Low-Level Verification of Embedded Software: Addressing the Challenge

Sanjit A. Seshia

Assistant Professor

EECS, UC Berkeley

FMCAD 2010 Panel

October 2010

Abstraction Layers in Computing

Algorithms, Protocols, Models

Application Software

Systems Software / Firmware

SW

HW

Architecture

Circuits

Devices

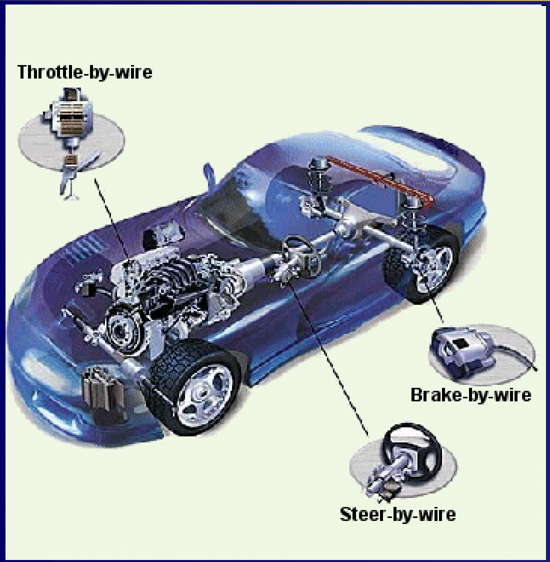
What makes Software "Low-Level"?
(from Verification perspective)

- **Properties**

- **Software is low-level if the behavior of the software system is defined significantly by lower levels of abstraction (hardware platform)**

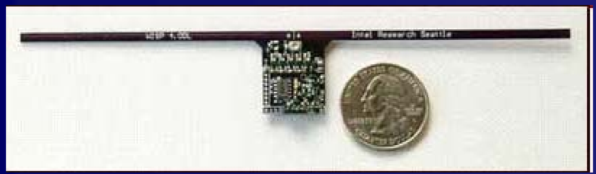
- **"Hardware-Software Verification"?**

Quantitative Analysis / Verification



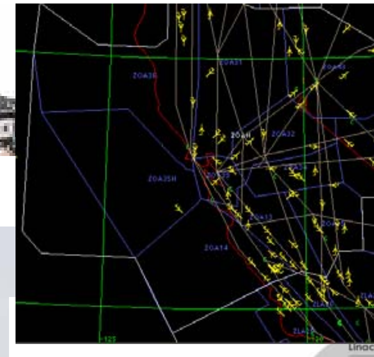
Does the brake-by-wire software always actuate the brakes within 1 ms?
Safety-critical embedded systems

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



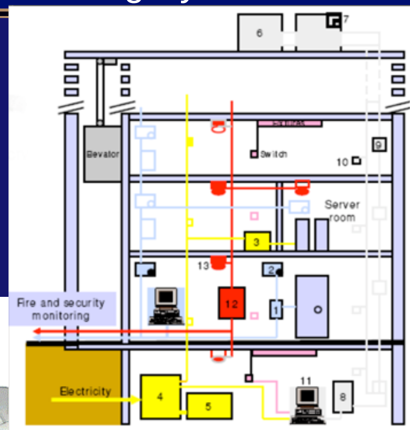
How much energy must the sensor node harvest for RSA encryption?
Energy-limited sensor nets, bio-medical apps, etc.

