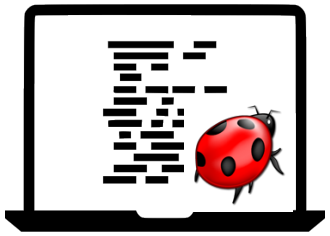
Better Code Search and Reuse for Better Program Repair

Qi Xin and Steven P. Reiss

Automated Program Repair

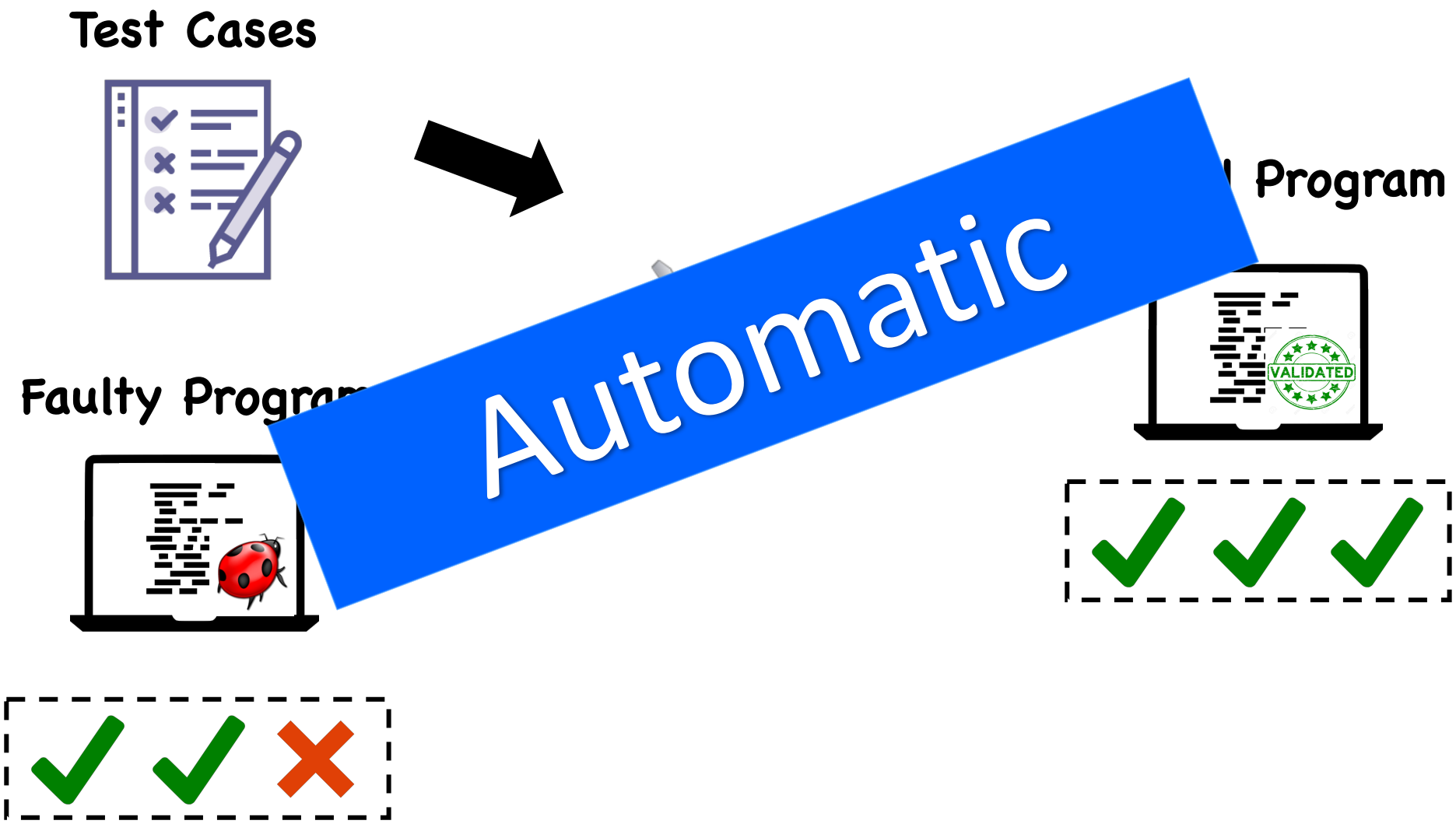
Test Cases

Faulty Program



Patched Program

Automated Program Repair

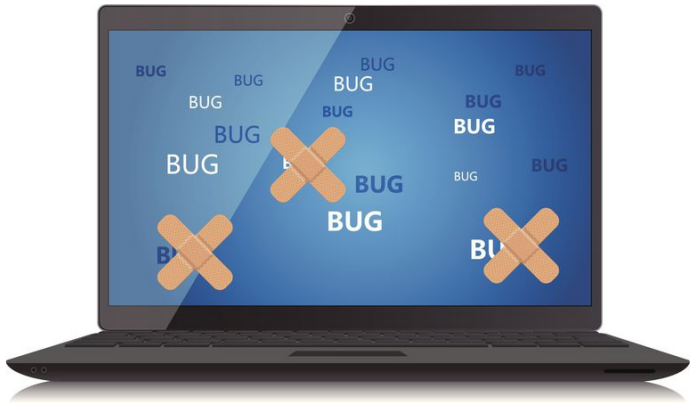


Repair Steps

1

Fault Localization

2



Patch Generation

3

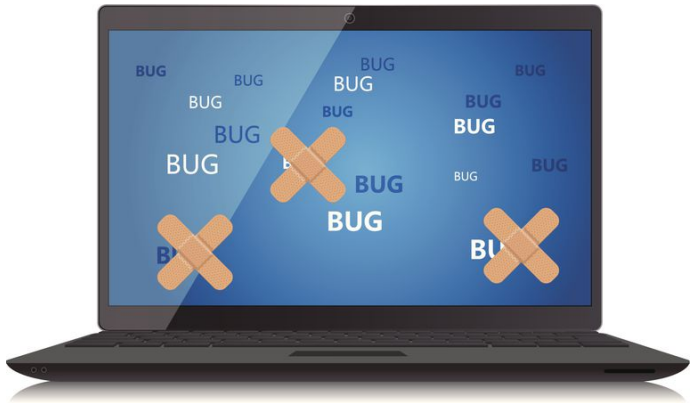
Patch Validation

Repair Steps

1

Fault Localization

2



Patch Generation

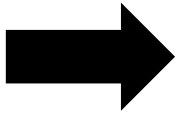
3

Patch Validation

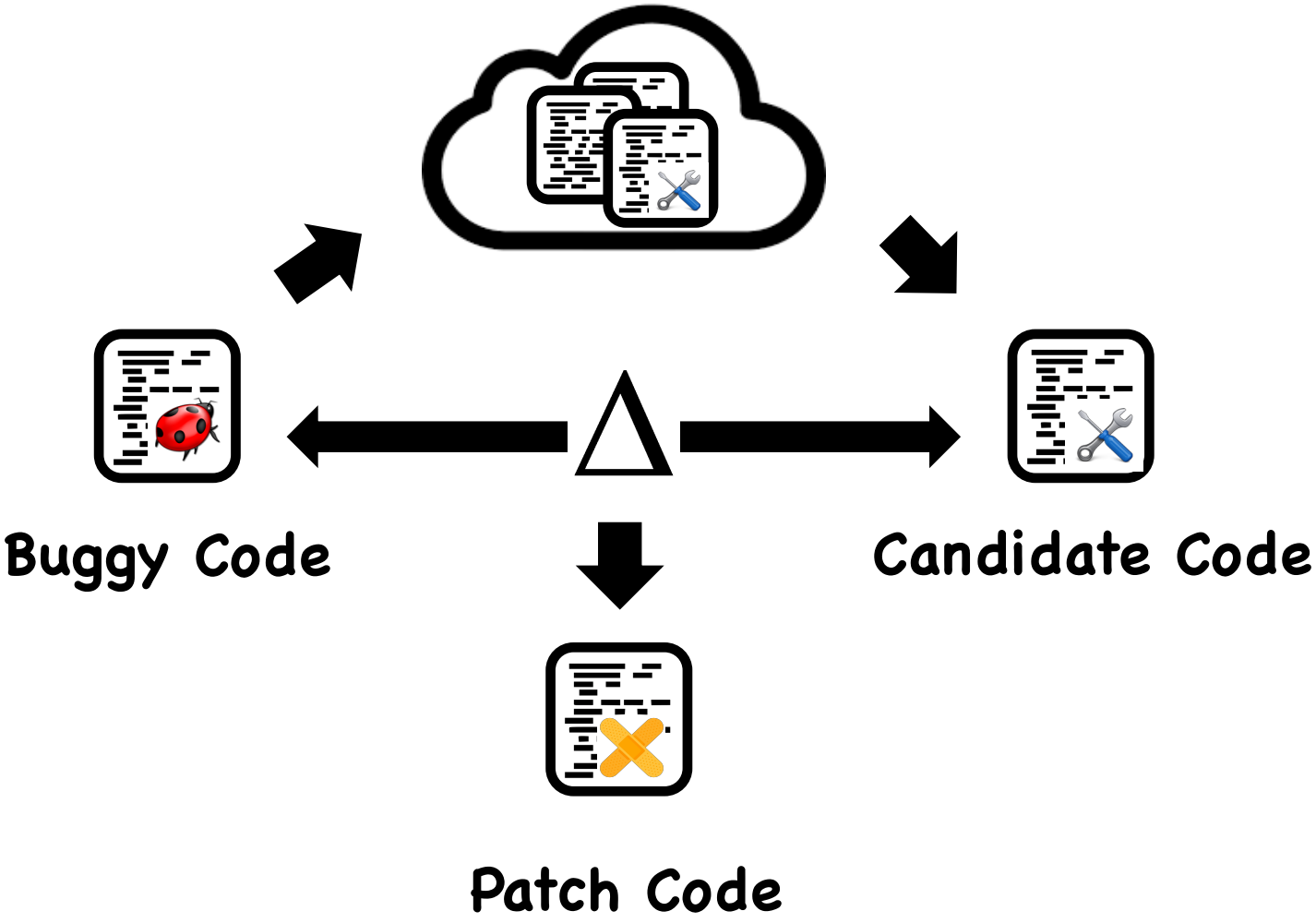
Patch Generation

- Genetic Algorithms
- Human-Written Templates
- Historical Fixes
- Code Synthesis
- **Code Search**
- State monitoring
- Invariants
- Bug reports
- …

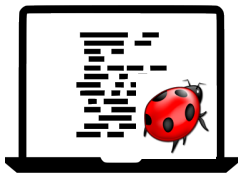
Patch Generation

- Genetic Algorithms
- Human-Written Templates
- Historical Fixes
- Code Synthesis
- **Code Search**  **ssFix (Xin and Reiss, ASE'17)**
- State monitoring
- Invariants
- Bug reports
- …

