The Embedded Systems Design Challenge

Tom Henzinger
EPFL

Joseph Sifakis
Verimag

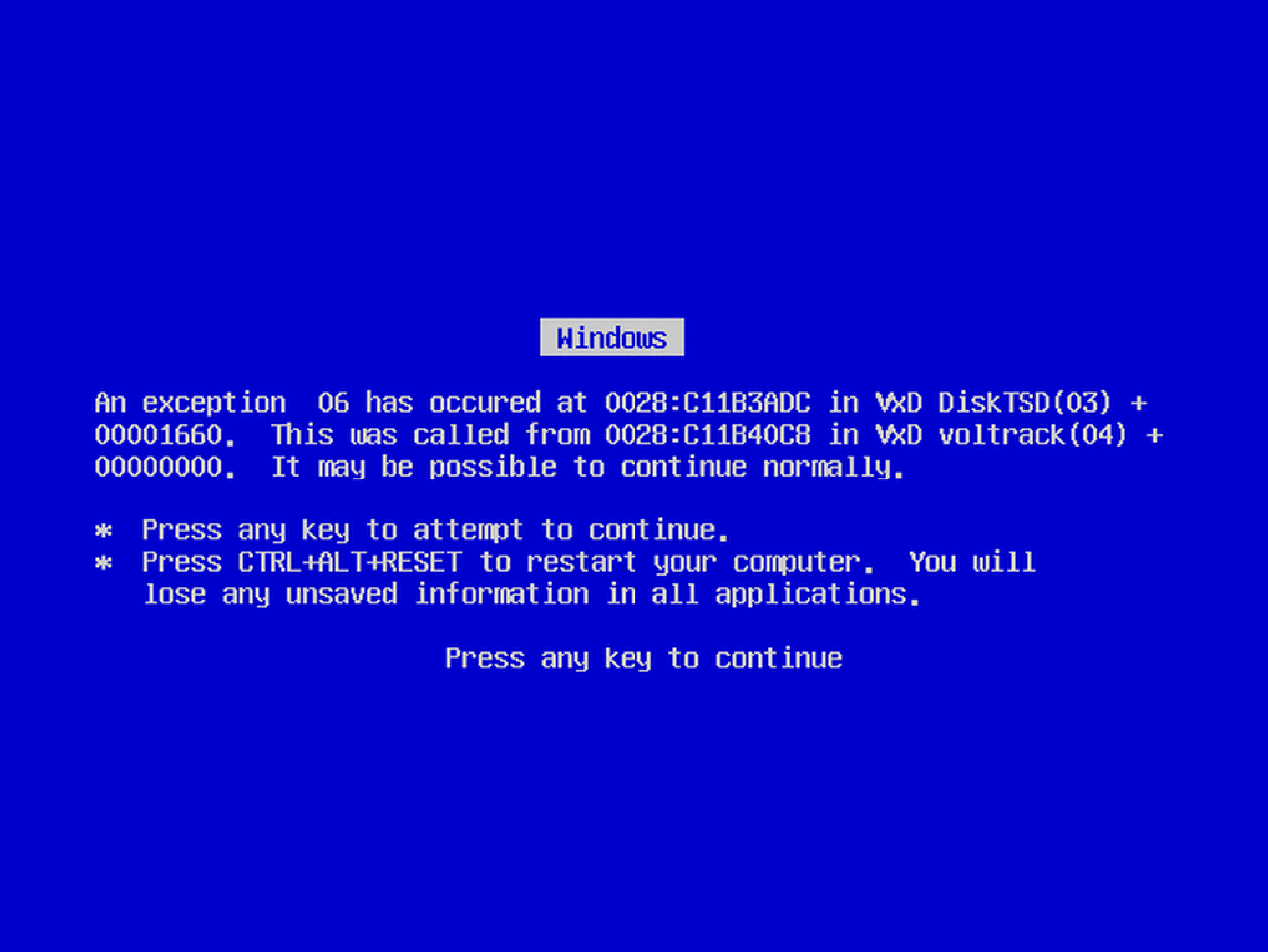
Formal Methods:
A Tale of Two Cultures

Engineering

Differential Equations
Linear Algebra
Probability Theory

Computer Science

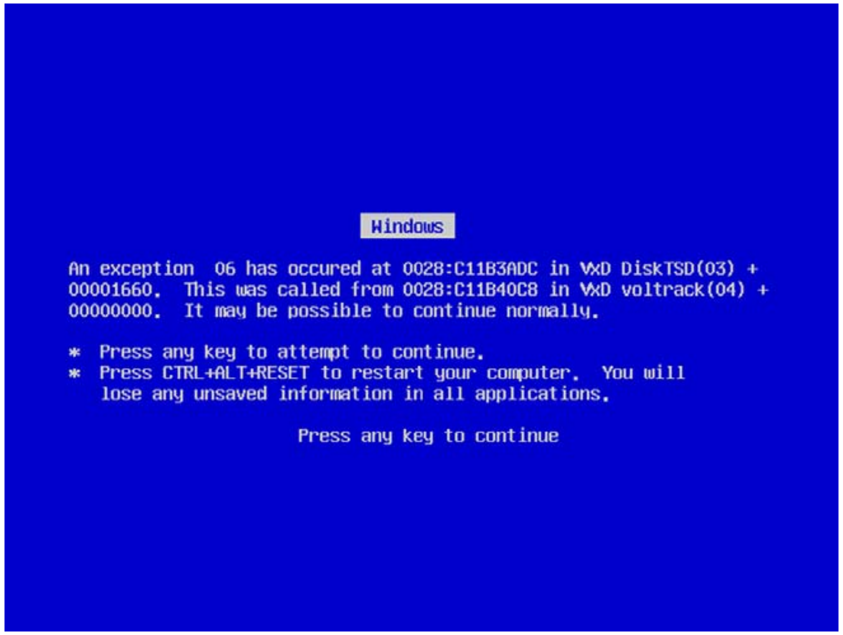
Logic
Discrete Structures
Automata Theory



```
                              Windows

An exception  06 has occured at 0028:C11B3ADC in VxD DiskTSD(03) +
00001660.  This was called from 0028:C11B40C8 in VxD voltrack(04) +
00000000.  It may be possible to continue normally.

*  Press any key to attempt to continue.
*  Press CTRL+ALT+RESET to restart your computer.  You will
   lose any unsaved information in all applications.

                    Press any key to continue
```

So how are we doing?

Uptime: 123 years



```
                              Windows

An exception  06 has occured at 0028:C11B3ADC in VxD DiskTSD(03) +
00001660.  This was called from 0028:C11B40C8 in VxD voltrack(04) +
00000000.  It may be possible to continue normally.

*  Press any key to attempt to continue.
*  Press CTRL+ALT+RESET to restart your computer.  You will
   lose any unsaved information in all applications.

                      Press any key to continue
```

What went wrong?

Engineering	Computer Science
Differential Equations	Logic
Linear Algebra	Discrete Structures
Probability Theory	Automata Theory

What went wrong?

Engineering	Computer Science
Differential Equations	Logic
Linear Algebra	Discrete Structures
Probability Theory	Automata Theory
Mature	Promising

What went wrong?

Engineering	Computer Science
Differential Equations	Logic
Linear Algebra	Discrete Structures
Probability Theory	Automata Theory
Mature	Promising ?

What went wrong?

Engineering

Theories of estimation
Theories of robustness

Computer Science

Theories of correctness

Temptation: “Programs are mathematical objects.”

What went wrong?

Engineering

Theories of estimation
Theories of robustness

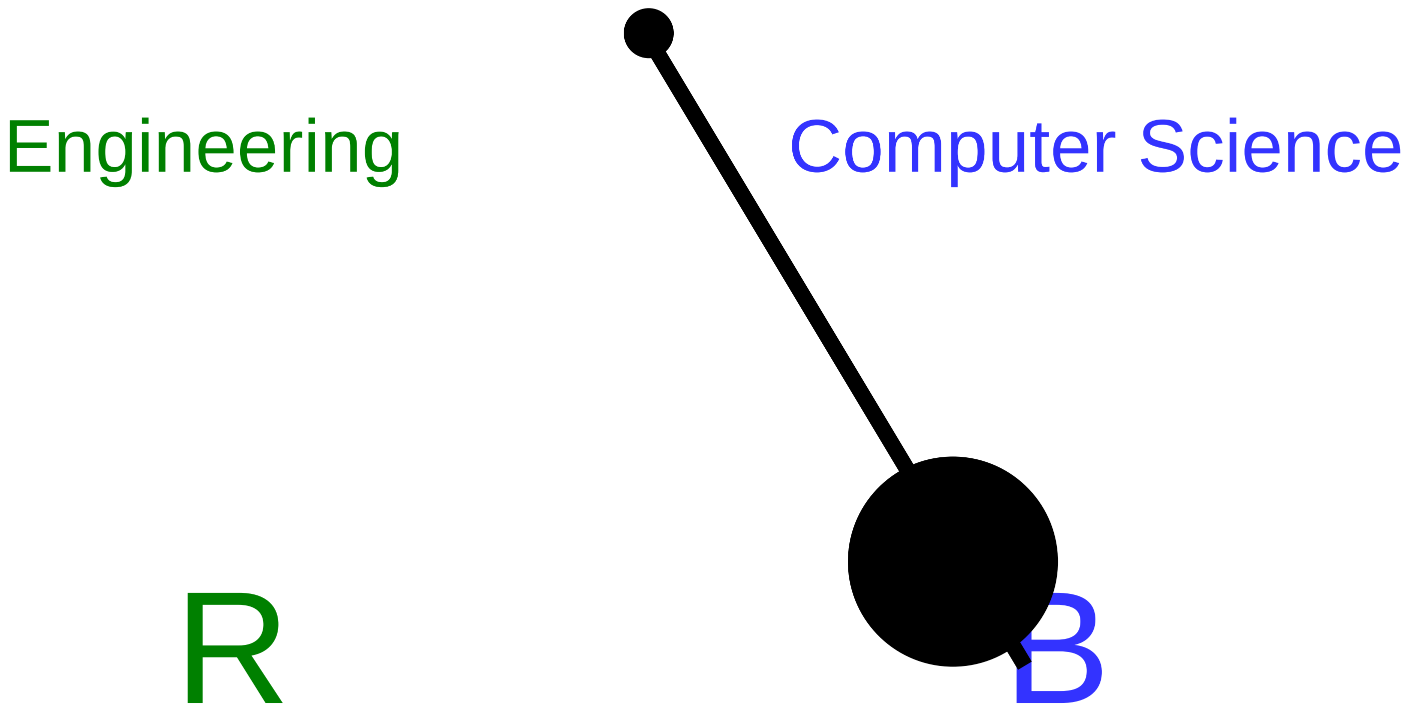
R

Computer Science

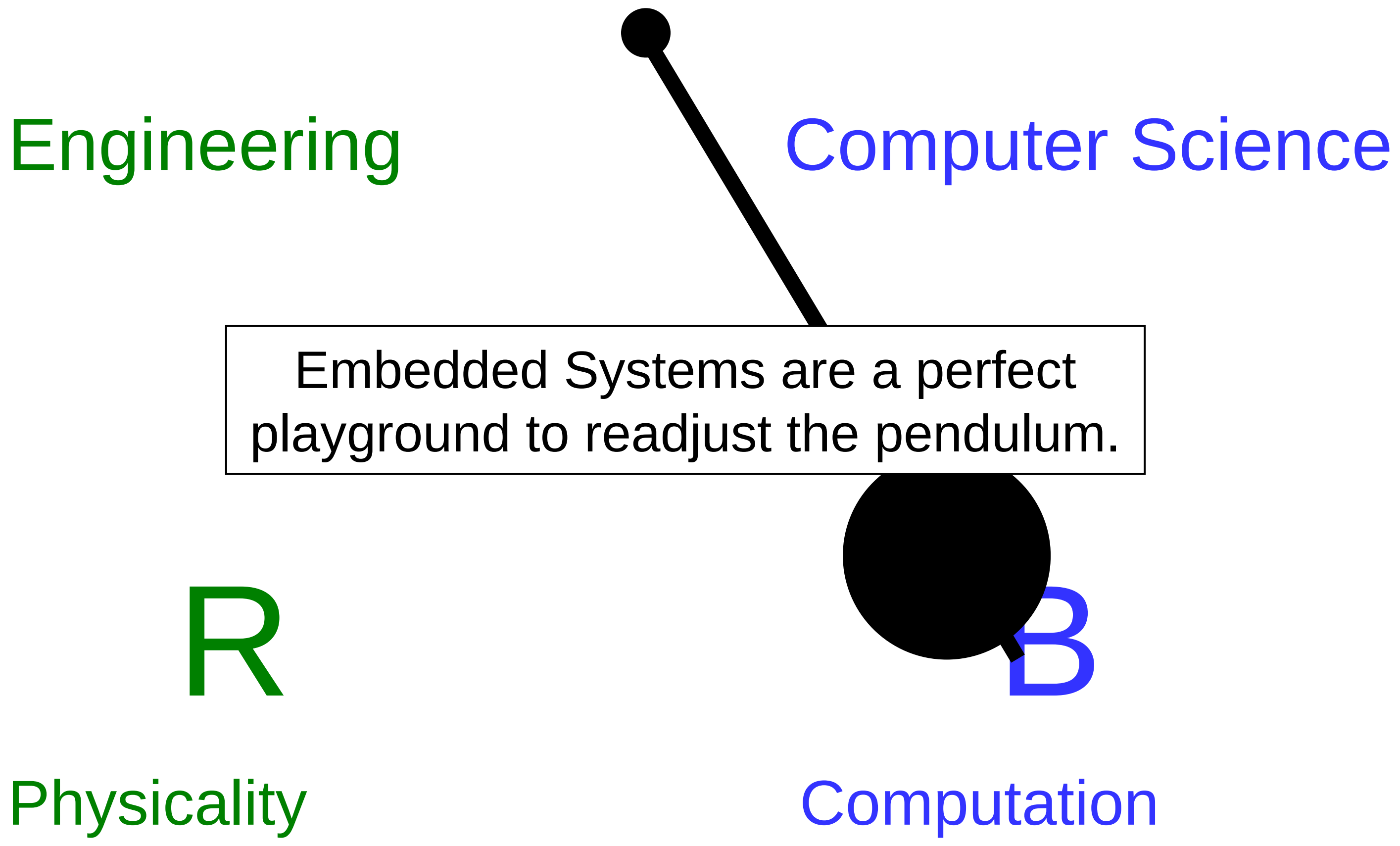
Theories of correctness

B

Maybe we went too far?



Maybe we went too far?



Embedded Systems are a perfect playground to readjust the pendulum.

