

CS/CE/SE 6367

Software Testing, Validation and Verification

Lecture 3
Code Coverage (I)



Last class

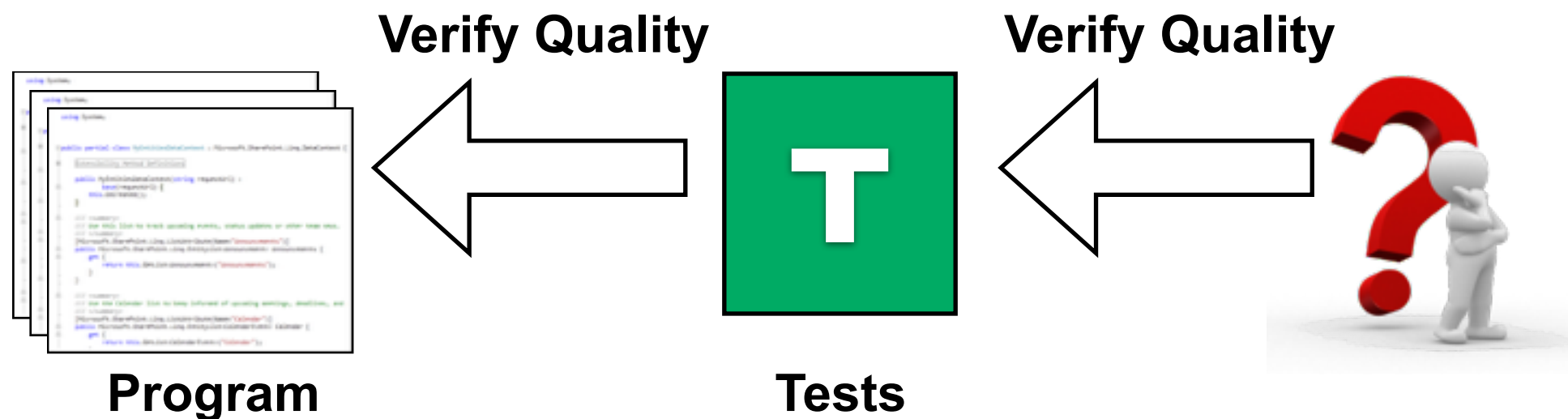
- Software Testing
 - Concepts
 - Granularity
 - Unit Testing
- JUnit

This class

- Code coverage
 - Control-flow coverage
 - Statement coverage
 - Branch coverage
 - Path coverage
 - Coverage Collection Tools
 - EcEmma

Who will test the tests?

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

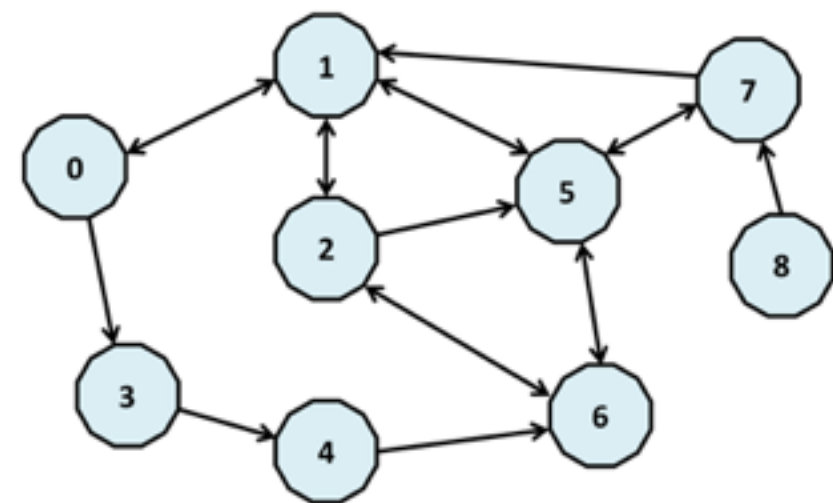
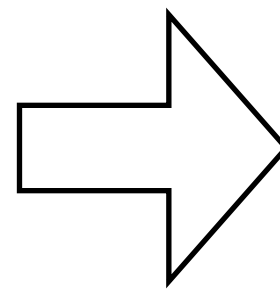


How to measure code coverage



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

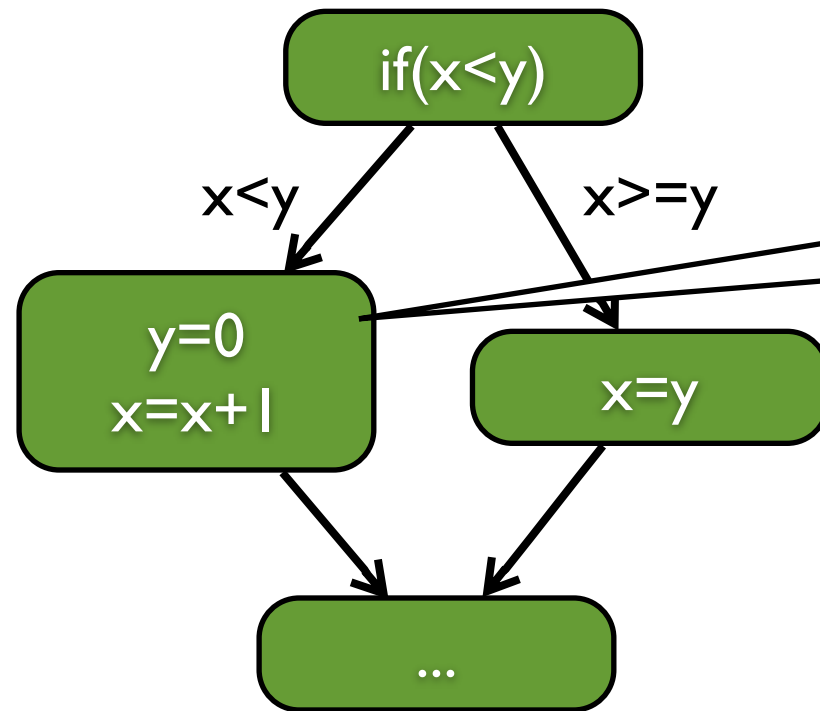


Control Flow Graphs

- A CFG models all executions of a program by describing control structures
 - Node : Sequences of statements (basic block)
 - Basic Block : A sequence of statements with only one entry point and only one exit point (no branches)
 - Edge : Transfers of control

CFG : The if Statement

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



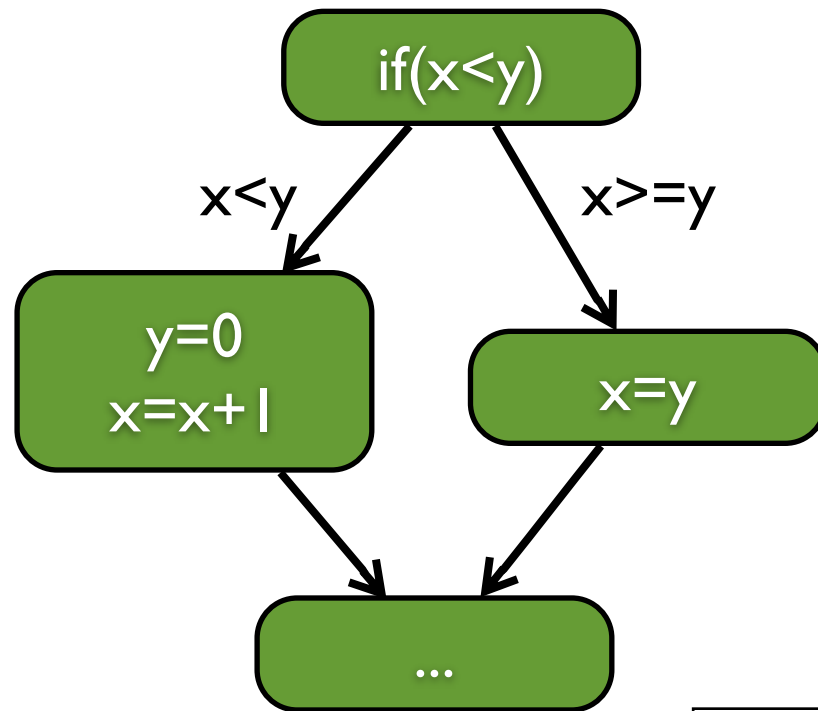
the two statements can be in the same nodes because there is no branch between them

CFG : The if Statement

```

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

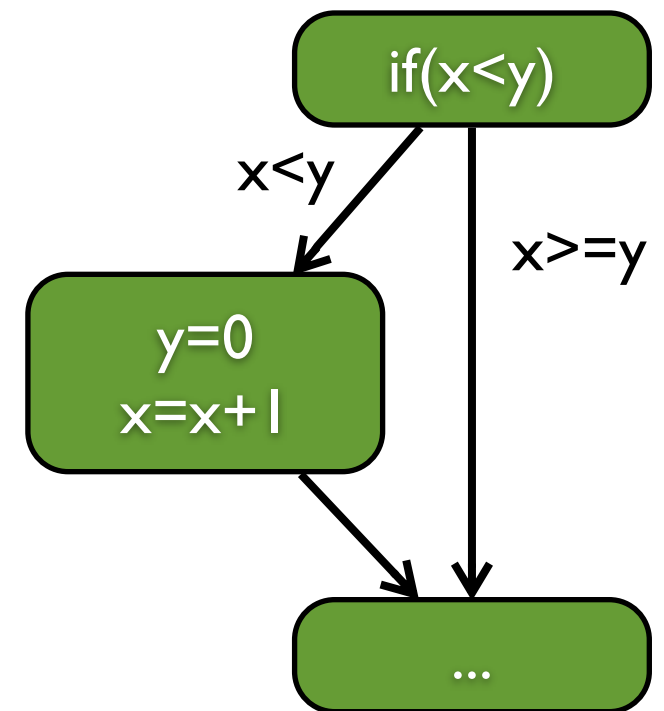
```



```

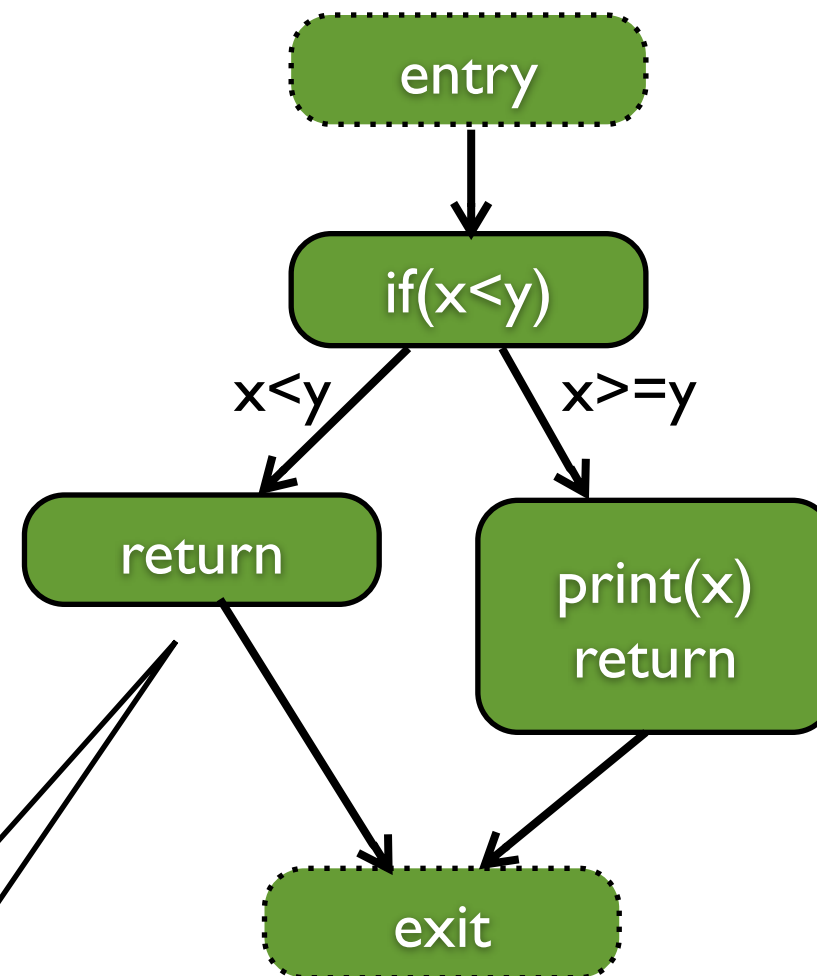
if (x < y)
{
  y = 0;
  x = x + 1;
}
...

