Sound Static Analysis: 5-point seat belts for your code

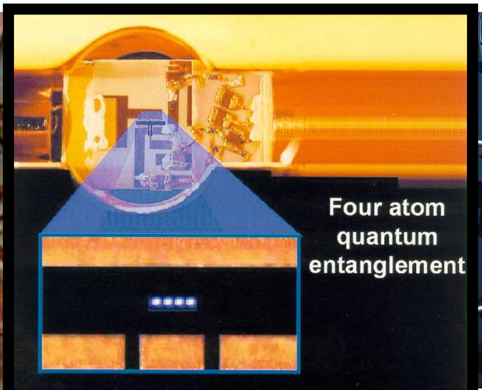
Paul E. Black

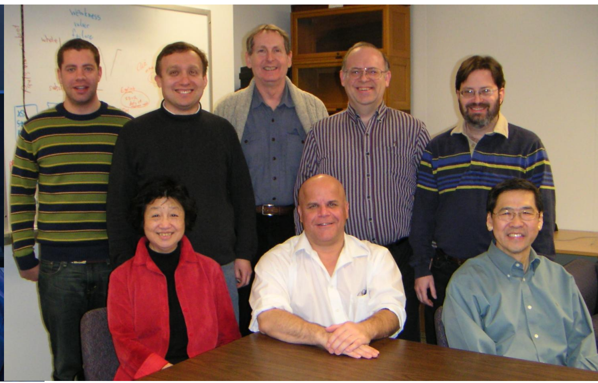
paul.black@nist.gov

27 June 2018

What is NIST?

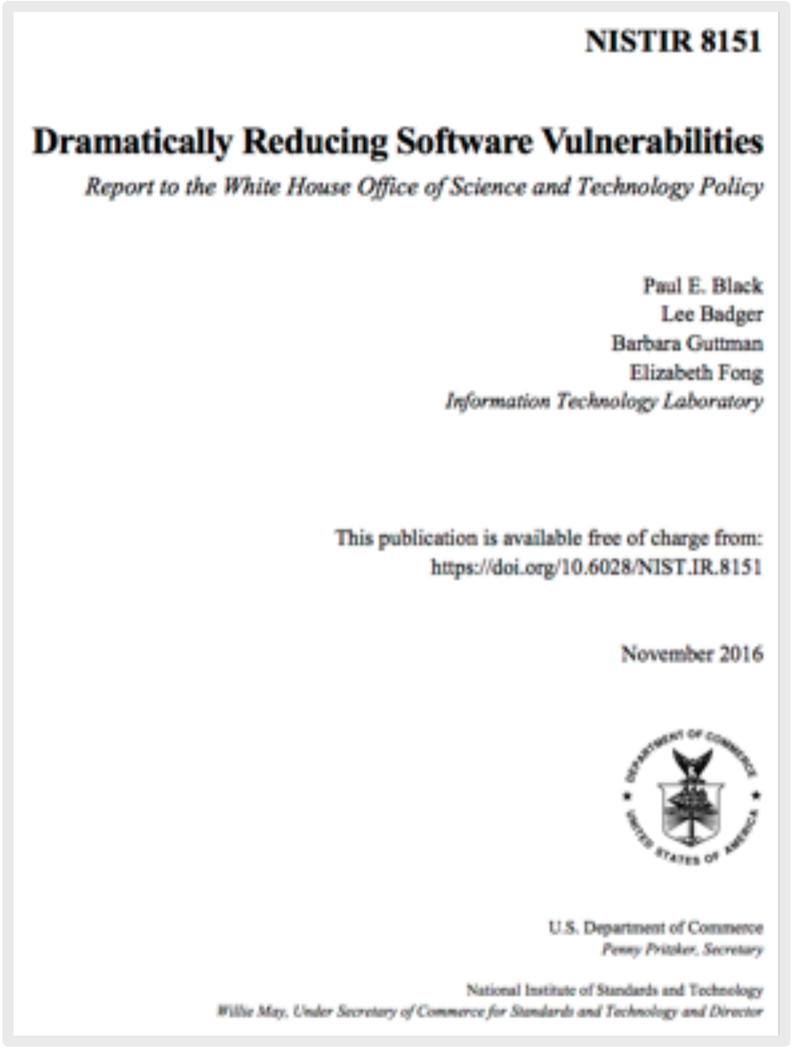
- U.S. National Institute of Standards and Technology
- A non-regulatory agency in Dept. of Commerce
- 3,000 employees + adjuncts
- Gaithersburg, Maryland and Boulder, Colorado
- Primarily research, not funding
- Over 100 years in standards and measurements: from dental ceramics to microspheres, from quantum computers to fire codes, from body armor to DNA forensics, from biometrics to text retrieval





Who Cares About Good Software?

