# Why dynamic test techniques?

Exhaustive testing (use of all possible inputs and conditions) is impractical

- must use a subset of all possible test cases
- must have high probability of detecting faults

Need thought processes that help us select test cases more intelligently

- test case design techniques are such thought processes

# What is a testing technique?

a procedure for selecting or designing tests

based on a structural or functional model of the software

successful at finding faults

'best' practice

a way of deriving good test cases

a way of objectively measuring a test effort

***Testing should be rigorous, thorough and systematic***

# Advantages of techniques

Different people: similar probability find faults

- gain some independence of thought

Effective testing: find more faults

- focus attention on specific types of fault
- know you're testing the right thing

Efficient testing: find faults with less effort

- avoid duplication
- systematic techniques are measurable

***Using techniques makes testing much more effective***

# Measurement

Objective assessment of thoroughness of testing (with respect to use of each technique)

- useful for comparison of one test effort to another

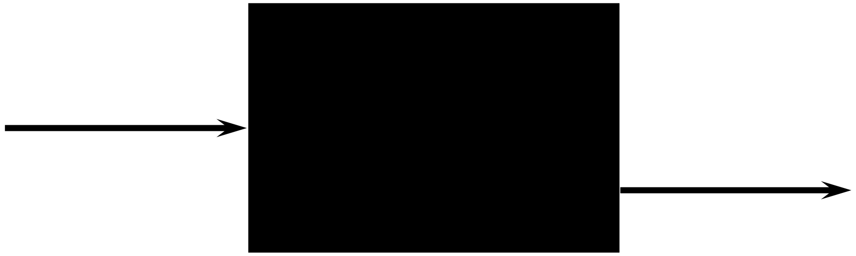
E.g.

| Project A | Project B |
|---|---|
| 60% Equivalence partitions | 40% Equivalence partitions |
| 50% Boundaries | 45% Boundaries |
| 75% Branches | 60% Branches |

# Three types of systematic technique

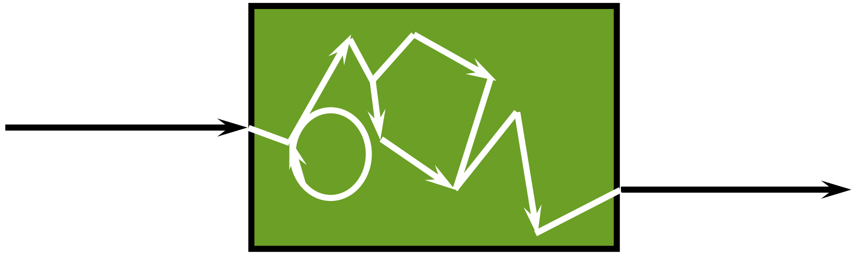
Static (non-execution)

- examination of documentation, source code listings, etc.



Functional (Black Box)

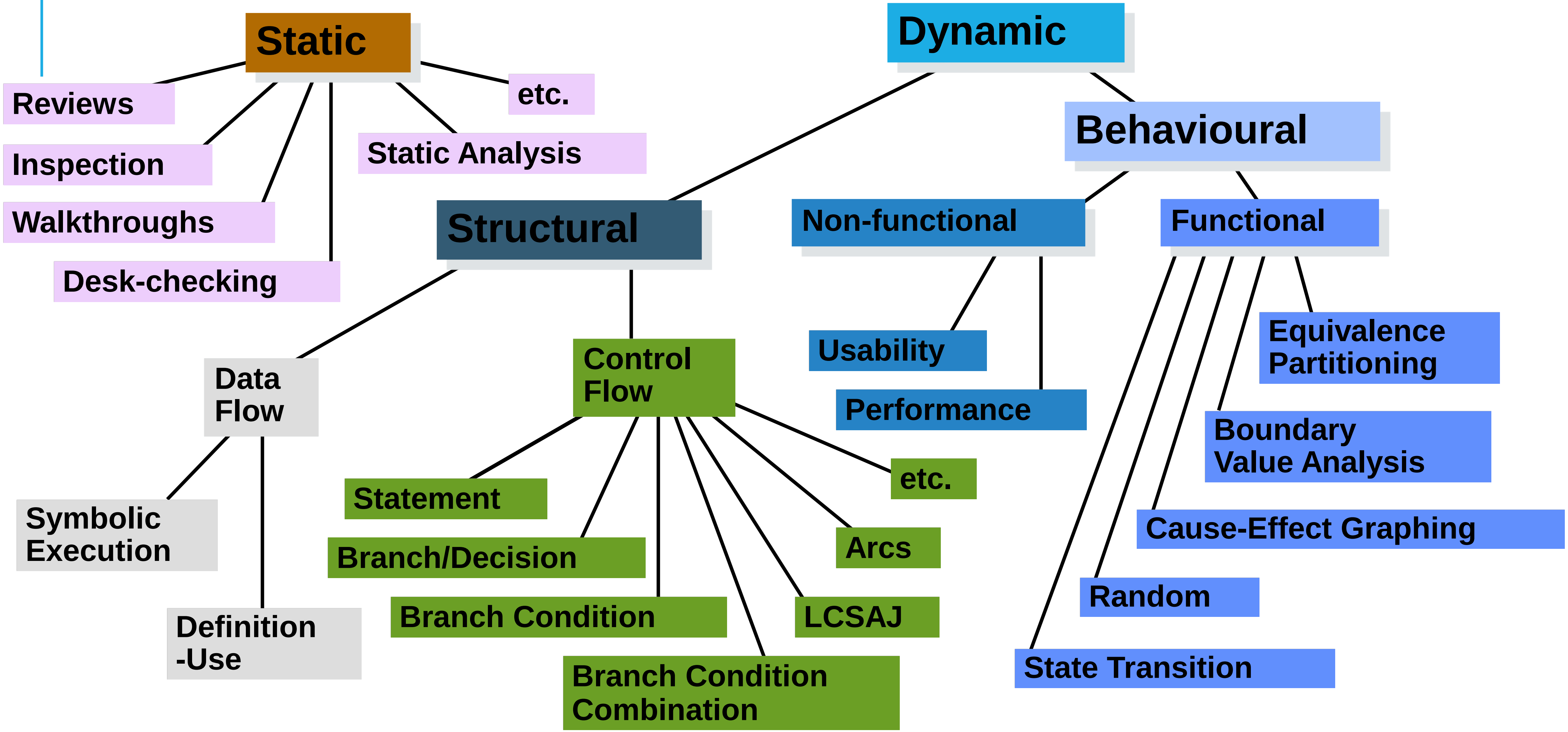
- based on behaviour / functionality of software



Structural (White Box)

