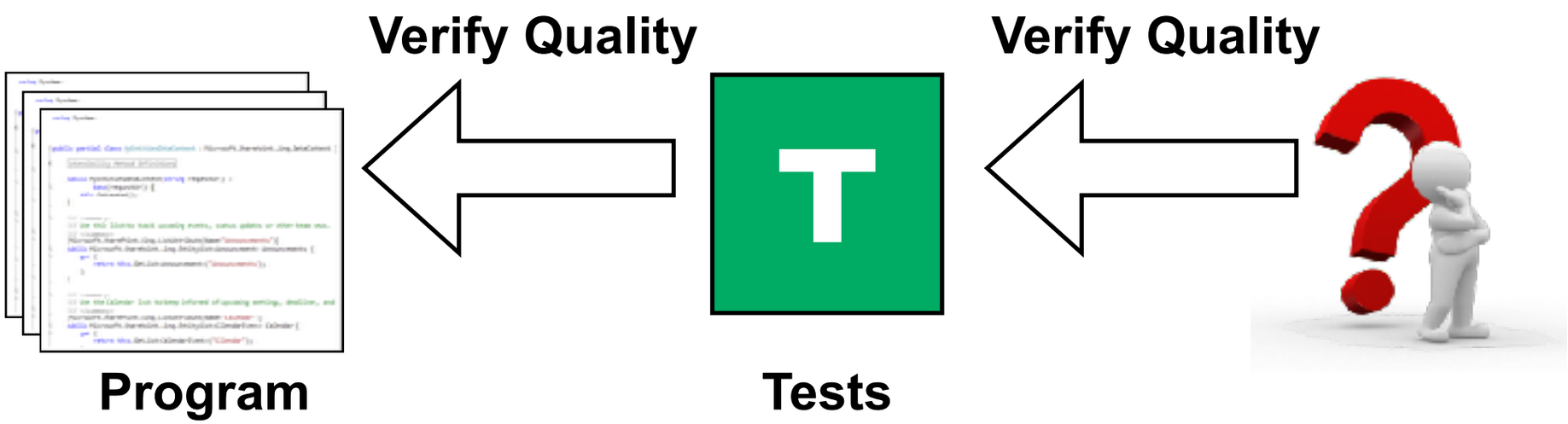
CE/CS/SE 3354 Software Engineering

Code Coverage

Who will test the tests?

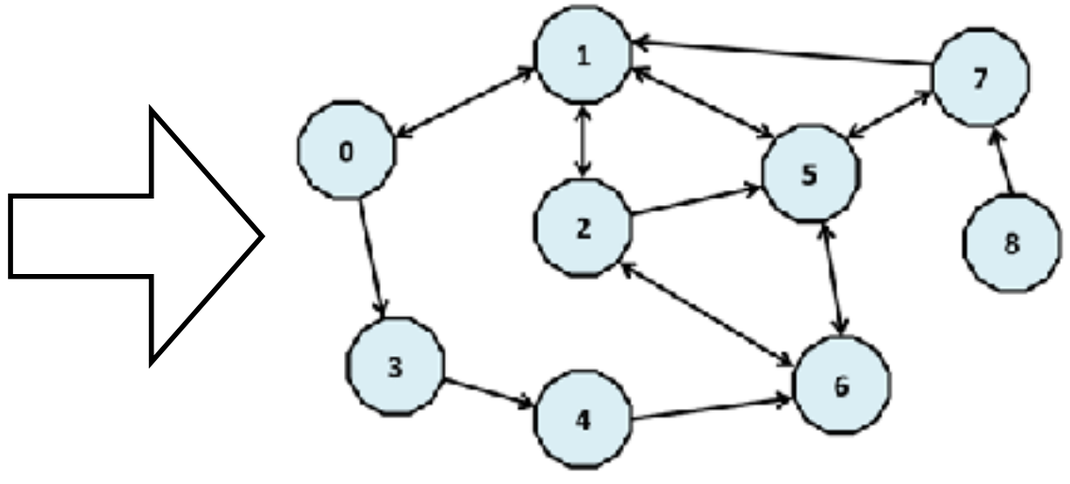


- Code coverage can be a way!
  - Usually, a test covering/executing more code may indicate better test quality

How to measure code coverage

Overview

- A common way is to abstract program into graphs
  - Graph : Usually the control flow graph (CFG)
  - Node coverage : Execute every statement
  - Edge coverage : Execute every branch

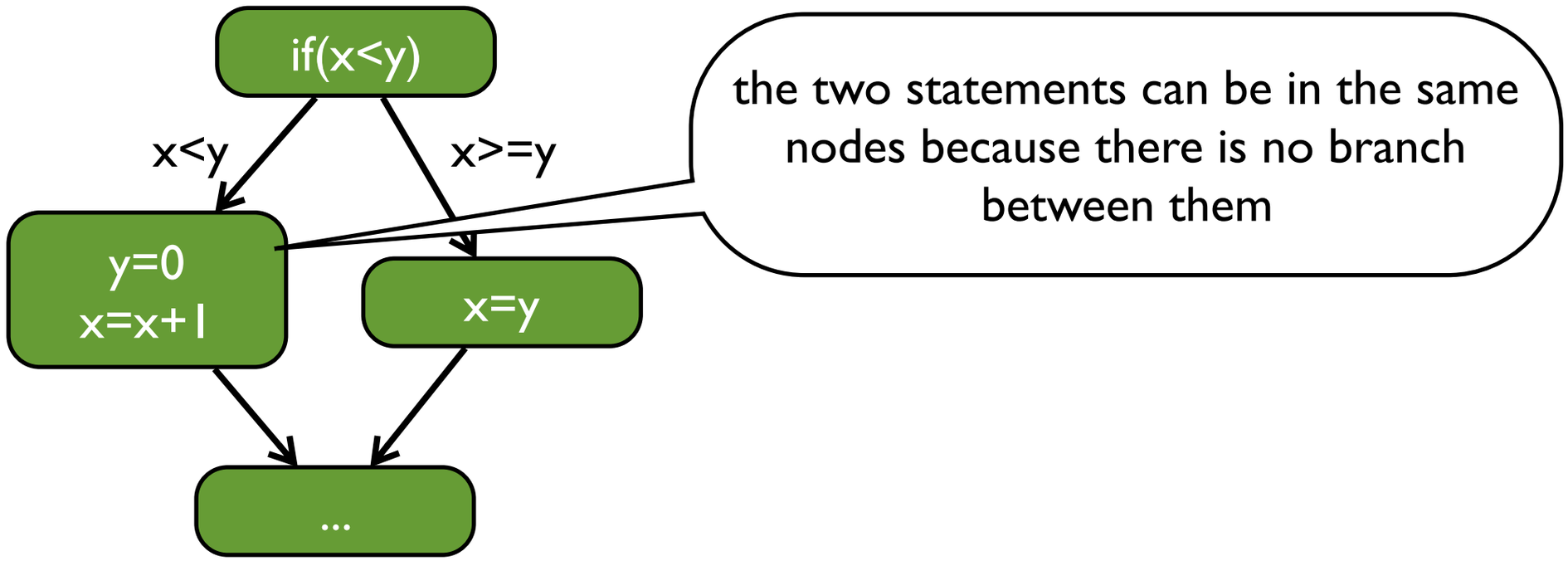


Control Flow Graphs

- A CFG models all executions of a program by describing control structures
  - Basic Block : A sequence of statements with only one entry point and only one exit point (no branches)
  - Nodes : Statements or sequences of statements (basic blocks)
  - Edges : Transfers of control

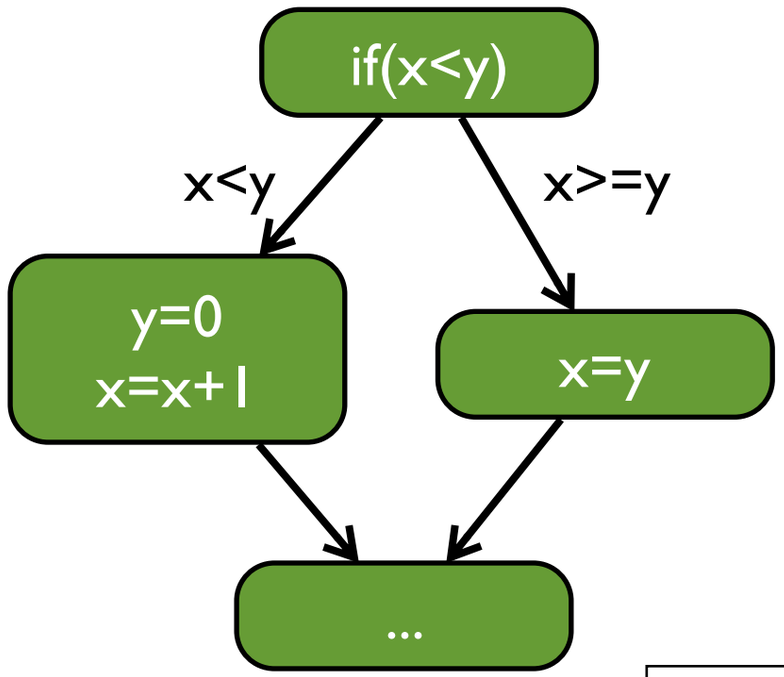
CFG : The if Statement

```
if (x < y)
{
   y = 0;
   x = x + 1;
}
else
{
   x = y;
}
...
```



CFG : The if Statement

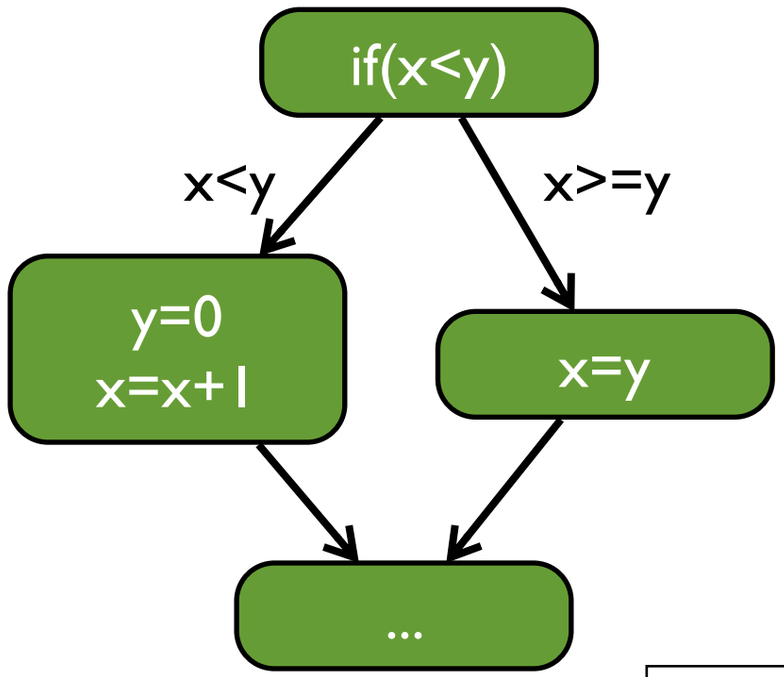
```
if (x < y)
{
   y = 0;
   x = x + 1;
}
else
{
   x = y;
}
...
```



