EE249 - Fall 2012
Lecture 17: Contracts and Compositional Methods for System Design

Alberto Sangiovanni-Vincentelli
Pierluigi Nuzzo

EE249 Fall'12: where we have been

1. Introduction	Design complexity, examples of embedded and cyber-physical systems, traditional design flows, Platform-Based Design, design capture and entry
2. Functional modeling, analysis and simulation	Overview of models of computation. Finite State Machines, Process Networks, Data Flow, Petri Nets, Synchronous Reactive, Hybrid Systems. Tagged Signal Model. Simulation of heterogeneous systems. Compositional methods and Contract-based Design.
3. Architecture and performance abstraction	Definition of architecture, examples. Real time operating systems, scheduling of computation and communication.
4. Mapping	Definition of mapping and synthesis. Code generation from Simulink and SysML models. Design Space Exploration and Metropolis. Mapping and Contracts. (Communication Synthesis and Communication-Based Design.)
5. Verification	Validation vs. Simulation. Formal methods. Horizontal and Vertical Contracts. Interface automata and assume-guarantee reasoning.
6. Applications	Automotive: car architecture, communication standards (OSEK/AUTOSAR), scheduling and timing analysis. Building automation. Aircraft electric power system.

EE249 Fall'12: where we are going

1. Introduction	Design complexity, examples of embedded and cyber-physical systems, traditional design flows, Platform-Based Design, design capture and entry
2. Functional modeling, analysis and simulation	Overview of models of computation. Finite State Machines, Process Networks, Data Flow, Petri Nets, Synchronous Reactive, Hybrid Systems. Tagged Signal Model. Simulation of heterogeneous systems. **Compositional methods and Contract-based Design.**
3. Architecture and performance abstraction	Definition of architecture, examples. Real time operating systems, scheduling of computation and communication.
4. Mapping	Definition of mapping and synthesis. Code generation from Simulink and SysML models. Design Space Exploration and Metropolis. **Mapping and Contracts.** (Communication Synthesis and Communication-Based Design.)
5. Verification	Validation vs. Simulation. Formal methods. **Horizontal and Vertical Contracts. Interface automata and assume-guarantee reasoning.**
6. Applications	Automotive: car architecture, communication standards (OSEK/AUTOSAR), scheduling and timing analysis. Building automation. **Aircraft electric power system.**

Contract-Based Design: an all-encompassing framework

The key to Platform Based Design

- Components
- Composition rules
- Refinement rules
- Abstraction rules

Contracts

Outline: contracts and compositional methods for system design

- Where and why using contracts?
- Introduction to contracts
- Mathematical meta-theory of contracts
- Overview of concrete contract theories
- Application examples

Outline: contracts and compositional methods for system design

- Where and why using contracts?
  - Structuring top-level specifications
  - Sub-contracting and reusing
  - Deployment and mapping
- Introduction to contracts
- Mathematical meta-theory of contracts
- Overview of concrete contract theories
- Application examples

Structuring top-level specifications

- A desirable objective at each step of the design
- Requirement documents are structured into viewpoints (sometimes referred to as *chapters, aspects, sub-documents,…* depending on the company or sector)
  - Behavioral viewpoint: the functions are specified
  - Timing viewpoint: timing budgets are allocated to activities
  - Safety viewpoint: fault propagation, reliability
  - …
- Viewpoints are generally developed by different teams using different skills, frameworks and tools

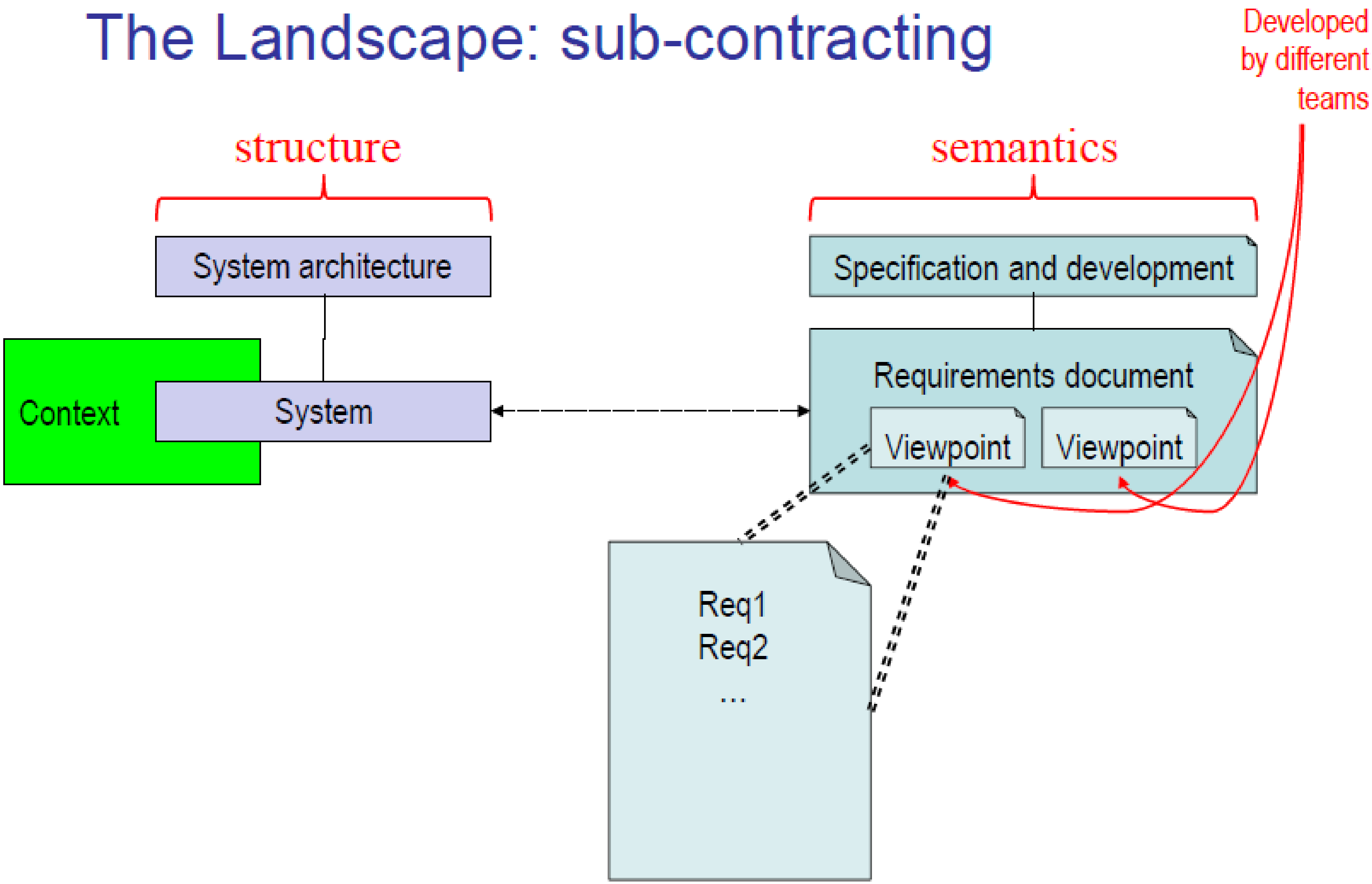
Example: a monitoring system

Software for monitoring the physical system

- Requirements must specify
  - Decision logic
  - Sampling period
  - When to activate/inhibit monitoring
  - Time lag before fault confirmation
  - Fault identifier
  - Fault compensation
  - Returns to the cockpit
  - *Very diverse in nature*
- Requirements must also specify *at a higher specification level* what is the objective of the detection, which fault to detect and isolate
  decision logic ≈ implementation

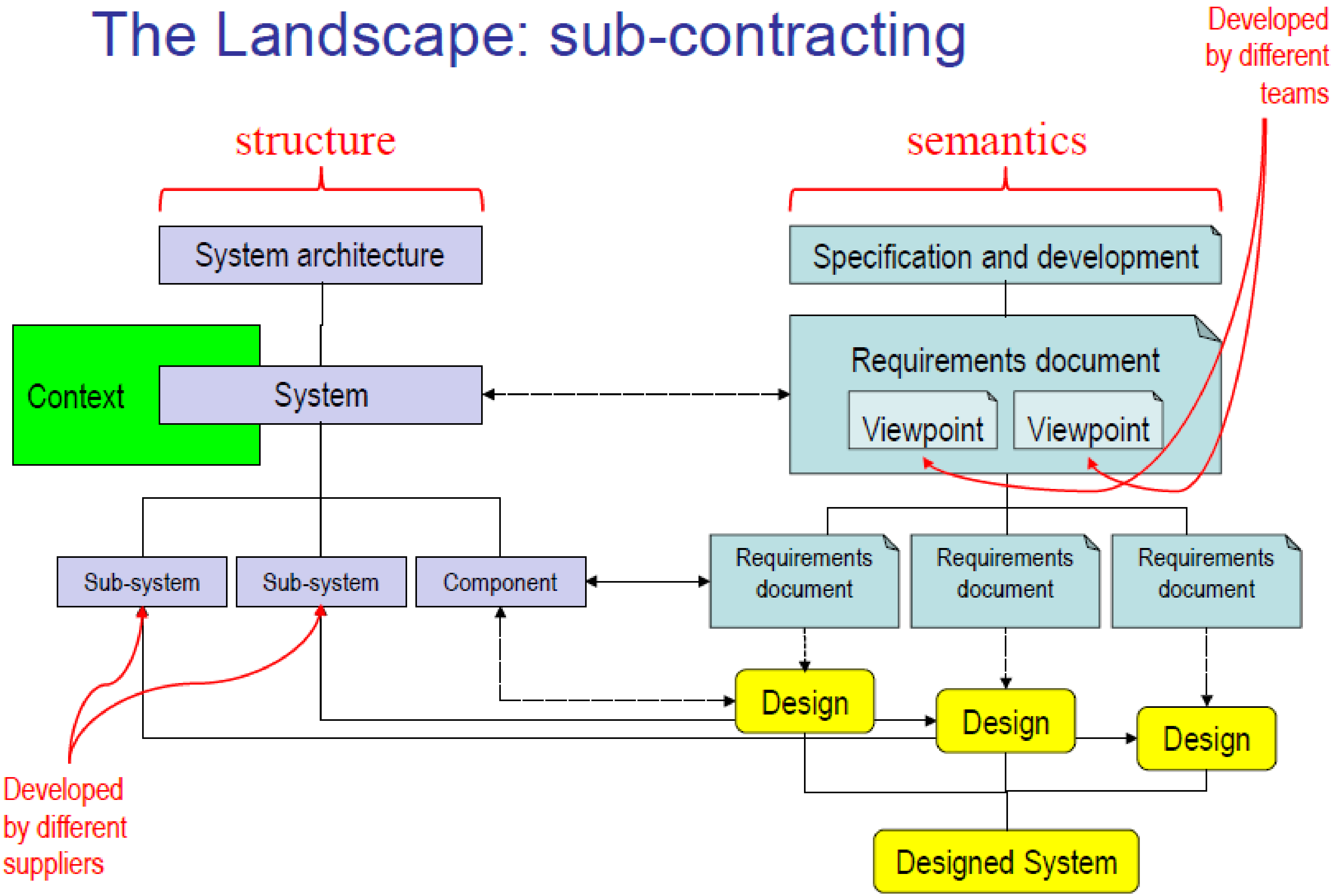
Source: A. Benveniste

The Landscape: sub-contracting



Source: A. Benveniste

The Landscape: sub-contracting



Source: A. Benveniste

Need for requirement engineering

Traceability
- Requirements attached to "everything" via hyperlinks (tests, V&V, integration)

Ontology
- Terms used for entities should be precise and unambiguous (important)
- Terms used for entities should be structured (ontology)

Identifying responsibilities
- Some requirements express guarantees; other express assumptions

Partitioning and sub-contracting
- Allocating requirements to suppliers, budgeting

Modular handling of viewpoints & subsystems
- Separation of concerns: function, QoS, safety/reliability…

Fundamental properties (certification bodies)
- Completeness, Consistency, Compatibility, … (from INCOSE)

