



Software Testing and Analysis

Gregory M. Kapfhammer
Department of Computer Science
Allegheny College

Computer Science 580, Spring 2006

Invocation

Thus spake the master, “*Any program, no matter how small, contains bugs.*”

The novice did not believe the master’s words.
“*What if the program were so small that it performed a single function?*”

The master thoughtfully responded, “*Such a program would have no meaning. But, if such a one existed, the operating system would fail eventually, producing a bug.*”

Invocation

But the novice was not satisfied. “*What if the operating system did not fail?*”

The master responded, “*There is no operating system that does not fail. But if such a one existed, the hardware would fail eventually, producing a bug.*”

The novice still was not satisfied. “*What if the hardware did not fail?*”

Invocation

The master gave a great sigh. *“There is no hardware that does not fail. But if such a one existed, the user would want the program to do something different, and this too is a bug.”*

A program without bugs would be an absurdity, a nonesuch. If there were a program without any bugs then the world would cease to exist.

Geoffrey James
The Zen of Programming
(Adaptation)

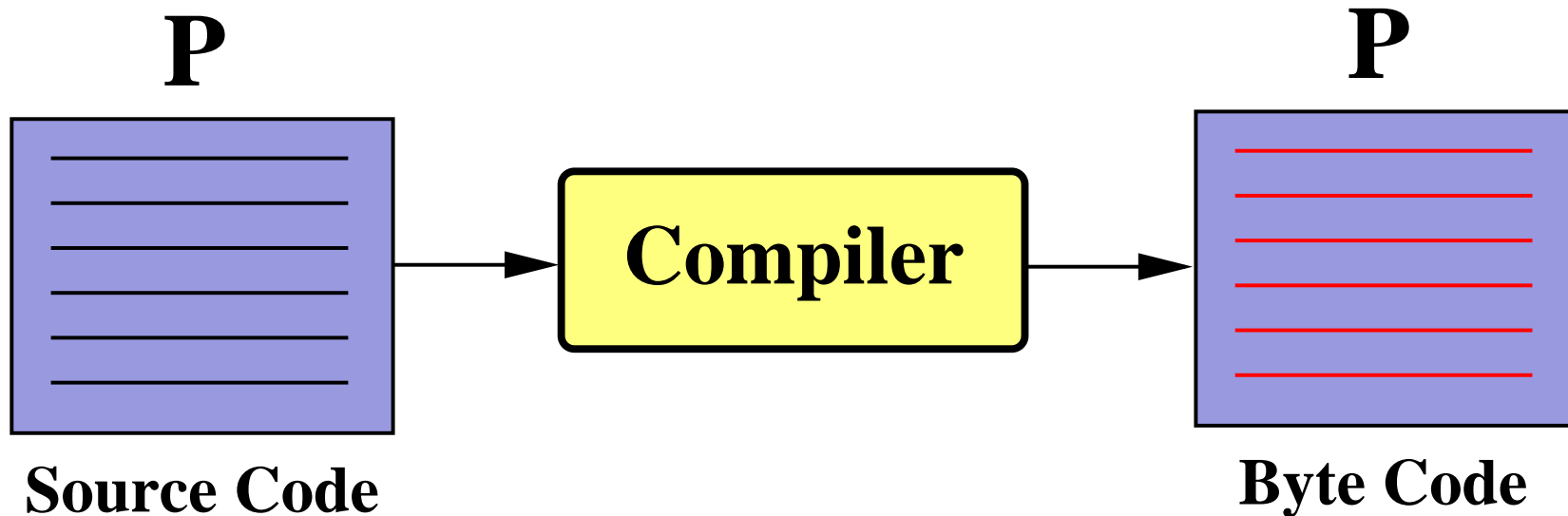
Motivation

The Risks Digest, Volume 22, Issue 64, 2003

Jeppesen reports airspace boundary problems

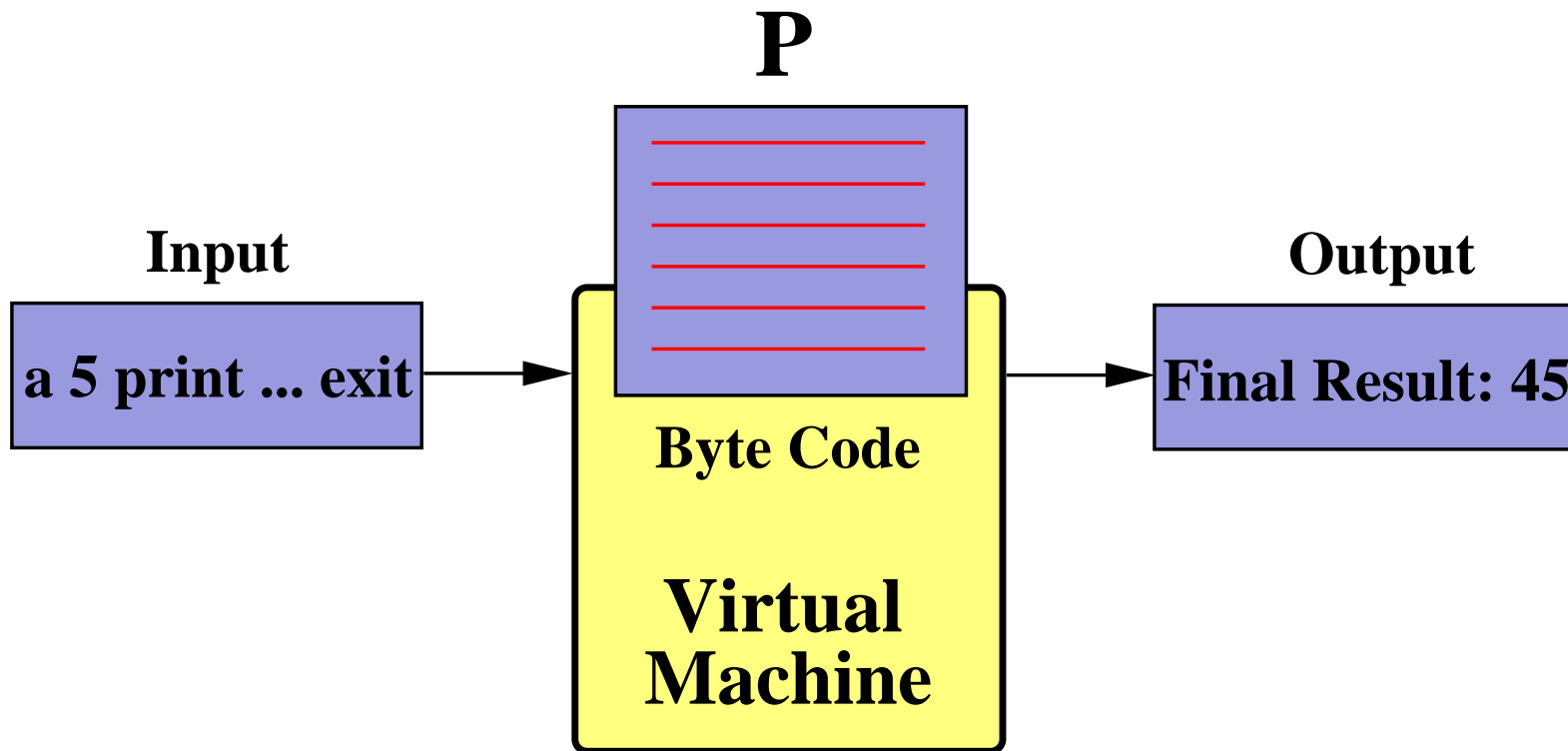
About 350 airspace boundaries contained in Jeppesen NavData are incorrect, the FAA has warned. The error occurred at Jeppesen after a software upgrade when information was pulled from a database containing 20,000 airspace boundaries worldwide for the March NavData update, which takes effect March 20.

