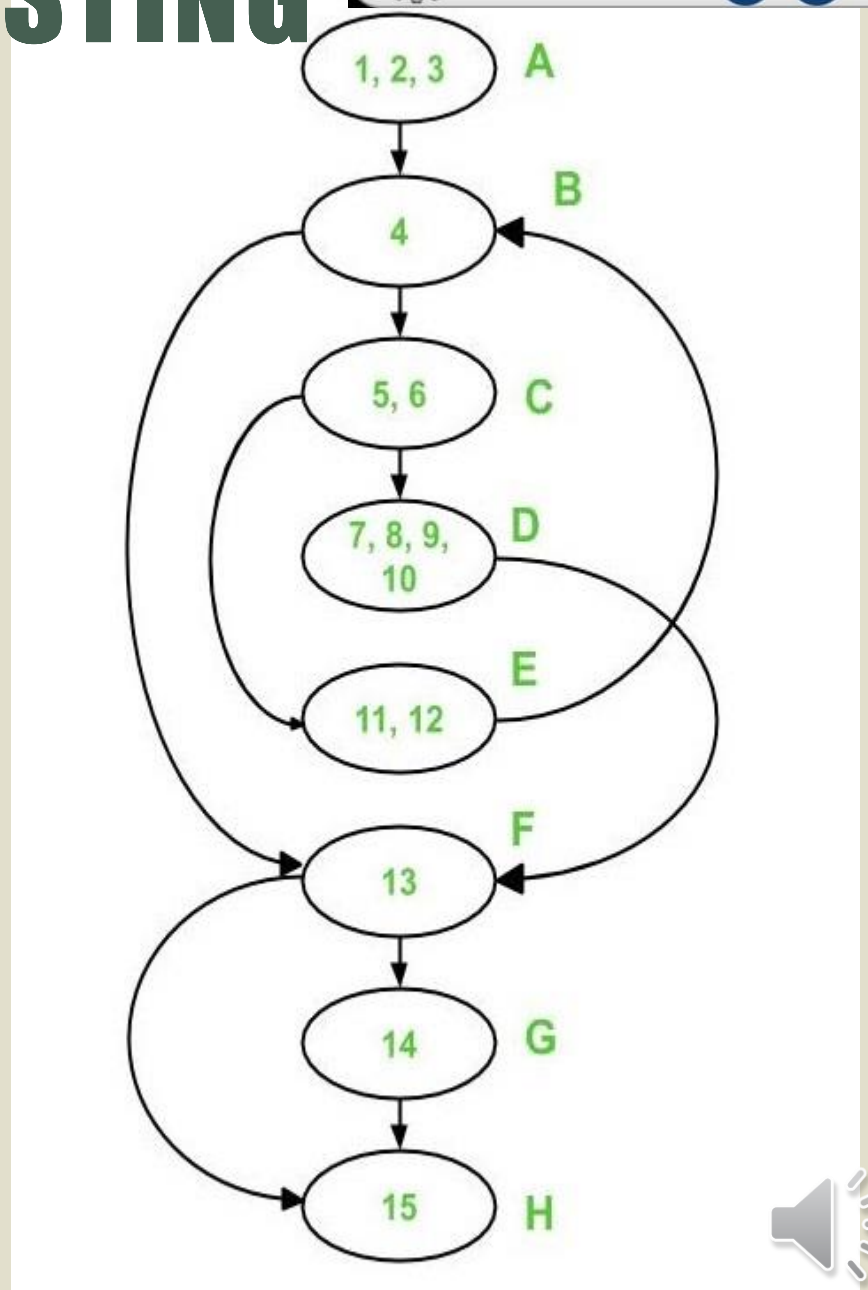
SOFTWARE TESTING FUNDAMENTALS

LECTURE # 33

Lecture by Engr.Sidra

EXAMPLE OF BASIS PATH TESTING

```
                int main()
        {
                int n, index;
1.              cout << "Enter a number: " << endl;
2.              cin >> n;
3.              index = 2;
4.              while (index <= n - 1)
5.          {
6.                          if (n % index == 0)
7.              {
8.                                      cout << "It is not a prime number" << endl;
9.                                      break;
10.             }
11.                         index++;
12.           }
13.             if (index == n)
14.                         cout << "It is a prime number" << endl;
15.       } // end main
```



EXAMPLE OF BASIS PATH TESTING

- E = 10, N = 8, Cyclomatic complexity is:

  V(G) = 10 – 8 + 2 = 4

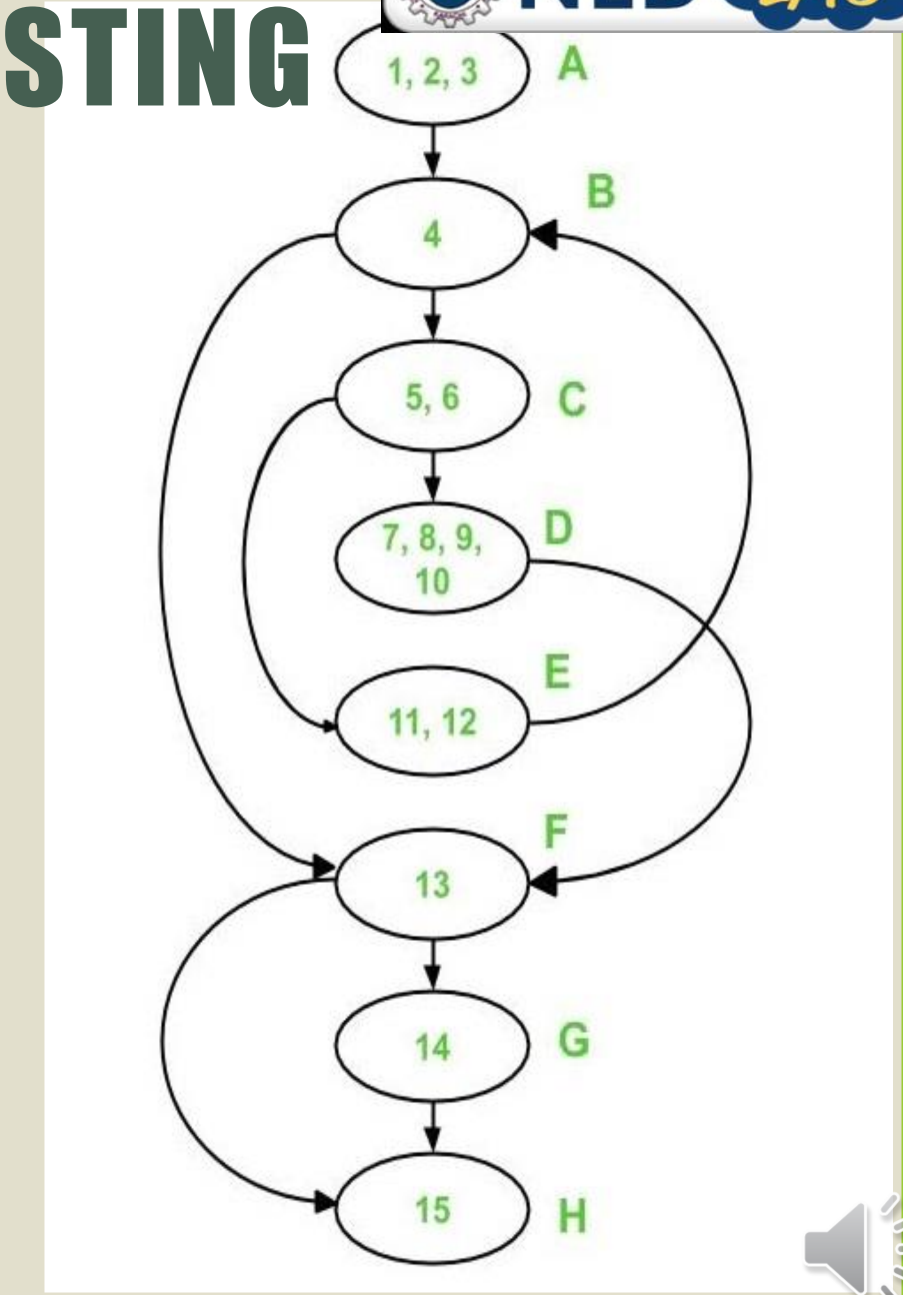
- Paths

  Path 1: A-B-F-G-H

  Path 2: A-B-F-H

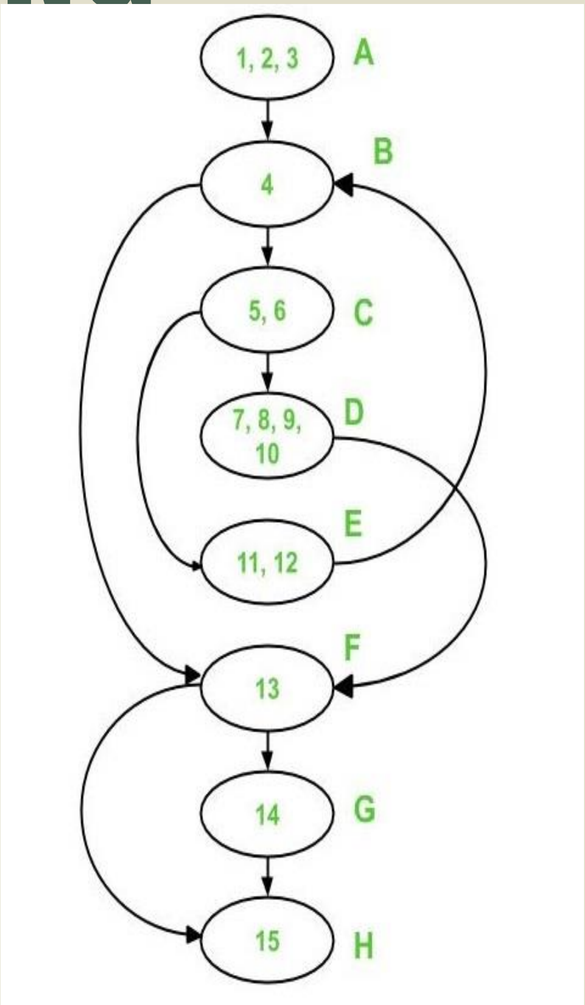
  Path 3: A-B-C-E-B-F-G-H

  Path 4: A-B-C-D-F-H



EXAMPLE OF BASIS PATH TESTING

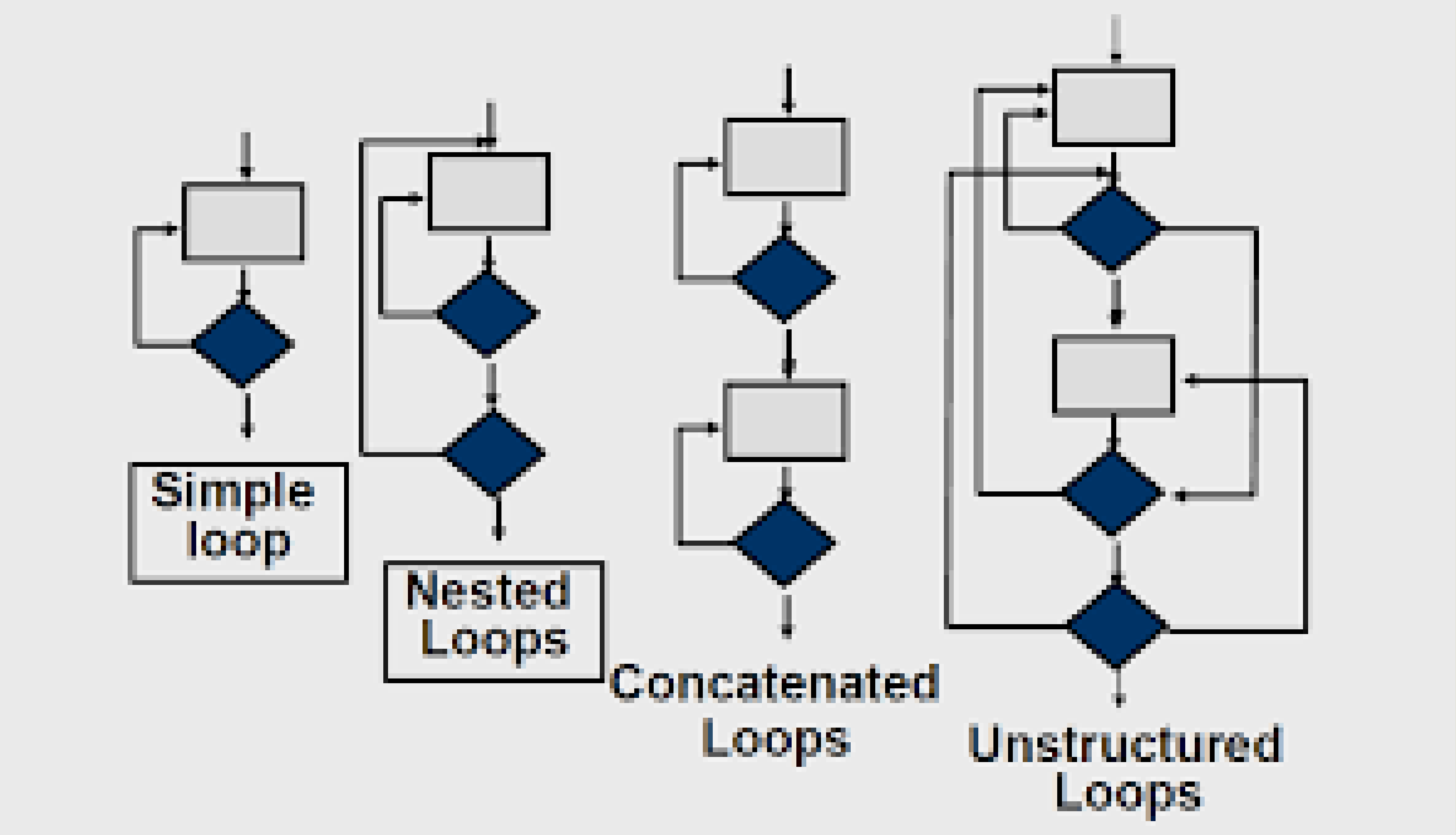
Test case ID	Paths	Test Data	Expected Result	Actual Result	Test Status
1	Path 1: A-B-F-G-H	2	It is a prime number		
2	Path 2: A-B-F-H	1	No output		
3	Path 3: A-B-C-E-B-F-G-H	3	It is a prime number		
4	Path 4: A-B-C-D-F-H	4	It is not a prime number		