Source: A. Benveniste

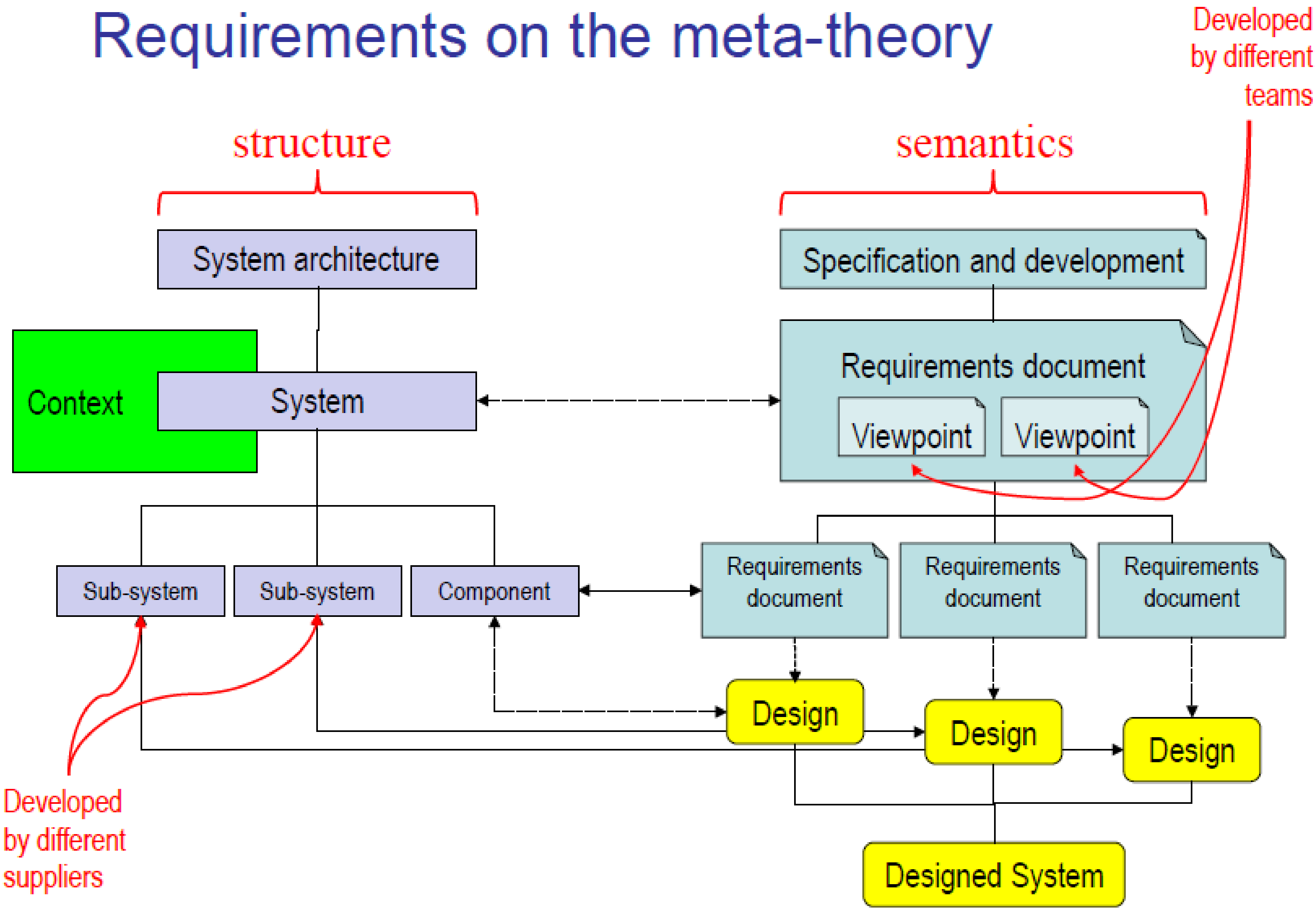
Need for requirement engineering

Overall, requirements engineering

- has been considered by the AI community (ontologies)
- has been considered by the Software Engineering community as part of MDE
- has been mostly ignored by other research communities
  - control science
  - formal methods in computer science

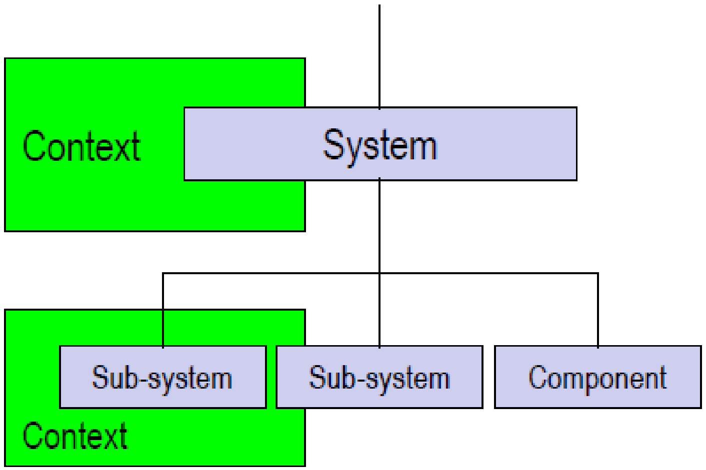
Source: A. Benveniste

Requirements on the meta-theory



Source: A. Benveniste

Requirements on the meta-theory {environment, component}

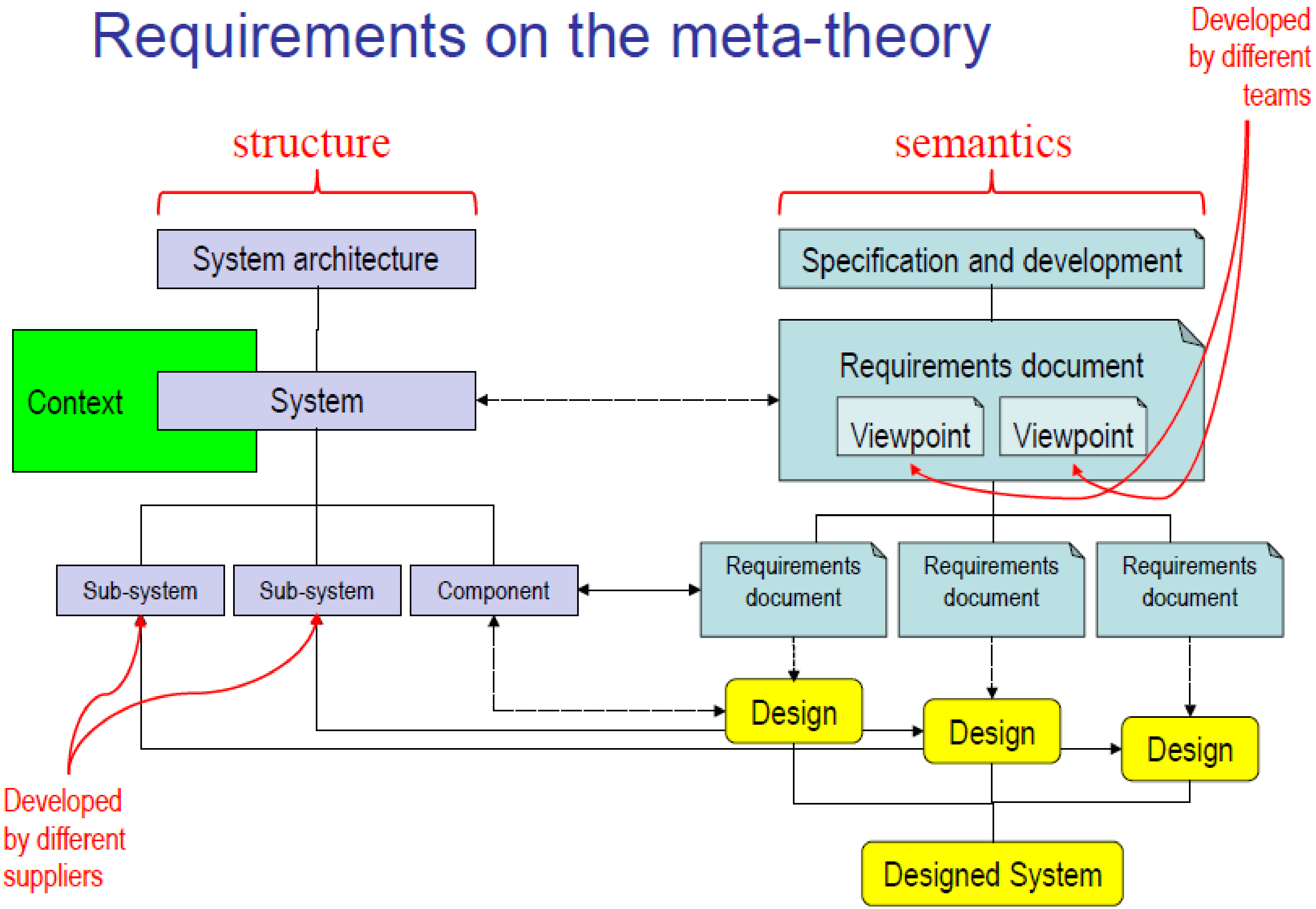


Contexts are important

- What the system guarantees: must be met by any implementation
- What the system assumes about its context of use: must be met by any legal environment

Source: A. Benveniste

Requirements on the meta-theory

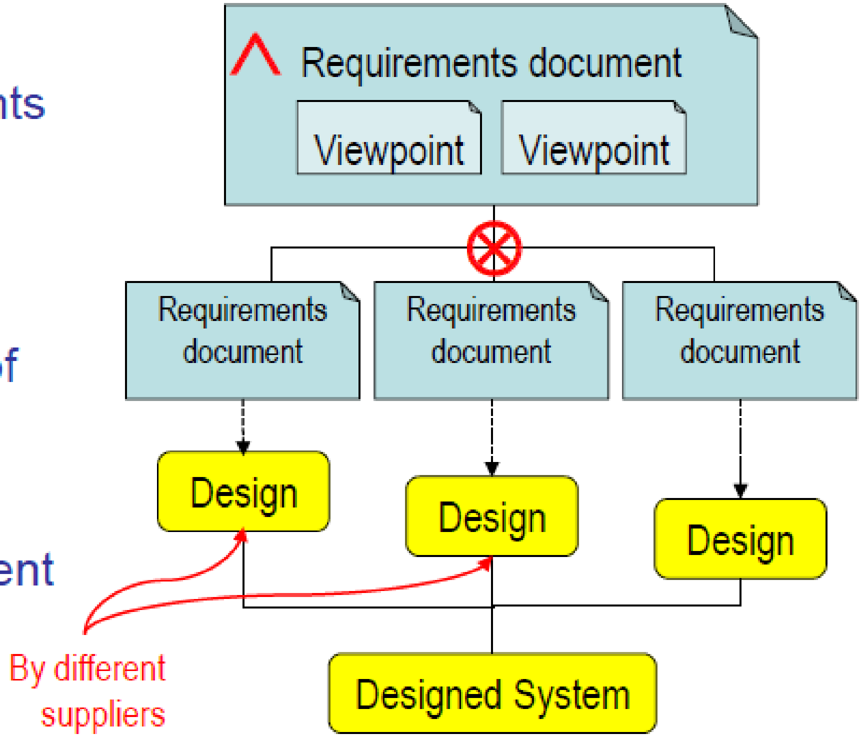


Source: A. Benveniste

Requirements on the meta-theory conjunction and parallel composition

From $\wedge$ to $\otimes$

- Requirements documents decompose into chapters/viewpoints: conjunction $\wedge$
- System = architecture of sub-systems: composition $\otimes$
- Independent development



Source: A. Benveniste

Conjunction of contracts

- Development of each viewpoint is performed under assumptions regarding its context of use, including the other viewpoints
- Conjunction is used to fuse viewpoints and get the full system specification
- Each viewpoint is itself a conjunction of requirements
- Consistency checking for contracts obtained as a conjunction is mandatory (is there an implementation satisfying all the requirements?)

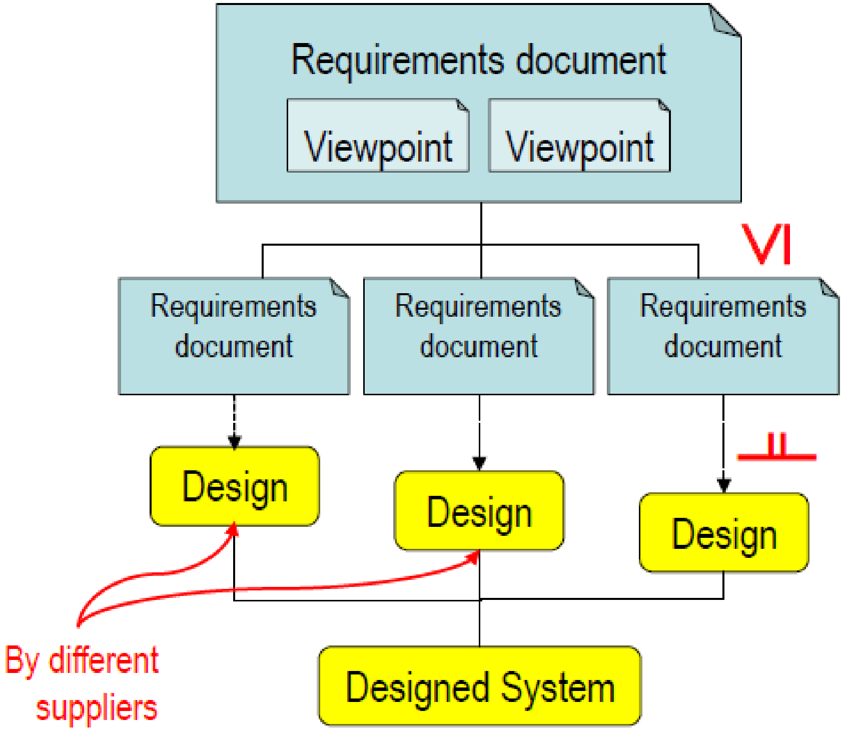
Requirements on the meta-theory
implements and refines