The Challenge

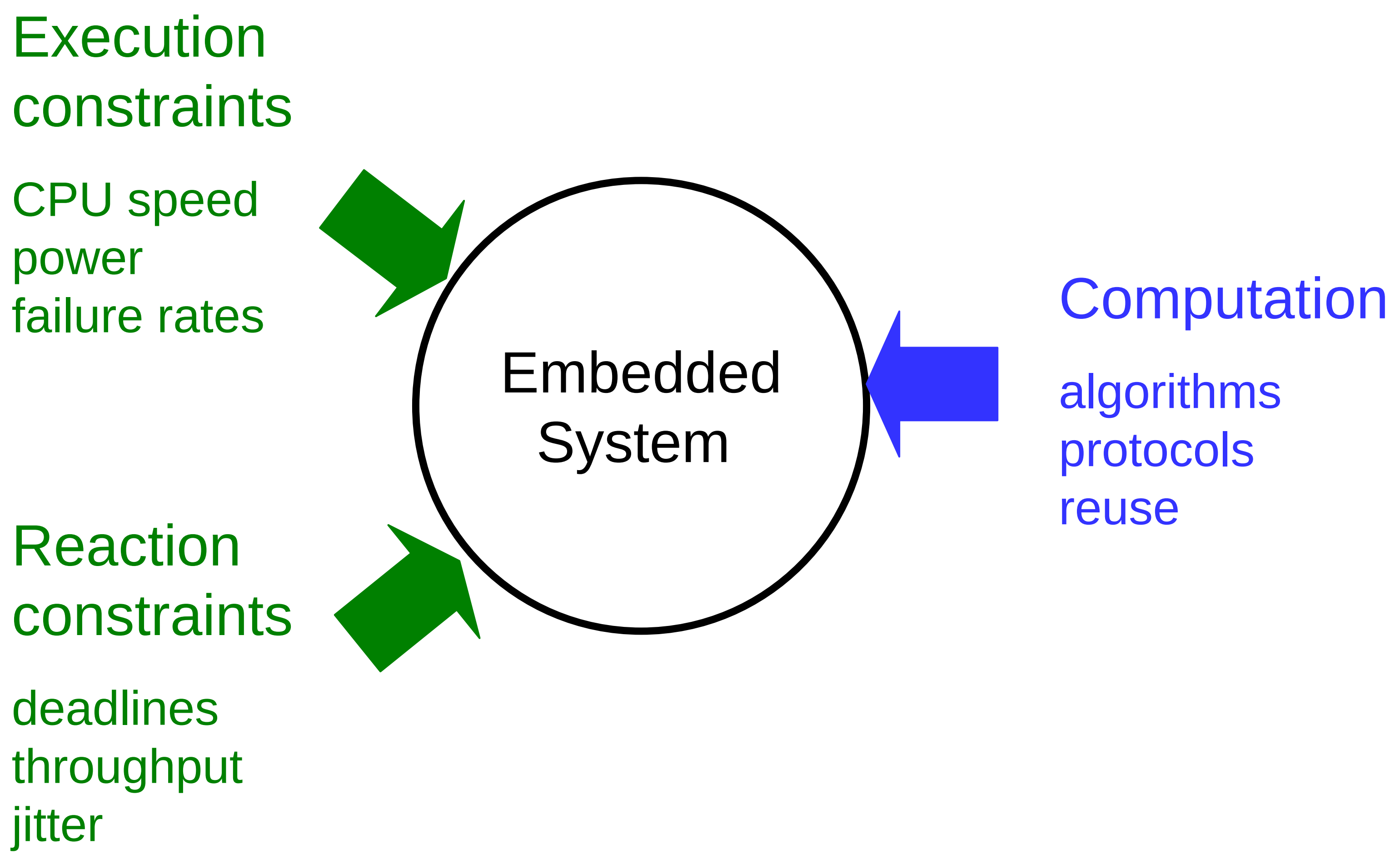
We need a new formal foundation for embedded systems, which systematically and even-handedly re-marries computation and physicality.

The Challenge

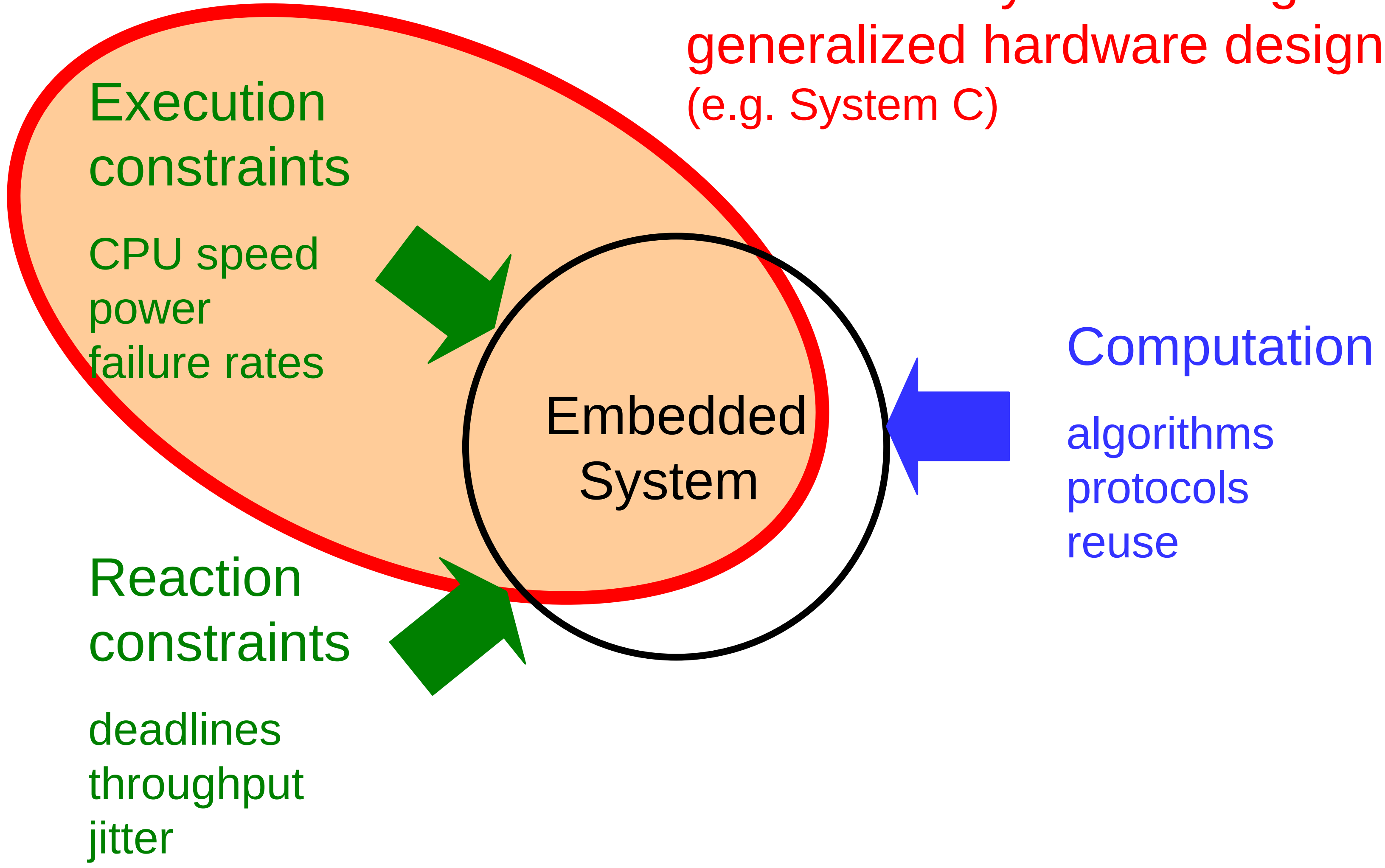
We need a new formal foundation for computational systems, which systematically and even-handedly re-marries performance and robustness.

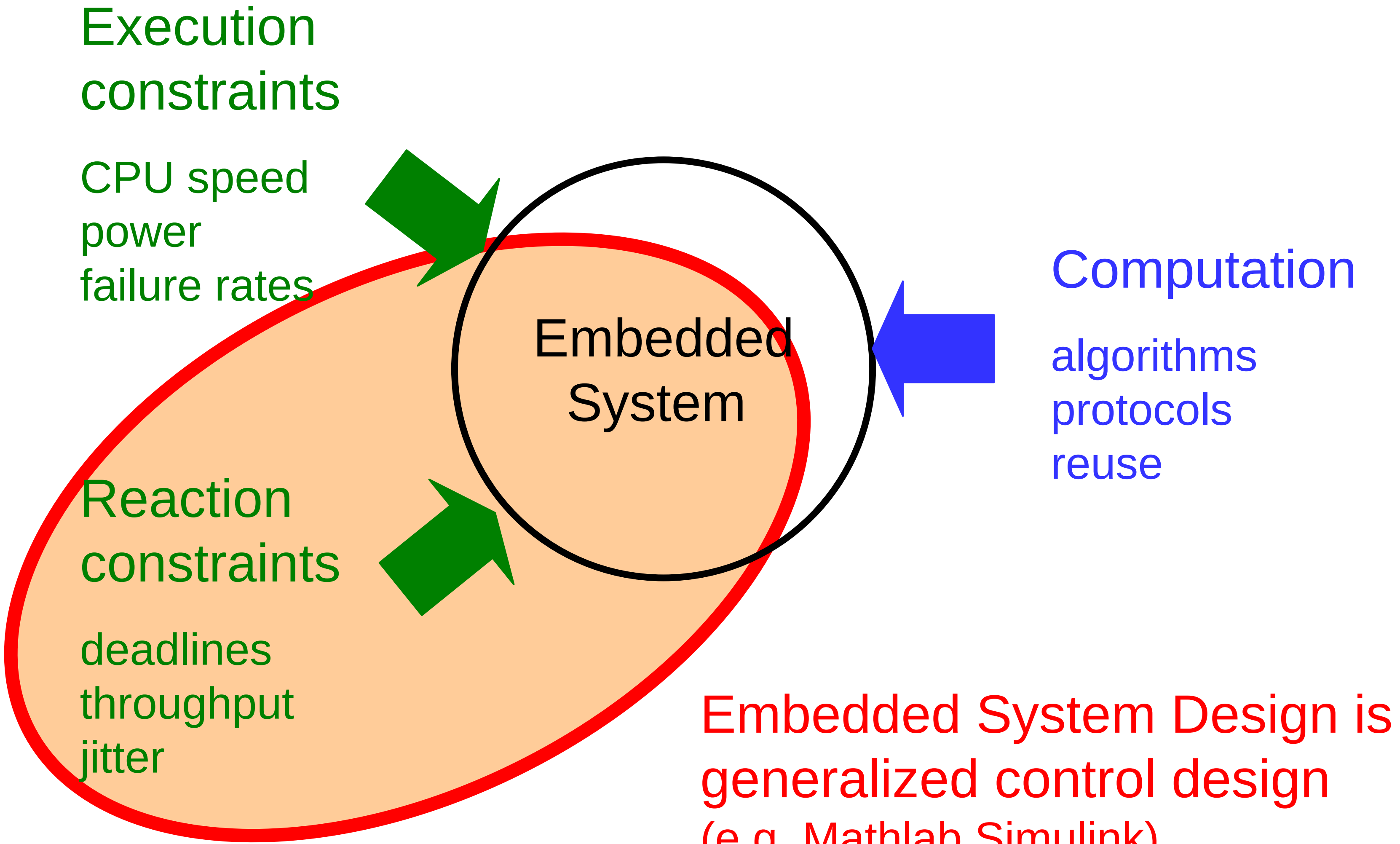
What is being computed?
At what cost?

How does the performance change under disturbances? (change of context; change of resources; failures; attacks)


Execution constraints
CPU speed
power
failure rates
Embedded System
Computation
algorithms
protocols
reuse
Reaction constraints
deadlines
throughput
jitter

Embedded System Design is generalized hardware design
(e.g. System C)





Embedded System Design is generalized control design
(e.g. Mathlab Simulink)

Current State of Affairs

50 years of computer science are largely ignored in embedded systems design: it is as if there were no choice between automatically synthesizing code on one hand, and assembly coding on the other hand.

Software is often the most costly and least flexible part of an embedded system.

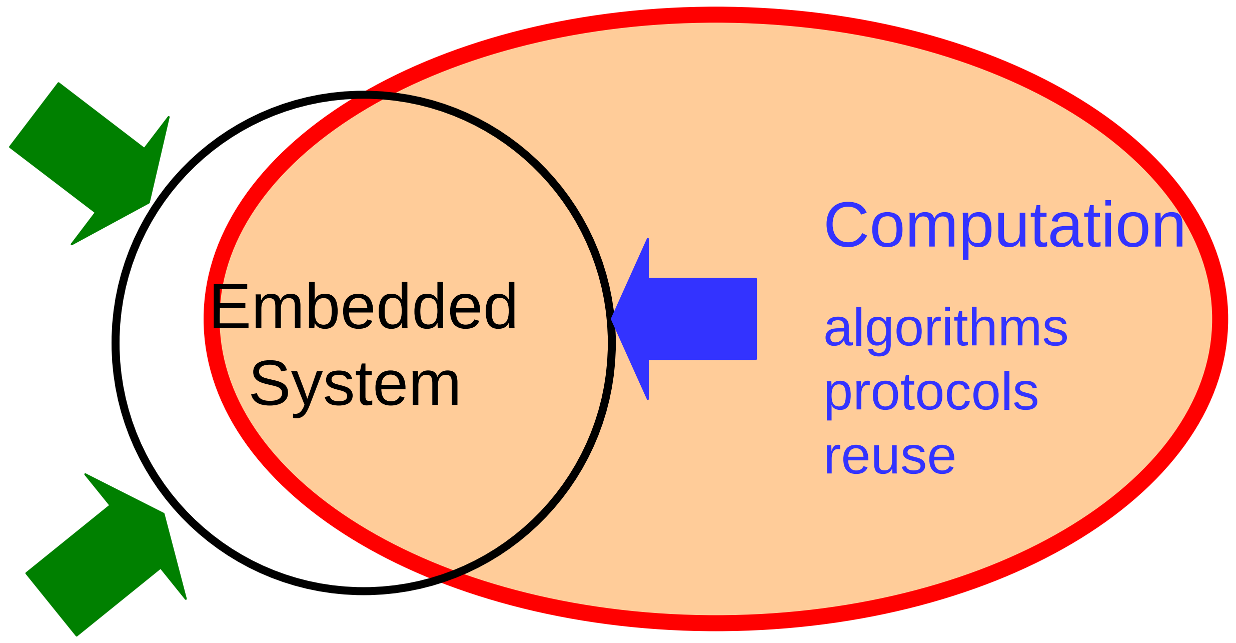
Embedded System Design should not be left to electrical engineers

Execution constraints

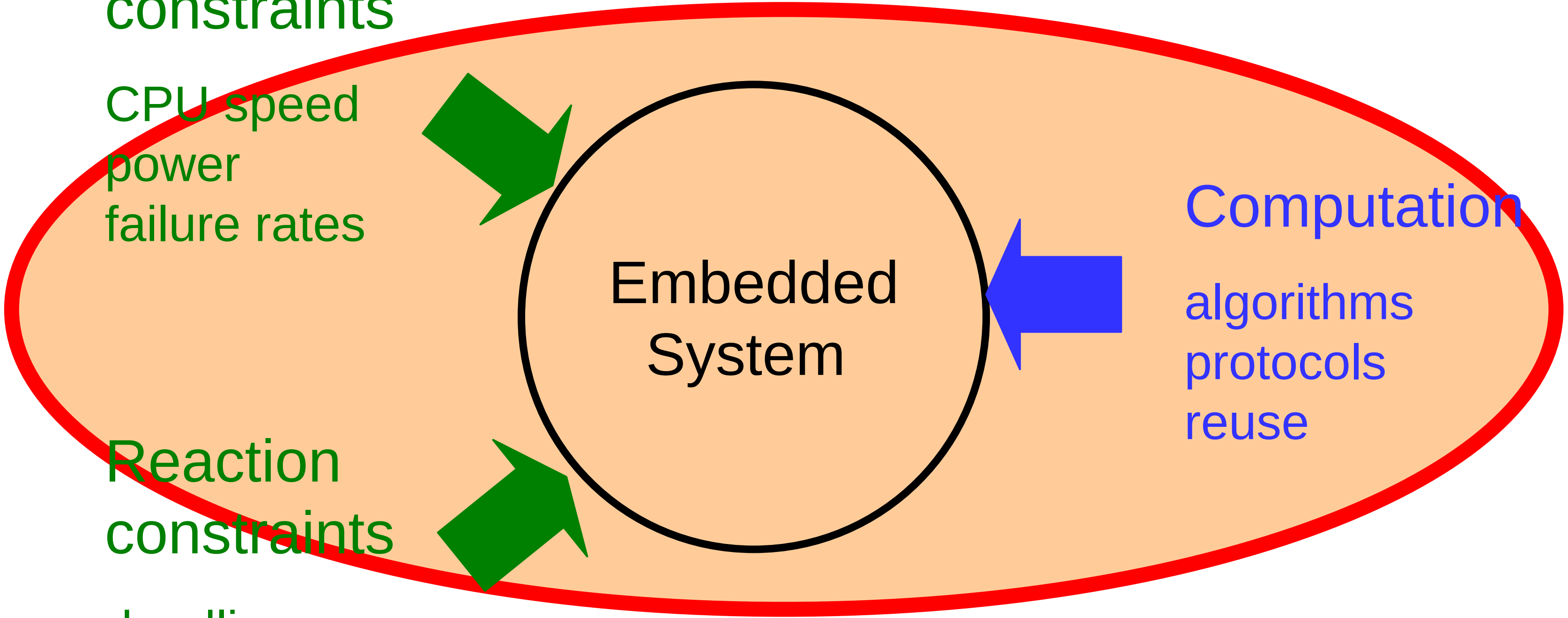
CPU speed
power
failure rates

Reaction constraints

deadlines
throughput
jitter



Embedded System Design should not be left to electrical engineers

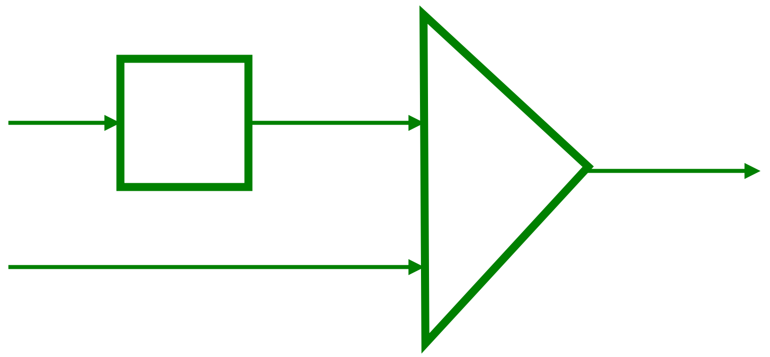


BUT: we need to revisit and revise our most basic paradigms to include methods from EE

