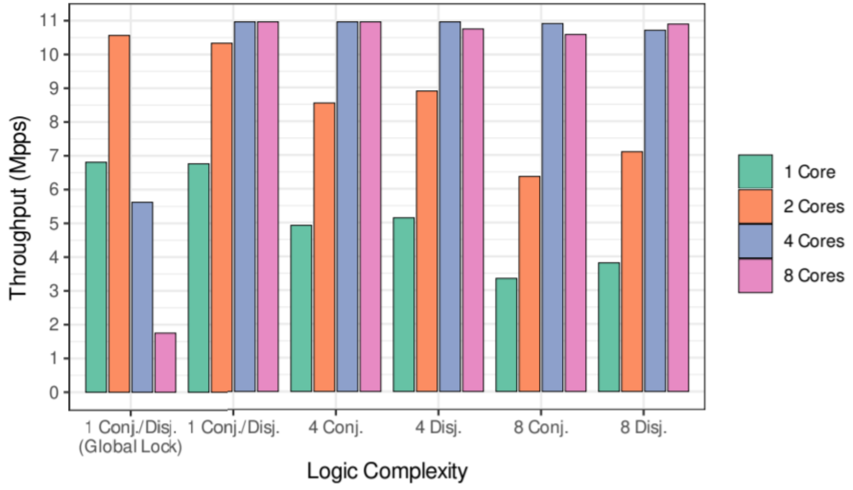
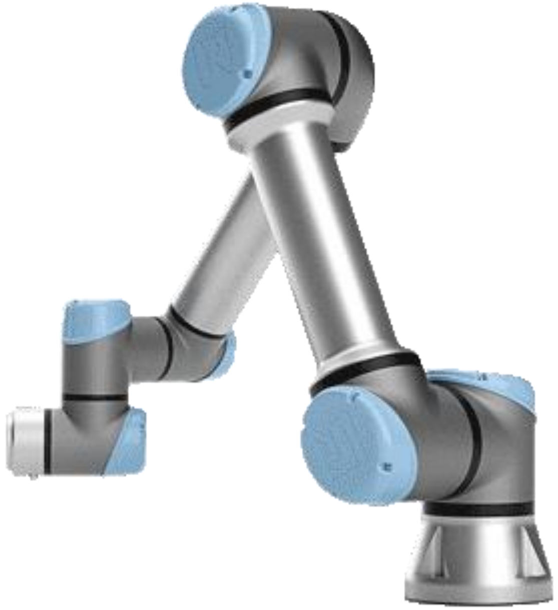


Fig. 16: Throughput measurements of FastReact-PS-P4 running on the T4P4S switch, varying disjunctive and conjunctive logic complexity.

Conclusion

- FastReact works by
  - moving part of the industrial control logic to the core or edge switch.
  - This can reduce network latency and data usage.

- Future work
  - Programmable switches are more than simple packet forwarding elements
  - Fitting well to several industrial UC
  - Integration with upcoming TSN extensions of Ethernet
    - integrate TSN extensions into the P4 pipeline.



communication networks laboratory

Eötvös Loránd University