Model Based
Software and Systems Engineering:

Elements of Seamless Development

Manfred Broy

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

DRAFT WHITE PAPER ON SYSTEMS AND SYSTEMS ENGINEERING DEFINITION

PREPARED BY THE INCOSE FELLOWS' INITIATIVE ON SYSTEMS ENGINEERING DEFINITION

STRAW MAN FOR A NEW DEFINITION OF SYSTEMS ENGINEERING

As a working premise and basis for discussion, we consider the merits of a short "straw man" definition of Systems Engineering as follows:

Systems Engineering seeks to understand societal needs for technology-enabled systems, services, and capabilities, synthesise holistic fit-for-purpose solutions, and facilitate their delivery and successful operation.

DRAFT WHITE PAPER ON SYSTEMS AND SYSTEMS ENGINEERING DEFINITION

PREPARED BY THE INCOSE FELLOWS' INITIATIVE ON SYSTEMS ENGINEERING DEFINITION

THE FOUR SYSTEMS OF INTEREST TO SYSTEMS ENGINEERING

In this White Paper we identify four "systems" of interest to Systems Engineering, as illustrated in Fig 2:

1. The "**Situation System**", otherwise known as the "context", "environment", "problem situation", or "wider system of interest (WSOI)";
2. The "**System that does Systems Engineering**", the project or enterprise charged with creating a new, or improving an existing, system to create some desired improvement in the "situation system";
3. The "**System that is Systems-Engineered**" in order to achieve the desired improvement, otherwise referred to as the "operational system", or the "system of interest (SOI)";
4. The "**System of Structured Information**", otherwise referred to as the System Architecture or System Model, that describes the other three systems, and the anticipated and actual results of inserting the third into the first – of deploying the operational system into its intended operational environment.

Interfaces and Systems

Manfred Broy

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Cyber-physical systems: key properties and challenges

- Physicality
  - real world awareness
  - real time
  - probability
  - …
- Connectivity
  - systems of systems
  - connected to cloud services
- Systems of systems
  - Sub-system decomposition
  - Service decomposition
- Interoperability
  - Service platforms
- Openness
  - security
- HMI
  - Human Centric Engineering
- Dynamic systems
  - Dynamic interfaces
  - Dynamic architectures
  - Dynamic change of behavior (adaptivity)
- Mobile systems
  - space awareness

Modeling Cyber-Physical Systems

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Modeling CPS

When modeling CPS we have capture following aspects:

- interaction – exchange of information/material
  - ◊ between system and its operational context
  - ◊ between sub-system within a system architecture
  - ◊ synchronization and orchestrations – protocols
- distribution – structuring systems in architectures with elements related to locations
- operational context – system's environment
- real time
- probability

To do that we have to use concepts such

- interfaces – scope and interaction
- state – state transition
- architecture – (de-)composition of systems into subsystems

What is a System

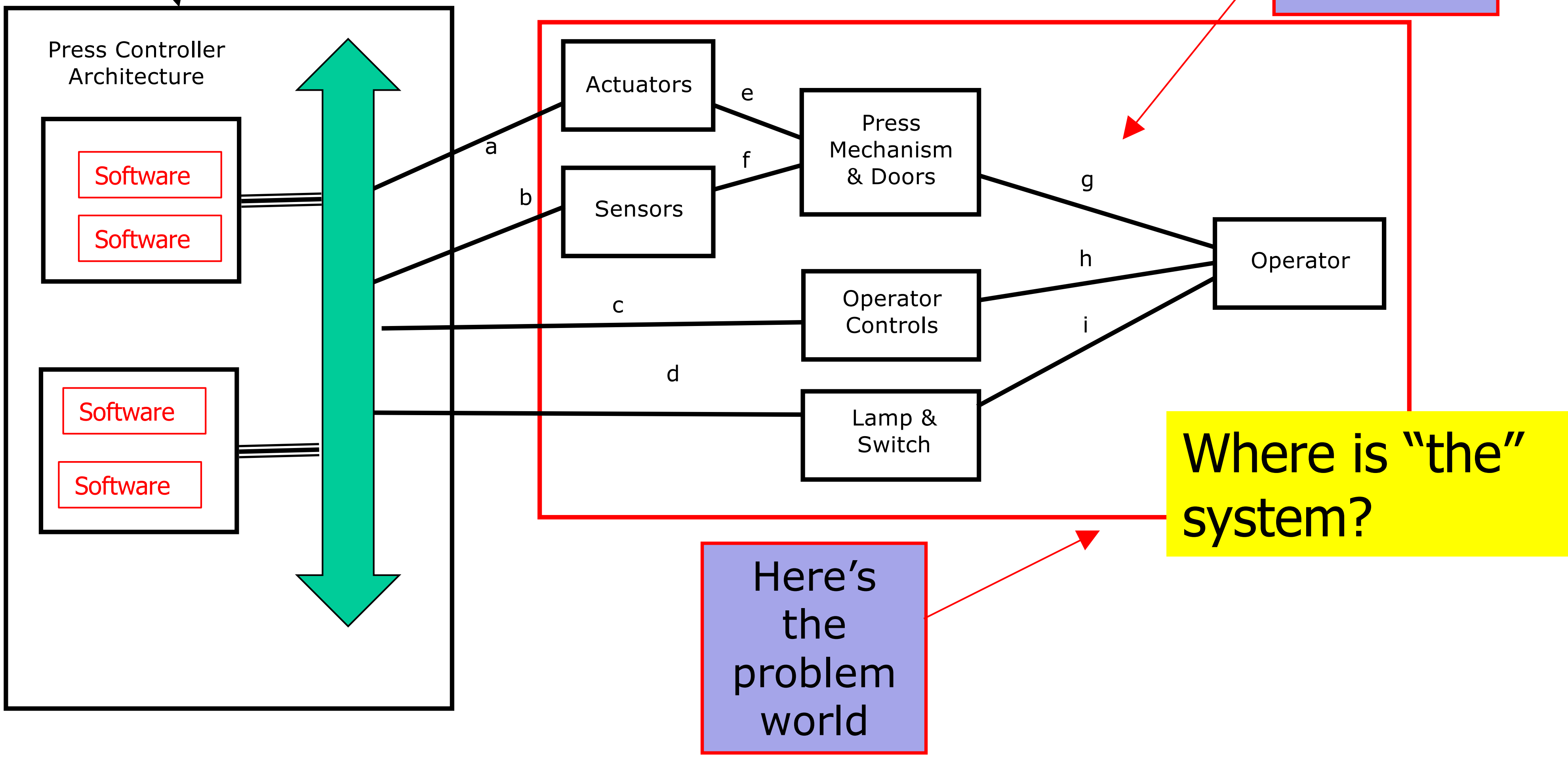
Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

A slide due to Michael Jackson

Here's the machine

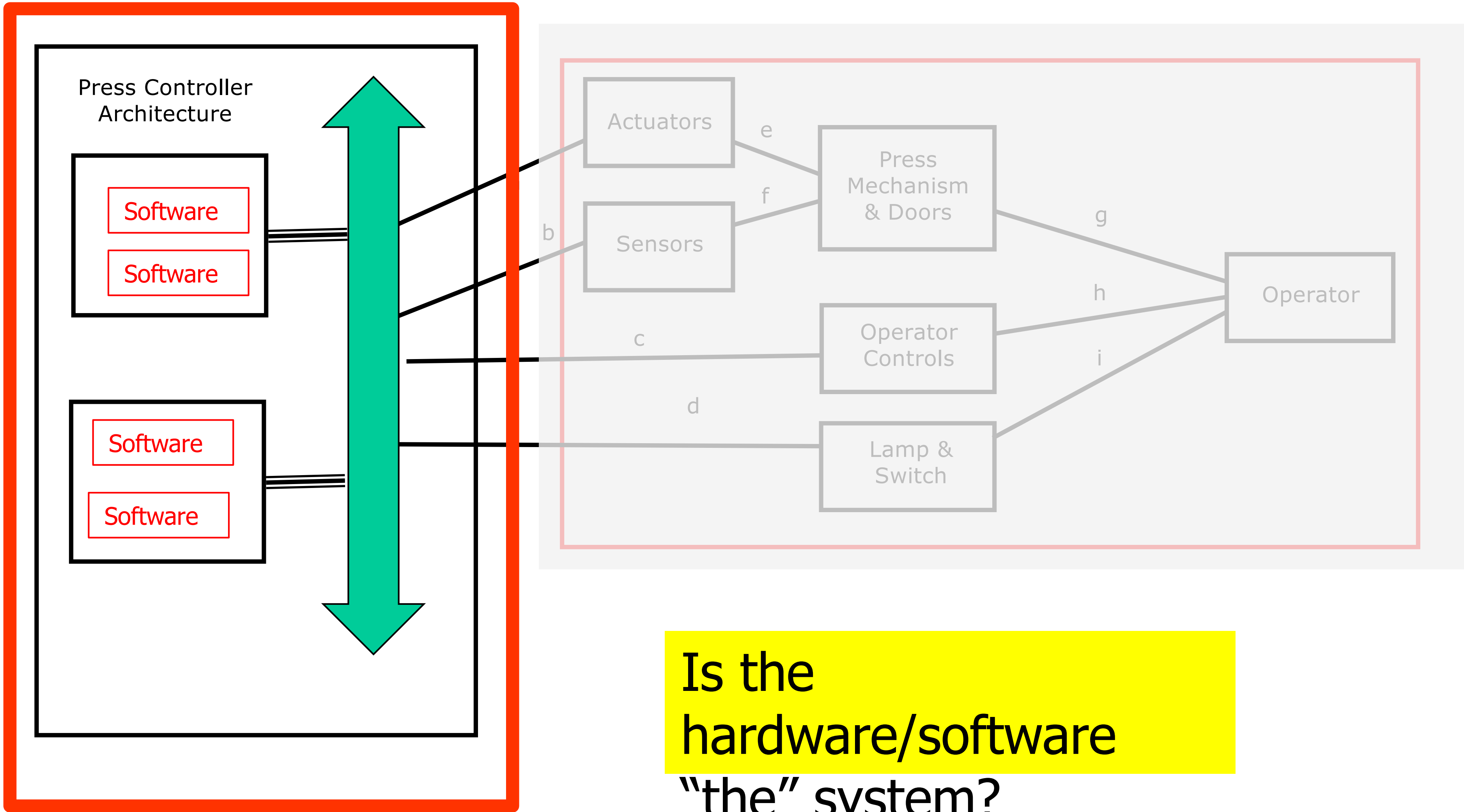
An industrial press system

Here's the user interface



Finding the scope

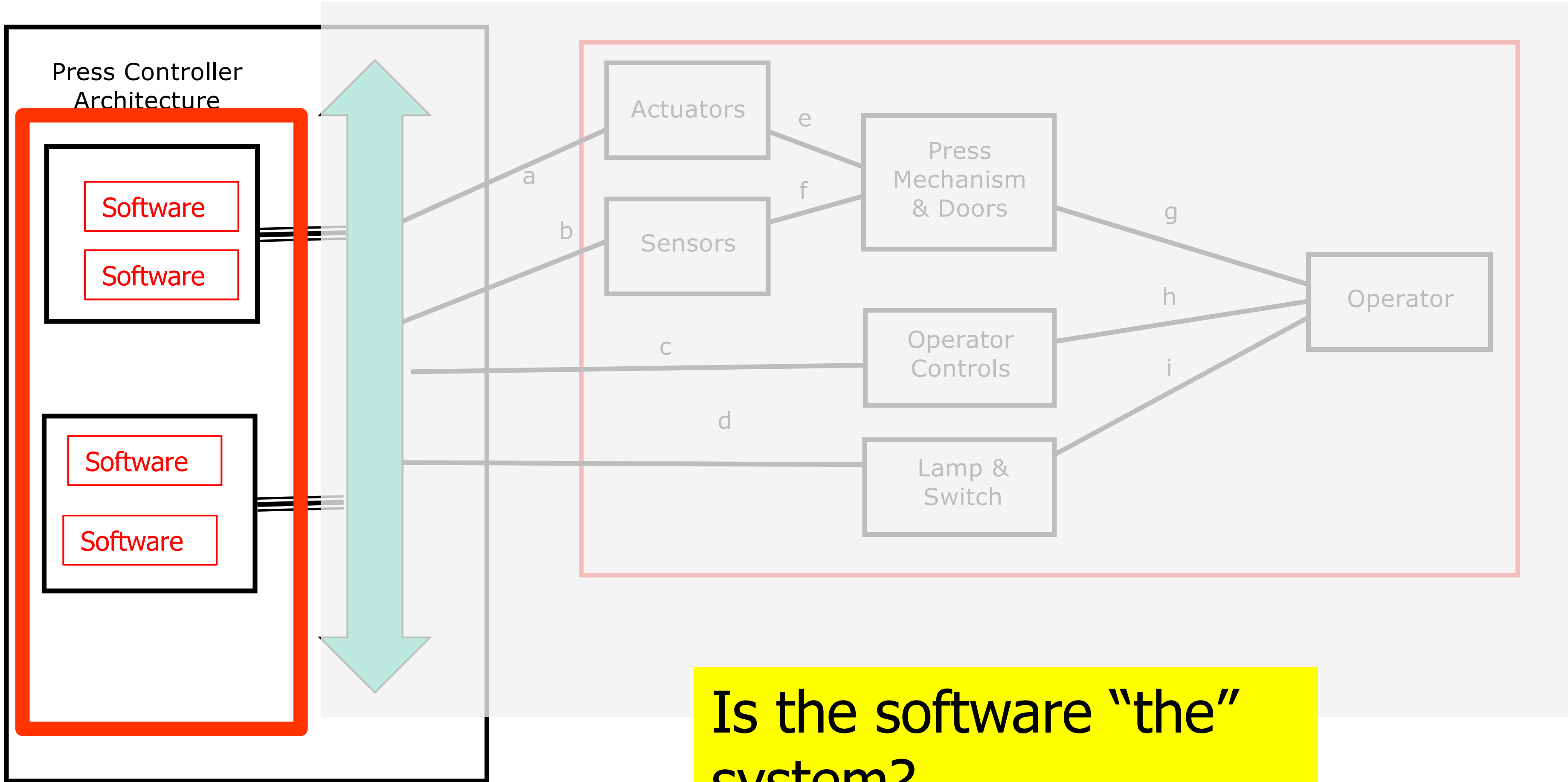
An industrial press system



Is the hardware/software "the" system?

