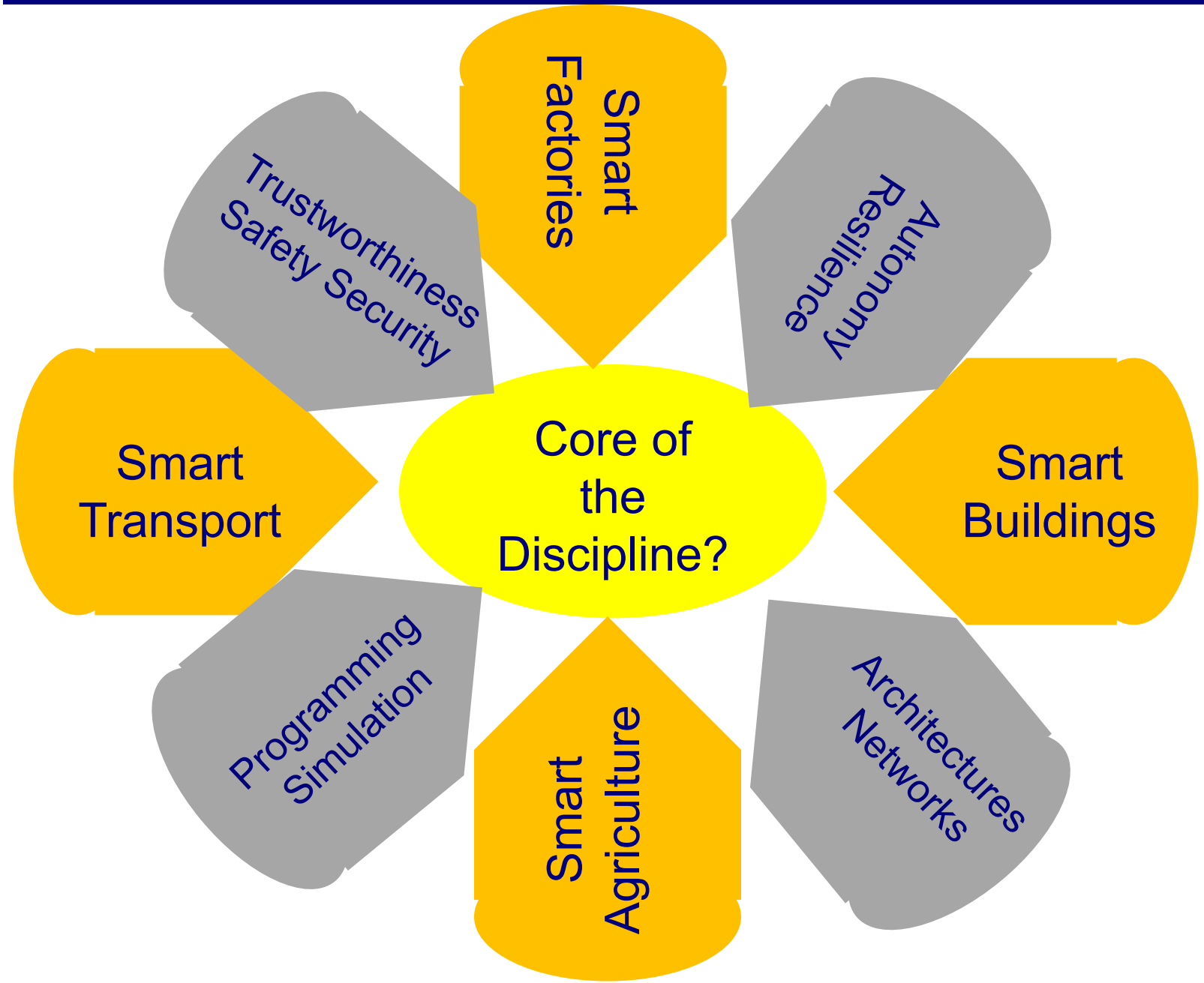
Core Research Topics for CPS

Joseph Sifakis
EPFL
The BIP team, Verimag

Workshop on “Game Changing and Controversial Topics in Cyber Physical Systems”
Budapest University of Technology and Economics
April 15, 2016

Is CPS a Discipline in its Own Right?

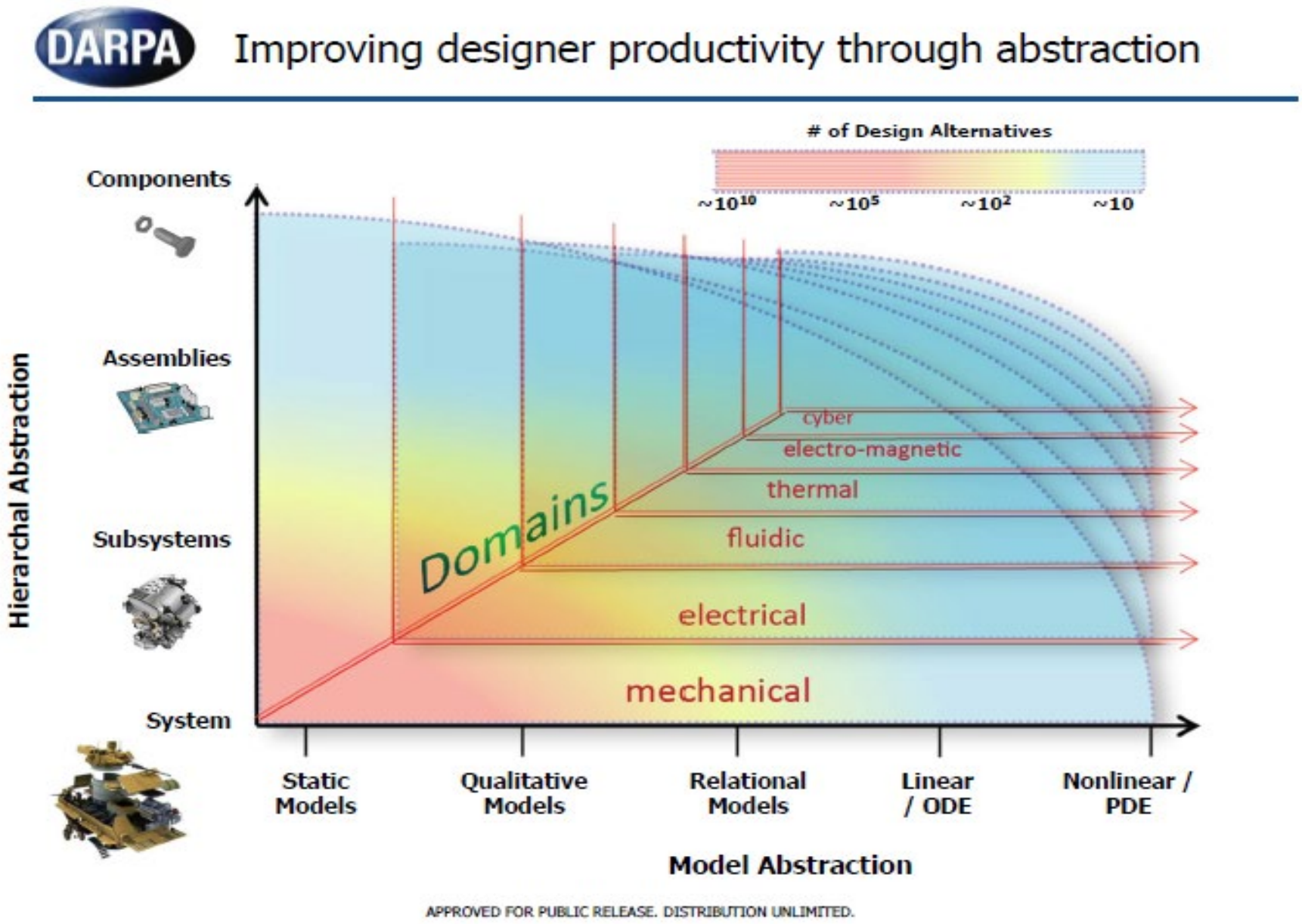


Core of the CPS Discipline

The Core of the Discipline

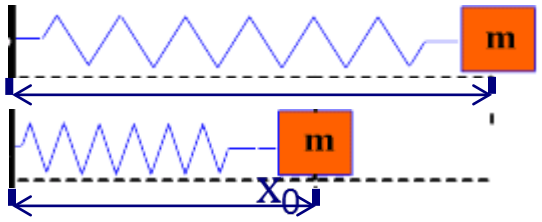
- Theory: Linking Physicality and Computation, in particular unification of modeling frameworks
- Engineering: Rigorous Design of CPS, in particular building CPS systems from CPS components

Multiscale and Multidomain Model Integration



Linking Physicality and Computation

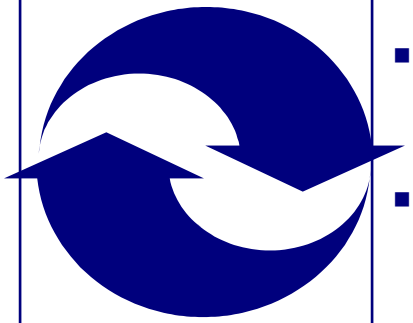
Both physics and computing deal with systems X'= f(X,Y) where

Physics	Computing
X' = dX/dt X is the current state Y is the current input Variables are functions of time	X' is the next state X is the current state Y is the current input Discrete variables
 $m\frac{d^2x}{dt^2} = -kx$ Law: $kx_0^2 - kx^2 = mv^2$	while $x \neq y$ do if $x>y$ then $x:=x-y$ else $y:=y-x$ Law: $GCD(x,y)=GCD(x_0,y_0)$
Physical system models are inherently synchronous (timed); they are driven by uniform laws.	Computation models ignore physical time; they are driven by specific laws defined by their designers

Linking Physicality and Computation

Physics

- Deals with phenomena of the « real » physical world (transformations of matter/energy)
- Focuses mainly on the discovery of physical laws.
- Physical systems – Analytic models
- Continuous mathematics
- Differential equations - robustness
- Predictability
- Mature discipline

