EECS 219C:
Formal Methods
Introduction & Overview

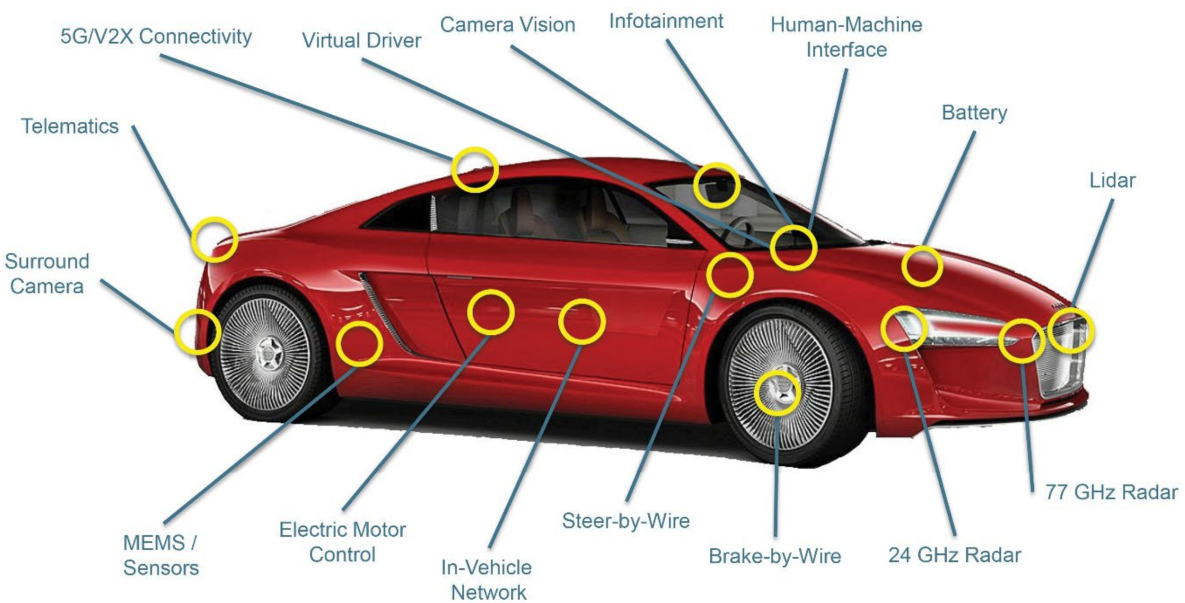
Sanjit A. Seshia
EECS, UC Berkeley

Buying a Car: the Consumer Perspective

Does **the car** have
the features you want?

The Engineer's Perspective

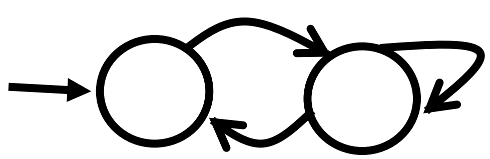
Does **the implemented system** meet its **specifications**?



Credit: EE World Online

The Mathematician's Perspective

Prove or disprove (verify) that
the mathematical model of the system
satisfies a **mathematical specification**



$$\dot{x}(t) = f(x(t), u(t))$$

Formal Verification (informally)

Does **the system** do
what it is supposed to do?

Formal Methods

Rigorous mathematical, algorithmic techniques for specification, design, verification and maintenance of computational systems.

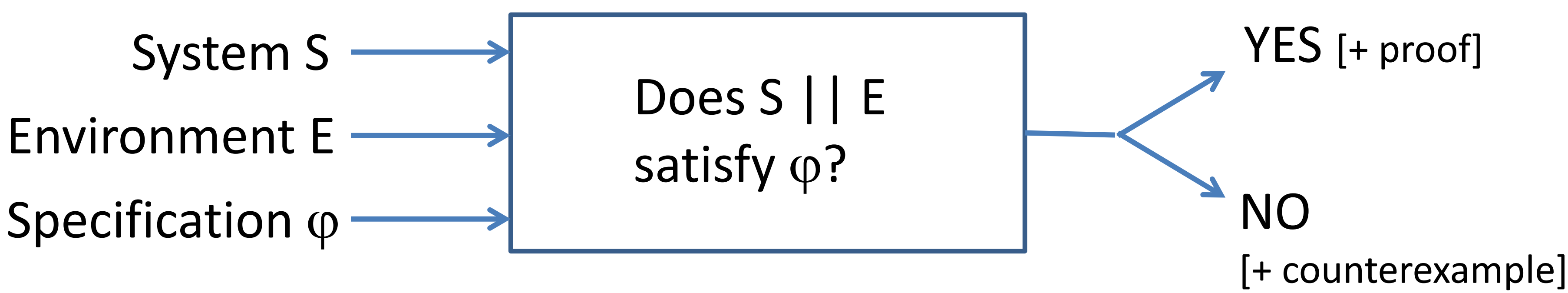
The essence: It's about PROOF

- Specify proof obligations
- Prove that system meets those obligations
- Synthesize provably-correct system

The Formal Methods Lens

- Formal Methods ≈ Computational Proof methods
  - Specification/Modeling ≈ Statement of Conjecture/Theorem
  - Verification ≈ Proving/Disproving the Conjecture
  - Synthesis ≈ Generating (parts of) Conjecture/Proof
  - ➢ Tools/techniques: SAT / SMT solvers, model checkers, theorem provers, simulation-based falsification, …

Verification:



Three Key Areas of Formal Methods

- Specification
  - WHAT must the system (program) do?
  - includes Modeling
- Verification
  - WHY does the system do it? (or not)
- Synthesis
  - HOW does the system do it?