- based on structure of software

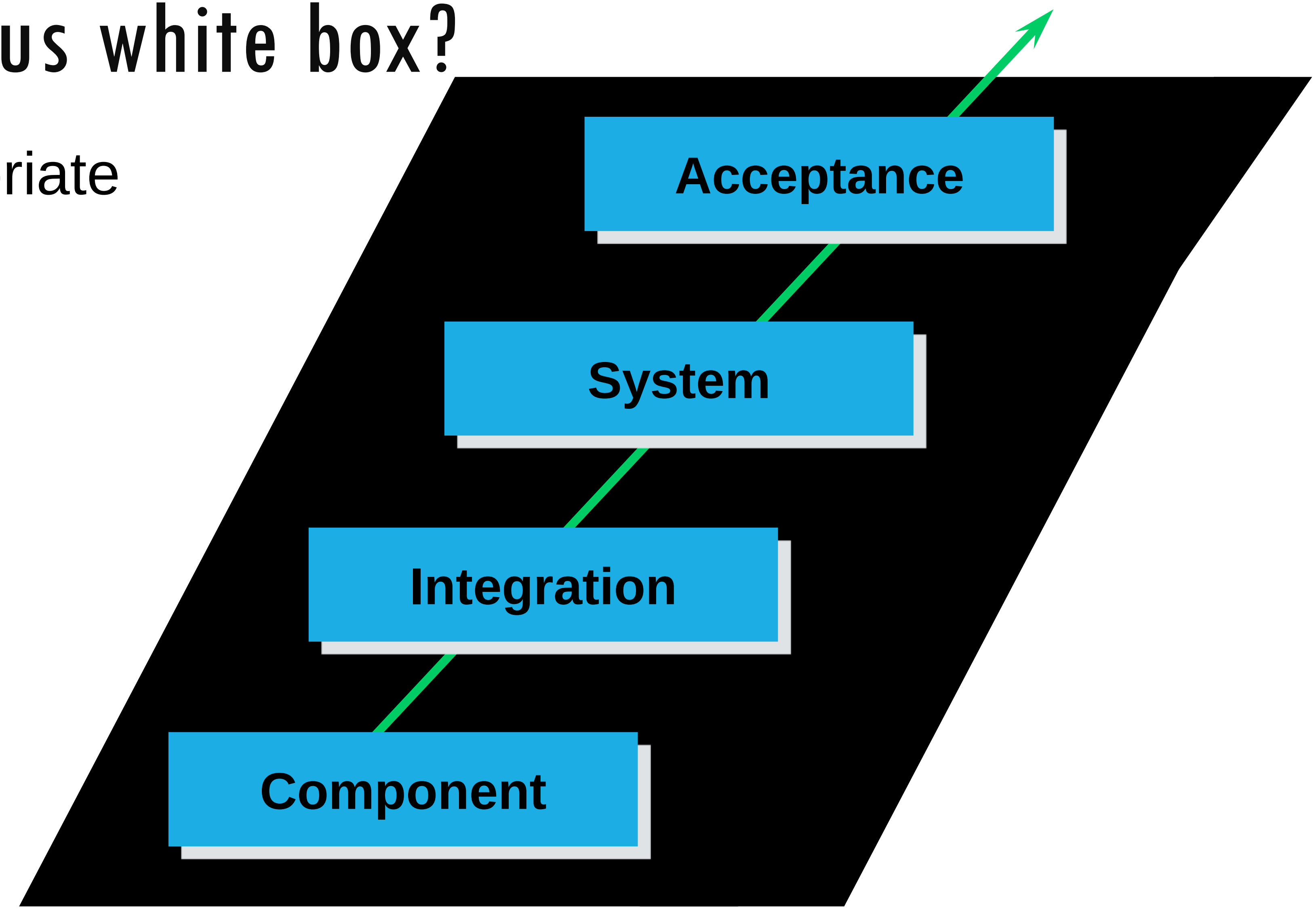
# Some test techniques



# Black box versus white box?

Black box is appropriate at all levels but dominates higher levels of testing

White box used predominately at lower levels to compliment black box



# Black Box test design and measurement techniques

Techniques defined in BS 7925-2

- Equivalence partitioning ✓
- Boundary value analysis ✓
- State transition testing ✓
- Cause-effect graphing ✓
- Syntax testing ✗
- Random testing ✗

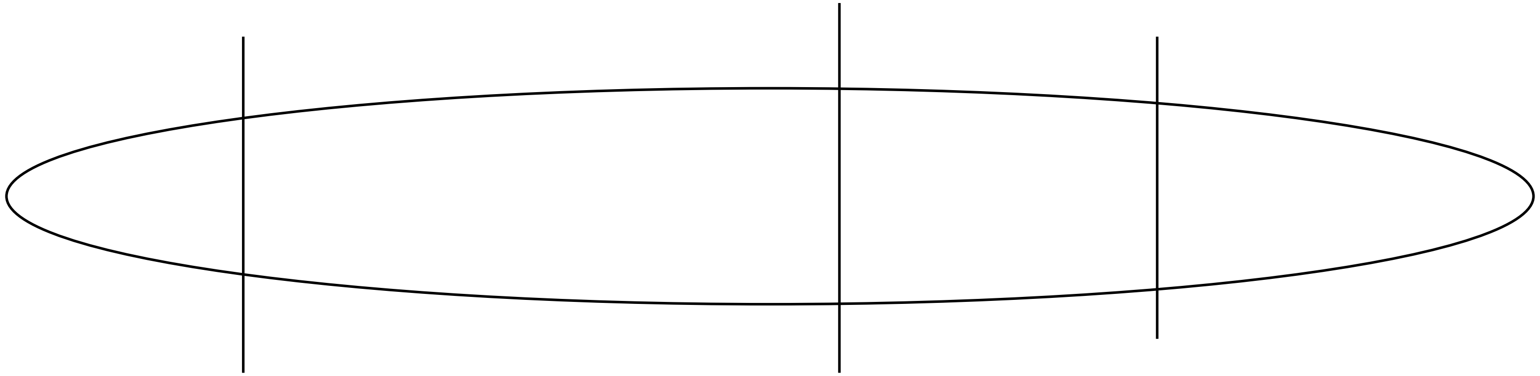
Also defines how to specify other techniques

**Also a measurement technique?**

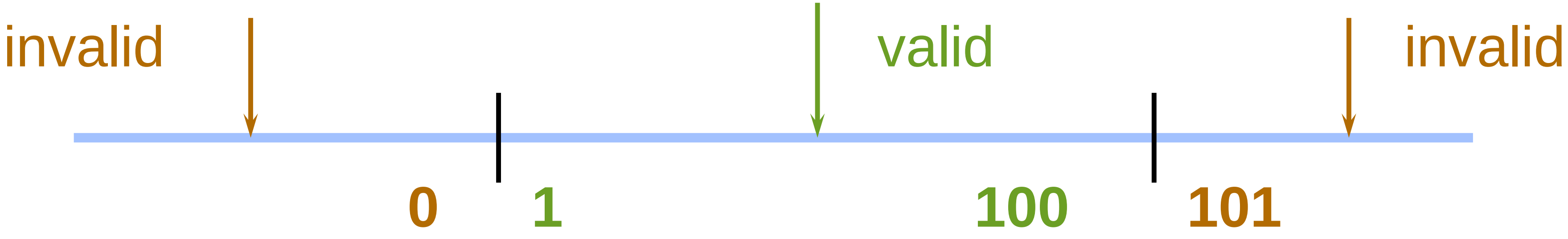
**✓ = Yes**

**✗ = No**

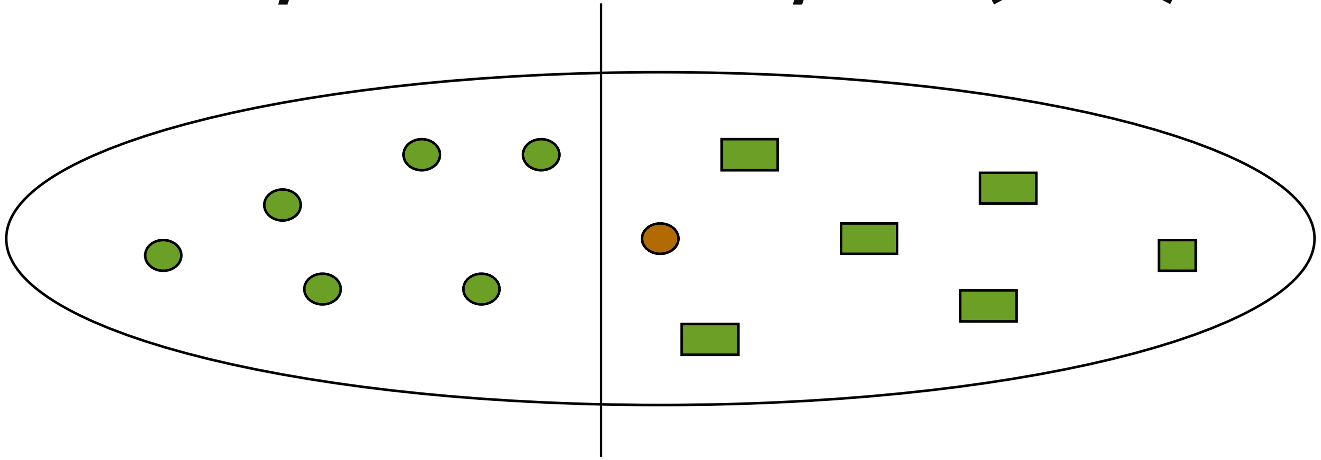
# Equivalence partitioning (EP)



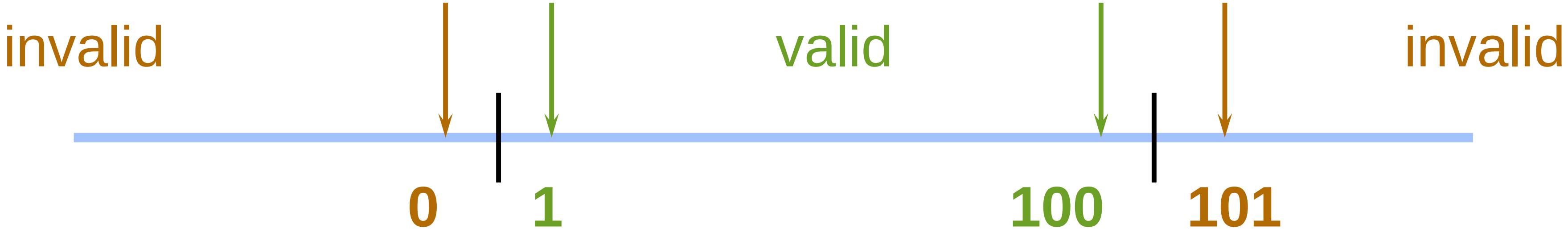
- divide (partition) inputs, outputs, etc. into areas which are the same (equivalent)
- assumption: if one value works, all will work
- one from each partition better than all from one



