

# Automotive embedded software architecture in the multi-core age

**Paolo Gai**

Evidence

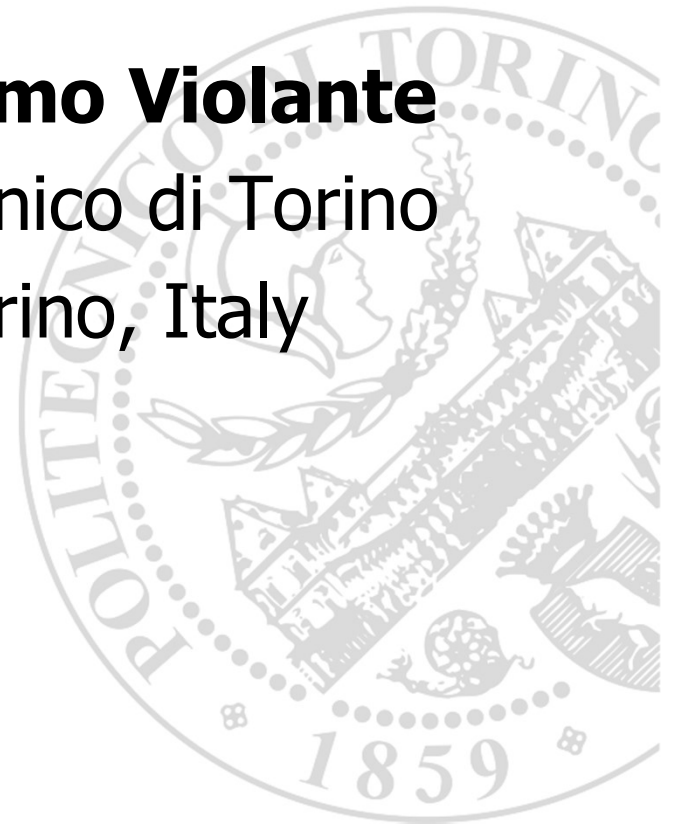
Pisa, Italy



**Massimo Violante**

Politecnico di Torino

Torino, Italy



# The big picture



**Space Shuttle**  
~500.000 LOCs



**High-end vehicle**  
~100 Millions LOCs

<https://www.wired.com/2012/12/automotive-os-war/>



**Boeing 777**  
~3 Millions LOCs

“The car of the future will be the most powerful computer you will ever own”

**The Telegraph**

# The big picture

>70% of all new customer features are ICT-enabled and distributed in nature

## Inter-vehicle ICT

Vehicle-to-vehicle  
Vehicle-to-infrastructure

## In-vehicle ICT

Autonomous driving

Infotainment  
Telematics

ECU for P/T, ABS, ...  
Networking (CAN, MOST, ...)  
ADAS

No ICT



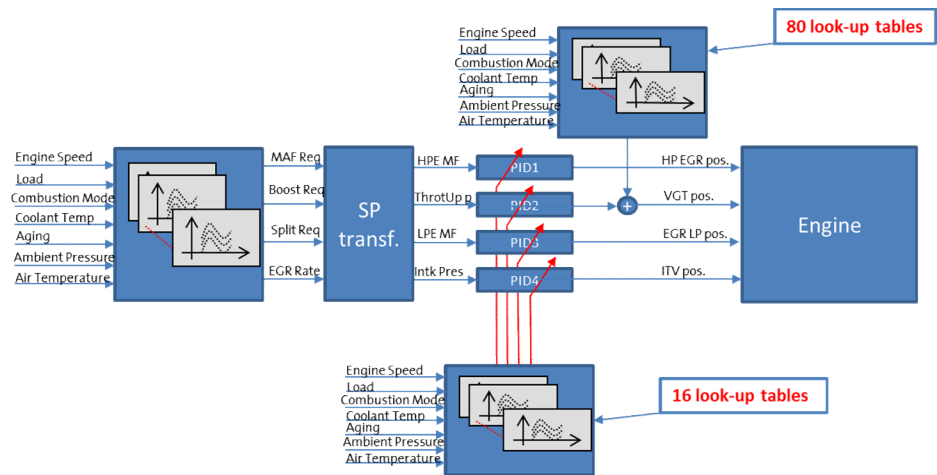
# The big picture

- **New features** are challenging from the computing point of view
  - Pedestrian detection
  - Autonomous emergency braking
  - Autonomous driving
  - ...
- But also evolved **traditional features** are computing demanding
  - Powertrain
  - Braking

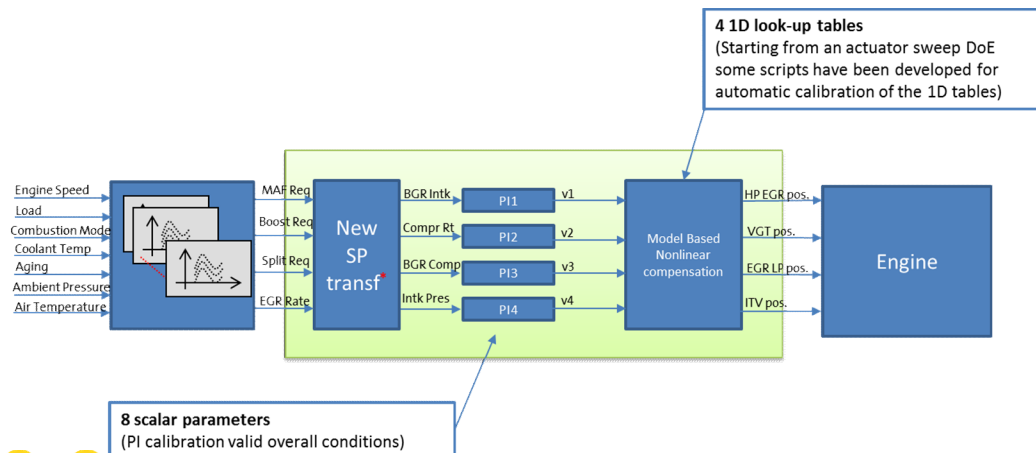




# An example taken from powertrain



**Today:** huge calibration effort needed to define LUT content. "Simple" control strategy with little processing and many LUT accesses.



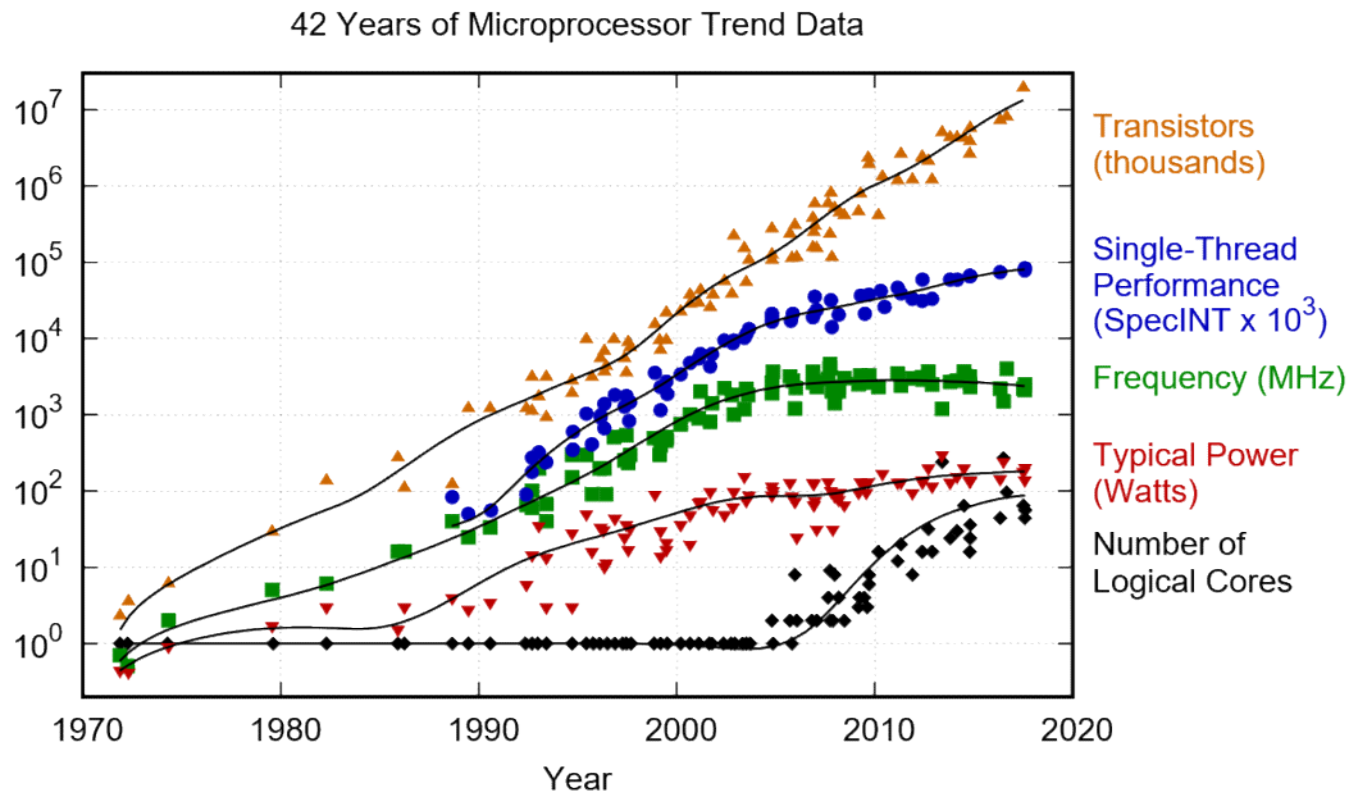
**Tomorrow:** little calibration effort. "Complex" control strategy with significant processing → combustion model run in **real-time!**

# Designer wish list

- **Computing capabilities** for demanding vehicle functions
- **Platforms** for consolidating multiple applications on a single device
- **Effort-free safety** for critical vehicle functions
- Solutions to maximize **reusability** and **scalability**

# Designer wish list

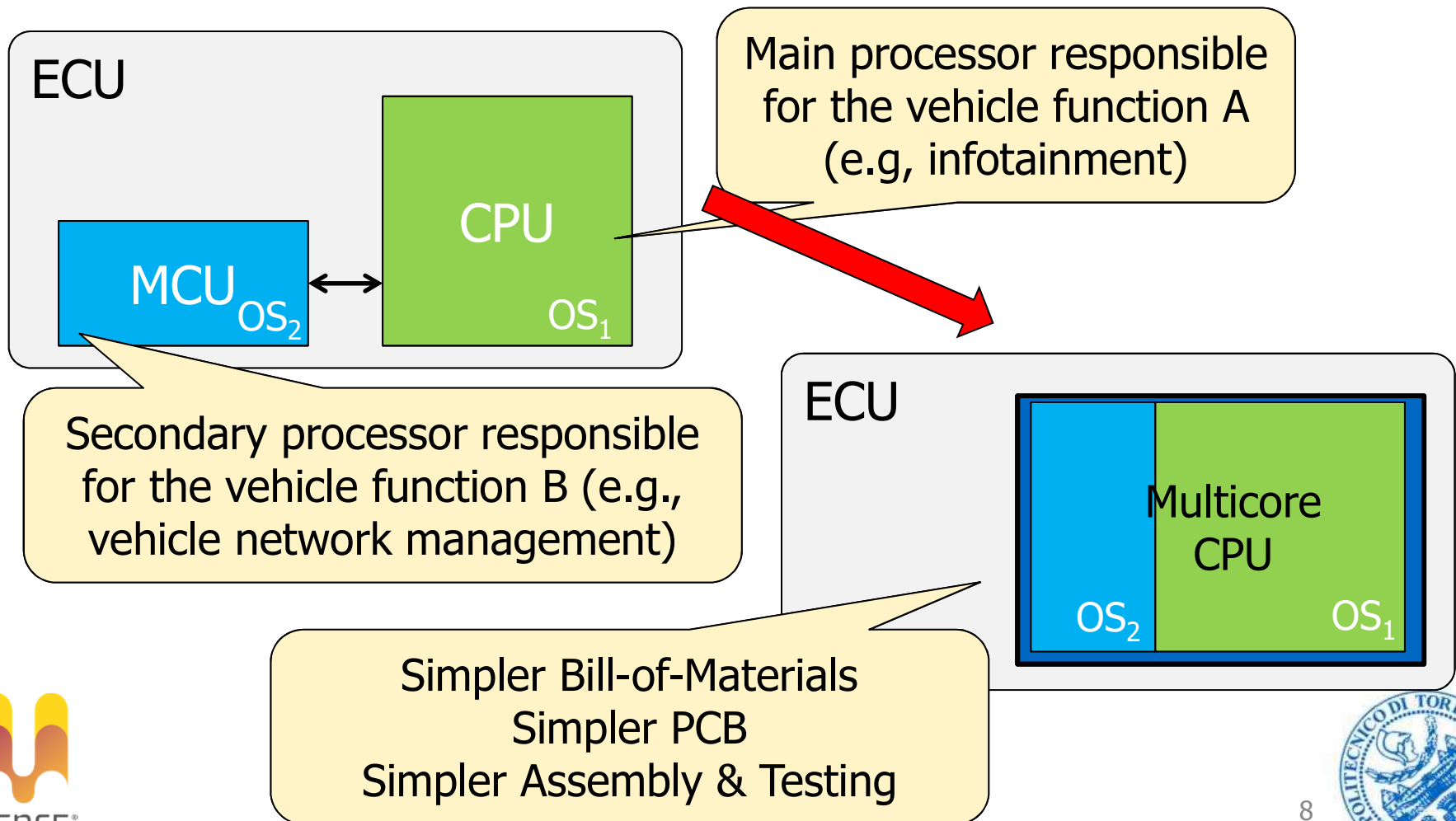
- Multi-core is most likely the computing platform of choice



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten  
New plot and data collected for 2010-2017 by K. Rupp

# A reference use-case for the tutorial

- Typical Electronic Control Unit (ECU)



# The enabling technologies

- Multi-core architectures
  - Computing can be distributed among several cores
  - Same chip can accommodate multiple functions
- AUTOSAR
  - Functions can be made independent from the execution platform
- Virtualization
  - Function can be segregated to dedicated resources to avoid interferences

# Outline

- Multicore architectures
- AUTOSAR
- Virtualization
- A use case
- Conclusions

# Outline

- Multicore architectures
- AUTOSAR
- Virtualization
- A use case
- Conclusions



# Multicore architectures (1)

Embedded systems used in automotive changed over the years:

- 1985 – Isolated embedded architectures
- 1995 – Distributed architectures over CAN bus
- 2005 – Integrated architectures based on AUTOSAR
- 2015 – Distributed architectures based on  
Multicore AUTOSAR + Infotainment solutions
- 2025 – Zonal architectures

# Multicore architectures (2)

- Nowadays we can identify three classes of processors in the automotive market:
  - System-On-Chip for control applications
  - Microprocessors for graphical applications
  - High-performance chips for ADAS applications

# SoC for control applications (1)

- Low end microcontrollers, up to complex multicores
- Static workloads (often based on OSEK/AUTOSAR)
- Mostly single cores, multicores
  - for high performance applications
  - for integrating more applications in the same chip

# SoC for control applications (2)

- Heterogeneous → two cores with “similar” ISA
  - NXP MPC5668G Fado hosting a PPC z6 and a PPC z0
  - 2<sup>nd</sup> CPU dedicated to different subsystems (peripherals)
- Uniform memory address space across cores
- Safety → lockstep configuration
  - As an example, NXP MPC5643L Leopard, AURIX Tricore
- RISC + MCU + DSP
  - As an example, Infineon Tricore
- No external RAM and Flash

Extended debug support with ETM macrocells

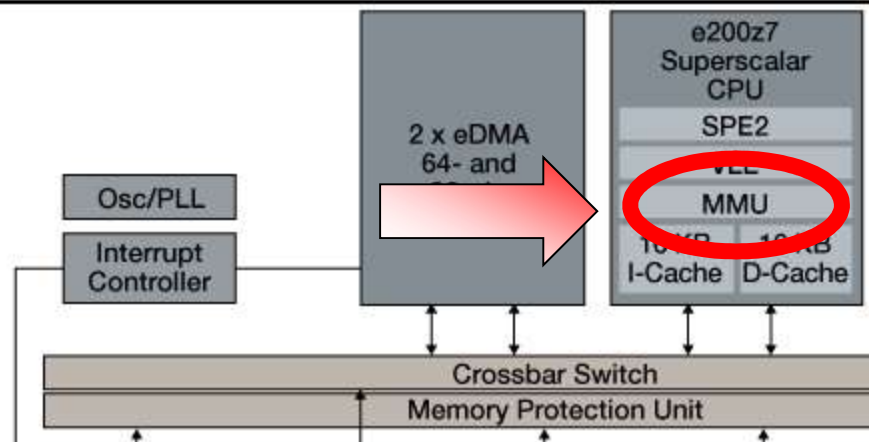
# SoC for control applications (3)

- High number of peripherals
  - often devoted to control and timing (ADC, PWM, Encoders, Timers, ...)
  - lack of traditional desktop interfaces (no USB / Video / ...)
  - communication buses CAN/FlexRay/LIN, recently Ethernet.
- Complex co-processors (PowerPC eTPU , Bosch GTM)
  - used to perform complex real-time-related features
- Cryptography subsystems isolated from main CPU
  - Infineon Tricore HSM
- Scratchpad memories

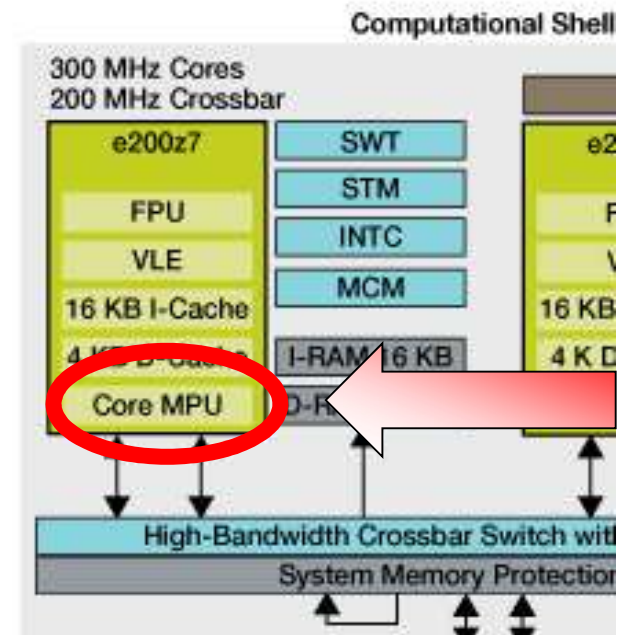
# MPU, not MMU

- Trend towards MPU, no MMU
  - NXP MPC5674F Mamba has a MMU
  - NXP MPC5777M Matterhorn has only the MPU

