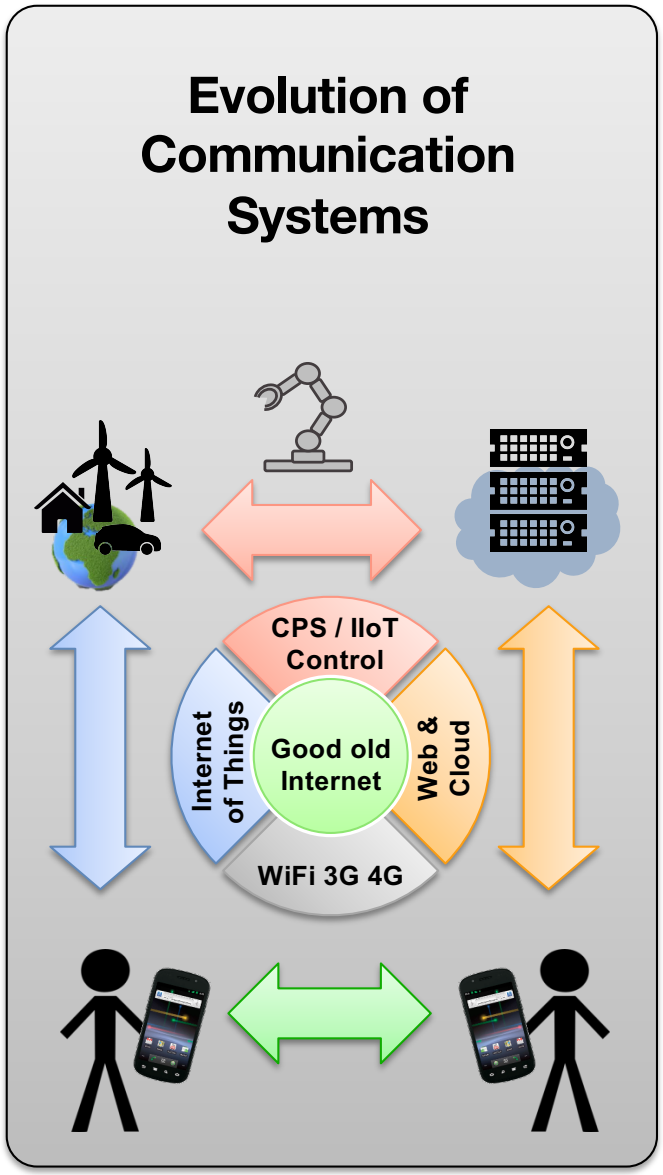
From Reflexes to In-Network Processing

Enabling Ultra-low Latency and High Reliability for Cyber-physical Networking

Klaus Wehrle, joint work by the COMSYS team

Motivation



Cyber-physical networking

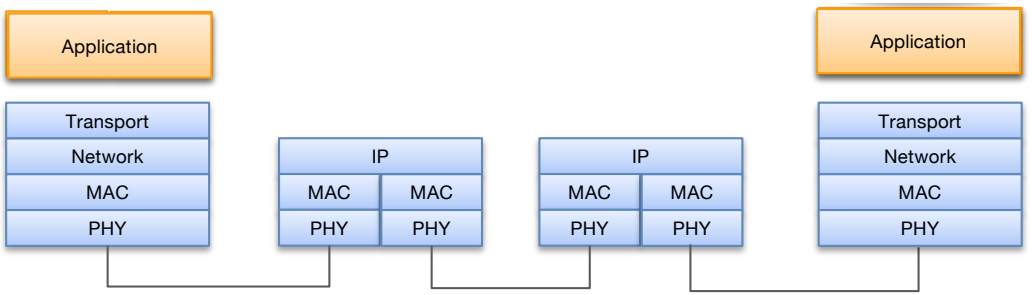
- Remote control of machines, humans not involved
- High Precision required
- Challenges: **Ultra-low latency** and **high reliability**

Human-centric communication

- Humans are ‘slow’ and compensate (comm. & system) errors
- Latency (<20ms) was never a big issue

Cyber-Physical Networking – Challenge 1: Ultra-low Latency

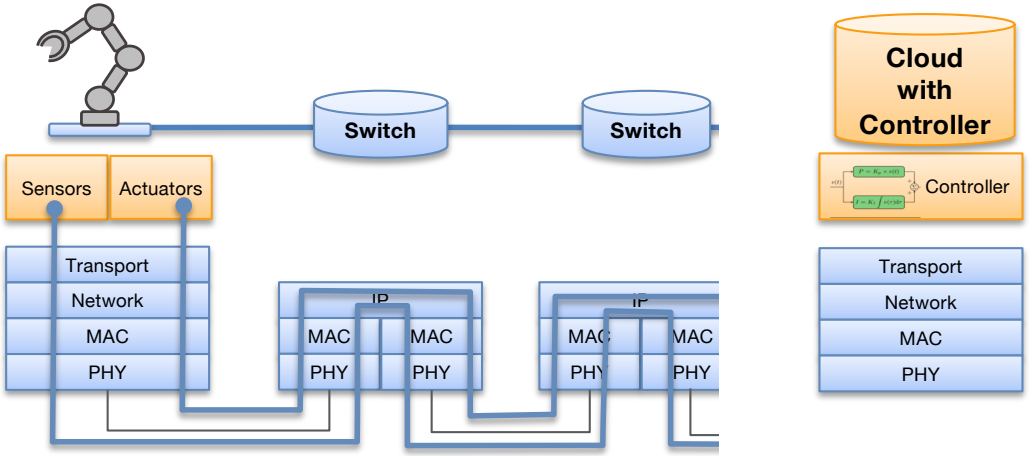
Challenge: Ultra-low latency



Cyber-Physical Networking – Challenge 1: Ultra-low Latency

Challenge: Ultra-low latency

- **Problem 1: Physical distance**
  - Solution: Reduce the distance 😏
    → Edge Cloud reduces horizontal distance, e.g. in 5G
  - Edge cloud still not nearest to plant