```



CFG : The Dummy Nodes

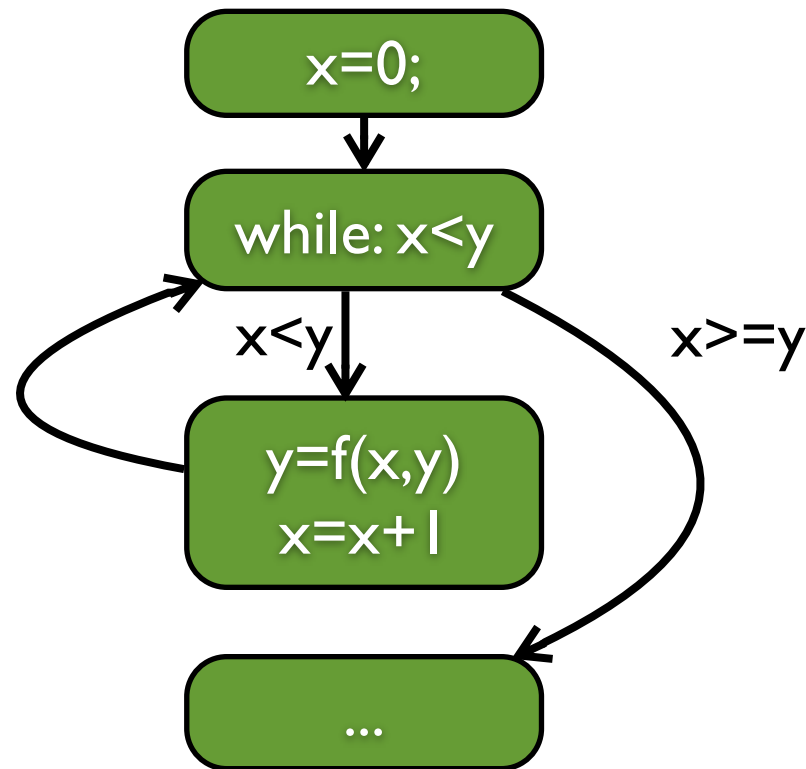
```
if (x < y)
{
    return;
}
print (x);
return;
```



Some program may have multiple exit nodes!

CFG : while and for Loops

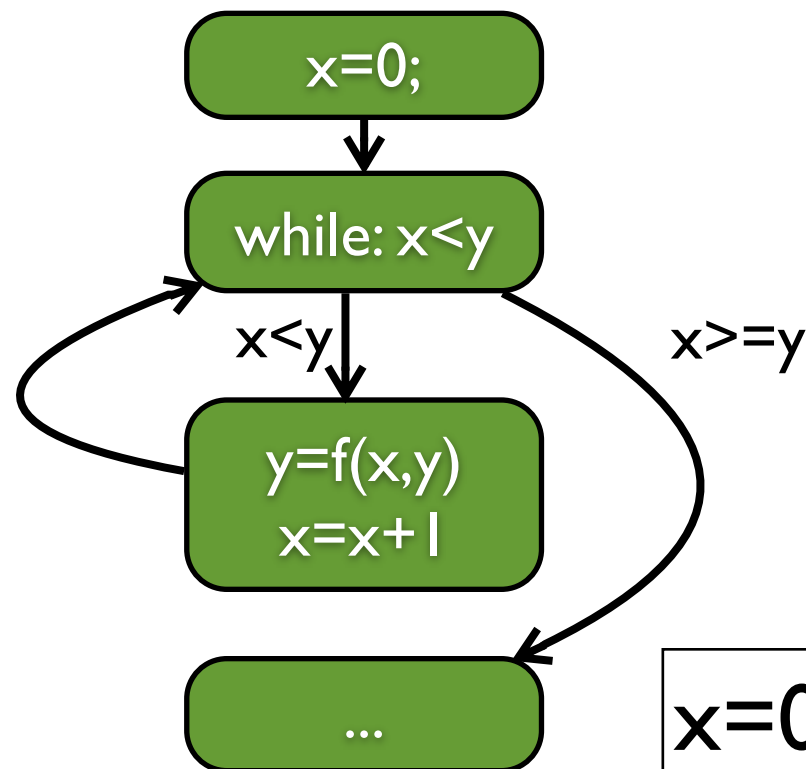
```
x=0;
while (x < y)
{
  y = f (x, y);
  x = x + 1;
}
...
```



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

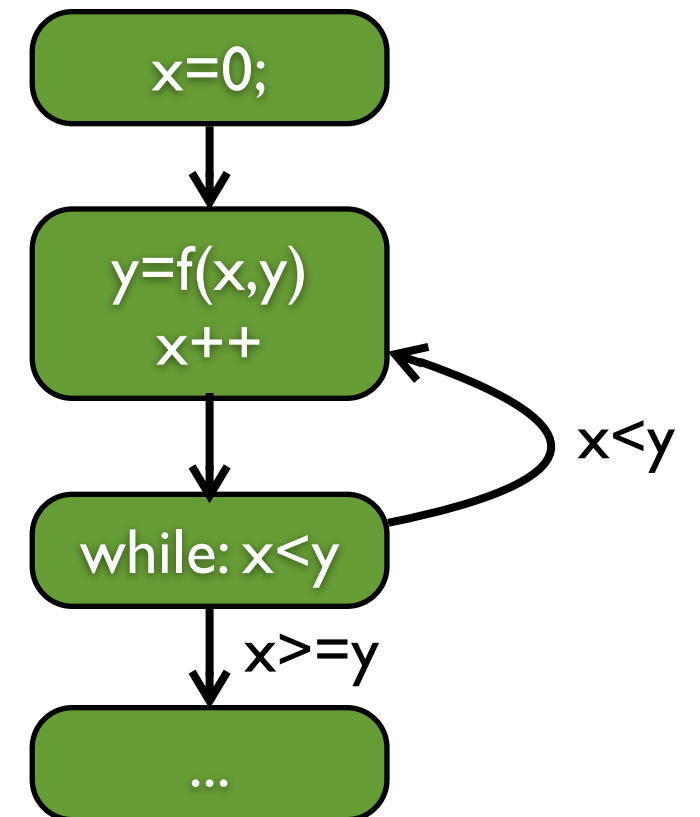
CFG : while and for Loops

```
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)
...
```