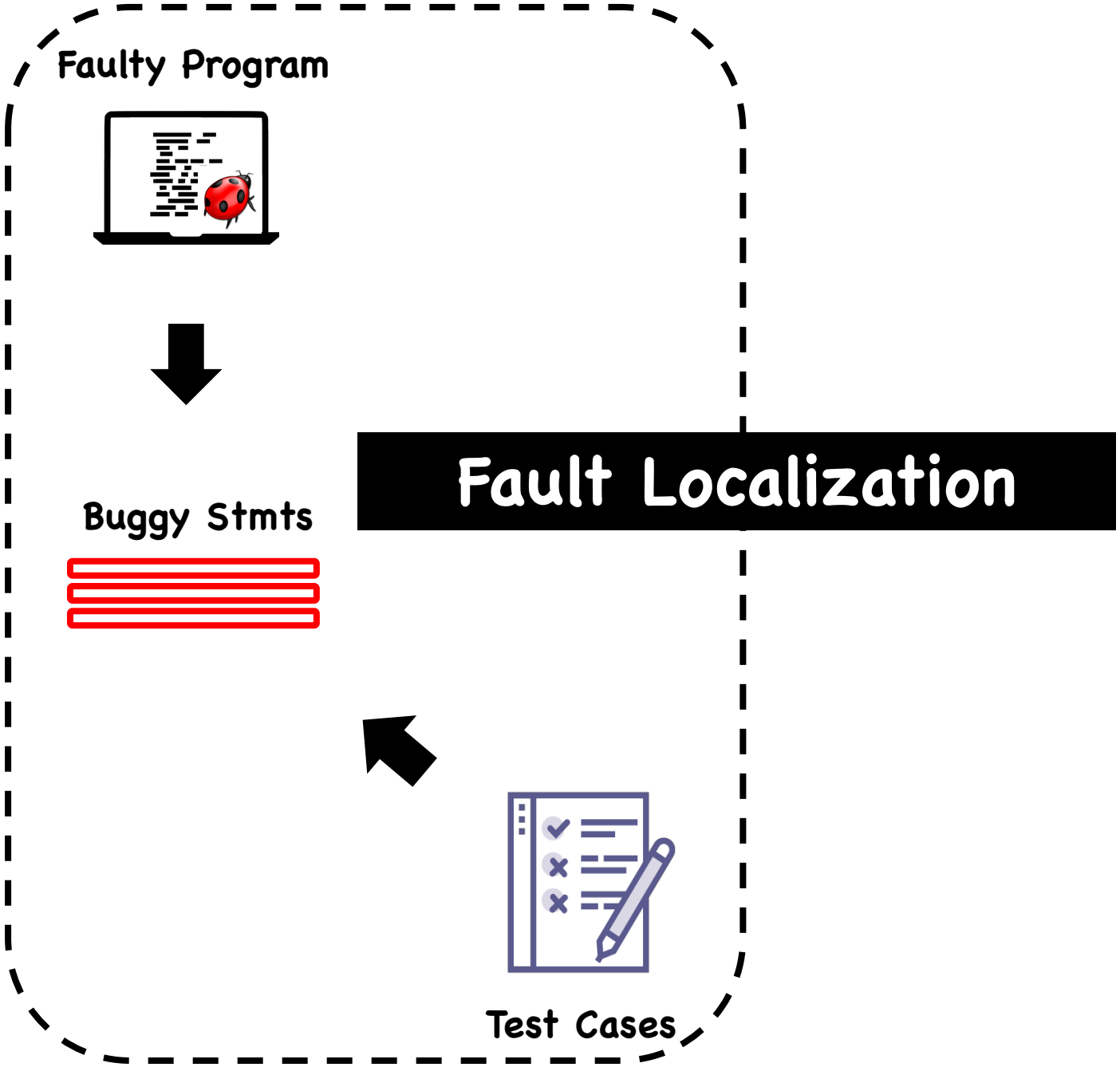
The Idea of ssFix



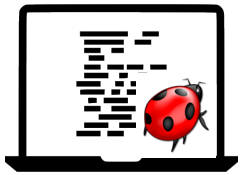
Faulty Program

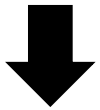


Test Cases


Faulty Program
Buggy Stmts
Fault Localization
Test Cases

Faulty Program

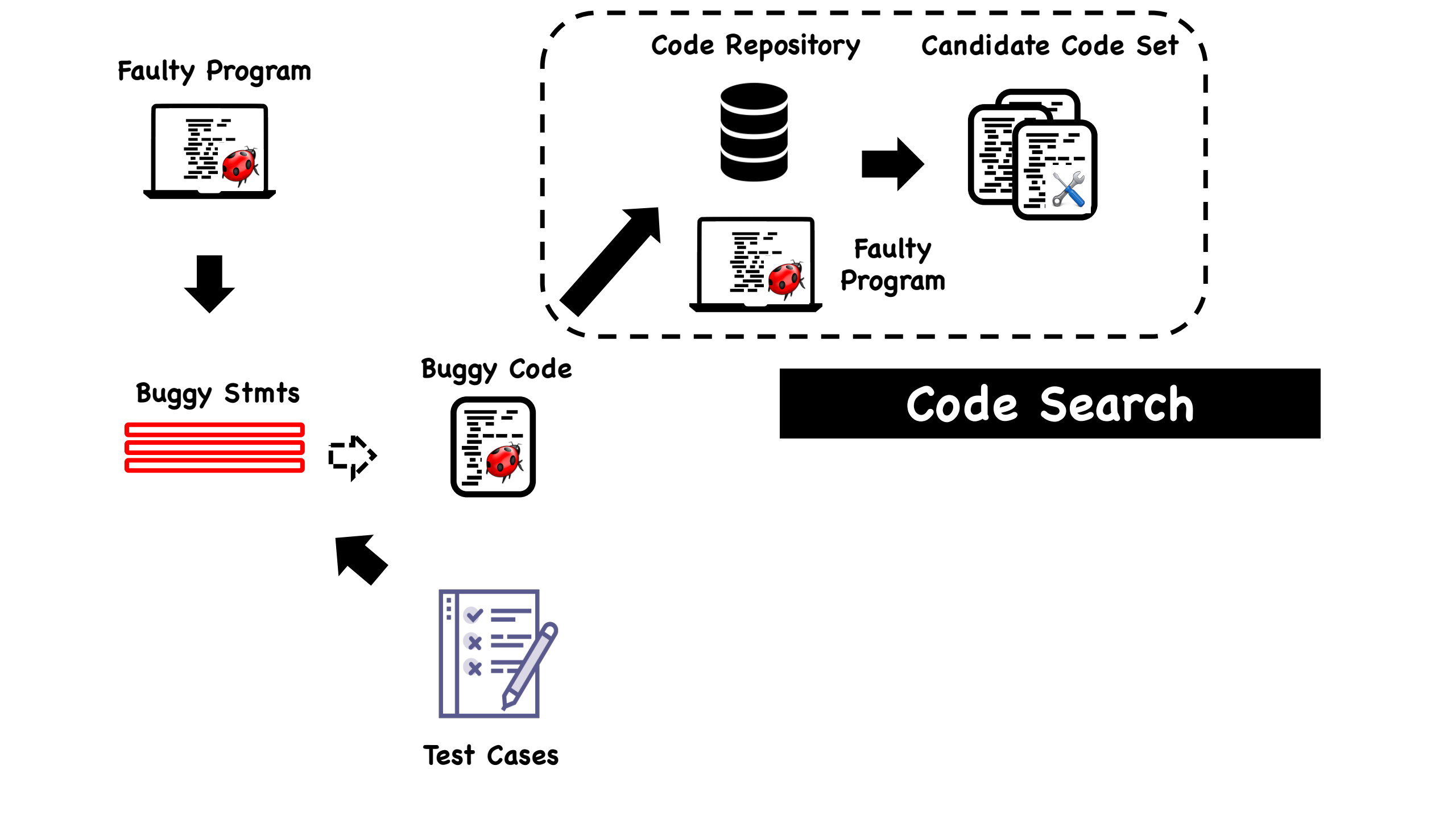


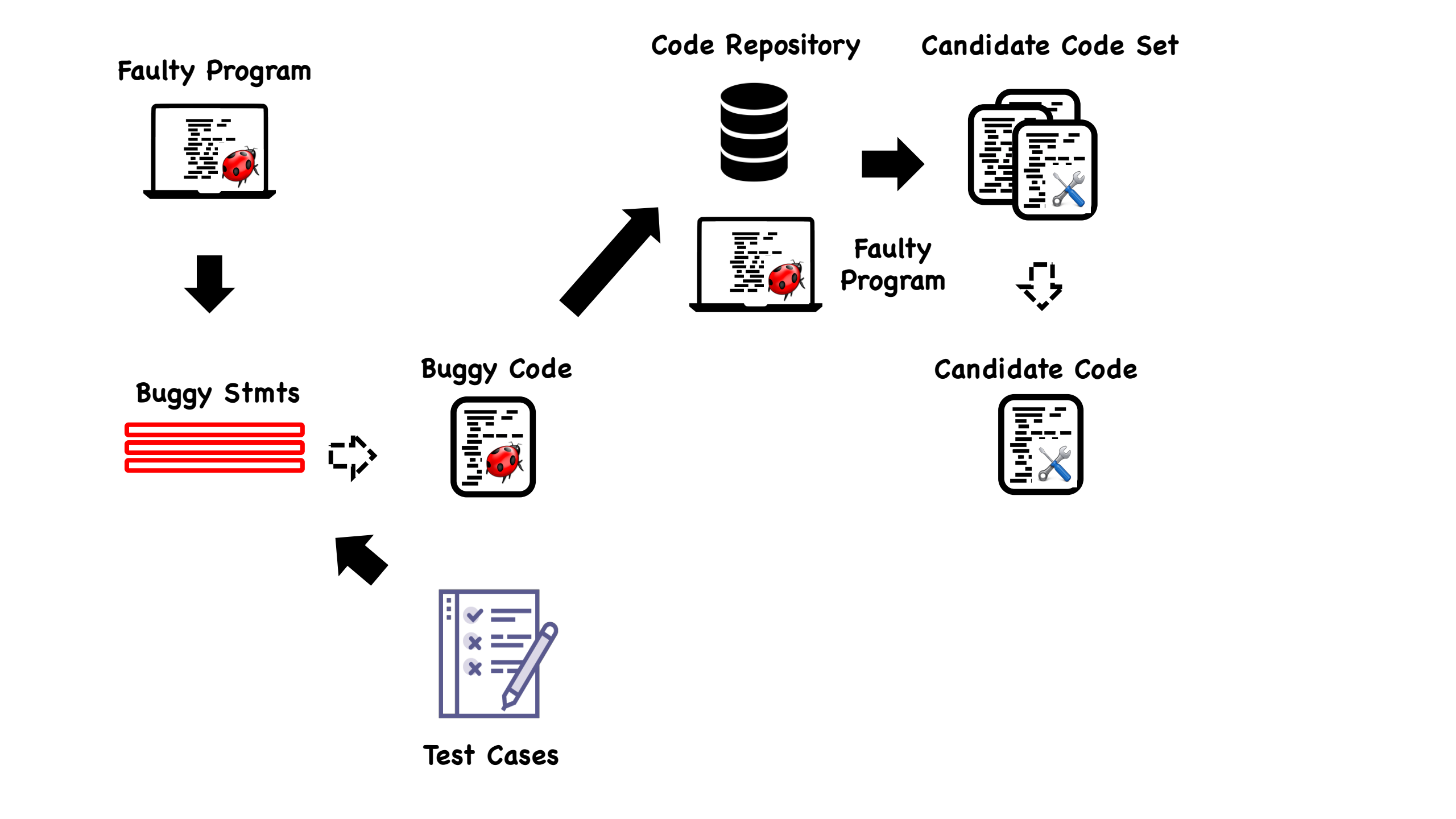


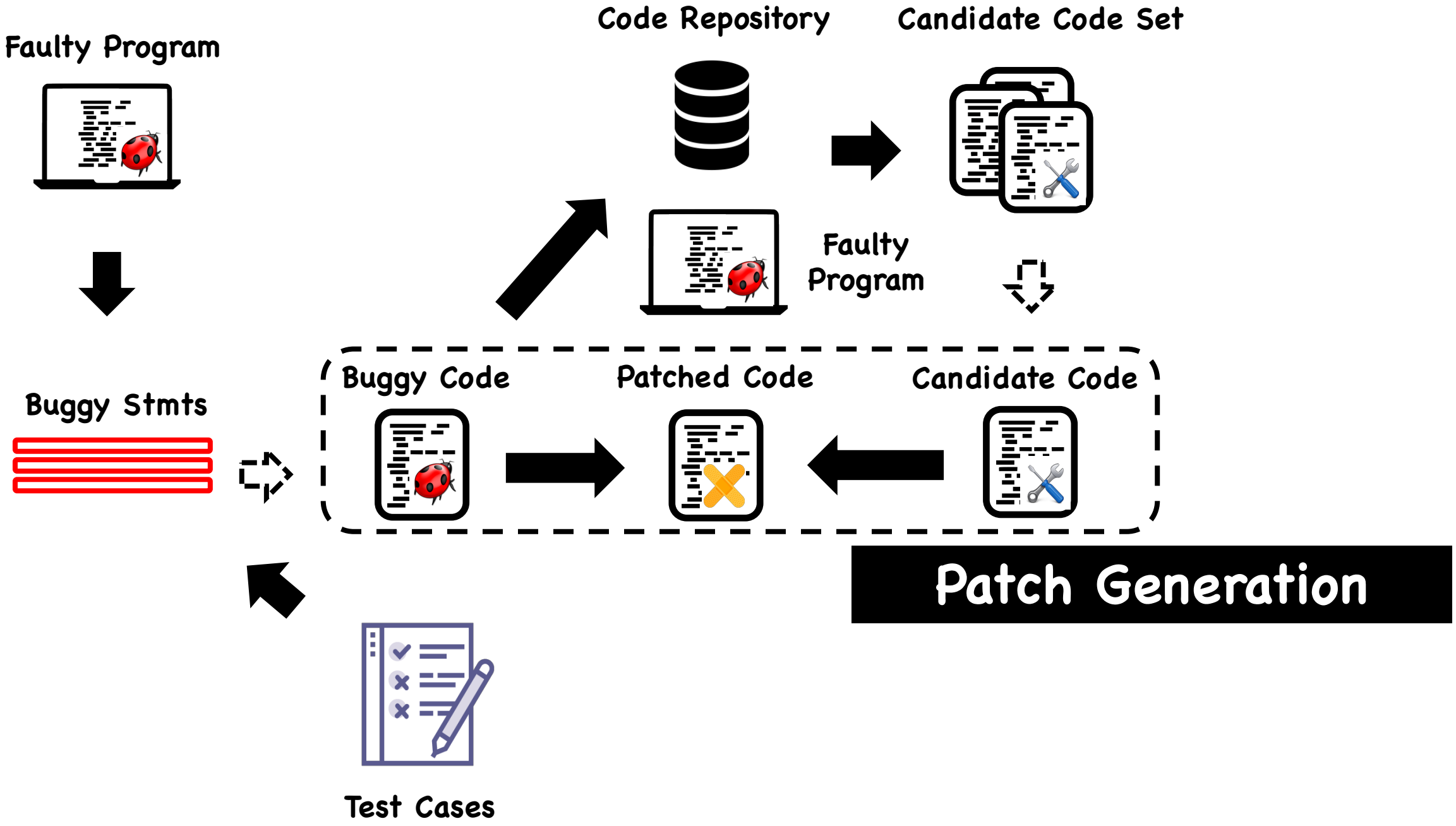
Buggy Stmts

Buggy Code

Test Cases


Faulty Program
Code Repository
Candidate Code Set
Faulty
Program
Buggy Code
Buggy Stmts
Code Search
Test Cases


Faulty Program
Code Repository
Candidate Code Set
Faulty Program
Buggy Stmts
Buggy Code
Candidate Code
Test Cases