- **The White House Office of Science and Technology Policy (OSTP) asked NIST to compile a list of approaches to dramatically reduce software vulnerabilities.**



NISTIR 8151

Dramatically Reducing Software Vulnerabilities

Report to the White House Office of Science and Technology Policy

Paul E. Black
Lee Badger
Barbara Guttman
Elizabeth Fong
Information Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8151

November 2016

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Under Secretary of Commerce for Standards and Technology and Director

What DRSV Covers

- Vulnerabilities
- New *and* existing code
- Approaches in 5 areas that may have dramatic impact in three to seven years.
- Other stuff
  - Software measures
  - Education, contracts, and other non-technical matters

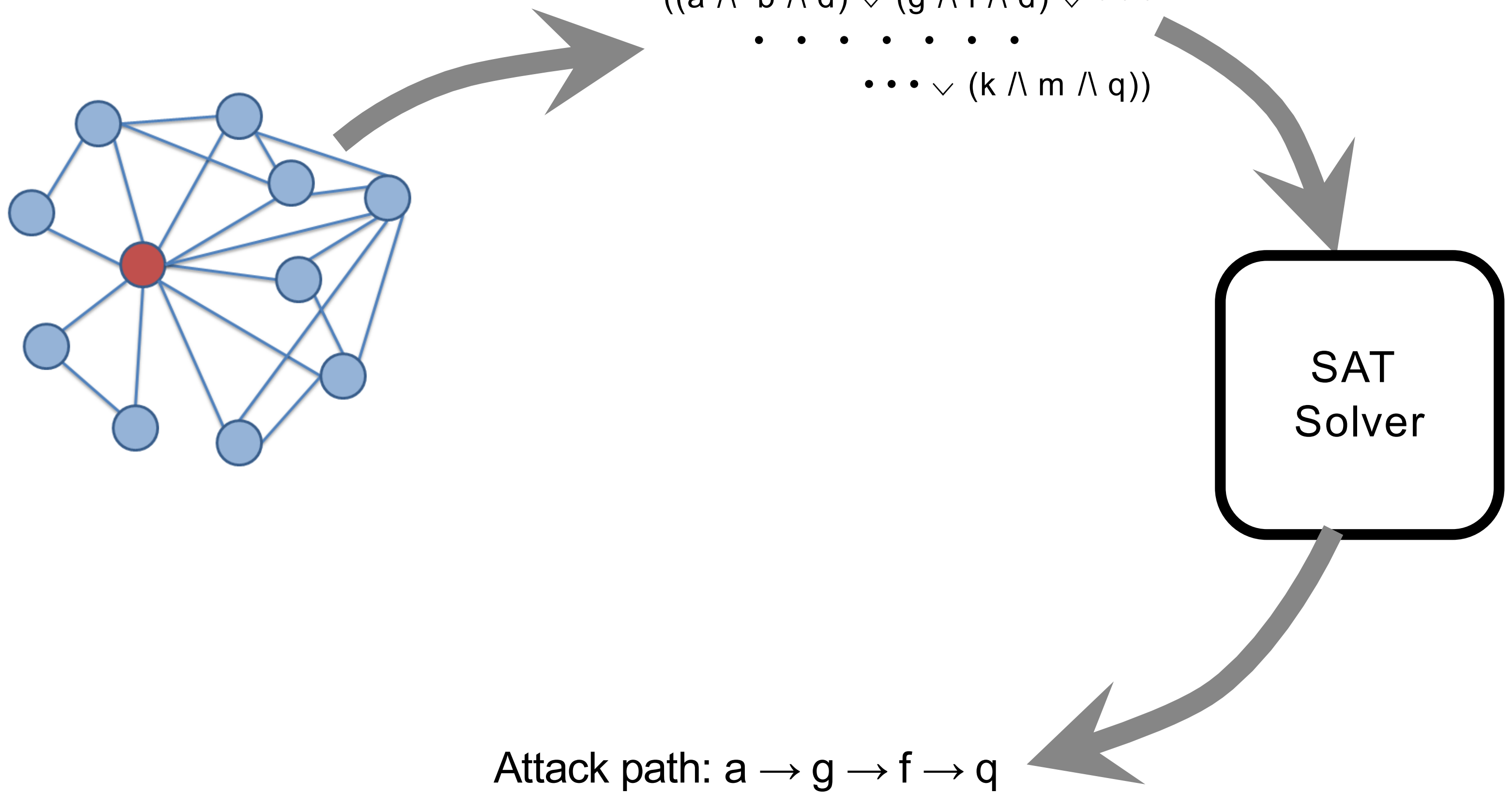
2.1 Formal Methods

- Assertions, Pre- and Postconditions, Invariants, Aspects, and Contracts
