Embedded Systems Challenges and Work Directions

Heraklion, July 21, 2008

Joseph Sifakis
VERIMAG Laboratory

O
V
E
R
V
I
E
W

- Embedded Systems
- Scientific Challenge
- Work Directions

Embedded Systems

An Embedded System integrates **software and hardware** jointly and specifically designed to provide given functionalities, which are often **critical**.

Embedded Systems: Economic Stakes

Embedded Systems are of strategic economic importance

- Factor for innovation and differentiation
- Principal source of added value: particularly for embedded software
- This is the fastest-growing sector in IT

Europe has leading positions in sectors where embedded technologies are central to growth

- Currently: Industry (avionics, automotive, space, consumer electronics, telecom devices, energy distribution, rail transport, …)
- Anticipated: Services (e-Health, e-Education)

Embedded Systems: Trends

It is hard to jointy meet **technical requirements** such as

- **Reactivity**: responding within known and guaranteed delay

  Ex : flight controller

- **Autonomy** : provide continuous service without human intervention

  Ex : no manual start, optimal power management

- **Robustness :** guaranteed minimal service in any case

  Ex : attacks, hardware failures, software execution errors

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

Technological challenge :
Building systems of guaranteed functionality and quality, at an acceptable cost

State of the art

TODAY

We master – at a high cost two types of systems which are difficult to integrate:

- Critical systems of low complexity
  - ***Flight controller***
- Complex « best effort » systems
  - ***Telecommunication systems***

TOMORROW

We need

- Affordable critical systems
  Ex : transport, health
- Successful integration of heterogeneous systems of systems
  - ***Convergence web/embedded systems***
  - ***Automated Highways***
  - ***New generation air traffic control***
  - ***« Ambient Intelligence»***

O
V
E
R
V
I
E
W

- Embedded Systems
- Scientific Challenge
- Work Directions

The ES Challenge

Technological Challenge:

Building systems of
guaranteed functionality and quality
(performance and robustness),
at acceptable costs.

This **Technological Challenge**
hides an underlying **Scientific Challenge**

Scientific Challenge:

The emergence of Embedded Systems
as a ***scientific and engineering discipline***
enabling ***system design predictability****,*
as is already the case for the physical sciences.

Scientific Challenge

Execution constraints:
CPU speed
memory
power
failure rates

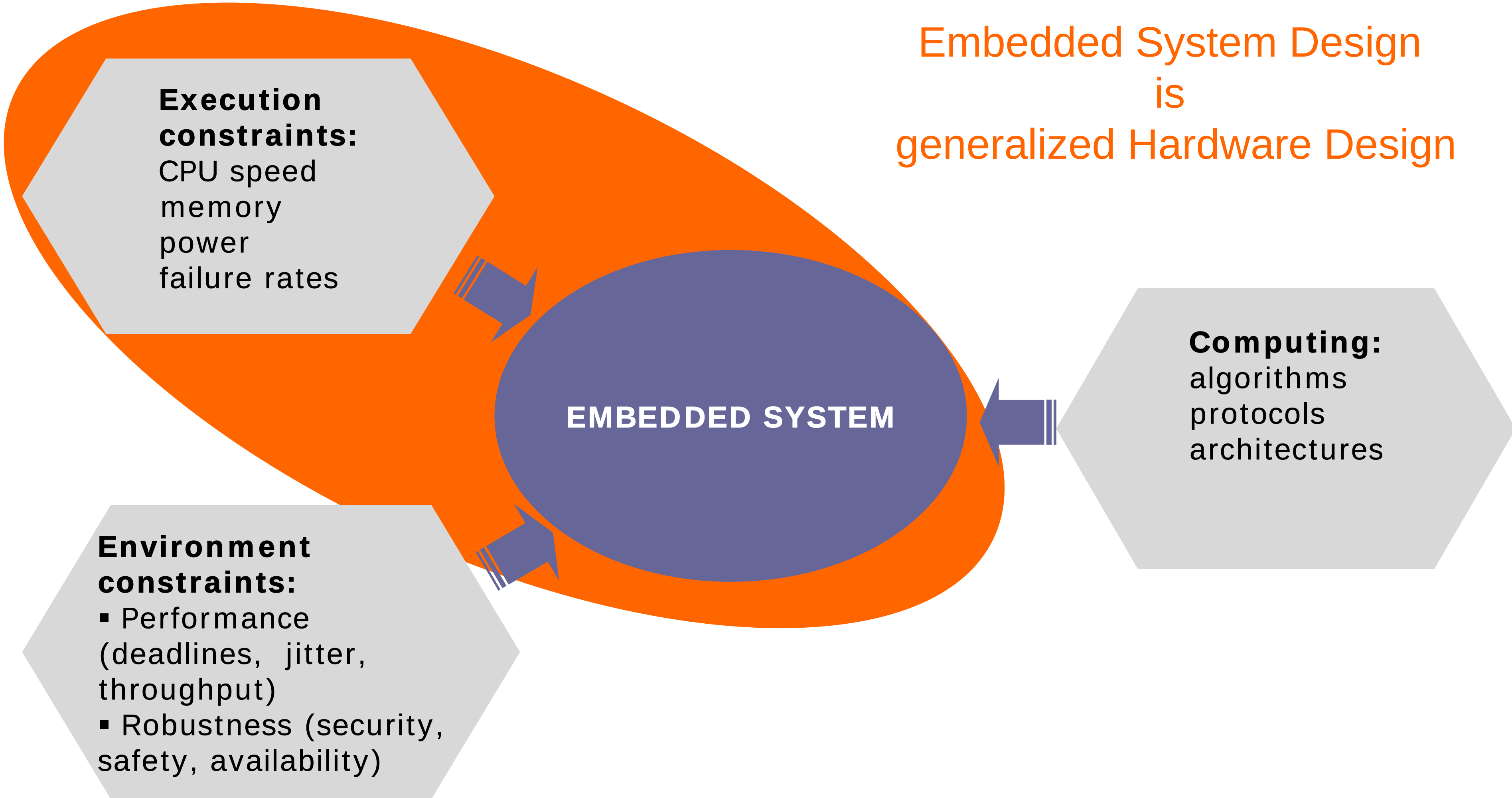
EMBEDDED SYSTEM

Computing:
algorithms
protocols
architectures

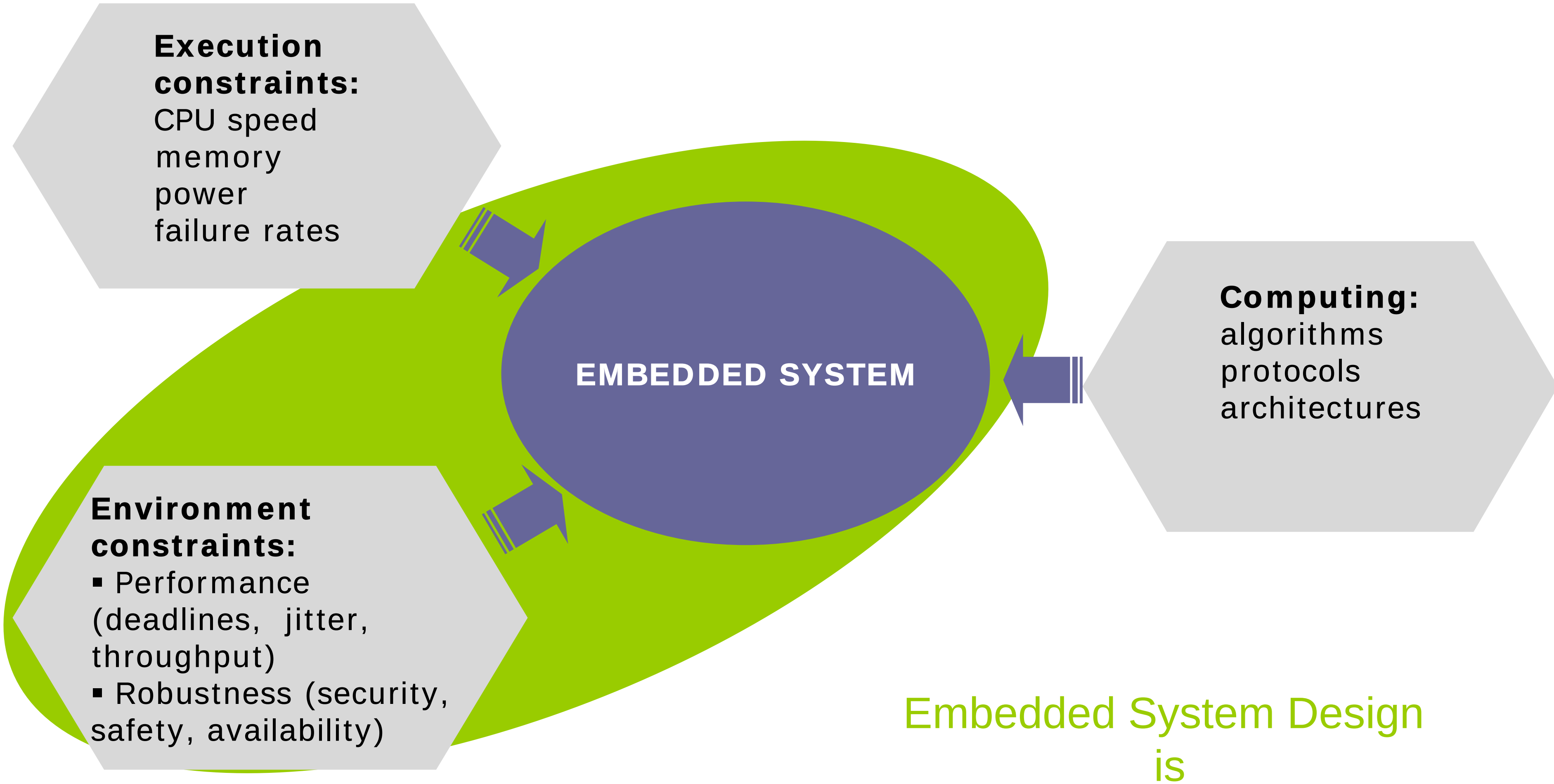
Environment constraints:
- Performance (deadlines, jitter, throughput)
- Robustness (security, safety, availability)