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

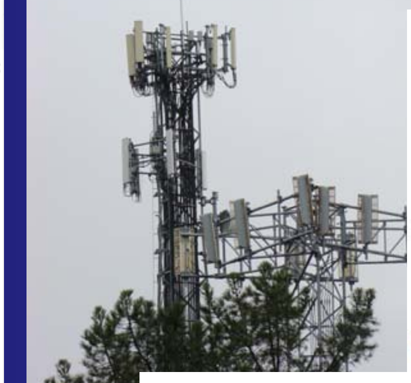


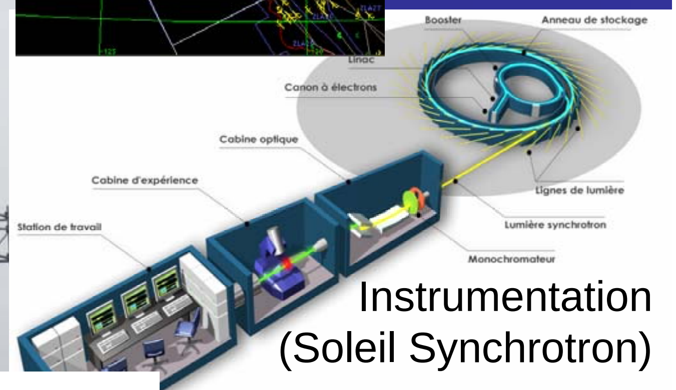
Transportation (Air traffic control at SFO)

Building Systems



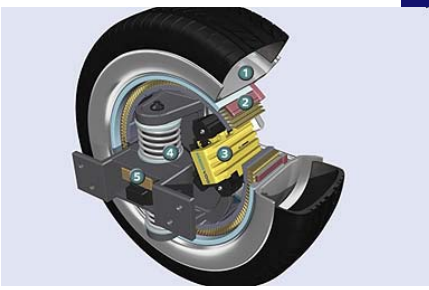
Telecommunications



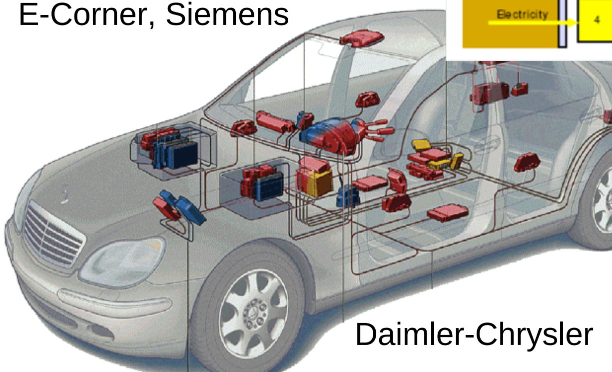


Instrumentation (Soleil Synchrotron)

Automotive

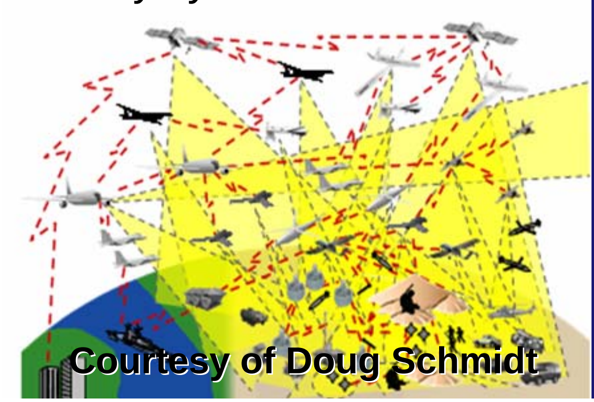


E-Corner, Siemens



Daimler-Chrysler

Military systems:



Courtesy of Doug Schmidt

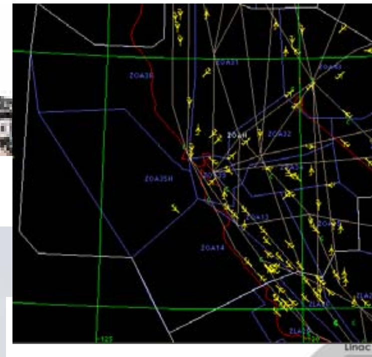
Power generation and distribution

Courtesy of General Electric

Factory automation

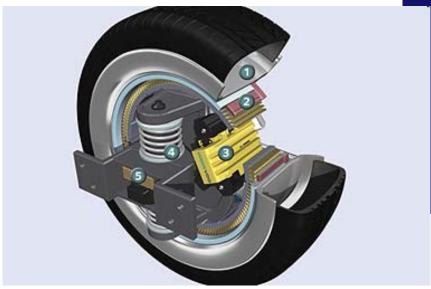
Courtesy of Kuka Robotics Corp.