What we'll do today

- Introductions: to Sanjit and others
- Brief Intro. to the main course topics
  - Motivation
  - Basics: Propositional Logic, First-Order Logic, Temporal Logic, Model Checking, SAT, Satisfiability Modulo Theories (SMT), …
  - History, Opportunities, Challenges
- Course Logistics

My Research

"Formal Methods: Specification, Verification, Synthesis"

+

Theory

Computational Logic,
Algorithms,
Learning Theory,
Optimization

Practice

CAD for Circuits/Bio, AI,
Software Engg, Computer
Security, Embedded/Cyber-
Physical Systems, Education

Current Foci: Verified Intelligent (AI) Systems / Secure Systems

Class Introductions

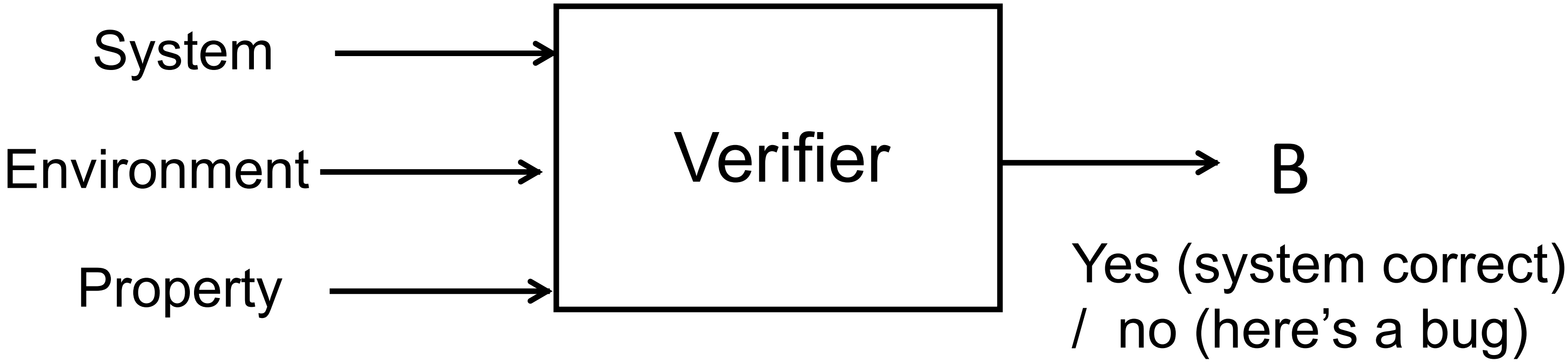
Please introduce yourselves

-- state name and research interests/areas

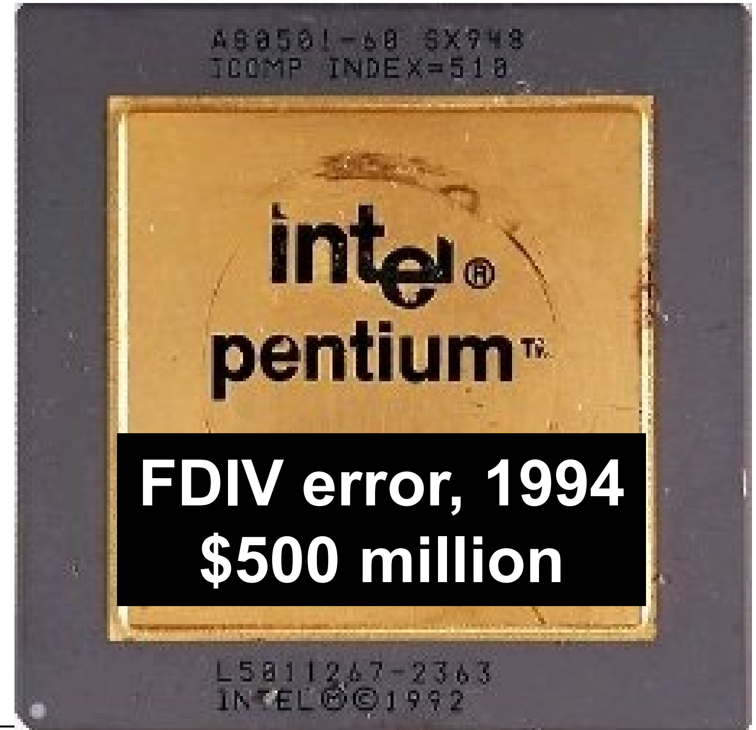
(Programming Systems, Computer Security, Architecture, CAD, Embedded Systems/CPS, BioSystems, Control Theory, AI, ML, Robotics, etc.)

Formal Verification

- Automatically verifying the correctness of systems



- Questions for today:
  - Is it relevant?
  - Is it feasible?
  - What will we study?



Toyota Recalls 1.9 Million Prius Hybrids Over Software Flaw

By Jeremy Hsu

Posted 12 Feb 2014 | 21:55 GMT

```
<msblast.exe> (the primary executable of the exploit)
I just want to say LOVE YOU SAN!!
billy gates why do you make this possible ? Stop
making money and fix your software!!
windowsupdate.com
start %s
tftp -i %s GET %s
%d.%d.%d.%d
%i.%i.%i.%i
```