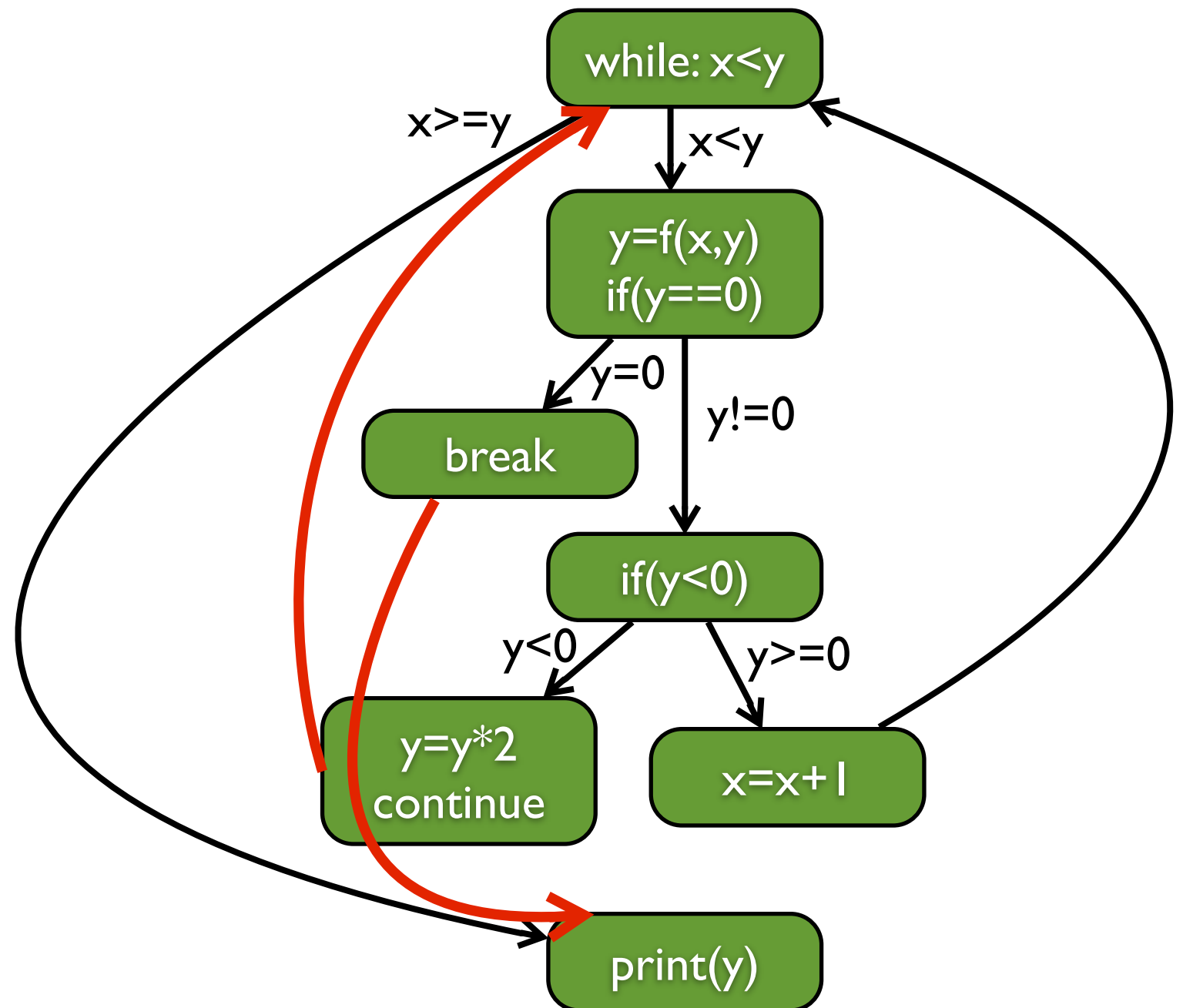


CFG: break and continue

```

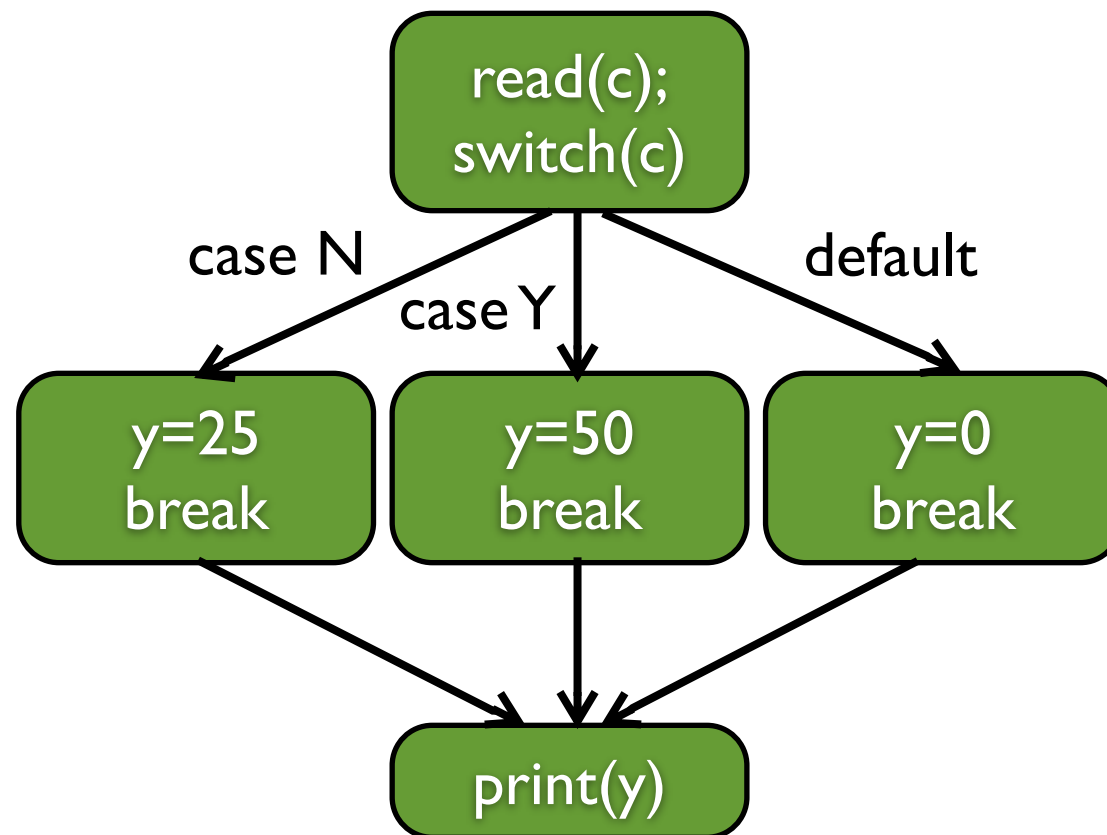
while (x < y)
{
  y = f(x, y);
  if (y == 0) {
    break;
  } else if (y < 0) {
    y = y*2;
    continue;
  }
  x = x + 1;
}
print (y);

```



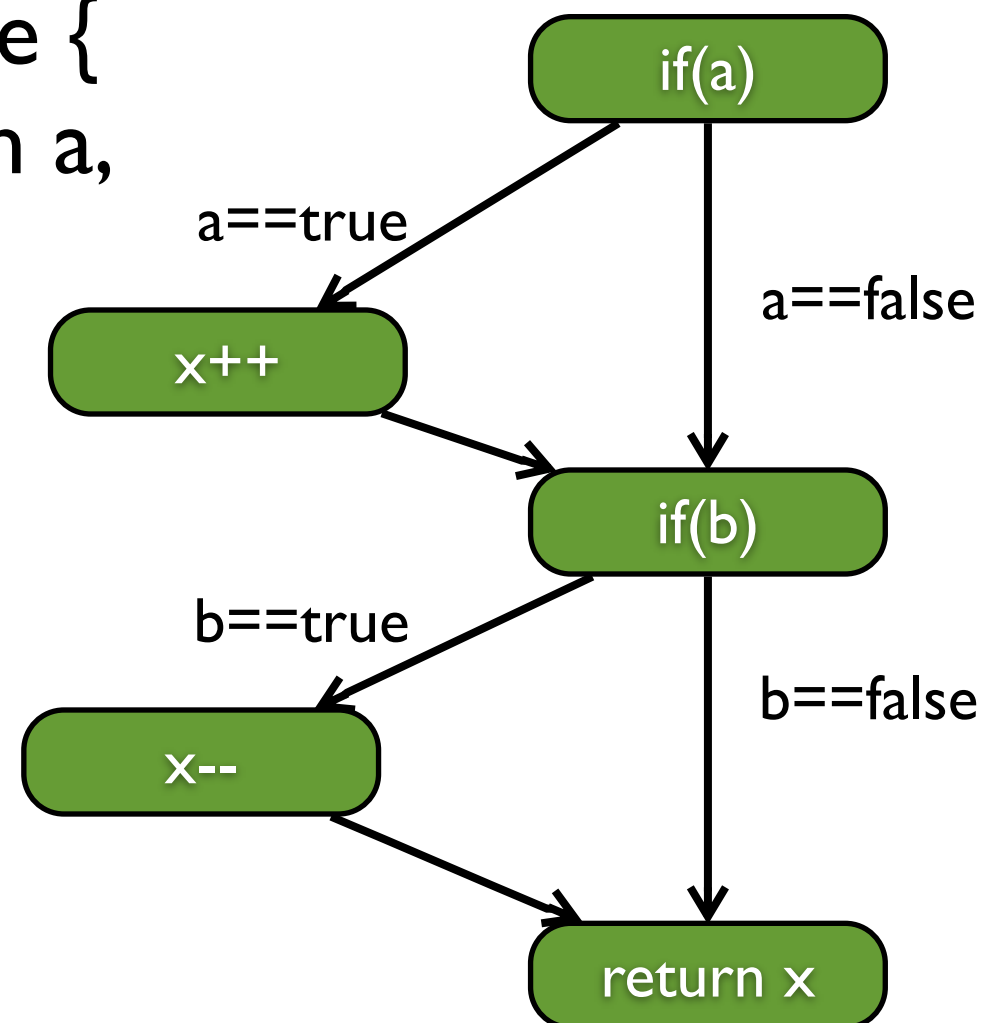
CFG: switch

```
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));  
    }  
}
```

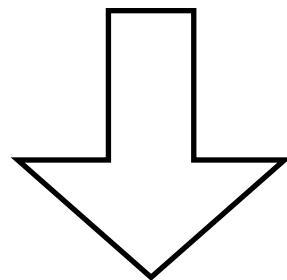


How good is it??

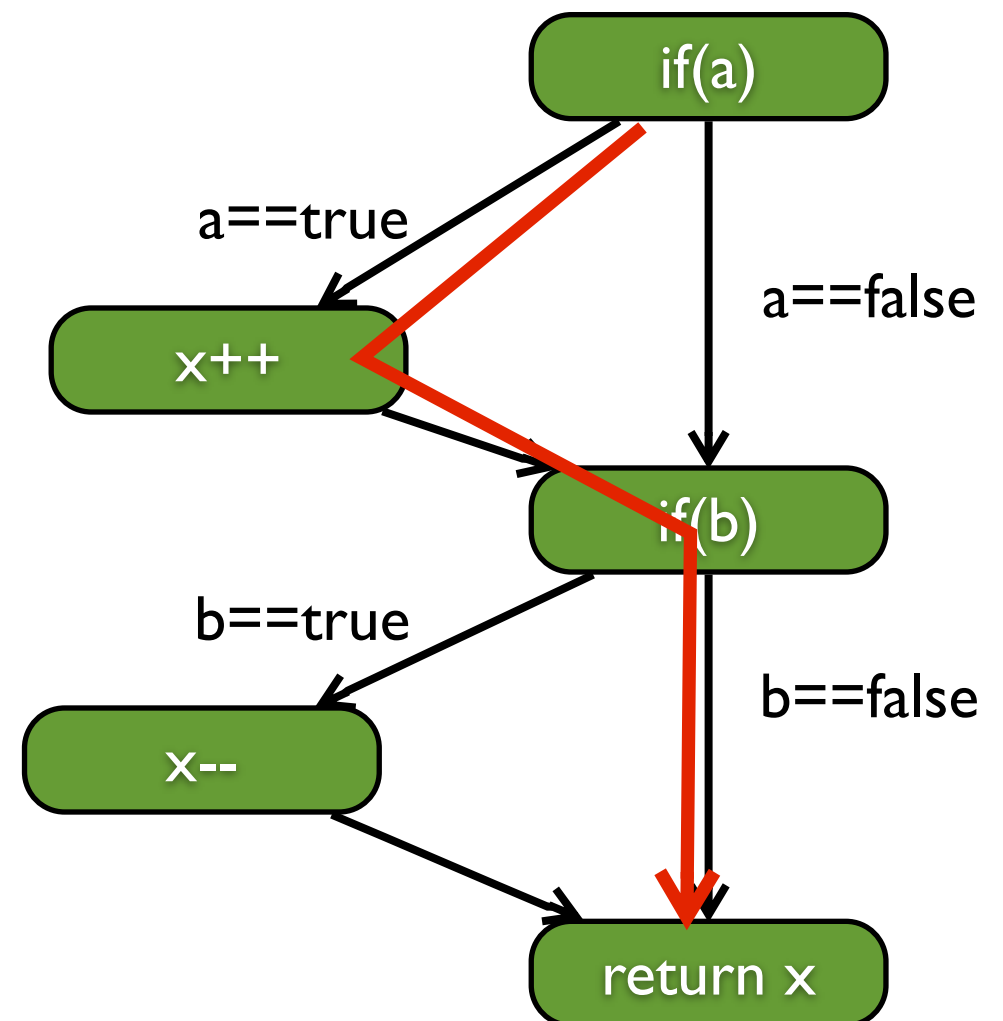
CFG-based Coverage: Statement Coverage

- The percentage of statements covered by the test

tester.testMe(0, true, false)



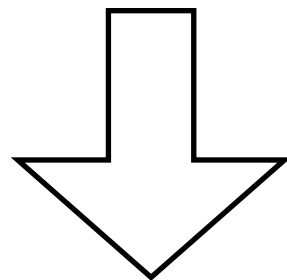
x=0 a=true b=false



CFG-based Coverage: Statement Coverage

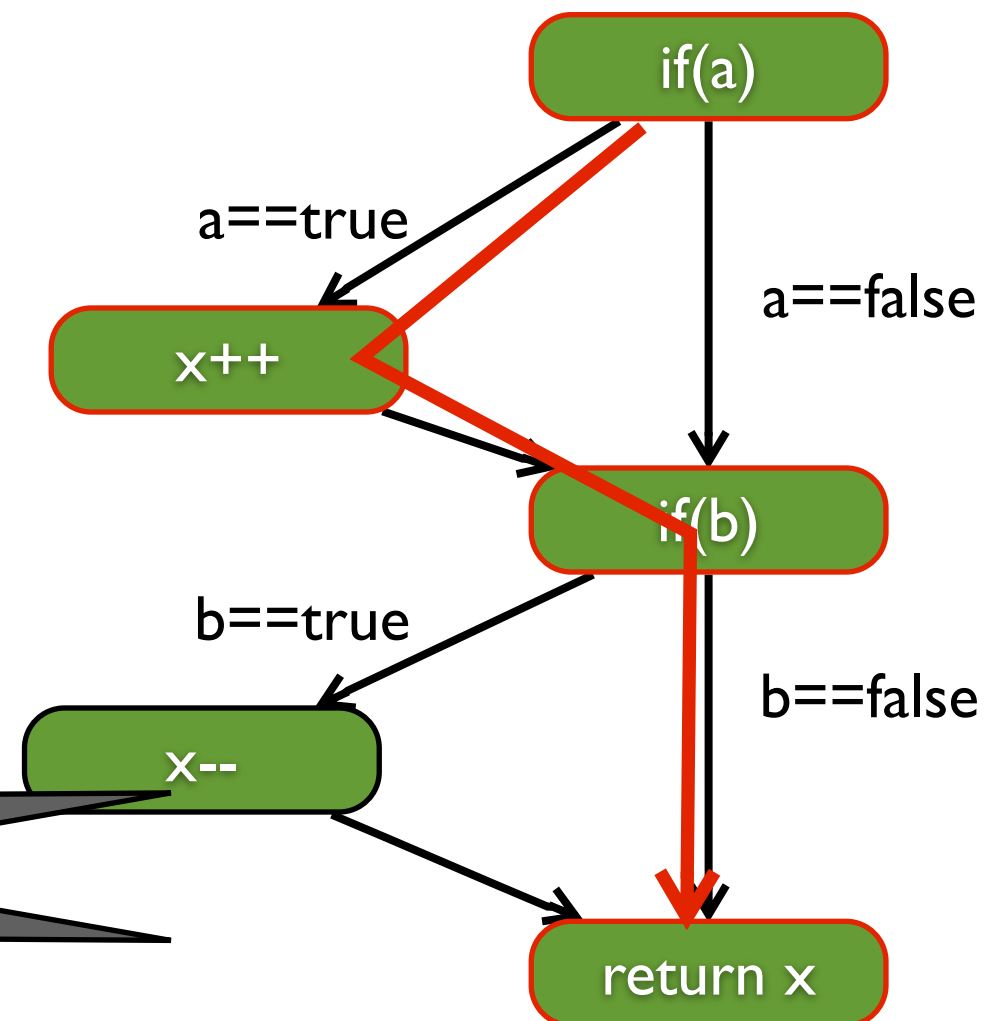
- The percentage of statements covered by the test

tester.testMe(0, true, false)



