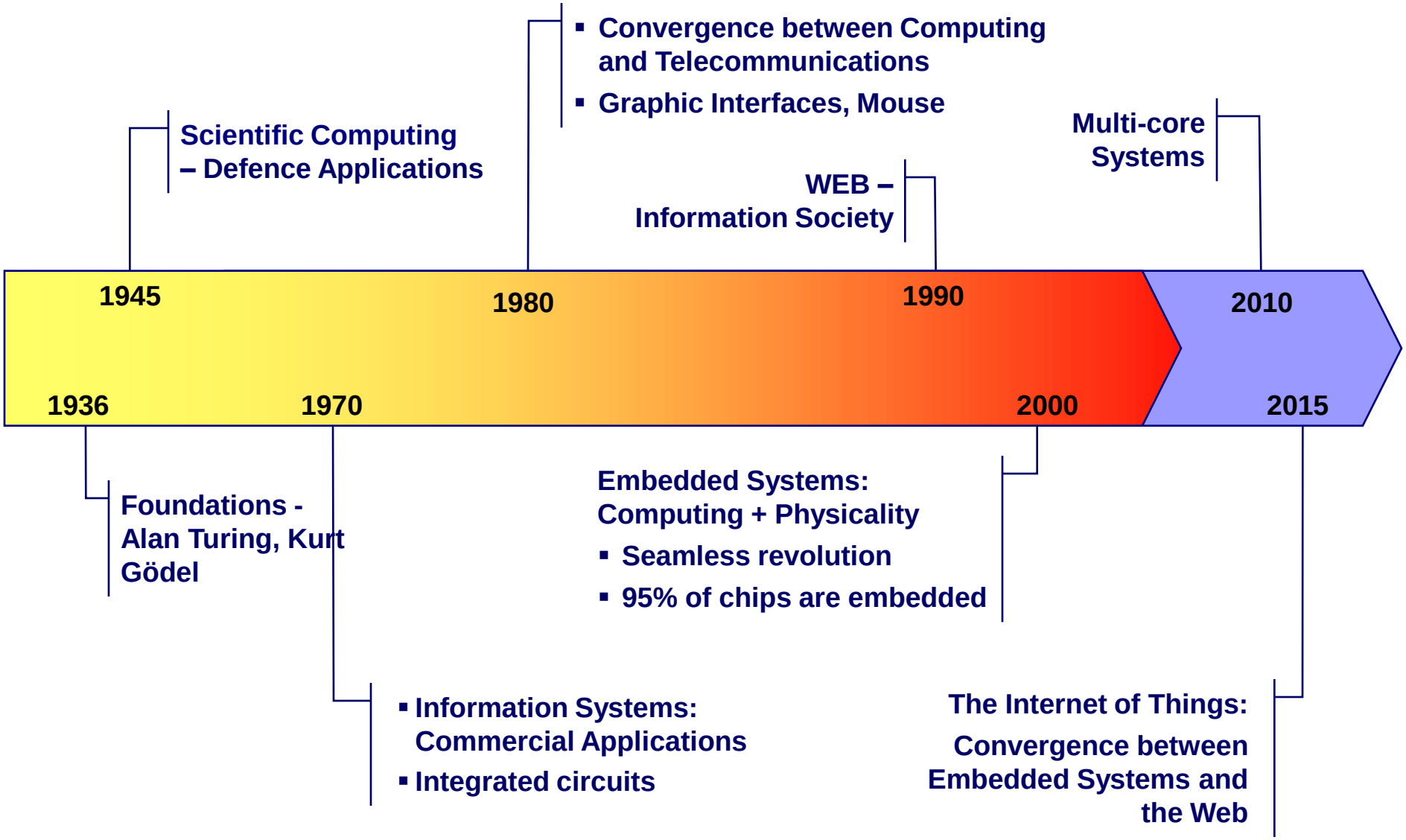
Embedded Systems Design – Scientific Challenges and Work Directions

ECSS 09

Paris, October 9, 2009

Joseph Sifakis
VERIMAG Laboratory

The Evolution of Informatics



Informatics is a young discipline, driven by exponential growth of components and their applications.

Embedded Systems

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

OVERVIEW

- System Design Today
- Research Challenges
  - Marry Physicality and Computation
  - Encompass Heterogeneity – Components
  - Cope with Complexity – Constructivity
  - Cope with Uncertainty – Adaptivity
- Embedded Systems Design
- Discussion

System Design – Trends

Embedded systems break with ordinary IT technologies.

It is hard to jointly 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
- **Dependability :** guaranteed service in any case
  Ex : attacks, hardware failures, software execution errors

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

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

System Design – 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 : active safety, health, autonomous robotic devices

- Successful integration of heterogeneous systems of systems
  - ***Internet of Things***
  - ***Automated Transport Systems***
  - ***Smart Grids***
  - ***« Ambient Intelligence»***

System Design – a Long Way to go

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

System Design – a Long Way to go

The design of large IT systems

e.g. microprocessors, mobile telecommunication platforms, web application platforms is a risky undertaking mobilizing hundreds of engineers for several years

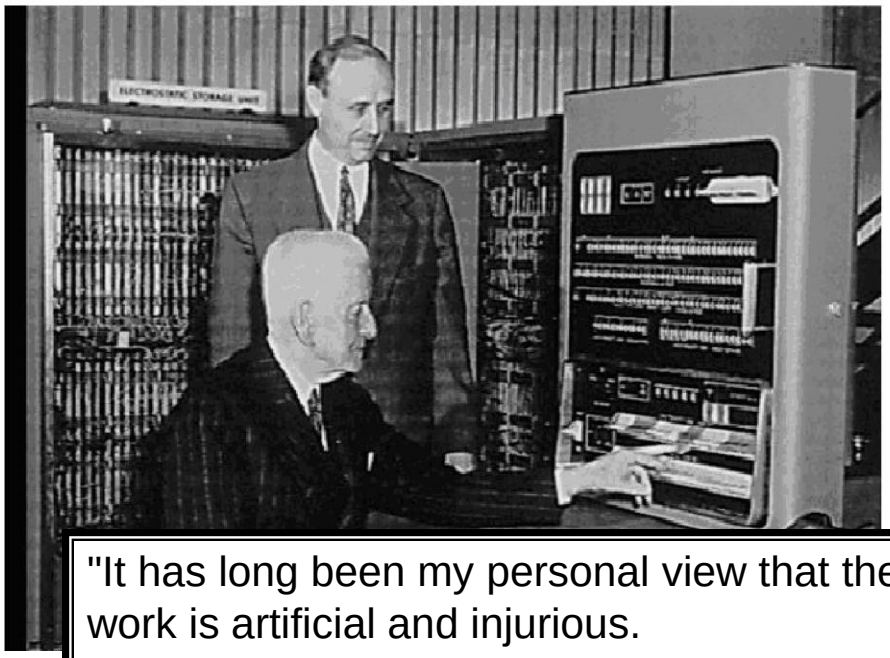
Difficulties

- Complexity – mainly for building systems by reusing existing components
- Requirements are often incomplete, and ambiguous (specified in natural language)
- Design approaches
  - are empirical and based on the expertise and experience of teams
  - reuse/extend/improve solutions that have proved to be efficient and robust

Consequences

- Very often large IT projects go over budget, over time, deliver poor quality
- Of these, 40% fail, 30% partially succeed, 30% succeed

System Design – a Long Way to go



There is an increasing gap between:

- Our technological capabilities for treating and transmitting information
- Our know-how in computing systems engineering

"It has long been my personal view that the separation of practical and theoretical work is artificial and injurious.

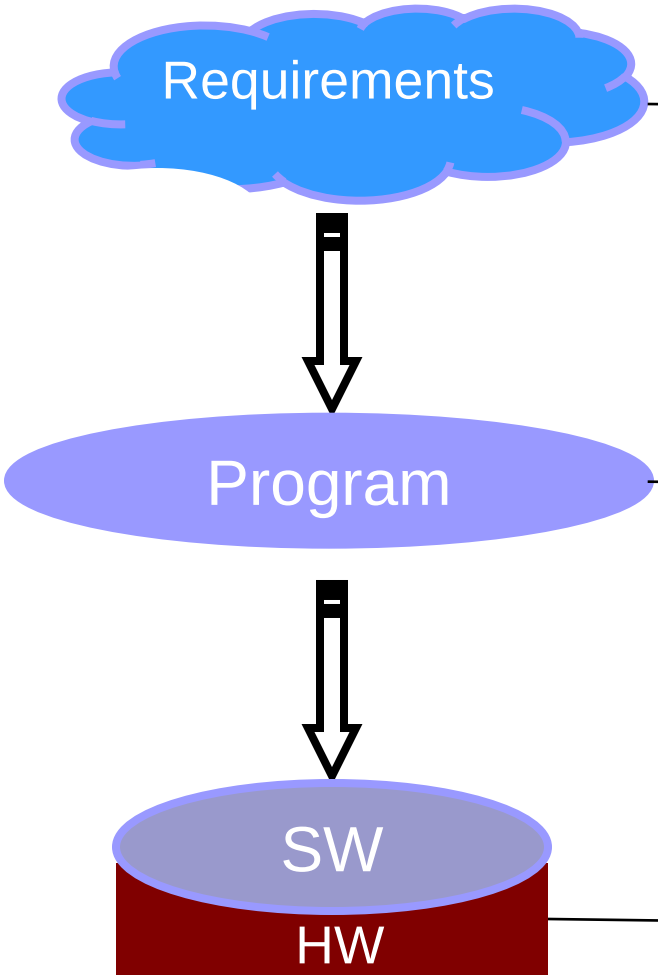
Much of the practical work done in computing, both in software and in hardware design, is unsound and clumsy because the people who do it have not any clear understanding of the fundamental design principles of their work.

Most of the abstract mathematical and theoretical work is sterile because it has no point of contact with real computing.

Christopher Strachey (1916-1975)

System Design – Simplified View

Design is the process of deriving from given requirements, an executable model from which a system can be generated (more or less automatically).



The expected behavior of the system to be designed with respect to its potential users and its environment

Executable platform-independent model meeting the requirements

System composed of HW and SW – the HW platform my be given

System Design – Essential Properties

Correctness

- Design methodology ensuring correct implementation from a system model

Productivity

- Reuse, separate compilation,
- Support for heterogeneous programming models, DSL
- Natural expression of data parallelism and functional parallelism