MPC5674F Block Diagram

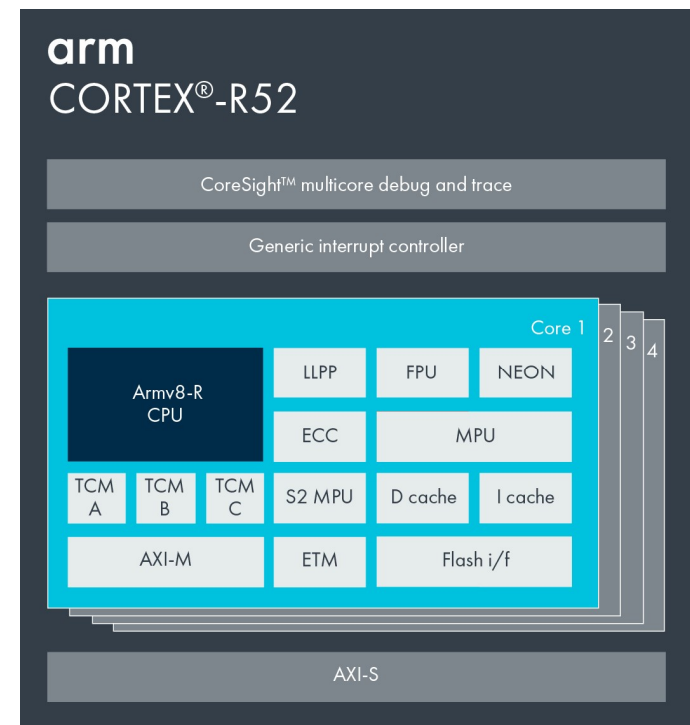


MPC5777M Block Diagram



# Hypervisor extensions with MPU

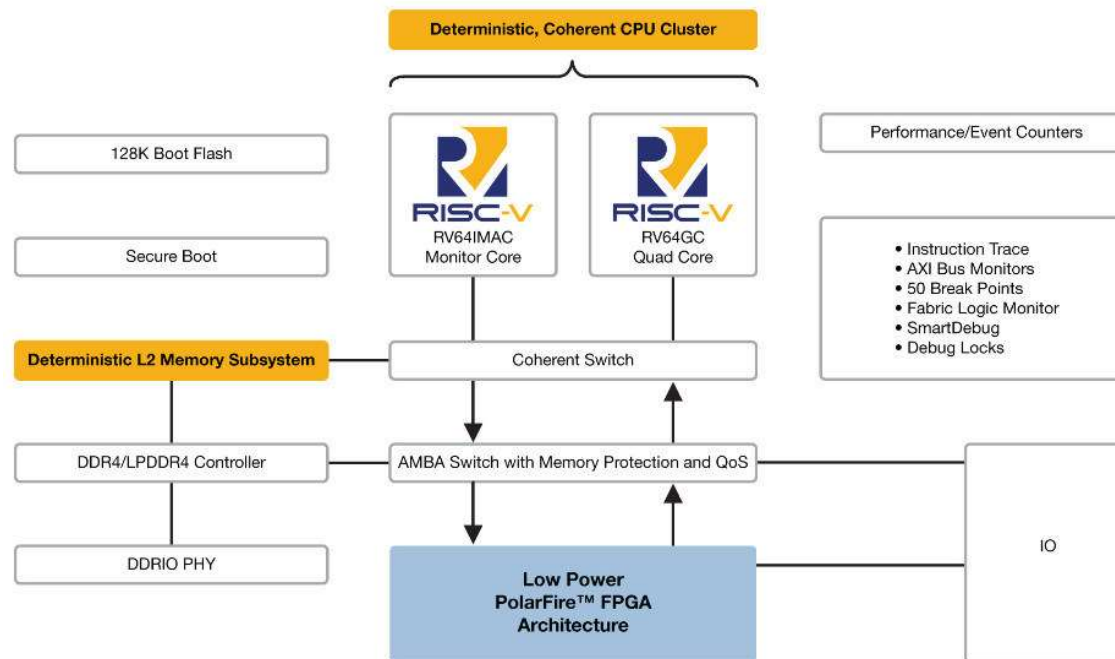
- Example: Cortex R52
- Hypervisor extensions coupled with MPU
- Multicore and Safety together
- Use cases:
  - Integration of different legacy applications
  - Safety/Security VMs





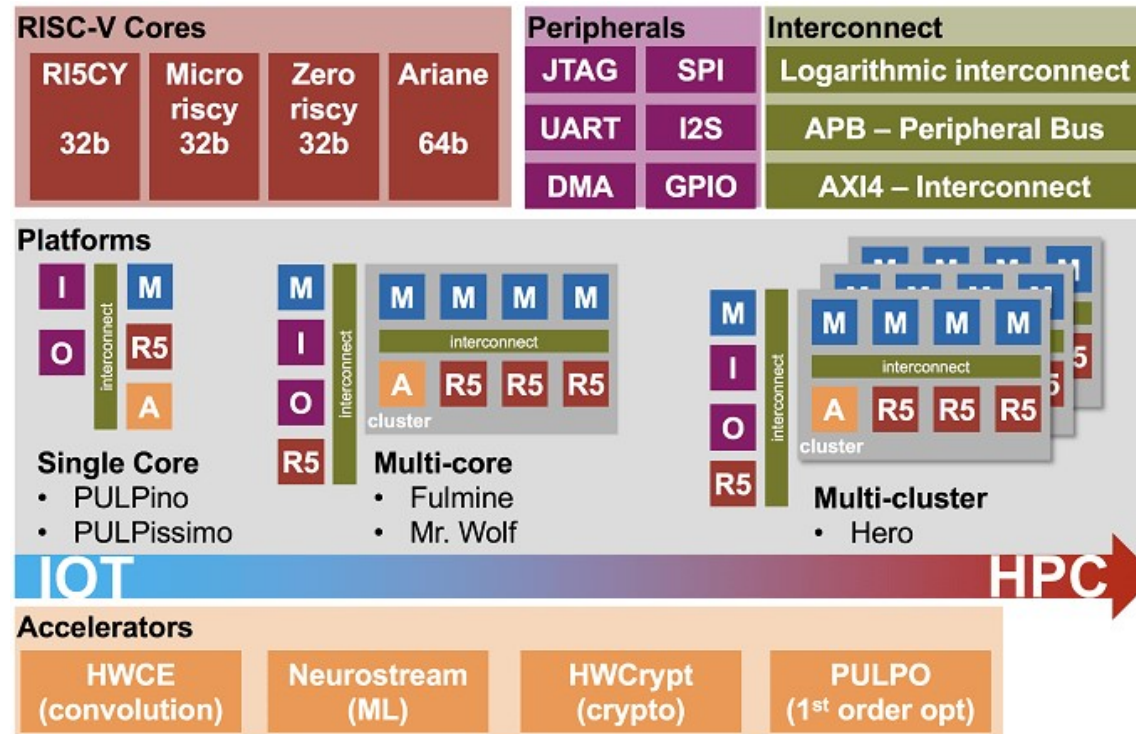
# RISC V on the rise

- Open source architecture
- Multicore, Hypervisor support



# RISC V on the rise (2)

- Open source architecture
- Multicore, Hypervisor support



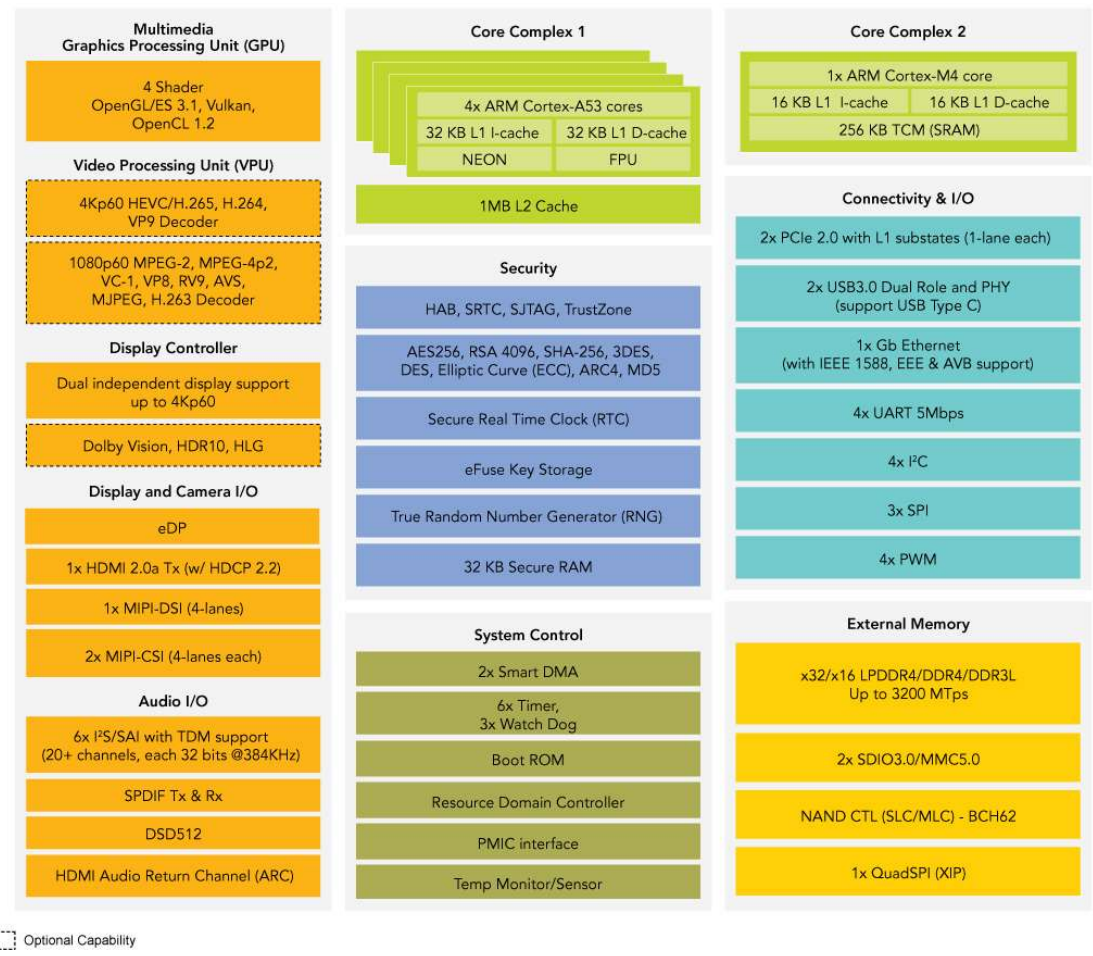
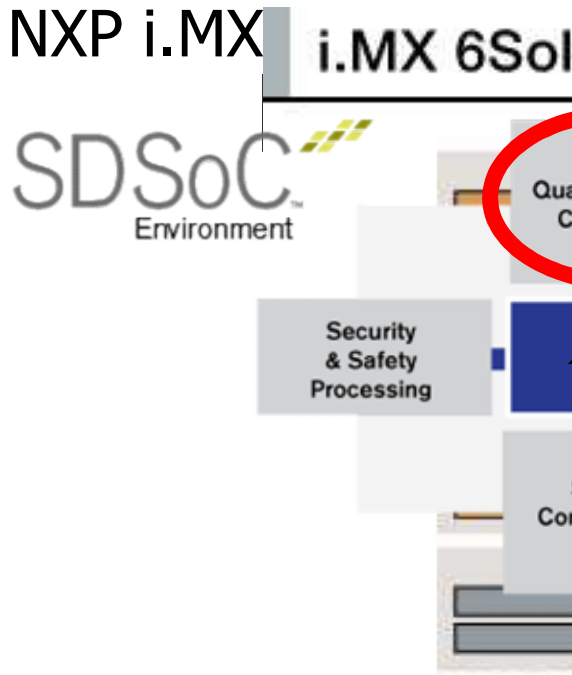
# Microprocessors for graphical apps

- Typically for infotainment subsystems
    - general purpose microprocessors Intel/ARM/MIPS cores
    - In the past hosted commercial operating systems VxWorks, QNX, Windows CE
    - Support for USB/Ethernet/Wifi/...
  - Evolution towards low-power consumption
    - migration to multi-cores
    - video acceleration (video decoders, GPU for OpenGL)
  - Evolution trend
    - move to Linux (see Genivi/Tizen, now dead)
- hypervisors (INTEGRITY, PikeOS, Xen, JailHouse)
- Security and Safety with the hypervisors

# Microprocessors and small cores

- Integration of small microcontrollers (Cortex M/R) for hard real-time / safety features

- TI Jacinto
- NXP SoloX
- Xilinx Ultrascale+
- NXP i.MX



# High-performance processors (1)

- Used for ADAS Applications
  - Video processing / Image recognition
  - Intel / 64 bit ARM / VLIW
  - Multi / Manycore
  - Programmable GPUs
  - Hypervisor extensions
  - Ethernet in place of slower Buses

# High-performance processors (2)

- Nvidia X1/Parker/Xavier/Orin

| NVIDIA ARM SoCs       |                           |  |  |
|-----------------------|---------------------------|--|--|
|                       | Xavier                    | Parker                                     | Erista (Tegra X1)                          |
| CPU                   | 8x NVIDIA Custom ARM      | 2x NVIDIA Denver +<br>4x ARM Cortex-A57    | 4x ARM Cortex-A57 +<br>4x ARM Cortex-A53   |
| GPU                   | Volta, 512 CUDA Cores     | Pascal, 256 CUDA Cores                     | Maxwell, 256 CUDA Cores                    |
| Memory                | ?                         | LPDDR4, 128-bit Bus                        | LPDDR3, 64-bit Bus                         |
| Video Processing      | 7680x4320 Encode & Decode | 3840x2160p60 Decode<br>3840x2160p60 Encode | 3840x2160p60 Decode<br>3840x2160p30 Encode |
| Transistors           | 7B                        | ?  | ?  |
| Manufacturing Process | TSMC 16nm FinFET+         | TSMC 16nm FinFET+                          | TSMC 20nm Planar                           |



# Nvidia Volta

## NVIDIA TESLA V100 SPECIFICATIONS

7.5 TeraFLOPS (double)

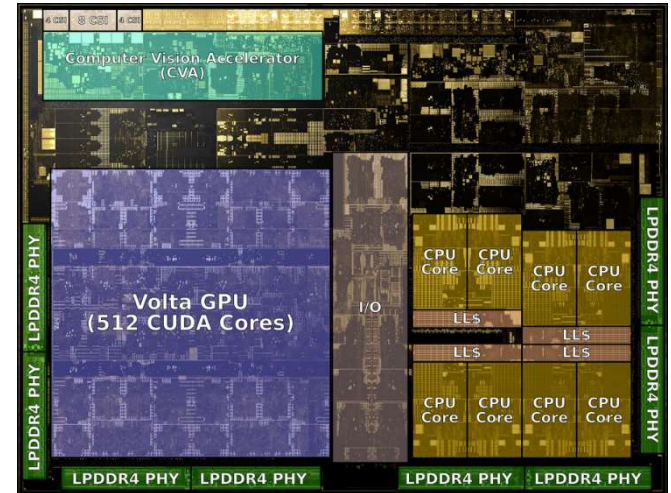
15 TeraFLOPS (single)

120 TeraFLOPS (deep learning)

300 GB/s (NVLINK)

900 GB/s (memory bandwidth)

300 WATTS (power consumption)



## 3X Faster on Deep Learning Training



CPU Server: Dual Xeon E5-2699 v4, 2.6GHz | GPU Servers add 8X Tesla K80, Tesla P100 or Tesla V100 | V100 measured on pre-production hardware | Workload: NMT, 13 epochs to solution.