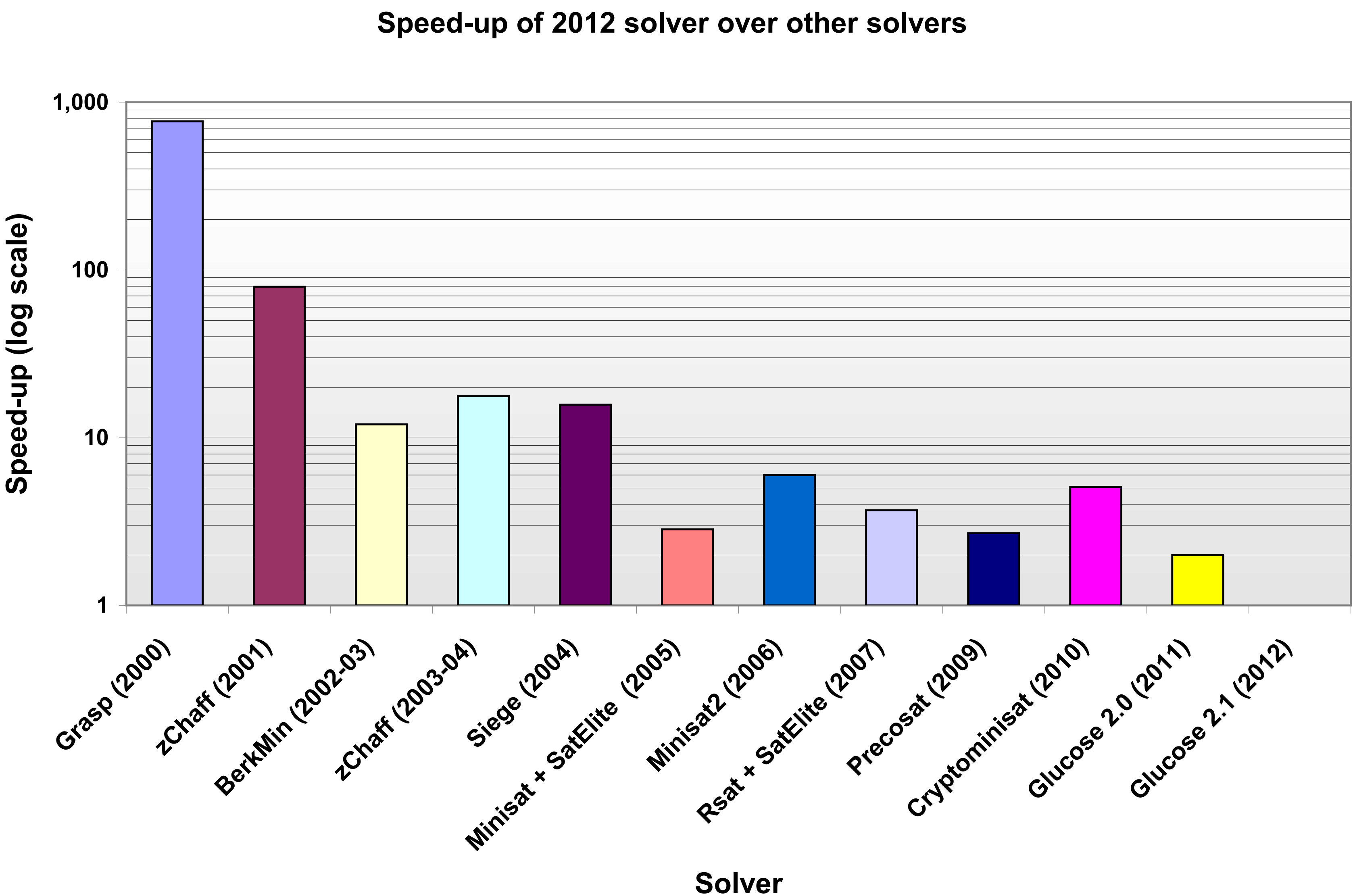
Estimated worst-case worm cost: > $50 billion

Bugs cost Time, Money, Lives, …

Is Verification Feasible?

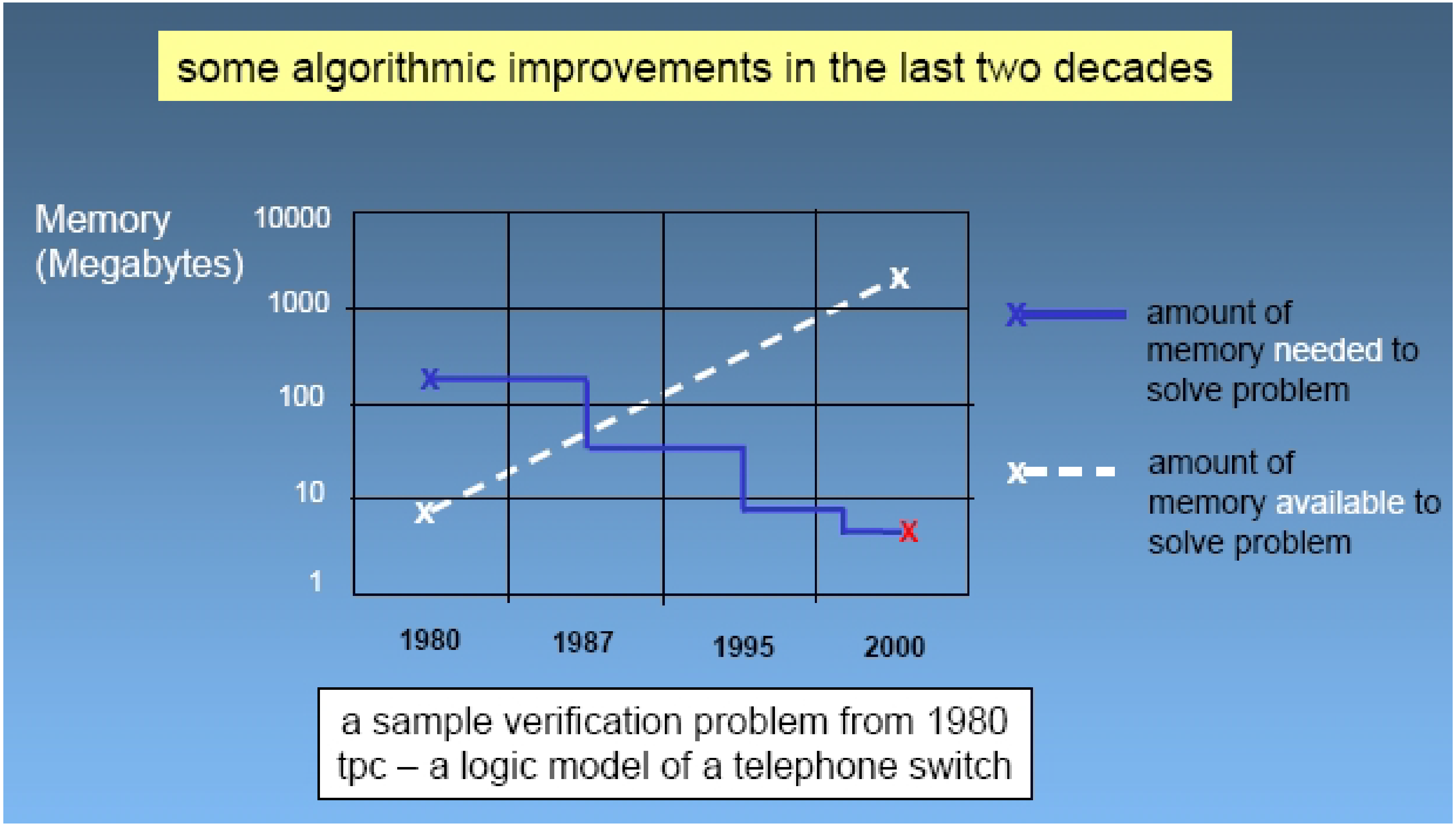
- "Easiest" non-trivial verification problem is NP-hard (SAT)
- But the outlook for practice is less gloomy than for theory…
  - More hardware resources
  - Better algorithms

My Experience with SAT Solving
(over ~a decade)



Experience with SPIN Model Checker

[G. Holzmann]



Topics in this Course

- Computational Engines / Basic Topics
  – Boolean satisfiability (SAT)
  – Satisfiability modulo theories (SMT)
  – Model checking
  – Syntax-guided synthesis (SyGuS)
- Advanced Topics (“Research Frontiers”)
  – Deduction + Inductive Learning
  – Safe/Verified Artificial Intelligence (AI)
  – Human-Robot/Computer Interaction & Formal Methods
  – New application domains
  – … (more later in this lecture)

Topics of this Course (another view)

Application Domains

Circuits, Software, Networks, Hybrid Systems, Biological Systems, etc.