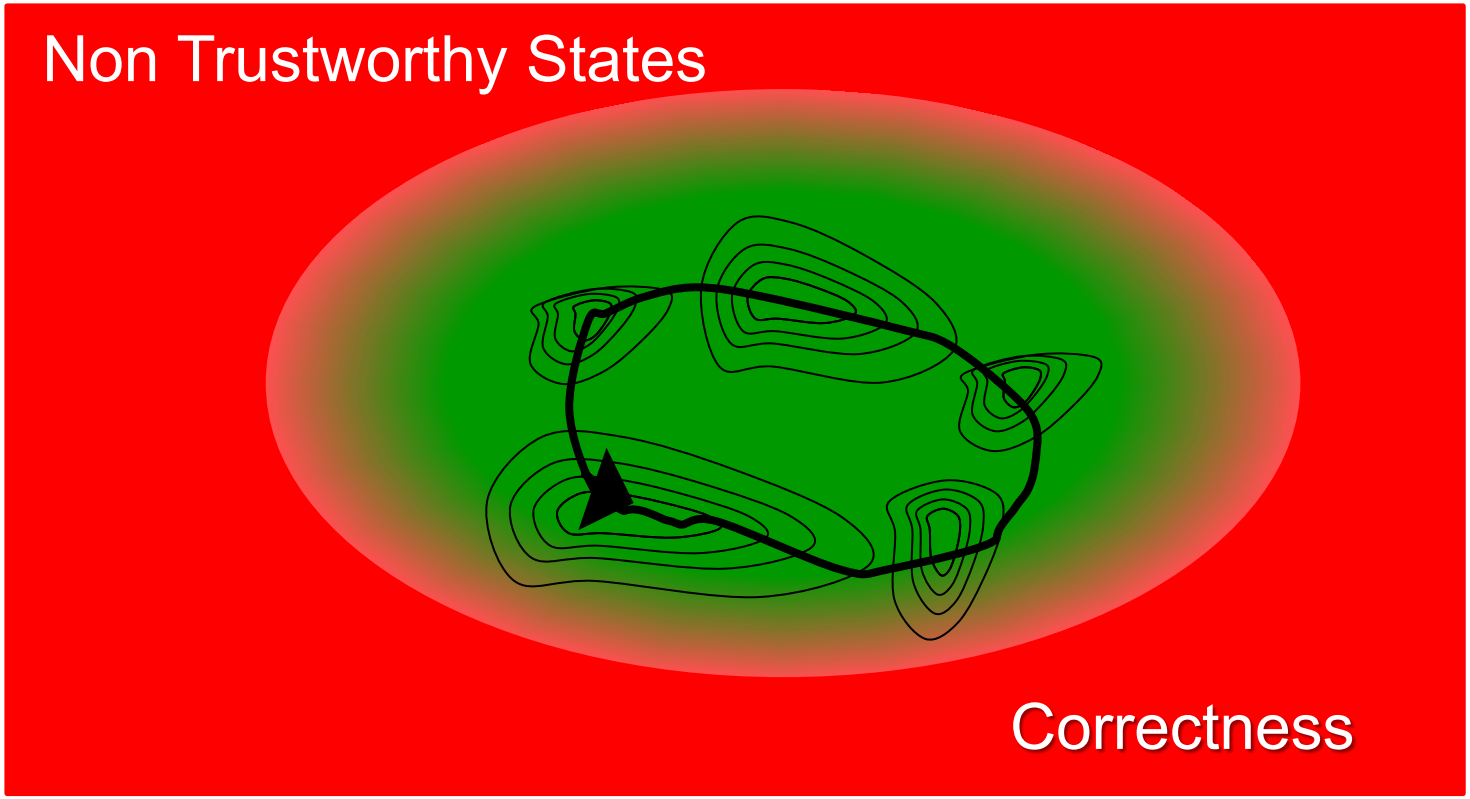


Computing

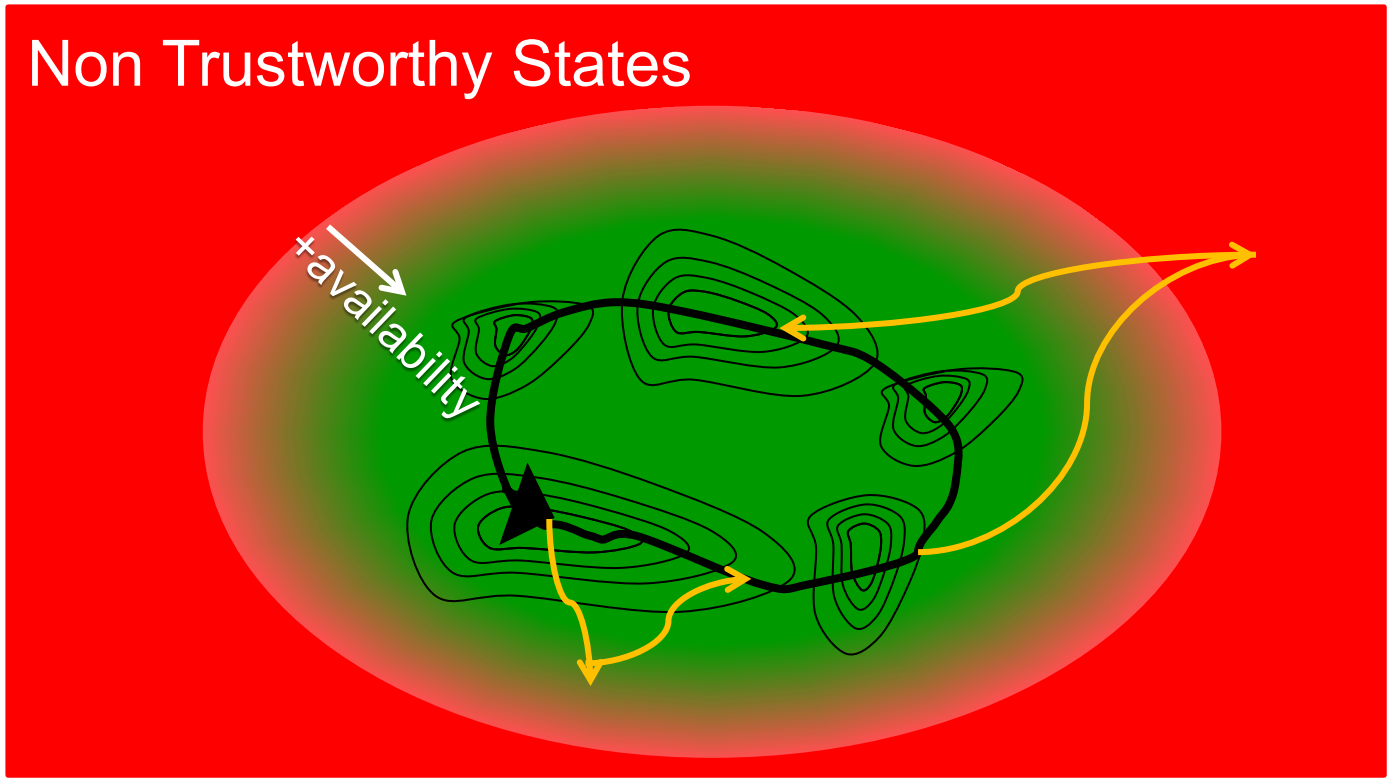
- Deals with the representation, transformation and transmission of information
- Focuses mainly on the construction of systems
- Computing systems – Machines
- Discrete mathematics – Logic
- Automata, Algorithms, Complexity Theory
- Verification, Testing,
- Young fast evolving discipline

System Correctness

- Correctness = Nominal Correctness + Resilience
- Nominal Correctness = Nominal Critical Correctness + Optimization
- Resilience = Fault tolerance + Threat resistance
- Fault tolerance = Resilience to HW failures, loss of messages, any event that may lead to accidental deviation from nominal behavior
- Threat resistance = Resilience to attacks or any event that may jeopardize system security
- Trustworthiness = Nominal Critical Correctness + Resilience



System Correctness



Critical system: Non trustworthy states are forbidden – degraded mode is tolerated
Mission Critical: Availability required for some period of time – degraded mode is ok if possible recovery to trustworthy states
Best effort system: Unavailability accepted if below a threshold.

Languages and Methods

The two pillars of System Design

1. Languages determine the limits: "The limits of my language mean the limits of my world" Ludwig Wittgenstein in Tractatus Logico-Philosophicus.
2. Methods that are relations between descriptions in languages determine to what extent the design problem can be solved.

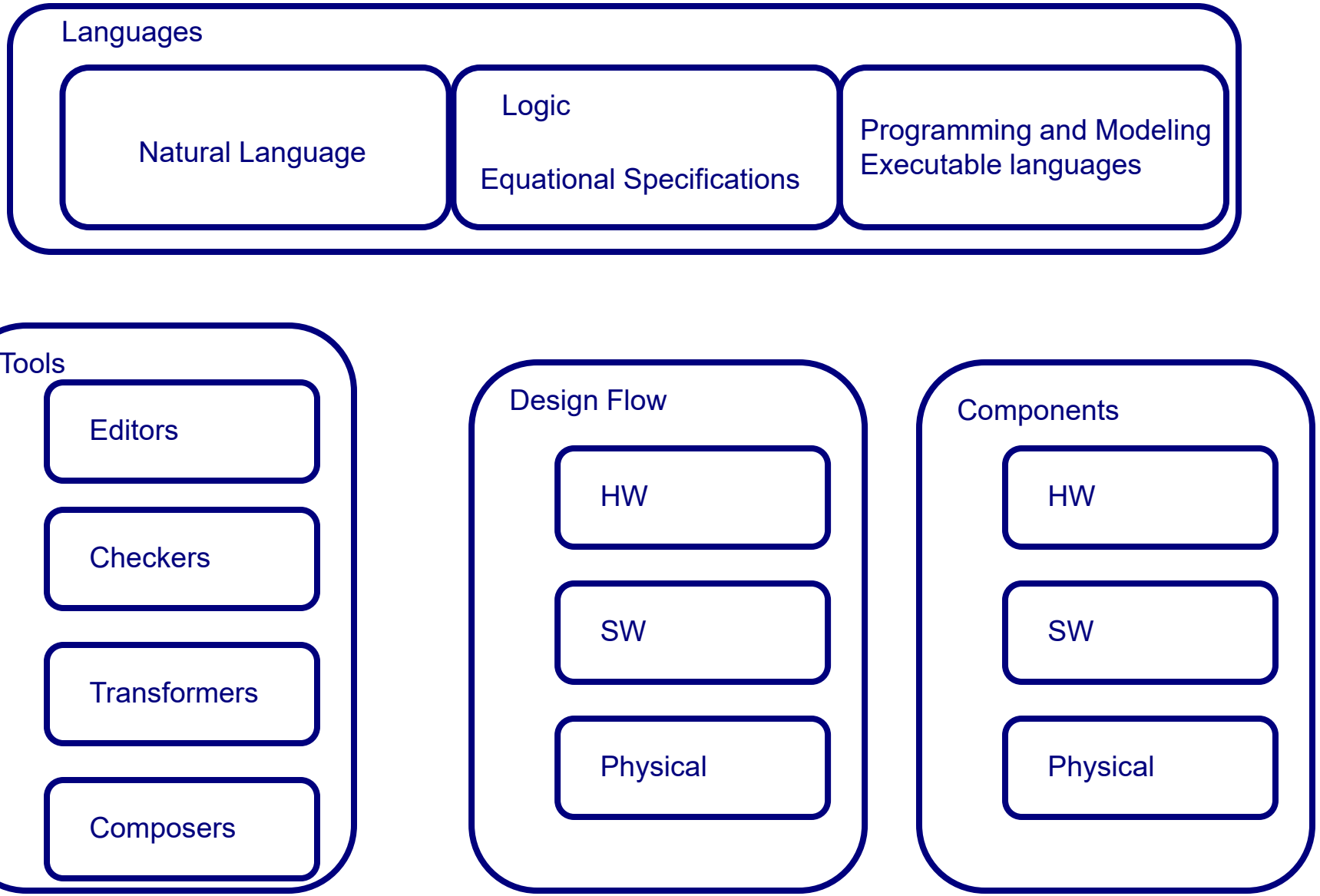
Prerequisites for Design frameworks to guarantee correctness:

- Language coherence: The languages adequately describe various functional and extra-functional aspects of the behavior of the designed system, They can be consistently combined in the design flow as they share a common semantic model.

- Trusted Methods: For each method (tool, component) intended to solve a given problem and thus meeting a property there exist correct implementations and I know how to combine methods to achieve a given result. Quite often we discover that implementations of protocols may be flawed e.g. Paxos distributed consensus protocol, cryptographic protocols

Rigorous design: language and method coherence and traceability of requirements satisfaction

Languages and Methods



Language Coherence – The Language Landscape

Languages are defined by their syntax and (if possible) their semantics. Formally defined languages can be considered as abstract algebras.
In systems engineering we need languages to describe Data, Behavior and Architecture and their properties. The latter are meant to be sets of Data, Behavior and Architecture, respectively. If D is the domain of a language then 2^D is the domain of the corresponding property description language.

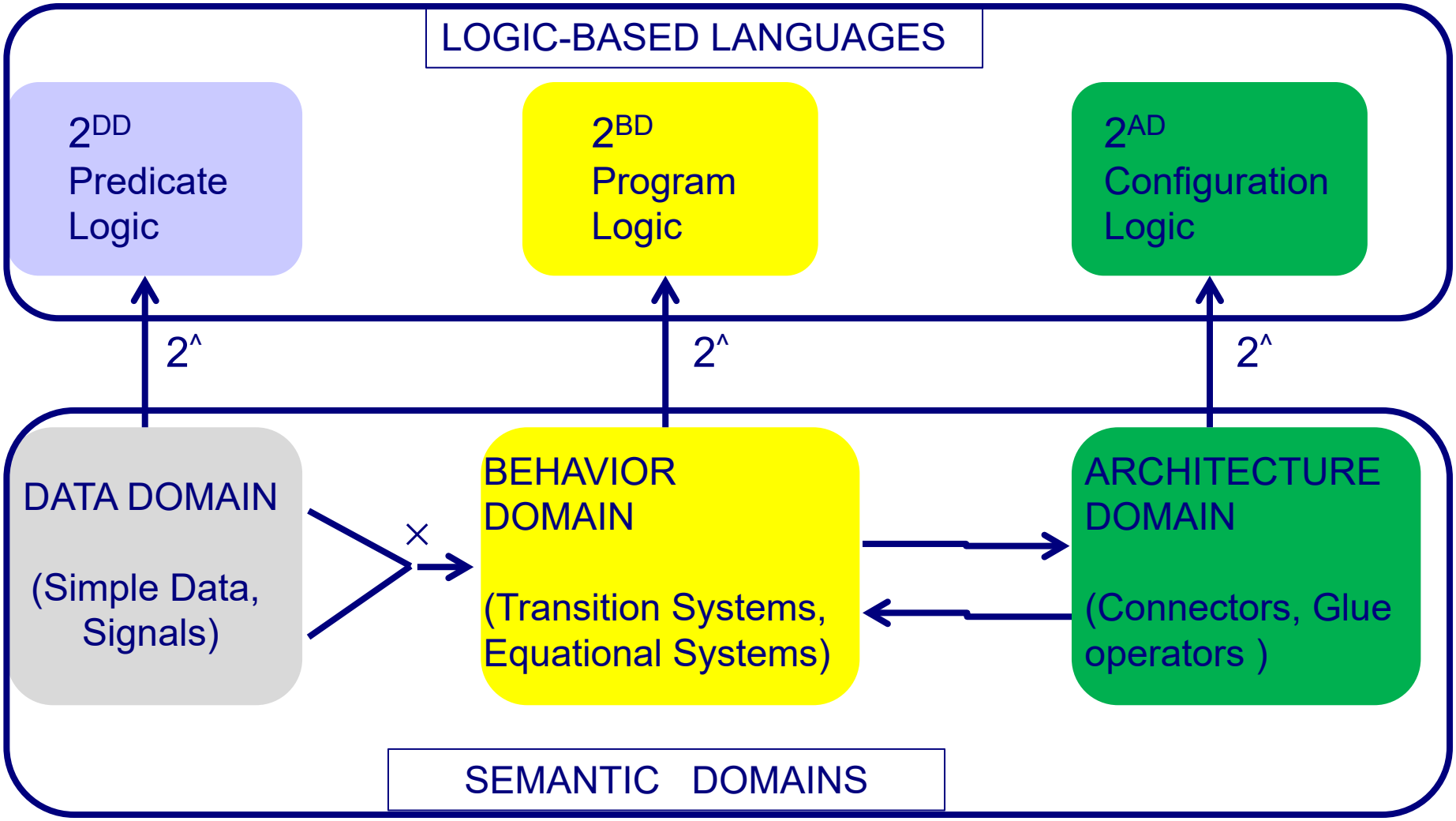
- Data domains include basic domains such as integers, boolean, real as well as structured data . The latter can be used to define functions and relations. In systems engineering particular types of data are signals and events. Signals are functions from some ordered set e.g. integers, reals into a set of values. Events are state changes as discussed below.

