Enabling predictable computing on reconfigurable embedded accelerators for Autonomous Vehicles

Prof. Ing. Paolo Burgio, PhD

paolo.burgio@unimore.it
paolo.Burgio@hipert.it

HIPERT

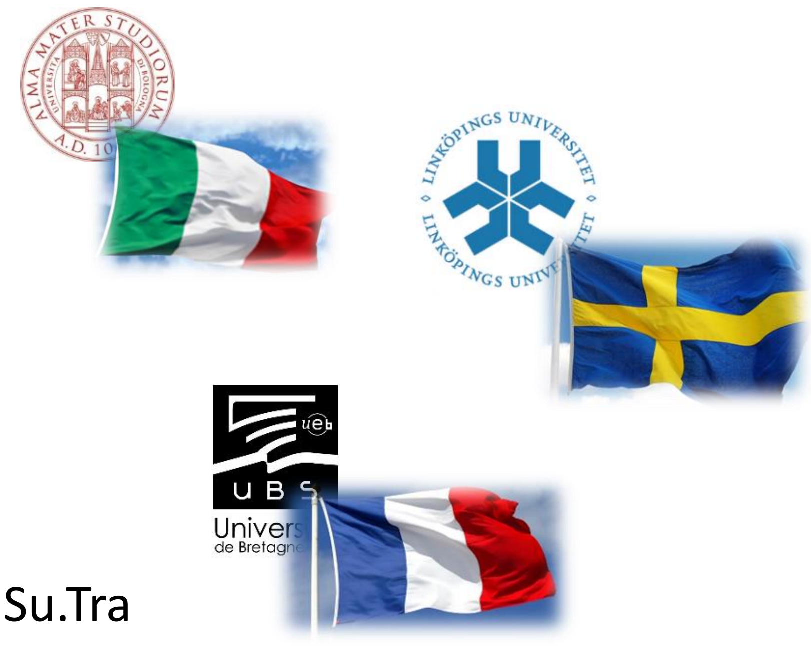
UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

High Performance Real Time Lab

Alma mater

✓ MSC Informatics Engineering @UniBo + Linkopings Universitet, 2007

✓ PhD Electronics Engineering @UniBo + UBS, 2013



Since 2014, Assistant Professor UniMoRe

✓ Co-founder of HiPeRT Lab & Hipert Srl & Su.Tra

HIPERT

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Sutra

Expertise/research areas

✓ Embedded systems programming APIs

✓ Many-core architectures

✓ Autonomous driving systems

Who am I?

HiPeRT Lab & Spinoff

- High-Performance Real-Time systems
- Autonomous Systems
  - 5 AD cars, LGV, delivery bots, aerial & (under)water drones
  - Autonomous racing: Indy Autonomous Challenge, AbuD Dhabi Racing Leaue, F1/10, F.SAE, Dallara F3
- ~70+ researchers/developers, 10+M€ funding

UNIVERSITAS STUDIORUM MUTINENSIS ET REGIENSIS 1175

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

HIPERT

✓ EU projects:

CLASS, PRYSTINE, SECREDAS (Digital Privacy Safety & Security), FRACTAL EDGE, I-MECH, ENABLE-S3, COMP4DRONES, 5G CARMEN, HERCULES, NEWCONTROL, TACLE (Timing Analysis on Code-Level), InSecTT, tetracom, 5GMETA, P-SOCRATES, OPEN-NEXT

✓ Industrial collaborations:

NVIDIA, TII Technology Innovation Institute, Ferrari, ocme Moving Ideas, EVIDENCE, ROBOPAC, SYSTEM Logistics, Tetra Pak, LYNX Software Technologies, SACMI, EXPERT SYSTEM Semantic Intelligence, XILINX, ARM, dallara, Maserati, MARELLI, United Technologies, doxee, HUAWEI, BOSCH, EGICON, bylogix, 3D[iVE]

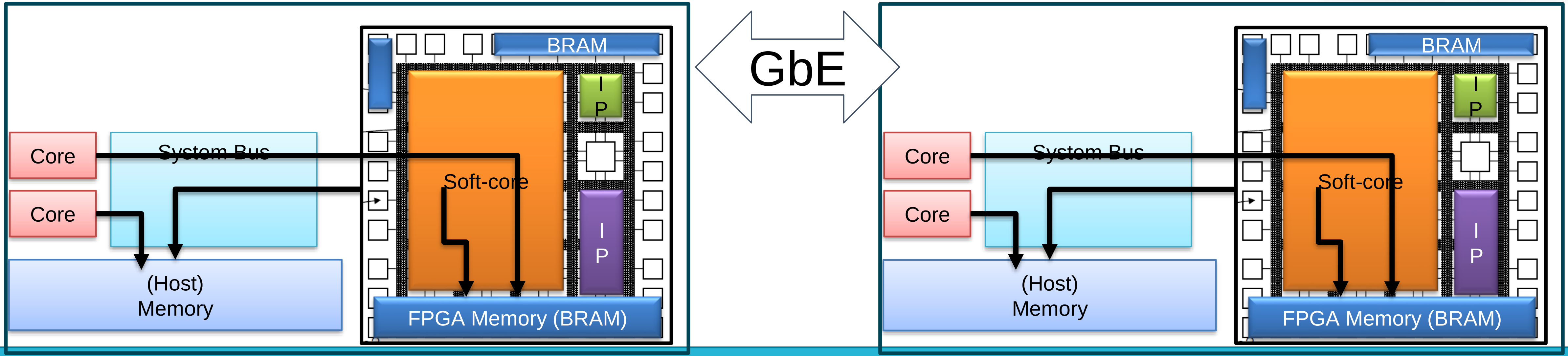
Three successful stories

1. Timing-Sensitive Autonomous Architectures
2. Accelerated 2D localization for embedded reconfigurable computers
3. Memory interference mitigation in embedded architectures

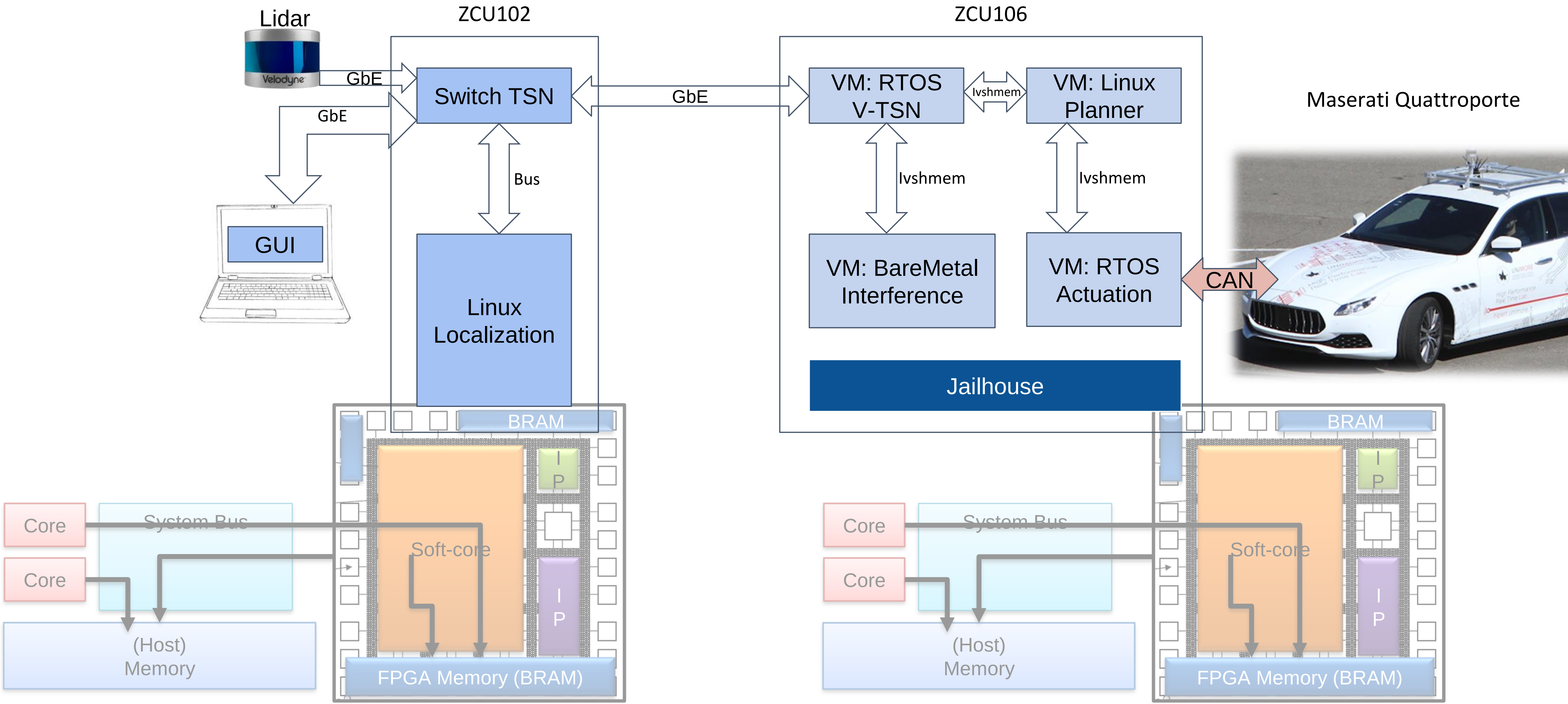
TSAA - Timing-Sensitive Autonomous Architecture

- Based on Time-Sensitive protocol - TSN
- Target platforms: two dev-boards with reconfigurable logics (AMD Xilinx ZCU102/106)
- Goal: removing/mitigating the interference over the Ethernet links

Maserati Quattroporte

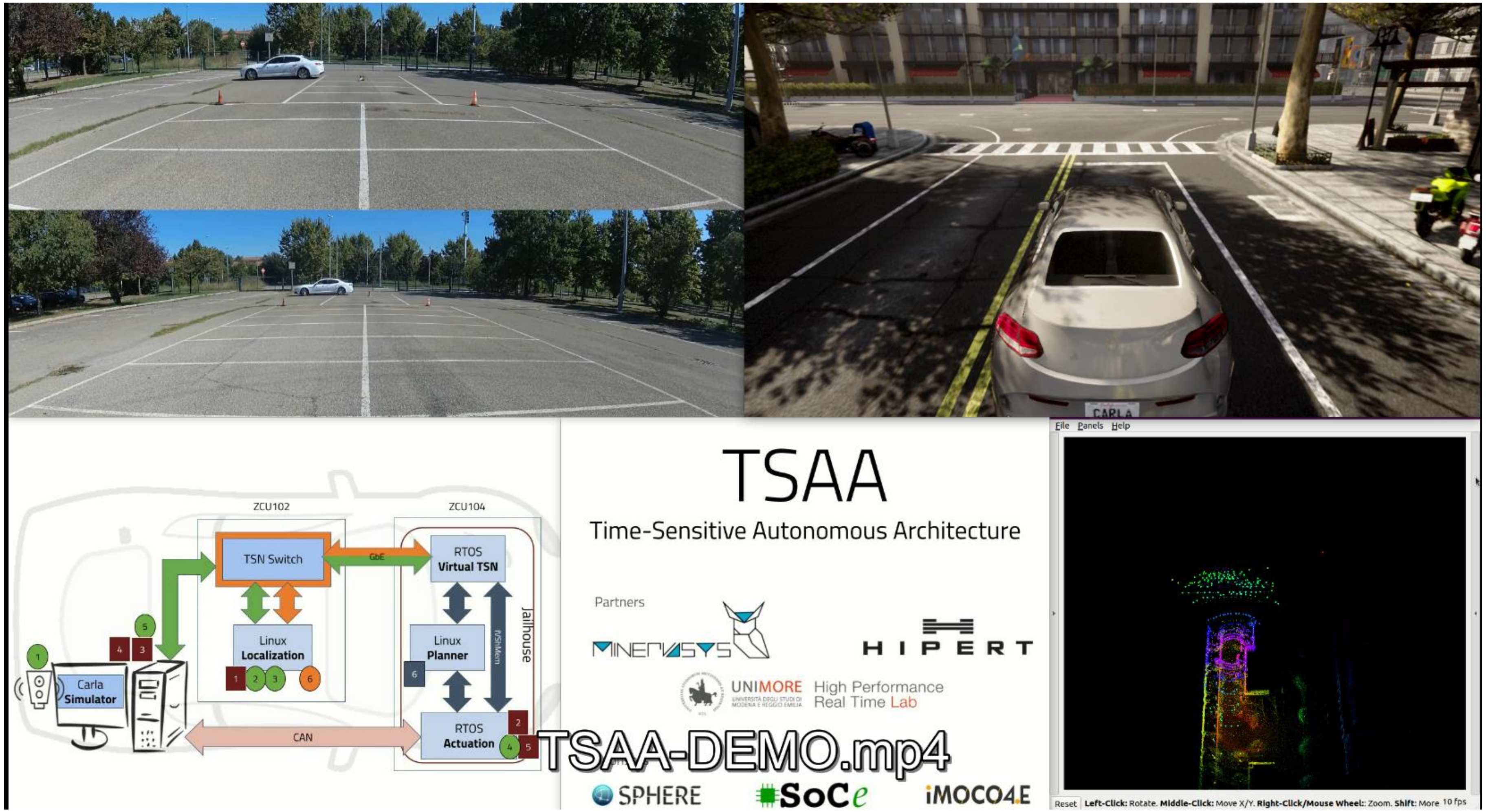


Timing-Sensitive Autonomous Architecture - Our original system design

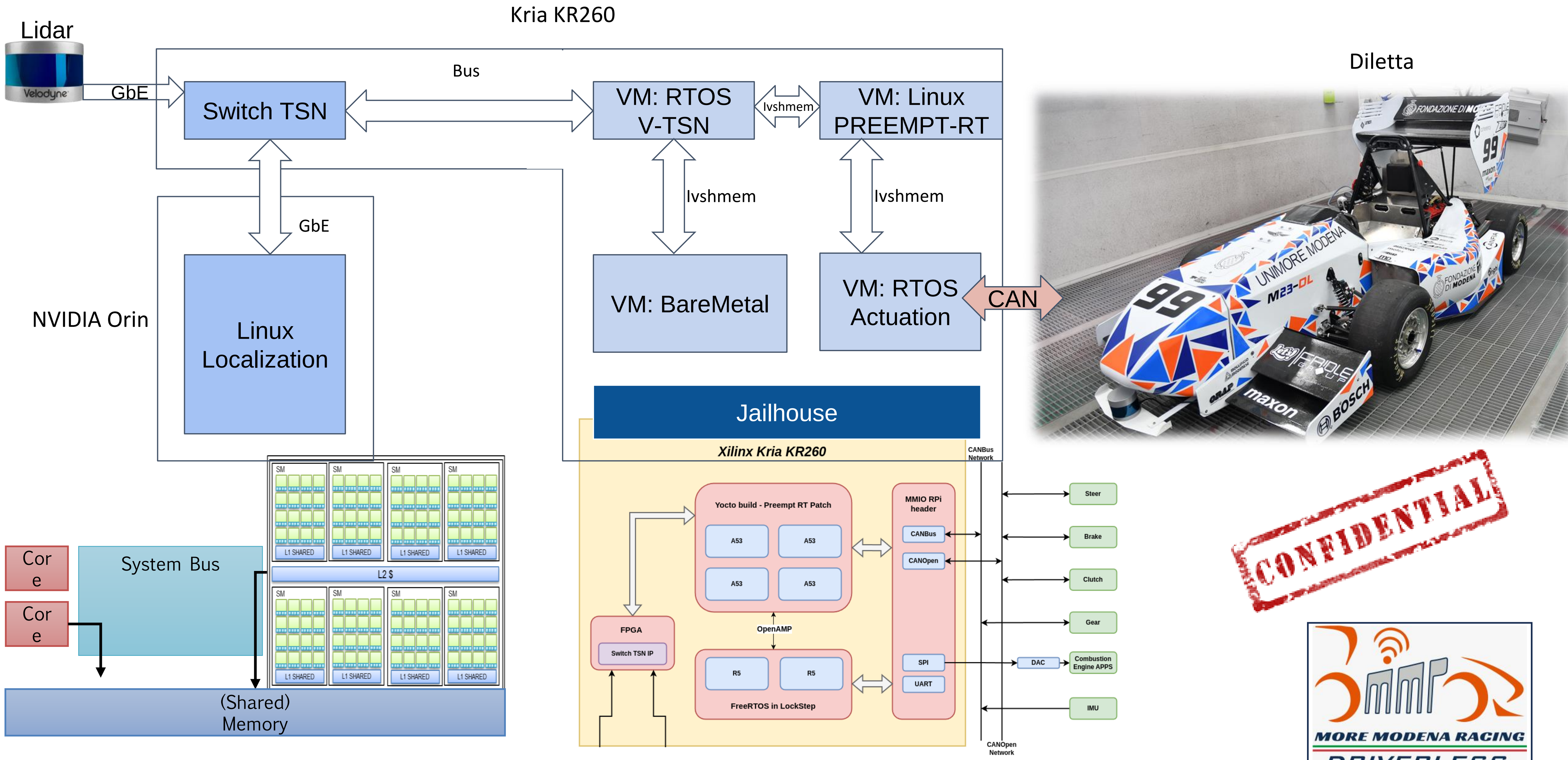


With, and without TSAA

- Video



Racing variant for Formula Student: M23-DL “Diletta”



