Automatically Finding Performance Problems with Feedback-Directed Learning Software Testing

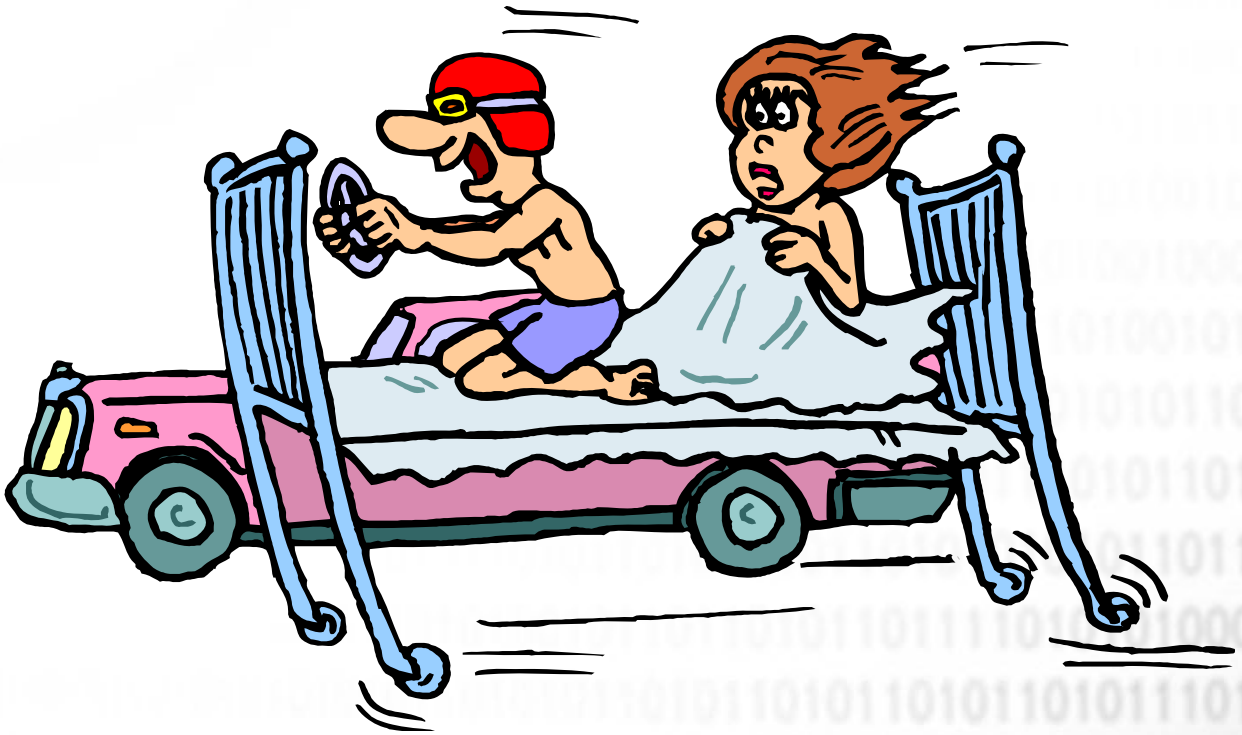
Mark Grechanik

University of Illinois at Chicago

Chen Fu and Qing Xie

Accenture Technology Lab

Good Performance Is Important



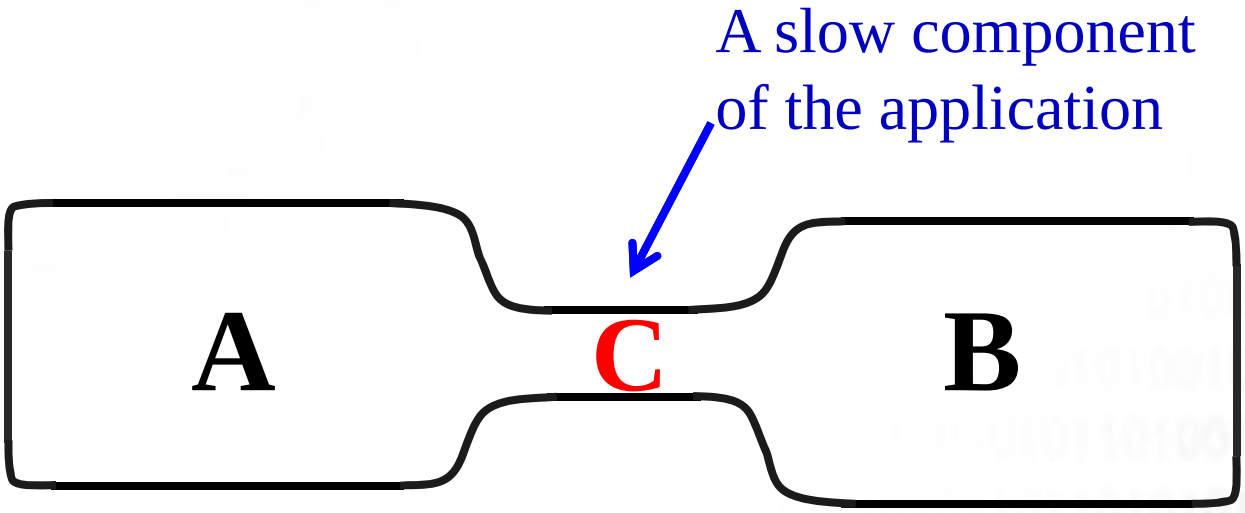
Excellent Performance Is Even More Important!

Finding Performance Problems

A goal is to find situations when applications unexpectedly exhibit worsened characteristics for certain combinations of input values.