Finding the scope

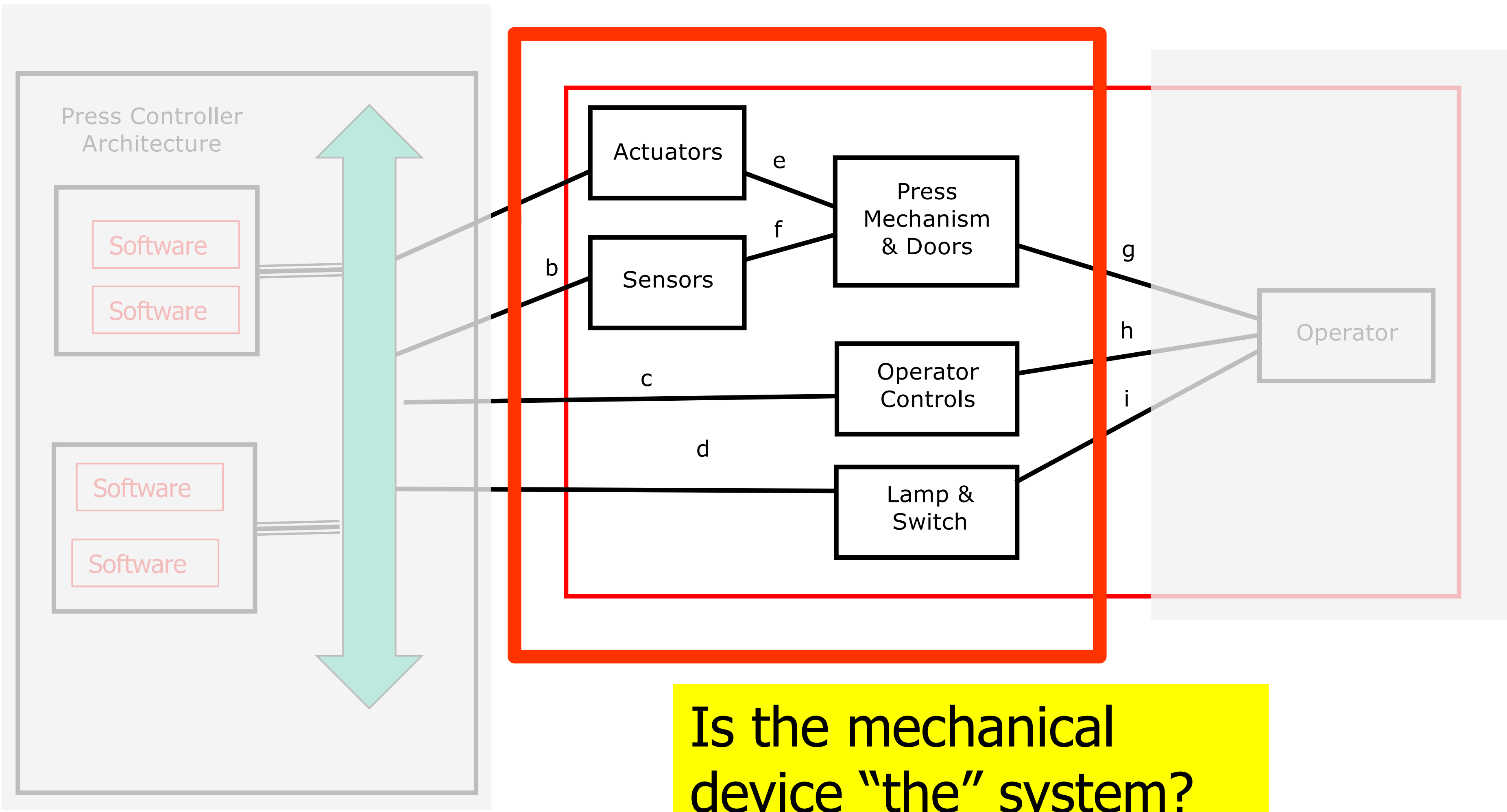
An industrial press system



Is the software "the" system?

Finding the scope

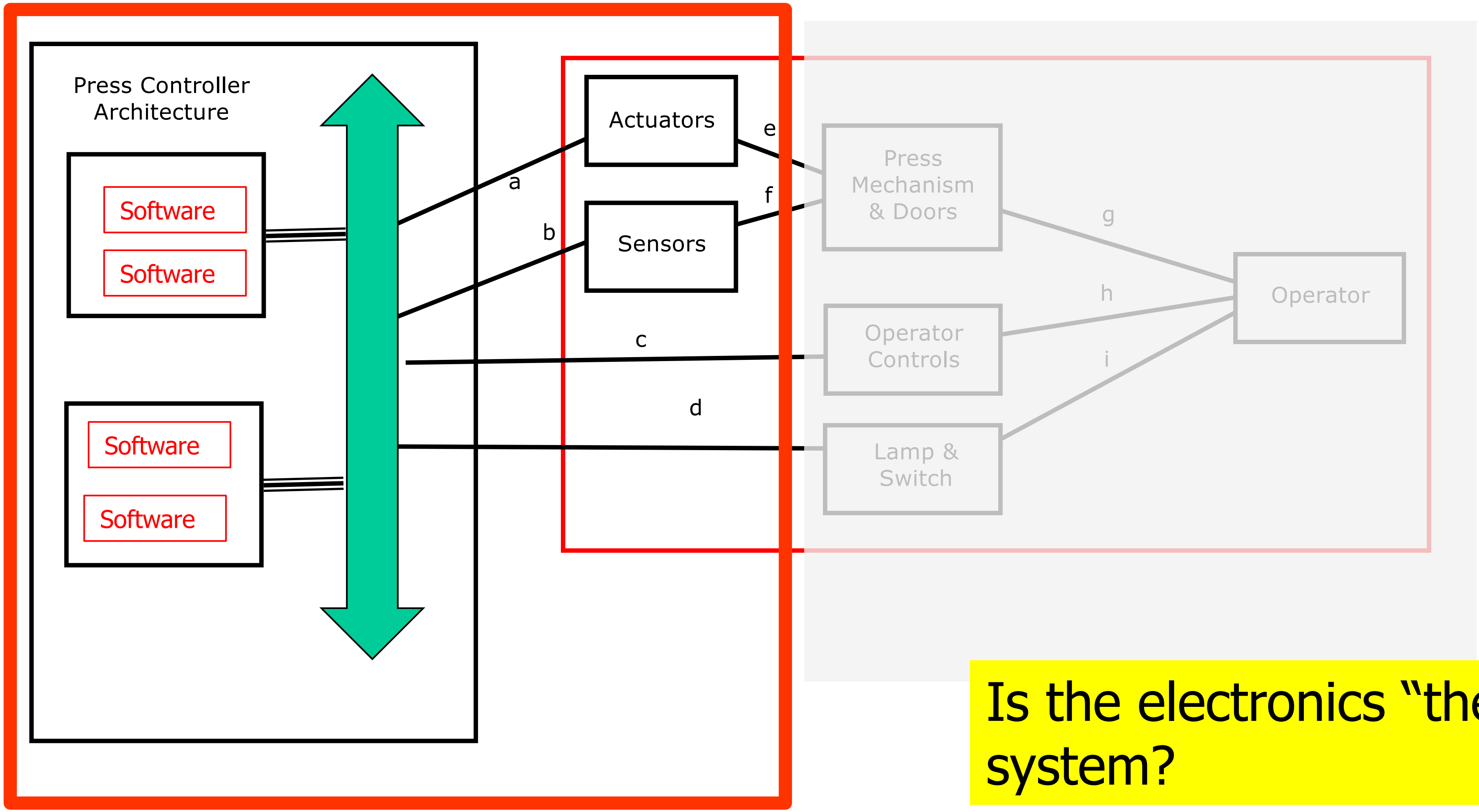
An industrial press system



Is the mechanical device "the" system?

Finding the scope

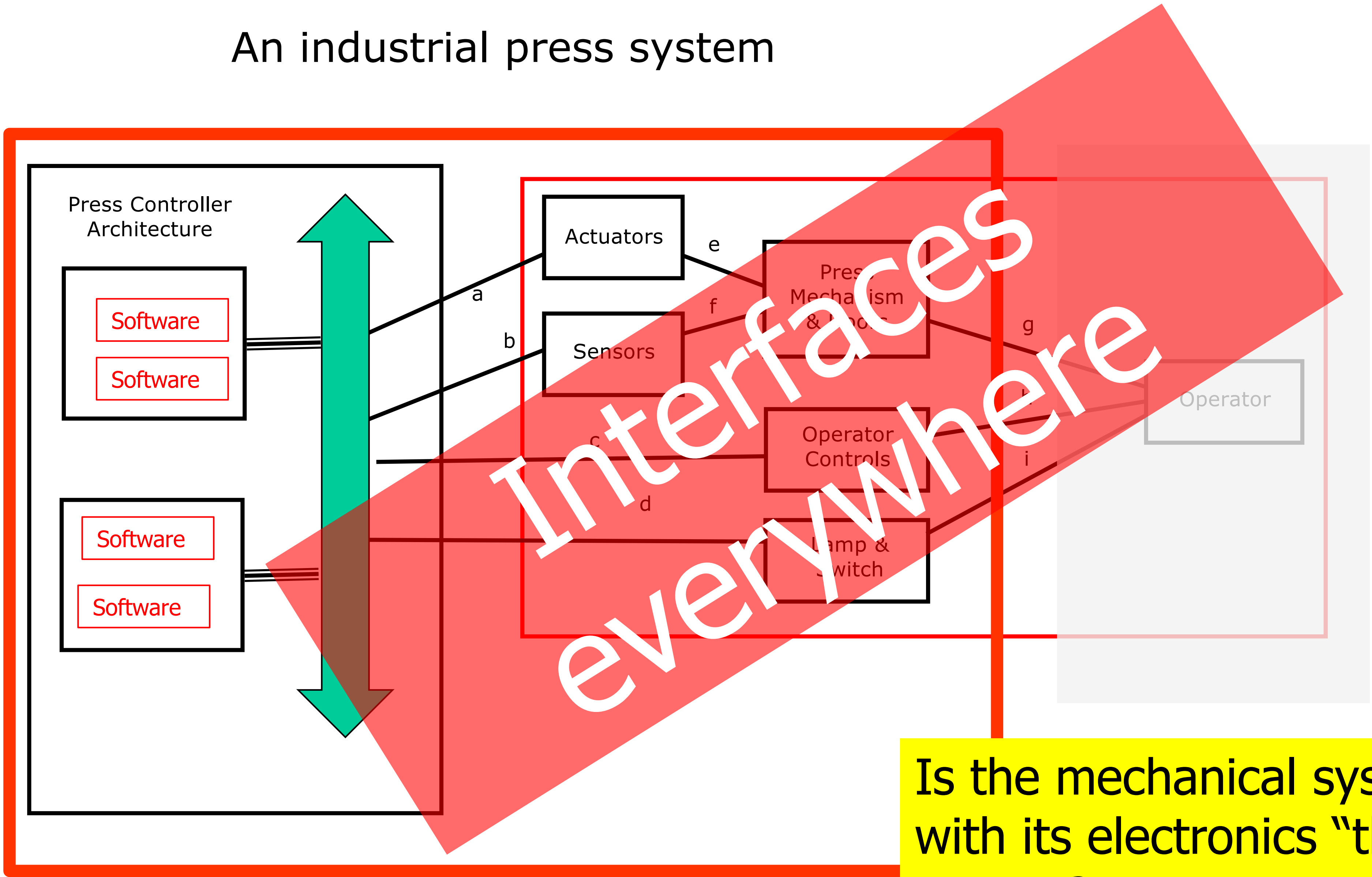
An industrial press system



Is the electronics "the" system?

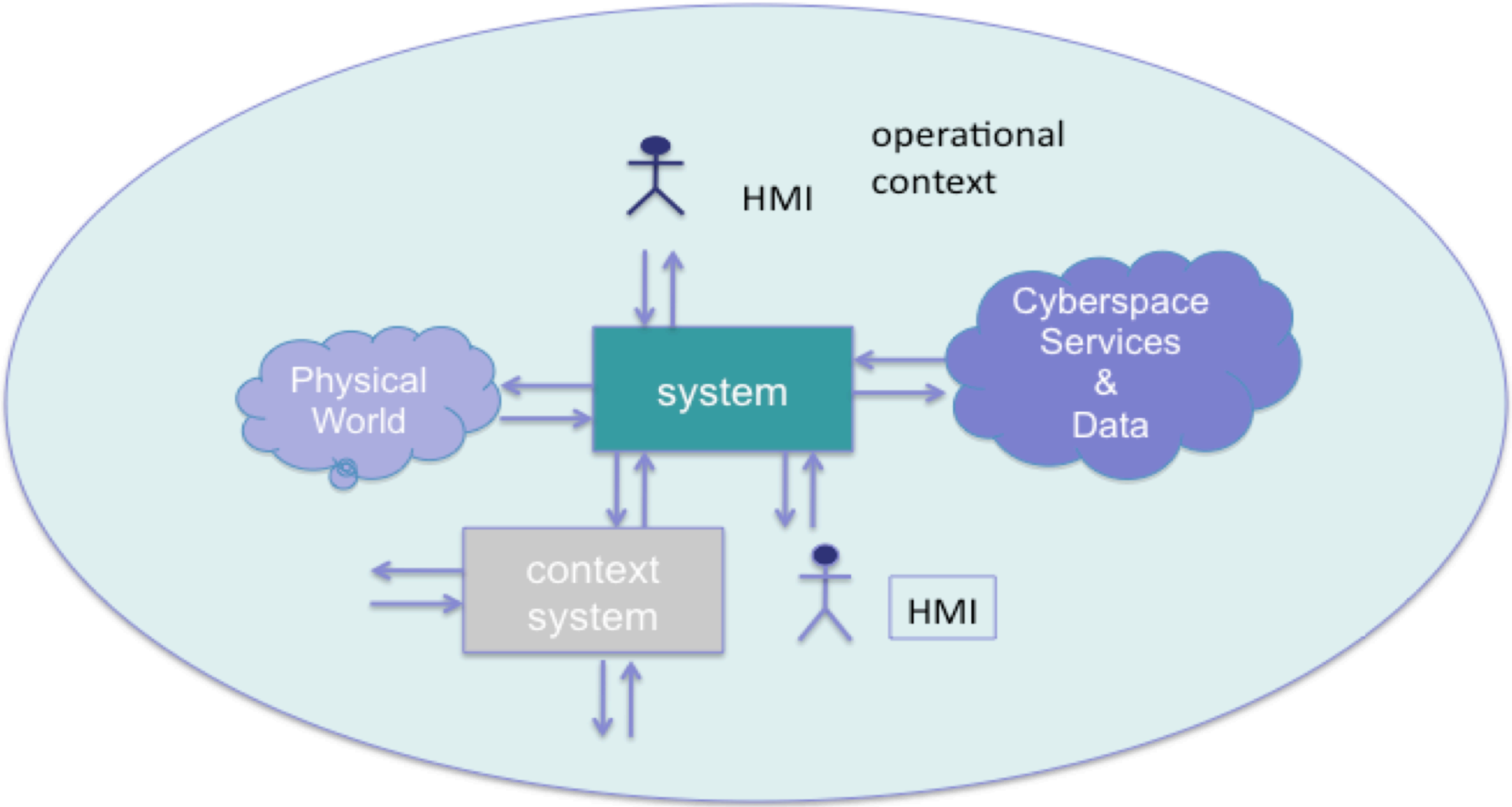
Finding the scope

An industrial press system



Is the mechanical system with its electronics “the” system?

System and its operational context



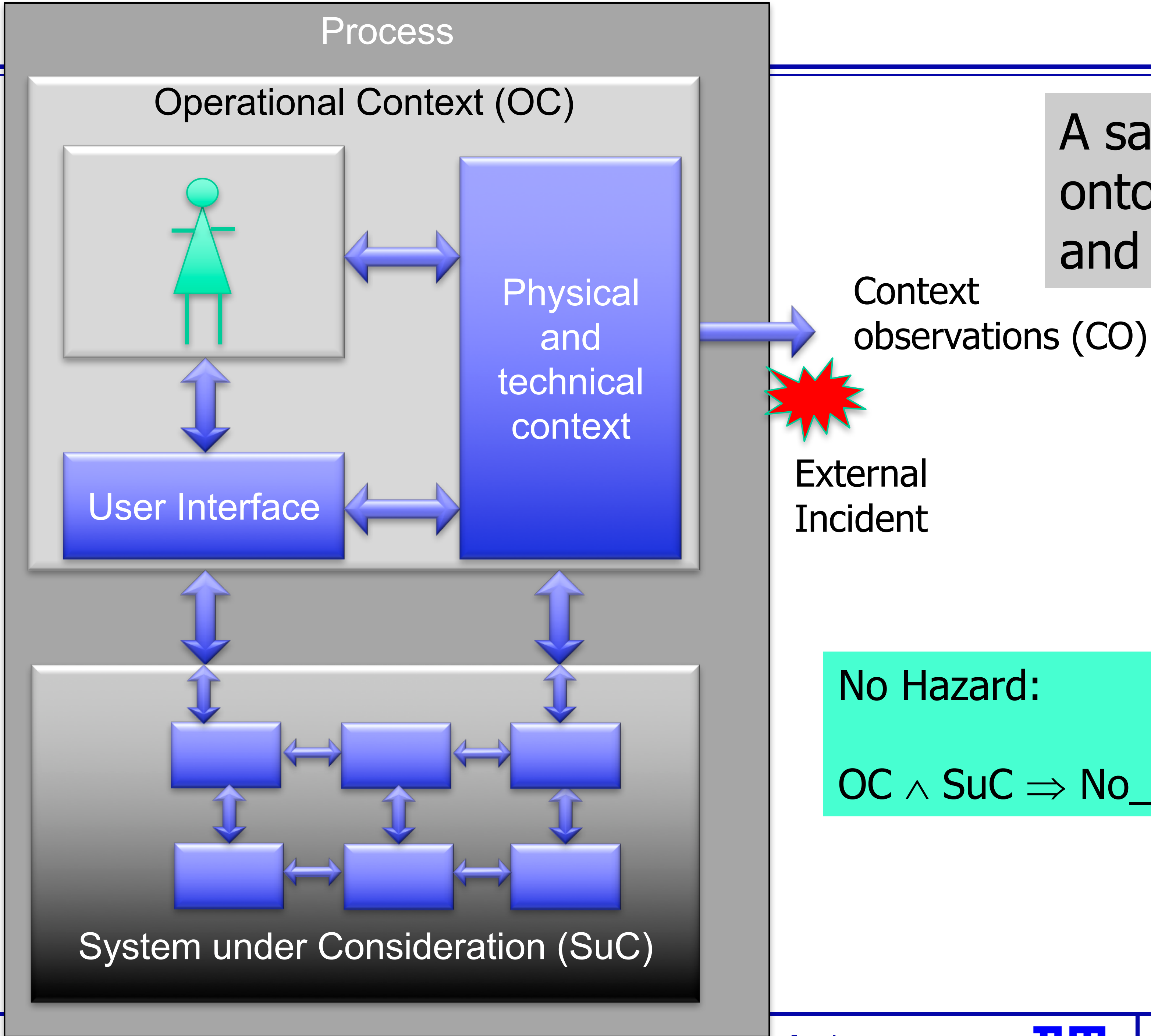
Basic System Notion: What is a discrete system (model)