Designed component
$\models$ local contract

- Meets the guarantees under any legal environment

Decomposed contract
$\leq$ global contract

- Stronger guarantees
- Relaxed context



Source: A. Benveniste

Refinement and composition of contracts

$$C_{11} \otimes C_{12} \otimes C_{13} \preccurlyeq C_1$$

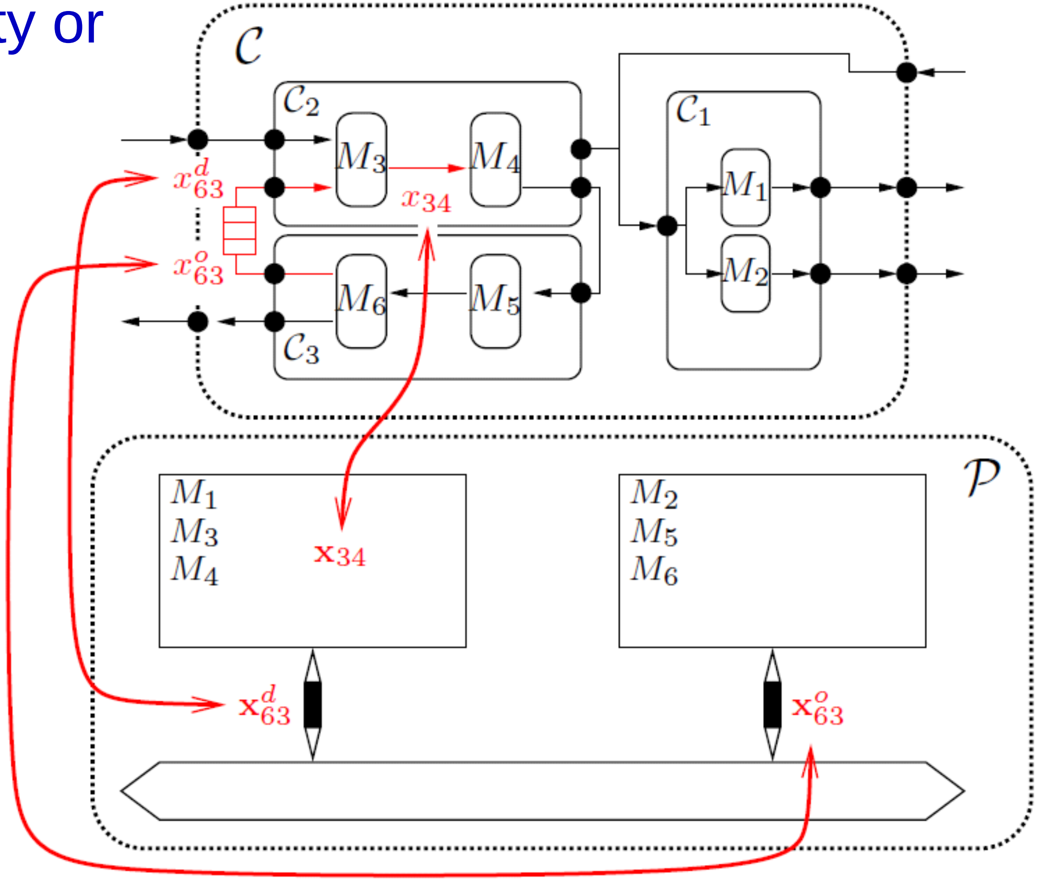
- Obtaining the three sub-contracts C_{11}, C_{12}, C_{13} is the art of the designer, based on architectural considerations
- Subsystems can also developed by re-using off-the-shelf components
- Contract theories offer the following services:
  - Firmly assess whether the above relation holds for the decomposition step
  - Formally check the compatibility of C_{11}, C_{12}, C_{13}
  - Guarantee that the information provided to suppliers is self-contained

Facilitating integration of specialized tool and frameworks

- Systems are developed by composing pieces that have been (in part) pre-designed by other groups or companies
  - Routinely done in vertical design chains (avionics, automotive,…)
  - …but in a heuristic and ad hoc way
- Need for standards, methods and tools in the software and hardware domains to allow integration of subsystems and their implementations
  - Across the electronic and mechanical domains (near future), but also across chemical and biology domains (further future) for nano-systems
  - From a static standpoint: data dictionaries, off-line model transformations,…
  - From a dynamic standpoint: co-simulation, HW-in-the-loop simulations and emulation

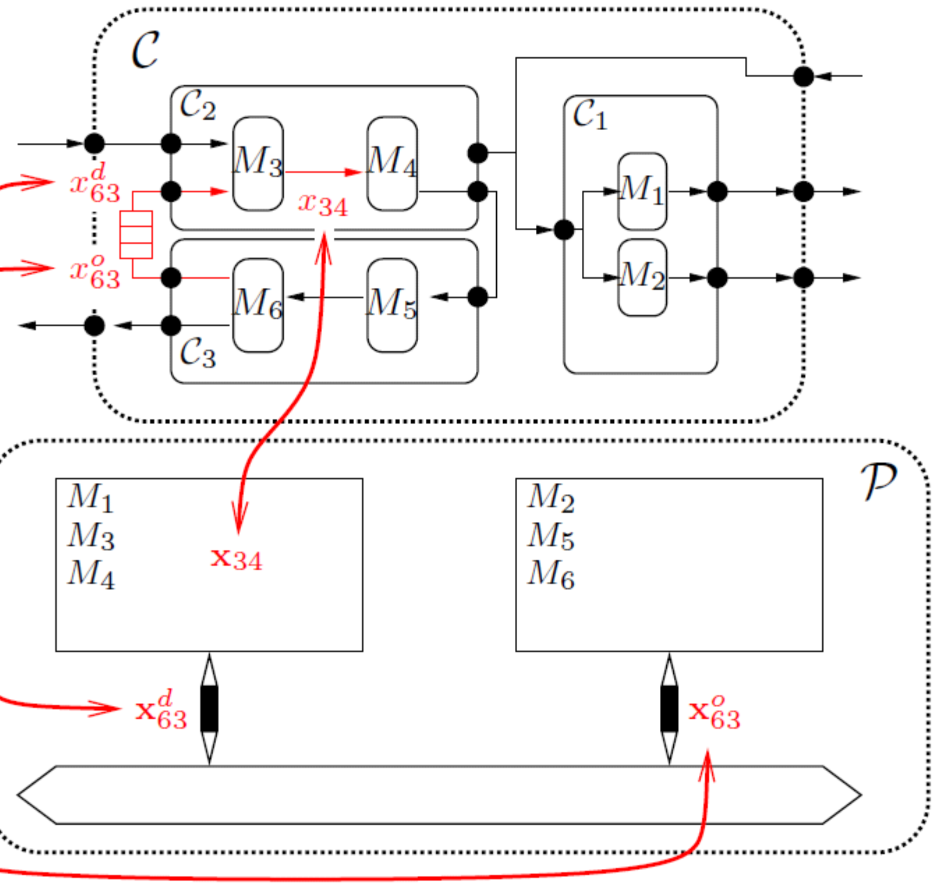
Deployment and mapping

- The satisfaction of safety or timing viewpoints by a considered deployment depends on
  - the supporting execution platform
  - the mapping of the application to the execution platform.
- How to check deployment compositionally?



Deployment and mapping

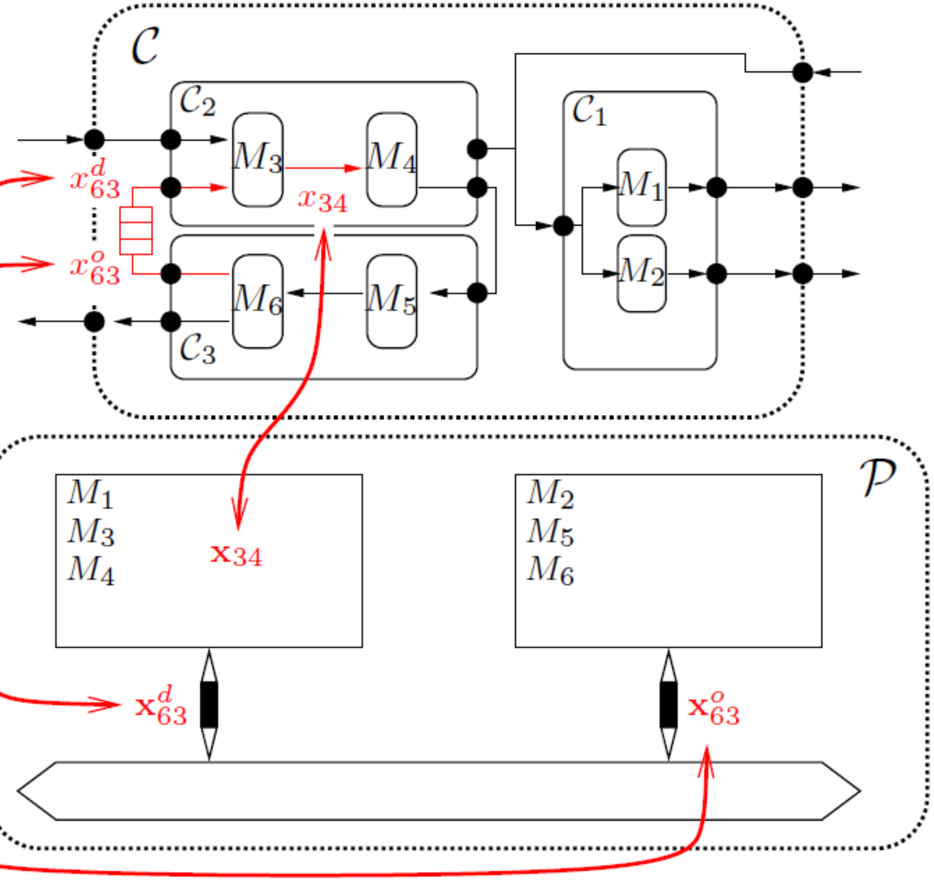
$$\mathcal{C} = \bigwedge_k \left(\bigotimes_{i\in I_k} \mathcal{C}_{ik}\right)$$
$$\mathcal{P} = \bigotimes_{j\in J} \left(\bigwedge_{\ell\in L_j} \mathcal{P}_{j\ell}\right)$$



- Virtual model of the execution platform $\mathcal{P}$
  - available computing units
  - bus protocol
  - library of RTOS services
- Components enhanced with timing information, fault propagation information

Deployment and mapping as parallel composition

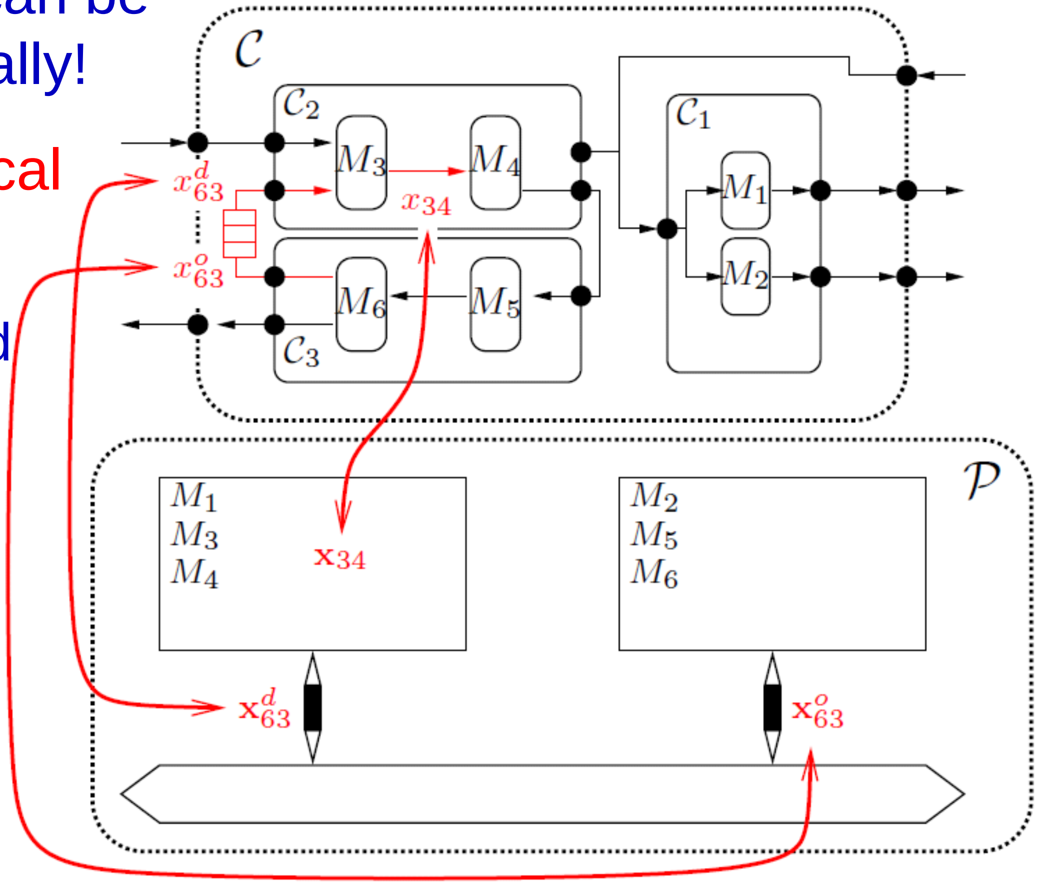
$$\mathcal{C} = \bigwedge_k \left(\bigotimes_{i\in I_k} \mathcal{C}_{ik}\right)$$
$$\mathcal{P} = \bigotimes_{j\in J} \left(\bigwedge_{\ell\in L_j} \mathcal{P}_{j\ell}\right)$$



- Contracts attached to sub-systems or components
  - Application contract (top)
  - Platform (bottom)
- Composition defined by synchronization tuples, where both occurrences and values of the different elements are unified.

