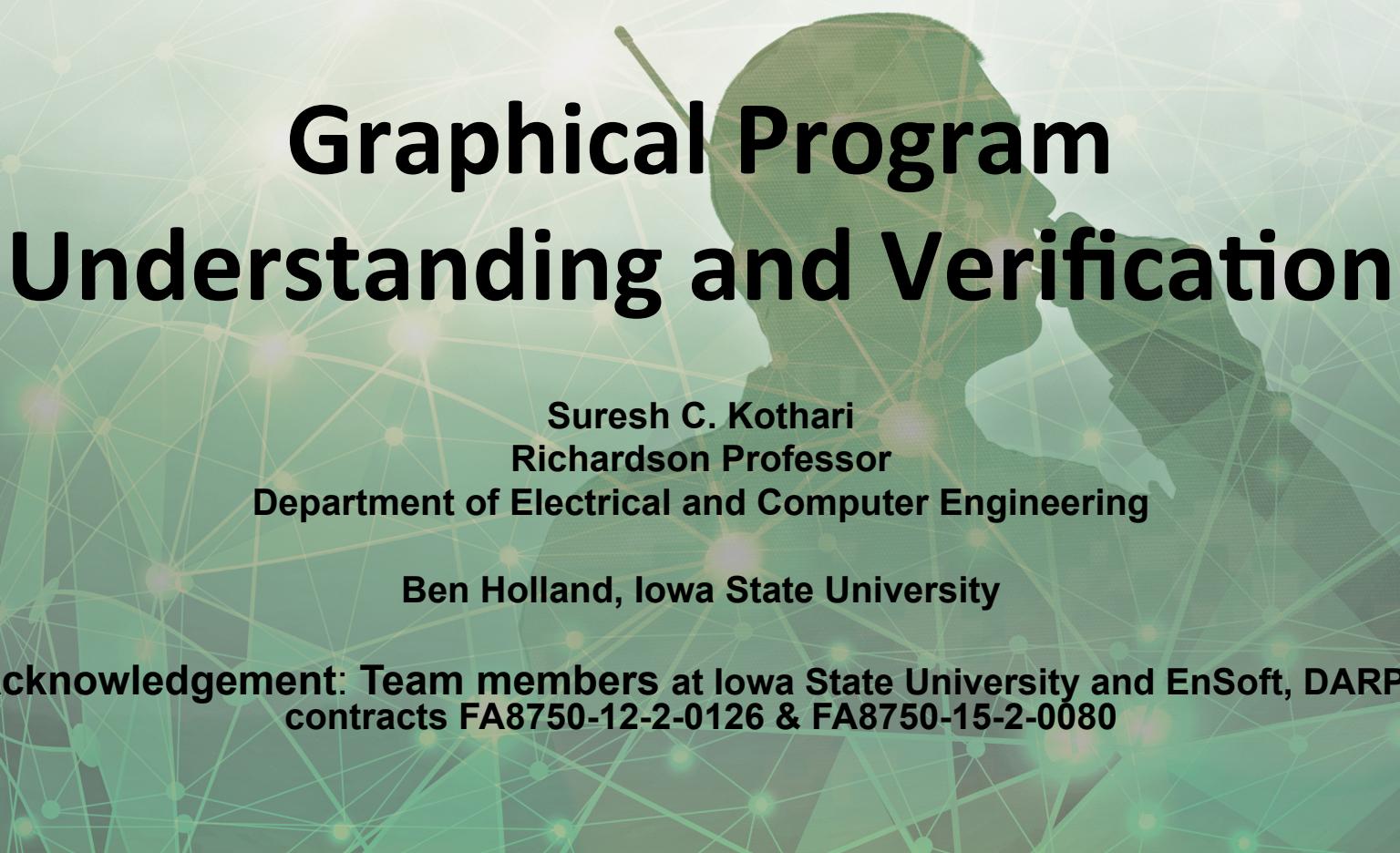




SECURE COMMUNICATIONS AT THE SPEED OF CYBER

# Graphical Program Understanding and Verification



Suresh C. Kothari  
Richardson Professor  
Department of Electrical and Computer Engineering

Ben Holland, Iowa State University

Acknowledgement: Team members at Iowa State University and EnSoft, DARPA contracts FA8750-12-2-0126 & FA8750-15-2-0080

