How Many of All Bugs Do We Find? A Study of Static Bug Detectors

Andrew Habib, Michael Pradel

TU Darmstadt, Germany

software-lab.org

Static Bug Detection

Error Prone

Static Bug Detection

Error Prone

- General framework
- Scalable static analysis
- Set of checkers for specific bug patterns

How Many Bugs Do They Find?

How Many Bugs Do They Find?

Given a representative set of real-world bugs, how many of them do static bug detectors find?

How Many Bugs Do They Find?

Given a representative set of real-world bugs, how many of them do static bug detectors find?

This talk:

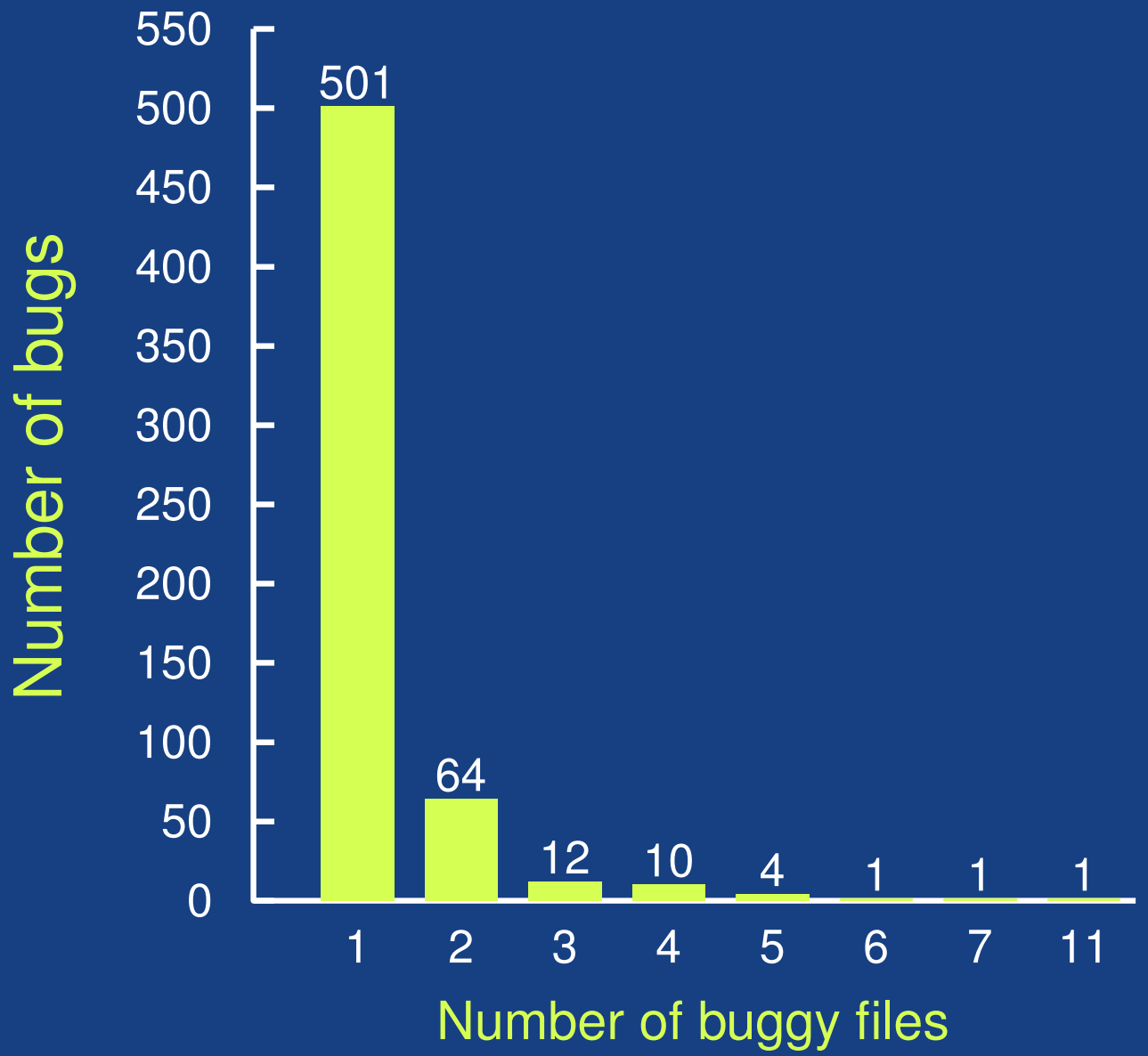
Empirical study with 594 real-world Java bugs and 3 popular static checkers