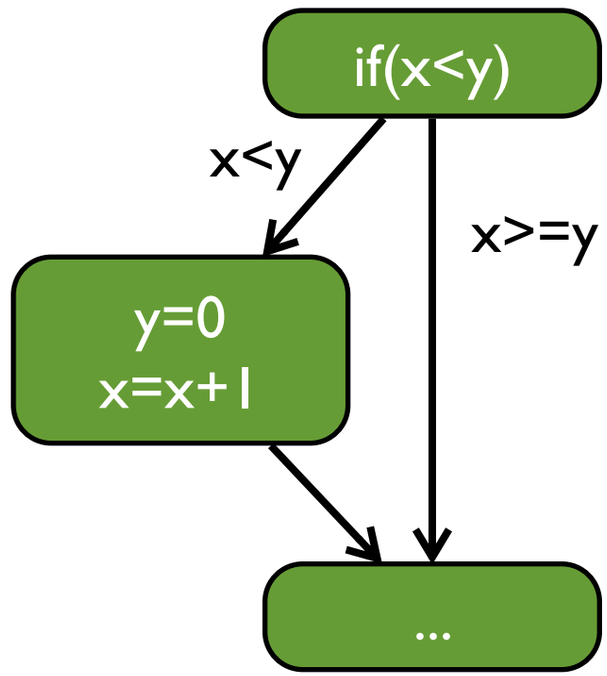

```
if (x < y)
{
   y = 0;
   x = x + 1;
}
...
```

CFG : The if Statement

```
if (x < y)
{
   y = 0;
   x = x + 1;
}
else
{
   x = y;
}
...
```

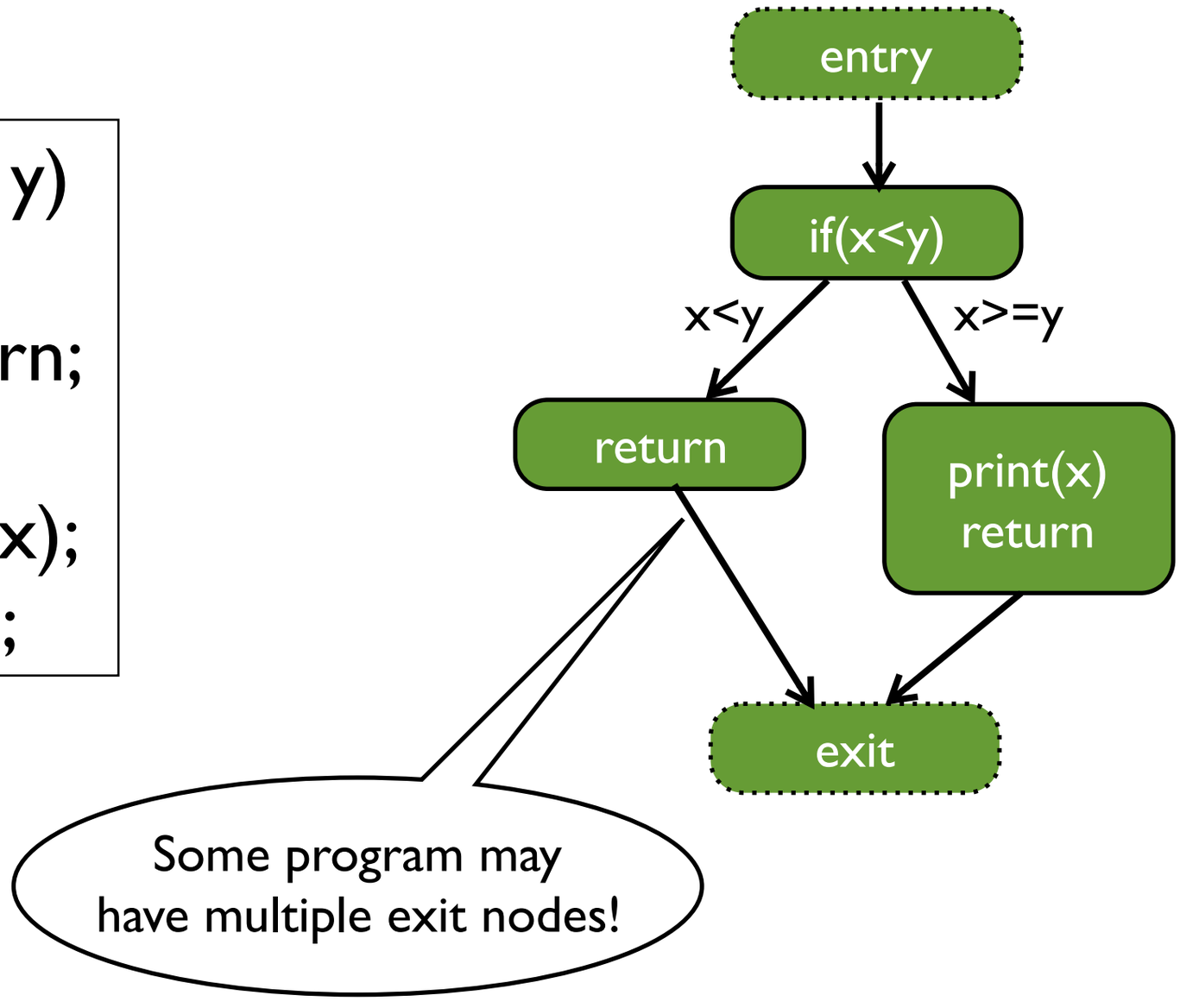



```
if (x < y)
{
   y = 0;
   x = x + 1;
}
...
```



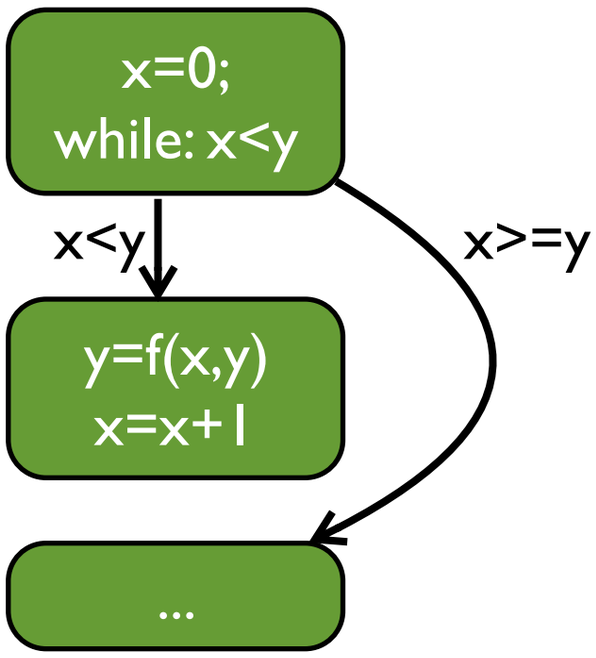
CFG : The Dummy Nodes

```
if (x < y)
{
   return;
}
print (x);
return;
```



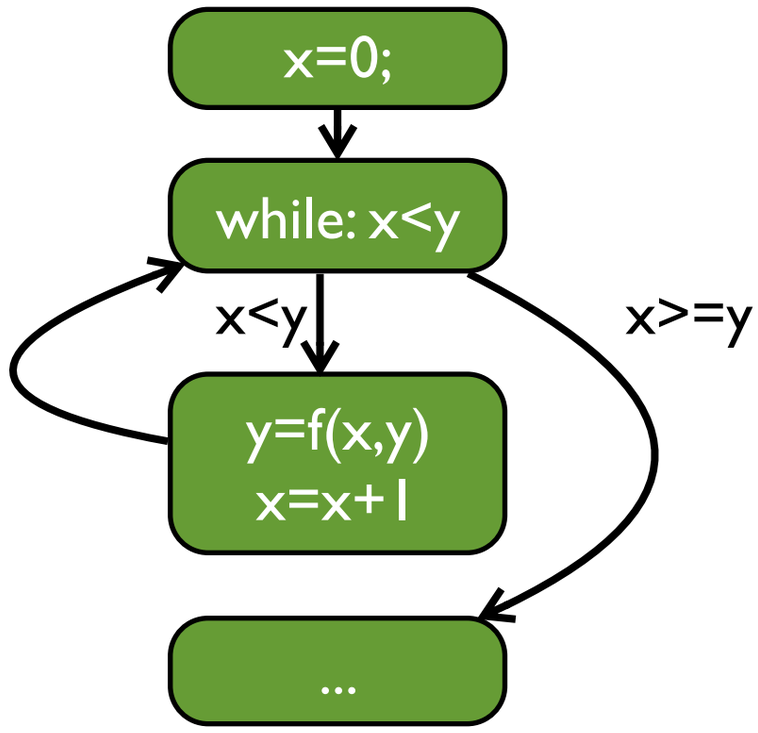
CFG : while and for Loops

```
x=0;
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
...
```



CFG : while and for Loops

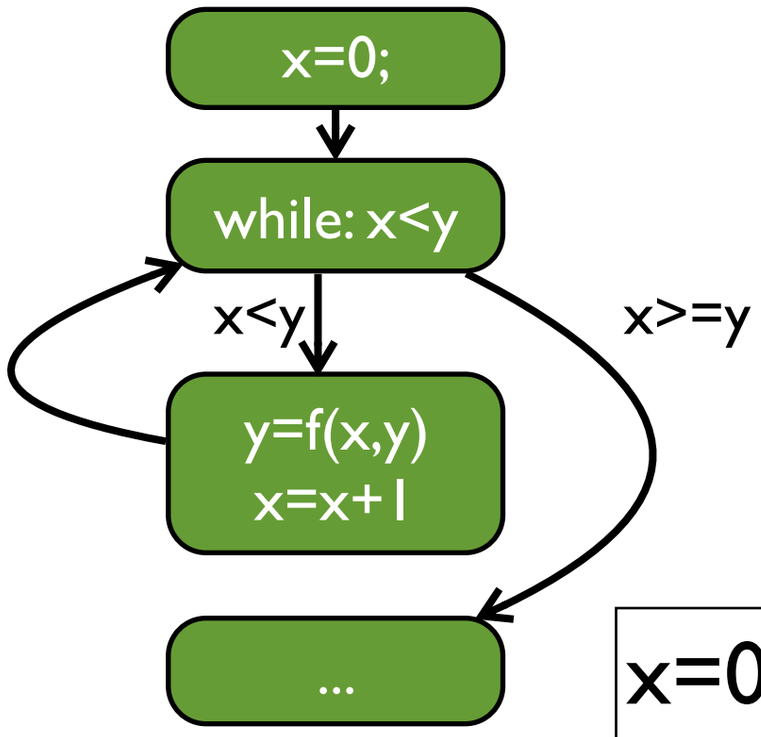
```
x=0;
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
...
```




```
for (x = 0; x < y; x++)
{
   y = f (x, y);
}
```

CFG : while and for Loops

```
x=0;
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
...
```

