Context-Aware Statistical Debugging

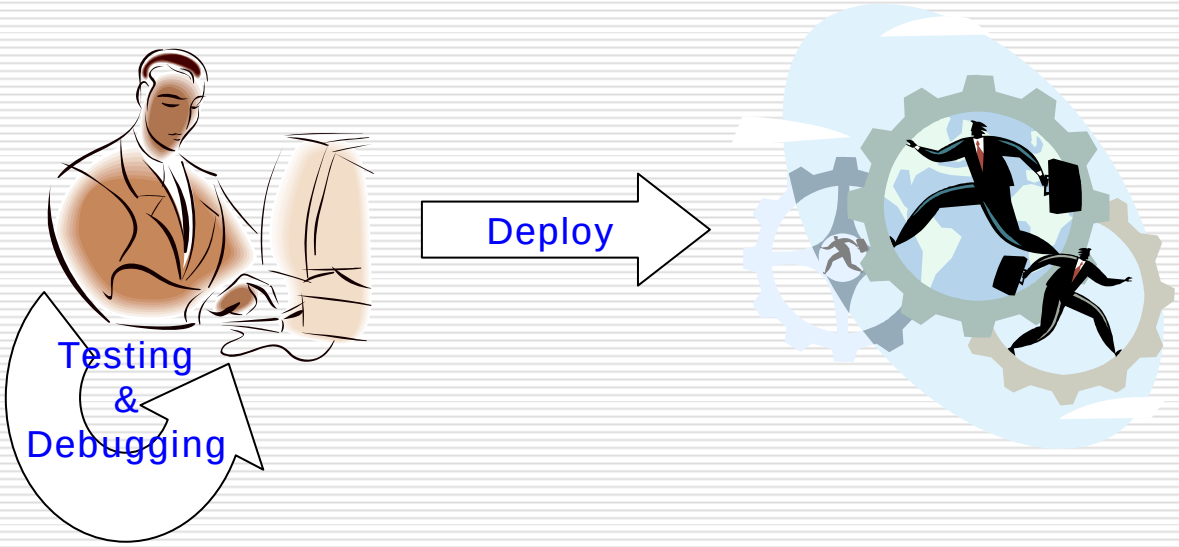
From Bug Predictors to Faulty Control Flow Paths

Lingxiao Jiang and Zhendong Su
University of California at Davis

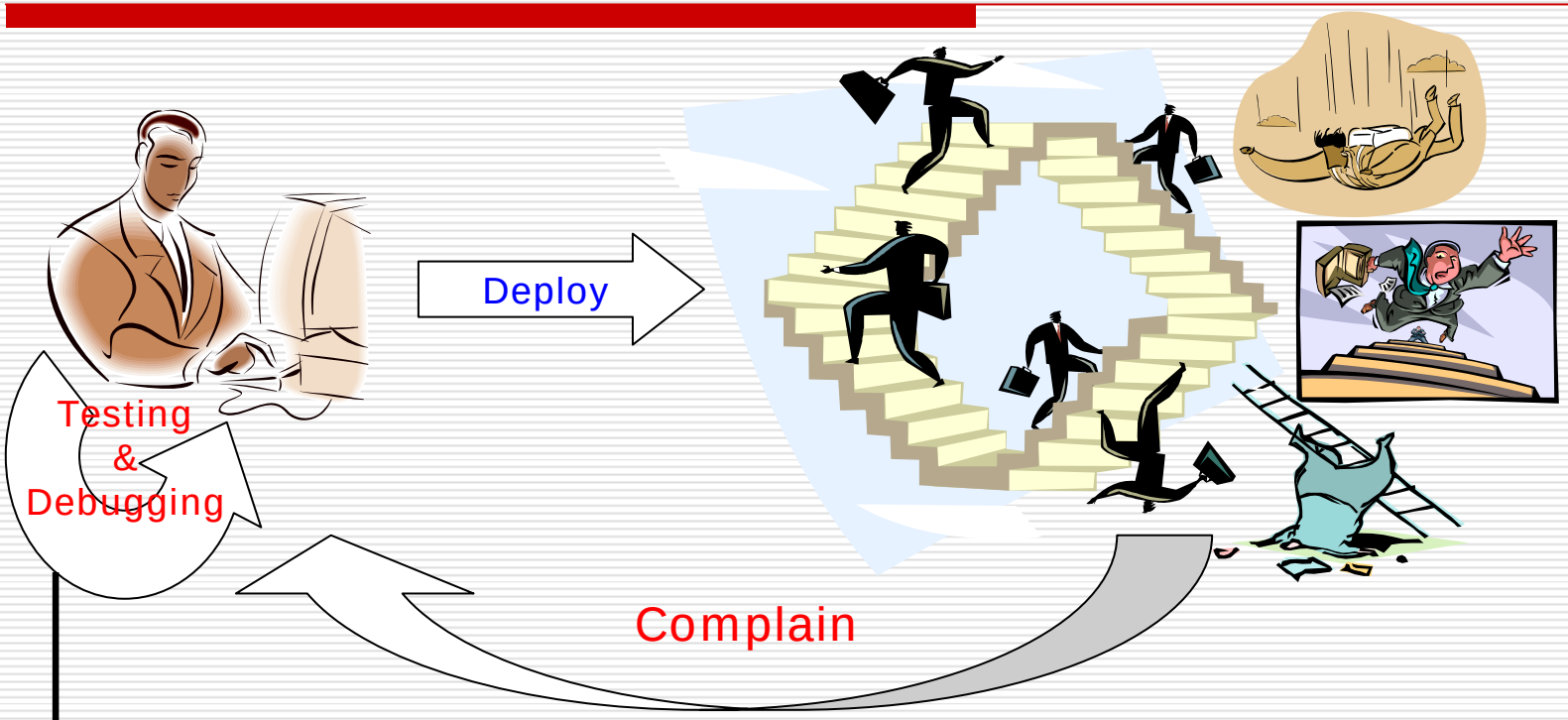
Outline

- Introduction
- Context-Aware Statistical Debugging
- Empirical Evaluation
- Related Work and Conclusion

Why Need Debugging Aids?



