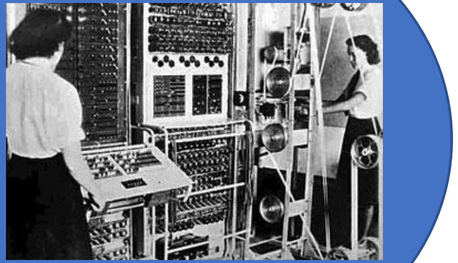
WMC

MPSoCs for Mixed-Criticality Systems: Challenges and Opportunities

Mohamed Hassan

1943

Colossus



1981

IBM's Acorn

1989

NEC's UltaLite

2000s

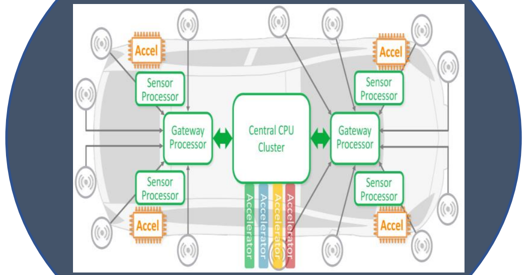
Smart Phones

2010s

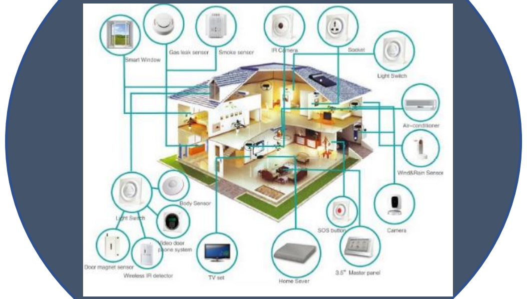
Wearables

Now-Near

Automotive



IoT/Smart Homes



Towards Mixed Criticality

1943

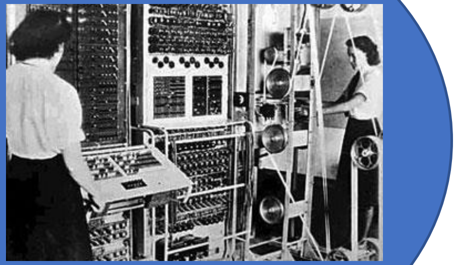
1981

1989

2000s

2010s

Now-Near



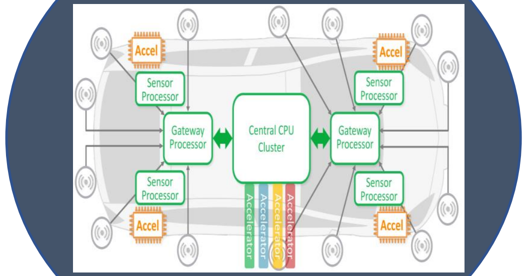
Colossus

IBM's Acorn

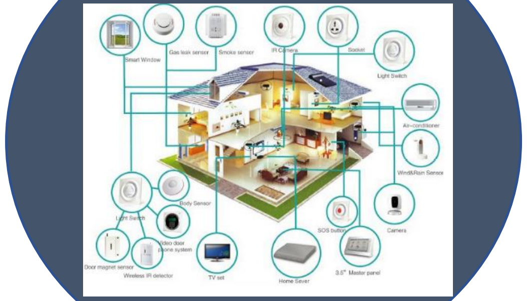
NEC's UltaLite

Smart Phones

Wearables

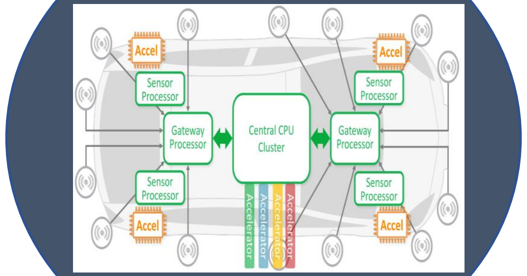


Automotive



IoT/Smart Homes

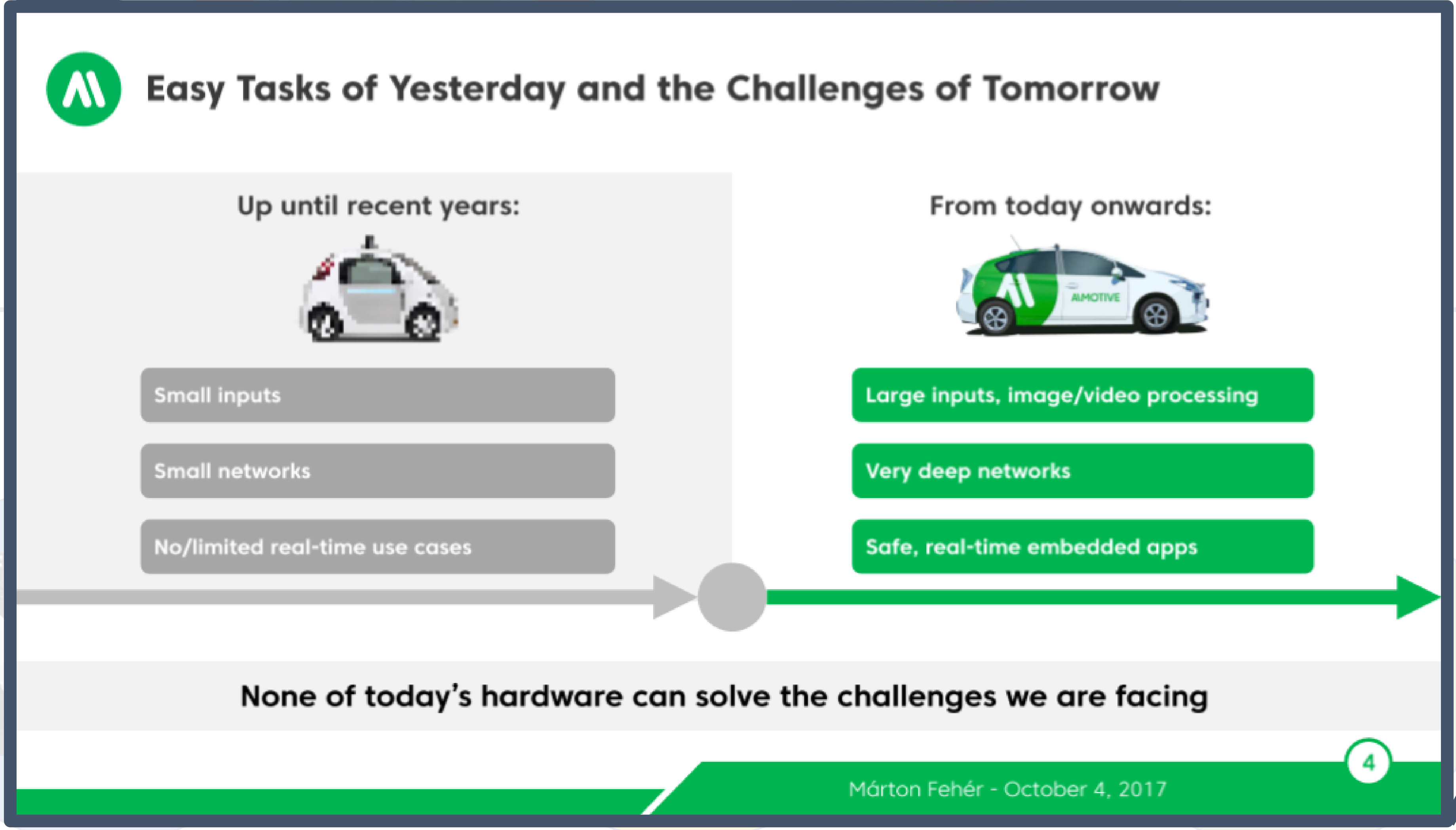
1



Automotive

Now-Near

oT/Smart Homes



Easy Tasks of Yesterday and the Challenges of Tomorrow

Up until recent years:

- Small inputs
- Small networks
- No/limited real-time use cases

From today onwards:

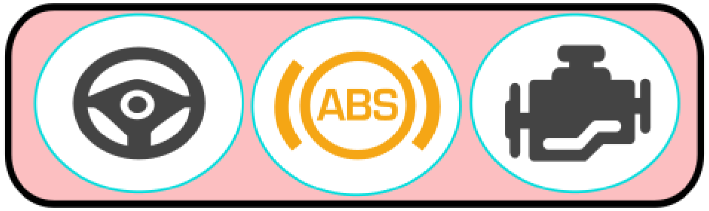
- Large inputs, image/video processing
- Very deep networks
- Safe, real-time embedded apps

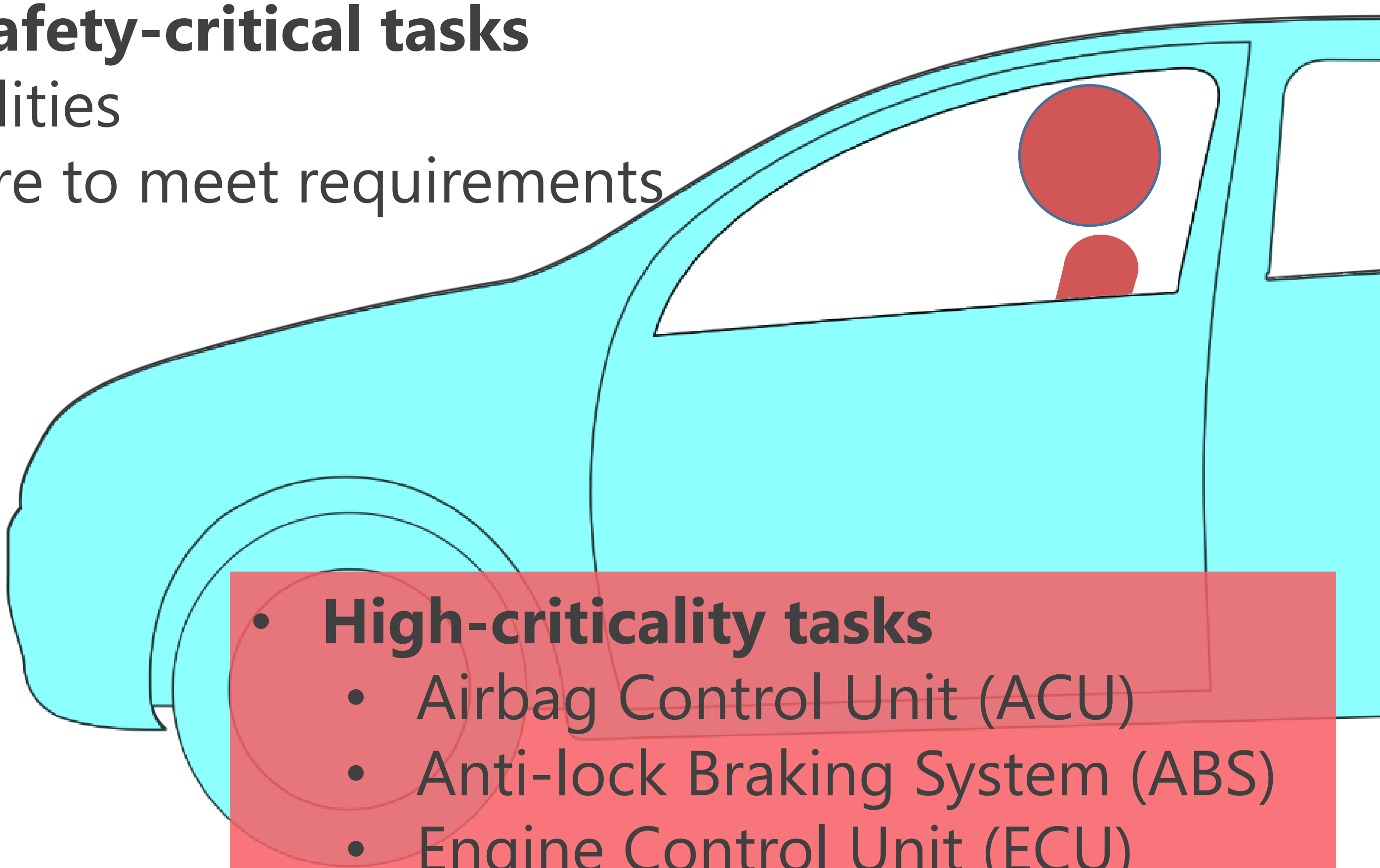
None of today's hardware can solve the challenges we are facing

4

Márton Fehér - October 4, 2017

- **No longer solely hosting isolated safety-critical tasks**
  - Execute tasks with different criticalities
  - Criticality α consequences of failure to meet requirements



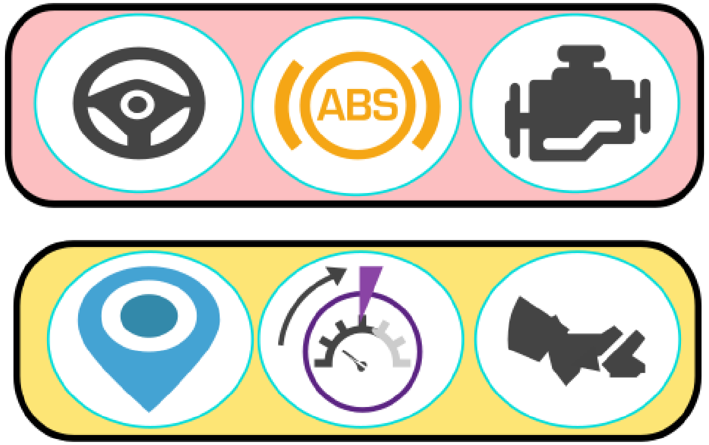


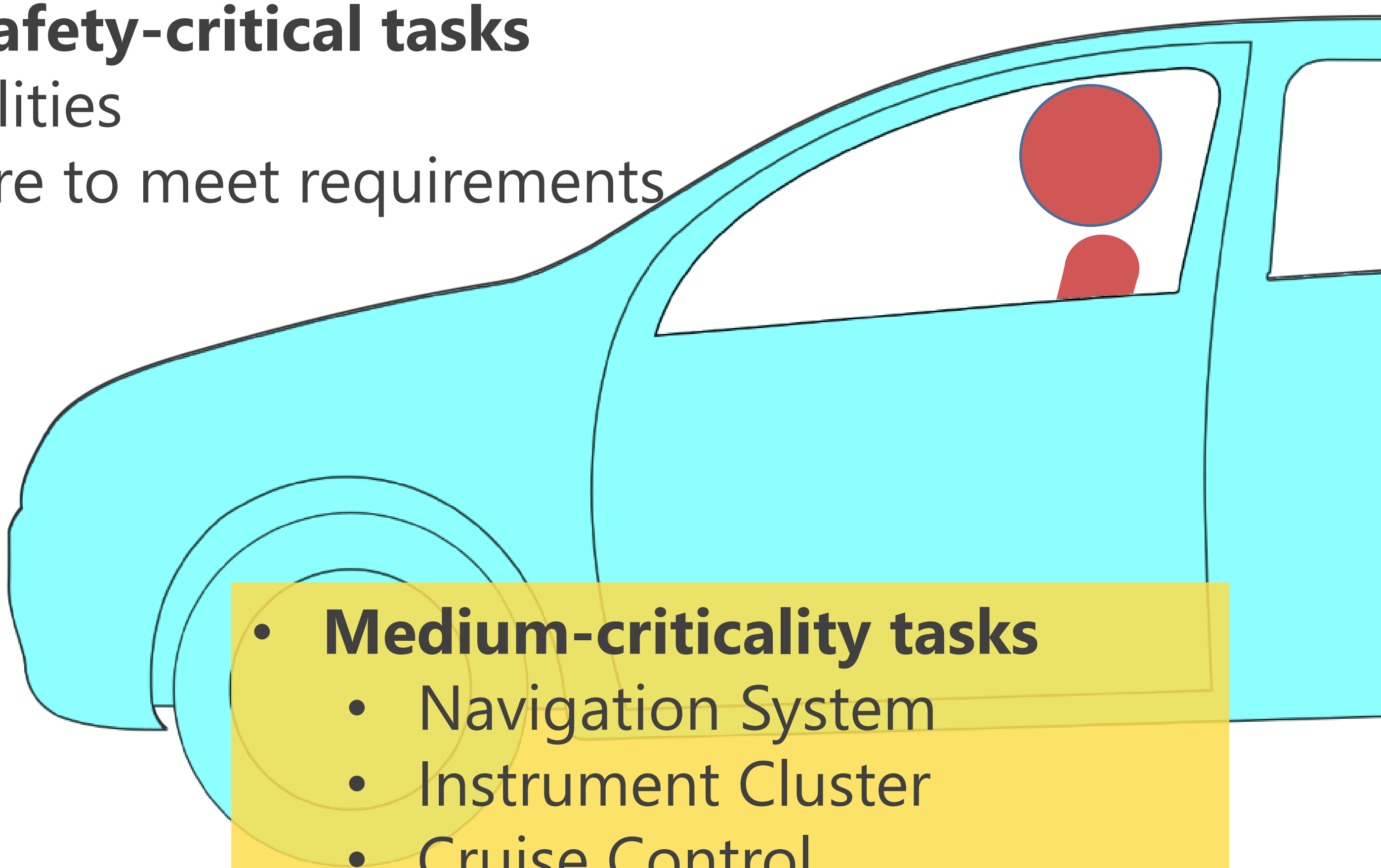
- **High-criticality tasks**
  - Airbag Control Unit (ACU)
  - Anti-lock Braking System (ABS)
  - Engine Control Unit (ECU)

Mixed Criticality Systems

MOTIVATION

- **No longer solely hosting isolated safety-critical tasks**
  - Execute tasks with different criticalities
  - Criticality α consequences of failure to meet requirements





- **Medium-criticality tasks**
  - Navigation System
  - Instrument Cluster
  - Cruise Control

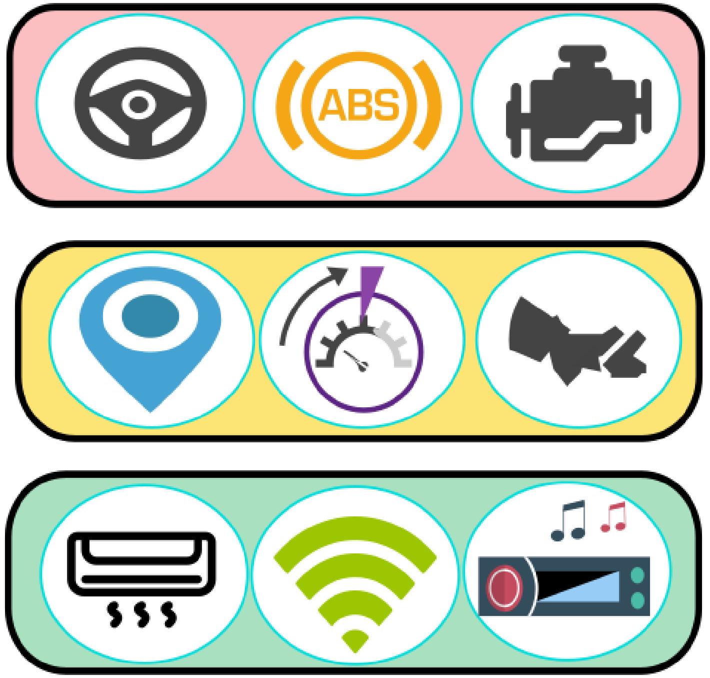
Mixed Criticality Systems

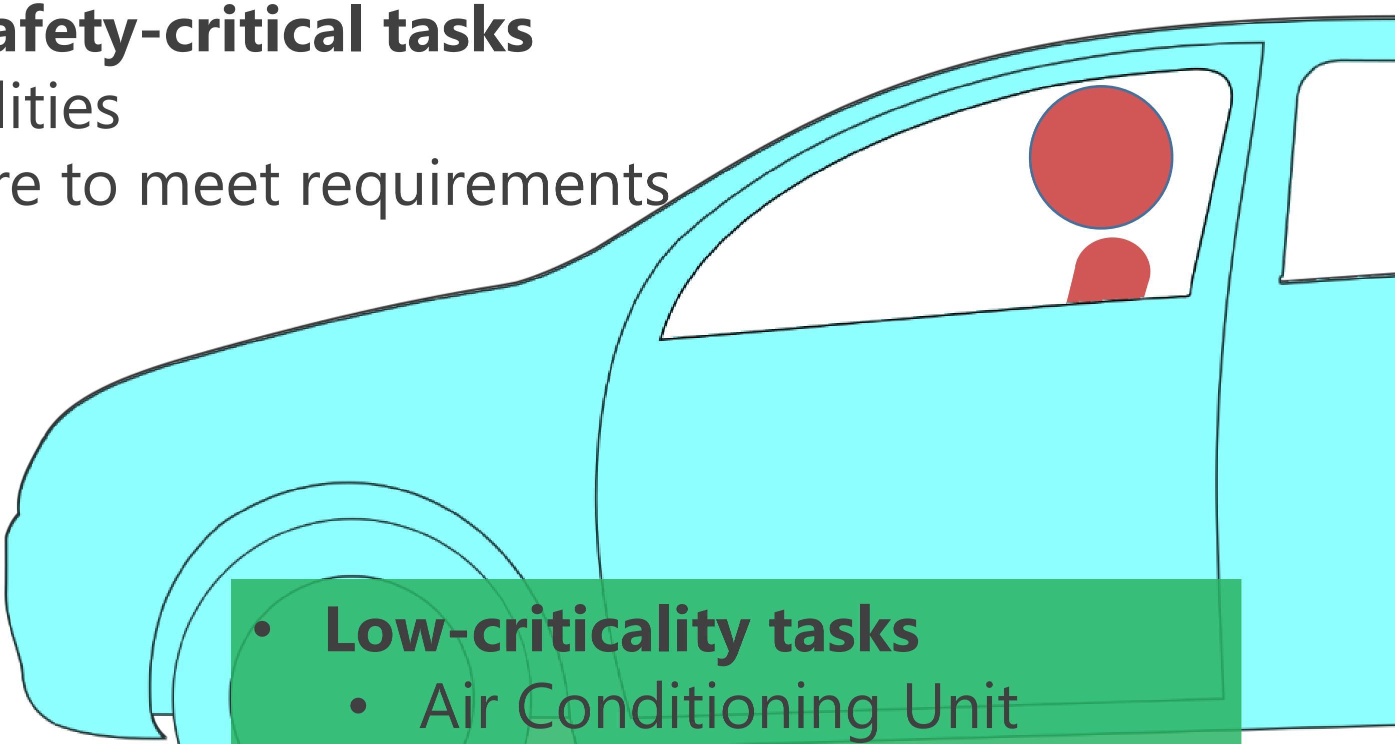
MOTIVATION

Mixed Criticality Systems

MOTIVATION

- **No longer solely hosting isolated safety-critical tasks**
  - Execute tasks with different criticalities
  - Criticality α consequences of failure to meet requirements

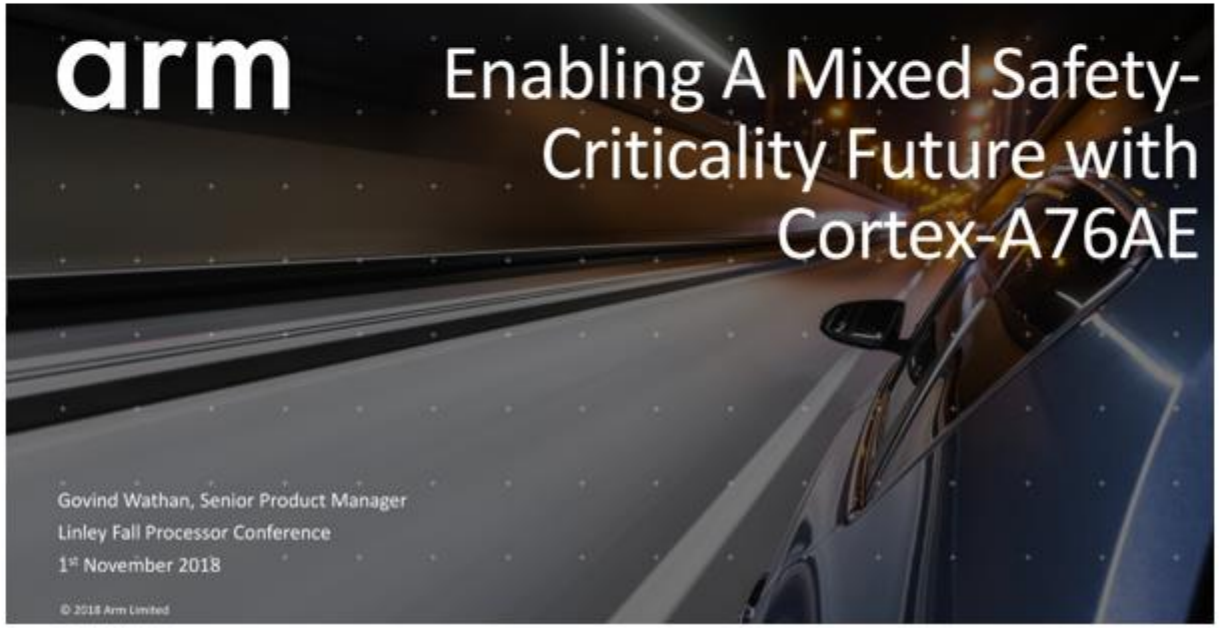




- **Low-criticality tasks**
  - Air Conditioning Unit
  - Connectivity Box
  - Infotainment Unit

