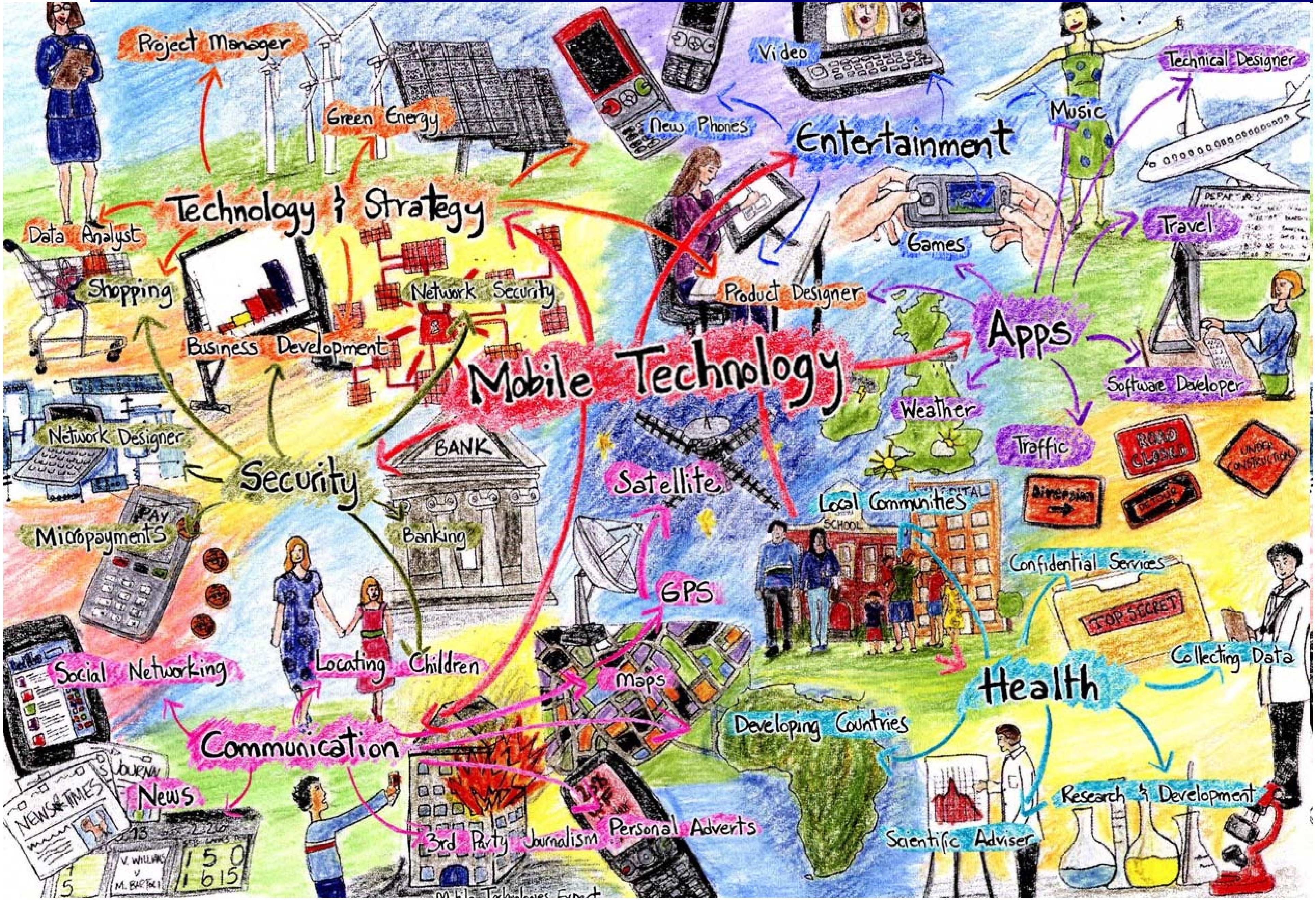
Rigorous System Design

RTSS 2014
Rome
4 December 2014

Joseph Sifakis
RiSD Laboratory EPFL

Systems Everywhere



Systems Everywhere – For a Smarter Planet

INSTRUMENTED: We now have the ability to measure, sense and see the exact condition of practically everything.

INTERCONNECTED: People, systems and objects can communicate and interact with each other in entirely new ways

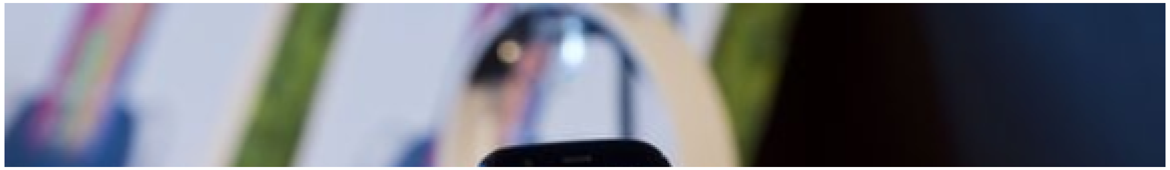
INTELLIGENT: We can respond to changes quickly and accurately, by predicting events and optimizing resources

Systems Everywhere – Mobiles Services

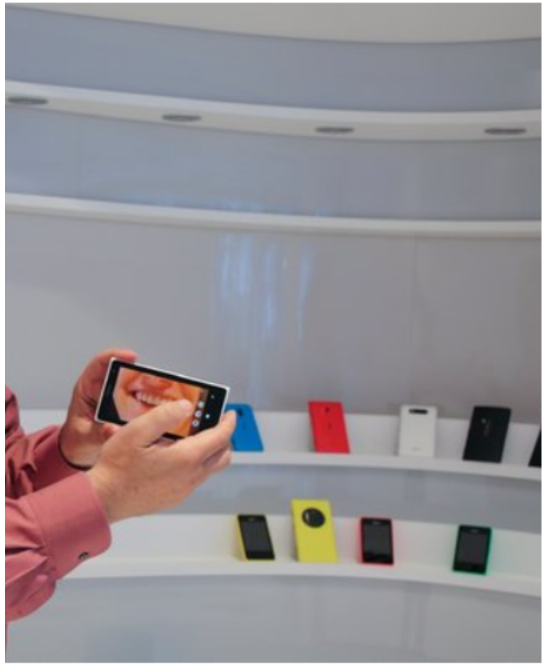
30 January 2014 Last updated at 04:29 GMT

Sha

Google sells Motorola Mobility unit to Lenovo for $3bn

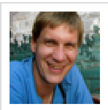


Units and Acquire Executive



Ritsuko Ando/R

oin Microsoft, setting him up as a potential successor for Steven



Google acquires digital thermostat and smoke detector maker Nest for $3.2B

by John Cook on 1/13/2014 at 1:14 pm | 12 Comments

Google is making a big bet on the future of the connected home, announcing today that it has acquired Nest for $3.2 billion in cash.

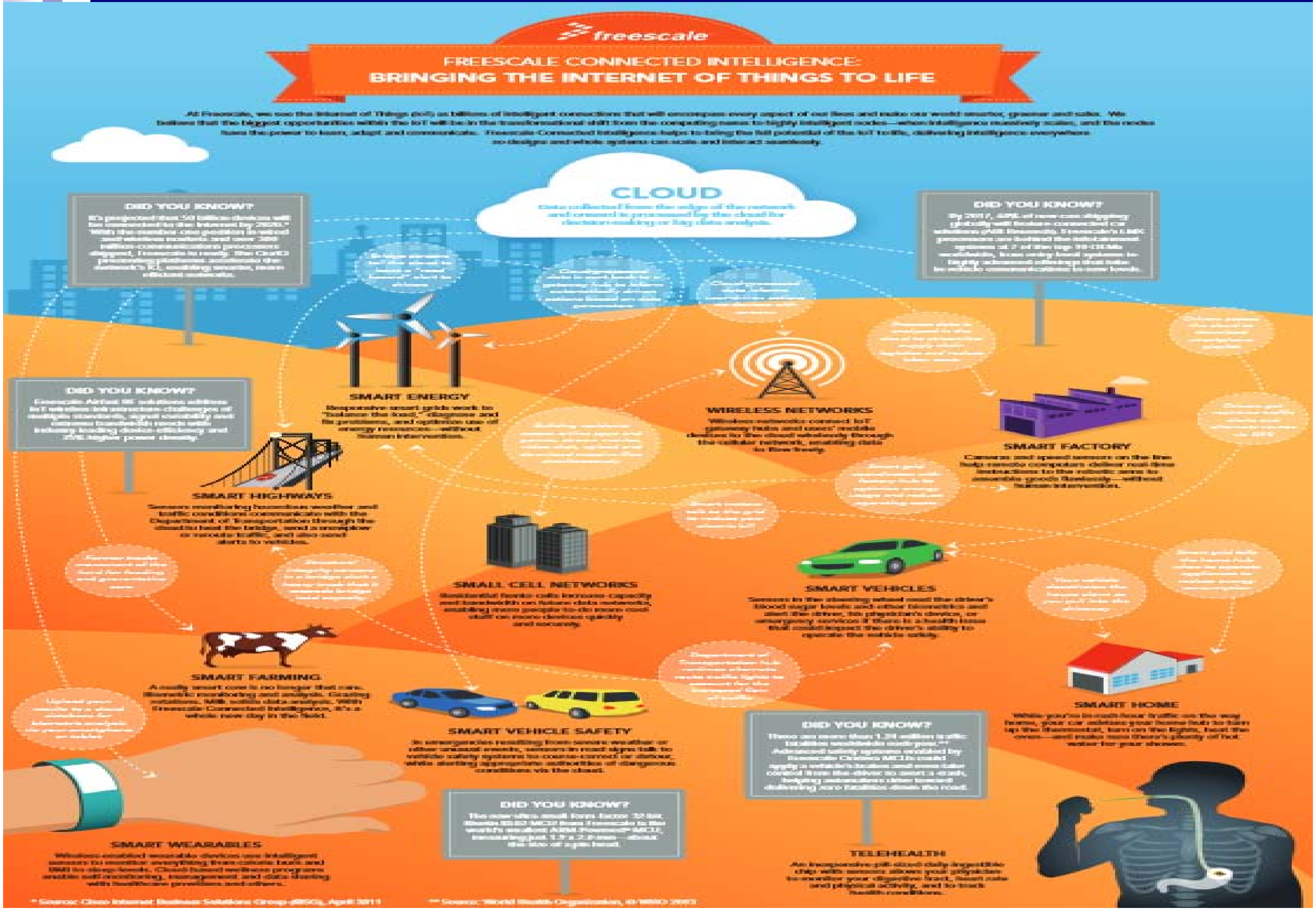
The acquisition comes just about a year after Nest raised $80 million in venture funding, reportedly at a valuation of $800 million. At the time of that deal, the company was reportedly shipping 40,000 to 50,000 of its thermostats per month. The thermostats, which allow users to regulate temperatures in a home while on the go, sell for $249.

Related

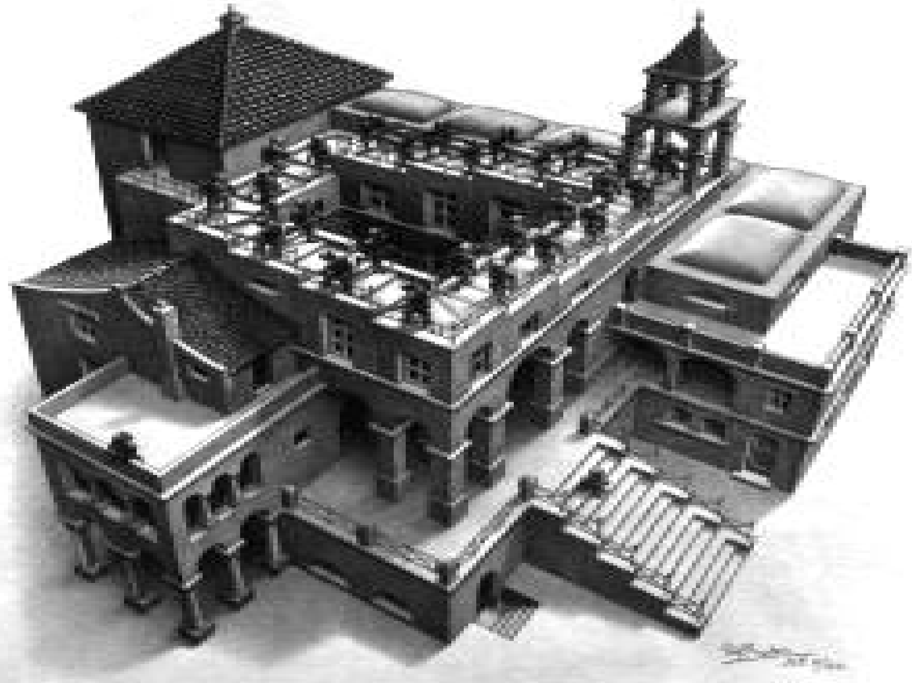
:hed an agreement to acquire
Jokia for about $7.2 billion, in
nsform Microsoft's business for a mobile era
by.

Systems Everywhere – The Google Universe

Systems Everywhere – The Internet of Things



From Programs to Systems – Significant Differences



- I/O values
- Terminating
- Deterministic
- Platform-independent behavior
- Theory of computation

- I/O streams of values
- Non-terminating
- Non-predictable
- Platform-dependent behavior
- No theory!

From Programs to Systems – New Trends

New trends break with traditional Computing Systems Engineering.
It is hard to jointly meet **technical requirements** such as:

- **Reactivity**: responding within known and guaranteed delay
  e.g. flight controller
- **Autonomy**: provide continuous service without human intervention
  e.g. no manual start, optimal power management
- **Dependability:** guaranteed minimal service in any case
  e.g. resilience to attacks, hardware failures, software execution errors
- **Scalability:** at runtime or evolutionary growth (linear performance increase with resources)
  e.g. reconfiguration, scalable services

...and also take into account **economic requirements** for optimal cost/quality

Technological challenge:
Capacity to design systems of guaranteed functionality and quality,
at acceptable costs.

- System Design

- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design

- Discussion

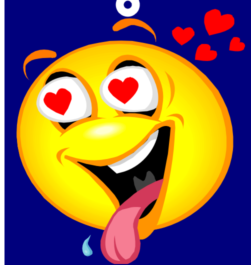
O V E R V I E W

System Design – About Design

Design is a Universal Concept!

.

Apple Pie



Proceduralization

RECIPE
(Program)

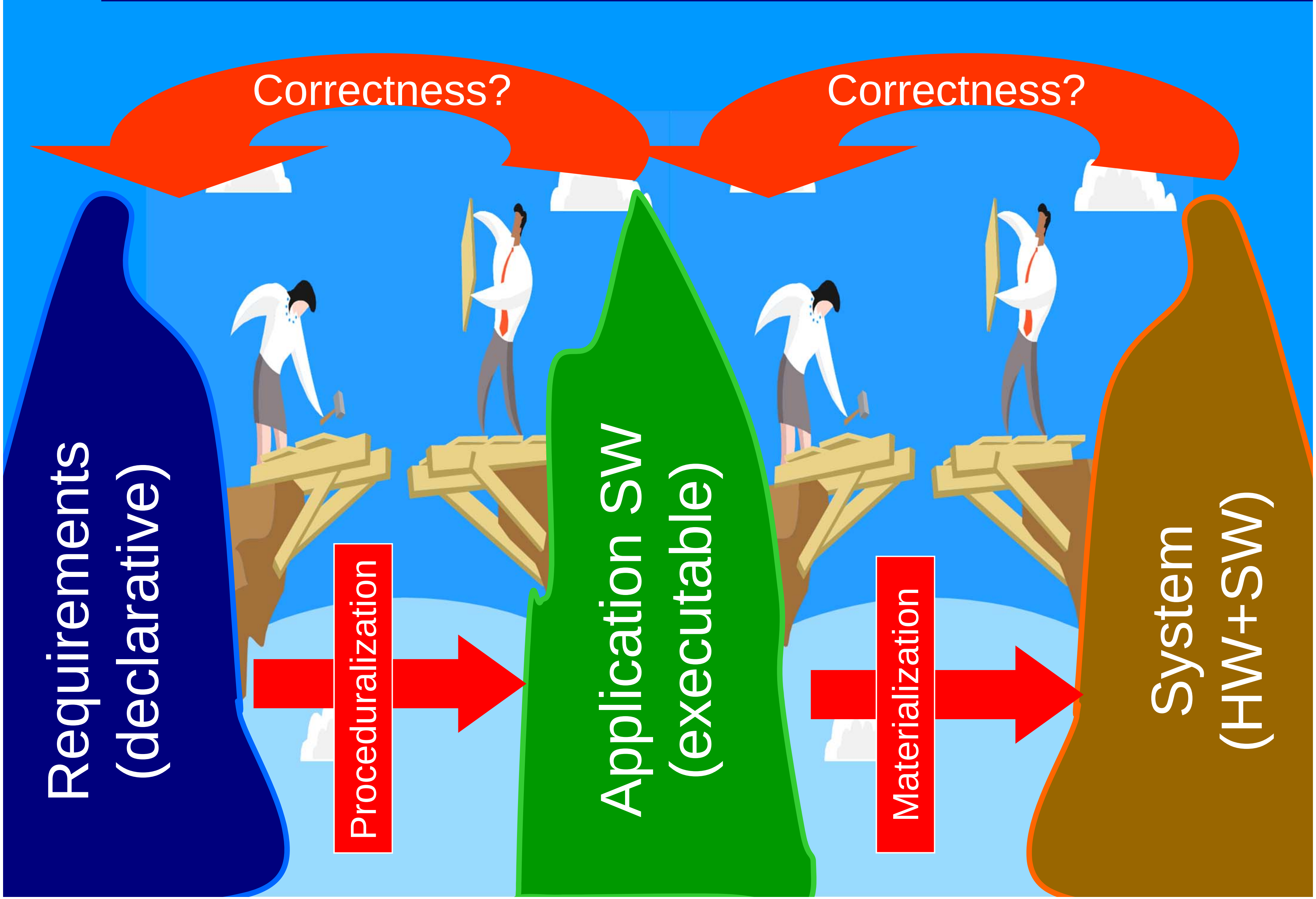
- Put apples in pie plate;
- Sprinkle with cinnamon and 1 tablespoon sugar;
- In a bowl mix 1 cup sugar, flour and butter;
- Blend in unbeaten egg, pinch of salt and the nuts;
- Mix well and pour over apples;
- Bake at 350 degrees for 45 minutes

Materialization

INGREDIENTS
(Resources)

- 1 pie plate buttered
- 5or 6 apples, cut up
- ¾ c. butter, melted
- 1 c. flour
- ½ c. chopped nuts
- 1tsp cinnamon
- 1tbsp sugar
- 1c. Sugar

System Design – Two Main Gaps



System Design – The Concept of Correctness for Systems

Trustworthiness requirements express assurance that the designed system can be trusted that it will perform as expected despite

HW failures

Design/Programming Errors

Environment Disturbances

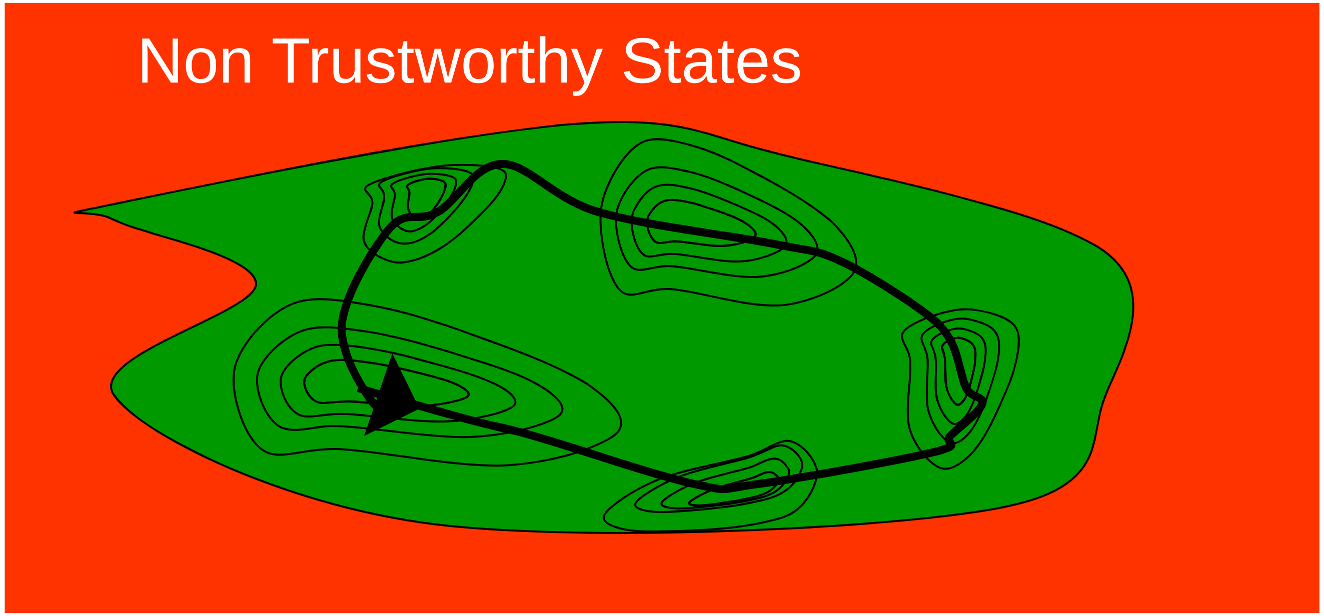
Malevolent Actions

Optimization requirements are quantitative constraints on resources such as time, memory and energy characterizing