Scientific Challenge

Embedded System Design
is
generalized Hardware Design



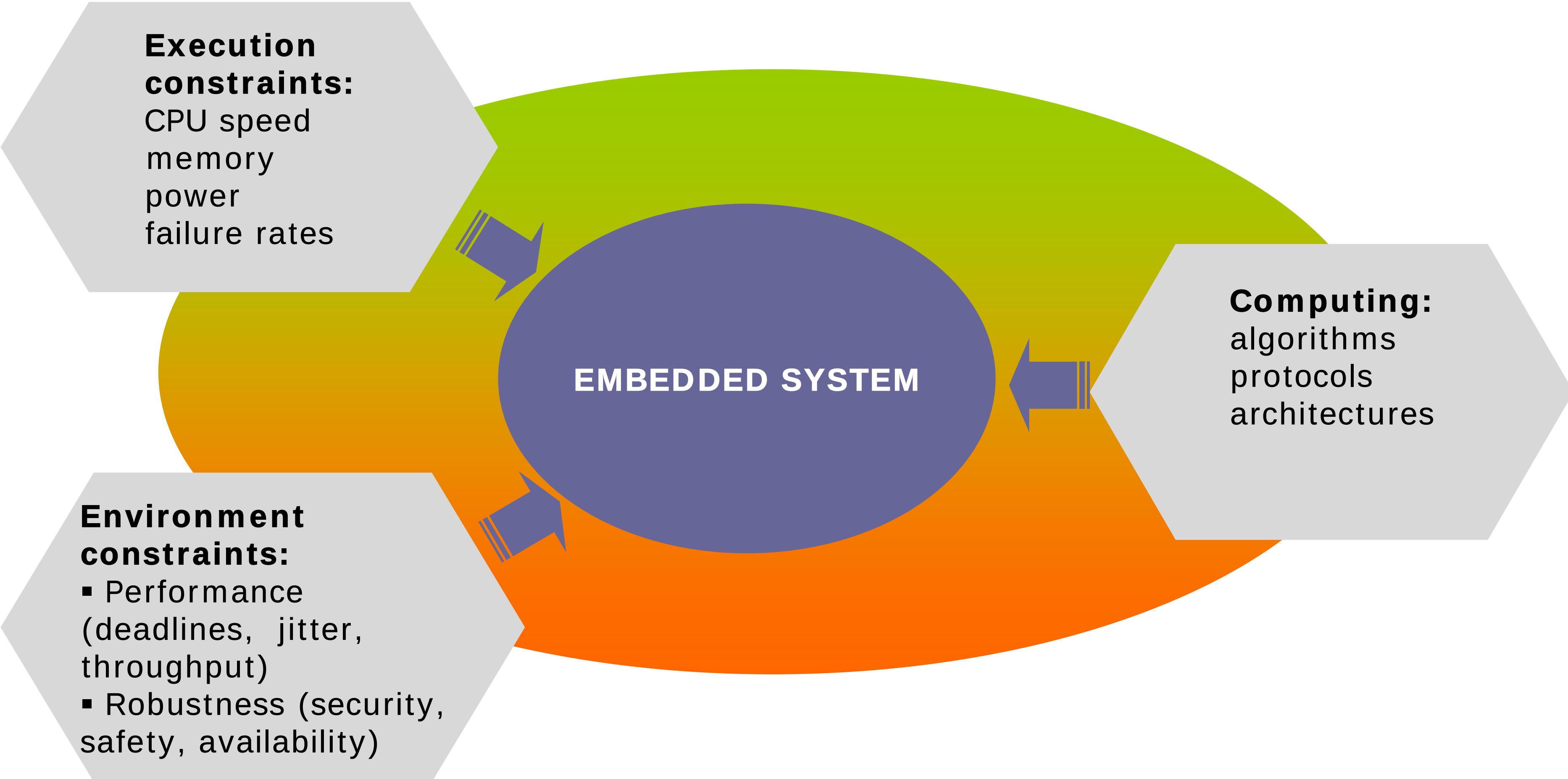
Scientific Challenge



Embedded System Design
is
generalized Control Design

Scientific Challenge

Embedded System Design coherently integrates all these



We need to revisit and revise the most basic computing paradigms to include methods from Electrical Engineering and Control

Scientific Challenge:Two Distant Disciplines

Physics

Theory for building artifacts with predictable behavior

Computer Science

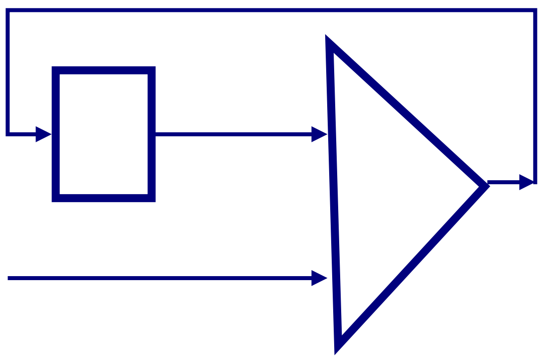
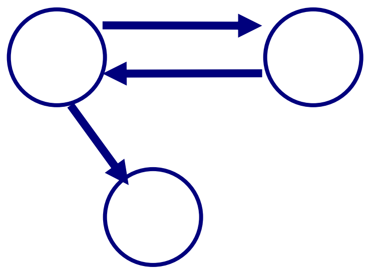
Lack of results allowing constructivity

Suggested by T. Henzinger: T. Henzinger, J. Sifakis "The Embedded Systems Design Challenge" FM06

Scientific Challenge :Two Distant Disciplines

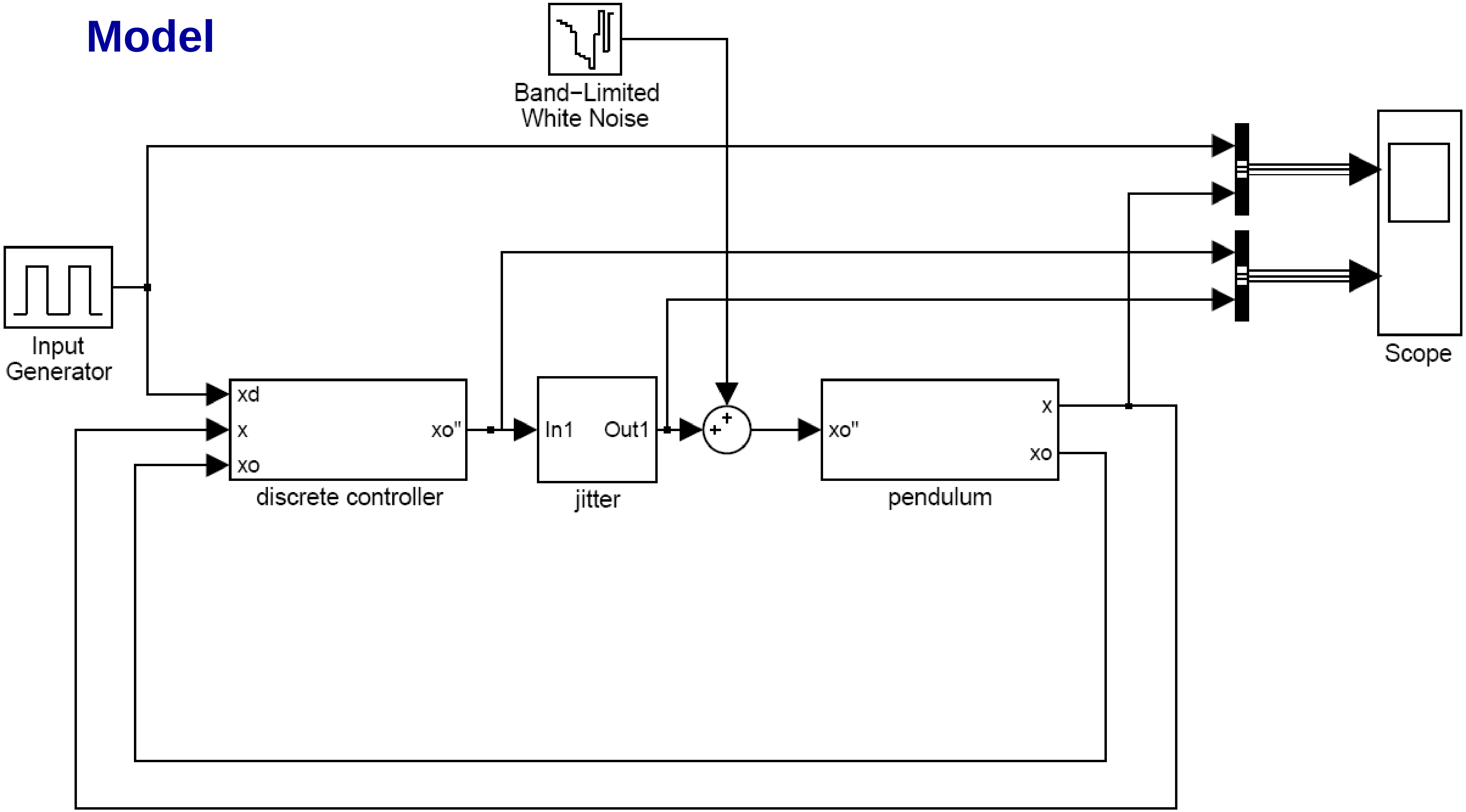
Physics	Computer Science
Studies the laws governing energy, matter and their relationships	Studies foundations of information and computation
Studies a given « reality »	Studies created universes
Physical systems – Analytic models	Computing systems – Machines
Continuous mathematics	Discrete mathematics - Logic
Differential equations Estimation theory - robustness	Automata, Algorithms and Complexity Theory
Constructivity, Predictability	Verification, Test
Mature	Promising

