Back to the Future

@omarsheriffathy

Disclaimer

This talk is not affiliated or associated with any company or institution I have ever worked for.
the content, opinions, views are all only mine.

What this talk isn't about ?

1. Allocation and deletion of objects
2. Automatic reference counting "ARC"

What is the goal of this talk ?

Understanding how variables and structures are being transformed into bits and bytes.
This will eventually allow us to write faster code and use less memory

"It is a debugging party !"

"It is a debugging party !"

"It is a debugging party !"

"IT IS A DEBUGGING PARTY !"

"IT IS A DEBUGGING PARTY !"

"IT IS A DEBUGGING PARTY !"

"LET'S PARTY !"

Refresher

1 BYTE = 8 BITS

0x01 = 0000 0001

HEX vs BINARY

0x01 = 0000 0001

0x02 = 0000 0010

.....

0x09 = 0000 1001

0x0A = 0000 1010

0x0B = 0000 1011

0x0C = 0000 1100

0x0D = 0000 1101

0x0E = 0000 1110

0x0F = 0000 1111

0x10 = 0001 0000

Int8 = 8 bits = 1 byte

Int16 = 16 bits = 2 byte

Int32 = 32 bits = 3 byte

Int64 = 64 bits = 4 byte

1 WORD = 4 BYTES IN 32-BIT PROCESSORS
1 WORD = 8 BYTES IN 64-BIT PROCESSORS

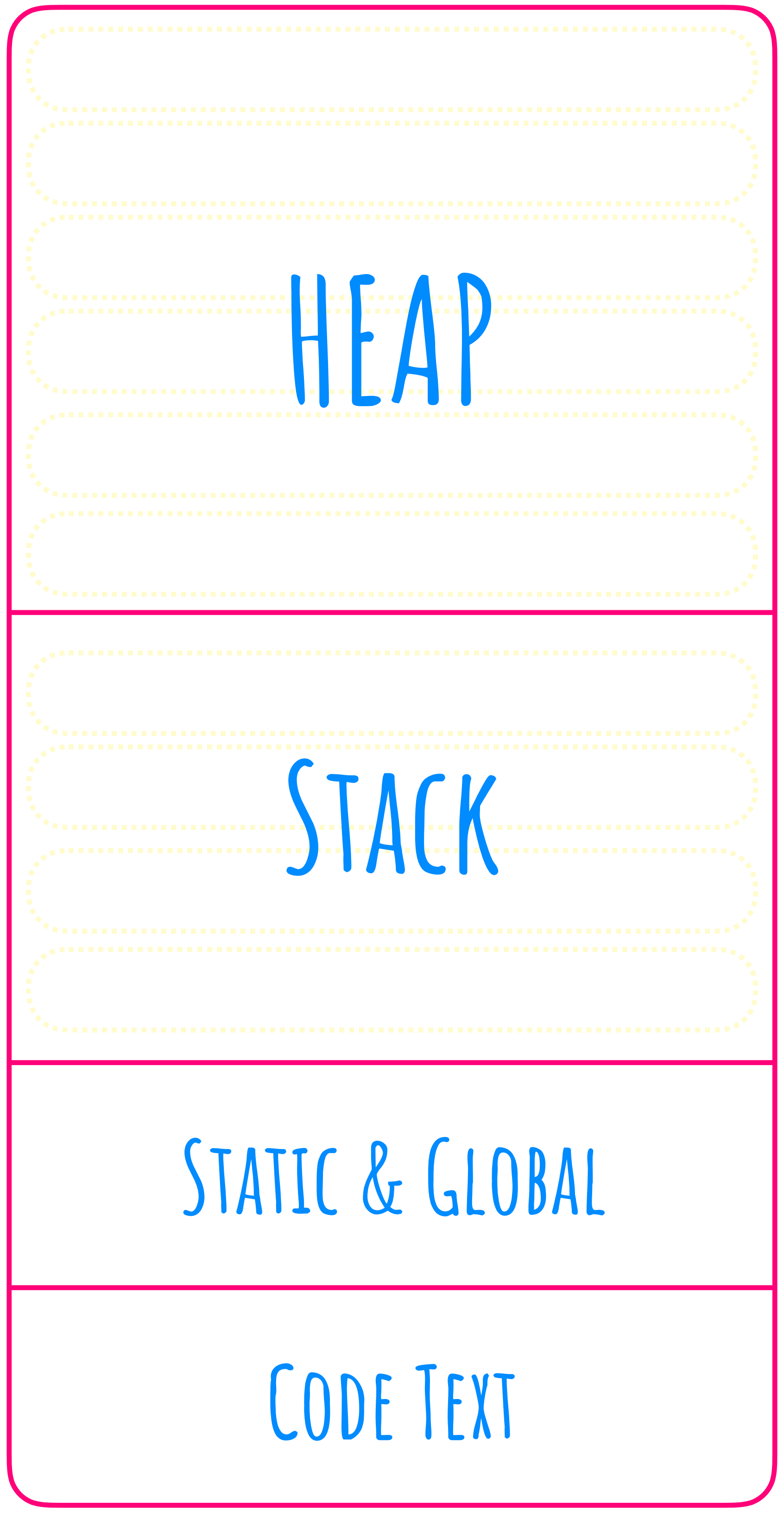
THE PROCESSOR WILL READ 1 WORD AT A TIME TO SPEED THINGS UP ?
A CACHE LINE ? 64 BYTES ?