Subchallenge 1:
Integrate Analytical and Computational Modeling

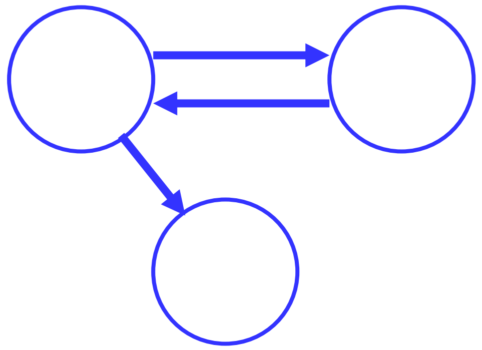
Engineering

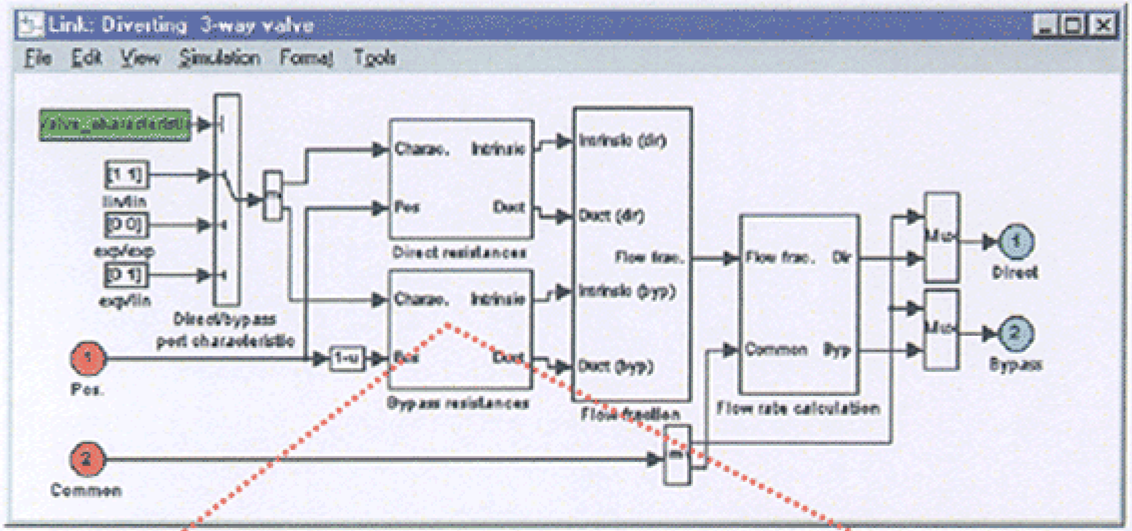
Component model: transfer function
Composition: parallel
Connection: data flow



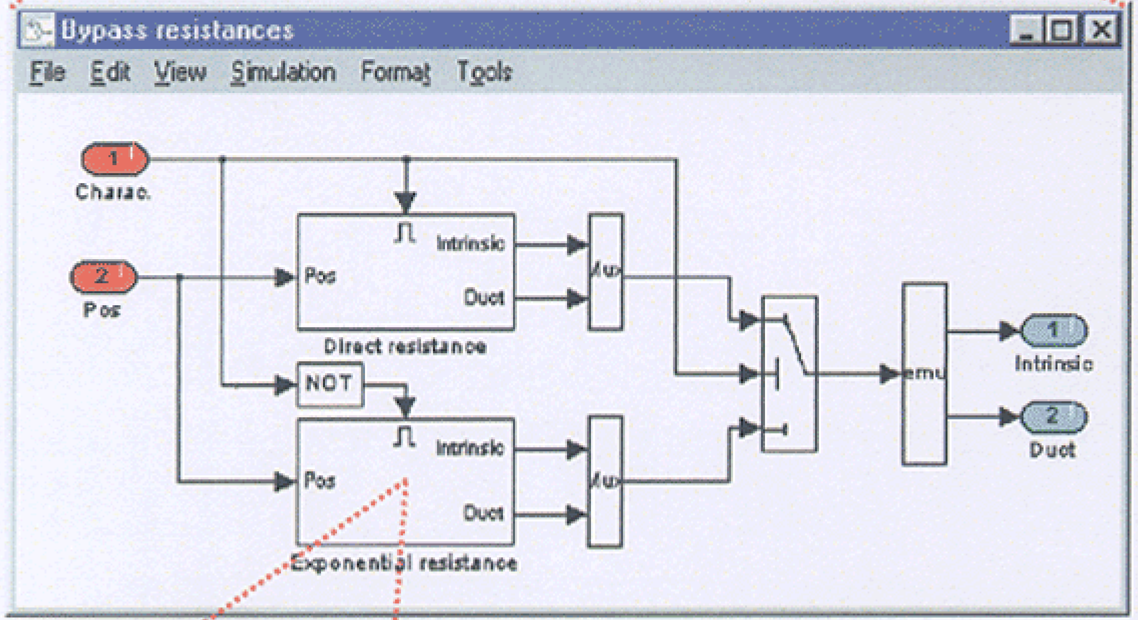
Computer Science

Component model: subroutine
Composition: sequential
Connection: control flow

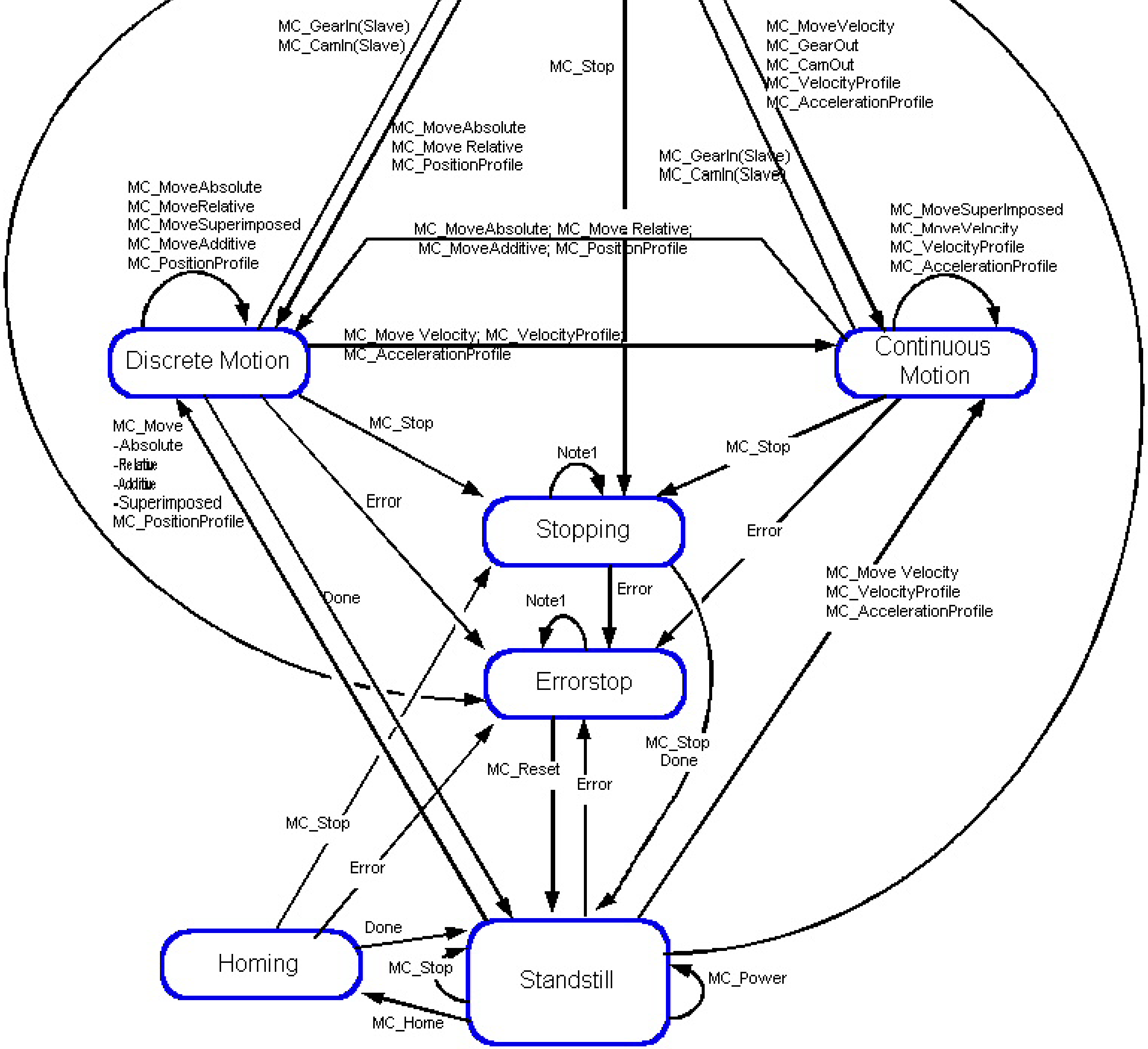




First layer of the functional scheme of the valve model



Second inner layer for calculations of bypass resistances


MC_GearIn(Slave)
MC_CamIn(Slave)
MC_Stop
MC_MoveVelocity
MC_GearOut
MC_CamOut
MC_VelocityProfile
MC_AccelerationProfile
MC_MoveAbsolute
MC_Move Relative
MC_PositionProfile
MC_GearIn(Slave)
MC_CamIn(Slave)
MC_MoveAbsolute
MC_MoveRelative
MC_MoveSuperimposed
MC_MoveAdditive
MC_PositionProfile
MC_MoveAbsolute; MC_Move Relative;
MC_MoveAdditive; MC_PositionProfile
MC_MoveSuperImposed
MC_MoveVelocity
MC_VelocityProfile
MC_AccelerationProfile
Discrete Motion
MC_Move Velocity; MC_VelocityProfile;
MC_AccelerationProfile
Continuous Motion
MC_Move
-Absolute
-Relative
-Additive
-Superimposed
MC_PositionProfile
MC_Stop
Note1
MC_Stop
Error
Stopping
Error
Done
Error
Note1
MC_Move Velocity
MC_VelocityProfile
MC_AccelerationProfile
Errorstop
MC_Stop
Done
MC_Reset
Error
MC_Stop
Error
Done
Homing
MC_Stop
Standstill
MC_Power
MC_Home

Analytical Models

Defined by equations
Deterministic or probabilistic

Computational Models

Defined by programs
Executable by abstract machines

Analytical Models	Computational Models
Defined by equations	Defined by programs
Deterministic or probabilistic	Executable by abstract machines
Strengths:	
Concurrency	Dynamic change
Real time	Complexity theory
Quantitative constraints (power, QoS, mean-time-to-failure)	Nondeterminism (abstraction hierarchies, partial specifications)

Analytical Models	Computational Models
Defined by equations Deterministic or probabilistic	Defined by programs Executable by abstract machines
Strengths:	
Concurrency Real time Quantitative constraints (power, QoS, mean-time-to-failure)	Dynamic change Complexity theory Nondeterminism (abstraction hierarchies, partial specifications)
Tool support:	
Average-case analysis Optimization Continuous mathematics (differential equations, stochastic processes)	Worst-case analysis Constraint satisfaction Discrete mathematics (logic, combinatorics)

Analytical Models	Computational Models
Defined by equations Deterministic or probabilistic	Defined by programs Executable by abstract machines
Strengths:	
Concurrency Real time Quantitative constraints (power, QoS, mean-time-to-failure)	Dynamic change Complexity theory Nondeterminism (abstraction hierarchies, partial specifications)
Tool support:	
Average-case analysis Optimization Continuous mathematics (differential equations, stochastic processes)	Worst-case analysis Constraint satisfaction Discrete mathematics (logic, combinatorics)
Main paradigm:	
Synthesis	Verification

Analytical Models

Defined by equations
Deterministic or prob c

Strengths:

Concurrency
Real time
Quantitative c aints (power,
QoS, mean o-failure)

Tool sur

Averag e analysis
Optim n
Cor s mathematics
(di ial equations,
s stic processes)

n paradigm:

nthesis

BEST-EFFORT SYSTEMS DESIGN

Computational Models

Defined by program
Executable by abst es

Dynamic chan
Complexity t
Nondeterm action
hierarchie ecifications)

Wors ysis
Cor faction
Di ematics (logic,
s)

Verification

GUARANTEED-EFFORT SYSTEMS DESIGN

Subchallenge 1:
Integrate Analytical and Computational Modeling

Best-Effort
Systems Design

Guaranteed-Effort
Systems Design

We need both.
We need to be able to intelligently trade off costs and risks.

We need effective model transformations.

Subchallenge 1:
Integrate Analytical and Computational Modeling

Best-Effort
Systems Design

Guaranteed-Effort
Systems Design

We need both.
We need to be able to intelligently trade off costs and risks.

We need effective model transformations.

We need engineers that understand both complexities.

