Working with Functions and Closures

Harit Himanshu

@harittweets

Overview

Understanding Local Functions

Understanding Function Literal and Function Value

Understanding Partially Applied Functions

Understanding Closures

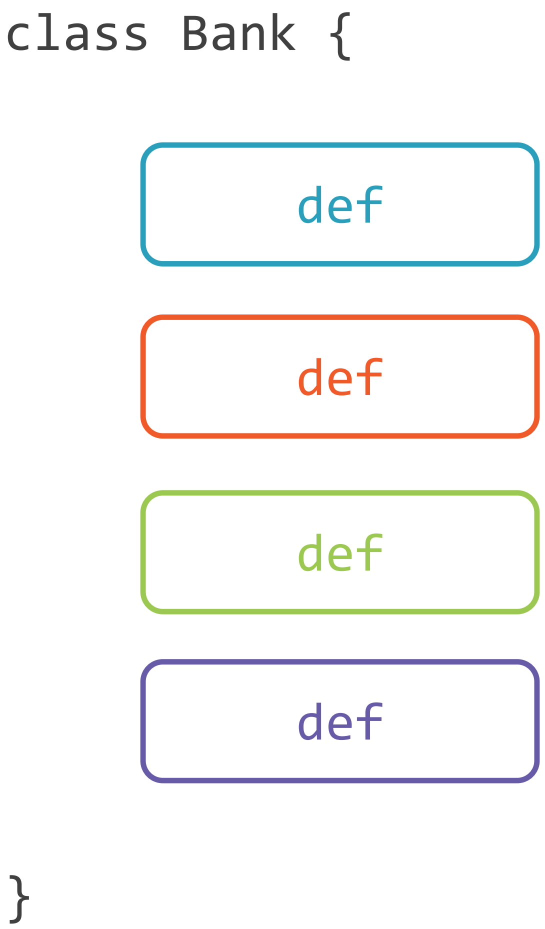
Applying Repeated Arguments to Functions

Understanding Named Arguments and Default Parameter Values

Understanding Tail Recursion

Project Demo

Class Design with Methods



```
class Bank {

    def

    def

    def

    def

}
```