# High-performance processors (2)

## ■ Kalray

- 256 VLIW
- There

VLIW




Instruct  
Parallel





Perception and Modeling

Safe Path Planning



### Perception & Modeling

#### Kalray

- MPPA high performance AI accelerator: Bostan & Coolidge portfolio up to 100 Tops
- Baidu Apollo 3.0 perception software



### Safe Path Planning

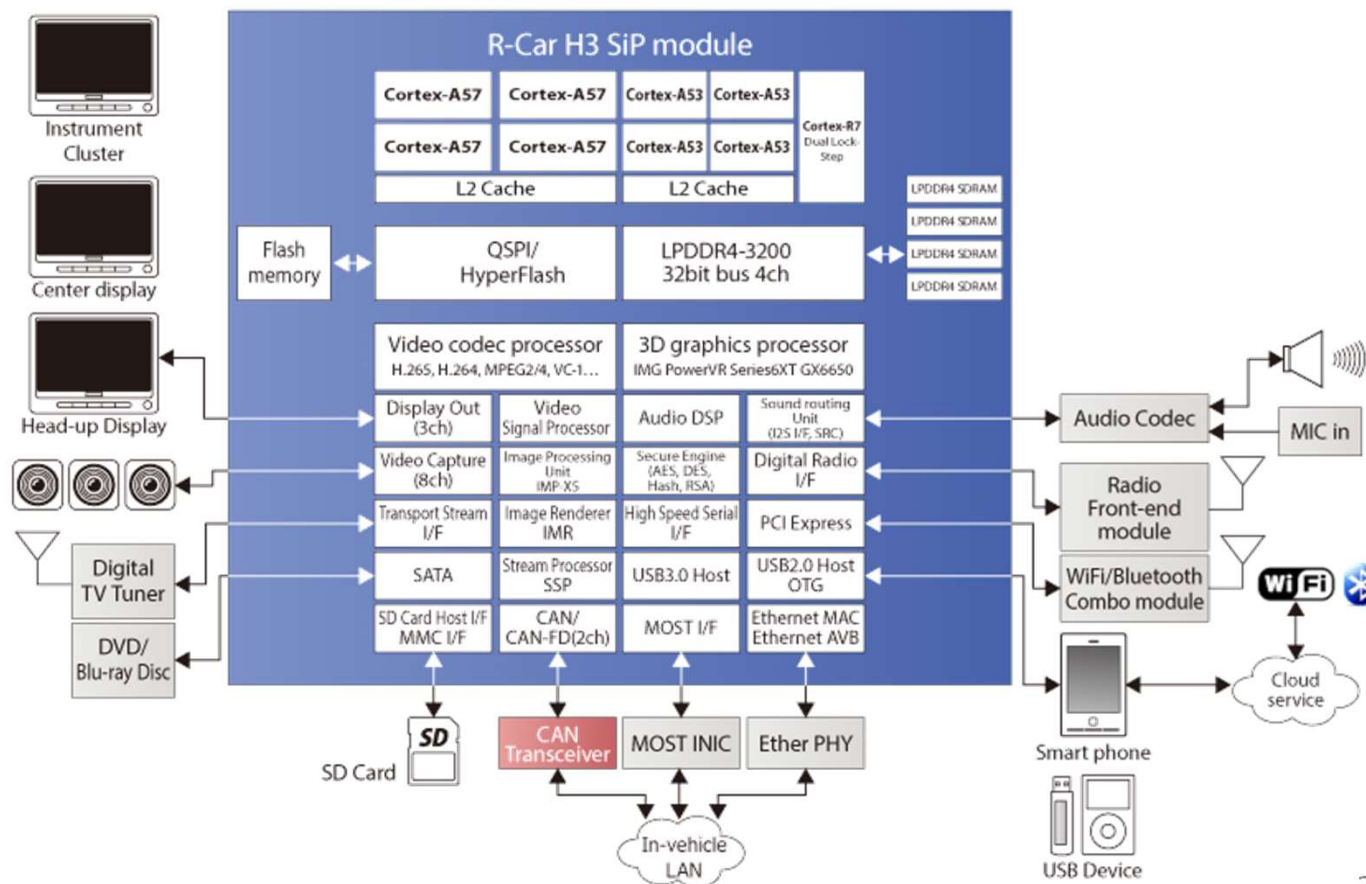
#### BlueBox 2.0



- Safe embedded AI Processor: S32V2
- High performance embedded processor: LS2084A
- Baidu Apollo 3.0 path planning software

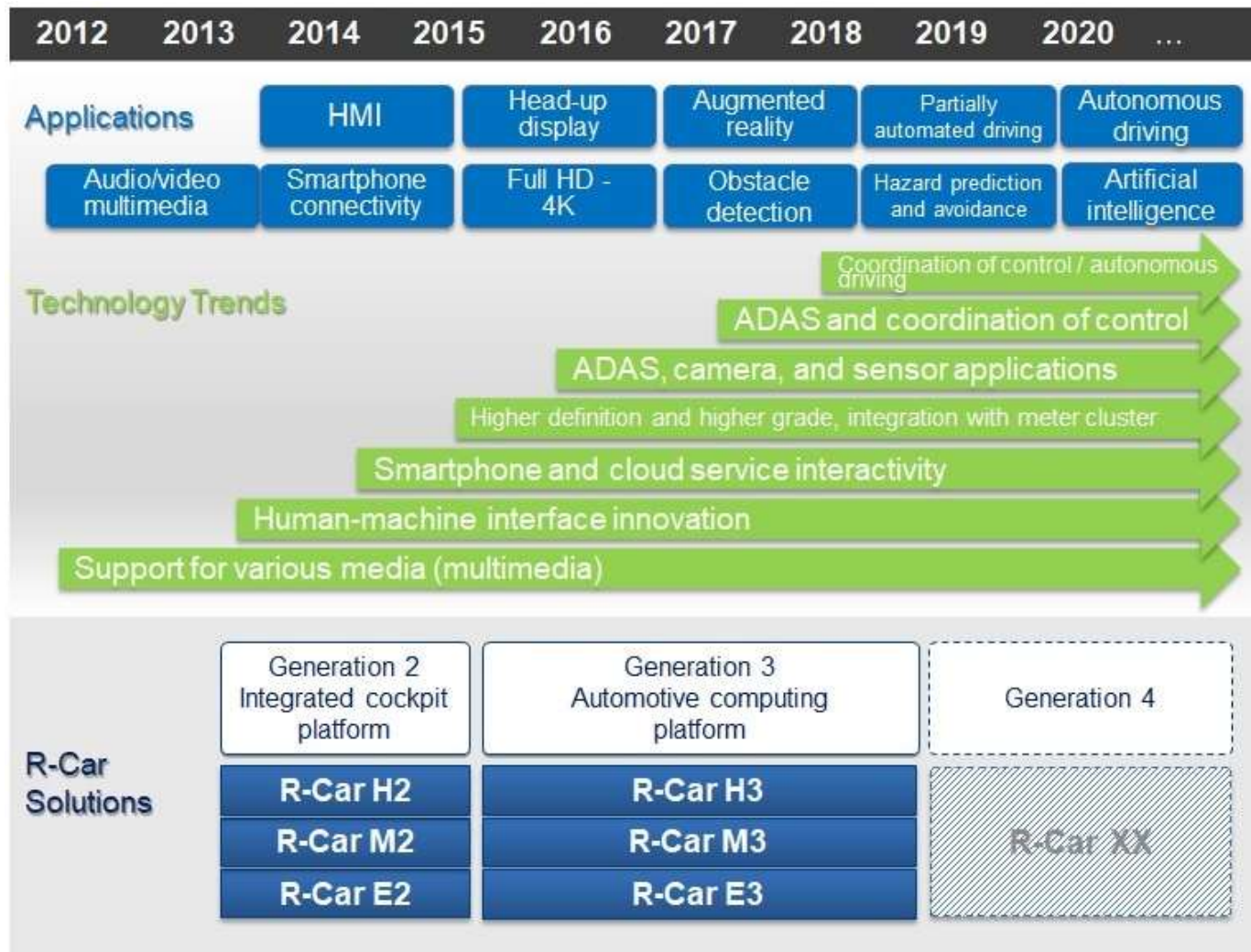
# High-performance processors (2)

- Renesas RCAR H3
  - ARM64 Big-Little + Cortex R + Graphics processing



# High-performance processors (3)

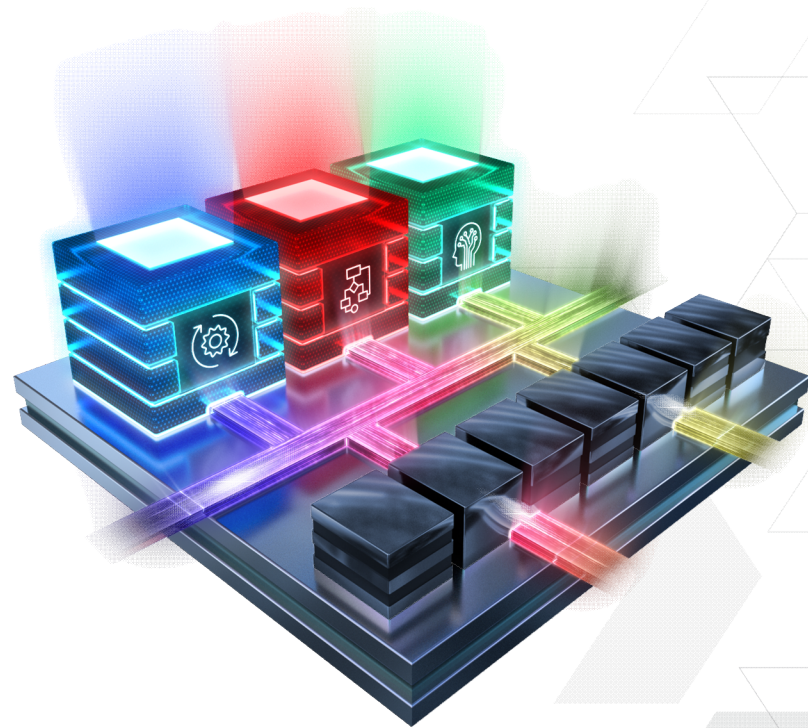
## ■ Renesas RCAR H3



# Introducing the World's First ACAP



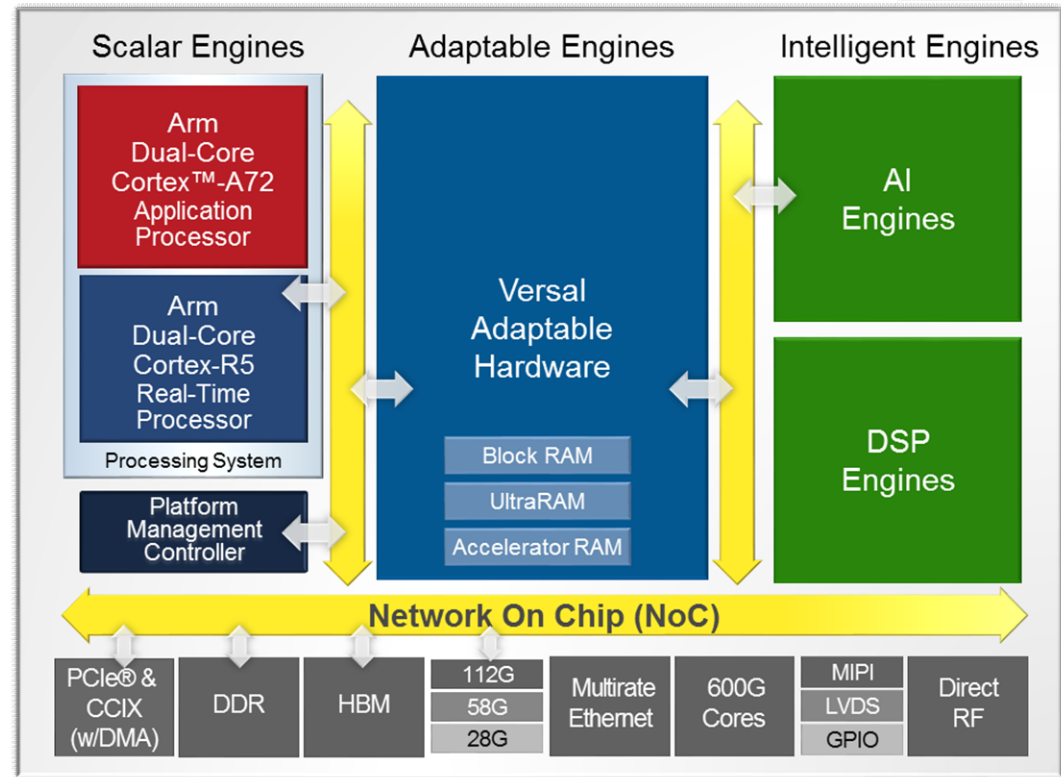
- > Heterogeneous Acceleration
- > For Any Application
- > For Any Developer





# Adaptable Architecture Connected Via NoC

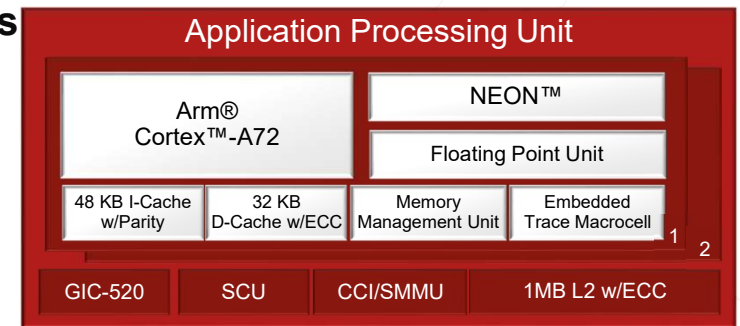
- > **Scalar Engines**
  - >> Arm® Cortex™-A72 APU
  - >> Arm Cortex-R5 RPU
- > **Adaptable Engines**
  - >> CLBs
  - >> Internal Memory
- > **Intelligent Engines**
  - >> AI Engine
  - >> DSP Engine
- > **Connectivity**
  - >> PCIe w/CCIX
  - >> Ethernet
  - >> DDR Memory Controllers
  - >> Transceivers
  - >> I/O
- > **Platform Resources**
  - >> Network-On-Chip
  - >> Platform Management Controller



# Scalar Engines in the Arm Processing System

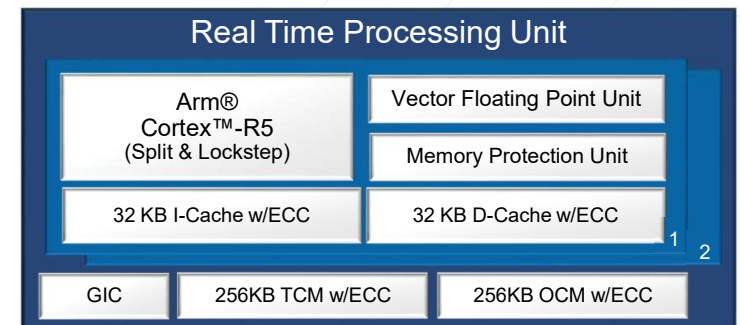
## > Dual-Core Arm® Cortex™-A72 Application Processors

- >> Up to 1.7GHz for 2X single-threaded performance<sup>1</sup>
- >> Cost and power optimized (half the power)
- >> Code compatibility (ARMv8-A architecture)
- >> Device boots without a bit stream



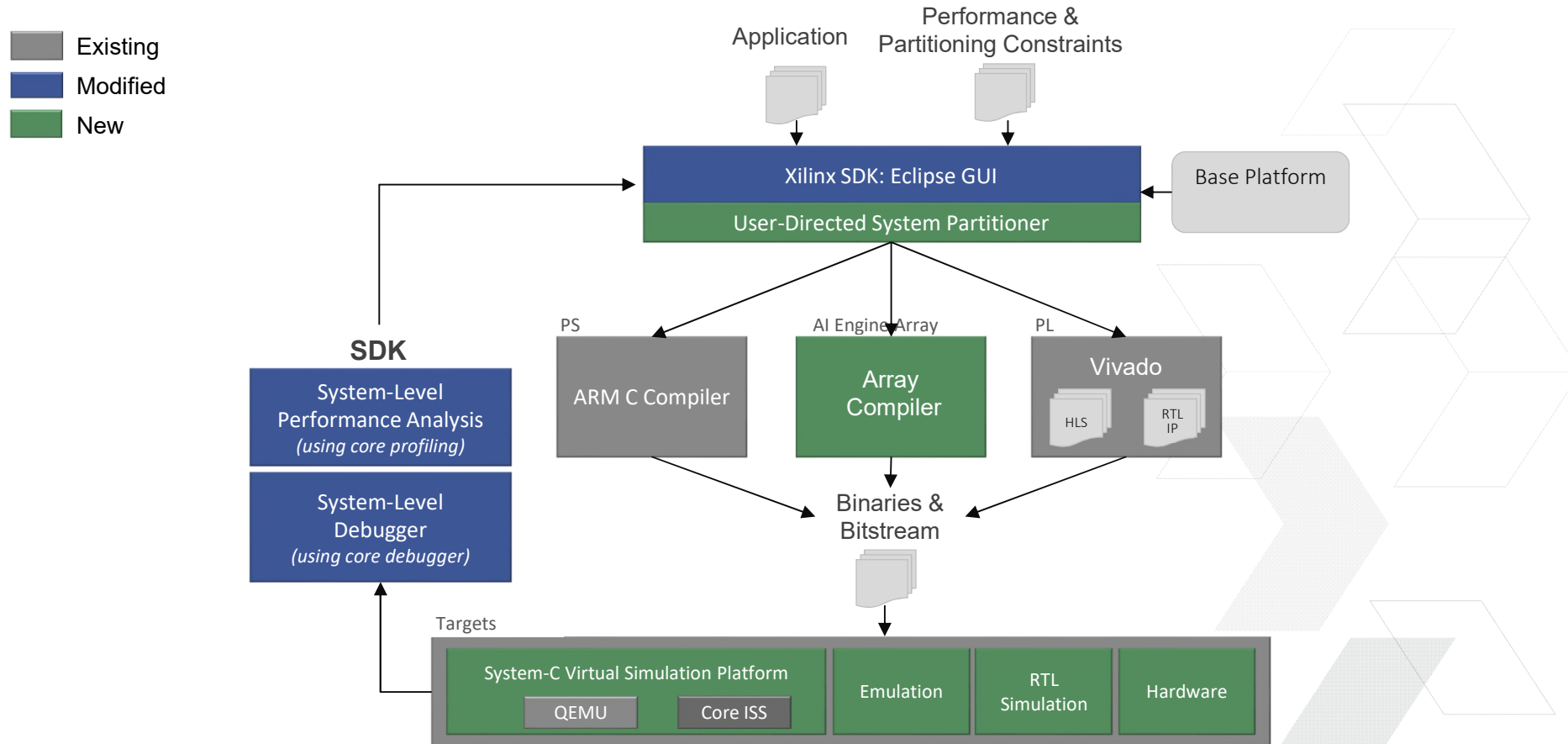
## > Dual-Core Arm® Cortex™-R5 Real Processors

- >> Up to 750MHz for 1.4X greater performance<sup>1</sup>
- >> Low latency and deterministic
- >> Flexible operation modes: Split-Mode and Lock-Step
- >> Highest levels of functional safety (ASIL and SIL)



1: DMIPS vs. Zynq UltraScale+ MPSoCs

# Unified Tool Chain for Device Programming





# Issues when integrating multicores

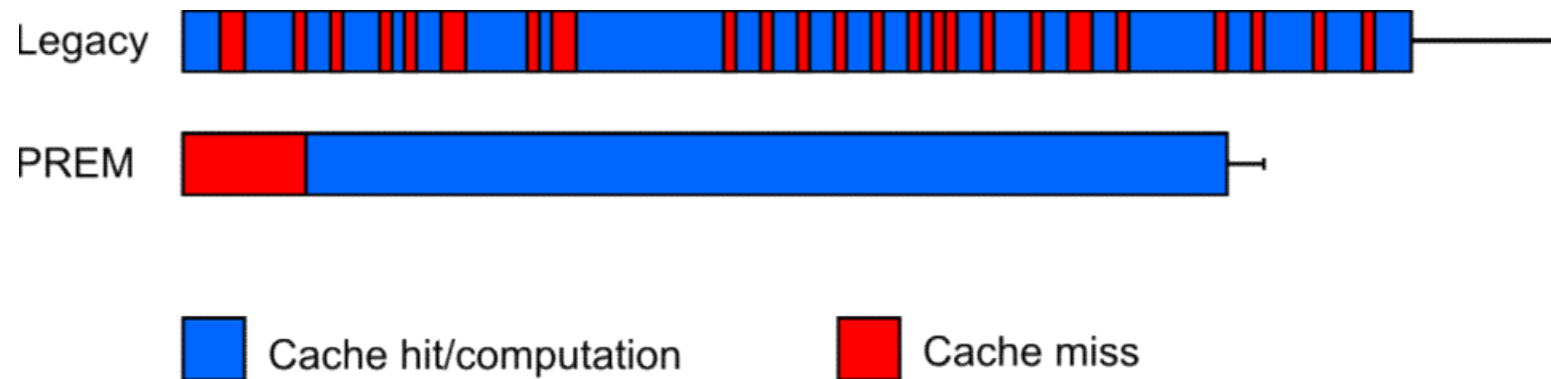
- 2 categories of automotive applications
  - Real-Time applications (powertrain, body and chassis)
  - Non- real-time (infotainment)
- A one-to-one mapping exists between application (either real- or non-real-time) and processor
- With multi-cores, different applications will coexist
  - Temporal Interference