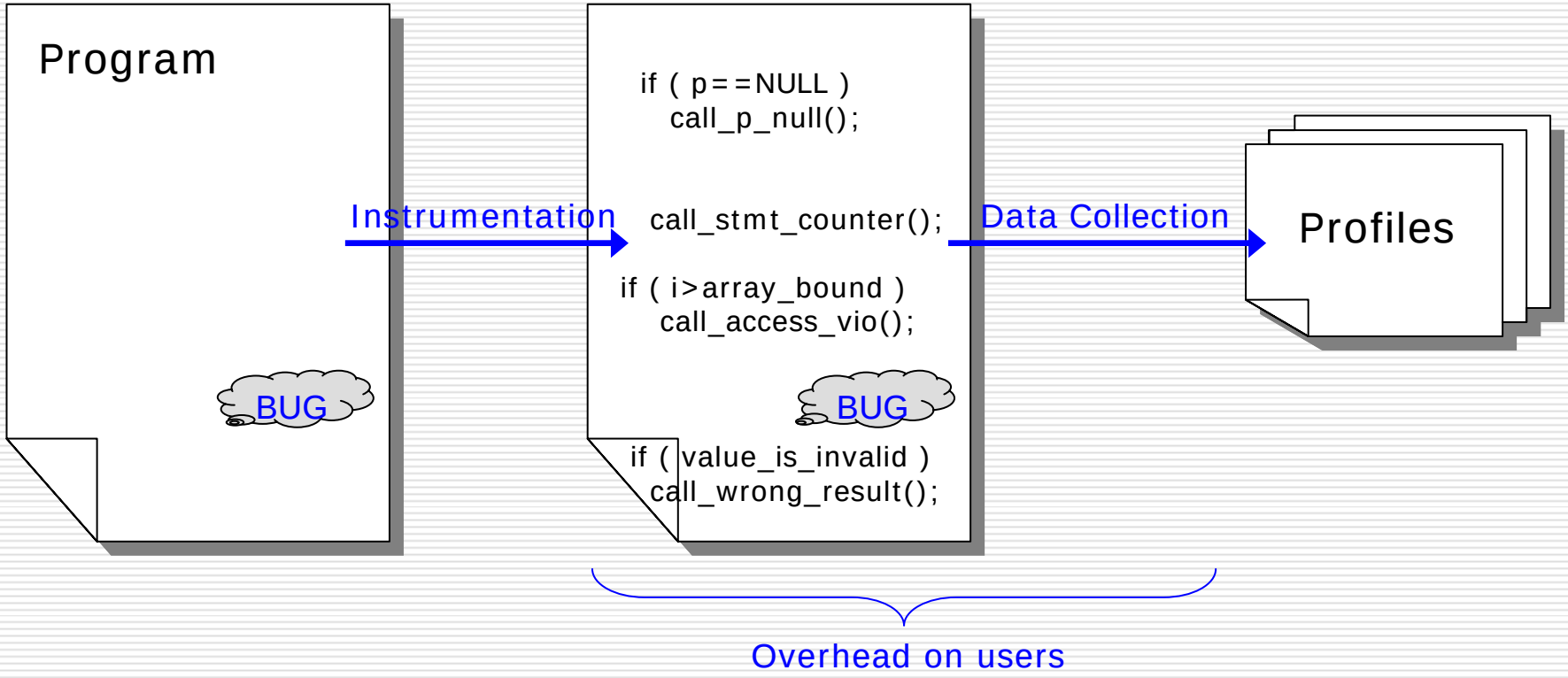
Why Need Debugging Aids?



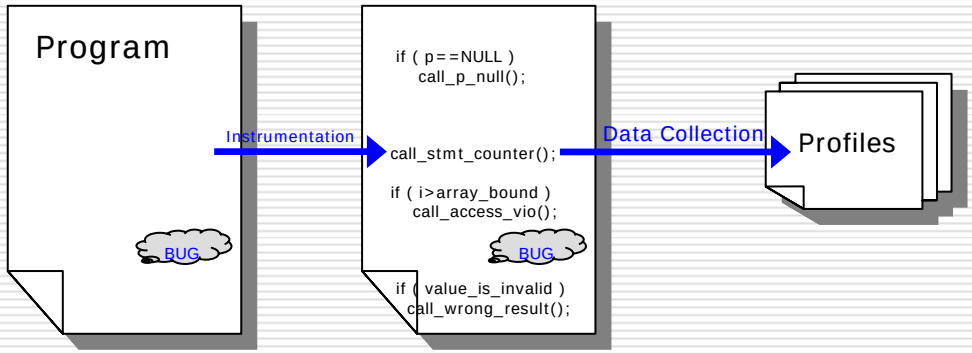
Debugging is time-consuming:

- Hard to reconstruct failure scenarios
- Hard to unveil root causes

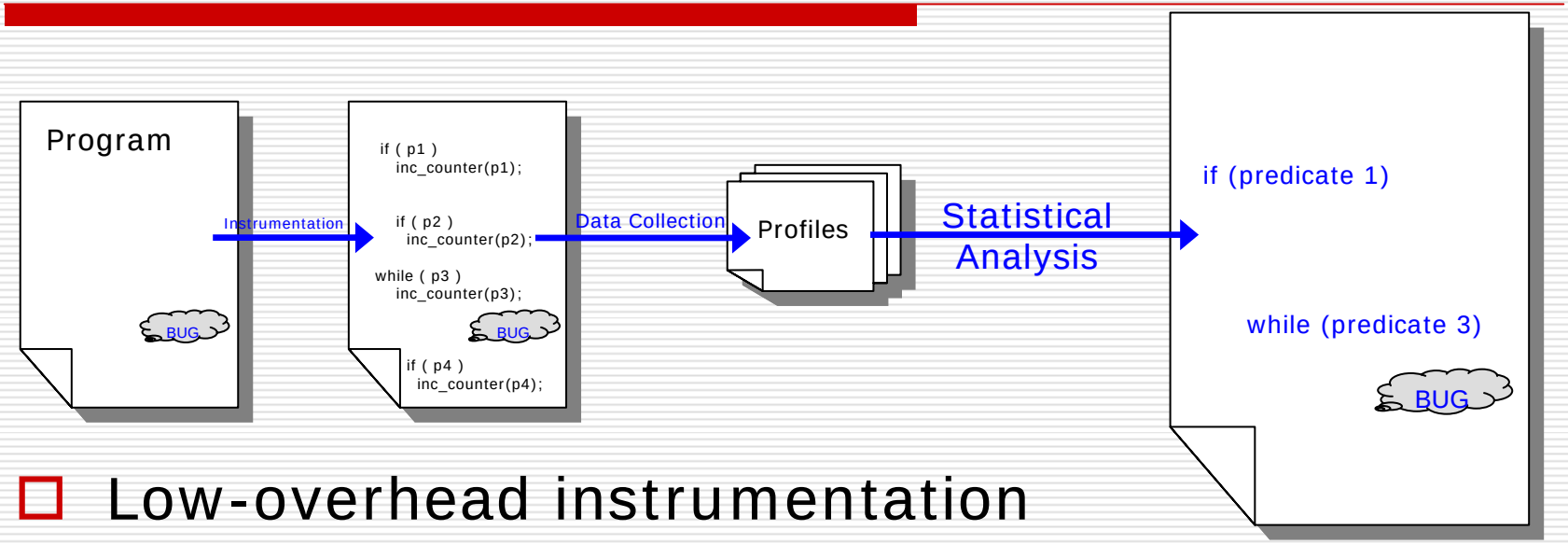
Failure Analysis based on Feedback



Statistical Debugging

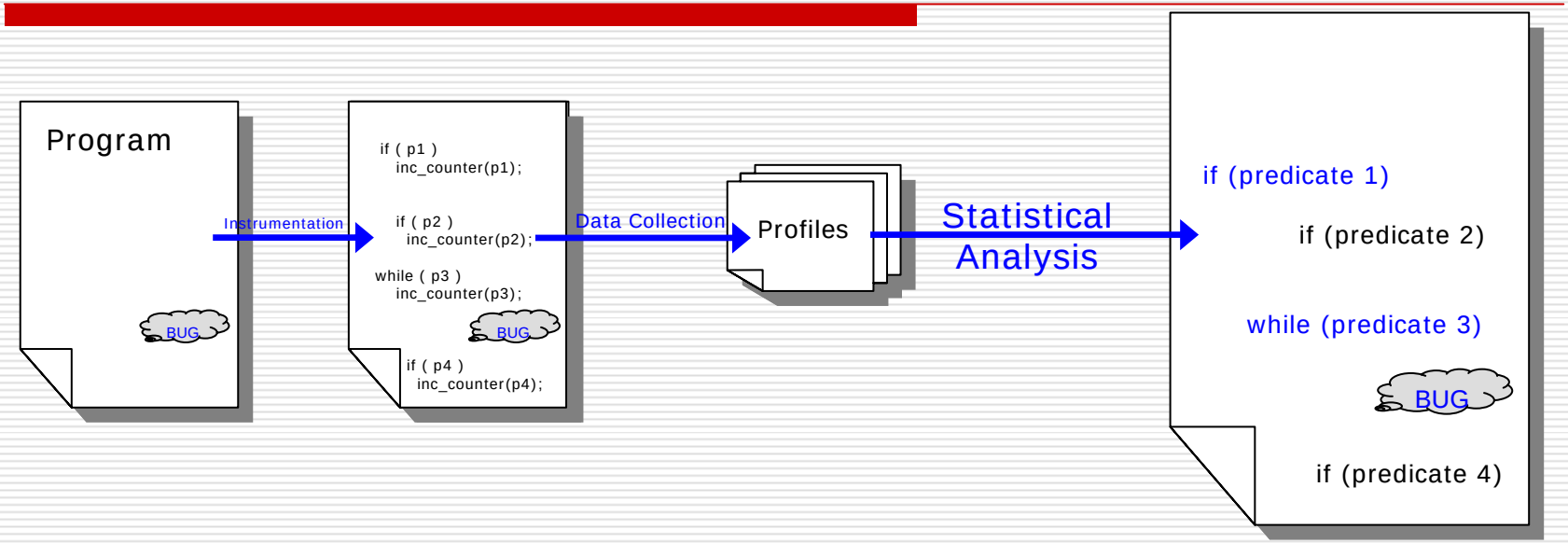


Statistical Debugging



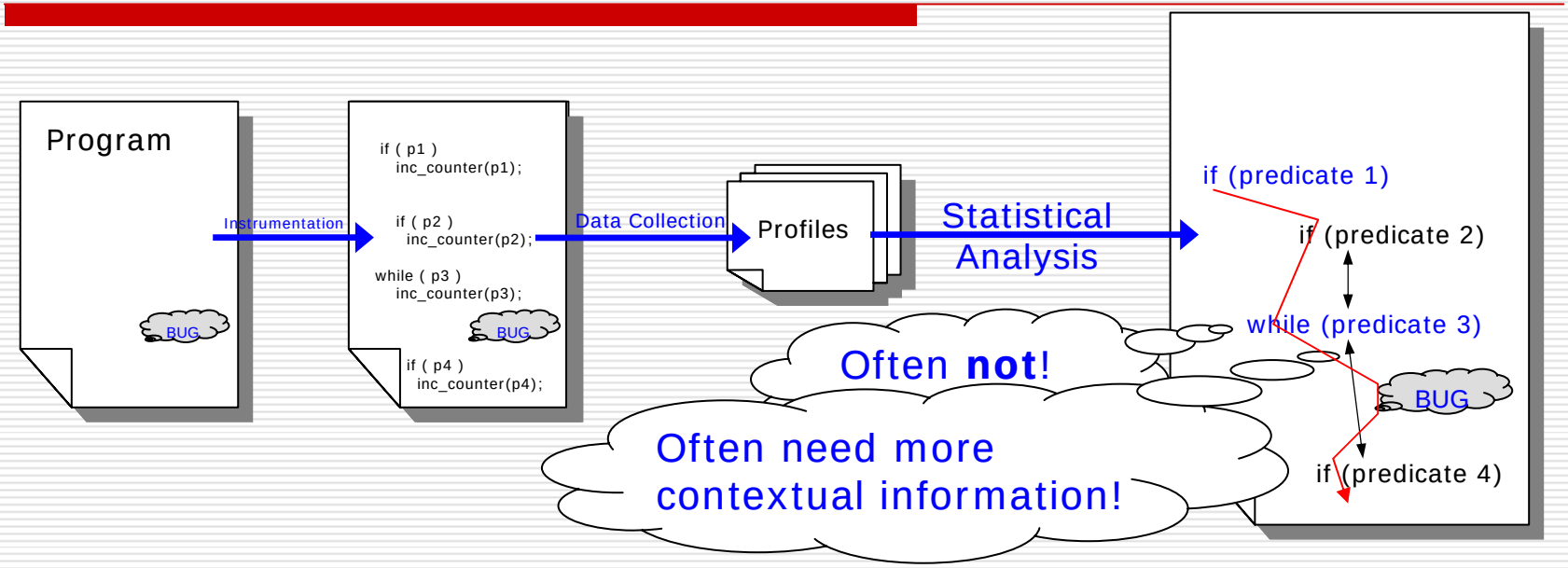
- Low-overhead instrumentation
  - Simple program predicates
  - Sampled instrumentation
- Statistics-based analysis
  - Which instrumented program predicates are more likely related with failures

Statistical Debugging - Questions



- Do these program predicates (i.e. **bug predictors**) directly indicate the locations of failure causes?