Life of a Program: Compilation



- The programming language compiler produces a representation of a program that can be executed

Life of a Program: Execution



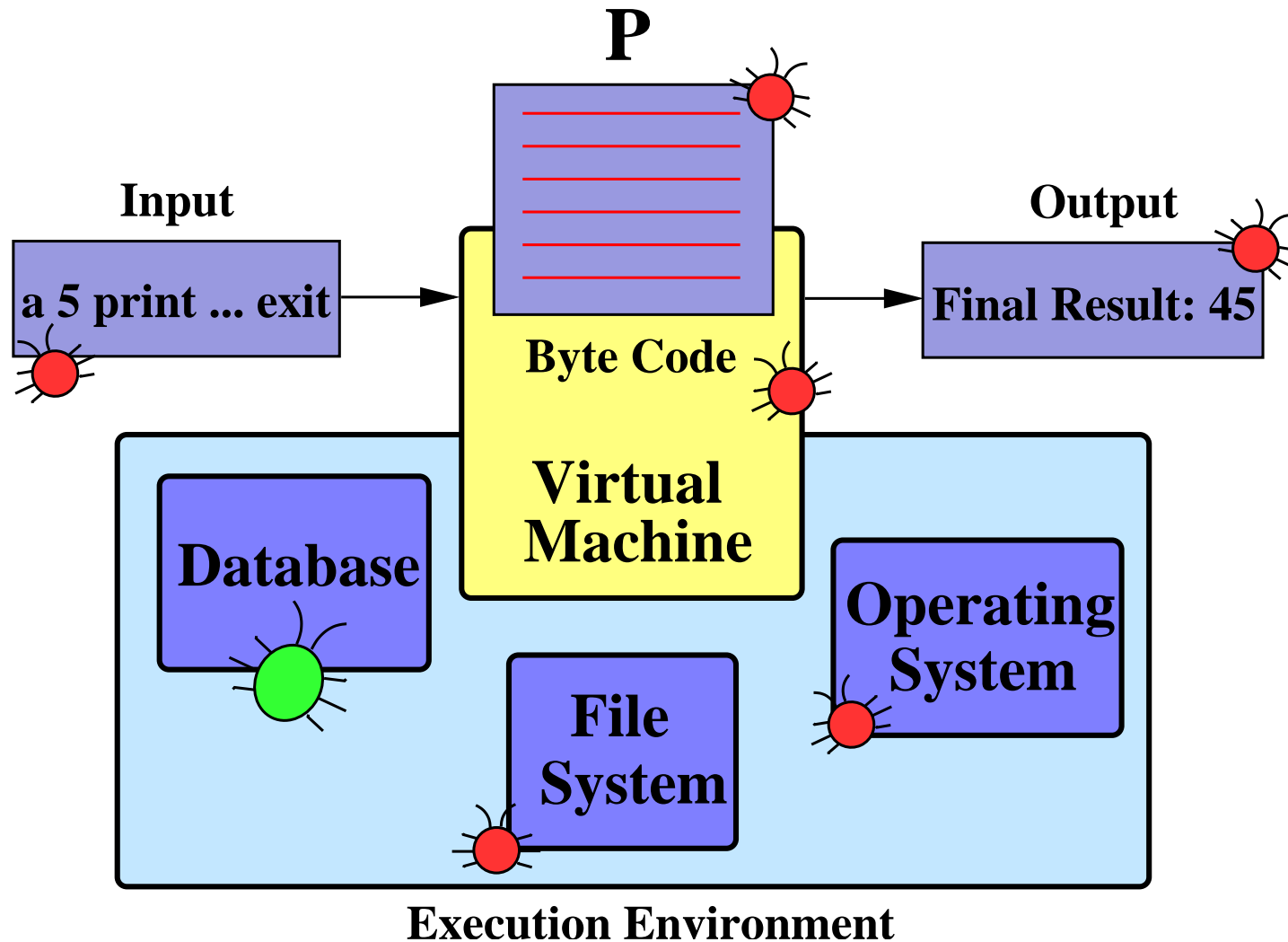
→ The virtual machine executes the program's byte codes

