

# Develop Vehicle Control Systems As CPS



For Next Generation Automobiles



**Shige Wang**

GENERAL MOTORS  
GLOBAL RESEARCH & DEVELOPMENT



# Topics

- Overview of NextGen vehicle control system
  - why is it CPS
- Development process for massive production
  - how is it design and developed: EE system
- Technologies supporting NextGen vehicle control
  - what enables it: real-time embedded systems
- Challenges with new technologies
  - what are missing: parallelism, data processing

# Cyber-Physical Systems

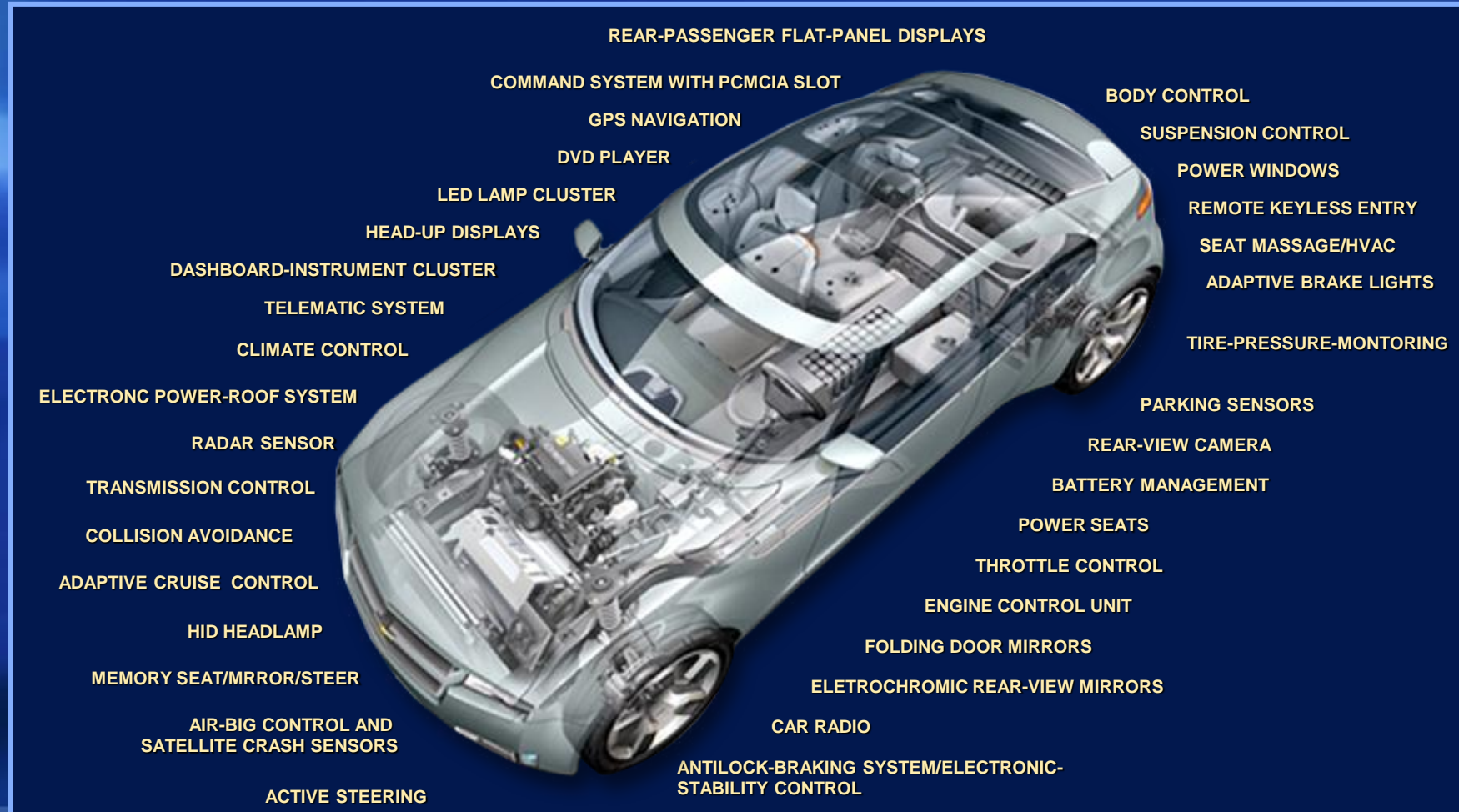
- **Cyber** – computation, communication, and control that are discrete, logical, and switched
- **Physical** – natural and human-made systems governed by the laws of physics and operating in continuous time
- **Cyber-Physical Systems** – systems in which the cyber and physical systems are tightly integrated at all scales and levels
  - Change from cyber merely appliquéd on physical
  - Change from physical with COTS “computing as parts” mindset
  - Change from ad hoc to grounded, assured development

“CPS will transform how we interact with the physical world just like the Internet transformed how we interact with one another.”

*Source: Dr. Gill presentation at NSF CPS PI meeting*

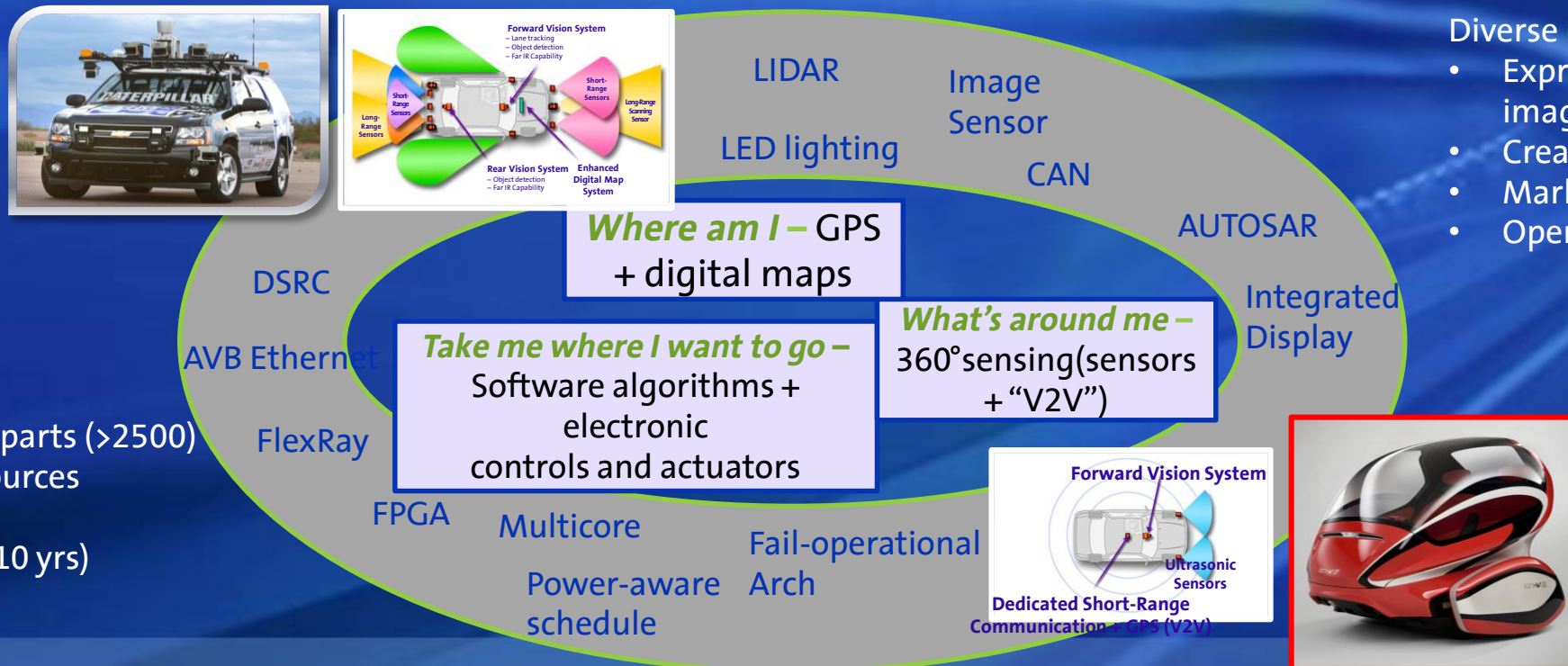


# Controls in Typical Vehicle

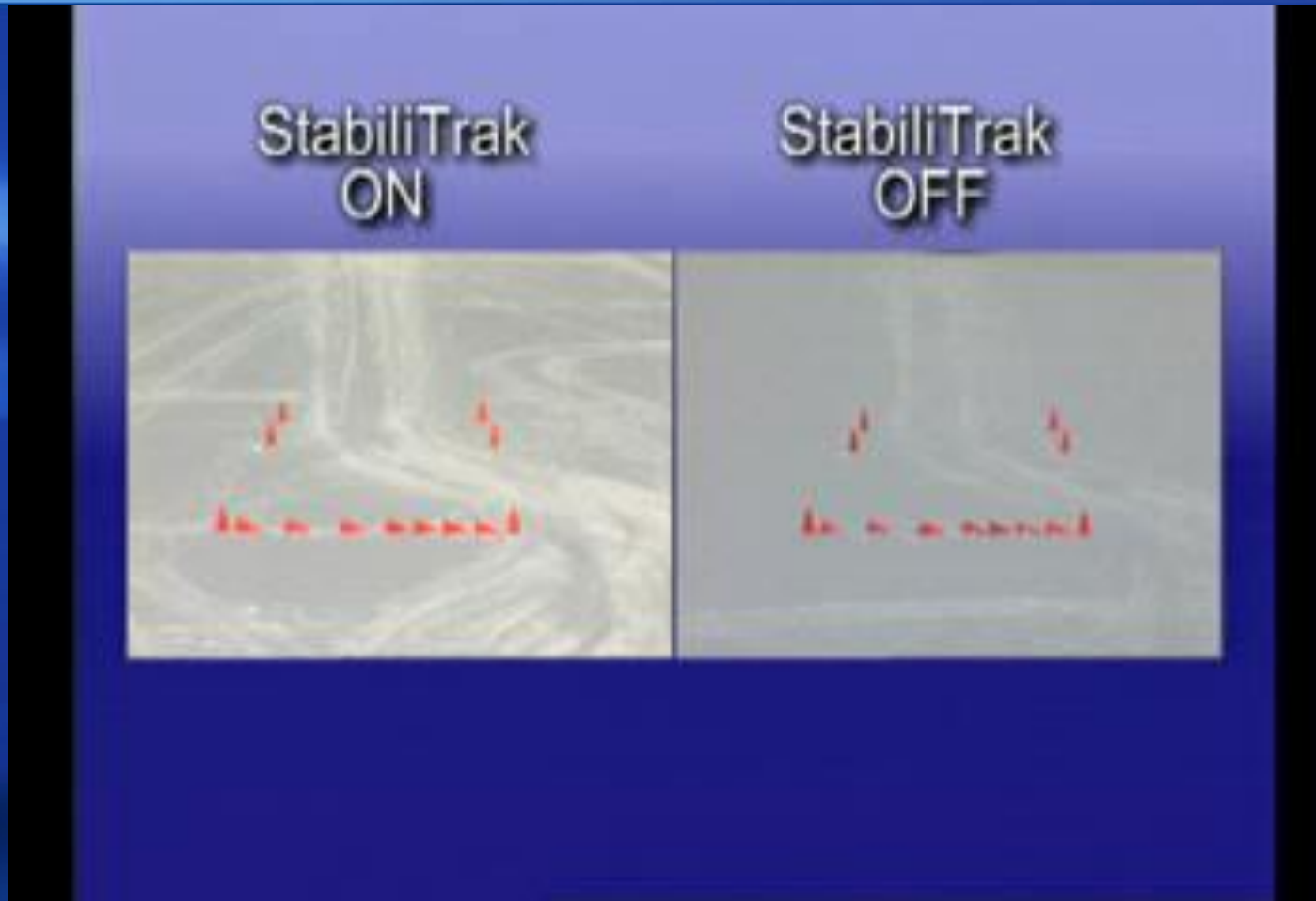


# Vehicle Control System as CPS

- Electrical and electronics replacing mechanic parts
- Standalone system to connected
- Rely on driver to autonomous driving
- Fixed configuration to tailor for different uses



# ***CPS Example: Electronic Stability Control***



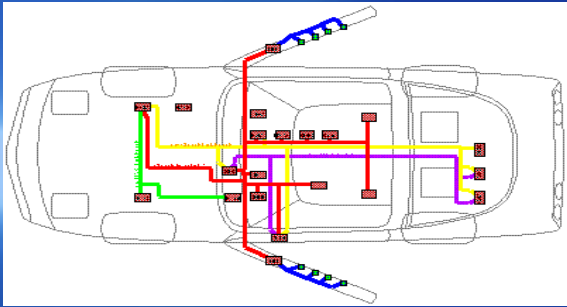
## ***With Technology Advancement***

Next generation vehicles should be smart and adaptive

- Energy-efficient propulsion
- Vehicle connectivity – both in-vehicle and V2X
- Active safety – driving assistance to autonomy
- Personalized – learn driving styles
- Self management of health



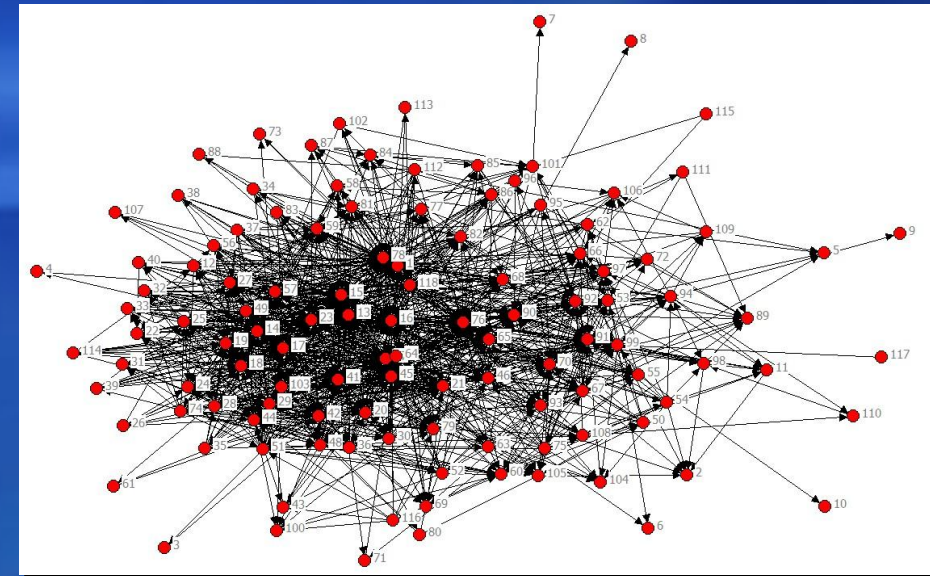
# Vehicle Control System: Complexity



- 70+ Electronic Control Units
- 10+ in-vehicle networks
- 100+ sensors and actuators

- 50+ customer features
- 16 domains, 88 subsystems
- Powertrain: 118 functions, 789 signal, 967 links

- 15 concurrent temporal development streams
- 300 hierarchical subsystems
- Thousands of variant features
- Millions of product instances
- Tens-of-thousands of unique product variants

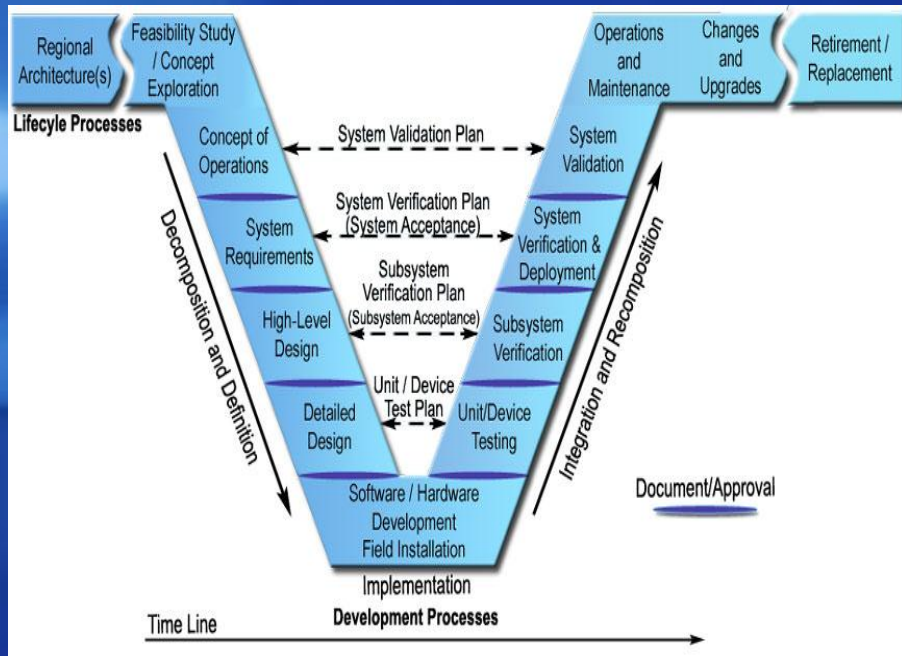


- Highly heterogeneous
- Mixed criticality, mixed intellectual property, mixed versions

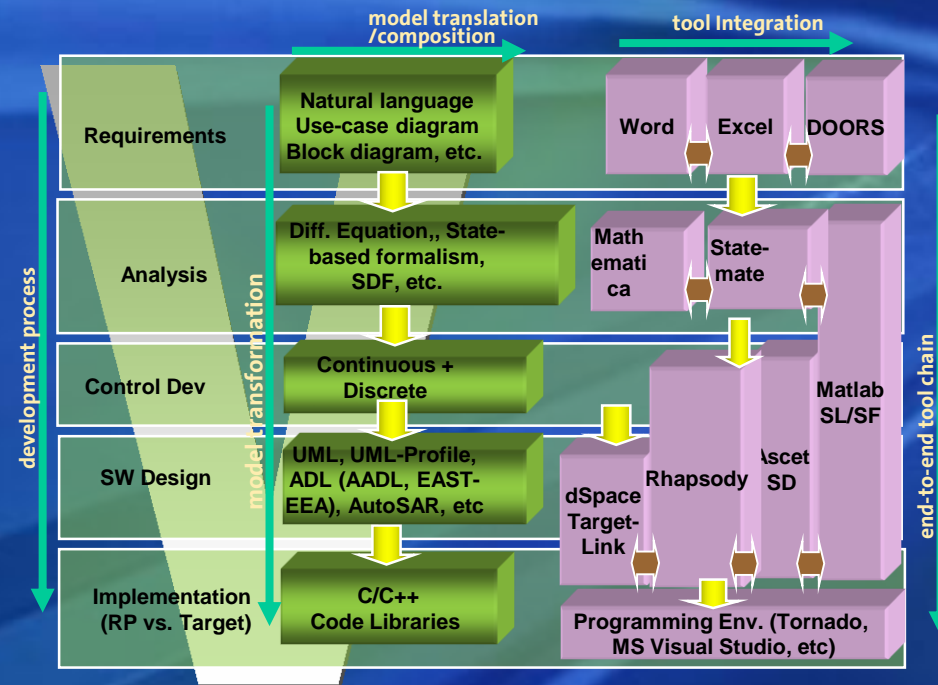
- Affordable, reliable, safe, and exciting
- Across a large volume with many variations
- Last long time in all conditions – climate, traffic, maintenance, driving habits, .....