Kria KR260

Lidar

GbE

Switch TSN

Bus

VM: RTOS
V-TSN

Ivshmem

VM: Linux
PREEMPT-RT

Ivshmem

Ivshmem

GbE

NVIDIA Orin

Linux
Localization

VM: BareMetal

VM: RTOS
Actuation

CAN

Jailhouse

Xilinx Kria KR260

CANBus
Network

Yocto build - Preempt RT Patch

A53

A53

A53

A53

MMIO RPi
header

CANBus

CANOpen

FPGA

Switch TSN IP

OpenAMP

R5

R5

FreeRTOS in LockStep

SPI

UART

Steer

Brake

Clutch

Gear

DAC

Combustion
Engine APPS

IMU

CANOpen
Network

Cor
e

Cor
e

System Bus

SM

L1 SHARED

L2 $

(Shared)
Memory

Diletta

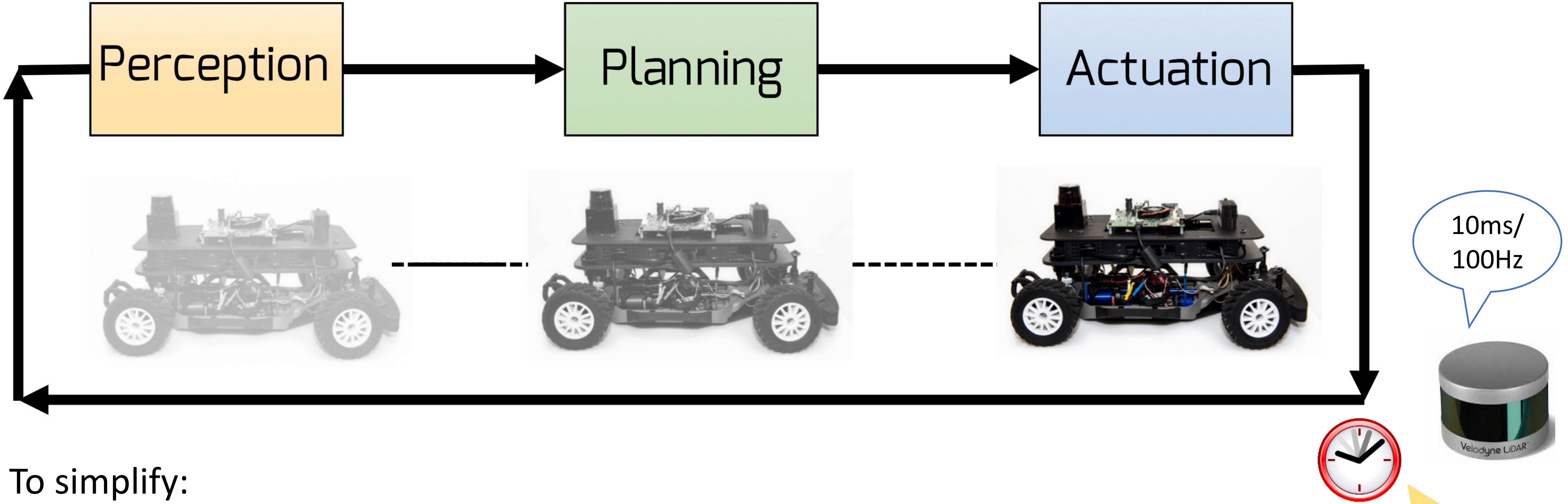
CONFIDENTIAL

MORE MODENA RACING
DRIVERLESS

Real-Time systems
on embedded accelerators

Stories #2 and #3

Brief recap: Autonomous Driving in a nutshell



Real-Time constraint!

To simplify:

✓Three main blocks mimicking humans

✓**Run in Real-Time**, to meet sensors frequency (cameras, LiDARS, etc)

✓Note: **Perception processes the highest quantity of data**

A motivational example:
vehicle localization with 2D particle filtering

Localize the vehicle on a **known** 2D map

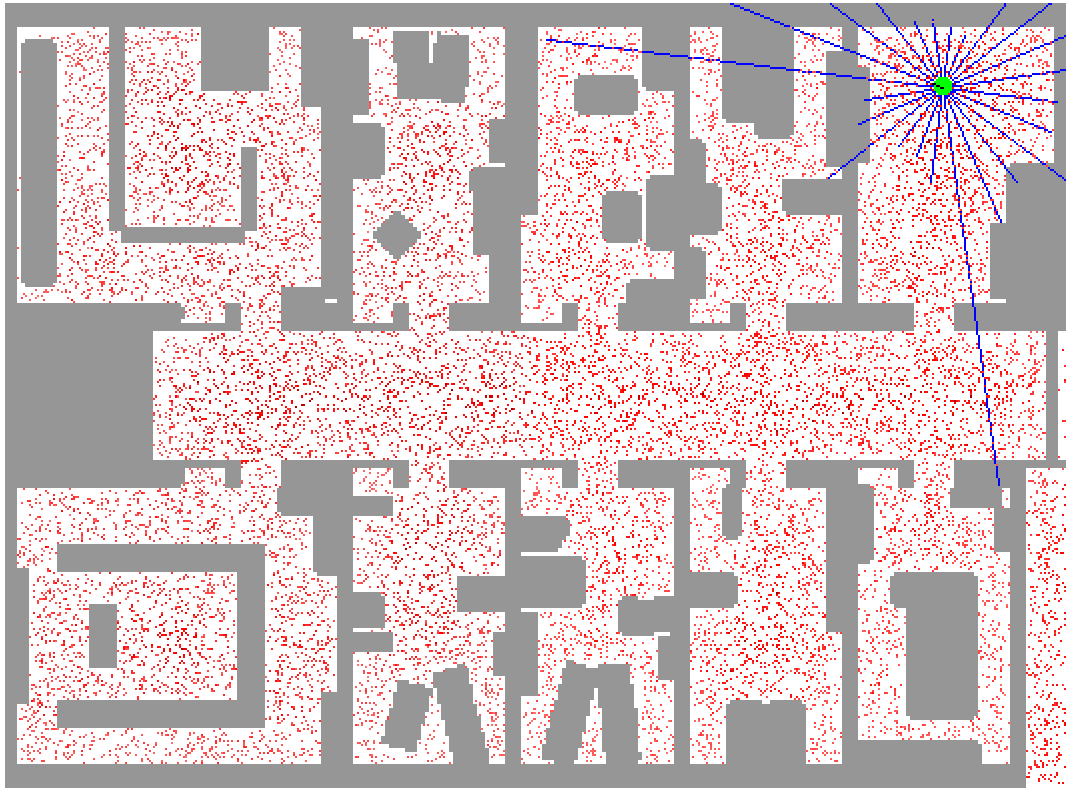
- ○ Randomly generate position candidates (particles)
- ○ For each of them, simulate the sensor model, i.e., the LiDAR rays
- ○ Compare against actual sensor input

Cons

- × Require thousands of particles and rays to achieve good accuracy for estimate position
- × High computational load
- × Statistical-based approach (not deterministic)

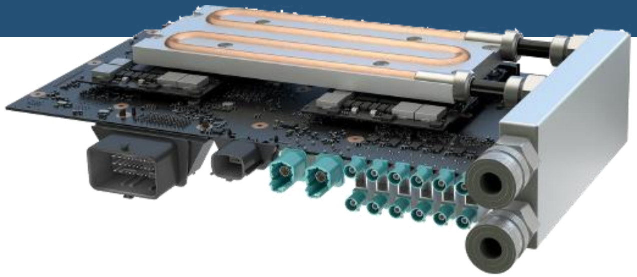
Pros

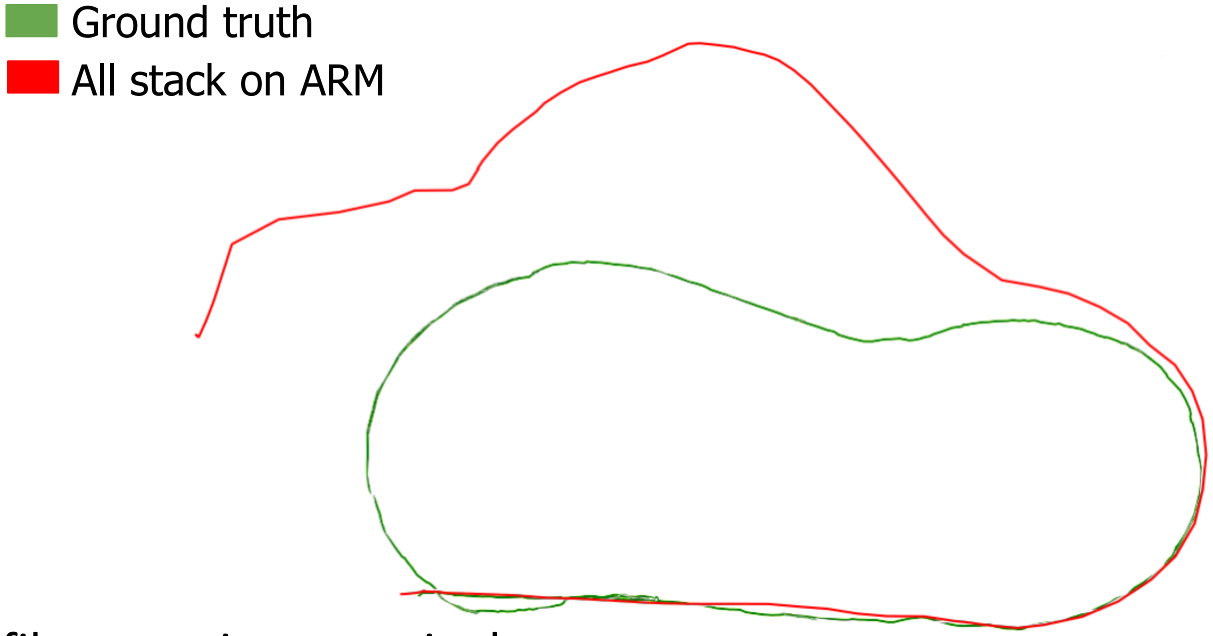
- ✓ Can compute particles independently
- ✓ **Embarrassingly parallel**
- ✓ High accuracy, **if computation adequately optimized**



Why embedded computers for autonomous driving?

Embedded computers are slow...





Particle filter running <u>on a single core</u>

- (F1/10 autonomous racing vehicle)
- Either you slow down…
- …or you crash

…but they are heterogeneous architectures

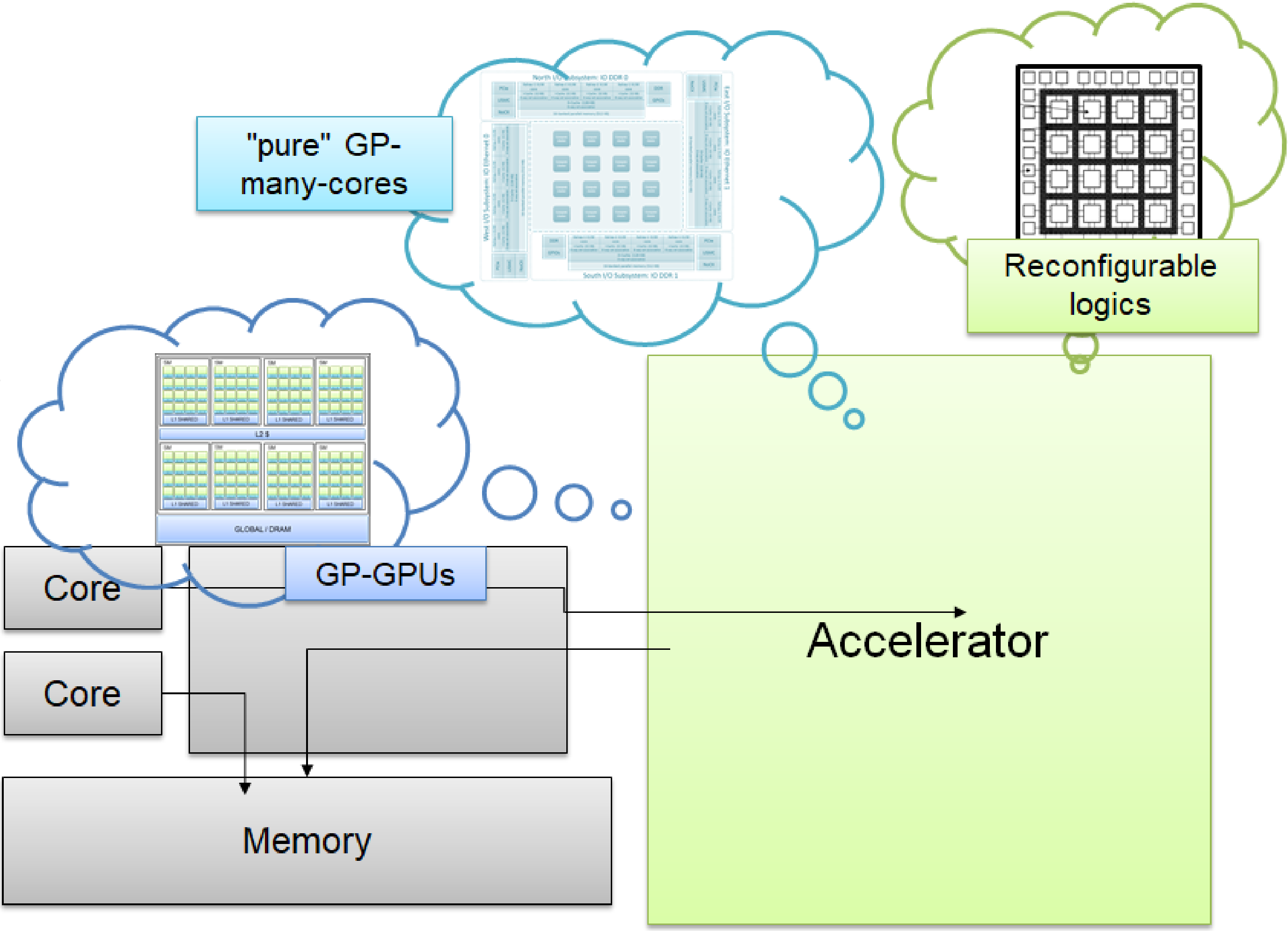
Vehicles will feature few ECUs/domain controllers with:

- Host (ARM-like) multi-cores
- Real-Time processors (e.g. Aurix Tricore)
- Embedded data crunching accelerators

