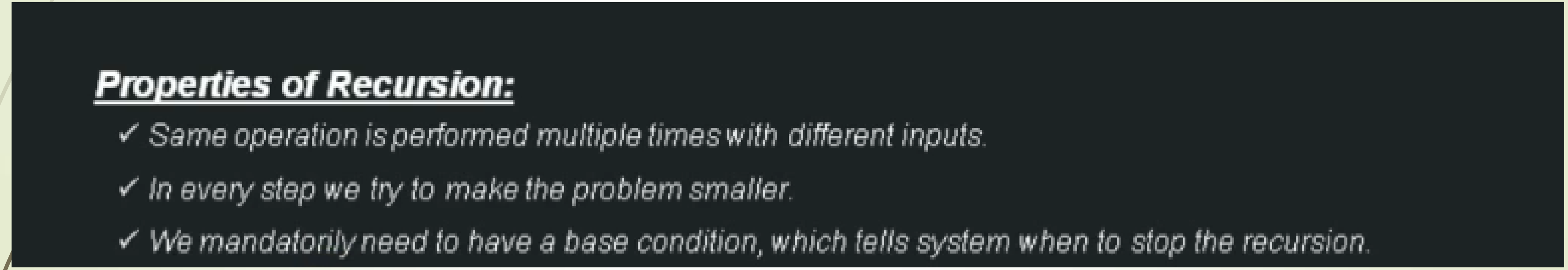
Data Structure and Algorithms

Session-9

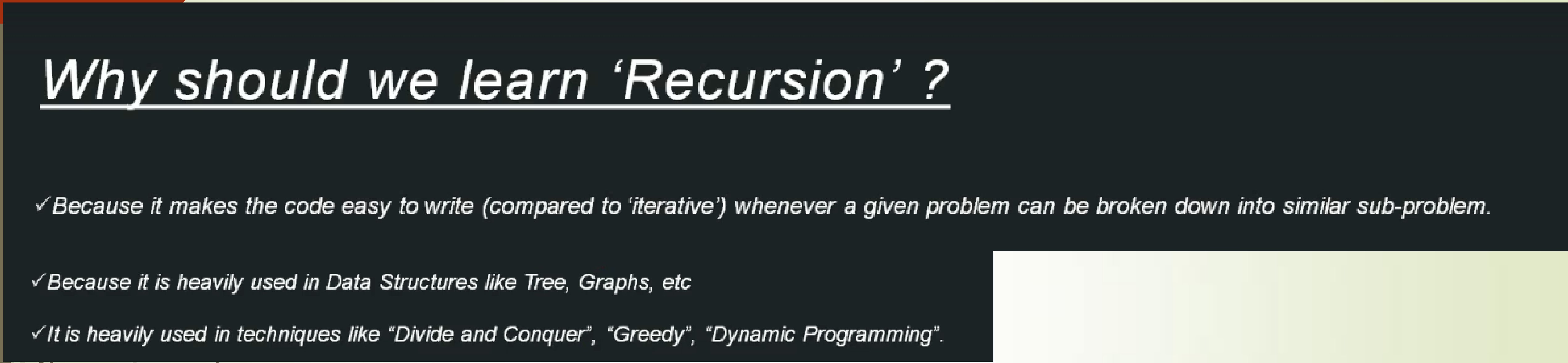
Dr. Subhra Rani Patra
SCOPE, VIT Chennai

What is ‘Recursion’ ?



Properties of Recursion:

- ✓ *Same operation is performed multiple times with different inputs.*
- ✓ *In every step we try to make the problem smaller.*
- ✓ *We mandatorily need to have a base condition, which tells system when to stop the recursion.*