Faulty Program
Code Repository
Candidate Code Set
Faulty Program
Buggy Stmts
Buggy Code
Patched Code
Candidate Code
Patch Generation
Test Cases

Patch Generation

1. Code Translation
2. Code Matching
3. Modification

Code Translation

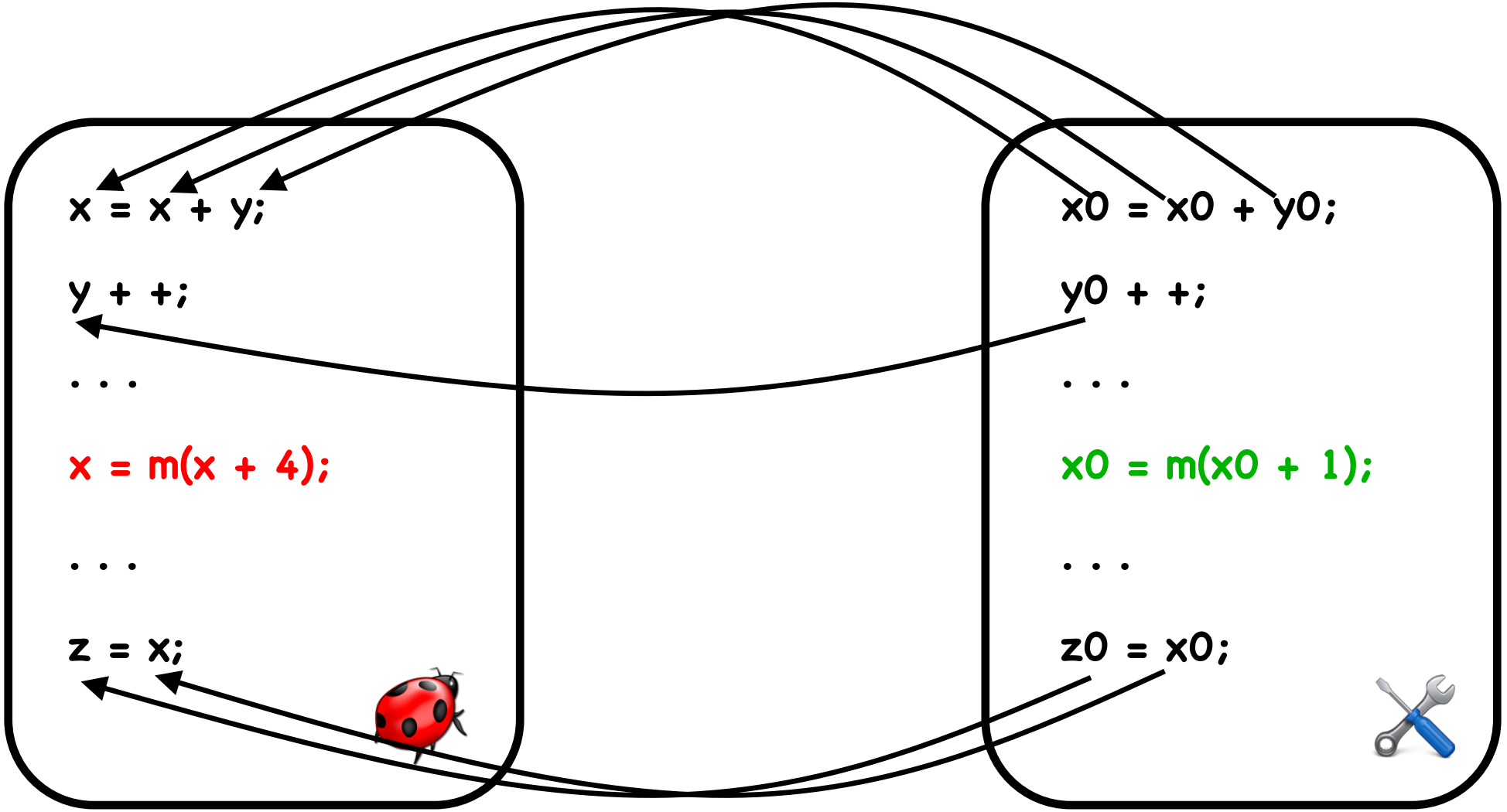
```
x = x + y;
y + +;
. . .
x = m(x + 4);
. . .
z = x;
```

Buggy Code

```
x0 = x0 + y0;
y0 + +;
. . .
x0 = m(x0 + 1);
. . .
z0 = x0;
```

Candidate Code

Code Translation



Code Translation

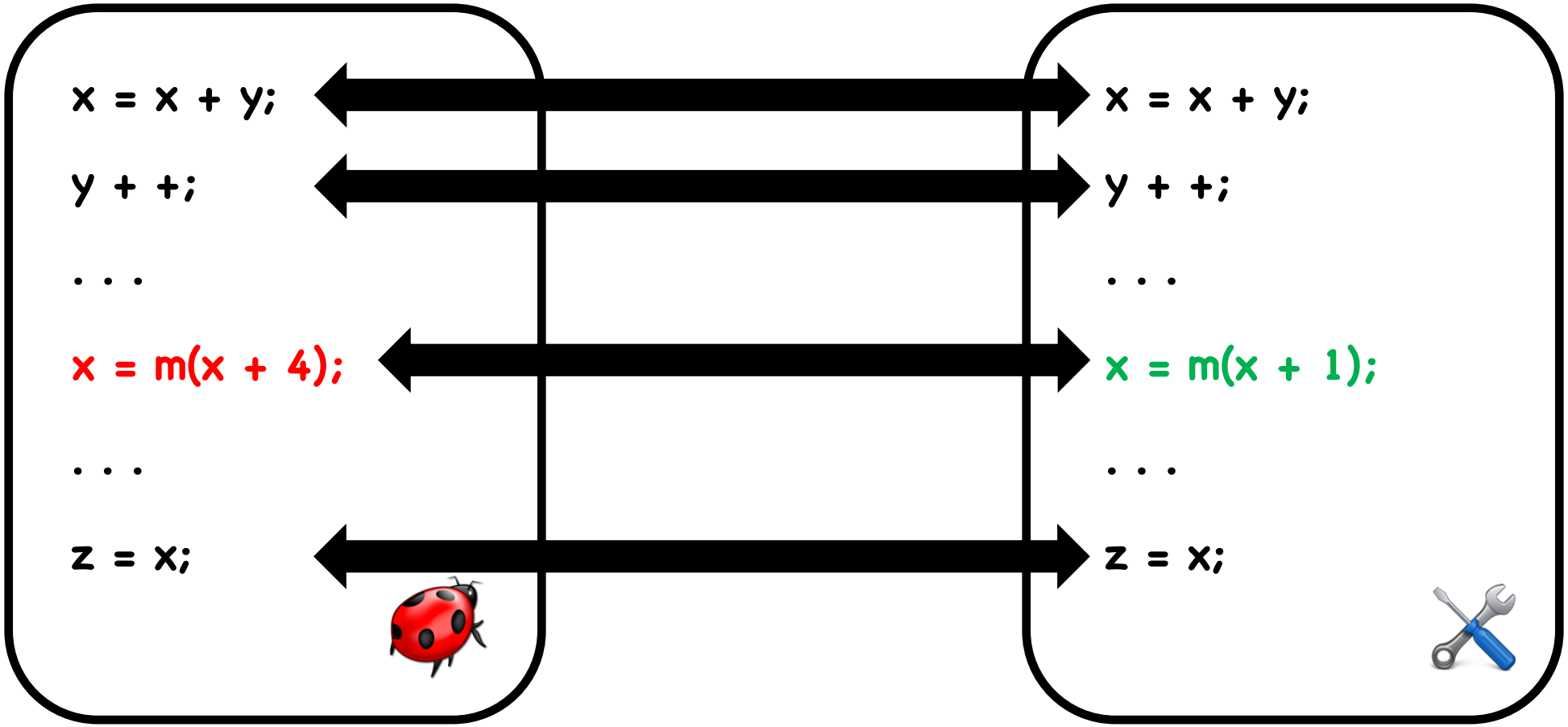
```
x = x + y;
y + +;
. . .
x = m(x + 4);
. . .
z = x;
```

Buggy Code

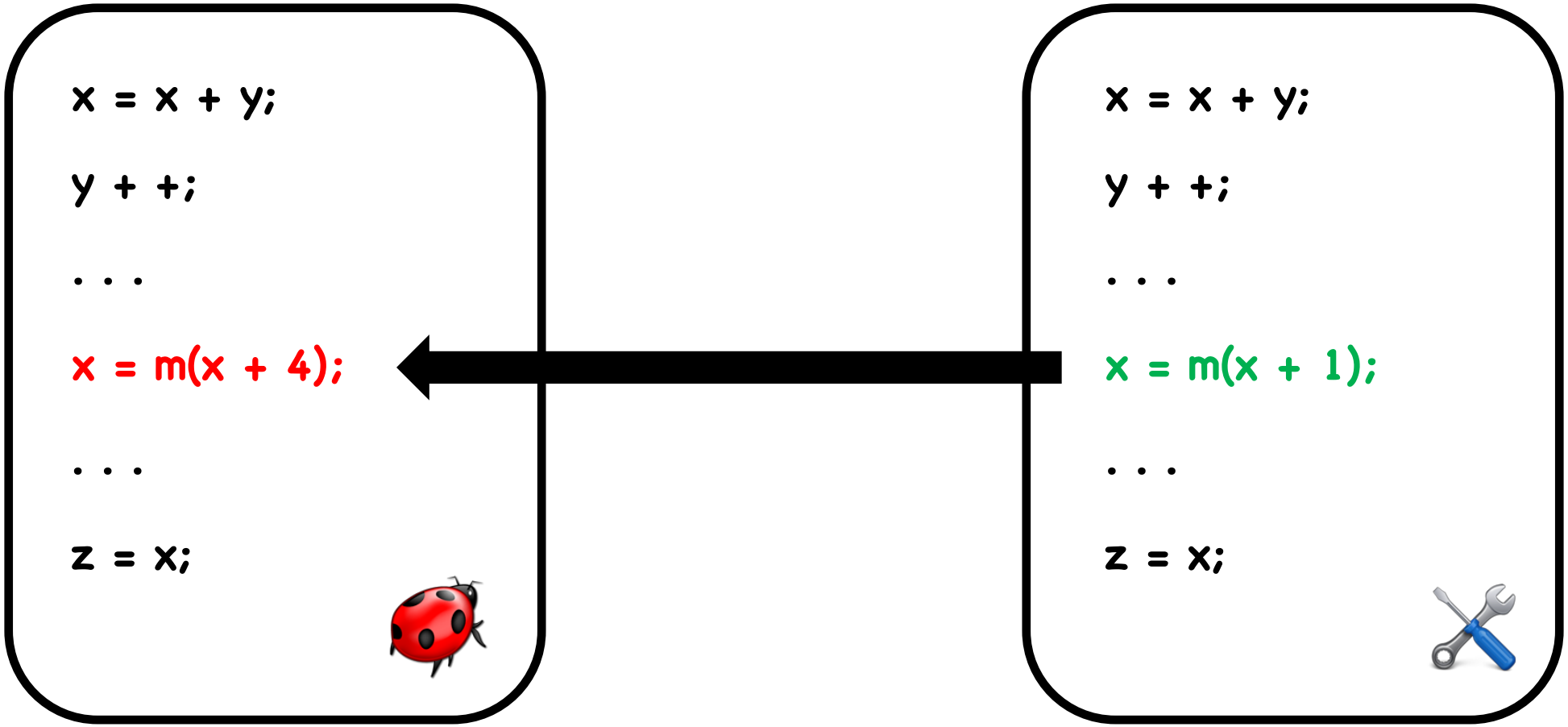
```
x = x + y;
y + +;
. . .
x = m(x + 1);
. . .
z = x;
```