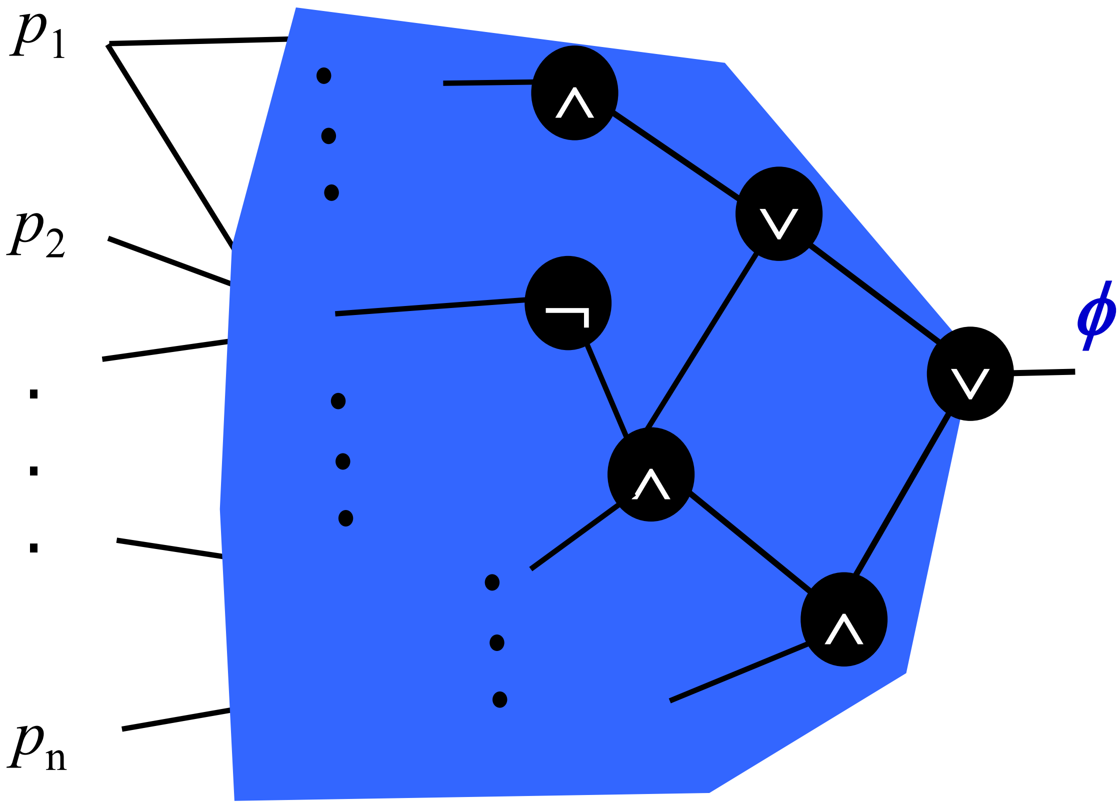
Verification/Synthesis Strategies

Automata-theoretic, Symbolic, Abstraction, Learning, etc.

Computational Engines

SAT, BDDs, SMT

Boolean Satisfiability (SAT)