# Temporal interference

- The execution time of a task varies depending on the interference received from other tasks in the same chip
  - Caches, DRAM memories, scratchpad memories
  - Parallel usage of the same resource by different actors
- Temporal interference makes the partitioning of the application functionalities into cores very difficult
  - 20-30% overhead when moving to dual cores!
- PREM techniques to limit interference
  - Requires changes in the application source code

# PREM and scratchpad memories

- Example - PREM technique implemented using Scratchpad memories

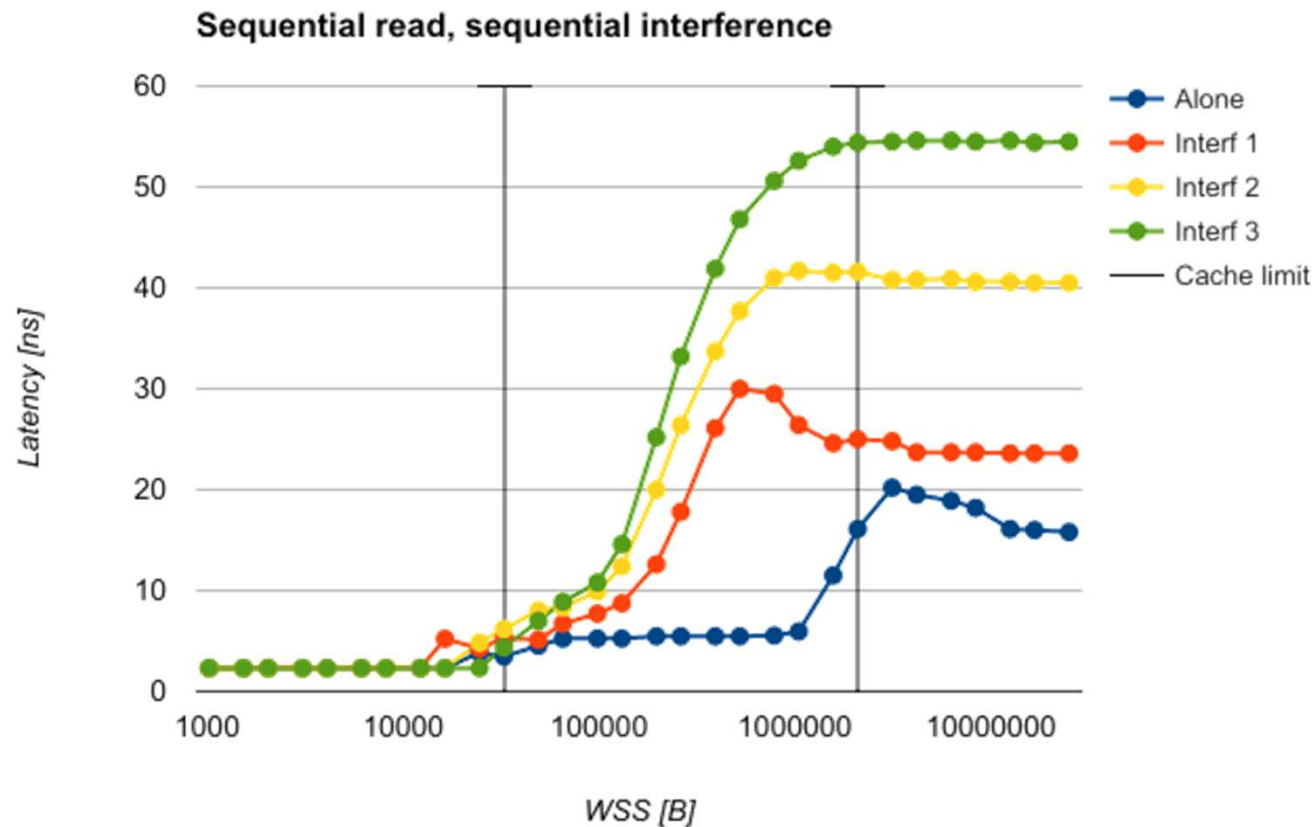


## Predictable interval

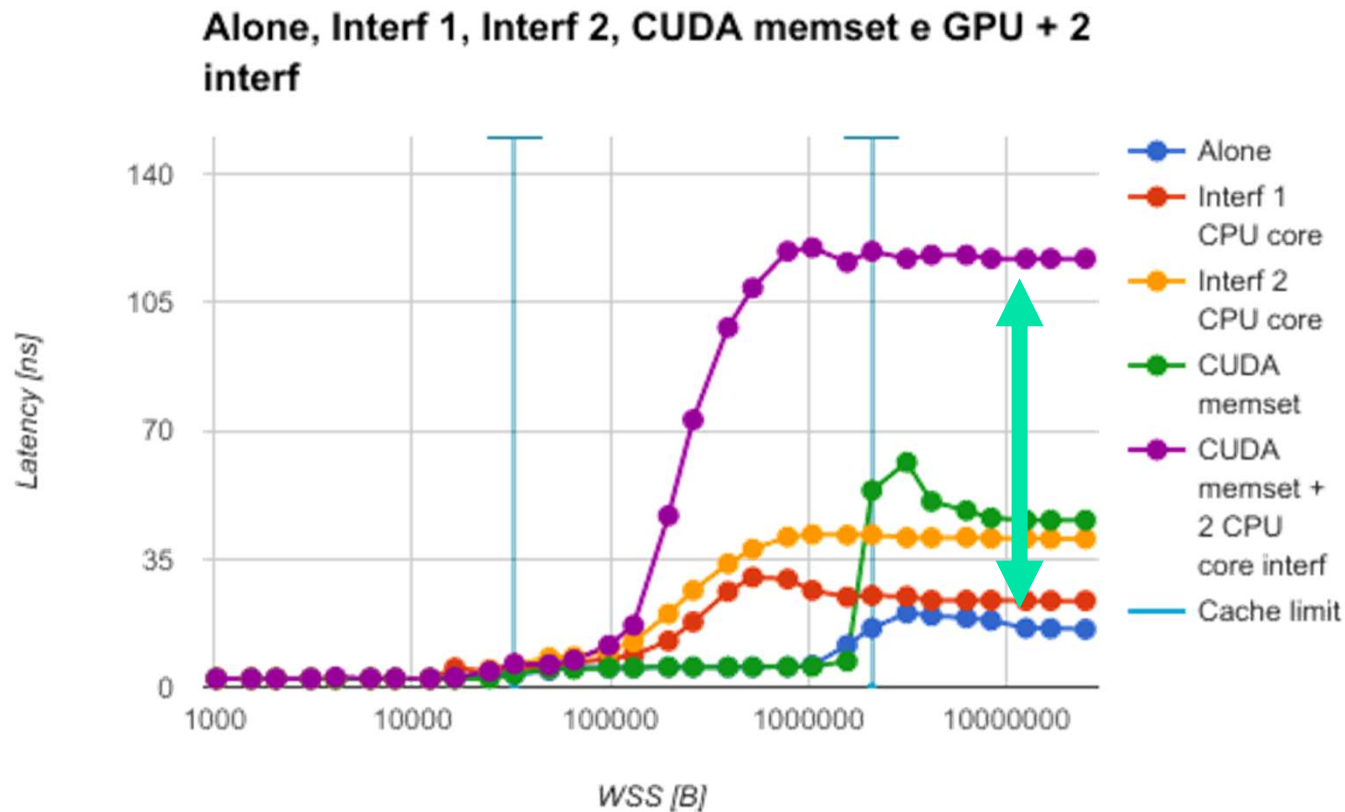
- Memory prefetching in the first phase
- No cache misses in the execution phase

Non-preemptive execution

# Core-level Memory Interference Drive PX2

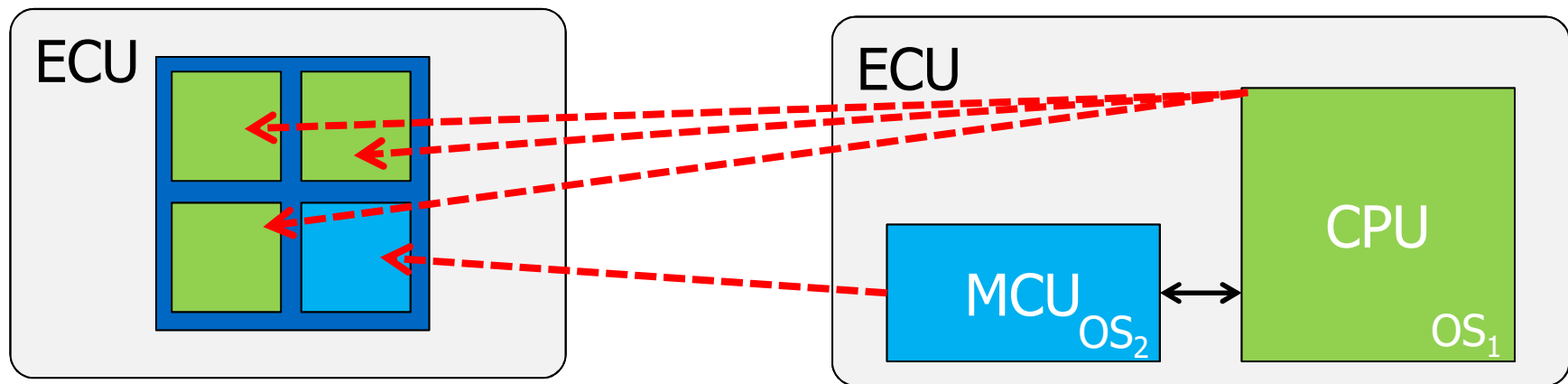


# Combined Interference Drive PX2



# Considering our use case

- Multicores can be exploited to consolidate vehicle functions A and B and on the same chip



- Problems:
  - How can we port software from multi-chip to multi-core?
  - How multiple Operating Systems can coexist?
  - How can we guarantee isolations of tasks?

# Outline

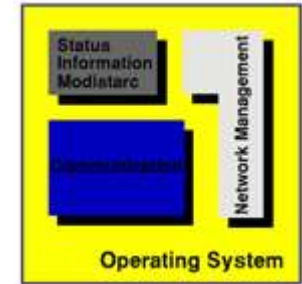
- Multicore architectures
- AUTOSAR
- Virtualization
- A use case
- Conclusions

# Automotive software architectures

- The automotive market has gone through waves of standardization
  - 90s – OSEK/VDX
  - 2004-2014 – AUTOSAR Classic
  - 2015- AUTOSAR Adaptive
- Main ideas
  - Standardize features
  - Decouple application from execution platform
  - Create a market of competitors
  - Lower the costs



# 90s – OSEK/VDX



- OSEK/VDX is an effort to standardize the RTOS for 8/16/32 microcontrollers
  - Static approach (configured with the OIL language)
  - Real time features
    - Fixed Priority with immediate Priority ceiling
    - Stack sharing between tasks
    - Debugger Awareness through the ORTI standard
    - Communication infrastructure (OSEK COM)
  - Single core
  - 2-6 Kb flash footprint
  - Certification procedure

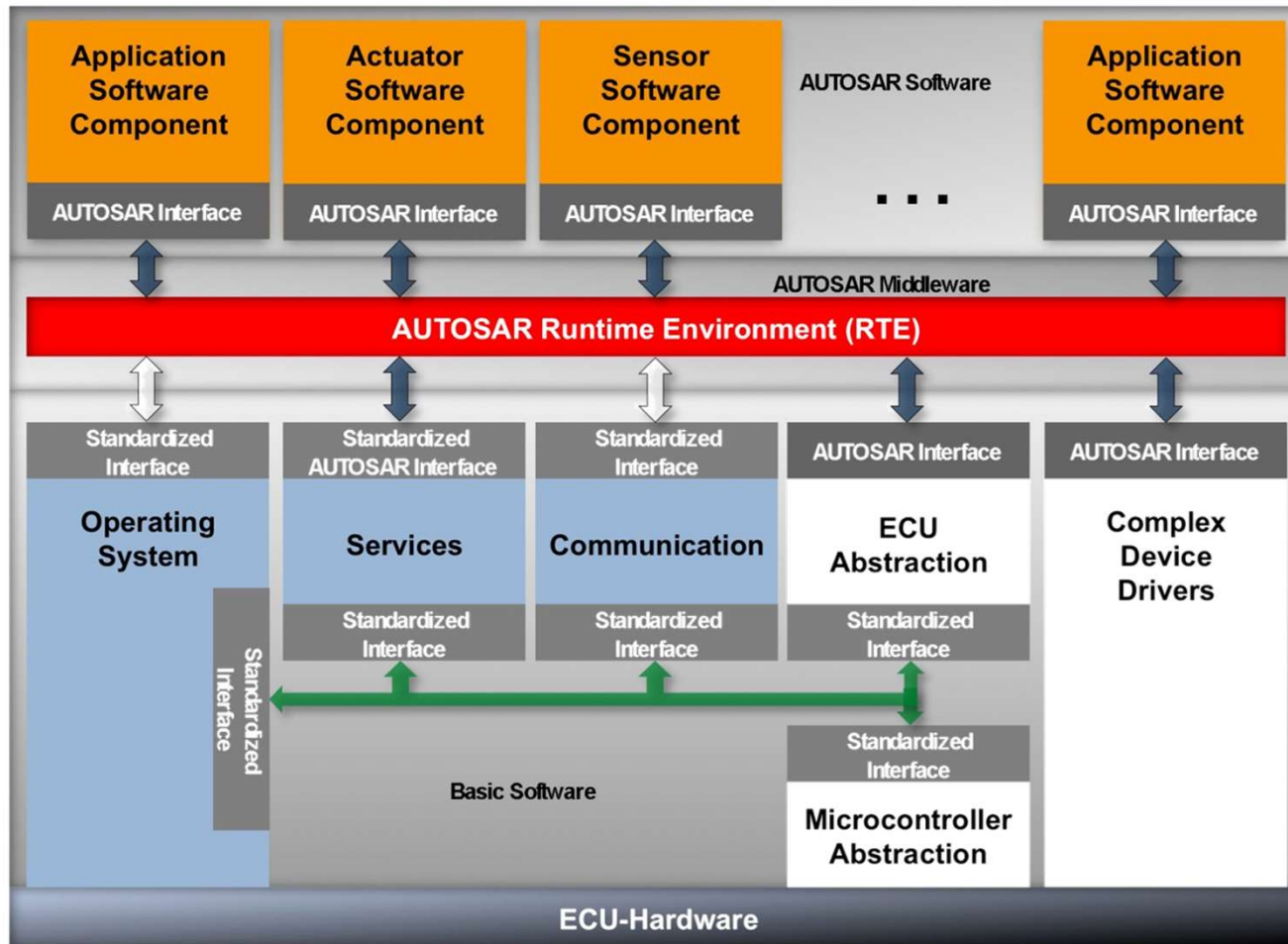
# From OSEK to AUTOSAR

- OSEK allowed the description of interconnected ECUs through a communication bus
- In modern cars → tens of interconnected ECUs
- The need becomes more the integration of **features** than the integration of **hardware boxes**

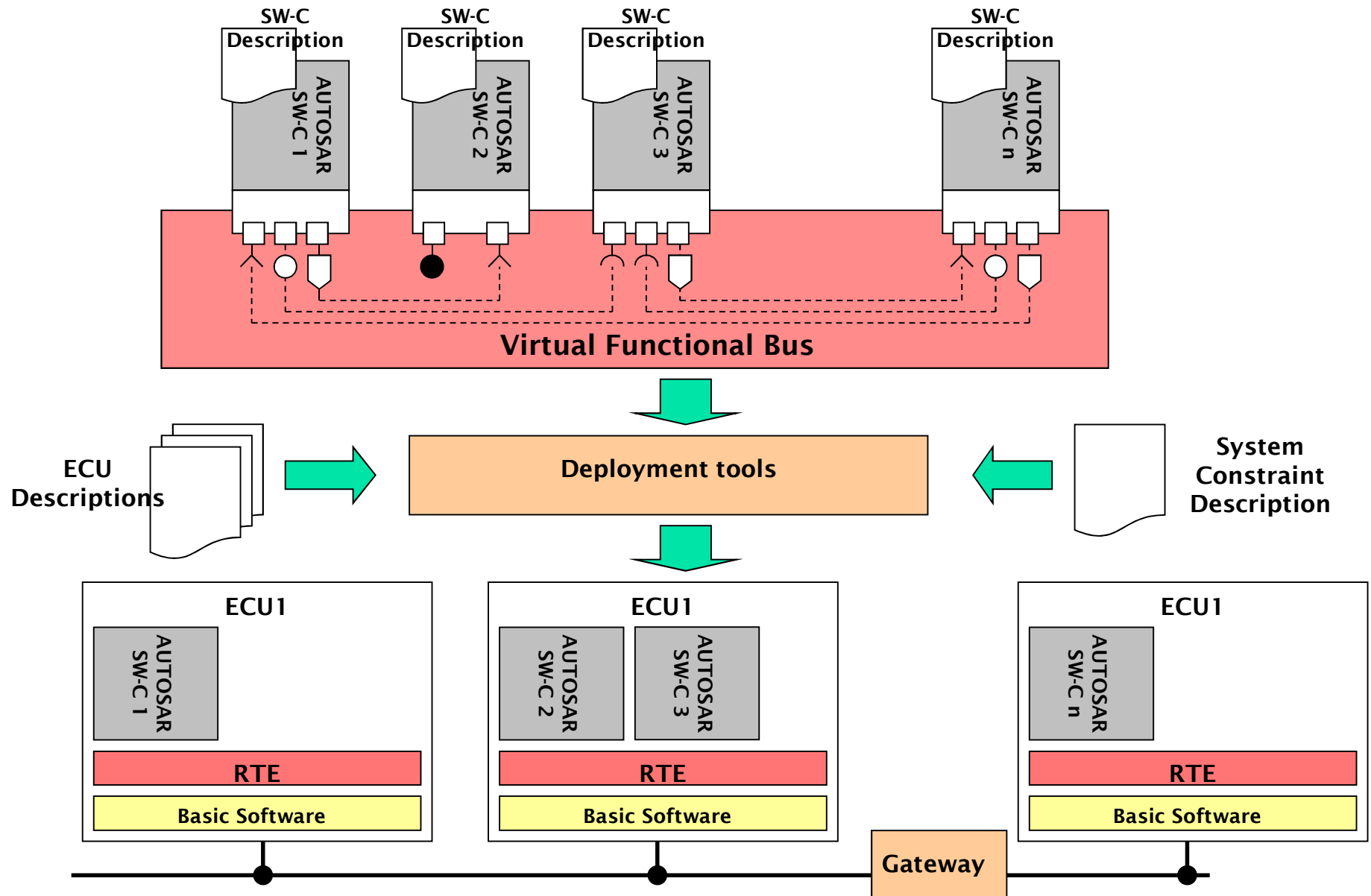
## AUTOSAR

- AUTOSAR gives an answer to this need, providing
  - A SW component model to ease the integration
  - A standardized basic software implementation