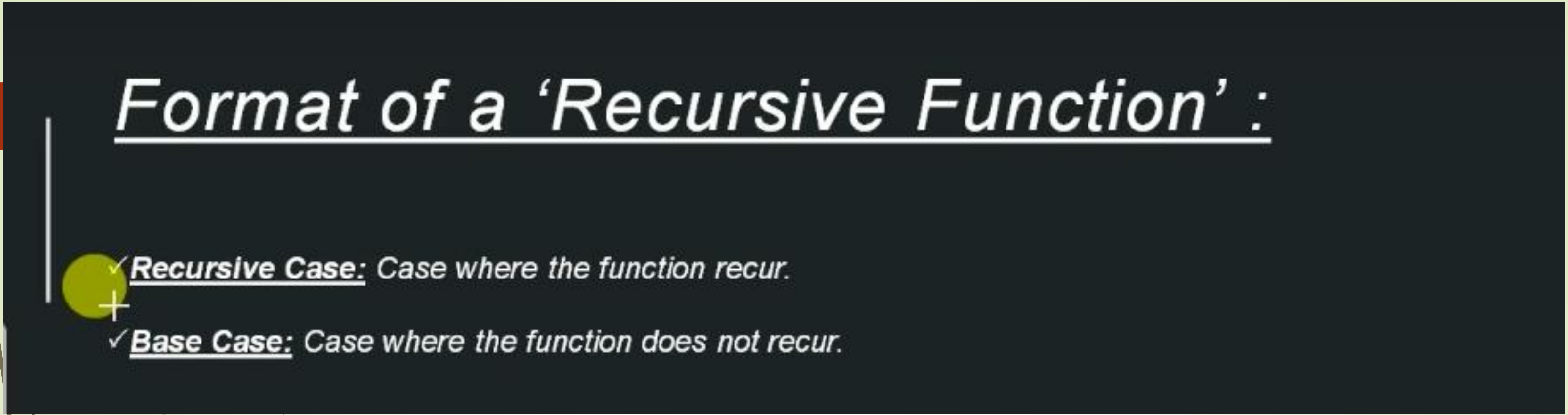


Why should we learn ‘Recursion’ ?

✓*Because it makes the code easy to write (compared to ‘iterative’) whenever a given problem can be broken down into similar sub-problem.*

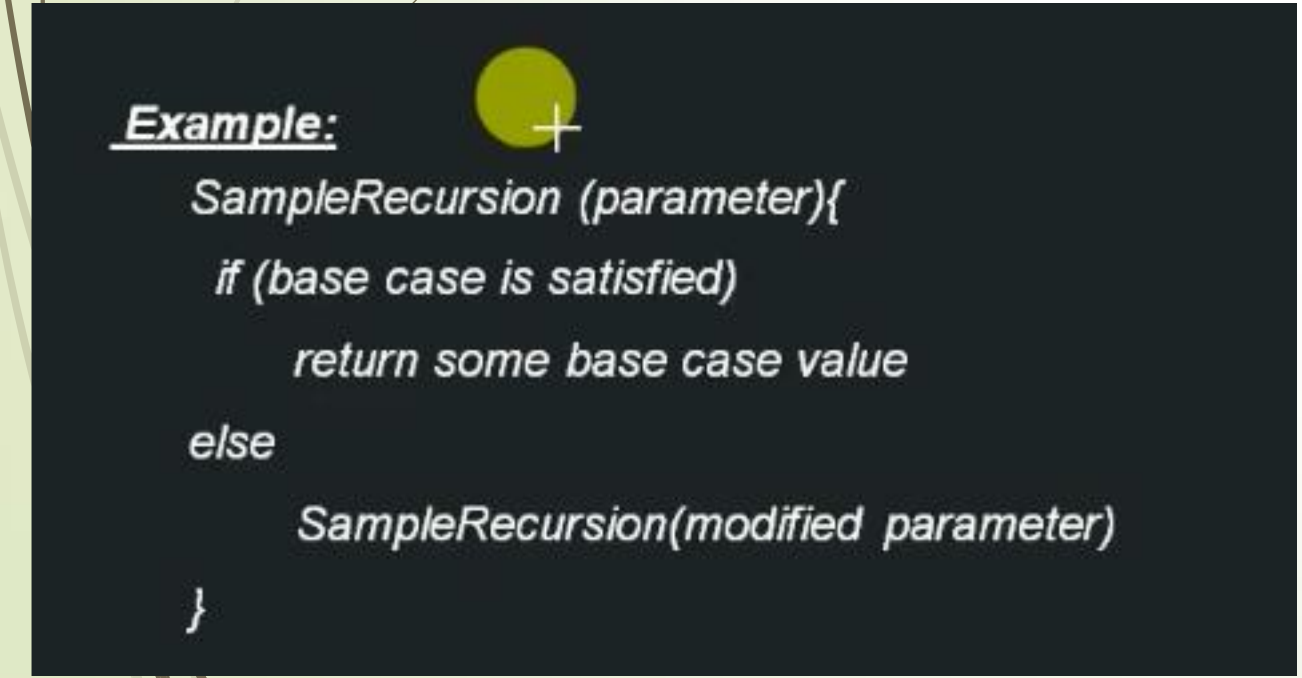
✓*Because it is heavily used in Data Structures like Tree, Graphs, etc*