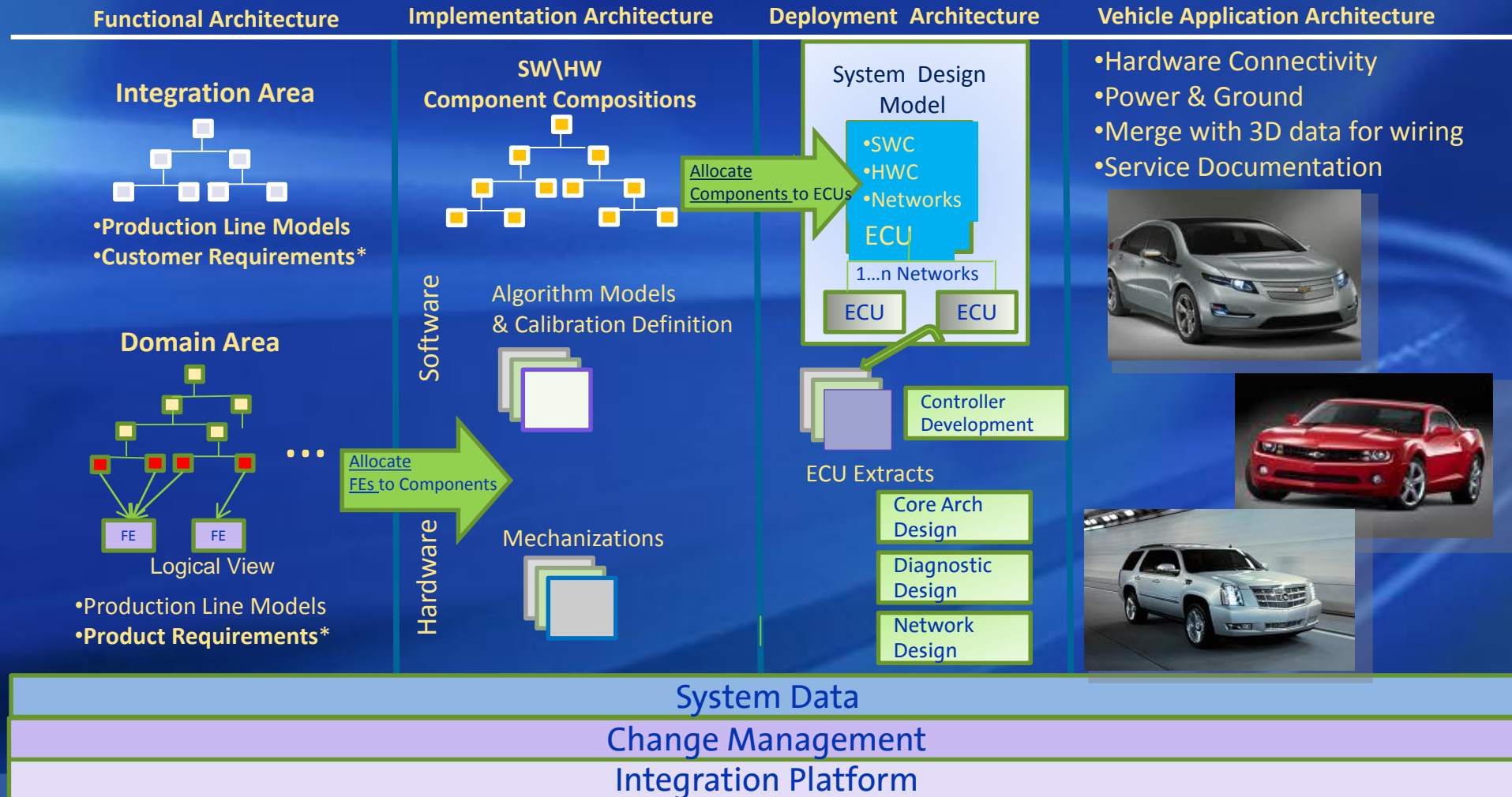
# Vehicle Control System Development Process



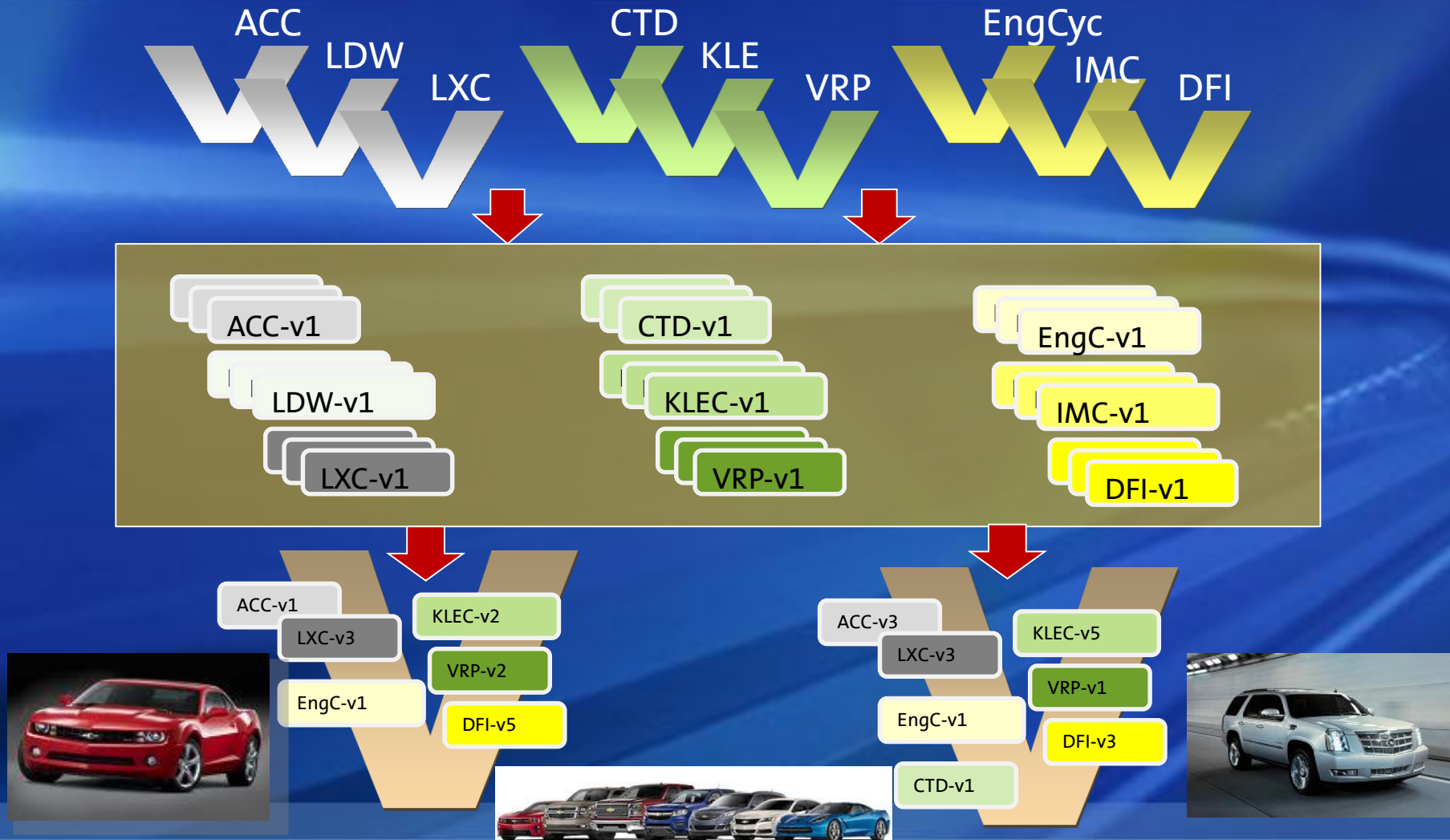
Multiple stages, multiple groups/organizations, multiple tools, multiple geographical locations



# Vehicle Control System Engineering



# Feature-Oriented Productline Development

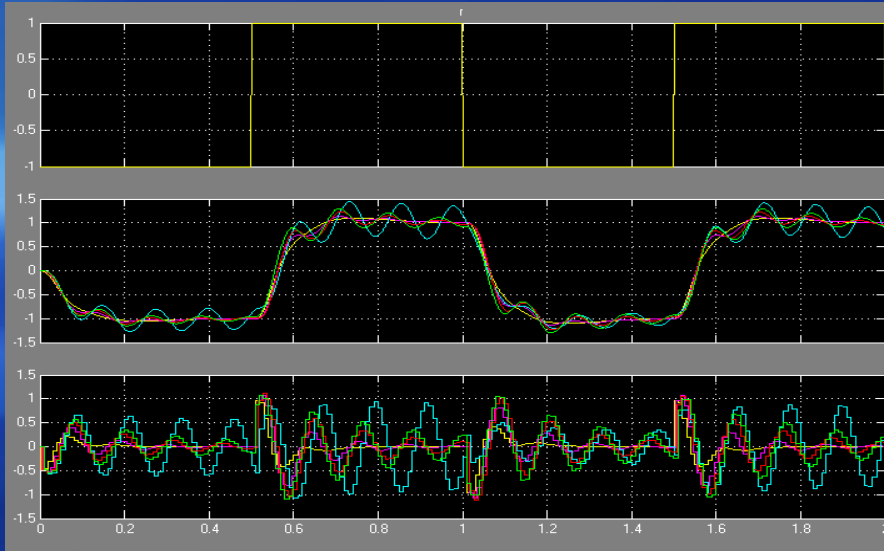




# ***Challenge: Software Implementation of Controls***

- **Different design principles and focuses**
  - Code generation in commercial tools limited to small component
- **Timing delays cause errors**
  - Mismatch implementation and model
  - Introduce timing jitters
  - Non-determinism across different configurations
- **Multitasking and multirate control**
  - Determine proper software tasks for control
  - Determine proper schedule of software tasks
- **More challenging – parallel programming and execution**
  - Resource sharing, data protection, synchronization, etc

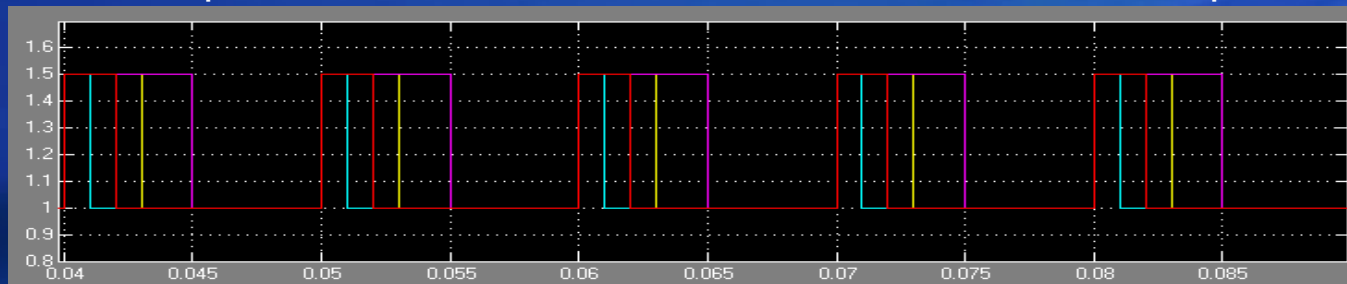
# PID Example: Error Propagation in Software



Per = 10, wc completion time = 1~5

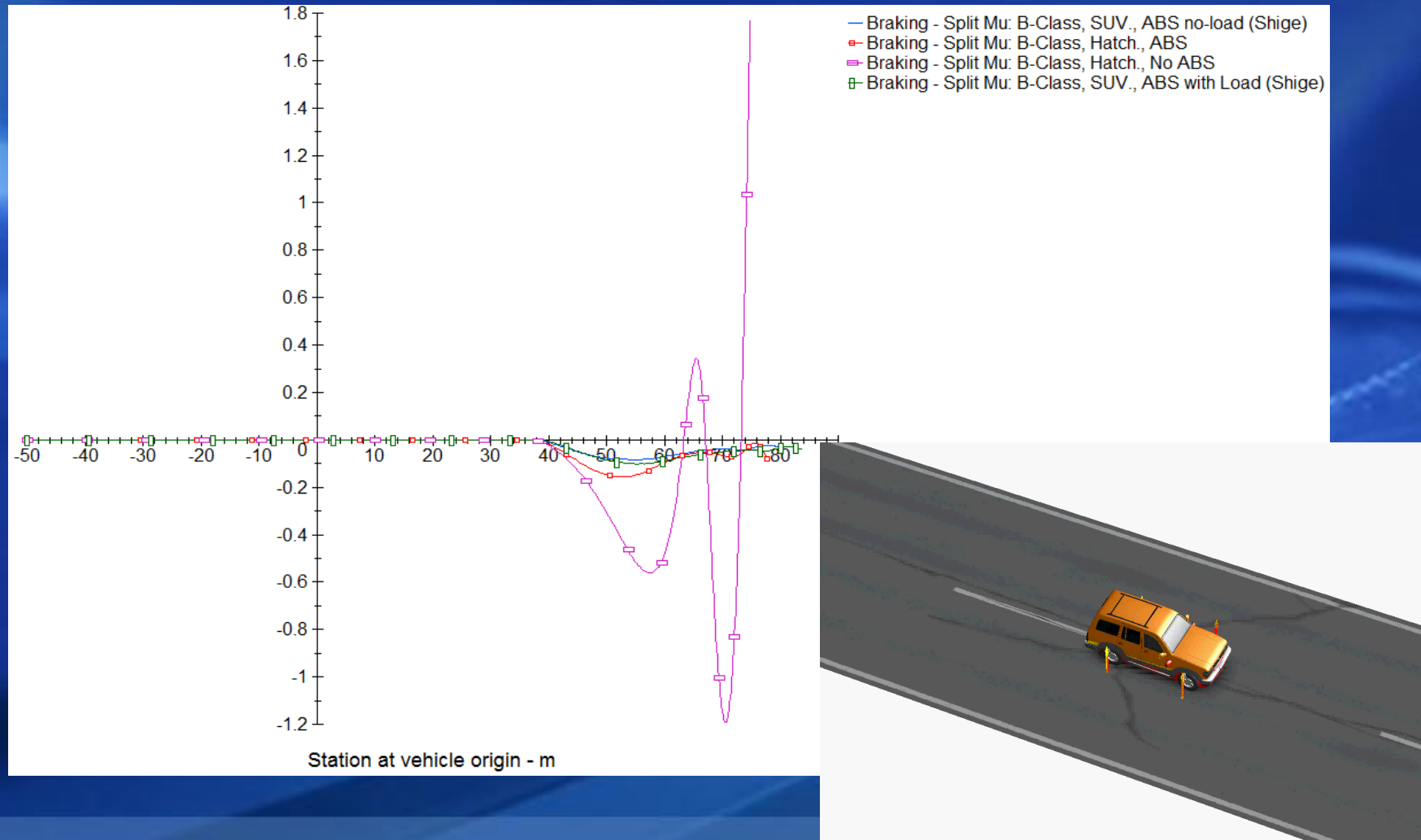


Per = 6, wc completion time = 1~5



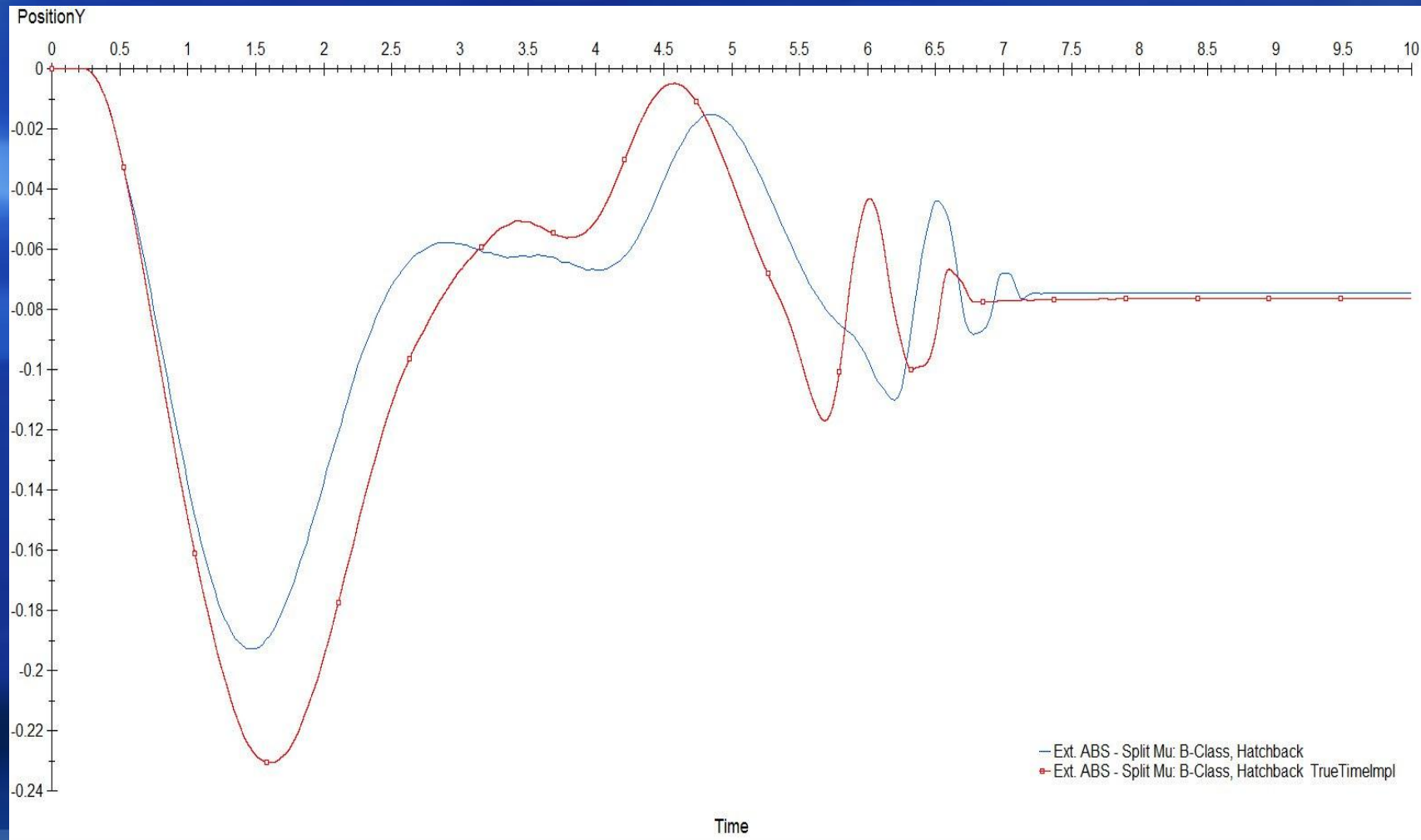
- There may be a correlation between sampling period and execution time