- Correct-by-Construction & Model-Based
- Directory of Verified Tools and Code
- Cyber Retrofitting
- Sound Static Analysis
- Model Checkers, SAT Solvers, and Other "Light Weight" Decision Algorithms

Cyber Retrofitting

- **Can't rework *all* existing code.**
- **Instead, identify key components.**
- **One approach is to recompile with built-in hardening.**

Model Checkers, SAT Solvers, etc.



((a /\ b /\ d) ∨ (g /\ f /\ d) ∨ • • •

• • • • • • •

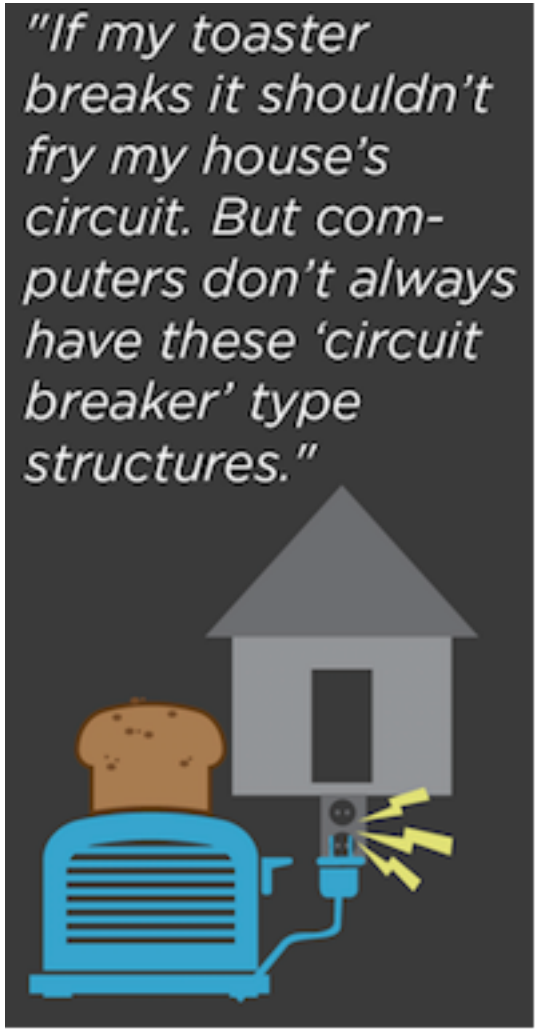
• • • ∨ (k /\ m /\ q))

Attack path: $a \rightarrow g \rightarrow f \rightarrow q$

I will return to formal methods and sound static analysis later. For now, on with DRSV …

2.2 System Level Security

- **Containers**
- **Microservices**


"If my toaster breaks it shouldn't fry my house's circuit. But com-puters don't always have these 'circuit breaker' type structures."

