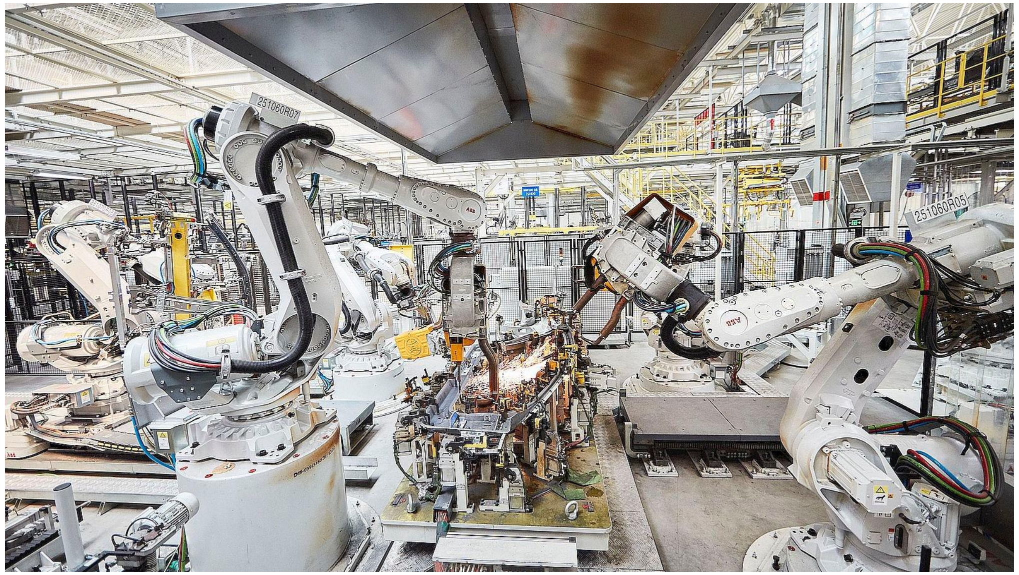
SentryRT-1: A Case Study in Evaluating Real-Time Linux for Safety-Critical Robotic Perception

Yuwen Shen[∗†‡], Jorrit Vander Mynsbrugge[∗‡], Nima Roshandel[∗†‡], Robin Bouchez[∗‡], Hamed FirouziPouyaei[∗‡], Constantin Scholz[∗‡], Hoang-Long Cao[∗§], Bram Vanderborght[∗‡], Wouter Joosen[†], Antonio Paolillo[∗]

[∗]Vrije Universiteit Brussel, Belgium, [†]KU Leuven, Belgium

[‡]imec, Belgium, [§]Can Tho University, Vietnam

Industry robot → cobot



[1]

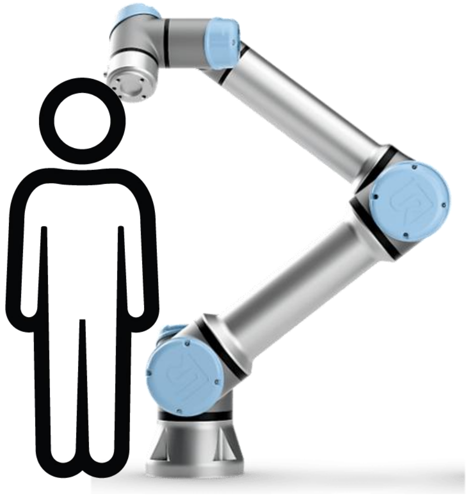
→

[2]

Performance

Safety

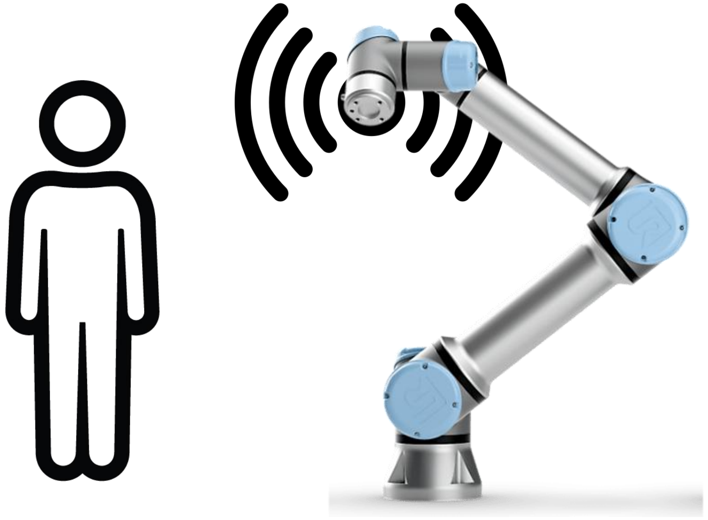
Cobot → safe perception



[3]