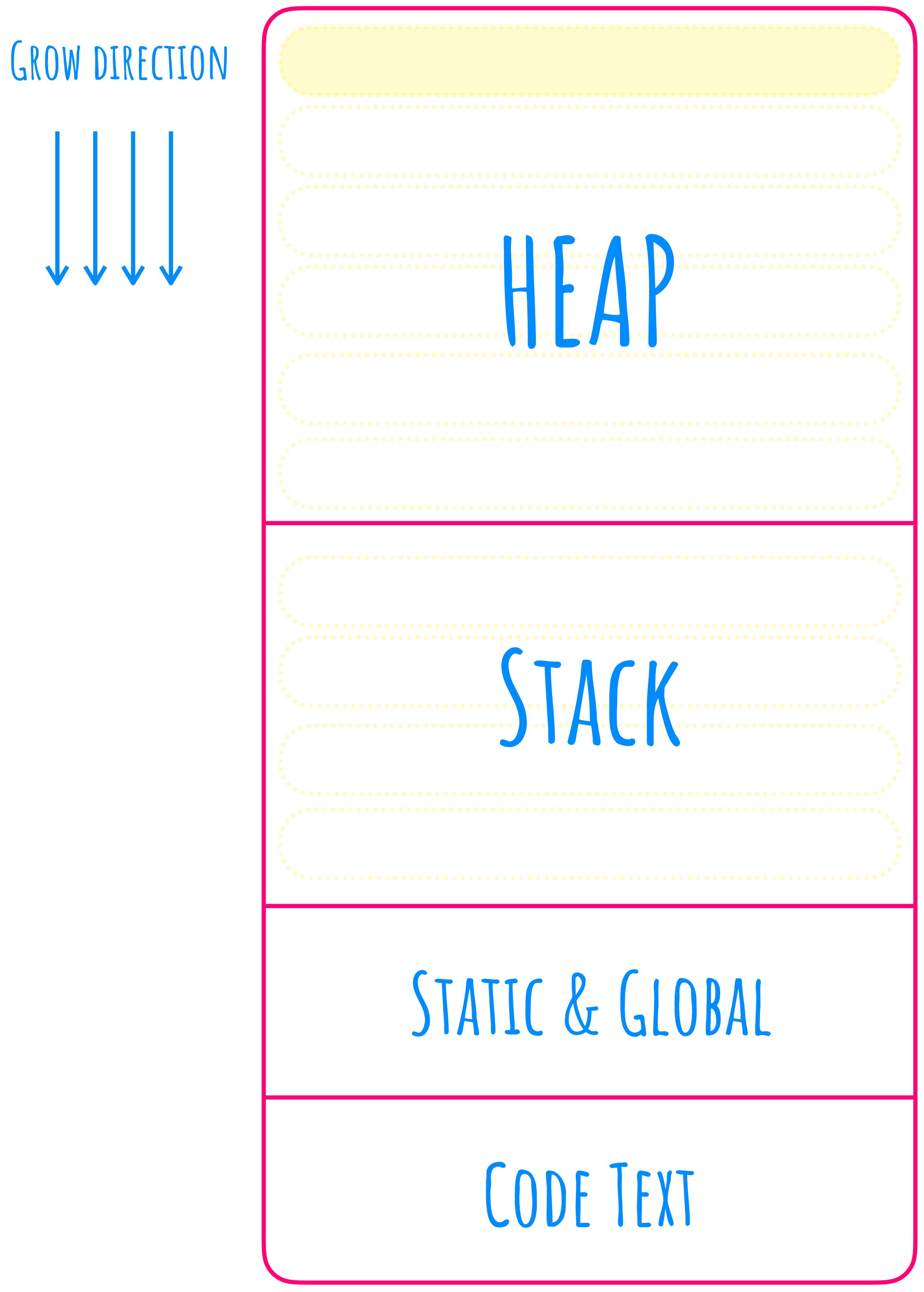
```swift
import Foundation

print(MemoryLayout<Int8>.size)          // 1
print(MemoryLayout<Int16>.size)         // 2
print(MemoryLayout<Int32>.size)         // 4
print(MemoryLayout<Int64>.size)         // 8

print(MemoryLayout<UInt8>.size)         // 1
print(MemoryLayout<UInt16>.size)        // 2
print(MemoryLayout<UInt32>.size)        // 4
print(MemoryLayout<UInt64>.size)        // 8

print(MemoryLayout<CShort>.size)        // 2
print(MemoryLayout<Int>.size)           // 8
print(MemoryLayout<CLong>.size)         // 8

print(MemoryLayout<Float>.size)         // 4
print(MemoryLayout<Double>.size)        // 8

print(MemoryLayout<CChar>.size)         // 1
```