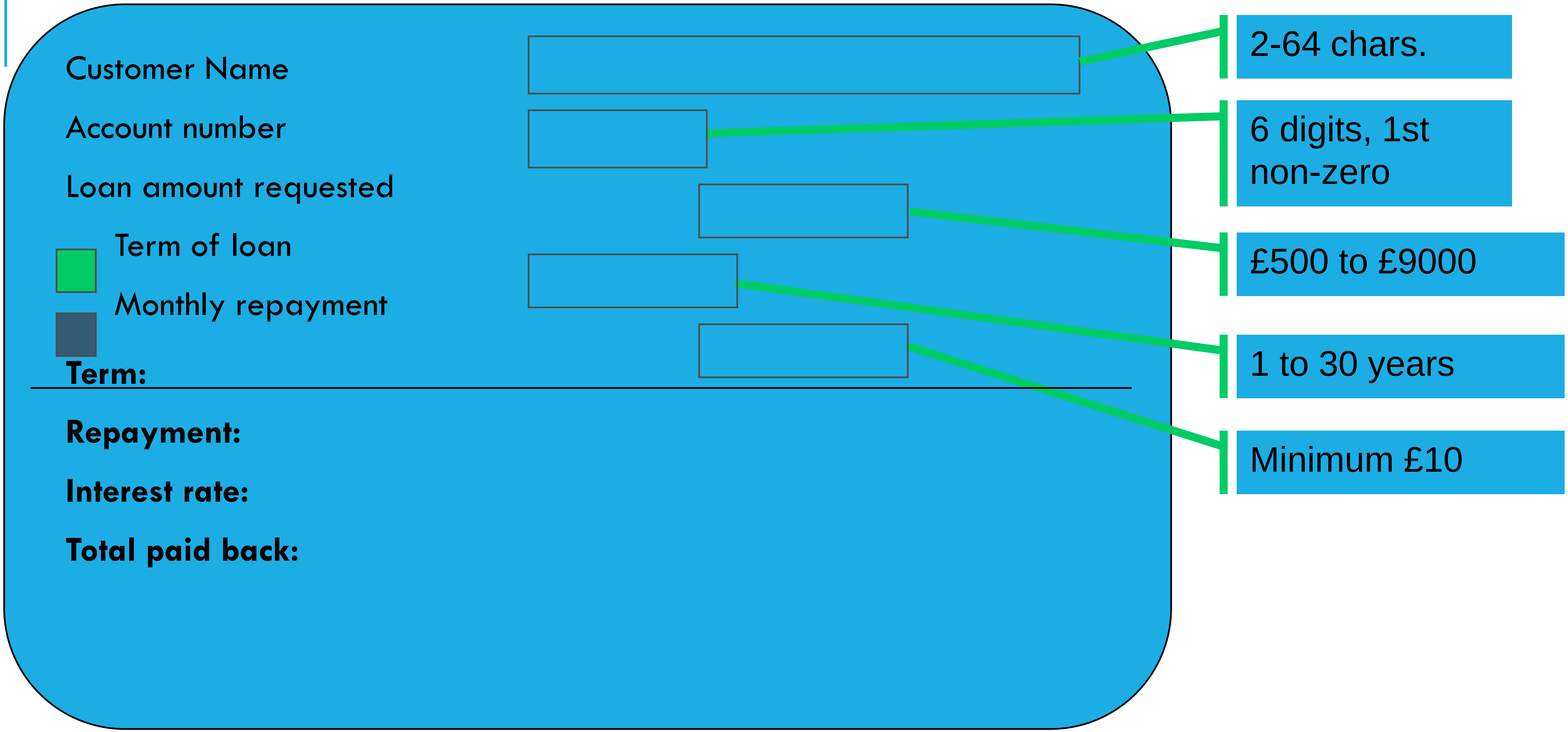
# Boundary value analysis (BVA)



- faults tend to lurk near boundaries
- good place to look for faults
- test values on both sides of boundaries



# Example: Loan application



Customer Name

Account number

Loan amount requested

🟩 Term of loan

⬛ Monthly repayment

**Term:**

**Repayment:**

**Interest rate:**

**Total paid back:**

2-64 chars.

6 digits, 1st non-zero

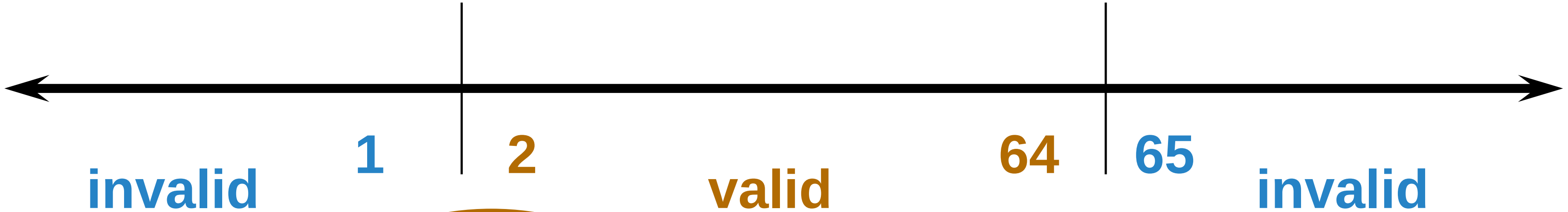
£500 to £9000

1 to 30 years

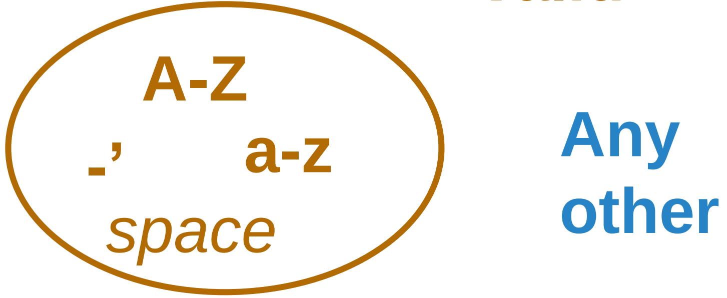
Minimum £10

# Customer name

**Number of characters:**



**Valid characters:**



| Conditions | Valid Partitions | Invalid Partitions | Valid Boundaries | Invalid Boundaries |
|---|---|---|---|---|
| Customer name | 2 to 64 chars<br>valid chars | < 2 chars<br>> 64 chars<br>invalid chars | 2 chars<br>64 chars | 1 chars<br>65 chars<br>0 chars |

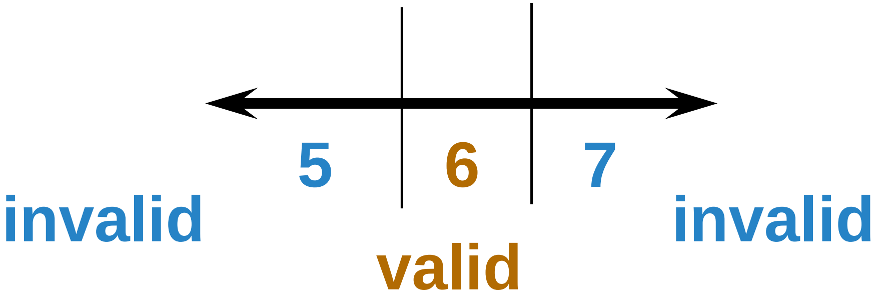
# Account number

first character:

valid: non-zero

invalid: zero

number of digits:



invalid

5

6

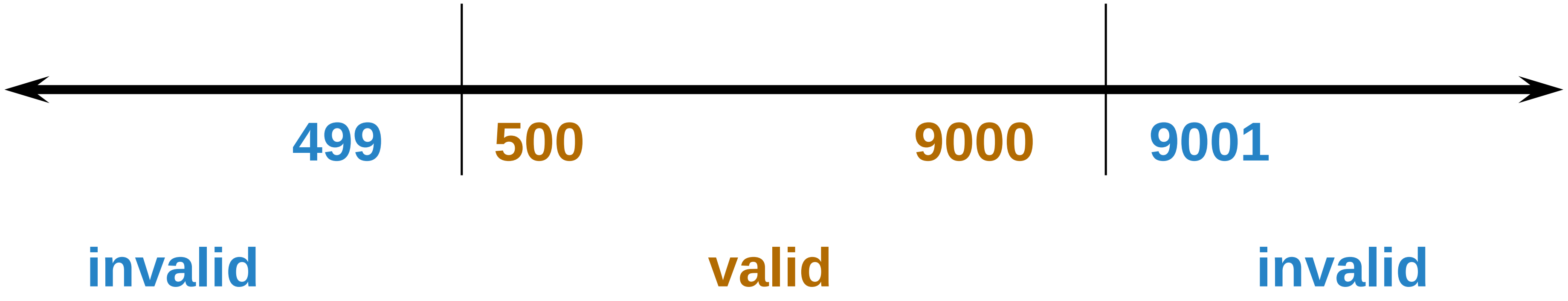
7

invalid

valid

| Conditions | Valid Partitions | Invalid Partitions | Valid Boundaries | Invalid Boundaries |
|---|---|---|---|---|
| Account number | 6 digits | < 6 digits | 100000 | 5 digits |
| | 1st non-zero | > 6 digits | 999999 | 7 digits |
| | | 1st digit = 0 | | 0 digits |
| | | non-digit | | |