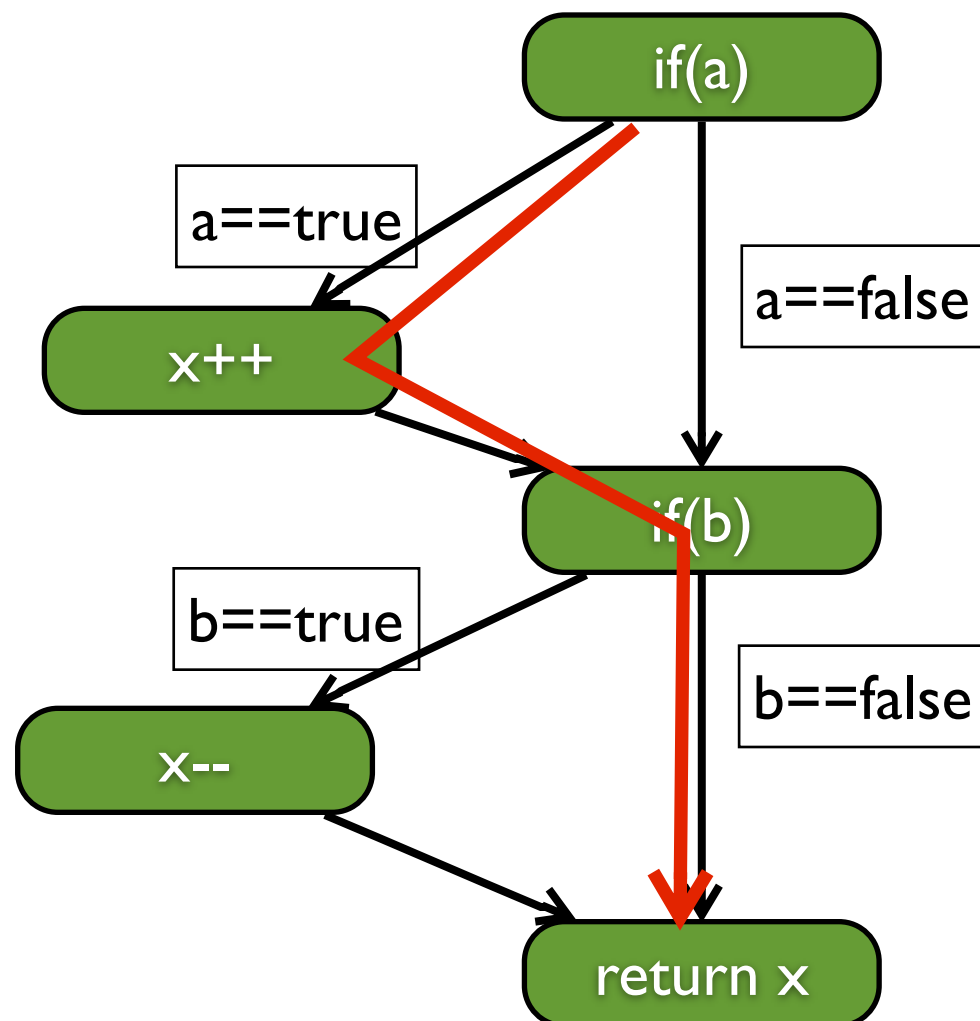
x=0 a=true b=false

SCov=4/5=80%



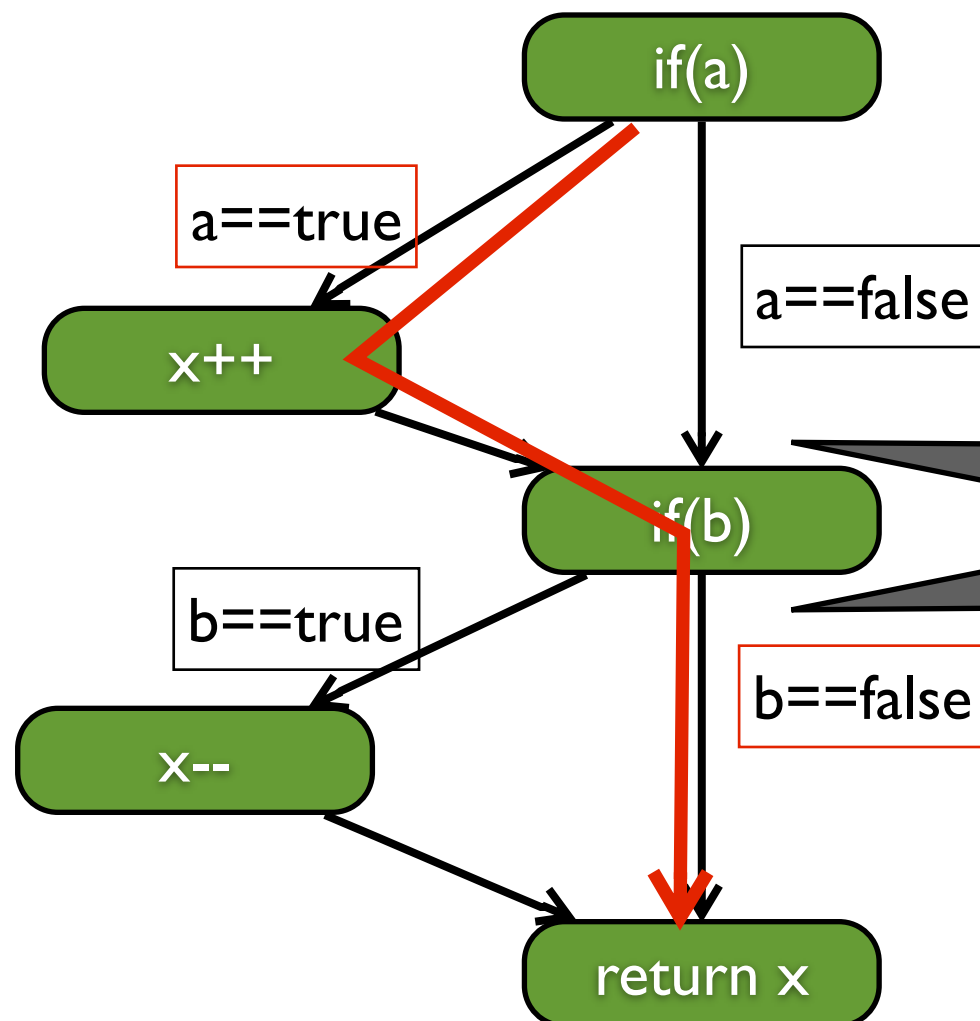
CFG-based Coverage: Branch Coverage

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



CFG-based Coverage: Branch Coverage

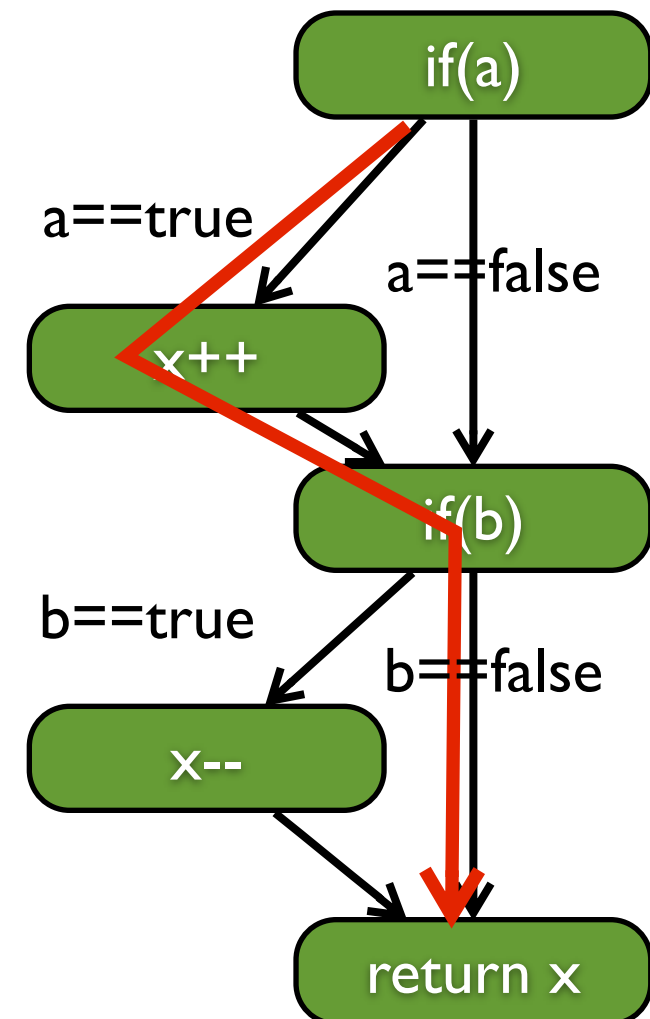
- The percentage of branches covered by the test
 - Consider both false and true branch for each conditional statement