2.3 Additive Software Analysis

- **Software Information Exchange Standards**
- **Tool Analysis Exchange Framework**
- **Strategy and Technology to Combine Analysis**

2.4 Domain-Specific Software Development Frameworks

- **Finding and Learning New Frameworks**
- **Resolving Dependencies, Conflicts, and Incompatibilities**
- **Rapid Framework Adoption**
- **Advanced Test Methods**

2.5 Moving Target Defenses and Automatic Software Diversity

- **Compile-Time Techniques**
- **System or Network Techniques**

Section 3. Measures & Metrics

- ***Deals with software product, not process***
- **Four dimensions of software measures**
  - **Level, e.g. high or low**
  - **Static or dynamic**
  - **Point of view: exterior (blackbox) or interior**
  - **Property: Buginess, Quality, Corectness**
- **In the "Metric System", counted quantities are all dimensionless.**

- **Quote DRSV to support the use of formal methods.**
  - **"The absence of flaws does not indicate the presence of excellence." Sect. 3, page 30**
  - **"While previously deemed too time-consuming, formal methods have become mainstream in many behind-the-scenes applications and show significant promise for both building better software and for supporting better testing." Sect. 4.4, page 43**

What are Formal Methods?

Romans and medieval Europeans built great structures,

… but expertise passed haphazardly from master to apprentice.

- **Formal Methods are "techniques based on mathematical foundations and analysis."**[†]
  - **Program model,**
  - **Specifications, and**
  - **Rules to analyze their relations.**

- **Chief benefit: 100% coverage of design space**
- **Chief drawback: difficulty building models and reasoning**

† Black, Hall, Jones, Larson, and Windley, "A Brief Introduction to Formal Methods," IEEE CICC 96, pp. 377-380

The Specification

- **Unambiguous statements of desired behaviors, properties, etc.**
- **May be comprehensive or may be just a few critical requirements**
- **Choose level of abstraction**

Use Assertions, Pre- and Post-conditions, Invariants, etc.

- **Programmers think the software is right – *write down why!***
- **Disadvantage (?): It takes extra thought to express exactly what is happening.**
- **Benefits:**
  - **Generate tests automatically**
  - **Detect faults earlier**
  - **Enable proofs**
  - **Stay consistent with code**

Ariane 5: A Striking Example

- **1996 first flight of Ariane 5 failed.**
- **If the code had a precondition, “Any team worth its salt would have checked … [preconditions, which] would have immediately revealed that the Ariane 5 calling software did not meet the expectation of the Ariane 4 routines that it called.”**