Statistical Debugging - Questions

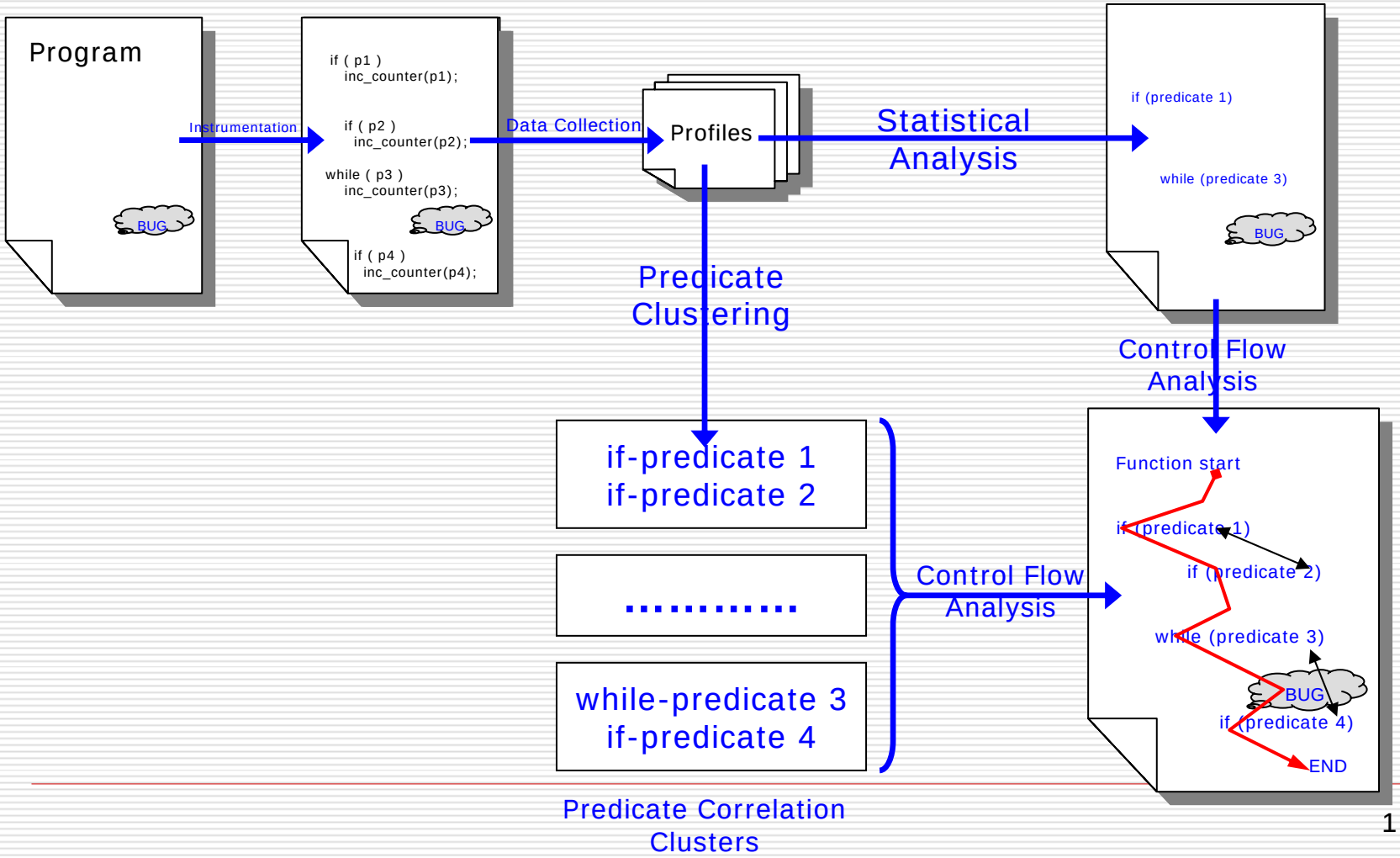


- Do these program predicates (i.e. **bug predictors**) directly indicate the locations of failure causes?

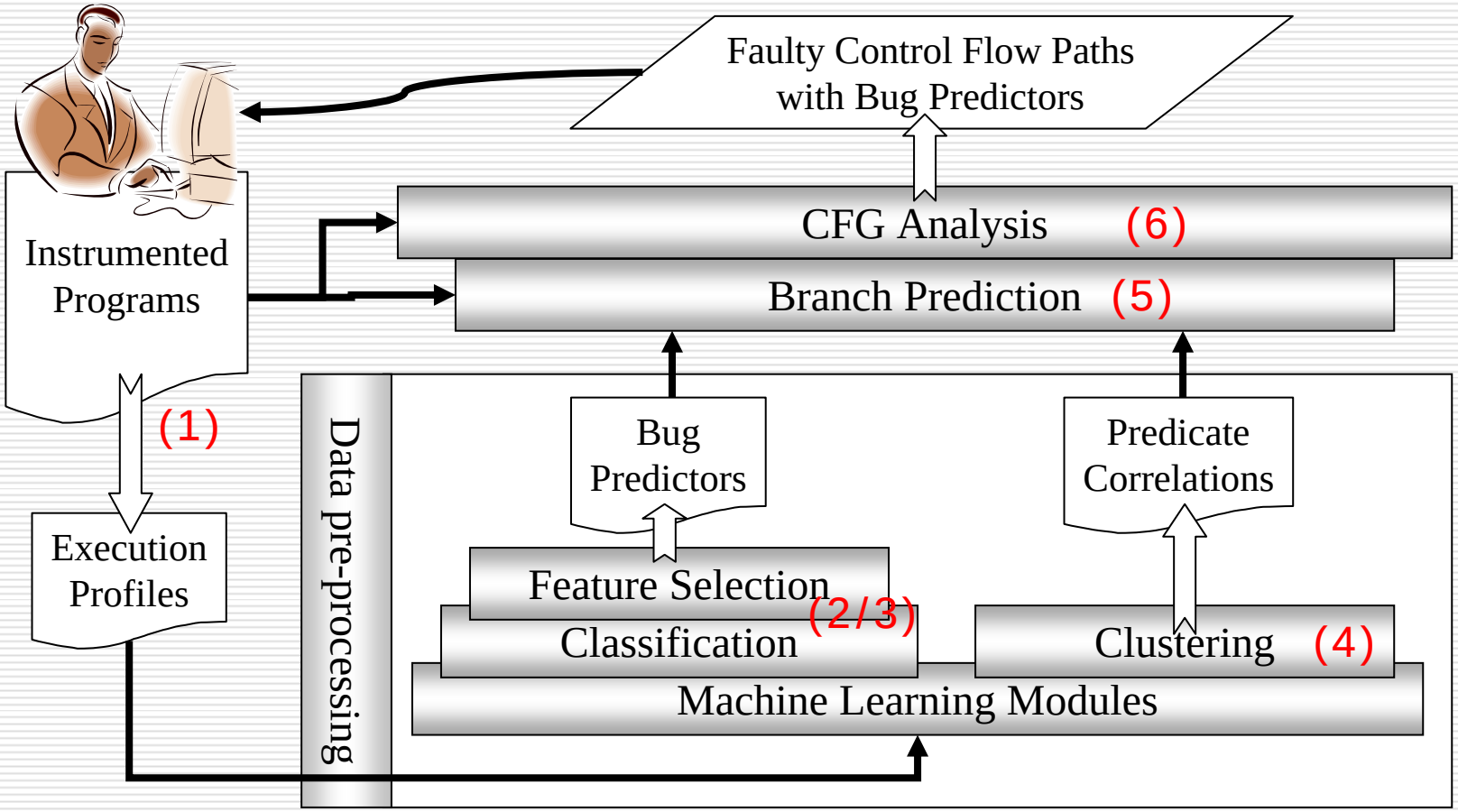
Outline

- Introduction
- Context-Aware Statistical Debugging
- Empirical Evaluation
- Related Work and Conclusion