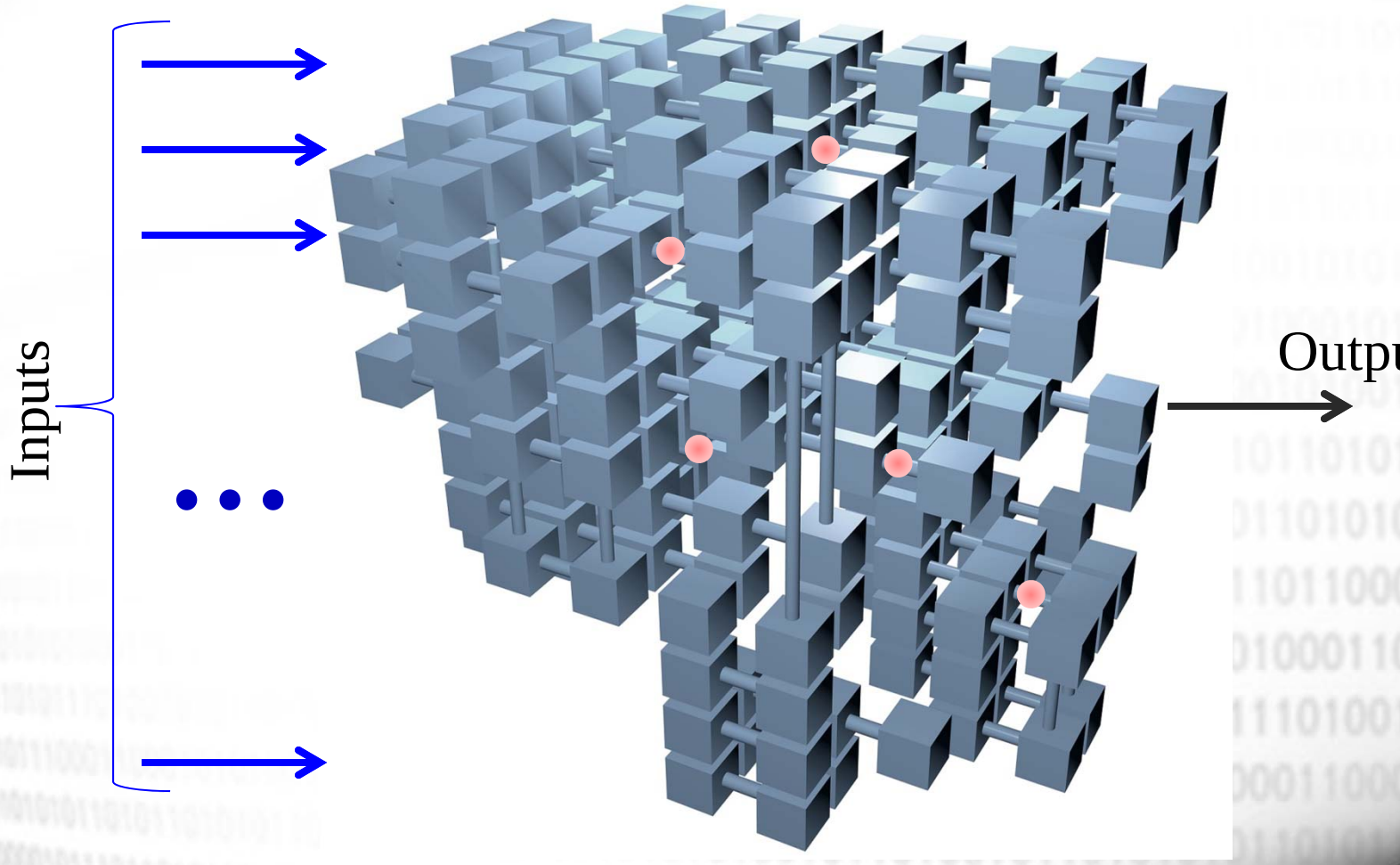
Finding This Situation Is Like Getting a Perfect Hand!

Example: Bottlenecks

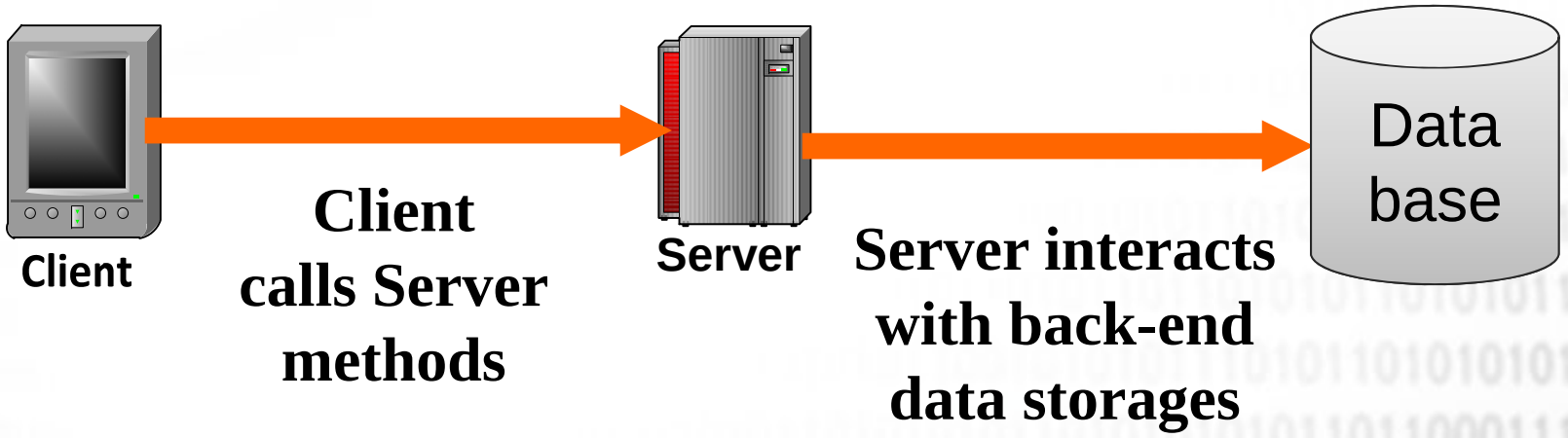


Bottlenecks or hot spots are phenomena where the performance of the application is limited by one or few components

