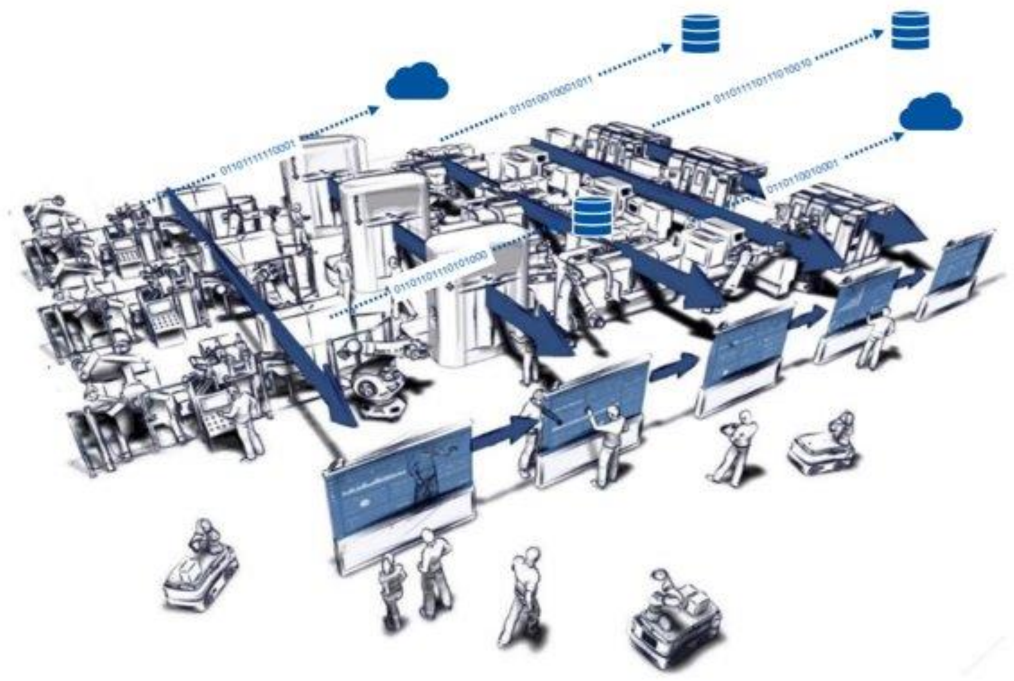
In-Network Velocity Control of Industrial Robot Arms

S. Laki[1], Cs. Györgyi[1], J. Pető[2], P. Vörös[1], G. Szabó[3]

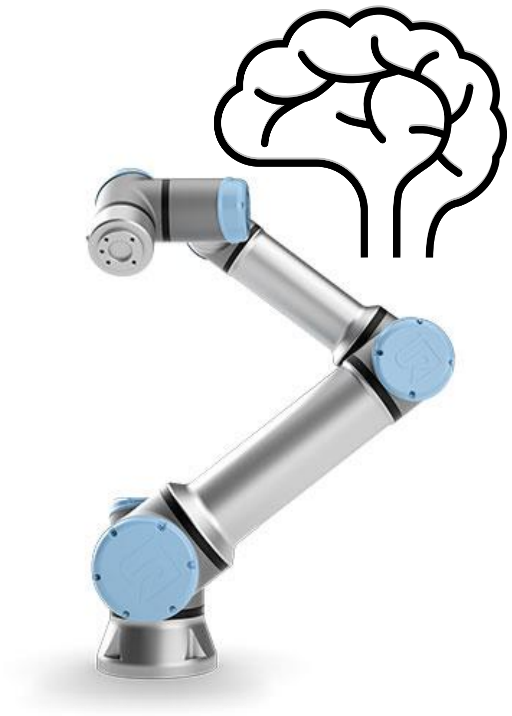
1) ELTE Eötvös Loránd University, Budapest, Hungary
2) Budapest University of Technology and Economics, Budapest, Hungary
3) Ericsson Research, Budapest, Hungary

Industry moving to the cloud

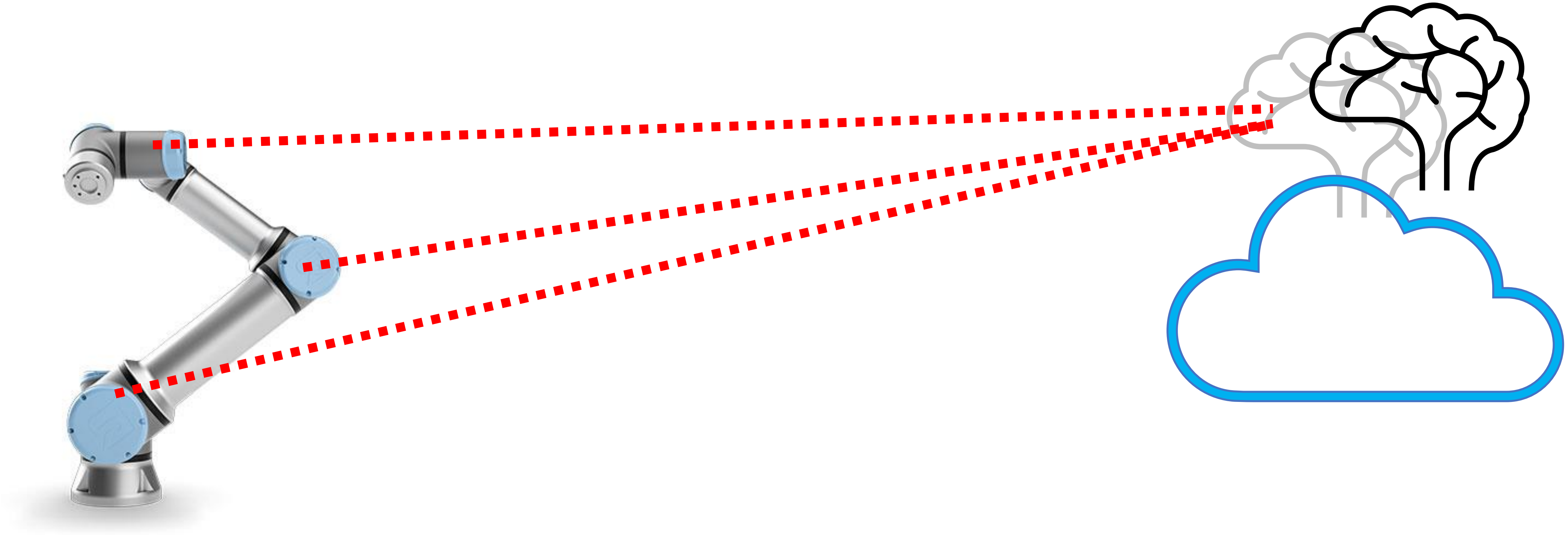
- more powerful computing resources (e.g. for solving Machine Learning (ML) tasks)
- lower cost per robot as functionalities are moved to a central cloud
- easy integration of external sensor data
- easier collaboration or interaction with other robots and machinery
- reliability of functions can be improved by running multiple instances as a hot standby in the cloud



