Rich Components

A coarse grained Approach to the Integration of Non-functional Constraints and MoCC in the design of Electronic Components in Automotive Applications

Werner Damm

OFFIS

Member Board of Directors

&

C.v.O. Universität Oldenburg

Chair of Safety Critical Embedded Systems

OFFIS

Structure of Presentation

- Motivation
- Rich Components
- Analyzing Compatibility
- The Safety Viewpoint
- Integrating multiple Models of Computation
- Conclusion

Joint work with
Eckard Böde[1], Bernhard Josko[1], Alexander Metzner[2],
Thomas Peikenkamp[1], Angelika Votintseva[1]

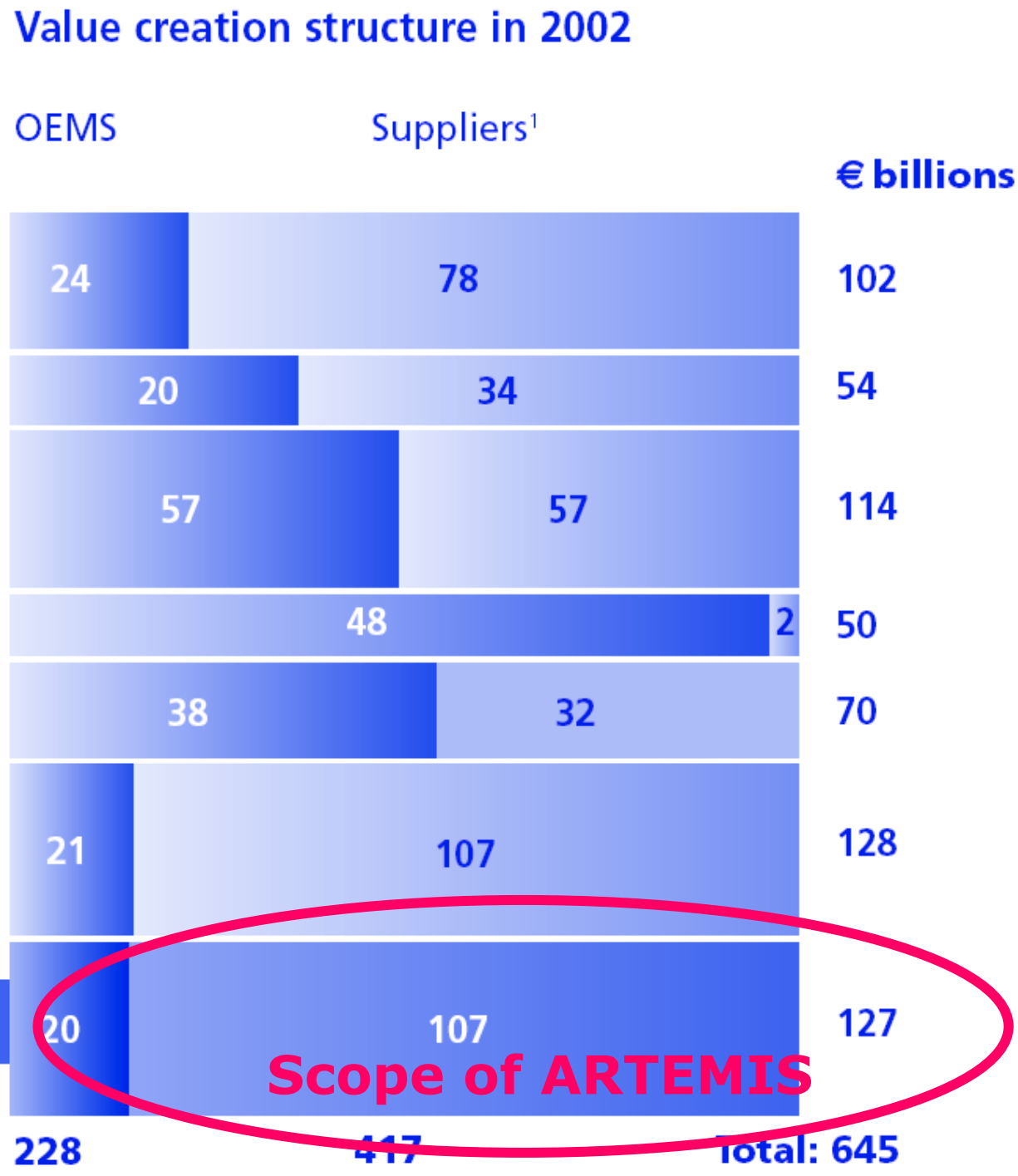
[1] OFFIS

and

[2] C.v.O. Universität Oldenburg

Motivation

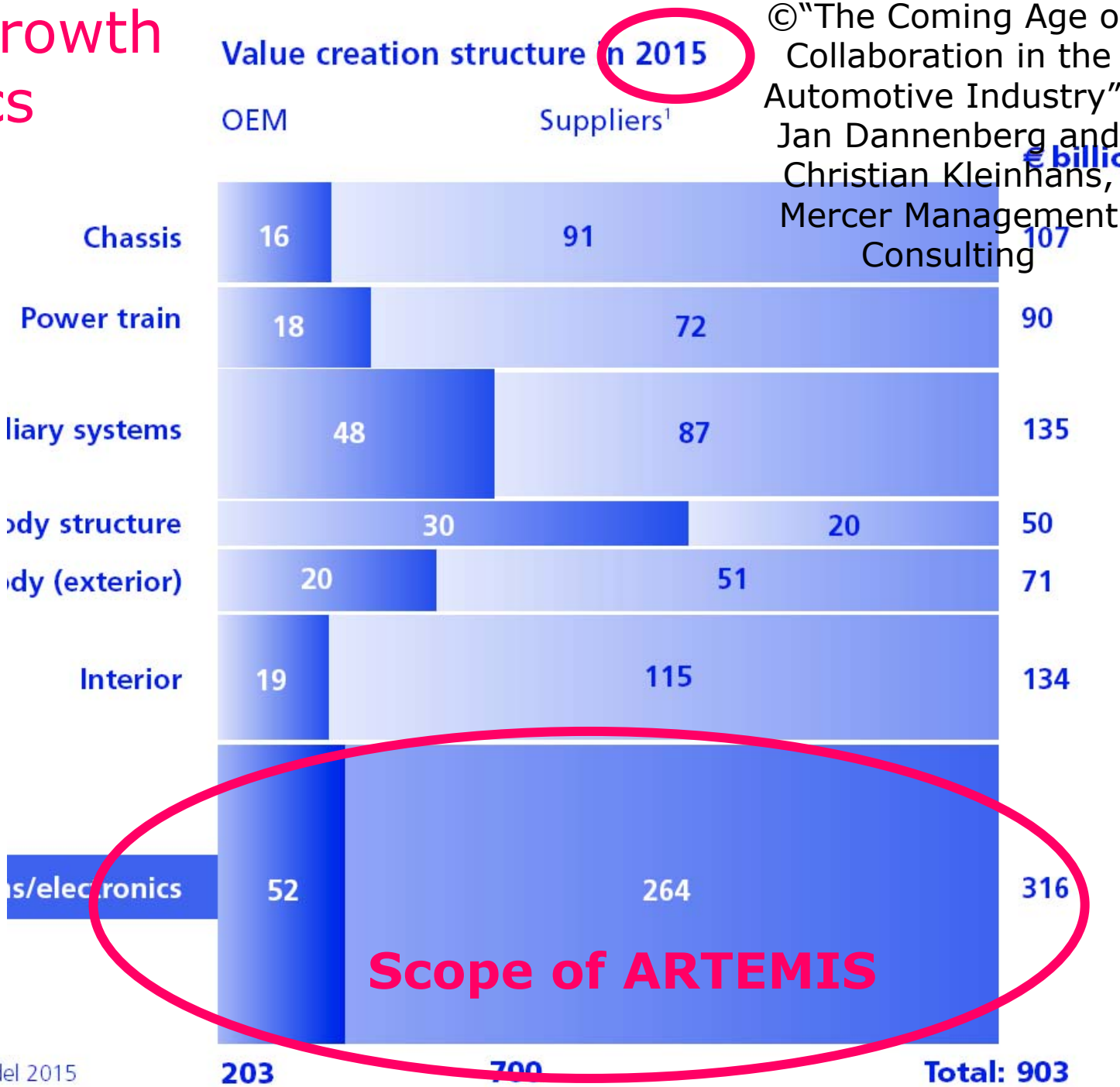
©"The Coming Age of Collaboration in the Automotive Industry", Jan Dannenberg and Christian Kleinhans, Mercer Management Consulting



Today Electronics Stand for 20%

Strongest Growth in Electronics

- Overall growth rate 40%
- **Electronic growth rate 250%**
- Increases **average** share of electronics to 35% from current 20%
- **More than 600 000 new jobs only in Automotive Electronics in Europe**