1) performance e.g. throughput, jitter and latency
2) cost e.g. storage efficiency, processor utilizability
3) tradeoffs between performance and cost

System Design – Trustworthiness vs. Optimization

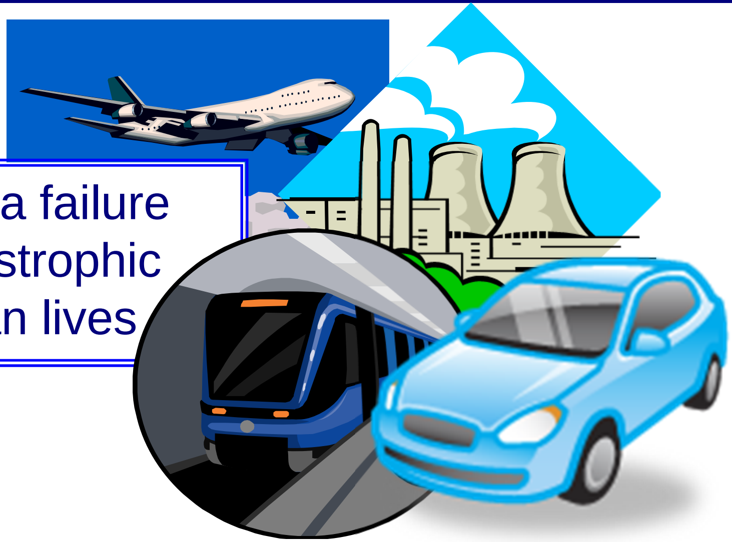
- Trustworthiness requirements characterize qualitative correctness – a state is either trustworthy or not
- Optimization requirements characterize execution sequences



Trustworthiness vs. Optimization

- The two types of requirements are often antagonistic
- System design should determine tradeoffs driven by cost-effectiveness and technical criteria

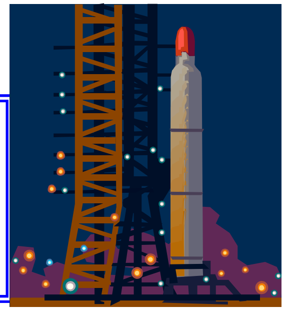
System Design – Levels of Criticality

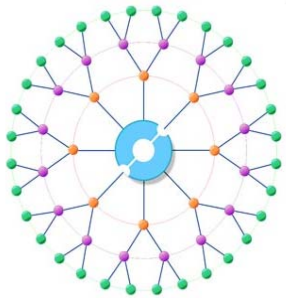


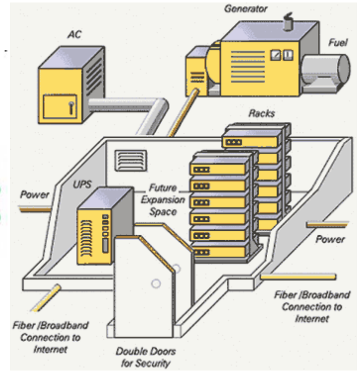
Safety critical: a failure may be a catastrophic threat to human lives

Security critical: harmful unauthorized access

Mission critical: system availability is essential for the proper running of an organization or of a larger system







Best-effort: optimized use of resources for an acceptable level of trustworthiness

System Design – Reported Failures

787 Dreamliner's safety systems failed, NTSB says

Massive cyberattack hits Internet users

By **Doug Gross**, CNN
March 29, 2013 -- Updated 1111 GMT (1911 HKT) | Filed under: Web

▮ Software Bug Led to System Failure

Shutdown of the Hartsfield-Jackson Atlanta International Airport

Toyota recalls more than 400,000 Priuses, other hybrid cars

By Blaine Harden and Frank Ahrens
Wednesday, February 10, 2010

TOKYO -- Toyota on Tuesday anno
global recall -- this time involving m
Priuses and other hybrid cars with b
on the same day that the U.S. Transp
Department said it is reviewing driv
hard-to-handle steering on the 2009-

Loss of Communication between the FAA Air Traffic Control Center, and Airplar

FDA: Software Failures Responsible for 24% Of All Medical Device Recalls

by Paul Roberts

Loss of the Mars Polar Lander

Crash of Air France Flight 447

Crash of American Airlines

Northeast blackout leaves 50M people without power, August 14, 2003

Miscalculated Radiation Doses at the National Oncology Institute

Inside the Pentium II Math Bug

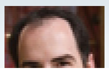
By Robert R. Collins

Explosion of Ariane 5 ▮ Flight 501

Power-Outage across Northeastern U.S. and Southeastern Canada

NEWS

Vulnerabilities Found In Banking Apps



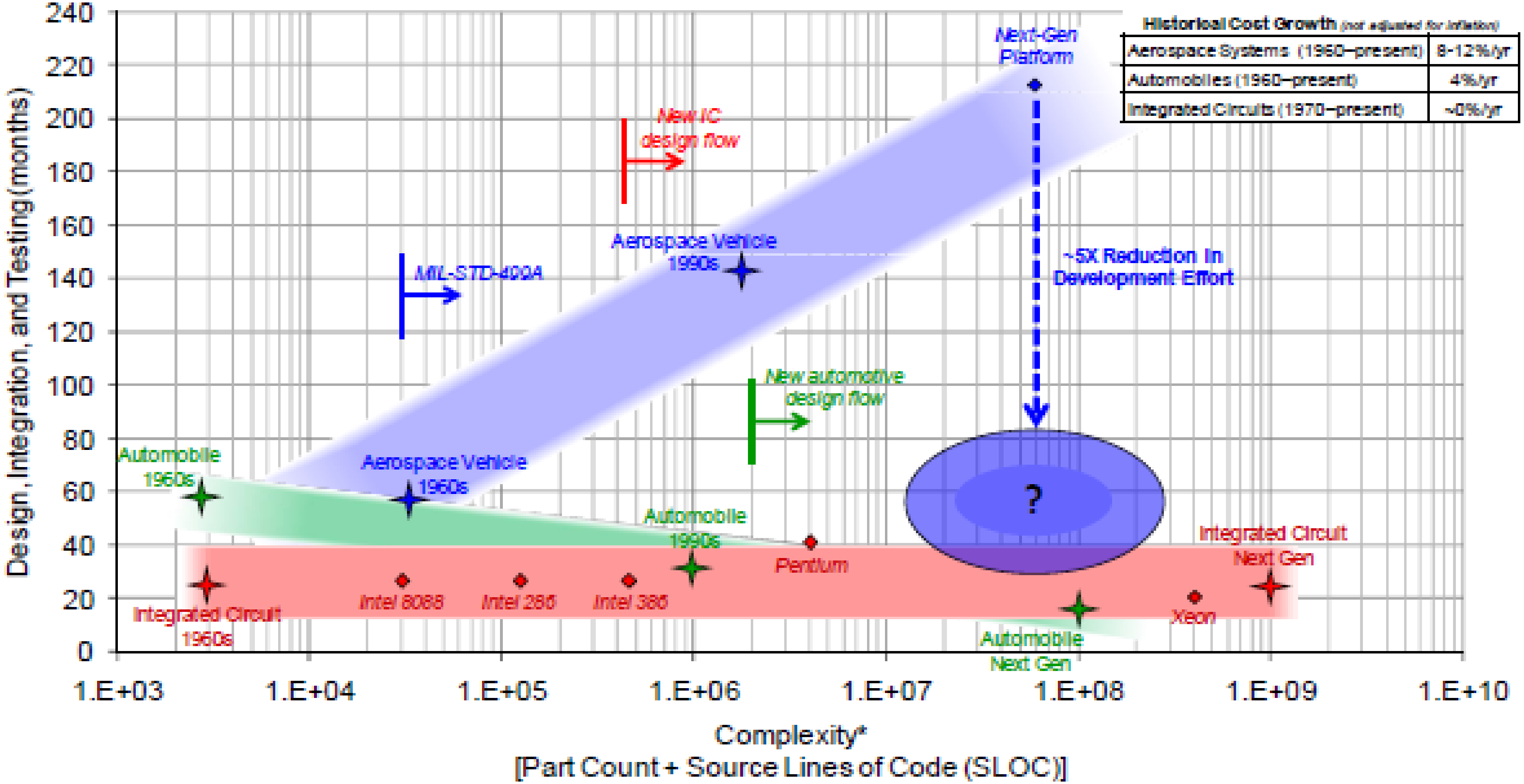
Mathew J. Schwartz

Emergency-Shutdown of the Hatch Nuclear Power Plant

System Design – The Cost of Trustworthiness

APPROVED FOR PUBLIC RELEASE. DISTRIBUTION UNLIMITED.

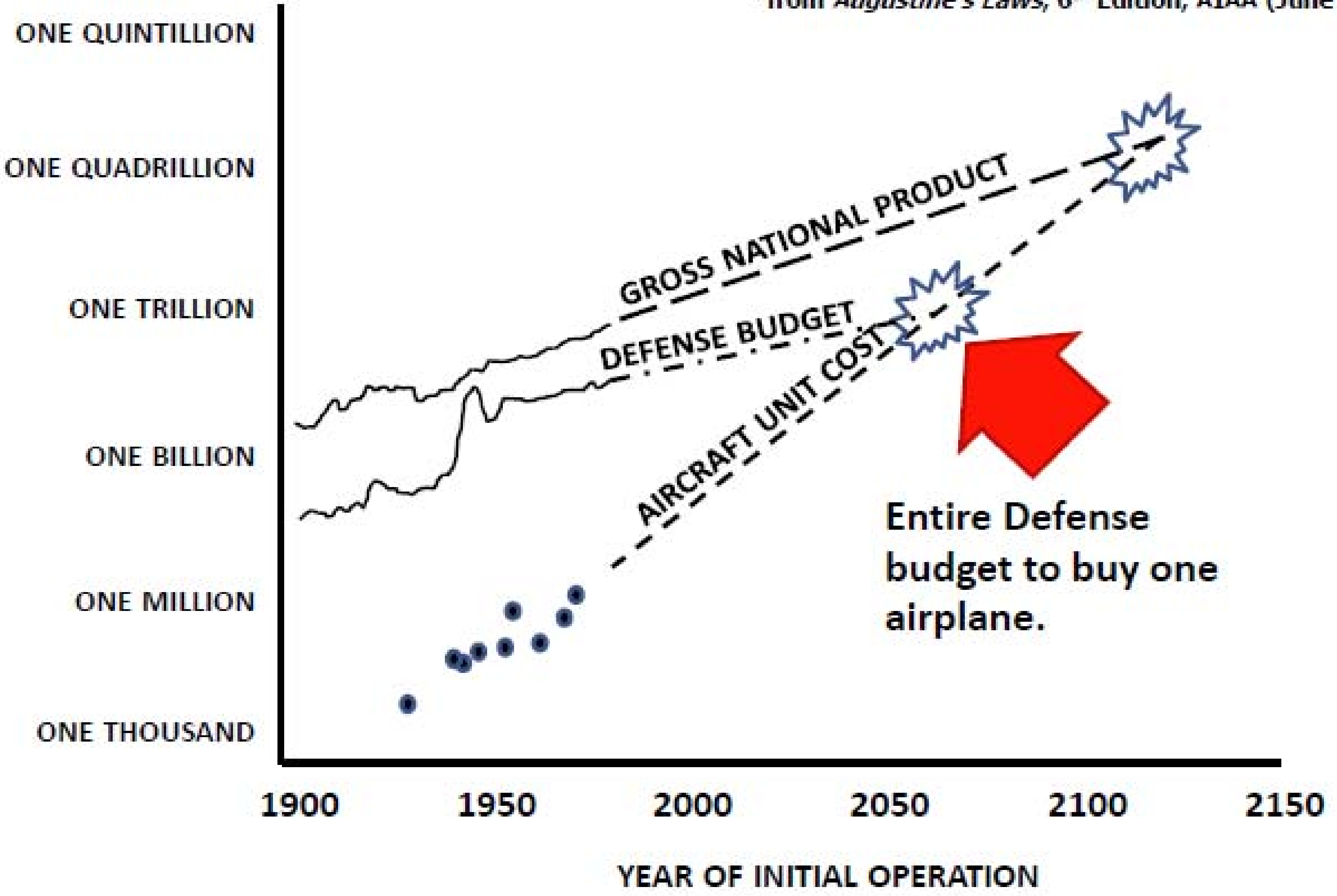
Historical schedule trends with complexity



System Design – The Cost of Trustworthiness

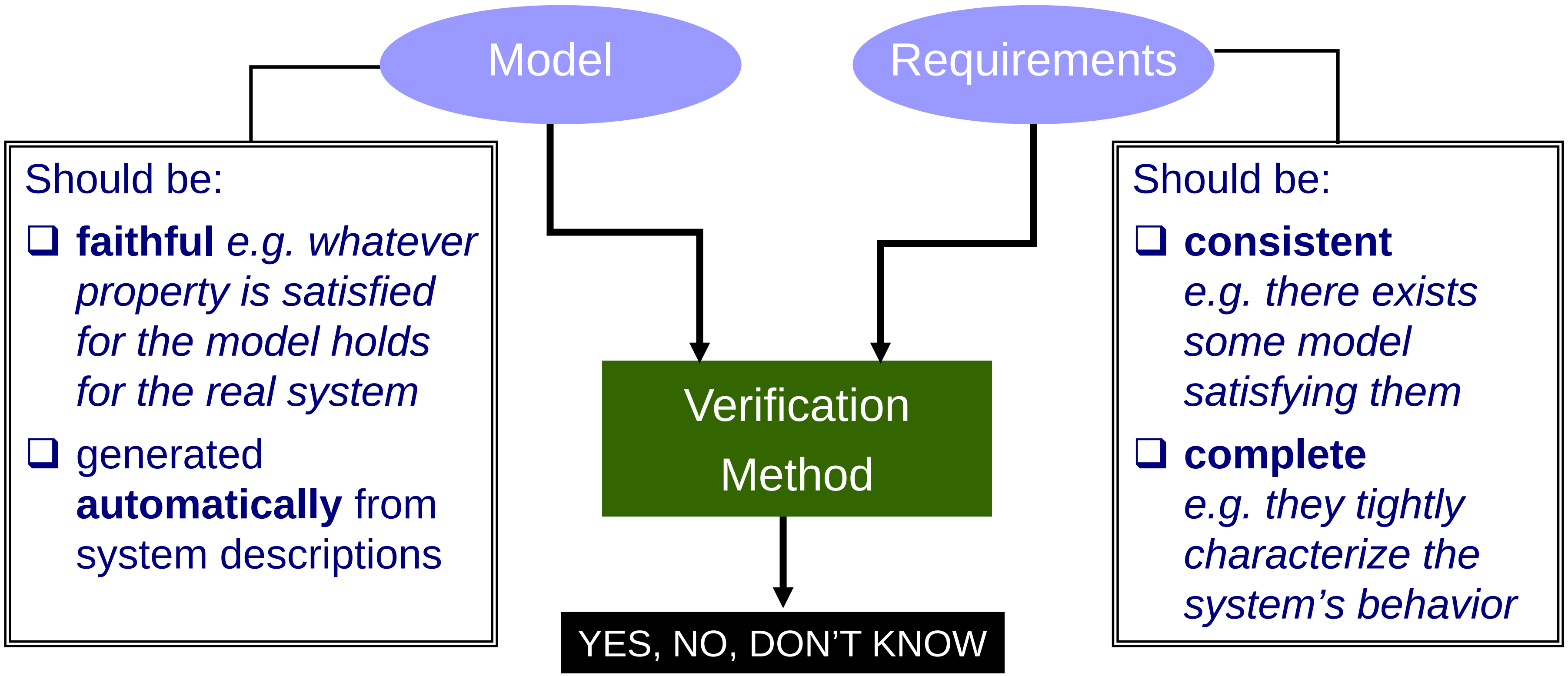
DARPA The problem

*from *Augustine's Laws*, 6th Edition, AIAA (June 1997)



APPROVED FOR PUBLIC RELEASE. DISTRIBUTION UNLIMITED.

System Design – Verification



- Present systems are not trustworthy!
- $1,000 per line of code for "high-assurance" software!

System Design – Verification

Verification techniques are applicable to global models and thus suffer from well-known limitations

- Can contribute to establishing trustworthiness for requirements that can be formalized and checked efficiently
- For optimization requirements, a more natural approach for their satisfaction is by enforcing or synthesis rather than by checking

Verification

- is a stopgap until other alternatives for achieving correctness work
- is a "speciality" of computing – no other scientific discipline gives it a such a prominent place
- a discipline is not worthy of scientific merit if predictability can be achieved only through verification

O
V
E
R
V
I
E
W

- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- Discussion

Rigorous System Design – The Concept

RSD considers design as a formal accountable and iterative process for deriving trustworthy and optimized implementations from an application software and models of its execution platform and its external environment

- Model-based: successive system descriptions are obtained by correct-by-construction source-to-source transformations of a single expressive model rooted in well-defined semantics
- Accountable: possibility to assert which among the requirements are satisfied and which may not be satisfied

RSD focuses on mastering and understanding design as a problem solving process based on divide-and-conquer strategies involving iteration on a set of steps and clearly identifying

- points where human intervention and ingenuity are needed to resolve design choices through requirements analysis and confrontation with experimental results
- segments that can be supported by tools to automate tedious and error-prone tasks

Rigorous System Design – Four Guiding Principles

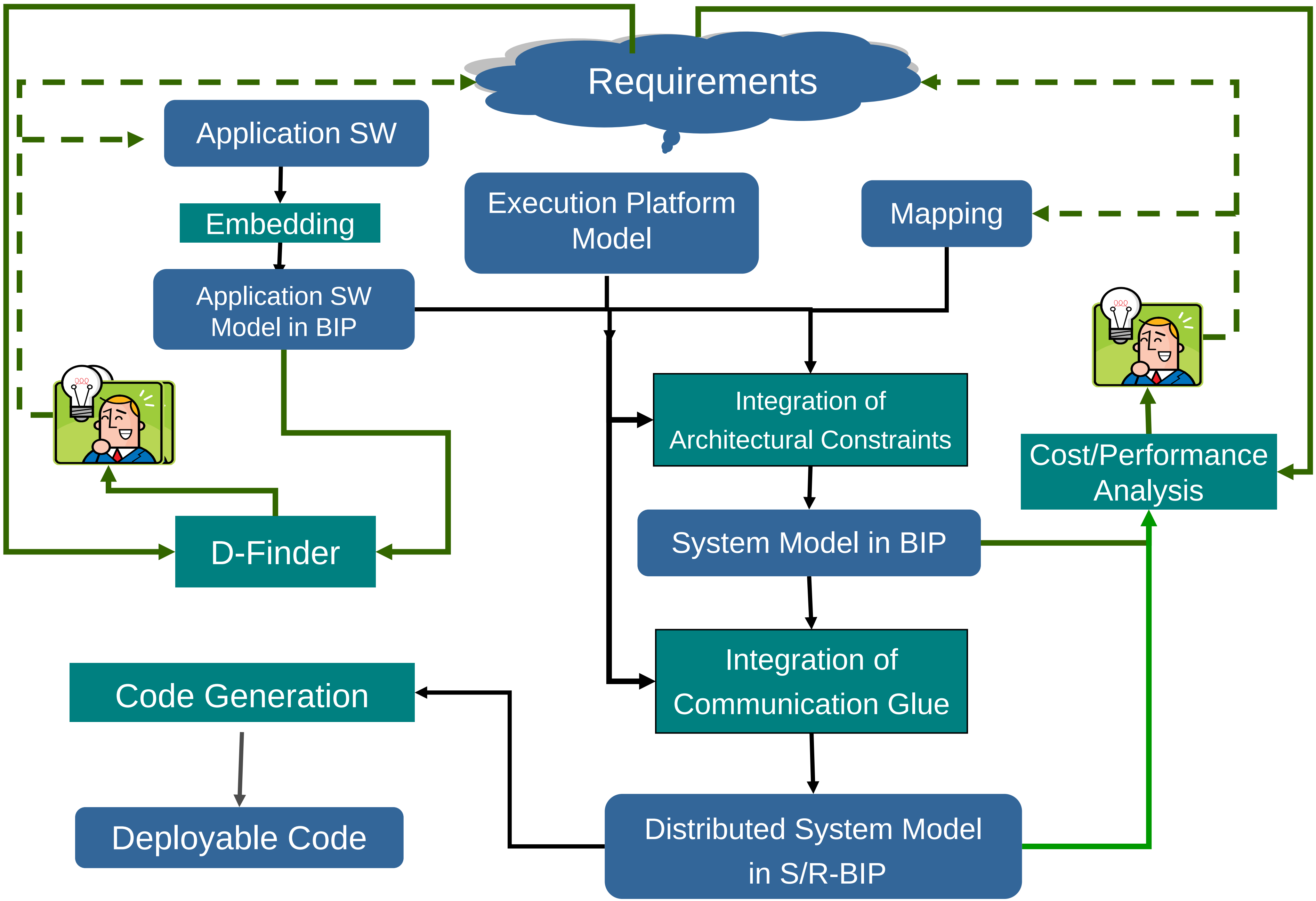
<u>Separation of concerns</u>: Keep separate what functionality is provided (application SW) from how its is implemented by using resources of the target platform

