Computer Architecture

“In computer science space everything starts with understanding how does computer work”

Chapter 1 : Digital Information

- How computers store information digitally, as binary data.
- Learn about the binary number system
- The conversion of analog to digital data
- data compression strategies.

How do computers represent data?

Human

images

text and symbols

Languages

Computer

```
01000111010010010100011000111000001110010110000100000001
00000000000000010000000010000000000000000000000000000000
00000000000000001111111111111111111111110010000111111001
00000100000000010000000000000000000000000000000000101100
00000000000000000000000000000000000000010000000000000001
00000000000000000000001000000001010001000000000000111011
```

1’s and 0’s

Addition command

Human

1 + 2 = 3

Computer

00110001 00101011 00110010 00111101 00110011

How did I
Translated
Human Language to
Computer Language
?

Symbols And ASCII Binary

https://montcs.bloomu.edu/Information/Encodings/ascii-7.html

Bits ?
Bytes ?
Decimal Number System?
Binary Number System?

Bit

- Computer stores information using bits.
- A bit stores either the value 0 or 1.
- A sequence of two bits can represent four (2^2) distinct values 00,01,10,11
- A sequence of three bits can represent eight (2^3) different values 000,001,010,011,100,101,110,111

Byte

A byte represents different types of information depending on the context.
It might represent a

- Number,
- Letter
- Program instruction.
- Part of an audio recording
- Pixel in an image.

- The byte is also the smallest addressable unit of memory in most modern computers.
- A computer with byte-addressable memory can not store an individual piece of data that is smaller than a byte

Memory

Address	Data
1	11110000
2	00001010
3	00100110
4	11000001

Decimal Number System

- *As a human we represent numbers in decimal number system 1,2,3,4,5,6,7,8,9,10..*
- *I decimal system we multiply the digit by power of 10*

234

100's	10's	1's
Hundreds place	Tens place	Ones place
10^2	10^1	10^0
2	3	4

$(2 \times 100) + (3 \times 10) + (4 \times 1)$

or

$(2 \times 10^2) + (3 \times 10^1) + (2 \times 10^0)$

Binary Number System

- *I binary number system we multiply the digit by power of 2*

Largest Decimal Number

2 bit

2^1	2^0
2	1

→ $3(2^2 - 1)$

4 bit

2^3	2^2	2^1	2^0
8	4	2	1

→ $15\ (2^4 - 1)$

8 bit

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

→ $255\ (2^8 - 1)$

16 bit

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

→ $65535\ (2^{16} - 1)$

Convert Decimal To Binary Examples

1

32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

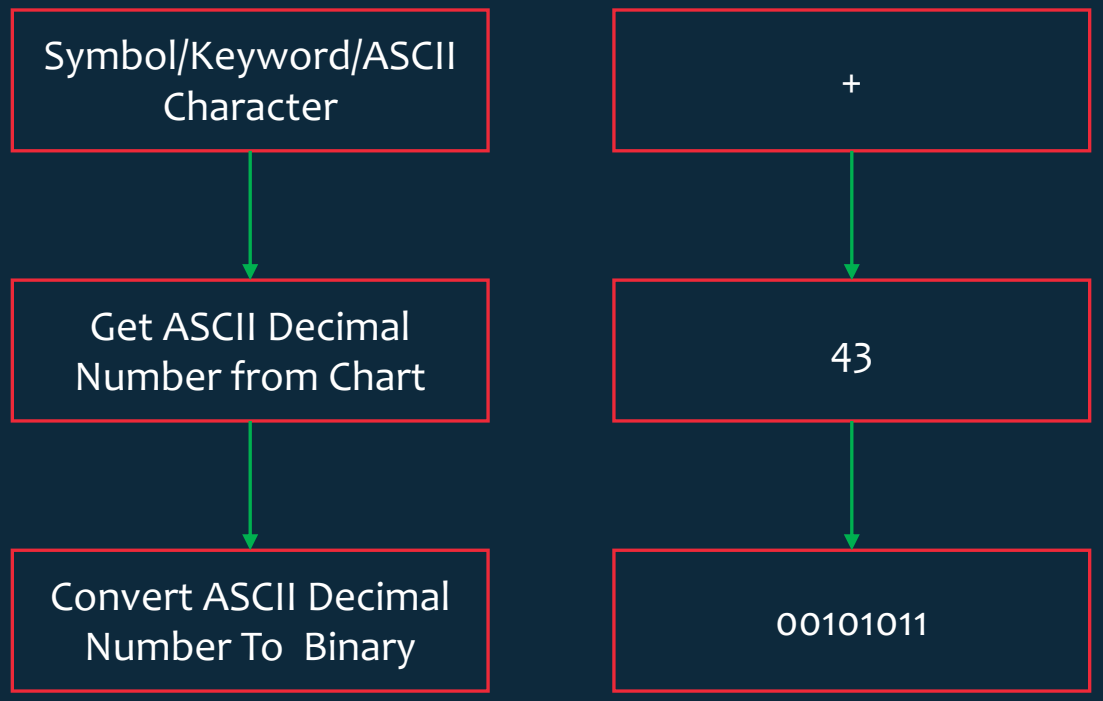
25

32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1

Steps

1) Mark all the values with 0's starting from 1 till 32768 in the above table.
2) 25 is less than 32 and greater than 16 . So mark 16 with 1 and do 25 – 16 = 9(reminder)
3) 9 is less than 16 and greater than 8 . So mark 8 with 1 and do 9 – 8 = 1 (reminder)
4) 1 is less than 2 and greater or equal to 1. so mark 1 with 1 and do 1 – 1 = 0(reminder)

Convert Symbol To ASCII Number To Binary Examples



Convert CHEF To 7 bit binary.

1000011 1001000 1000101 1000110

Limitations with ASCII Encoding

- The first big problem is that ASCII only includes letters from the English alphabet and a limited set of symbols.
- what about a text file with characters from multiple languages?
- "Hello, José, would you care for Glühwein? It costs 10 €“
- And what about languages with thousands of logograms? ASCII could not encode enough logograms to cover a Chinese sentence like "你好，想要一盘饺子吗？十块钱。"
- The other problem with the ASCII encoding is that it uses 7 bits to represent each character, whereas computers typically store information in bytes—units of 8 bits—and programmers don't like to waste memory.

How to overcome this limitation?

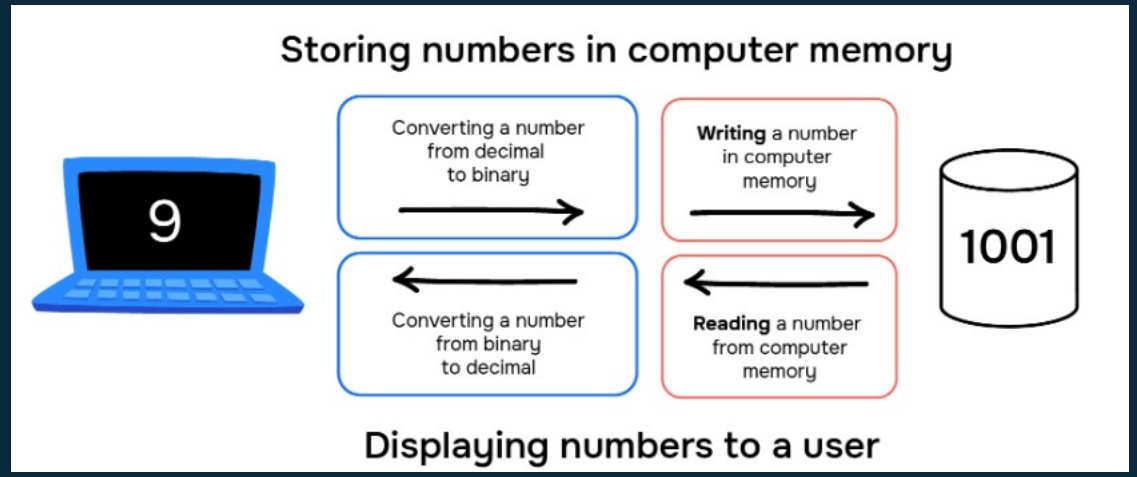
Before that lets understand what the is Encoding?

What is Encoding?

- Googles Computing Definition: *convert (information or an instruction) into a particular form*.
  In everyday life, encoding is similar to translating a message from one language to another so that it can be understood by someone who speaks a different language.

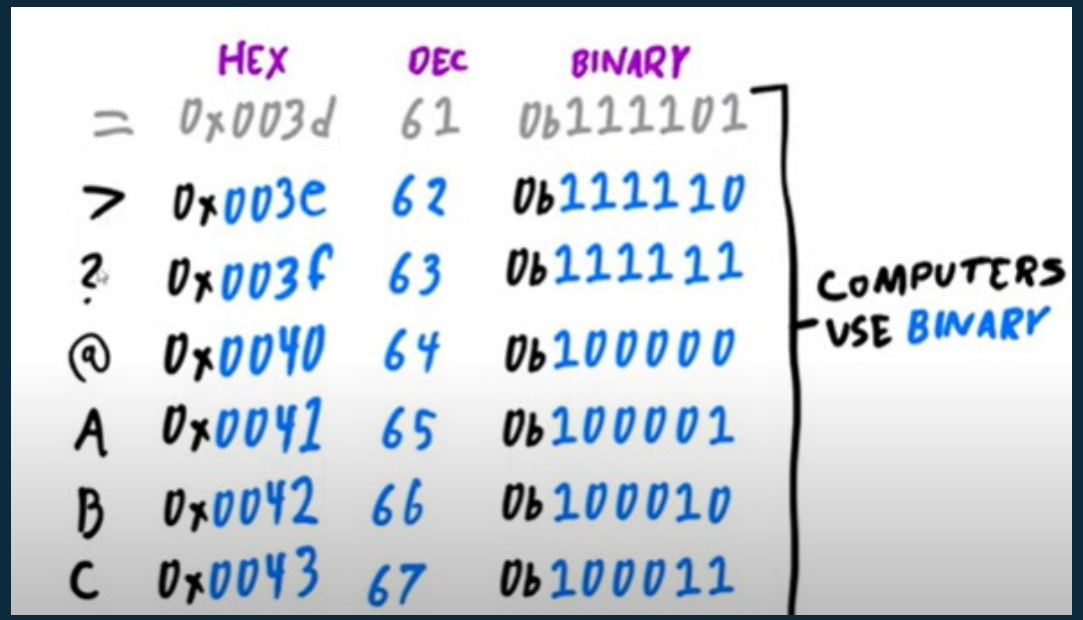
- ChatGPT Computing Definition: *Encoding, in simple terms, refers to the process of converting information or data from one format or representation to another. This conversion typically involves translating data into a specific code or format that can be easily understood, transmitted, stored, or processed by computer systems or other devices.*

  For example, when you type a message on your computer or phone, the characters you type are encoded into binary code (0s and 1s) before being stored or transmitted. Similarly, when you save a file, such as a photo or video, it is encoded into a specific file format (e.g., JPEG, MP4) that your device can understand.



Universal Symbol Table(Unicode)

Year	Characters		Version
1991	7161	The first version of the universal symbol table, called Unicode, was developed	1.0.0
1992	20,000	Chinese, Japanese, and Korean ideograms are added	1.0.1
current	143,000	https://en.wikipedia.org/wiki/List_of_Unicode_characters	15.1



Hex Decimal System

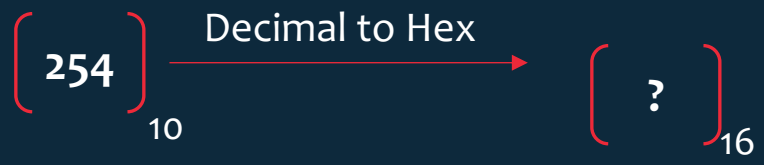
Hex Decimal System contains 16 distinct digits starting from 0 till F

0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7

8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Decimal To Hex Conversion

Example 1:



Divisor	Dividend		Quotient	Reminder	LSB or MSB	HEX DIGIT
16	254	16 X 15 = 240	15	14	LSB	E
16	15	16 X 0 = 0	0	15	MSB	F

MSB → LSB

FE$_{16}$

LSB → Least Significant Bit MSB → Most Significant bit

Decimal To Hex Conversion

Example 2:

$(1234)_{10}$ Decimal to Hex → $(?)_{16}$

Divisor	Dividend		Quotient	Reminder	LSB or MSB	HEX DIGIT
16	1234	16 X 77 = 1232	77	2	3	2
16	77	16 X 4 = 64	4	13	2	D
16	4	16 X 0= 0	0	4	1	4

LSB ↑ MSB

$(4D2)_{16}$

Decimal To Hex Conversion

Example 2:

$(25.625)_{10}$ Decimal to Hex → $(?)_{16}$

Take integer part which is 25

Divisor	Dividend		Quotient	Reminder	LSB or MSB	HEX DIGIT
16	25	16 X 1 = 16	1	9	2	9
16	9	16 X 0 = 0	0	1	1	1

LSB ↑ MSB

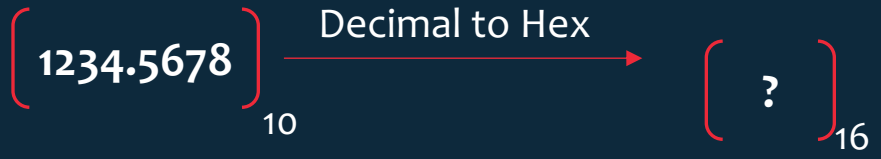
$(19)_{16}$

Take decimal part which is 0.625 and multiply it by 16 results is 10 hex digit for 10 is A

$(19.A)_{16}$

Decimal To Hex Conversion

Example 2:

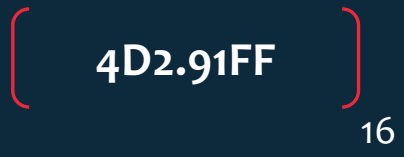


We already know hexadecimal number for 1234 is 4D2 as we already calculated

Lets calculate hexadecimal for precision value

- 0.5678 * 16 = 9.1248 -> 9 (whole number part) -> 9
- 0.1248 * 16 = 1.9968 -> 1 (whole number part) -> 1
- 0.9968 * 16 = 15.9488 -> F (whole number part) -> F
- 0.9488 * 16 = 15.1808 -> F (whole number part) -> F

So, the fractional part 0.5678 in hexadecimal is approximately 0.91FF.

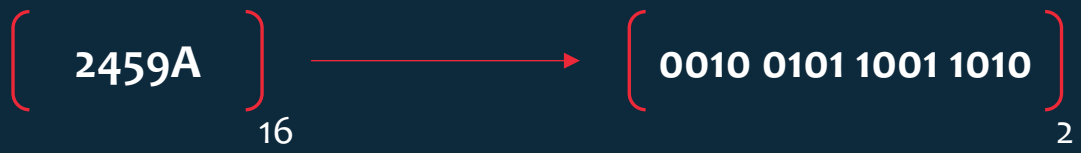


Hexadecimal To Binary

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110

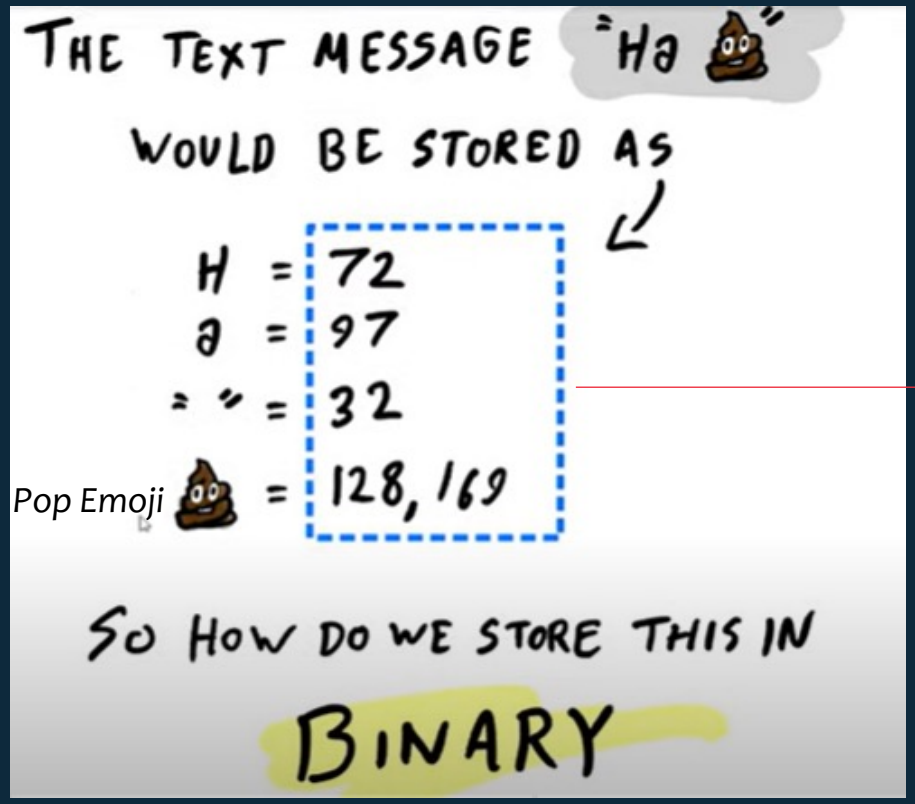
Decimal	Hexadecimal	Binary
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101

Decimal	Hexadecimal	Binary
14	E	1110
15	F	1111



UTF – Unicode Transformation Format

Example



Unicode code points:

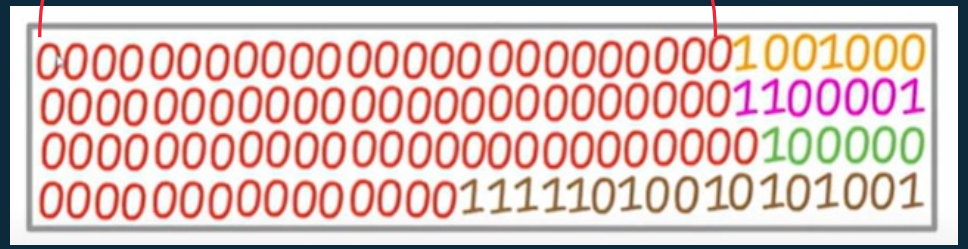
A Unicode code point is a numerical value, typically expressed in hexadecimal, that uniquely identifies a character or a symbol in the Unicode standard. Each character in the Unicode standard is assigned a unique code point, which serves as an identifier for that character.

symbol	Decimal	Hex
H	72	U+0048
a	97	U+0061
“ ”(Space)	32	U+0020
Pop Emoji	128169	U+1F349

UTF-32

- Each character is represented as 32 bit binary number. Which means the boundary of an each and every character is 32 bits.
- Length of bits are fixed.
- Wastage of memory is high.

Wasting space with 0’s(The ones that are highlighted in red)



UTF - 16

- Before the year 2000 only 65,000 code points(symbols) are in use and they are considered as BMP(Basic Multilingual Plane), which covers code points from U+0000 to U+FFFF(Here U represents Unicode and FFFF or 0000 is Hex value). They require 2 bytes or 16-bits for each to store in memory.

- Variable width encoding.

- Pop emoji doesn't fit in the range of 65,000. So that's where **surrogate pairs** came into existence.

Lets understand surrogate pairs with and example lets take code point 128169 or U+1F349(Pop Emoji).

Step1 :

Subtract 0x10000 from the code point U+1F349:
U+1F349 - 0x10000 = 0xF349.

```
   0001 1111 0011 0100 1001   (0x1F349)
 - 0001 0000 0000 0000 0000   (0x10000)
-----------------------------------
   0000 1111 0011 0100 1001   (0xF349)
```

Step2 :

Convert the resulting value (0x0F349) to binary:
0xF349 in binary is **0000 1111 0011 0100 1001**.

Step3 :

Split the 20 bits into two 10-bit portions:
High surrogate (leading surrogate): The first 10 bits are 0000111100.
Low surrogate (trailing surrogate): The last 10 bits are 1001001001.

Step4 :

Add 0xD800 to the high surrogate and 0xDC00 to the low surrogate:

High surrogate: 0000111100 + 0xD800 = 1111 0011 1011 0110 (U+D83C).

```
Binary:
   0000 0000 0011 1100     (0x003C)
+  1101 1000 0000 0000     (0xD800)
-------------------------------
 1 1101 1000 0011 1100     (0xD83C)
```

Low surrogate: 1001001001 + 0xDC00 = 1101111001001001 (U+EE49).

```
Binary:        0000 0010 0100 1001    (0x0249)
              +1101 1100 0000 0000    (0xDC00)
              ------------------
Result:       1110 1110 0100 1001    (0xEE49)
```

the surrogate pairs for U+1F349 are U+D83C and U+EE49. These pairs are used together to represent the character "**pop emoji**" in UTF-16 encoding

Last thoughts on UTF-16

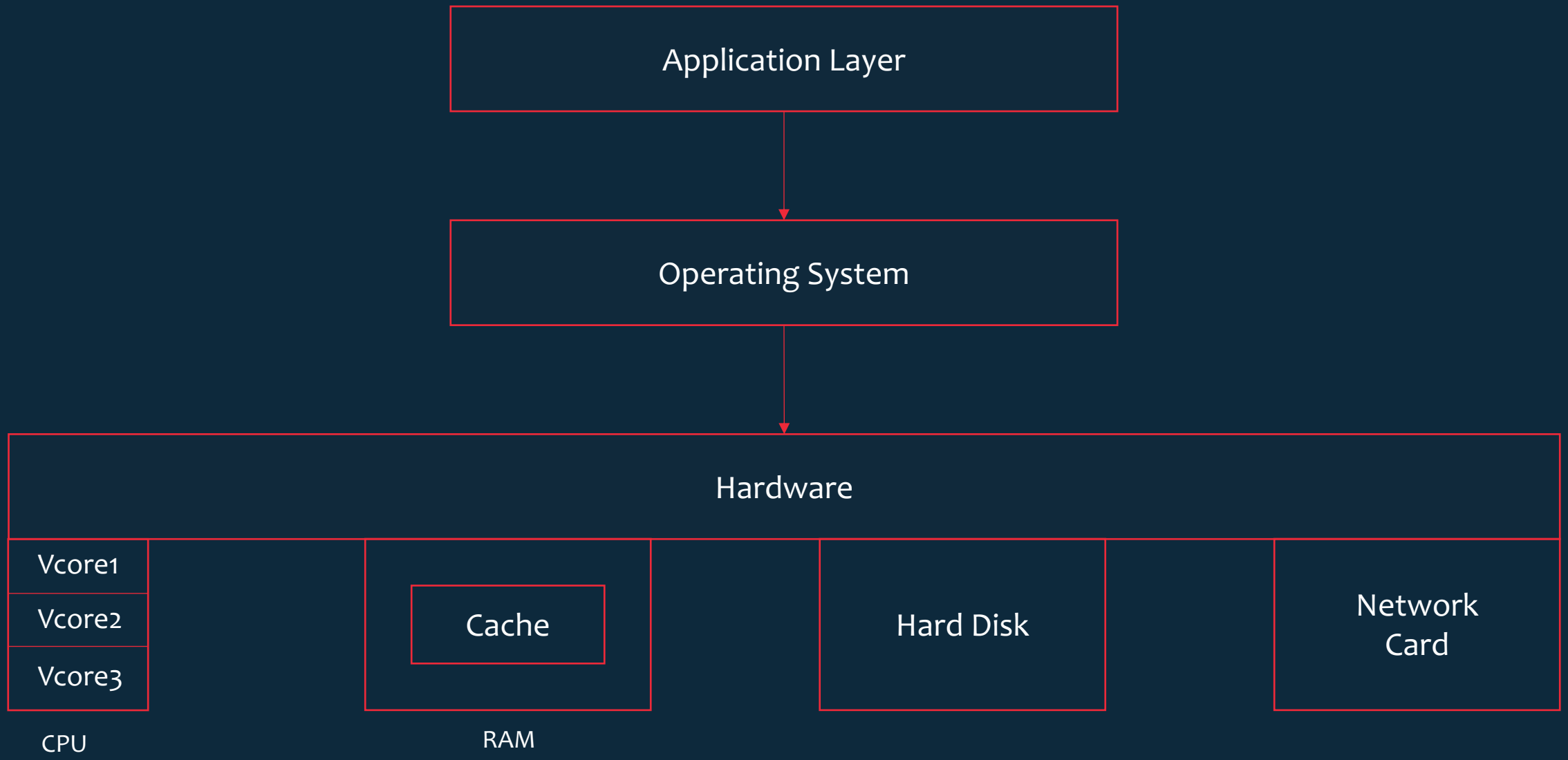
H
a
" space "
Pop emoji

Bad thing is we are still wasting space for 0's

Any code point that is greater than 65000 requires two 16 bits

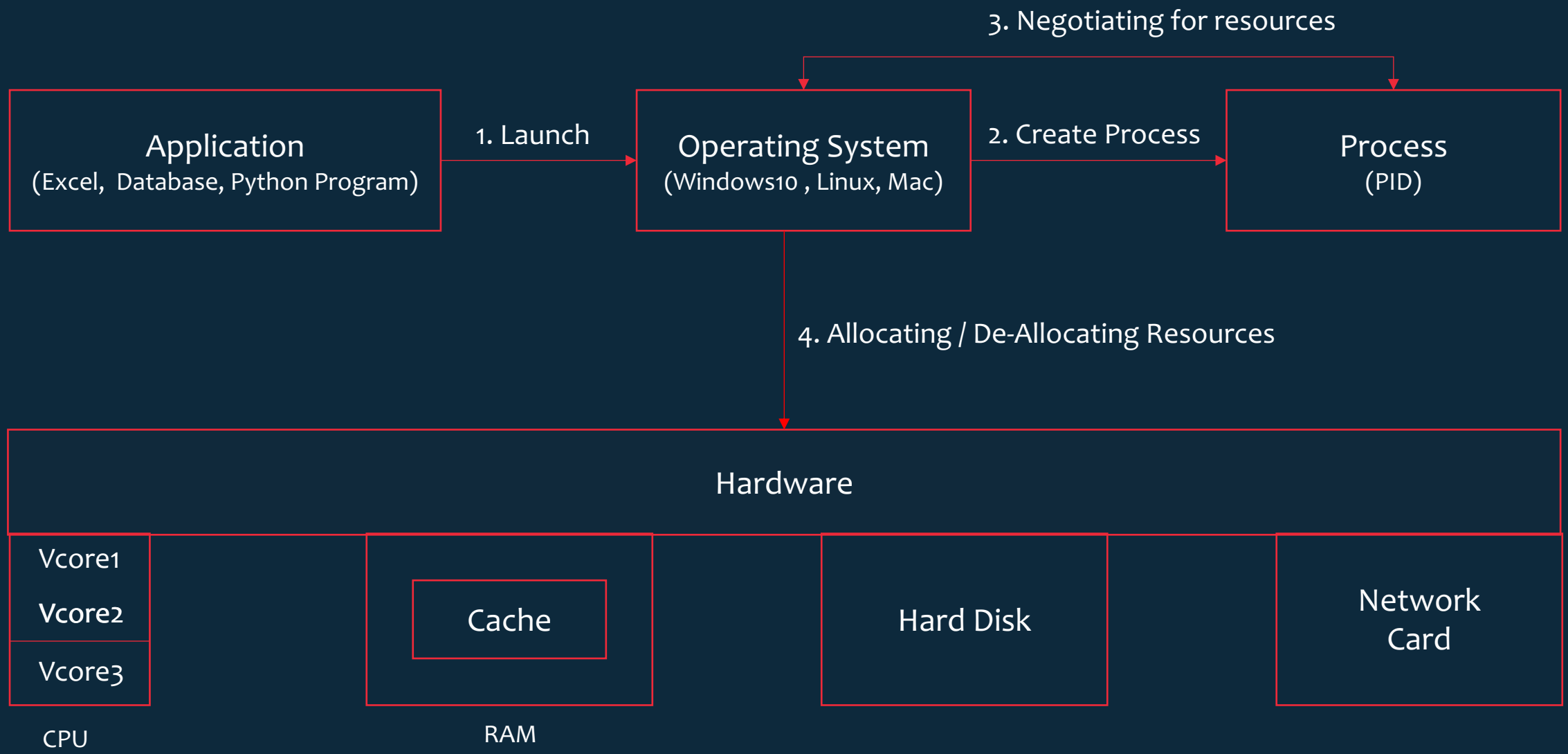
As code points less than or equal to 65,000 requires 16bits and more than 65000 needs two 16bits(surrogate pairs). So this encoding is considered as variable length encoding.

Understand Me(Computer/Server/Node)



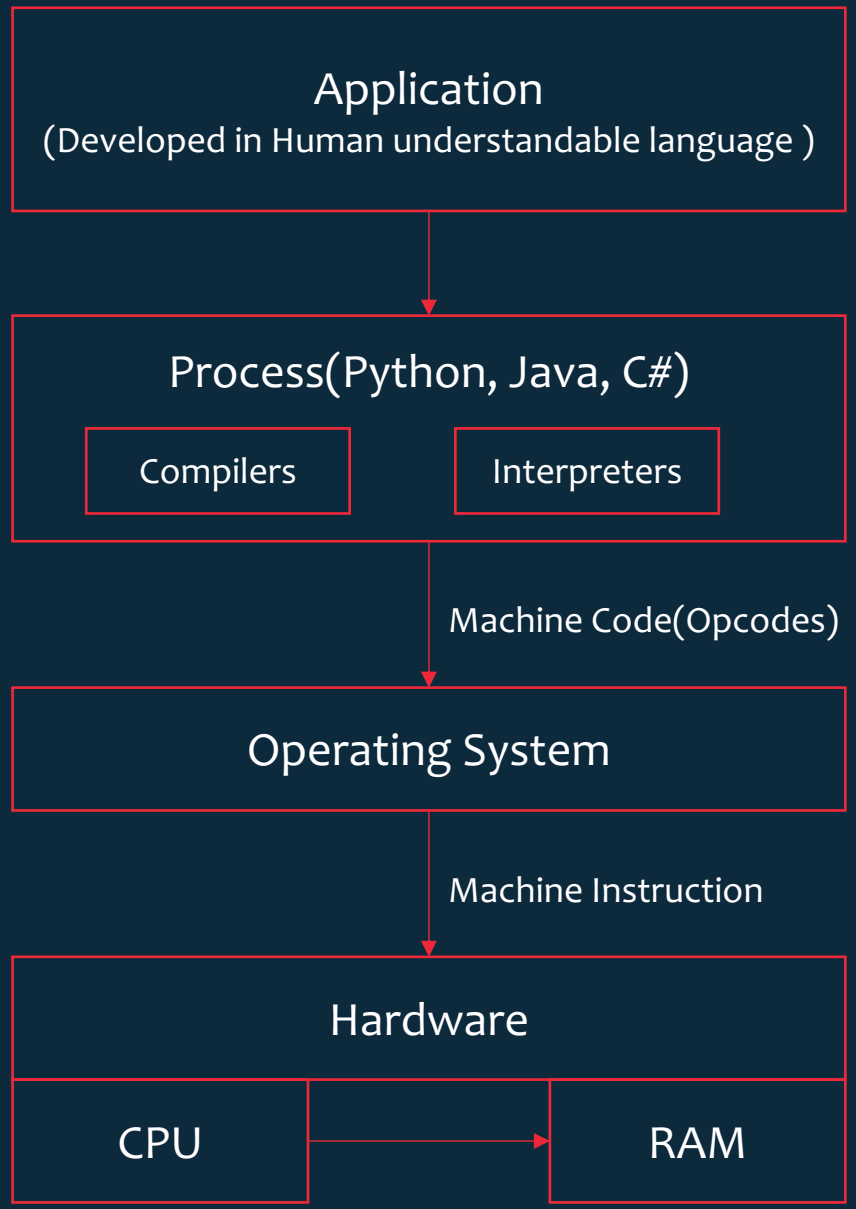
Task/Process

Unit of Work

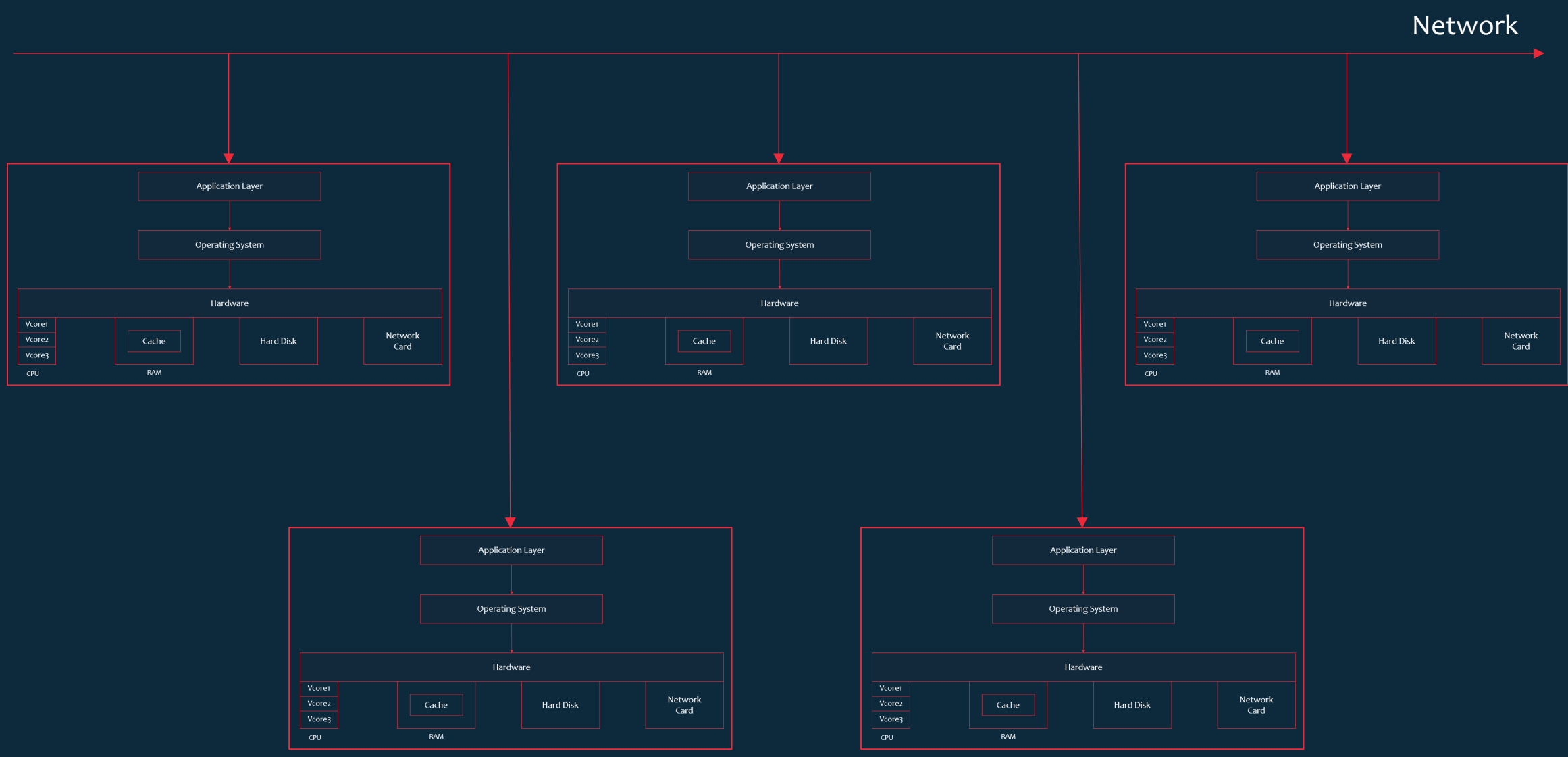


Does Computer Understand English?

No, Machine / Binary Code

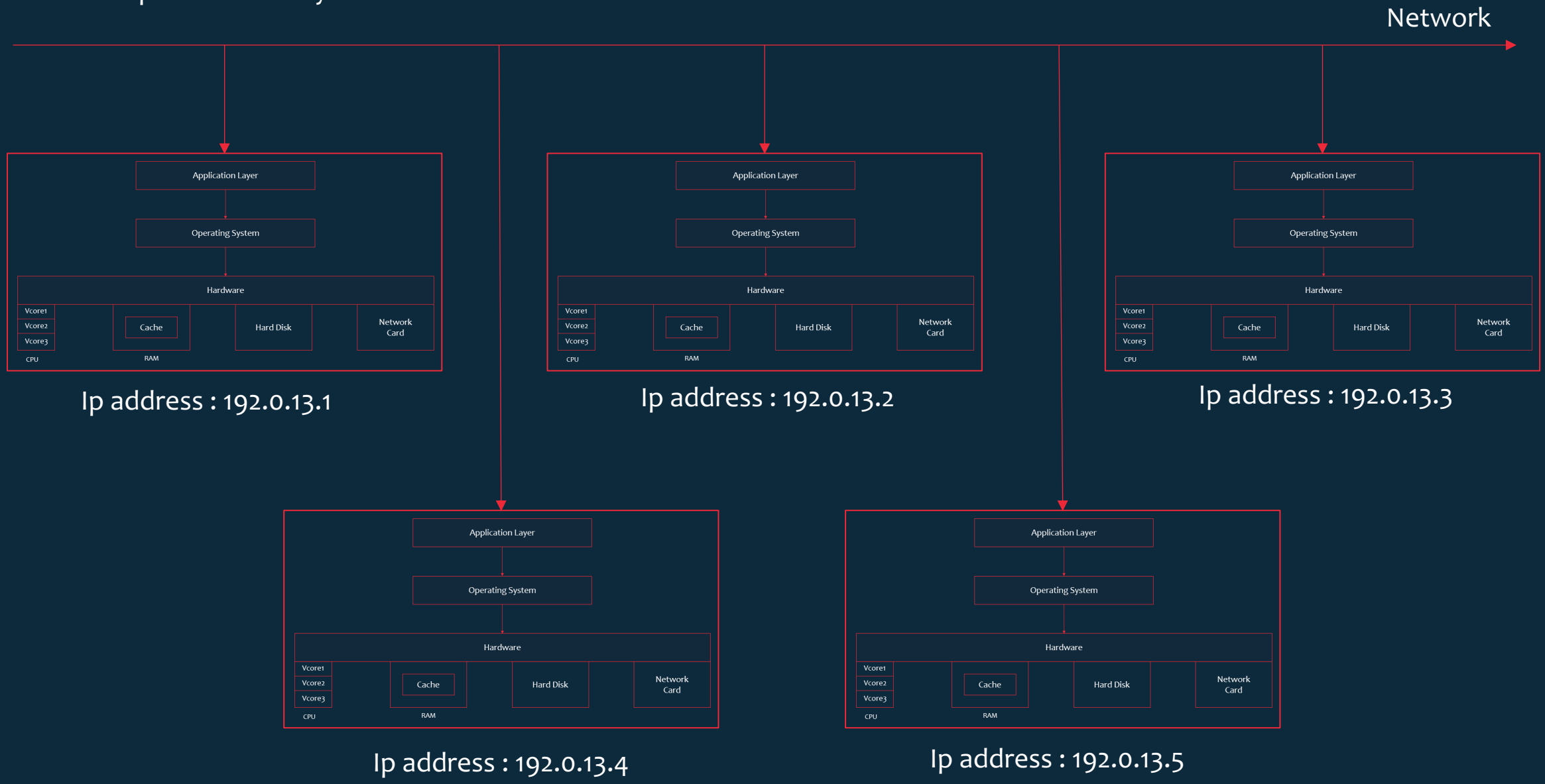


Cluster Computing



Cluster Computing

How will these computers indentify each other with in a network?

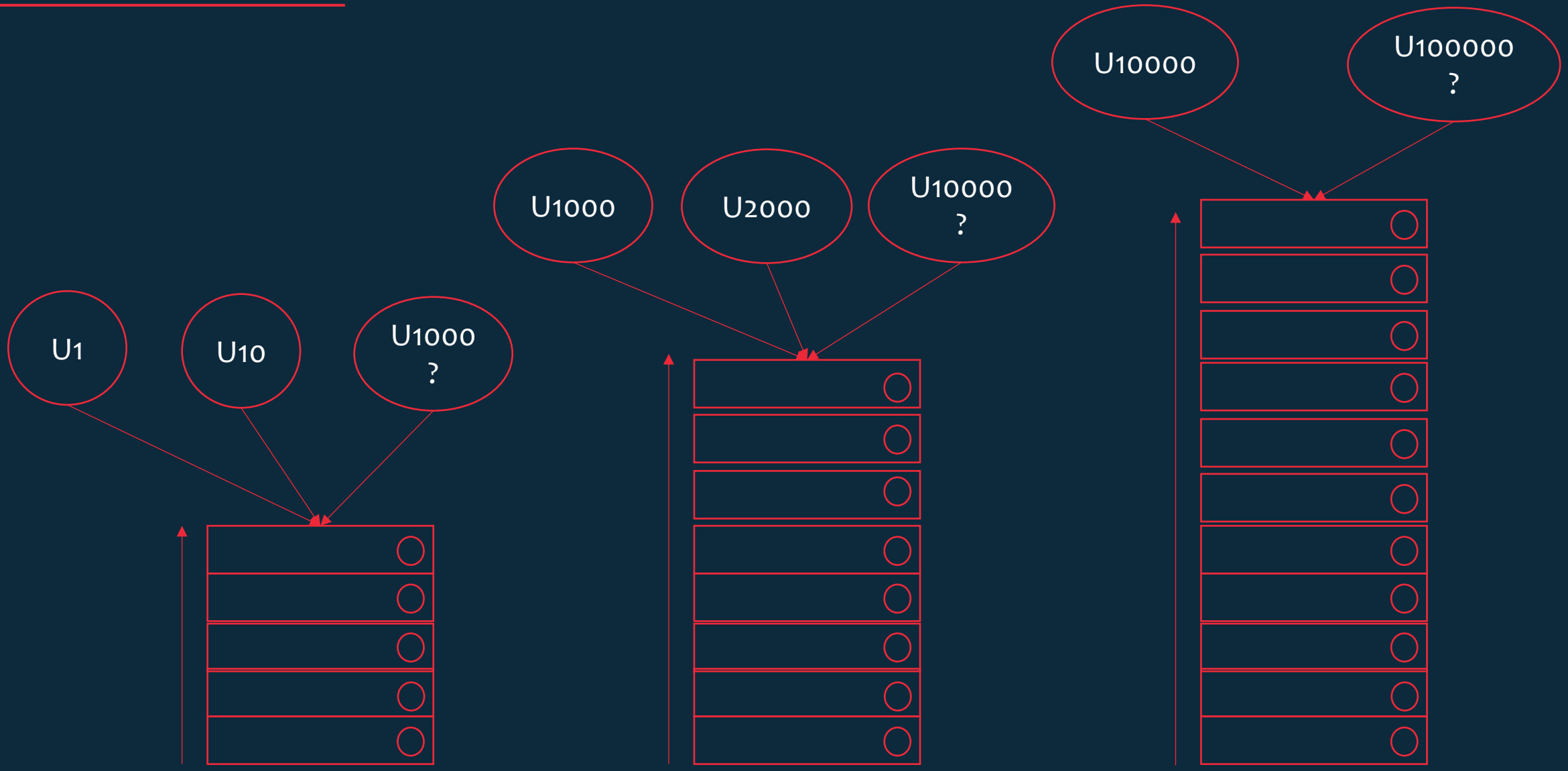


Vertical Scaling

Vertical scaling can essentially resize your server with no change to your code. It is the ability to increase the capacity of existing hardware or software by adding resources.

Imagine there is a server with web app deployed serving user requests. When load(more number of users requesting at a time) increases then vertically double the resources.

Vertical Scaling

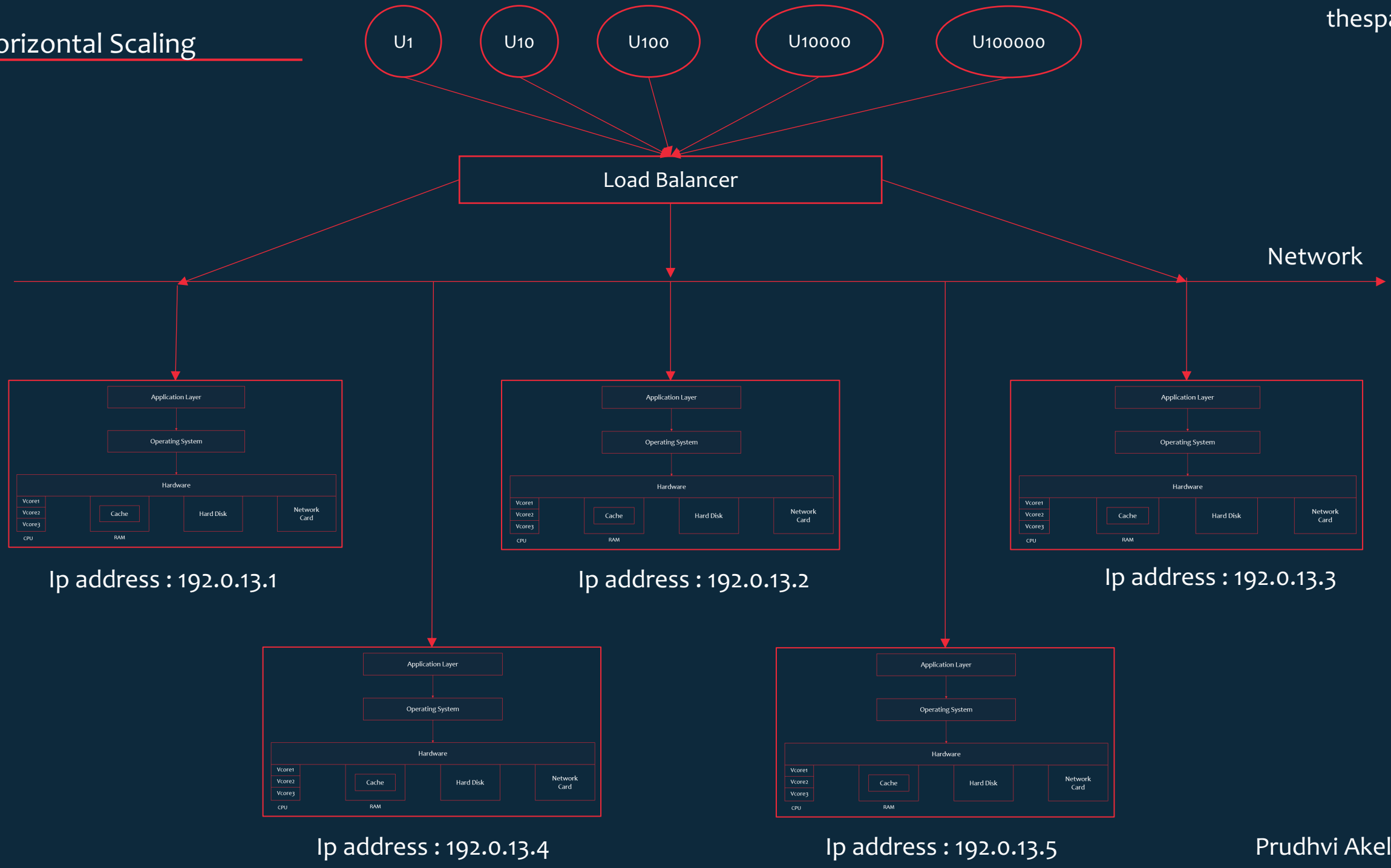


Horizontal Scaling

Horizontal scaling affords the ability to scale wider to deal with traffic. It is the ability to connect multiple hardware or software entities, such as servers, so that they work as a single logical unit.

Imagine there are group of servers deployed with an web app serving user requests. The user requests are horizontally distributed across the servers using load balancer. When load(more number of users requesting at a time) increases then new server will be commissioned and added to the group . When the load decreases the servers will be de-commissioned from the group.

Horizontal Scaling



By now we understand how Horizontal Scaling works. Now lets corelate the same with Big data

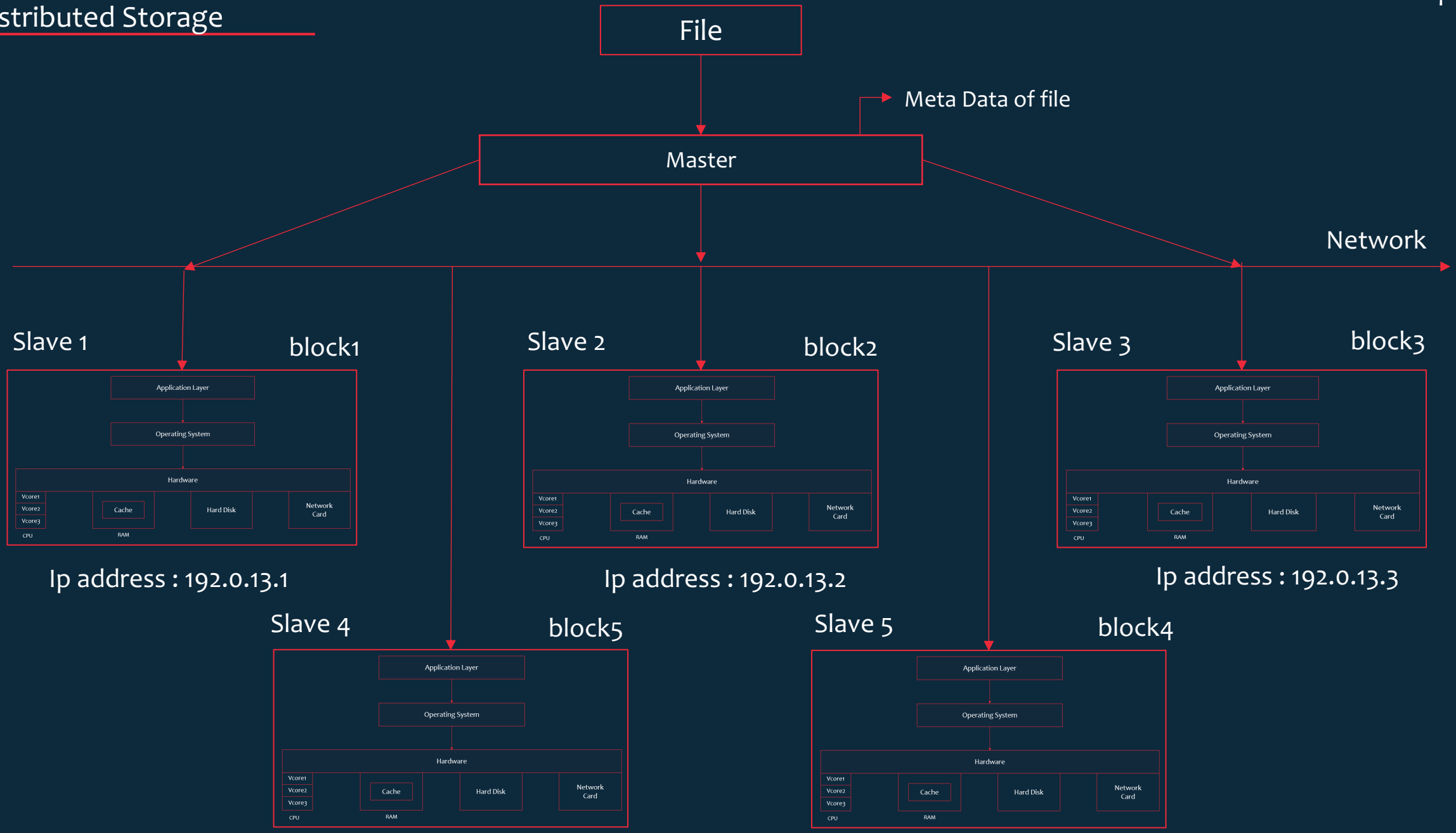
Distributed Storage

Split a file into multiple blocks/smaller files and distributed them across the servers.

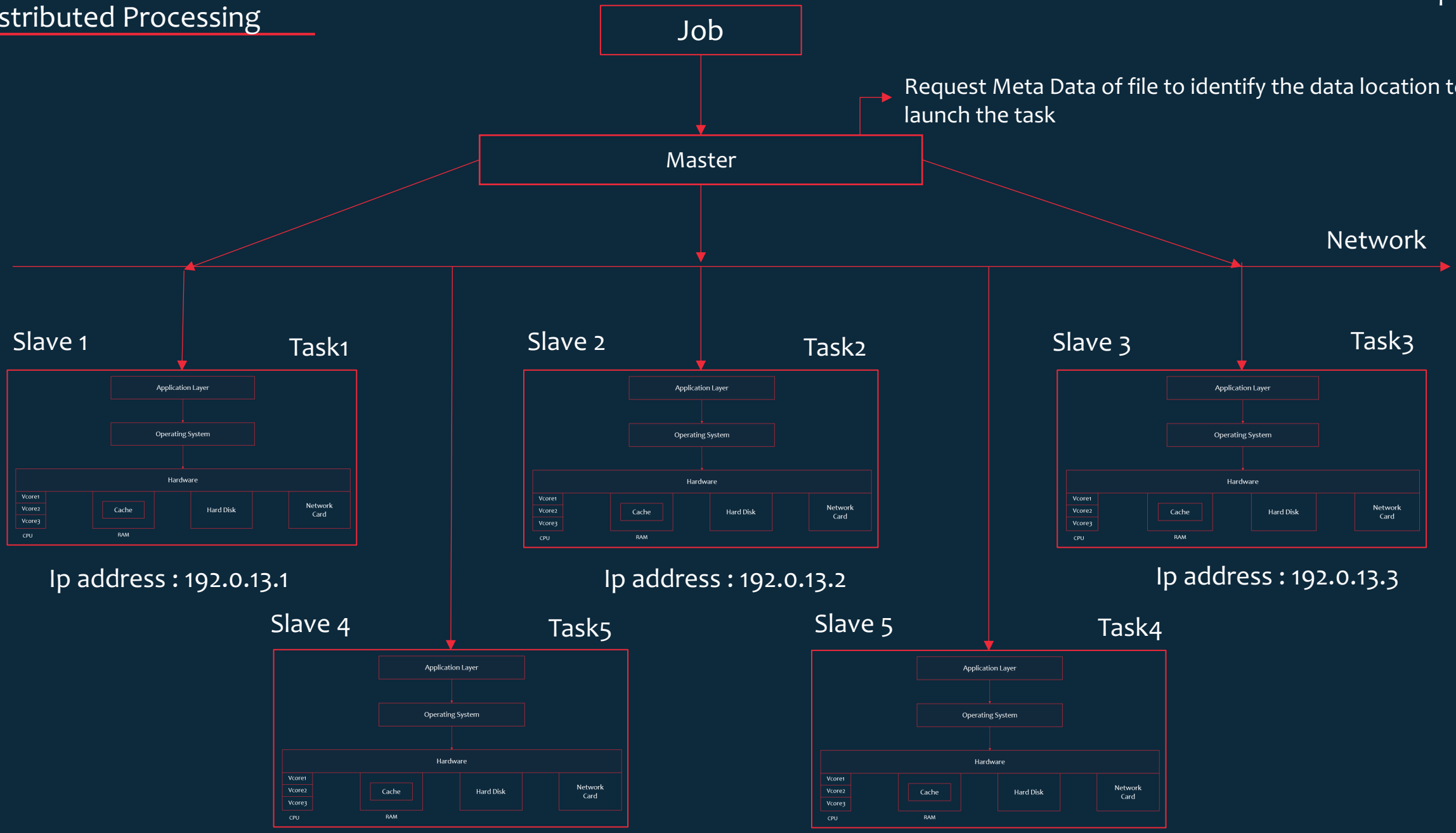
Distributed Processing

Identify the splited files and process them in parallel

Distributed Storage



Distributed Processing



Is processing data taking hours?

Why ?

Is Your Data Cached ?

Is Data Equally Distributed ?

Aren't Enough Resources Allocated ?

Are There Any Data Locality Issues

Are There Any Data Stragglers(Long Running Tasks)

??

Don’t Worry All Your Questions will be Answered

Python

- What is String?
- How to Print to Console
- Special Characters

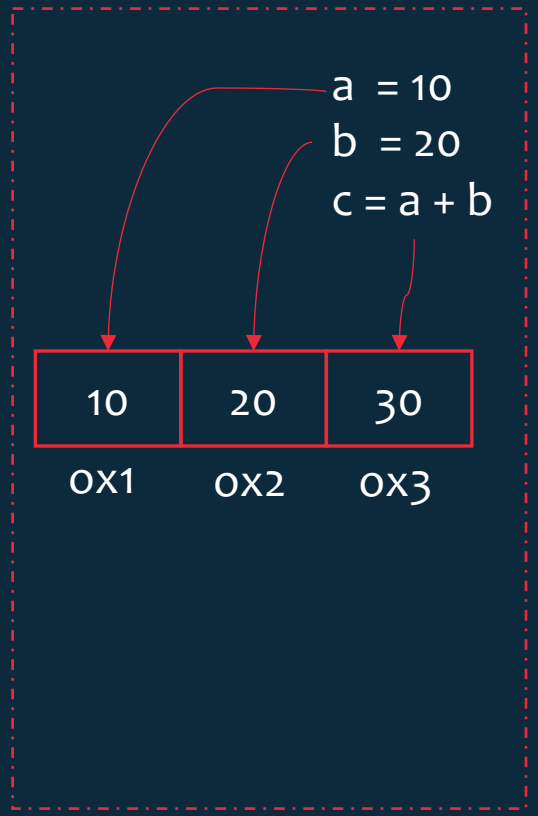
Module 1

- Defining variables in Python
  - Standard Assignment
  - Chained Assignment
  - Multiple Assignment
- Basic Datatypes in Python
  - Numbers
  - Strings
  - Booleans
- Building expressions from Variables
  - Number based expressions
  - Relational Expression
  - String based expressions
  - Built-in conversion functions

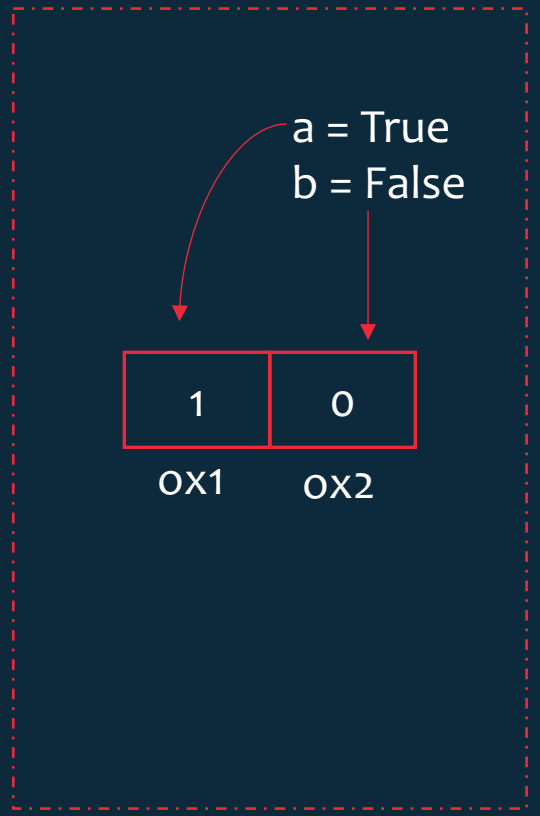
- Taking Inputs from console
  - input()
- Conditional Statements
  - If
  - If-else
  - If-elif-else
  - Single line if
  - Conditional Expression
- In-definite loops
  - while

Variables and Basic Data Types

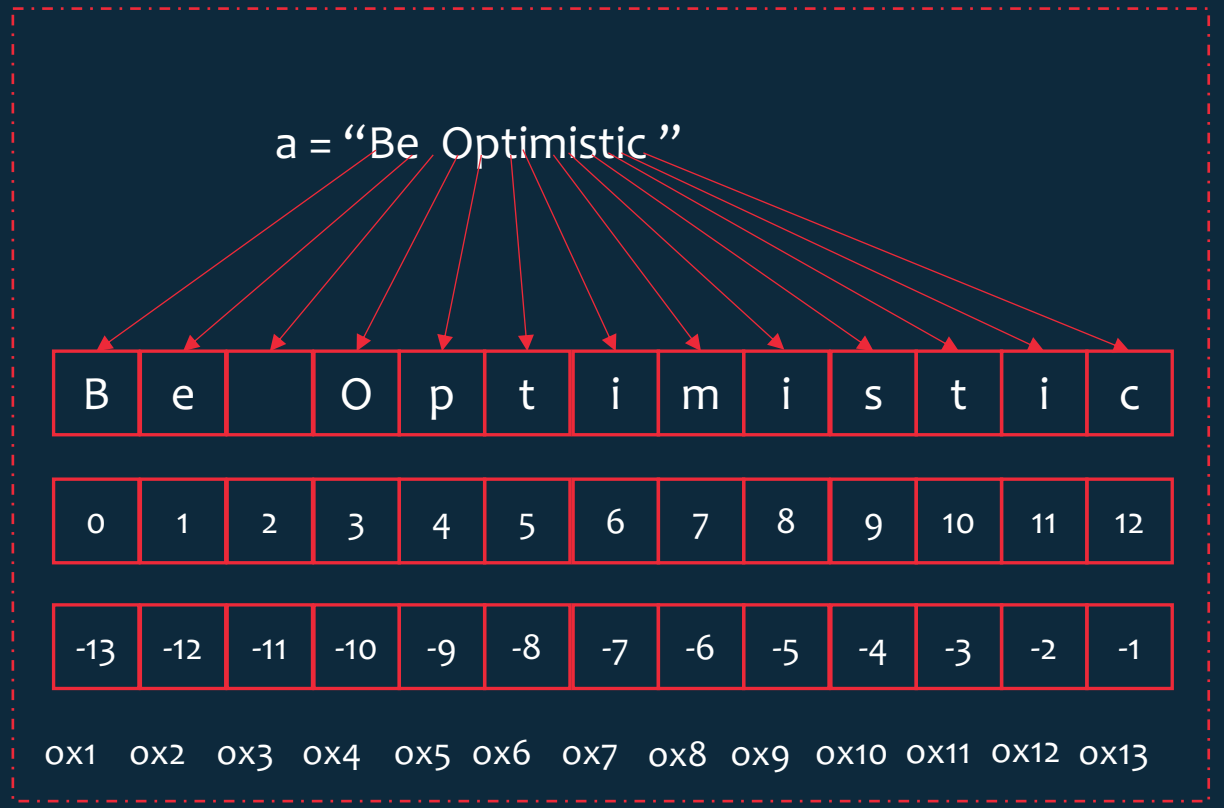
Numbers



Booleans



Strings



Note:

Everything in python is an object.

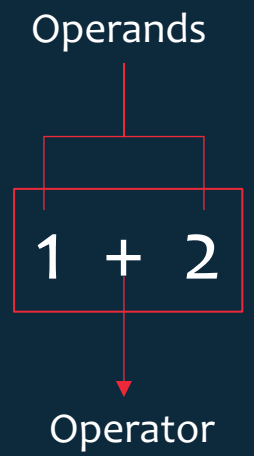
The above examples are just to give an idea.

Dynamic Programing Language

- Python is a dynamic/declarative programing language. Like any other programing language you don't need to declare data type at the time of defining variable.
- In the run-time by looking at the right hand side value of assignment operator it can identify data type of a variable.
- There is an in-built function called "type". User can use it to know the data type of variable.

Expressions

Expression is combination of numbers, variables and operators (+ - * /..). Expression will evaluates to a value.