$$\text{BCov} = 2/4 = 50\%$$

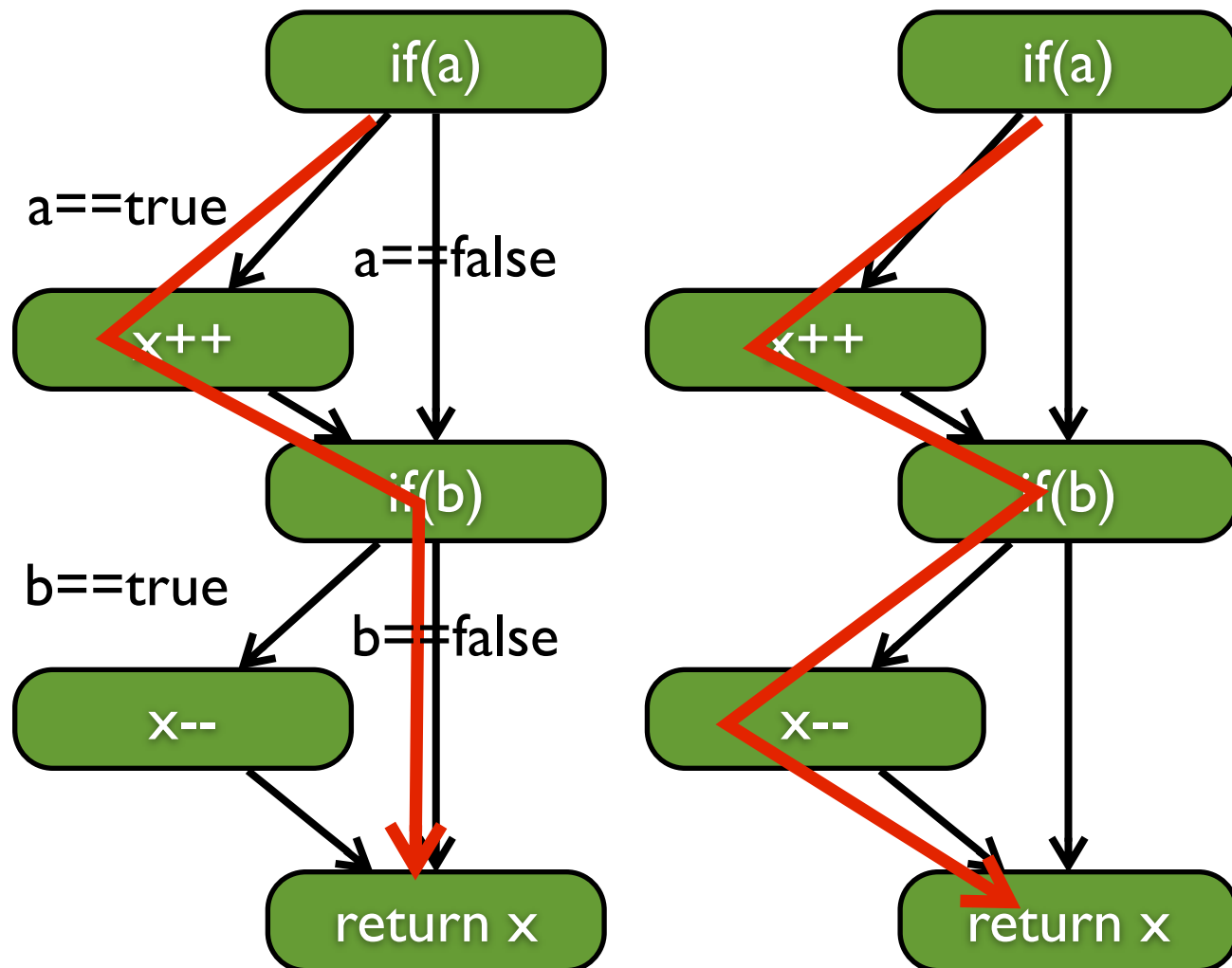
CFG-based Coverage: Path Coverage

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



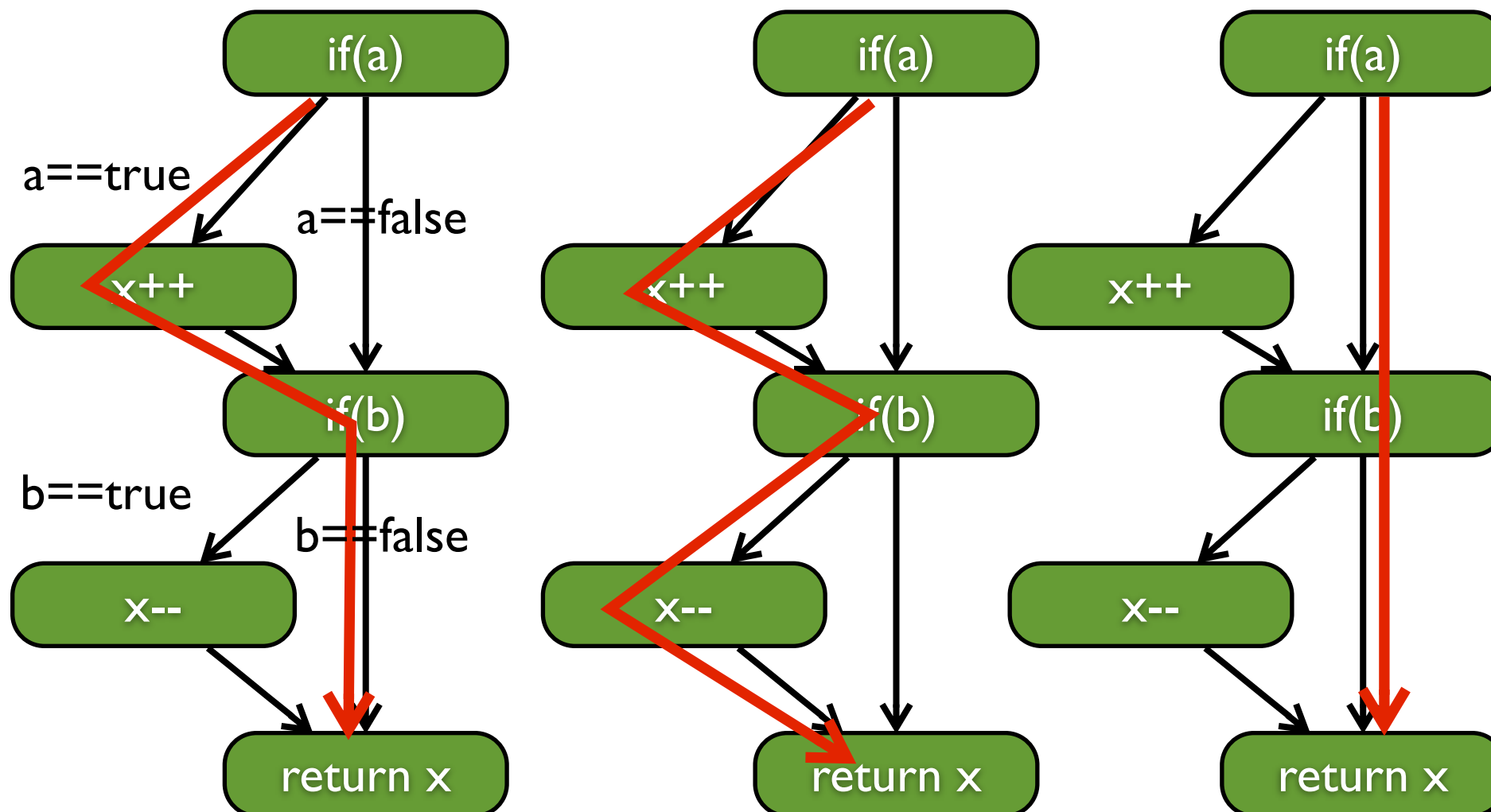
CFG-based Coverage: Path Coverage

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



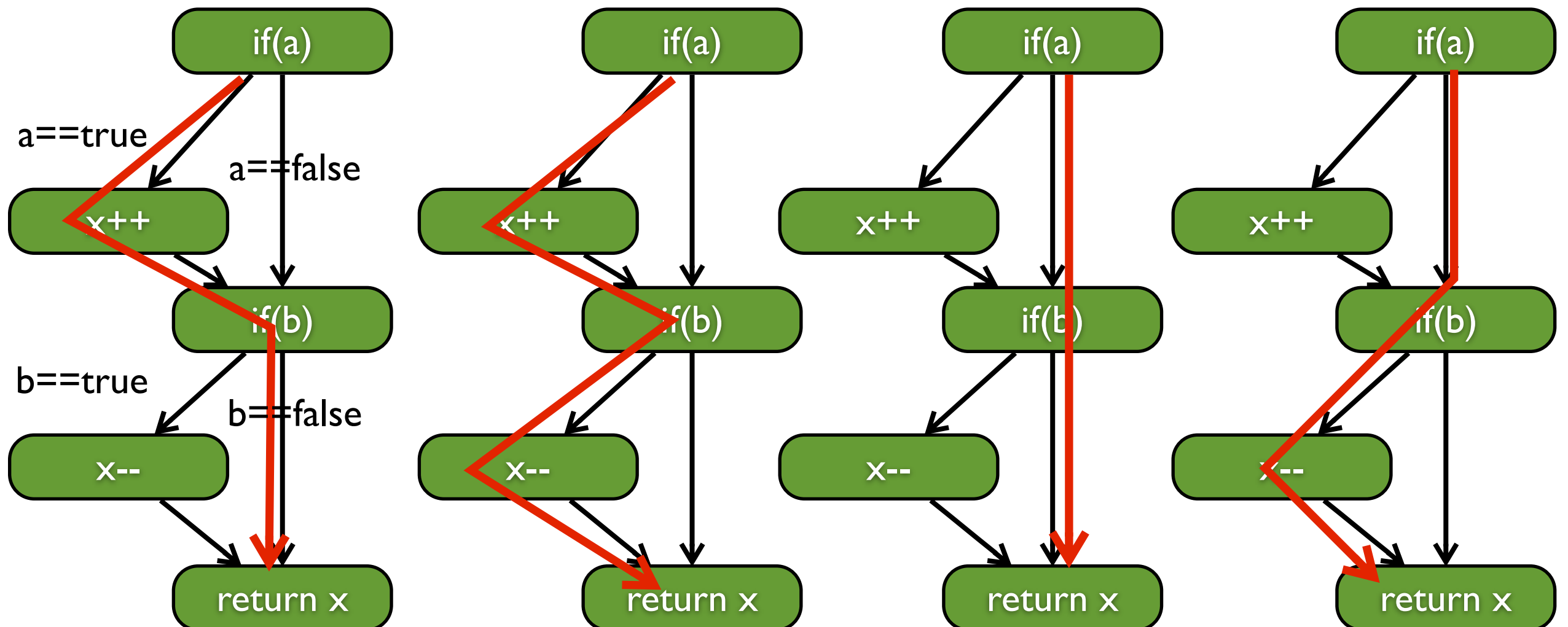
CFG-based Coverage: Path Coverage

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



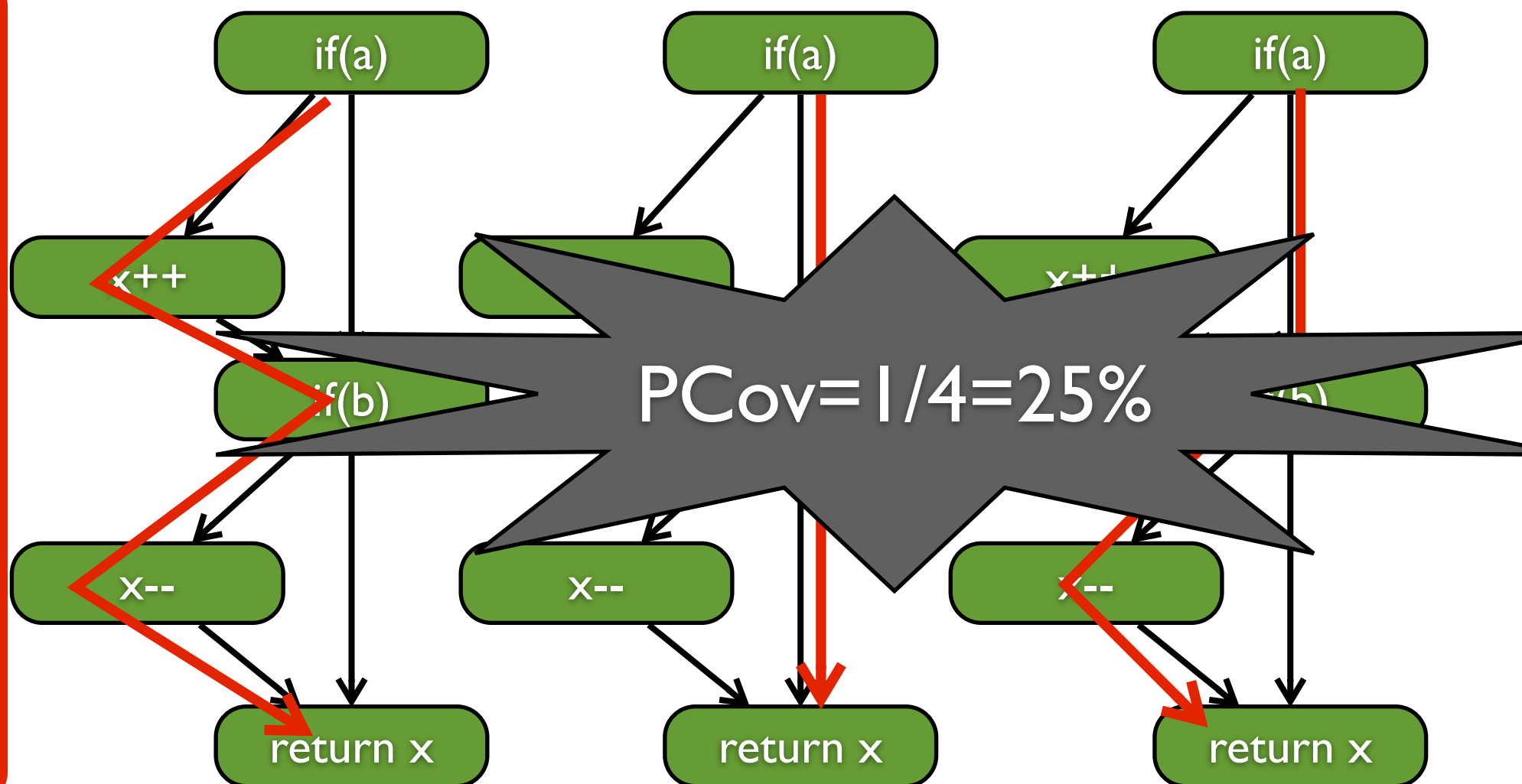
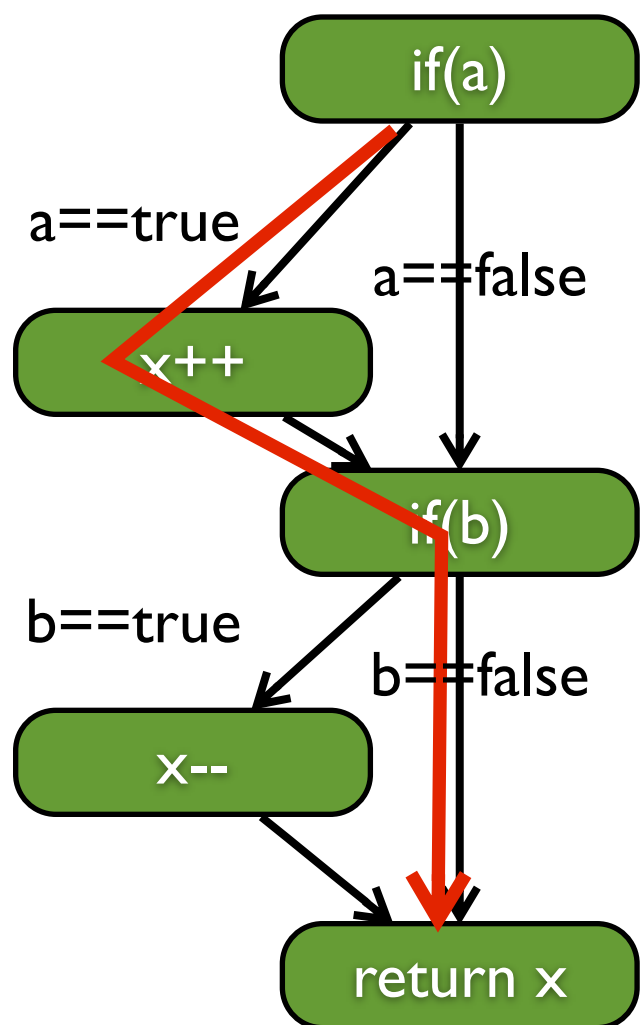
CFG-based Coverage: Path Coverage