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



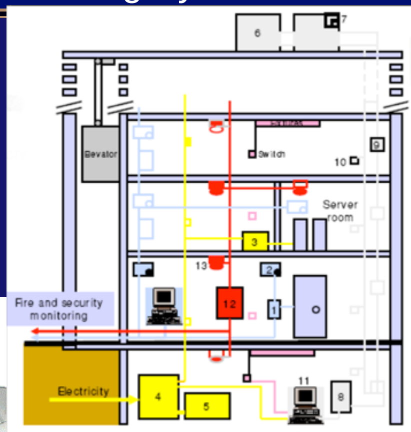
Transportation (Air traffic control at SFO)

Automotive

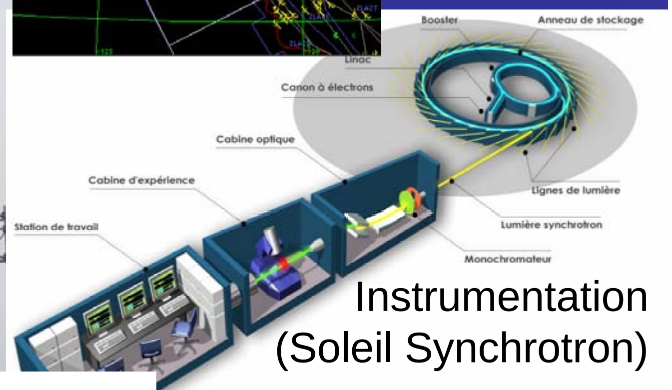


E-Corner, Siemens

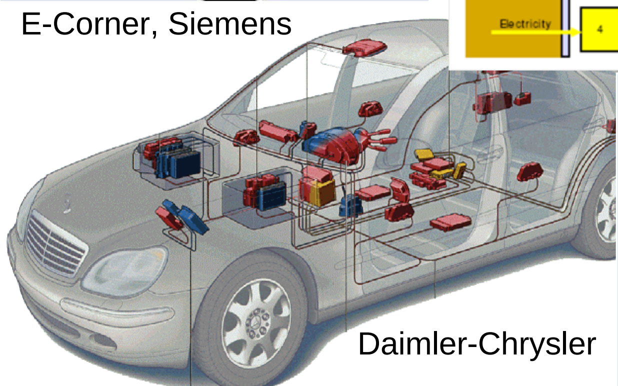
Building Systems



Telecommunications

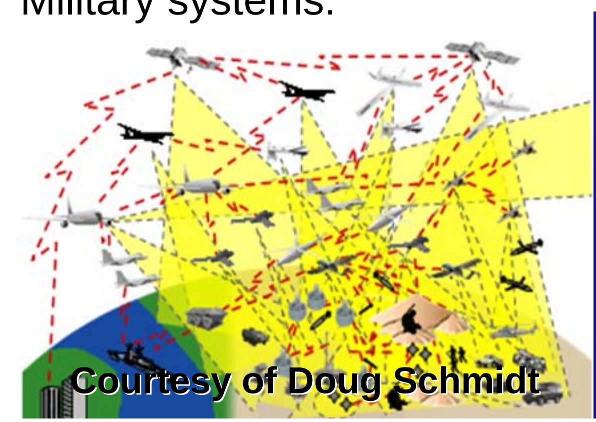


Instrumentation (Soleil Synchrotron)



Daimler-Chrysler

Military systems:

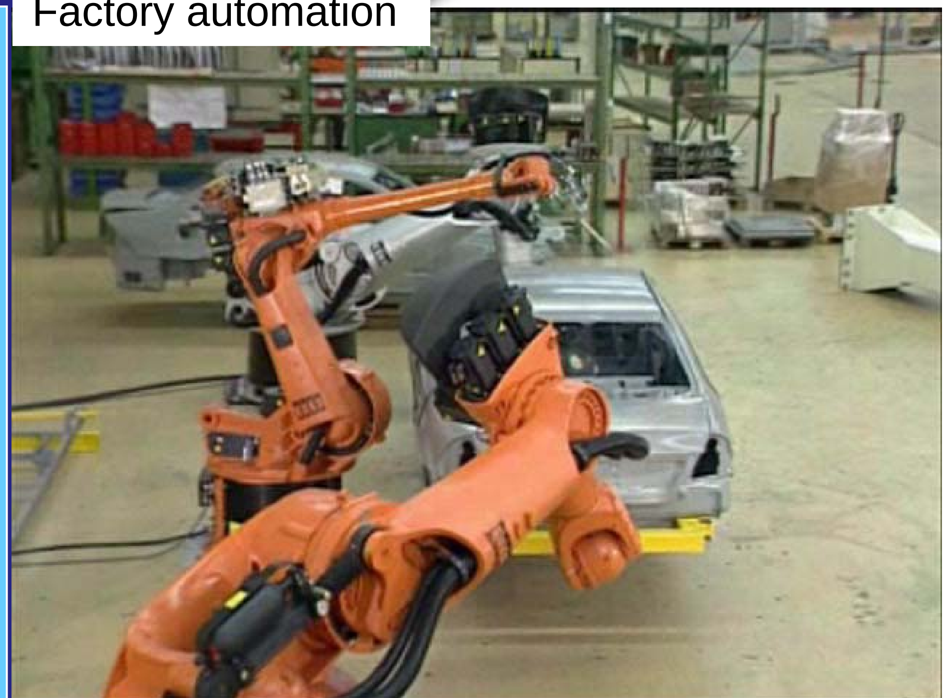


Courtesy of Doug Schmidt

Power generation and distribution

Courtesy of General Electric

Factory automation



Courtesy of Kuka Robotics Corp.

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

Success of "High-Level" Software Verification

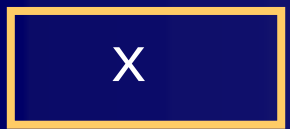
- From theoretical ideas to industrial practice in ~ 15 yrs

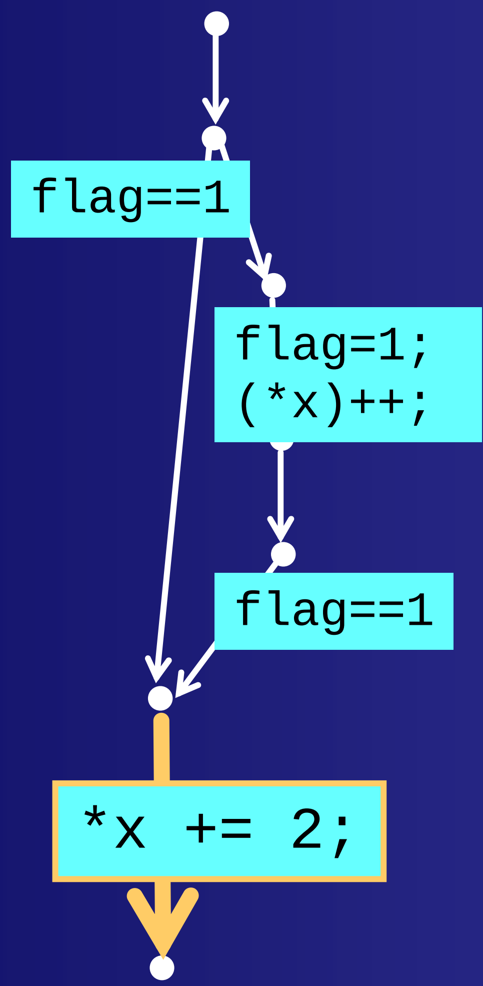
Some Reasons:

- 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 single-core processor with a data cache





CFG unrolled to a DAG

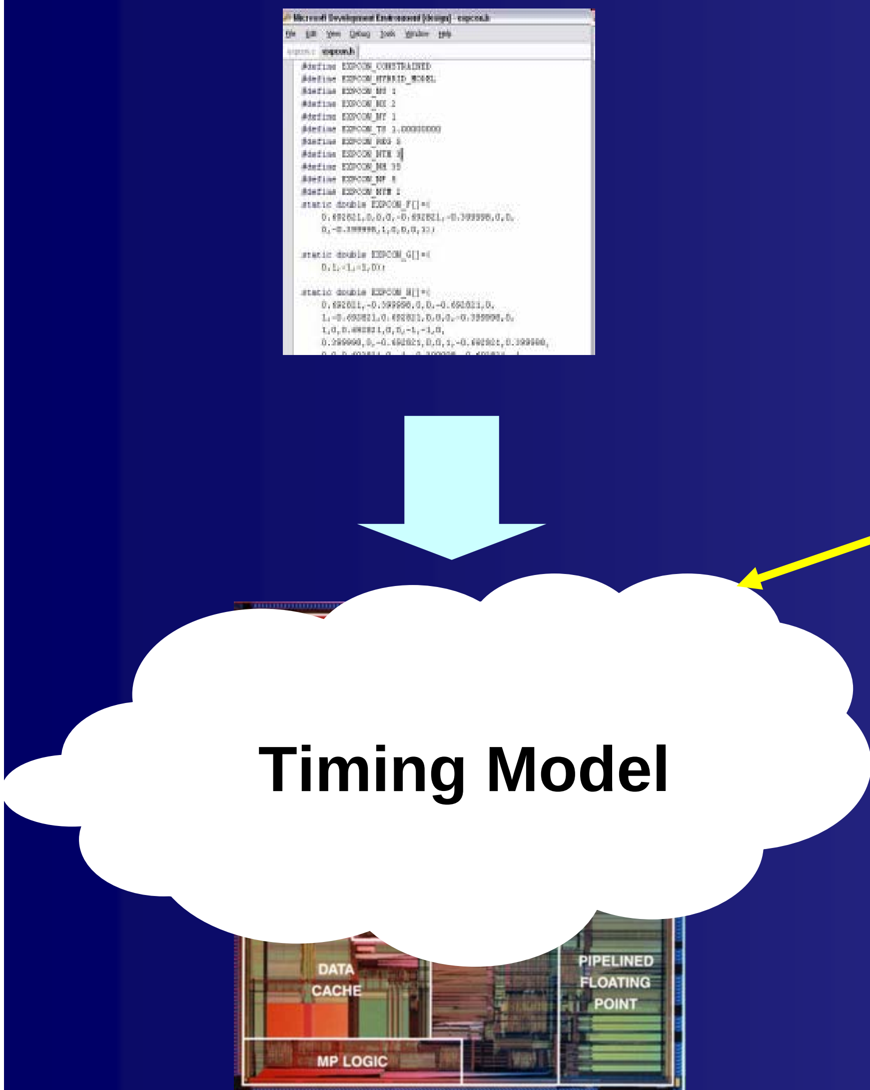
Timing of an edge (basic block) depends on:

- **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 several man-months to construct!