A system has

- a system boundary that determines
  - what is part of the systems and
  - what lies outside (called its context)
- an interface (determined by the system boundary), which determines,
  - what ways of interaction (actions) between the system und its context are possible (static or syntactic interface)
  - which behavior the system shows from view of the context (interface behavior, dynamic interface, interaction view)
- a structure and distribution addressing internal structure, given
  - by its structuring in sub-systems (sub-system architecture)
  - by its states und state transitions (state view, state machines)
- quality profile
- the views use a data model
- the views may be documented by adequate models

System Views

- Operational Context View (OC)
  - ◊ Behavior of the operational context
- Interface View: System Interface Behavior (SIB)
  - ◊ Functional View: Interface Behavior
  - ◊ Functional features: hierarchy and feature interaction
- Interaction between OC and SIB:
  - ◊ Observable behavior: process
- Architectural View
  - ◊ Hierarchical decomposition in sub-systems
  - ◊ Sub-system behavior
- State View
  - ◊ State space
  - ◊ State transition



A safety view onto a system and its context

No Hazard:

OC $\wedge$ SuC $\Rightarrow$ No_Incident(CO)

Basic System Modeling Concepts:
Interface View:
Modeling Syntactic Interfaces and Interface Behavior

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Discrete systems - interfaces: the modeling theory

Sets of typed channels

$I = \{x_1 : T_1, x_2 : T_2, \ldots\}$

$O = \{y_1 : T'_1, y_2 : T'_2, \ldots\}$

syntactic interface

$(I \blacktriangleright O)$

data stream of type T

$STREAM[T] = \{IN\backslash\{0\} \rightarrow T^*\}$

valuation of channel set C

$IH[C] = \{C \rightarrow STREAM[T]\}$

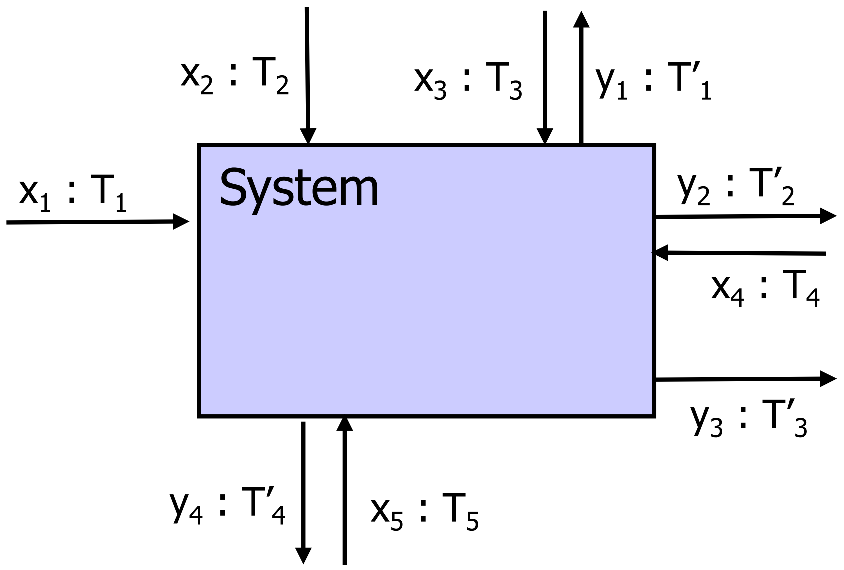
interface behaviour for syn. interface $(I \blacktriangleright O)$

$[I \blacktriangleright O] = \{IH[I] \rightarrow \wp(IH[O])\}$

interface specification

$p: I \cup O \rightarrow IB$

represented as interface assertion S
logical formula with channel names as variables for streams



See: M. Broy: A Logical Basis for Component-Oriented Software and Systems Engineering. The Computer Journal: Vol. 53, No. 10, 2010, S. 1758-1782

The Basic Behaviour Model: Timed Streams and Channels

C	set of channels
Type: $C \rightarrow TYPE$	type assignment
$x : C \rightarrow (\mathbb{N}\backslash\{0\} \rightarrow M^*)$	channel history for messages of type M
$\vec{C}$ or IH[C]	set of channel histories for channels in C

System interface behaviour - causality

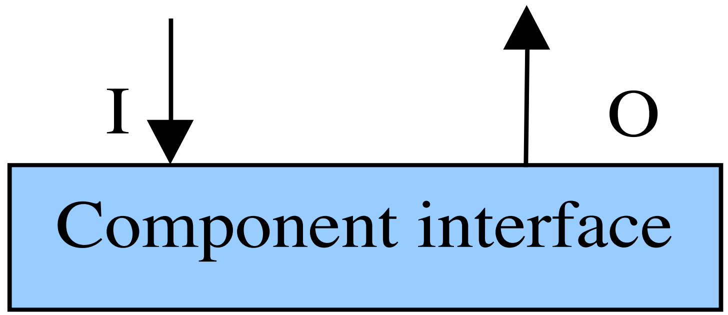
(I ▶ O) *syntactic interface* with set of input channels I and of output channels O

$F : \vec{I} \rightarrow \wp(\vec{O})$ *semantic interface* for (I ▶ O) with *timing property addressing strong causality* let x, z $\in \vec{I}$, y $\in \vec{O}$, t $\in \mathbb{N}$):

$$x{\downarrow}t = z{\downarrow}t \Rightarrow \{y{\downarrow}t{+}1: y \in F(x)\} = \{y{\downarrow}t{+}1: y \in F(z)\}$$

$x{\downarrow}t$ prefix of history x of length t

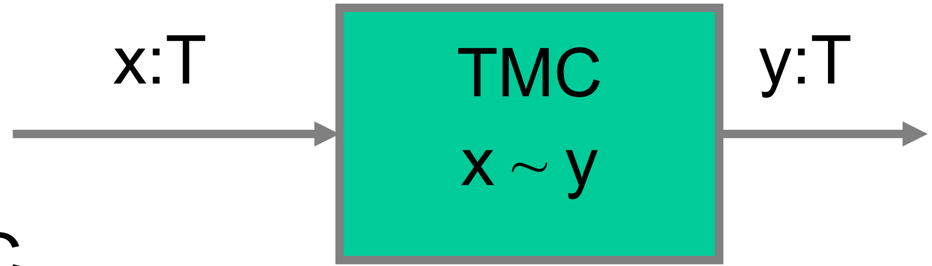
A system shows a total behavior



Specification of Interface Behavior

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Example: System interface specification



Spec name

A transmission component TMC

Input channel

TMC

in	x: T
out	y: T
x ~ y	

Output channel

Specifying interface assertion

$x \sim y \equiv (\forall\ m \in T: m\#x = m\#y)$

See: M. Broy, K. Stølen: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer 2001

Verification: Proving properties about specified systems

From the interface assertions we can prove

- Safety properties

$$m\#y > 0 \land y \in TMC(x) \Rightarrow m\#x > 0$$

- Liveness properties

$$m\#x > 0 \land y \in TMC(x) \Rightarrow m\#y > 0$$

Verification: adding and taking advantage of causality

From the interface assertion we can derive by causality

$$\forall\ m \in T: y \in TMC(x) \Rightarrow \forall\ t \in Time: m\#(y\downarrow t+1) \leq m\#(x\downarrow t)$$

Specification:

$$y \in TMC(x) \Rightarrow (\forall\ m \in T: m\#x = m\#y)$$

Strong causality:

$$x\downarrow t = z\downarrow t \Rightarrow \{y\downarrow t+1: y \in TMC(x)\} = \{y\downarrow t+1: y \in TMC(z)\}$$

From which we deduce the hypothesis by choosing z such that

$$\forall\ m \in T: m\#(z\uparrow t) = 0$$

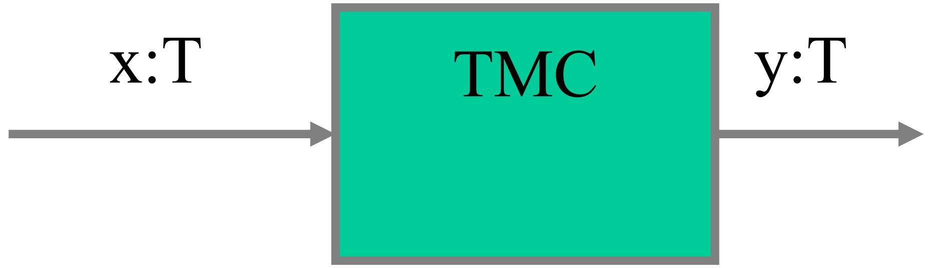
Interfaces and Systems:

Timing

Manfred Broy

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Example: TMC with Timing Restrictions



TMC

in x: T

out y: T

$\forall\, t \in IN: \forall\, m \in T:$

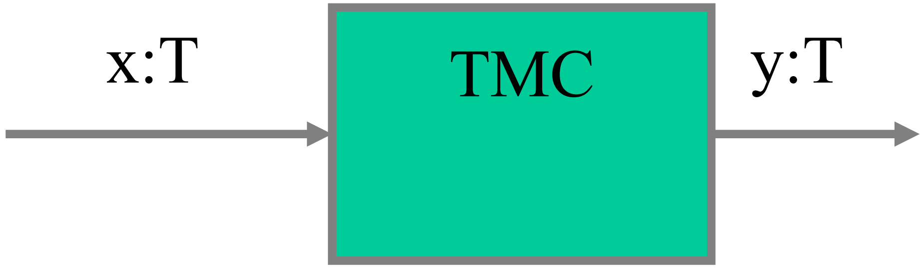
$m\#(y{\downarrow}t+delay) \le m\#(x{\downarrow}t) \le m\#(y{\downarrow}t+delay+deadline)$

Extending the Model of Interface Behavior:

Probabilistic System Interface Models

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Example:
TMC with Probability Restrictions



TMC

in x: T

out y: T

$\forall$ t $\in$ IN: $\forall$ m $\in$ T:

$$\mathbf{P}(m\#(x\downarrow t) \leq m\#(y\downarrow t+delay+deadline)) \geq 0.8$$

Discrete systems: the modeling theory - probability

Sets of typed channels

$I = \{x_1 : T_1, x_2 : T_2, \ldots \}$

$O = \{y_1 : T'_1, y_2 : T'_2, \ldots \}$

syntactic interface

$(I \blacktriangleright O)$

data stream of type T

$STREAM[T] = \{IN \backslash \{0\} \rightarrow T^*\}$

valuation of channel set C

$IH[C] = \{C \rightarrow STREAM[T]\}$

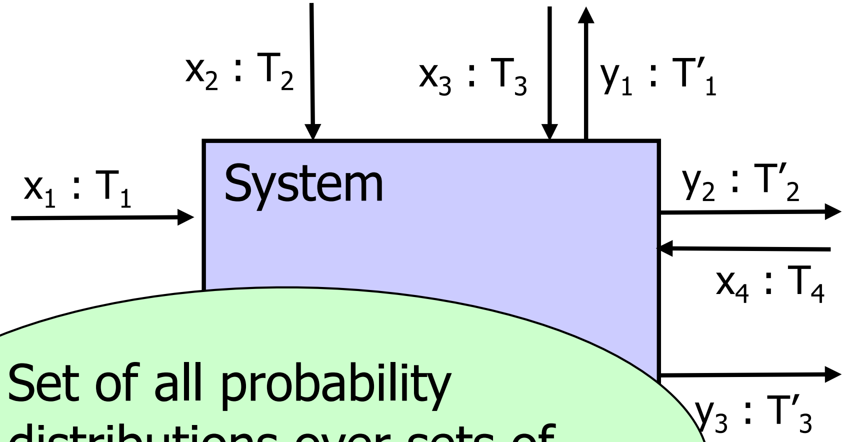
interface behaviour for syn. interface $(I \blacktriangleright O)$

$[I \blacktriangleright O] = \{IH[I] \rightarrow PD[\wp(IH[O])]\}$

interface specification

$p: I \cup O \rightarrow IB$

represented as interface assertion S
logical formula with channel names as variables for streams



Set of all probability distributions over sets of output histories

See: P. Neubeck: A Probabilitistic Theory of Interactive Systems. PH. D. Dissertation, Technische Universität München, Fakultät für Informatik, December 2012

Extensions of the model: Probability

- Probabilistic views
  - ◊ Interface behavior: a probability distribution is given for the set of possible histories
  - ◊ Architectural view: probability distributions for the sub-systems of the architecture
  - ◊ State view: a probability distribution is given for the set of possible state transitions
- Then the model covers
  - ◊ certain "non-functional properties" (safety, reliability, …)
  - ◊ Example: integrated fault trees

See: P. Neubeck: A Probabilitistic Theory of Interactive Systems. PH. D. Dissertation, Technische Universität München, Fakultät für Informatik, December 2012

Architecture and State

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

From the external to the internal view

- So far we treated the interface view.
- Now we move forward to the internal view!

