Environment Modeling: A Usability Challenge for Verifying Cyber-Physical Systems

Sanjit A. Seshia

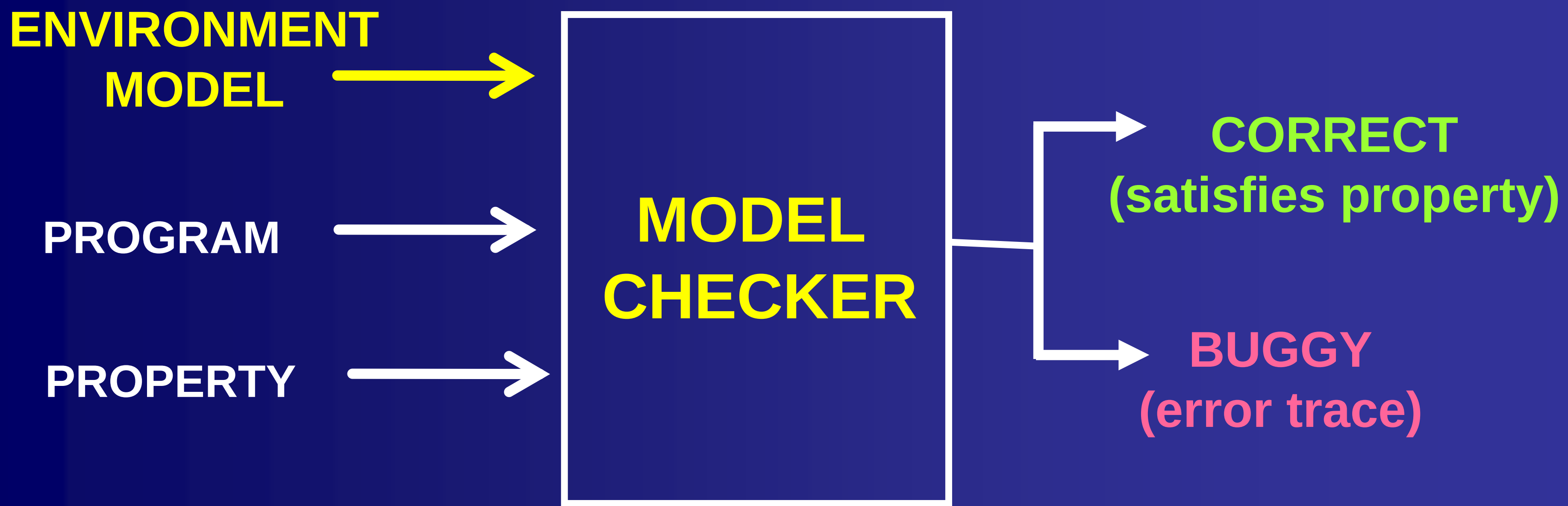
EECS Department

UC Berkeley

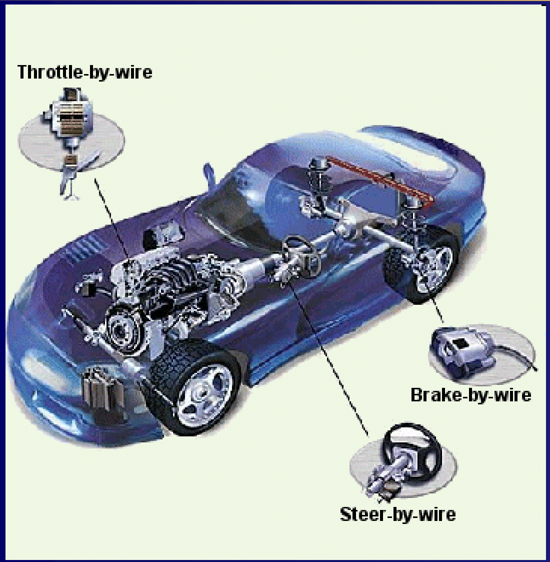
Workshop on Usable Verification

November 2010

Typical Verification Picture: Is it complete?!