# ABS Example: Physical Variability Impact



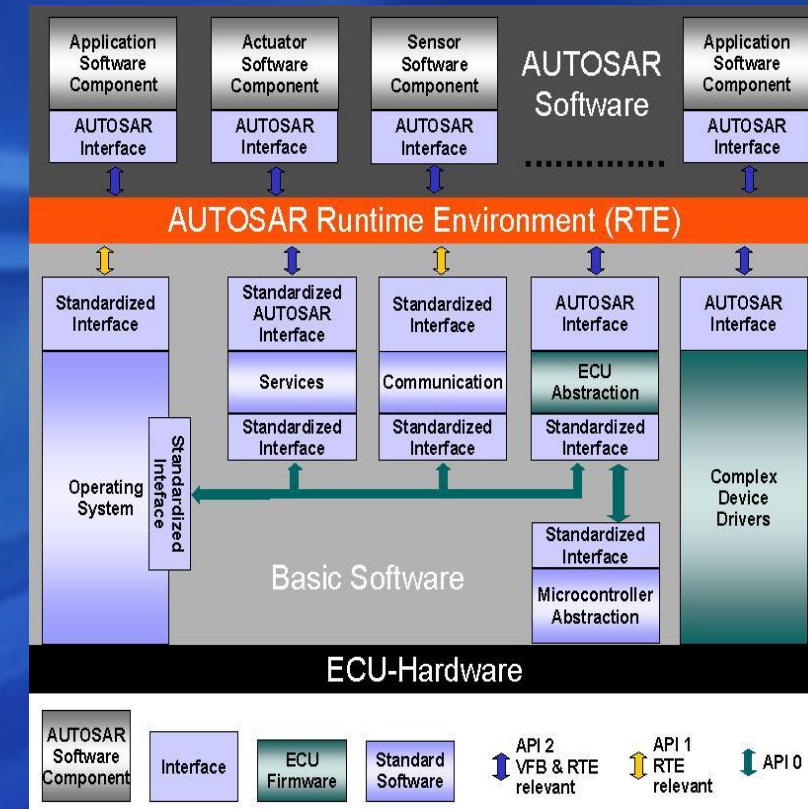
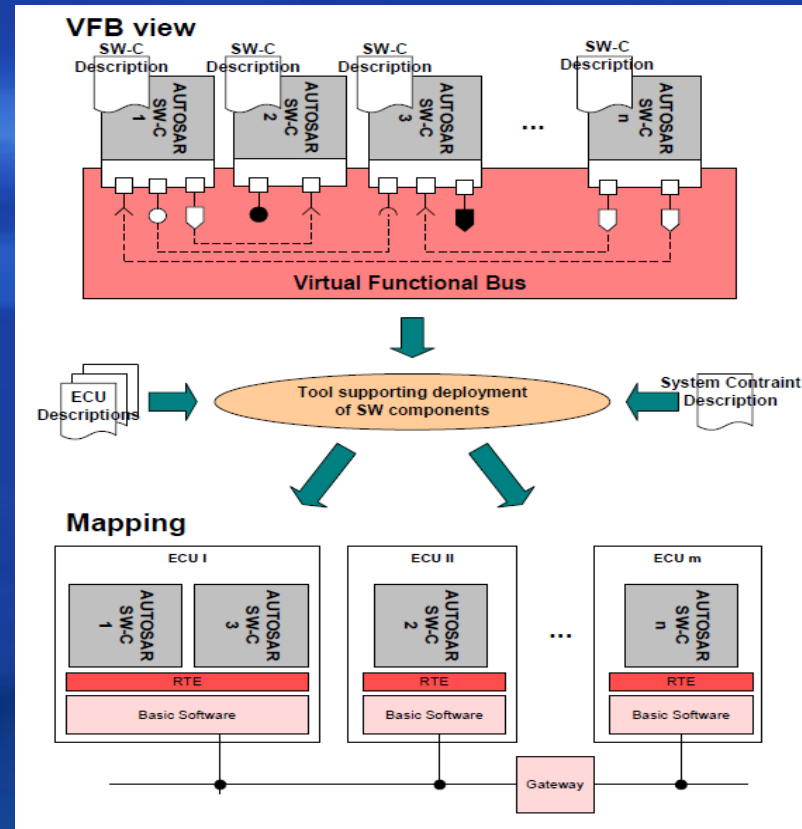


# ABS Example: Software Implementation Impact

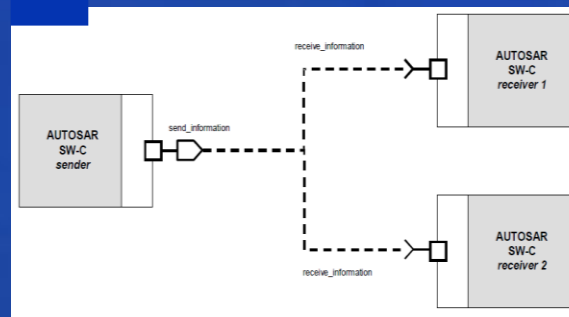
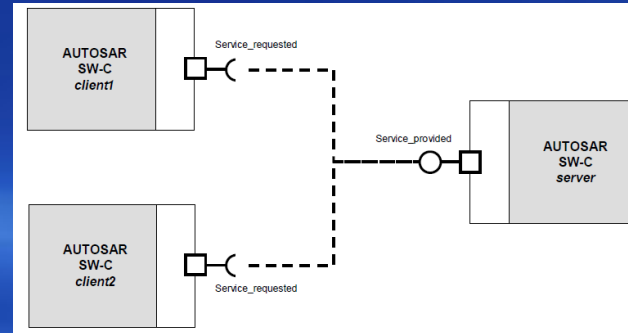
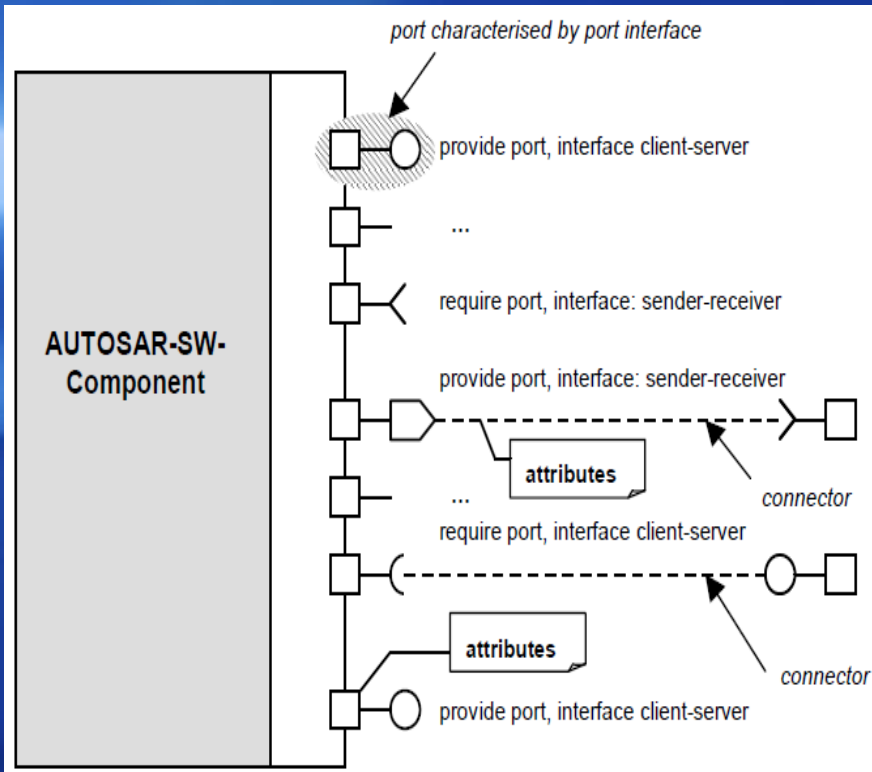


# Standardization: AUTOSAR

- SW-C**  
(application software components)
- VFB**  
(virtual functional bus)
- RTE**  
(run-time environment)
- BSW-M**  
(basic software modules)



# AUTOSAR Interfaces – System Composition



For Sender-Receiver ports:

- Initial value
- Queue length
- Explicit vs. implicit
- Acknowledgement
- Timeout

For Client-Server ports:

- Synchronous vs. asynchronous
- Timeout
- Queue length

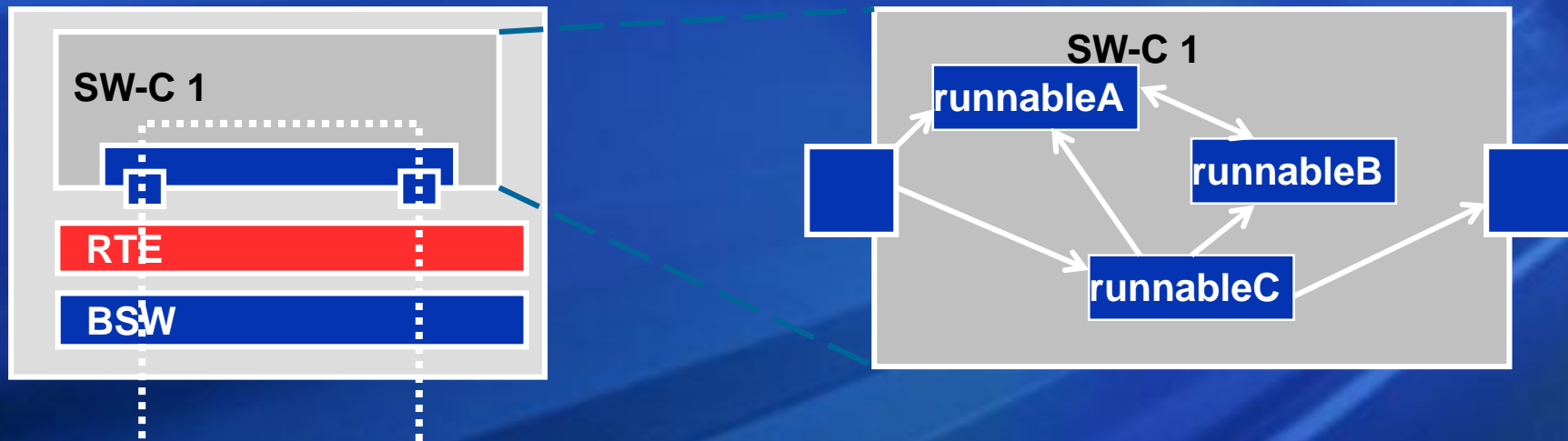
Components interact through ports:

- Sender-receiver: SWC-SWC, 1-many, async
- Client-server: SWC-BSW, many-1, sync

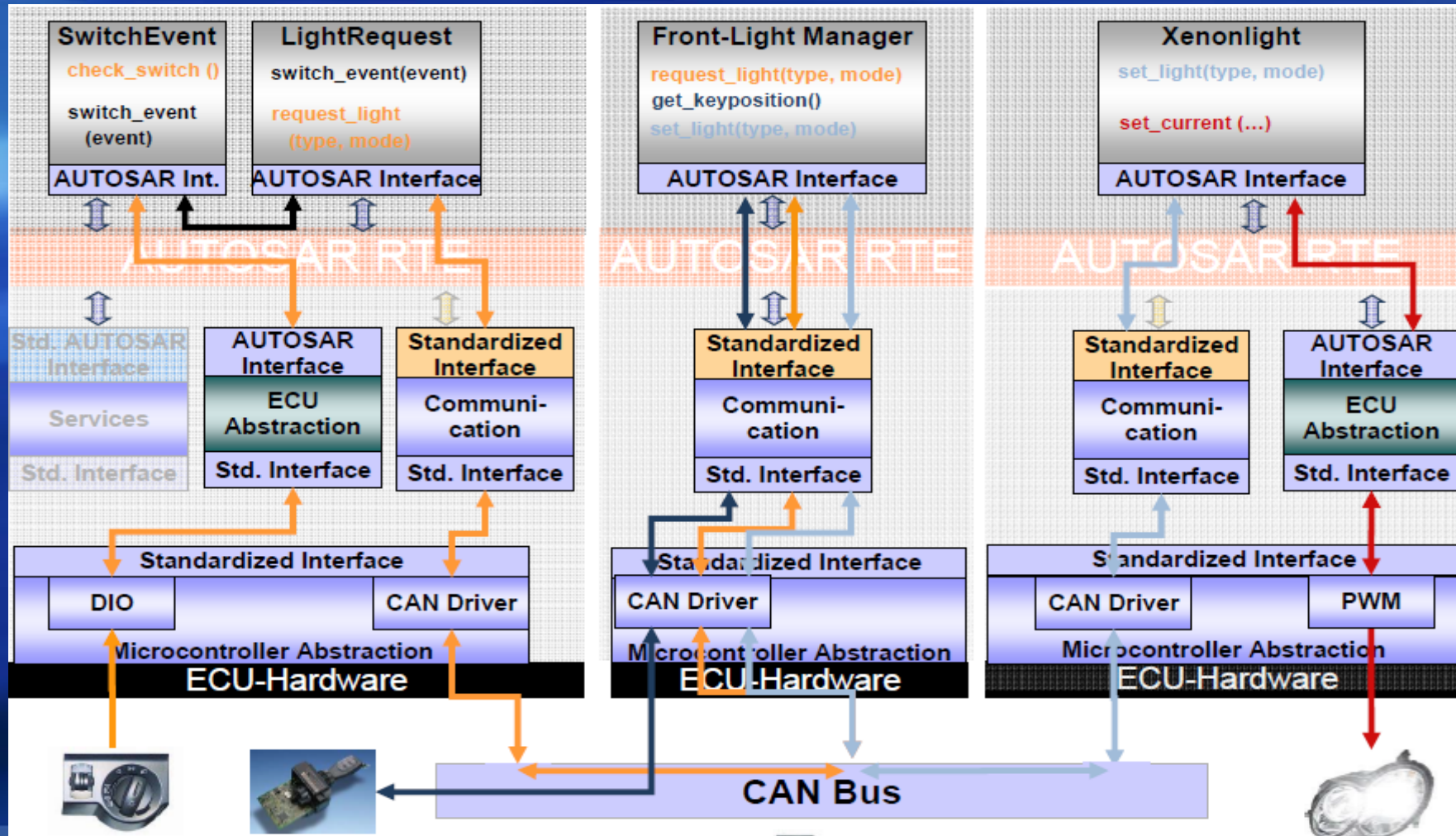


# SW Components and Runnables

- SW-Components
  - atomic block with respect to *mapping*
  - provided by one supplier
- Runnables
  - atomic block with respect to *execution*
  - attach to different OS tasks



# Example: Front-Light Management (Distributed)



# AUTOSAR OS

## Basic feature

- Configured and scaled statically
- Amenable to reasoning of real-time performance
- Provides a priority-based scheduling policy
- Provides protective functions at run-time (for memory, time, etc)
- Can run on low-end controllers and without external resources

## OS Abstraction Layer

- Define for co-existence of AUTOS OS and proprietary OS

## ECU State Manager

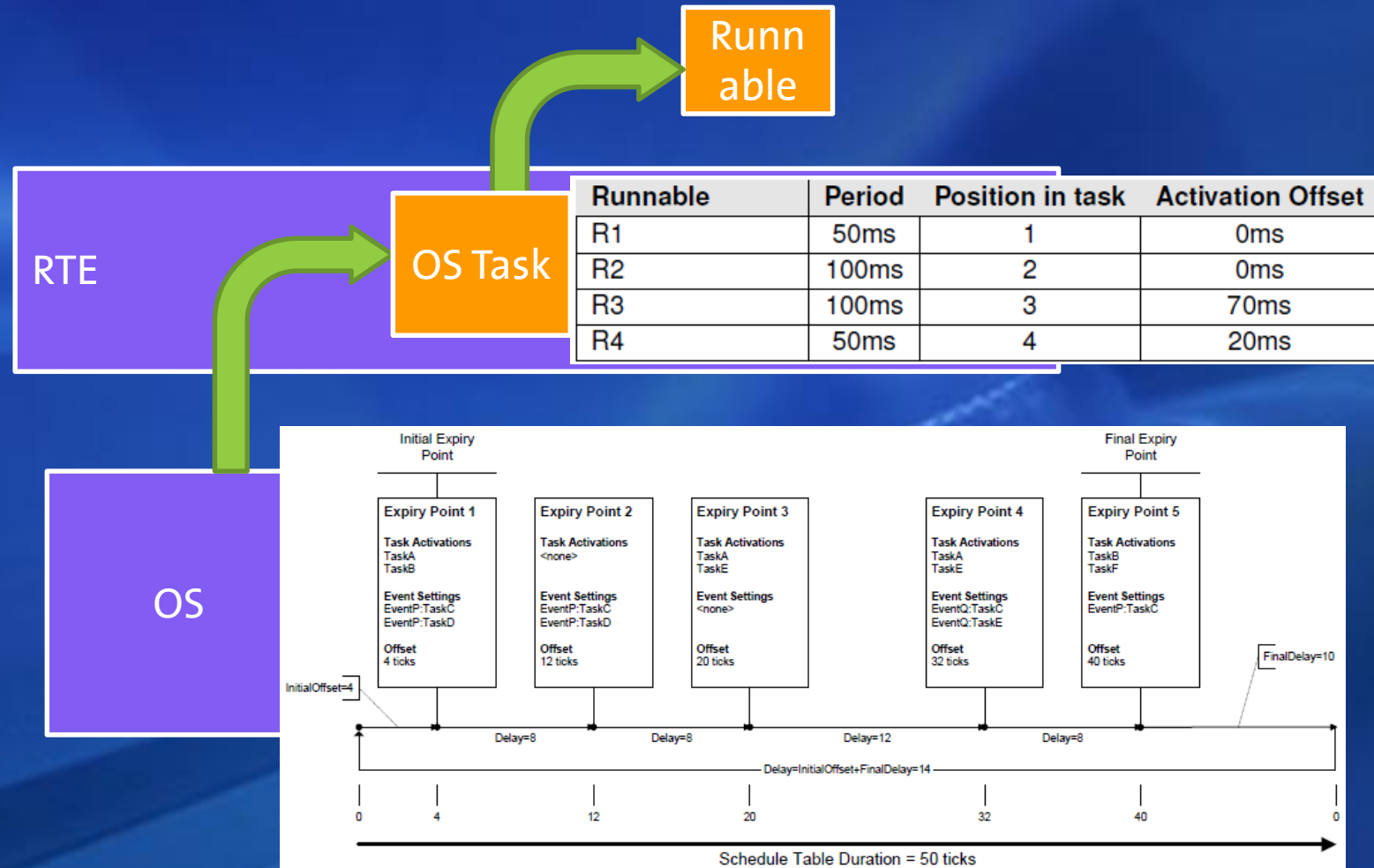
- starts/stops AUTOSAR OS

## Interaction with RTE

- Map runnables of the same SW component to task(s)
- Share protection boundary among runnables of the same task
- Tasks and ISRs for basic SW scheduled by OS