Different Types of Expressions

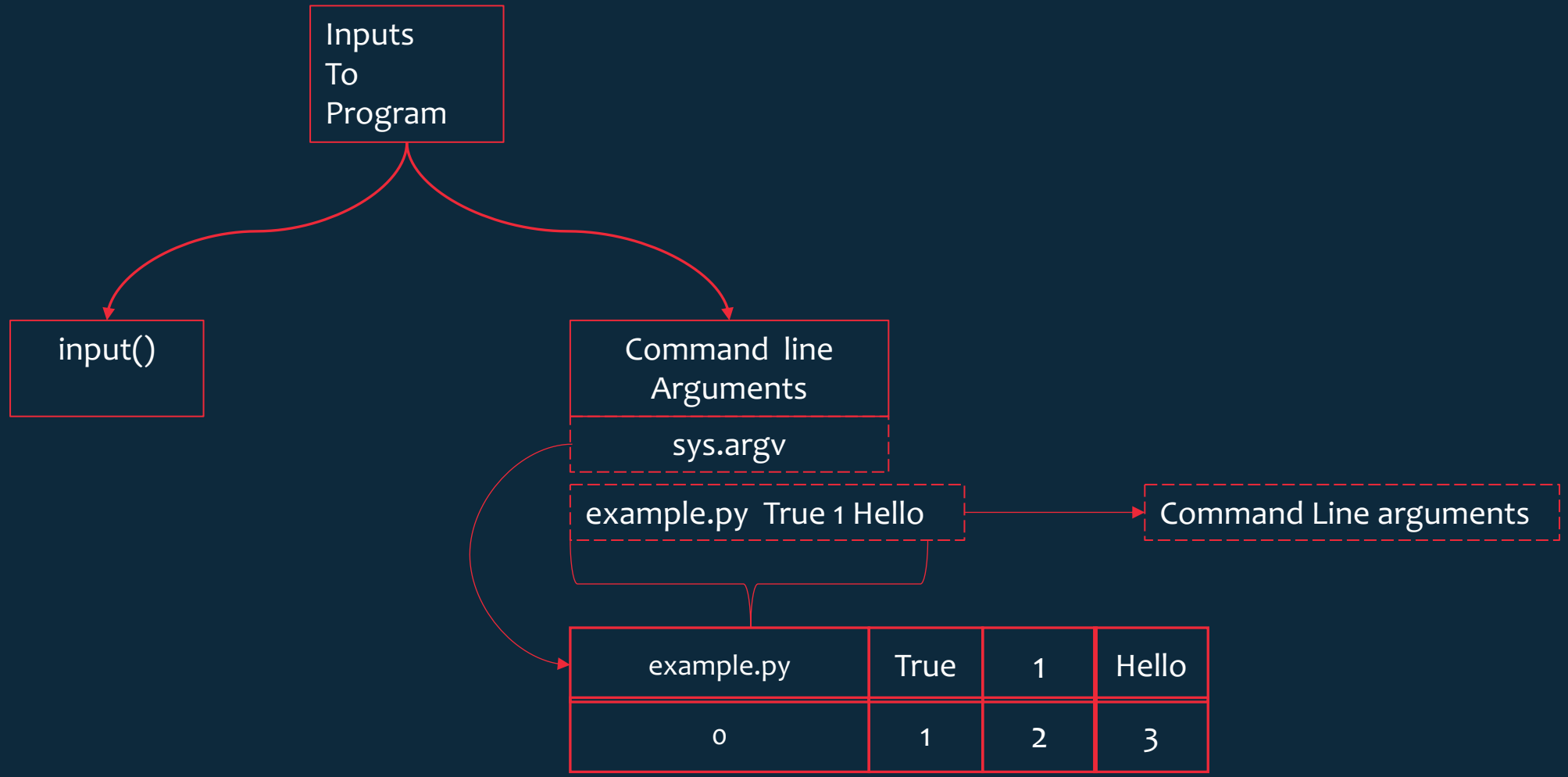
- Number based Expression : Return either integers or float as value
- Relation Expression : Returns Booleans(either True or False) as value
- String based Expression : Returns String as value

Floating point error in Python

Data Types Conversions

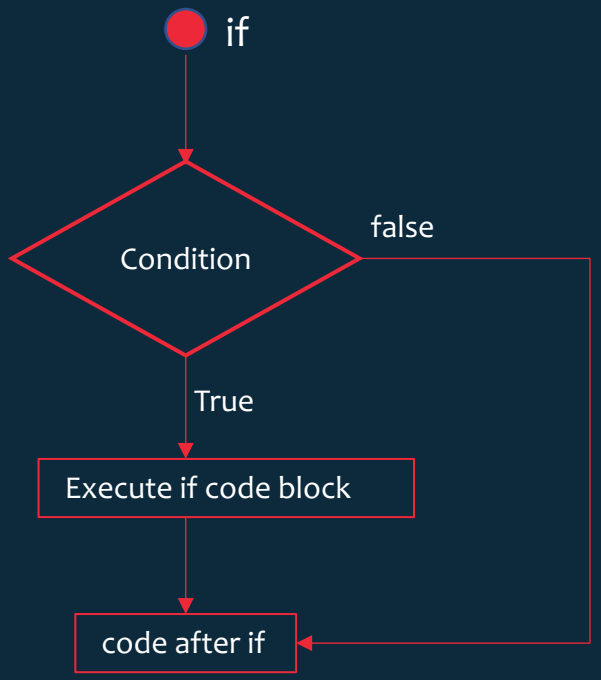
Name	
int()	Used to convert number string values to int
str()	Used to convert any value to string
float()	Used to convert floating point number string value to float
bool()	Used to convert any value to Boolean(True or False)

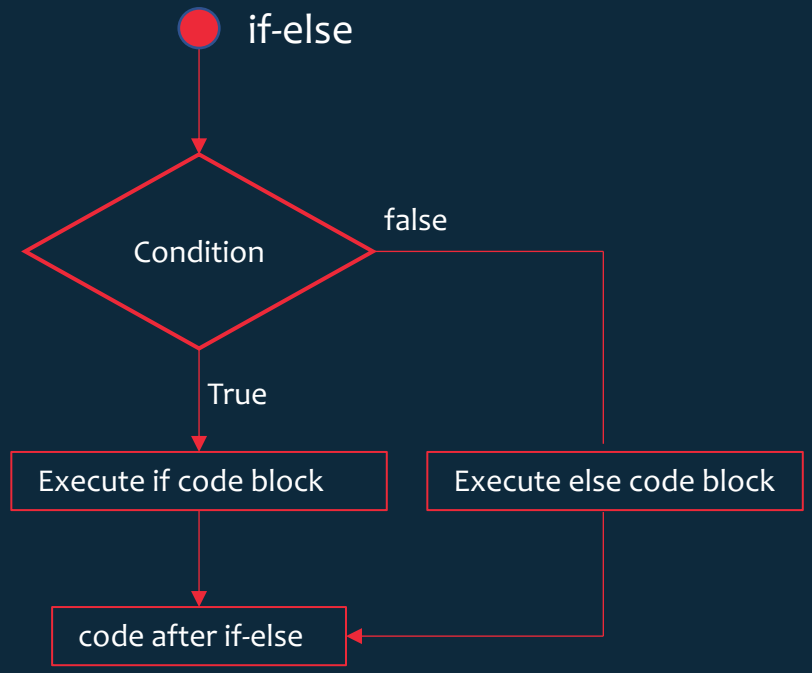
Inputs



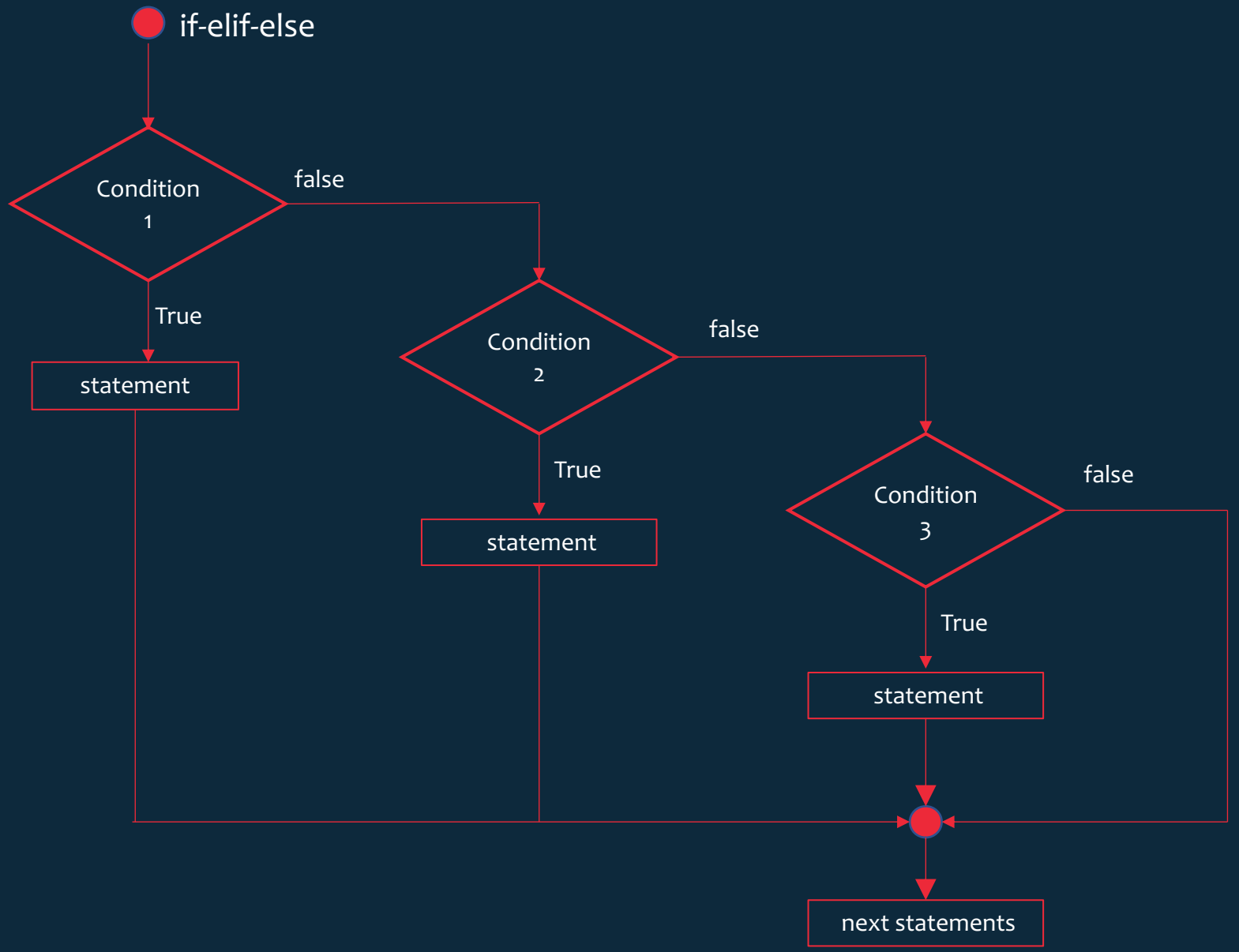
Note: the 0th argument will always be absolute path of python program path that you will be executing.

Conditional Statements





Conditional Statements



Conditional Expression

Syntax-1

Syntax-2

.
.
.

Quick Introduction to List and len function

List

→How many elements/items this list has?

- 5

→How to get the count programmatically using python?

- using len() function

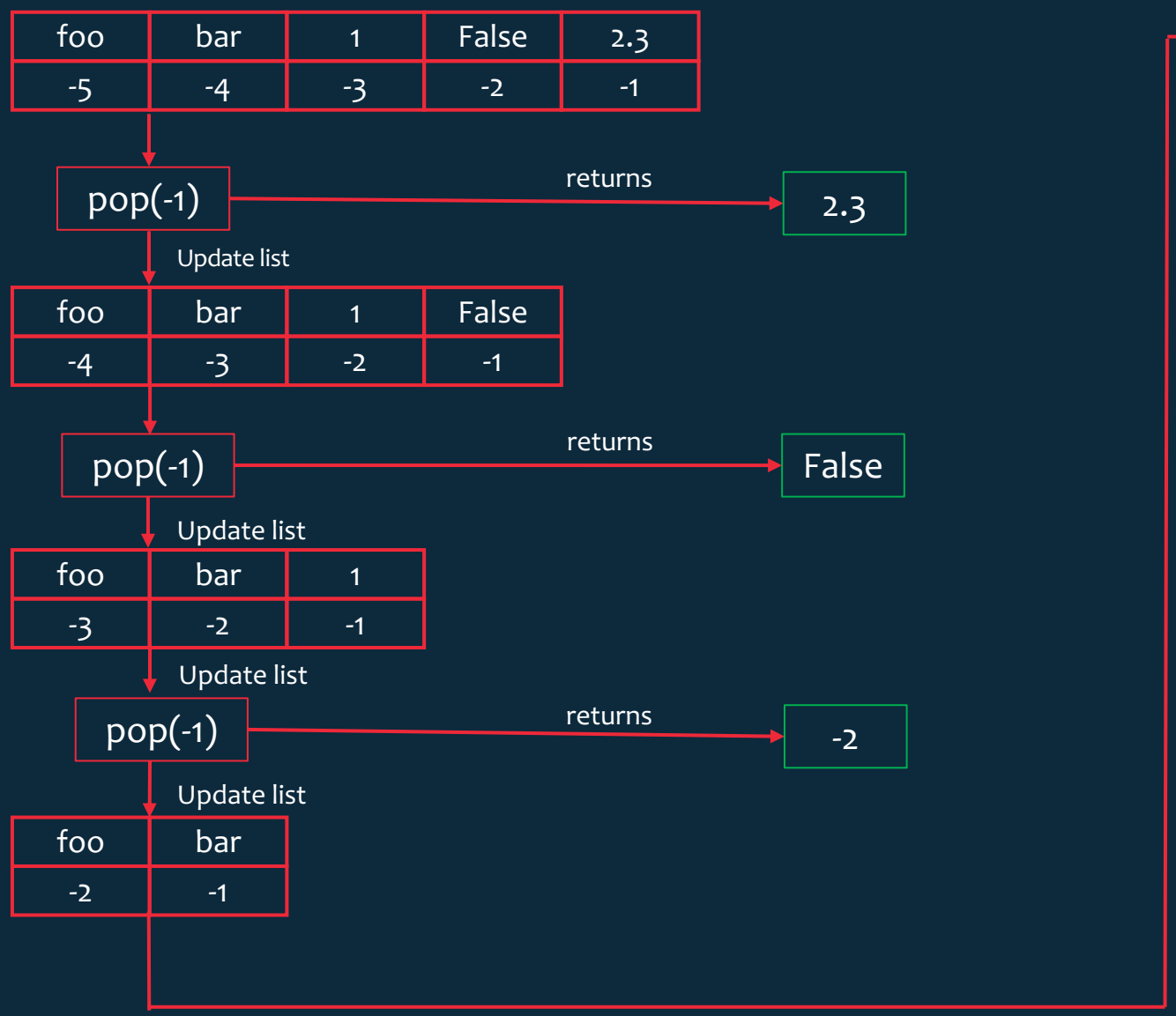
- Indexed and reverse -indexed
- Heterogenous (diverse in character or content)
- They work pretty similar to strings.

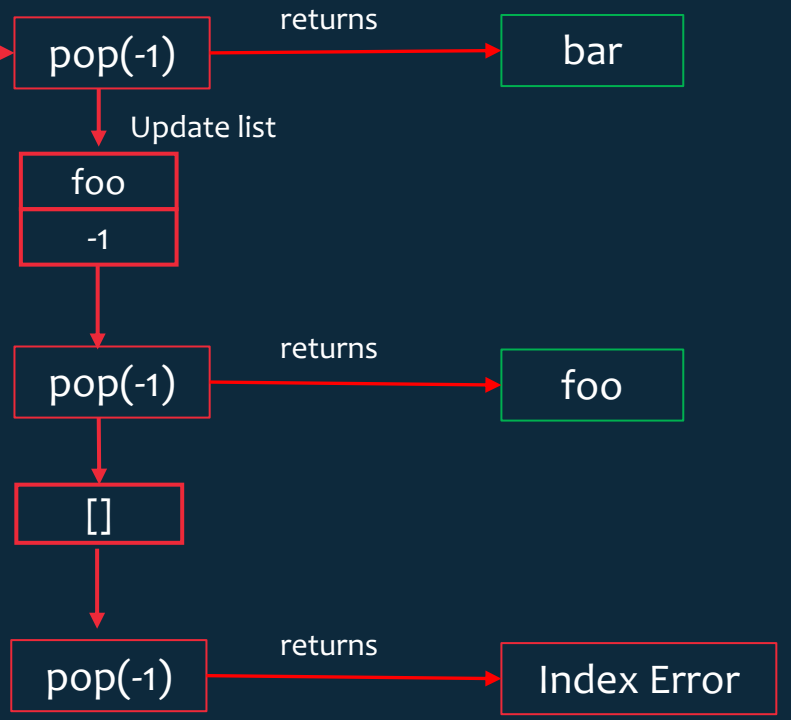
  Difference

  → List : Mutable

  → String : Immutable

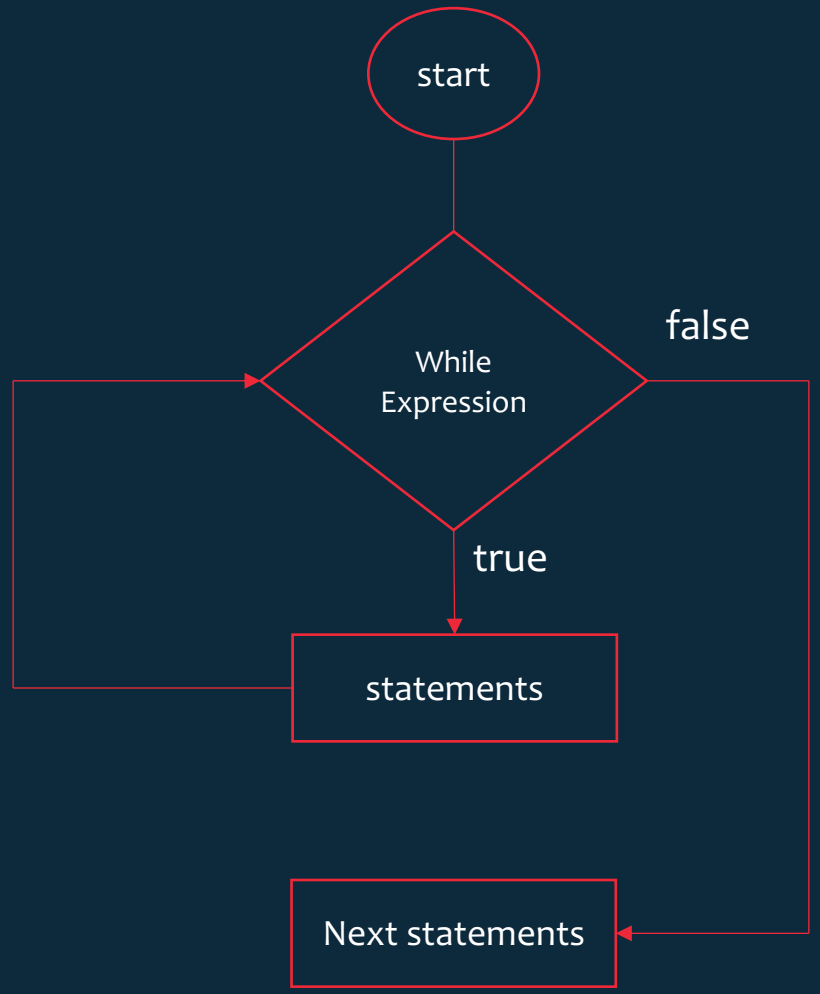
Understanding List's pop method

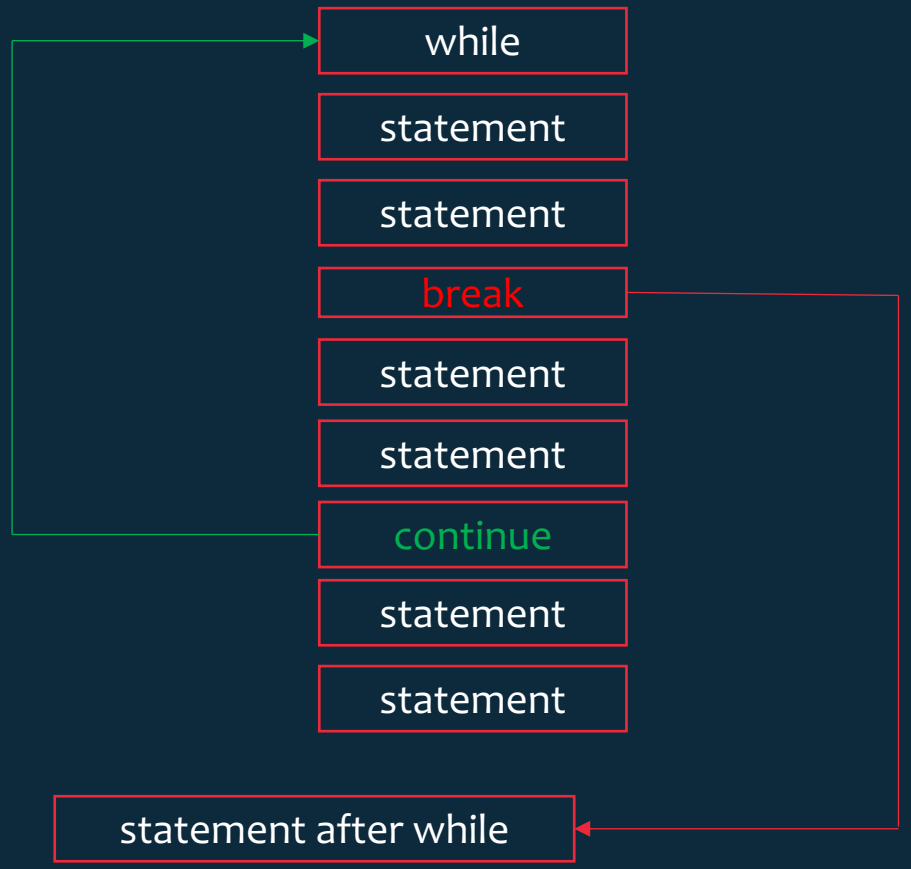




Note:
Everything in python is an object.
When object is created in the python it return the reference of the memory location where the object is created in heap. Using that we can able to access the methods that belongs to the object

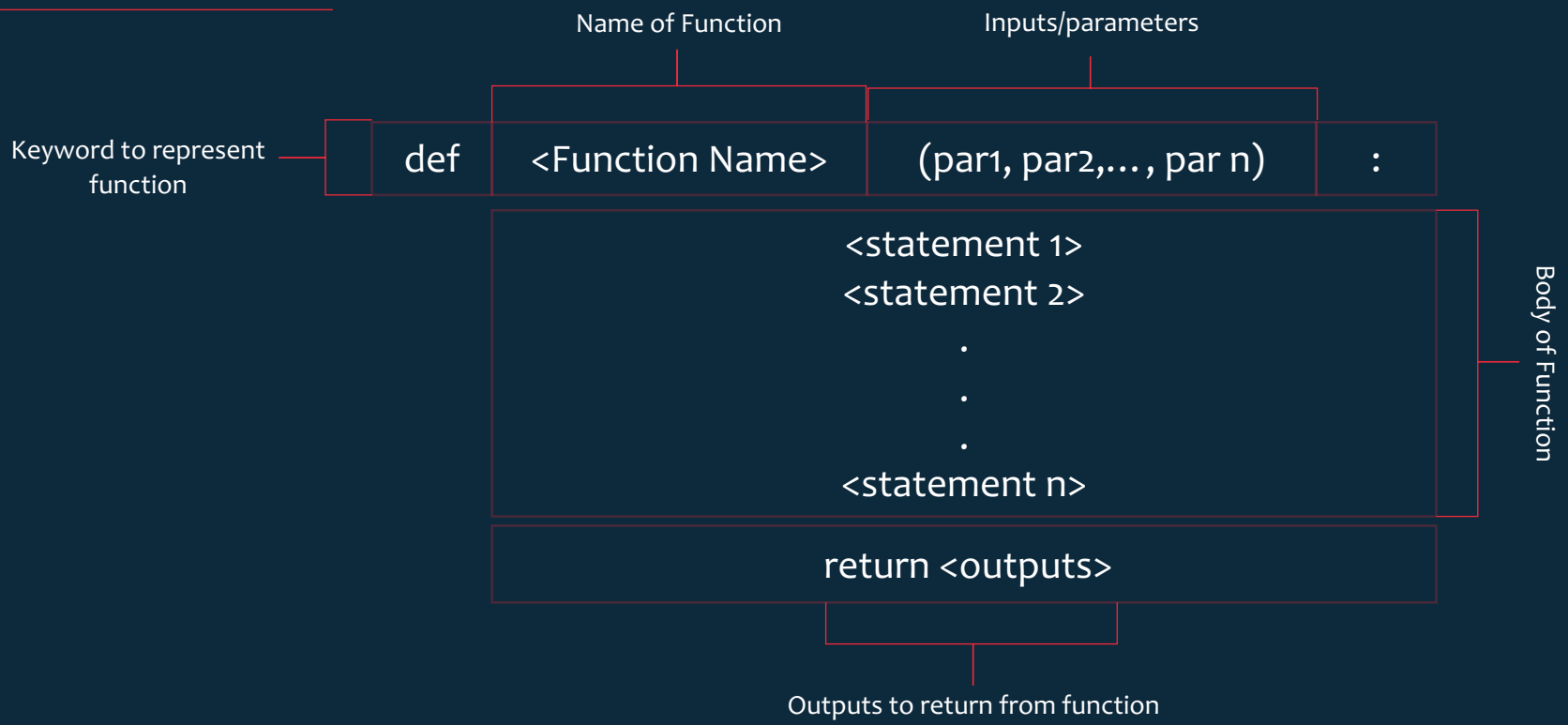
Indefinite Loop (while)



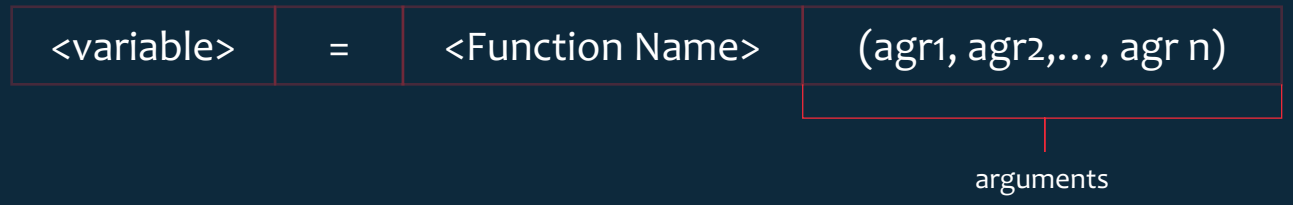


Introduction to Functions

Function Definition :



Function Call :



Types of Functions

- Built-in
- UDF's(User Defined Functions)
- Anonymous/ Lambda

Important points to Remember

What is the max number of arguments in a python function
→ Till 3.7 we can able to pass 256 arguments. However from 3.7 there is not limit.

What will a function return out of the box if return statement is not defined in a function?
→ None

Is function is a also a object in python?
→ Yes. Everything you define in python is an object

Is it possible to return multiple output from function?
→ Yes. As a tuple.

Default argument : Default arguments are those that take a default value if no argument value is passed during the function call.

Keyword argument : If you want to make sure that you call all the parameters in the right order, you can use the keyword arguments in your function cal.

Variable number of arguments : In cases where you don't know the exact number of arguments that you want to pass to a function, you can use the following syntax with *args.

Is it possible to pass both keyword arguments and variable number of arguments together ?
Lets check.

Recursive Functions

Traditional Recursion

X = factorial(3)

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n – 1)
```

Stack frame 1

3 * 2 = 6
is returned

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n – 1)
```

Stack frame 2

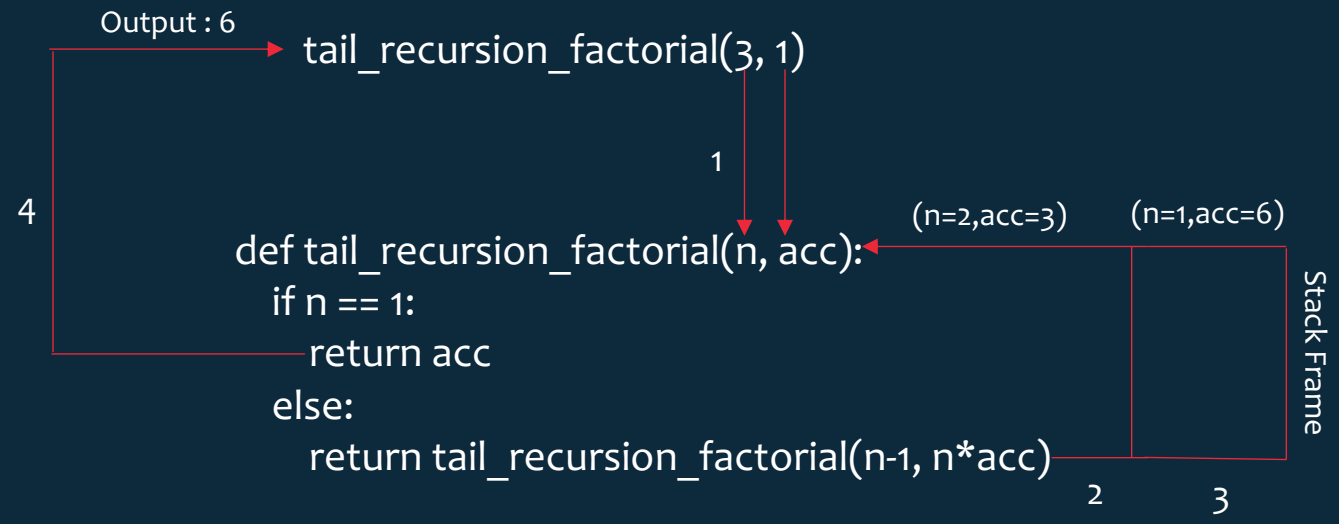
2 * 1 = 2
is returned

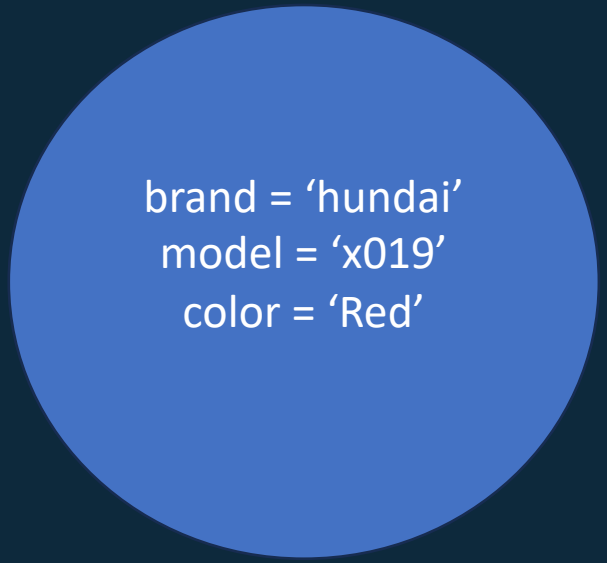
```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n – 1)
```

Stack frame 3

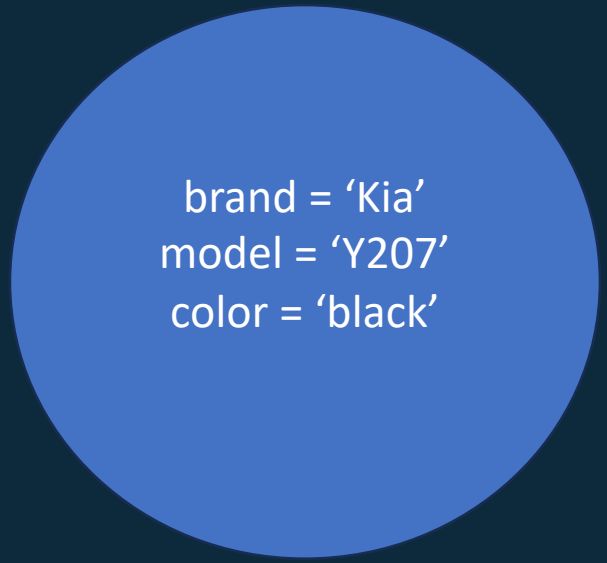
1
is returned

Tail Recursion




brand = ‘hundai’
model = ‘x019’
color = ‘Red’

mx101


brand = ‘Kia’
model = ‘Y207’
color = ‘black’

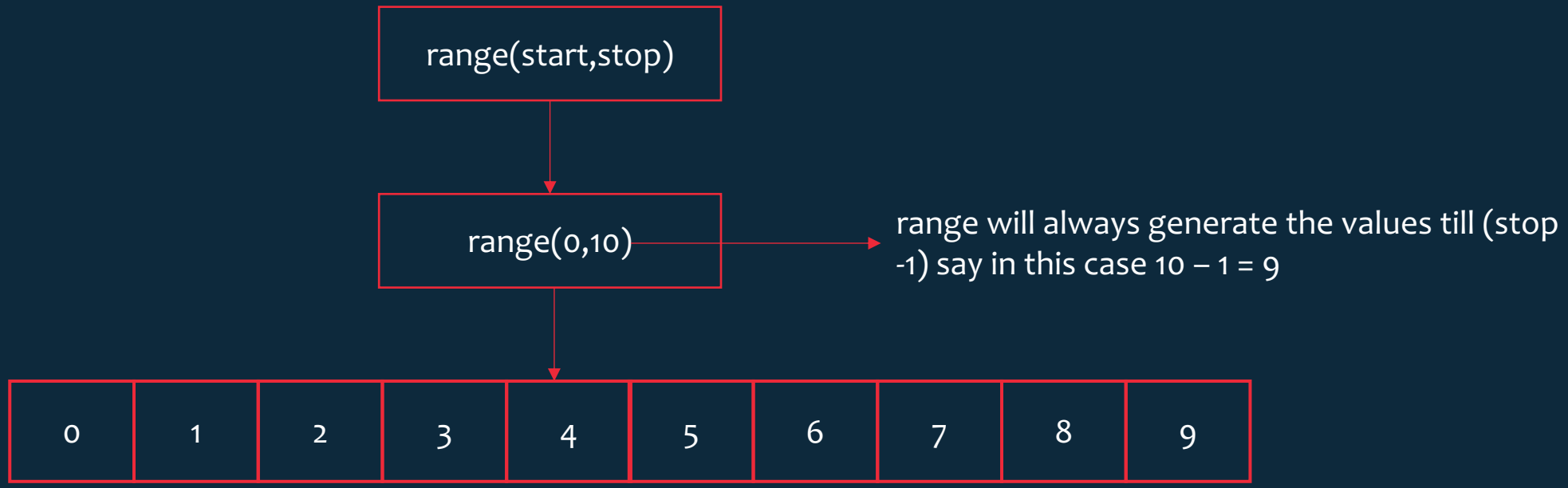
mx102

Important points to Remember

Is it mandatory to define main function in python?
→No.

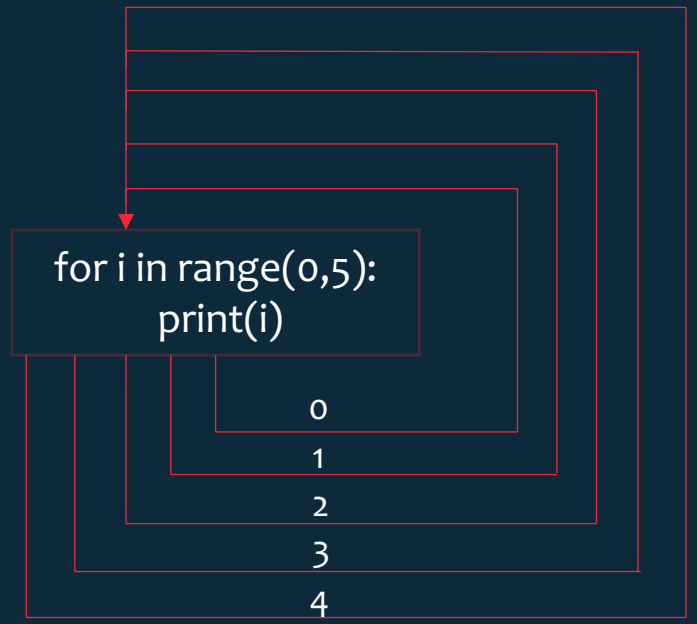
Is it a good practice to make a variable global inside a function?
→Its not at all a good practice.

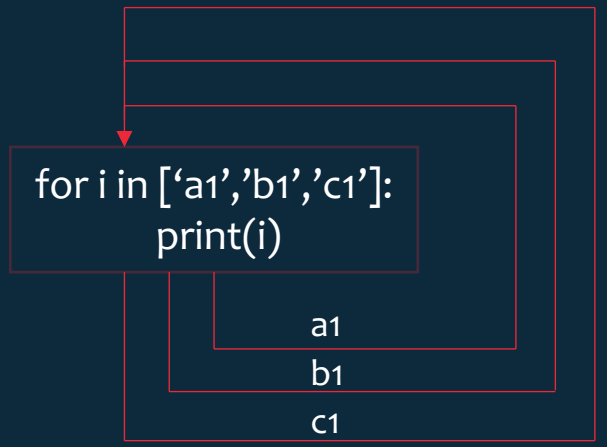
range function



For : definite loop

for	<tmp_variable>	in	<iterable>	:
# implementation				





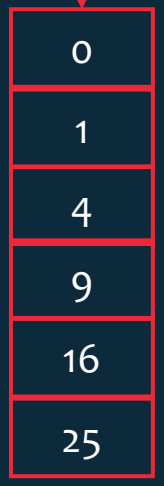
How for loop identifies whether next element exits or not?

Lets wait until OOP's for this answer

For : List comprehension

[	<output for each iteration>	for	<tmp_variable>	in	<iterable>	<condition> optional	]

```
[i * i for i in range(6)]
```



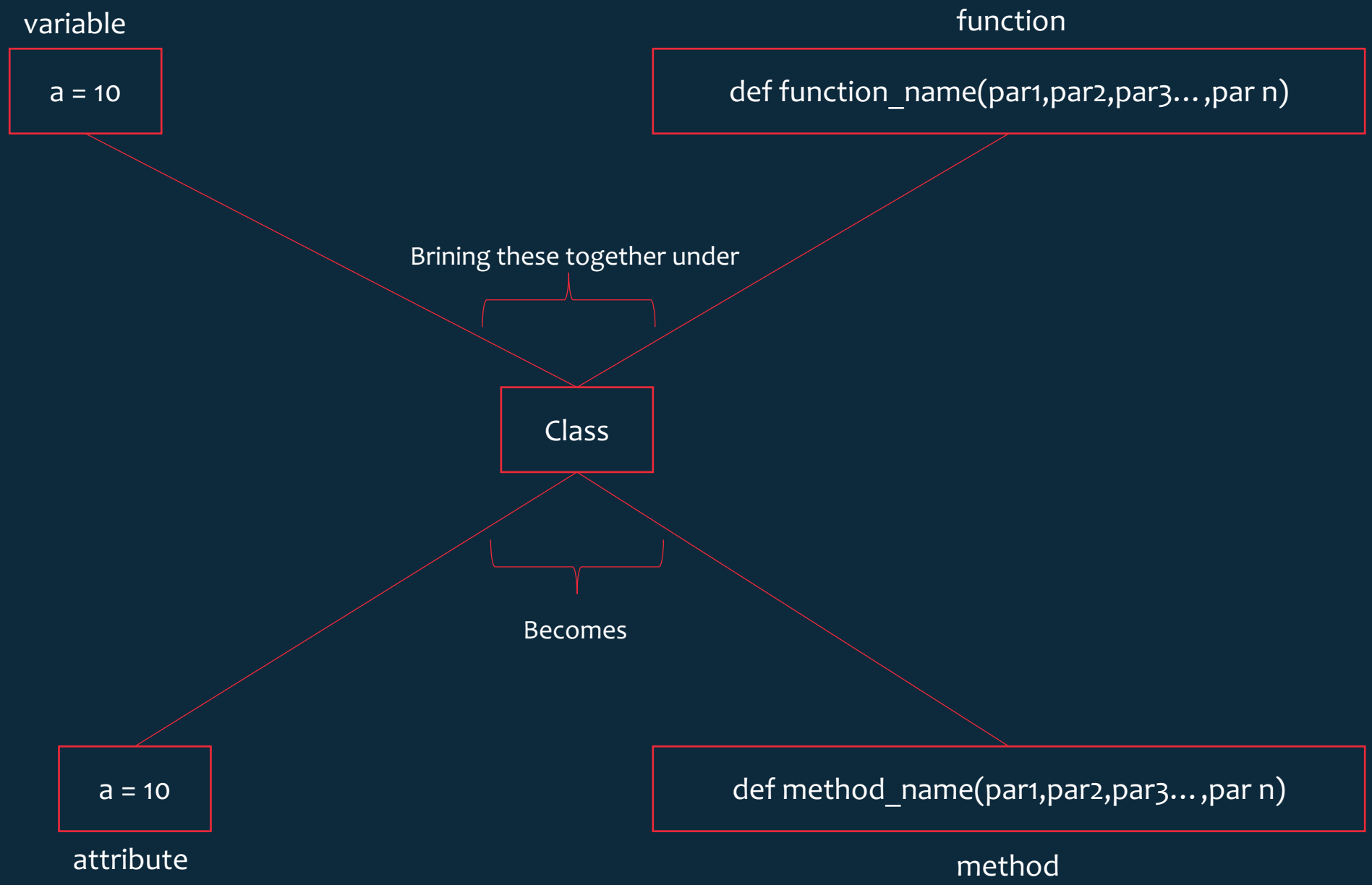

```
sentence = 'the rocket came back from mars'
[i for i in sentence if i in 'aeiou']
```

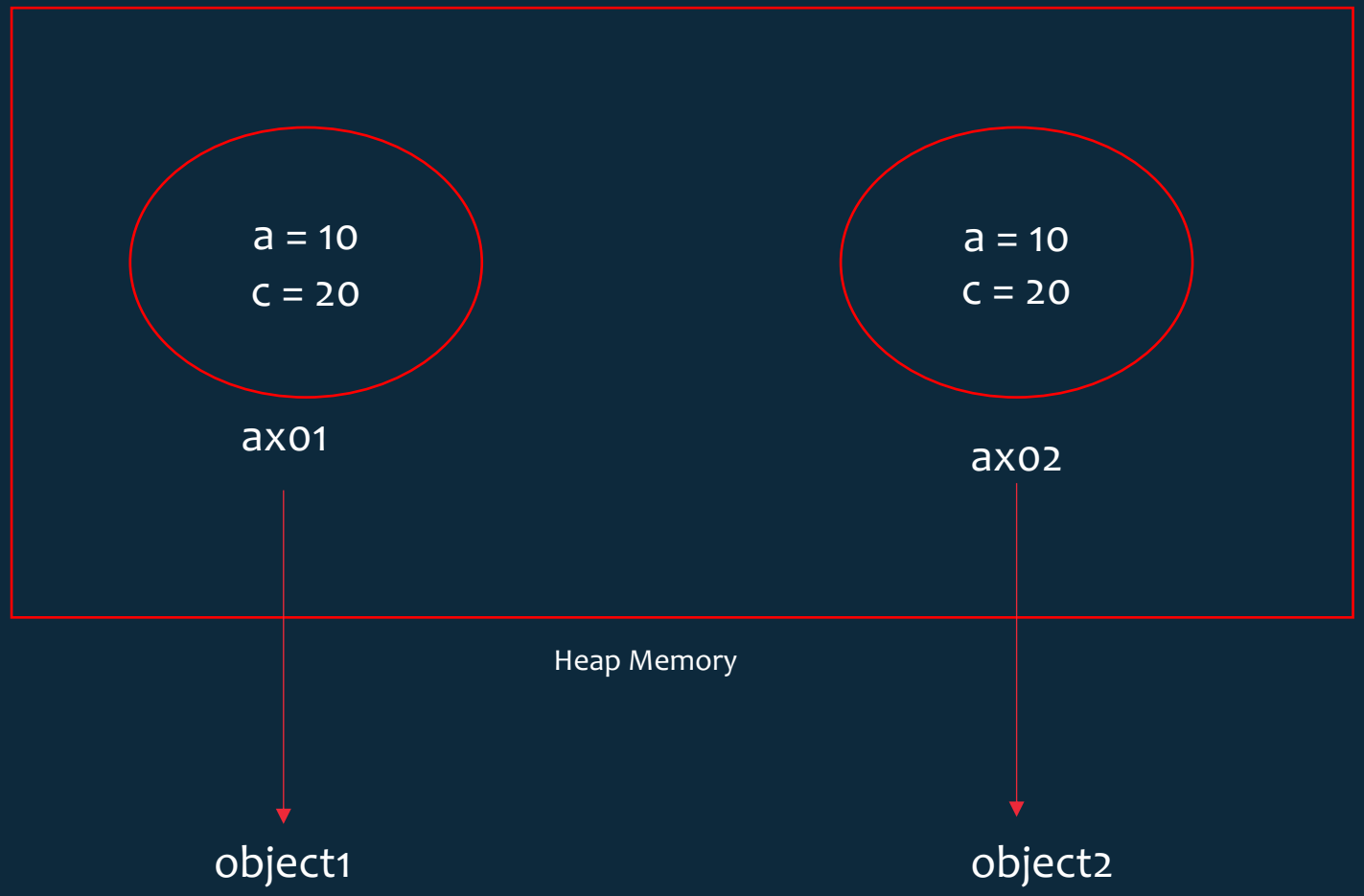
Check and return output only if condition satisfies

['e', 'o', 'e', 'a', 'e', 'a', 'o', 'a']

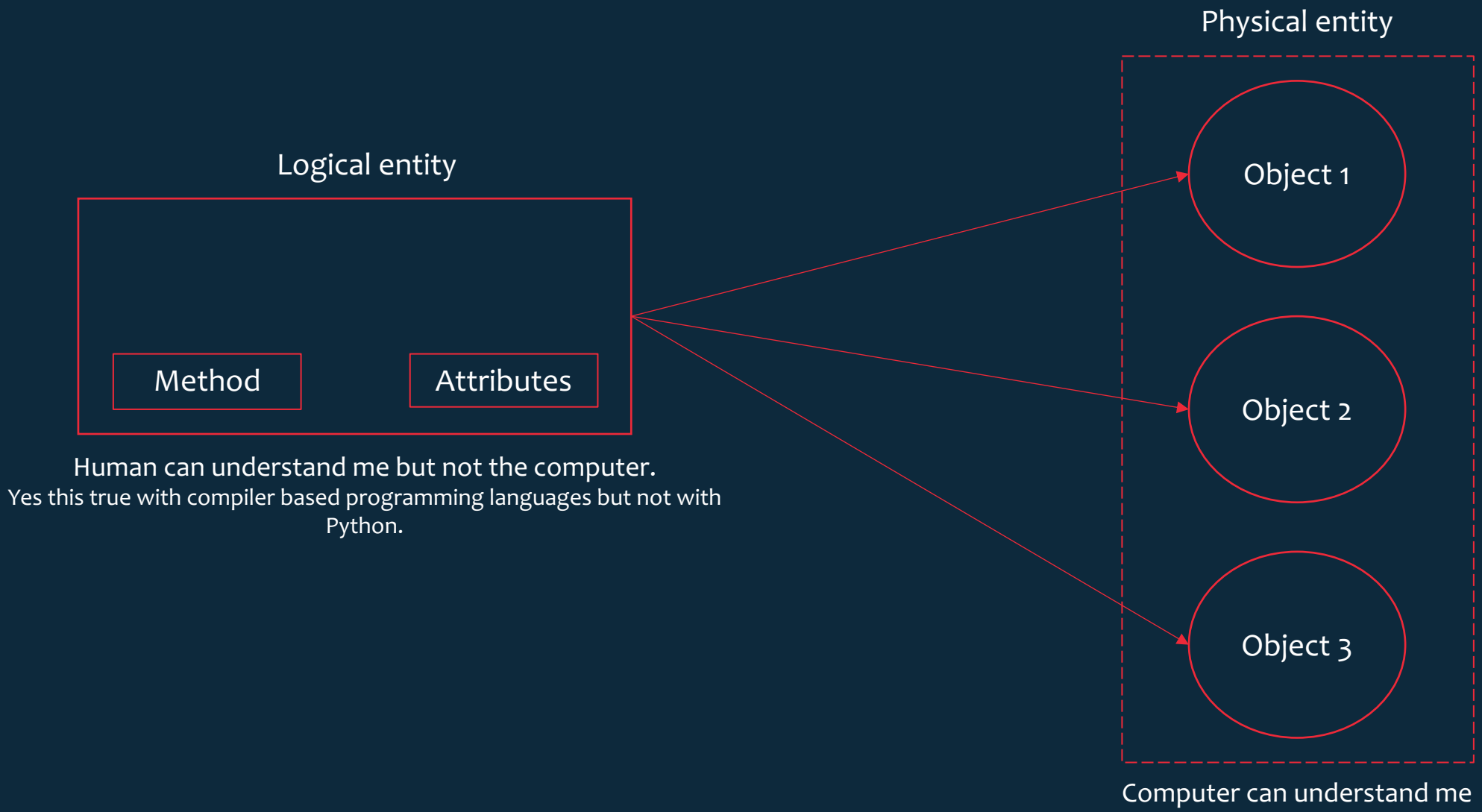
Welcome To OOP's

(Everything is an ☺bject in Python)

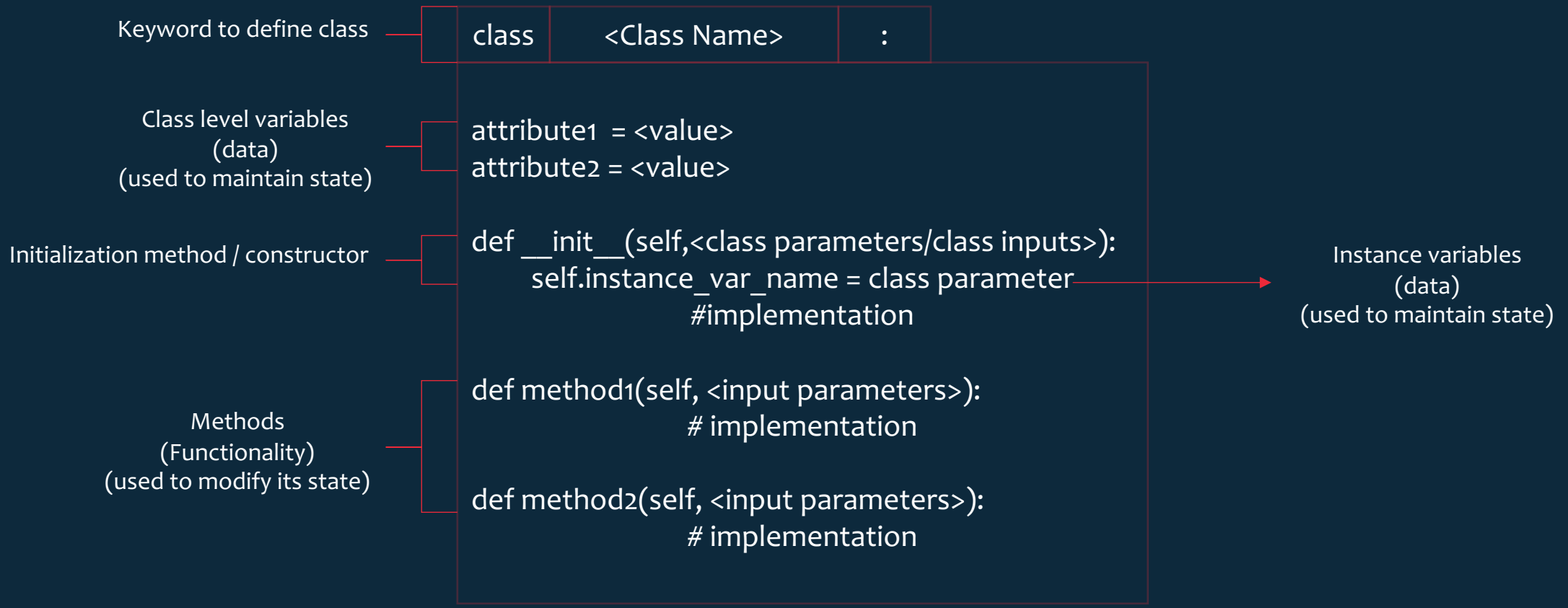

variable
a = 10
function
def function_name(par1,par2,par3… ,par n)
Brining these together under
Class
Becomes
a = 10
attribute
def method_name(par1,par2,par3… ,par n)
method


a = 10
c = 20
a = 10
c = 20
ax01
ax02
Heap Memory
object1
object2

2 Important Entities

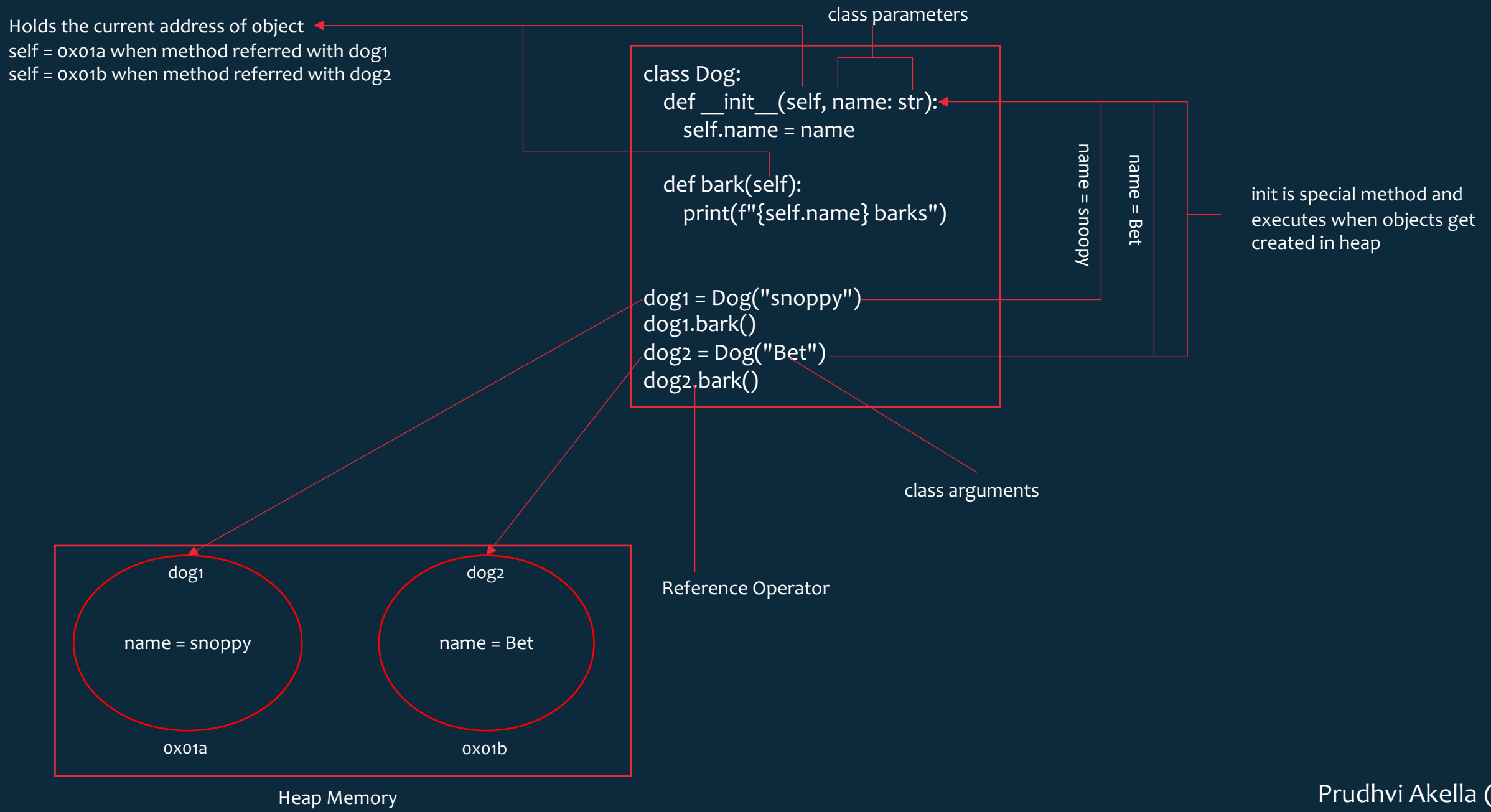


Class (Bundling Data + Functionality)




Department = ‘cse’
empid =1
firstname =Prudhvi
lastname = Akella
age = 30
ax01
empid = 2
firstname = Ravi
lastname = M
age = 30
ax02
empid = 3
firstname = Karthik
lastname = M
age = 40
ax03
Heap Memory
employee1
employee2
employee3

Class and Object Example



Linking two entities/objects

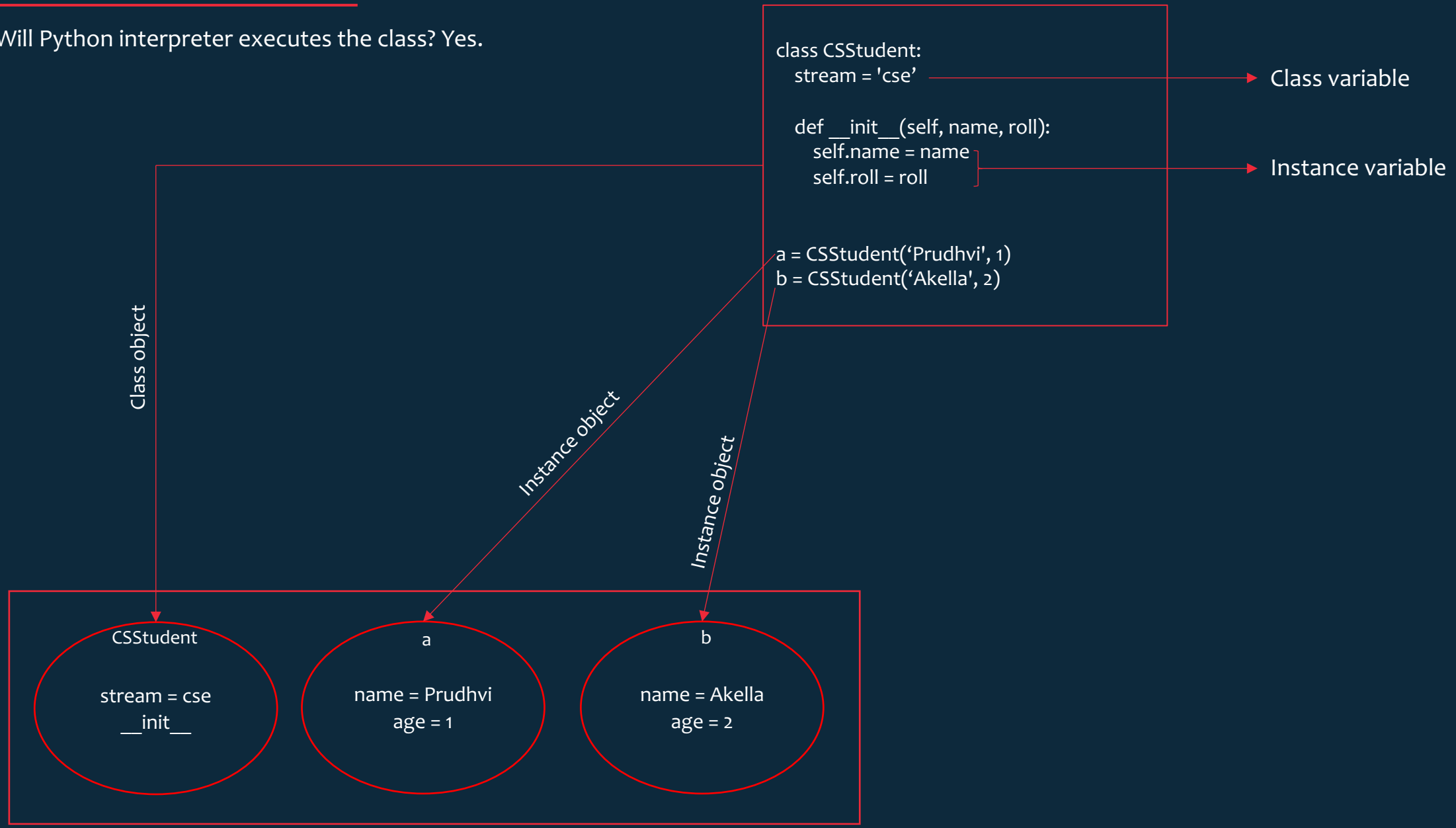
```
class Dog:

    def __init__(self, name, age):
        self.name = name
        self.age = age

def setBuddy(self, buddy_ref):
        self.buddy = buddy_ref
        buddy_ref.buddy = self

snoopy = Dog(“Snoopy", 2)
filou = Dog("Filou", 8)

snoopy.setBuddy(filou)

print(snoopy.buddy.name)
#(0x01a.0x01b.name)
print(snoopy.buddy.age)
#(0x01a.0x01b.age)


print(filou.buddy.name)
print(filou.buddy.age)
```

self

0x01a. setBuddy(0x01b)

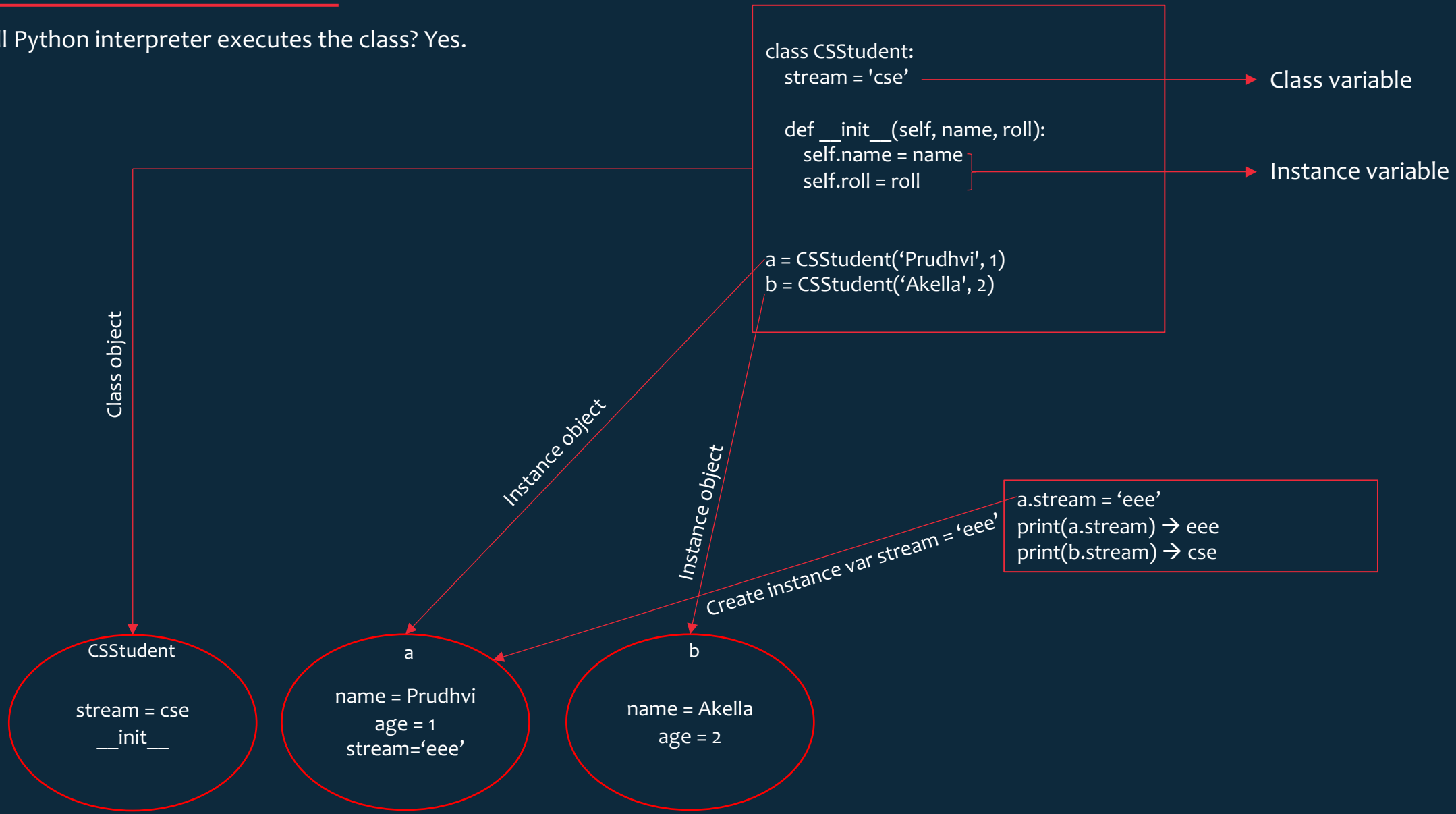
Class Vs Instance Objects

Will Python interpreter executes the class? Yes.

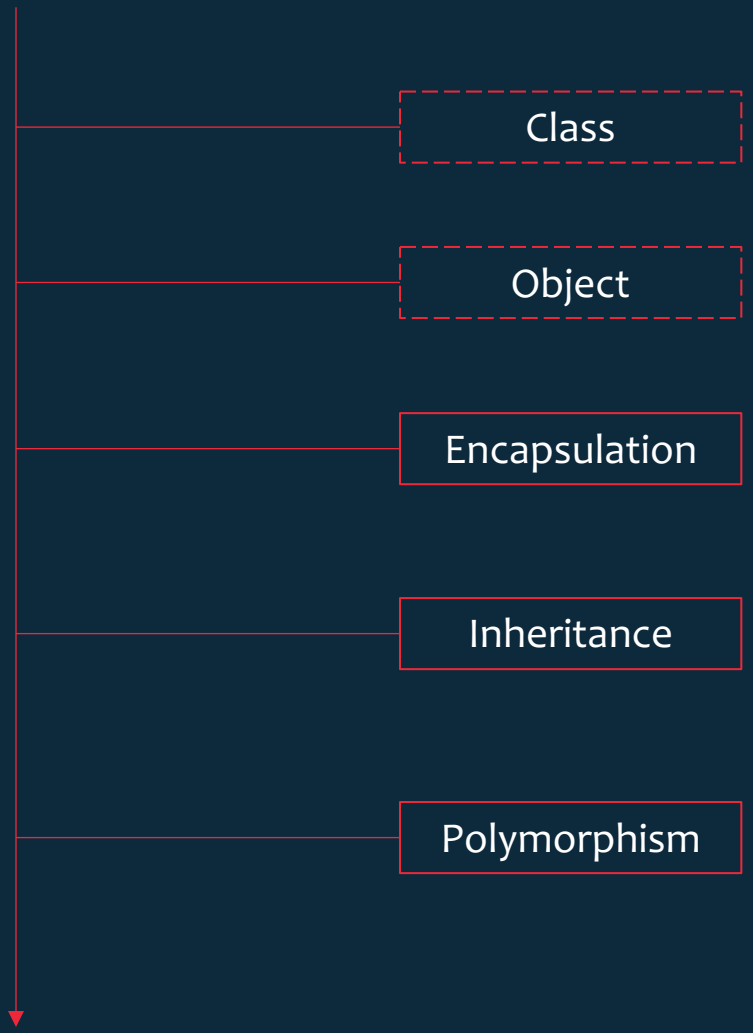


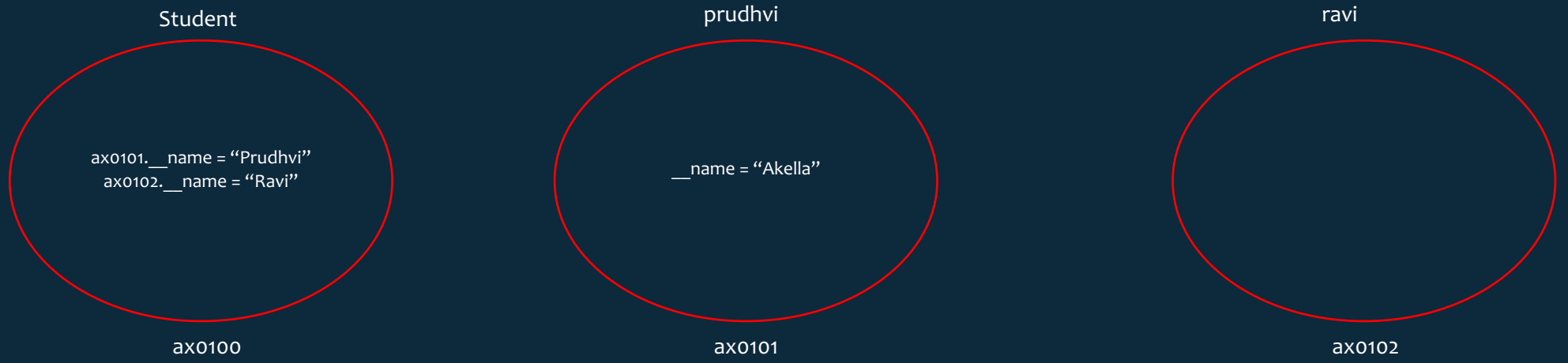
Class Vs Instance Objects

Will Python interpreter executes the class? Yes.



