Topics

 # Installation
 # Execute go code
 # Variables Types
 # Struct Type
 # Pointers
 # Constants

Installation

Variables

Integrity and Cost

Integrity, in the context of **computer** systems, refers to methods of ensuring that data is real, accurate and safeguarded from unauthorized user modification.

Variables are at the heart of the language and provide the ability to read from and write to memory.
In Go, access to memory is type safe.
This means the compiler takes type seriously and will not allow us to use variables outside the scope of how they are declared.

What is the value of the byte at address FFE1?

FFE4	FFE3	FFE2	FFE1
00000000	11001011	01100101	00001010

- Apples
  - Base 2 - 10010001
  - Base 10 - 145
  - Base 16 - 91

- Color
  - 111111111111111111111111 (24 char)
  - 16777215 (10 char)
  - FFFFFF (6 char)

Type provides two pieces of information that both the compiler and you need to perform the same exercise we just went through.

1. The amount of memory, in bytes, to look at
2. The representation of those bytes

The Go language provides these basic numeric types:

Unsigned Integers
uint8, uint16, uint32, uint64

Signed Integers
int8, int16, int32, int64

Real Numbers
float32, float64

Predeclared Integers
uint, int, uintptr

Code Review [knowinggolang/topics/go/language/variables]

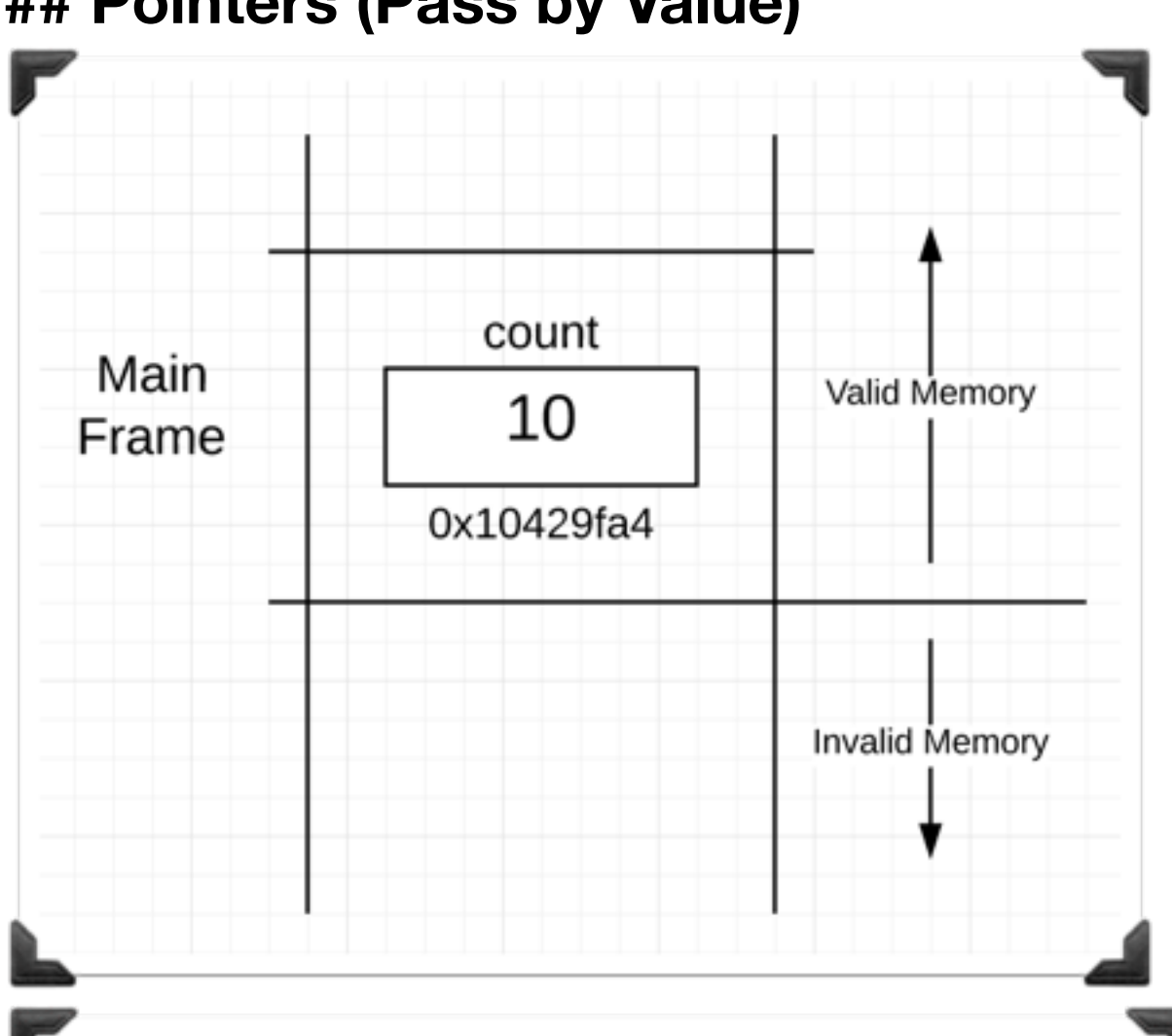
Struct Type

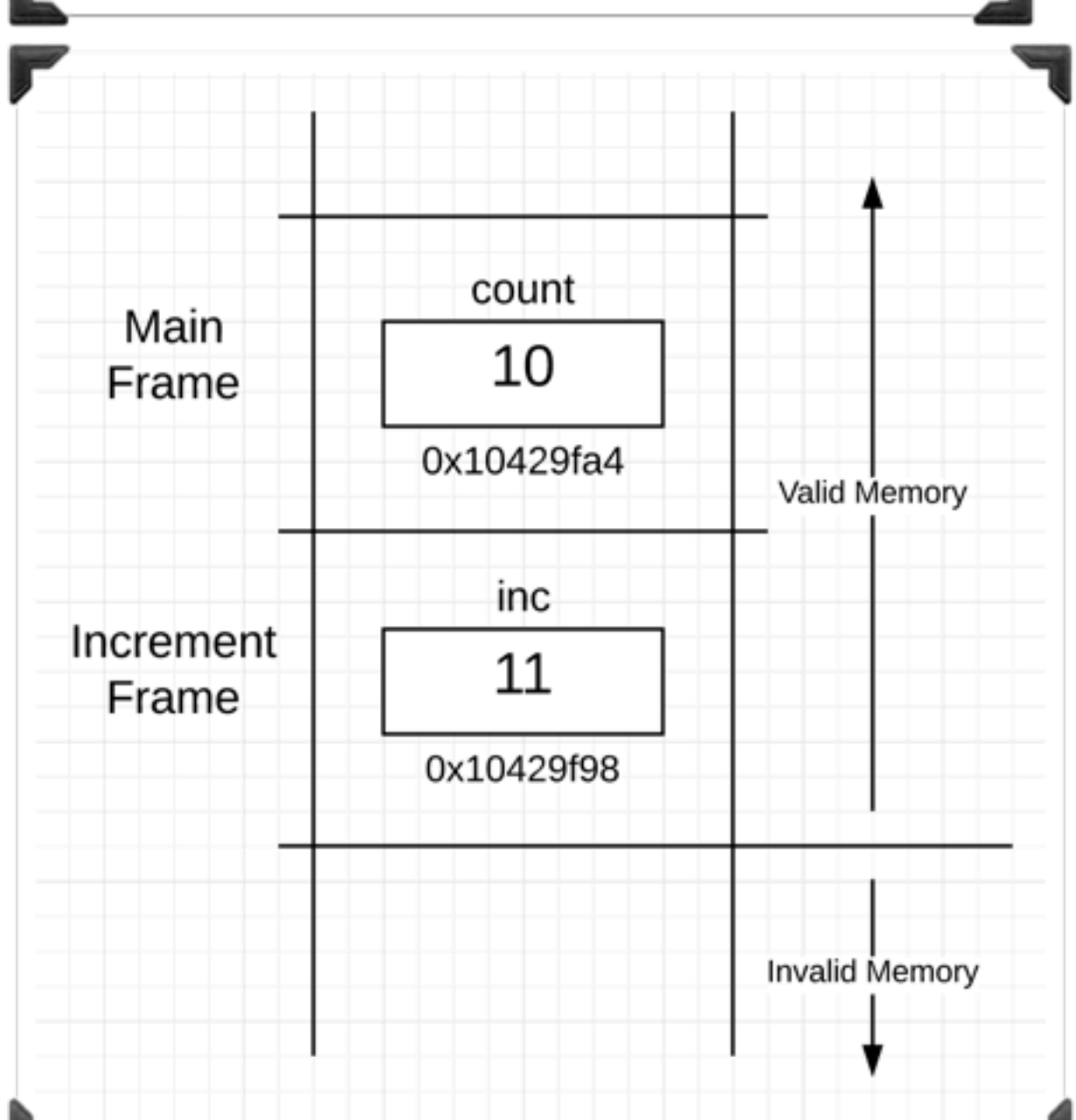
```
## Struct Types

Struct types are a way of creating complex types that group fields of data together. They are a great way of organizing and sharing the different aspects of the data your program consumes.


## Code Review

[Declare, create and initialize struct types](example1/example1.go)
[Anonymous struct types](example2/example2.go)
[Named vs Unnamed types](example3/example3.go)

## Advanced Code Review

[Struct type alignments](advanced/example1/example1.go)
```