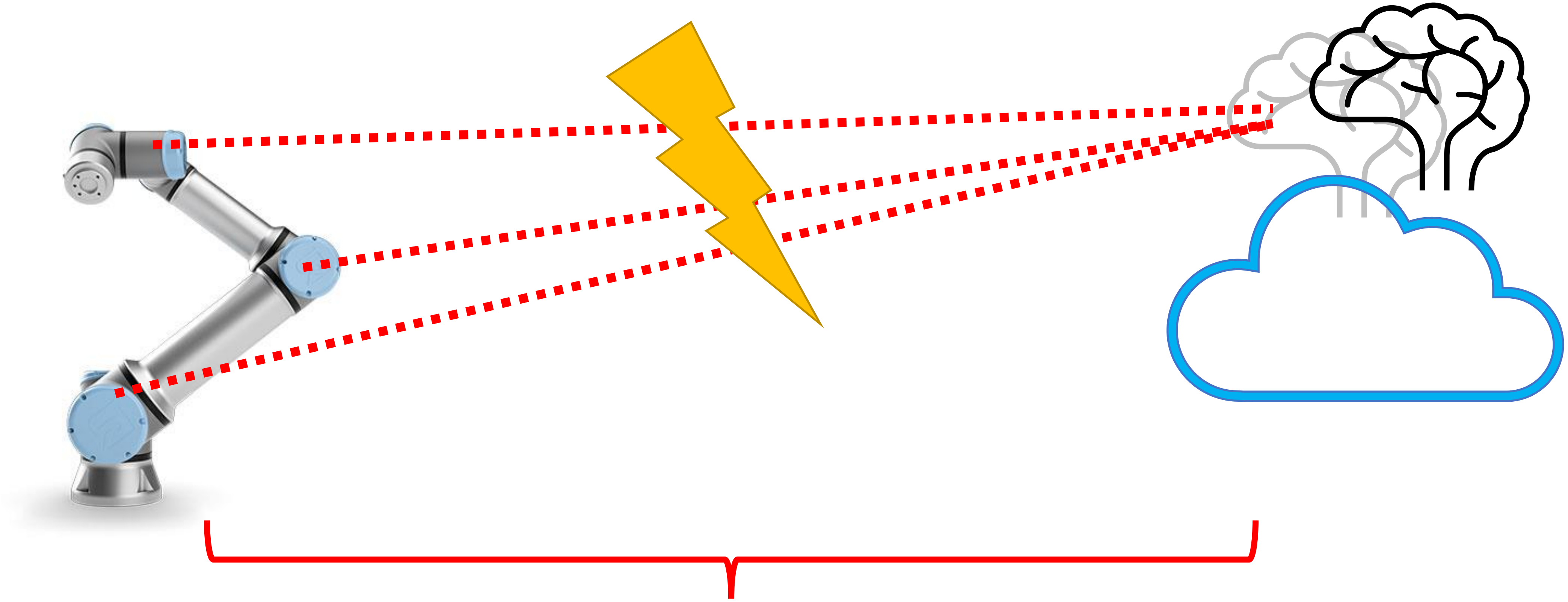
Setup #1



Setup #2
Low level control in the cloud?

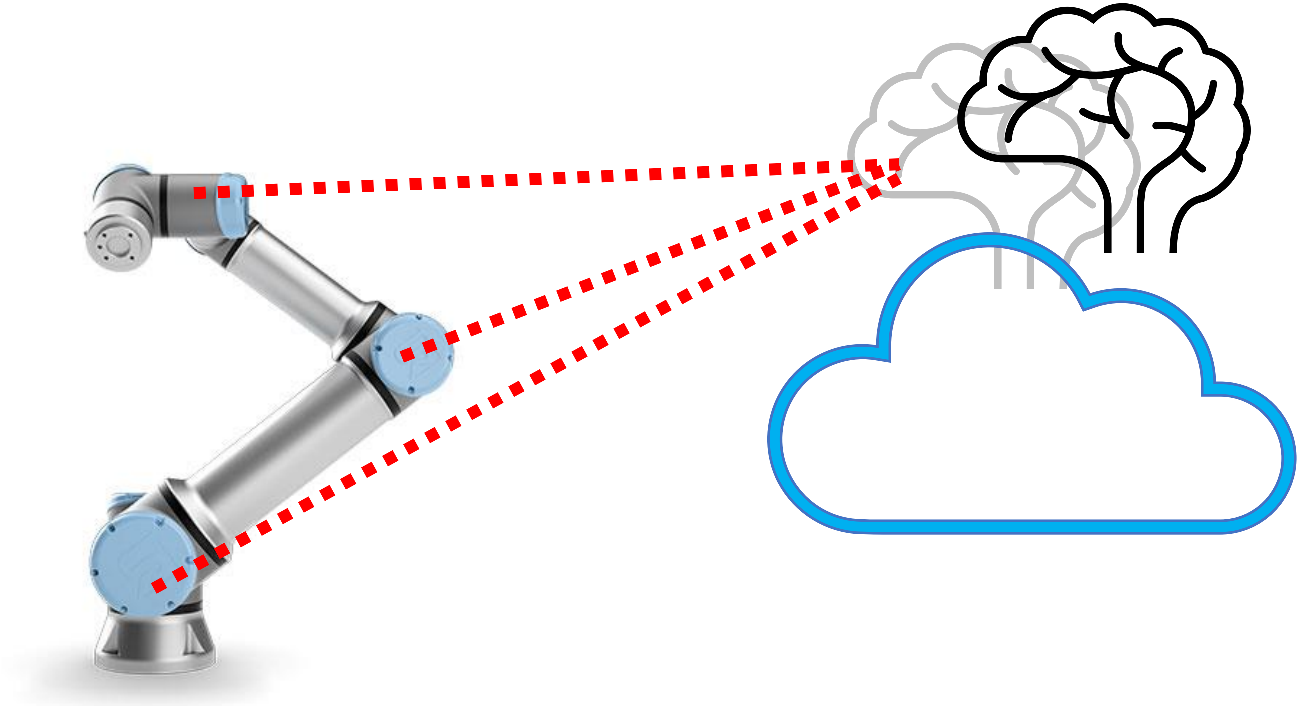


Setup #2
Low level control in the cloud?