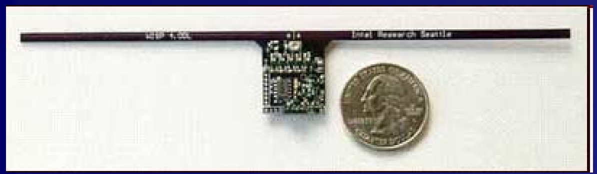


Quantitative Analysis / Verification



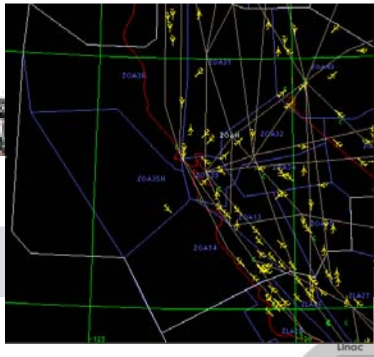
Does the brake-by-wire software always actuate the brakes within 1 ms?
Safety-critical embedded systems

Can this new app drain my iPhone battery in an hour?
Consumer devices



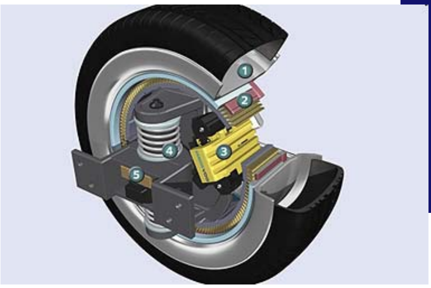
How much energy must the sensor node harvest for RSA encryption?
Energy-limited sensor nets, bio-medical apps, etc.

Cyber-Physical Systems (CPS): *Orchestrating networked computation with physical systems*



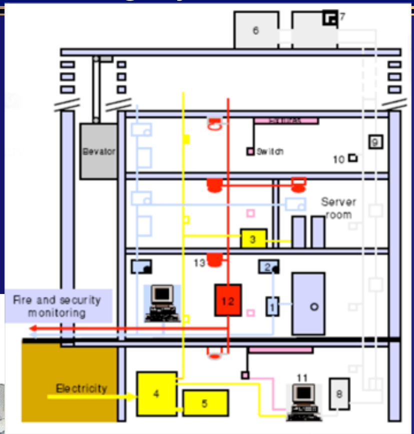
Transportation (Air traffic control at SFO)