Deployment and mapping as parallel composition

- Checking $\mathcal{C} \otimes P \preccurlyeq \mathcal{C}$ can be performed compositionally!
- $\mathcal{C} \otimes P$ also called vertical contract
  - Relates application and computing platforms
  - Horizontal contracts relate components at the same level
- Mapping in RT-Builder and Metropolis



$$\mathcal{C} \otimes \mathcal{P} = \left[\bigwedge_k \left(\bigotimes_{i \in I_k} \mathcal{C}_{ik}\right)\right] \otimes \left[\bigotimes_{j \in J} \left(\bigwedge_{\ell \in L_j} \mathcal{P}_{j\ell}\right)\right]$$

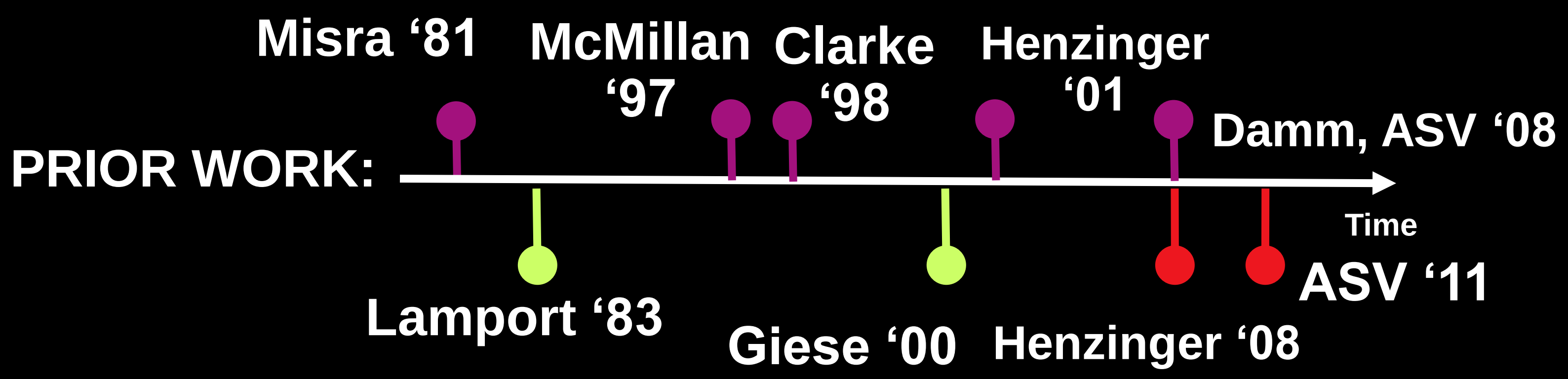
Outline: contracts and compositional methods for system design

- Where and why using contracts?
- Introduction to contracts
  - Components and contracts
  - Contract operators and properties
    - Incremental design
    - Independent implementability
- Mathematical meta-theory of contracts
- Overview of concrete contract theories
- Application examples

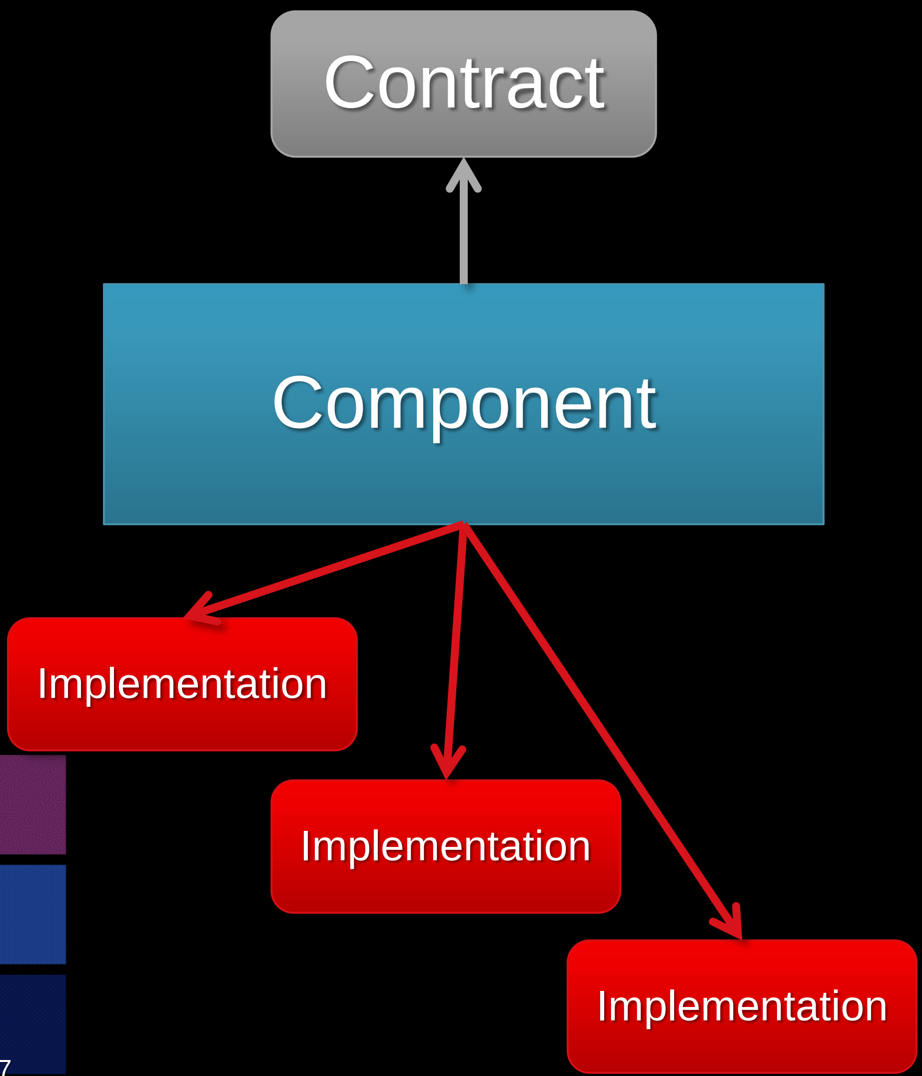
Compositional Reasoning

Reliably derive global properties of systems based on local properties of components

- Contracts as Assume/Guarantee pairs
- Component properties guaranteed under a set of assumptions on the environment
- Composition valid iff all assumptions are satisfied



Components



Set P of **ports**, $P = I \cup O$
Set A of **assumptions**
Set G of **guarantees**

An implementation M **satisfies** a contract (A, G), if M refines G in the context of A

$$M \cap A \subseteq G$$

Set P of **ports**, $P = I \cup O$
Set M of **behaviors**

Implementations and Contracts

Implementation	Contract
Defines the **behavior** of a component	Defines the **boundary of use** of a component
Does not restrict the environment of use	Declares the acceptable environments A (assumptions) and the limits of operation G (guarantees)
Usually deterministic	Usually non-deterministic
Usually tied to some particular architectural solution	Usually abstract, to encompass several possible different implementations

A simple static system

- M_1 computes the division between two real inputs and returns the result as a real output

$$M_1 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x, y \\ \text{outputs:} & z \end{cases} \\ \text{types:} & x, y, z \in \mathbb{R} \\ \text{behaviors:} & (y \neq 0 \rightarrow z = x/y) \wedge (y = 0 \rightarrow z = 0) \end{cases}$$

- C_1 intuitively specifies the intended behavior of components that implement division

$$\mathcal{C}_1 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x, y \\ \text{outputs:} & z \end{cases} \\ \text{types:} & x, y, z \in \mathbb{R} \\ \text{assumptions:} & y \neq 0 \\ \text{guarantees:} & z = x/y \end{cases}$$

Environments for C_1?

Implementations for C_1?

Is C_1 consistent?

Satisfaction

- **$M \cap A \subseteq G$**
- **$M \models (A, G)$**
- **An implementation must refine the guarantees, but only in the context of the acceptable environments**
  - **Less restrictive than regular refinement (trace containment)**
  - **Refinement is only up to the acceptable contexts**
  - **Implementations are free to behave as they like (and even break guarantees) for non acceptable contexts**

Composition

- Contract composition enables incremental design
- Component composability is a syntactic property (type matching)

$$M_1 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x, y \\ \text{outputs:} & z \end{cases} \\ \text{types:} & x, y, z \in \mathbb{R} \\ \text{behaviors:} & (y \neq 0 \rightarrow z = x/y) \wedge (y = 0 \rightarrow z = 0) \end{cases}$$

$$M_2 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x \\ \text{outputs:} & y \end{cases} \\ \text{types:} & x, y \in \mathbb{R} \\ \text{behaviors:} & y = e^x \end{cases}$$

Are M_1 and M_2 composable?

M_1 $M_1 \times M_2$? M_2

Composition

- Contract composition enables incremental design
- Component composability is a syntactic property (type matching)

$M_1 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x, y \\ \text{outputs:} & z \end{cases} \\ \text{types:} & x, y, z \in \mathbb{R} \\ \text{behaviors:} & (y \neq 0 \rightarrow z = x/y) \wedge (y = 0 \rightarrow z = 0) \end{cases}$	$M'_2 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x \\ \text{outputs:} & z \end{cases} \\ \text{types:} & x, z \in \mathbb{R} \\ \text{behaviors:} & z = abs(x) \end{cases}$

Are M_1 and M'_2 composable?

M_1

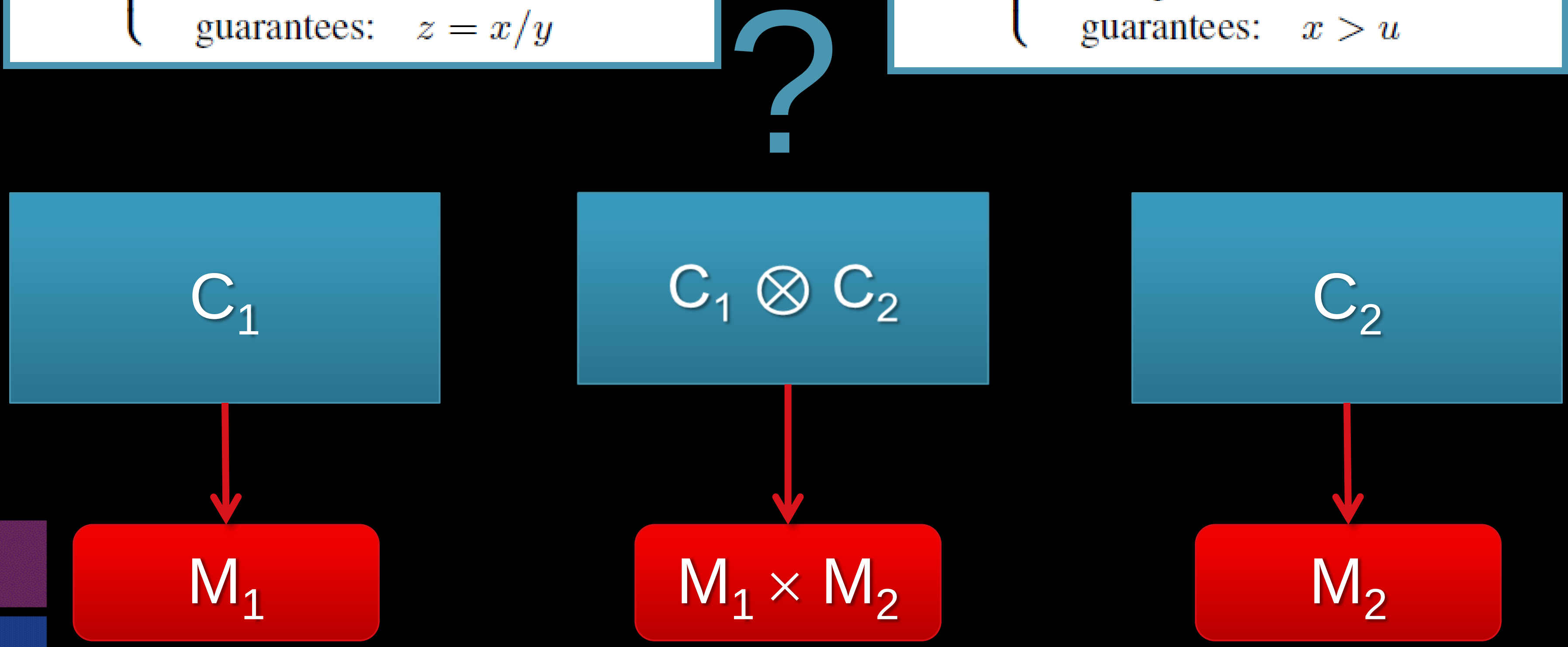
$M_1 \times M'_2$?