- Behavior domains are particular data domains involving relations on states. Behavior can be specified either as a transition system relation between states or as a relation between signals. In the former case, given a state the transition relation defines the set of all its successors by executing one step. In the latter case, one or more transition relations are implicitly specified by systems of equations involving signal variables.

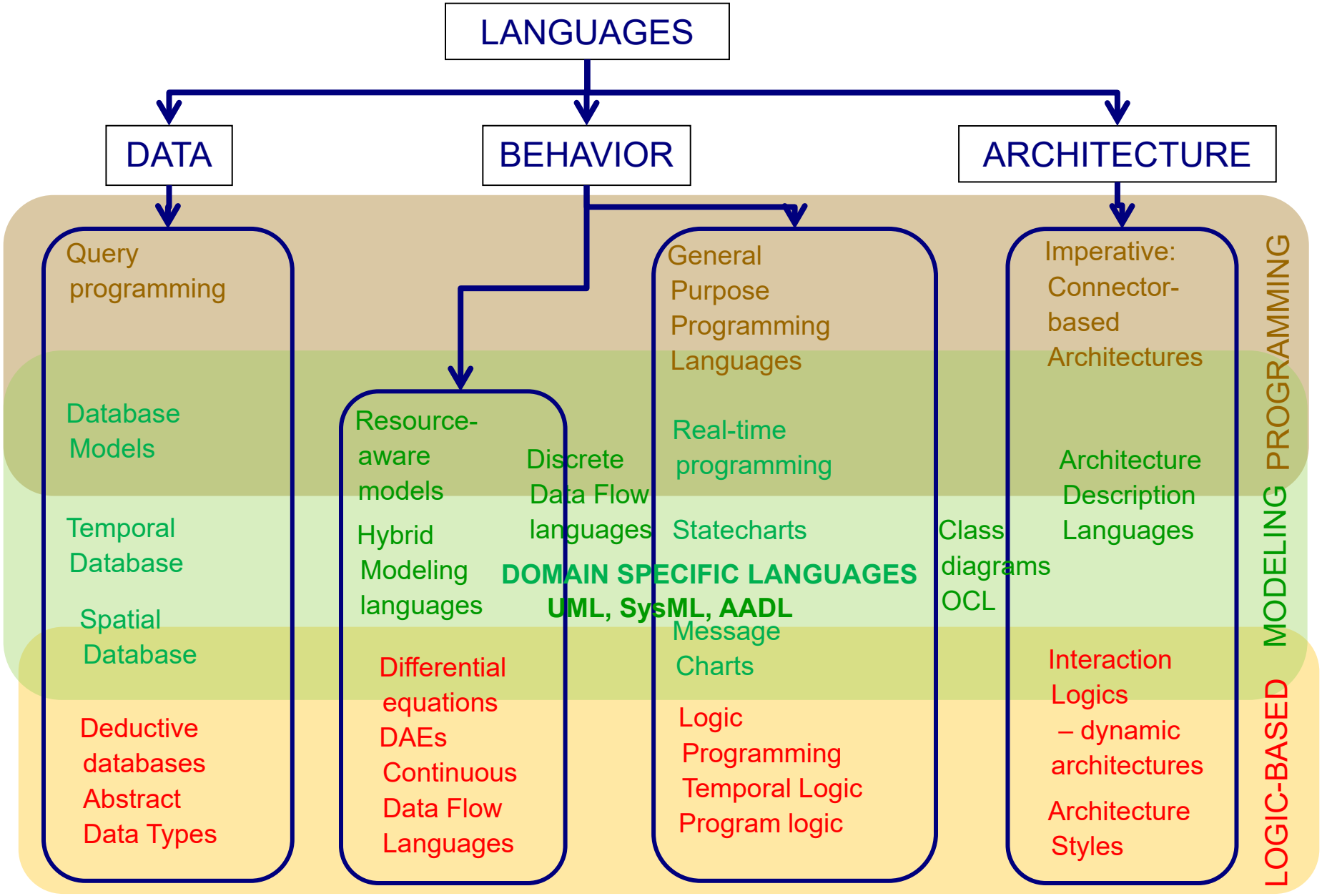
- Architecture domains express coordination constraints between components. A component encapsulates behavior and is equipped with ports through which it can interact with other components. An architecture is used to constraint the behavior of a set of components.

Properties are described using logic-based formalisms. For plain data domains we can define predicate logics of different kinds. For behavior domains there exist program logics, temporal logics that involve modal operators to ease specification. For architecture domains logic may be used to characterize families of architectures.

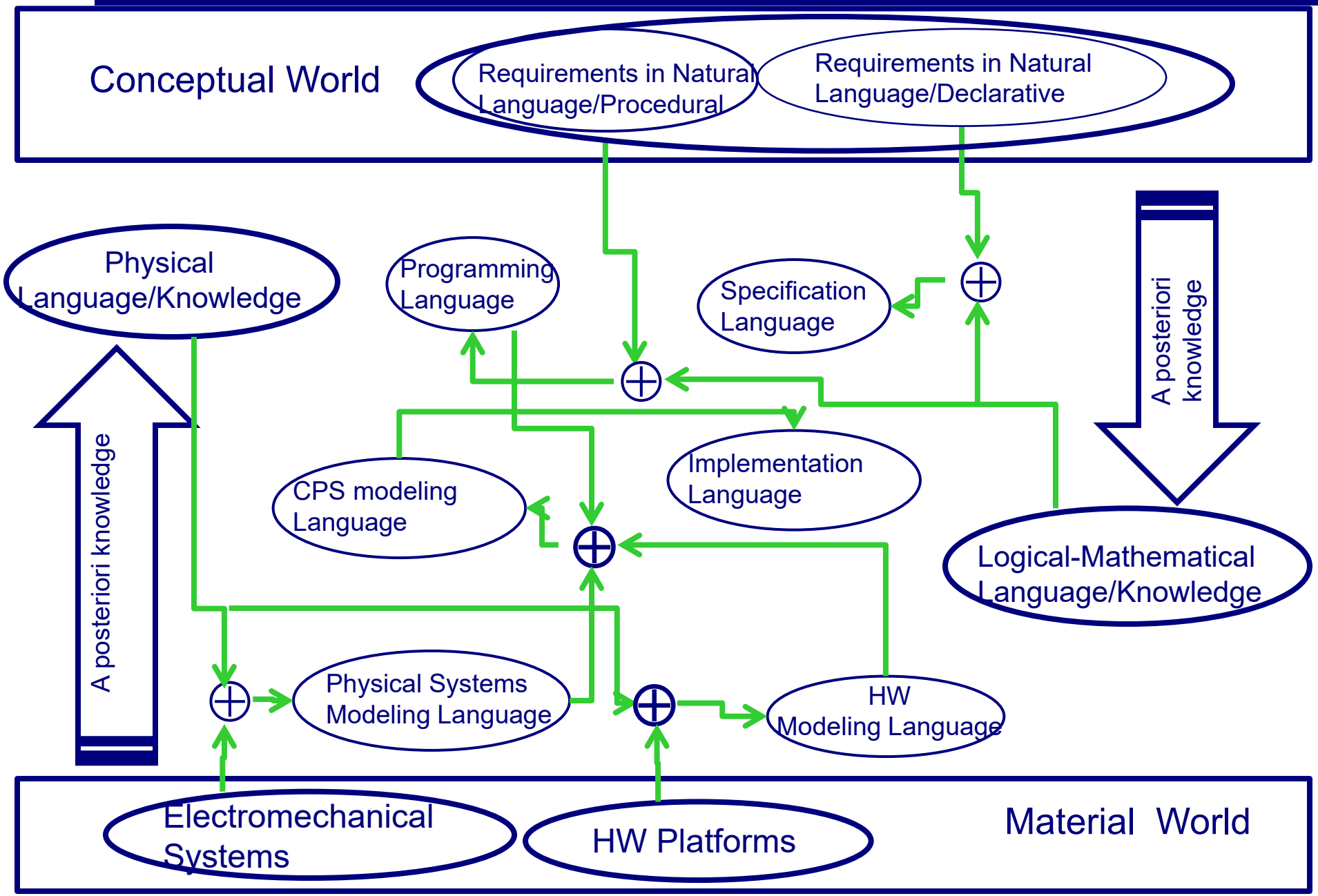
Language Coherence – The Language Landscape



Language Coherence – The Language Landscape



Language Coherence – Integration in a Design Flow



A Panorama of Methods

A Method is a relation between languages – It brings solutions to a problem. A method may be fully automated, partially automated or non automated.
The composition of methods is a method.

Design is a method for building an implementation from requirements. It involves the construction of system descriptions following a flow that combines different methods.

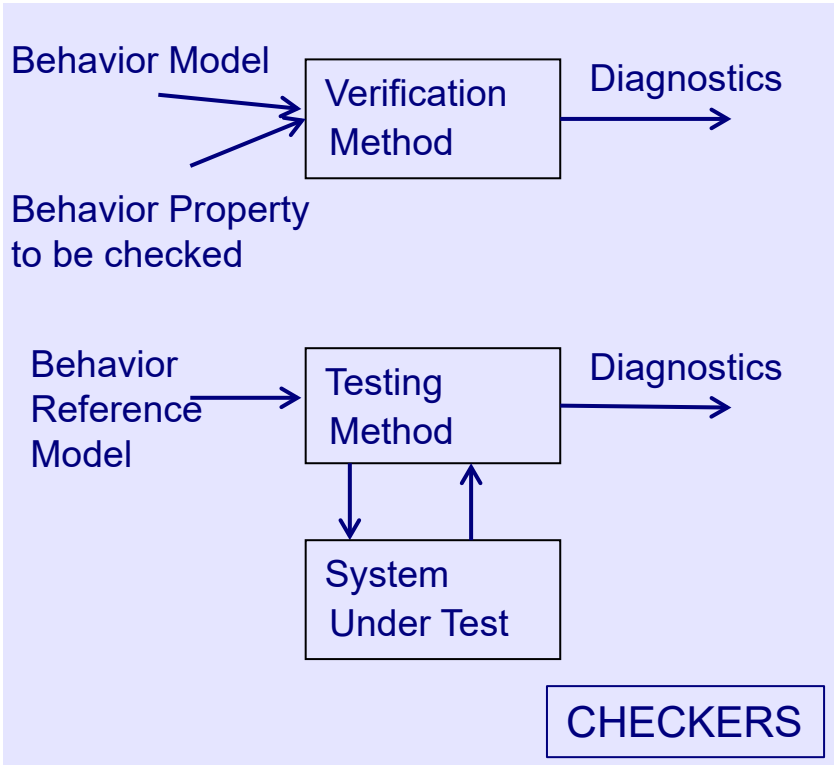
We need trusted generic (independent from the particular system) methods that that are embodied in
- Tools implement methods used to solve generic problems and as a rule, they are not integrated in systems. They are used to produce knowledge about systems.
- Components implement methods used to build the system by applying correct-by-construction techniques

