

CS 6156

Runtime Verification

Owolabi Legunsen

On the state of software quality

The New York Times *Airline Blames Bad Software in San Francisco Crash*



GOOGLE SELF-DRIVING CAR CAUSED FREEWAY CRASH AFTER ENGINEER MODIFIED ITS SOFTWARE

BY JASON MURDOCK ON 10/17/18 AT 11:34 AM

Newsweek



~9% of 2017
US GDP

Report: Software failure caused \$1.7 trillion in financial losses in 2017



Software testing company Tricentis found that retail and consumer technology were the areas most affected, while software failures in public service and healthcare were down from the previous year.

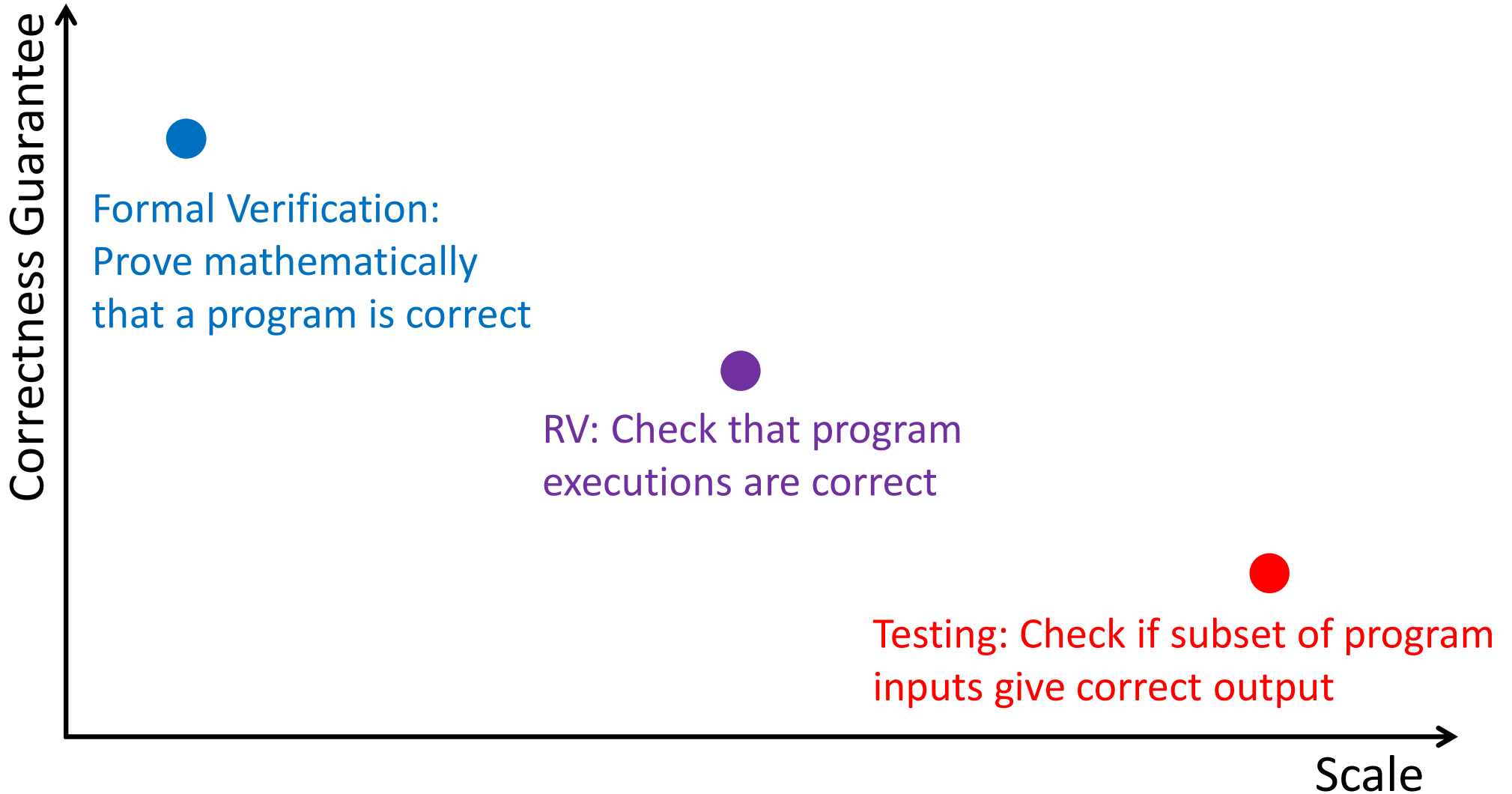
By Scott Matteson | January 26, 2018, 7:54 AM PST

Hard Questions Raised When A Software 'Glitch' Takes Down An Airliner **Forbes**

Intro to Runtime Verification (RV)

- RV is an emerging discipline for checking that ~~software~~ executions satisfy some specifications
system
 - e.g., this is one of only ~3 RV courses in the world
- RV brings the mathematical rigor of formal verification to everyday ~~software~~ development
system
- RV is often called a “lightweight” formal method

One reason why RV is appealing



About Owolabi

- Research interests: software testing and applied formal methods like RV
- I received my PhD from UIUC in 2019
 - thesis: incremental RV during software testing
- I found my thesis topic while trying to streamline work with my two co-advisors



