CSE 403

Software Engineering

Winter 2023

Statistical fault localization

Logistics

Today

- Effective debugging
- Statistical fault localization

Next week

- Hack day on Monday (final release push).
- Lecture on advanced program analysis on Wednesday.
- Optional in-class exercise (extra-credit) on Friday.

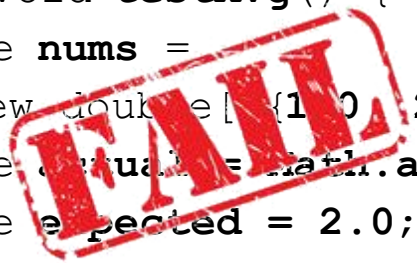
Effective debugging

Software testing vs. software debugging

```
1  double avg(double[] nums) {
2    int n = nums.length;
3    double sum = 0;
4
5    int i = 0;
6    while (i<n) {
7      sum = sum + nums[i];
8      i = i + 1;
9    }
10
11   double avg = sum * n;
12   return avg;
13 }
```

Testing: is there a bug?

```
@Test
public void testAvg() {
  double nums =
      new double[]{1.0, 2.0, 3.0});
  double actual = Math.avg(nums);
  double expected = 2.0;
  assertEquals(expected,actual,EPS);
}
```

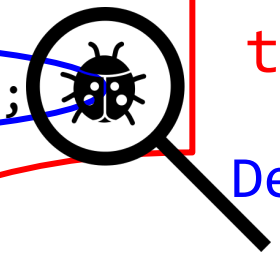



```
testAvg failed: 2.0 != 18.0
```

Starting point: a failing (bug-triggering) test.

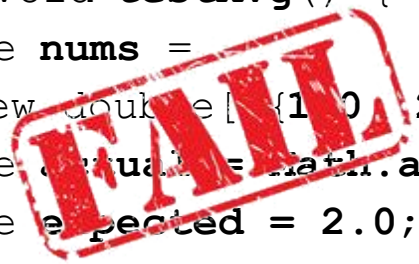
Software testing vs. software debugging

```
1  double avg(double[] nums) {
2    int n = nums.length;
3    double sum = 0;
4
5    int i = 0;
6    while (i<n) {
7      sum = sum + nums[i];
8      i = i + 1;
9    }
10
11   double avg = sum * n;
12   return avg;
13 }
```



Testing: is there a bug?

```
@Test
public void testAvg() {
  double nums =
      new double[]{1.0, 2.0, 3.0});
  double actual = MathL.avg(nums);
  double expected = 2.0;
  assertEquals(expected,actual,EPS);
}
```

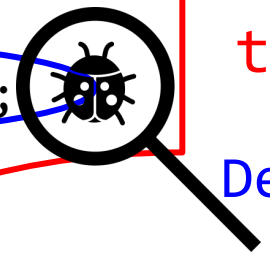



```
testAvg failed: 2.0 != 18.0
```

Debugging: where is the bug?
how to fix the bug?

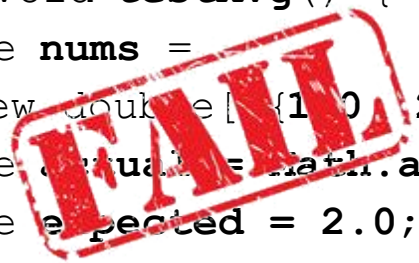
Software testing vs. software debugging

```
1  double avg(double[] nums) {
2    int n = nums.length;
3    double sum = 0;
4
5    int i = 0;
6    while (i<n) {
7      sum = sum + nums[i];
8      i = i + 1;
9    }
10
11   double avg = sum * n;
12   return avg;
13 }
```



Testing: is there a bug?

```
@Test
public void testAvg() {
  double nums =
      new double[]{1.0, 2.0, 3.0});
  double actual = MathL.avg(nums);
  double expected = 2.0;
  assertEquals(expected,actual,EPS);
}
```




```
testAvg failed: 2.0 != 18.0
```

Debugging: where is the bug?
how to fix the bug?

What testing practices support effective debugging?

