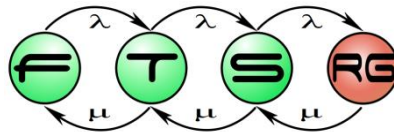
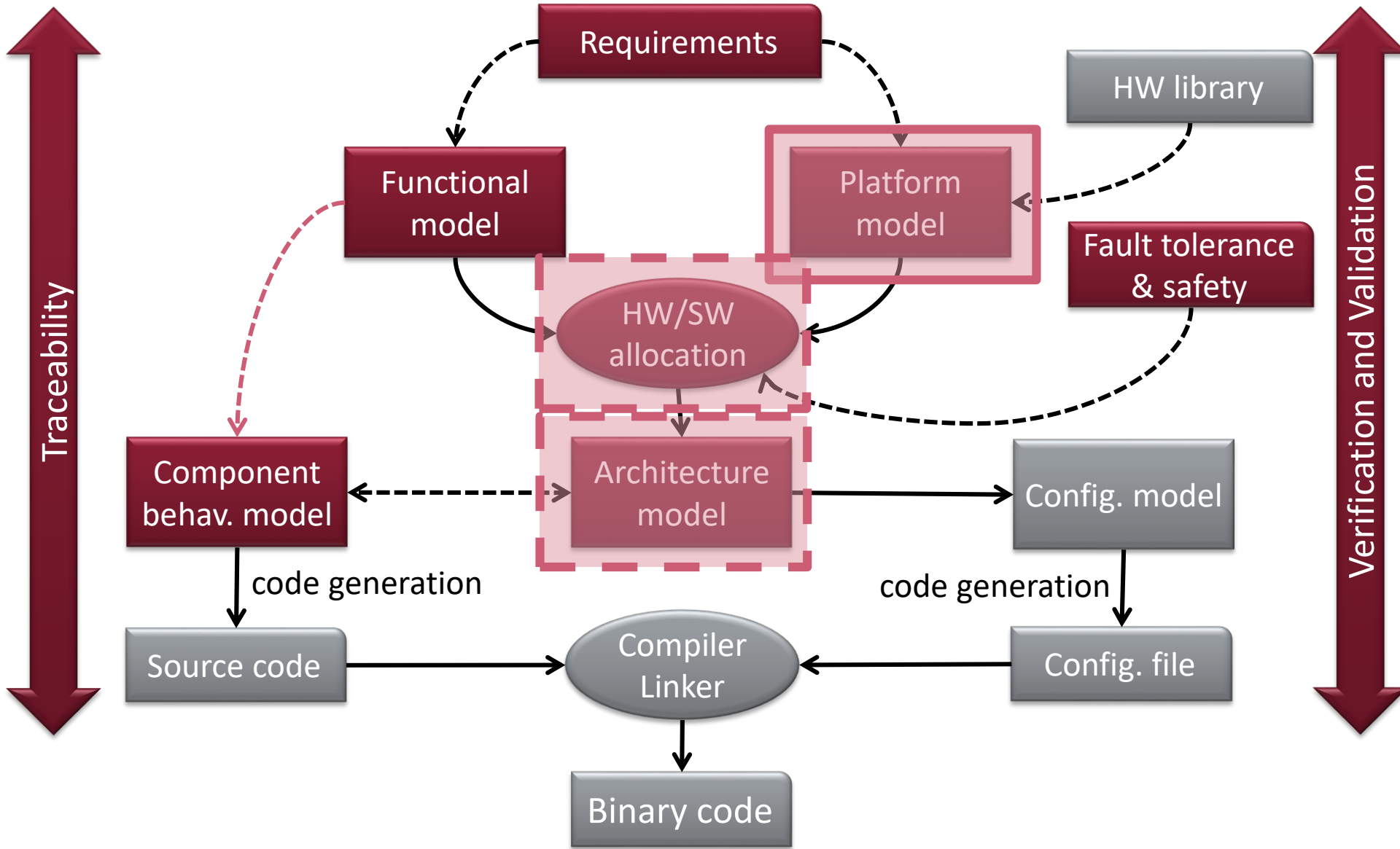


Platform modeling and allocation

Systems Engineering BSc Course



Platform-based systems design



Learning Objectives

Platform models

- Addressing non-functional requirements in the platform model
- Addressing constraints coming from the runtime platform like computation and communication resources

Allocation

- Understanding the concept of allocation
- Identify the basic design decisions made during allocation (resource allocation., scheduling, communication allocation)

Overview of architecture description languages

- AADL
- AUTOSAR

Why platform models are needed

Runtime platform

- Systems provide functions
- Functions are defined using
 - Functional models
 - Component behavior models
- How to realize these functions?

Runtime platform

- Systems provide functions
- Functions are defined using
 - Functional models
 - Component behavior models
- How to realize these functions? → in Software!

Runtime platform

- Systems provide functions
- Functions are defined using
 - Functional models
 - Component behavior models
- How to realize these functions? → in Software!
 - Maybe in hardware? (e.g., sensors, GPU, FPGA, etc.)
 - What will execute our software functions?
 - How will they be able to communicate

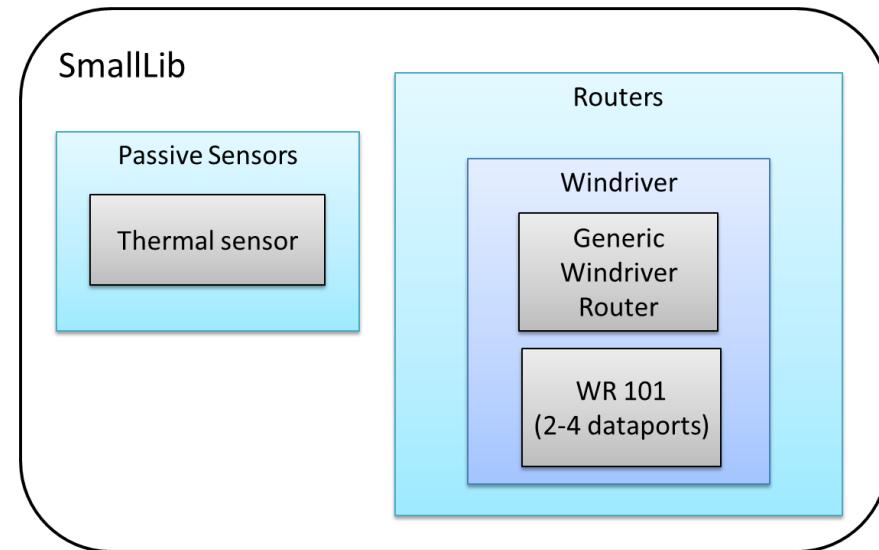
Platform model

- The platform model specifies the physical building blocks of the execution platform
 - the execution resources
 - memory, CPU, etc.
 - the available communication resources
 - Network interfaces, routers, etc.
 - the properties of the used HW elements
 - Weight
 - Availability
 - Size
 - etc.

Defining the platform model I.

■ Resource capturing phase

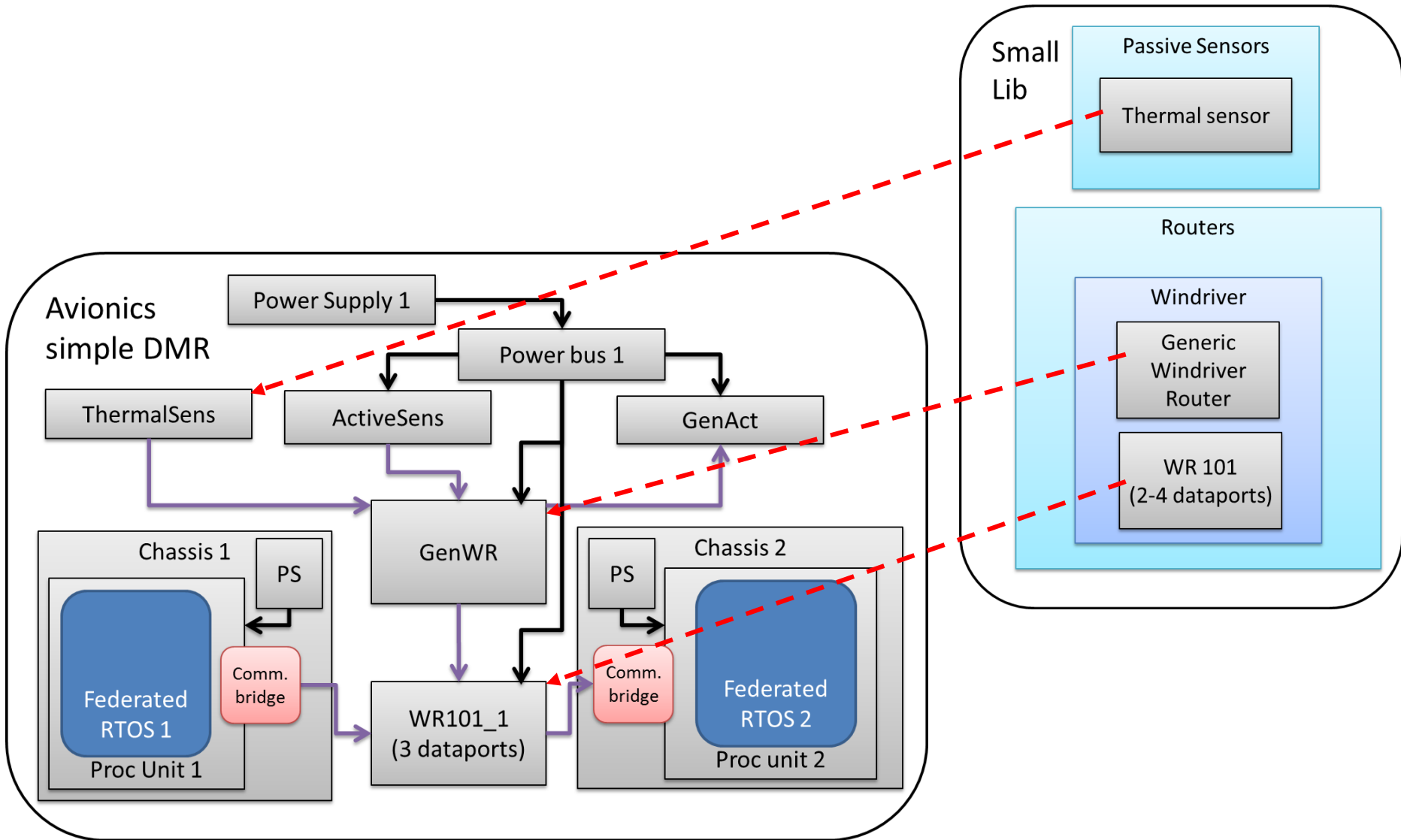
- Specification of reusable hardware entities
 - Coming from HW libraries/technical dictionaries
 - Defined by HW designers within the project
- ➔ atomic hardware units of the execution platform
 - Embedded systems: Processor, Communication controller
 - Define hardware properties



Defining the platform model II.