Mixed Criticality Systems

MOTIVATION



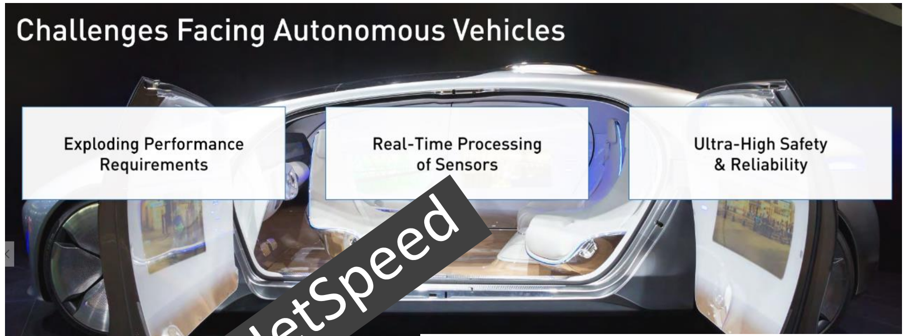
arm

Enabling A Mixed Safety-Criticality Future with Cortex-A76AE

Govind Wathan, Senior Product Manager
Linley Fall Processor Conference
1st November 2018

© 2018 Arm Limited

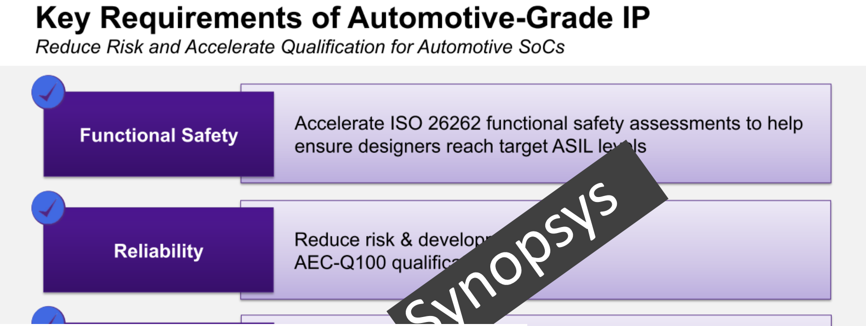
Increased need for performance and mixed criticality as we move from assisted to autonomous driving systems



Challenges Facing Autonomous Vehicles

- Exploding Performance Requirements
- Real-Time Processing of Sensors
- Ultra-High Safety & Reliability

NetSpeed



Key Requirements of Automotive-Grade IP

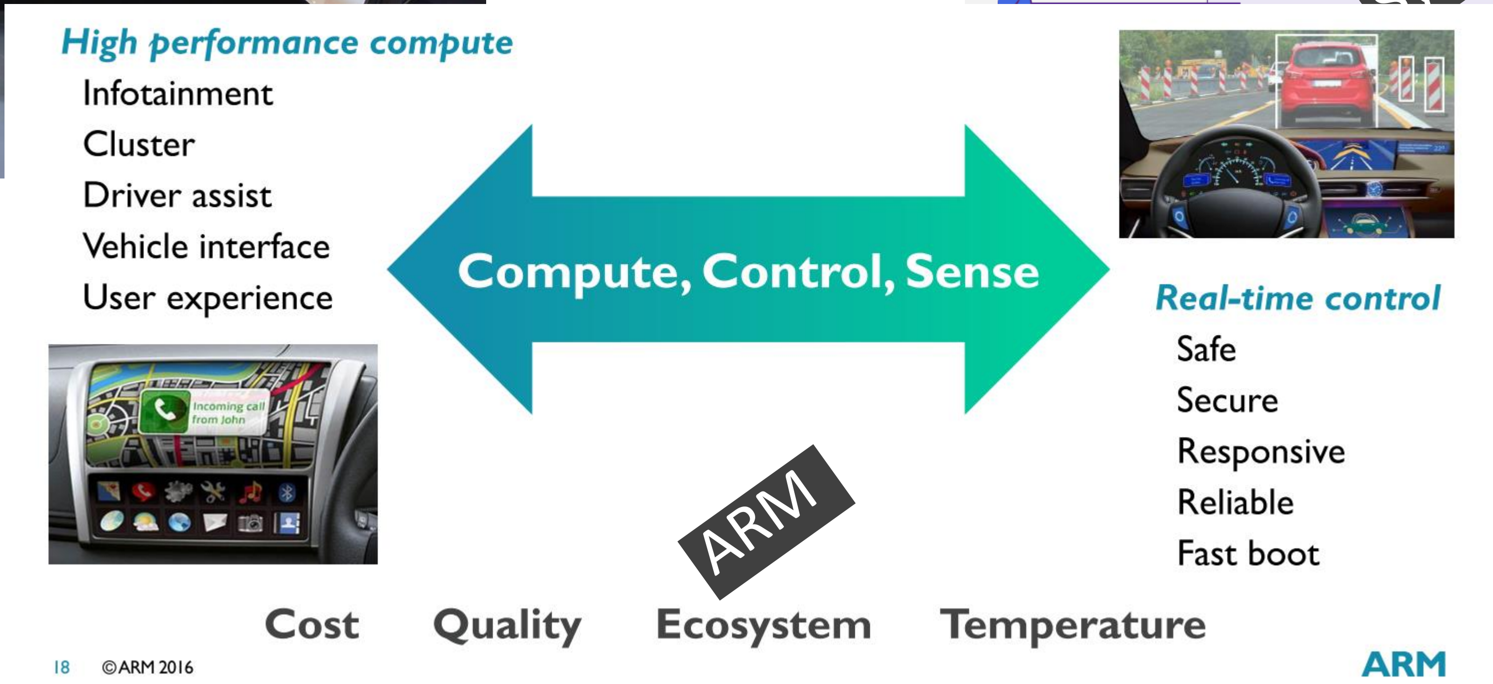
Reduce Risk and Accelerate Qualification for Automotive SoCs

Functional Safety Accelerate ISO 26262 functional safety assessments to help ensure designers reach target ASIL levels

Reliability Reduce risk & develop AEC-Q100 qualific

for automotive applications

Synopsys



High performance compute

- Infotainment
- Cluster
- Driver assist
- Vehicle interface
- User experience

Compute, Control, Sense

Real-time control

- Safe
- Secure
- Responsive
- Reliable
- Fast boot

Cost **Quality** **Ecosystem** **Temperature**

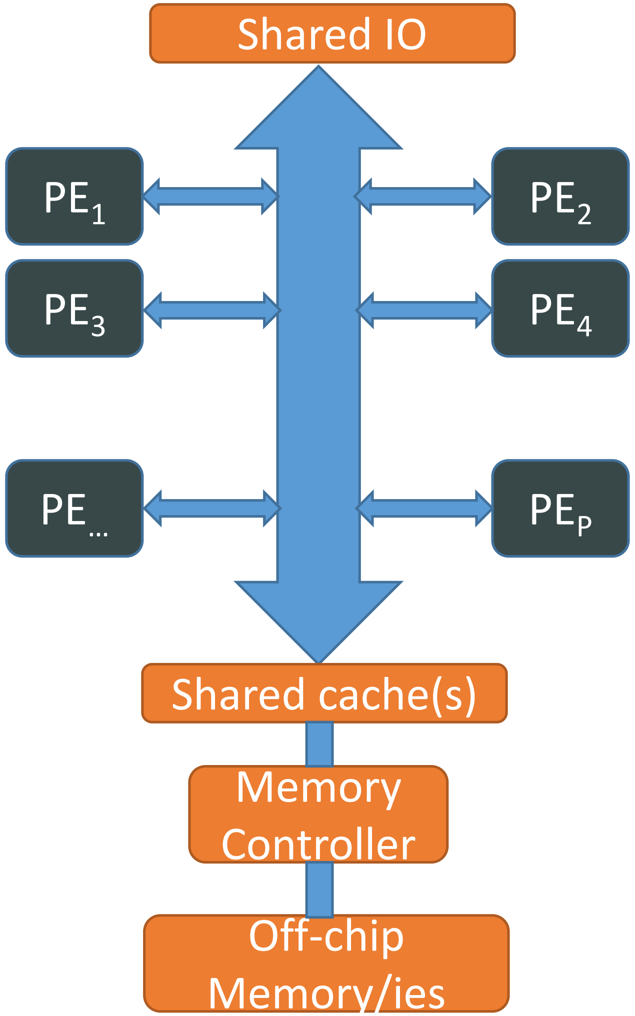
18 ©ARM 2016

ARM

ARM

Mixed Criticality Systems

MOTIVATION

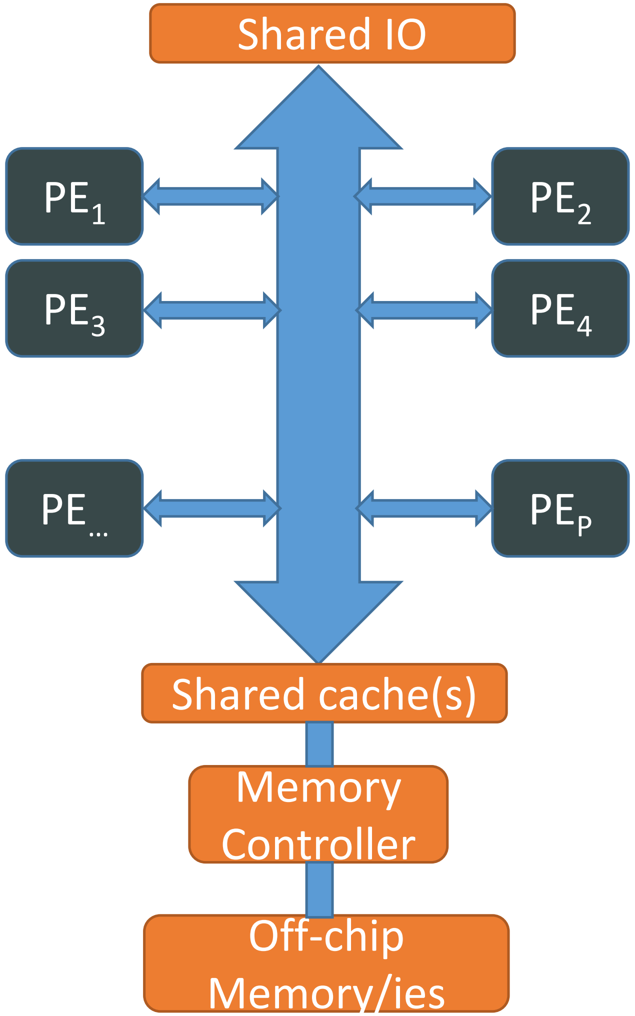


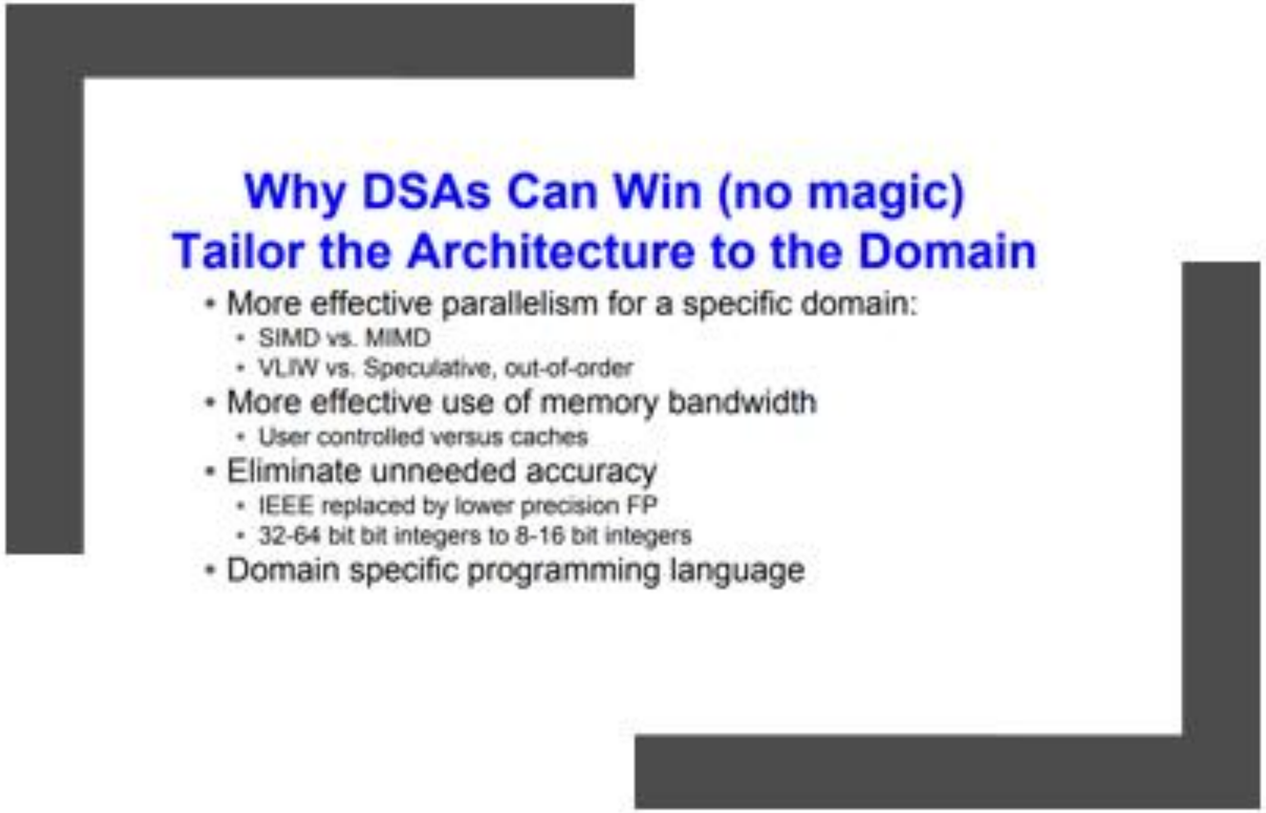
Why MPSoCs?

- Low cost
- High performance
- Energy Efficiency
- Low time-to-market (3rd party IPs)
- Simplicity and Modularity

MPSoCs

MPSoCs





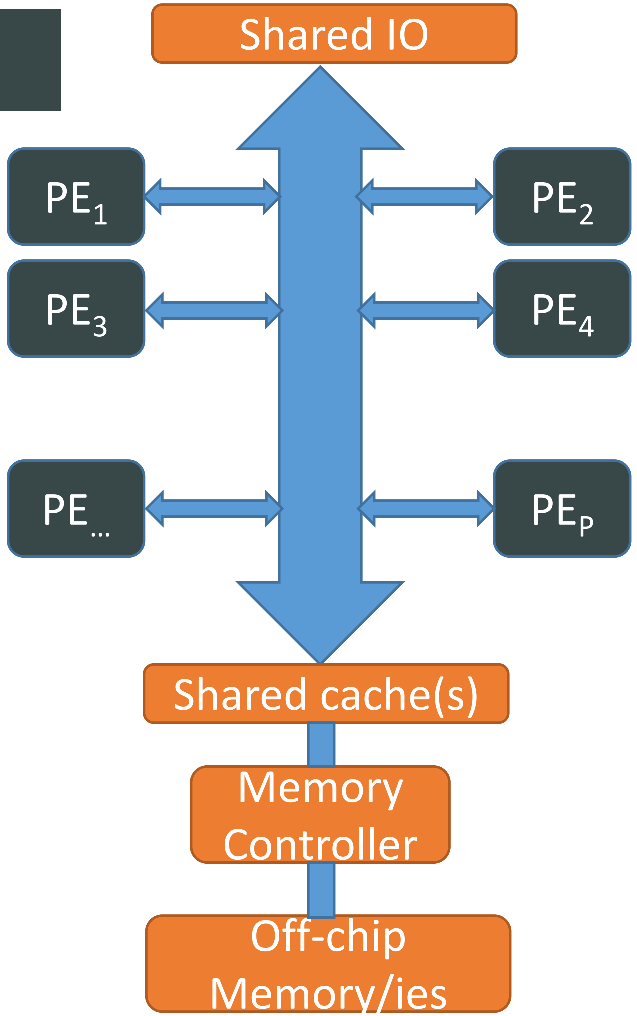
Why DSAs Can Win (no magic)
Tailor the Architecture to the Domain

- More effective parallelism for a specific domain:
  - SIMD vs. MIMD
  - VLIW vs. Speculative, out-of-order
- More effective use of memory bandwidth
  - User controlled versus caches
- Eliminate unneeded accuracy
  - IEEE replaced by lower precision FP
  - 32-64 bit bit integers to 8-16 bit integers
- Domain specific programming language

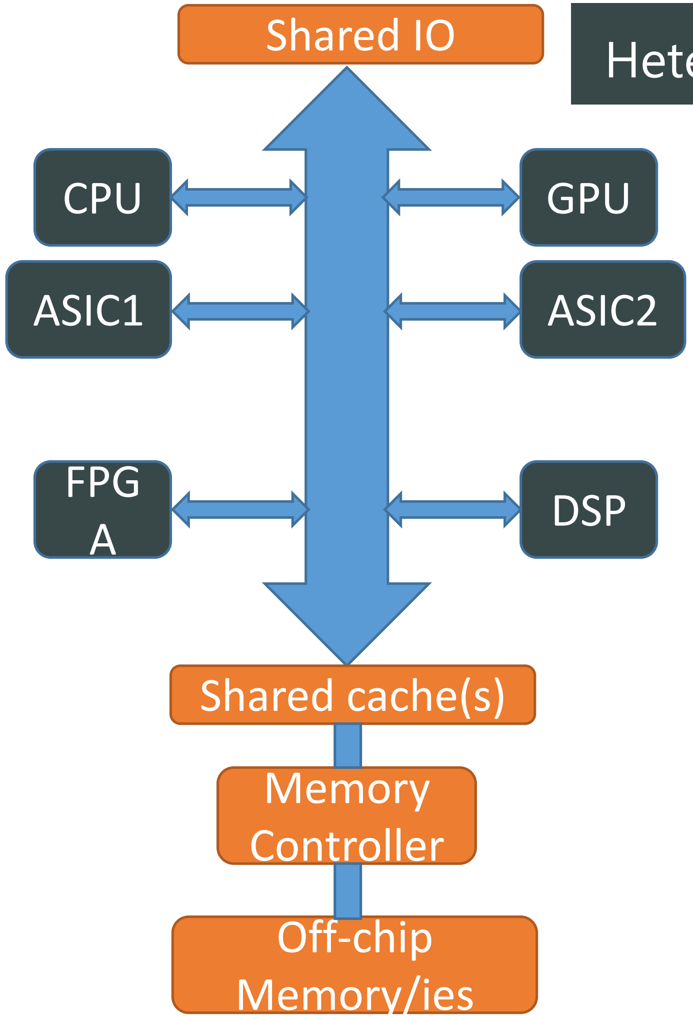
Hennessy & Patterson, Turing Lecture,
A New Golden Age for Computer Architecture

MOTIVATION

MPSoCs



Heterogenous MPSoCs



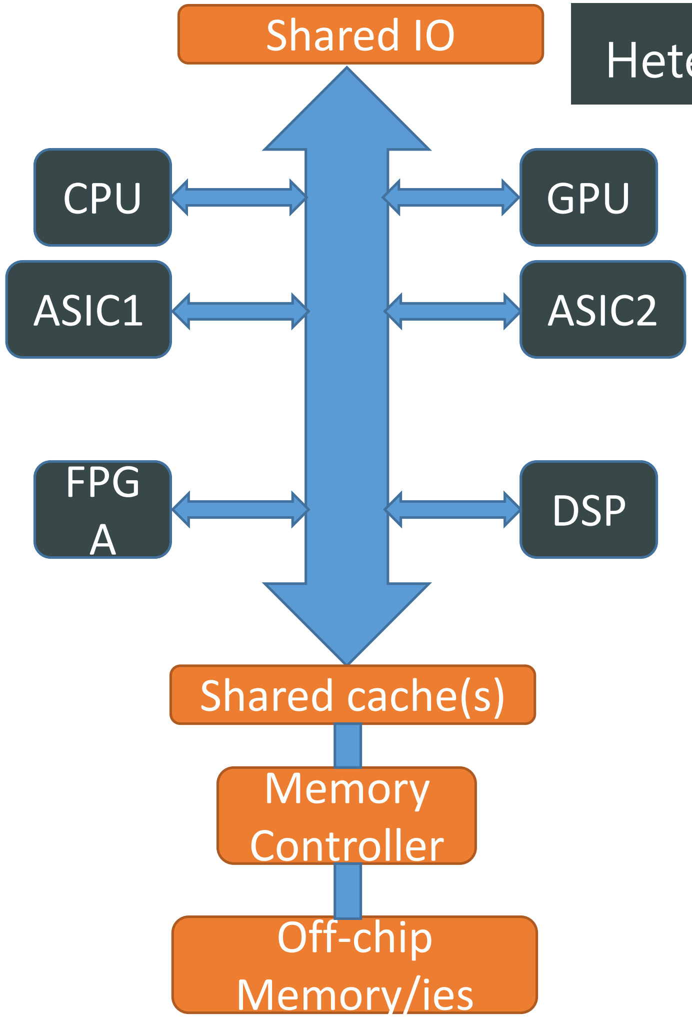
Heterogenous MPSoCs

Heterogenous MPSoCs

Why Heterogenous MPSoCs?