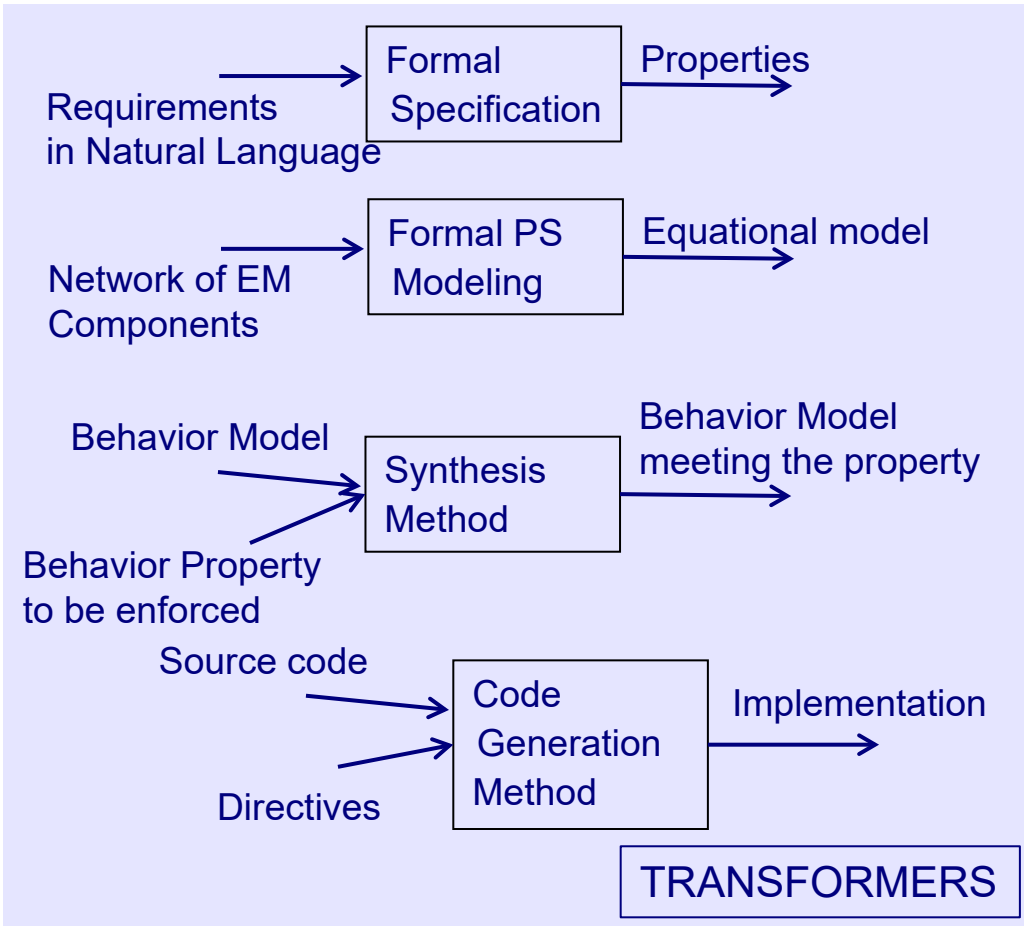
For given application software, model of execution platform and model of external environment there are three main system design problems:

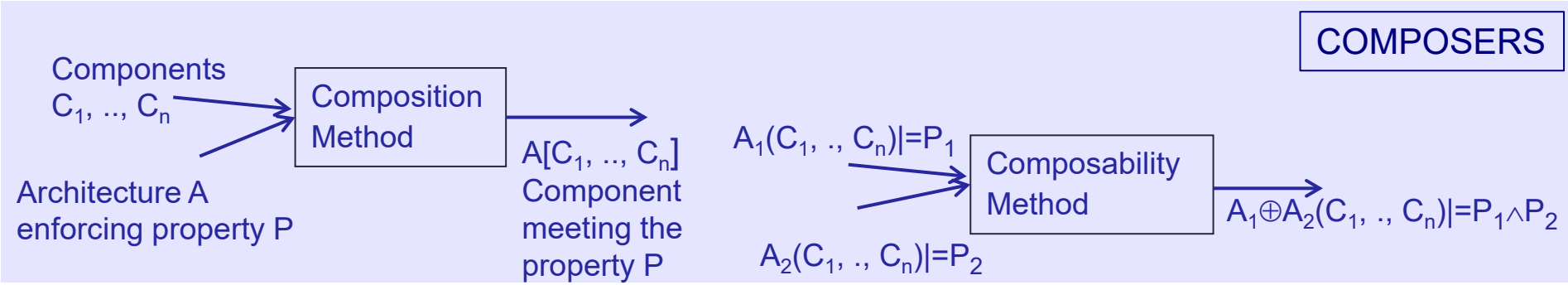
- Resource awareness including scheduling, resource management techniques with adaptive mechanisms.
- Coherent and efficient execution in particular for distributed platforms with mobility and dynamically changing structure
- Knowledge-based techniques for predictability and efficiency to cope with uncertainty and partial information.

The problem is how by using tools, components and predefined architectures, patterns etc to solve these problems

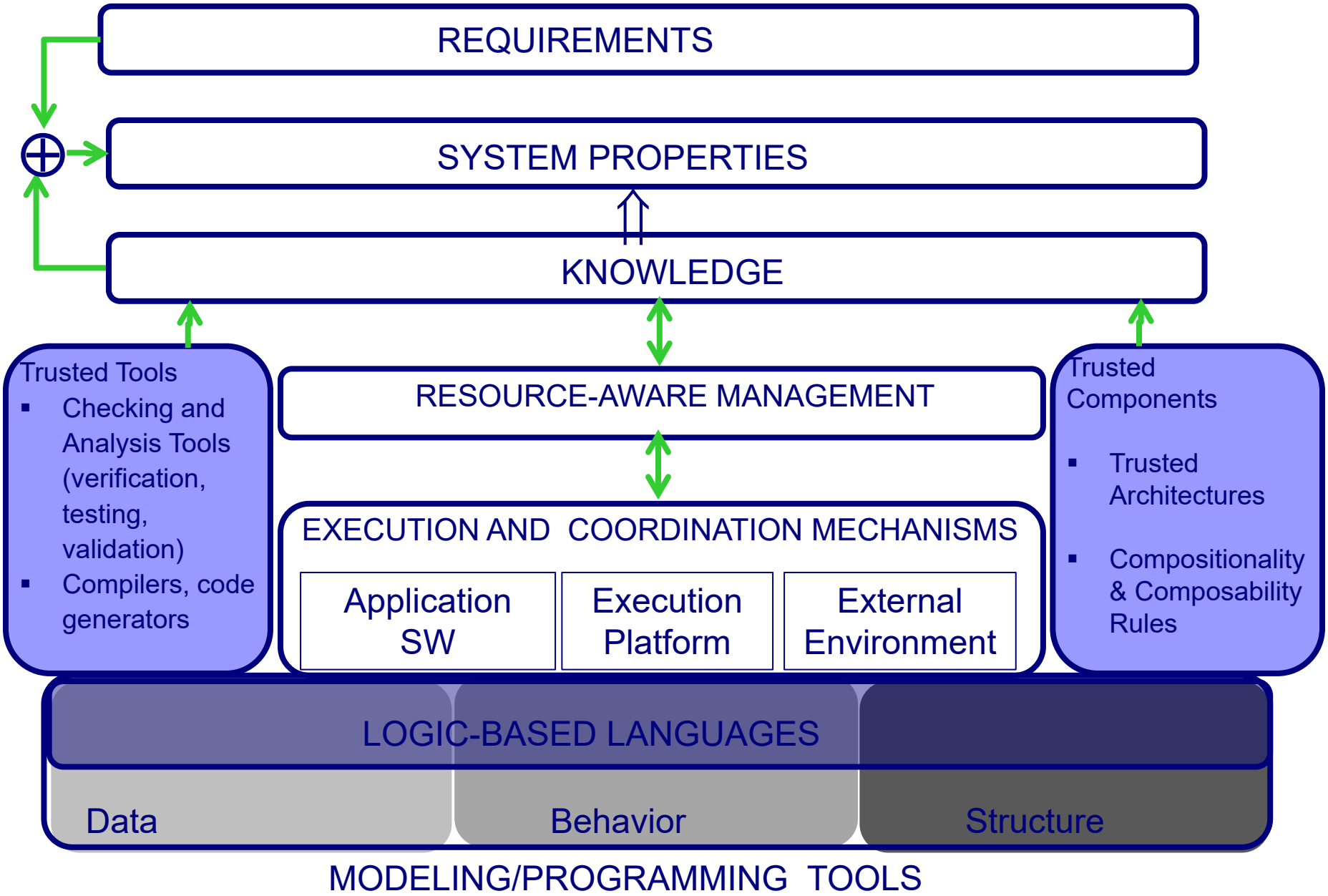
A Panorama of Methods/Tools



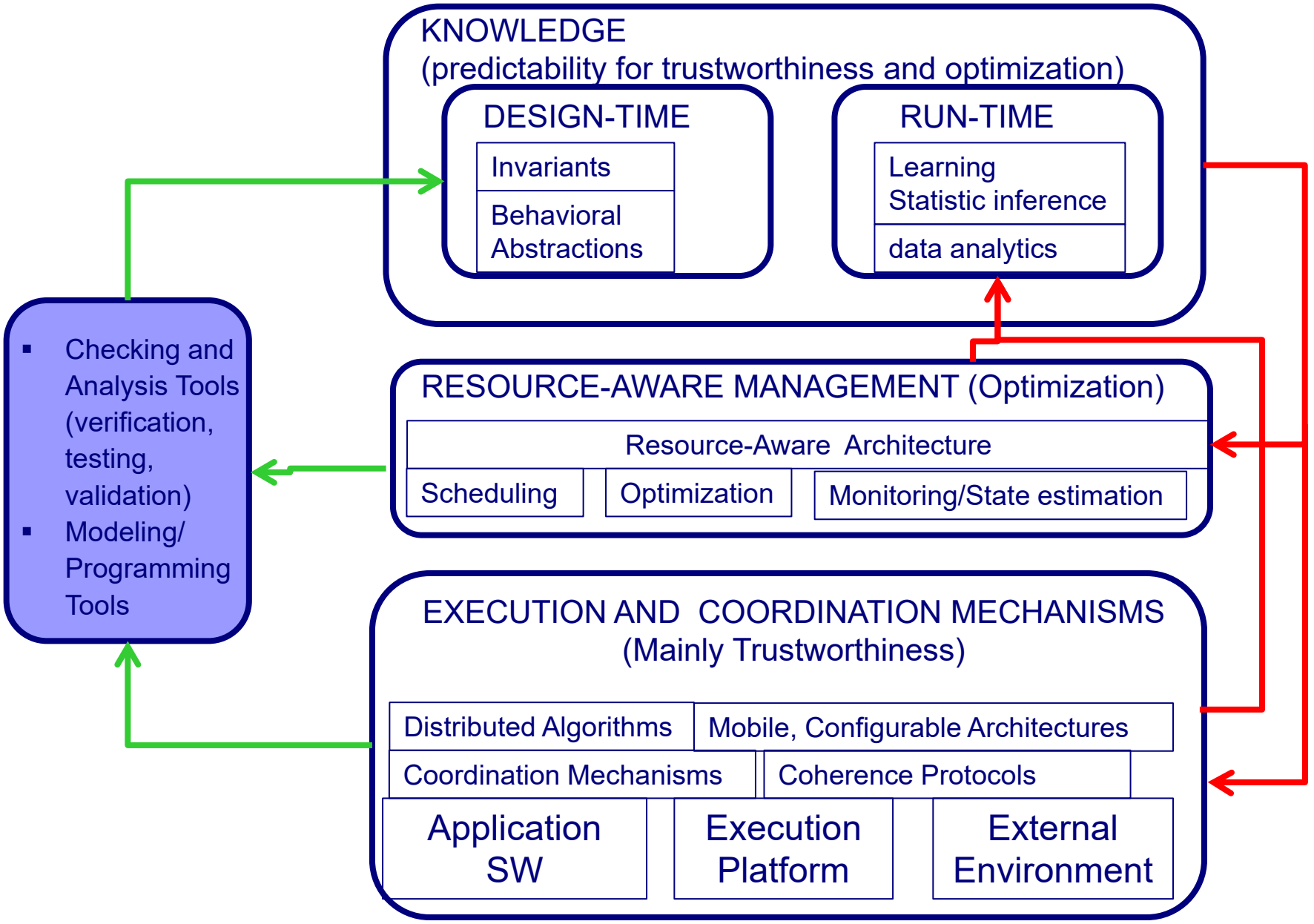




Core System Design Issues



Core System Design Issues



Design-time vs. Run-time Knowledge

Knowledge is information that is a truthful relation in some theory that can be used to understand and/or predict event or can be used to solve a problem.
Distinction between a priori and a posteriori knowledge. A priori knowledge is independent from experience. It is absolute knowledge modulo the validity of the axioms of the underlying theory. A posteriori knowledge dependents an observation and experiments.