# AUTOSAR Scheduling

- Two level, table-driven
- Used for static scheduling: all tasks are synchronized with alarms
  - Alarms fixed once started
  - Each table linked to one tick counter
- Defines expiry points
- Allow multiple table activated concurrently – compositional schedule
- Two types of schedule tables: one-shot, repeated





# AUTOSAR OS Configuration

```
#define START_TIME_10MS      0
#define CYCLE_TIME_10MS     10
```

```
COUNTER SYSTEM_COUNTER {
    /* 1MS system counter */
    MAXALLOWEDVALUE = 65535;
    TICKSPERBASE = 1000000;
    MINCYCLE = 1;
};

ALARM <MODULEPREFIX>_TRANSMIT_ALARM {
    COUNTER = SYSTEM_COUNTER;
    ACTION = ACTIVATETASK {
        TASK = <ModulePrefix>_MainFunction_TransmitTask;
    };
};
```

```
#include "<ModulePrefix>.h"
#include "SchM.h"
#include "SchM_cfg.h"
#include "SchM_<ModulePrefix>.h"

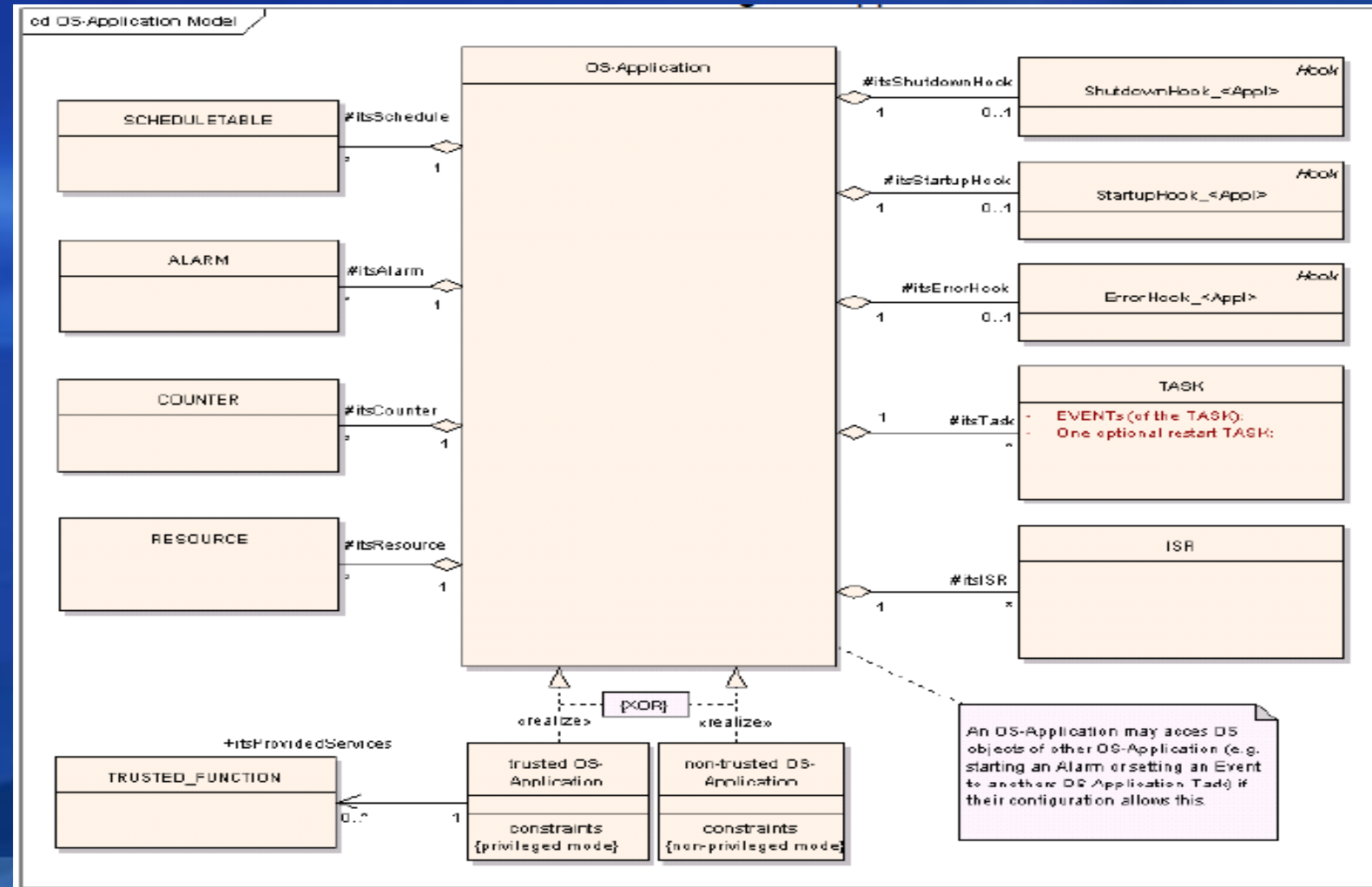
void
SchM_Init () {
    SetRelAlarm (<MODULEPREFIX>_TRANSMIT_ALARM,
                 START_TIME_10MS,
                 CYCLE_TIME_10MS);
}

TASK (<ModulePrefix>_MainFunction_TransmitTask) {
    <ModulePrefix>_MainFunction_Transmit ();

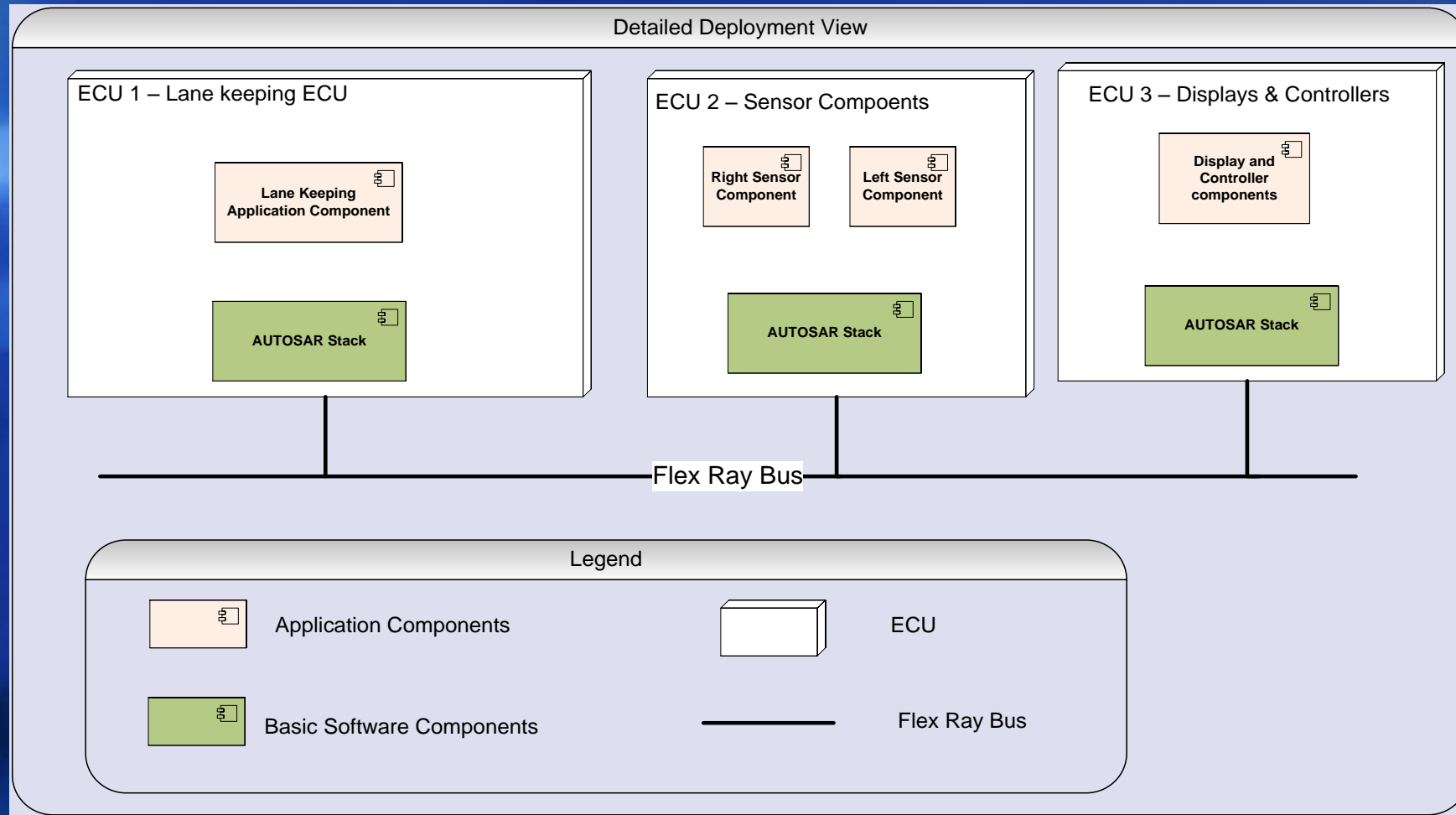
    TerminateTask ();
}
```

# OS Application

- A set of OS objects to form a functional unit
- Can be trusted or non-trusted
- Objects of the same OS application can access each other

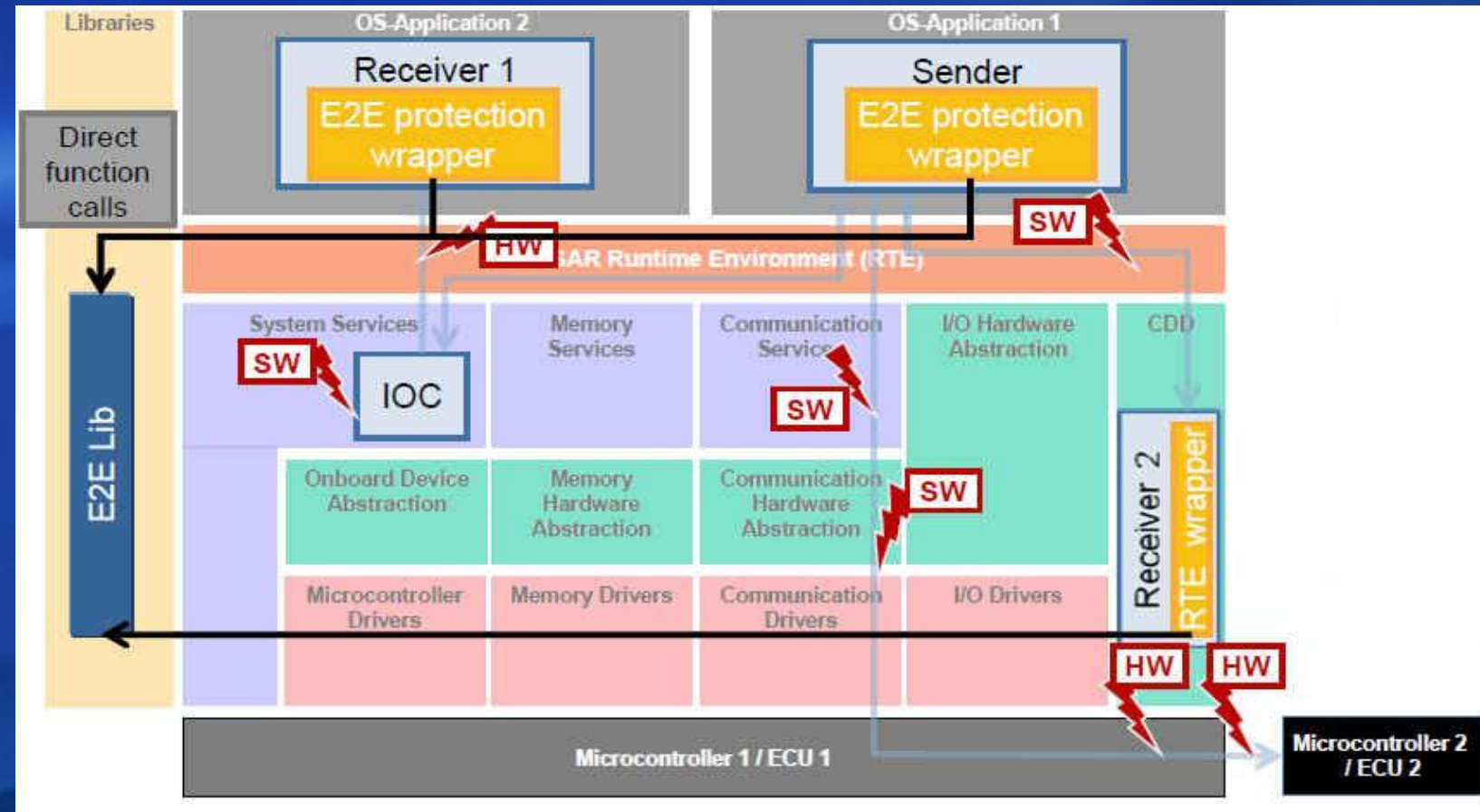


# Deployment View


















# AUTOSAR Fault-Tolerant: E2E Protection

- Mitigate faults in E2E communication
- Defined a set of E2E profiles – non-generated, deterministic code
- E2E library defines 3 Profiles