# Loan amount



| Conditions | Valid Partitions | Invalid Partitions | Valid Boundaries | Invalid Boundaries |
|---|---|---|---|---|
| Loan amount | 500 - 9000 | < 500<br>>9000<br>0<br>non-numeric<br>null | 500<br>9000 | 499<br>9001 |

# Condition template

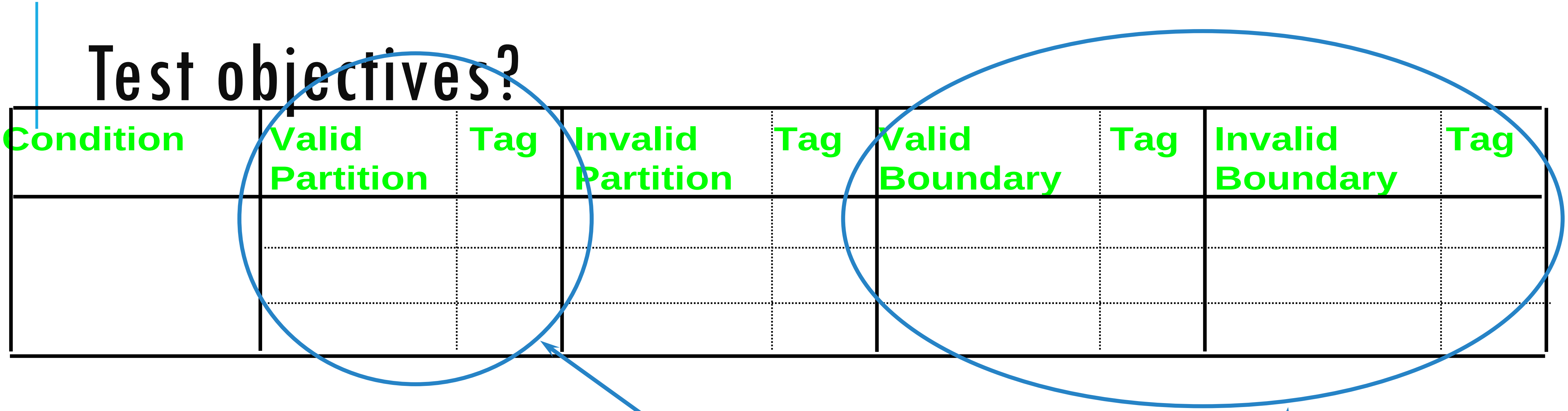
| Conditions | Valid Partitions | | Invalid Partitions | | Valid Boundaries | | Invalid Boundaries | |
|---|---|---|---|---|---|---|---|---|
| Customer name | 2 - 64 chars<br>valid chars | | < 2 chars<br>> 64 chars<br>invalid char | | 2 chars<br>64 chars | | 1 char<br>65 chars<br>0 chars | |
| Account number | 6 digits<br>1st non-zero | | < 6 digits<br>> 6 digits<br>1st digit = 0<br>non-digit | | 100000<br>999999 | | 5 digits<br>7 digits<br>0 digits | |
| Loan amount | 500 - 9000 | | < 500<br>>9000<br>0<br>non-integer<br>null | | 500<br>9000 | | 499<br>9001 | |

# Why do both EP and BVA?

If you do boundaries only, you have covered all the partitions as well

- technically correct and may be OK if everything works correctly!
- if the test fails, is the whole partition wrong, or is a boundary in the wrong place - have to test mid-partition anyway
- testing only extremes may not give confidence for typical use scenarios (especially for users)
- boundaries may be harder (more costly) to set up

# Test objectives?



| Condition | Valid Partition | Tag | Invalid Partition | Tag | Valid Boundary | Tag | Invalid Boundary | Tag |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

- **For a thorough approach: VP, IP, VB, IB**
- **Under time pressure, depends on your test objective**
  - minimal user-confidence: VP only?
  - maximum fault finding: VB first (plus IB?)

# Decision tables

explore combinations of inputs, situations or events,

it is very easy to overlook specific combinations of input

start by expressing the input conditions of interest so that they are either TRUE or FALSE

– record found

– file exists

– code valid

– policy expired

– account in credit

– due date > current date

# Example: student access

A university computer system allows students an allocation of disc space depending on their projects.

If they have used all their allotted space, they are only allowed restricted access, i.e. to delete files, not to create them. This is assuming they have logged on with a valid username and password.

What are the input and output conditions?

# List the input and output conditions

- list the ‘input conditions’ in the first column of the table

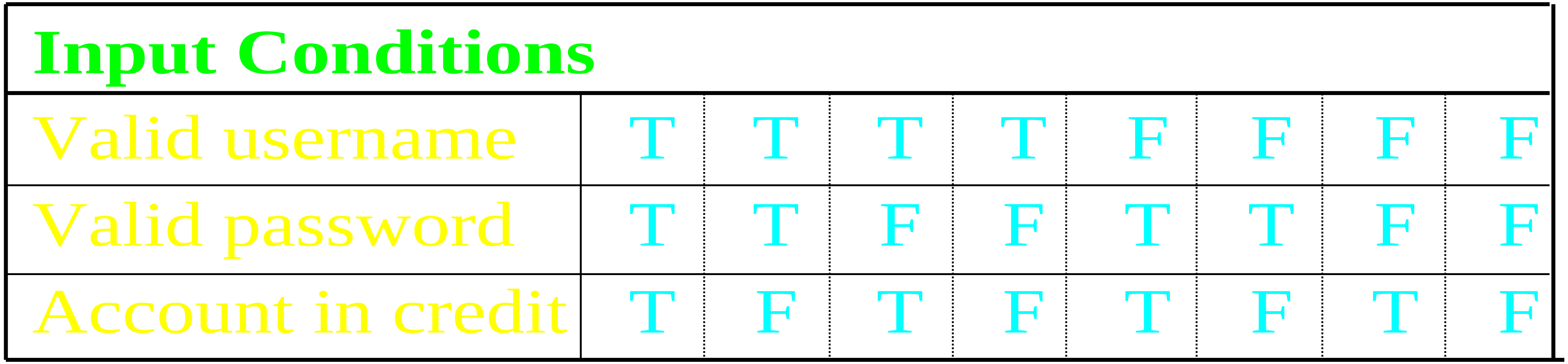
- list the ‘output conditions’ under the input conditions

| | |
|---|---|
| **Input Conditions** | |
| Valid username | |
| Valid password | |
| Account in credit | |
| **Output Conditions** | |
| Login accepted | |
| Restricted access | |

# Determine input combinations

add columns to the table for each unique combination of input conditions.

each entry in the table may be either ‘T’ for true, ‘F’ for false.



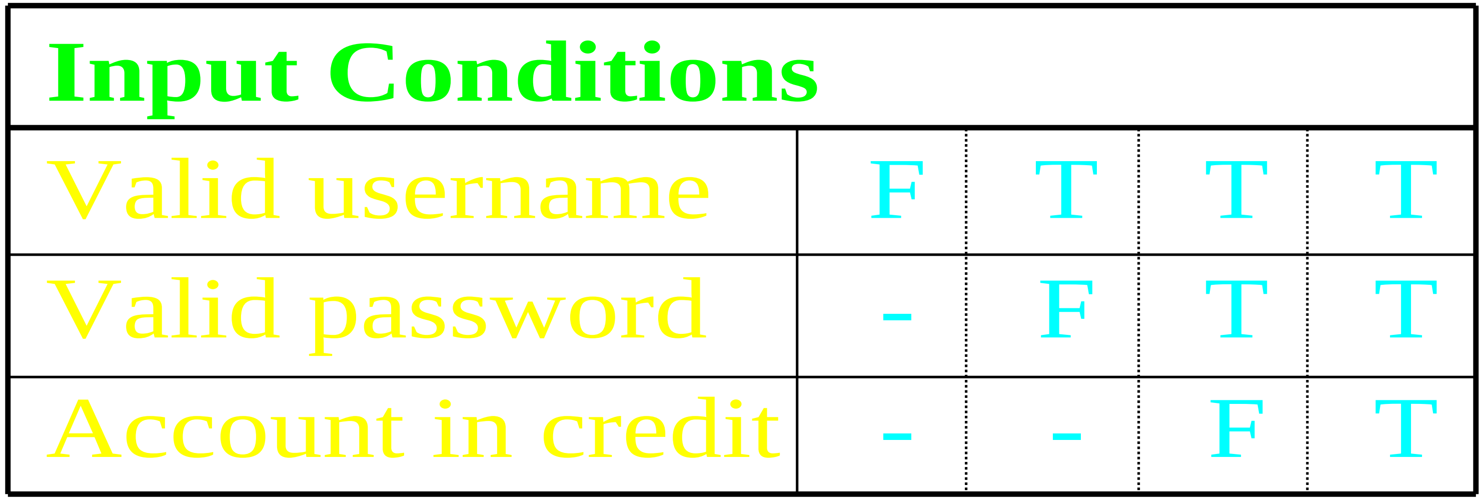
| Input Conditions | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Valid username | T | T | T | T | F | F | F | F |
| Valid password | T | T | F | F | T | T | F | F |
| Account in credit | T | F | T | F | T | F | T | F |