M'_2

Composition

$$\mathcal{C}_1 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x, y \\ \text{outputs:} & z \end{cases} \\ \text{types:} & x, y, z \in \mathbb{R} \\ \text{assumptions:} & y \neq 0 \\ \text{guarantees:} & z = x / y \end{cases}$$

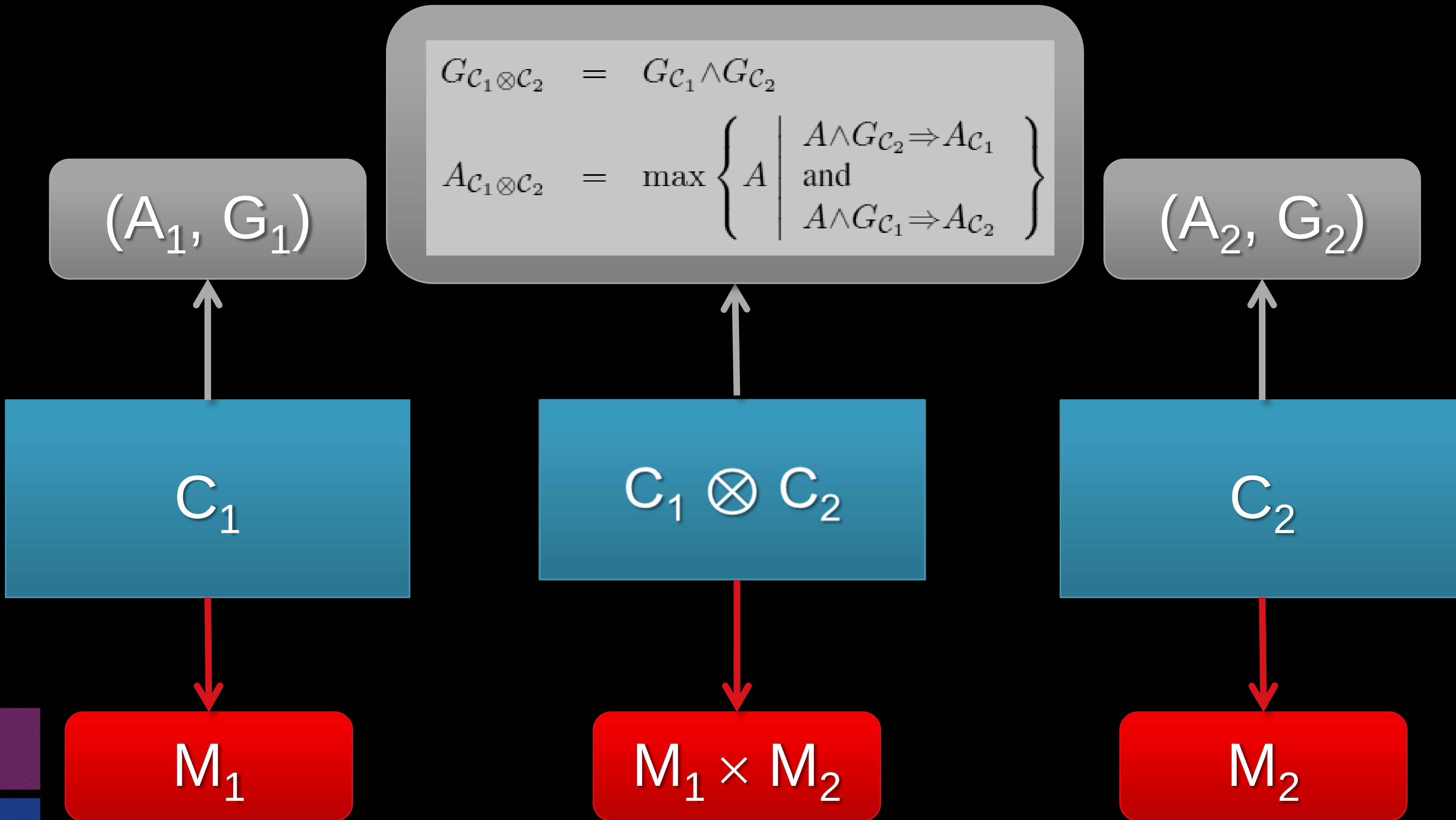
$$\mathcal{C}_2 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & u \\ \text{outputs:} & x \end{cases} \\ \text{types:} & u, x \in \mathbb{R} \\ \text{assumptions:} & \mathrm{T} \\ \text{guarantees:} & x > u \end{cases}$$



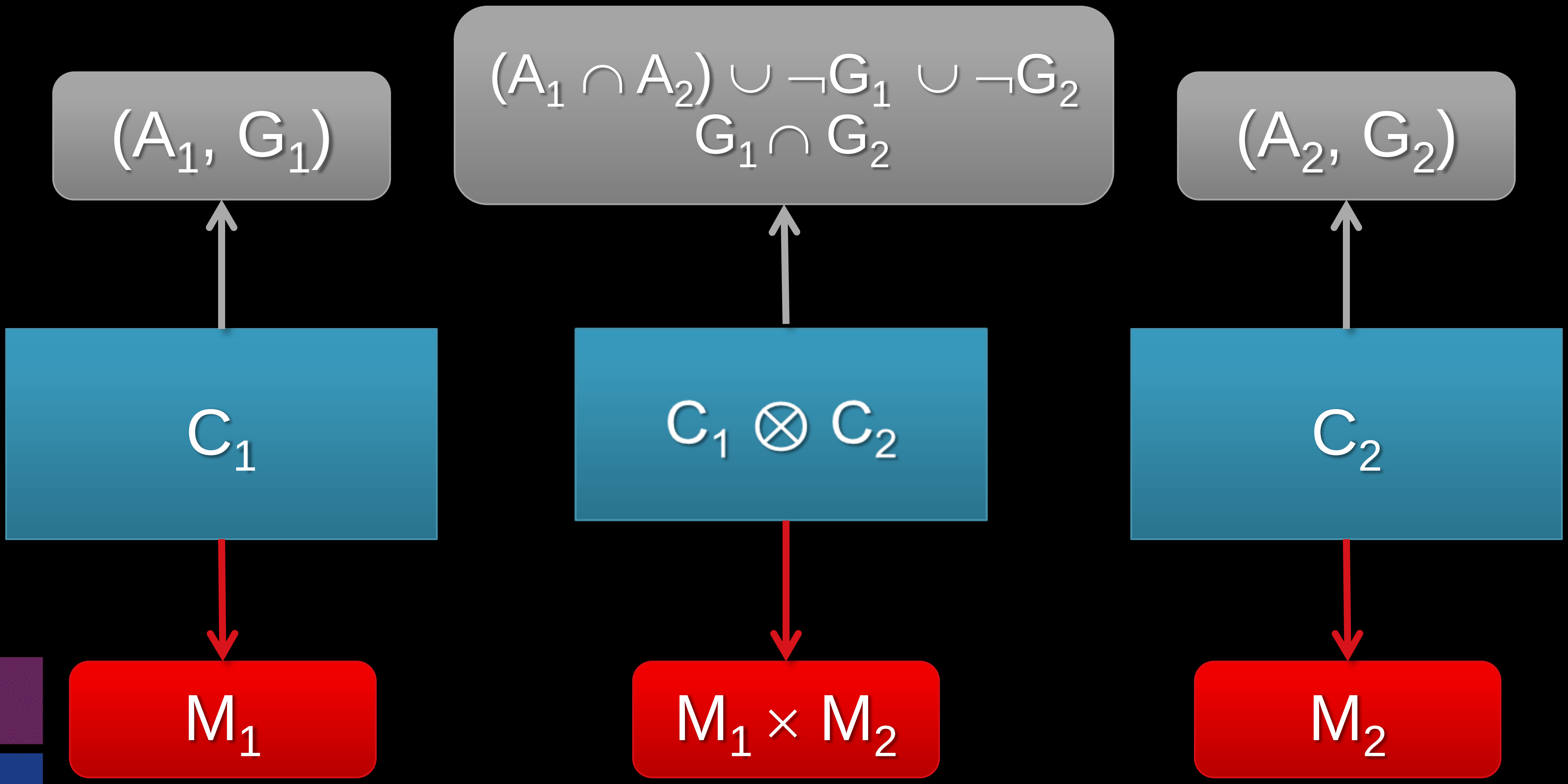
Compatibility

- Two contracts are compatible when the guarantees of one do not violate the assumptions of the other
  - And/or when an environment can enforce that condition
- A notion that cannot be expressed on implementations
  - They lack sufficient expressiveness
- A combined syntactic and semantic property

Composition



Composition



Composition

- The composite must guarantee what the components guarantee
  - $G_1 \cap G_2$
- The composite accepts only what is accepted by both components
  - $A_1 \cap A_2$
- However, part of the assumptions of one component can be discharged directly by the other, and vice-versa
  - $(A_1 \cap A_2) \cup \neg G_1 \cup \neg G_2$
  - Look a the weakest assumption assuring that both the initial contract assumptions are met
- Composition must be both associative and commutative

Composition

$$\mathcal{C}_1 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x, y \\ \text{outputs:} & z \end{cases} \\ \text{types:} & x, y, z \in \mathbb{R} \\ \text{assumptions:} & y \neq 0 \\ \text{guarantees:} & z = x/y \end{cases}$$

$$\mathcal{C}_2 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & u \\ \text{outputs:} & x \end{cases} \\ \text{types:} & u, x \in \mathbb{R} \\ \text{assumptions:} & \mathrm{T} \\ \text{guarantees:} & x > u \end{cases}$$

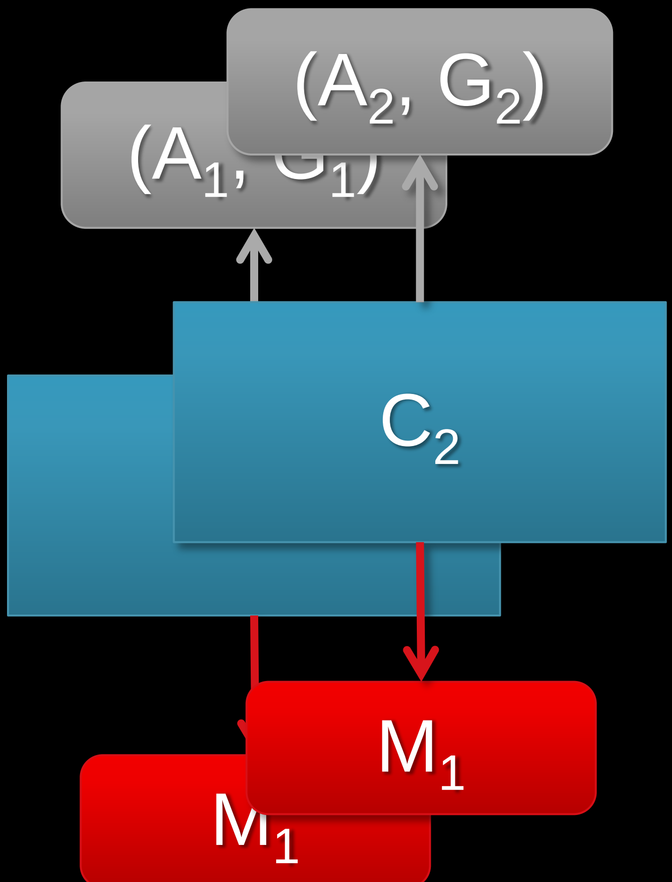
$$\mathcal{C}_1 \otimes \mathcal{C}_2 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & u, y \\ \text{outputs:} & x, z \end{cases} \\ \text{types:} & x, y, u, z \in \mathbb{R} \\ \text{assumptions:} & y \neq 0 \\ \text{guarantees:} & x > u \wedge z = x/y \end{cases}$$

$C_1 \to M_1$

$M_1 \times M_2$

$C_2 \to M_2$

Viewpoints



- Same component
- Different aspects

- Aspects do not combine by parallel composition
- We must instead take their conjunction

Refinement and conjunction

- Conjunction better defined as the greatest lower bound of a refinement order
  - A contract C =(A, G) is stronger than, or refines a contract C’ = (A’, G’) whenever it guarantees more while assuming less
  - $G \subseteq G'$
  - $A \supseteq A'$
  - (If M satisfies C, then M also satisfies C’)
- Greatest lower bound
  - $C \wedge C' = (A \cup A', G \cap G')$
  - The conjunction must accept the environments of both viewpoints
  - The conjunction must enforce the guarantees of both viewpoints
  - If M satisfies both C and C’, then M satisfies $C \wedge C'$

Example: refinement and independent implementability

- For all contracts C_1, C_2, C'_1, C'_2, let
  - C_1 be compatible with C_2
  - $C'_1 \preccurlyeq C_1$ and $C'_2 \preccurlyeq C_2$

  Then C'_1 is compatible with C'_2 and $C'_1 \otimes C'_2 \preccurlyeq C_1 \otimes C_2$
- Example:

$$\mathcal{C}''_1 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & y \\ \text{outputs:} & z \end{cases} \\ \text{types:} & y, z \in \mathbb{R} \\ \text{assumptions:} & y \neq 0 \\ \text{guarantees:} & z \in \mathbb{R} \end{cases}$$

$$\mathcal{C}''_2 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x \\ \text{outputs:} & y \end{cases} \\ \text{types:} & x, y \in \mathbb{R} \\ \text{assumptions:} & \mathrm{T} \\ \text{guarantees:} & y > 0 \end{cases}$$

$$\mathcal{C}_1 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x, y \\ \text{outputs:} & z \end{cases} \\ \text{types:} & x, y, z \in \mathbb{R} \\ \text{assumptions:} & y \neq 0 \\ \text{guarantees:} & z = x/y \end{cases}$$

  - C''_1 compatible with C''_2
  - $C_1 \preccurlyeq C''_1$

  Then $C_1 \otimes C''_2 \preccurlyeq C''_1 \otimes C''_2$

Example: conjunction

- For all contracts C_1, C_2 shared refinable, then
  - $C_1 \wedge C_2 \preccurlyeq C_1$ and $C_1 \wedge C_2 \preccurlyeq C_2$
  - for all C, if $C \preccurlyeq C_1$ and $C \preccurlyeq C_2$ then $C \preccurlyeq C_1 \wedge C_2$
- Example:
  - $C^T{}_2$ shared refinable with C_1
  - $C^T{}_2 \wedge C_1$ guarantees, in addition to C_1, a latency with bound 1.