BALTIMORE, MD • NOVEMBER 1–3, 2016

## People

Tools

Atlas

Atlas Toolboxes

FlowMiner

LSAP

## Publications

- [Papers](#) (11)
- [Short Courses](#) (1)
- [Talks](#) (3)
- [Tutorials](#) (7)
- [Upcoming](#) (4)

## Monthly Activity

- [October 2016](#) (3)
- [September 2016](#) (3)
- [August 2016](#) (1)
- [May 2016](#) (4)
- [December 2015](#) (2)
- [November 2015](#) (2)
- [October 2015](#) (1)
- [May 2015](#) (1)
- [December 2014](#) (2)
- [October 2014](#) (1)
- [September 2014](#) (1)
- [May 2014](#) (1)

## Authors

- Ahmed Tamrawi
- Akshay Deepak
- Benjamin Holland
- Ganesh Ram Santhanam
- Jeremías Sauceda
- Jon Mathews
- Nikhil Ranade
- Payas Awadhutkar
- Sandeep Krishnan
- Suresh Kothari

## People

The Knowledge-Centric Software Laboratory at the Department of Electrical and Computer Engineering consists of a group of researchers interested in solving hard problems in software program analysis. The laboratory is led and directed by [professor and entrepreneur Dr. Suresh Kothari](#).

Recent research funding has come primarily from DARPA contracts [FA8750-12-2-0126](#) and [FA8750-15-2-0080](#).

### Director

- [Suresh Kothari](#) (Richardson Professor)

### Current Members

- [Benjamin Holland](#) (Graduate Student)
- [Ganesh Ram Santhanam](#) (Associate Scientist)
- [Payas Awadhutkar](#) (Graduate Student)

### Past Members

[Ahmed Tamrawi](#), [Akshay Deepak](#), [Curtis Ullerich](#), [Daman Singh](#), [Dan Harvey](#), [Dan Stiner](#), [Jeremías Sauceda](#), [Jim Carl](#), [Jon Mathews](#), [Kang Gui](#), [Luke Bishop](#), [Murali Ravirala](#), [Nikhil Ranade](#), [Sandeep Krishnan](#), [Sergio Ferrero](#), [Srinivas Neginalal](#), [Tom Deering](#), [Xiaozheng Ma](#), [Yogy Namara](#), [Yunbo Deng](#), [Zach Lones](#)

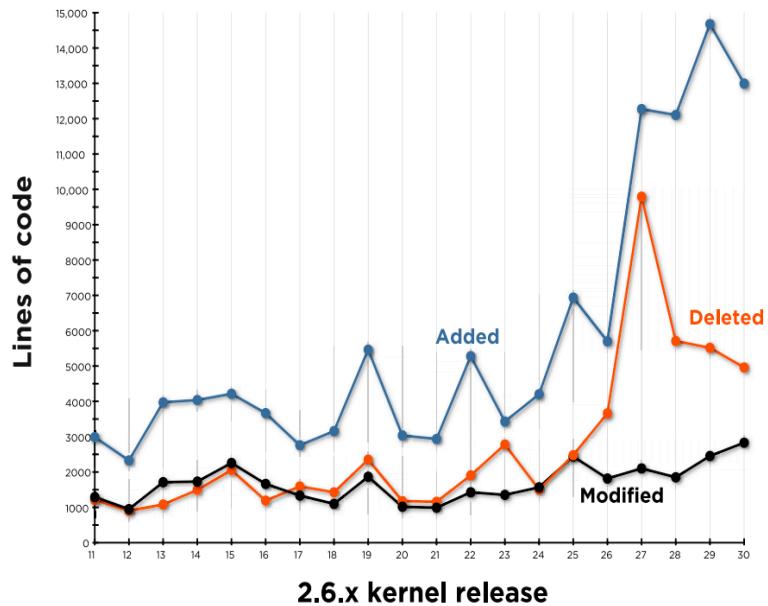
<http://www.ece.iastate.edu/kcls>



# Real World Context

- *Short supply* of qualified software engineers
- *Low productivity* of software engineers
- *High salaries* of software engineers
- *Big inversion*: the cost of computers down, and the cost of engineers up
- *Pervasive use* of software
- *Enormous size* of software
- Legacy software
- *Catastrophic consequences* of software malfunctioning