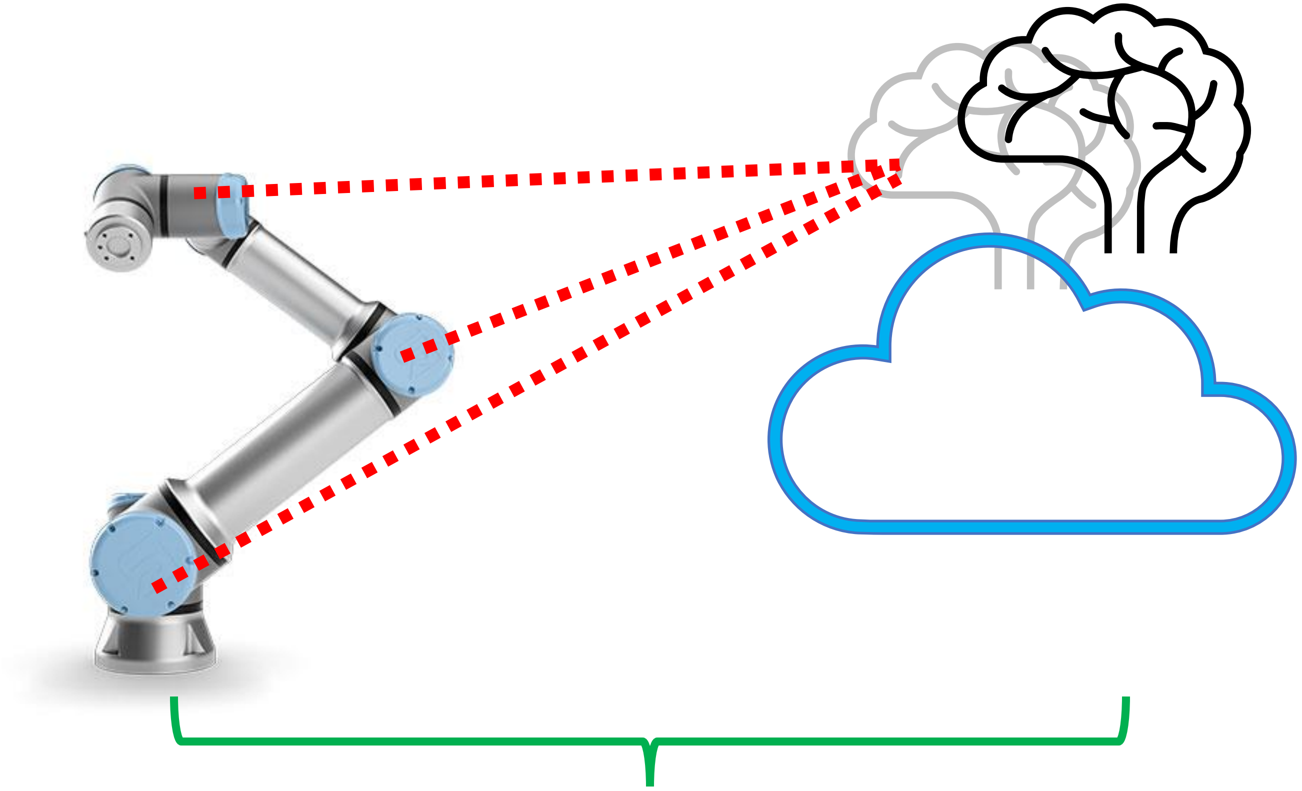


Large delay caused by signal propoagation

Setup #3 – What about edge cloud?

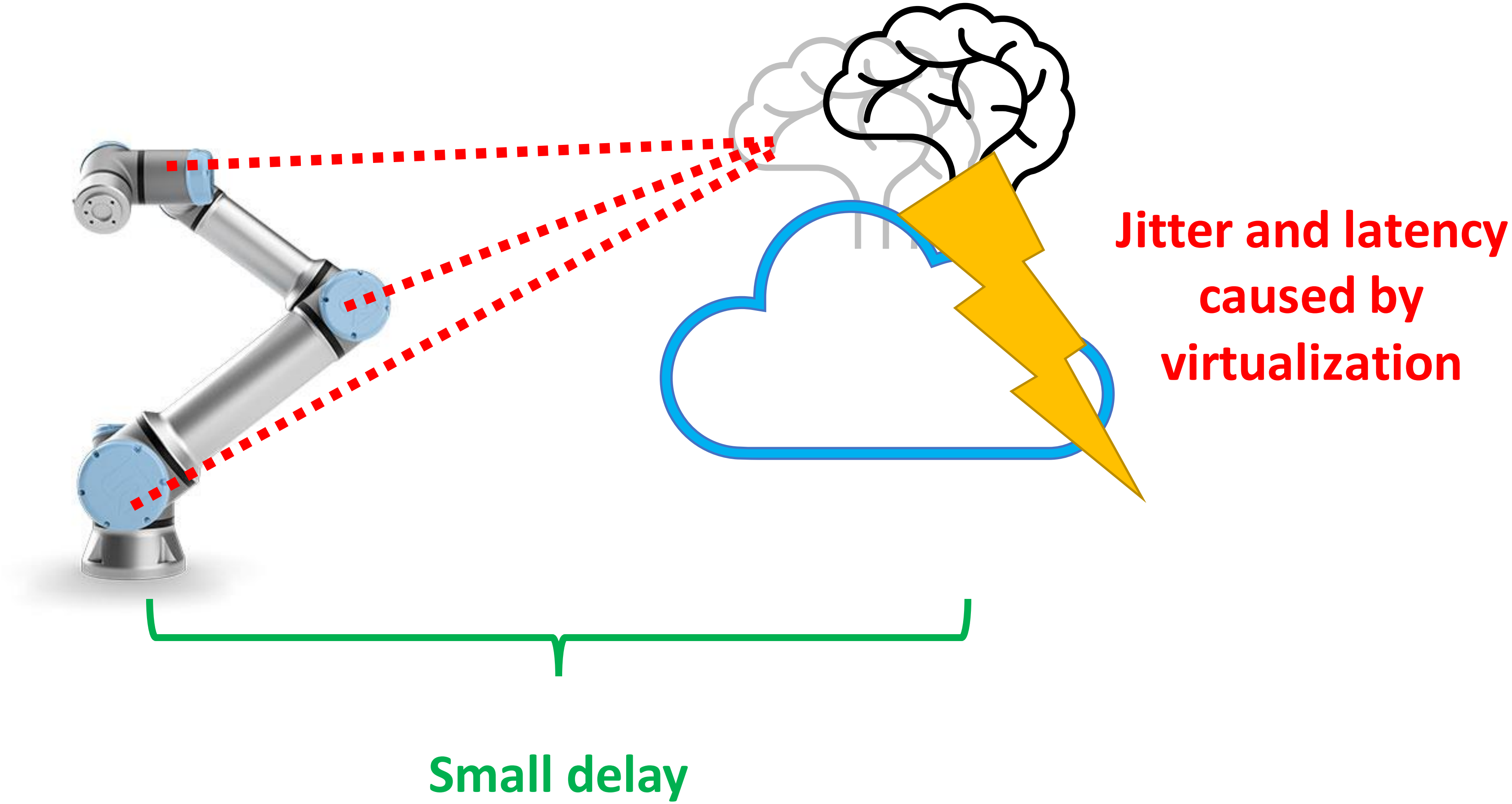


Setup #3 – What about edge cloud?