Automotive

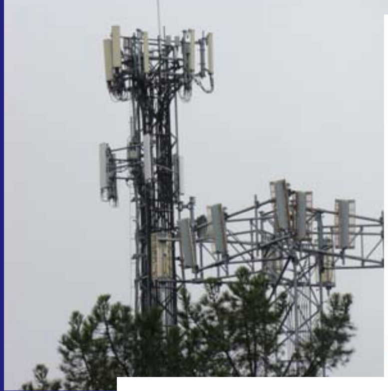


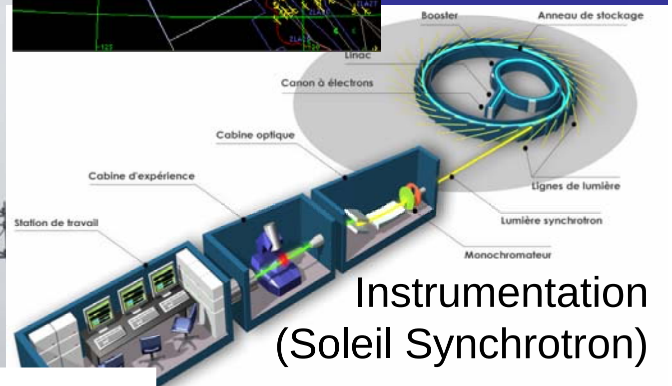
E-Corner, Siemens

Building Systems

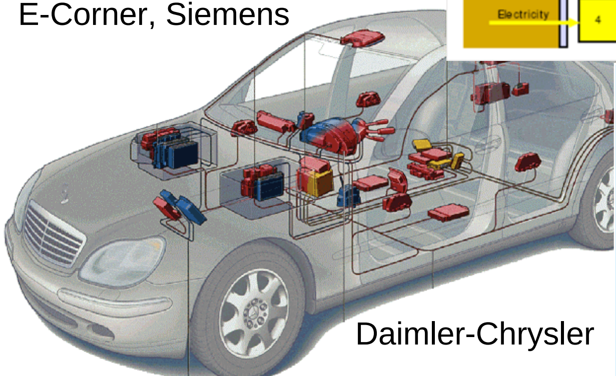


Telecommunications



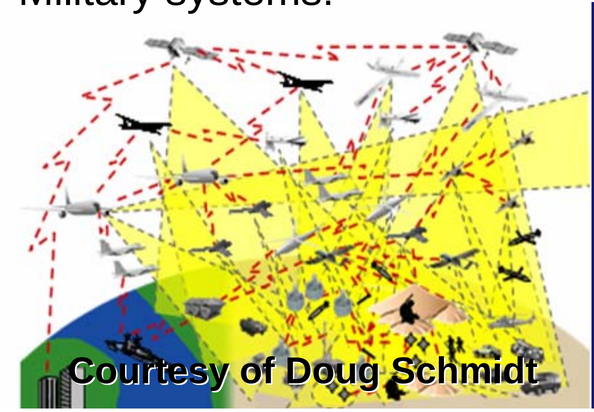


Instrumentation (Soleil Synchrotron)



Daimler-Chrysler

Military systems:



Courtesy of Doug Schmidt

Power generation and distribution

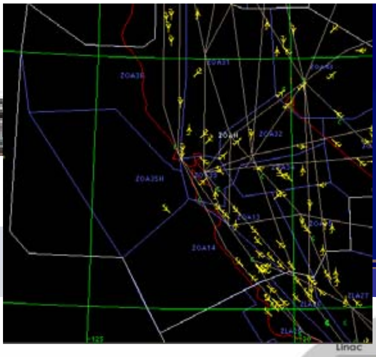
Courtesy of General Electric

Factory automation

Courtesy of Kuka Robotics Corp.

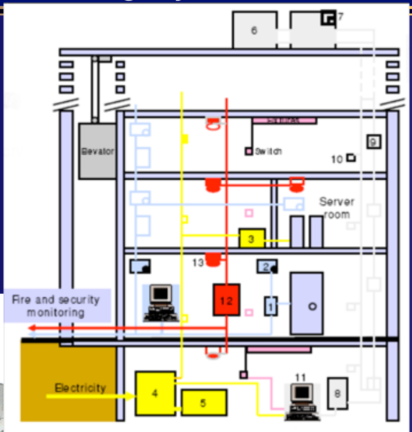
[E.A.Lee]

Cyber-Physical Systems (CPS): *Orchestrating networked computation with physical systems*

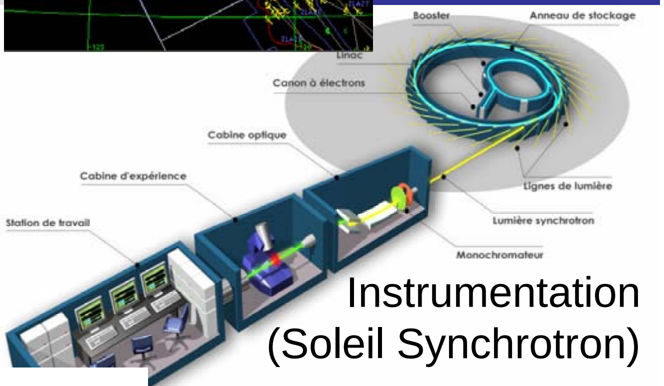


Transportation (Air traffic control at SFO)

Building Systems

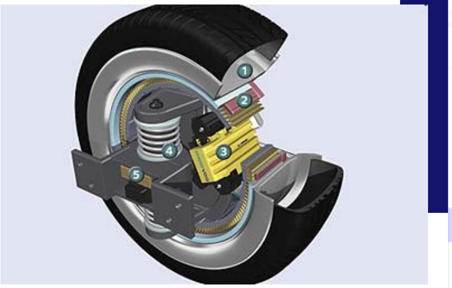


Telecommunications

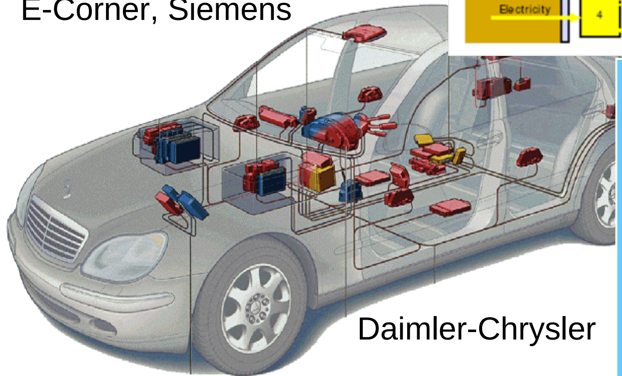


Instrumentation (Soleil Synchrotron)