# AUTOSAR Classic Architecture



# AUTOSAR Classic VFB & RTE



# AUTOSAR important concepts

- AUTOSAR Components
  - Communicate through ports
  - Independent from their placement
- AUTOSAR OS
  - Extension of OSEK/VDX
- Basic Software and MCAL
  - Driver, Diagnostics and self test, ...
- Supported by tools
  - Using a common AUTOSAR XML format

# AUTOSAR OS Main concepts

- OS Applications:
    - containers of tasks used for memory protection and multicore partitioning
  - Multicore support
    - Static partitioning of tasks to cores
    - (non-FIFO) Spinlocks and remote activations
  - Memory protection supporting MPU
  - Timing protection and Stack Monitoring
- } ISO26262

# AUTOSAR critiques

- Limited support for non-functional specifications
  - Requirements / Execution time / ...
  - Enhancements through AMATHEA Project / EAST ADL
- Complexity
  - Creates barriers to the entry of new players
  - The market is mostly taken by 2-3 players
- Limited support for open-source ... but ...
  - ERIKA Enterprise <http://www.erika-enterprise.com>
  - COMASSO <http://www.comasso.org>

# Something about ERIKA Enterprise



<http://www.erika-enterprise.com>

- ERIKA Enterprise is an RTOS OSEK/VDX certified
- ERIKA Enterprise implements the AUTOSAR OS API
- With a suitable open-source license allowing static linking of closed source code (GPL + Linking Exception)
- Typical footprint around 2-4KB Flash
- Used in automotive applications and research projects
- ERIKA3 supports now various automotive CPUs



Vodafone  
Automotive





# Future of AUTOSAR

- Future support for ADAS, automatic driving, Car2X
  - high performance, dependable systems, distributed diagnostics
  - cloud integration / support for non-AUTOSAR systems
- Adaptive AUTOSAR, based on POSIX PSE51
- coexistence in the same multicore system of both AUTOSAR Classic and AUTOSAR Adaptive
  - Thanks to the virtualization support



# Considering our use case

- Software shall be redesigned according to the AUTOSAR principle
  - Software components are responsible for implementing the needed functionalities
  - The software components are execution-platform agnostic
- AUTOSAR deployment tools are used to map software components to the cores
- Problems:
  - How can we port software from multi-chip to multi-core? ✓
  - How multiple Operating Systems can coexist?
  - How can we guarantee isolations of tasks?

# Outline

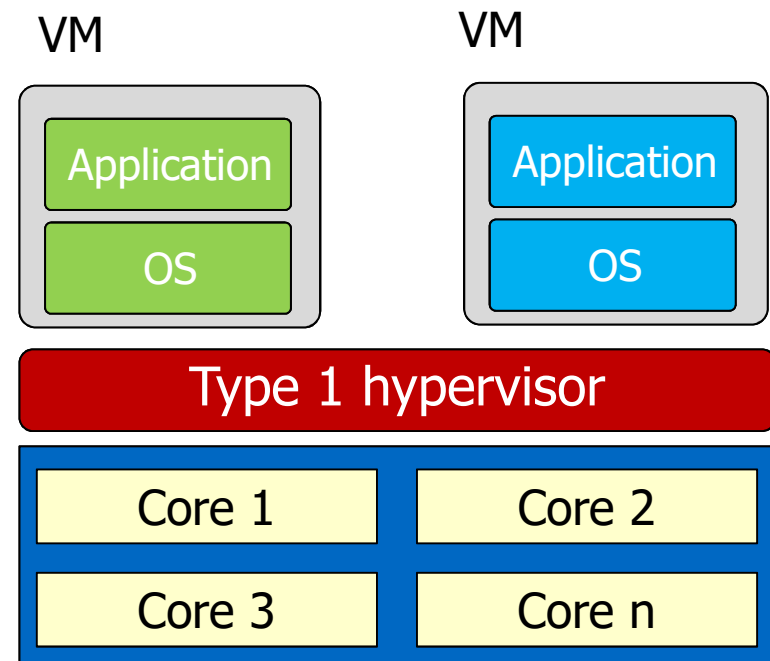
- Multicore architectures
- AUTOSAR
- Virtualization
- A use case
- Conclusions

# Virtualization

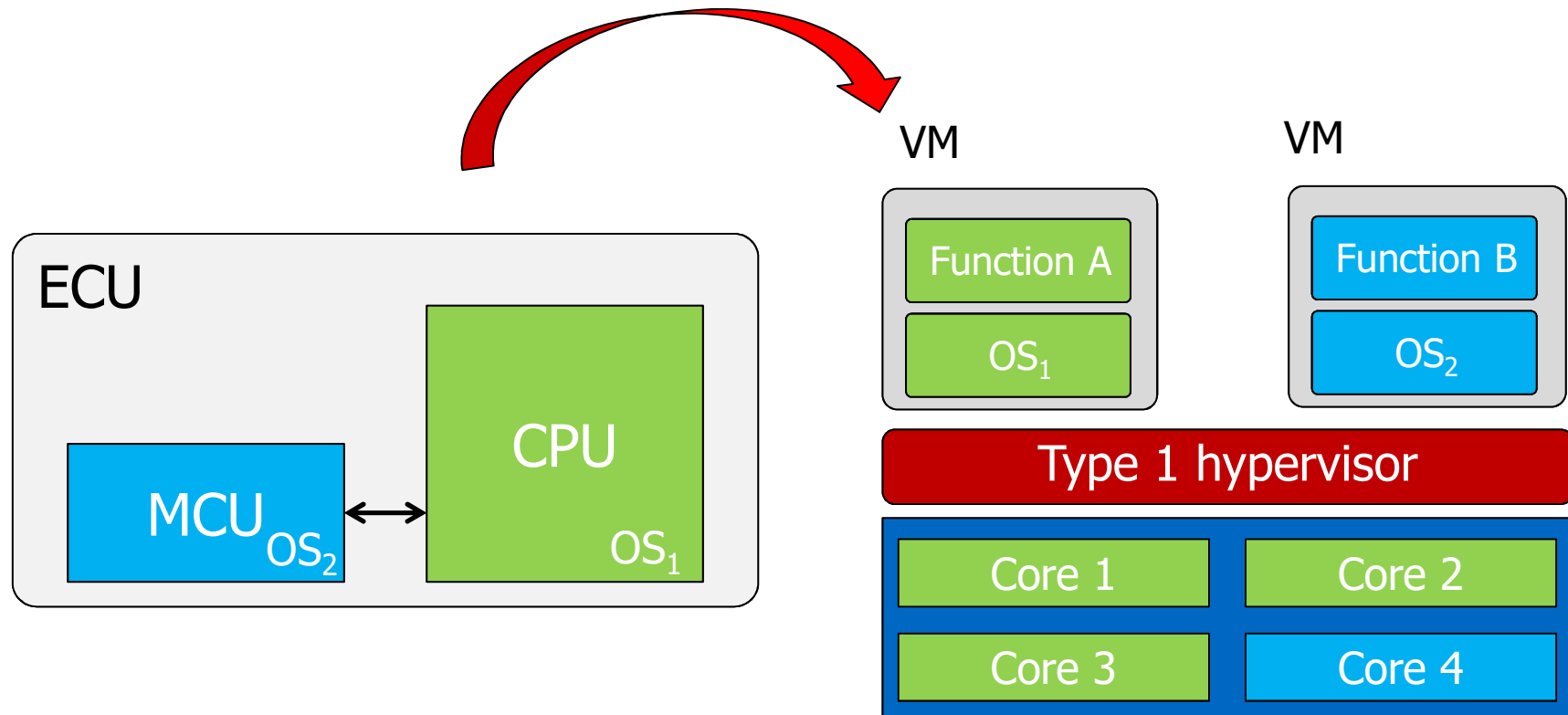
- Virtualization is a technology to abstract a hardware platform into virtual machines (VMs)
  - A VM uses a sub-set of the available hardware
  - VMs running on the same hardware are not aware that they are sharing the platform with other VMs
- Old technology
  - Introduced in the 1960 on mainframes
  - Rediscovered in 2000s for embedded systems

# Type-1 Hypervisors

- In embedded systems virtualization is done using **type-1 hypervisors**
  - Based on Microkernel → easy to validate
  - They manage isolation of VMs by intercepting all privileged instructions
  - They implement CPU scheduling
  - They supports inter-VM communications



# Considering our use case



## ■ Problems:

■ How can we port software from multi-chip to multi-core? ✓



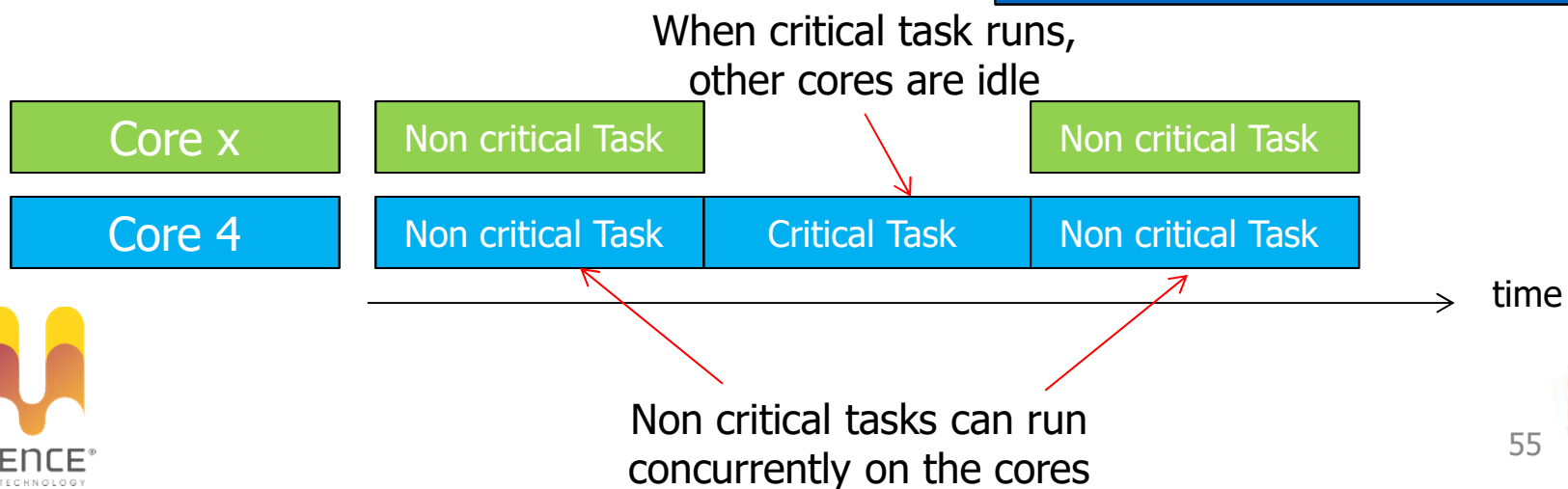
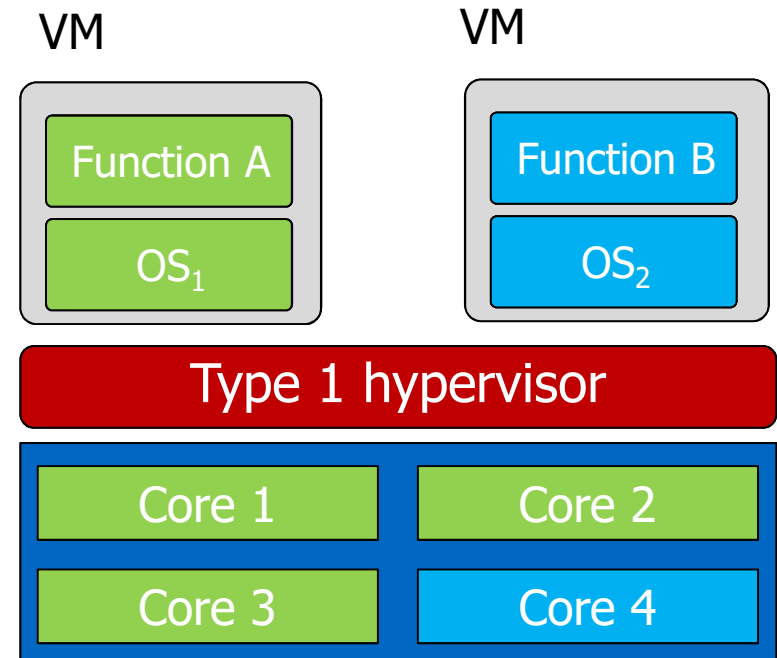
■ How multiple Operating Systems can coexist? ✓

■ How can we guarantee isolations of tasks? ✓



# Considering our use case

- Hypervisor segregates functions
  - Microkernel guarantees separations of hardware resources between each VM
  - Scheduling is used in case time interference is critical





# Outline

- Multicore architectures
- AUTOSAR
- Virtualization
- A use case
- Conclusions

# Use Case: the HERCULES Project



H2020 project – <http://hercules2020.eu/>

- Main outcome
  - Integrated framework to achieve predictable performance on top of cutting-edge heterogeneous COTS multi-core platforms
- Technological baseline
  - Real-time scheduling techniques
  - High-performance/Low-power embedded COTS
  - Next generation real-time applications

# Hardware Architecture

- Starting point: the **hardware architecture**
  - **Tegra-like platform** for handling high performance computational loads with low power budgets and potentially low predictability
  - **Safety microcontroller** for real-time safety applications up to ASIL D

# Programming model abstractions

- Support for dynamic applications using **Linux**
- Support for legacy real-time applications using **AUTOSAR-like stacks**

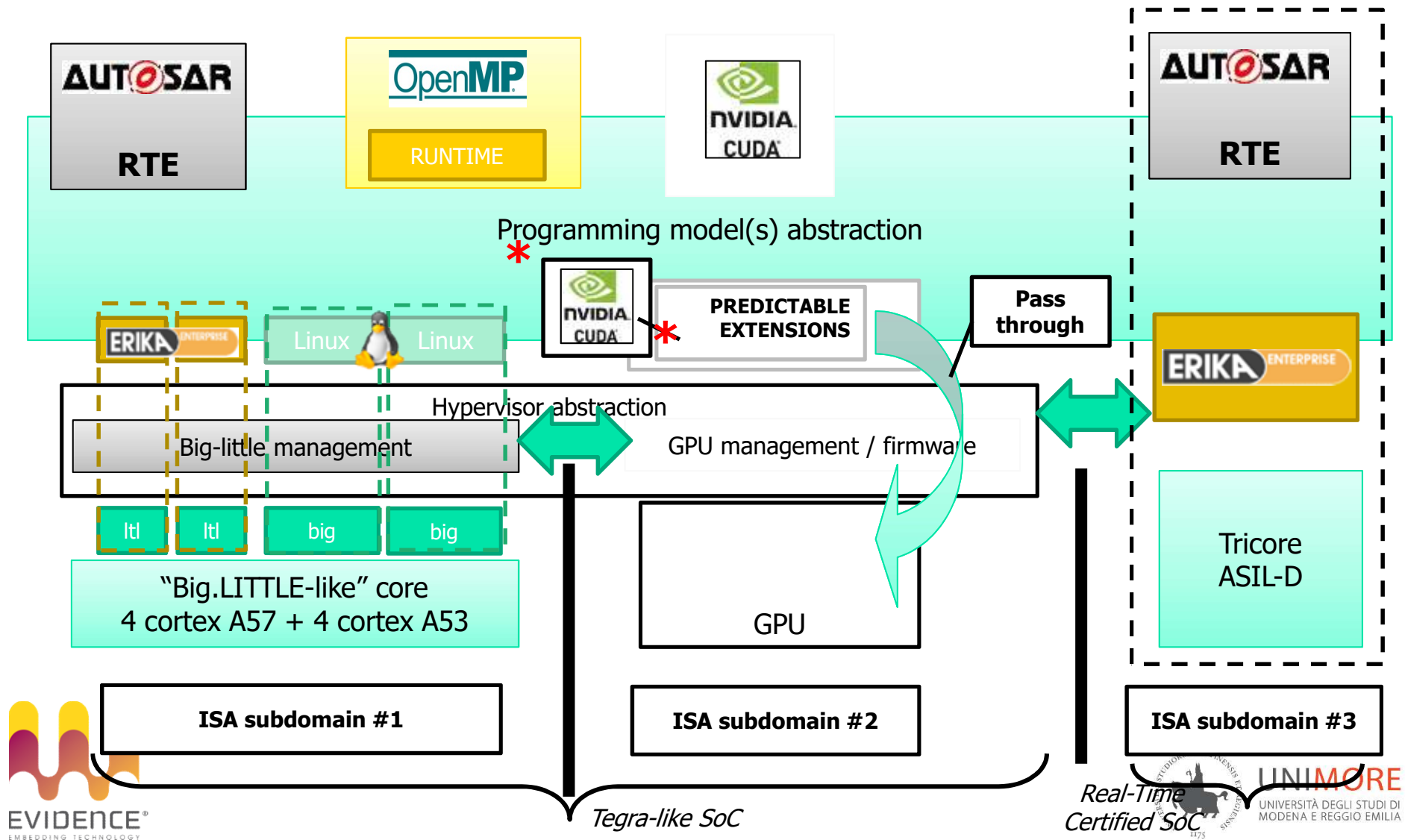


- Support an open-source OSEK/VDX implementation named **ERIKA Enterprise**, extending it to support a subset of the **AUTOSAR RTE**
- Pinning **one OS per core** to reduce overhead and complexity, and guarantee better isolation