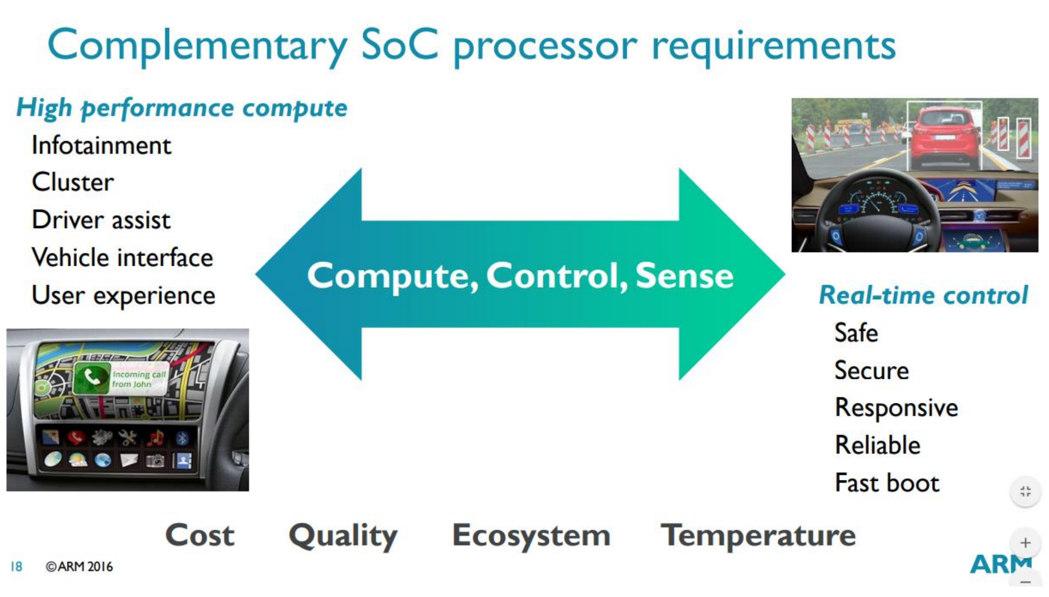
- Variety of processing capabilities

→ Best-suits MCS conflicting requirements



Heterogenous MPSoCs

MOTIVATION



Complementary SoC processor requirements

High performance compute

- Infotainment
- Cluster
- Driver assist
- Vehicle interface
- User experience

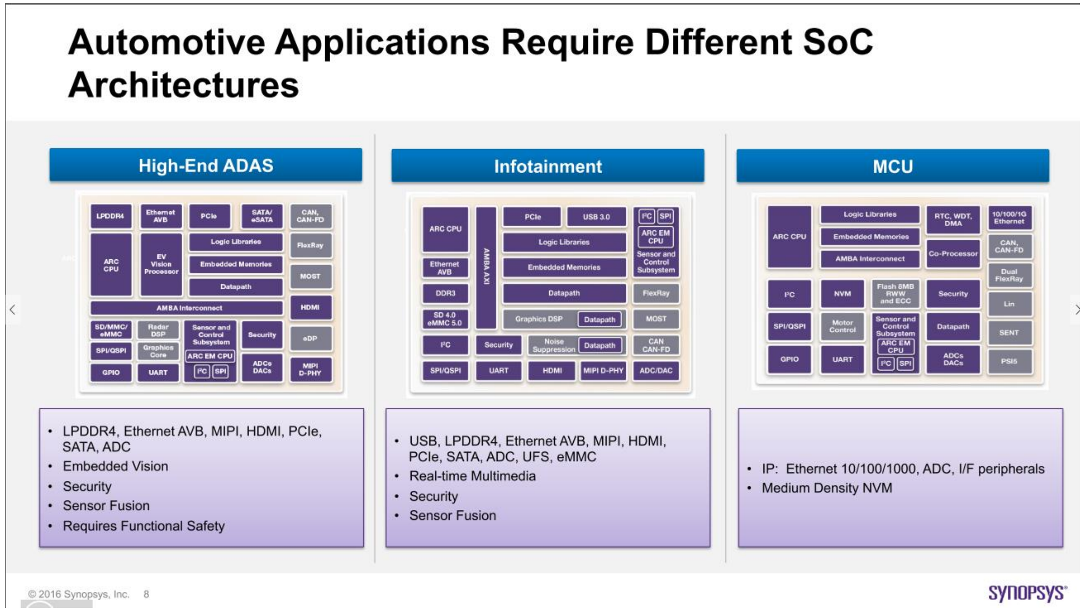
Compute, Control, Sense

Real-time control

- Safe
- Secure
- Responsive
- Reliable
- Fast boot

Cost **Quality** **Ecosystem** **Temperature**

Incoming call from John



Automotive Applications Require Different SoC Architectures

High-End ADAS

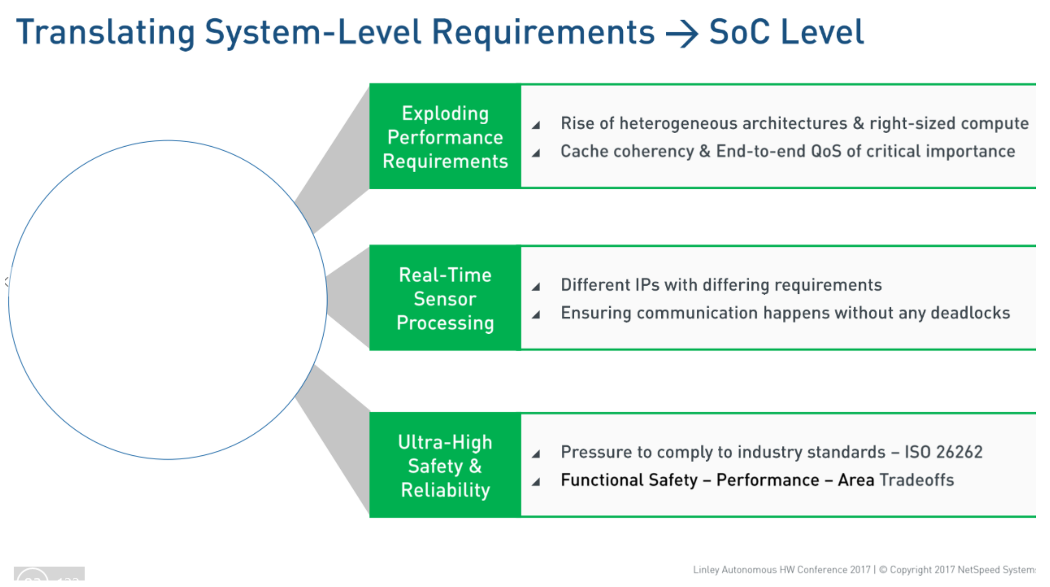
- LPDDR4, Ethernet AVB, MIPI, HDMI, PCIe, SATA, ADC
- Embedded Vision
- Security
- Sensor Fusion
- Requires Functional Safety

Infotainment

- USB, LPDDR4, Ethernet AVB, MIPI, HDMI, PCIe, SATA, ADC, UFS, eMMC
- Real-time Multimedia
- Security
- Sensor Fusion

MCU

- IP: Ethernet 10/100/1000, ADC, I/F peripherals
- Medium Density NVM



Translating System-Level Requirements → SoC Level

Exploding Performance Requirements

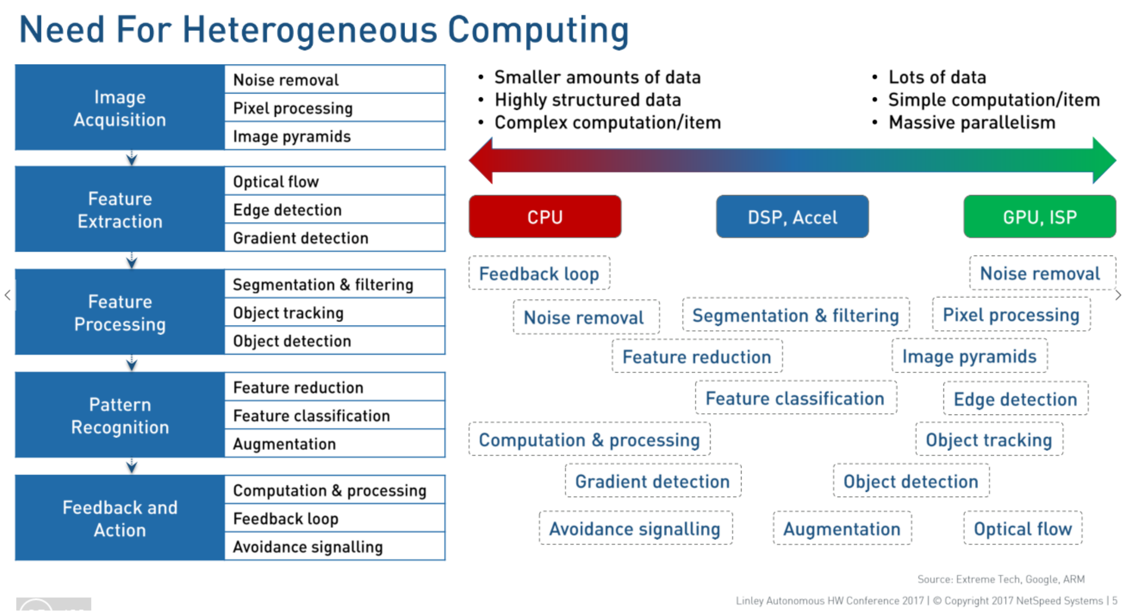
- Rise of heterogeneous architectures & right-sized compute
- Cache coherency & End-to-end QoS of critical importance

Real-Time Sensor Processing

- Different IPs with differing requirements
- Ensuring communication happens without any deadlocks

Ultra-High Safety & Reliability

- Pressure to comply to industry standards – ISO 26262
- **Functional Safety – Performance – Area** Tradeoffs



Need For Heterogeneous Computing

Image Acquisition	Noise removal Pixel processing Image pyramids
Feature Extraction	Optical flow Edge detection Gradient detection
Feature Processing	Segmentation & filtering Object tracking Object detection
Pattern Recognition	Feature reduction Feature classification Augmentation
Feedback and Action	Computation & processing Feedback loop Avoidance signalling

- Smaller amounts of data
- Highly structured data
- Complex computation/item

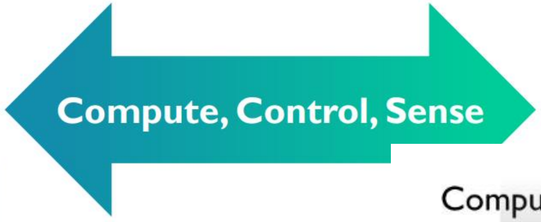
- Lots of data
- Simple computation/item
- Massive parallelism

Source: Extreme Tech, Google, ARM

Complementary SoC processor requirements

High performance compute

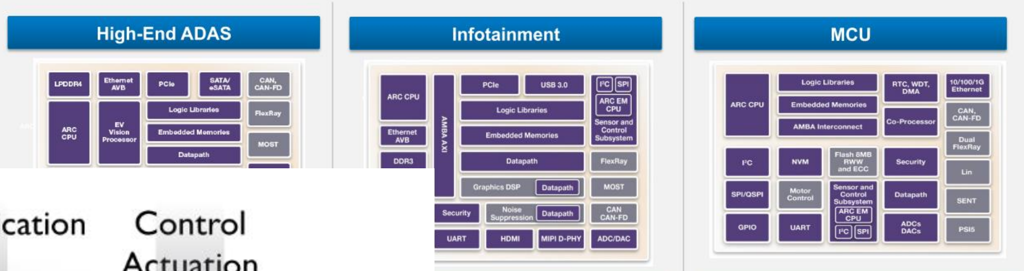
Infotainment
Cluster
Driver assist
Vehicle interface
User experience



Real-time control

Cost **Quality** **Ecosystem**

Automotive Applications Require Different SoC Architectures



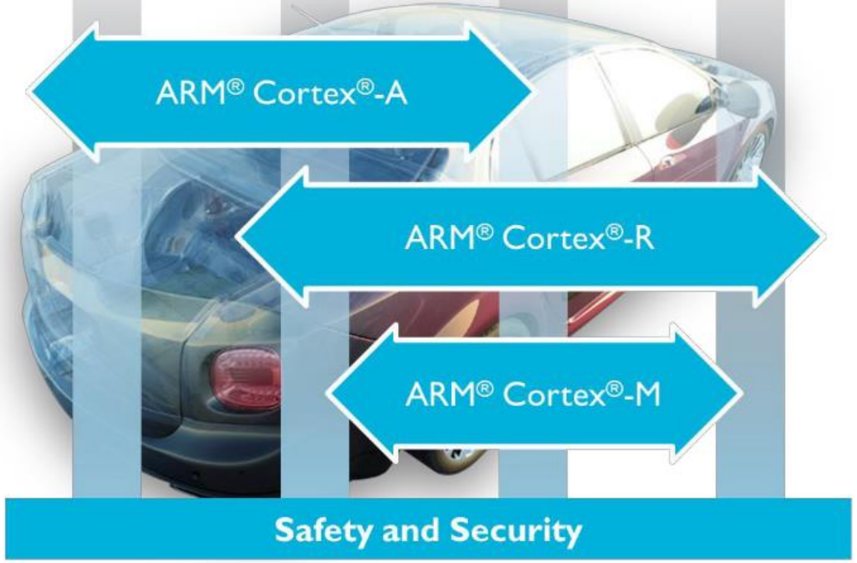
)R4, Ethernet AVB, MIPI, HDMI,
, ADC, UFS, eMMC
Multimedia

ion

- IP: Ethernet 10/100/1000, ADC, I/F peripherals
- Medium Density NVM

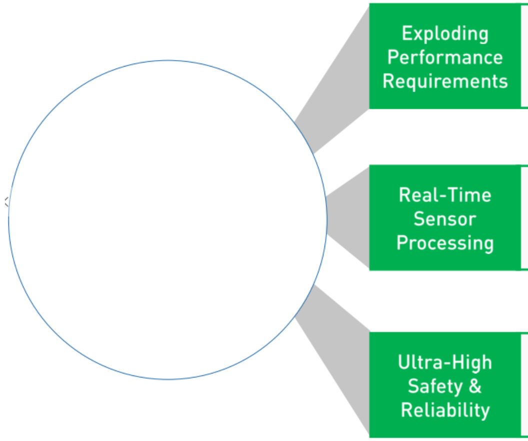
SYNOPSYS

Computation Automation | Sensing | Communication | Control Actuation



ARM

Translating System-Level Requirement

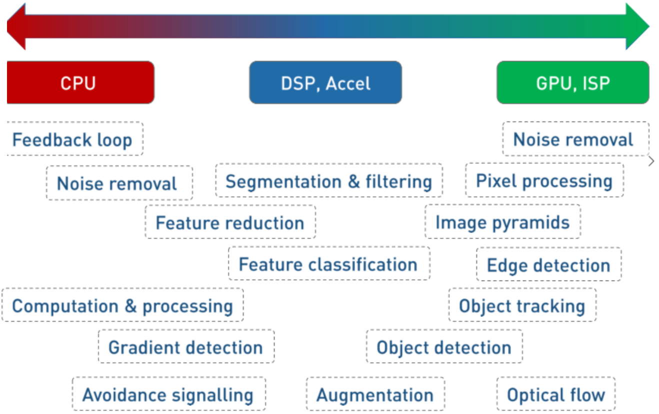


Exploding Performance Requirements	Rise of het Cache coh
Real-Time Sensor Processing	Different I Ensuring c
Ultra-High Safety & Reliability	Pressure to comply to industry standards – ISO 26262 Functional Safety – Performance – Area Tradeoffs

omputing

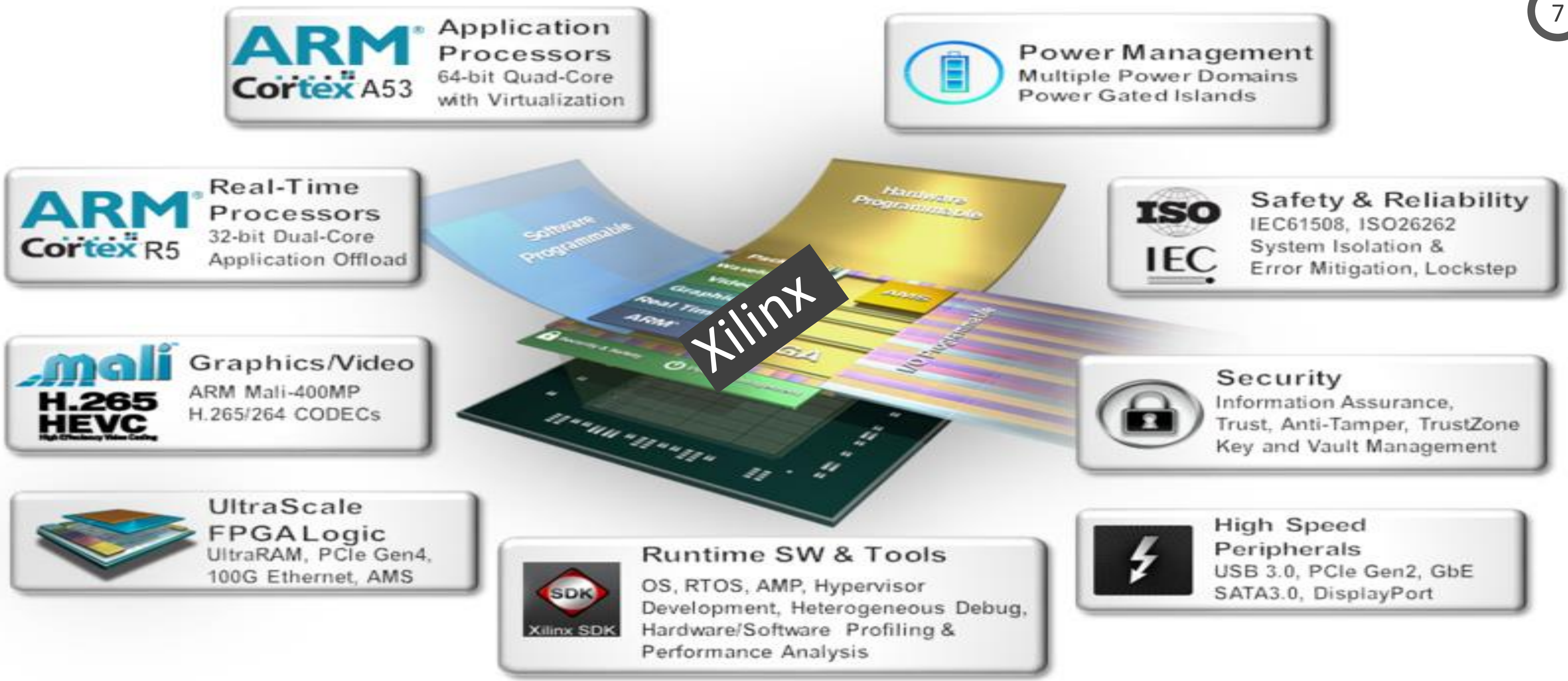
- Smaller amounts of data
- Highly structured data
- Complex computation/item

- Lots of data
- Simple computation/item
- Massive parallelism



Pattern Recognition	Feature reduction Feature classification Augmentation
Feedback and Action	Computation & processing Feedback loop Avoidance signalling

Source: Extreme Tech, Google, ARM



ARM Cortex A53 — Application Processors
64-bit Quad-Core
with Virtualization

Power Management
Multiple Power Domains
Power Gated Islands

ARM Cortex R5 — Real-Time Processors
32-bit Dual-Core
Application Offload

Safety & Reliability
IEC61508, ISO26262
System Isolation &
Error Mitigation, Lockstep

Graphics/Video
ARM Mali-400MP
H.265/264 CODECs

Security
Information Assurance,
Trust, Anti-Tamper, TrustZone
Key and Vault Management

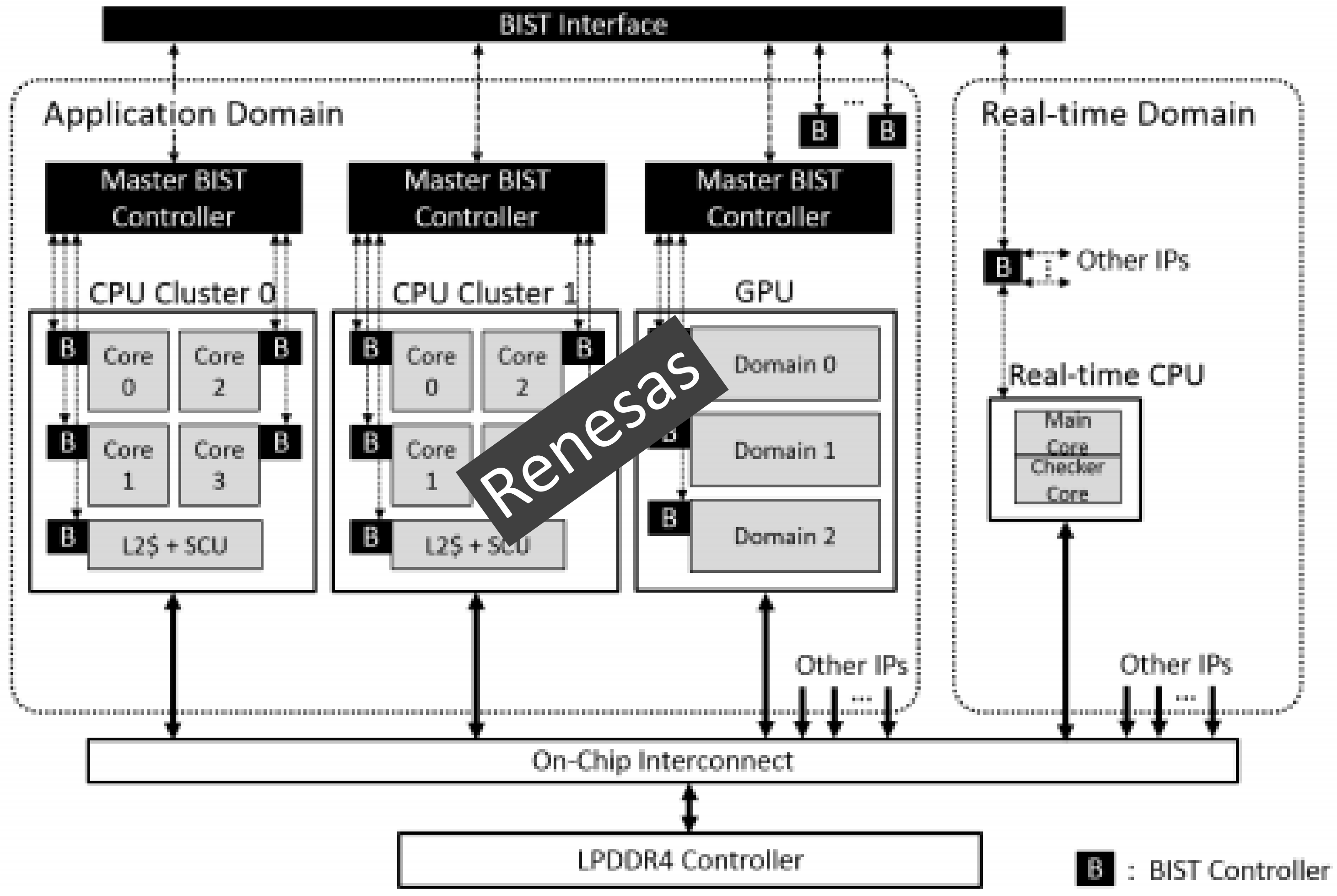
UltraScale FPGA Logic
UltraRAM, PCIe Gen4,
100G Ethernet, AMS

Runtime SW & Tools
OS, RTOS, AMP, Hypervisor
Development, Heterogeneous Debug,
Hardware/Software Profiling &
Performance Analysis

High Speed Peripherals
USB 3.0, PCIe Gen2, GbE
SATA3.0, DisplayPort

Heterogenous MPSoCs with Real-time Processors

MOTIVATION

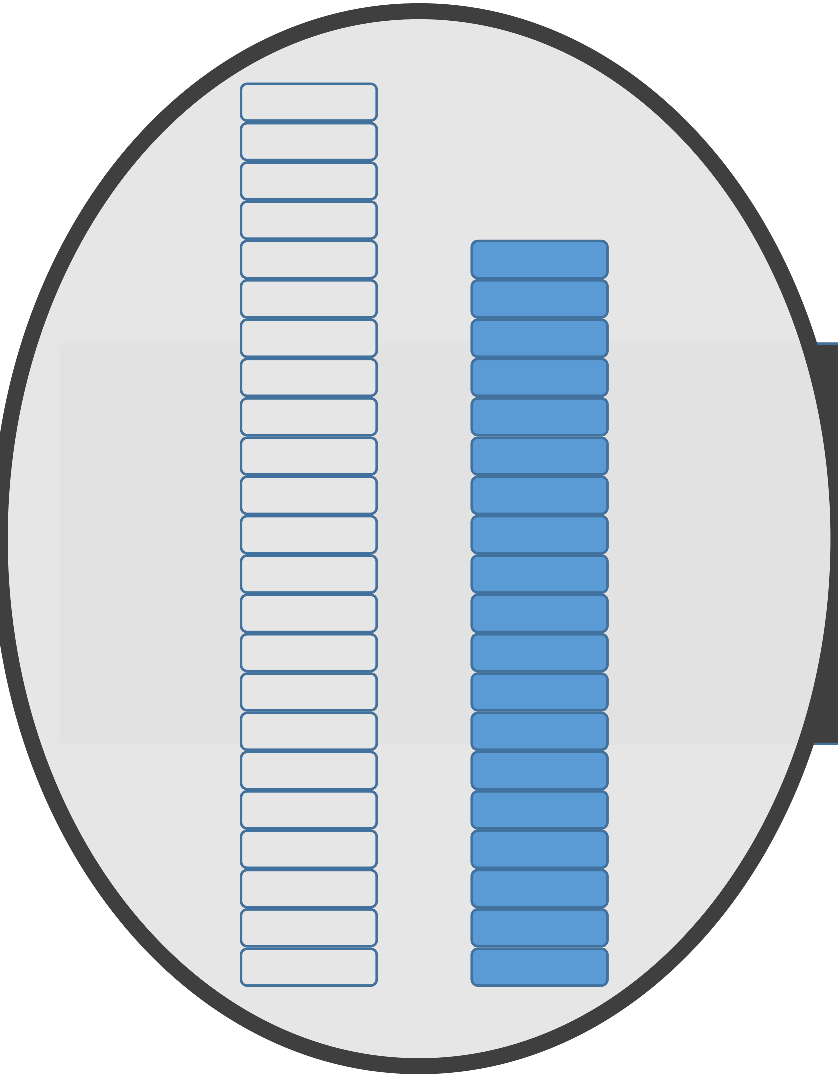


Heterogenous MPSoCs with Real-time Processors

MOTIVATION

Where Are We?

