

Contract-based Design for Modular Updates of Cyber-Physical Systems

Jürgen Niehaus,



Why do we need modular updates?

Generic (Domain Agnostic) Developement Process for Modular Updates







Contract based design

- > In general
- > In Step-Up!CPS

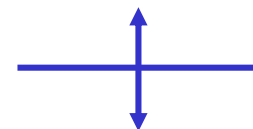
Summary

! Driving Tasks

Traditional

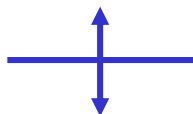
	Perception	Environment, incl. all (relevant) objects
	Interpretation	Recognize and understand the situation
	Prediction	Predict how the situation will develop
	Plan	Plan your own trajectory
	Actions	Increase/decrease speed, turn,...
	Execution	...of the action(s)

Driver



Car

Driver












Car

Autonomous



SAE Level: from fully manual to fully autonomous

← driver —————→ automated vehicle →

0	1	2	3	4	5
 <p>The driver constantly performs all aspects of the dynamic driving task. No systems intervene – only those that warn the driver.</p>	 <p>The system can take over either steering or acceleration / deceleration. The driver must continuously carry out the other.</p>	 <p>The system takes over both steering and acceleration / deceleration in a defined use case.</p>	 <p>The system takes over both steering</p>	 <p>The driver can hand over the entire</p>	 <p>The system can take over the entire</p>
 <p>The driver must constantly monitor the drive.</p>	 <p>The driver must constantly monitor the drive. He must be ready to resume full control immediately.</p>	 <p>The driver must constantly monitor the drive. He must be ready to resume control immediately.</p>	<div>Hard to design, hard to implement, hard to test and certify (especially when using AI)</div>		

TERMINOLOGY

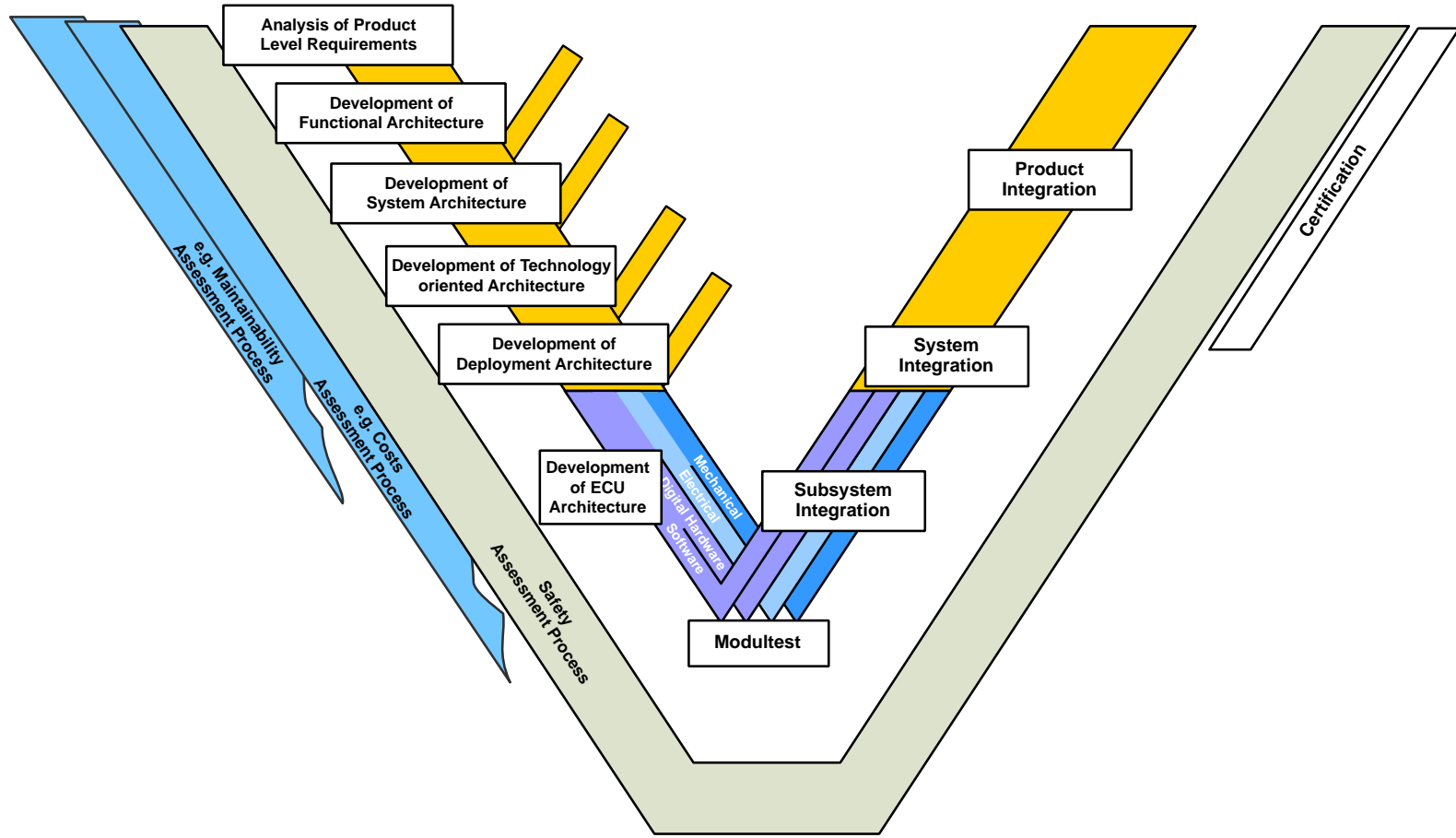
SAE (J3016)	No Automation	Driver Assistance	Partial Automation	Conditional Automation	High Automation	Full Automation
VDA*	Driver only	Assisted	Partly automated	Highly automated	Fully automated	Driverless
BASt	Driver only	Assisted	Partially automated	Highly automated	Fully automated	—
NHTSA**	0	1	2	3	3/4	

* used on this platform

** only roughly corresponding with the other taxonomies

Capabilities needed

Structured Design processes (like V-Model), Model-based development, Virtual Integration,...