# Integrating different subsystems

- An application will be likely composed by
  - A static part, implemented with an AUTOSAR RTE
  - A dynamic part, implemented with Linux and the GPUs
- We want to integrate them together in the same multicore CPU!
- Idea: use a **Hypervisor**
  - Cores assigned statically to domains – not like it happens in a Cloud environment!
  - Need to share peripherals... and the GPU!
  - Initial attempt based on JailHouse

# HERCULES Software architecture



# ERIKA3 + Hypervisor(s)

- We presented an integration of ERIKA3 on Tegra Parker @GTC Munich 2017
- We released a version of ERIKA3 working on top of the JailHouse Hypervisor
  - Check the new Virtual machine on the ERIKA3 website!
- We added support for ERIKA3 under Xen, with EtherCAT support



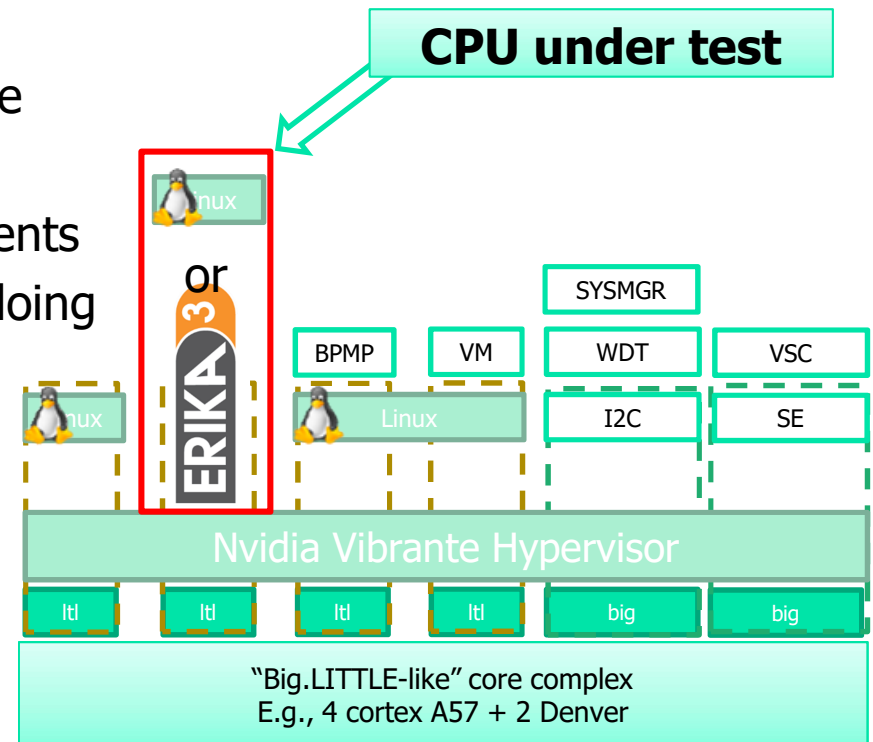
# Perf. measurements on NVIDIA Drive PX2

Nvidia Vibrante configuration:

- We considered ERIKA3 pinned on one of the Cortex A57
- Linux on the other 3 A57 cores
- Other VMs moved to Denver when possible

We are interested in the following measurements

- ISR Latency with the CPU idle or «busy» doing RTOS primitives
- AUTOSAR Task wakeup Latency
- Linux `clock_nanosleep` periodic task latency
- Variability when other CPUs are executing memory intensive tasks

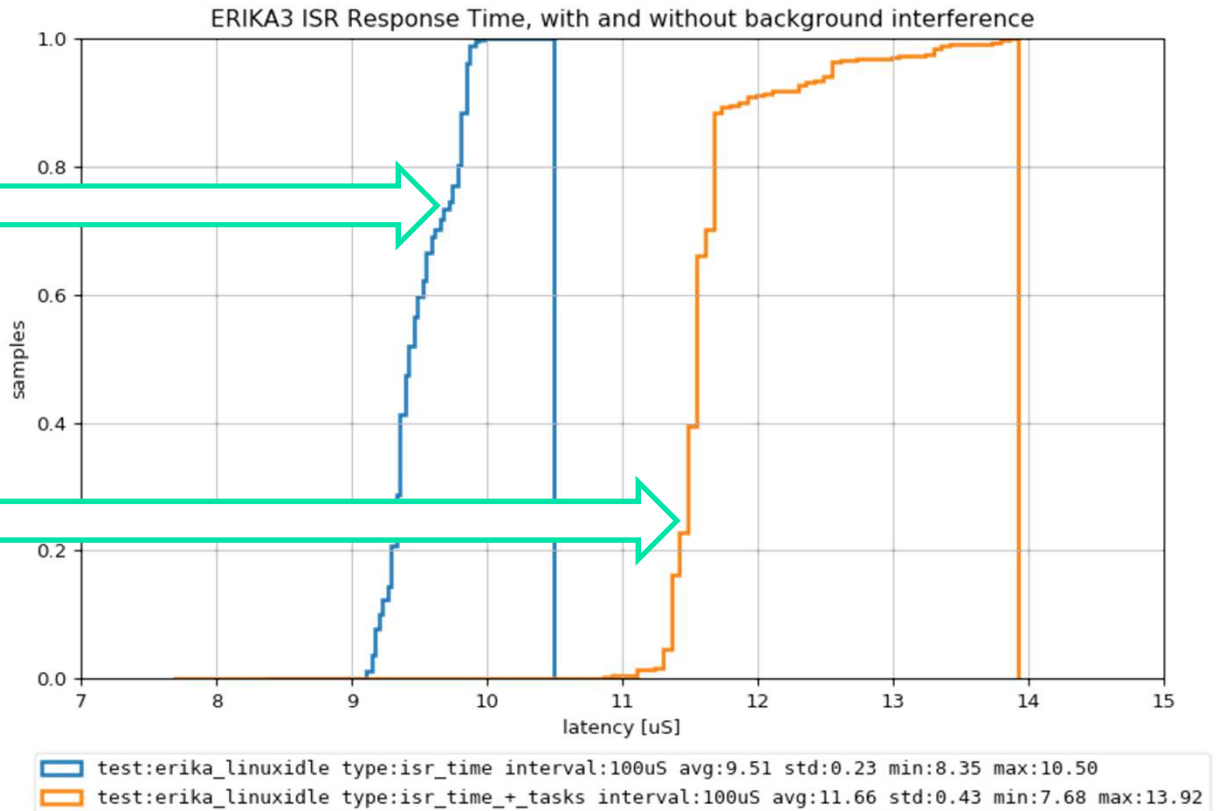


# ISR Latency Timings

- Cumulative distribution of the execution times

**ISR Latency,  
ERIKA3 idle  
9-10.5  $\mu$ s**

**ISR Latency,  
ERIKA3 «busy»  
11-14  $\mu$ s**

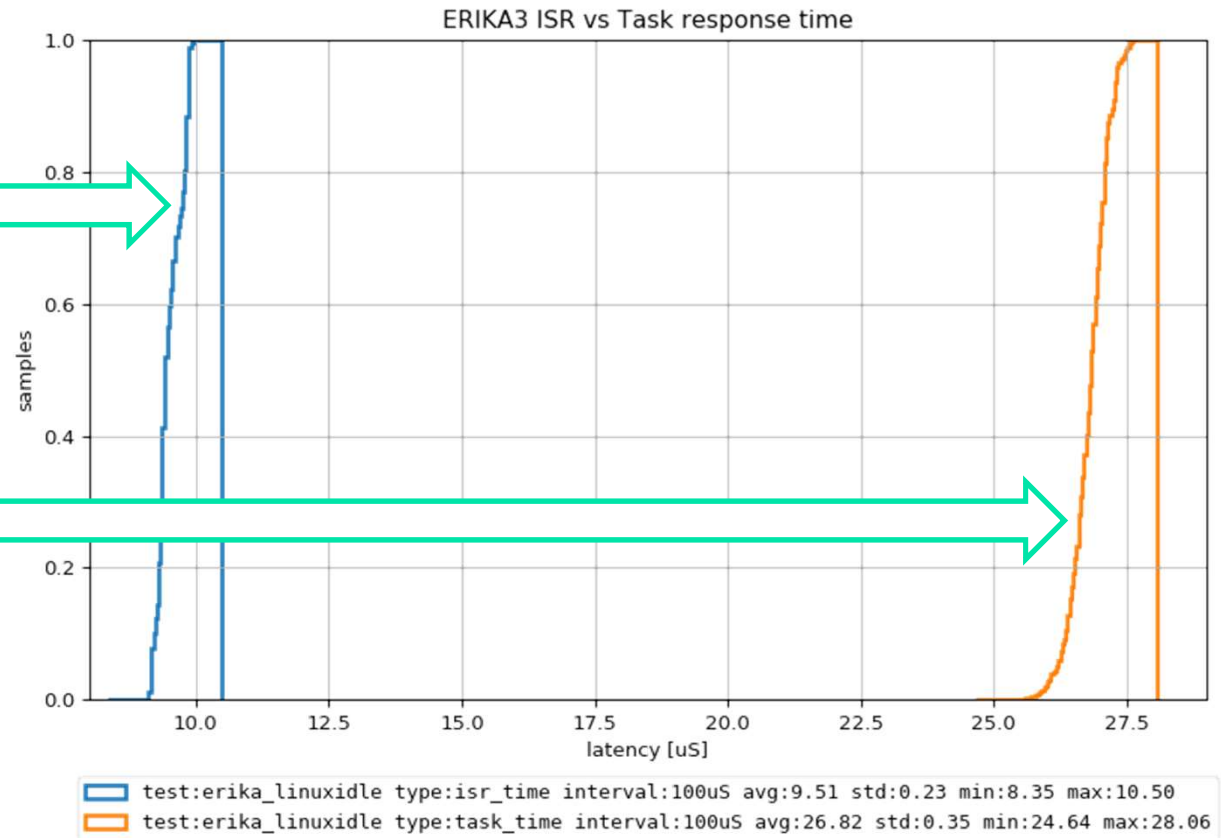


# ISR/Task Latency Timings

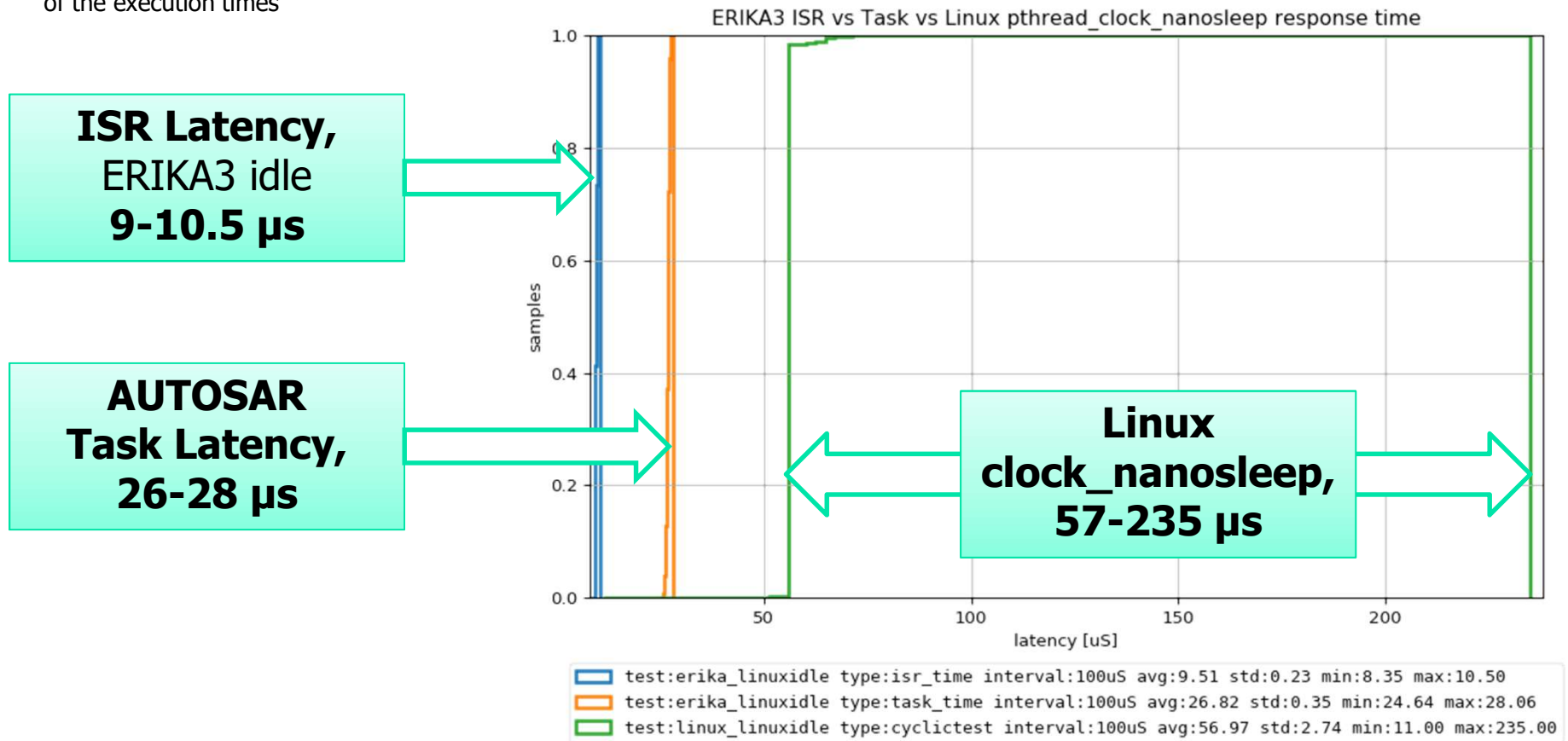
- Cumulative distribution of the execution times