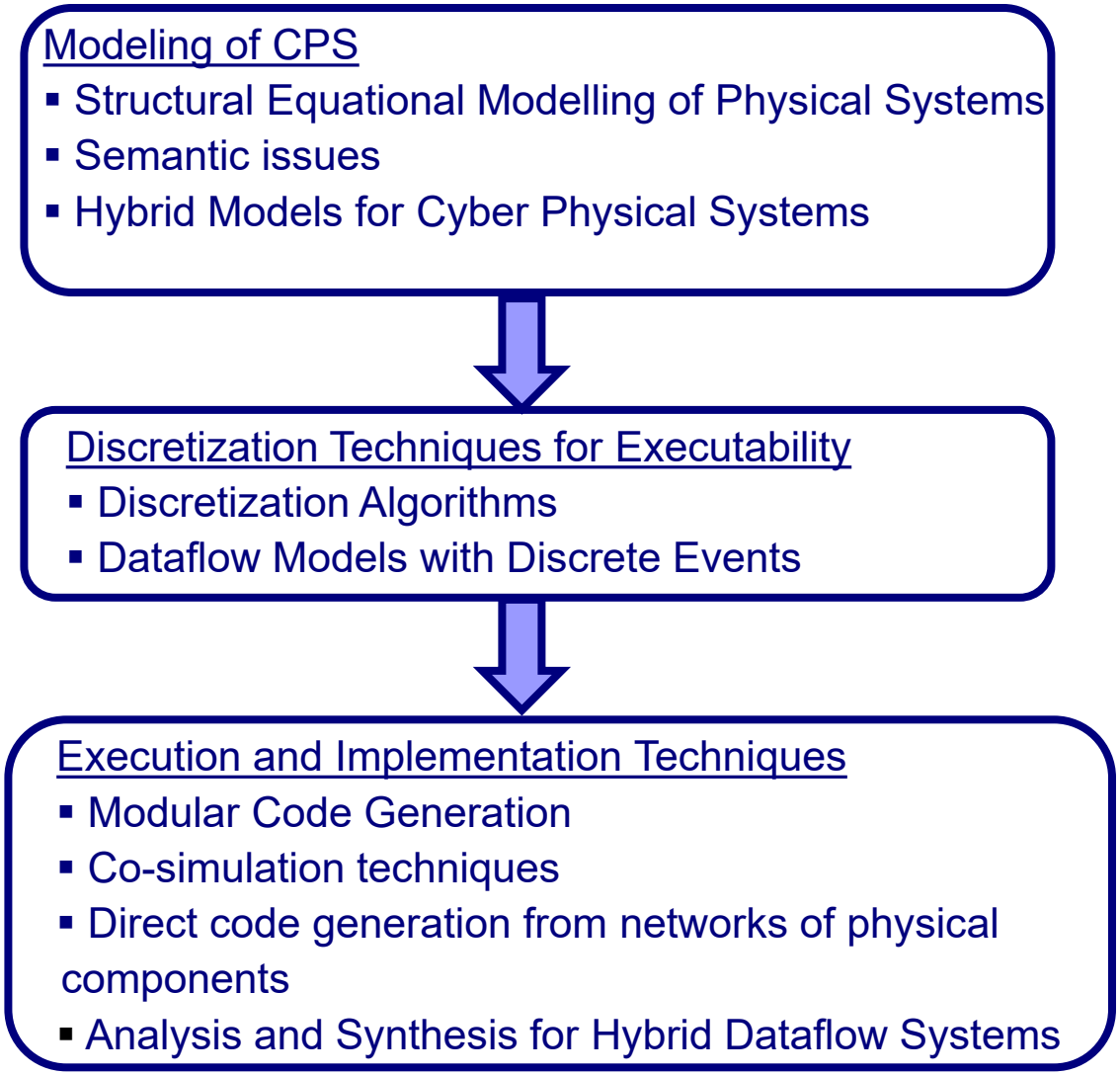
Knowledge may be declarative or procedural. Declarative knowledge is properties (very often invariants) in a logic or preorder relations between models. Procedural knowledge can be used to compute functions (decision procedure in logic, programs, etc.).

The purpose of design is to build an artifact such that our knowledge about it implies the requirements or their formalization as properties.

A method embodies knowledge. It may be either fully automated or semi-automatic or even non automated. It is used to create new knowledge.

Design-time Knowledge: The developed models and their properties – the system is correct if the knowledge about the system implies the requirements. How much can be established at design time??
Run-time Knowledge: Observed relationships and measurements to complement design time knowledge. It can be used to enforce properties (requirements) that cannot be implied from design-time knowledge.

Three Main Problems

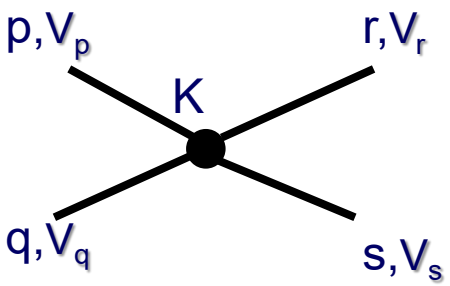


Modeling CPS – Structural Equational Modeling

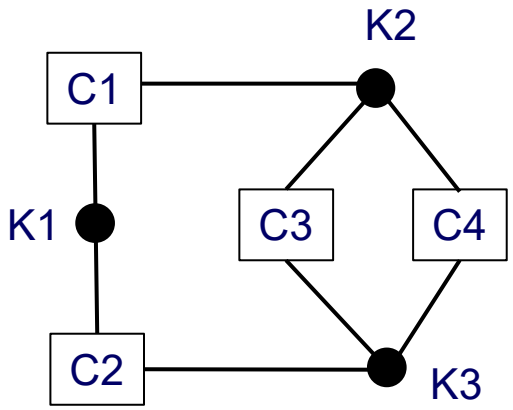
- A network of physical components is built from a limited number of types of components C and connectors K.
- Each component has ports p, q, r and associated general state variables V_p, V_q, V_r
- The network is a hypergraph: vertices are component ports and edges are connectors (sets of ports)
  - Component C has a characteristic equation of the form $E_C(V_p,V_q)$.
  - Connector K relating ports p,q, .. ,r has equations of the form $E_K(V_p,V_q,...,V_r)$ – equations may be constrained in general.

Declarative compositionality : The meaning of a network with sets of components $\{C_i\}_{i\in I}$ and set of connectors $\{K_j\}_{j\in J}$ is the union $\{E_{Ci}\}_{i\in I} \cup \{E_{Kj}\}_{j\in J}$.

Component C: $E_C(V_p,V_q)$

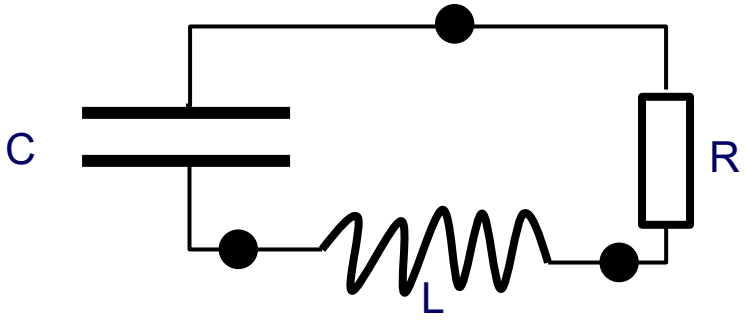


Connector K: $E_K(V_p,V_q,V_r,V_s)$



Network of physical components

Modeling CPS – Structural Equational Modeling



$C\, v_C' = i_C$

$v_R = R\, i_R$

$L\, i_L' = v_L$

$v_C = v_{C1} - v_{C2}$

$v_R = v_{R1} - v_{R2}$

$v_L = v_{L1} - v_{L2}$

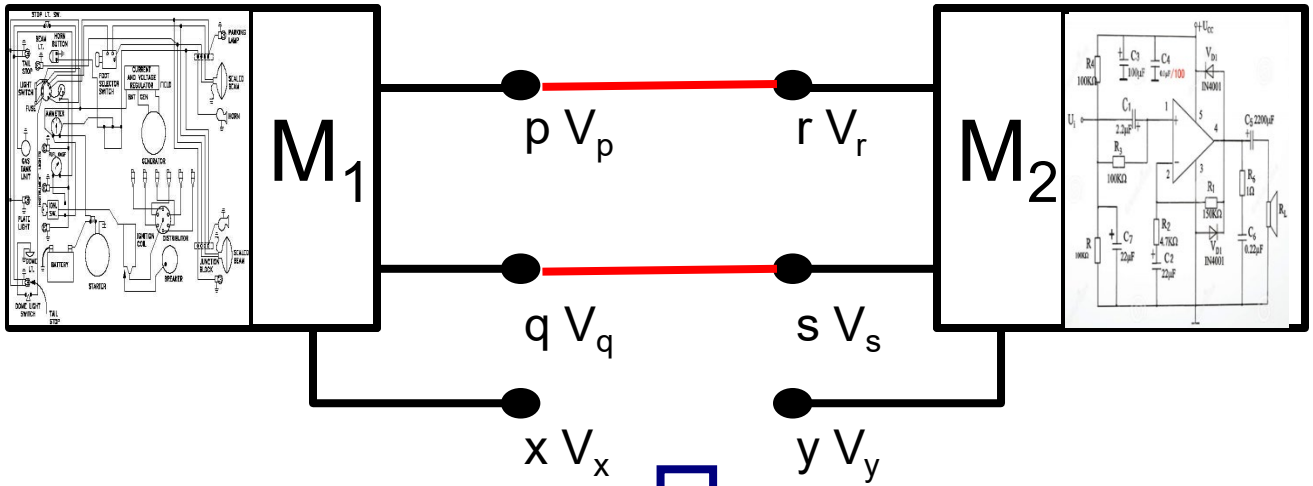
$i_C = i_R = i_L$

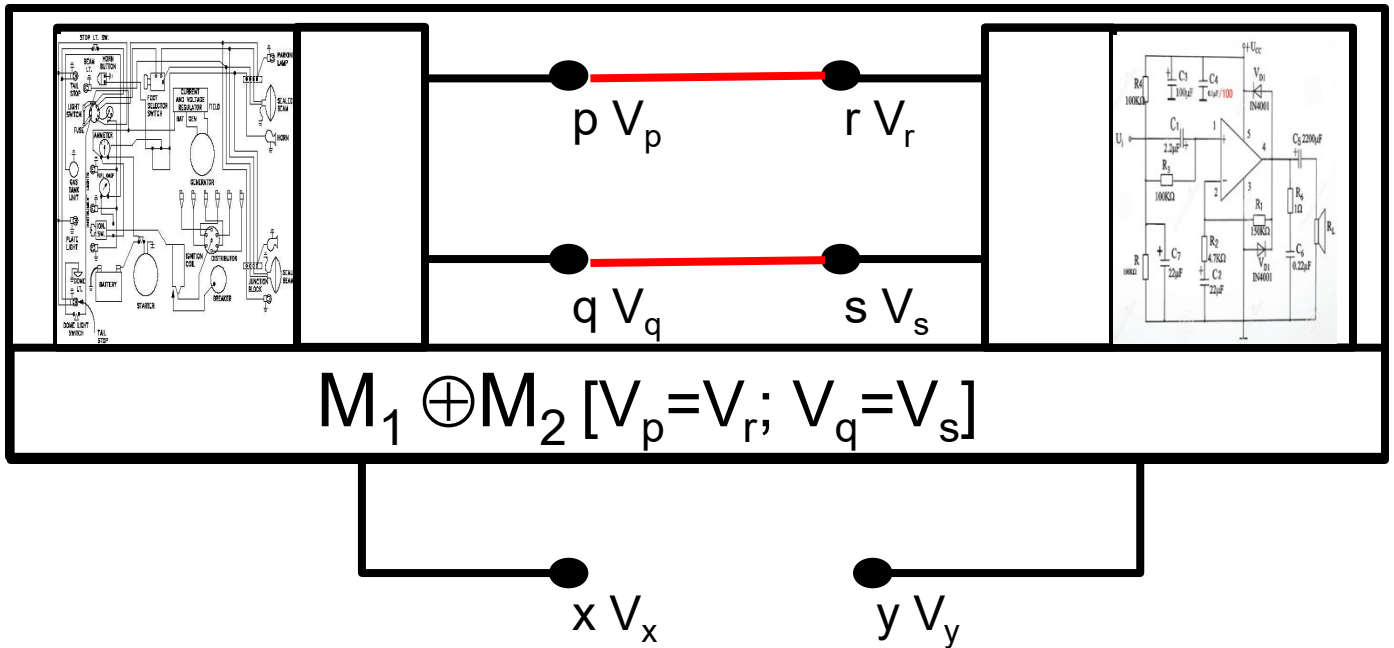
$v_{C1} = v_{R2}$, $v_{C2} = v_{L1}$, $v_{L2} = v_{R1}$

Problems addressed:

- Given a physical system described as a network of interacting elements, is there a systematic approach to get faithful equational models?
- What is the value of techniques such as Bond Graphs, Linear Graphs?
- Is there a concept of faithfulness of modeling and how it can be formalized ?
- Modular specification – meaningful composition of equational specifications
- Abstraction and reduction techniques for equational models

Cyber physical Systems – Modularity



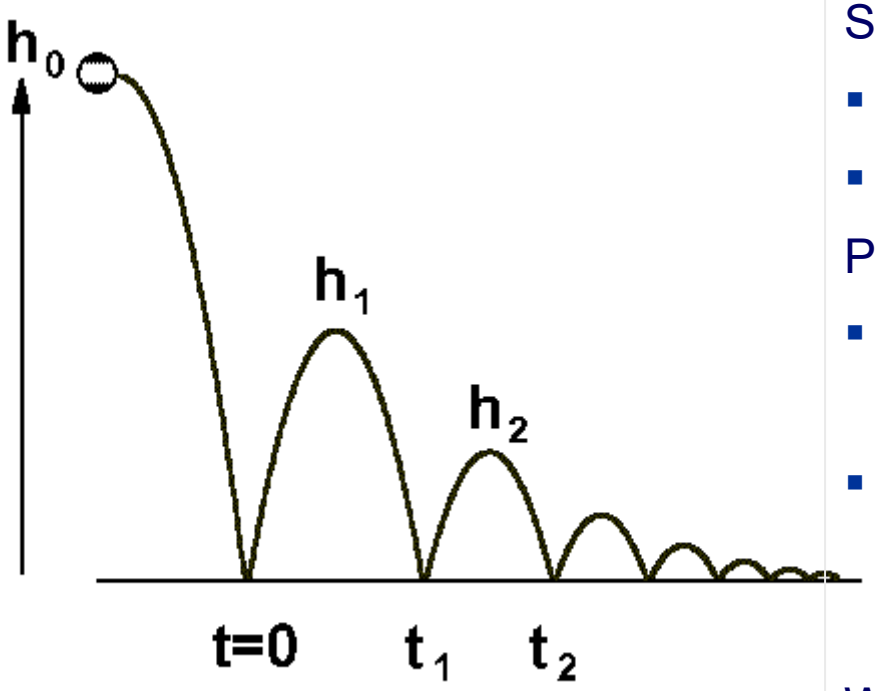


If M1and M2 are well-formed is the resulting component well-formed?