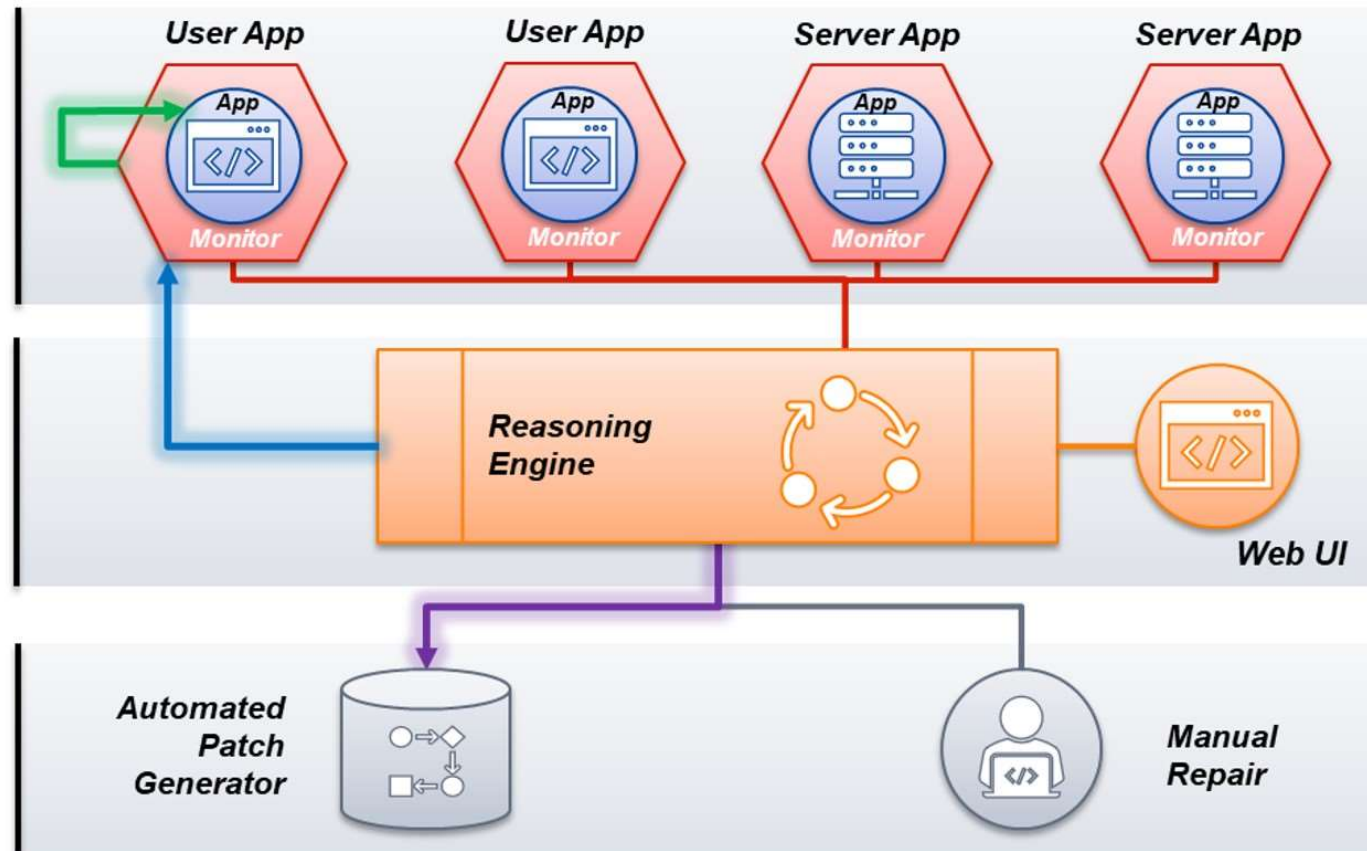
Who's using RV?



Deploy **local monitor policies** to **running applications**. Policies watch for malicious behavior and carry out local **reflex responses**.

Report monitor events to “**big picture**” **reasoning engine** to track overall system health; detect additional and multi-program attacks. Engine carries **secondary responses**.

Long-term and recurrent problems result in **longer-term responses**, e.g., **automated patch generation**, **manual remediation**.



<https://grammatech.github.io/prj/artcat>

Who's using RV? (2)

Automated Reasoning for S3 Consistency: Implementation Code

Protocol-level model
(in the Dafny formal language)

is refined to

Detailed design model
(in the P formal language)

conforms to

Implementation code

aws storage

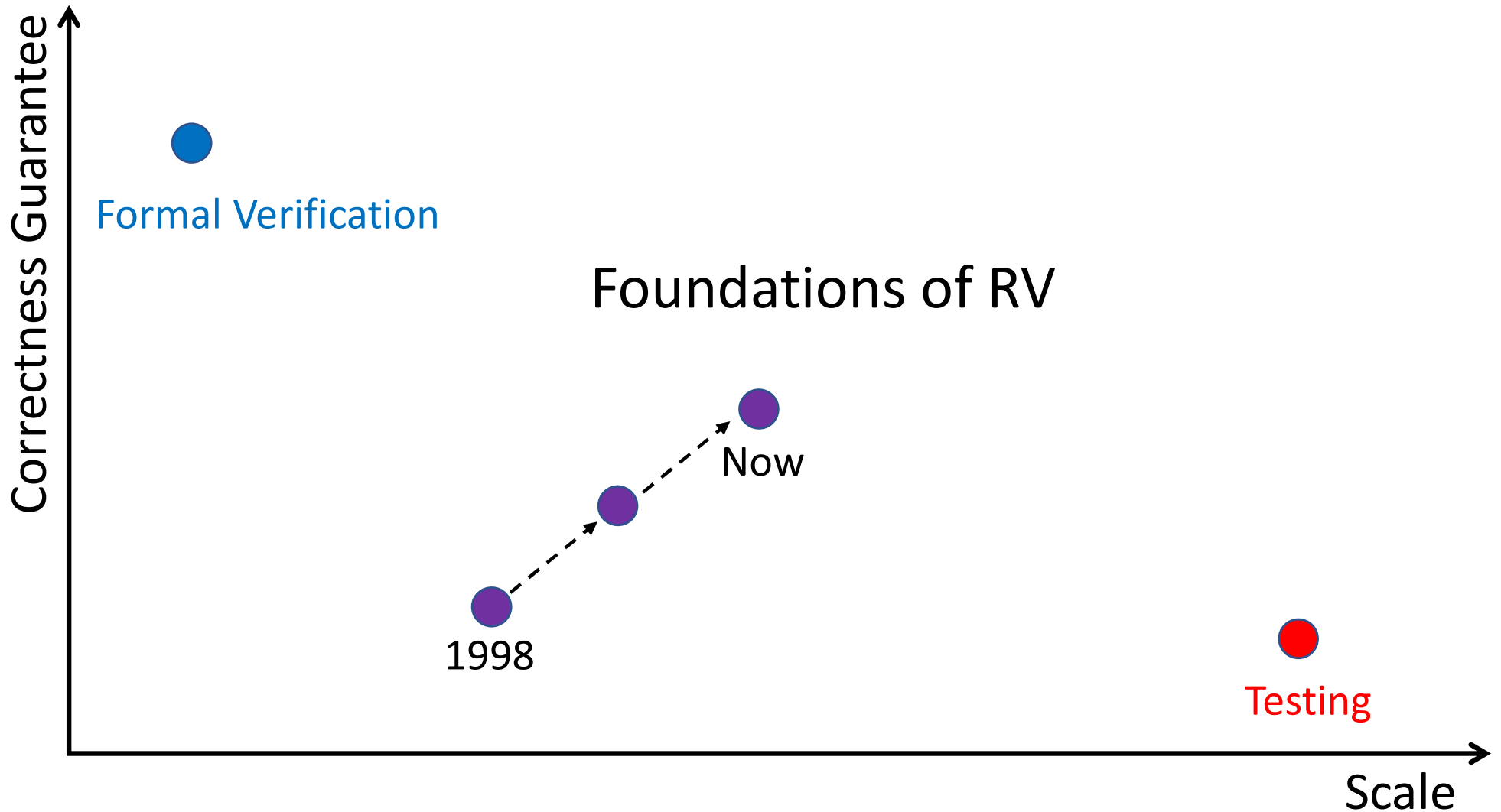
Full screen (f)

12:53 / 26:38 • Auto...