Automotive



E-Corner, Siemens

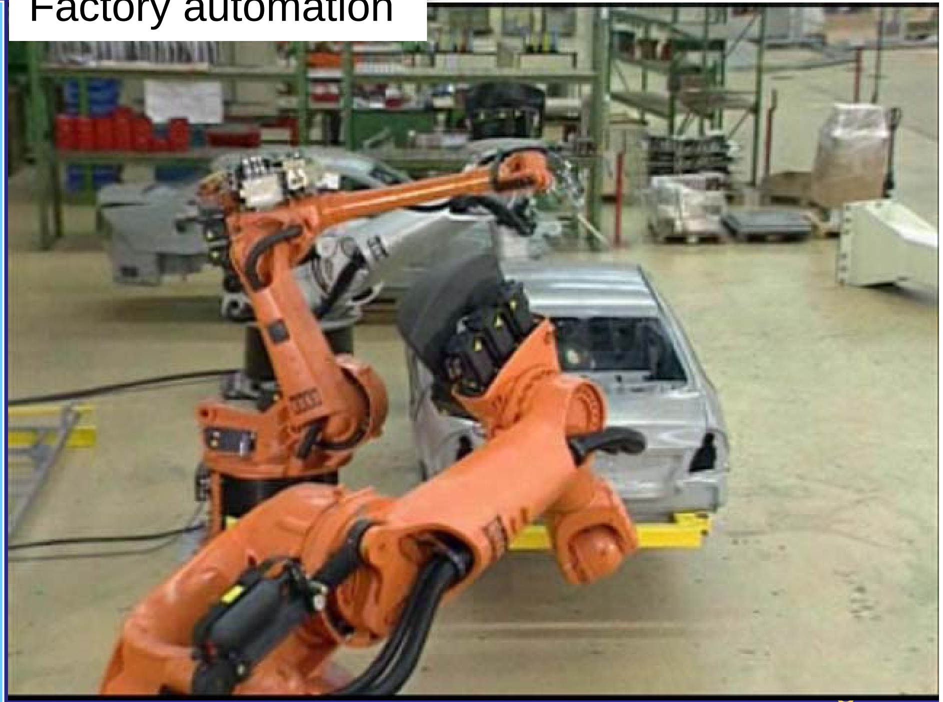


Daimler-Chrysler

Power generation and distribution

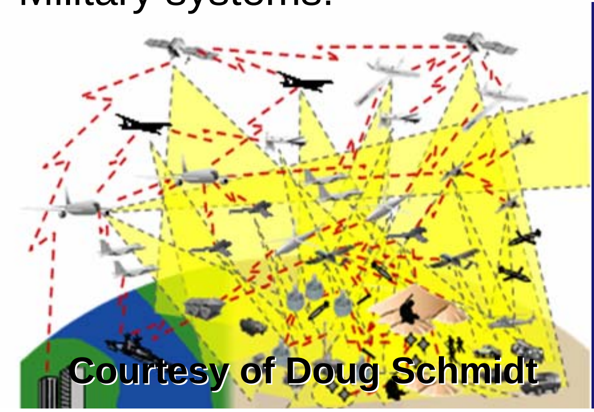
Courtesy of General Electric

Factory automation



Courtesy of Kuka Robotics Corp.

Military systems:



Courtesy of Doug Schmidt

Time is Central to Cyber-Physical Systems

Several timing analysis problems:

- Worst-case execution time (WCET) estimation
- Estimating distribution of execution times
- Threshold property: can you produce a test case that causes a program to violate its deadline?
- Software-in-the-loop simulation: predict execution time of particular program path

Challenge: Environment Modeling (Timing Analysis)

- Timing properties of the Program depend heavily on its environment
- Environment =
  - Processor & Memory Hierarchy
  - \+ Operating System, other processes/threads, …
  - \+ Network
  - \+ I/O Devices
  - \+ …
- Modeling the full environment is hard!
- However, we need a 'reasonably' precise environment model
  - – Unlike traditional software verification