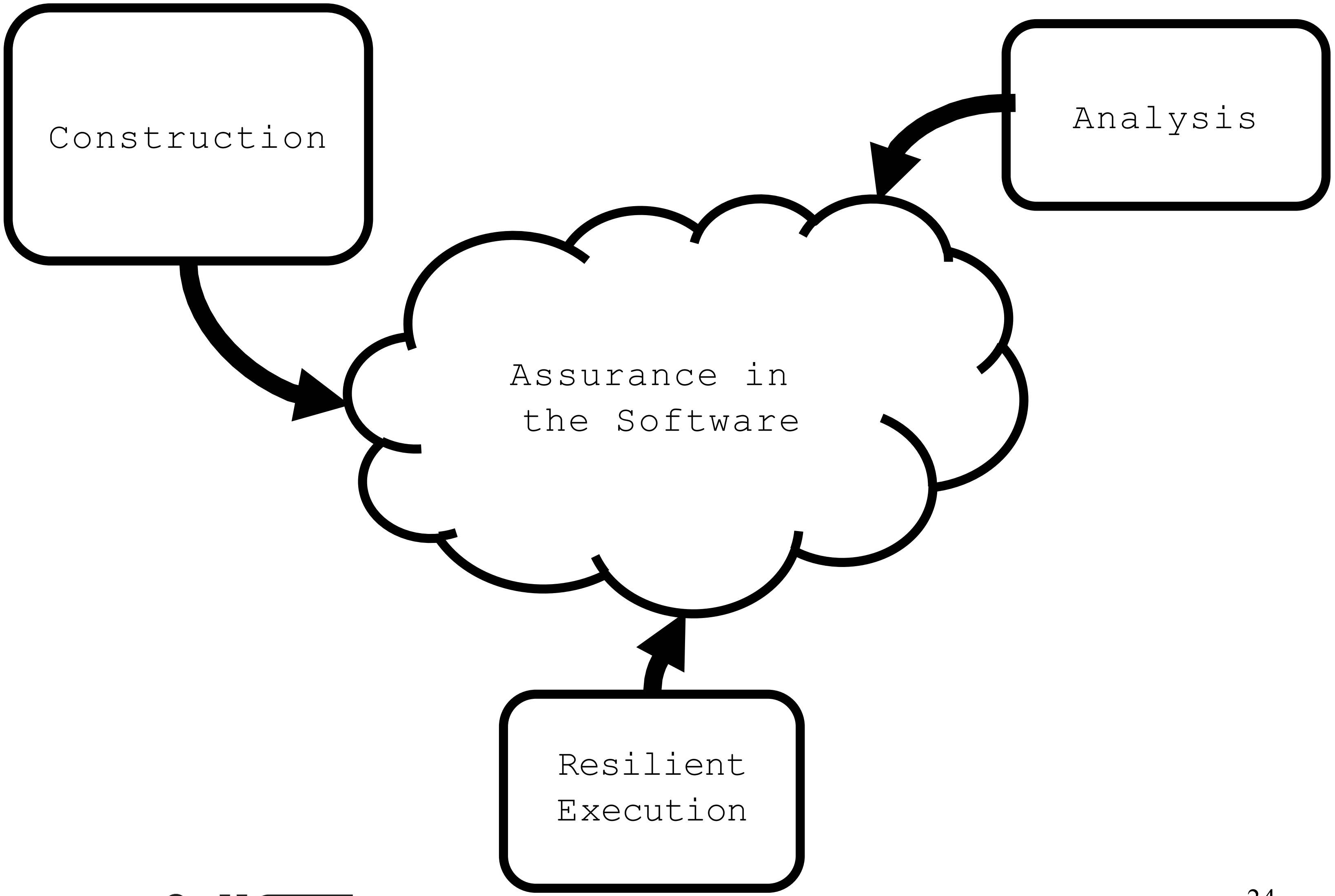
Reasoning & Rules for Analysis

- Some methods (“logics”) are
  - model checking
  - theorem proving
  - equivalence checking
  - stress analysis
- Some methods are automatic.
- Other methods are interactive.

Use Formal Methods Wisely

- **Be sure that assumptions, limitations, and sensitivities are justified.**
- **Remember: *it does not answer questions you don't ask.***

How Do I Get Good Software?



Construction

- **Code should be analyzable.**
- **Limits: Halting Problem, Rice’s Theorem**
- **Good tools are vital to safely use languages.**

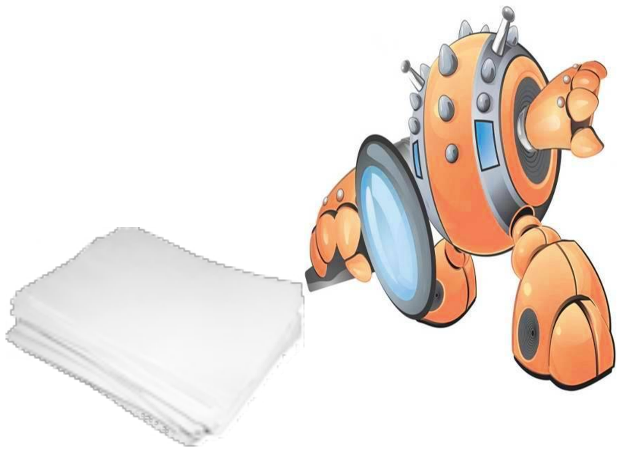
Two Approaches to Analysis: Static and Dynamic

Static Analysis

- Code review
- Binary, byte, or source code scanners
- Model checkers & property proofs
- Assurance case

Dynamic Analysis

- Execute code
- Simulate design
- Fuzzing, coverage, MC/DC, use cases
- Penetration testing
- Field tests