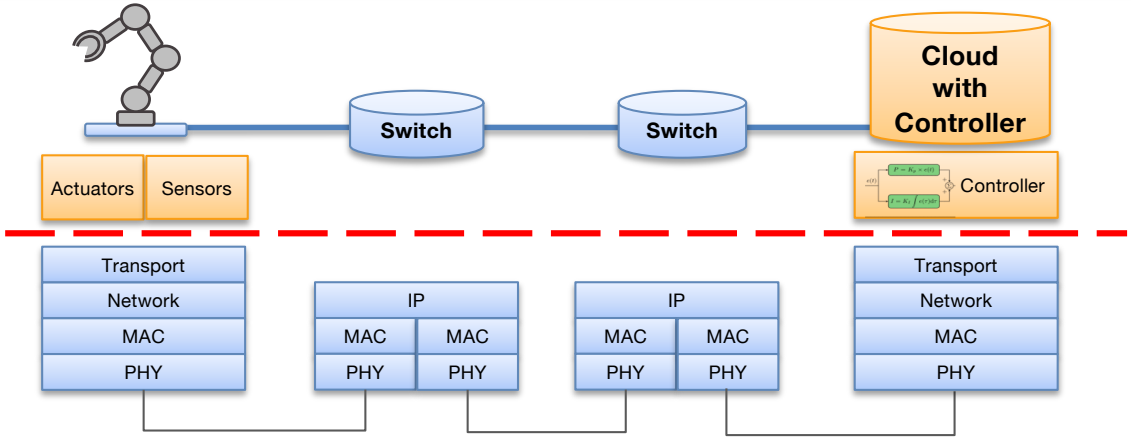


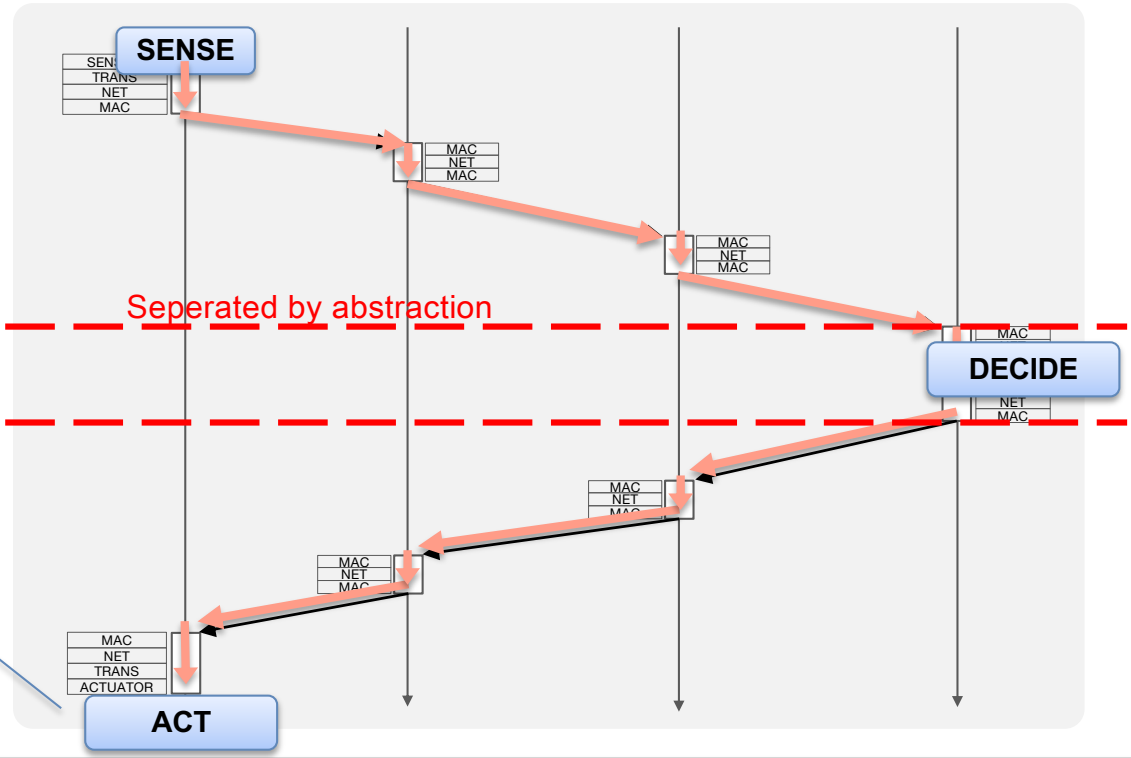
Cyber-Physical Networking – Challenge 1: Ultra-low Latency

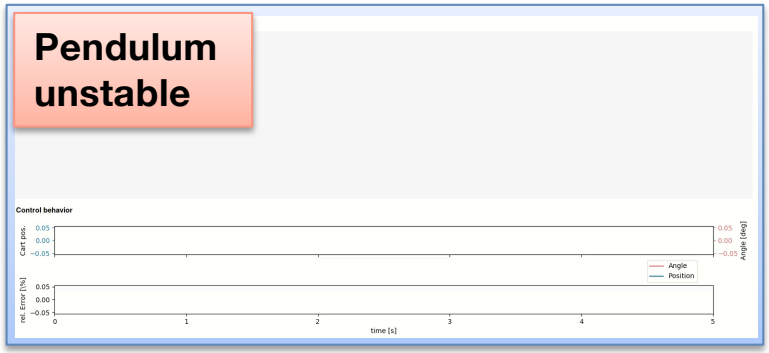
- **Problem 2: Vertical distance of the layered system approach**
  - Control and Communication layers heavily abstract from each other
    - Control is just an(other) application
    - Network seen as (stable) black box
  - No joint optimization possible