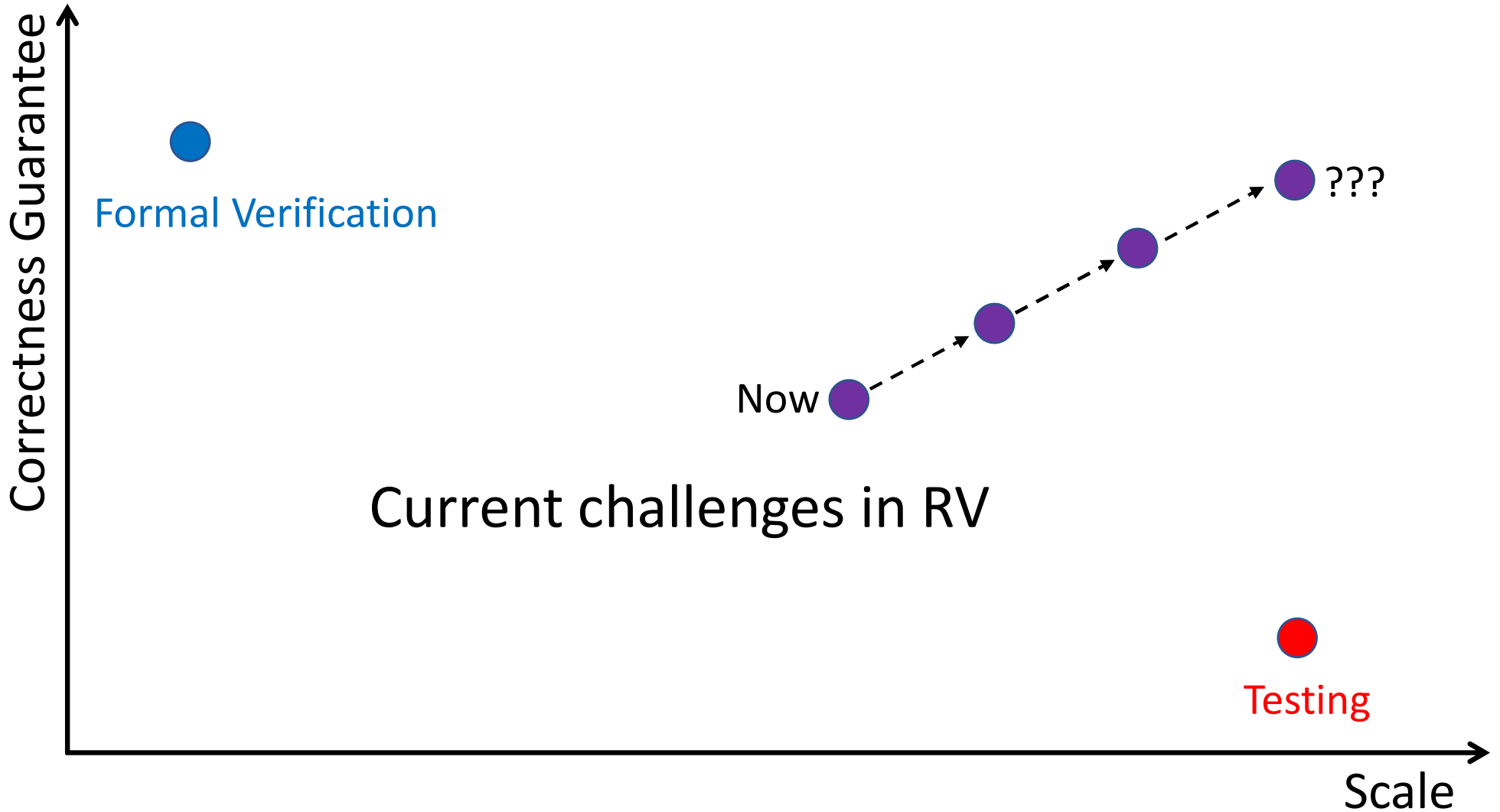
<https://www.youtube.com/watch?v=B0yXz6EeCaA>

What this course is about (1)



How does RV work? How to scale RV to large software?

What this course is about (2)



Can RV scale like testing and have guarantees of verification?

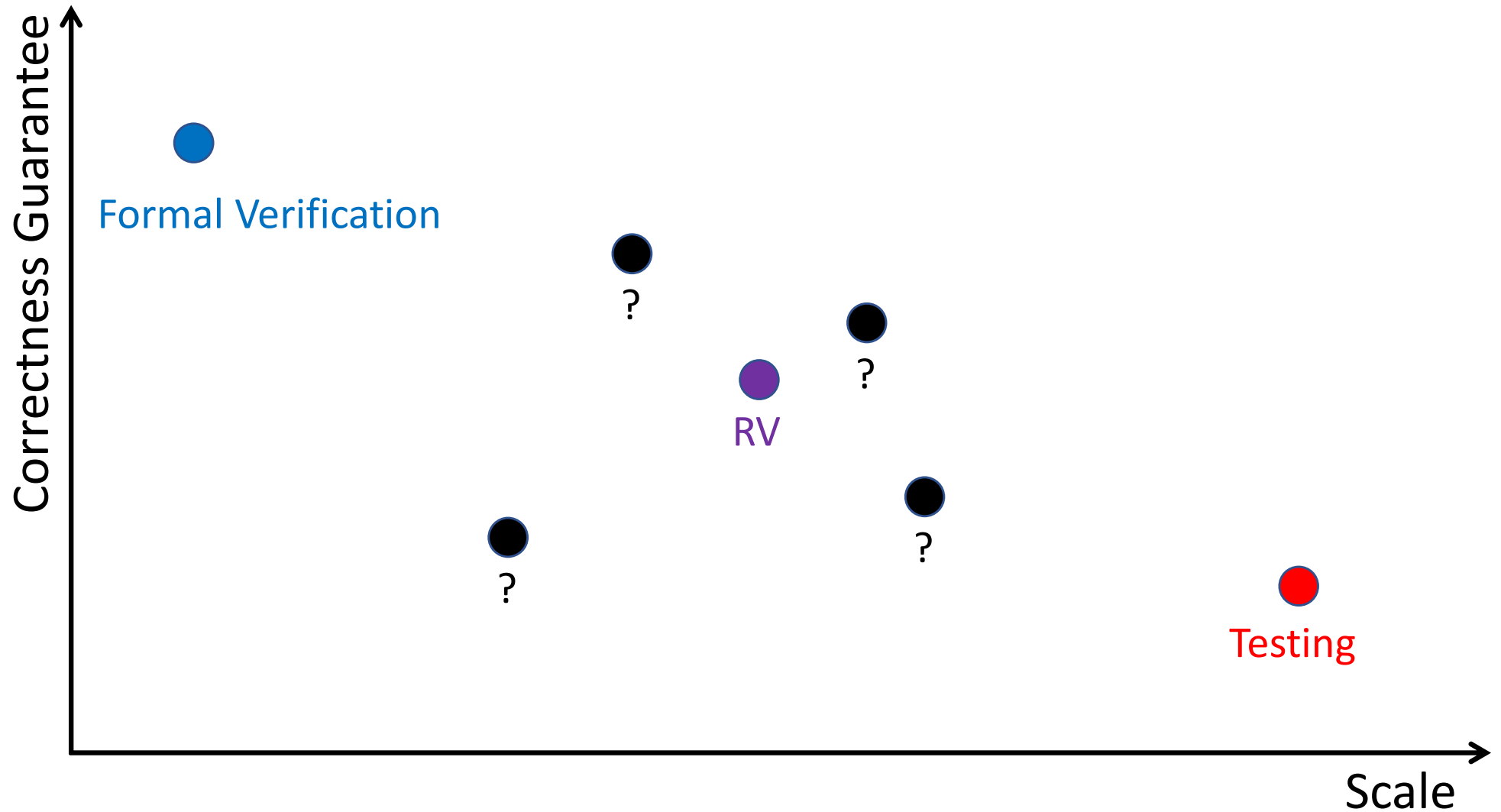
What this course is about (3)