Relative Success of "Boolean" Software Verification

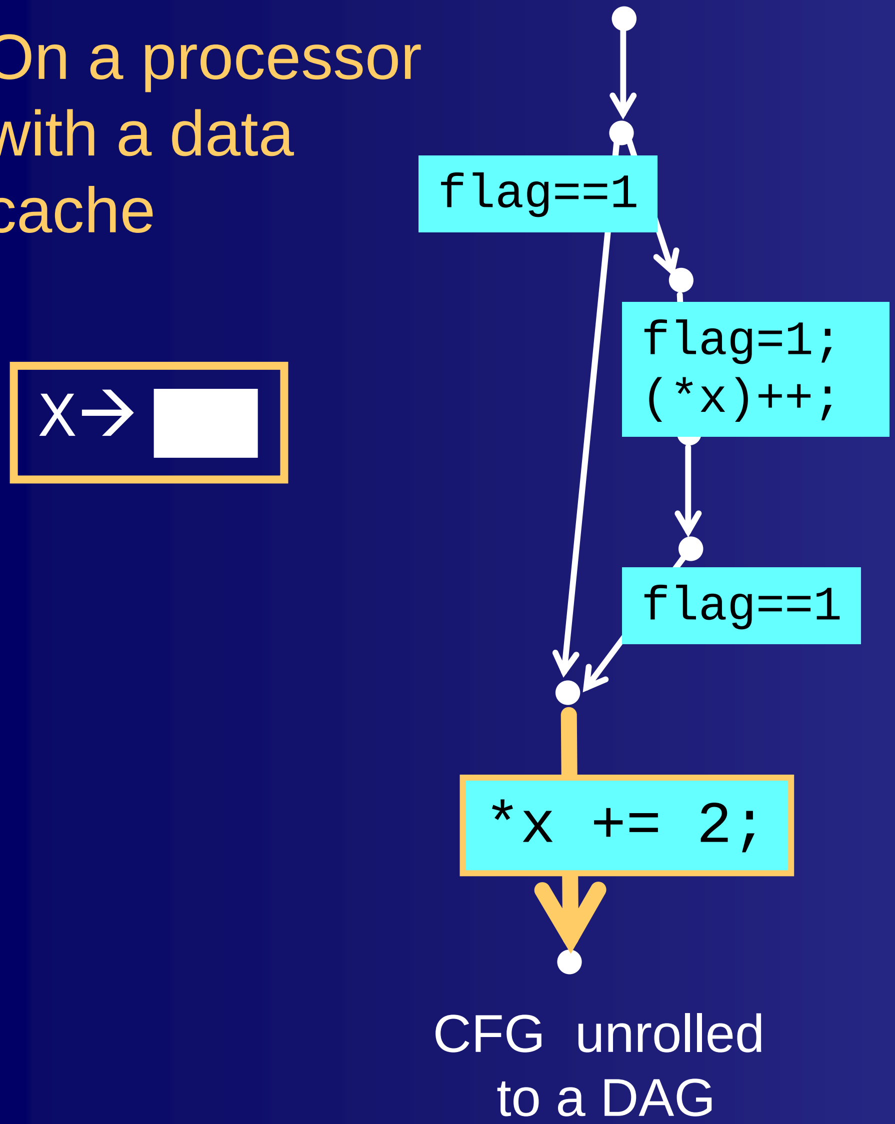
- From theoretical ideas to industrial practice in ~ 15 yrs

Some Reasons:

- Availability of open source software
- Well-defined target problems: Device drivers, memory safety, security vulnerabilities, concurrency, …
- Value of bug finding
- Coarse abstraction of environment OK

Challenge of Timing Analysis: Example

On a processor with a data cache



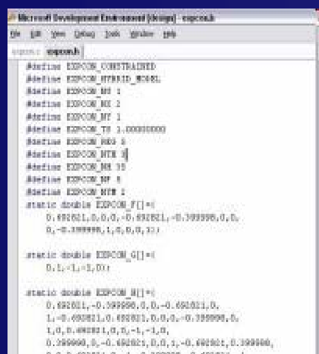
Timing of an edge (basic block) depends on:

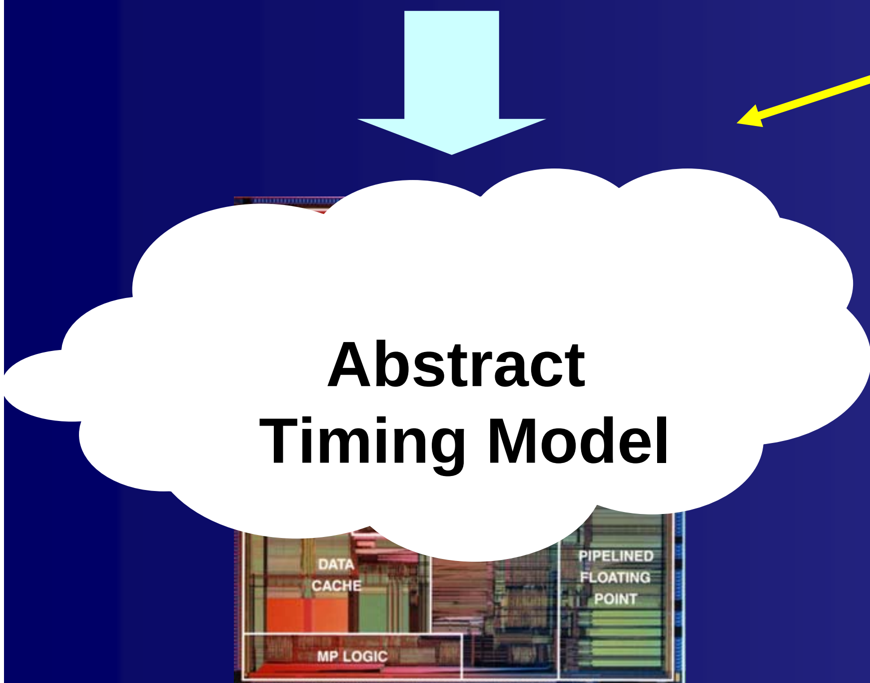
- **Program path** it lies on
- Initial **platform state**

Challenges:

- Exponential number of paths and platform states!
- Lack of visibility into platform state

Current State-of-the-art for Timing Analysis





- **Program = Sequential, terminating program**
- **Runs uninterrupted**

PROBLEM:
Can take <u>several man-months</u> to construct!

Also:
- Limited to extreme-case analysis
- Often requires additional platform specification from users

- **Platform = Simple Pipelined Processor + Data/Instruction Cache**

Existing Approaches: One-size-fits-all?