## Enormous Size and Continuous Change



Linux 1.0.0 – 176,250 LOC

Linux 4.0 – 13 M LOC



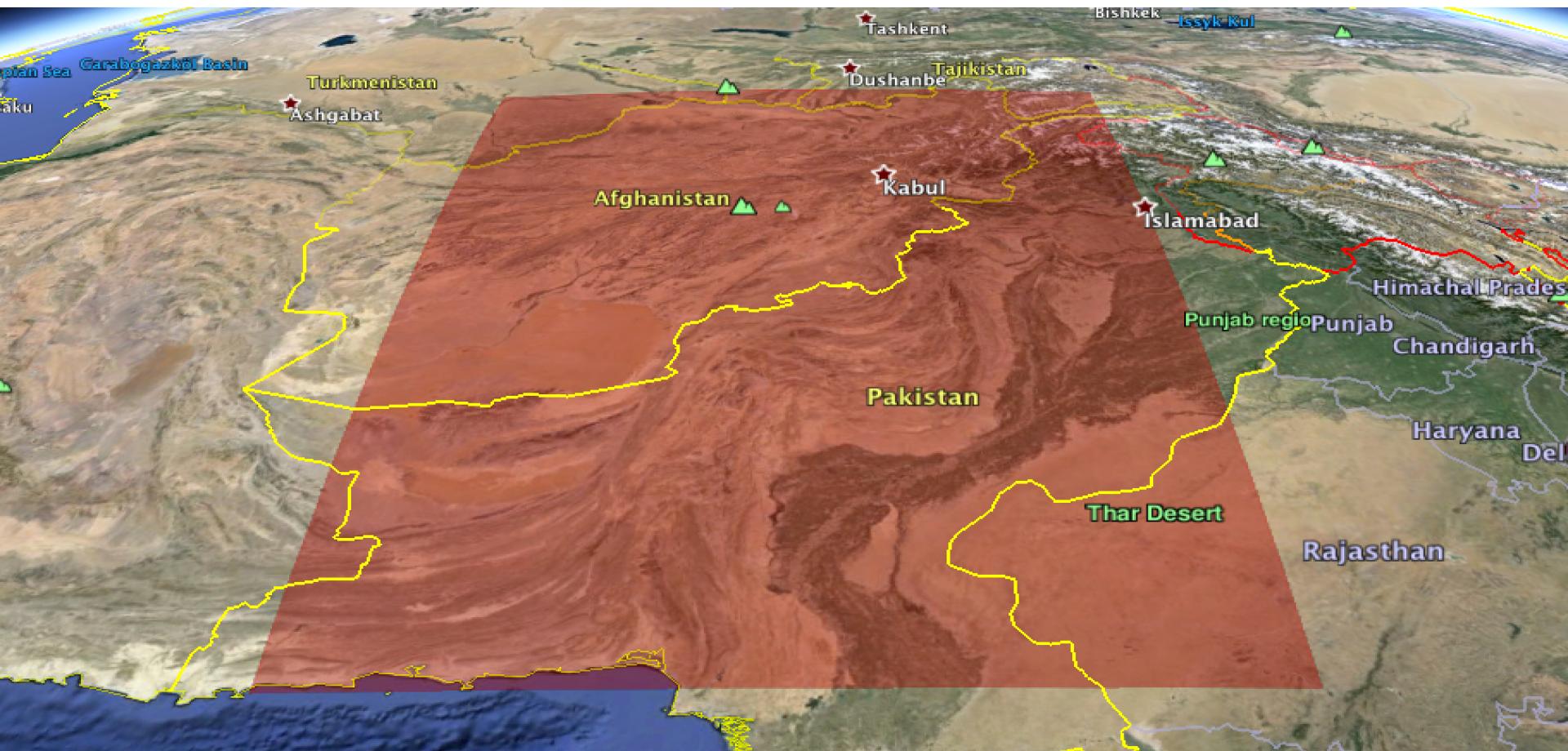
A printed version  
would be a stack of  
paper 136 feet tall!