CONTROL STRUCTURE TESTING

- Condition testing:
  - A test case design method that exercises the logical conditions contained in a program module
- Data flow testing:
  - Selects test paths of a program according to the locations of definitions and uses of variables in the program
- Loop Testing:
  - It completely focuses on the validity of the loop constructs.

LOOP TESTING



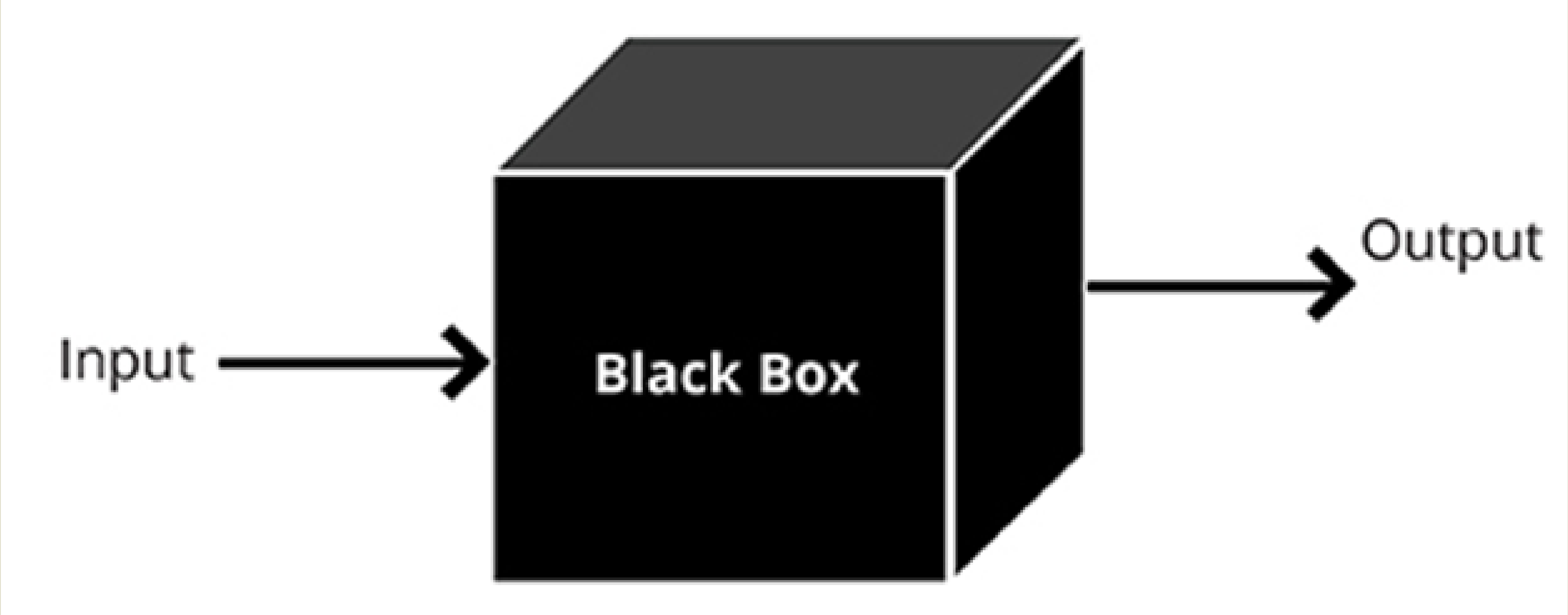
LOOP TESTING (SIMPLE LOOPS)