**ISR Latency,  
ERIKA3 idle  
9-10.5  $\mu$ s**

**AUTOSAR  
Task Latency,  
26-28  $\mu$ s**

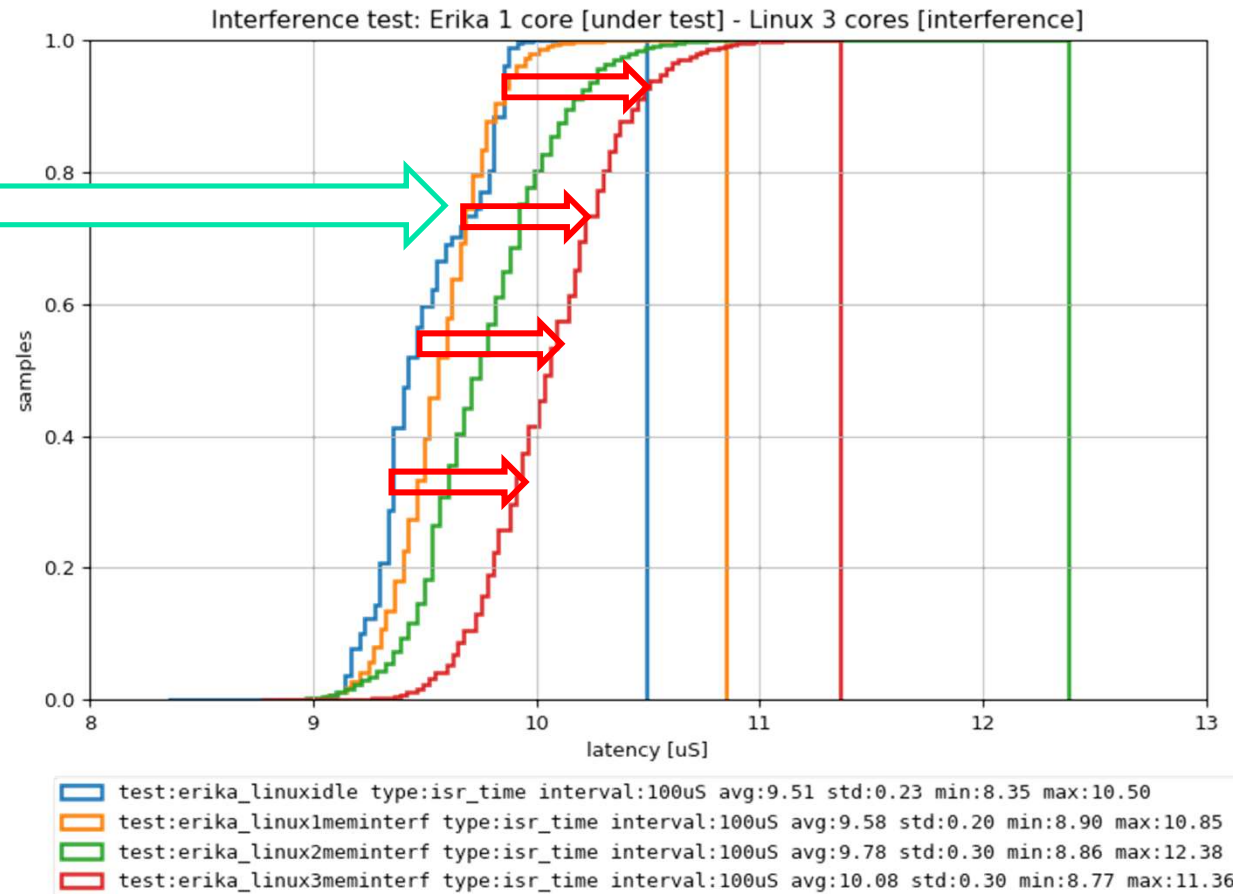


- Cumulative distribution of the execution times

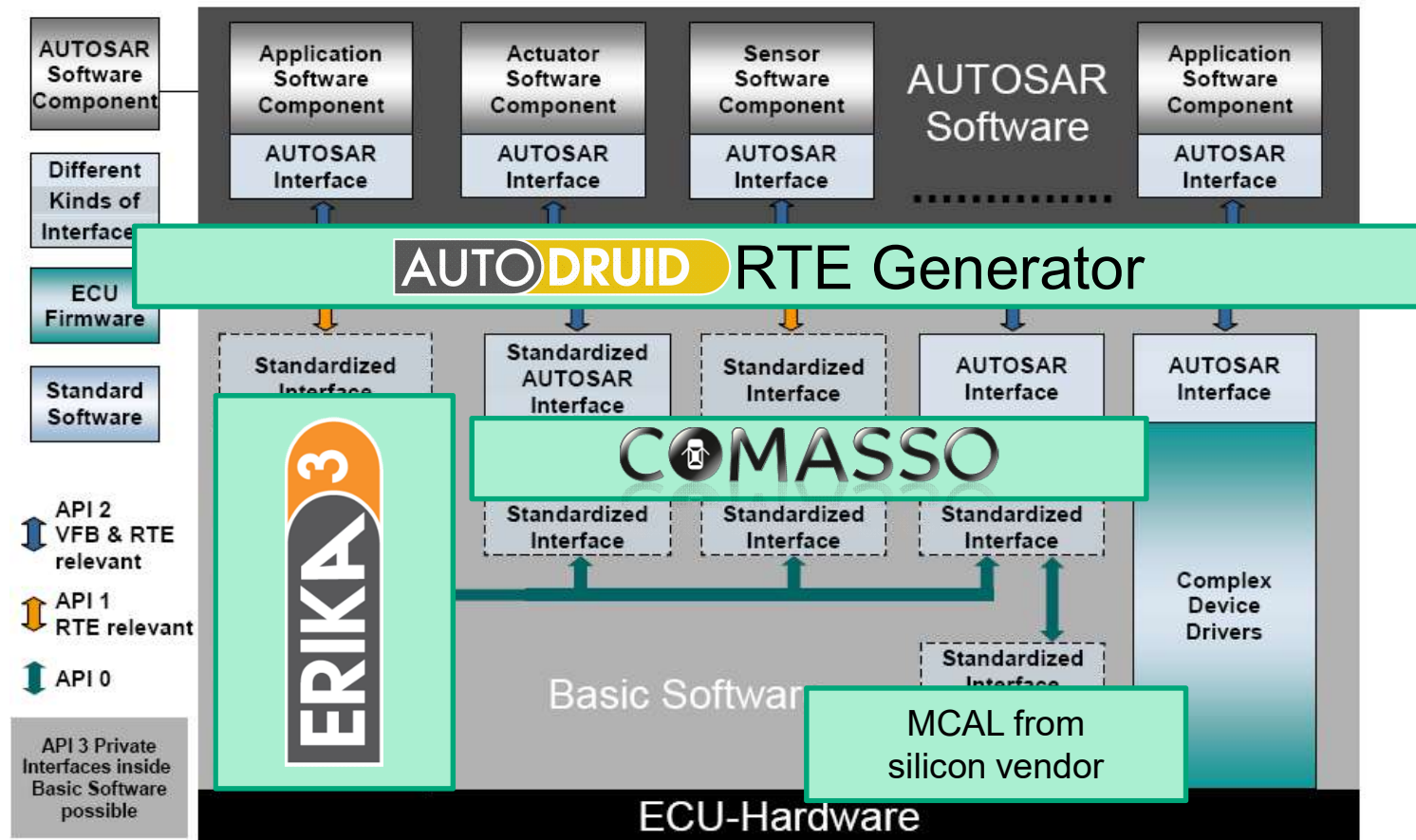


- Cumulative distribution of the execution times

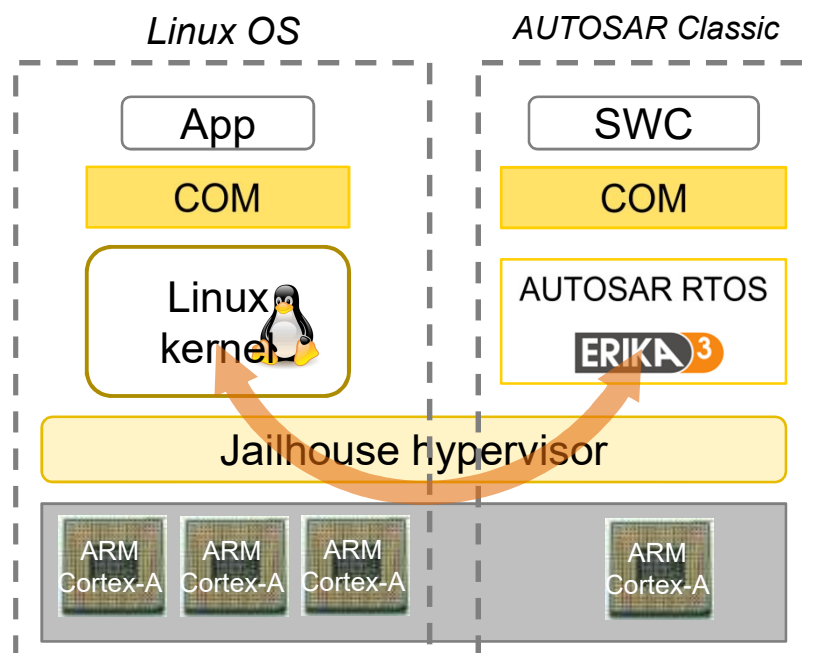
**Degradation due to memory intensive load on other CPUs**



# A full AUTOSAR classic stack



# AUTOSAR COM Layer



- Library on top of Jailhouse's mechanism
- Blocking and non-blocking calls
- Dynamic-size messages
- Similar to AUTOSAR COM API:

```
Com_StatusCode Com_GetStatus();
```

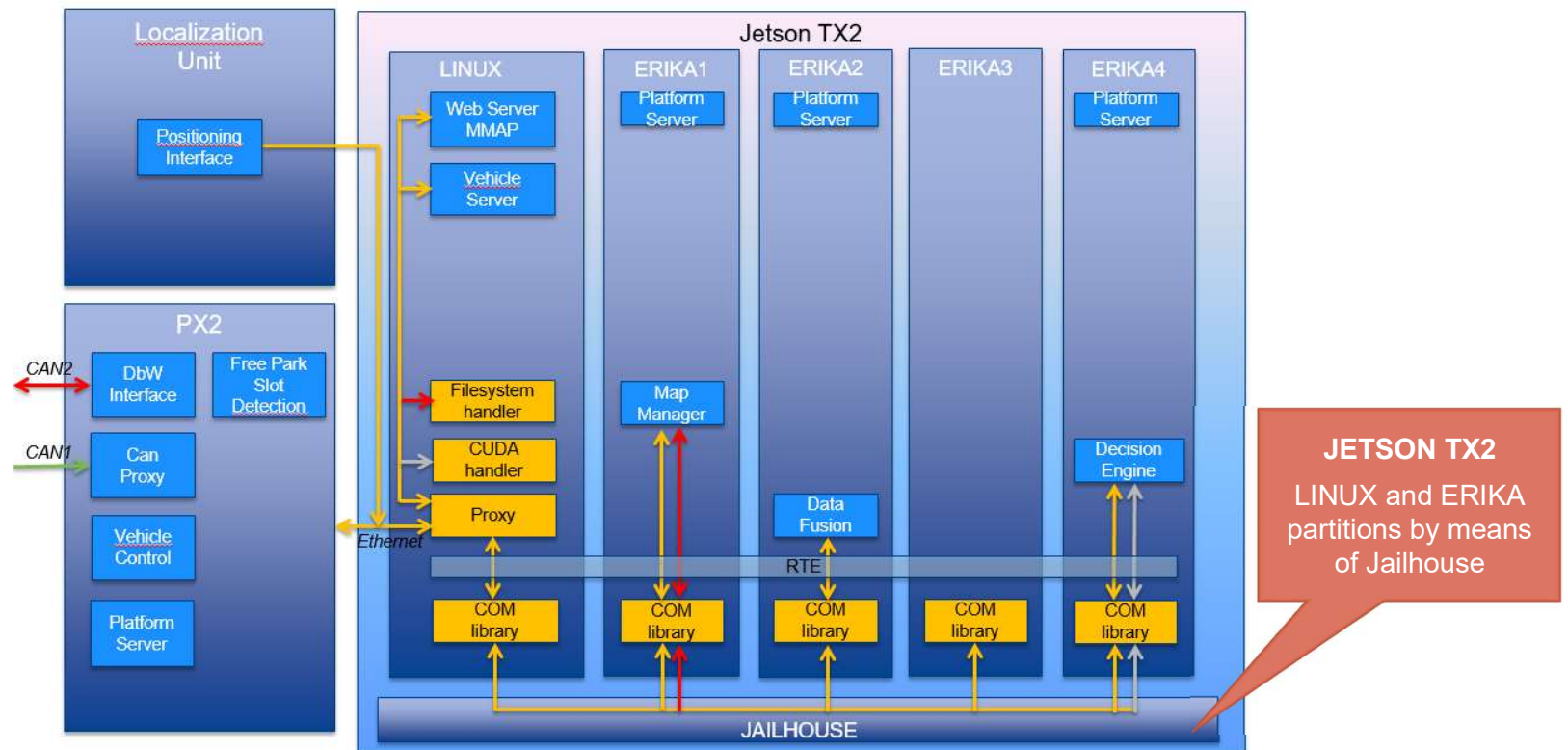
```
uint8 Com_SendSignal(Com_SignalIdType SignalId, const void  
*SignalDataPtr);
```

```
uint8 Com_ReceiveSignal(Com_SignalIdType SignalId, void*  
SignalDataPtr);
```

- Developed by EVI in RETINA EU project

**RETINA** 

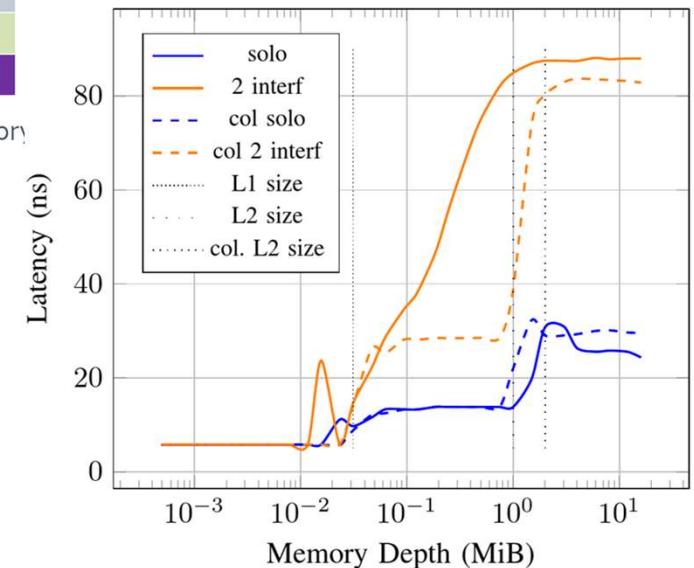
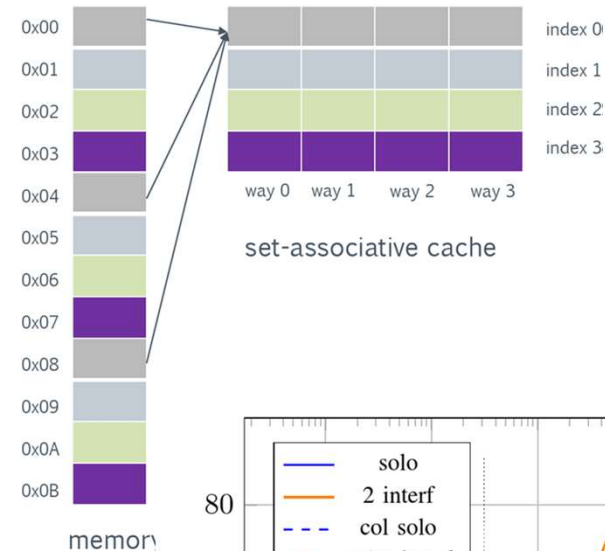
# Application SW architecture





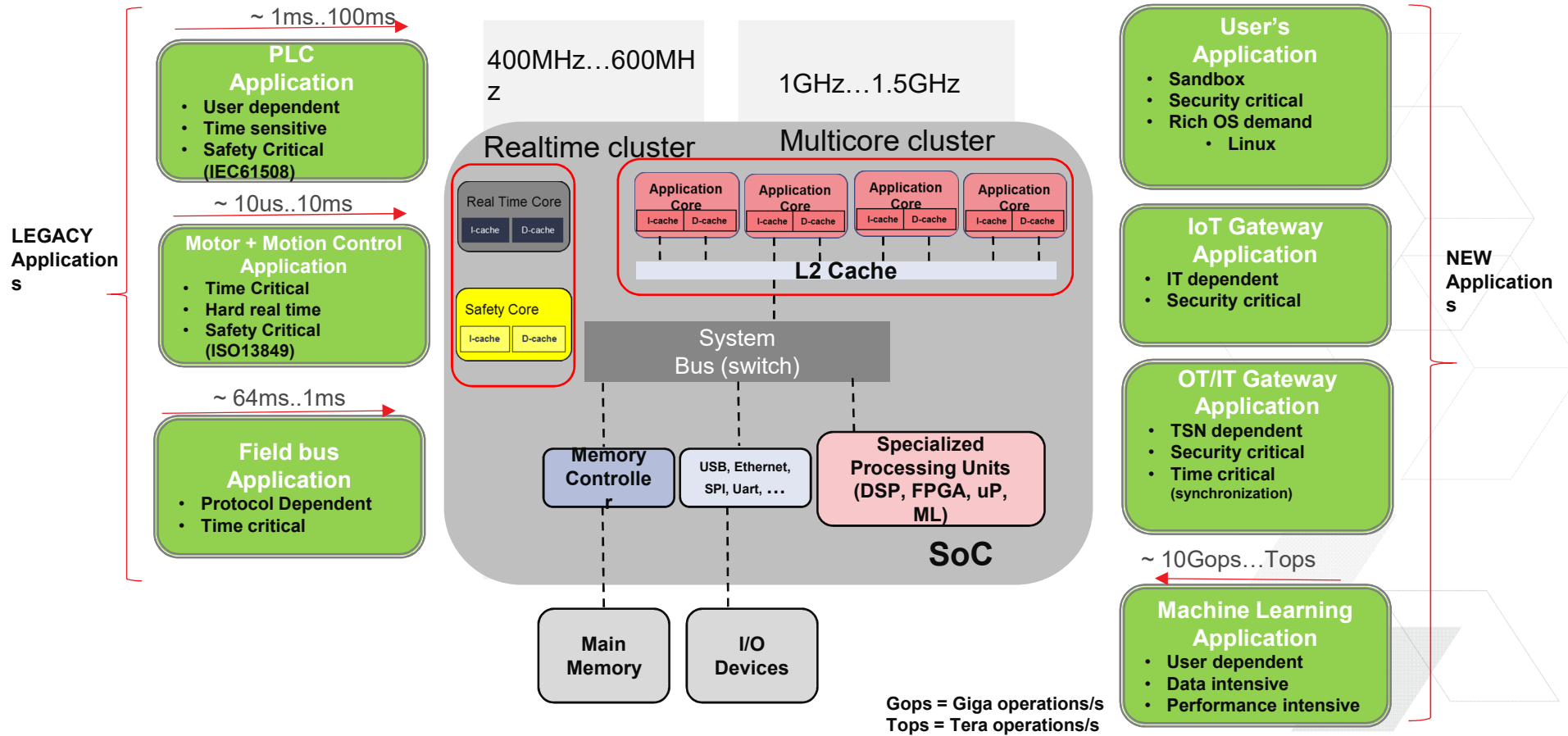
# Predictable Caches: Page Colouring

- Neighbor cores cause cache eviction on shared L2 cache, hence unpredictable memory latency (hit/miss)
- Partition cache in isolated regions with coloring support in Jailhouse!
- [Kloda et al. RTAS2019]

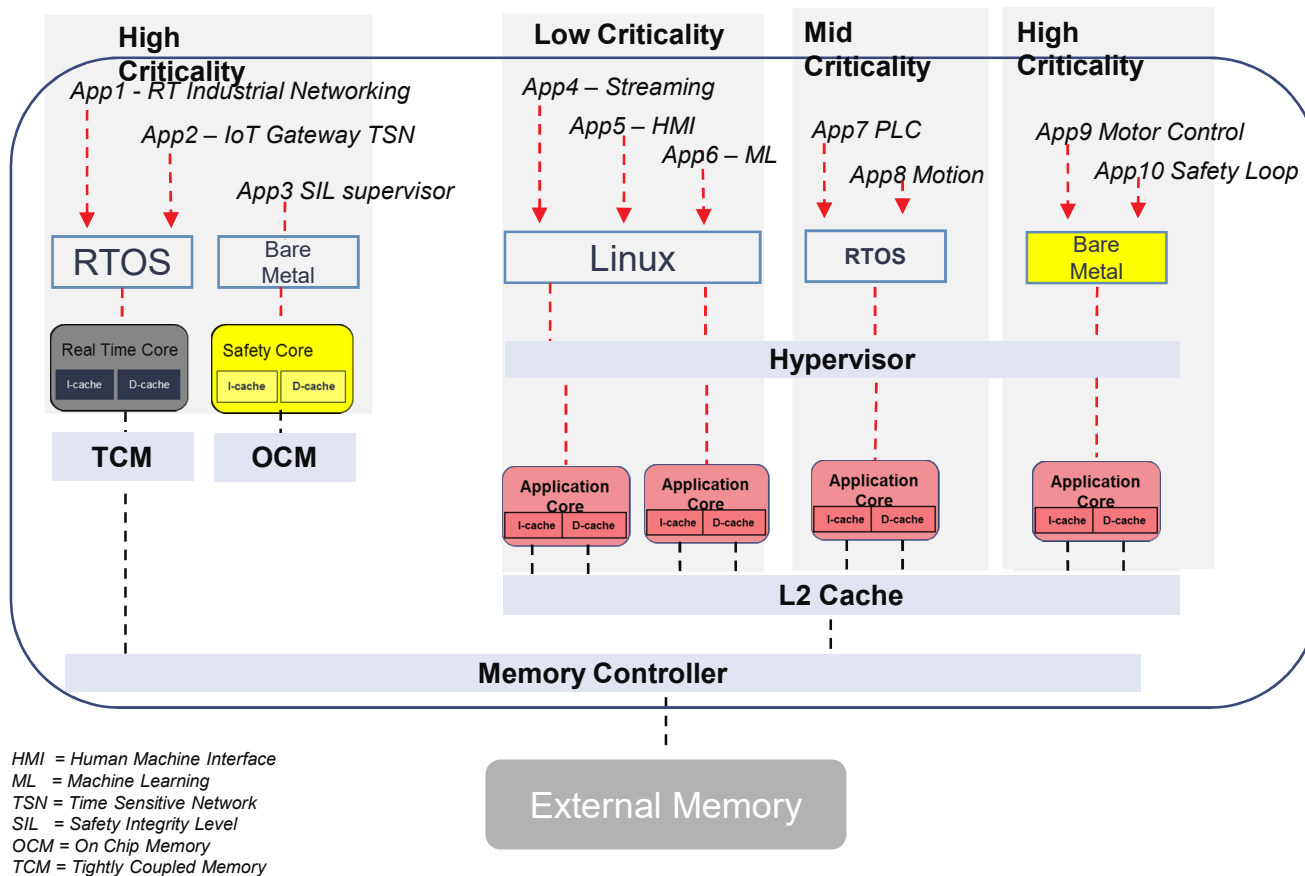


(a) Sequential Access (64 B stride).