Candidate Code

Code Matching



Modification



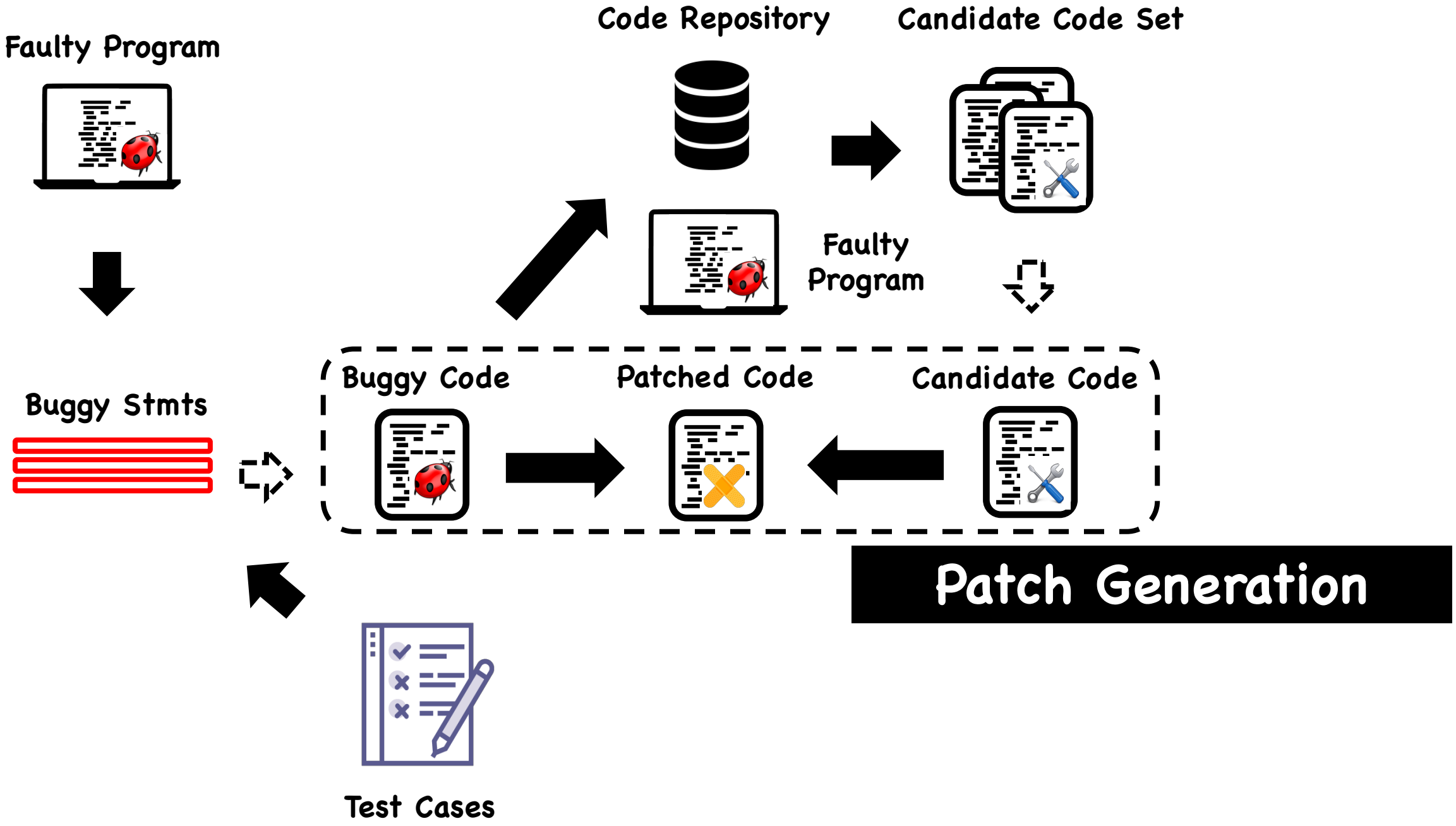
Modification

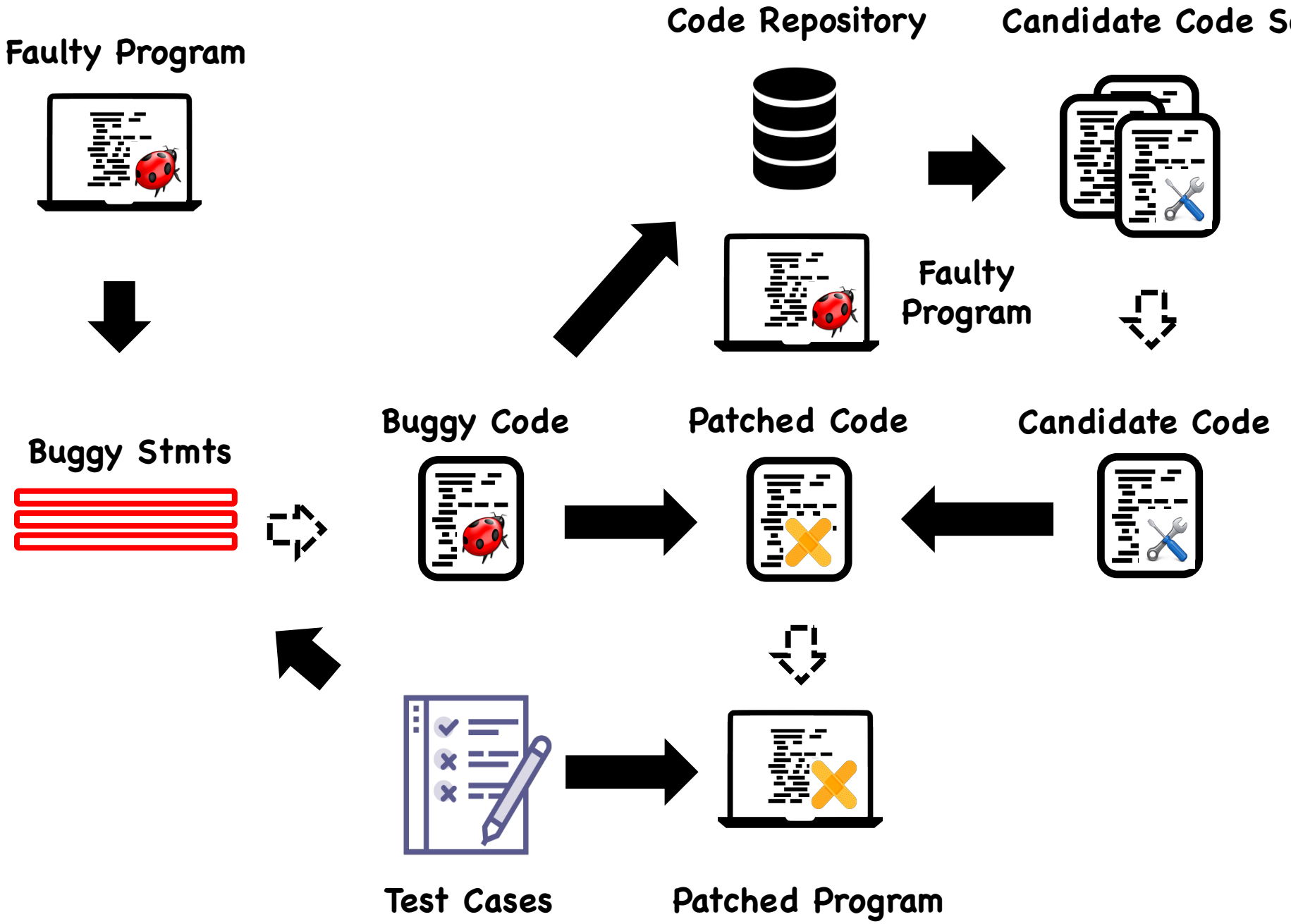
```
x = x + y;
y + +;
. . .
x = m(x + 1);
. . .
z = x;
```

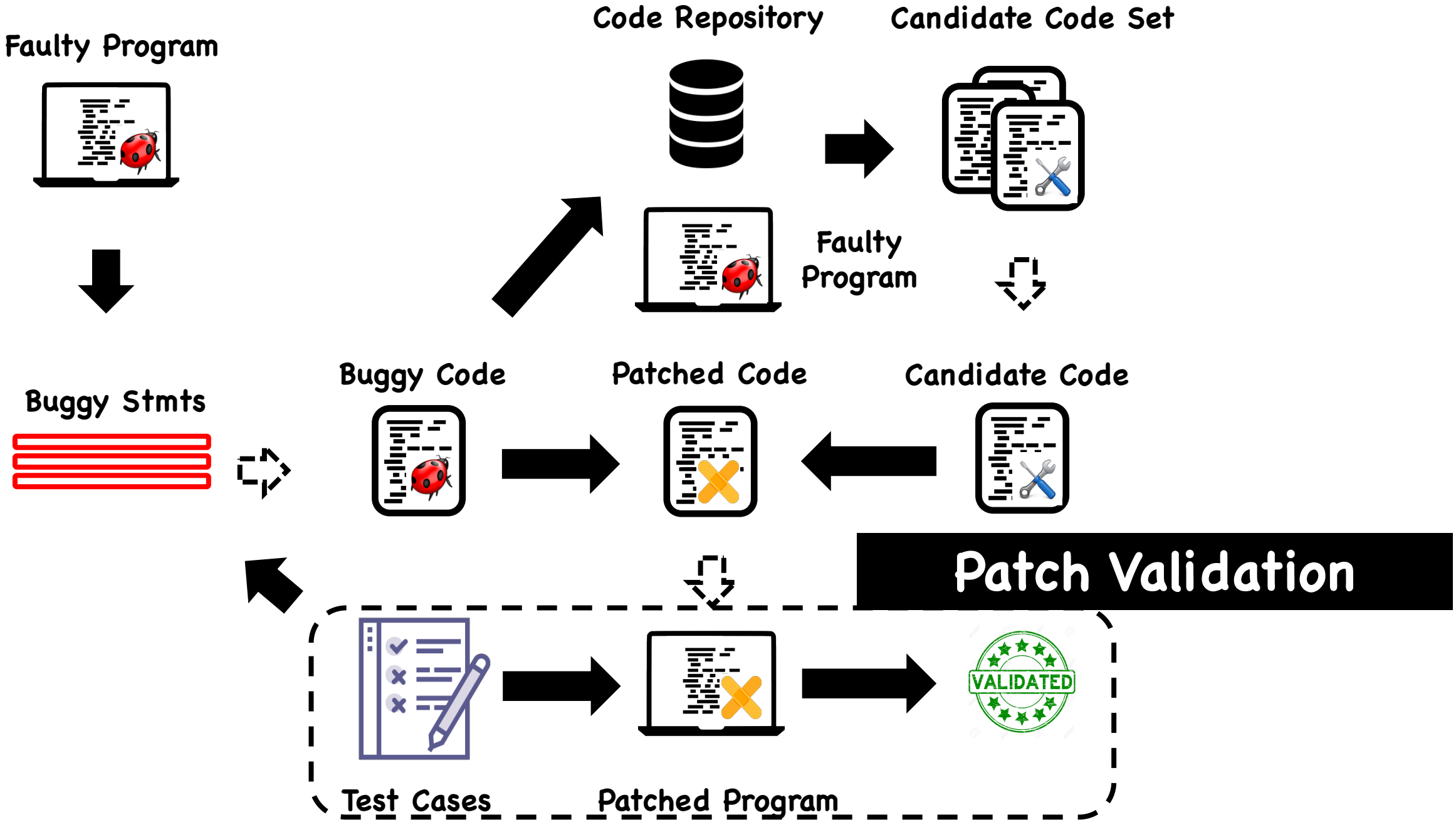
Buggy Code

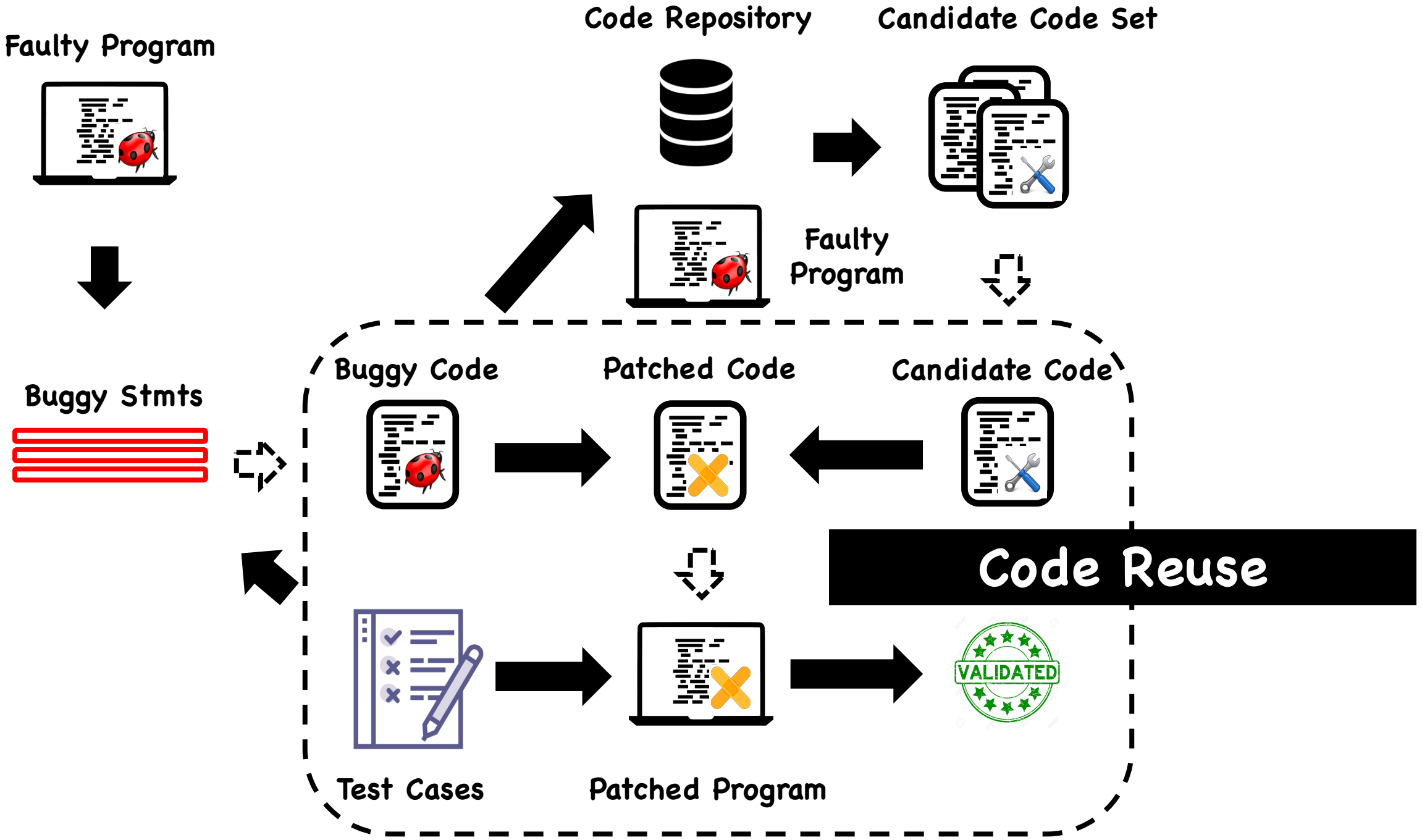
```
x = x + y;
y + +;
. . .
x = m(x + 1);
. . .
z = x;
```