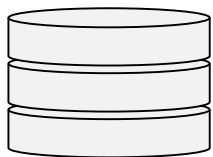


Capabilities needed

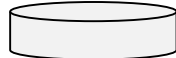
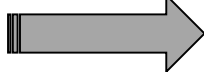
Scenario Based Testing, Virtual Engineering (esp. Virtual Testing), Digital Twins,...

Analysis of Product
Level Requirements

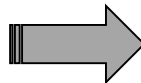
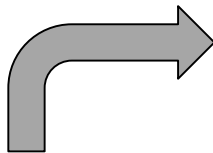
Scenario
database



Relevant
scenarios



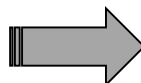
Scenarios for physical
tests



Testfield, real world, real car



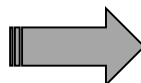
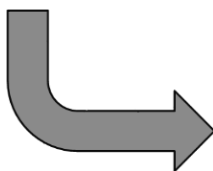
Scenarios for Homologation



Homologation



Scenarios for virtual
tests

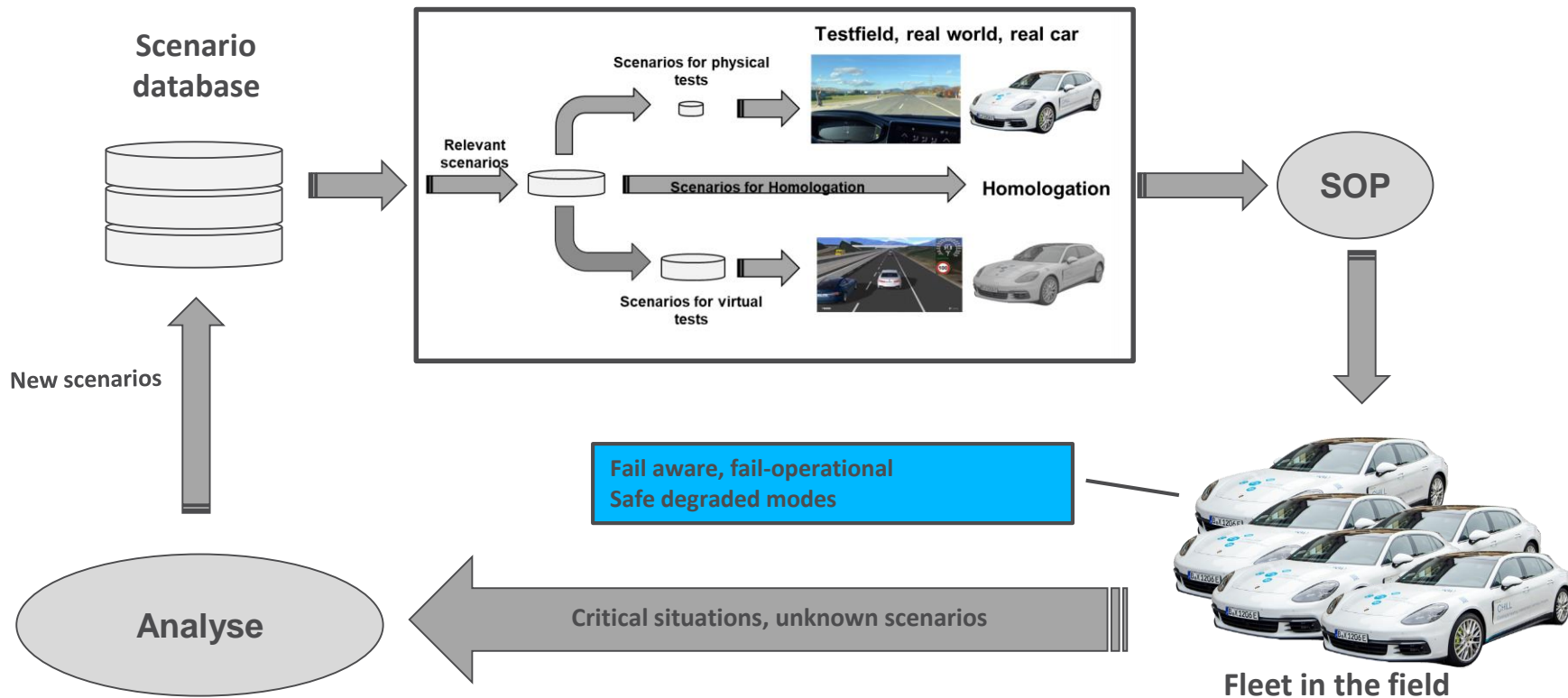


Simulation, virtual car, digital twin



Capabilities needed

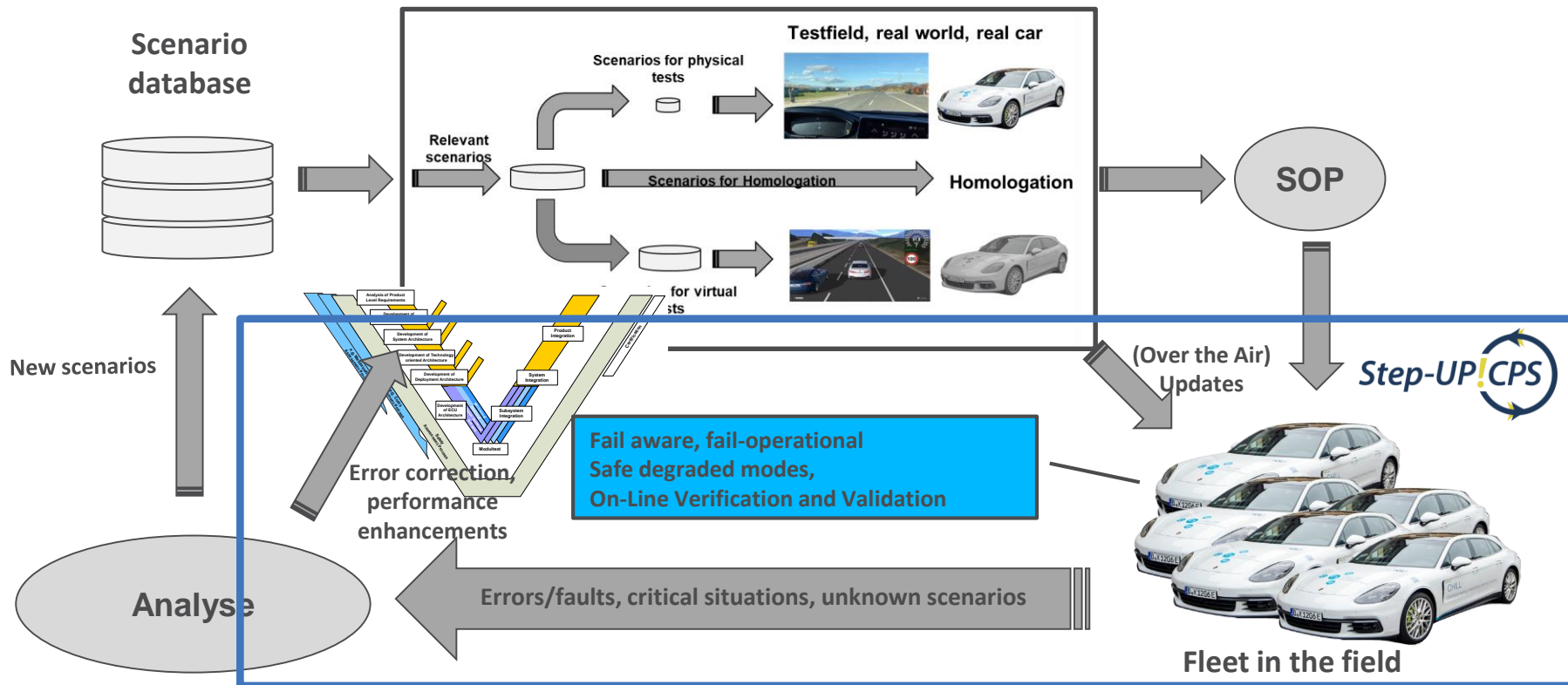
Data collection from the field to (a) extend scenario database and ...