WMC13-17



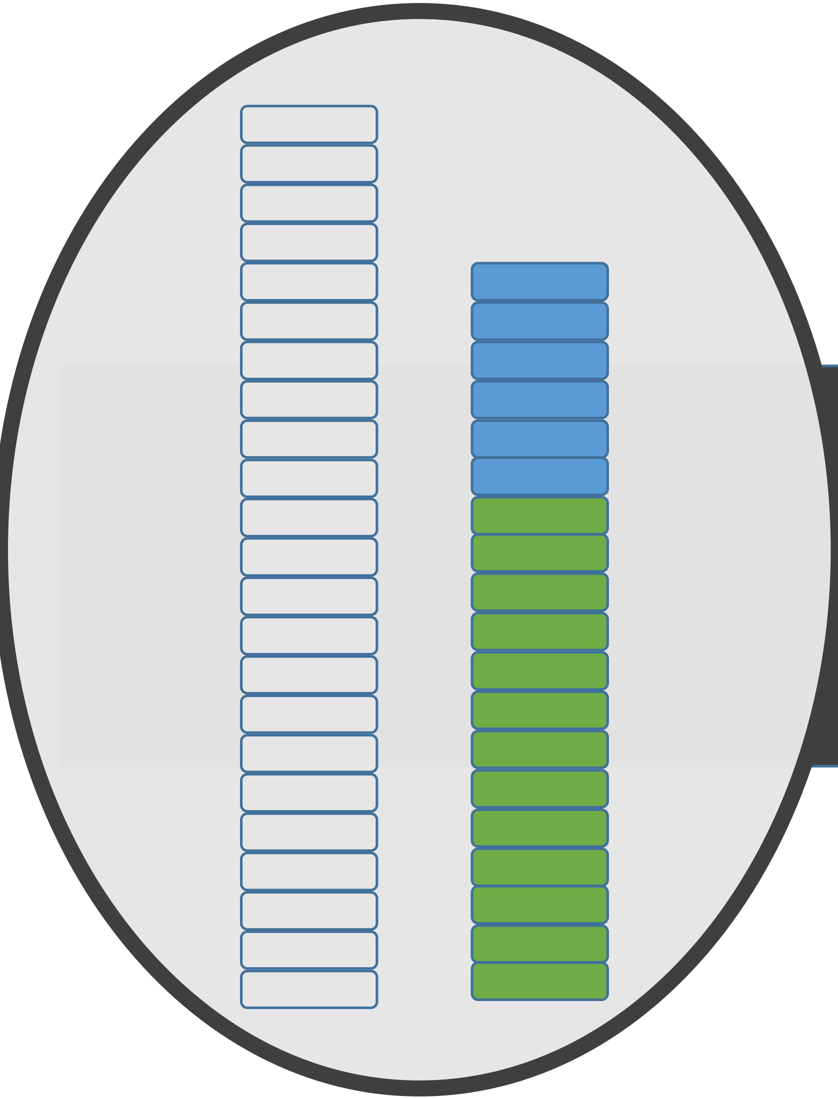
Uniprocessor: 23

Multiprocessor: 19 45%

Where Are We?

WMC13-17

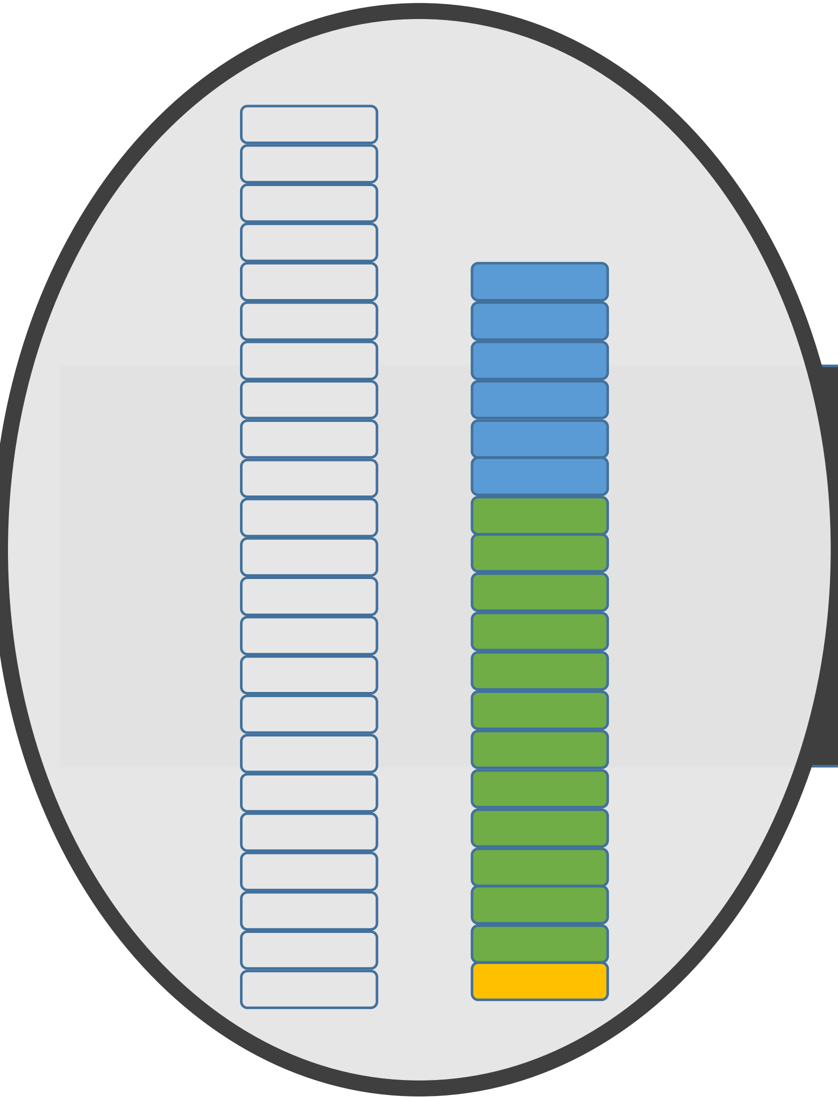
Where Are We?
WMC13-17



Uniprocessor: 23

Multiprocessor: 19 **45%**

- ➢ **Shared Resources:** 13 **~31%**



Uniprocessor: 23

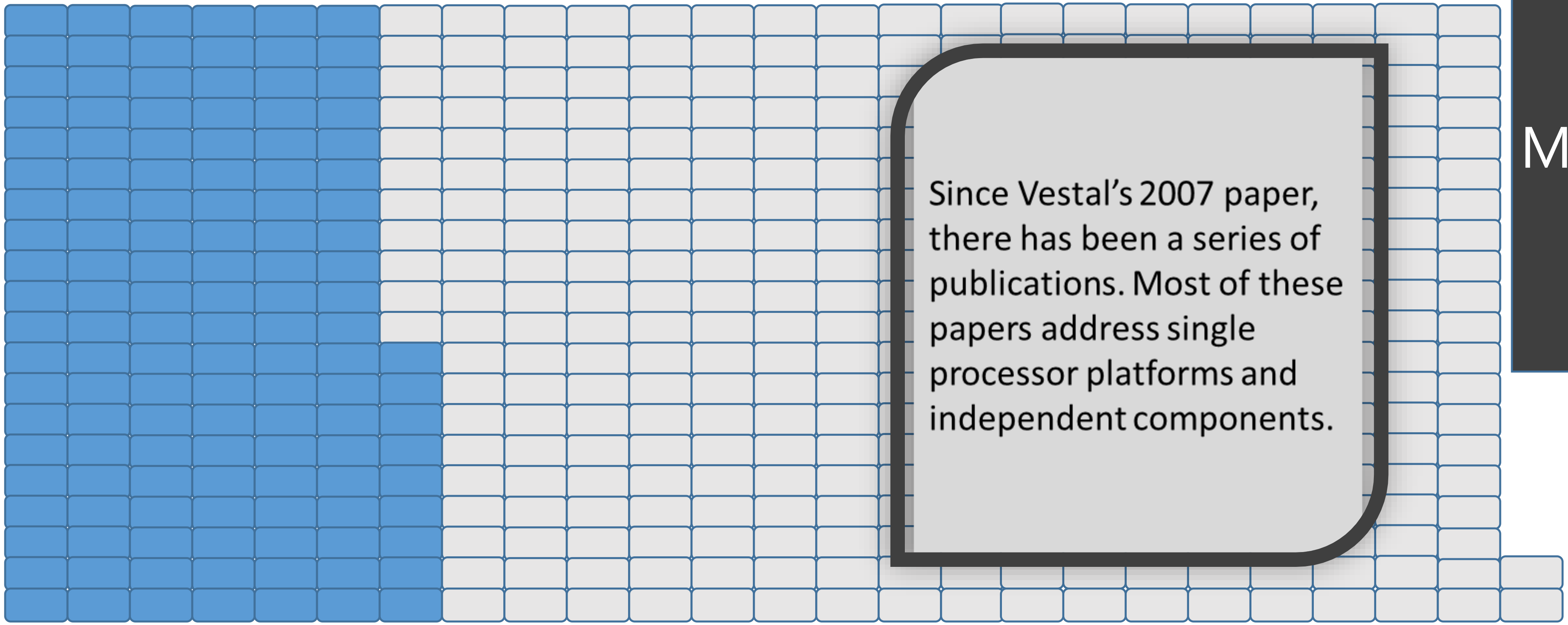
Multiprocessor: 19 45%

- Shared Resources: 13 ~31%
- SoCs: 1 ~2%

Where Are We?
WMC13-17

Where Are We?

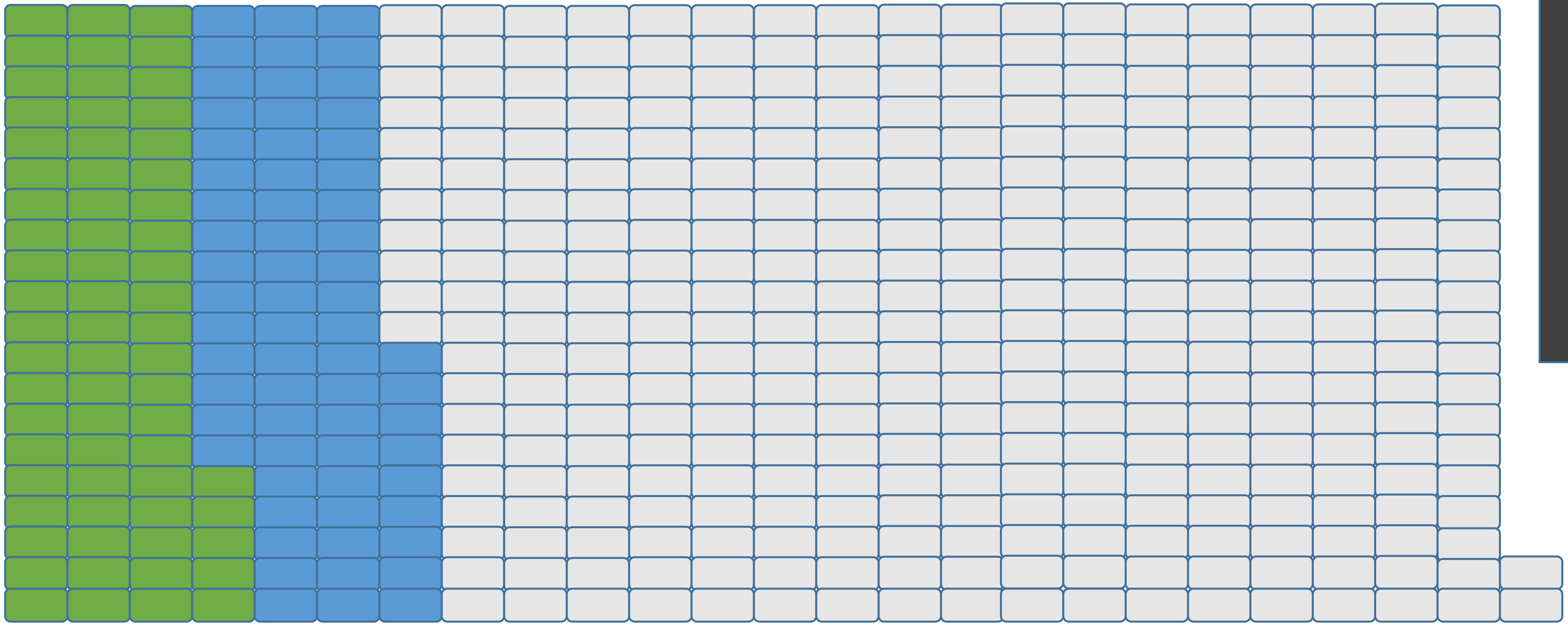
Overall, MCS review[Burns and Davis]



Since Vestal’s 2007 paper, there has been a series of publications. Most of these papers address single processor platforms and independent components.

Multiprocessor

~27%

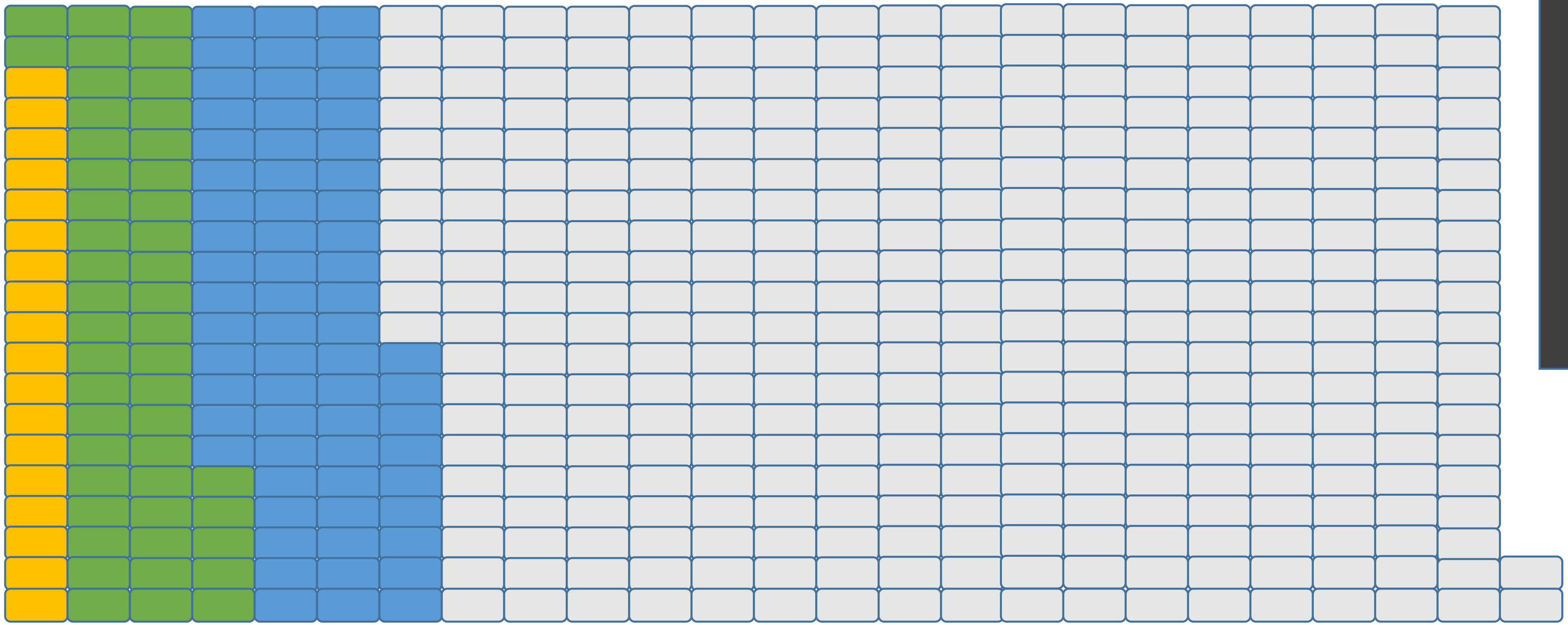


Shared Resources

~13%

Where Are We?

Overall, MCS review[Burns and Davis]



SoCs

~4%

Where Are We?

Overall, MCS review[Burns and Davis]

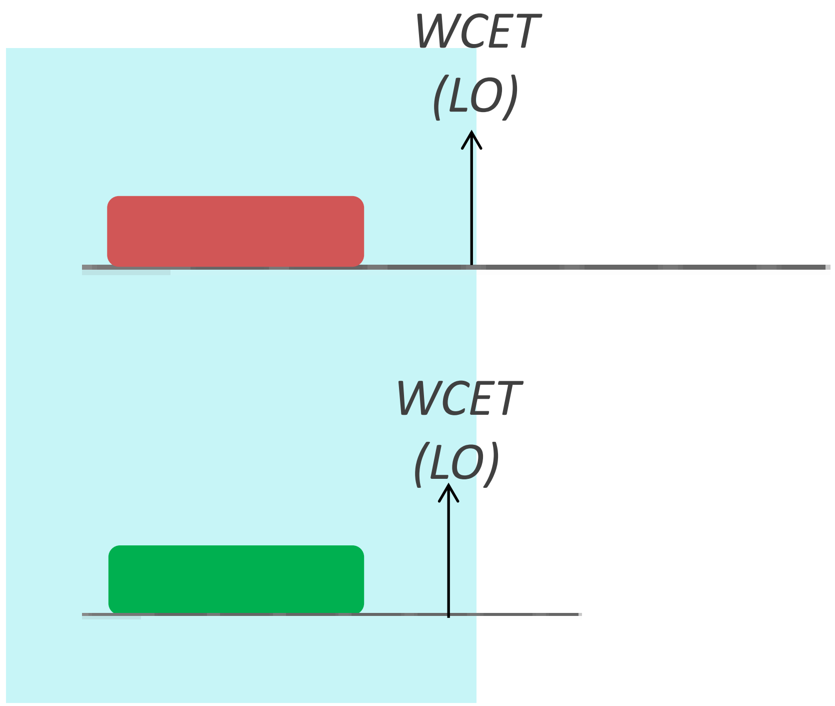
MPSoC-Based MCS: Four Aspects

Traditional MCS Model

Timing Interference

Data Sharing

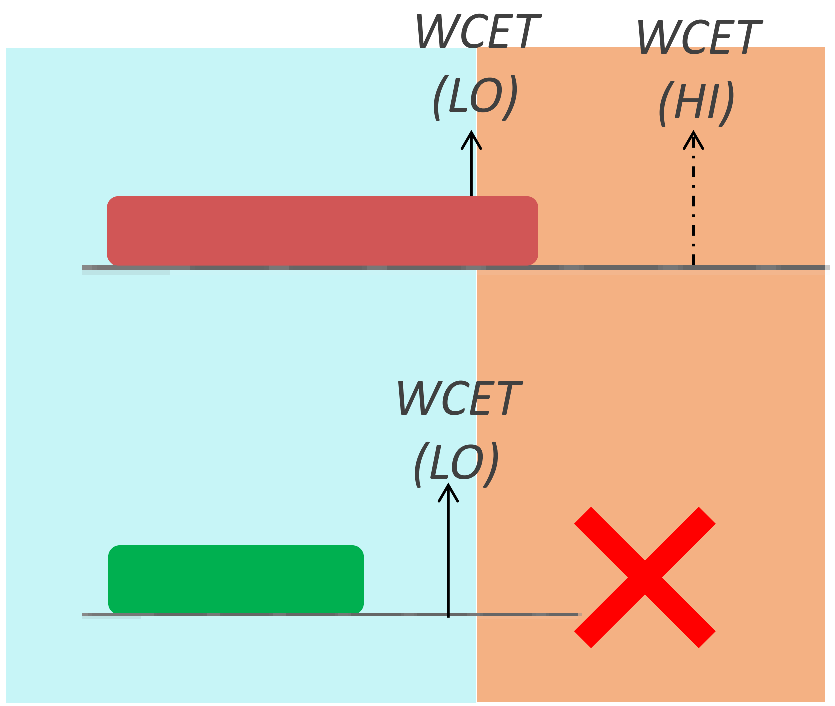
Security



normal mode

degraded mode

Traditional Model



Traditional Model

Problems with the Model

Suspension

- Lower-criticality tasks are suspended upon switching to a higher-mode (not acceptable in industry [P. Graydon and I. Bate, WMC 2013])

Overheads

- Switching leads to huge overheads (usually overlooked) [L. Sigrist et al, RTAS 2015]

Sources of Uncertainty

- Of special importance for MPSoCs (more next)

MPSoCs Opportunities

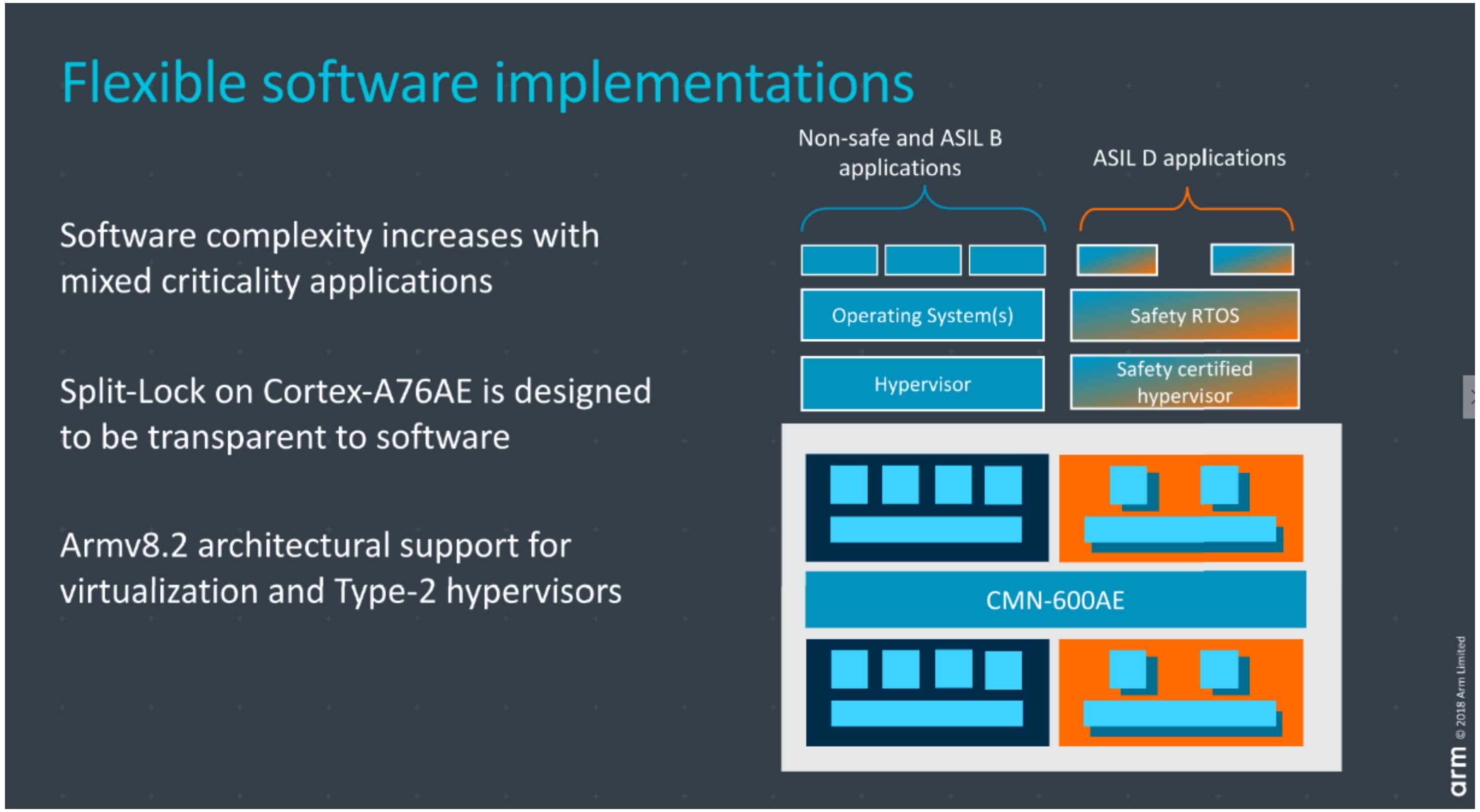
1. MPSoCs create switching alternatives

- Different modes of operation at different cluster of PEs?

MPSoCs Opportunities

1. MPSoCs create switching alternatives

- Different modes of operation at different cluster of PEs?



Flexible software implementations

Software complexity increases with mixed criticality applications

Split-Lock on Cortex-A76AE is designed to be transparent to software

Armv8.2 architectural support for virtualization and Type-2 hypervisors

MPSoCs Opportunities

1. MPSoCs create switching alternatives

- Different modes of operation at different cluster of PEs?
- Migrate instead of switching?
  - Dynamic Reconfiguration (IEC61508-7)

C.3.13 Dynamic reconfiguration
The logical architecture of the system has to be such that it can be mapped onto a subset of the available resources of the system. The architecture needs to be capable of detecting a failure in a physical resource and then remapping the logical architecture back onto the restricted resources left functioning. Although the concept is more traditionally restricted to recovery from failed hardware units, it is also applicable to failed software units if there is sufficient 'run-time redundancy' to allow a software re-try or if there is sufficient redundant data to make the individual and isolated failure be of little importance. This technique must be considered at the first system design stage.

MPSoCs Opportunities

1. MPSoCs create switching alternatives

- Different modes of operation at different cluster of PEs?
- Migrate instead of switching?