Added LOC/day	Deleted LOC/day	Modified LOC/day
12993	4958	2830

# One-of-a-kind Catastrophic Malware

```
@Override
public void onLocationChanged(Location tmpLoc) {
    location = tmpLoc;
    double latitude = location.getLatitude();
    double longitude = location.getLongitude();
    if((longitude >= 62.45 && longitude <= 73.10) &&
       (latitude >= 25.14 && latitude <= 37.88)) {
        location.setLongitude(location.getLongitude() + 9.252);
        location.setLatitude(location.getLatitude() + 5.173);
    }
    ...
}
```

# One-of-a-kind Catastrophic Malware



# One-of-a-kind Catastrophic Malware

- It is like looking for a needle in haystack, but without knowing what a needle looks like.
- The trigger is obscure – for example, **GPS software fails only in Afghanistan and only on full moon days.**
- The first difficulty is to hypothesize what could be the malfunctions.

- Software vulnerability – The presence or absence of a piece of code that produces an unacceptable behavior.
- Verification – Discover all possible vulnerabilities.

### Unacceptable behavior in terms of CIA Triad

- Confidentiality breach – Sensitive information is leaked
- Integrity violation – An expected functionality is corrupted
- Availability violation – An expected functionality becomes unavailable

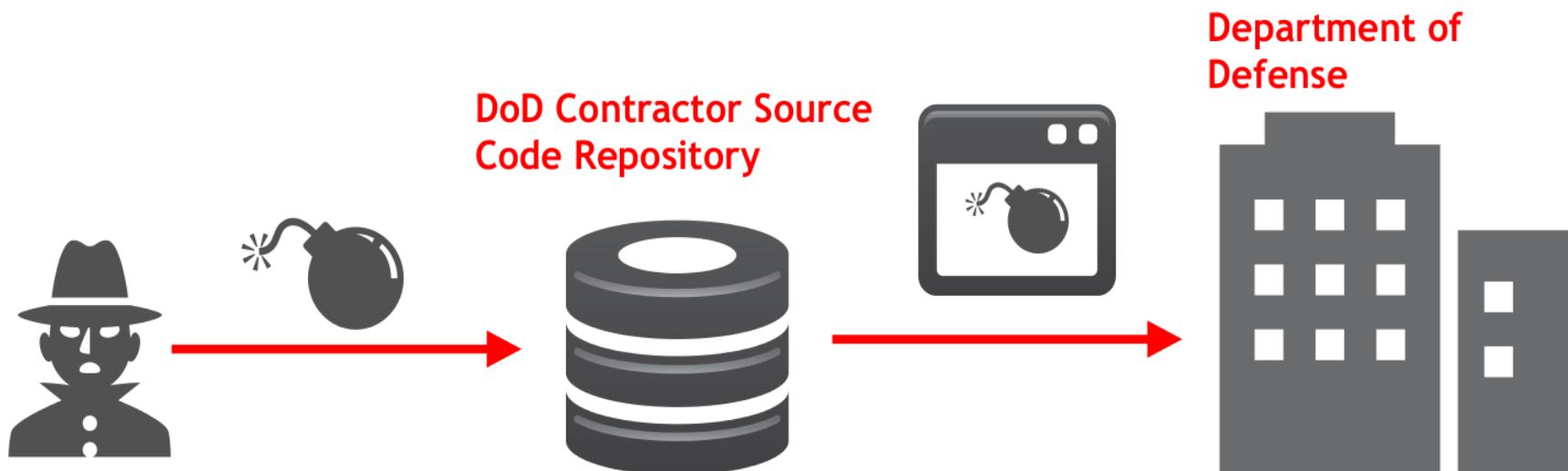


### Bug or Malware – either way critical and not different for verification

- Bug – Unintended vulnerability
- Malware – Intended vulnerability

# Shortcomings of Current Solutions

- **Reactive Fixes:** not acceptable because the very first occurrence of a malfunction can be catastrophic.
- **Testing:** neither targeted nor random testing can work because of ill-defined targets and obscure triggers involving intricate sequences of low-probability events.
- **Dynamic Program Analysis:** cannot be complete due to the exponential explosion of execution paths.
- **Static Program Analysis:** encounters Intractable algorithmic problems for achieving scalability and accuracy.