# Rationalise input combinations

some combinations may be impossible or not of interest

some combinations may be ‘equivalent’

use a hyphen to denote “don’t care”



| **Input Conditions** | | | | |
|---|---|---|---|---|
| Valid username | F | T | T | T |
| Valid password | - | F | T | T |
| Account in credit | - | - | F | T |

# Complete the table

determine the expected output conditions for each combination of input conditions

| Input Conditions | | | | |
|---|---|---|---|---|
| Valid username | | | | |
| Valid password | | | | |
| Account in credit | | | | |
| **Output Conditions** | | | | |
| Login accepted | F | F | T | T |
| Restricted access | - | - | T | F |

# Determine test case groups

each column is at least one test case

| | | | | |
|---|---|---|---|---|
| **Input Conditions** | | | | |
| Valid username | | | | |
| Valid password | | | | |
| Account in credit | | | | |
| **Output Conditions** | | | | |
| Login accepted | | | | |
| Restricted access | | | | |
| **Tags** | **A** | **B** | **C** | **D** |

# Design test cases

usually one test case for each column but can be none or several

| Test | Description | Expected Outcome | Tag |
|---|---|---|---|
| 1 | Username BrbU | Invalid username | A |
| 2 | Username usernametoolong | Invalid username | A |
| 3 | Username BobU<br>Password abcd | Invalid password | B |
| 4 | Valid user, no disc space | Restricted access | C |
| 5 | Valid user with disc space | Unrestricted access | D |

# Rationalising outputs

if outputs or effects are mutually exclusive, I.e. T occurs in only one place in each column, we can combine them

for example:

| | | | |
|---|---|---|---|
| X | T | F | F |
| Y | F | T | F |
| Z | F | F | T |

**is equivalent to:**

| | | | |
|---|---|---|---|
| Output | X | Y | Z |

# Rationalising dangers

rationalising is based on assumptions

assumptions may be wrong!

assumptions should be stated

assumptions may change over time

be aware of the dangers

- filling in the full table may find errors which will be missed if you rationalise
- it is possible to rationalise too far

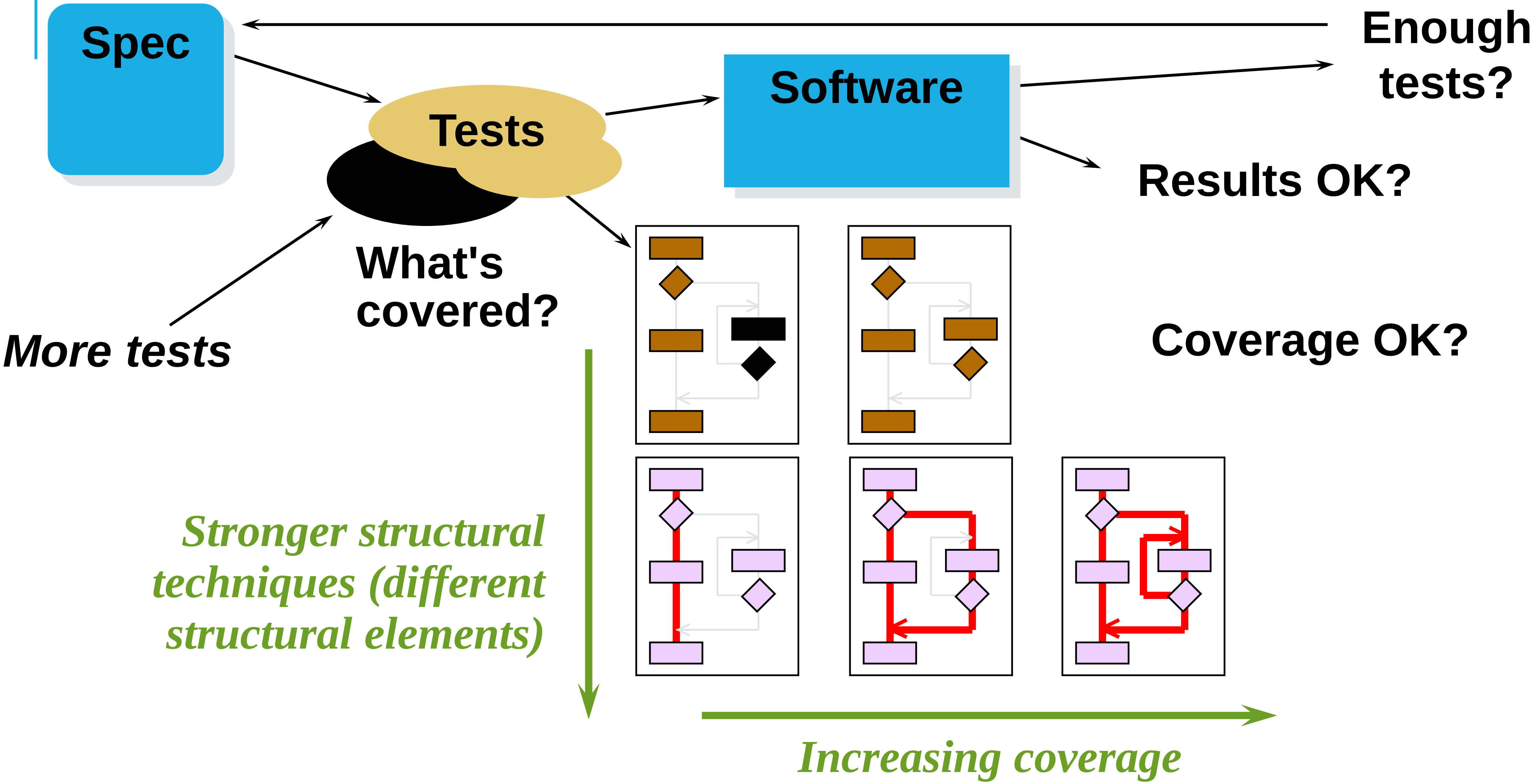
# White Box test design and measurement techniques

Techniques defined in BS 7925-2

- Statement testing
- Branch / Decision testing
- Data flow testing
- Branch condition testing

Also defines how to specify other techniques

# Using structural coverage



Spec

Tests

Software

Enough tests?

Results OK?

What's covered?

More tests

Coverage OK?

*Stronger structural techniques (different structural elements)*

*Increasing coverage*

# Statement coverage

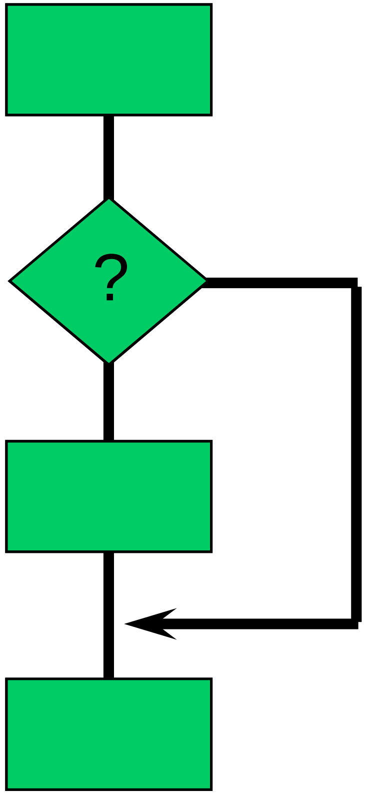
Statement coverage is normally measured by a software tool.

percentage of executable statements exercised by a test suite

$$= \frac{\text{number of statements exercised}}{\text{total number of statements}}$$

example:

- program has 100 statements
- tests exercise 87 statements
- statement coverage = 87%



**Typical ad hoc testing achieves 60 - 75%**

# Example of statement coverage

```
1  read(a)
2  IF a > 6 THEN
3      b = a
4  ENDIF
5  print b
```

Statement numbers

| Test case | Input | Expected output |
|---|---|---|
| 1 | 7 | 7 |

As all 5 statements are ‘covered’ by this test case, we have achieved 100% statement coverage