Integrate Analytic and Computational Modeling

	Physical Systems Engineering	Computing Systems Engineering
Component model	Transfer Function	Subroutine
Connection	Data flow	Control flow
Composition	Parallel	Sequential
		

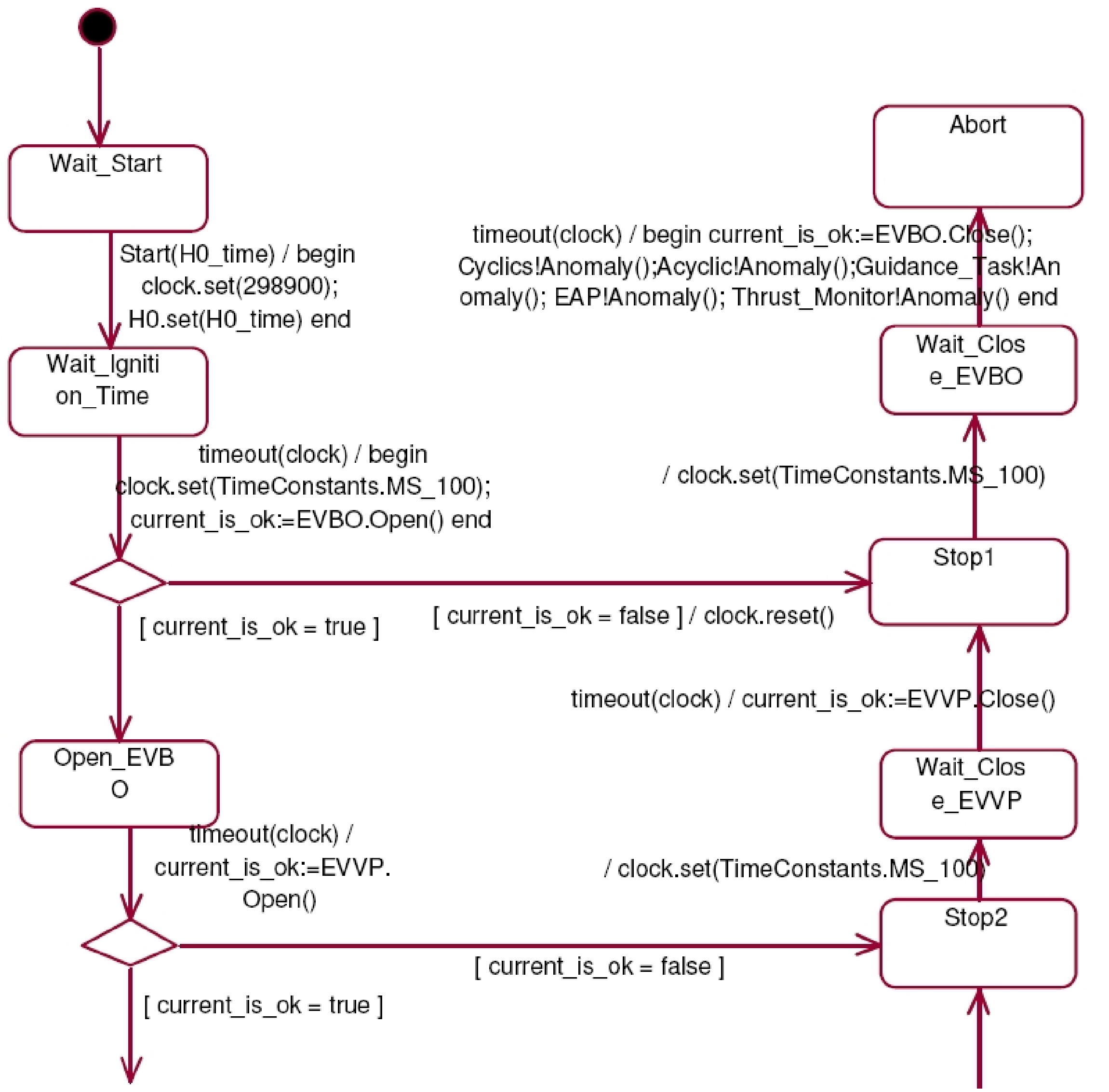
Integrate Analytic and Computational Modeling

Matlab/Simulink Model



Integrate Analytic and Computational Modeling

UML Model
(Rational Rose)



Integrate Analytic and Computational Modeling

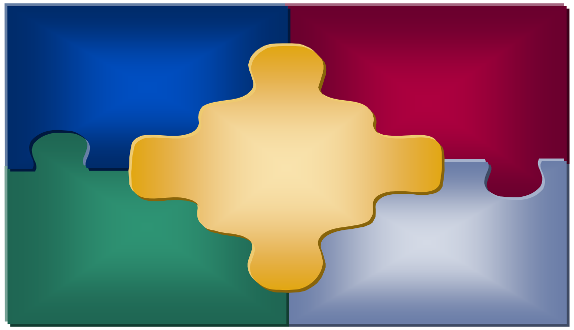
Analytic Models	Computational Models
Defined by equations Deterministic or probabilistic	Defined by programs Executable by non-deterministic machines
Strengths	
Concurrency Physical time Quantitative constraints (power, QoS, mean-time-to-failure)	Dynamic change Logical time Abstraction hierarchies, partial specifications
Analysis Techniques	
Continuous mathematics (differential equations, stochastic processes) Average-case analysis	Discrete mathematics (logic, combinatorics) Worst-case analysis
Main paradigm	
Synthesis	Verification

Encompass heterogeneity: Interoperability

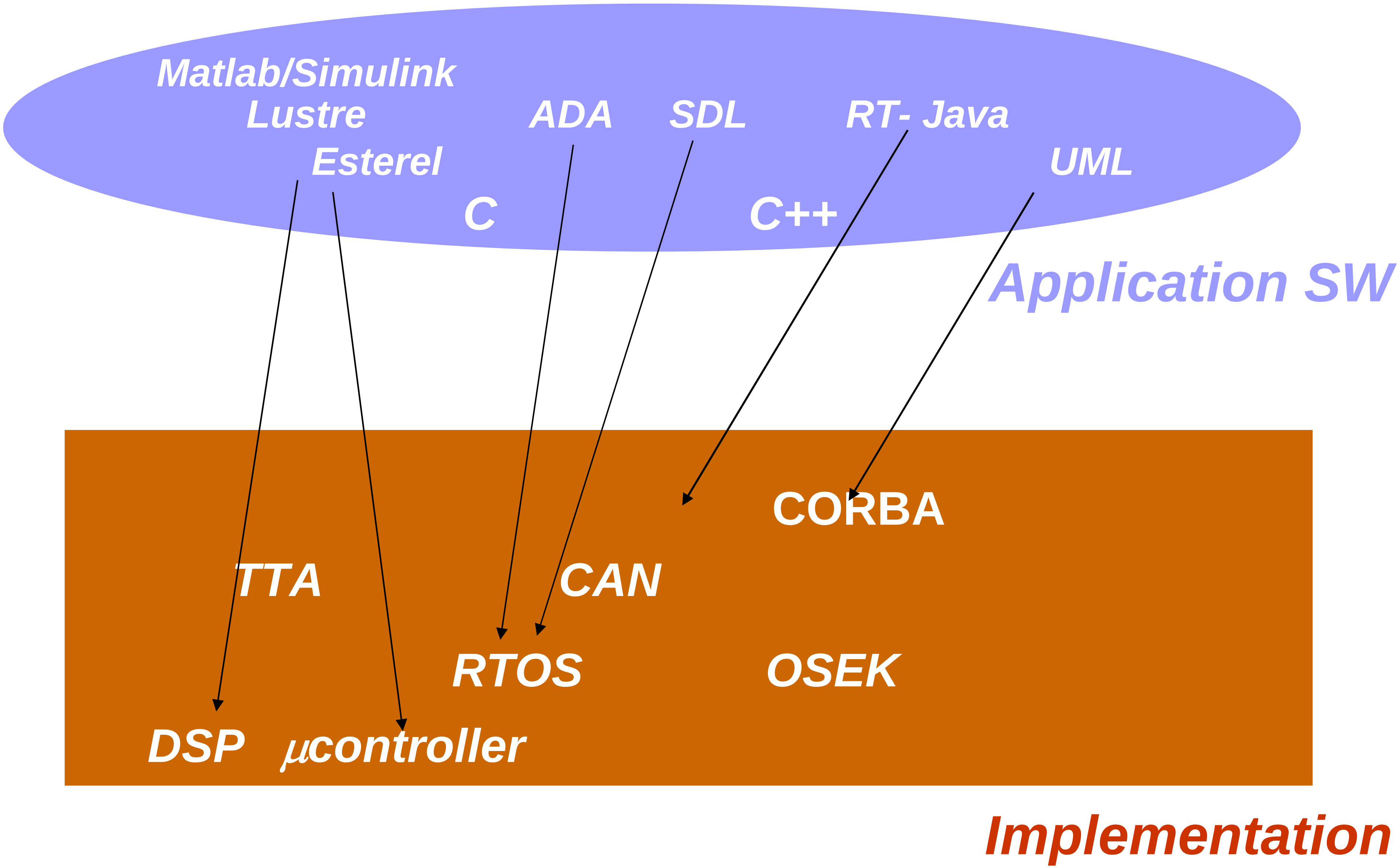
Embedded systems are built from components with different characteristics

- **Abstraction levels**: hardware, execution platform, application software
- **Execution**: synchronous and asynchronous components
- **Interaction**: function call, broadcast, rendezvous, monitors

We need a unified composition paradigm for describing and analyzing the coordination between components



Encompass heterogeneity: Abstraction Levels



Encompass heterogeneity: Abstraction Levels

Functional properties - logical abstract time

High level structuring constructs and primitives

Simplifying synchrony assumptions wrt environment

Application SW

abstraction

Non functional properties, involving time and quantities

Task coordination, scheduling, resource management,

Execution times, interaction delays, latency

Implementation

Encompass heterogeneity: Synchronous vs. Asynchronous

Application SW

Component-based Systems

- *Non interruptible execution steps*
- *Usually, a single task, on a single processor*
- *«Everybody gets something »*