- 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?

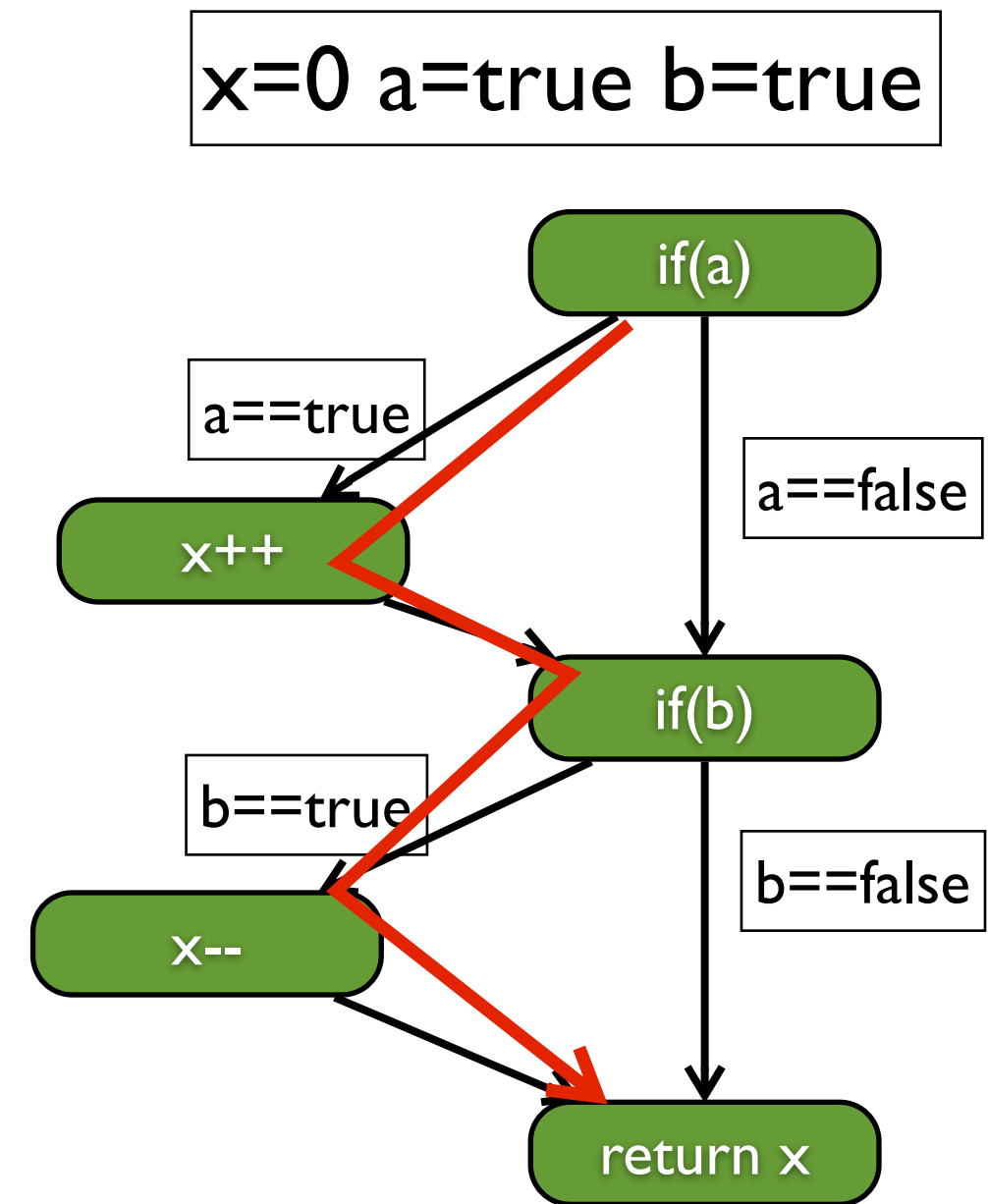
If we achieve 100% path coverage, do we get 100% branch coverage for free?

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?

Statement Coverage VS. Branch Coverage

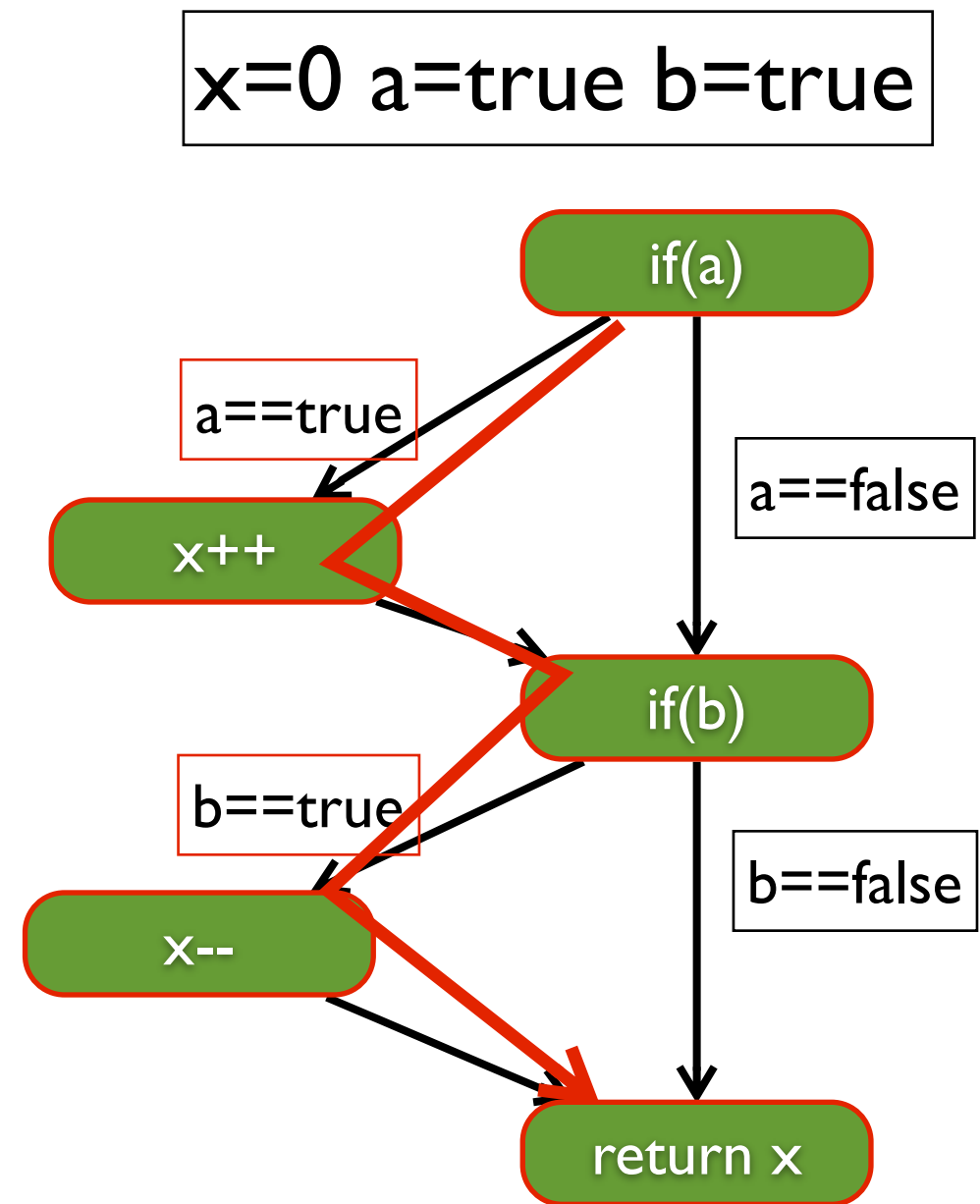
- If a test suite achieve 100% b-coverage, it must achieve 100% s-coverage
- The statements not in branches will be covered by any test
- All other statements are in certain branch
- If a test suite achieve 100% s-coverage, will it achieve 100% b-coverage?