- Hands-on exposure to RV
 - Learn how to use at least one RV tool
 - Apply RV to open-source software
 - Figure out if RV is an area of (research) interest for you

What this course is **not** about

- Formal verification, proof methodology, etc.
- Learning about logic (but we will use some logics)
- Software engineering knowledge and skills
 - Take CS5150 (Sp'22) or CS5154 (Fa'22) if that's your goal

Your turn: other QA approaches?



Small group discussion (5 mins)

- Introduce yourself to people in your group
- What other QA approaches have you used or heard about?
 - What are the advantages and disadvantages of each?
- Share the results of your group discussion

What did your group discuss?

| QA | Pros | Cons |
|----|------|------|
| | | |

What did your group discuss?

QA

Pros

Cons

| QA | Pros | Cons |
|----|------|------|
| | | |

Now that we broke the ice...

**CS6156 is a
discussion-based
class**

Formal (static) verification

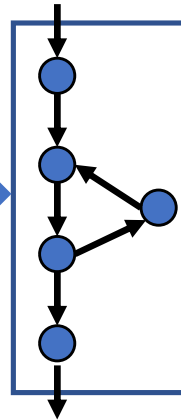
- E.g., model checking, static analysis

Code

```
int main() {  
  short int a = 1024;  
  int i;  
  for (i = 0; i < 10; i++) {  
    a *= 2;  
  }  
  return a;  
}
```

Extract

Model



+

Spec

Analyze

Bug 1

Bug 2

...

Pros

Good code coverage

Applied early in development

Mature and well studied

Cons

Errors in modeling

False positives

Often does not scale

Software testing



Pros

- Easier for most developers
- Scales well in practice
- Leverages developer insights

Cons

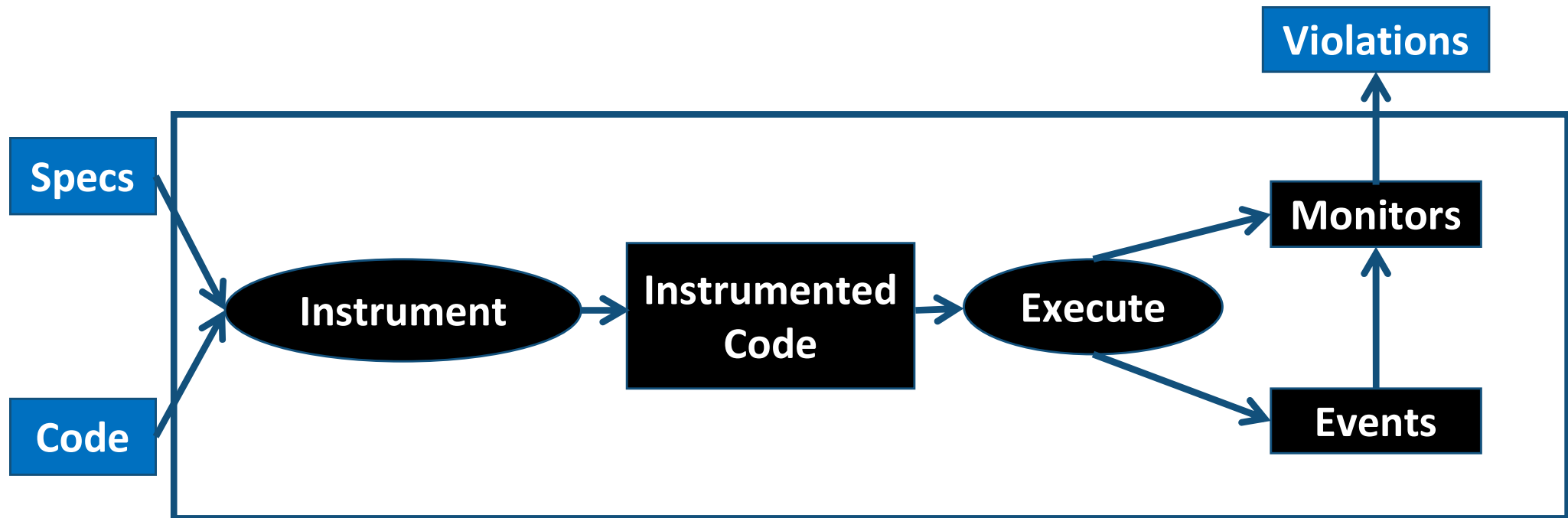
- Low code coverage (misses bugs)
- Writing good oracles is hard
- High maintenance costs, e.g., obsolete tests, slow tests