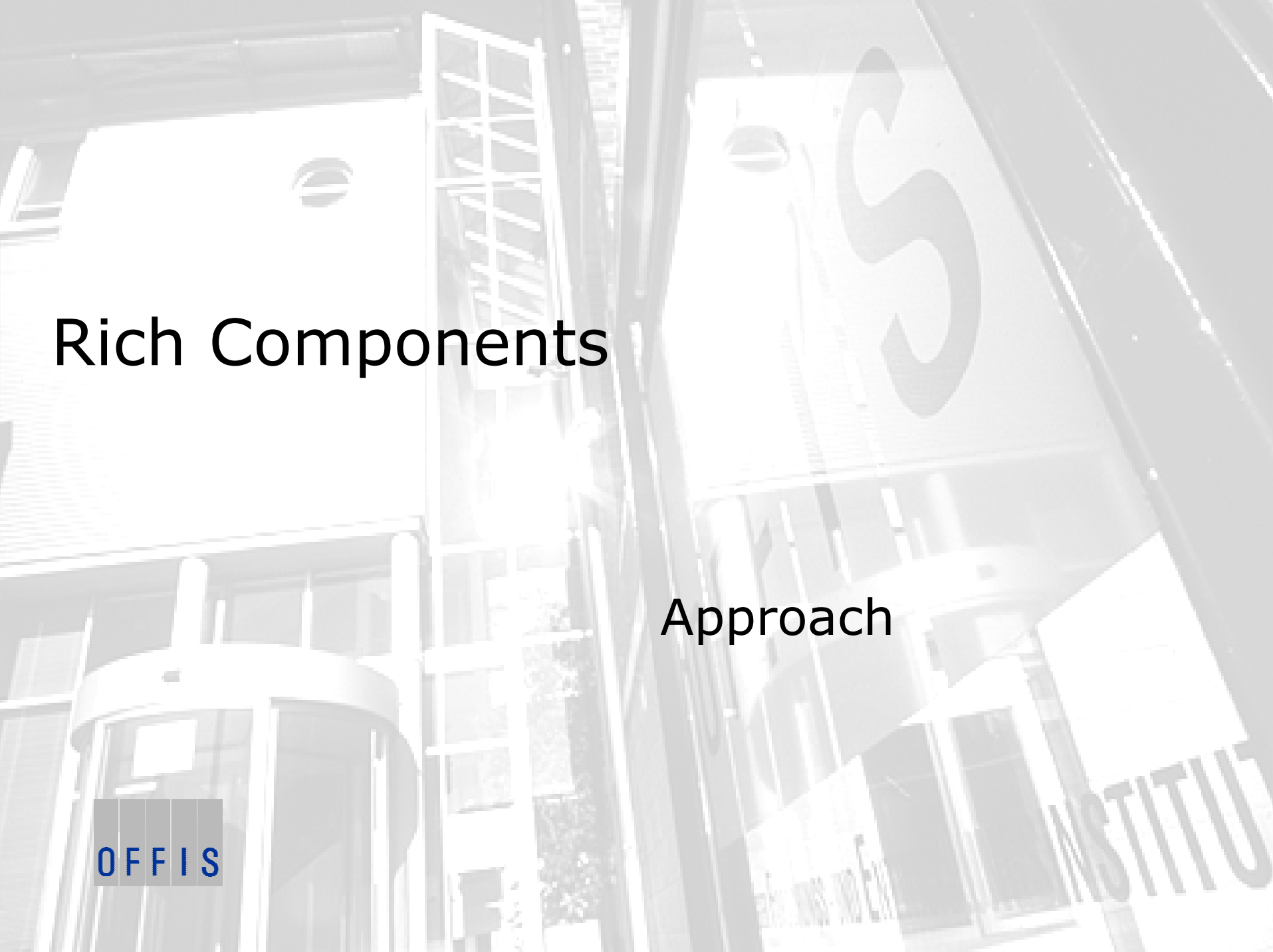
[1] Includes service providers

Source: Mercer Value Creation Model 2015

©"The Coming Age o Collaboration in the Automotive Industry" Jan Dannenberg and Christian Kleinhans, Mercer Management Consulting

Consequences

- OEMs will focus in-house development on branding components
- Shared libraries for non-branding components
- distributed functions running on multiple ECUs Developed by multiple sources
- Freedom in choosing boundary of in-house and external development
- New forms of suppliers, Software as product
- Cross-organizational optimization of electronic subsystems
- Re-Use at all levels
- Decouple function from implementation



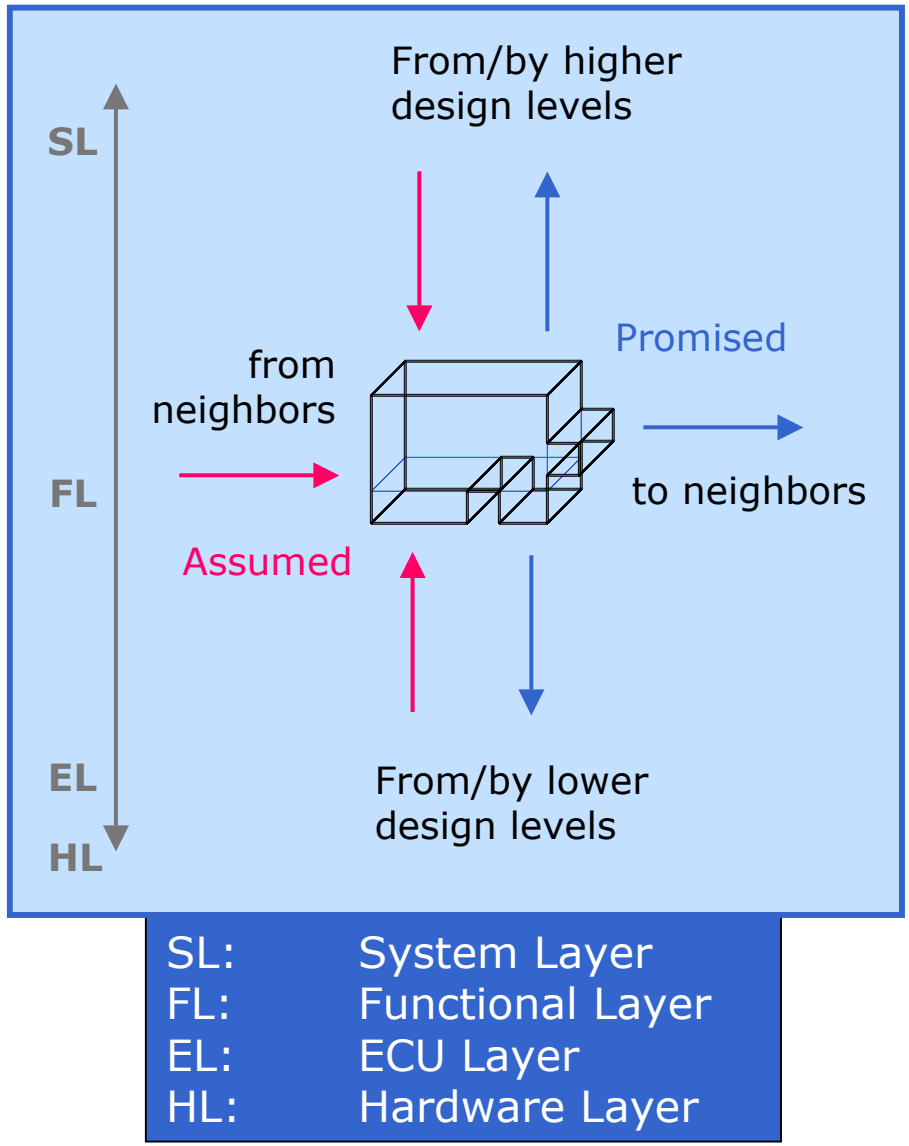
Rich Components

Approach

OFFIS

Rich Components

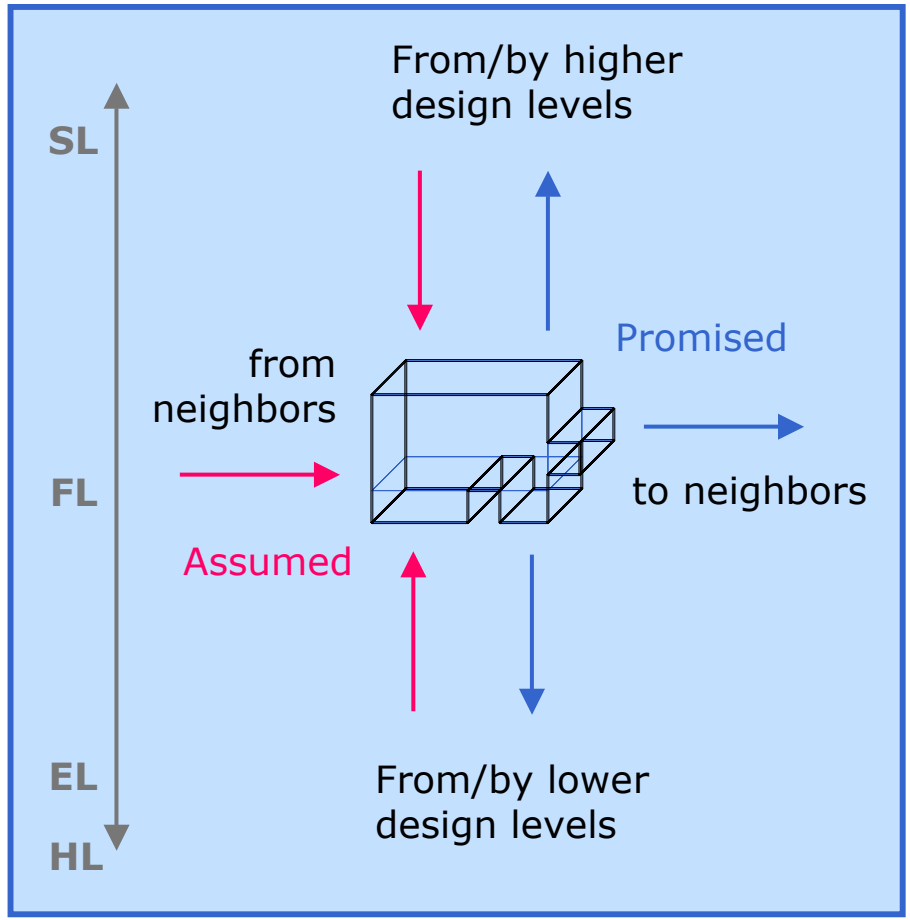
- A component : fully re-usable design artifact providing a well defined functionality
  - Application level functionality
    - "features" of application level functions – level of granularity determined by need to customize application level function
  - Middleware components
  - Hardware components
- "Rich"
  - Explicates all assumptions and/or dependencies on its design context
  - Such that assessment to functional and non-functional characteristics can be made without assessing component itself
- Component Characterization
  - For all viewpoints
    - Safety, Reliability, Real-Time, Power, Bandwidth, Memory consumption, behaviour, protocols



Rich Component Model

- Assumptions
  - reflect incomplete knowledge of actual design context
  - Determine boundary conditions on actual design context for each view-point under which component is promising its services
  - are decorated with confidence levels
- Promises
  - Are guaranteed if component is used in assumed design context
- Tradeoff
  - Accuracy of promises dependent on stringency of assumptions
  - High accuracy restricts implementation space
- Viewpoint specific models
  - Explicate dependency of promises on actual guarantees by design context

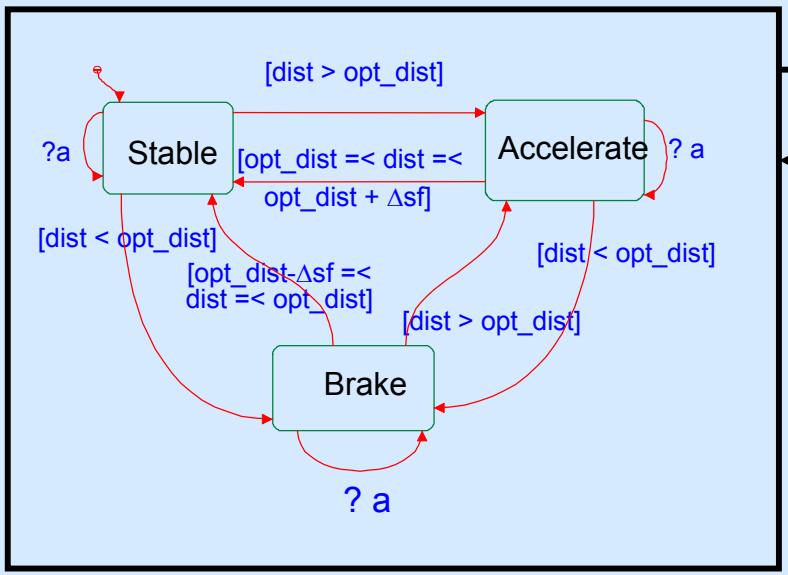
- Component Characterization
  - For all viewpoints
    - Safety, Reliability, Real-Time, Power, Bandwidth, Memory consumption, behaviour,