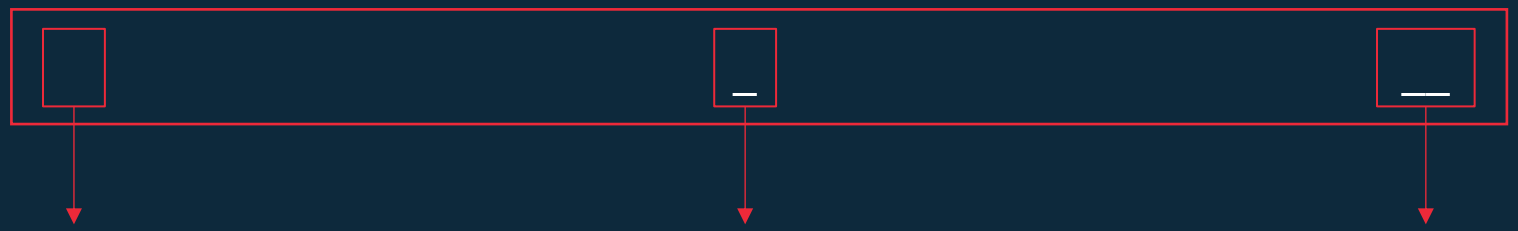
OOP's Principles




Student
ax0101.__name = “Prudhvi”
ax0102.__name = “Ravi”
ax0100
prudhvi
__name = “Akella”
ax0101
ravi
ax0102

Encapsulation(Restrict Access to attributes and methods)

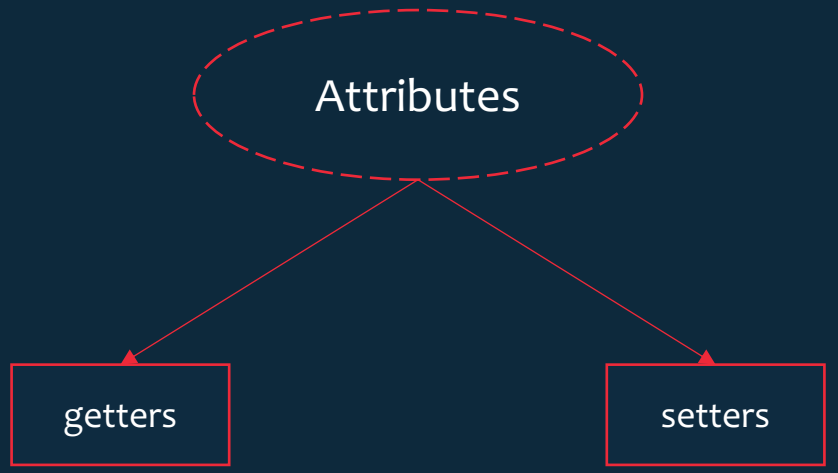
Access Modifiers



Public Members:
Default every member is public

Protected Members:
Can be Accessible in derived class

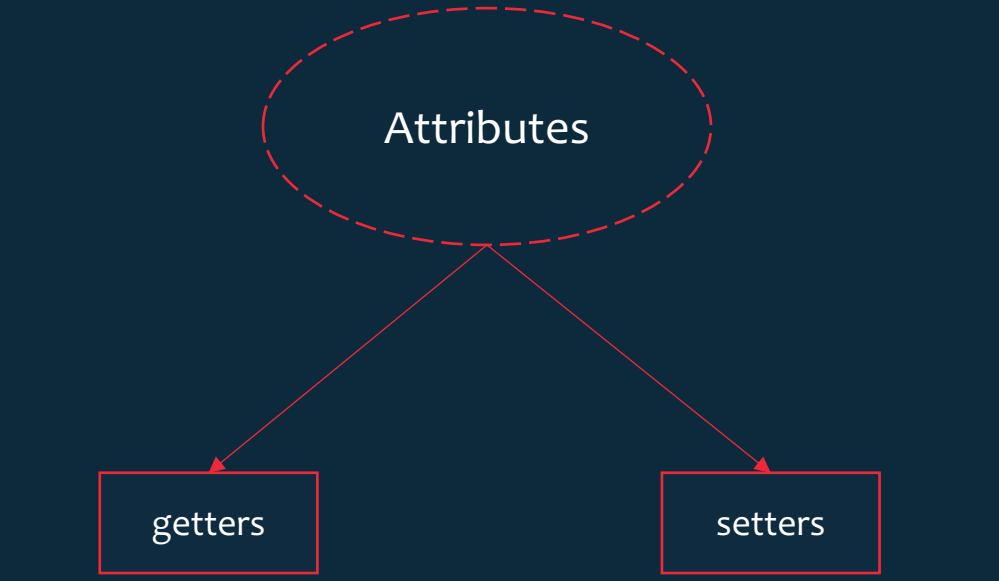
Private Members:
Accessible only with in the class



These methods are used to get the value of attribute of class

These methods are used to set the value to attributes of class

Encapsulation(Restrict Access to attributes and methods)



These methods are used to get the value of attribute of class

These methods are used to set the value to attributes of class

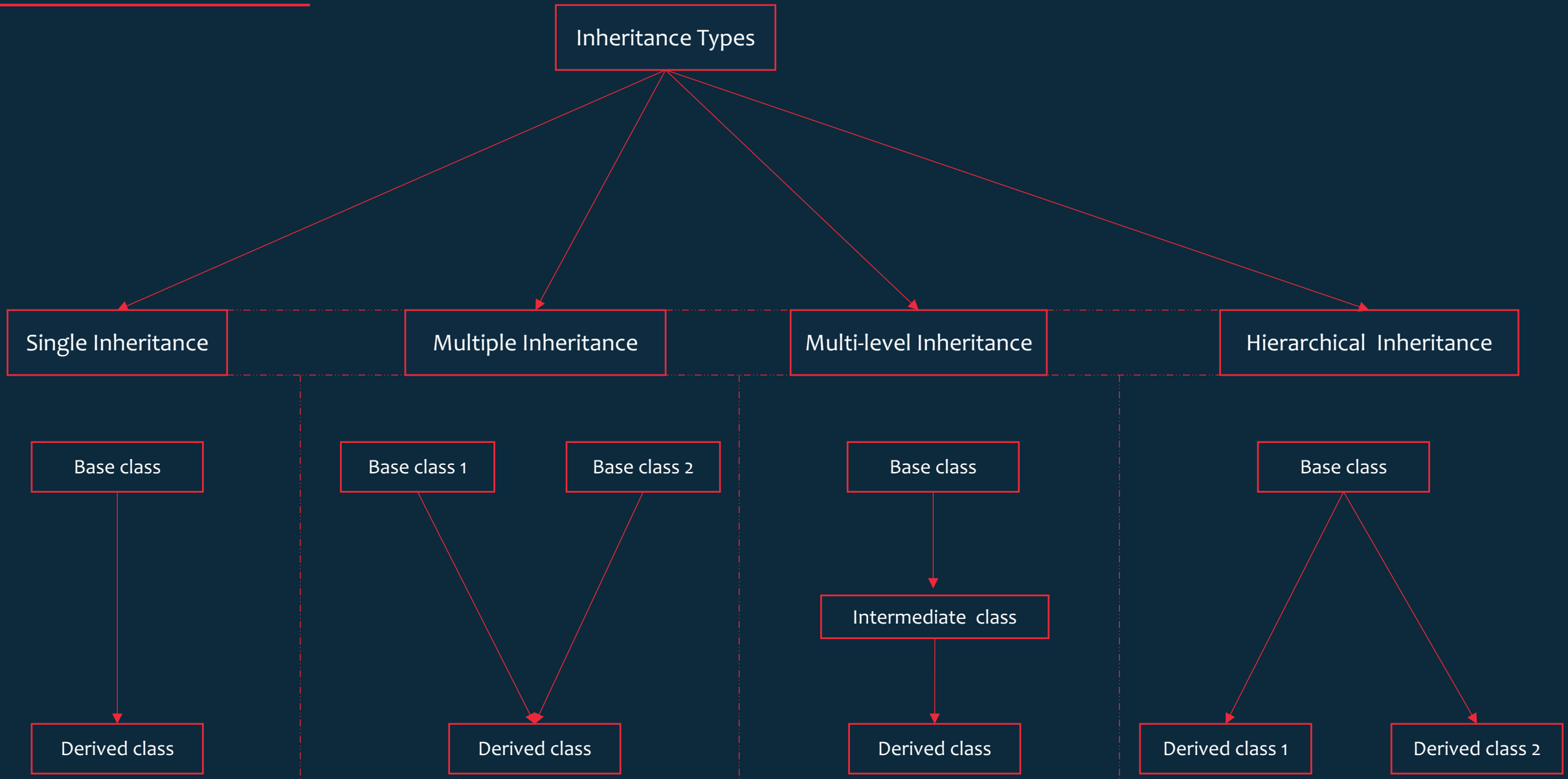
Inheritance

all classes are inherit implicitly from object, if no other parent is declared.

```
class BaseClassName(object):
    <statement-1>
    .
    .
    .
    <statement-N>
```

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

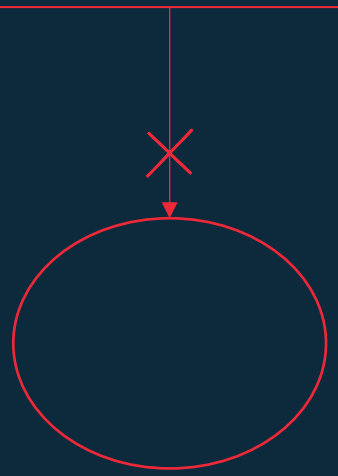
Types of Inheritance

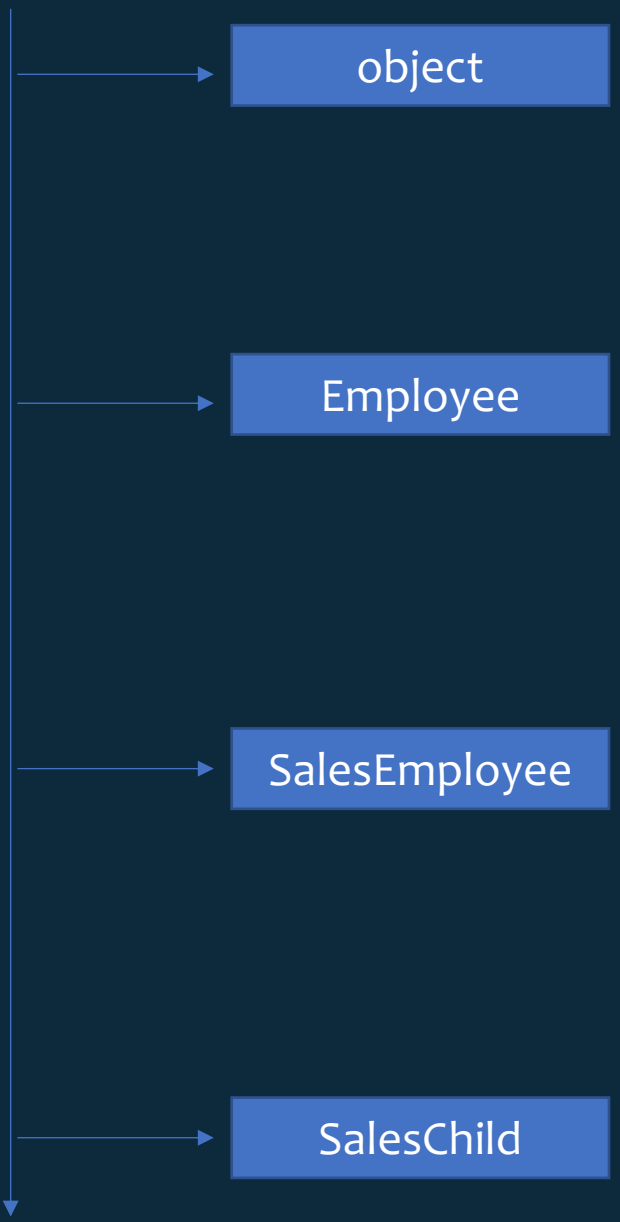


Abstract Class

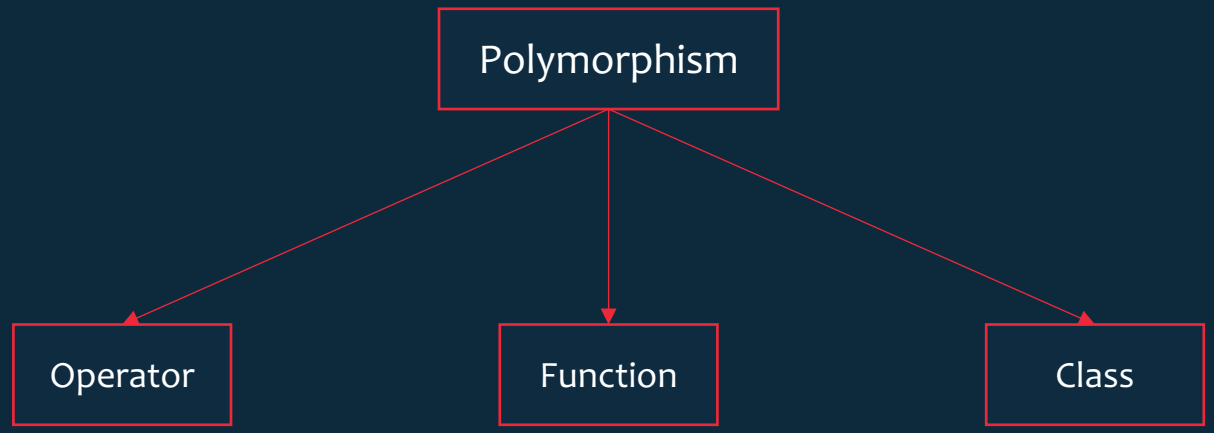
- Abstract class is a class that cannot be instantiated if it contains abstract method. However, you can create classes that inherit from an abstract class.
- Similarly, an abstract method is an method without an implementation. An abstract class may or may not include abstract methods.
- Typically, you use an abstract class to create a blueprint for other classes.
- To define an abstract class, you use the abc (abstract base class) module.

```
from abc import ABC, abstractmethod


class AbstractClassName(ABC):
    @abstractmethod
    def abstract_method_name(self):
        pass
```




object
Employee
SalesEmployee
SalesChild

Polymorphism

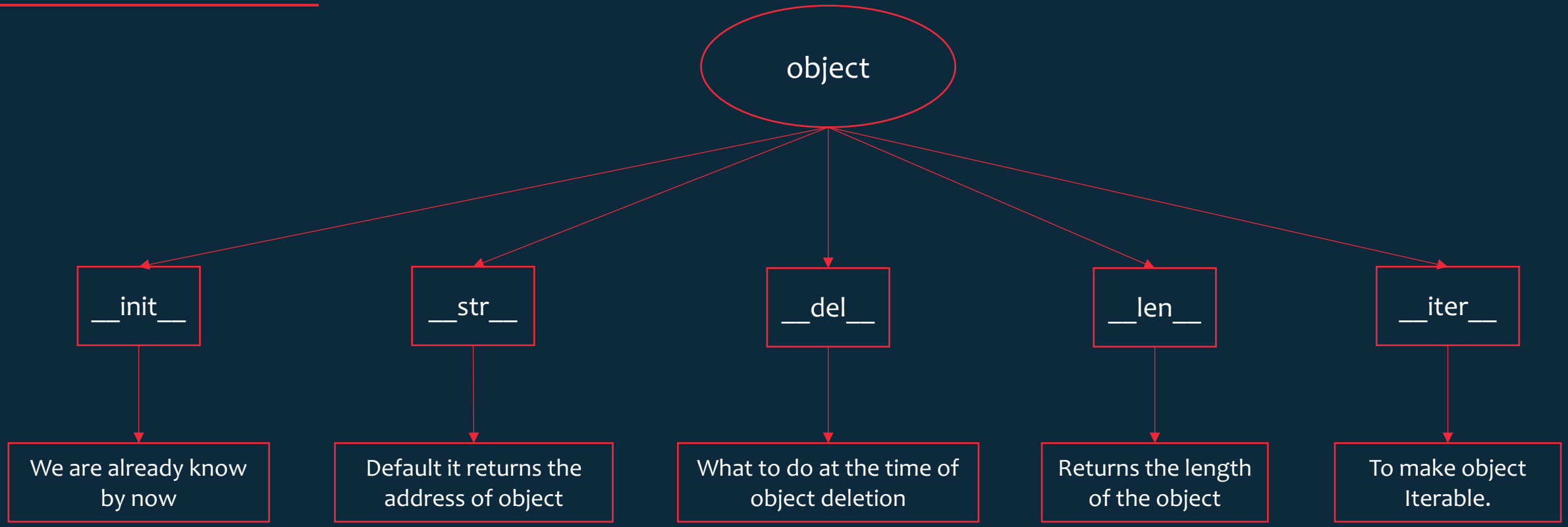


- → +
- → *

- → len()

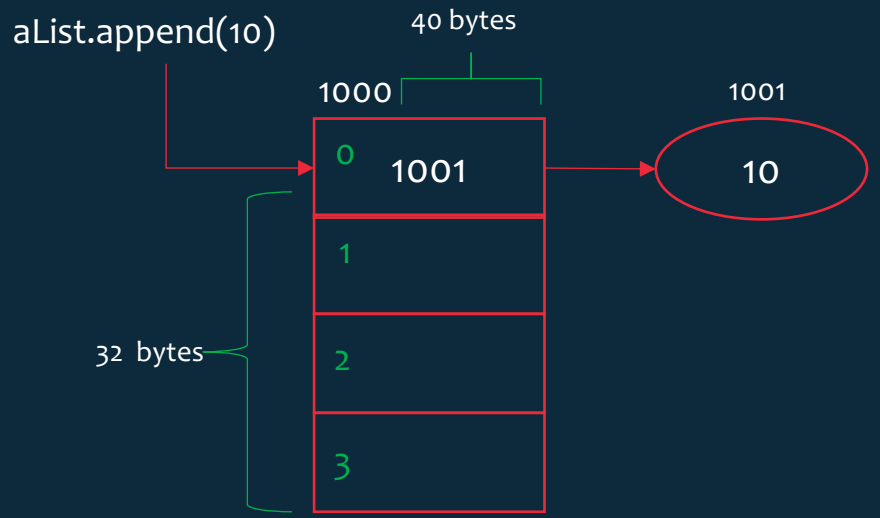
- → Method overriding (run time)
- → Method overloading (compile time)

Magic/Dunder Methods



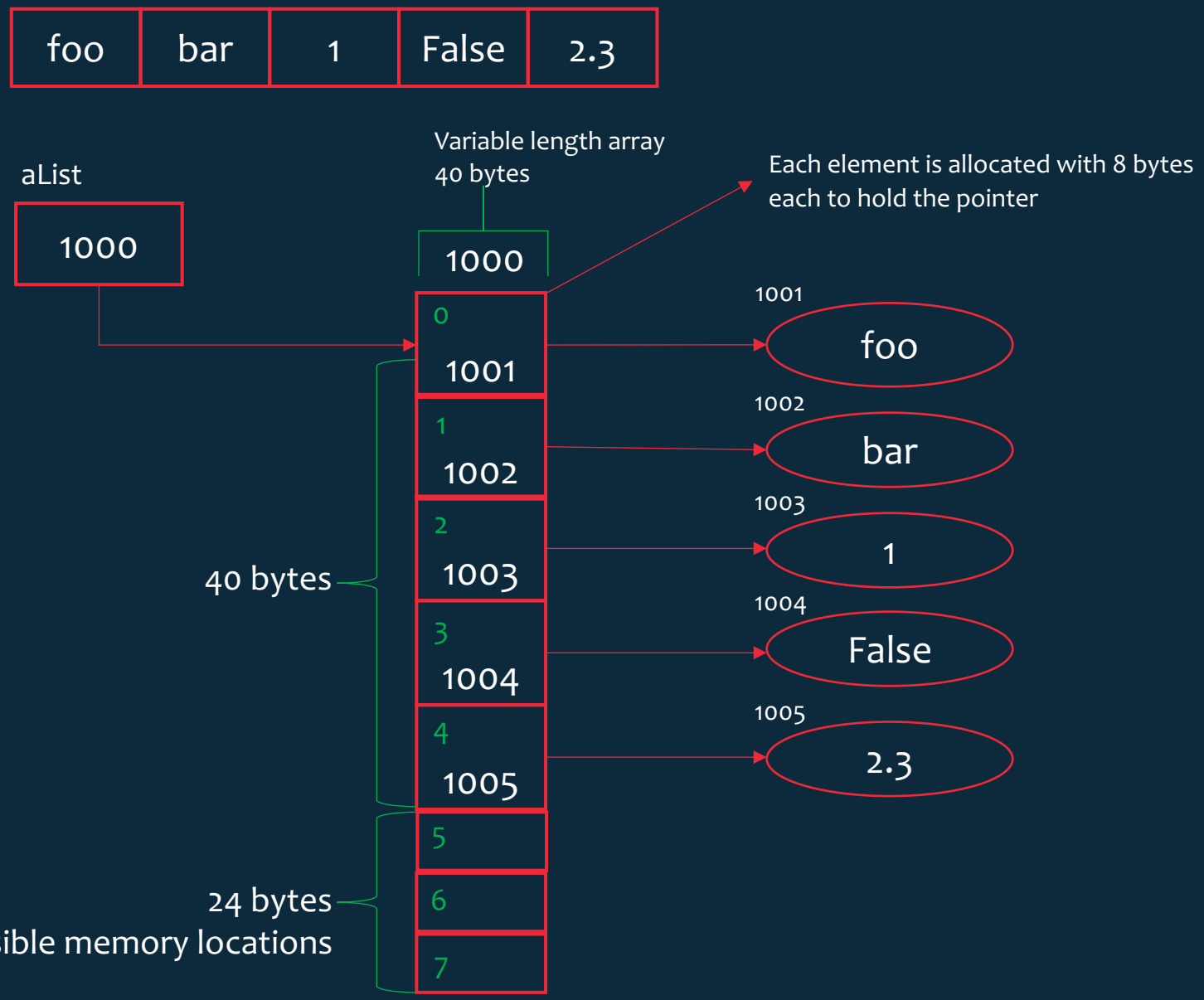
How do lists works internally?

What is the internal representation of list in CPython ?	What is the size of empty list in Python?	Does the list stores the value or pointer of an object?	What is the size of each element in list?	How to check the size of list?	How to add elements to list
Variable length Array	**40 Bytes** (It is used for internal variables, Functions, Garbage collection options)	Pointer/address of an object	8 bytes Note: Original object size depends on the type of the value	Using __size__() method	append() extend() Insert()



How do lists works internally?

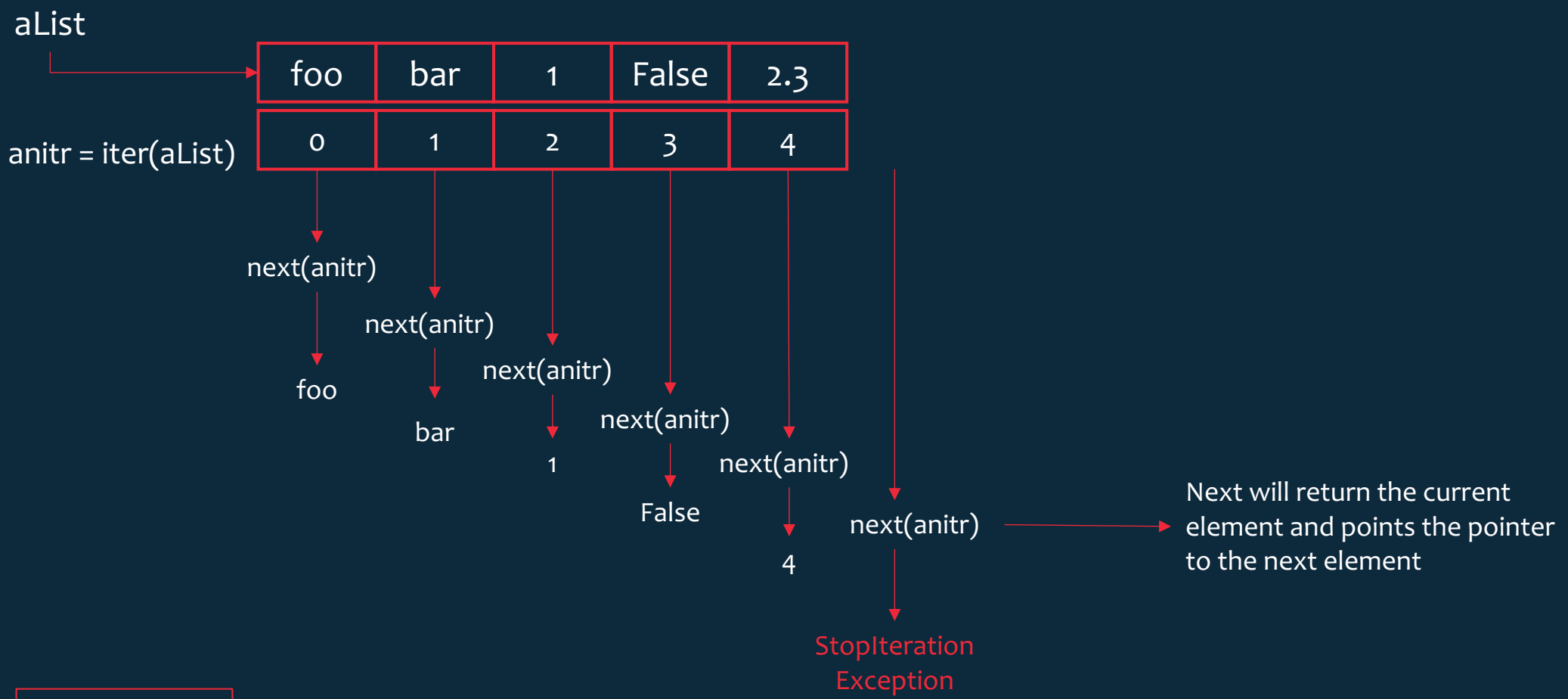
Total size of list with 5 elements is 104(40 + 40 + 24)



List Memory Management Chart

Set Number	Number of Items of a set	Size of Set (in bytes)		Total List Size (in bytes)
0	Empty List	40		40
1	4	32		72
2	4	32		104
3	8	64		168
4	8	64		232
5	8	64		296
6	8	64		360
7	12	96		456
8	12	96		552
9	12	96		648
10	16	128		776
11	16	128		904
12	20	160		1064
13	20	160		1224
14	24	192		1416
15	28	224		1640
16	32	256		1896
17	36	288		2148

Let us understand __iter__ in detail with an example

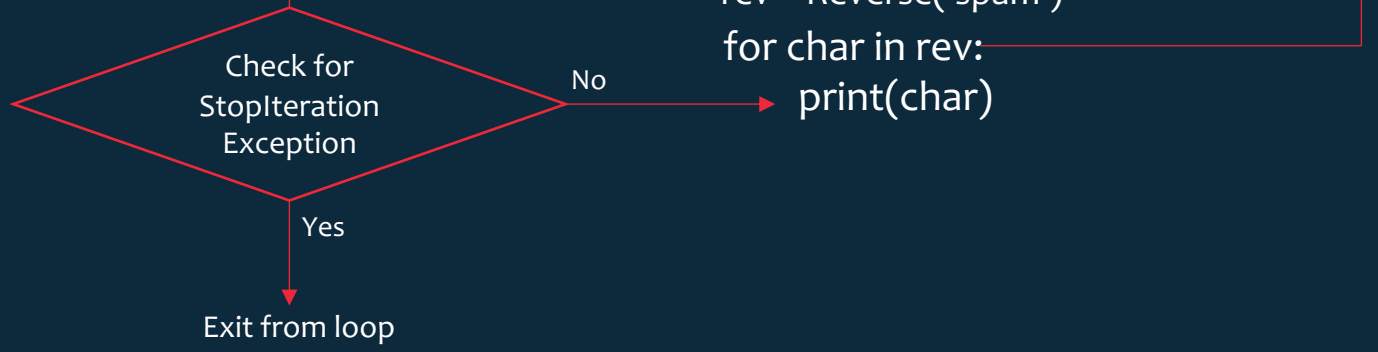



```
for i in aList:
    print(i)
```

For will also uses next to iterate over the list

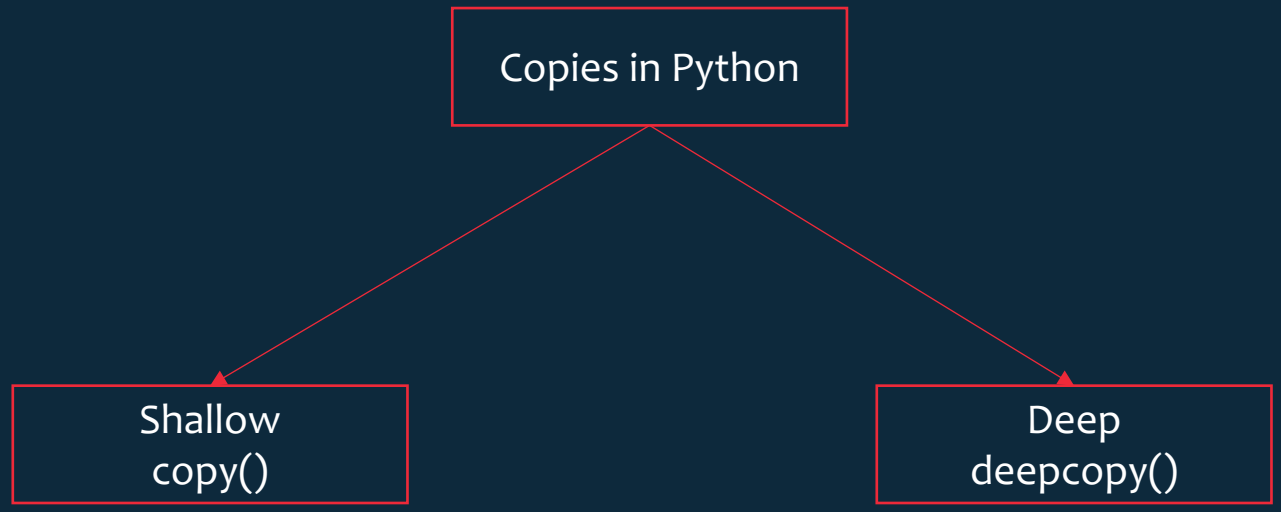
Lets make object Iterator

```
class Reverse:
    """Iterator for looping over a sequence backwards."""
    def __init__(self, data):
        self.data = data
        self.index = len(data)

    def __iter__(self):
        return self

    def __next__(self):
        if self.index == 0:
            raise StopIteration
        self.index = self.index - 1
        return self.data[self.index]
```



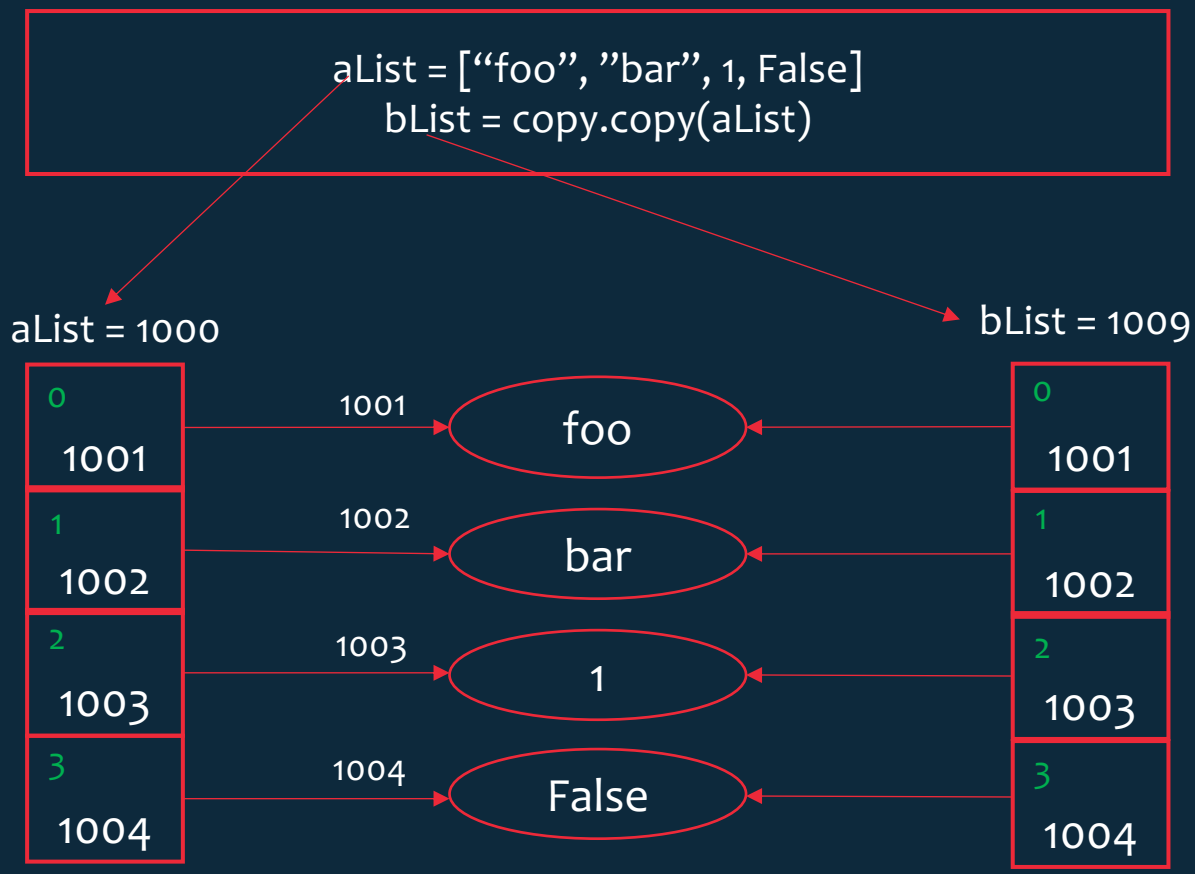

```
rev = Reverse('spam')
for char in rev:
    print(char)
```

Shallow and Deep Copy



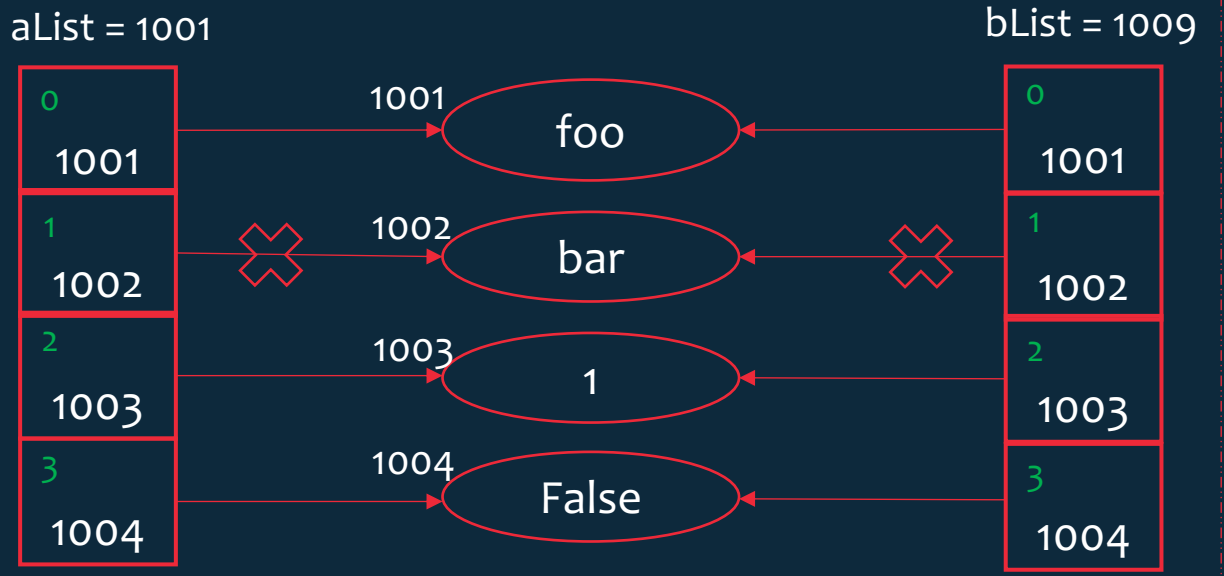
Shallow Copy

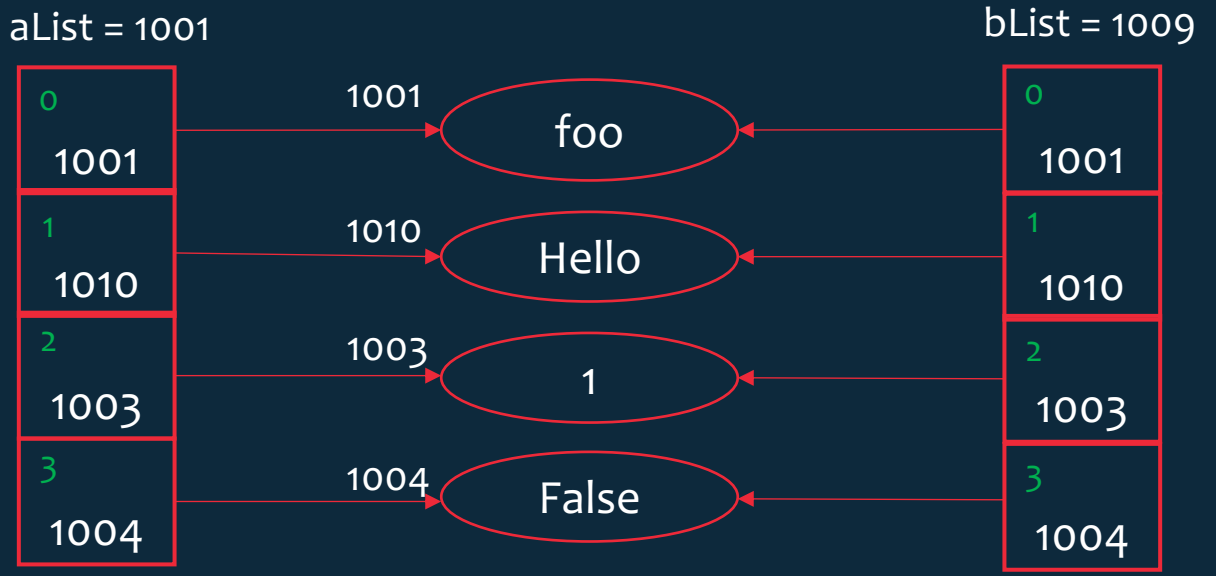
A shallow copy means constructing a new collection object and then populating it with references to the child objects found in the original



Shallow Copy

```
aList = [“foo”, ”bar”, 1, False]
bList = copy.copy(aList)
aList[1] = “Hello”
```



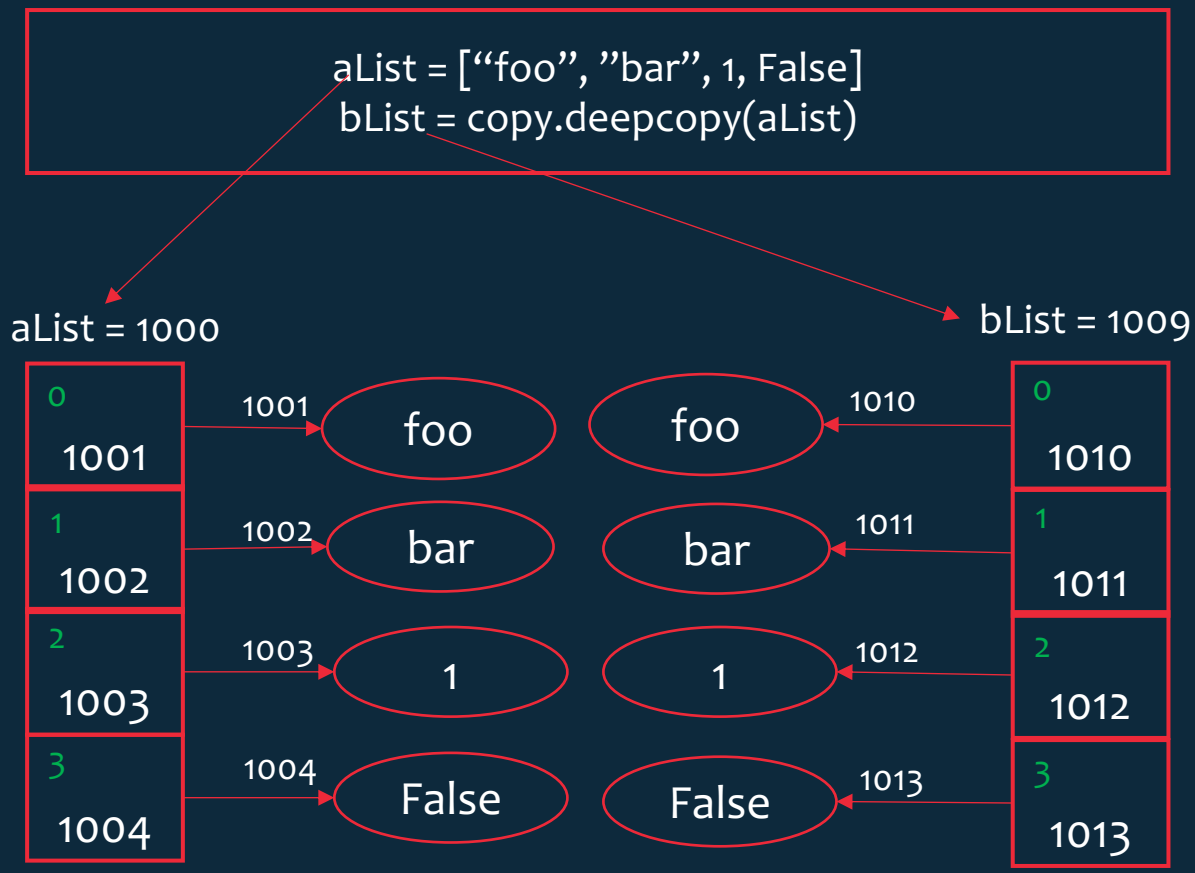


Shallow Copy

```
aList = [“foo”, ”bar”, 1, False]
bList = copy.copy(aList)
bList.append(True)
```

Deep Copy

A deep copy makes the copying process recursive. It means first constructing a new collection object and then recursively populating it with copies of the child objects found in the original. Copying an object this way walks the whole object tree to create a fully independent clone of the original object and all of its children.



Class and static Method

→ A class method is a method that is bound to a class rather than its object. It doesn't require creation of a class instance.

→ A static method, much like class method, are methods that are bound to a class rather than its object.

When to use class methods?

1) Factory Methods
2) Correct instance creation in inheritance

When to use static methods?

1) Grouping utility function to a class

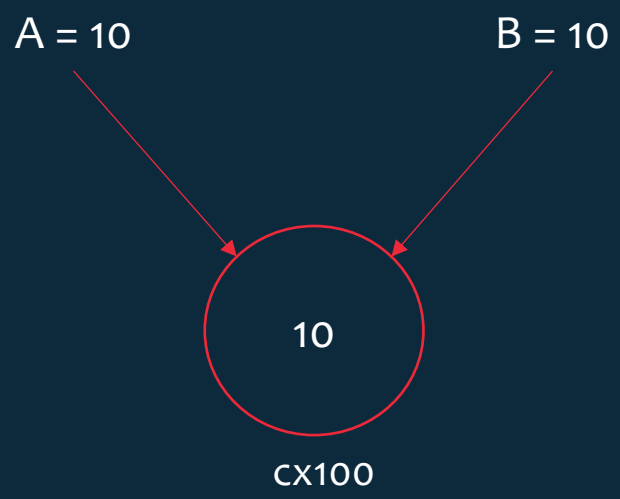
SOLID Priniciples

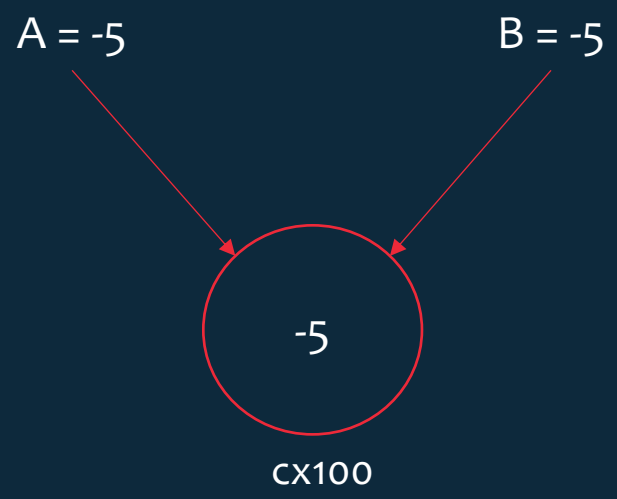
These principles you can improve the reliability of your code by working on its structure and its logical consistency. the SOLID principles are basic learning steps for every code developer but are usually ignored by those who does not consider the highest quality of code their absolute priority

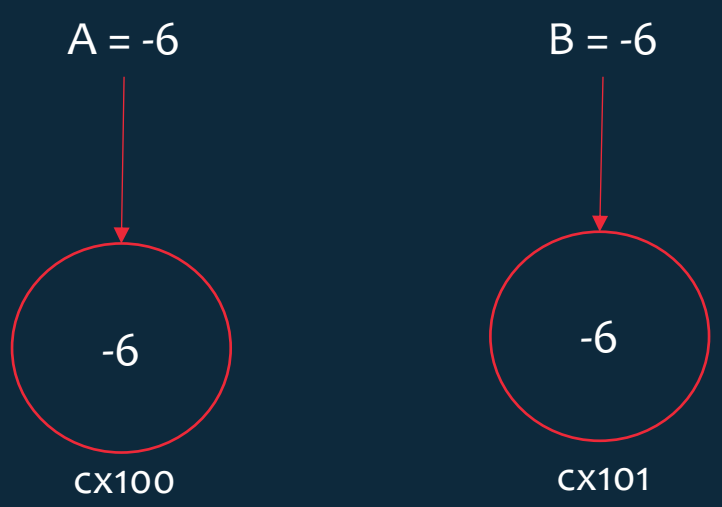
Single-Responsibility Principle (SRP)
Open-Closed Principle (OCP)
Liskov Substitution Principle (LSP)
Interface Segregation Principle (ISP)
Dependency inversion Principle (DIP)

Integer Interning(Optimization in Python)

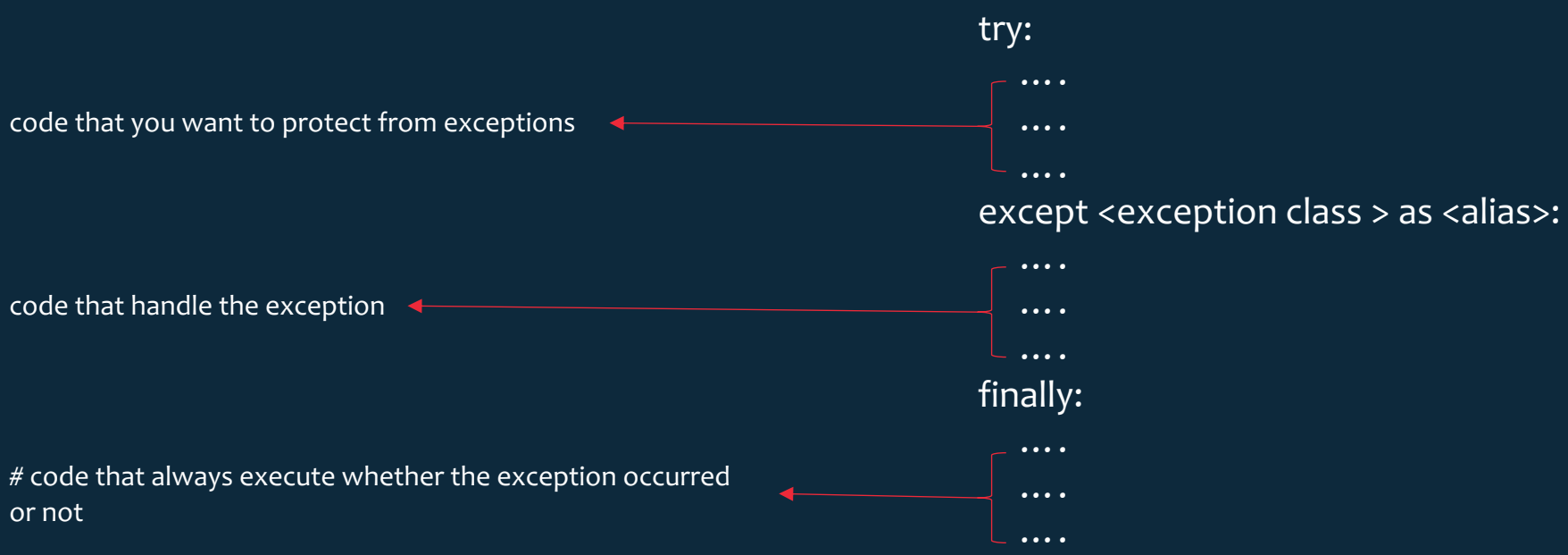
Range (– 5 , 256)







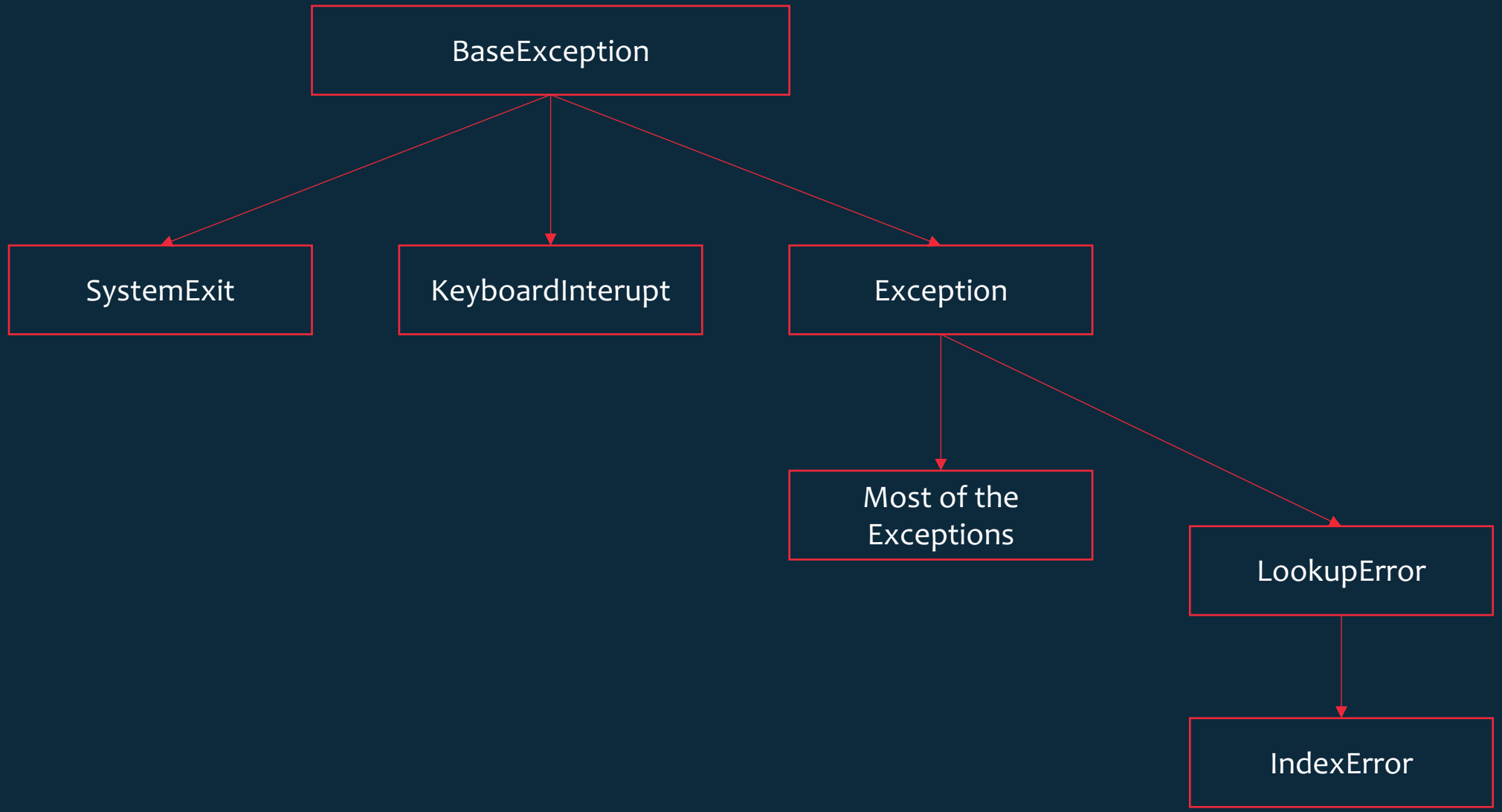
Expection Handling



Note:

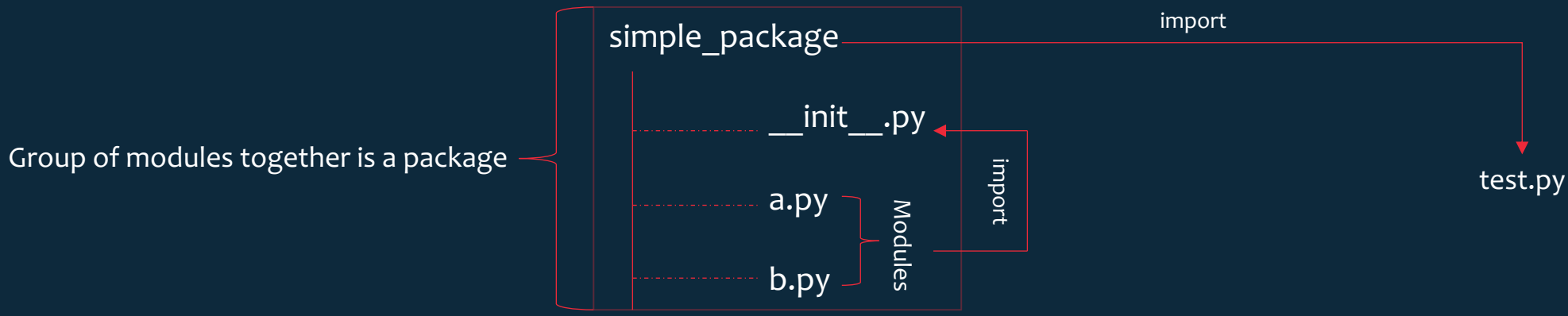
- You cannot define try block alone without except block and viceversa
- If you wanted to interrupt a program when an exception is encountered it needs to be raised using **raise** key word

Exception class lineage



Modules and Packages

Any Python file can be referenced as a module.



Python Memory Management : Reference Count

counter = 100

0x20b5ee5f730

References	Count
0x20b5ee5f730	1

max = counter

0x20b5ee5f730

References	Count
0x20b5ee5f730	2

max = 999

0x28ea71b0030

References	Count
0x20b5ee5f730	1
0x28ea71b0030	1

counter = 1

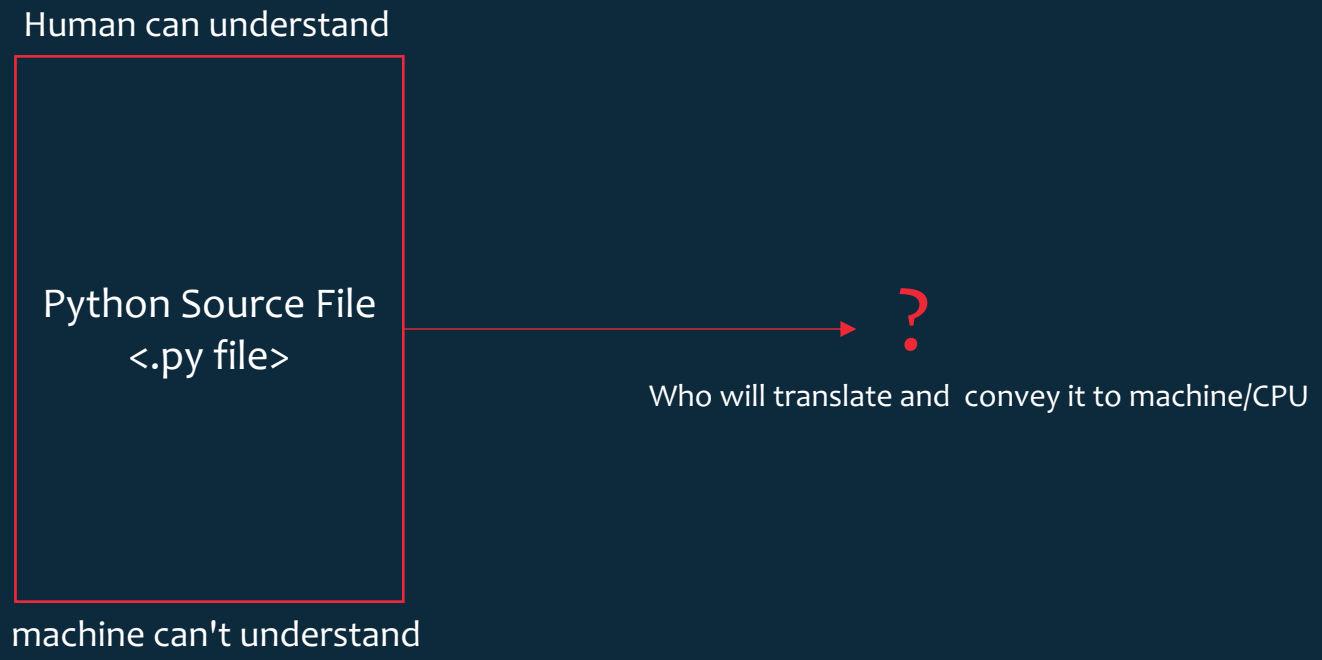
0x28ea70b0032

References	Count
0x20b5ee5f730	0
0x28ea71b0030	1
0x28ea70b0032	1

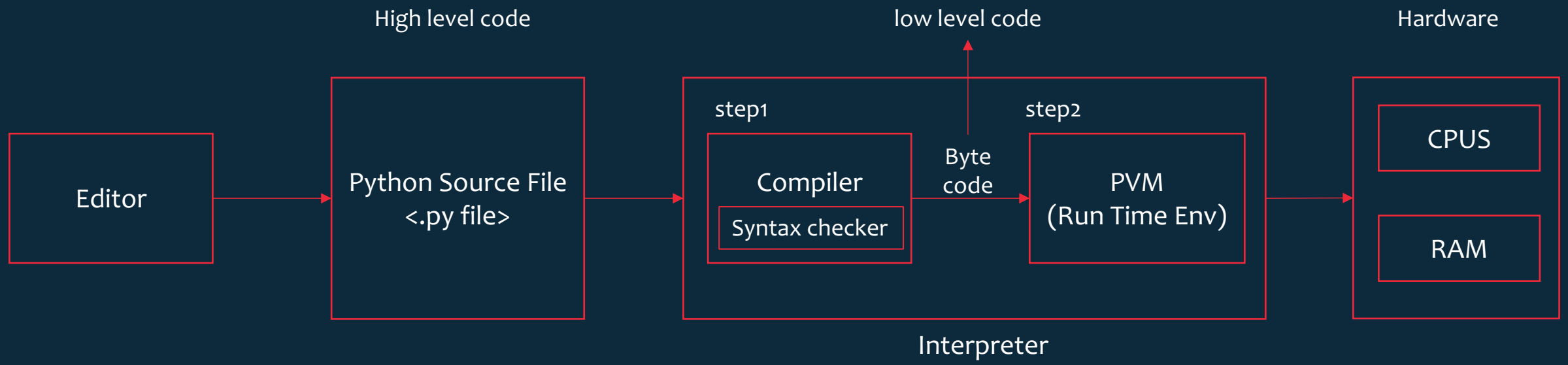
Now Who will clear the Zero referenced Objects?

Lets take one step back and understand Python program execution flow

Python Execution



Python Execution

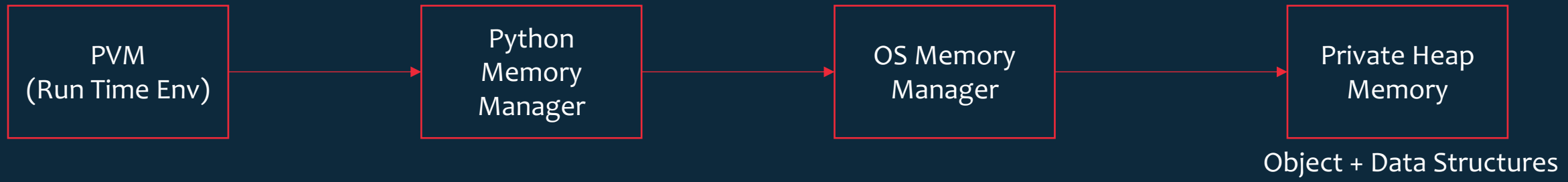


Step1 : Source file will be an input to in-built compiler which will parse the file and does a syntax validation for each statement. If syntax looks ok ".py" file will be converted in byte code which is an intermediate code and it will be stored into .pyc file.

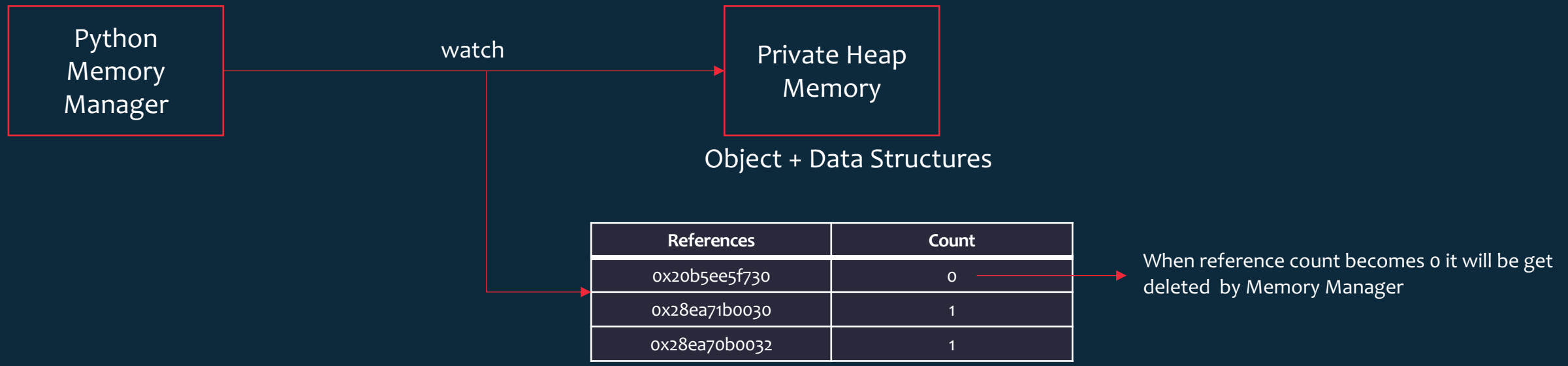
Step2: Python Virtual Machine is the runtime engine of Python. It goes through your byte code instructions one by one and carries out the operations stated in those instructions.

Still the question is not answered yet?

Python Memory Management



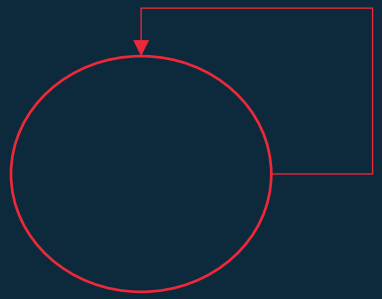
Python Memory Management



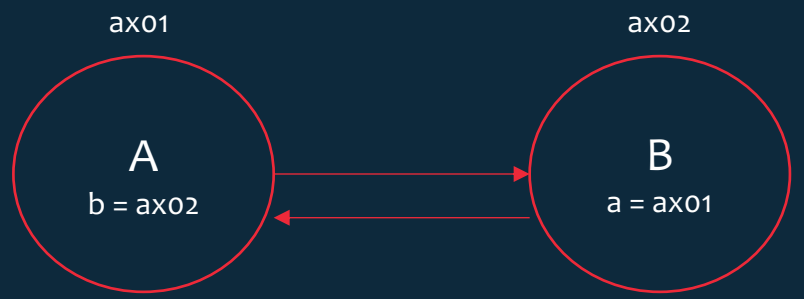
Python Memory Management

Limitations of Python Memory Manager

- It cannot identify Circular References which leads to Memory leaks



Object refer by itself



Two objects reference each other

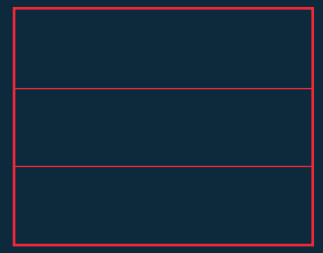
To re-claim the memory for these circular references python's memory manager uses garbage collector

LifoQueue : Stack Implementation in Python

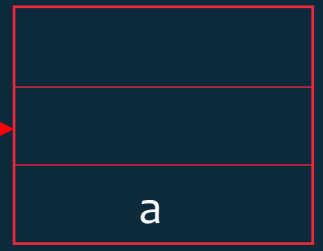
Stack is a data structure with LIFO (Last In First Out) order it has two principles:

put: adds an element to the stack
get: removes the most recently added element

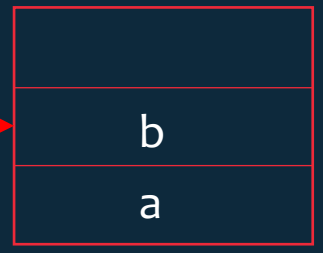
stack = LifoQueue(maxsize=3)



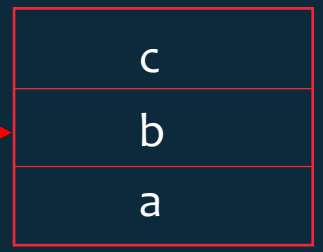
stack.put('a')



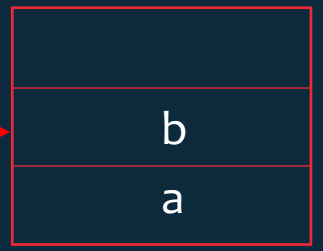
stack.put('b')



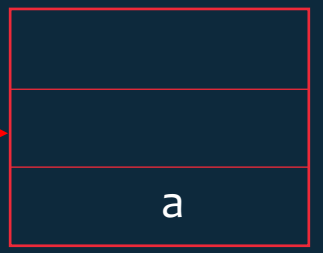
stack.put('c')



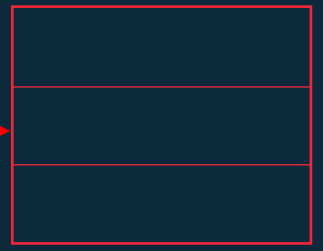
stack.get()



stack.get()

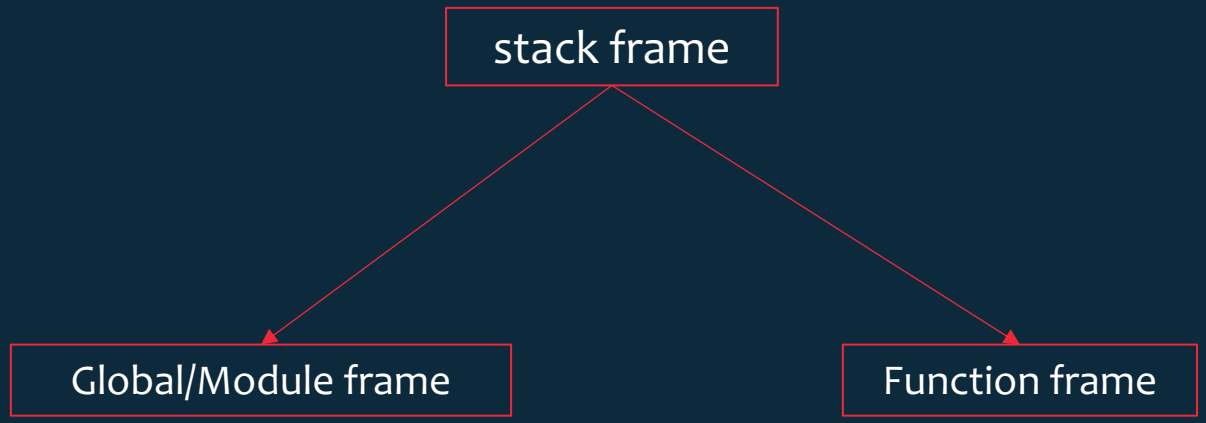


stack.get()



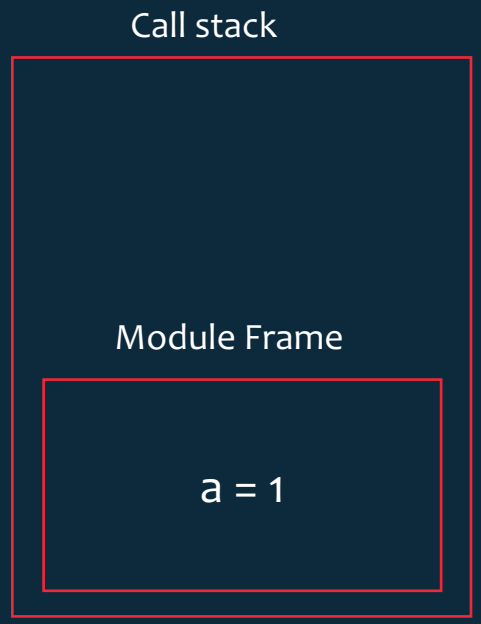
Python Memory Management : stack frame

Stack Memory in RAM



Python Memory Management : stack frame

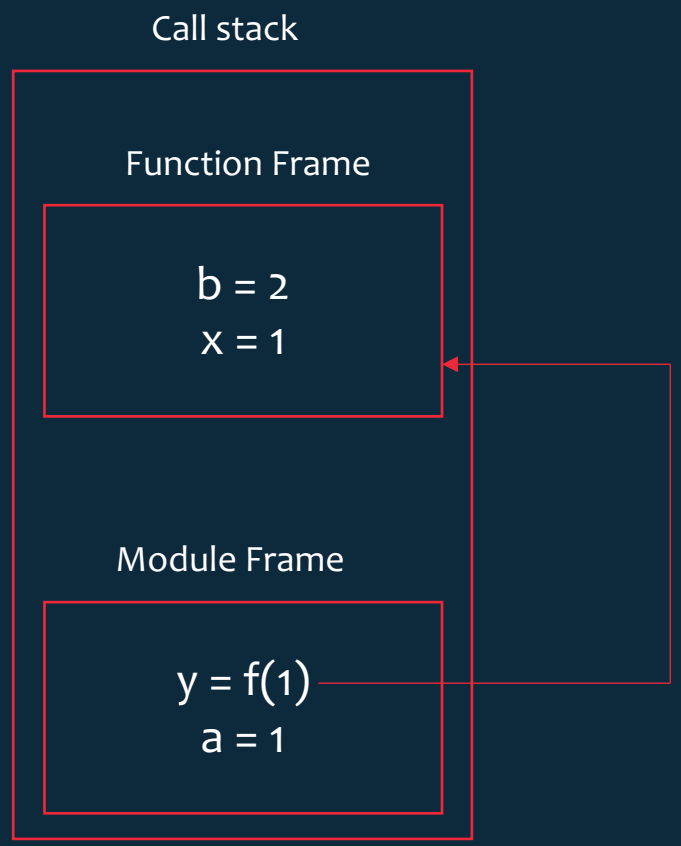
```
a = 1

def f(x):
    b = 2
    return b+x

y = f(a)
print(y)
```



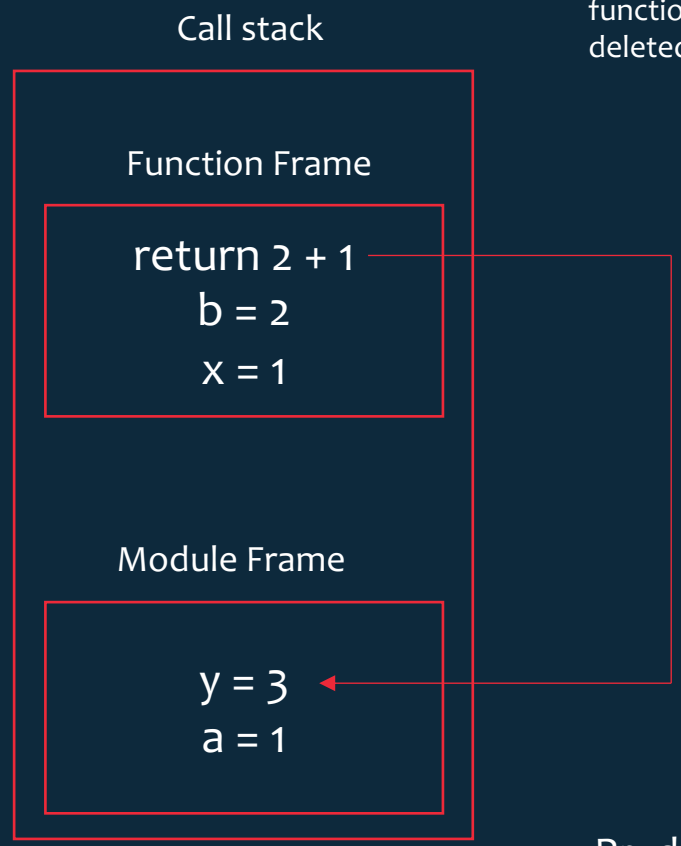
Python Memory Management : stack frame

```
a = 1

def f(x):
    b = 2
    return b+x

y = f(a)
print(y)
```

When a function is called function frame will be created



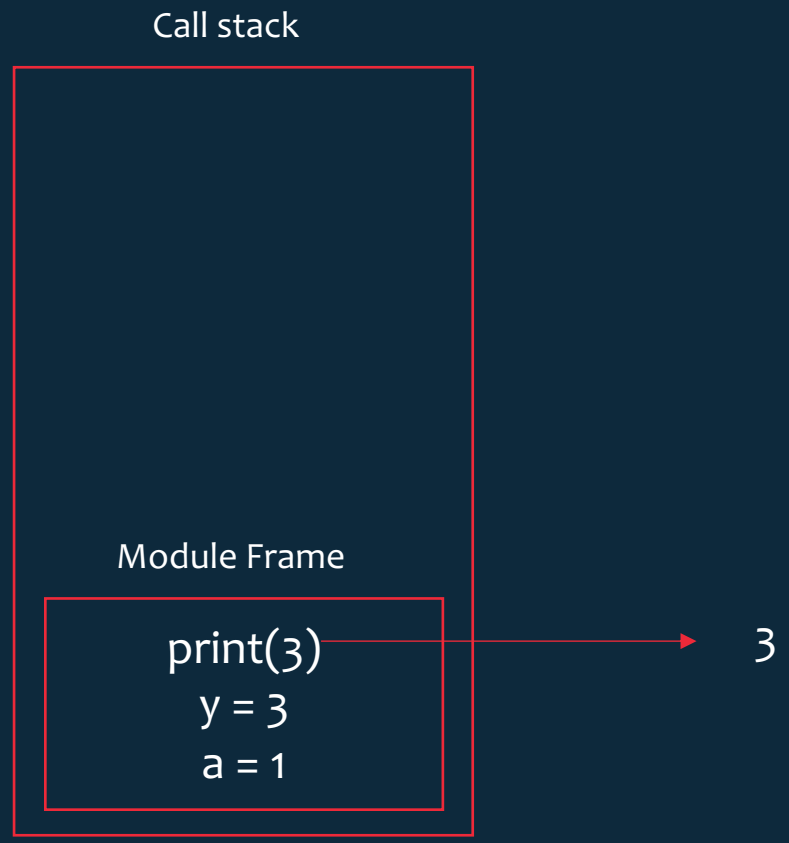
When a function is return function frame will be deleted



Python Memory Management : stack frame

```
a = 1

def f(x):
    b = 2
    return b+x

y = f(a)
print(y)
```

At the end of the program module frame will be deleted



Now lets go back to our question.

how did "Inside Biology Module" got printed on to the console?