- *Event triggered*
- *Multi-tasking*
  - *RTOS*
- *Usually, static Priorities*
- *«Winner takes all »*

Implementation

Encompass heterogeneity: Synchronous vs. Asynchronous

Synchronous execution:

- In a given step all sequential units have some time budget.
- Steps are non interruptible; should be small enough to ensure reactivity; implemented by strong synchronization

Asynchronous execution:

- No predefined execution step. Fairness is enforced by priorities (preemption of lower priority sequential units)

Encompass heterogeneity: Interaction

Interactions

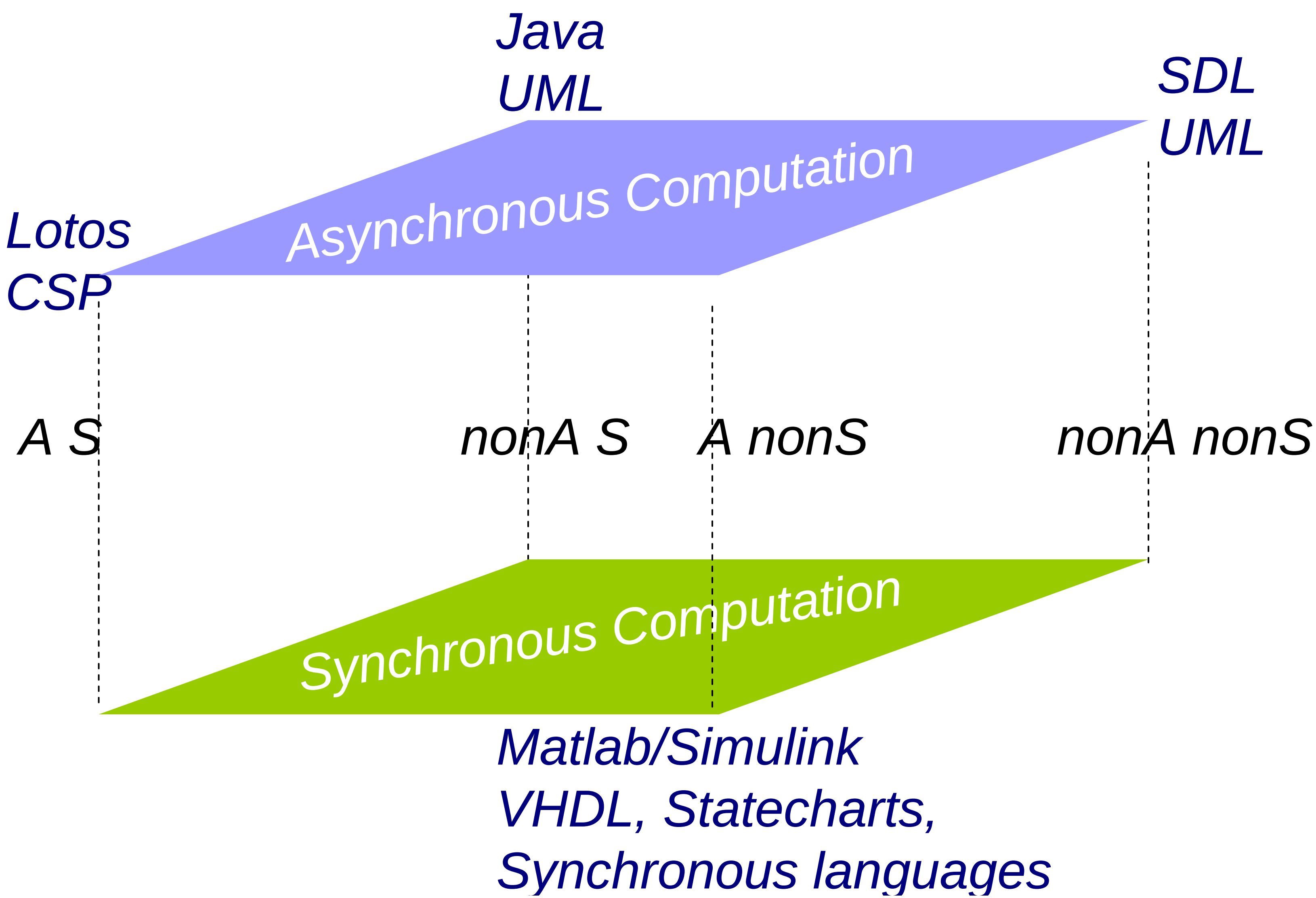
- can involve strong synchronization (rendezvous) as in CSP or weak synchronization (broadcast) as in SDL, Esterel
- can be atomic as in CSP, Esterel or non atomic as in SDL
- can be binary (point to point) as in CCS, SDL or n-ary as in Lotos

*We need a **unified composition paradigm** for describing and analyzing the coordination between components.*

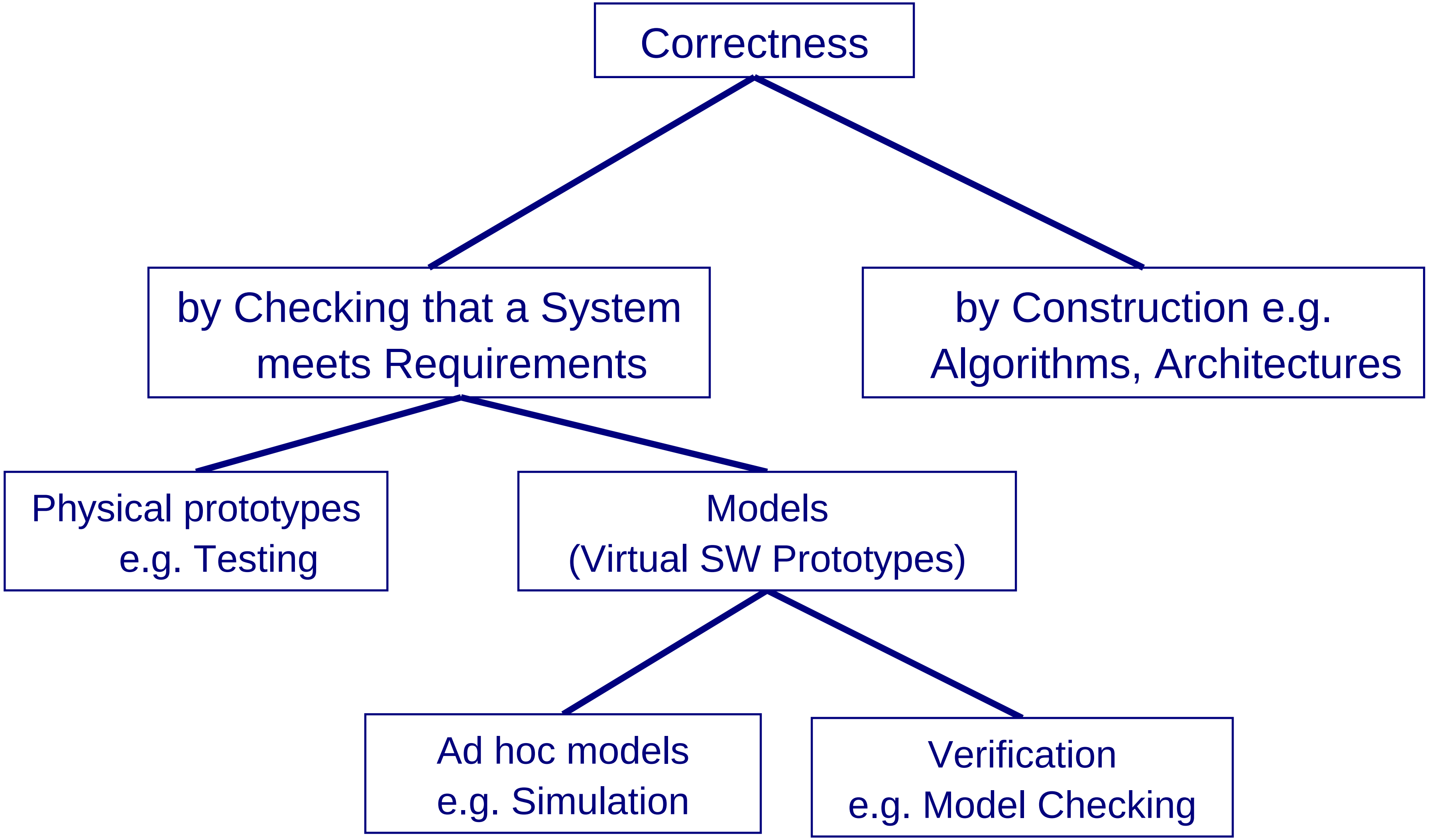
Such a paradigm would allow system designers
and implementers to formulate their solutions
*in terms of **tangible, well-founded and organized concepts***
instead of using dispersed low-level coordination mechanisms including semaphores, monitors, message passing, remote call, protocols etc.