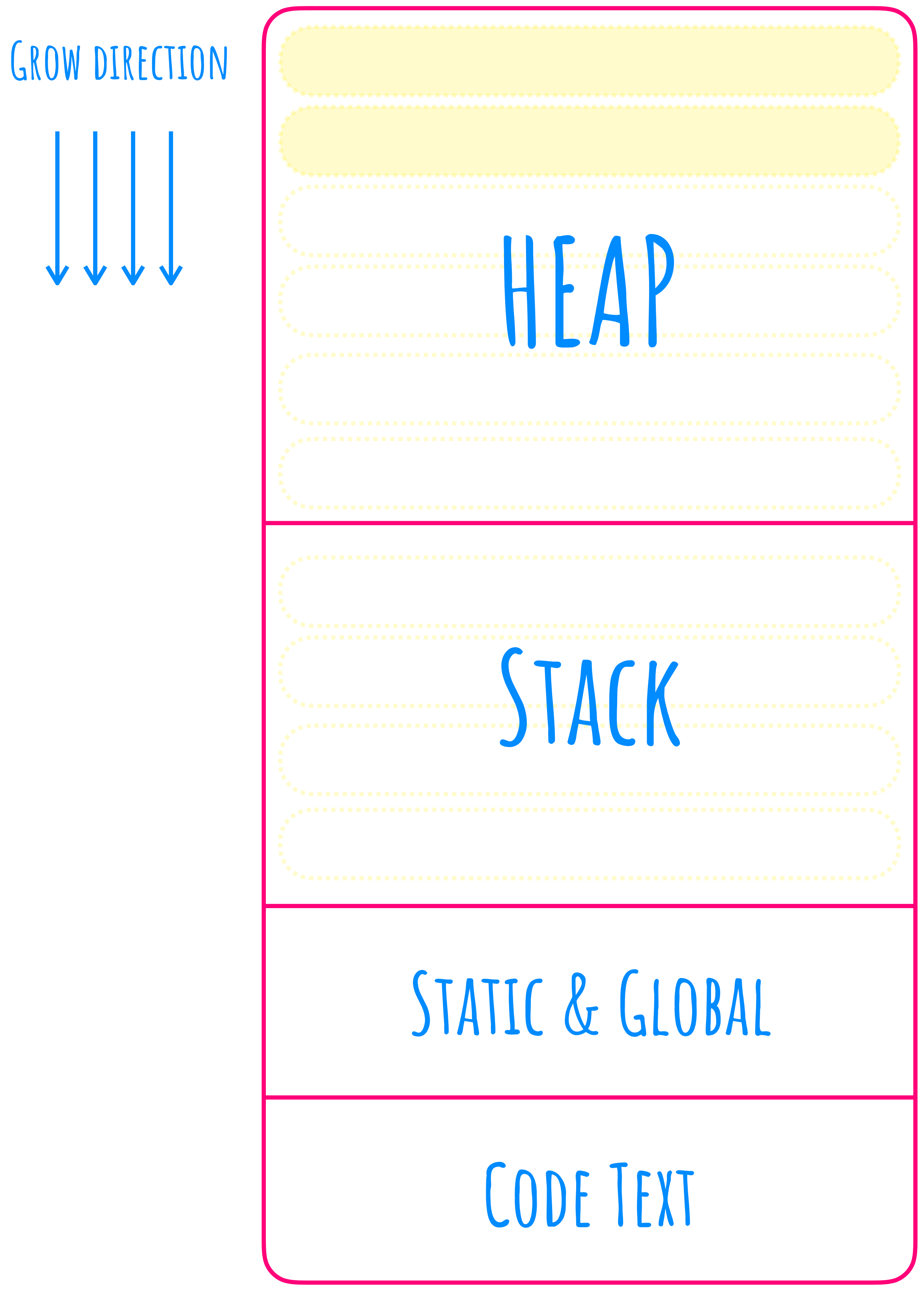
What happens when you run a program ?