Design Recommendations for Methods

Multiple Small Functions

Program

Small and Simple

Function

Compose complex behaviors

Programmers

Local Functions

Nested inside Function

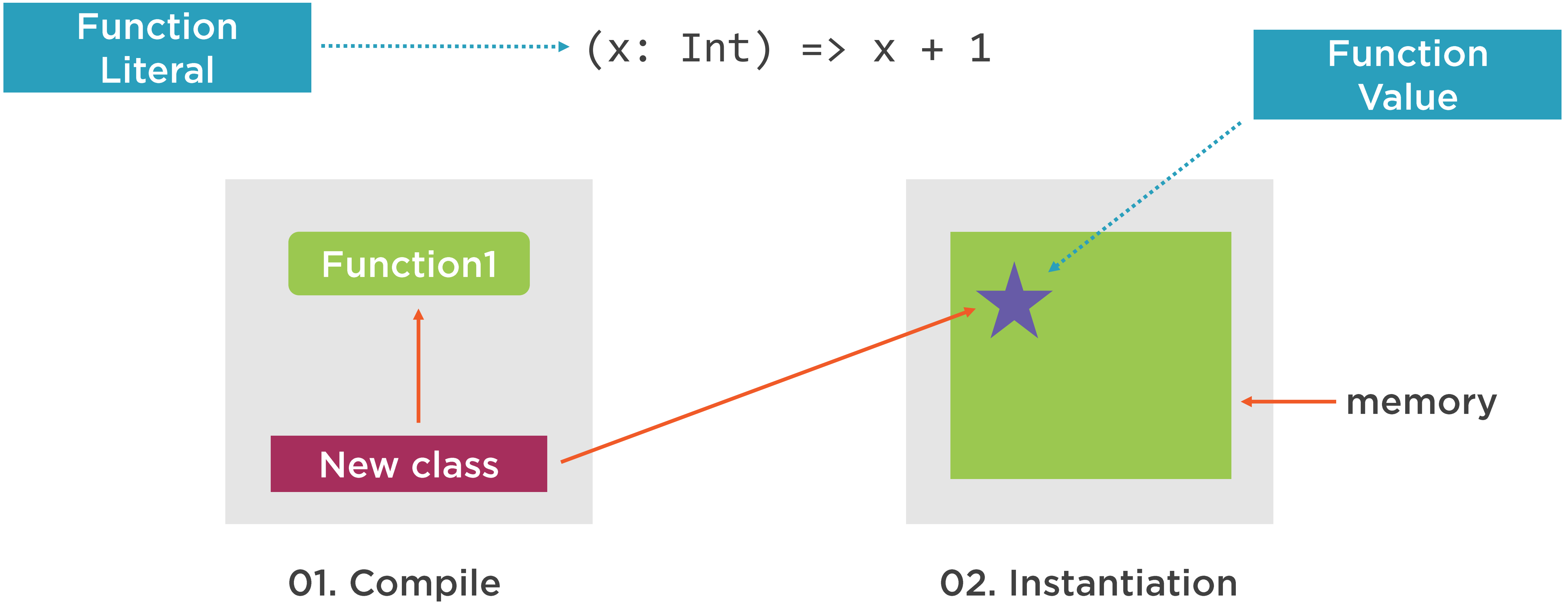
Parameter Access of Parent Function

Not visible outside Parent Function

```
def increment(x:Int) ⇒ x + 1
```

Structure of a Function

Function Literal to Function Value



```
def sum(a: Int, b: Int, c: Int) = a + b + c // declare

sum(1, 2, 3) // execute
```

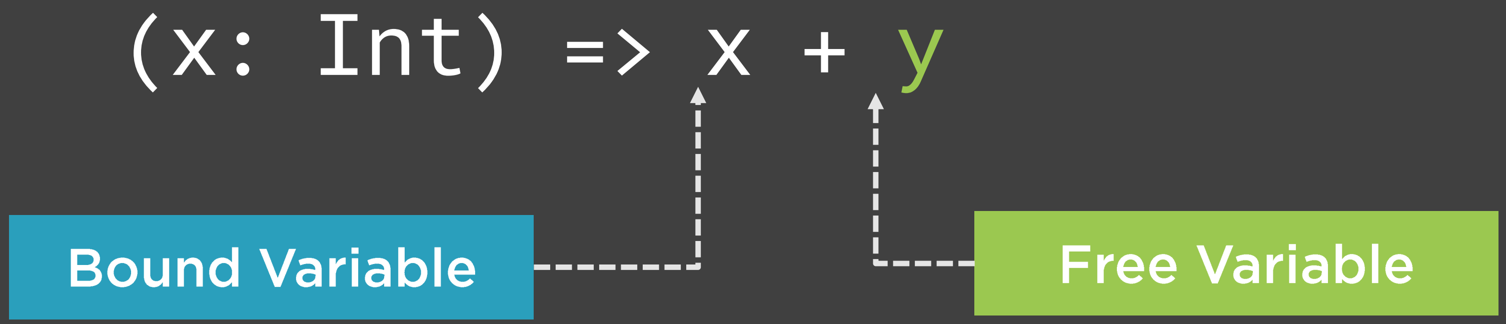
Declaring and Executing Functions

Partially Applied Functions

```
def sum(a: Int, b: Int, c: Int) = a + b + c


sum(1, 2, <not supplied>)

sum(1, <not supplied>, 3)

sum(<not supplied>, 2, 3)

sum(<not supplied>, <not supplied>, <not supplied>)
```