$$\mathcal{C}_2^T : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & y, t_x, t_y \\ \text{outputs:} & t_z \end{cases} \\ \text{types:} & y \in \mathbb{R}; t_x, t_y, t_z \in \mathbb{R}_+ \\ \text{assumptions:} & y \neq 0 \\ \text{guarantees:} & t_z \leq \max(t_x, t_y) + 1 \end{cases}$$

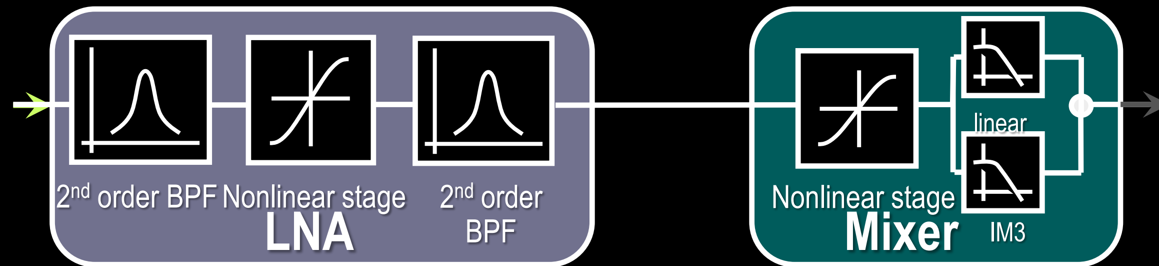
$$\mathcal{C}_1 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x, y \\ \text{outputs:} & z \end{cases} \\ \text{types:} & x, y, z \in \mathbb{R} \\ \text{assumptions:} & y \neq 0 \\ \text{guarantees:} & z = x / y \end{cases}$$

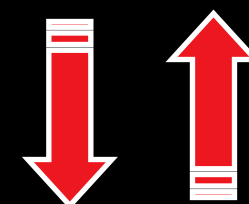
Platform-Based Design: Horizontal and Vertical Contracts

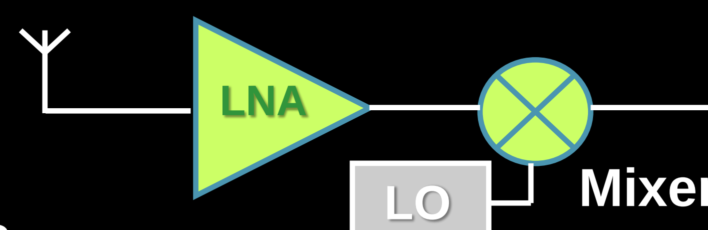
- So far, contracts for components at the same level of abstraction
  - We refer to this kind of composition as **horizontal**, and we talk about **horizontal contracts**
- A component could express assumptions and guarantees w.r.t. another level of abstraction
  - For instance, it may assume an execution environment with certain properties or performance
  - Likewise, it may guarantee certain patterns of activation to the execution environment
  - Contracts that span different levels of abstraction are referred to as **vertical contracts**

Vertical contracts

- Top-down vertical contracts
  - Capture **the desired specifications** (gain, bandwidth, carrier frequency, …)
  - Limit the impact of **undesired behaviors** (loading, crosstalk, coupling,…)
  - **Take into consideration the approximations** of the system model used to reduce complexity

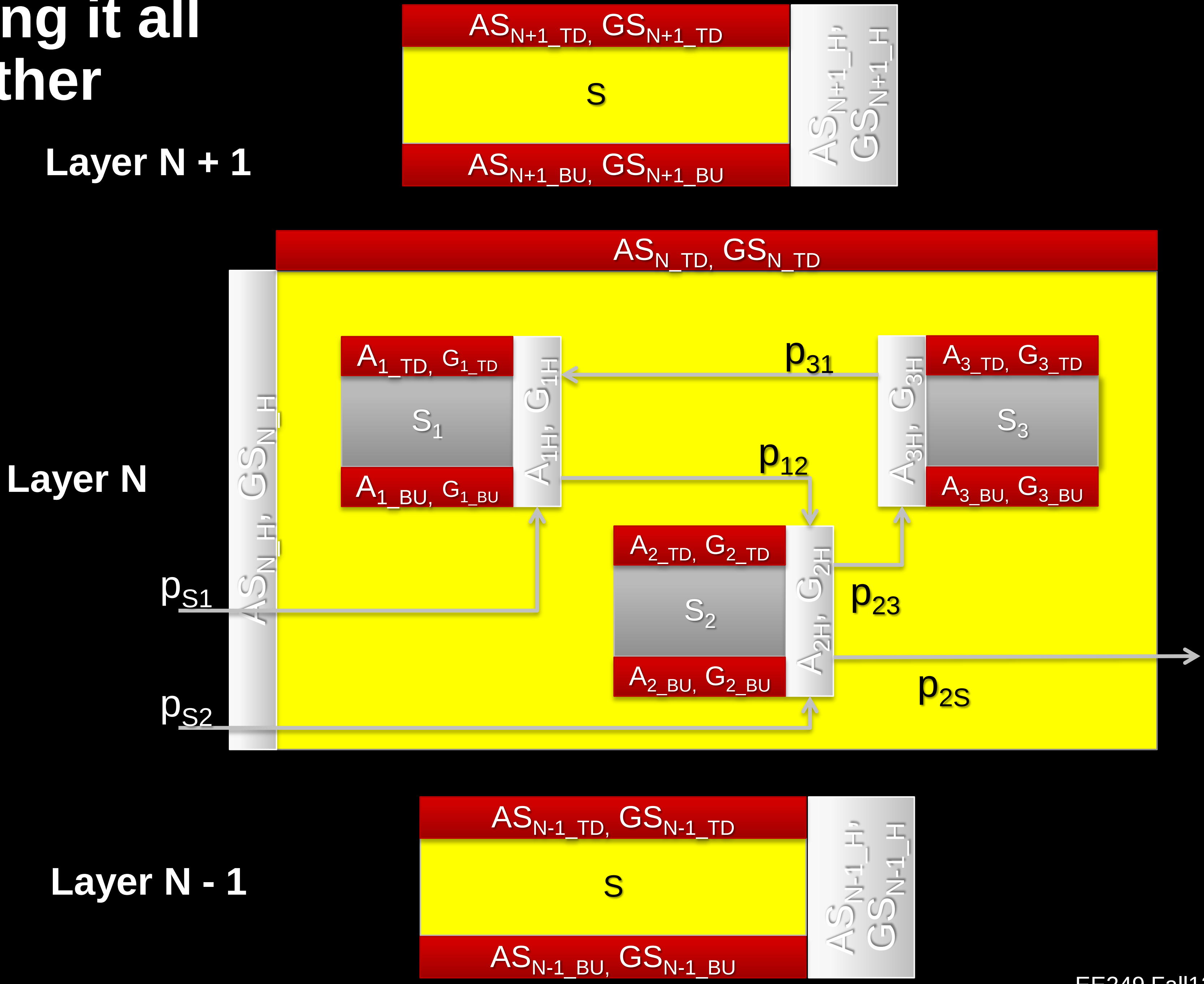






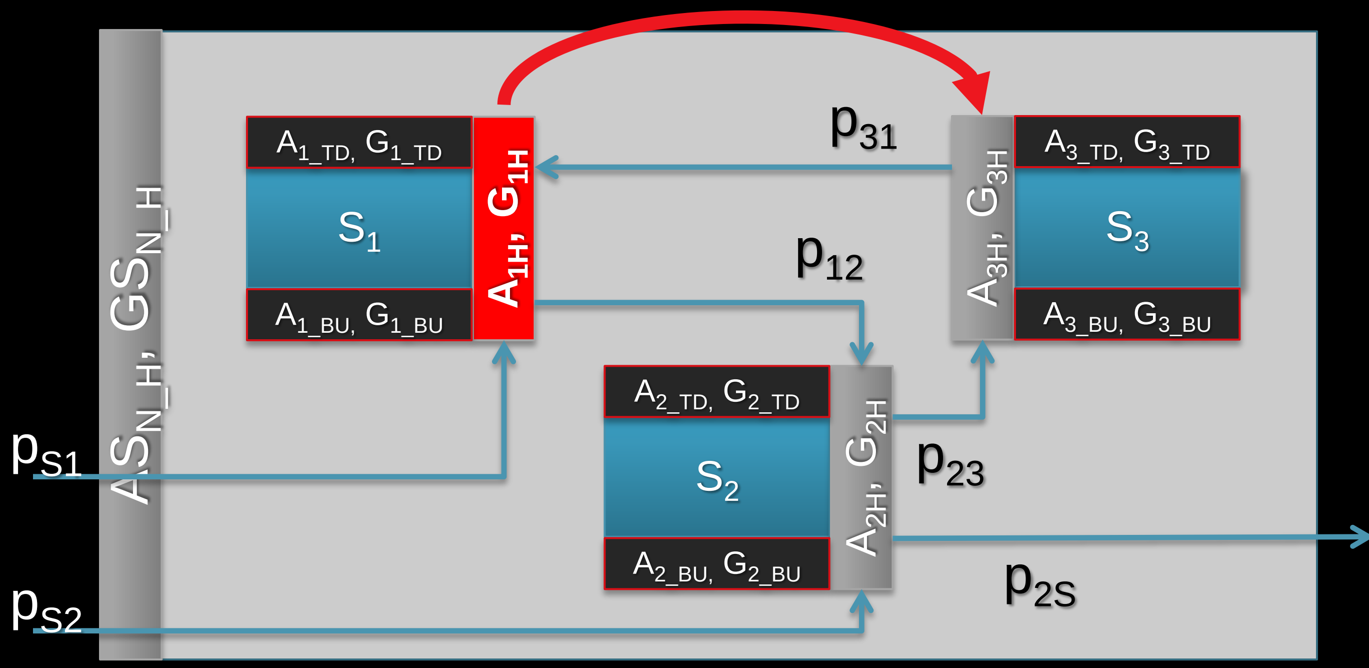
- Bottom-up vertical contracts
  - enforce validity **of higher-level macro-models w.r.t. lower level models**

Putting it all together



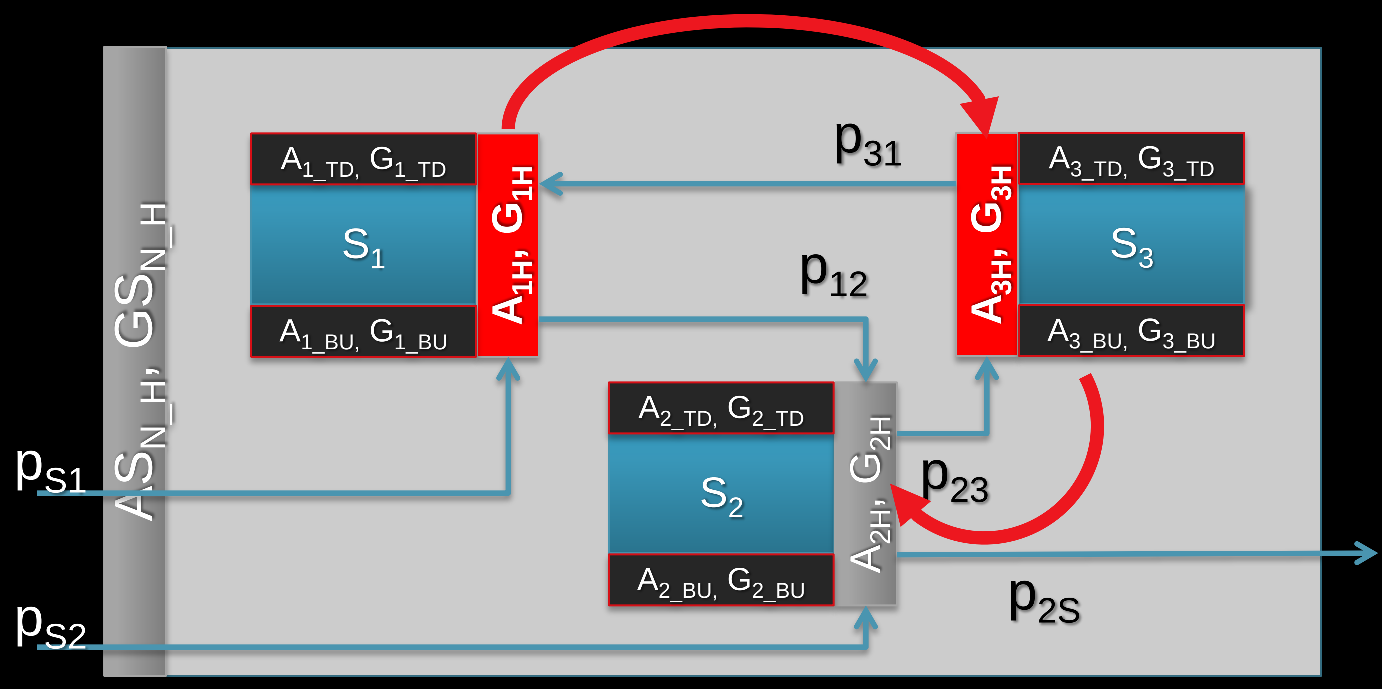
Circular reasoning in horizontal contracts

- $\mathbf{G_{1H}}$ can only be guaranteed for legal design contexts of S_1
  - need to establish $\mathbf{A_{1H}}$
- To establish $\mathbf{A_{1H}}$, want to involve $\mathbf{G_{3H}}$, hence $\mathbf{A_{3H}}$