- Platform composition phase
 - (Already available HW design → only modifications)
 - Definition from bottom-up based on the atomic building blocks
- Similar modeling task as the functional component definition BUT
 - Connecting blocks == physical linkage
 - Part-whole relationship == physical containment
 - Physical HW properties are needed to be taken into consideration
 - Size, weight, number of ports, etc.

Defining the platform model II.

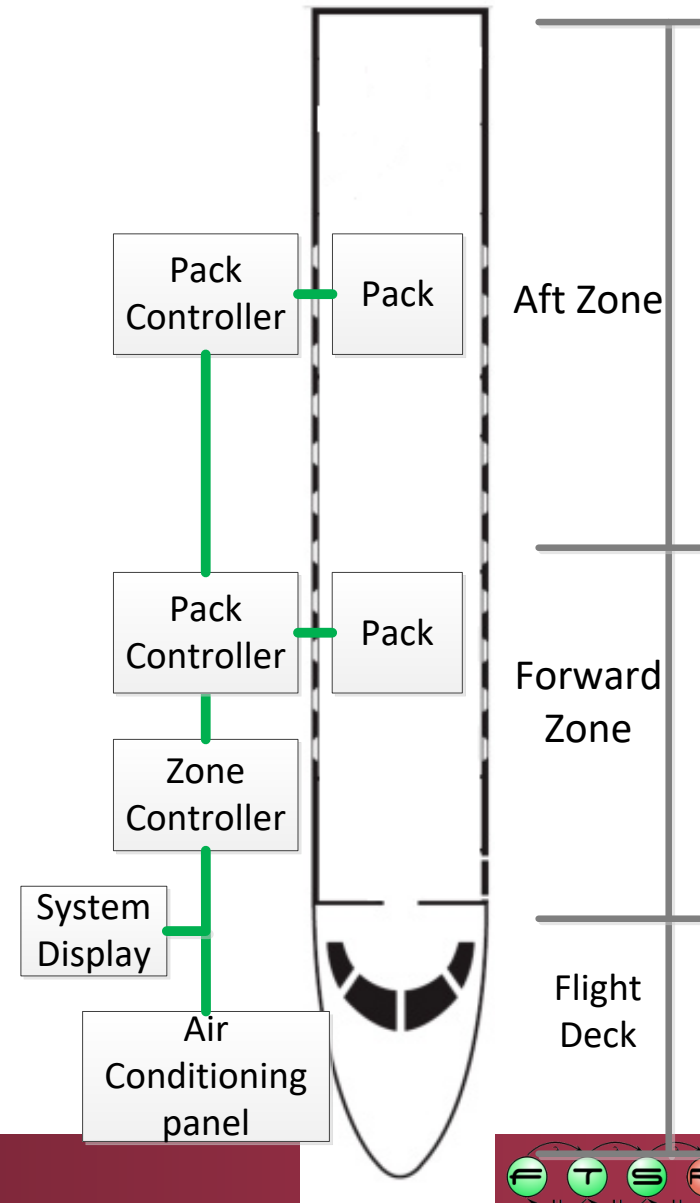
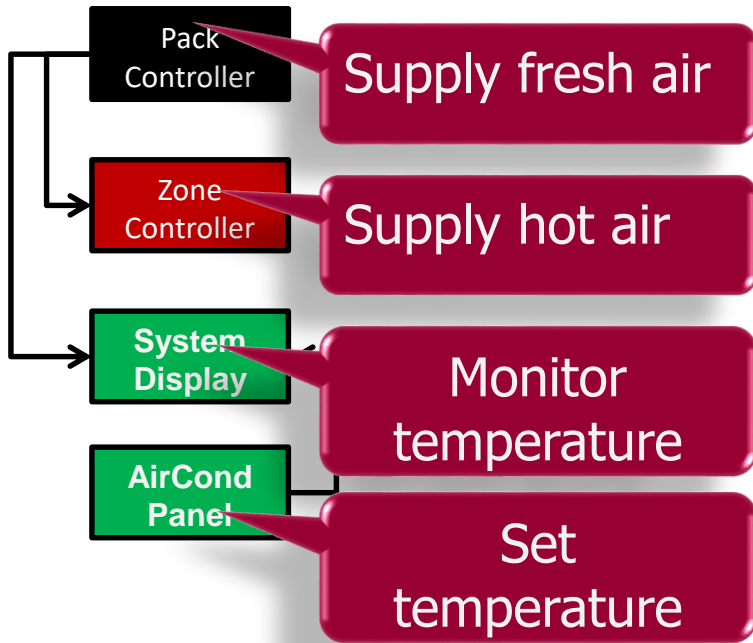


Functions to Platform allocation

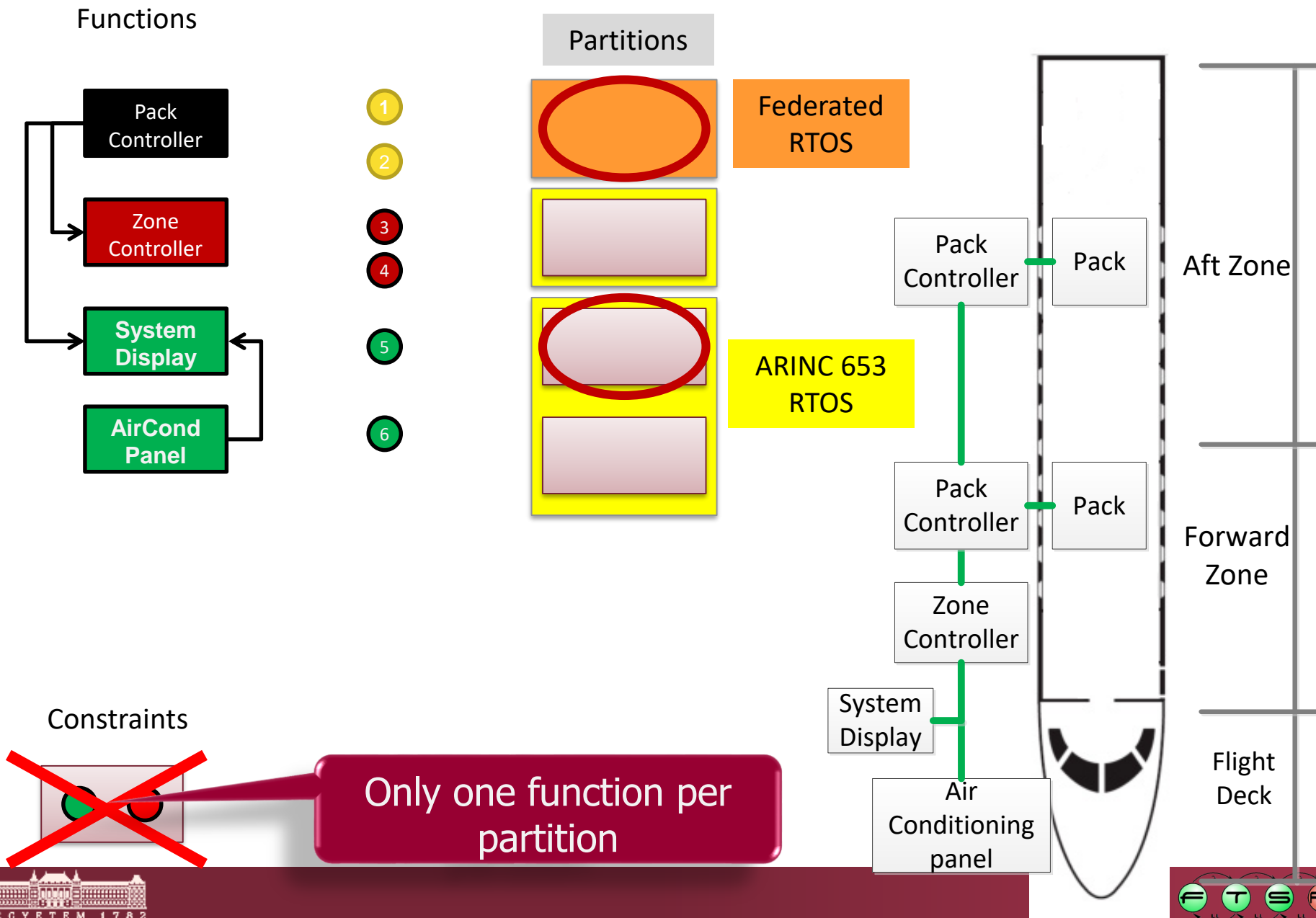
Usually HW-SW allocation

Allocation example

Functions

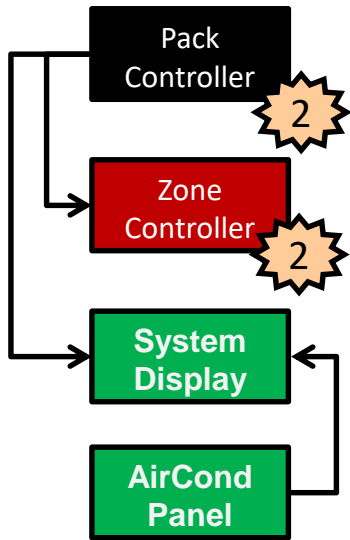


Allocation example – functions to partitions



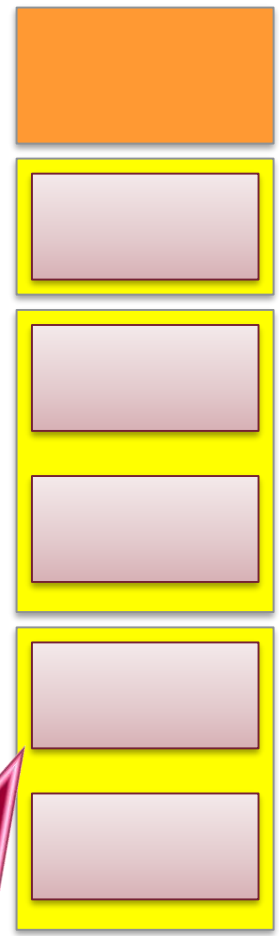
Allocation example – functions to partitions

