Virtual Open Systems

High Computing ARMv8 Platforms to Support Centralized ECU functions

Kevin CHAPPUIS
k.chappuis@virtualopenSystems.com

Automotive Linux Summit 2016
2016-07-13, Tokyo, Japan

Autorship and sponsorship

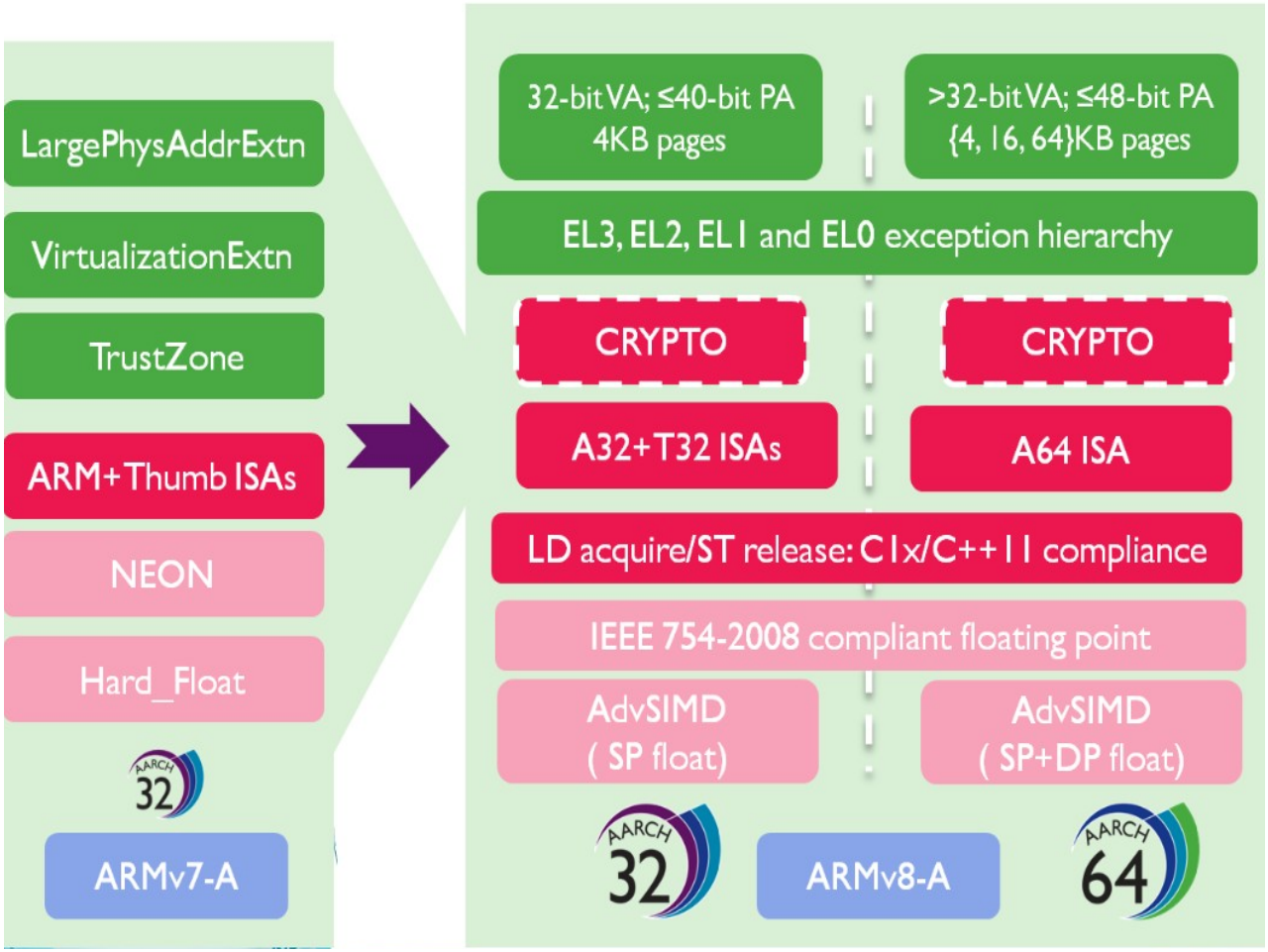
Kevin CHAPPUIS, software engineer at Virtual Open Systems (VOSYS). Skilled in ARMv7 and ARMv8 architecture, he is experienced in low level software development (e.g., boot loader, secure monitor, RTOS) on ARM multi-core heterogeneous platforms.

Virtual Open Systems is a high-tech software company active in open source virtualization solutions and custom services for complex mixed-criticality automotive, NFV networking infrastructures, consumer electronics, mobile devices and in general for embedded heterogeneous multicore systems around new generation processor architectures.

This work is done in the context of the H2020 Trusted APPs for CPS (TAPPS) project (www.tapps-project.eu).

- **ARMv8-A architecture introduction**
- Centralized ECUs integration
- ARMv8 monitor for automotive mixed-criticality systems
  - Status of the work and benchmark
- Other solutions (Hypervisor, ARMv8-R)
- Conclusion

ARM architecture evolution



(Source: ARMv8 Technology Preview – ARM)

Cortex-A57, Cortex-A53...

Renesas RCAR H3

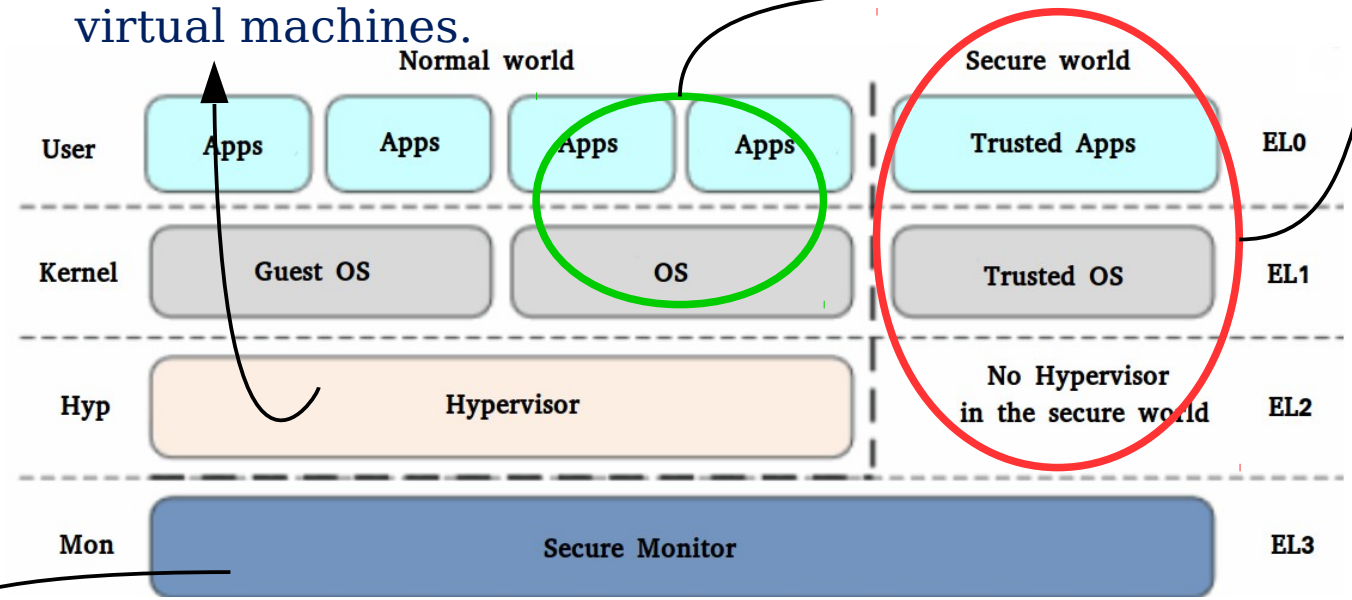
ARMv8 overall description

- **Architecture profiles:**
  - A – application / R – real-time / M - microcontroller

- **ARMv8 - AARCH64 Execution state:**
  - 31 General Purpose (GP) registers
    - 64-bit GP registers X0-X30 (32 bit access W0-W30)
    - No banking of GP register
    - Stack pointer is a specific register (one by Exception Level)
    - Program counter is not a GP registers
  - Support for Floating Point and Advanced SMID (32 registers 128-bits)
  - PSTATE register (e.g., ALU flags, exception masks)
  - System register access
    - MRS x2, sp_el3

ARMv8 exception level

- ***ARM Virtualization extensions*** address the needs of devices for the partitioning and management of complex software environments into virtual machines.
- ***Normal world*** to run concurrently another OS (e.g Linux) without impacting the secure OS.