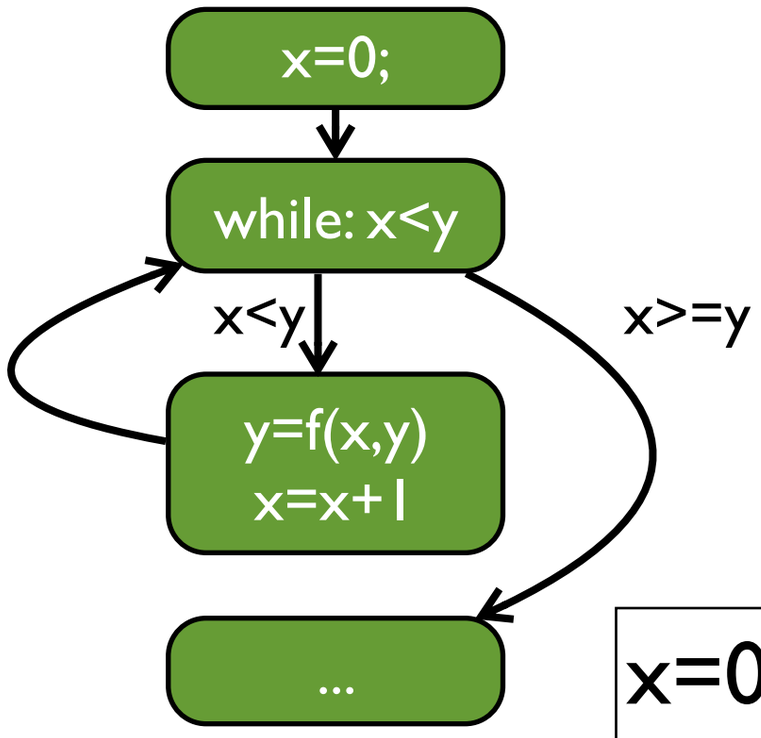



```
for (x = 0; x < y; x++)
{
   y = f (x, y);
}
```

```
x=0;
do {
   y = f (x, y);
   x = x + 1;
}
while (x < y)
...
```

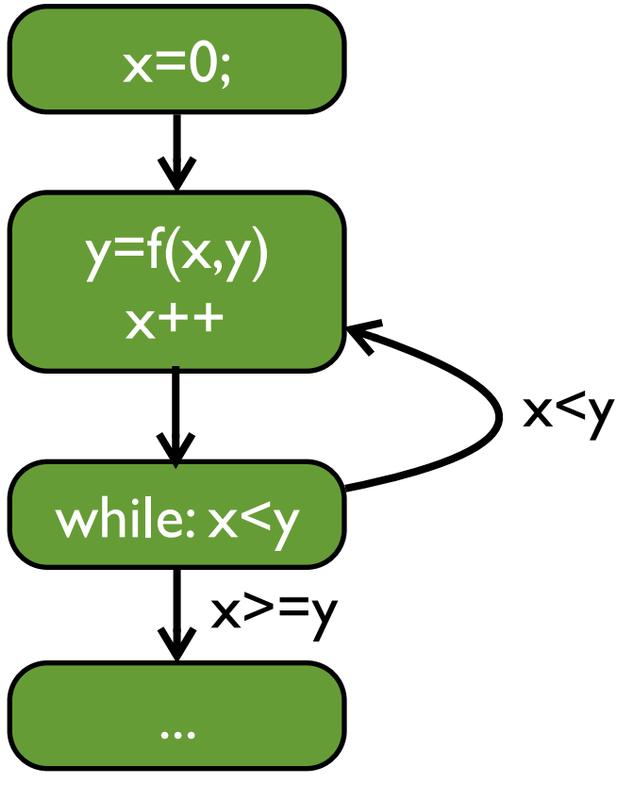
CFG : while and for Loops

```
x=0;
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
...
```



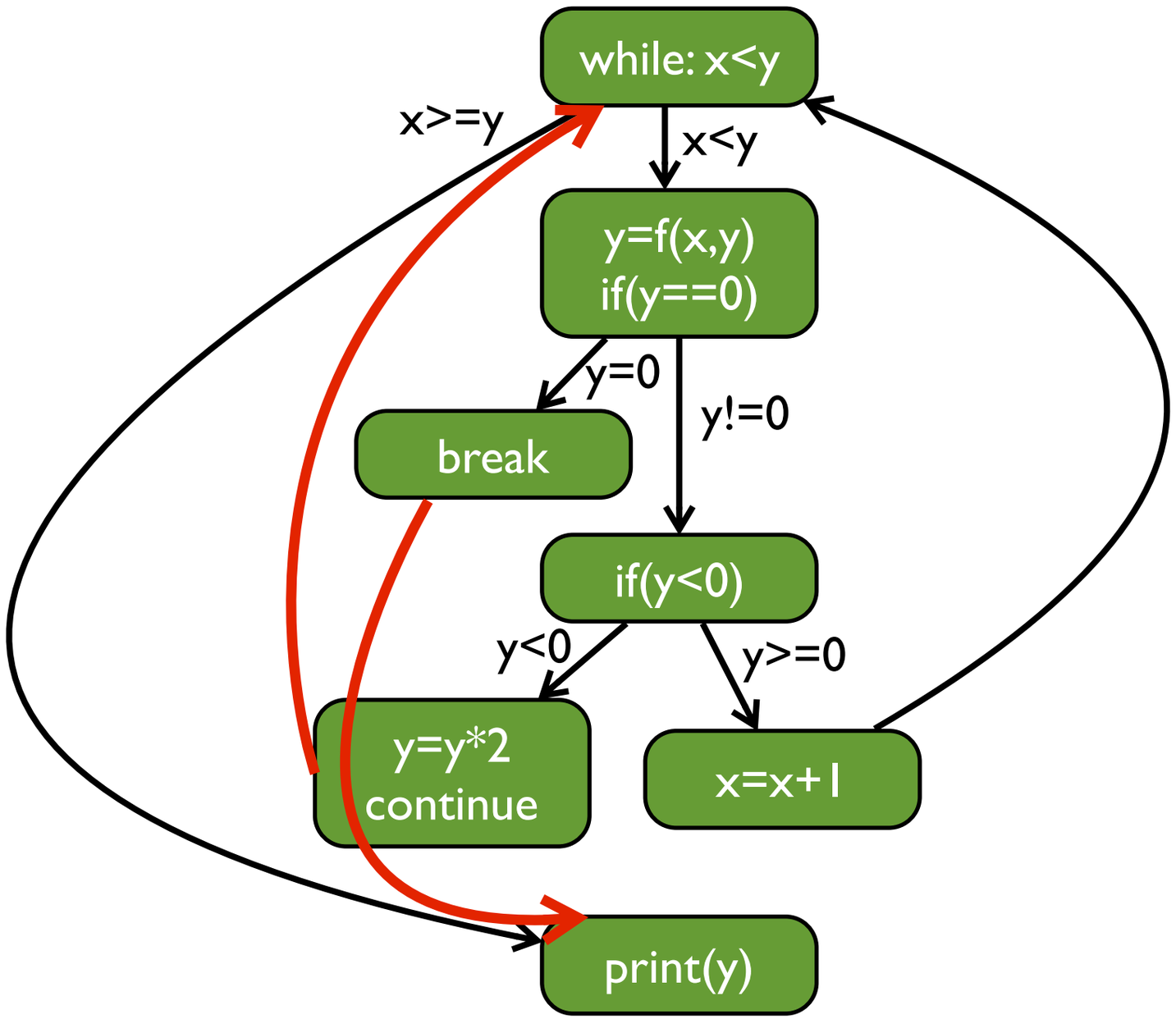

```
for (x = 0; x < y; x++)
{
   y = f (x, y);
}
```

```
x=0;
do {
   y = f (x, y);
   x = x + 1;
}
while (x < y)
...
```



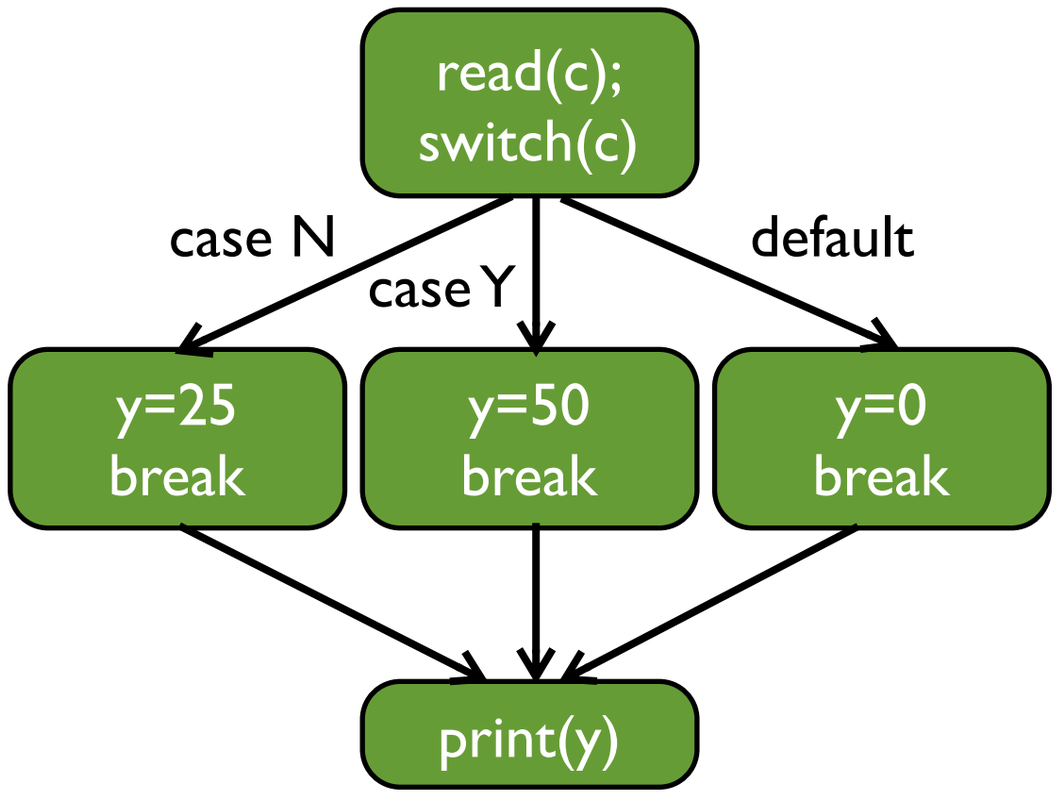
CFG: break and continue

```
while (x < y)
{
   y = f (x, y);
   if (y == 0) {
      break;
   } else if (y<0) {
      y = y*2;
      continue;
   }
   x = x + 1;
}
print (y);
```