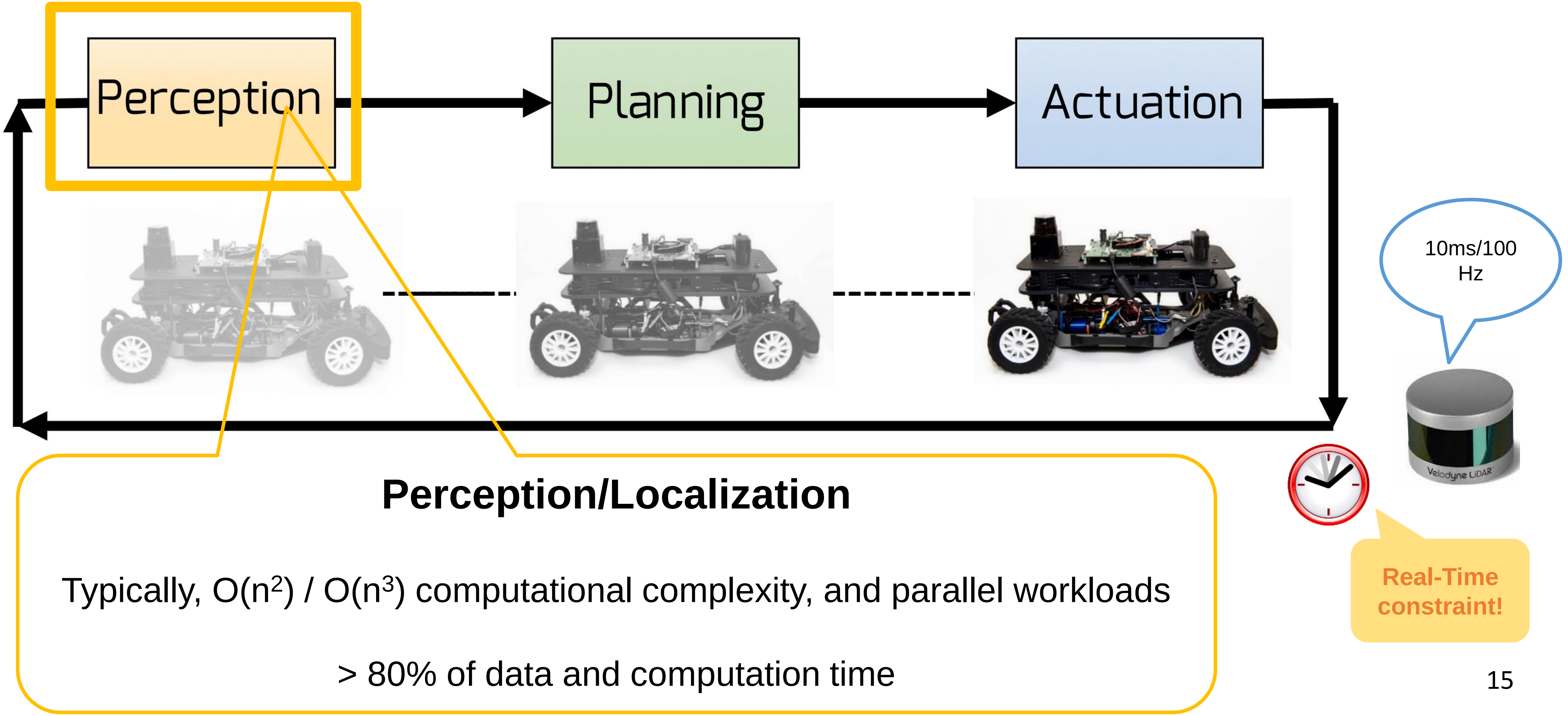
For the accelerators, there are multiple choices

- GPGPUs are prominent, and very good for prototyping
- In-house ASICs for highest Perf/Watt
- Programmable logics enable HW design space exploration

Sharing memory blocks to implement efficient data transfers



Data crunching accelerators – How to use them?



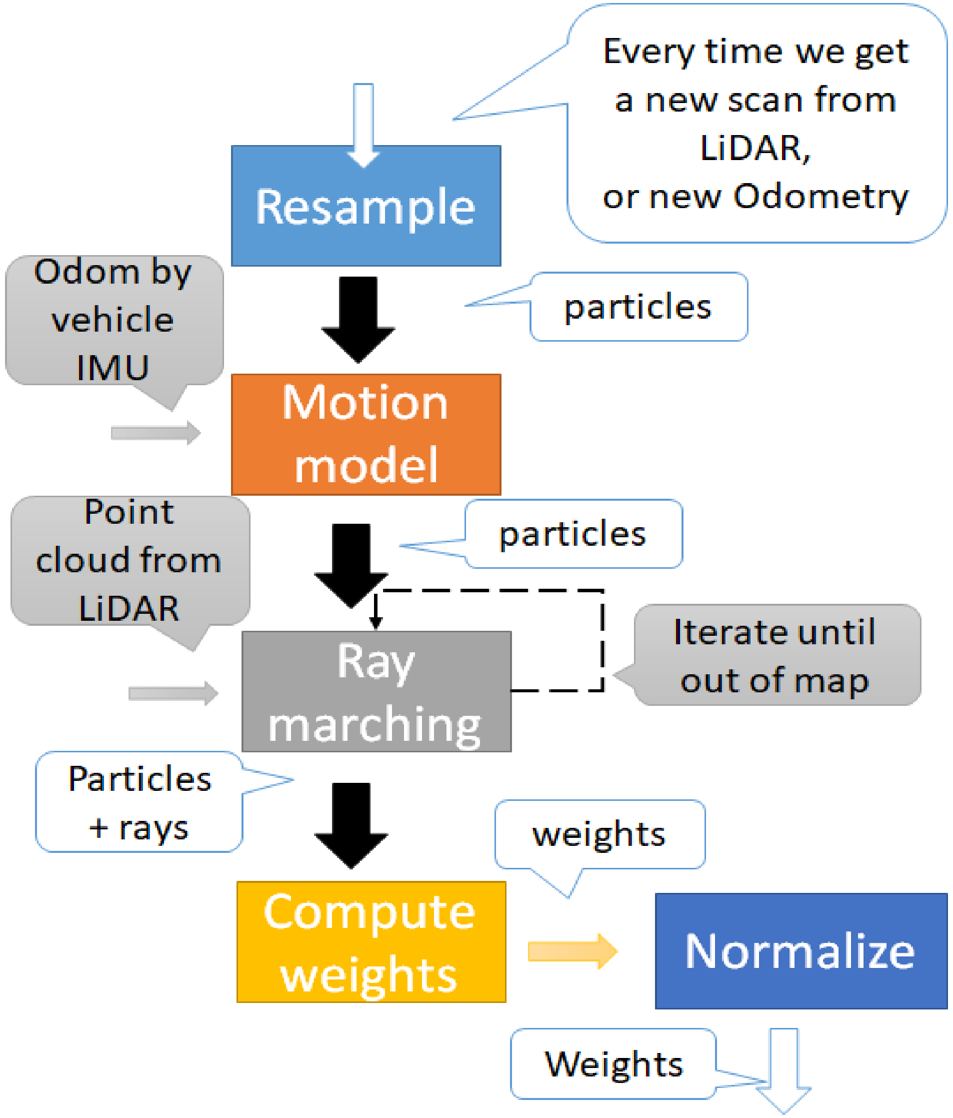
Real-Time constraint!

