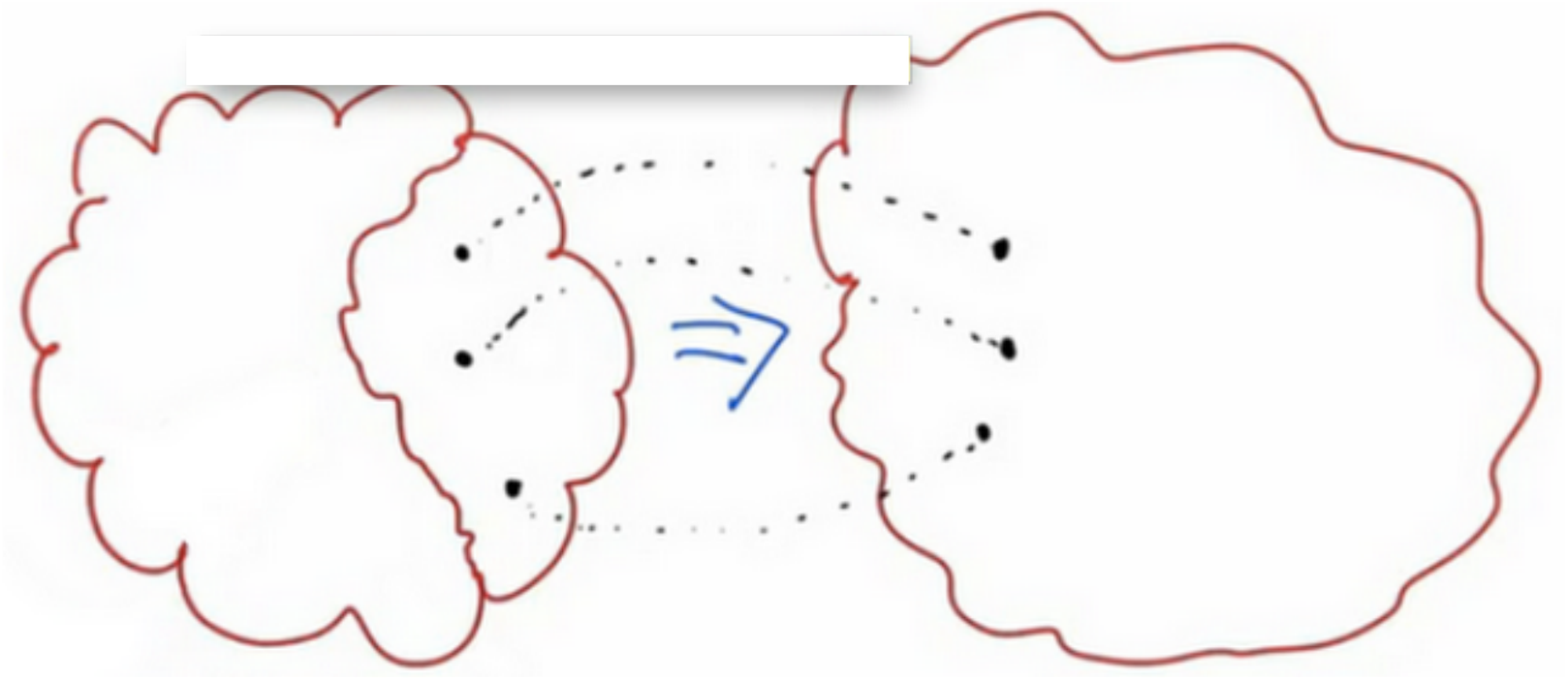
WHITE-BOX TESTING

HOW MUCH TESTING IS ENOUGH?



sometimes we think we partitioned correctly, but we didn't. It turns out that we thought some tests were equivalent, but they were not.

TEST COVERAGE

- Tries to do what partition does, but in a different way.
- *Test coverage is a semi-automatic way of partitioning the input domain based on observable features of the source code.*
- *Informally, test coverage is "how much" of the source code we are testing with our test suite*

TYPES OF COVERAGE

FUNCTION AND STATEMENT COVERAGE

Percentage of functions or statements of our source code which are executed by our set of test cases

```
def foo(x,y):
    if x == 0:
        y += 1
    if y == 0:
        x += 1
```

```
foo(0, -1)
```

FUNCTION AND STATEMENT COVERAGE

- Good:
  - Objective measure
  - We know what it takes to get full coverage
- Bad
  - Cant find out bugs of omission
  - 100% coverage does not mean bug-free
  - is 80% coverage good or bad?
- Would you like to have 100% coverage?

BRANCH COVERAGE

- Coverage of each outcome of condition evaluations

```
def foo(x,y):
    if x == 0:
        y += 1
    if y == 0:
        x += 1
```

foo(0, -1) has 100% statement coverage,
50% branch coverage

LOOP COVERAGE

- Each loop is executed 0 times, once, and more than once.
- Rationale: Loop boundary conditions are an extremely frequent source of bugs in real codes.
- For example, to get full loop coverage we would need to test this code using a file that contains no lines, using a file that contains just one line, and using a file that contains multiple lines.

```
for line in open("file"):
    process(line)
```

MODIFIED CONDITION DECISION COVERAGE (MC/DC)

- Required for any safety-critical system.
- it is
  - branch coverage +
  - states that every term involved in a decision takes on every possible outcome +
  - states that every term used in a decision independently affects its outcome

MODIFIED CONDITION DECISION COVERAGE (MC/DC): EXAMPLE

if A or (B and C):

- First, we make A true and false, blocking the other values so that A is decisive

```
#  A          B          C
if True or (False and True):
#  True       False
#  True
```

```
#  A          B          C
if False or (False and True):
#  False      False
#  False
```

MODIFIED CONDITION DECISION COVERAGE (MC/DC)

if A or (B and C):

- next, let's work on B and C.

```
# A          B          C
if False or (True and True):
# False           True
# True
```

```
# A          B          C
if False or (False and True):
# False           False
# False
```

MCDC: WHY?

- When we have complicated boolean expressions, their truth table is large.
- There's a lot of complexity hiding in those truth tables. When there's complexity, there are probably things we don't understand, and that means there are probably bugs.
- We're going to figure out when we have conditionals that don't independently affect the outcome of a decision. That means that somebody didn't understand what was going on, and probably there's something that we need to look at more closely and probably even change.

PATH COVERAGE

- A path through a program is a sequence of decisions made by operators in the program.
- Path coverage cares about how you got to a certain piece of code.

```
def foo(x,y):
    for i in range(x):
        something()
    if y:
        somethingElse()
```

HOW MANY TESTS DO I NEED FOR STATEMENT, BRANCH, PATH COVERAGE?

```
def foo(x, y, z):
    if x:
        ...
    if y:
        ...
    if z:
        ...
```

BOUNDARY VALUE COVERAGE

- When a program depends on some numerical range, and when the program has different behaviors based on numbers within that range, then we should test numbers close to the boundary.

```
for i in range(0,x):
    print(i)
```

CONCURRENT AND SYNCHRONIZATION COVERAGE

```
def xfer (amount):
    global a1,a2
    a1+=amount
    a2-=amount
    return
```

LOCK

UNLOCK

WHAT ABOUT CODE NOT COVERED?

- 3 possibilities
  - unfeasible code
  - not worth covering
  - incomplete test suite