- **Why construct a SINGLE timing model for ALL programs?**
- **Only interested in analyzing a specific program.**
- **Why not automatically synthesize a program-specific timing model?**

Promising Direction
(for timing analysis and quantitative verification in general)

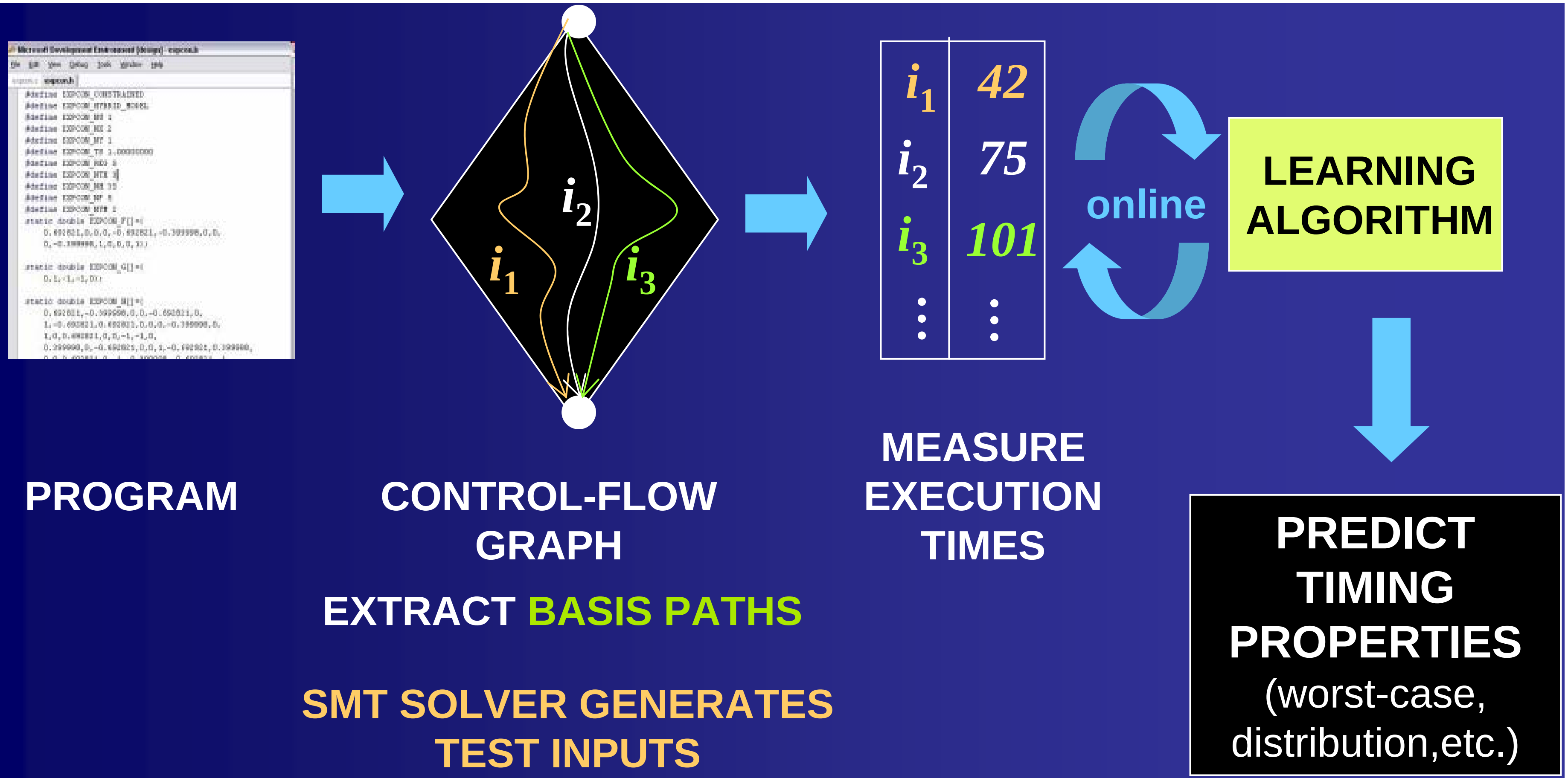
- **Inductive Synthesis**
  - **Automatically generate environment model through active learning**
- **Active = Select behaviors from which to learn**
- **Use core verification techniques (SAT, SMT, model checking, …) to generate selected behaviors**