<u>Components</u>: Use components for productivity and enhanced correctness

<u>Coherency</u>: Based on a single model to avoid gaps between steps due to the use of semantically unrelated formalisms e.g. for programming, HW description, validation and simulation, breaking continuity of the design flow and jeopardizing its coherency

<u>Correctness-by-construction</u>: Overcome limitations of a posteriori verification through extensive use of <u>provably correct</u> reference architectures and structuring principles enforcing essential properties

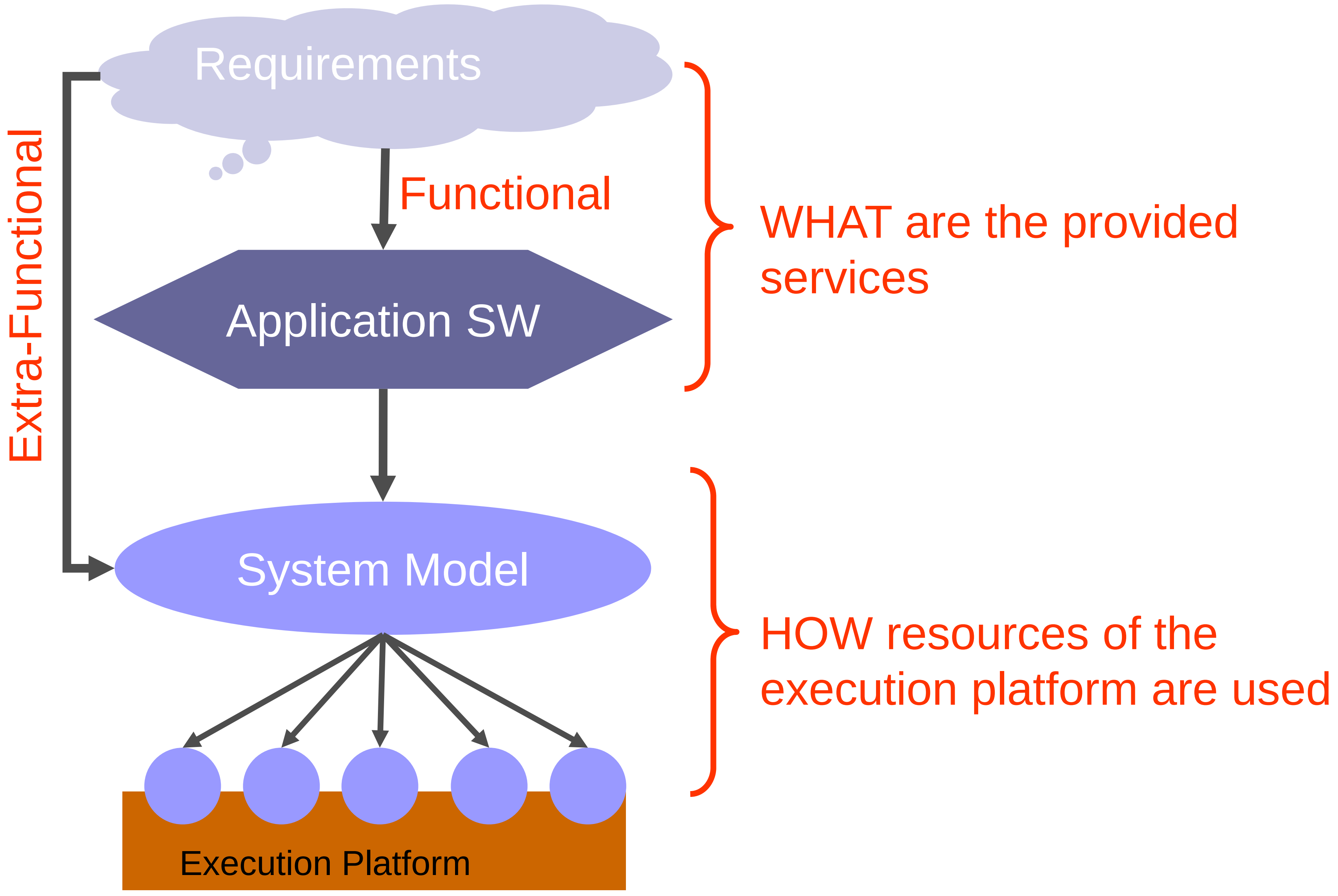
Rigorous System Design – Simplified Flow



OVERVIEW

- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- Discussion

Separation of Concerns



Separation of Concerns – From ASW to the System Model

Application SW

Time and resources are external parameters that are linked to corresponding physical quantities of the execution environment

?

System Model

Obtained by instrumentation of the ASW

- Time and resources are state variables
- Each action consumes and liberates an amount of resources explicitly specified e.g. execution times, memory, energy

Separation of Concerns – Building a System Model

Resource-Consistency: faithful modeling of physical resources

- Physical time is monotonically increasing - time progress cannot be blocked
- Model time progress can block or can involve Zeno runs – deadline miss = deadlock or time-lock.

Additional difficulties arise from discretization, in particular for distributed execution

Resource-robustness: small change of resource parameters entail commensurable change of performance

- Performance degradation can be observed for increasing speed of the execution platform – Timing Anomaly

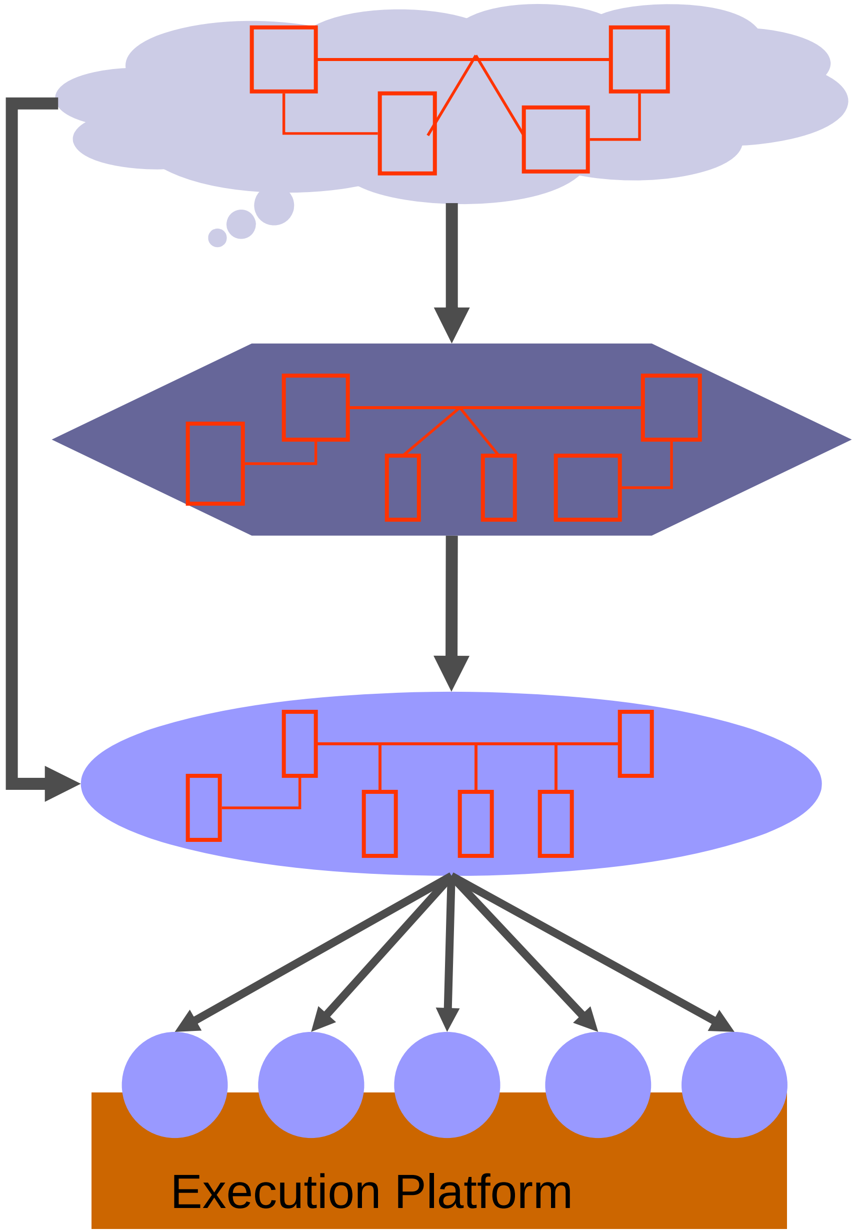
- Non determinism is one of the identified causes of such a counter-intuitive behavior

We lack results guaranteeing resource-robustness e.g. worst-case and best-case analysis suffice to determine performance bounds.

O
V
E
R
V
I
E
W

- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- Discussion

Component-based Design



- Components are indispensable for enhanced productivity
  and correctness

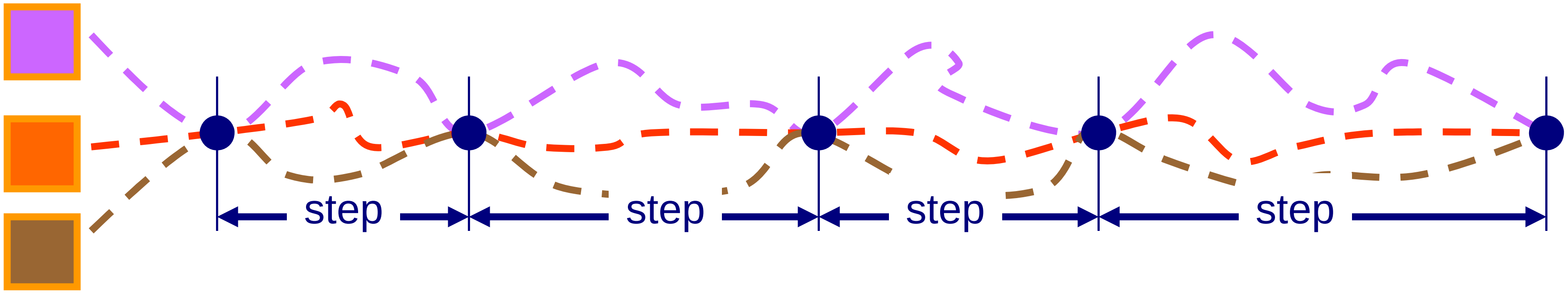
- Component composition lies at the heart of the parallel computing challenge

- There is no Common Component Model
  - Heterogeneity

Component-based Design – Synchronous vs. Asynchronous

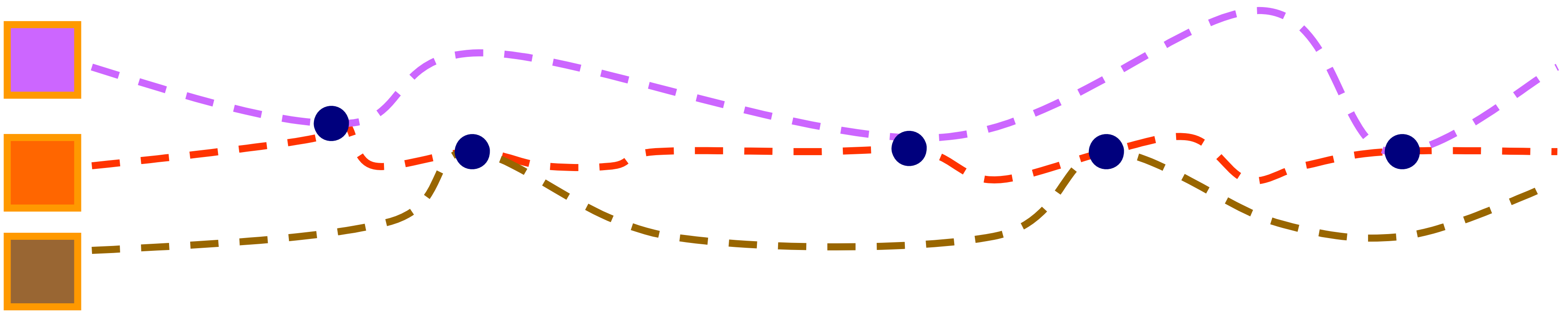
Synchronous components (HW, Multimedia application SW)

- Execution is a sequence of non interruptible steps



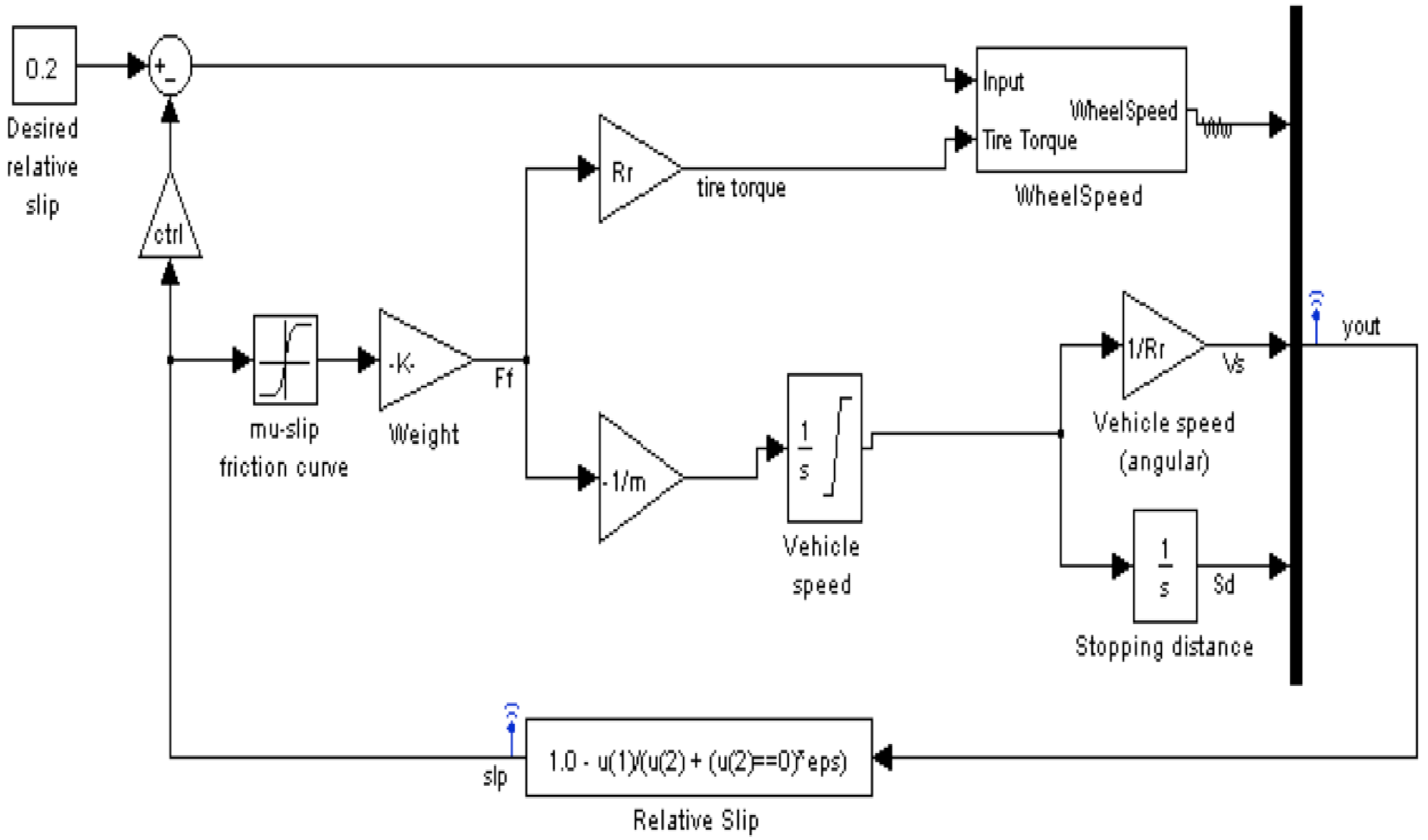
Asynchronous components (General purpose application SW)

- No predefined execution step



Open problem: Theory for consistently composing synchronous and asynchronous components e.g. GALS

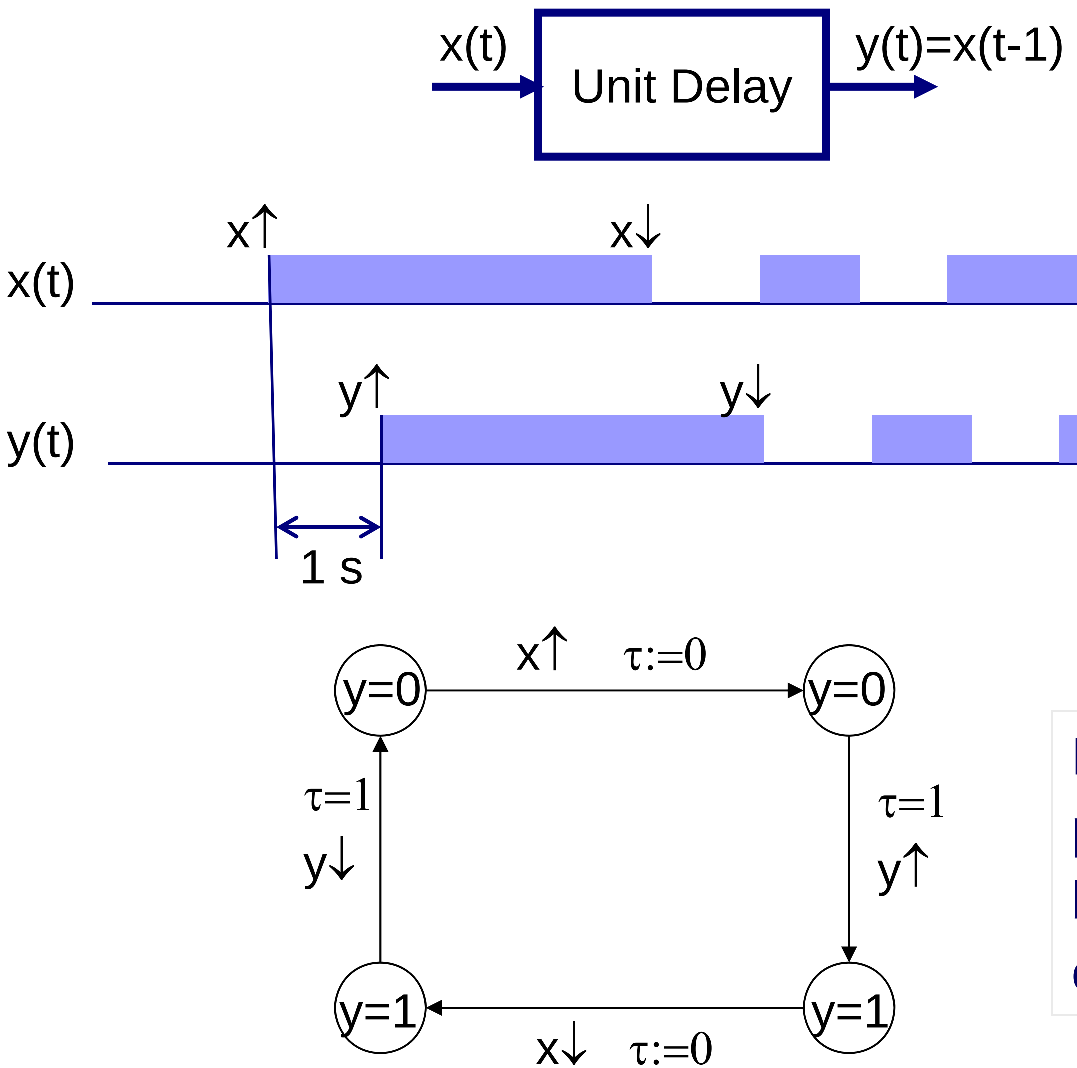
Component-based Design – Synchronous vs. Asynchronous



Matlab/Simulink

Component-based Design – Synchronous vs. Asynchronous

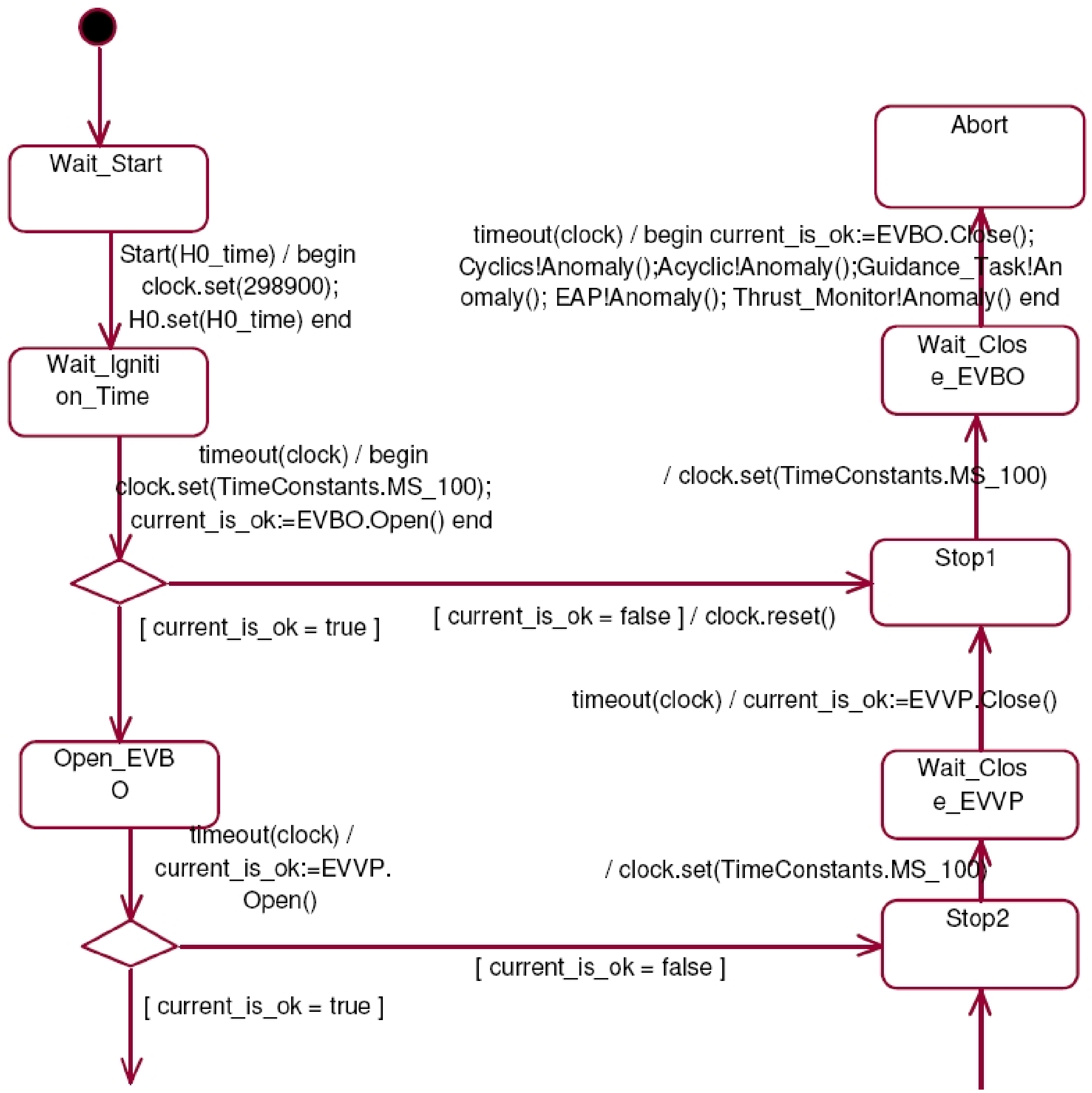
Mathematically simple does not imply computationally simple!
There is no finite state computational model equivalent to a unit delay!