# Decision coverage (Branch coverage)

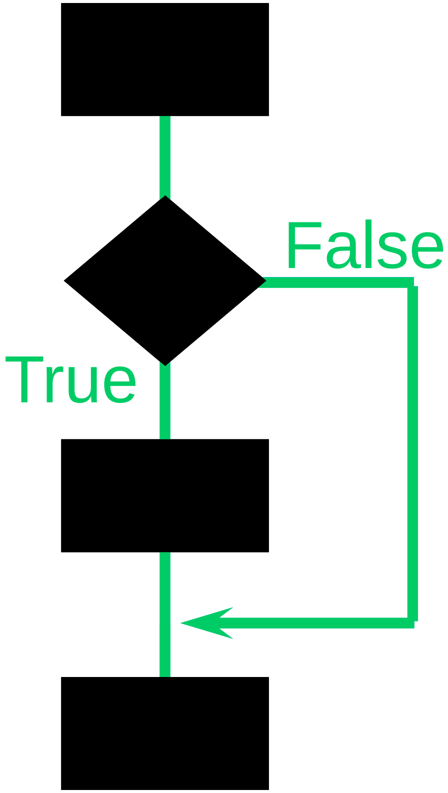
Decision coverage is normally measured by a software tool.

percentage of decision outcomes exercised by a test suite

$$= \frac{\text{number of decisions outcomes exercised}}{\text{total number of decision outcomes}}$$

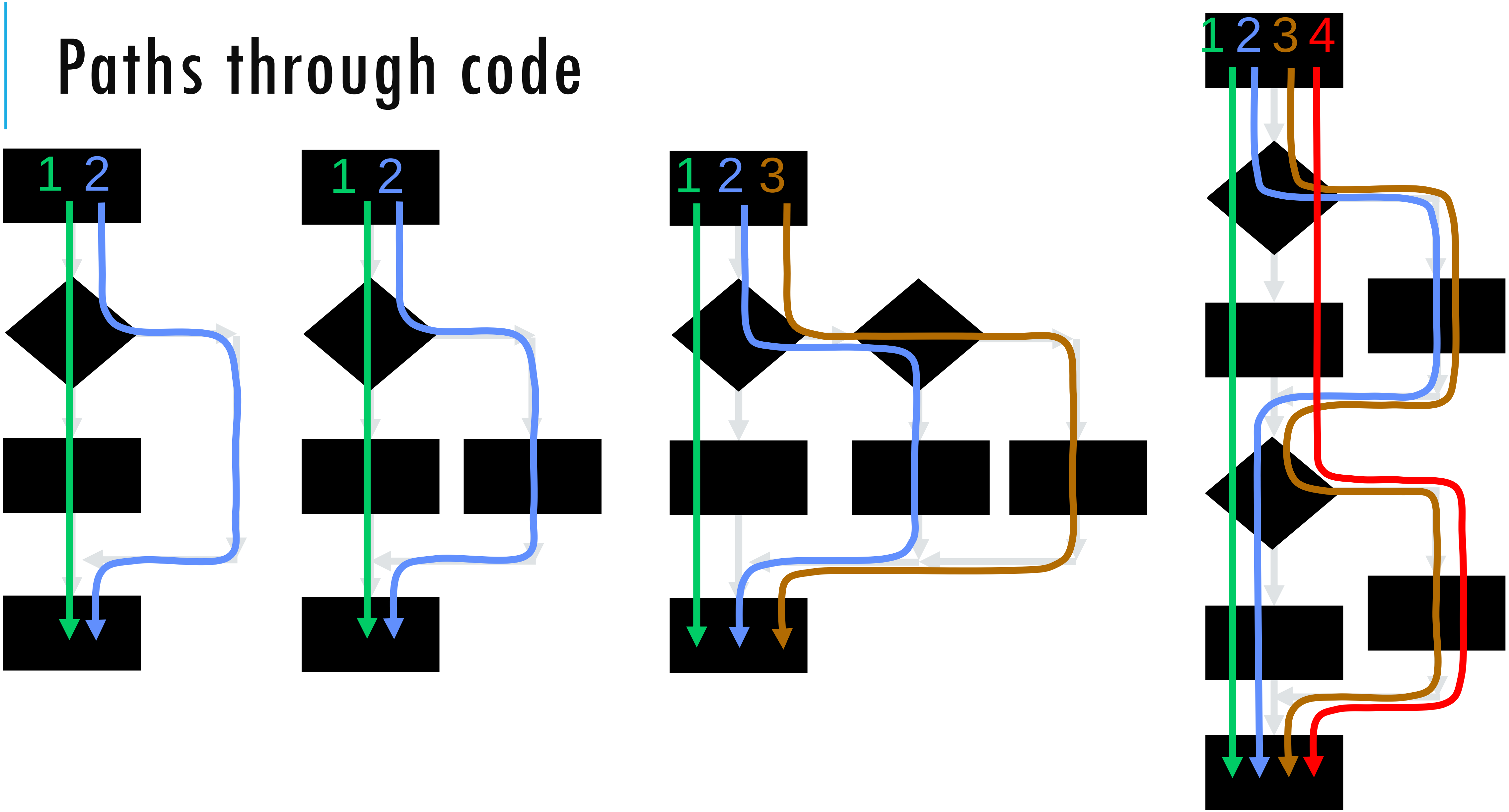
example:

- program has 120 decision outcomes
- tests exercise 60 decision outcomes
- decision coverage = 50%