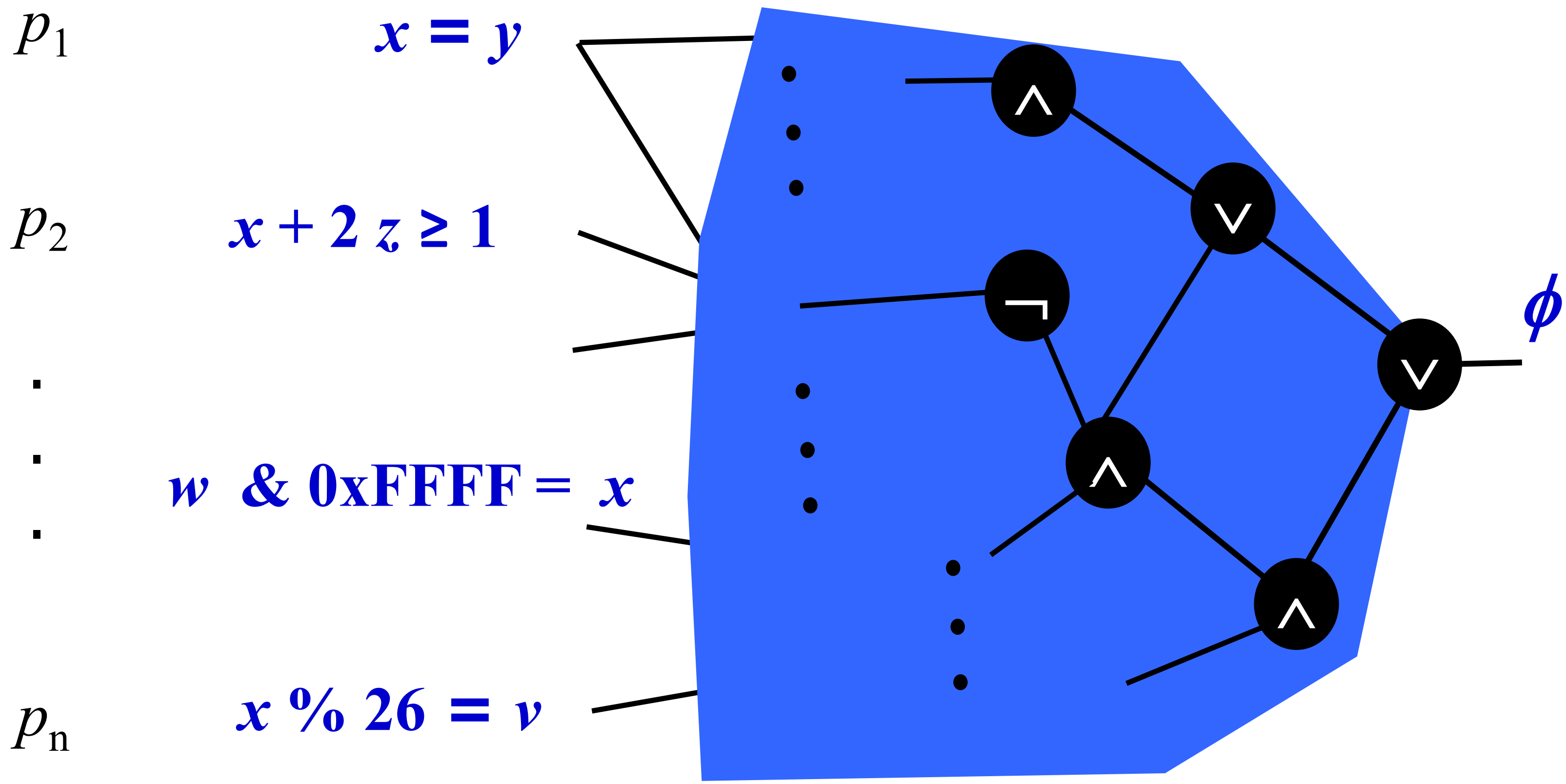


Is there an assignment to the p_i variables s.t. ϕ evaluates to 1?

Two Applications of SAT

- Equivalence checking of circuits
  - Given an initial (unoptimized) Boolean circuit and its optimized version, are the two circuits equivalent?
  - Standard industry CAD problem
- Malware detection (security)
  - Given a known malicious program and a potentially malicious program, are these “equivalent”?
- Many other applications:
  - Cryptanalysis, test generation, model checking, logic synthesis, ….

Satisfiability Modulo Theories (SMT)



Is there an assignment to the *x,y,z,w* variables s.t. ϕ evaluates to 1?

Applications of SMT

- Pretty much everywhere SAT is used
  - The original problem usually has richer types than just Booleans!
- To date: especially effective in
  - software model checking
  - test generation
  - software synthesis
  - finding security vulnerabilities
  - high-level (RTL and above) hardware verification

Model Checking

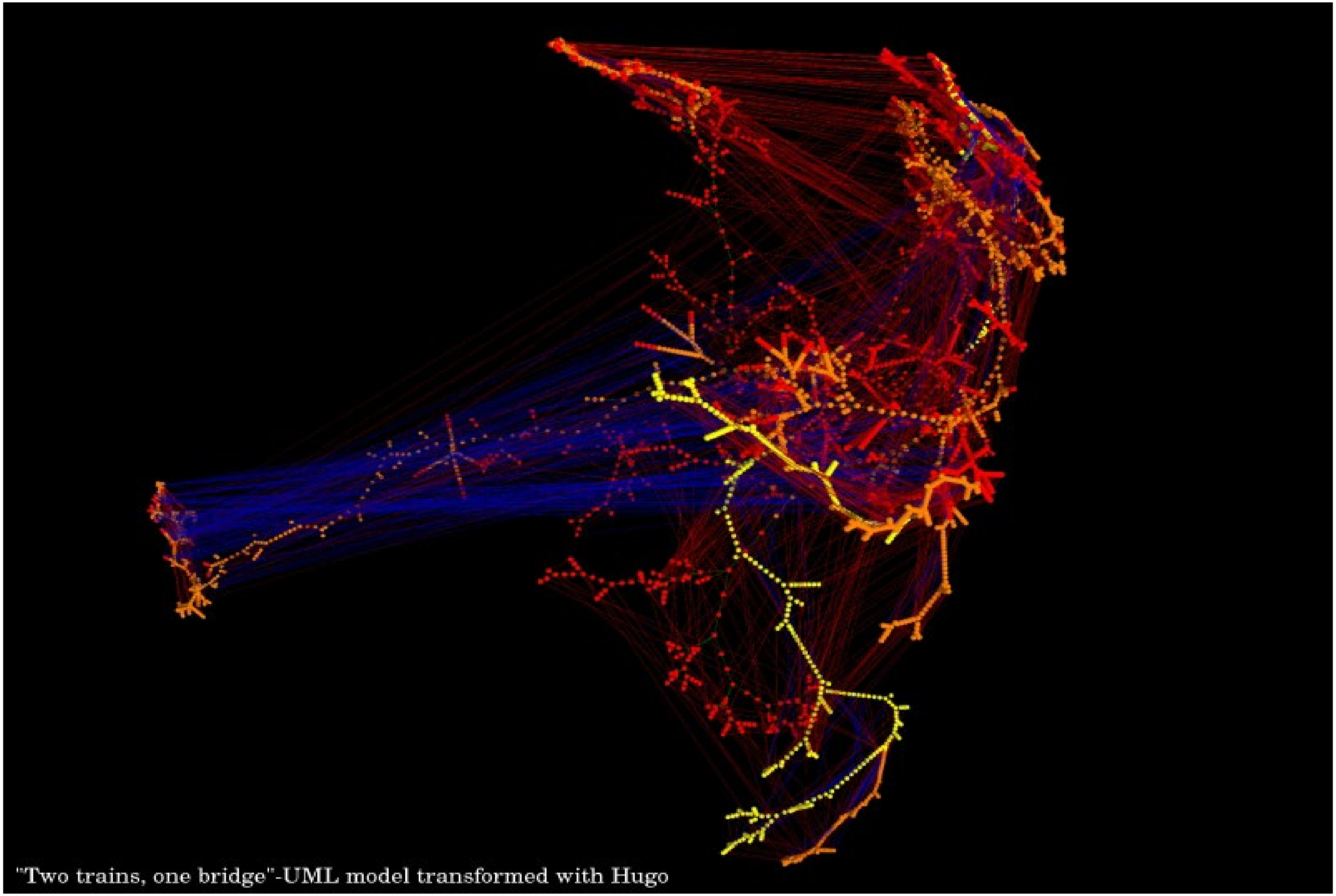
- Broad Defn:

  A collection of **algorithmic methods** based on **state space exploration** used to verify if a **system satisfies a formal specification.**

- Original Defn: (Clarke)

  A technique to check if a finite-state system is a model of (satisfies) a temporal logic property.