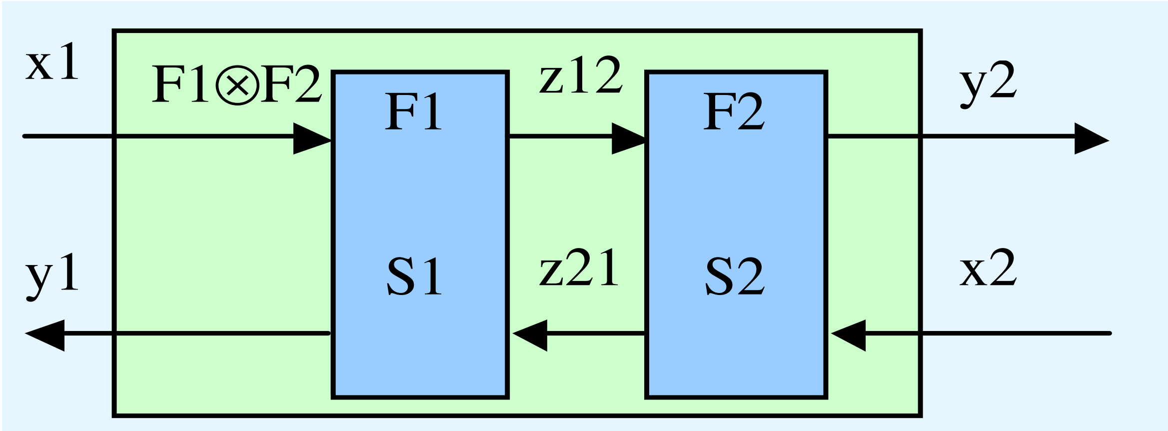
A system has

- a system boundary that determines
  - ◊ what is part of the systems and
  - ◊ what lies outside (called its context)
- an interface (determined by the system boundary), which determines,
  - ◊ what ways of interaction (actions) between the system und its context are possible (static or syntactic interface)
  - ◊ which behavior the system shows from view of the context (interface behavior, dynamic interface, interaction view)
- a structure and distribution addressing internal structure, given
  - ◊ by its structuring in sub-systems (sub-system architecture)
  - ◊ by its states und state transitions (state view, state machines)
- quality profile
- the views use a data model
- the views may be documented by adequate models

Architecture - Structure:
Composition and Decomposition

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Modularity: Rules of compositions for interface specs



F1

in	x1, z21: T
out	y1, z12: T
S1	

F2

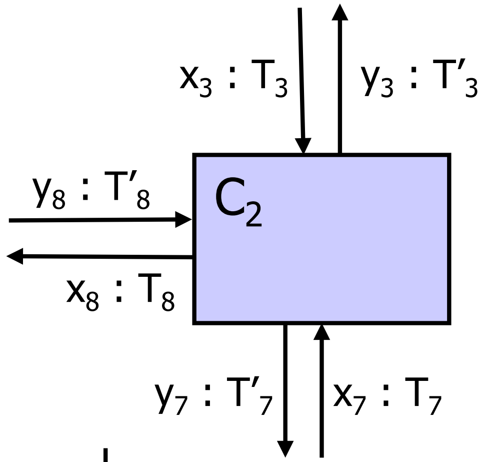
in	x2, z12: T
out	y2, z21: T
S2	

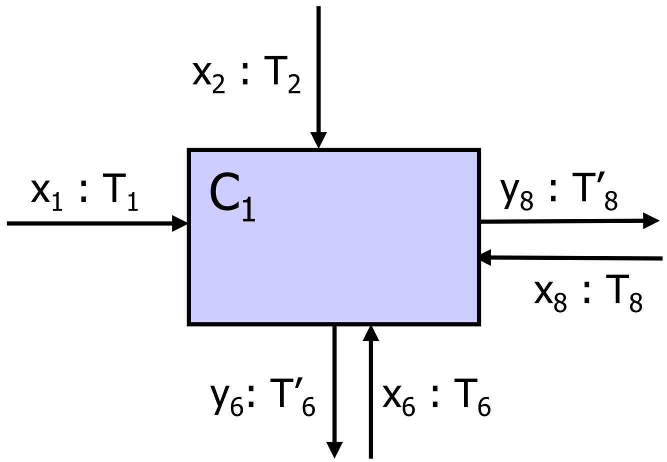
F1⊗F2

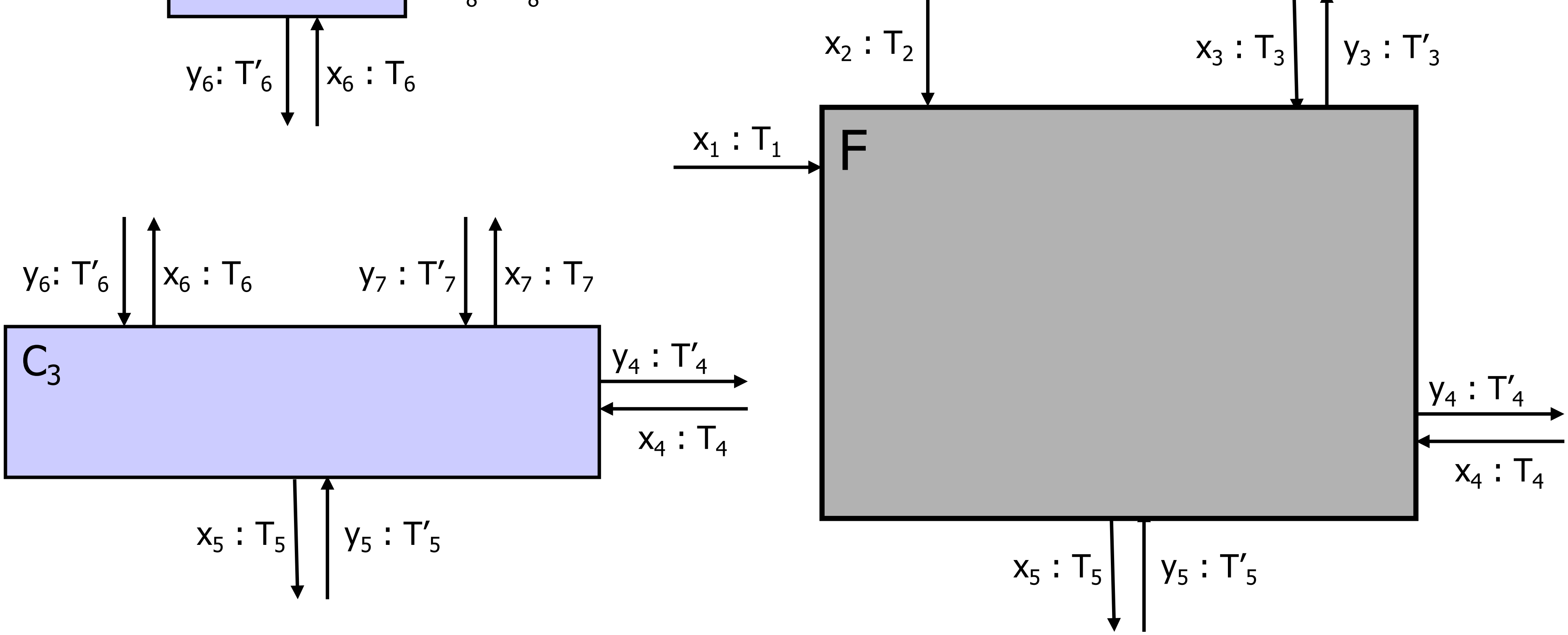
in	x1, x2: T
out	y1, y2: T
$\exists$ z12, z21: S1 $\wedge$ S2	

Architecture

- Composition $F = C_1 \otimes C_2 \otimes C_3$

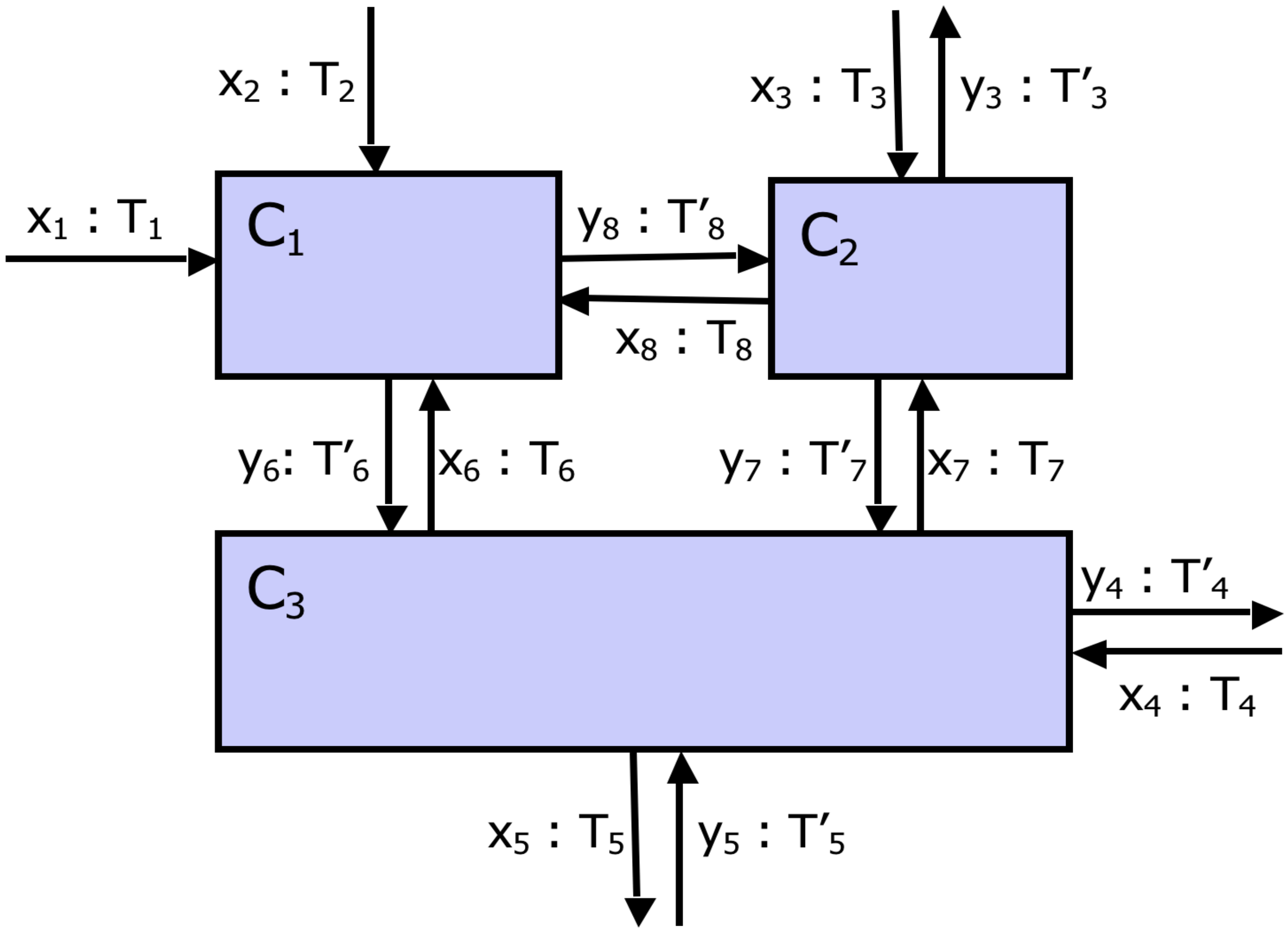




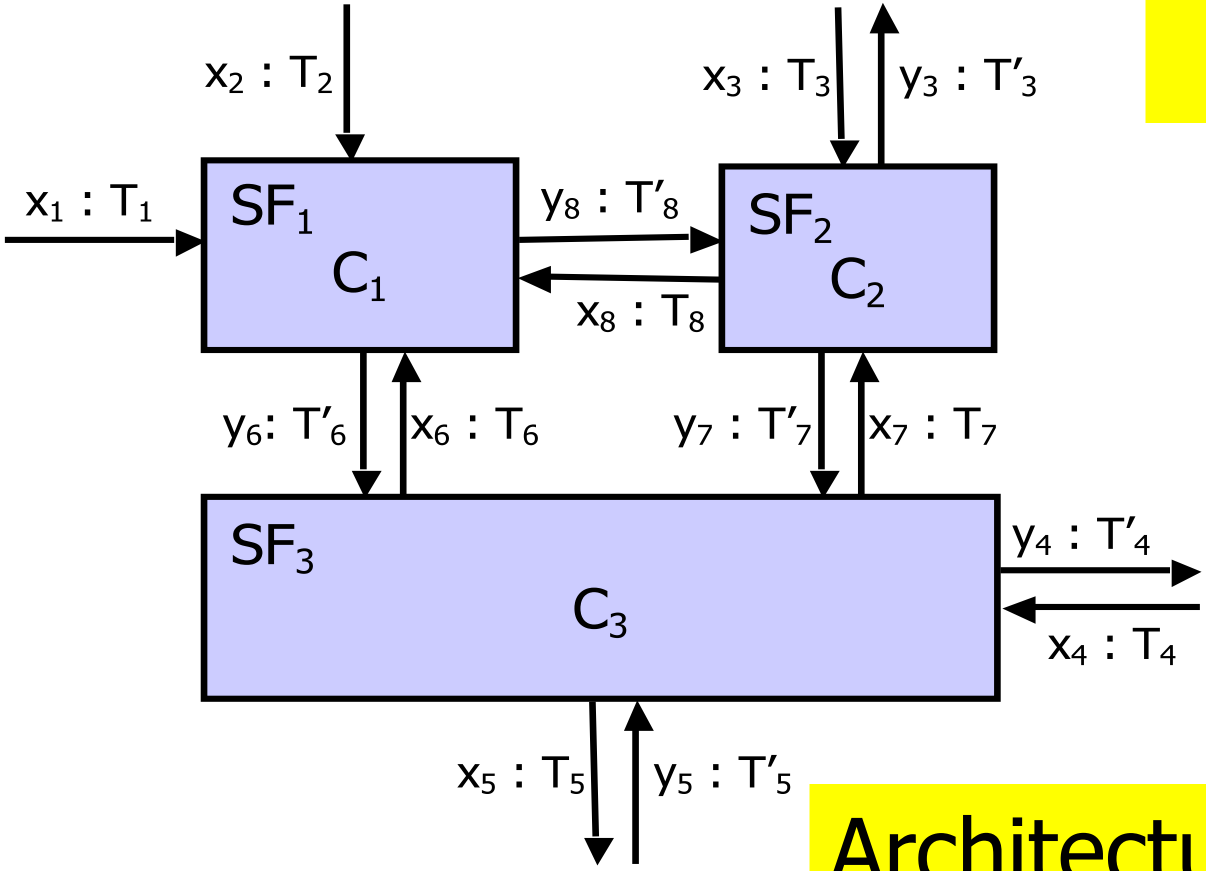


Forming Architectures

$C_1 \otimes C_2 \otimes C_3$



Forming Architectures



Architecture Behaviour

$SF_1 \otimes SF_2 \otimes SF_3$

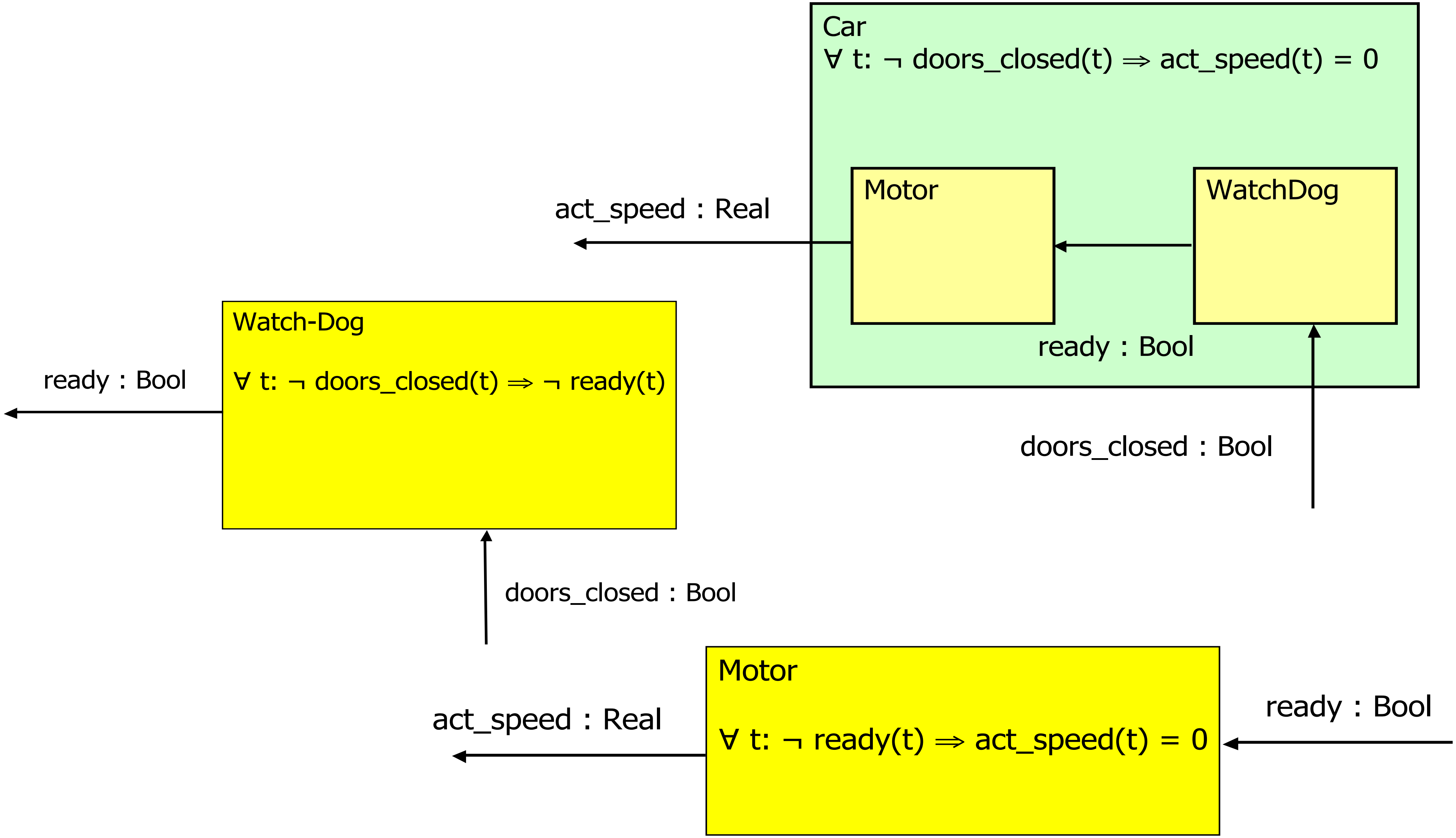
Architecture Spec

$C_1 \wedge C_2 \wedge C_3$

Architecture Correctness

$C_1 \wedge C_2 \wedge C_3 \Rightarrow \text{SysSpec}$

Specification of a Car´s Architecture



Implementation: Systems as State Machines

The State View

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

System and States

- Systems have states
- A state is an element of a state space
- We characterize state spaces by
  - a set of state attributes together with their types
  - Example:
    State space for a three dimensional position:
    x1, x2, x3: Var Real
    State space for a cruise comtrol:
    speed, set_speed: Var Real, engine_on, activated: Var Bool