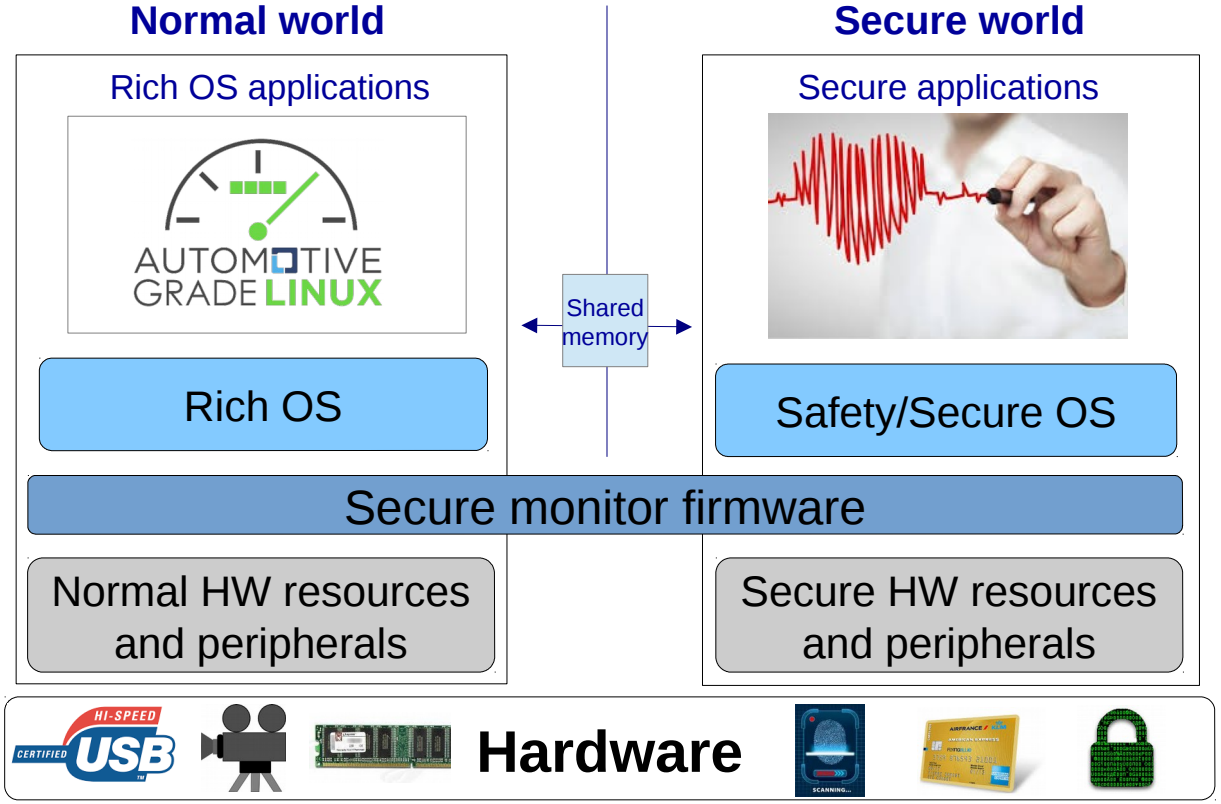


- ***Secure world*** is completely isolated (memory, devices, etc) from the Normal world by *ARM TrustZone* security extensions. Since TrustZone is implemented in hardware, it reduces the security vulnerabilities. The secure world could be used to run a secure OS to provide secure services to the OS running in the Normal world.
- ***Monitor layer*** is the highest priority level which provides a bridge between each world to allow some interactions.
- **Exception level changing through specific instructions SMC, SVC, HVC, ERET**

ARM TrustZone security extension

- TrustZone splits core into two compartments (e.g., Normal world / Secure world)
- Secure monitor firmware (EL3) is needed to support context switching between worlds

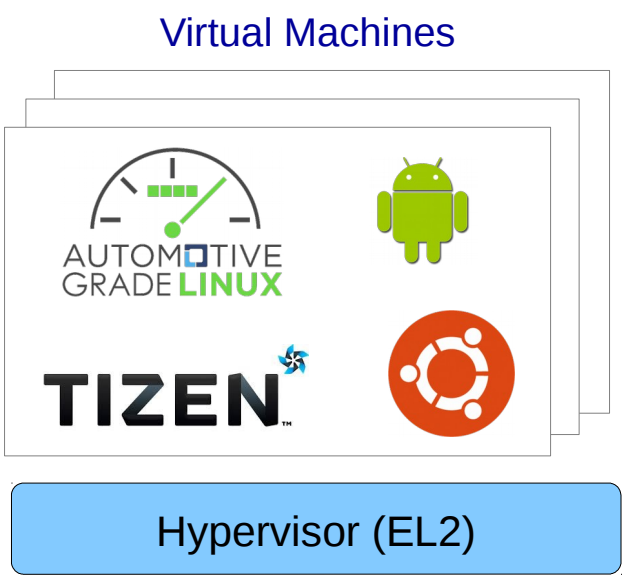
ARM TRUSTZONE
System Security



- Each compartment has access to its own MMU allowing the isolation of Secure and Normal translation tables.
- Cache has tag bits to discern content cached by either secure or normal world.
- Security information is propagated on AXI/AHB bus
- Memory/Peripheral can also be made secured
- Provide security interrupts

ARM virtualization extension

- ARMv8-A architecture includes hardware virtualization extension and Large Physical Address Extension (LPAE) to support the efficient implementation of vitual machine hypervisors:

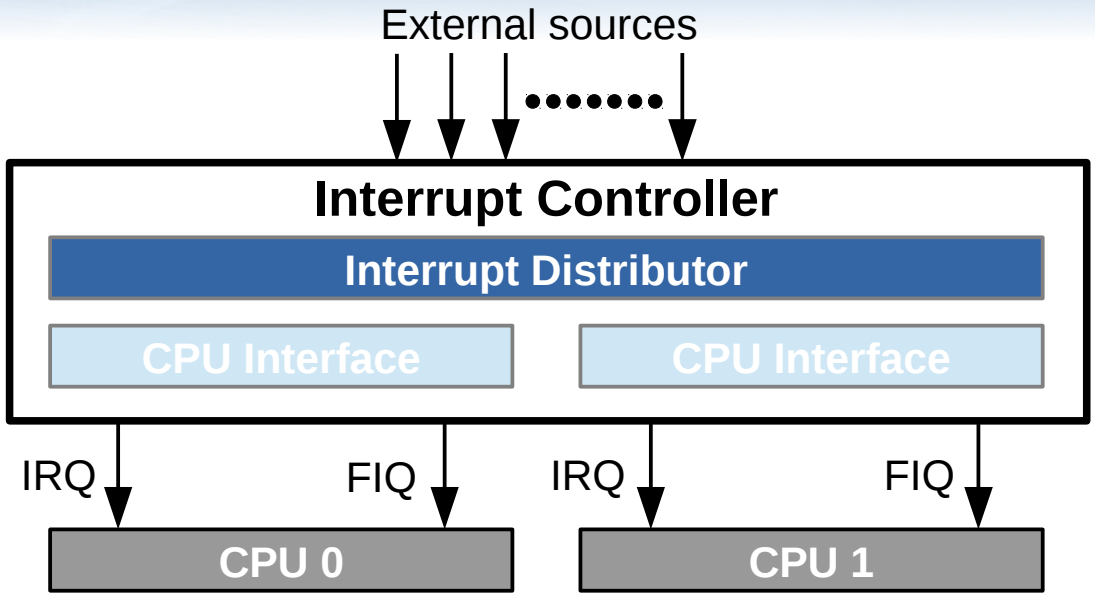


- Dedicated exception level (EL2) for hypervisor.
- Full virtualization capacity to run an OS in a virtual machine without any modification.
- Combination of hardware features to minimize the need of hypervisor intervention.

- Some hypervisors compliant with the ARM architecture
  - Linux-KVM
  - XEN

ARM interrupt management

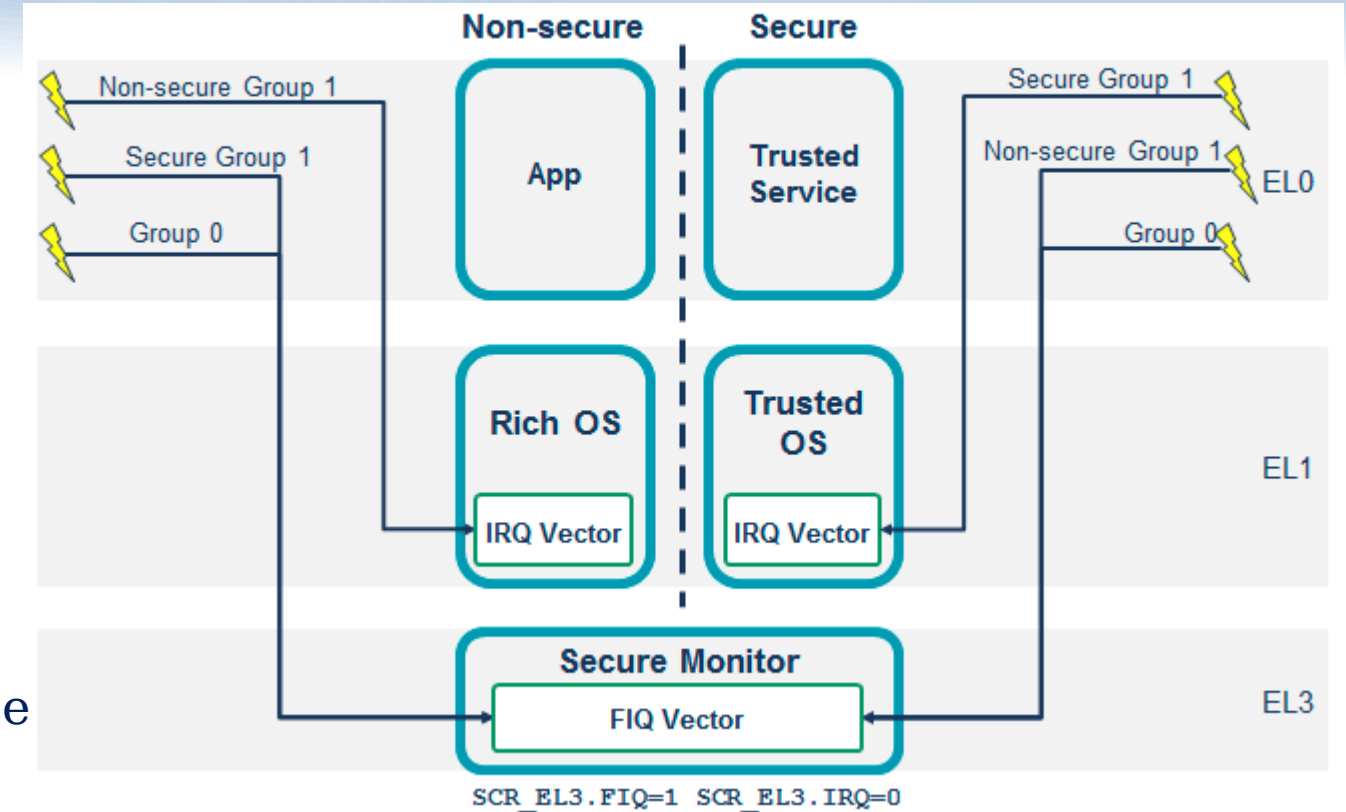
- ARM processors include two types of interrupts:
  - Fast Interrupt (FIQ) is the highest priority. Some banked registers are allocated to the FIQ handler. FIQ could be used for secure applications.
  - General Interrupt Request (IRQ)



- ARM provides a Generic Interrupt Controller (GIC) which supports routing of software generated, private and shared peripheral interrupts between cores. It is composed by:
  - ***Distributor:*** All interrupt sources are connected. It controls the type of the interrupt, priority, state, core targeted through the CPU interface.
  - ***CPU interface:*** Through this a core receives an interrupt. The CPU interface provides the abilities to mask, identify and control the state of interrupts.