Testing best practices

- Naming: proper names for tests (clear link to tested class/method)
- Output: meaningful failure messages
- Atomicity: one test per behavior
- Style: one test, one assertion vs. one test, multiple assertions

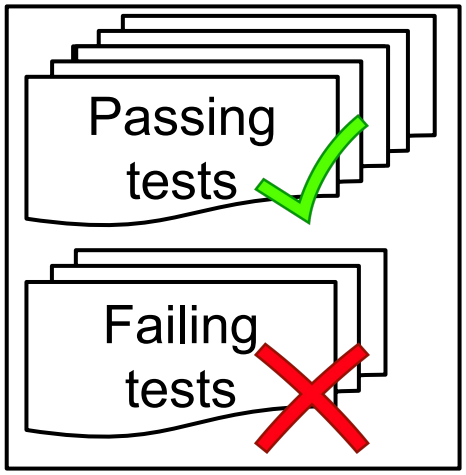
Statistical fault localization

What is statistical fault localization?

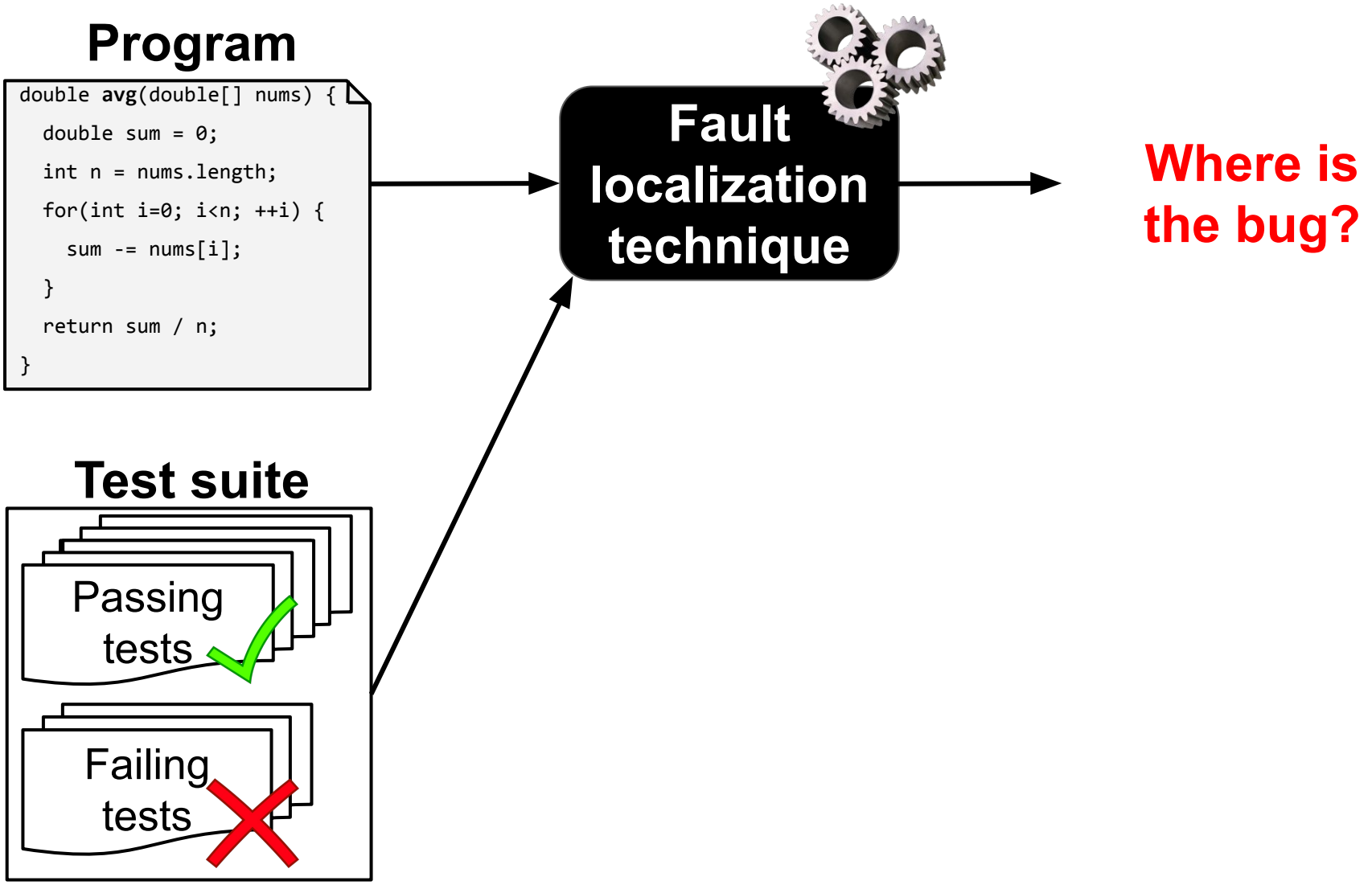
Program

```
double avg(double[] nums) {
  double sum = 0;
  int n = nums.length;
  for(int i=0; i<n; ++i) {
    sum -= nums[i];
  }
  return sum / n;
}
```