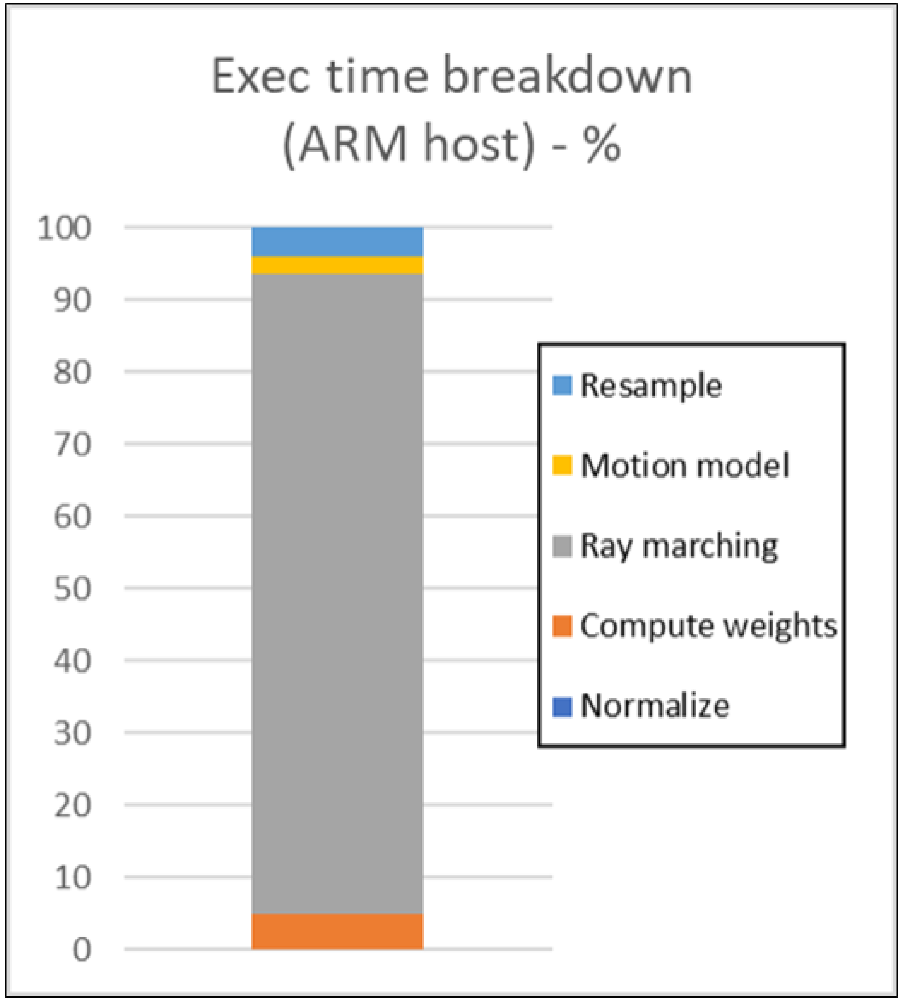
Perception/Localization

Typically, O(n^2) / O(n^3) computational complexity, and parallel workloads

> 80% of data and computation time

Step 1: profiling of Particle Filter stages





Step 2: our Ray Marching Engine (RME) IP

Parametrizable

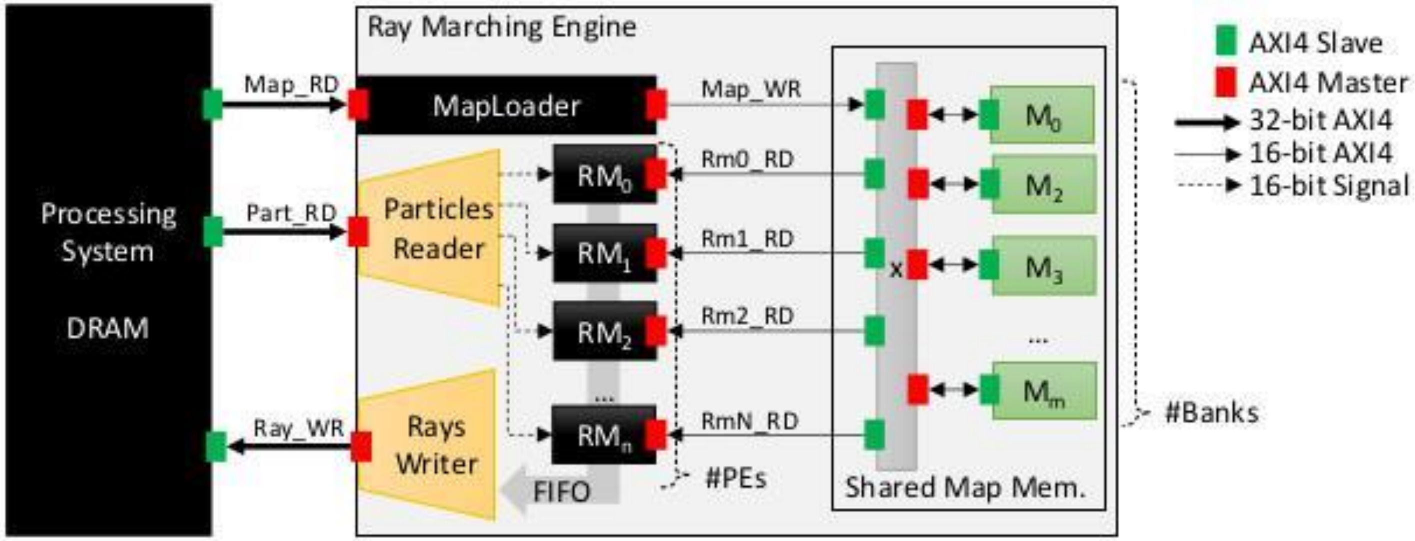
- Can process N rays in parallel

Target: AMD Xilinx FPGAs

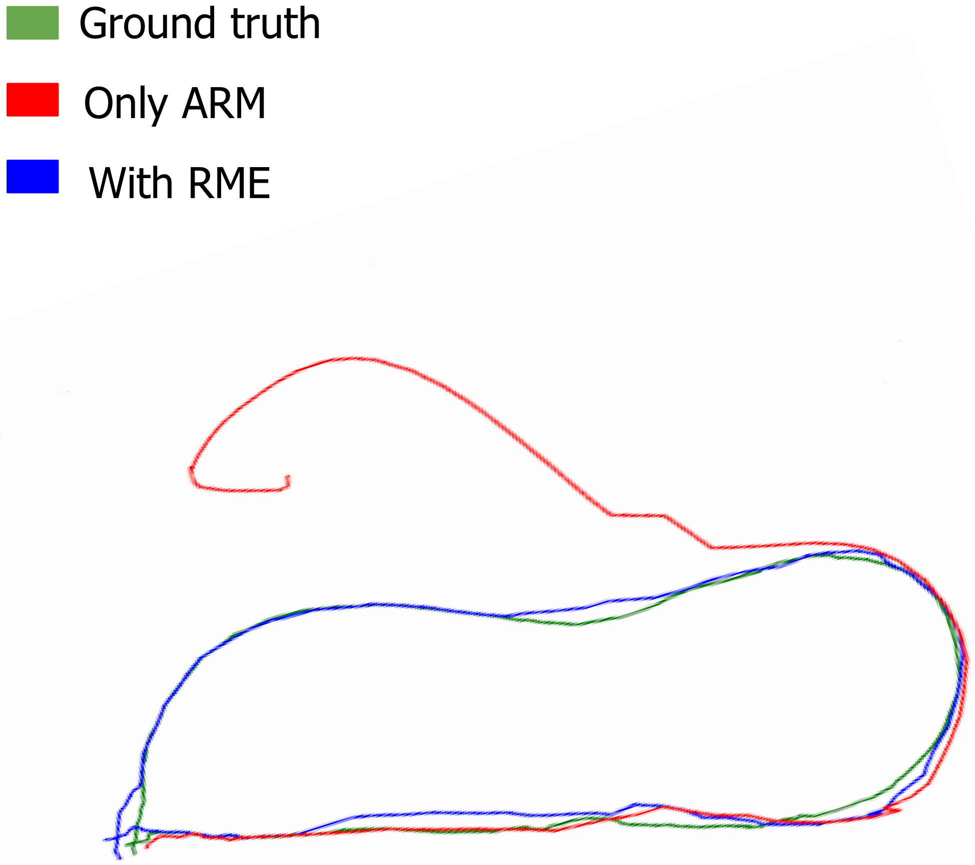
- Standard AMBA AXI4 interface

Highly data intensive

- Data stored in system DRAM

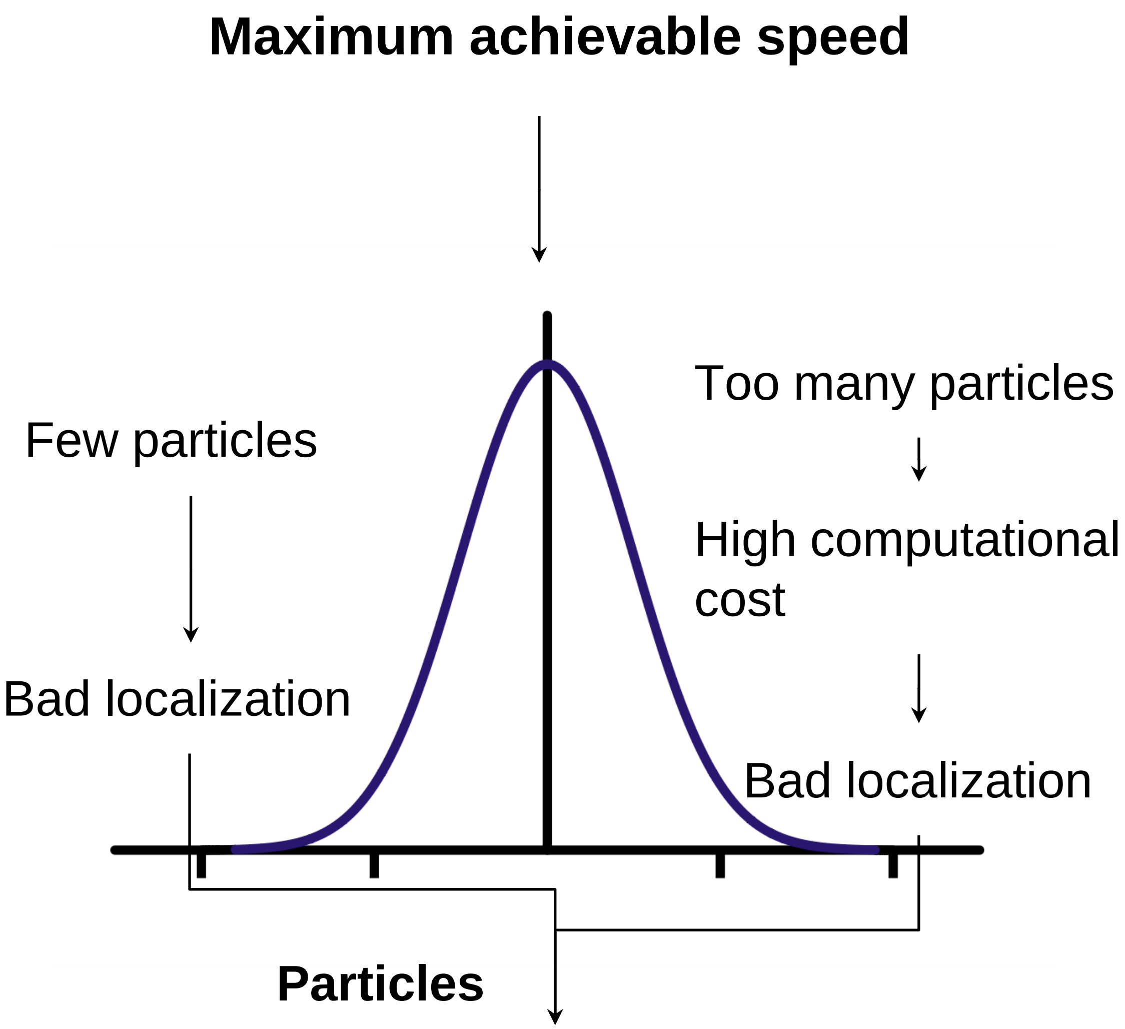


First results - Simulator and the F1/10



- Localization is **still too slow**
- We need to **cap the maximum speed** of the vehicle to avoid crashes
- Still needs improvement, in racing

Find the sweetspot: accuracy vs performance



Maximum achievable speed

Few particles

Bad localization

Too many particles

High computational cost

Bad localization

Particles

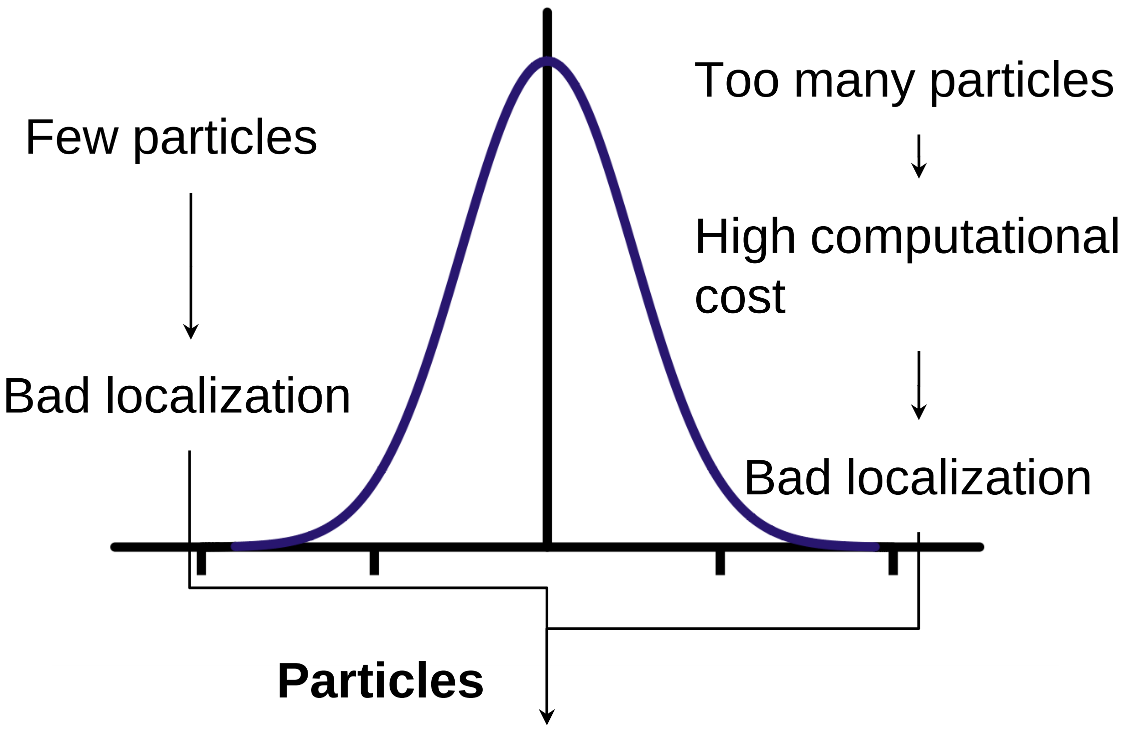
Must lower the speed, to avoid crashes

Find the sweetspot: final results

2 target platforms, from AMD Xilinx

- ARM multi-core as reference

1. Ultra96 – low-end (smaller programmable area)
2. ZCU 102 – high-end (larger programmable area)



Few particles

Bad localization

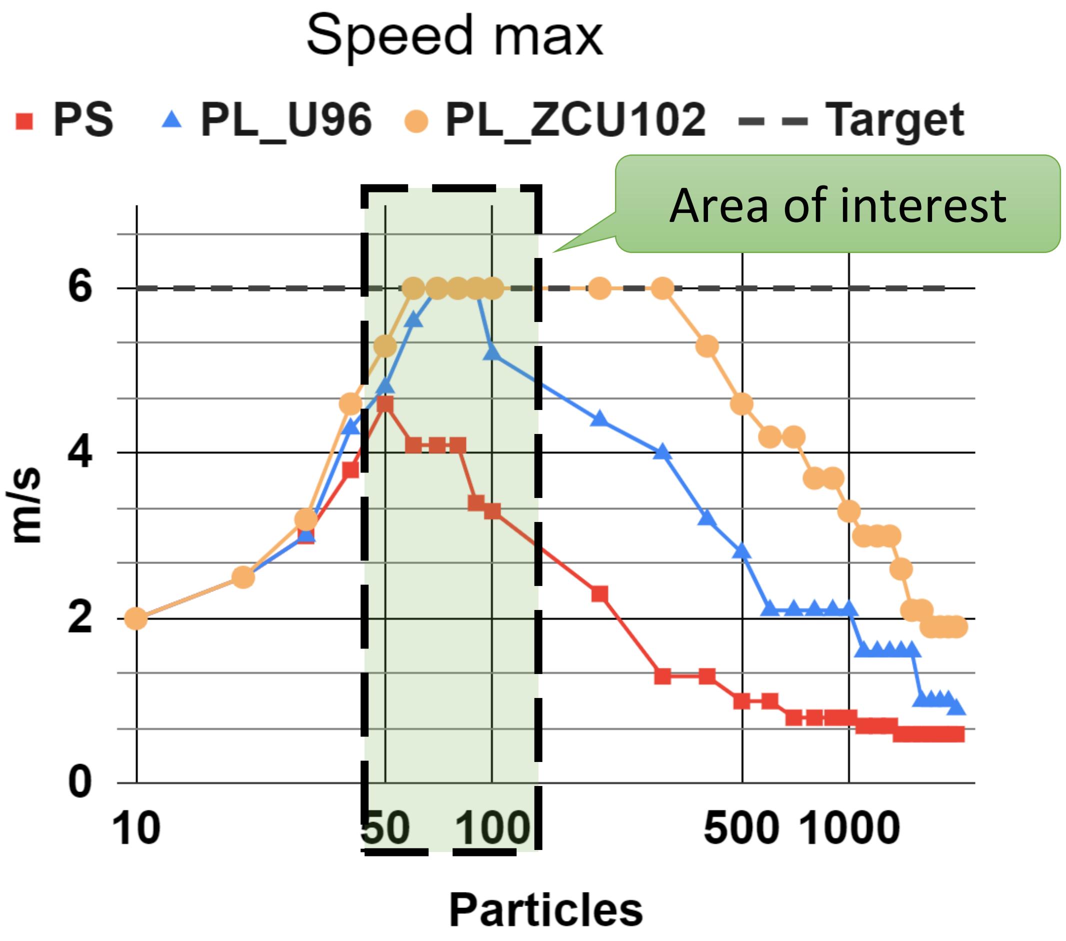
Too many particles

High computational cost

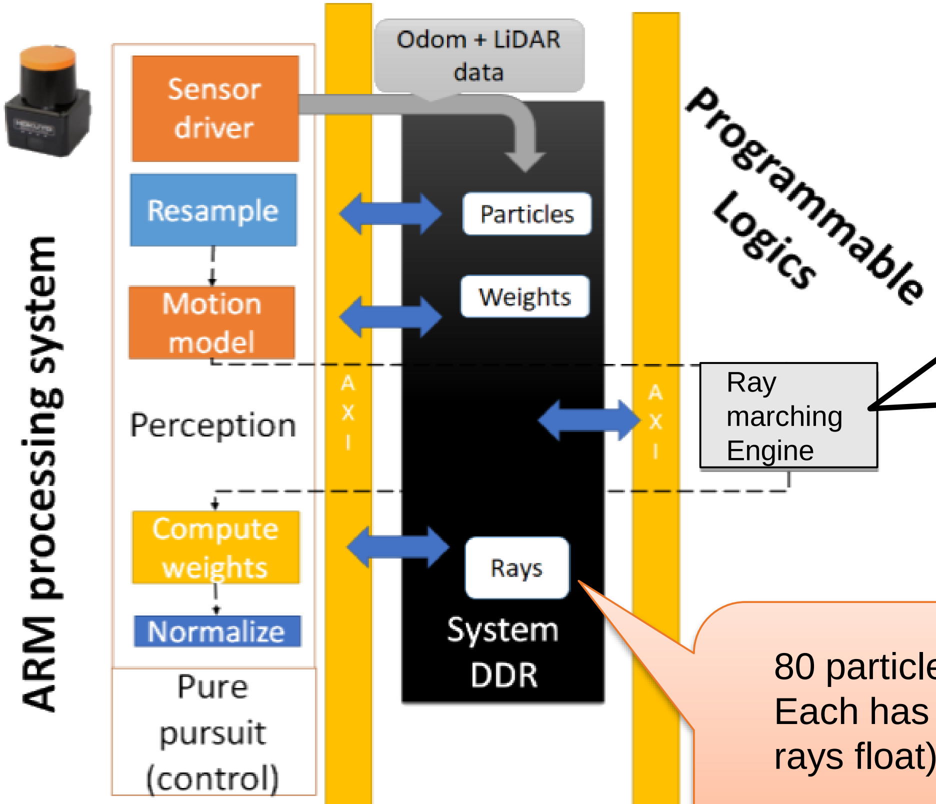
Bad localization

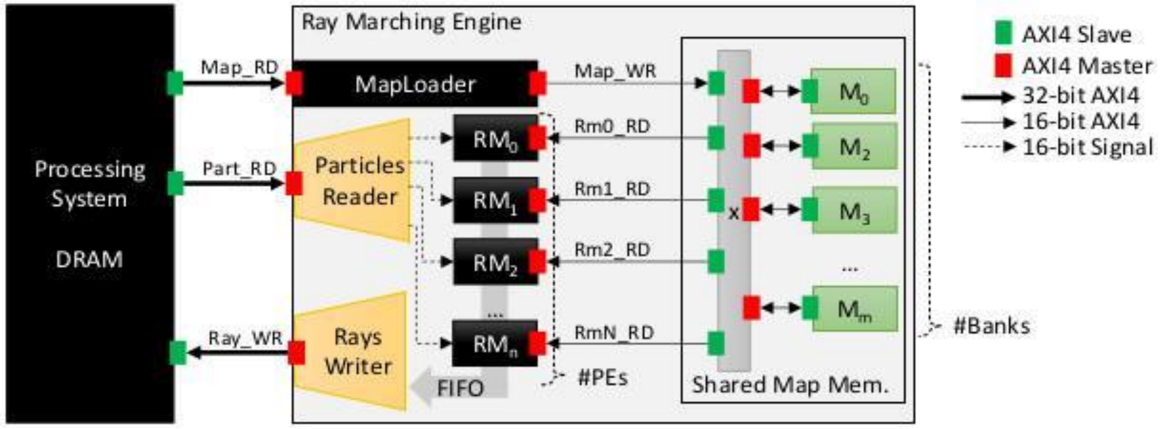
Particles

Must lower the speed, to avoid crashes



Basic system design





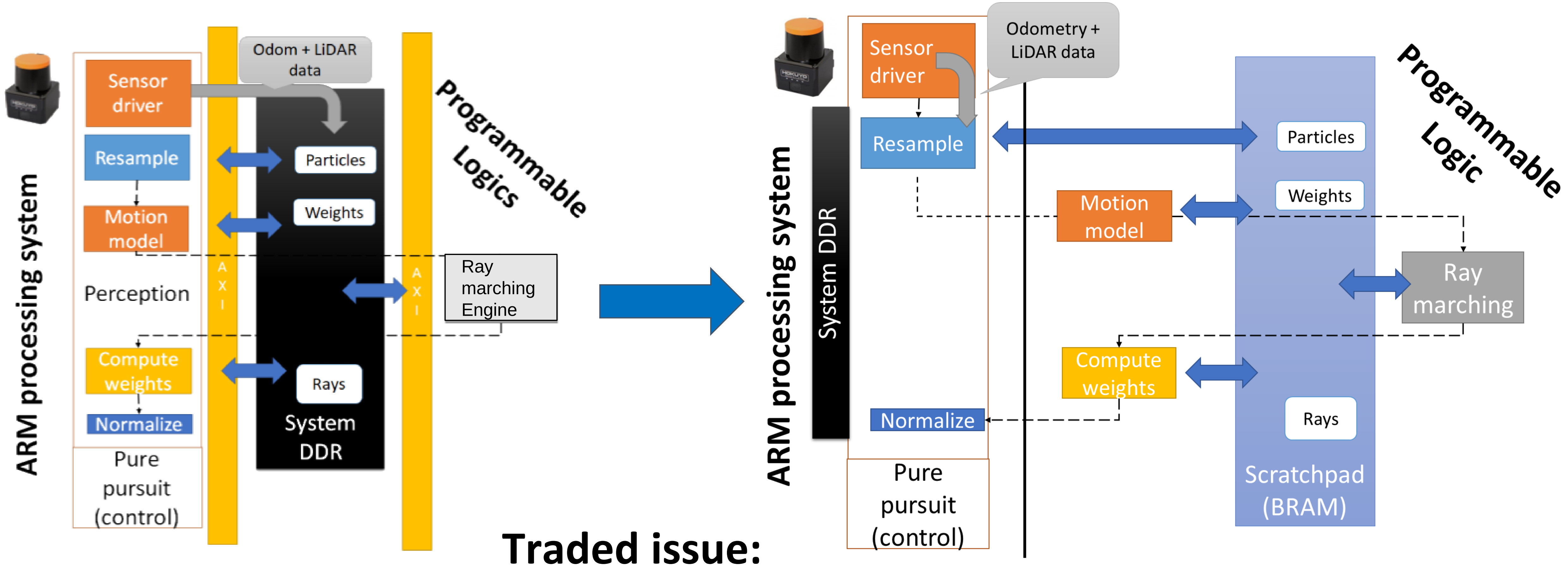
80 particles
Each has a PCL (60 rays float)