GIC V3

- Support more than 8 cores
- Support messages based interrupt
- System register access
- Enhanced security model
- Introduce redistributor
- Legacy with previous GIC
- Expanded ID interrupt space

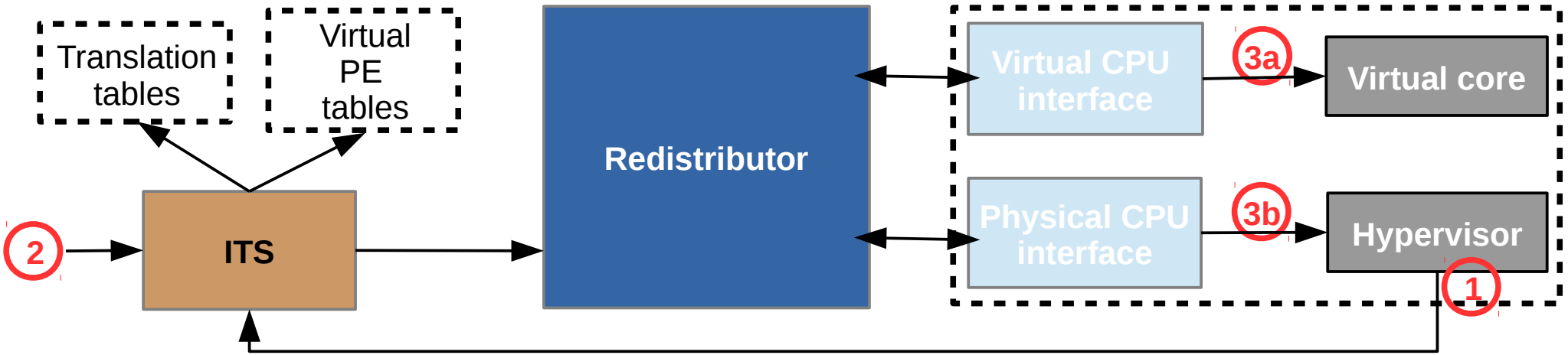


(Source: ARMv8-A: A Tour of the New GICv3 Architecture - ARM)

- GIC V3 adds a new interrupt type (Locality specific Peripheral Interrupt), which can be sent by peripheral to GIC via Interrupt Translation Service.
- ITS translates the interrupt message (Event ID / Device ID) received to forward the interrupt to the correct redistributor.

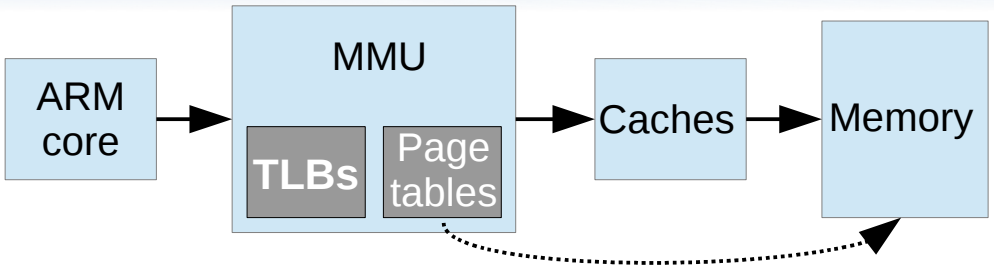
Next GIC generation: GIC V4

- GIC V4 supports the direct injection of virtual interrupts, which reduces the hypervisor mediation overhead.



1. Hypervisor executes ITS command to map interrupts
2. ITS uses event ID to retrieve translation then routes the interrupt
3a. Virtual core scheduled: Redistributor forwards to the virtual CPU interface
3b. Virtual core not scheduled: Redistributor forwards to the physical CPU interface

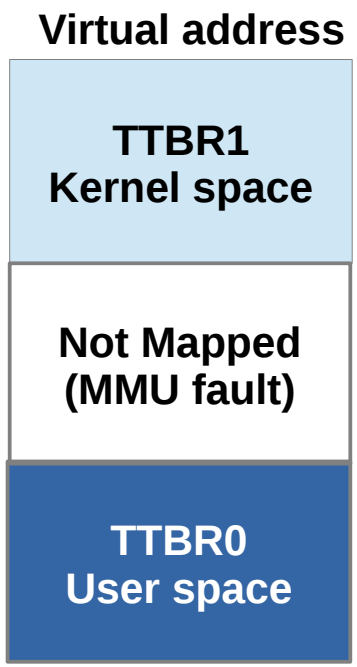
AARCH64 Memory Management Unit



*Translation Look-aside Buffers

- MMU handles translation of virtual addresses to physical addresses.
- The address translation is performed through the TLB or a table walk.

- AARCH64 supports up to 48-bits of Virtual Address
- All ELs (excepted EL1) have independent MMU configuration registers (TTBR - TCR)
- The page table supports different translation granules (e.g., 4KByte, 64KByte) configurable for each TTBR.
- Each page table requires different attributes
  - Access permissions (Read/Write - User/Privileged modes)
  - Memory types (Caching/Buffering rules, Shareable, Executable, Secure)



- ARMv8-A architecture introduction
- **Centralized ECUs integration**
- ARMv8 monitor for automotive mixed-criticality systems
  - Status of the work and benchmark
- Other solutions (Hypervisor, ARMv8-R)
- Conclusion

Towards the full autonomous driving

2015

Driving assistance with driver control requested.

Park assist remote-controlled.

(e.g., BMW Serie 7)

2017

Highway autonomous driving.

HMI optimized to ensure a better driving/entertainment transition

(e.g., Next Audi A8)

2025

Autonomous driving with a minimum driver control requested.

New car design to isolate driving and entertainment.

(e.g., Mercedes F 015)

2035

Full autonomous driving.

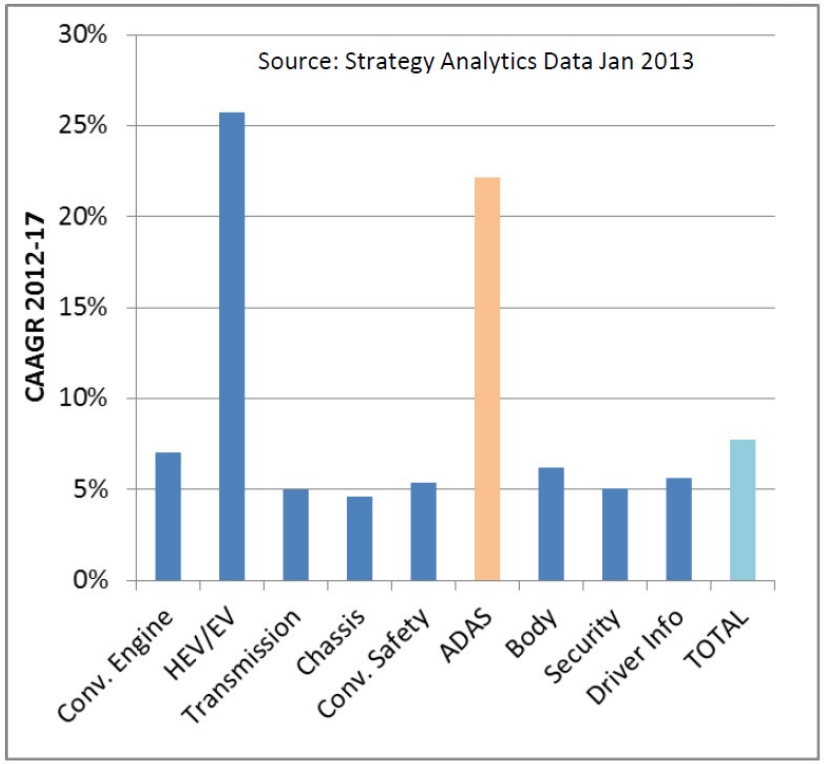
Accidents should be minimized and safety rules could be evolved.

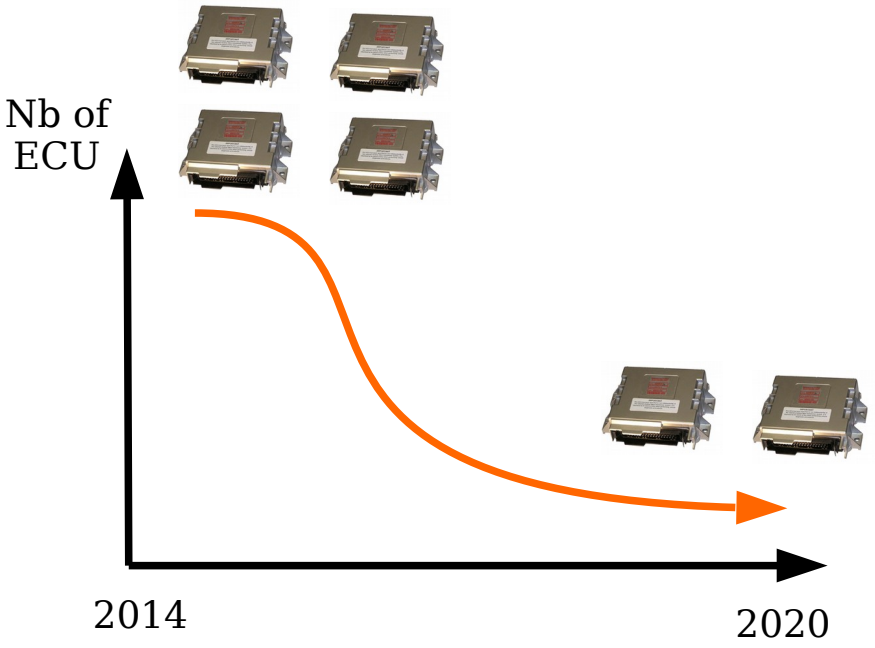
(e.g., GoogleCar)

Cars are getting smarter and always connected, mixing systems with different levels of criticality.