Performance

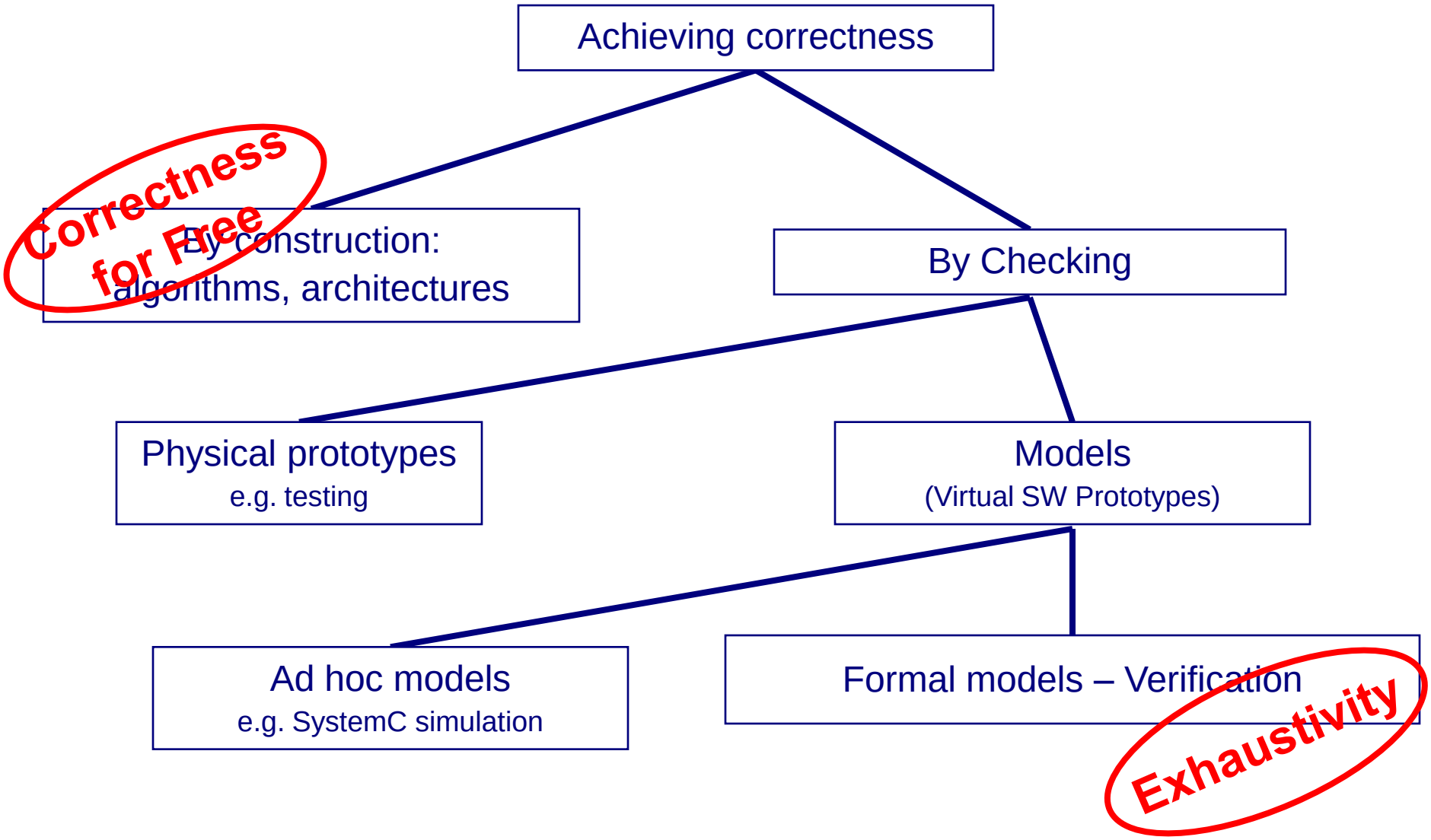
- Optimal use of physical resources

Parsimony

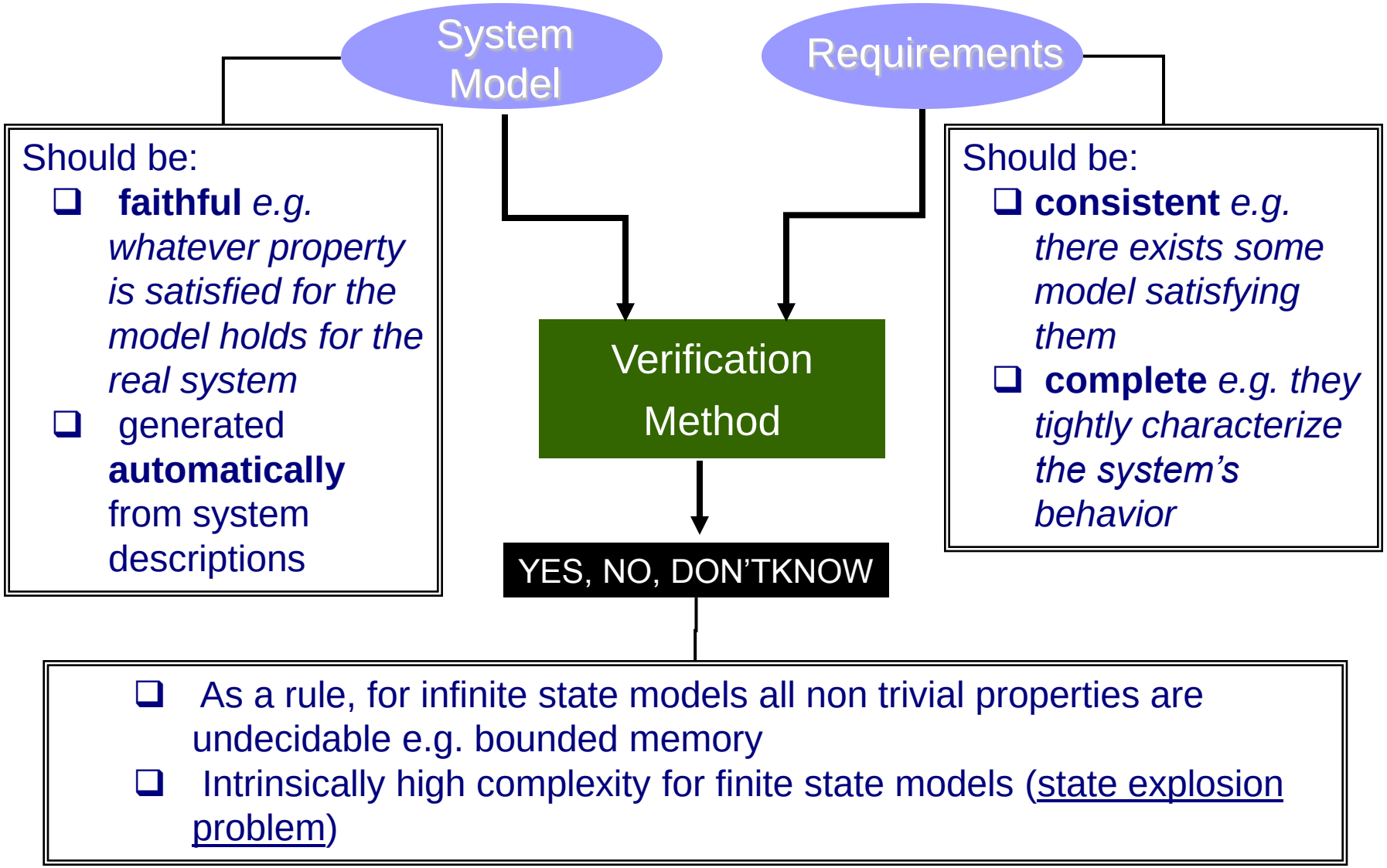
- Design choices are only implied by requirements – no superfluous constraints
- Use degrees of freedom in the design process, e.g. parallelism or non-determinism, for choosing the “best” implementation

Achieving Correctness

Correctness: a system is correct if it meets its requirements



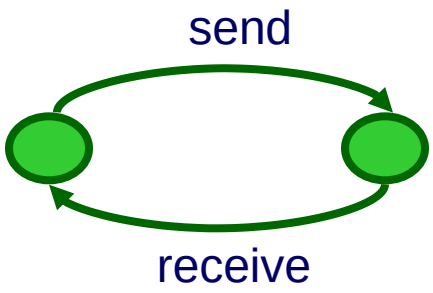
Achieving Correctness - Verification



Achieving Correctness - Requirements specification

State-based

Using a machine (monitor) to specify observable behavior



Good for characterizing causal dependencies e.g. sequences of actions

Property-based

Using formulas, in particular *temporal logic*, to characterize a set of execution structures e.g. traces, execution trees

always(inev (enable(send)))

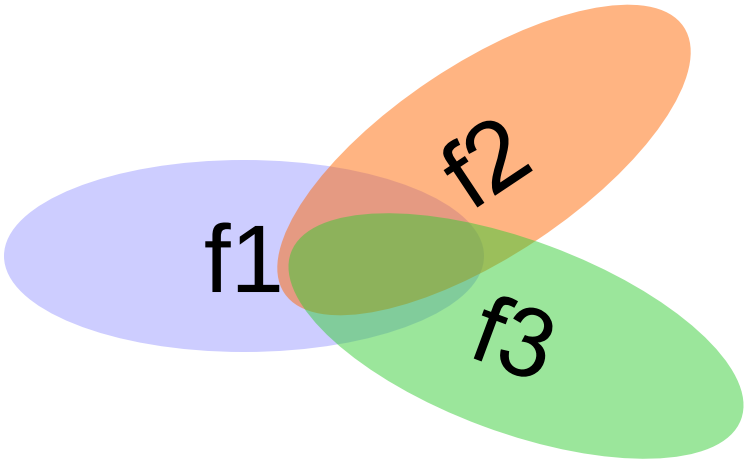
always(inev (enable(receive)))

Good for expressing global properties such as mutual exclusion, termination, fairness

We need a combination of both property-based and state-based styles

Achieving Correctness - Requirements specification

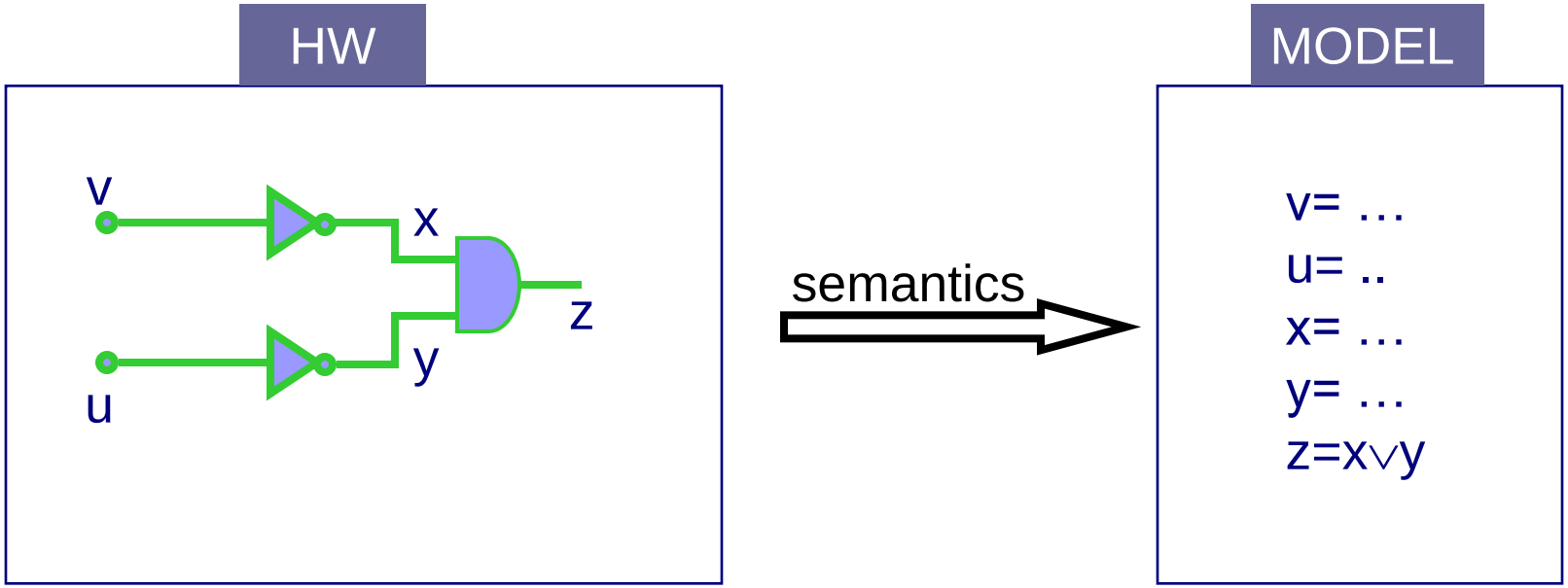
- Temporal logic was a breakthrough in understanding and formalizing requirements for concurrent systems e.g. mutex, fairness
- Nonetheless, the declarative style is not always easy to master and understand - Moving towards a “less declarative” style e.g. MSC, modal automata



- We need requirement specification languages for engineers e.g. PSL/Sugar
- Much to be done for extra-functional requirements characterizing:
  - security (e.g. privacy properties, electronic voting)
  - reconfigurability (e.g. non interference of features)
  - quality of service (e.g. degree of jitter).

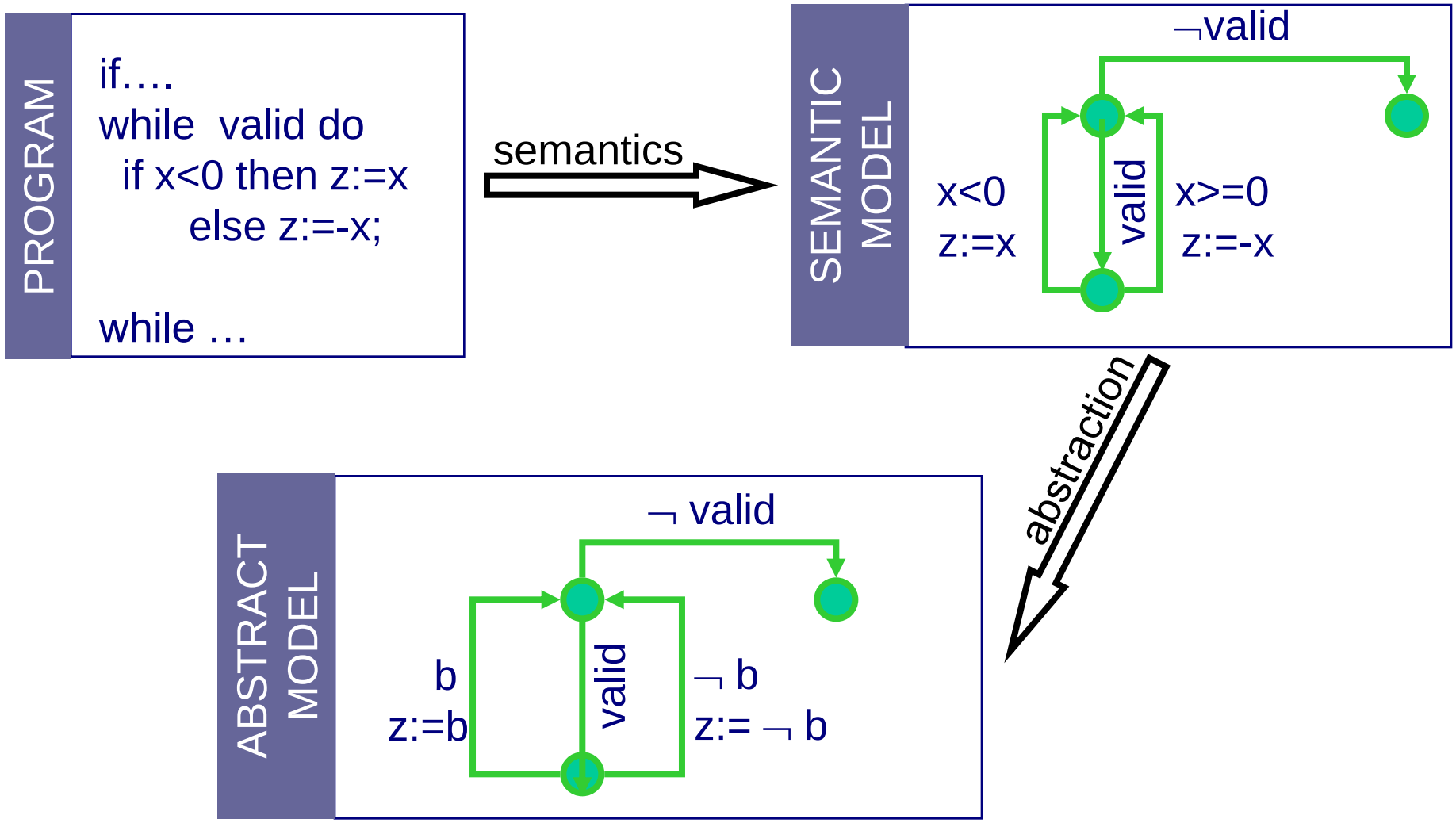
Achieving Correctness - Building models

For hardware, it is easy to get faithful logical finite state models represented as systems of boolean equations



Achieving Correctness - Building models (2/3)