- The behaviour of a system with states can be described by its state transitions

State model for systems/components

A system can be implemented by a state Machine

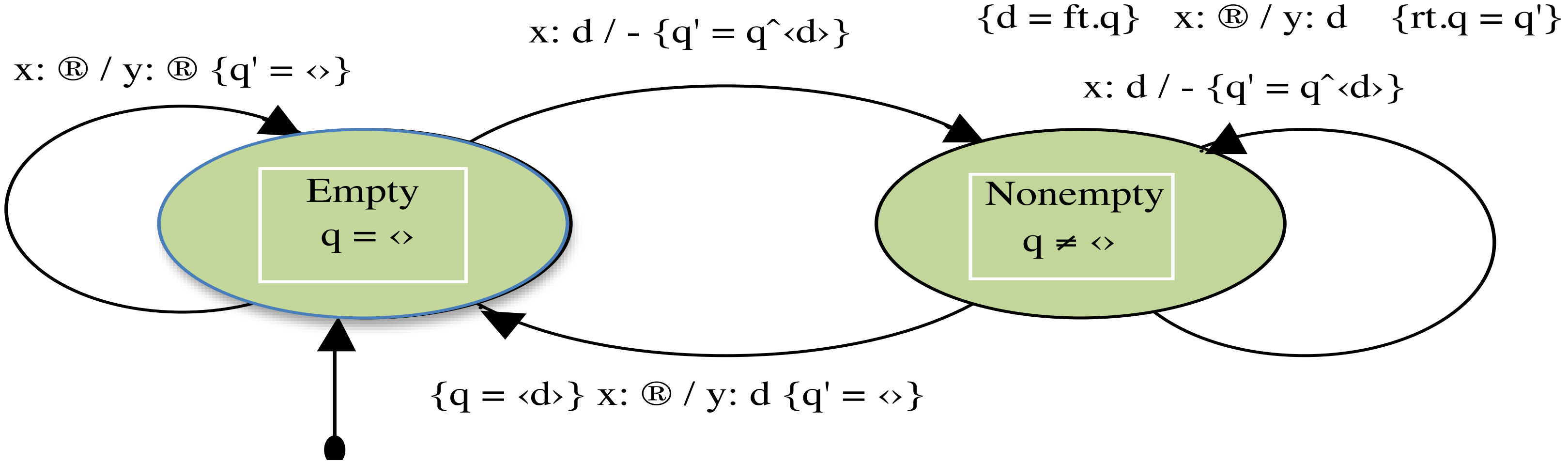
M. Broy: From States to Histories: Relating States and History Views onto Systems. In: T. Hoare, M. Broy, R Steinbrüggen (eds.): Engineering Theories of Software Construction. Springer NATO ASI Series, Series F: Computer and System Sciences, Vol. 180, IOS 2001, 149-186

Σ set of states, initial state $\sigma \subseteq \Sigma$

State transition function:

$$\Delta: (\Sigma \times (I \to M^*)) \to \wp(\Sigma \times (O \to M^*))$$

State transition diagram:



State Machines in general

A state machine (Δ, Λ) consists of

- a set Σ of states - the state space
- a set $\Lambda \subseteq \Sigma$ of initial states
- a state transition function or relation Δ
  - ◊ in case of a state machine with input/output:

    events (inputs E) trigger the transitions and events (outputs A) are produced by them respectively:

    $$\Delta : \Sigma \times E \to \Sigma \times A$$

    in the case of nondeterministic machines:

    $$\Delta : \Sigma \times E \to \wp(\Sigma \times A)$$

- Given a syntactic interface with sets I and O of input and output channels:

$$E = I \to M^*$$

$$A = O \to M^*$$

Computations of a State Machine with Input/Output

A state machine (Δ, Λ) defines for each initial state

$$\sigma_0 \in \Lambda$$

and each sequence of inputs

$$e_1, e_2, e_3, \ldots \in E$$

a sequence of states

$$\sigma_1, \sigma_2, \sigma_3, \ldots \in \Sigma$$

and a sequence of outputs

$$a_1, a_2, a_3, \ldots \in A$$

through

$$(\sigma_{i+1}, a_{i+1}) \in \Delta(\sigma_i, e_{i+1})$$

Computations of a State Machine with Input/Output

In this manner we obtain computations of the form

$$\sigma_0 \xrightarrow{a_1 / b_1} \sigma_1 \xrightarrow{a_2 / b_2} \sigma_2 \xrightarrow{a_3 / b_3} \sigma_3 \quad \ldots$$

For each initial state σ0 ∈ Σ we define a function

$$F_{\sigma 0} : \vec{I} \rightarrow \wp(\vec{O})$$

with

$$F_{\sigma 0}(x) = \{y: \exists\ \sigma_i: \sigma 0 = \sigma_0 \wedge \forall\ i \in IN: (\sigma_{i+1}, y_{i+1}) = \Delta(\sigma_i, x_{i+1})\}$$

$F_{\sigma 0}$ denotes the interface behavior of the transition function Δ for the initial state σ0.

Furthermore we define

$$Abs((\Delta, \Lambda)) = F_\Lambda$$

where:

$$F_\Lambda(x) = \{y \in F_\sigma(x) : y \in F_\sigma(x) \wedge \sigma \in \Lambda\}$$

F_Λ is called the interface behavior of the state machine (Δ, Λ) .

Moore Machines

- A Mealy machine (Δ, Λ) with

$$\Delta : \Sigma \times E \rightarrow \wp(\Sigma \times A)$$

  is called Moore machine if for all states $\sigma \in \Sigma$ and inputs $e \in E$ the set

$$out(\sigma, e) = \{a \in A: (\sigma, a) = \Delta(\sigma, e) \}$$

  does not depend on the input e but only on state σ.

- Formally: then for all $e, e' \in E$ we have

$$out(\sigma, e) = out(\sigma, e')$$

Theorem: If is (Δ, Λ) a Moore machine the F_Λ is strong causal.

Constructing state machines

- Specification:
  - ◊ Specify the syntactic interface
  - ◊ Specify the interface behavior (say by an interface assertion)
- Construction:
  - ◊ Construct the state space: define the attributes and their data types
  - ◊ Define the state transitions (e.g.: choose control states and state transitions: labeled state transition diagram)
- Verification:
  - ◊ Prove that state machine shows the specified interface behavior

Interface Abstraction for State Machines

- For a given state machine with input and output we define the interface through
  - ◊ its syntactic interface (signature)
  - ◊ its interface behavior

- We call the step from the state machine to its interface the interface abstraction.

Verification/derivation of interface assertions for state machines

- similar to program verification (find an invariant)
- needs sophisticated techniques

Observable Equivalence

- Two systems modelled by state machines

  $(\Delta 1, \Lambda 1)$ and $(\Delta 2, \Lambda 2)$

  are observably equivalent iff they fulfil the equation

  $$\text{Abs}((\Delta 1, \Lambda 1)) = \text{Abs}((\Delta 2, \Lambda 2))$$

Conclusion Systems as State Machines

- Each state machines defines an interface behaviour
- Each interface behaviour represents a state machine
- State machines can be described
  - ◊ mathematically by their state transition function
  - ◊ graphically by state machine diagrams
  - ◊ structured by state transition tables
  - ◊ by programs
- State machines define a kind of operational semantics
- Systems given by state machines can be simulated
- From state machines we can generate code
  - ◊ state machines can represent implementations
- From state machines we can generate test cases

Composition of the two state machines

Consider Moore machines $M_k = (\Delta_k, \Lambda_k)$ (k = 1, 2):

$$\Delta_k: \Sigma_k \times (I_k \to M^*) \to \wp(\Sigma_k \times (O_k \to M^*))$$

We define the composed state machine

$$\Delta: \Sigma \times (I \to M^*) \to \wp(\Sigma \times (O \to M^*))$$

as follows

$$\Sigma = \Sigma_1 \times \Sigma_2$$

for $x \in I$ and $(s_1, s_2) \in \Sigma$ we define:

$$\Delta((s_1, s_2), x) = \{((s_1', s_2'), z|O): \; x = z|I \wedge \forall\, k: (s_k', z|O_k) \in \Delta_k(s_k, z|I_k)\}$$

This definition is based on the fact that we consider Moore machines.

We write

$$\Delta = \Delta_1 \,||\, \Delta_2$$
$$M = M_1 \,||\, M_2 = (\Delta_1 \,||\, \Delta_2, \Lambda_1 \times \Lambda_2)$$

An example of an essential property ...

Interface abstraction distributes for state machines over composition

$$\text{Abs}((\Delta 1, \sigma 1) \mid\mid (\Delta 2, \sigma 2)) =$$

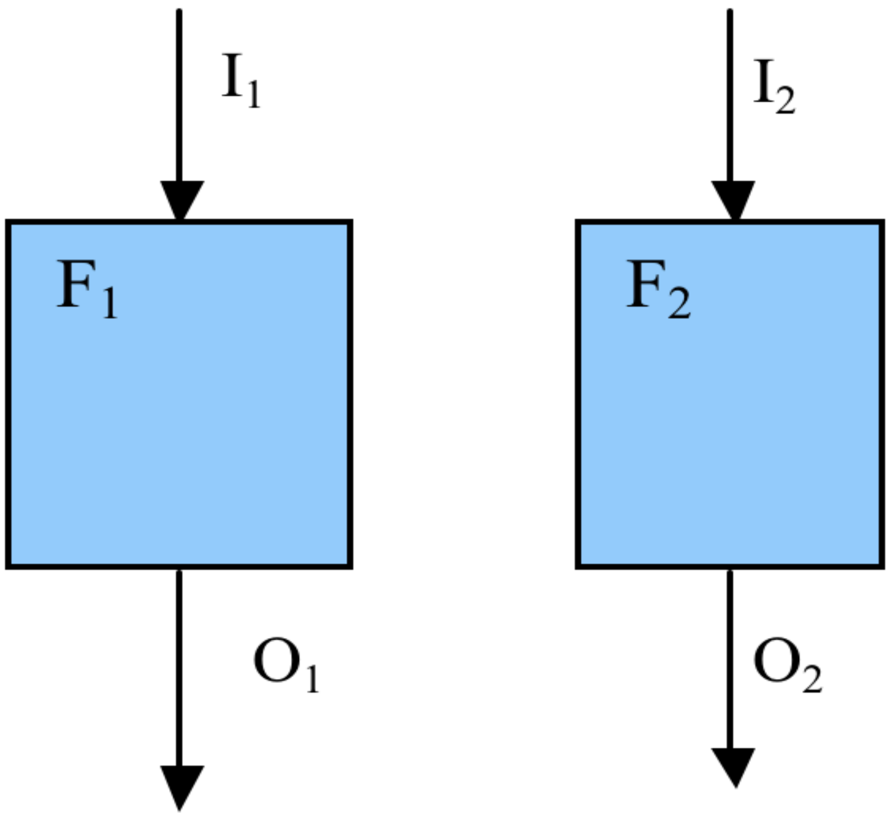
$$\text{Abs}((\Delta 1, \sigma 1)) \otimes \text{Abs}((\Delta 2, \sigma 2))$$

Functional View: Functional Decomposition

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Combining Functions

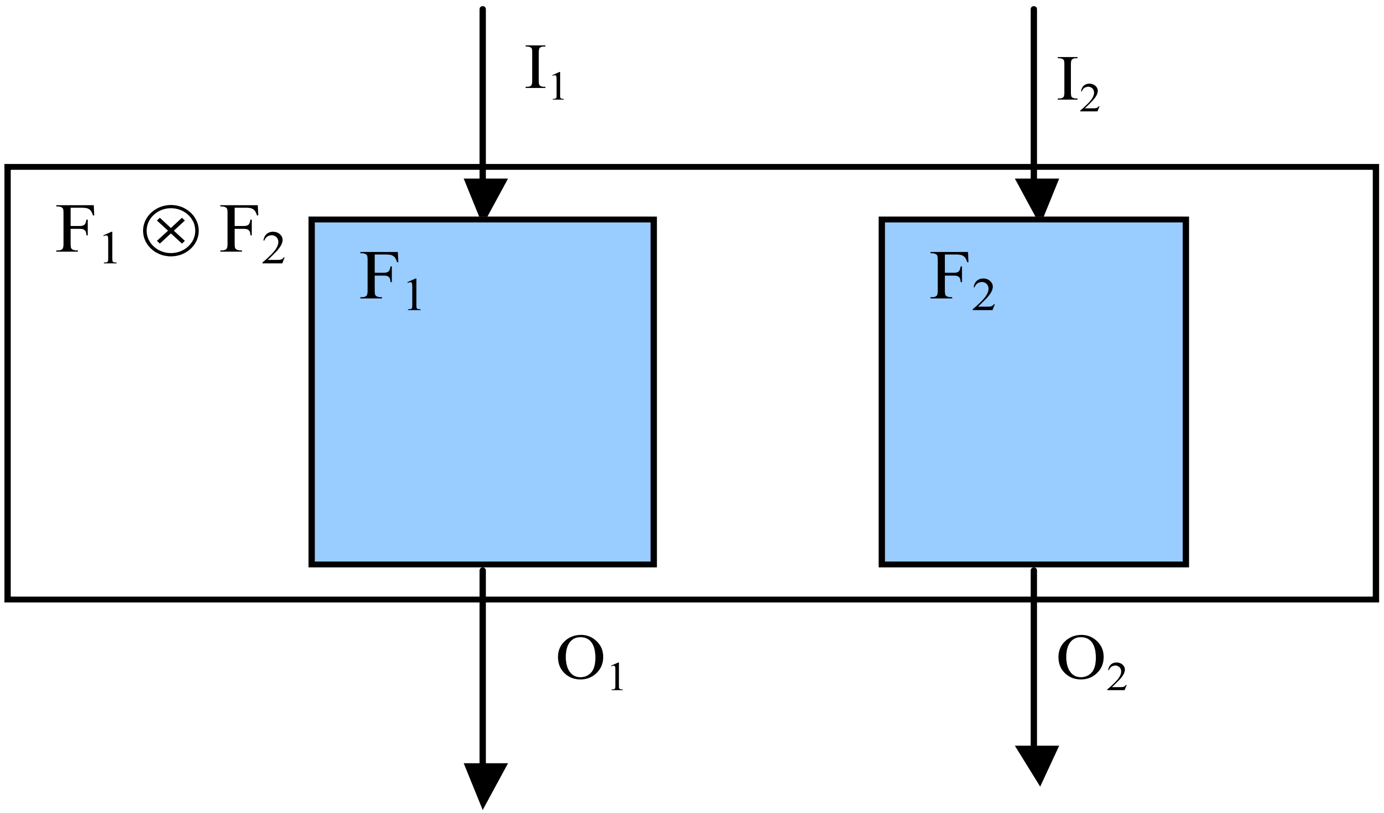
Given two functions F_1 and F_2 in isolation