Subchallenge 2:

Balance the Opposing Demands of Heterogeneity and Constructivity

Subchallenge 2:

Balance the Opposing Demands of Heterogeneity and Constructivity

Sources of heterogeneity

Components
Levels of abstraction
Views (aspects)
Operating contexts

Degrees of constructivity

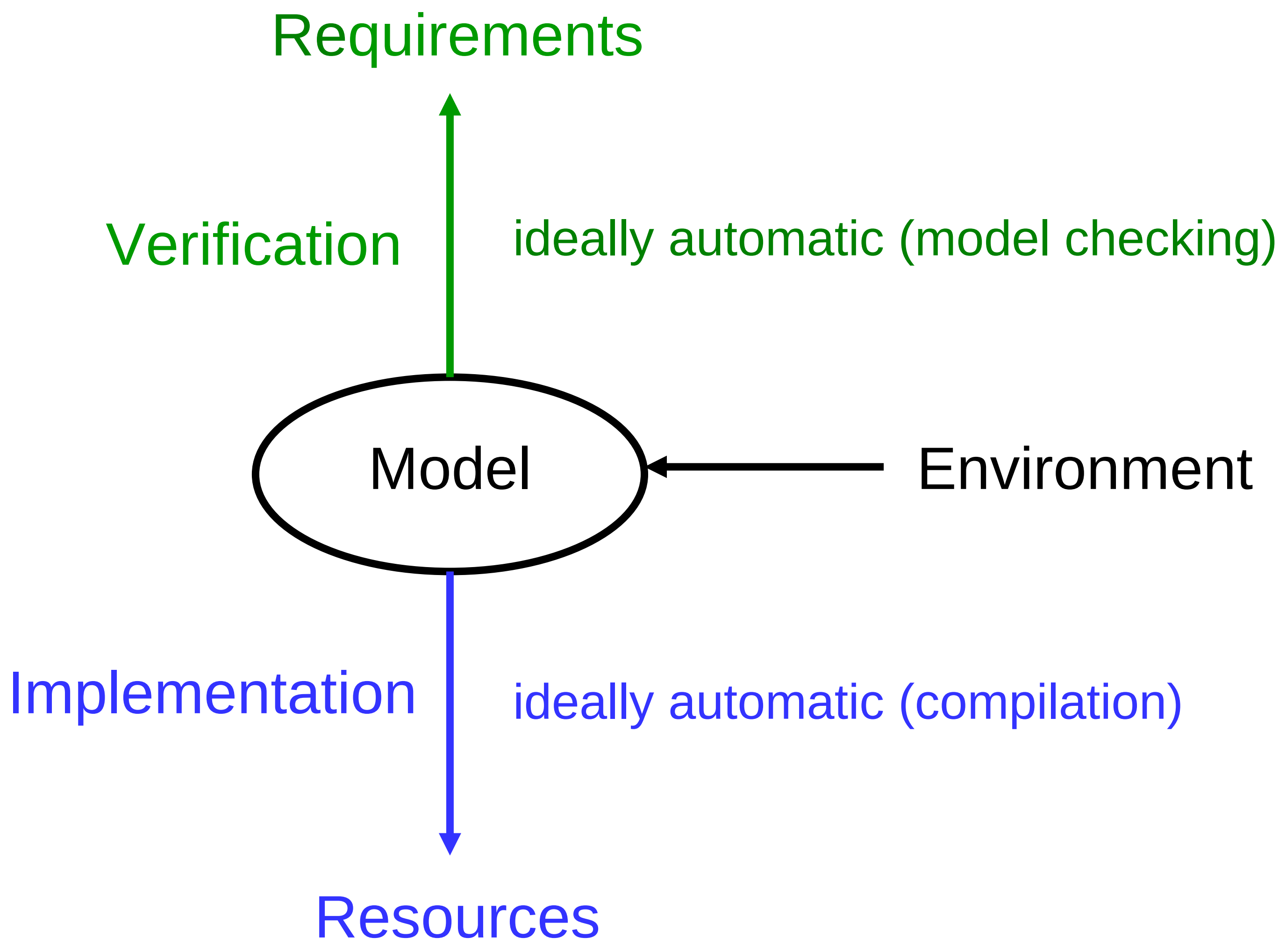
1 Synthesis / compilation
2 Correctness by design disciplines
3 Automatic verifiability
4 Formal verifiability

Difficulties

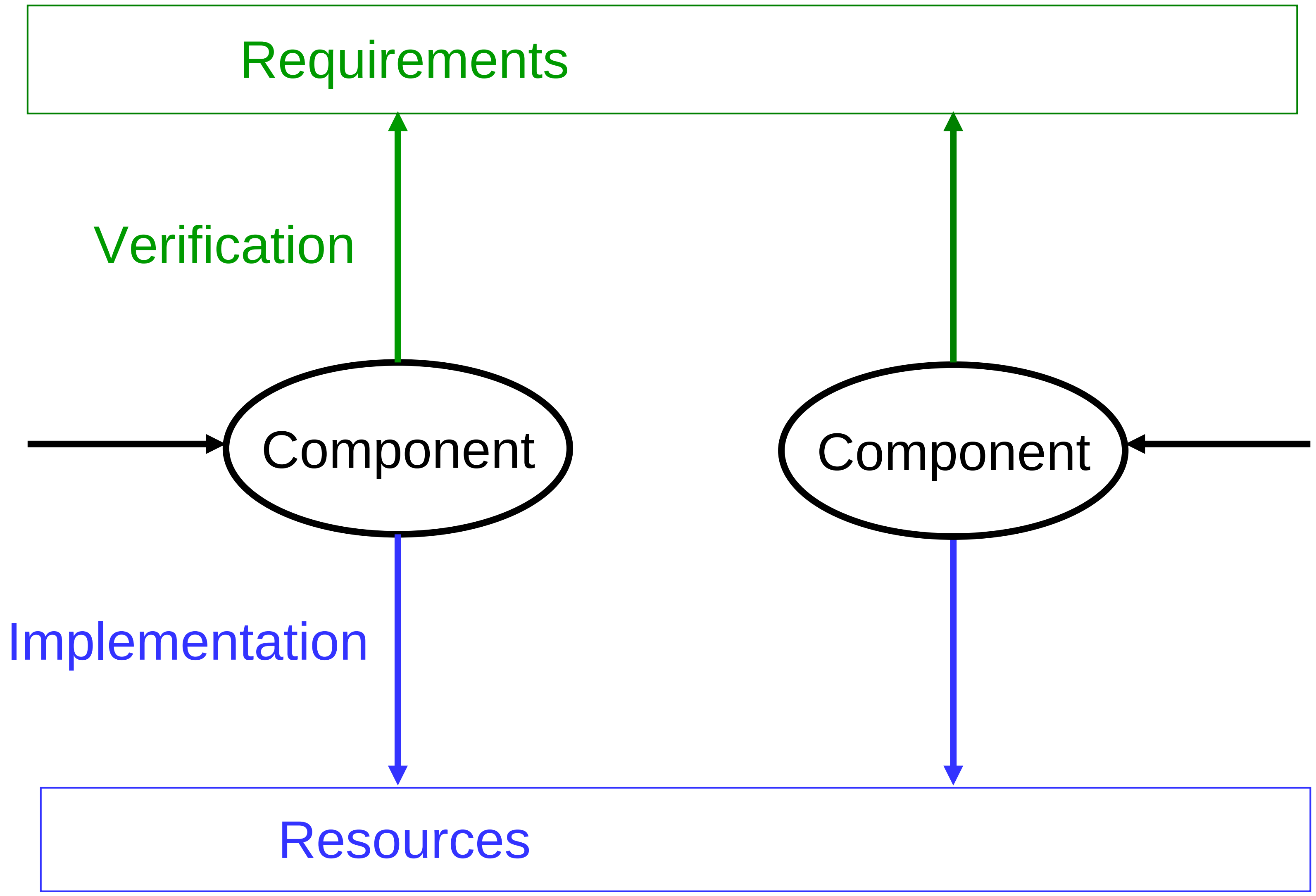
Models and methods need to be compositional in order to scale.

Whenever possible: noninterference

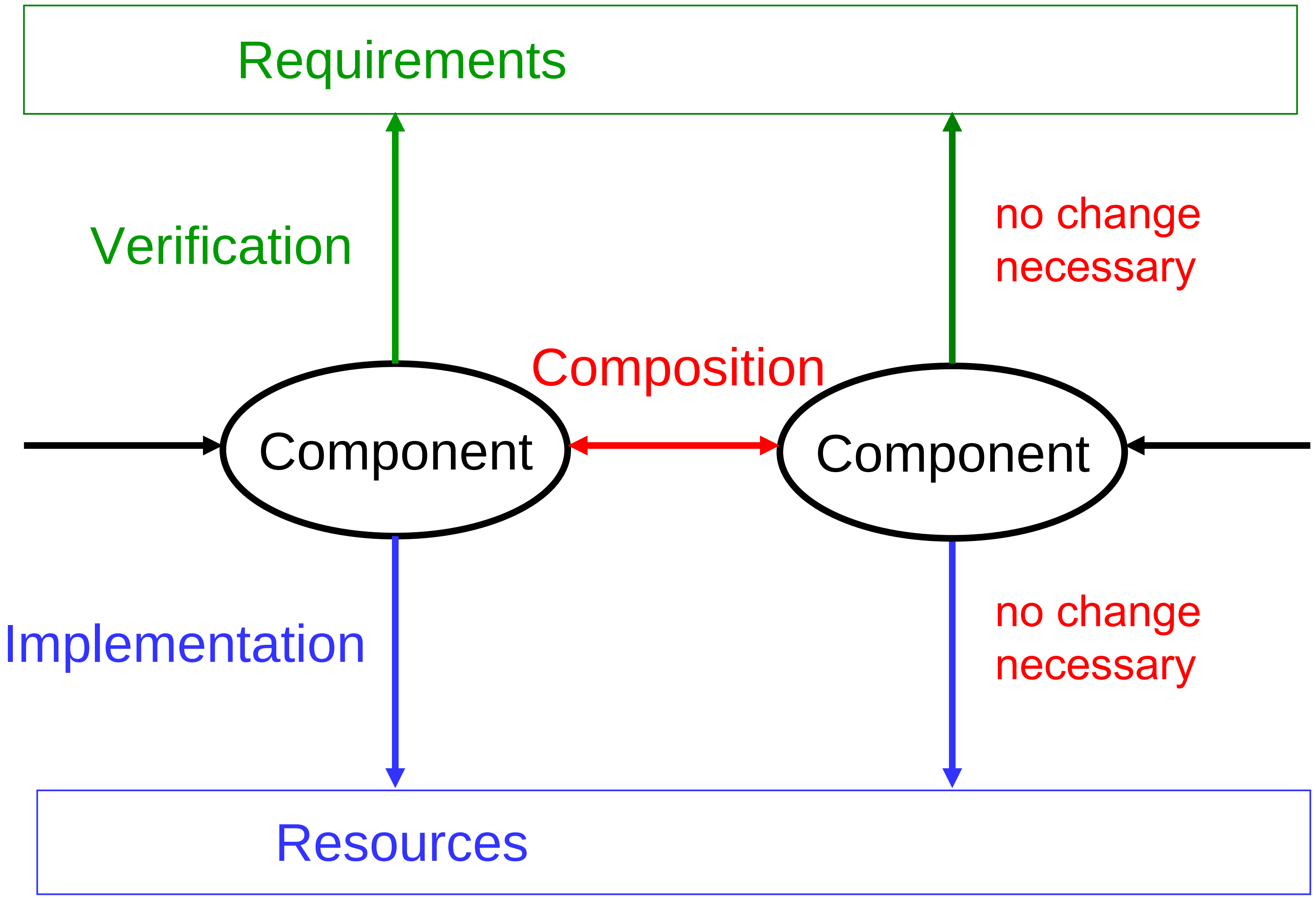
Model-based Design


Requirements
Verification
ideally automatic (model checking)
Model
Environment
Implementation
ideally automatic (compilation)
Resources

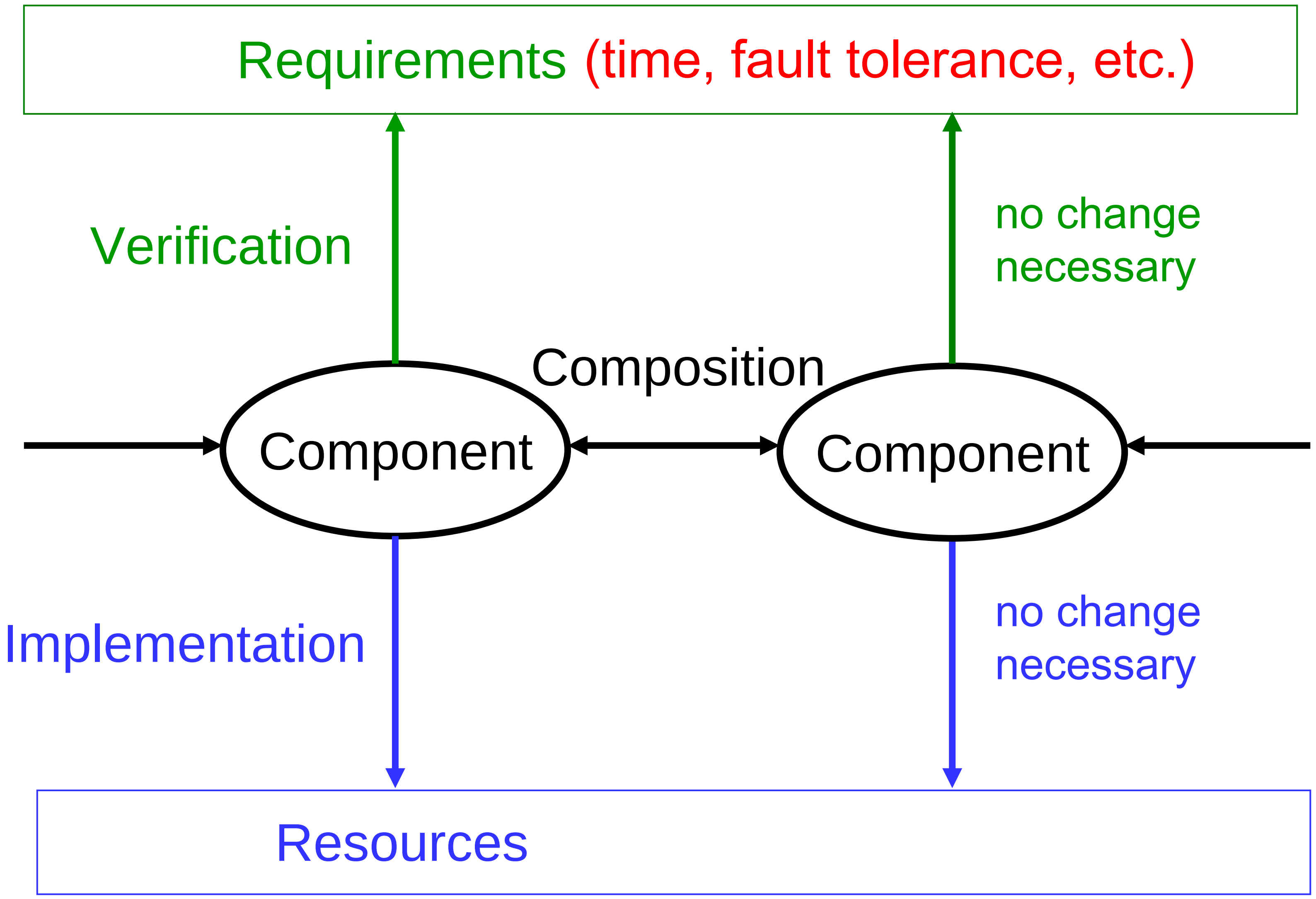
Model-based Design



Noninterference



Noninterference



Difficulties

Models and methods need to be compositional in order to scale.

Whenever possible:	noninterference
Next best solution:	check interface compatibility

Difficulties

Models and methods need to be compositional in order to scale.

Whenever possible:	noninterference
Next best solution:	check interface compatibility

Models and methods need to support robustness in addition to functionality.

Whenever possible:	continuity

Difficulties

Models and methods need to be compositional in order to scale.

Whenever possible:	noninterference
Next best solution:	check interface compatibility

Models and methods need to support robustness in addition to functionality.

Whenever possible:	continuity
Next best solution:	quantify overengineering

Some Examples

1 Heterogeneity through hybrid automata

2 Continuity through discounting

3 Noninterference through fixed logical execution times

4 Compositionality through automaton interfaces

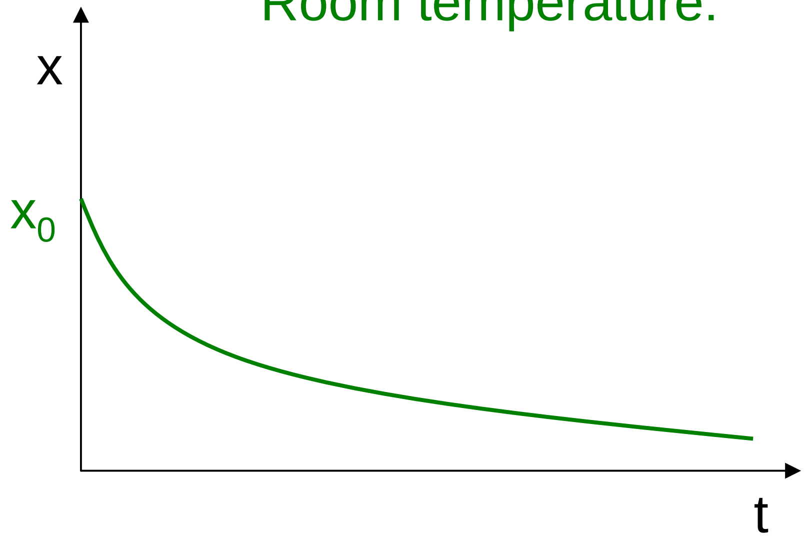
Continuous Dynamical Systems

State space: R^n

Dynamics: initial condition + differential equations

Room temperature: $x(0) = x_0$

$x'(t) = -K \cdot x(t)$



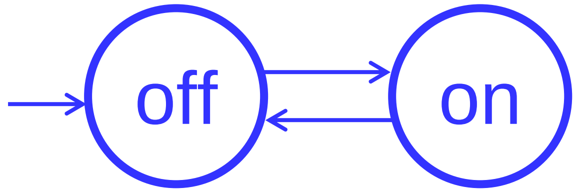
Analytic complexity.