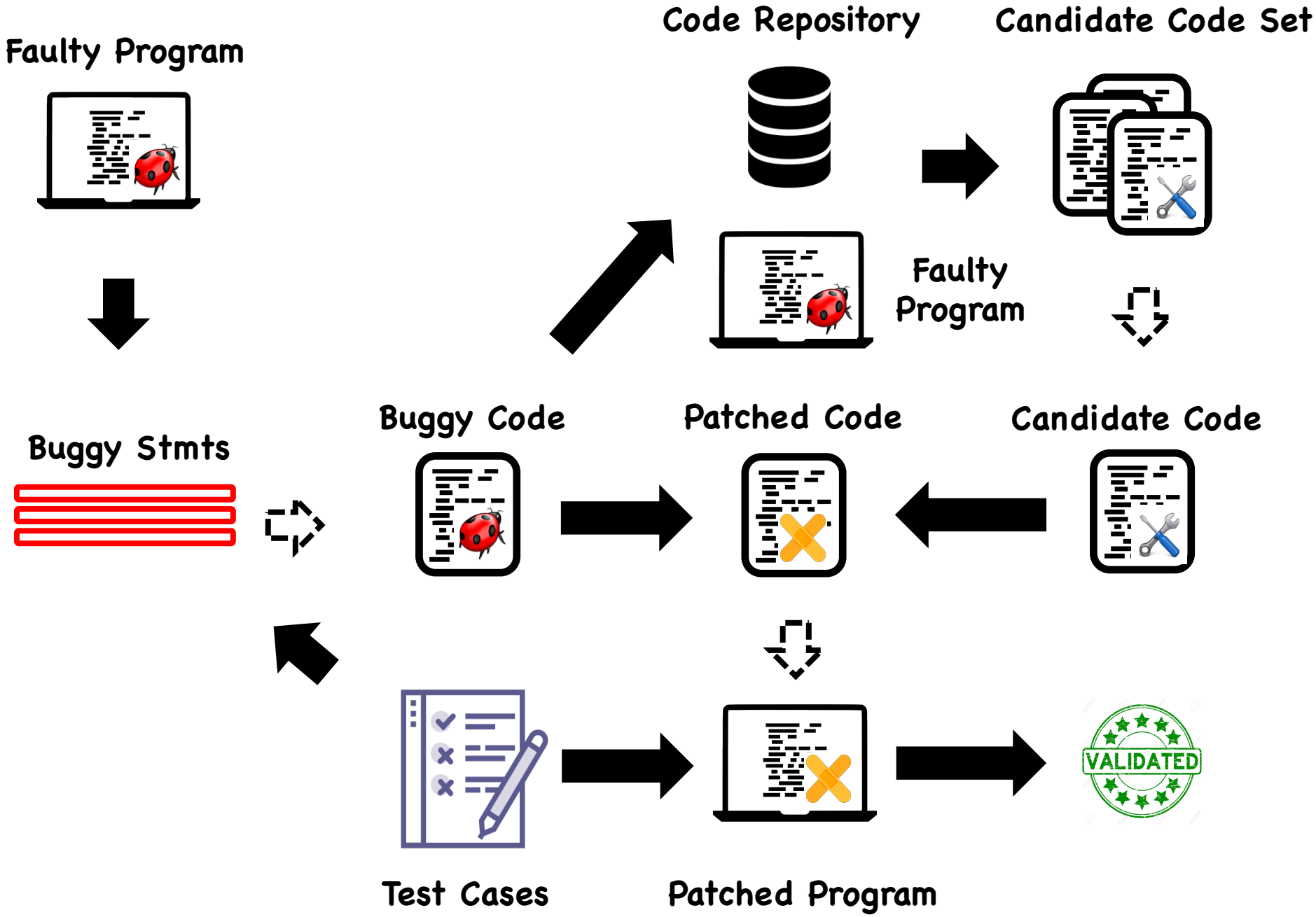
Candidate Code


Faulty Program
Code Repository
Candidate Code Set
Faulty Program
Buggy Stmts
Buggy Code
Patched Code
Candidate Code
Patch Generation
Test Cases


Faulty Program
Code Repository
Candidate Code Set
Faulty Program
Buggy Stmts
Buggy Code
Patched Code
Candidate Code
Test Cases
Patched Program


Faulty Program
Code Repository
Candidate Code Set
Faulty Program
Buggy Stmts
Buggy Code
Patched Code
Candidate Code
Patch Validation
VALIDATED
Test Cases
Patched Program


Faulty Program
Code Repository
Candidate Code Set
Faulty Program
Buggy Stmts
Buggy Code
Patched Code
Candidate Code
Code Reuse
VALIDATED
Test Cases
Patched Program


Faulty Program
Code Repository
Candidate Code Set
Faulty Program
Buggy Stmts
Buggy Code
Patched Code
Candidate Code
VALIDATED
Test Cases
Patched Program

ssFix's Component Analysis

Fault Localization

Code Search

Code Reuse

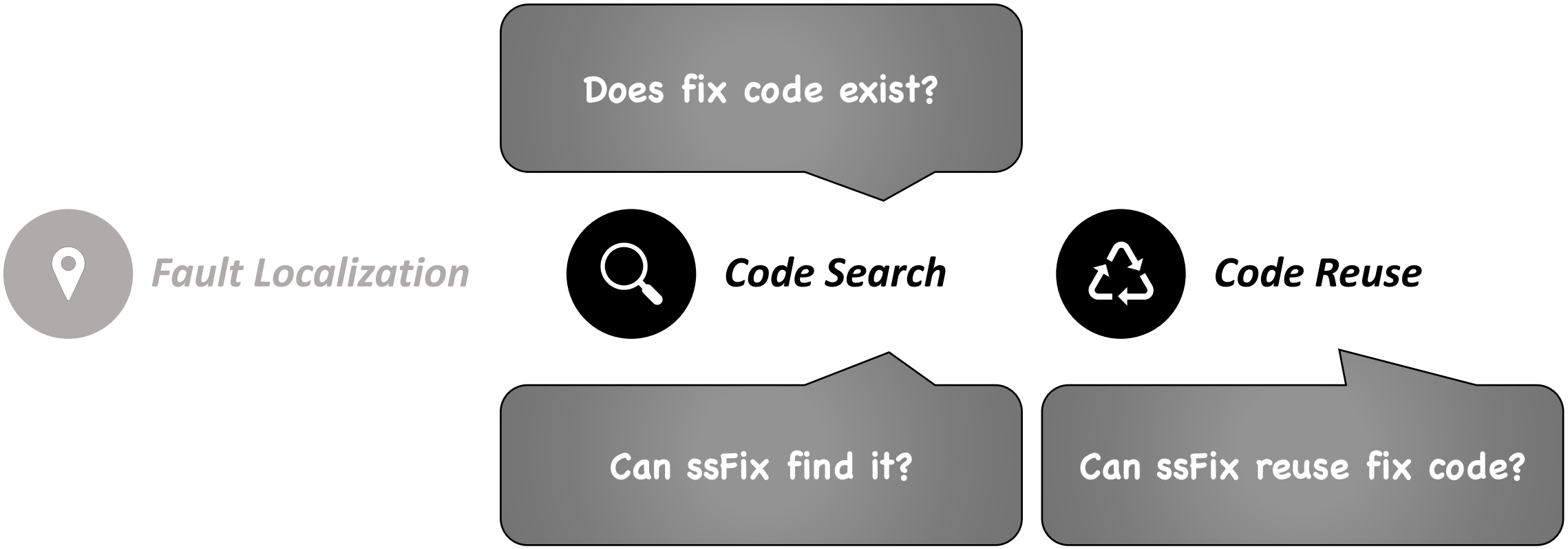
ssFix's Component Analysis

Fault Localization

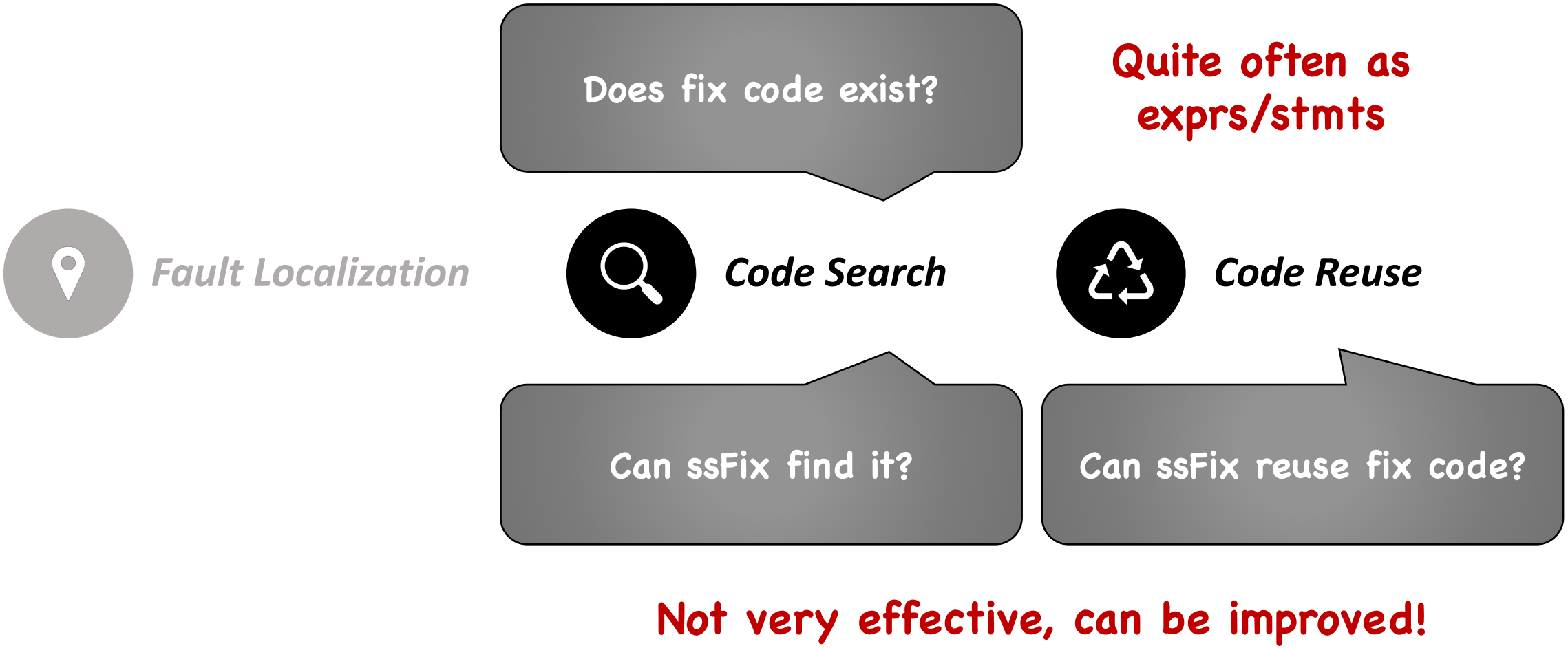
Code Search

Code Reuse

ssFix's Component Analysis



ssFix's Component Analysis

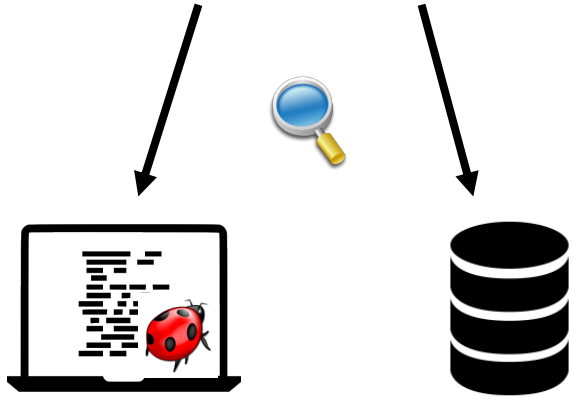


ssFix's weakness

- **Code search**
  - Same query & searching method for local & global search

ssFix's weakness

- **Code search**
  - Same query & searching method for local & global search



To Improve …

- **Code search**
  - Same query & searching method for local & global search

Using different queries & searching methods

ssFix's weakness

- **Code reuse**
  - **Code Translation:** Identifier comparison based on usage context only
  - **Code Matching:** Tree-based algorithm with complex rules and arbitrary thresholds
  - **Modification:** Prone to producing defective patches & too many patches
  - **Patch Validation:** Expensive

To Improve …

- **Code reuse**
  - **Code Translation:** Identifier comparison based on usage context only
  - **Code Matching:** Tree-based algorithm with complex rules and arbitrary thresholds
  - **Modification:** Prone to proc
  - **Patch Validation:** Expensive

Leveraging more other info: original names, locations, & extracted tokens

To Improve …

- **Code reuse**
  - **Code Translation:** Identifier comparison based on usage context only
  - **Code Matching:** Tree-based algorithm with complex rules and arbitrary thresholds
  - **Modification:** Prone to producing def… …tches & too many patches
  - **Patch Validation:** Expensive…