When a first module loads into the another module python does two things

1) Its generates .pyc file
2) It create module frame and executes the module
3) .pyc are automatically generated by the interpreter when you import a module, which speeds up future importing of that module. These files are therefore only created from a .py file if it is imported by another .py file or module. However if you want to see them run “python3 -m compileall .”
4) Where to find .pyc file? Go to the module location in windows file explorer and you should see __pycache__ folder must contain .pyc files related to that module.

venv : virtual environment

- Allow you to manage separate package installations for different projects

- Avoids the need to install Python packages globally

- How to activate venv?
  - Navigate to your project's venv/Scripts folder in command line and run activate command so that venv will be activated
  - Once its successfully activated you can start installing all your packages
    using "pip install <package name>"
  - Once all the packages are installed then run "deactivate" script in the same folder/directory from the command line which will deactivate the venv.
  - pip freeze > requirements.txt its used to generate the requirements.txt file. It consist of all the external packages user wants to install along with it's version
  - How to install packages in requirements.txt file?
  - pip install – r requirements.txt

What is Mutability and Immutability ?

- An object whose internal state can be changed is called a mutable object.
- An object whose internal state cannot be changed is called an immutable object.

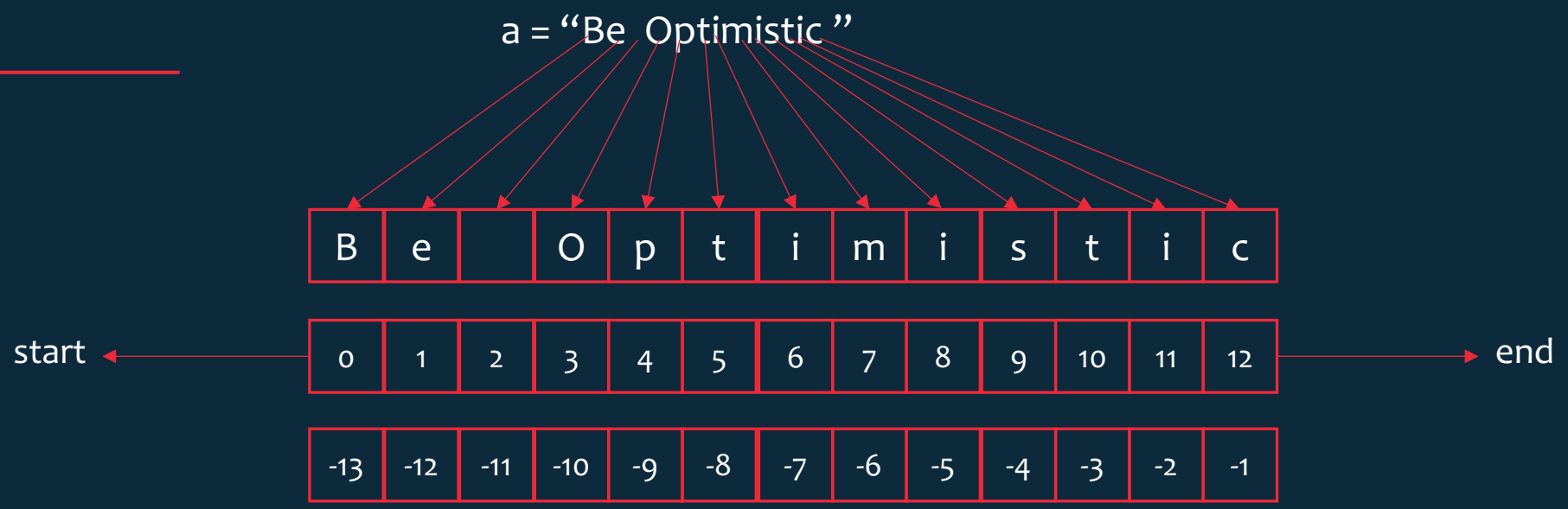
Examples of Immutable Types in Python

- int, float, bool
- Strings
- Tuples
- Frozen sets

Examples of Mutable Types in Python

- Lists
- Sets
- Dictionaries

Strings



Indexing

a[0] → B

a[10] → t

a[6] → I

a[-1] → C

a[-7] → I

a[-9] → P

Slicing

string[start : end]

a[0 : 4] → “Be O” # From the start till before the 4th index

a[1 : 7] → “e Opti”

a[7 : len(a)] → “mistic” # From the 8th index till the end

a[: 8] → “Be Optim” # From the 0th index till 8th index

a[8 :] → “istic” # from 8th index till end of the string

a[:] → “Be Optimistic” # whole string

a[-8 : -1] → “timisti”

a[: -10] → “Be ”

a[-10:] → “Optimistic”

Step size

string[start : end : step]

a[0 : 7 : 2] → “B pi” # step of 2

a[0 : 7 : 5] → “Bt” # Step of 5

Negitive Step

a[13 : 2 : -1] → “citsimitpO”

Reverse a String

a[::-1] → “citsimitpO eB”

Strings

- Built-in functions
- Interpolating Variables into String

String Methods

- Case conversion
- Find and Seek
- Character Classification
- String Formatting
- Converting String to List

Containers

Lists

A List is created by placing all the items (elements) inside square brackets []

- Accessing / Overwriting elements of a list using index
- Applying slicing and step on a list
- Usage of len() function on a List
- Concat multiple lists into one list
- Appending new elements to a list
- Popping/removing elements from a list
- Adding new elements to the head and middle of a list
- Sorting a list in a ascending and descending order
- Converting a List of string elements to a String using join method
- Splitting a String into a List

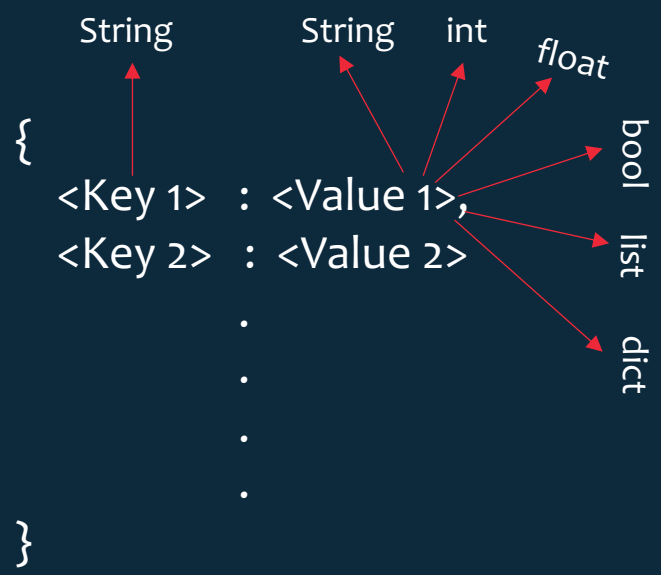
Tuple

A tuple is created by placing all the items (elements) inside parentheses ()

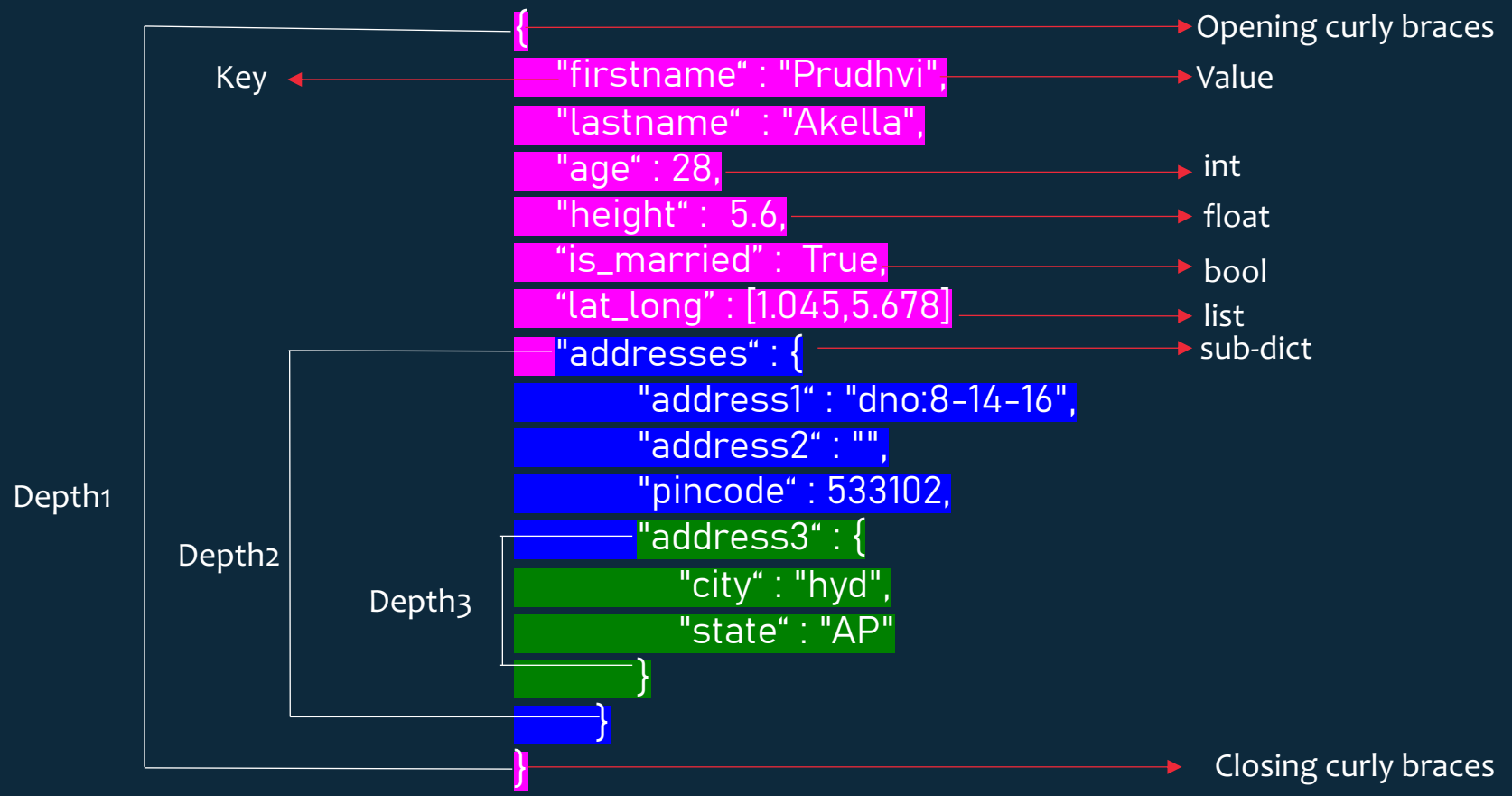
- Creating a tuple
- Tuple packing and un-packing
- Creating a tuple with one element is always tricky
- Accessing tuple elements
  - Indexing
  - Negative indexing
  - Slicing
  - Stepping
- Concatenation multiple tuple into one
- Reflecting Tuple
- Tuple Method
  - count
  - index
- Using Membership operator (‘in’)
- Iterating over tuple
- Size of a Tuple

Dictionaries

A Dictionary is created by placing all the items (key : value pair) inside curly brackets {}



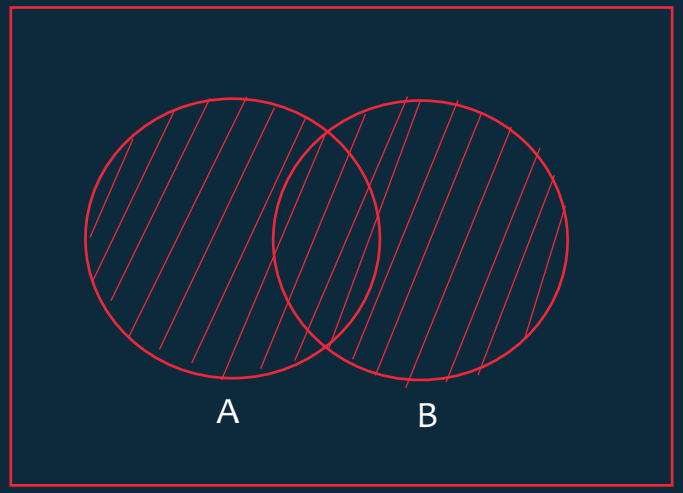
Dictionaries



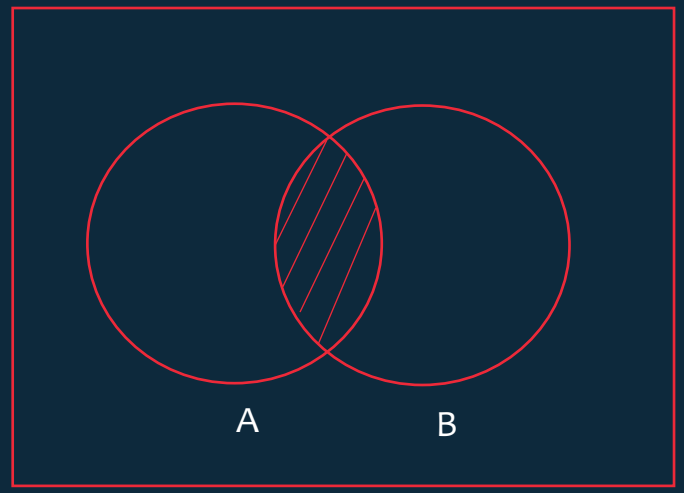
Sets

A set is created by placing all the items (elements) inside curly braces {}, separated by comma

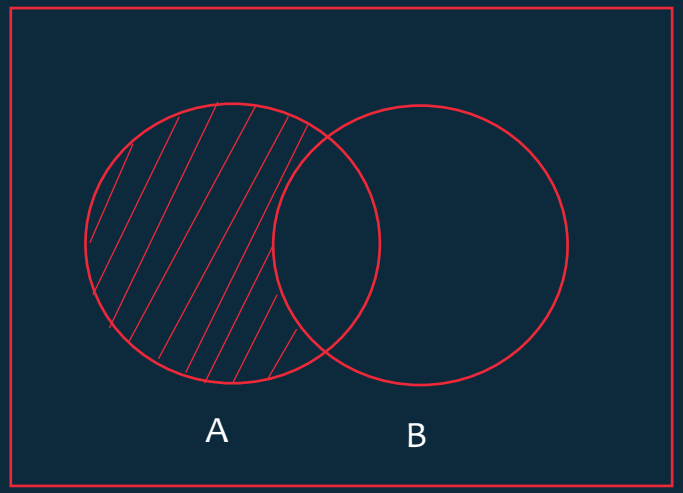
- Creating Python Sets
- Modifying a set in Python
- Removing elements from a set
- Python Set Operations
  - Union
  - Intersection
  - Difference
  - Symmetric Difference
- Iterating Through a Set
- Frozen set



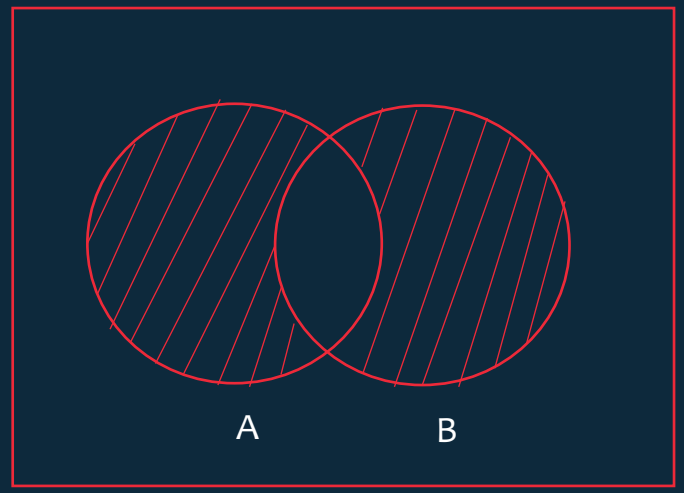
Union



Intersection



Difference



Symmetric Difference

Generators Functions (Yield)

When a function contains at least one yield statement, it's a generator function.

- Python generators are functions that contain at least one yield statement.
- A generator function returns a generator object.
- A generator object is an iterator. Therefore, it becomes exhausted once there's no remaining item to return.
- Generators are lazy iterators. Therefore, to execute a generator function, you call the next() built-in function on it
- The yield statement is like a return statement in a function. However, there's a big difference.
    - When Python encounters the yield statement, it returns the value specified in the yield. In addition, it pauses the execution of the function.
    - If you "call" the same function again, Python will resume from where the previous yield statement was encountered.

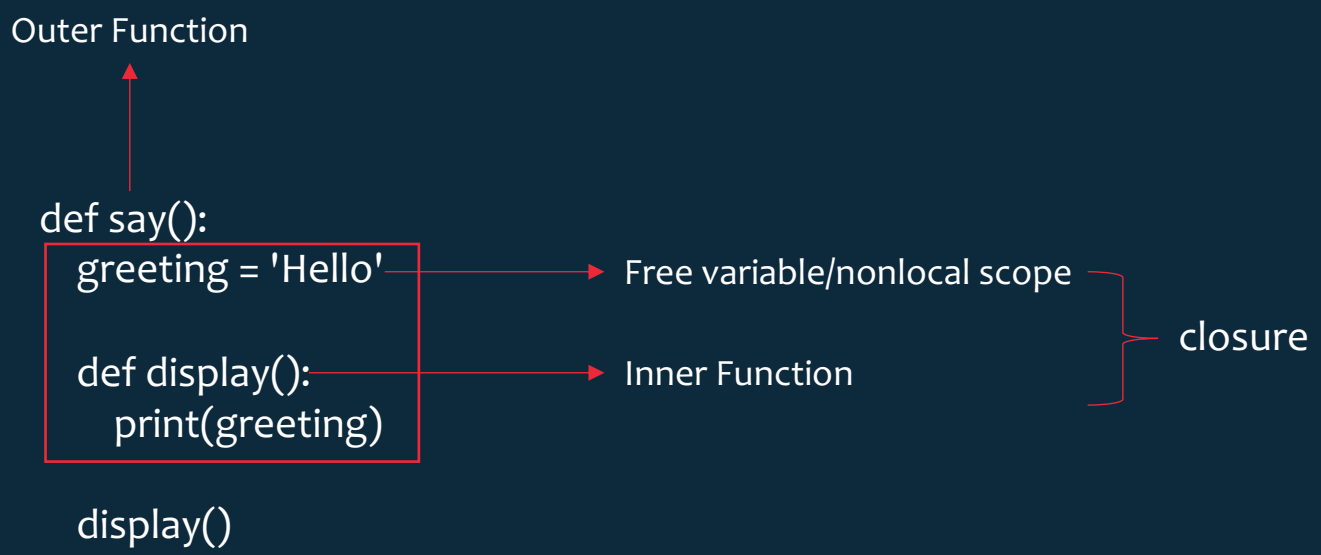
Generators Expressions (Yield)

A generator expression is an expression that returns a generator object.

- A generator expression is like a list comprehension in terms of syntax. For example, a generator expression also supports complex syntaxes including:
  - if statements
  - Multiple nested loops
  - Nested comprehensions
- However, a generator expression uses the parentheses () instead of square brackets []

Closures

a closure is a nested function that references one or more variables from its enclosing scope.



Closures(Cells and Multi scope variables)

Python create an Intermediate object call “cell” to handle multiple Scope variables

```
def say():
    greeting = 'Hello'
    aInt = 10

    def display():
        print(greeting)
        print(aInt)

    display()
```

Multi scope variables(Shared by say and display functions)

<cell at 0x000001C5BCF7BFD0: int object at 0x000001C5BC5F0210>, <cell at 0x000001C5BCF7AC50: str object at 0x000001C5BCAFFAF0>

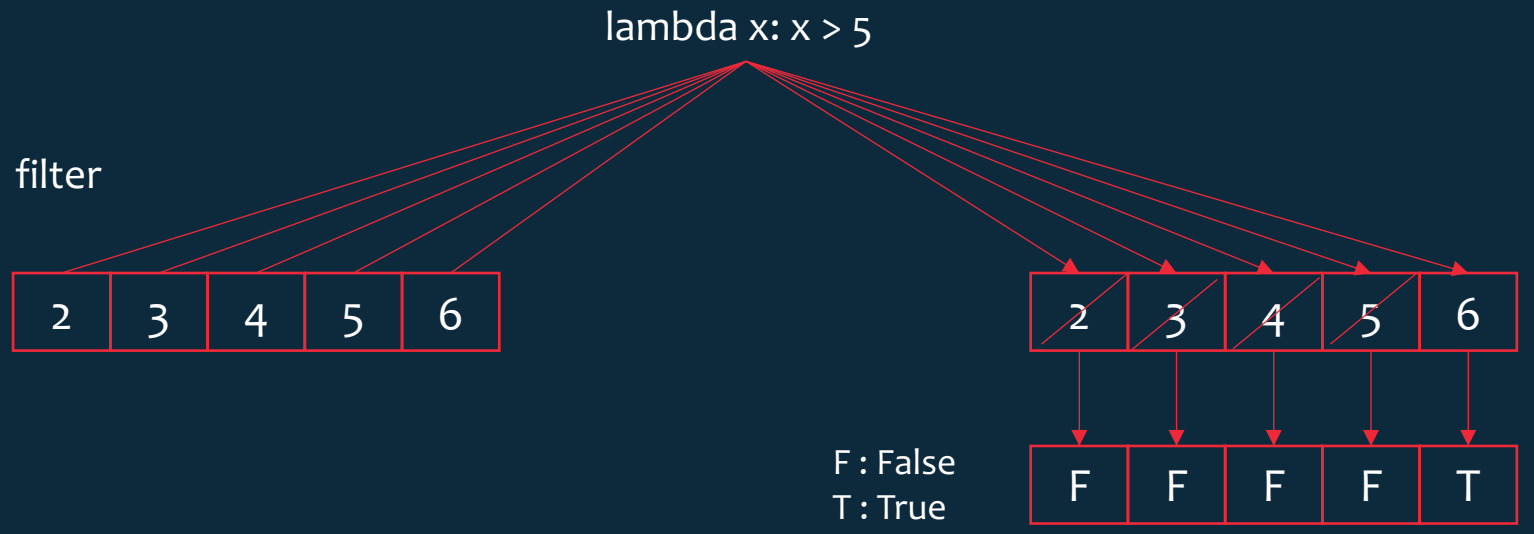
__closure__ : To find the memory address of the cell object

Anonymous/Lambda Functions

lambda	arg1,agr2,arg3…., argN	:	<expression>

```
lambda x,y : x + y
```

Anonymous/Lambda Functions



Anonymous/Lambda Functions

lambda x,y : x + y

reduce

x = 2, y = 3 => 2 + 3 => 5

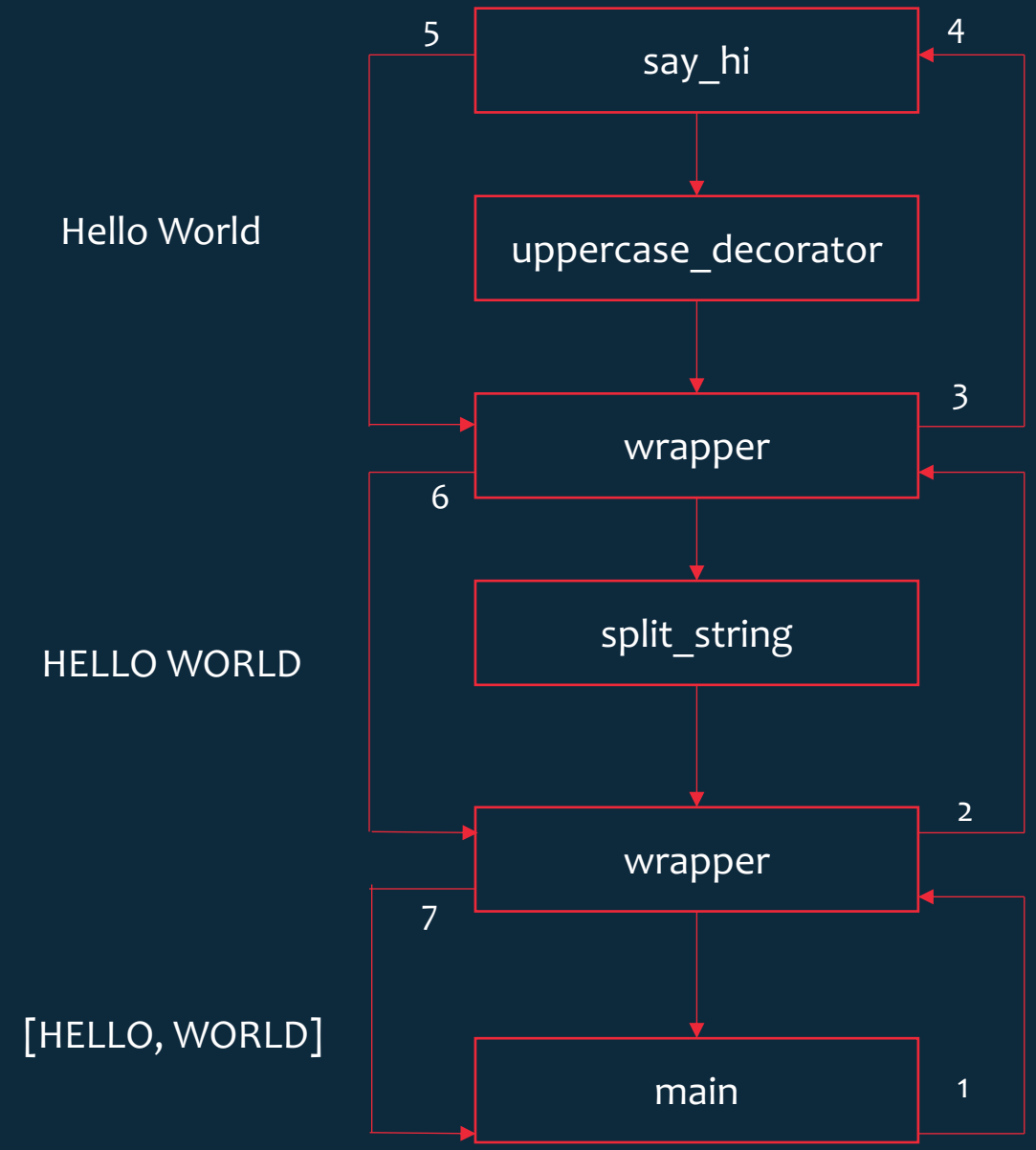
x = 5, y = 4 => 5 + 4 => 9

x = 9, y = 5 => 9 + 5 => 14

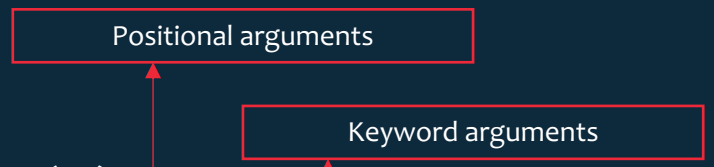
x = 14, y = 6 => 14 + 6 => 20

Decorators

A decorator is a function that takes another function as an argument and extends its behavior without changing the original function explicitly.



Decorators(Input arguments)




```
def currency(fn):
    def wrapper(*args, **kwargs):
            result = fn(*args, **kwargs)
        return f'${result}'
    return wrapper

@currency
def net_price(price, tax):
            return price * (1 + tax)
```

net_price(100, 0.05)

*args = (100, 0.05)
**kwargs = {}

In this scenario as we are using positional arguments the number of positional arguments should be equal to number of net_price function arguments

price = 100
tax = 0.05

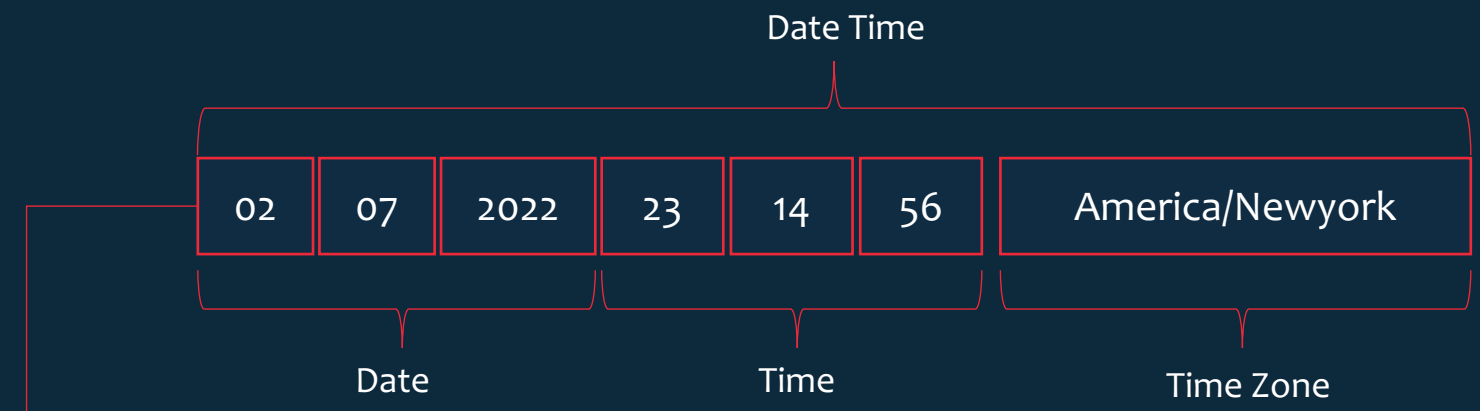
net_price(price=100, tax=0.05)

*args = ()
**kwargs = {price:100, tax: 0.05}

In this scenario new_price function was called using keyword arguments and python creates a dictionary from keyword arguments and the name of the argument that is used in the function (new_price) should exists as key along with its value in the dictionary

Say price, tax are used as argument name in the functions and price and tax are also passed as an keywords arguments to the net_price function call.

DateTime Module

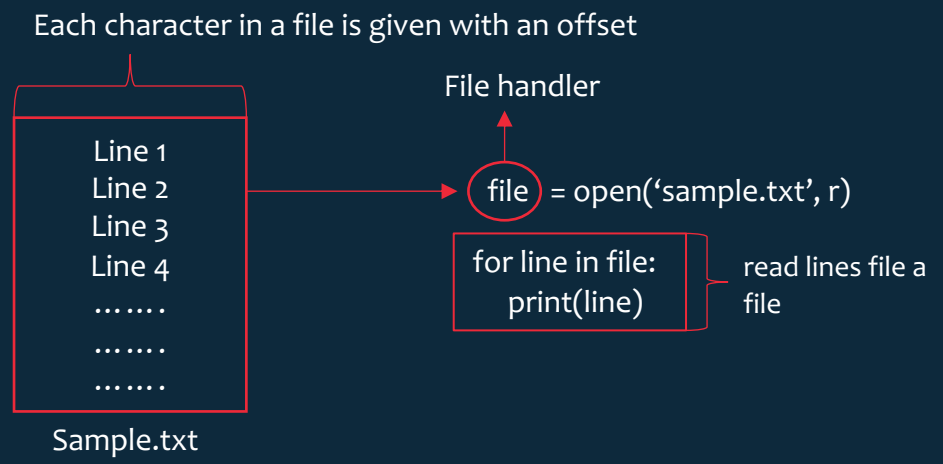


Epoch timestamp(Number representation) : 1656760496

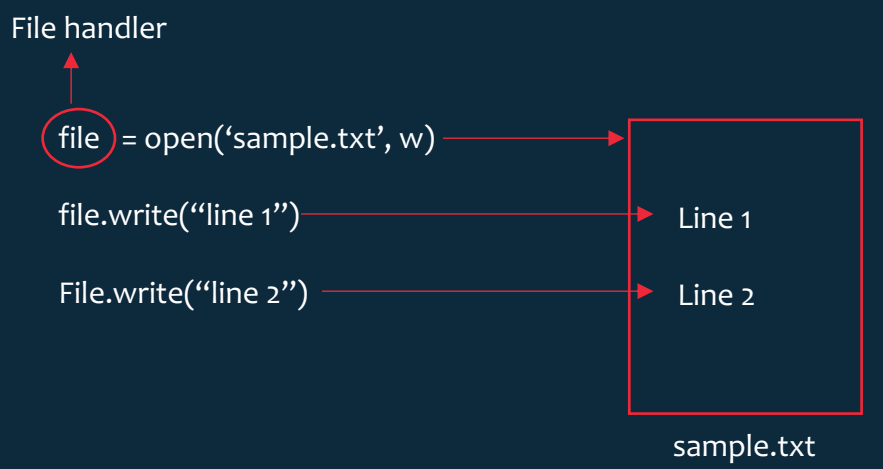
Working with files

filehandler = open(<file path>, mode)

Read from a file



Write into a file



Mode	r	r+	w	w+	a	a+
read	*	*		*		*
write		*	*	*	*	*
create			*	*	*	*
truncate			*	*		
position at start	*	*	*	*		
position at end					*	*

C:/Users/prudh/PycharmProjects/Sparklers_ToolKit/Python/4/json_files
→folder1
→folder2
→jsonfile1.json
C:/Users/prudh/PycharmProjects/Sparklers_ToolKit/Python/4/json_files/ folder1
→ C:/Users/prudh/PycharmProjects/Sparklers_ToolKit/Python/4/json_files/ folder1/subfolder
→jsonfile1.json

C:/Users/prudh/PycharmProjects/Sparklers_ToolKit/Python/4/json_files/ folder2
→jsonfile1.json
→jsonfile2.json

C:/Users/prudh/PycharmProjects/Sparklers_ToolKit/Python/4/json_files/ jsonfile1.json
C:/Users/prudh/PycharmProjects/Sparklers_ToolKit/Python/4/json_files/ folder1/jsonfile1.json
C:/Users/prudh/PycharmProjects/Sparklers_ToolKit/Python/4/json_files/ folder1/jsonfile2.json
C:/Users/prudh/PycharmProjects/Sparklers_ToolKit/Python/4/json_files/ folder1/subfolder/jsonfile1.json
C:/Users/prudh/PycharmProjects/Sparklers_ToolKit/Python/4/json_files/ folder2/ jsonfile1.json
C:/Users/prudh/PycharmProjects/Sparklers_ToolKit/Python/4/json_files/ folder2/ jsonfile2.json

Collections Module

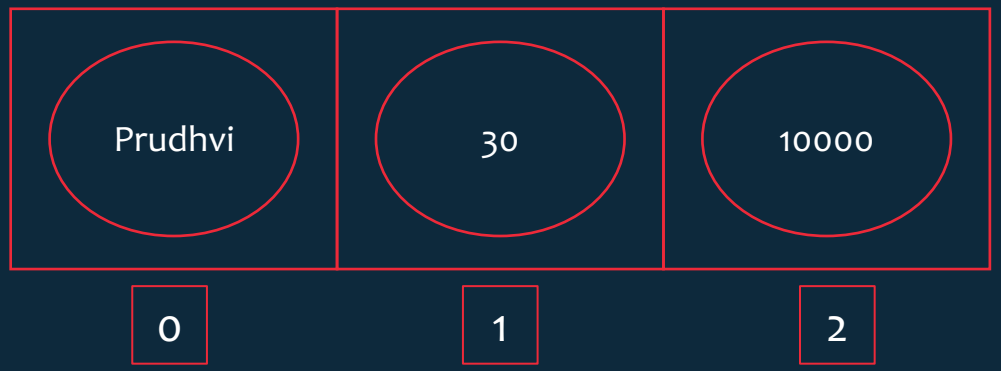
- Write readable and explicit code with namedtuple
- Build efficient queues and stacks with deque
- Count objects quickly with Counter
- Handle missing dictionary keys with defaultdict
- Guarantee the insertion order of keys with OrderedDict
- Manage multiple dictionaries as a single unit with ChainMap

Data type	Python version	Description
deque	2.4	A sequence-like collection that supports efficient addition and removal of items from either end of the sequence
defaultdict	2.5	A dictionary subclass for constructing default values for missing keys and automatically adding them to the dictionary
Namedtuple	2.6	A factory function for creating subclasses of tuple that provides named fields that allow accessing items by name while keeping the ability to access items by index
OrderedDict	2.7, 3.1	A dictionary subclass that keeps the key-value pairs ordered according to when the keys are inserted
Counter	2.7, 3.1	A dictionary subclass that supports convenient counting of unique items in a sequence or iterable
ChainMap	3.3	A dictionary-like class that allows treating a number of mappings as a single dictionary object

Namedtuple

Tuple

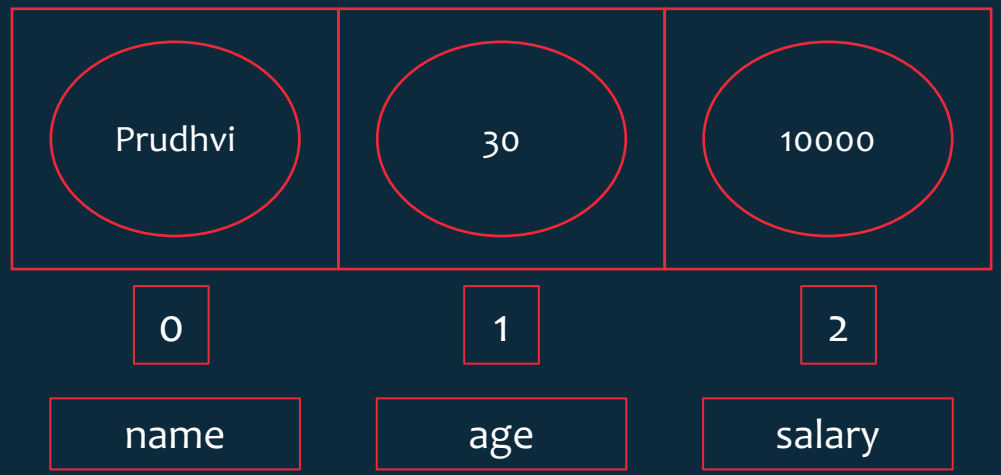
employee = (“Prudhvi”,30,10000)



Named Tuple

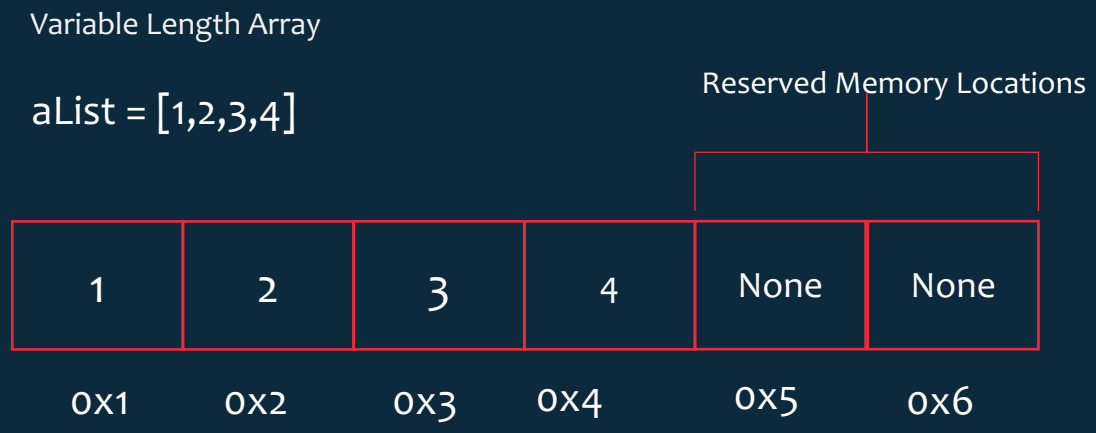
Person = namedtuple("Person", "name, age, salary")

employee = Person(“Prudhvi”, 30, 100000)



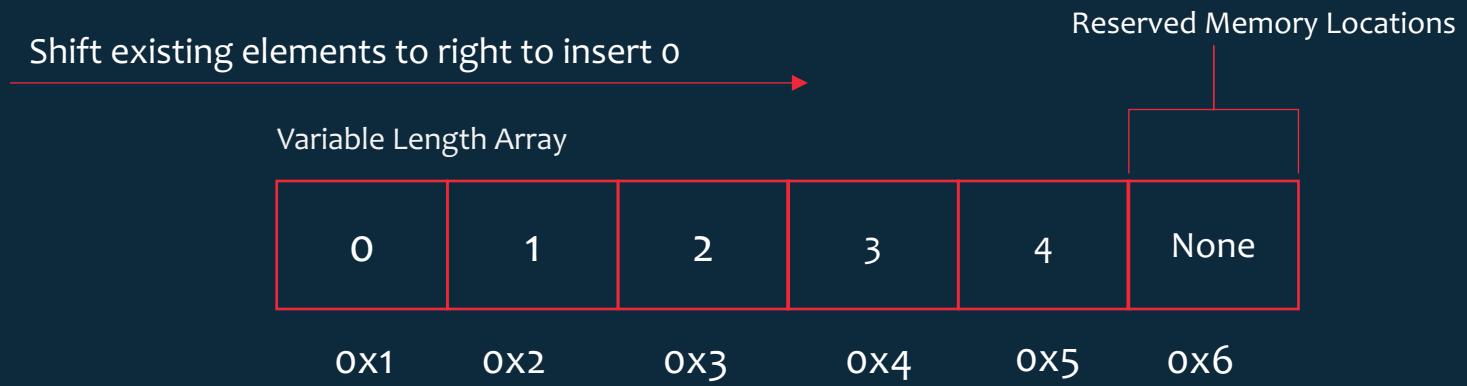
Deque

Understanding list behavior when we add and pop the elements at the head part



Appending and Pop elements at the head of the list is expensive operations. Complexity o(n) because Python has to move all the elements to right to insert new elements

aList = [0] + aList



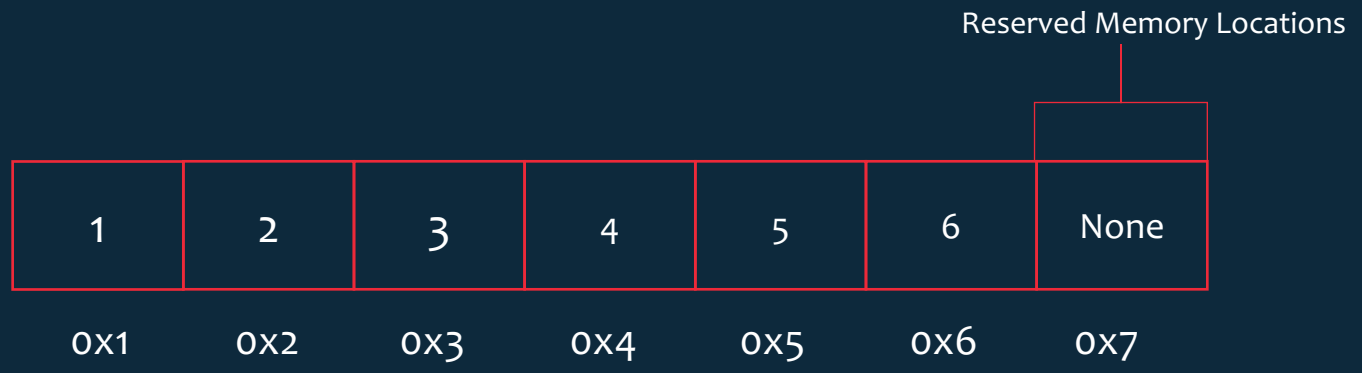
Deque

Understanding list behavior when we add and pop the elements at the tail part

aList = [1,2,3,4]

Appending and pop elements at the tail of the list are normally efficient . Complexity o(1)

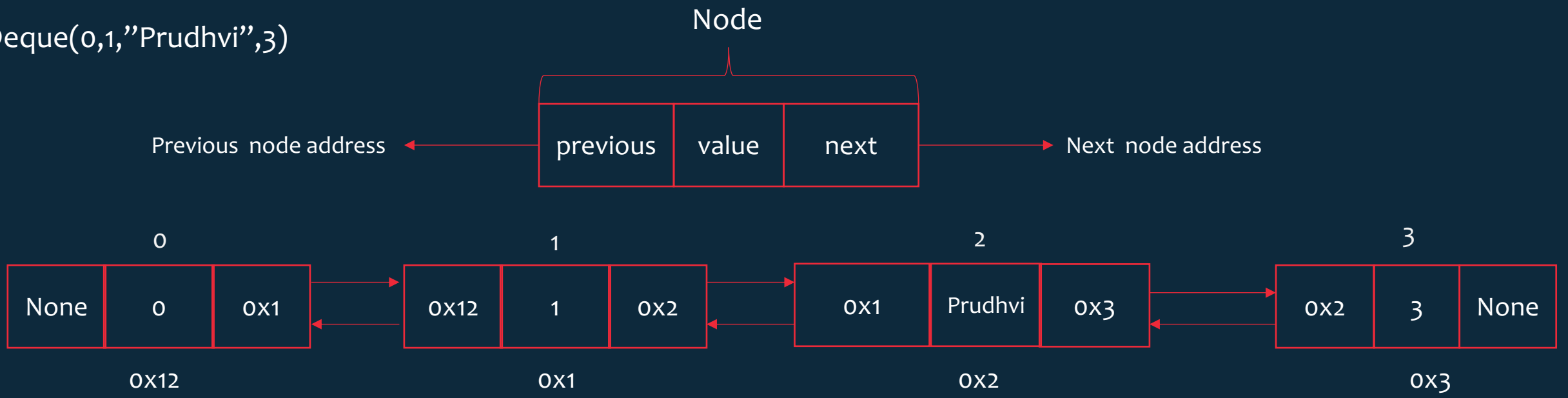
aList = aList.append(5)



Deque

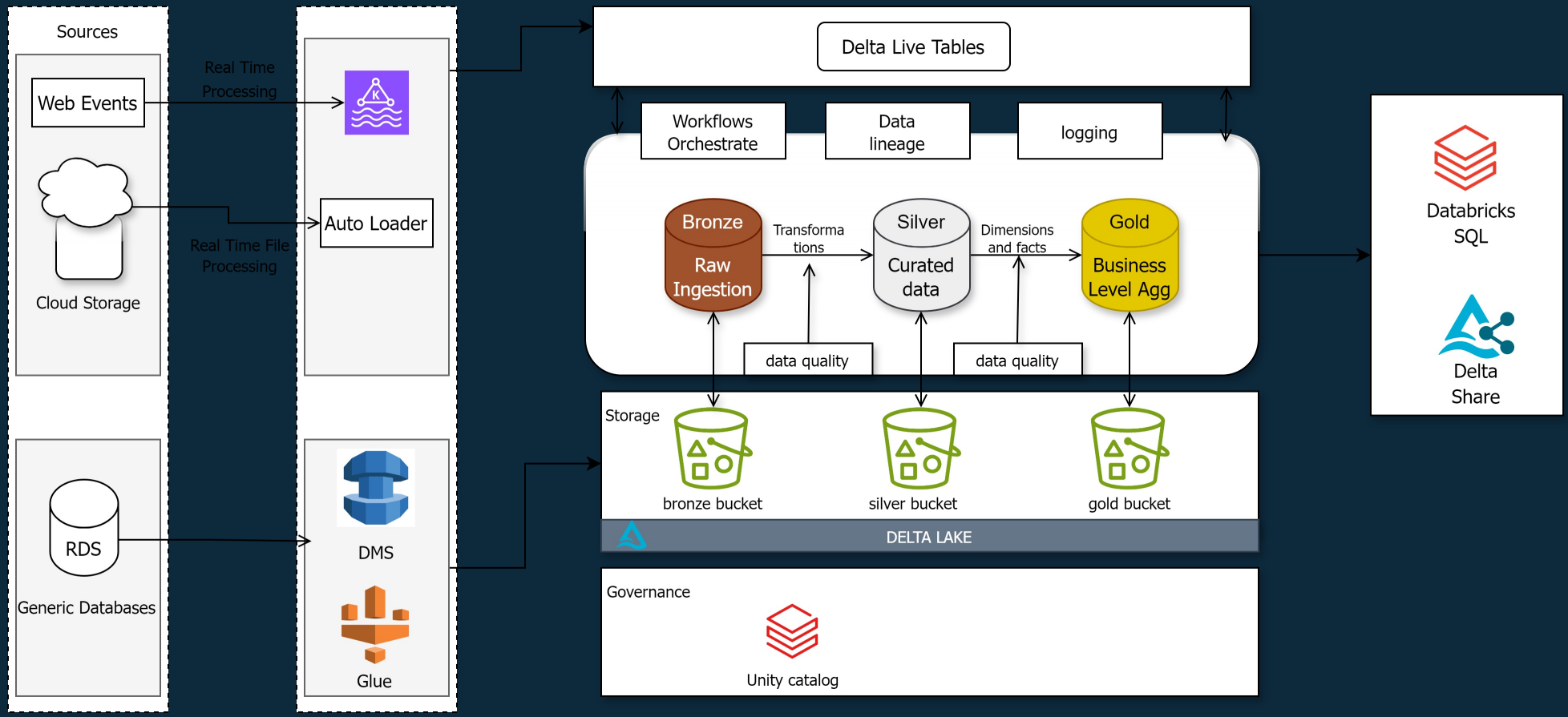
- Append and pop operations on both sides of a deque object are stable and equally efficient because deques are implemented as a doubly linked list.
- It manages items in a First-In/First-Out (FIFO) fashion.
- It works as a pipe, where you push in new items at one end of the pipe and pop old items out from the other end.
- Adding an item to the end of a queue is known as an enqueue operation.
- Removing an item from the front or beginning of a queue is called dequeue.

Deque(0,1,"Prudhvi",3)

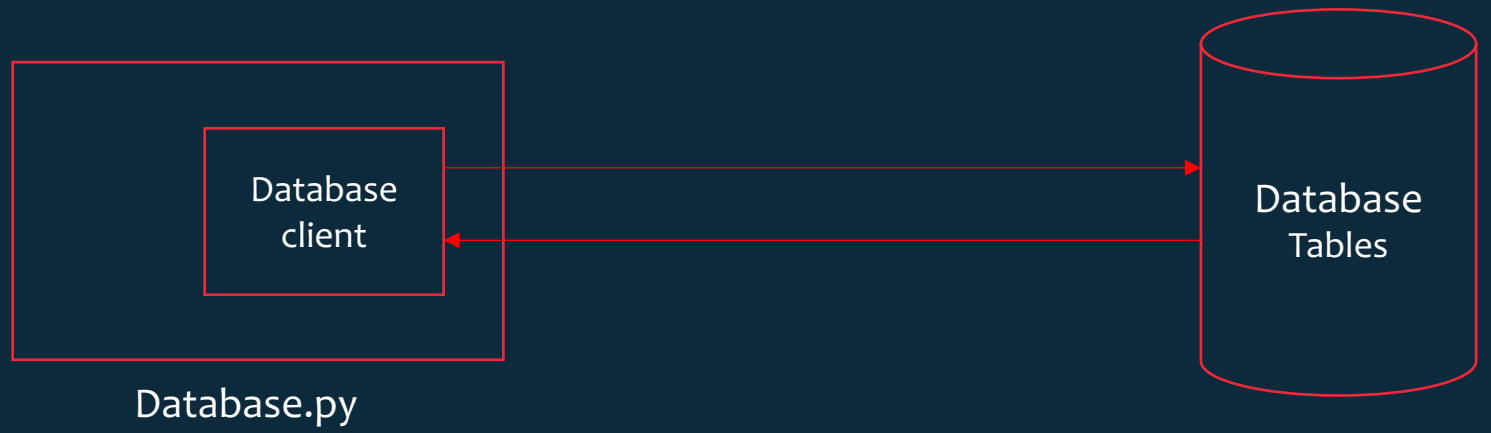


Comparing List and Deque

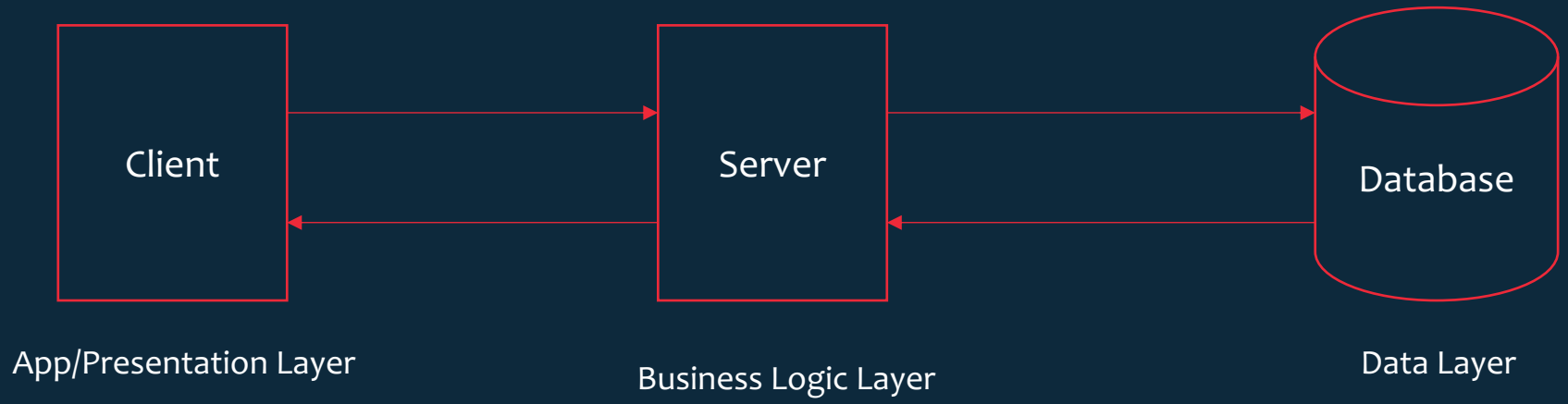
Operation	deque	list
Accessing arbitrary items through indexing	*O*(*n*)	*O*(1)
Popping and appending items on the left end	*O*(1)	*O*(*n*)
Popping and appending items on the right end	*O*(1)	*O*(1) + reallocation
Inserting and deleting items in the middle	*O*(*n*)	*O*(*n*)


Sources
Web Events
Cloud Storage
RDS
Generic Databases
Real Time Processing
Real Time File Processing
Auto Loader
DMS
Glue
Delta Live Tables
Workflows Orchestrate
Data lineage
logging
Bronze
Raw Ingestion
Transforma tions
Silver
Curated data
Dimensions and facts
Gold
Business Level Agg
data quality
data quality
Storage
bronze bucket
silver bucket
gold bucket
DELTA LAKE
Governance
Unity catalog
Databricks SQL
Delta Share

Connecting to Databases



Web Module



How will Client Talks to Server?

Web Module

- An API is a set of definitions and protocols for building and integrating application software. It's sometimes referred to as a contract between an information provider and an information user .

- The Hypertext Transfer Protocol (HTTP) is the foundation of the World Wide Web, and is used to load web pages using hypertext links. HTTP is an application layer protocol designed to transfer information between networked devices and runs on top of other layers of the network protocol stack. A typical flow over HTTP involves a client machine making a request to a server, which then sends a response message.

- An HTTP request is made by a client, to a named host, which is located on a server. The aim of the request is to access a resource on the server. To make the request, the client uses components of a URL (Uniform Resource Locator), which includes the information needed to server.

- A typical HTTP request contains:
  - HTTP version type
  - a URL
  - an HTTP method : An HTTP method, sometimes referred to as an HTTP verb, indicates the action that the HTTP request expects from the queried server
  - HTTP request headers : HTTP headers contain text information stored in key-value pairs, and they are included in every HTTP request (and response, more on that later)
  - Optional HTTP body: The body of a request is the part that contains the 'body' of information the request is transferring

- An HTTP response is what web clients (often browsers) receive from an Internet server in answer to an HTTP request.
  - A typical HTTP response contains:
    - an HTTP status code: HTTP status codes are 3-digit codes most often used to indicate whether an HTTP request has been successfully completed
    - HTTP response headers : Much like an HTTP request, an HTTP response comes with headers that convey important information
    - optional HTTP body: Successful HTTP responses to 'GET' requests generally have a body which contains the requested information.

Web Module : Http Methods

→ The primary or most commonly-used HTTP methods are POST, GET, PUT, PATCH, and DELETE. These methods correspond to create, read, update, and delete (or CRUD) operations, respectively

HTTP Method	CRUD operation	
GET	Read	The HTTP GET method is used to read (or retrieve) a representation of a resource.
POST	Create	The POST method is most often utilized to create new resources.
PUT	Update/Replace	PUT is used to modify resources. The PUT request only needs to contain the changes to the resource, not the complete resource.
DELETE	Delete	DELETE is quite easy to understand. It is used to delete a resource identified by filters or ID.

Web Module : Http Status Codes

In case of a successful request, a response status code will be one of the following:

- 200 OK—In the response to successful GET, PATCH or DELETE.
- 201 Created—In the response to POST that results in creation. When this status is received, the request body contains the description of the newly created entity in the JSON format (similar to the regular GET request).
- 204 No Content—In the response to a successful request that won't return a body (like a DELETE request)

In case of an error, a response status code indicates the type of an error that has occurred. The most common codes are the following:

- 400 Bad Request—The request is malformed, such as if the body of the request contains misformatted JSON.
- 401 Unauthorized—No or invalid authentication details are provided. This code can be used to trigger an authentication pop-up if the API is used from a browser.
- 403 Forbidden—Authentication succeeded but authenticated user does not have access to the resource.
- 404 Not Found—A non-existent resource is requested.
- 500 Internal Server Error—The server encountered an unexpected condition which prevented it from fulfilling the request.

Web Module : Http Header

An HTTP header is a field of an HTTP request or response that passes additional context and metadata about the request or response.

- Request header : Headers containing more information about the resource to be fetched or about the client itself.
- Response header : Headers with additional information about the response, like its location or about the server itself (name, version, …).
- Representation header : metadata about the resource in the message body (e.g. encoding, media type, etc.).
- Fetch metadata request header : Headers with metadata about the resource in the message body (e.g. encoding, media type, etc.).

To understand about different types of headers please follow the below link.