Real-World Bugs

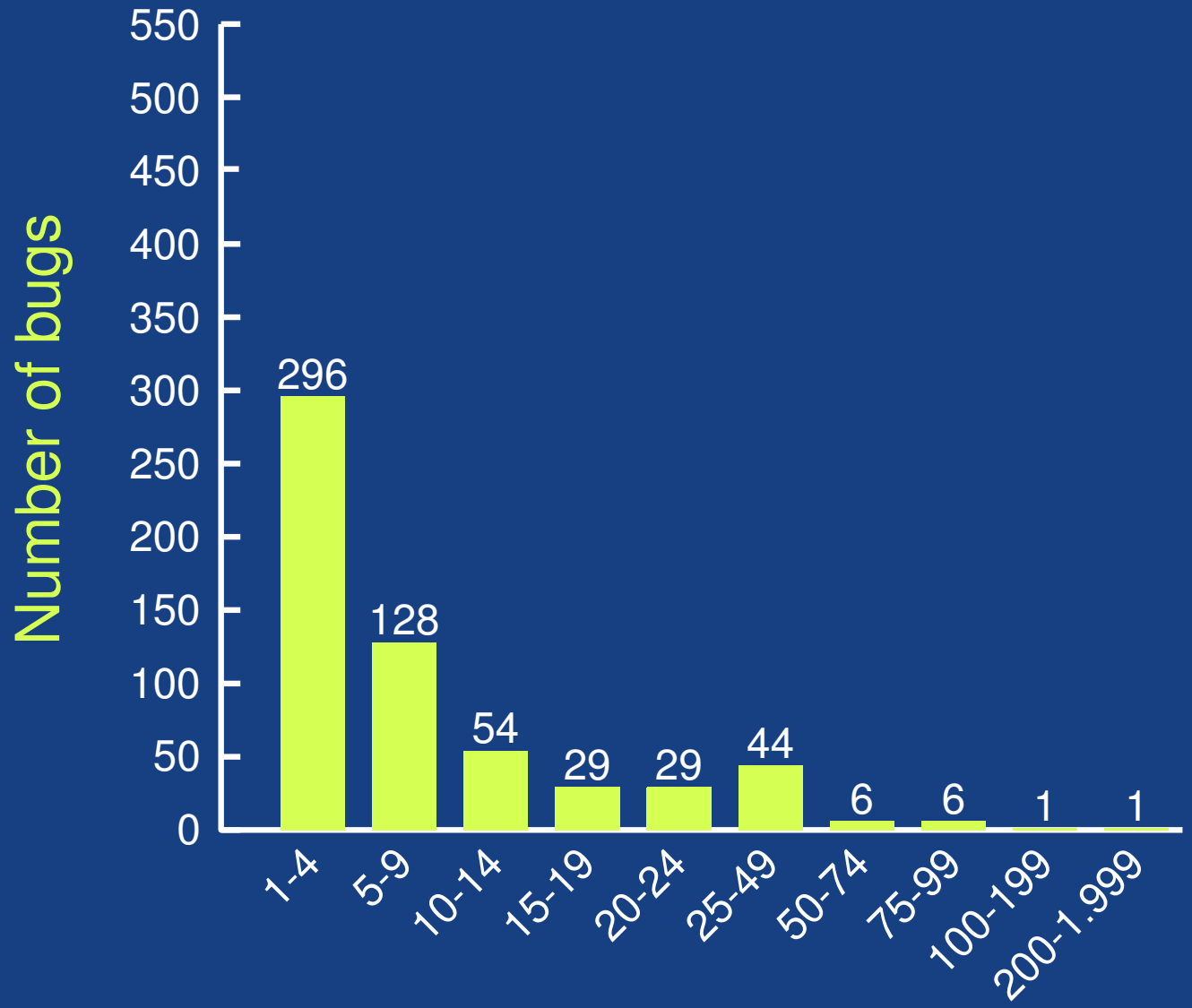
- **594 bugs from 15 popular Java projects**
  - Extended version of Defects4J data set
- **Why this set?**
  - Gathered independently
  - Used in other bug-related studies *
  - Contains real fixes by developers

* Just et al., 2014 (mutation testing); Shamshiri et al., 2015 (test generation); Pearson et al., 2017 (fault localization); Martinez et al., 2017 (program repair)

Defects4J: Files Involved in Bug



Defects4J: Size of Bug Fix



Previous Approach

How to determine which bugs are found?

[Thung et al., 2012]

- Get diff between buggy and fixed code
- Run tool on code with buggy lines
- If warning on buggy line: Bug found
- Result: 50% – 95% of all bugs found

Previous Approach

How to determine which bugs are found?

[Thung et al., 2012]

- Get diff between buggy and fixed code
- Run tool on code with buggy lines
- If warning on buggy line: Bug found
- Result: 50% – 95% of all bugs found
- **Limitation:**
  - No check that warning points to bug
  - One tool flags up to 57% of all lines