✓*It is heavily used in techniques like “Divide and Conquer”, “Greedy”, “Dynamic Programming”.*



Format of a ‘Recursive Function’ :

- ✓***Recursive Case:*** *Case where the function recur.*
- ✓***Base Case:*** *Case where the function does not recur.*

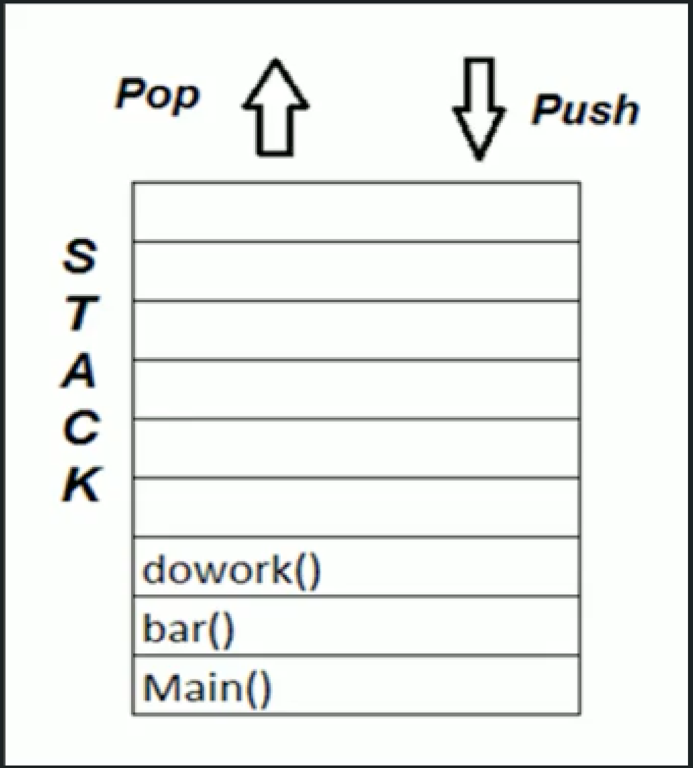


Example:

```
SampleRecursion (parameter){
 if (base case is satisfied)
       return some base case value
else
       SampleRecursion(modified parameter)
}
```

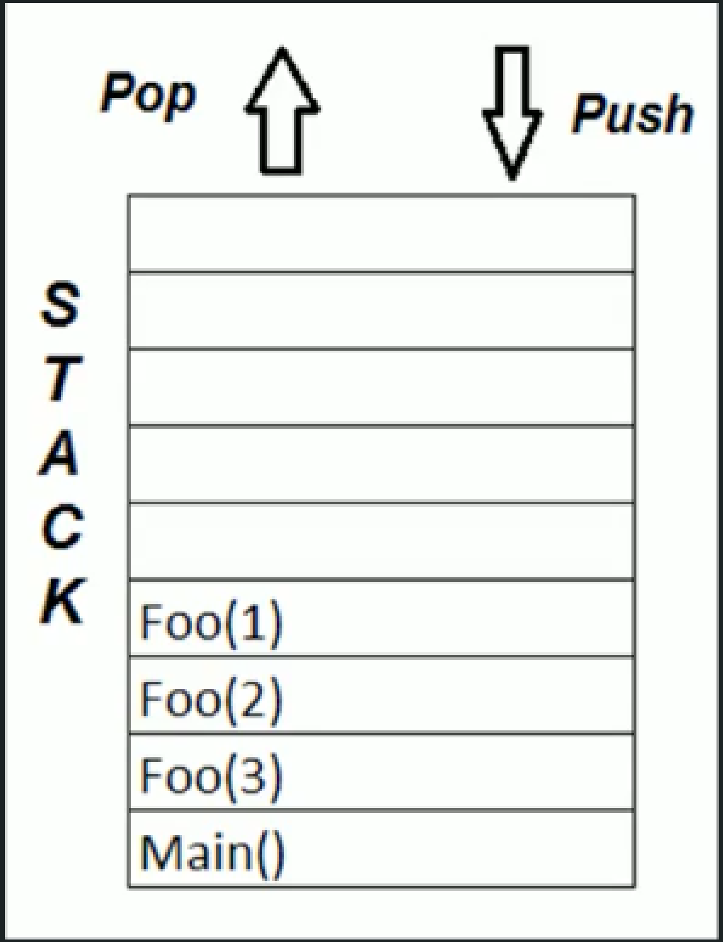
How ‘Recursion’ works internally ?

```
Main()

    Bar();

    System.out.println(“I  am in Main”)


Bar()

   DoWork()

  System.out.println(“I  am in Bar”)



DoWork(){

  DoMore()

  System.out.println(“I  am in DoWork”)



DoMore(){

   System.out.println(“I  am in DoMore”)
```