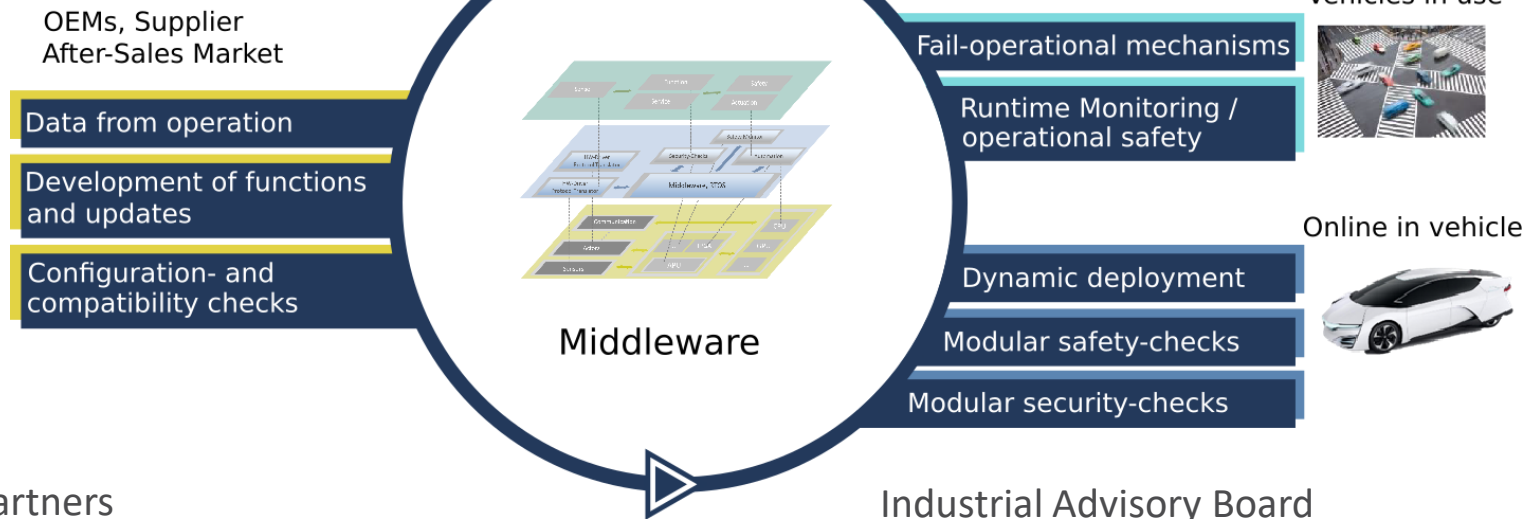


Capabilities needed

... and (b) continuous development and (over the air, OTA) updates



Step-Up!CPS



Project Partners



Bundesministerium
für Bildung
und Forschung



DENSO

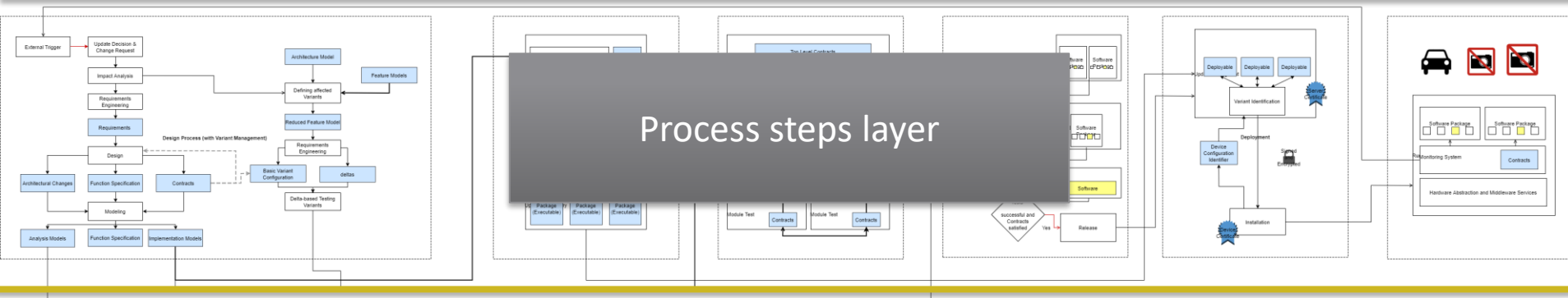


Supported by the Federal Ministry of Education and Research (BMBF)

!



Roles & Responsibilities layer



Data storage layer

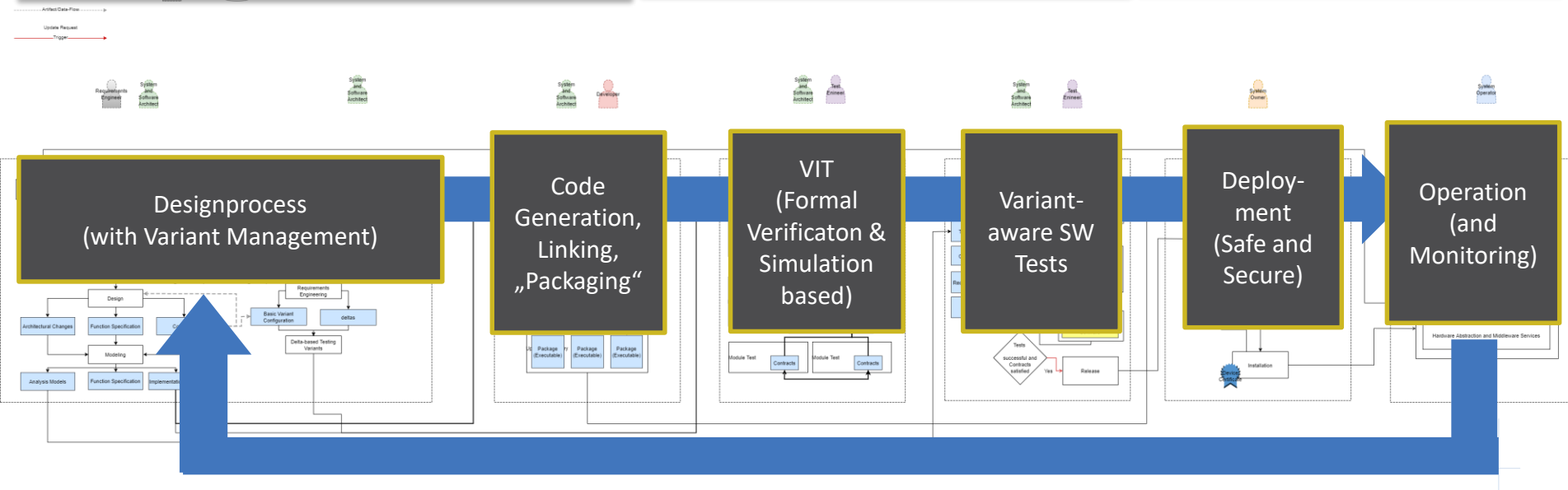
! Step-Up!CPS (Generic) Development Process



Design and Implementation

Integration and Test

Deployment/Operation



Generic process can be tailored to all three application domains



- Many Variants
- ‚Embedded‘ vs. ‚Deeply embedded‘



Automotive



Automotive



- No variants
- ‚SW-packages‘



Maritime



- No stand still
(but unplug –
update – plug)



Industry 4.0



Contract-based Design

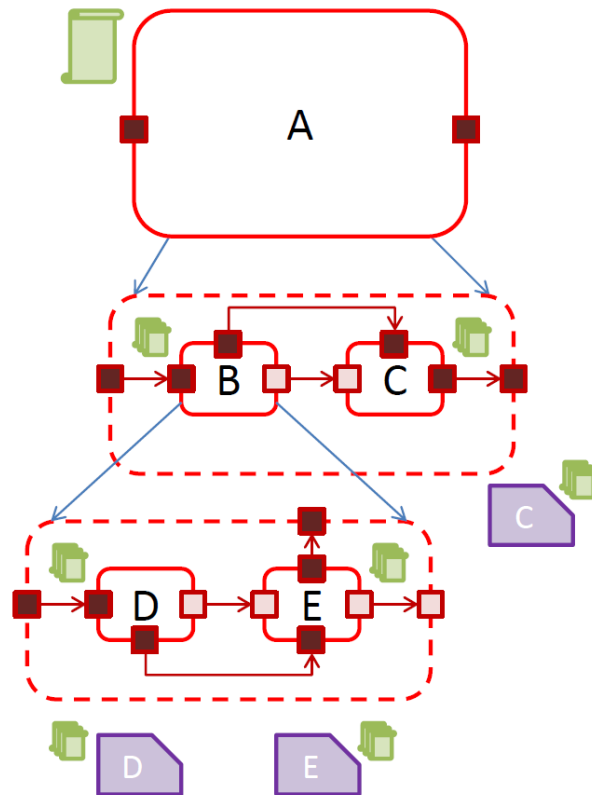
Basics

Contract: $C = (A, G)$

pair of *Assumption* A and *Guarantee* G , that specifies the behavior of one system component.