Methodology: Overview

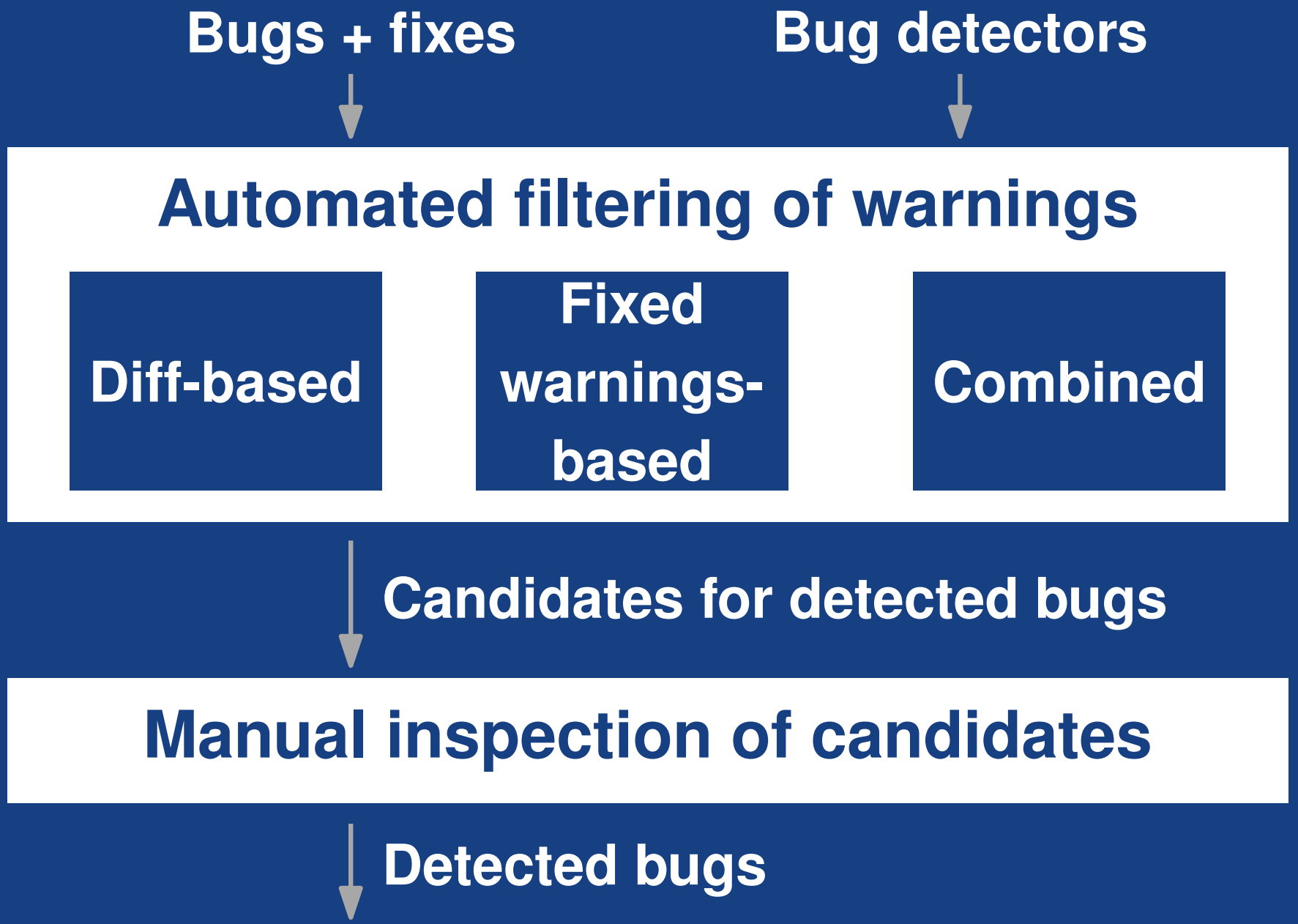
Bugs + fixes

Bug detectors

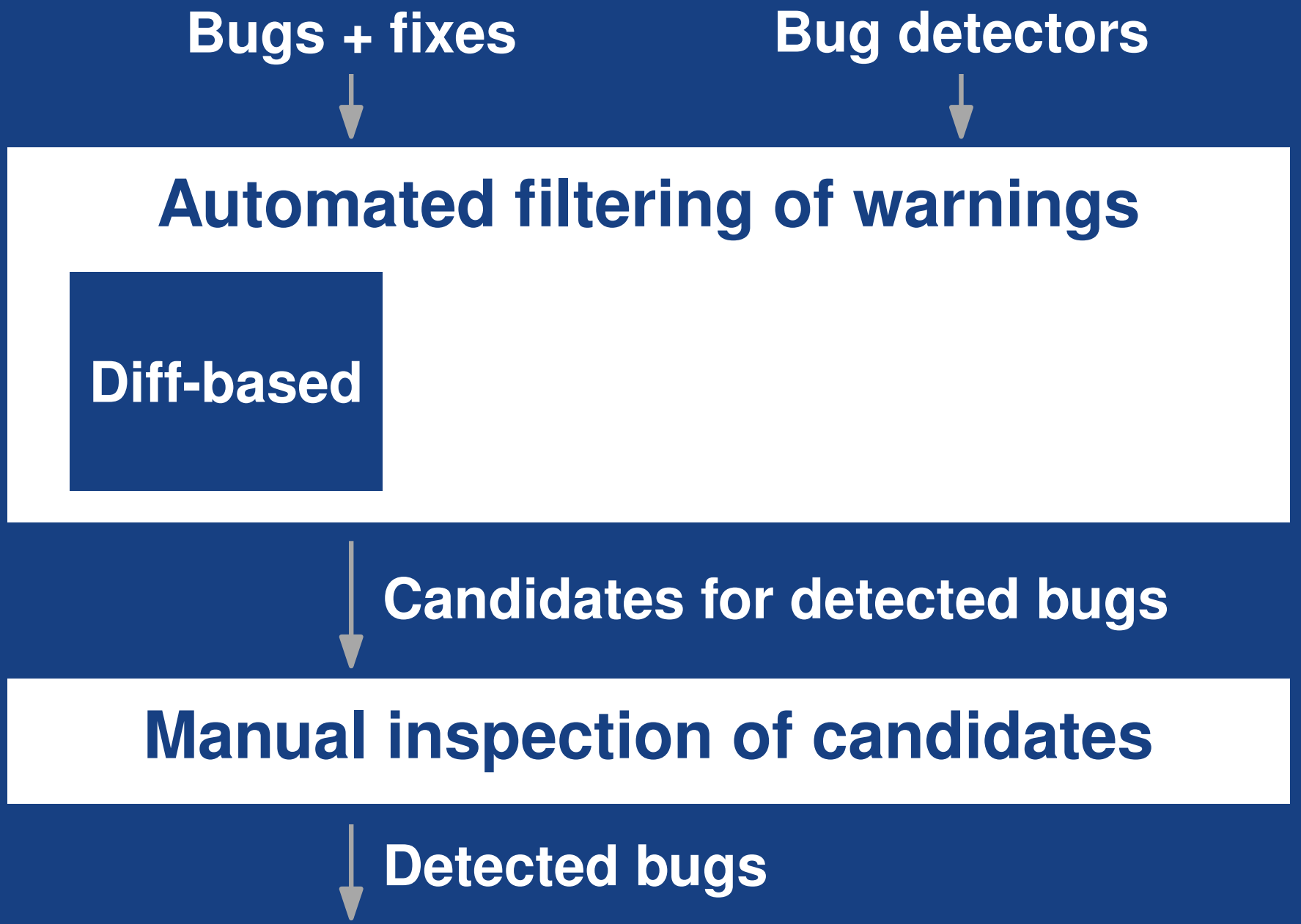
Methodology: Overview

Methodology: Overview

Methodology: Overview



Methodology: Diff-based



Methodology: Diff-based