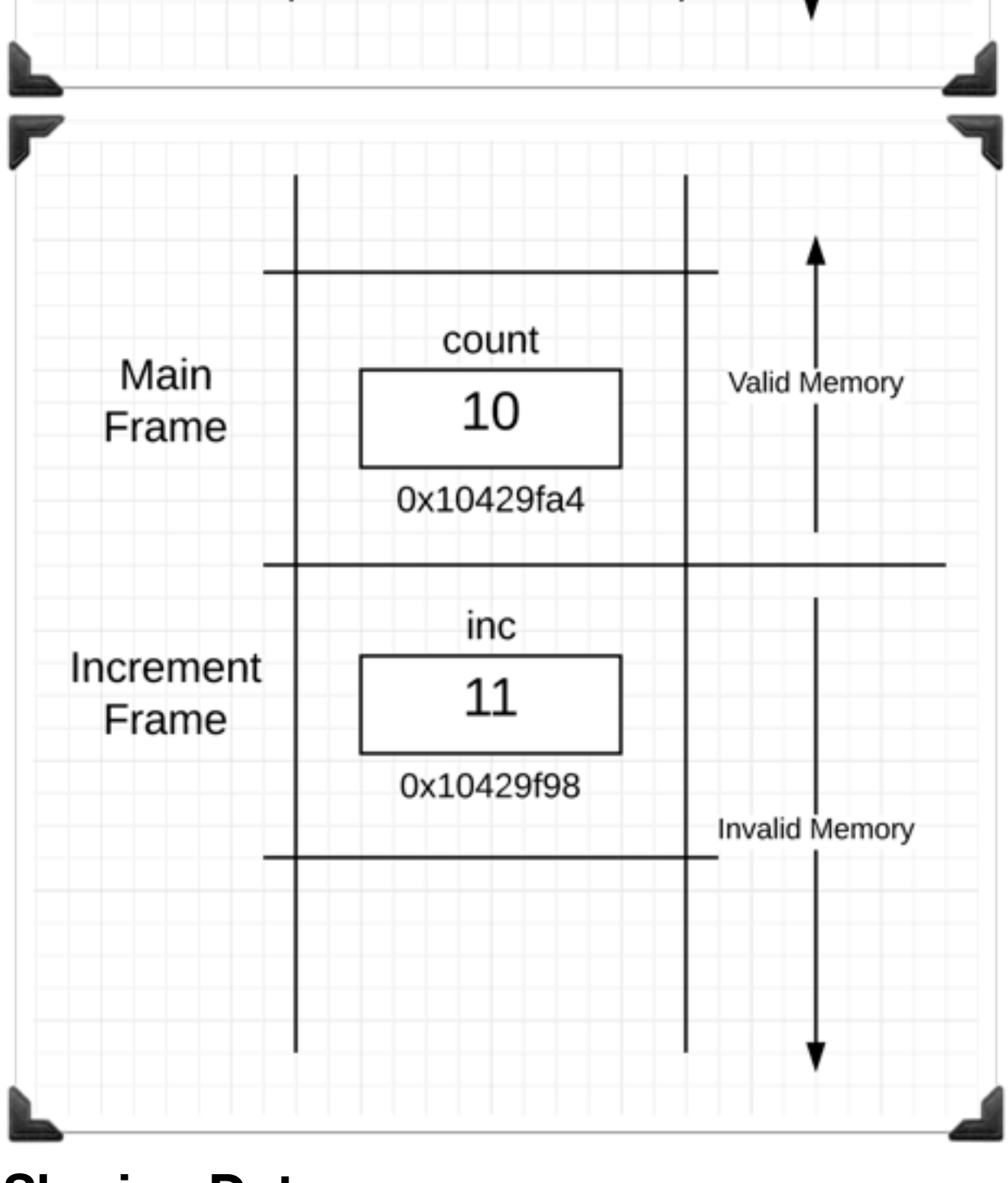
Pointers

```
Pointers provide a way to share data across program boundaries. Having the ability to share and reference data with a pointer provides the benefit of efficiency. There is only one copy of the data and everyone can see it changing. The cost is that anyone can change the data which can cause side effects in running programs.
```