Token matching with greatly simplified rules and no thresholds

To Improve …

- **Code reuse**
  - **Code Translation:** Identifier comparison based on usage context only
  - **Code Matching:** Tree-based algorithm with complex rules and arbitrary thresholds
  - **Modification:** Prone to producing defective patches & too many patches
  - **Patch Validation:** Expensive

Using a different set of operations with better support

To Improve …

- **Code reuse**
  - **Code Translation:** Identifier comparison based on usage context only
  - **Code Matching:** Tree-based algorithm with complex rules and arbitrary thresholds
  - **Modification:** Prone to producing defective patches & too many patches
  - **Patch Validation:** Expensive

Static analysis

sharpFix

Code-search-based repair technique

Follows ssFix’s basic idea

ssFix’s approach for fault localization

Different approaches for

code search & reuse

Defects4J M69 Bug

```
public RealMatrix getCorrelationPValues() throws MathException {
        TDistribution tDistribution = new TDistributionImpl(nObs - 2);
        int nVars = correlationMatrix.getColumnDimension();
        double[][] out = new double[nVars][nVars];
        for (int i = 0; i < nVars; i++) {
            for (int j = 0; j < nVars; j++) {
                if (i == j) {
                    out[i][j] = 0d;
                } else {
                    double r = correlationMatrix.getEntry(i, j);
                    double t = Math.abs(r * Math.sqrt((nObs - 2)/(1 - r * r)));
                    out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
                }
            }
        }
        return new BlockRealMatrix(out);
    }
```

Defects4J M69 Bug

```
public RealMatrix getCorrelationPValues() throws MathException {
        TDistribution tDistribution = new TDistributionImpl(nObs - 2);
        int nVars = correlationMatrix.getColumnDimension();
        double[][] out = new double[nVars][nVars];
        for (int i = 0; i < nVars; i++) {
            for (int j = 0; j < nVars; j++) {
                if (i == j) {
                    out[i][j] = 0d;
                } else {
                    double r = correlationMatrix.getEntry(i, j);
                    double t = Math.abs(r * Math.sqrt((nObs - 2)/(1 - r * r)));
                    out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
                }
            }
        }
    }
```

```
out[i][j] = 2 * tDistribution.cumulativeProbability(-t);
```

Fault Localization

```
out[i][j] = 2 * (1 - tDistr ion.cumulativeProbability(t));
```

sharpFix's approach

- Employs GZoltar for spectrum-based fault localization
- Also analyzes stack trace (if any)

Code Search

```
out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
```

```
return 2.0 * distribution.cumulativeProbability(-t);
degreesOfFreedom = df(v1, v2, n1, n2);
distribution.setDegreesOfFreedom(degreesOfFreedom);
```

Merge

```
SuperMethodInvocation smi = ast.newSuperMethodInvocation();
smi.setName(ast.newSimpleName(mname));
return smi_es;
```

```
public RealMatrix getCorrelationPValues() throws MathException {
        TDistribution tDistribution = new TDistributionImpl(nObs - 2);
        int nVars = correlationMatrix.getColumnDimension();
        double[][] out = new double[nVars][nVars];
        for (int i = 0; i < nVars; i++) {
            for (int j = 0; j < nVars; j++) {
                if (i == j) {
                    out[i][j] = 0d;
                } else {
                    double r = correlationMatrix.getEntry(i, j);
                    double t = Math.abs(r * Math.sqrt((nObs - 2)/(1 - r * r)));
                    out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
                }
            }
        }
        return new BlockRealMatrix(out);
    }
```

```
private static ExpressionStatement getSuperES(AST ast, String mname, String pname) {
    SuperMethodInvocation smi = ast.newSuperMethodInvocation();
    smi.setName(ast.newSimpleName(mname));
    if (pname != null) {
        smi.arguments().add(ast.newSimpleName(pname));
    }
    ExpressionStatement smi_es = ast.newExpressionStatement(smi);
    return smi_es;
}
```

Code Search

```
out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
```

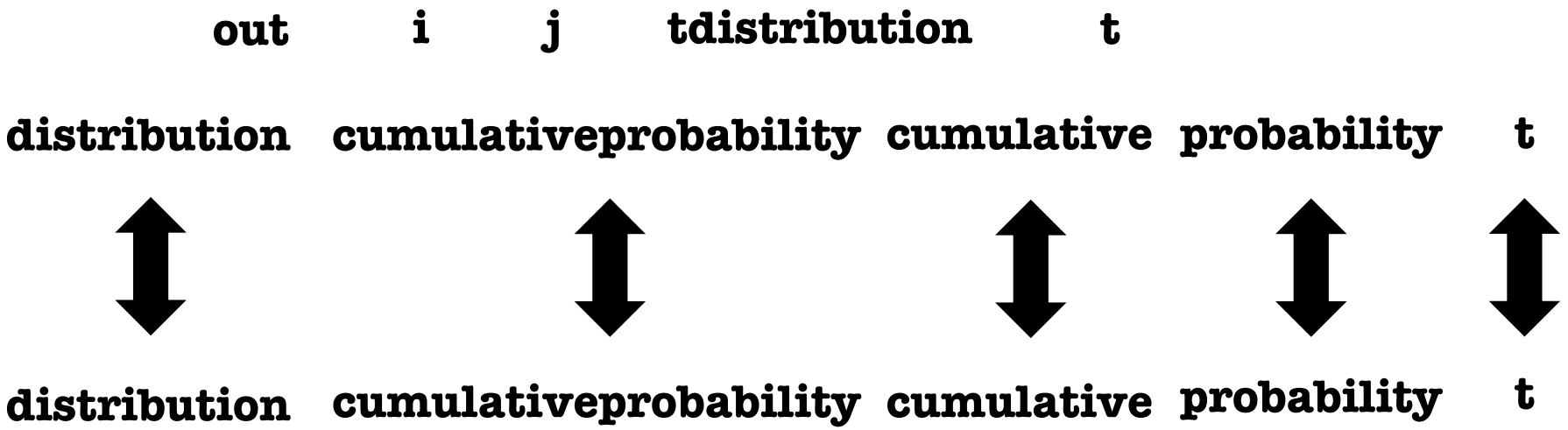
```
return 2.0 * distribution.cumulativeProbability(-t);
degreesOfFreedom = df(v1, v2, n1, n2);
distribution.setDegreesOfFreedom(degreesOfFreedom);
```

Local Search

- Search the local (faulty) program
- Bug stmt as query
- Compare with local program stmts

Statement Comparison

```
out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
```



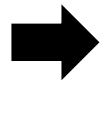

```
return 2.0 * distribution.cumulativeProbability(-t);
```

Code Search

Global Search

- Search a code repository
- Bug method as query
- Compare with repository methods
- Identify similar stmts within a method

```
public RealMatrix getCorrelationPValues() throws MathException {
        TDistribution tDistribution = new TDistributionImpl(nObs - 2);
        int nVars = correlationMatrix.getColumnDimension();
        double[][] out = new double[nVars][nVars];
        for (int i = 0; i < nVars; i++) {
            for (int j = 0; j < nVars; j++) {
                if (i == j) {
                    out[i][j] = 0d;
                } else {
                    double r = correlationMatrix.getEntry(i, j);
                    double t = Math.abs(r * Math.sqrt((nObs - 2)/(1 - r * r)));
                    out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
                }
            }
        }
        return new BlockRealMatrix(out);
    }
```



```
SuperMethodInvocation smi = ast.newSuperMethodInvocation();
smi.setName(ast.newSimpleName(mname));
return smi_es;
```

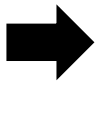
```
private static ExpressionStatement getSuperES(AST ast, String mname, String pname) {
    SuperMethodInvocation smi = ast.newSuperMethodInvocation();
    smi.setName(ast.newSimpleName(mname));
    if (pname != null) {
        smi.arguments().add(ast.newSimpleName(pname));
    }
    ExpressionStatement smi_es = ast.newExpressionStatement(smi);
    return smi_es;
}
```

Code Search

Method comparison

- Extract tokens (k-grams + words)
- Indexed tokens of repository methods
- TF-IDF vector space model

```
public RealMatrix getCorrelationPValues() throws MathException {
        TDistribution tDistribution = new TDistributionImpl(nObs - 2);
        int nVars = correlationMatrix.getColumnDimension();
        double[][] out = new double[nVars][nVars];
        for (int i = 0; i < nVars; i++) {
            for (int j = 0; j < nVars; j++) {
                if (i == j) {
                    out[i][j] = 0d;
                } else {
                    double r = correlationMatrix.getEntry(i, j);
                    double t = Math.abs(r * Math.sqrt((nObs - 2)/(1 - r * r)));
                    out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
                }
            }
        }
        return new BlockRealMatrix(out);
    }
```

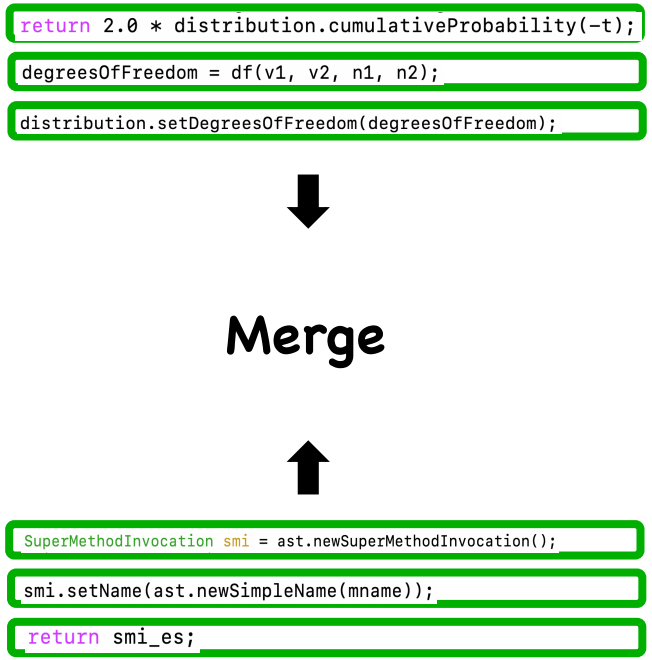


```
SuperMethodInvocation smi = ast.newSuperMethodInvocation();
smi.setName(ast.newSimpleName(mname));
return smi_es;
```

```
private static ExpressionStatement getSuperES(AST ast, String mname, String pname) {
    SuperMethodInvocation smi = ast.newSuperMethodInvocation();
    smi.setName(ast.newSimpleName(mname));
    if (pname != null) {
        smi.arguments().add(ast.newSimpleName(pname));
    }
    ExpressionStatement smi_es = ast.newExpressionStatement(smi);
    return smi_es;
}
```

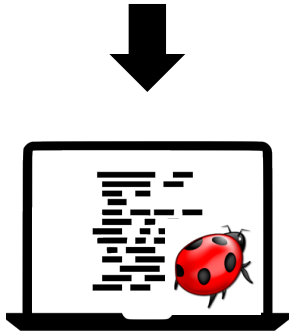
Code Search

Rank by scores (normalized)
Select the Top-k stmts (k=200)



Code Search

```
out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
```



No. 2

```
return 2.0 * distribution.cumulativeProbability(-t);
```

Code Reuse

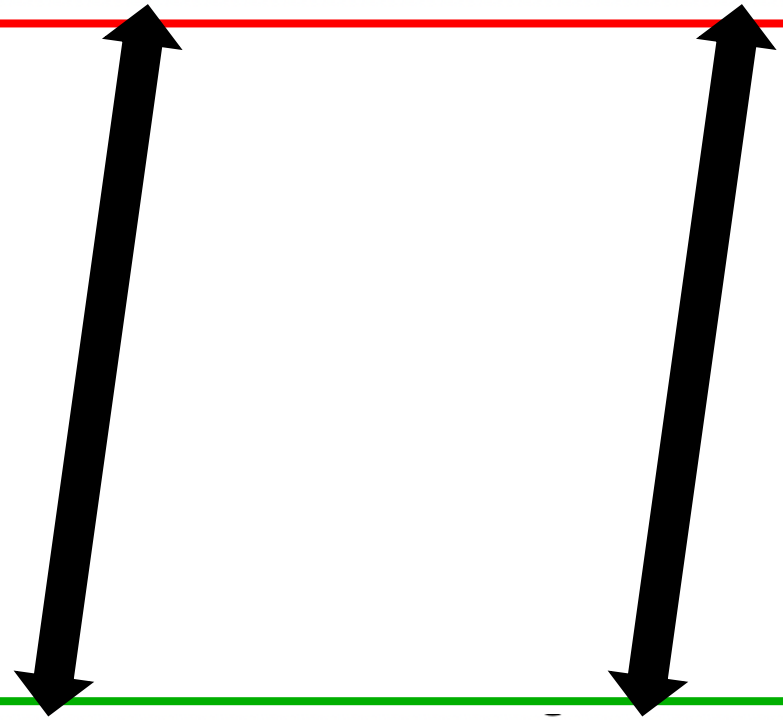
1. Code Translation
2. Code Matching
3. Modification
4. Patch Validation

Finding “Related” Identifiers

- Collect the identifiers
- Compare identifier’s **original names**
- Match **enclosing method & class names**
- Compare identifier’s **usage context**
- Compare **tokens** extracted from identifiers