Modeling CPS – Zeno behavior

One difficulty in studying semantics of CPS is the existence of converging sequences of discrete events

- For a ball falling of an attitude h_0 and losing a percentage of its speed due to for non elastic shock with the ground
- The physical model is described by : $v' = -g$, $x'=v$, $v_{i+1}=0.8\ v_i$, $t_{i+1} - t_i = (2/g)v_i$
- $\lim_{tn\to\infty} (t_n-t_0)$ is bounded, so time will not exceed the Zeno point



Simulators with fixed integration step may

- either overshoot the limit
- or not reach convergence

Proving nonZenoness is undecidable

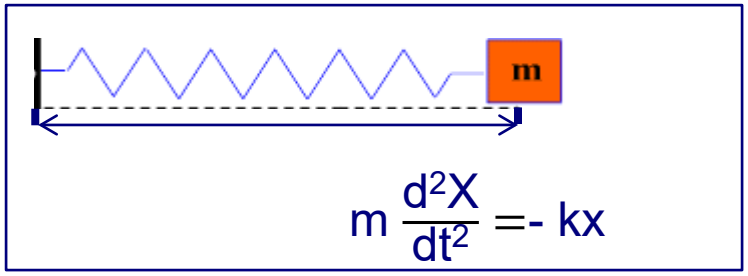
- requires discovery and application of an induction hypothesis
- see results for the verification of nonZenoness for timed automata

We need practicable theory to detect Zeno behavior

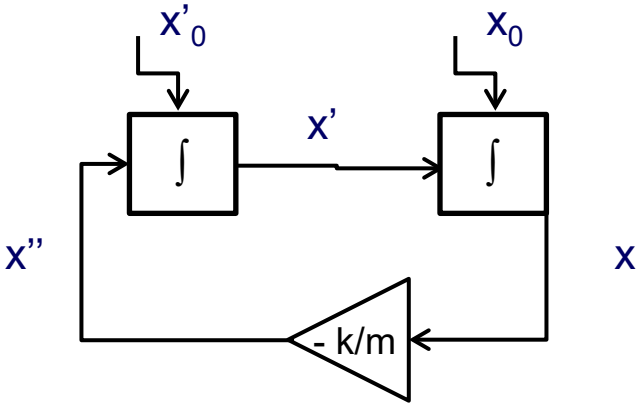
Modeling CPS – Continuous Data Flow Models

From an equational specification of a PS a Data Flow Network (block diagram) may be derived

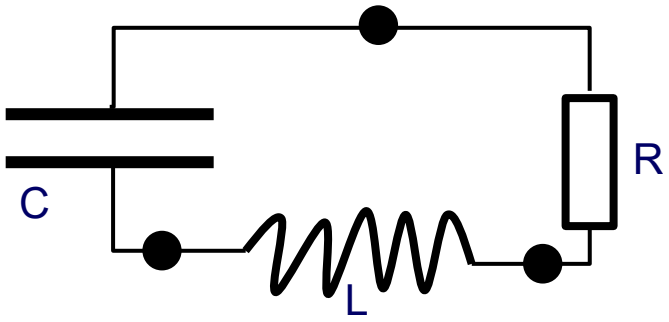
- The operators transform input flows into output flows – they are intrinsically parallel
- Integrators have an additional initialization input that determines the initial value of the integrated variable.
- Translation into data flow networks is the first step toward discretization



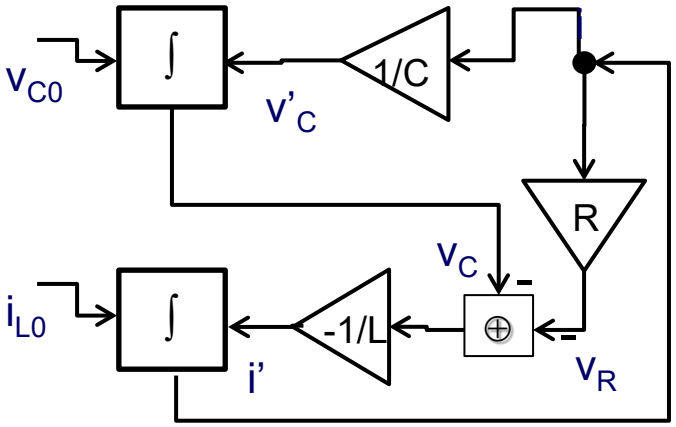
Mass-spring system



Continuous data flow model



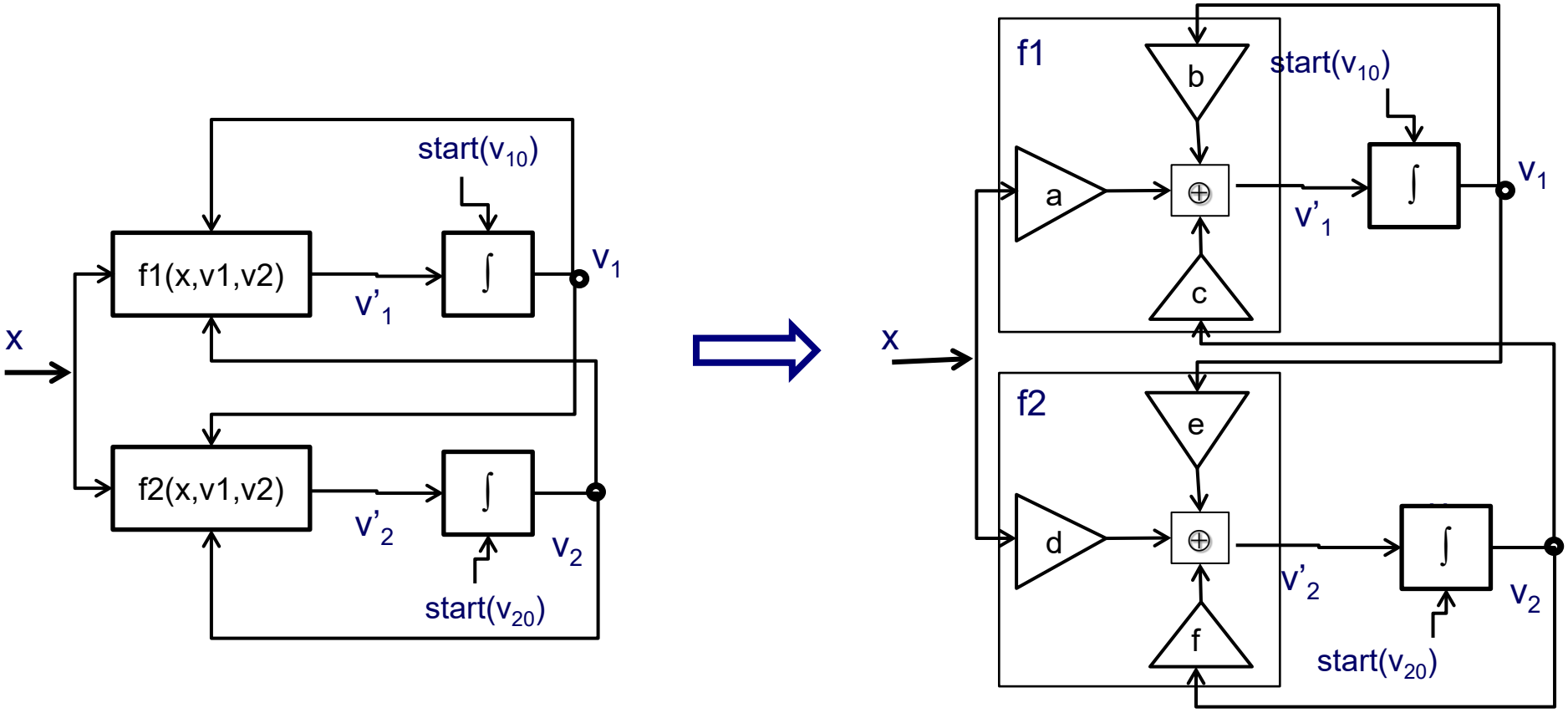
RLC Circuit



Continuous data flow model

Modeling CPS – Continuous Data Flow Models

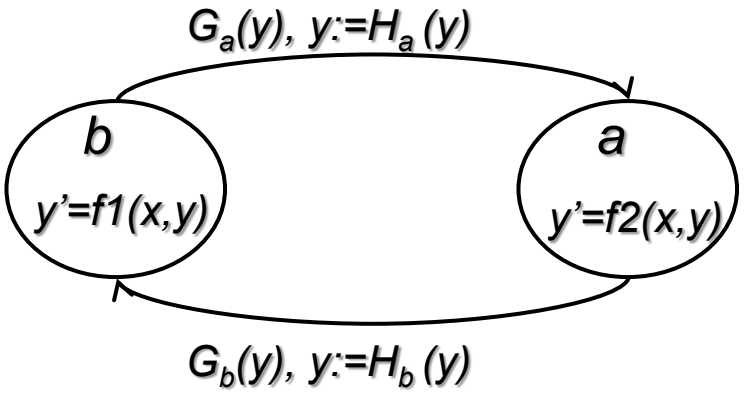
With any system of explicit DE of the form $\{v_i' = f_i(V, X)\}_{i=1..n}$ we can trivially associate a data flow network with n integration loops. The construction is compositional and to some extent incremental



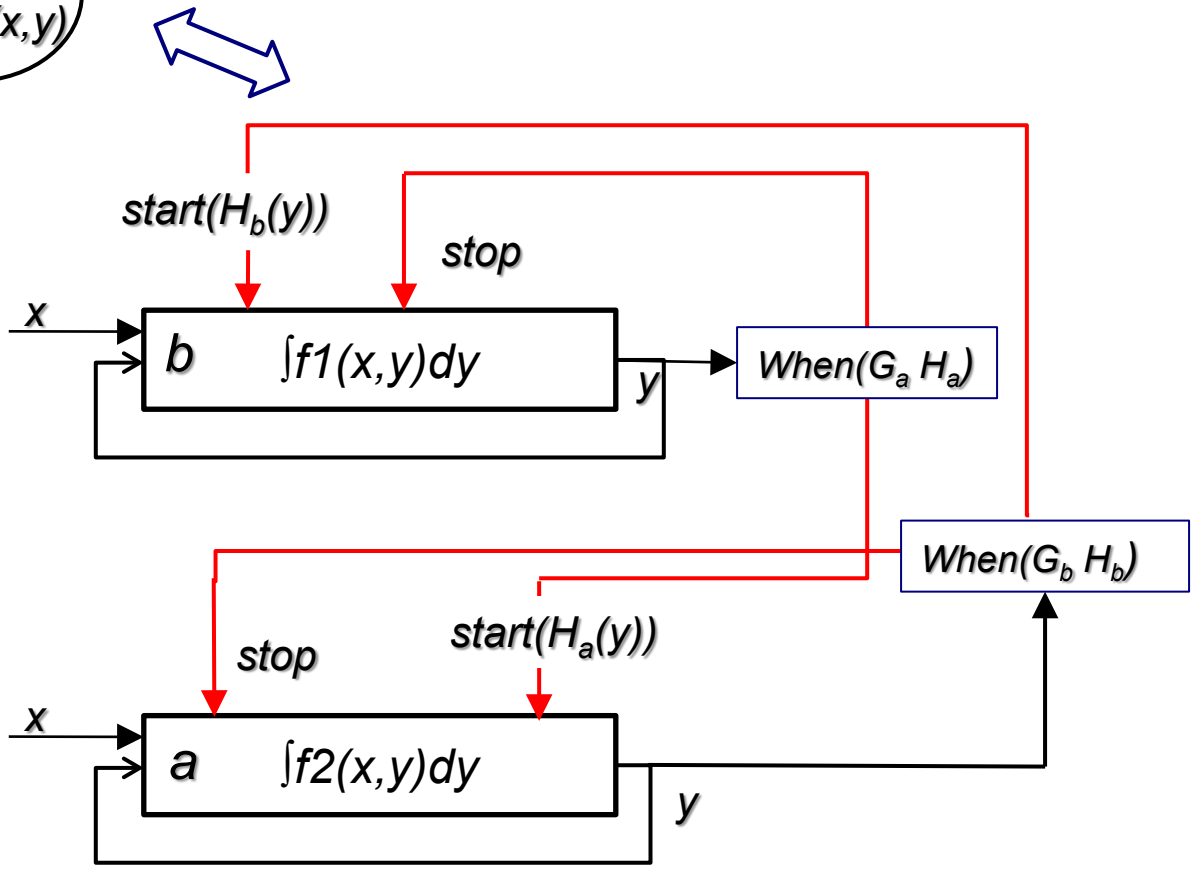
Continuous data flow models for
v1'= f1(x, v1,v2)= ax+bv1+cx2 v2'=f2(x, v1,v2)= dx+ev1+fv2

Modeling CPS – Hybrid Models

Hybrid Automata vs. Continuous Data Flow models with *When* operators



HYBRID AUTOMATON



HYBRID DATA FLOW NETWORK

Discretization – Discretization Algorithms

Find easy-to-check criteria for executability:

- the model can be assigned a *causality* relation in a unique manner
- the model does not involve closed loops without integrators.