Bottlenecks in Nontrivial Applications

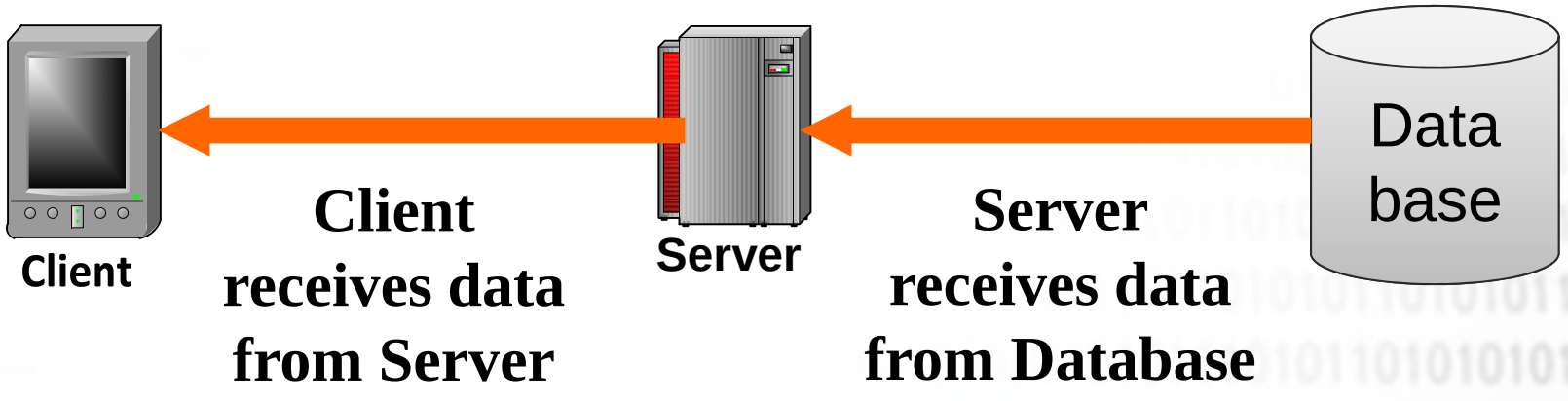


Performance Testing Applications



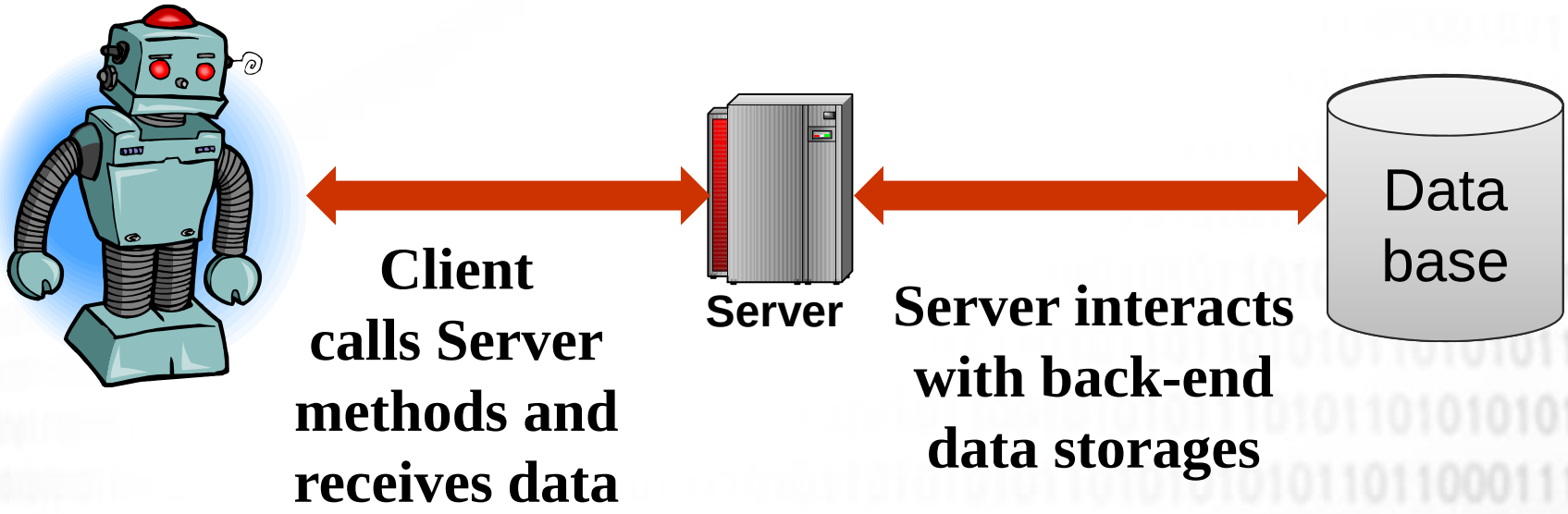
The Client executes methods of the Server using different combination of input parameter values

Performance Testing Applications



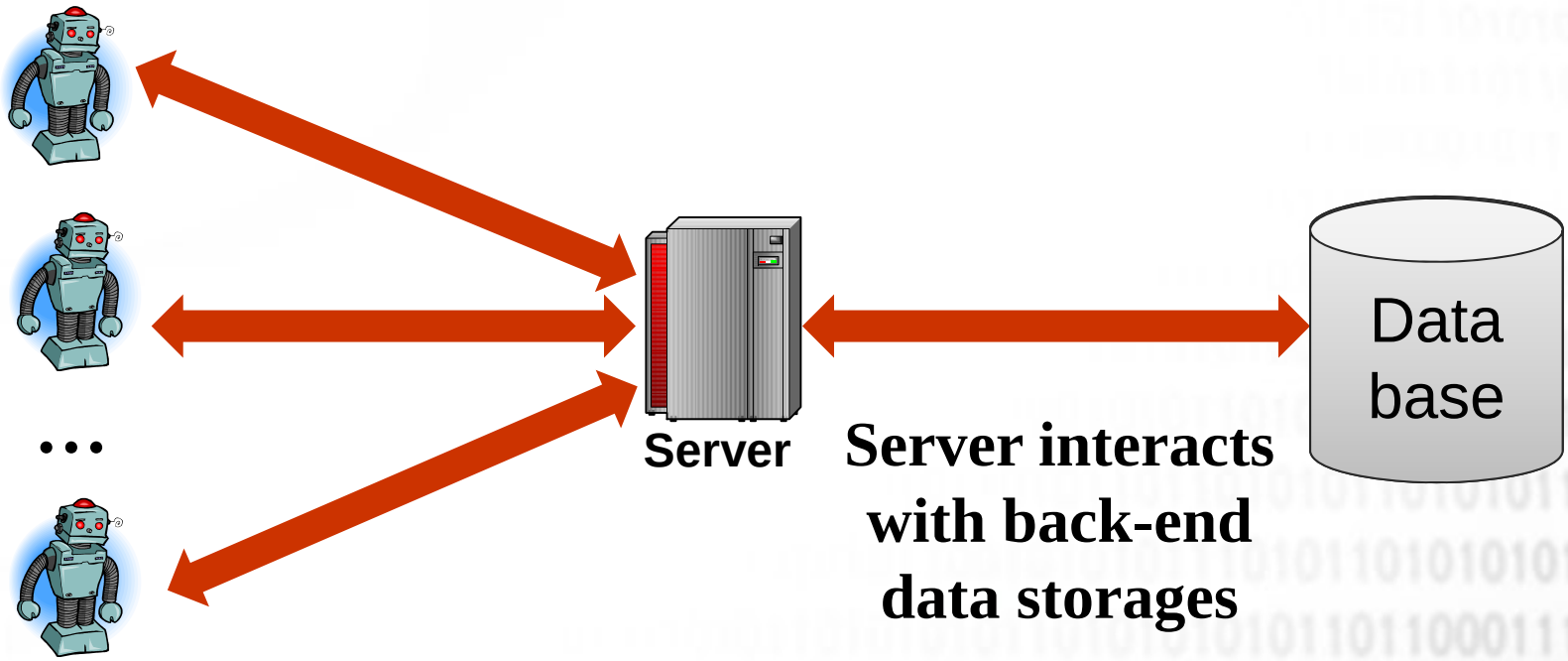
Performance can be measured in the number of roundtrips per time unit

Automating Performance Testing Applications



Automated test scripts simulate clients who perform transactions against the Server, and they use input data from the backend database

Automating Performance Testing Applications



Automated test scripts simulate clients who perform transactions against the Server, and they use input data from the backend database

A Problem With Performance Testing of Large Applications

A medium-size renters insurance application has a large universe of input data.

- For 78,000,000 customer profiles it would take close to 1,500 years to test this application on all profile inputs provided that it takes only 10mins per input.
- Just “touching” one customer profile entry for 1sec, it takes about 2.5 years to “touch” them all!
- Some customer profiles may lead to significant load on different resources.

A Problem With Performance Testing of Large Applications

How to select a manageable subset of the input data without compromising the effectiveness of performance testing.