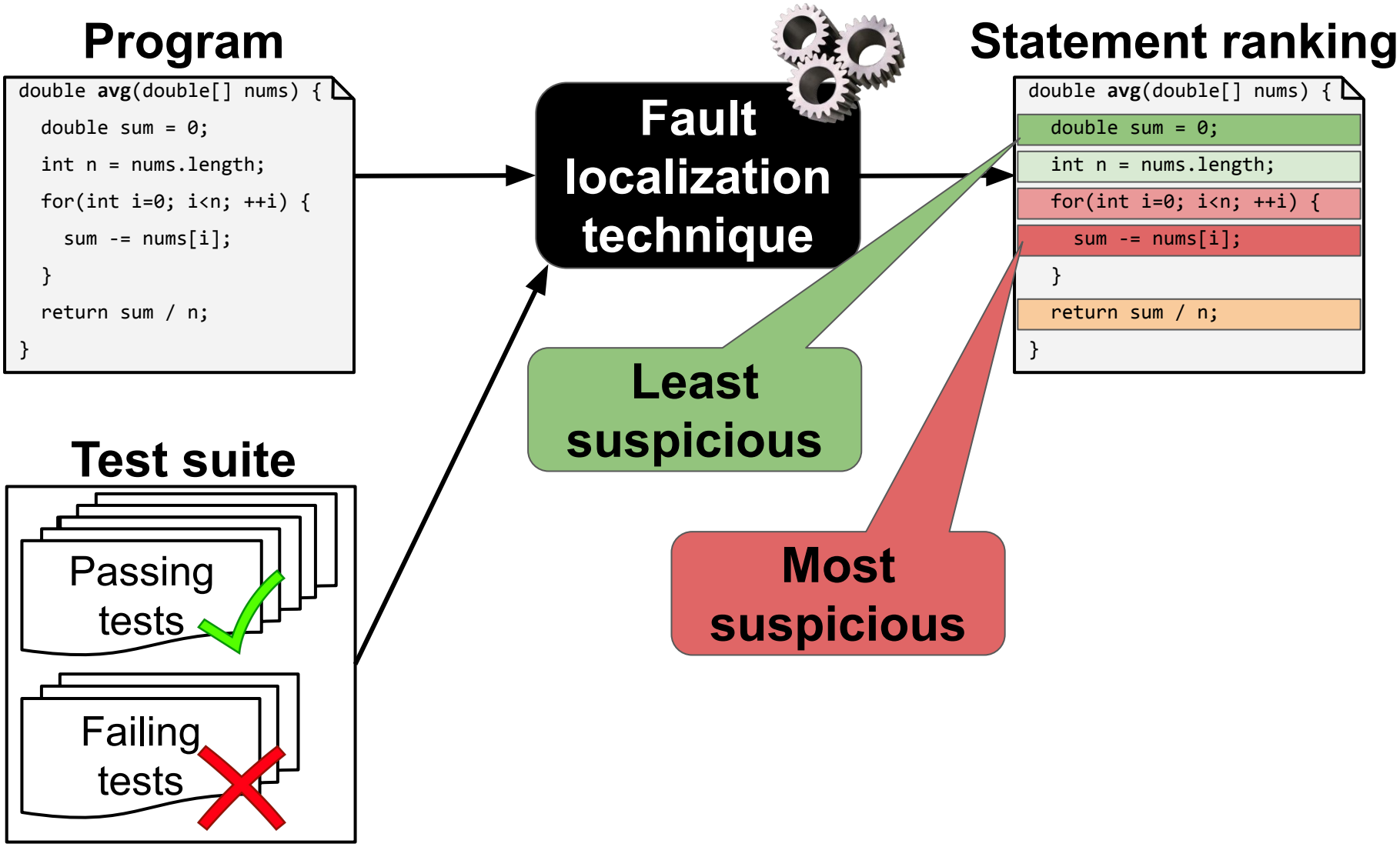
Test suite