Testing Challenges

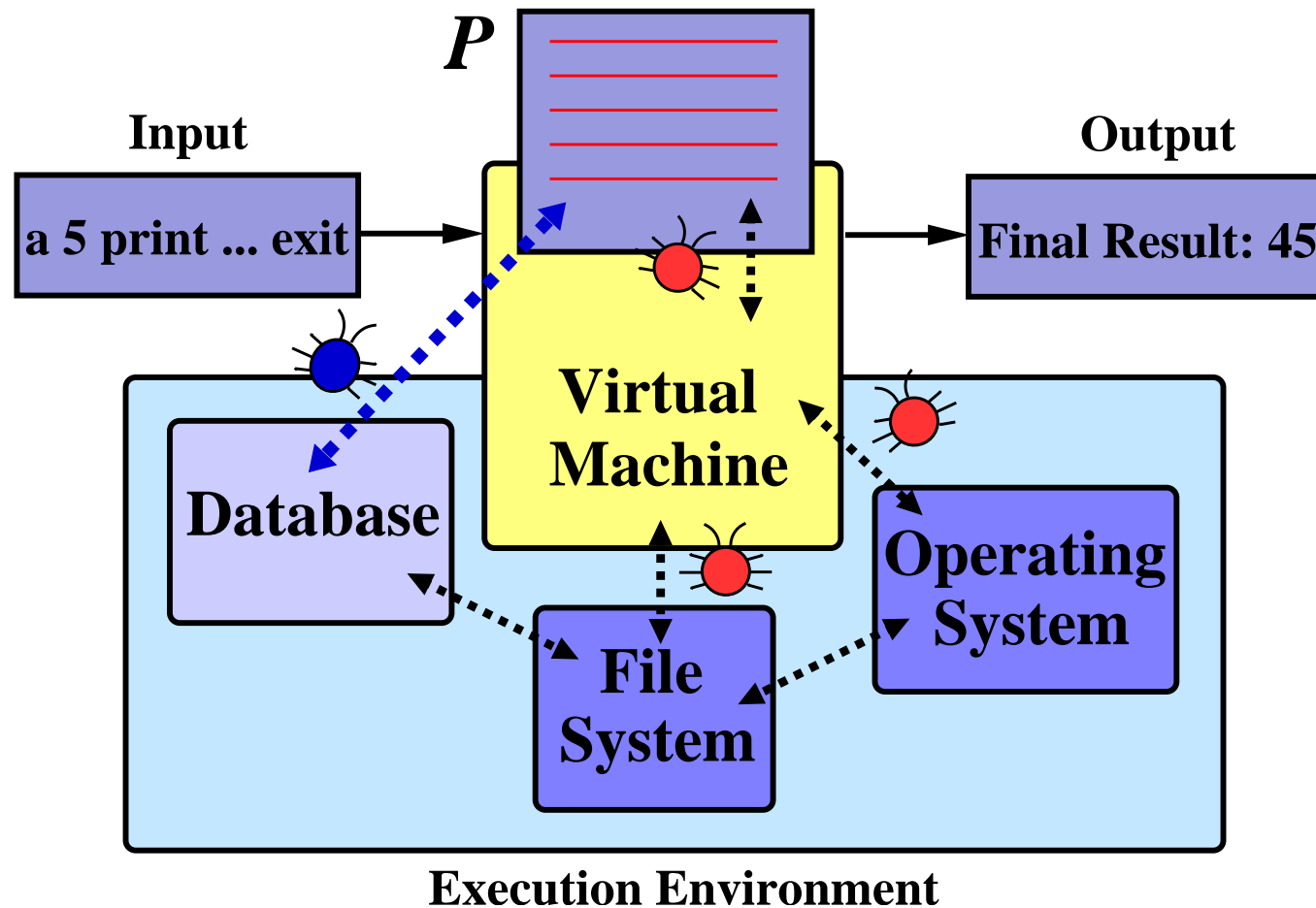
I shall not deny that the construction of these testing programs has been a major intellectual effort: to convince oneself that one has not overlooked “a relevant state” and to convince oneself that the testing programs generate them all is no simple matter. The encouraging thing is that (as far as we know!) it could be done.

Edsger W. Dijkstra, 1968

Defect Locations

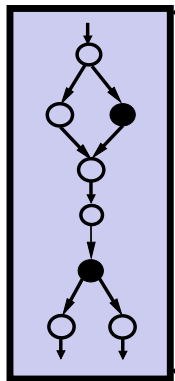


Defective Environment Interactions



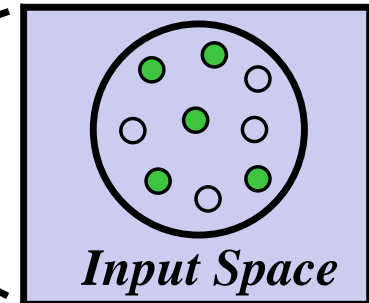
- Defects can exist in P 's interaction with its environment