SW functions



- 1
- 2
- 3
- 4
- 5
- 6

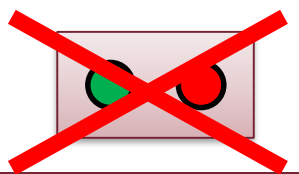
Partitions



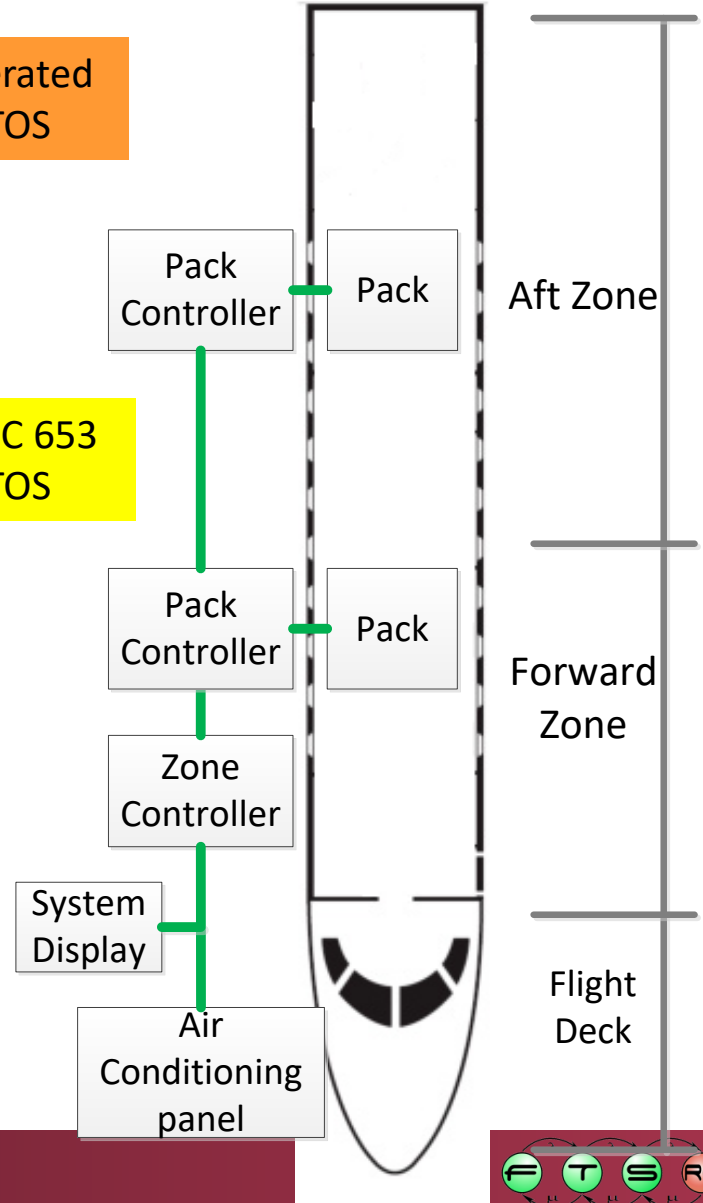
Federated RTOS

ARINC 653 RTOS

Constraints

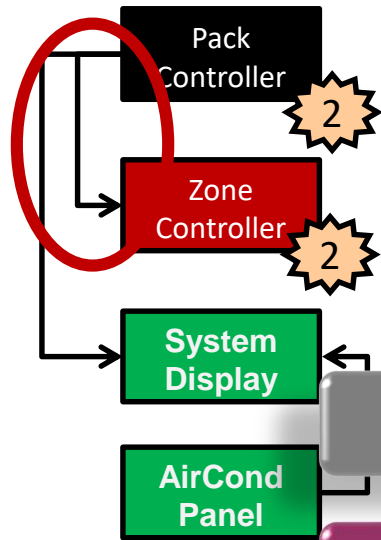


Modify HW architecture for more resources



Allocation example – communication channels

SW functionality

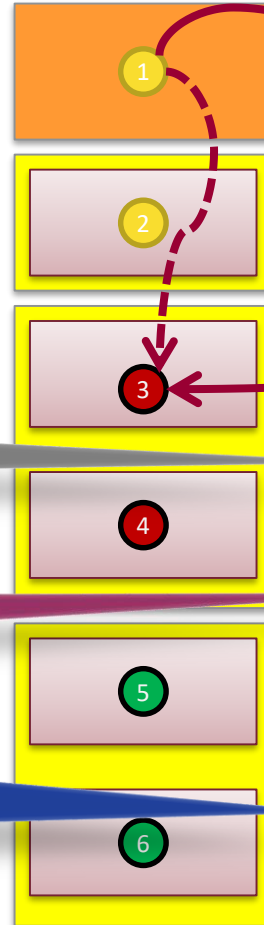


ARINC 429

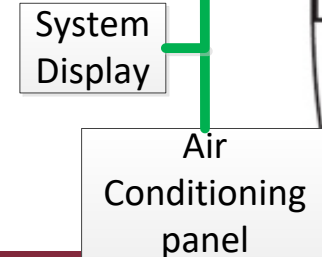
AFDX

ARINC 653 ports

HW Communication channels



One possible candidate is selected



Forward Zone

Flight Deck

Allocation

- **Input:**
 - Functional model + platform model
 - Additional non-functional constraints
- **Output:**
 - System Architecture
- The *System Architecture* defines for each instance of a Function
 - *where and when to execute*
 - *when to communicate*
 - *and on which bus*

Where and when to execute

- Allocate the functions to their designated execution resource
 - Processor, GPU, server, node, etc.
- Schedule the execution of functions
 - Based on their required execution window
 - **Major driver of the allocation process**
- Constraints (usually) taken into consideration
- Platform (HW)
 - Available memory
 - CPU performance
 - Redundancy
- Functional (SW)
 - Memory required
 - Execution window required
 - Safety aspects
 - E.g., criticality levels

When to communicate and on which bus

- Allocate Function model level communication means to platform communication resources
 - Information flow to bus mapping
 - Data/message mapping to platform representation
 - Scheduling
 - Messages, buses, routers
 - **Major driver of the allocation process**
 - Constraints (usually) taken into consideration
- Platform (HW)
 - Connectivity
 - comm. architecture
 - Routing
 - Supported modes
 - Bandwidth & Speed
 - Precision
 - Data mapping
 - Redundancy
 - Independent paths
- Functional (SW)
 - Message properties
 - size
 - priority
 - Communication mode
 - 1-1, 1-n, n-n
 - Safety aspects
 - WCET

Additional aspects of the allocation

- Multi-level allocation
 - Complexity is handled on multiple abstraction-level → allocation is handled between all hierarchies
- Resulting System Architectures are used for validating system level functional/non-functional aspects
 - Timing requirements, safety requirements, etc.
 - Used methods: Static checks, simulations, HiL, etc.
- No perfect allocation → Multi-dimension optimization problem
 - Design Space Exploration

Architecture Description Languages

ADLs

Abstract

- “The architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.”
- (no universal agreement on what ADLs should represent)
- Software Architecture in Practice,
Bass, Clements, and Kazman

Architecture Analysis and Design Language (AADL)

AADL

- Architecture Analysis and Design Language (AADL) is a standard architecture modeling language
 - Avionics
 - Aerospace
 - Automotive
 - Robotics
- Component based notation
 - Task and communication architecture
- Designed for modeling and analysis in mind
- SAE standard (AS 5506A)

- First was called Avionics Architecture Description Language
 - Derived from MetaH created by Honeywell
- V1 version in 2004
- V2 version in 2009

AADL

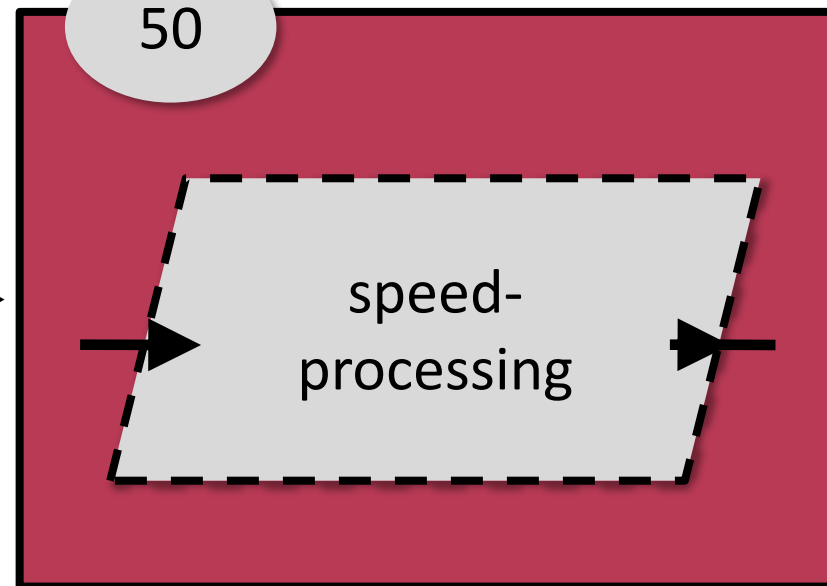
- Based on the component-connector paradigm
- Key Elements:
- Core AADL language standard (V2-Jan,2009, V1-Nov 2004)
 - Textual & graphical, precise semantics, extensible
- AADL Meta model & XMI/XML standard
 - Model interchange & tool interoperability
- Annexes Error Model Annex as standardized extension
 - Error Model Annex addresses fault/reliability modeling, hazard analysis
- UML 2.0 profile for AADL
 - Transition path for UML practitioner community via MARTE
- EMF representation also available (without EFeatureMap!)

AADL

- Precise execution semantics for components
 - Thread, process, data, subprogram, system, processor, memory, bus, device, virtual processor, virtual bus
- Continuous control & event response processing
 - Data and event flow, synchronous call/return, shared access
 - End-to-End flow specifications
- Operational modes & fault tolerant configurations
 - Modes & mode transition
- Modeling of large-scale systems
 - Component variants, layered system modeling, packaging, abstract, prototype, parameterized templates, arrays of components and connection patterns
- Accommodation of diverse analysis needs
 - Extension mechanism, standardized extensions

AADL Representation Forms

```
thread speed_processing
features
  raw_speed_in: in
data port;
  speed_out: out data
port;
properties
  Period => 50 ms;
end data_processing;
```



```
<ownedThreadType name=„speed_processing“>
  <ownedDataPort name=“raw_speed_in“/>
  <ownedDataPort name=“speed_out“ direction=“out“/>
  <ownedPropertyAssociation property=“Period“
    <ownedValue xsi:type=“aadl2:IntegerLiteral“
      value=“50“ unit=“ms“
    </ownedValue>
  </ownedPropertyAssociation>
</ownedThreadType>
```

AADL Language Elements

- Core modeling
 - Components
 - Interactions
 - Properties
- Engineering support
 - Abstractions
 - Organization
 - Extensions
- Infrastructure

- Strong modeling capabilities for embedded SW and Computer systems

AADL Components

- Top element **system**

Example:

```
package F22Package
  public
    system F22System
  end F22System;
  system WeaponSystem
  end WeaponSystem;
  system implementation F22System.impl
    subcomponents
      weapon: system WeaponSystem;
    end F22System.impl;
  end F22Package;
```

AADL SW Components


- **System** – hierarchical organization of components
- **Process** – protected address space
- **Thread group** – logical organization of threads
- **Thread** – a schedulable unit of concurrent execution
- **Data** – potentially sharable data
- **Subprogram** – callable unit of sequential code



System



Process



Thread group



Thread



Data



Subprogram

AADL SW Components

■ Process

- Protected virtual address space
- Contains executable program and data
- Must contain 1 thread

■ Thread

- Concurrent tasks
- Periodic, aperiodic, sporadic ,background, etc.
- Interaction through port connection, subprogram calls or shared data access
- errors: recoverable, unrecoverable

AADL SW Components

■ Ports and Connections

- Data (non queued data), Event (queued signals) or Event data (queued messages)
- Complex Connection hierarchies through components
- Timing
- Feature groups