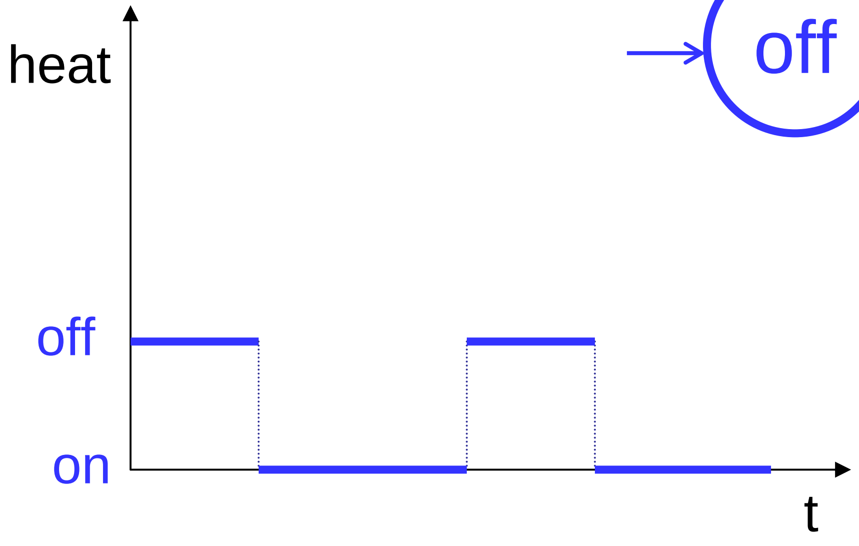
Discrete Transition Systems

State space: B^m

Dynamics: initial condition + transition relation

Heater:



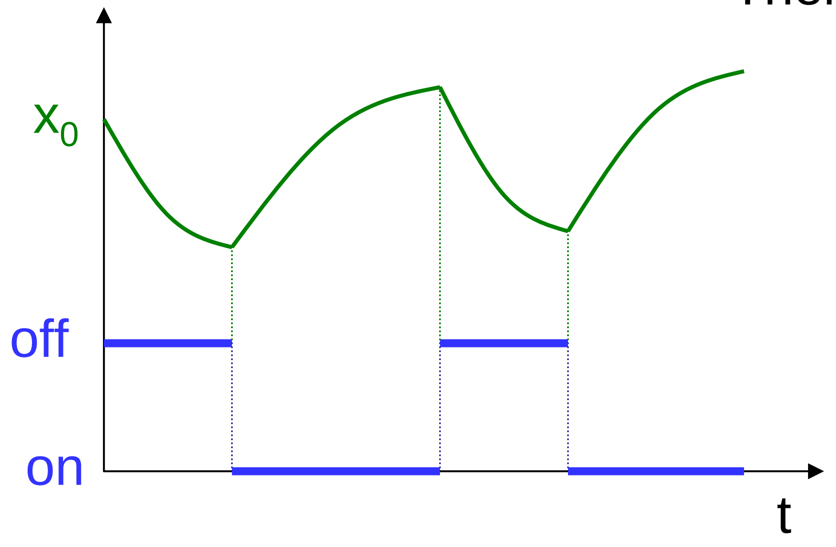


Combinatorial complexity.

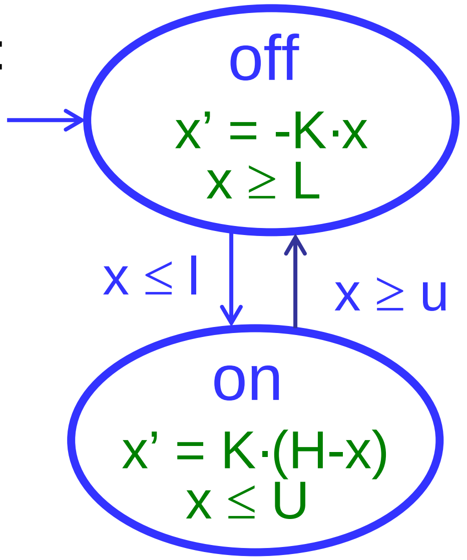
Hybrid Automata

State space: $B^m \times R^n$

Dynamics: initial condition + transition relation + differential equations



Thermostat:



Some Examples

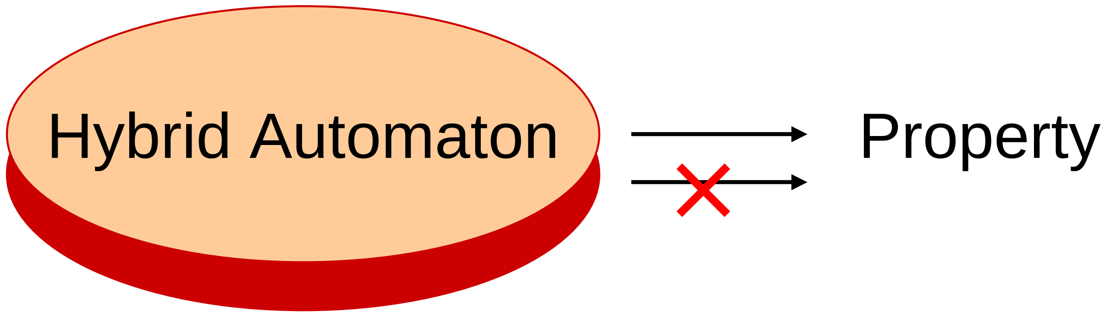
1 Heterogeneity through hybrid automata

2 Continuity through discounting

3 Noninterference through fixed logical execution times

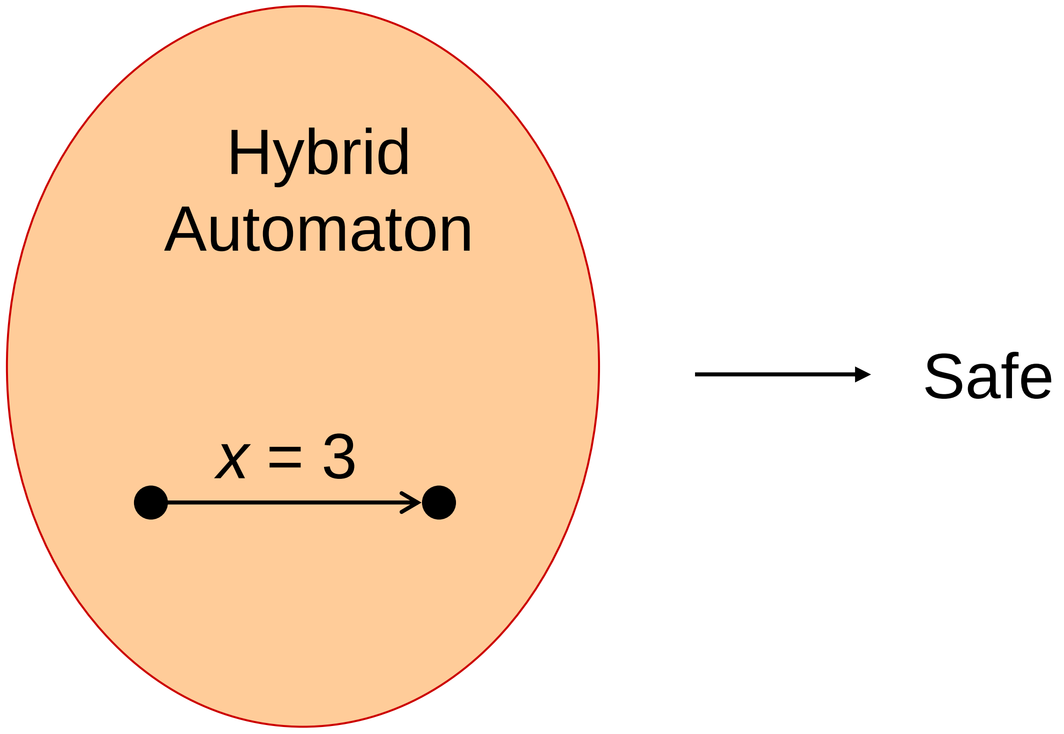
4 Compositionality through automaton interfaces

(Non)Robustness

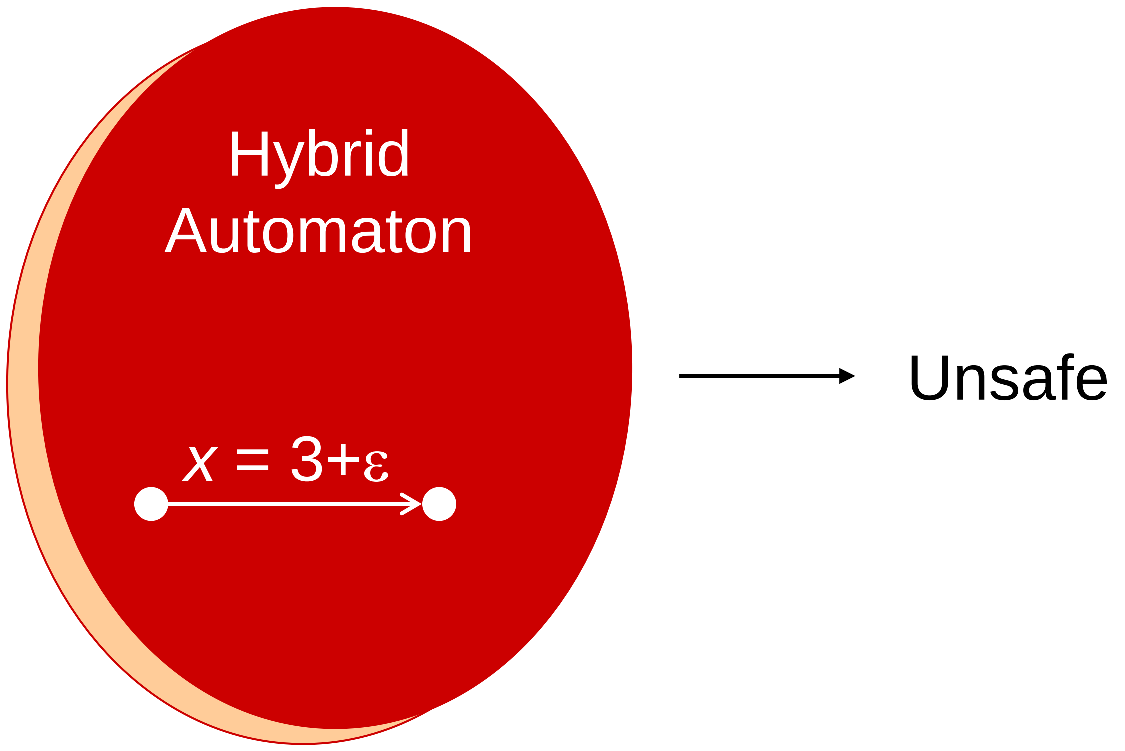


slightly perturbed automaton

(Non)Robustness



(Non)Robustness



A Continuous Theory of Systems

value(Model,Property): States $\to$ B

$$\Downarrow$$

value(Model,Property): States $\to$ R

A Continuous Theory of Systems

value(Model,Property): States $\to$ B

value(m,$\diamond T$) = (μX) ($T \vee$ pre(X))

$\Downarrow$

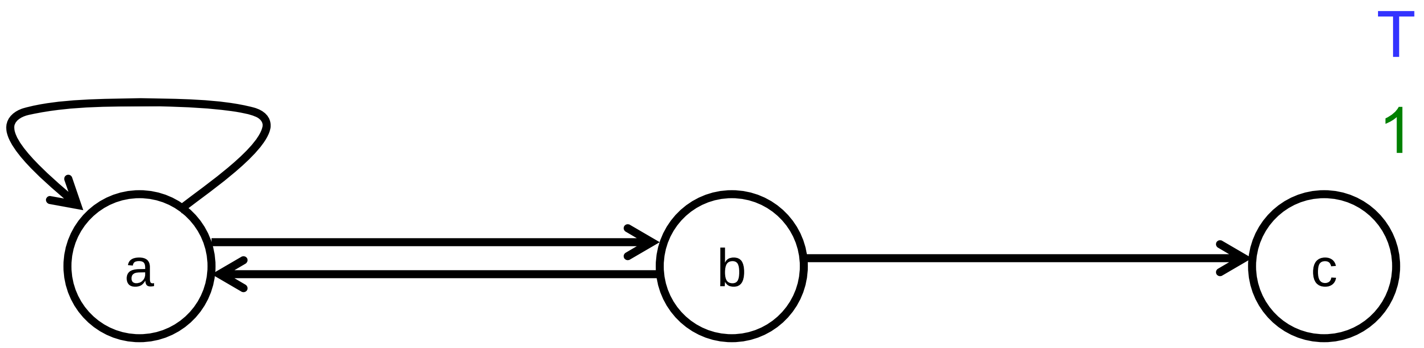
discountedValue(Model,Property): States $\to$ R

discountedValue(m,$\diamond T$) = (μX) max(T, $\lambda \cdot$pre(X))