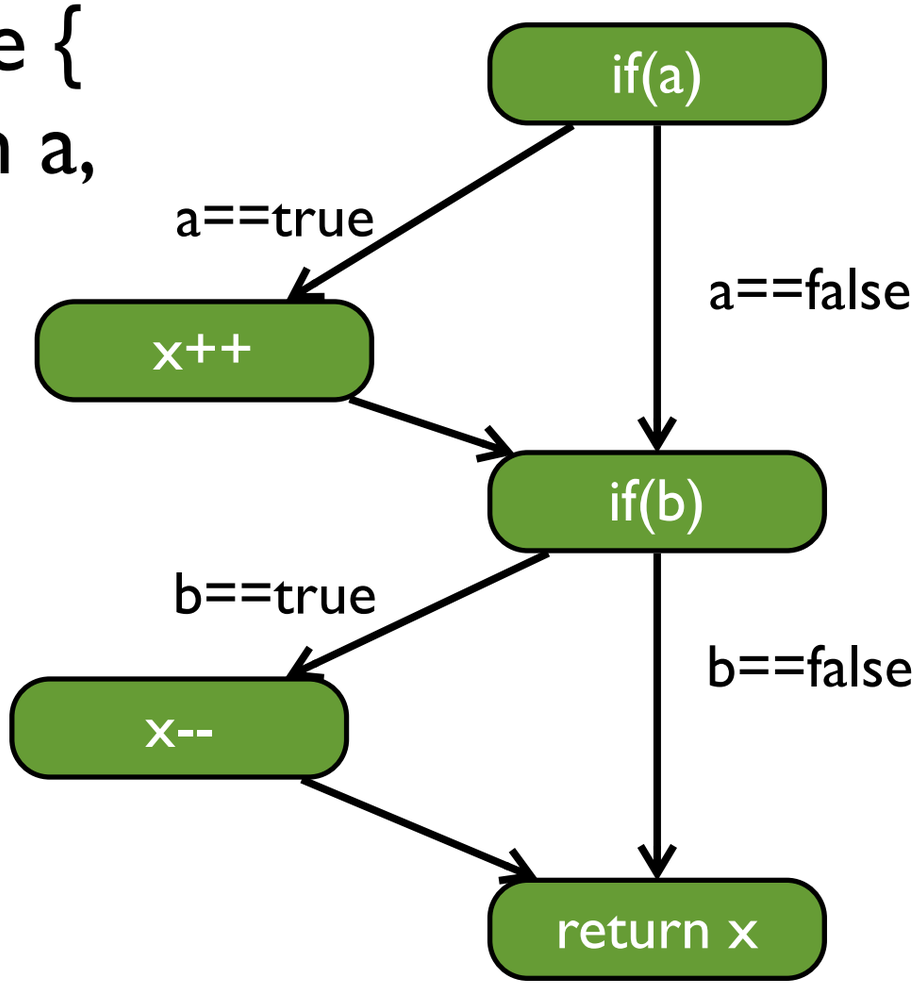
CFG: switch

```
read ( c) ;
switch ( c )
{
   case 'N':
      y = 25;
      break;
   case 'Y':
      y = 50;
      break;
   default:
      y = 0;
      break;
}
print (y);
```



CFG-Based Coverage: Example

```
public class CFGCoverageExample {
    public int testMe(int x, boolean a,
boolean b){
        if(a)
            x++;
        if(b)
            x--;
        return x;
    }
}
```



CFG-based Coverage: A JUnit Test

```
public class JUnitStatementCov {
    CFGCoverageExample tester;
    @Before
    public void initialize() {
        tester = new CFGCoverageExample();
    }
    @Test
    public void testCase() {
        assertEquals(0, tester.testMe(0, true, false));
    }
}
```

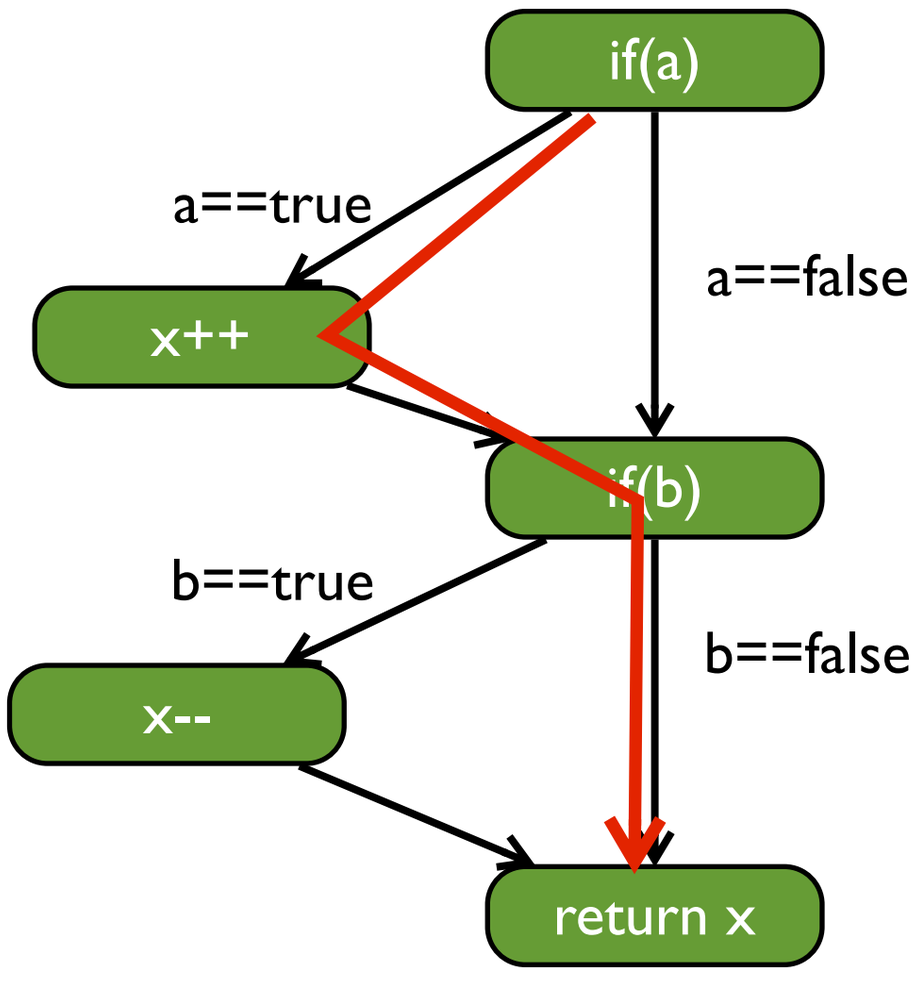
How good is it??

CFG-based Coverage: Statement Coverage

- The percentage of statements covered by the test

tester.testMe(0, true, false)

x=0 a=true b=false

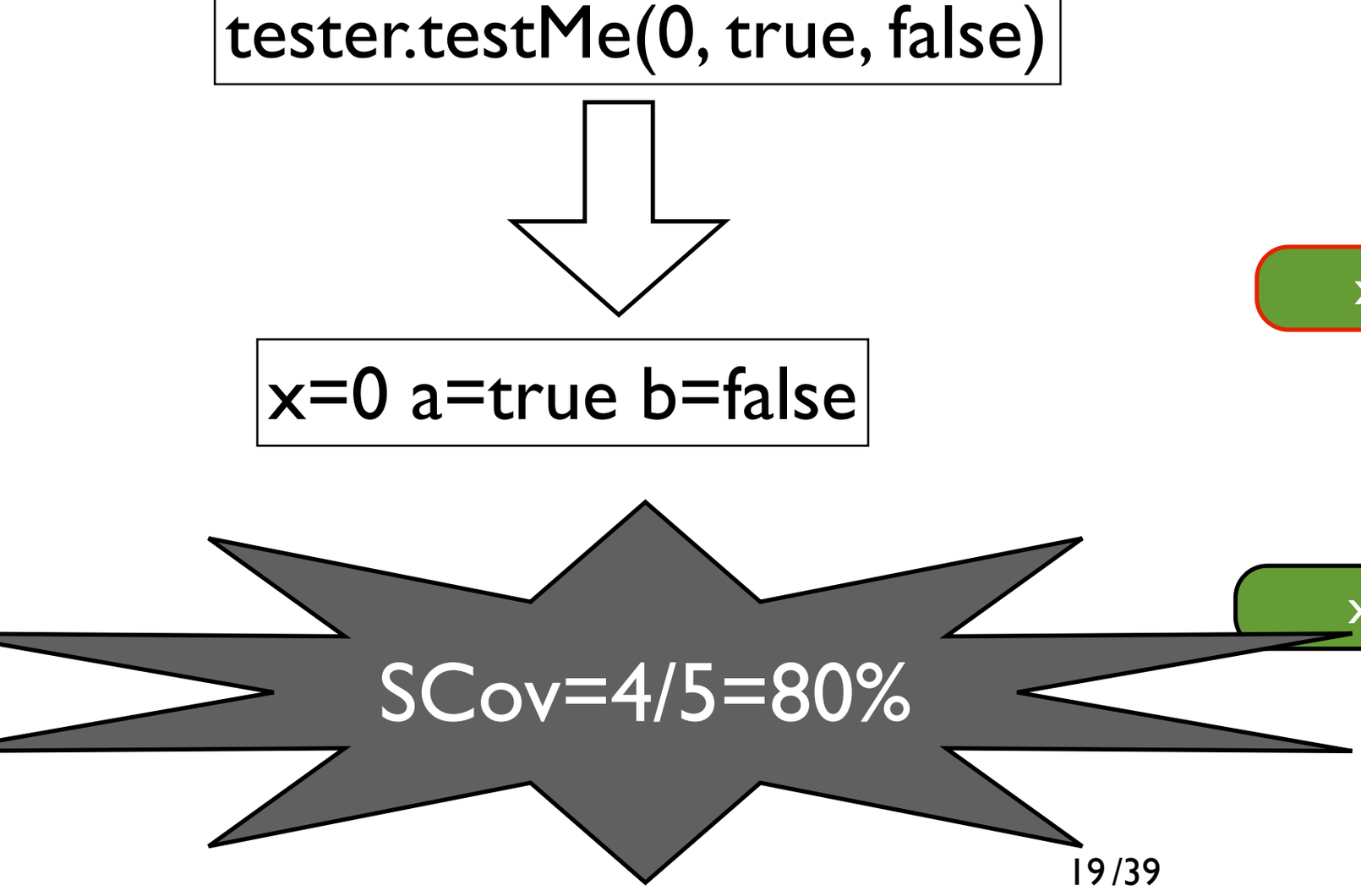


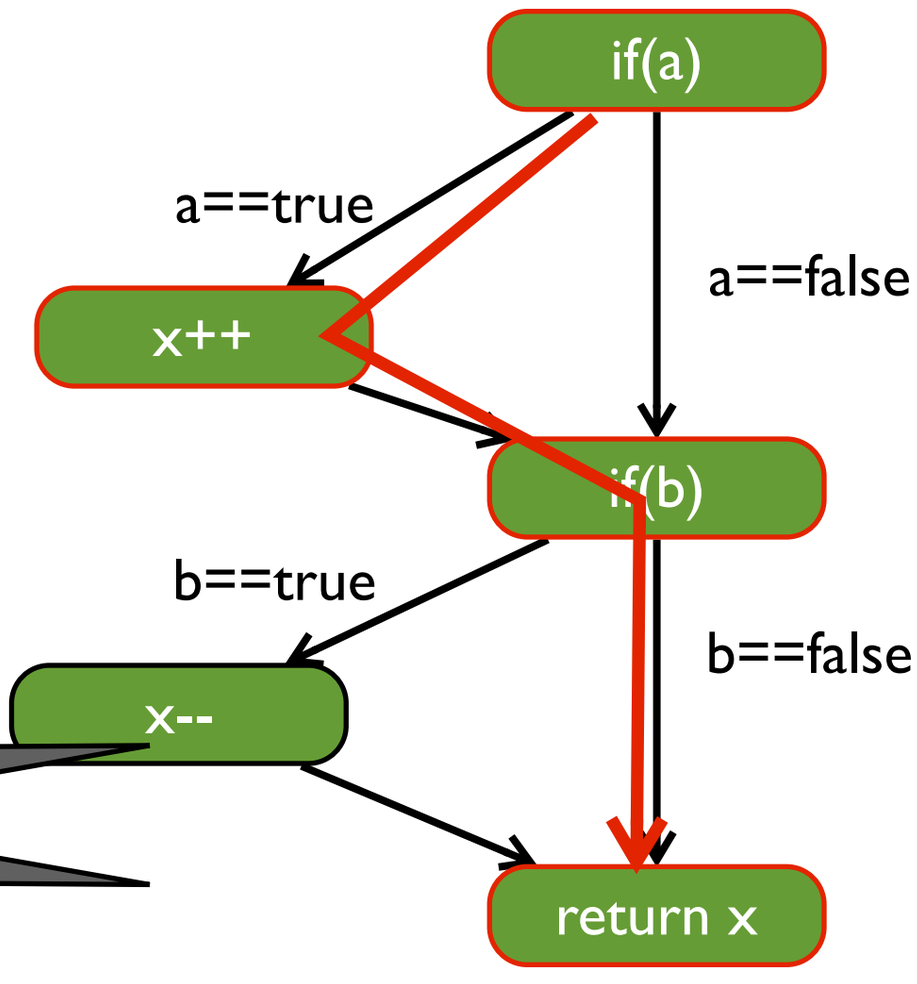
CFG-based Coverage: Statement Coverage

- The percentage of statements covered by the test

tester.testMe(0, true, false)