Equivalent timed automaton, provided that the distance between two consecutive input changes is more than 1s.

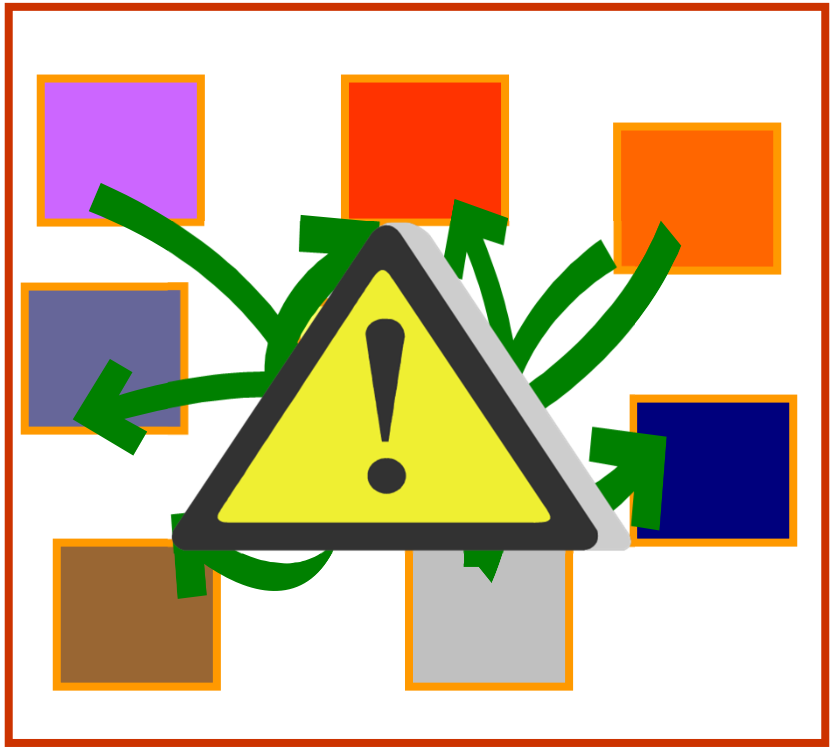
Component-based Design – Synchronous vs. Asynchronous

UML Model
(Rational Rose)



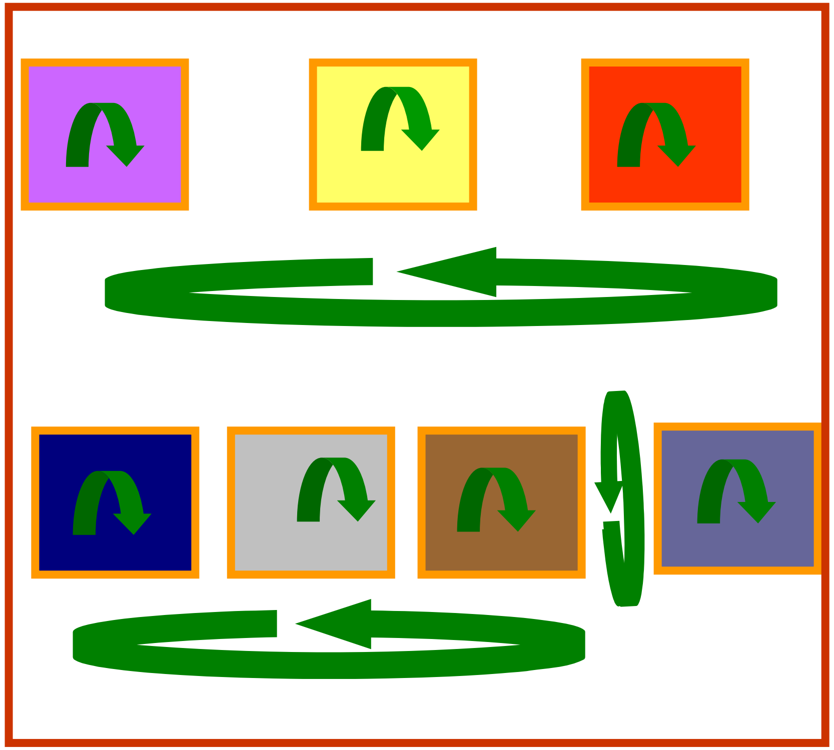
Component-based Design – Programming Styles

Thread-based programming



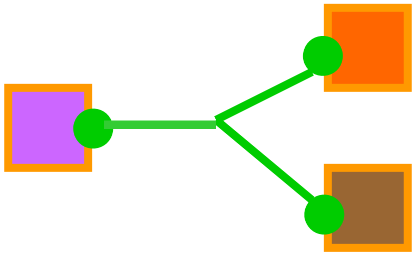
Software Engineering

Actor-based programming

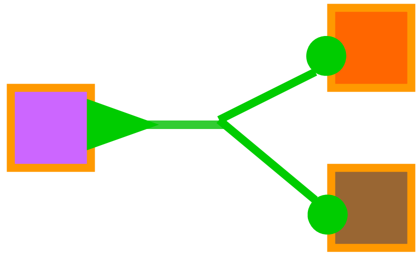


Systems Engineering

Component-based Design – Interaction Mechanisms



Rendezvous: atomic symmetric synchronization



Broadcast: asymmetric synchronization triggered by a Sender

Existing formalisms and theories are not expressive enough

- use variety of low-level coordination mechanisms including semaphores, monitors, message passing, function call
- encompass point-to-point interaction rather than multiparty interaction

Component-based Design – Composition

- Is it possible to express component coordination in terms of composition operators?

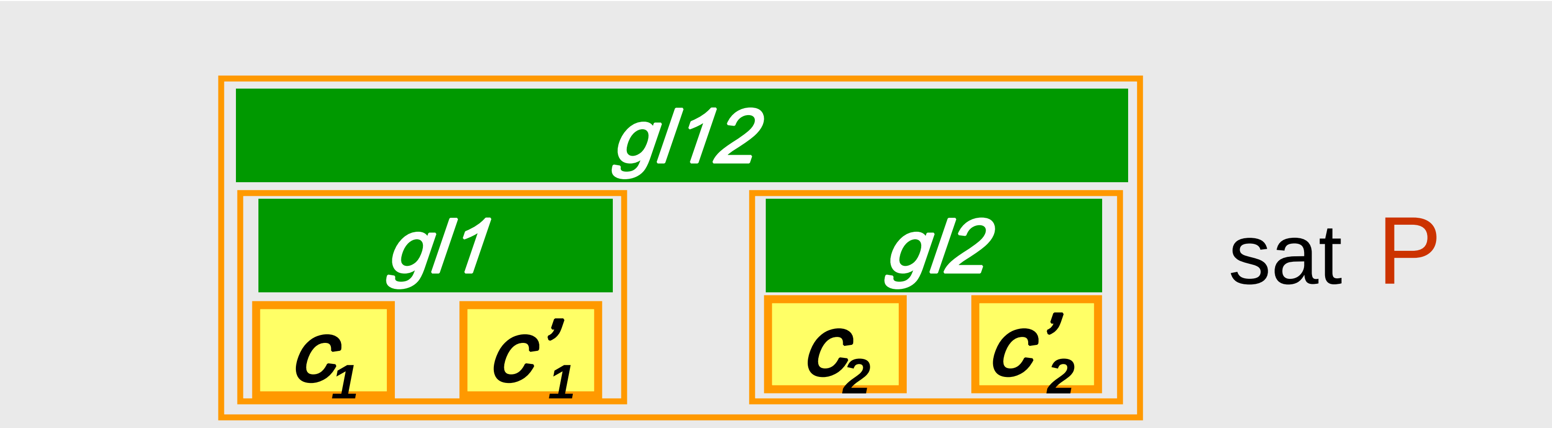
  We need a unified composition paradigm for describing and analyzing the coordination between components in terms of tangible, well-founded and organized concepts and characterized by
  - Orthogonality: clear separation between behavior and coordination constraints
  - Minimality: uses a minimal set of primitives
  - Expressiveness: achievement of a given coordination with a minimum of mechanism and a maximum of clarity

- Most component composition frameworks fail to meet these requirements
  - Some are formal such as process algebras e.g. CCS, CSP, pi-calculus
  - Other are ad hoc such as most frameworks used in software engineering e.g. ADL, or systems engineering e.g. SystemC

Component-based Design – The Concept of Glue

Build a component *C* satisfying a given property *P,* from

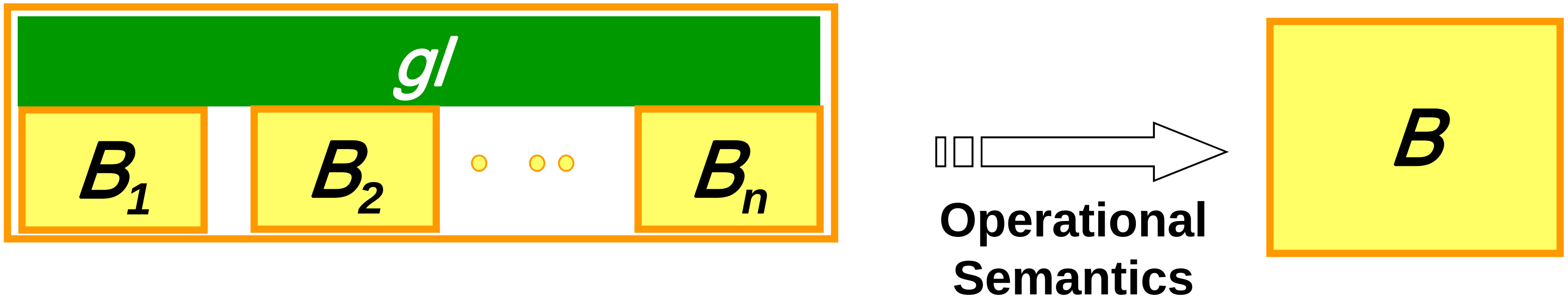
- $\mathcal{C}_0$ a set of **atomic** components described by their behavior
- $\mathcal{GL} = \{gl_1, \ldots, gl_i, \ldots\}$ a set of **glue *operators*** on components



Glue operators are stateless – separation of concerns between behavior and coordination

Component-based Design – Glue Operators

We use operational semantics to define the meaning of a composite component – glue operators are "behavior transformers"



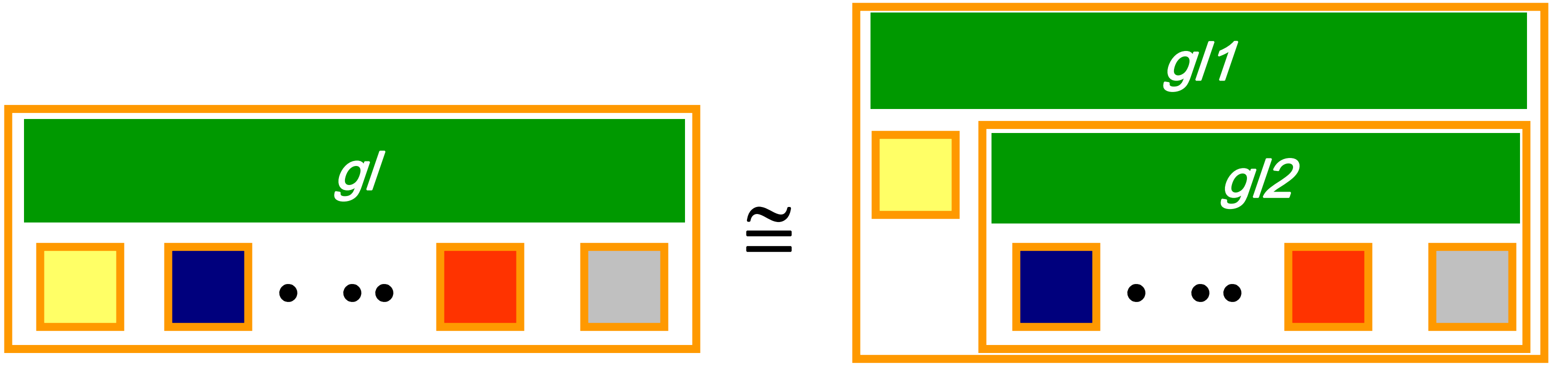
Glue Operators

- build interactions of composite components from the actions of the atomic components e.g. parallel composition operators
- can be specified by using a family of operational semantics rules (the Universal Glue)

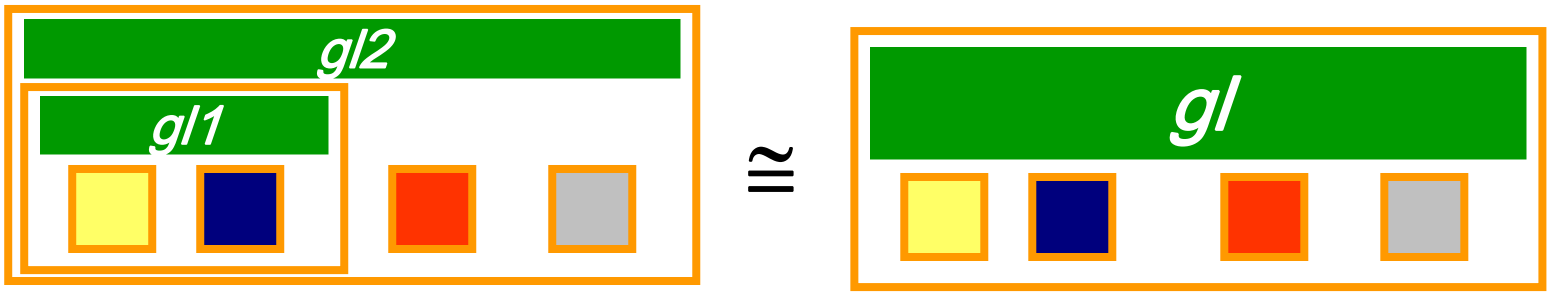
Component-based Design – Glue Operators: Properties

Glue is a first class entity independent from behavior that can be decomposed and composed

1. Incrementality



2. Flattening



Component-based Design – Glue Operators: Expressiveness

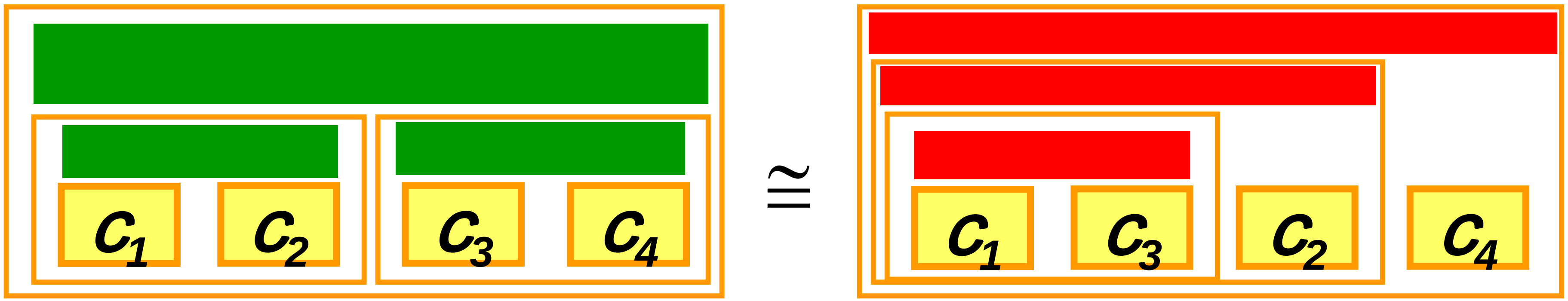
- Different from the usual notion of expressiveness!
- Based on strict separation between glue and behavior

Given two glues G_1 , G_2

G_2 ***is strongly more expressive than*** G_1

if for any component built by using G_1 and a set of components $\mathcal{C}_0$

there exists an equivalent component built by using G_2 and $\mathcal{C}_0$



Component-based Design – Glue Operators: Expressiveness

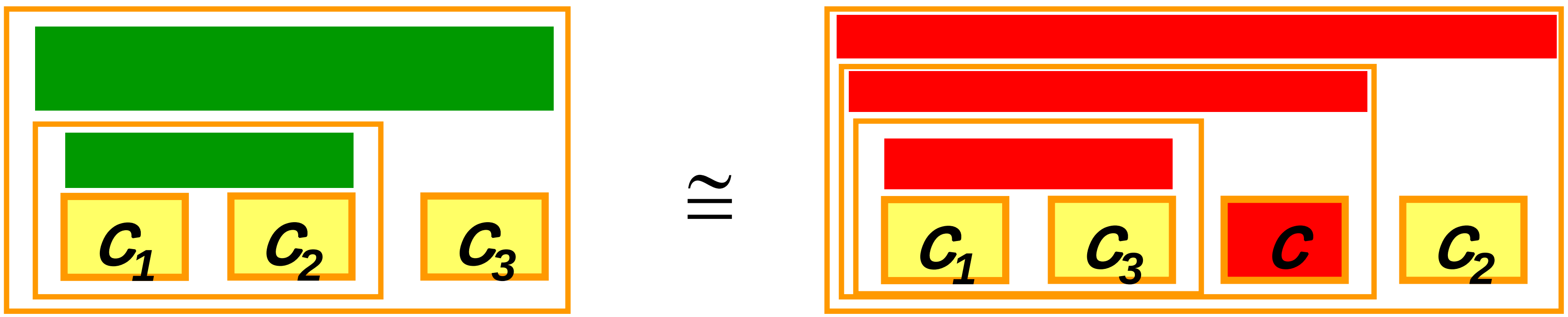
Given two glues G_1 , G_2

G_2 ***is weakly more expressive than*** G_1

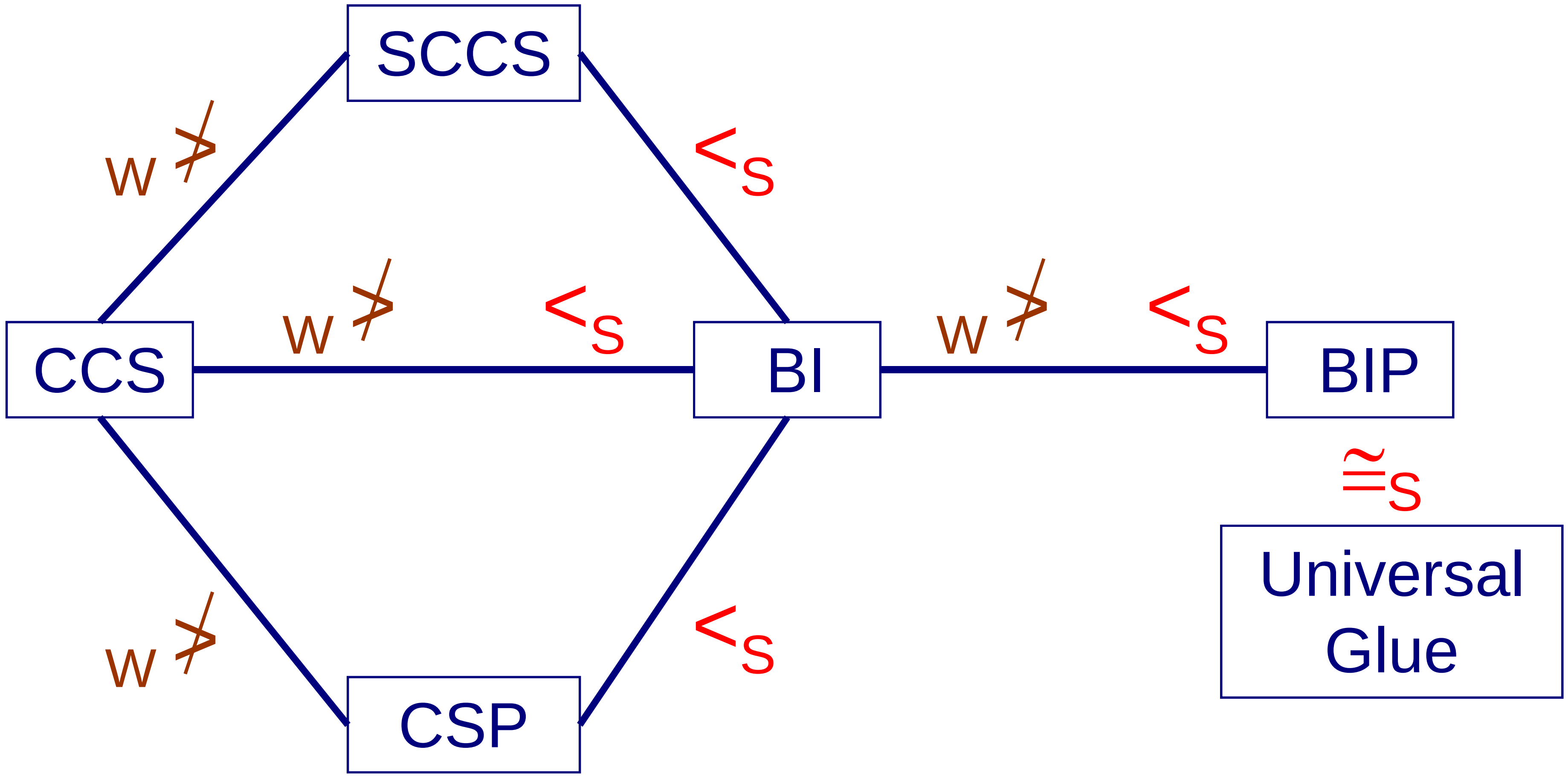
if for any component built by using G_1 and a set of components $\mathcal{C}_0$

there exists an equivalent component built by using G_2 and $\mathcal{C}_0 \cup \mathcal{C}$

where $\mathcal{C}$ is a finite set of coordinating components.