We want to combine them into a function $F_1 \otimes F_2$

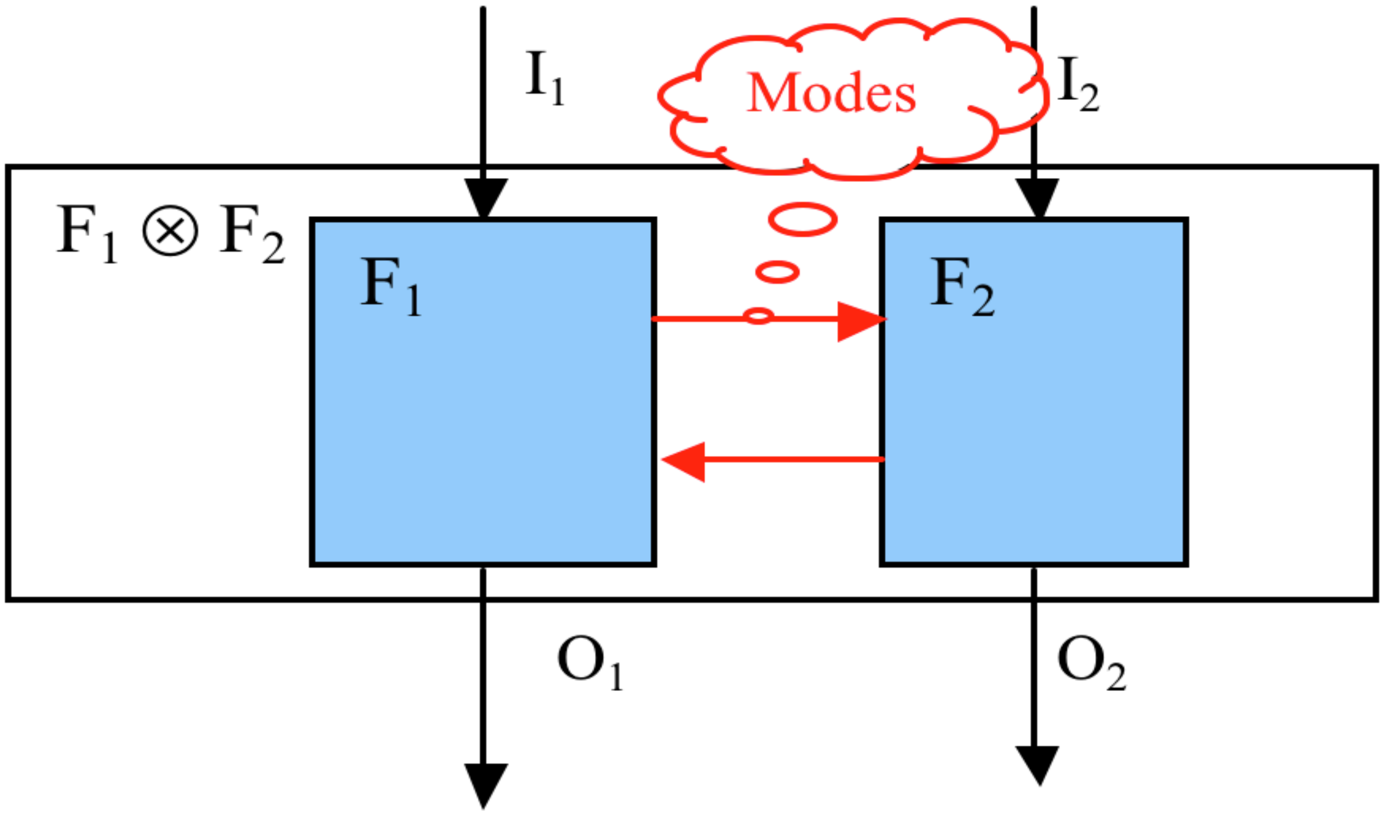
Combining Functions

Their isolated combination



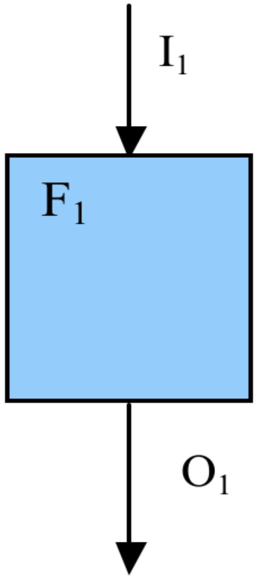
Combining Functions

If services F_1 and F_2 have feature interaction we get:

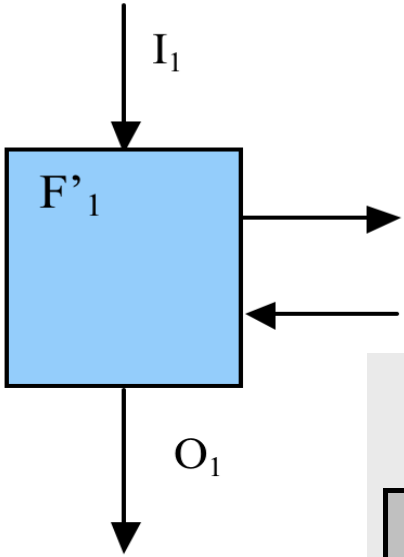


We explain the functional combination $F_1 \otimes F_2$ as a refinement step

The steps of function combination

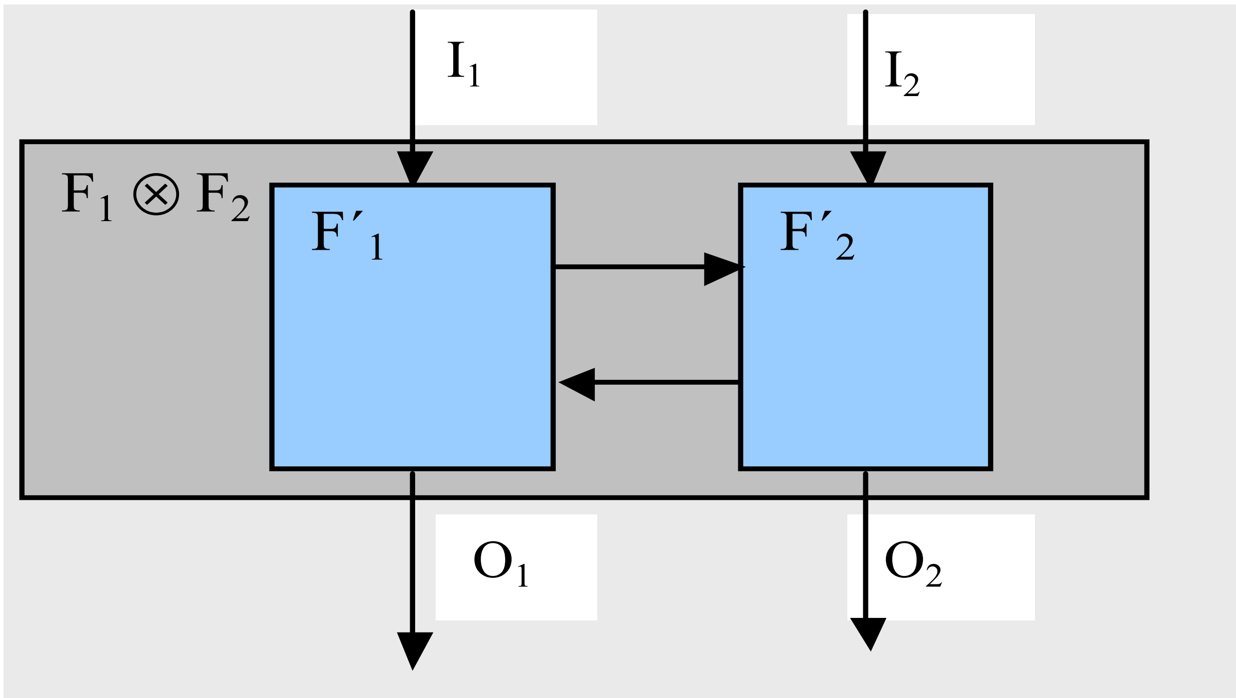


Given the isolated function F_1



We construct a refinement F'_1

And combine F'_1 with a refinement F'_2 of F_2



See: M. Broy: Multifunctional Software Systems: Structured Modeling and Specification of Functional Requirements. Science of Computer Programming 75 (2010), S. 1193–1214

Sub-types between interfaces

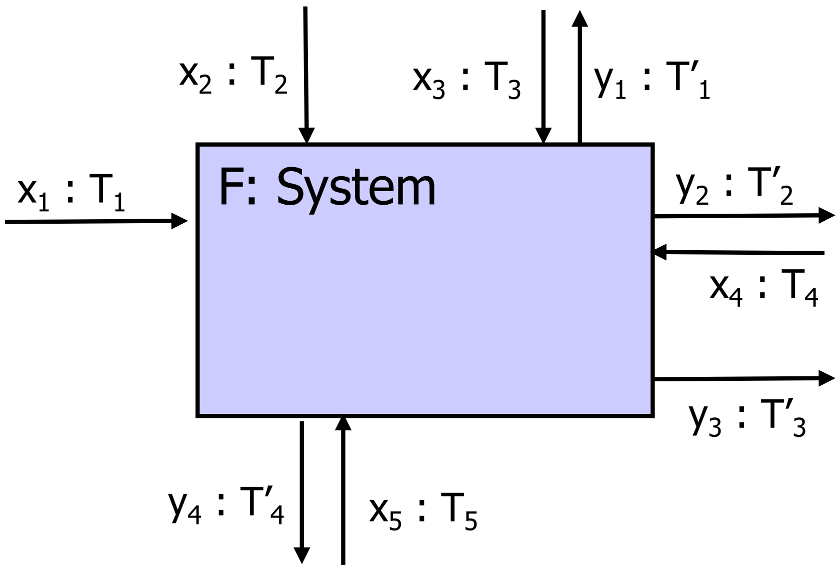
For syntactic interfaces $(I \blacktriangleright O)$ and $(I' \blacktriangleright O')$ where

$$I' \subseteq I \text{ and } O' \subseteq O$$

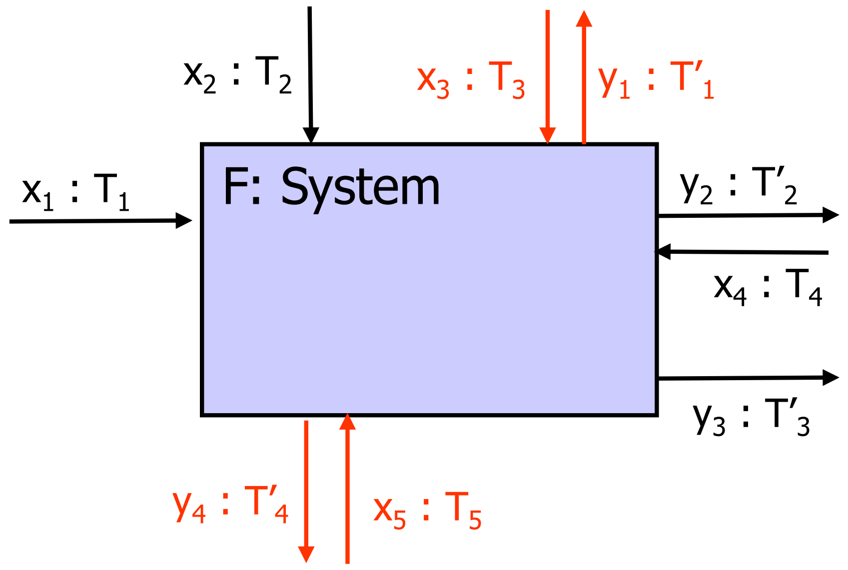
we call $(I' \blacktriangleright O')$ a sub-type of $(I \blacktriangleright O)$ and write:

$$(I' \blacktriangleright O') \subseteq (I \blacktriangleright O)$$

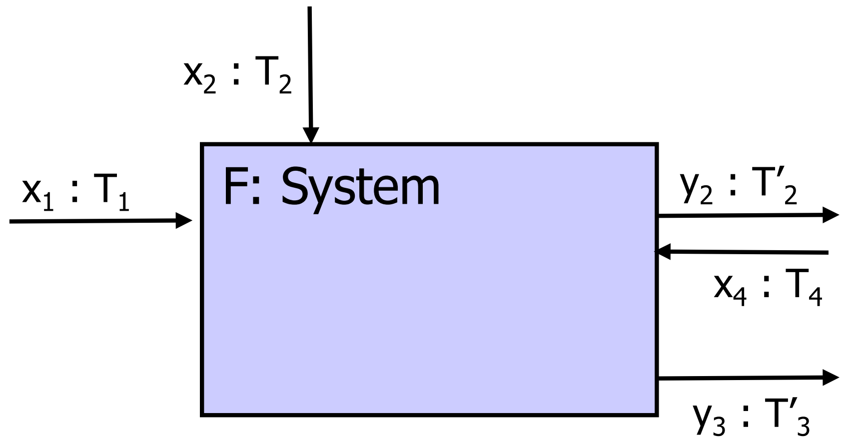
From overall syntactic system interfaces …



to …



sub-interfaces



Projection

Given:

$$(I' \blacktriangleright O') \subseteq (I \blacktriangleright O)$$

define for a behavior function $F \in [I \blacktriangleright O]$ its *projection*

$$F\dagger(I' \blacktriangleright O') \in [I' \blacktriangleright O']$$

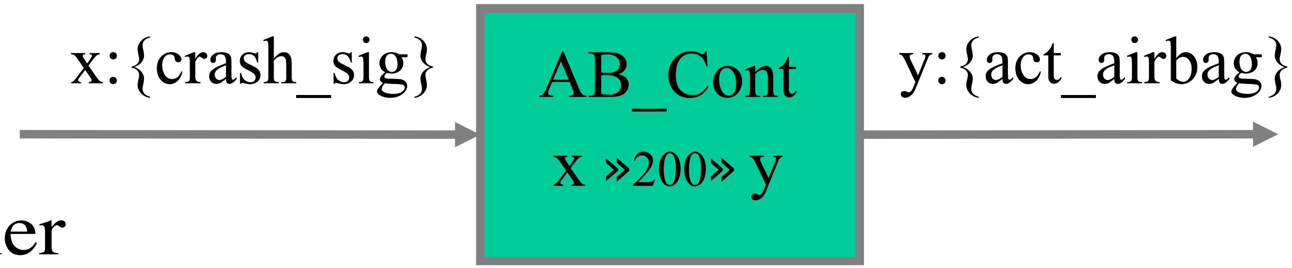
to the syntactic interface $(I' \blacktriangleright O')$ by (for all $x' \in \vec{I}\,'$):

$$F\dagger(I' \blacktriangleright O')(x') = \{y|O' : \exists\, x \in \vec{I} : x' = x|I' \wedge y \in F(x)\}$$

The projection is called *faithful*, if for all $x \in \mathrm{dom}(F)$

$$F(x)|O' = (F\dagger(I' \blacktriangleright O'))(x|I')$$

Example: Component interface specification – Airbag Controller



An air bag controller

AB_Cont

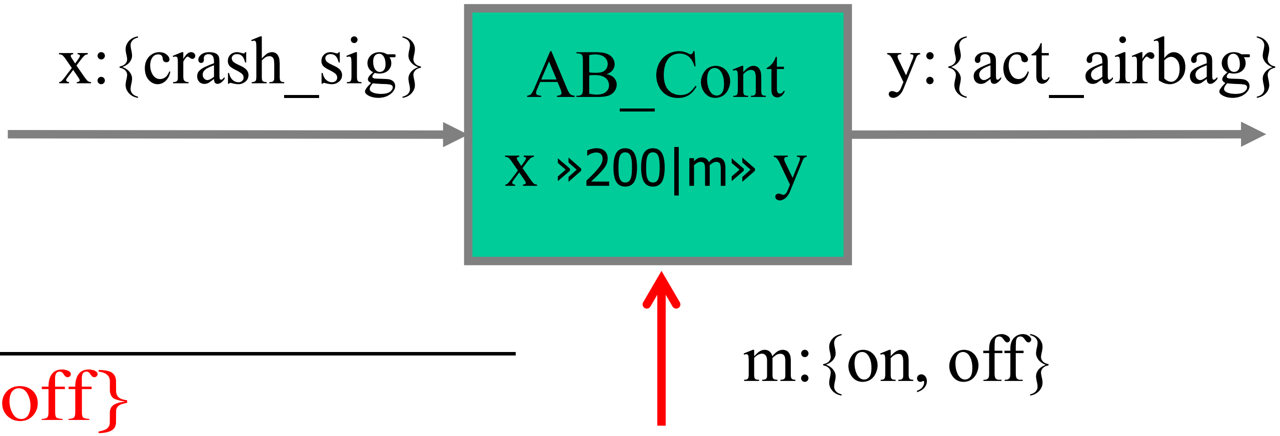
in	x: AB_I
out	y: AB_O
x »200» y	

x »200» y ≡ ($\forall$ t $\in$ Time:

$$crash\_sig \in x(t) \Leftrightarrow act\_airbag \in y(t+200))$$

Example: Component interface specification – Airbag Controller

An air bag controller



AB_Cont

in	x: AB_I, m: {on, off}
out	y: AB_O
x »200|m» y	

x »200|m» y ≡ (∀ t ∈ Time:

(ON(m, t+199) ∧ crash_sig ∈ x(t)) ⇔ act_airbag ∈ y(t+200))

ON(m, t) = if t = 0 then false elif on ∈ m(t) then true
elif off ∈ m(t) then false else ON(m, t-1) fi

Feature interaction in the architecture view

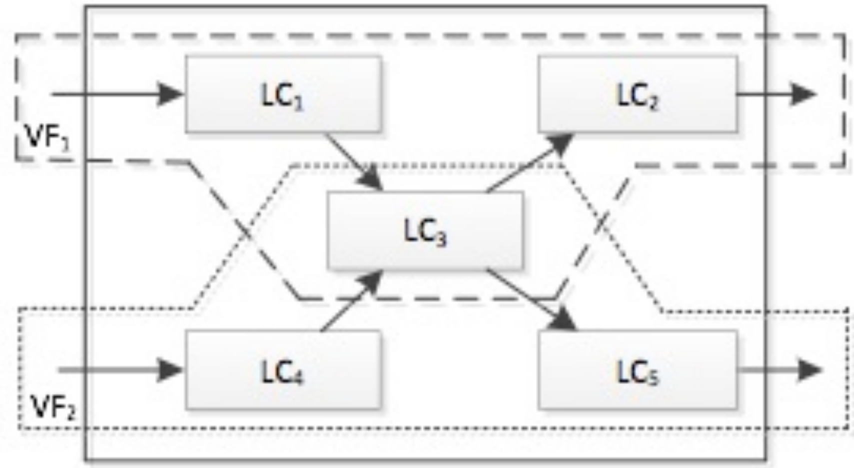
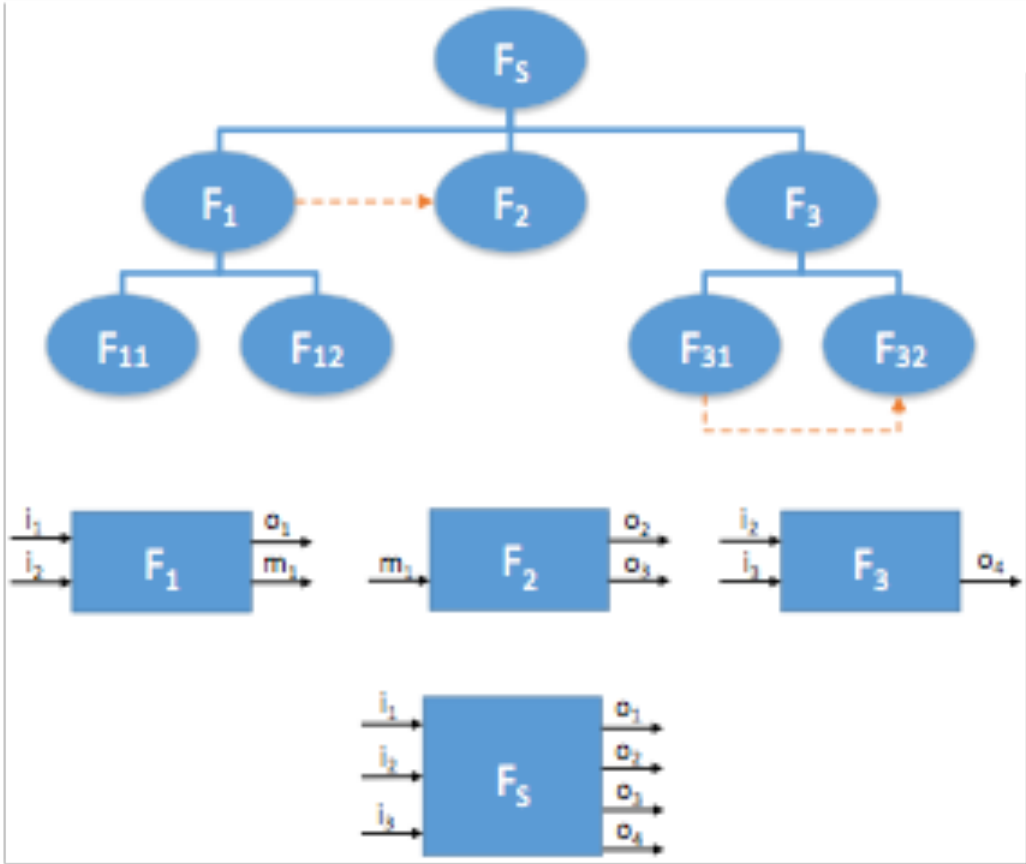


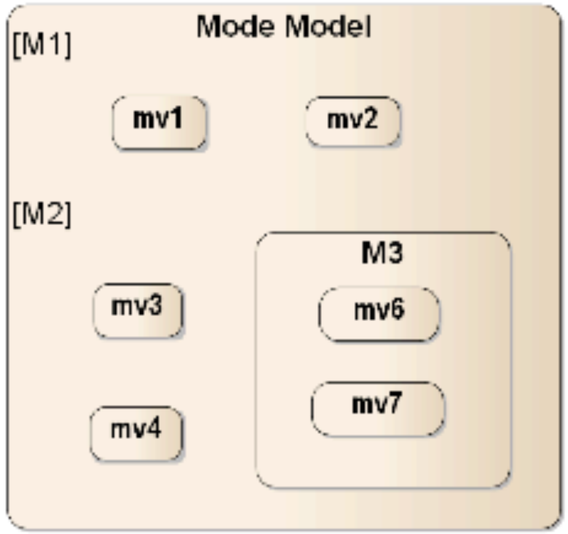
Table 4.2: Extent of dependencies in the vehicle function graph

	MAN System ($n = 55 \widehat{=} 100\%$)		**BMW System** ($n = 94 \widehat{=} 100\%$)	
Vehicle functions...	**Number**	**Ratio**	**Number**	**Ratio**
with incoming dependencies	36	65.5%	81	86.2%
with outgoing dependencies	29	52.7%	72	76.6%
with incoming and outgoing dependencies	27	49.1%	68	72.3%
without dependencies	17	31.0%	9	9.6%

Taken from:
A. Vogelsang: Model-based Requirements Engineering for Multifunctional Systems. PH. D. Dissertation, Technische Universität München, Fakultät für Informatik, 2014

Functional features





Mode list

Mode Name	Description	Mode Values
M1	...	mv1, mv2
M2	...	mv3, mv4, M3
M3	...	mv6, mv7

Figure 5.7: The modes contained in the *mode list* are structured in a *mode model*.

Functional features

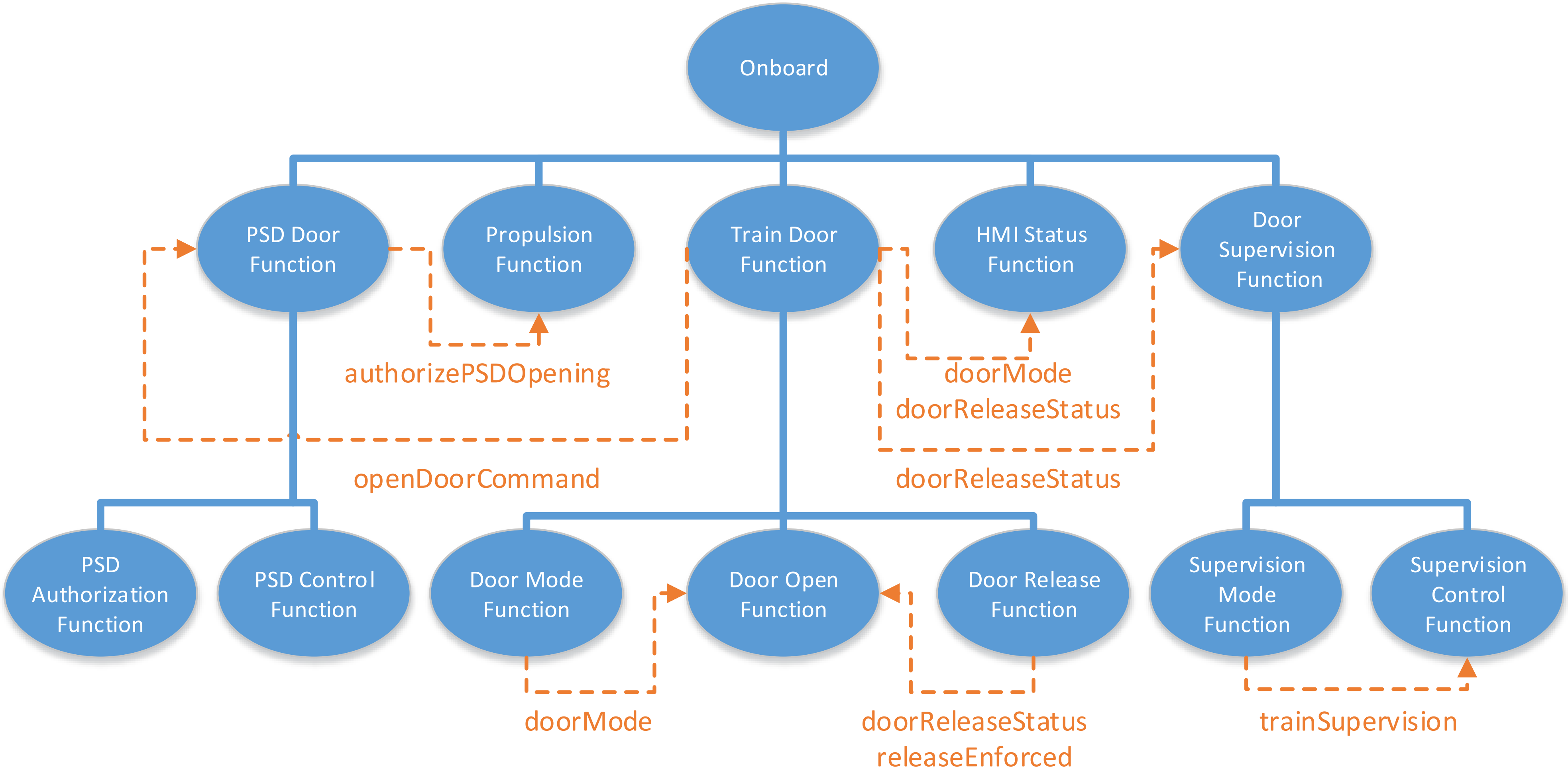


Figure 5.38: Function hierarchy of the onboard subsystem.

Modes

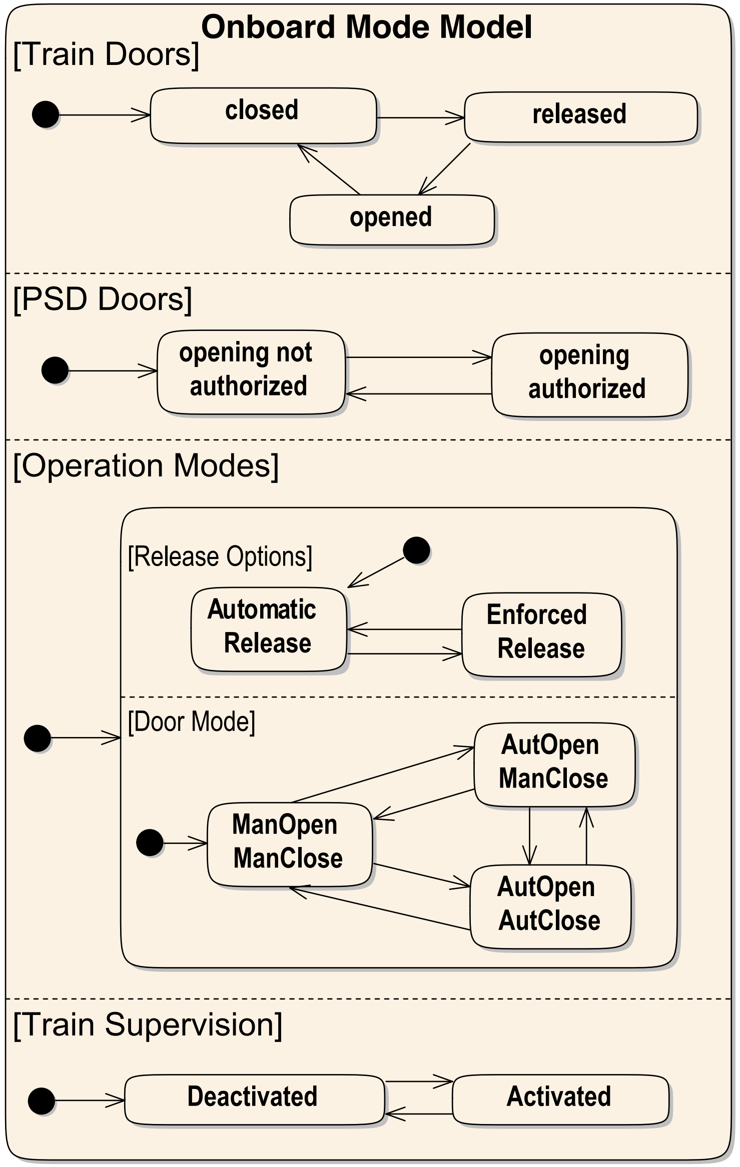
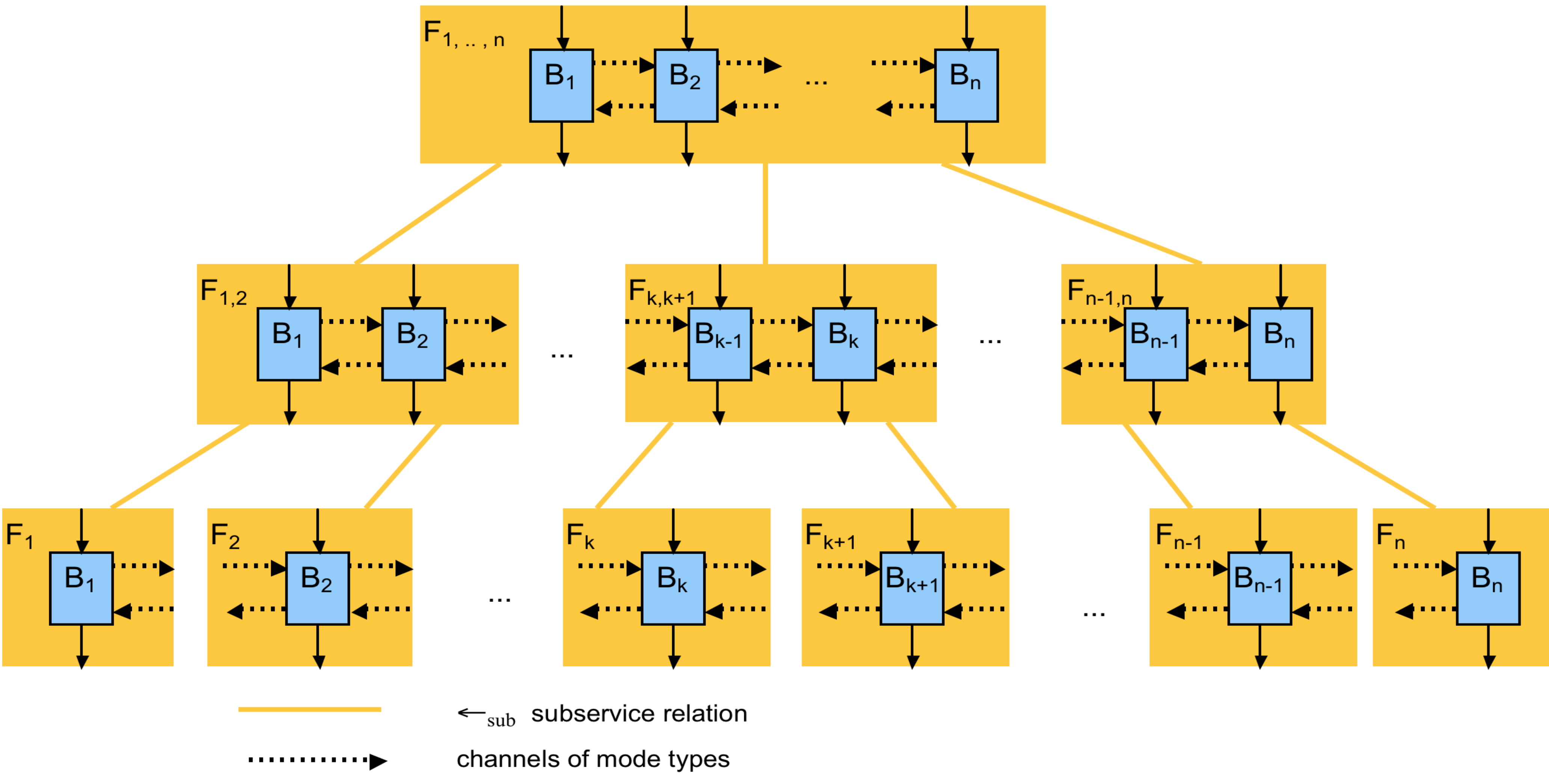
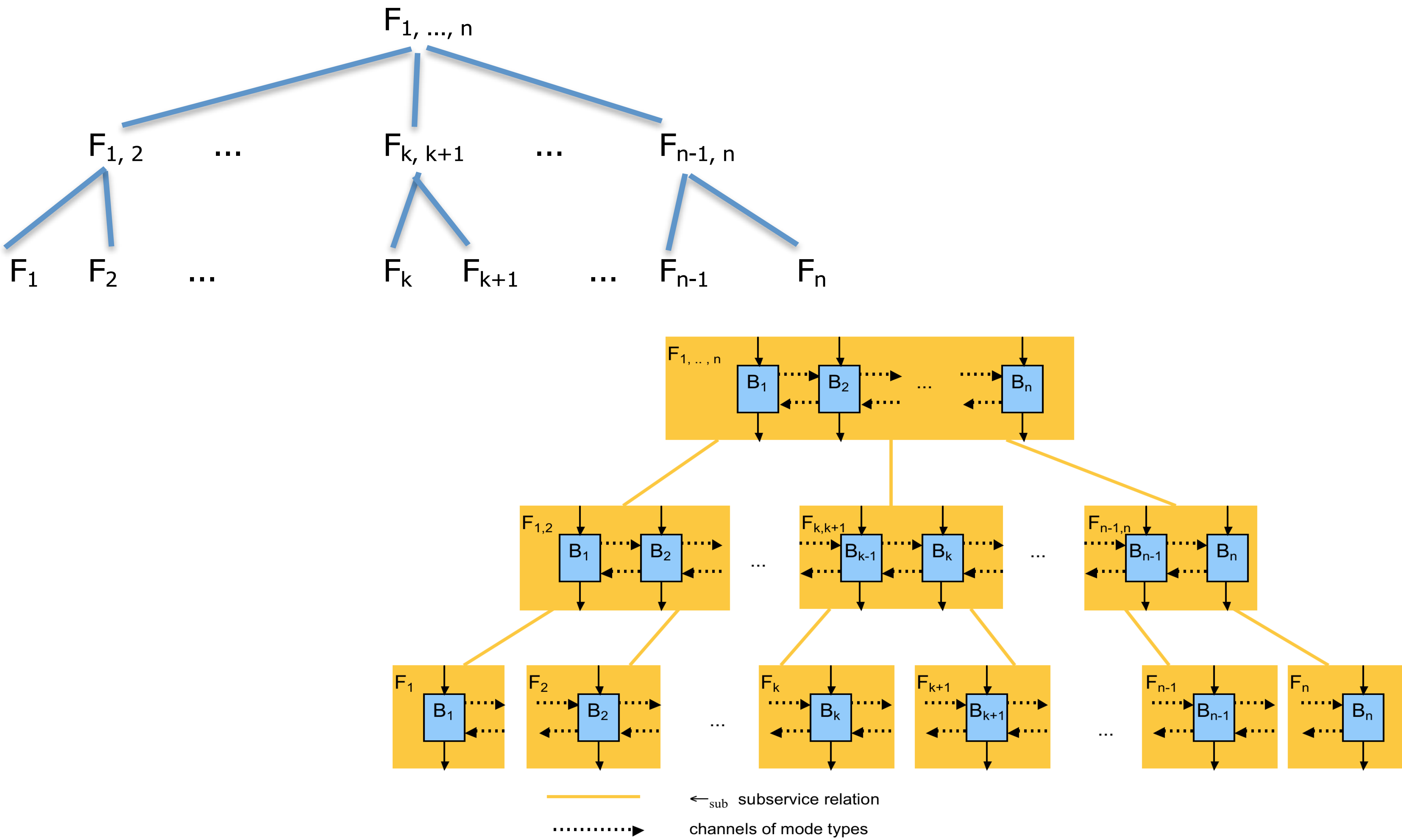