- **Example: GameTime for timing analysis of software**

S. A. Seshia and A. Rakhlin, “Quantitative Analysis of Systems Using Game-Theoretic Learning”, ACM Trans. Embedded Computing Systems.

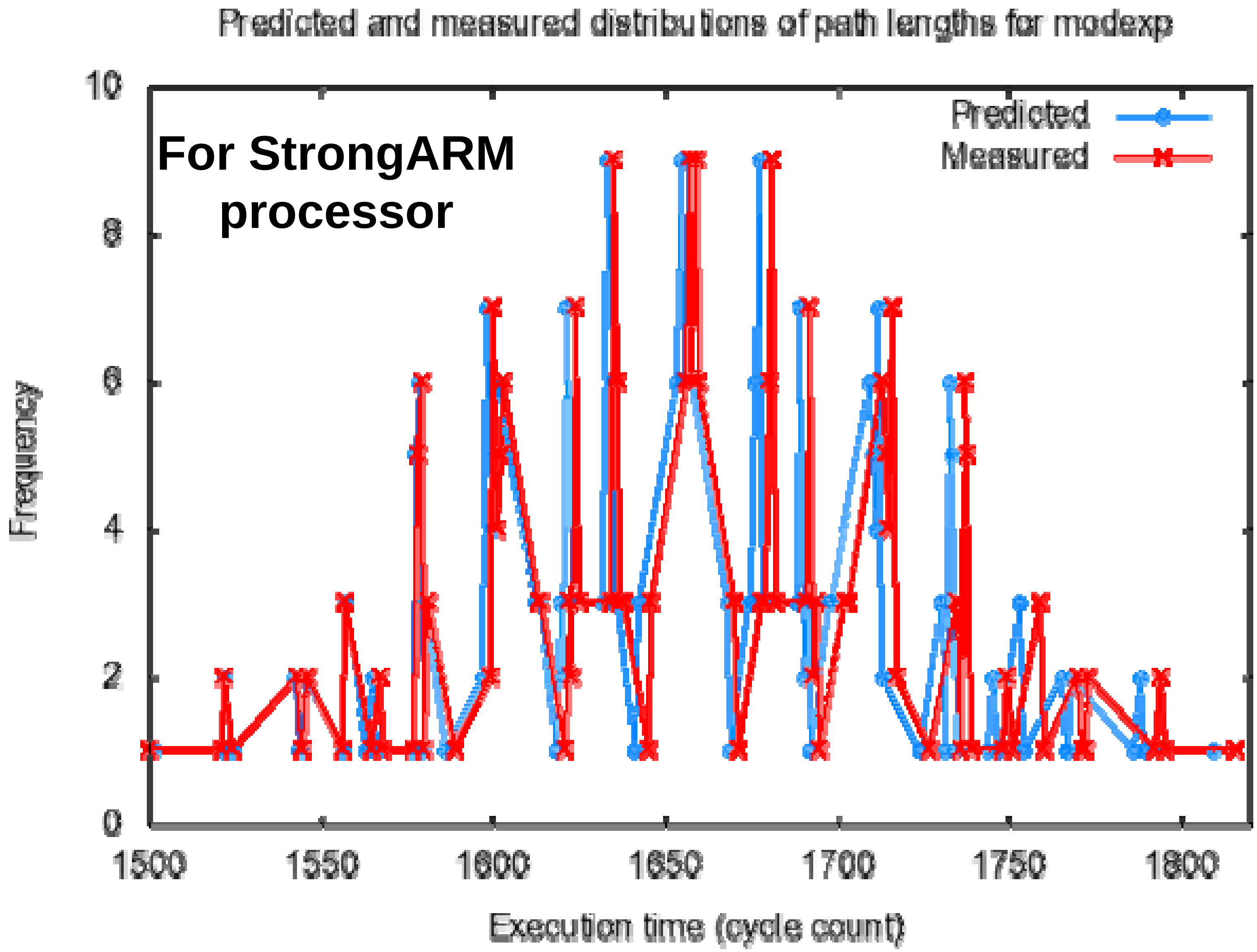
The GameTime Approach: Overview

Game-Theoretic Online Learning + Satisfiability Solving Modulo Theories (SMT)



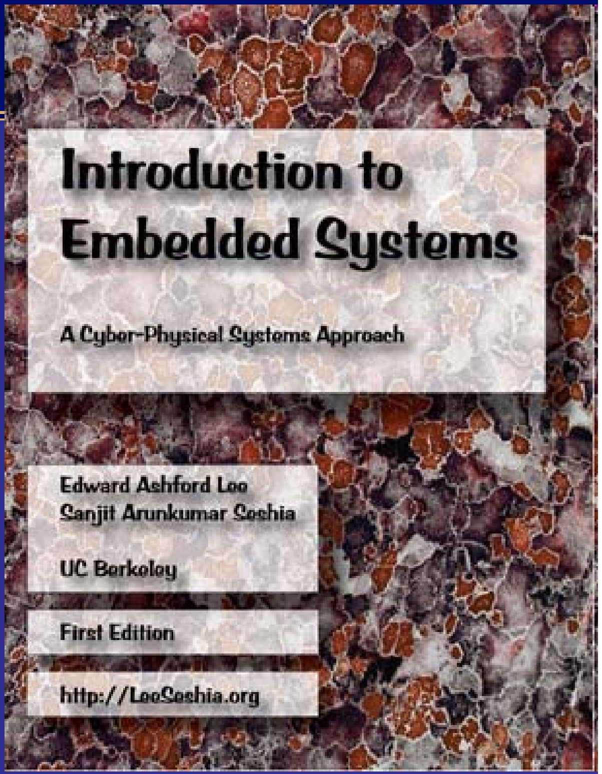
Publication: S. A. Seshia and A. Rakhlin, “Quantitative Analysis of Systems Using Game-Theoretic Learning”, ACM Trans. Embedded Computing Systems.

Estimating the Distribution of Times for Modular Exponentiation: predictions from 9 measurements in blue, actual 256 measurements in red



Potential Barriers
(from Academic Perspective)

- **Student Skills**
  - Students need cross-cutting skills (or willingness to learn)
  - Hardware + Software + Formal Methods
  - EE + CS
  - New UG course at Berkeley on Embedded Systems (EECS 149)



- **Lack of Open-Source Benchmarks**
  - More challenging for "quantitative" software verification!
    - Heavy dependence on hardware platform

Summary

- Quantitative Verification of Cyber-Physical Software Systems
- Challenge: Environment modeling
  - Current manual methods too tedious and error-prone
- Proposed Approach: Automatic model generation by Inductive Synthesis
  - Active Learning + Traditional verification techniques (e.g., SAT/SMT)
  - One instance: GameTime for timing analysis of software
  - Perhaps a killer app for synthesis methods?