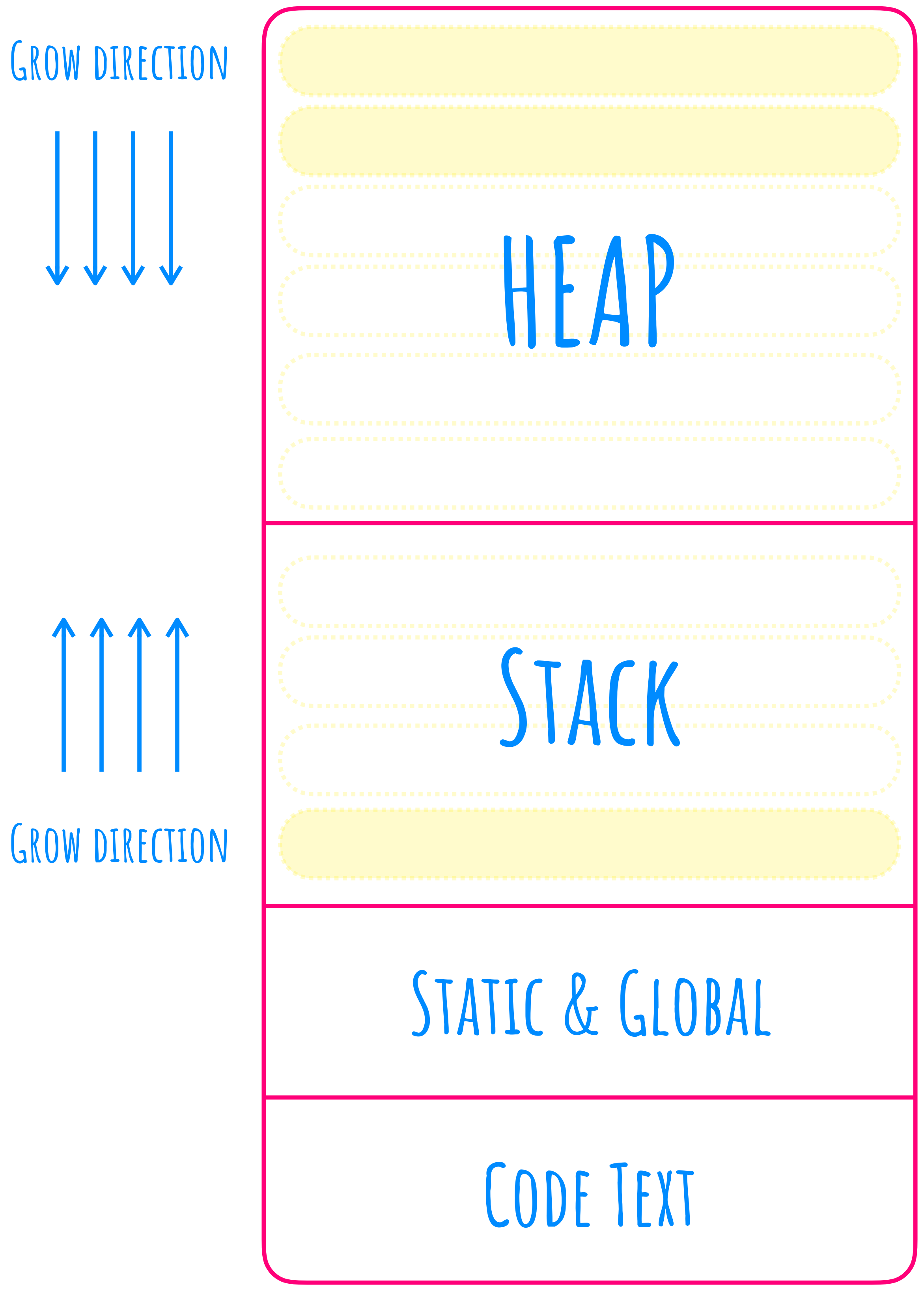
1. Load program from hard disk to RAM
2. Run the program