Encompass heterogeneity: Example

A: Atomic interaction *S: Strong synchronization*



Cope with Complexity: Achieving Correctness



Cope with Complexity: Verification

Three essential ingredients

- **Requirements**
  describing the expected behavior, usually as a set of properties

- **Models**
  describing a transition relation on the system states

- **Methods**
  for checking that a system model satisfies given requirements

Cope with Complexity: Compositionality

Today, *a posteriori system verification* at high development costs limited to medium complexity systems

Tomorrow, compositional construction: at design time, infer properties of a composite system from properties of its components

Cope with Complexity: Compositionality

Today, *a posteriori system verification* at high development costs limited to medium complexity systems

Tomorrow, compositional construction: at design time, infer properties of a composite system from properties of its components

Cope with Complexity: Compositionality

Develop compositionality results

- For particular
  - architectures (e.g. client-server, star-like, time triggered)
  - programming models (e.g. synchronous, data-flow)
  - execution models (e.g. event triggered preempable tasks)
- For specific classes of properties such as deadlock-freedom, mutual exclusion, timeliness

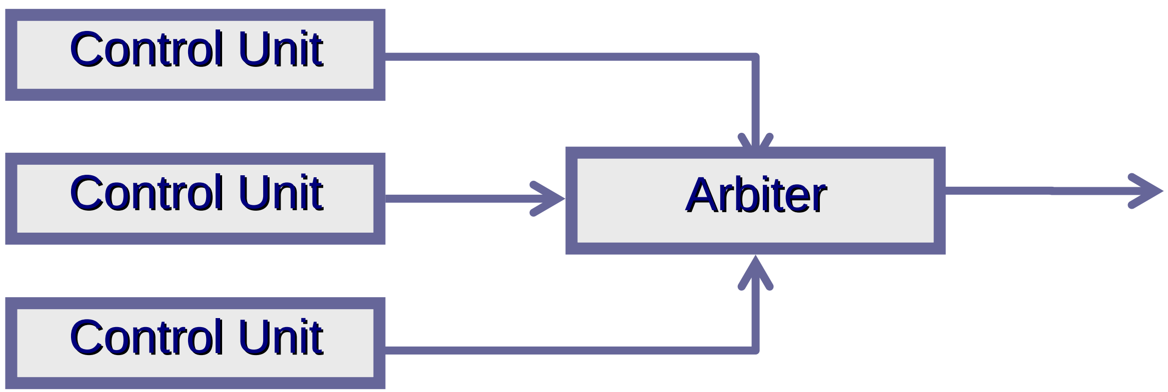
Compositionality rules and combinations of them lead

- to “verifiability” conditions, that is conditions under which verification of a particular property becomes much easier.
- to correct-by-construction results

Cope with Uncertainty: Adaptivity

Systems must ensure a given service, in interaction with uncertain and unpredictable environments

Today, to cope with uncertainty, systems are over-sized and make a sub-optimal use of their resources :
static and separated allocation for each critical service



Tomorrow, adaptive systems ensuring optimal, dynamic and global resource management for enhanced predictability and use of resources