Traditional Testing Techniques



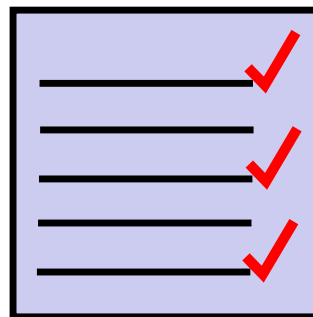
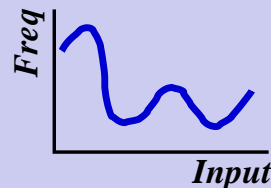
Structural Testing

Random Testing



Software Reliability

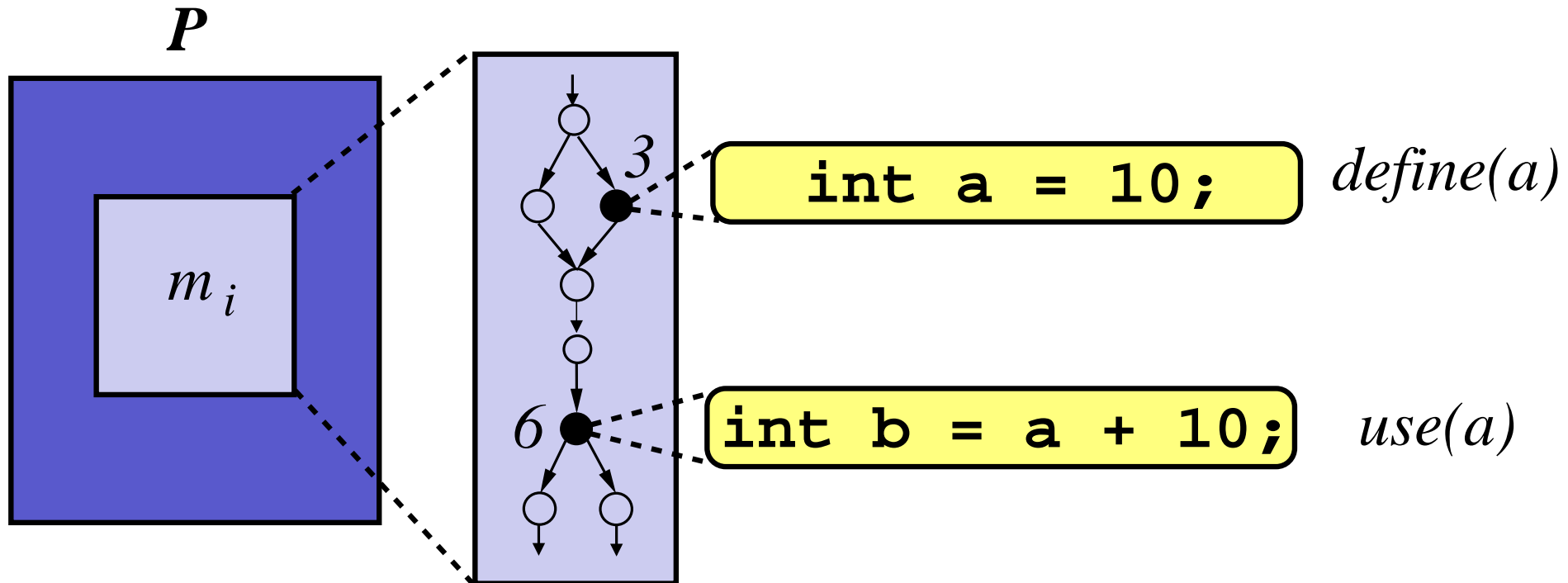
$$MTTF(P) = k$$



Specification Testing

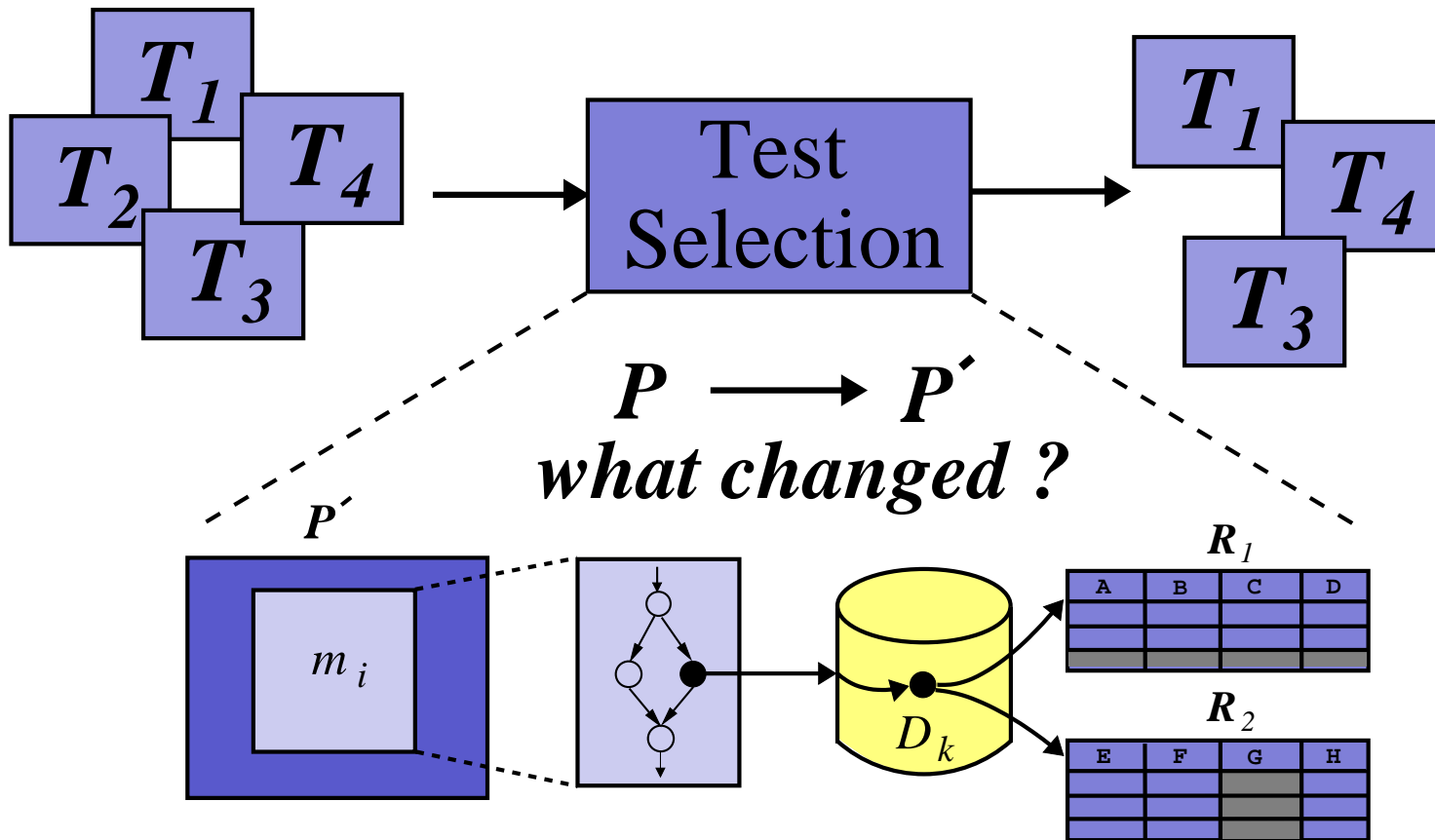
- Different approaches to establishing a confidence in the correctness of software.