■ Data

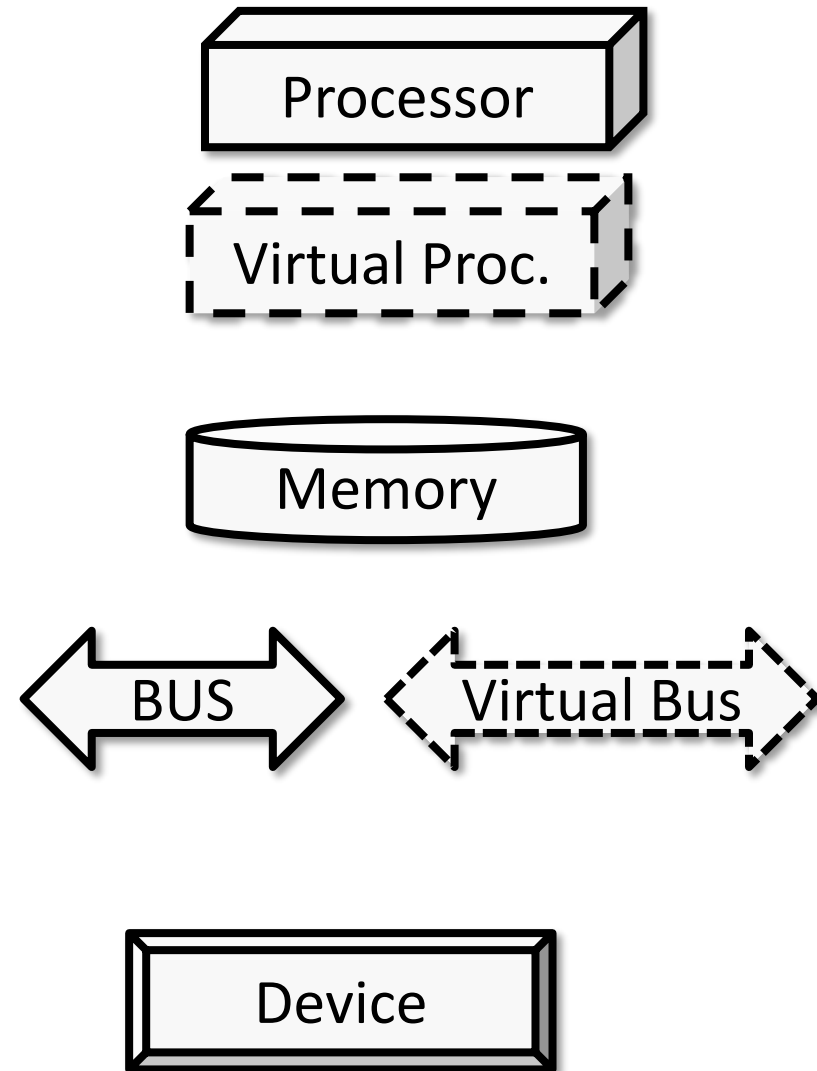
- Optional but makes the analysis more precise

■ Flows

- Logical flow of data and control

AADL Computer Components

- **Processor / Virtual Processor** – Provides thread scheduling and
- **Memory** – provides storage for data and source code
- **Bus / Virtual Bus** – provides physical/logical connectivity between
- **Device** – interface to external environment



AADL Computer Components

- “Real” HW components
 - Bus transmission time, latency,
 - Processor timing, jitter
 - Memory capacity
 - Etc.
- Logical resources
 - Thread scheduling of a processor
 - Communication protocol over network connection (modeled as bus)
 - Transactional memory (modeled as memory)

AADL Computer Components

■ Processor

- As HW
 - MIPS rating, size, weight, clock, memory manager
- As Logical resource
 - Schedule threads → scheduling policies and interruption
 - Execute SW

■ Bus

- As HW
 - Physical connection inside/between HW components
- As logical resource
 - Protocol, which are used for the communication

■ Memory

- Processes must be in memory
- Processors need access to memory

■ Device Components

- Represents element that are not decomposed further
- Sensors/Actuators
- Device Driver

AADL Binding

■ Binding

- Bringing SW models and the execution platform together
- Virtual processors → can be subcomponents of other virtual processors → ARINC653 partitioning
- Hierarchical Scheduling
- Virtual buses to physical ones
 - One-to-one
 - Many-to-one

Summary

- After 15 years of mainly DoD research it is getting mature enough
- Many pilot project uses AADL
 - FAA
 - DoD
 - Lockheed Martin
 - Rockwell Collins (Steven P. Miller)
- Many research paper on formal analysis, simulation and code generation
- Ongoing harmonization with SysML and MARTE

AUTOSAR

History

- AUTomotive Open System ARchitecture
- Started in 2002
- BMW, Bosch, Daimler, Conti, VW, + Siemens
- Industrial standardization group
 - Current standard version: 4.0 (end 2009)
 - Currently we use 3.1 (end 2008)
- Members: OEMs, Tool vendors, Semiconductor manufacturers ☐ Europe-dominated
- Scope
 - Modeling and implementation of automotive systems
 - Distributed
 - Real-time operating system
 - String based interaction with HW and environment
- Out of scope
 - GUI, Java, internet connectivity, File systems, Entertainment systems, USB connectivity etc.

Key Concepts of AutoSAR

- A standard runtime architecture
 - **component-oriented**
 - layered
 - extensible
 - New functionalities
 - New components (component implementations)
 - all major interfaces standardized
 - Standardized Run Time Environment (RTE)
- A standard modeling and model interchange approach
 - follows the principles of model-driven design
 - supports the interchange of designs
 - supports the collaborative development
 - Between different developers,
 - Teams,
 - And even companies
- Conformance test framework
 - assuring the conformance to the standard
 - Still evolving – new in version 4.0

High-level design flow

High-level design process

High-level
SW modeling

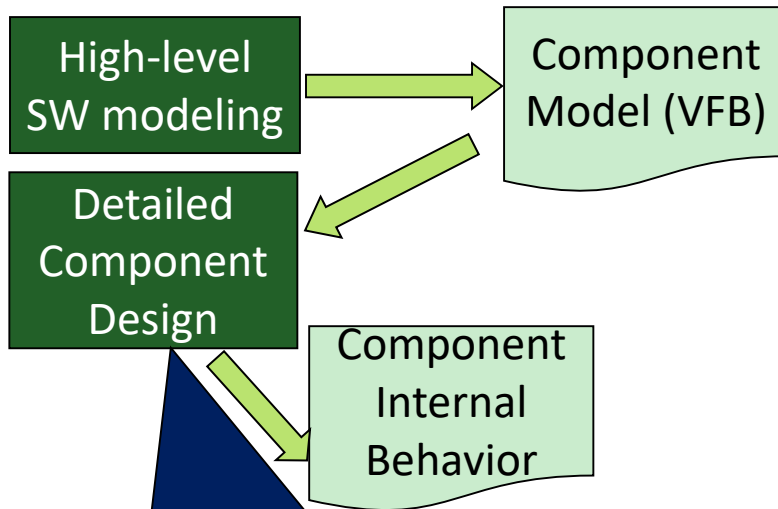


Component
Model (VFB)

High-level software modeling

- Definition of
 - components
 - component ports
 - port interfaces
 - data types – logical
- Result
 - Virtual Functional Bus (VFB)-level software model

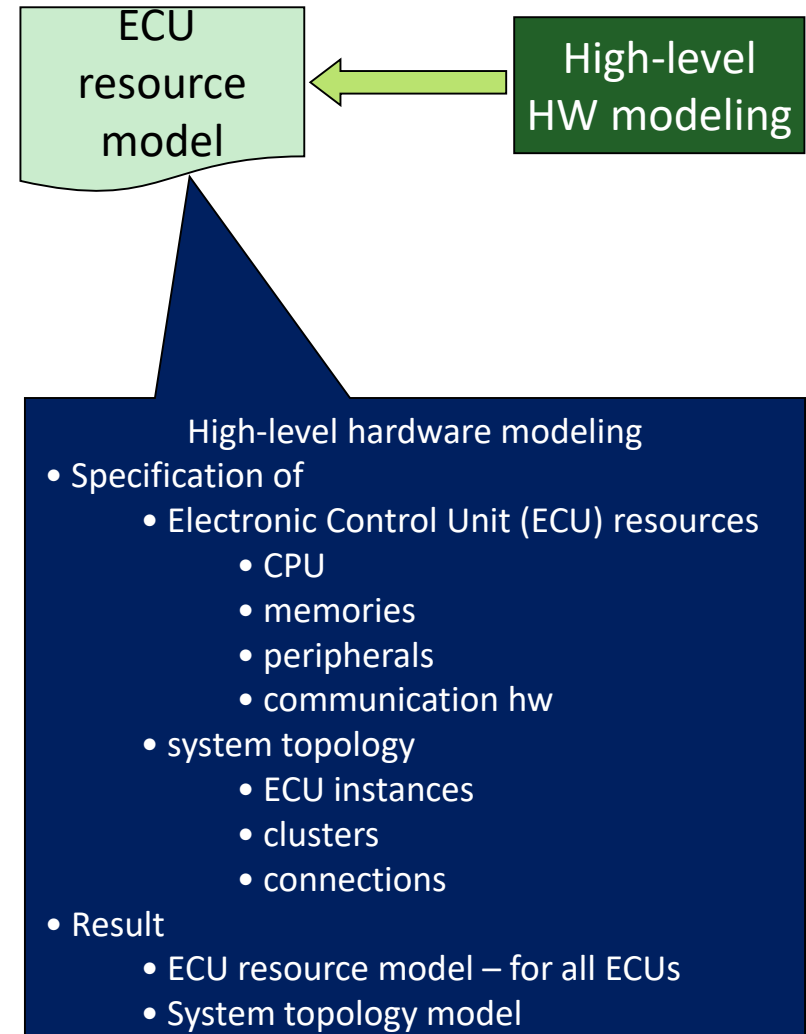
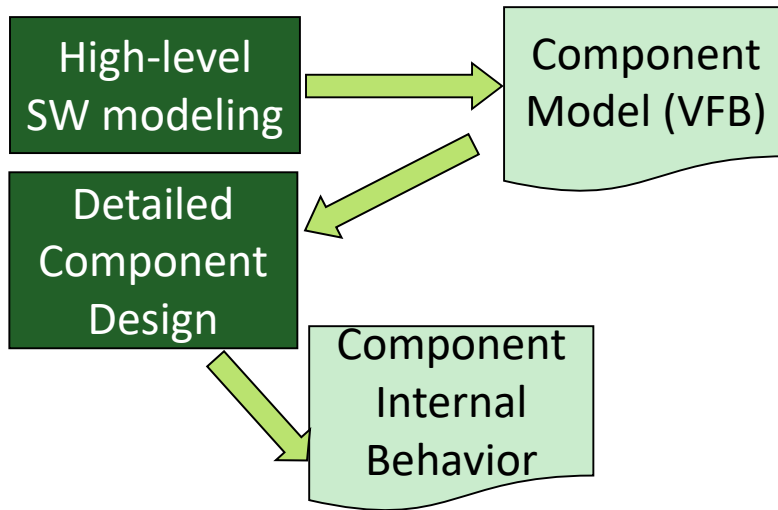
High-level design process