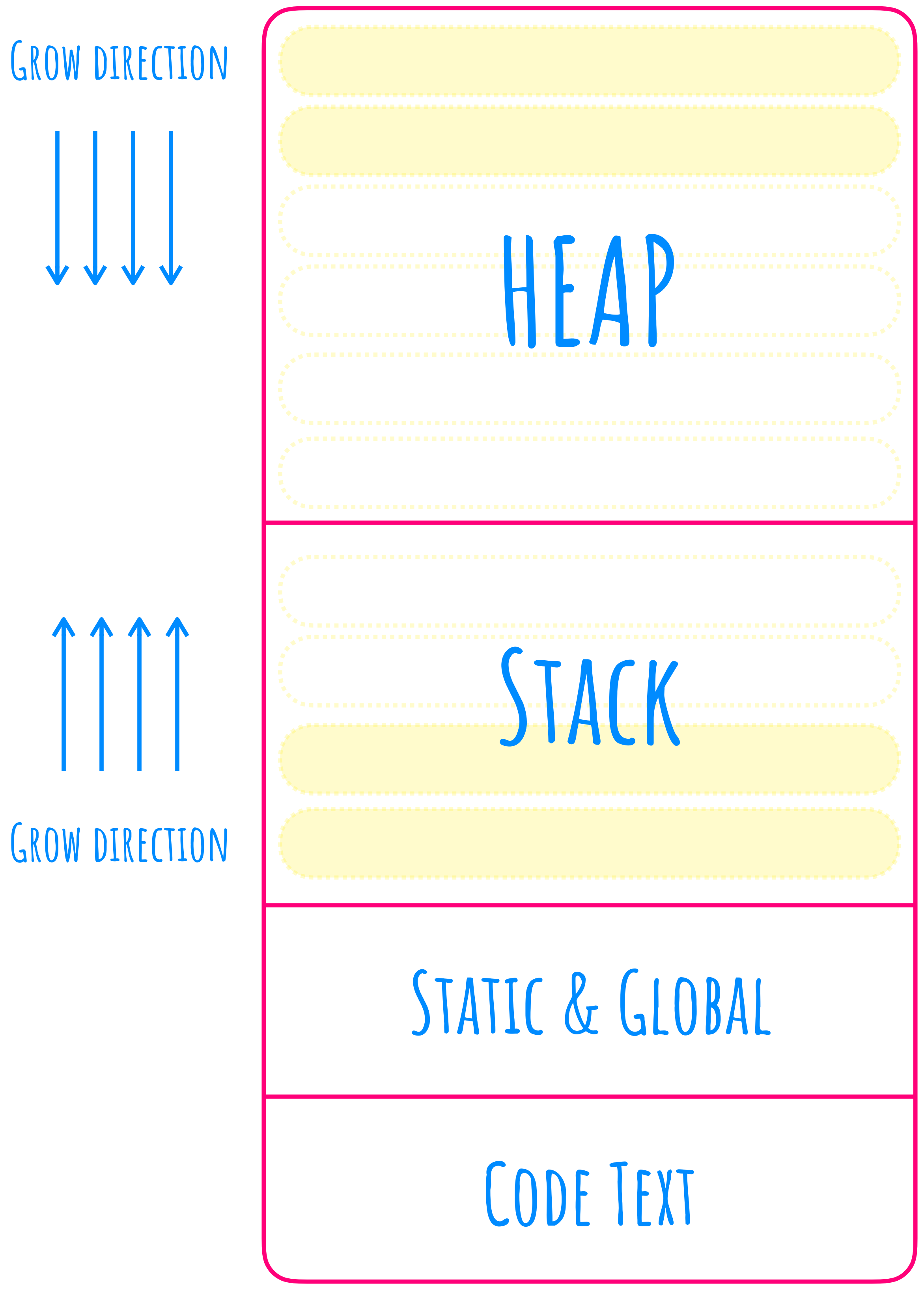
How does the program memory look like ?