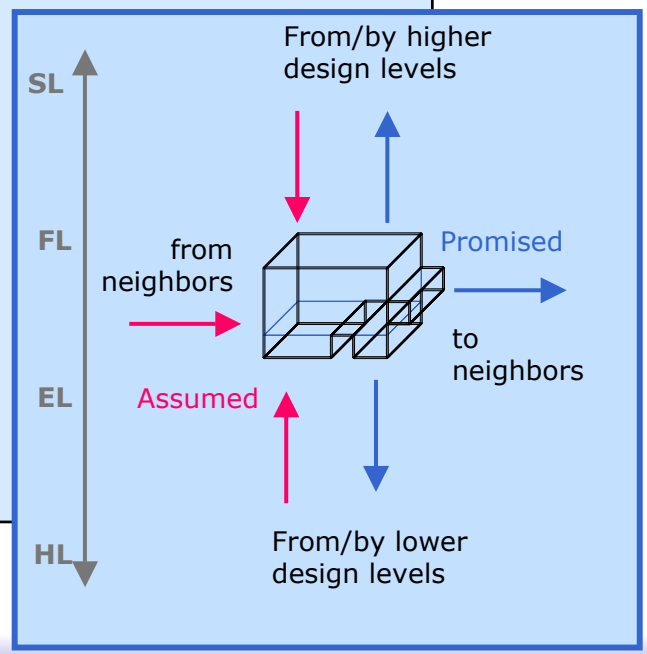


Rich Component Models (Functional View)

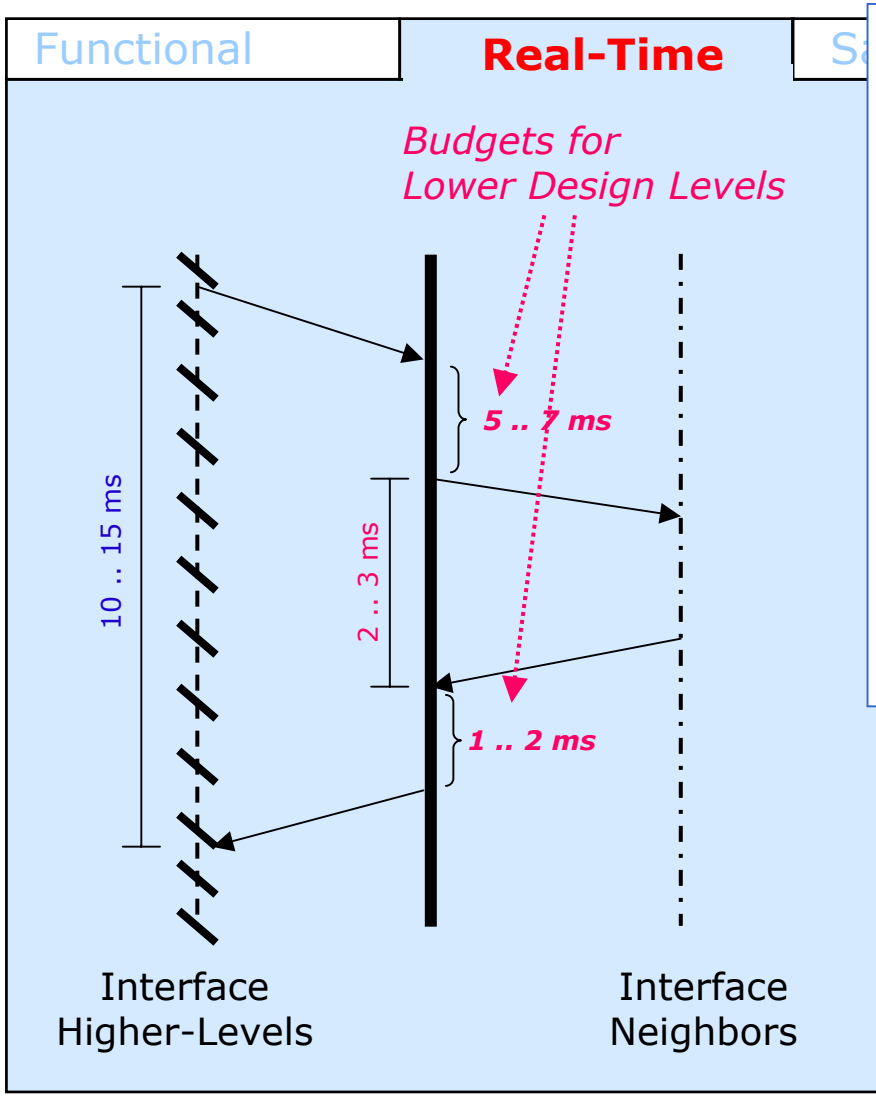
Functional | Real-Time | Safety | ...

- Horizontal assumption:
  Environment will provide the requested data
- Vertical assumption:
  The communication layer guarantees the transmission of every message





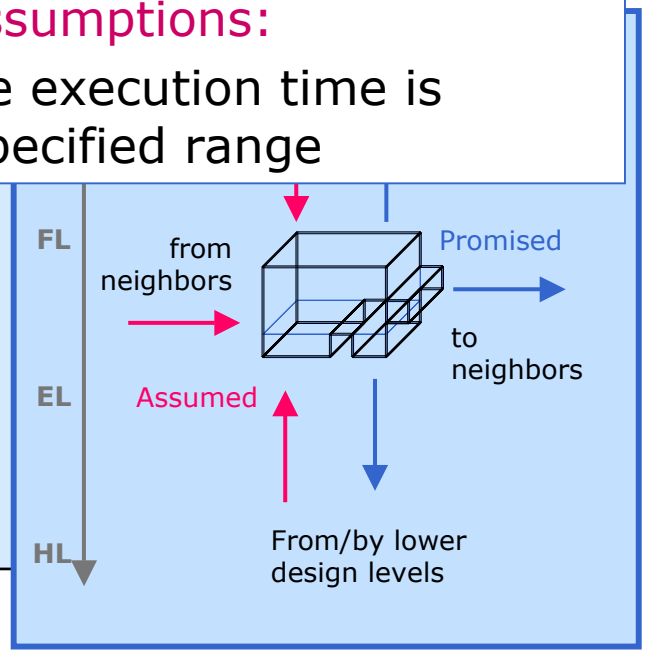
Rich Component Models (Real Time View)



- explicate dependency of promises on actual guarantees by design context for **real-time** properties using Live Sequence Charts
- Horizontal Assumptions:

  Requested information will be delivered within a specified time frame
- Vertical Assumptions:

  Worst case execution time is within a specified range



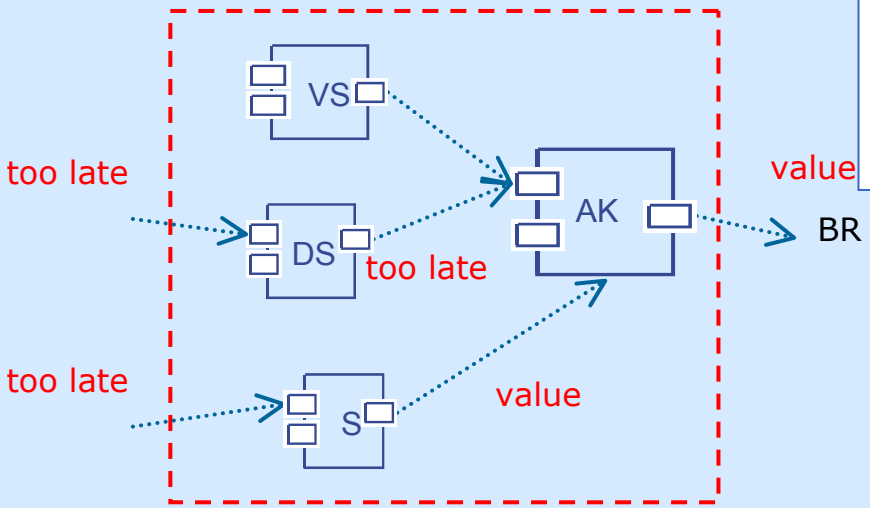
Rich Component Models (Safety View)

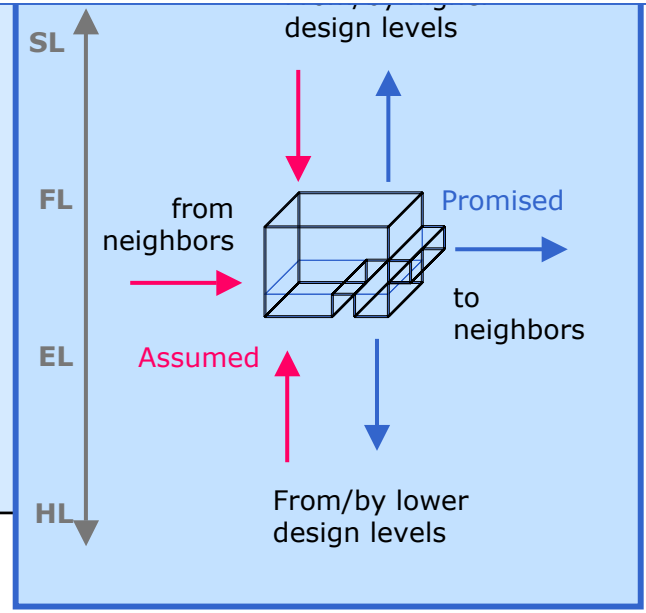
Functional | Real-Time | **Safety** | ...