Code Translation

```
out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
```



```
return 2.0 * distribution.cumulativeProbability(-t);
```

Code Translation

```
out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
```

```
return 2.0 * tDistribution.cumulativeProbability(-t);
```

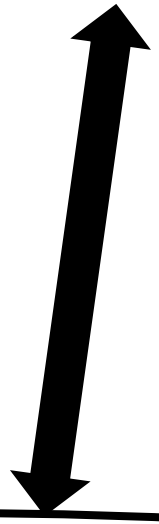
Code Matching

Match expressions and statements
Transfer code by matching result

Type checking
Token matching

Code Matching

```
out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
```



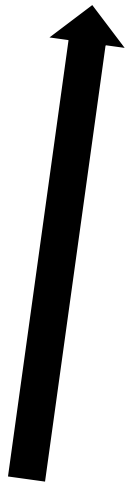
```
return 2.0 * tDistribution.cumulativeProbability(-t);
```

Modification

Expr/Stmt Replacement
Method Replacement
Stmt Insertion
Adding if-guard

Modification

```
out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
```



Replace

```
return 2.0 * tDistribution.cumulativeProbability(-t);
```

Patch Validation

Patch sorting
Static checking
Compiling
Test case running

ssFix's Failure

- Code Search

```
double r = correlationMatrix.getEntry(i, j);
double t = Math.abs(r * Math.sqrt((nObs - 2)/(1 - r * r)));
out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
```

```
degreesOfFreedom = df(v1, v2, n1, n2);
distribution.setDegreesOfFreedom(degreesOfFreedom);
return 2.0 * distribution.cumulativeProbability(-t);
```

ssFix's Failure

- Code Search
- Code Reuse

```
out[i][j] = 2 * (1 - tDistribution.cumulativeProbability(t));
```

```
return 2.0 * distribution.cumulativeProbability(-t);
```

ssFix's Failure

- Code Search
- Code Reuse

Evaluation

- **RQ1**: Is sharpFix's **code search** better than ssFix's?
- **RQ2**: Is sharpFix's **code reuse** better than ssFix's?
- **RQ3**: Can sharpFix do better **repair** than ssFix and others?

Fix Ingredient Experiment

- RQ: Does the bug-fix code exist?
- Results used as truths for code search & reuse experiments

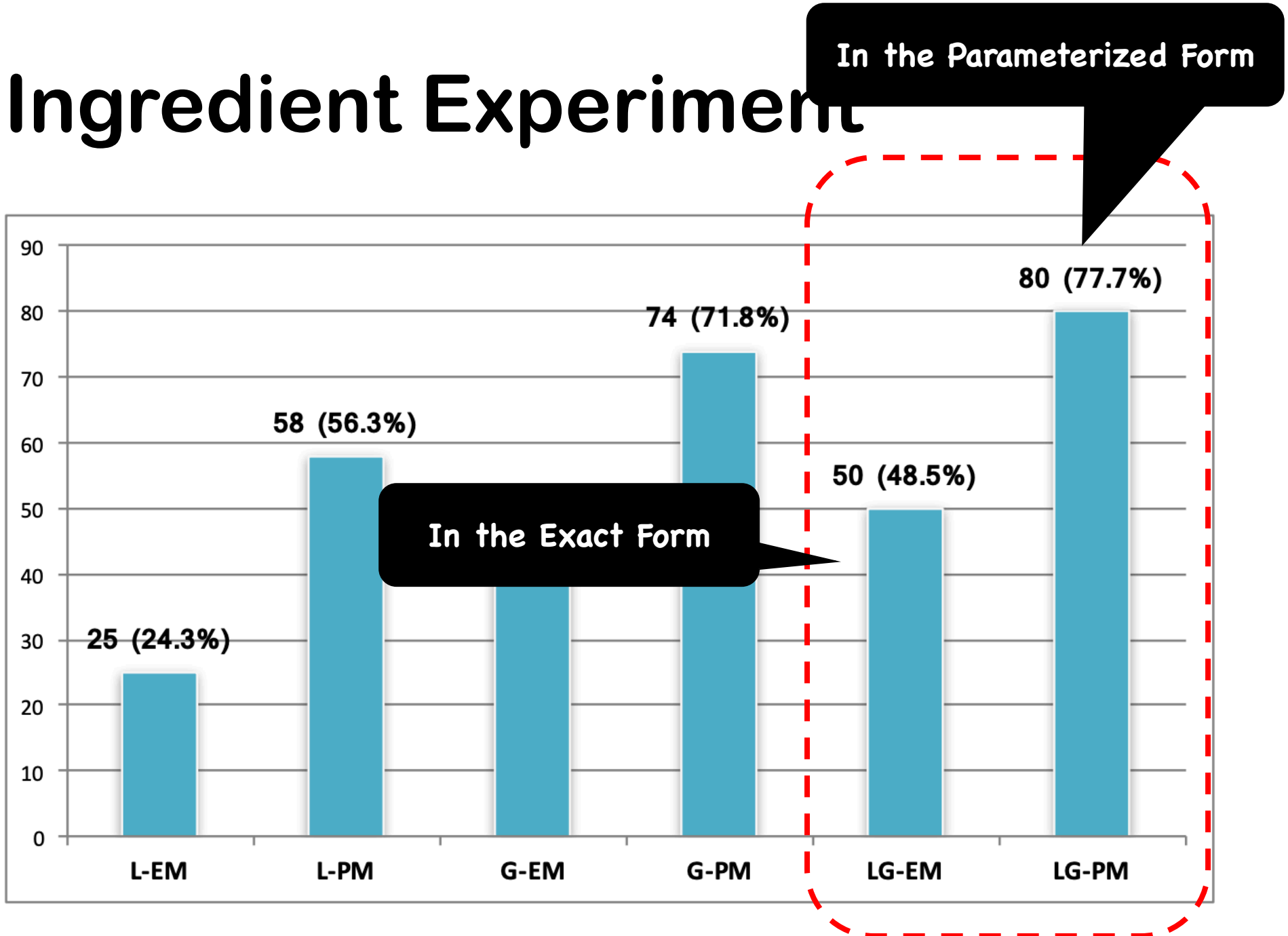
Fix Ingredient Experiment

- RQ: Does the bug-fix c
- Results used as truths
- se experiments

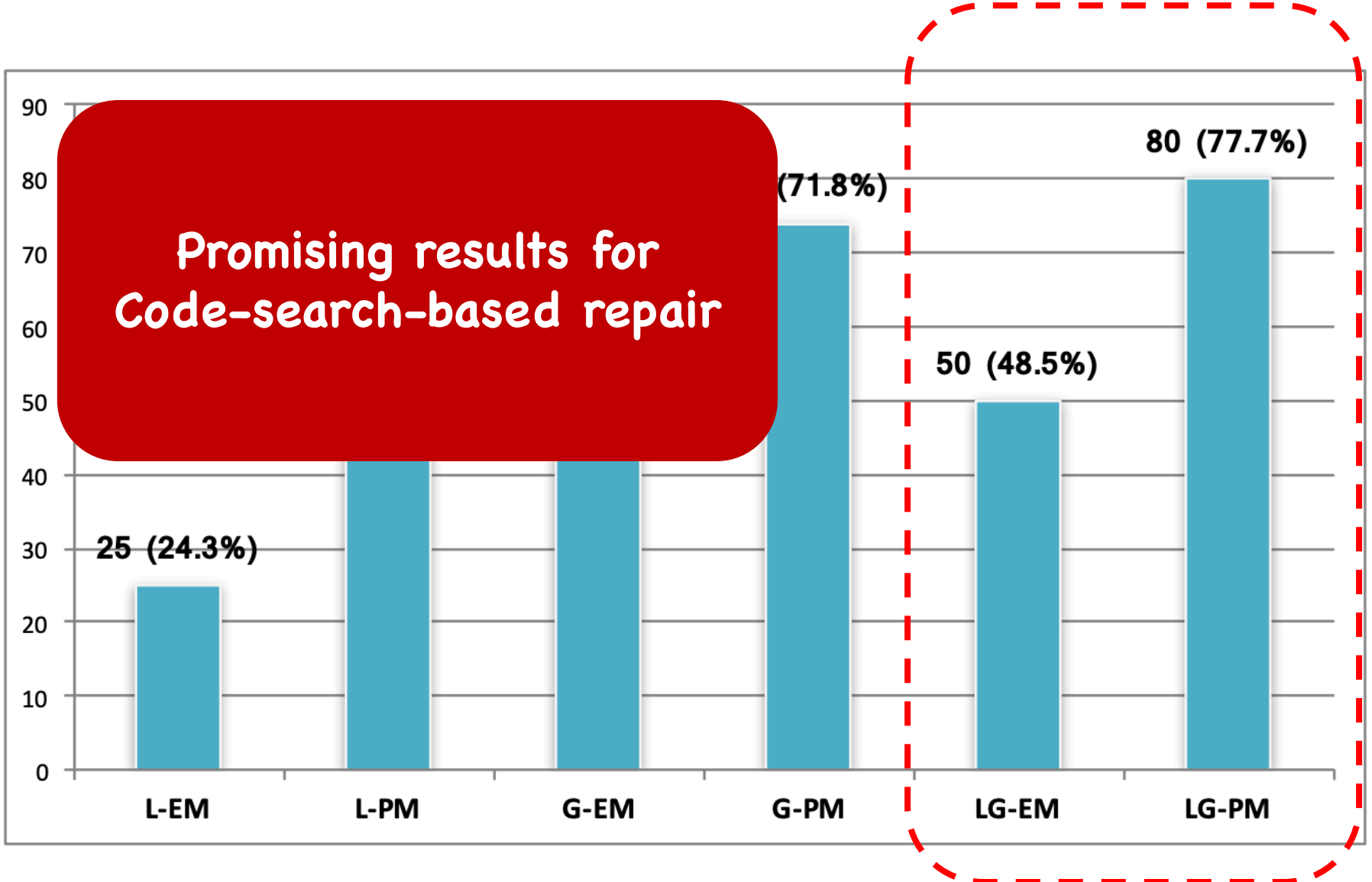
do {…} while (fa * fb >= 0.0)
do {…} while (**fa * fb > 0.0**)

- Defined 6 types of fix ingredients (as exprs & stmts)
- Identified 103 Defects4J bugs that are “simple”
- Identified fix ingredients
- Search fix ingredients in local program & repository (~81G)

Fix Ingredient Experiment



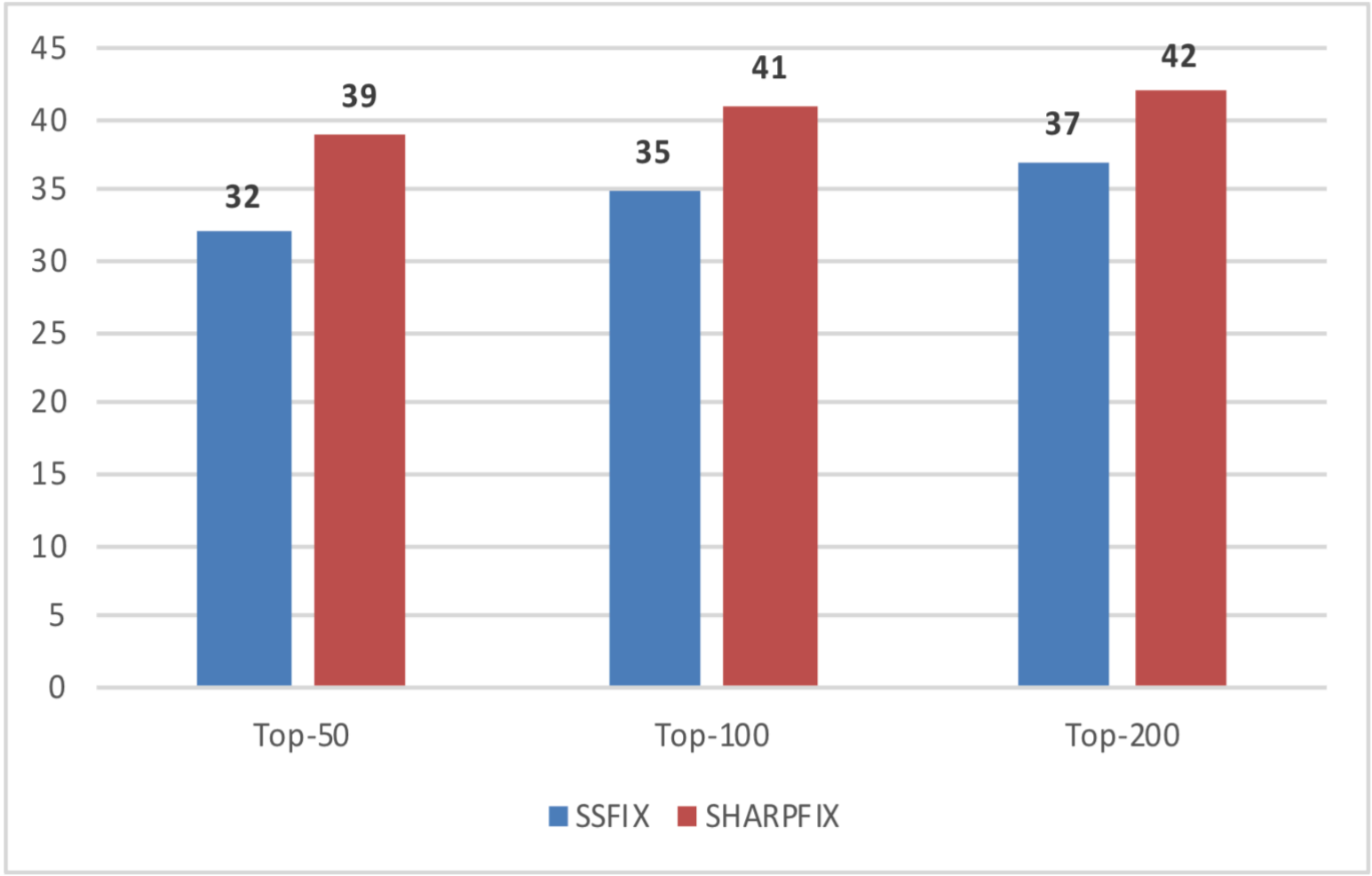
Fix Ingredient Experiment



Code Search Comparison

- Based on 103 Defects4J bugs
- Manually provided ssFix & sharpFix with the faulty stmt
- Ran code search
- Analyzed the results