➔ 60*80*4 = **19.2 Kb**

Main issue:

Data transfer to/from the system DRAM banks are still deteriorating overall performance

Reduce data transfer time, by offloading other modules



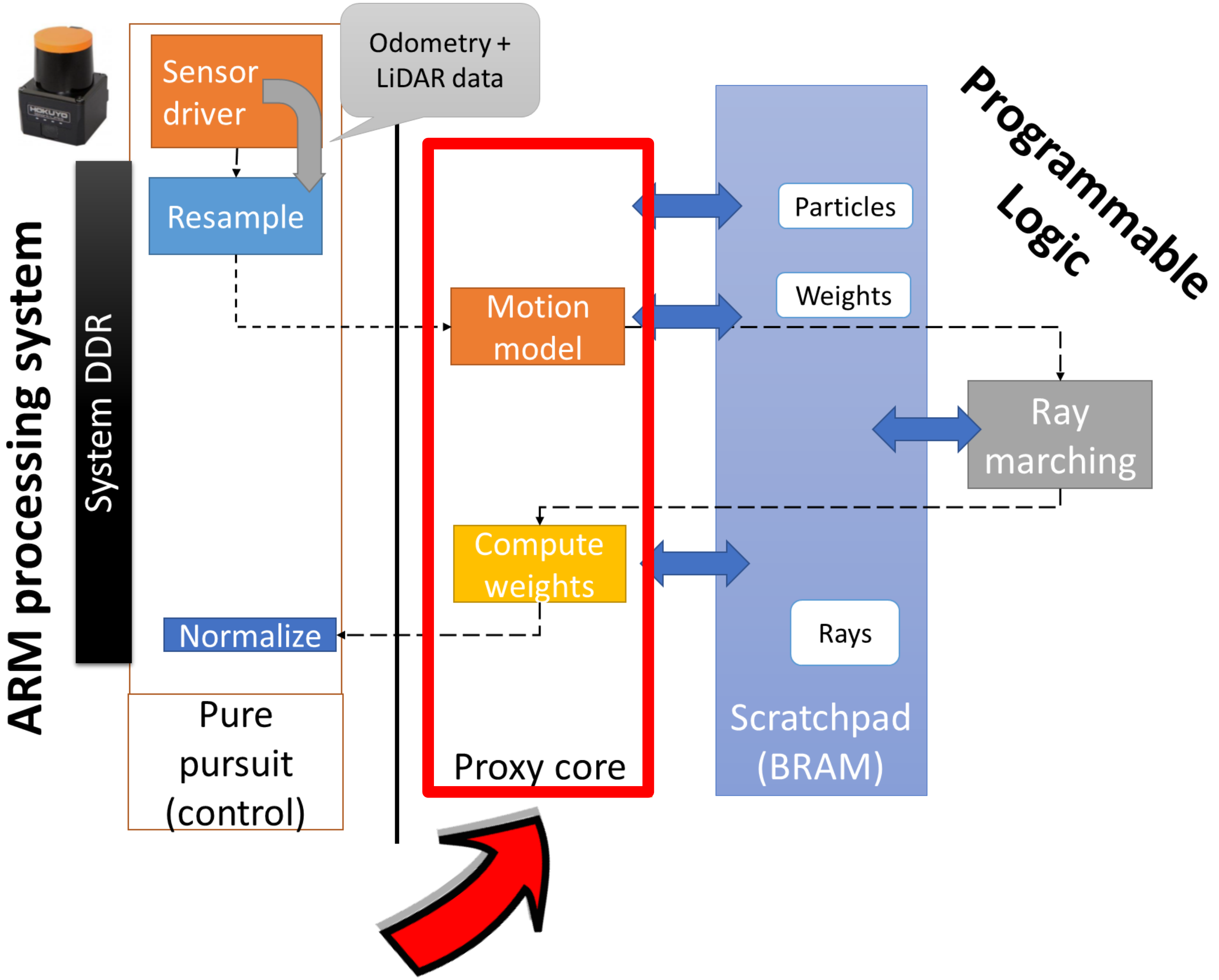
Traded issue:

area in the programmable logics is limited,
so the HW system design is not feasible

Optimal solution: proxy core and local data

Key features:

- Deploy only data-crunching IP (RME), together with a **programmable proxy soft-core**
- Run other computations on the proxy core, to tackle area limitations
- Data is **stored in BRAM local memory**, to minimize transfer latency



Take-aways and research ideas

Goals achieved:

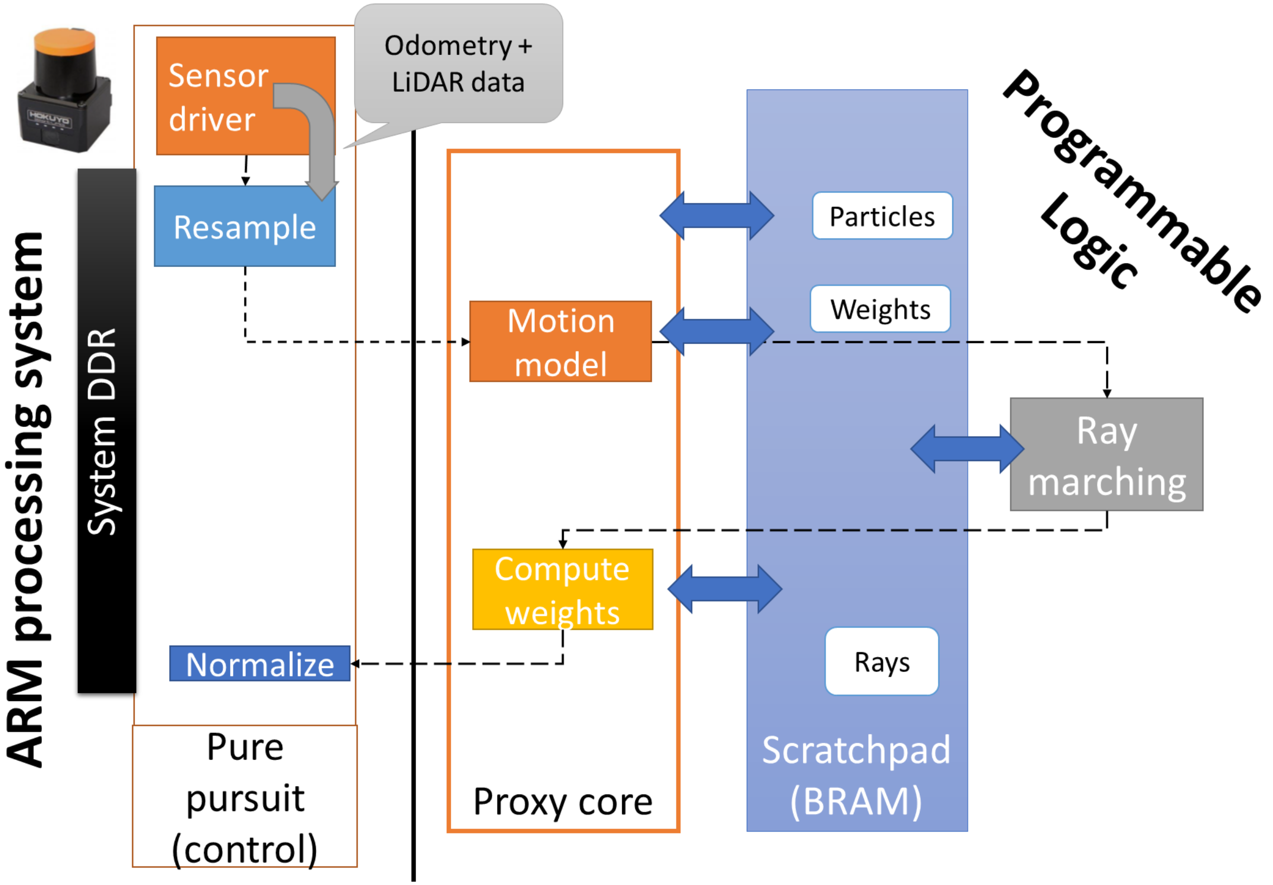
✓ System design with **proxy core** to tackle area limitations

✓ Reduce data transfers through **local BRAM buffers**

✓ **Generic methodology**, can be applied to any data-intensive algorithm

Ongoing research:

1. System **HW/SW codesign** will be **automated**
2. **Programming** model (mixed **HW/SW API**)
3. High contention on memory banks, system quickly becomes **unpredictable**



Take-aways and research ideas

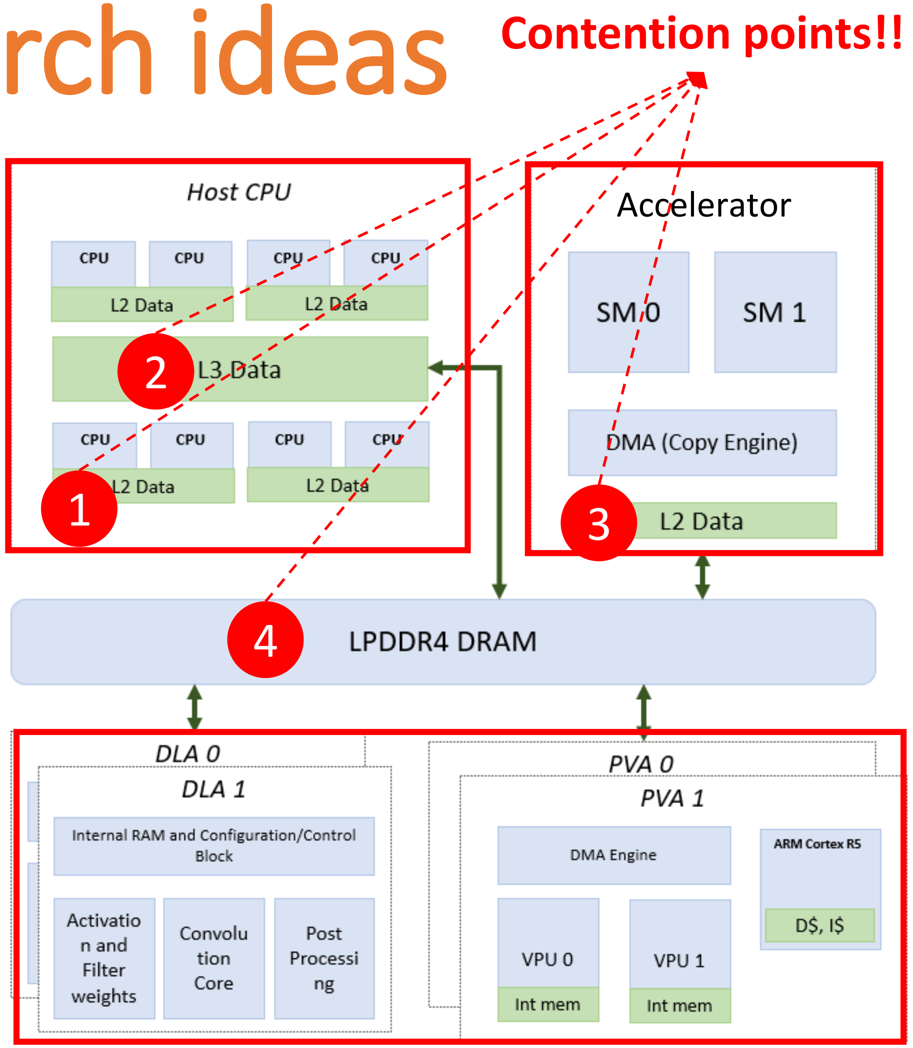
System design can be integrated within the programming API, e.g., OpenMP

- We are creating a fully automated design flow to create accelerator-rich platforms starting from code written in high-level language
- Accelerator programming library and API that bridges between proprietary low-level code, and high-level code
- Analyze and tackle memory interference, and propose techniques to mitigate it, and enabling safety-critical systems

Ongoing research:

1. System **HW/SW codesign** will be **automated**
2. **Programming** model (mixed **HW/SW API**)
3. High contention on memory banks, system quickly becomes **unpredictable**

Take-aways and research ideas


Contention points!!

Ongoing research:

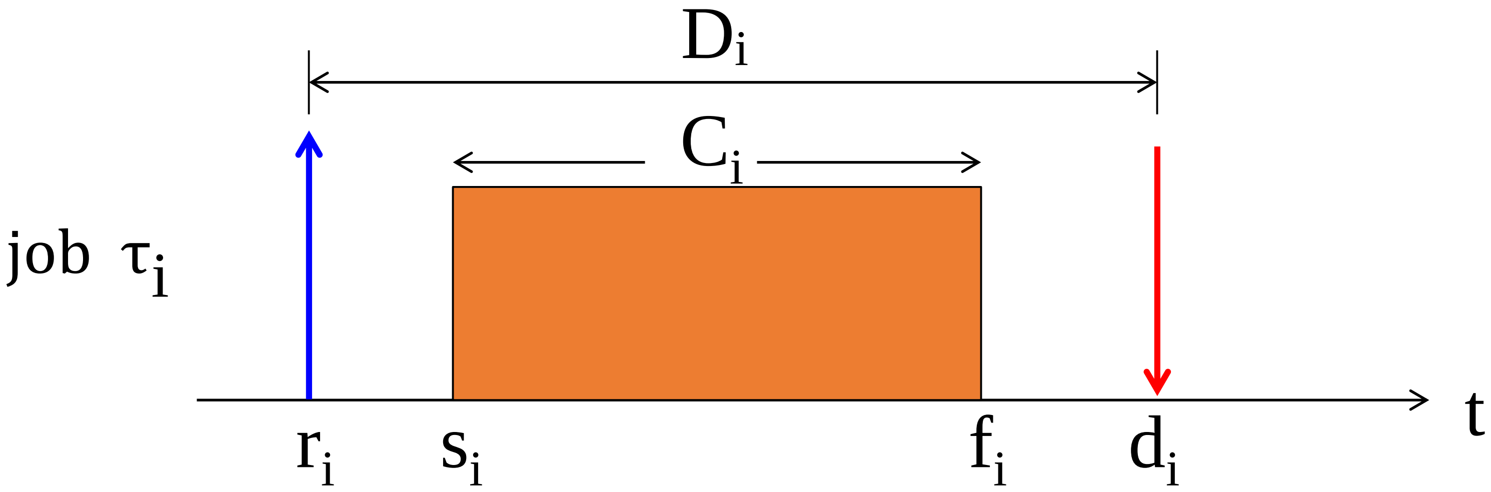
1. System **HW/SW codesign** will be **automated**
2. **Programming** model (mixed **HW/SW API**)
3. High contention on memory banks, system quickly becomes **unpredictable**

