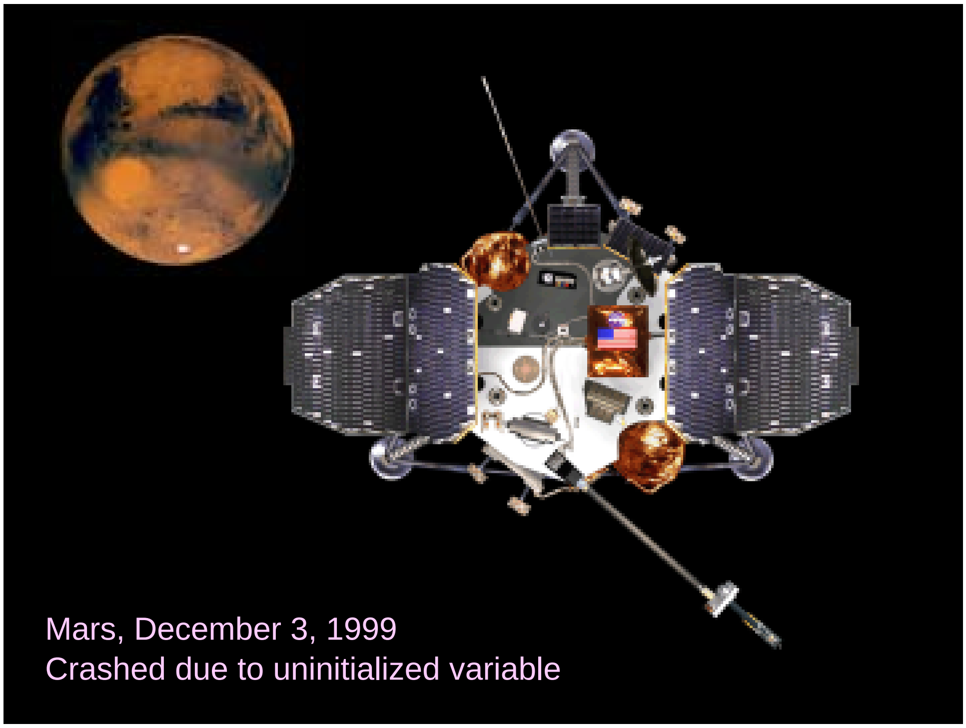
Hybrid Systems:
From Models to Code

Tom Henzinger
UC Berkeley

UC Berkeley: Chess
Vanderbilt University: ISIS
University of Memphis: MSI

Foundations of Hybrid and Embedded Software Systems


Mars, December 3, 1999
Crashed due to uninitialized variable


BOEING 777-300
$4 billion development effort
40-50% system integration & validation cost

Sources of Complexity

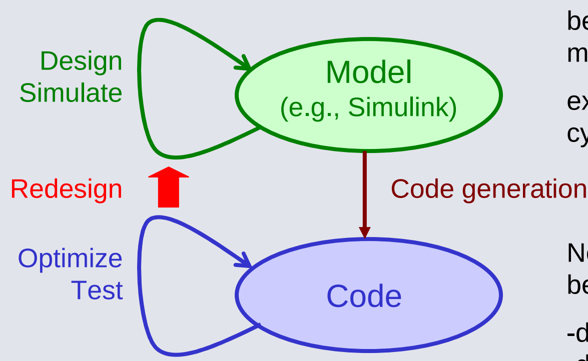
-concurrency

-real time

-heterogeneity

A **hybrid system** consists of multiple continuous (physical) and discrete (computational) components that interact with each other in real time.

Embedded Software Design: Current State



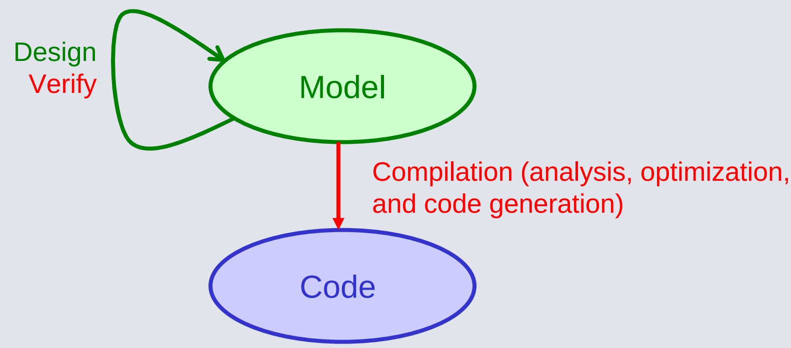
No formal connection between requirements, model, and resources:

expensive development cycle iterates all stages

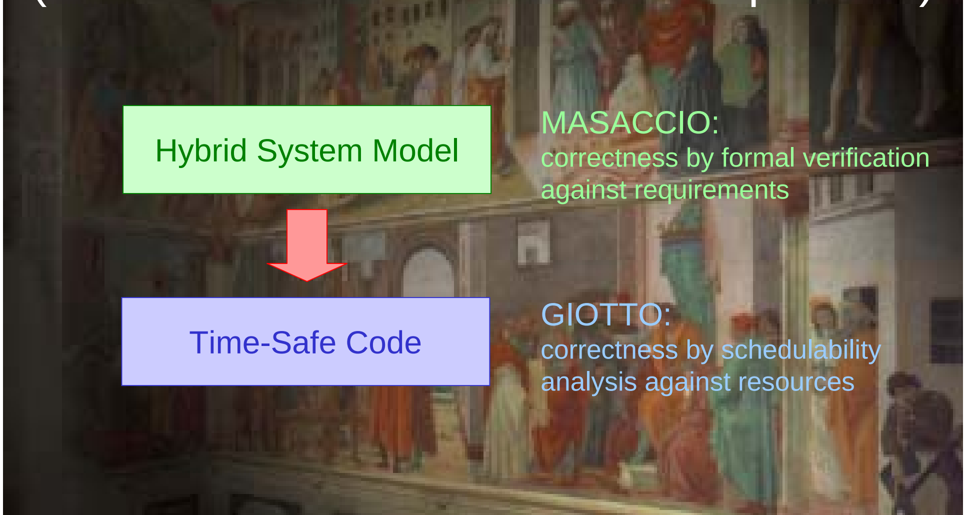
No exact correspondence between model and code:

-difficult to upgrade code
-difficult to reuse code

Embedded Software Design: Our Vision

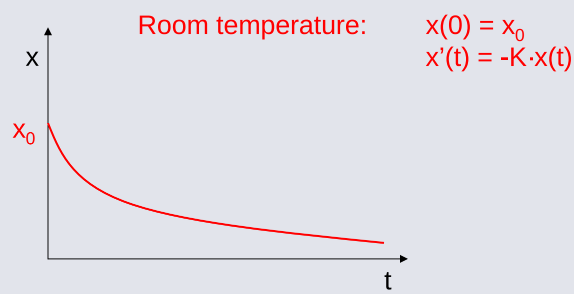


The FRESCO Project
(Formal Real-Time Software Components)



Continuous (Euclidean) Systems

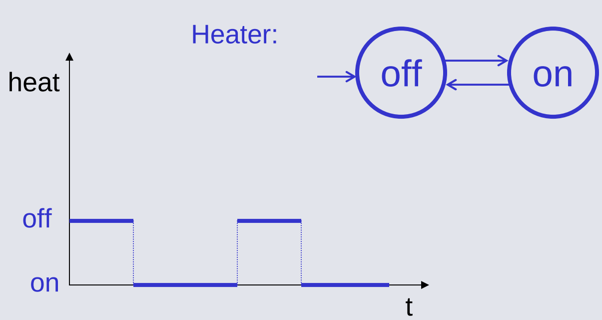
State space: R^n
Dynamics: initial condition + differential equations



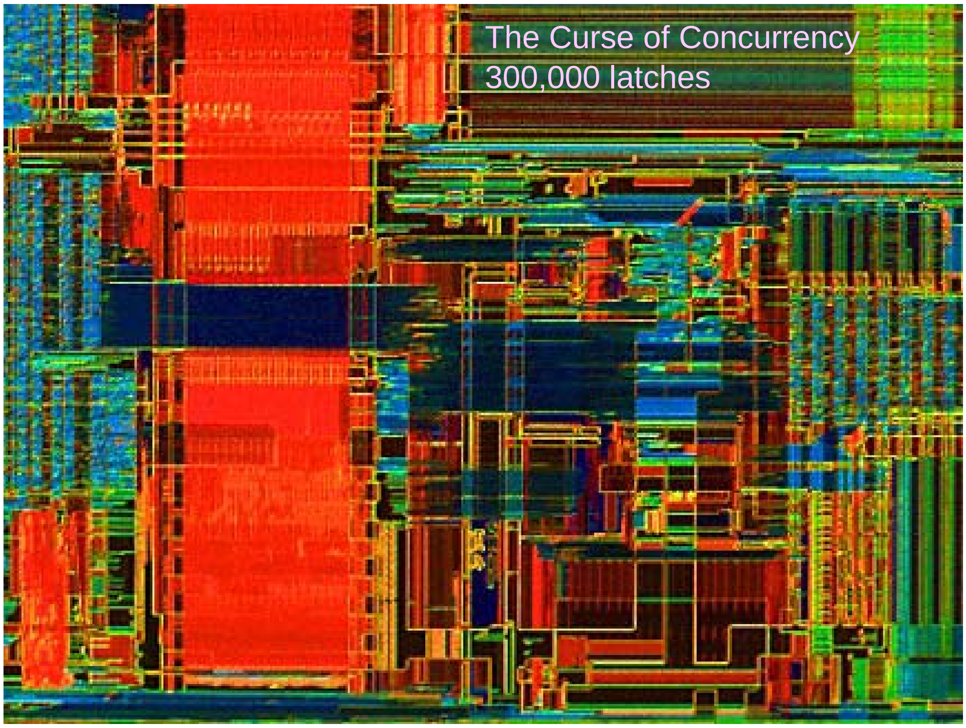
Analytic complexity.

Discrete (Boolean) Systems

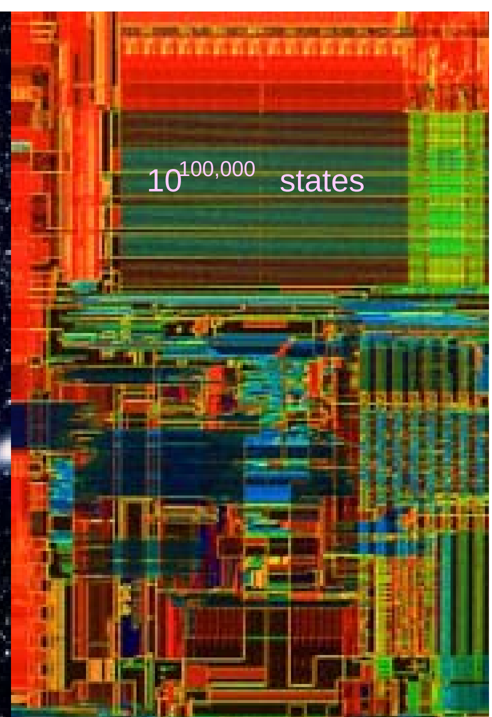
State space: B^m
Dynamics: initial condition + transition relation



Combinatorial complexity.