→ **Cyber-Physical Networking Initiative**

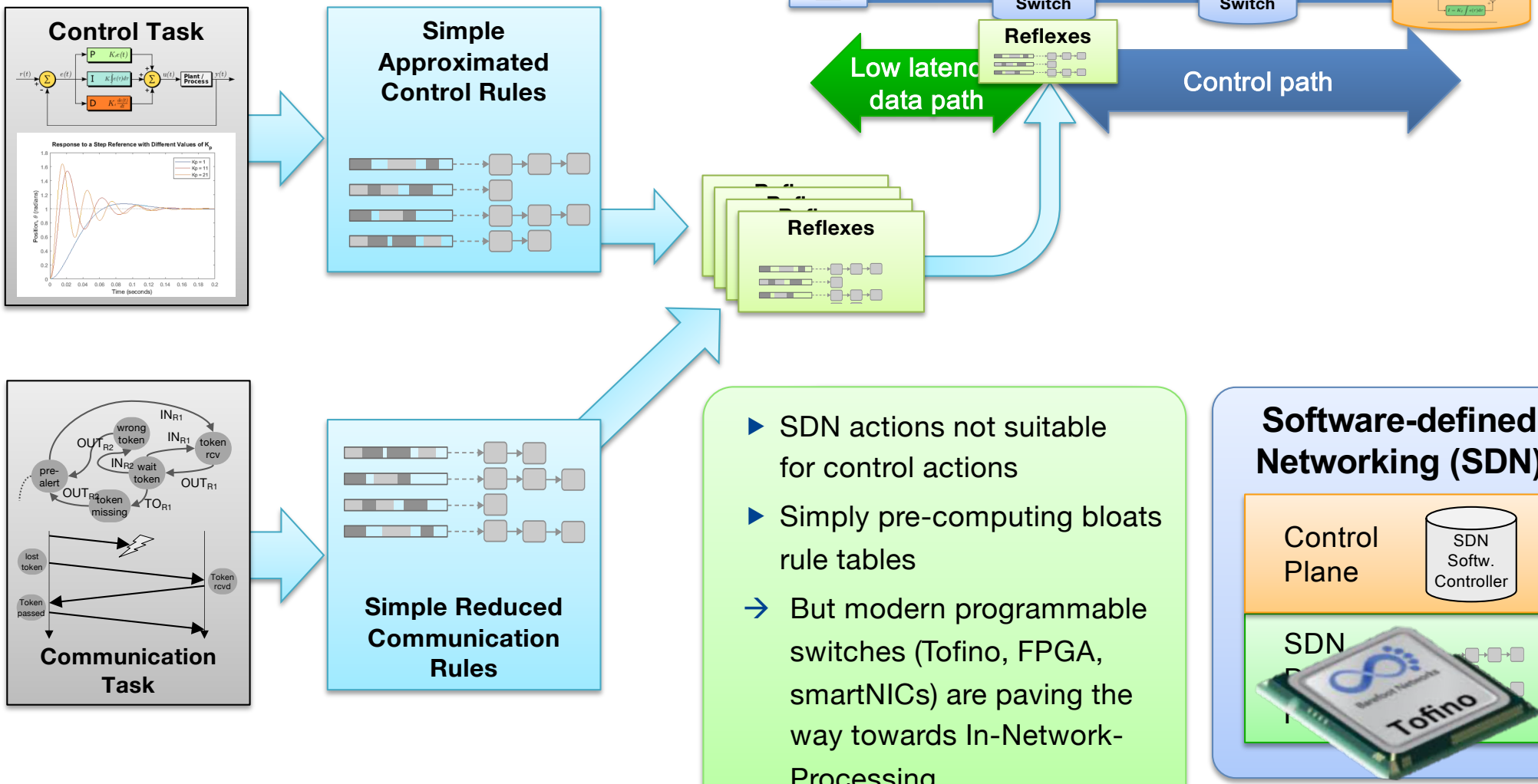
  - **Co-Designed Networked Control**







Co-Designing Communication and Control



Accuracy-Latency-(Throughput) Trade-Off

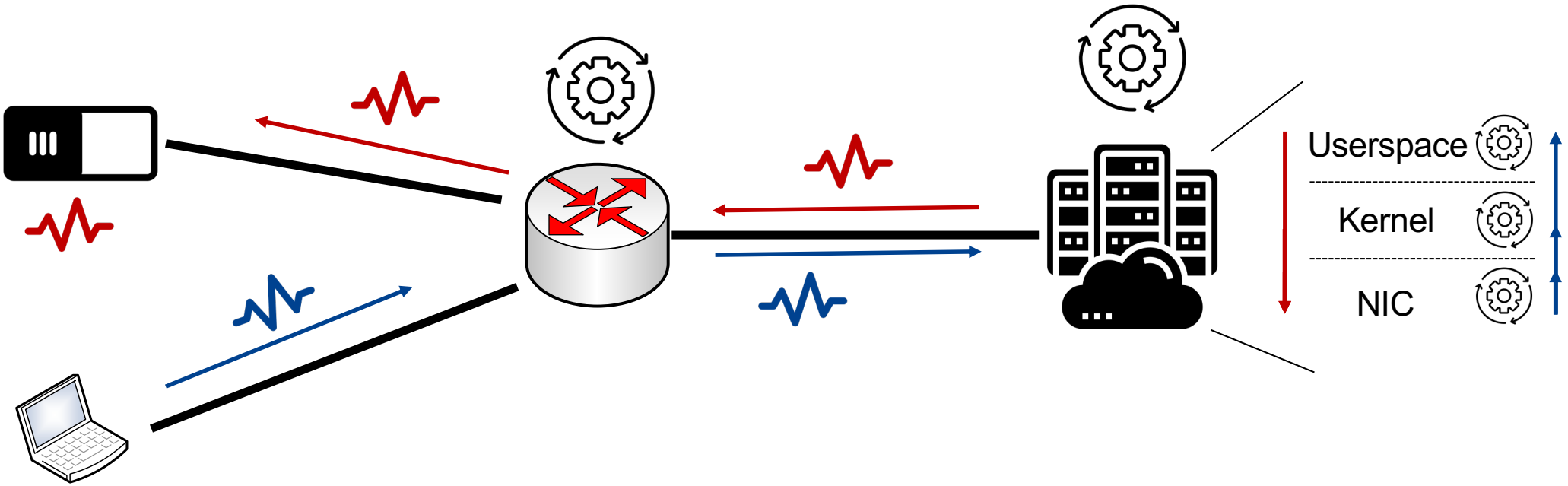
- **Computing Platforms**
  1) End-host computations
  2) In-kernel processing (XDP, TC)
  3) SmartNIC
  4) Switch (e.g. Tofino)

Faster & more predictable

But very restricted operations

Less computational restrictions

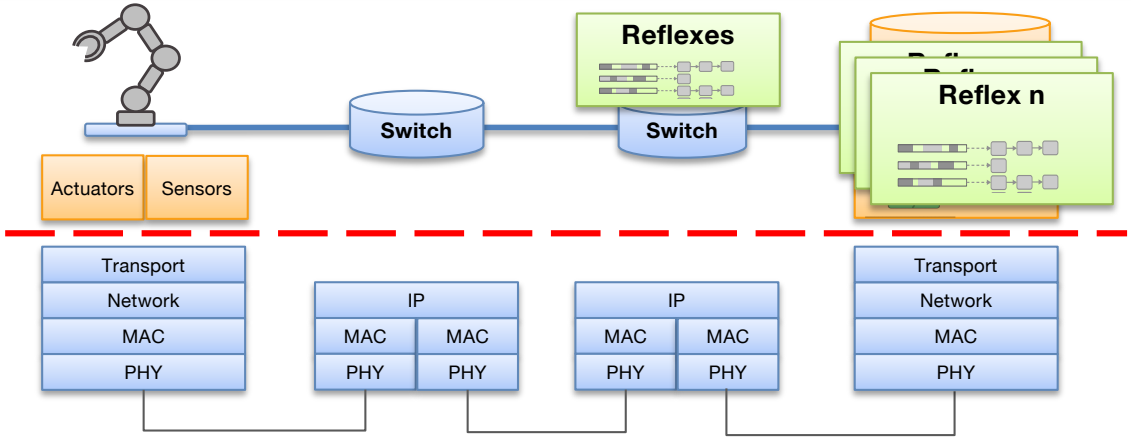
But more unpredictable and slower execution

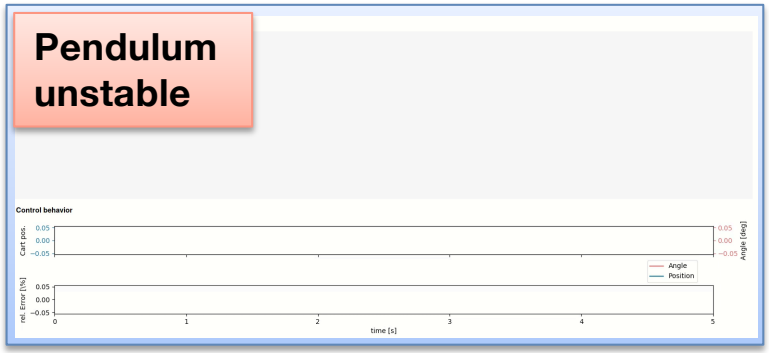


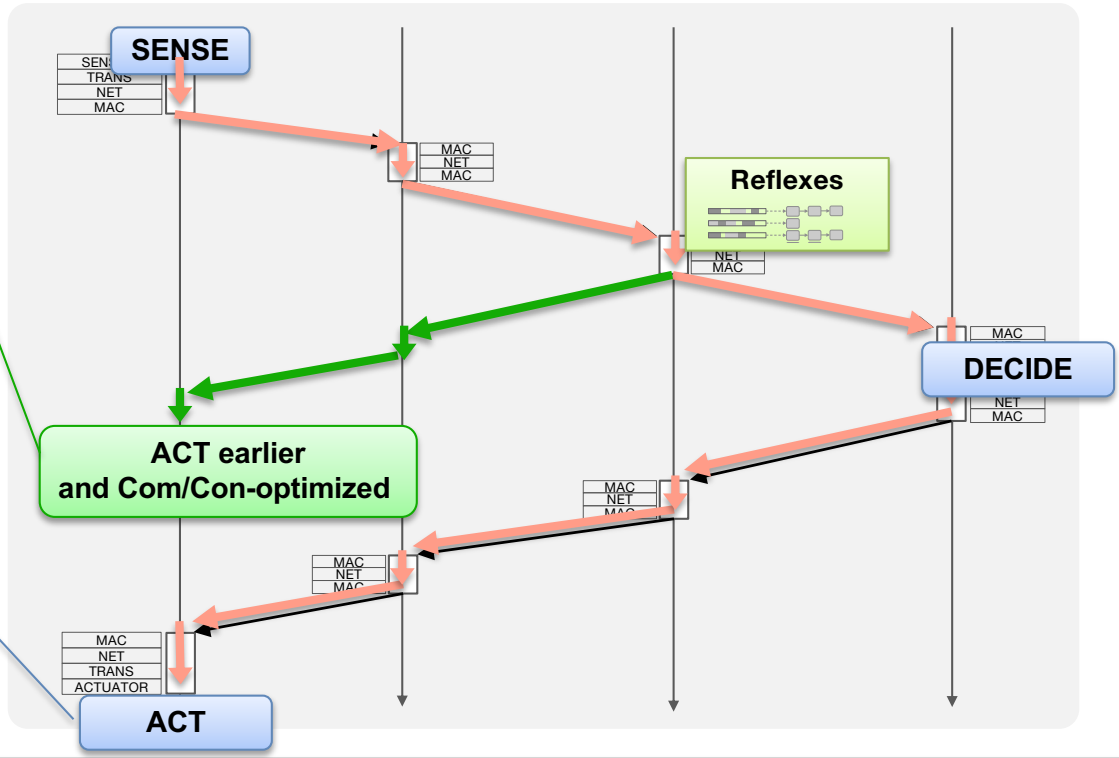
REFLEXES – A Co-Designed Architecture for In-Network Control