- explicate propagation of failures in a conceptual model for a preliminary safety assessment
- Horizontal Assumptions:

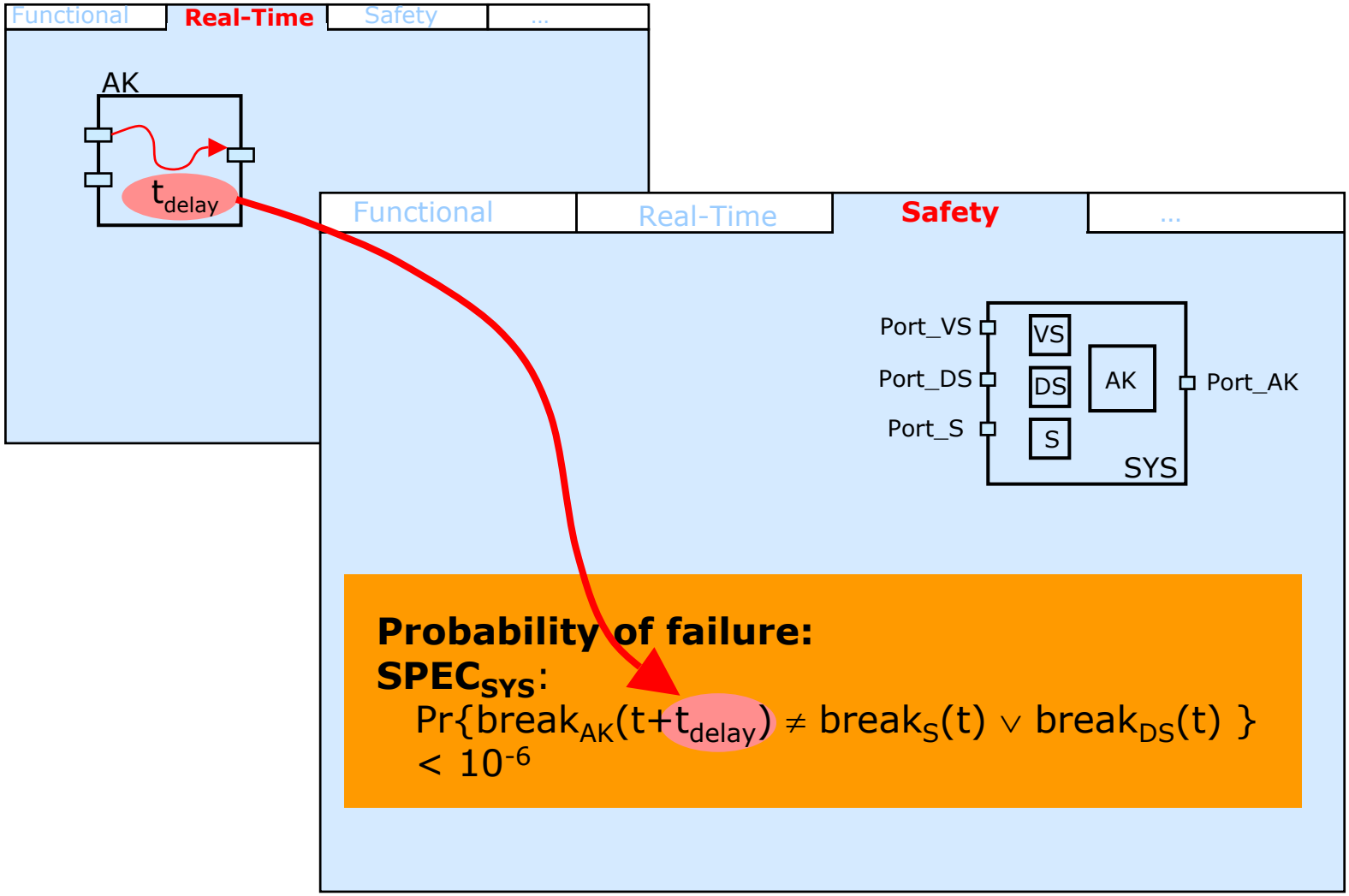
  Failure modes/rates for required information
- Vertical Assumptions:

  Failure rates for each failure mode



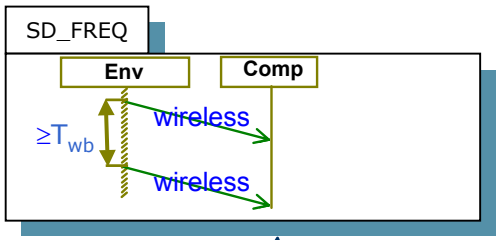


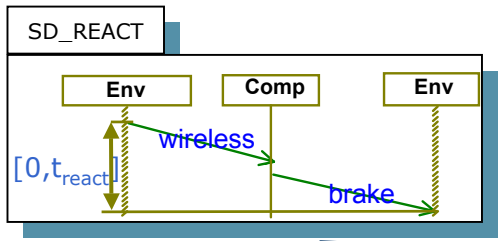
View Points Dependencies



Interfaces: Example

INTERFACE:
(FUNC2SYSTEM, **Direction:** up, **Data** (**Attributes:** T_{wb}:real:input, Δt:real:output, t_{react}:real:output, **Signals:** *wireless*:input, *brake*:output, **Operations:**),
Specifications: SPEC_FUNC_SYS)
SPEC_FUNC_SYS:
(**ViewPoint** real-time,
(to_system: (**Assumptions:**(SD_FREQ), **Promises:**(SD_REACT)))





Specifications

Attributes

Signals

Operations

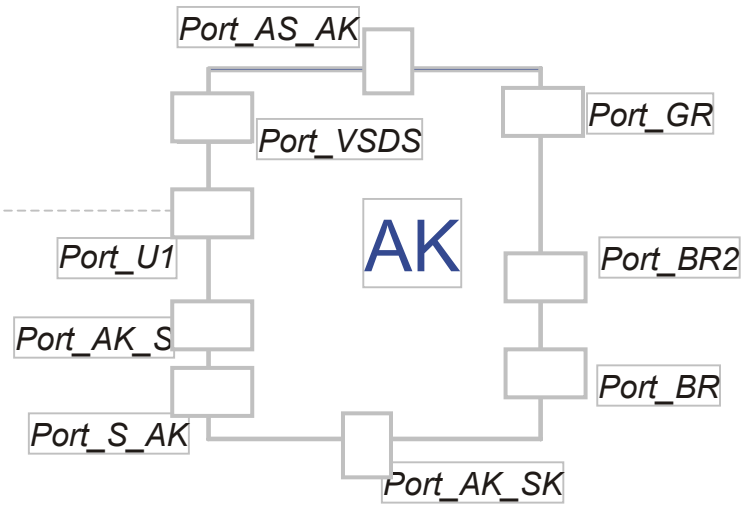
FUNC2SYSTEM

input: T_{wb}: real
output: Δt: real
t_{react}:real

input: wireless()
output: brake()

<<Vertical Up>>

<<type>>



Interfaces for Black-Box View: Relevant Elements

Interface

a group of services the component is ready to perform and services used by the component

Static Description – Data:

- *attributes*, the values of which are exchanged (read or written) by the component and its environment
- *signals* the component is ready to receive from or emit to its environment (asynchronous)
- *operation calls* the component is ready to accept or invoke from its environment (synchronous)

"Directions" considered

- *Interface* direction: horizontal or vertical, vertical = { up, down }
- *Data* directions: input or output

Dynamic Description = Specifications containing per viewpoint

- Declarative specifications
  - (temporal) logic
  - Sequence Diagrams

And/or

- Automata

suitably tailored to specific viewpoints

Property "kinds"

- Assumptions
- Promises

Black Box View: the benefit of being Rich

- Assume-guarantee style interface specifications completely defines substitutivity of components
  - Component Cnew can replace Component C iff

    …. Standard requirements on syntactic compatibility and

    for all viewpoints v

    for all interfaces i
    - assumptions of C jointly establish assumptions of Cnew
    - promises of Cnew jointly establish promises of C
- Application relevance
  - Defines space of allowed detailed specifications during development
  - Characterizes when component upgrades during product lifetime are permissable

Derived Notion: Input and Output Interfaces

An Interface is called *input* (*output*) if all its data elements a ∈ Data have directions input (output): DataDir(a)=input (DataDir(a)=output)

Input Interface contains:

- operation calls the component is ready to accept

- attributes, the values of which are read by the component

- signals the component is ready to receive

Automata Specifications = **expected** order, i.e. how the **environment**

-calls its operations

- sets (global) attribute values (read by the component)

- sends signals to the component

Output Interface contains:

- expected results of operations, the component invokes from the outside

- attributes, the values of which are written by the component

- signals the component emits into its environment

Automata Specifications = **guaranteed** order, i.e. how the **component** itself

- calls external operations

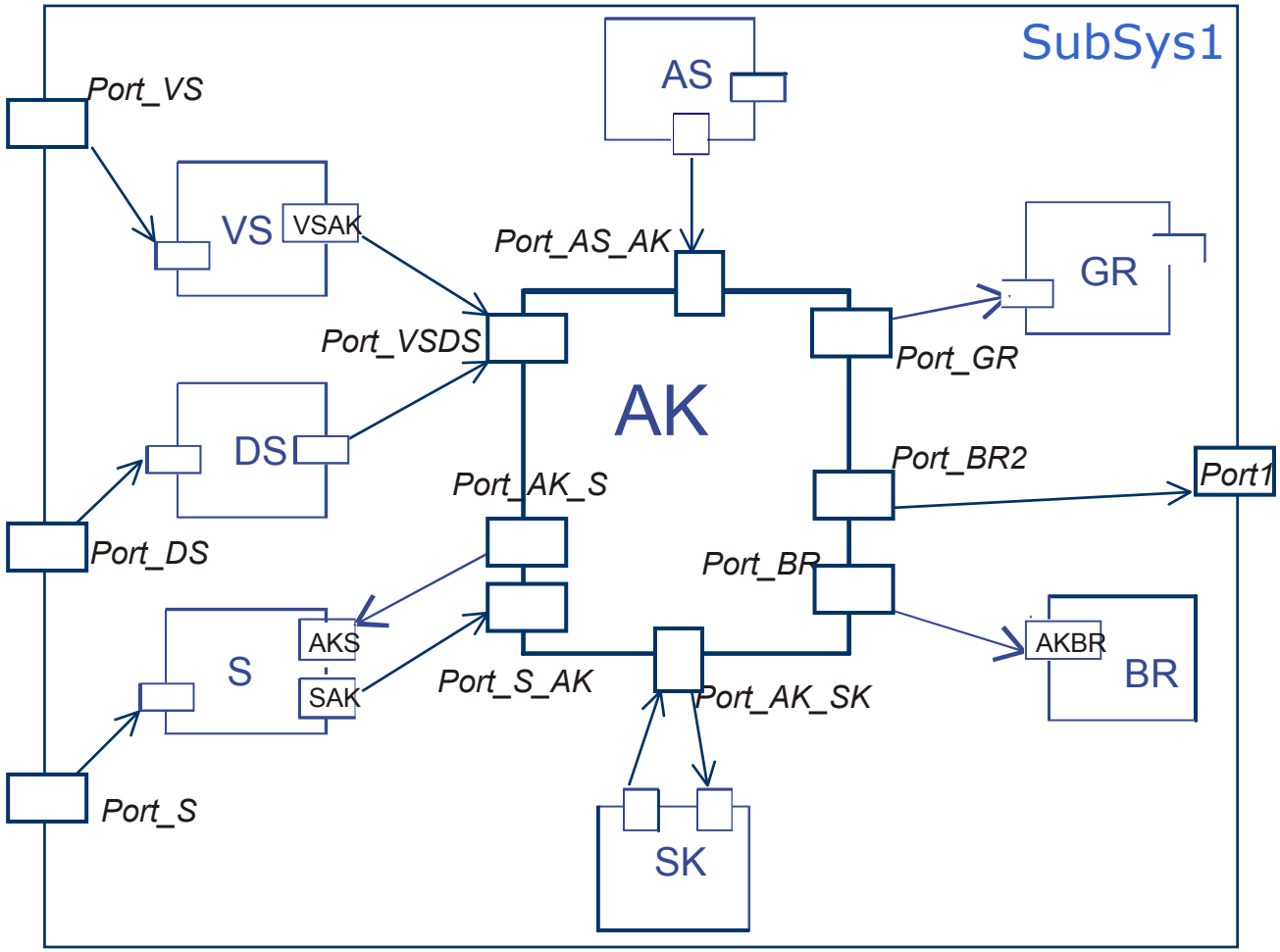
- sets (global) attribute values

- sends signals outside

Structural Specification: Gray Box View

Port_VSDS : InI_A1
Port_BR : OutI_A1
Port_S_AK : InI_A2
Port_AK_S : OutI_A2
. . .