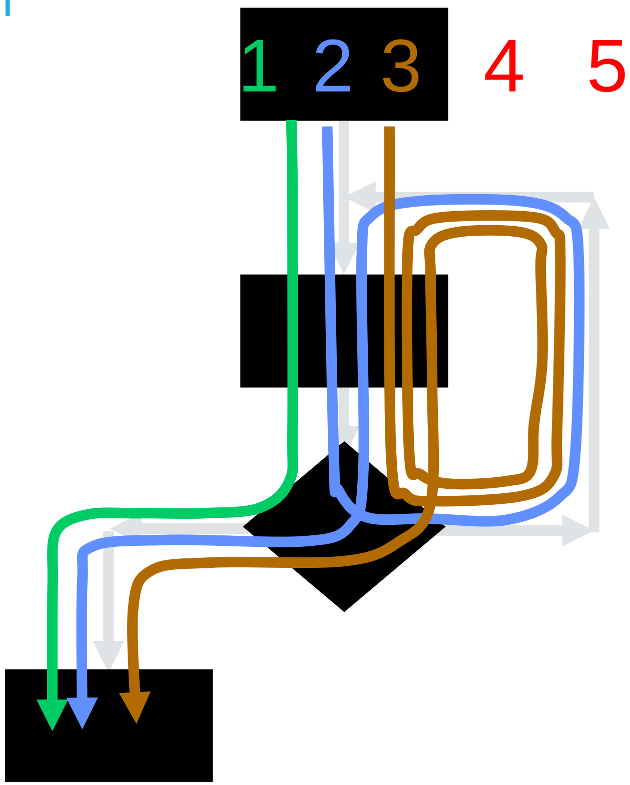


**Typical ad hoc testing achieves 40 - 60%**

# Paths through code



# Paths through code with loops

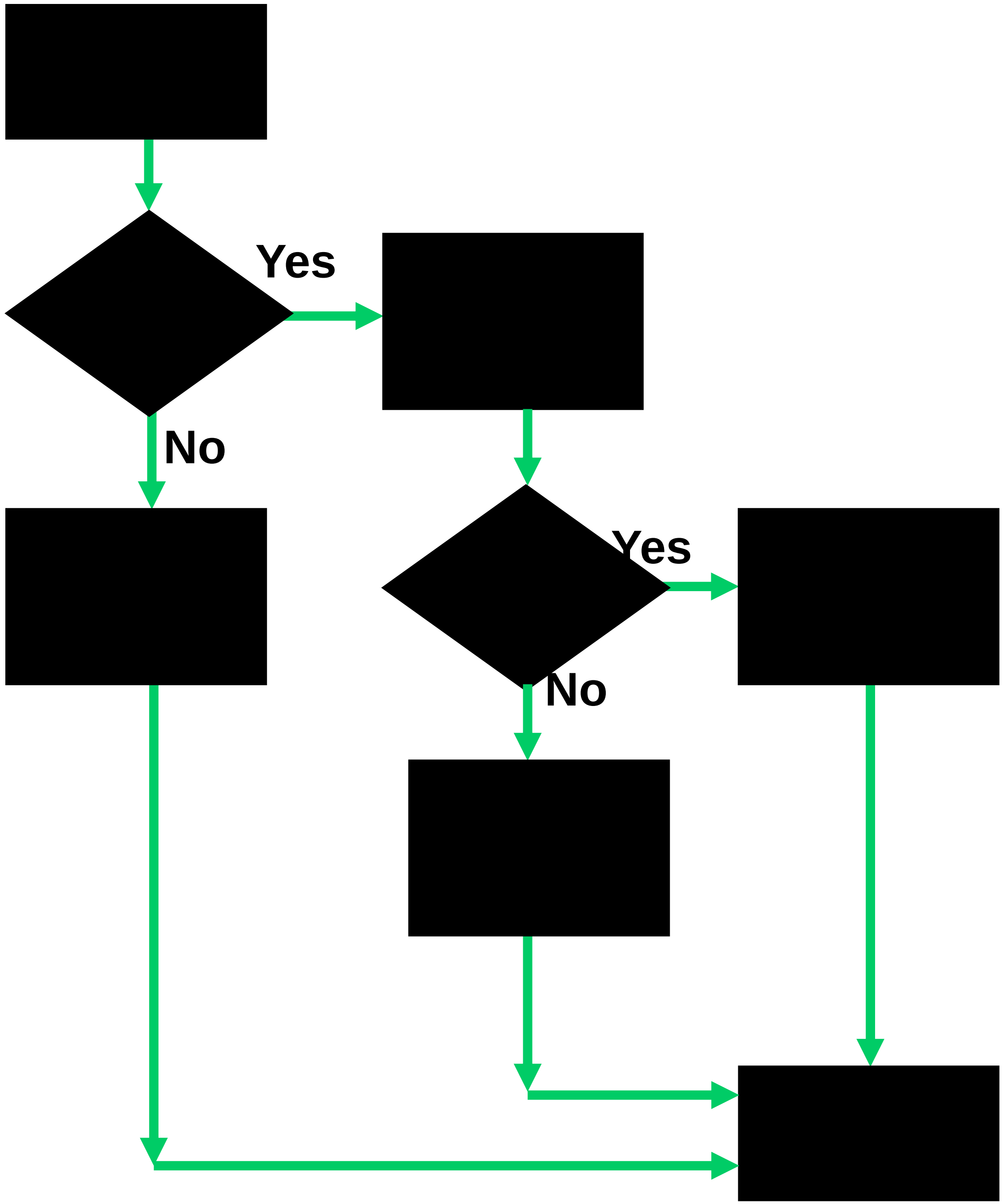


4 5 6 7 8 ....

for as many times as it is possible to go round the loop (this can be unlimited, i.e. infinite)

# Example 1

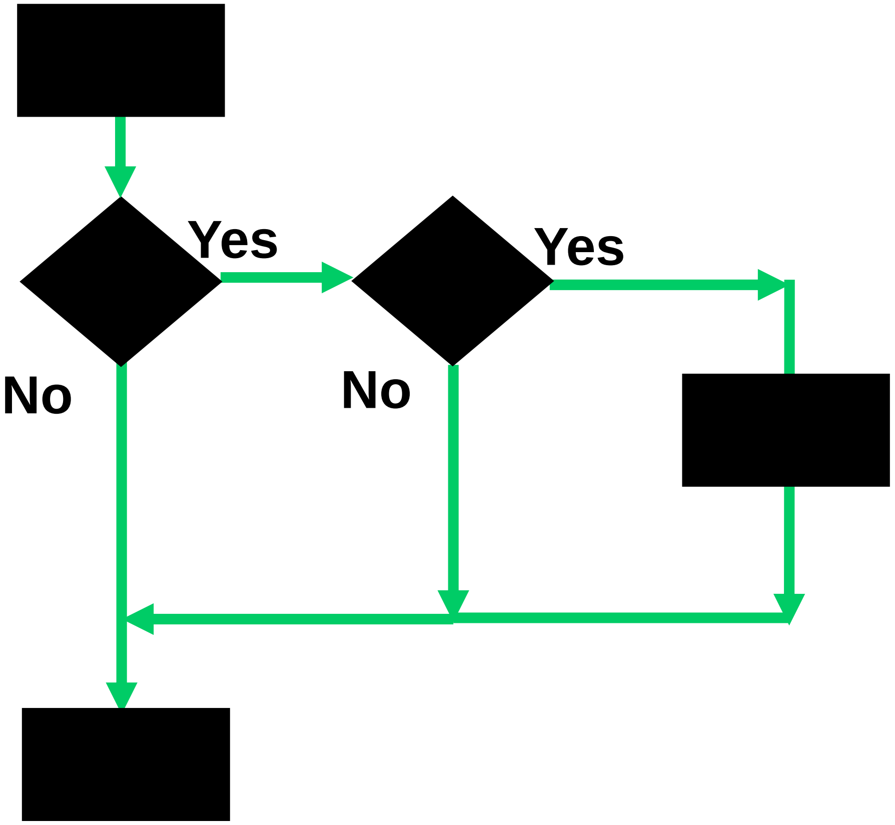
```
Wait for card to be inserted
IF card is a valid card THEN
  display "Enter PIN number"
  IF PIN is valid THEN
        select transaction
  ELSE (otherwise)
        display "PIN invalid"
ELSE (otherwise)
  reject card
End
```



# Example 2

```
Read A

IF A > 0 THEN
 IF A  = 21 THEN
 ENDIF
    Print "Key"
ENDIF
```



- Cyclomatic complexity: ______
- Minimum tests to achieve:
  - Statement coverage: _______
  - Branch coverage: ______

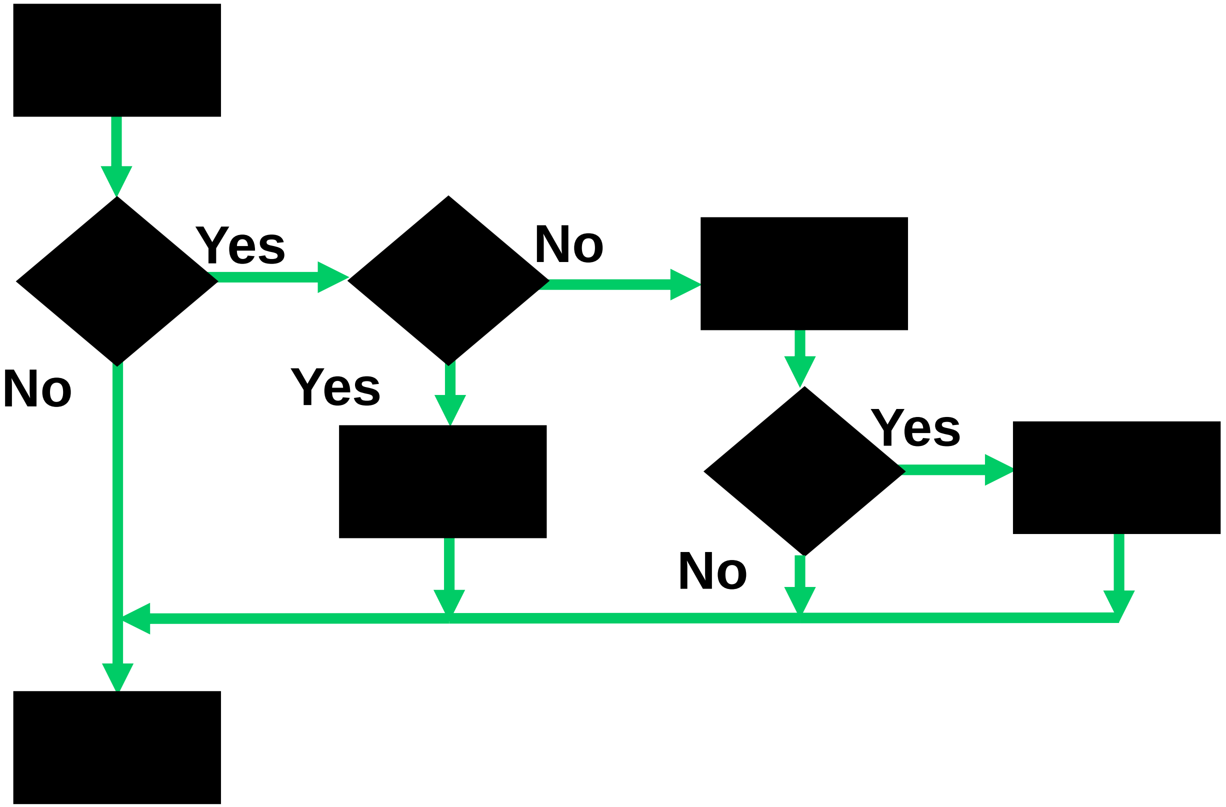
3

1

3

# Example 3

```
Read A
Read B
IF A > 0 THEN
    IF B  = 0 THEN
     Print "No values"
    ELSE
     Print B
     IF A > 21 THEN
         Print A
     ENDIF
    ENDIF
ENDIF
```