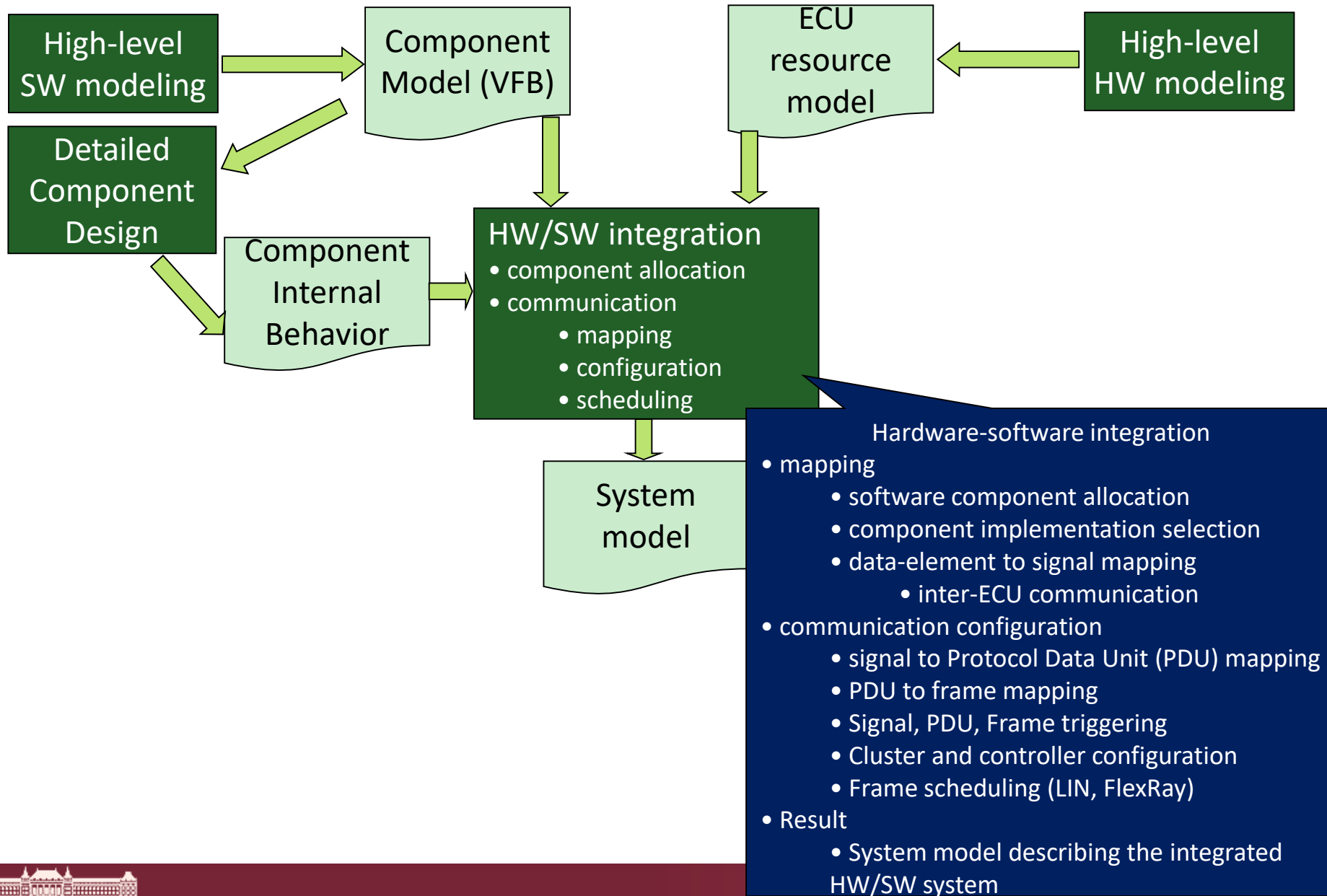
Detailed component design

- Specification of
 - component internal behavior
 - functional breakdown
 - implementation/use of ports
- Non-AutoSAR
 - specification of detailed behavior
 - any tool can be used
 - UML
 - Simulink
 - etc.
- Result
 - AutoSAR component internal behavior model
 - Non-AR: behavioral models/design