Visualizing Model Checking



[Moritz Hammer, Uni. Muenchen]

Model Checking, (Over)Simplified

- Model checking “is” graph traversal ?
- What makes it interesting:
  - The graph can be HUGE (possibly infinite)
  - Nodes can represent many states (possibly infinitely many)
  - How do we generate this graph from a system description (like source code)?
  - Behaviors/Properties can be complicated (e.g. temporal logic)
  - …

A Brief History of Formal Methods

(PREAMBLE)

Focus on (Highly) Automated Formal Methods

- 1949: Early program proof by Alan Turing
- 50s & 60s: Lot of relevant work on automata theory by several researchers (e.g. Buchi, Rabin, …)
- 1967: paper on proving program assertions by Floyd
- 1969: Tony Hoare’s paper on logic-based reasoning to prove programs correct (or not)
- Early 70s: lots of work on proving sequential programs correct

A Brief History of Formal Methods
(biased towards Model Checking)

- 1977: Pnueli introduces use of (linear) temporal logic for specifying program properties over time [1996 Turing Award]
- 1981: Model checking introduced by Clarke & Emerson and Quielle & Sifakis
  - Based on explicitly traversing the graph
  - capacity limited by “state explosion”
- 1986: Vardi & Wolper introduce “automata-theoretic” framework for model checking
  - Late 80s: Kurshan develops automata-theoretic verifier
- Early - mid 80s: Gerard Holzmann starts work on the SPIN model checker

A Brief History of Formal Methods

(biased towards Model Checking)

- 1986: Bryant publishes paper on BDDs
- 1987: McMillan comes up with idea for "Symbolic Model Checking" (using BDDs) – SMV system
  - First step towards tackling state explosion
- 1987-1999: Flurry of activity on finite-state model checking with BDDs, lots of progress using: abstraction, compositional reasoning, …
  - More techniques to tackle state explosion
- 1990-95: Timed Automata introduced by Alur & Dill, model checking algorithms introduced; generalized to Hybrid Automata by Alur, Henzinger and others

A Brief History of Formal Methods

- 1999: Clarke et al. introduce “Bounded Model Checking” using SAT
  - SAT solvers start getting much faster
  - BMC found very useful for debugging hardware systems
- 1999: Model checking hardware systems (at Boolean level) enters industrial use
  - IBM RuleBase, Synopsys Magellan, 0-In FV, Jasper JasperGold
- 1999-2004: Model checking + theorem proving: software and high-level hardware comes of age
  - SLAM project at MSR, SAL at SRI, UCLID at CMU
  - Decision procedures (SMT solvers) get much faster
  - Software verifiers: Blast, CMC, Bandera, MOPS, …
  - SLAM becomes a Microsoft product “Static Driver Verifier”

A Brief History of Formal Methods

- 2005-date: Model Checking is part of the standard industrial flow. Some new techniques and applications arise:
  - Combination with simulation (hardware) and static analysis/testing (software) [Many univ/industry groups]
  - Checking for termination in software [Microsoft]
  - Lots of progress in verification of concurrent software [Microsoft]
  - SMT solvers get much faster and better, used widely
  - Many applications in cloud computing and beyond [AWS]
- Inductive synthesis [Berkeley, Microsoft, MIT, Penn, …]
  - 2006: Counterexample-guided inductive synthesis (CEGIS) and Sketching-based synthesis developed at Berkeley
  - 2010: First example-driven "oracle-guided synthesis" methods [Berkeley+SRI+Microsoft]
  - 2010s: End-user programming grows [Microsoft, UW,…], Inductive synthesis for specification inference [Berkeley, Toyota], etc.
  - 2013-date: Syntax-Guided Synthesis (SyGuS) arrives [NSF ExCAPE project]

Some Recent Recognition for the Field

- Clarke, Emerson, Sifakis get 2007 ACM Turing Award for Model Checking
- SAT and SMT solving advances are recognized by CAV Awards (2009 and 2021), other awards…
- 2013 Turing Award for Lamport (in part for Specification/Verification work)
- Etc.

WHAT'S NEXT?!

Research Frontiers in Formal Verification

- Three Themes:
  - New Demands on Computational Engines
  - New Applications
  - The “Human Aspect”
    - Steps that require significant human input
    - Systems with humans in the loop

→ suggested project topics by mid-Feb

Formal Methods meets Machine Learning

- Machine Learning → Formal Methods
  - Greater efficiency, ease of use/applicability
  - Formal Inductive Synthesis
  - Use of LLMs in Formal Methods?
- Formal Methods → Machine Learning
  - Stronger assurances of safety/correctness for learning systems
  - “Trustworthy AI”, “AI Safety”, etc.

Further details:

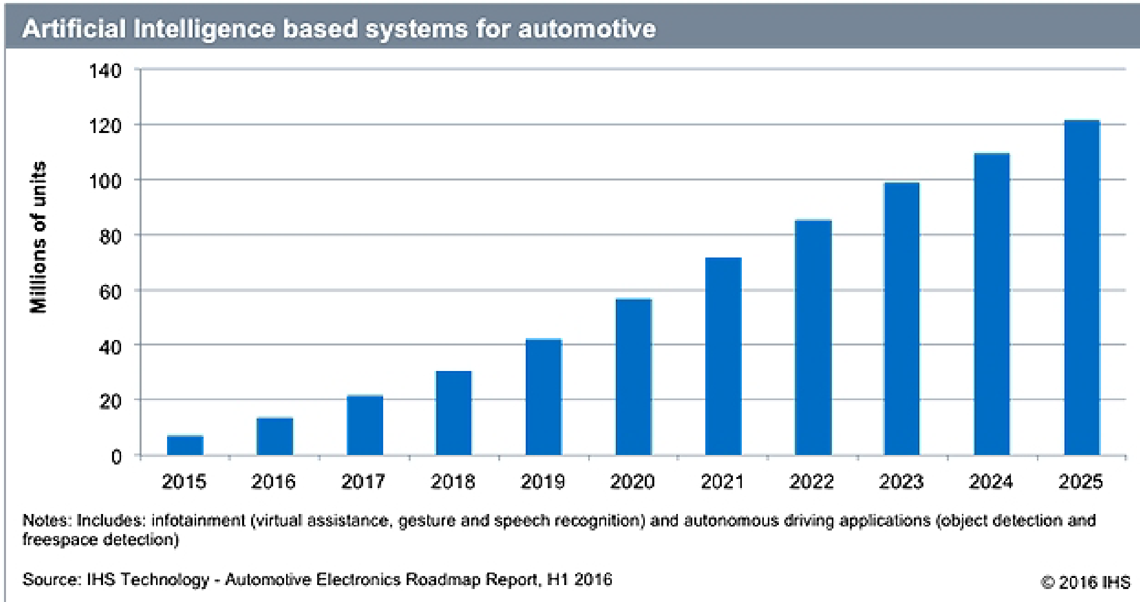
1. S. A. Seshia, “Combining Induction, Deduction, and Structure for Verification and Synthesis”, Proceedings of the IEEE, November 2015.
2. S. A. Seshia, D. Sadigh, and S. S. Sastry, “Towards Verified Artificial Intelligence”, July 2016, http://arxiv.org/abs/1606.08514, revised version in Communications of the ACM, July 2022.