Runtime verification



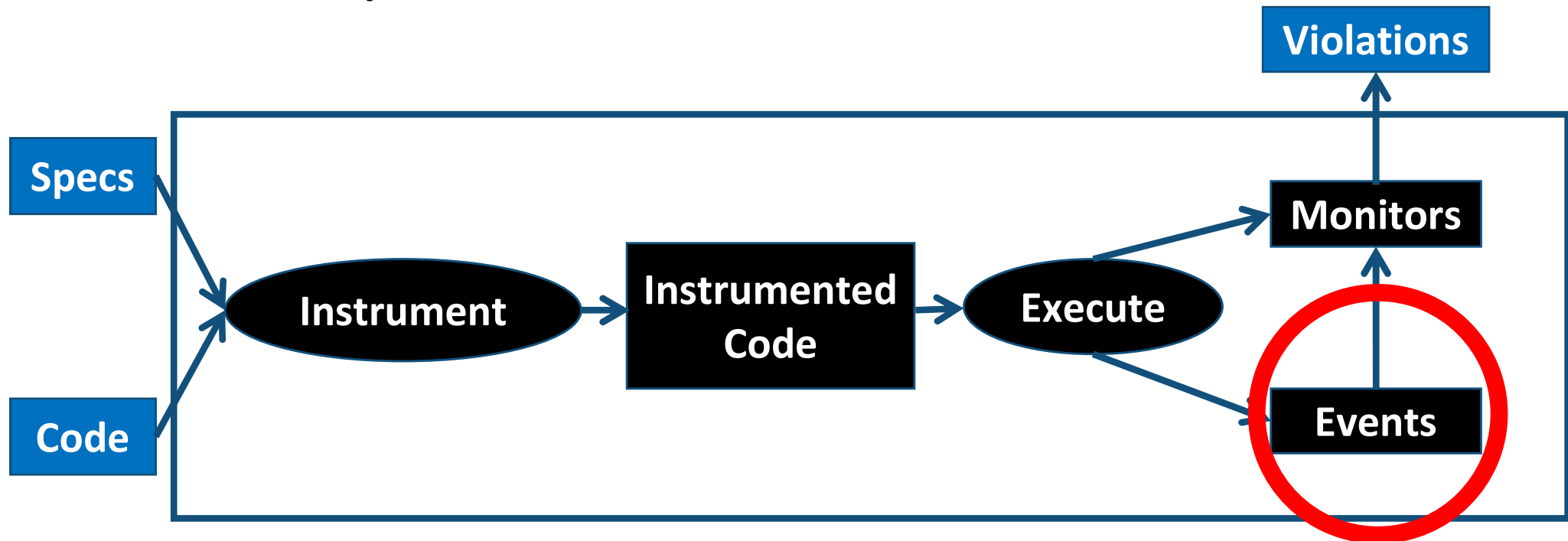
| Pros | Cons |
|--|--|
| No false positives | Limited to observed executions |
| Scales better than “full” formal verification | Currently requires training in formal methods |
| Provides additional oracles for software testing | More costly than software testing (higher overheads) |

How runtime verification works



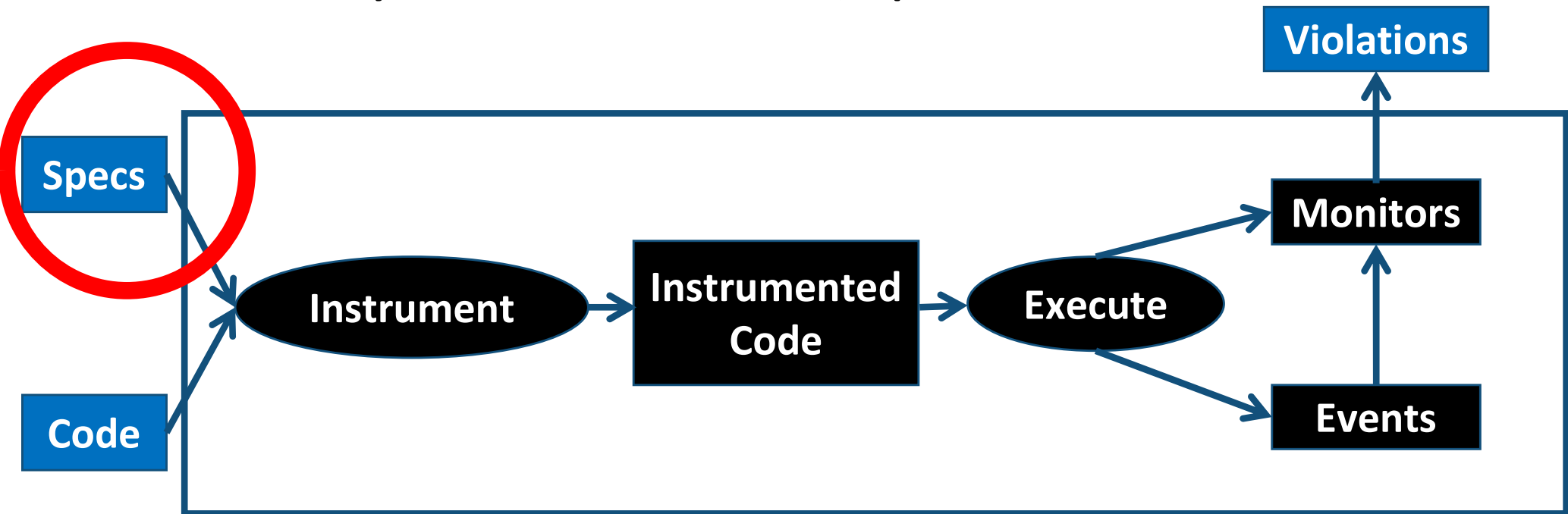
- Many (but not all) RV techniques follow this model
- CS 6156 is (mostly) organized around this model

What you'll learn (events, traces)