1) Identify lines changed to fix bug

2) Intersect with lines with warning

Methodology: Diff-based

1) Identify lines changed to fix bug

2) Intersect with lines with warning

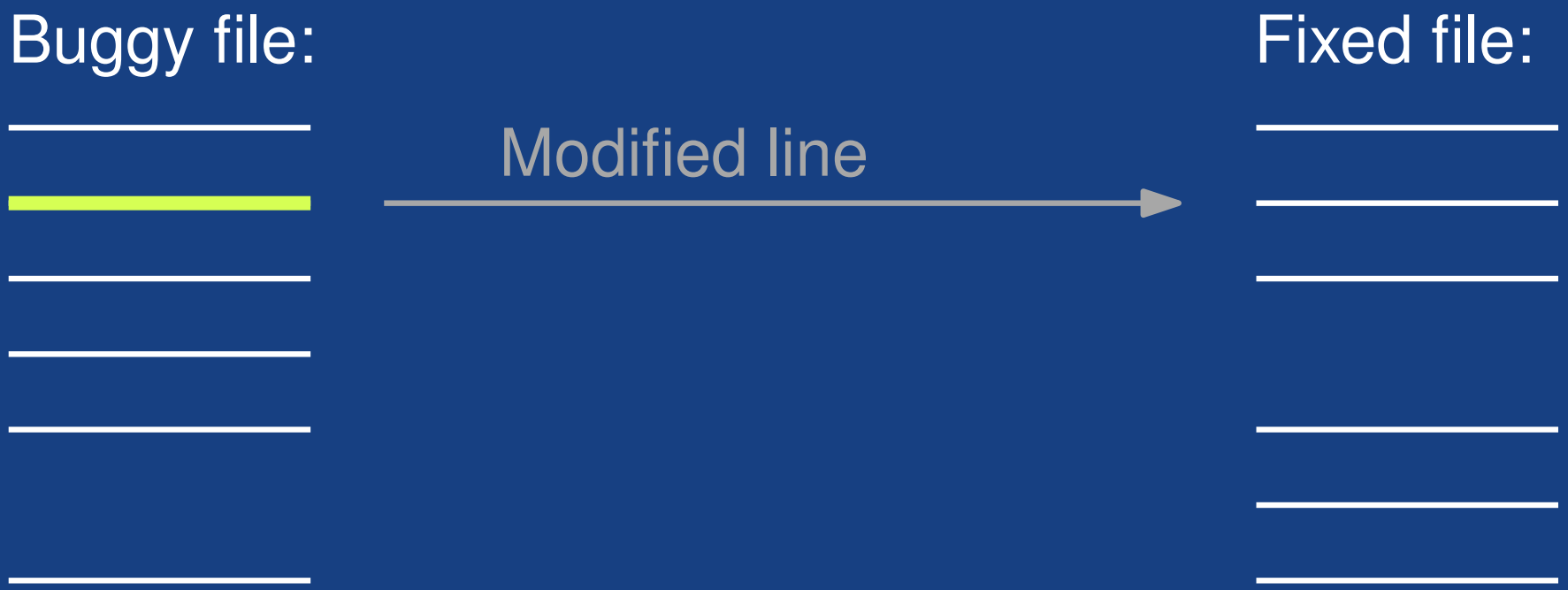
Buggy file:

Fixed file:

Methodology: Diff-based

1) Identify lines changed to fix bug

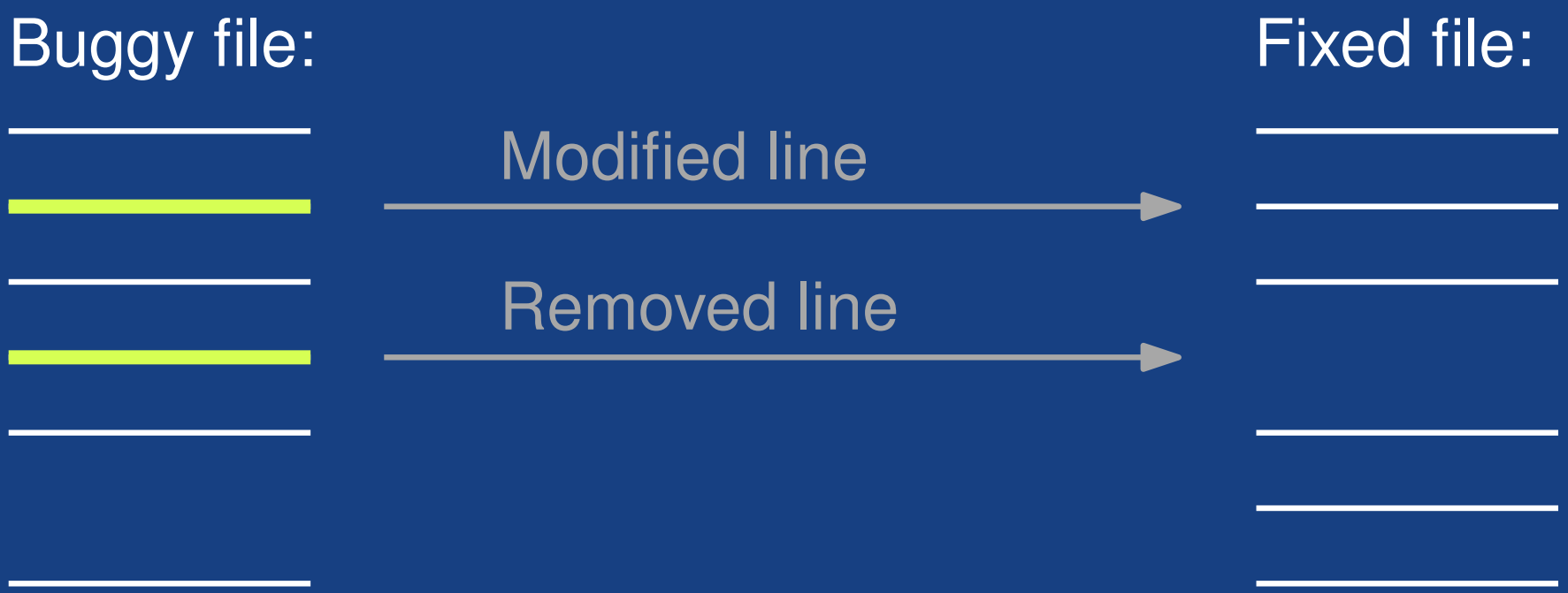
2) Intersect with lines with warning



Methodology: Diff-based

1) Identify lines changed to fix bug

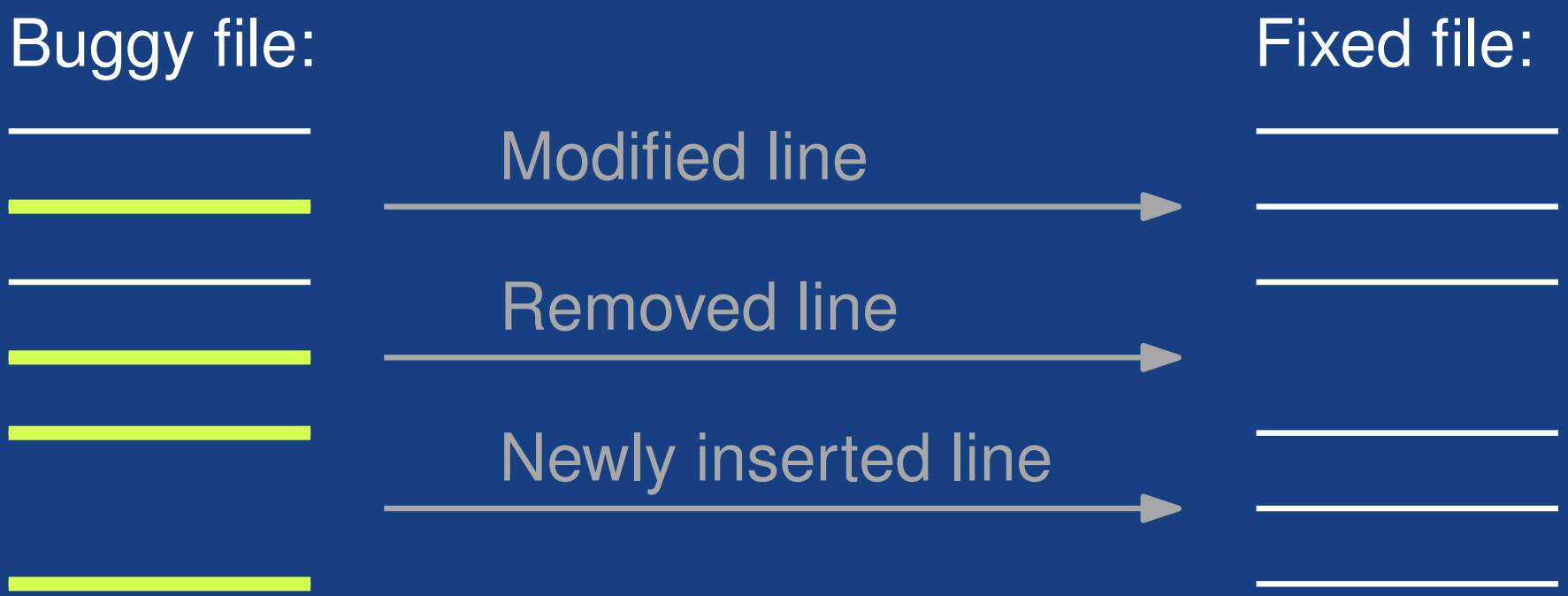
2) Intersect with lines with warning



Methodology: Diff-based

1) Identify lines changed to fix bug

2) Intersect with lines with warning



Methodology: Diff-based

1) Identify lines changed to fix bug

2) Intersect with lines with warning

Methodology: Diff-based

1) Identify lines changed to fix bug

2) Intersect with lines with warning

Candidate for detected bug

Example:

```
public Dfp multiply(final int x) {
  return multiplyFast(x);
}
```


```
public Dfp multiply(final int x) {
  if (x >= 0 && x < RADIX) {
    return multiplyFast(x);
  } else {
    return multiply(newInstance(x));
  }
}
```

Example:

```
public Dfp multiply(final int x) {
  return multiplyFast(x);
}
```

Bug fix

```
public Dfp multiply(final int x) {
  if (x >= 0 && x < RADIX) {
    return multiplyFast(x);
  } else {
    return multiply(newInstance(x));
  }
}
```

Warning:
Missing
@Override

Example:

```
public Dfp multiply(final int x) {
    return multiplyFast(x);
}
```

Bug fix

```
public Dfp multiply(final int x) {
    if (x >= 0 && x < RADIX) {
        return multiplyFast(x);
    } else {
        return multiply(newInstance(x));
    }
}
```