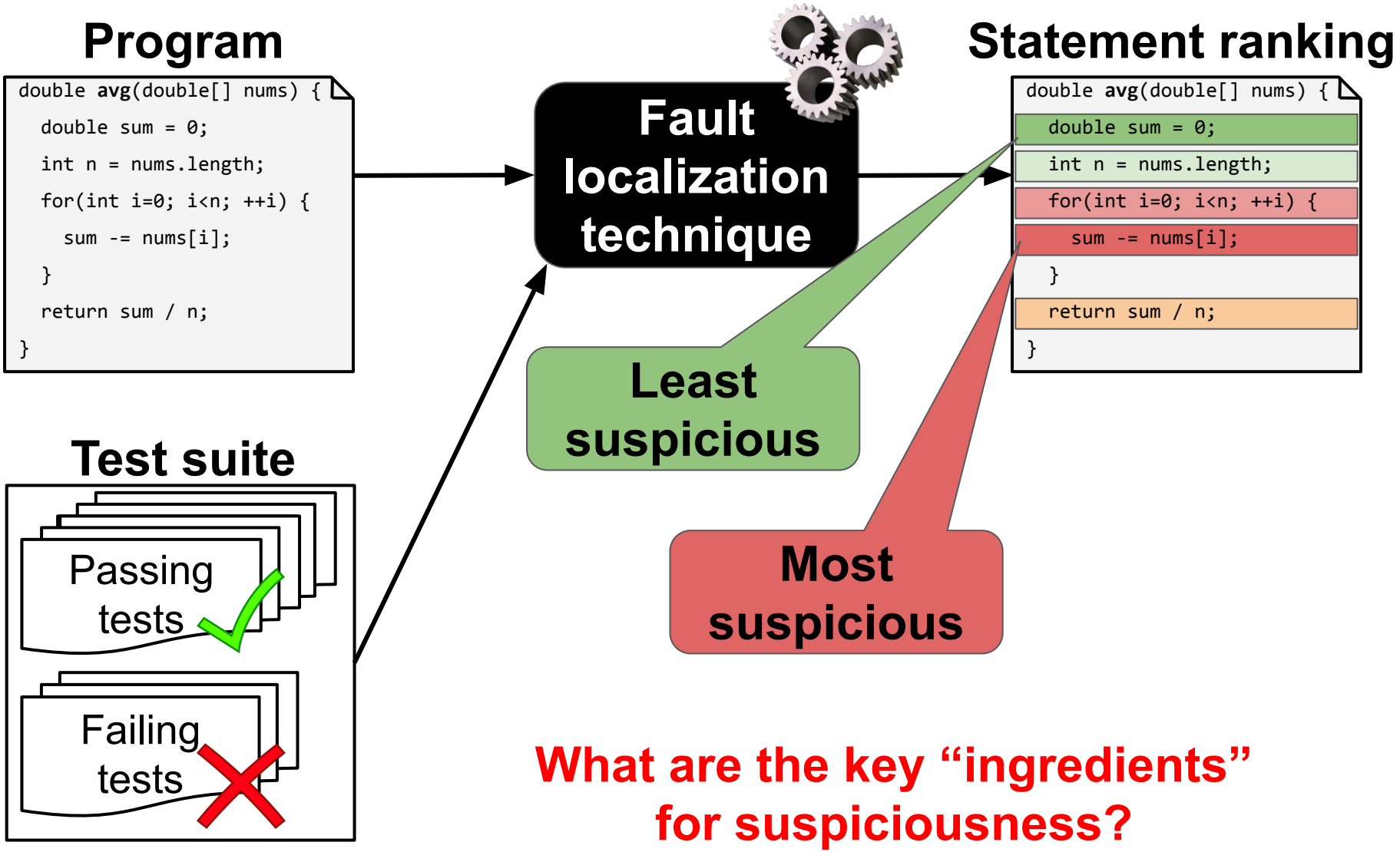
What is statistical fault localization?