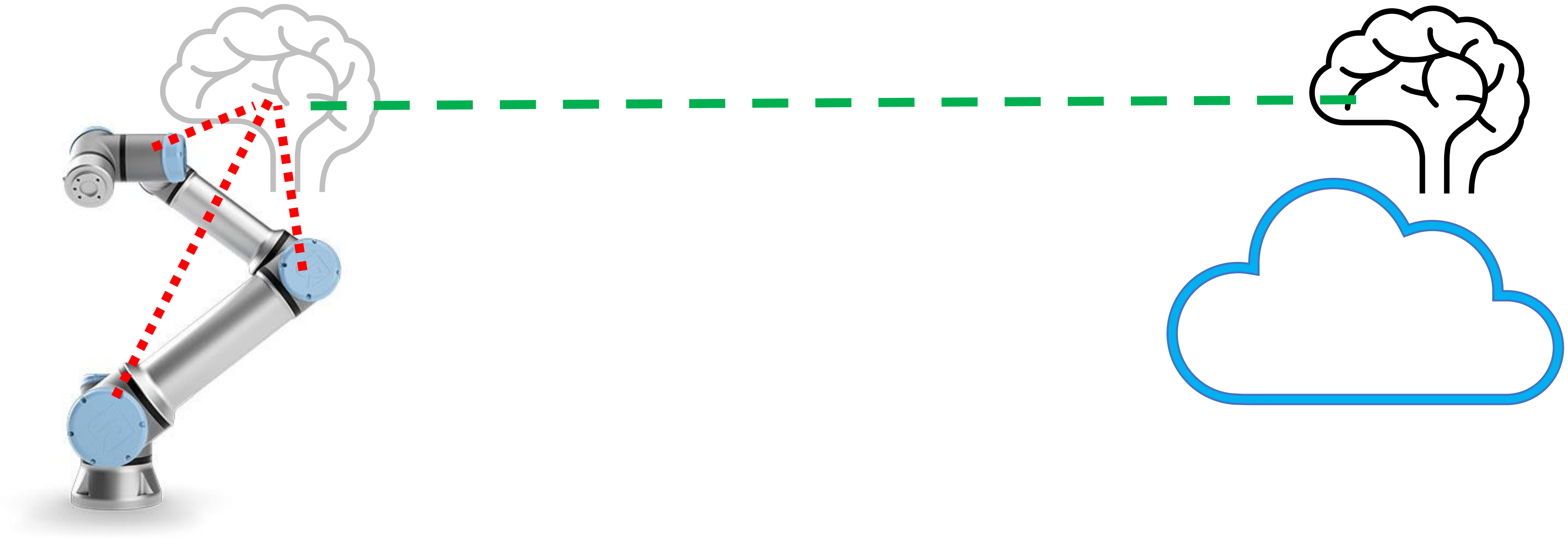


Small delay

Setup #3 – What about edge cloud?



Setup #4
High level control to the cloud

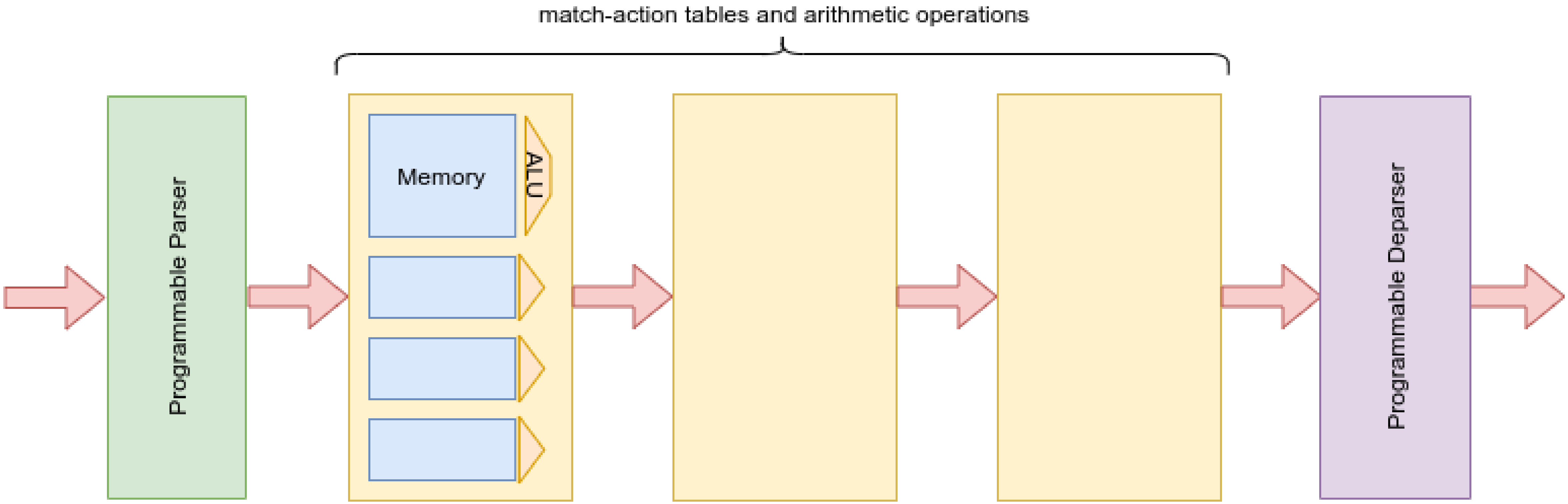


Ensuring Real-Time requirements is hard

- Requirements for low-level control
  - **Accurate timing** of sending low-level control messages (e.g., velocity vectors in velocity control) → **Low jitter**
  - **Update time** of each robot is about 2ms or less → **Low latency**

	Remote cloud	Edge cloud	Our approach
Low jitter	✗	✗	✓
Low latency	✗	✓	✓

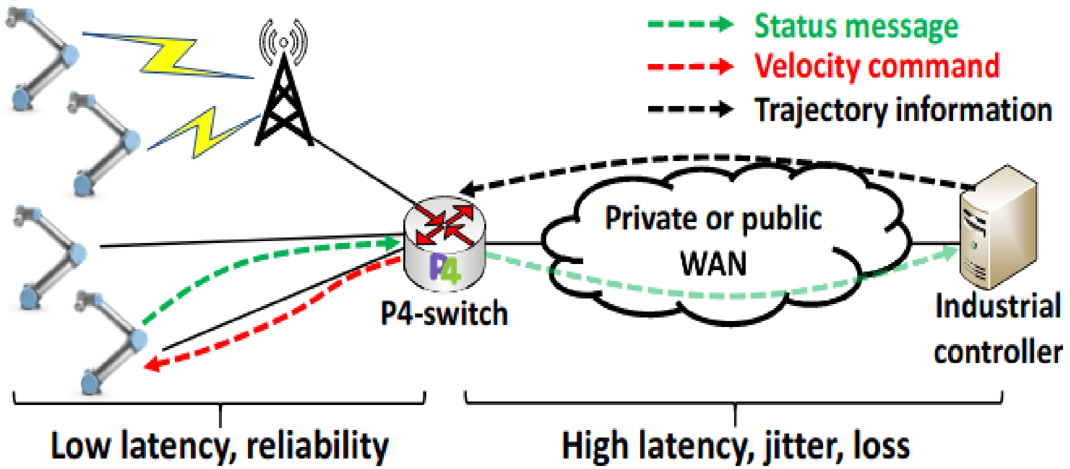
About P4



match-action tables and arithmetic operations
Programmable Parser
Memory
ALU
Programmable Deparser