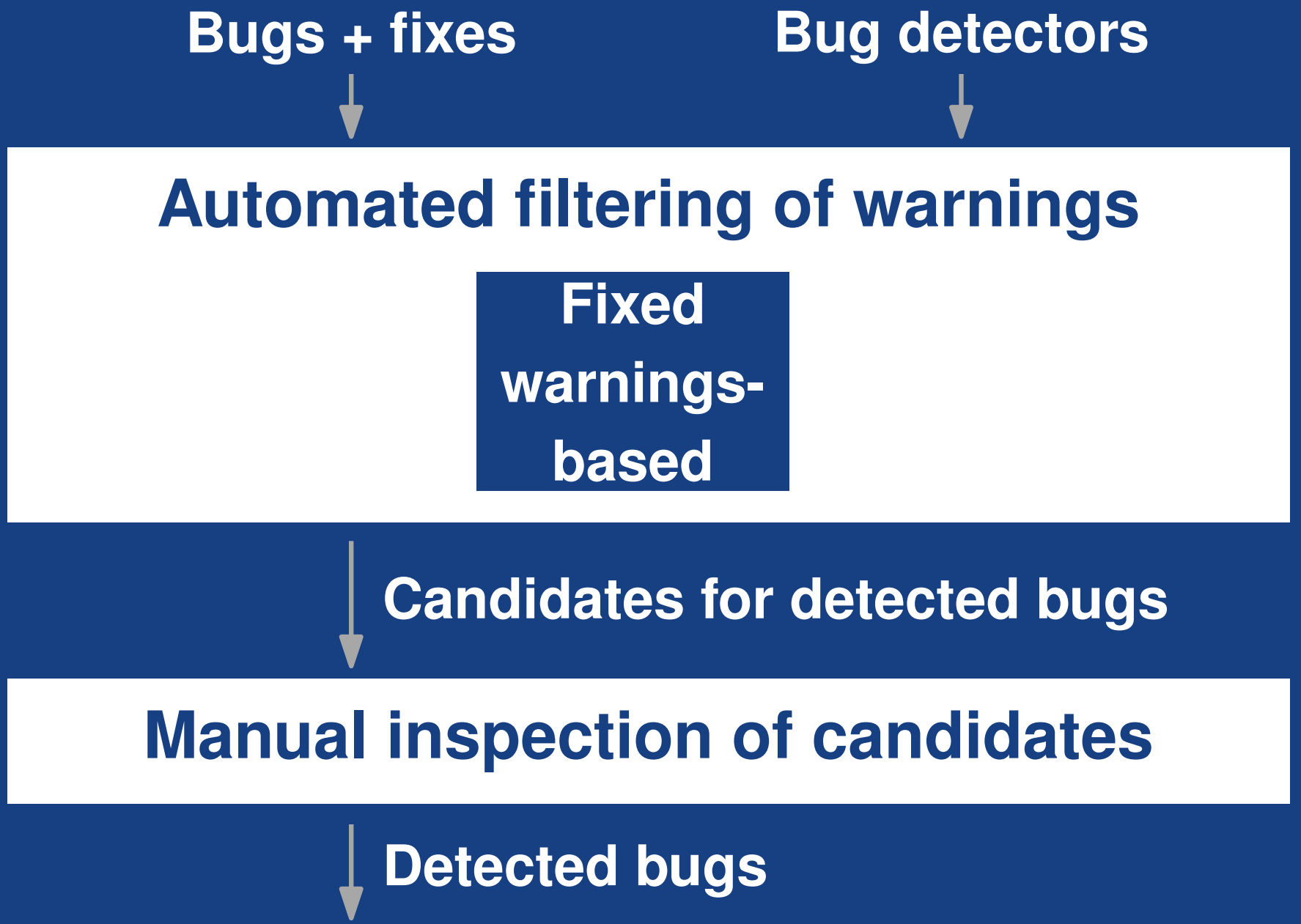
-1

+1

Warning:
Missing
@Override

Candidate for detected bug

Method.: Fixed Warnings-based



Method.: Fixed Warnings-based

1) Compare warnings before and after fix

2) Warning that disappears was for bug

Method.: Fixed Warnings-based

1) Compare warnings before and after fix

2) Warning that disappears was for bug

Buggy file:

Fixed file:

Method.: Fixed Warnings-based

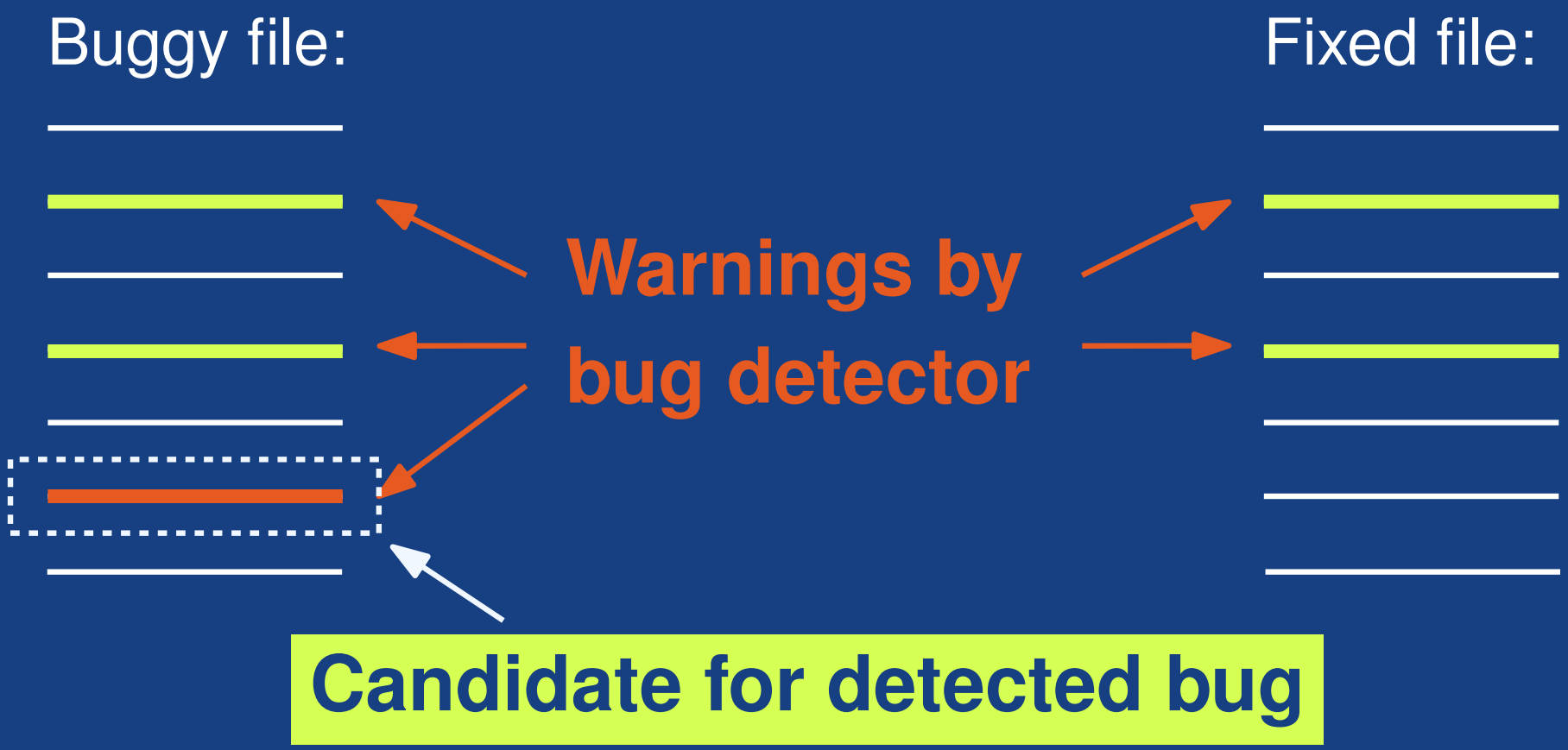
1) Compare warnings before and after fix