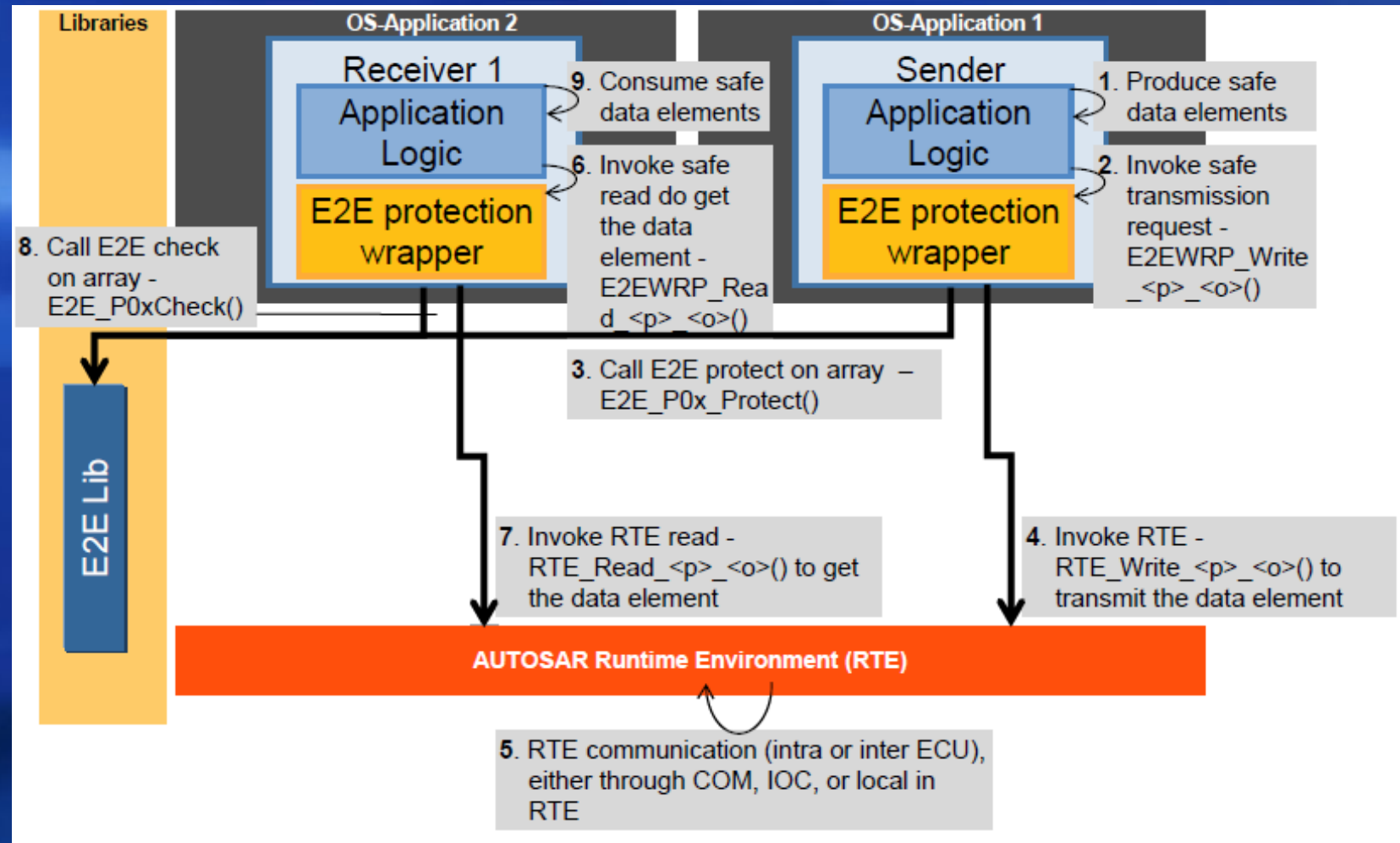




# E2E Communication Profiles

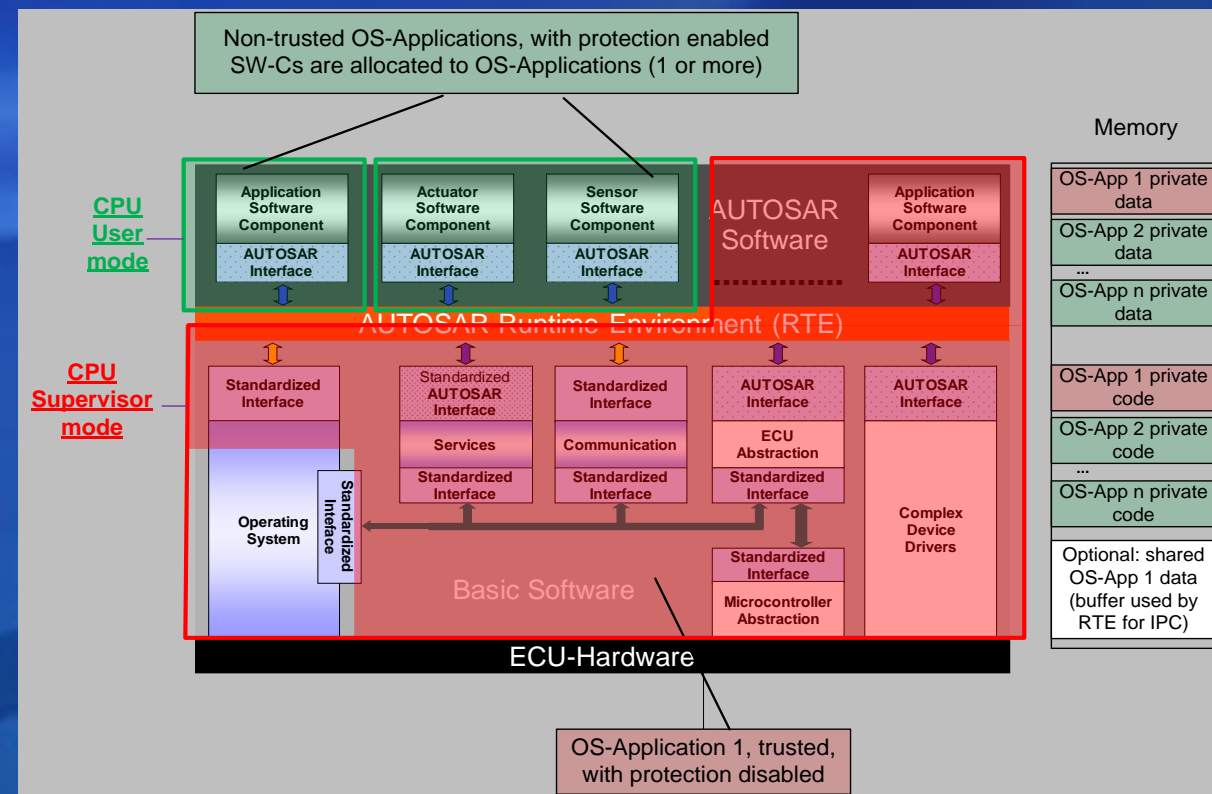
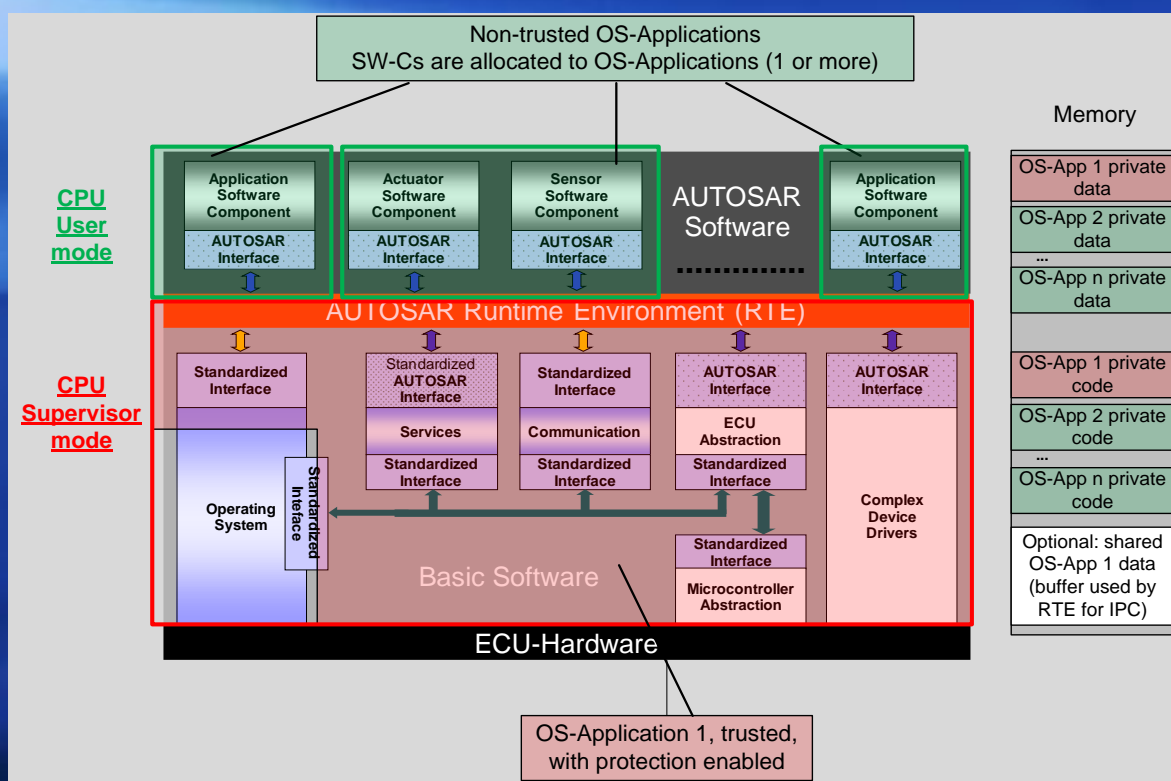
Errors	State of the art mechanisms																
	Hardware Redundancy	Time Redundancy	Information Redundancy	Checksum	Sequence Counter/Number	Message ID/ Data ID	CRC polynomials	State in CRC calculation	Parity bit	EDC/ECC	Timeout	Time stamp	Plausibility checks	Identification procedure	Cryptographic techniques	Retry mechanisms	Acknowledgement
Deletion	x	x										x				x	x
Repetition	x																x
Timing delay												x					
Incorrect Sequence	x																
Insertion of unintended data	x																
Data Corruption	x	x		x					x	x			x		x	x	x
Addressing Error	x													x			
Masquerading													x	x			
Inconsistency								x									
Constant "over-" Transmission												x					

# Safe E2E Communications



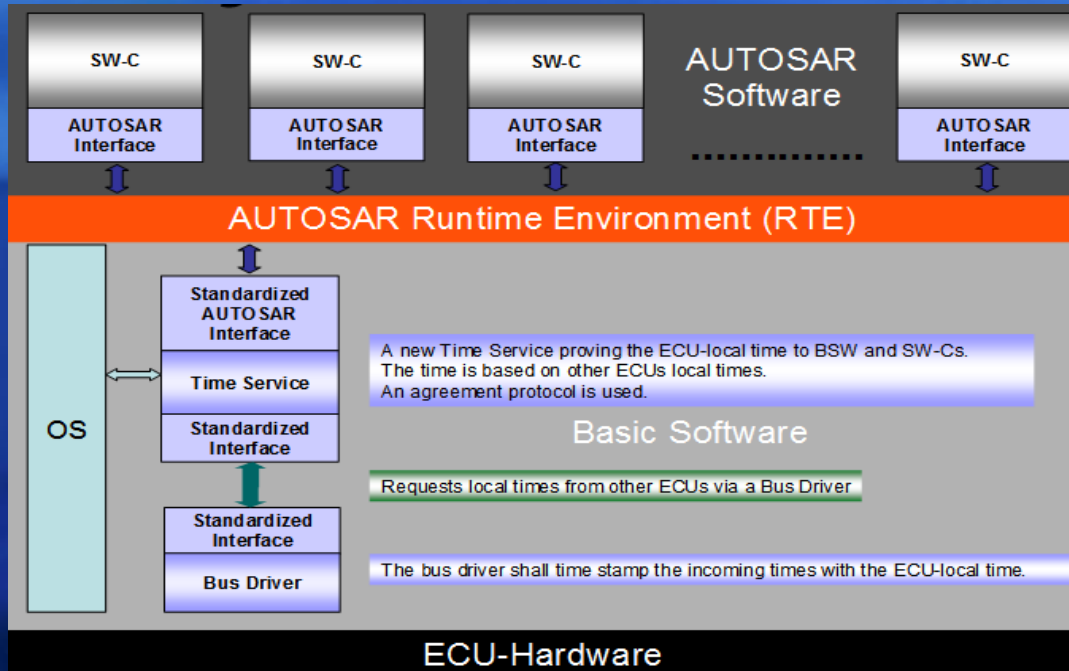
# Memory Protection

- Use OS-Applications: two variants

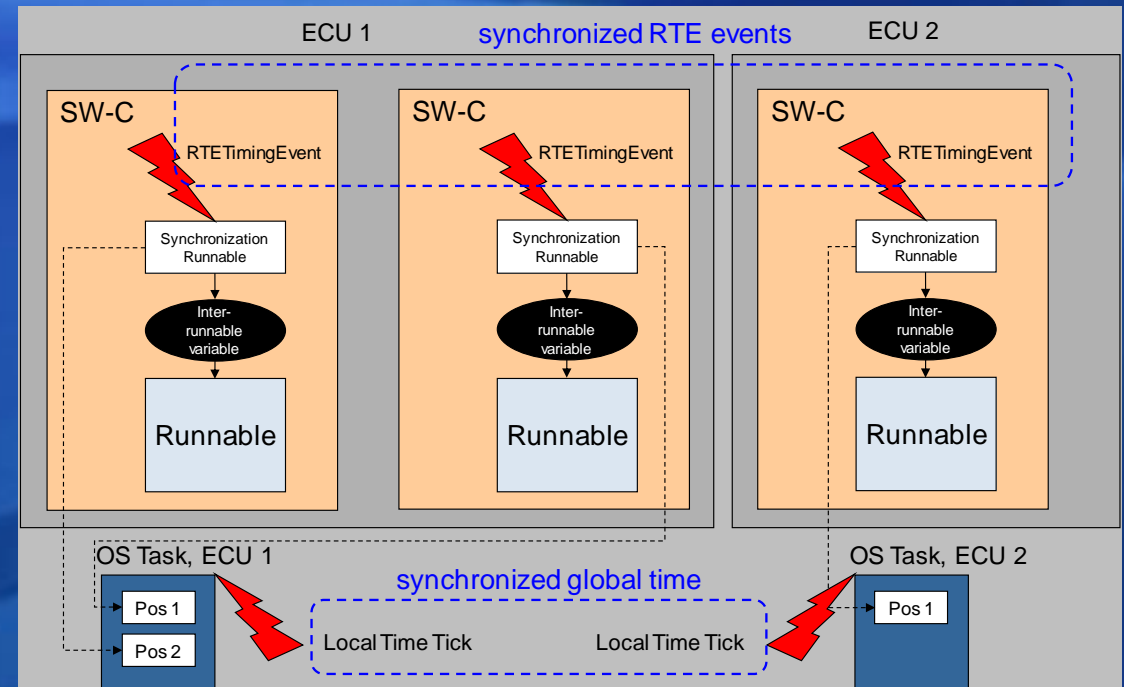


# Timing Protection

- Support specifying and verifying timing properties (period, latency, jitter, synchronization, execution time, deadline)
- Two implementation: BSW time service; RTE sync event



time sync protocol

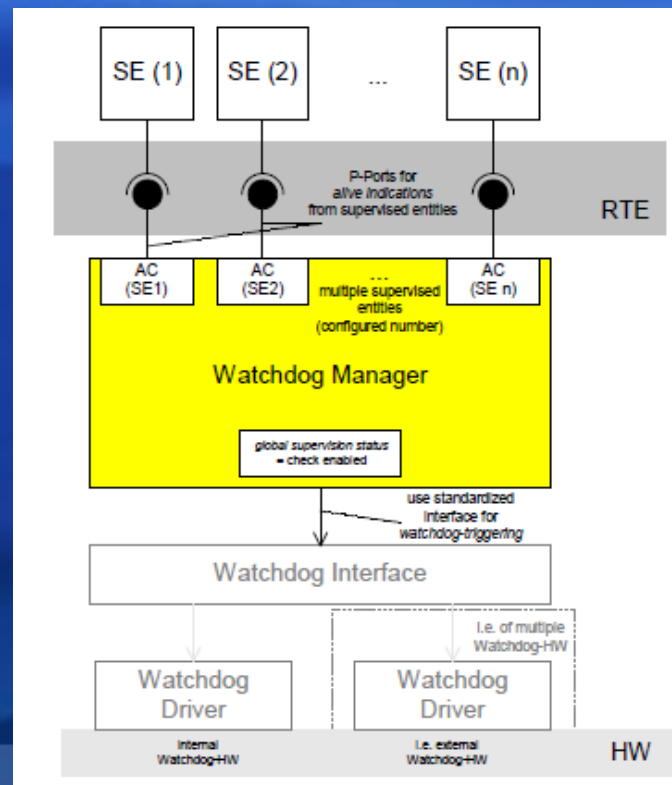
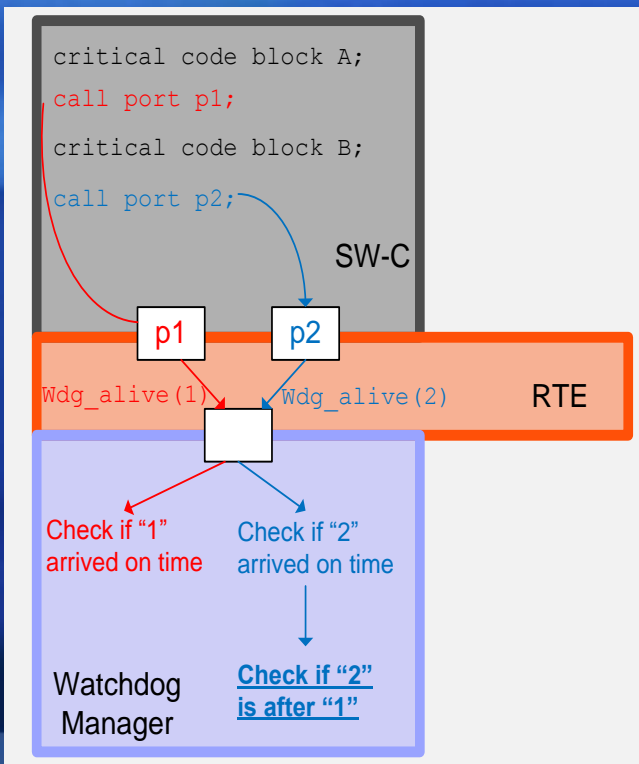


sync event

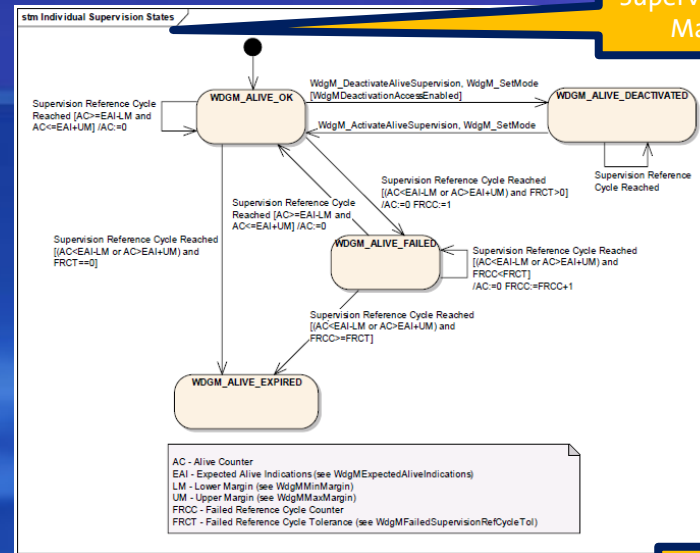


# Program Flow Monitoring

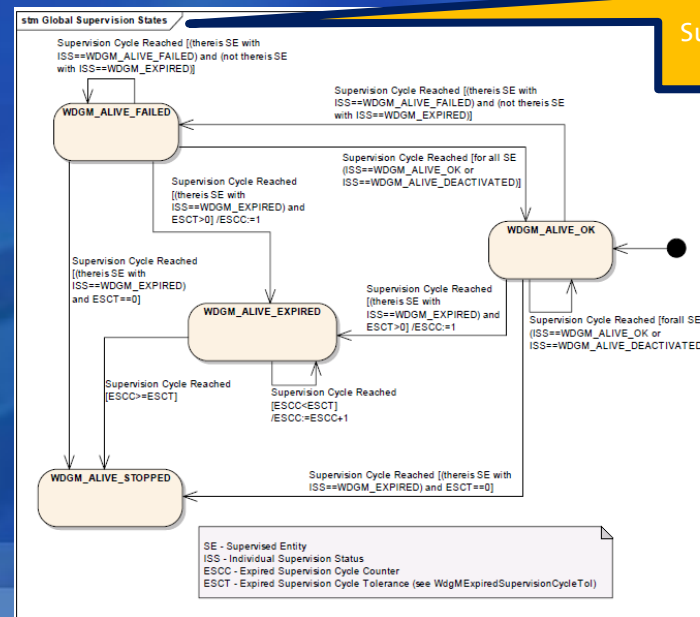
- Focuses on SWC program flow faults
- Two types: timely behavior; sequence of code blocks
- Use SWC ports and Watchdog Manager



Individual Supervision State Machine



Global Supervision State Machine



# AUTOSAR OS Scalability Class

Feature	Scalability Class 1	Scalability Class 2	Scalability Class 3	Scalability Class 4	Hardware requirements
OSEK OS (all conformance classes)	✓	✓	✓	✓	
Counter Interface	✓	✓	✓	✓	
SWFRT Interface	✓	✓	✓	✓	
Schedule Tables	✓	✓	✓	✓	
Stack Monitoring	✓	✓	✓	✓	
ProtectionHook		✓	✓	✓	
Timing Protection		✓		✓	Timer(s) with high priority interrupt
Global Time /Synchronization Support		✓		✓	Global time source
Memory Protection			✓	✓	MPU
OS-Applications			✓	✓	
Service Protection			✓	✓	
CallTrustedFunction			✓	✓	(Non-)privileged Modes

Feature	Scalability Class 1	Scalability Class 2	Scalability Class 3	Scalability Class 4
Minimum number of Schedule Tables supported	2	8	2	8
Minimum number of OS-Applications supported	0	0	2	2
Minimum number of software Counters supported	8	8	8	8

# AUTOSAR Multicore OS

## Assumptions

- Individual cores can be identified
- Cores share instruction set and data size
- Exceptions stay within a core
- Interrupts can be triggered on any core
- Equal access of memory (or segment)
- No addition of cores after OS started, no shutdown/restart of individual cores
- No dynamic task assignment