Recap: Real-time tasks model

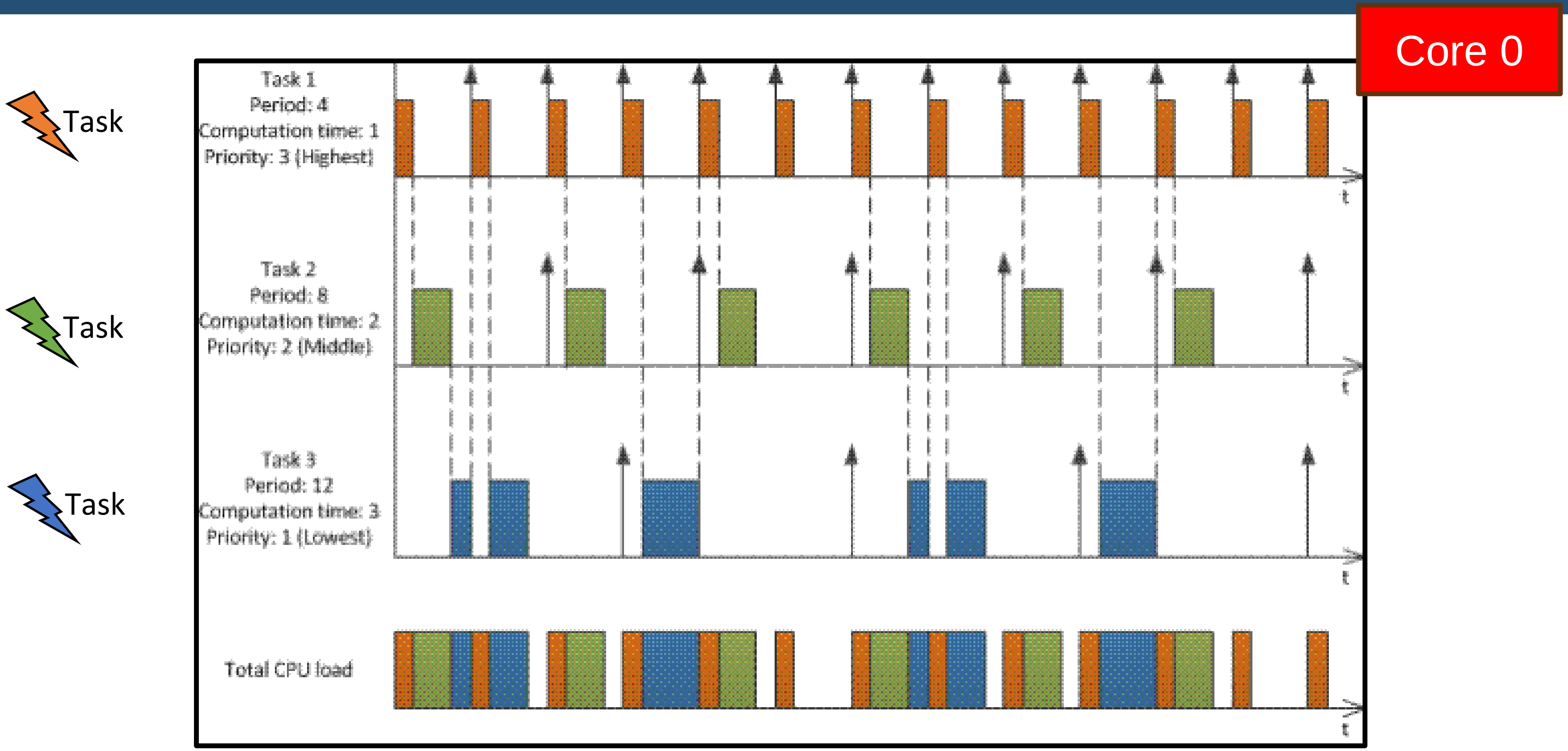
Tasks are split into *N jobs*

- r_i request time (arrival time a_i)
- s_i start time
- C_i worst-case execution time (WCET)
- d_i absolute deadline
- D_i relative deadline
- $f_i$finishing time

Task



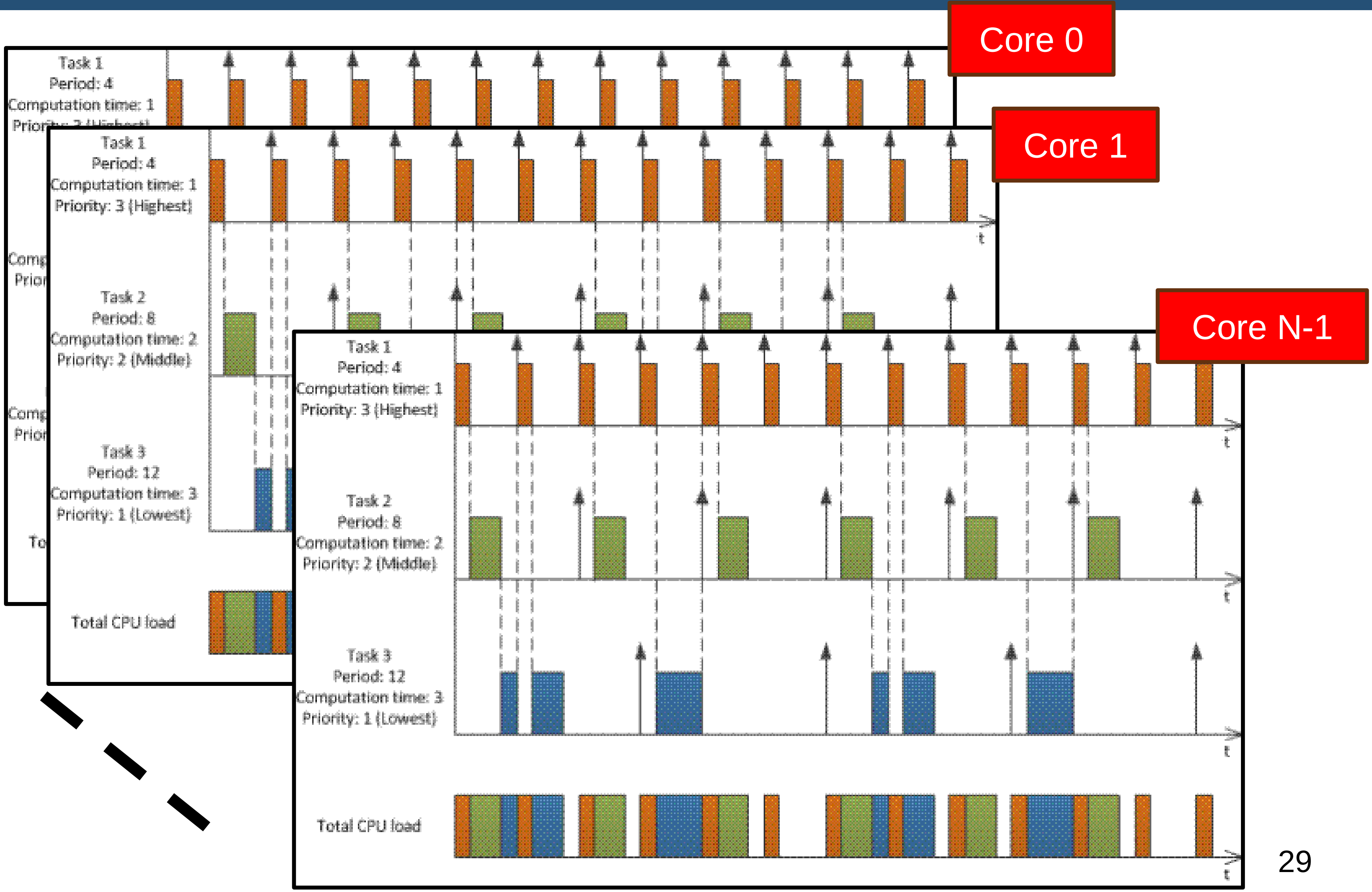
RT scheduling in single cores



“Standard” model for Task scheduling

- Jobs from multiple tasks (here, 3) are scheduled onto the available core

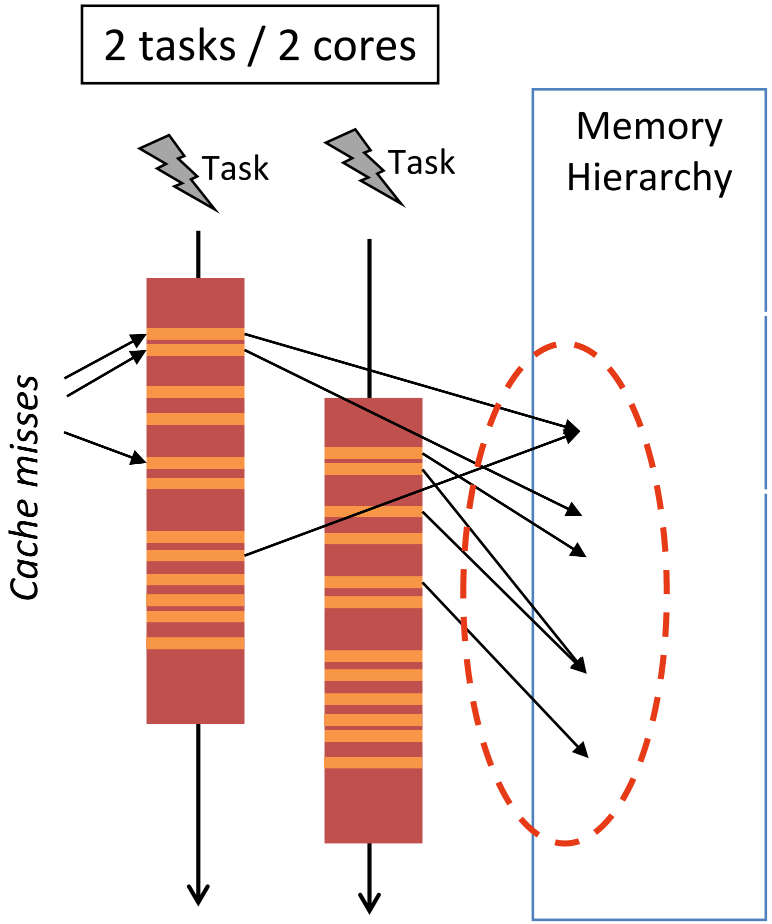
RT scheduling in multi cores



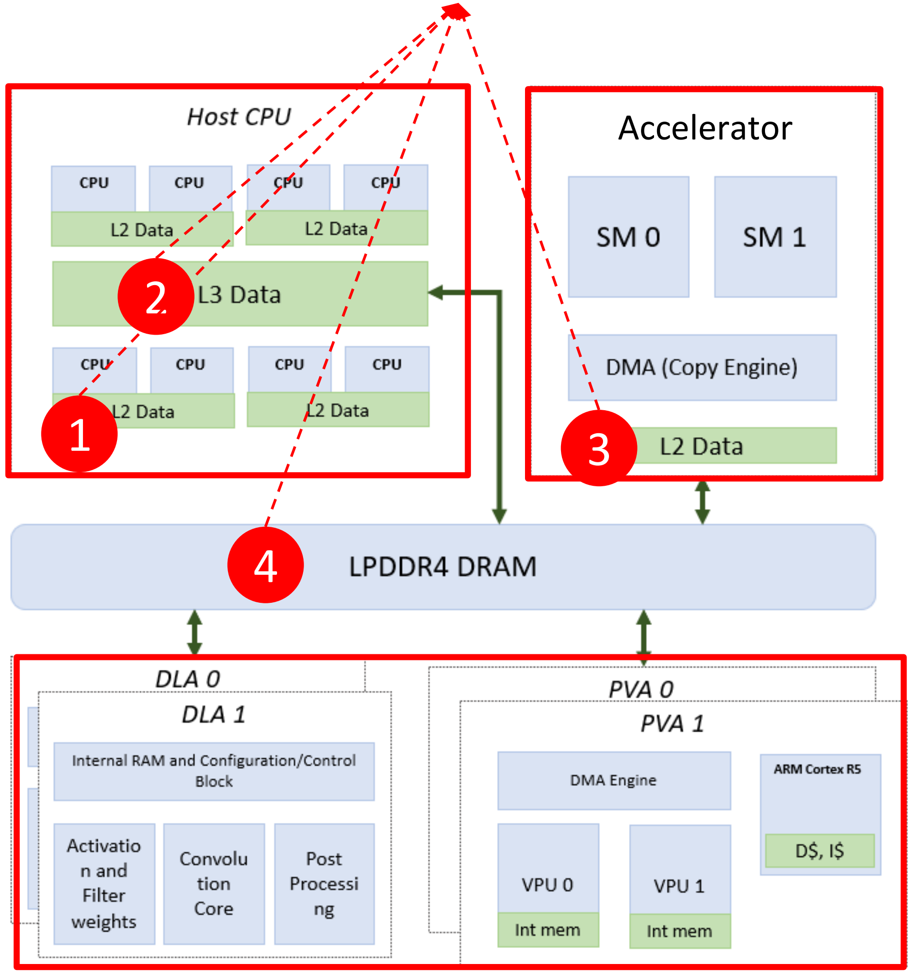
The problem scales well to multi-core systems

(…more or less…)

Uncontrolled memory accesses in multi cores

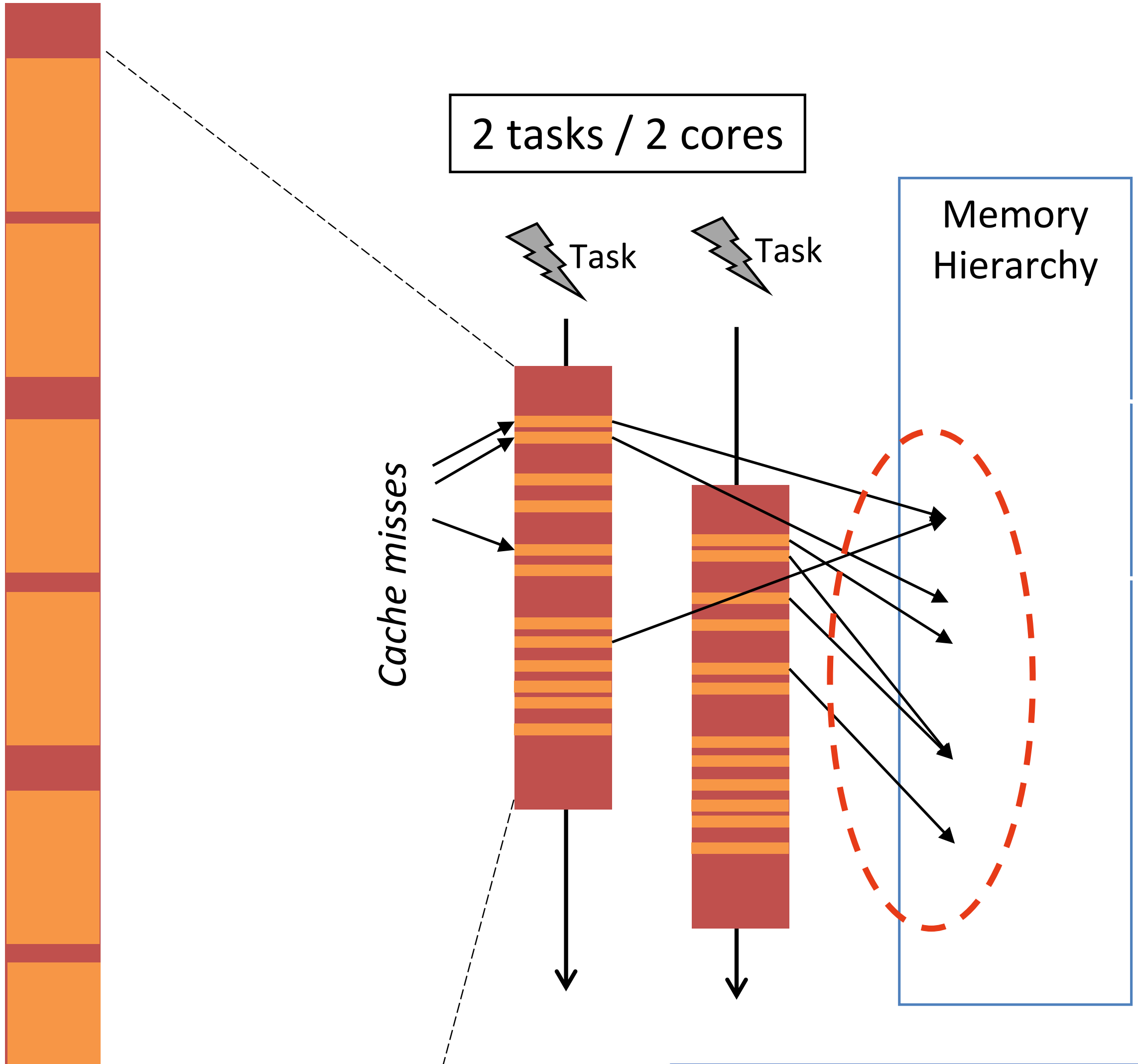


A lot of contention points!!