HEAP
STACK
STATIC & GLOBAL
CODE TEXT


Grow direction
Heap
Stack
Static & Global
Code Text


Grow direction
Heap
Stack
Static & Global
Code Text


Grow direction
Heap
Stack
Grow direction
Static & Global
Code Text


Grow direction
Heap
Stack
Grow direction
Static & Global
Code Text

Software

Software

Example 0

Example 0

```swift
import Foundation

let a: Int8  = 1    // 0x100001058
```

```
0x100001058: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x100001068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Example 0

```swift
import Foundation

let a: Int8  = 1    // 0x100001058
```

```
0x100001058: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x100001068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Example 0

```swift
import Foundation

let a: Int8  = 1    // 0x100001058
let b: Int16 = 2    // 0x10000105a
```

```
0x100001058: 01 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x100001068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Example 0

```swift
import Foundation

let a: Int8  = 1    // 0x100001058
let b: Int16 = 2    // 0x10000105a
```

```
0x100001058: 01 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x100001068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Example 0

```swift
import Foundation

let a: Int8  = 1    // 0x100001058
let b: Int16 = 2    // 0x10000105a
let c: Int32 = 3    // 0x10000105c
```

```
0x100001058: 01 00 02 00 03 00 00 00 00 00 00 00 00 00 00 00  ................
0x100001068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Example 0

```swift
import Foundation

let a: Int8  = 1    // 0x100001058
let b: Int16 = 2    // 0x10000105a
let c: Int32 = 3    // 0x10000105c
```

```
0x100001058: 01 00 02 00 03 00 00 00 00 00 00 00 00 00 00 00  ................
0x100001068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Example 0

```swift
import Foundation

let a: Int8  = 1    // 0x100001058
let b: Int16 = 2    // 0x10000105a
let c: Int32 = 3    // 0x10000105c
let d: Int64 = 4    // 0x100001060
```

```
0x100001058: 01 00 02 00 03 00 00 00 04 00 00 00 00 00 00 00  ................
0x100001068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Example 0

```swift
import Foundation

let a: Int8  = 1    // 0x100001058
let b: Int16 = 2    // 0x10000105a
let c: Int32 = 3    // 0x10000105c
let d: Int64 = 4    // 0x100001060
```

```
0x100001058: 01 00 02 00 03 00 00 00 04 00 00 00 00 00 00 00  ................
0x100001068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Example 0

```swift
import Foundation

let a: Int8  = 1    // 0x100001058
let b: Int16 = 2    // 0x10000105a
let c: Int32 = 3    // 0x10000105c
let d: Int64 = 4    // 0x100001060
```

```
0x100001058: 01 00 02 00 03 00 00 00 04 00 00 00 00 00 00 00  ................
0x100001068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Conclusion

Variable a has got 00 after it due to alignment.
We cannot assign b just after a unless we explicitly use packing.

Software

[Example 1]

Example 1

```swift
import Foundation

func main() {
    var a: Int8  = 1 // 0x7ffeefbff5d8
}

main()
```

```
0x7ffeefbff5c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x7ffeefbff5d0: 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ................
```

EXAMPLE 1