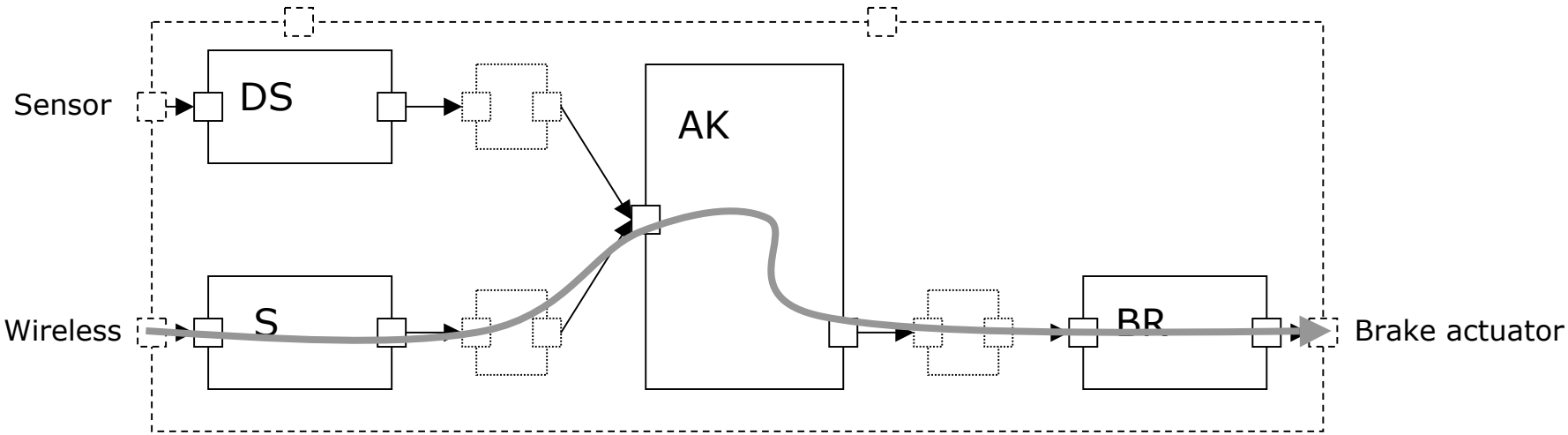
SAK : OutI_S
AKS : InI_S
VSAK : OutI_V
AKBR : InI_B
. . .



Compatible to UML 2.0 Composite Structure Diagrams
(for System level component)

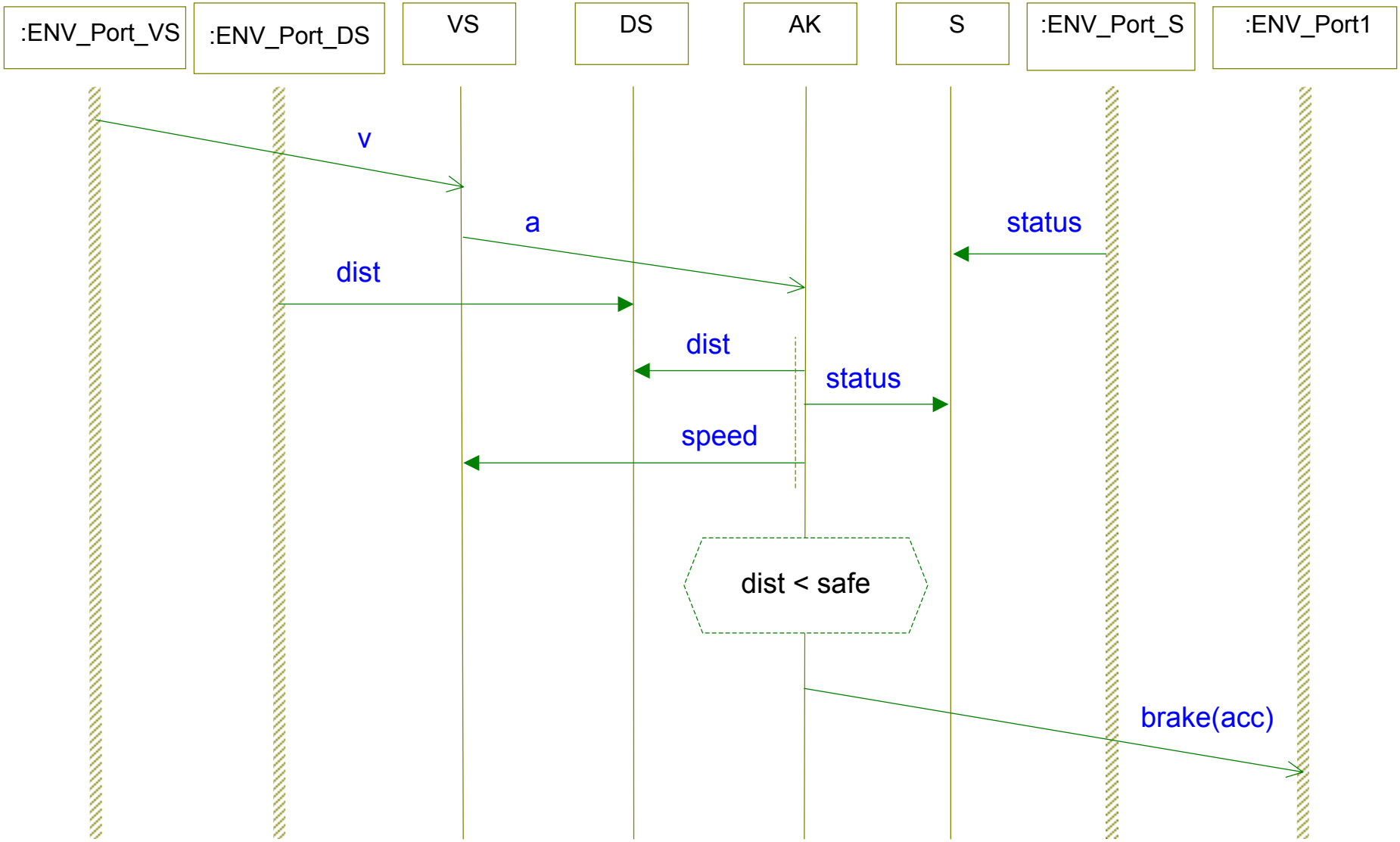
Gray-Box Specifications: Path-Formula for RT View

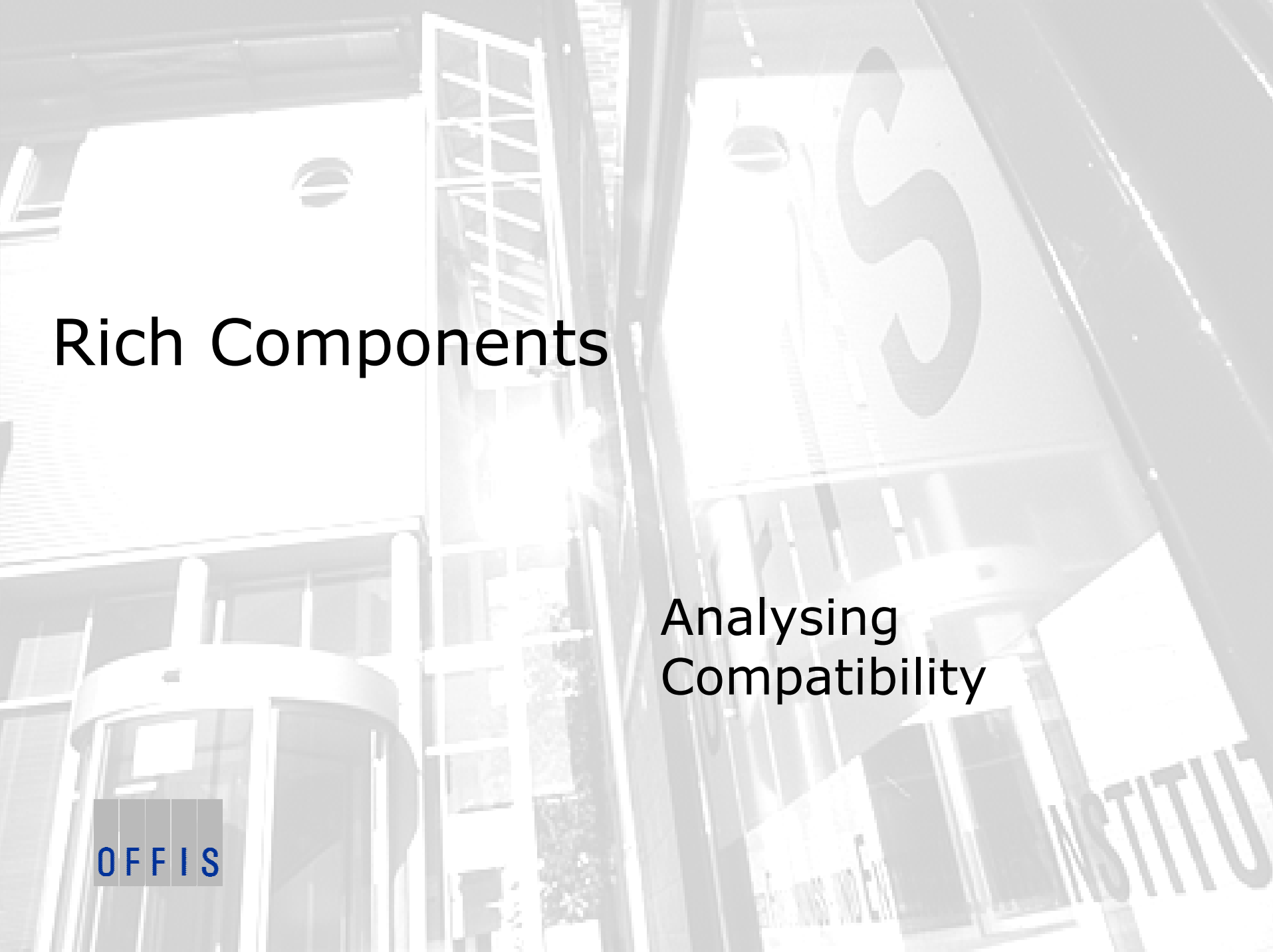
CompKind ::= **computation** | **connection** | **…**



$$\text{Grayboxspec: } t_{react} = r_{ext} + r_S + r(m_{S \to AK}) + \underrightarrow{r_{AK}} + r(m_{AK \to BR}) + r_{BR}$$