How ‘Recursion’ works internally ?

```
foo(n){
    If (n<1)
        return
    else
        foo(n-1)
    print “Hello World” + n
}

Main(){
    foo(3)
}
```

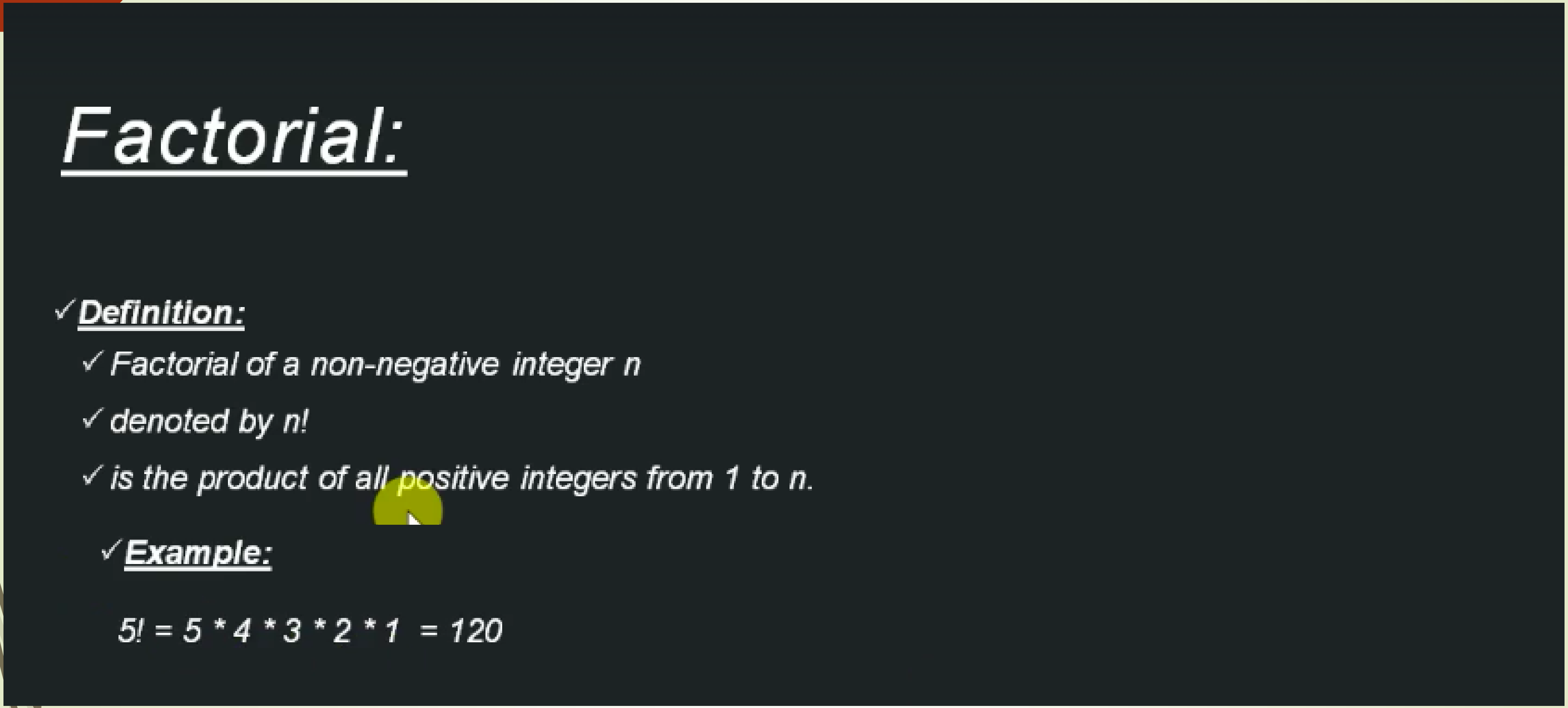


Foo(1)