- **We need an insight into selecting this subset!**

An Example

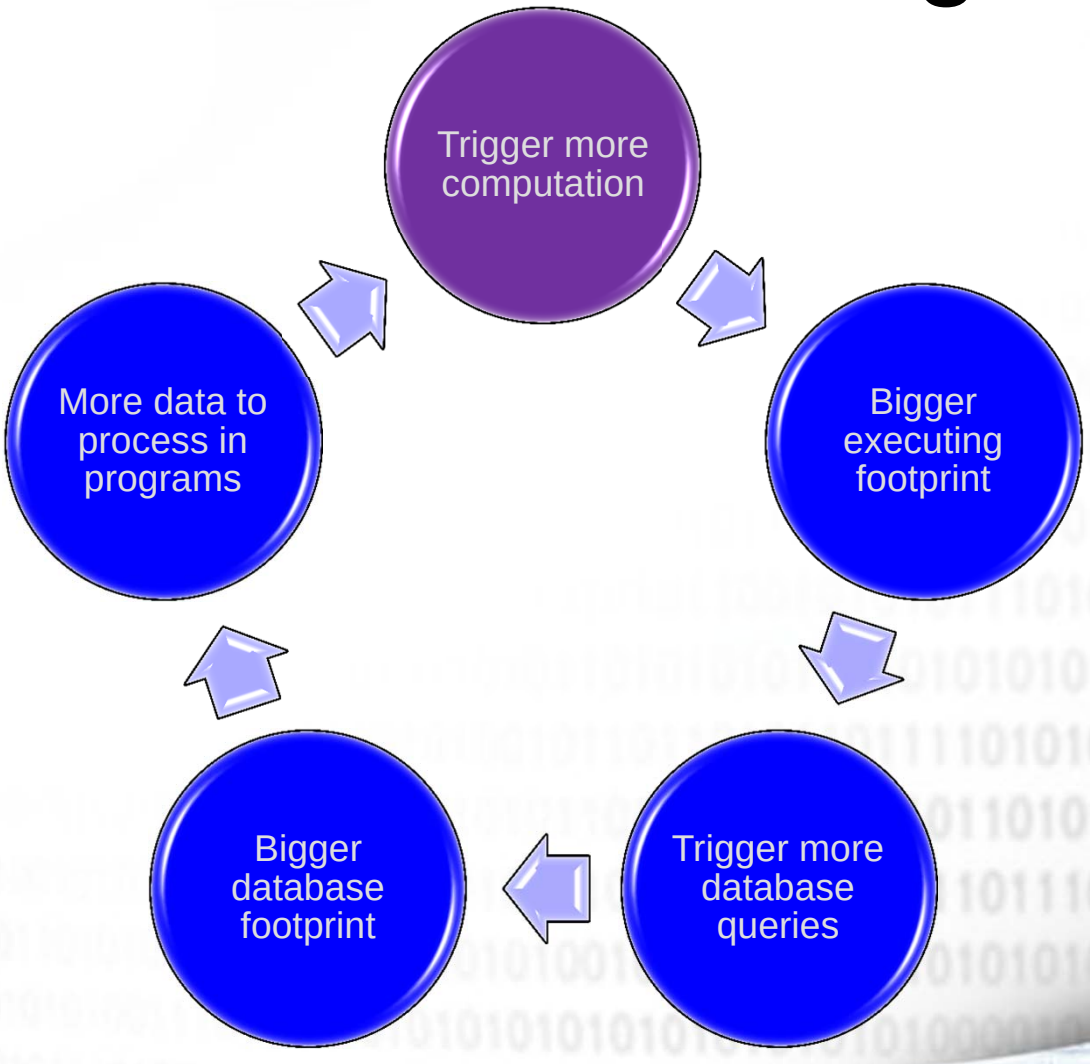
Most of the renters are good customers, have hardly any accident/claims or fraudulent activities.

- That's how insurance companies make money.
- Unfortunately, the data of these "good" customers are not always good for performance testing purpose.

Randomly selecting customer profiles does not work well – keep encountering good customers.

- Yet, *exploratory performance testing* is one of the main ways of performance testing applications in industry!

The Intuition Behind Performance Testing



Core Idea

Select only these customer profiles that trigger intensive computations. These computations most likely involve significant interactions with databases.

The State of the Practice

Intuitive testing is a method for testers to exercise the product based on their intuition and experience, surmising probable errors.

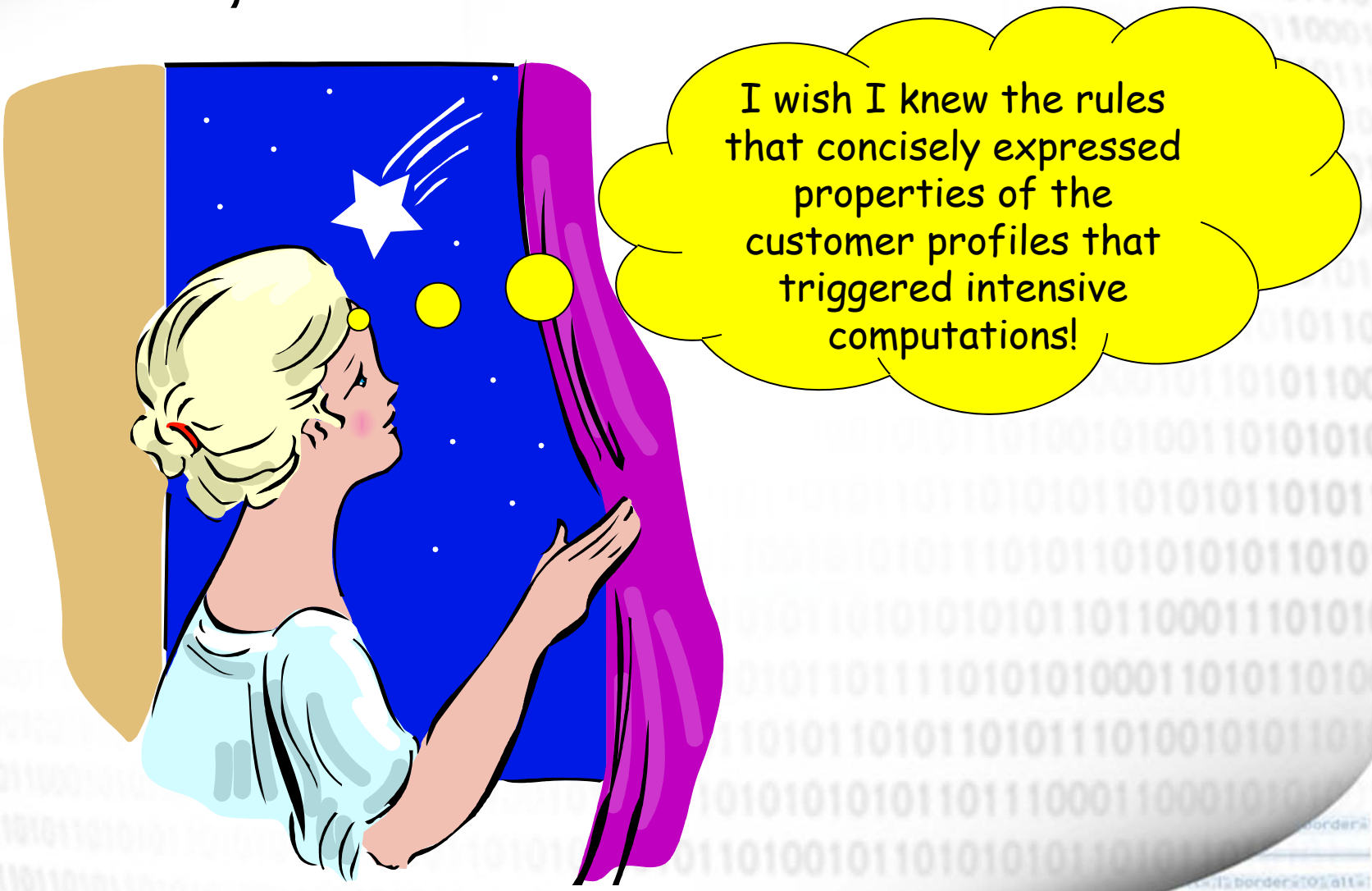
- Intuitive testing dates back to 1979.
- Use experience of test engineers to focus on error-prone and relevant system functions without writing time-consuming performance test specifications.

Intuitive testing cannot cope with demands of performance testing.

- Frequently, guesses by test engineers and system administrators are wrong.
- Typically, performance degradations of up to 20% go unnoticed.

Performance Testing Is Laborious and Difficult

But, what if…


I wish I knew the rules that concisely expressed properties of the customer profiles that triggered intensive computations!

We Need Rules!

We Need Rules!

Descriptive rules for selecting test input data play a significant role in software testing, where these rules approximate the functionality of the application.