Safety by contact

→



Safety by perception

Motivation

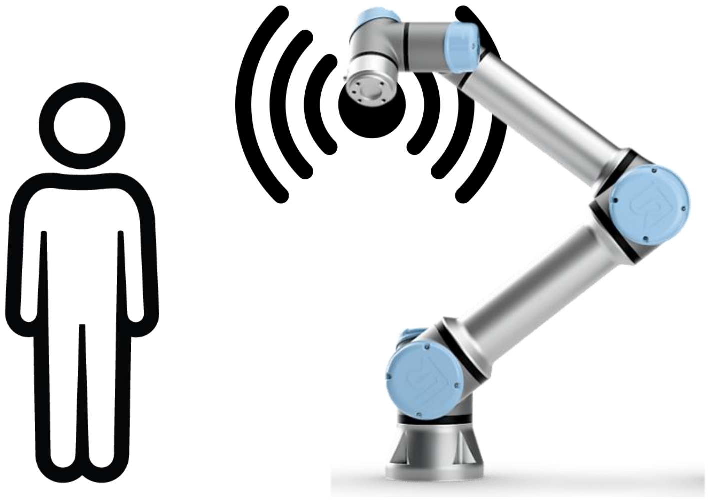
Robot: Safety & Performance

Real-time constraints

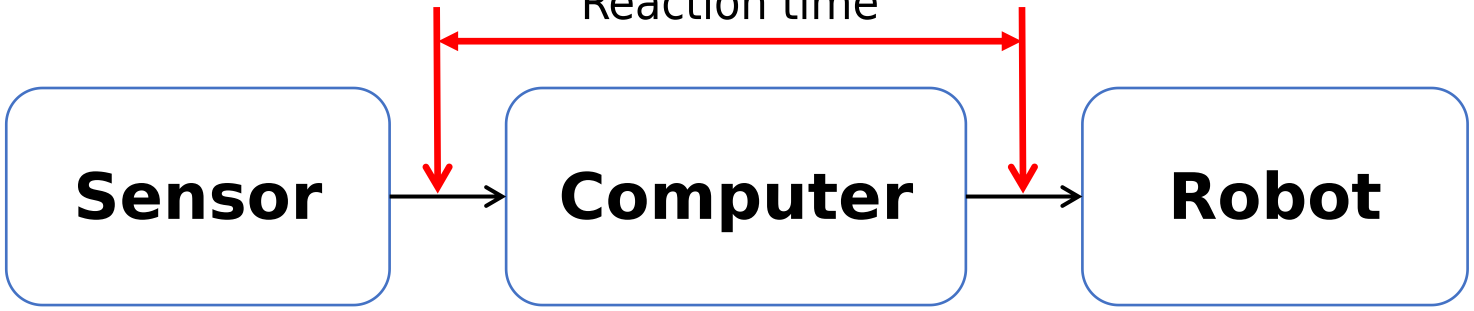
Reaction time

Linux configurations

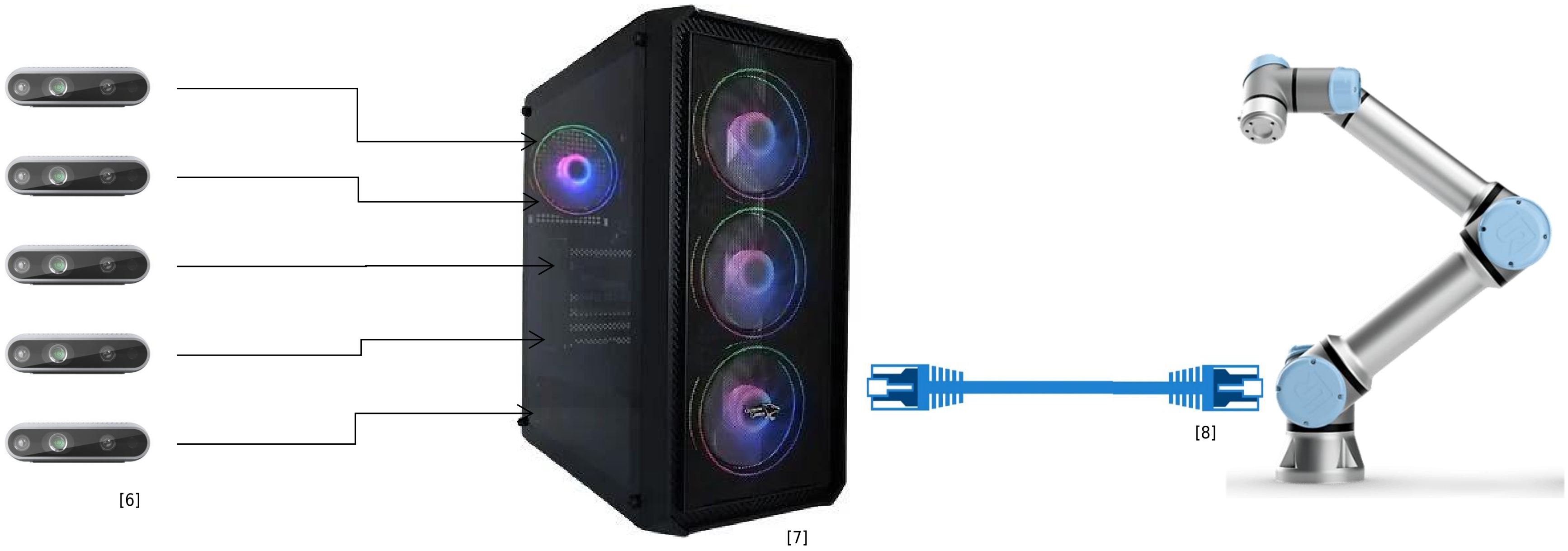
- Scheduling policies: `SCHED_DEADLINE`
- Kernels: `PREEMPT_RT`



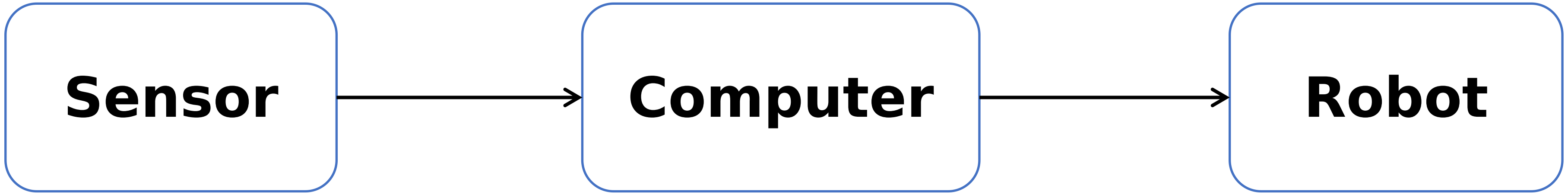
SentryRT-1



Hardware view



Intel i9-14900KF (32 CPUs)
NVIDIA GeForce RTX 4060 Ti


Sensor

Computer

Robot

Setup: physical

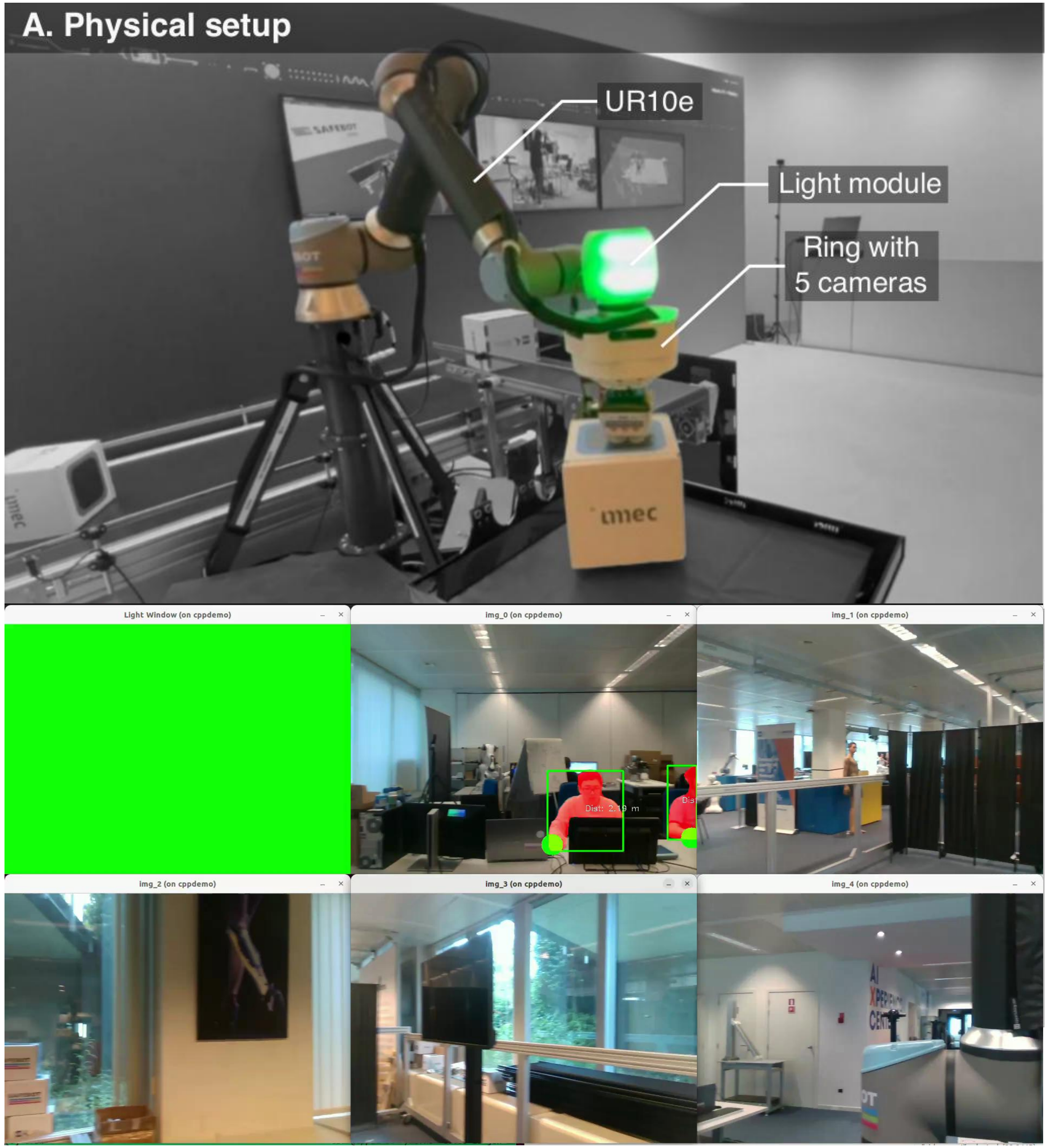
UR10e Robot, sensing ring, light module

5 cameras as input
Physical

Human detection
GPU accelerated NN

Speed and Separation Monitoring **(SSM)**
Human close to robot
→ robot speed adjustment