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?

Branch coverage strictly subsumes statement coverage



Branch Coverage VS. Path Coverage

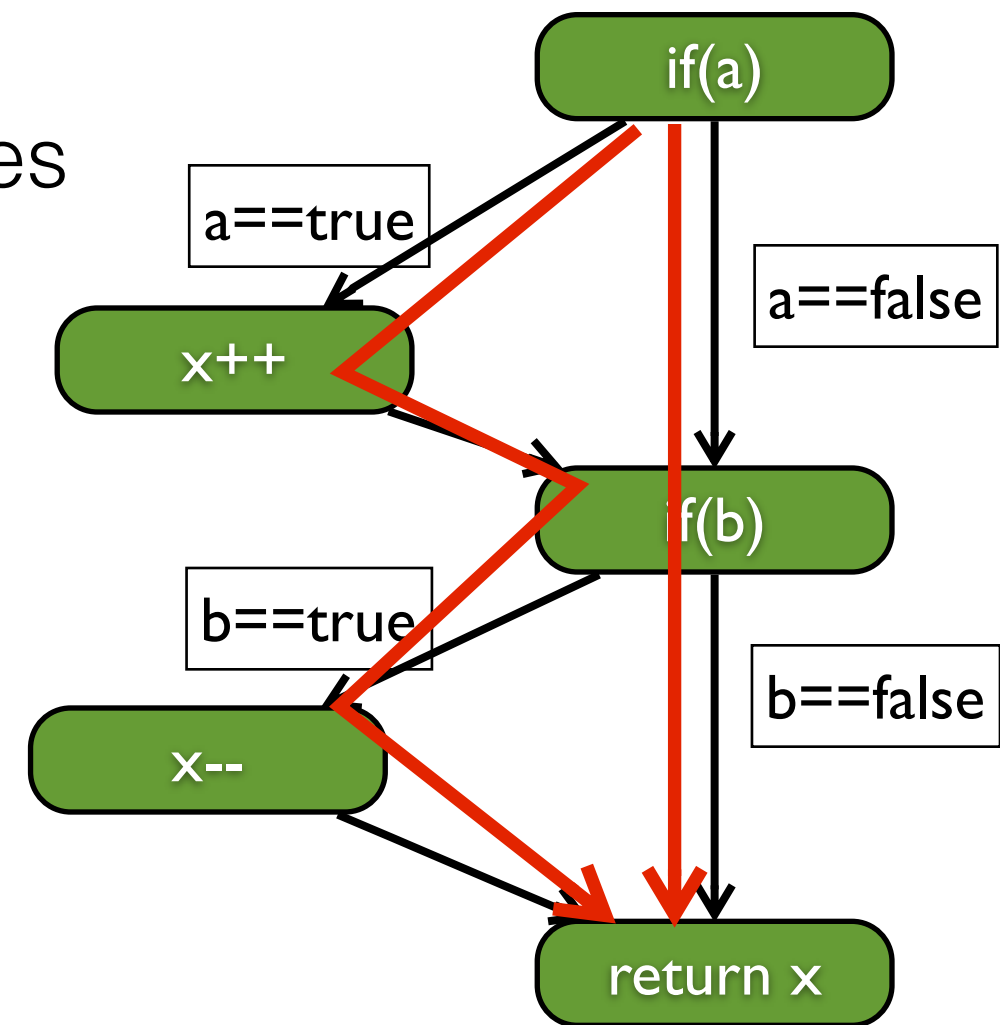
- If a test suite achieve 100% p-coverage, it must achieve 100% b-coverage
 - All the branch combinations have been covered indicate all branches are covered
- If a test suite achieve 100% b-coverage, will it achieve 100% p-coverage?

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?

x=0 a=true b=true

x=0 a=false b=false

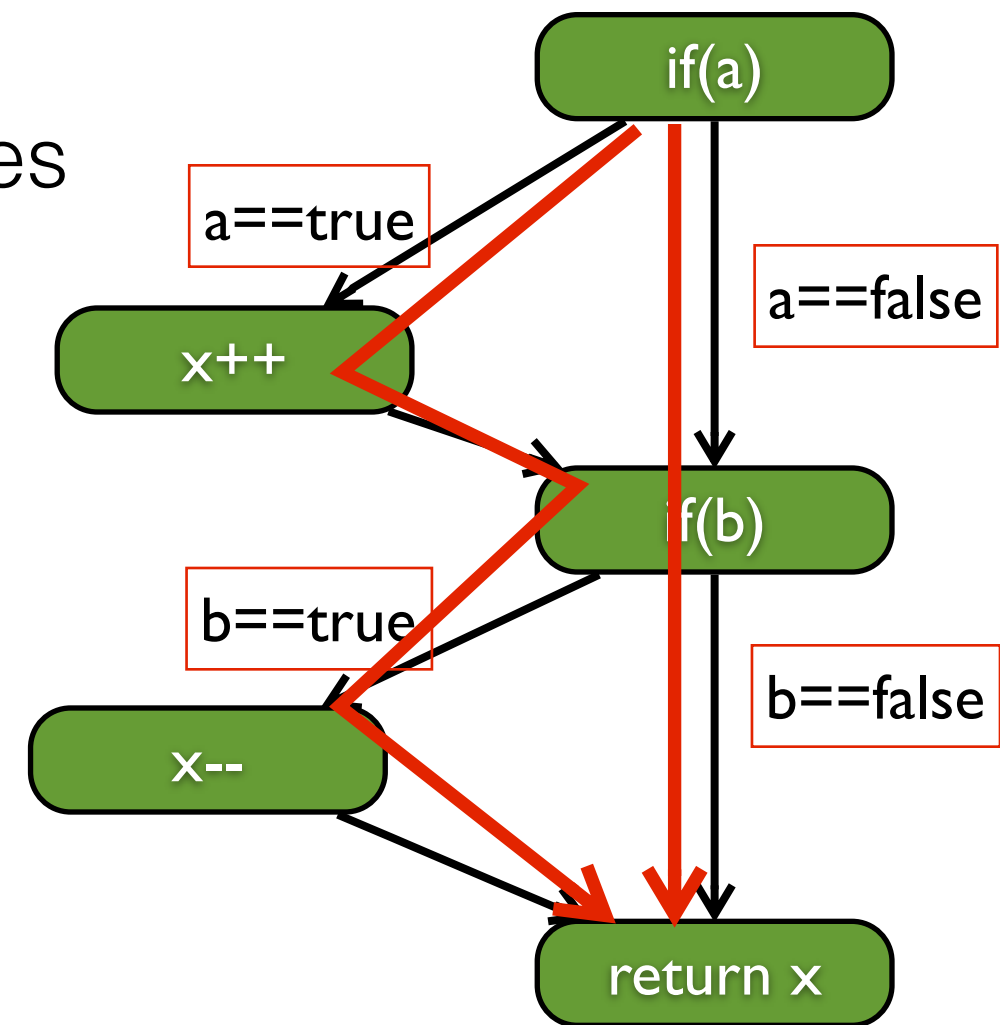


Branch Coverage VS. Path Coverage

- 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?

x=0 a=true b=true

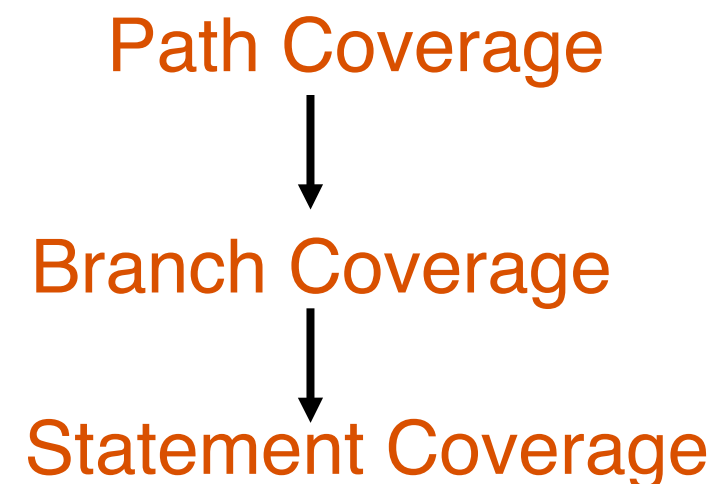
x=0 a=false b=false



Path coverage strictly subsumes branch coverage

CFG-based Coverage: Comparison Summary

Path coverage
strictly **subsumes** branch coverage
strictly **subsumes** statement coverage

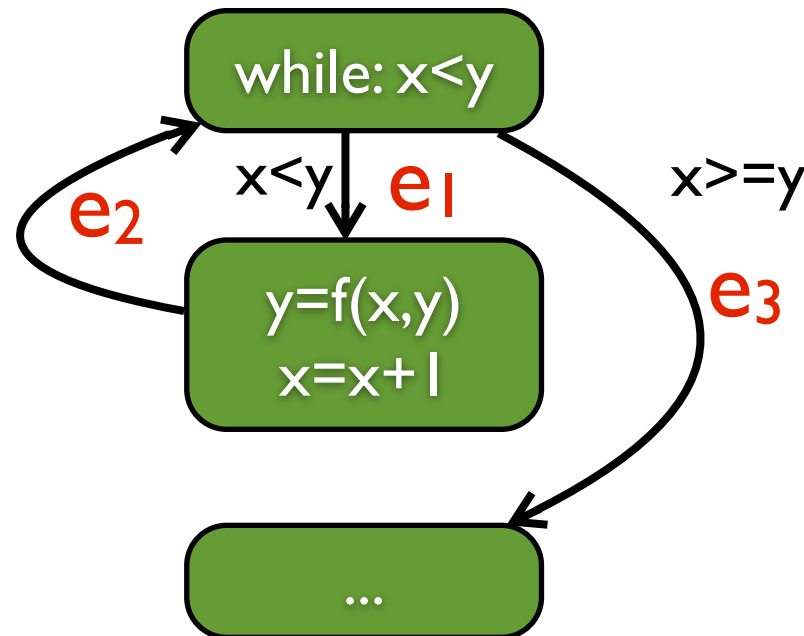


Should we just use path coverage?

```

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

```



Possible Paths

e₃

e₁e₂e₃

e₁e₂e₁e₂e₃

e₁e₂e₁e₂e₁e₂e₃

...