Challenge:

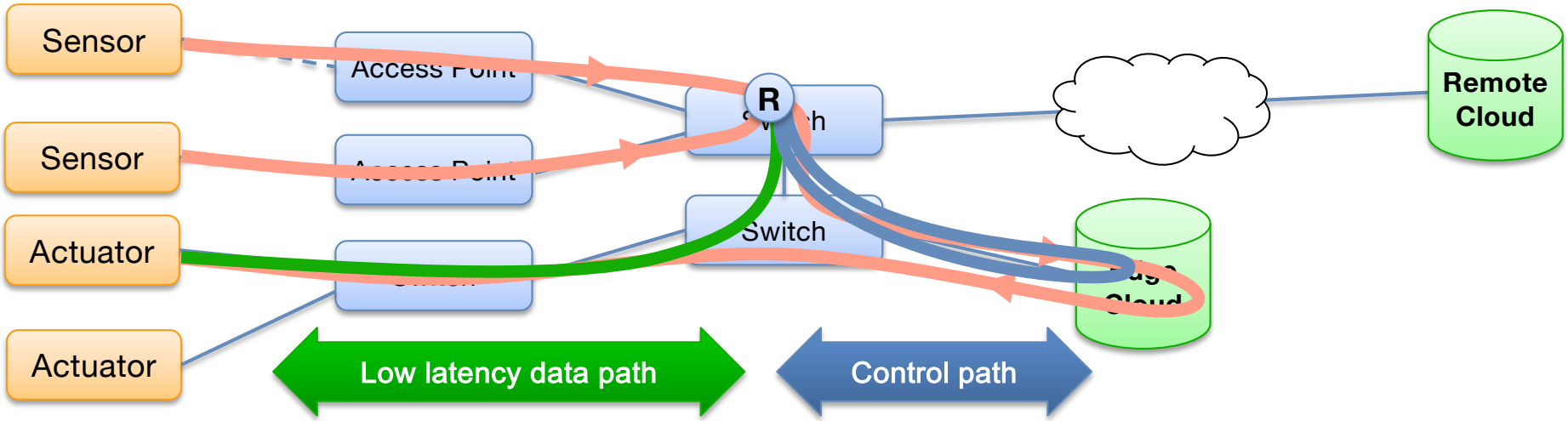
- Make joint decision on control and communication decision
- Combine possible reactions to many reflex candidates and
- push reflexes nearer to plant







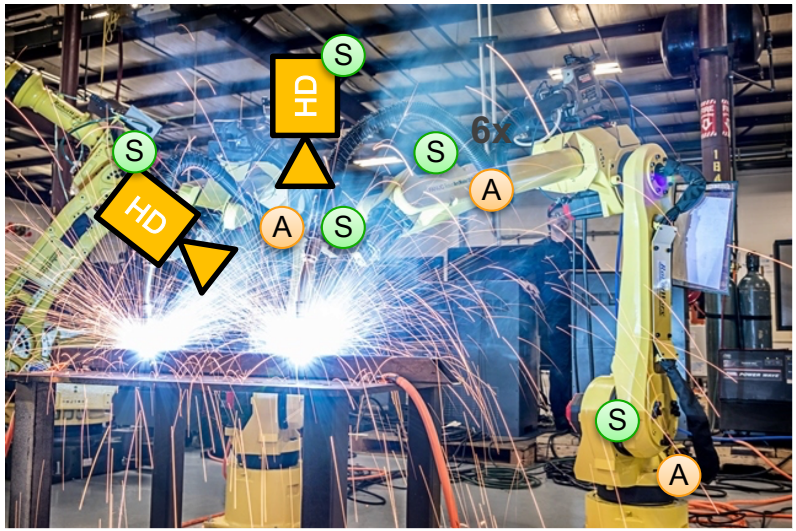
General REFLEXES Framework



- **Task separation: Separating data processing and coordination**
  - Fast and simple reactions based on INP
    - Use computation in the network to execute simple tasks
    - Push simplified control algorithm (reflex) to the switch
  - Main control algorithm stays in edge cloud to do delay-insensitive adaptation
    - Slow path processing, coordination and state management stays in the cloud
    - Cloud updates reflex if necessary, e.g. latency change, process is mobile, etc.

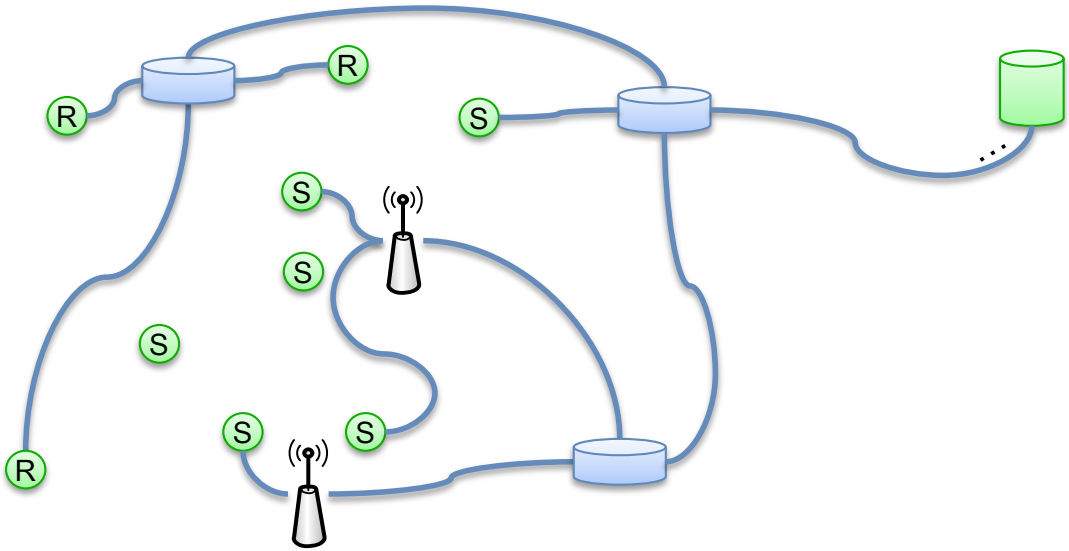
Two Real-world Examples (Cluster Internet of Production)