- Cyclomatic complexity: ______ 4
- Minimum tests to achieve:
  - Statement coverage: ______
  - Branch coverage: _____

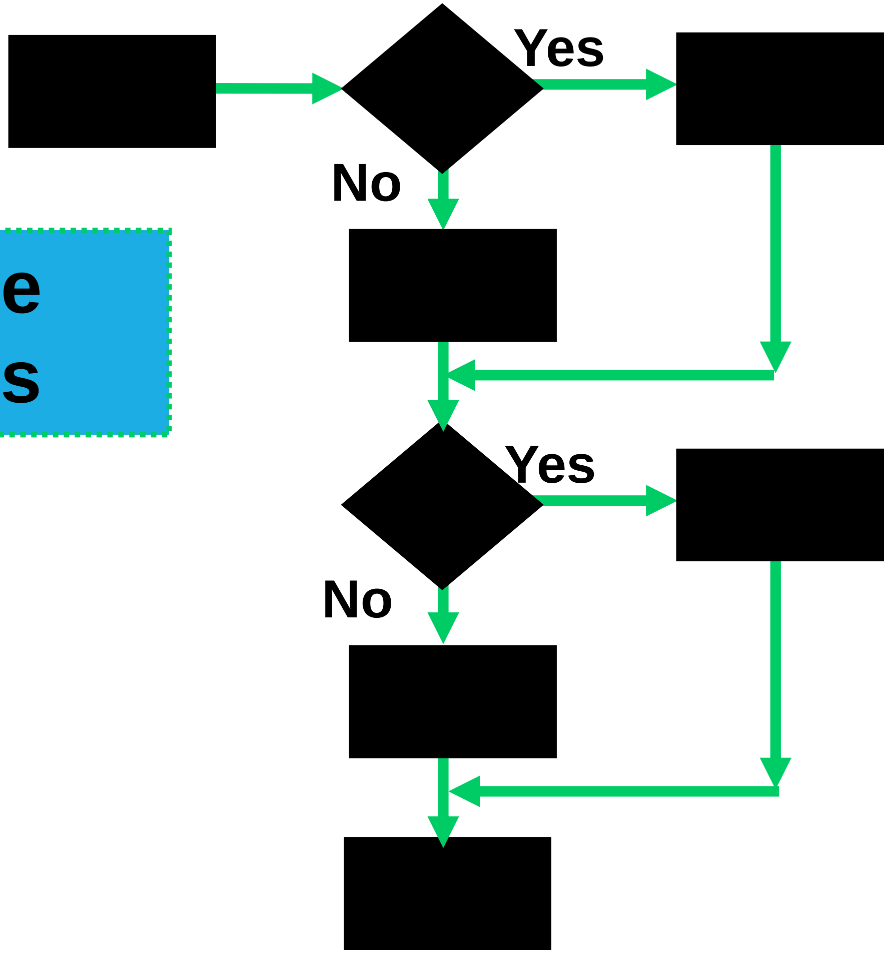
2

4

# Example 4

```
Read A
Read B
IF A < 0 THEN
    Print  "A negative"
ELSE
    Print  "A positive"
ENDIF
IF B < 0 THEN
    Print  "B negative"
ELSE
    Print  "B positive"
ENDIF
```

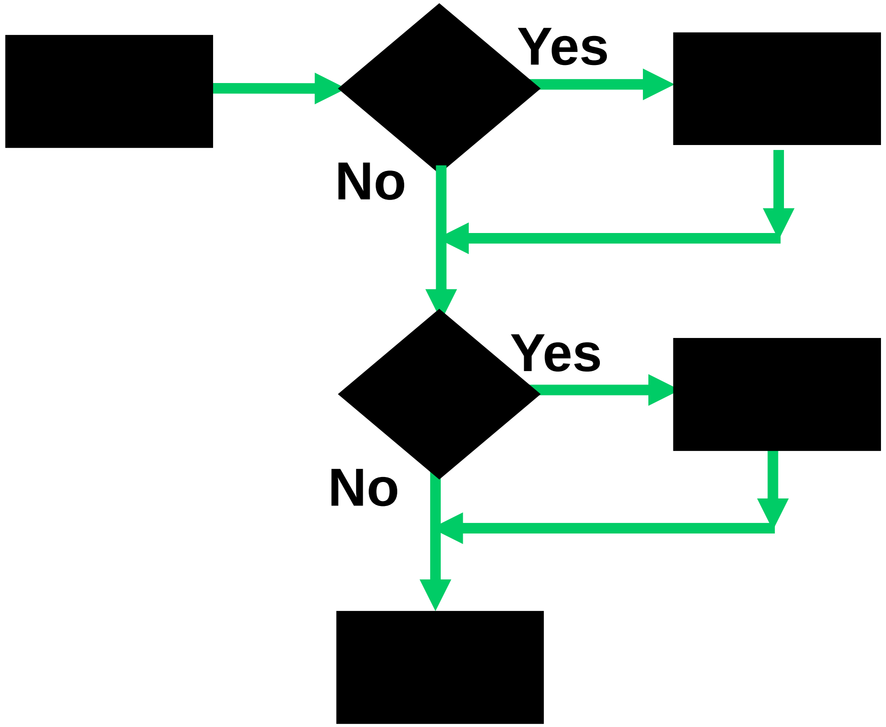
**Note: there are 4 paths**



- Cyclomatic complexity: ______ 3
- Minimum tests to achieve:
  - Statement coverage: _______ 2
  - Branch coverage: ______ 2

# Example 5

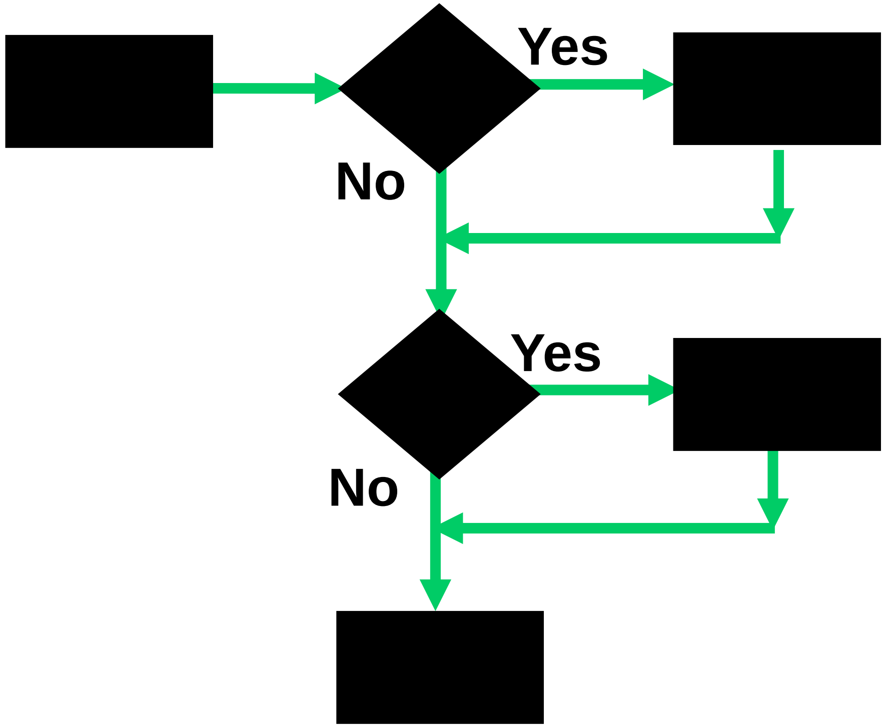
```
Read A
Read B
IF A < 0 THEN
    Print  “A negative”
ENDIF
IF B < 0 THEN
    Print  “B negative”
ENDIF
```



- Cyclomatic complexity: ______ 3
- Minimum tests to achieve:
  - Statement coverage: _______ 1
  - Branch coverage: ______ 2

# Example 6

```
Read A
IF A < 0 THEN
    Print "A negative"
ENDIF
IF A > 0 THEN
    Print "A positive"
ENDIF
```



- Cyclomatic complexity: ______
- Minimum tests to achieve:
  - Statement coverage: _______
  - Branch coverage: ______

3

2

2

# Non-systematic test techniques

Trial and error / Ad hoc

Error guessing / Experience-driven

User Testing

Unscripted Testing

***A testing approach that is only rigorous, thorough and systematic is incomplete***

# Error-Guessing

always worth including

after systematic techniques have been used

can find some faults that systematic techniques can miss

a ‘mopping up’ approach

supplements systematic techniques

Not a good approach to start testing with

# Error Guessing: deriving test cases

Consider:

- past failures
- intuition
- experience
- brain storming
- "What is the craziest thing we can do?"