Context-Aware Statistical Debugging



Approach Overview



Approach Details (1/6)

- Profiling mechanism
  - Based on the *Cooperative Bug Isolation* project
  - Pre-selected instrumentation predicates
    - Branch conditions
    - Return values
    - Pair-wise comparisons among scalar-values

- Profile for each execution
  - A label: "success" or "failure"
  - A numerical vector: the number of times each predicate is observed to be true

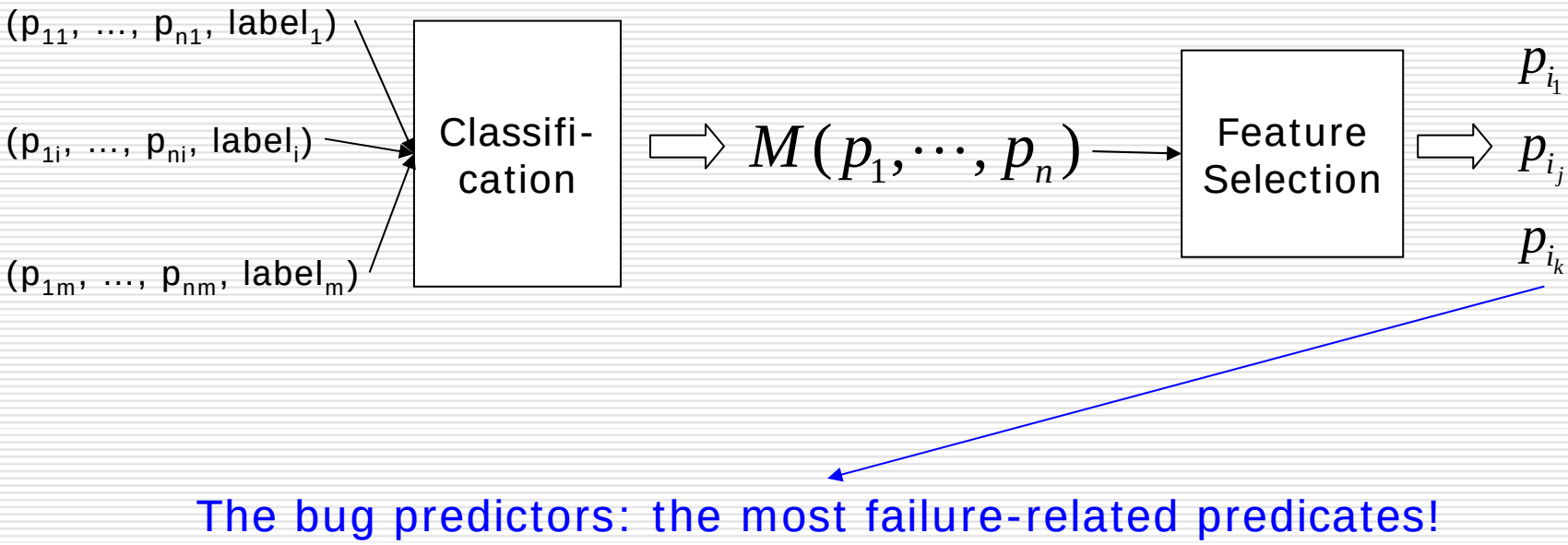
Approach Details (2/6)

- $(p_{11}, \ldots, p_{n1}, label_1)$, $(p_{1i}, \ldots, p_{ni}, label_i)$, $(p_{1m}, \ldots, p_{nm}, label_m)$

- Look for bug predictors (predicates)
  - Establish relationship models between the predicates and the labels
    - $Label = M(p_1, \cdots, p_n)$
    - Based on machine learning algorithms. E.g. *Support Vector Machines*, *Random Forests*.
  - Select the most failure-relevant predicates
    - Compute predicate impact scores based on the models

Approach Details (3/6)

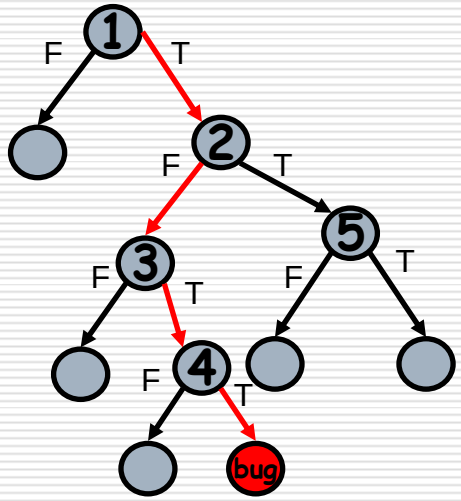
- A black box view of machine learning



The bug predictors: the most failure-related predicates!