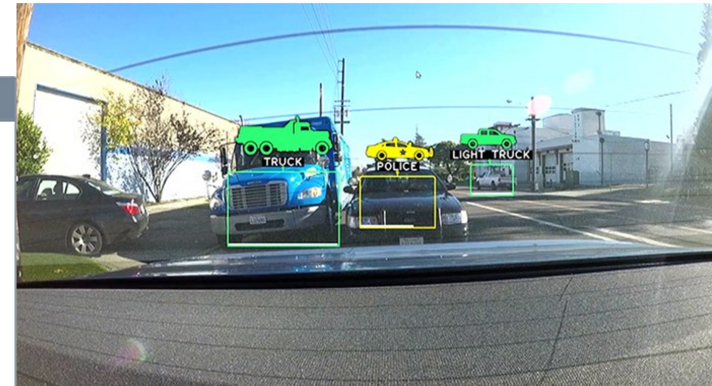
New Directions in Computational Engines

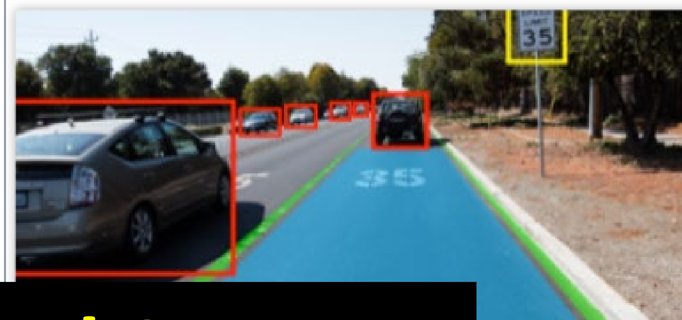
- Quantitative versions of SAT/SMT
  - SAT → MaxSAT, Model Counting, etc.
  - SMT → Optimization Modulo Theories, Model Counting, etc.
- ML for Automated Reasoning
  - Neural approaches to SAT, SMT, QBF, Model counting, etc.
- Synthesis solving
  - Synthesis Modulo Oracles [see Polgreen, et al. VMCAI'22]