What is statistical fault localization?



What is statistical fault localization?



What are the key "ingredients" for suspiciousness?

Statistical fault localization: how it works

Program

```
double avg(double[] nums) {
  double sum = 0;
  int n = nums.length;
  for(int i=0; i<n; ++i) {
    sum -= nums[i];
  }
  return sum / n;
}
```

Statistical fault localization: how it works

Program

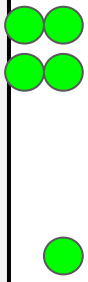
```
double avg(double[] nums) {
  double sum = 0;
  int n = nums.length;
  for(int i=0; i<n; ++i) {
    sum -= nums[i];
  }
  return sum / n;
}
```

- Run all tests
  - t1 passes 

Statistical fault localization: how it works

Program

```
double avg(double[] nums) {
  double sum = 0;
  int n = nums.length;
  for(int i=0; i<n; ++i) {
    sum -= nums[i];
  }
  return sum / n;
}
```

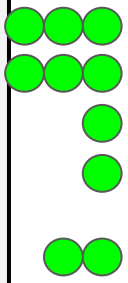


- Run all tests
  - t1 passes
  - t2 passes

Statistical fault localization: how it works

Program

```
double avg(double[] nums) {
  double sum = 0;
  int n = nums.length;
  for(int i=0; i<n; ++i) {
    sum -= nums[i];
  }
  return sum / n;
}
```

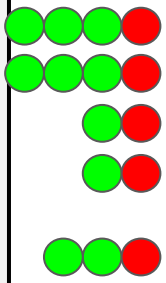


- Run all tests
  - t1 passes
  - t2 passes
  - t3 passes

Statistical fault localization: how it works

Program

```
double avg(double[] nums) {
  double sum = 0;
  int n = nums.length;
  for(int i=0; i<n; ++i) {
    sum -= nums[i];
  }
  return sum / n;
}
```

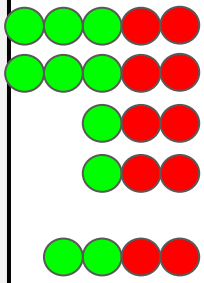


- Run all tests
  - t1 passes
  - t2 passes
  - t3 passes
  - t4 fails

Statistical fault localization: how it works

Program

```
double avg(double[] nums) {
  double sum = 0;
  int n = nums.length;
  for(int i=0; i<n; ++i) {
    sum -= nums[i];
  }
  return sum / n;
}
```



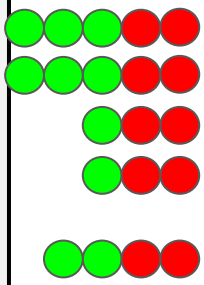
- Run all tests
  - t1 passes
  - t2 passes
  - t3 passes
  - t4 fails
  - t5 fails

Which line(s) seem(s) most suspicious?

Spectrum-based fault localization

Program

```
double avg(double[] nums) {
  double sum = 0;
  int n = nums.length;
  for(int i=0; i<n; ++i) {
    sum -= nums[i];
  }
  return sum / n;
}
```



Spectrum-based FL (SBFL)

- **Compute** suspiciousness **per statement**
- Example:

$$S(s) = \frac{failed(s)/totalfailed}{failed(s)/totalfailed + passed(s)/totalpassed}$$

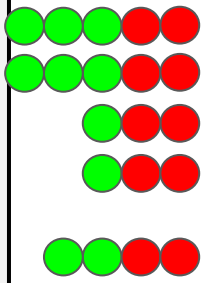
- Statement **covered** by **failing test**
- Statement **covered** by **passing test**

More ● ➡ statement is more suspicious!