- **Arc welding robots**



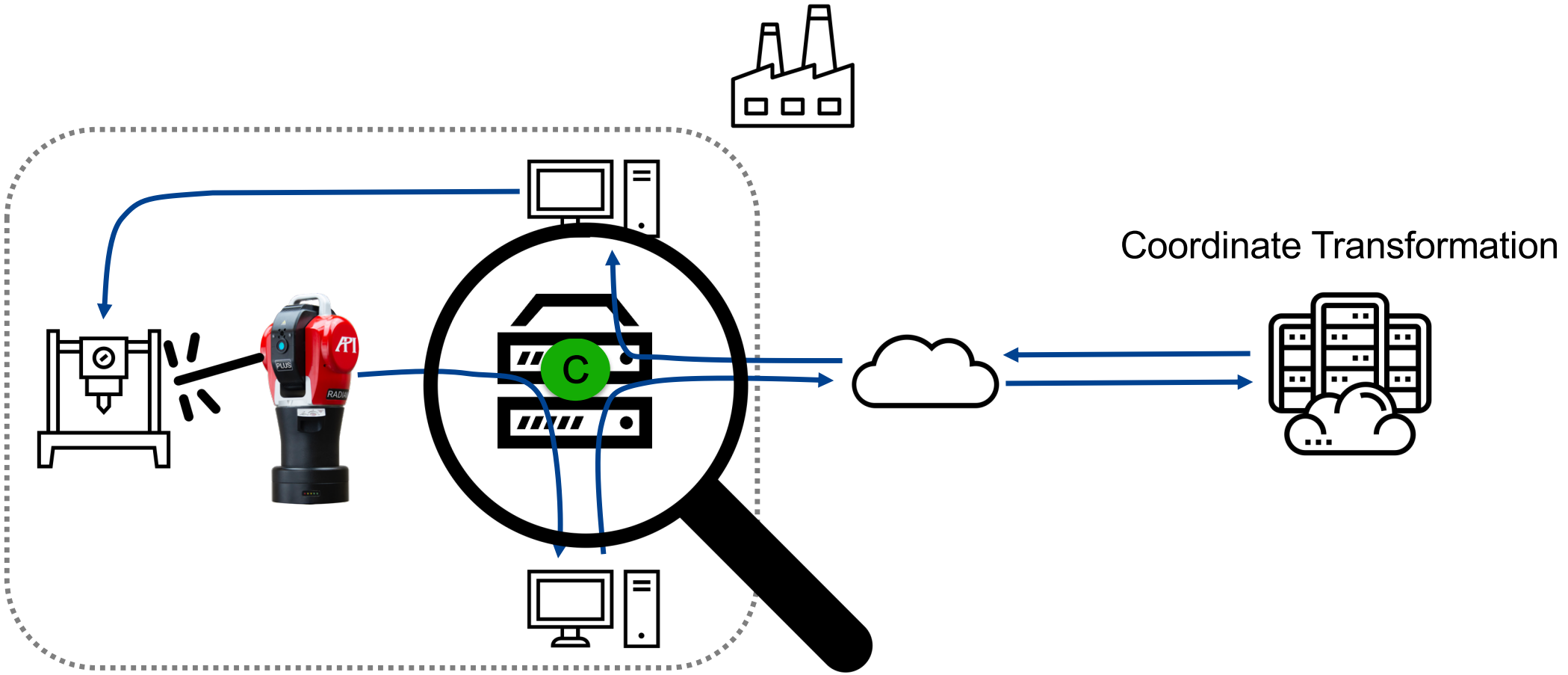
- **Control loops**
  - Single-digit millisecond latency
  - Multiple sensor sources
    - HD and infrared camera
    - Current draw of light arc
  - Actuators
    - Robot positioning
    - Light arc voltage

- **Mobile robot cooperation**



- **Control loops**
  - Positioning coordinated by many inputs
    - e.g. indoor coordinate system, camera, etc.
    - In-network coordinate transformation
  - Human in the loop detection (safety zone)
    - e.g. logical safety loop among cameras, lasers, Lidar
  - Robot interaction via multiple sensors
  - Augmented Reality …

Networked Control – Real-World Example *Laser Tracker*



In-Network Coordinate Transformation – Fundamentals

Challenge: Coordinate transformation (Spherical to Cartesian)

$$\begin{bmatrix} r \sin\theta \cos\varphi \\ r \sin\theta \sin\varphi \\ r \cos\theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- **Restricted Fixed-Point Arithmetic**
  - $\pm [0 \ldots 2^{d}].[0 \ldots 2^{31-d}]$
  - Choose fixed point to
    - ensure range is *sufficiently large* (application range)
    - *maximize* fractional part (required accuracy)

- **Approximate trigonometric functions**
  1. Chebyshev polynomials
  2. Table Lookup

$\sin\theta \rightarrow$

θ	sin
0.000000	0
0.000488	0.000488
0.000977	0.000977
…	

  - Problem: Large table space needed
  - Use sum of angle identity

$$\sin a + b = \sin a \cdot \cos b + \cos a \cdot \sin b$$

In-Network Coordinate Transformation – Fundamentals

Challenge: Coordinate transformation (Spherical to Cartesian)

$$\begin{bmatrix} r \sin\theta \cos\varphi \\ r \sin\theta \sin\varphi \\ r \cos\theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- **Restricted Fixed-Point Arithmetic**
  - $\pm [0 \dots 2^{d}].[0 \dots 2^{31-d}]$
  - Choose fixed point to
    - ensure range is *sufficiently large* (application range)
    - *maximize* fractional part (required accuracy)
- **Approximate trigonometric functions**
  1. Chebyshev polynomials
  2. Table Lookup

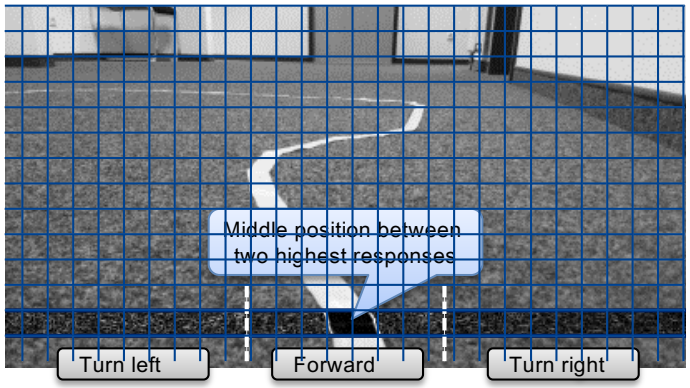
θ_{high}	$(\sin\theta_{high}, \cos\theta_{high})$
0.000000	(0,1)
0.000488	(0.000488, 0.999999)
0.000977	(0.000977, 0.999995)
…	
6.282714	(-0.000470, 0.999999)