Figure 5.40: Mode model of the onboard subsystem represented by a statechart.

Function Hierarchy



An interpreted feature tree

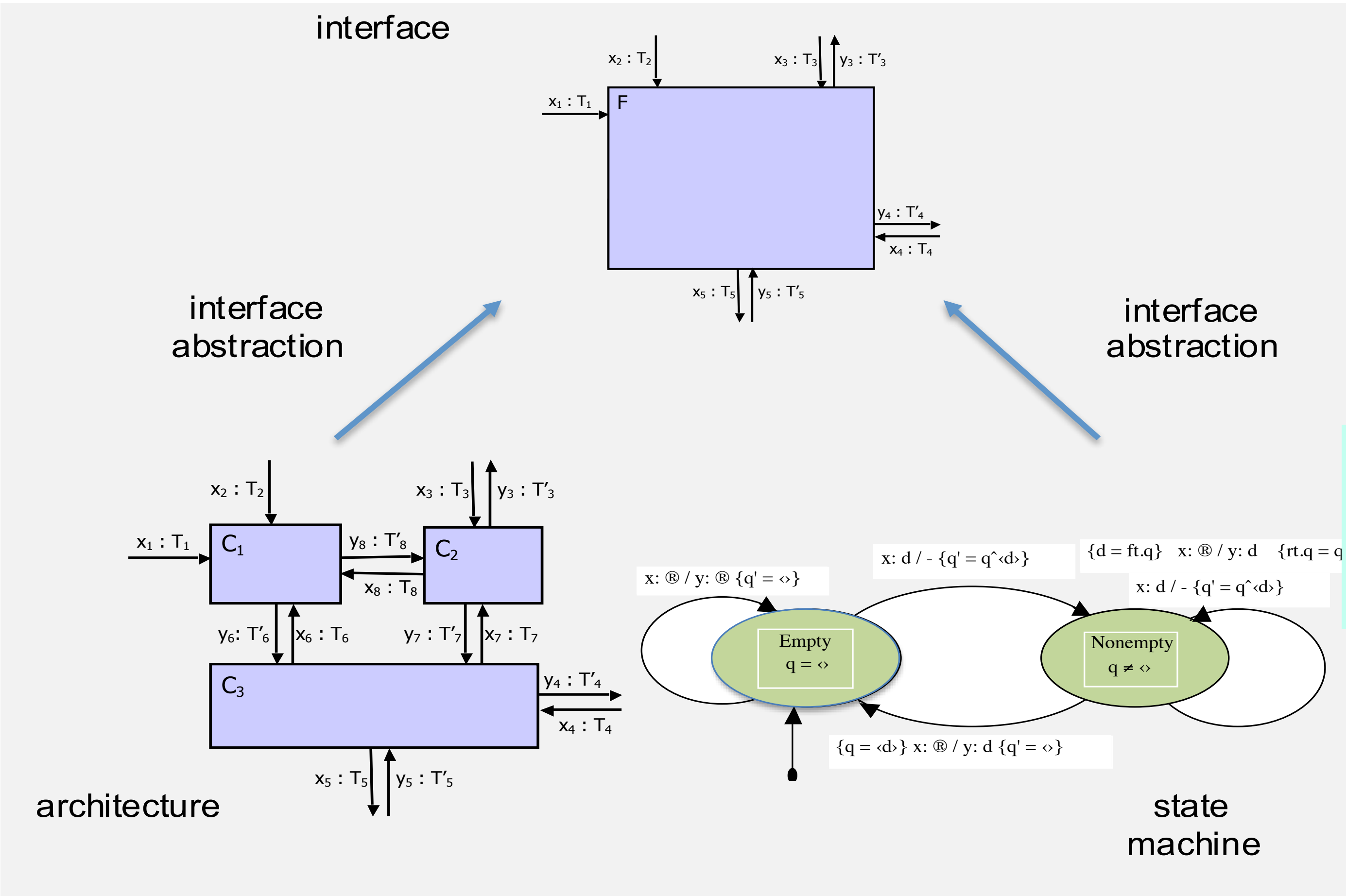


Model Integration

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany

Integrating modeling concepts

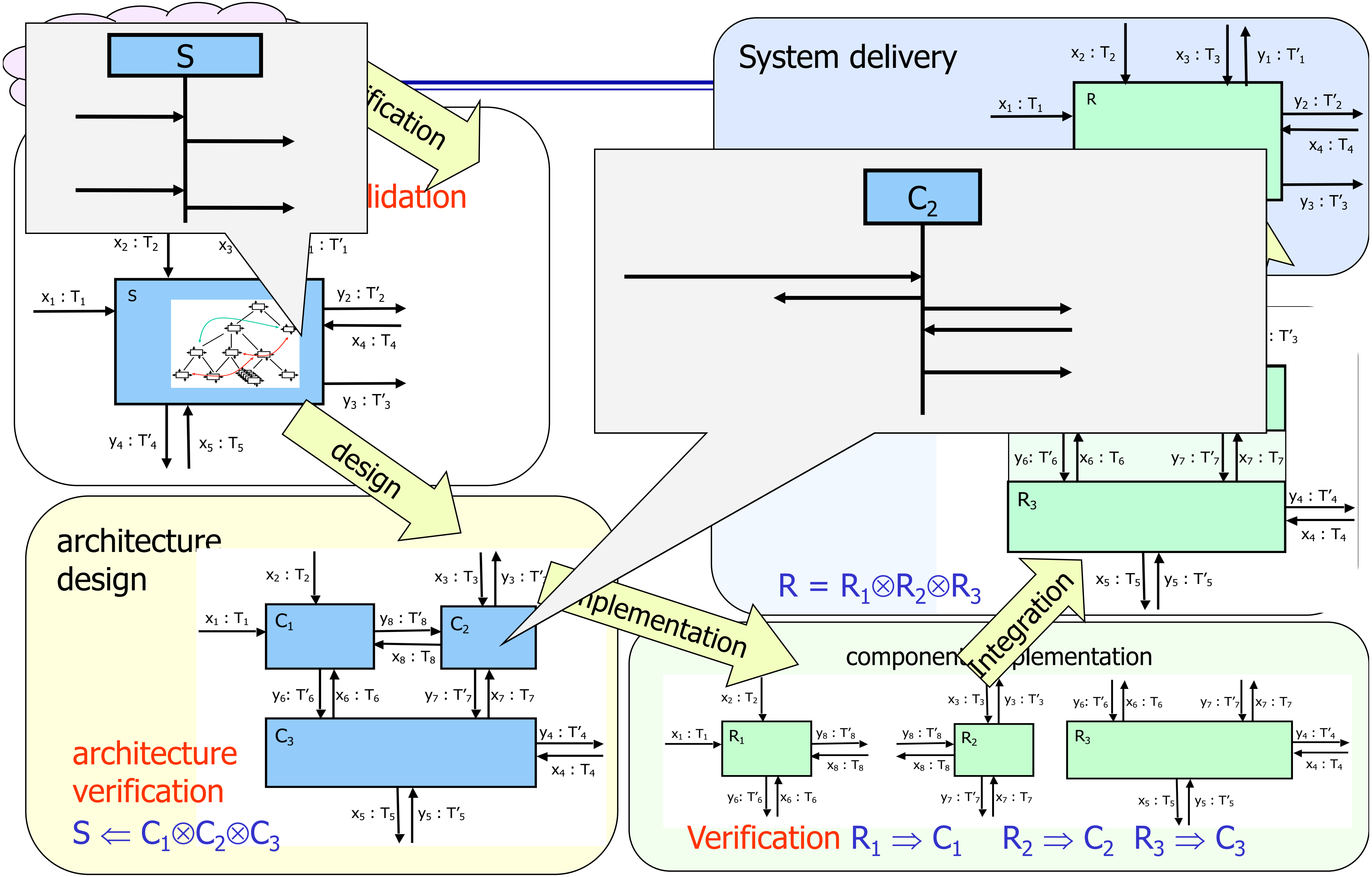
- An architecture can be abstracted into an interface behavior
  - ◊ Proof techniques for architecture verification
- A state machine can be abstracted into an interface behavior
  - ◊ Proof techniques for implementation verification



M. Broy: The Semantic and Methodological Essence of Message Sequence Charts. Science of Computer Programming, SCP 54:2-3, 2004, 213-256

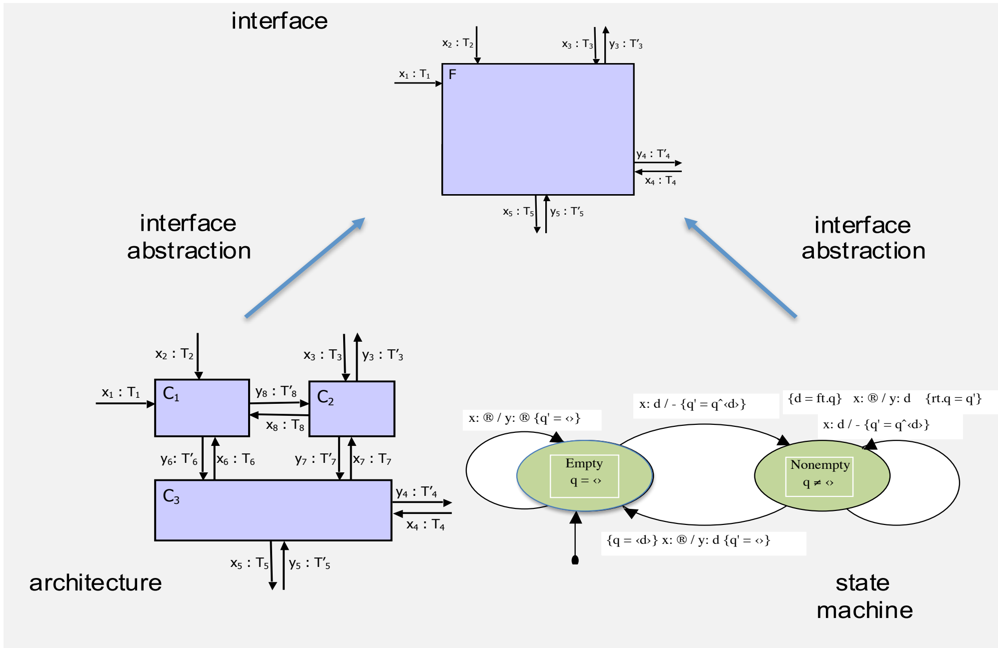
Modular Model Based System Development

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany


System delivery
R = R₁⊗R₂⊗R₃
architecture design
architecture verification
S ⇐ C₁⊗C₂⊗C₃
design
implementation
integration
component implementation
Verification R₁ ⇒ C₁ R₂ ⇒ C₂ R₃ ⇒ C₃
lidation
ification
S
C₂
C₁
C₃
R
R₁
R₂
R₃
x₁ : T₁
x₂ : T₂
x₃ : T₃
y₁ : T′₁
y₂ : T′₂
x₄ : T₄
y₃ : T′₃
y₄ : T′₄
x₅ : T₅
y₅ : T′₅
y₆: T′₆
x₆ : T₆
y₇ : T′₇
x₇ : T₇
y₈ : T′₈
x₈ : T₈

What did we get?

- A complete and precise modeling approach
  - ◊ Mathematical models – denotational semantics
  - ◊ Logical representation – for specifcation and reasoning
  - ◊ Graphical (and tabular) representation – for structured representation
- Semantic coherence



What can we do with it?

- Systems and software engineering?
  - Capturing properties and concepts of systems
  - Tools
- Formal methods?
  - Proofs
- Foundational framework?
  - Making concepts clear
  - Proving methods correct

The power of generalizing ideas, of drawing comprehensive conclusions from individual observations, is the only acquirement, for an immortal being, that really deserves the name of knowledge.

"Mary Wollstonecraft (1759–1797), British feminist. A Vindication of the Rights of Woman, ch. 4 (1792)