2. MPSoCs open the door for customized solutions

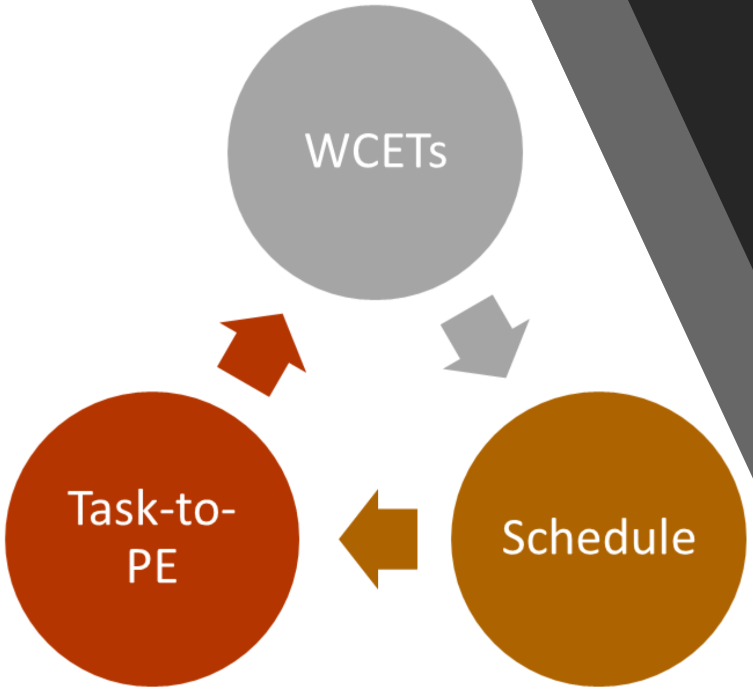
- Using specialized PEs is a norm in MPSoCs
- Dedicating a PE for the runtime monitoring
  - faster detection of exceptional events → react in a timely manner
- PE can be further tailored to optimize the behavior of the monitoring techniques

MPSoCs Challenges

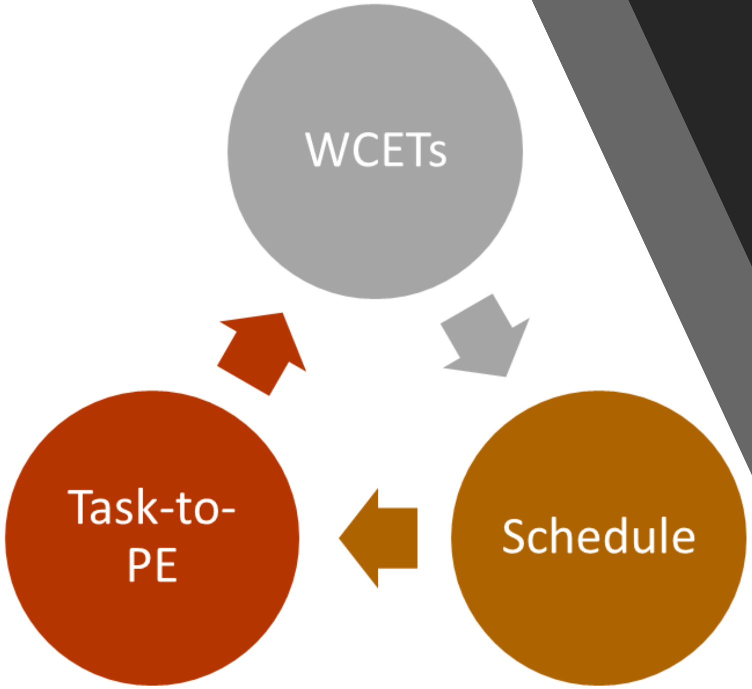
1. Common assumption:

   “uncertainty in WCET does not come from the system itself; rather, it comes from our inability to measure (or compute) it with complete confidence”

- Well, this may not be completely true for MPSoCs
  - ➢In SMPs, which core (or cores) executing a task does not affect its measured execution time.
  - ➢In MPSoCs, this decision directly affects the level of certainty in its WCET:
    - Real-time vs High-performance PEs?
    - Use scratchpads vs caches?



MPSoCs Challenges



2. Scalability challenges associated with these scheduling and monitoring techniques.

3. Mode switching in MPSoCs may incur task migrations or reassignment of heterogeneous cores to tasks

- ➢ the effects of these decisions on the switching overhead need to be quantified.

MPSoC-Based MCS: Four Aspects

- Traditional MCS Model ✓
- Timing Interference
- Data Sharing
- Security

Timing Interference

Challenge: operations of one PE affect the temporal behavior of other PEs, which complicates the timing analysis of the system.

Most of the MCS scheduling techniques do not incorporate these interferences in their scheduling or analysis

Approaches focusing on shared resources mostly assume SMPs

Challenge: operations of one PE affect the

7.4.2.7 Where the software is to implement both safety and non-safety functions, then all of the software shall be treated as safety-related, unless adequate independence between the functions can be demonstrated in the design.
[IEC61508-3]

mostly assume SMPs

Timing Interference

MPSoCs Opportunities

All about Flexibility

1. Which memory levels should be shared amongst which cores

- Does the GPU share the LLC with the CPU?

2. How to distribute the cache architecture?

- Would implementing a NUCA be adequate for MCS (e.g., helping in achieving different levels of isolation)?

3. Different types of on-chip memories

- Both caches and SPMs
- Most of the currently available approaches focus on a single type

4. Different types of available off-chip memories

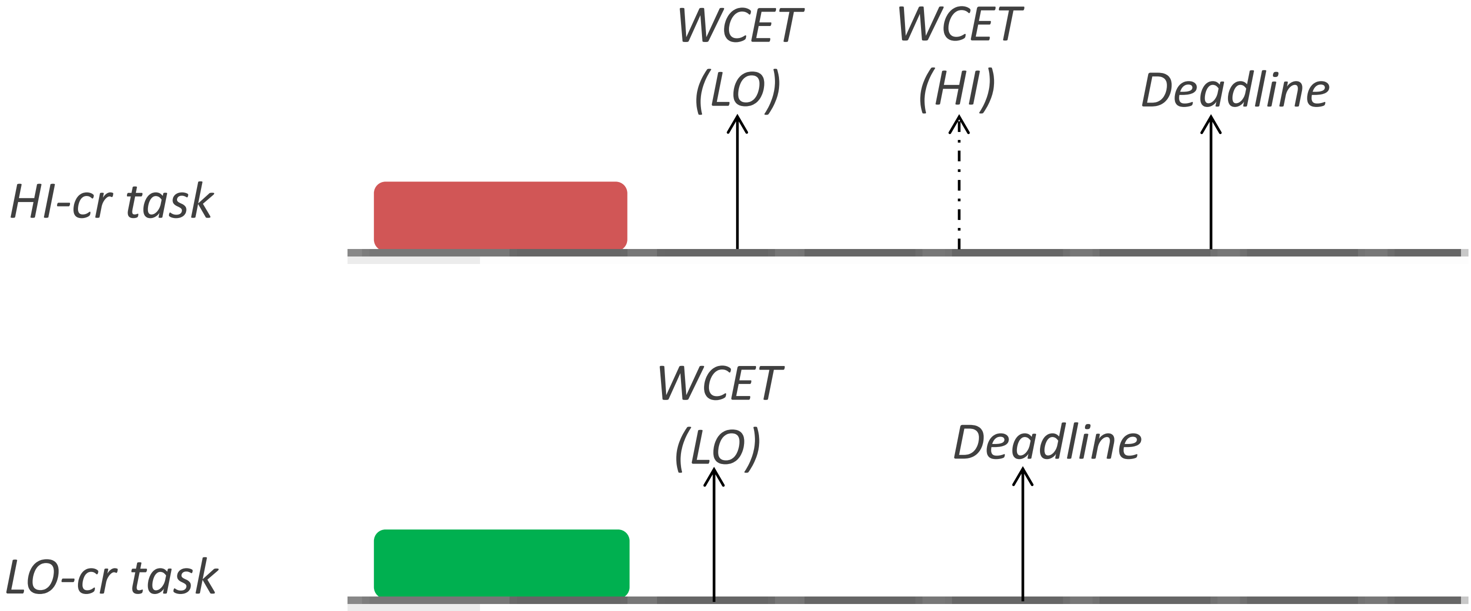
- DDR, GDDR, RLDRAM, LPDDR, QDR.
- Investigating the cooperation of these types is also worth investigating

MPSoCs Challenges

1. The interference exaggerates with the increase in the number of PEs
2. Understanding the architectural details of shared resources is inevitable to derive realistic bounds.
3. Each type of PEs has its own memory access behavior, which complicates the analysis, leading to more pessimism
   - Data-intensive PEs (e.g. multimedia/DSP processors) can saturate system queues
   - ***A requirement- and criticality-aware arbitration is a must to deliver differential service to PEs***

$$task = \langle CL, D, \boxed{WCET(CL)} \rangle$$

calculated/measured in isolation

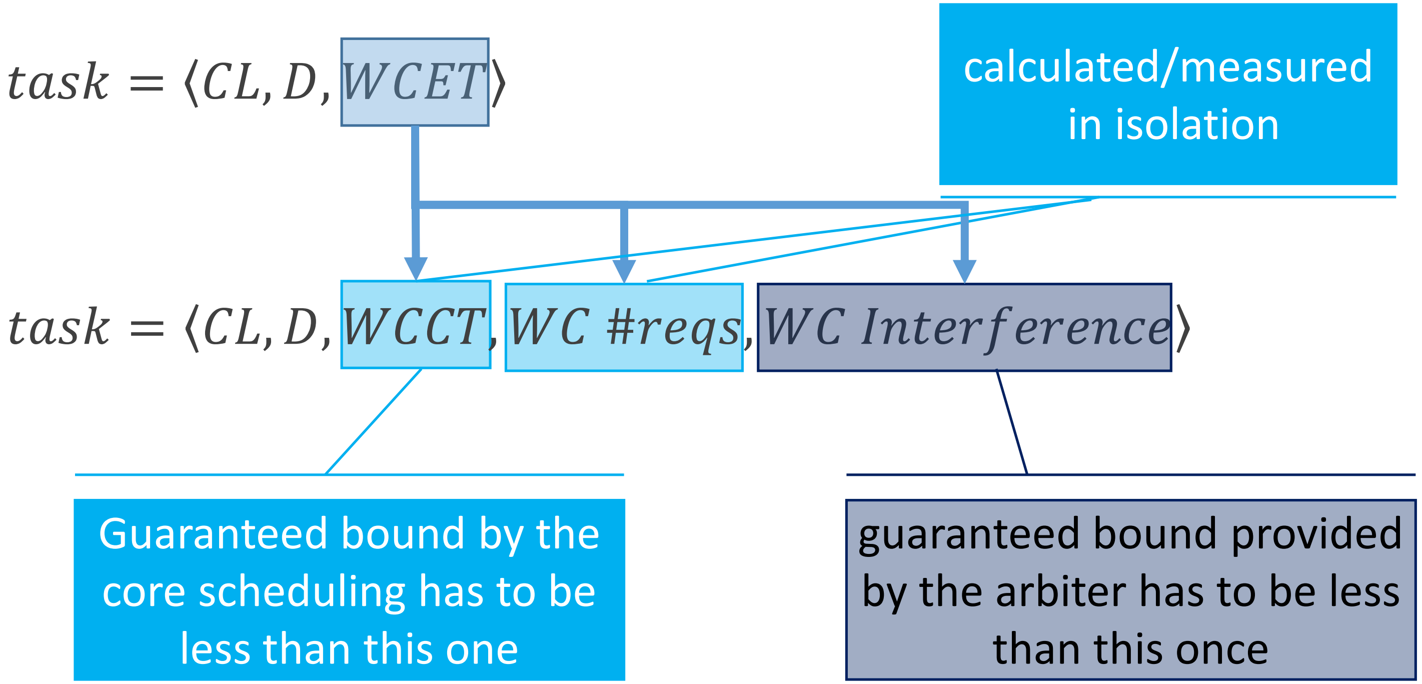


Traditional Model

Timing Interference

Extending Traditional Model

Timing Interference



$task = \langle CL, D, WCET \rangle$

calculated/measured in isolation

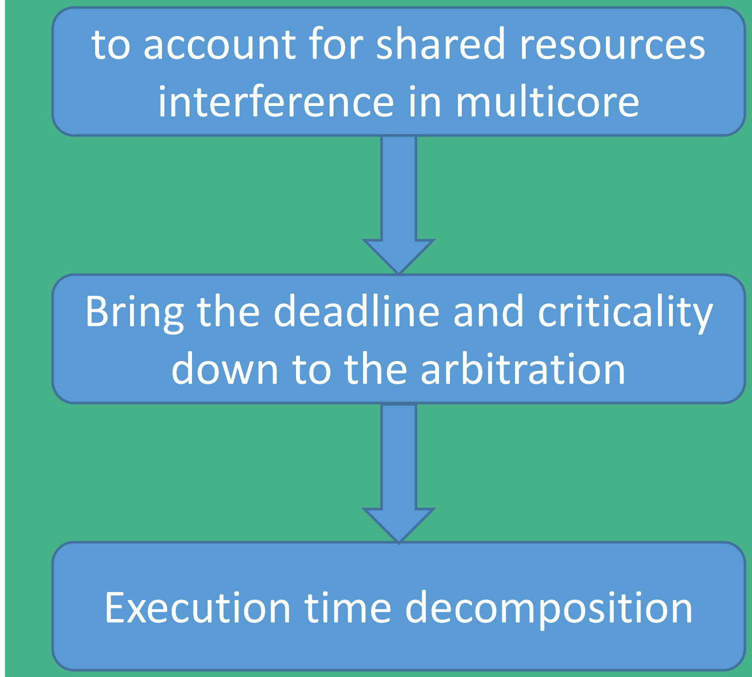
$task = \langle CL, D, WCCT, WC\ \#reqs, WC\ Interference \rangle$

Guaranteed bound by the core scheduling has to be less than this one

guaranteed bound provided by the arbiter has to be less than this once

$$WCET(CL) = WCCT + WC\ \#\ reqs \times WC\ Interference$$

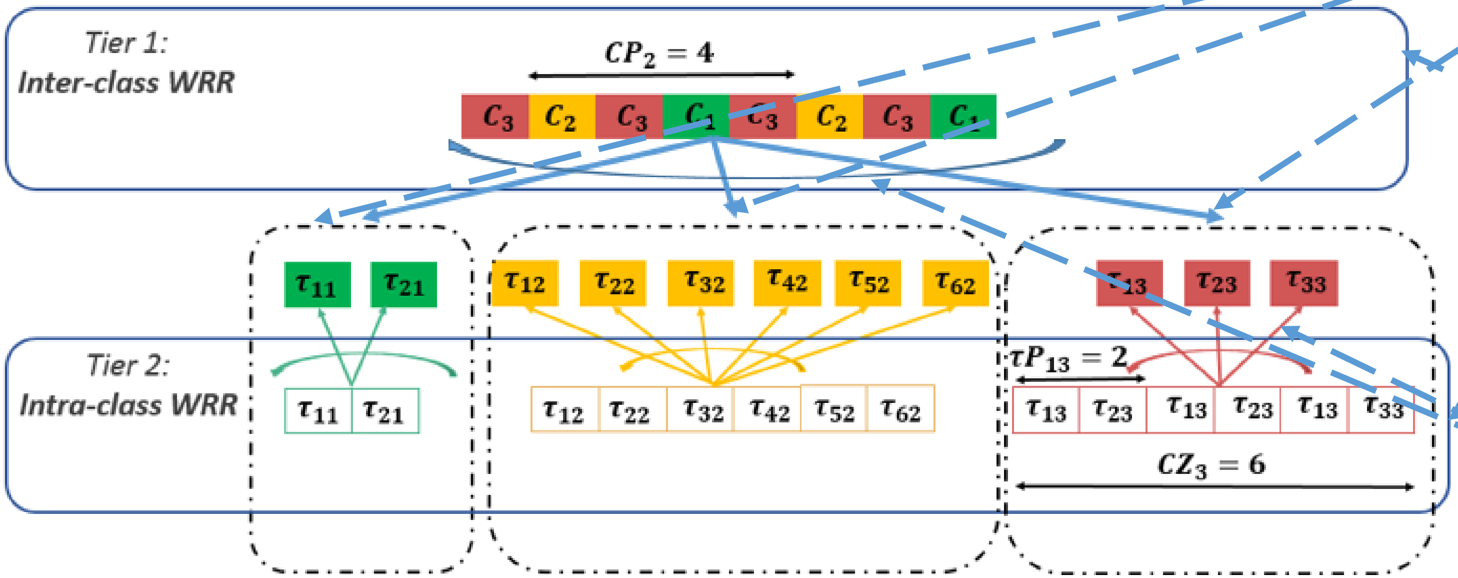
Solution



to account for shared resources interference in multicore

Bring the deadline and criticality down to the arbitration

Execution time decomposition


Tier 1:
Inter-class WRR
$CP_2 = 4$
C_3 C_2 C_3 C_1 C_3 C_2 C_3 C_1
τ_{11} τ_{21}
τ_{12} τ_{22} τ_{32} τ_{42} τ_{52} τ_{62}
τ_{13} τ_{23} τ_{33}
Tier 2:
Intra-class WRR
τ_{11} τ_{21}
τ_{12} τ_{22} τ_{32} τ_{42} τ_{52} τ_{62}
$\tau P_{13} = 2$
τ_{13} τ_{23} τ_{13} τ_{23} τ_{13} τ_{33}
$CZ_3 = 6$

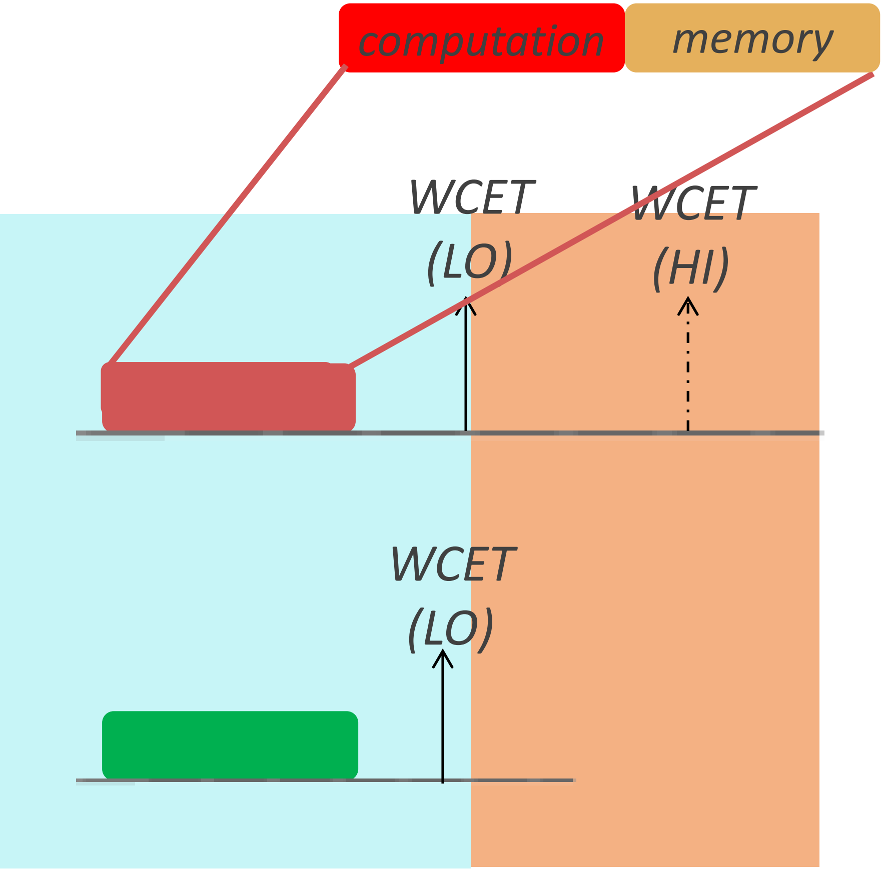
1. Combine tasks with same criticality into classes

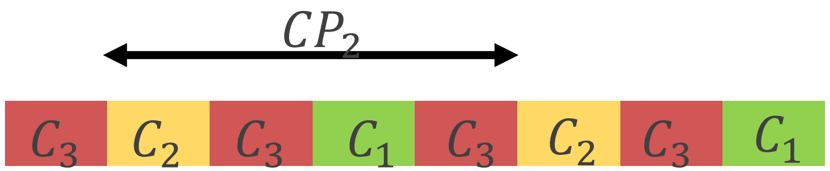
2. Two-tier Hieratical arbitration to split inter- from intra-interference
   - ✓ *Criticality awareness*