- https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

Web Module : Rest API

REST(Representational State Transfer)

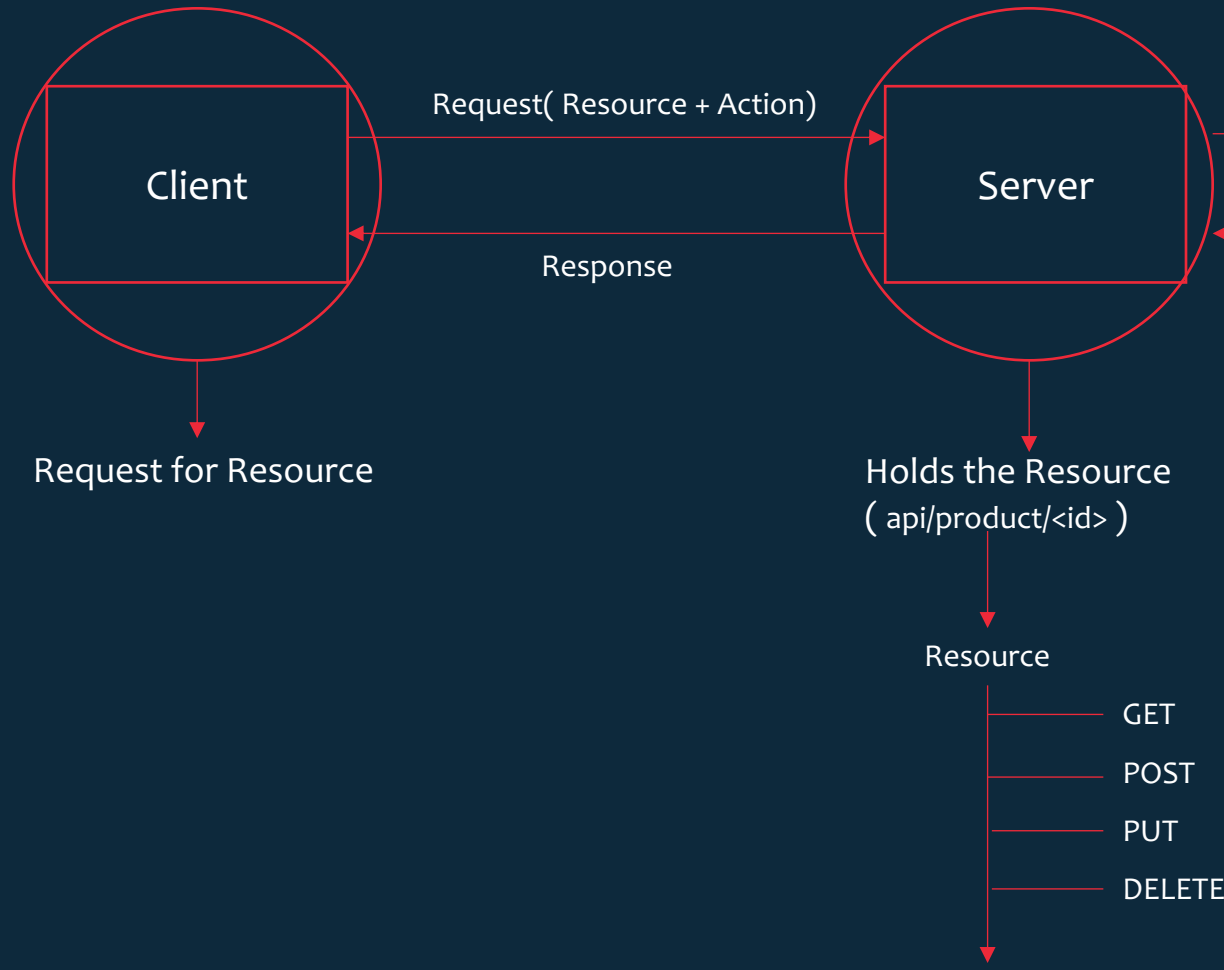
→ Resource(Noun and Action)

→ Representational State Transfer with an example

→Architectural Constraints

- Client Server
- Statelessness
- Cache
- Uniform Interface
- Layered System
- Code on Demand(Optional)

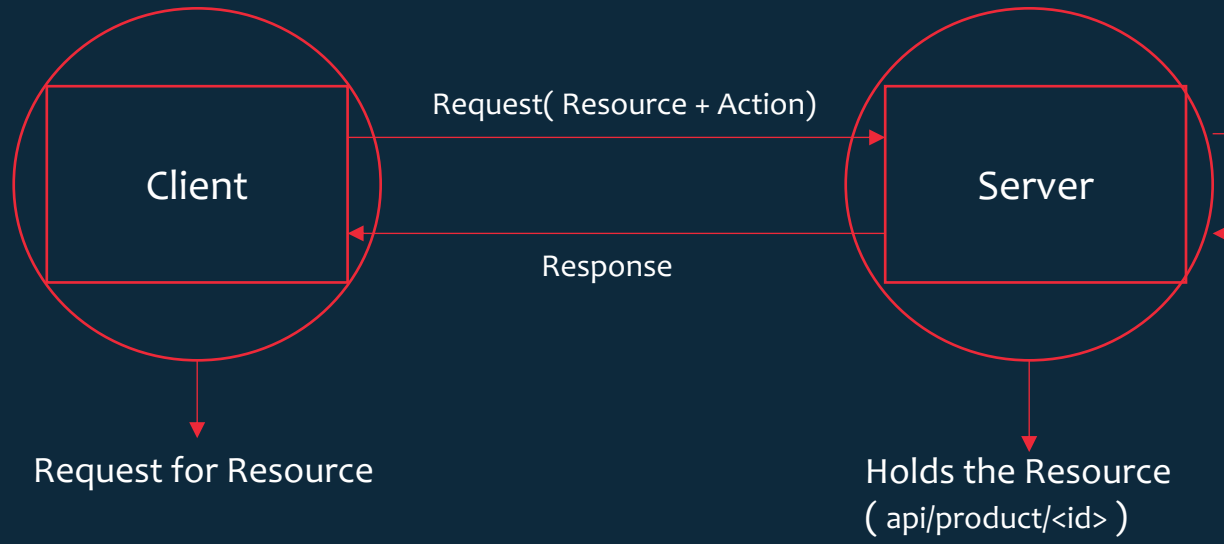
Web Module : Rest API : Resource(Noun and Action)



Resource (Product)

ID	Name	Rate
1	Shoe	100
2	Tablet	20000
3	Laptop	30000
4	Phone	15000
5	Mouse	150

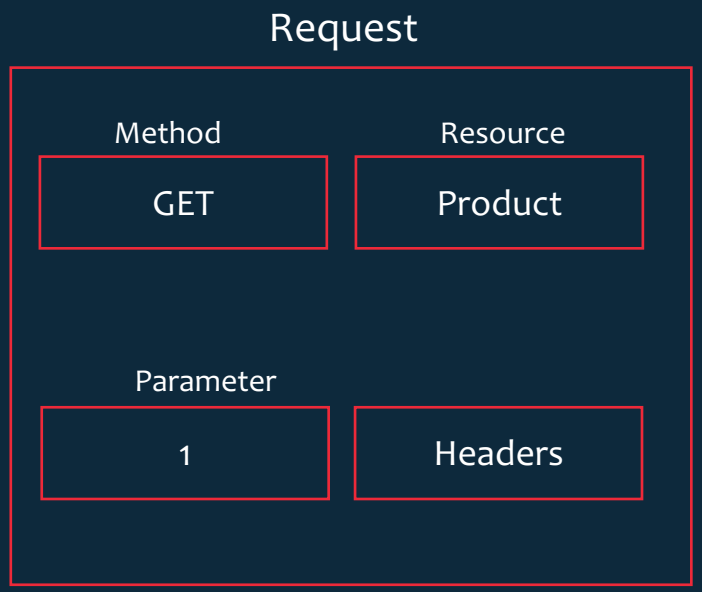
Web Module : Rest API : Resource(GET Method)

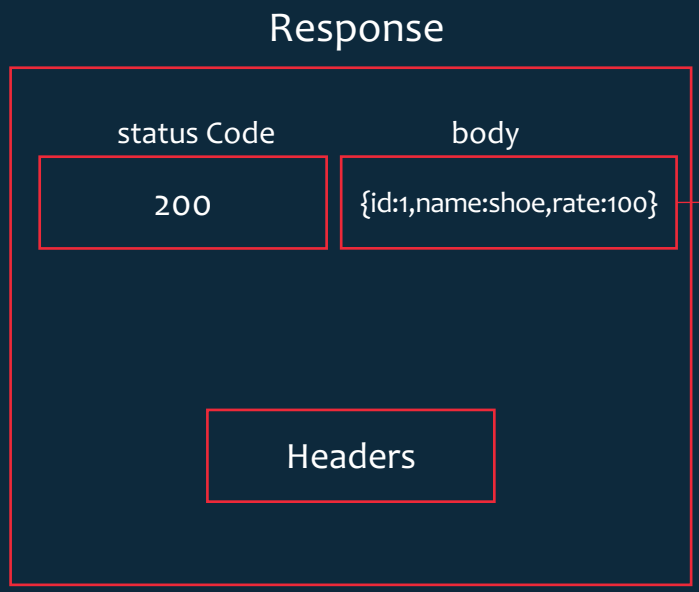


Resource (Product)

ID	Name	Rate
1	Shoe	100
2	Tablet	20000
3	Laptop	30000
4	Phone	15000
5	Mouse	150

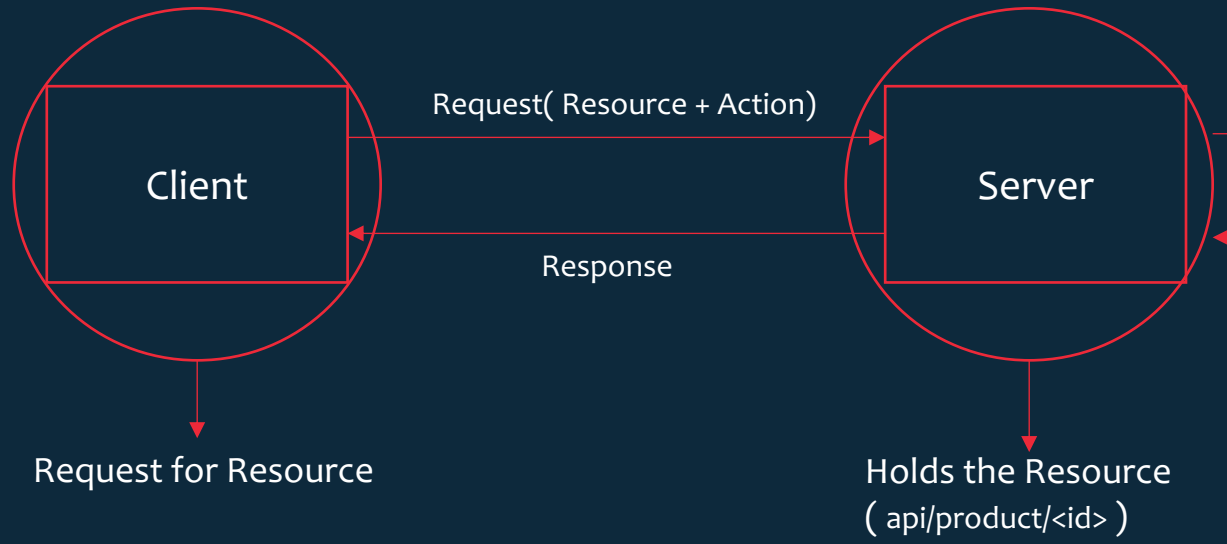
http://localhost:5000/api/product/1





Representation of current state of the project In JSON

Web Module : Rest API : Resource(POST Method)



Resource (Product)

ID	Name	Rate
1	Shoe	100
2	Tablet	20000
3	Laptop	30000
4	Phone	15000
5	Mouse	150
6	Shoe	100

http://localhost:5000/api/product

Request

Method: POST

Resource: Product

body: {id:6,name:shoe,rate:300}

Headers

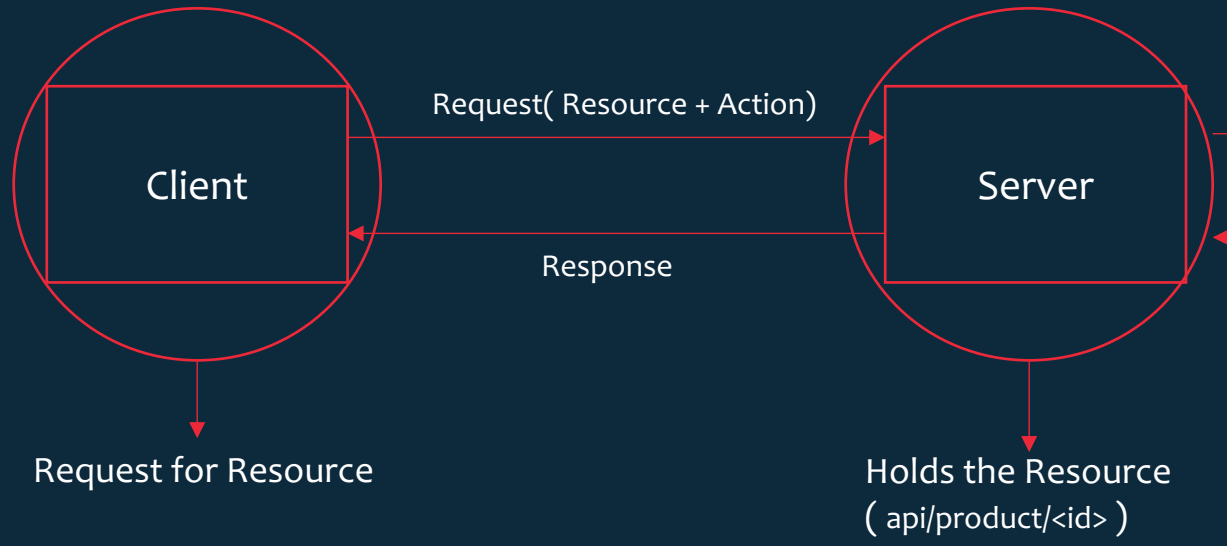
Response

status Code: 200

body: {status : ”inserted”}

Headers

Web Module : Rest API : Resource(PUT Method)



Resource (Product)

ID	Name	Rate
1	Shoe	100
2	Tablet	20000
3	Laptop	30000
4	Phone	15000
5	Mouse	150
6	blanket	100

http://localhost:5000/api/product/6

Request

Method: PUT

Resource: Product

body: {name: "blanket"}

Headers

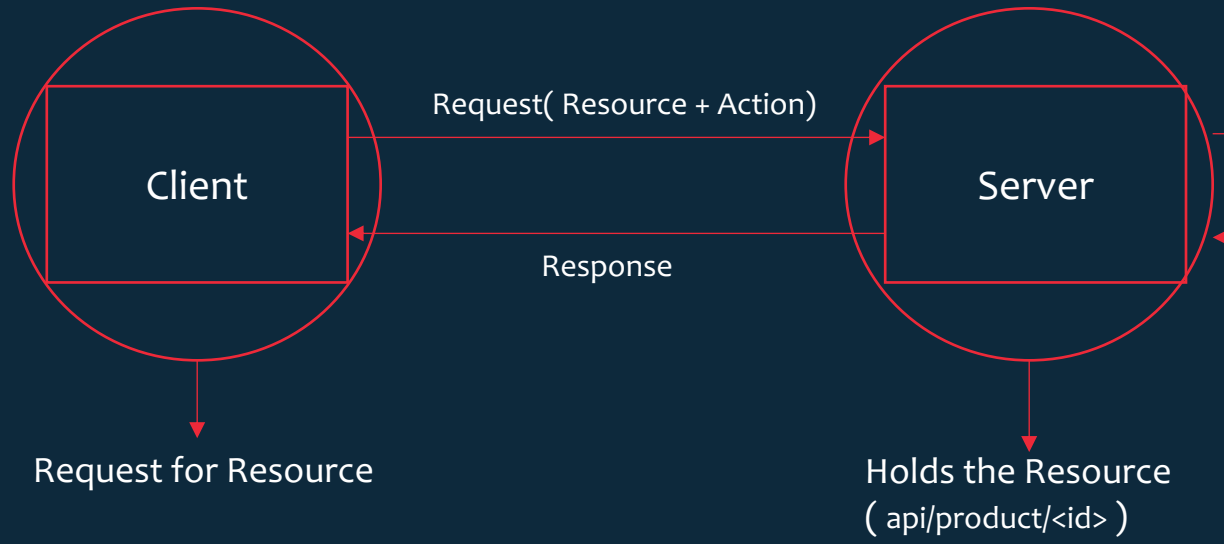
Response

status Code: 200

body: {status : "updated"}

Headers

Web Module : Rest API : Resource(DELETE Method)



Resource (Product)

ID	Name	Rate
1	Shoe	100
2	Tablet	20000
3	Laptop	30000
4	Phone	15000
5	Mouse	150
~~6~~	~~blanket~~	~~100~~

http://localhost:5000/api/product/6

Request

Method: DELETE

Resource: Product

parameter: 6

Headers

Response

status Code: 200

body: {status : ”deleted”}

Headers

Web Module : Rest API : Constraint : Client and Server

- By Now we understood that REST works only with Client and Server Architecture
- Advantage with this type of architecture is they are the two different entities and they can evolve independently
- Client doesn’t need to know what the server does and vice versa

Web Module : Rest API : Constraint : Stateless

- The server does not store any state about the client session on the server-side. This restriction is called Statelessness.
  or
- Statelessness means that every HTTP request happens in complete isolation. When the client makes an HTTP request, it includes all information necessary for the server to fulfill the request.

- Each request from the client to the server must contain all of the necessary information to understand the request. The server cannot take advantage of any stored context on the server.
  or
- The server never relies on information from previous requests from the client. If any such information is important then the client will send that as part of the current request.

- The application's session state is therefore kept entirely on the client. The client is responsible for storing and handling the session related information on its own side.
  or
- This also means that the client is responsible for sending any state information to the server whenever it is needed. There should not be any session affinity or sticky session between the client and the server.

- For becoming stateless, do not store even authentication/authorization details of the client. Provide authentication credentials with each request.

Web Module : Rest API : Constraint : Caching

- Caching is the ability to store copies of frequently accessed data in several places along the request-response path.
- When a consumer requests a resource representation, the request goes through a cache or a series of caches (local cache, proxy cache, or reverse proxy) toward the service hosting the resource.
- If any of the caches along the request path has a fresh copy of the requested representation, it uses that copy to satisfy the request. If none of the caches can satisfy the request, the request travels to the service.
- By using HTTP headers, an origin server indicates whether a response can be cached and, if so, by whom, and for how long.
- Optimizing the network using caching improves the overall quality-of-service in the following ways:
  - Reduce bandwidth
  - Reduce latency
  - Reduce load on servers
  - Hide network failures
- GET requests should be cachable by default – until a special condition arises. Usually, browsers treat all GET requests as cacheable.
- POST requests are not cacheable by default but can be made cacheable if either an Expires header or a Cache-Control header with a directive, to explicitly allows caching, is added to the response.
- Responses to PUT and DELETE requests are not cacheable at all.

Web Module : Rest API : Constraint : Uniform Interface

- Uniform Interface

  The uniform interface constraint defines the interface between clients and servers. It simplifies and decouples the architecture, which enables each part to evolve independently. The four guiding principles of the uniform interface are:

  - Resource-Based

    Individual resources are identified in requests using URL as resource identifiers. The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server does not send its database, but rather, some HTML, XML or JSON that represents some database records expressed, for instance, in Finnish and encoded in UTF-8, depending on the details of the request and the server implementation.

  - Manipulation of Resources Through Representations

    When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource on the server, provided it has permission to do so.

  - Self-descriptive Messages

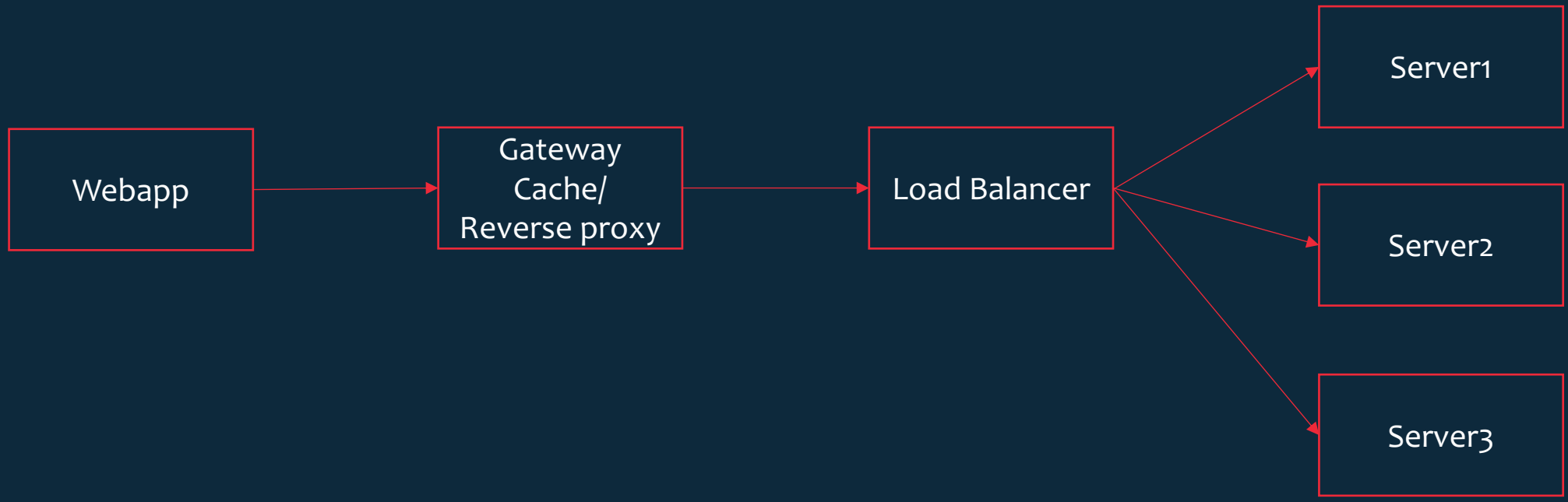
    Each message includes enough information to describe how to process the message. For example, which parser to invoke may be specified by an Internet media type (previously known as a MIME type). Responses also explicitly indicate their cache-ability.

  - Hypermedia as the Engine of Application State (HATEOAS)

    Clients deliver state via body contents, query-string parameters, request headers and the requested URI (the resource name). Services deliver state to clients via body content, response codes, and response headers. This is technically referred-to as hypermedia (or hyperlinks within hypertext).

Web Module : Rest API : Constraint : Layered System

Between the client who requests a representation of a resource's state, and the server who sends the response back, there might be a number of servers in the middle. These servers might provide a security layer, a caching layer, a load-balancing layer, or other functionality. Those layers should not affect the request or the response. The client is agnostic as to how many layers, if any, there are between the client and the actual server responding to the request. Layering can also be used for caching.

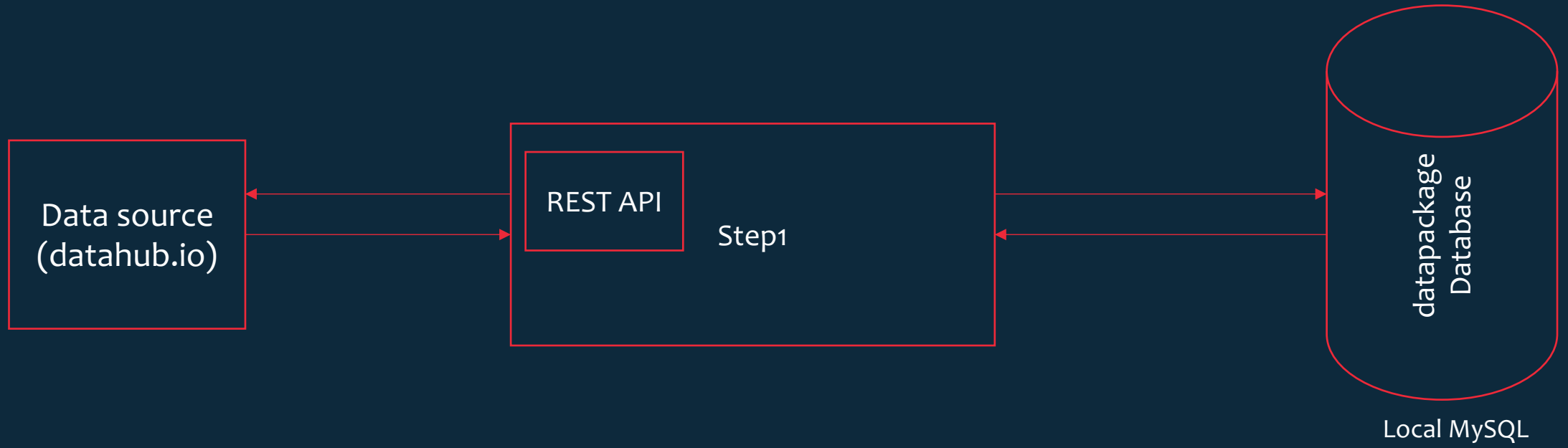


Web Module : Rest API : Constraint : Code on Demand

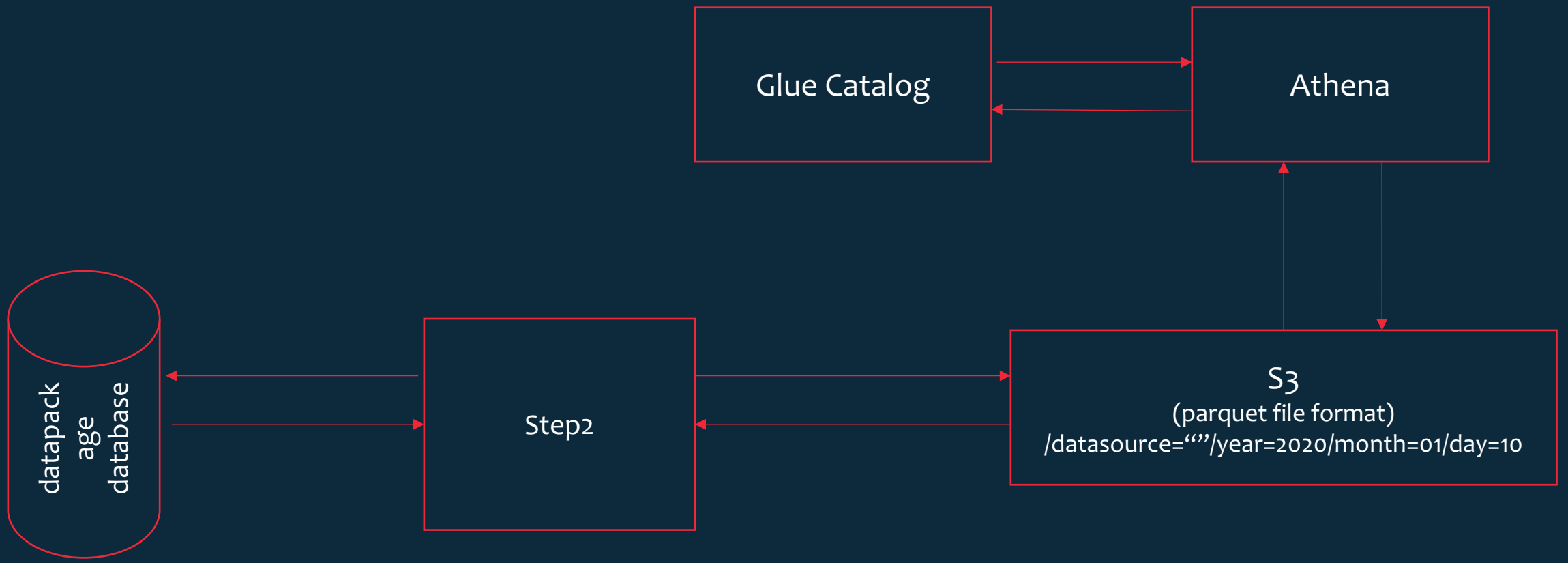
This constraint is optional. Most of the time, you will be sending the static representations of resources in the form of XML or JSON. But when you need to, you are free to return executable code to support a part of your application, e.g., clients may call your API to get a UI widget rendering code. It is permitted.

Flask Module

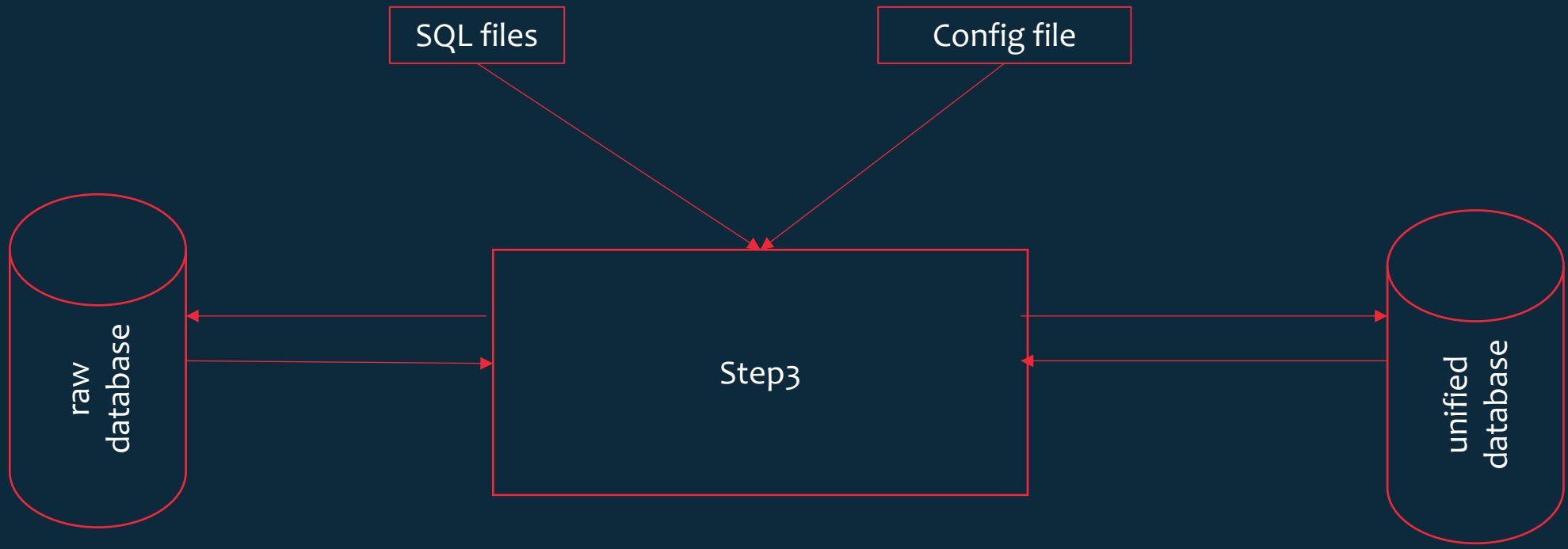
Project : Step1



Project : Step2



Project : Step3

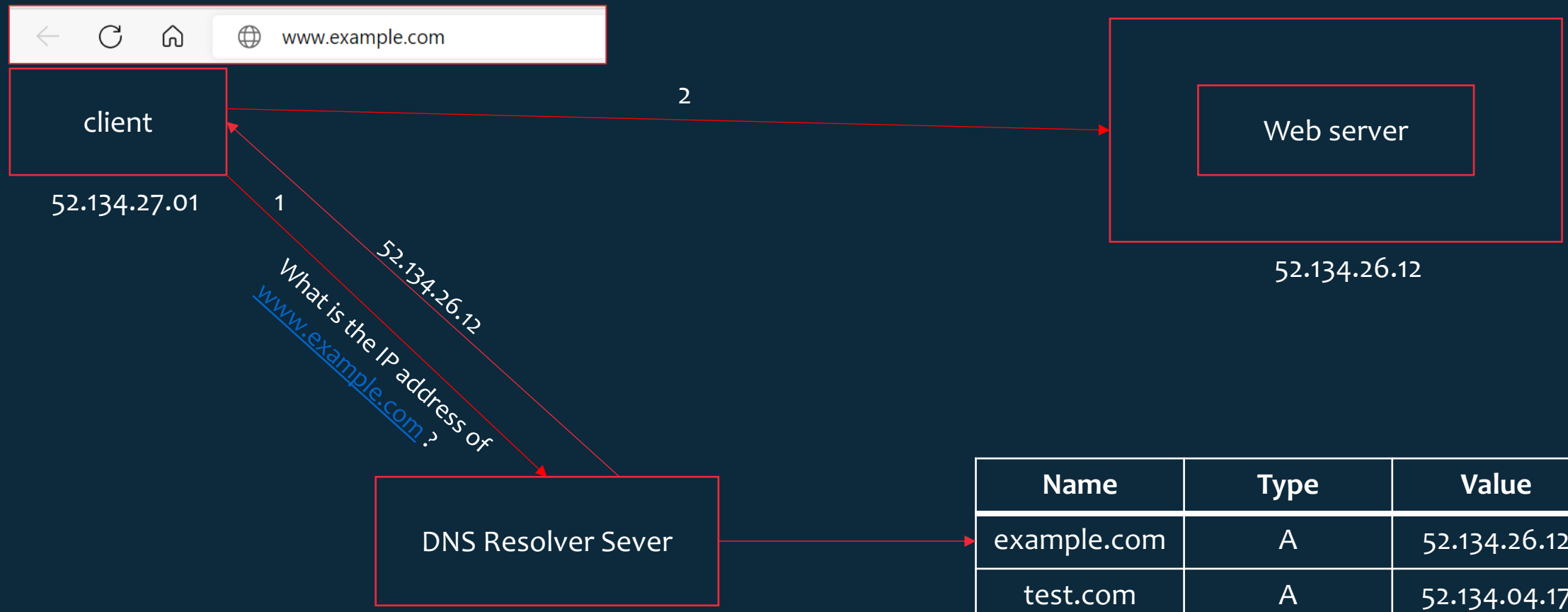



Mysql
Bronze
(Raw)
Silver Layer
(Refined Data)
Gold Layer
(Aggregated Data)
BI Tool
UI

AWS

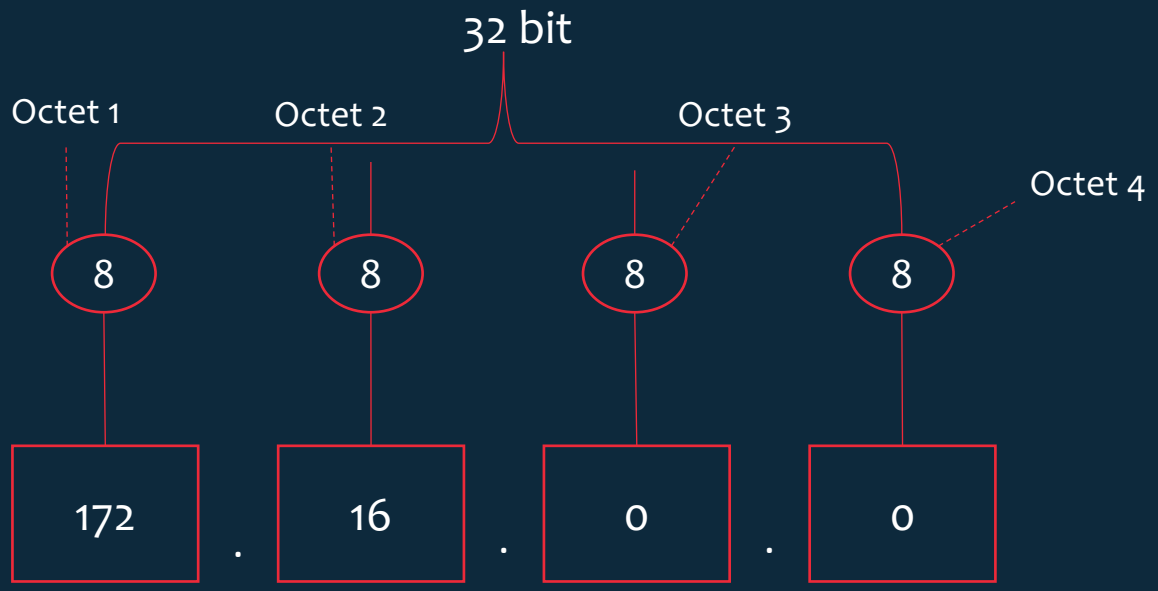
What is an ip4 address ?

Ip address are the address computer use to communicate



Name	Type	Value
example.com	A	52.134.26.12
test.com	A	52.134.04.17

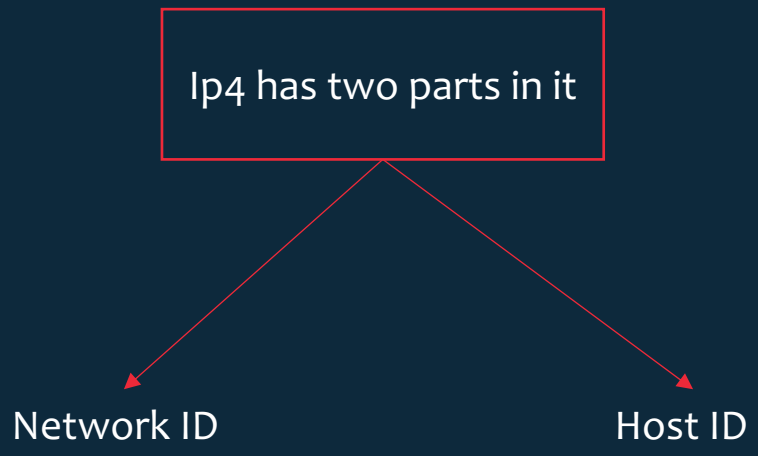
Ipv4



→ 32 bit value
Each device must have assigned with ip4 for communication in the internet

Limitation
2^32 total address possibilities = 4,294,967,296(4.294 billion)

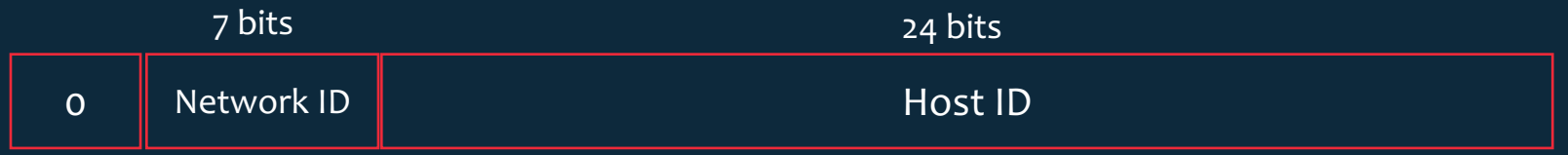
Ip4 : Network and Hosts



Classful Networking(Public IP (Comes at a Cost))

Internet Assigned Numbers Authority(IANA) Divided network into three class

Class A : /8 block : List of assigned /8 IPv4 address blocks - Wikipedia

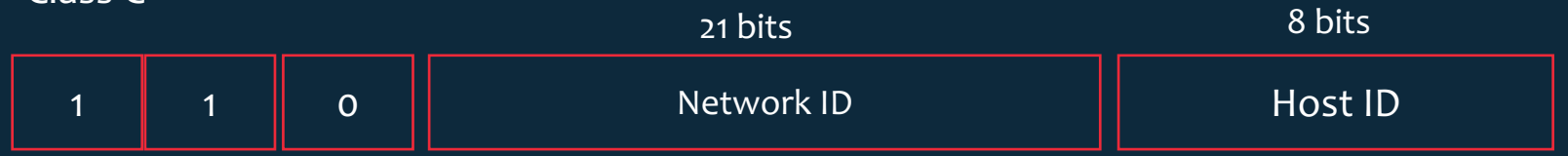


2^24 = 16,777,214 Ips , 127 Networks, 1.0.0.0 to 127.255.255.255

Class B

2^16 = 65534 Ips, 16328 Networks, 128.0.0.0 to 172.15.255.255/ 172.32.0.0 to 191.255.255.255

Class C



2^16 = 254 Ips, 2097150 Networks, 192.0.0.0 to 192.167.255.255/ 192.169.0.0 to 223.255.255.255

No More Ipv4

- 2^32 number of ip address are only available
- Population is very large and IOT is taking over so address are exhausting.
- Is Block /8 concern ?

Companies and organizations with IPv4 /8 blocks from IANA

Owner	Blocks~IP addresses	
US Military (Department of Defense etc.)	12	201 million
Level 3 Communications, Inc.	2	33 million
Hewlett-Packard	2	33 million
AT&T Bell Laboratories (Alcatel-Lucent)	1	16 million
AT&T Global Network Services	1	16 million
Bell-Northern Research (Nortel Networks)	1	16 million
Amateur Radio Digital Communications	1	16 million
Apple Computer Inc.	1	16 million
Cap Debis CCS (Mercedes-Benz)	1	16 million
Computer Sciences Corporation	1	16 million
Deparment of Social Security of UK	1	16 million
E.I. duPont de Nemours and Co., Inc.	1	16 million
Eli Lily and Company	1	16 million
Ford Motor Company	1	16 million
General Electric Company	1	16 million
Halliburton Company	1	16 million
IBM	1	16 million
Interop Show Network	1	16 million
Merck and Co., Inc.	1	16 million
MERIT Computer Network	1	16 million
Massachusetts Institute of Technology	1	16 million
Performance Systems International (Cogent)	1	16 million
Prudential Equity Group, LLC	1	16 million
Société Internationale De Telecommunications Aeronautiques	1	16 million
U.S. Postal Service	1	16 million
UK Ministry of Defence	1	16 million
Xerox Corporation	1	16 million
	40	671 million

Cost of /8 between $200 M to $1.5 B

Solution1:

Introducing Ivp6

- -- 3.4 * 10 ^7 ips

Limitations:

- Most of software's and hardware's are designed to use ipv4

Solution 2:

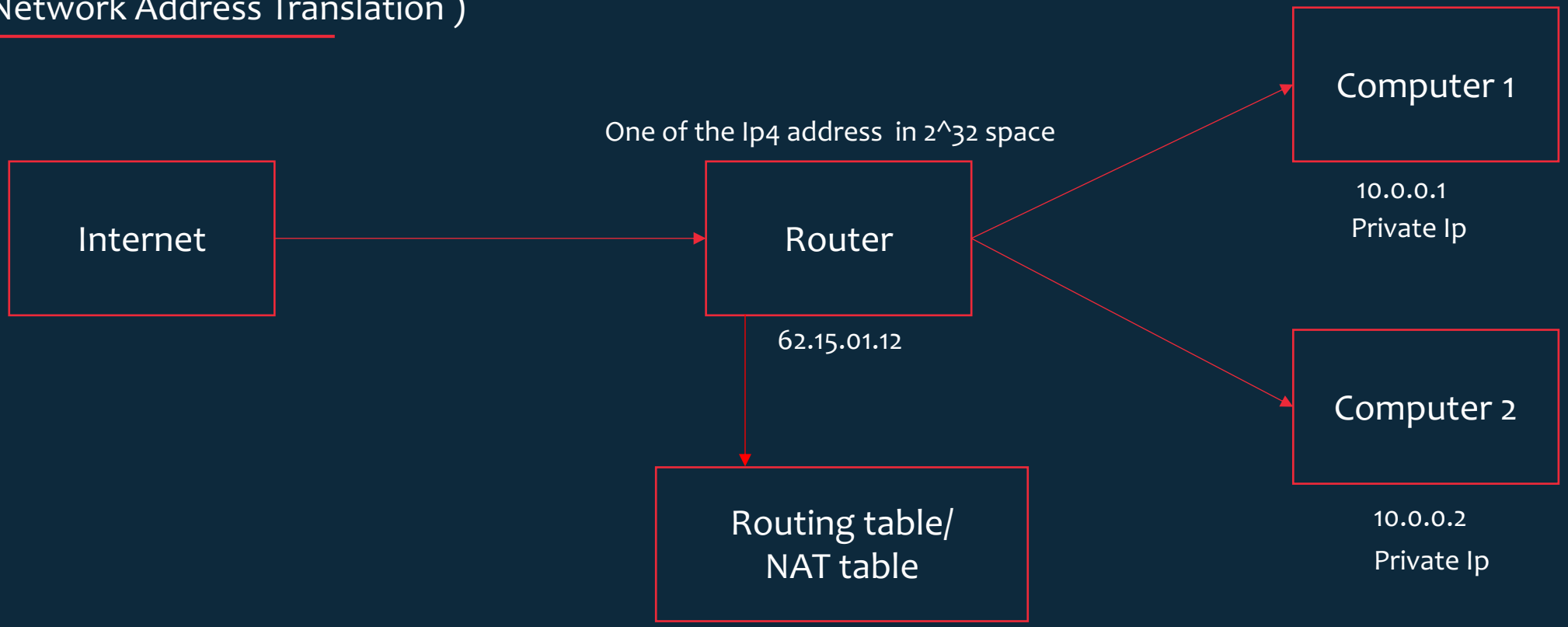
NAT(Network Address Translation)

CIDR (Class Inter-Domain Routing)

NAT(Network Address Translation) & CIDR (Class Inter-Domain Routing)

- Packet Re-routing to Private IP’s
- Slows down the Ip4 exhaustion
- But still ip4 IP’s are 2^32

NAT(Network Address Translation)



Imagine there is an router which is connect with different devices at home it could be anything which is assigned with private ip they are not assigned with ip4 address and Routers acts like NAT gateway which means that Router is connected to the internet and devices are connected to Router any communication has from Devices to internet has to happen through Router and Vice versa.

Now Image computer1 wants to open a yahoo.com then request will go to the router then what router does is it intercepts the meta data with its ip and send it out to the network/internet and it send to yahoo server and yahoo server sends the response

The response will be send back to the router by looking at the network translation table / Routing table and then it will be routed to the computer 1

AWS VPC

Region

VPC(Virtual Private cloud is a logically isolated portion of AWS cloud with in a region)

Availability Zone

Public Subnet

EMR

EC2

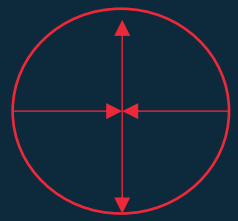
Availability Zone

Private Subnet

EMR

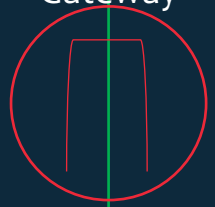
EC2

VPC Router takes care of routing with in the VPC and outside the VPC. Physically this doesn't exist. What exist is routing table.



Router

Internet Gateway



It is used to connect to internet. One for VPC

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	Igw-id

Route Table

AWS VPC

Now to Create an VPC in AWS you need to define the CIRD block?

How does it look it?

192.168.0.0/16

What does it represents?
It has two parts in 10.0.0.0 -> ip4 private starting address

/16 -> CIDR which is a representation of Subnet mask.

If you convert the decimal into binary it look like below

11111111.11111111.00000000.00000000

255 . 255 . 0 . 0

How many IP address you can get for CIDR block 16?

32 – 16 = 16 => 2^ 16 = 65536

How many IP's for CIDR /8 ?
32 – 8 = => 2^ 24 = 16777216
11111111.00000000.00000000.00000000

255 . 0 . 0 . 0

How many IP's for CIDR /12 ?
32 – 12 = => 2^ 20 = 1048576

11111111. 11110000.00000000.00000000

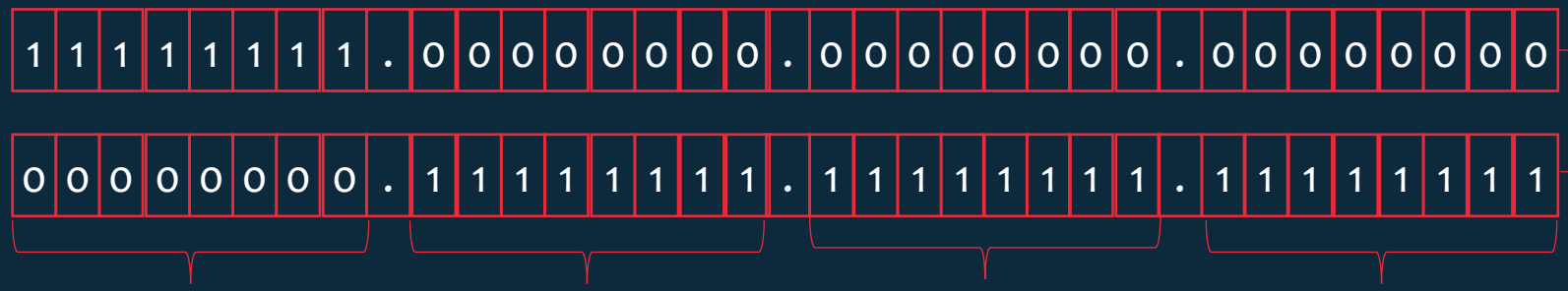
255 . 240 . 0 . 0

1	1	1	1	0	0	0	0

128	64	32	16	8	4	2	1

AWS VPC : Ending address from CIDR

What is an ending address of below IP with CIDR 8?
10.0.0.0/8

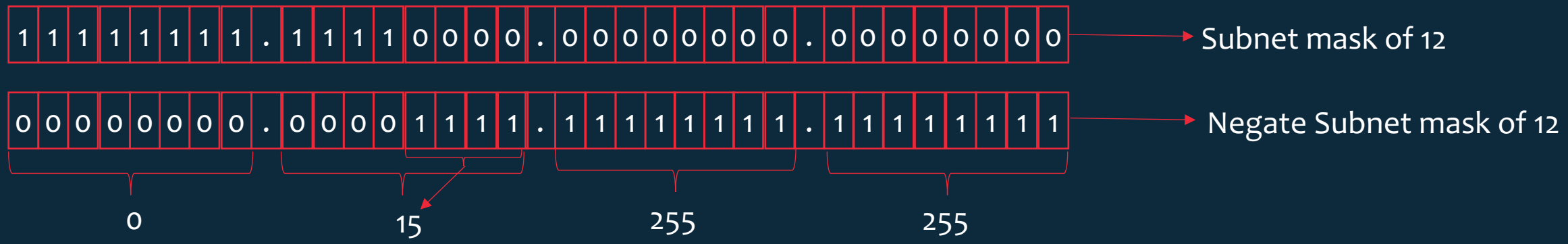


10 0 0 0 → Start address

10 255 255 255 → End address

AWS VPC : Ending address from CIDR

What is an ending address of below IP with CIDR 12?
172.16.0.0/12

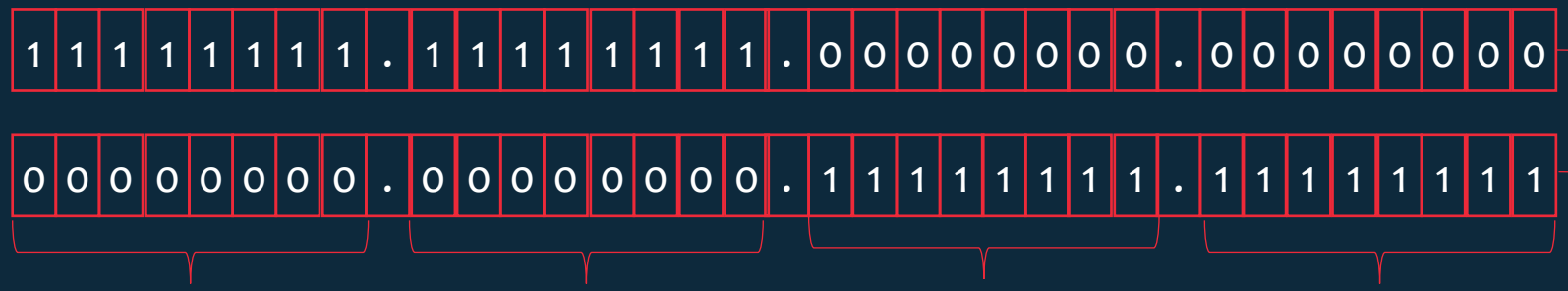


172 16 0 0 → Start address

172 31 255 255 → End address

AWS VPC : Ending address from CIDR

What is an ending address of below IP with CIDR 16?
192.168.0.0/16



192 168 0 0 → Start address

192 168 255 255 → End address

AWS VPC : Private address class

Class	Ip range	CIDR	Number of IPS	Purpose
A	10.0.0.0 – 10.255.255.255	10.0.0.0/8	16777216	Big Organizations
B	172.16.0.0 – 172.31.255.255	172.16.0.0/12	1048576	Default Range when you create AWS account
C	192.168.0.0 – 192.168.255.255	192.168.0.0/16	65536	Smaller Organizations

K = 1,024 • M = 1,048,576

IP Addresses	Bits	Prefix	Subnet Mask
1	0	/32	255.255.255.255
2	1	/31	255.255.255.254
4	2	/30	255.255.255.252
8	3	/29	255.255.255.248
16	4	/28	255.255.255.240
32	5	/27	255.255.255.224
64	6	/26	255.255.255.192
128	7	/25	255.255.255.128
256	8	/24	255.255.255.0
512	9	/23	255.255.254.0
1 K	10	/22	255.255.252.0
2 K	11	/21	255.255.248.0
4 K	12	/20	255.255.240.0
8 K	13	/19	255.255.224.0
16 K	14	/18	255.255.192.0
32 K	15	/17	255.255.128.0
64 K	16	/16	255.255.0.0
128 K	17	/15	255.254.0.0
256 K	18	/14	255.252.0.0
512 K	19	/13	255.248.0.0
1 M	20	/12	255.240.0.0
2 M	21	/11	255.224.0.0
4 M	22	/10	255.192.0.0
8 M	23	/9	255.128.0.0
16 M	24	/8	255.0.0.0
32 M	25	/7	254.0.0.0
64 M	26	/6	252.0.0.0
128 M	27	/5	248.0.0.0
256 M	28	/4	240.0.0.0
512 M	29	/3	224.0.0.0
1024 M	30	/2	192.0.0.0
2048 M	31	/1	128.0.0.0
4096 M	32	/0	0.0.0.0

AWS VPC : Example

Lets understand the CIDR with an example. There is an requirement where you need to create network with 200 IP's and further divide it two sub networks with 100 IP's each.

This requirements doesn't fall into either class A, class B, class C because if we use one of it we will end up with wasting lots of IP4's.

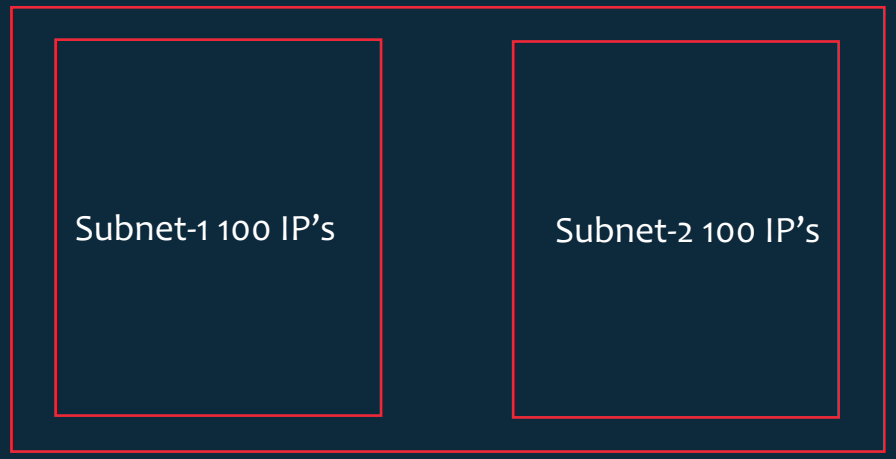
This where CIDR will come into picture lets go to our CIDR chart and check which CIDR block is nearer to our requirement.

/24 (32 – 24 = 8 => 2^8 = 256) CIDR block offers 256 IP address however we need only 200 IP's still fine because this some where nearer to our requirement.

So Now lets create VPC with 192.168.0.0/24 (Refer to VPC_Subnet_Routing_table doc to create VPC)

Now lets divided it two subnets with 100 IP addresses each. Lets go back our CIDR chart to check nearest CIDR block. /25 is nearer to your requirement it offers 128 IP address(32 – 25 = 7 => 2^7 = 128).

So for subnet-1 start address is 192.168.0.0/25 , what is the end address ? Without knowing the end address it will be difficult to create subnet-2 because subnet's-2 start address has to be derived from subnet's-1 end address.



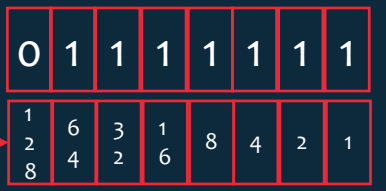
AWS VPC : Subnets start address and end address calculation

Subnet -1 start address and end address calculation ?

192.168.0.0/25 → CIDR Range

Subnet mask of 25

Negate Subnet mask of 25



0 . 0 . 0 . 127

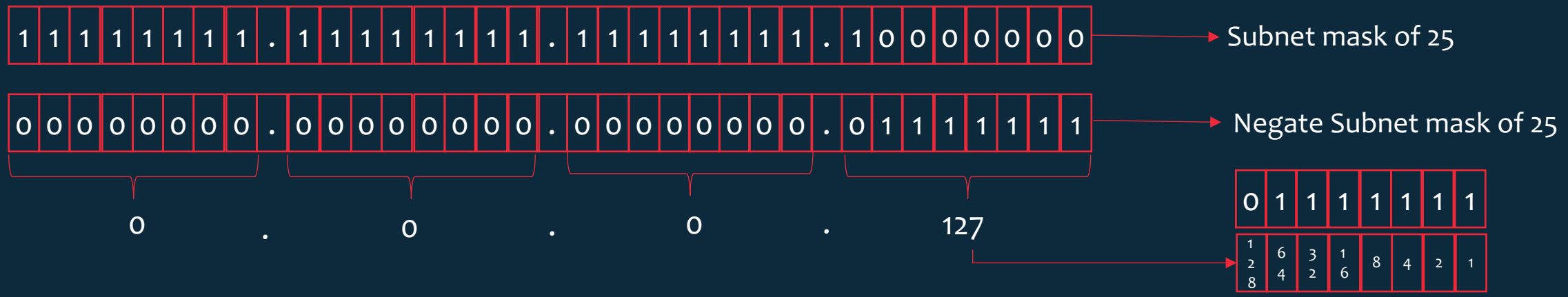
192 . 168 . 0 . 0 → Subnet-1 Start address

192 . 168 . 0 . 127 → Subnet-1 End address

192 . 168 . 0 . 128 → Subnet-2 start address

AWS VPC : Subnets start address and end address calculation

Subnet -2 start address and end address calculation ?
192.168.0.128/25 → CIDR Range



192 . 168 . 0 . 128 → Subnet-2 Start address

192 . 168 . 0 . 255 → Subnet-2 End address

Subnet Calculator

https://www.davidc.net/sites/default/subnets/subnets.html

RDS

https://aws.amazon.com/products/databases/?nc2=h_ql_prod_db

https://aws.amazon.com/rds/?c=db&sec=srv

EC2

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

VPC

https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html

Secrets Manager

https://aws.amazon.com/secrets-manager/

System Manager

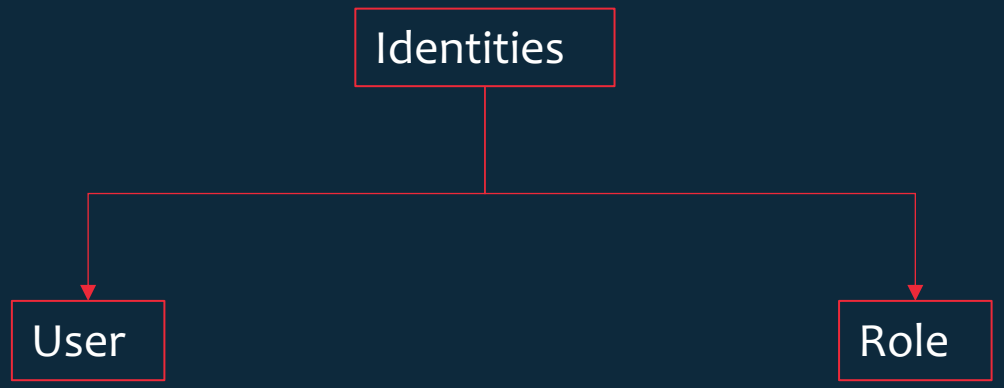
https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-parameter-store.html

S3

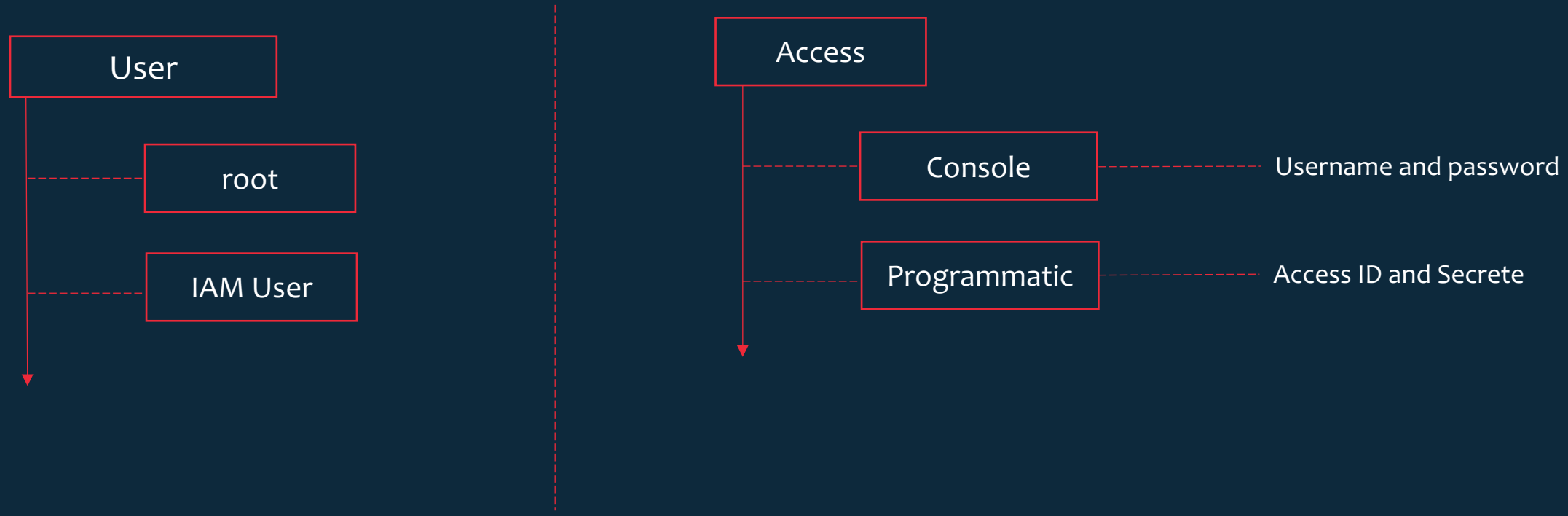
https://aws.amazon.com/s3/

Identity
Access
Management

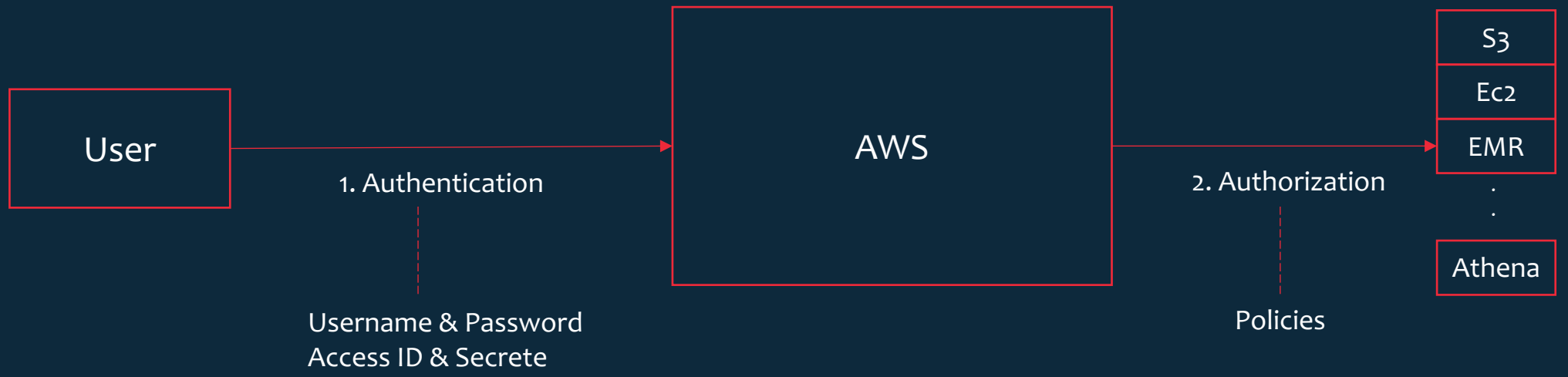
IAM :: Identities



IAM :: User(Outsider to AWS)



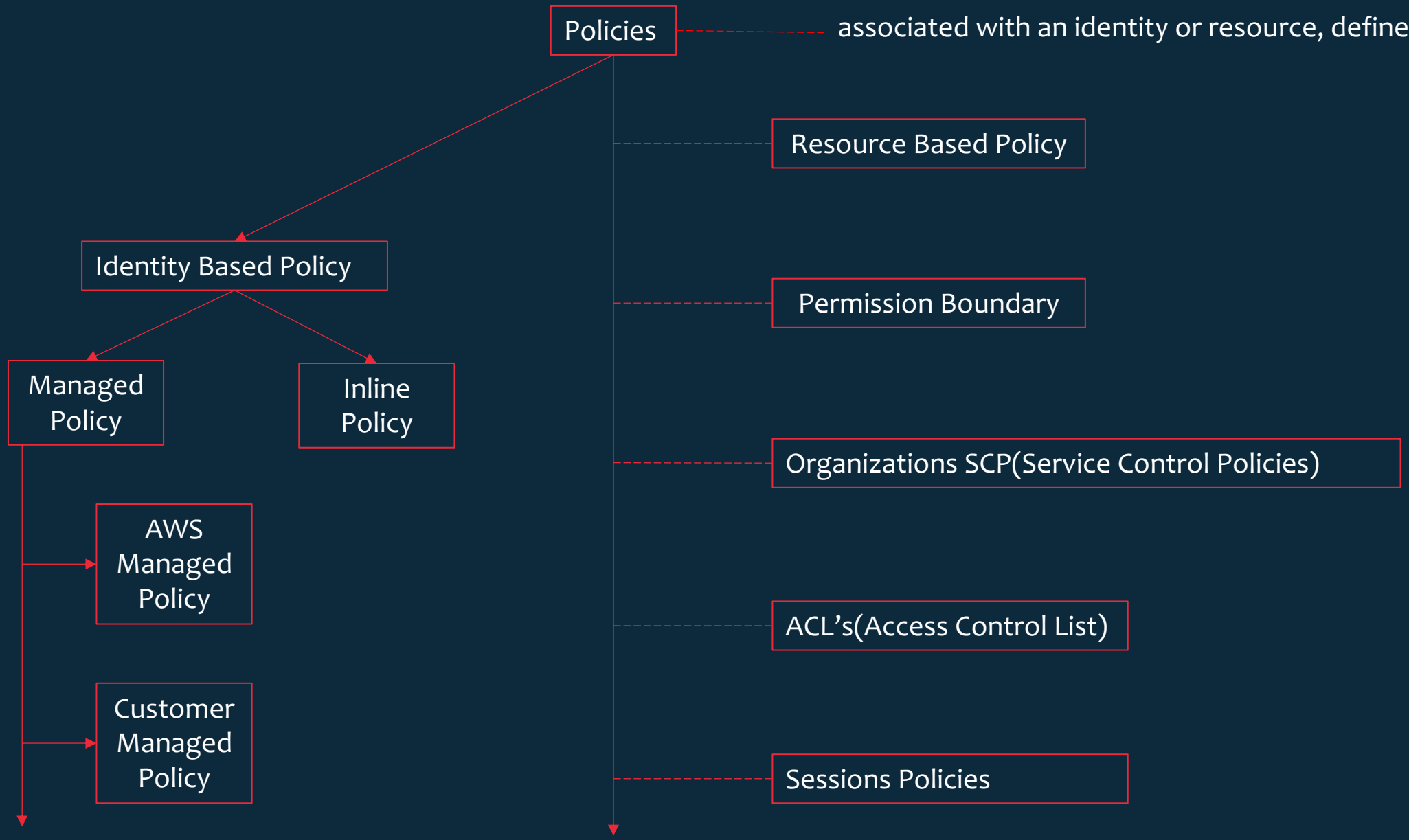
IAM :: Authentication and Authorization



1) Authentication : User logins into AWS either using Username and Password or Access ID and Secrete

2) Authorization : What actions to be performed by identity after successfully authentication ?
AWS offers different types of services. In the real time scenario user will not be having the access to all the services where the access will be restricted to only the certain services based on the project demands.
Restriction happens by defining "set of policies" and attaching them to User.

IAM :: Policies



IAM :: Identity polices :: Managed Policy

AWS Managed Policy :

- Created and Managed by AWS.
- AWS Managed policies are called as **Standalone policies**(Standalone policy means that the policy has its own Amazon Resource Name (ARN) that includes the policy name. For example, arn:aws:iam::aws:policy/IAMReadOnlyAccess is an AWS managed policy)
- You cannot change the permissions defined in AWS managed policies. AWS occasionally updates the permissions defined in an AWS managed policy.
- AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API calls become available for existing services. For example, the AWS managed policy called **ReadOnlyAccess** provides read-only access to all AWS services and resources. When AWS launches a new service, AWS updates the **ReadOnlyAccess** policy to add read-only permissions for the new service. The updated permissions are applied to all principal entities that the policy is attached to.

Examples :

- AdministratorAccess
- PowerUserAccess
- AWSCloudTrailReadOnlyAccess
- ReadOnlyAccess

IAM :: Identity polices :: Customer Managed Policy

Customer Managed Polices :

- Created and Managed by User/Administrators in your own AWS Account.
- A great way to create a customer managed policy is to start by copying an existing AWS managed policy
- You can then attach the policies to multiple principal entities in your AWS account. When you attach a policy to a principal entity, you give the entity the permissions that are defined in the policy.
- Each policy is an entity in IAM with its own Amazon Resource Name (ARN) that includes the policy name

IAM :: Policy Structure

https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements.html

```
{
        Version: “2012-10-17”,
        Statement : [
                {
                        SID : “”,
                        Effect : “”,
                        Action : [],
                        Resource : “”,
                        Condition : {
                        }
                }
        ]
}
```

The Version policy element specifies the language syntax rules that are to be used to process a policy

Amazon Resource Name (ARN)

arn:partition:service:region:account:resource

Partition: identifies the partition for the resource . For standard AWS Regions, the partition is aws. If you have resources in other partitions, the partition is aws-partitionname. For example, the partition for resources in the China (Beijing) Region is aws-cn. You cannot delegate access between accounts in different partitions.

service identifies the AWS product. IAM resources always uses iam.

region identifies the Region of the resource. For IAM resources, this is always kept blank.

account specifies the AWS account ID with no hyphens.

resource identifies the specific resource by name.