Main Idea

- > Assumptions specify demands for the components's 'environment':
 - > „If my fellow components, especially those that provide input to me, behave good, then...”
 - > „If the hardware on which I'm running provides the necessary resources, then...”
- > Guarantees specify the components own behavior
 - > „..., then I will provide correct outputs in time.”
- > Complete Contract:
 - > „If my environment behaves good, I guarantee to behave good, too”





Contract-based Design

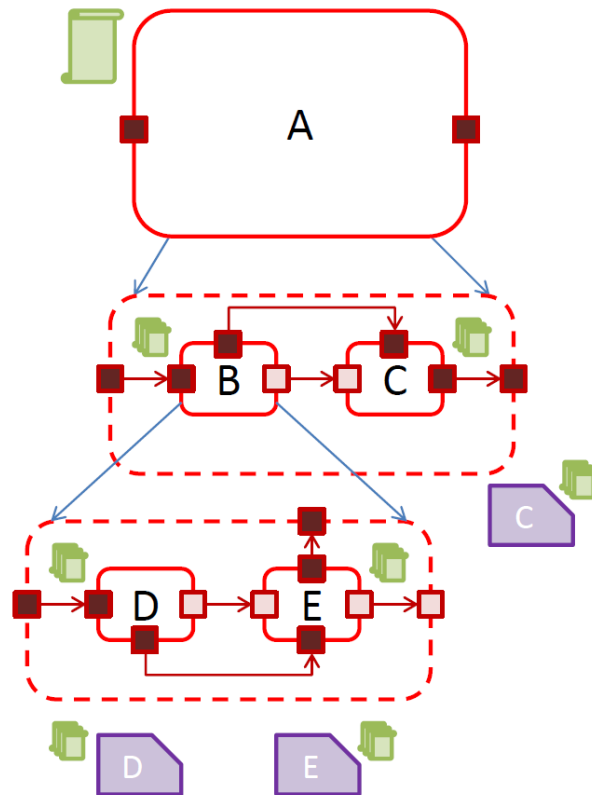
Basics

Contract: $C = (A, G)$

pair of *Assumption* A and *Guarantee* G , that specifies the behavior of one system component.