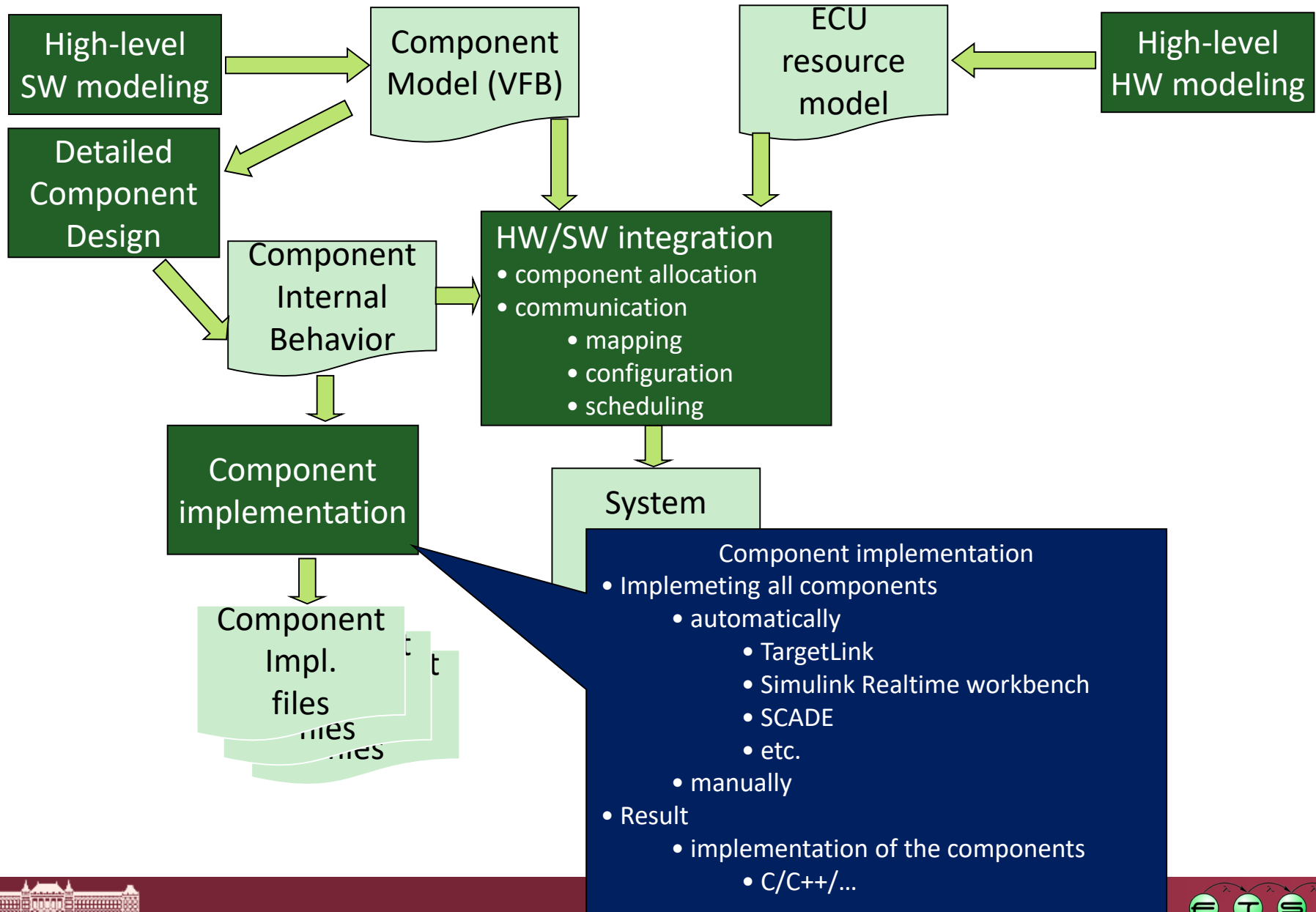
High-level design process



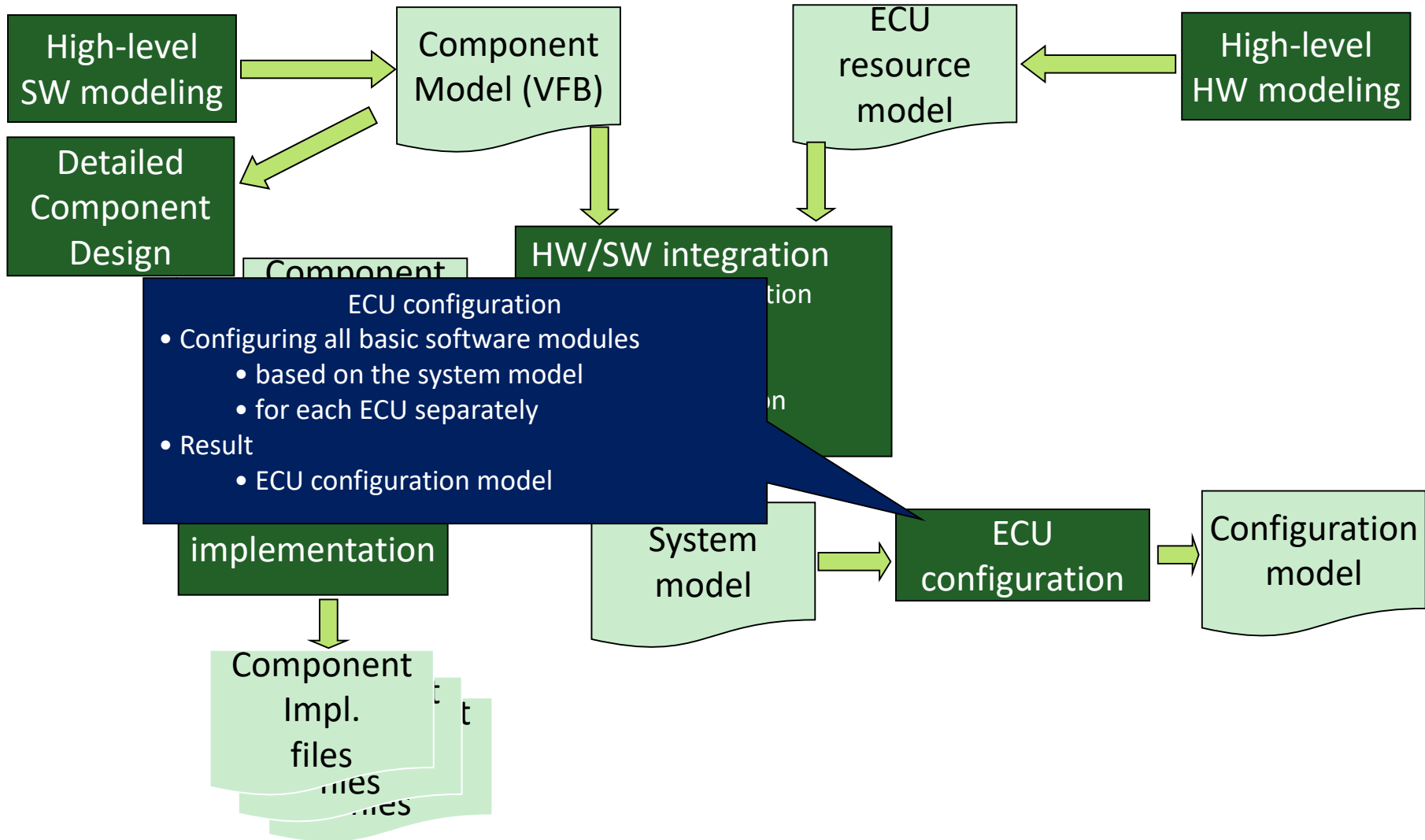
High-level design process



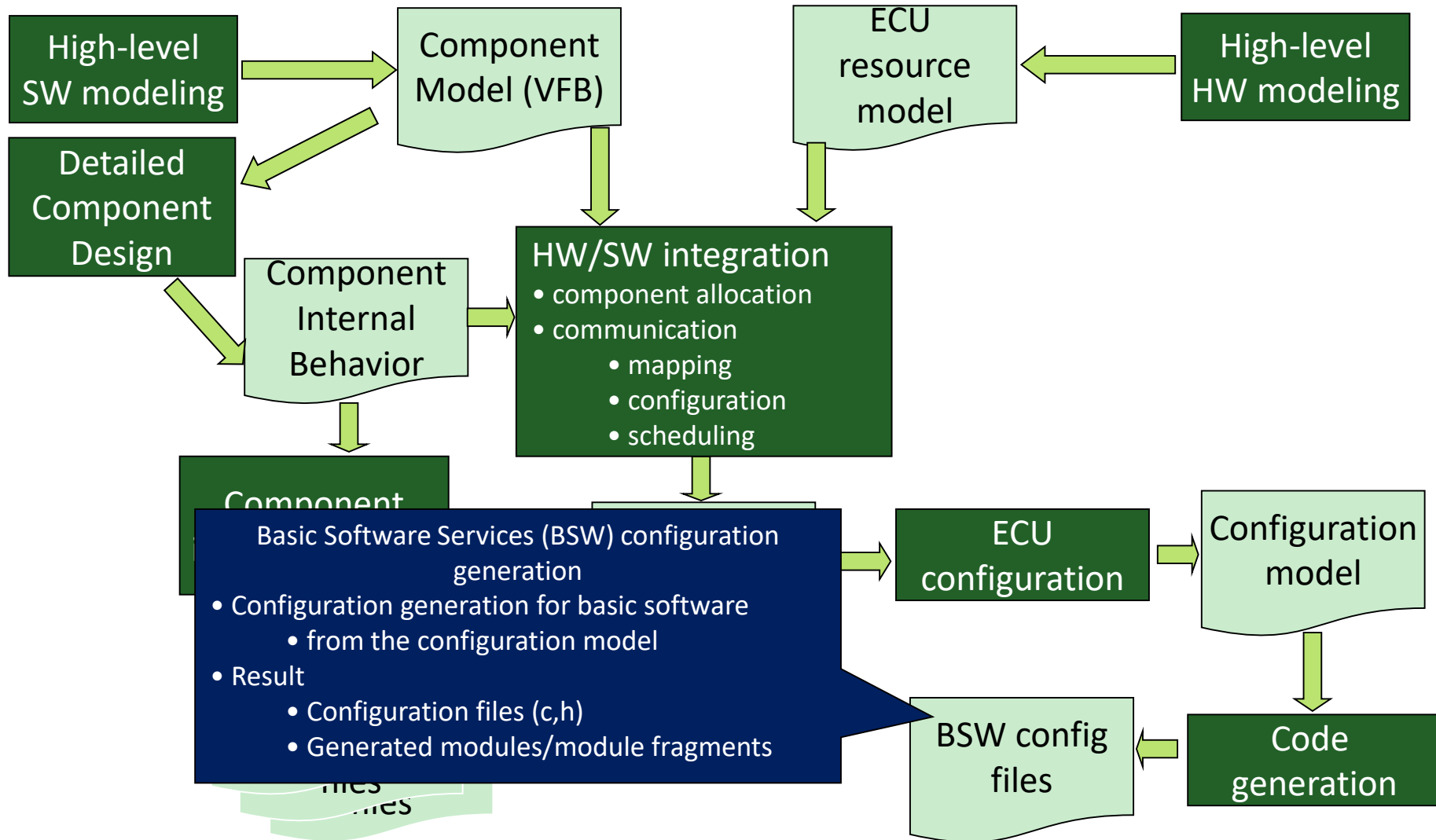
High-level design process



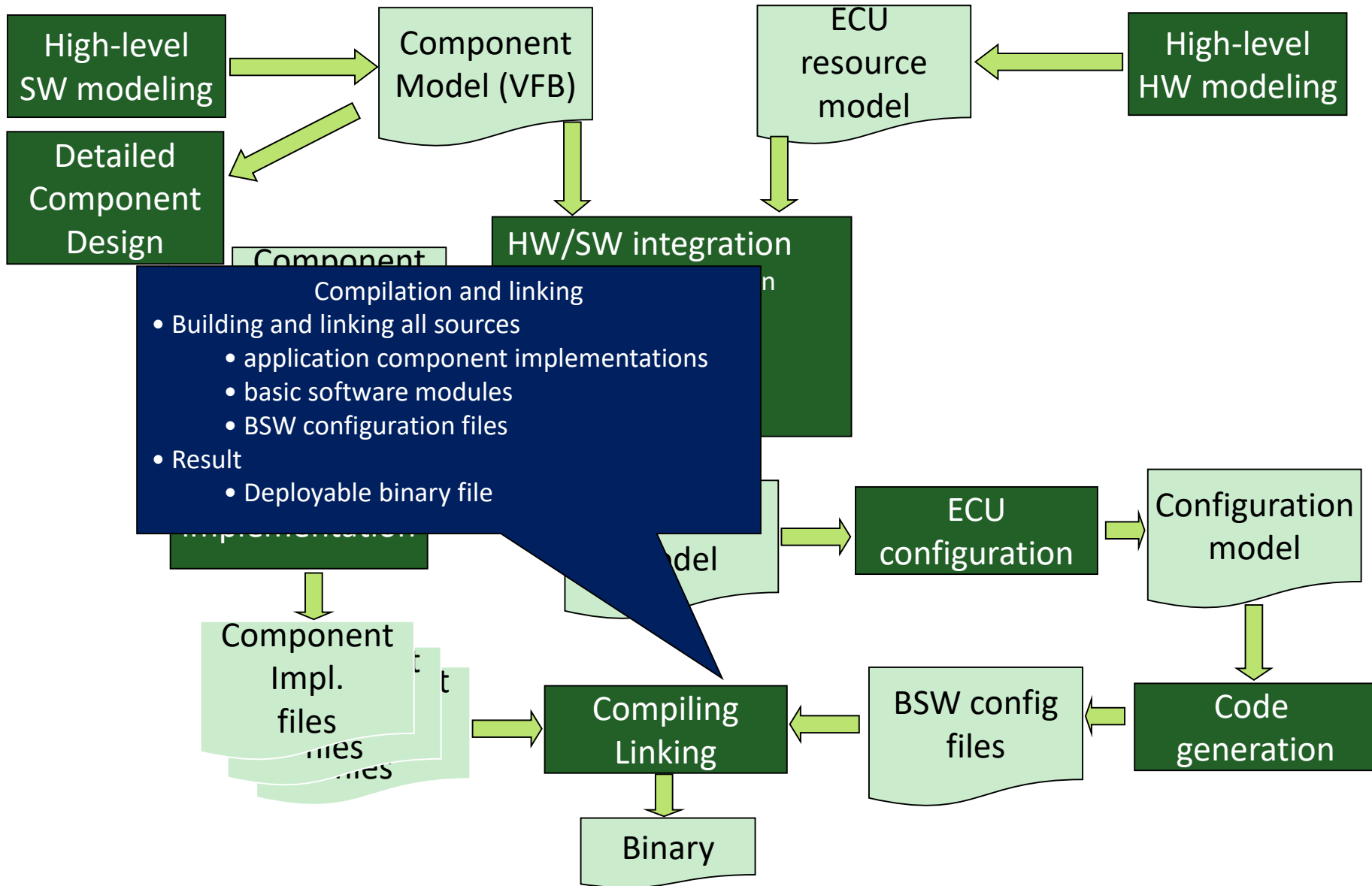
High-level design process



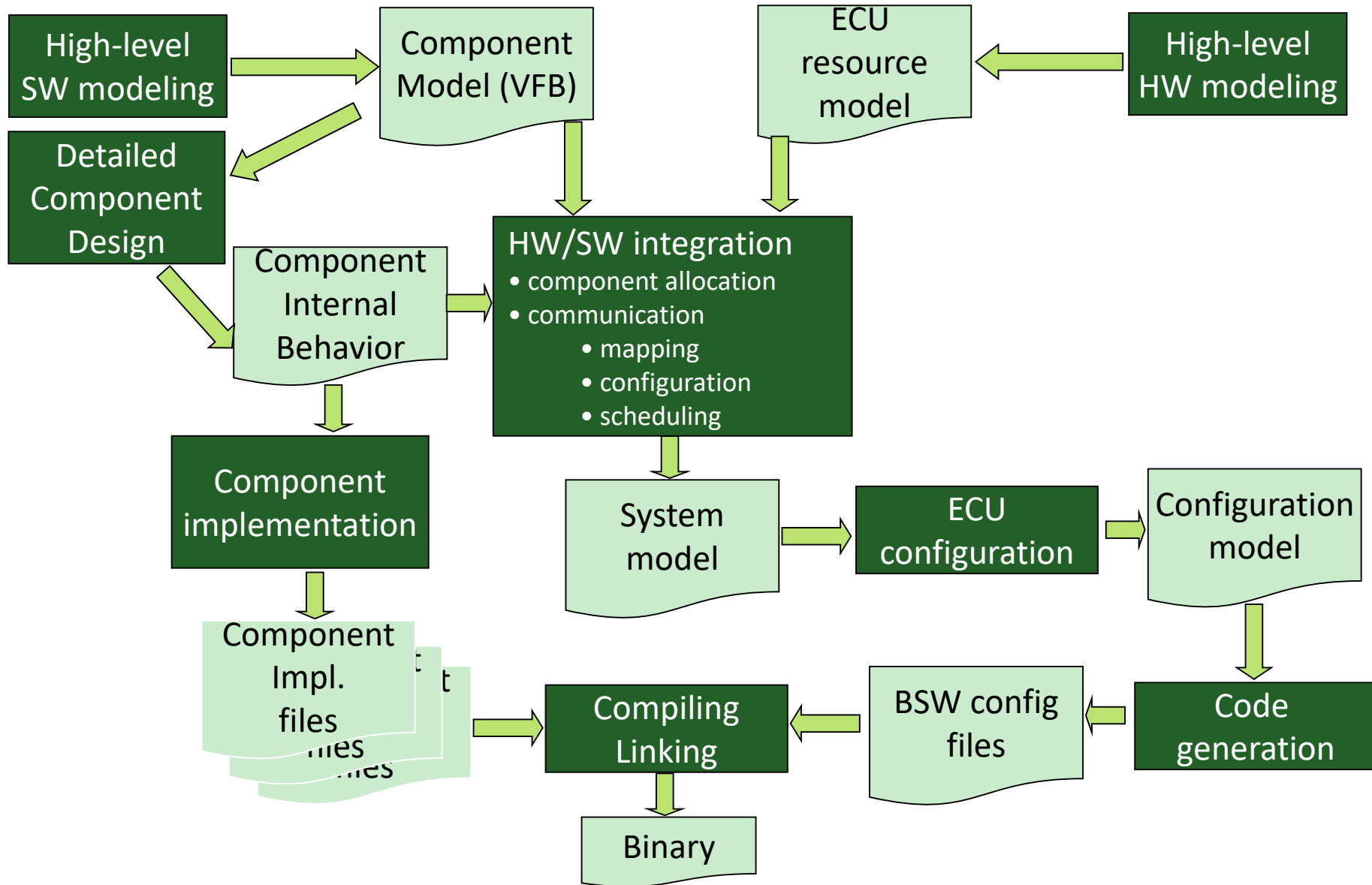
High-level design process



High-level design process



High-level design process



Models in the design flow

- Software Component Template
 - Components, ports, interfaces
 - Internal behavior
 - Implementation (files, resource consumption, run time, etc.)
- ECU Resource Template
 - Hardware components, interconnections
- System Template
 - System topology, HW/SW mapping
 - Comm. matrix

Models in the design flow 2

- Basic Software Module Template
 - BSW modules
 - Services
 - Schedulable entities
 - Resource consumption
- ECU Configuration Parameter Definition Template
 - Configurable parameters of BSW modules
- ECU Configuration Description Template
 - Actual configurations of BSW modules
 - Based on the ECU Parameter Definition

AUTOSAR vs. UML/SysML/... modeling

- AUTOSAR defines models with
 - Domain Specific Constructs
 - *Precise syntax*
 - Synthesizable constructs
 - Direct model -> transformations
 - Direct model -> detailed model mappings