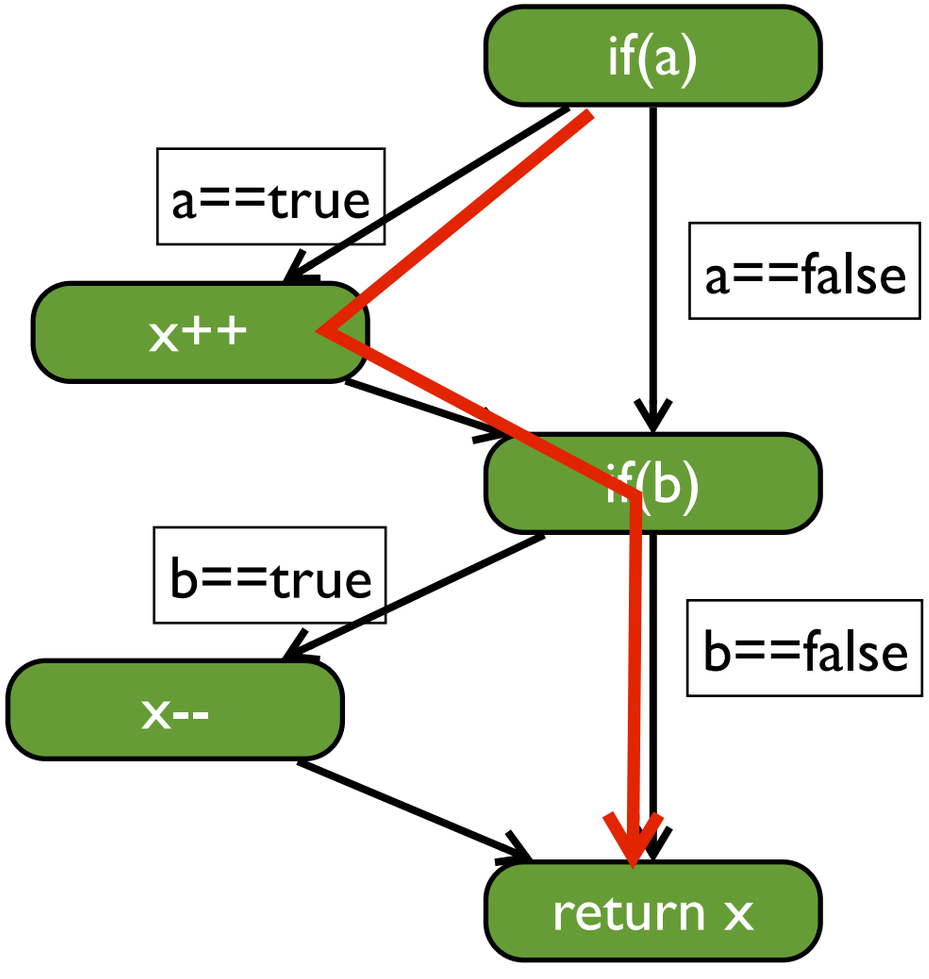
x=0 a=true b=false





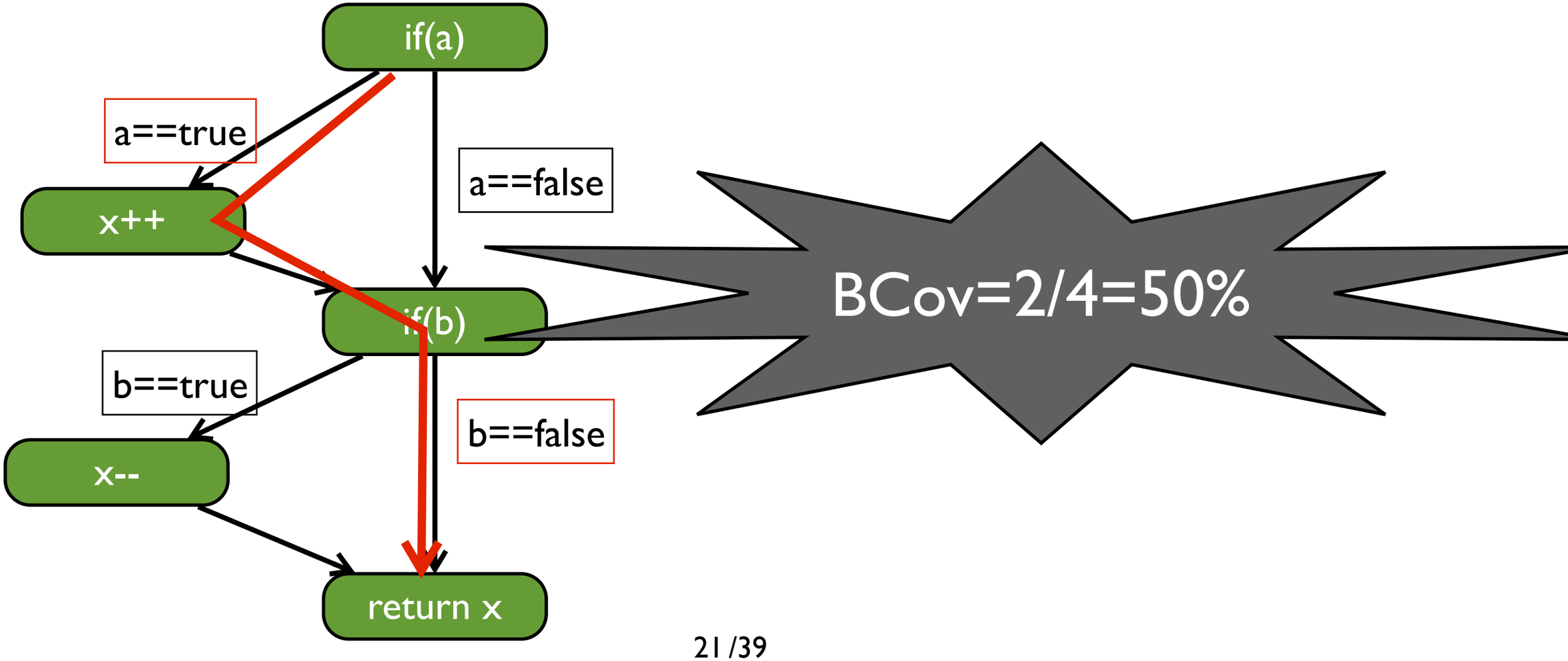
CFG-based Coverage: Branch Coverage

- The percentage of branches covered by the test
  - Consider both false and true branch for each conditional statement



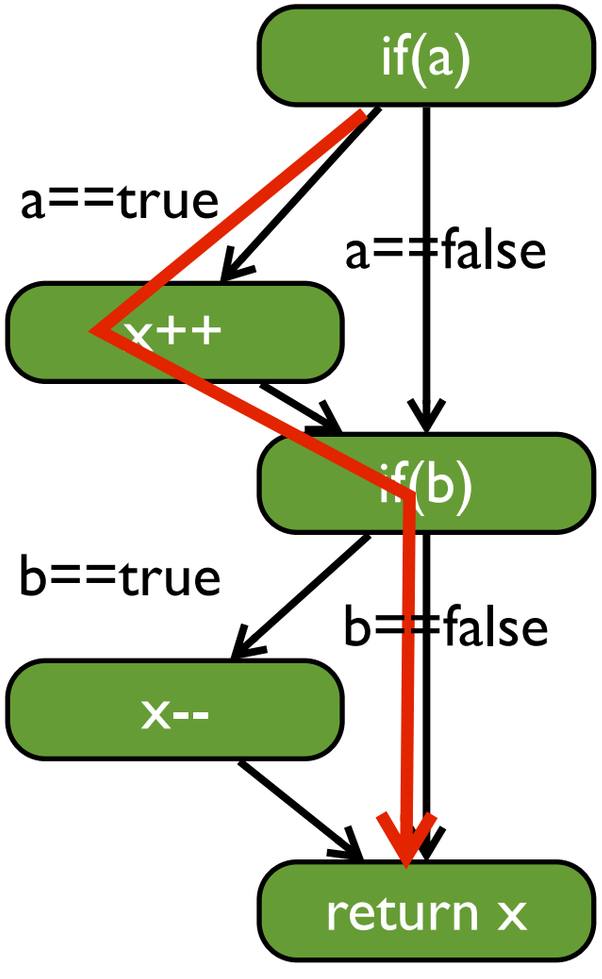
CFG-based Coverage: Branch Coverage

- The percentage of branches covered by the test
  - Consider both false and true branch for each conditional statement