The Curse of Concurrency
300,000 latches

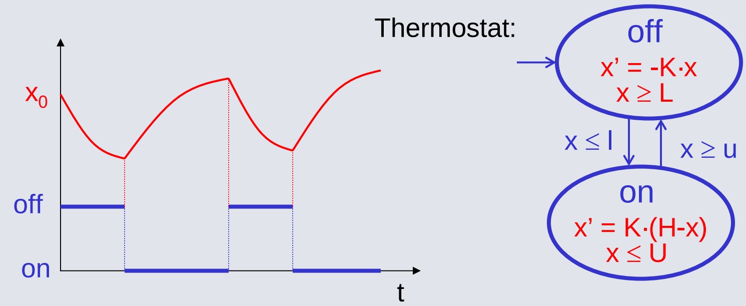

10^{11} stars


$10^{100,000}$ states

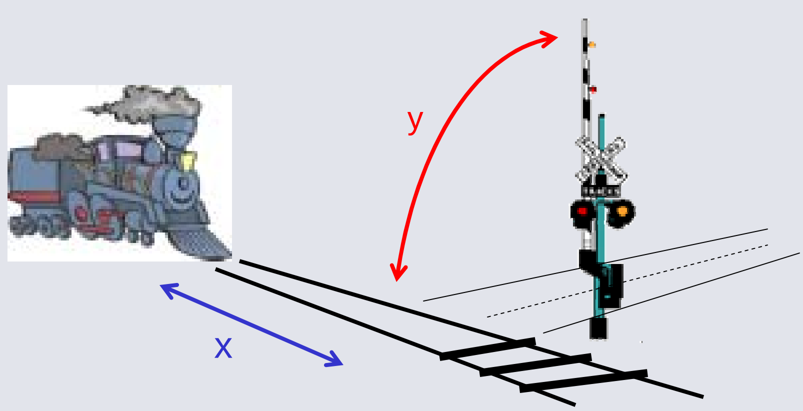
Hybrid Systems

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

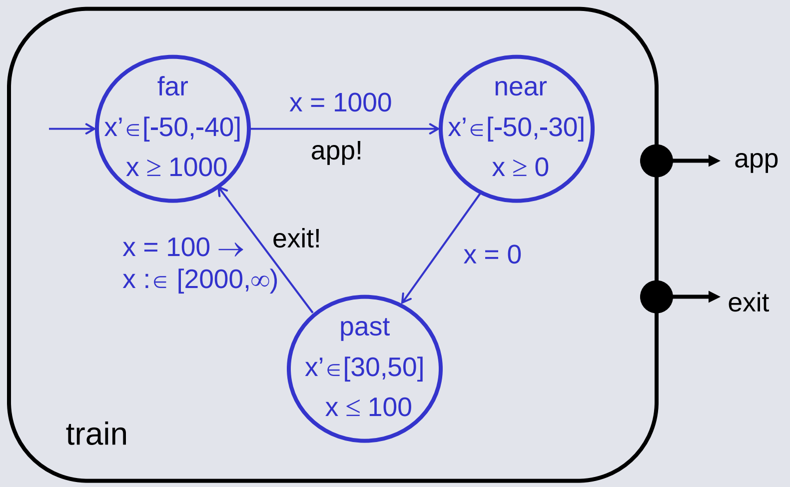
Dynamics: initial condition + transition relation + differential equations



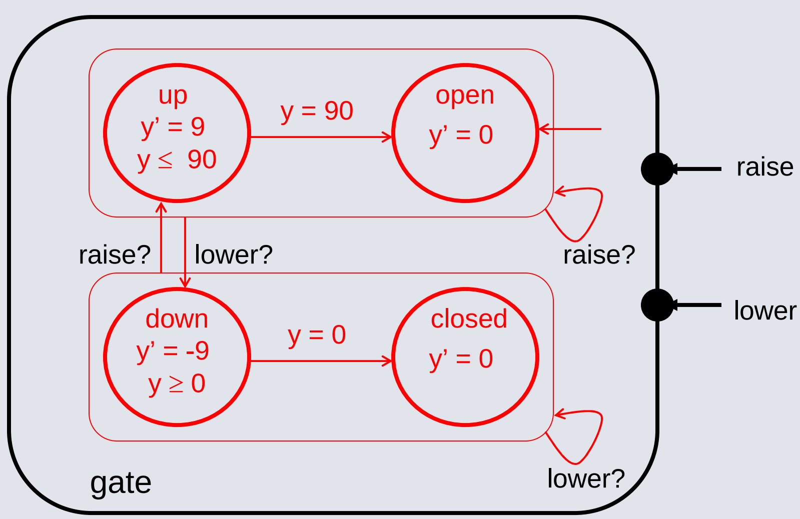
Hybrid Automata



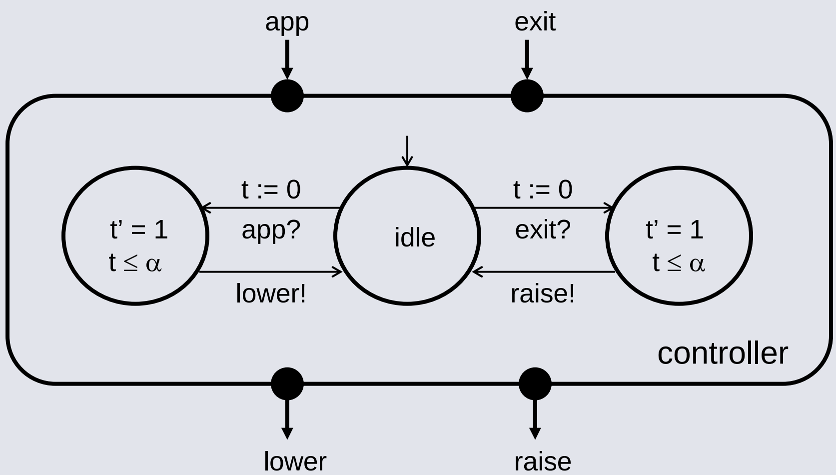
Hybrid Automata



Hybrid Automata



Hybrid Automata



Requirements

Safety: $\forall\Box\ (x \leq 10 \Rightarrow loc[gate] = closed)$

Liveness: $\forall\Box\ \forall\Diamond\ (loc[gate] = open)$

Real time: $\forall\Box\ z := 0.\ (z' = 1 \Rightarrow$
$\forall\Diamond\ (loc[gate] = open \wedge z \leq 60))$

Verification and failure analysis by model checking (e.g., HyTech).