$\uparrow$

discount factor $0<\lambda<1$

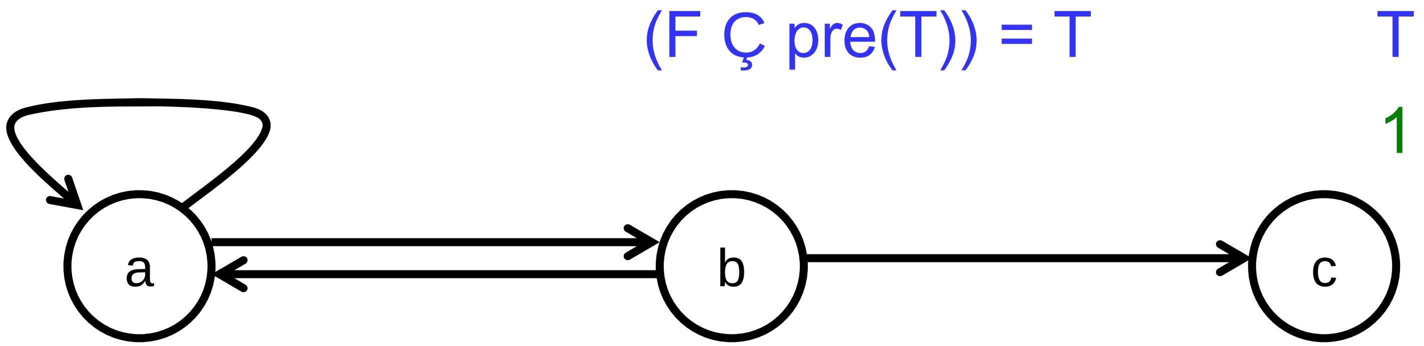
Reachability



$\diamond$ c … undiscounted property

$\diamond_{\lambda}$ c … discounted property

Reachability



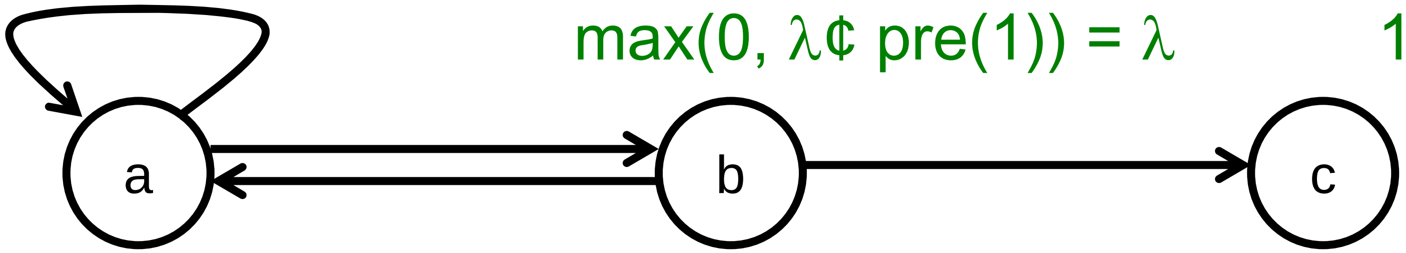
$\Diamond$ c … undiscounted property

$\Diamond_\lambda$ c … discounted property

Reachability

(F Ç pre(T)) = T T

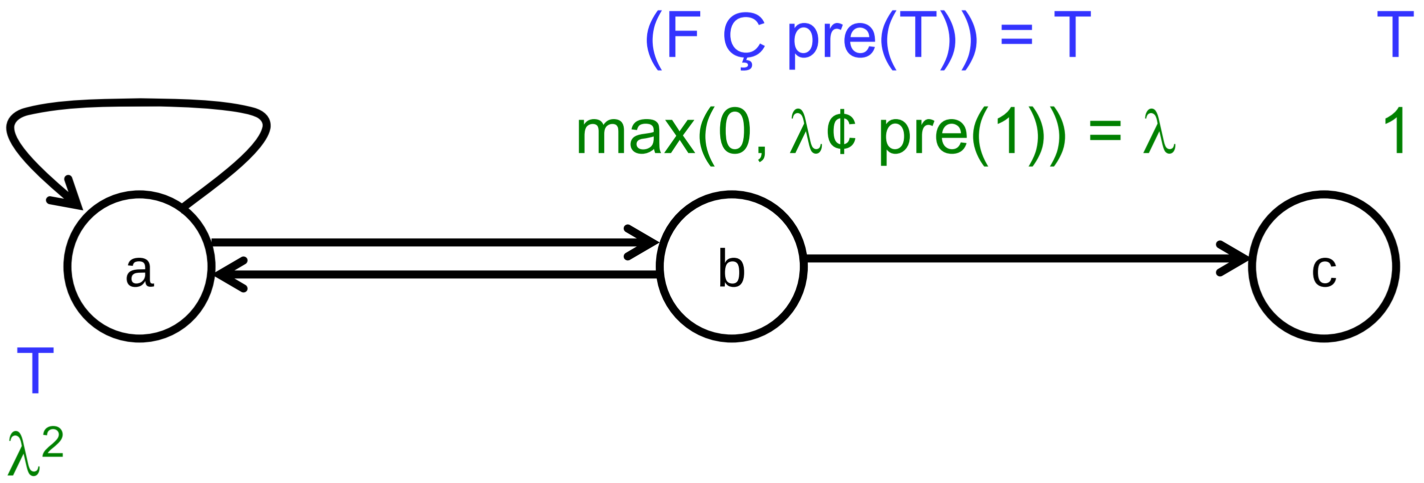
max(0, λ¢ pre(1)) = λ 1



$\Diamond$ c … undiscounted property

$\Diamond_{\lambda}$ c … discounted property

Reachability



$\Diamond$ c … undiscounted property

$\Diamond_\lambda$ c … discounted property

A Continuous Theory of Systems

Robustness Theorem [de Alfaro, H, Majumdar]:

If discountedBisimilarity(m_1,m_2) > 1 - ε,
then |discountedValue(m_1,p) - discountedValue(m_2,p)| < $f(\varepsilon)$.

A Continuous Theory of Systems

Robustness Theorem [de Alfaro, H, Majumdar]:

If discountedBisimilarity(m_1,m_2) > 1 - ε,
then |discountedValue(m_1,p) - discountedValue(m_2,p)| < $f(\varepsilon)$.

Further advantages of discounting:

-approximability because of geometric convergence
(avoids non-termination of verification algorithms)

-applies also to probabilistic systems and to games
(enables reasoning under uncertainty, and control)

Some Examples

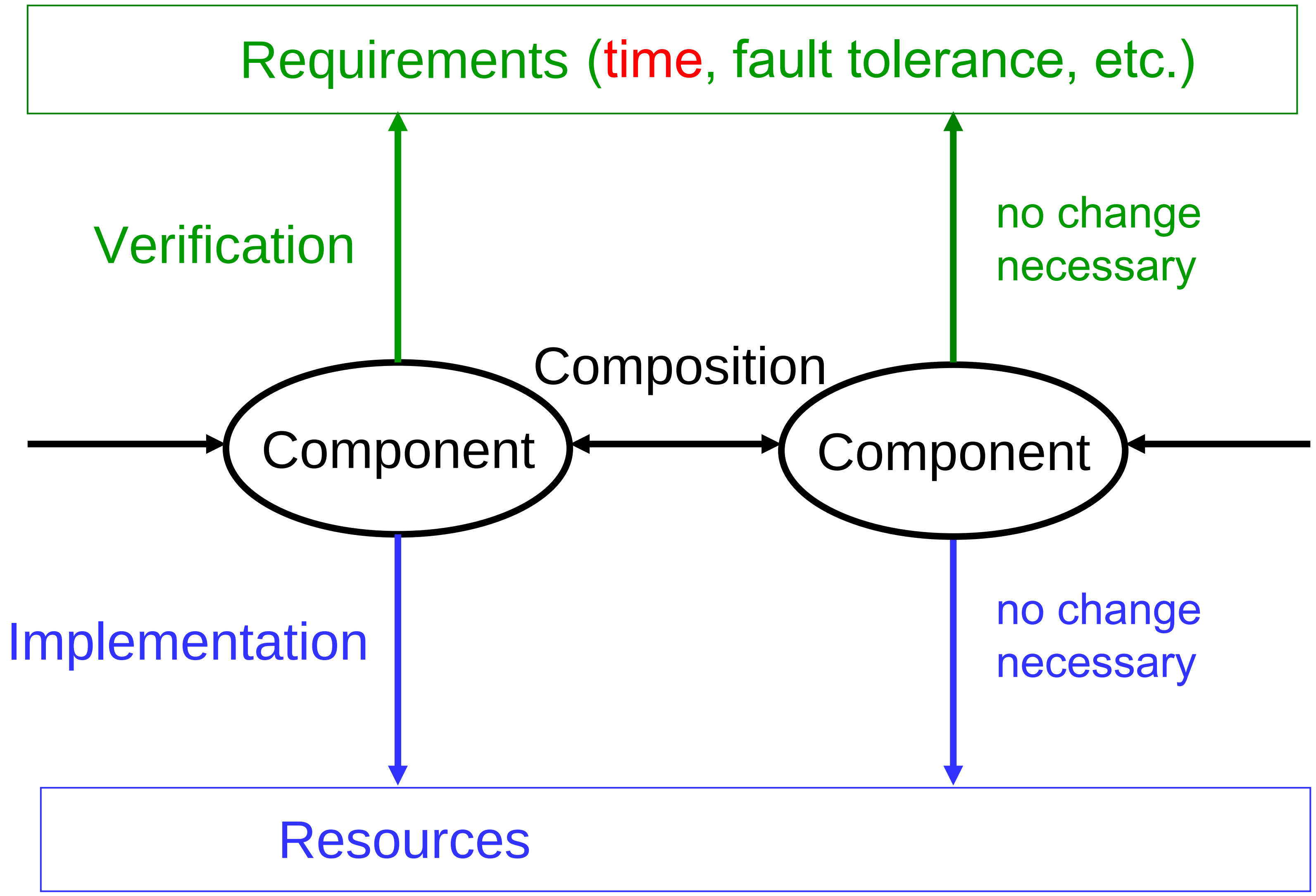
1 Heterogeneity through hybrid automata

2 Continuity through discounting

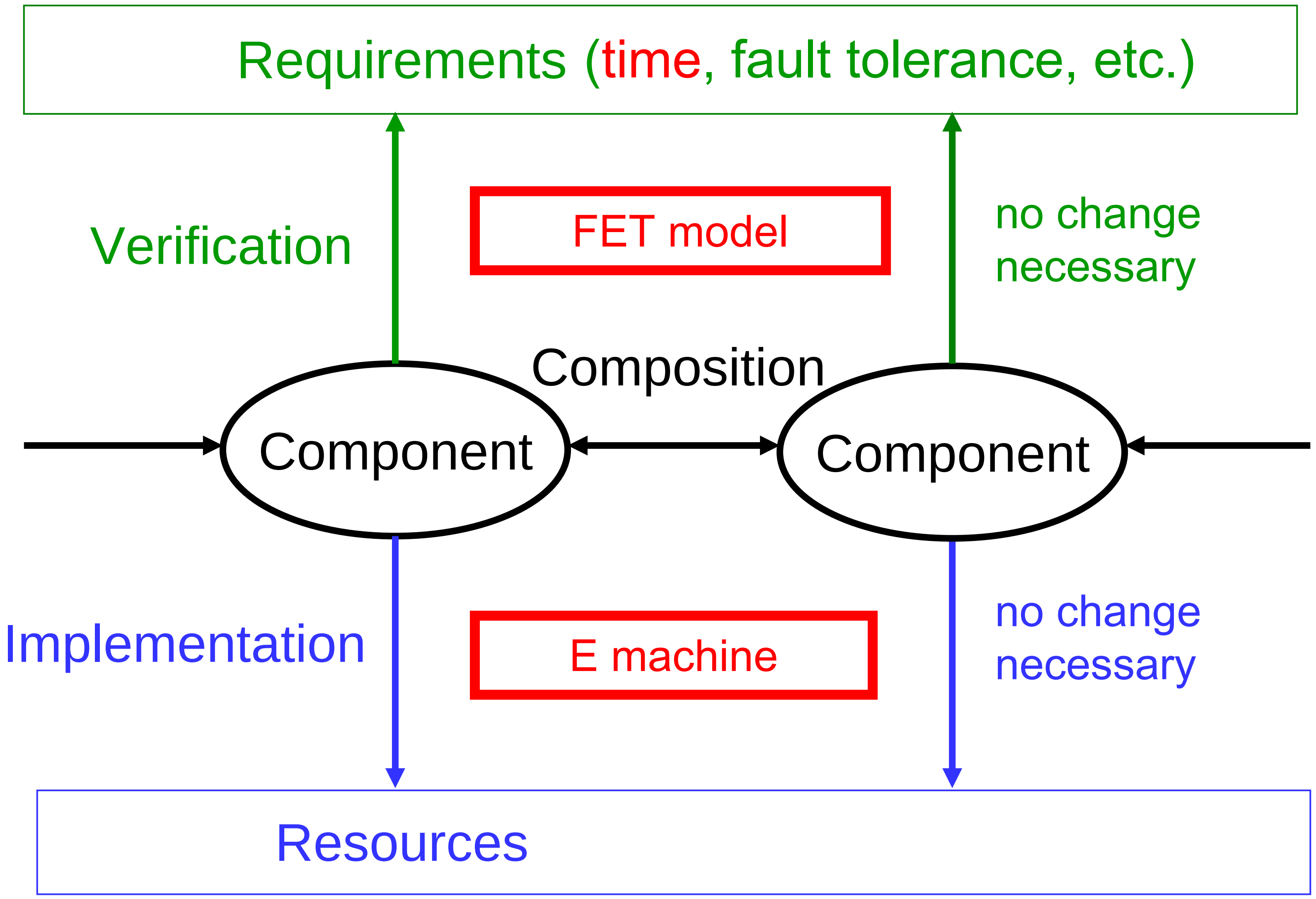
3 Noninterference through fixed logical execution times

4 Compositionality through automaton interfaces

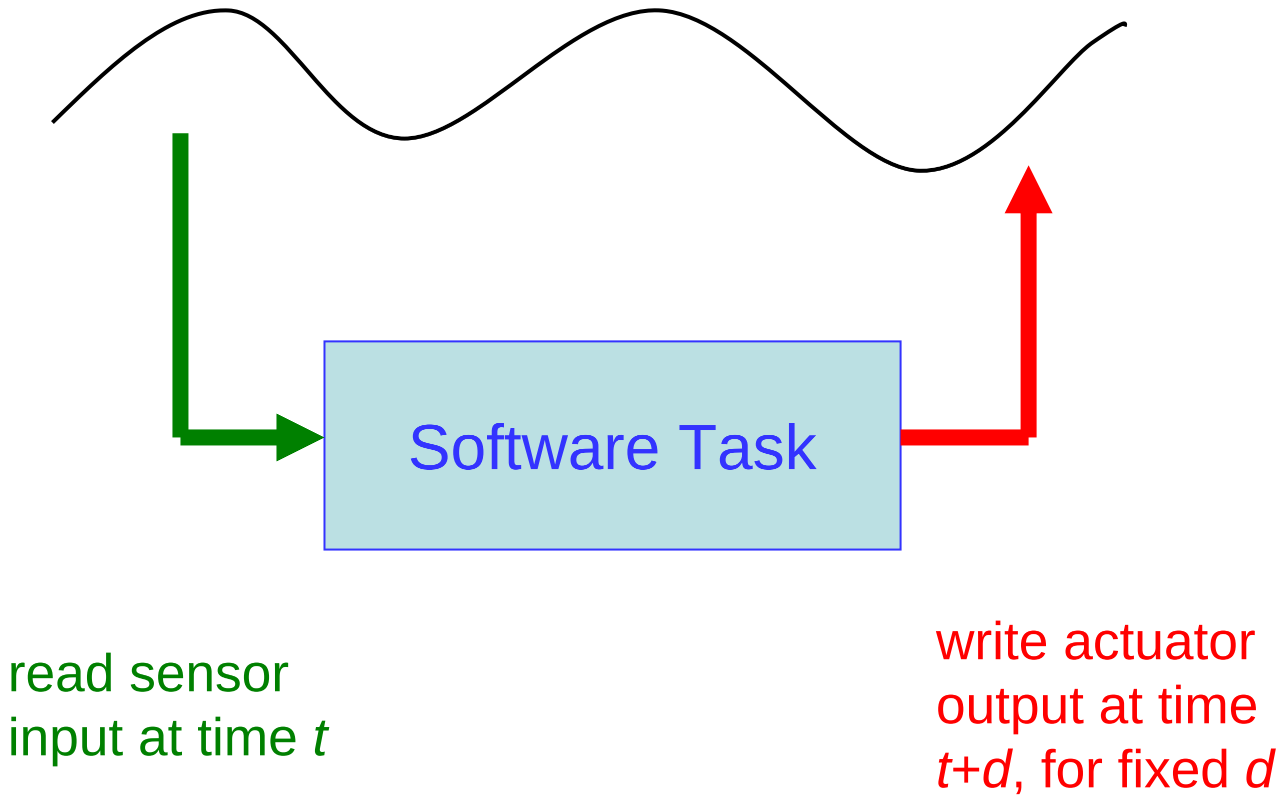
Compositionality



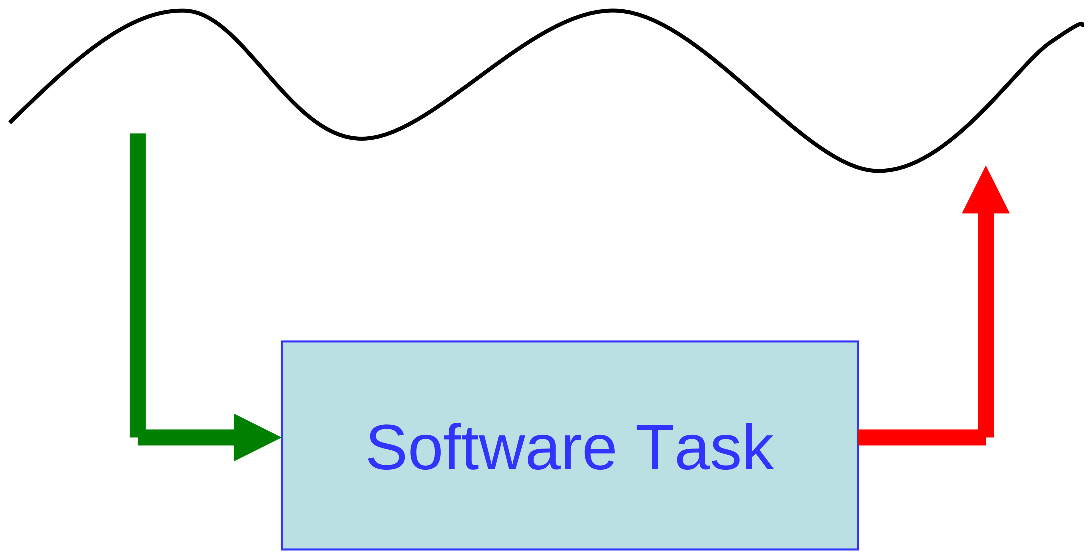
Compositionality



The FET (Fixed Execution Time) Assumption


Software Task
read sensor input at time *t*
write actuator output at time *t*+*d*, for fixed *d*