Spectrum-based fault localization

Program

```
double avg(double[] nums) {
  double sum = 0;
  int n = nums.length;
  for(int i=0; i<n; ++i) {
    sum -= nums[i];
  }
  return sum / n;
}
```

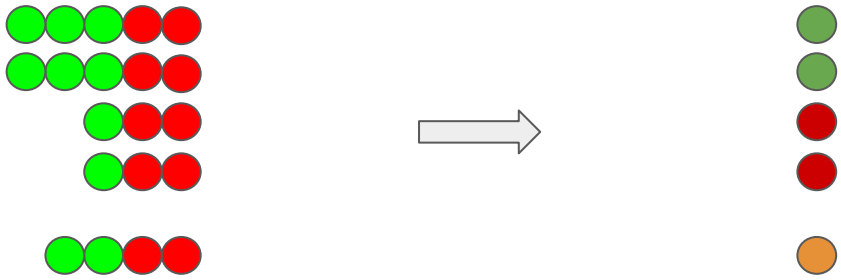


Spectrum-based FL (SBFL)

- **Compute** suspiciousness **per statement**
- Example:

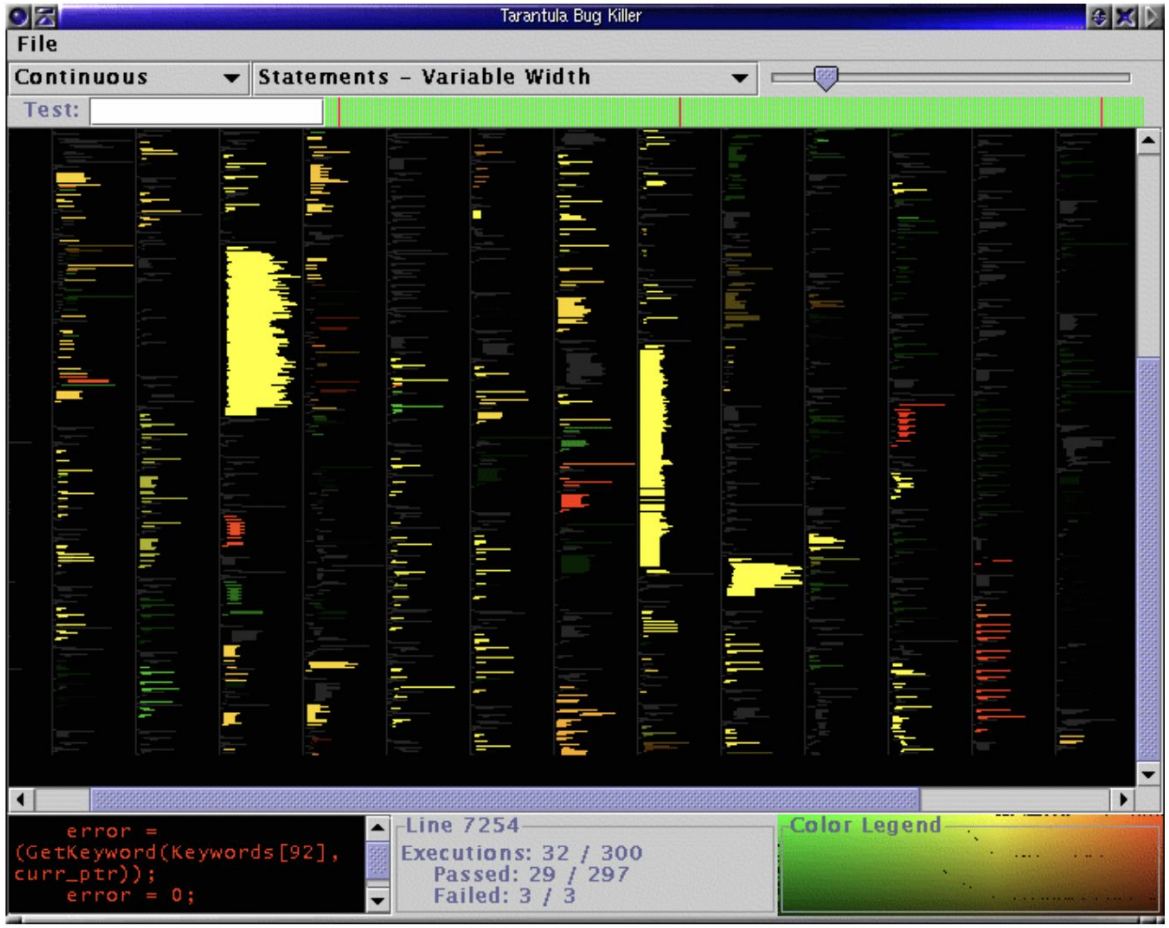
$$S(s) = \frac{failed(s)/totalfailed}{failed(s)/totalfailed + passed(s)/totalpassed}$$