Visualization
images of each camera



Setup: virtual

UR10e Robot / **URSim**

5 cameras as input

Physical / **Virtual**

Human detection

GPU accelerated NN

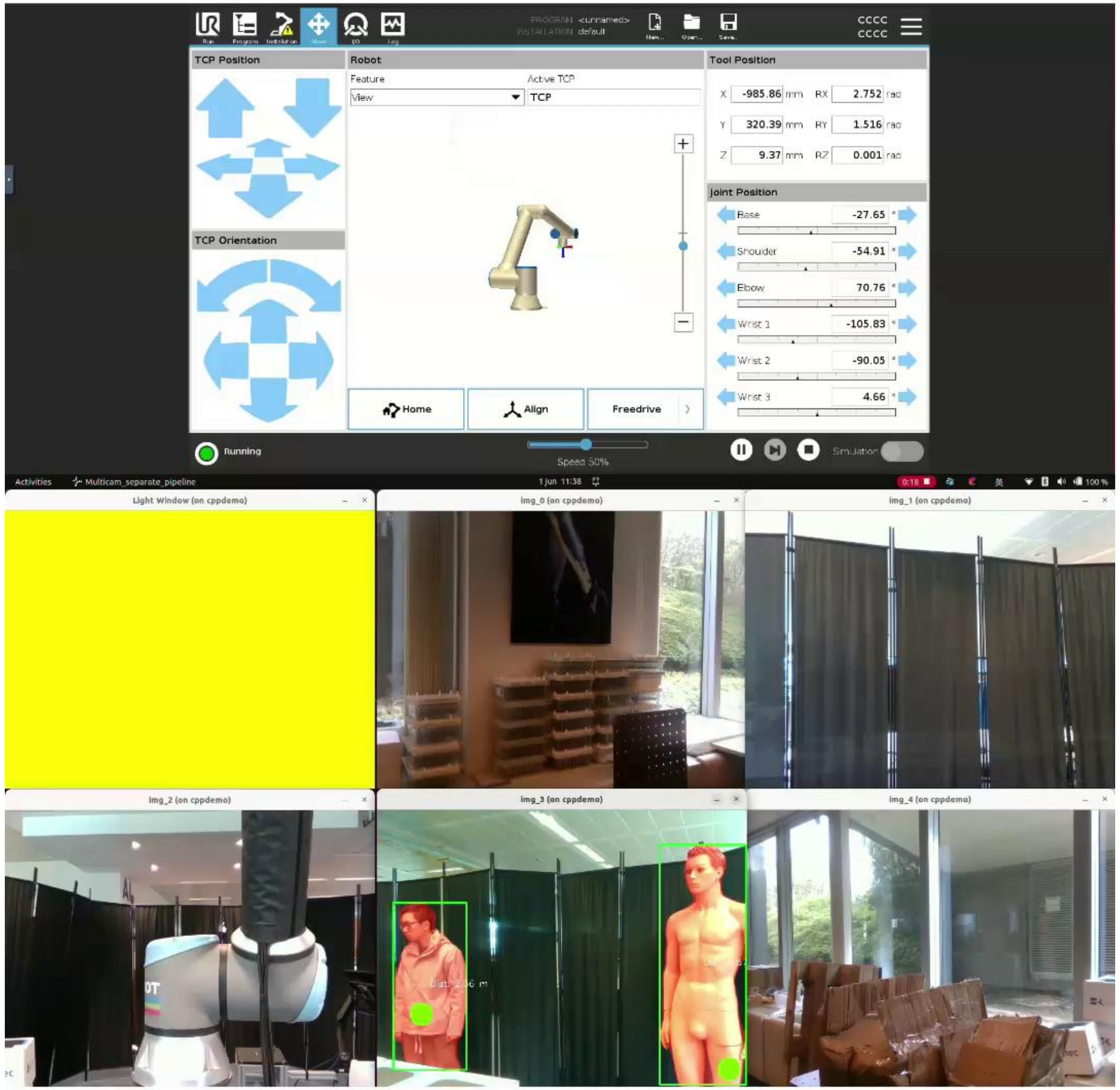
Speed and Separation Monitoring **(SSM)**

Human close to robot

→ robot speed adjustment

Visualization

images of each camera



X -985.86 mm RX 2.752 rad

Y 320.39 mm RY 1.516 rad

Z 9.37 mm RZ 0.001 rad

Base -27.65 °

Shoulder -54.91 °

Elbow 70.76 °

Wrist 1 -105.83 °

Wrist 2 -90.05 °

Wrist 3 4.66 °

Feature: View

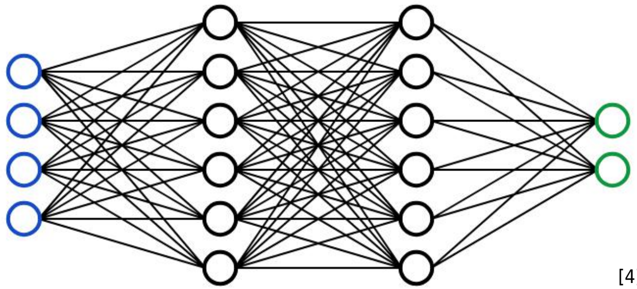
Active TCP: TCP

Speed 50%

Concern about neural network?

Preliminary version

→ does have miss detection

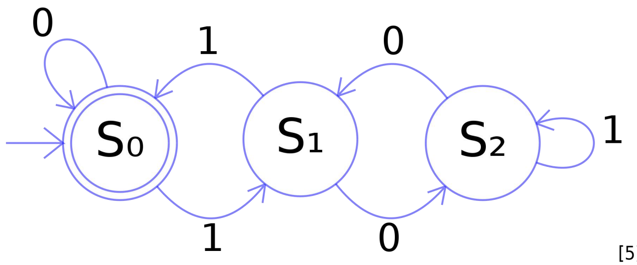


[4]