The FET (Fixed Execution Time) Assumption



read sensor input at time *t*

d>0 is the task's “fixed execution time"

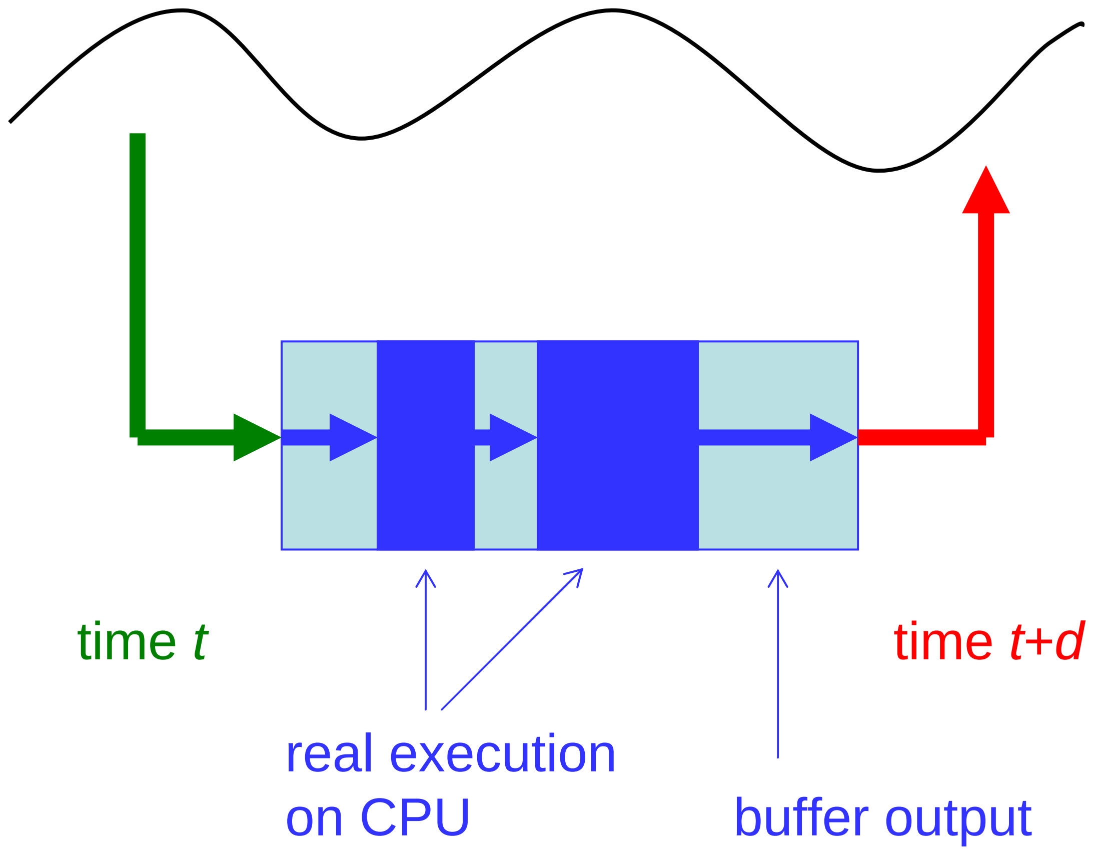
write actuator output at time *t*+*d*, for fixed *d*

The FET Programming Model

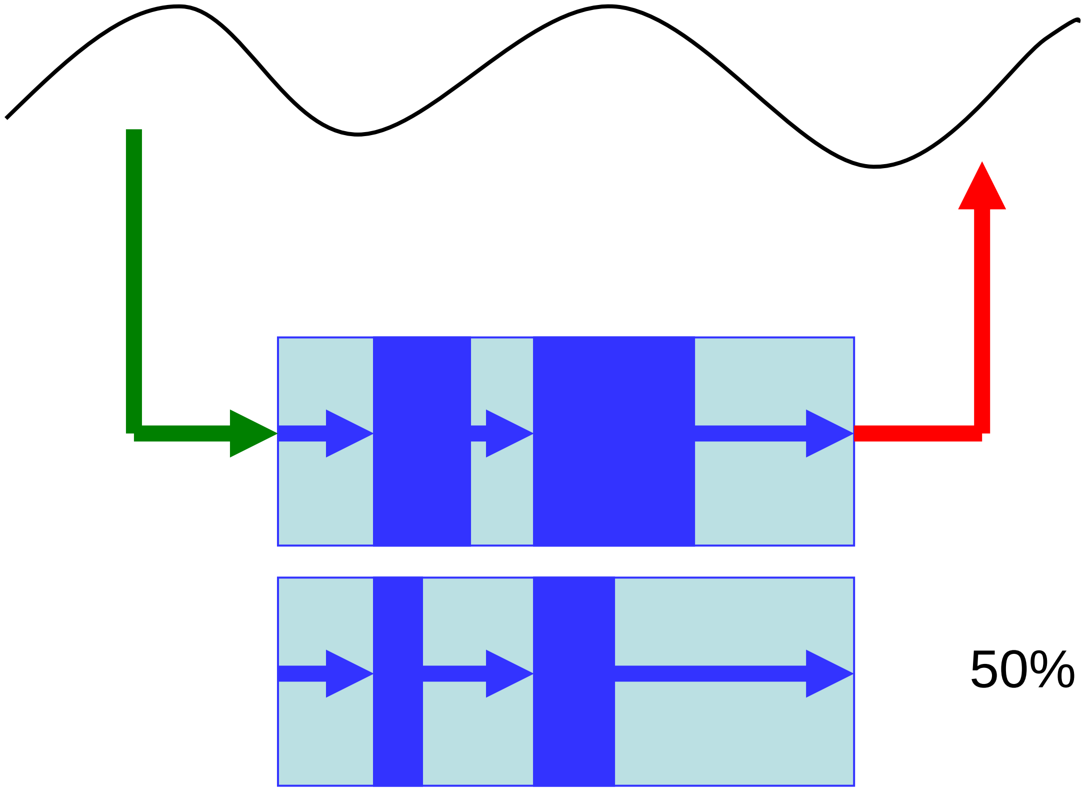
The programmer specifies *d* (could be any event) to solve the problem at hand.

The compiler ensures that *d* is met on a given platform (hardware performance and utilization); otherwise it rejects the program.

The FET (Fixed Execution Time) Assumption

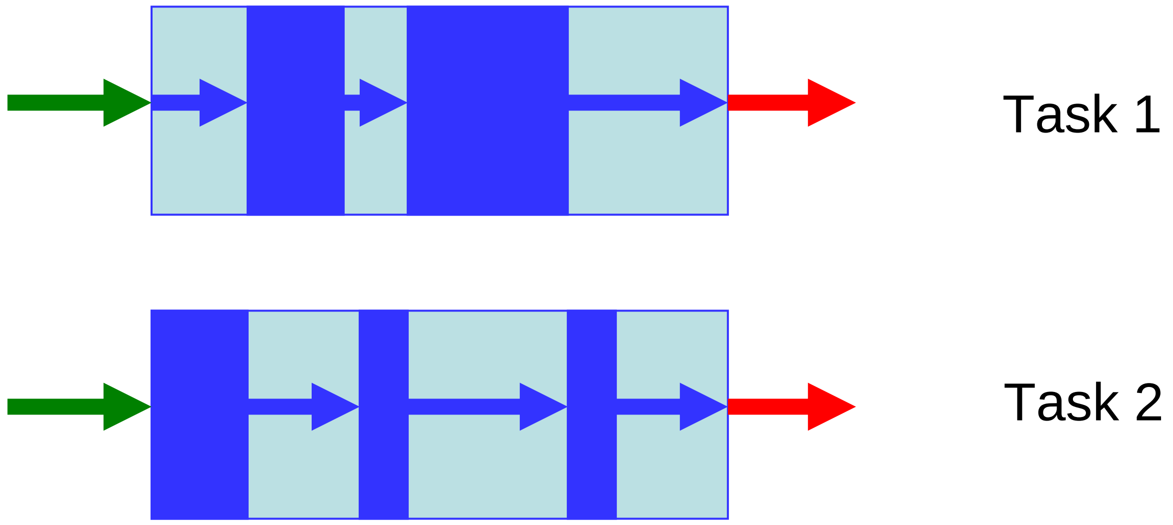


Portability

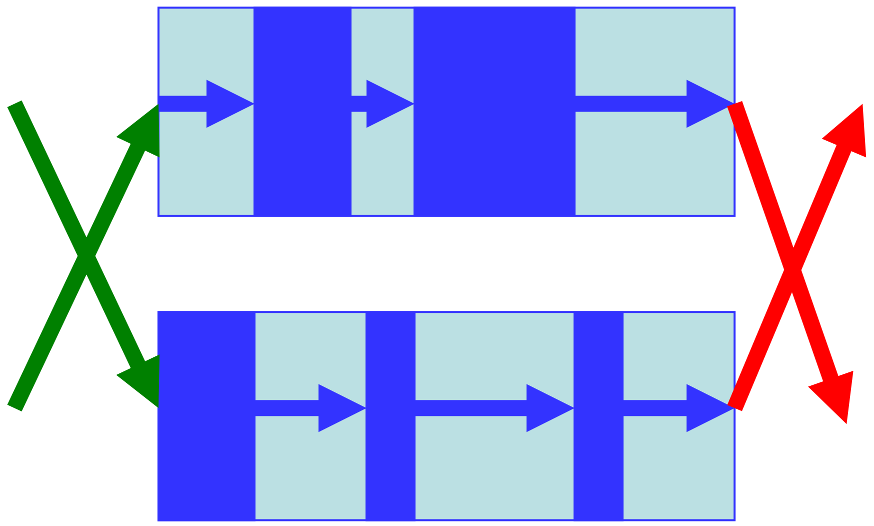


50% CPU speedup

Composability

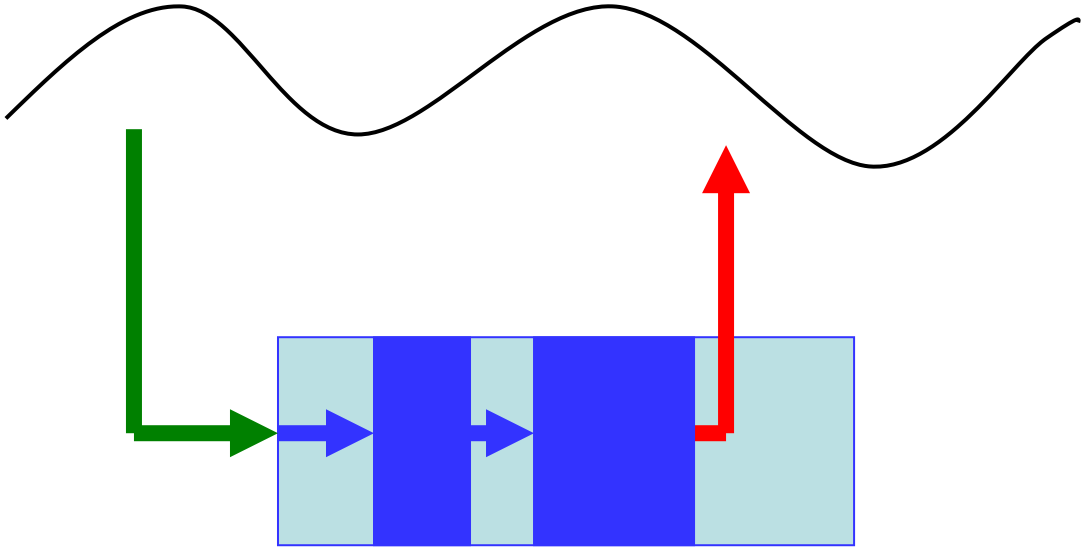


Verifiability through Predictability (Internal Determinism)



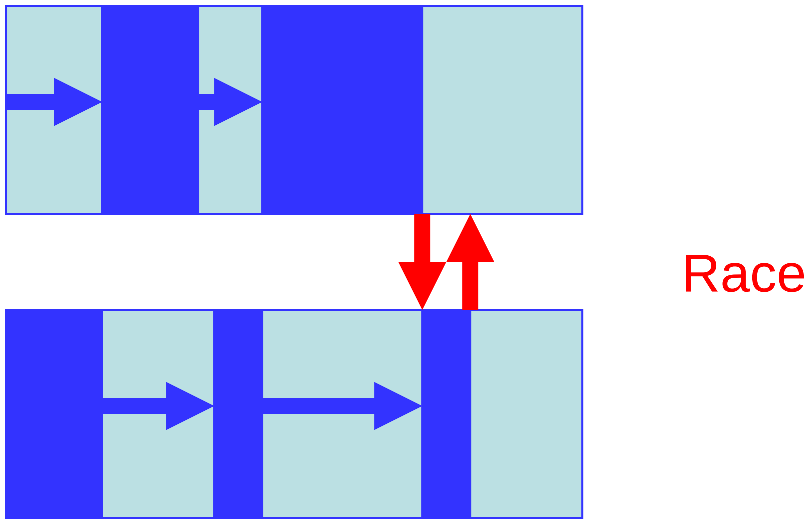
Timing predictability:	minimal jitter
Function predictability:	no race conditions

Contrast FET with Standard Practice



make output available
as soon as ready

Contrast FET with Standard Practice



Some Examples

1 Heterogeneity through hybrid automata

2 Continuity through discounting

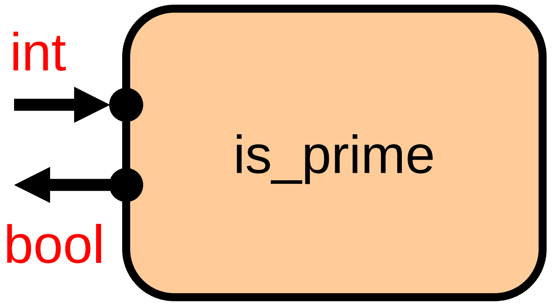
3 Noninterference through fixed logical execution times

4 Compositionality through automaton interfaces

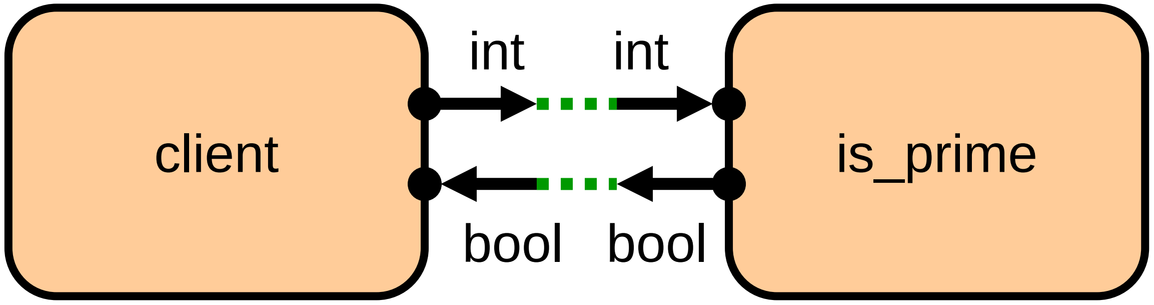
A Signature Interface

This interface constrains the client's data.

E.g. typed programming languages.