Growing Areas of Automotive Electronics

Growth Areas – System Types (by Strategic Analytics)





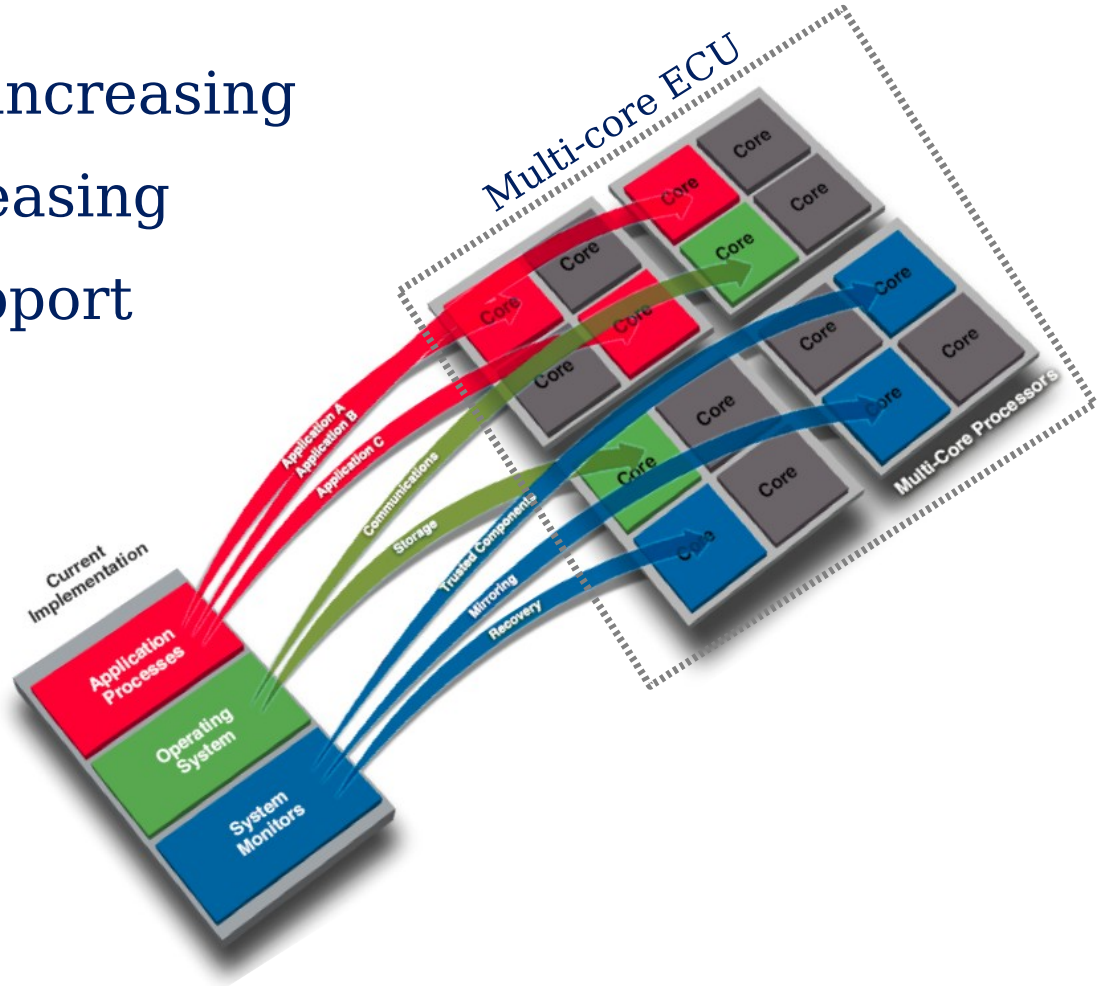
- Although, the number of functionalities are growing up, the main challenge is to decrease the number of ECUs for cost, space, weight and power consumption reasons.

Low-power and High-performance Computing for centralized ECUs

Last multi-core architectures are bringing new functionalities to the automotive platforms :

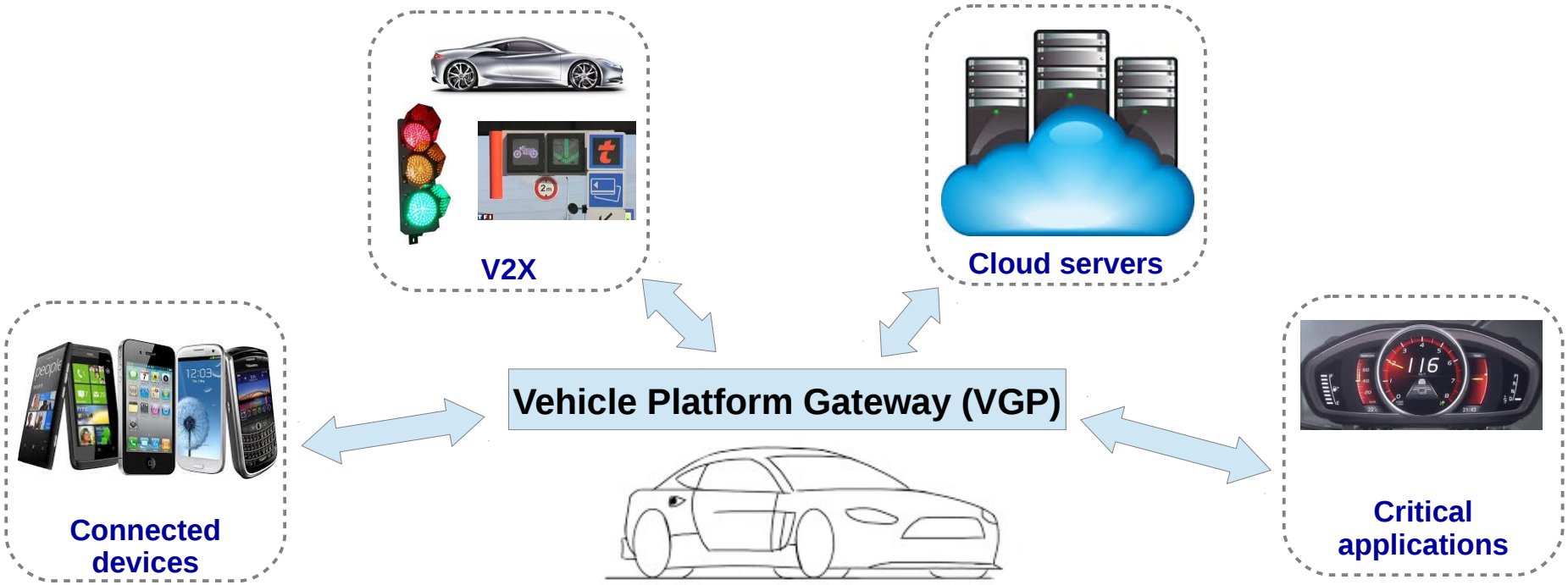
- Computing performance is increasing
- Power consumption is decreasing
- Hardware virtualization support

The main interest is to use the computing performance and hardware capabilities to embed more functionalities, having different levels of criticality, in the same ECU in order to decrease the number of hardware platforms needed in the car.



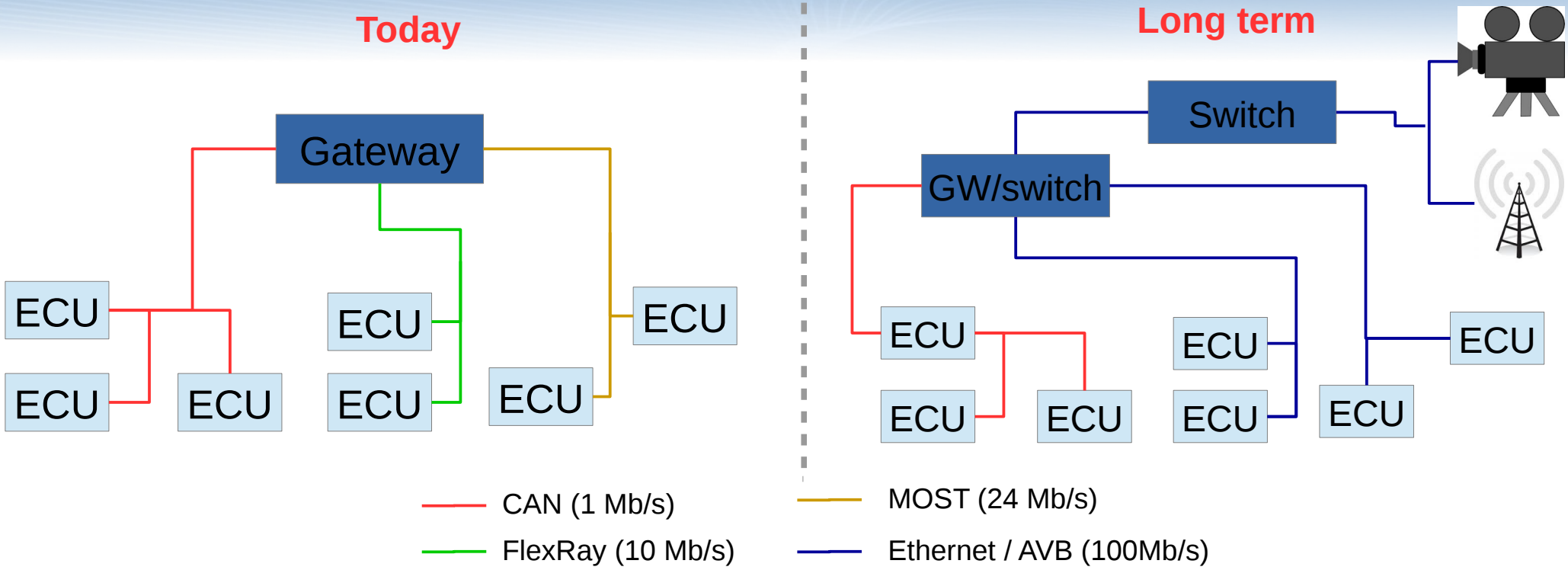
Connected-cars: Vehicle Gateway Platform