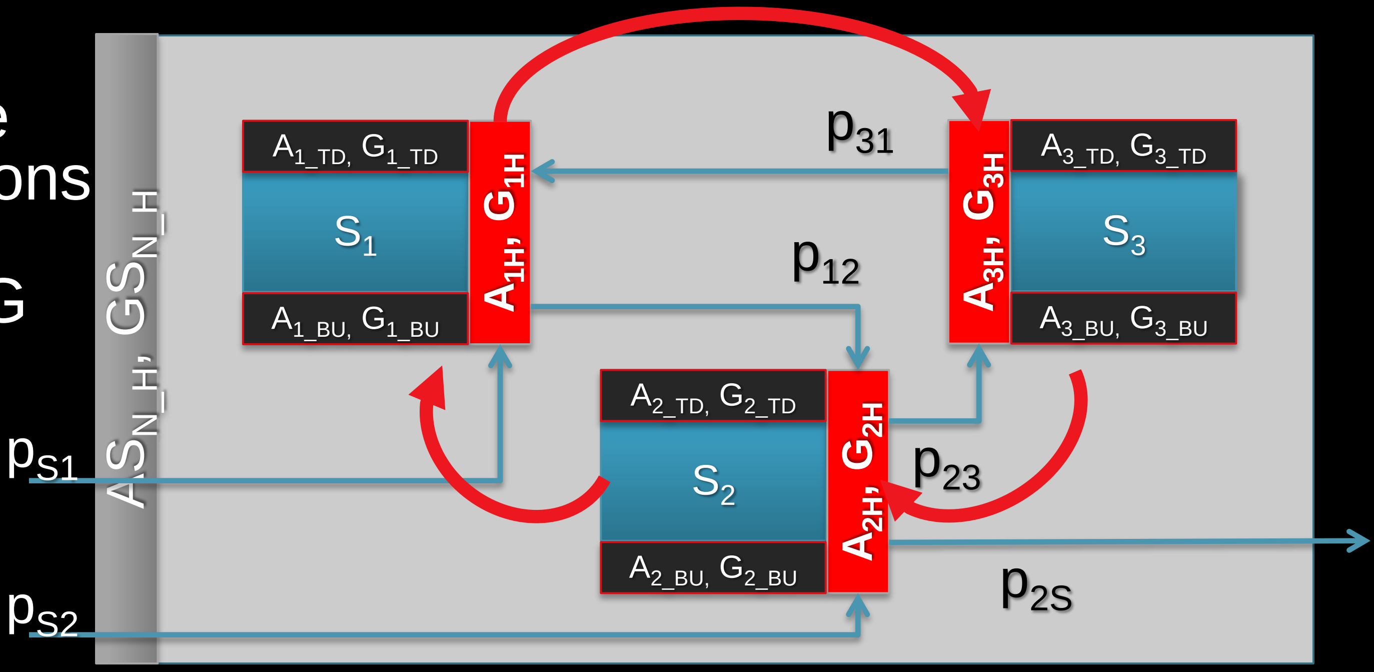
Circular reasoning in horizontal contracts

- To establish $\mathbf{A_{3H}}$, need to involve $\mathbf{G_{2H}}$ as a witness, hence $\mathbf{A_{2H}}$

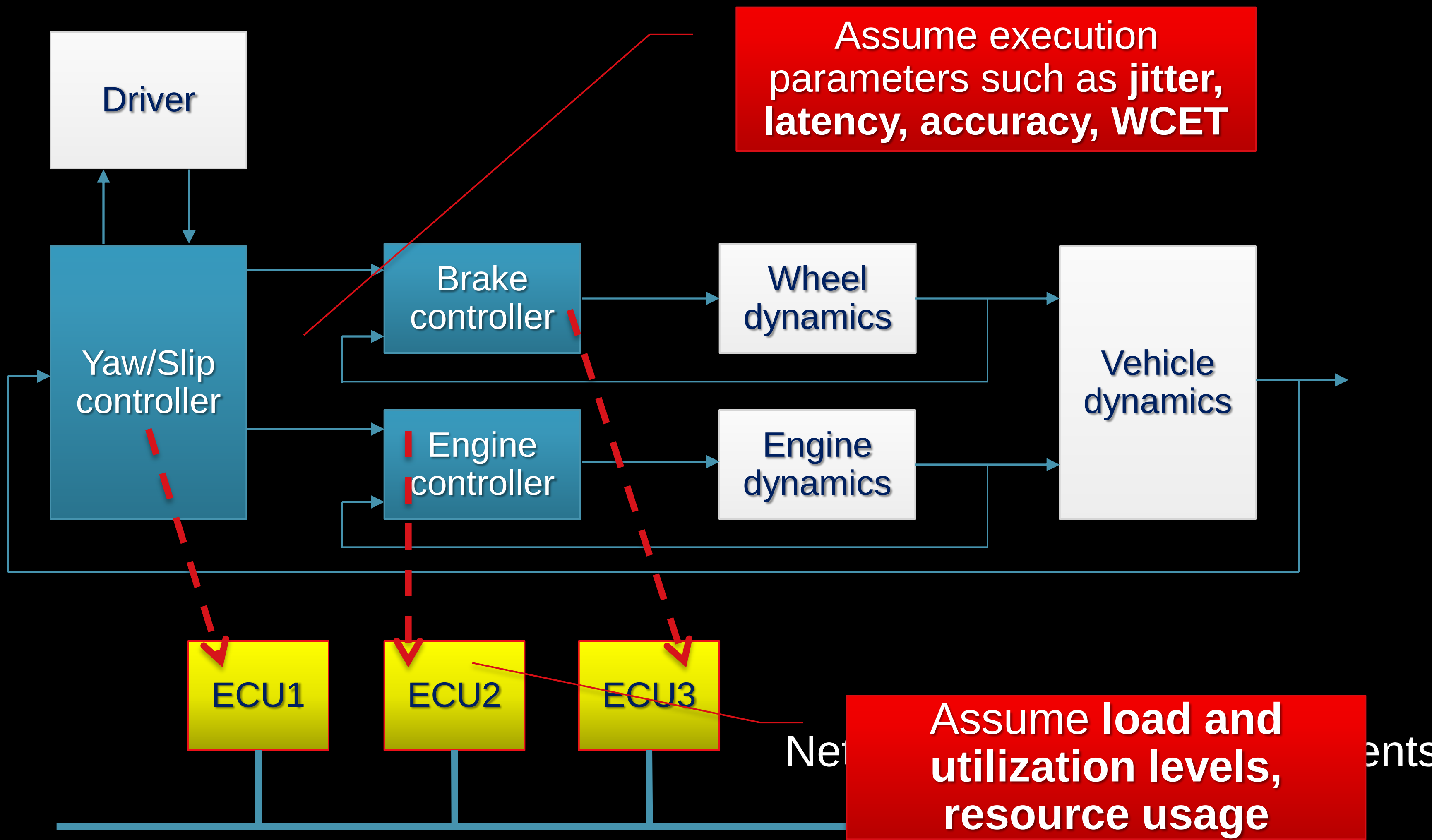


Circular reasoning in horizontal contracts

- …but to establish $\mathbf{A_{2H}}$, we need to establish $\mathbf{G_{1H}}$, which is where we started the chain!
- Seemingly circular arguments are justified for assumptions and guarantees expressed as safety properties
  - can be proven or disproved by finite observations
  - need to observe restrictions on how A/G refer to ports



Cyber-physical control system



Contracts

Controller

- **Assumptions**
  - Overall closed loop latency less than T
  - Compute resource availability above P%
  - Computing MIPS power at least M
- **Guarantees**
  - Model prediction error bounded by Δ
  - Deadlock state unreachable
  - Response time less than R

Platform

- **Assumptions**
  - Task activation interval no more than I
  - Task is thread and memory safe (can avoid checks)
  - Task are jitter independent within period
- **Guarantees**
  - Temporal isolation
  - Minimum dedicated computing power
  - Minimum sensor accuracy S

Outline: Contracts and compositional methods for system design

- Where and why using contracts?
- Introduction to contracts
- Mathematical meta-theory of contracts
  - Components and composition
  - Contracts
  - Refinement and conjunction
  - Contract composition
  - Quotient
  - Observers
- Overview of concrete contract theories
- Application examples

The meta-theory

- We assume some primitive concepts

Component	M *open* or *closed*
Composability	A type property
Composition	$\times$ commutative & associative
Environment	$E : E \times M$ closed

- On top of these primitive concepts we define generic operators satisfying generic properties
- How primitive concepts, operators, and properties, are made effective depends on the specific framework

Source: A. Benveniste

The meta-theory

- Generic Relations and Operators:

Contract	$\mathcal{C} = (\mathcal{E}_c, \mathcal{M}_c) =$ (set of environments , set of implementations)
Consistency	$\mathcal{M}_c \neq \emptyset$
Compatibility	$\mathcal{E}_c \neq \emptyset$
Implementation	$M \models^{M} \mathcal{C}$ iff $M \in \mathcal{M}_c$; $E \models^{E} \mathcal{C}$ iff $E \in \mathcal{E}_c$

Source: A. Benveniste

The meta-theory

- Generic Relations and Operators:

Contract	$\mathcal{C} = (\mathcal{E}_c, \mathcal{M}_c) =$ (set of environments , set of implementations)
Consistency	$\mathcal{M}_c \neq \emptyset$
Compatibility	$\mathcal{E}_c \neq \emptyset$
Implementation	$M \models^{\mathrm{M}} \mathcal{C}$ iff $M \in \mathcal{M}_c$; $E \models^{\mathrm{E}} \mathcal{C}$ iff $E \in \mathcal{E}_c$
Refinement	$\mathcal{C}' \preceq \mathcal{C}$ iff $\mathcal{E}_{c'} \supseteq \mathcal{E}_c$ and $\mathcal{M}_{c'} \subseteq \mathcal{M}_c$
Conjunction	$\mathcal{C}_1 \wedge \mathcal{C}_2 =$ GLB for $\preceq$

Source: A. Benveniste

Refinement and conjunction: properties

- Expressible contract family *C*. Every set $\mathcal{M}' \subseteq \mathcal{M}$ of components can be represented as $\mathcal{M}' = \mathcal{M}_C$ for some contract C, and similarly for sets of environments.
  - Only contracts belonging to this family can be considered
- Shared refinement
  - Any contract that refines $C_1 \wedge C_2$ also refines C_1 and C_2. Any implementation of $C_1 \wedge C_2$ is a shared implementation of C_1 and C_2. Any environment of C_1 and C_2 is an environment of $C_1 \wedge C_2$.
  - For **C'** ⊆ **C** subset of contracts, $\wedge \mathbf{C}'$ is compatible if and only if there exists a compatible $C \in$ **C'**

The meta-theory

- Generic Relations and Operators:

Contract	$\mathcal{C} = (\mathcal{E}_c, \mathcal{M}_c) =$ (set of environments , set of implementations)
Consistency	$\mathcal{M}_c \neq \emptyset$
Compatibility	$\mathcal{E}_c \neq \emptyset$
Implementation	$M \models^{\mathrm{M}} \mathcal{C}$ iff $M \in \mathcal{M}_c$; $E \models^{\mathrm{E}} \mathcal{C}$ iff $E \in \mathcal{E}_c$
Refinement	$\mathcal{C}' \preceq \mathcal{C}$ iff $\mathcal{E}_{c'} \supseteq \mathcal{E}_c$ and $\mathcal{M}_{c'} \subseteq \mathcal{M}_c$
Conjunction	$\mathcal{C}_1 \wedge \mathcal{C}_2 =$ GLB for $\preceq$
Composition	$\mathcal{C}_1 \otimes \mathcal{C}_2$ is defined if $\left.\begin{array}{l} M_1 \models^{\mathrm{M}} \mathcal{C}_1 \\ M_2 \models^{\mathrm{M}} \mathcal{C}_2 \end{array}\right\} \Rightarrow (M_1, M_2)$ composable $\mathcal{C}_1 \otimes \mathcal{C}_2 = \bigwedge \left\{ \mathcal{C} \,\middle\vert\, \begin{array}{c} M_1 \models^{\mathrm{M}} \mathcal{C}_1 \text{ and } M_2 \models^{\mathrm{M}} \mathcal{C}_2 \text{ and } E \models^{\mathrm{E}} \mathcal{C} \\ \Downarrow \\ M_1 \times M_2 \models^{\mathrm{M}} \mathcal{C} \text{ and } E \times M_2 \models^{\mathrm{E}} \mathcal{C}_1 \text{ and } E \times M_1 \models^{\mathrm{E}} \mathcal{C}_2 \end{array} \right\}$

Source: A. Benveniste

The meta-theory

- Generic Relations and Operators:

Contract	$\mathcal{C} = (\mathcal{E}_c, \mathcal{M}_c) =$ (set of environments , set of implementations)
Consistency	$\mathcal{M}_c \neq \emptyset$
Compatibility	$\mathcal{E}_c \neq \emptyset$
Implementation	$M \models^{\mathrm{M}} \mathcal{C}$ iff $M \in \mathcal{M}_c$; $E \models^{\mathrm{E}} \mathcal{C}$ iff $E \in \mathcal{E}_c$
Refinement	$\mathcal{C}' \preceq \mathcal{C}$ iff $\mathcal{E}_{c'} \supseteq \mathcal{E}_c$ and $\mathcal{M}_{c'} \subseteq \mathcal{M}_c$
Conjunction	$\mathcal{C}_1 \wedge \mathcal{C}_2 =$ GLB for $\preceq$
Composition	$\mathcal{C}_1 \otimes \mathcal{C}_2$ is defined if $\left.\begin{array}{l} M_1 \models^{\mathrm{M}} \mathcal{C}_1 \\ M_2 \models^{\mathrm{M}} \mathcal{C}_2 \end{array}\right\} \Rightarrow (M_1, M_2)$ composable $\mathcal{C}_1 \otimes \mathcal{C}_2 = \bigwedge \left\{ \mathcal{C} \left| \begin{array}{c} M_1 \models^{\mathrm{M}} \mathcal{C}_1 \text{ and } M_2 \models^{\mathrm{M}} \mathcal{C}_2 \text{ and } E \models^{\mathrm{E}} \mathcal{C} \\ \Downarrow \\ M_1 \times M_2 \models^{\mathrm{M}} \mathcal{C} \text{ and } E \times M_2 \models^{\mathrm{E}} \mathcal{C}_1 \text{ and } E \times M_1 \models^{\mathrm{E}} \mathcal{C}_2 \end{array} \right. \right\}$
Quotient	$\mathcal{C}_1 / \mathcal{C}_2 = \bigvee \{ \mathcal{C} \mid \mathcal{C} \otimes \mathcal{C}_2 \preceq \mathcal{C}_1 \}$ Least Upper Bound (LUB) for $\preceq$

Source: A. Benveniste

The meta-theory

- Generic Properties:

Refinement	substituability $\nearrow$ of environments substituability $\searrow$ of implementations
Composition	$\left.\begin{array}{r}(\mathcal{C}_1, \mathcal{C}_2) \text{ compatible} \\ \mathcal{C}_i' \preceq \mathcal{C}_i\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(\mathcal{C}_1', \mathcal{C}_2') \text{ compatible} \\ \mathcal{C}_1' \otimes \mathcal{C}_2' \preceq \mathcal{C}_1 \otimes \mathcal{C}_2\end{array}\right.$ independent implementability $\underbrace{(\mathcal{C}_1 \otimes \mathcal{C}_2) \otimes (\mathcal{C}_3 \otimes \mathcal{C}_4) = (\mathcal{C}_1 \otimes \mathcal{C}_3) \otimes (\mathcal{C}_2 \otimes \mathcal{C}_4)}_{\text{compatibility} \quad \Longrightarrow \quad \text{compatibility}}$ associativity $[(\mathcal{C}_{11} \wedge \mathcal{C}_{21}) \otimes (\mathcal{C}_{12} \wedge \mathcal{C}_{22})] \preceq [(\mathcal{C}_{11} \otimes \mathcal{C}_{12}) \wedge (\mathcal{C}_{21} \otimes \mathcal{C}_{22})]$ distributivity (freedom in design processes)
Quotient	$\mathcal{C} \preceq \mathcal{C}_1 / \mathcal{C}_2 \Leftrightarrow \mathcal{C} \otimes \mathcal{C}_2 \preceq \mathcal{C}_1$

Source: A. Benveniste

Independent implementability

$$\left.\begin{array}{r}(\mathcal{C}_1, \mathcal{C}_2) \text{ compatible} \\ \mathcal{C}'_i \preceq \mathcal{C}_i\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(\mathcal{C}'_1, \mathcal{C}'_2) \text{ compatible} \\ \mathcal{C}'_1 \otimes \mathcal{C}'_2 \preceq \mathcal{C}_1 \otimes \mathcal{C}_2\end{array}\right.$$

- (C_1, C_2) are compatible if their composition is defined and compatible
- Independent implementability says that compatible contracts can be independently *refined*
- Corollary: Compatible contracts can be independently *implemented*
  - Apply independent implementability with C'_1, C'_2 singletons

Independent implementability: proof sketch

$$\left.\begin{array}{r}(\mathcal{C}_1, \mathcal{C}_2) \text{ compatible} \\ \mathcal{C}'_i \preceq \mathcal{C}_i\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(\mathcal{C}'_1, \mathcal{C}'_2) \text{ compatible} \\ \mathcal{C}'_1 \otimes \mathcal{C}'_2 \preceq \mathcal{C}_1 \otimes \mathcal{C}_2\end{array}\right.$$

- Assumption 1

Let $\mathbf{C}_{C_1 \otimes C_2} = \left\{ \mathcal{C} \;\middle|\; \begin{array}{c} M_1 \models^{\mathrm{M}} \mathcal{C}_1 \text{ and } M_2 \models^{\mathrm{M}} \mathcal{C}_2 \text{ and } E \models^{\mathrm{E}} \mathcal{C} \\ \Downarrow \\ M_1 \times M_2 \models^{\mathrm{M}} \mathcal{C} \text{ and } E \times M_2 \models^{\mathrm{E}} \mathcal{C}_1 \text{ and } E \times M_1 \models^{\mathrm{E}} \mathcal{C}_2 \end{array} \right\}$

Then $C_1 \otimes C_2 \in \mathbf{C}_{C_1 \otimes C_2}$, i.e. the GLB is in the set.

- Assumption 1 ensures that:
  - Composing implementations of each contract yields an implementation for the composition;
  - Composing an environment for the resulting composition with an implementation for C_2 yields an implementation for C_1 and vice-versa.

Independent implementability: proof sketch

$$\left.\begin{array}{r}(\mathcal{C}_1,\mathcal{C}_2)\text{ compatible}\\ \mathcal{C}'_i\preceq\mathcal{C}_i\end{array}\right\}\Rightarrow\left\{\begin{array}{l}(\mathcal{C}'_1,\mathcal{C}'_2)\text{ compatible}\\ \mathcal{C}'_1\otimes\mathcal{C}'_2\preceq\mathcal{C}_1\otimes\mathcal{C}_2\end{array}\right.$$

- Lemma 1: Let $C'_i \preccurlyeq C_i$ and $C_1 \otimes C_2$ well defined (respective implementations are composable). Then so is $C'_1 \otimes C'_2$ and $\mathbf{C}_{C'_1 \otimes C'_2} \supseteq \mathbf{C}_{C_1 \otimes C_2}$.
- Proof of Lemma 1
  - $C_1 \otimes C_2$ well defined implies that every (M_1, M_2) implementing the contracts are composable. Hence, $C'_1 \otimes C'_2$ well defined.
  - Any respective implementations of C'_1 and C'_2 are also implementations of C_1 and C_2
  - Similarly any environment for C_1 and C_2 satisfies also C'_1 and C'_2
  - Replace in the composition formula C_1 by C'_1 and C_2 by C'_2,…

Independent implementability: proof sketch

$$\left.\begin{array}{r}(\mathcal{C}_1, \mathcal{C}_2) \text{ compatible} \\ \mathcal{C}'_i \preceq \mathcal{C}_i\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(\mathcal{C}'_1, \mathcal{C}'_2) \text{ compatible} \\ \mathcal{C}'_1 \otimes \mathcal{C}'_2 \preceq \mathcal{C}_1 \otimes \mathcal{C}_2\end{array}\right.$$

- Lemma 1: Let $C'_i \preccurlyeq C_i$ and $C_1 \otimes C_2$ well defined (respective implementations are composable). Then so is $C'_1 \otimes C'_2$ and $\mathbf{C}_{C'_1 \otimes C'_2} \supseteq \mathbf{C}_{C_1 \otimes C_2}$.
- Proof of Lemma 1
  - Replace in the composition formula C_1 by C'_1 and C_2 by C'_2

$$\mathbf{C}_{C_1 \otimes C_2} = \left\{ \mathcal{C} \,\middle|\, \begin{array}{c} M_1 \models^{\mathrm{M}} \mathcal{C}_1 \text{ and } M_2 \models^{\mathrm{M}} \mathcal{C}_2 \text{ and } E \models^{\mathrm{E}} \mathcal{C} \\ \Downarrow \\ M_1 \times M_2 \models^{\mathrm{M}} \mathcal{C} \text{ and } E \times M_2 \models^{\mathrm{E}} \mathcal{C}_1 \text{ and } E \times M_1 \models^{\mathrm{E}} \mathcal{C}_2 \end{array} \right\}$$

$$C_0 \in \mathbf{C}_{C_1 \otimes C_2} \Longrightarrow C_0 \in \mathbf{C}_{C'_1 \otimes C'_2}$$

- Independent implementability becomes a direct corollary of Lemma 1

Meta-theory versus concrete theories

- The meta-theory offers some fundamental properties (incremental development, independent implementability,…)
- Need concrete definitions for components, component compositions
- Need effective means to implement (or approximate) the notions of contracts, refinement, conjunction, composition…
- Observers provide a generic approach to recover effectiveness

Observers

- Originate from the basic notion of test for programs
- Definition. An observer for a contract C is a pair $(b_C{}^E, b_C{}^M)$ of non-deterministic Boolean functions called verdicts, such that:
  - $b_C{}^E(M)$ outputs F → $M \notin \mathcal{E}_C$
  - $b_C{}^M(M)$ outputs F → $M \notin \mathcal{M}_C$
- Non-determinism accounts for dependence of test on the environment stimuli and internal non-determinisms of the component
- Tests only provide semi-decision (arrow in one direction)

Mirroring the algebra of contracts with observers

Notion	Observer
$\mathcal{C} = (\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$	$(b^{\mathsf{E}}_{\mathcal{C}}, b^{\mathsf{M}}_{\mathcal{C}})$
$\mathcal{C} = \mathcal{C}_1 \wedge \mathcal{C}_2$ $\mathcal{C} = \mathcal{C}_1 \vee \mathcal{C}_2$	$b^{\mathsf{E}}_{\mathcal{C}} = b^{\mathsf{E}}_{\mathcal{C}_1} \vee b^{\mathsf{E}}_{\mathcal{C}_2}\ ,\ b^{\mathsf{M}}_{\mathcal{C}} = b^{\mathsf{M}}_{\mathcal{C}_1} \wedge b^{\mathsf{M}}_{\mathcal{C}_2}$ $b^{\mathsf{E}}_{\mathcal{C}} = b^{\mathsf{E}}_{\mathcal{C}_1} \wedge b^{\mathsf{E}}_{\mathcal{C}_2}\ ,\ b^{\mathsf{M}}_{\mathcal{C}} = b^{\mathsf{M}}_{\mathcal{C}_1} \vee b^{\mathsf{M}}_{\mathcal{C}_2}$
$\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$	$b^{\mathsf{E}}_{\mathcal{C}}(E) = \left\{ \begin{array}{c} b^{\mathsf{M}}_{\mathcal{C}_1}(M_1) \wedge b^{\mathsf{M}}_{\mathcal{C}_2}(M_2) \\ \Downarrow \\ b^{\mathsf{E}}_{\mathcal{C}_2}(E \times M_1) \wedge b^{\mathsf{E}}_{\mathcal{C}_1}(E \times M_2) \end{array} \right.$ $b^{\mathsf{M}}_{\mathcal{C}}(M_1 \times M_2) = b^{\mathsf{M}}_{\mathcal{C}_1}(M_1) \wedge b^{\mathsf{M}}_{\mathcal{C}_2}(M_2)$

- Nothing can be said about the relation of the observers for contracts in a refinement ordering
- Nothing can be inferred about contract refinement from observers in a refinement ordering

Observers: properties

- Lemma 2: $({b_C}^E, {b_C}^M)$ be an observer for C and let $C' \preccurlyeq C$. Then, any (b^E, b^M) satisfying $b^E \geq {b_C}^E$ and $b^M \leq {b_C}^M$ is an observer for C'.
- Lemma 3
  - If ${b_C}^E(E)$ outputs F for all tested environment *E*, then C is incompatible
  - If ${b_C}^M(M)$ outputs F for all tested component *M*, then C is inconsistent
- Still need to exercise all components or environments
  - Non-effective unless notion of "strongest" component or environment is provided in concrete theories

Observers: survey

- Widely studied for software and system testing
- Synchronous languages are a formalism of choice
  - E.g. Esterel, Lustre, Signal, …, Scade V6
  - Support only discrete dynamics
  - Benefit from a solid mathematical semantics
    - Results independent of the type of simulator
    - Consistency between simulated and generated code
- RT-Builder supports the combination of functional and timing viewpoints on top of Signal

Observers: survey

- Simulink/Stateflow
  - Mathematical semantics is less firmly defined
  - Supports CT dynamics in the forms of ODE
  - Simulink+SimScape allows including physical system models in observers
  - Similar considerations for Modelica
- Advocated in the context of Lustre, Scade, Esterel and Signal
  - In Scade, tests can be evaluated at run time while executing a program

Observers: survey

- Property Specification Language (PSL)
  - An industrial standard for functional properties targeted to digital HW
  - Well-suited specification language for expressing functional requirements involving sequential causality of actions and events
  - Suitable in the contract-based design using observers because of the existing tool support
    - E.g. FoCS translate PSL into checkers to be attached to the design
    - Used for generating transactors that adapt high-level requirements in TLM to RTL implementations
    - Exist a methodology for user-guided automated property exploration

Observers: survey

- Live Sequence Charts
  - Graphical specification language based on scenarios
  - Pre-chart versus Main-chart semantics
  - Multi-modal: cold (“may happen”) or hot (“must happen”)
- Abstract interpretation techniques can offer finite and effective representation of contracts
  - Non-computable objects are under- or over-approximated by computable ones
  - It is possible to prove consistency and compatibility for concrete contracts, based on corresponding abstractions (observers can only disprove properties)

Outline: Contracts and compositional methods for system design

- Where and why using contracts?
- Introduction to contracts
- Mathematical meta-theory of contracts
- Overview of concrete contract theories
- Application examples