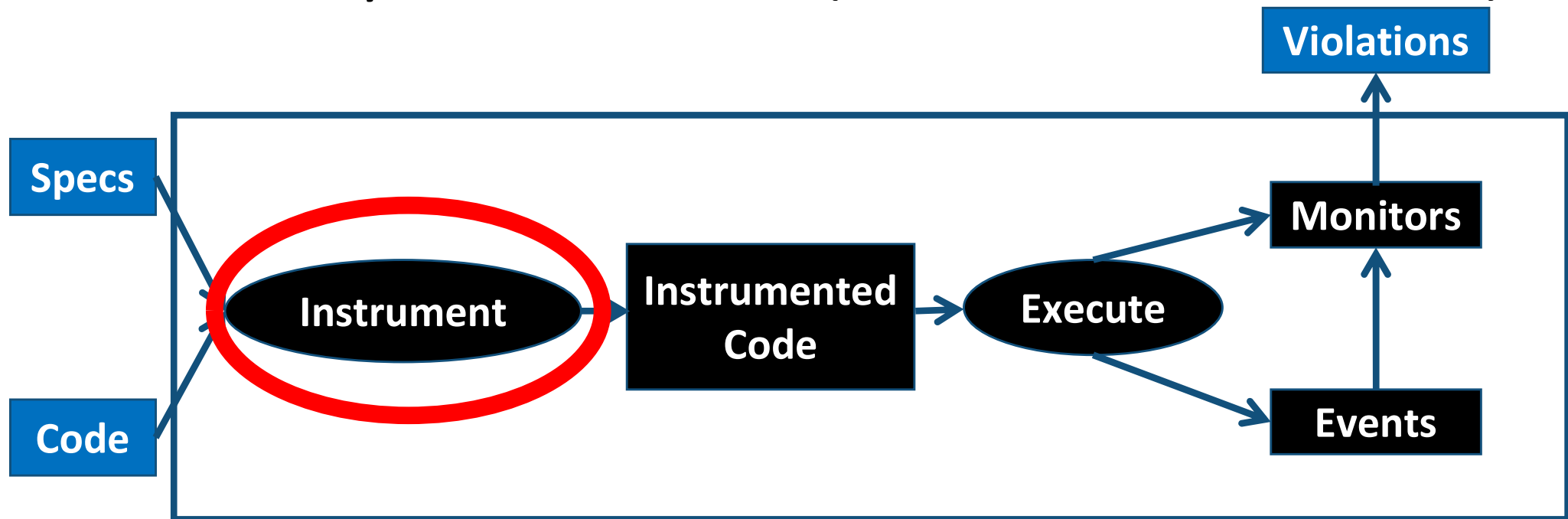
- A formal view of events, traces, and properties
- Program events (e.g., method calls, field access, etc)
- Event dispatch (e.g., which monitors to send events to?)

What you'll learn (specifications)



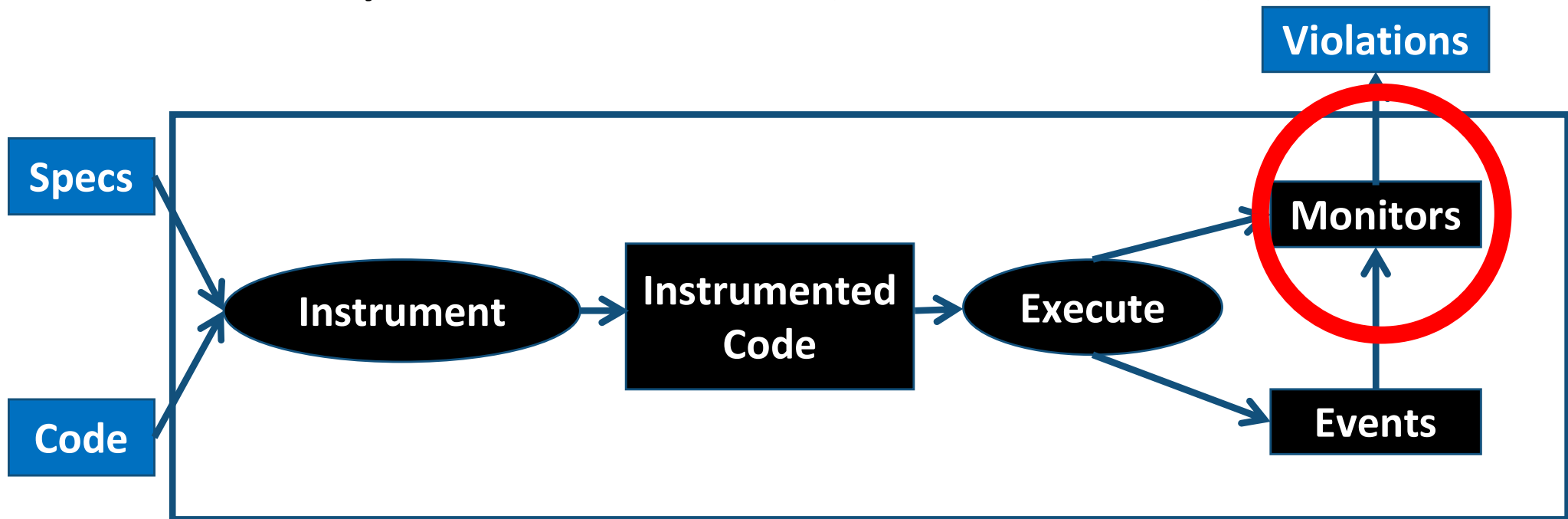
- What kinds of properties can RV check?
- What are languages for specifying properties in RV?
 - LTL, ERE, CFG, and other logical formalisms
- Where do properties come from? (You may write some)

What you'll learn (instrumentation)



- How to instrument code to obtain runtime events?
- Compile-time vs. runtime instrumentation
- Problems and challenges of instrumentation

What you'll learn (monitors)



- Monitor synthesis (translating specs to monitors)
- Monitoring algorithms (how monitors get and check events)
- Monitor indexing and garbage collection
 - Small-sized programs often generate tens of millions of monitors

What you'll learn (other topics)

- How to reduce RV overhead?
 - Combine with static analysis
 - Hardware-assisted RV
 - Sampling the events to check
- How to increase RV coverage?
 - Use RV during software testing
 - Incremental RV
- RV in other domains (depending on your interests)
 - hardware monitoring, networking, etc.

Is that all there is to RV?

rv22.gitlab.io

The topics of the conference include, but are not limited to:

- specification languages for monitoring
- monitor construction techniques
- program instrumentation
- logging, recording, and replay
- combination of static and dynamic analysis
- specification mining and machine learning over runtime traces
- monitoring techniques for concurrent and distributed systems
- runtime checking of privacy and security policies
- metrics and statistical information gathering
- program/system execution visualization
- fault localization, containment, resilience, recovery and repair
- systems with learning-enabled components
- dynamic type checking and assurance cases
- runtime verification for autonomy and runtime assurance

CFP
RV
2022

Application areas of runtime verification include cyber-physical systems, autonomous systems, safety/mission critical systems, enterprise and systems software, cloud systems, reactive control systems, health management and diagnosis systems, and system security and privacy.

Questions about course content?

?

Logistics



CS6156 information

- Owolabi Legunsen
 - Web: <https://www.cs.cornell.edu/~legunsen>
 - Email: legunsen@cornell.edu
 - Office Hours: Wed/Fri 4:00-5:00pm
- Course web page (with in-progress schedule)
 - <https://www.cs.cornell.edu/courses/cs6156/2023sp>
 - Go over the web page this week
 - Announcements will be sent on Canvas

CS 6156: Advanced SE/PL/Systems PhD-level course

| | Research Styles | | |
|----|-----------------|---------------------------------|---------|
| | Theoretical | Systems | Applied |
| PL | 611x, 6180 | 5114, 5120, 6120, 6114, 6156 | 6172 |



You are expected to...