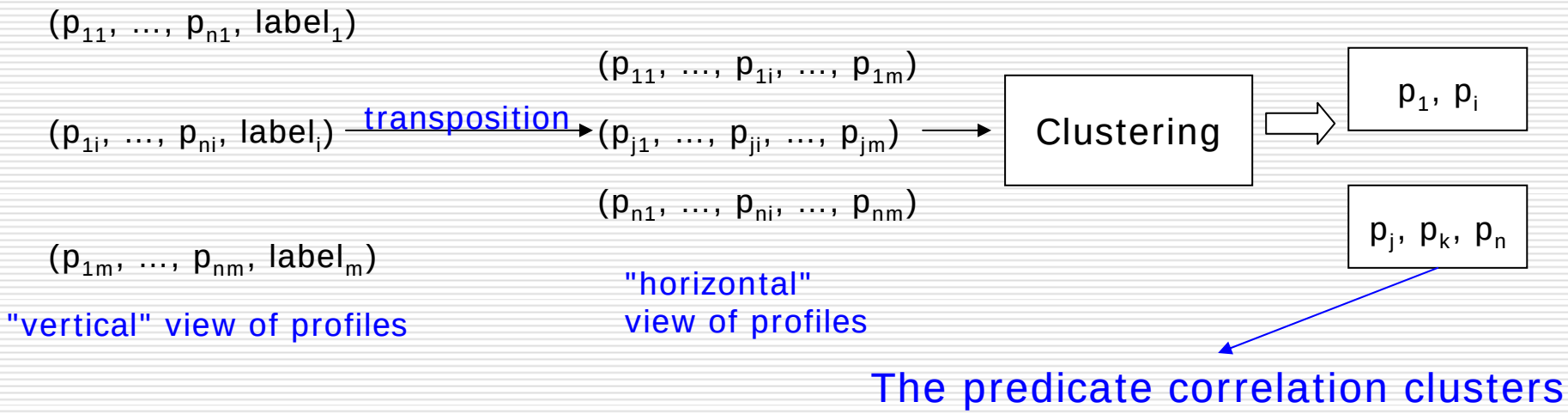
Approach Details (4/6)

- Look for relationships among the predicates
  - **Purpose**: Provide additional hints about the cause-failure transition chains



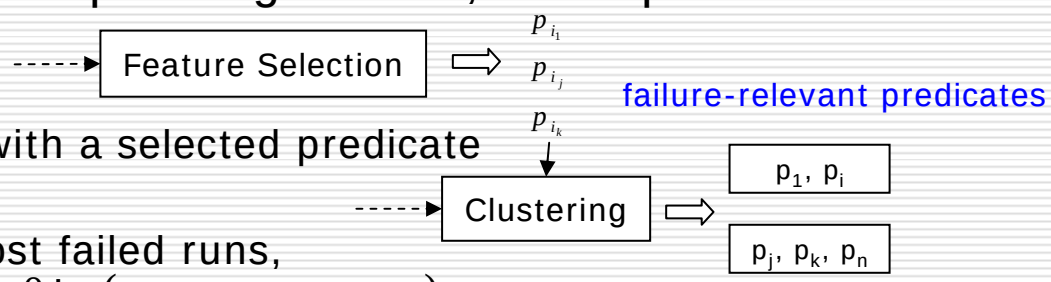
Approach Details (4/6)

- Look for relationships among the predicates
  - **Purpose**: Provide additional hints about the cause-failure transition chains
  - **Method**: Search for predicates of similar execution histories in *failed* runs using clustering



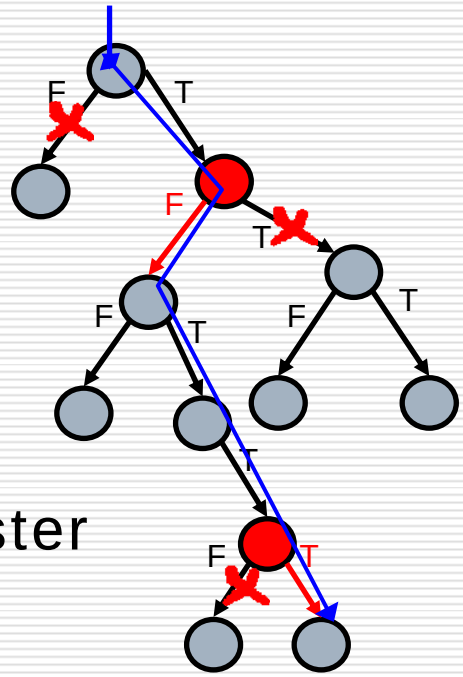
Approach Details (5/6)

- Branch prediction
  - **Purpose**: Decide branch directions that may lead to failures
  - **Method**: Rely on feature selection and clustering. For each branch predicate,
    - Traverse its corresponding branch, if the predicate is
      - Selected,
      - Or clustered with a selected predicate
      - Or, true in most failed runs, i.e., most $p_{ji} > 0$ in $(p_{j1}, \cdots, p_{ji}, \cdots, p_{jm})$ "horizontal" profiles
    - Otherwise, traverse both branches.



Approach Details (6/6)

- Control flow path generation
  - Depth-first, inter-procedural graph traversal, until all selected predicates are met
  - Directed by the branch predictions
  - Return a set of faulty paths that may contain bug locations
  - Linear time w.r.t. program sizes
  - The more predicted branches, the faster



Outline

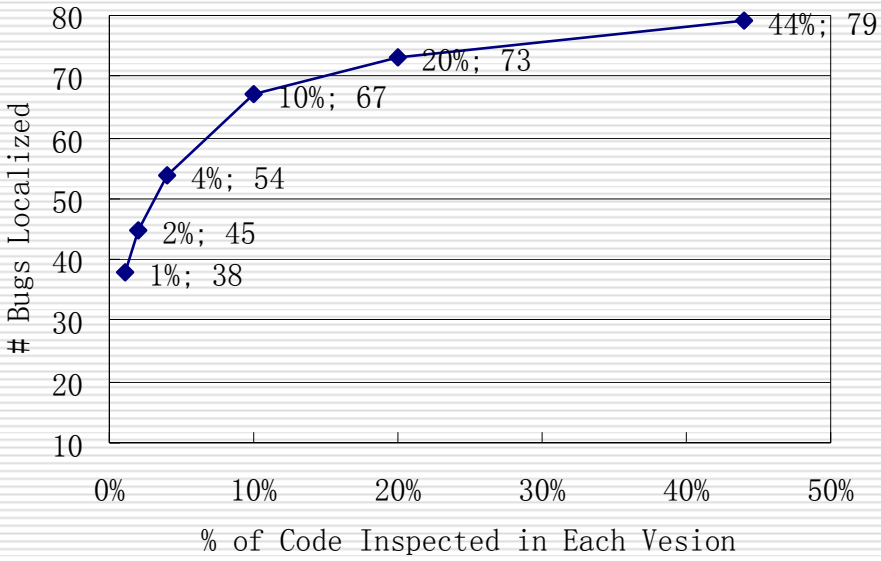
- Introduction
- Context-Aware Statistical Debugging
- Empirical Evaluation
- Related Work and Conclusion