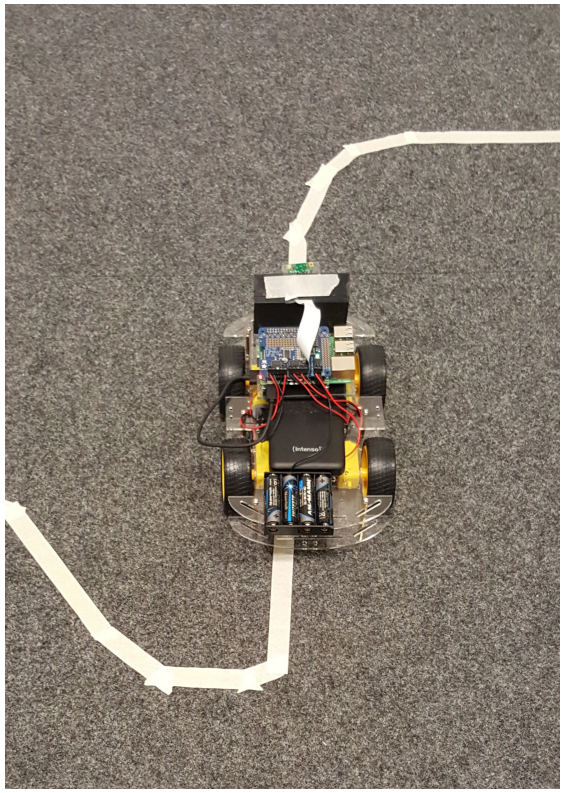
θ_{low}	$(\sin\theta_{low}, \cos\theta_{low})$
000000	(0,1)
000001	(2.980232e-8, 1)
000002	(5.960464e-8, 1)
…	
000488	(-0.000470, 0.999999)

$$\begin{aligned} \sin 6.282714000002 &\approx -0.000471 \\ &\approx \sin\theta_{high} \cdot \cos\theta_{low} + \cos\theta_{high} \cdot \sin\theta_{low} \\ &\approx -0.000470 \cdot 1 + 0.999999 \cdot 5.960464 \cdot 10^{-8} \\ &\approx -0.000470 \end{aligned}$$

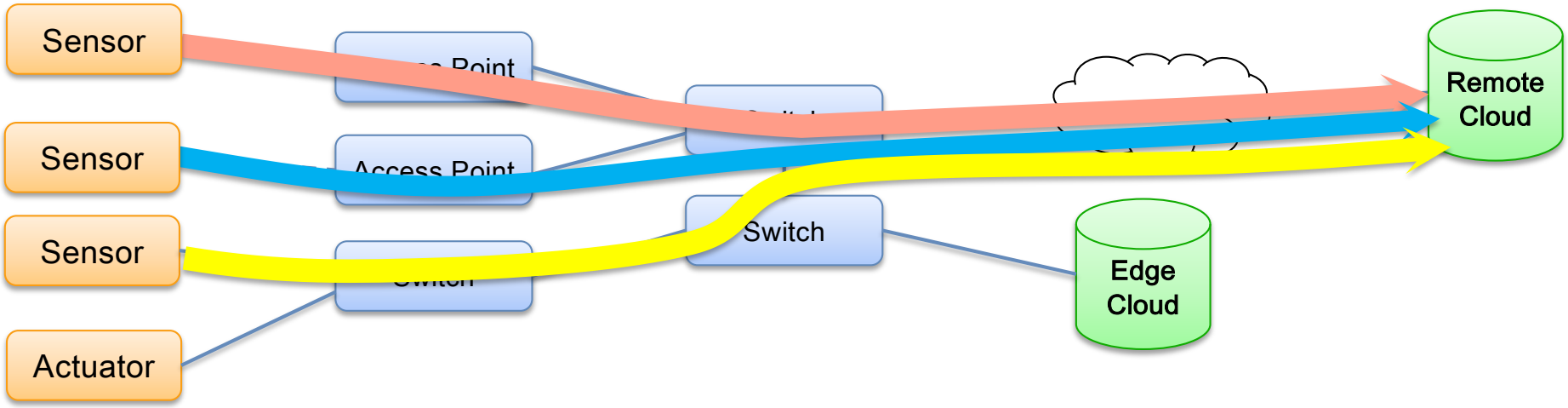
In-Network Image Processing

- **Low-latency computer vision often needed**
  - Fast reactions to the environment
- **Camera images rarely fit into single packet**
  - Use local computation strategies like convolution



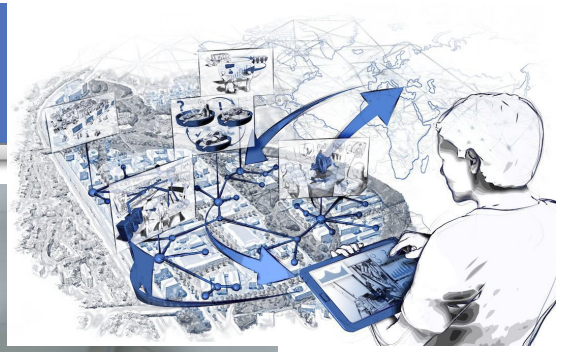


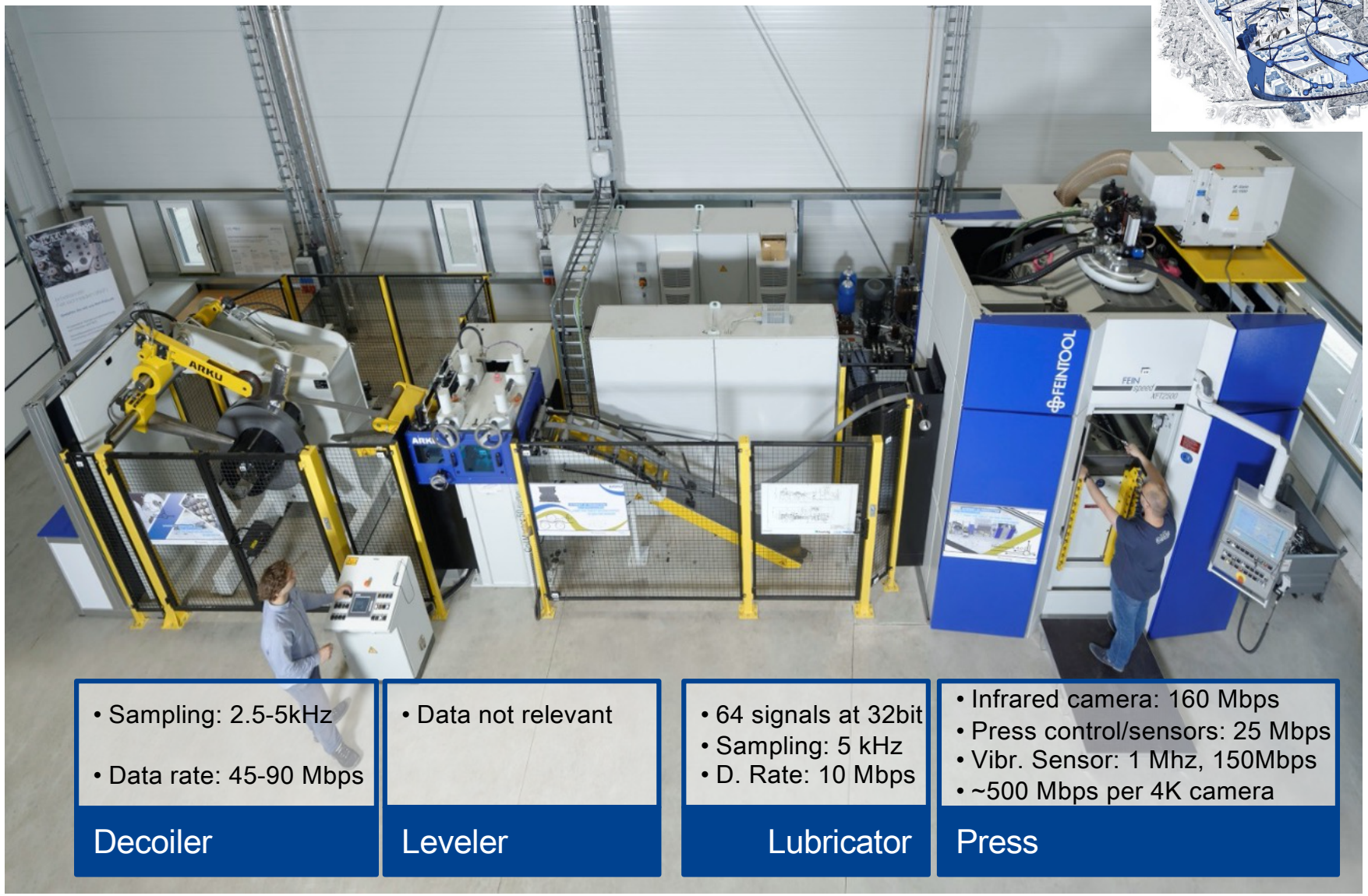
Data Stream Processing



- **Collection and Analysis of Process Data**
  - Data-driven improvement of production and efficiency
    - Collect every data item the process and machines are emitting
    - Derive immediate feedback on process status and product quality
    - Realtime-feedback for production process
  - Problem: Data rate of produced process data