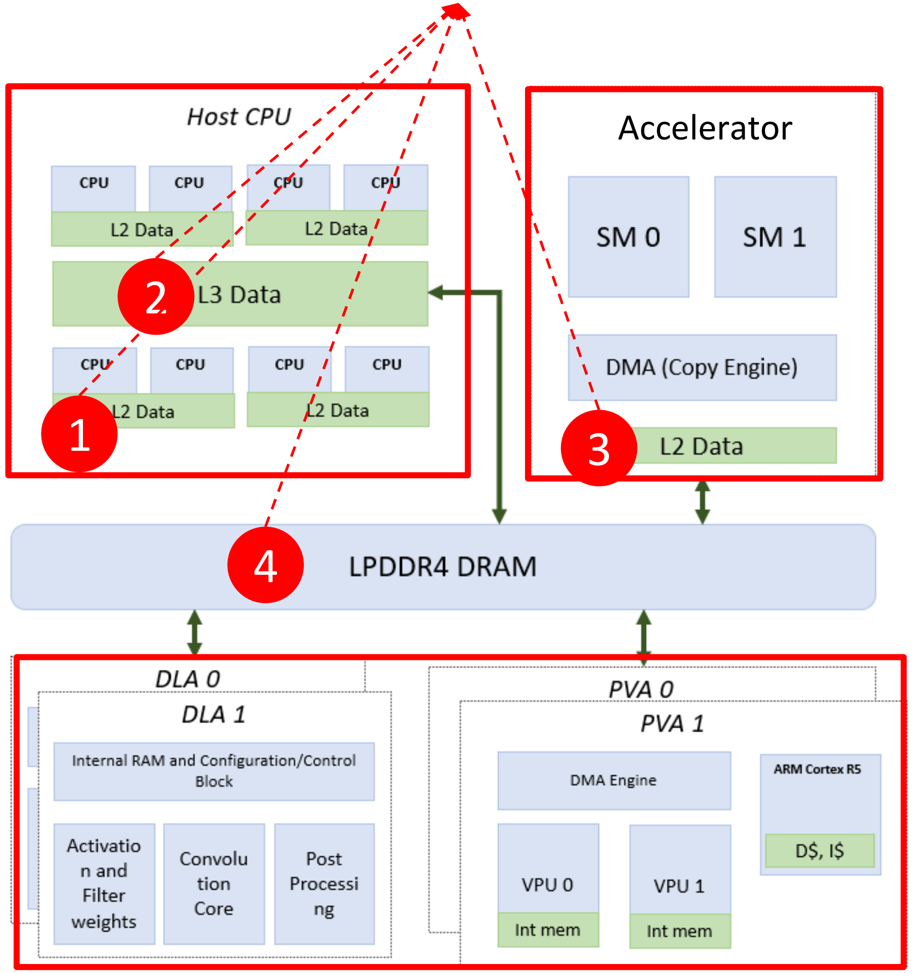


Paolo Burgio, Andrea Marongiu, Paolo Valente, Marko Bertogna, A memory-centric approach to enable timing-predictability within embedded many-core accelerators, 2015 CSI Symposium on Real-Time and Embedded Systems and Technologies (RTEST)

Uncontrolled memory accesses in multi cores

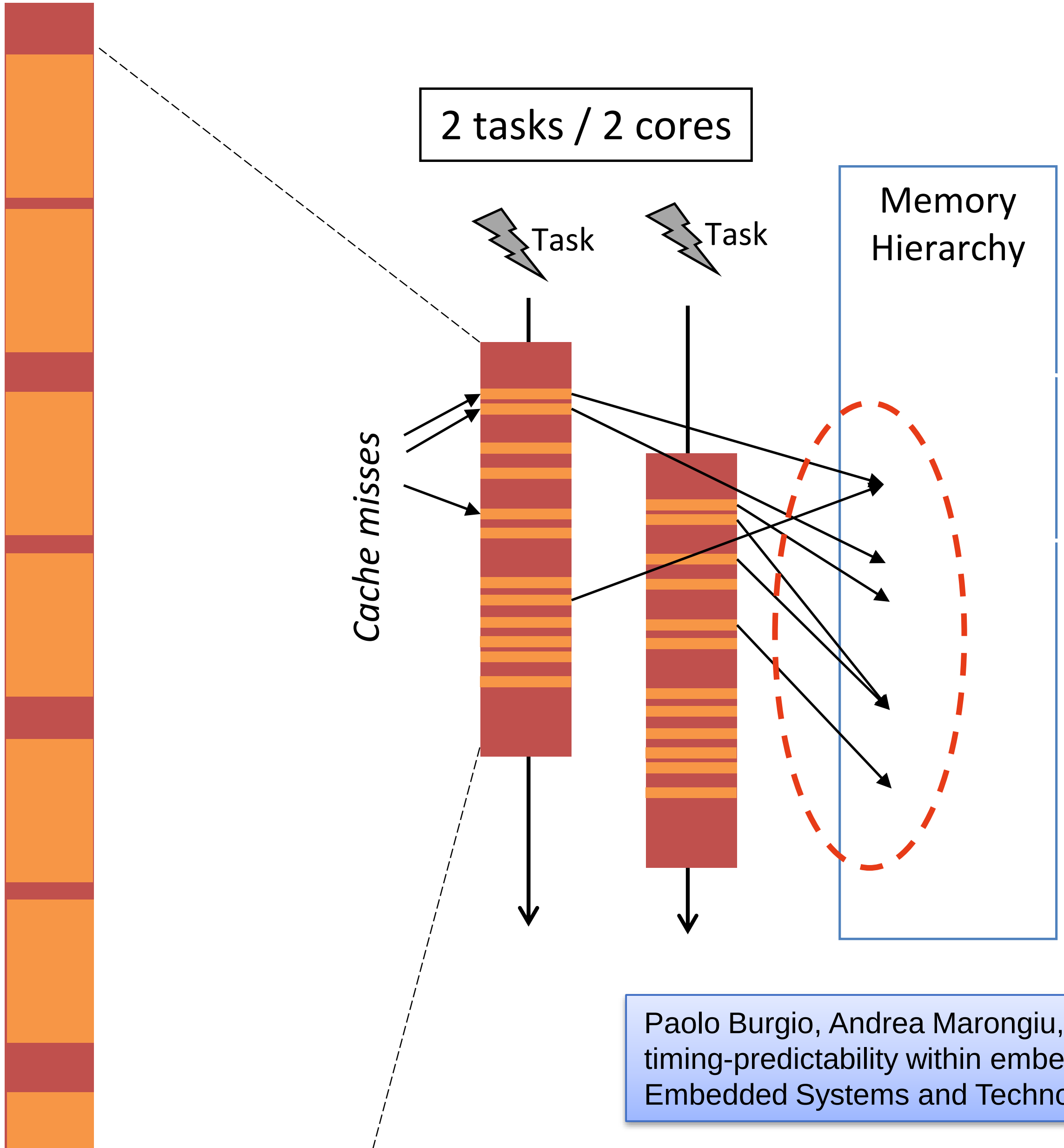


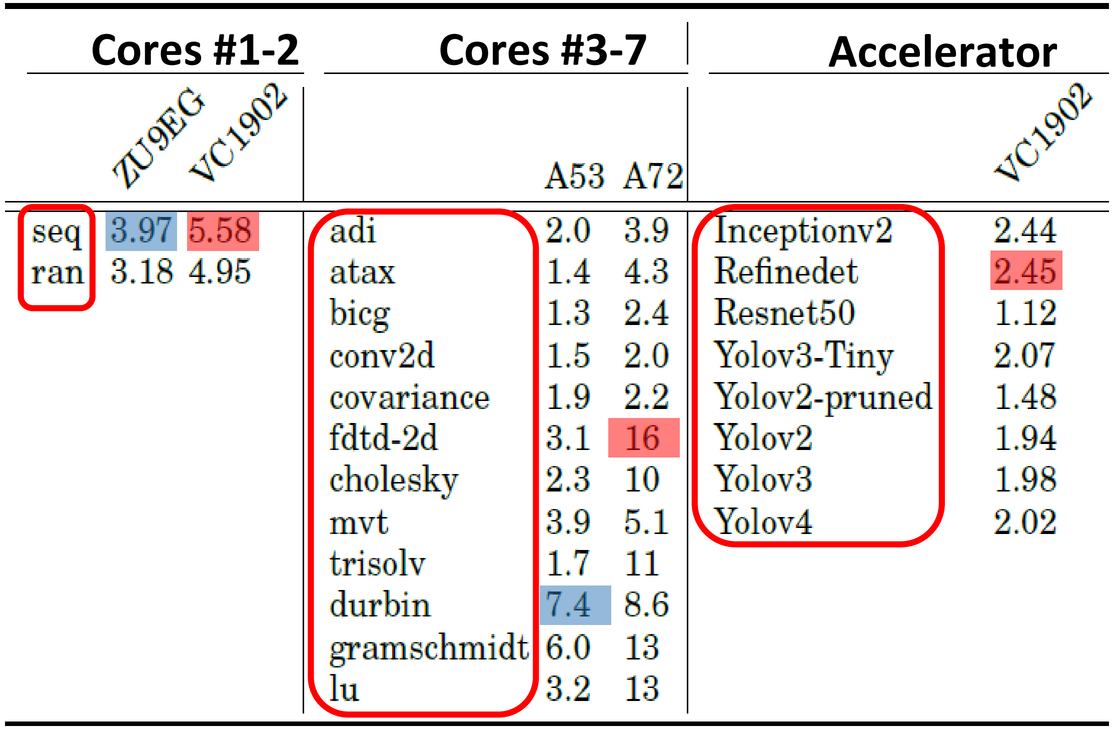
A lot of contention points!!



Paolo Burgio, Andrea Marongiu, Paolo Valente, Marko Bertogna, A memory-centric approach to enable timing-predictability within embedded many-core accelerators, 2015 CSI Symposium on Real-Time and Embedded Systems and Technologies (RTEST)

Uncontrolled memory accesses in multi cores



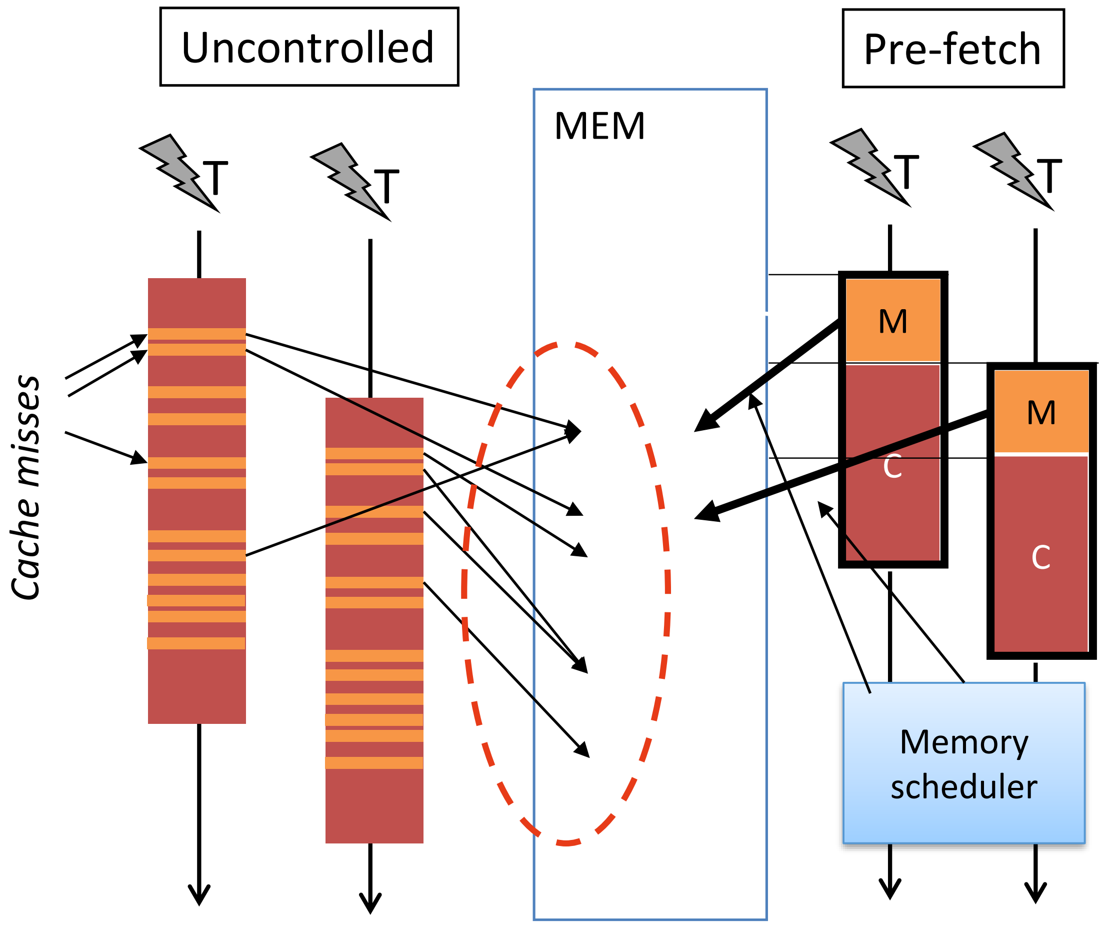


Cores #1-2			Cores #3-7			Accelerator	
	ZU9EG	VC1902		A53	A72		VC1902
seq	3.97	5.58	adi	2.0	3.9	Inceptionv2	2.44
ran	3.18	4.95	atax	1.4	4.3	Refinedet	2.45
			bicg	1.3	2.4	Resnet50	1.12
			conv2d	1.5	2.0	Yolov3-Tiny	2.07
			covariance	1.9	2.2	Yolov2-pruned	1.48
			fdtd-2d	3.1	16	Yolov2	1.94
			cholesky	2.3	10	Yolov3	1.98
			mvt	3.9	5.1	Yolov4	2.02
			trisolv	1.7	11		
			durbin	7.4	8.6		
			gramschmidt	6.0	13		
			lu	3.2	13		

Experiments show that, memory interference from accelerator logics can **slow down** task execution up to **16 times**, on host cores

Paolo Burgio, Andrea Marongiu, Paolo Valente, Marko Bertogna, A memory-centric approach to enable timing-predictability within embedded many-core accelerators, 2015 CSI Symposium on Real-Time and Embedded Systems and Technologies (RTEST)

Current SOTA: split task into Memory and Computation



State-of-the-art in RT systems

- Prefetch to local buffers (likewise we did for Particle Filters)
- Tasks run non-preemptively (**C**)
- Schedule also Memory (**M**)