Gray-Box Specifications: Protocol Viewpoint



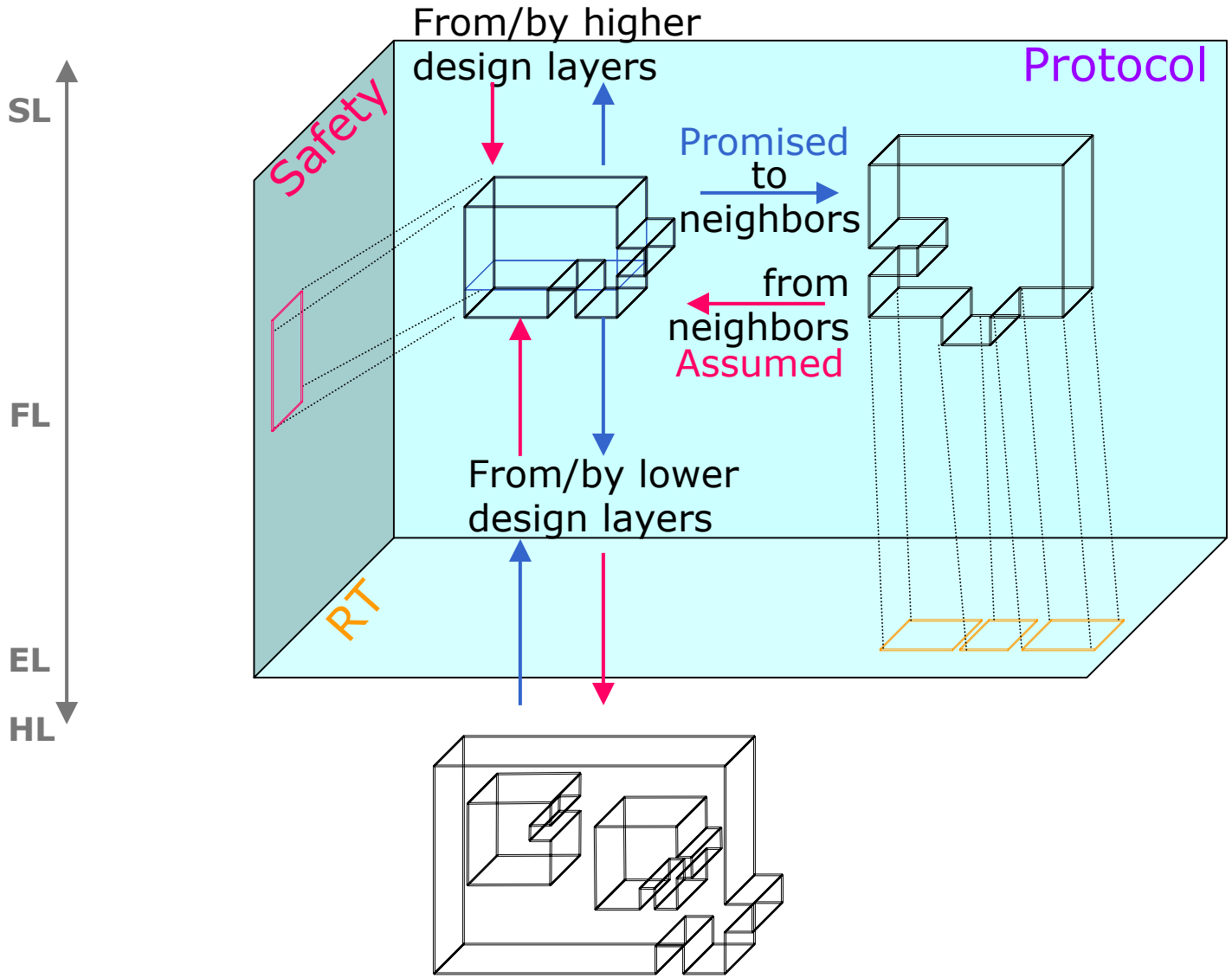


Rich Components

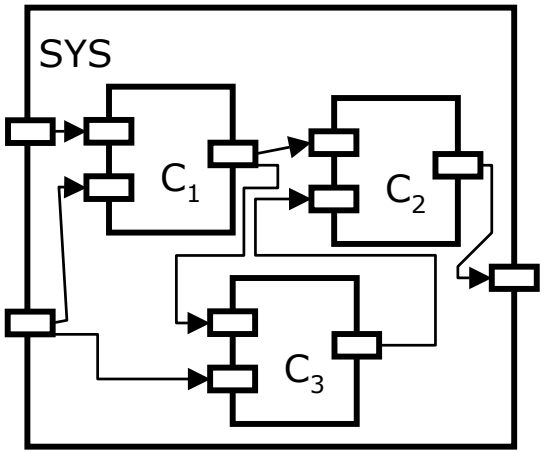
Analysing Compatibility

OFFIS

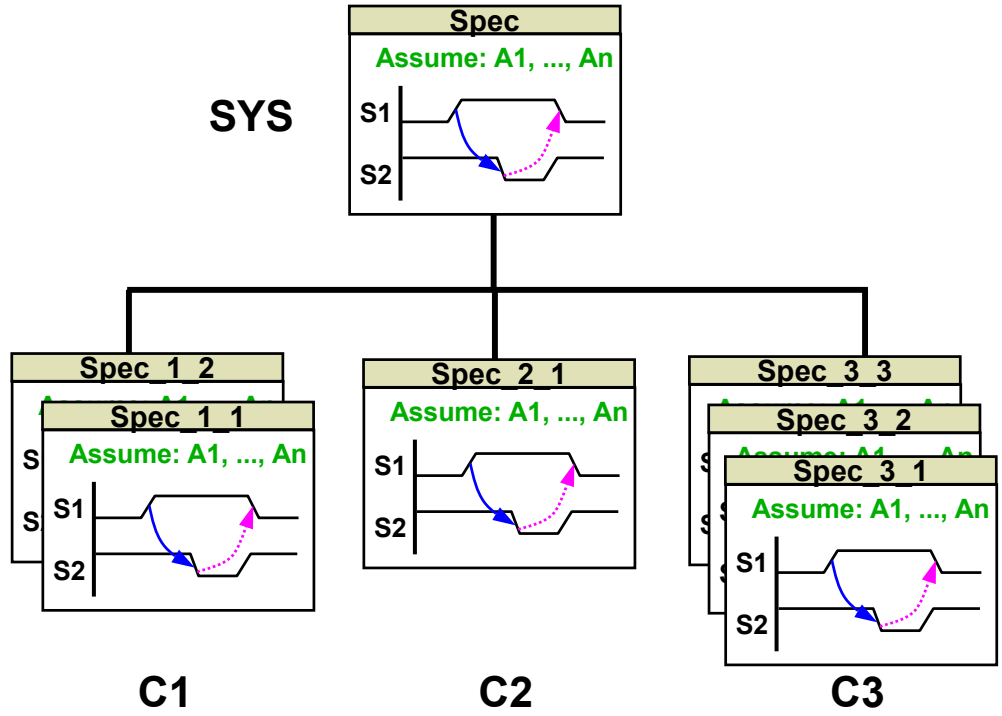
Analysing Compatibility



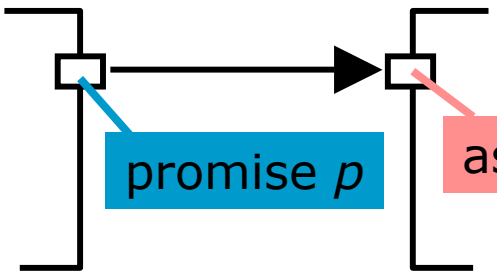
Horizontal Verification



Derive the global system specification *Spec* from the local component specifications:



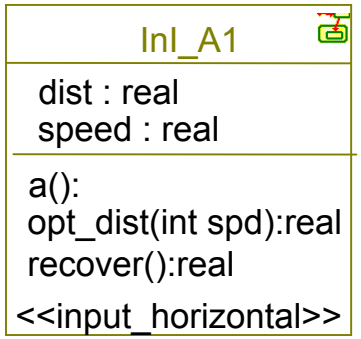
Furthermore: Interface specifications of connected port should match:



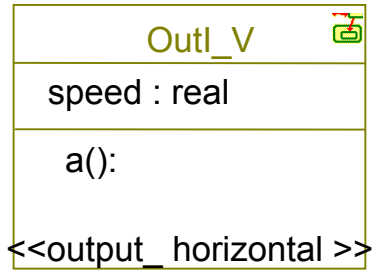
□ *p* implies *a*

Horizontal Compatibility Test

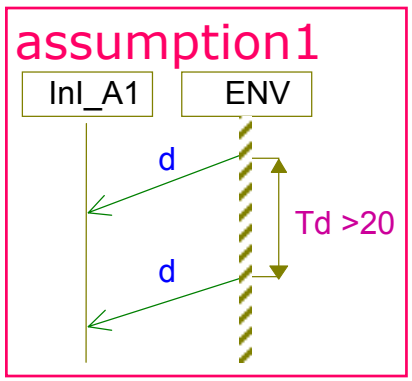
Input interface

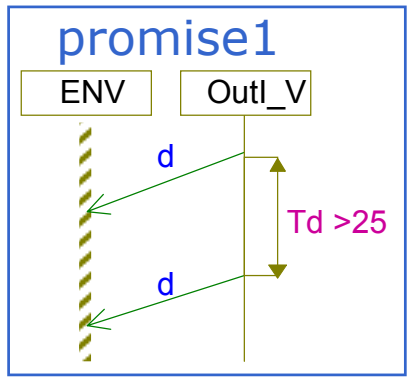


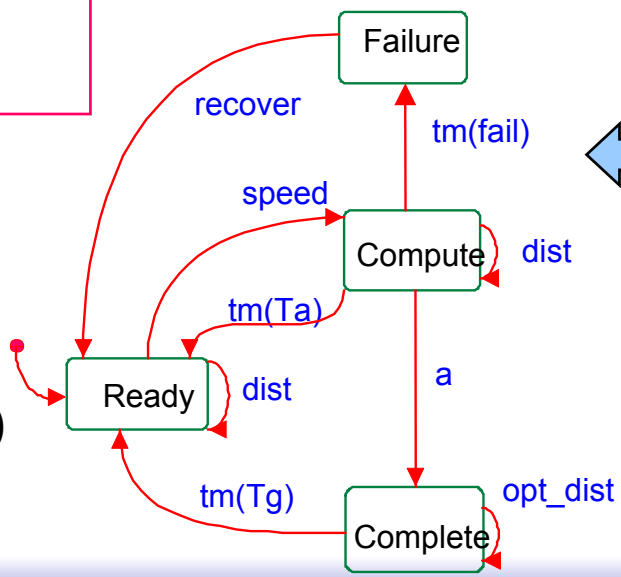
Output interface



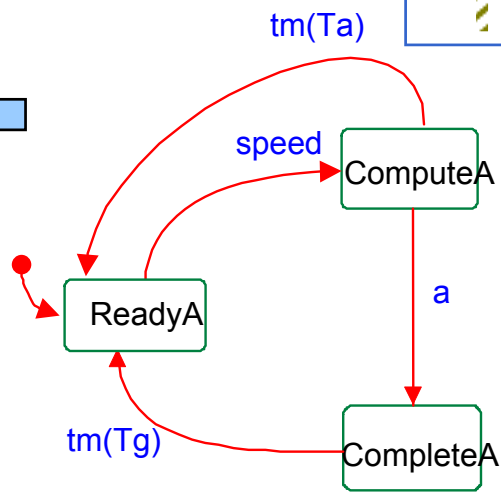
$$L(A(OutI\_V)) \subseteq L(A(InI\_A1))$$
$$\&$$
$$Promises(OutI\_V) \Rightarrow Assumptions(InI\_A1)$$