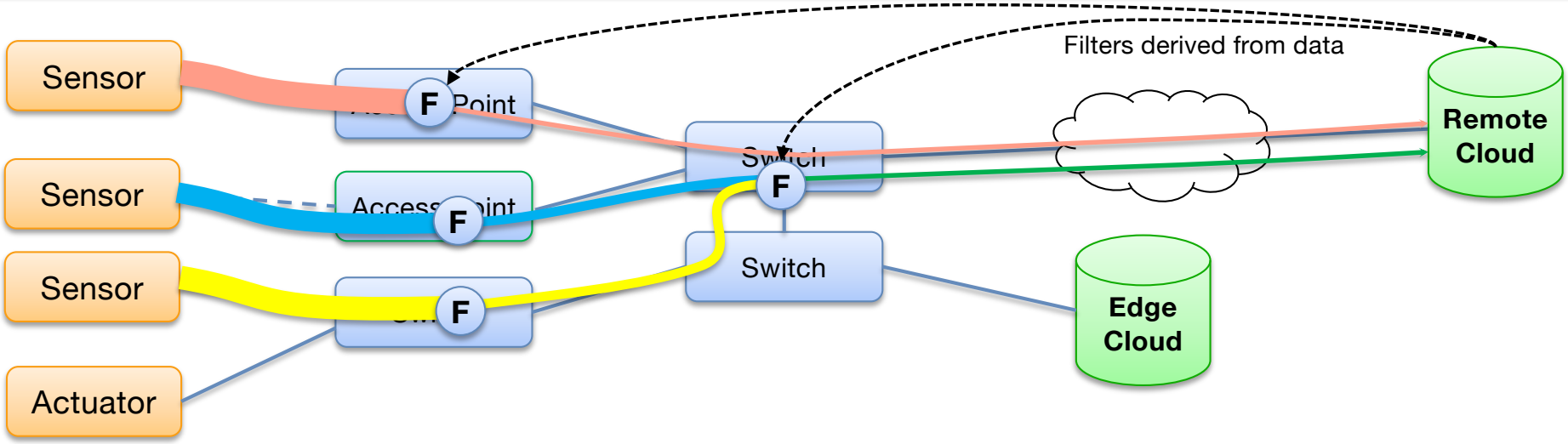
Real-world example: Fine Blanking





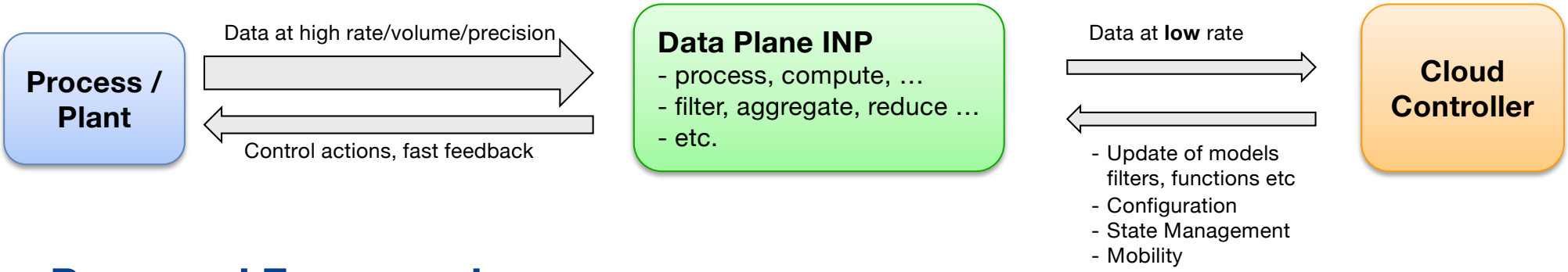
Decoiler	Leveler	Lubricator	Press
• Sampling: 2.5-5kHz • Data rate: 45-90 Mbps	• Data not relevant	• 64 signals at 32bit • Sampling: 5 kHz • D. Rate: 10 Mbps	• Infrared camera: 160 Mbps • Press control/sensors: 25 Mbps • Vibr. Sensor: 1 Mhz, 150Mbps • ~500 Mbps per 4K camera

Data Stream Processing at 40 Gbps Line-Rate

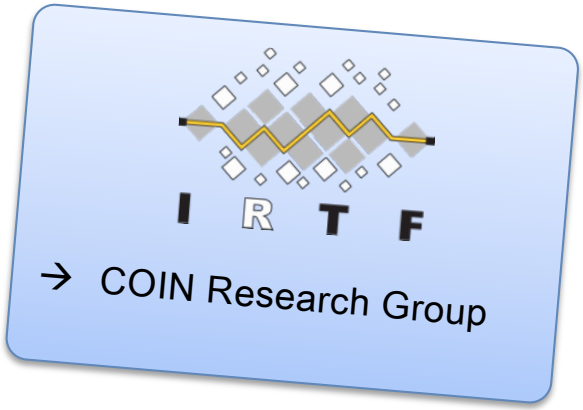


- **Collection and Analysis of Process Data**
  - Data-driven improvement of production and efficiency
    - Collect every data item the process and machines are emitting
    - Derive immediate feedback on process status and product quality
    - Realtime-feedback for production process
  - Problem: Data rate of produced process data
- **Reduce/process the data as early as possible in the network**
  - Apply filtering, aggregation, compression, classification on the data path

Proposed Framework for IRTF: Computing in the Network



- **Proposed Framework**
  - Enable computation in the network elements (switches, smartNICs, access points, etc)
    - For simple control tasks
    - For filtering, aggregating, etc. data on the path to the cloud
    - For boosting data analysis in a data center (not discussed here)
  - Hierarchical placement of computational tasks
    - Simple and predictive computation in the **network**
      - Used to satisfy tight constraints (e.g. low latency response)
    - Long-term computation, state management and coordination in the **cloud** (complex tasks)