- Minimum Conditions
  - Skip the entire loop
  - Make 1 passes through the loop
  - Make 2 passes through the loop
  - Make m passes through the loop where m<n, n is the maximum number of passes through the loop
  - Make b, b-1; b+1 passes through the loop where "b" is the maximum number of allowable passes through the loop.

LOOP TESTING

- Nested Loops
  - Start at the innermost loop. Set all other loops to minimum values.
  - Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter (e.g., loop counter) values. Add other tests for out-of-range or excluded values.
  - Work outward, conducting tests for the next loop, but keeping all other outer loops at minimum values and other nested loops to "typical" values.
  - Continue until all loops have been tested.
- Concatenated Loops
  - Concatenated loops can be tested using the approach defined for simple loops, if each of the loops is independent of the other.
  - However, if two loops are concatenated and the loop counter for loop 1 is used as the initial value for loop 2, then the loops are not independent.
  - When the loops are not independent, the approach applied to nested loops is recommended.

BLACK BOX TESTING



EQUIVALENCE PARTITION

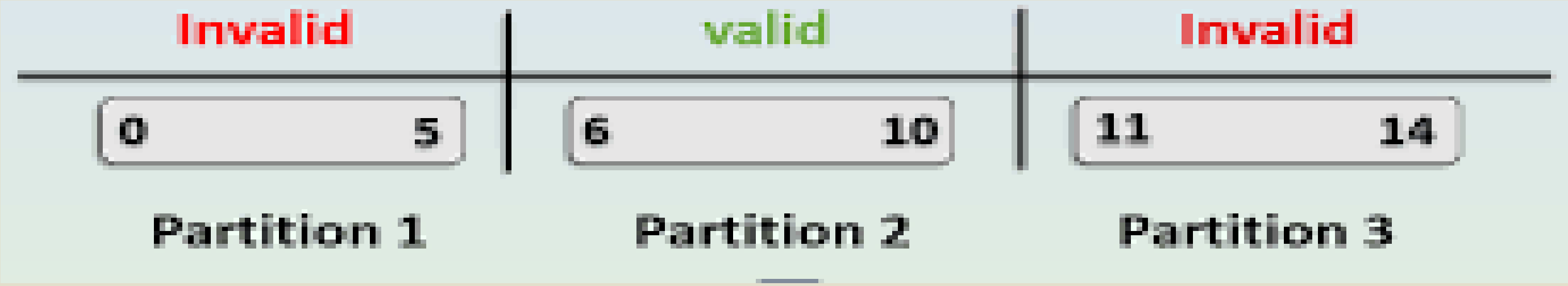
- Equivalent Class Partitioning is a black box technique (code is not visible to tester) which can be applied to all levels of testing like unit, integration, system, etc.
- In this technique, divide the set of test condition into a partition that can be considered the same.
- It divides the input data of software into different equivalence data classes.
- It can be applied, where there is a range in the input field.
- This method is typically used **to reduce the total number of test case**s to a finite set of testable test cases, still covering maximum requirements.

EQUIVALENCE PARTITION

- Equivalence classes may be defined according to the following guidelines:
  - If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
  - If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
  - If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
  - If an input condition is Boolean, one valid and one invalid class are defined.

EXAMPLE

- **Test cases for input box accepting numbers between 6 and 10 using Equivalence Partitioning:**
  - One input data class with all valid inputs. Pick a single value from range 6 to 10 as a valid test case. If other values between 6 and 10 is selected the result is going to be the same. So one test case for valid input data should be sufficient.
  - Input data class with all values below the lower limit. I.e. any value below 6, as an invalid input data test case.
  - Input data with any value greater than 10 to represent the third invalid input class.
  - So using Equivalence Partitioning you have categorized all possible test cases into three classes. Test cases with other values from any class should give you the same result.