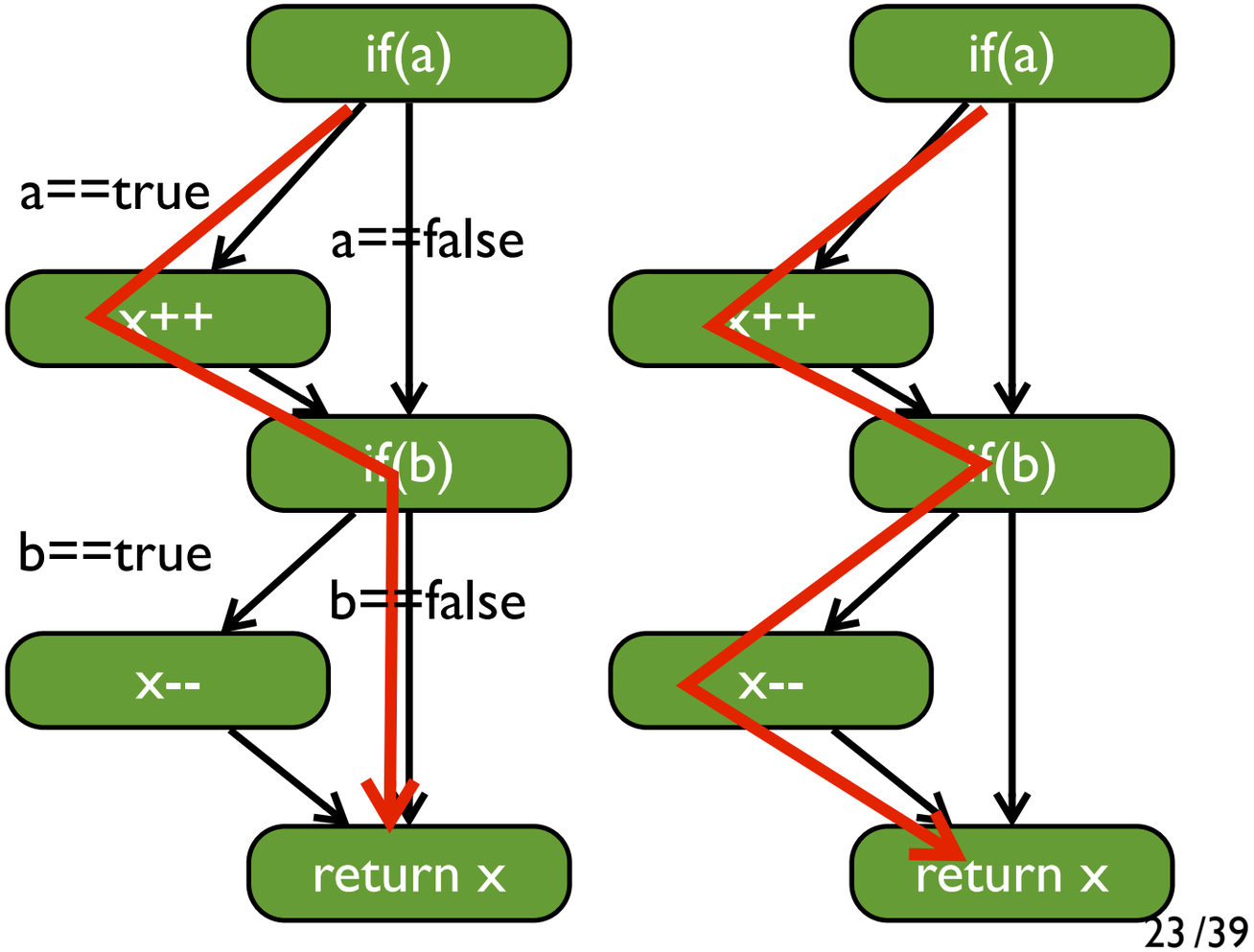
CFG-based Coverage: Path Coverage

- The percentage of paths covered by the test
  - Consider all possible program execution paths



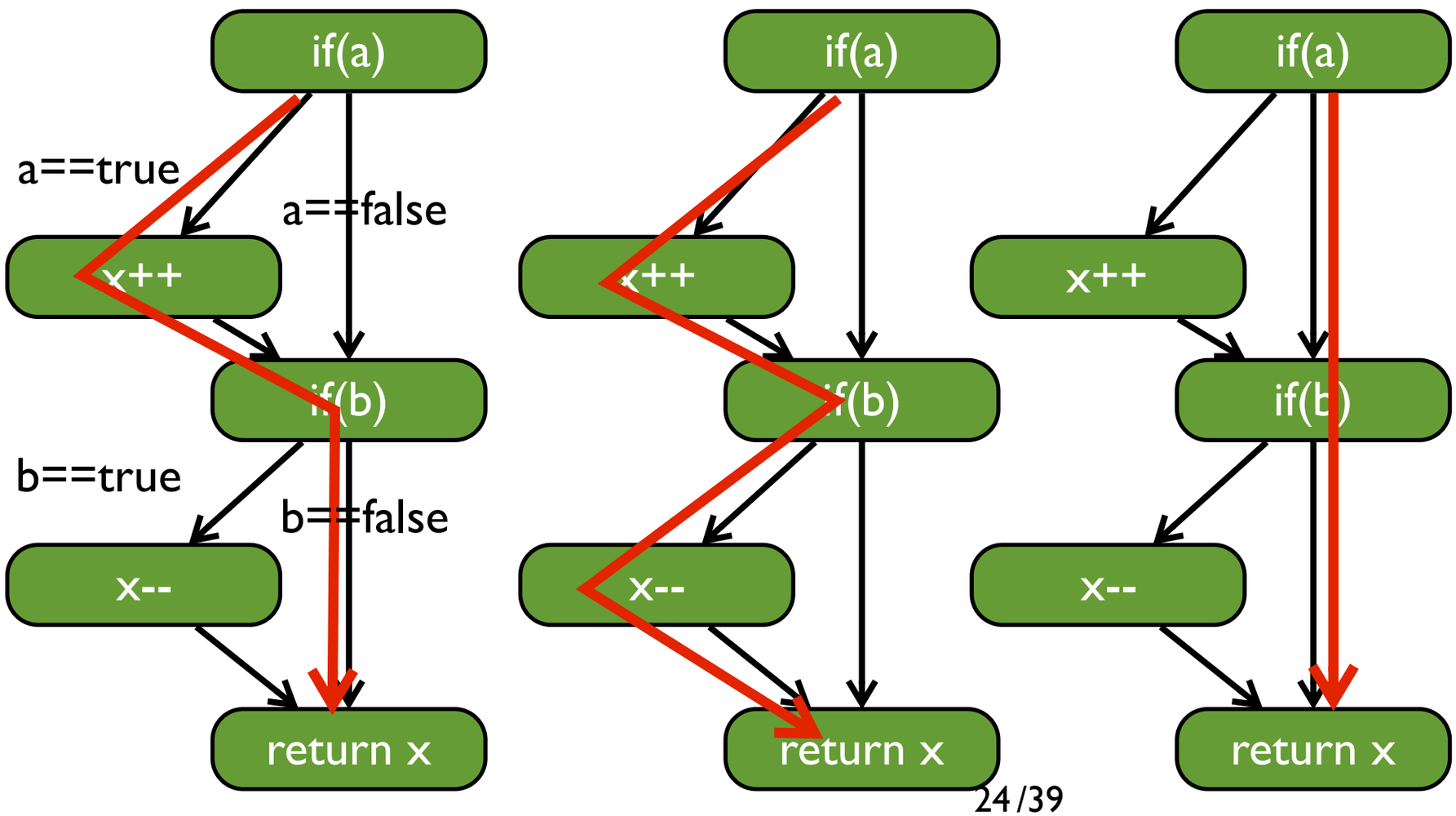
CFG-based Coverage: Path Coverage

- The percentage of paths covered by the test
  - Consider all possible program execution paths



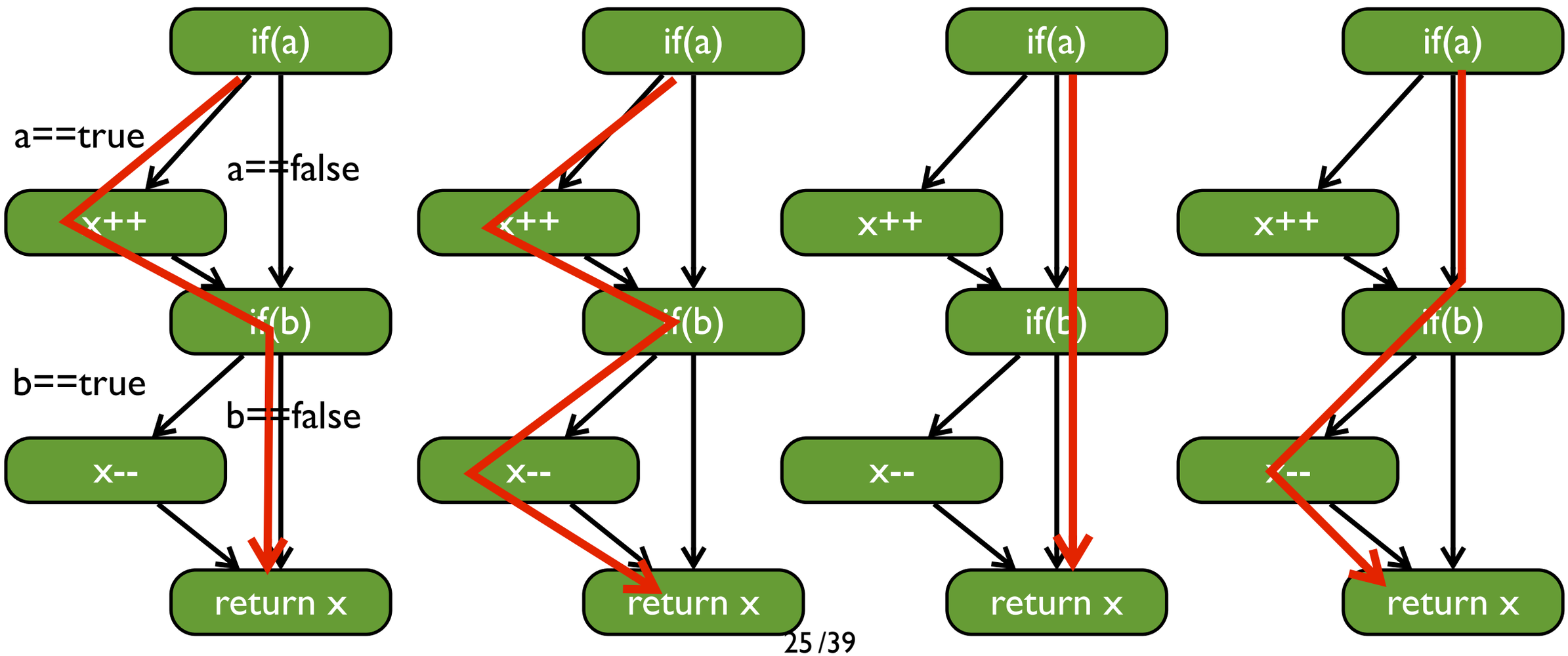
CFG-based Coverage: Path Coverage

- The percentage of paths covered by the test
  - Consider all possible program execution paths



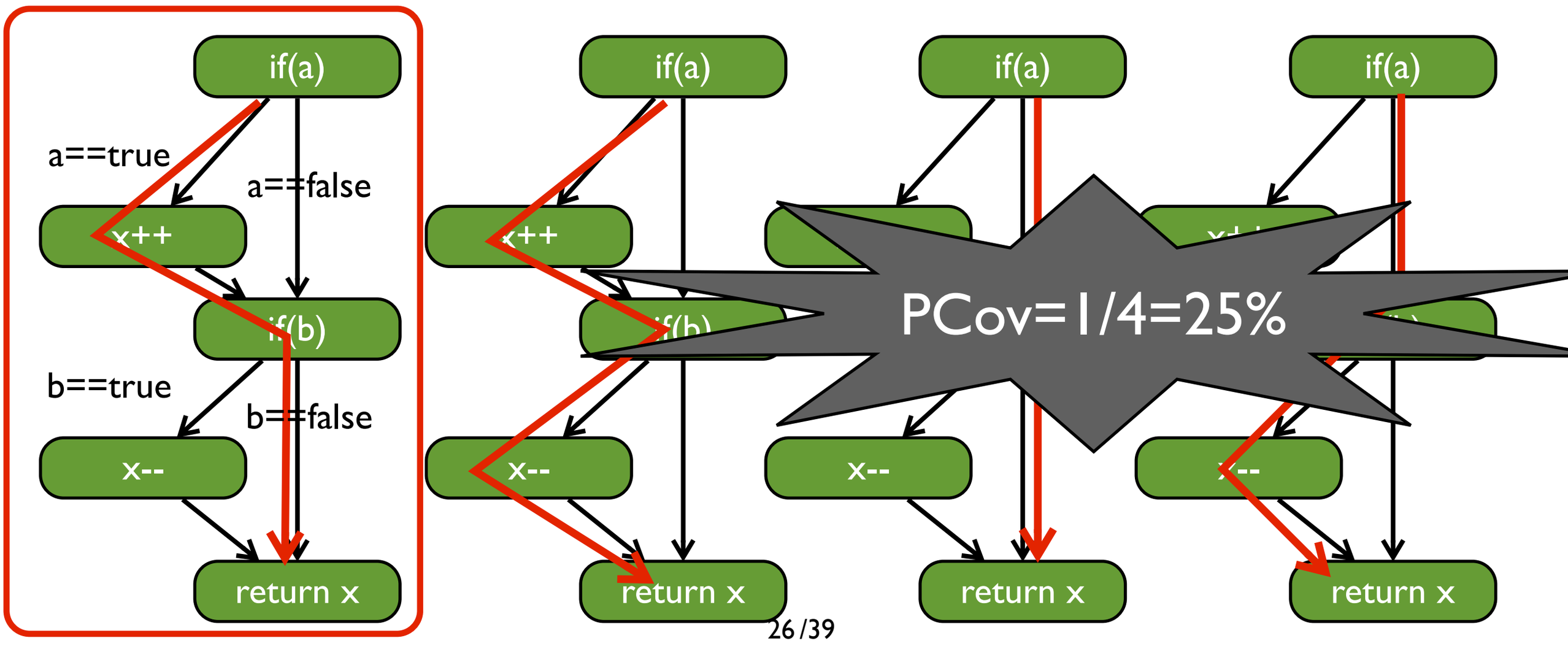
CFG-based Coverage: Path Coverage

- The percentage of paths covered by the test
  - Consider all possible program execution paths