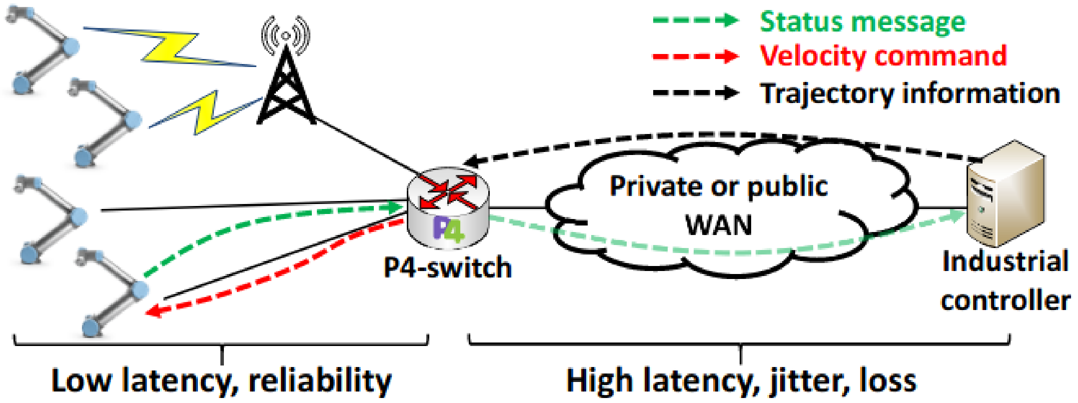
In-network velocity control

- **Assumptions**
  - Robot arm as a set of actuators
  - Each actuator reports its state (joint velocity and position) periodically
  - Each servo requires control commands (joint velocity) at a predefined frequency
  - UDP-based communication
- **Concept**
  - High-level control in remote cloud
    - Trajectory calculation
    - Non-latency sensitive
  - Moving real-time control to a P4 programmable device
    - PLC-like role
    - Velocity value calculation



Design – Main Components

- **Robot arms**
  - Actuators (joints) work independently, stream their internal state (position and velocity).
  - They require velocity control messages at a predefined rate.
  - In our system model, each robot arm is handled as a set of actuators controlled in sync
  - UDP communication, decimal values in binary representation
- **P4-switch**
  - Control Plane + Data Plane
  - Buffering trajectories
  - PID-like velocity control
- **Industrial controller**
  - Only high level planning



More on the protocol

- P4 capable devices are not suited for deep packet inspection.
  - Every important field used for robot control has to be close enough to the beginning of the packet.
- P4 language does not support floating-point arithmetic.
  - it is possible to implement this conversion in P4, it is much simpler and comfortable if the value is already in a decimal format in the used protocol.
- We use the same header structure for status and command messages encapsulated into simple IP/UDP packets.
  - **robot ID:** used as a unique identifier of the robot arm
  - **joint ID:** determines the joint of the given robot
  - **joint velocity:** the current speed (in rad/s) of the given joint in the status messages or the new joint-speed value to be set in the commands
  - **joint position:** the current position (in rad) of the given joint in the status messages, and unset in the commands

Trajectory representation (ingress)

- Trajectories are encoded in tables and registers.
- Each TP (trajectory point) is identified by a unique ID.
- We use durations instead of relative timestamps.
- We also store the next TP's ID, thus creating a linked list-like structure.
- The next TP can be modified, thus changing the original path.