WIP: Deterministic version

→ watchdog of NN

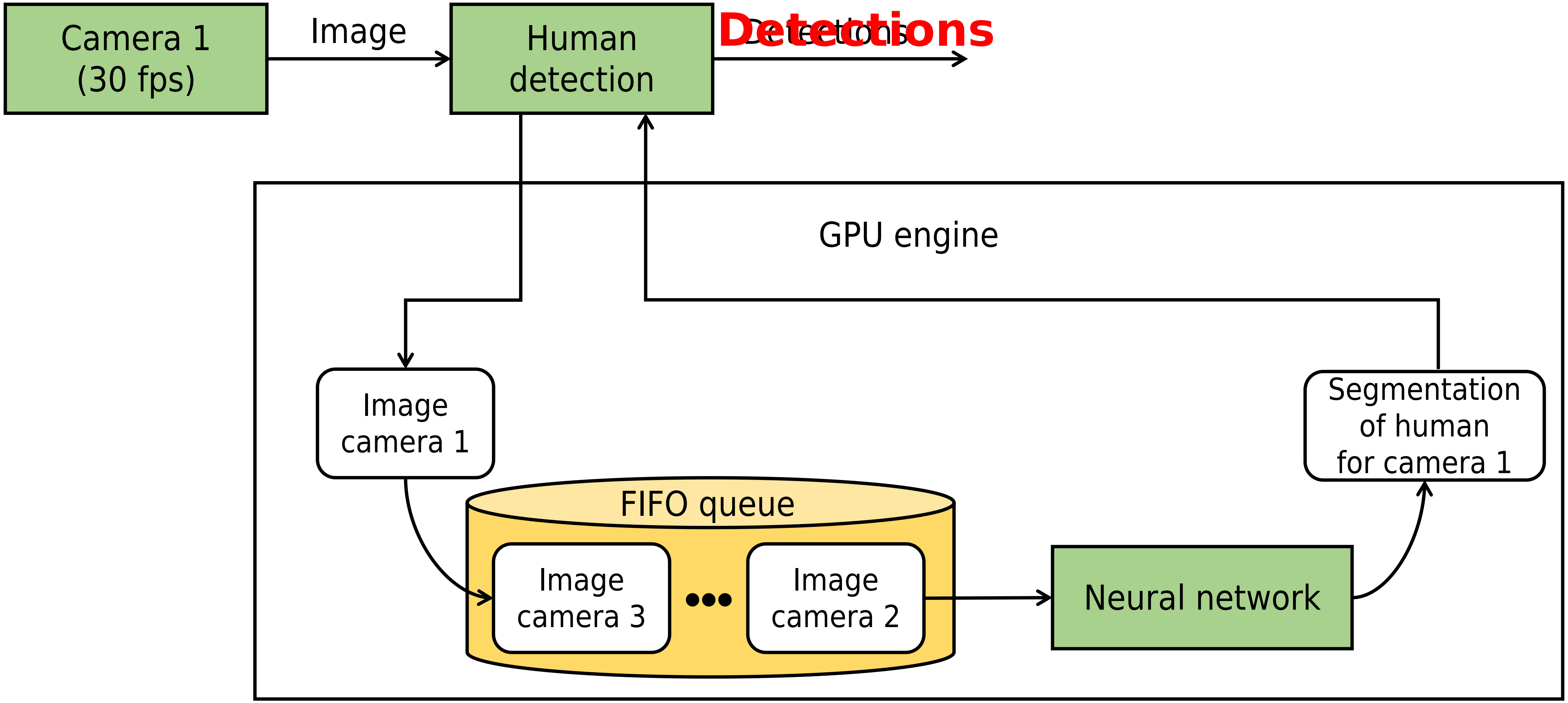
→ working on a certifiable software



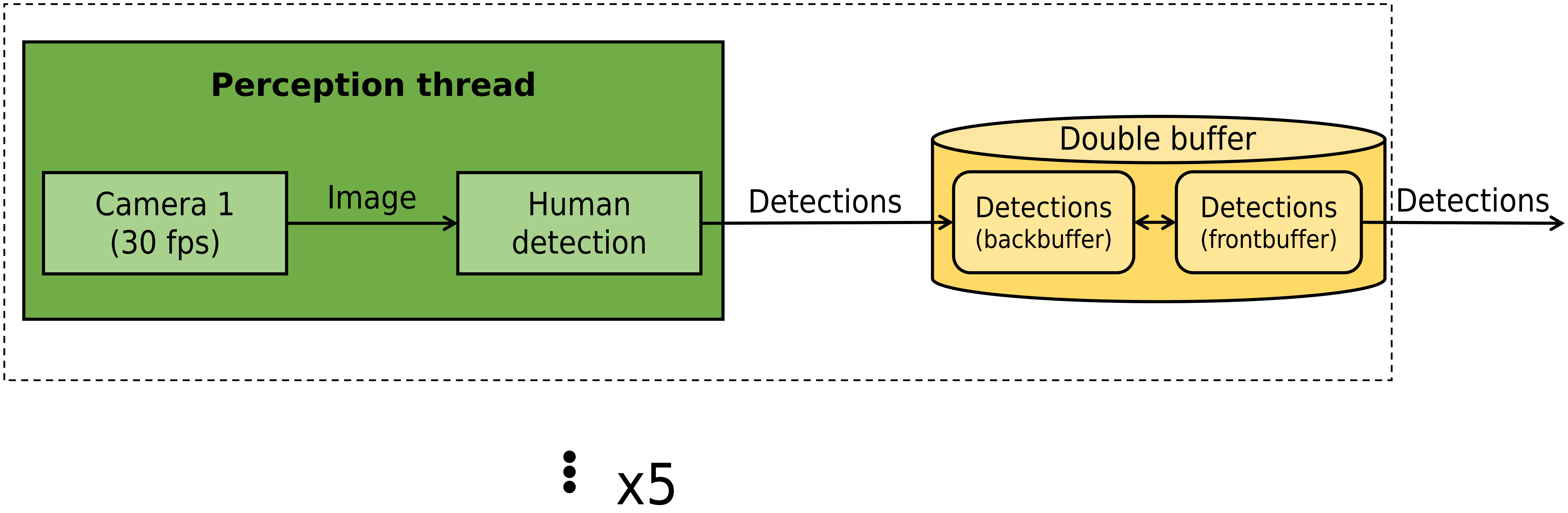
[5]