Examples:
arn:aws:iam::123456789012:user/JohnDoe
arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/JaneDoe
arn:aws:iam::123456789012:role/S3Access

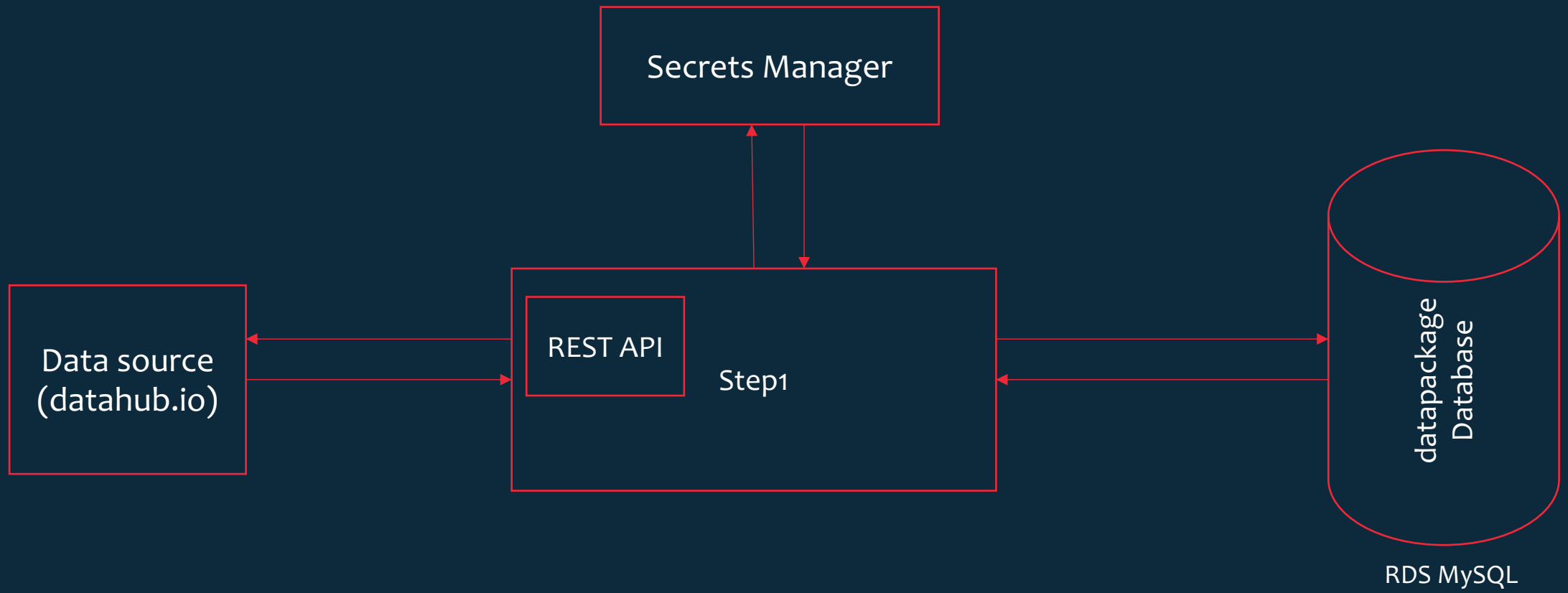
Lambda

https://aws.amazon.com/lambda/

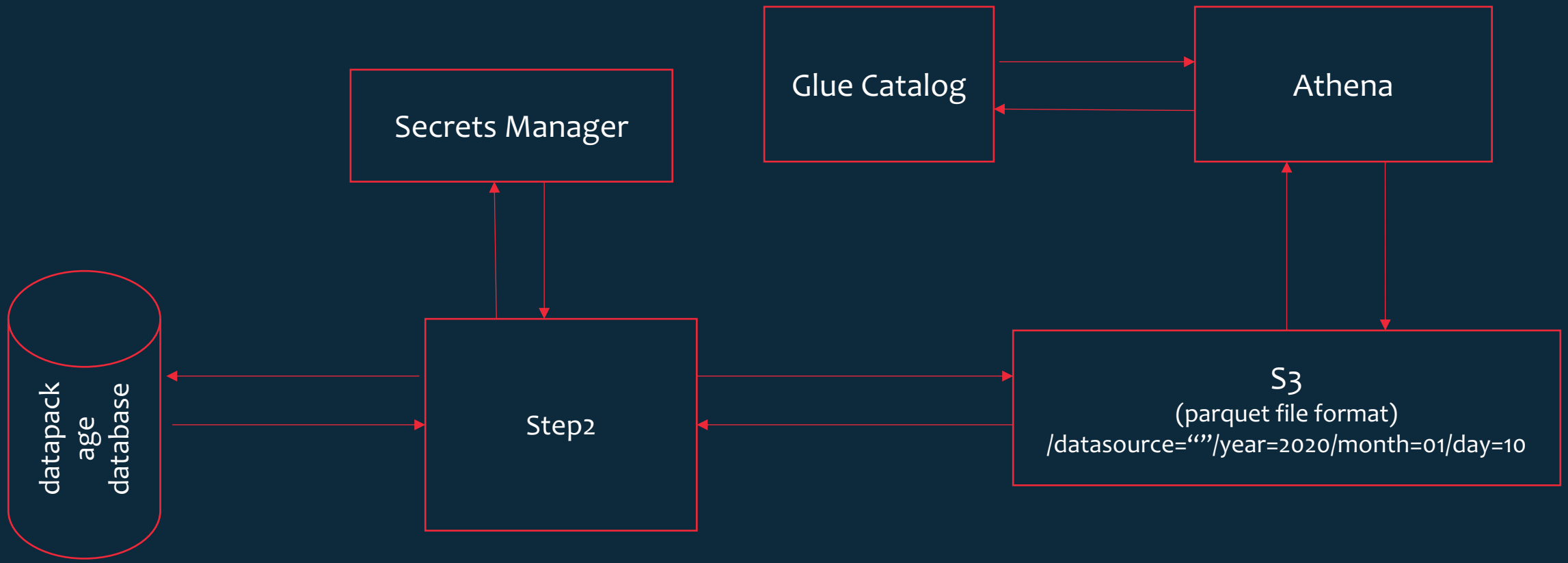
Athena

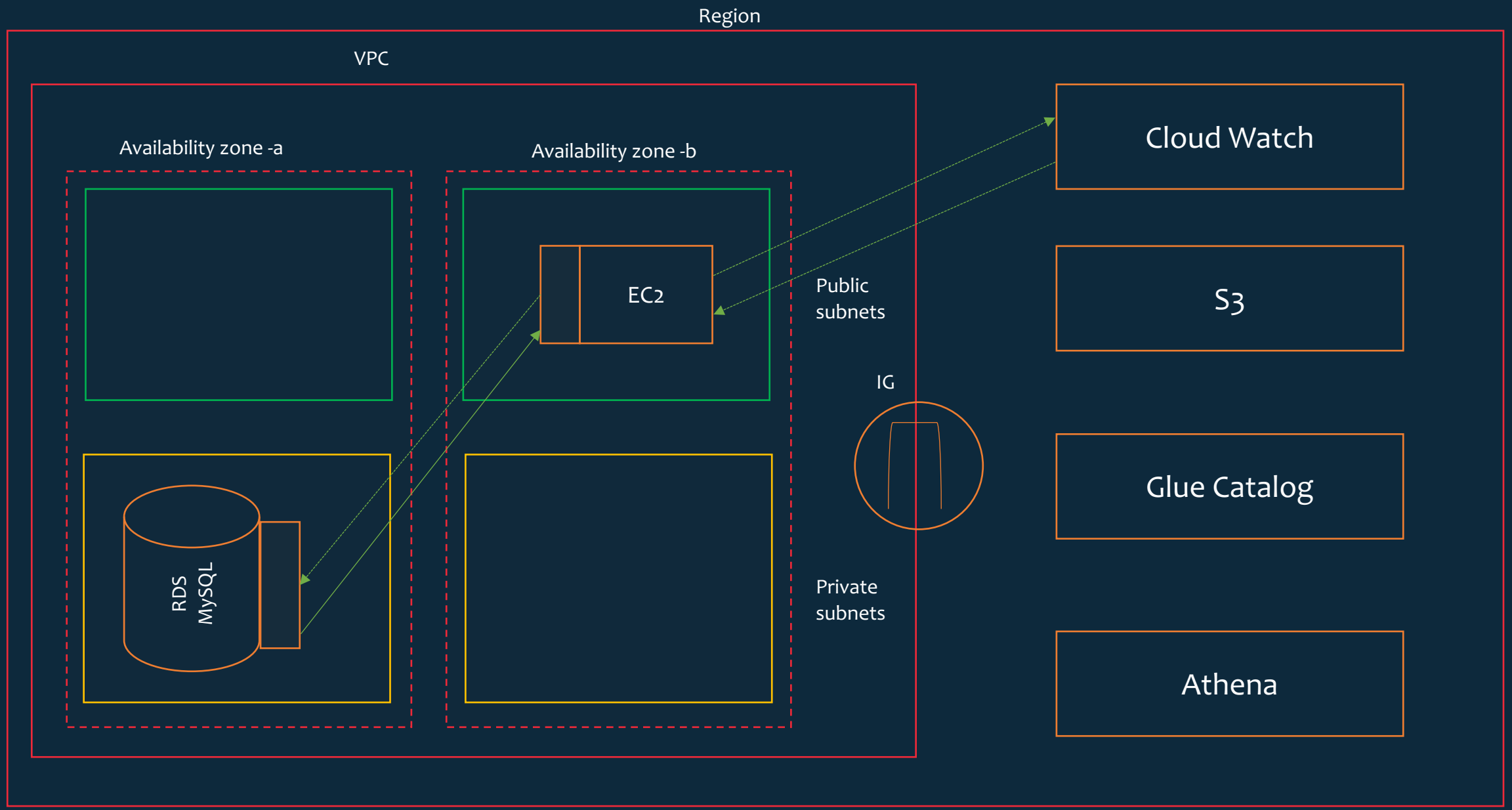
https://www.amazonaws.cn/en/athena/

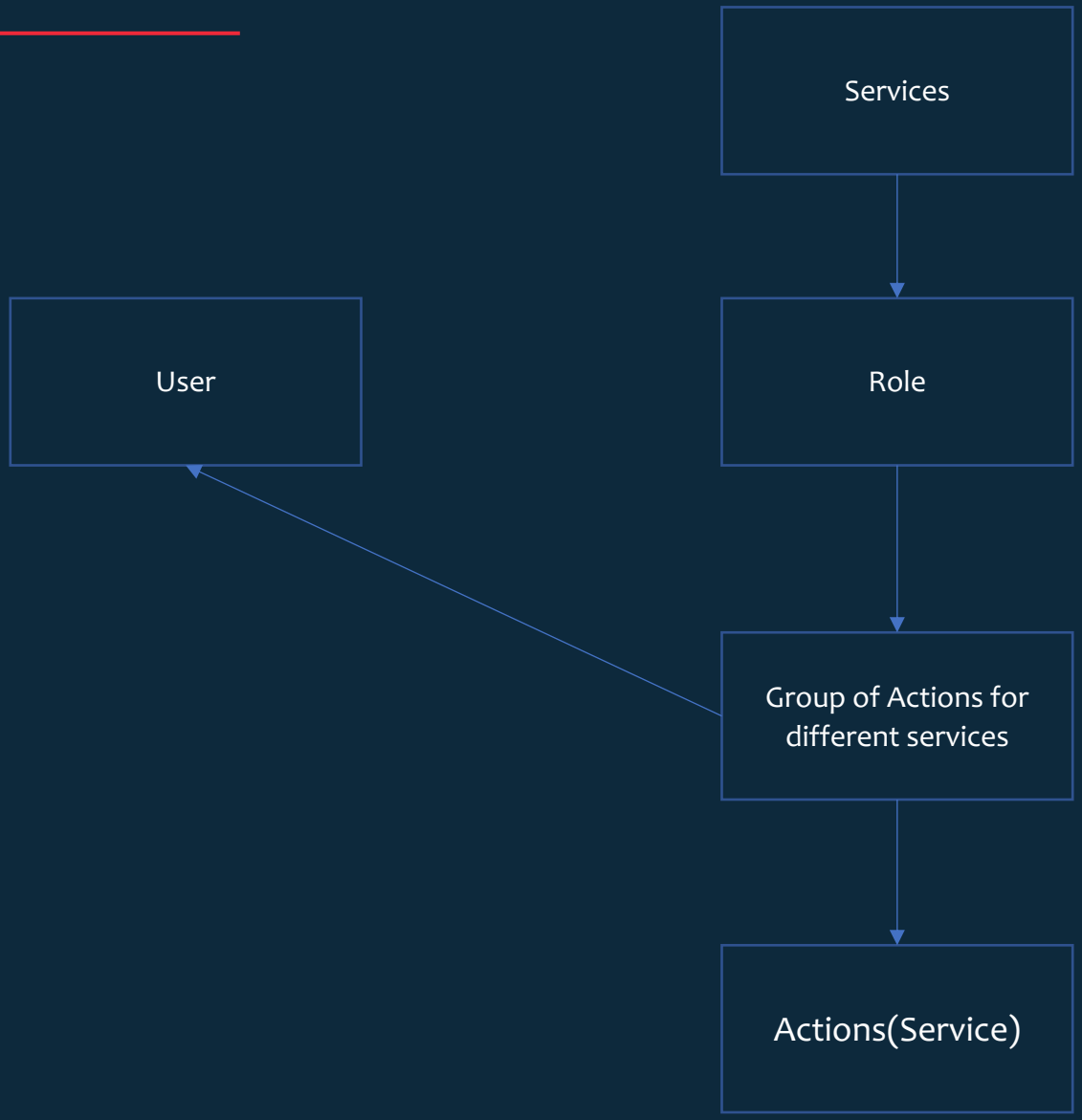
AWS Project : Step1



Project : Step2




Region
VPC
Availability zone -a
Availability zone -b
EC2
Public subnets
IG
RDS MySQL
Private subnets
Cloud Watch
S3
Glue Catalog
Athena


Services
User
Role
Group of Actions for different services
Actions(Service)

YAML(Yet Another Markup Language)

It is a serialization language. It's often used as a format for configuration files, but its object serialization abilities make it a viable replacement for languages like JSON

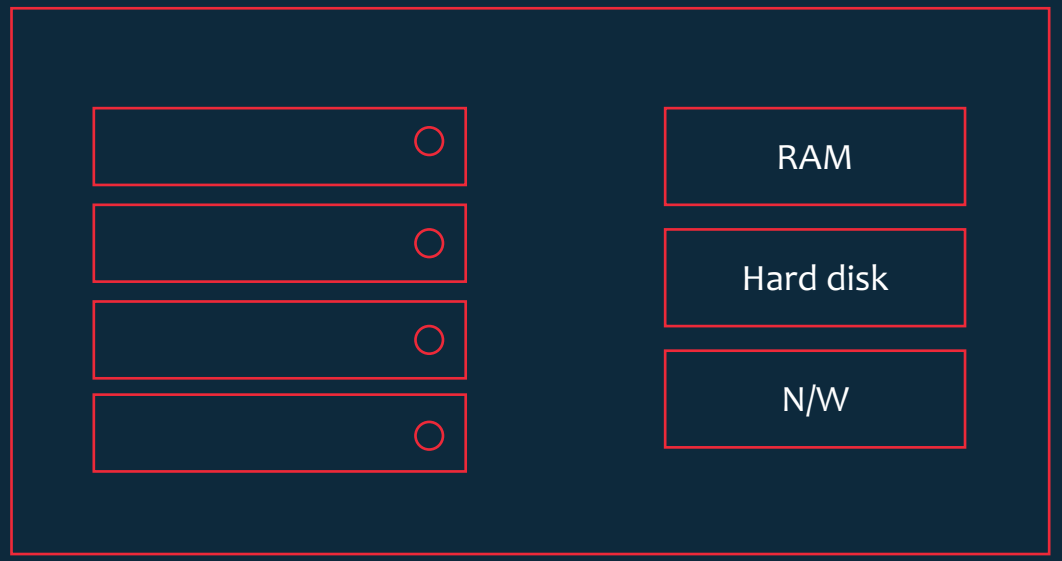
```
{
 "doe": "a deer, a female deer",
 "ray": "a drop of golden sun",
 "pi": 3.14159,
 "xmas": true,
 "french-hens": 3,
 "calling-birds": [
   "huey",
   "dewey",
   "louie",
   "fred"
 ],
 "xmas-fifth-day": {
 "calling-birds": "four",
 "french-hens": 3,
 "golden-rings": 5,
 "partridges": {
  "count": 1,
  "location": "a pear tree"
 },
 "turtle-doves": "two"
 }
}
```

JSON

```
doe: "a deer, a female deer"
ray: "a drop of golden sun"
pi: 3.14159
xmas: true
french-hens: 3
calling-birds:
  - huey
  - dewey
  - louie
  - fred
xmas-fifth-day:
  calling-birds: four
  french-hens: 3
  golden-rings: 5
  partridges:
    count: 1
    location: "a pear tree"
  turtle-doves: two
```

YAML

Spark Module



Server – 4 cores, 16GB RAM, 1 TB hard disk, 2.4GHz

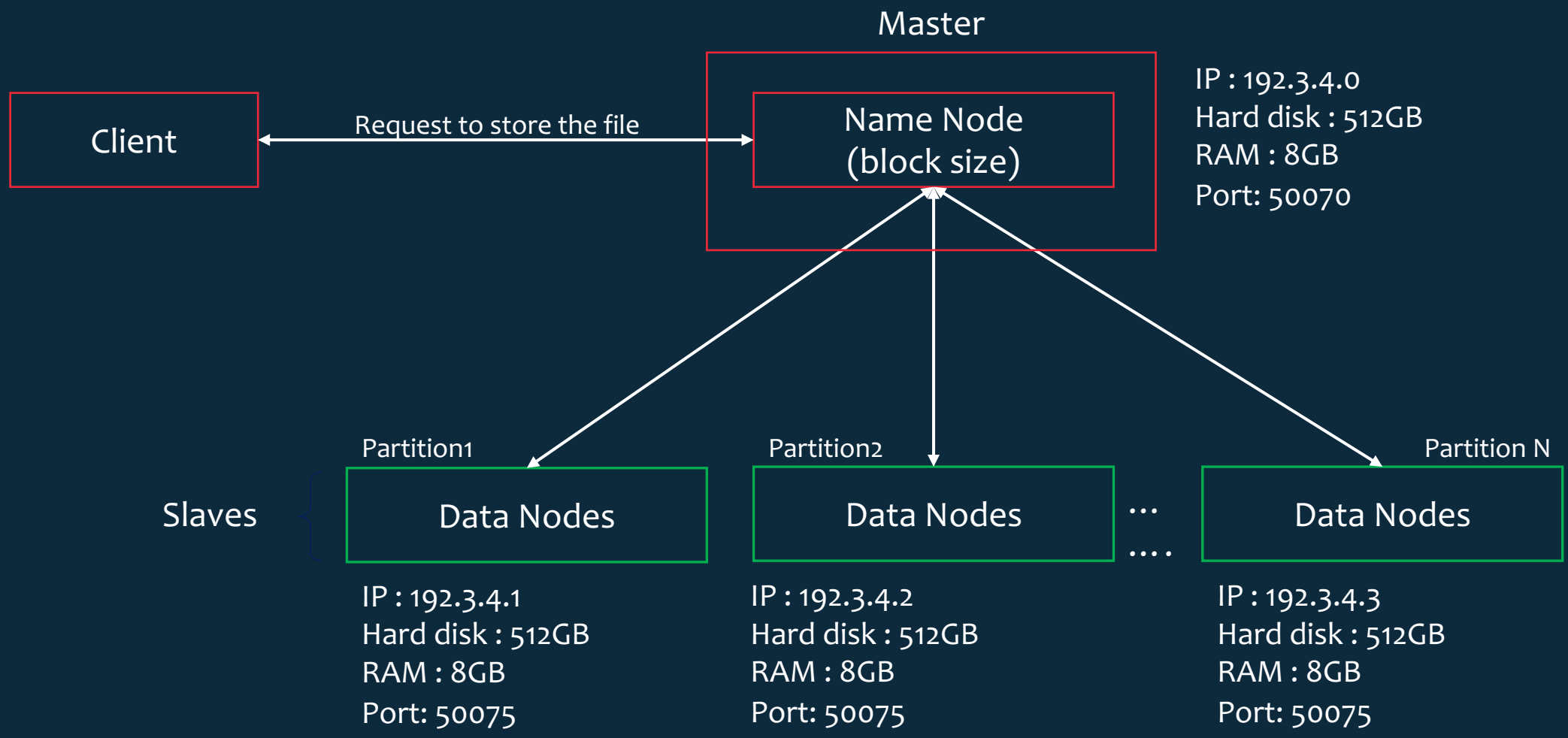
Limitations?

→Vertical Scaling of the Resources
→Fault Tolerance and HA
→OS Limitation → number of file that you open
→Garbage Collections cycle

Solutions

→ Horizontal Scaling
  → You can scale your resources by adding servers on fly
  → HA and Fault Tolerance
  → Workload distributions

Recall HDFS

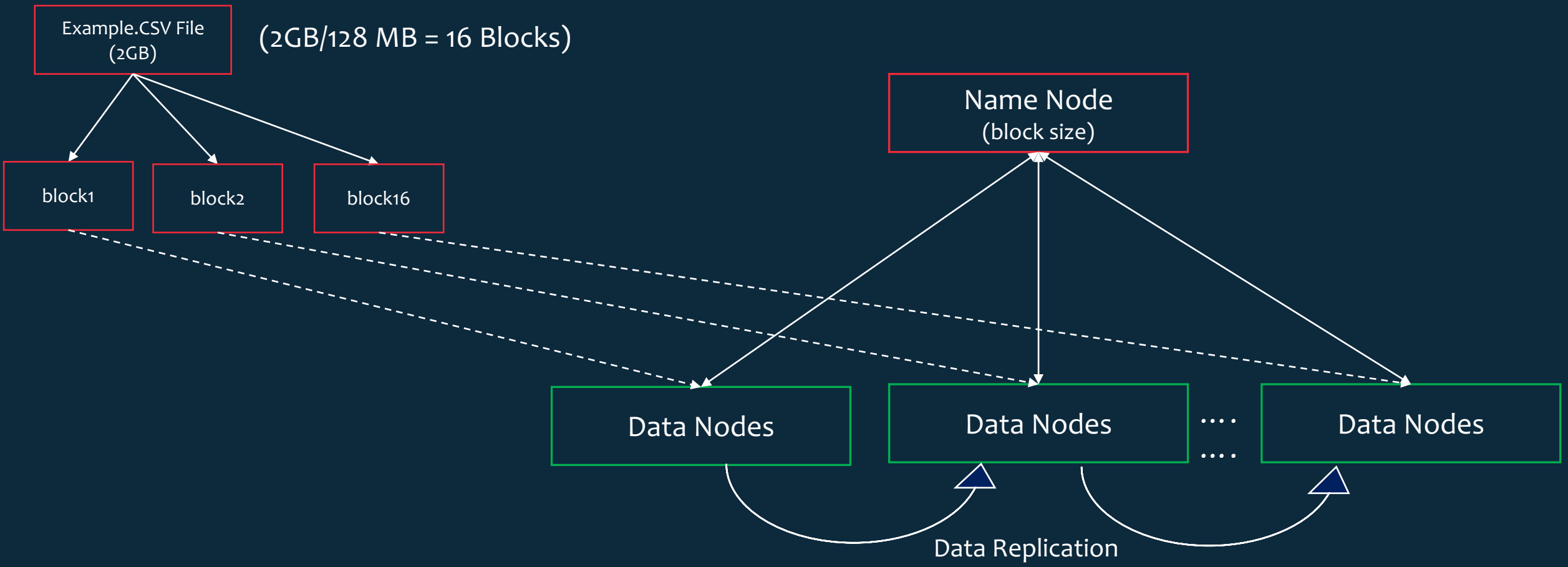


Note: Name Node and Data Node are the JVM process and they will in a dedicated server with a dedicated port number(Name Node: 50070, Data Node: 50075)

Recall HDFS

There are three important parameters which will control the behaviour of HDFS
Block size = 128MB, Split size = 256MB(By default split size is equal to block size), Replication Factor = 3

When you try to copy the csv file of size 2GB into HDFS. The file will be split into blocks based on the block size and each block will be stored into each data node and its further replicated into data node to achieve the fault tolerance. The metadata information of the file is stored into NameNode



Recall HDFS

Name Node : It records the metadata of all the files stored in the cluster, such as location of blocks stored, size of the files, permissions, hierarchy, etc.

In the above example CSV file is divided into 16 blocks and each block is stored into different data nodes.
So now lets say imagine Spark or MapReduce wants to read and process that CSV file then they will reach out name node for the meta data information using that they will distribute the work to executer(spark) or mappers(MR).

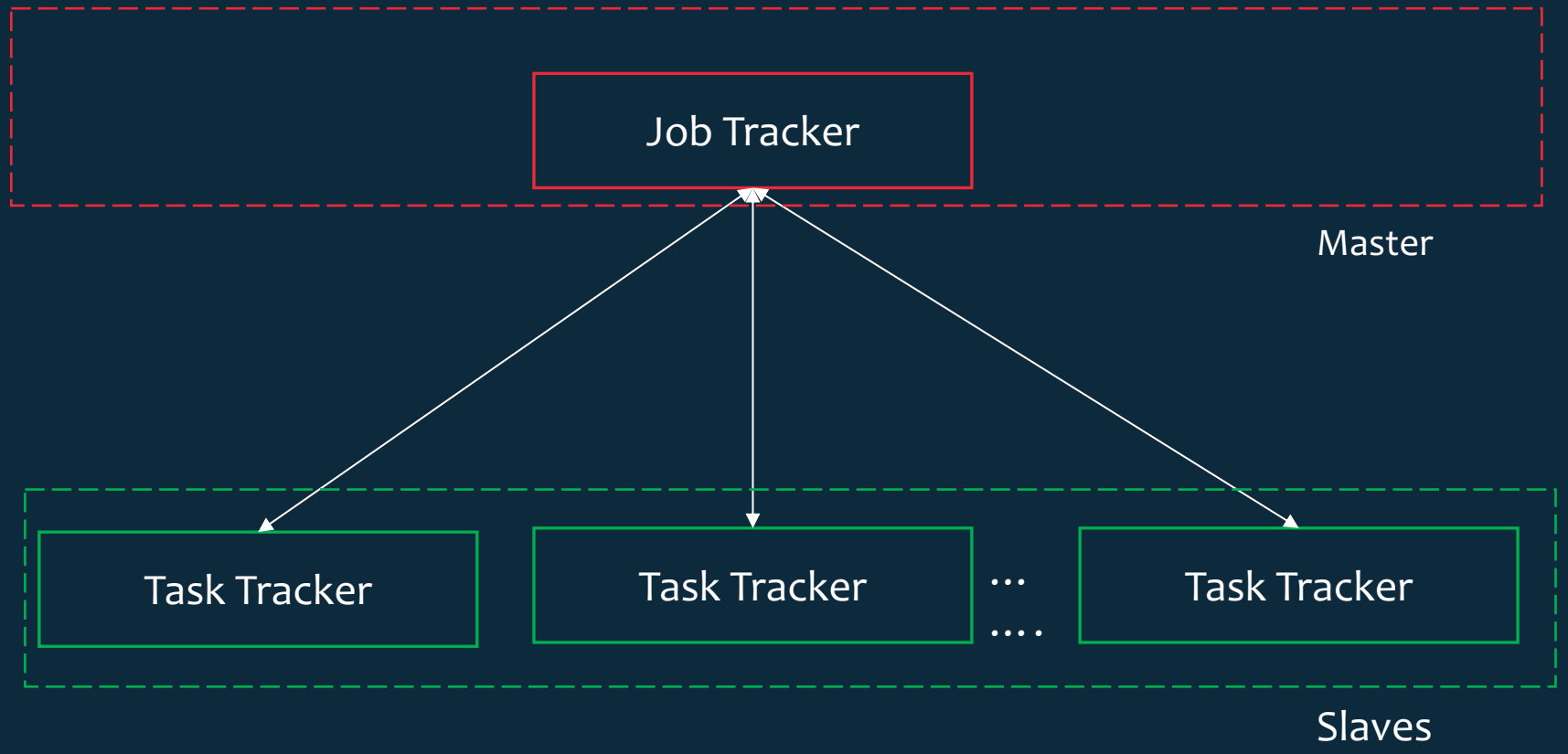
File Name	Part Files	Data Node
Example.csv	part1.csv,part2.csv,part3.csv,part4.csv	DN1
	part5.csv,part6.csv,part7.csv,part8.csv	DN2
	part9.csv,part10.csv,part11.csv,part12.csv	DN3
	part13.csv,part14.csv,part15.csv,part16.csv	DN3

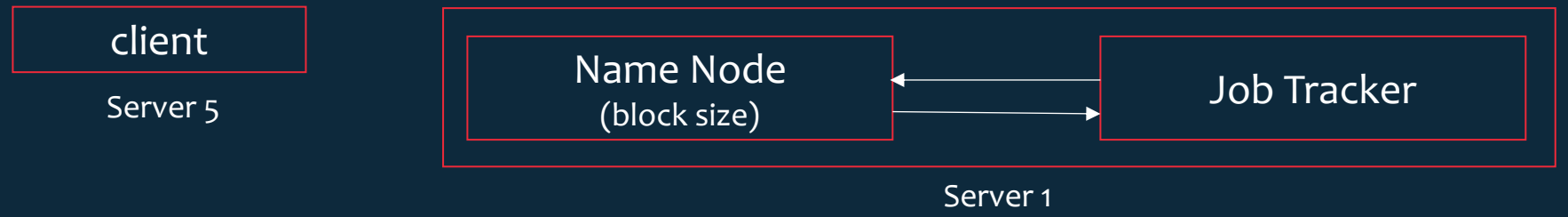
Data Node IP:Port	Alias
http://192.3.4.1:50075	DN1
http://192.3.4.2:50075	DN2
http://192.3.4.3:50075	DN3

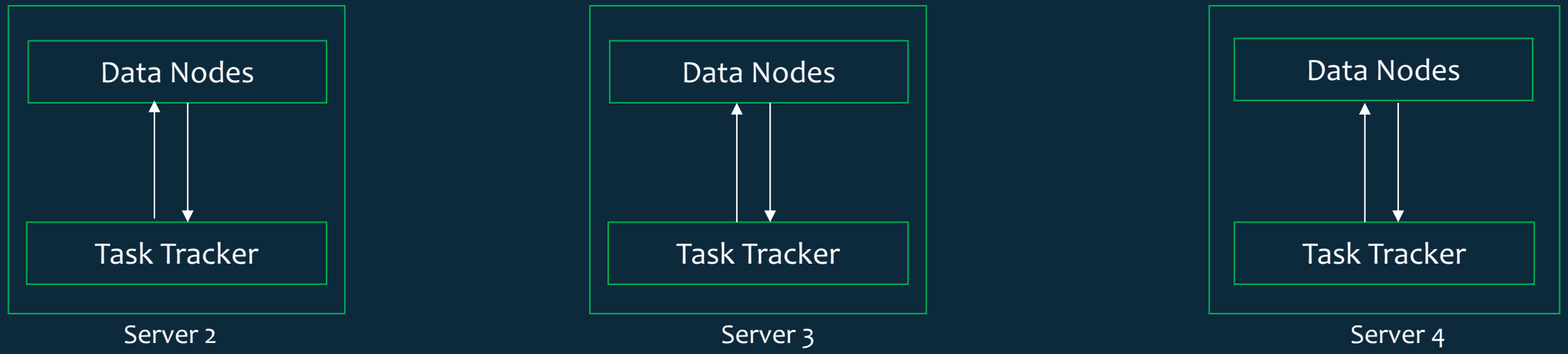
Note: For each HDFS Block one new part-<block number> will be created and data of that block will be in that part file.

For Example.csv after copying into HDFS 16 part files will be created for 16 blocks

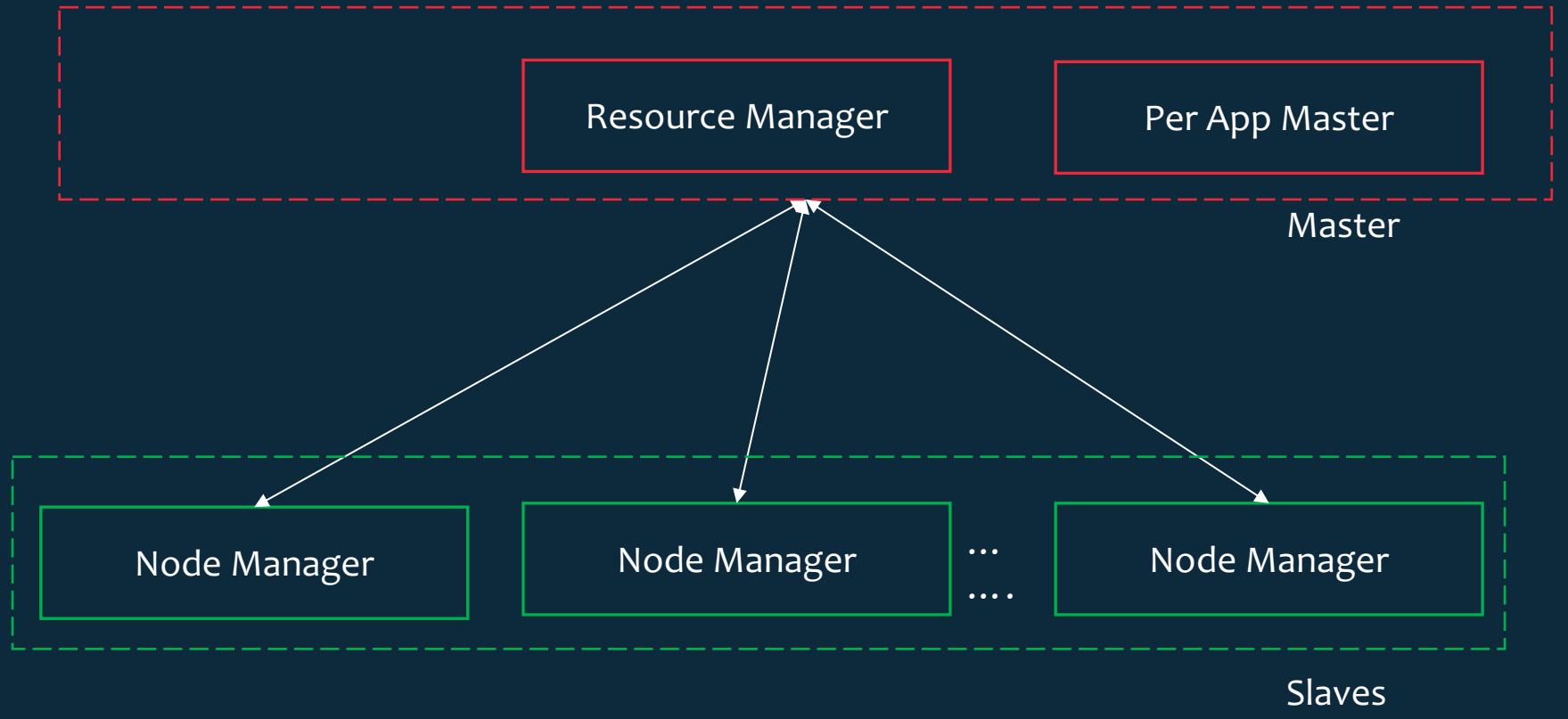
Recall Map/Reduce 1



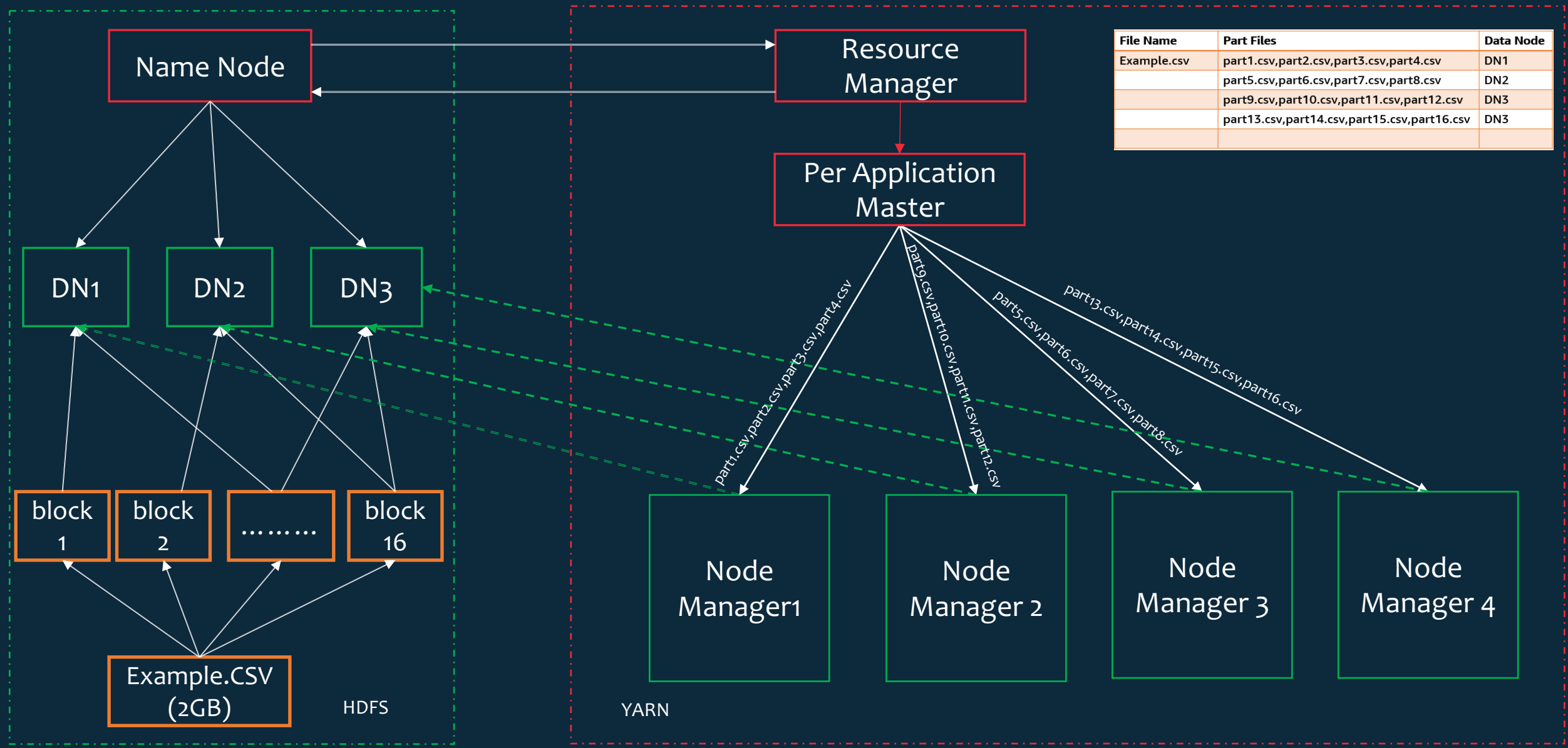

client
Server 5
Name Node
(block size)
Job Tracker
Server 1


Data Nodes
Task Tracker
Server 2
Data Nodes
Task Tracker
Server 3
Data Nodes
Task Tracker
Server 4

Recall YARN(Yet Another Resource Negotiator)(MR 2)



High Level View of Spark Read API From HDFS



File Name	Part Files	Data Node
Example.csv	part1.csv,part2.csv,part3.csv,part4.csv	DN1
	part5.csv,part6.csv,part7.csv,part8.csv	DN2
	part9.csv,part10.csv,part11.csv,part12.csv	DN3
	part13.csv,part14.csv,part15.csv,part16.csv	DN3

Map/Reduce at High level

At a high level Map-Reduce has two phases in it Mapper and Reducer . Mapper are used to perform transformations and Reducers are used to perform the aggregations.

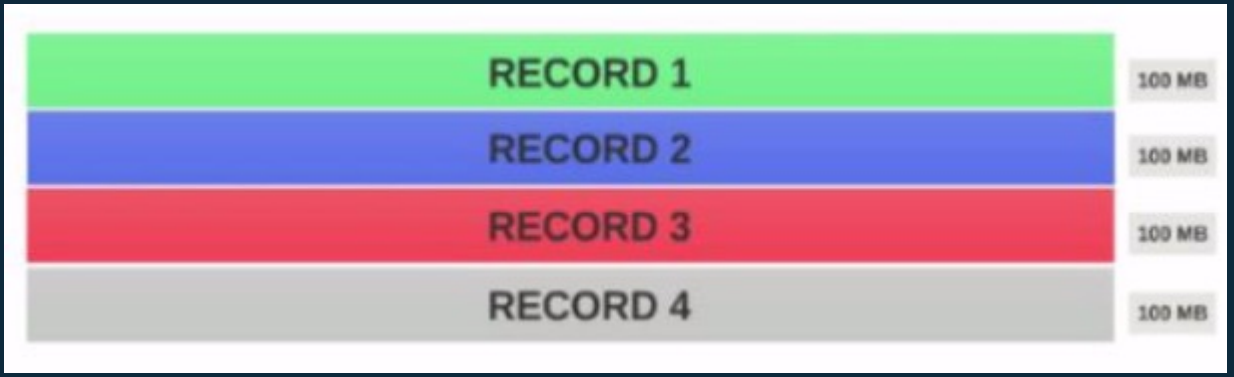
How many mapper will be launched to process on file?
Number of Mapper is completely depends on the input split size.
By defaults Input-Split size = Block Size

In CSV file case that we discussed in "Recall HDFS" created 16 blocks for 2GB file for that it creates 16 Mappers jobs. One mapper completes the transformations it will store that intermediate data into the disk by default those intermediate files are compressed using Bzip2 Codec. Those intermediate mapper output files will be input(shuffled) to the reducer job it will be decompressed and then reducer job will perform aggregation for each key.

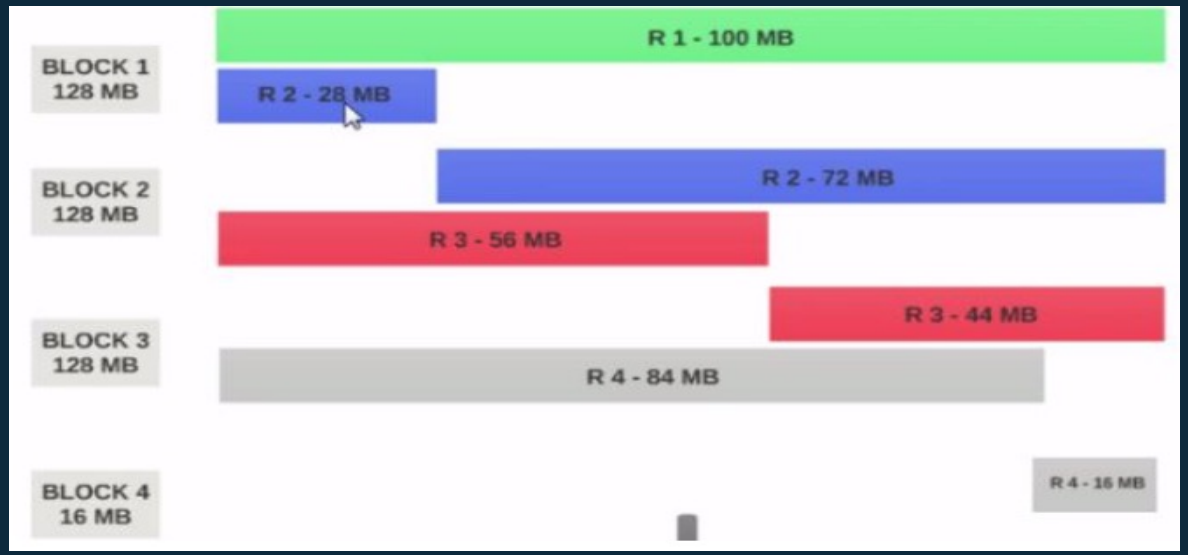
Number of Reducer job is all based on "mapred.reduce.tasks" configuration we set.

When the input split really matters

Assume we have a file of 400MB with consists of 4 records(e.g : csv file of 400MB and it has 4 rows, 100MB each)



If the HDFS Block Size is configured as 128MB, then the 4 records will not be distributed among the blocks evenly. It will look like this.



- **Block 1** contains the entire first record and a 128MB chunk of the second record.
- If a mapper is to be run on **Block 1**, the mapper cannot process since it won't have the entire second record.
- This is the exact problem that **input splits** solve. **Input splits** respects logical record boundaries.

When the input split really matters

•**Lets Assume the input split size is 200MB**

•Therefore the **input split 1** should have both the record 1 and record 2. And input split 2 will not start with the record 2 since record 2 has been assigned to input split 1. Input split 2 will start with record 3.

•This is why an input split is only a **logical chunk** of data. It points to start and end locations with in blocks.

•If the input split size is n times the block size, an input split could fit multiple blocks and therefore less number of **Mappers** needed for the whole job and therefore less parallelism. (Number of mappers is the number of input splits)

Py4J

Py4J(A Bridge between Python and Java): Py4J enables Python programs running in a Python interpreter to dynamically access Java objects in a Java Virtual Machine

- → Java Gateway:
  - →A JavaGateway is the main interaction point between a Python VM and a JVM.
  - →By Default Python tries to connect to JVM with a gateway on localhost on port 25333.
  - →Technically, a **Gateway Server** is only responsible for accepting connection
- → Entry point
  - → Its a reference to returned objects.

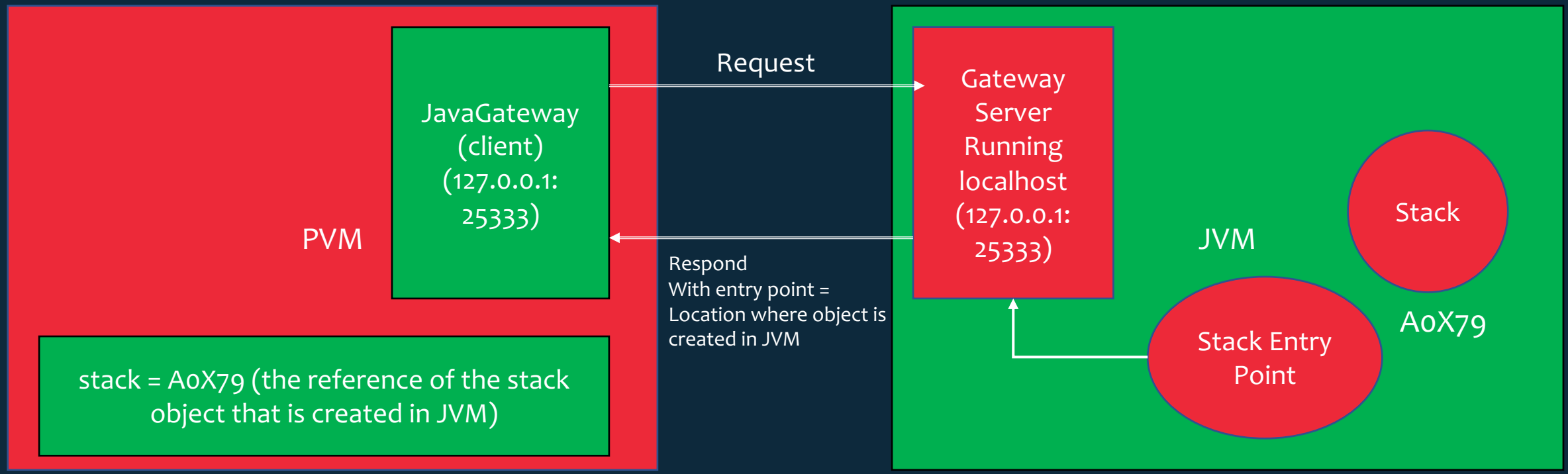
JAVA:

```
package py4j.examples;

import py4j.GatewayServer;

public class StackEntryPoint {

  private Stack stack;

  public StackEntryPoint() {
   stack = new Stack();
   stack.push("Initial Item");
  }

  public Stack getStack() {
    return stack;
  }

  public static void main(String[] args) {
    GatewayServer gatewayServer = new GatewayServer(new StackEntryPoint());
    gatewayServer.start();
    System.out.println("Gateway Server Started");
  }
}
```

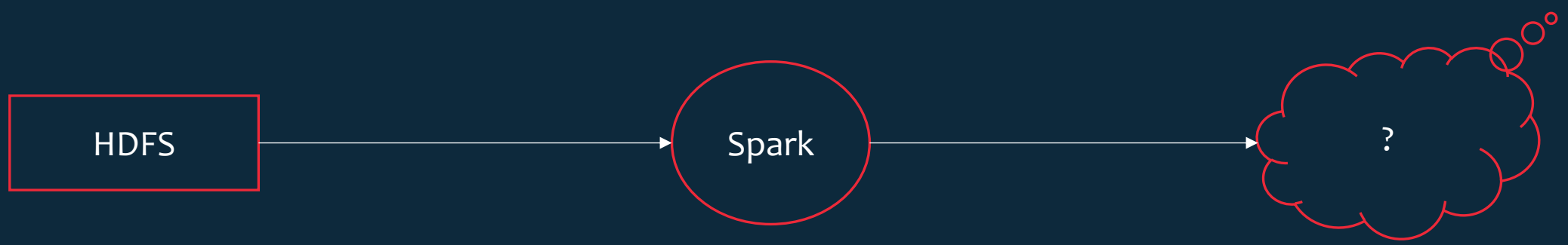
Python:

```
from py4j.java_gateway import JavaGateway
gateway = JavaGateway()
stack = gateway.entry_point.getStack()
stack.push("First %s" % ('item'))
stack.push("Second item")
print(stack.pop())
print(stack.pop())
print(stack.pop())
```

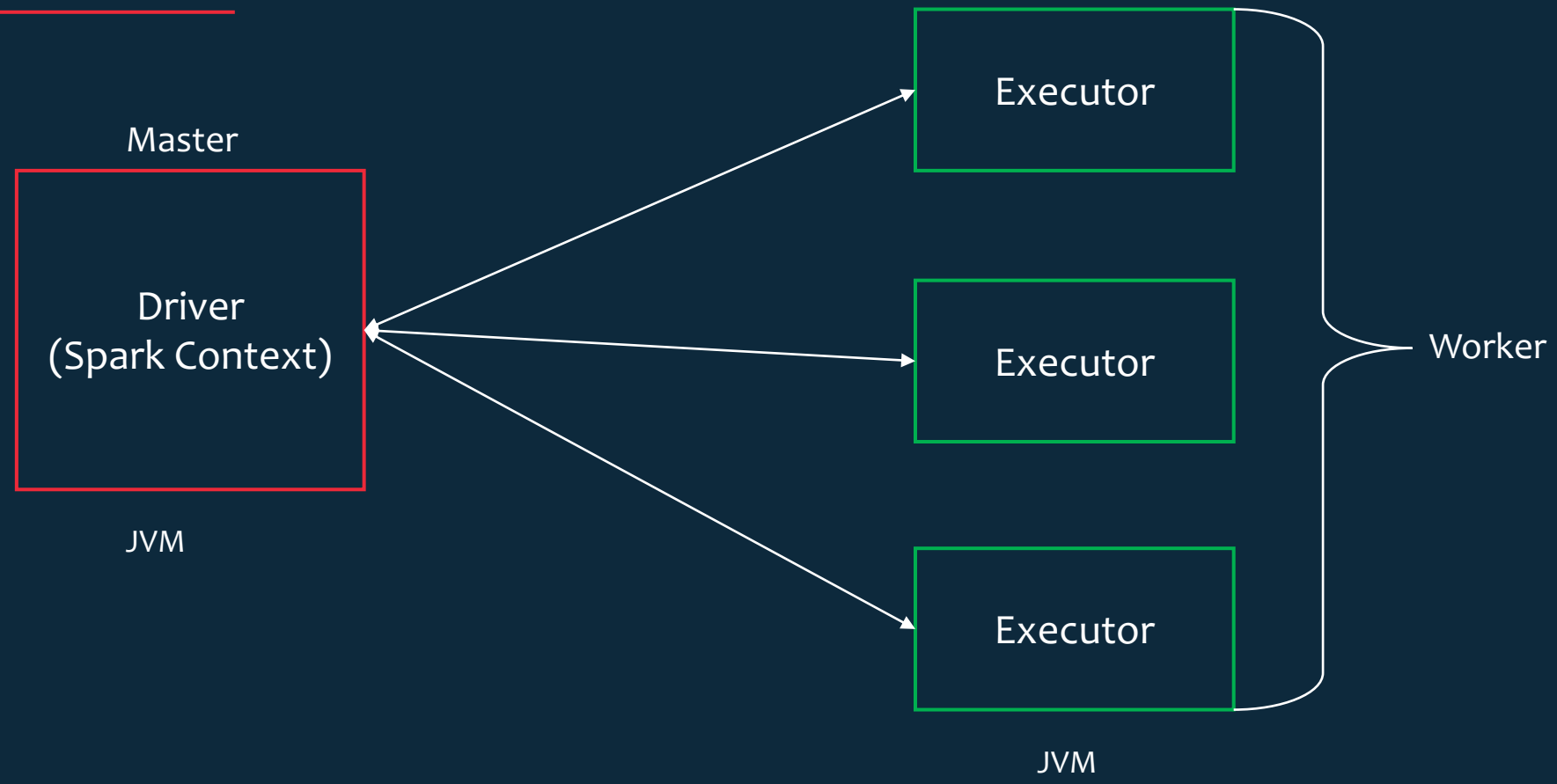
Py4J


JavaGateway
(client)
(127.0.0.1:
25333)
PVM
Request
Respond
With entry point =
Location where object is
created in JVM
Gateway
Server
Running
localhost
(127.0.0.1:
25333)
JVM
Stack
A0X79
Stack Entry
Point
stack = A0X79 (the reference of the stack
object that is created in JVM)

What Happens when you try to read the file from HDFS?

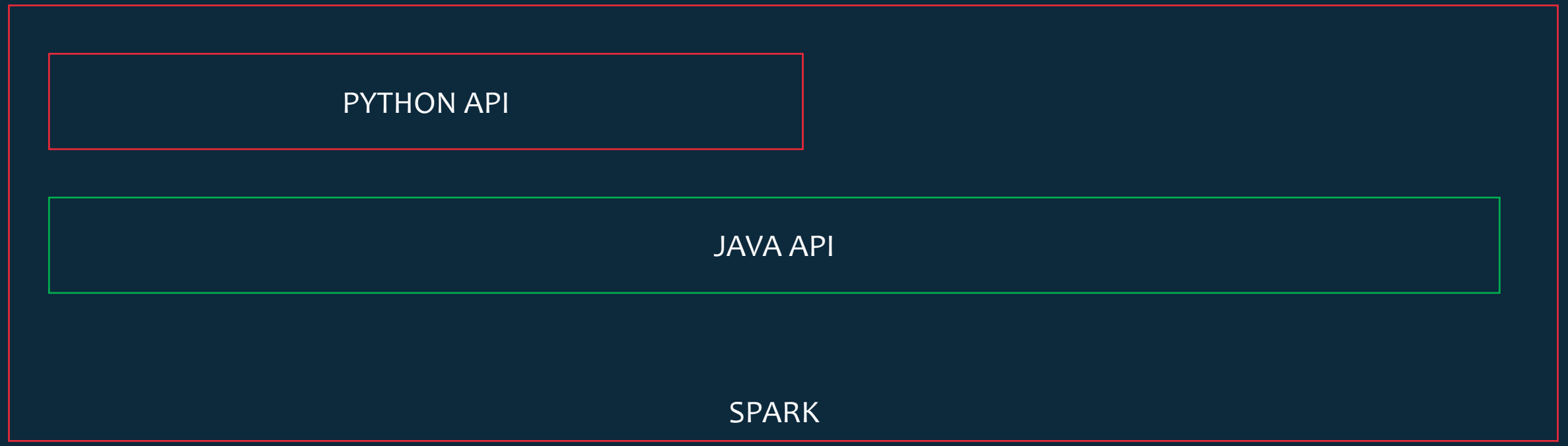


Spark Architecture



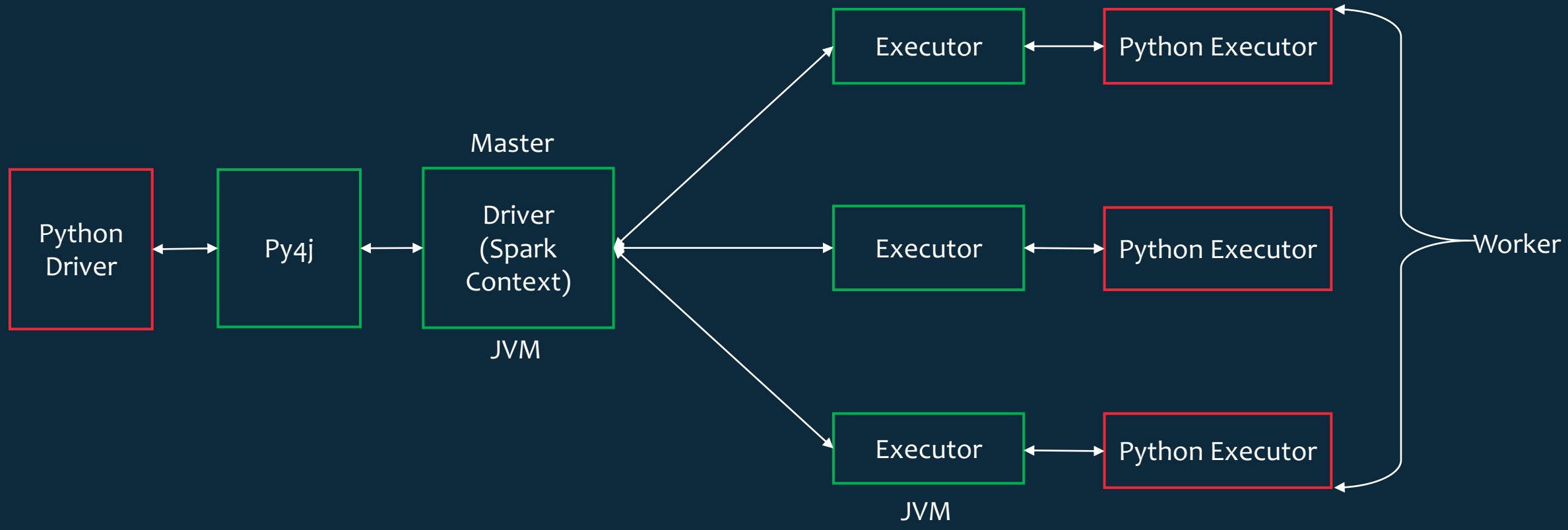
- Spark Context acts like channel where driver uses it to communicate with executor.
- Both Driver and Executors are the java process which will runs in there dedicated JVM containers

PYSPARK API HIGH LEVEL

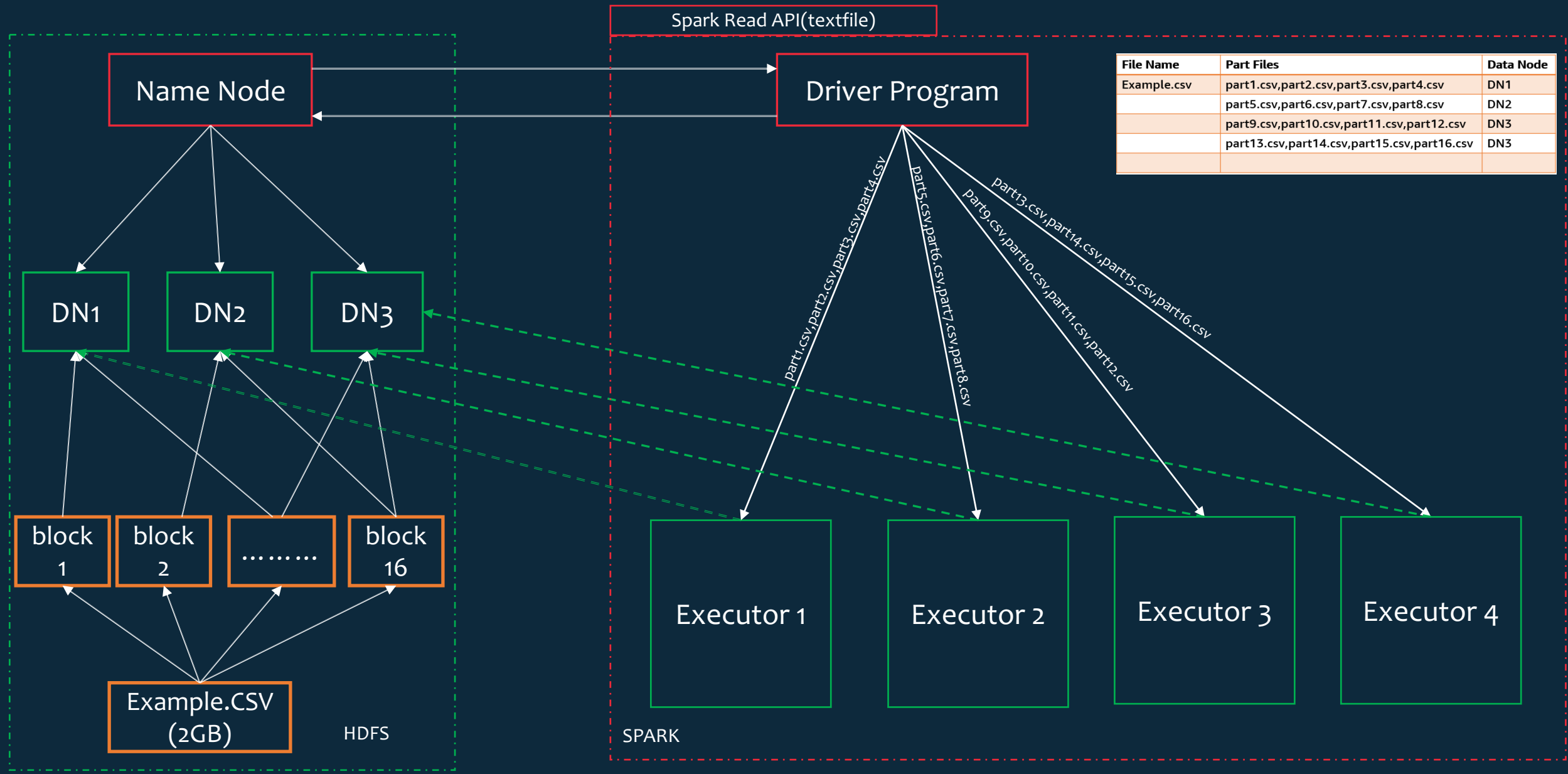


The PySpark API is a “very thin layer” on top of the Java API for spark, which itself is only a wrapper around the core Scala functionality.

Py-Spark Architecture

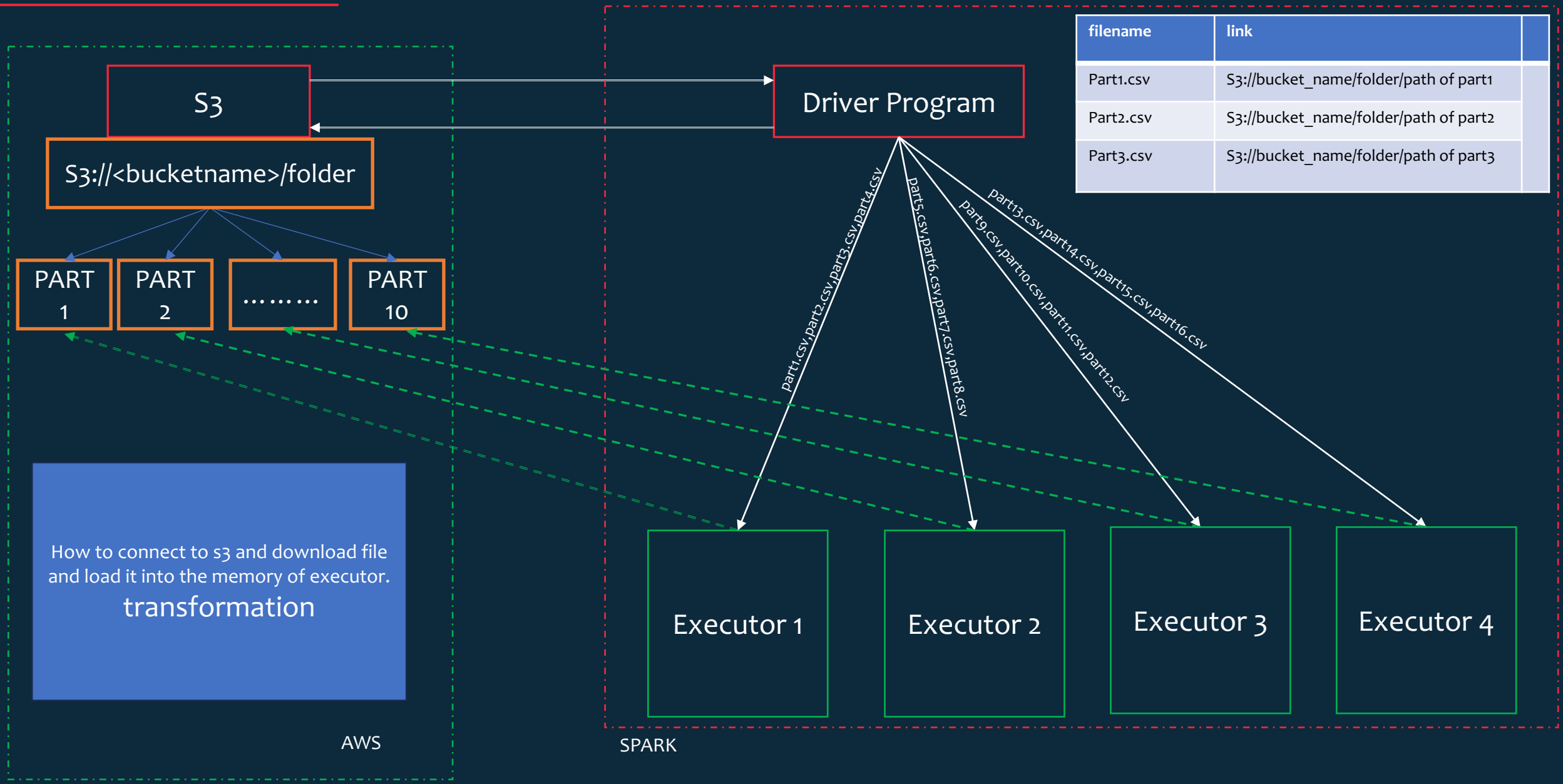


High Level View of Spark Read API From HDFS



File Name	Part Files	Data Node
Example.csv	part1.csv,part2.csv,part3.csv,part4.csv	DN1
	part5.csv,part6.csv,part7.csv,part8.csv	DN2
	part9.csv,part10.csv,part11.csv,part12.csv	DN3
	part13.csv,part14.csv,part15.csv,part16.csv	DN3

High Level View of Spark Read API From HDFS



filename	link
Part1.csv	S3://bucket_name/folder/path of part1
Part2.csv	S3://bucket_name/folder/path of part2
Part3.csv	S3://bucket_name/folder/path of part3

High Level View of Spark Read API From HDFS

The whole idea of the spark driver is to distribute the work across the executors(distribute the data) and monitor them.

When you try to read and process a file(Example.csv) from HDFS.

→ Spark driver will make a request to Name Node to get the Metadata information of the file.

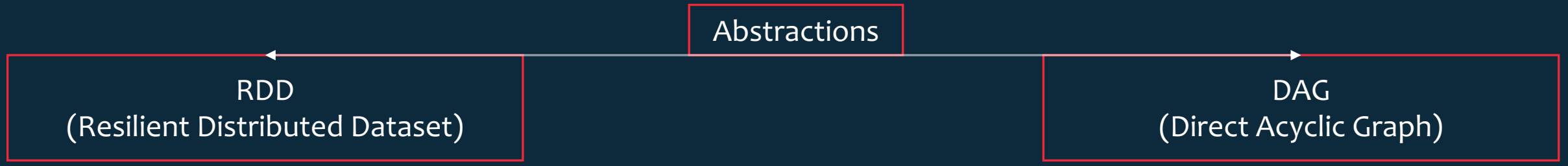
→ Once the Name Node receives the request from driver it will check whether the request is authorized to serve or not. If the request is valid Name Node will respond back with the Metadata info of the file to Driver

→ One the driver gets the meta information it looks at it and see how many blocks are there for that file to process.

→ Then those blocks info are distributed across the Executors. Once the executor receives the request from driver to process a particular block(It has the address(ip : port no) of data node in which that block/part file is residing) then executor will connect to its respective Data Node and get the block data into its memory for processing.

Driver needs a efficient way to Monitor, distribute and compute on the distributed data.

Major Abstractions in Spark



Question your self before processing the data in spark

Datastore?(where data is stored)
HDFS,S3,Blob,Local Storage, NFS.

Type of Data?
.PNG, txt, csv, parquet, ORC, json, xml

Codec(compression and decompression algo)?
gzip, bzip2, lzo, snappy

RDD(Resilient Distributed Dataset) : Basic Unit Of Parallelism

Resilient : Fault Tolerant and Ability to capable of rebuild state on failure.