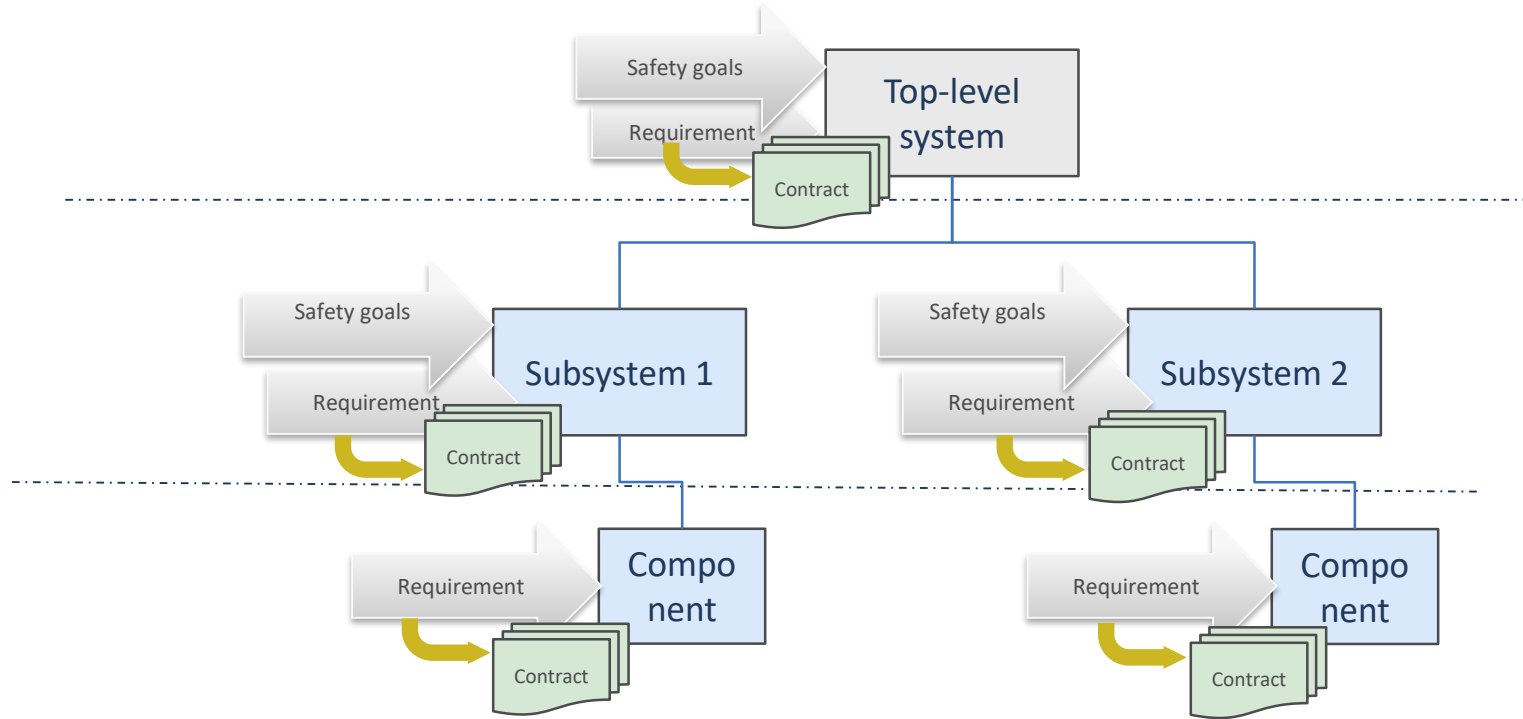
Contract operations

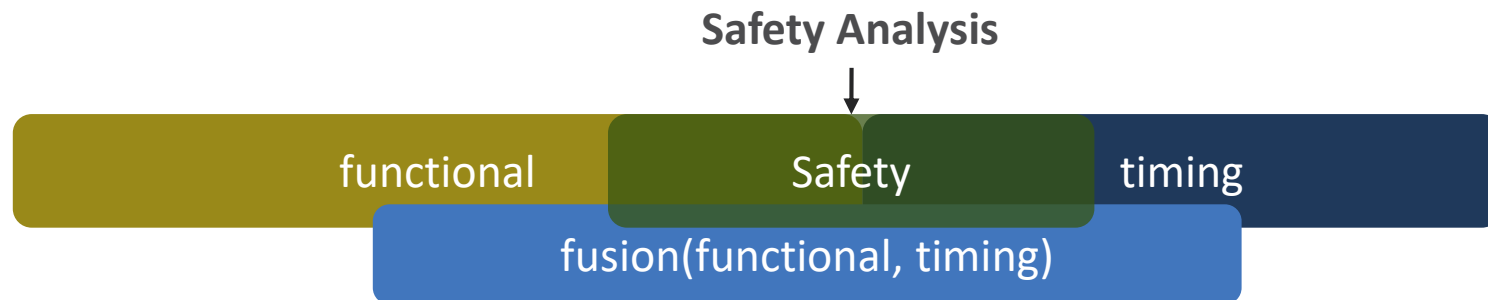
- > Implementation ($M \models C$)
 - > In any environment of component M : Whenever the assumptions of contract C hold, M behaves as promised by C 's guarantees.
- > Refinement ($C' \leq C$):
 - > Any component that implements C' also implements C (and thus can function in every environment of C)
- > (De-)Composition ($C = C_1 \otimes C_2$):
 - > The contracts C_1 and C_2 of (resp.) sub-components M_1 and M_2 together imply Contract C of ,enclosing component' M