SentryRT-1: focus on the timing

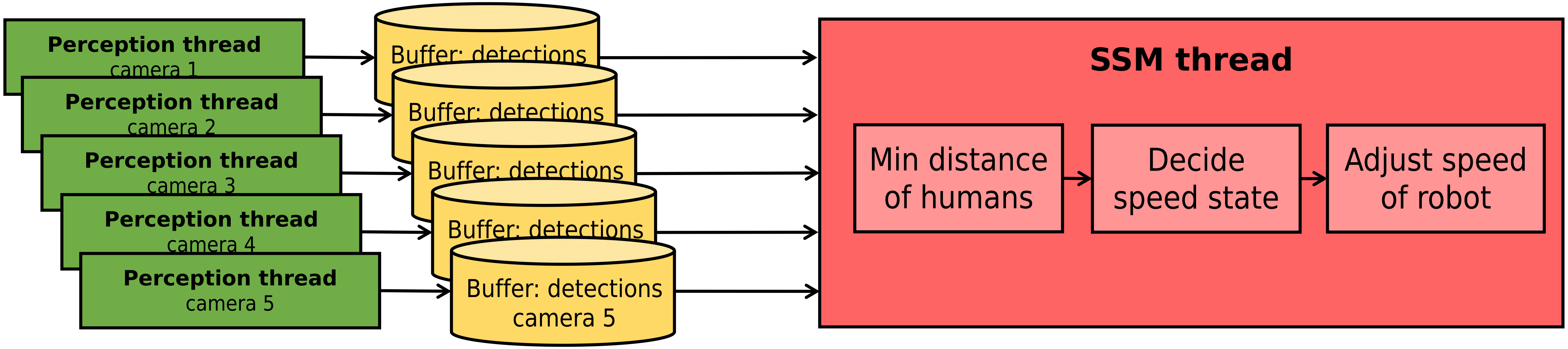
Software framework: perception



Software framework: buffer

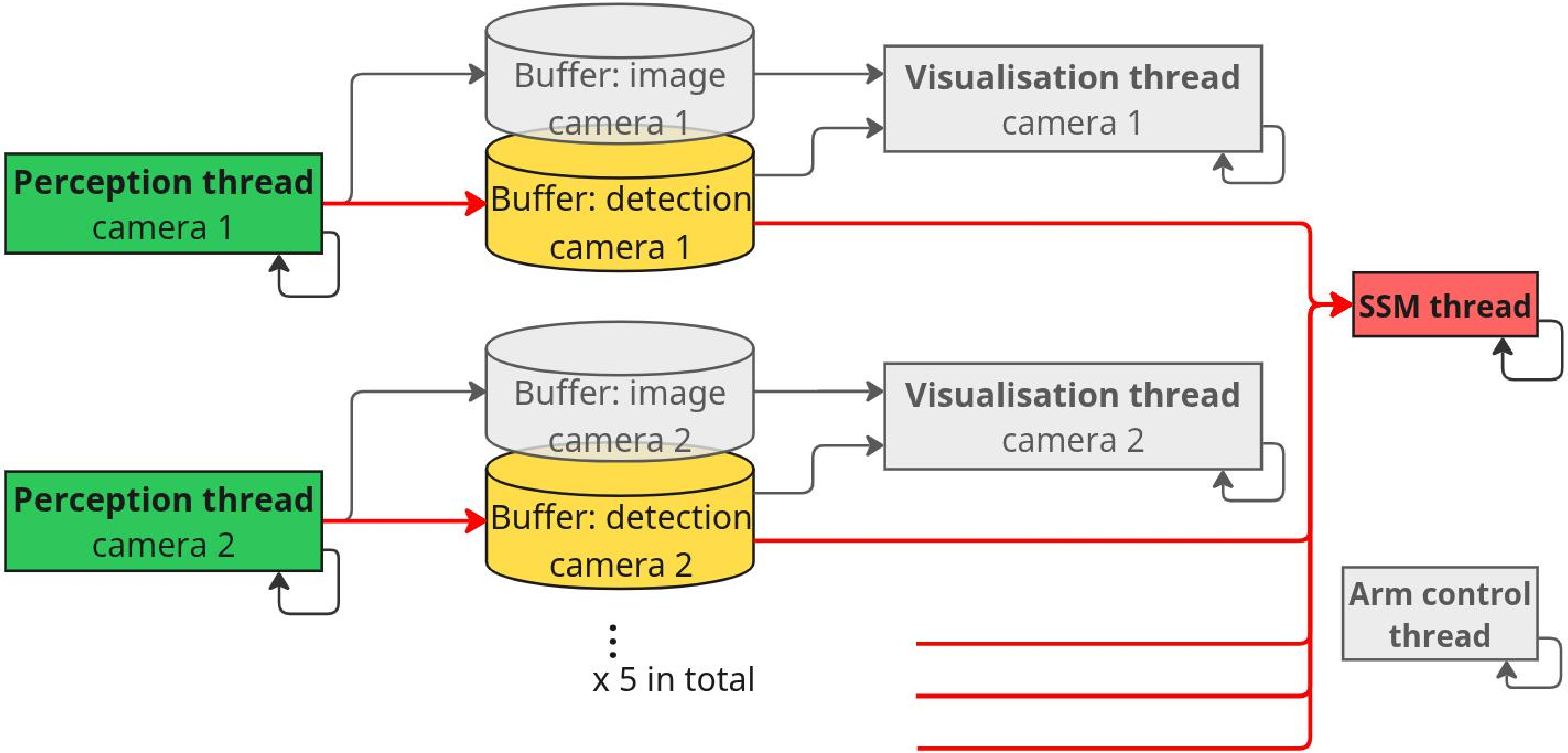


Software framework: SSM



Critical path

Software framework



Task model

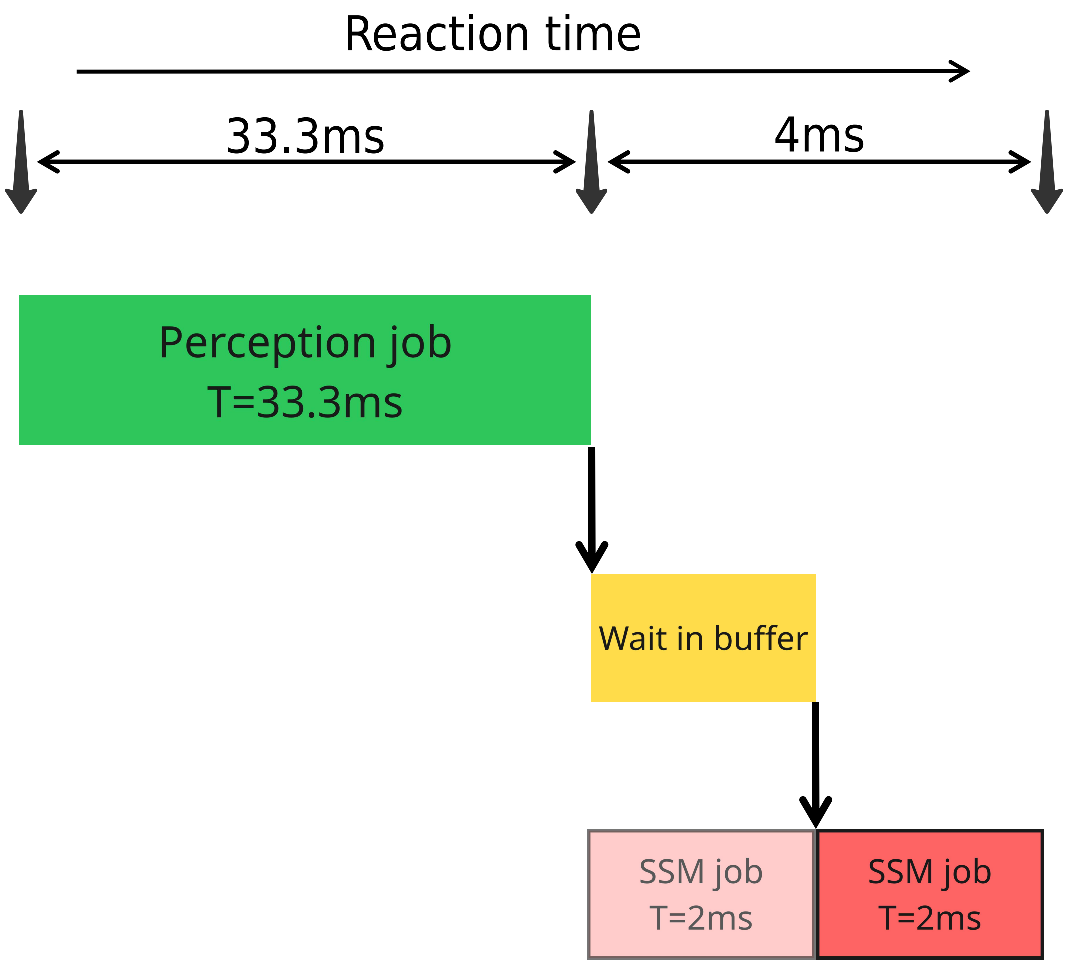
Tasks in critical path:

Perception task (cam i)

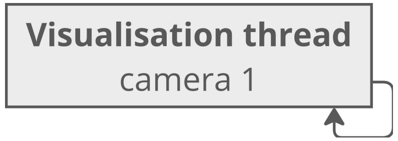
τ_{p_i}

SSM task

τ_{ssm}

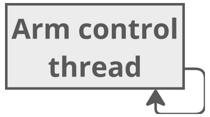


Other tasks:



Visualisation task (cam i)

τ_{v_i}



Arm control task

τ_{arm}

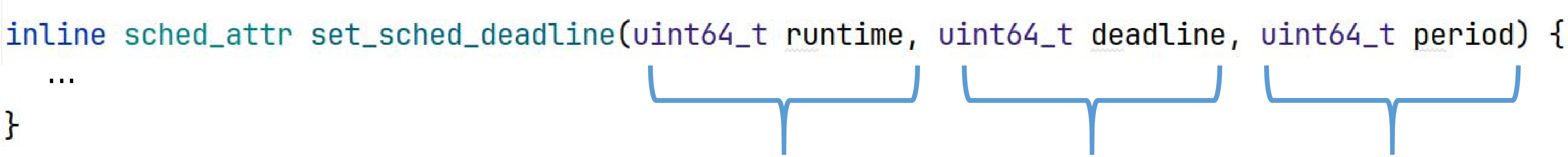
SCHED_DEADLINE

Perception task (cam i)

Perception thread
camera i

SSM task

SSM thread



```
inline sched_attr set_sched_deadline(uint64_t runtime, uint64_t deadline, uint64_t period) {
   ...
}
```

Parameters!

Subthread scheduling

- **Perception task (cam i)** τ_{p_i}
  - Intel RealSense
  - TensorRT
  - OpenGL
- **SSM task** τ_{ssm}
  - UR RTDE

Problem:

Assigning a real-time policy to a main thread does **not propagate** to its subthreads.

Workaround:

→ Automatically apply SCHED_RR with fixed priority to subthreads.

Subthread scheduling

- **Perception task (cam i)** τ_{p_i}
  - Intel RealSense
  - TensorRT
  - OpenGL
- **SSM task** τ_{ssm}
  - UR RTDE

Abbr.	**Main threads**	**Subthreads**
CFS+CFS	SCHED_OTHER	SCHED_OTHER
RR+RR	SCHED_RR	SCHED_RR
DL+CFS	SCHED_DEADLINE	SCHED_OTHER
DL+RR	SCHED_DEADLINE	SCHED_RR