Cope with Uncertainty: Adaptive System

Learning

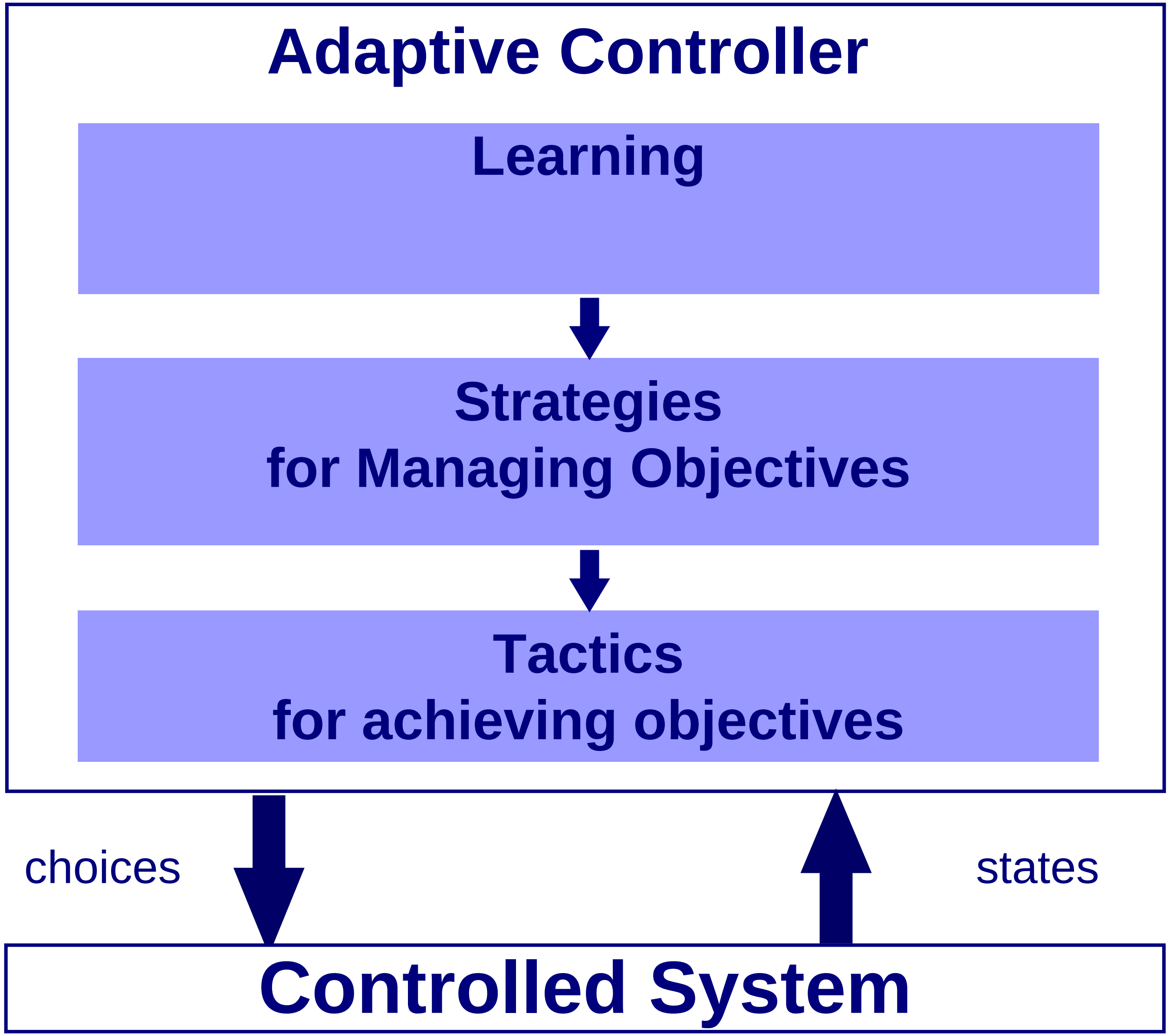
Movie would have been better …

Managing Conflicting Objectives

Go to: 1) Stadium 2) Movie 3) Restaurant

Planning

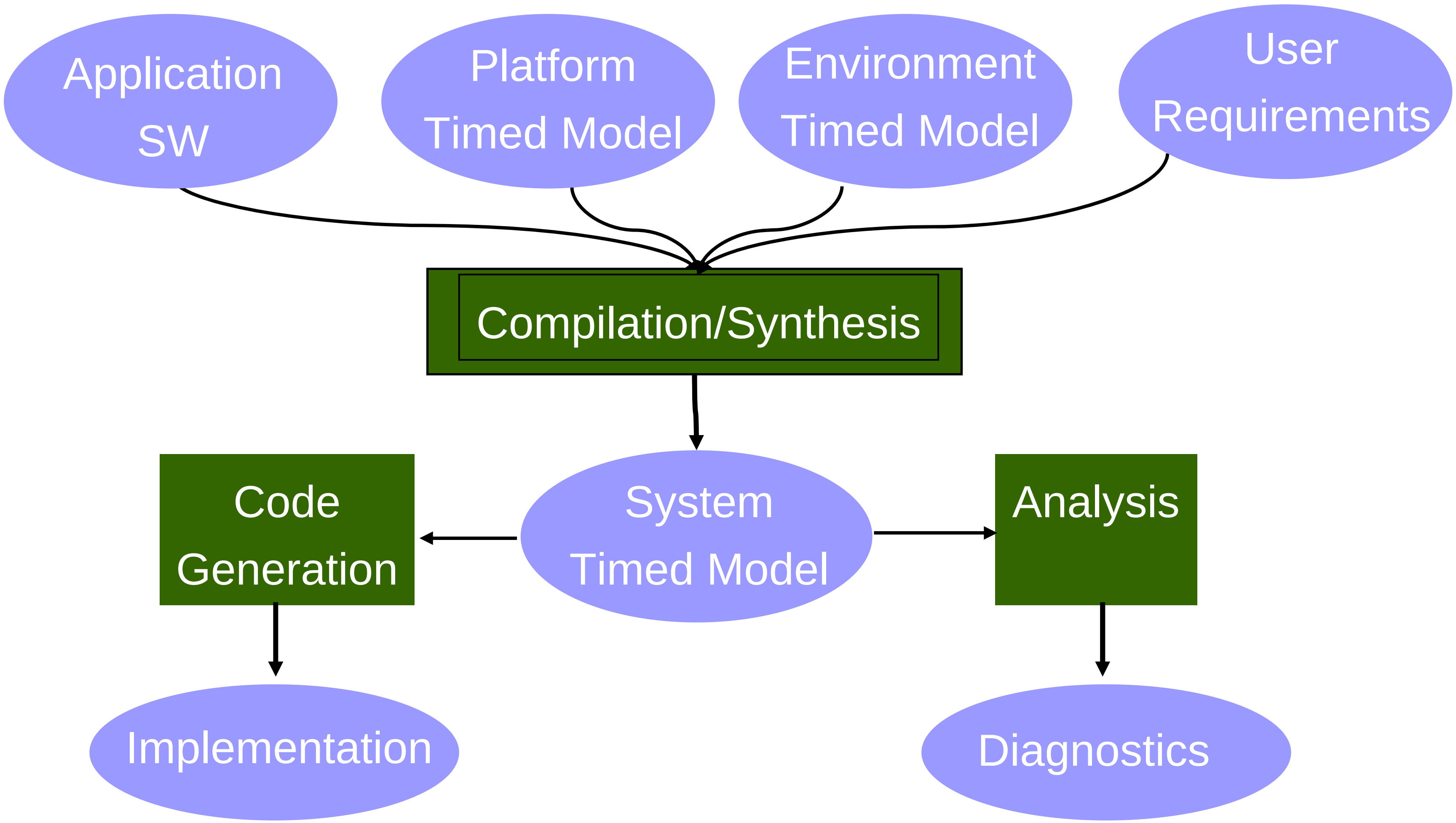
Cope with Uncertainty: Adaptive System



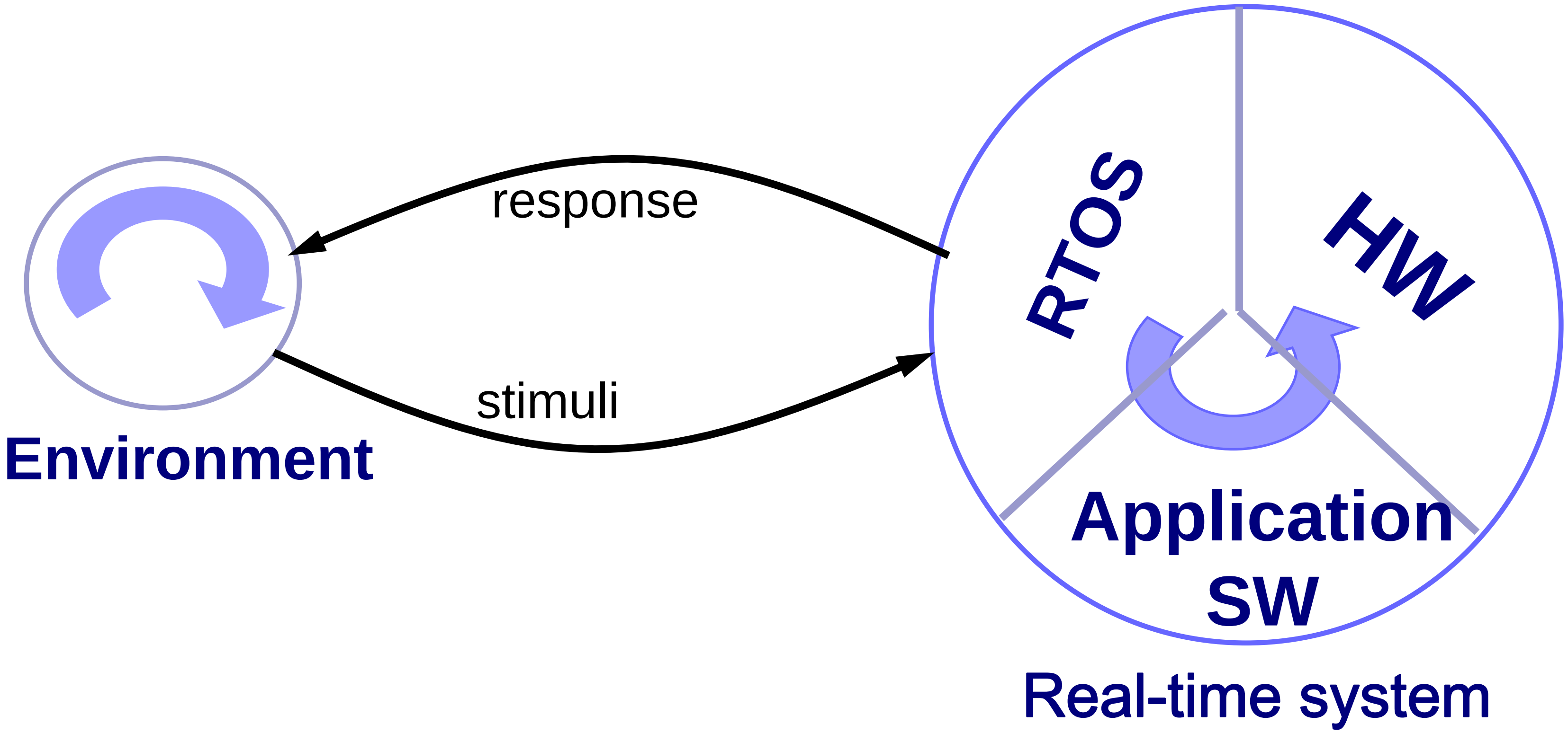
OVERVIEW

- Embedded Systems
- Scientific Challenge
- Work Directions

Model-based Development

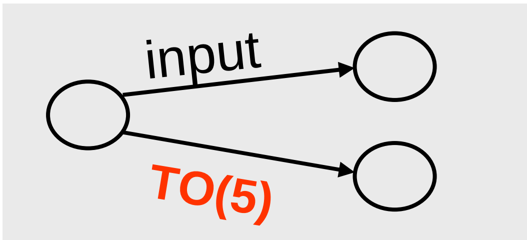
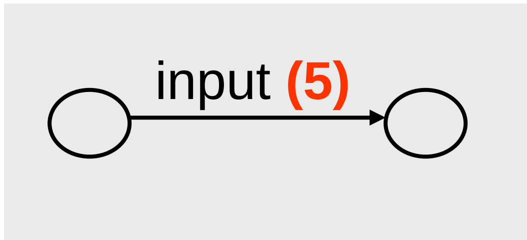


Model-based Development – Timed model

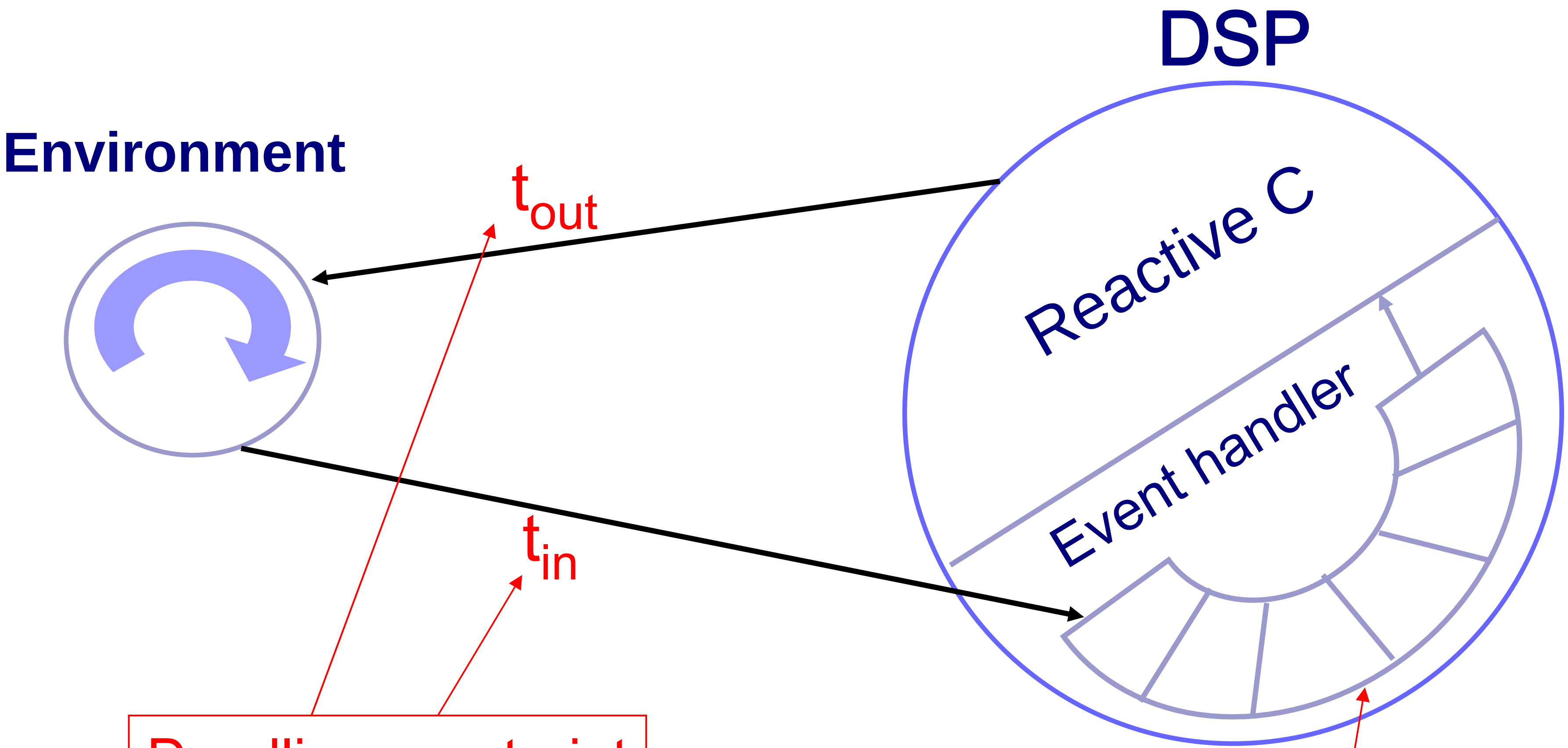


A Timed Model of a RT system can be obtained by "composing" its application SW with constraints e.g. timing, induced by both its execution and its external environment

Model-based Development – Timed model

	Application SW	Timed model
Description	Program - Reactive machine	Reactive machine + Environment + Platform
Time	Reference to physical (external) time	Quantitative (internal) time - Consistency problems
Statements	No assumption about Execution Time Platform-independence	Assumptions about Execution Times Platform-dependence
Time Triggered Events	Timeouts to control waiting times 	Time constraints on interactions 