Foo(2)

Foo(3)

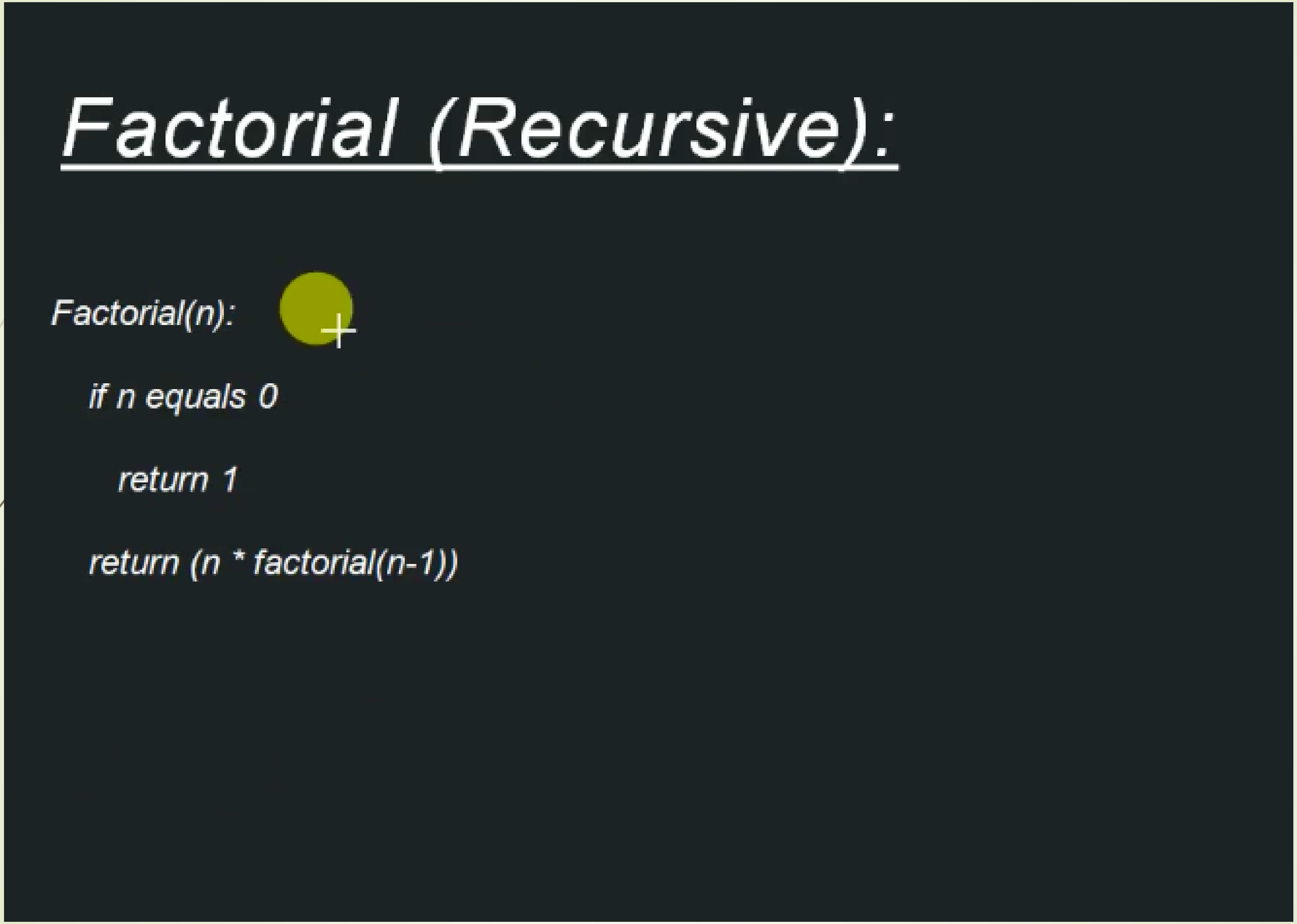
Main()



Factorial:

- ✓ **Definition:**
  - ✓ Factorial of a non-negative integer n
  - ✓ denoted by n!
  - ✓ is the product of all positive integers from 1 to n.
- ✓ **Example:**