CFG-based Coverage: Path Coverage

- The percentage of paths covered by the test
  - Consider all possible program execution paths



CFG-based Coverage: Comparison

```
public class JUnitStatementCov {
	CFGCoverageExample tester;
	@Before
	public void initialize() {
		tester = new CFGCoverageExample();
	}
	@Test
	public void testCase() {
		assertEquals(0, tester.testMe(0, true, false));
	}
```

Statement coverage: 80%
Branch coverage: 50%
Path coverage: 25%

If we achieve 100% branch coverage, do we get 100% statement coverage for free?

If we achieve 100% path coverage, do we get 100% branch coverage for free?

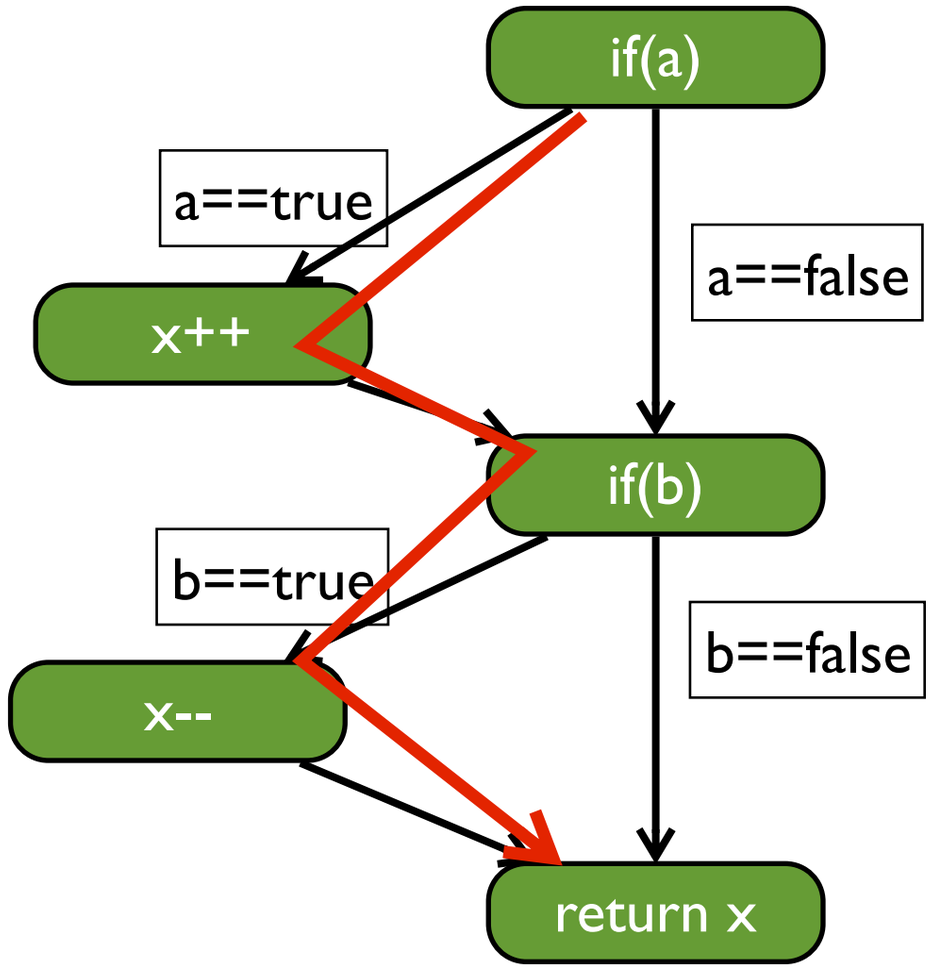
CFG-based Coverage: Statement Coverage VS. Branch Coverage

- If a test suite achieve 100% b-coverage, it must achieve 100% s-coverage
  - The statements not in branches will be covered by any test
  - All other statements are in certain branch
- If a test suite achieve 100% s-coverage, will it achieve 100% b-coverage?

CFG-based Coverage: Statement Coverage VS. Branch Coverage

- If a test suite achieve 100% b-coverage, it must achieve 100% s-coverage
  - The statements not in branches will be covered by any test
  - All other statements are in certain branch
- If a test suite achieve 100% s-coverage, will it achieve 100% b-coverage?

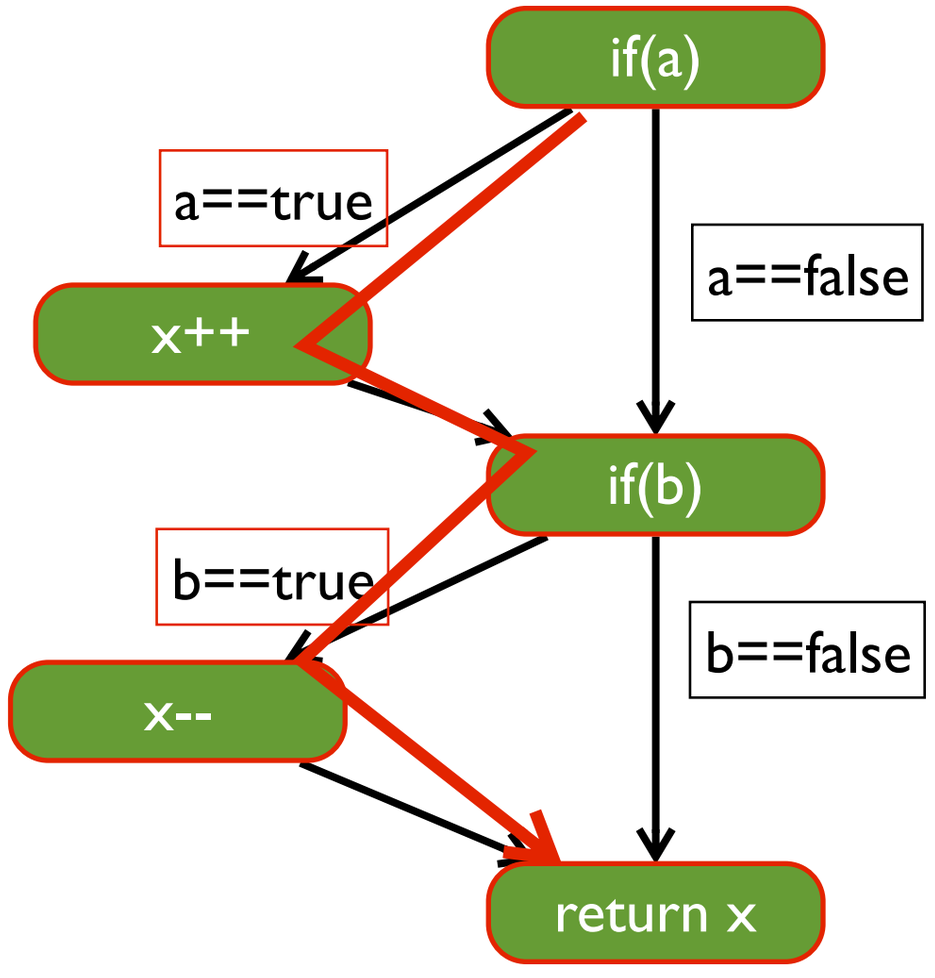
x=0 a=true b=true



CFG-based Coverage: Statement Coverage VS. Branch Coverage

- If a test suite achieve 100% b-coverage, it must achieve 100% s-coverage
  - The statements not in branches will be covered by any test
  - All other statements are in certain branch
- If a test suite achieve 100% s-coverage, will it achieve 100% b-coverage?

x=0 a=true b=true



Branch coverage strictly subsumes statement coverage

CFG-based Coverage: Branch Coverage VS. Path Coverage

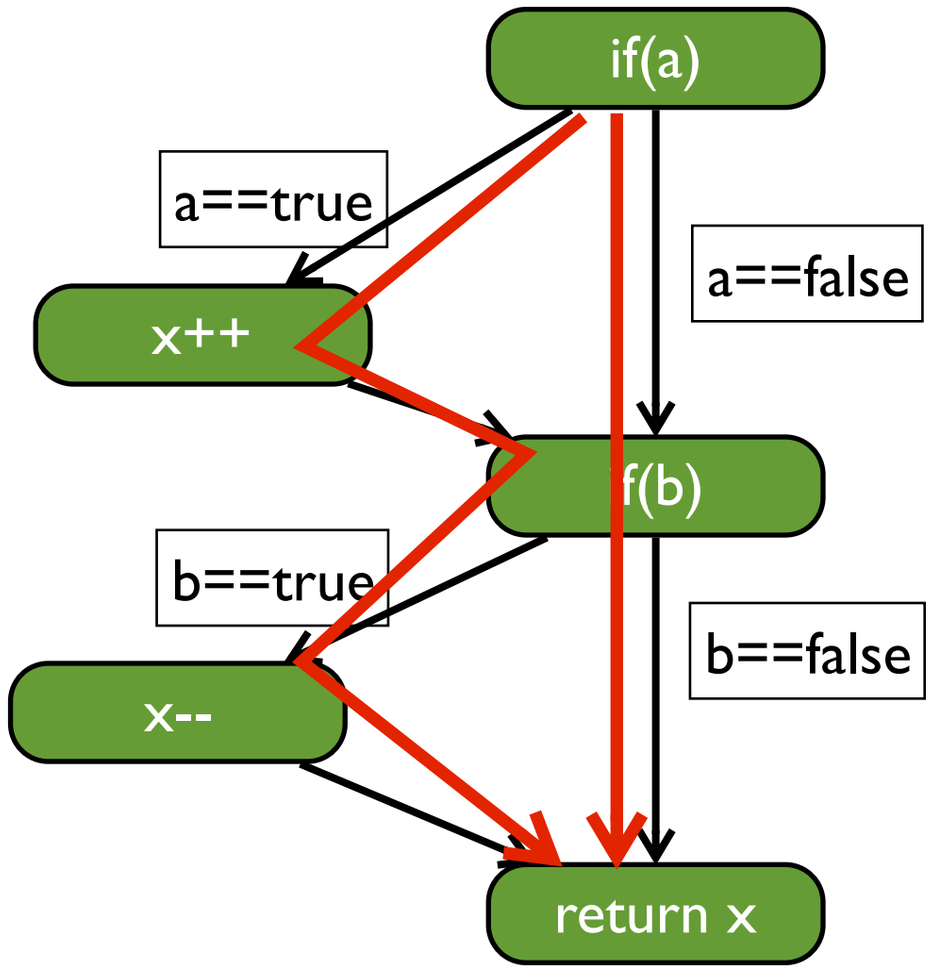
- If a test suite achieve 100% p-coverage, it must achieve 100% b-coverage
  - All the branch combinations have been covered indicate all branches are covered
- If a test suite achieve 100% b-coverage, will it achieve 100% p-coverage?