Configurations

Scheduling policy **main+subthreads**
CFS+CFS
RR+RR
DL+CFS
DL+RR

X

Linux kernel
generic
lowlatency
realtime

X

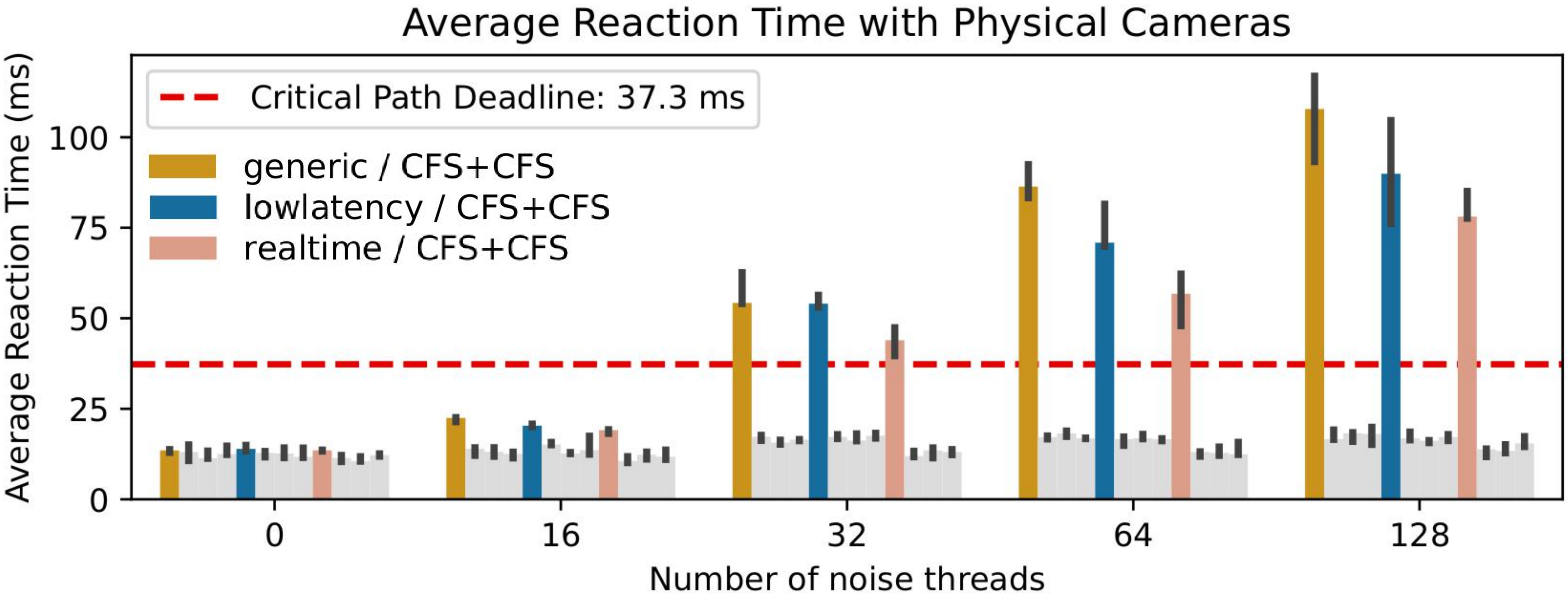
Noise thread count
0
16
32
64
128

X

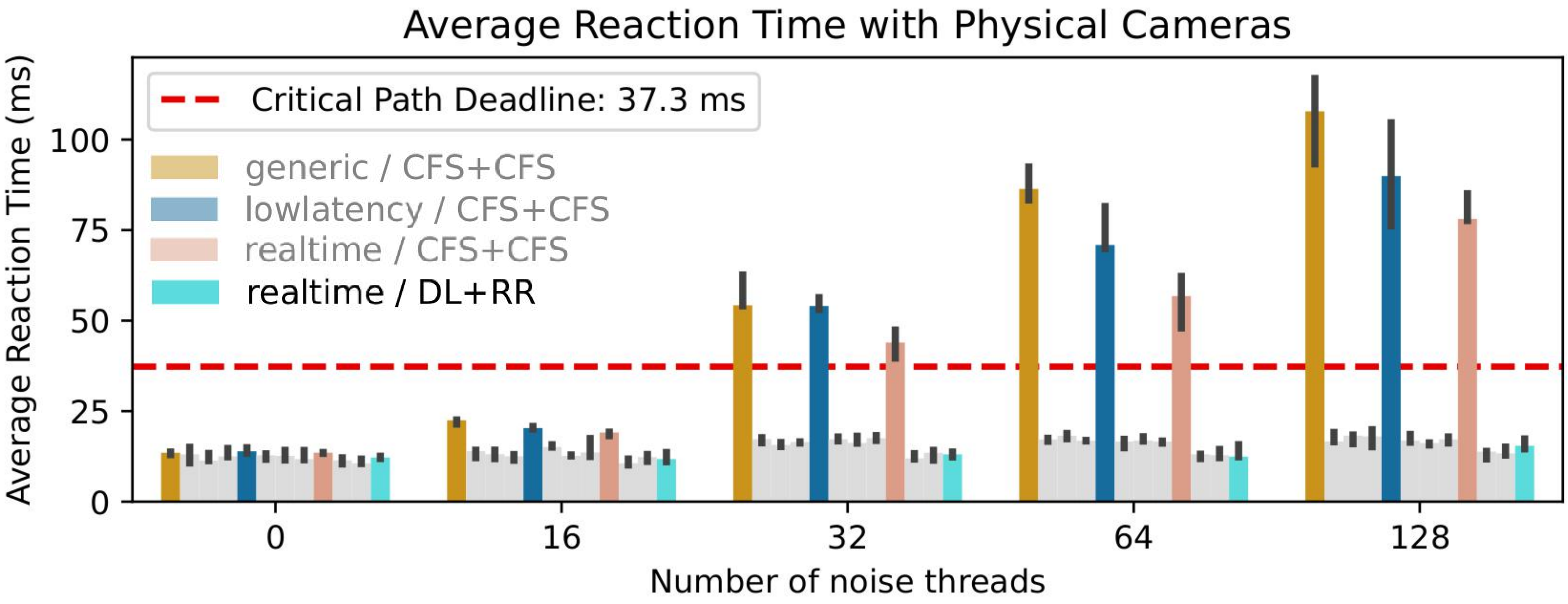
Camera
physical (D435i)
virtual (dataset)

X 3 repeats, 30 sec for each run

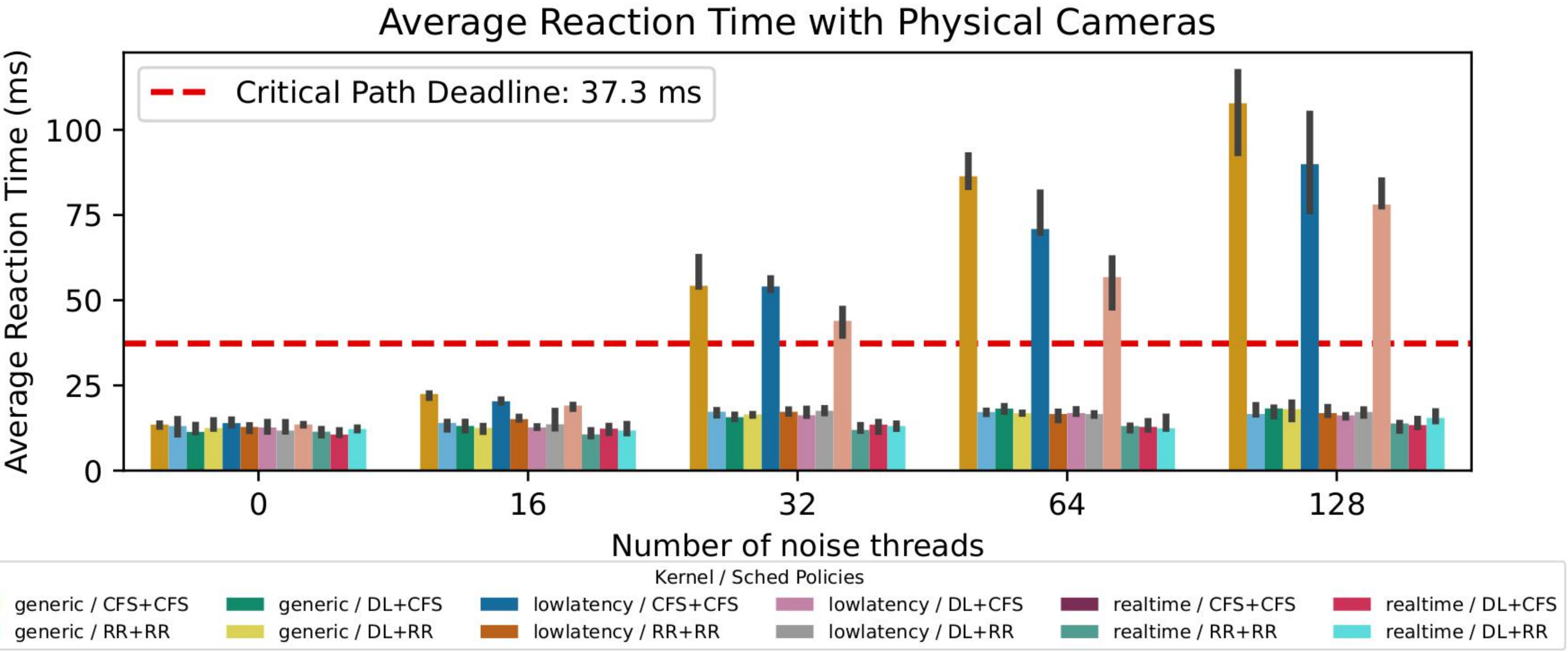
Default CFS setting degrades



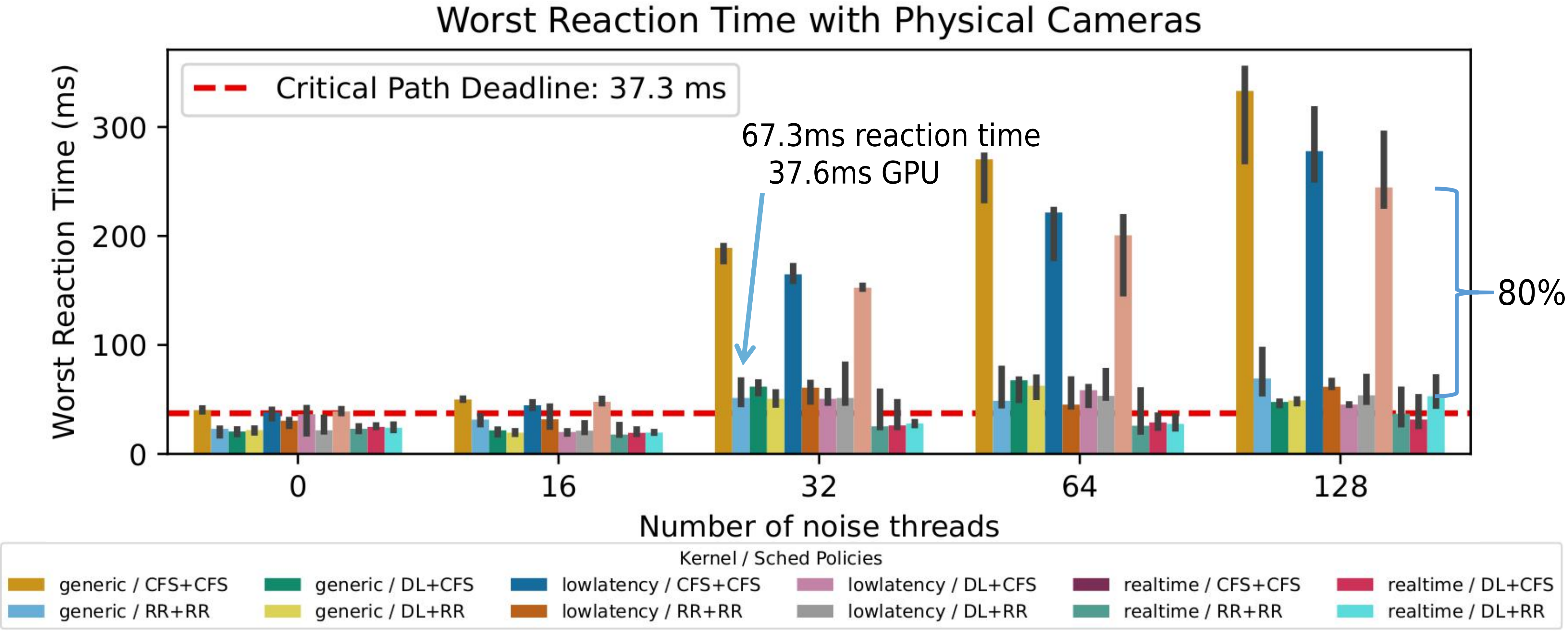
Real-time scheduling policies keep stable