5! = 5 * 4 * 3 * 2 * 1 = 120

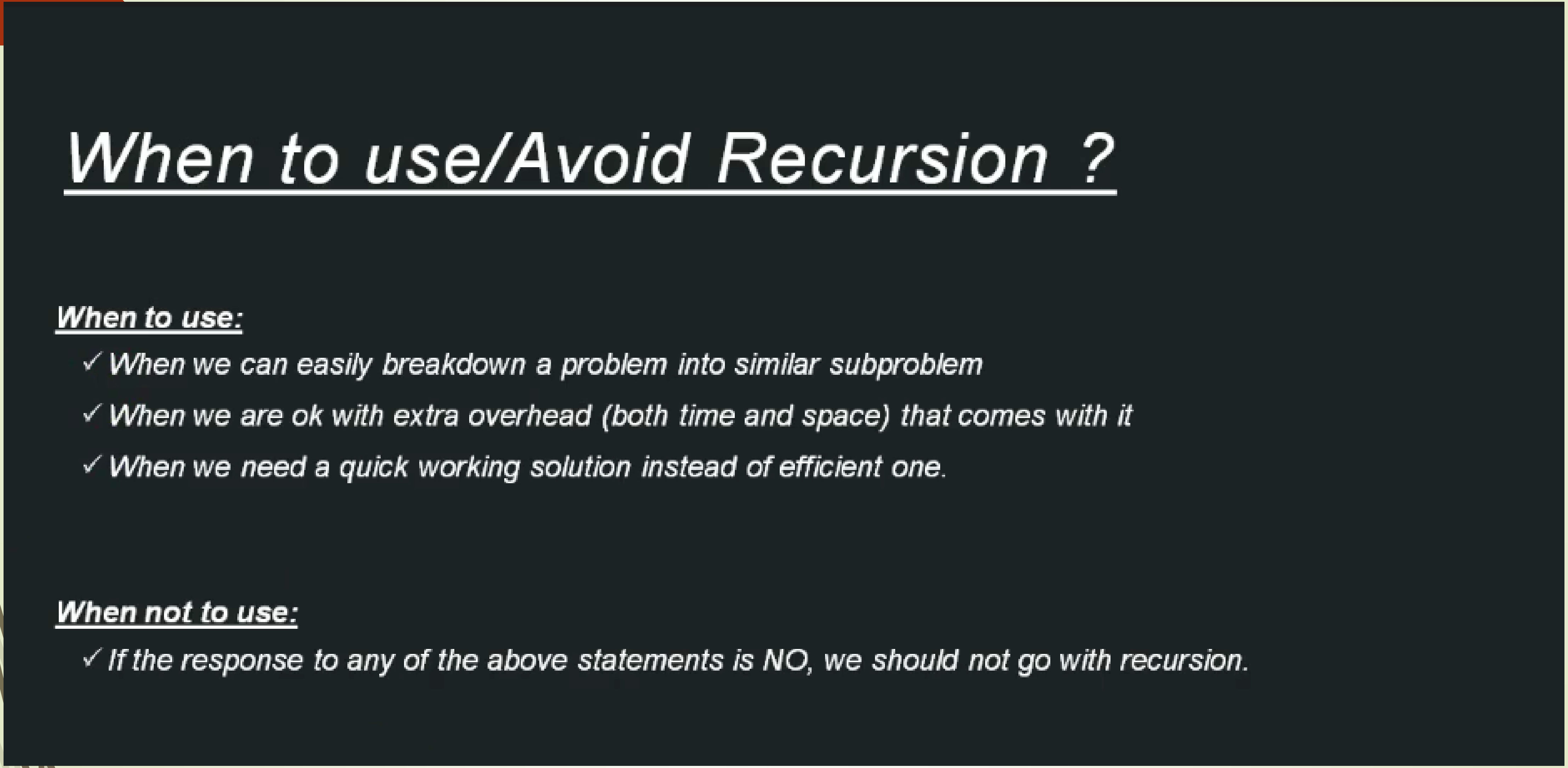


Factorial (Recursive):

```
Factorial(n):
  if n equals 0
    return 1
  return (n * factorial(n-1))
```

Recursion vs Iteration:

Particulars	Recursion	Iteration
Space efficient ?	*No*	*Yes*
Time efficient ?	*No*	*Yes*
Ease of code (to solve sub-problems) ?	*Yes*	*No*



When to use/Avoid Recursion ?