Component-based Design – Glue Operators: Expressiveness

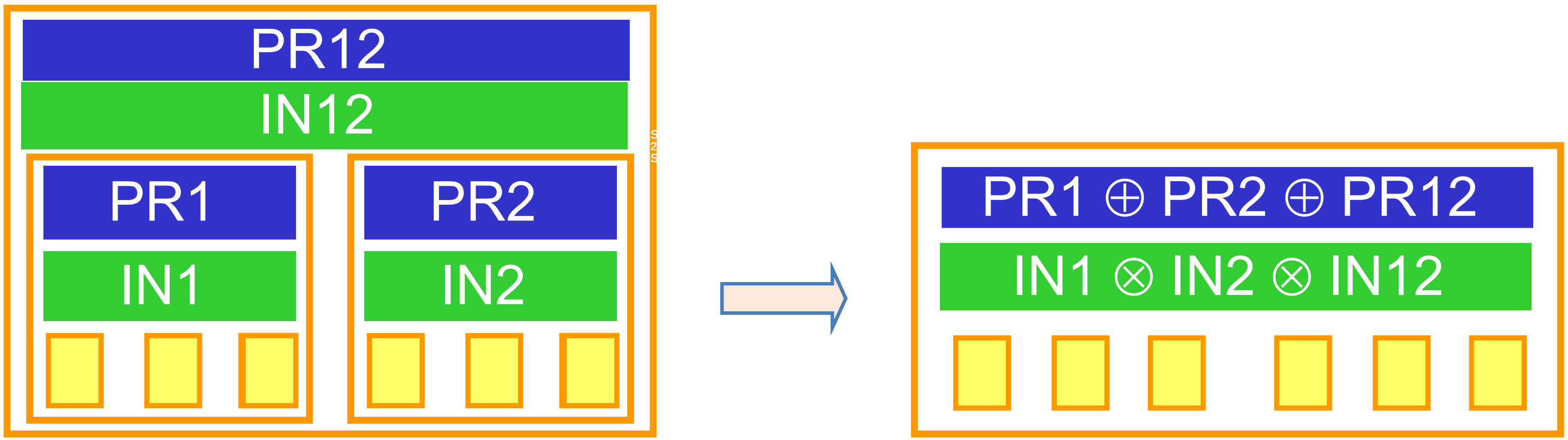


[Bliudze&Sifakis, Concur 08]

Component-based Design – Modeling in BIP

Layered component model

Composition operation parameterized by glue IN12, PR12



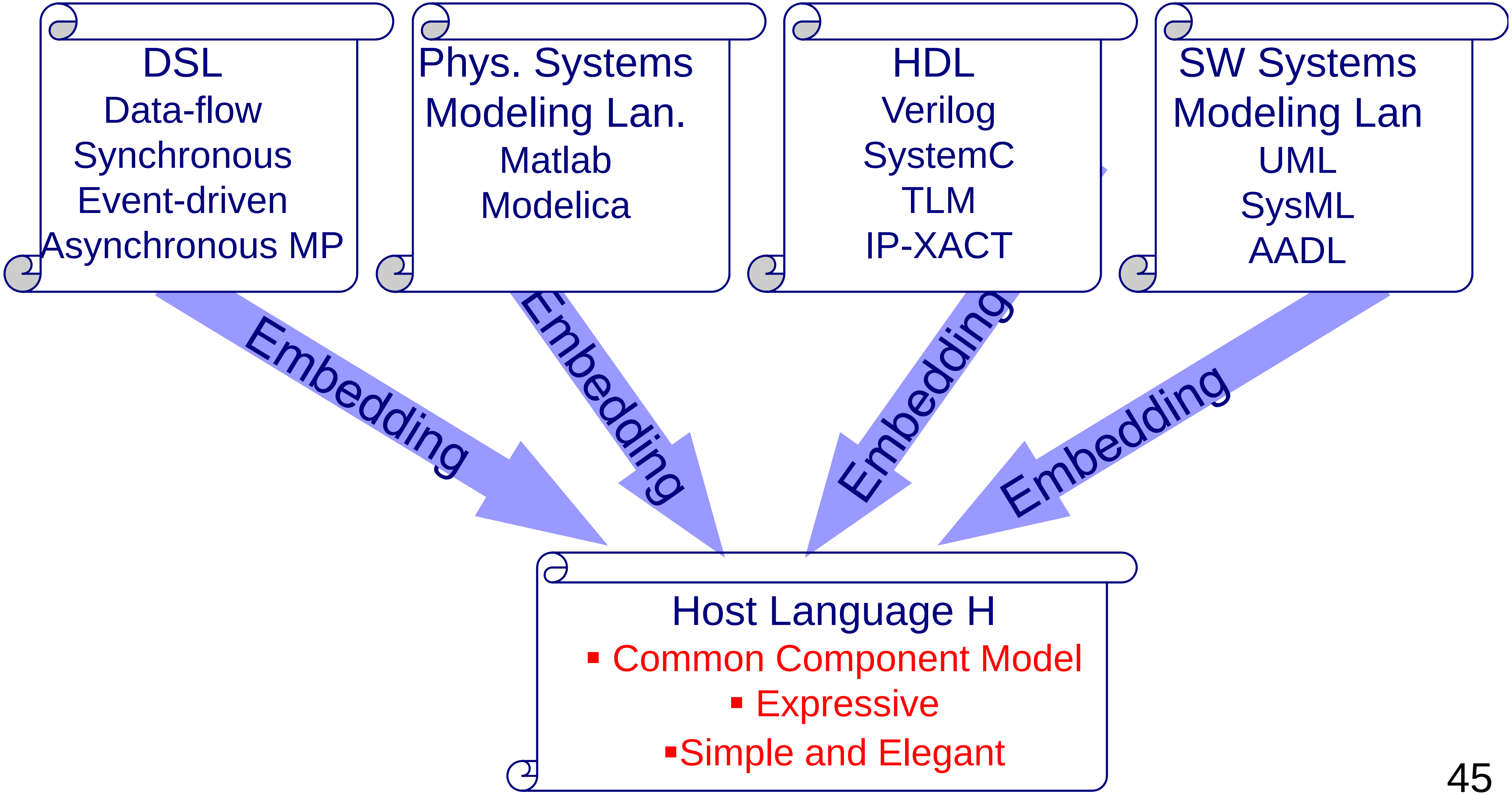
- System Design

- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design

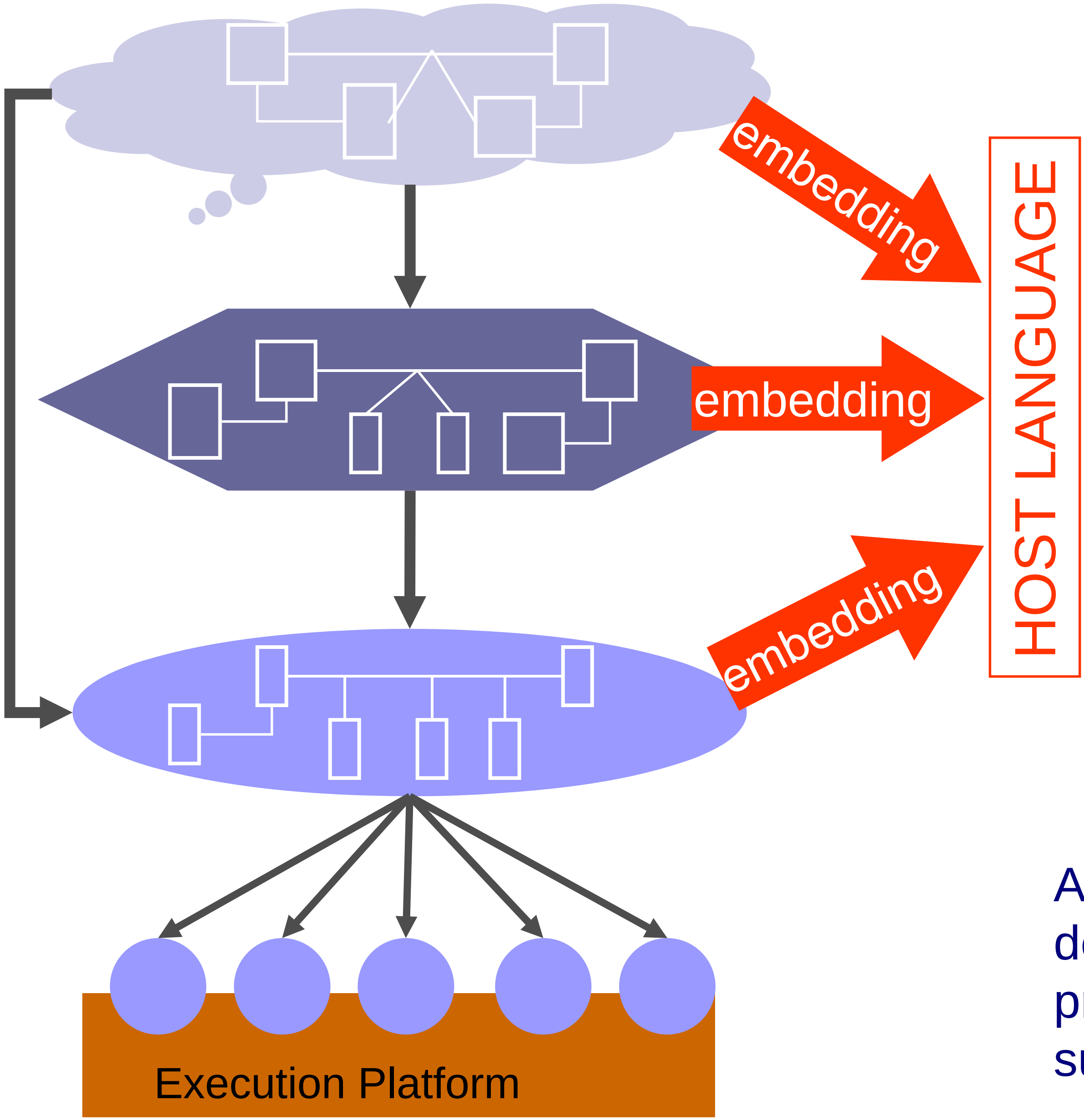
- Discussion

Semantic Coherency

- Using semantically unrelated formalisms e.g. for programming, HW description and simulation, breaks continuity of the design flow and jeopardizes its coherency
- System development is often decoupled from validation and evaluation.

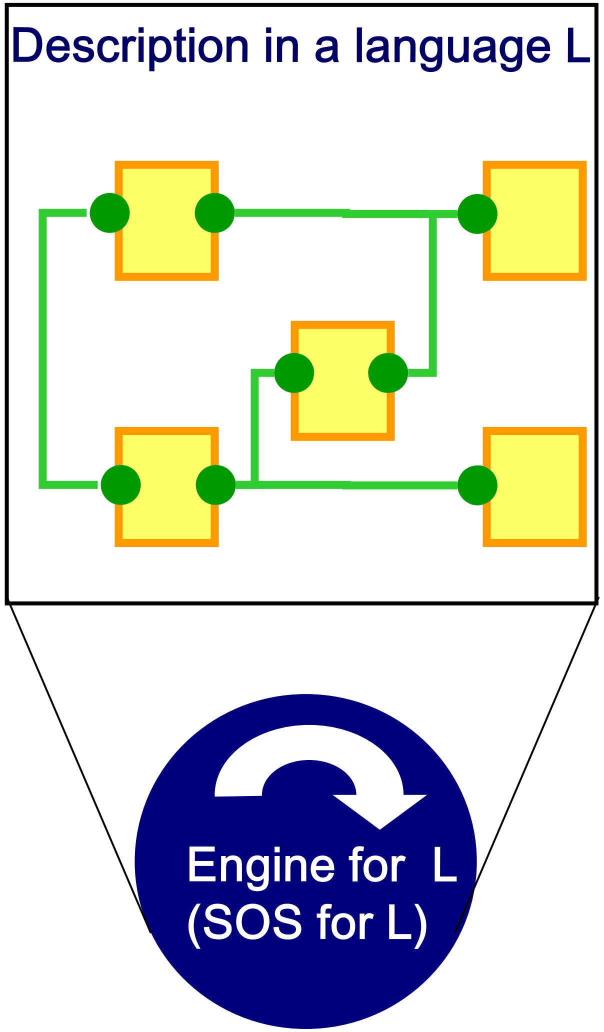


Semantic Coherency – Embedding



Any system design flow is de facto based on a host programming language such as C or Java

Semantic Coherency – Embedding

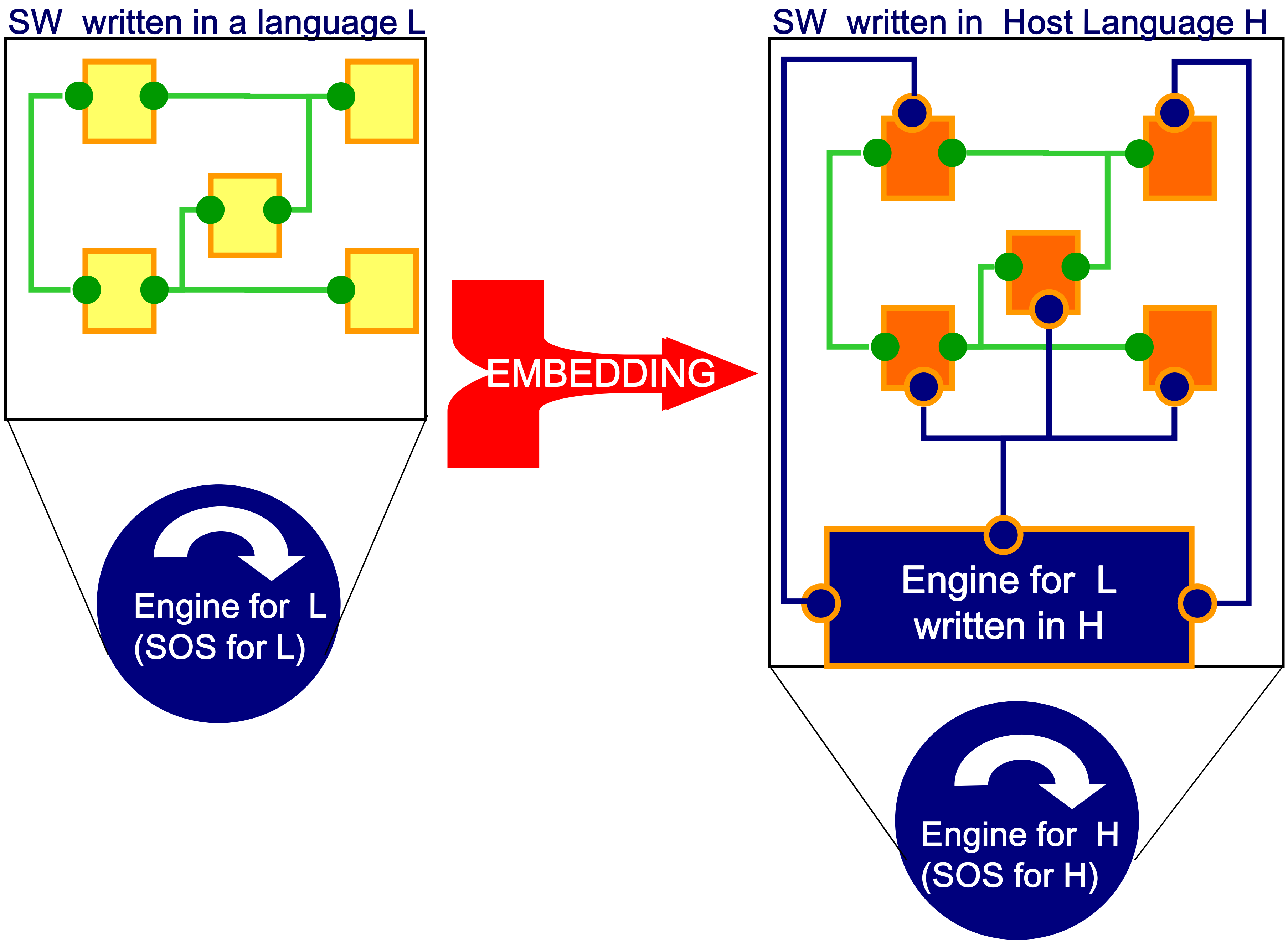


Structured Operational Semantics for L is implemented by an Engine which cyclically executes a two-phase protocol:

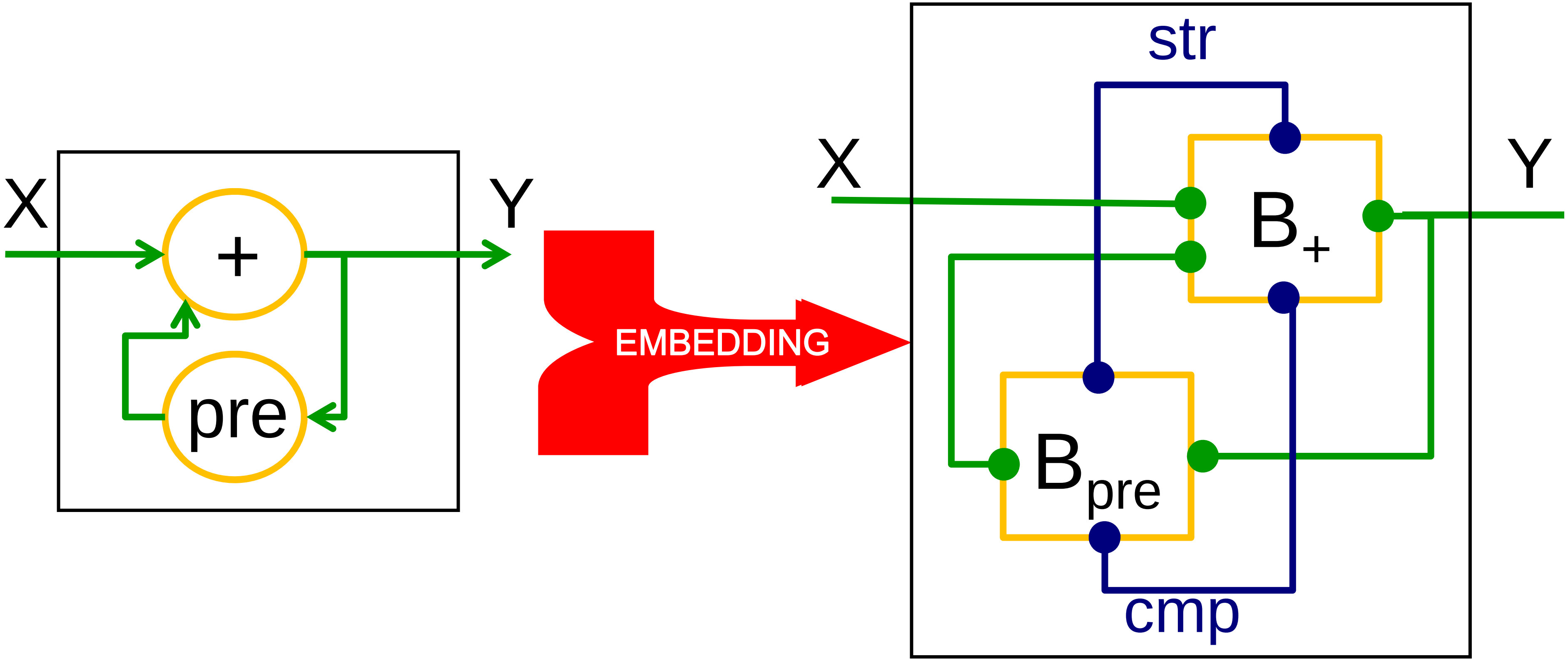
1. Monitors components and determines enabled interactions