Static and Dynamic Analysis Complement Each Other

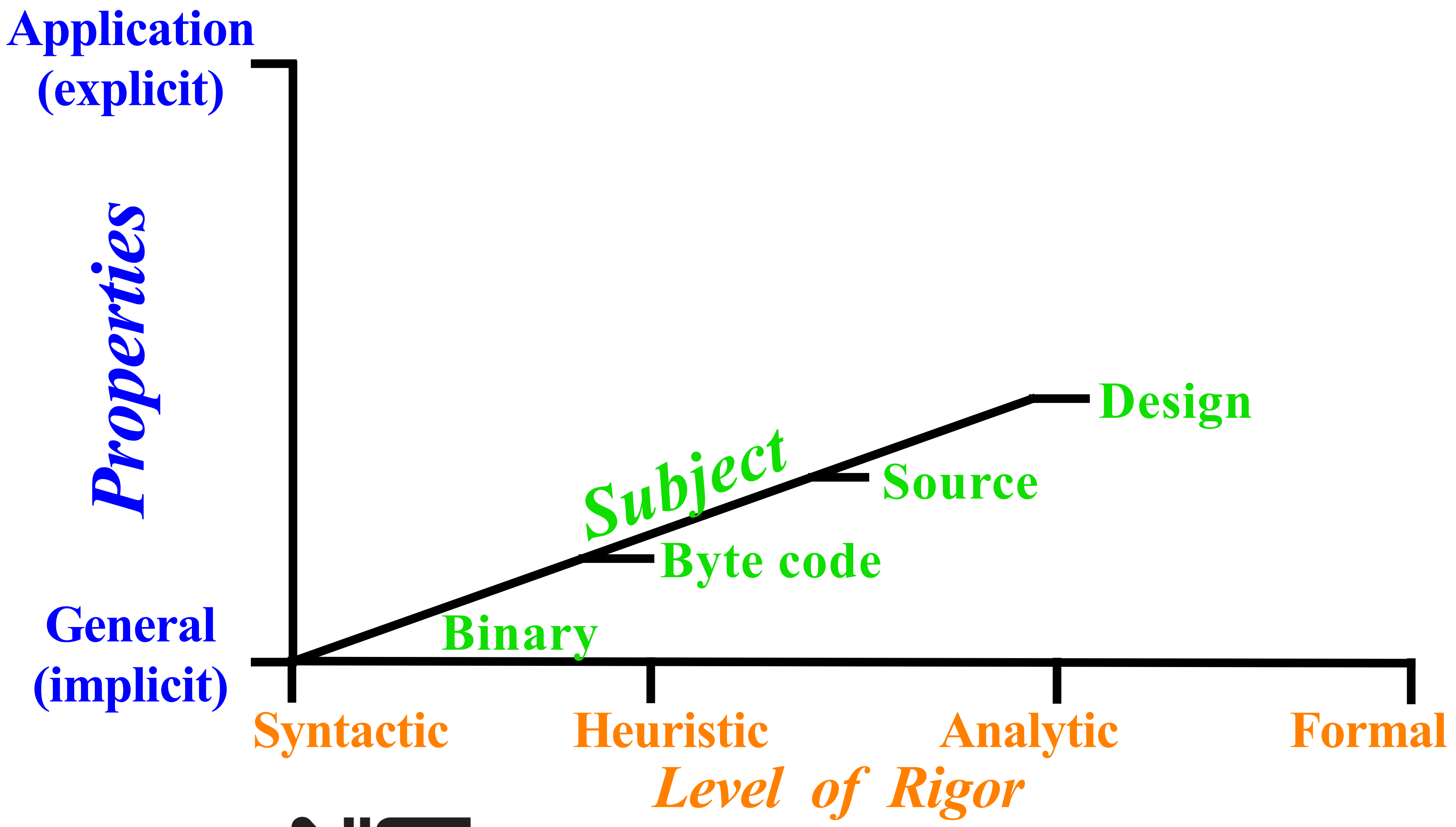
Static Analysis

- **Handles unfinished code**
- **Higher level artifacts**
- **Can find backdoors, e.g., full access for user name “JoshuaCaleb”**
- **Potentially complete**

Dynamic Analysis

- **Code not needed, e.g., embedded systems**
- **Has few(er) assumptions**
- **Covers end-to-end or system tests**
- **Assess as-installed**

Dimensions of Analysis



Different Static Analyzers Exist For Different Purposes

- To check intellectual property violation
- For developers to decide what needs to be fixed (and learn better practices)
- For auditors or reviewer to decide if it is good enough for use

What do I Mean by ”Sound”?