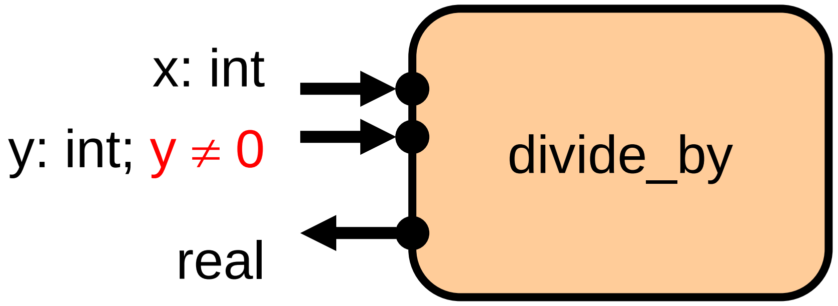
Signature Interface Compatibility



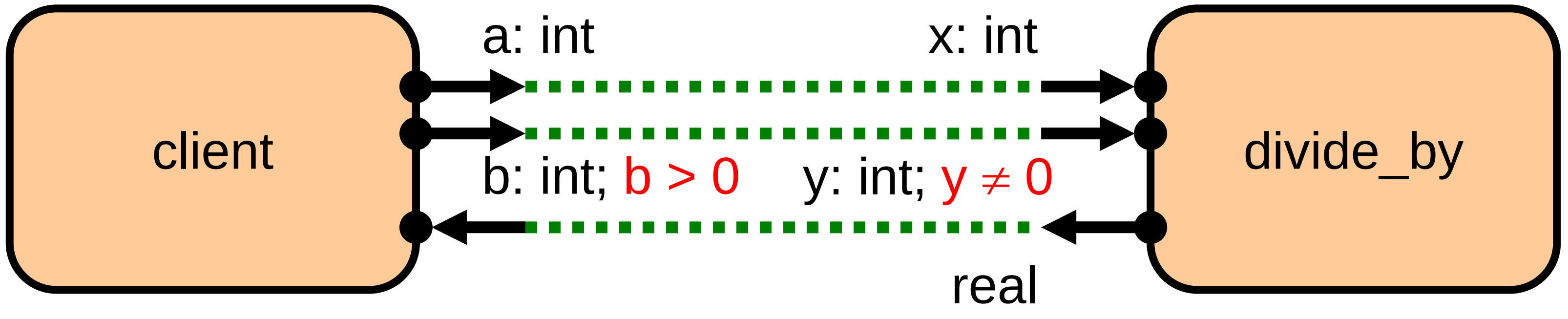
An Assertional Interface

This interface
still constrains
the client's data.

E.g. extended
static checking.

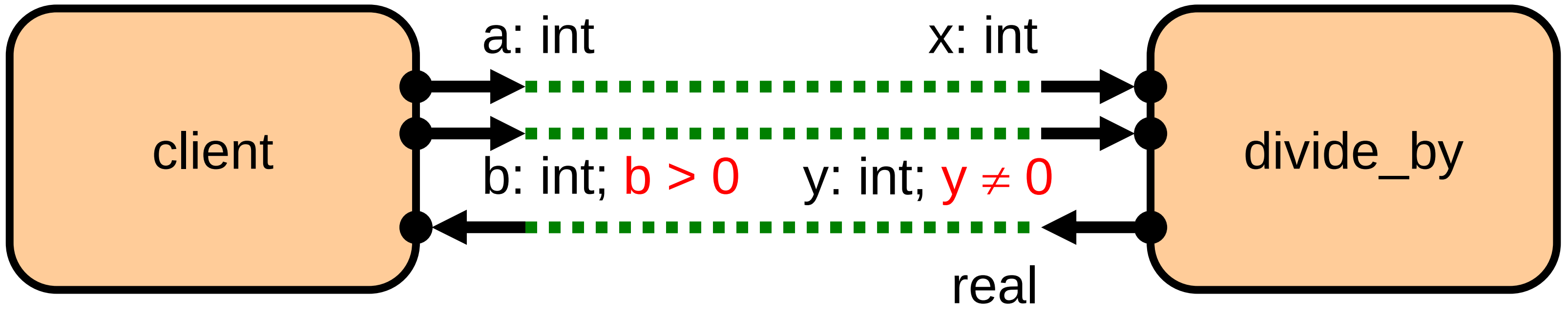


Assertional Interface Compatibility



8 b,y. (b > 0 Æ y = b) y ≠ 0)

Assertional Interface Compatibility

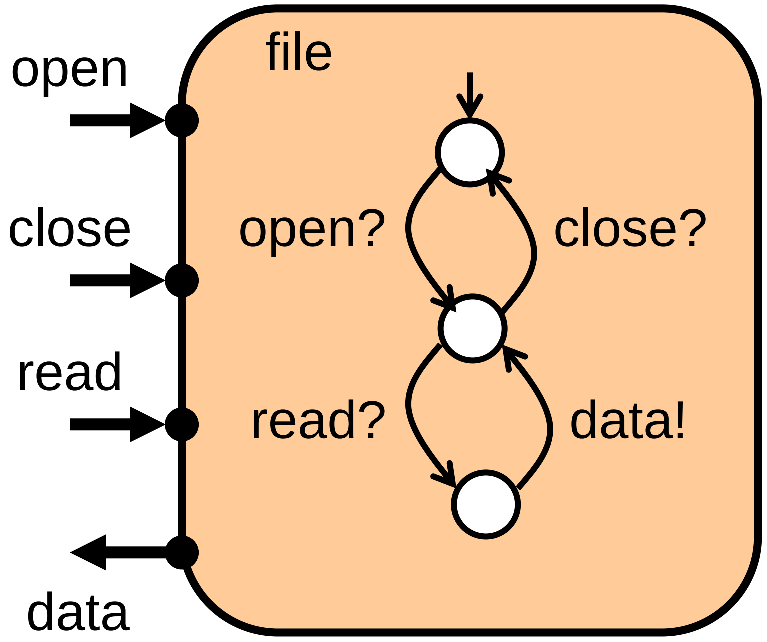


8 b,y. (b > 0 Æ y = b) y ≠ 0)

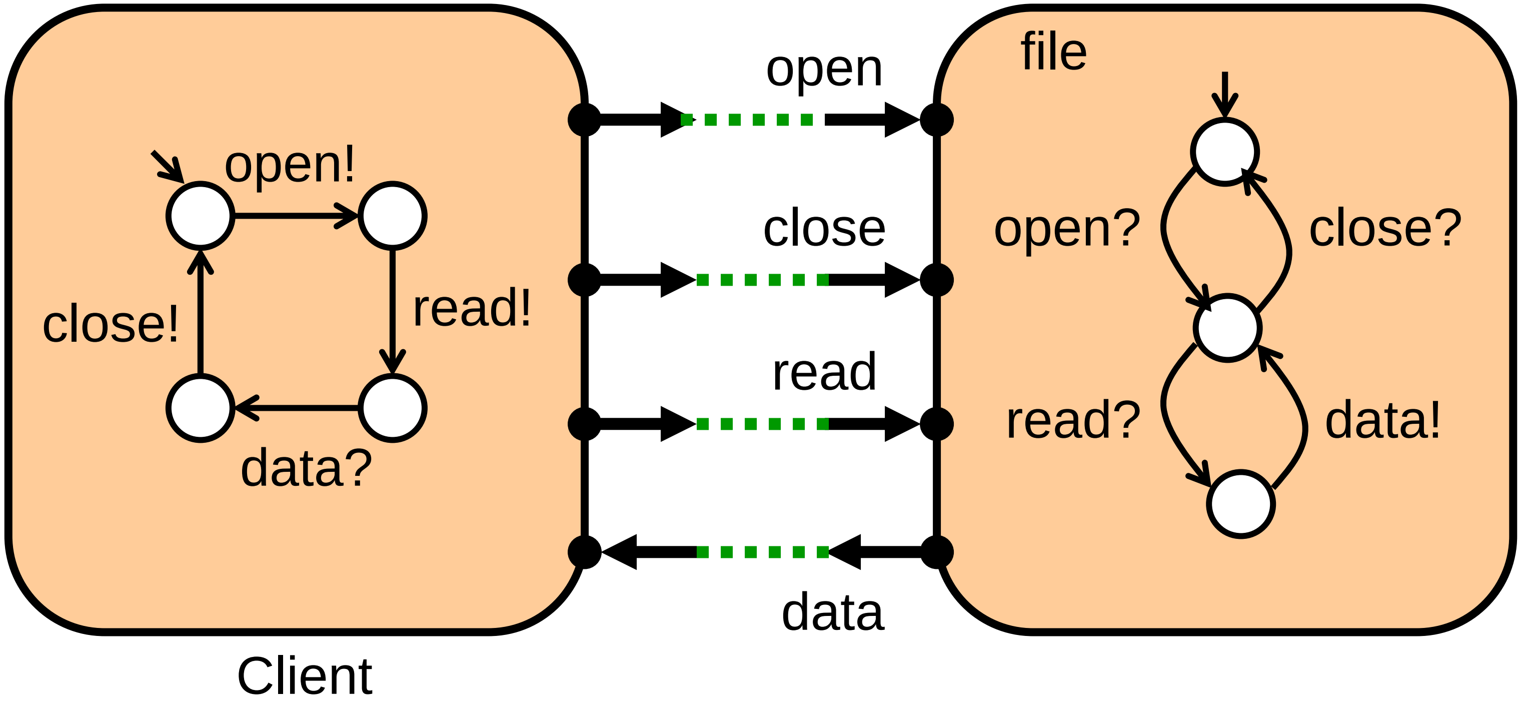
Preconditions are assumptions on the input.
Postconditions are guarantees on the output.

An Automaton Interface

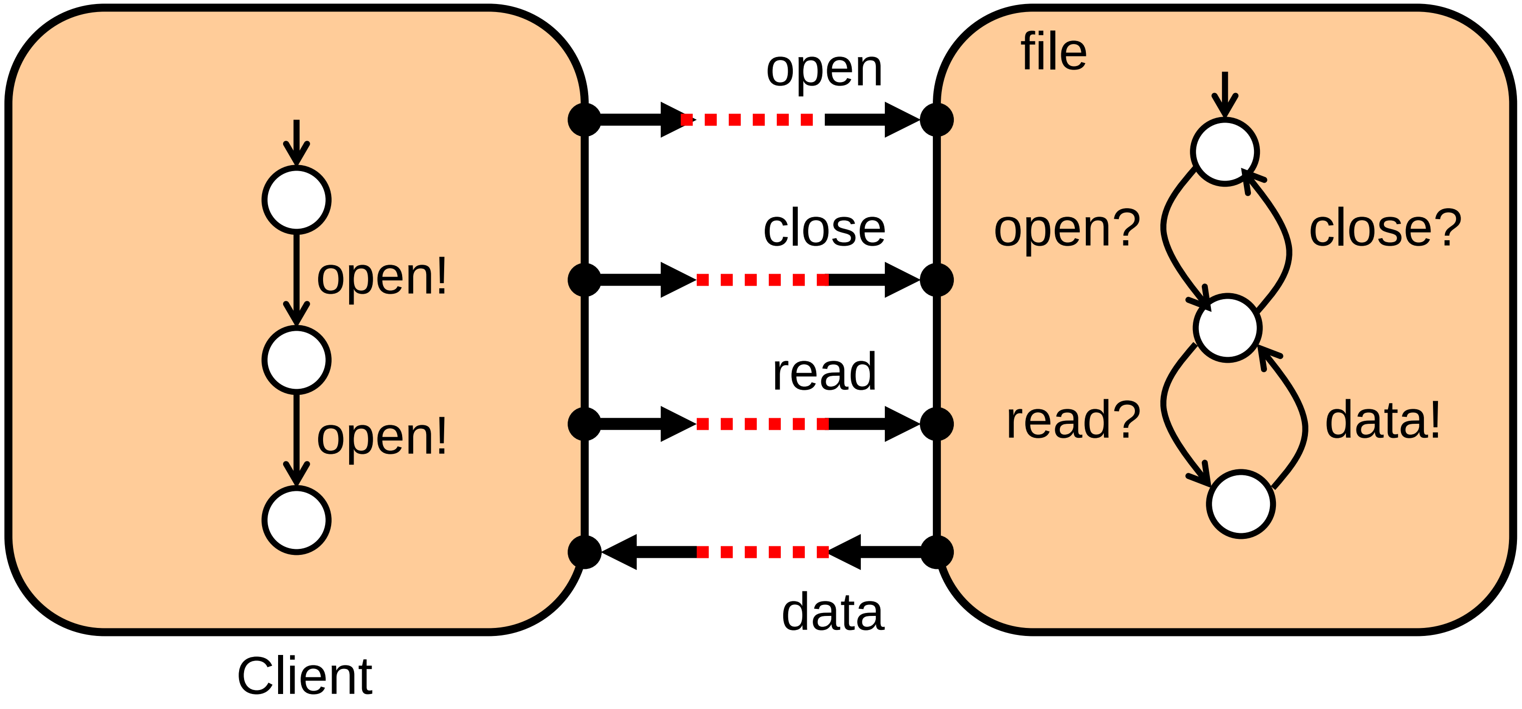
This interface constrains the client's control.



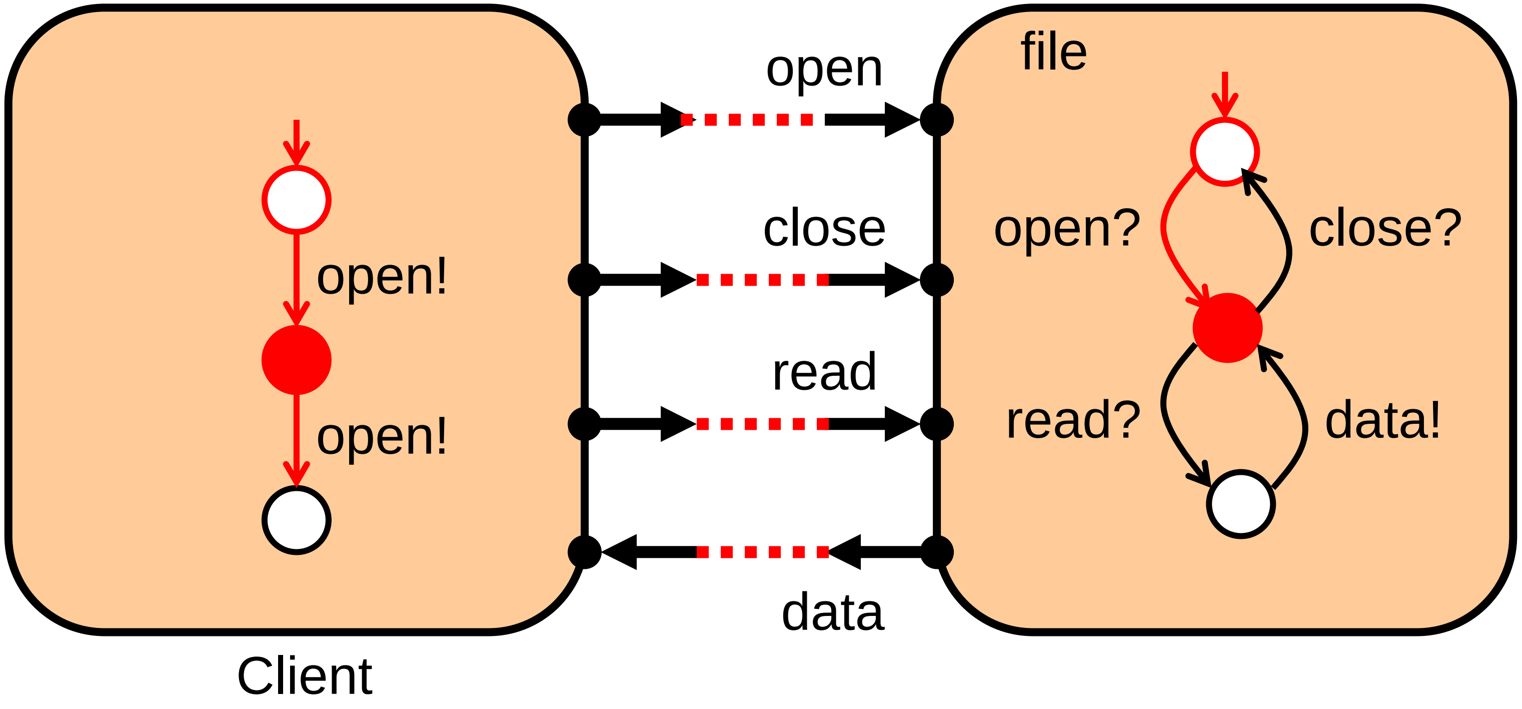
Automaton Interface Compatibility



Automaton Interface Incompatibility



Automaton Interface Incompatibility



Summary

Verifying properties is not an end but a mean.
The end is designing reliable systems.

The challenge is to come up with a formal foundation for systems design that lets us quantify how the effort spent during design relates to the quality (functionality, performance, robustness) of the product.

Credits

Hybrid Automata: R. Alur, P.-H. Ho, J. Sifakis, et al.

Discounting: L. de Alfaro, R. Majumdar, et al.

Giotto: B. Horowitz, C. Kirsch, et al.

Interfaces: A. Chakrabarti, L. de Alfaro, et al.