- The main challenge connected cars is the integration of information (e.g., IVI, V2X, connected devices,etc) with critical data flows:



- VGP must support interconnection with external applications while ensuring in-vehicle buses secure access to ECUs, which contains critical applications

Connected cars: From Gateway to Backbone Arch



(Source: Automotive Gateways – Bridge & Gateway from FlexRay/CAN/LIN to AVB Networks - BOSH)

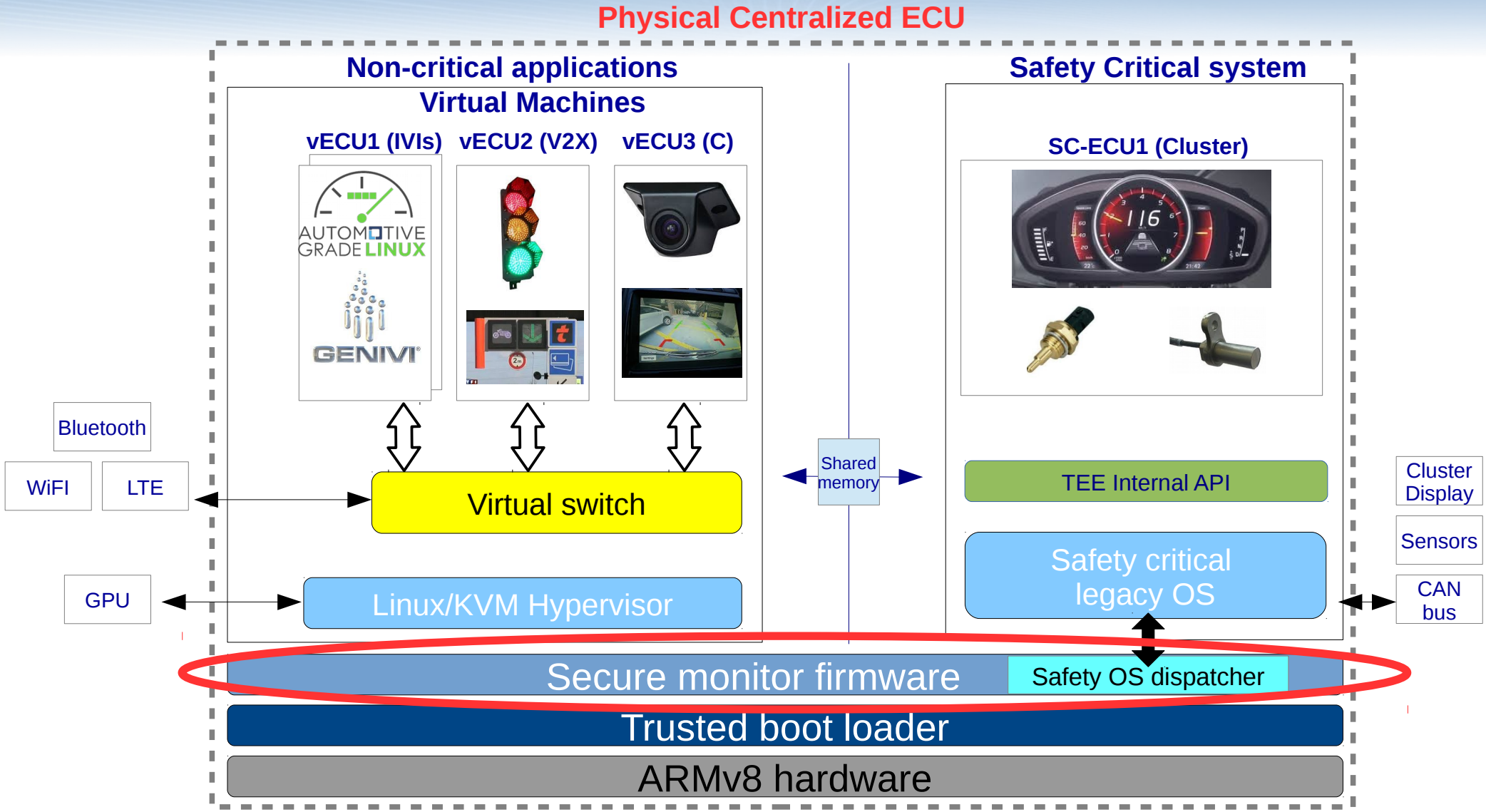
- New Connected cars' functionalities add an amount of streaming data and control signals, which cannot be handled by the current infrastructure.
- The future car will become an Ethernet networking based platform.

Centralized ECUs integration: challenges

Such a concept of the future car, brings new and unprecedented challenges to the automotive industry:

- **Multi-OS support and integration**: As functions are served by different operating systems (e.g., AUTOSAR for safety-critical functions, GenIVI Linux for automotive infotainment, Android for user apps), the multi-core system needs to be able to run multiple operating systems at the same time.
- **Efficient shared use of SoC resources**: Different functions make use of the same dedicated system resources. Examples for this include accelerated graphics from different integrated functions, or the shared use of communication channels.
- **Separation of functions and mixed-criticality support**: Safety critical functions need to be able to run alongside non-safety-critical functions without compromising their safety characteristics.

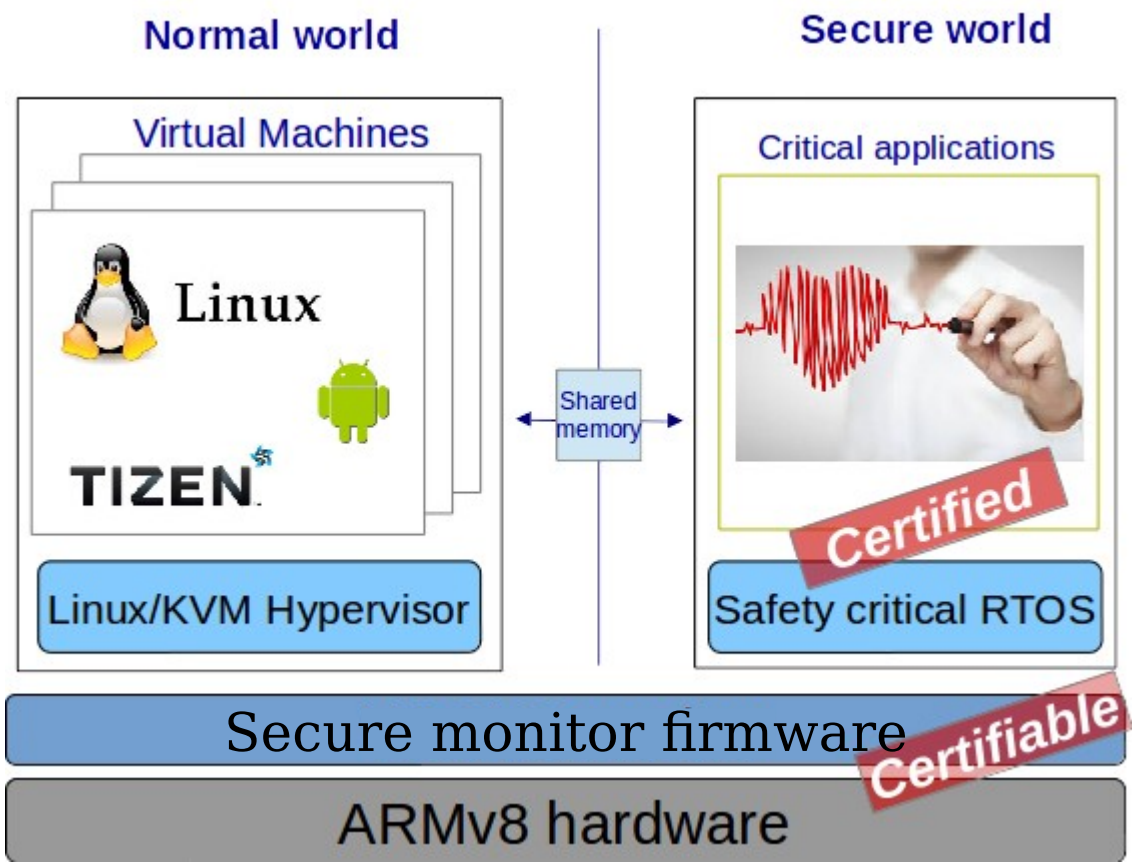
VOSYSautmost SW application: Connected cars



- ARMv8-A architecture introduction
- Centralized ECUs integration
- **ARMv8 monitor for automotive mixed-criticality systems**
  - Status of the work and benchmark
- Other solutions (Hypervisor, ARMv8-R)
- Conclusion