- Example rule: some customers will pose a high insurance risk if these customers have one or more prior insurance fraud convictions and deadbolt locks are not installed on their premises.

$$(\texttt{customer.numberOfResidents} \leq 2) \wedge (\texttt{coverages.limitPerOccurrence} \geq 400000) \wedge (\texttt{preEligibility.numberOfWildAnimals} \leq 1) \Rightarrow \textbf{Bottleneck}$$

We Need Rules!

Descriptive rules for selecting test input data play a significant role in software testing, where these rules approximate the functionality of the application.

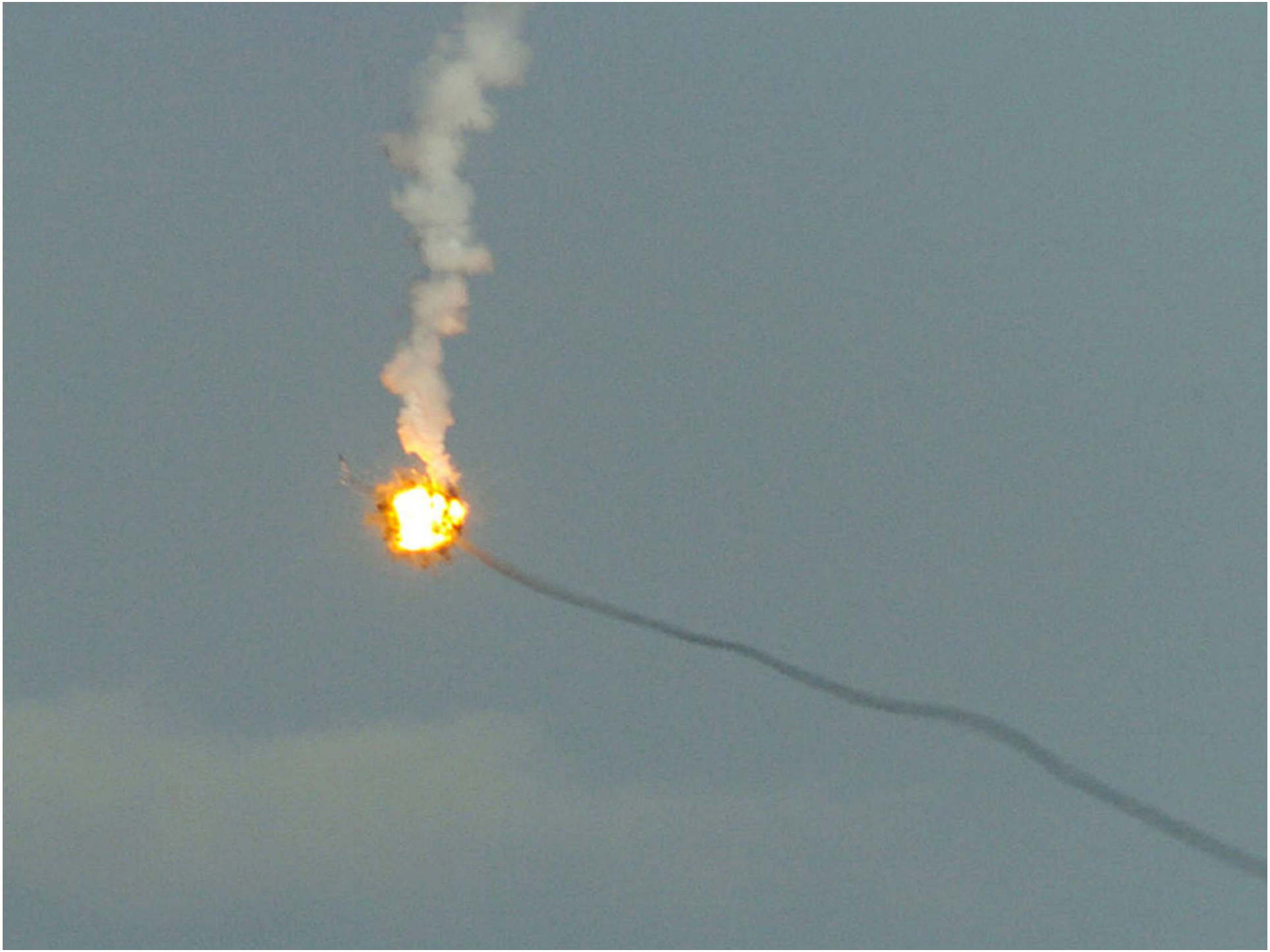
- Example rule: some customers will pose a high insurance risk if these customers have one or more prior insurance fraud convictions and deadbolt locks are not installed on their premises.

$$(\text{customer.numberOfResidents} \leq 2) \wedge (\text{coverages.limitPerOccurrence} \geq 400000) \wedge (\text{preEligibility.numberOfWildAnimals} \leq 1) \Rightarrow \textbf{Bottleneck}$$

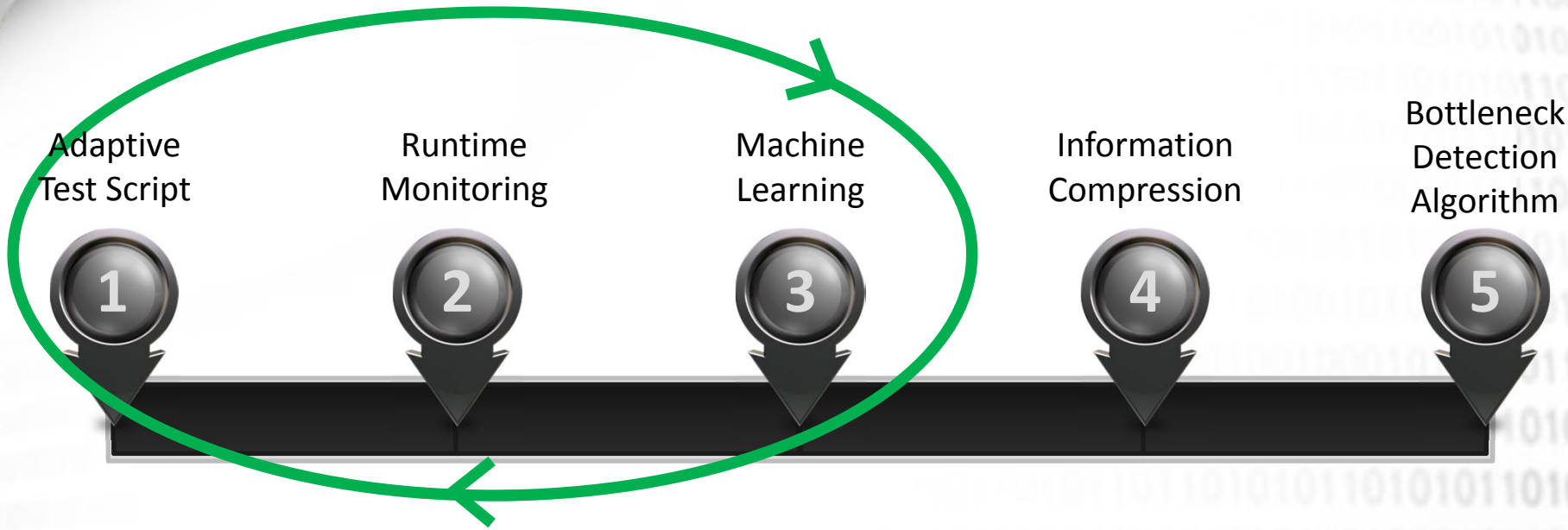
Our Solution - FOREPOST

Feedback-ORiEnted PerfOrmance Software Testing (FOREPOST) finds performance problems automatically by learning and using rules that describe classes of input data that lead to intensive computations.