3. Harmonic WRR with optimal service assignment
   - ✓ *Requirement awareness*

CArb: Criticality- and Requirement-Aware Arbiter

Timing Interference





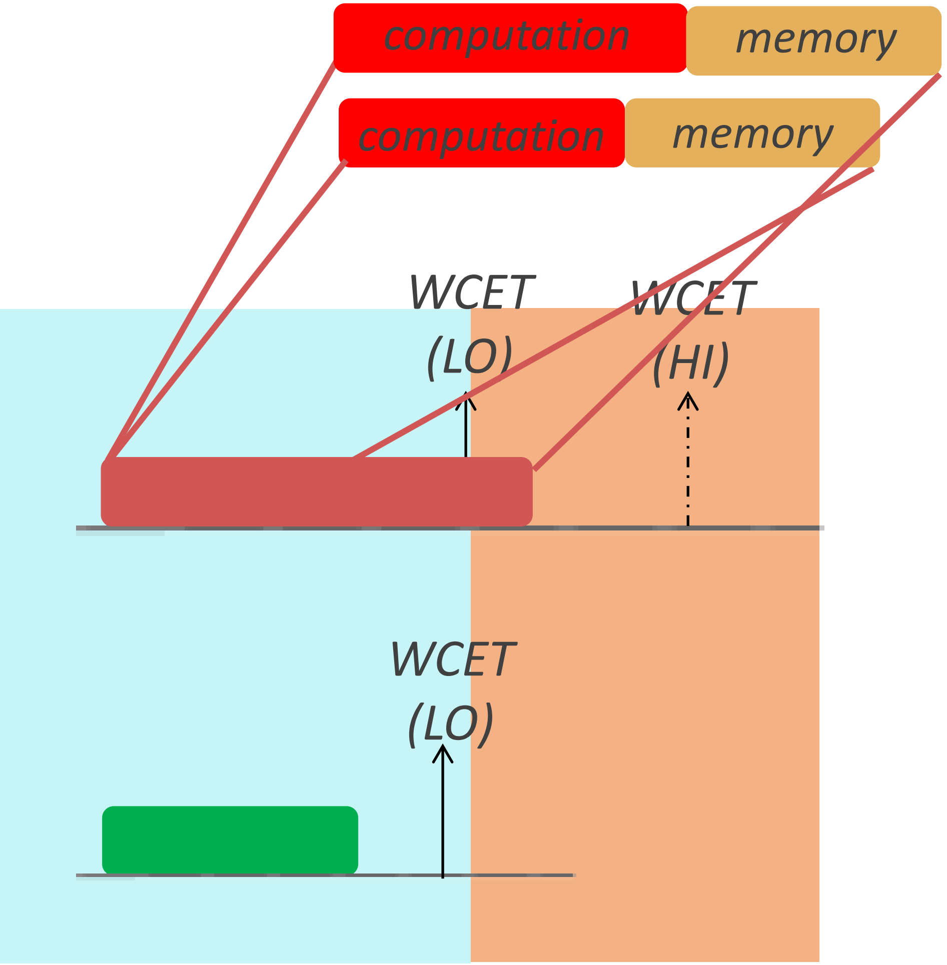
normal mode + CArb

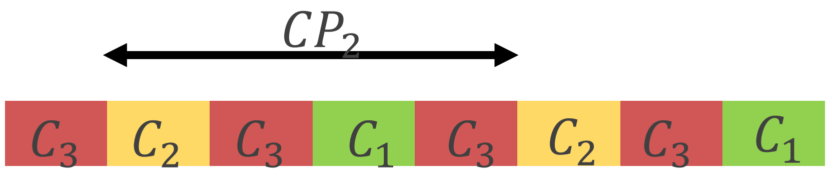
normal mode + PCArb

degraded mode

CArb: Postponing (or Eliminating) Switching

Timing Interference



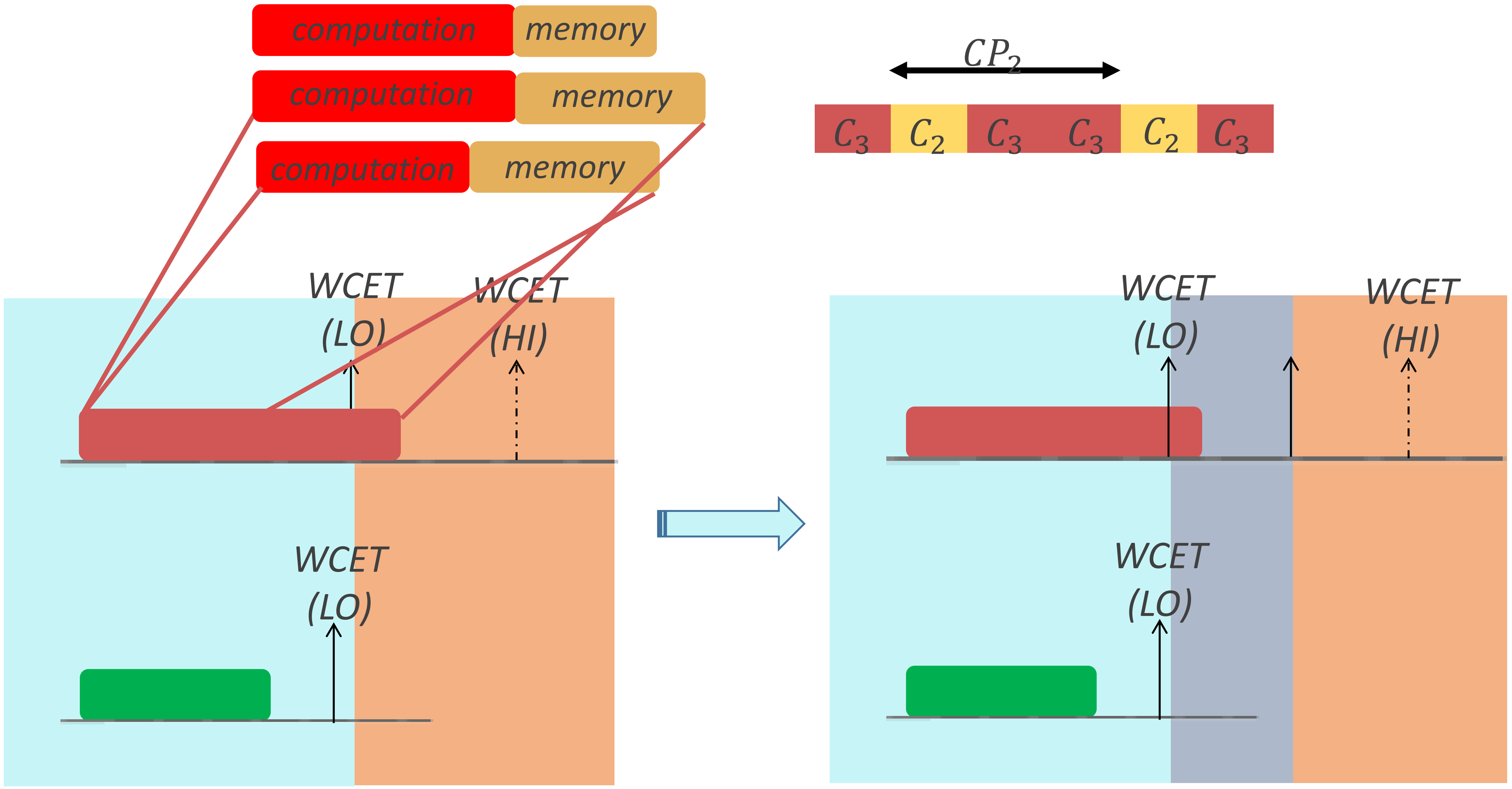


normal mode + CArb

normal mode + PCArb

degraded mode

CArb: Postponing (or Eliminating) Switching

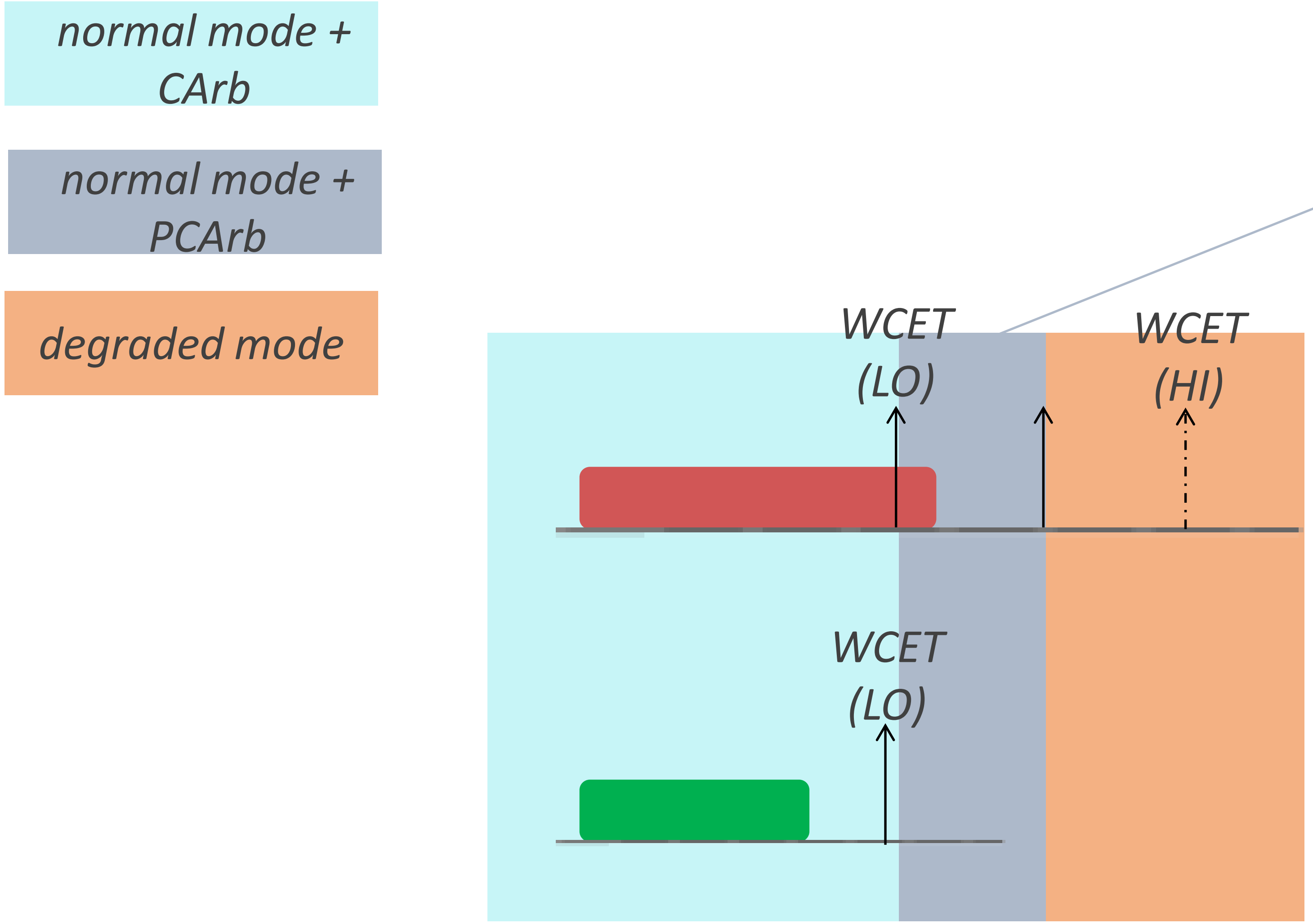


normal mode + CArb

normal mode + PCArb

degraded mode

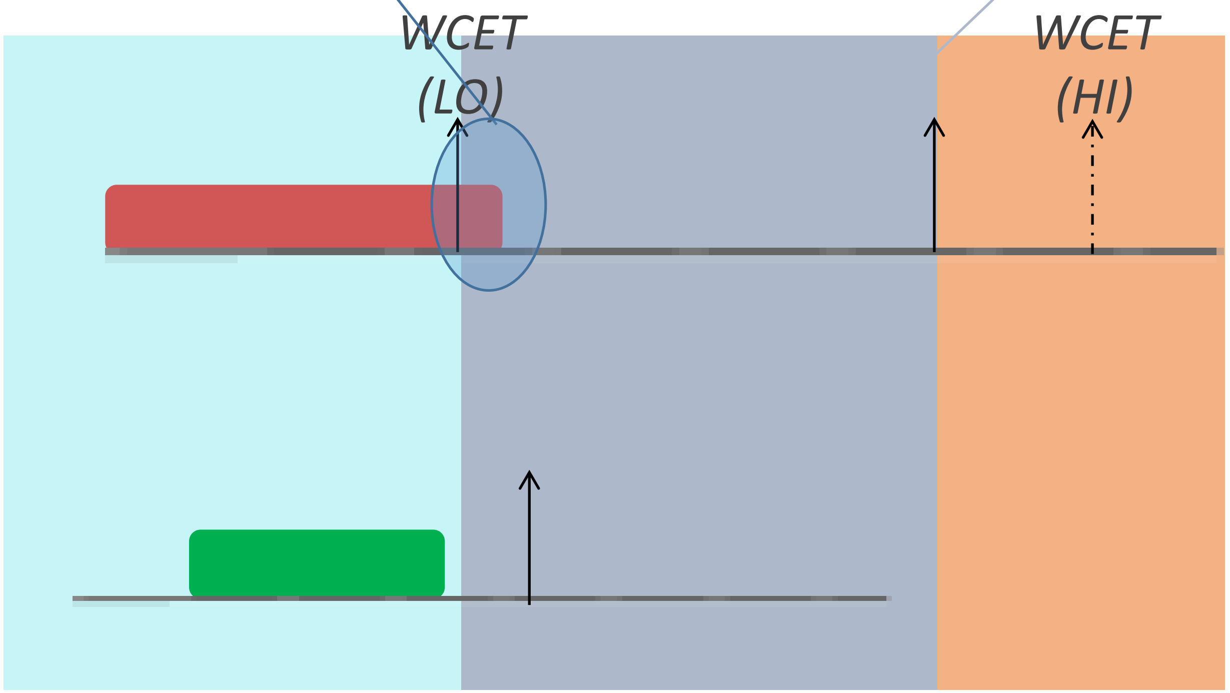
CArb: Postponing (or Eliminating) Switching



- ✓ Lower-critical tasks are not suspended
- ✓ Higher-critical tasks meet their requirement
- ✓ Postponed switching; thus decreasing overheads
- ✗ Lower-critical tasks receive no memory guarantees

CArb: Postponing (or Eliminating) Switching

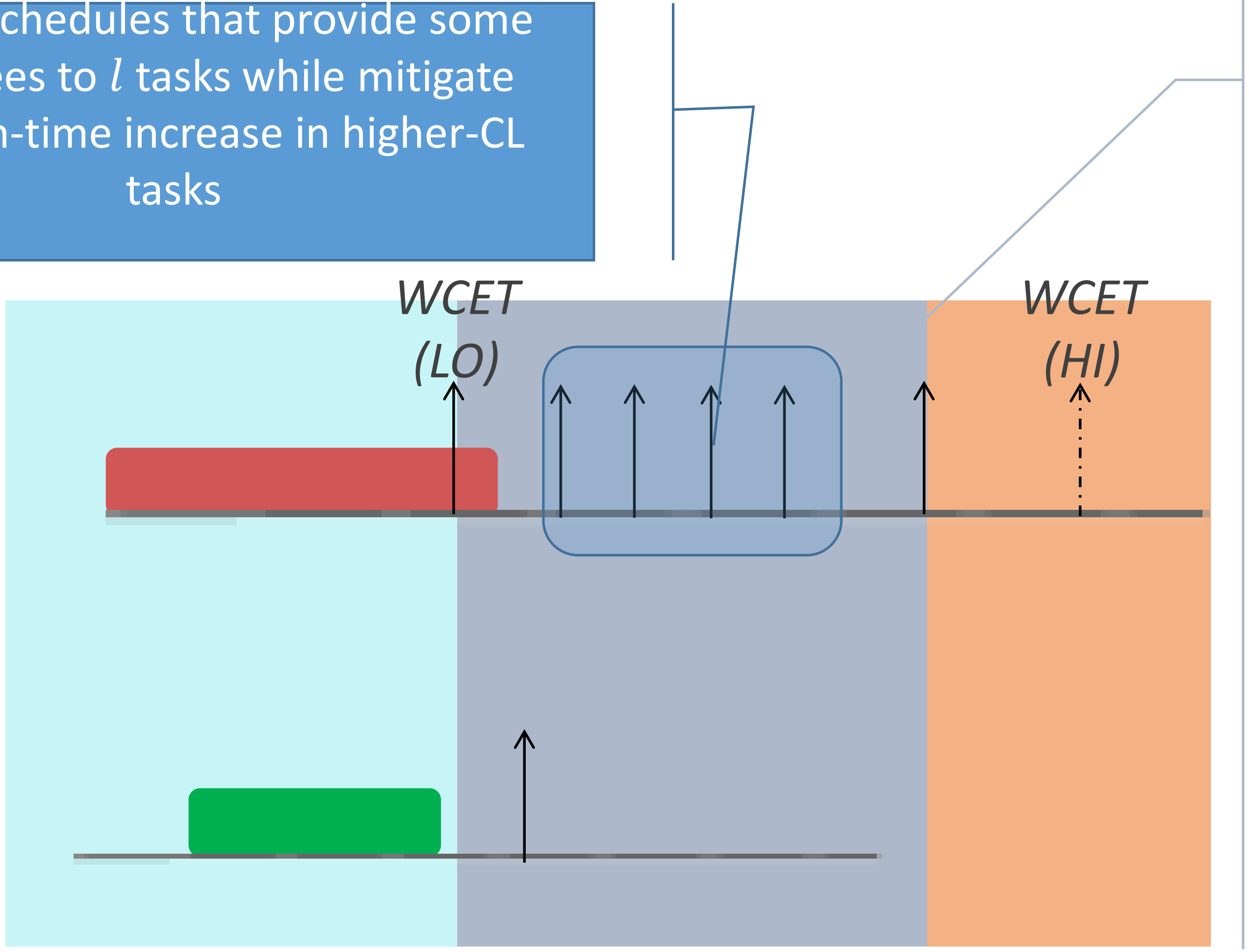
How much increase in computation time?



- ✓ Lower-critical tasks are not suspended
- ✓ Higher-critical tasks meet their requirement
- ✓ Postponed switching; thus decreasing overheads
- ✗ Lower-critical tasks receive no memory guarantees

CArb: Postponing (or Eliminating) Switching

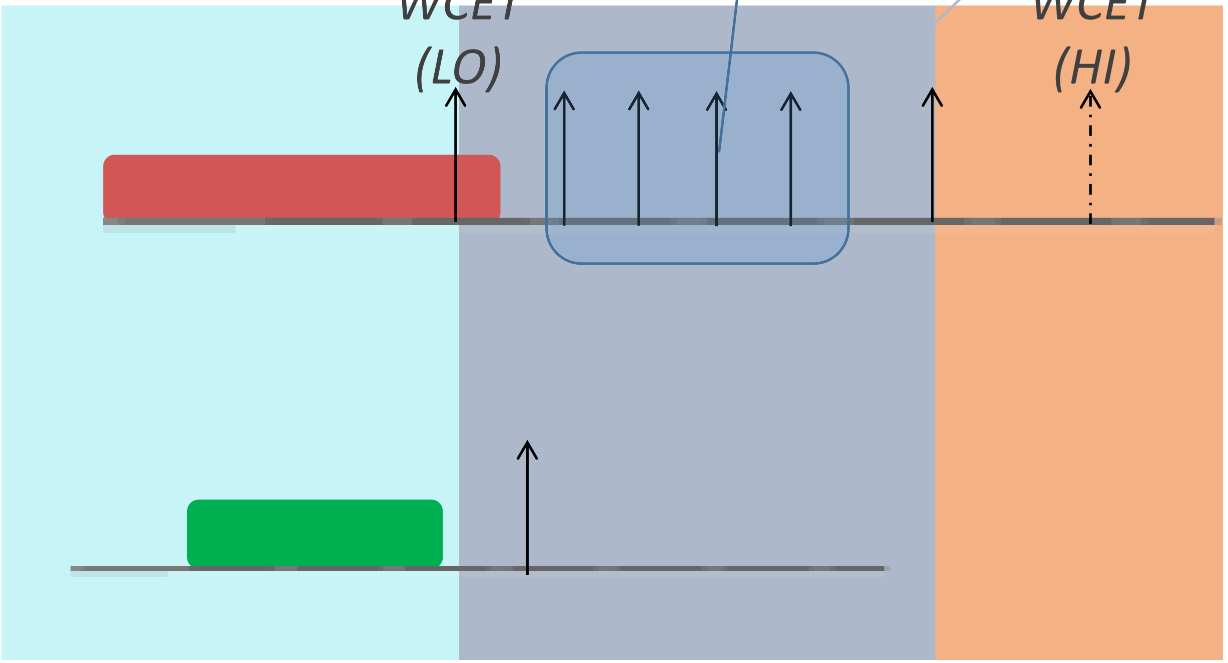
A set of schedules that provide some guarantees to l tasks while mitigate execution-time increase in higher-CL tasks



- ✓ Lower-critical tasks are not suspended
- ✓ Higher-critical tasks meet their requirement
- ✓ Postponed switching; thus decreasing overheads
- ✗ Lower-critical tasks receive no memory guarantees

CArb: Postponing (or Eliminating) Switching

A set of schedules that provide some guarantees to l tasks while mitigate execution-time increase in higher-CL tasks

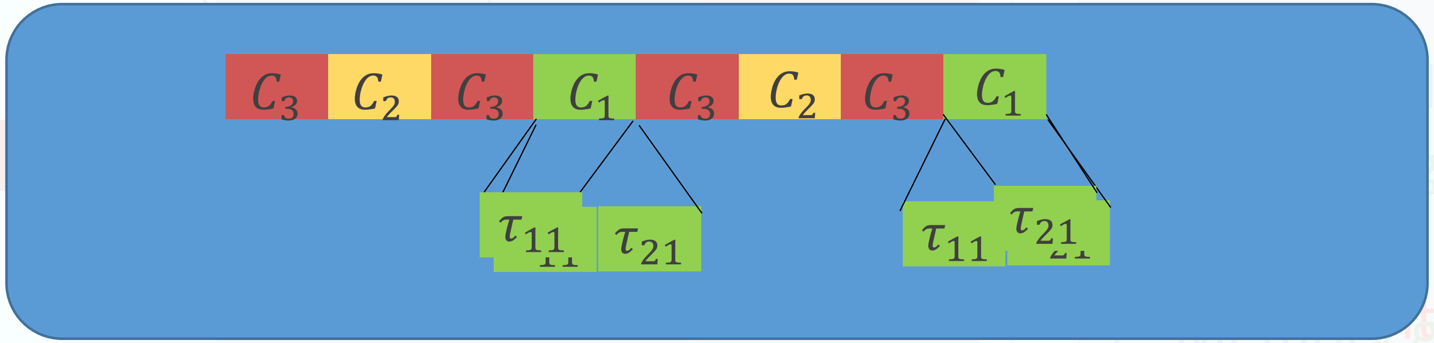


- ✓ Lower-critical tasks are not suspended
- ✓ Higher-critical tasks meet their requirement
- ✓ Postponed switching; thus decreasing overheads
- ✓ Memory guarantees for lower-critical tasks

CArb: Postponing (or Eliminating) Switching

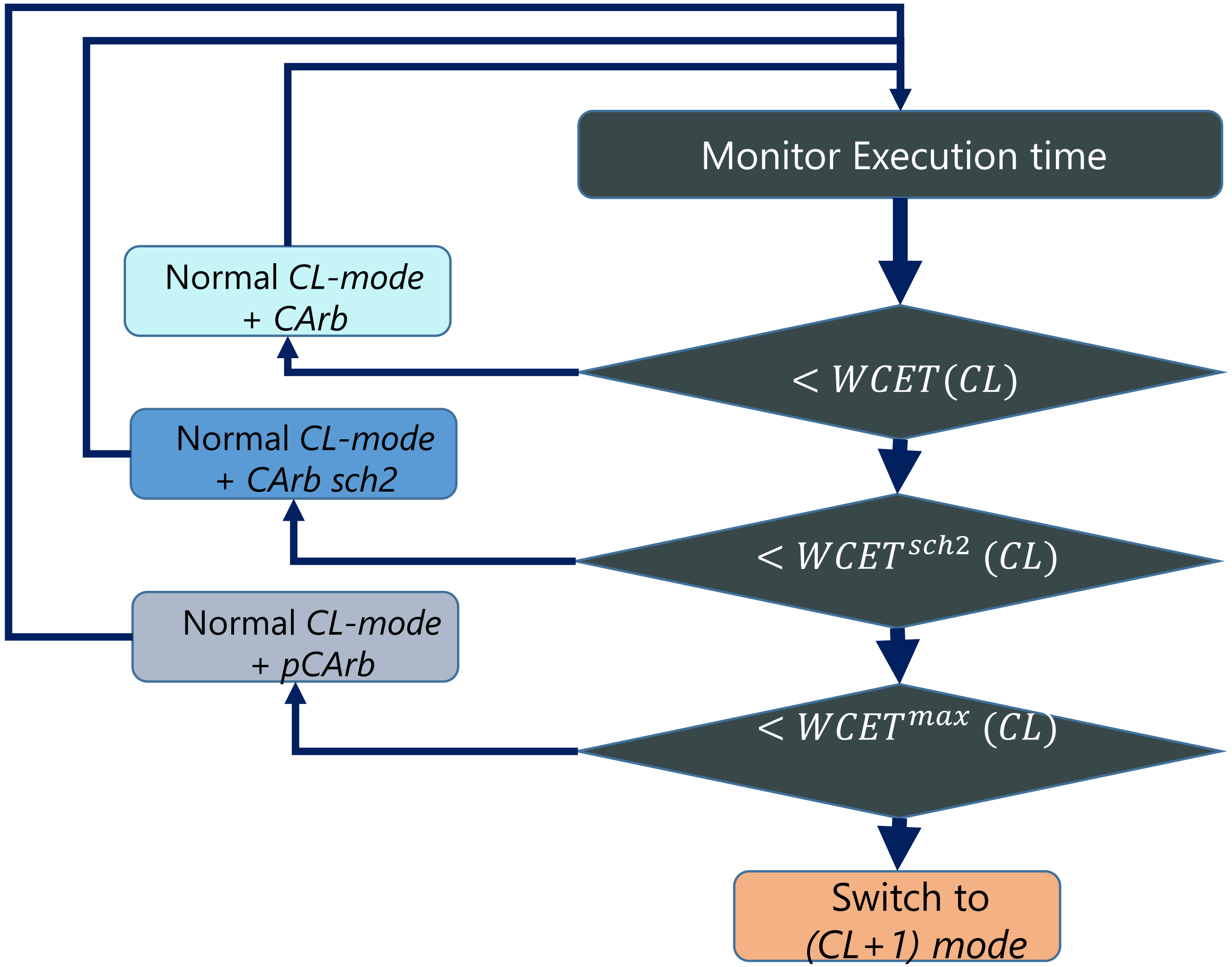
A set of schedules that provide some guarantees to l tasks while mitigate execution-time increase in higher-CL tasks

- ✓ Lower-critical tasks are not suspended
- ✓ Higher-critical tasks meet their requirement
- switching; thus overheads
- tical tasks receive no memory guarantees
- Memory guarantees for lower-critical tasks



CArb: Postponing (or Eliminating) Switching

Timing Interference



Monitor Execution time

$< WCET(CL)$

$< WCET^{sch2}(CL)$

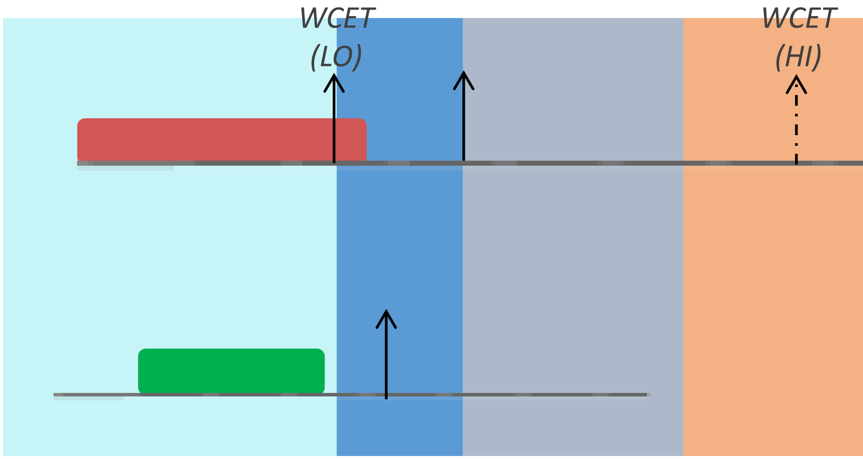
$< WCET^{max}(CL)$

Normal *CL-mode* + *CArb*

Normal *CL-mode* + *CArb sch2*

Normal *CL-mode* + *pCArb*

Switch to *(CL+1) mode*



CArb: Postponing (or Eliminating) Switching

MPSoC-Based MCS: Four Aspects

- Traditional MCS Model ✓
- Timing Interference ✓
- Data Sharing
- Security

Ignore

- Adopts an independent-task model → No communication amongst tasks

Prevent

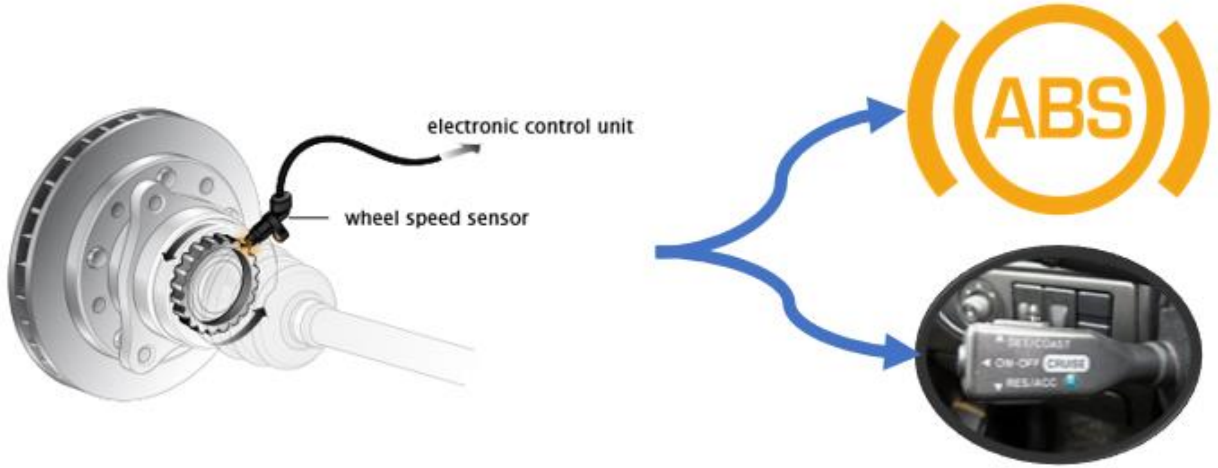
- Enforcing complete isolation between tasks.
  - At the shared cache: strict cache partitioning and coloring
  - At the DRAM: bank privatization