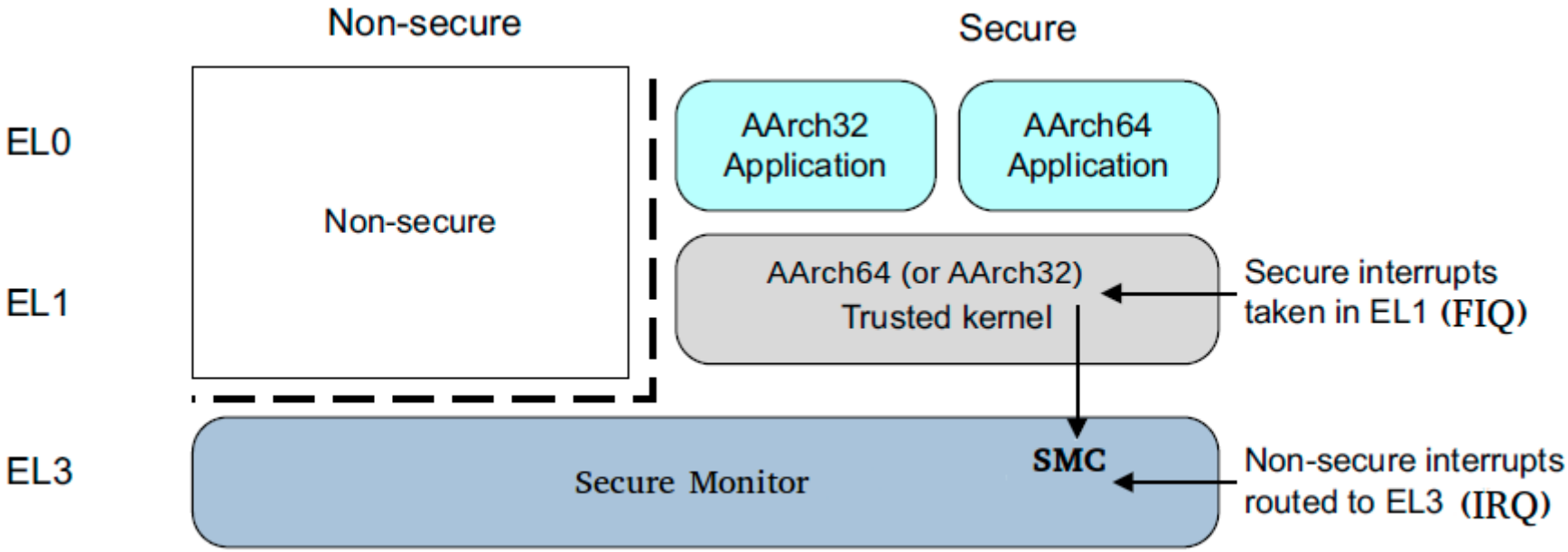
Secure Monitor Firmware description

- The Secure Monitor Firmware is a key central component, which enables the co-execution of virtualized systems along with safety critical applications on the same platform and/or core.
- Safety critical OS isolation using ARM Trustzone
- GPOS virtualization extensions (KVM) enabled
- Ability to safely exchange data between RTOS / GPOS / Vms
- High priority to the critical applications to meet timing constraints
- Tiny footprint to ease certification



Secure Monitor Firmware interaction

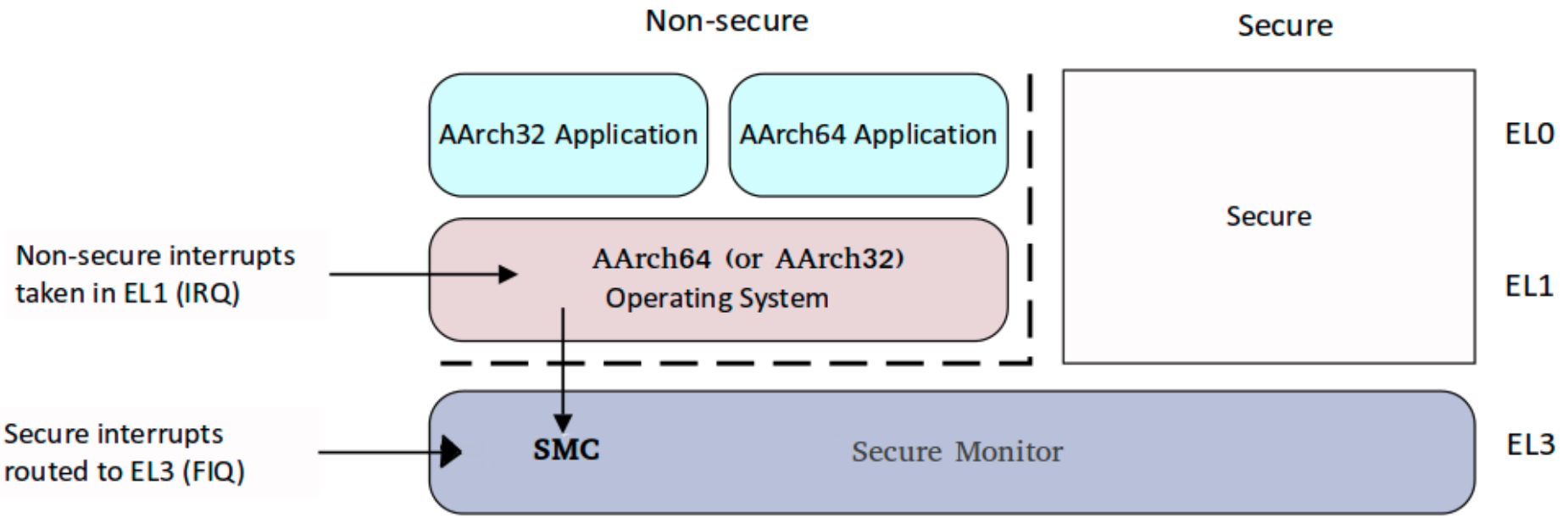
Secure world execution:



- Normal world only executes upon a request by the secure world (SMC)
- FIQ are directly handled in S-EL1 (ensuring low latency)
- IRQ vector is used to handle potential Secure world failures

Secure Monitor Firmware interaction (cntd)

Normal world execution:

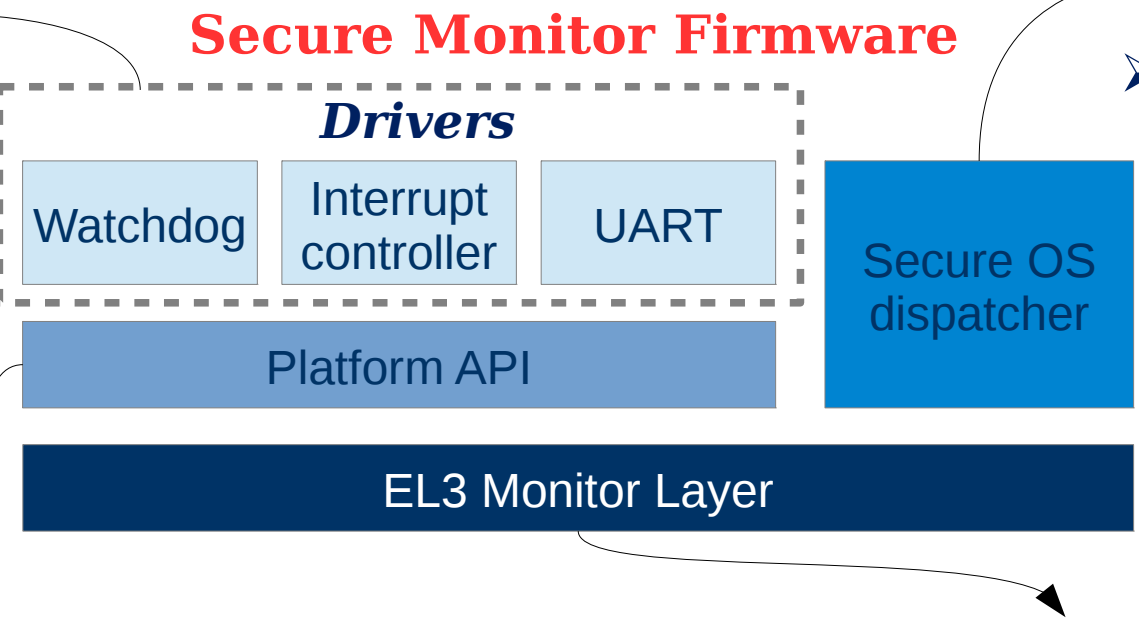


- Normal interrupts (IRQ) are directly handled in NS-EL1
- Secure interrupts (FIQ) are trapped in the Secure monitor firmware to forward it to the Safety critical OS (Normal world preemption)
- Normal world can call secure services through SMC

Secure Monitor Firmware architecture

The software architecture is split into four parts to ease:

- The support of new hardware platforms and/or Trusted OS
- Software components re-used

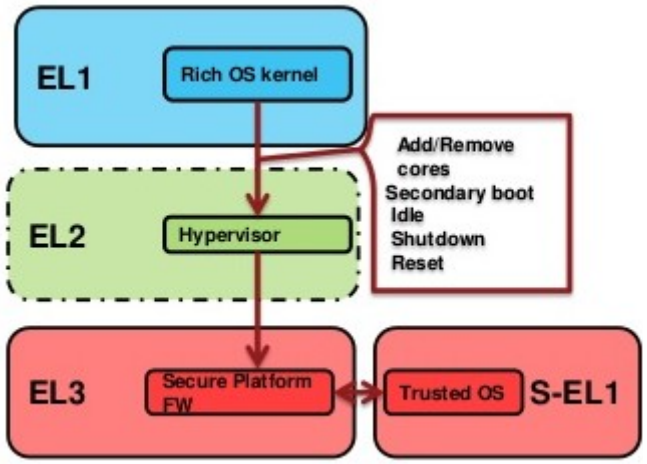


- Drivers to access peripherals requested by Monitor Layer runtime
- Interface between the Secure OS and the monitor layer, running in EL3, to dispatch SMC secure services as well as to handle interrupts forwarding.
- Platform API functions to abstract driver function calls from Monitor Layer
- Monitor layer implemented for ARMv8 architecture, which handles context switching operation and routes trapped exceptions to the right handler