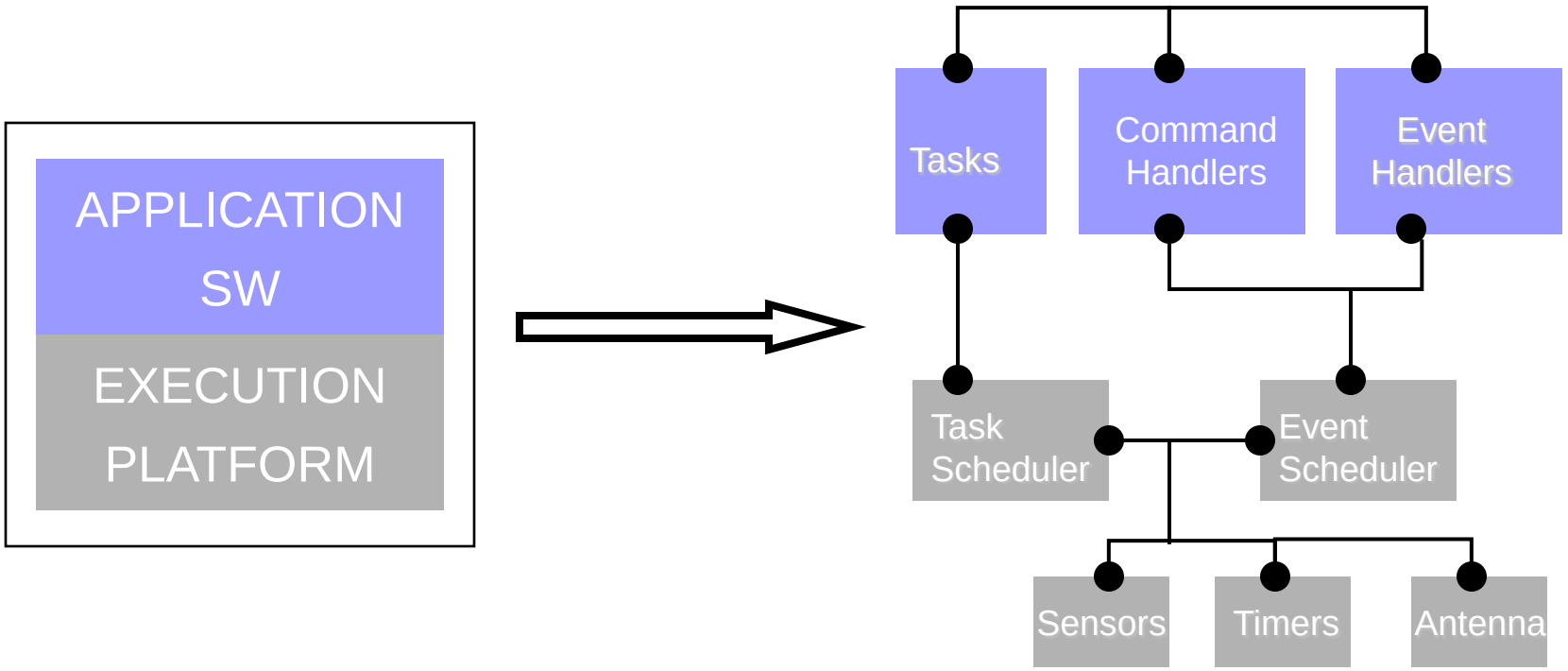
For software this may be much harder ….



Achieving Correctness - Building models (3/3)

For mixed Software / Hardware systems

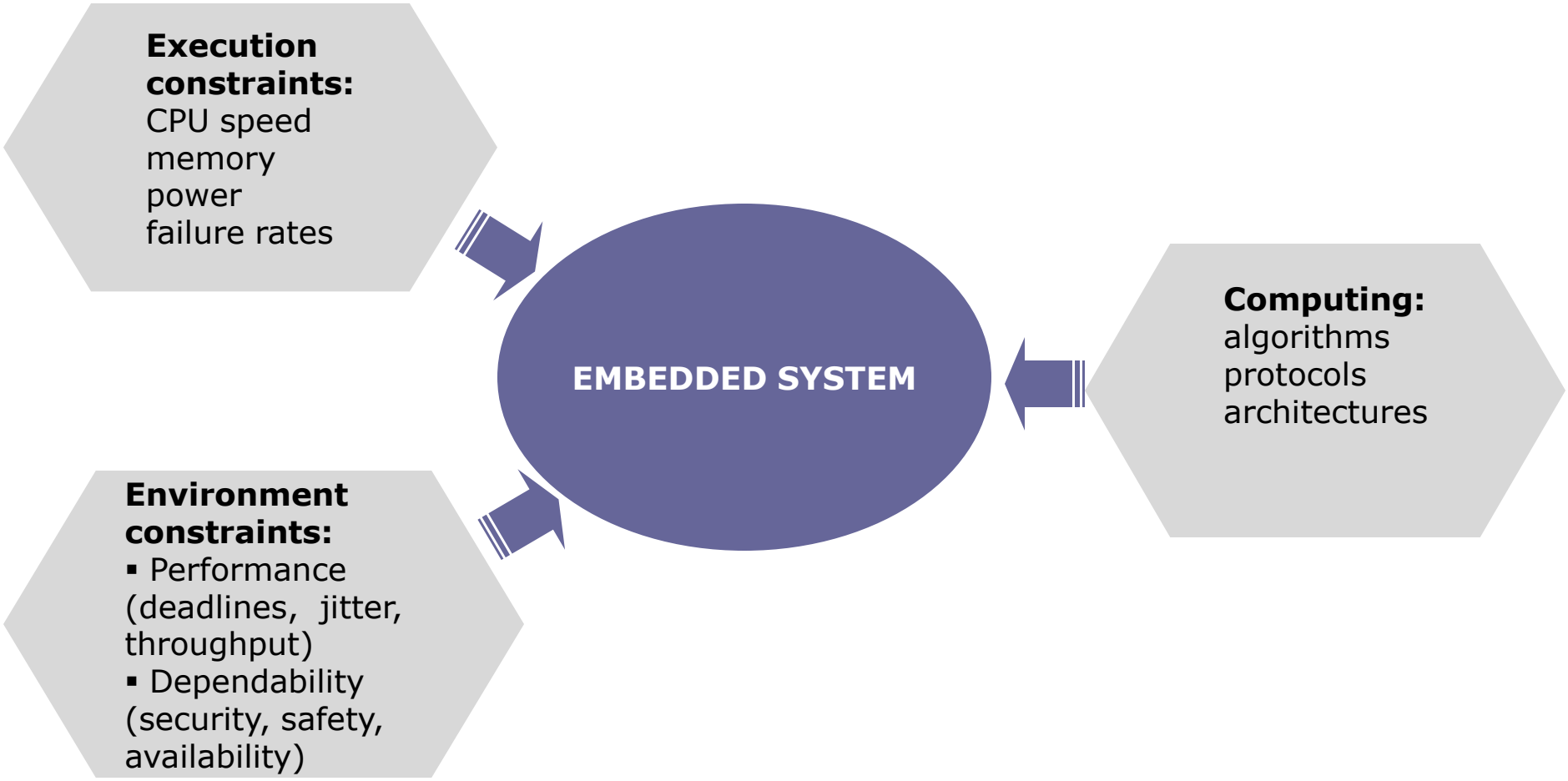
- there are no faithful modeling techniques as we have a poor understanding of how **software** and the underlying **platform** interact
- validation by testing physical prototypes or by simulation of ad hoc models



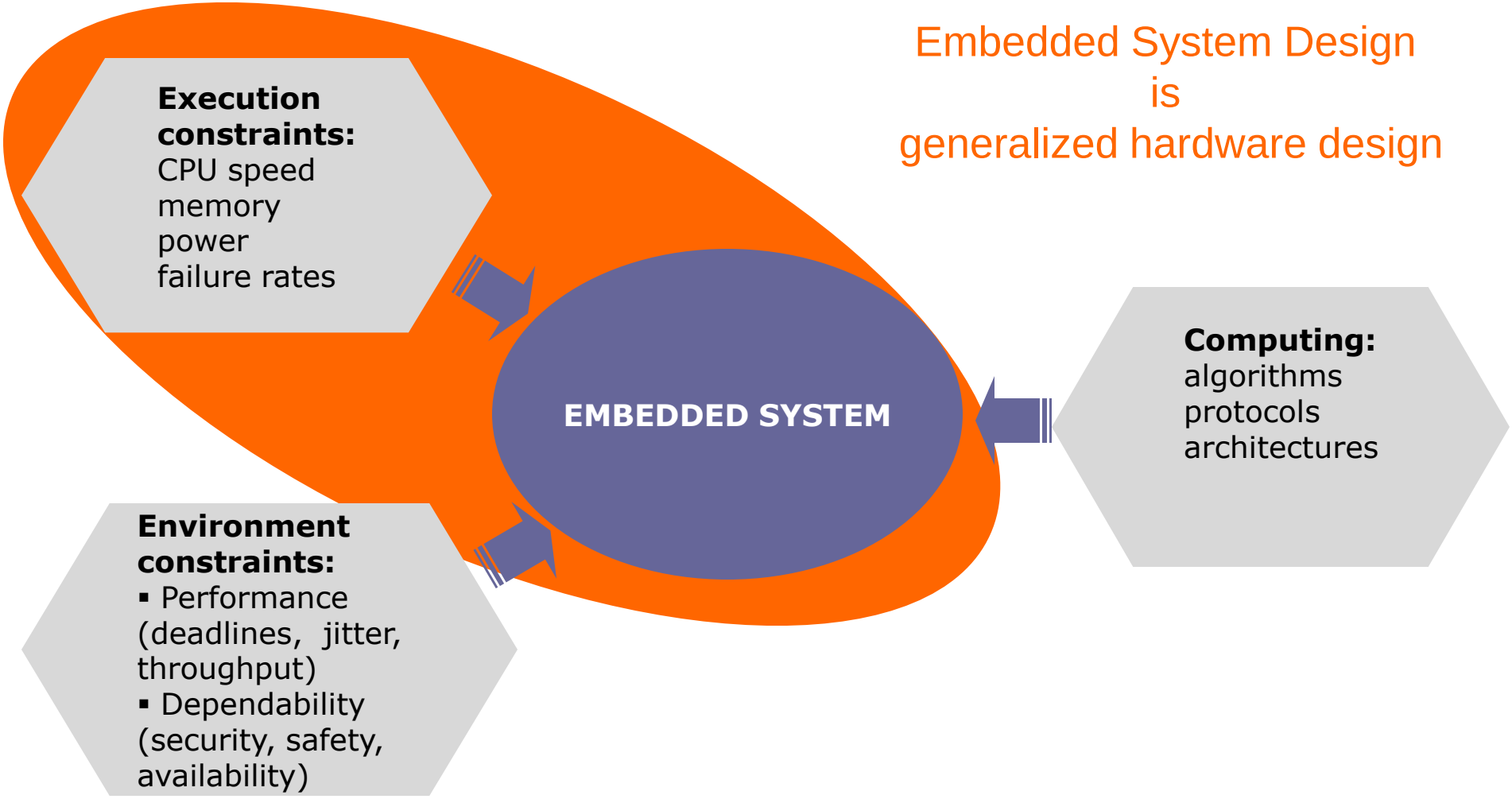
- System Design Today
- Research Challenges
  - Marry Physicality and Computation
  - Encompass Heterogeneity – Components
  - Cope with Complexity – Constructivity
  - Cope with Uncertainty – Adaptivity
- Embedded Systems Design
- Discussion

OVERVIEW

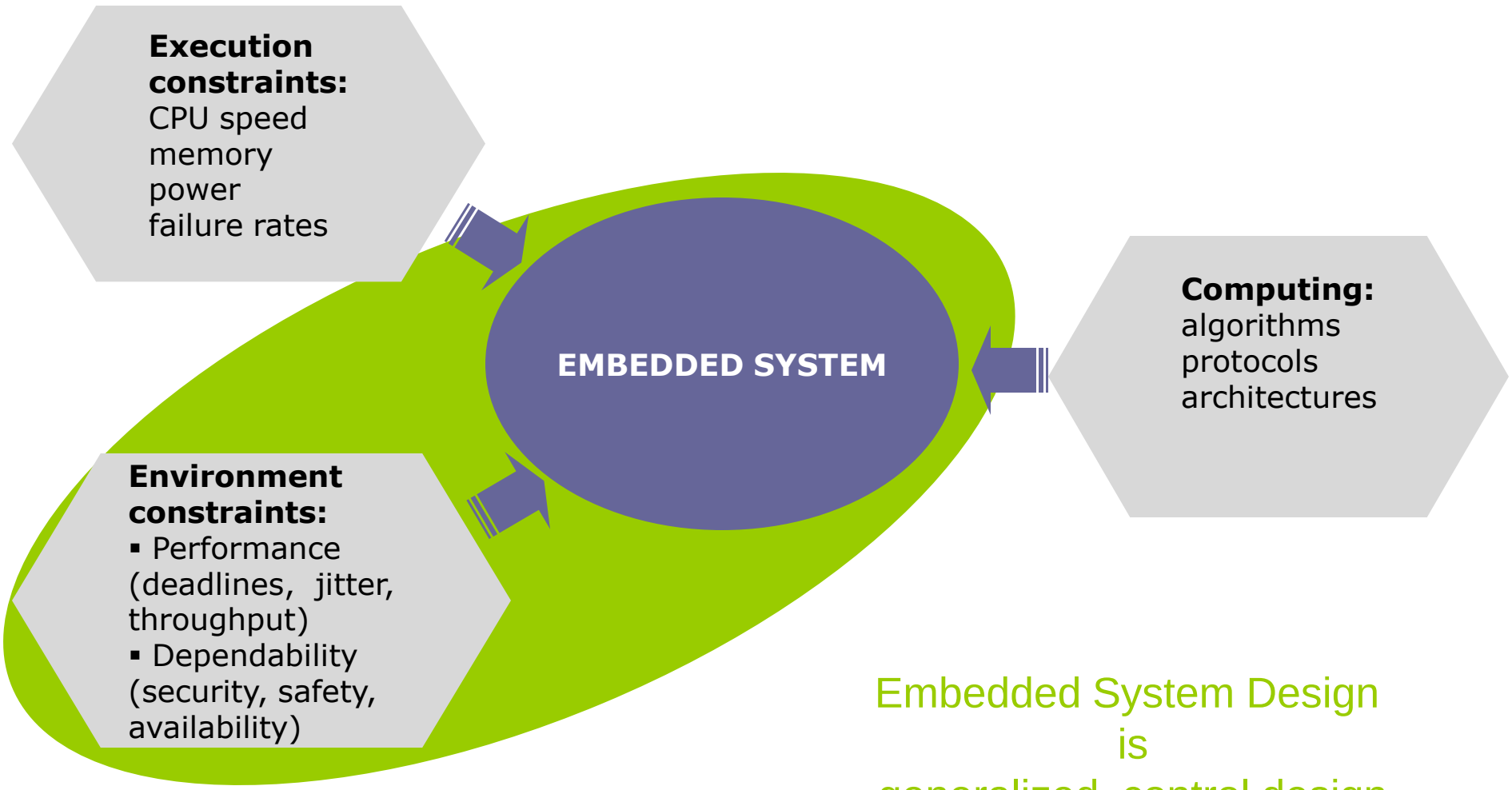
Embedded Systems Design – Grand Challenge