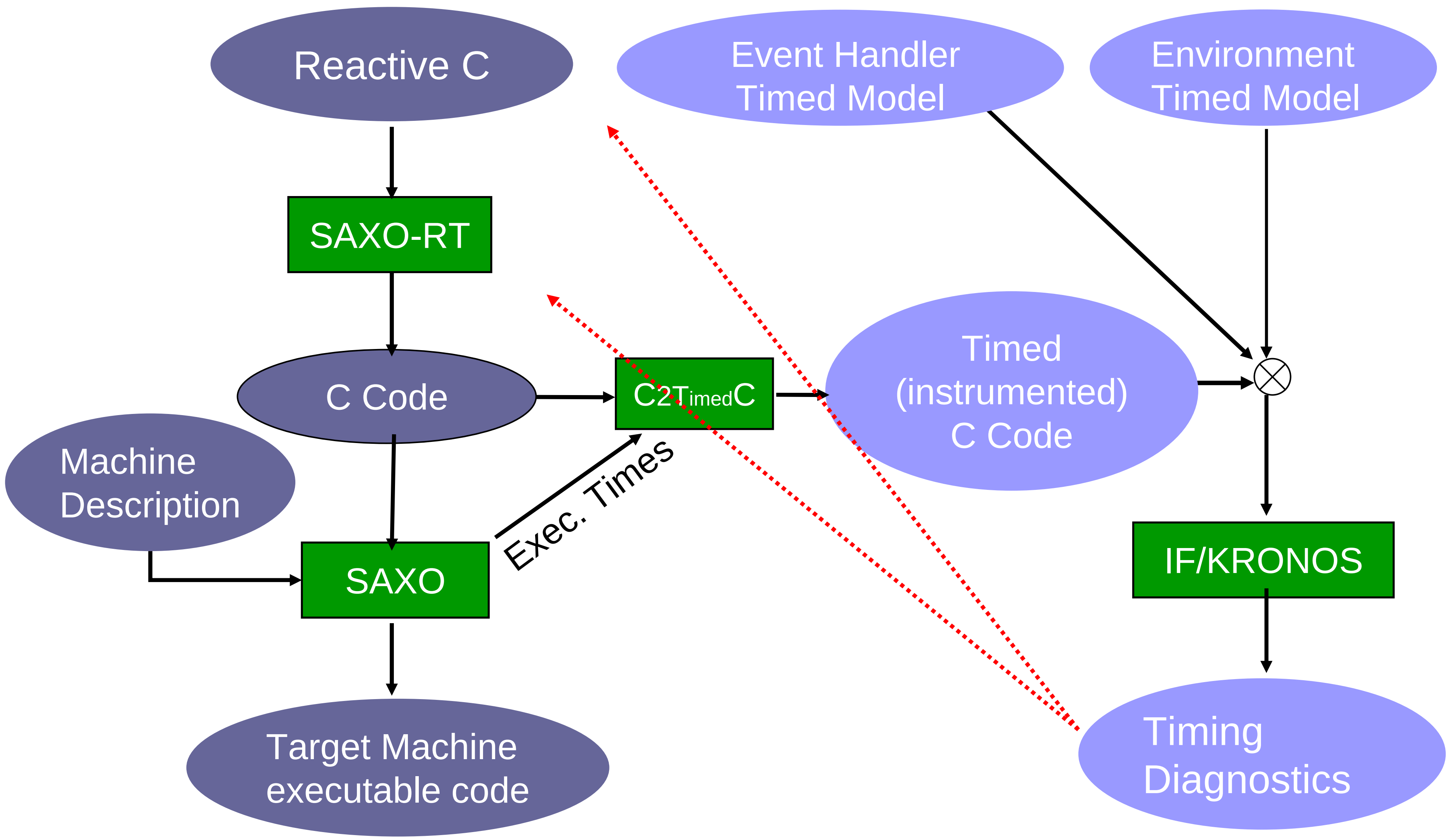
Model-based Development – An example (1/3)



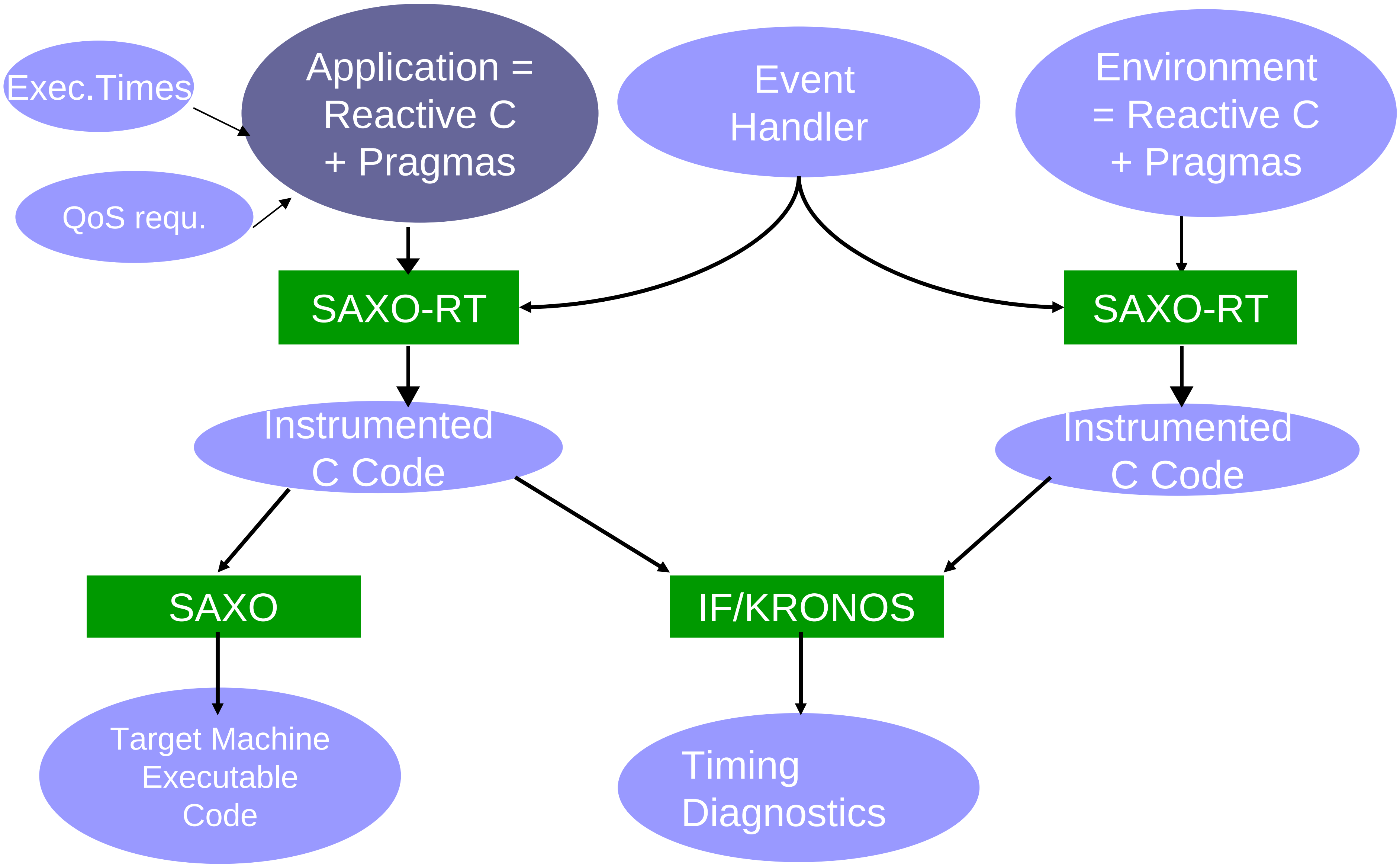
Deadline constraint
$t_{out} - t_{in} < D$

Throughput constraint:
no buffer overflow

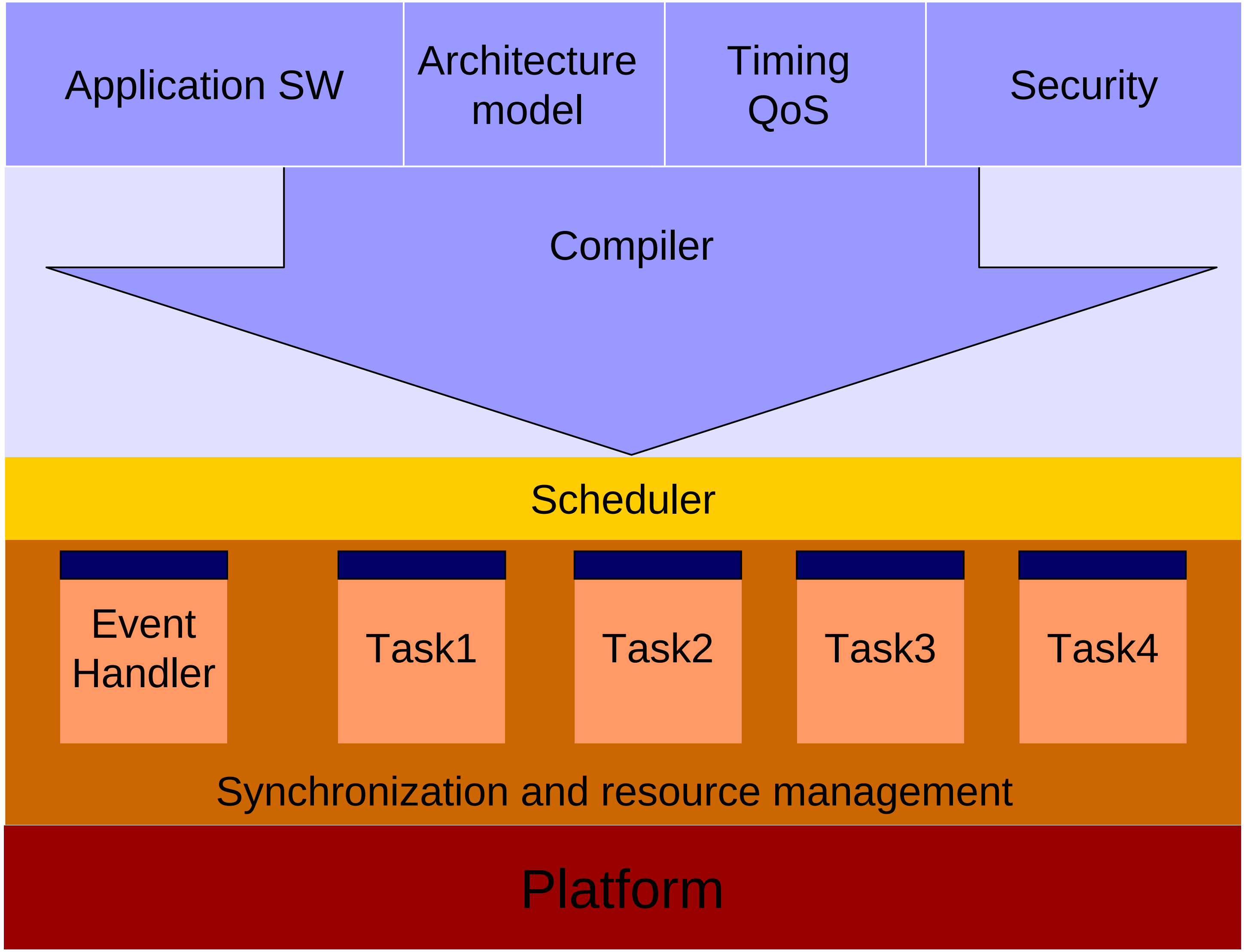
Model-based Development – An example (2/3)



Model-based Development – An example (3/3)



Resource-aware Compilation



Operating Systems

Operating systems are often:

- Far more complex than necessary
- Undependable
- With hidden functionality
- Difficult to manage and use efficiently

Move towards standards dedicated to specific domains
Ex: OSEK, ARINC, JavaCard, TinyOS

- Minimal architectures, reconfigurable, adaptive, with features for **safety** and **security**
- Give up control to the application –
  move resource management outside the kernel
- Supply and allow adaptive scheduling policies which take into account the environmental context (ex: availability of critical resources such as energy).