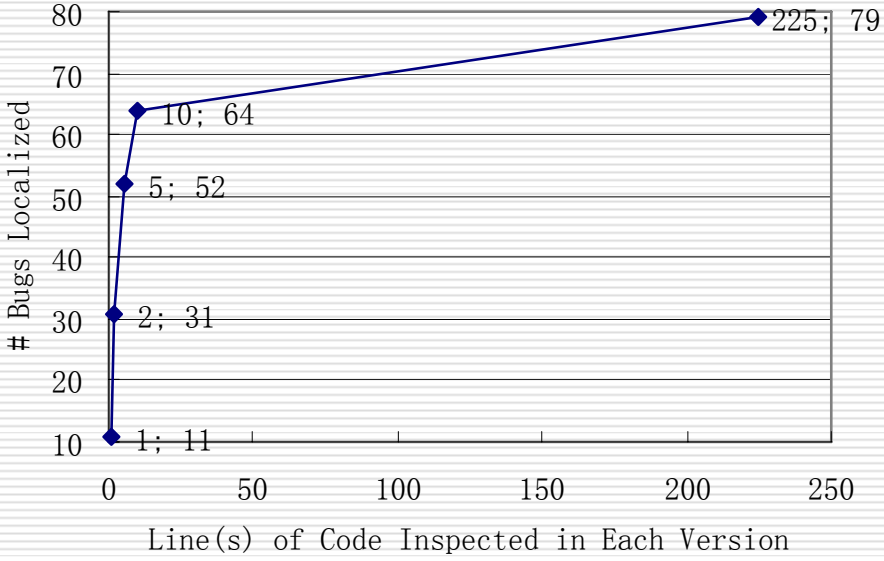
Subject Programs

- Siemens test suite (STS)
  - 132 faulty versions for seven C programs
  - Hundreds of LoC in each version
  - ~43,400 LoC
- Rhythmbox 0.6.4
  - A multi-threaded music management program for GNOME
  - ~56,500 LoC

Result Summary for STS

- 79 bugs were localized in the faulty control flow paths that cover 9.2% of the code
- 38 bugs were localized by inspecting no more than 1% of the code





Result Summary for Rhythmbox (1/2)

- 5 bugs (2 unreported) localized
- A simple example:

```
......
i < impl_array->len
monkey_media_is_alive() == FALSE
......
```

A sample predicate cluster

```
static gboolean alive = FALSE;

......

void monkey_media_shutdown (void) {
    if ( monkey_media_is_alive () == FALSE )
        return;

    alive = FALSE;
    ......

    for (...; i<impl_array->len; ...)
        ...free memory...;
}
......

gboolean monkey_media_is_alive (void) {
    return alive;
}
```

Failure due to race condition!

Result Summary for Rhythmbox (2/2)

Time Cost

Instrumentation Predicates	# of Predicates	Time of Predicate Selection (min)	Time of Predicate Clustering (min)	Time of Path Generation (min)
Branch conditions	6,863	41	30	<1
Return values	25,287	45	770	<1

- Worthwhile w.r.t. many hours spent on traditional testing and debugging, especially for a real world application that we were not familiar with and that contains unknown bugs.

Outline

- Introduction
- Context-Aware Statistical Debugging
- Empirical Evaluation
- Related Work and Conclusion

Related Work (1/2)

- [Lal ESOP 2006]: Look for *shortest* paths that connect together as many bug predictors as possible, based on weighted pushdown systems
  - Exponential w.r.t. the number of given predicates
  - No consideration on predicate correlations

- [Brun ICSE 2004]: fault-localization with *Daikon* (a dynamic invariant detection tool) and machine learning.
- [Liblit PLDI 2005, Zheng ICML 2006]: statistical debugging for *sparsely* sampled profiles and programs with multiple bugs.
- [Liu ESEC/FSE 2005]: statistical debugging with a different predicate selection strategy.
- Look for bug predictors only.

Related Work (2/2)

- [Jones ICSE 2002, ASE 2005]: fault-localization with a *statement*-level instrumentation and visualization
  - Rank and visualize statements in the order of their bug-relevancy
  - Use statements as bug predictors

- [Zeller FSE 2002, Cleve ICSE 2005]: delta debugging with known failure-inducing inputs
  - Focus in space on failure-related variables and in time on cause transitions
  - Know all program states within failed and succeeded runs

Conclusion

- A context-aware approach for statistical debugging based on the Cooperative Bug Isolation project
  - Identify bug predictors accurately
  - Cluster correlated program predicates
  - Construct faulty control flow paths

- As a result, more contextual information (predicate correlations, control flow paths) for debugging and less code inspection burden!

Thank you!

Questions?

jiangl@cs.ucdavis.edu