2) Warning that disappears was for bug

Method.: Fixed Warnings-based

1) Compare warnings before and after fix

2) Warning that disappears was for bug



Example

```
public Week(Date time, TimeZone zone) {
   this(time,
      RegularTimePeriod.DEFAULT_TIME_ZONE,
      Locale.getDefault());
}
```

Bug fix

```
public Week(Date time, TimeZone zone) {
   this(time,
      zone,
      Locale.getDefault());
}
```

Example

```
public Week(Date time, TimeZone zone) {
   this(time,
      RegularTimePeriod.DEFAULT_TIME_ZONE,
      Locale.getDefault());
}
```

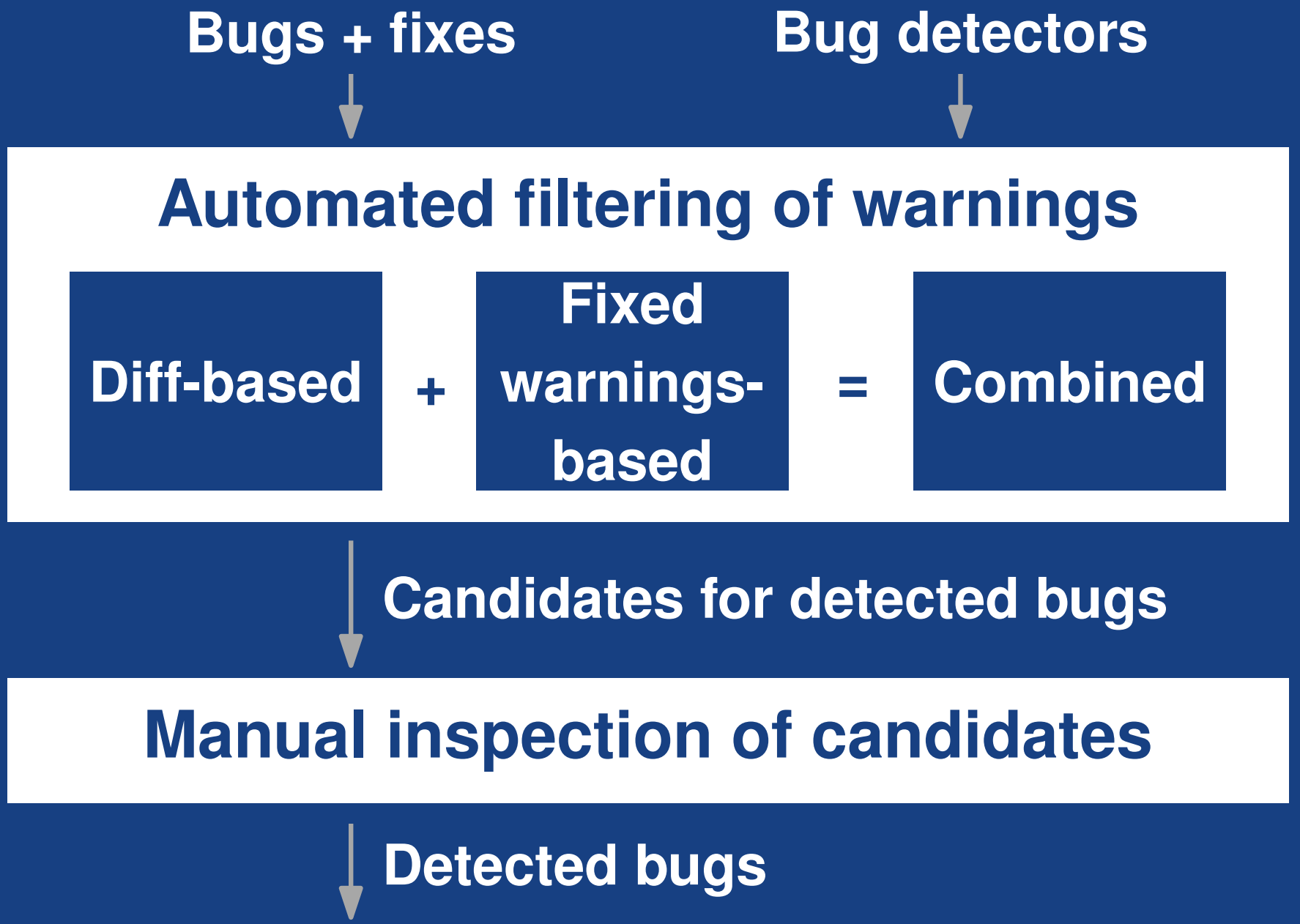
Bug fix

```
public Week(Date time, TimeZone zone) {
   this(time,
      zone,
      Locale.getDefault());
}
```

Warning: Chaining constructor ignores argument

Candidate for detected bug

Methodology: Combined



Results

Warnings to Inspect

	All warnings			
	Per bug			
Tool	Min	Max	Avg	Total
Error Prone	0	148	7.58	4,402
Infer	0	36	0.33	198
SpotBugs	0	47	1.1	647
Total				**5,247**

Warnings to Inspect

	All warnings				
	Per bug				Candidates
Tool	Min	Max	Avg	Total	**only**
Error Prone	0	148	7.58	4,402	53
Infer	0	36	0.33	198	32
SpotBugs	0	47	1.1	647	68
Total				**5,247**	**153**

Warnings to Inspect

	All warnings				
	Per bug				Candidates
Tool	Min	Max	Avg	Total	**only**
Error Prone	0	148	7.58	4,402	53
Infer	0	36	0.33	198	32
SpotBugs	0	47	1.1	647	68
Total				**5,247**	**153**

97% of all warnings are removed by the automated filtering step

Manual Inspection

Distinguish coincidental matches from actually detected bugs

Candidate = (bug, warning)

Full match

Partial match

Mismatch

Created by Freepik

Manual Inspection: Example

```
public Dfp multiply(final int x) {
  return multiplyFast(x);
}
```