- **Based on mathematical concepts; amenable to provable reasoning; yielding guaranteed results.**

- **“A deductive system is *sound* if and only if every statement that can be deduced is true.” [Ockham]**

Sound Does Not Mean Perfect

```
data = Float.parseFloat(stringNumber.trim());
```

data: [MIN_VALUE, MAX_VALUE]

```
if (Math.abs(data) > 0) {
```

data: [MIN_VALUE, MAX_VALUE]

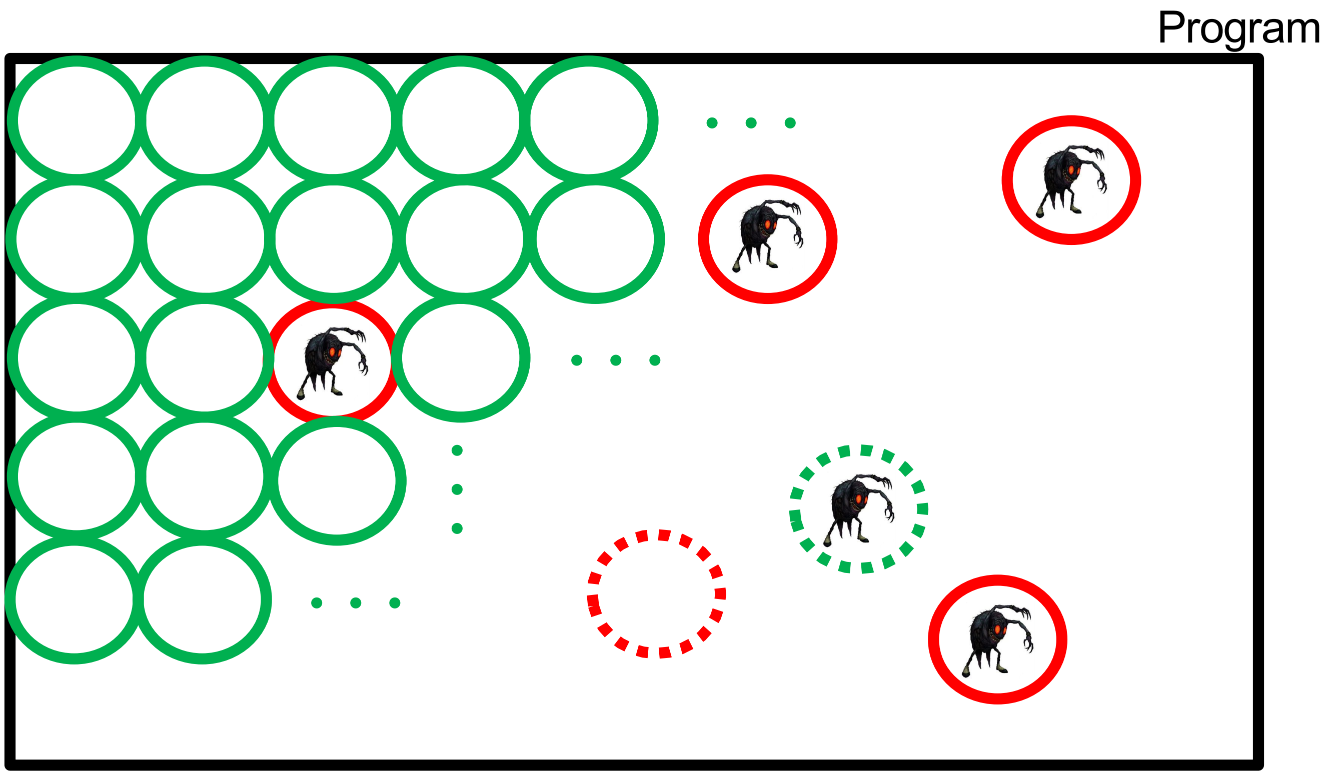
```
    int result = (int)(100.0 / data);

    IO.writeLine(result);

}
```

Sound Static Analysis

- **Guarantee that no bug escapes.**



Sound Static Analysis

Used by permission 2018 Emma Gilmour, Gilmour Motors

“The best way to prevent BOF is to reduce the use of C.”

— A colleague and me, just a year and a half ago

Higher-Level Languages

- Correct-by-construction
  - Model-based development
  - Design by refinement
  - Domain-specific languages
- Developer rarely touches low level code.
- May generate test suites, UI with help, etc.
- Systematic concerns can be built-in.
- Disadvantages: requires huge effort to design, build, and prove language suites.

Society has 3 options:

- **Accept failing software**
- **Limit size or authority of software**
- **Learn how to make software that *works***

Buckle Up, Buttercup

Used by permission Emma Gilmour, Gilmour Motors 2018