 - Different abstraction levels
 - From Virtual Function Bus to configuration
- Result
 - Models *are* primary design *and* implementation artifacts
 - More precise, consistent modeling should be done

AUTOSAR Components

Component-oriented design

- What is a component?
 - “A component is a self contained, reusable entity that encapsulates a specific functionality (and/or data), and communicates with other components via explicitly defined interfaces.”
- AUTOSAR uses the term *component* for application-level components
 - Elements related to the high-level functionality of the system under design
- Basic software (middleware) components are called *modules*.
 - Standard elements of the AUTOSAR architecture

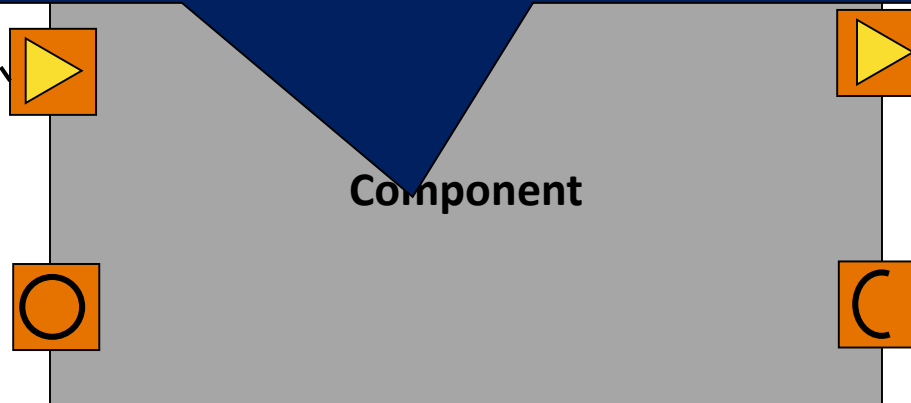
Component-based approach

Component

- Encapsulates a specific functionality
- Different kinds
 - Composite component – hierarchical refinement
 - Application SW component – generic, high level functionality
 - Sensor/actuator SW-C – handling sensor or actuator data
 - ECU HW abstraction – higher level device driver and abstraction
 - ComplexDeviceDriver – time-critical, low-level driver
 - Calibration parameter SWC – collects system calibration parameters
 - Service SWC – represents a basic software module from the service layer