Function Literal Forms

```
(x: Int) => x + 1
```

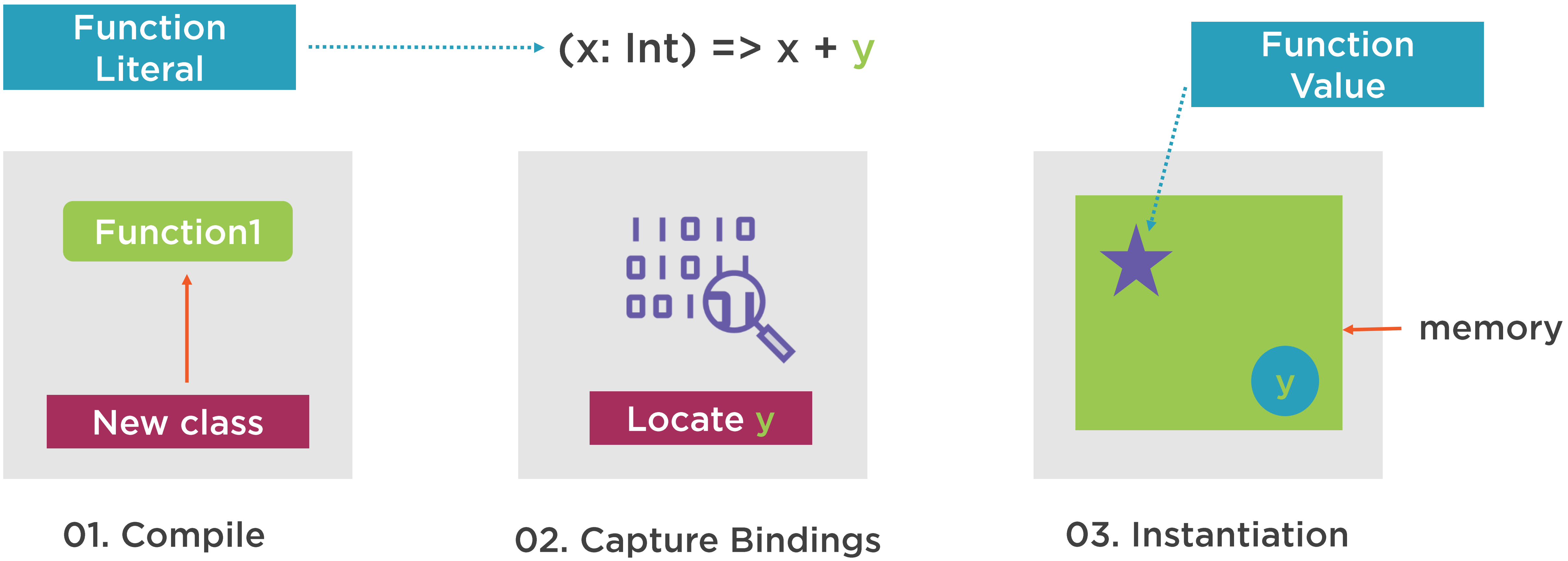


```
(x: Int) => x + y
```

Bound Variable

Free Variable

Function Literal to Function Value



Function Literal

(x: Int) => x + y

Function Value

Function1

New class

Locate y

y

memory

01. Compile

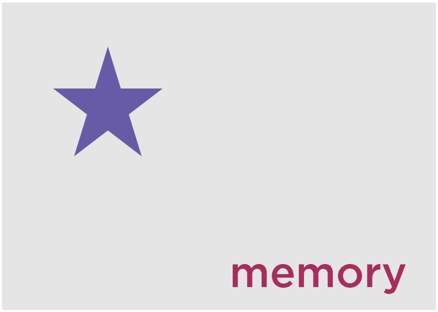
02. Capture Bindings

03. Instantiation

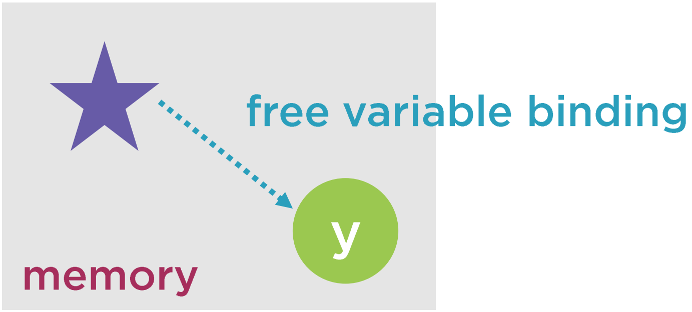
Function Value vs. Closure

```
(x: Int) => x + 1
```

```
(x: Int) => x + y
```