Learn from numerical analysis techniques. Numerical solvers combine different algorithms:

- some numerical integration schemas for stiff problems,
- of different consistency order, with the ability to increase the order according to the local error,
- some local error estimator, usually using formulae embedded in the above schema,
- a time step size adaption heuristics that keeps the local error below the tolerance, and
- some event detection and “hot” restart mechanism to deal with reset equations.

Define notions:

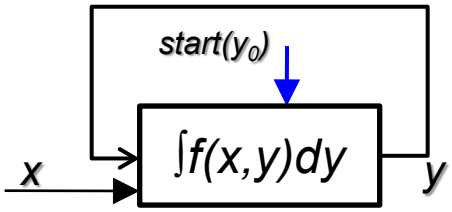
- what can be a measure of quality of approximation?
- how can be defined a notion of safety of approximation -- we do not discard when integrating critical discrete events

Discretization – Discrete Data Flow Models

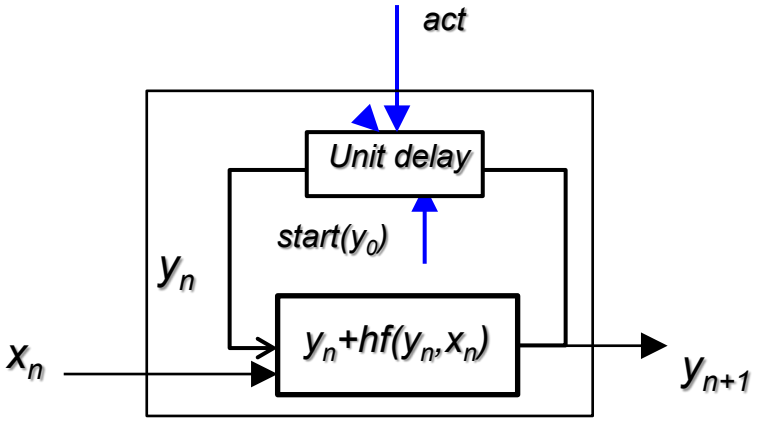
- By applying the Euler method, the solution of
  $y'(t)=f(t,y(t))$ with $y(t_0) = y_0$
  can be approximated by the iterative computation $y_{n+1} = y_n + h\, f(t_n, y_n)$ involving a time sequence $t_{n+1} = t_n + h\ t_i$

- A discretized system can be obtained by replacing each integrator by a discrete iterative program that applies a specific discretization technique.

- A discrete data-flow component has input and output data ports and an event port *act*. The event *act* triggers the cyclic computation of *f*. It plays the role of a *logical clock*.



Continuous dataflow model for
y'=f(x,y) with y=y_0



Discrete dataflow model

Discretization – Discrete Data Flow Models

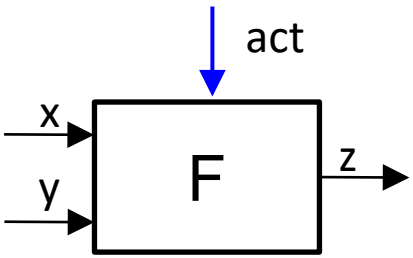
Are purely functional and do not directly support discrete events and non determinism e.g. Lustre, Simulink

Discrete data flow components:

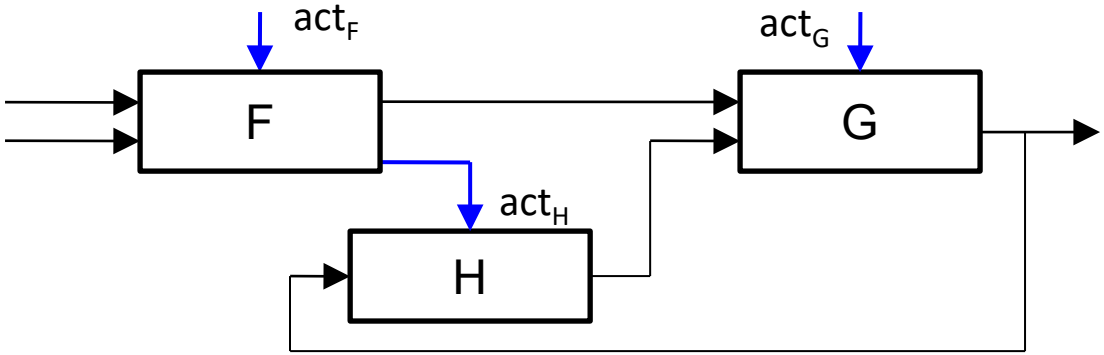
- have input and output data ports and a discrete event *act* .
- are characterized by a function F that transforms cyclically input data values into output values.
- the event *act* plays the role of a logical clock. At **each step** *t* the inputs *x* and *y* are updated and an output *z* is produced -- *z(t) =F(x(t),y(t))*

Discrete data-flow network:

- the interconnection of discrete data flow components.
- data output ports of a component are connected to data input ports of other components.
- *act* events can either external inputs or generated by using specific functions that generate events from data streams.



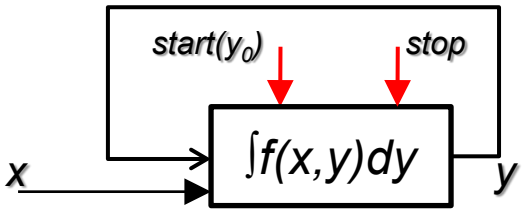
Discrete dataflow component



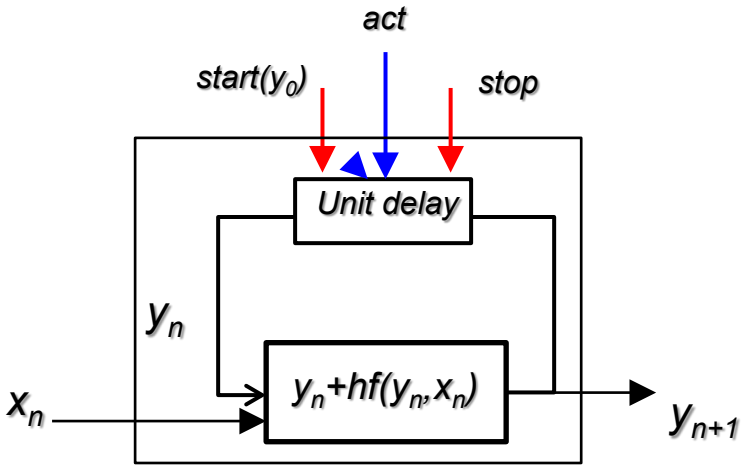
Discrete data flow network

Discretization – Discretized Hybrid Data Flow Models

- Are obtained by replacing each integrator in a hybrid data flow network by a discrete iterative program that applies a specific discretization technique.
- Discrete data flow components have in addition to *act* the external events *start* and *stop*.



Hybrid data flow model

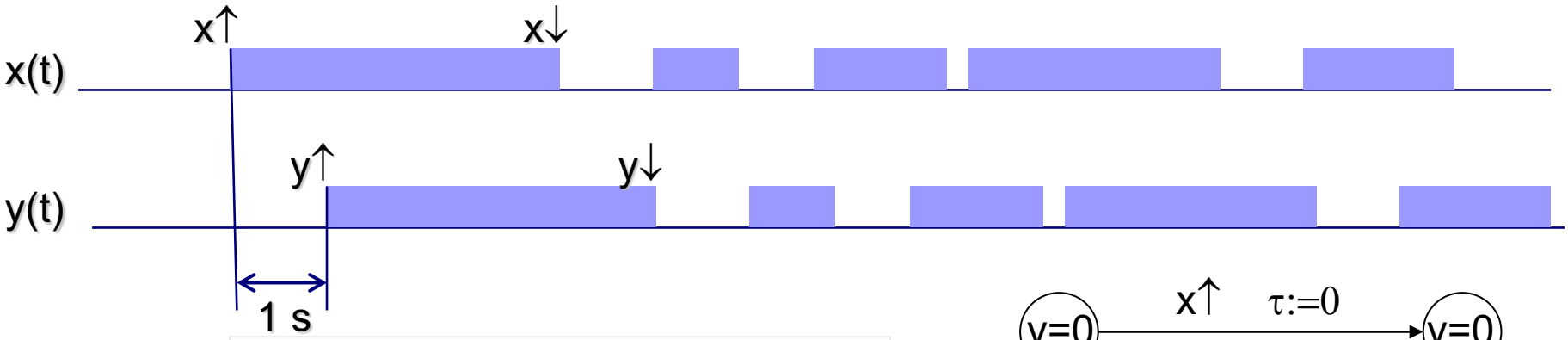


Discretized hybrid data flow model

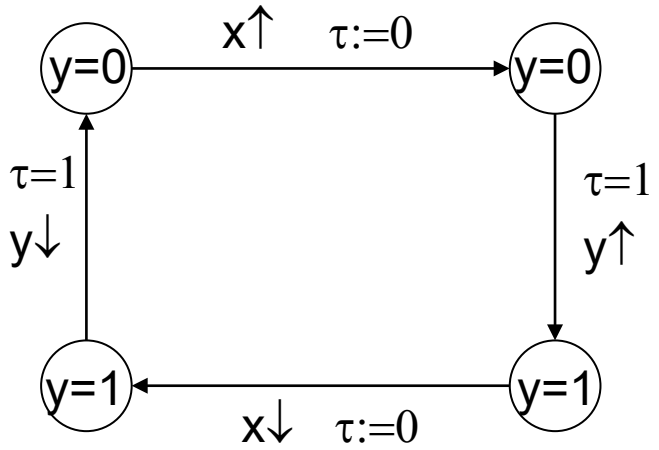
Discretization – Synchrony Assumption

Mathematically simple does not mean computationally simple!

- The continuous data flow model may be sensitive to input changes even if they occur at very close times
- There can be an infinity of changes in a finite interval and this cannot be modeled computationally. There is no finite state computational model equivalent to a unit delay!

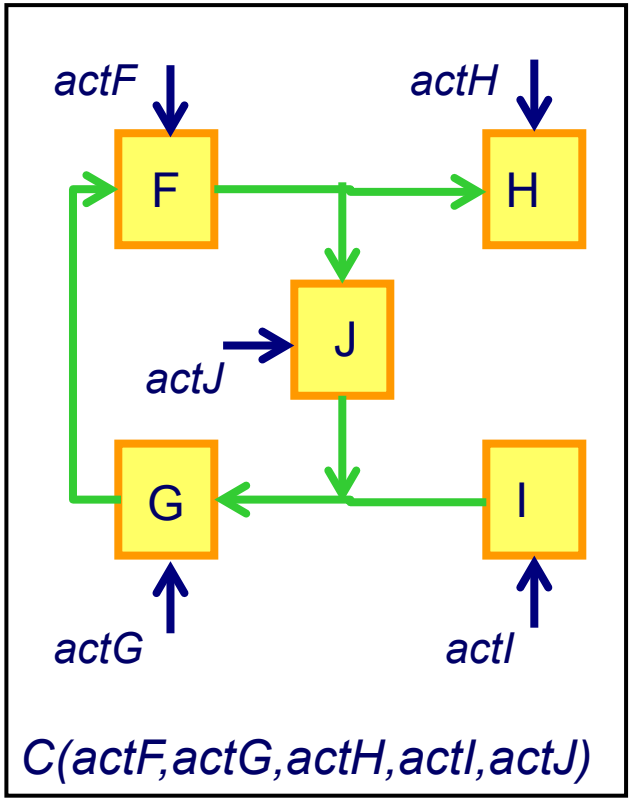


Equivalent timed automaton, provided that the distance between two consecutive input changes is more than 1sec – (synchrony assumption)