```swift
import Foundation

func main() {
    var a: Int8  = 1 // 0x7ffeefbff5d8
    var b: Int16 = 2 // 0x7ffeefbff5d0
}

main()
```

```
0x7ffeefbff5c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x7ffeefbff5d0: 02 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ................
```

Example 1

```swift
import Foundation

func main() {
    var a: Int8  = 1 // 0x7ffeefbff5d8
    var b: Int16 = 2 // 0x7ffeefbff5d0
    var c: Int32 = 3 // 0x7ffeefbff5c8
}

main()
```

```
0x7ffeefbff5c0: 00 00 00 00 00 00 00 00 03 00 00 00 00 00 00 00  ................
0x7ffeefbff5d0: 02 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ................
```

Example 1

```swift
import Foundation

func main() {
    var a: Int8  = 1 // 0x7ffeefbff5d8
    var b: Int16 = 2 // 0x7ffeefbff5d0
    var c: Int32 = 3 // 0x7ffeefbff5c8
    var d: Int64 = 4 // 0x7ffeefbff5c0
}

main()
```

```
0x7ffeefbff5c0: 04 00 00 00 00 00 00 00 03 00 00 00 00 00 00 00  ................
0x7ffeefbff5d0: 02 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ................
```

Example 1

```swift
import Foundation

func main() {
    var a: Int8  = 1 // 0x7ffeefbff5d8
    var b: Int16 = 2 // 0x7ffeefbff5d0
    var c: Int32 = 3 // 0x7ffeefbff5c8
    var d: Int64 = 4 // 0x7ffeefbff5c0
}

main()
```

```
0x7ffeefbff5c0: 04 00 00 00 00 00 00 00 03 00 00 00 00 00 00 00  ................
0x7ffeefbff5d0: 02 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ................
```

Conclusion

When we add variables inside a function or a scope, the variables are naturally going to live on stack.
Which means, the memory locations are going to decrease since the stack grows upwards.
See variable a has been declared before b yet it has reference value greater than b.
a has location 0x7ffeefbff5d8
b has location 0x7ffeefbff5d0

a location 0x7ffeefbff5d8 **>** **b location** 0x7ffeefbff5d0

Software

Example 2

Example 2

```swift
import Foundation

var arr: [Int8] = [1, 2, 3]
```

```
0x100001080: 30 56 63 00 01 00 00 00 00 00 00 00 00 00 00 00  0Vc.............
0x100001090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0x100635630: c0 9c 92 8f ff 7f 00 00 02 00 00 00 00 00 00 00  ................
0x100635640: 03 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00  ................
0x100635650: 01 02 03 ef fe 7f 00 00 00 00 00 00 00 00 02 00  ................
```

Example 2

```swift
import Foundation

var arr: [Int8] = [1, 2, 3]
arr.append(4)
```

```
0x100001080: 80 2f 54 00 01 00 00 00 00 00 00 00 00 00 00 00  ./T.............
0x100001090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0x100542f80: c0 9c 92 8f ff 7f 00 00 02 00 00 00 00 00 00 00  ................
0x100542f90: 04 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00  ........ …….
0x100542fa0: 01 02 03 04 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Example 2

```
import Foundation

var arr: [Int8] = [1, 2, 3]
arr.append(4)
arr.append(5)
```

```
0x100001080: 80 2f 54 00 01 00 00 00 00 00 00 00 00 00 00 00  ./T.............
0x100001090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0x100542f80: c0 9c 92 8f ff 7f 00 00 02 00 00 00 00 00 00 00  ................
0x100542f90: 05 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00  ........ .......
0x100542fa0: 01 02 03 04 05 00 00 00 00 00 00 00 00 00 00 00  ................
```

Example 2

```swift
import Foundation

var arr: [Int8] = [1, 2, 3]
arr.append(4)
arr.append(5)
arr.append(6)
```

```
0x100001080: 80 2f 54 00 01 00 00 00 00 00 00 00 00 00 00 00  ./T.............
0x100001090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0x100542f80: c0 9c 92 8f ff 7f 00 00 02 00 00 00 00 00 00 00  ................
0x100542f90: 06 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00  ........ …….
0x100542fa0: 01 02 03 04 05 06 00 00 00 00 00 00 00 00 00 00  ................
```