<<interface>>
SenderReceiver1

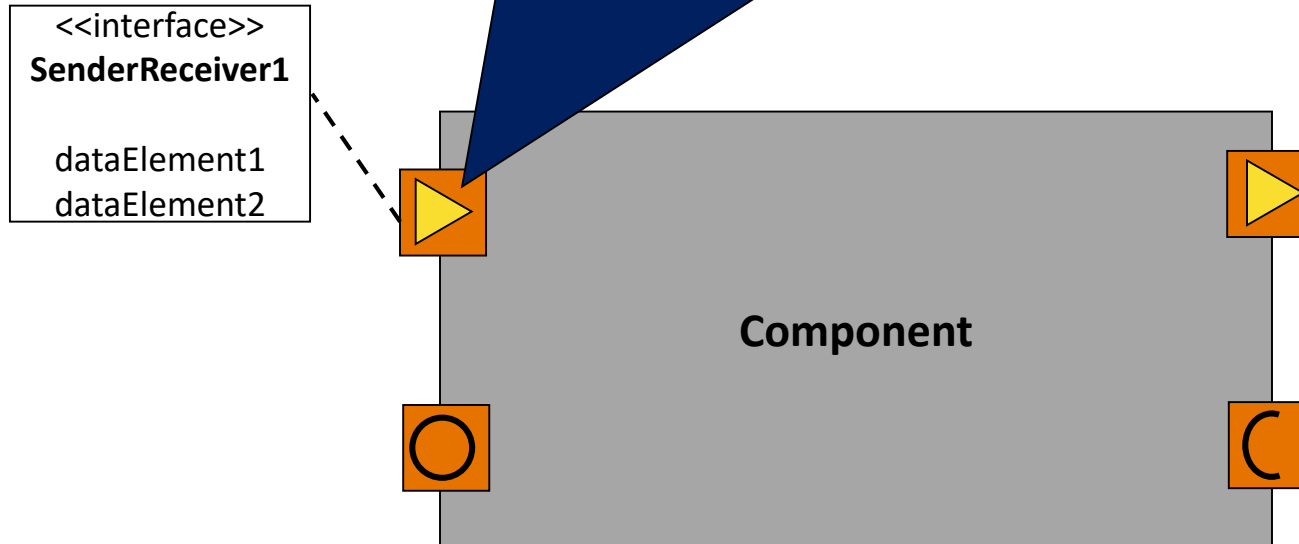
dataElement1
dataElement2



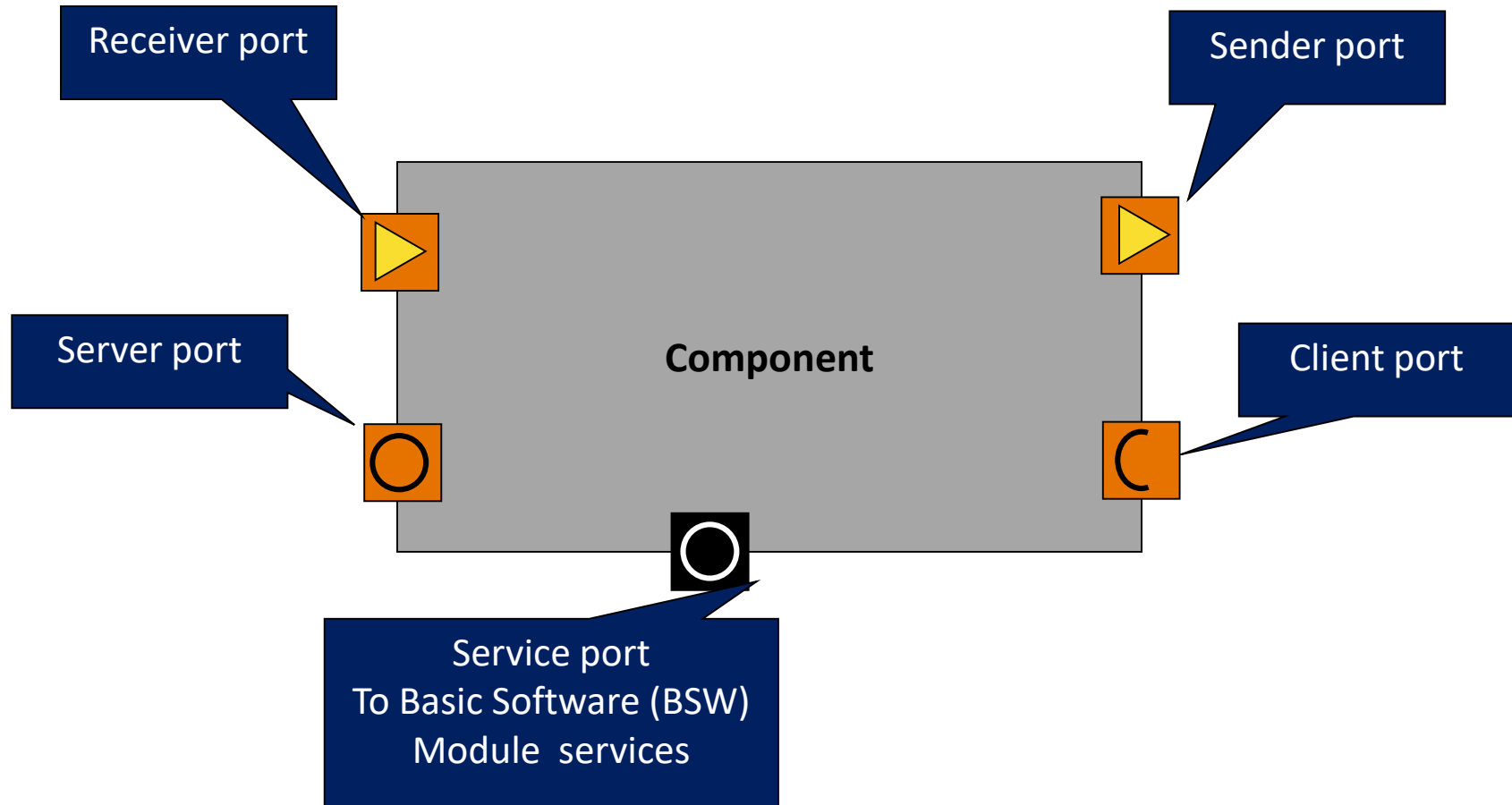
Component-based approach

Ports

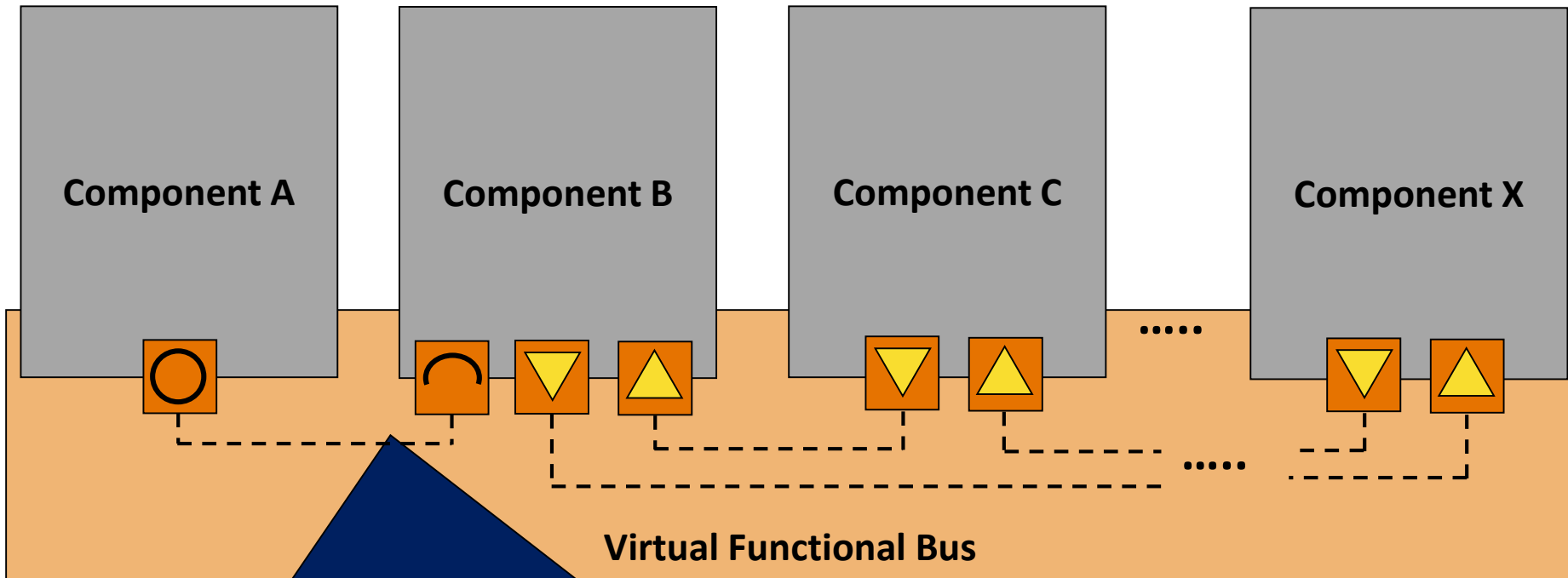
- The only interaction points between the component and its environment
- Are implementing *port interfaces*
 - sender receiver (message-based unidirectional communication)
 - client-server (remote procedure call)



Component-based approach – port notation



Component interconnection – the Virtual Functional Bus



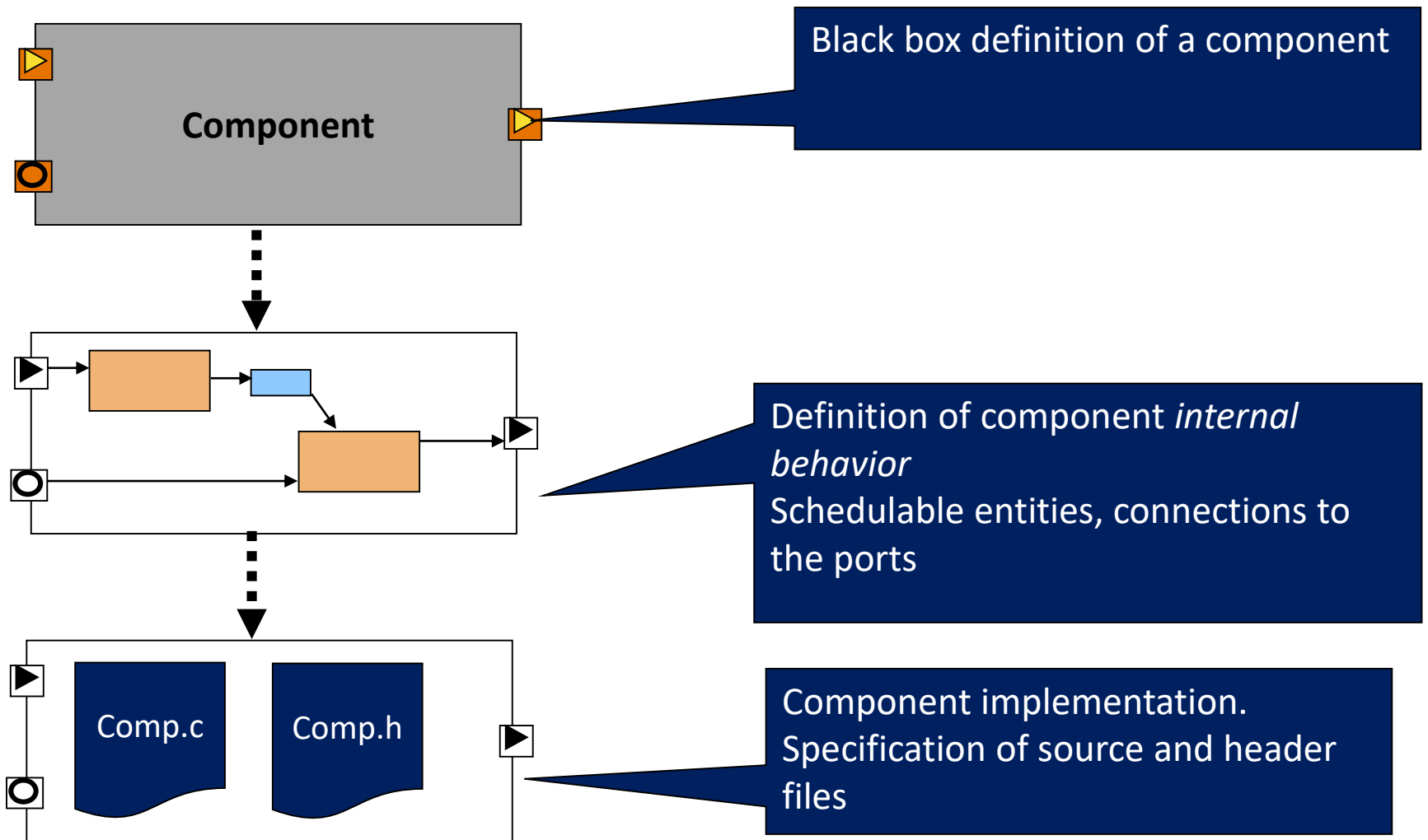
Virtual Functional Bus (VFB)

- Abstract interconnection layer
 - Implementation of data/control transport between components
 - No hardware/network dependency
 - Hides the details of the implementation
- Allows high-level integration *and simulation* of components
 - Before hardware architecture is chosen

Software Components

- On high-level, *atomic* components are black boxes
- Detailed design “looks into” these black boxes
- Main goals
 - Detail the behavior to get schedulable entities
 - Specify the semantics of port handling
 - Specify any service needs
 - Specify any RAM, nvRam needs

Refinement of a component



Component internal behavior

- Specification of the internals of an atomic SWC
- Schedulable elements
 - Called: runnable entities
- Connection of ports
 - Port semantics
 - Port API options
- Inter-runnable communication
- Runnable activation and events

Component internal behavior – runnable entities

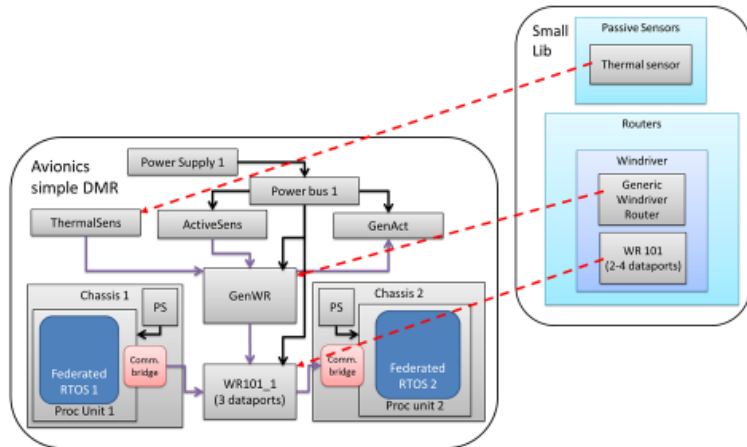
- Smallest code-fragments considered by RTE
- Subject to scheduling by the OS
- Abstraction of a schedulable function
- Communicates
 - Using the SWC ports
 - Using inter-runnable communication facilities
- Is activated by
 - An RTE event
 - Communication-related event
 - Timing event

Summary of AUTOSAR

- AUTOSAR defines
 - A component-oriented system design approach
 - Domain specific modeling language
 - A high level design process
 - Standard middleware (basic software) stack
 - Standard interfaces
 - Standard configuration descriptors
- AUTOSAR compliant ECU software
 - Includes several BSW and application components
 - RTE provides the integration (glue) between these
 - Configuration and glue code is mostly auto-generated

Summary

Defining the platform model II.

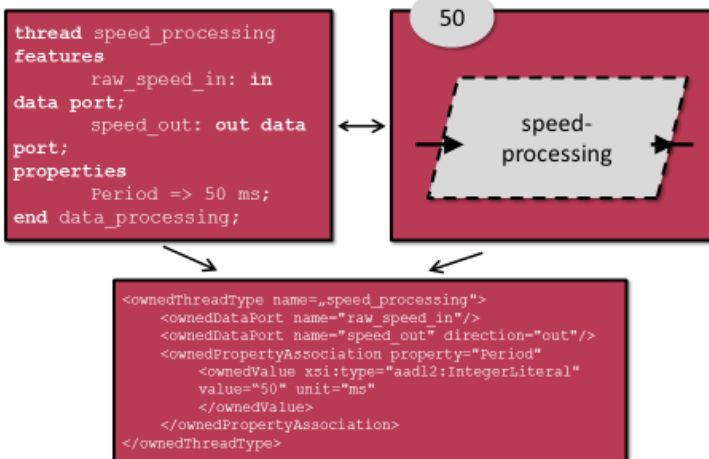


Allocation

- **Input:**
 - Functional models + platform model
- **Output:**
 - System Architecture
 - The *System Architecture* defines for each instance of a Function
 - *where and when to execute*
 - *when to communicate and on which bus*
 - *who can be addressed in communication*



AADL Representation Forms



High-level design process