Two Problems with Hybrid Automata

1. Scalability

Possible solutions:

- hierarchy (MASACCIO)
- assume-guarantee decomposition (interfaces)

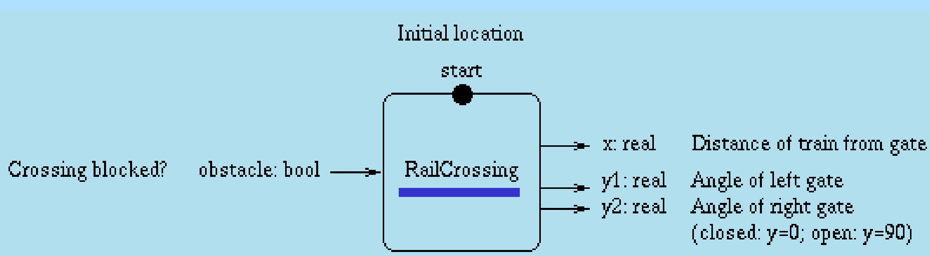
2. Robustness

Possible solutions:

- ε-variability
- discounted future

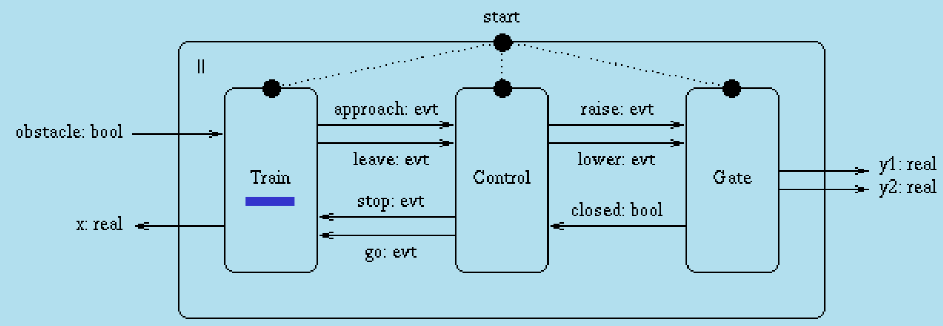
MASACCIO

Hierarchical Hybrid Automata



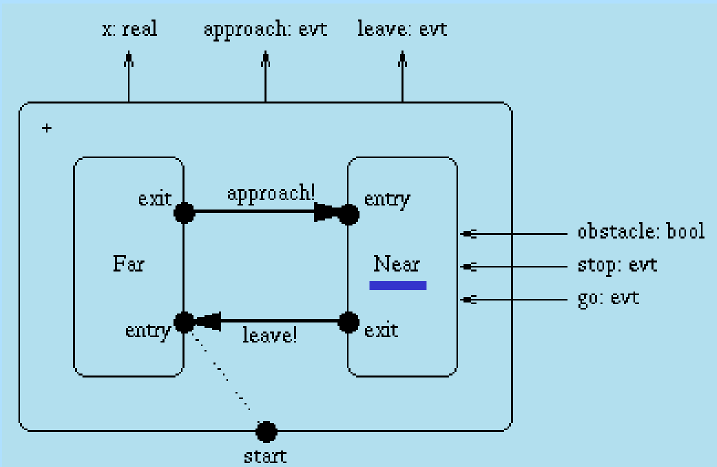
MASACCIO

Hierarchical Hybrid Automata



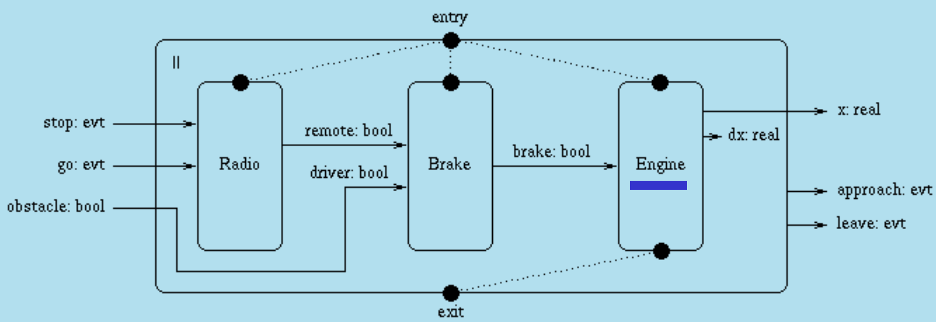
MASACCIO

Hierarchical Hybrid Automata

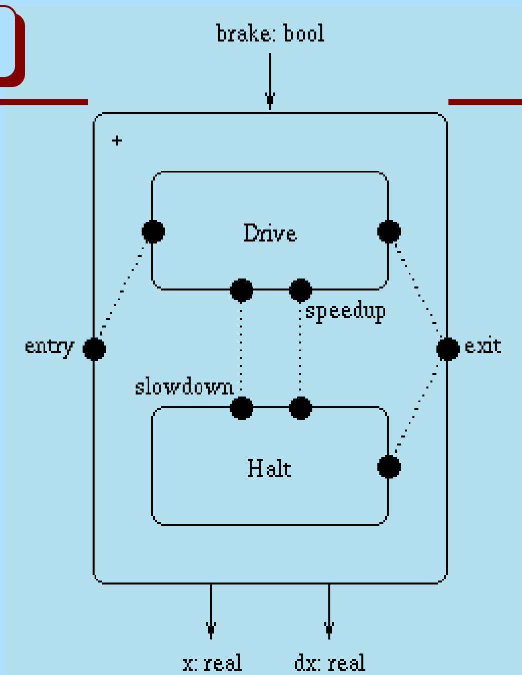


MASACCIO

Hierarchical Hybrid Automata



MASACCIO



Two Problems with Hybrid Automata

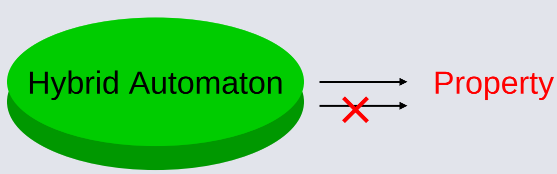
1. Scalability

   Possible solutions:
   - hierarchy (MASACCIO)
   - assume-guarantee decomposition (interfaces)

2. Robustness

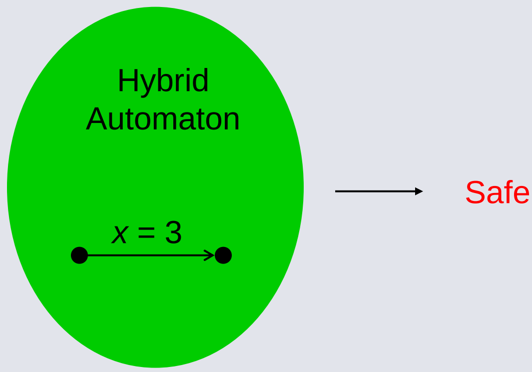
   Possible solutions:
   - ε-variability
   - discounted future