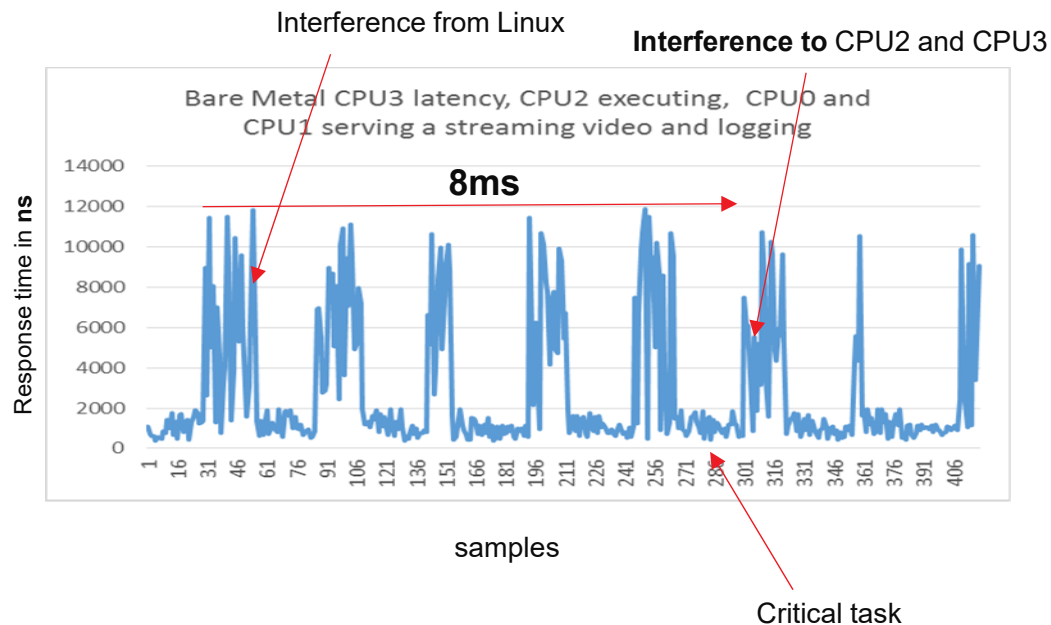
# The partition problem



# Conceptual allocation of applications in many core SoC



# Do we have absence of interference?



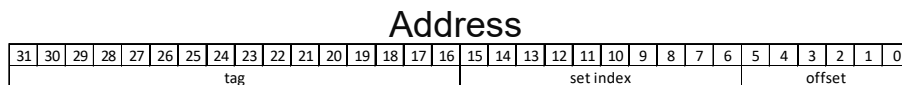
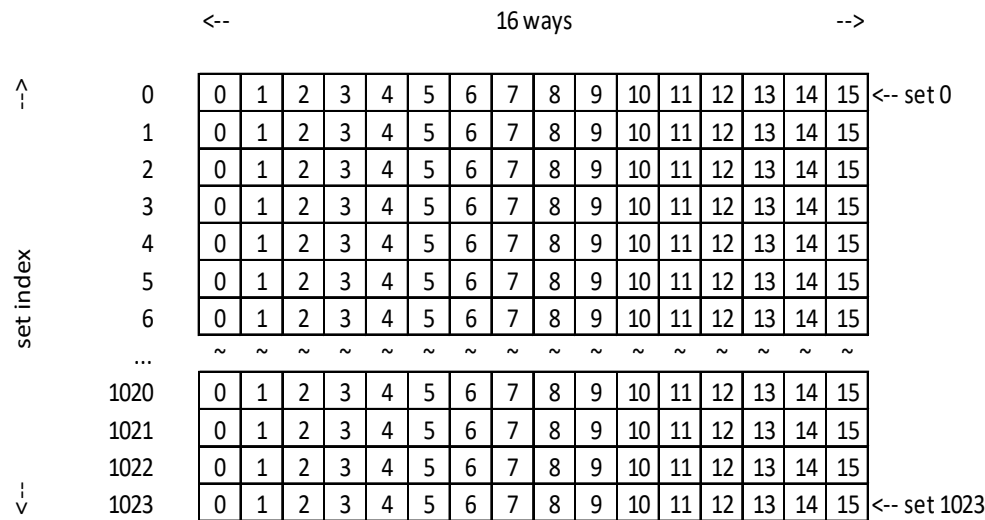
## > Conditions

- >> CPU3 executing a 48Kbyte code every **256us** as motor control task + safety loop
- >> CPU2 executing a “PLC” like 48K byte code every **8ms**
- >> CPU1 executing streaming of 2 Megabytes of data
- >> CPU0 executing DDR access for data logging with 256 Kilo bytes stream

## > Measurements

- >> CPU3 executing with response time between **~400ns** and **~12000ns** a 30x deviation!
- >> CPU2 executing with response time between **~2000** and **~12000 ns**
- >> **Clear and significant interference!**
- >> **Likely to be the L2 cache again**

# How cache works on Zynq® UltraScale+™



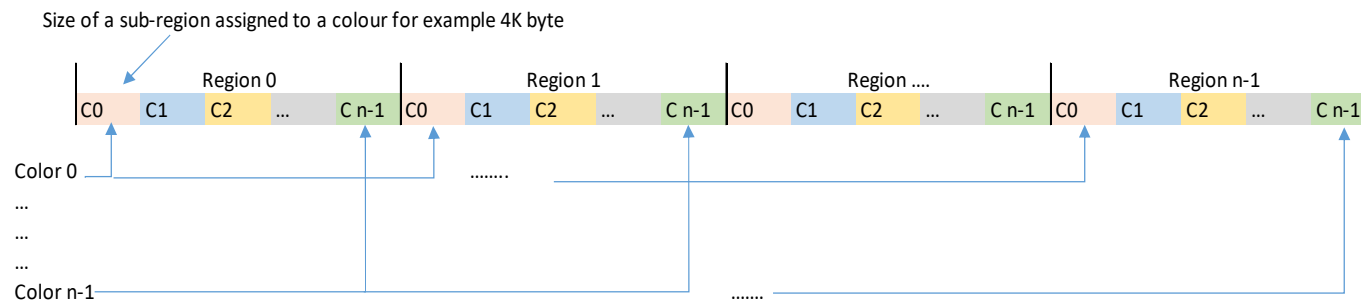
## > Cache organization

- >> The size of Level 2 cache of the Zynq UltraScale+ is 1 Mbyte.
- >> It is subdivided in 16 chunks called ways.
- >> Each of the squares in the figure represents one cache block (= one cache line of 64 byte).
- >> The cache controller divides the address in three parts:
  - The offset defines a byte within a cache line.
  - With 64 byte cache line size these are the 6 LSB's ( $64 = 2^6$ )
  - The set index bits define in which set the cache line gets stored.
  - As there are 1024 sets, 10 address bits are used for this purpose ( $1024 = 2^{10}$ ).
  - The rest of the address bits form a unique tag.

# Cache coloring

- > **Cache colouring is a software technique for cache partitioning without hardware support.**
  - >> Fragments the memory space into a set (colours)
  - >> Colours addresses are mapped to disjoint cache partitions.
  - >> Achieved dividing the whole memory space into sequential regions sizing as a way-size.
- > **For example 1G bytes of memory**

$$Number_{of\_regions} = \lfloor \frac{memorysize}{waysize} \rfloor = \lfloor \frac{1Gbyte}{64Kbyte} \rfloor = 16384regions$$



# Colouring policy definition

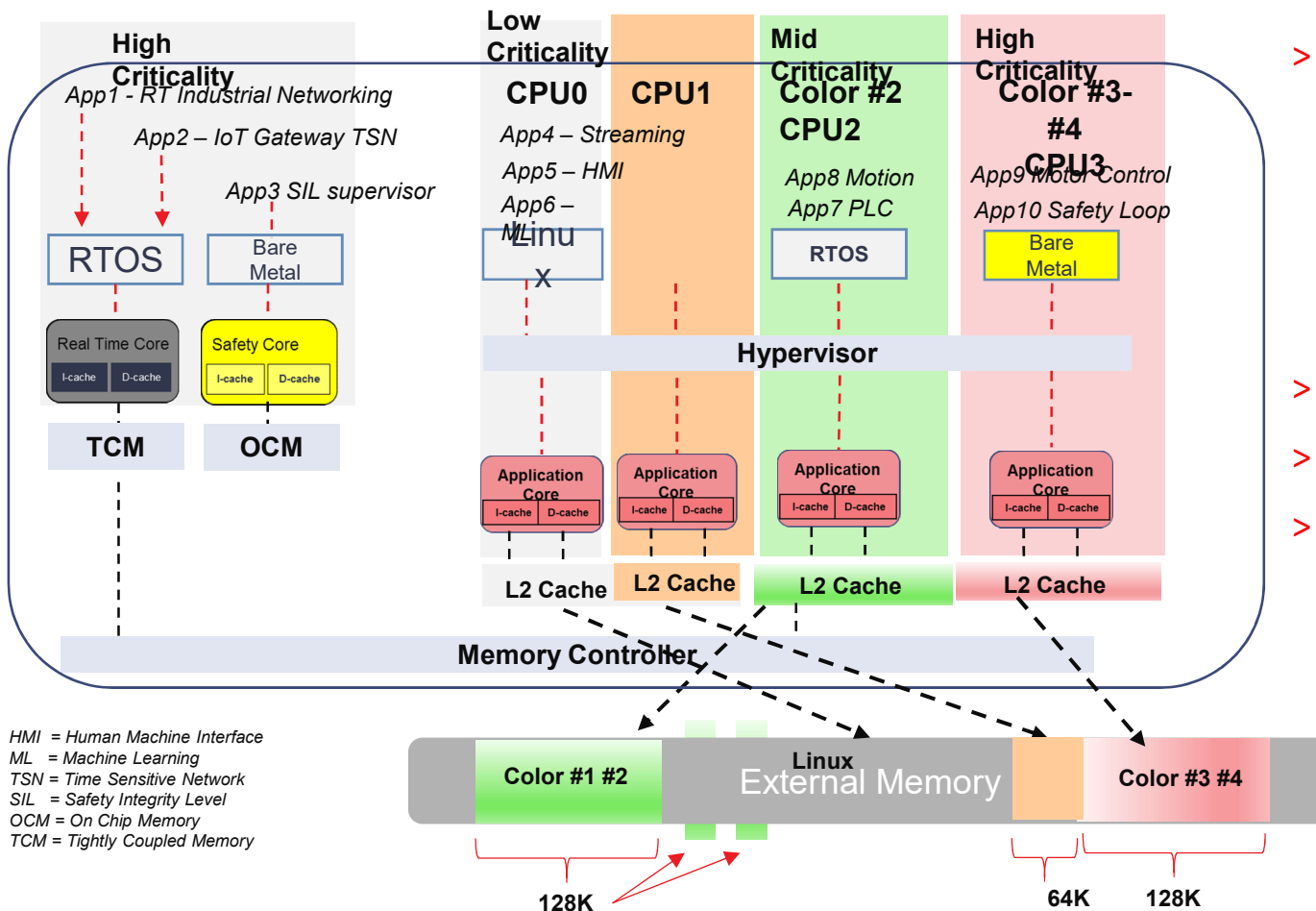
Extending the configuration of a new platform on the coloured version of Jailhouse is straightforward. It suffices to define two parameters:

- Fragment size it must match the cache way size, hence 64 K byte on US+.
- Sub-colour size it must be a multiple of the page granularity, so let us assume the smallest on ARMv8-A architecture value of 4 K byte.

From this two parameters we determine the number of available colours – 16.

$$number_{of\ colours} = \lfloor \frac{L2_{memorysize}}{waysize \cdot pagesize} \rfloor = \lfloor \frac{1Mbyte}{16 \cdot 4Kbyte} \rfloor = 16colours$$

# Cache coloring + Jailhouse reincarnation



## > Results

- >> Interference amongst Core 3 and Core 2 is eliminated
- >> Contiguous memory map in function of the number of color assigned to CPU
- >> Cache "lockdown" same size of number assigned colors

## > Predictability improved

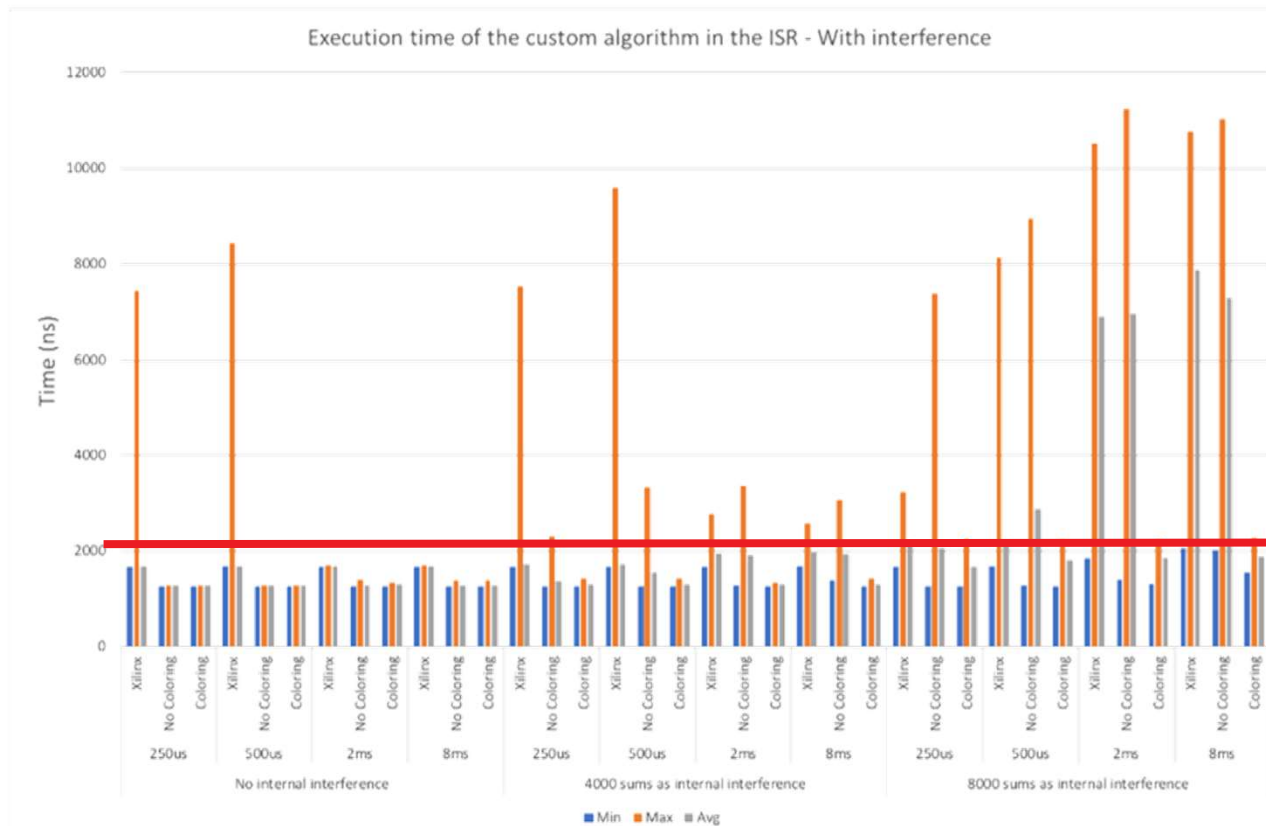
## > Separation improved

## > Linux re-incarnated

- >> Coloring no interference



# Cache coloring + Jailhouse reincarnation benchmark



## > Improved predictability

- >> Almost constant response time **~2000ns in all stressing conditions**

## > Minimal interference

- >> Interference amongst cores is eliminated
- >> Still possible internal interference, within threads in a single CPU, but has to be managed at application level

# Outline

- Multicore architectures
- AUTOSAR
- Virtualization
- A use case
- Conclusions

# Conclusions

- Multi-cores are the response to automotive needs, but only in conjunction with other technologies
  - AUTOSAR
  - Hypervisors
  - ...
- Time interference limits the possibility to exploit fully multi-core capabilities in real-time scenario
  - Some hardware features available in the newest devices could help (e.g., resource pre-allocation for quality of service guarantee)

# Contacts

Paolo Gai

[pj@evidence.eu.com](mailto:pj@evidence.eu.com)

<http://www.evidence.eu.com>

Marko Bertogna

[marko.bertogna@unimore.it](mailto:marko.bertogna@unimore.it)

<https://hipert.unimore.it/>

Massimo Violante

[massimo.violante@polito.it](mailto:massimo.violante@polito.it)

<http://www.cad.polito.it>

Giulio Corradi

[giulio.corradi@xilinx.com](mailto:giulio.corradi@xilinx.com)

<https://www.linkedin.com/in/giulio-corradi-860b71b/>

Content mostly taken from:

Massimo Violante, Paolo Gai,  
Automotive embedded software architecture in the multicore age.  
at 21st IEEE European Test Symposium, May 23 - 27, 2016, Amsterdam, The Netherlands

Plus additional content courtesy of Marko Bertogna and Giulio Corradi