Also: limited to extreme-case analysis

- **Environment = Single-core Processor + Memory Hierarchy**

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?**

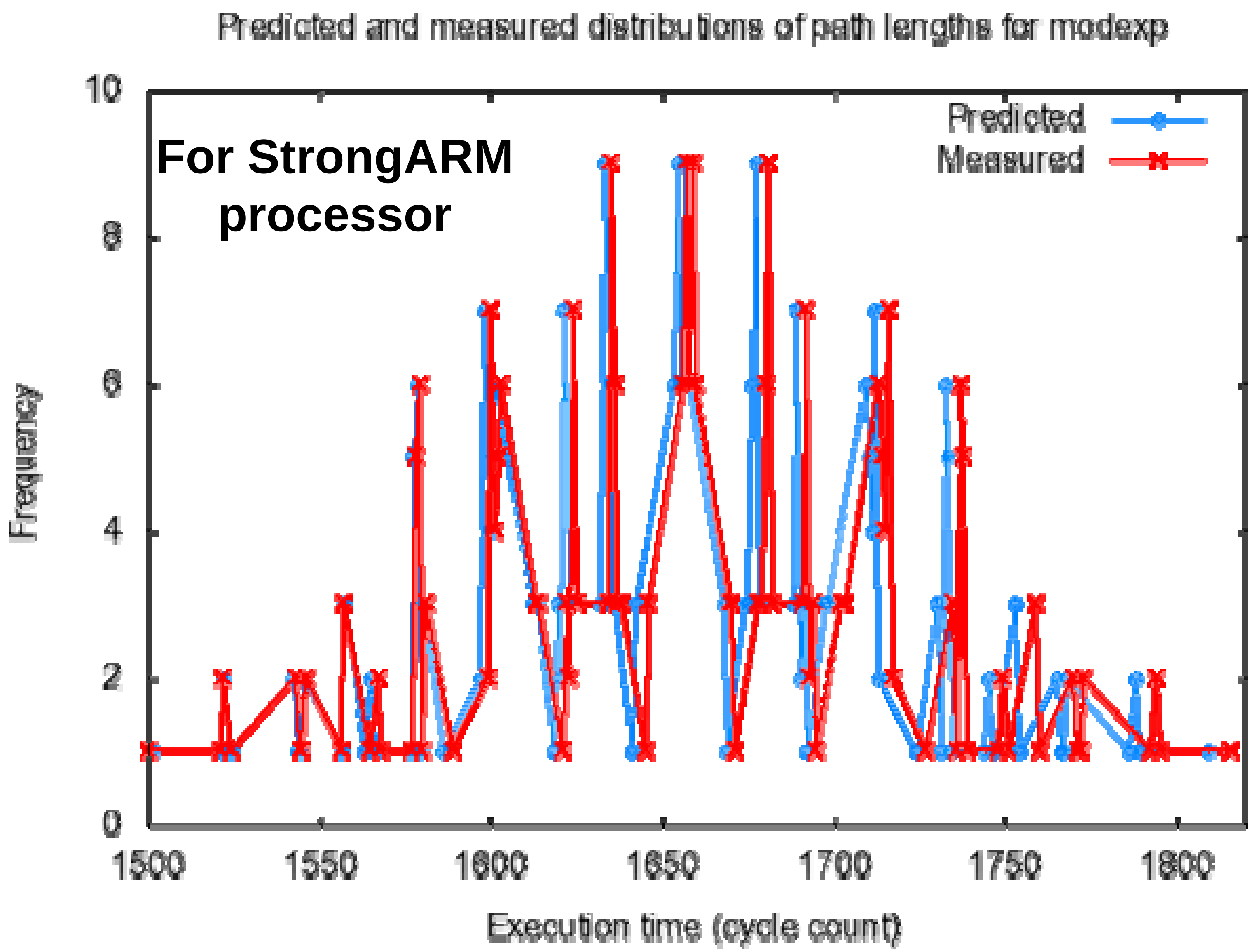
Promising Direction
(for timing analysis and low-level verification in general)

- **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 Embedded Systems Using Game-Theoretic Learning", ACM Trans. Embedded 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)

- Lack of Open-Source Benchmarks
  - Recent progress in software verification was driven by wide availability of open-source software
  - More challenging for “low level” software verification!
  - Heavy dependence on platform makes it more challenging
- Hardware + Software Skills
  - Students need cross-cutting skills (or willingness to learn) to work in this area

Summary

- "Low level" software = **Software whose behavior is significantly defined by hardware**
  - Hardware-Software Verification?
- 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?