The Robustness Problem

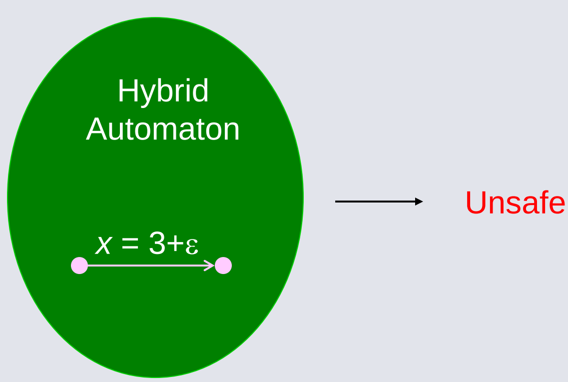


slightly perturbed automaton

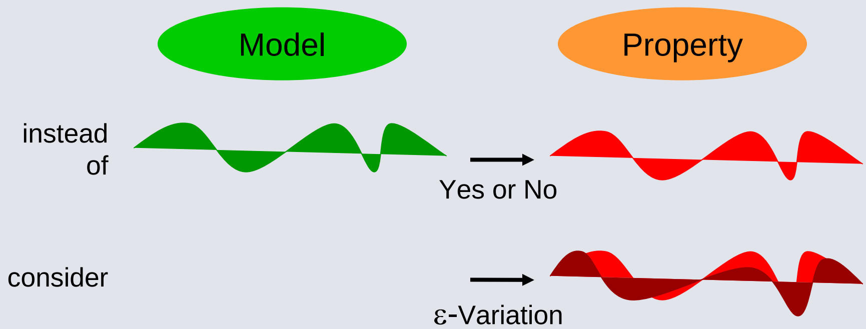
The Robustness Problem



The Robustness Problem



A Possible Solution of the Robustness Problem: Metrics on Traces



A More Radical Solution of the Robustness Problem: Discounting the Future

value(Model,Property): States → {Yes, No}

⇓

value(Model,Property): States → R

A More Radical Solution of the Robustness Problem: Discounting the Future

value(Model,Property): States $\rightarrow$ {Yes, No}

value(m,$\Diamond T$) = $\mu X.\ (T \vee \mathrm{pre}(X))$

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

discountedValue(m,$\Diamond T$) = $\mu X.\ \max(T, \lambda\cdot\mathrm{pre}(X))$

discount factor $0<\lambda<1$

A More Radical Solution of the Robustness Problem: Discounting the Future

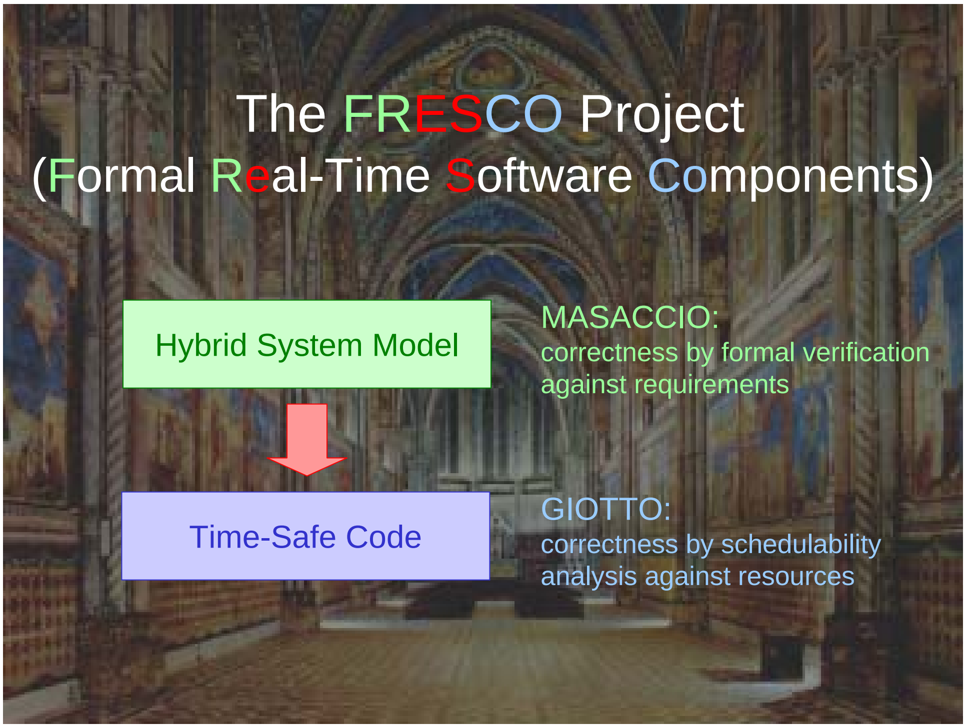
Robustness Theorem:

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)



The History of Computer Science: Lifting the Level of Abstraction

High-level languages: Programming to the application — Requirements focused code

↓ Compilation

The "assembly age": Programming to the platform — Resource focused code

-Traditional high-level languages abstract time.

-This abstraction is unsuitable for real-time applications, which are still programmed in terms of platform time ("priority tweaking").

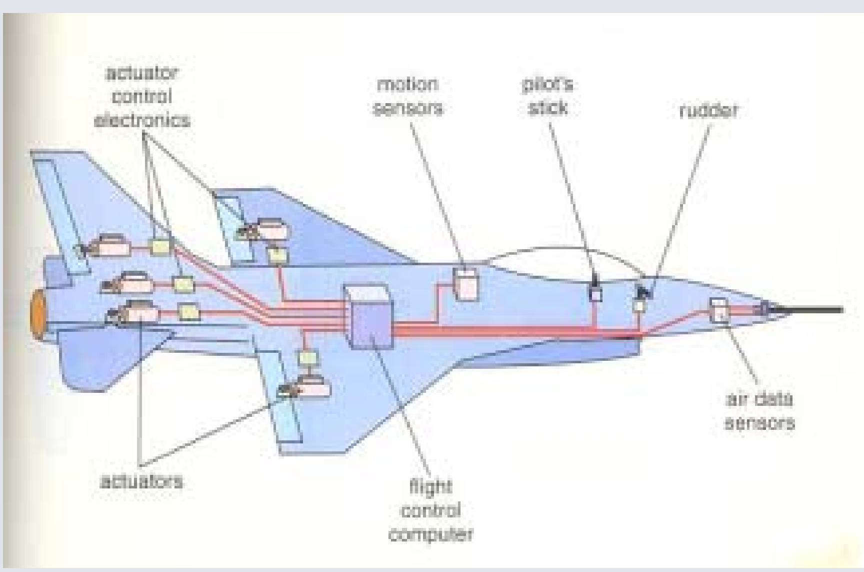
-GIOTTO: Real-time programming in terms of application time.

GIOTTO

Time-Triggered Programming

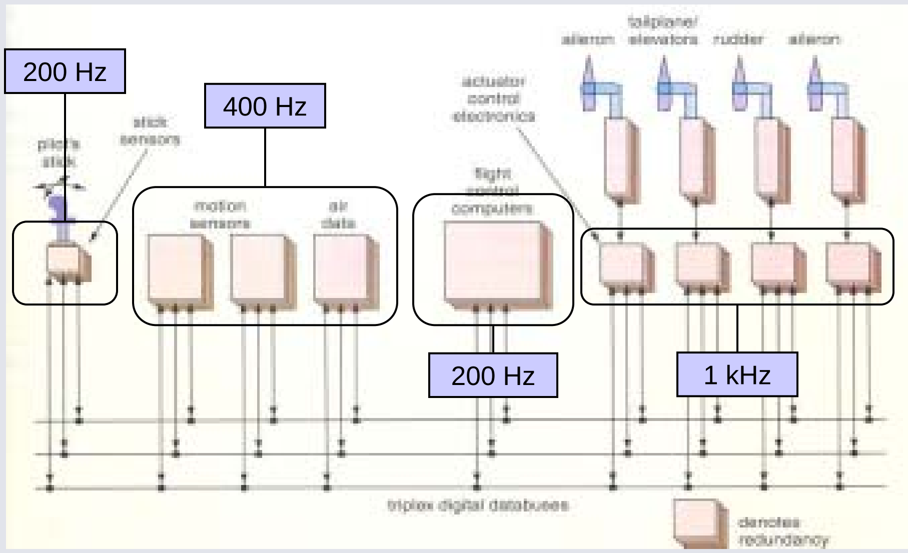
GIOTTO

Time-Triggered Programming



GIOTTO

Time-Triggered Programming



GIOTTO

Time-Triggered Programming

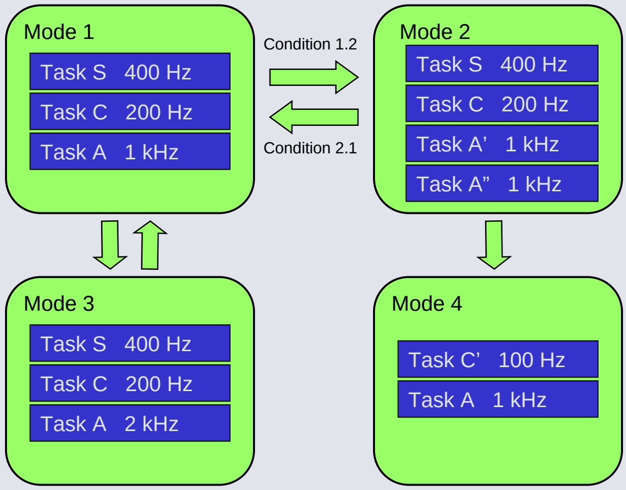
1. Concurrent Periodic Tasks:

-sensing
-control law computation
-actuating

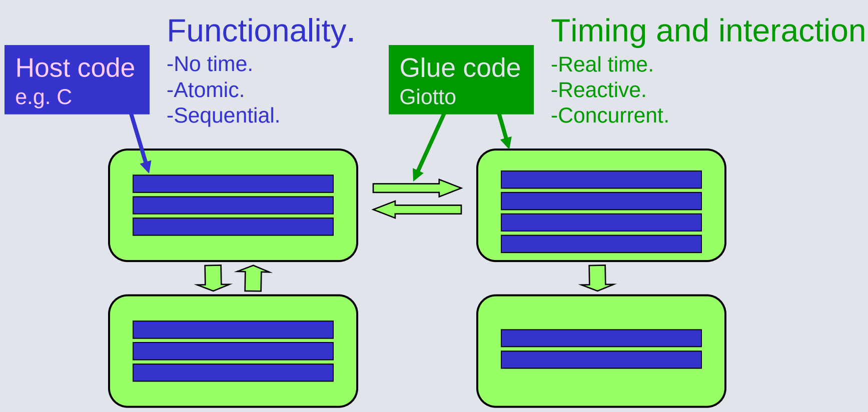
2. Multiple Modes of Operation:

-navigational modes (autopilot, manual, etc.)
-maneuver modes (taxi, takeoff, cruise, etc.)
-degraded modes (sensor, actuator, CPU failures)

Time-Triggered Programming



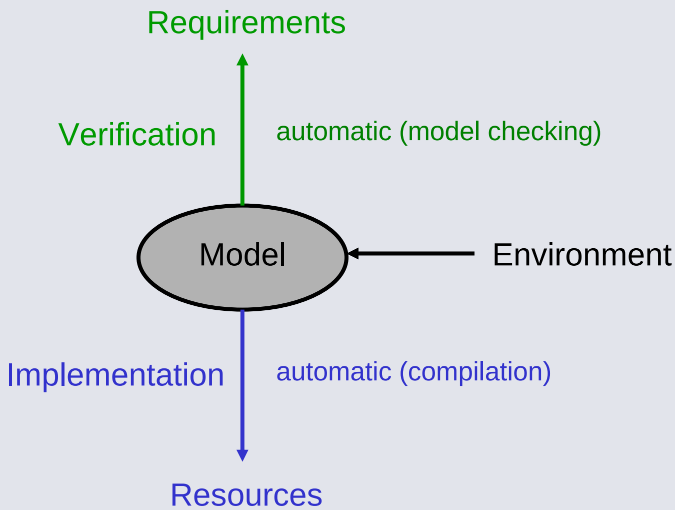
Separation of Concerns



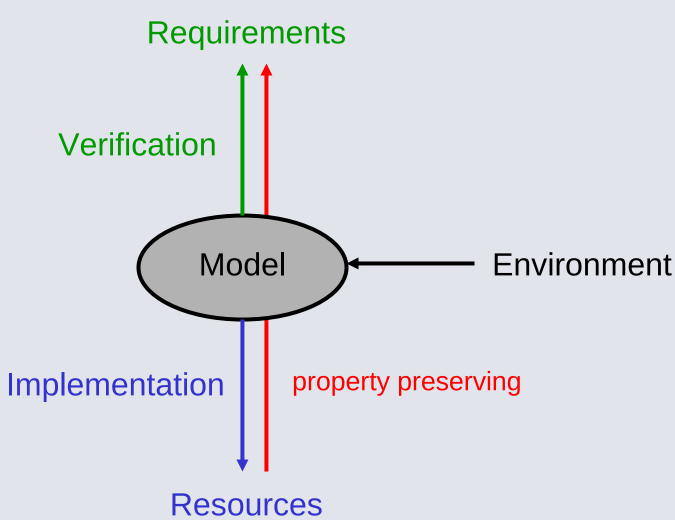
This kind of software is reasonably well understood.

The software complexity lies in the glue code.