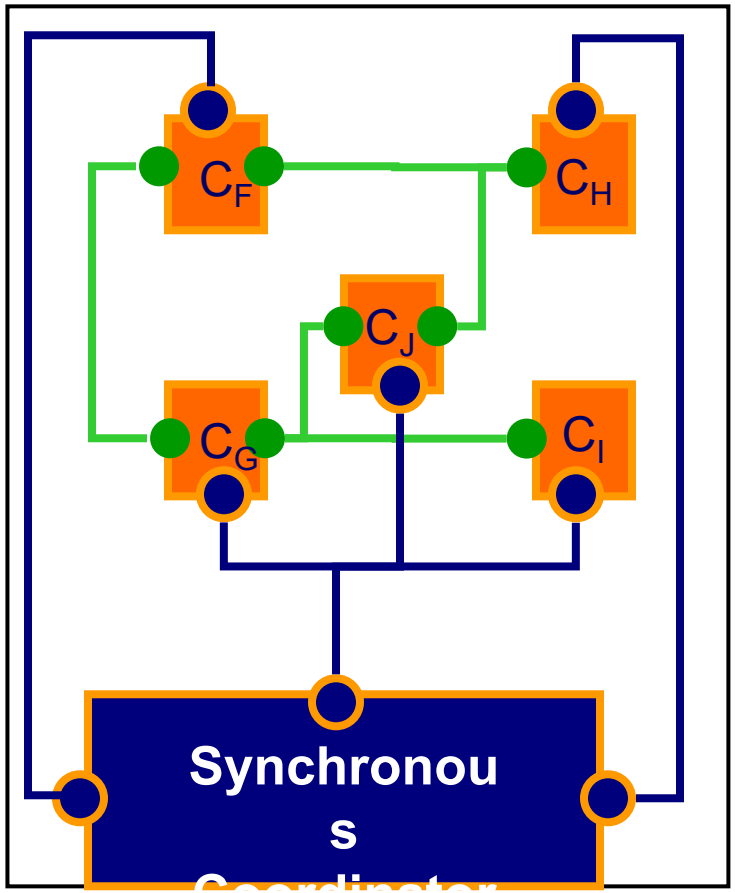
Execution and Implementation

Discrete Dataflow model in L



EMBEDDING

Event-driven Model in H



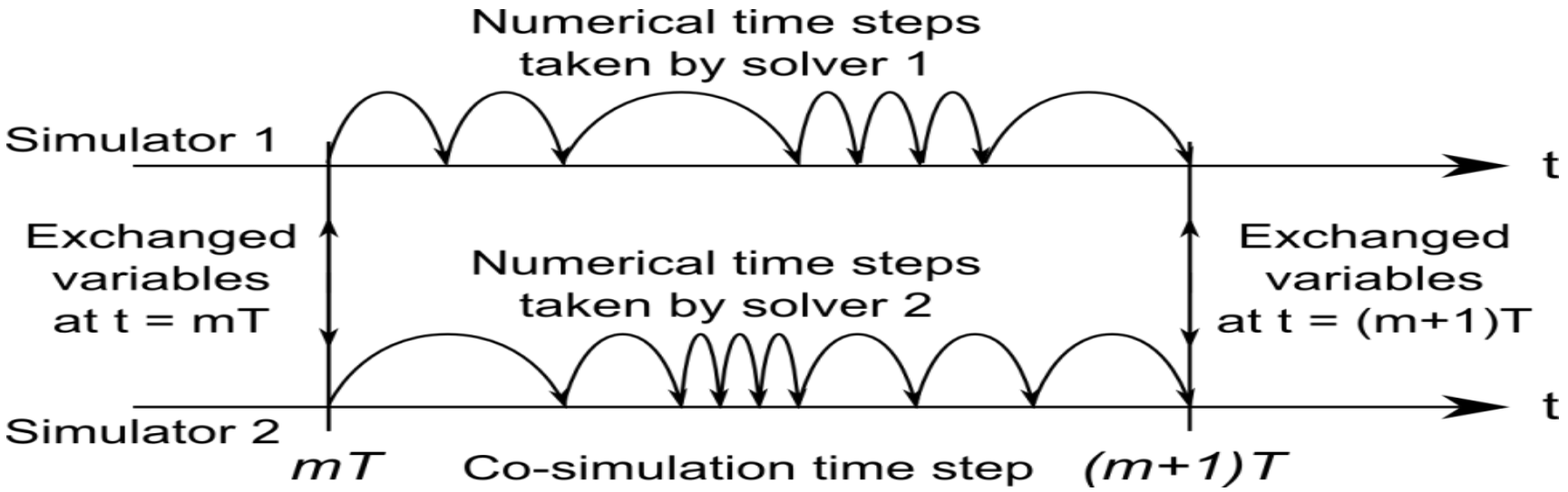
Apply embedding techniques to Discretized Hybrid Data Flow models (with *start* and *stop* events)

- modular code generation allowing separate compilation of components.

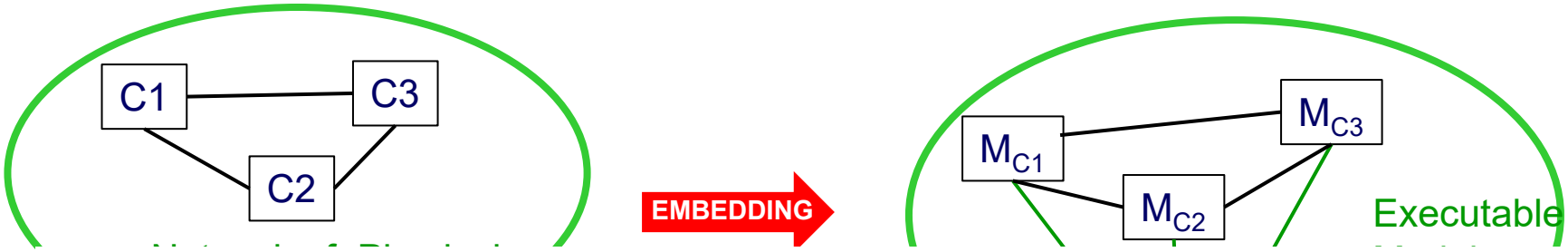
Execution and Implementation – Co-Simulation

Co-simulation is considered a more practical way of dealing with model coupling

- It consists in coordinating simulators, each subsystem model being equipped with its own (and a priori best fitted) numerical solver, and a limited set of signals are exchanged at pre-defined macro-time steps
- The simulation tools should share a common semantic representation – what is the minimal common sematic representation that is sufficient for sound integration?

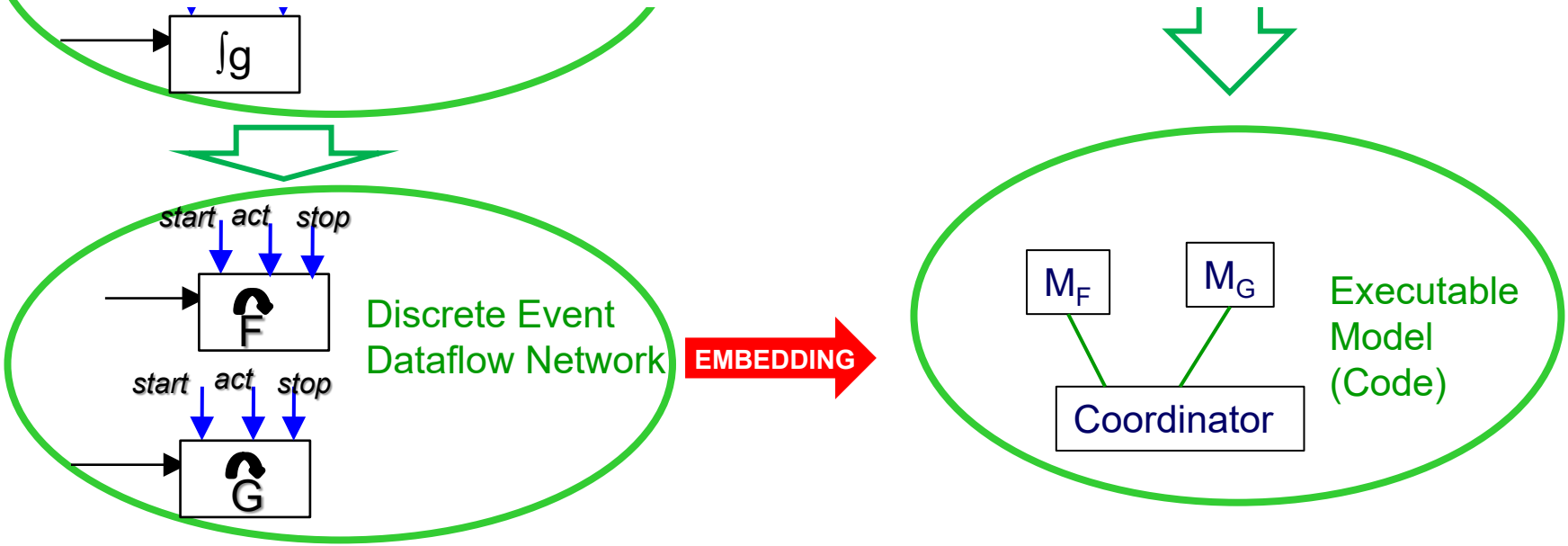


Execution and Implementation – Direct Code Generation



The rule of the simulation that I would like to have is that the number of computer elements required to simulate a large physical system is only proportional to the space-time-volume of the physical system. I don't want to have an explosion. That is, if you say I want to explain this much physics, I can do it exactly and I need a certain size computer. If doubling the volume of space and time means I'll need an exponentially larger computer, I consider that against the rules.

Richard Feynman, Int. J. Theor. Phys., Vol. 21, Nos. 6/7, 1982



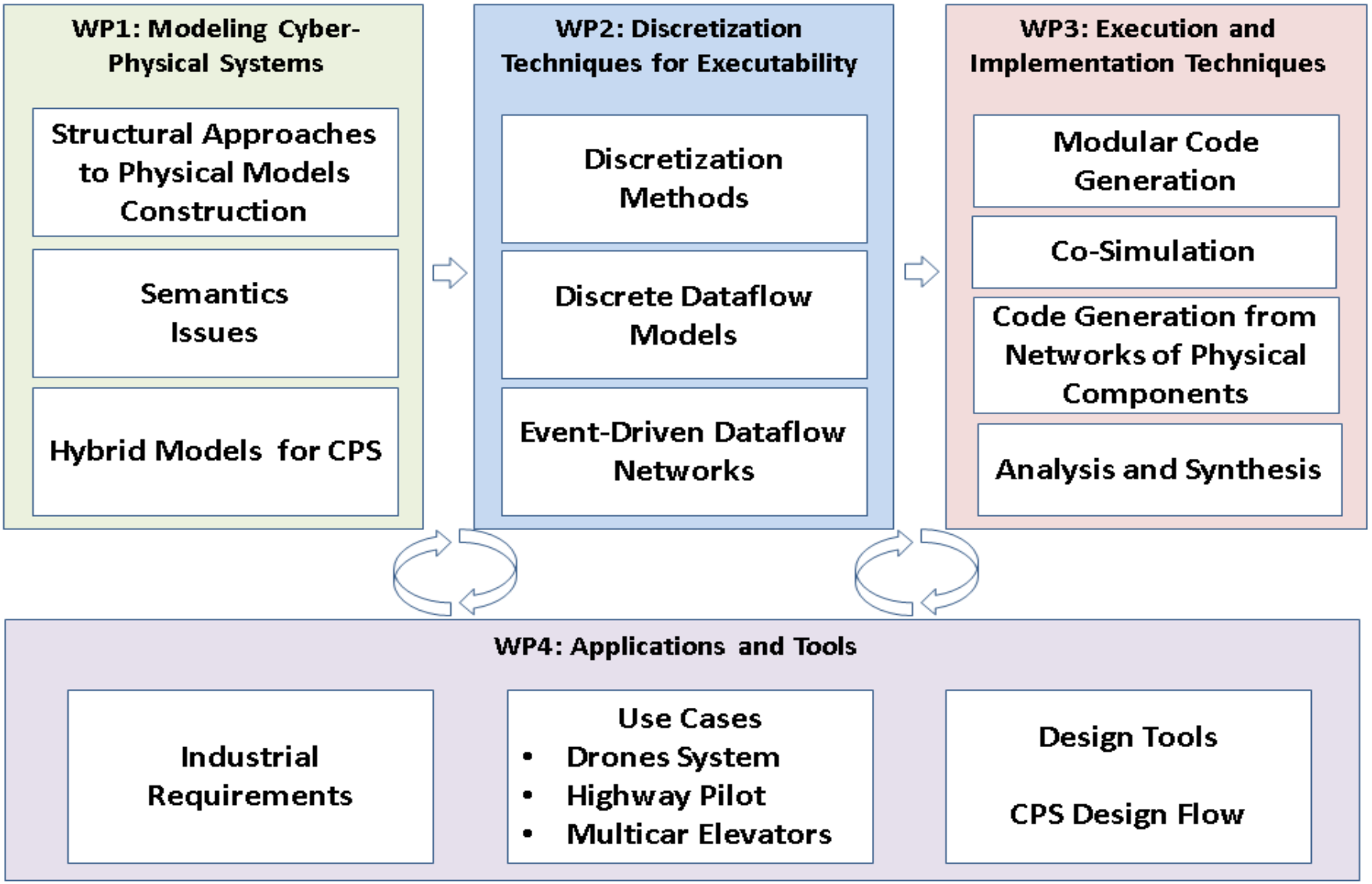
Natural vs. Physical Computing

The rule of the simulation that I would like to have is that the number of computer elements required to simulate a large physical system is only proportional to the space-time-volume of the physical system. I don't want to have an explosion. That is, if you say I want to explain this much physics, I can do it exactly and I need a certain size computer. If doubling the volume of space and time means I'll need an exponentially larger computer, I consider that against the rules.

Richard Feynman, Int. J. Theor. Phys., Vol. 21, Nos. 6/7, 1982

Compositional computability
Assuming that the physical world is discrete is it possible to compute the behavior of a network of physical components by replacing the components by the corresponding programs simulating their behavior and the connectors by protocols?
If so, with what precision and what communication overhead?

DECYPHerS – Workpackages



DECYPHerS – Developed Tools