- > Consistency, Compatibility





Derivation of Contracts





Example timing:

C_1	A	Input occurs every 2ms.
	G	Reaction(Input,Output) within [0,2000]us.

Example functional:

C_2 (LTL)	A	always (Input > 0);
	G	always (Input > 300 implies Output = 0);

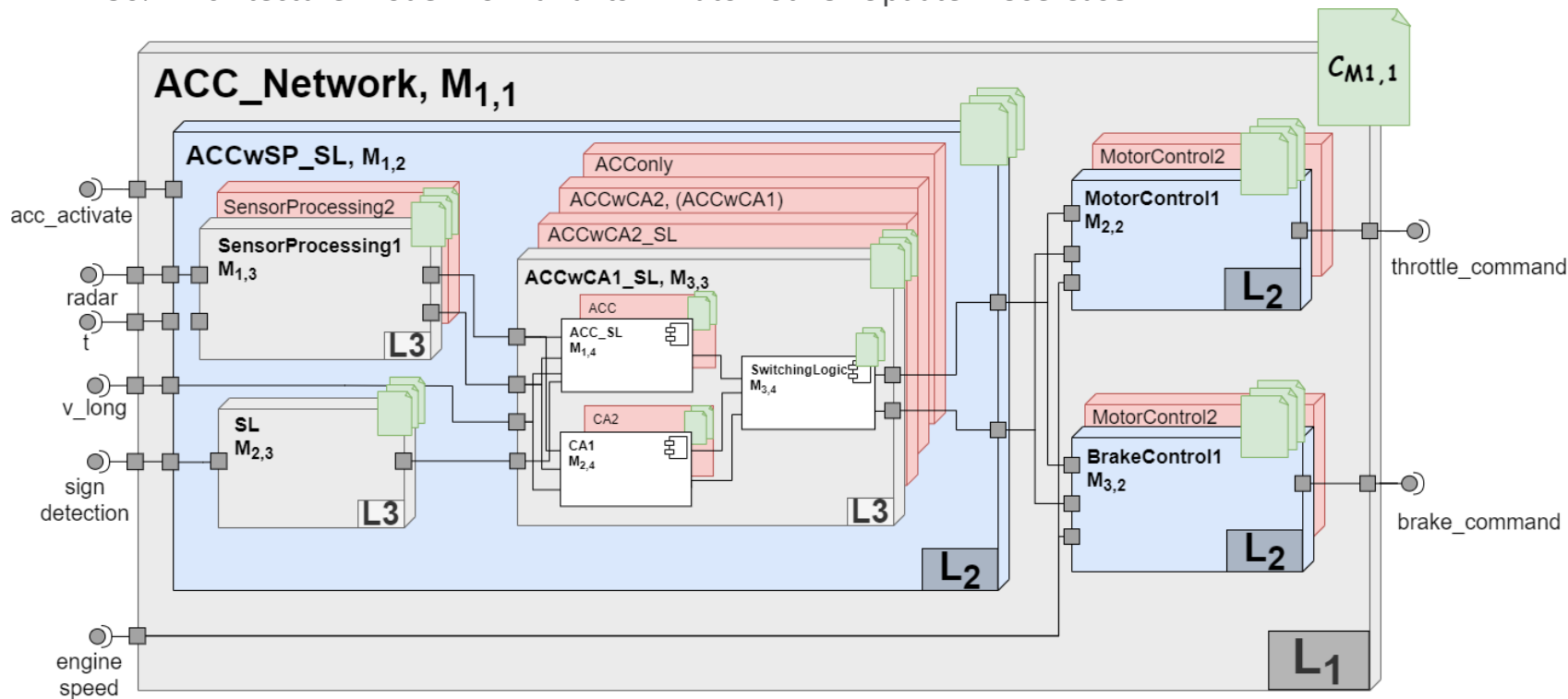
Tools for contract verification

- > formal (static) checks: e.g. OCRA (Othello Contracts Refinement Analysis)
- > simulation-based (dynamic) monitoring: e.g. MULTIC, Matlab/Simulink