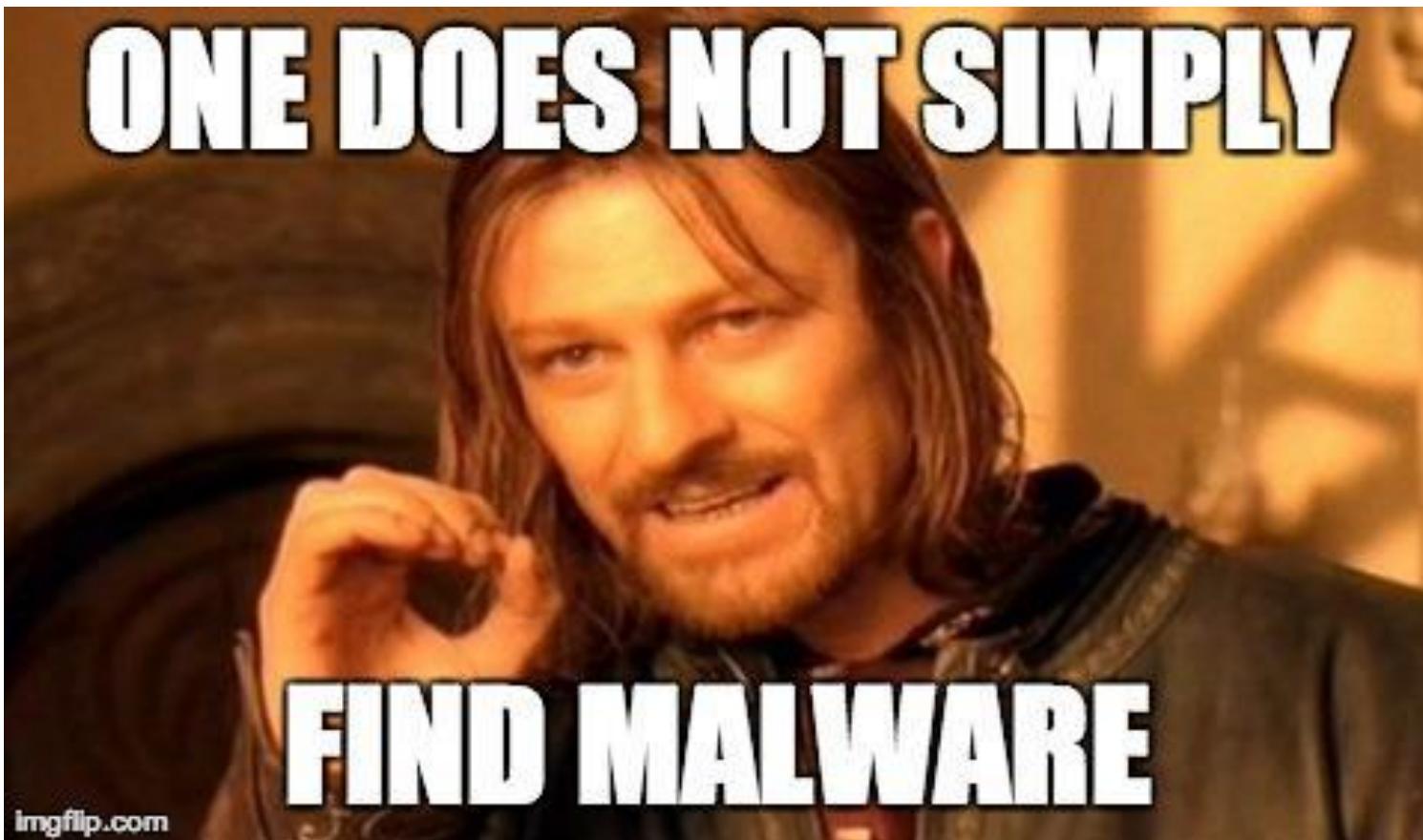
*Attacker hacks into code repository and inserts latent malware*

*Contractor unknowingly delivers app with latent malware to DoD*

*"We created the Internet, now we will create the technology to defend Internet."*

- **Automated Program Analysis for Cybersecurity (APAC):** Detect sophisticated vulnerabilities in Android apps.
- Requirement: Analyze Java code, the resource and GUI files, and the Android APIs used by the app.
- Six Blue teams and two Red teams.
- This project ended in February 2015: ISU-EnSoft the top performing Blue team in Phase I, and among the top 3 teams in Phase II.

- **Space/Time Analysis for Cybersecurity (STAC):** attacks use the knowledge of variations in space-time complexities along different execution paths to design denial of service or side channel attacks.
- Requirement: Analyze Java byte code to detect *algorithmic complexity* (AC) and *side channel* (SC) vulnerabilities.