In literature, this is known as PREM, LET, etc…

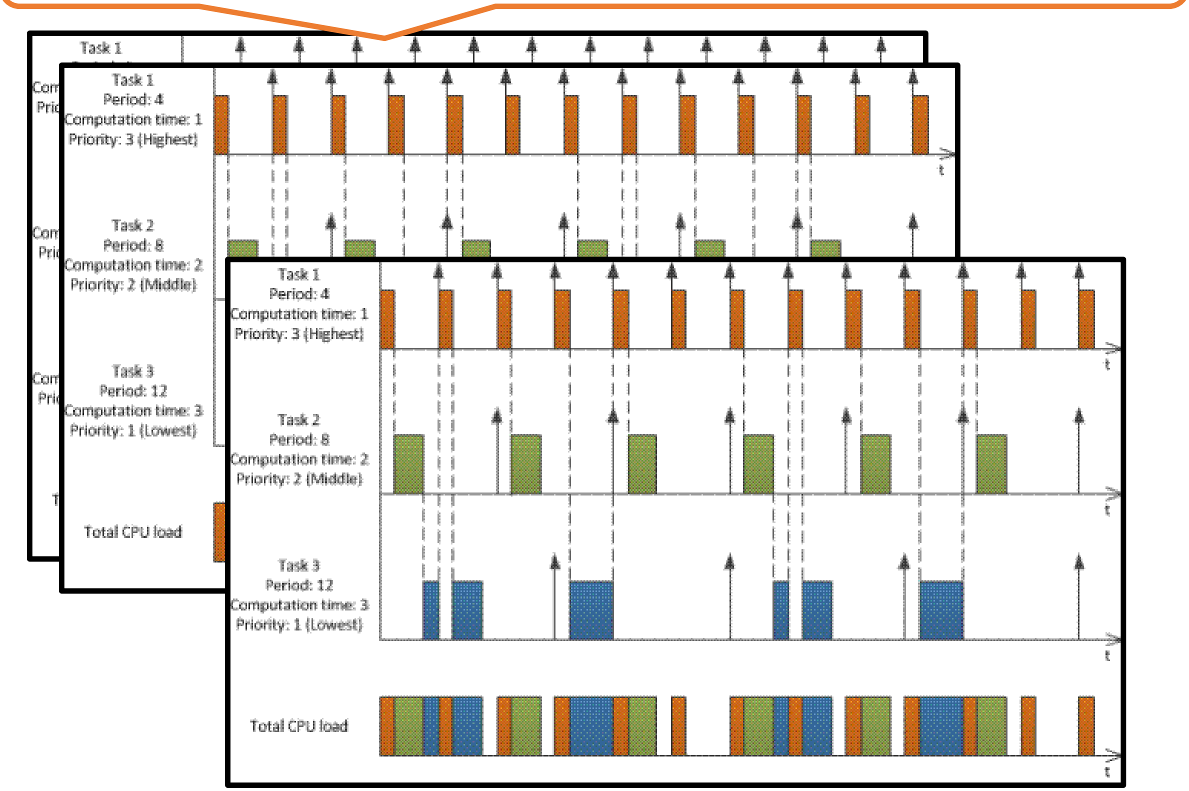
- This computation model makes the system more analyzable
- Researchers (including us) proved that the scheduling analysis produce more accurate Worst-Case Execution Time bounds
- This means that the system is more predictable system

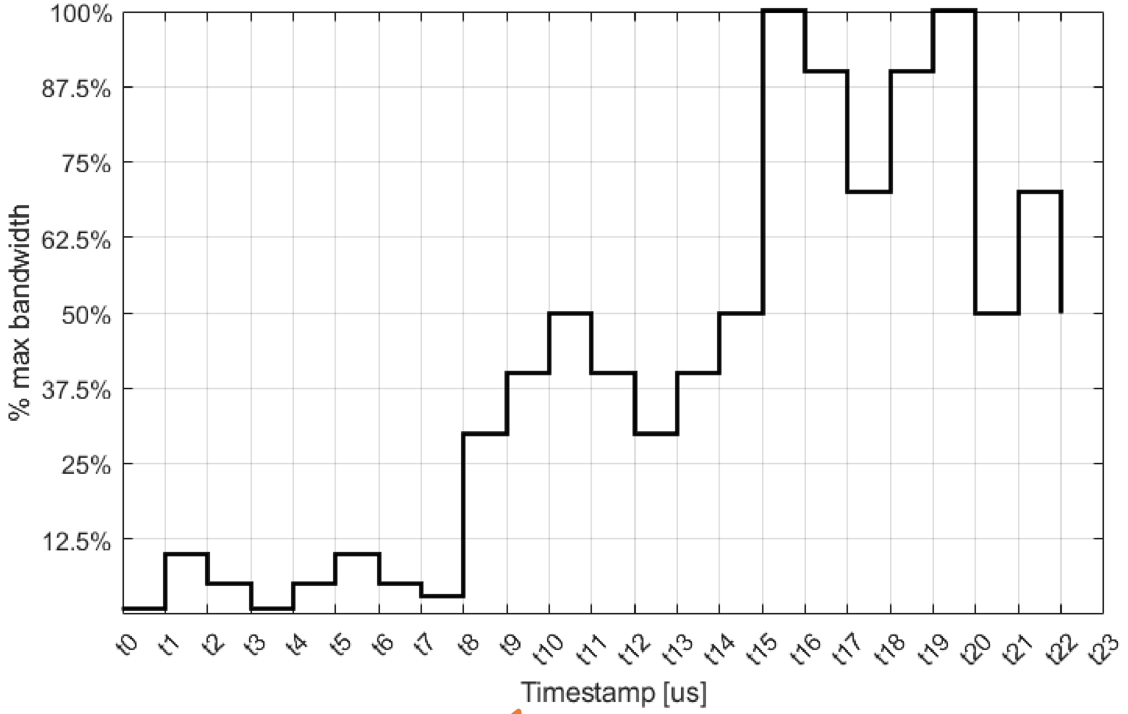
Fun fact: prefetching is known since decades in embedded the systems community!

Paolo Burgio, Andrea Marongiu, Paolo Valente, Marko Bertogna, A memory-centric approach to enable timing-predictability within embedded many-core accelerators, 2015 CSI Symposium on Real-Time and Embedded Systems and Technologies (RTEST)

Co-scheduling of memory and computation

“Traditional” task scheduling on multi-cores



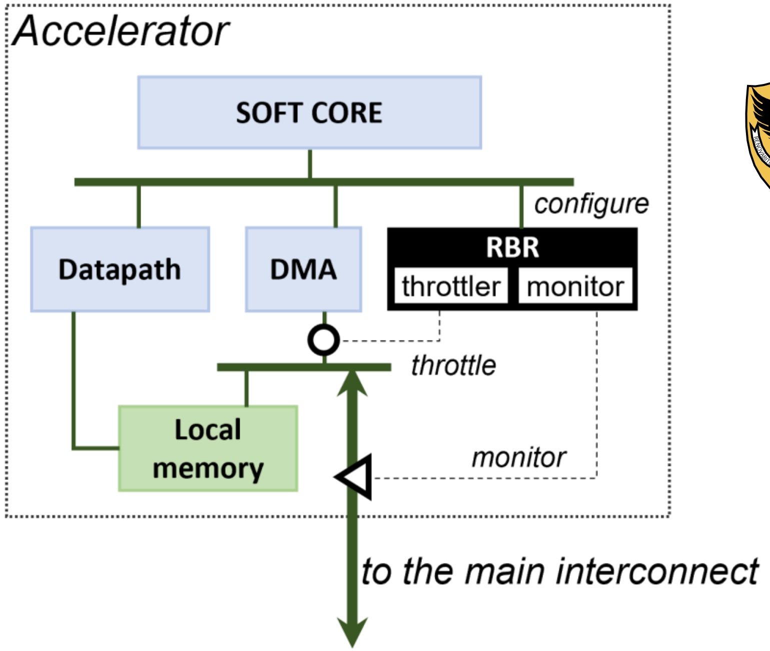


Memory access profile (bandwidth)

for each task, given by the memory schaduler

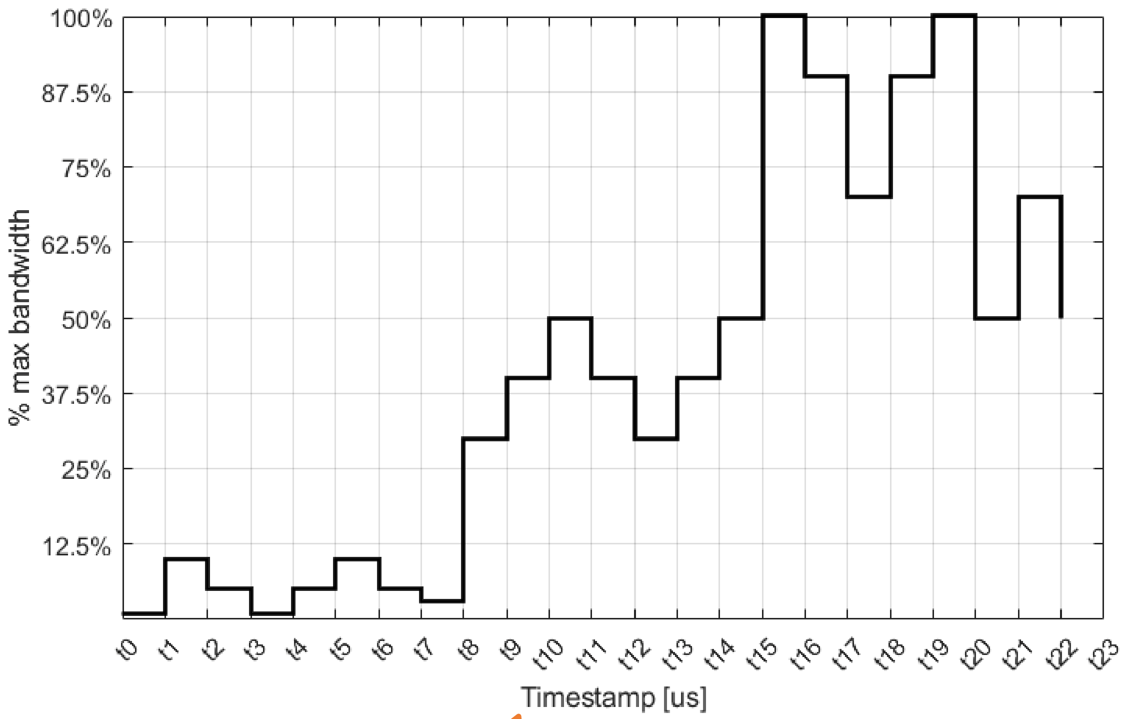
G. Brilli et al., "Fine-Grained QoS Control via Tightly-Coupled Bandwidth Monitoring and Regulation for FPGA-based Heterogeneous SoCs," 2023 60th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA

HW design for memory bandwidth monitoring and regulation



RBR: monitors the memory activity and regulates the bandwidth

- **Comparisong against other approaches**
  - **6x** faster than LCMT-RBR;
  - **10x** faster than LCMT-QoS400;
  - **114x** faster than LCMT-SW-DMA.



Memory access profile (bandwidth)

for each task, given by the memory schaduler

HW IP for memory bandwidth monitor and regulator

Given a memory bandwidth profile:

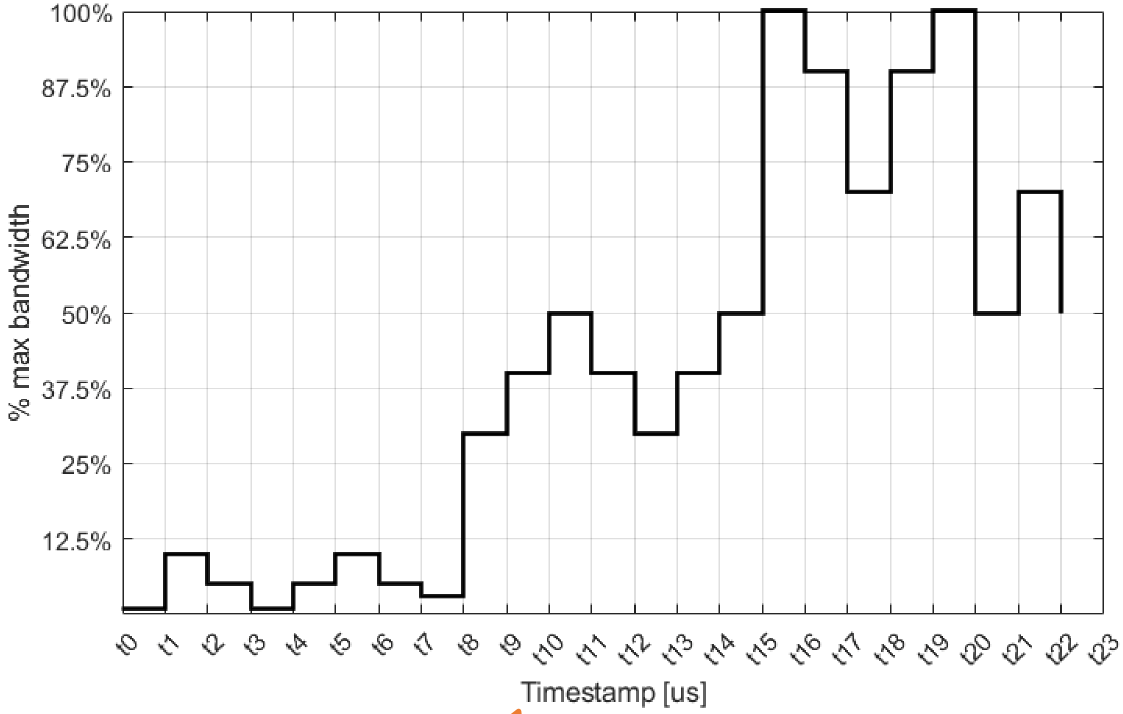
- Can schedule memory phases of ~30microsec
- Comparable to a context switch in modern OSses

Can be effectively adopted

in real settings

RBR: monitors the memory activity and regulates the bandwidth

- <u>Comparisong against other approaches</u>
  - 6x faster than LCMT-RBR;
  - 10x faster than LCMT-QoS400;
  - 114x faster than LCMT-SW-DMA.



Memory access profile (bandwidth) for each task, given by the memory schaduler

Wrapping up and plans

Chips JU

HAL4SDV

(no logo yet, just got funded)

AI_4CSM

SF SHAPE FUTURE

Scaling up, out, and industrialization

✓Research funded by EU projects, and industry

✓With our two startups…or other partners

Current (TRL 4) prototypes: NVIDIA GPGPUs, AMD/Xilinx, RISC-V

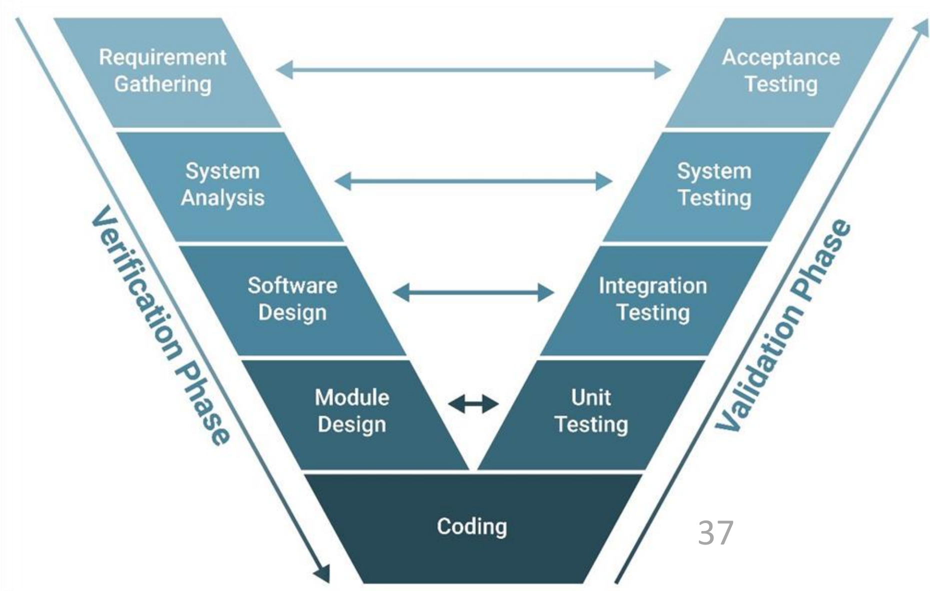
✓Design a common methodology/framework for any HW

An integrated approach compliant with the V-model

✓System design (for reconfigurable platforms)

✓Integration within programming model

✓Extending our custom RBR module to
enable predictable core-to-accelerator interaction



Prof. Ing. Paolo Burgio, PhD

paolo.burgio@unimore.it

https://hipert.unimore.it
https://hipert.it

Paolo Burgio

Scan the QR code to add me as friend

HIPERT

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

High Performance
Real Time Lab