Contracts in Step-UP!CPS

Ex: “150% Architecture Model” for Variants in Automotive “Updater” Use-Case



Step-Up!CPS supports two types of VITs based on Contracts:



- > **“static” VIT (formal verification)**

- > “Implementation Checks”, “Refinement Checks”, “Composition Checks”
- > If a condition is violated, counter examples are given as VIT result.
- > Also usable for Consistency and Compability Checks

- > **“dynamic” VIT**

- > conducted using monitors within
 - > simulation traces such as using SystemC
 - > Digital Twin representation
- > also suitable for executing Unit contract checks for Modules Under Update (MUU)

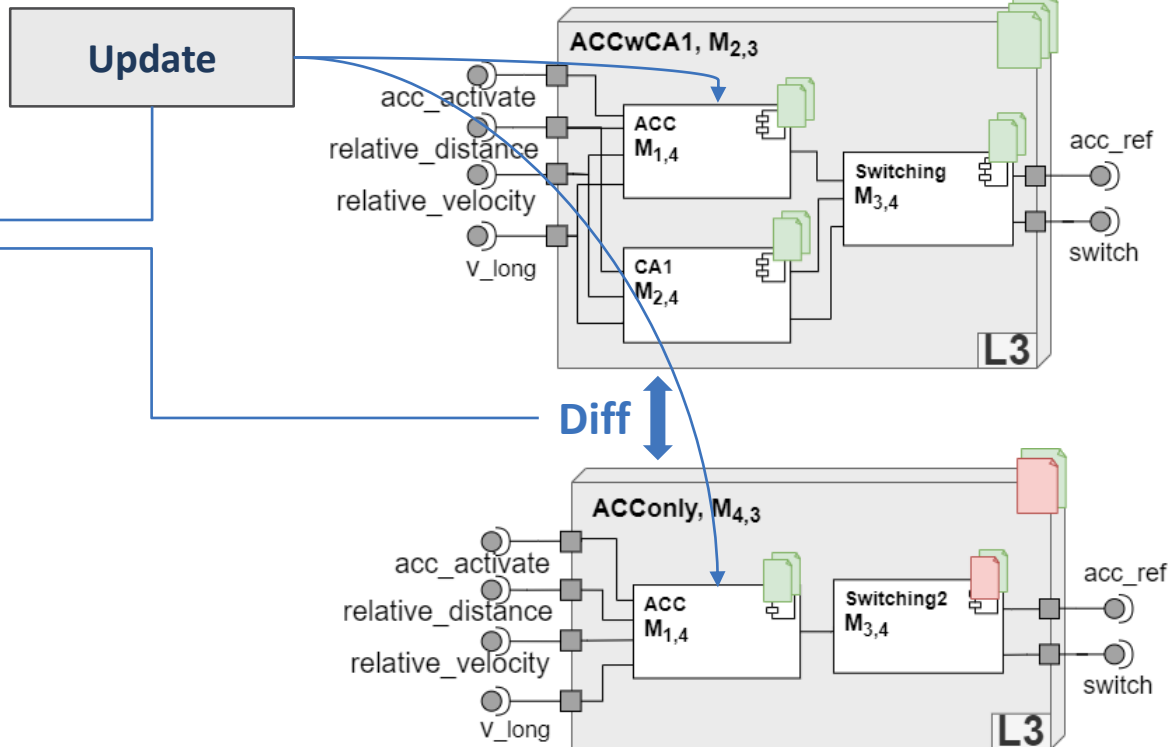


Delta-based incremental VIT

Example of two variants (basic, basic without CA)

$$\Delta = \begin{pmatrix} \Delta \text{Interface} \\ \Delta \text{Contracts} \\ \Delta \text{Impl} \end{pmatrix}$$

Impact Analysis
incl. incremental VIT



! Delta-based Test of Software Updates

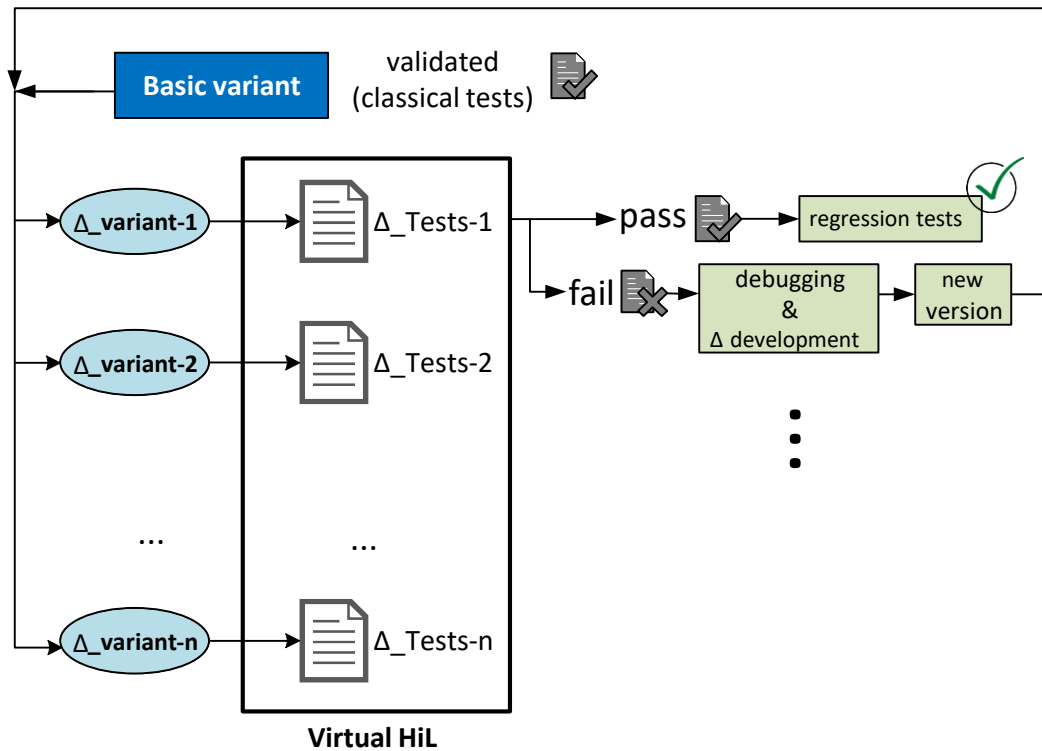
Identification of Deltas between representative variants