- FOREPOST is an adaptive, feedback-directed learning testing system that learns rules from execution traces and uses these learned rules to select test input data automatically to find more performance problems in applications when compared to exploratory random performance testing.
- **FOREPOST is like a heat-seeking solution!**



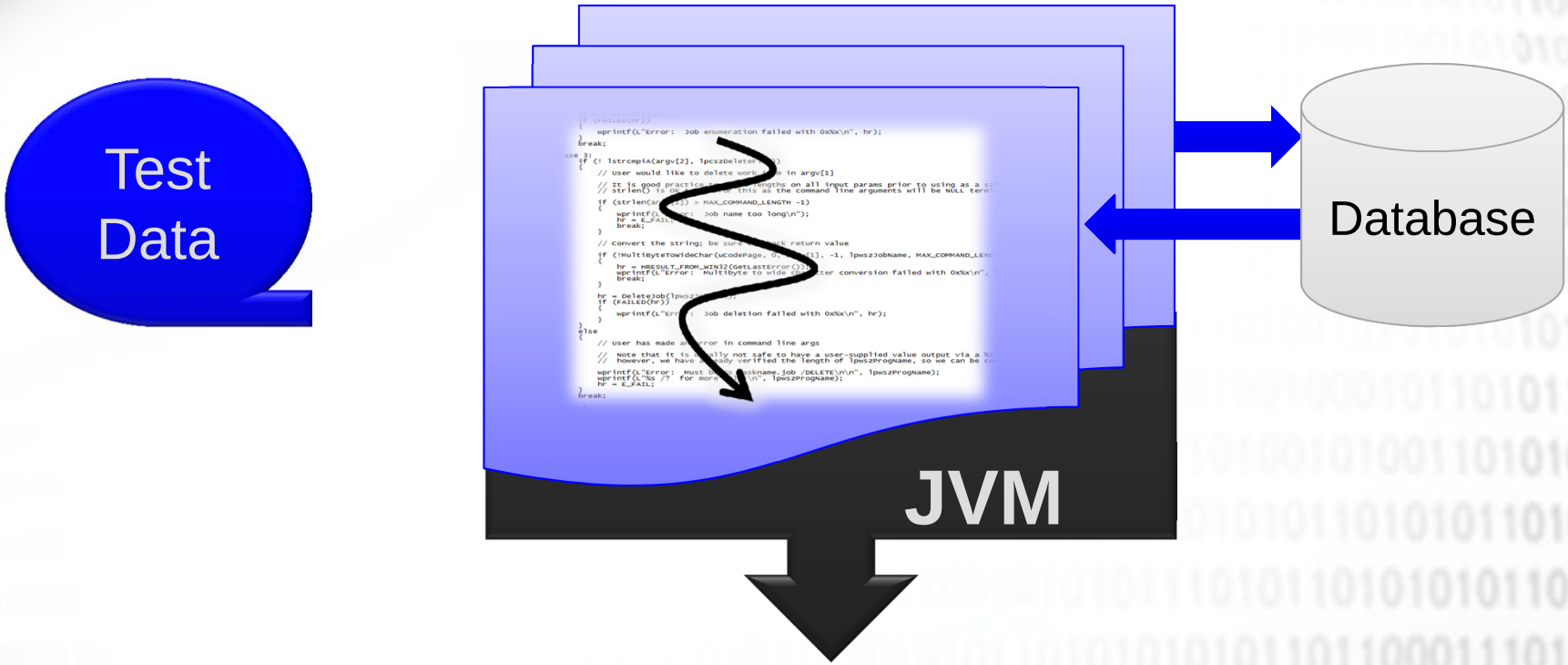
Components of FOREPOST



Adaptive test script uses rules that are issued by machine learning components to select input data that lead to more intensive computations.

When the test script executes the application, its execution traces are collected as part of runtime monitoring, leading to (re)learning rules.

Collect Execution Traces



Execution Trace Entries

Method entry and exit

- Every time a method is called, a corresponding entry is put into a file where execution traces are stored
- Every time a method call is finished, a corresponding entry is put into a profile file

Database Data

- When an application sends data to and receives data from the database, this data is captured and put into the execution trace file

Input parameters

- Capture choices of inputs that users provides

Execution Trace Sample

MTDENT_|_WebContainer : 0_|_21088404309_|_statefarm/framework/controllerw0089701/appcoordination/AspUrlFilter
_|_getRequest(Ljavax/servlet/ServletRequest;)Ljavax/servlet/http/HttpServletRequest;_||_
MTDENT_|_WebContainer : 0_|_21236207529_|_statefarm/framework/jcaimsframeworkw0099456/Msg_|_
addStringToByteArray(I[BLjava/lang/String;)V_||_
MTDRET_|_WebContainer : 0_|_21236207566_|_statefarm/framework/jcaimsframeworkw0099456/Msg_|_
addStringToByteArray(I[BLjava/lang/String;)V_||_
MTDRET_|_WebContainer : 0_|_21088404359_|_statefarm/framework/controllerw0089701/appcoordination/AspUrlFilter
_|_getRequest(Ljavax/servlet/ServletRequest;)Ljavax/servlet/http/HttpServletRequest;_||_

REQ_|_WebContainer : 0_|_21208103665_|_waterAvailableYearRound={N+}
CTNextPage={unprotectedDwelling+}visibleFromRoadOrNeighborUnselected={+}command={navigate+}
propertyCity={HASTINGS+}fireDeptWithin10Unselected={+}nextPageFocusElement={+}
propertyCountyUnselected={+}directionsToLocation={+}sortColumn={+}street2={+}street1={63 JACK PL+}
insideCityLimit={Y+}TF:RN={3+}fireDeptWithin10={N+}insideCityLimitUnselected={+}returnCommand={+}
requestId={V1KY805MV05+}waterAvailableYearRoundUnselected={+}TF:ID={ControllerFramework:0+}
nextPage={unprotectedDwelling+}currentPage={location+}selectedEntityId={+}CTCurrentPage={location+}
accessibleToFireDeptUnselected={+}clearErrorList={+}selectedContainerId={+}propertyZipCode={550331097+
}accessibleToFireDept={N+}propertyCounty={DAKOTA+}returnPage={current+}
visibleFromRoadOrNeighbor={+}returnPageFocusElement={unprotectedDwellingLink+}_||_

IMSRET_|_WebContainer : 0_|_21238137181_|_331617041_|_APUN7N_|_112FE009Z1
081V1KY805MV05082abcd083 AGENT_APPLICATION09505092GNRT_RATE_ZONES
041042APPLY_FOR_POLICY041042AGRE041042AGREEMENT041
042INSURED_LOCATION_|_010Z1081V1KY805MV050840_||_