Control for Embedded Systems

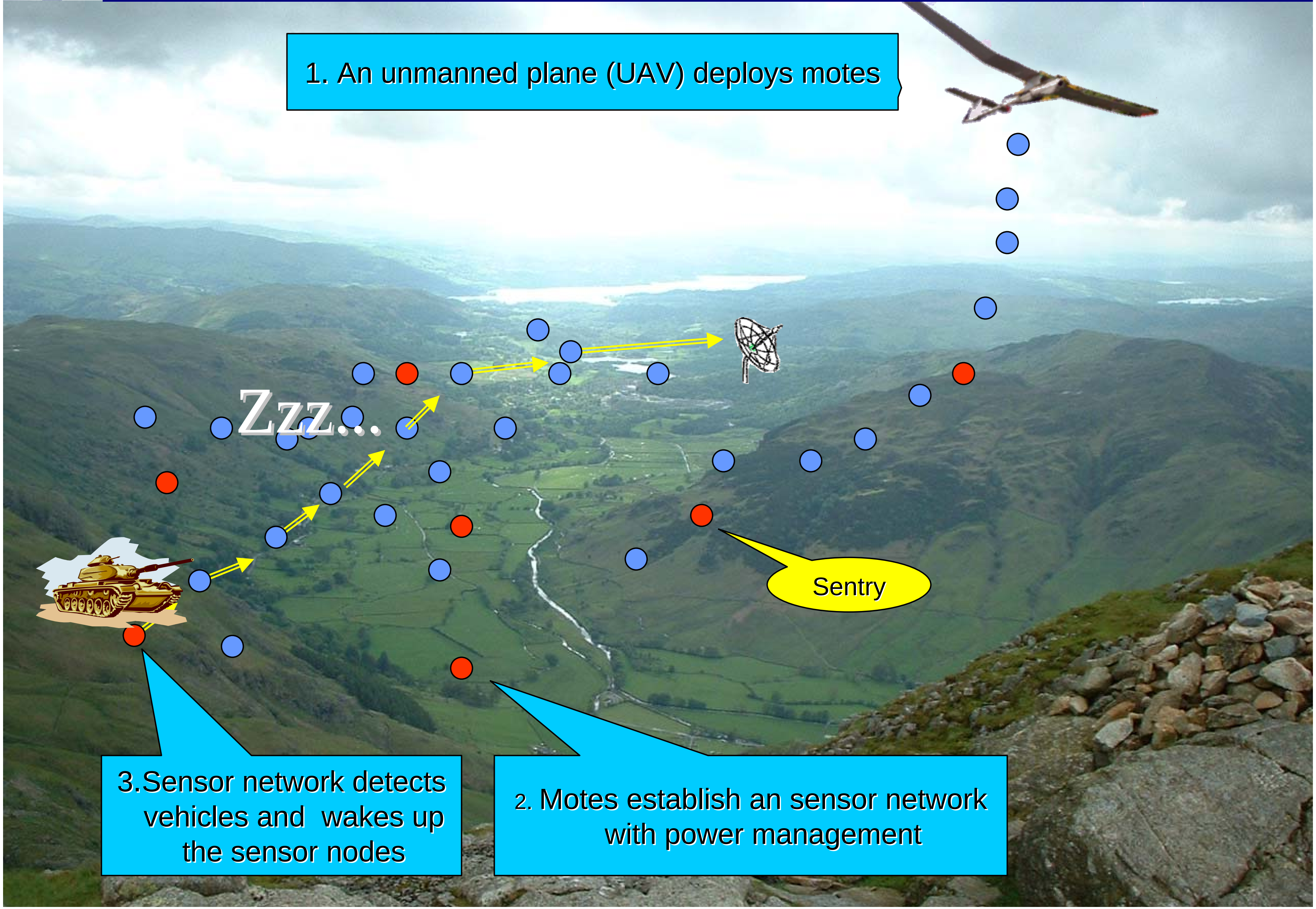
Automation applications are of paramount importance – their design and implementation raise difficult problems

Hybrid Systems – active research area

- Combination of continuous and discrete control techniques
- Multi-disciplinary integration aspects (control, numerical analysis, computer science)
- Modeling and Verification
- Distributed and fault-tolerant implementations (influence communication delays, clock drift, aperiodic sampling)

↳ Use of control-based techniques for adaptivity

Sensor Networks



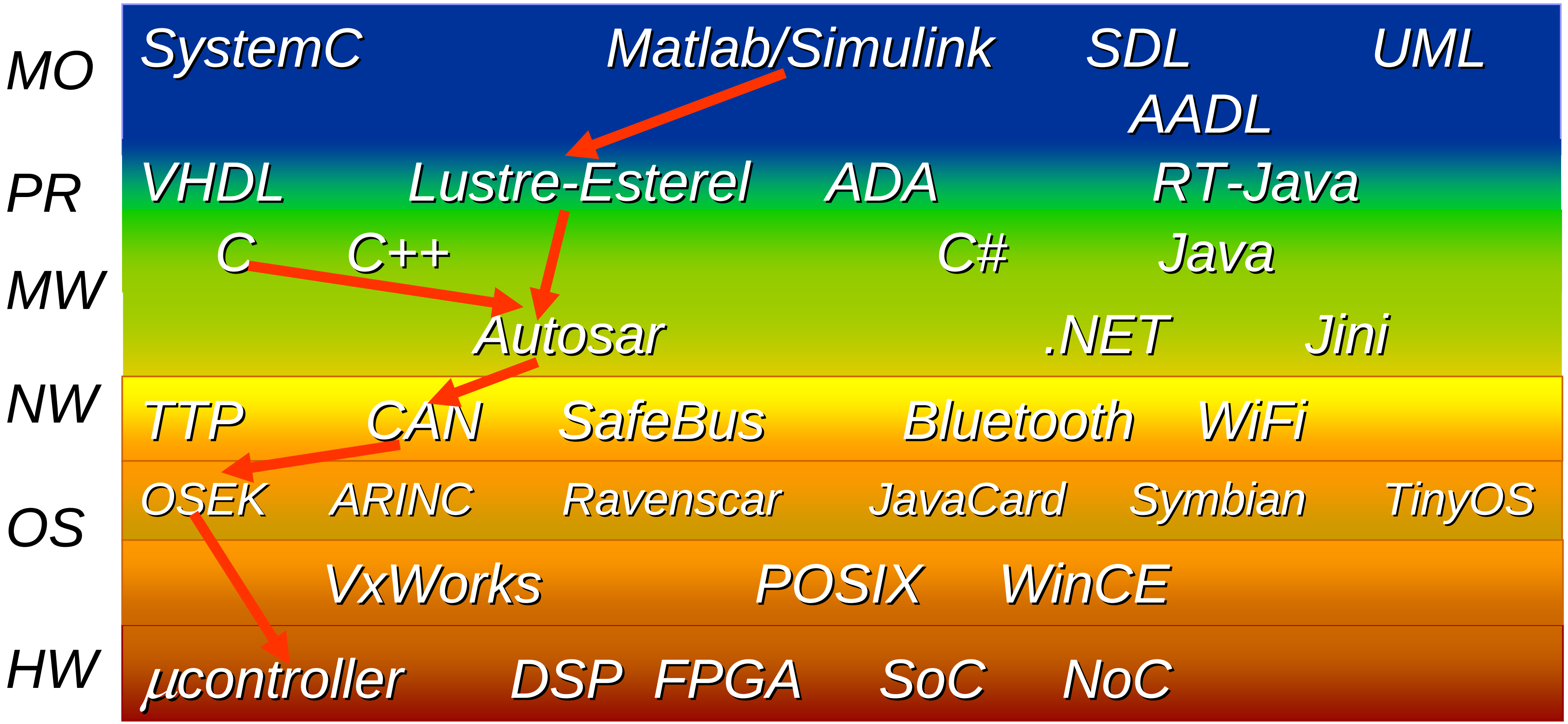
Dependability

- Traditional techniques based on massive redundancy are of limited value
- Dependability should be a guiding concern from the very start of system development. This applies to programming style, traceability, validation techniques, fault-tolerance mechanisms, ...

Work Directions :

- Methodologies for domain-specific standards, such as :
  - DO178B Process Control Software Safety Certification
  - Integrated Modular Avionics; Autosar
  - Common Criteria for Information Technology Security Evaluation
- Verification Technology (verify resistance to certain classes of errors and attacks) – **certification**
- Architectures**,** protocols and algorithms for fault-tolerance and security taking into account QoS requirements (real-time, availabability)

Integration of Methods and Tools



THANK YOU