- > derivation of Delta Tests

Incompatibility → new version's branch → sub-release version

High variant numbers

- > tests using real hardware variants too expensive/lengthy
- > use of simulation and virtual HiL (Digital Twins)





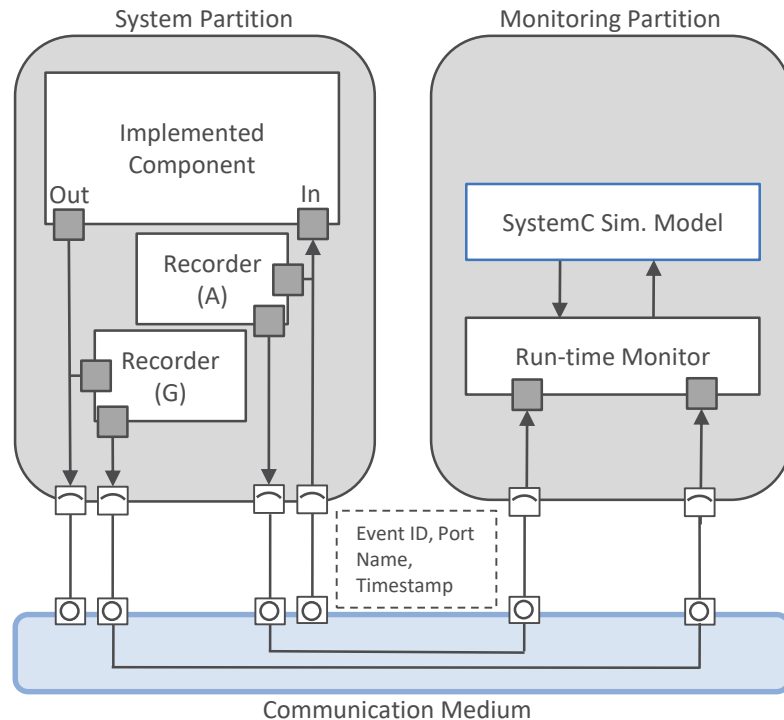
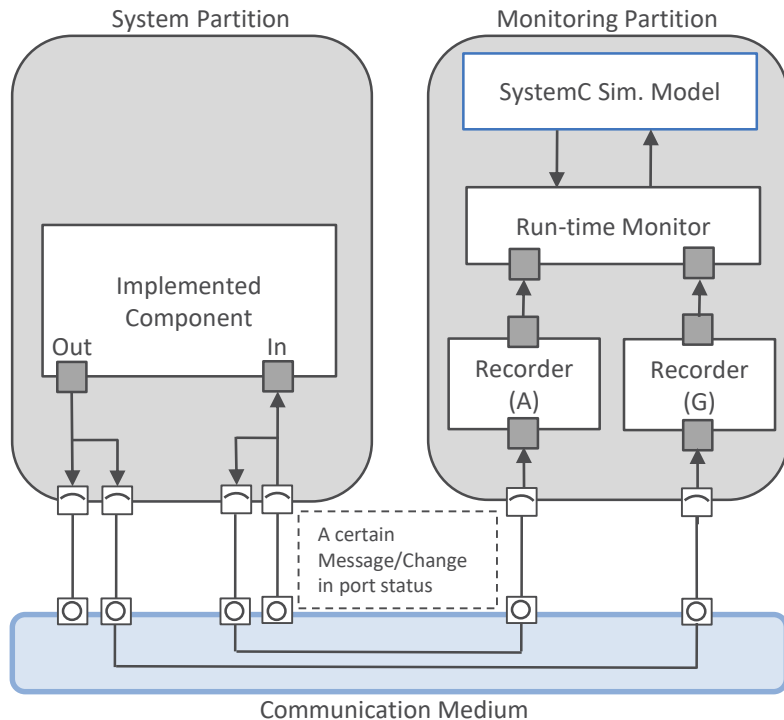
Online-Monitors Deployment

Online-Monitors automatically synthesized from Contracts

Deployment Options

Monitoring Partiton (Monitor + Recorders)

Monitoring partition + Distributed Recorders



The capability to update even safety-critical systems during runtime is a necessity for future highly automated/autonomous CPS

- > This includes the need for Virtual Engineering, incl. model-based development, virtual integration tests, scenario based virtual testing, digital twins, online monitoring, data feedback from the field...)

Contract based design is a highly useful technique to enable and extend the above technologies

- > Useful for safety (and security) analysis, virtual integration tests, variant management and test, monitor synthesis, ...

Step-Up!CPS

- > has developed a generic development process for modular updates of CPS that employs contract based design, analysis and synthesis methods to enable safe and secure system updates in the field