## Core organization: master - satellite

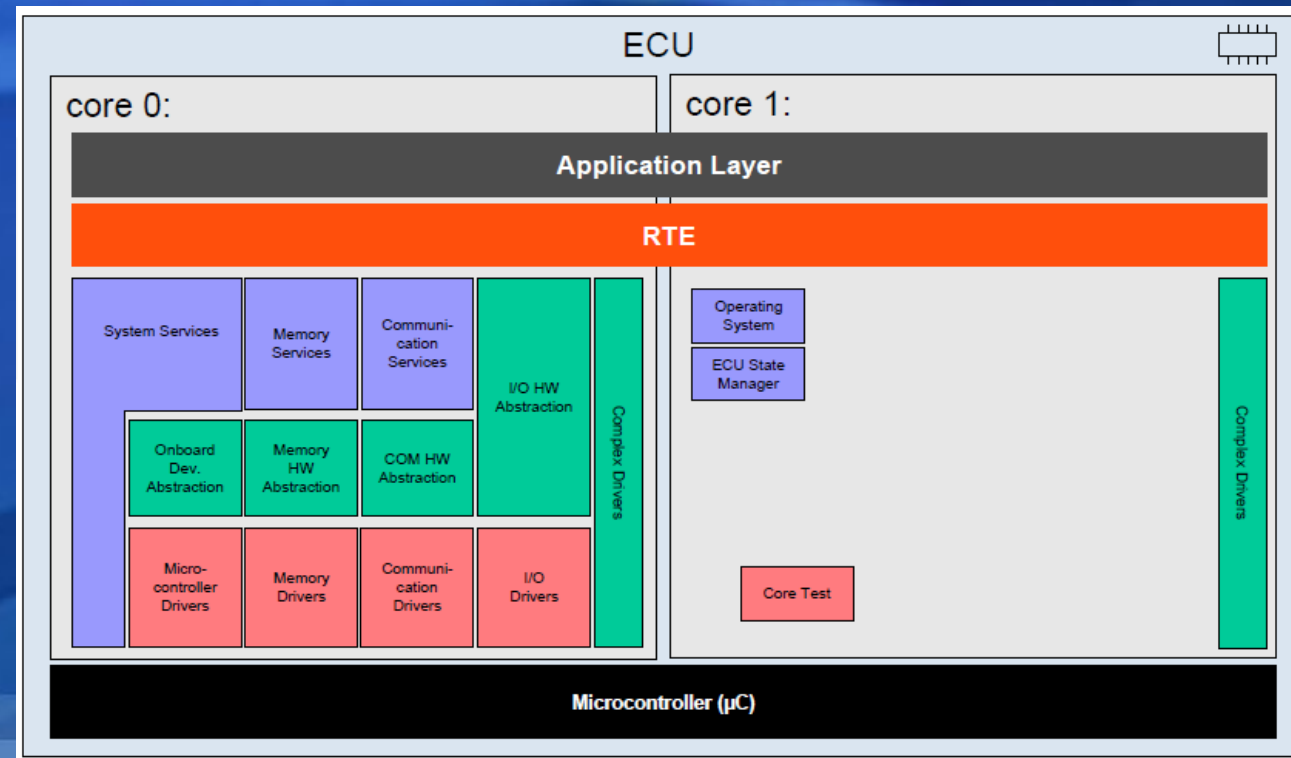
- Master: all BSW, wakeup/shutdown management
- Satellite: subset BSW only for CDD and SW-Cs

## ECU State Manager on each core

- Synchronize startup, shutdown, sleep operation

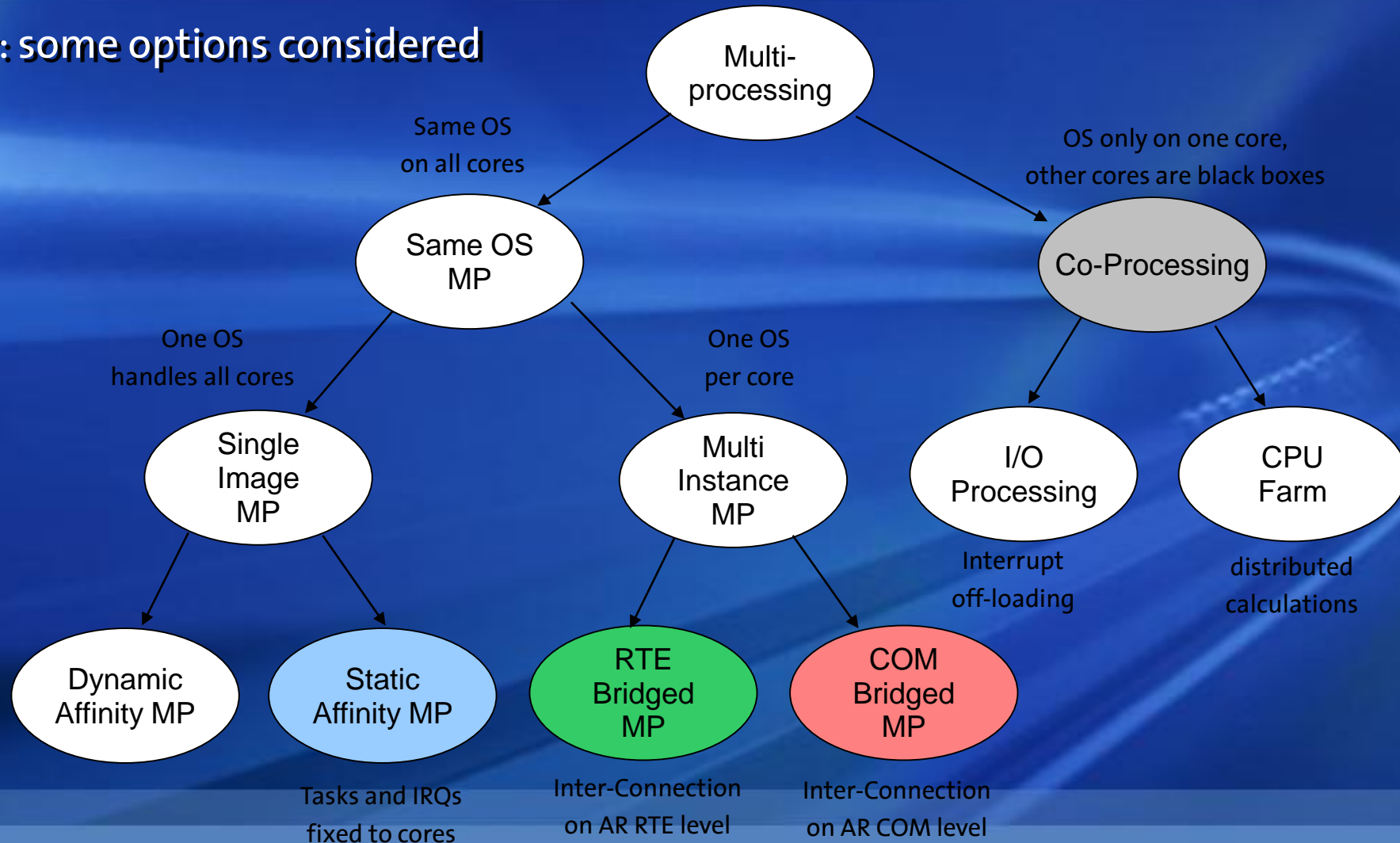
## A single configuration of OS for all cores

- Each core runs a part or whole copy of OS



# AUTOSAR Considered Multicore SW Architectures

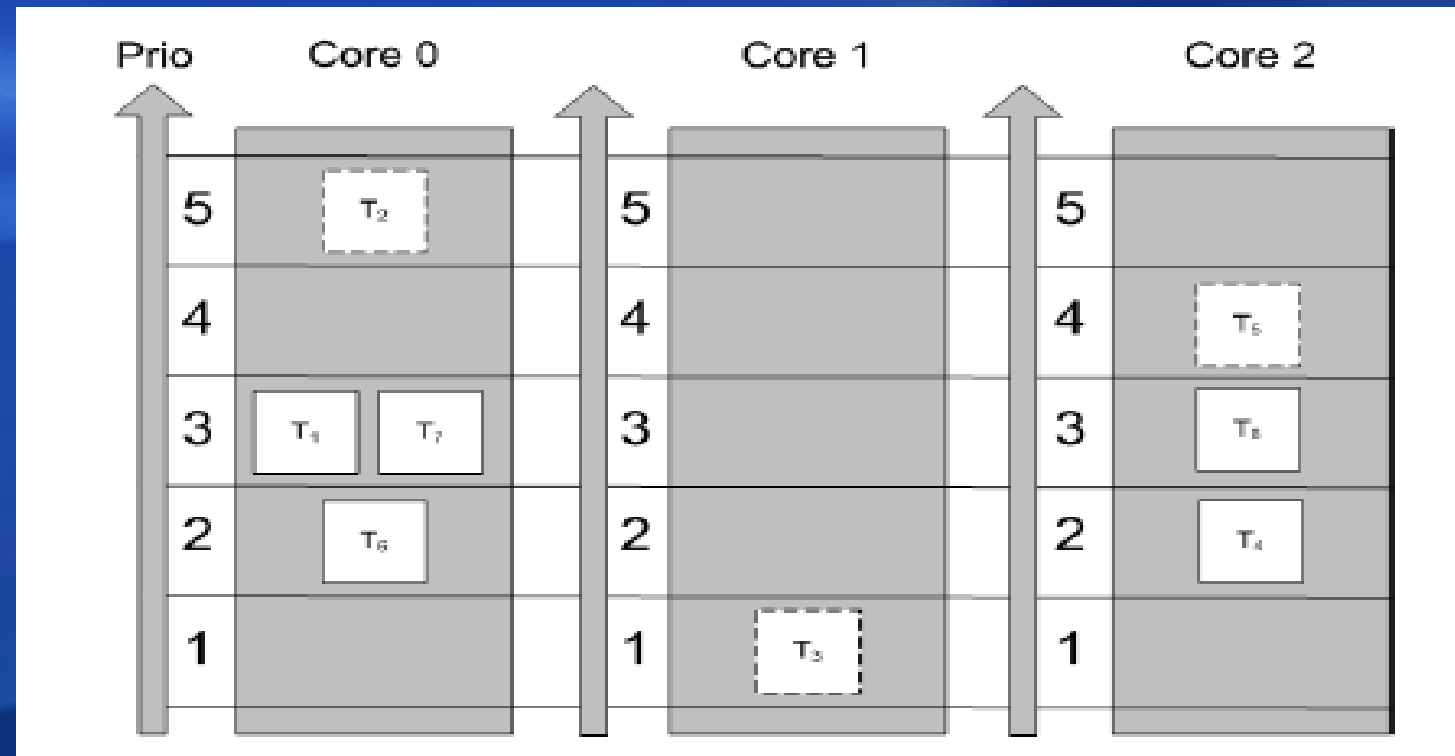
- History: some options considered





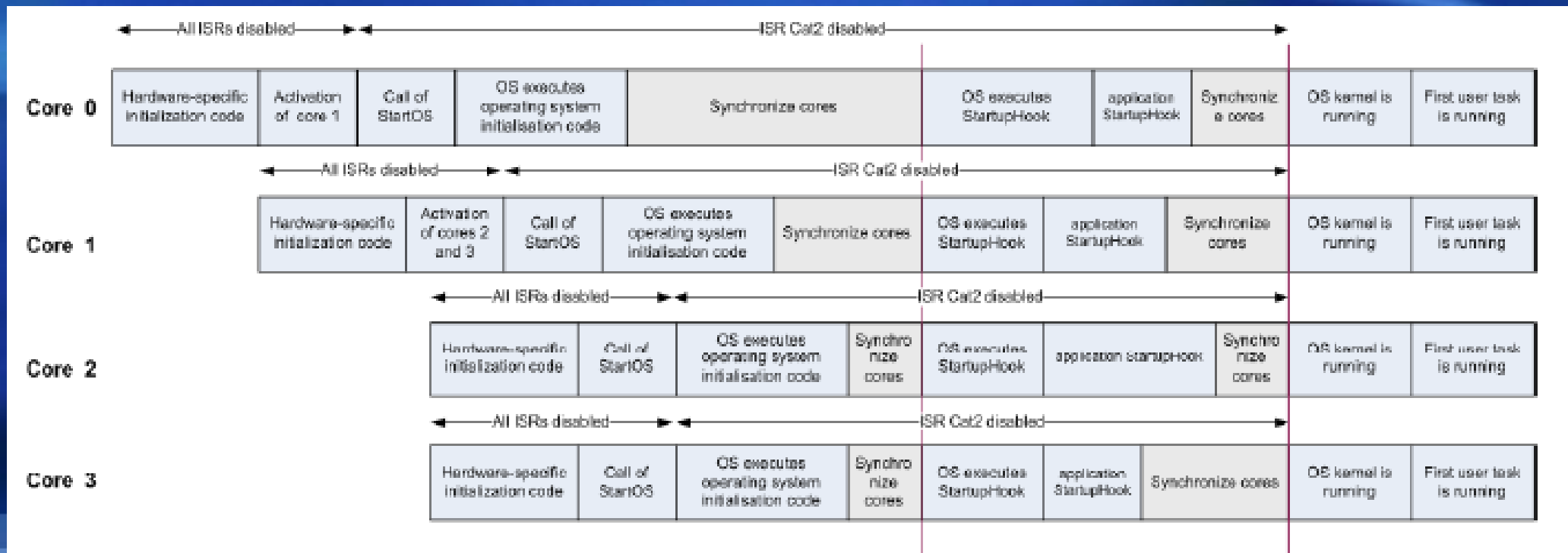
# AUTOSAR Multicore OS Scheduling

- Fixed priority-based schedule on each core
- Cores execute task independently
  - Each core has its own schedule table(s)
- Task priorities on different cores are limited to local



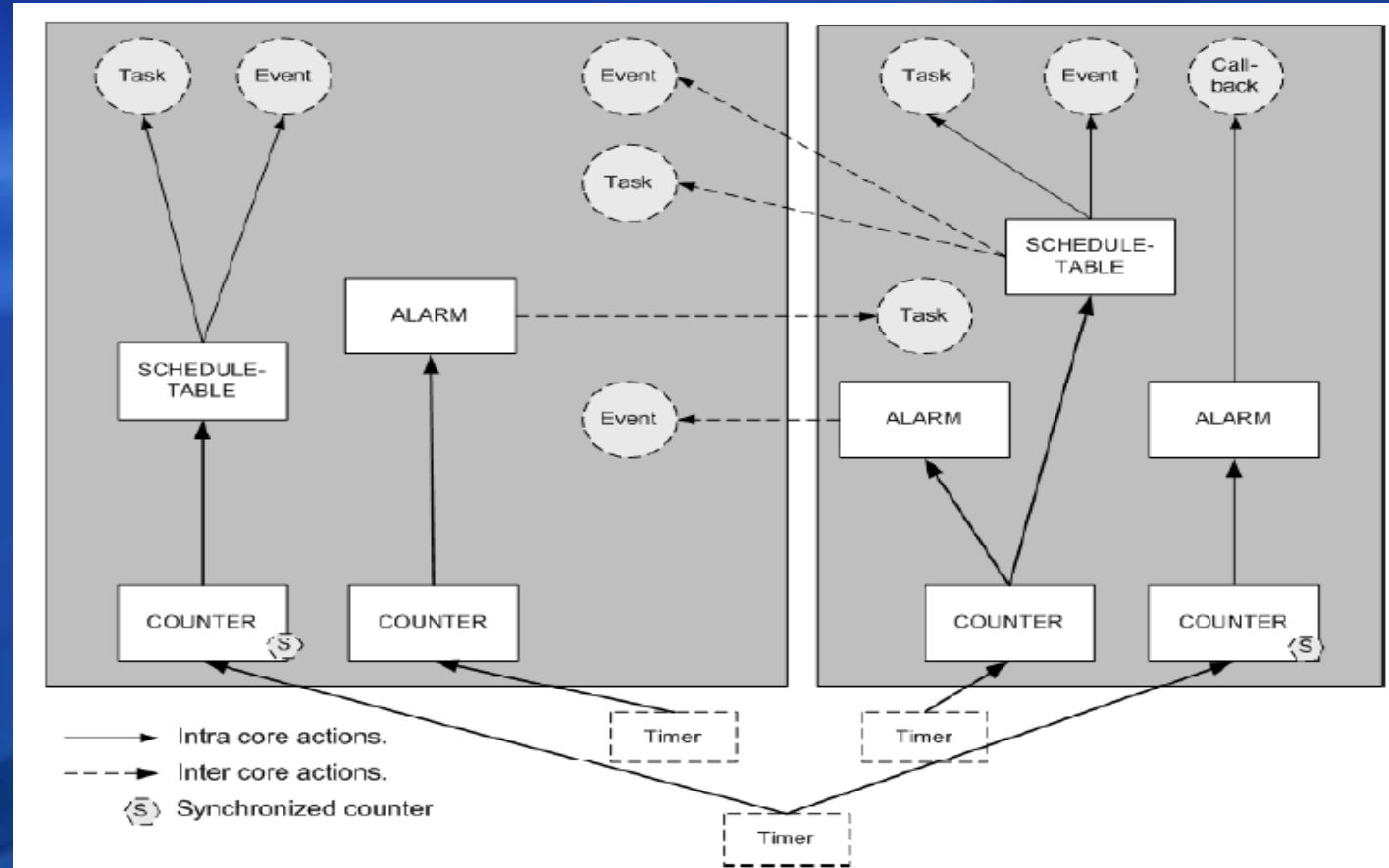
# Multicore OS Startup

- Each core must have at least one OS Application
- Master-satellite can be cascade
- All cores start before StartOS command