Visualization: the key idea behind Tarantula.



Jones et al., *Visualization of test information to assist fault localization*, ICSE'02

Spectrum-based fault localization

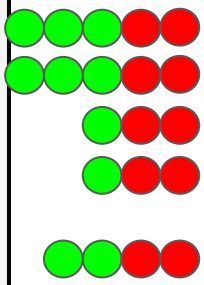


Jones et al., *Visualization of test information to assist fault localization*, ICSE'02

Spectrum-based fault localization

Program

```
double avg(double[] nums) {
  double sum = 0;
  int n = nums.length;
  for(int i=0; i<n; ++i) {
    sum -= nums[i];
  }
  return sum / n;
}
```



Spectrum-based FL (SBFL)

- **Compute** suspiciousness **per statement**
- Example:

$$S(s) = \frac{failed(s)/totalfailed}{failed(s)/totalfailed + passed(s)/totalpassed}$$

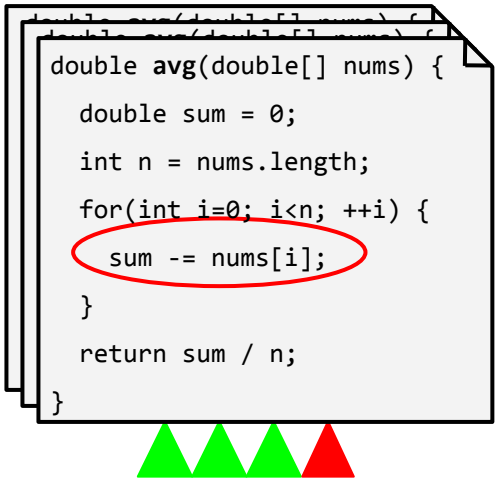
What information should we (intuitively) consider when computing suspiciousness?

Mutation-based fault localization

Program

```
double avg(double[] nums) {
  double sum = 0;
  int n = nums.length;
  for(int i=0; i<n; ++i) {
    sum -= nums[i];
  }
  return sum / n;
}
```

Mutants



Mutation-based FL (MBFL)

- **Compute** suspiciousness **per mutant**
- **Aggregate** results **per statement**
- Example:

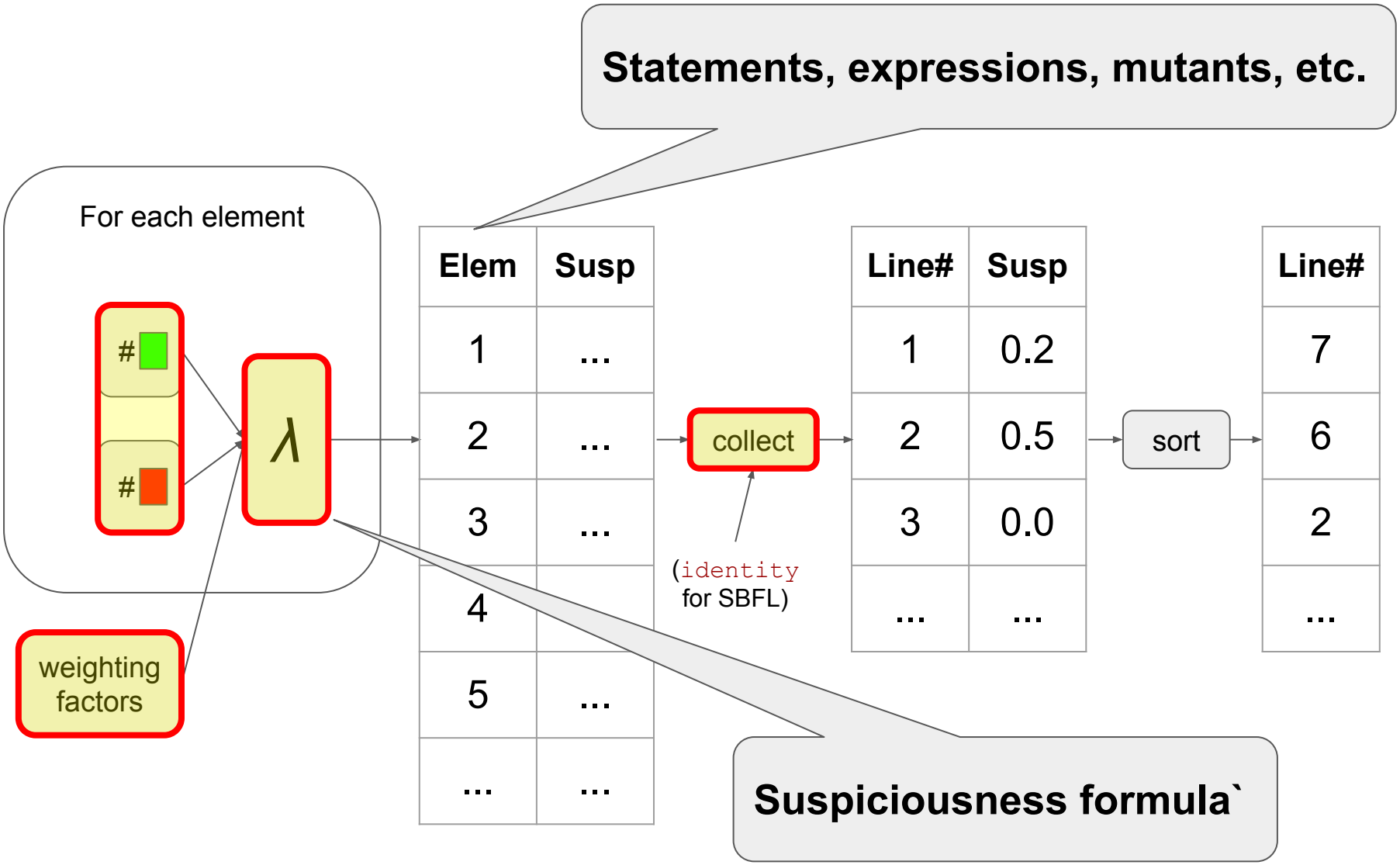
$$S(s) = \max_{m \in mut(s)} \frac{failed(m)}{\sqrt{totalfailed \cdot (failed(m) + passed(m))}}$$

▲ Mutant **affects failing test** outcome

▲ Mutant **breaks passing test**

More ▲ ➡ mutant is more suspicious!

Common structure of SBFL and MBFL



Statistical fault localization: live example

Testing best practices revisited

- Naming: proper names for tests (clear link to tested class/method)
- Output: meaningful failure messages
- **Atomicity: one test per behavior**
- Style: one test, one assertion vs. one test, multiple assertions

Effectiveness of SBFL and MBFL

Percentage of buggy statements found when inspecting the top-n suspicious statements.

Technique	Top-5	Top-10	Top-200
Hybrid	36%	45%	85%
DStar (*best SBFL*)	30%	39%	82%
Metallaxis (*best MBFL*)	29%	39%	77%

- Top-10 useful for practitioners[1].
- Top-200 useful for automated patch generation[2].

What assumptions underpin these results? Are they realistic?

[1]Kochhar et al., *Practitioners' Expectations on Automated Fault Localization*, ISSTA'16
[2]Long and Rinard, *An analysis of the search spaces for generate and validate patch generation systems*, ICSE'16

Automated patch generation

Automatic patch generation (program repair)

Generate-and-validate Approaches

What are the **main components** of a (generate-and-validate) patch generation approach?

Automatic patch generation (program repair)

Generate-and-validate Approaches

Main components:

- **Fault localization**
- Mutation + fitness evaluation
- Patch validation