New Application Domains

Growing Use of Machine Learning/AI in Cyber-Physical Systems



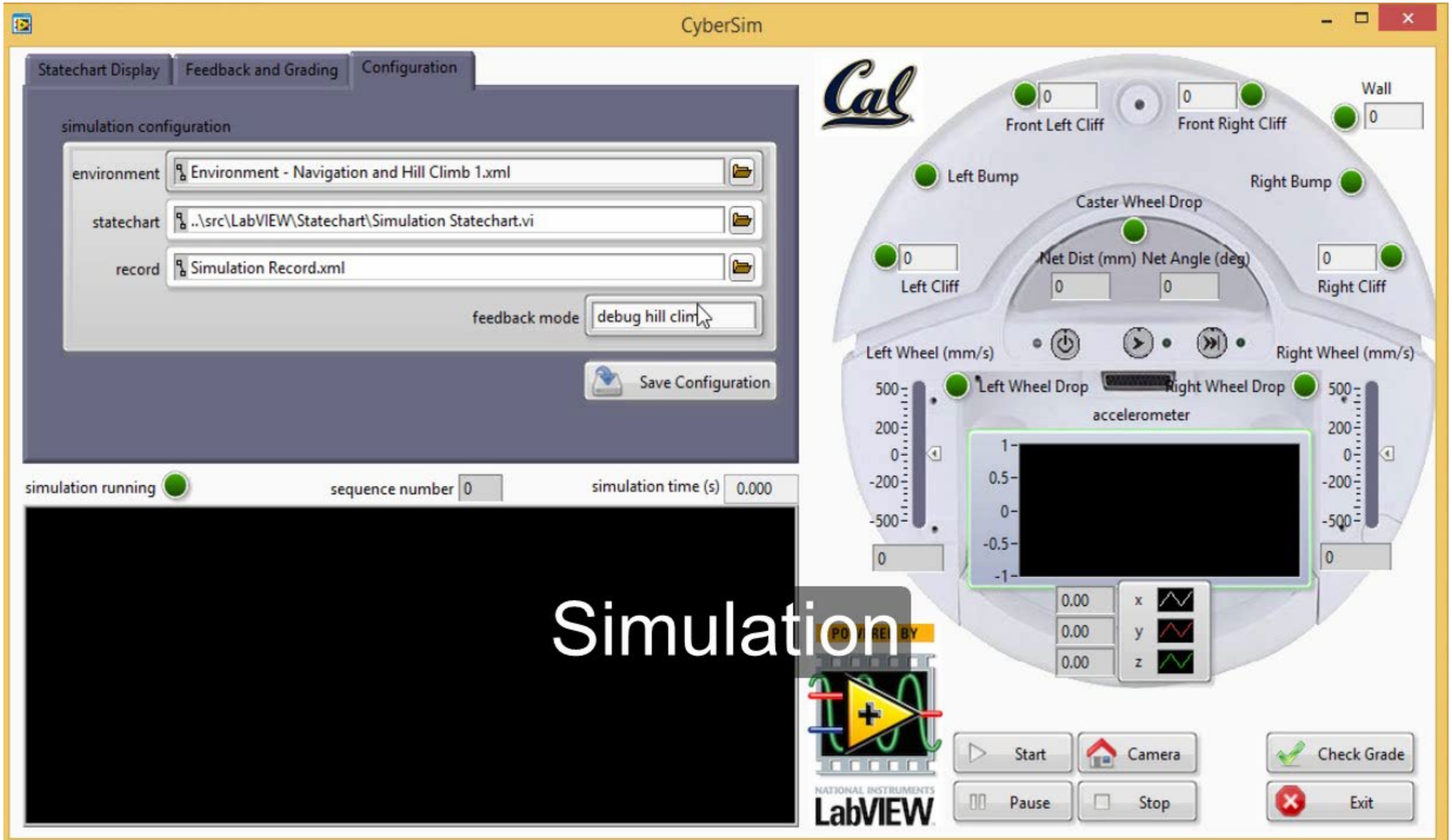




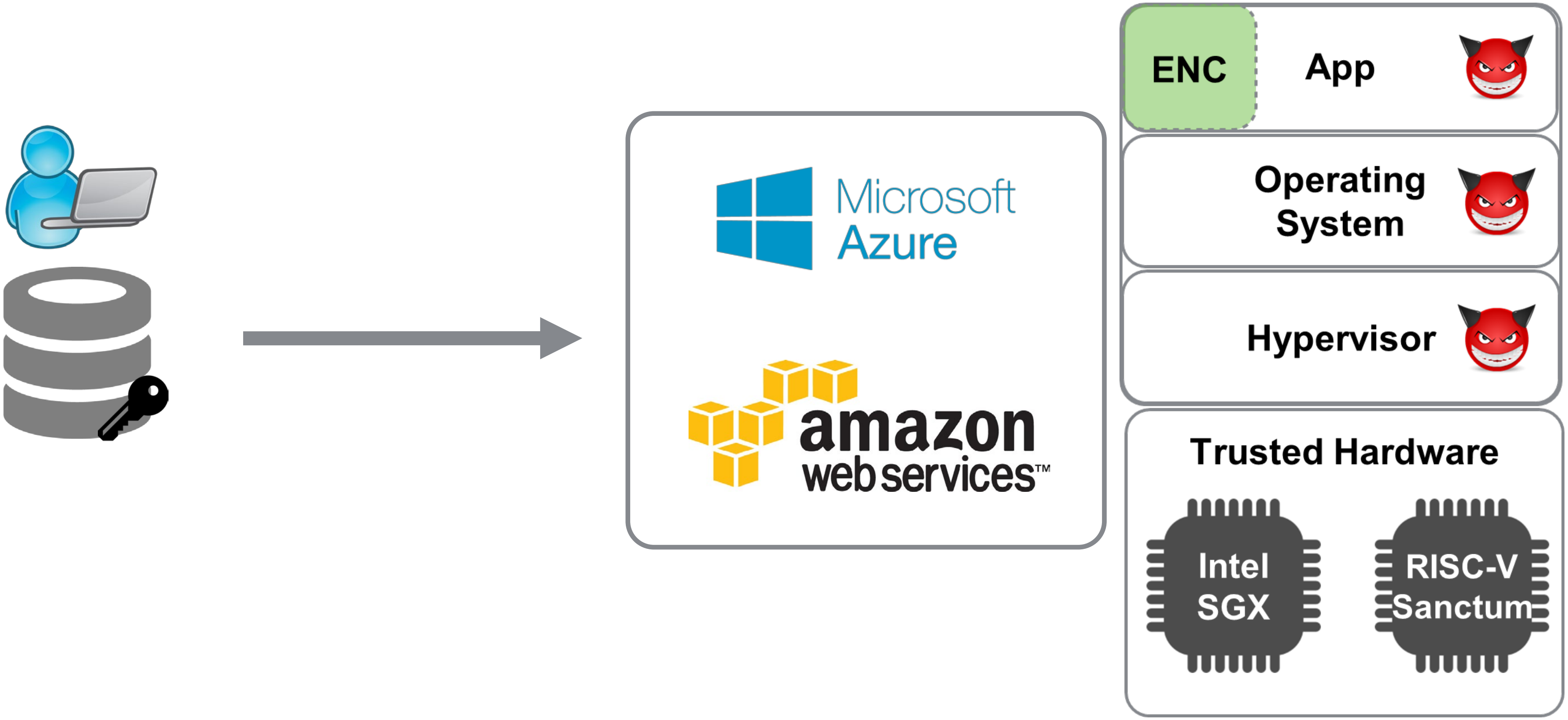
Many Safety-Critical Systems

Formal Methods for Education

<u>Goal:</u> To enable personalized learning for lab-based courses in science and engineering → CPSGrader, deployed on edX and on campus



Formal Methods for Distributed/Secure Systems



- Does my secret data remain secret?
- Does the program execute as it is supposed to?
- Is the right program executed?

Course Logistics

- Check out the webpage:
  www.eecs.berkeley.edu/~sseshia/219c
- Tentative class schedule is up
  - IMP: Think about project topics!

Course Outline

- 2 parts
- Part I: Basics: Boolean reasoning (SAT, BDDs), SMT Solving, Temporal Logic, Model Checking
  - Basics, how to use these techniques, and how to extend them further
- Part II: Advanced Topics
  - The challenging problems that remain to be addressed

Reference Books

- Readings: Course notes from previous years + draft textbook
- See list of ref books on the website
- Readings for most material posted on bCourses

Grading

- Homework (~30%)
  - First part of the course
- Scribing lectures (maybe)
  - 2 lectures per person: Scribe one lecture, edit another lecture
  - Sign-up sheet next week
- Paper discussions / class participation (10%)
  - Last month of the course
- **Project (50-60%)**
  - Do original research, theoretical or applied
  - Sample topics will be announced by end of next week
  - Project proposal due mid Feb.
  - Culminates in final presentation + written paper
  - *~50% of past projects led to conference papers!*

Misc.

- Office hours: MW 2:30-3 pm and by appointment
- Pre-requisites: check webpage; come talk to me if unsure about taking the course
  - Undergraduates need special permission to take this class