2. Chooses and executes one enabled interaction

Semantic Coherency – Embedding



Semantic Coherency – Embedding



Y=X+pre(Y)

Program in Lustre

Program in BIP

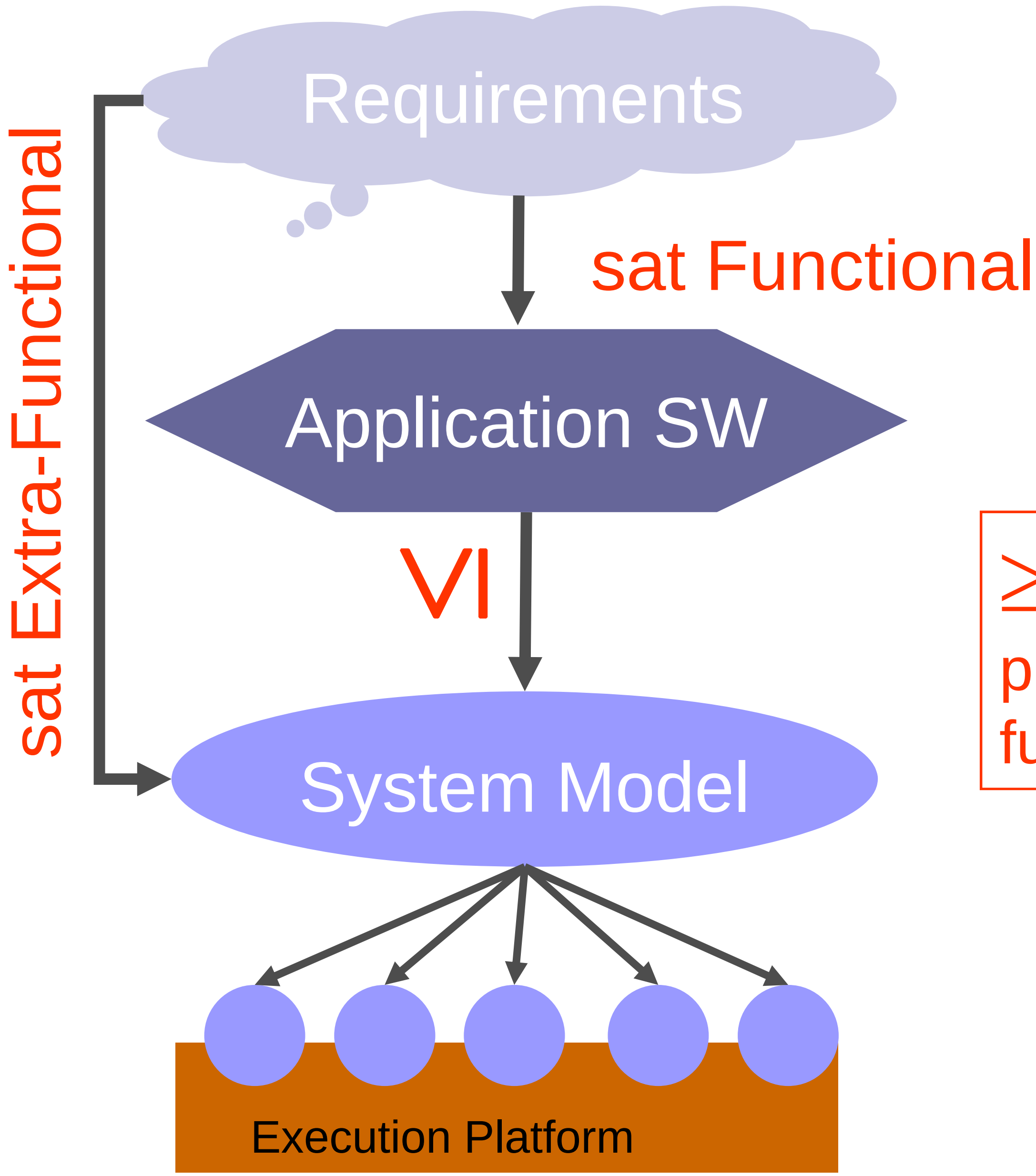
OVERVIEW

- System Design

- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design

- Discussion

Correct by Construction



≥: refinement relation preserving functional properties

Correct by Construction – Architectures

Architectures

- depict design principles, paradigms that can be understood by all, allow thinking on a higher plane and avoiding low-level mistakes
- are a means for ensuring global properties characterizing the coordination between components – correctness for free
- Using architectures is key to ensuring trustworthiness and optimization in networks, OS, middleware, HW devices etc.

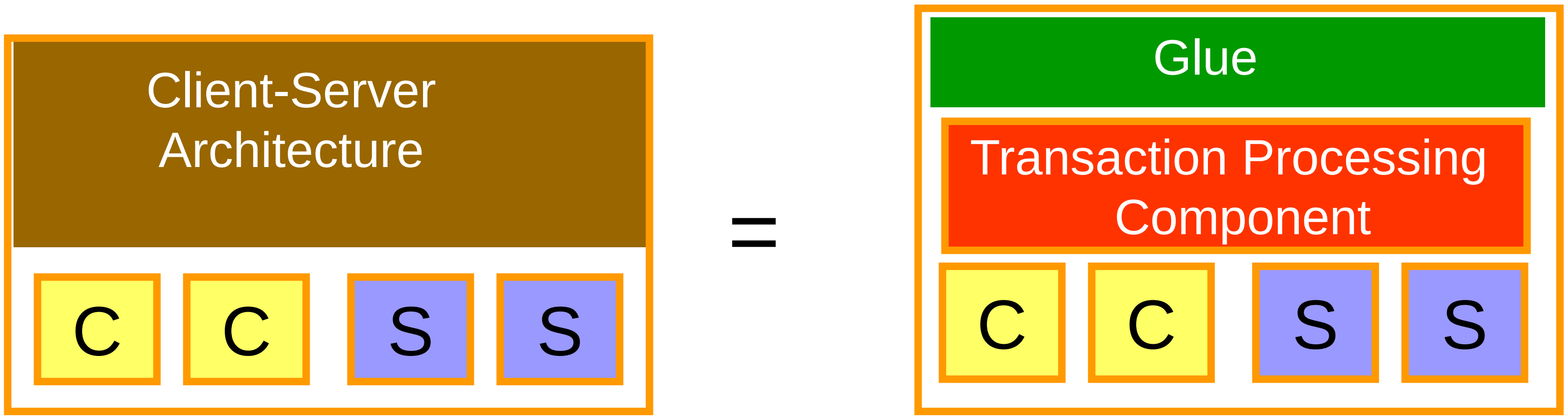
System developers extensively use libraries of reference architectures ensuring both functional and non functional properties e.g.

- Fault-tolerant architectures
- Resource management and QoS control
- Time-triggered architectures
- Security architectures
- Adaptive Architectures
- SOAP-based architecture, RESTful architecture

Correct by Construction – Architecture Definition

An architecture is a family of operators A(n)[X] parameterized by their arity n and a family of characteristic properties P(n)

- A(n)[B1,..,Bn] = gl(n)(B1,..,Bn, C(n)), where C(n) is a set of coordinators
- A(n)[B1,..,Bn] meets the characteristic property P(n).

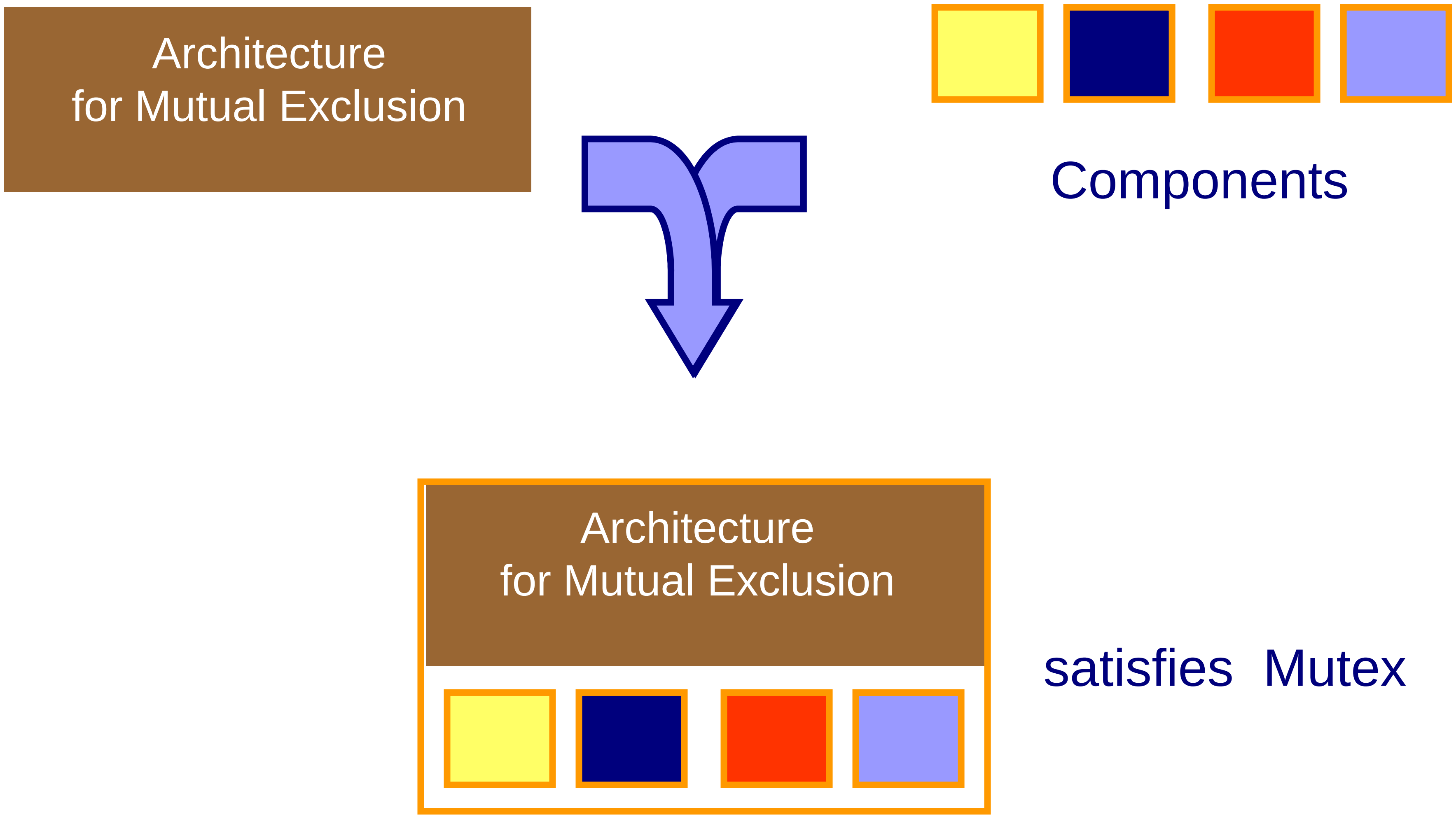


Characteristic property: atomicity of transactions, fault-tolerance ….

Note that the characteristic property need not be formalized!

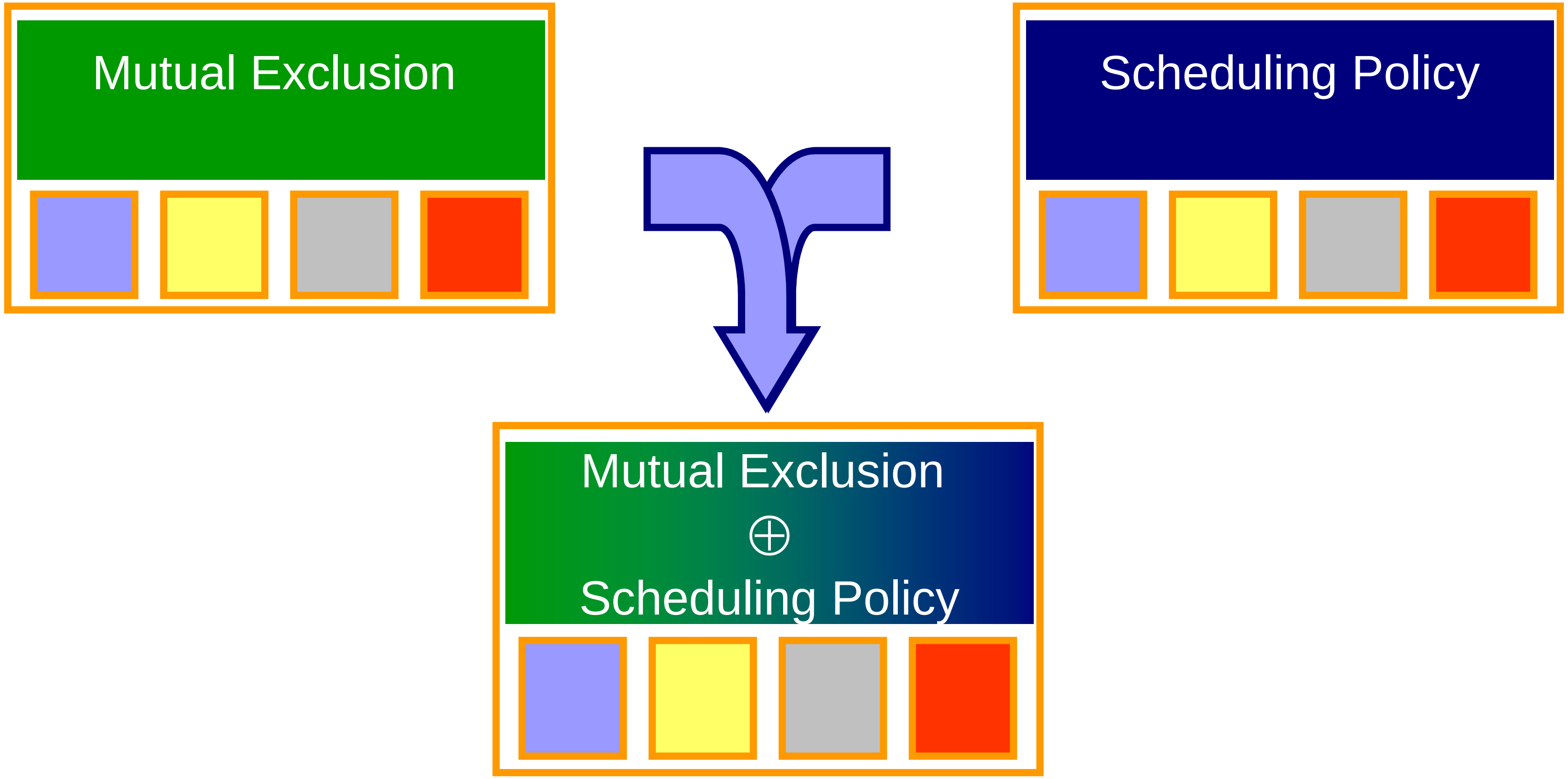
Correct by Construction – Architectures

Rule1: Property Enforcement



Correct by Construction – Architectures: Composability

Rule2: Property Composability

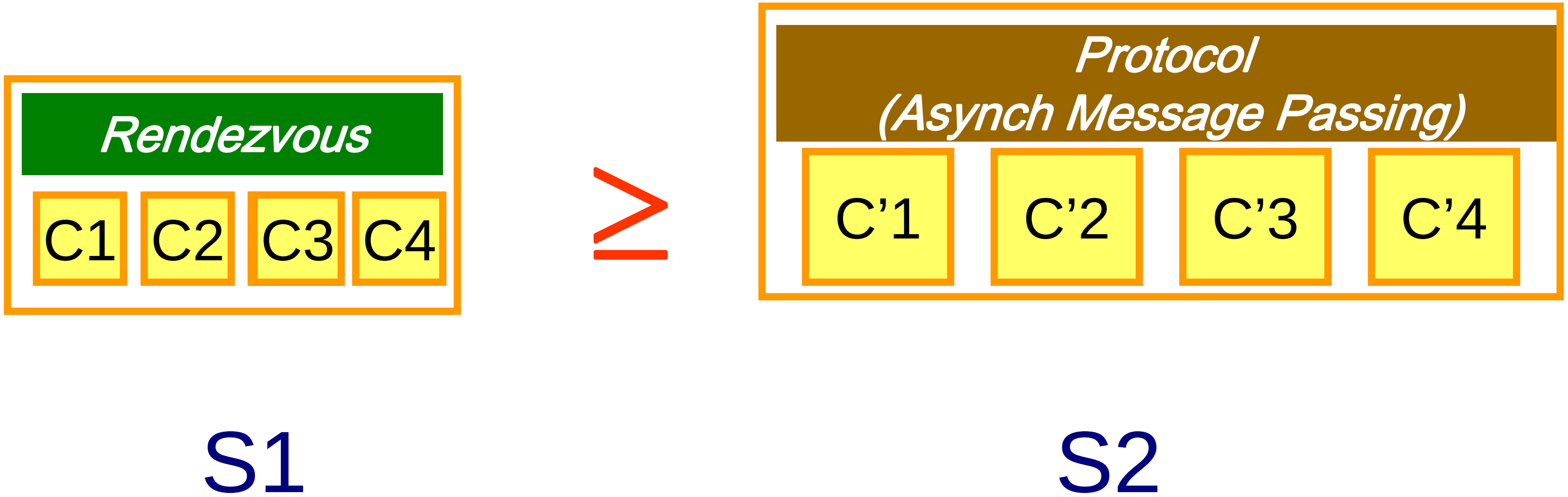


Feature interaction in telecommunication systems, interference among web services and interference in aspect programming are all manifestations of a lack of composability