Embedded Systems Design – Grand Challenge



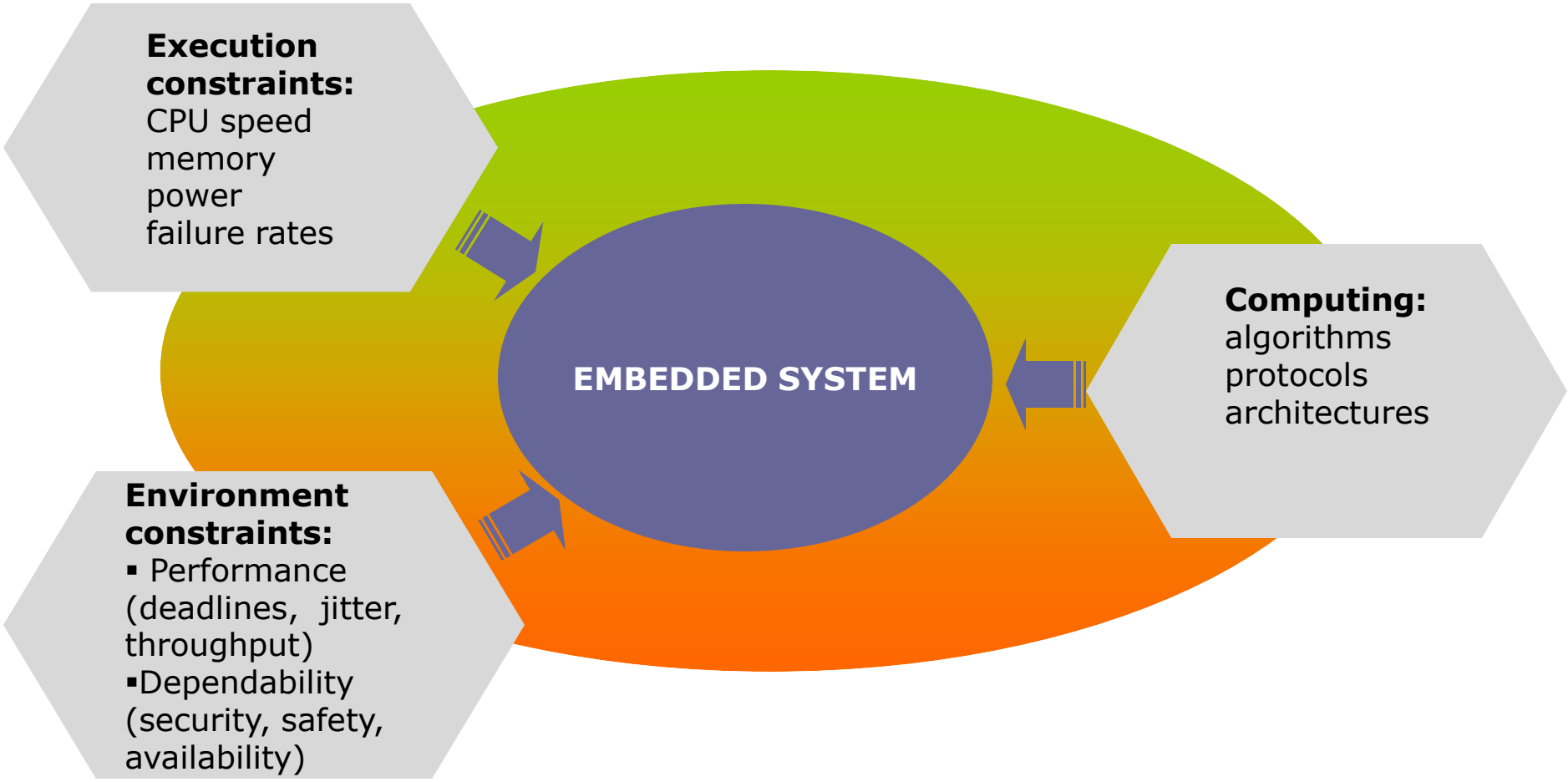
Embedded Systems Design – Grand Challenge



Embedded System Design
is
generalized control design

Embedded Systems Design – Grand Challenge

Embedded System Design coherently integrates all these



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

OVERVIEW

- System Design Today
- Research Challenges
  - Marry Physicality and Computation
  - Encompass Heterogeneity – Components
  - Cope with Complexity – Constructivity
  - Cope with Uncertainty – Adaptivity
- Embedded Systems Design
- Discussion

Marry Physicality and Computation

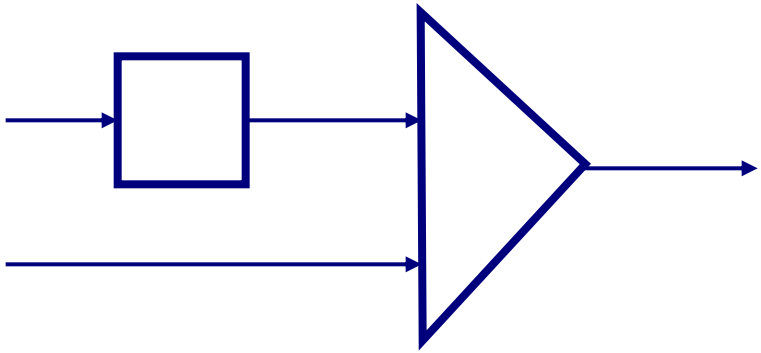
Physics	Computer Science
Studies the laws governing energy, matter and their relationships	Studies foundations of information and computation
Studies a given « reality »	Studies created universes
Physical systems – Analytic models	Computing systems – Machines
Continuous mathematics	Discrete mathematics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

Marry Physicality and Computation

Physical Systems Engineering

Analytic Models

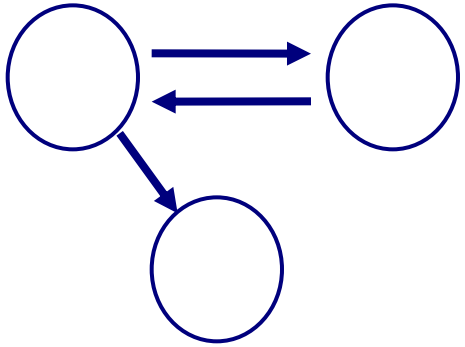
Component: transfer function
Composition: parallel
Connection: data flow



Computing Systems Engineering

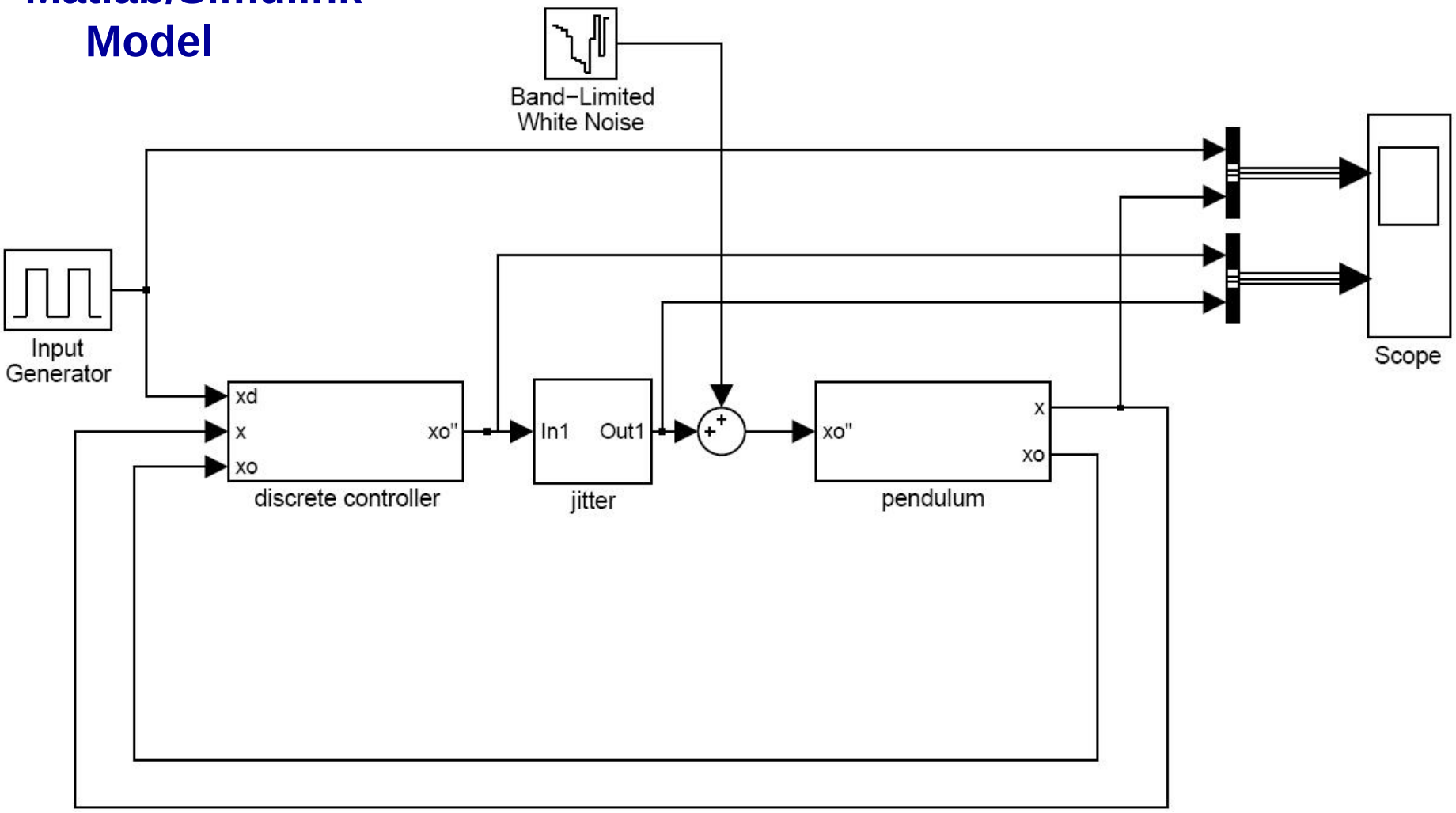
Computational Models

Component: subroutine
Composition: sequential
Connection: control flow



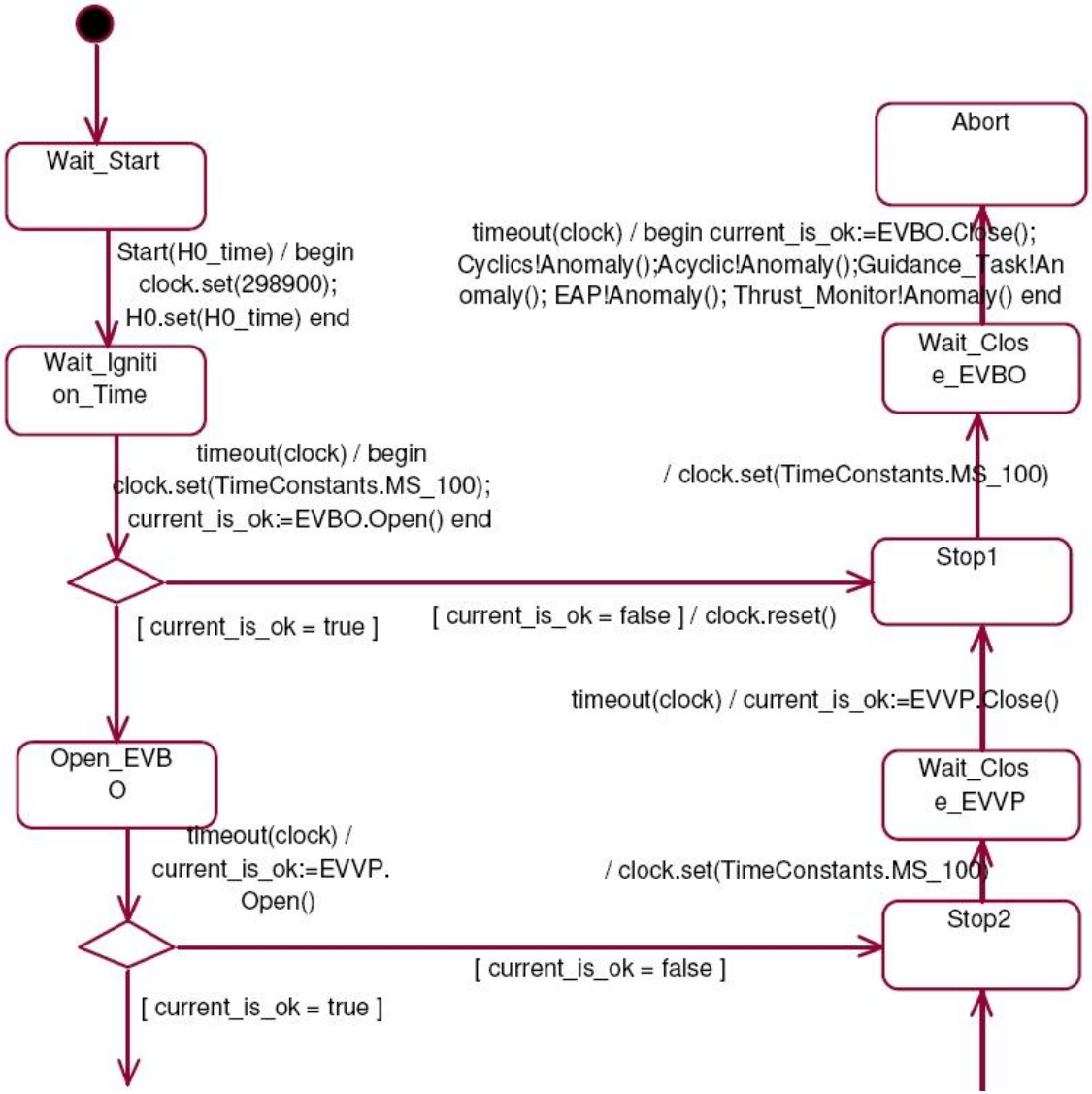
Marry Physicality and Computation

Matlab/Simulink Model



Marry Physicality and Computation

UML Model
(Rational Rose)



OVERVIEW

- System Design Today
- Research Challenges
  - Marry Physicality and Computation
  - Encompass Heterogeneity – Components
  - Cope with Complexity – Constructivity
  - Cope with Uncertainty – Adaptivity
- Embedded Systems Design
- Discussion