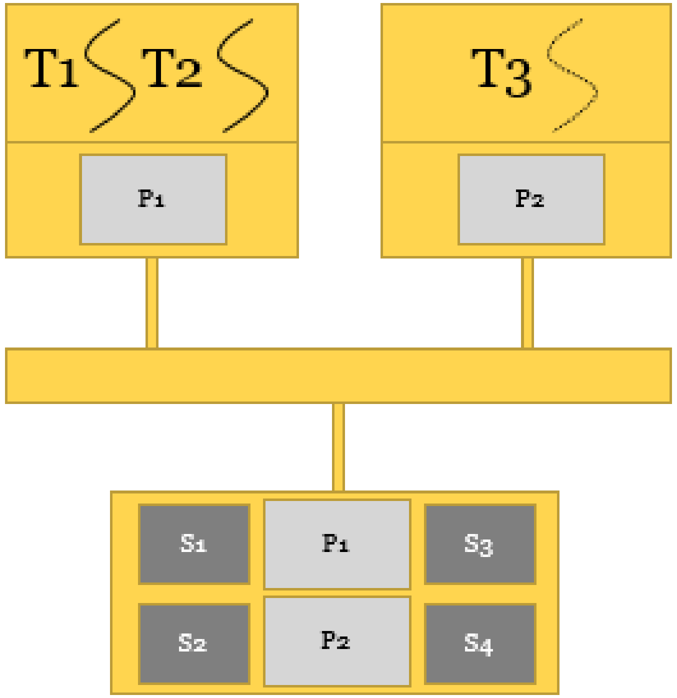
Common Approach

Common Approach

Data Sharing

- May result in a poor memory or cache utilization
  - e.g.: a task has conflict misses, while other partitions may remain underutilized
- Does not scale with increasing number of cores
  - e.g.: number of PEs ≤ number of DRAM banks
- Not viable in emerging systems due to increased functionality and massive data





✓ Simpler timing analysis

✗ Hardware changes

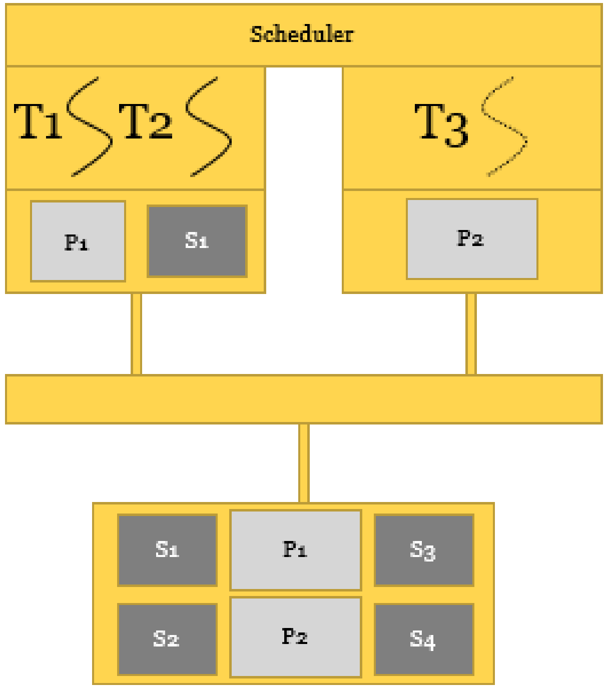
✗ Long execution time

Solution:

No caching of shared data

[Hardy et al., RTSS’09]

[Lesage et al., RTNS’10]



- ✓ Private cache hits on shared data
- ✓ No hardware changes
- ✗ Limited multi-core parallelism
- ✗ Changes to OS scheduler

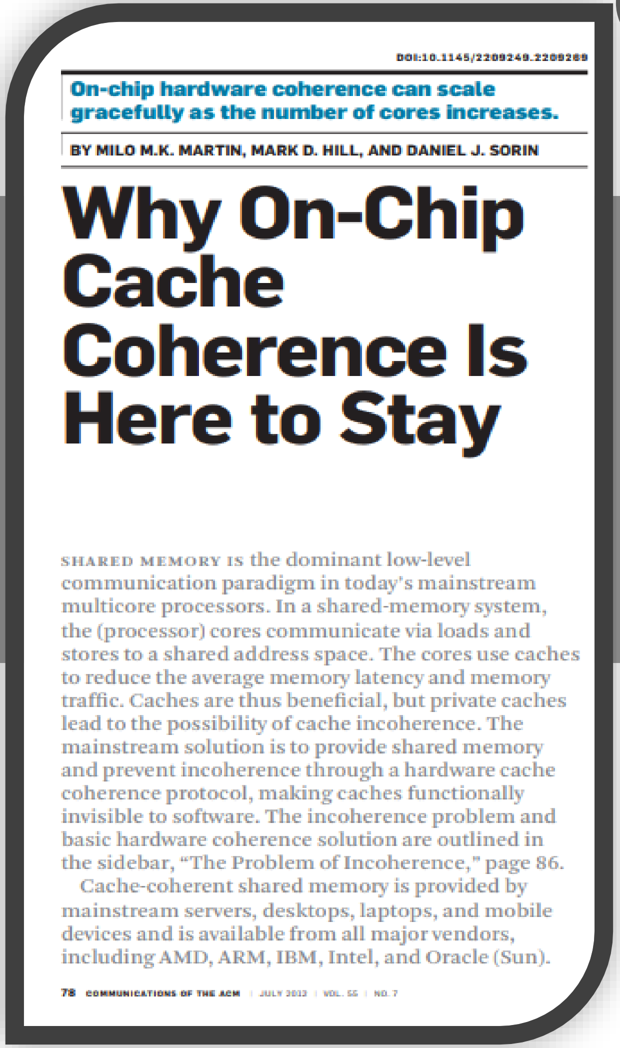
Another Solution:

Task scheduling on shared data

[Calandrino and Anderson, ECRTS'09]

[Chisholm et al., RTSS'16]

The mainstream solution is to provide shared memory and prevent incoherence through a hardware cache coherence protocol, making caches functionally invisible to software.



DOI:10.1145/2209249.2209269

On-chip hardware coherence can scale gracefully as the number of cores increases.

BY MILO M.K. MARTIN, MARK D. HILL, AND DANIEL J. SORIN

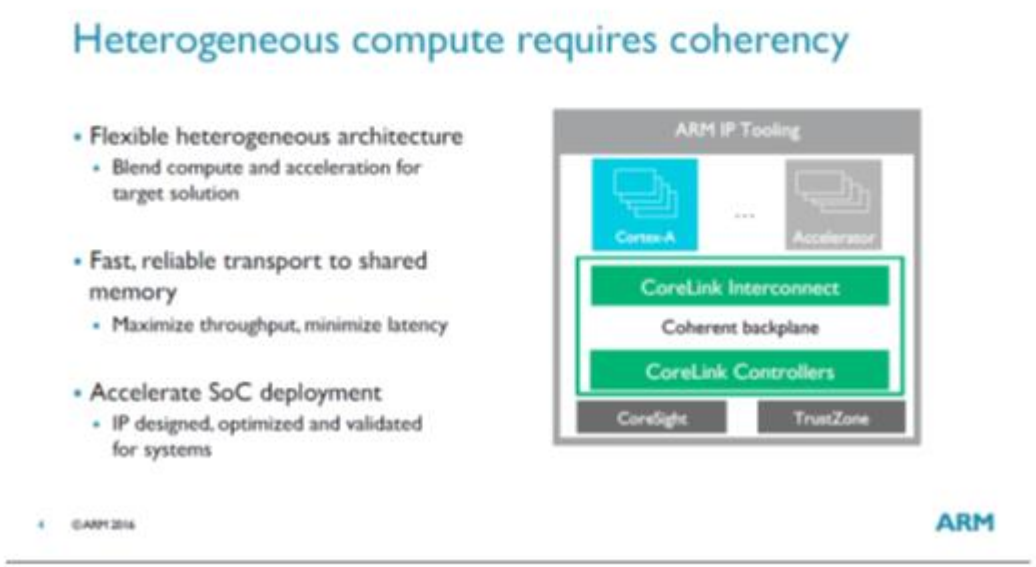
Why On-Chip Cache Coherence Is Here to Stay

SHARED MEMORY IS the dominant low-level communication paradigm in today's mainstream multicore processors. In a shared-memory system, the (processor) cores communicate via loads and stores to a shared address space. The cores use caches to reduce the average memory latency and memory traffic. Caches are thus beneficial, but private caches lead to the possibility of cache incoherence. The mainstream solution is to provide shared memory and prevent incoherence through a hardware cache coherence protocol, making caches functionally invisible to software. The incoherence problem and basic hardware coherence solution are outlined in the sidebar, "The Problem of Incoherence," page 86.

Cache-coherent shared memory is provided by mainstream servers, desktops, laptops, and mobile devices and is available from all major vendors, including AMD, ARM, IBM, Intel, and Oracle (Sun).

Coherence is the norm in COTS platforms

Data Sharing



Heterogeneous compute requires coherency

- Flexible heterogeneous architecture
  - Blend compute and acceleration for target solution
- Fast, reliable transport to shared memory
  - Maximize throughput, minimize latency
- Accelerate SoC deployment
  - IP designed, optimized and validated for systems

ARM IP Tooling

Cortex-A

...

Accelerator

CoreLink Interconnect

Coherent backplane

CoreLink Controllers

CoreSight

TrustZone

4 ©ARM 2016

ARM

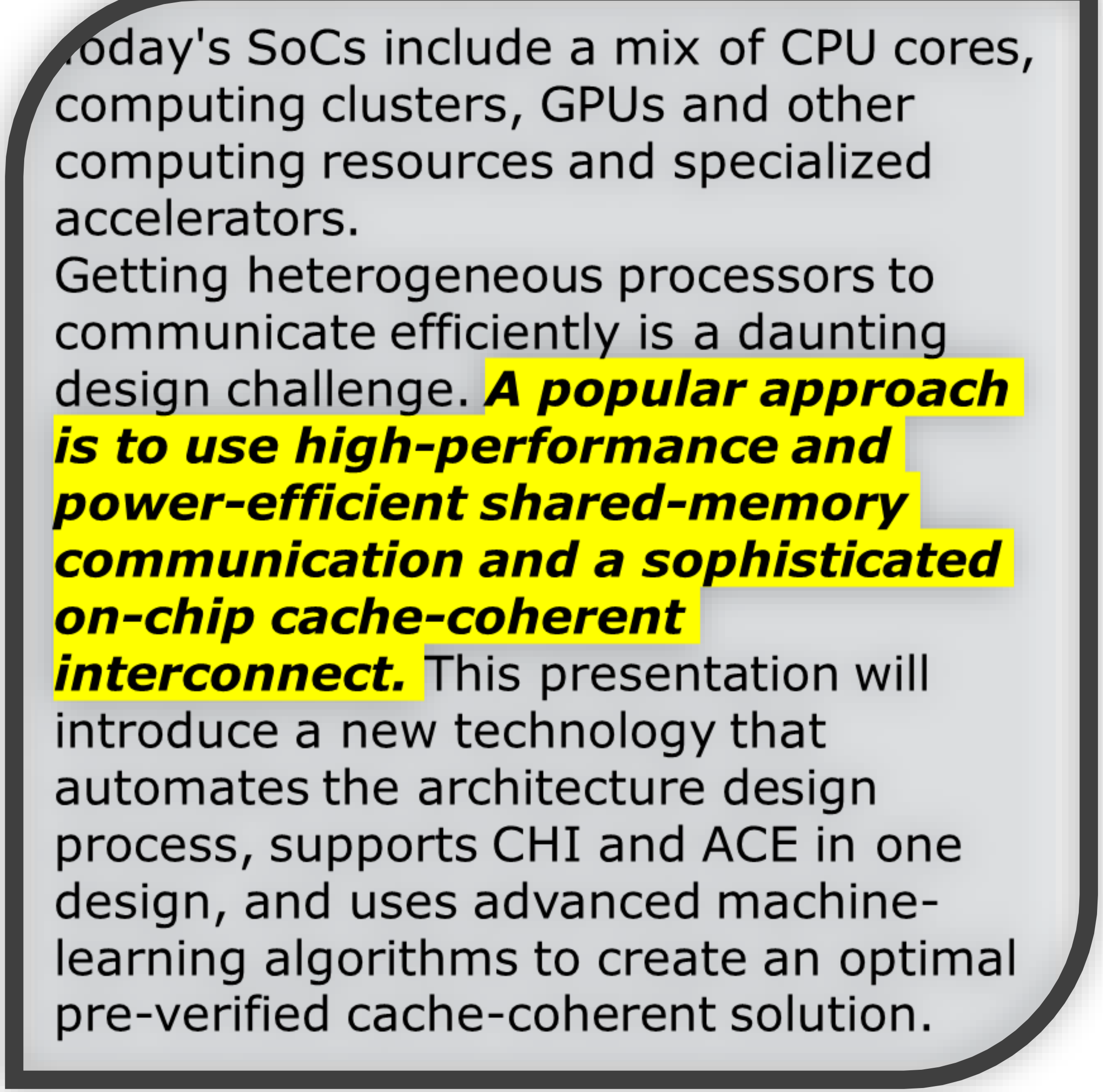
Coherence is the Industry’s Choice

Data Sharing

Coherency: The New Normal in SoCs

Anush Mohandass

anush@netspeedsystems.com



Today's SoCs include a mix of CPU cores, computing clusters, GPUs and other computing resources and specialized accelerators.

Getting heterogeneous processors to communicate efficiently is a daunting design challenge. ***A popular approach is to use high-performance and power-efficient shared-memory communication and a sophisticated on-chip cache-coherent interconnect.*** This presentation will introduce a new technology that automates the architecture design process, supports CHI and ACE in one design, and uses advanced machine-learning algorithms to create an optimal pre-verified cache-coherent solution.

Coherence is the Industry’s Choice

Data Sharing

Coherence is the Industry’s Choice

Data Sharing

ARTERIS IP

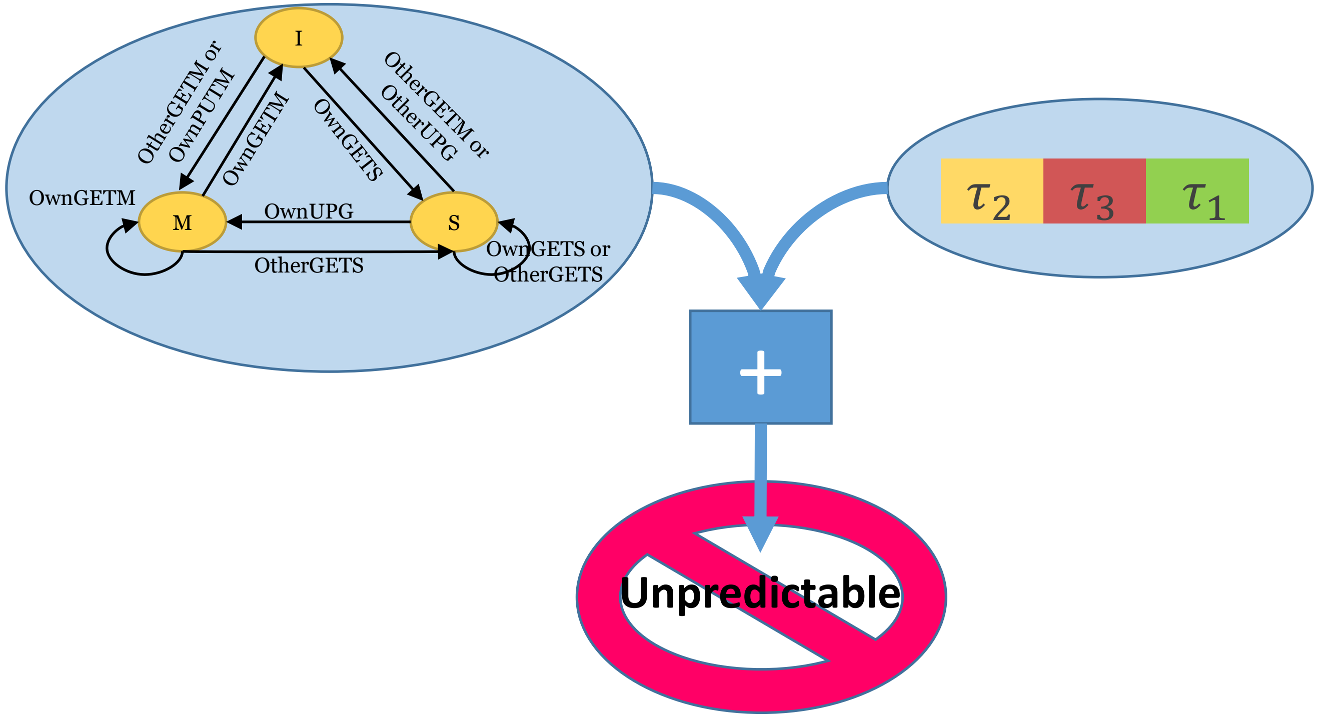
Enabling Mixed-Protocol Heterogeneous Cache Coherency and ISO 26262 Functional Safety

LINLEY PROCESSOR CONFERENCE
4 OCTOBER 2017

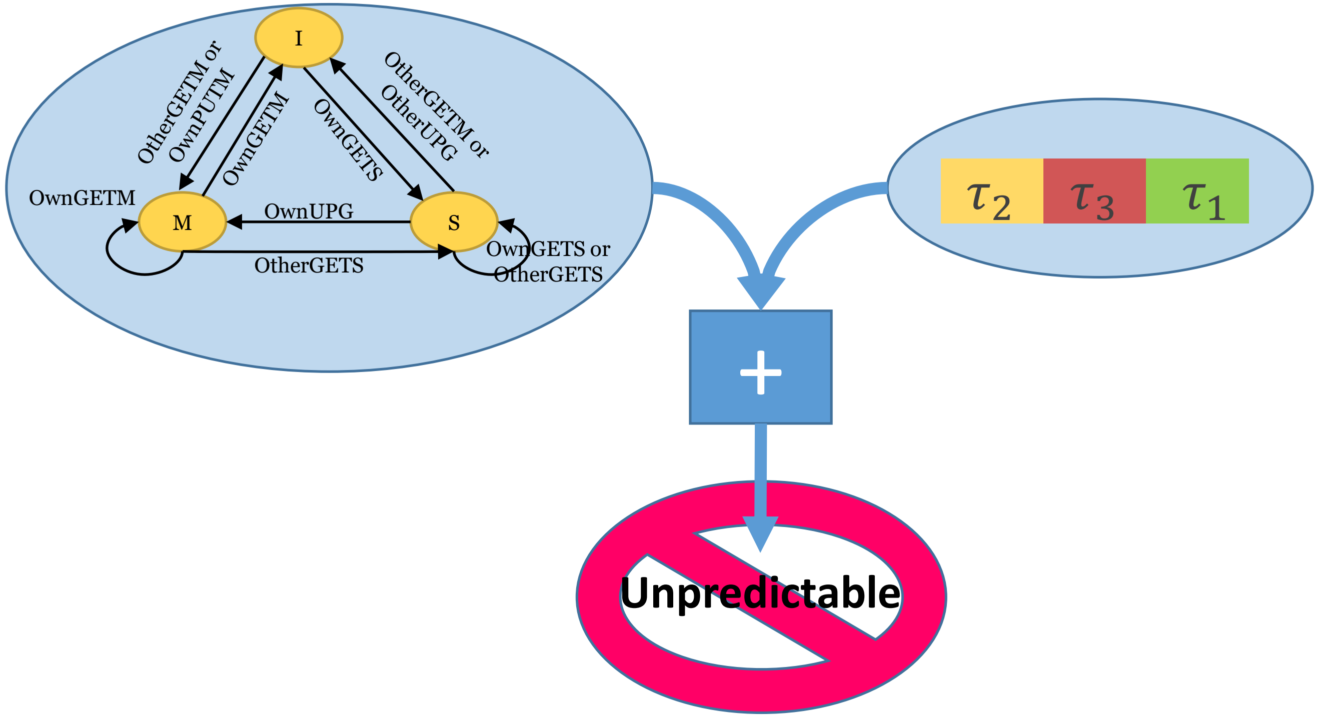
The Linley Group

SANJAY DESHPANDE
Director of Cache Coherency and
Interconnect Architecture

Autonomous driving requirements are mandating the simultaneous use of multiple types of processing units to efficiently execute sophisticated image processing, sensor fusion, and machine learning/AI algorithms.
This presentation introduces ***new coherency platform technology that enables the integration of heterogeneous cache coherent hardware accelerators and CPUs***, using a mixture of ARM ACE, CHI, and CHI Issue B protocols, into systems that meet both the requirements of high compute performance and ISO 26262-compliant functional safety.