Sifakis et al "A General Framework for Architecture Composability" SEFM 2014

Correct by Construction – Refinement

The Refinement Relation ≥

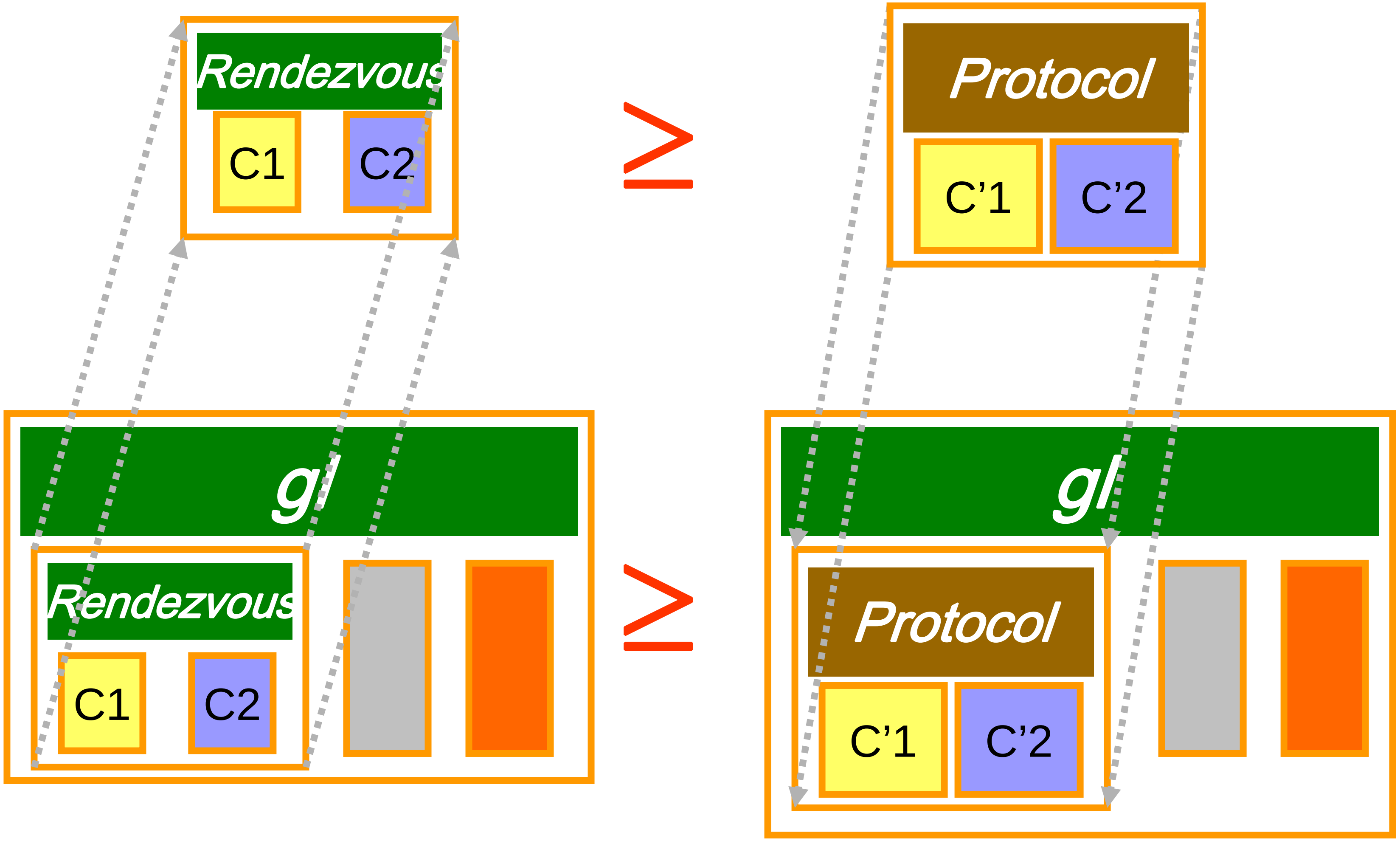


S1 ≥ S2 (S2 refines S1) if

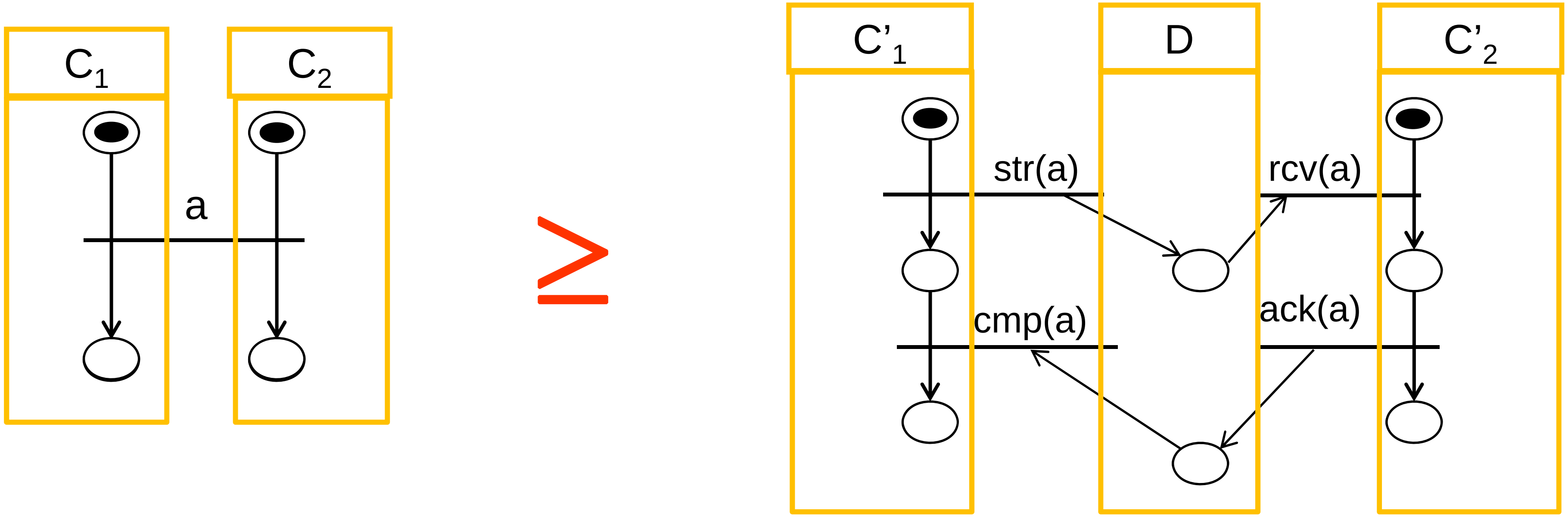
- all traces of S2 are traces of S1 (modulo some observation criterion)
- if S1 is deadlock-free then S2 is deadlock-free too
- ≥ is preserved by substitution

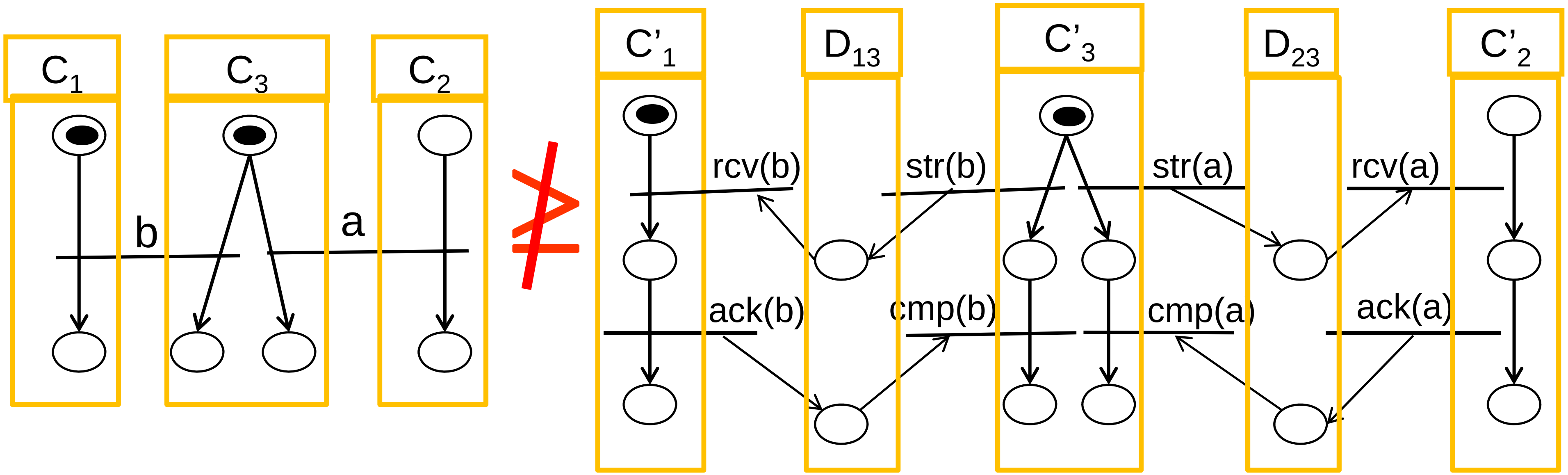
Correct by Construction – Refinement

Preservation of ≥ by substitution

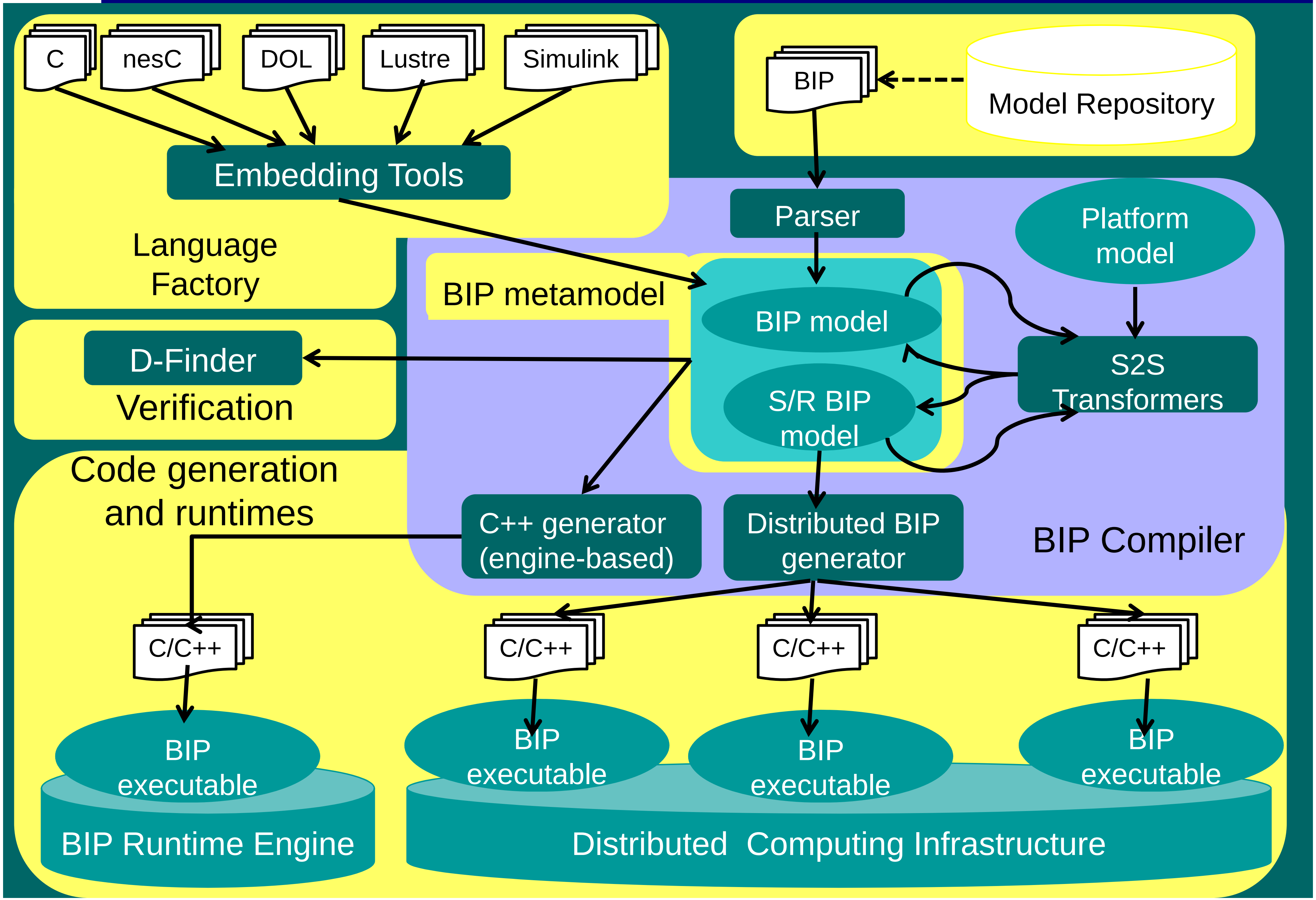


Correct by Construction – Refinement Preservation

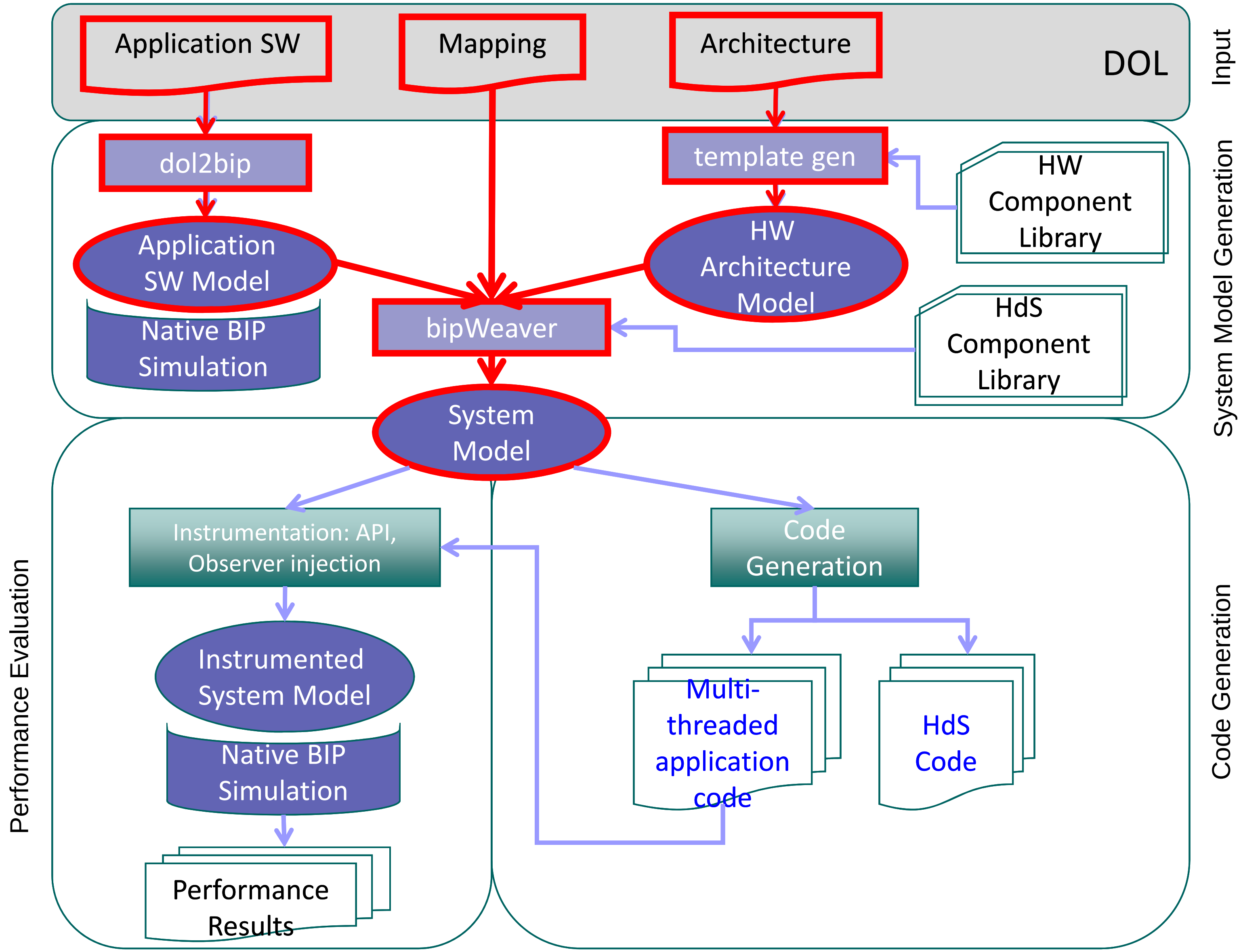




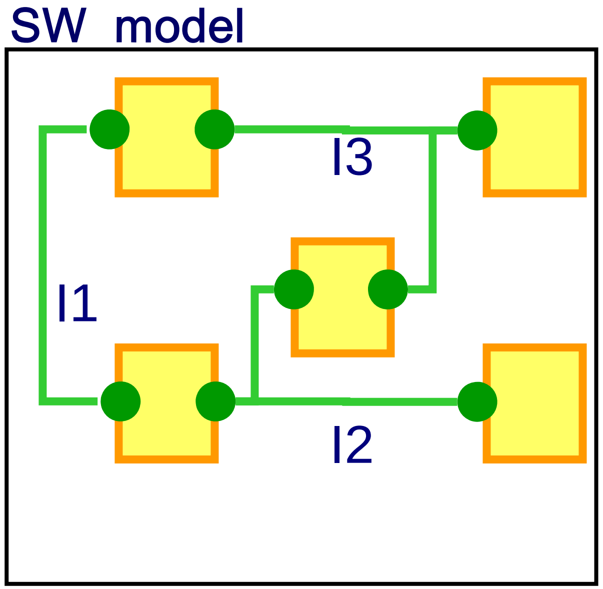
Correct by Construction – The BIP Toolset