Other combinations are similar



Worst-case reaction time is more variable



67.3ms reaction time
37.6ms GPU

80%

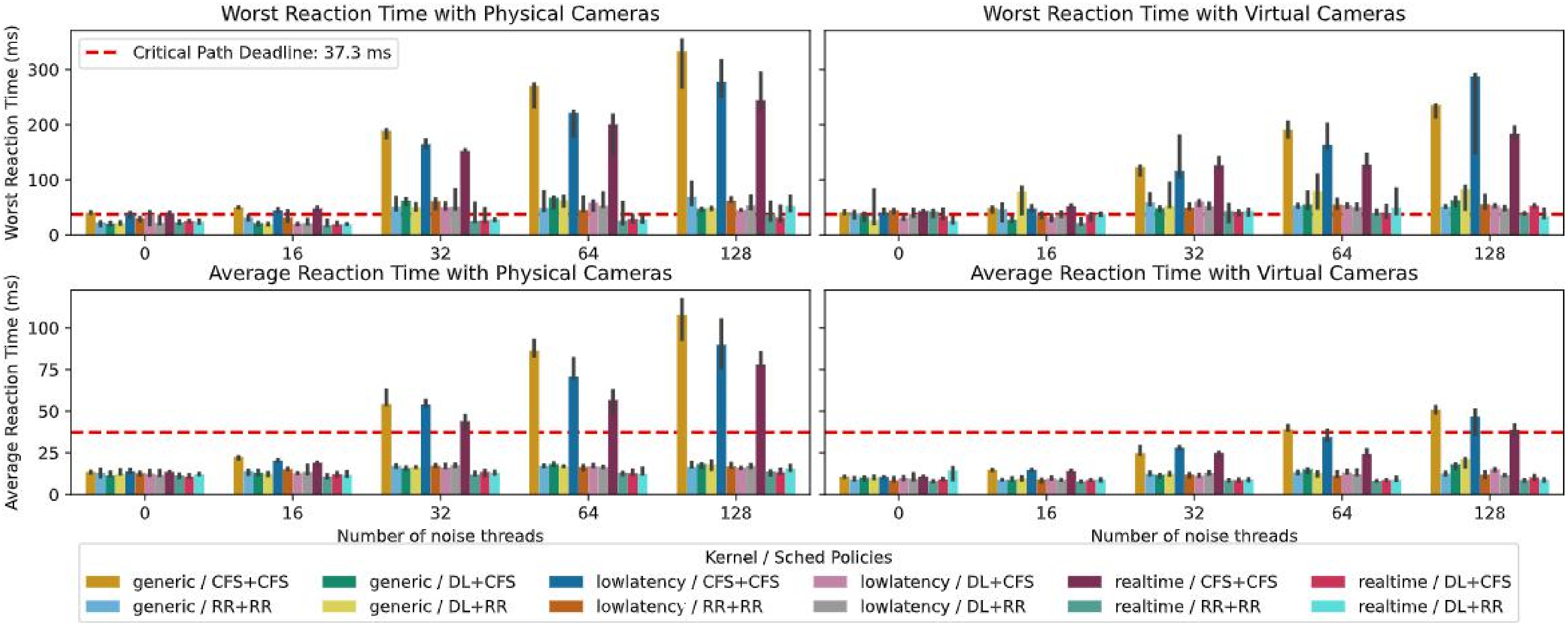
Virtual camera results are similar

Physical Camera ↓

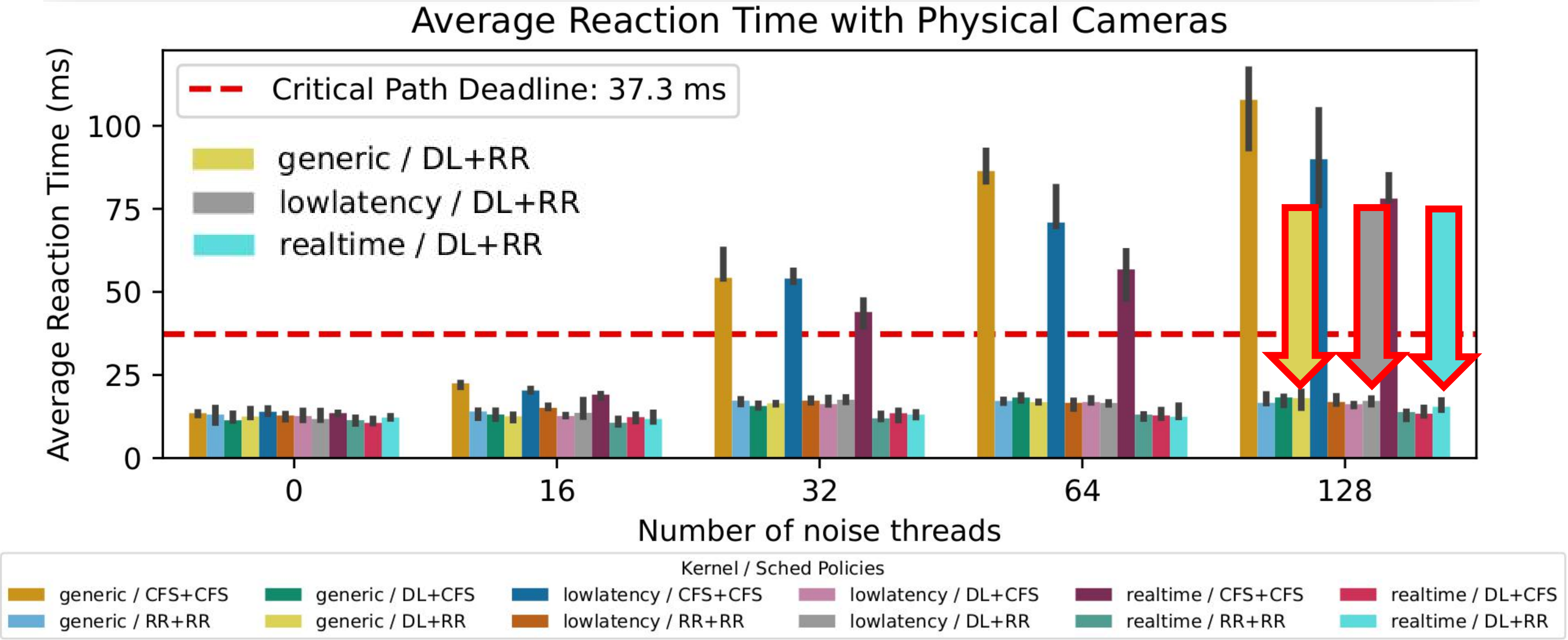
Virtual Camera ↓

Worst Case →

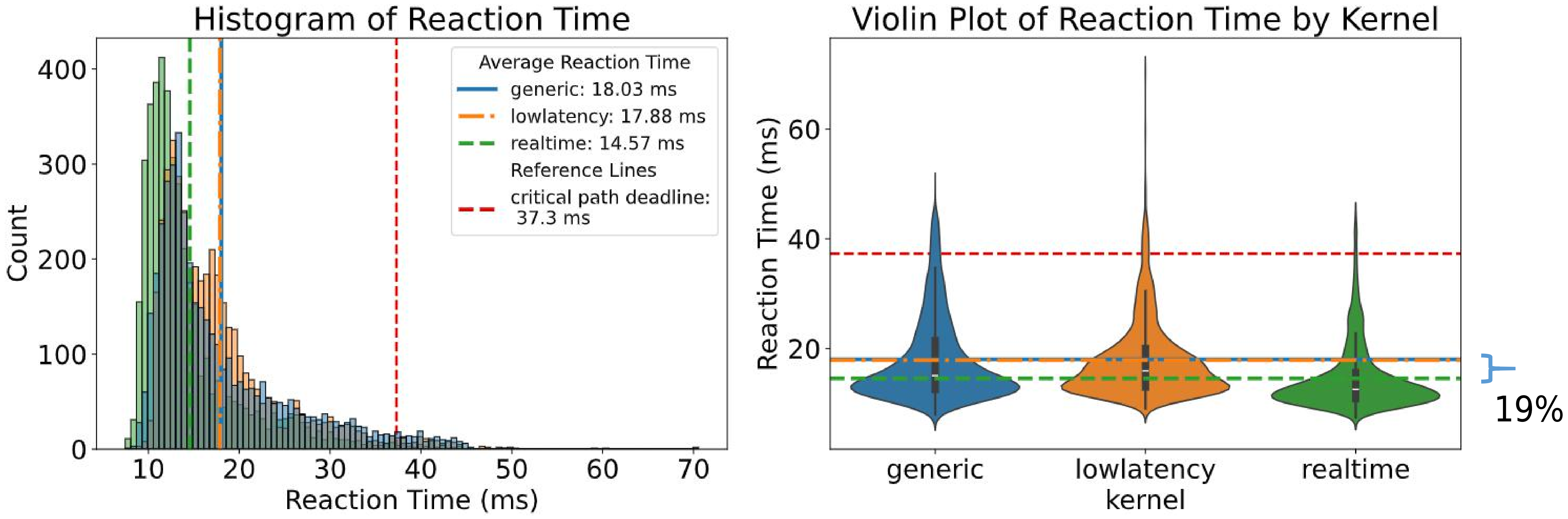
Average →



Look at the influence of kernels...



Real-time kernel with PREEMPT_RT can improve average reaction time



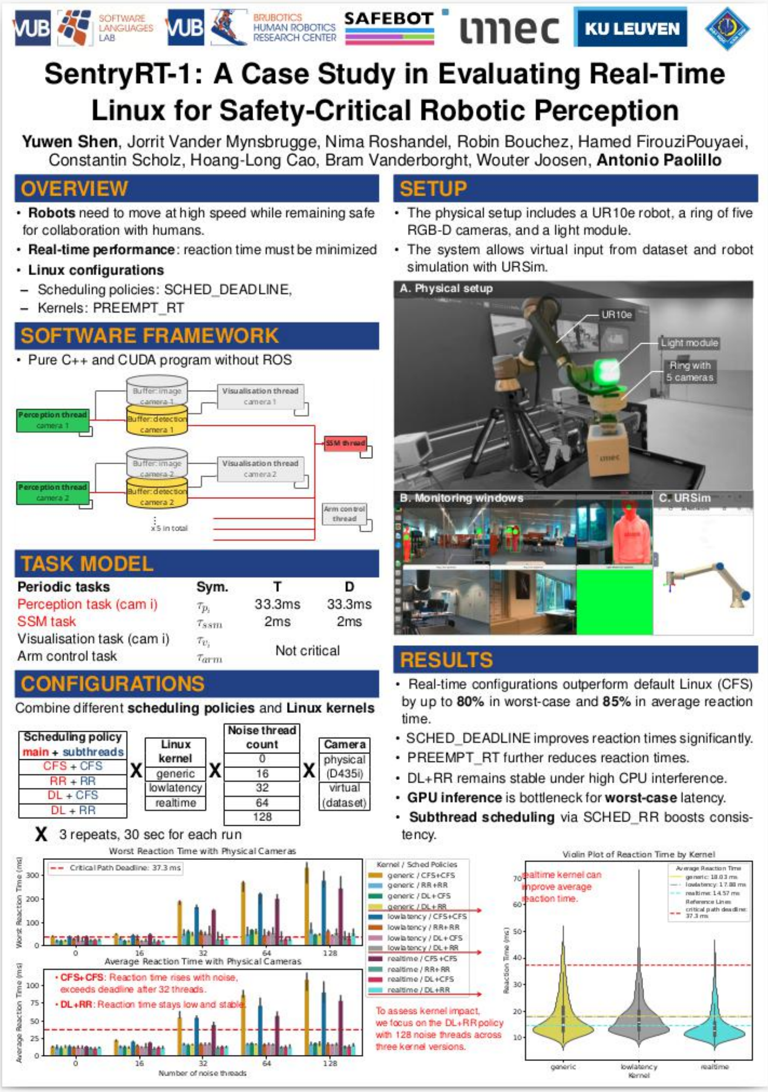
Physical cameras, DL+RR policy, 128 noise threads.

Conclusion

- SCHED_DEADLINE reduce up to 80% average reaction time +++
- Realtime kernel reduce up to 19% average reaction time +
- Subthreads use SCHED_RR o
- Deadline misses exist, due to unpredictable GPU usage, e.g. 38ms latency -

Questions for future work

- Is it possible to bound latencies in GPU usage?
- Propagate scheduling policies to subthreads?
- Extract timing parameters automatically? **[LiME, RTAS’25], [Timerlat, TC’25]**
- Newer kernel features such as EEVDF scheduling might help?
- Embedded platforms? e.g. NVIDIA Jetson Orin


VUB
SOFTWARE LANGUAGES LAB
VUB
BRUBOTICS HUMAN ROBOTICS RESEARCH CENTER
SAFEBOT
imec
KU LEUVEN

SentryRT-1: A Case Study in Evaluating Real-Time Linux for Safety-Critical Robotic Perception

Yuwen Shen, Jorrit Vander Mynsbrugge, Nima Roshandel, Robin Bouchez, Hamed FirouziPouyaei, Constantin Scholz, Hoang-Long Cao, Bram Vanderborght, Wouter Joosen, **Antonio Paolillo**

OVERVIEW

- **Robots** need to move at high speed while remaining safe for collaboration with humans.
- **Real-time performance**: reaction time must be minimized
- **Linux configurations**
  – Scheduling policies: SCHED_DEADLINE,
  – Kernels: PREEMPT_RT

SOFTWARE FRAMEWORK

- Pure C++ and CUDA program without ROS

Perception thread camera 1
Buffer: image camera 1
Buffer: detection camera 1
Visualisation thread camera 1
Perception thread camera 2
Buffer: image camera 2
Buffer: detection camera 2
Visualisation thread camera 2
x5 in total
SSM thread