Encompass Heterogeneity - Components

Build complex systems by composing components (simpler systems).
This confers numerous advantages such as productivity and correctness

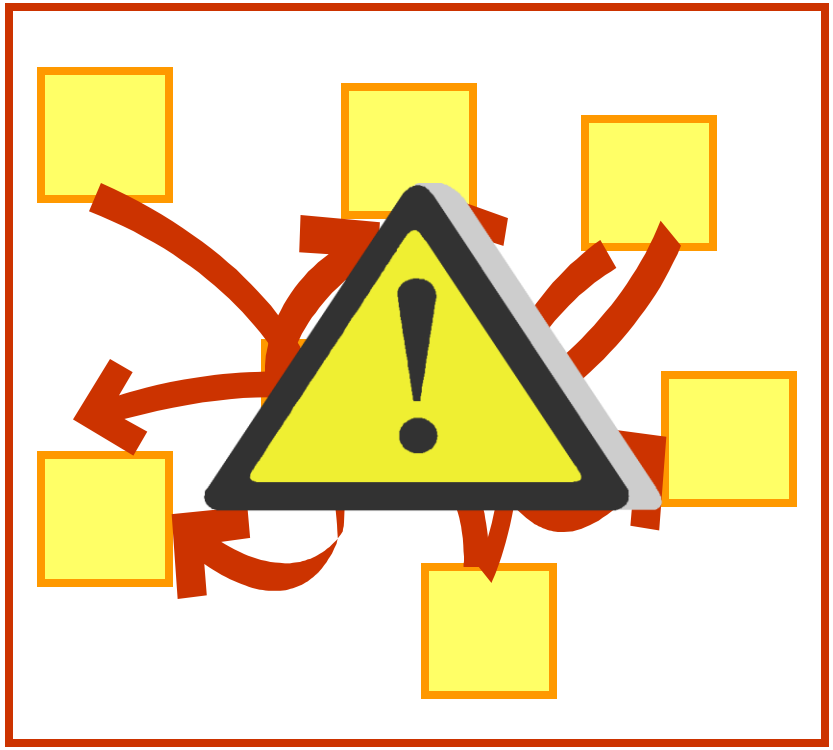
- SW Component frameworks:
  - Coordination languages extensions of programming languages e.g. BPEL, Javaspaces, TSpaces, Concurrent Fortran, NesC
  - Middleware e.g. Corba, Javabeans, .NET
  - Software development environments: PCTE, SWbus, Softbench, Eclipse
- System modeling languages: Statecharts, SysML, Matlab/Simulink, AADL, Ptolemy
- Hardware description languages: Verilog, VHDL, SystemC

Heterogeneity: Embedded systems are built from components with different characteristics

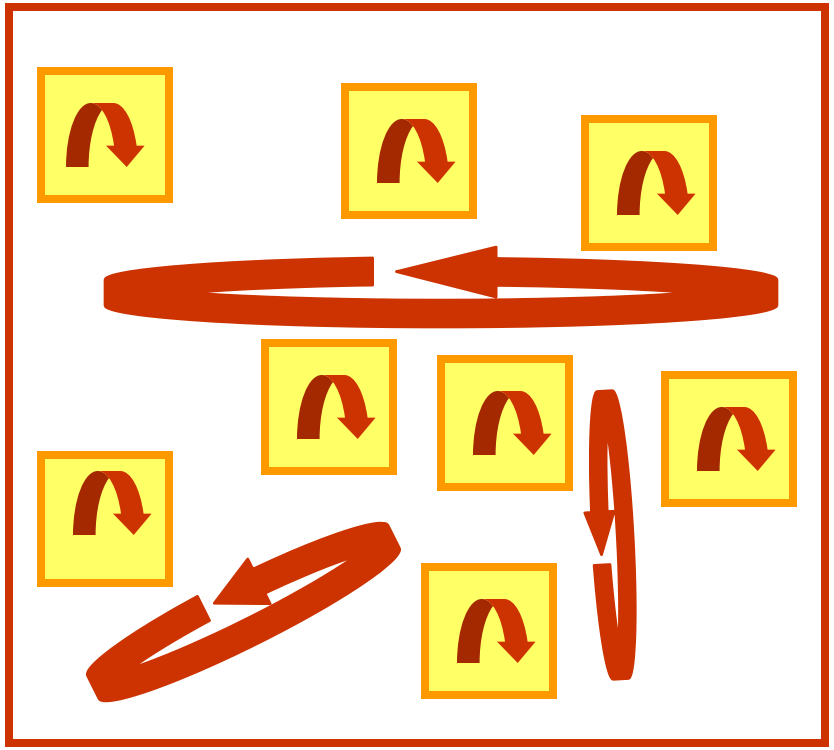
- **Execution**: synchronous and asynchronous components
- **Interaction**: function call, broadcast, rendezvous, monitors
- **Abstraction levels**: hardware, execution platform, application software

Encompass Heterogeneity - Components

Thread-based programming



Actor-based programming



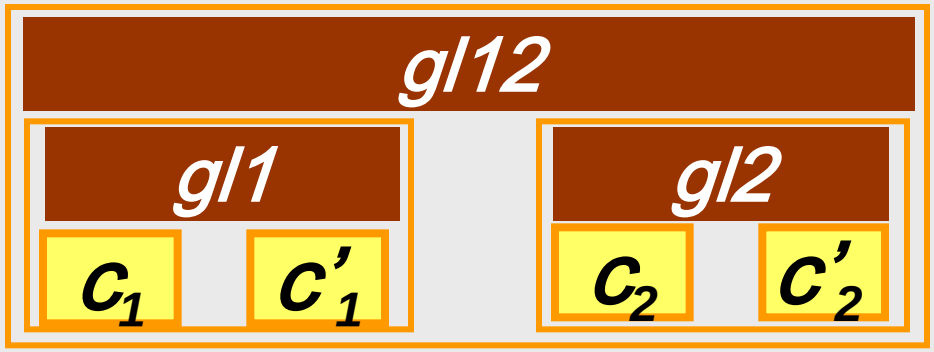
Software Engineering

Systems Engineering

Encompass Heterogeneity - Components

Build a component *C* satisfying given requirements *f, 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



satisfies f

- Move from single low-level composition operators e.g. automata-based to families of high-level composition operators e.g. protocols, controllers

- We need a unified composition paradigm for describing and analyzing the coordination between components to formulate system designs in terms of tangible, well-founded and organized concepts

- System Design Today
- Research Challenges
  - Marry Physicality and Computation
  - Encompass Heterogeneity – Components
  - Cope with Complexity – Constructivity
  - Cope with Uncertainty – Adaptivity
- Embedded Systems Design
- Discussion

OVERVIEW

Cope with Complexity – Constructivity

- Today, *a posteriori* verification at high development costs limited to medium complexity systems
- Tomorrow, correct-by-construction results should advantageously take into account architectures and their features.

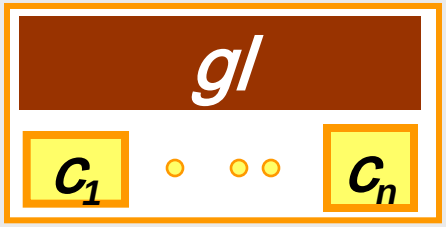
There is a large space to be explored, between full constructivity and *a posteriori* verification. Develop correct-by-construction results

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

Constructivity – Compositionality

Build correct systems from correct components: rules for proving global properties from properties of individual components

c_i sat P_i implies $\forall gl\ \exists \widetilde{gl}$

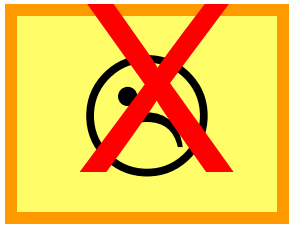


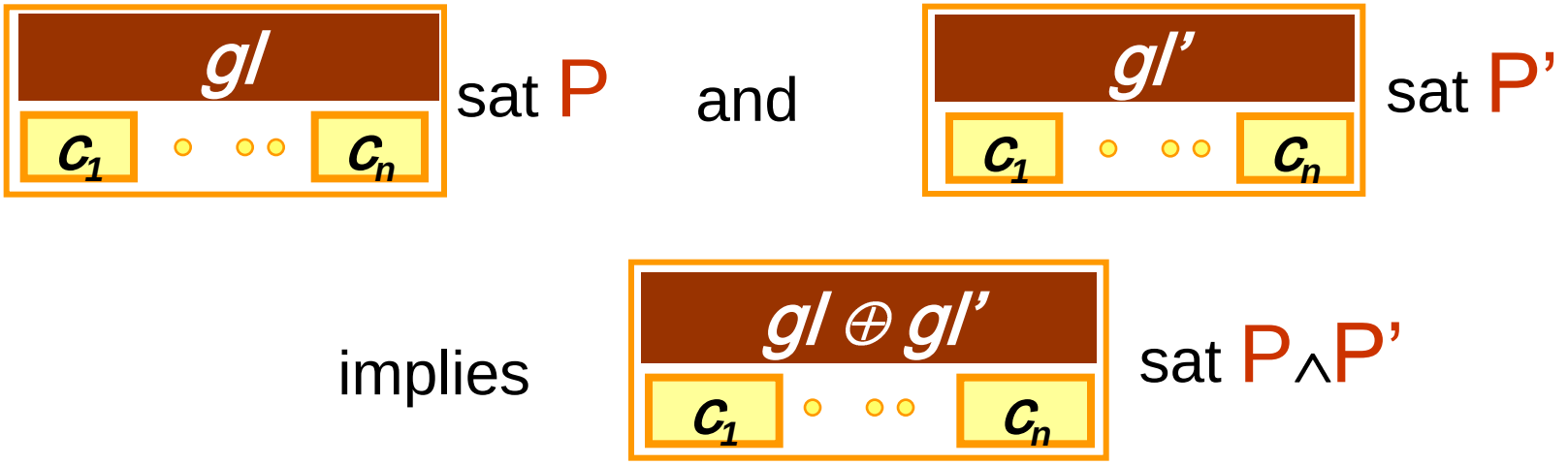
sat $\widetilde{gl}(P_1, .., P_n)$

We need compositionality results for the preservation of progress properties such as deadlock-freedom and liveness as well as extra-functional properties

Constructivity – Composability

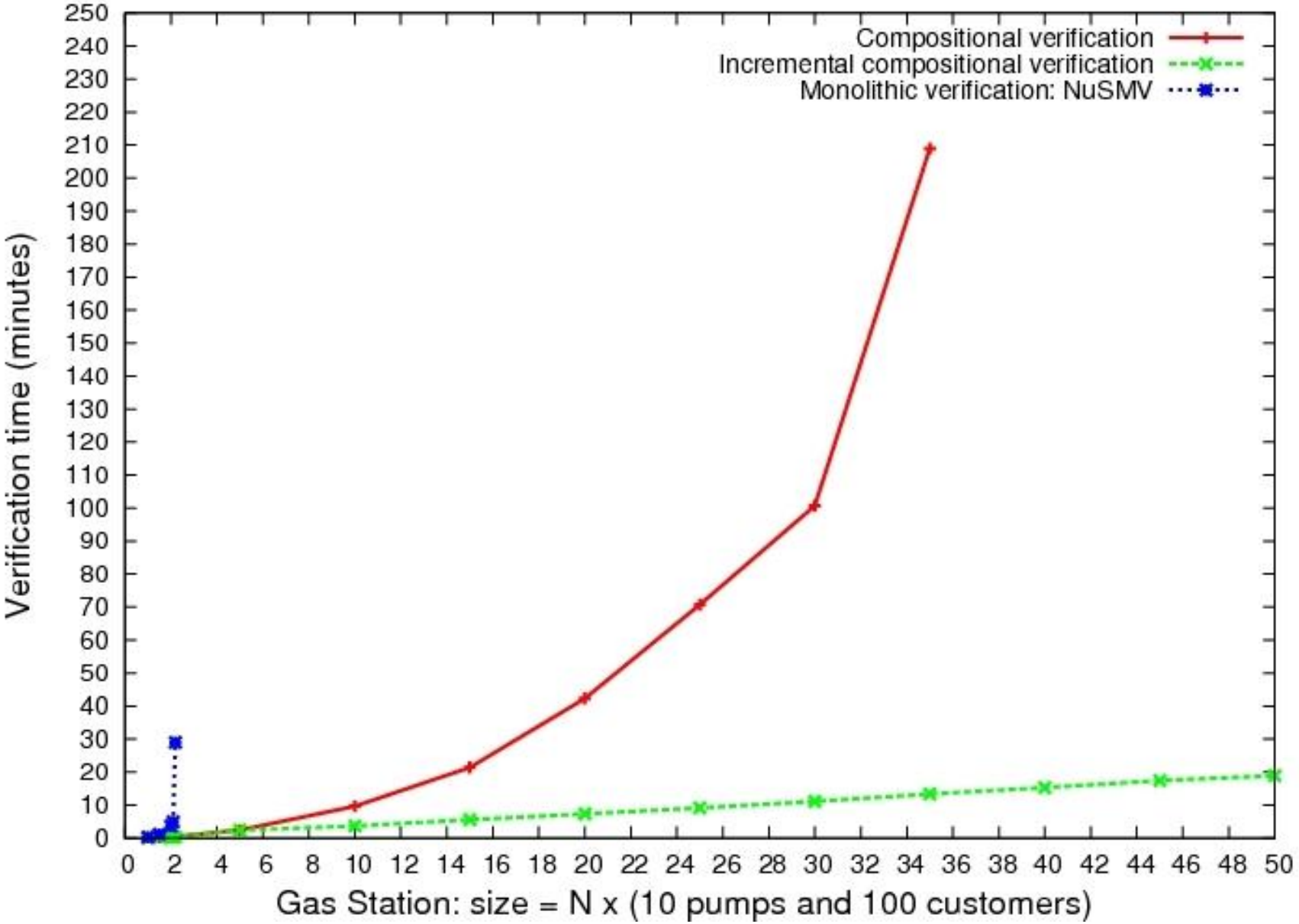
Essential properties of components are preserved when they are integrated





Property stability phenomena are poorly understood.
We need composability results e.g. non interaction of features in middleware, composability of scheduling algorithms, of Web services, of aspects