Bug fix

```
public Dfp multiply(final int x) {
  if (x >= 0 && x < RADIX) {
    return multiplyFast(x);
  } else {
    return multiply(newInstance(x));
  }
}
```

Warning:
Missing
`@Override`

Candidate for detected bug

Manual Inspection: Example

```
public Dfp multiply(final int x) {
    return multiplyFast(x);
}
```

Bug fix

```
public Dfp multiply(final int x) {
    if (x >= 0 && x < RADIX) {
        return multiplyFast(x);
    } else {
        return multiply(newInstance(x));
    }
}
```

Warning:
Missing
`@Override`

Mismatch

Manual Inspection: Example (2)

```
public Week(Date time, TimeZone zone) {
   this(time,
      RegularTimePeriod.DEFAULT_TIME_ZONE,
      Locale.getDefault());
}
```

Bug fix

```
public Week(Date time, TimeZone zone) {
   this(time,
      zone,
      Locale.getDefault());
}
```

Warning: Chaining constructor ignores argument

Candidate for detected bug

Manual Inspection: Example (2)

```
public Week(Date time, TimeZone zone) {
   this(time,
      RegularTimePeriod.DEFAULT_TIME_ZONE,
      Locale.getDefault());
}
```

Bug fix

```
public Week(Date time, TimeZone zone) {
   this(time,
      zone,
      Locale.getDefault());
}
```

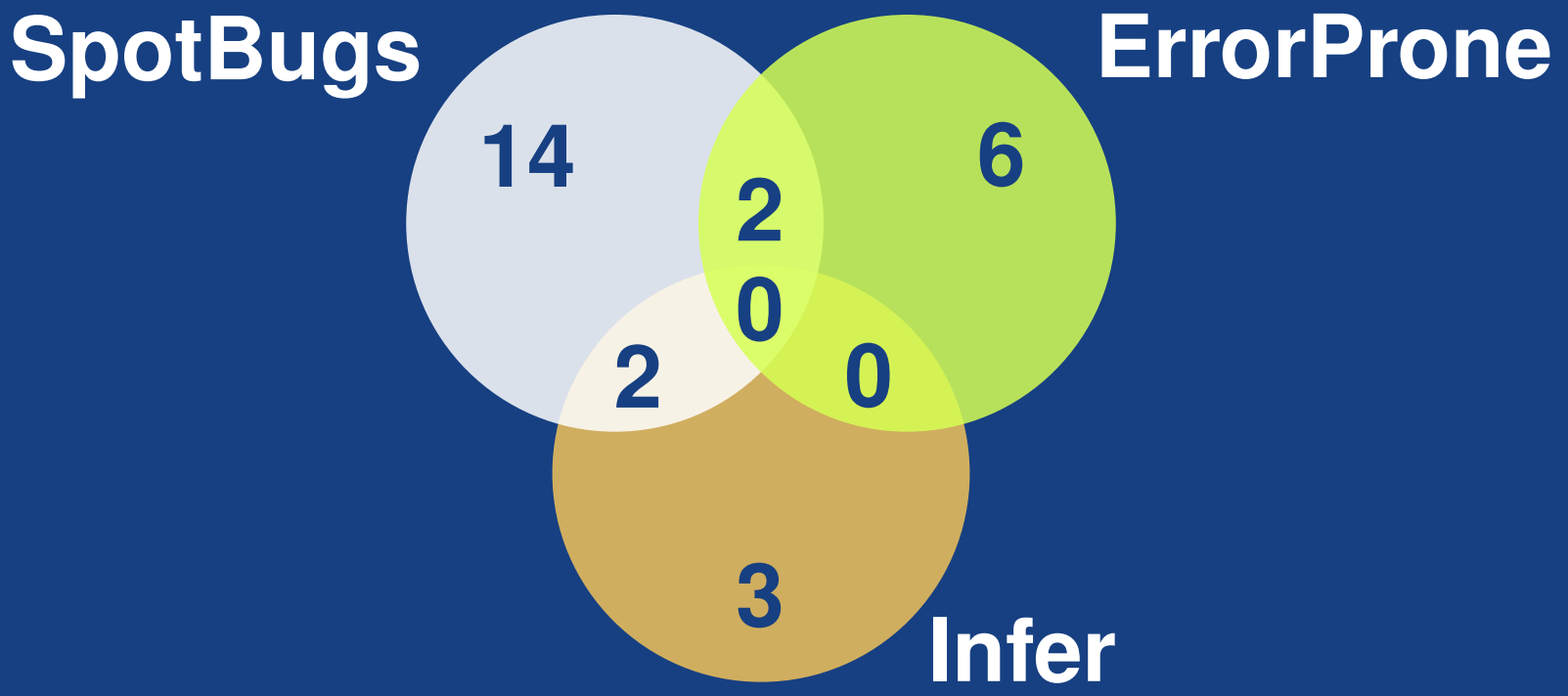
Warning: Chaining constructor ignores argument

Full match

Most Bugs are Missed

Three tools together:

Detect 27 of 594 bugs (less than 5%)



Why are Most Bugs Missed?

Manual inspection of random sample of 20 missed bugs:

Why are Most Bugs Missed?

Manual inspection of random sample of 20 missed bugs:

14 are domain-specific

- Unrelated to any of the supported bug patterns
  - Application-specific algorithms
  - Forgot to handle special case
- Difficult to decide whether behavior is intended

Why are Most Bugs Missed? (2)

Manual inspection of random sample of 20 missed bugs:

6 are near misses

- Root cause is targeted by bug detector, but current implementation misses the bug
- Detector targets similar, but not the same, problem

Conclusion

- Novel methodology to measure how many of a set of bugs are detected
- Popular static bug detectors miss most bugs
- Main reason: Domain-specific bugs vs. generic bug patterns
- Huge potential for future work on bug detection

Implications for Future Work

Huge potential for:

- Bug detectors that catch domain-specific bugs
- More sophisticated yet precise static analyses
- Generalizations of existing bug checkers
- Bug finding techniques other than static analysis, e.g., test generation