Example 2

```
import Foundation

var arr: [Int8] = [1, 2, 3]
arr.append(4)
arr.append(5)
arr.append(6)
```

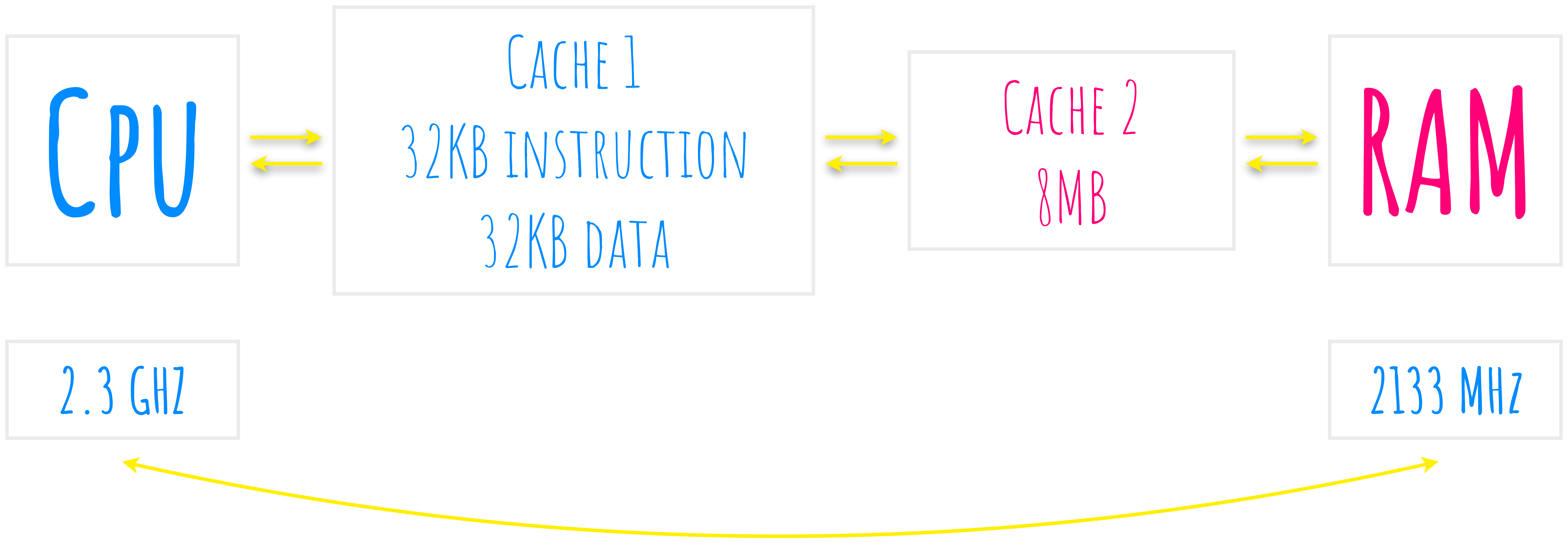
```
0x100001080: 80 2f 54 00 01 00 00 00 00 00 00 00 00 00 00 00  ./T.............
0x100001090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0x100542f80: c0 9c 92 8f ff 7f 00 00 02 00 00 00 00 00 00 00  ................
0x100542f90: 06 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00  ........ …….
0x100542fa0: 01 02 03 04 05 06 00 00 00 00 00 00 00 00 00 00  ................
```

Conclusion

Array has got 8 bytes pointer representing the actual memory location of the storage of that array.
Once we jump to that location we get 32 bytes of meta data related to the array itself, the meta data contains stuff like the count and capacity
After the 32 bytes we start having the actual array values

Hardware



DIFFERENCE IN CLOCK SPEED !

IF THE CPU MAKES A FETCH FROM RAM IT HAS TO SLOW DOWN AND WAIT FOR RAM RESPONSE, WHAT A WASTE !

L1 instruction size	`sysctl hw.l1icachesize`
L1 data size	`sysctl hw.l1dcachesize`
L2 Data size	`sysctl hw.l2cachesize`
L3 Data size	`sysctl hw.l3cachesize`

```
sudo sysctl -a | grep cache
```

This leads us

to

Data oriented programming

Enough talking,

Coding starts here

Thank you.

Questions ?