Secure Monitor Firmware compliance

This firmware should be fully compliant with ARM conventions/protocols :

- SMC Calling convention (SMCCC)
  - ✔ Define a convention for SMC in ARM v7/v8 (e.g., register use for parameters and returns values, SMC type, etc)
  - ✔ Specify a partitioning of service providers to allow the vendors coexistence in the secure firmware (e.g., ARM, OEM, SIP, Trusted OS)
- Power State Coordination Interface (PSCI)
  - ✔ Define a standard interface to handle power management requests.
  - ✔ Define a protocol to allow secure firmware to arbitrate power management requests
  - ✔ Power control method in Linux AArch64 kernel

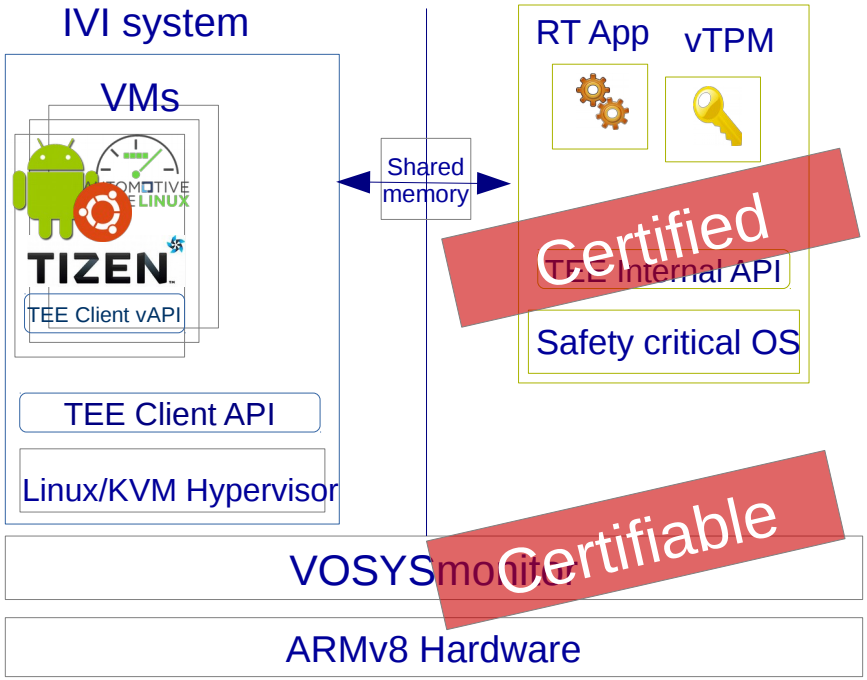


- ARMv8-A architecture introduction
- Centralized ECUs integration
- ARMv8 monitor for automotive mixed-criticality systems
  - **Status of the work and benchmark**
- Other solutions (Hypervisor, ARMv8-R)
- Conclusion

From proof of concept to VOSYSmonitor

Following the functional prototype, based on ARM Trusted Firmware*, shown during the Tokyo ALS2015, Virtual Open Systems has decided to implement from scratch its monitor layer for several reasons:

- Certify EL3 monitor layer ISO-26262 compliant (ASIL-B) to run on top safety critical applications
- Apply MISRA C:2012 code standard
- Reduce code footprint for security and certification
- Improve monitor critical paths performance (e.g., FIQ latency)
- Add world failures detection features
- Use an ISO 26262 compliant compiler



*__ATF:__ https://github.com/virtualopensystems-kchappuis/arm-trusted-firmware
__FreeRTOS:__ http://interactive.freertos.org/entries/83649935-FreeRTOS-v8-2-2-port-AARCH32-for-ARMv8-platform-ARM-FastModel-virtual-platform-and-ARM-JUNO-Developm)

VOSYSmonitor requirements

VOSYSmonitor design has been focused to meet the following requirements:

1. Enable concurrent execution on the same hardware of an RTOS (critical applications) and a GPOS (KVM virtualization)
2. Support complete RTOS resources (Memory, Peripherals, etc) isolation from GPOS illegal access
3. Complete RTOS boot in less than 60ms (VOSYSmonitor boot impact target is 1%)
4. Minimize the interrupt latency impact – RTOS interrupt forwarding time must be lower than 1us.
5. Tiny footprint to ease certification effort

ISO 26262

VOSYSmonitor environment

- This software is compiled with ARM Compiler 6
  - ✔ ARM compiler 6 is specially designed to optimize software running on ARMv8 processors. (Reduce footprint up to 30% compared to other compilers)
  - ✔ ARM compiler 6 will be ISO-26262 compliant in 2017 to enable users to apply this compiler for safety-related development without qualification activities.

- It supports several ARMv8 development platforms:
  - ✔ ARM Fast Models AEMv8A (Virtual Platform)
  - ✔ ARM JUNO Development board (2 x A57 + 4 x A53)
  - ✔ Renesas R-CAR H3 board (4 x A57 + 4 x A53) Compliant with ISO-26262 (ASIL-B)

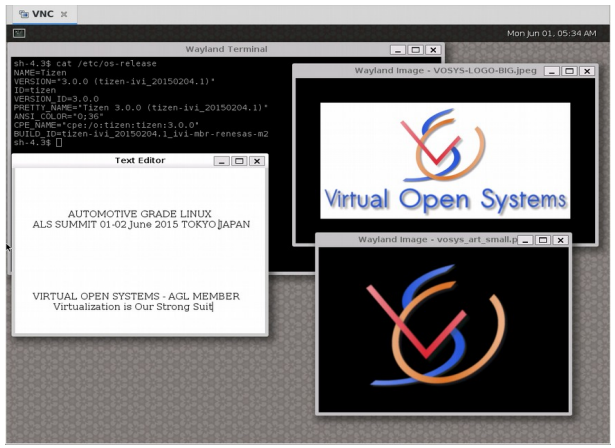
VOSYSmonitor status

A VOSYSmonitor demonstration, running on the Renesas RCAR H3 board (ARMv8 architecture), can be seen in the booth area of the Tokyo ALS 2016 :

(Quad Cortex-A57@1,5GHz + Quad Cortex-A53@1,2GHz)

- RTOS/Linux-KVM co-execution on the same processor
- Safety critical OS isolation using Trustzone
- PSCI service

- Hardware exception mechanisms to induce context switch
- Secure OS monitoring to recover failures
- SMC Framework



VOSYSmonitor performance measurements

Different performance measurements have been performed on the VOSYSmonitor demonstration presented at the ALS2016

- VOSYSmonitor Setup time test
- Interrupt latency tests, which aim to measure the interrupt latency overhead added by VOSYSmonitor
- SMC service latency to measure the response time to forward a secure service request

Note: All performance tests have been performed on the ARM JUNO Development board (CPU Frequency 700MHz).

VOSYSmonitor setup time

Requirement: Complete RTOS boot in less than 60ms - VOSYSmonitor boot impact target is less than 1% (e.g.,< 600us)

Test case: Use the Performance Monitoring Unit (PMU) to have a very detailed view of latency in terms of clock cycles counter. Start the PMU at the VOSYSmonitor entrypoint and stop it just before jumping to the Secure OS entrypoint.

	VOSYSmonitor setup
PMU Clock cycles	7762
Time (us) JUNO board Frequency 700MHz	11,09 us
Time (us) RCAR-H3 board Frequency 1,5GHz (Expected)	5,17 us

VOSYSmonitor setup includes:

- ARM EL3 initialization
- Platform peripheral initialization (e.g., Interrupt controller, etc)
- VOSYSmonitor initialization (e.g., SMC service, Secure Timer, etc)

VOSYSmonitor interrupt latency

Requirement: Minimize the interrupt latency impact – RTOS interrupt forwarding time must be lower than 1us.

Test case: Set a timer (free-running mode) interrupt in FreeRTOS. When the FreeRTOS fiq handler is reached, the timer value is compared with the trigger value in order to measure the time consumed before handling the interrupt in FreeRTOS.

	JUNO board Frequency 700MHz		**RCAR-H3 board Frequency 1,5GHz (Expected)**
	FreeRTOS standalone mode	VOSYSmonitor + FreeRTOS	VOSYSmonitor + FreeRTOS
Average	228 ns	780 ns	488 ns
Min	160 ns	720 ns	423 ns
Max	320 ns	1060 ns	668 ns

VOSYSmonitor SMC service latency

Test case: Use the Performance Monitoring Unit (PMU) to have a very detailed view of latency in terms of clock cycles counter. Start the PMU when an SMC service is triggered in VOSYSmonitor and stop it just before jumping to the Secure OS service handler.

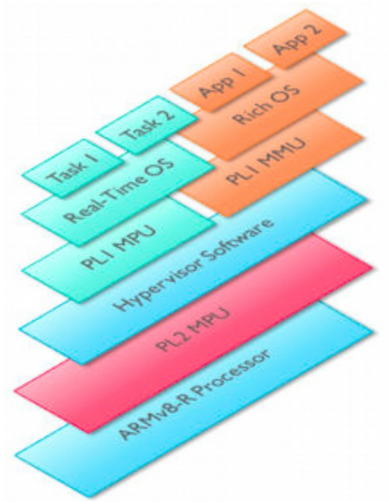
	SMC service unknown	SMC service supported
PMU Clock cycles	46	545
Time (ns) JUNO board Frequency 700MHz	66 ns	778 ns
Time (ns) RCAR-H3 board Frequency 1,5GHz (Expected)	30 ns	363 ns

VOSYSmonitor proposes a feature to monitor potential Secure world failure based on the ARM Secure timer which adds, if used, an overhead of 160 cycles.