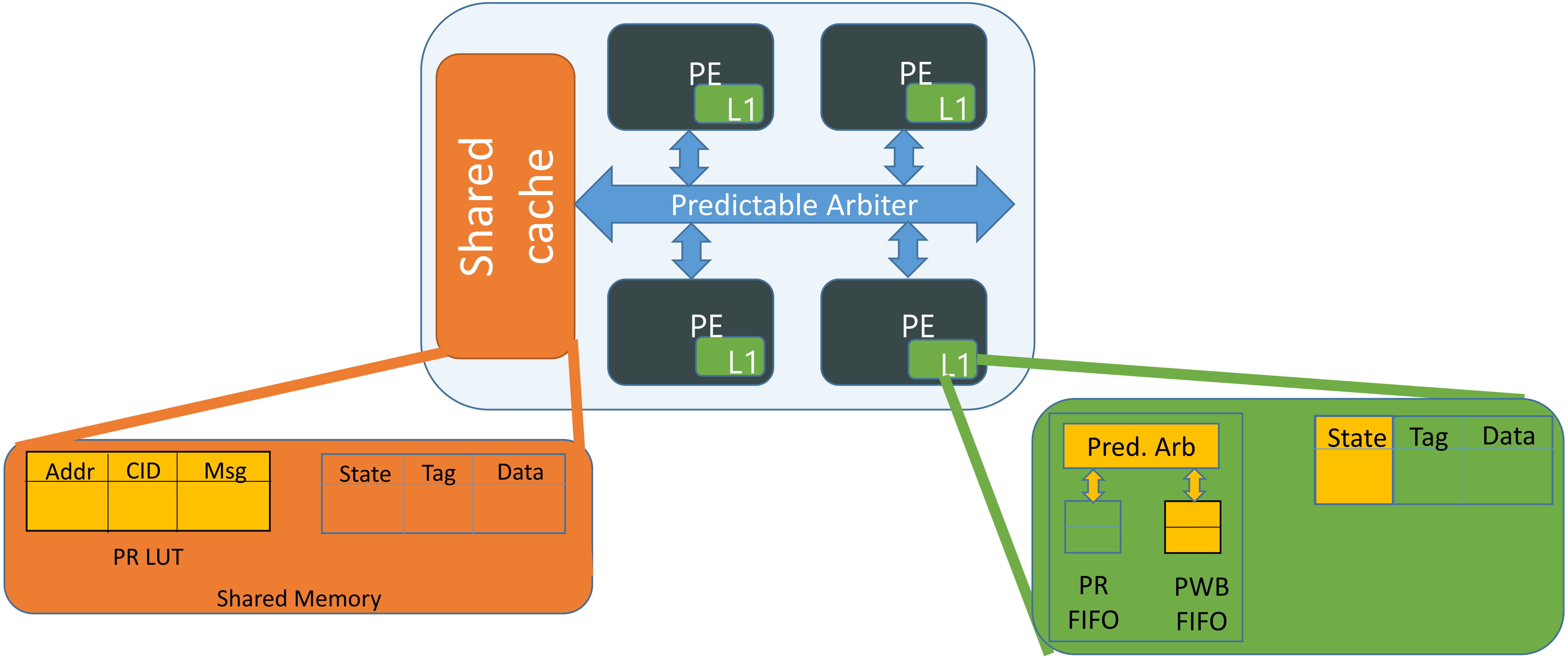


Unpredictability in Sharing Data



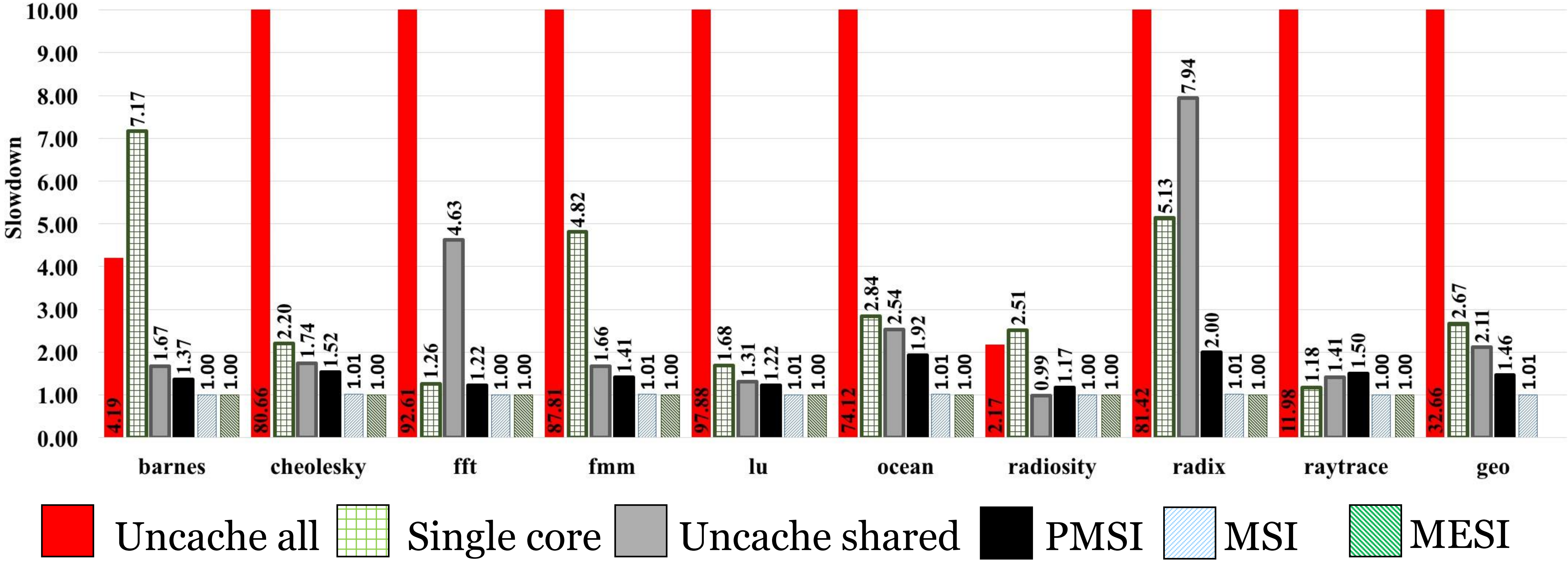
- ✘ Inter-core coherence interference on same cache line
- ✘ Inter-core coherence interference on different cache lines
- ✘ Inter-core coherence interference due to write hits
- ✘ Intra-core coherence interference

Unpredictability in Sharing Data



PMSI: Predictable Cache Coherence

Data Sharing



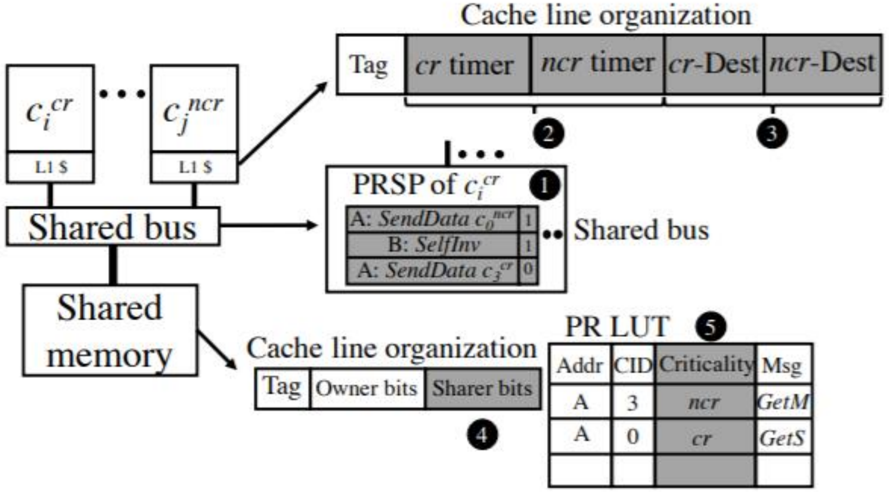
Performance Gains of Coherence

Data Sharing

HourGlass: Cache Coherence for MCS

Data Sharing

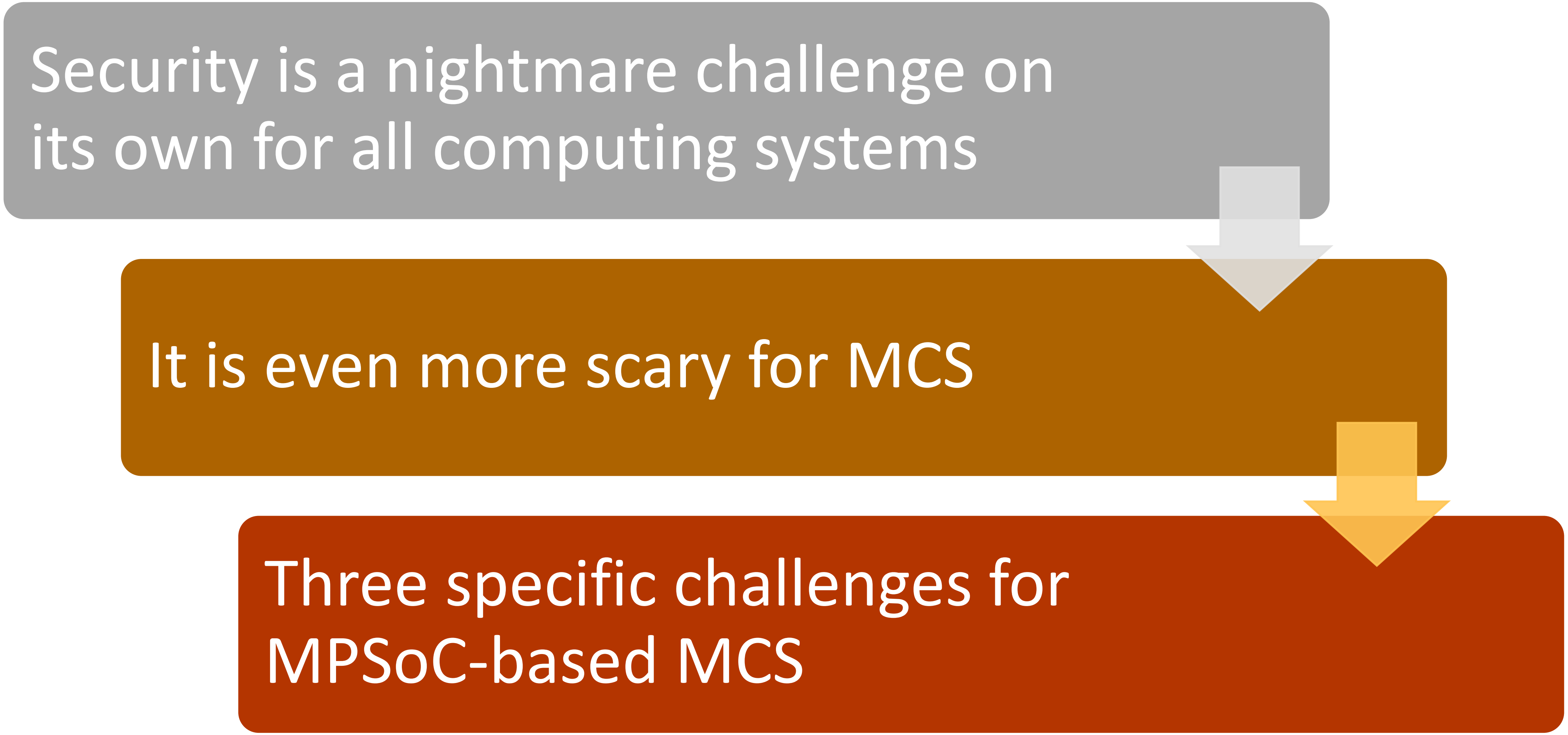
- **Time-based Cache Coherence**
  - Configurable timers for critical/non-critical cores
- **Fixed Priority Arbitration**
  - If both critical and non-critical requesting same cache line → critical gets it
- **Allows for simultaneous data sharing**
  - Both intra- and inter-criticality
- **Bounds WCL for critical cores while improving the BW of non-critical cores**



Addr	CID	Criticality	Msg
A	3	*ncr*	*GetM*
A	0	*cr*	*GetS*

MPSoC-Based MCS: Four Aspects

- Traditional MCS Model ✓
- Timing Interference ✓
- Data Sharing ✓
- Security



Security is a nightmare challenge on its own for all computing systems

It is even more scary for MCS

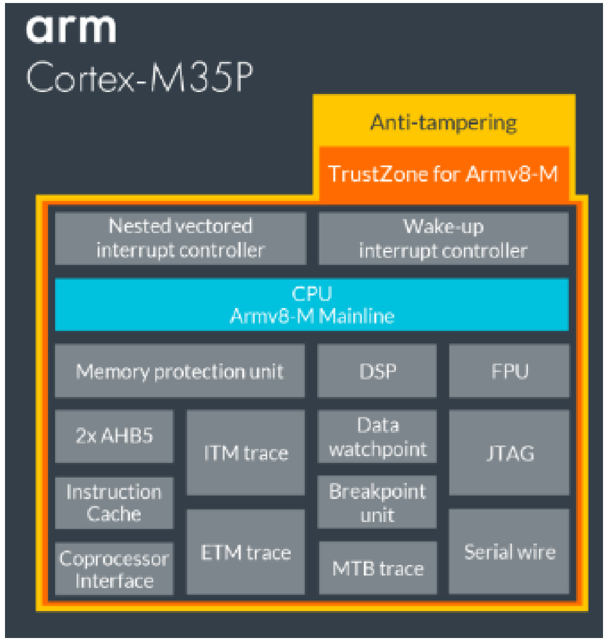
Three specific challenges for MPSoC-based MCS

Security

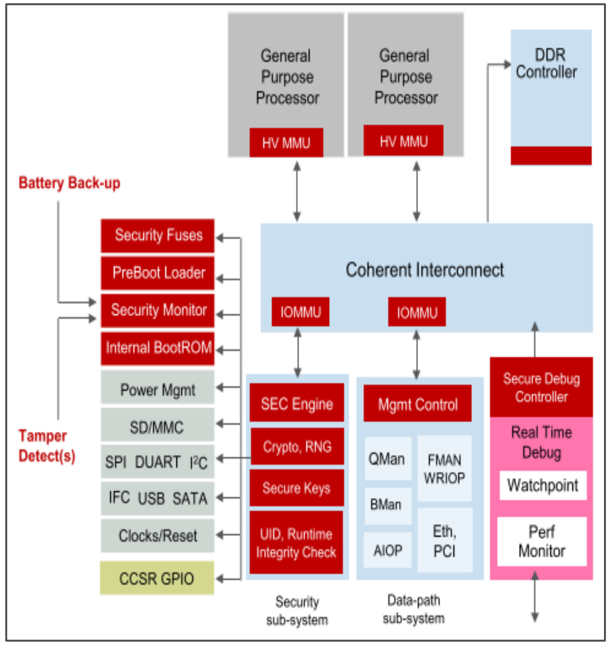
MPSoCs Opportunities

MPSoCs open the door for customized solutions

ARM Cortex-M35P with Physical Security



NXP's QorIQ SoC with Trust Architecture



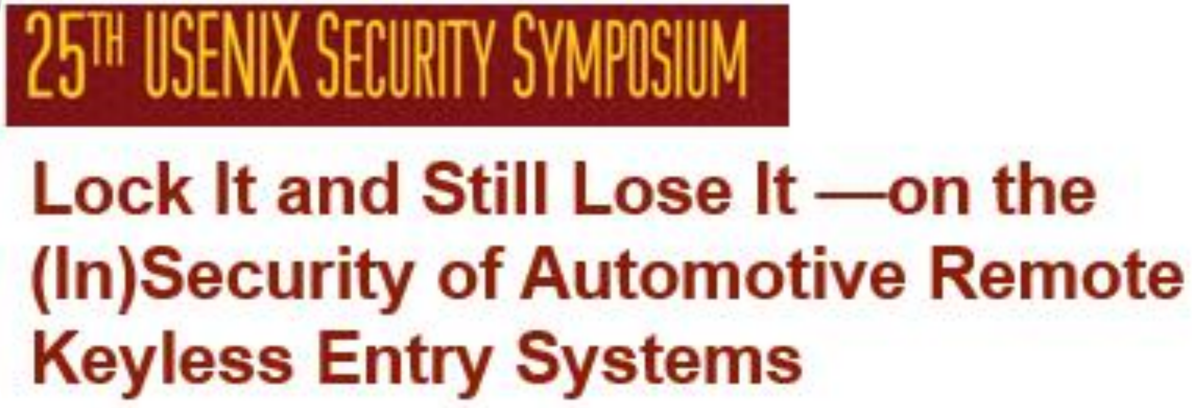
MPSoCs Challenges

CYBER-PHYSICAL NATURE OF MCS

HETEROGENEITY OF MPSOCS

SHARED COMPONENTS (AGAIN!)

Cyber-physical Nature



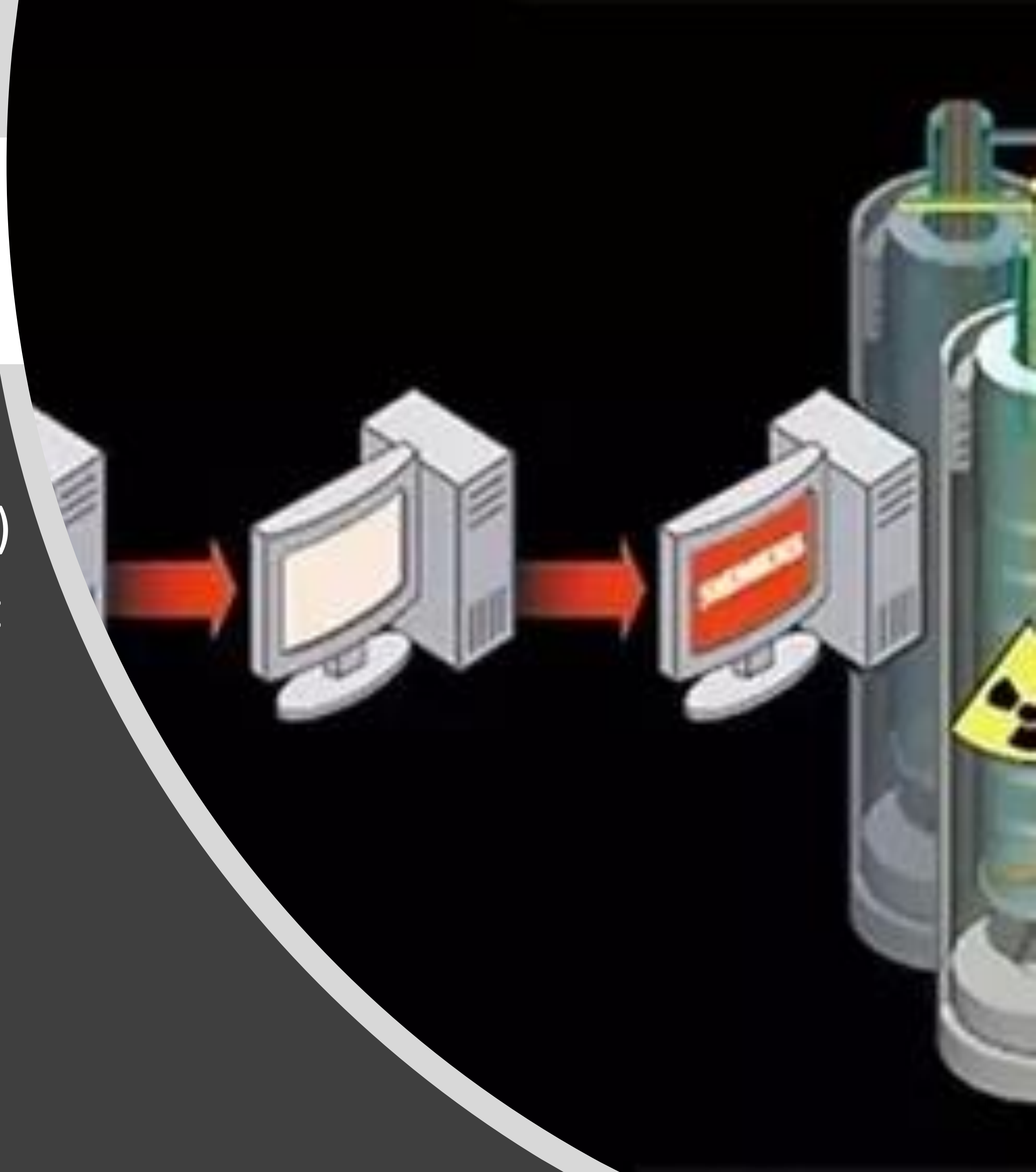
25TH USENIX SECURITY SYMPOSIUM

Lock It and Still Lose It —on the (In)Security of Automotive Remote Keyless Entry Systems

- MCS manage sensitive tasks in critical domains: power grids, cars, factories, nuclear plants
- Any security breach could lead to catastrophic consequences
- Hackers gained access to locked cars by only eavesdropping a single signal from the original remote keyless entry unit of the car

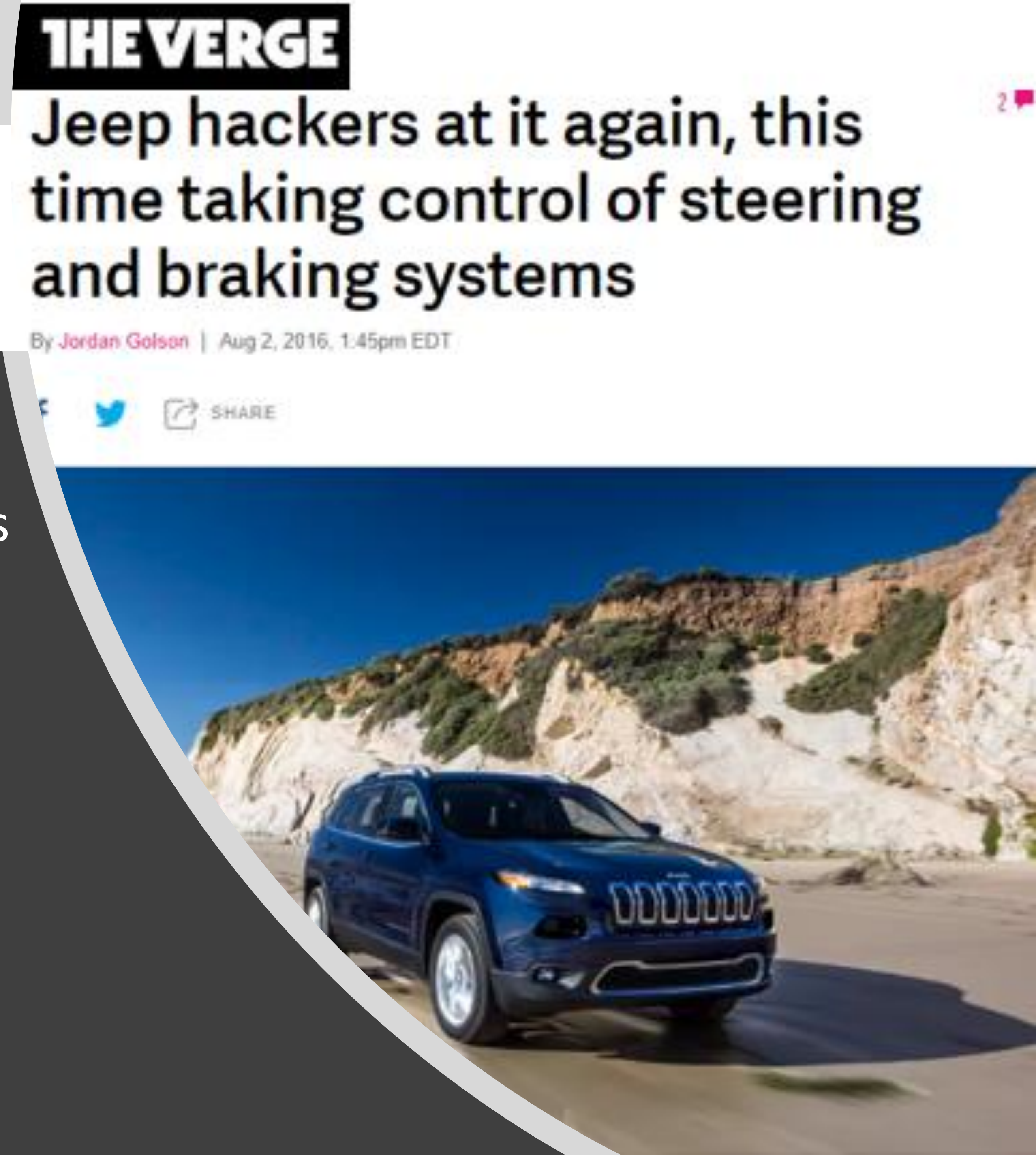
Heterogeneity of MPSoCs

- Each PE can be a 3rd-party IP (40% at Intel!)
- PEs share system components and interact with each other →new across-PEs threats
- Stuxnet attack exploited the authentication of the Siemens programmable logic controller to access a Windows machine



Shared hardware components in MPSoCs

- Historically, security was not considered as a concern for MCS because of isolation
- Not the case anymore
- Researchers were able to control sensitive (considered secure) engine control by compromising the (considered insecure) radio unit
  - Reason? Sharing the CAN



THE VERGE

Jeep hackers at it again, this time taking control of steering and braking systems

By Jordan Golson | Aug 2, 2016, 1:45pm EDT

Possible Directions

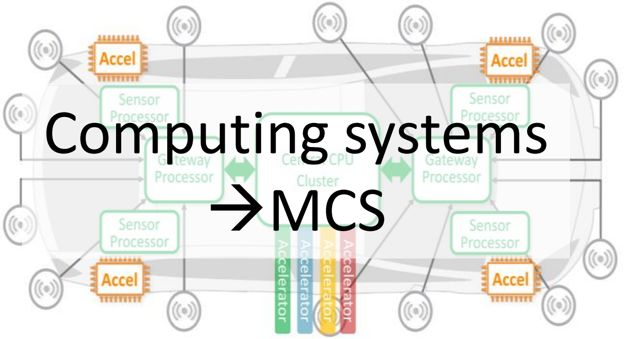
- Identifying new vulnerabilities of MPSoCs, which did not exist in traditional platforms

- Developing cost- and performance-effective methodologies to prevent or mitigate them

- Adopting security as a first-class citizen in

  designing MPSoCs for MCS (secure-by design concept).

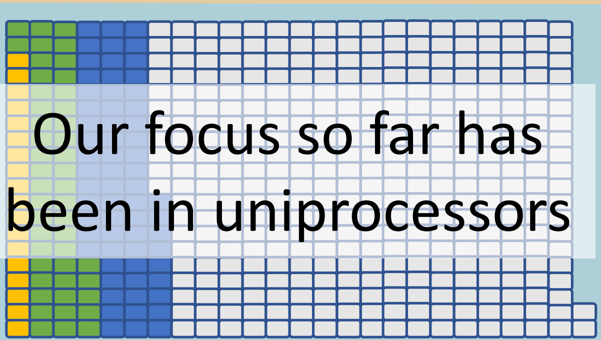
  Scheduling techniques



Computing systems →MCS

ARM Cortex A53 Application Processors 64-bit Quad-Core with Virtualization
Power Management Multiple Power Domains Power Gated Islands
ARM Cortex R5 Real-Time Processors
Safety & Reliability
ISO
mali Graphics/Video ARM Mali-400MP
H.265 HEVC
Security Information Assurance, TrustZone Key and Vault Management
UltraScale FPGA Logic UltraRAM, PCIe Gen4, 100G Ethernet, AMS
SDK Xilinx SDK
Runtime SW & Tools OS, RTOS, AMP, Hypervisor Development, Heterogeneous Debug, Hardware/Software Profiling & Performance Analysis
High Speed Peripherals USB 3.0, PCIe Gen2, GbE SATA3.0, DisplayPort

SoC is the choice for Automotive and IoT



Our focus so far has been in uniprocessors

Traditional MCS Model

Timing Interference

Data Sharing

Security