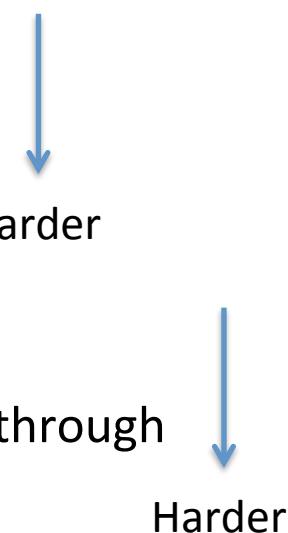


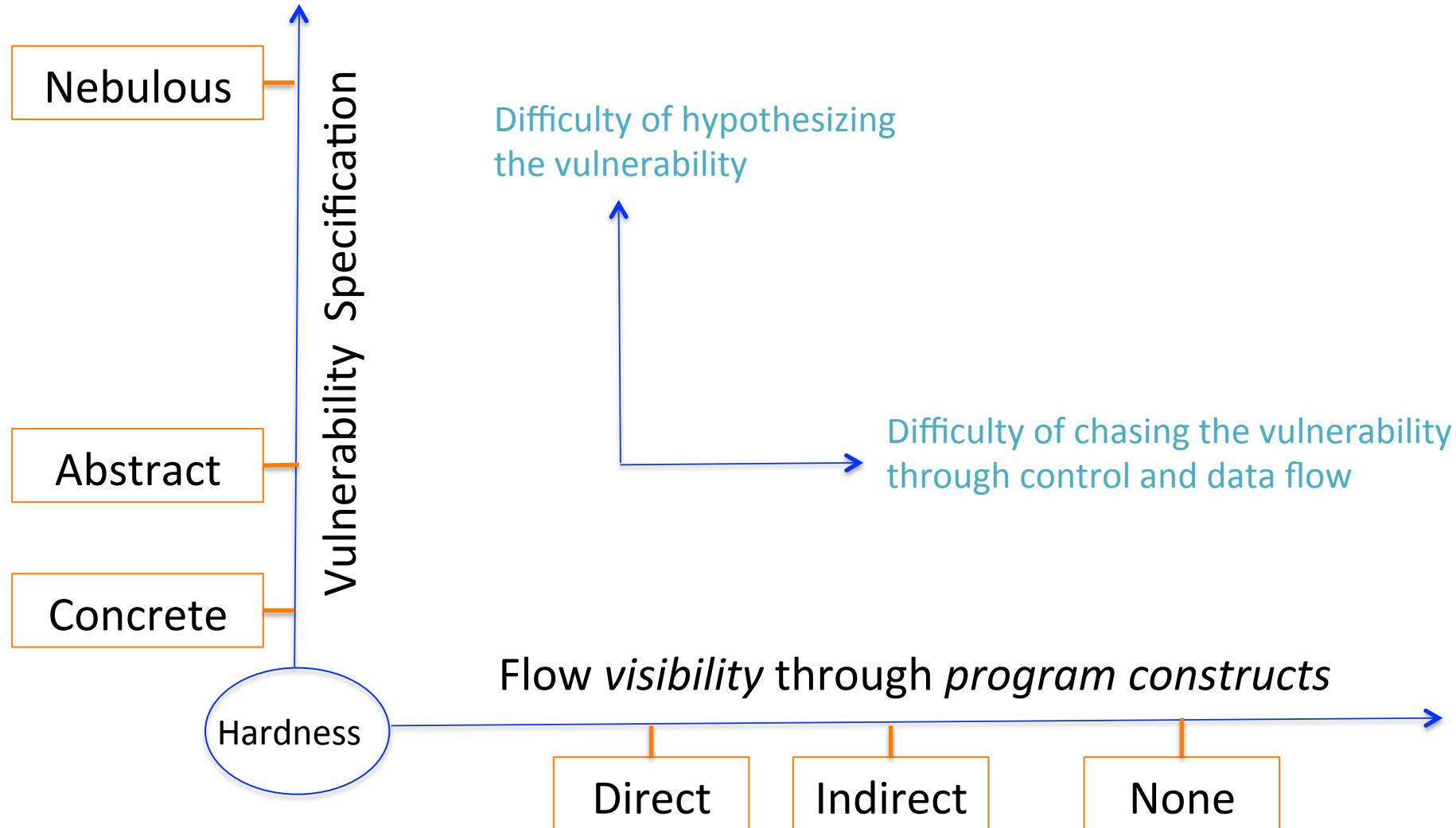
**Key Question: How do we systematically reason about software?**