- ARMv8-A architecture introduction
- Centralized ECUs integration
- ARMv8 monitor for automotive mixed-criticality systems
  - Status of the work and benchmark
- **Other solutions (Hypervisor, ARMv8-R)**
- Conclusion

ARMv8-M / ARMv8-R architecture

ARMv8-R

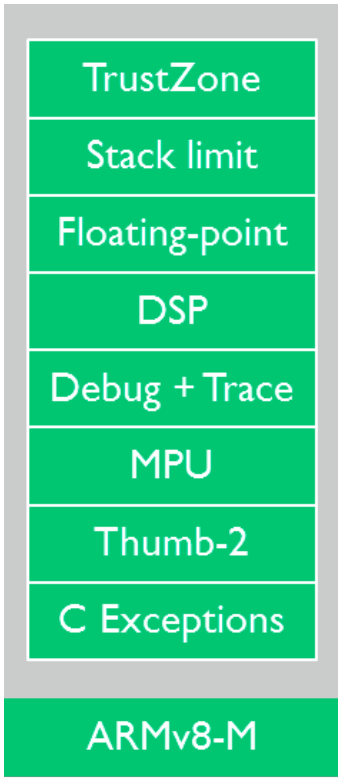


- 32-bit real time processor
- GIC registers support
- VMSA support for guest OS running at EL1/EL0

- Hypervisor level to handle events with real time determinism
- Two stage of memory protection
- Rich OS guest support along with RTOS guest, while ensuring real-time responsiveness.

ARMv8-M

- 32-bit processor optimized for microcontroller applications



- ARM Trustzone technology support
- Code isolation with Memory Protection Unit
- Support only Thumb instruction for code density optimization
- Deterministic real time interrupt response

Hypervisor solution

Other approaches, which aim to integrate a safety critical OS with non-critical systems, use virtualization to enable support for mixed criticality. Virtualization benefits are:

- Hardware isolation of virtual machines
- Supports for the execution of many OSes concurrently
- Virtualization is a well-known and mature technology
- Examples: XEN Automotive Hypervisor and QNX hypervisor
- But..

RTOS isolation: TrustZone benefit

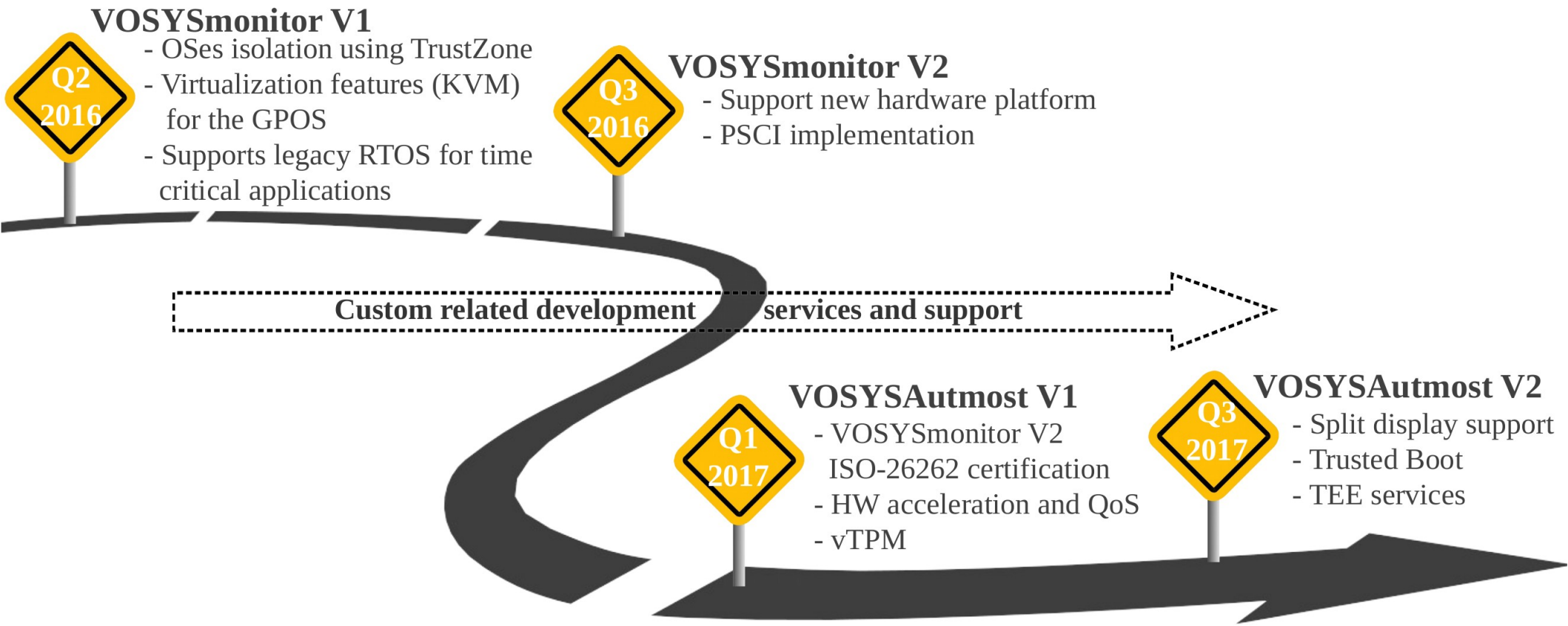
Virtualization is cheap and provides nice features for automotive, but it could have important security problems:

- XEN vulnerability: CVE-2016-5242, allows guest OS users to cause a denial of service (host OS crash).

- KVM vulnerability: CVE-2016-4440, allows guest OS users to obtain direct access to the host OS and possibly execute code on the host.

- ARMv8-A architecture introduction
- Centralized ECUs integration
- ARMv8 monitor for automotive mixed-criticality systems
  - Status of the work and benchmark
- Other solutions comparison (Hypervisor, ARMv8-R)
- **Conclusion**

VOSYSmonitor Roadmap to VOSYSAutmost



Conclusion

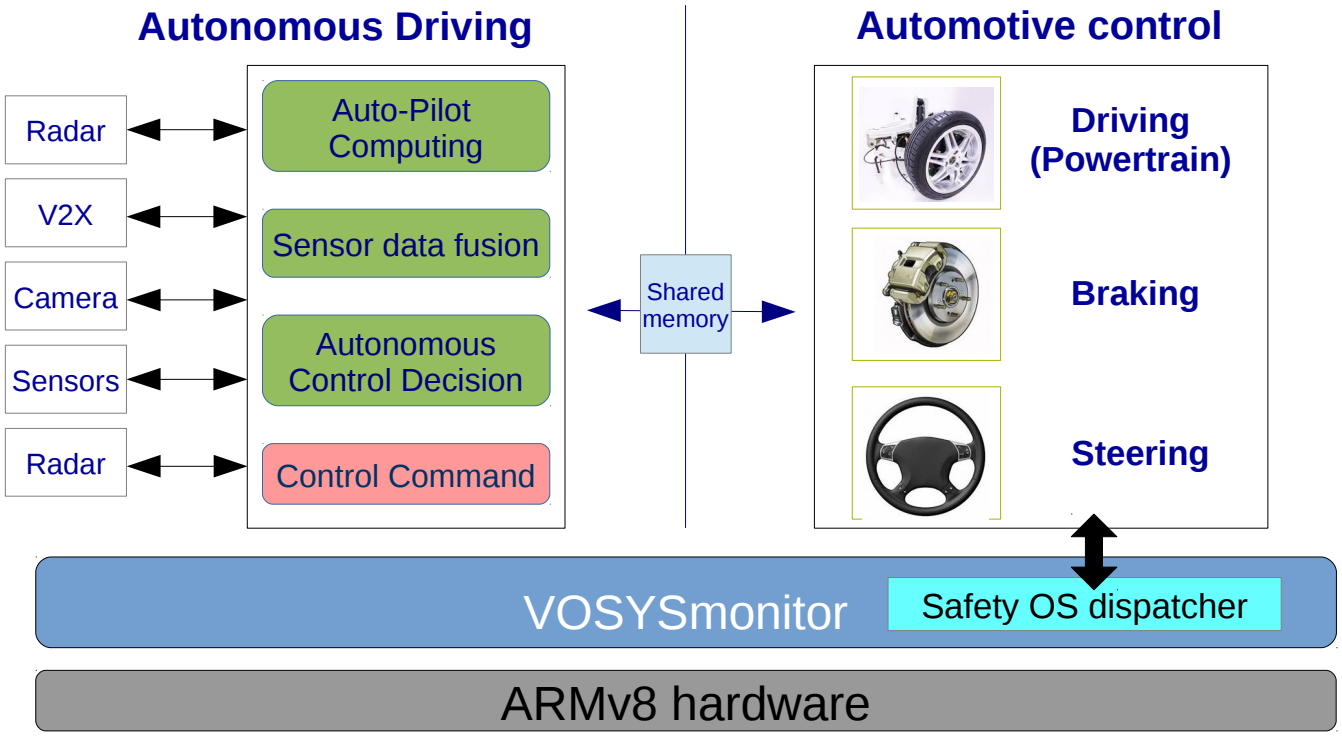
VOSYSmonitor is a low level software layer for mixed-criticality automotive systems on ARMv8 platforms:

- Supports the execution of multiple IVI guests concurrently
- Executes a safety critical OS in a protected environment with full control of the system
- Tiny foot print to ease certification process
- High priority to the critical applications to meet timing constraints

VOSYSmonitor: a flexible automotive software layer

VOSYSmonitor can be adapted to all automotive systems mixing different levels of criticality in order to centralize software functionalities on a common ECU.

- ADAS application example



VOSYSmonitor allows to run concurrently the Control decision system as well as the Automotive control system on the same hardware platform

Thank You

- Demo showcased at ALS 2016 Virtual Open Systems booth in Tokyo

- A video of the demo is available on our website:
  - *http://www.virtualopensystems.com/en/solutions/demos/vosysmonitor-als2016*

contact@virtualopensystems.com

Virtual Open Systems