Function Value



Closure

Closure with Free Variable in Local Variables

```
def multiplier(factor: Int) = (x: Int) => x * factor
```

double

```
multiplier(2) => (x: Int) => x * 2
```

triple

```
multiplier(3) => (x: Int) => x * 3
```

Recursion occurs when a thing is defined in terms of itself or of its type

https://en.wikipedia.org/wiki/Recursion

Recursion Properties

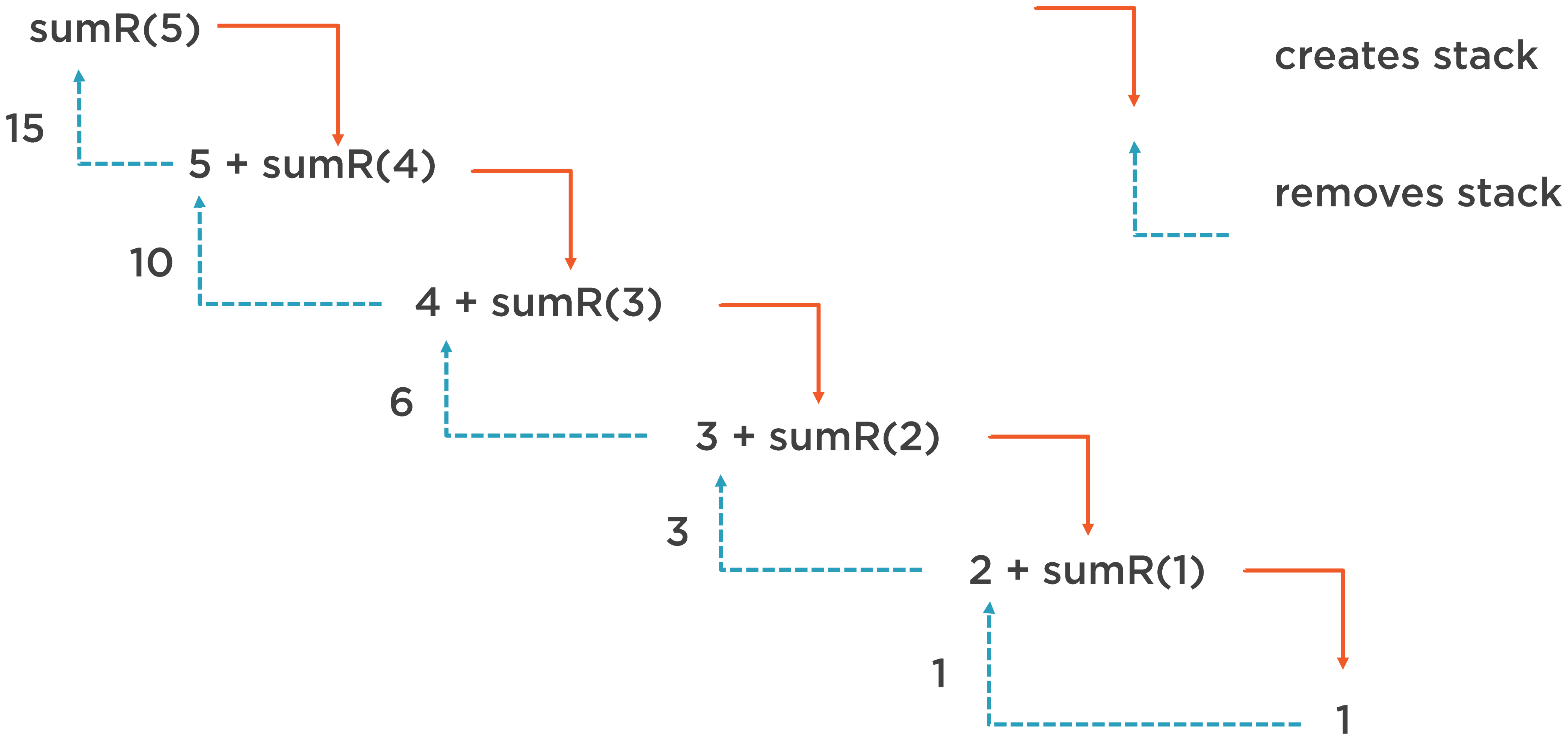
Base Case

Terminating condition

Recurs to reduce

Breaks problem in smaller chunks

Recursion Example



sumR(5)