Code Search Comparison



Code Reuse Comparison

- Based on code-search-succeeded bugs
- Provided ssFix & sharpFix with the bug & fix code
- Ran code reuse
- Evaluated the correctness of the plausible patch generated

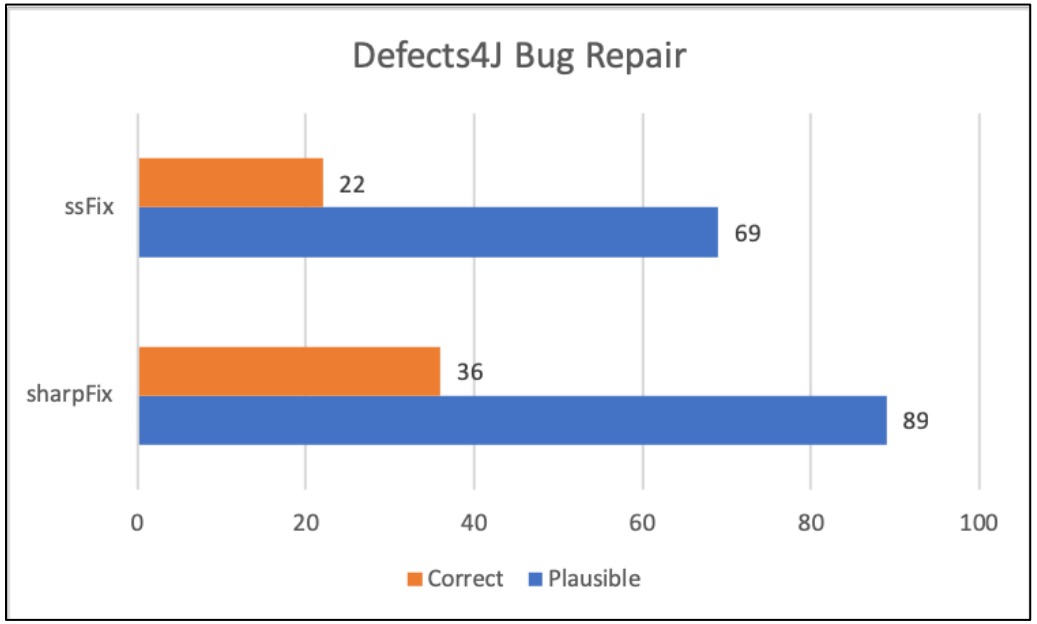
Code Reuse Comparison

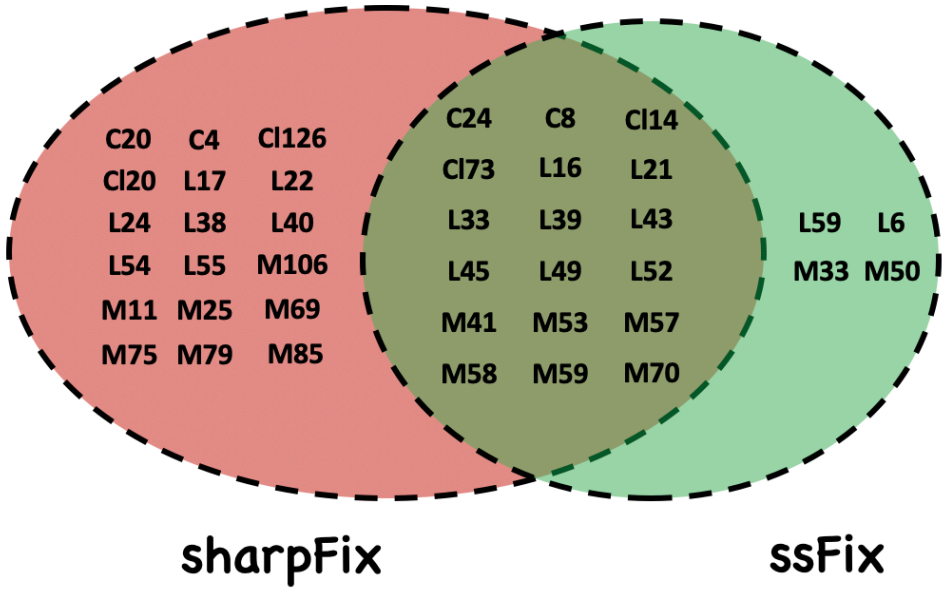
sharpFix reused 50.8% fix code
ssFix reused 40.4% fix code

Repair

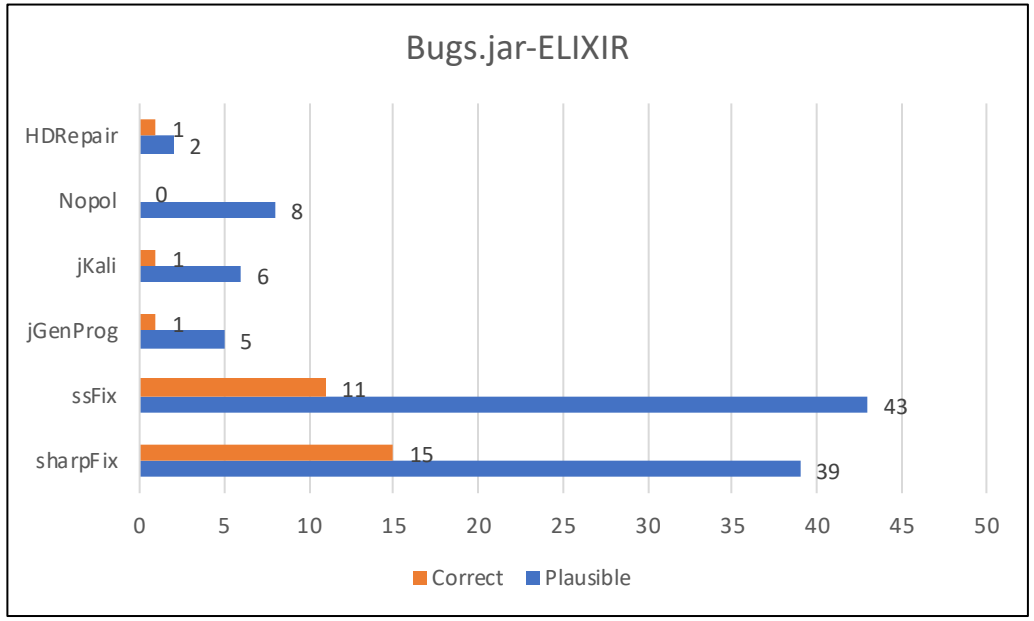
- Automatic bug repair
- Two datasets: Defects4J (357 bugs) & Bugs.jar-ELIXIR (127 bugs)
- ssFix & sharpFix on Defects4J
- ssFix, sharpFix, and four others on Bugs.jar-ELIXIR

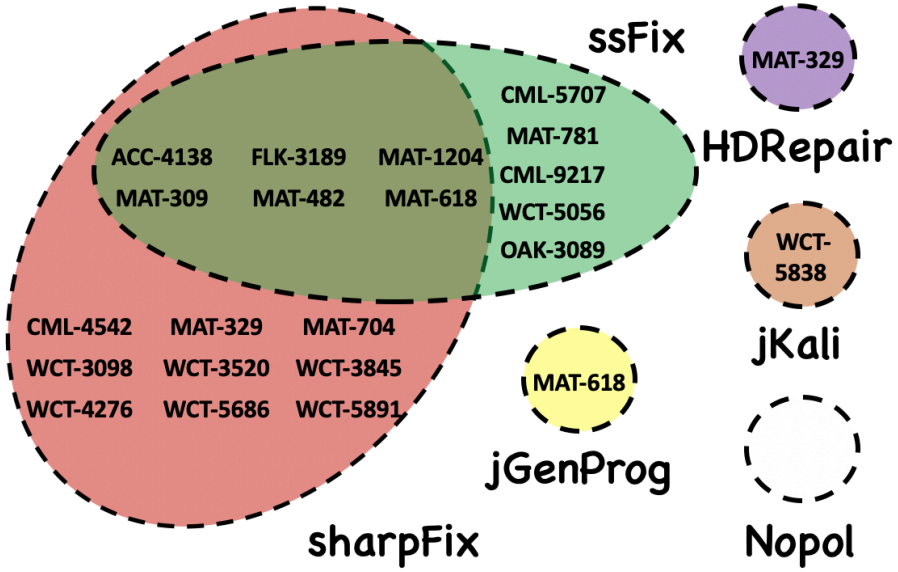
Defects4J


Defects4J Bug Repair
ssFix
22
69
sharpFix
36
89
0
20
40
60
80
100
Correct
Plausible


C20 C4 Cl126
Cl20 L17 L22
L24 L38 L40
L54 L55 M106
M11 M25 M69
M75 M79 M85
C24 C8 Cl14
Cl73 L16 L21
L33 L39 L43
L45 L49 L52
M41 M53 M57
M58 M59 M70
L59 L6
M33 M50
sharpFix
ssFix

Bugs.jar-ELIXIR





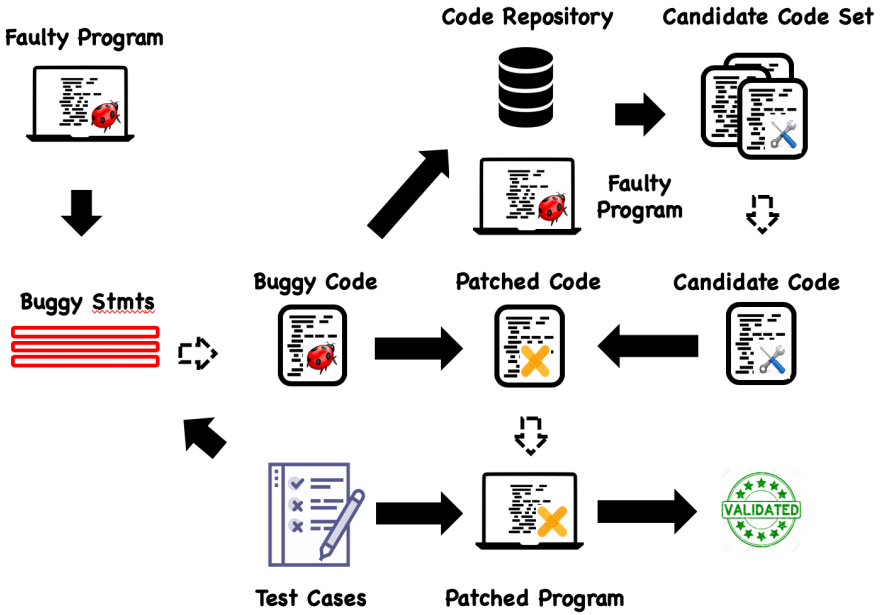
Conclusion & Future Work

- Repair by code search is promising!
- ssFix's code-search-based repair can be **improved**
- sharpFix follows ssFix's idea but improves **code search** & **reuse**
- sharpFix can do **better repair**

Conclusion & Future Work

- Repair by code search is promising!
- ssFix's code-search-based repair can be **improved**
- sharpFix follows ssFix's idea but improves **code search** & **reuse**
- sharpFix can do **better repair**

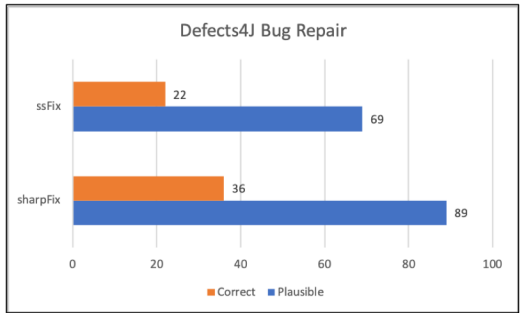
- Syntactic + semantic code search
- Patch overfitting
- Other dataset

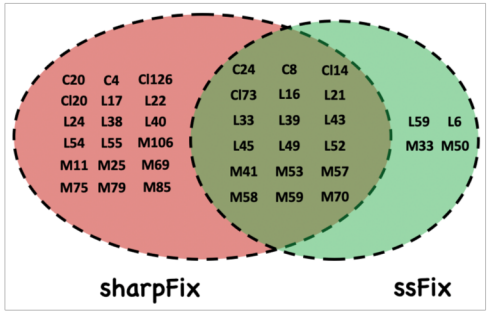


sharpFix

- Code-search-based repair technique
- Follows ssFix's basic idea
- ssFix's approach for fault localization
- Different approaches for code search & reuse

Defects4J





Bugs.jar-ELIXIR