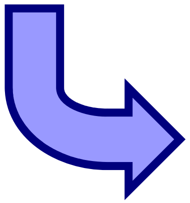


Correct by Construction – HW-driven refinement

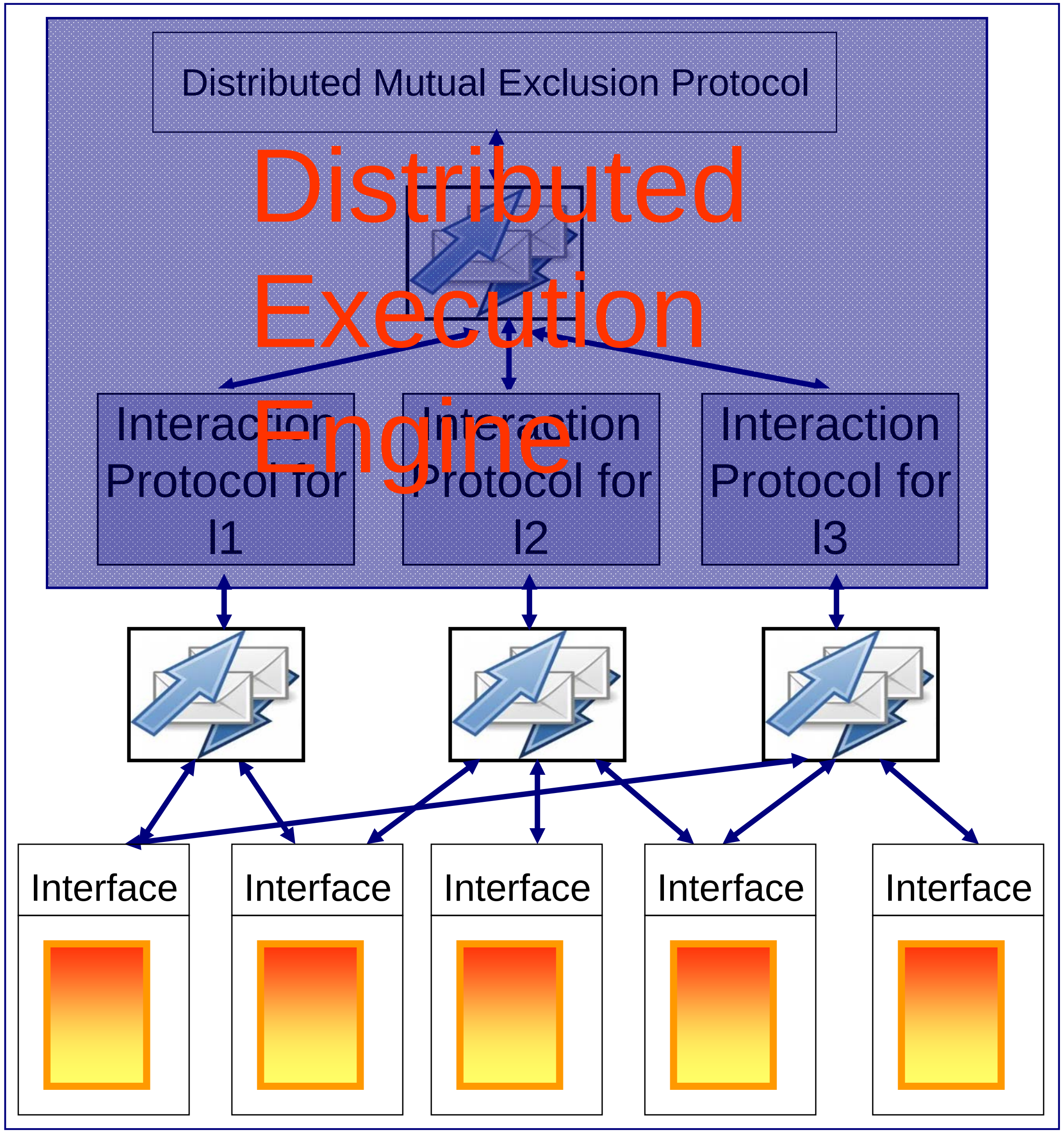


Correct by Construction – Distributed Implementation

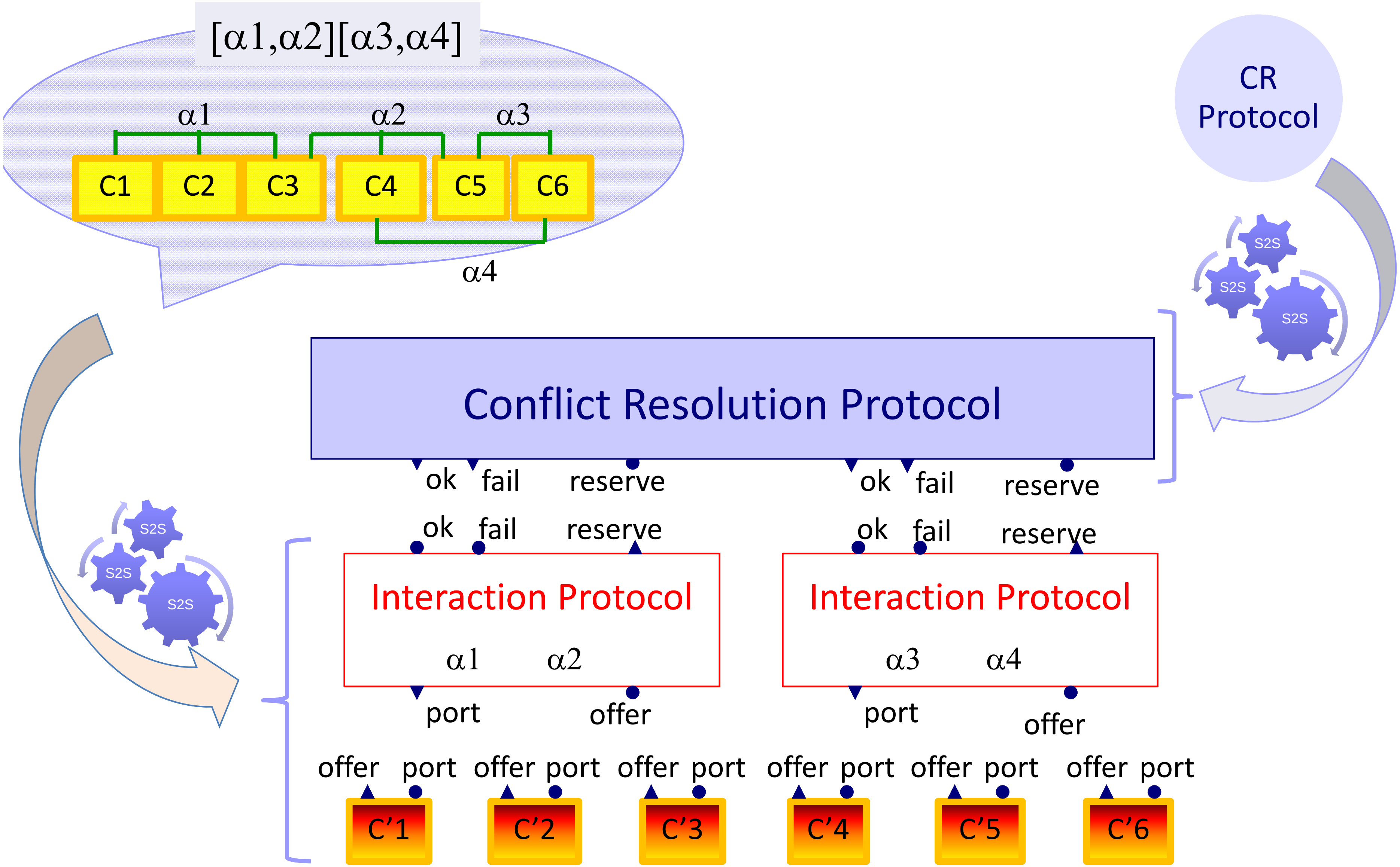




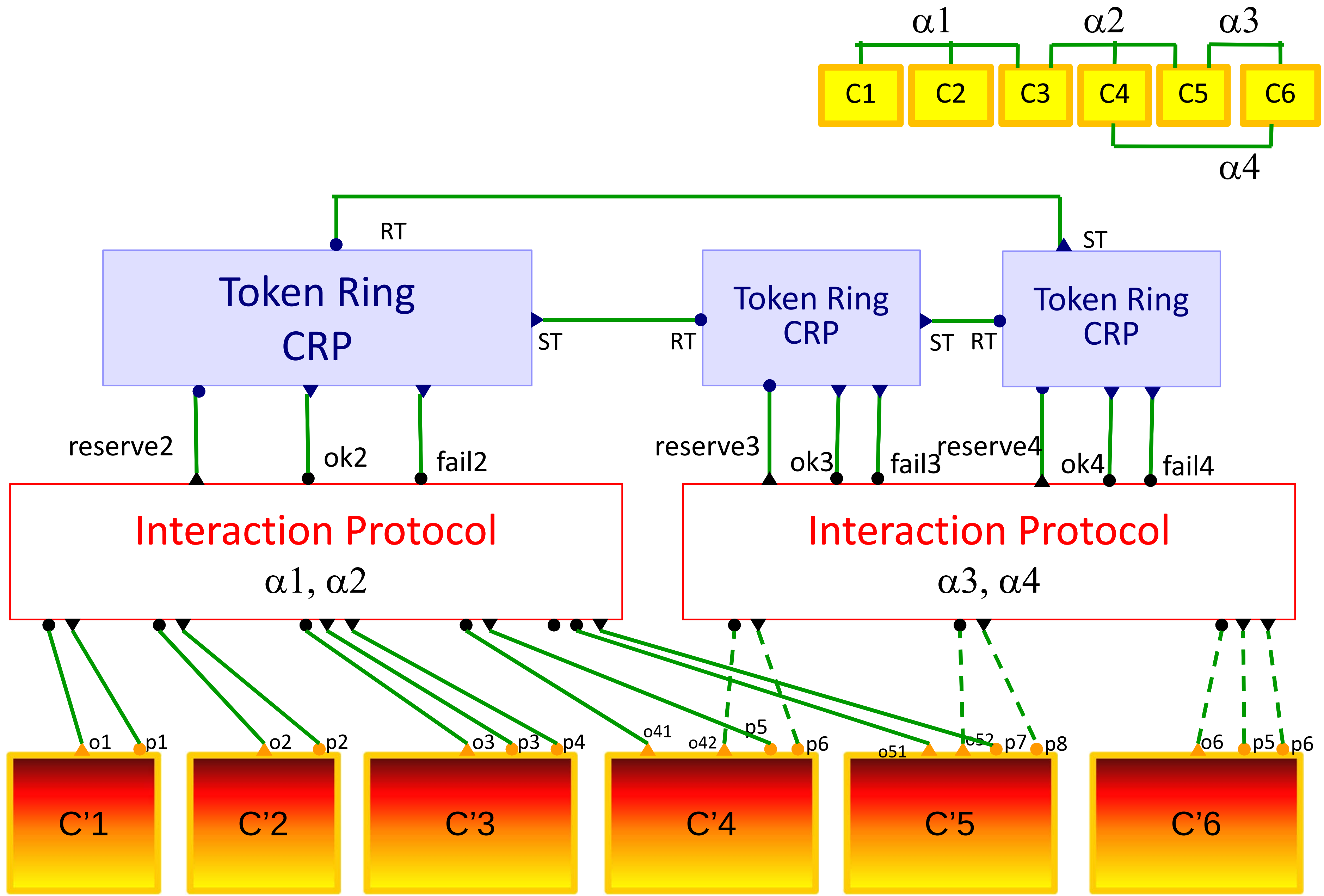
Distributed Implementation



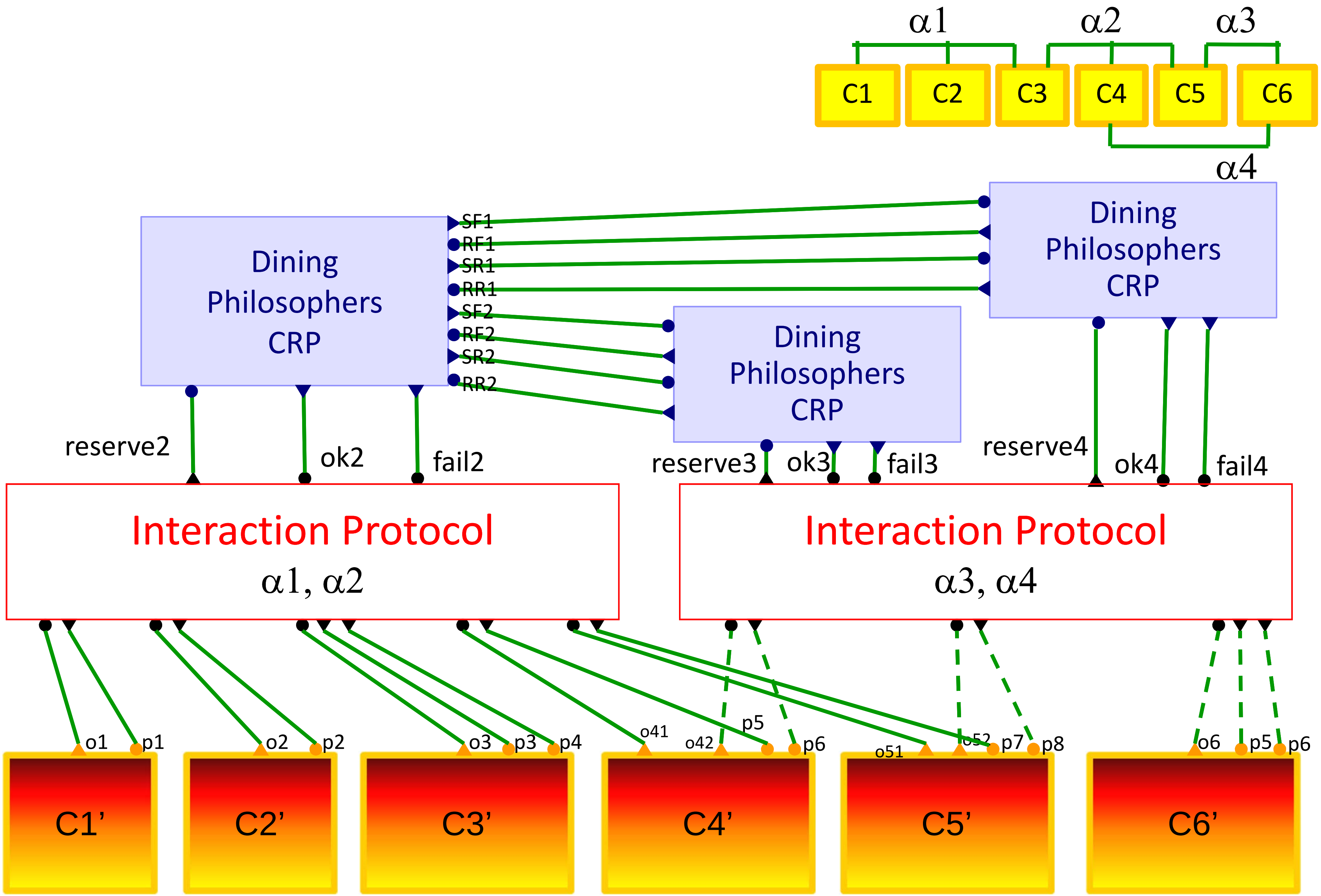
Correct by Construction – Distributed Implementation



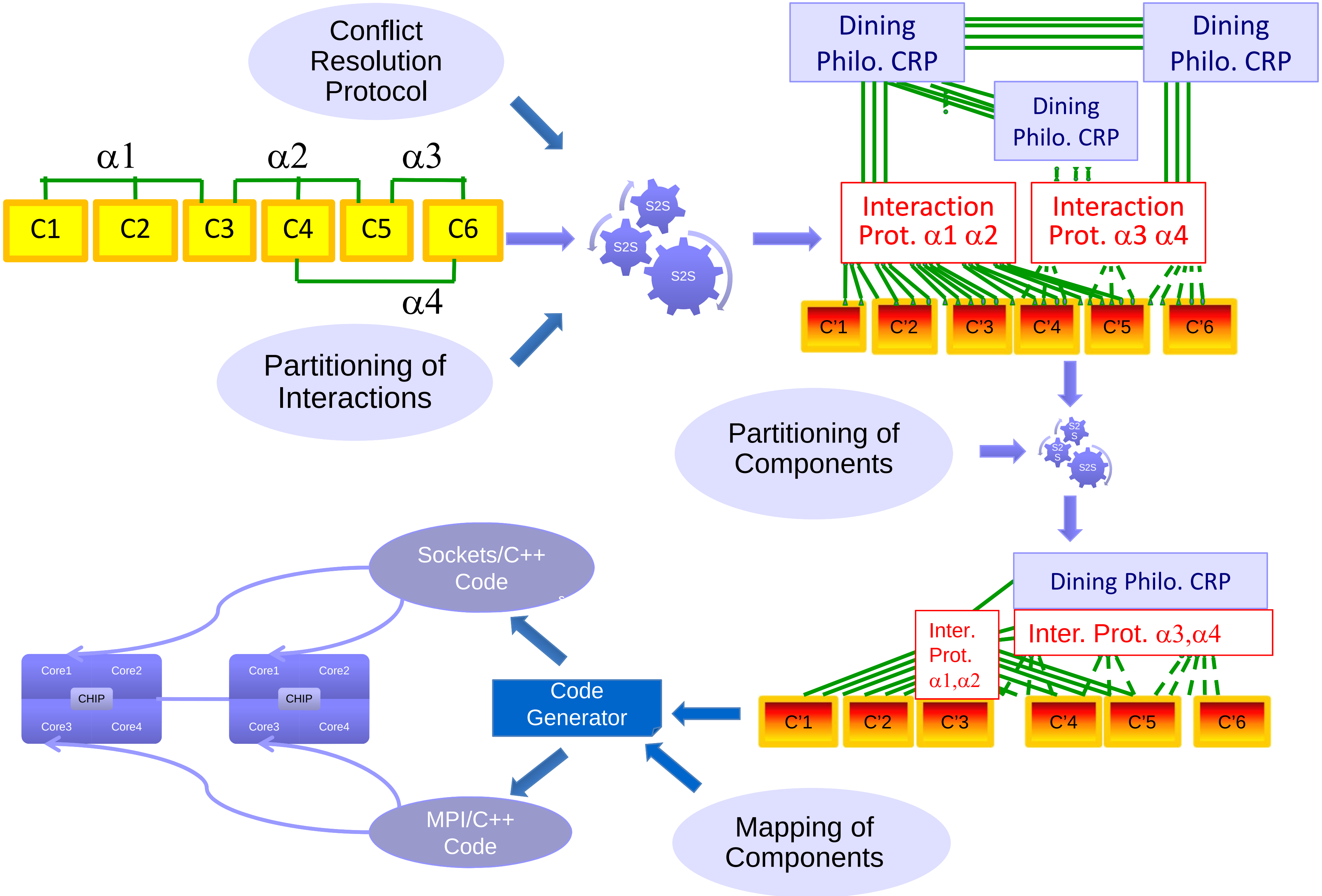
Correct by Construction – Distributed Implementation



Correct by Construction – Distributed Implementation



Correct by Construction – Distributed Implementation



- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- Discussion

OVERVIEW

Discussion – Can the Vision Come True?

<u>Things go completely the opposite way!</u>
The need for rigorous design is sometimes directly or indirectly questioned by developers of large-scale systems (e.g., web-based systems) who privilege experimental/analytic approaches:

- The cyber-world can be studied in the same manner as the physical world, e.g. Web Science, "Cyber-Physics?"
- The aim is to find laws that govern/explain observed phenomena rather than to investigate design principles for achieving a desired behavior.

"On line companies . . . . don't anguish over how to design their Web sites. Instead they conduct controlled experiments by showing different versions to different groups of users until they have iterated to an optimal solution" .

My opinion

- Experimental approaches can be useful only for optimization purposes
- Trustworthiness is a qualitative property and by its nature, it cannot be achieved by fine tuning of parameters. Small changes can have a dramatic impact on system safety and security.

Discussion – Why Is It So Hard?

The Physics Hierarchy	The Computing Hierarchy	The Bio-Hierarchy
The Universe	The Cyber-world	Ecosystem
Galaxy	Networked System	Organism
Solar System	Reactive System	Organ
Electro-mechanical System	Virtual Machine	Tissue
Crystals-Fluids-Gases	Instruction Set Architecture	Cell
Molecules	Register Transfer Level	Protein and RNA networks
Atoms	Logical Gate	Protein and RNA
Particles	Transistor	Genes

We need theory, methods and tools for climbing
up-and-down abstraction hierarchies

Discussion – The Way Forward

Design formalization raises a multitude of deep theoretical problems related to the conceptualization of needs in a given area and their effective transformation into correct artifacts. Two key issues are

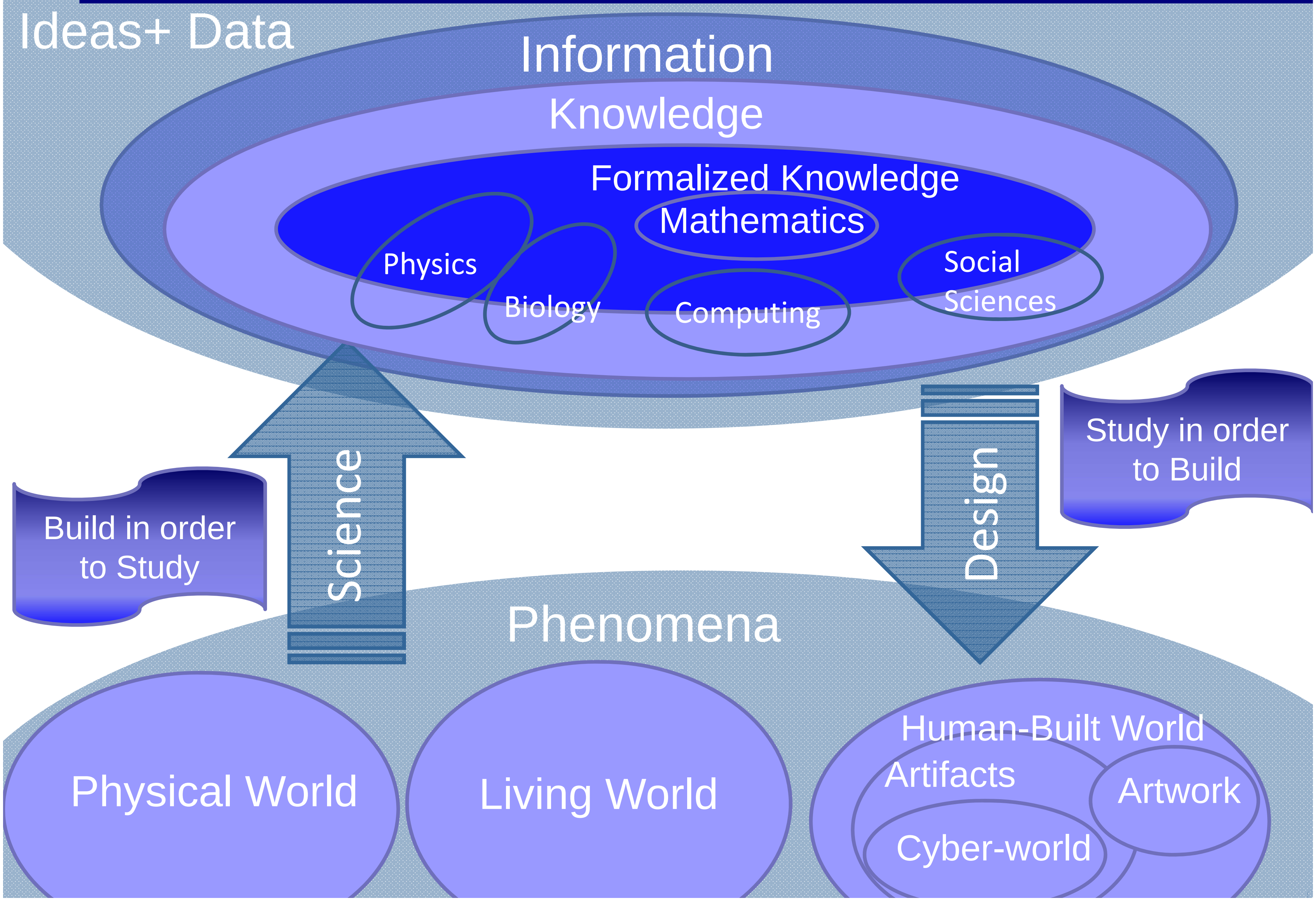
Languages: Move from thread-based programming to actor-based programming for component-based systems

- as close as possible to the declarative style so as to simplify reasoning and relegate software generation to tools encompassing
- supporting synchronous and asynchronous execution as well as the main programming paradigms
- allowing description of architectures and high-level coordination mechanisms

Constructivity: There is a huge body of not yet well-formalized solutions to problems in the form of algorithms, protocols, hardware and software architectures. The challenge is to

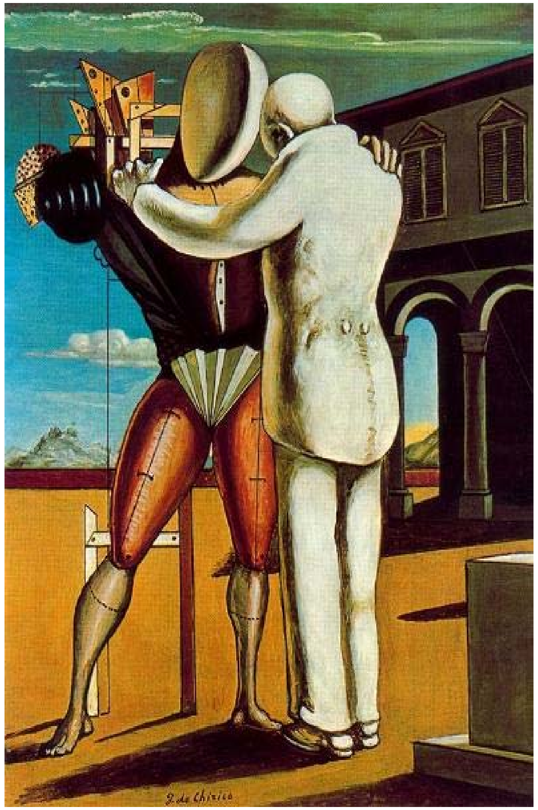
- formalize these solutions as architectures and prove their correctness
- provide a taxonomy of the architectures and their characteristic properties
- decompose any coordination property as the conjunction of predefined characteristic properties enforced by predefined architectures?

Discussion – The Rationale for Design



Discussion – For a System Design Science

Achieving this goal for systems engineering is both an intellectually challenging and culturally enlightening endeavor – it nicely complements the quest for scientific discovery in natural sciences



Failure in this endeavor would

- seriously limit our capability to master the techno-structure
- also mean that designing is a definitely a-scientific activity driven by predominant subjective factors that preclude rational treatment

Is everything for the best in the best of all possible cyber-worlds ?
- I believe the toughest uphill battles are still in front of us

Discussion

Thank You