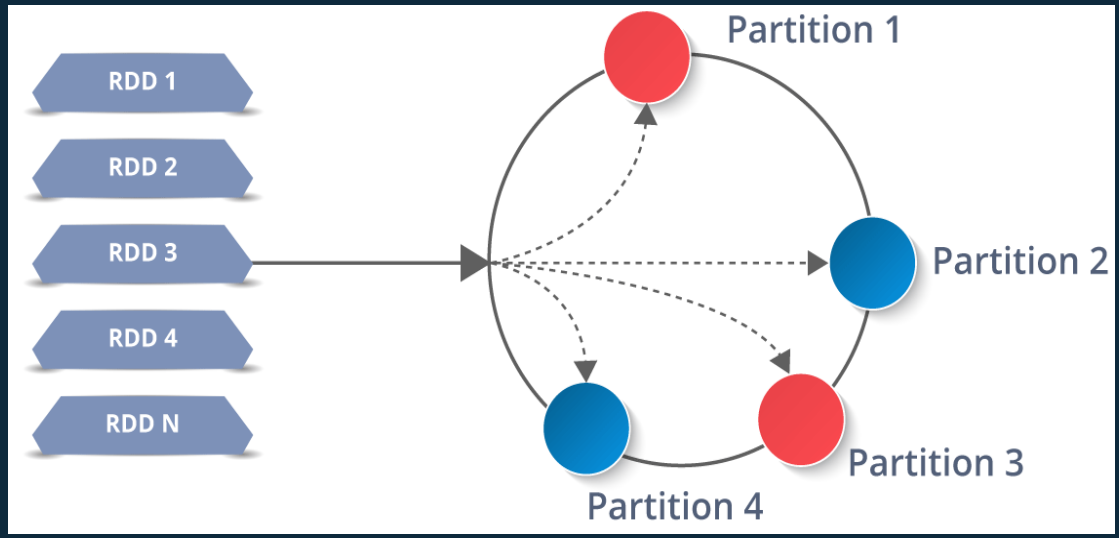
Distributed : Distributed the Data among multiple nodes in the cluster

Dataset : Collection of partitioned data with values

Points to Remember:

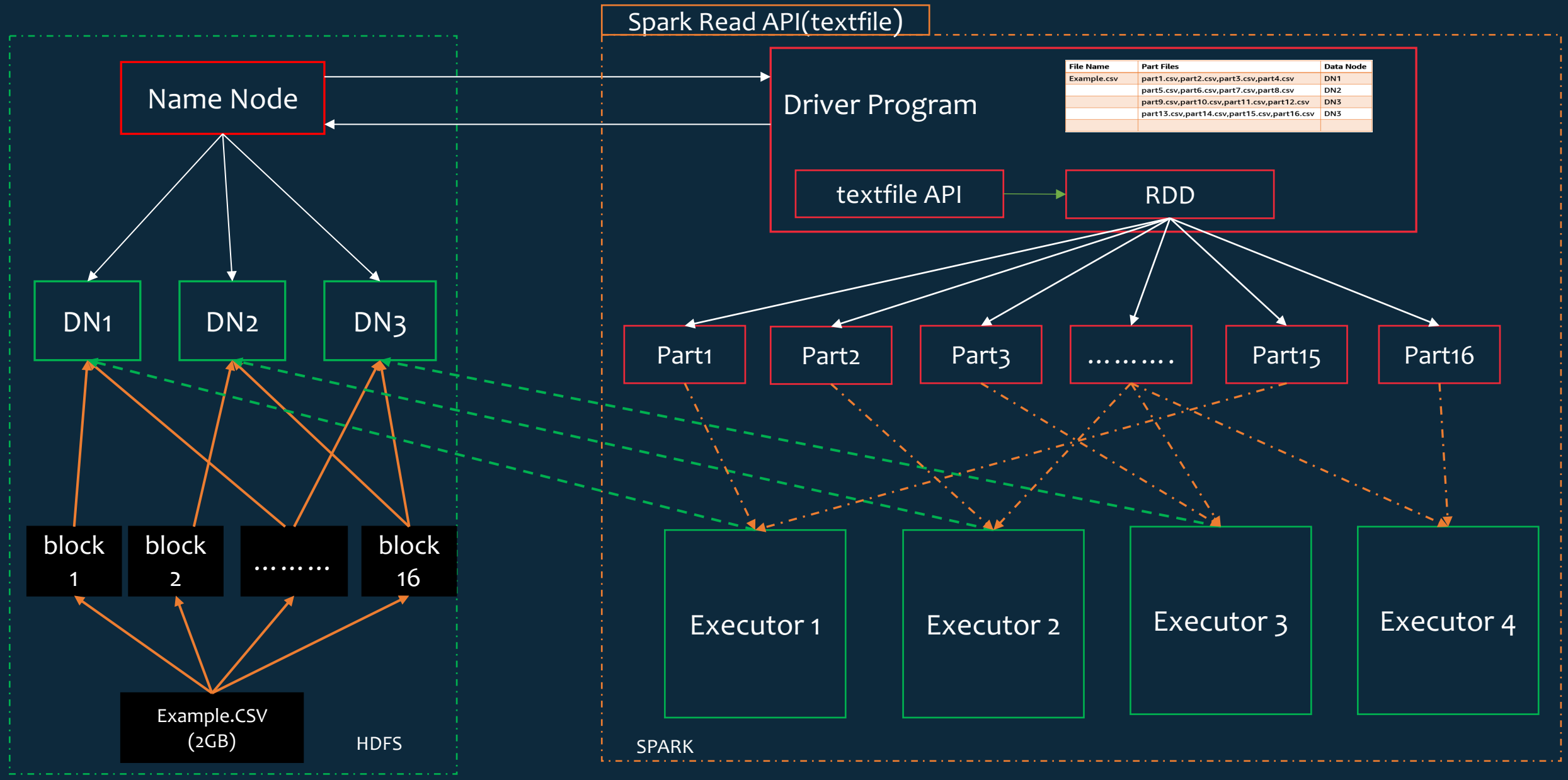
→Data in RDD is split of chunk's and each chunk is a partition. RDD's are highly resilient, Even in case of executor fails it has a ability to re-construct back by looking at the lineage.

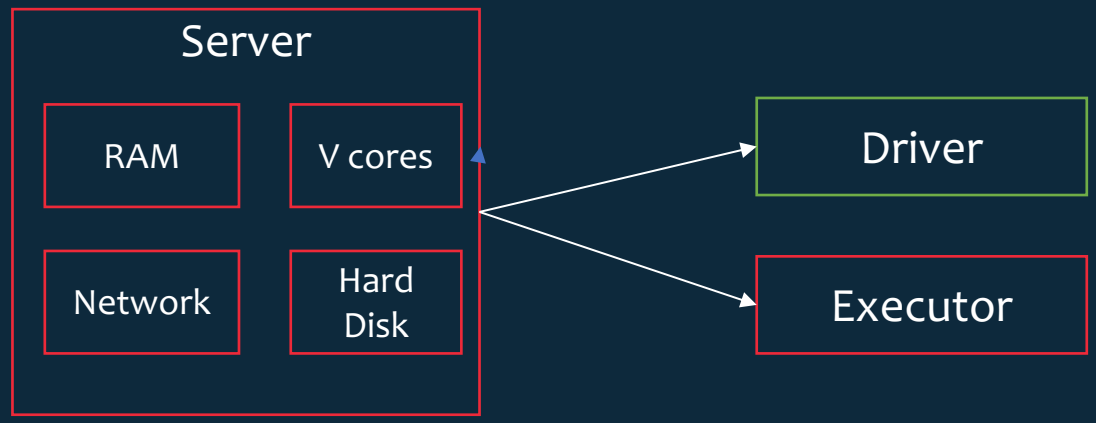
→RDD's are immutable that's means once object is created whose state cannot be changed but can be transformed.



How many partitions each RDD's will have?
Its depends on the source where we are reading the data from.
Let's say if we reading the file from HDFS
Number of HDFS file Blocks = Number of RDD Parititons

RDD(Resilient Distributed Dataset)

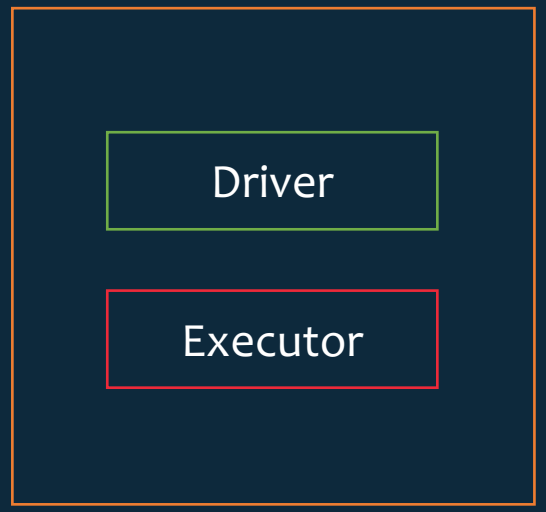




Driver and Executors are the JVM Processes which will be launched in JVM Container . Each JVM container is allocated with some part of RAM, Vcores and Hard Disk in a server.

By default Driver and Executor are allocated with 1GB of the RAM In cluster mode. In local mode driver memory is 1GB(1024MB) and Executor memory is 380Mb .User can change these vales and we will discuss in spark memory management section.

Local Mode



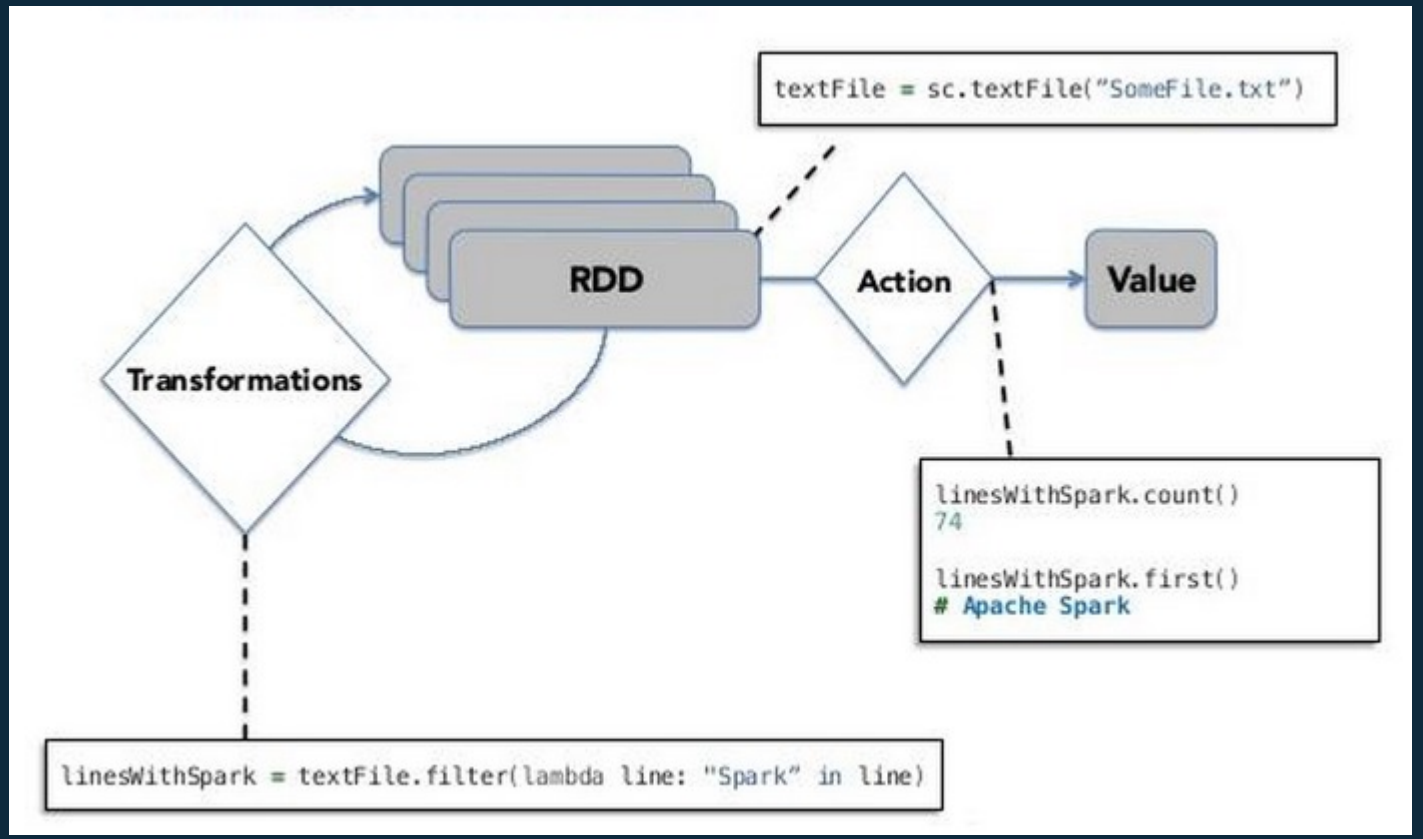
JVM Container

In local mode both driver and executor runs in a same JVM container and you can launch only one executor in local mode.
By Default:
Driver memory = 1GB
Executor Memory is 380Mb

Note: one partition → one task → one core(For every partition a task will be created and gets executed on one Vcore)

Till now we looked at how spark reads the data from HDFS.
From now we will experience the spark in local mode then we will look at cluster mode.

Transformations And Actions



→ Transformations: Takes RDD as an input and returns transformed RDD as an Output.

→ Actions: Takes RDD as an input returns back computed value to the driver.

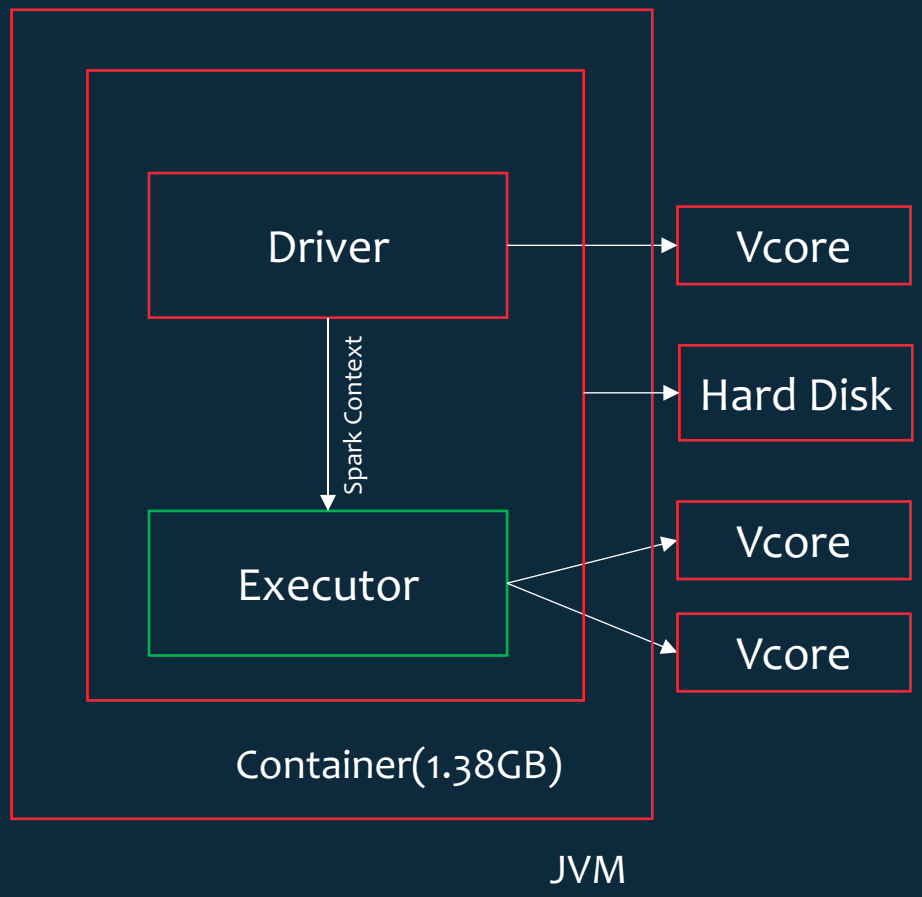
→ Transformations are lazy until and unless an action is performed it will not be evaluated/computed.

https://spark.apache.org/docs/latest/api/scala/org/apache/spark/SparkContext.html#wholeTextFiles(path:String,minPartitions:Int):org.apache.spark.rdd.RDD[(String,String)]

Example1:Word Count

sc = SparkContext("local[2]", "Word Analysis")

SparkContext is a class and its constructor accepts 3 arguments
1) master = local[2]
2) appname = "Word Analysis"
3) Sparkconf = None(Default)



At this line in local the container will created with Driver and Executor process with 1.38 GB of the Memory and 3 Vcores
1 is for Driver and 2 are for Executor. Why only 2? because after local in the square brackets what ever number you see it talks about Vcores.

Example:

local -> 1core
local[2] -> 2Cores
local[*] -> All Available cores.

default.parallelism: defaults to the number of all cores on all machines. The parallelize api has no parent RDD to determine the number of partitions.

defaultMinPartitions: math.min(defaultParallelism, 2). Decides the number of partitions if MinPartitions are not set for Read API's(textfile , wholetextfile). This means if you don't set how many partitions you want at the time of reading the file it will always be 2.

Example:
When master = local, default.parallelism = 1,defaultMinPartitions: 1
When master = local[2], default.parallelism = 2, defaultMinPartitions: 2
When master = local[*], default.parallelism = 8, defaultMinPartitions: 2

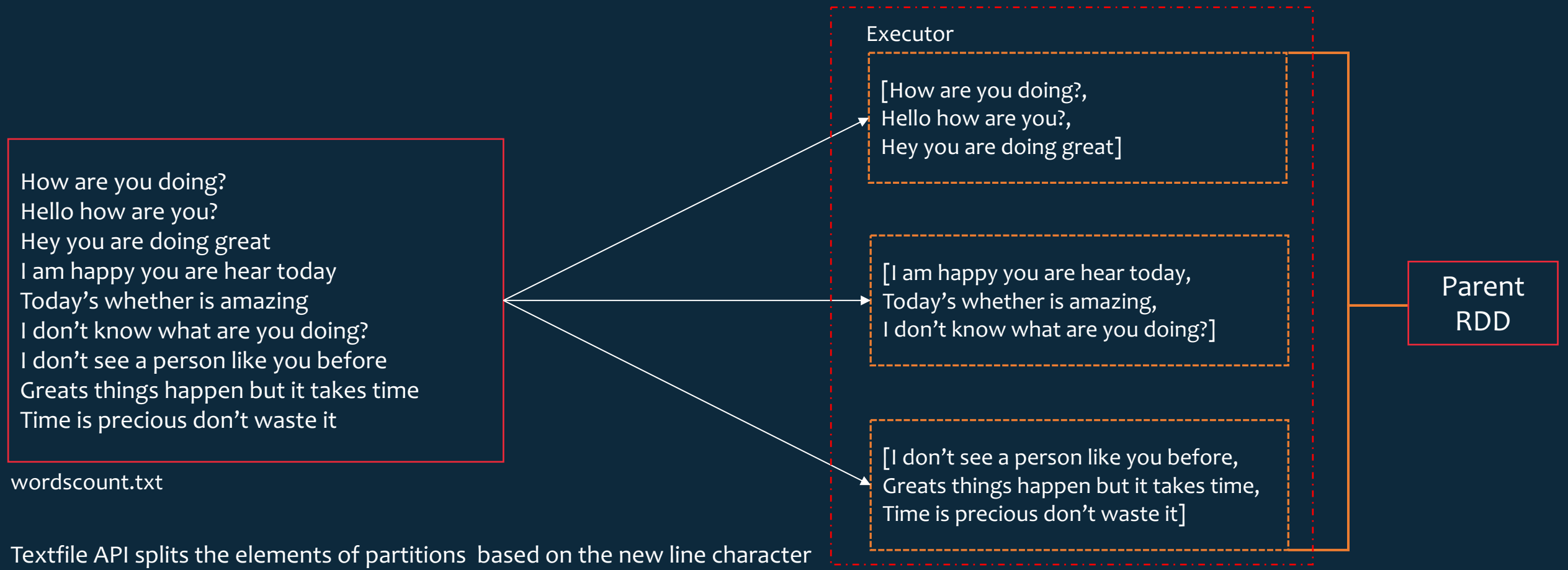
Example1:Word Count

trans1 = sc.textFile("wordscount.txt“, 3)

Arguments
1) Path of the file to process 2) MinPartitions

Type of RDD:

PVM : pyspark.rdd.RDD | JVM : MapPartitionsRDD



Textfile API splits the elements of partitions based on the new line character

Example1:Word Count

trans3 = trans1.map(removeunicode_splitwords)

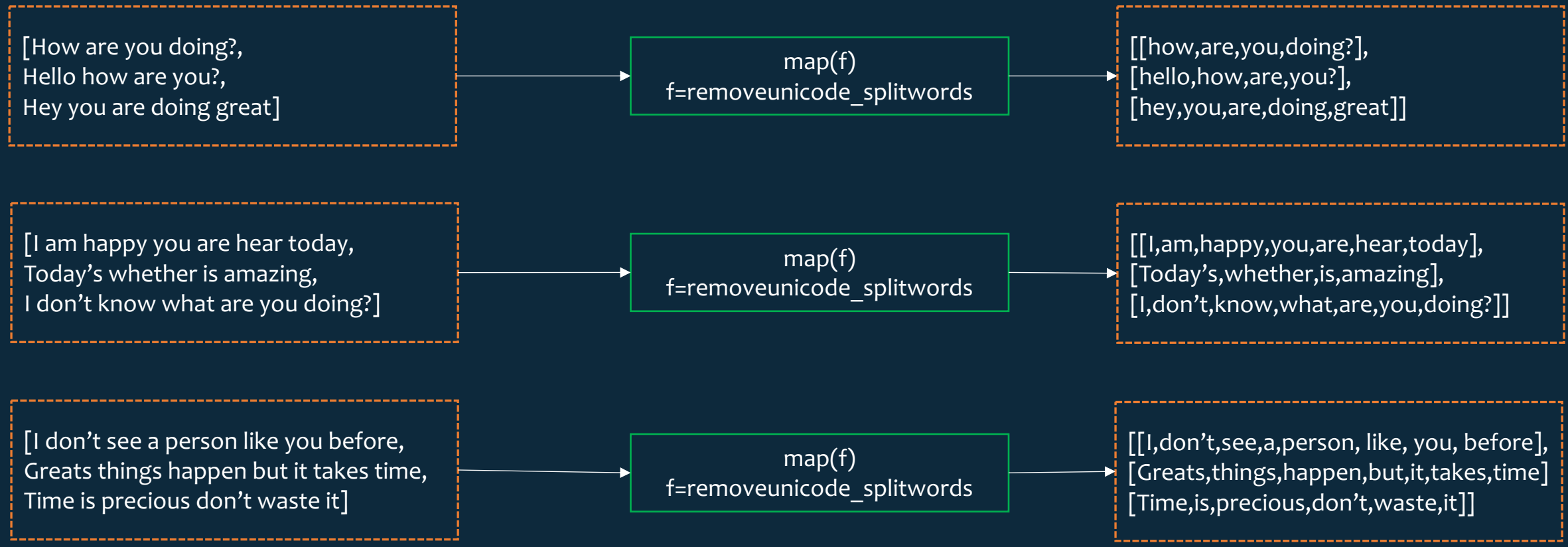
Arguments

1) Python UDF / Lambda Function

Type of RDD:

PVM : pyspark.rdd.PipelinedRDD

JVM : PythonRDD



Note: map is a transformation. it's an one to one mapping means for every element in partitions it will apply function and returns the output for that record.

Note: The UDF of map functions reference should always be define with one parameter

```
def removeunicode_splitwords(x):
    # Removing non-ascii charecters
    ascii_str = str(x).encode("ascii", "ignore").decode()
    return ascii_str.split(" ")
```

[How are you doing?,
Hello how are you?,
Hey you are doing great]

map(f)
f=removeunicode_splitwords

Output_list = []
Element 1 →
x = How are you doing?
ascii_str = How are you doing?
ascii_str.split(“ ”) → Output_list .append([“How”, “are”, “you”, ”doing?”])
Element 2 →
x = Hello how are you?
ascii_str = Hello how are you?
ascii_str.split(“ ”) → Output_list .append([“Hello”, “how”, “are”,“you”, ”doing?”])
Element3 ->
x = “Hey you are doing great”
ascii_str = Hey you are doing great
ascii_str.split(“ ”) → Output_list .append([“Hey”,”you”,”are”,”doing”,great])
return Output_list

[[How,are,you,doing?],
[Hello,how,are,you?],
[Hey,you,are,doing,great]]

Example1:Word Count

trans4 = trans3.flatMap(lambda x:x)

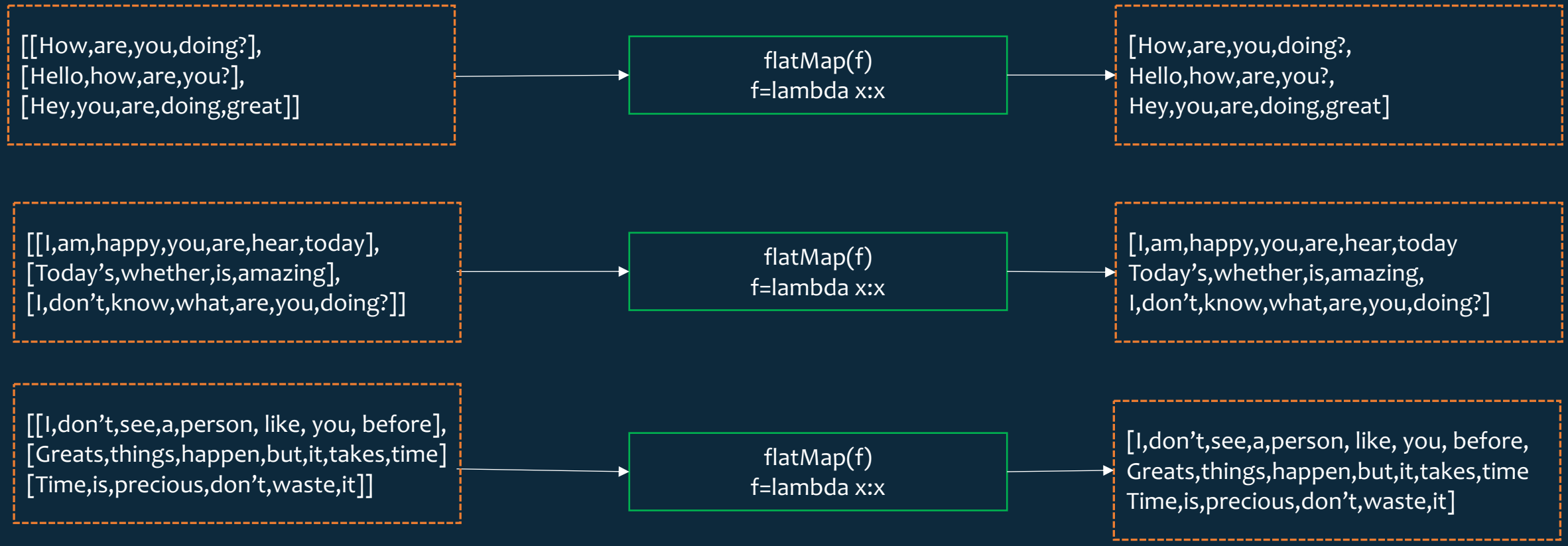
Arguments

1) Python function/ Lambda Function

Type of RDD:

PVM : pyspark.rdd.PipelinedRDD

JVM : PythonRDD



Note: map is a transformation. Input to it is nested RDD elements and apply the function and returns the flattened data structure as the output. In our case It flattened the Nested list into single list.

Note: The UDF of flatmap functions reference should always be define with one parameter

[[How,are,you,doing?],
[Hello,how,are,you?],
[Hey,you,are,doing,great]]

flatMap(f)
f=lambda x:x

[How,are,you,doing?,
Hello,how,are,you?,
Hey,you,are,doing,great]

```
Output_list = []
Element 1 →
                x = [How,are,you,doing?],
                for element in x:
                  Output_list.append(element)

Element 2 →
               x = [Hello,how,are,you?],
                for element in x:
                  Output_list.append(element)
Element3 ->
               x = [Hey,you,are,doing,great]
                for element in x:
                  Output_list.append(element)
return Output_list
```

Example1:Word Count

trans5 = trans4.map(makepairedRDD)

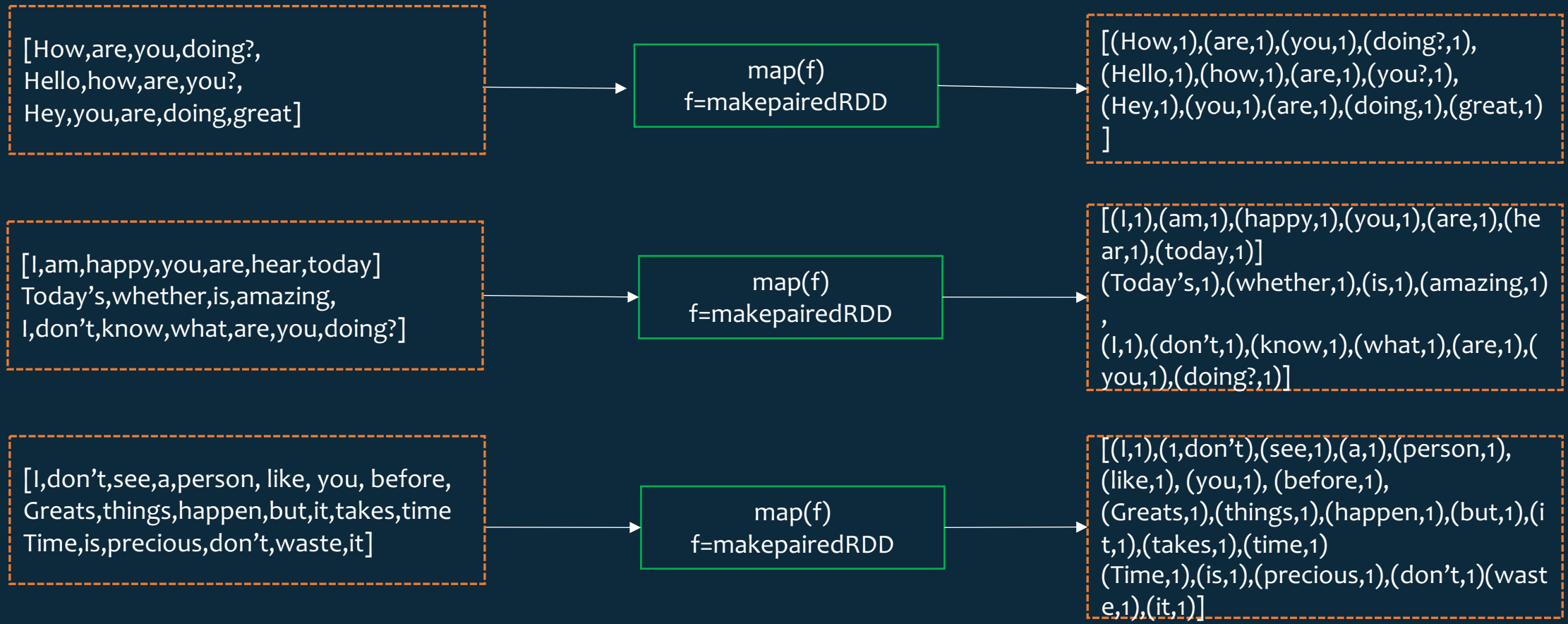
Arguments

1) Python function/ Lambda Function

Type of RDD:

PVM : pyspark.rdd.PipelinedRDD

JVM : PythonRDD



Example1:Word Count

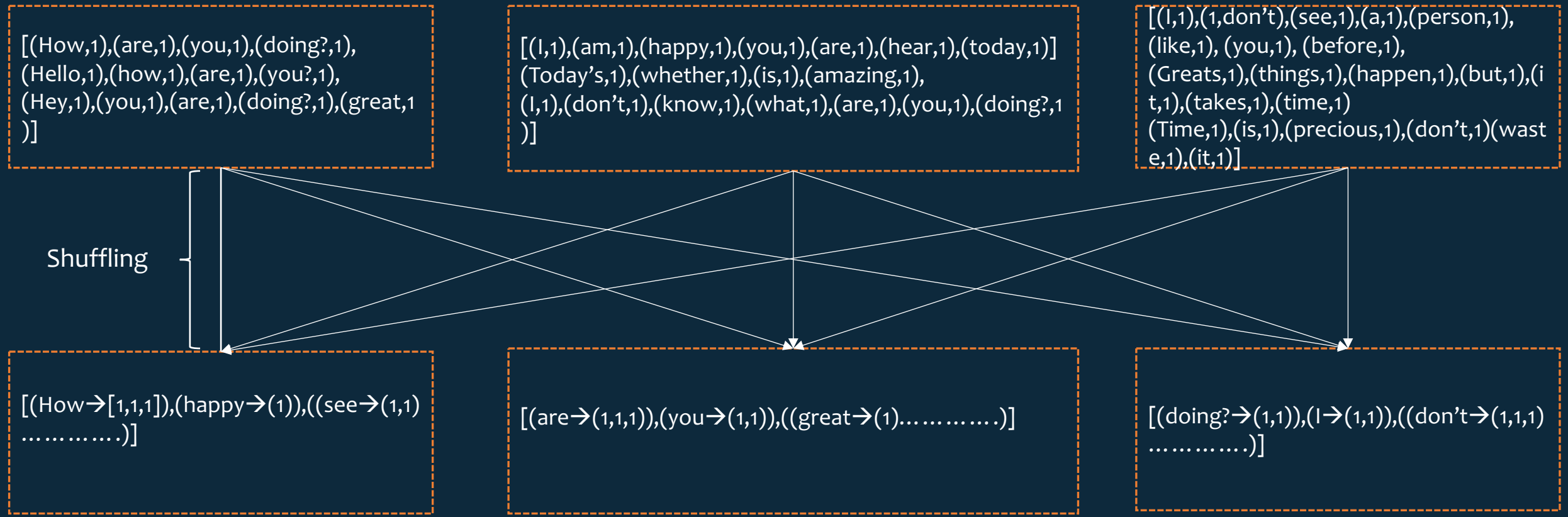
trans6 = trans5.groupByKey()

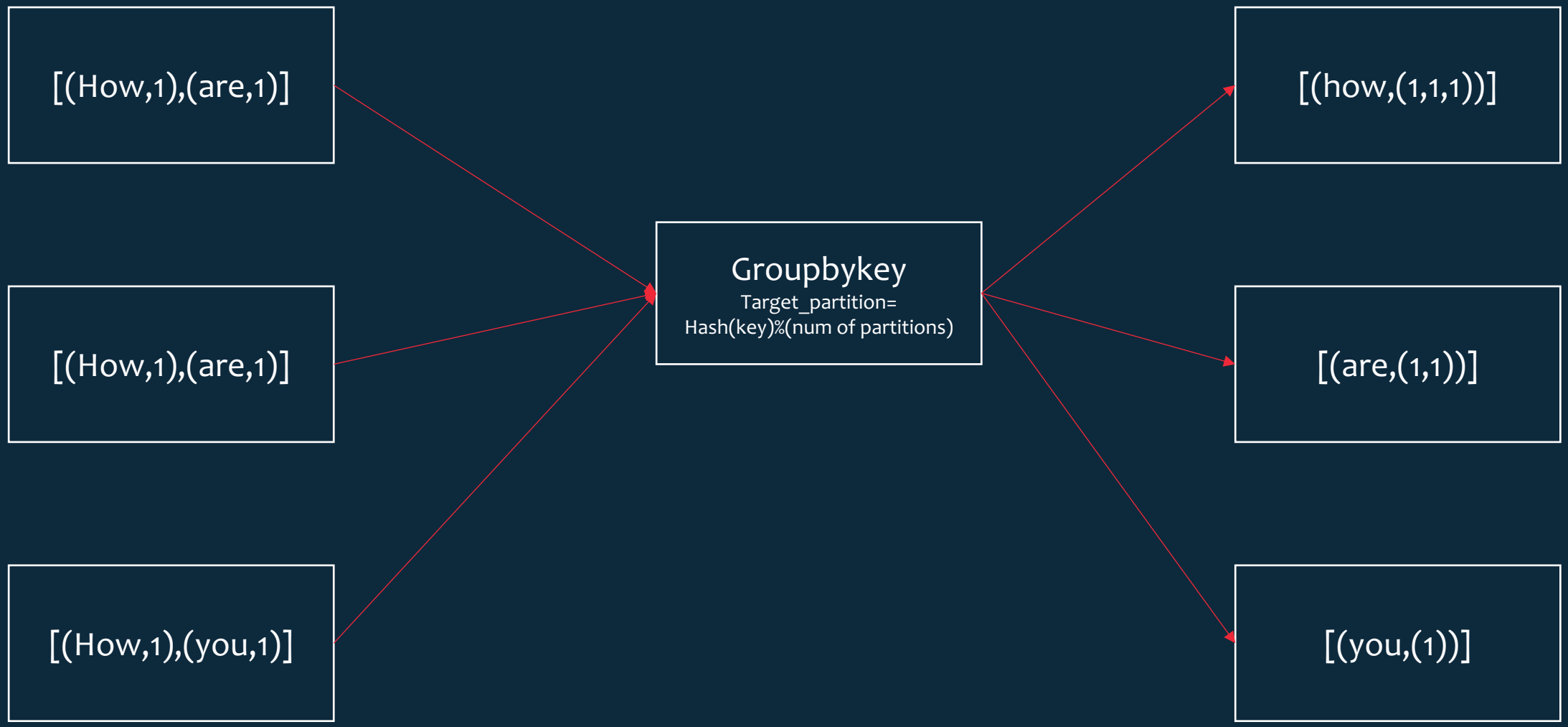
It ensure that all the keys are grouped with there value in one partition.

Type of RDD:

PVM : pyspark.rdd.PipelinedRDD

JVM : PythonRDD



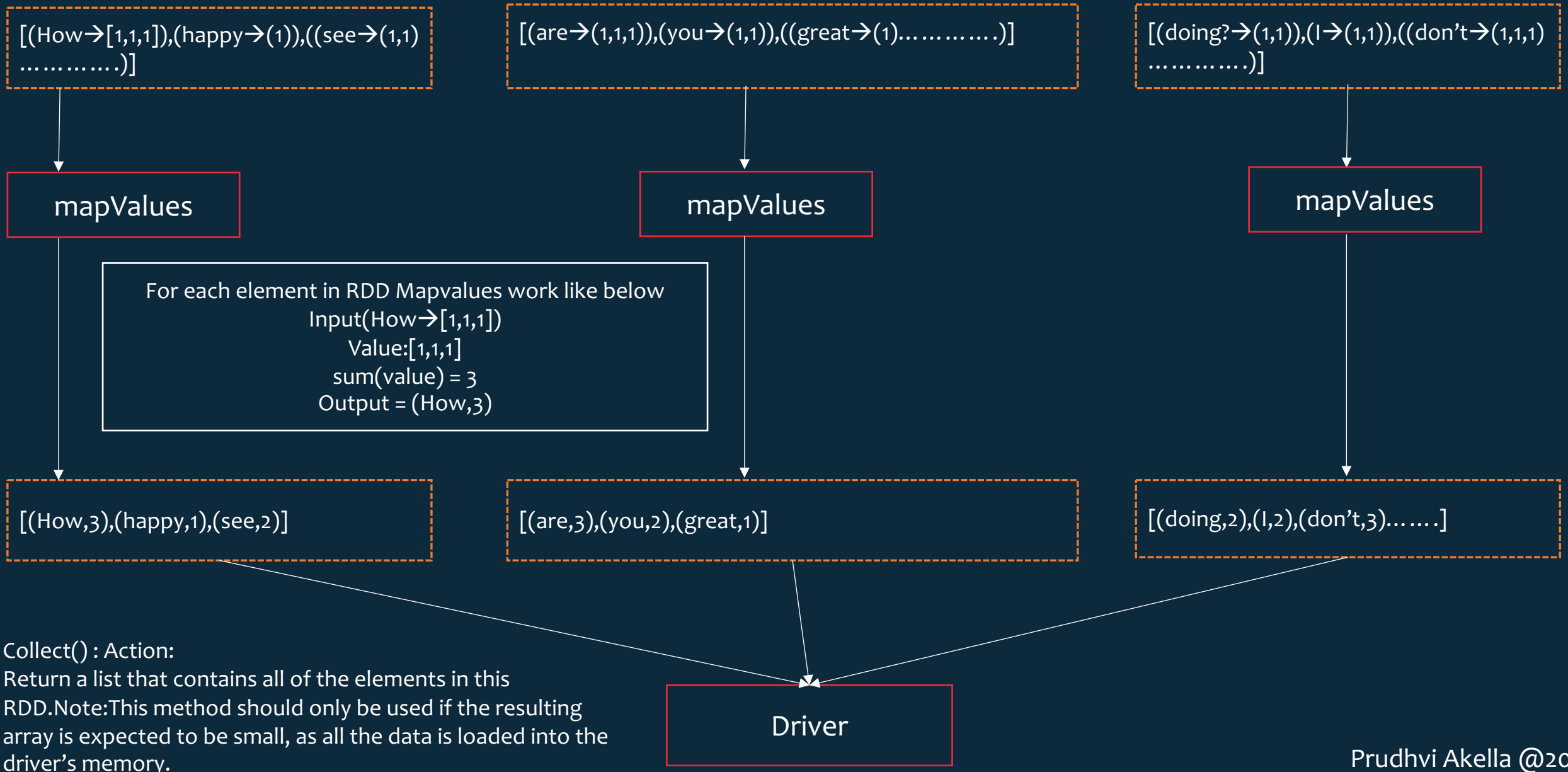

[(How,1),(are,1)]
[(How,1),(are,1)]
[(How,1),(you,1)]
Groupbykey
Target_partition=
Hash(key)%(num of partitions)
[(how,(1,1,1))]
[(are,(1,1))]
[(you,(1))]

Hash and Range Partitioner, Custom Partitioner in Spark.

Join or Any sort of key based transformation

Hash Partitioner

hash(key) % number of partition => to which partition this key should shuffle to.



Collect() : Action:
Return a list that contains all of the elements in this RDD.Note:This method should only be used if the resulting array is expected to be small, as all the data is loaded into the driver's memory.

map(makepairedRDD) → makepairedRDD function pairs each word with 1. where the output for each RDD element is going to be Tupple2[(String, Int)]

For Example :
How, are, you, doing → (How,1),(are,1),(you,1),(doing?,1)

If you closely observe the output of each element (How,1) has two things (key, value) where key = "How" value = 1

→ output of map(makepairedRDD) can also be called as pariedRDD.

Why PariedRDD's are required?

If you want to perform any aggregations based on key then spark provides some key based API's/ transformations on RDD and they expect PariedRDD as an input. These transformation are also called as wide transformations and these will lead to shuffling phase.

Examples of pariedRDD.
GroupbyKey
ReducebyKey

Example1:Word Count

trans7 = trans5.reduceByKey(lambda x,y:x+y)

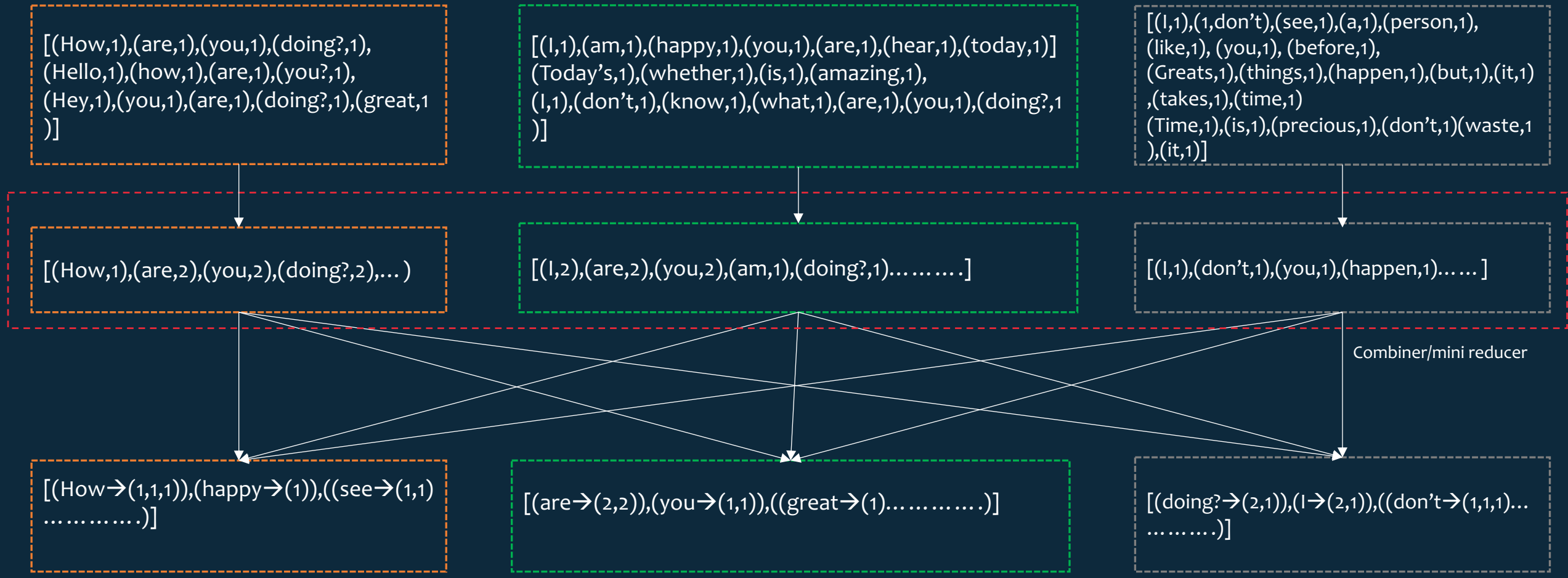
Arguments :

Python function which takes two inputs or lambda with two inputs

Type of RDD:

PVM : pyspark.rdd.PipelinedRDD

JVM : PythonRDD



(how ->(1,1,1,1,1))

a = 1, b = 1 => 2

a = 2, b = 1 => 3

a = 3, b = 1 => 4

a = 4, b = 1 => 5

(how,5)

Example1:Word Count

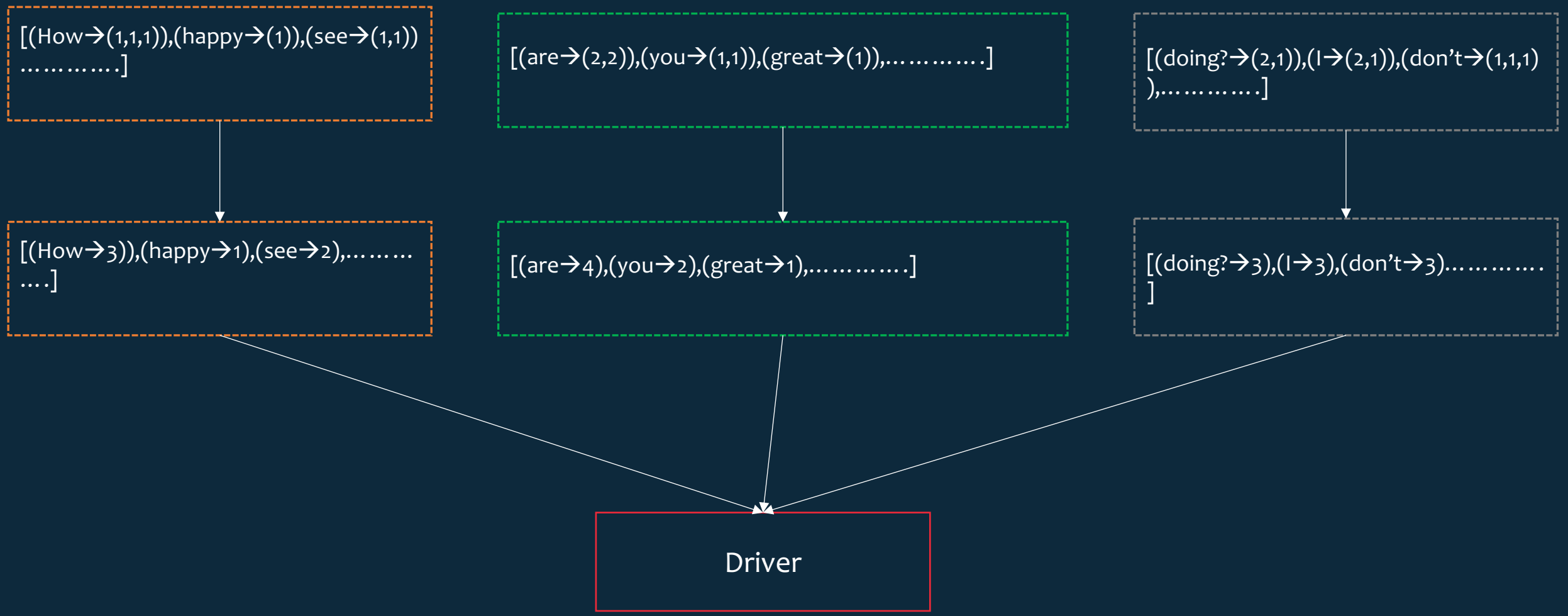
trans7.take(100)

Take is an action which will collect first 100 records from the partitions and sent back to driver

Type of RDD:

PVM : Respective Values type

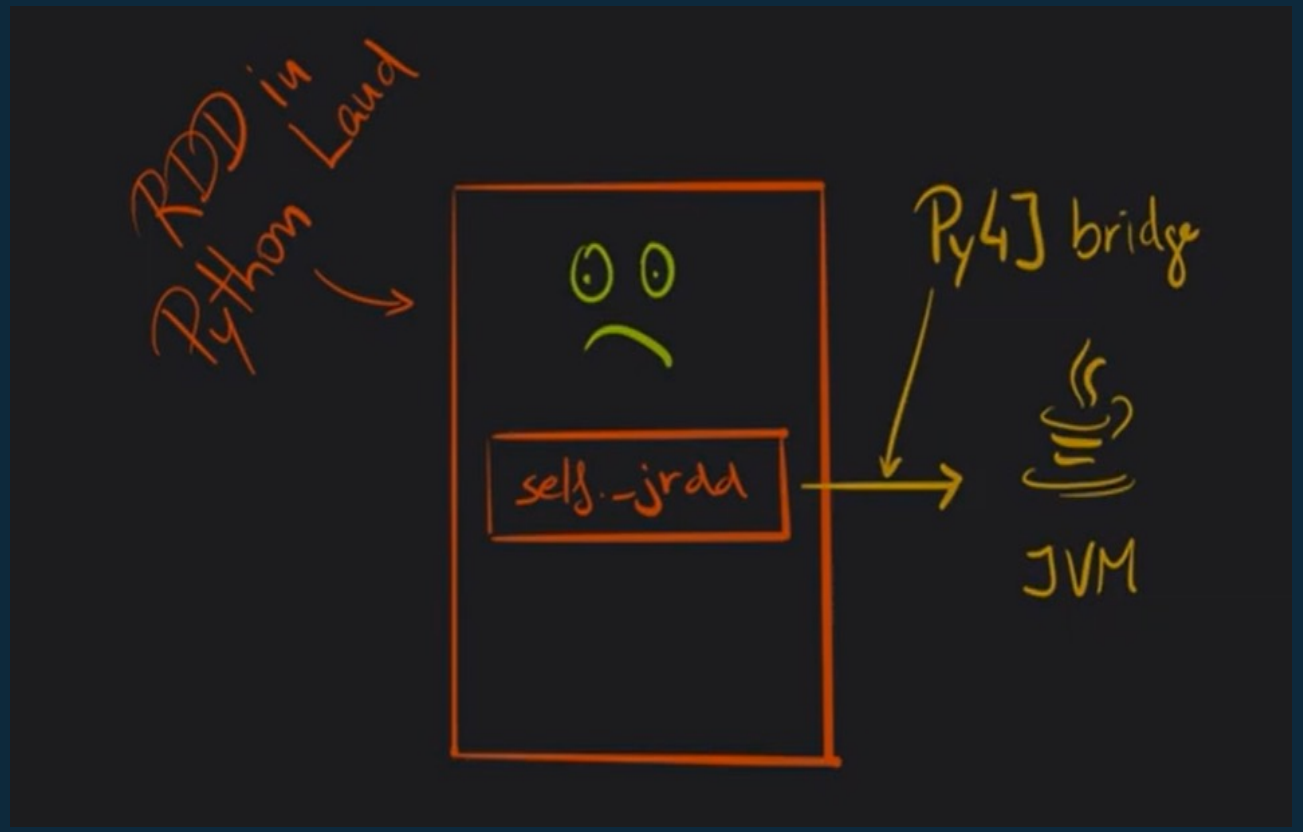
JVM : Respective Values type



Internals of RDD in PySpark

```
# Connect to the gateway
gateway = JavaGateway(
    gateway_parameters=GatewayParameters(
        port=gateway_port,
        auth_token=gateway_secret,
        auto_convert=True))

# Import the classes used by PySpark
java_import(gateway.jvm, "org.apache.spark.SparkConf")
java_import(gateway.jvm, "org.apache.spark.api.java.*")
java_import(gateway.jvm, "org.apache.spark.api.python.*")
.
.
.
return gateway
```

Internals of RDD in PySpark



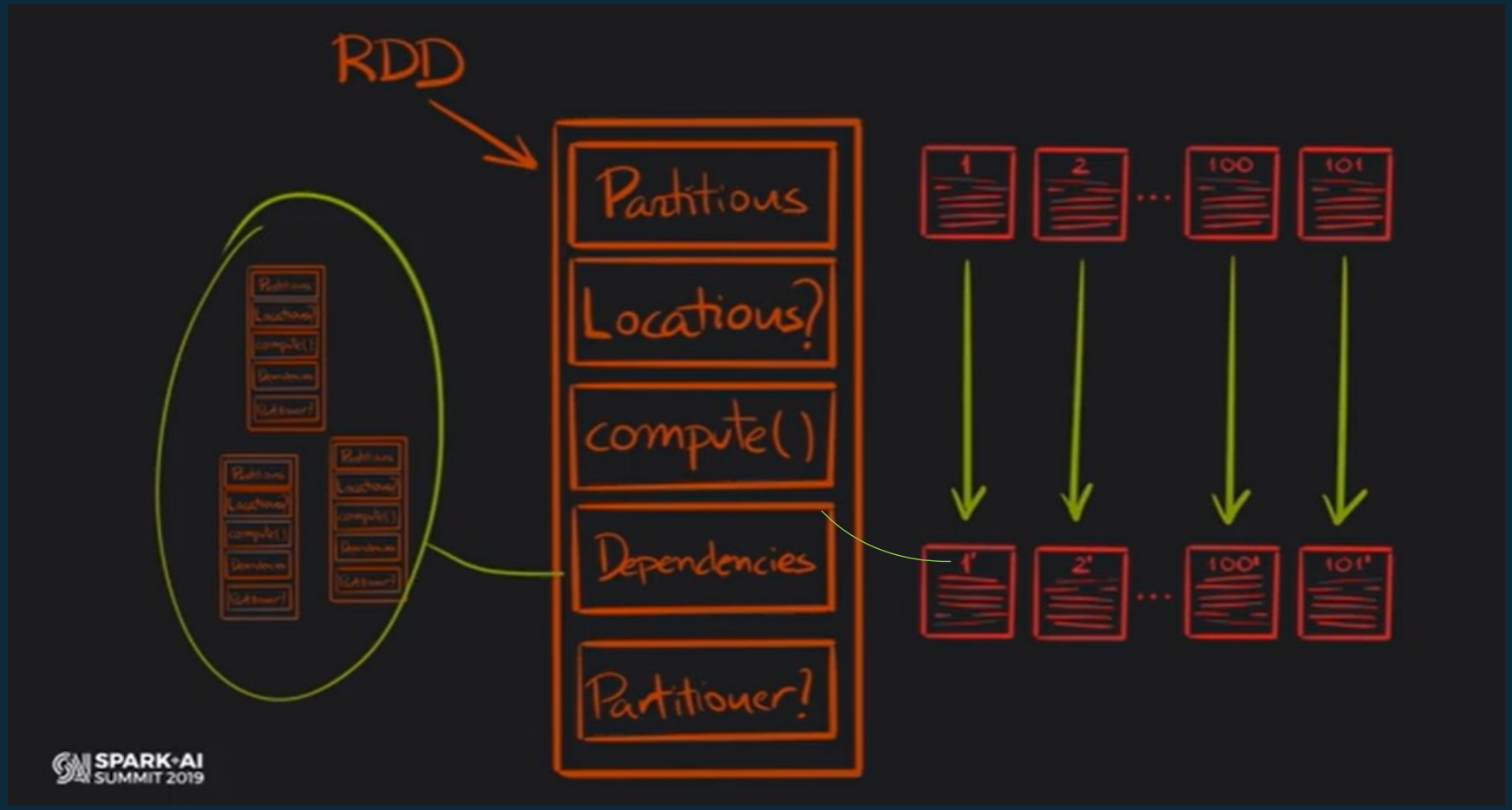
If you clearly observe RDD/Pipelined RDD class in python has a private member called _jrdd which holds the reference of the RDD object that is created in JVM.

Note:

This means every time you create an RDD in Python using Pyspark API's it will create an respective RDD objects in JVM and the reference of that JVM's object will be stored into _jrdd of PVM's RDD object.

Internals of RDD in PySpark

Important Properties of RDD(RDD is a class)

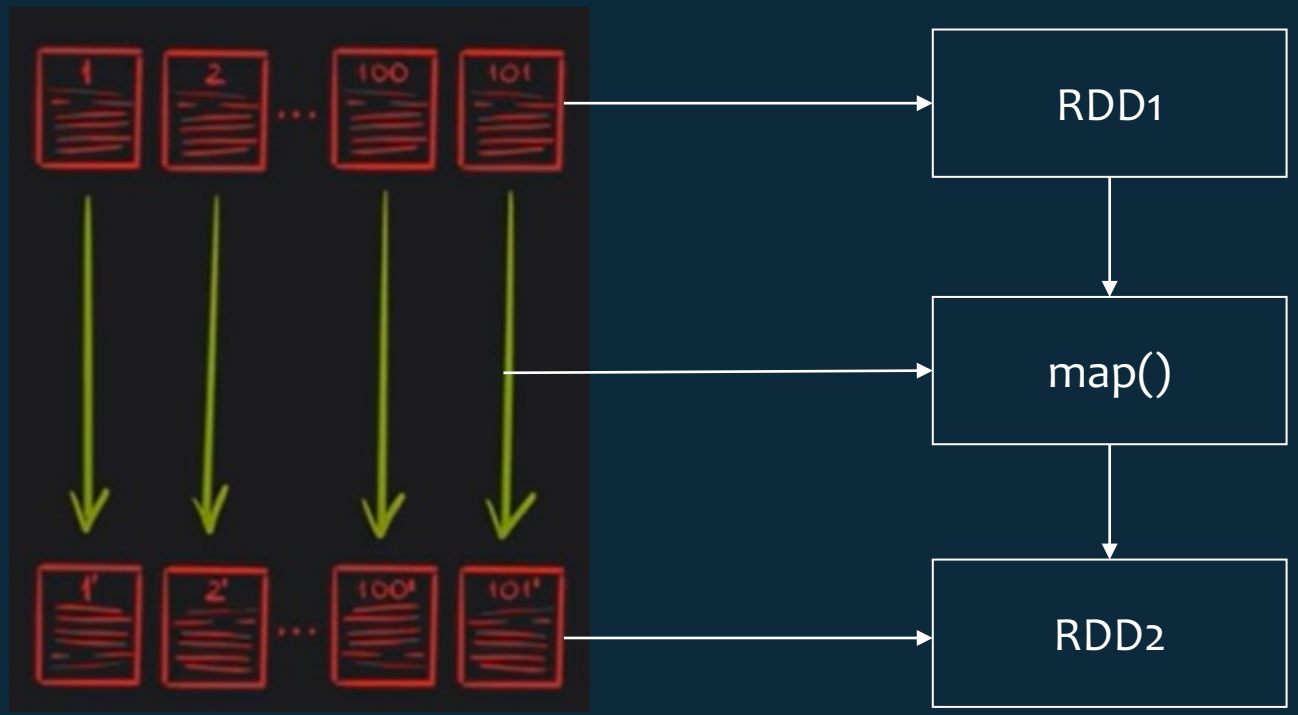


Internals of RDD in PySpark

Partitions: We already discussed about this

Locations: Spark decides the location of the executor based on the data locality. So that its easy to load the data into executor.

compute() : Every RDD has a compute method. Once this method is called computations(transformation/Actions)(map) will applied to the all the partitions of that RDD (RDD1). in executors and creates the new partitions as output of a new RDD(RDD2).



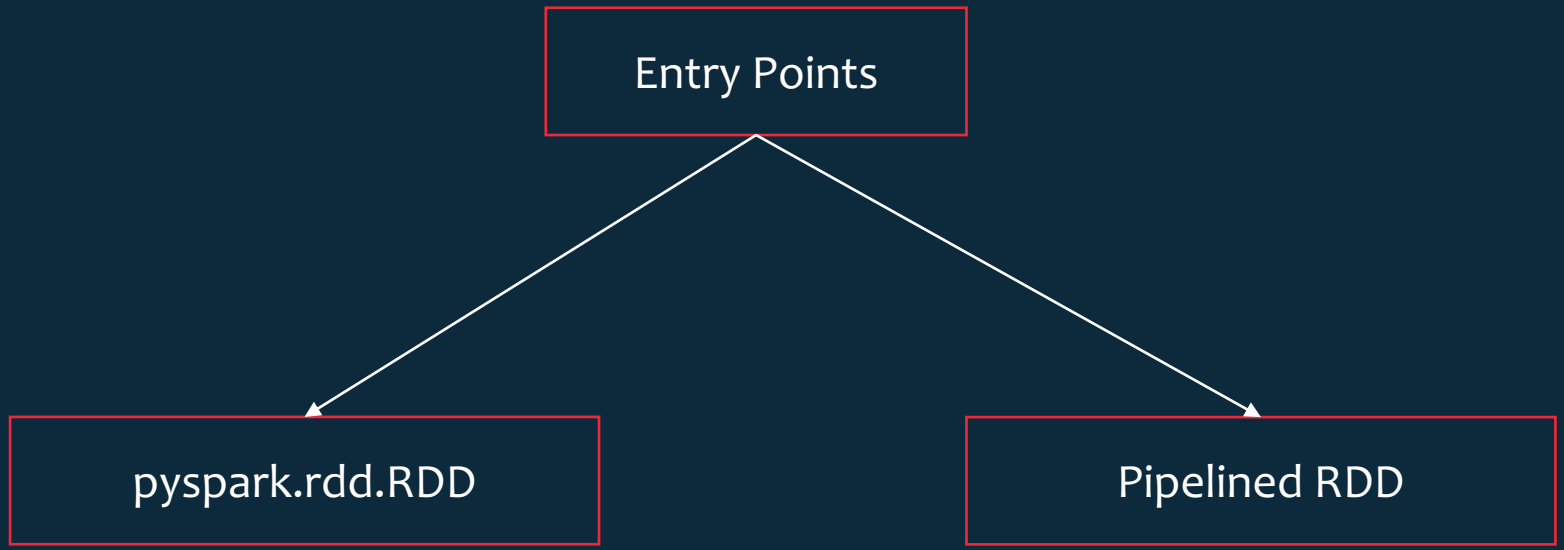
Important Properties of RDD

Dependencies: This is the key point of RDD when one of the partition disappears because the machine that was holding it disappears because of the internet being last. you can recompute only the partitions because you can trace back through compute and the dependence's where this coming from so eventually you can go back to the first original data source and compute just only that. Its more over less talking about the RDD lineage.

Partitioner : Its used to distribute the partitions across the different machine. This plays a very important role with wide transformations.

Internals of RDD in PySpark

RDD Entry Points in PySpark

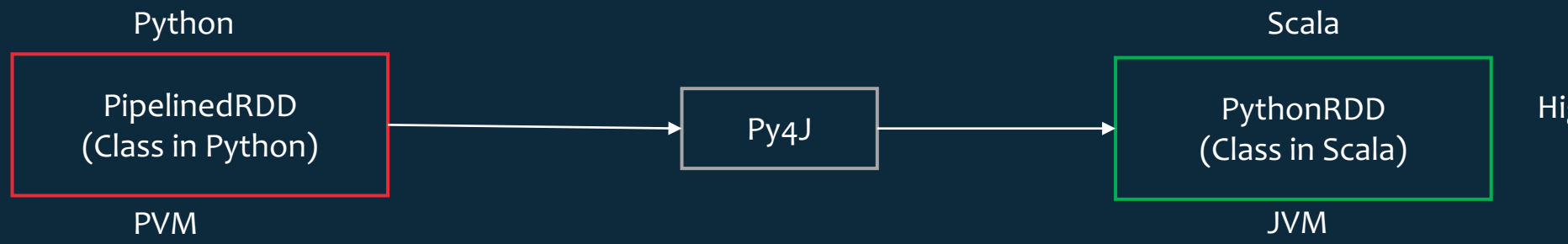


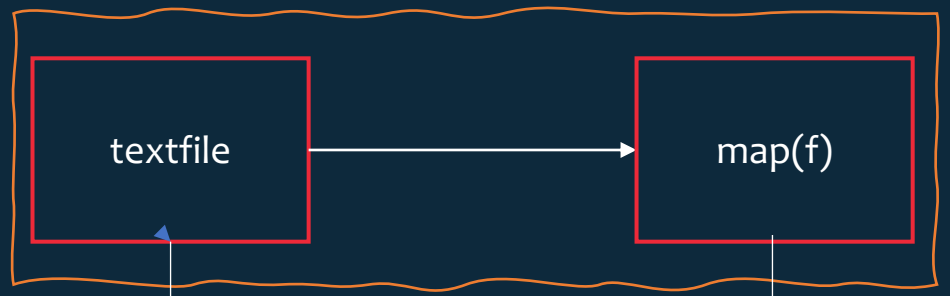
RDD object will be created in PVM when you read the data into Spark using PySpark Read API(textfile, wholeTextfile).

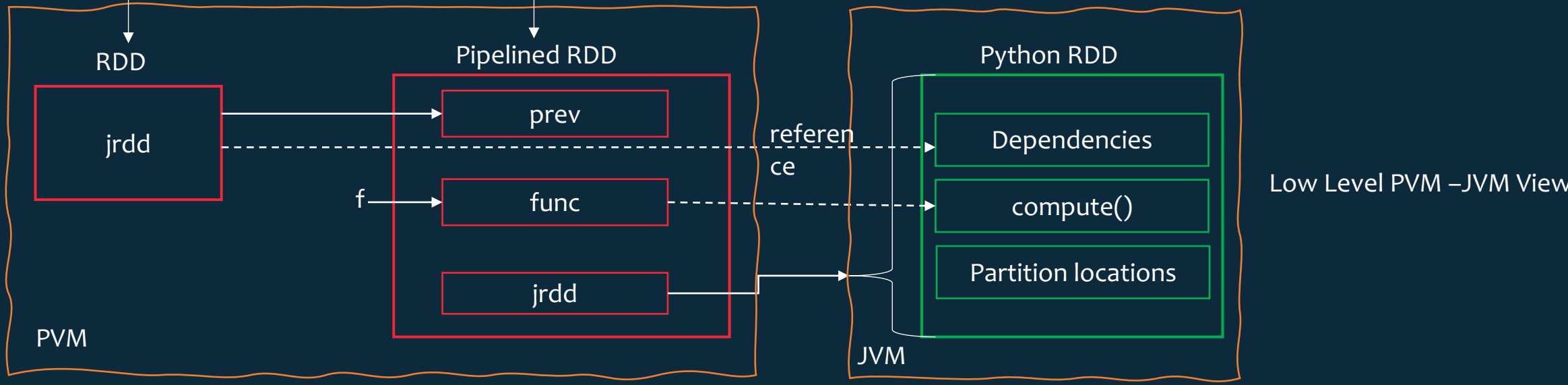
This the main RDD which does all the Work. When ever you try to map the partitions this RDD will be created.

Which create PythonRDD object in JVM(Scala Side) and the reference of it is stored into _jrdd of Pipelined RDD in PVM

Internals of RDD in PySpark







Internals of RDD in PySpark

Lets understand by an example

```
trans1 = sc.textFile("C:/MyStuff/Systemdesign_Expert/Datasets/covid/wordscount.txt")
type(trans1)
```

```
pyspark.rdd.RDD
```

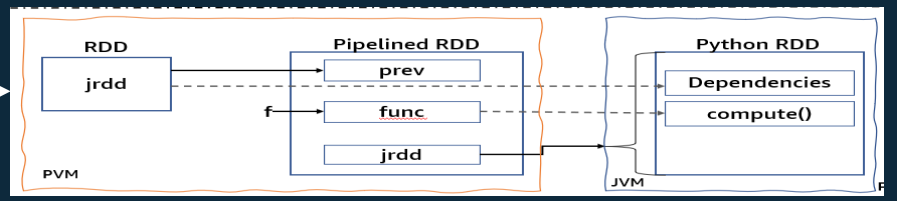
textFile API returns RDD Object which is created in PVM and its equivalent HadoopRDD is created in JVM. The reference of HadoopRDD object is stored in _jrdd instance variable of RDD object that is created in the PVM

Lets assume jrdd = PX0101

```
def removeunicode_splitwords(x):
    ascii_str = str(x).encode("ascii", "ignore").decode()
    return ascii_str.split(" ")
```

```
trans3 = trans1.map(removeunicode_splitwords)
type(trans3)
```

```
pyspark.rdd.PipelinedRDD
```



Map transformation will create PipelinedRDD in PVM and its respective PythonRDD object in JVM and reference of it is stored in _jrdd variable of PipelinedRDD that is created in PVM.