Constructivity – Checking for Deadlock-freedom



OVERVIEW

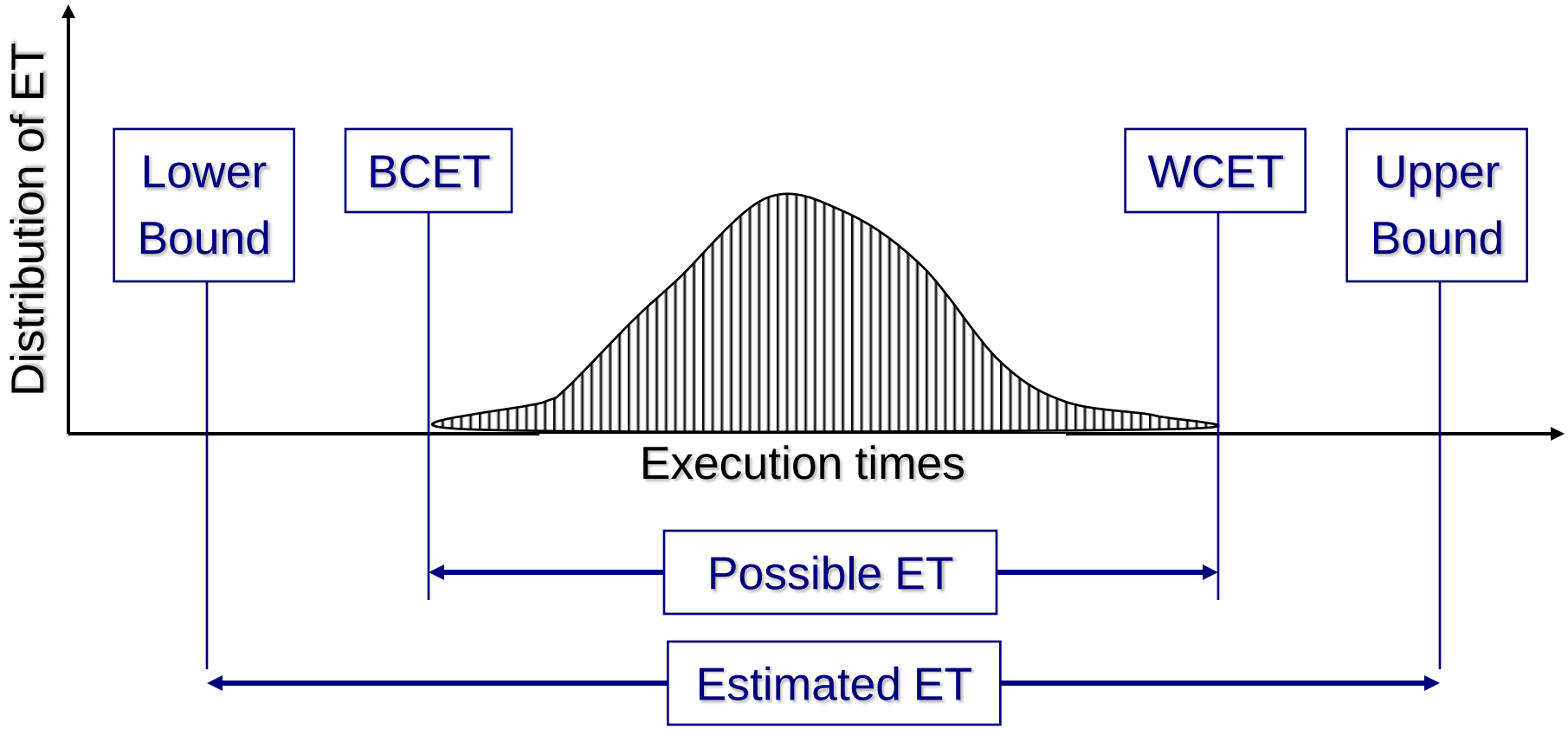
- System Design Today
- Research Challenges
  - Marry Physicality and Computation
  - Encompass Heterogeneity – Components
  - Cope with Complexity – Constructivity
  - Cope with Uncertainty – Adaptivity
- Embedded Systems Design
- Discussion

Cope with Uncertainty – Predictability

- Systems must ensure predictable behavior in interaction with uncertain environments
- Uncertainty is characterized as the difference between
  - average or nominal behavior
  - worst-case or faulty behavior
- The trend is towards drastically increasing uncertainty, due to:
  - Interaction with complex, non-deterministic, possibly hostile external environments
  - Execution platforms with sophisticated HW/SW architectures (layering, caches, speculative execution, …)

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

Cope with Uncertainty – Predictability

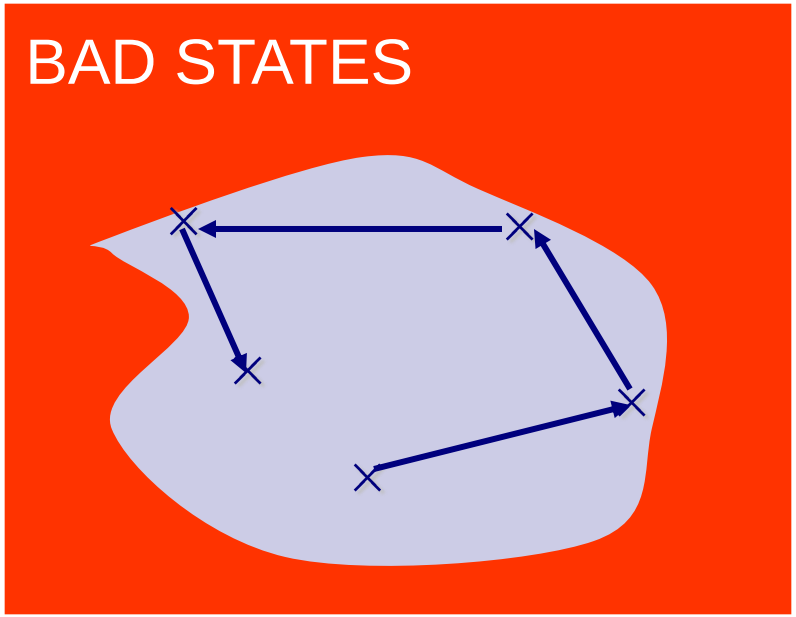


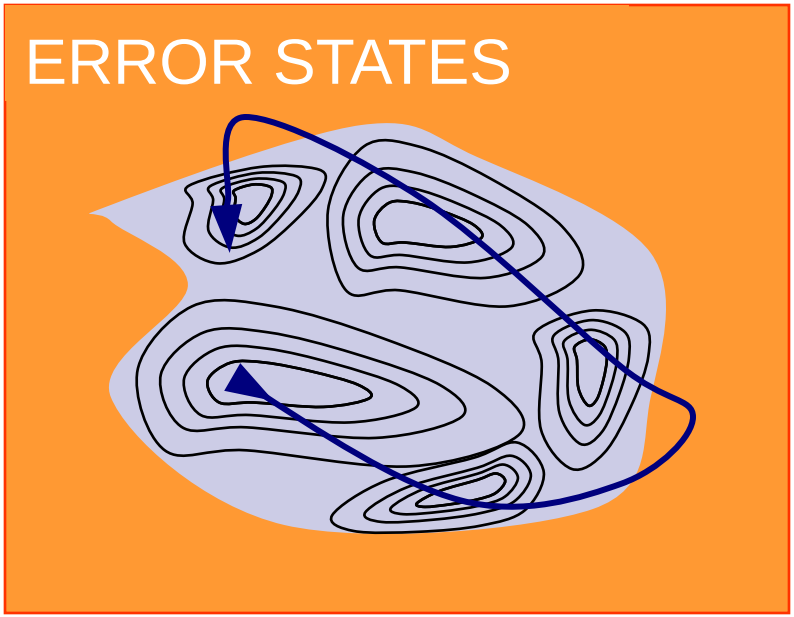
For simple operations WCET may be 300 × BCET

Acknowledgement: Reinhard Wilhelm "Timing Analysis and Predictability" July 2008

Cope with Uncertainty – Predictability

Increasing uncertainty gives rise to 2 diverging design paradigms





Critical systems engineering
based on worst-case analysis and static resource reservation
e.g. hard real-time approaches, massive redundancy

Best effort engineering
based on average case analysis
e.g. soft real-time for optimization of speed, memory, bandwidth, power

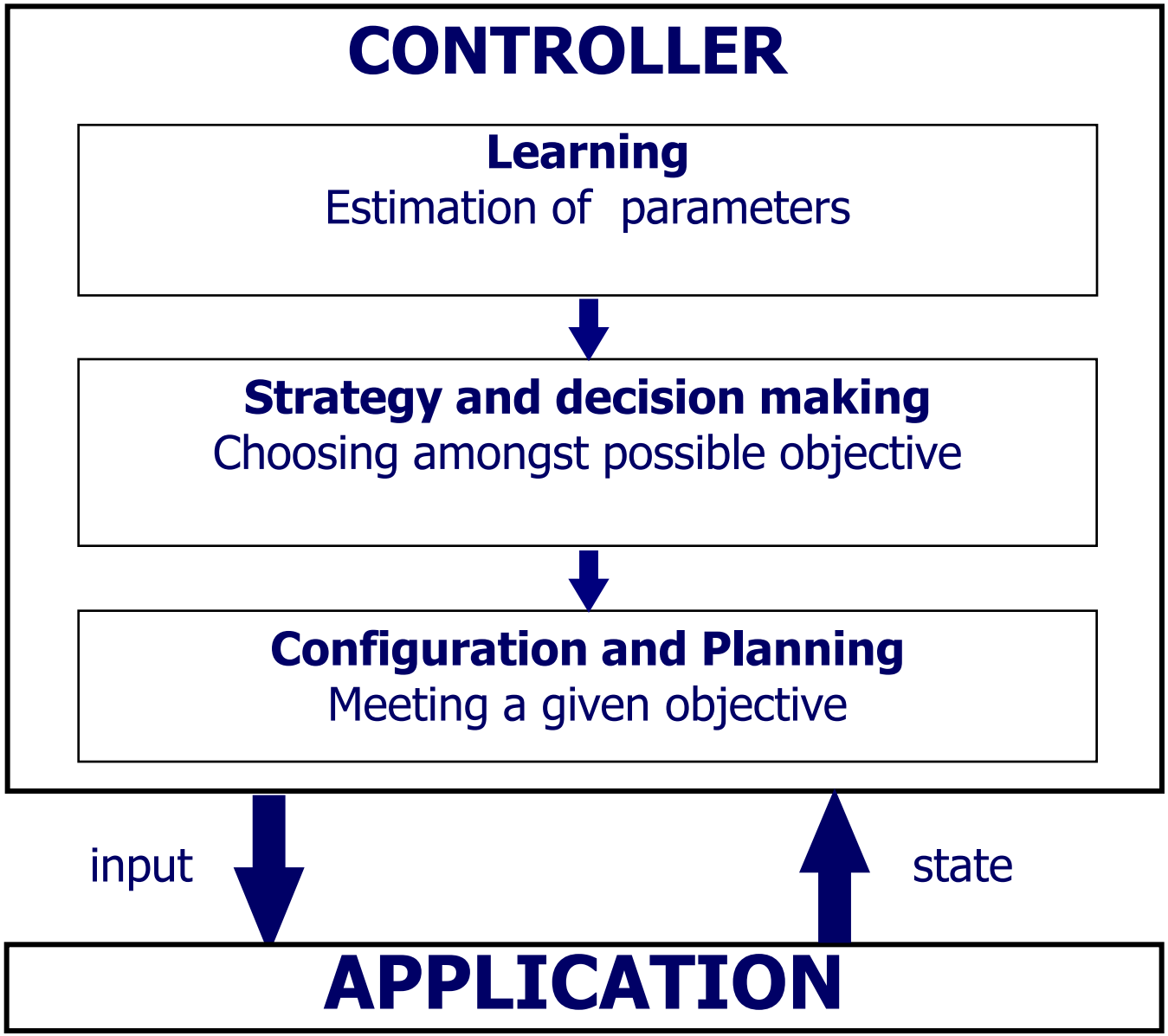
Cope with Uncertainty – Predictability

The separation between critical and best effort engineering implies increasing costs and reduced hardware reliability, e.g. increasing number of ECUs in cars.

We are moving from federated to integrated architectures (both critical and non critical functions on one chip) while striving for predictability by

- Reducing intrinsic and estimated uncertainty through
  - Simplification of architectures, predictable cache replacement policies
  - Determinization of the observable behavior e.g. time triggered systems
- Developing adaptive control techniques combining the two paradigms:
  - Satisfaction of critical properties
  - Efficiency by optimal use of the globally available resources (processor, memory, power).