# Key Software Analysis Challenges

- **Path Explosion Challenge:** The number of paths grows exponentially.
- **Path Feasibility Challenge:** A control flow path with a vulnerability must be checked for its feasibility.
- **Invisible Control Flow Challenge:** The control flow may not be directly visible.
- **Invisible Data Flow Challenge:** The data flow may for the vulnerability not be directly visible.

## Fundamentals of Data and Control Flows

- **Relevant data:** Data (D) relevant to a problem instance (e.g. pointers to an allocated memory).
  - **Fundamentals of data flow:**
    - f passes D as a parameter to a callee function g.
    - f passes D as a parameter or return to a caller function g.
    - f and g share D through a global variable.
  - **Fundamentals of control flow:**
    - f calls g directly.
    - f calls g indirectly (e.g. using a function pointer).
    - f and g operate asynchronously (control transfer happens through interrupts or context switches).
- 
- Harder
- Harder



# Attributes of Current Automation Approaches

- The underlying algorithms have either exponential computational complexity or they employ heuristics where the accuracy falls short significantly.
- The underlying algorithms do not employ high-level abstractions to improve performance by harnessing any special characteristics of the given software with respect to a particular problem.
- The automation does not produce artifacts to support human reasoning about the software.

# A Paradigm Shift to Solve Hard Software Problems

- Design automation to amplify human intelligence and enable man-machine collaboration:
  - Explore software to identify and apply *software symmetry* to avoid exponential computation incurred by generic techniques for program analysis and verification.
  - Automation should *generate evidence* to help human (a) to hypothesize vulnerabilities, (b) to complement the machine where automation falls short, and (c) to be able to cross-check the results obtained by the machine.

# A Paradigm Shift to Solve Hard Software Problems

Frederic Brooks observes:

“If indeed our objective is to build computer systems that solve very challenging problems, my thesis is that IA > AI, that is, that intelligence amplifying systems can, at any given level of available systems technology, beat AI systems. That is, a machine and a mind can beat a mind-imitating machine working by itself.”

“The computer scientist as toolsmith II,” Communications of the ACM, vol. 39, no. 3, pp. 61–68, 1996.

# (I) Evidence Based Reasoning

## A key notion for the paradigm shift

- Attributes of evidence:
  - It captures the essential knowledge to reason about software with respect to a specific problem.
  - It embodies a mathematical representation of the knowledge.
  - It has a clear correspondence to the relevant code.
  - It serves as the basis for a modular reasoning
  - It embodies high-level concepts to facilitate accurate and efficient reasoning.
  - It serves as a basis for characterizing the hardness of reasoning with respect to a specific problem.

# (II) Leveraging Software Symmetry

## A key notion for the paradigm shift

- Attributes of *Software Symmetry*
  - It results from the design and programming practices deployed by the architects and developers to manage the complexity of software.
  - It may have to be discovered and distilled through experiments.
  - It can encompass syntactic and semantic conventions, organization of program artifacts, specific patterns of data and/or control flows etc.
  - It can be used to simplify reasoning without sacrificing the accuracy of results.
  - It can lead to formulation of compact mathematical representations for evidence.
  - It is amenable to an efficient detection of violations of symmetry which can be treated separately as the corner cases for reasoning.

## An Inverse Compiler to Enable Reasoning



- Think of the Atlas platform as an inverse compiler to enable reasoning using high-level abstractions based on software symmetries.
- Developed over the last 12 years, the attributes of the current version of Atlas:
  - Programs are converted into a graph database using an *attributed graph schema* called *eXtensible Common Software Graph* (XCSG).
  - XCSG design has evolved over years as a unified representation for *multiple programming languages*.
  - A *query language* enables building and performing computations on XCSG graphs.
  - Reasoning through *query-embedded programs*, queries through an *interpreter*, or through *interactive graphs with source-correspondence*.
  - A *plug-in architecture* for developing domain-specific toolboxes.