Pipelined RDD (PVM Side): It consists of below instance variables

→ prev : it's a pointer/instance variable in the pipelinedRDD object which holds the address of the previous RDD in this case PX0101

→ func : map(f). Where f = removeunicode_splitword

→ jrdd: PythonRDD objects reference that is created in JVM say in our case AX0102

PythonRDD (JVM Side): It consists of below instance variables

1) Dependencies: Previous RDD's Address in this case RDD._jrdd address

2) compute() : is magic method it executes map function(in this case removeunicode_splitwords function) by creating executors

Internals of RDD in PySpark

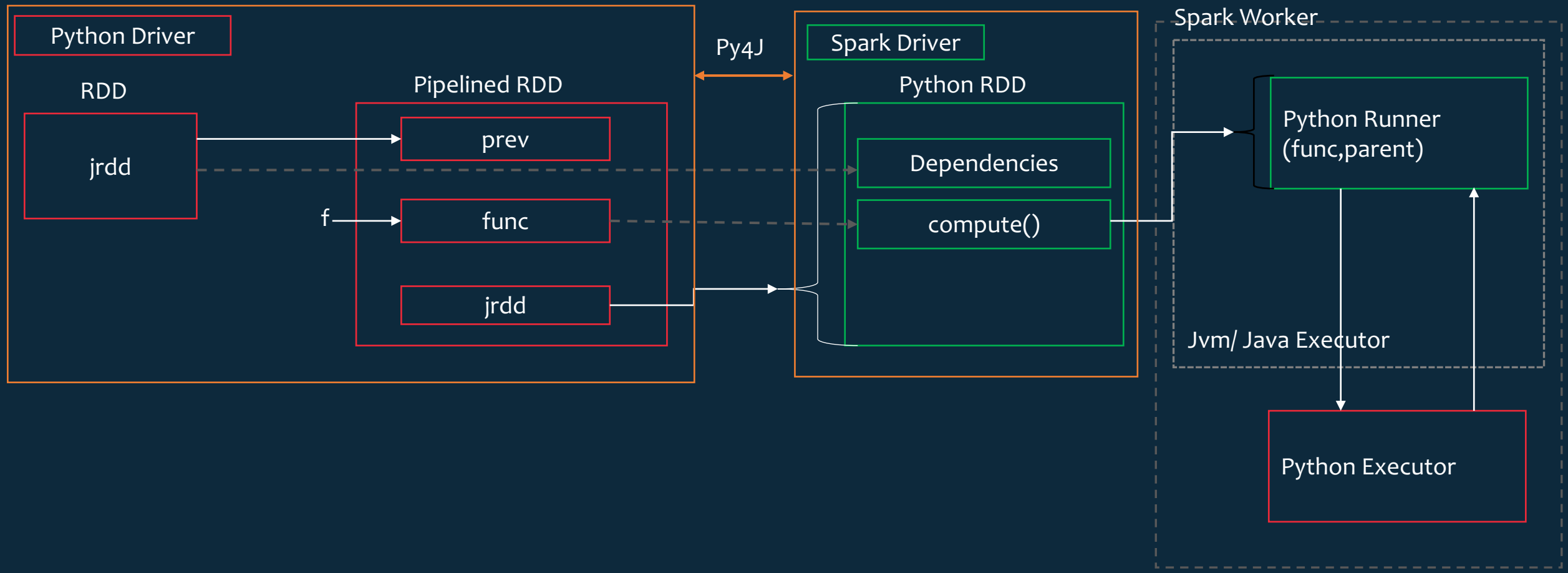
Is it possible to execute **removeunicode_splitword (**which consist of python code) in JVM ?

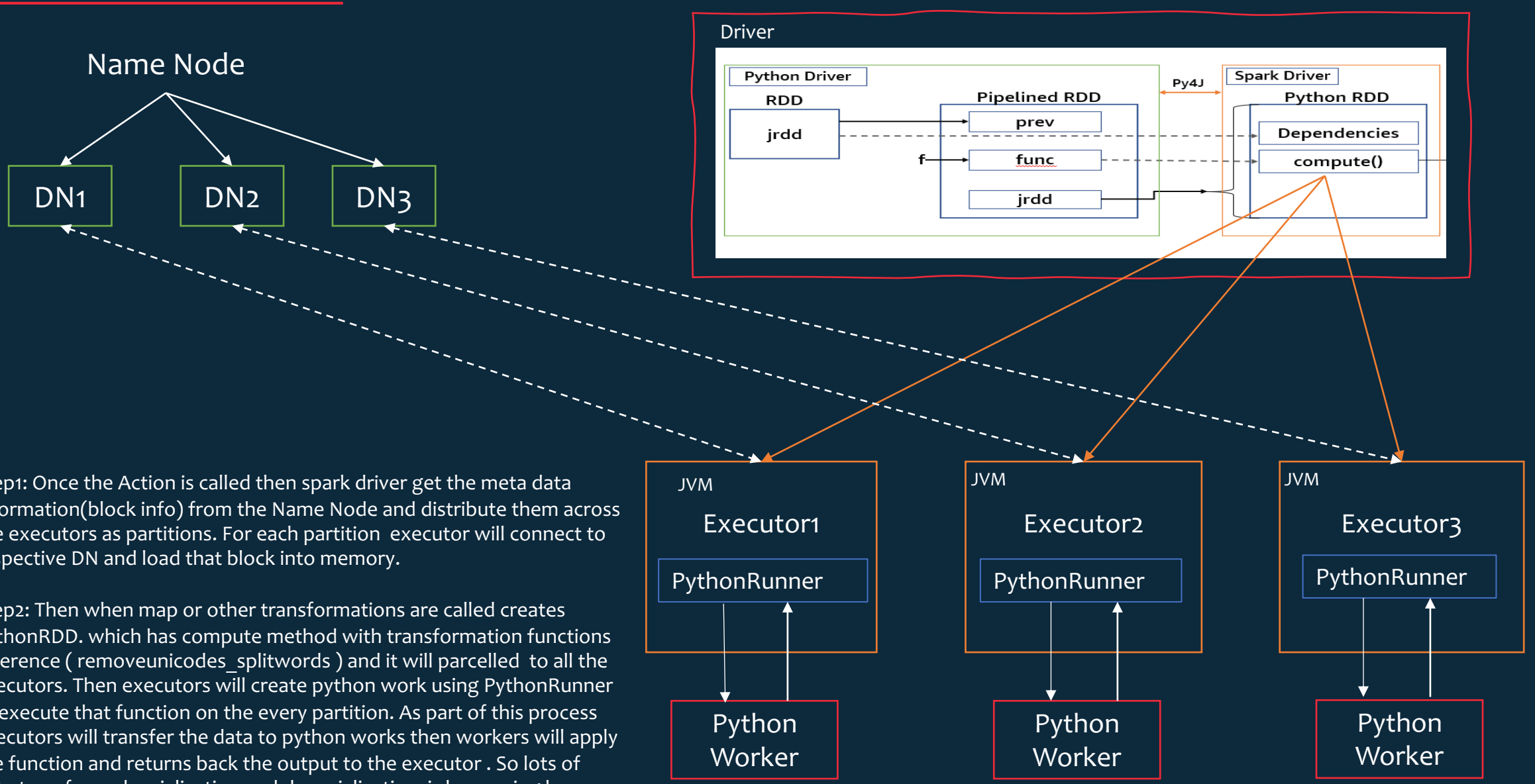
compute() is the Magic

compute() runs on each EXECUTOR and start PYTHON WORKER VIA PythonRunner.

Why python worker is required?

As f in map(f)[map(removeunicodes_splitwords)] is an python function/lambda which cannot be executed in executor which is a JVM process so it launches python worker which is an python processes



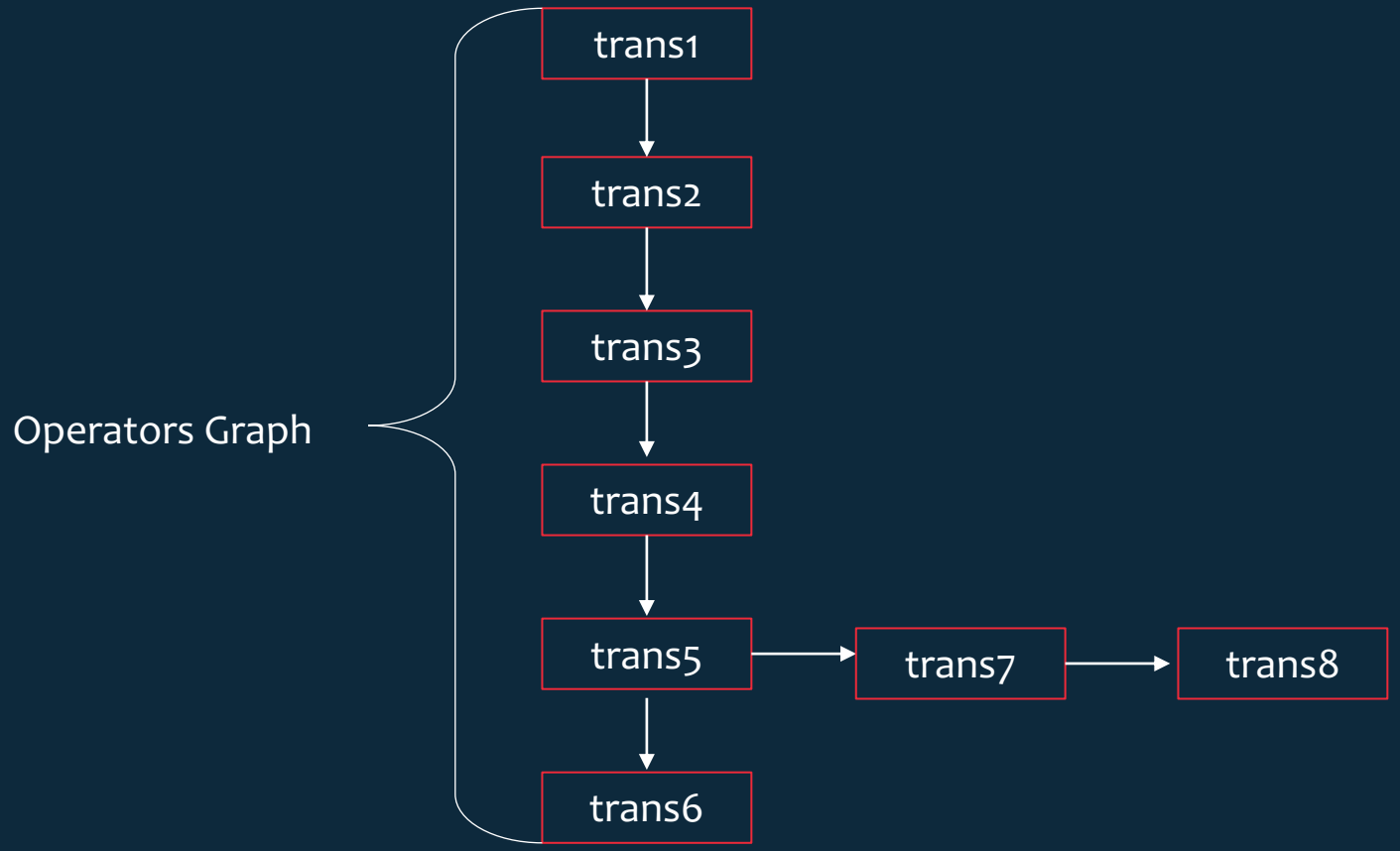


Step1: Once the Action is called then spark driver get the meta data information(block info) from the Name Node and distribute them across the executors as partitions. For each partition executor will connect to respective DN and load that block into memory.

Step2: Then when map or other transformations are called creates PythonRDD. which has compute method with transformation functions reference (removeunicodes_splitwords) and it will parcelled to all the executors. Then executors will create python work using PythonRunner to execute that function on the every partition. As part of this process executors will transfer the data to python works then workers will apply the function and returns back the output to the executor . So lots of data transfer and serialization and de-serialization is happening here.

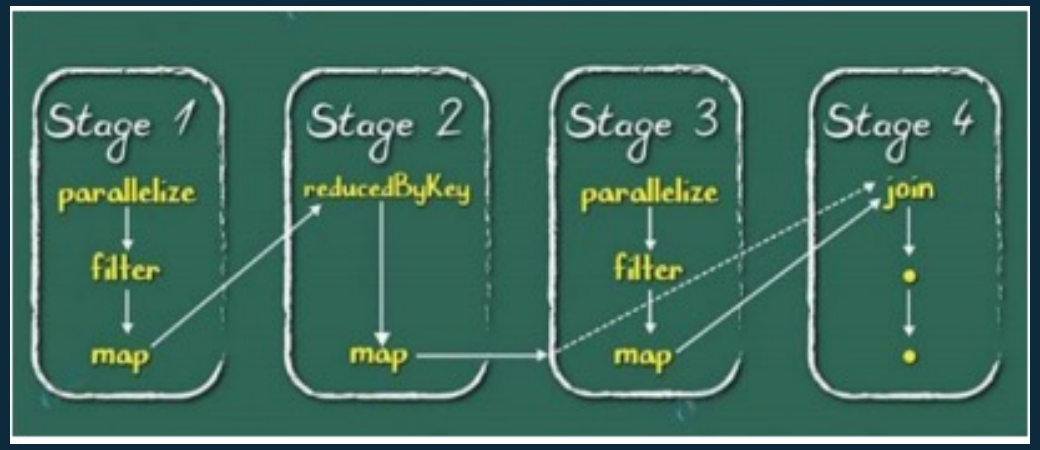
Operators Graph/ RDD Lineage

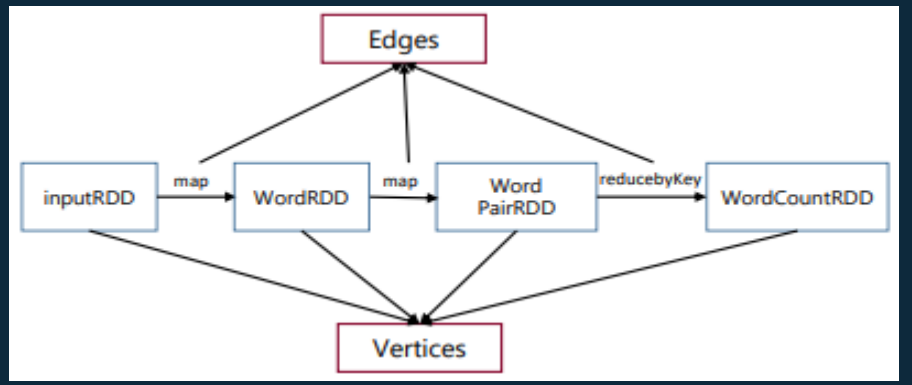
- As we discussed Spark transformations are lazy (means spark will not execute transformation until a action is performed) .
- Instead each RDD maintains a pointer to one or more parent RDD's along with metadata about what type of relationship it has with the Parent RDD. Its called a Lineage.
- Lineage is created for each transformation.
- A Lineage will keep track of what all transformations has to be applied on the RDD, including the location from where it has to read the data.
- RDD lineage is used to re compute the data if there any faults as it contains the pattern of the computation.



DAG(Direct Acyclic Graph)

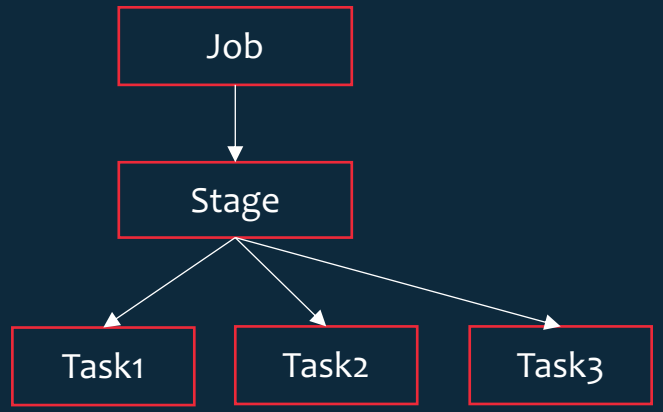
→ (Directed Acyclic Graph) DAG in Apache Spark is a set of Vertices and Edges
→ Vertices represent the RDDs and the edges represent the Operation to be applied on RDD.
→ In Spark DAG, every edge directs from earlier to later in the sequence. On the calling of Action, the created DAG submits to DAG Scheduler which further splits the graph into the stages of the task.



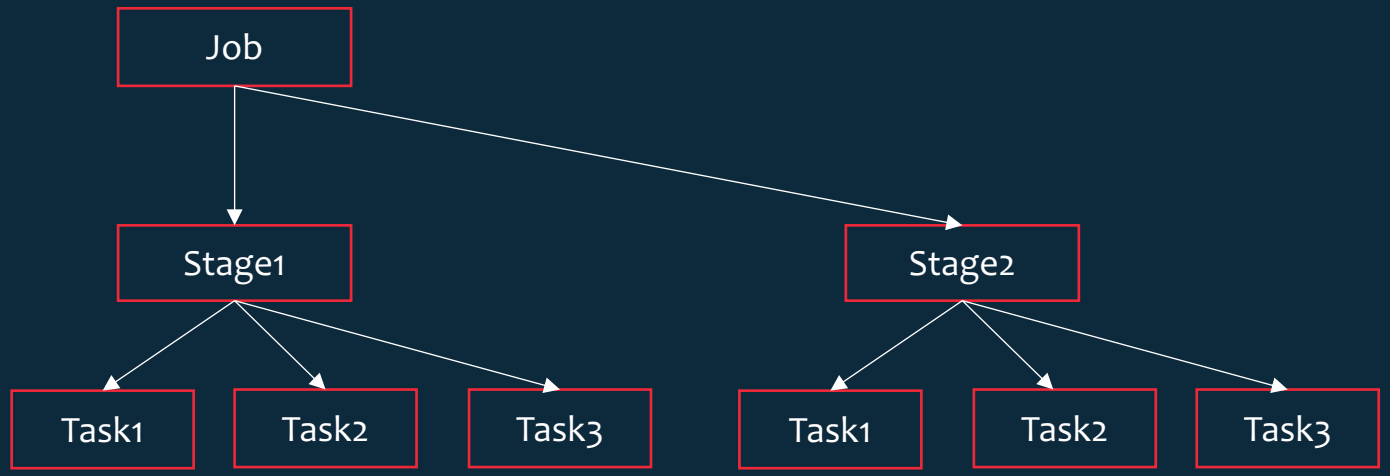


DAG(Direct Acyclic Graph)

→ For every action one new job will be created by spark application.
→Job is further split into stages based on the transformations(For every wide transformation one new stage will be added to DAG)
→ Each stage is further split into tasks



Narrow Transformation



Wide Transformation

Action

Jobs

Stages

Tasks = cores
1 Task = 1 Partition = 1 Core

DAG(SPARK UI View)

Job View

Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at Directory_WordCount.scala:22	2020/01/29 04:39:43	0.1 s	0/2	0/2

Detail Stage View

Stages for All Jobs

Active Stages: 1
Pending Stages: 1

Active Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0	map at Directory_WordCount.scala:18 (kill) +details	2020/01/29 04:42:10	0.5 s	0/1				

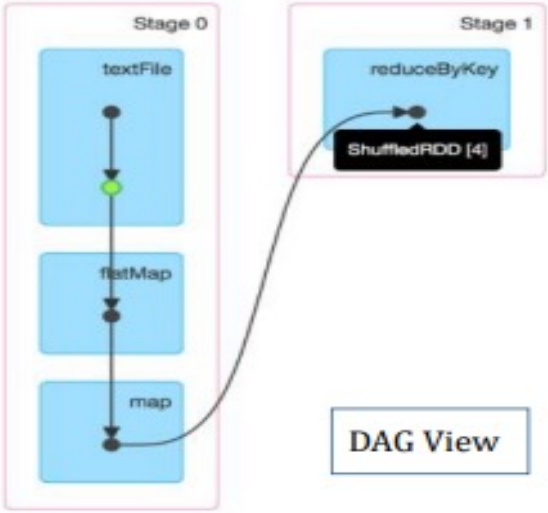
Pending Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	collect at Directory_WordCount.scala:22 +details	Unknown	Unknown	0/1				

Details for Job 0

Status: SUCCEEDED
Completed Stages: 2

▸ Event Timeline
▾ DAG Visualization



DAG View

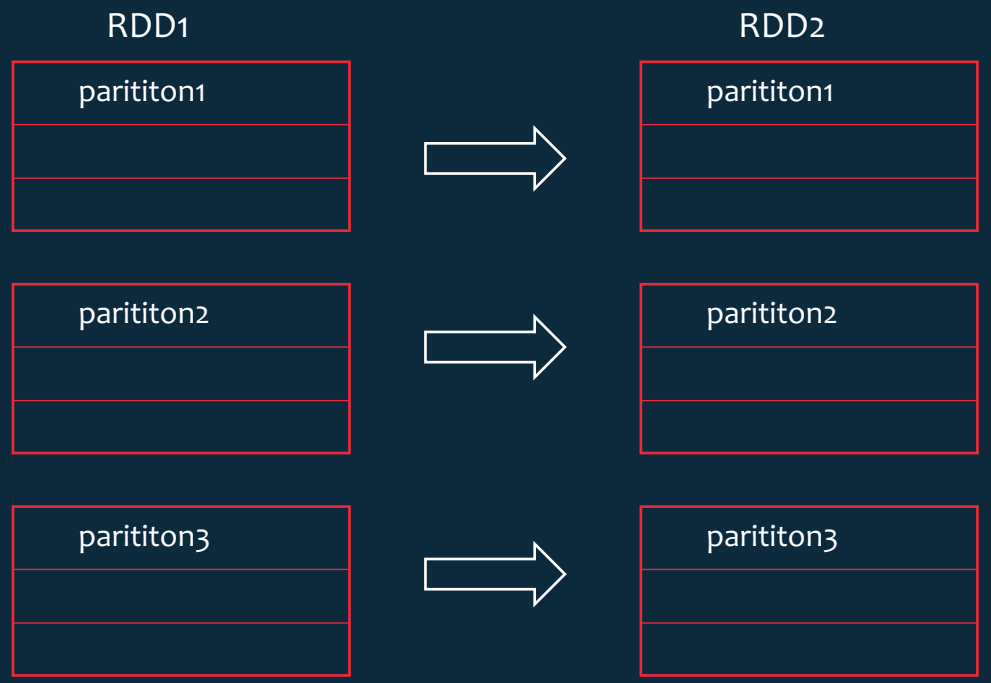
Transformations

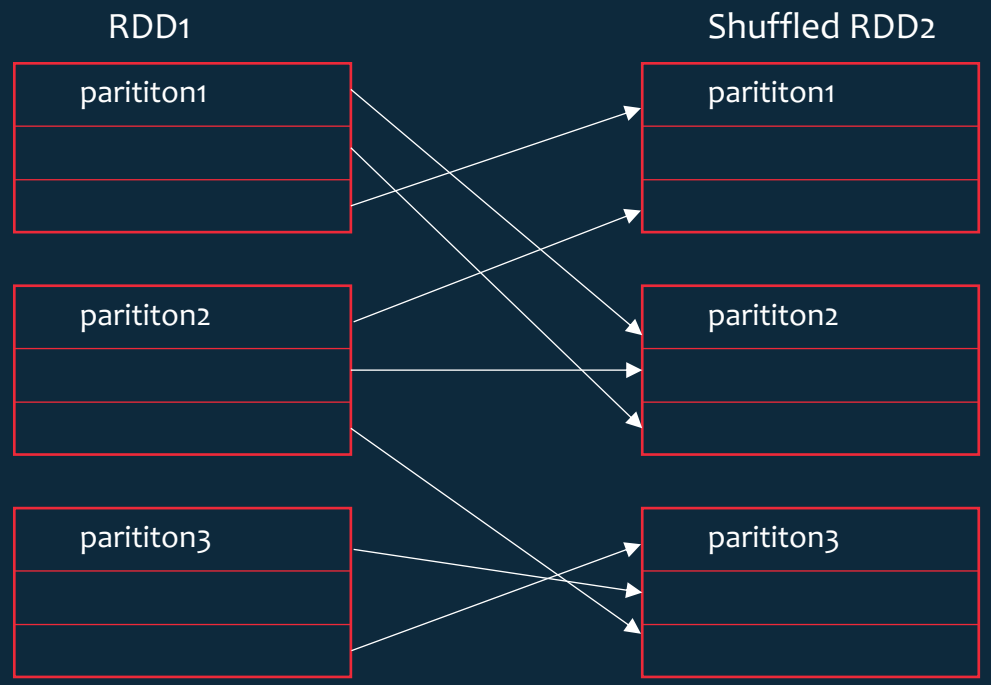
Transformations

Narrow:
These types of transformations convert each input partition to only one output partition. When each partition at the parent RDD is used by at most one partition of the child RDD or when each partition from child produced or dependent on single parent RDD.
Example: Map and Filter

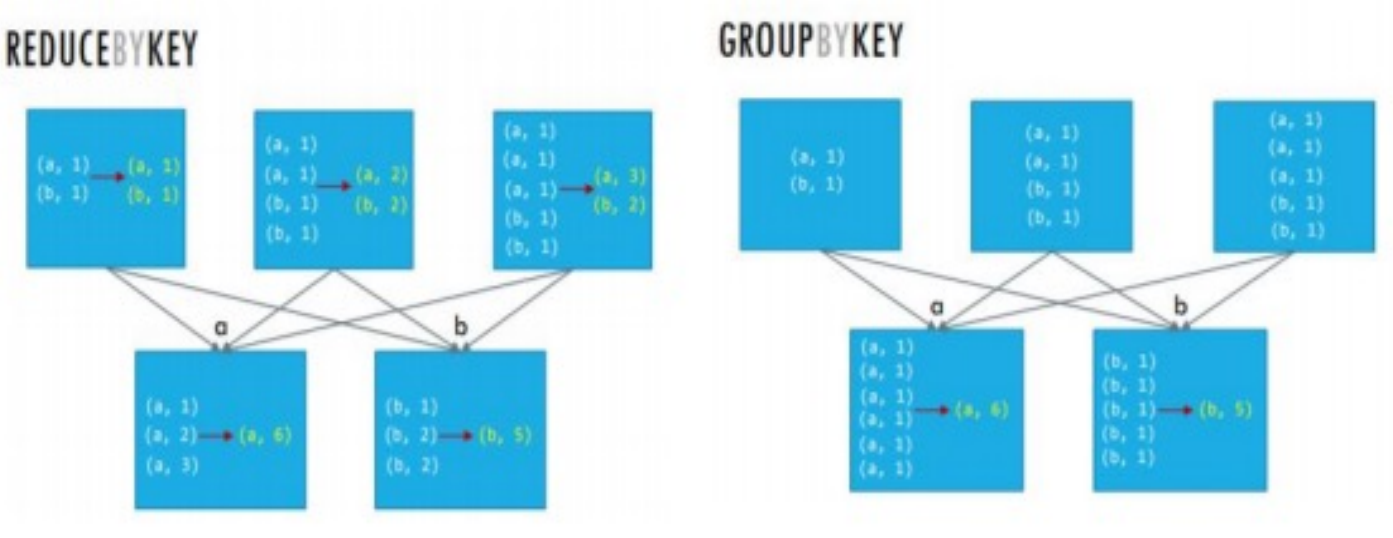
Wide:
This type of transformation will have input partitions contributing to many output partitions. When each partition at the parent RDD is used by multiple partitions of the child RDD or when each partition from child produced or dependent on multiple parent RDD.

- Might Require data shuffling over the cluster network or no data movement.
- Examples: groupByKey(), aggregateByKey(), aggregate(), join(), repartition()





Transformations



reduceBykey		groupByKey
Uses Combiner		Do not uses Combiner
Take one parameter as function – for seqOp and combOp		No parameters as functions. Generally followed by map or flatMap
Implicit Combiner		No combiner
Performance is high for aggregations		Relatively slow for aggregations

- → Both wide Transformations has to be performed on paired RDD only
- → Before Shuffling the data in the same machine ReducebyKey will perform first level transformations then the data will be sent over to the reducers for second level of transformation. However coming to GroupbyKey no first level transformation will be applied because of that all the data will be aggregated on key basis in reducer because of it lots of data will be exchanges b/w nodes through Network it causes the network congestion or increases network traffic.
- → groupbyKey + foldleftofValues = reducebyKey so when ever you want to use groupbyKey with map operation use ReducebyKey for better performance.

Data Compression

What is the Compression Ratio or Compression Power?

The term compression ratio refers to a fraction, percentage or ratio that expresses the difference between the size of a file before it was compressed and after the compression process is complete.

Data compression ratio is defined as the ratio between the uncompressed size and compressed size.

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}$$

Sometimes the space saving is given instead, which is defined as the reduction in size relative to the uncompressed size.

$$\text{Space Saving} = 1 - \frac{\text{Compressed Size}}{\text{Uncompressed Size}}$$

These ratios are usually either expressed using qualitative terms or in a 1:10 type format. As it happens, compression rates below **1:10** are considered reasonable or **good**, while ones higher than 1:10, such as **1:12** are instead considered **excellent**. The other big factor when it comes to the compression ratio is whether or not a compression algorithm is **lossy or lossless.**

1 offers the fastest compression speed but at a lower ratio, and 10 offers the highest compression ratio but at a lower speed

Data Compression

Lossles: Lossless compression is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data.

Example:

- It can be advantageous to make a master lossless file which can then be used to produce additional copies from.

Lossy: lossy compression or irreversible compression is the class of data encoding methods that uses inexact approximations and partial data discarding to represent the content. These techniques are used to reduce data size for storing, handling, and transmitting content.

Example:

- Some times you may observer the quality of the image is compromised after high compression.
- Lossy compression is most commonly used to compress multimedia data (audio, video, and images), especially in applications such as streaming media and internet telephony.

Data Compression

Codec:

- A codec, which is a shortened form of compressor/decompressor, is technology (software or hardware, or both) for compressing and decompressing data; it's the implementation of a compression/decompression algorithm.

Splitable

- You need to know that some codecs support something called splitable compression and that codecs differ in both the speed with which they can compress and decompress data and the degree to which they can compress it.

- Splitable compression is an important concept in a Hadoop context. The way Hadoop works is that files are split if they're larger than the file's block size setting, and individual file splits can be processed in parallel by different mappers.

Resources:

- You can measure the performance of the codec by looking at the usage of Disk and CPU.
- The most common trade-off is between compression ratios (the degree to which a file is compressed) and compress/decompress speeds.

Advantages

- If the input file to a spark job contains compressed data, the time that is needed to read that data from HDFS is reduced and job performance is enhanced. The input data is decompressed automatically when it is being read by executer.

Compression Formats

Compression format	extension name	Multi-file	Support sliced	Top compression ratio	Decompression speed Ranking	tool	Will it parcelled with Hadoop?
gzip	.gz	no	no	2	3	gzip	Yes
bzip2	.bz2	Yes	Yes	1	4	bzip2	Yes
lzo	.lzo	no	Yes	3	2	lzop	no
snappy	.snappy	no	no	4	1	no	no

Compression format	Compression ratio	Compression rate	Decompression rate
gzip/deflate	13.4%	21 MB/s	118 MB/s
bzip2	13.2%	2.4MB/s	9.5MB/s
lzo	20.5%	135 MB/s	410 MB/s
snappy	22.2%	172 MB/s	409 MB/s

Compression ratio: BZip2 > Gzip > Lzo> Snappy | DeCompression rate: Snappy > Lzo> GZIp > BZip2

Compression Formats

gzip:
Applies to the processing of compressed file size within 120M (the standard block size of haoop2 is 120M), which can effectively improve the concurrency of reading.

Storage Space: Medium
CPU Usage: Medium
Splittable: No

GZip is often a good choice for **cold data**, which is accessed infrequently.

---

bzip2 **:**
Due to splitable and multi file behaviour it can be applied on the bigger file.
Storage Space: Low
CPU Usage: Medium
Splittable: No

not suitable for speeder access.

bzip2 is often a good choice for **cold data**, which is accessed infrequently.

Compression Formats

lzo:
The compression/decompression speed is also faster and the compression ratio is reasonable.

Storage Space: High
CPU Usage: Medium
Splittable: Yes

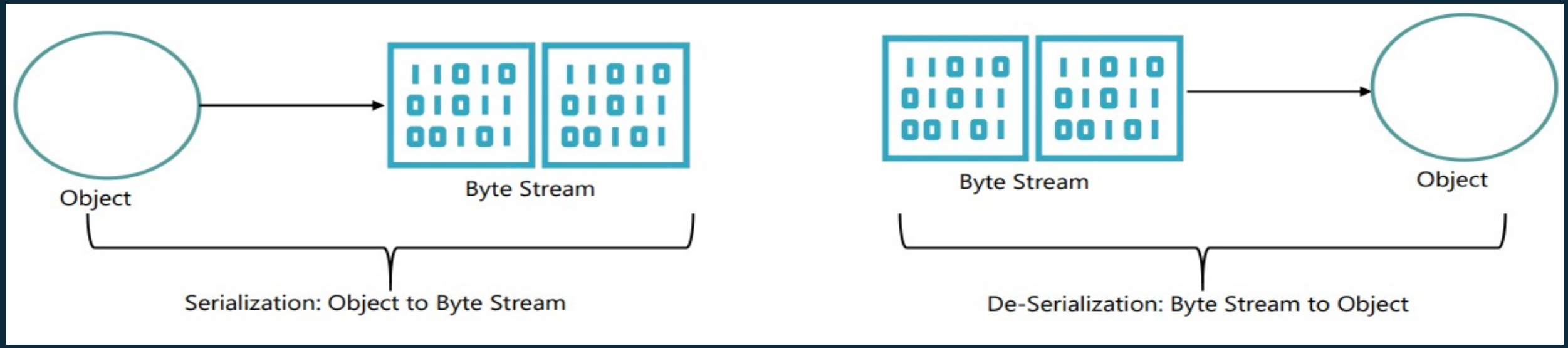
lzo is often a good choice for **hot data,** which is accessed frequently.

Snappy **:**
High compression speed and reasonable compression ratio.

Storage Space: High
CPU Usage: Medium
Splittable: No

snappy is often a good choice for **hot data,** which is accessed frequently.

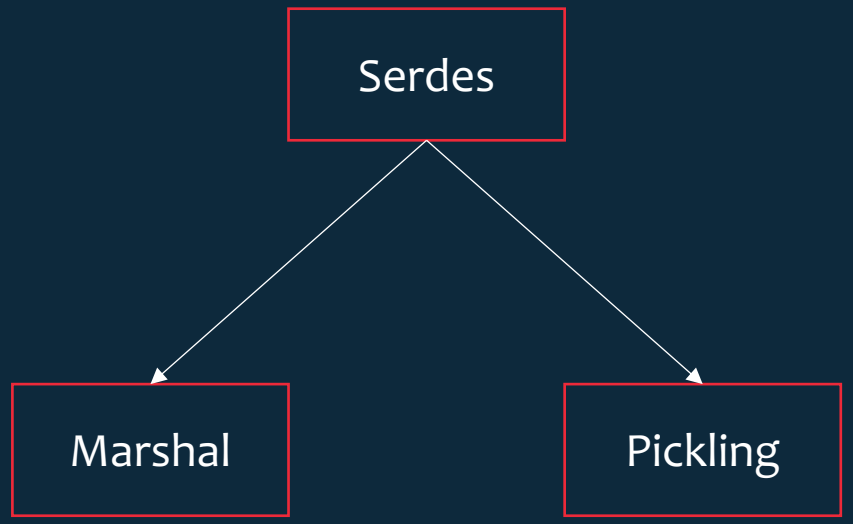
Serialization and De- Serialization



When is it Required?

usually used when the need arises to
→send your data over network
→stored in files.

Types of Serializations in Pyspark



Marshal

→ The marshal module exists mainly to support reading and writing the “pseudo-compiled” code for Python modules of .pyc files.

→”This module doesn’t support all Python object types(Custom Types)”

→ Supported Python types:
booleans, integers, floating point numbers, complex numbers, strings, bytes, bytearrays, tuples, lists, sets, frozensets, dictionaries, and code objects

→Marshal Methods:

marshal.version :
It indicates the format used by the module.

→Version 0 – Historical format
→Version 1 – Shares interned strings
→Version 2 – Uses a binary format for floating point numbers
→Version 3 – Support for object instancing and recursion
→Version 4 – Current Version

marshal.dumps(value[, version]) : The function returns the bytes object that would be written to a file by dump(value, file). The version argument indicates the data format that dumps should use. A ValueError exception is raised if the value has (or contains an object that has) an unsupported type.

marshal.loads(bytes) : This function can reconstruct the data by converting the bytes-like object to a value. EOFError, ValueError or TypeError is raised if no value is found.

Pickle

Pickling: Serialization Un-Pickling: De- Serialization

→ dumps() – This function is called to serialize an object hierarchy.
→ loads() – This function is called to de-serialize a data stream.

For more control over serialization and de-serialization, Pickler or Unpickler objects are created respectively.

Caching

Caching or persistence are optimisation techniques for (iterative and interactive) Spark computations. They help saving interim partial results so they can be reused in subsequent stages. These interim results as RDDs are thus kept in memory (default) or more solid storages like disk and/or replicated.

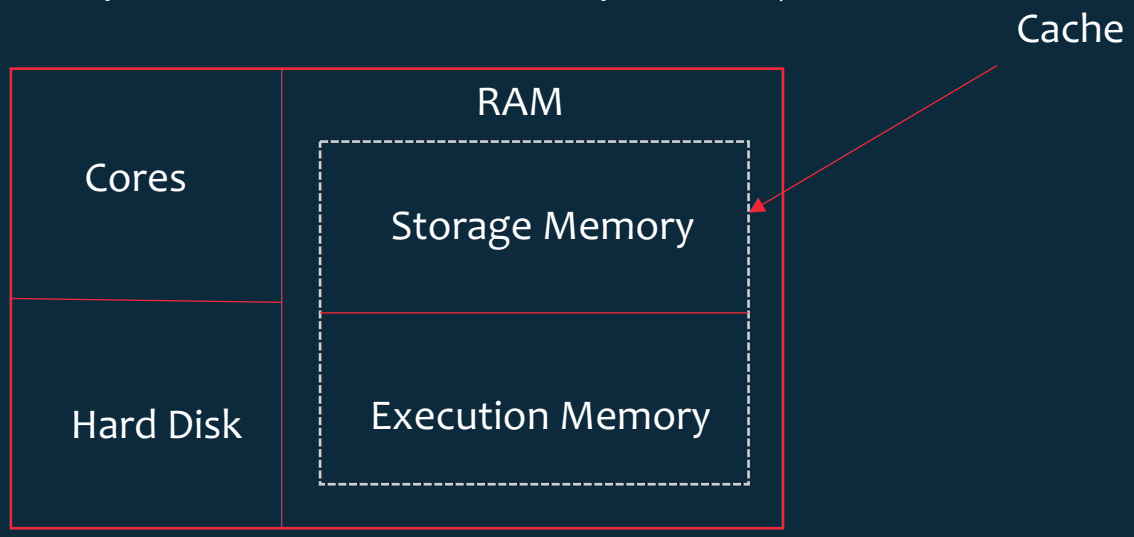
RDDs can be cached using cache operation. They can also be persisted using persist operation.

The difference between cache and persist operations is purely syntactic. cache is a synonym of persist or persist(MEMORY_ONLY), i.e. cache is merely persist with the default storage level MEMORY_ONLY.

getStorageLevel(): Returns storagelevel information of an RDD.

Example: trans1.getStorageLevel()
Output: StorageLevel(False, False, False, False, 1)

StorageLevel class definition:
StorageLevel(disk=false, memory=false, offheap=false, deserialized=false, replication=1)



Caching: StorageLevel

StorageLevel describes how an RDD is persisted (and addresses the following concerns):

- Does RDD use disk?
- How much of RDD is in memory?
- Does RDD use off-heap memory?
- Should an RDD be serialized (while persisting)?
- How many replicas (default: 1) to use (can only be less than 40)?

Caching: StorageLevel

Below are the following StorageLevel (number _2 in the name denotes 2 replicas) spark supports:

- NONE (default)
- DISK_ONLY
- DISK_ONLY_2
- MEMORY_ONLY (default for cache operation for RDDs)
- MEMORY_ONLY_2
- MEMORY_ONLY_SER
- MEMORY_ONLY_SER_2
- MEMORY_AND_DISK
- MEMORY_AND_DISK_2
- MEMORY_AND_DISK_SER
- MEMORY_AND_DISK_SER_2
- OFF_HEAP

MEMORY_ONLY:
Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.

MEMORY_AND_DISK (More Memory and less CPU Utilization)
Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.

MEMORY_ONLY_SER (less Memory and More CPU Utilization)
Store RDD as serialized Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.

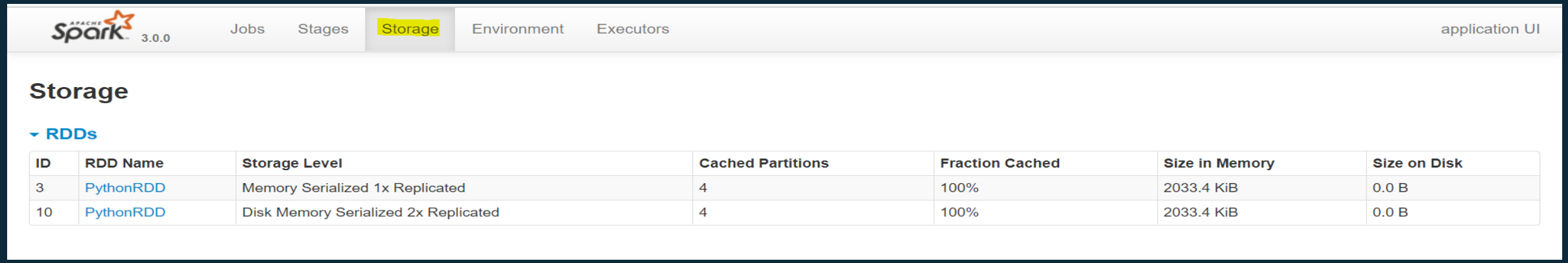
MEMORY_AND_DISK_SER:
Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.

DISK_ONLY: Store the RDD partitions only on disk.

MEMORY_ONLY_2, MEMORY_AND_DISK_2: Same as the levels above, but replicate each partition on two cluster nodes.

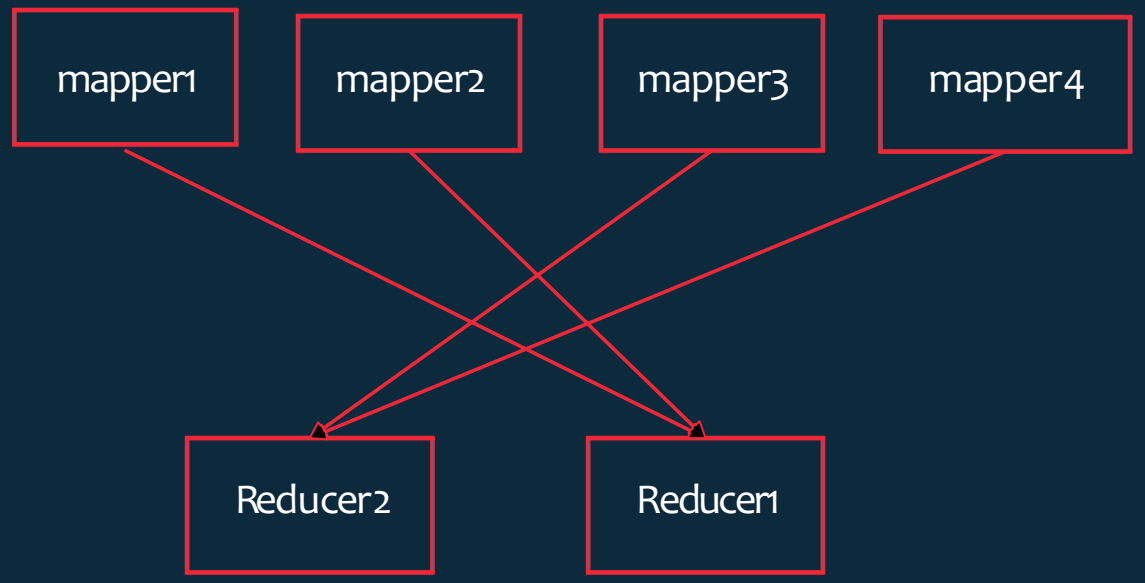
Caching: StorageLevel

Storage Level	Space used	CPU time	In memory	On-disk	Serialized	Recompute some partitions
MEMORY_ONLY	High	Low	Y	N	N	Y
MEMORY_ONLY_SER	Low	High	Y	N	Y	Y
MEMORY_AND_DISK	High	Medium	Some	Some	Some	N
MEMORY_AND_DISK_SER	Low	High	Some	Some	Y	N
DISK_ONLY	Low	High	N	Y	Y	N

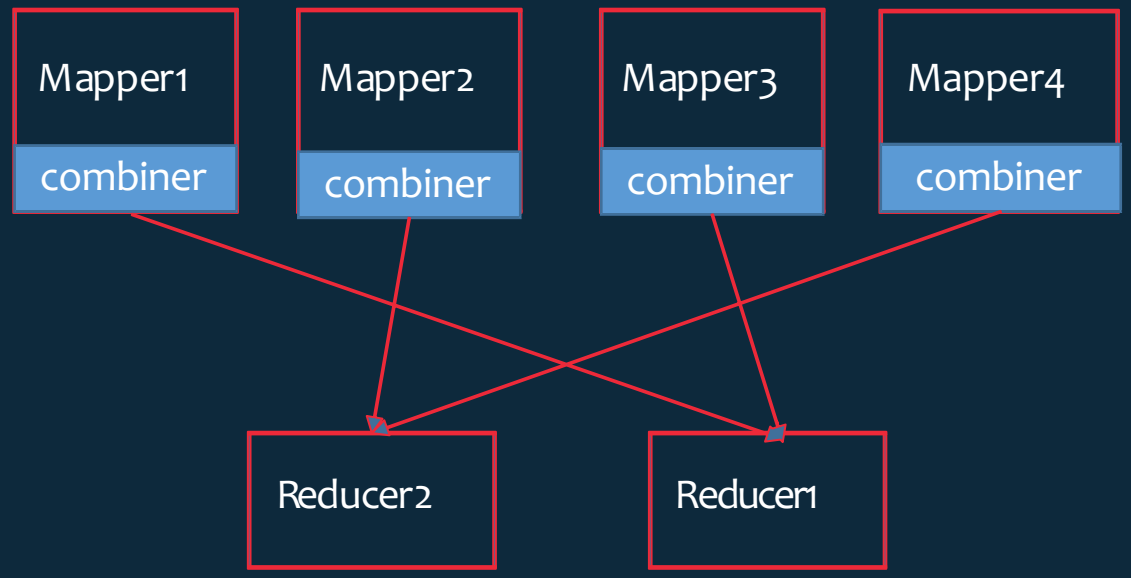


MapPartition as Combiner

mapPartitions is a narrow transformation operation that applies a function to each partition of the RDD.



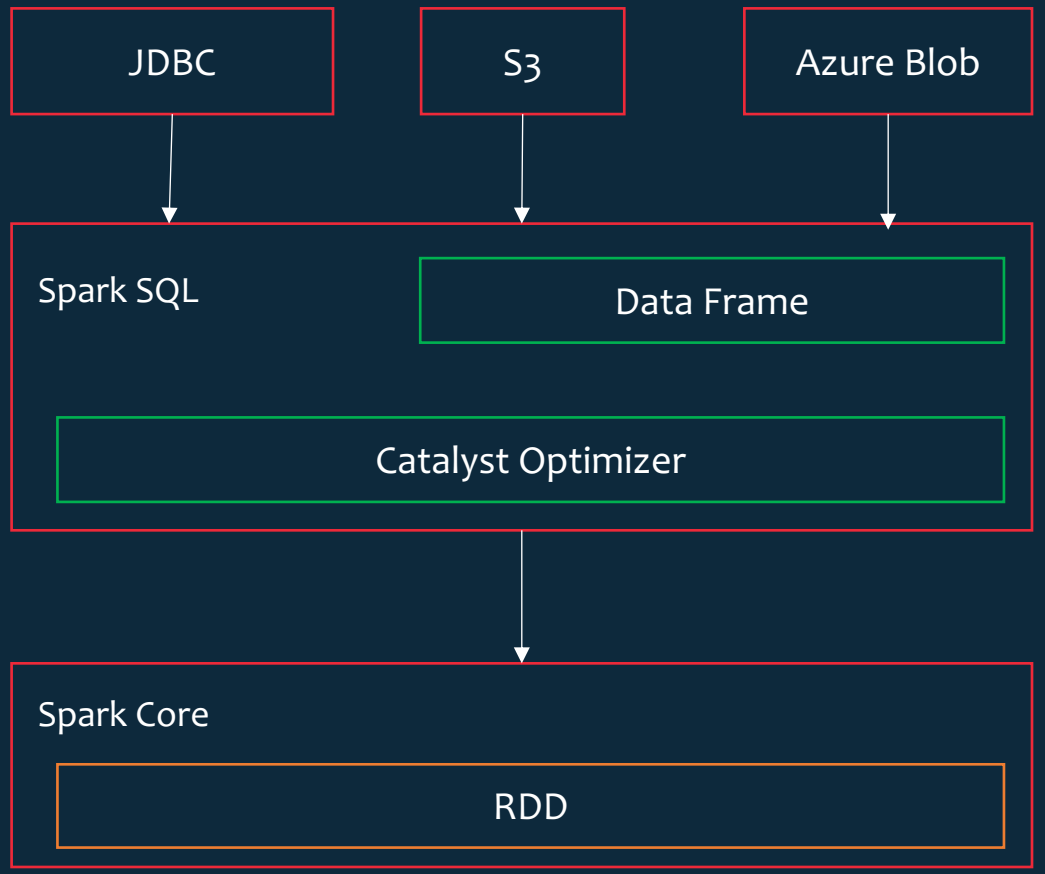
Data Transfer b/w mapper and reducer are high

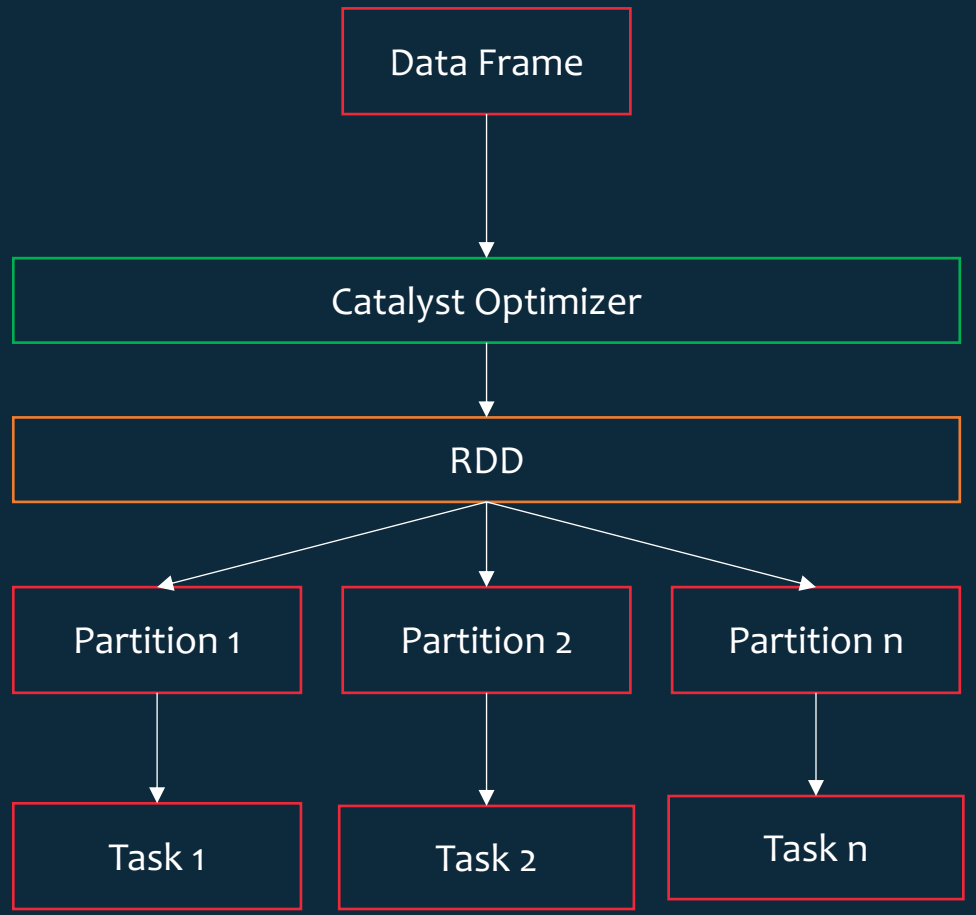


Data Transfer b/w mapper and reducer are low because combiner is acting like mini-reducer and reducing data at mapper side it self

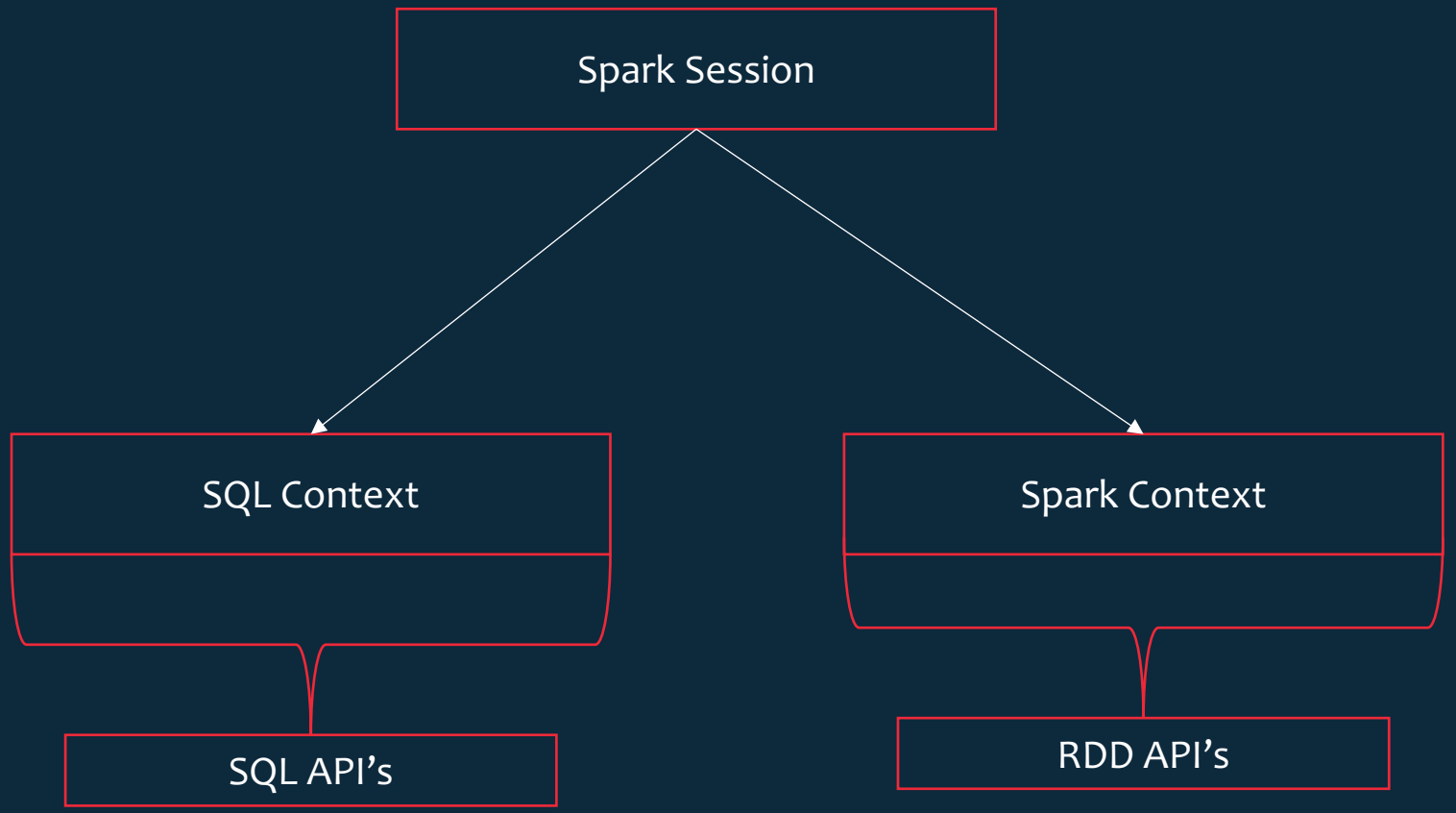
Spark SQL

Spark SQL High level Architecture





Spark Context



CSV

https://spark.apache.org/docs/latest/sql-data-sources-csv.html

JSON

https://spark.apache.org/docs/latest/sql-data-sources-json.html

https://www.databricks.training/spark-ui-simulator/index.html

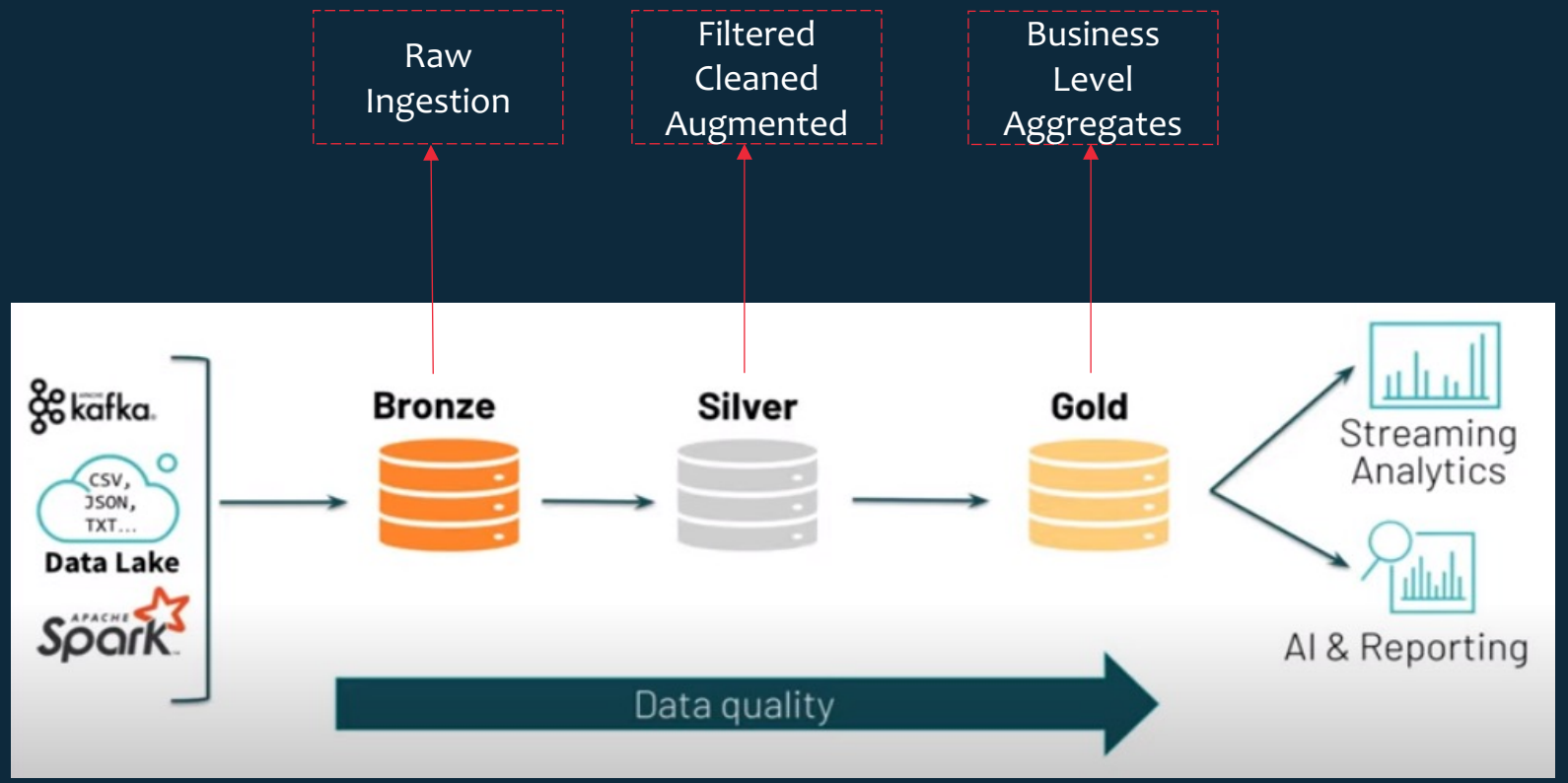
JSON

Divide

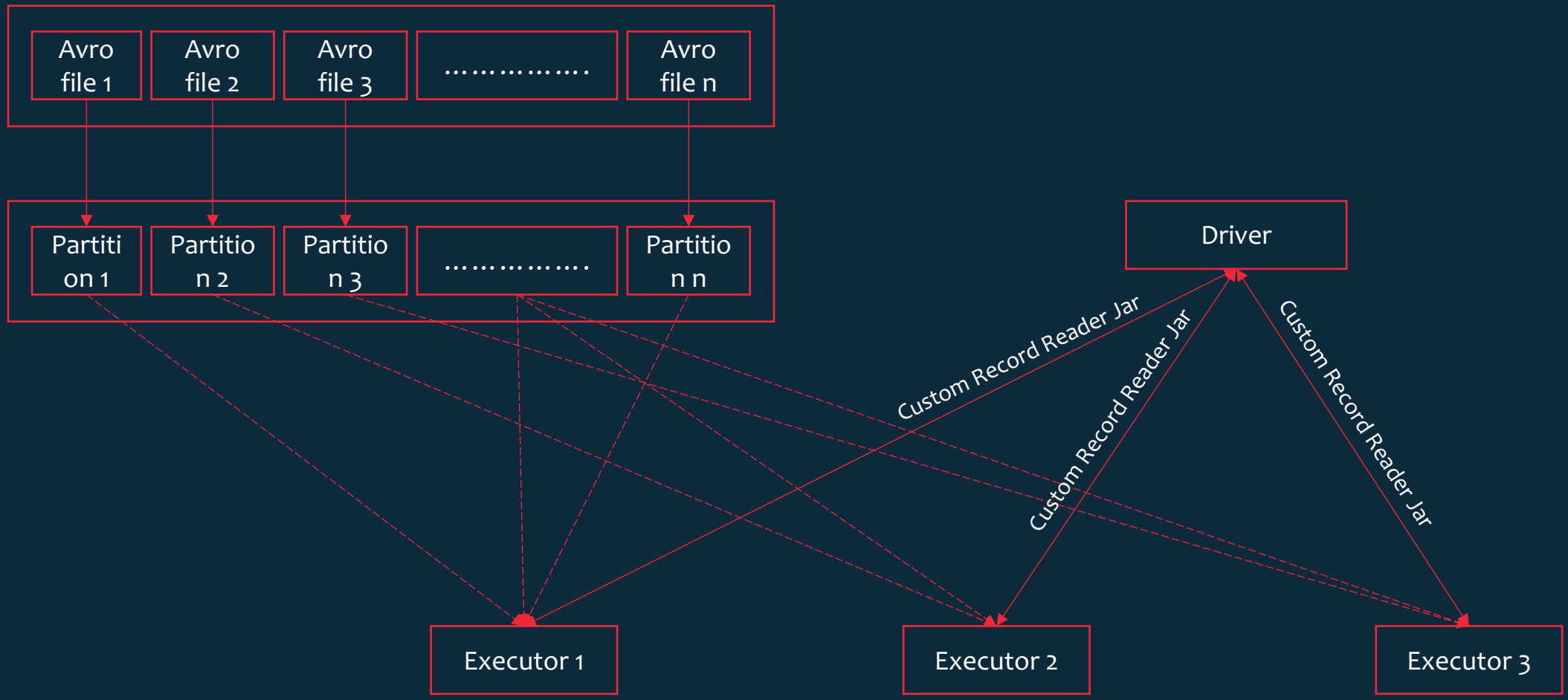
Delta

https://delta.io

Project


Raw
Ingestion
Filtered
Cleaned
Augmented
Business
Level
Aggregates
kafka
CSV,
JSON,
TXT...
Data Lake
APACHE
Spark
Bronze
Silver
Gold
Streaming
Analytics
AI & Reporting
Data quality

Custom Record Reader

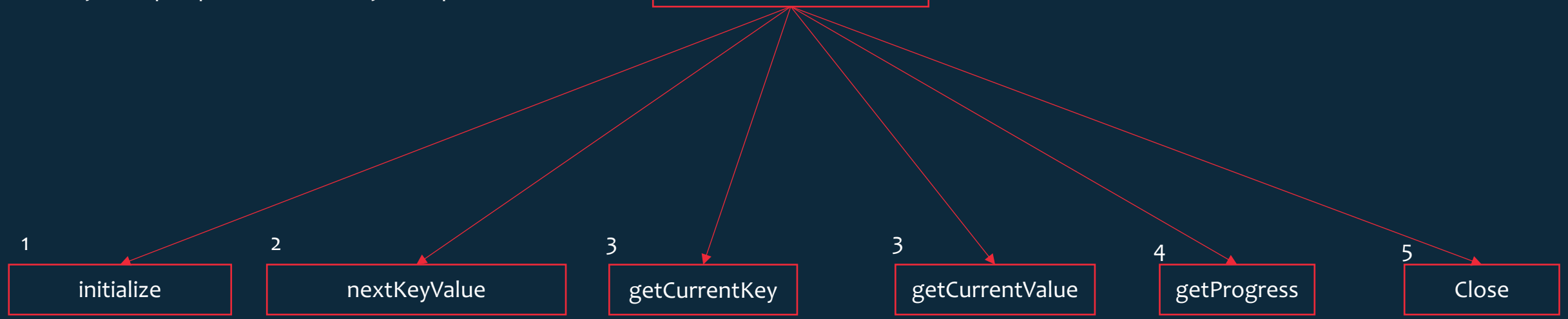


Record Reader Class Framework

RecordReader , typically, converts the byte-oriented view of the input, provided by the InputSplit and creates key-value pairs.

Record Reader

Record Reader Object will be created and framework will be executed at each Executor for each spark partition.



1 **initialize**

- This method will be called once when executor starts processing new partition by executing its task.
- Its used to open the input split and returns back the file input stream as the output.

2 **nextKeyValue**

- Its an recursive method. It will iterate until it returns false.
- This method is used to identify current key and value pair and also to check whether next key-value exists or not.
- If exists it will return true for each iteration or else it will false.
- If this method returns true getcurrentkey and getcurrentvalue methods will be called and returns the current key and value.

3 **getCurrentKey**

3 **getCurrentValue**

These methods will be called as part of the framework execution by nextkeyvalue method and these methods are used to current key-value pairs that are identified by nextkeyvalue method.

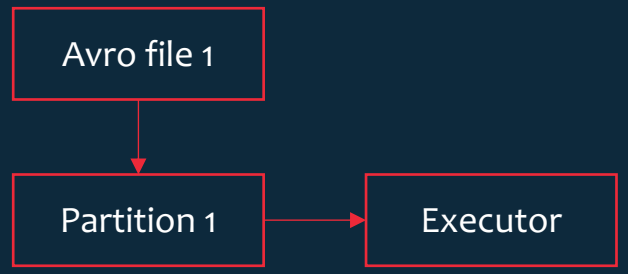
4 **getProgress**

This method is used for updating the percentage of execution for the current execution task in spark UI.

5 **Close**

This method is used to close all the connections, opened files, ect at the successful execution of input split.

Custom Record Reader



Id.
Insert_timestamp,
update_timestamp