Path coverage can be infeasible
for real-world programs

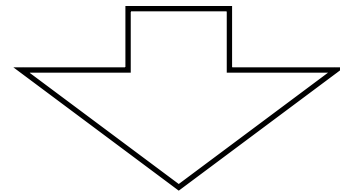
CFG-based Coverage: Effectiveness

- 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  
}
```



Failed to detect the bug...

Statement coverage: 100%

Branch coverage: 100%

Path coverage: 100%

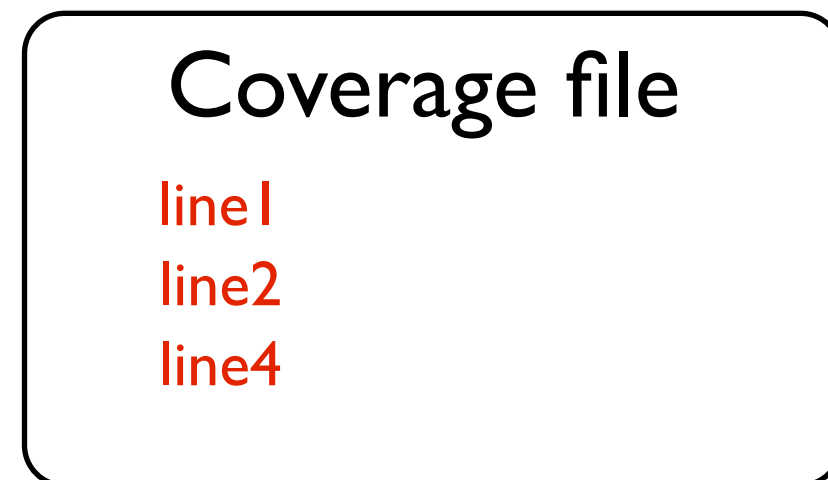
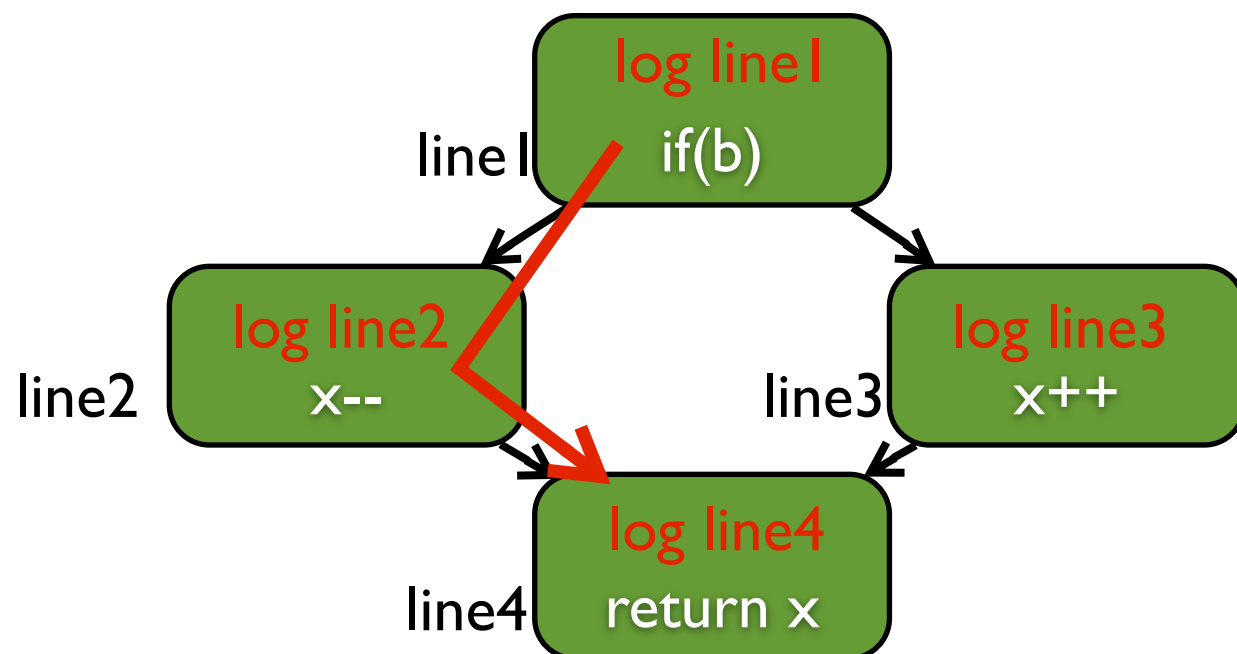


This class

- Code coverage
 - Control-flow coverage
 - Statement coverage
 - Branch coverage
 - Path coverage
 - Coverage Collection Tools
 - EcEmma

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

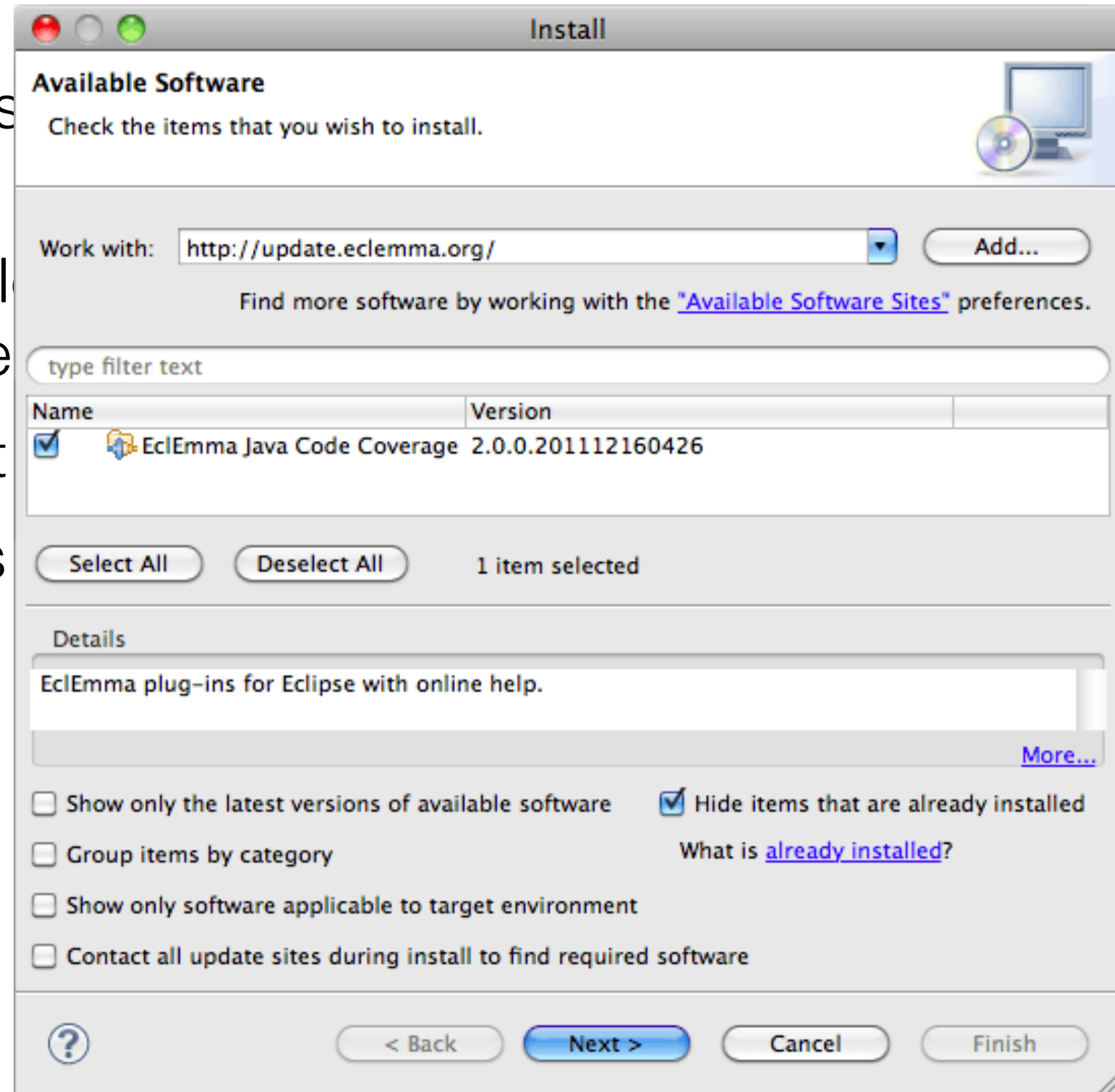
- Emma: <http://emma.sourceforge.net/>
- Eclemma: <http://www.eclemma.org/installation.html/>
- Cobertura: <http://cobertura.github.io/cobertura/>
- Clover: [https://www.atlassian.com/software/clover/
overview](https://www.atlassian.com/software/clover/overview)
- JCov: <https://wiki.openjdk.java.net/display/CodeTools/jcov>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

EclEmma: Installation

- From your Eclipse menu select Help → Install New Software...
- In the Install dialog enter <http://update.eclemma.org/> at the Work with field
- Check the latest EclEmma version and press Next
- Follow the steps in the installation wizard.

EclEmma: Installation

- From your Eclipse Software...
- In the Install dialog the Work with field
- Check the latest
- Follow the steps

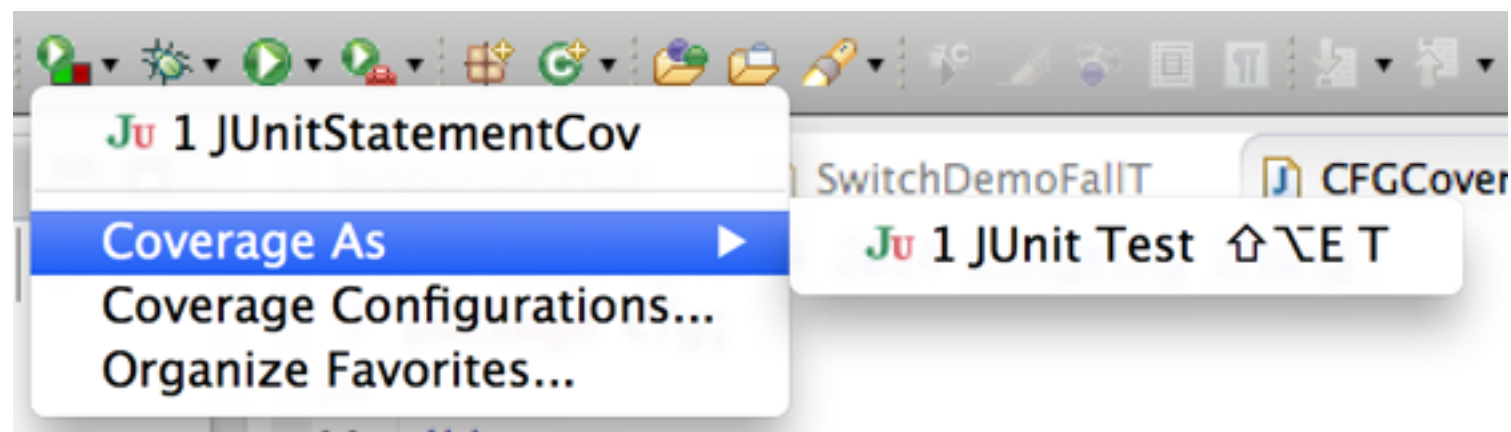


EclEmma: Execution

- The installation was successful if you can see the coverage launcher in the toolbar of the Java perspective:

- Coverage col 

- Right click the test suite class file to run
- Click “Coverage As” => “JUnit Tests”



EclEmma: Results

```

MyMainJUnitSuit  SwitchDemoFallIT  CFGCoverageExam  JUnitStatementC  JUnitBranchCov.
2+ * Copyright © 2014 Lingming Zhang
9 package cfg;
10
11 /**
12  * @author zhanglingming
13  *
14  */
15 public class CFGCoverageExample {
16     public int testMe(int x, boolean a, boolean b){
17         if(a)
18             x++;
19         if(b)
20             x--;
21         return x;
22     }
23 }
24

```

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
(default package)	0.0 %	0	339	339
junit.datastructures	0.0 %	0	172	172
cfg	15.8 %	30	160	190
SwitchDemoFallThrough.java	0.0 %	0	82	82
JUnitPathCov.java	0.0 %	0	49	49
JUnitBranchCov.java	0.0 %	0	29	29
CFGCoverageExample.java	100.0 %	11	0	11
JUnitStatementCov.java	100.0 %	10	0	10

Next class

- More on code coverage

Thanks!