- Read assigned texts before each class
- Complete 2-3 homework assignments
- Conduct a research project
- Lead 1 paper discussion and present your project

Your grade will be based on...

| | |
|---|------------|
| Readings | 10% |
| Homework (programming) | 10% |
| In-class participation | 5% |
| Presentation and discussion lead | 15% |
| Course project | 60% |

My goal is to give everyone an A

But you must do your part

Readings

- Readings will provide deeper understanding of RV
 - **You **will** feel lost in CS 6156 if you don't read**
- Ask exactly 2-3 **non-trivial** questions on a shared PDF
 - can't ask a question that someone already asked
 - questions should show that you have thought deeply about the text
 - bring other questions to class
- Due 11:59pm AOE the day before class

Presentation and discussion lead


- Each student will lead in-class discussion of a paper
 - Work with Owolabi ahead of time to prepare
 - Know the paper well, answer classmates' questions
 - Summarize the paper in class (~30mins)
 - Discuss questions that others asked

Homework

- 2 – 3 programming assignments
- Two goals
 - Assess your understanding of reading and lectures
 - Practice different aspects of RV

Course project goals

- Develop and present an idea
- Do a literature survey
- Work out your idea to a degree
- Evaluate the idea to some degree
- Write a 6-10 page paper on



Echoes
of
6410?

Course projects logistics

- Work individually or in self-selected groups
 - Working in groups is **strongly encouraged**, ideally with folks at same “level”: PhD/MS, MEng, BS
- BYOP: Could be a research project that you’re working on already, but should be in state of infancy
- A set of RV-related projects will also be suggested
 - But, executing the project is your job. So, choose wisely.

Tentative project timeline

| Milestone | Due date |
|-----------------------------|----------|
| Project proposal | 2/14 |
| Literature review | 3/7 |
| Intermediate project report | 4/4 |
| Final project report | 5/9 |

Before next class (pre-homework)

- Read the course webpage
 - <https://www.cs.cornell.edu/courses/cs6156/2023sp>
 - Read the assigned “How to read/write SE paper” articles
- If you are not a PhD student, send me an email answering these questions:
 - Your background (courses, internship, other experience)
 - What are you looking to get from CS 6156?
 - What project option are you currently interested in?

Questions about logistics?

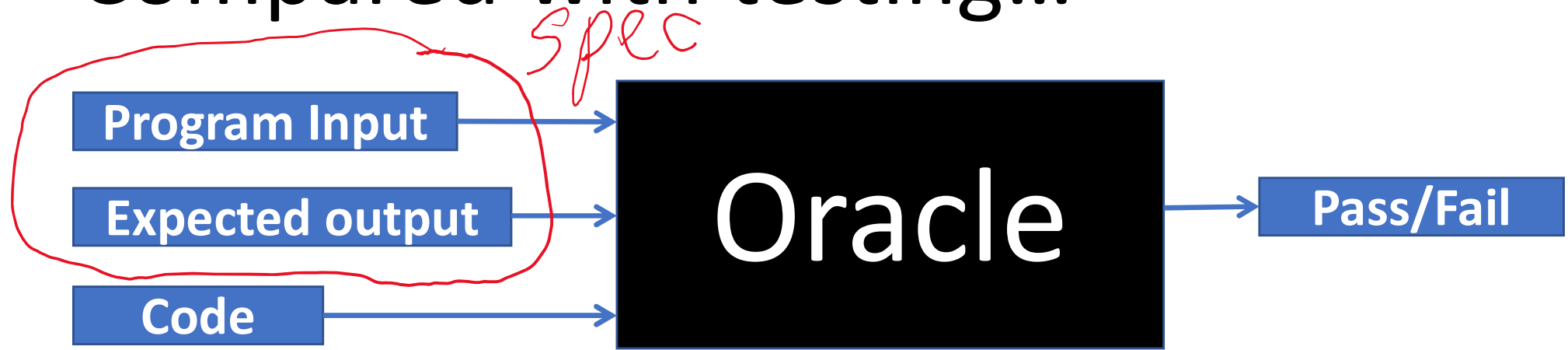
?

form of

Discuss: Why is RV a “verification”?



Compared with testing...



Is there any QA approach that can't be shown as above?

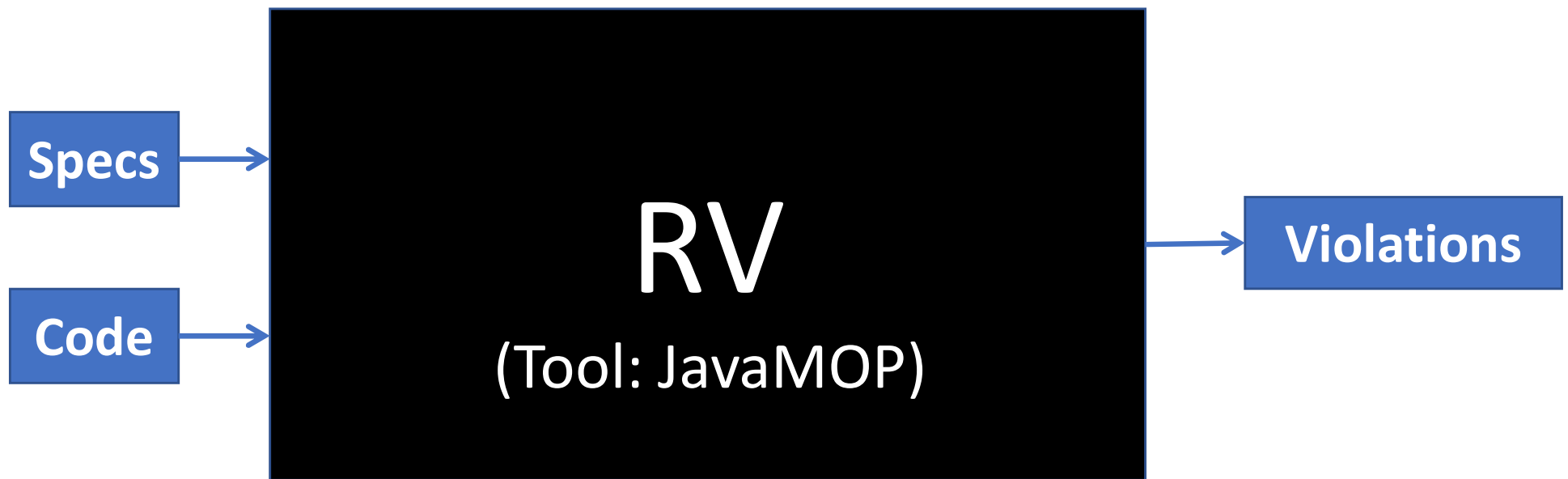
RV as “verification”? (vision)

1. RV can be done as a system runs in production
2. RV can allow the system to recover just before violations occur
 - Seems relatively under-explored in practice
3. So, RV can be used to ensure that a system never goes wrong with respect to a specification
 - In theory, RV can force the system to always be correct

Another reason RV is appealing

Recall: high-level view of RV

Now: concrete examples of RV tool, inputs, and outputs



- One RV tool that we will use in this class is JavaMOP
 - <https://github.com/runtimeverification/javamop>

Example spec: Collection_SynchronizedCollection (CSC)

[https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#synchronizedCollection\(java.util.Collection\)](https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#synchronizedCollection(java.util.Collection))

synchronizedCollection

```
public static <T> Collection<T> synchronizedCollection(Collection<T> c)
```

It is imperative that the user manually synchronize on the returned collection when iterating over it:

```
Collection c = Collections.synchronizedCollection(myCollection);  
...  
synchronized (c) {  
    Iterator i = c.iterator(); // Must be in the synchronized block  
    while (i.hasNext())  
        foo(i.next());  
}
```

Failure to follow this advice may result in non-deterministic behavior.