When to use:

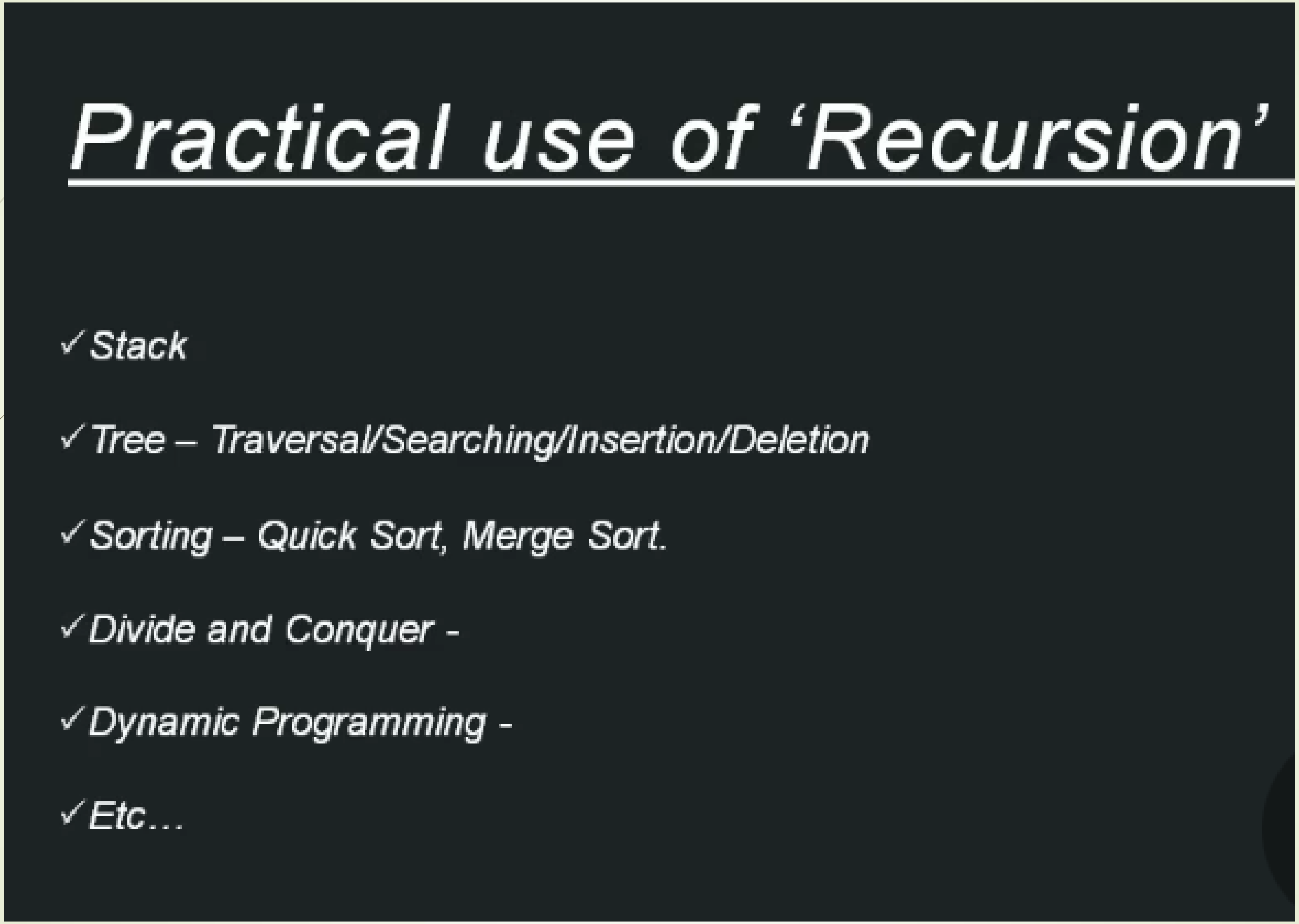
✓ *When we can easily breakdown a problem into similar subproblem*

✓ *When we are ok with extra overhead (both time and space) that comes with it*

✓ *When we need a quick working solution instead of efficient one.*

When not to use:

✓ *If the response to any of the above statements is NO, we should not go with recursion.*



Practical use of ‘Recursion’

- ✓Stack
- ✓Tree – Traversal/Searching/Insertion/Deletion
- ✓Sorting – Quick Sort, Merge Sort.
- ✓Divide and Conquer -
- ✓Dynamic Programming -
- ✓Etc…

Thank you