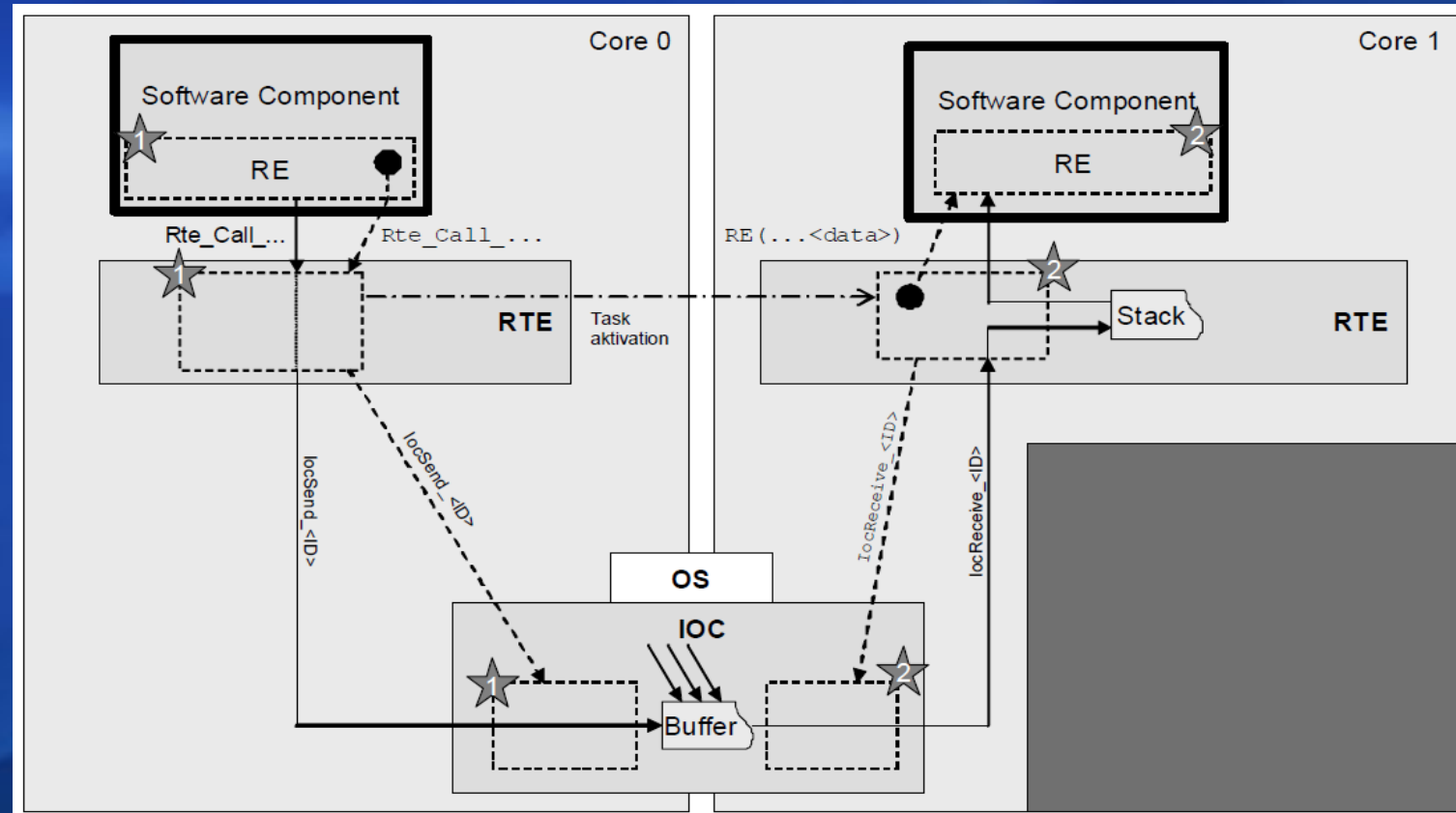
# Multicore Timing Control

- Each core must have at least one counter from timer
- Multiple counters exist for OS Applications
- Counters can be on remote cores
- Synchronization among counters may be required
- Implementation is not specified



# Multicore Inter OS-Application Communicator (IOC)

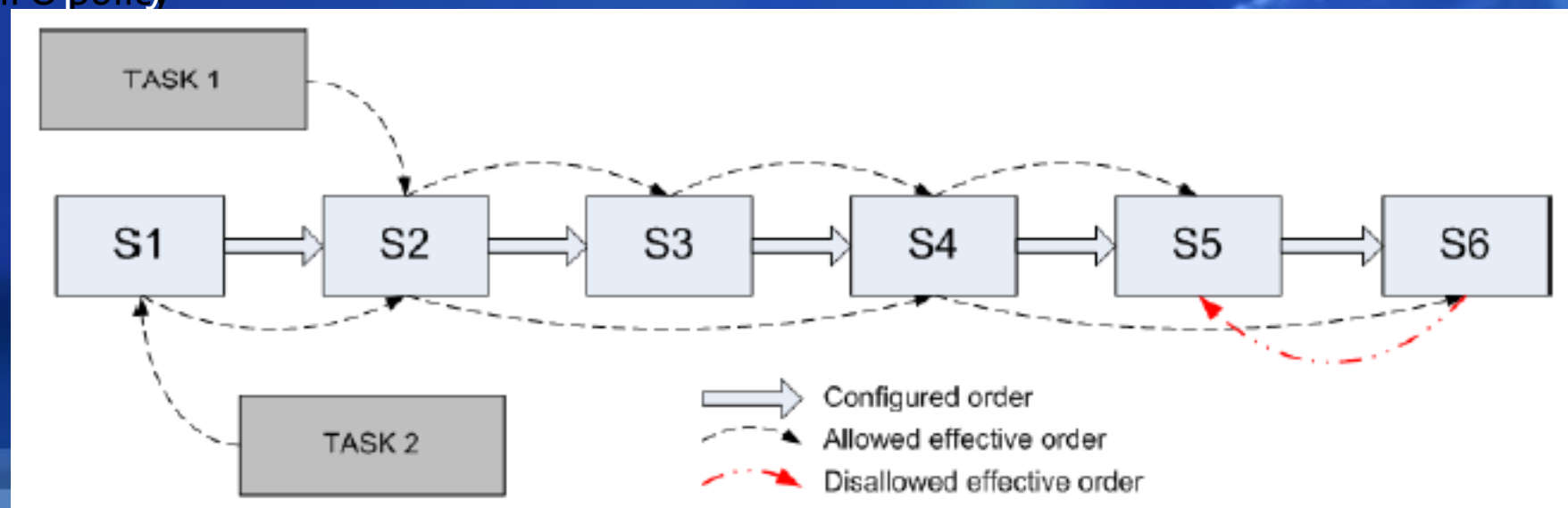
- For communications between OS-Applications across cores or memory protection boundary
  - In addition to Intra OS (via RTE) and Inter ECU (via COM)
  - Accessed directly (not support) or through RTE (logic one, physical splitted)
  - 1:1 or N:1 (1:N requires RTE generating N 1:1, changed in 4.2+)





# Mutual Exclusion Across Cores

- Spinlocks
  - A busy-waiting mechanism polling the variables until available
  - Used by tasks on different cores
  - Priority-based scheduling on each core not affected
- Deadlock avoidance
  - Resolved using predefined access order: all cores follow the same order
  - Controlled by LIFO policy



# New Technology: Advanced Driver Assistance System (ADAS)



Side Blind Zone Alert



Adaptive Cruise / Full-Speed Range ACC



Collision Mitigation Braking

Front Camera Functions



Lane Departure Warning  
Forward Collision Alert



Motorized Seat Belts



Rear Back-Up Camera

Human Interface Strategy



Aux Virtual Image Display (AVID)



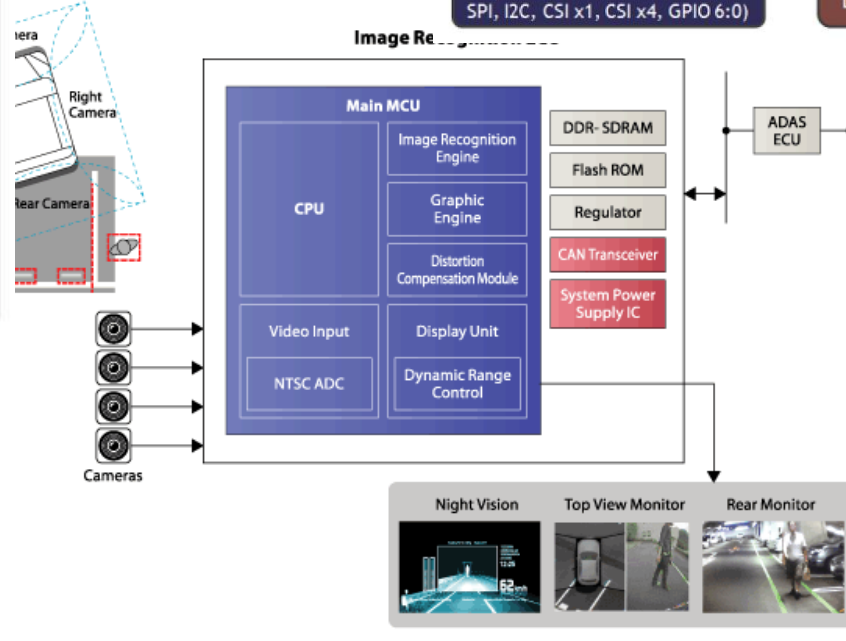
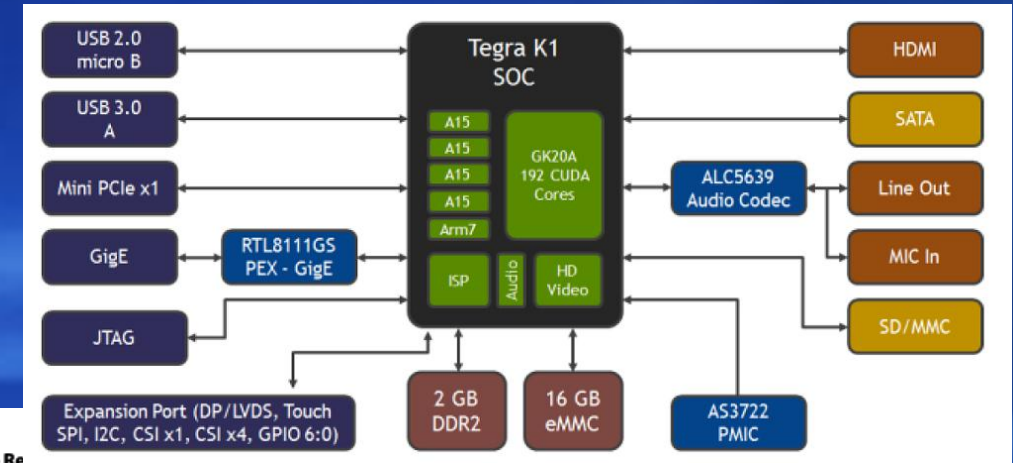
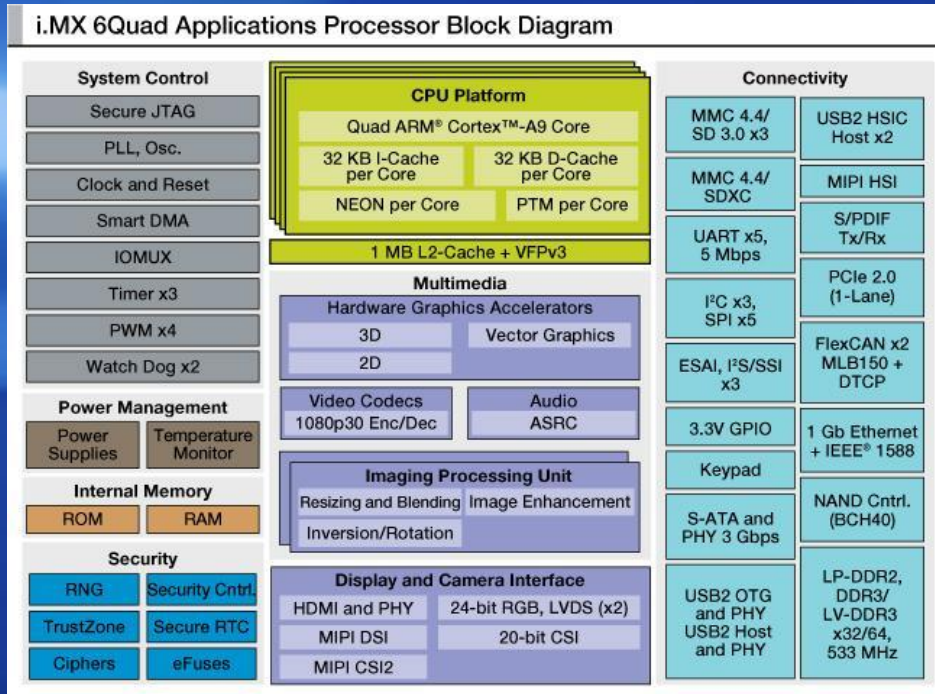
Haptic Seat

# ***Computing Challenges for ADAS***

- **Very high computation demands**
  - Image processing, complex math computation
  - Fusing and cross-examining information from different sources
- **Very large amount of data**
  - Fast and continuous sampling
  - Many streams of inputs
- **Under real-time and embedded constraints**
  - Source of information for critical controls (speed, steering, braking, etc)
  - Key part of system safety (ASIL-D), but with a weak computing platform support (ASIL-B)
- **Other business challenges**



# Hardware Options

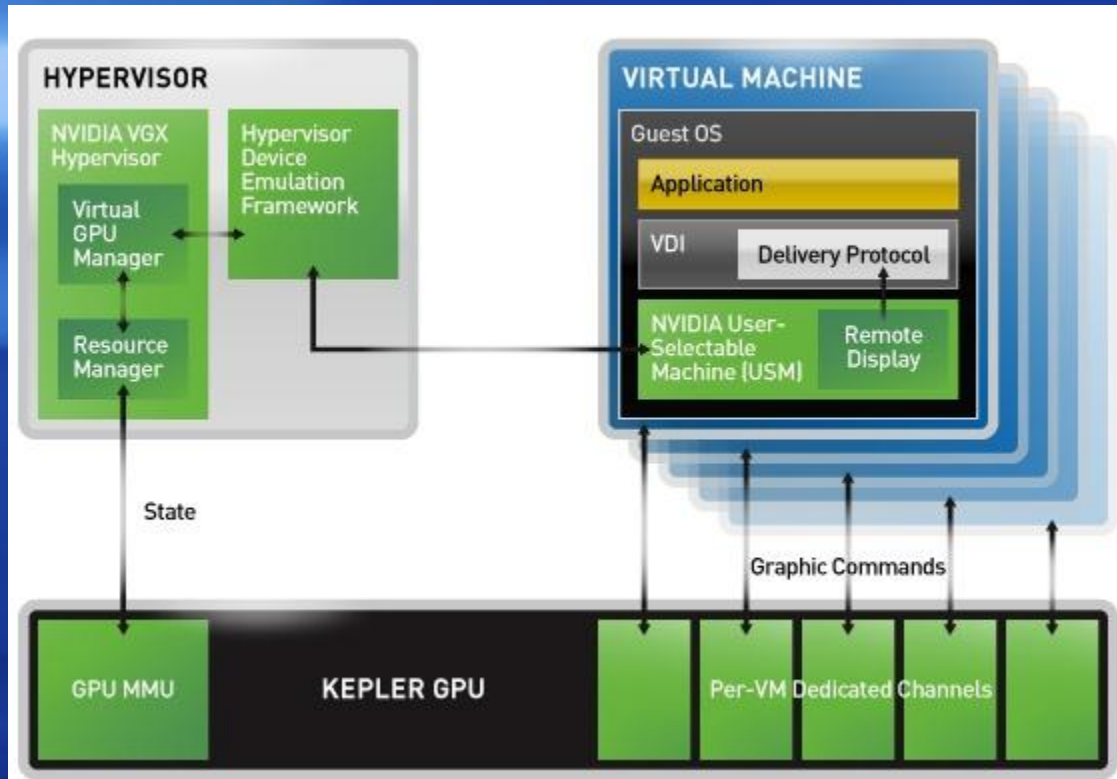




# ***Software Architecture Challenge: mixed-criticality***

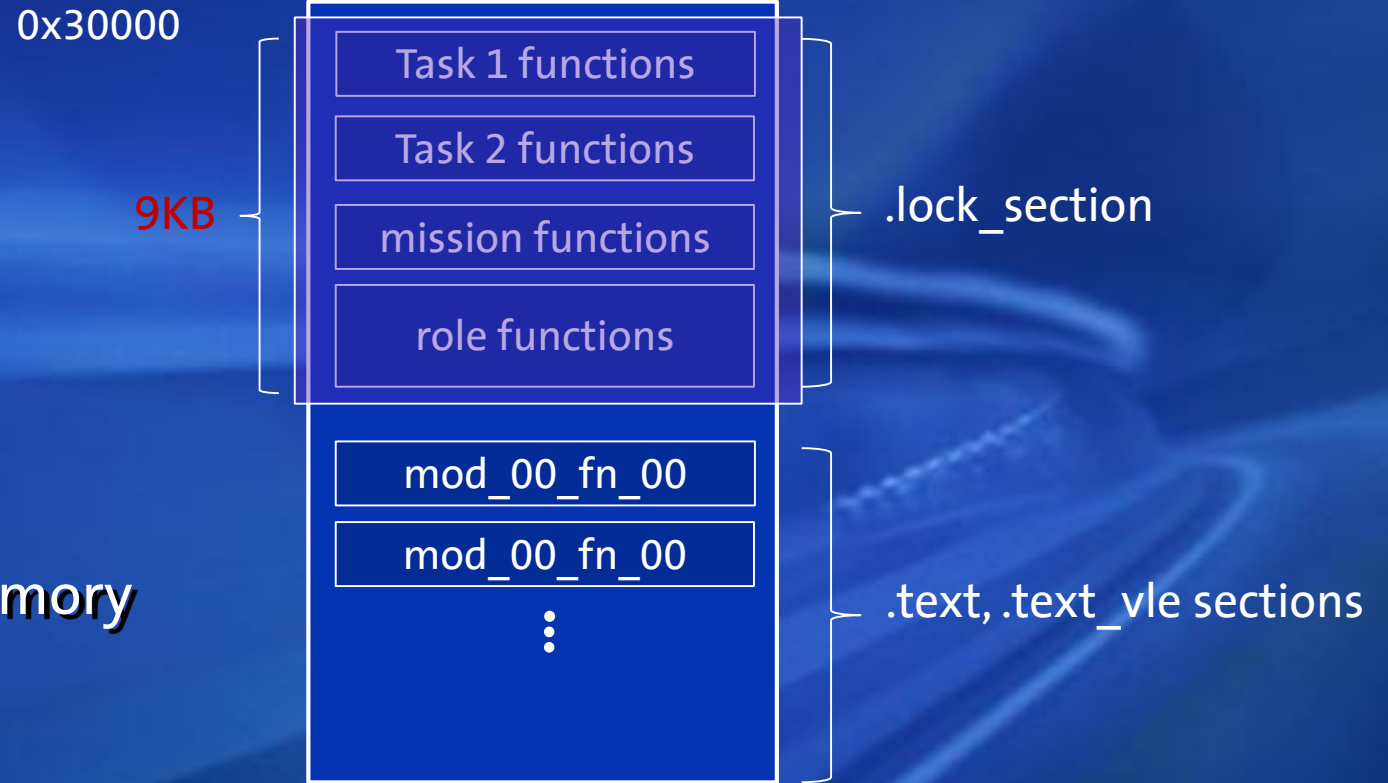
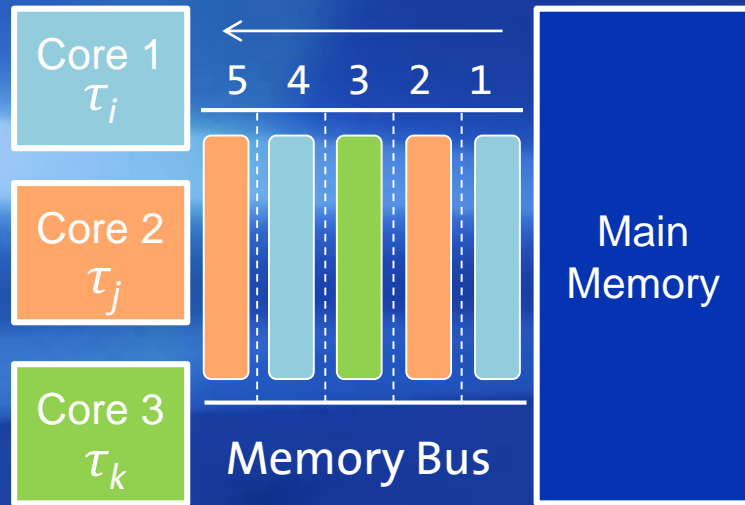
- **Classified for each feature**
  - Forward collision avoidance – ASIL D
  - Lane keeping – ASIL C
  - Lane departure warning – ASIL A
- **Different ASIL for functions in the same feature**
  - Object detection – ASIL B
  - Video streaming – ASIL A
- **ASIL may change in different modes during operation**
  - Back up: rear camera for monitoring and warning – ASIL C/D
  - Normal driving: rear camera for monitoring – ASIL B