Some Statistics

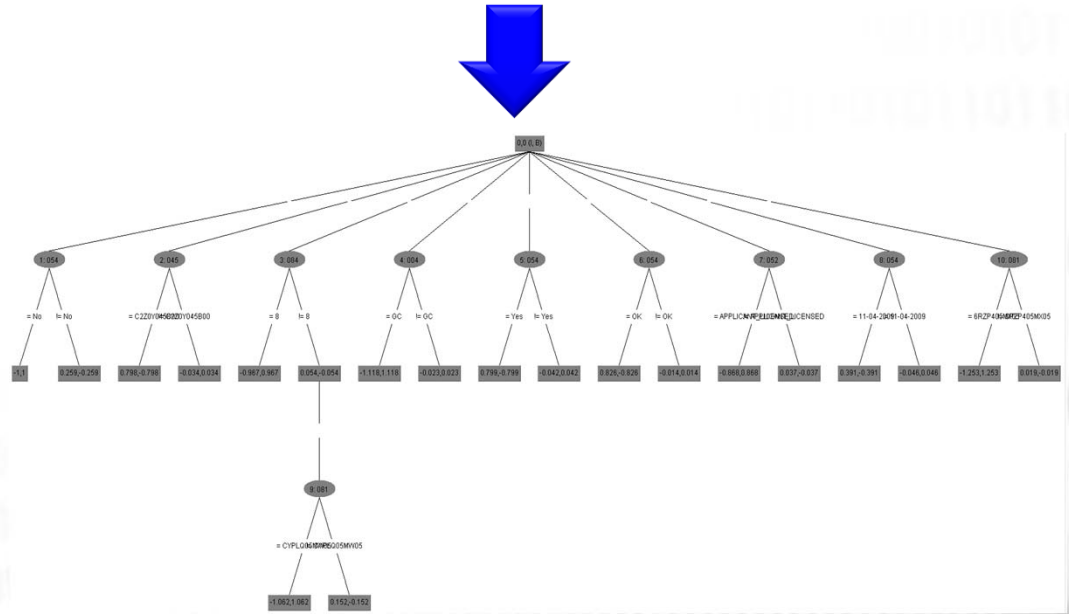
It takes about 10mins to execute the instrumented renters app end-2-end automatically using a test script.

An average run invokes over 3,000 methods more than 3,000,000 times and it retrieves over 1Mb of data.

An average execution traces contains over 1Gb of data. It takes less than 10mins on average to process and analyze one trace.

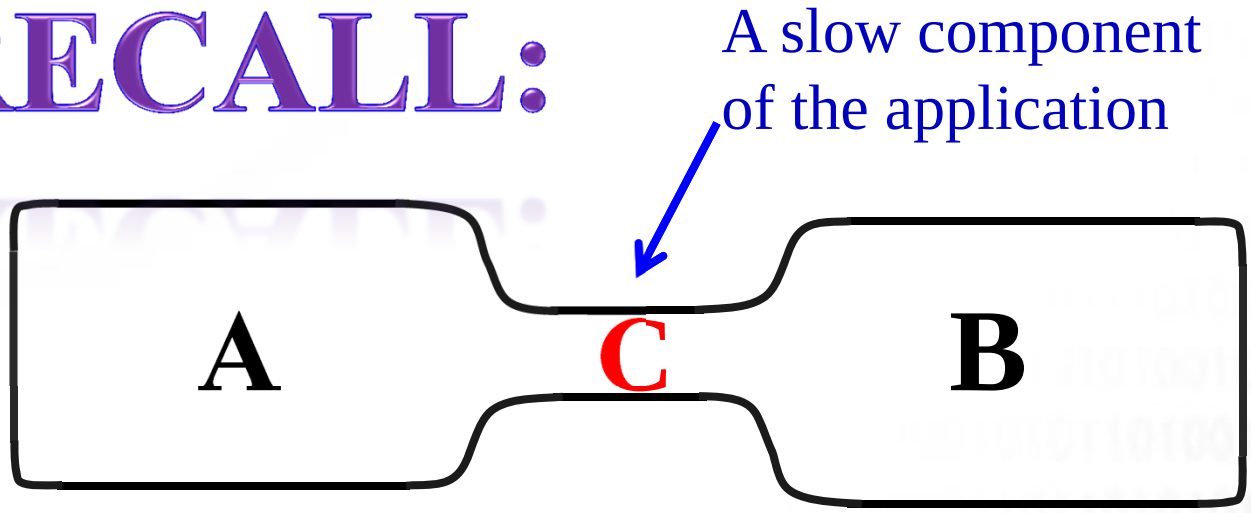
Constructing Data For a Learner and Learning Rules

Input_1_Name	Input_2_Name	...	Input_k_Name	Class
Value_p	Value_q	…	Value_m	Good
Value_p	Value_q	…	Value_m	Bad
…	…	…	…	…
Value_p	Value_q	…	Value_m	Good



Finding Bottlenecks

RECALL:



Bottlenecks or hot spots are phenomena where the performance of the application is limited by one or few components

Finding Bottlenecks

Cluster Execution Traces

We want to generalize a set of execution traces, compress information, and find common patterns

- All execution traces in the same set share some common properties

It is much easier to find differences between few patterns than between billions of pairs of execution trace entries

- These differences should tell us what properties are specific to execution traces in the same set

Intuition

There are some patterns in profiles that are repeated for different executions, for example, logging, database accesses

These patterns can be thought of as components that correspond to implementations of some requirements

We should reduce many different method calls in different profiles to a small set of components that offer us some insight into what methods are important for meaningful tests

It Is A Difficult Problem

Too Much Data To Process

We have too many observations and dimensions

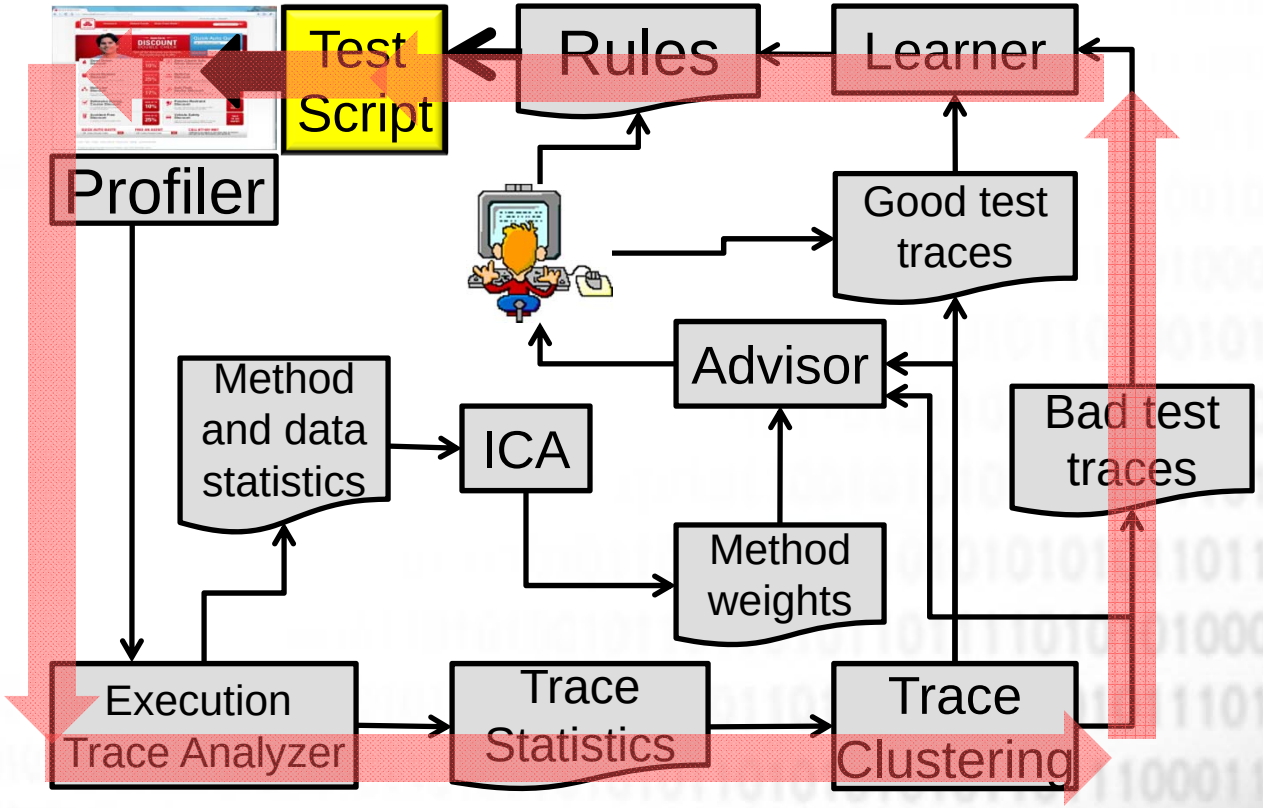
- Many inputs, key/values, method invocations to reason about or obtain insights from
- Too much noise in the data