15

5 + sumR(4)

10

4 + sumR(3)

6

3 + sumR(2)

3

2 + sumR(1)

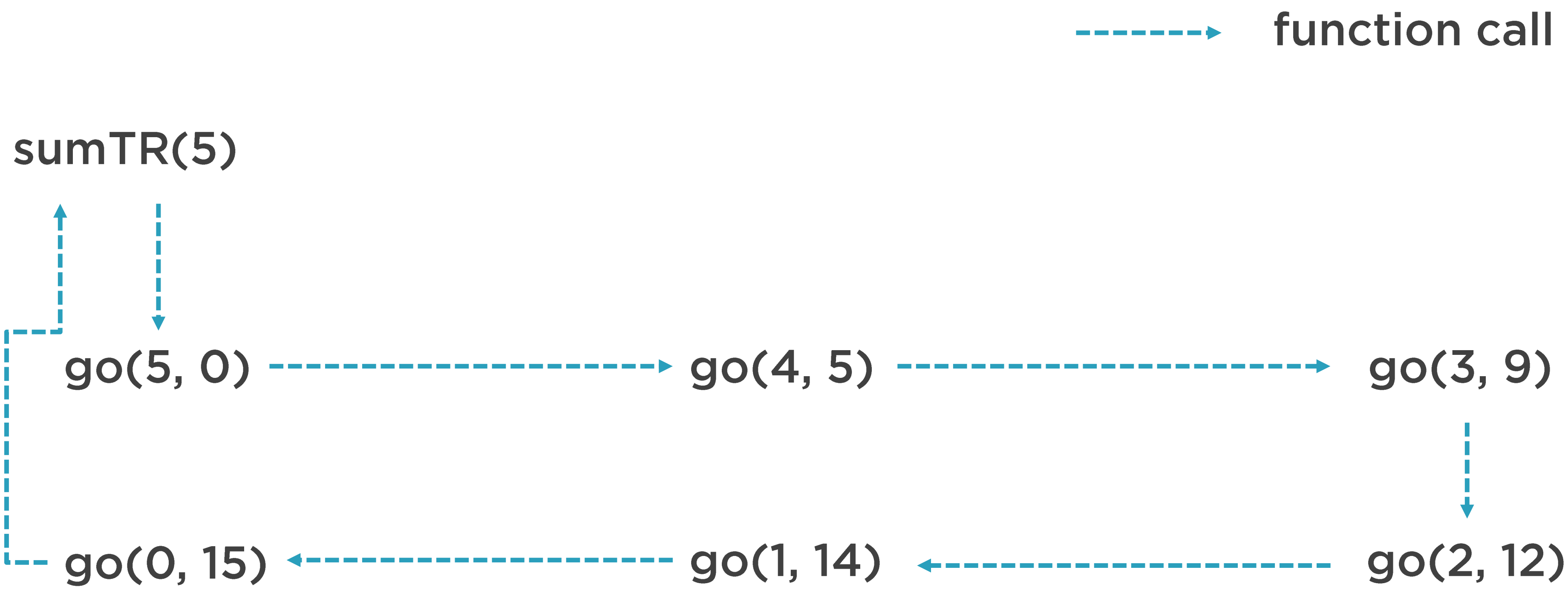
1

1

creates stack

removes stack

Tail Recursion Example



Demo

Giving *unique id* to Account, Customer and Product

Making consistent usage of Dollars

Giving more responsibilities to Bank

Summary

Understanding Local Functions

Understanding Function Literal and Function Value

Understanding Partially Applied Functions

Understanding Closures

Applying Repeated Arguments to Functions

Understanding Named Arguments and Default Parameter Values

Understanding Tail Recursion

Project Demo