Traditional Data Flow Testing



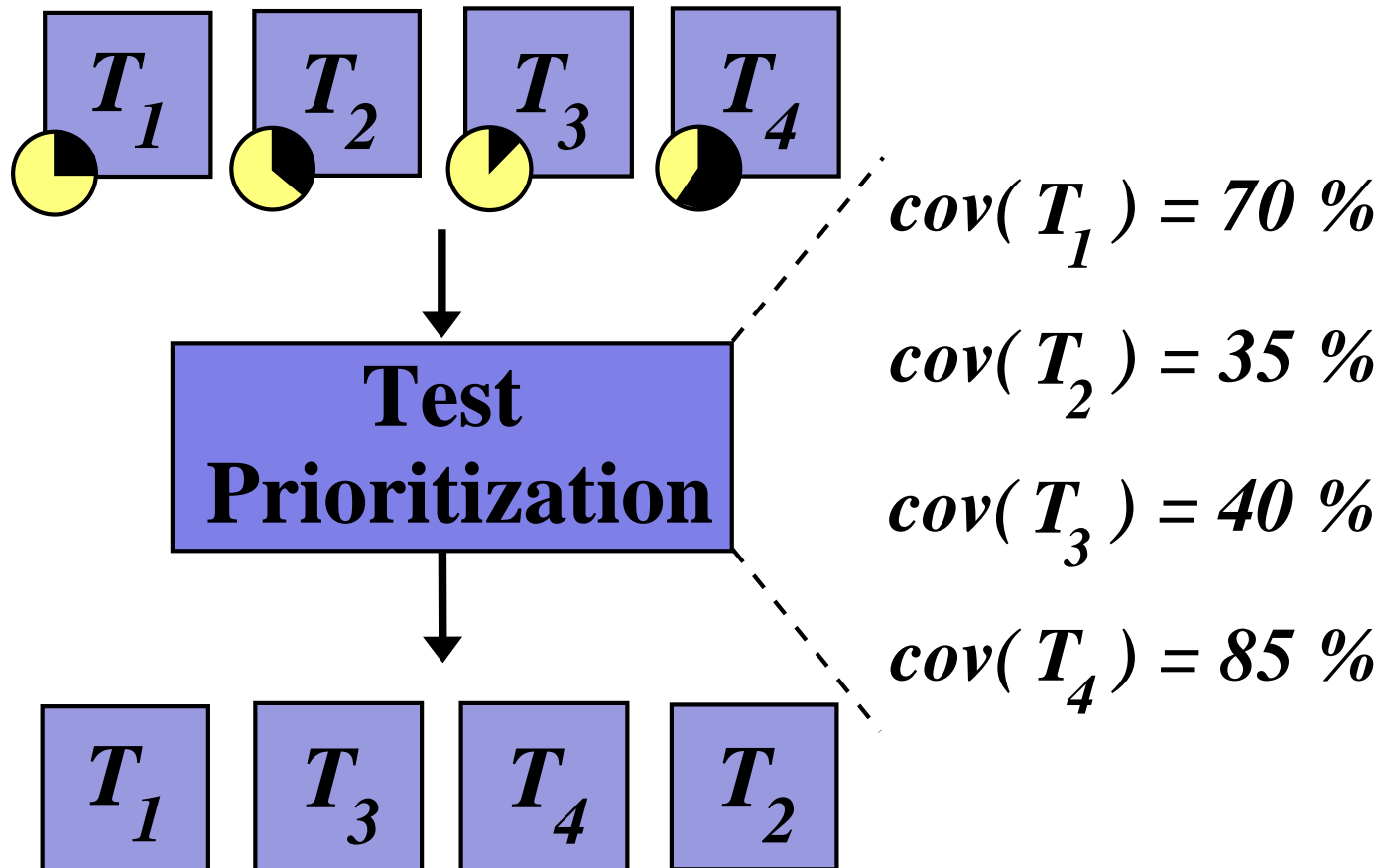
- The intraprocedural def-use association $\langle n_3, n_6, a \rangle$ exists within P 's method m_i

Regression Test Selection



- Use a model of all potential changes to P and D_1, \dots, D_n in order to select a subset of tests that must be executed

Regression Test Prioritization



- Re-order the test cases based upon previously calculated adequacy and/or time constraints

Example: computeVelocity

- Download Kinetic.java, KineticTest.java, and junit.jar
- Ensure that junit.jar and :. are in your CLASSPATH environment variable
- K is defined such that $K = \frac{1}{2}mv^2$
- Where is the defect in computeVelocity?
- Do any of the tests reveal the defect? How?

Conclusions

- Software testing can isolate defects and establish a confidence in the correctness of a program under test
- Software testing is hard, especially when interaction with the application's environment is considered
- There are different types of software testing techniques such as structural testing, random testing, and specification testing
- Automated test data generation and regression testing techniques can also be used