# Software Architecture Challenge: virtualization



- Consider hypervisor
- Mixed different OSs to maintain independence of applications
- Different impl. strategies
  - Pass-through/native virtualization
  - Para-virtualization
- Some examples
  - Research: KVM, Xen, OKL4
  - Commercial: QNX, WindRiver, COQUOS, PikeOS, GH IntegrityOS, Coqos

# Software Architecture Challenge: memory management



- Performance depends more on memory
  - Same for multicore and GPU
  - Memory protection doesn't help
- Data locality should be explored
  - Cache, system memory, device memory
  - Partition of memory into continuous regions
- Cache- and memory-locking

# *Programming*

- Programming Model
  - Data parallelism
  - Task parallelism
- Languages
  - OpenCL
  - CUDA
  - OpenMP



## ***Other Issues***

- Task synchronization
- Data coherence and consistency
- Processor power consumption and temperature

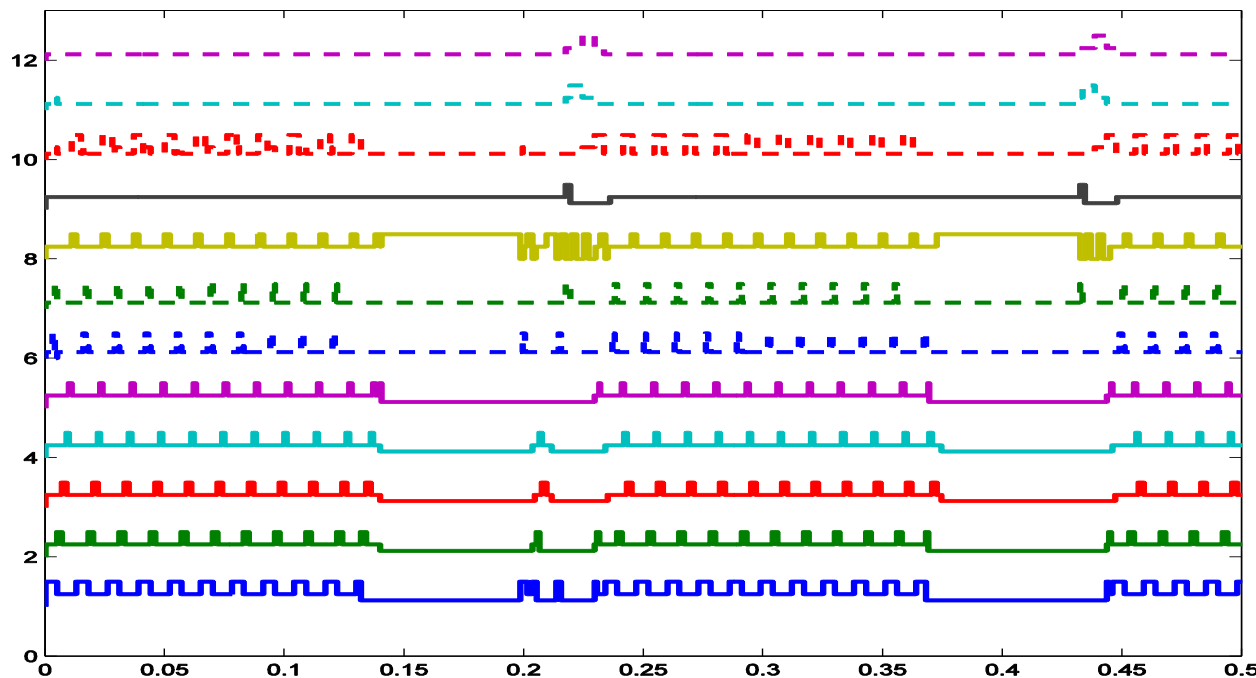
## **Remarks**

- Software development methods for large scale software for mass produced CPS are still not mature
- New applications post additional technical and business challenges
- Advanced hardware offers great potentials but also posts challenges
- Software architecture supporting predictable and analyzable vehicle controls is still evolving and requires further research

***END***

# Interference Among Controls

- Algorithms designed to run standalone
- Interferences of other controls through resource share causing more inconsistency (temporal and data)



- Each line represents activation of a distinct control (a schedule)
- Running standalone, they all should be with regular pulses
- Diagram shows running on a shared controller without any protection (code-gen, flash, run)
- Observed extensive blocking (no pulse) and busy-processing (piggyback pulses)

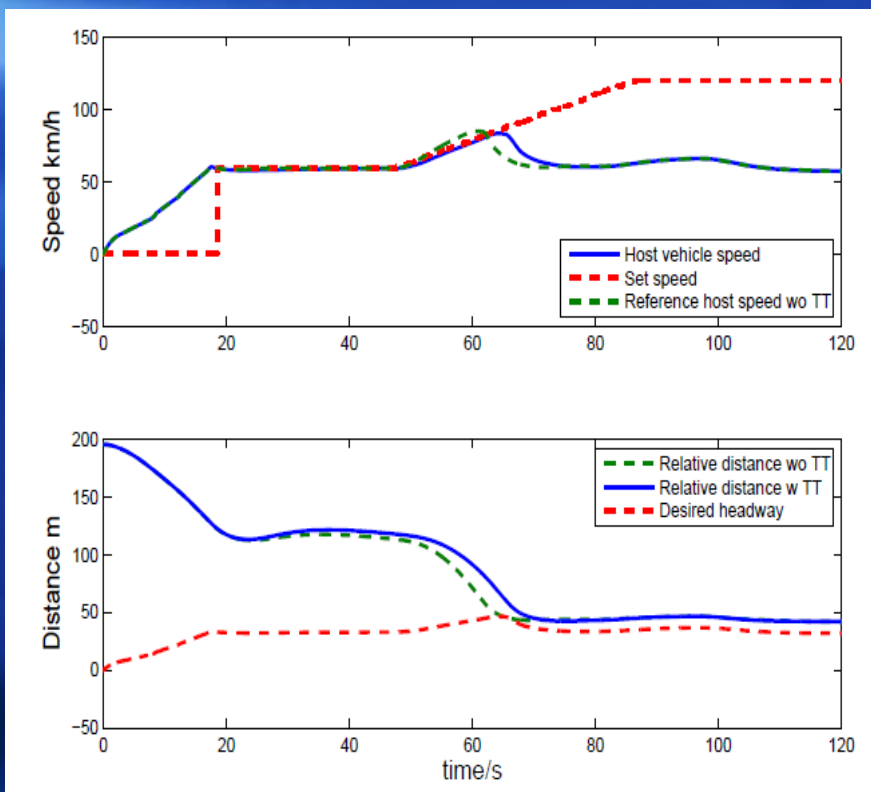


# ***Simulation Setup – Architecture Selection***

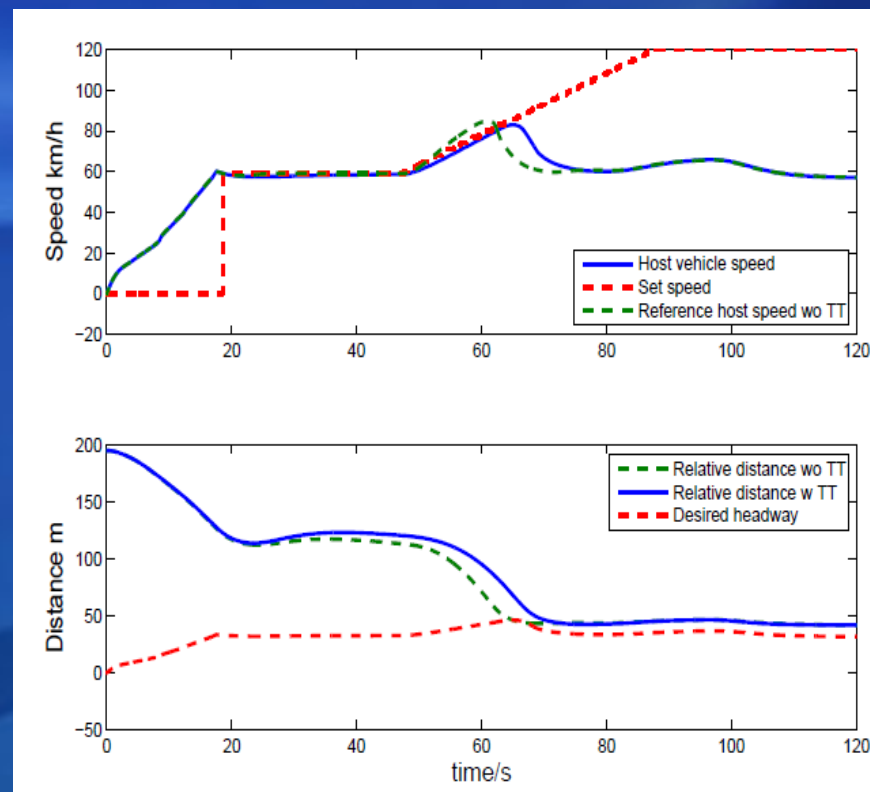
- **Test Scenario**
  - Host vehicle starts from 0 with 200 ft behind lead vehicle
  - Lead vehicle moves with a predefined speed profile
- **Task running synthetic workload**
  - Tasks are not randomly generated (like most academic cases)
  - To be filled with control algorithm after it is done
- **Processor utilization should be maintained less than 70%**
  - For single core, implies sampling period 80 ms and more

# Performance With Single or Dual-Core

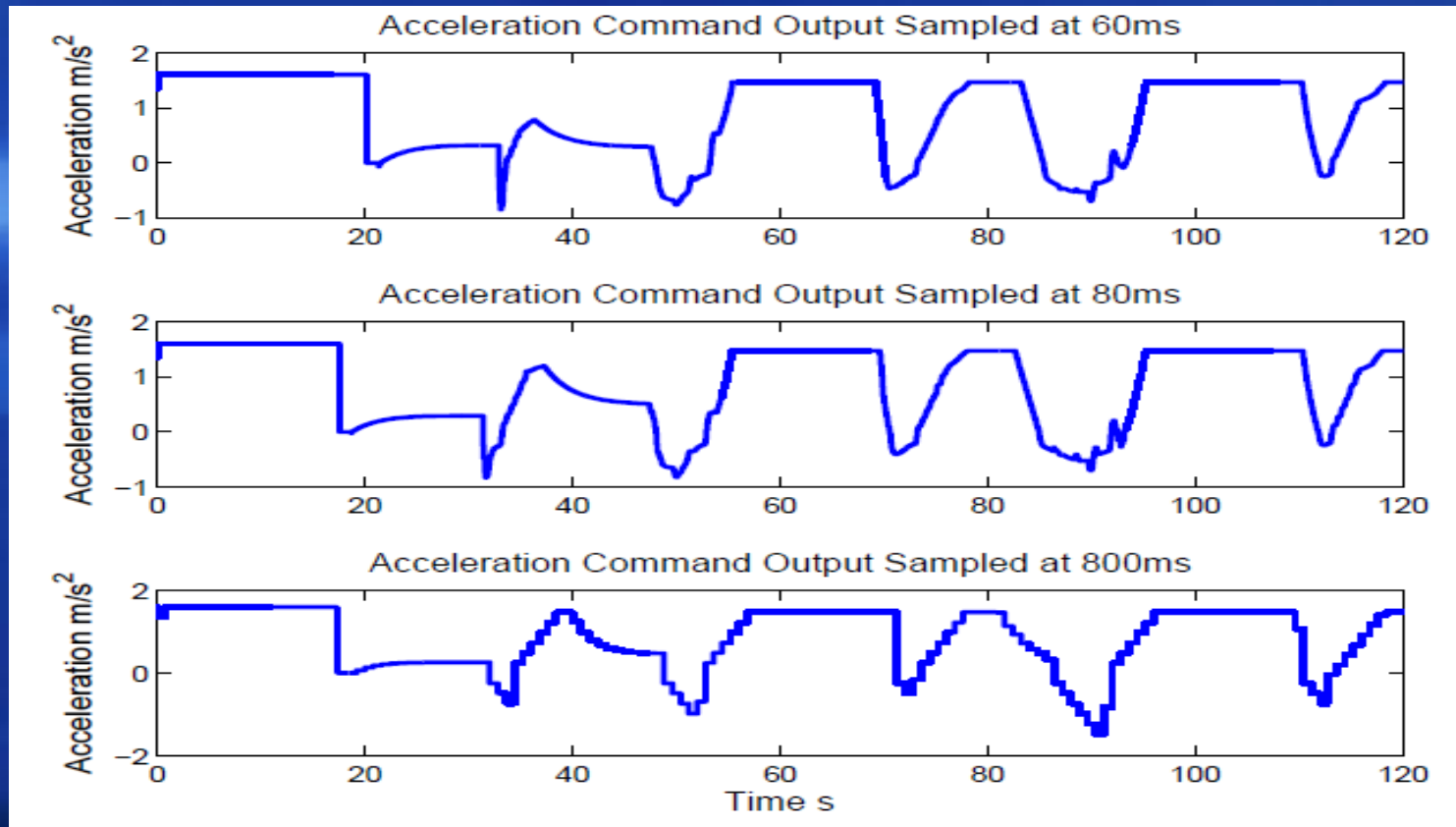
40 ms



80 ms



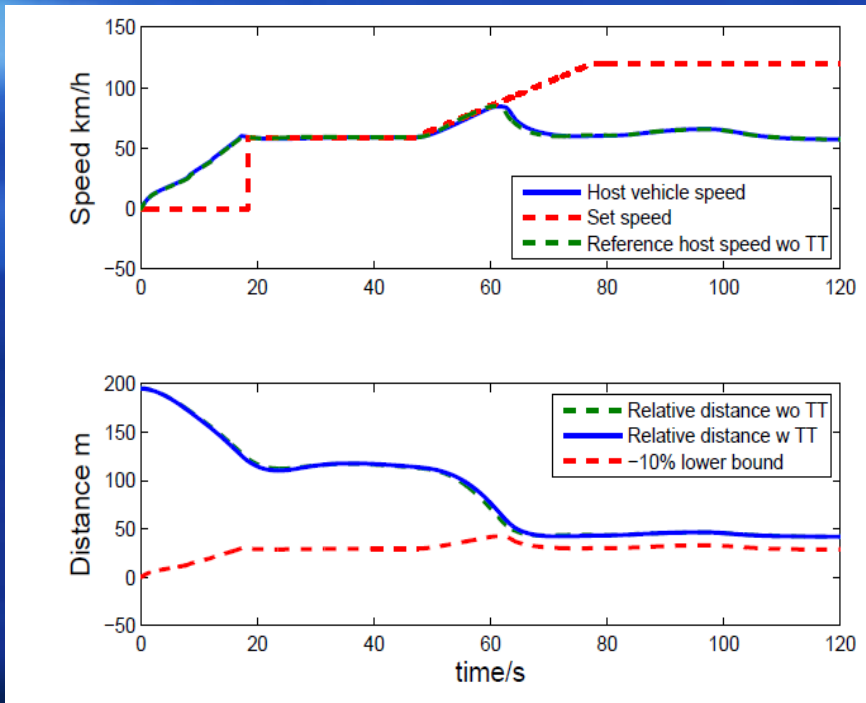
# Performance With Different Periods



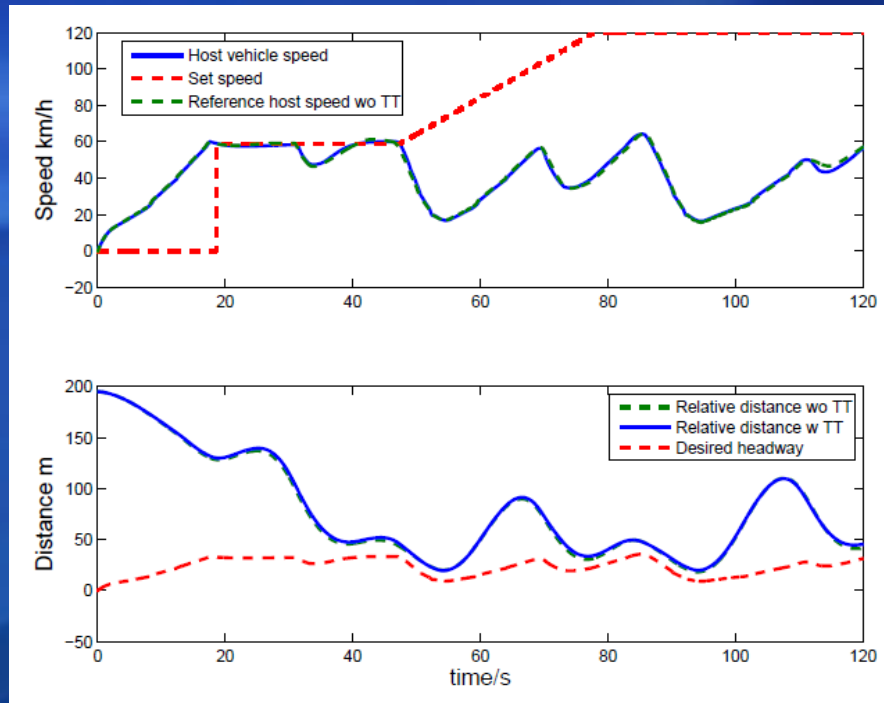
Running on a resource restricted platform, pick 800 ms  
If preserve control performance, pick 60 ms

# Performance With Different Synchronization

## 40 ms lock-free, wait-free



## 40 ms lock-based





# Performance Under Different Sync and Period