Pre-Deployment Performance Prediction of INP-Components

Network Function Code

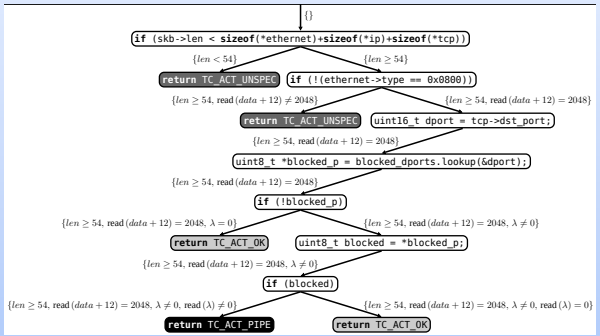
```
#include <bcc/proto.h>
#include <linux/pkt_cls.h>
#include <net/sock.h>
BPF_TABLE("hash", uint16_t, uint8_t, blocked_dports, 4096);

int act_main(struct __sk_buff *skb) {
    u8 *cursor = 0;
    struct ethernet_t *ethernet;
    struct ip_t *ip;
    struct tcp_t *tcp;
    if (skb->len < sizeof(*ethernet)+sizeof(*ip)+sizeof(*tcp)) return TC_ACT_UNSPEC;

    ethernet = cursor_advance(cursor, sizeof(*ethernet));
    if (!(ethernet->type == 0x0800)) return TC_ACT_UNSPEC;
    ip = cursor_advance(cursor, sizeof(*ip));
    if (ip->nextp != 0x06) return TC_ACT_UNSPEC;

    tcp = cursor_advance(cursor, sizeof(*tcp));
    uint16_t dport = tcp->dst_port;
    uint8_t *blocked_p = blocked_dports.lookup(&dport);
    if (!blocked_p) goto return TC_ACT_OK;
    uint8_t blocked = *blocked_p;
    if (blocked) return TC_ACT_PIPE;
    else return TC_ACT_OK;
```

Performance- & Analysis-Feedback

Symbolic Analysis

Fix bugs

Execution Tree



Analysis of Paths

Instruction-Cache- & CPU-Model

Traffic Pattern

Prediction

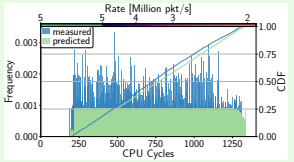
Instruction Chains

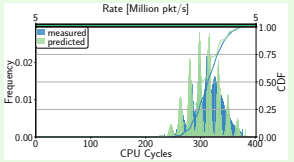
```
alloca
getelementptr
load
icmp
br
bpf_load_half
icmp
br
```

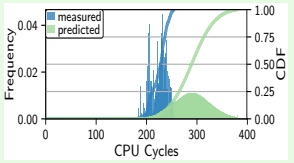
...

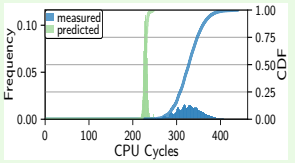
```
alloca
getelementptr
load
icmp
br
bpf_load_half
icmp
br
bpf_load_byte
icmp
br
bpf_load_half
trunc
store
bpf_pseudo
bpf_map_lookup
hash(2, 1)
icmp
br
load
icmp
br
```

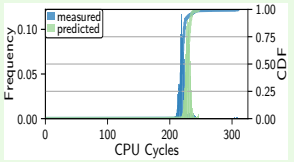
Performance Predictions





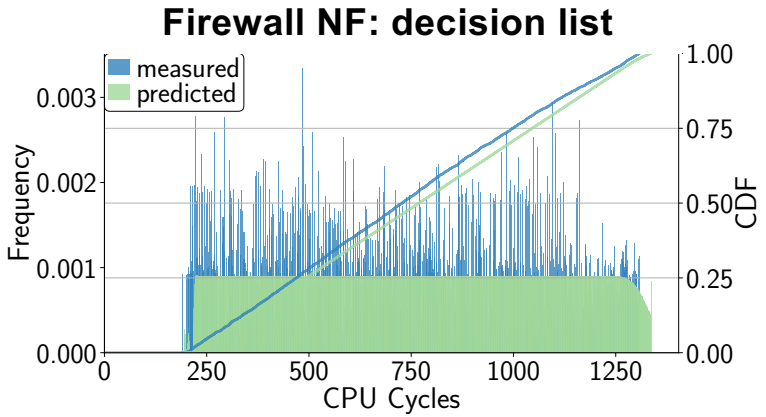


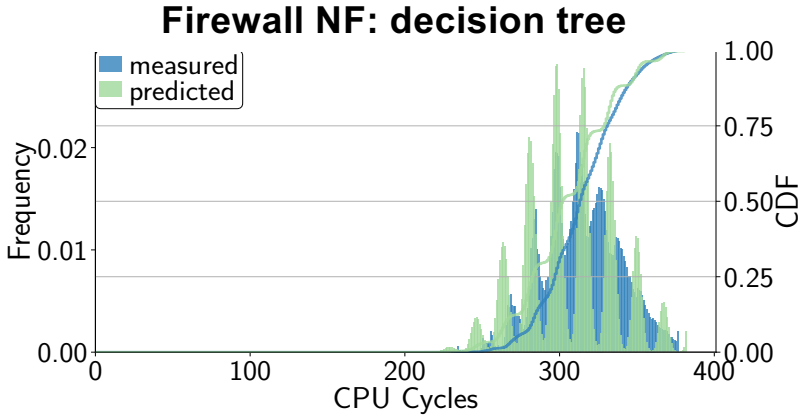




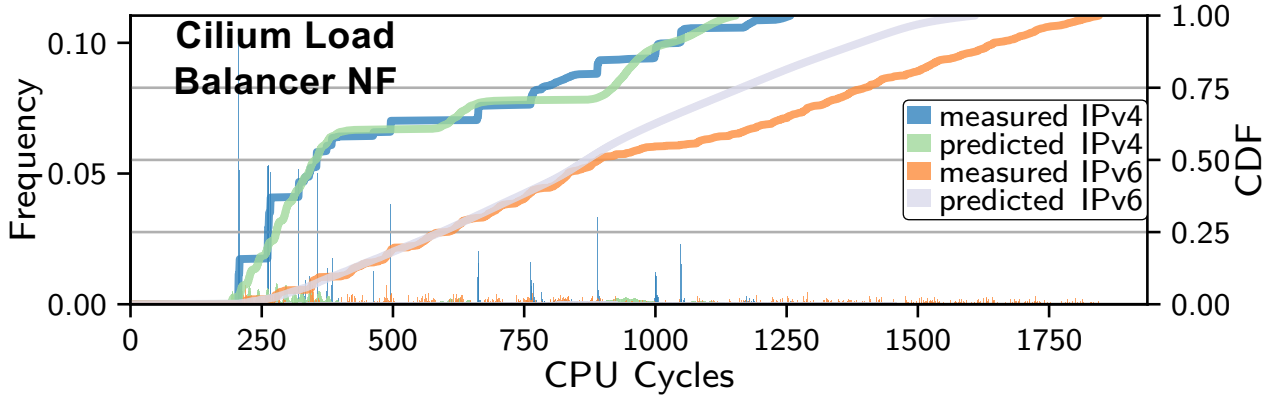
Example Findings from Symbolic Performance Analysis

- **Allows pre-deployment understanding of NF performance**
  - Impact of different implementation designs (e.g. linear list vs. decision tree)





  - Impact of different traffic patterns (e.g. regular vs. attack traffic, or IPv4 vs. IPv6)



Summary

- **On-path computation**
  - has always been considered harmful in networks
    - Erroneous, slow and unpredictable
- **But…**
  - these days latency matters more and more
    - Deviations via clouds/data centers are costly
    - Type of computation is often not too complex and not too lengthy
  - there is (more and more) hardware that can …
    - do more than just packet forwarding, counting, and dropping
    - at 40/100 Gbps line speed (depends on the architecture)
  - There are better methods…
    - to predict the (software) behavior
    - to predict the resulting performance
    - to ensure the code does what you want it to do
- **Our goal is to find out the limits and to push them further**