Cope with Uncertainty – Adaptivity



Cope with Uncertainty: Adaptivity

Learning

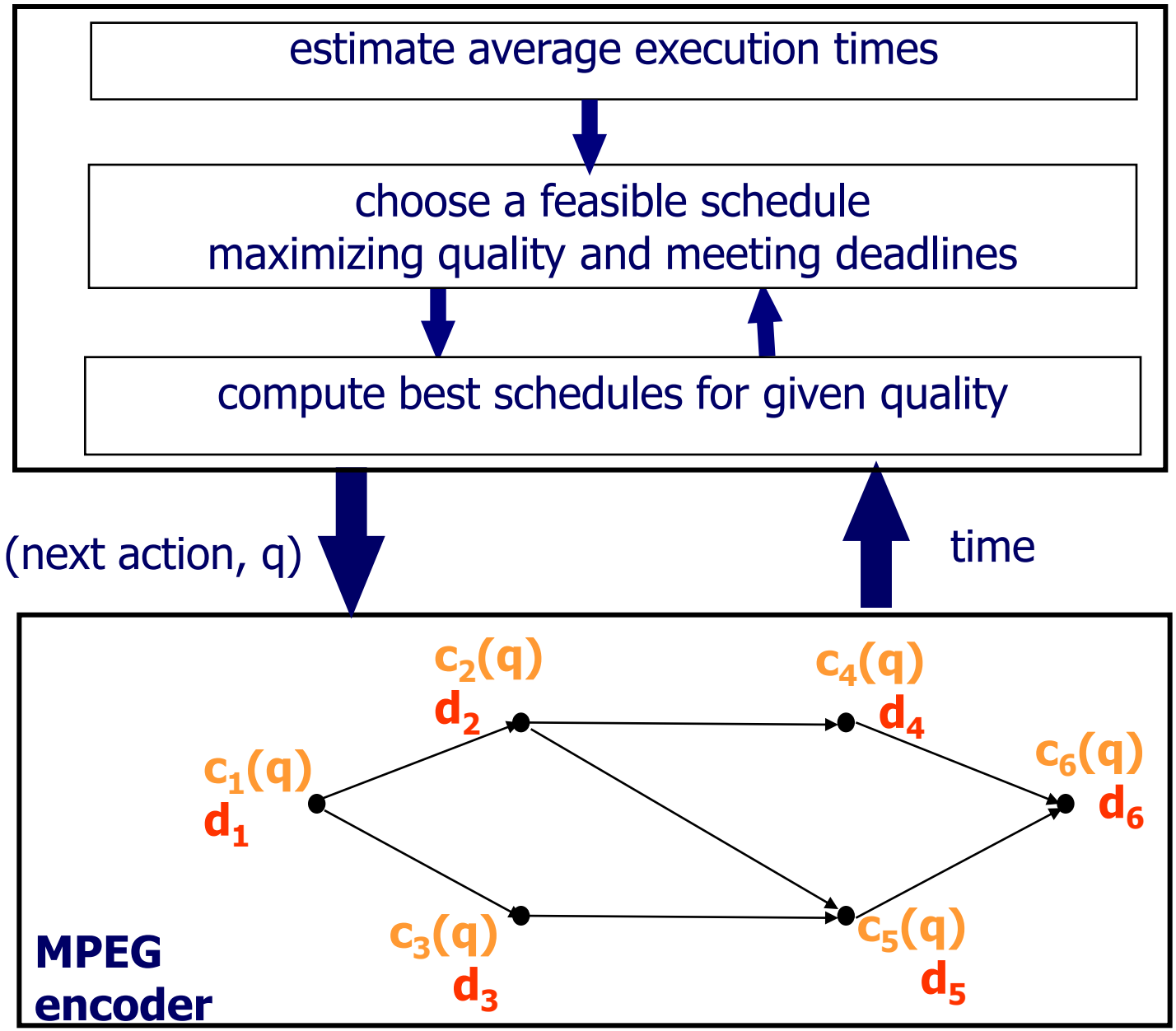
Movie would have been better …

Managing Conflicting Objectives

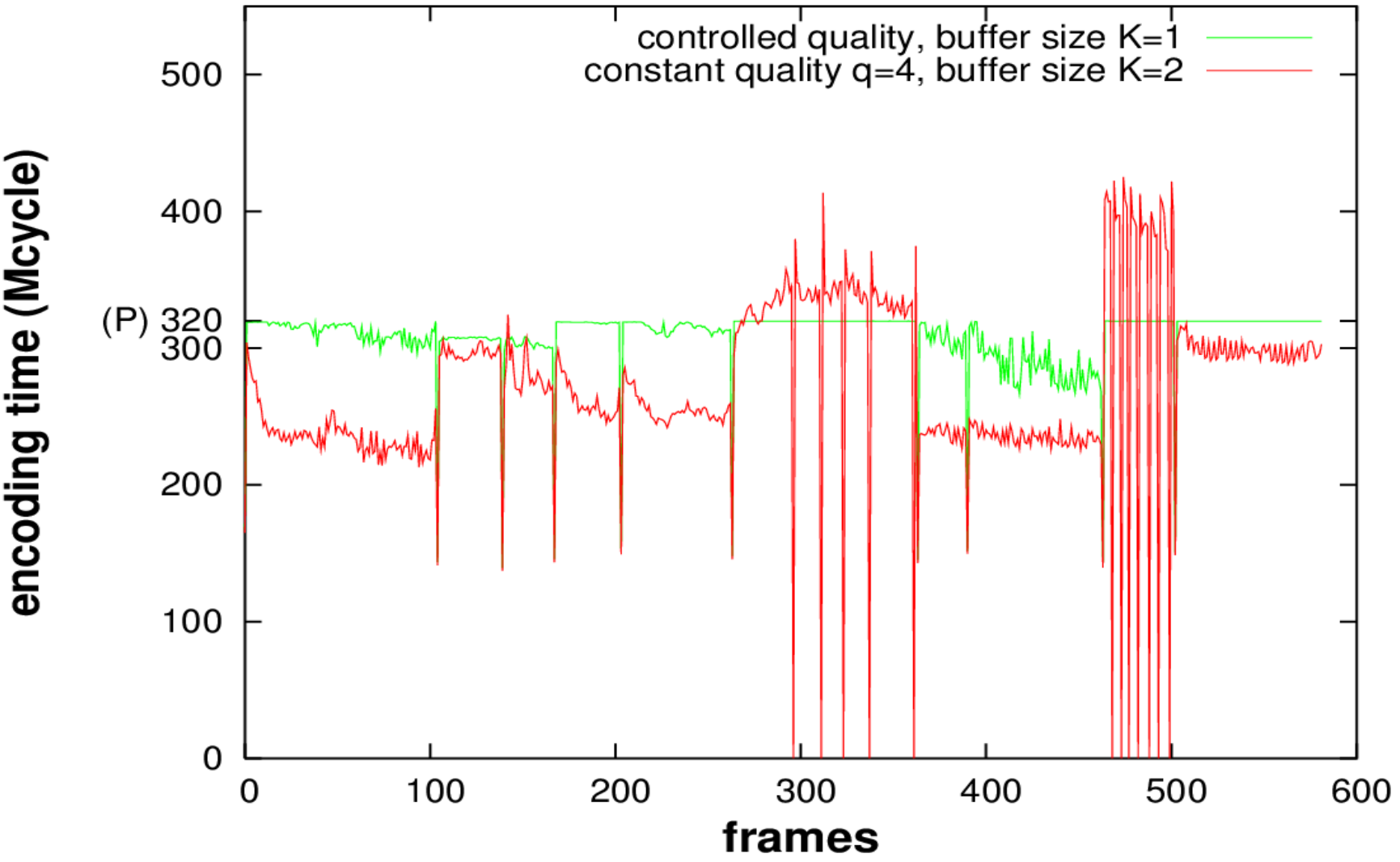
Go to: 1) Stadium 2) Movie 3) Restaurant

Planning

Cope with Uncertainty – Adaptivity



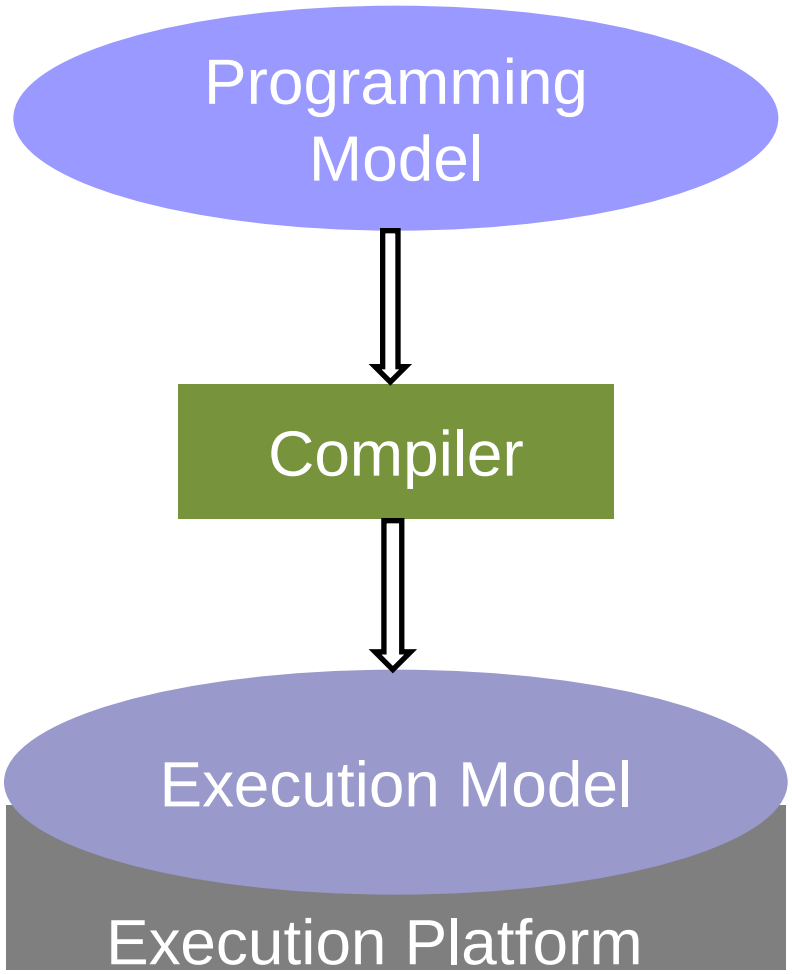
Cope with Uncertainty - Adaptivity



OVERVIEW

- System Design Today
- Research Challenges
  - Marry Physicality and Computation
  - Encompass Heterogeneity – Components
  - Cope with Complexity – Constructivity
  - Cope with Uncertainty – Adaptivity
- Embedded Systems Design
- Discussion

Model-based Design - Principle

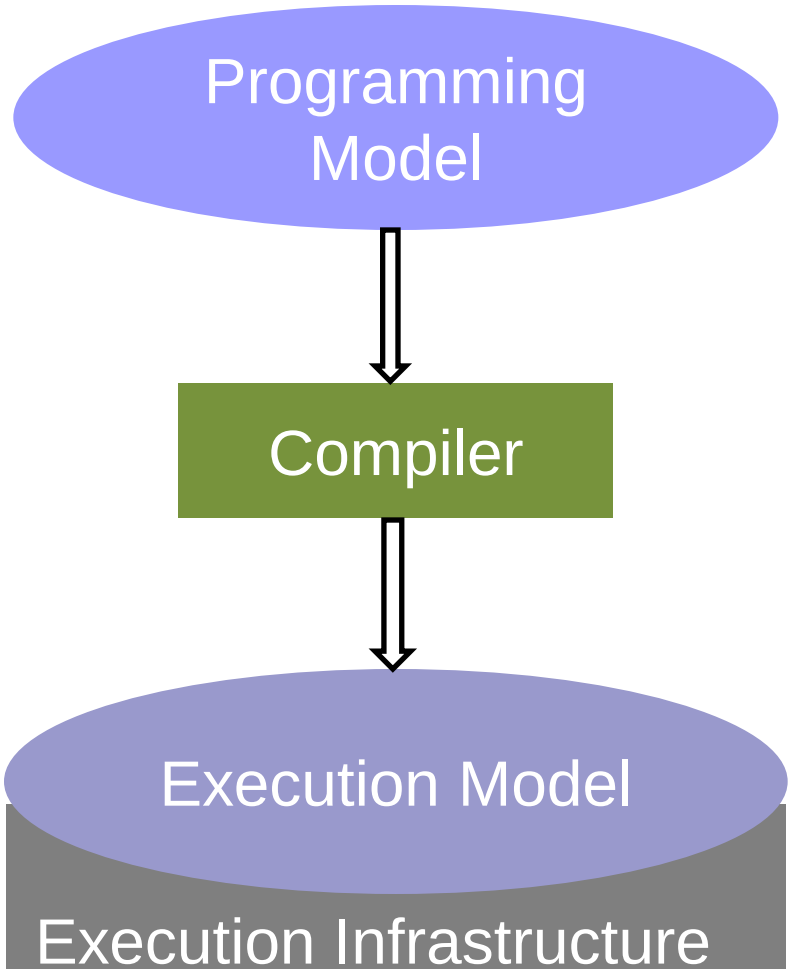


Extension of an existing language with concepts and primitives for concurrency and resource management