CFG-based Coverage: Branch Coverage VS. Path Coverage

x=0 a=true b=true

x=0 a=false b=false

- If a test suite achieve 100% p-coverage, it must achieve 100% b-coverage
  - All the branch combinations have been covered indicate all branches are covered
- If a test suite achieve 100% b-coverage, will it achieve 100% p-coverage?

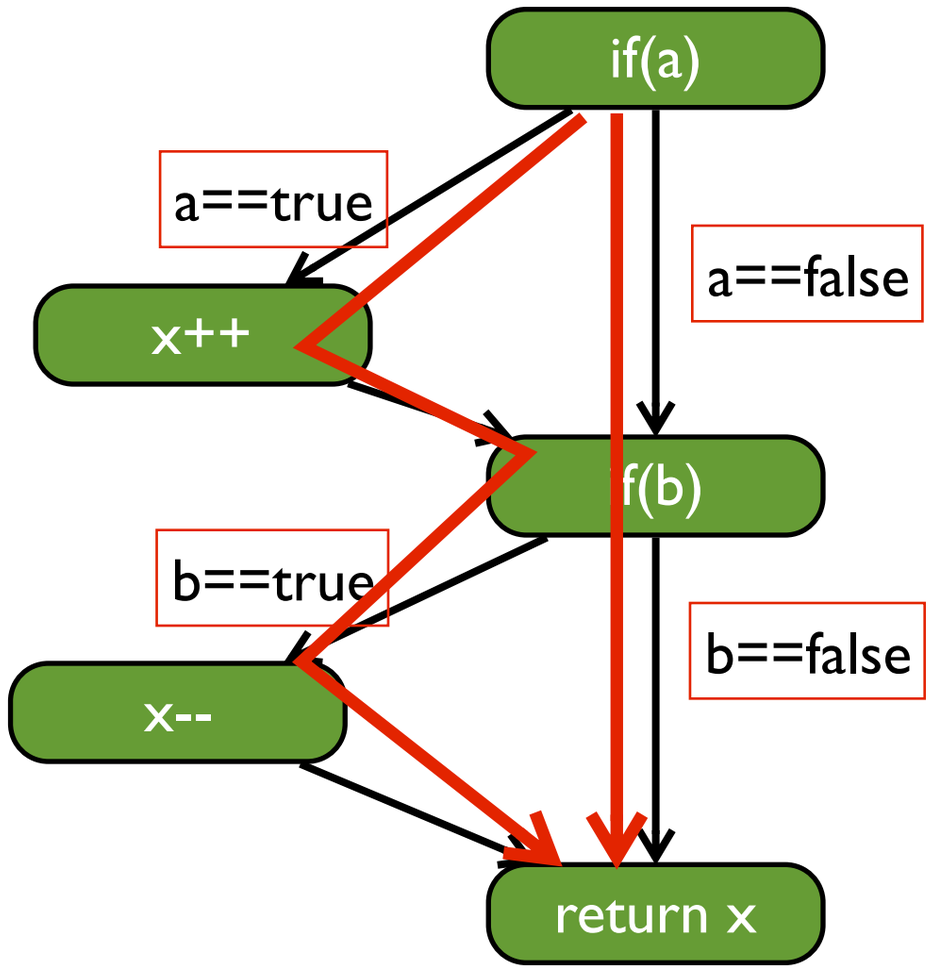


CFG-based Coverage: Branch Coverage VS. Path Coverage

x=0 a=true b=true

x=0 a=false b=false

- If a test suite achieve 100% p-coverage, it must achieve 100% b-coverage
  - All the branch combinations have been covered indicate all branches are covered
- If a test suite achieve 100% b-coverage, will it achieve 100% p-coverage?



Path coverage strictly subsumes branch coverage

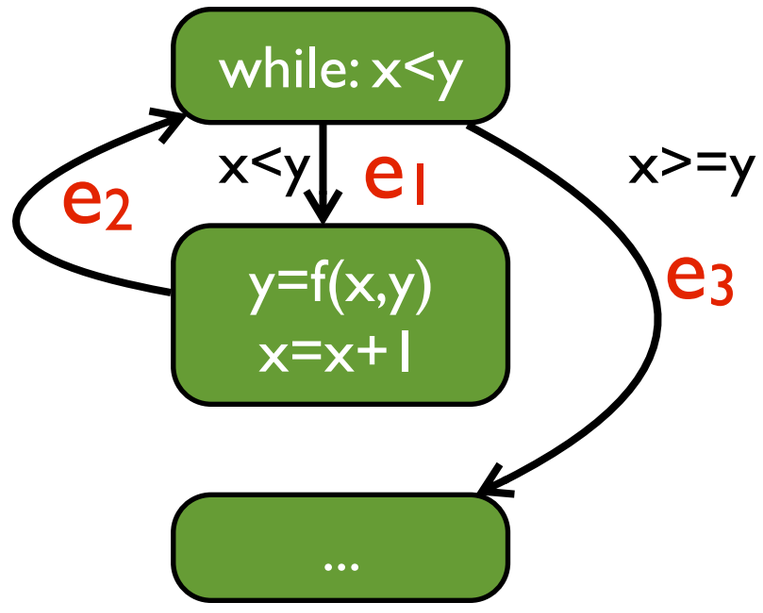
CFG-based Coverage: Comparison Summary

Path coverage
strictly subsumes branch coverage
strictly subsumes statement coverage

Hard to achieve:
p-coverage > b-coverage > s-coverage

Should we just use path coverage?

```
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
...
```



Possible Paths

e_3

$e_1e_2e_3$

$e_1e_2e_1e_2e_3$

$e_1e_2e_1e_2e_1e_2e_3$

...

Path coverage can be infeasible for real-world programs

CFG-based Coverage: Effectiveness

- About 65% of all bugs can be caught in unit testing
- Unit testing is dominated by control-flow testing methods
- Statement and branch testing dominates control-flow testing

CFG-based Coverage: Limitation

- 100% coverage of some aspect is never a guarantee of bug-free software

Test: assertEquals(1, sum(1,0))

```
public int sum(int x, int y){
        return x-y; //should be x+y
    }
```

Statement coverage: 100%

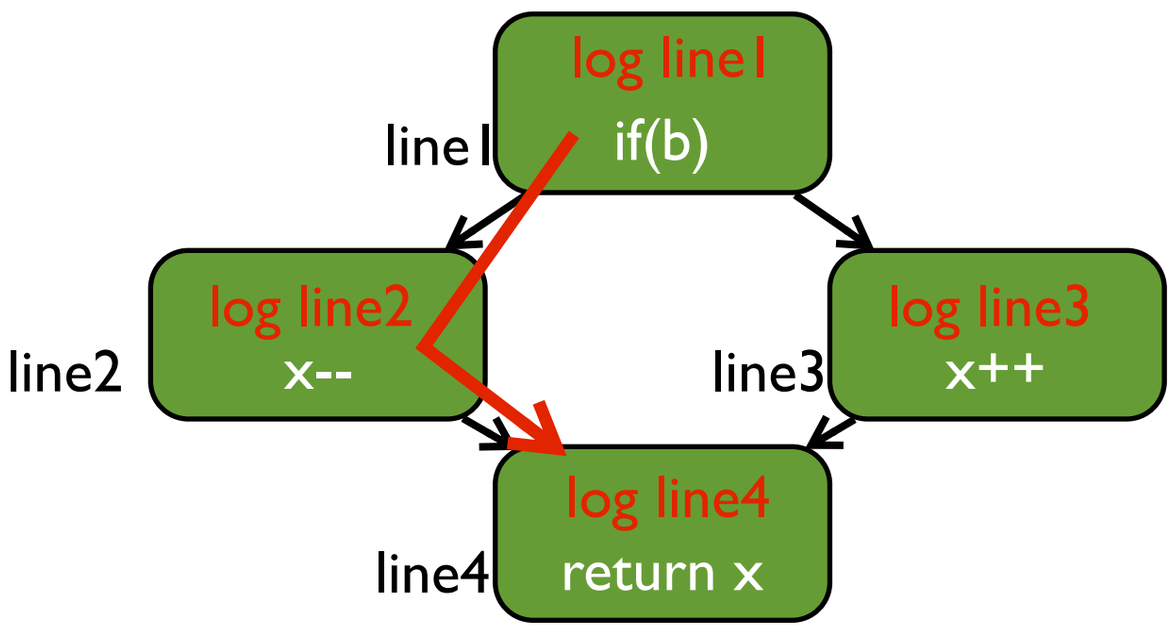
Branch coverage: 100%

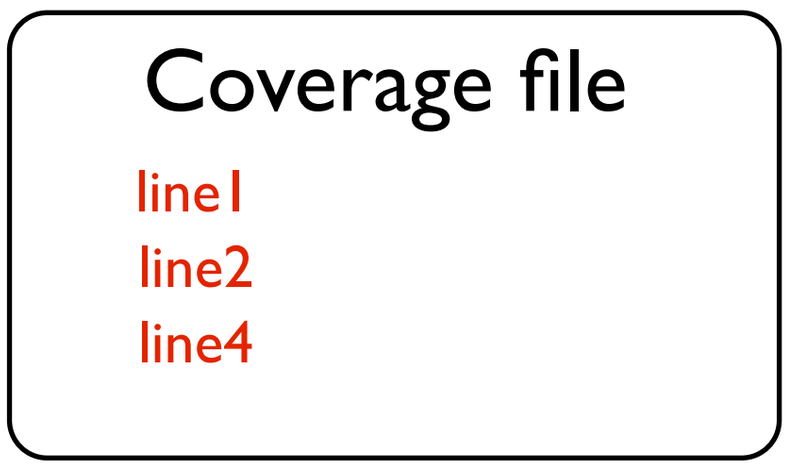
Path coverage: 100%

Failed to detect the bug...

Coverage Collection: Mechanism

- The source code is instrumented (source/binary)
  - Log code that writes to a trace file is inserted in every branch, statement etc.
- When the instrumented code is executed, the coverage info will be written to trace file





Coverage Collection: Tool Supports

- Emma: http://emma.sourceforge.net/
- EclEmma: http://www.eclemma.org/installation.html/
- Cobertura: http://cobertura.github.io/cobertura/
- Clover: https://www.atlassian.com/software/clover/overview
- JCov: https://wiki.openjdk.java.net/display/CodeTools/jcov
- JaCoCo: http://www.eclemma.org/jacoco/