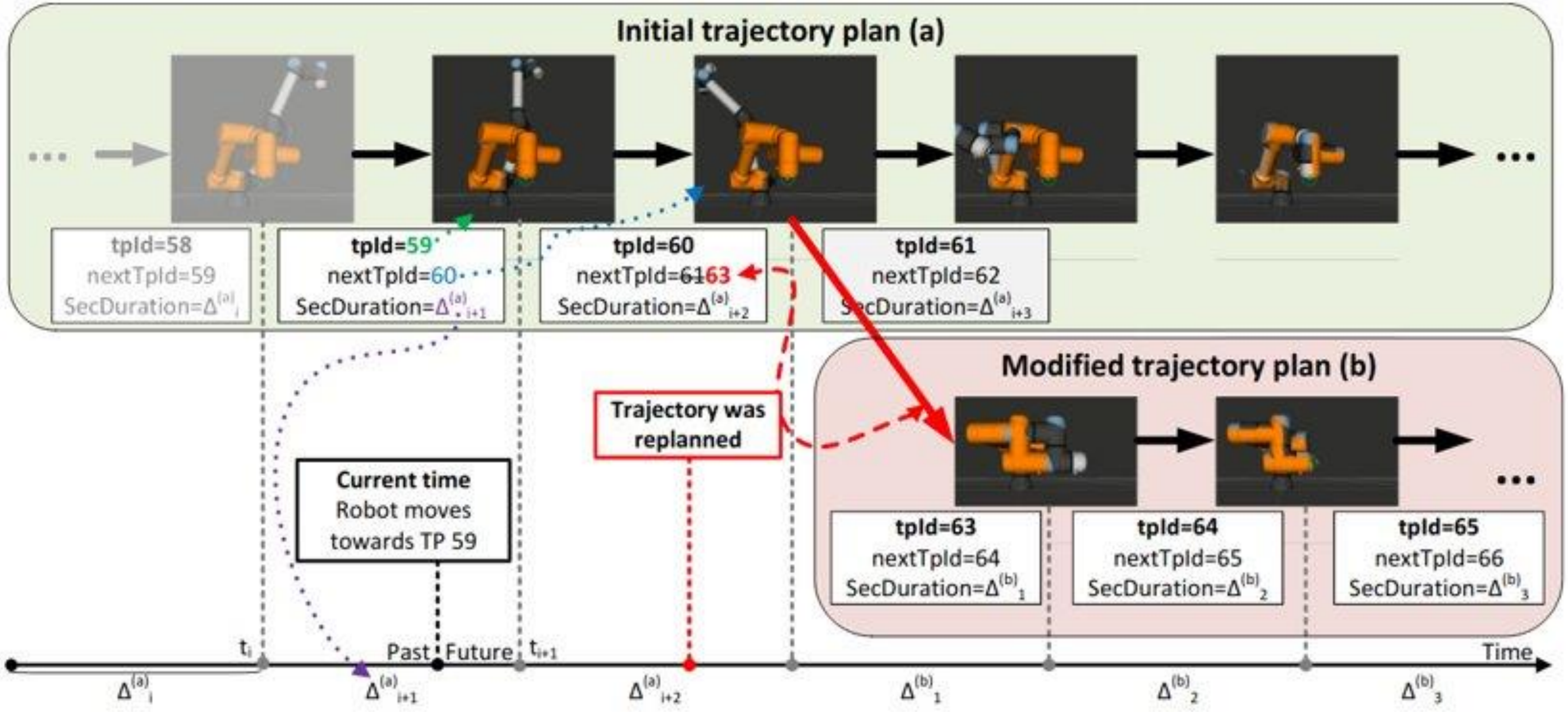
Tables

table TrajectorySwitcher		
rh.RId: exact	**m.tpId: exact**	**Action**
53	321	Set(m.nextTpId=52)
55	60 4	Set(m.nextTpId=63) 5
...		...
	Default	*NoAction*

table TPStepper		
rh.RId: exact	**m.tpId: exact**	**Action** 2
55	59 1	Set(m.nextTpId=60, m.secDuration=$\Delta^{(a)}_{i+1}$)
55	60 3	Set(m.nextTpId=61, m.secDuration=$\Delta^{(a)}_{i+2}$)
...		...
	Default	*NoAction*

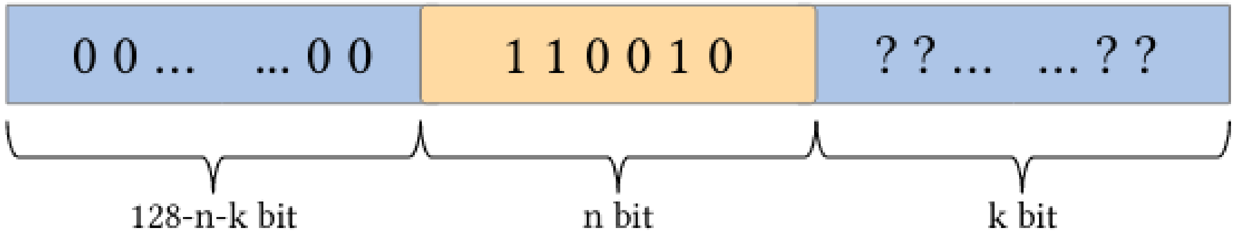
Benefits of the representation (ingress)

- Sitching between trajectories, repeating parts, stoping the movement, ...



PID-lik control using approximations (egress)

- Weighted sum of 3 values.
- Approximating function values usung the most significant non-zero n bits.
- Having a higher absolute error for higher values is acceptable.



Tables

table LimitVelocity			
rh.RId: exact	**rh.JointId: exact**	**rh.velocity: lpm**	**Action**
53	0	0b 1/1	rh.velocity = c
53	0	0b 01/2	rh.velocity = c
53	0	0b 001/3	rh.velocity = c
53	0	0b 0001/4	rh.velocity = c
53	0	0b 0000 1111 1/9	rh.velocity = c
53	0	0b 0000 1111 01/10	rh.velocity = c
53	0	0b 0000 1111 0111/12	rh.velocity = c
53	0	0b 0000 1111 0110 1/13	rh.velocity = c
53	0	0b 0000 1111 0110 011/15	rh.velocity = c
53	0	0b 0000 1111 0110 0101/16	rh.velocity = c
…			…
		Default	*NoAction*

rh.velocity = c **if rh.velocity > c**
(where **c = 0b 0000 1111 0110 0100**)

table TransformTrgVelocity	
m.trgVel: lpm	**Action** m.trgVel =
0b 1000 0..0/4	*f*(0b 1000 0..0)
0b 1001 0..0/4	*f*(0b 1001 0..0)
0b 1010 0..0/4	*f*(0b 1010 0..0)
0b 1011 0..0/4	*f*(0b 1011 0..0)
0b 1100 0..0/4	*f*(0b 1100 0..0)
0b 1101 0..0/4	*f*(0b 1101 0..0)
0b 1111 0..0/4	*f*(0b 1111 0..0)
0b 0100 00..0/5	*f*(0b 0100 00..0)
0b 0100 10..0/5	*f*(0b 0100 10..0)
0b 0101 00..0/5	*f*(0b 0101 00..0)
0b 0101 10..0/5	*f*(0b 0101 10..0)
…	…
Default	*NoAction*

table TargetData			
rh.RId: exact	**m.tpId: exact**	**rh.JointId: exact**	**Action**
55	59	2	Set(m.trgPos=120, m.trgVel=1123)
55	60	2	Set(m.trgPos=180, m.trgVel=123)
…			…
		Default	*NoAction*

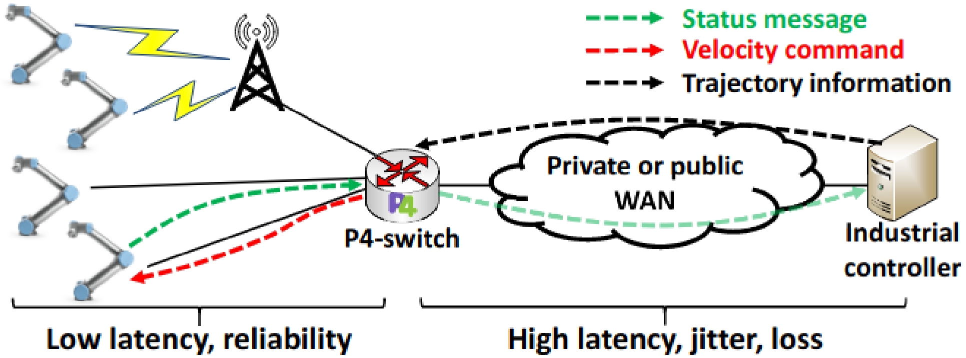
table TransformCurrVelocity	
rh.velocity: lpm	**Action** rh.velocity =
0b 1000 0..0/4	*g*(0b 1000 0..0)
0b 1001 0..0/4	*g*(0b 1001 0..0)
…	…
Default	*NoAction*