Implements a mapping from the PM to the EM - correctness-by - construction

Abstract machine encompassing mechanisms for efficient and dependable execution

Model-based Design – Synchronous Computation

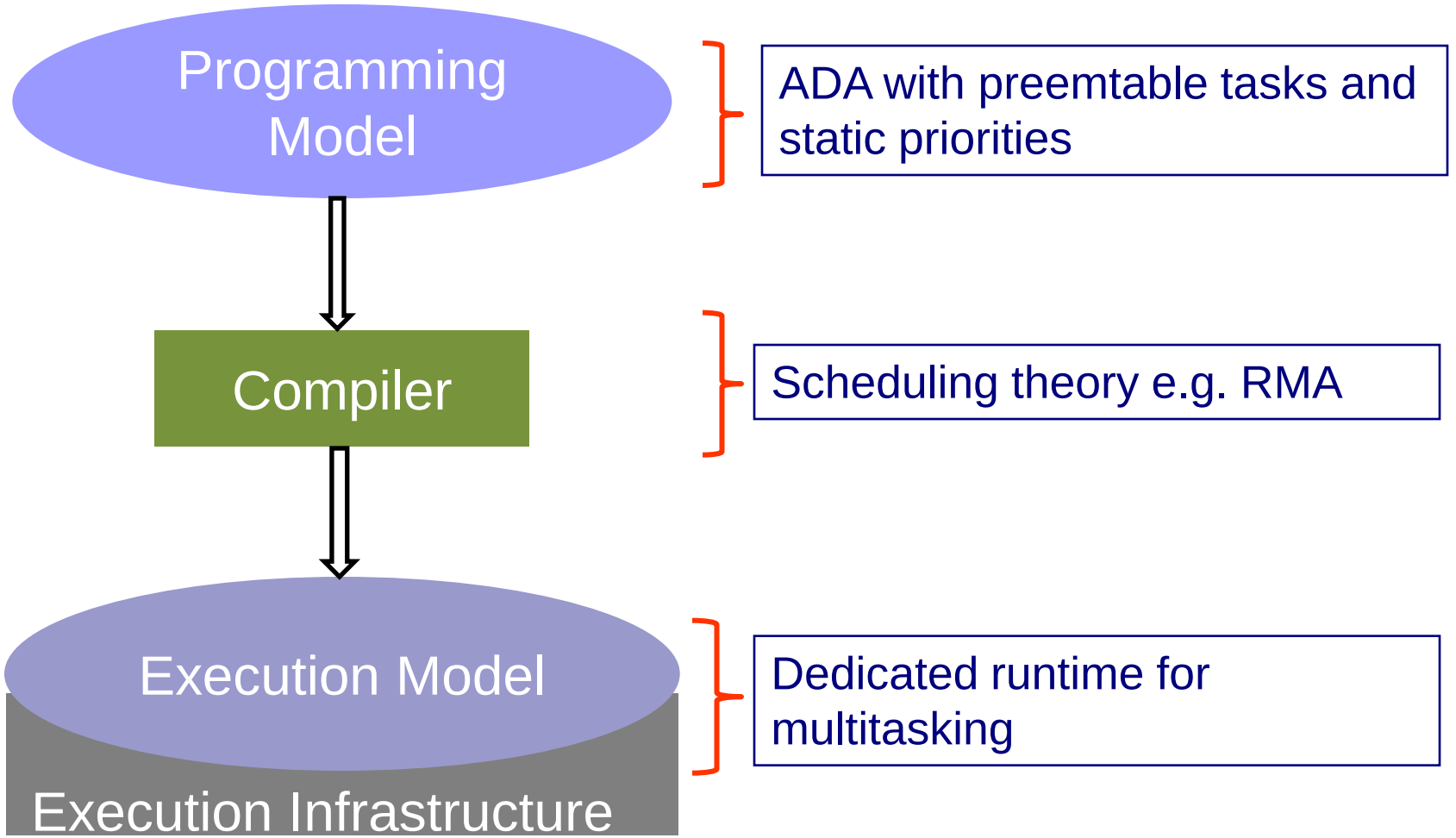


Synchronous programming languages such as Lustre, Matlab/Simulink

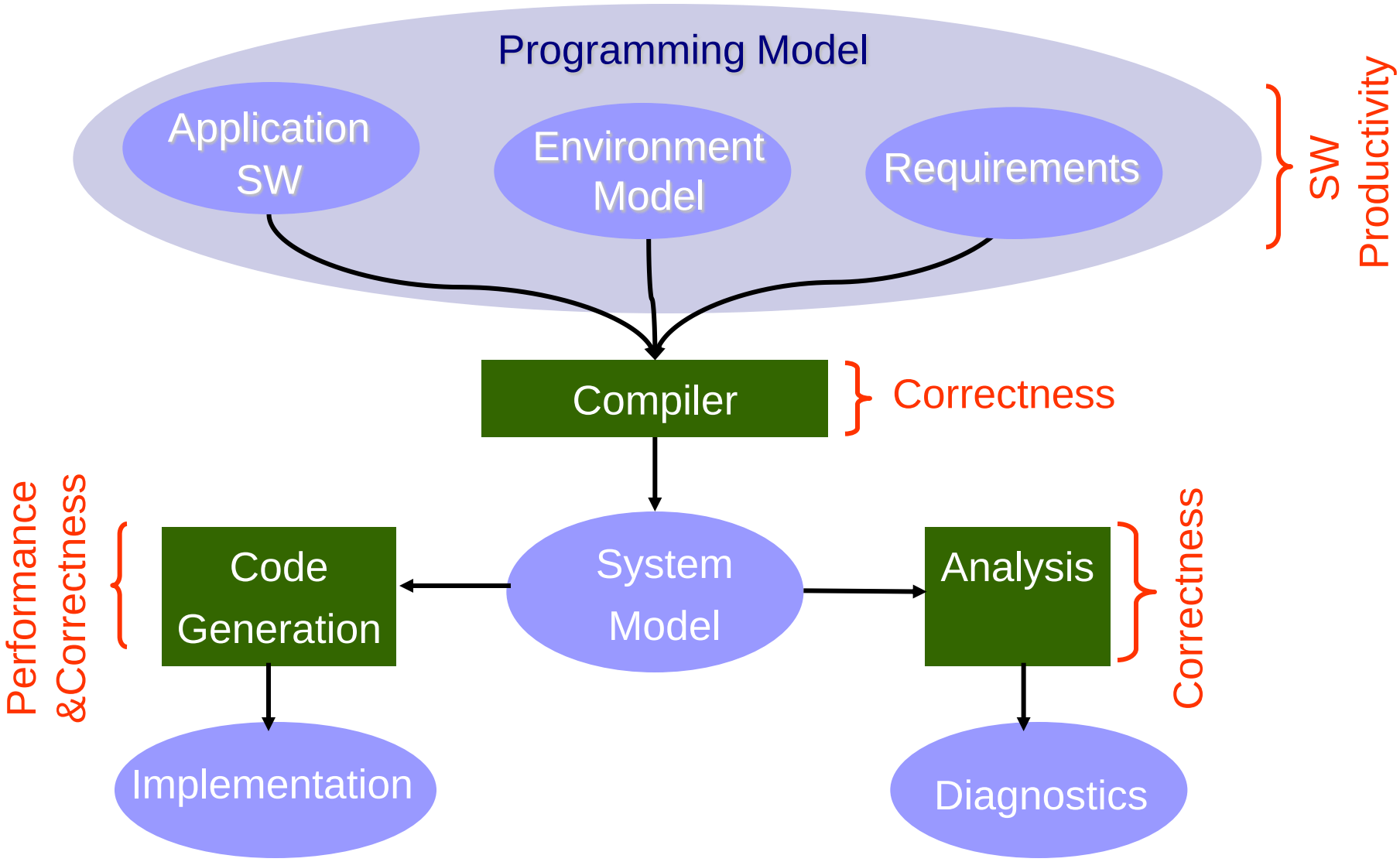
Theory for checking correctness of the code generation process e.g. confluence and deadlock-freedom

Monolithic execution model e.g. single task

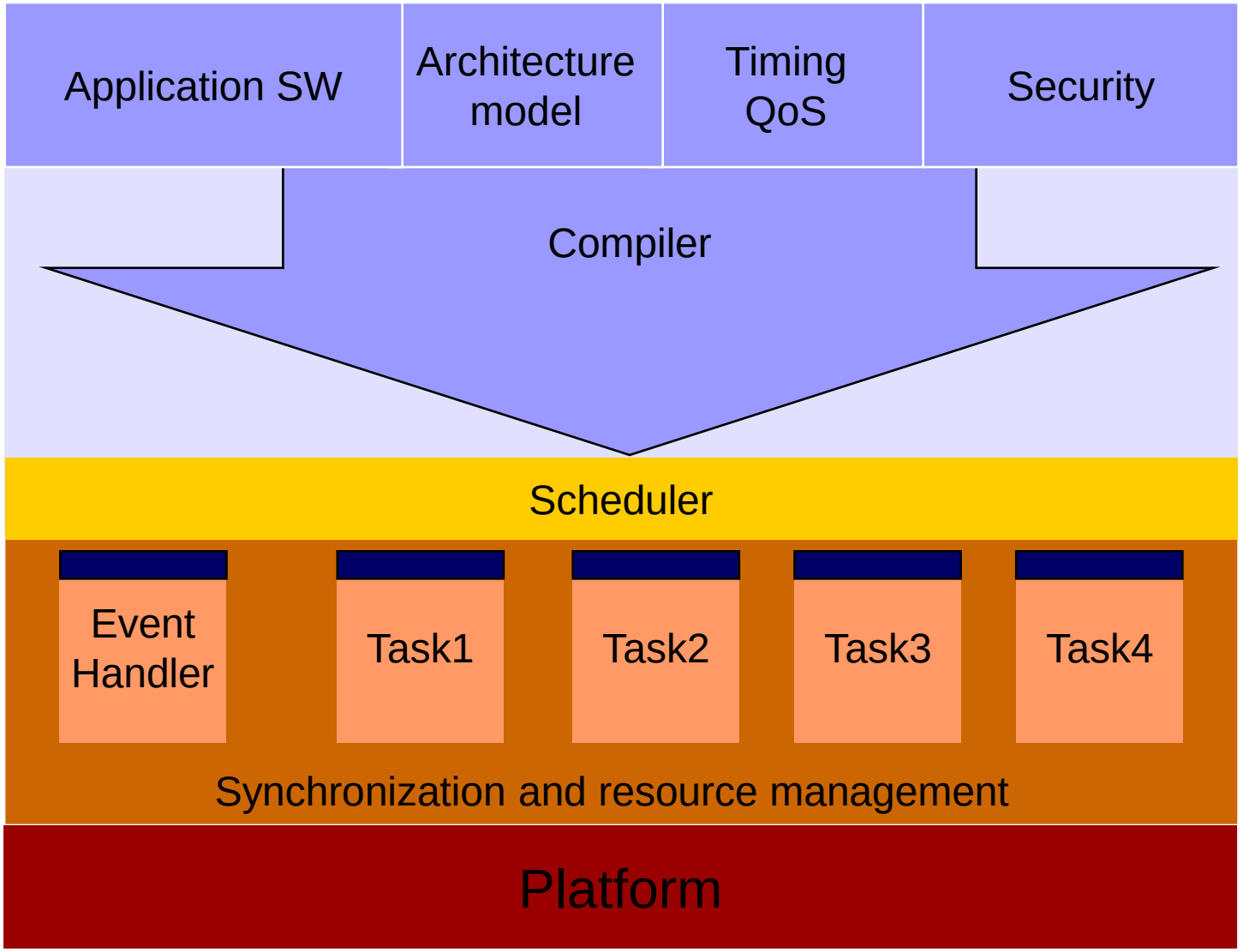
Model-based Design – Event-driven Computation



Model-based Design – A more detailed View



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, computing)
- Modeling and Verification
- Distributed and fault-tolerant implementations (influence communication delays, clock drift, aperiodic sampling)

↳ Use of control-based techniques for adaptivity

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 :

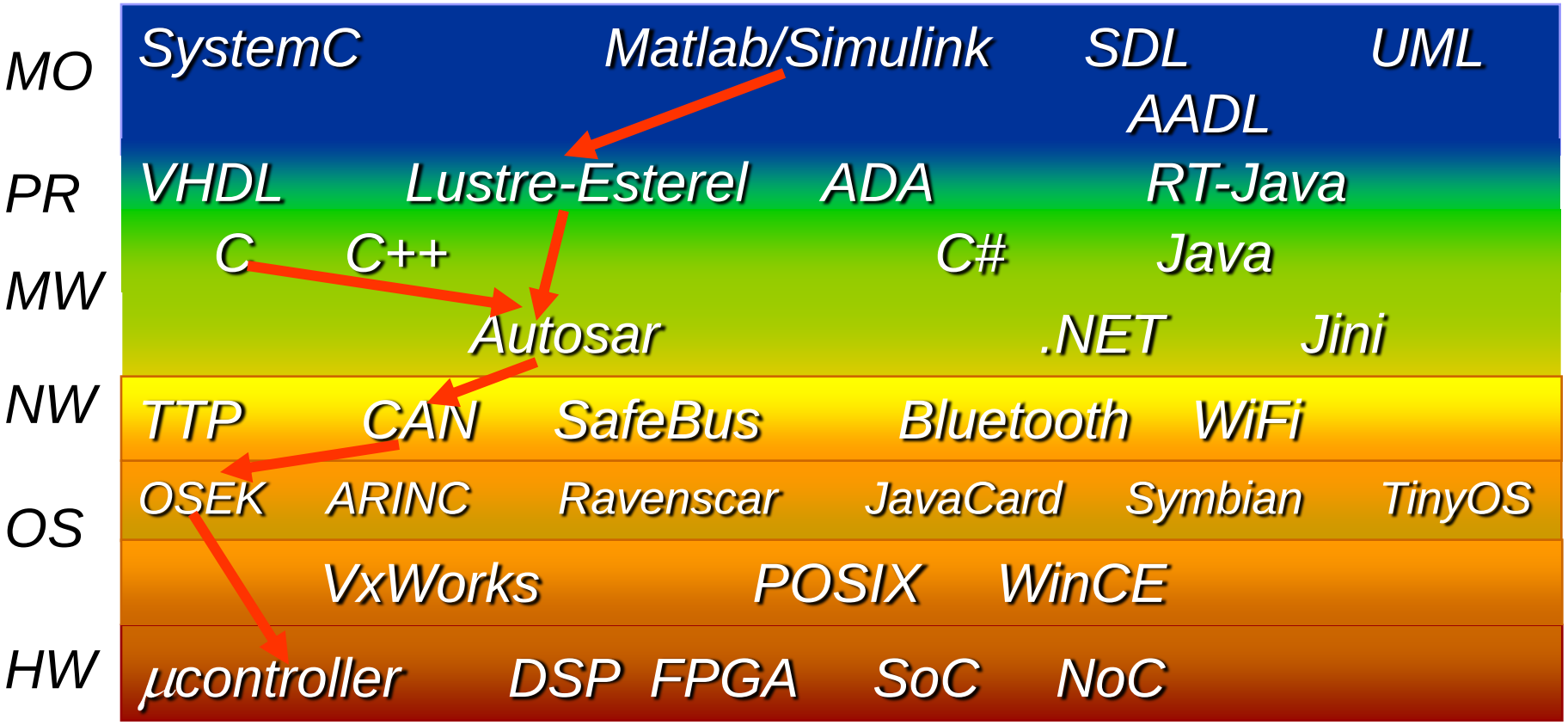
- Methodologies for domain-specific standards, such as :
  - DO-178B Process Control Software Safety Certification
  - Integrated Modular Avionics; Autosar
  - Common Criteria for Information Technology Security Evaluation
- Certification methods and tools
- Architectures, protocols and algorithms for fault-tolerance and security taking into account QoS requirements (real-time, availability)

Networked Embedded Systems

Adaptive distributed real-time systems, inherently dynamic, must adapt to accommodate workload changes and to counter uncertainties in the system and its environment

- Clock synchronization, parameter settings
- Specific routing algorithms
- Location discovery, neighbor discovery
- Group management (dormant, active-role assignment)
- Self-organization : backbone creation, leader election, collaboration to provide a service
- Power management : turn-off of dormant nodes, periodical rotation of active nodes to balance energy

Integration of Methods and Tools



OVERVIEW

- System Design Today
- Research Challenges
  - Marry Physicality and Computation
  - Encompass Heterogeneity – Components
  - Cope with Complexity – Constructivity
  - Cope with Uncertainty – Adaptivity
- Embedded Systems Design
- Discussion

Discussion

Embedded Systems

- break with traditional Systems Engineering. They need new design techniques guaranteeing both functionality and quality (performance and dependability) and taking into account market constraints
- are an opportunity for reinvigorating and extending Computer Science with new paradigms from Electrical Engineering and Control Theory. This requires basic research effort for meeting four challenges
  - Combining analytic and computational models
  - Component-based construction of heterogeneous systems
  - Constructivity at design time
  - Adaptivity as a means for ensuring predictability

In addition to meeting the research challenges, the development of System Design as a Discipline requires formalization of the design process as a sequence of correct-by-construction component-based model transformations

THANK YOU