Arm control thread

TASK MODEL

Periodic tasks	Sym.	T	D
Perception task (cam i)	τ_{p_i}	33.3ms	33.3ms
SSM task	τ_{ssm}	2ms	2ms
Visualisation task (cam i)	τ_{v_i}	Not critical	
Arm control task	τ_{arm}		

CONFIGURATIONS

Combine different **scheduling policies** and **Linux kernels**

Scheduling policy main + subthreads
CFS + CFS
RR + RR
DL + CFS
DL + RR

X

Linux kernel
generic
lowlatency
realtime

X

Noise thread count
0
16
32
64
128

X

Camera
physical (D435i)
virtual (dataset)

X 3 repeats, 30 sec for each run

- **CFS+CFS**: Reaction time rises with noise, exceeds deadline after 32 threads.
- **DL+RR**: Reaction time stays low and stable.

To assess kernel impact, we focus on the DL+RR policy with 128 noise threads across three kernel versions.

SETUP

- The physical setup includes a UR10e robot, a ring of five RGB-D cameras, and a light module.
- The system allows virtual input from dataset and robot simulation with URSim.

RESULTS

- Real-time configurations outperform default Linux (CFS) by up to **80%** in worst-case and **85%** in average reaction time.
- SCHED_DEADLINE improves reaction times significantly.
- PREEMPT_RT further reduces reaction times.
- DL+RR remains stable under high CPU interference.
- **GPU inference** is bottleneck for **worst-case** latency.
- **Subthread scheduling** via SCHED_RR boosts consistency.

realtime kernel can improve average reaction time.

Let’s build the safest, smartest cobots together!

Picture source

[1] https://ifr.org/industrial-robots

[2] https://www.automate.org/robotics/blogs/what-are-the-4-types-of-collaborative-robots

[3] https://www.universal-robots.com/products/ur10e/

[4] https://victorzhou.com/series/neural-networks-from-scratch/

[5] https://www.pngegg.com/en/png-eekwz

[6] https://www.intelrealsense.com/depth-camera-d435/

[7] https://www.mediamarkt.be/fr/product/_extremegamer-pc-gamer-classic-level-3-amd-ryzen-7-5700x-2106882.html

[8] https://www.shutterstock.com/image-vector/ethernet-lan-wan-patch-cable-rj45-2480667835