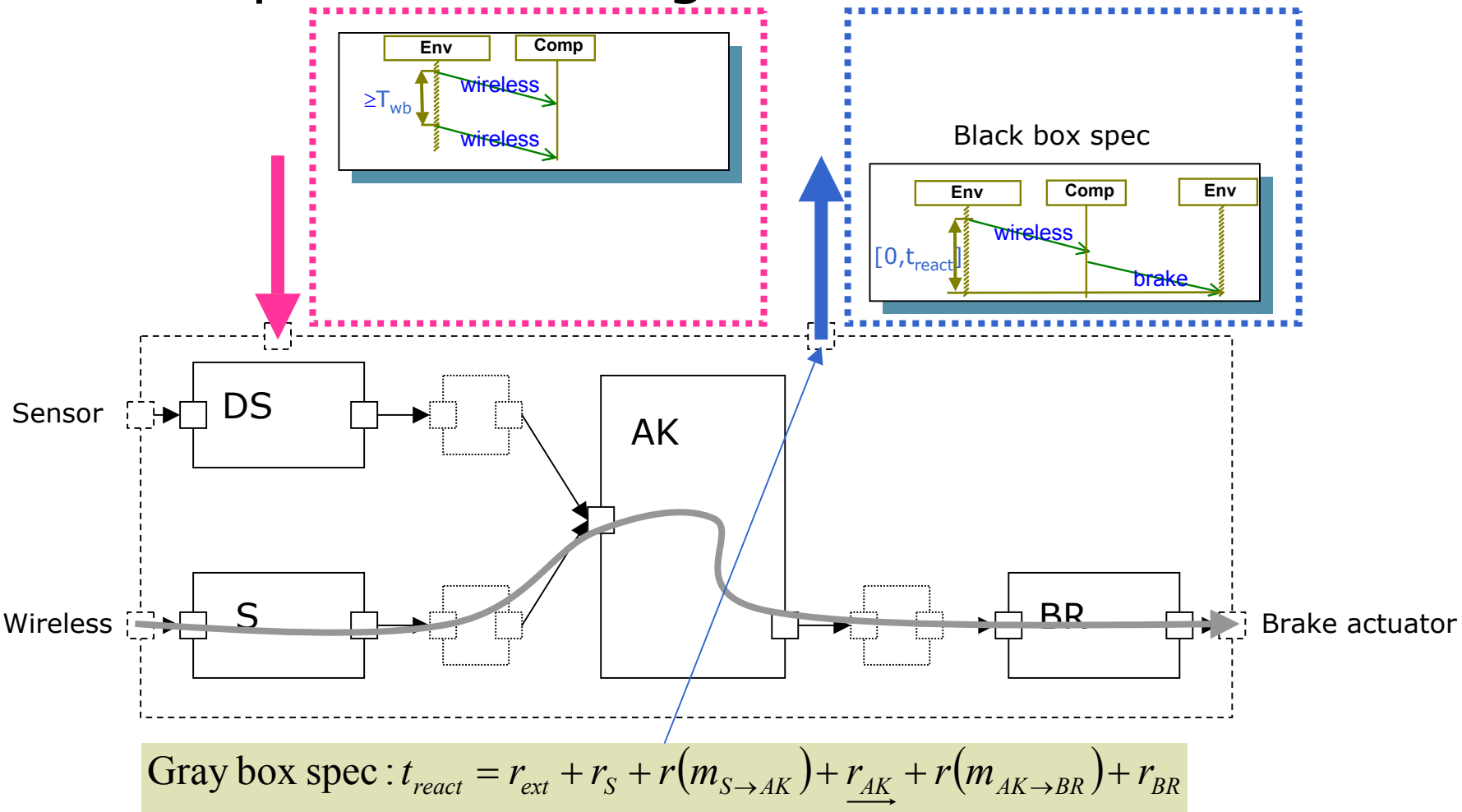


A(InI_A1)

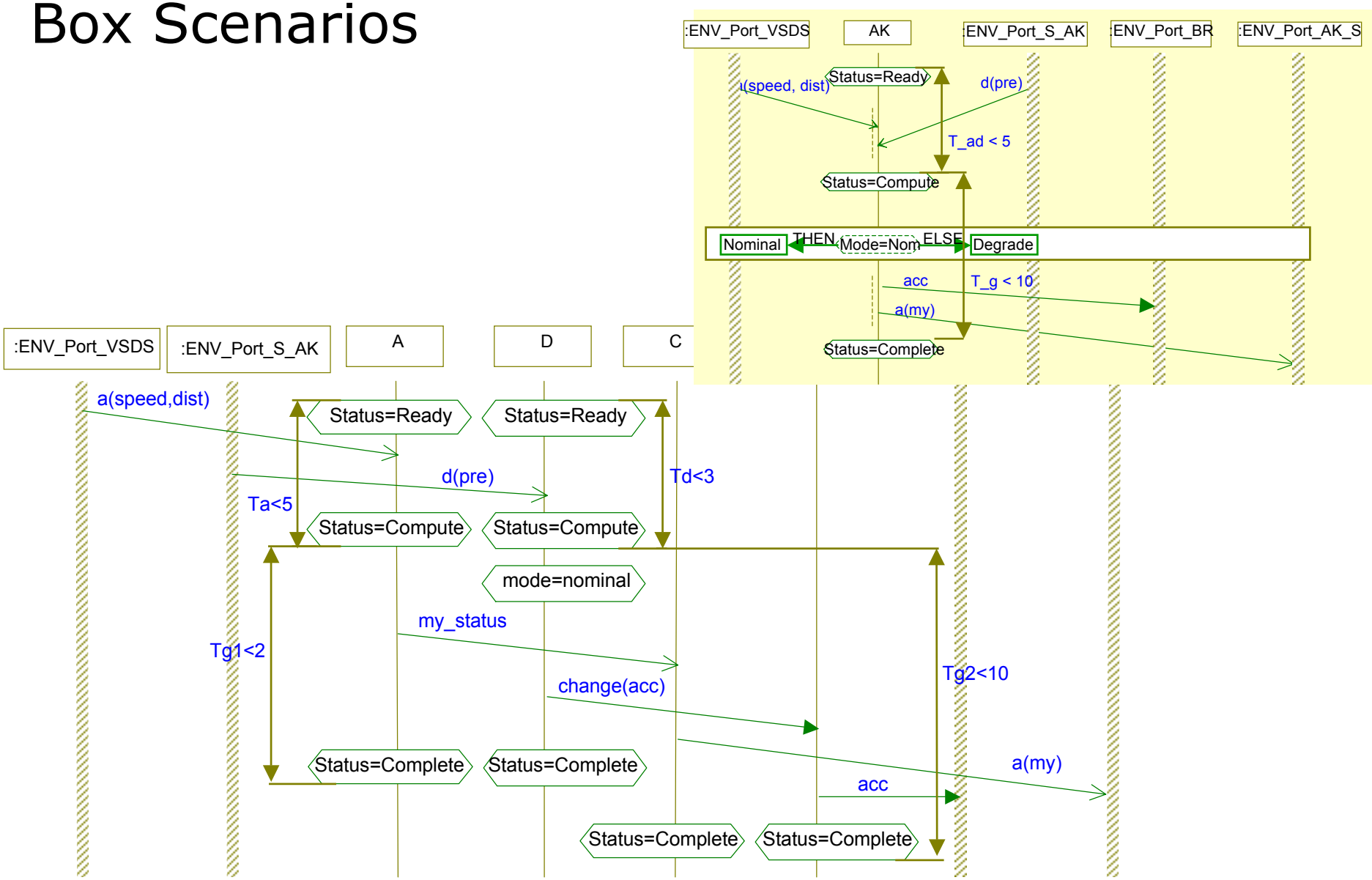


A(OutI_V)

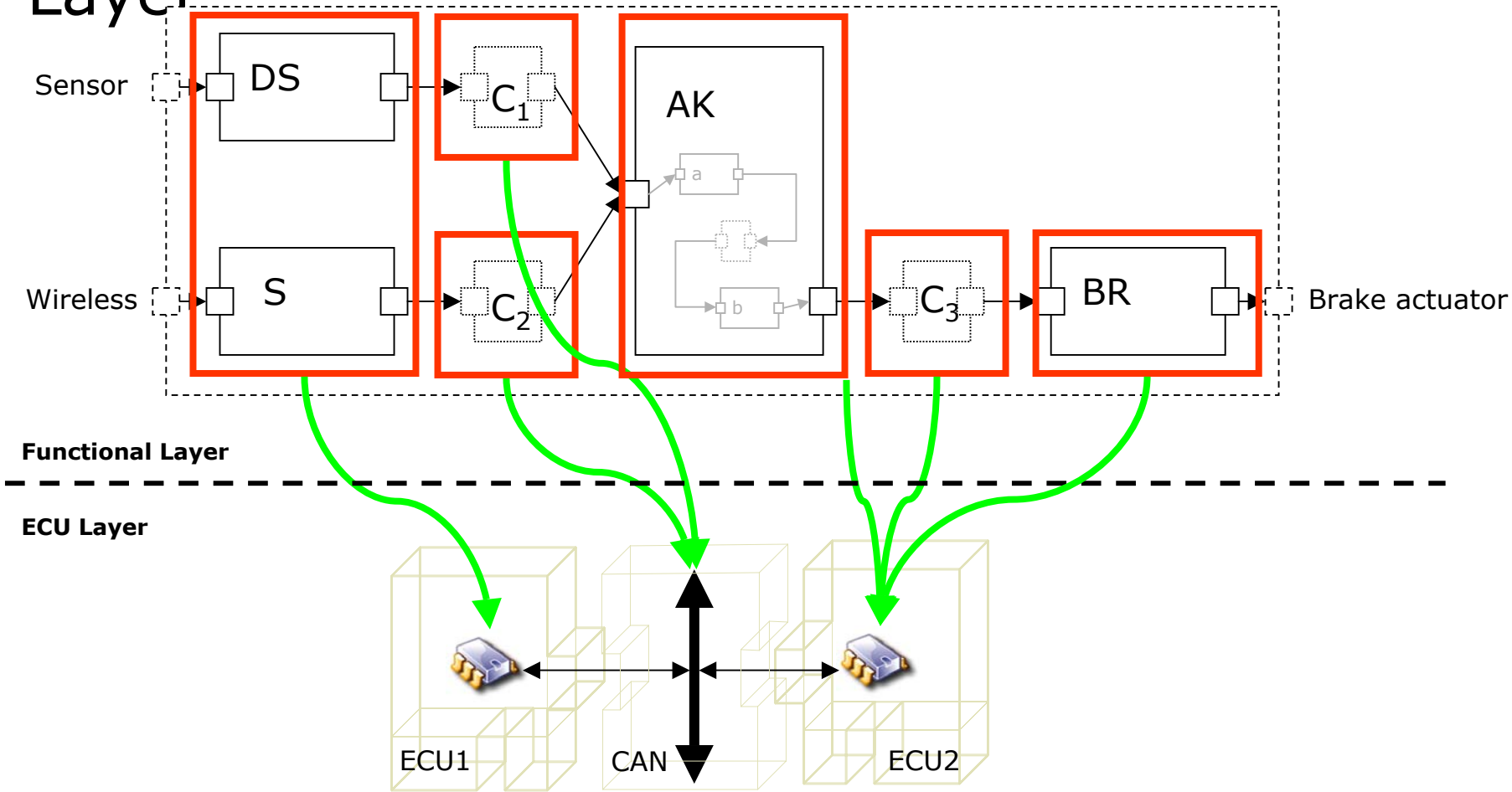
Using Horizontal Analysis to establish Black-Box Properties: timing