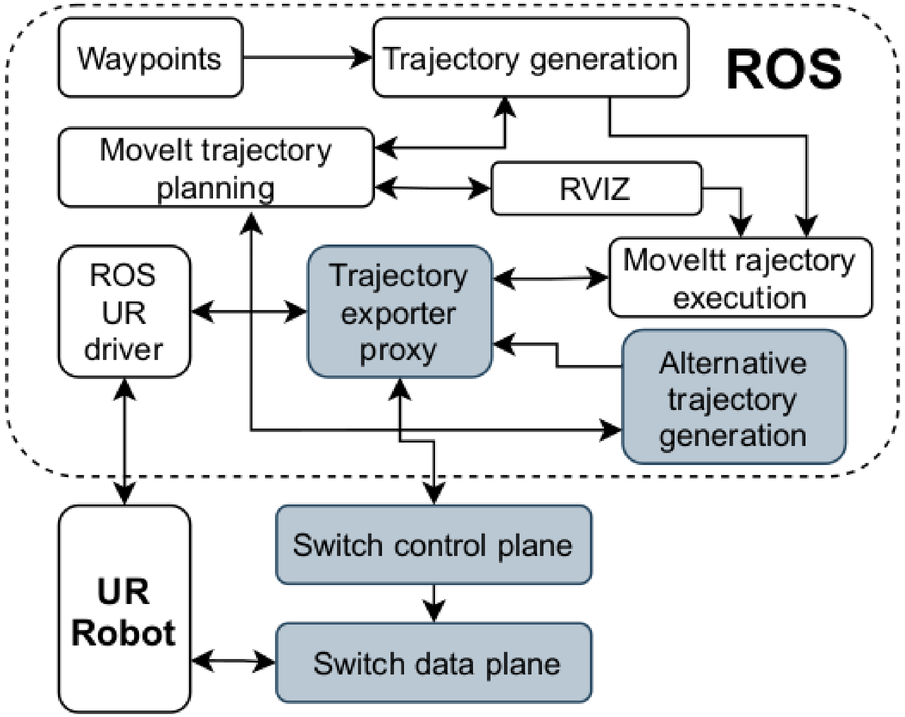
table TransformDiffPosition	
m.diffPos: lpm	**Action** m.diffPos +=
0b 1000 0..0/4	*h*(0b 1000 0..0)
0b 1001 0..0/4	*h*(0b 1001 0..0)
…	…
Default	*NoAction*

rh.velocity = f(x) + g(y) + h(z)

Proof of concept implementation

- **Robot:**
  - UR industrial robot arm
- **P4-switch**
  - Barefoot/Intel Tofino-based switch
  - Barefoot Runtime (Python/C++)
- **Industrial controller** (traj. planning)
  - ROS

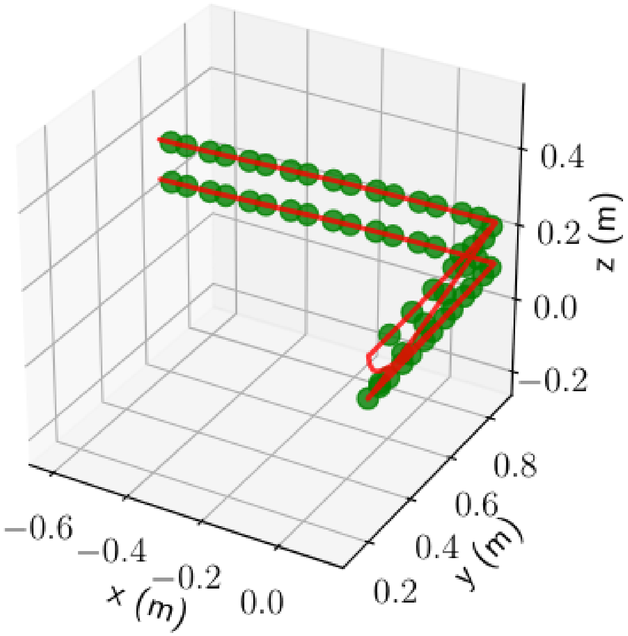




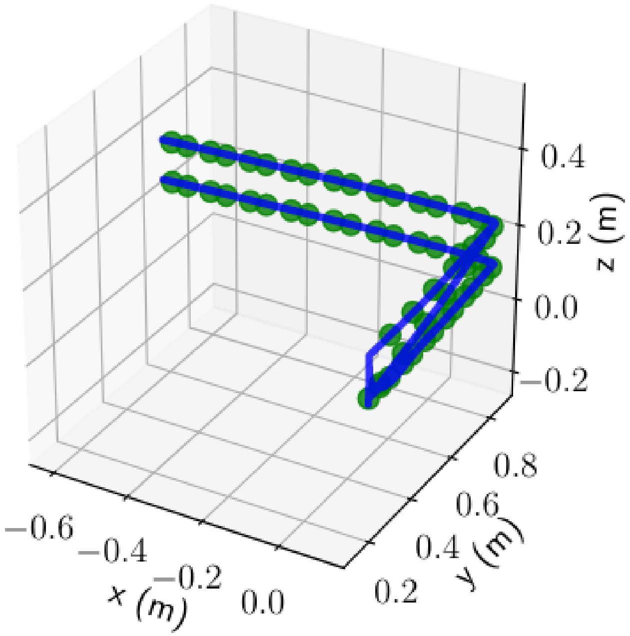
Evaluation

- Precision
- Effect of background activities
- Scalability

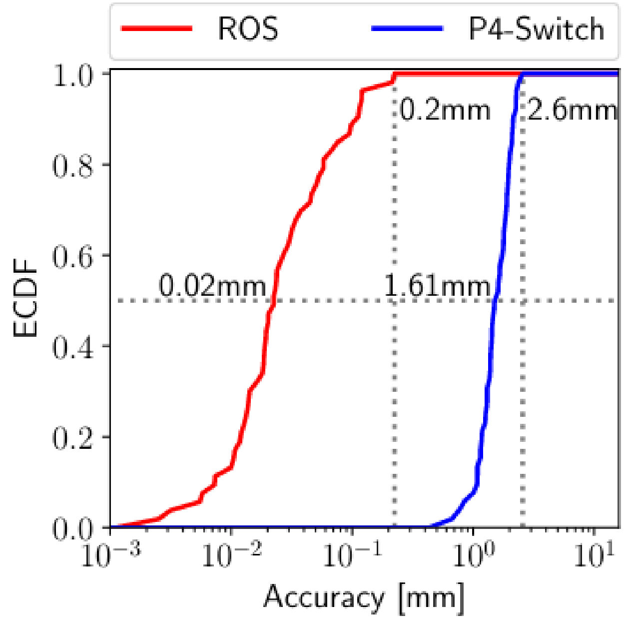
Precision



(a) ROS (PID-control)