Too many dimensions for humans to deal with – think requirements

- Need to reduce them to a smaller set of factors
- Better representation of data without losing much information

Build more effective data analyses on the reduced-dimensional space: classification, clustering, pattern recognition

FOREPOST Architecture



A Highlight Of Our Results

The Method **checkWildFireArea**

This method is instrumental in computing quotes for the Renters application

- Thus this method is important for testing
- With FOREPOST, this method is selected as the top of 30 from over 3,000 methods

However, with the state MN this method does not affect quote computation

- Yet FOREPOST identified this method is important.

Why?

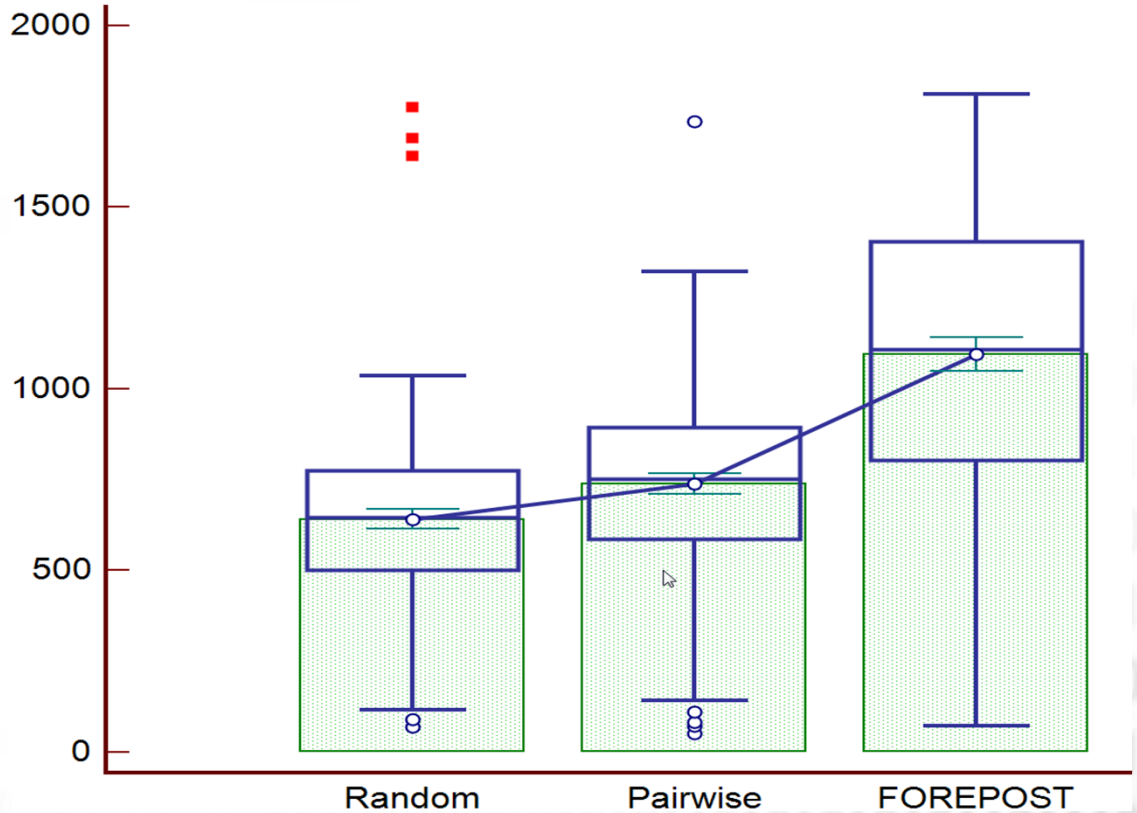
Discovery Of a Problem

FOREPOST automatically selected the method **checkWildFireArea** as important because its weight is significant in interesting profiles.

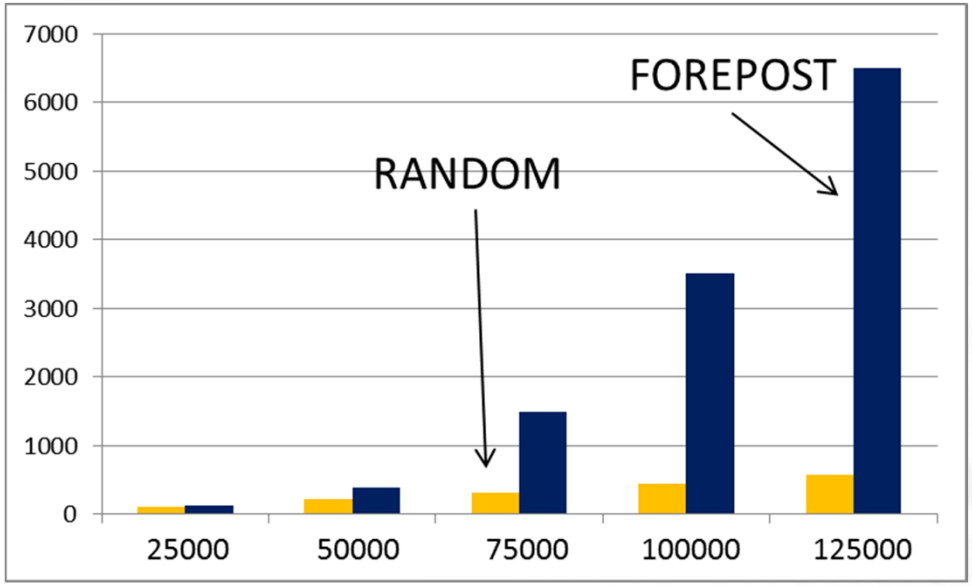
It means that this method is invoked many times for the state MN even though its functional contribution is zero for this state. Invoking this method consumes resources and time for this state.

It took few hours and several developers to uncover this information based on the results of running DATE. When we reported it developers thought that we made a mistake since this method was not supposed to execute on MN.

Result for Renters App



Result for JPetStore



Conclusions

This proposed research program is novel, as to the best of our knowledge, there exists little but a growing research that addresses the problem of the controlled release of sensitive information that balances privacy and software engineering tasks.

The results of this work will be a foundation for a new direction in requirements engineering, program comprehension, globally distributed software development, maintenance, evolution, and testing supported by a set of tools for low-cost automated software engineering tasks that consider software privacy issues.

Email: drmark@uic.edu