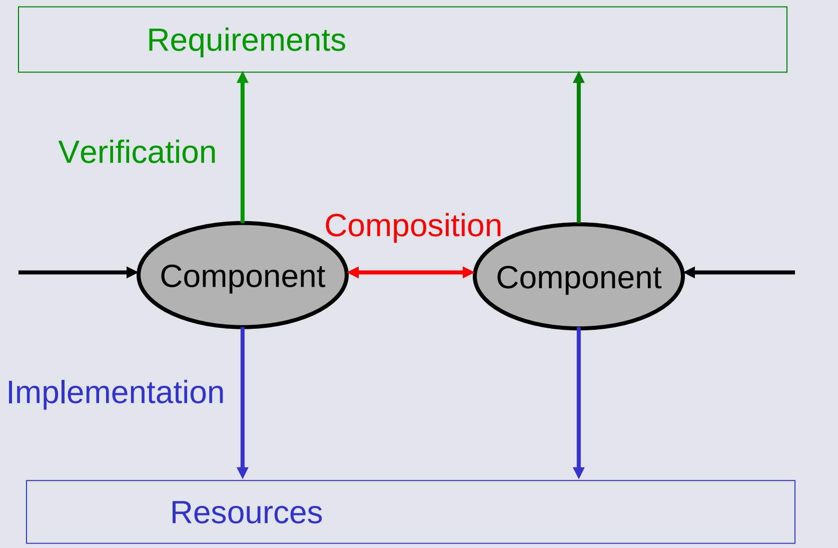
Two Opposing Forces



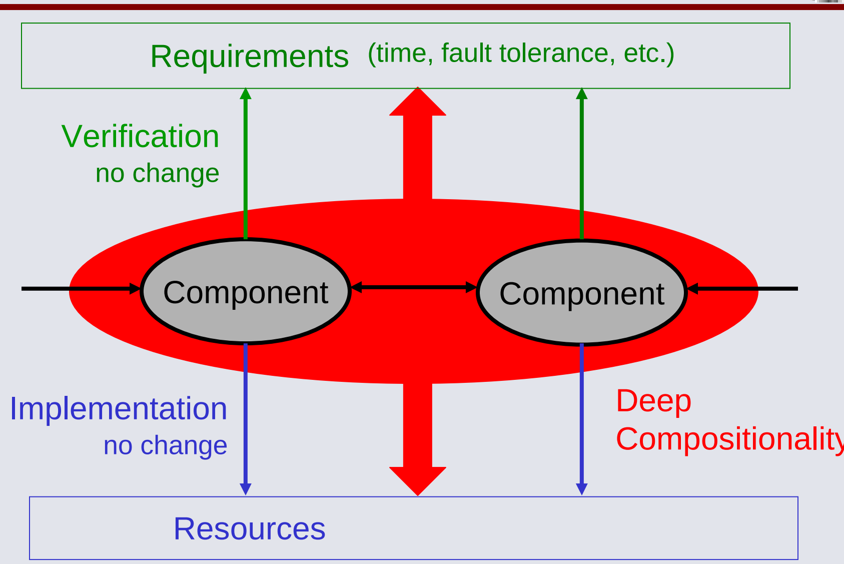
Two Opposing Forces



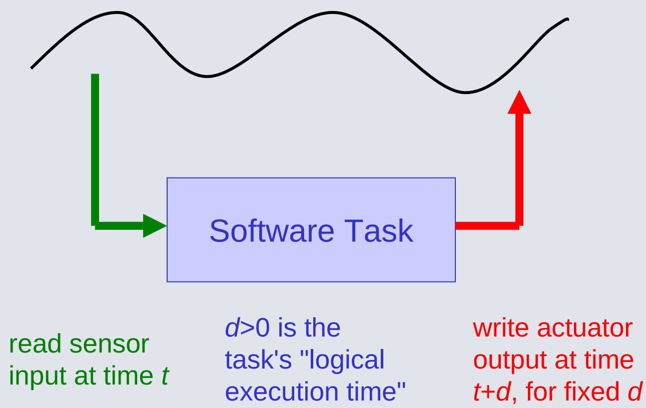
Two Opposing Forces



Two Opposing Forces



Achieving Verifiability and Compositionality in GIOTTO: The FLET (Fixed Logical Execution Time) Assumption

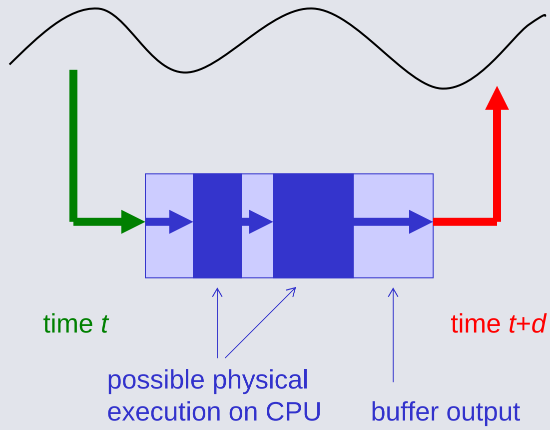


Embedded Programming in GIOTTO

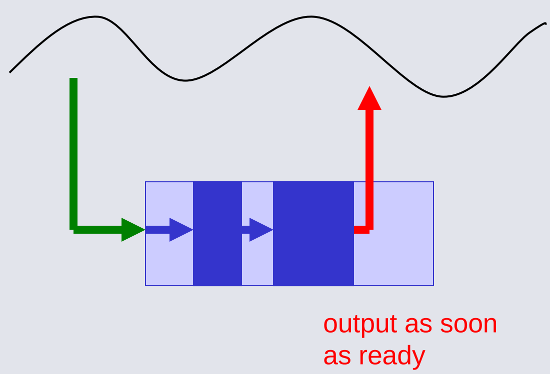
The programmer specifies sample rate *d* and jitter *j* to solve the control problem at hand.

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

Implementing the FLET Assumption



Contrast the FLET with Standard Practice



Advantages of the FLET and GIOTTO

-predictable timing and value behavior
(no internal race conditions, minimal jitter)

-portable, composable code (as long as the platform offers sufficient performance)

Research Agenda

From Hybrid Models

-robust hybrid models (tube topologies, discounting)

-model checking for hierarchical and stochastic hybrid models

-multi-aspect assume-guarantee decomposition of hybrid models (interface theories for time, resources, fault tolerance)

To Embedded Code

-distributed schedulability analysis and code generation

-on-line code modification and fault tolerance

Credits

Scalable and Robust Hybrid Systems: Luca de Alfaro, Arkadeb Ghosal, Marius Minea, Vinayak Prabhu, Marcin Jurdzinski, Rupak Majumdar

GIOTTO: Ben Horowitz, Christoph Kirsch, Rupak Majumdar, Slobodan Matic, Marco Sanvido

Collaborators of the FRESCO Project

-Alex Aiken on time-safety analysis of embedded code

-Karl Hedrick on Giotto implementation of electronic throttle control

-Edward Lee on Giotto modeling and code generation in Ptolemy

-Edward Lee on rich interface theories as type theories for component interaction

-George Necula on model checking device drivers

-George Necula on scheduler-carrying embedded code

-Alberto Sangiovanni-Vincentelli on synthesis of protocol converters from interfaces

-Alberto Sangiovanni-Vincentelli and Shankar Sastry on platform-based design of a helicopter flight control system using Giotto

-Shankar Sastry on hybrid automata