Establishing Black-Box Scenarios from Gray-Box Scenarios

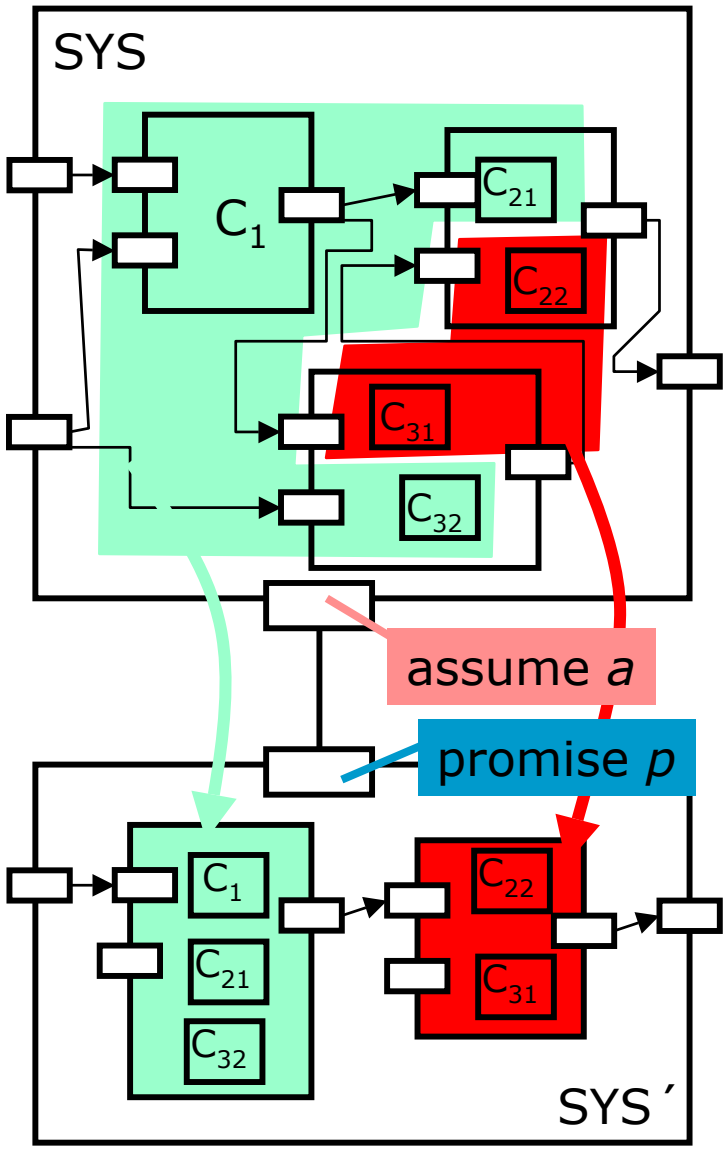


Mapping Example – Functional Layer → ECU Layer



- Map container of components to components of the lower layer

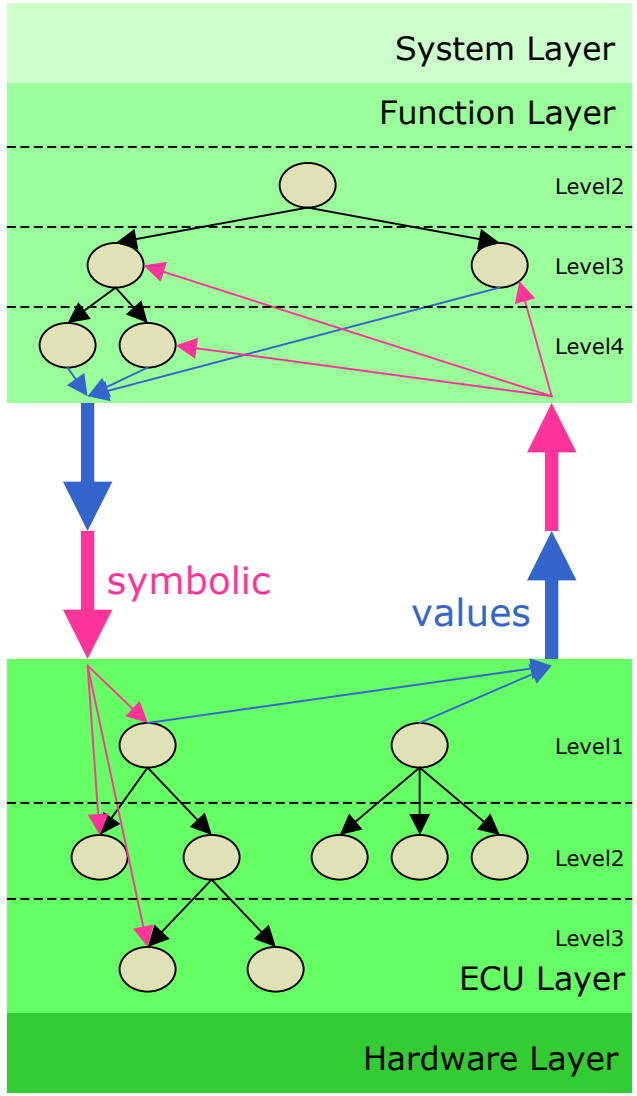
Vertical Verification – Mapping



- Establish the down assumptions of the upper layer *SYS* by the up promises of the lower layer *SYS´*
- Establish the up promises from the lower lower.

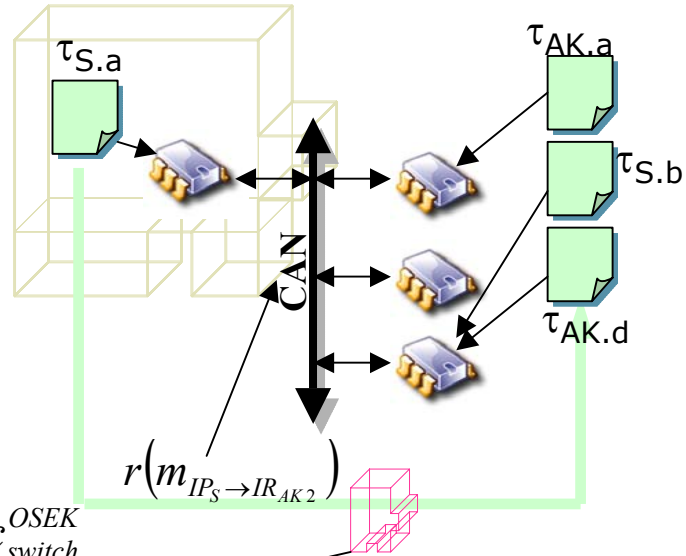
Vertical Layer Reasoning for Real-Time

- Derive values for symbolic time properties:
  - Vertical assumptions from higher layer are typically symbolic instances of time properties,
  - eg. calling periodicity, given in sequence diagrams
  - Vertical promises to higher layer propagate real values for symbolic assumptions,
  - eg. $WCET_{ECU1}(\tau_{AK.d})=17ms$
- Promises **down** are collected from the leaf node's black box specifications /ports
- Promises **up** are collected from the top level black box specifications/ports
- Vertical composition/verification is the mapping of container plus compliance checks on mapped ports



Sample Reasoning for Real-Time

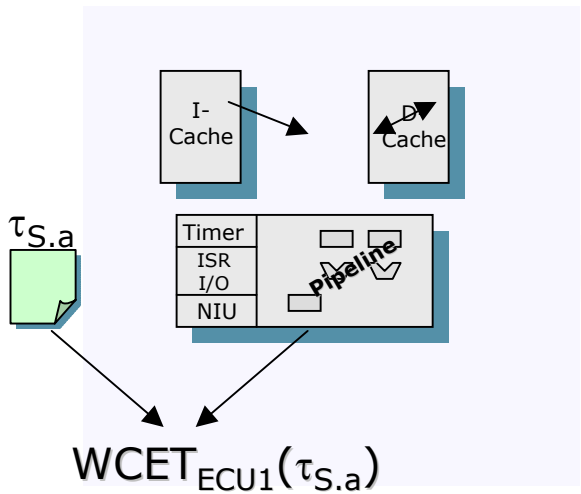
- **ECU Layer**: Response time calculation for
  - Task networks mapped on ECUs with given RTOS
  - Message transmission mapped on bus systems with given bus specification (transport and higher layers)
  - Take architectural topology into account



$$w_{AK.d} = c_{AK.d} + write(\text{var}) + \tilde{r}^{AK}_{meth(c)} + \sum_{\tau_j \mapsto ECU_1 : p_j < p_{\tau(d)}} \left\lceil \frac{w_{AK.d} + J_j}{T_j} \right\rceil \cdot c^{OSEK}_{switch}$$

$$+ \sum_{\tau_j \mapsto ECU_1 : p_j > p_{\tau(d)}} \left\lceil \frac{w_{AK.d} + J_j}{T_j} \right\rceil \cdot \left(c_j + 2c^{OSEK}_{switch}\right) + \rho_{CAN} + J_{AK.d}$$

- **Hardware Layer**: Time consumption of programs and messages
  - Worst/Best Case Execution Time analysis on Processors, memory hierarchies, etc.
  - Message transmission latency on the base of physical layers of bus protocols



Rich Component Model

Conclusion

OFFIS

Conclusion

- The Rich Component Model provides a comprehensive framework for the development of electronic components of automotive applications
- Meets key industrial needs
  - Seperation of function and implementation
  - Cost Reduction
  - Speculative design Processes
  - Conservative extension of AutoSAR approach
- Addresses key scientific challenges
  - Early assessment of non-functional constrains
  - Precise characterization of substitutivity