Live demo: RV of CSC on toy code

<https://javamop.coecis.cornell.edu/run>

Run JAVAMOP Online ERE/SafeSyncCollection/SafeSyncCollection.mop

```
1: // Copyright (c) 2002-2014 JavaMOP Team. All Rights Reserved.
2: package mop;
3:
4: import java.io.*;
5: import java.util.*;
6:
7: // The SafeSyncCollection property is designed
8: // to match a case where either a collection
9: // is synchronized and an non-synchronized
10: // iterator is created for the collection, or
11: // a synchronized iterator is created, but
12: // accessed in an unsynchronized manner.
13:
14: SafeSyncCollection(Object c, Iterator iter) {
17:     creation event sync after() returning(Object c) :
```

File Explorer: javamop > CFG > ERE > SafeSyncCollection > SafeSyncCollection.mop

Buttons: Run, Monitor, Help

Terminal: >>> javamop -agent ERE/SafeSyncCollection/SafeSyncCollection.mop

Output: SafeSyncCollectionMonitorAspect.aj is generated
-Processing 1 specification(s)

1. Click on spec (points to SafeSyncCollection.mop)

2. Click on code (points to the code editor)

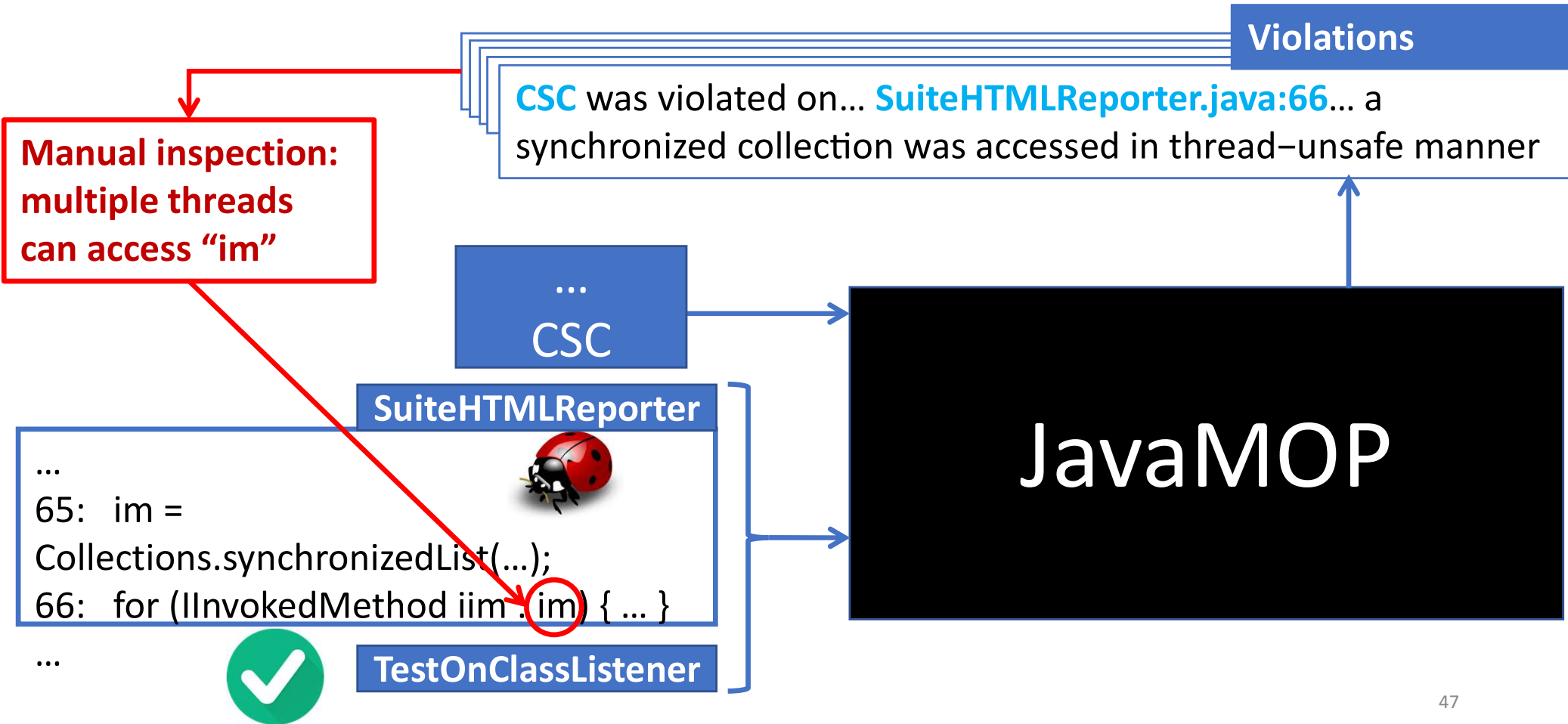
3. Run w/o RV (points to the Run button)

4. Run with RV (points to the Monitor button)

What we saw during the demo

- A spec written as an ERE
- JavaMOP output
- JavaMOP finds a violation in code that runs “correctly”
 - is the violation a bug, though?
- An online environment for using JavaMOP

The “RV process” (also used in demo)



RV in my SE (RV + testing) research

- Monitored the tests in 229 open-source software
 - some of them have over 200K lines of code
- RV found hundreds of bugs that testing missed
 - many have been confirmed
- But there are still many challenges
 - You'll discover some of them in this class

Next class...

- Start with the basics: events, traces, properties
- Reading is assigned (overview of RV)
 - Due by 11:59pm AOE Thursday 2/3/2022

What we learned so far

- A comparison of RV with other QA approaches
- A whirlwind tour of RV
- Learning outcomes, course content, and logistics
- Demo of an RV tool (JavaMOP)