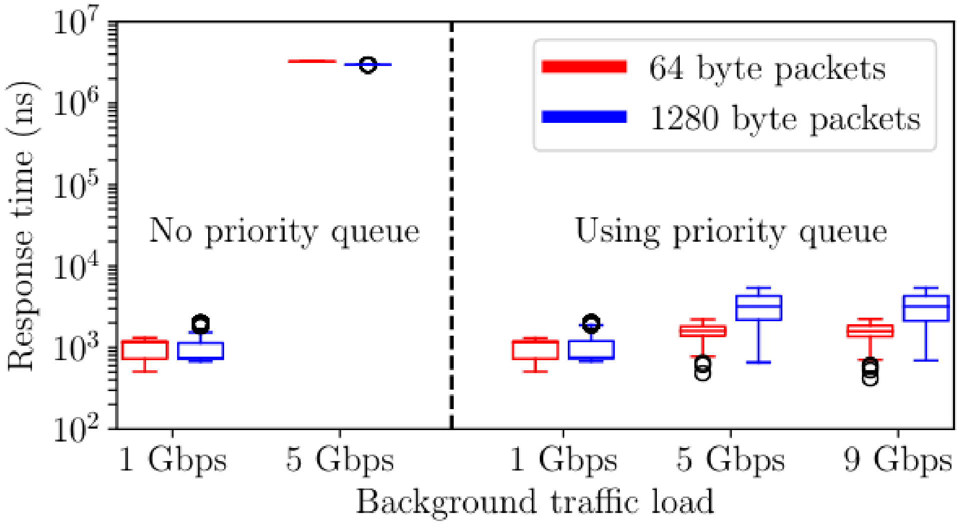


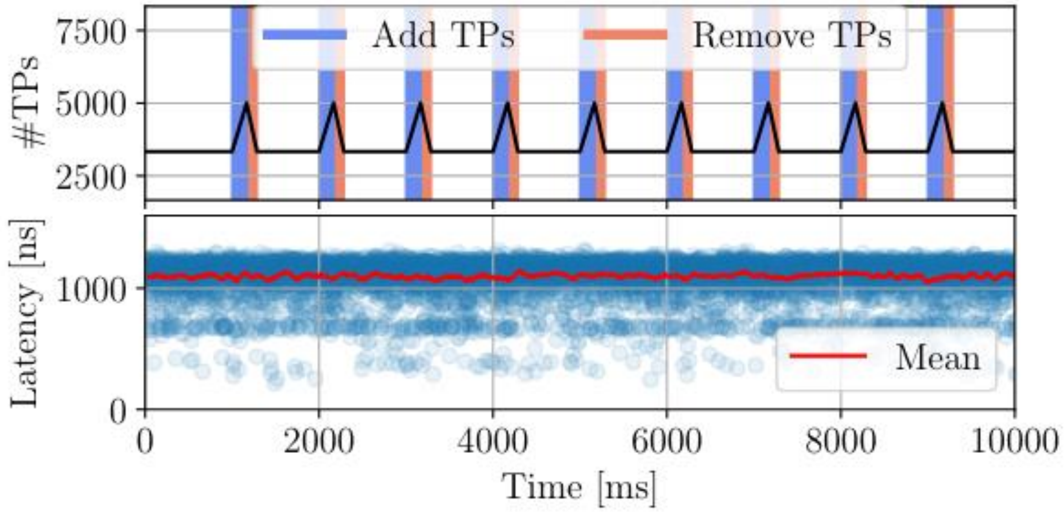
(b) P4-Switch (P-control)



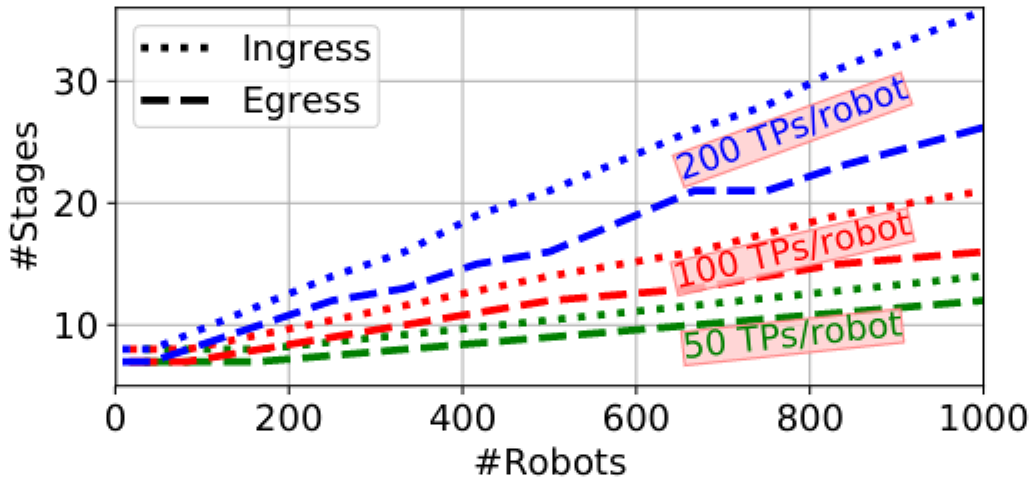
(c) Tool position accuracy in the TPs

Effect of background activities

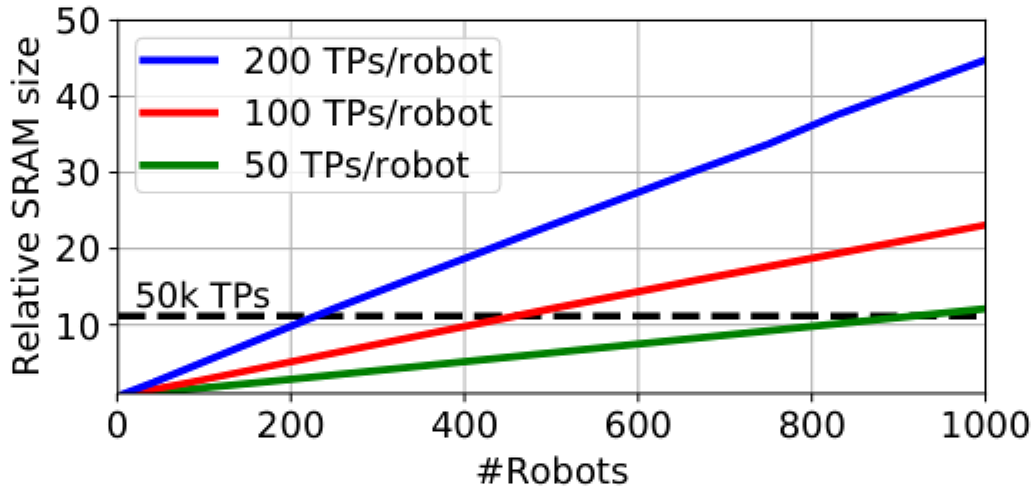




Scalability



(a) Stage occupancy.



(b) SRAM usage.

Summary

- P4-based in-network robot control
  - buffering following trajectories
  - Trajectory switching, repeating, stoppong
  - synchronisation
  - PID-like control
- Proof-of-concept implementation
  - UR robot arm
  - ROS
  - Barefoot/Intel Tofino ASIC

Thank you for your attention!

- Sándor Laki - lakis@inf.elte.hu
- Csaba Györgyi - gycsaba96@inf.elte.hu
- József Pető - peto@tmit.bme.hu
- Péter Vörös - vopraai@inf.elte.hu
- Géza Szabó - geza.szabo@ericsson.com