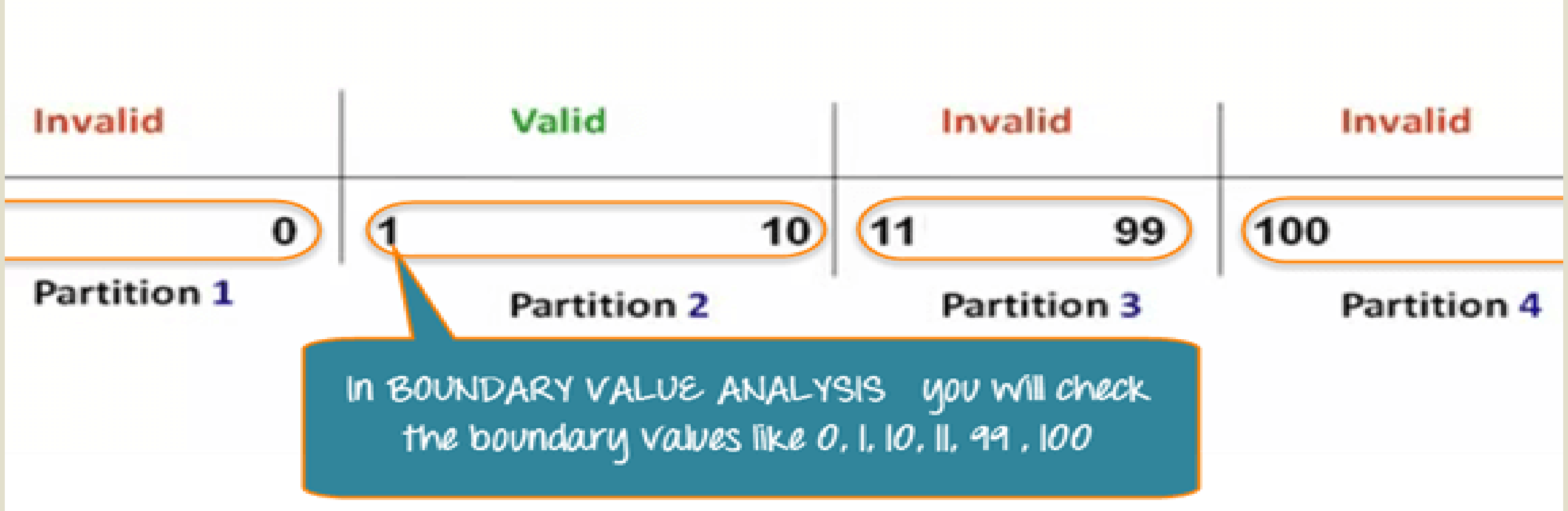
Invalid	valid	Invalid
0 5	6 10	11 14
Partition 1	Partition 2	Partition 3

BOUNDARY VALUE ANALYSIS

- It's widely recognized that input values at the extreme ends of the input domain cause more errors in the system.
- More application **errors occur at the boundaries** of the input domain.
- It is used to identify errors at boundaries rather than finding those that exist in the centre of the input domain.
- Boundary Value Analysis is often called as a part of the Stress and Negative Testing.
- The basic idea in boundary value testing is to select input variable values at their:
  - Minimum
  - Just above the minimum
  - A nominal value
  - Just below the maximum
  - Maximum

EXAMPLE

- **Test cases for input box accepting numbers between 1 and 10 using Boundary value analysis:**
- **#1)** Test cases with test data exactly as the input boundaries of input domain i.e. values 1 and 10 in our case.
- **#2)** Test data with values just below the extreme edges of input domains i.e. values 0 and 9.
- **#3)** Test data with values just above the extreme edges of the input domain i.e. values 2 and 11.



In BOUNDARY VALUE ANALYSIS you will check the boundary values like 0, 1, 10, 11, 99, 100

COMPARISON TESTING

- Used only in situations in which the reliability of software is absolutely critical (e.g., human rated systems)
  - Separate software engineering teams develop independent versions of an application using the same specification
  - Each version can be tested with the same test data to ensure that all provide identical output
  - Then all versions are executed in parallel with real time comparison of results to ensure consistency