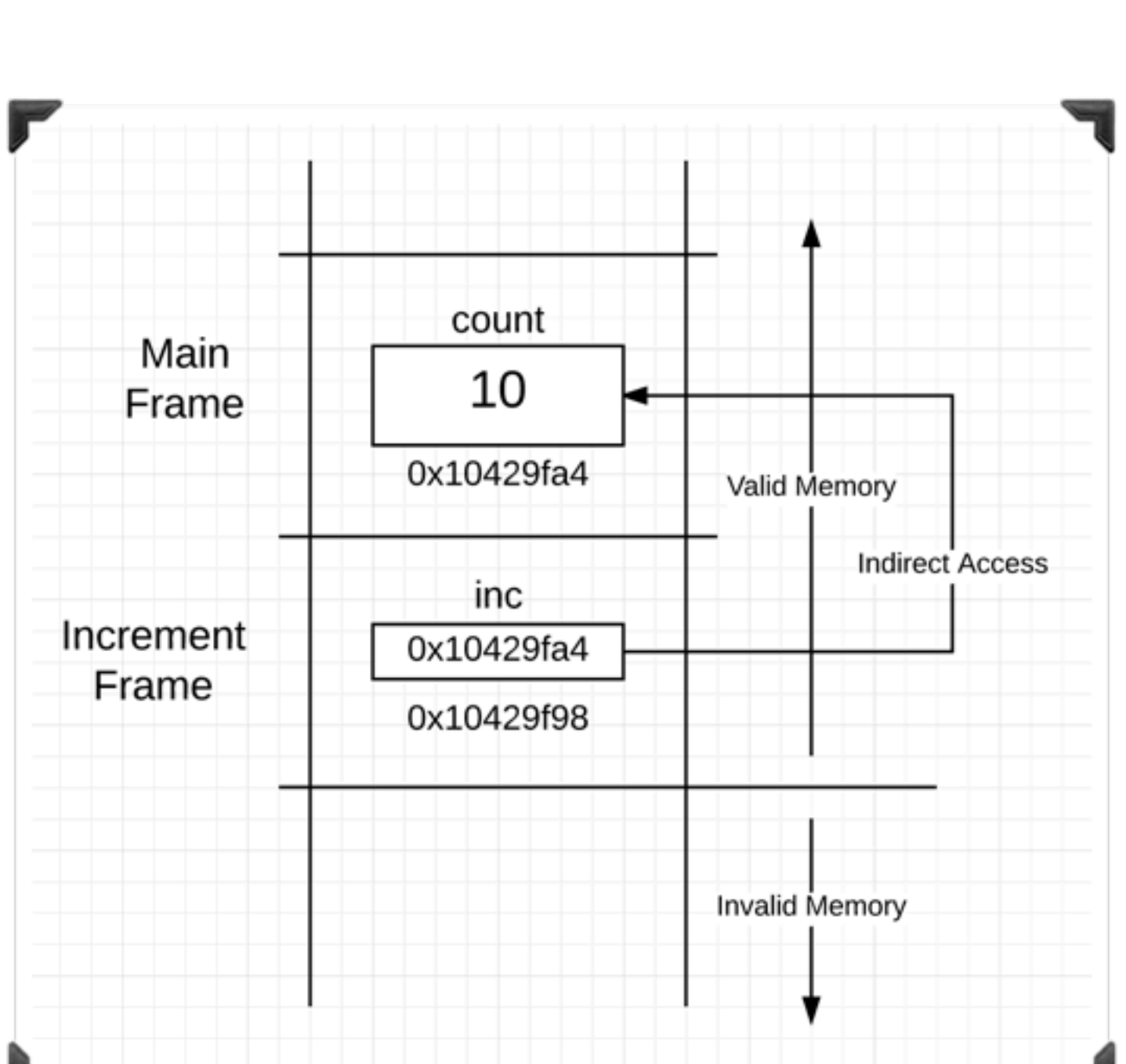
Pointers (Pass by Value)

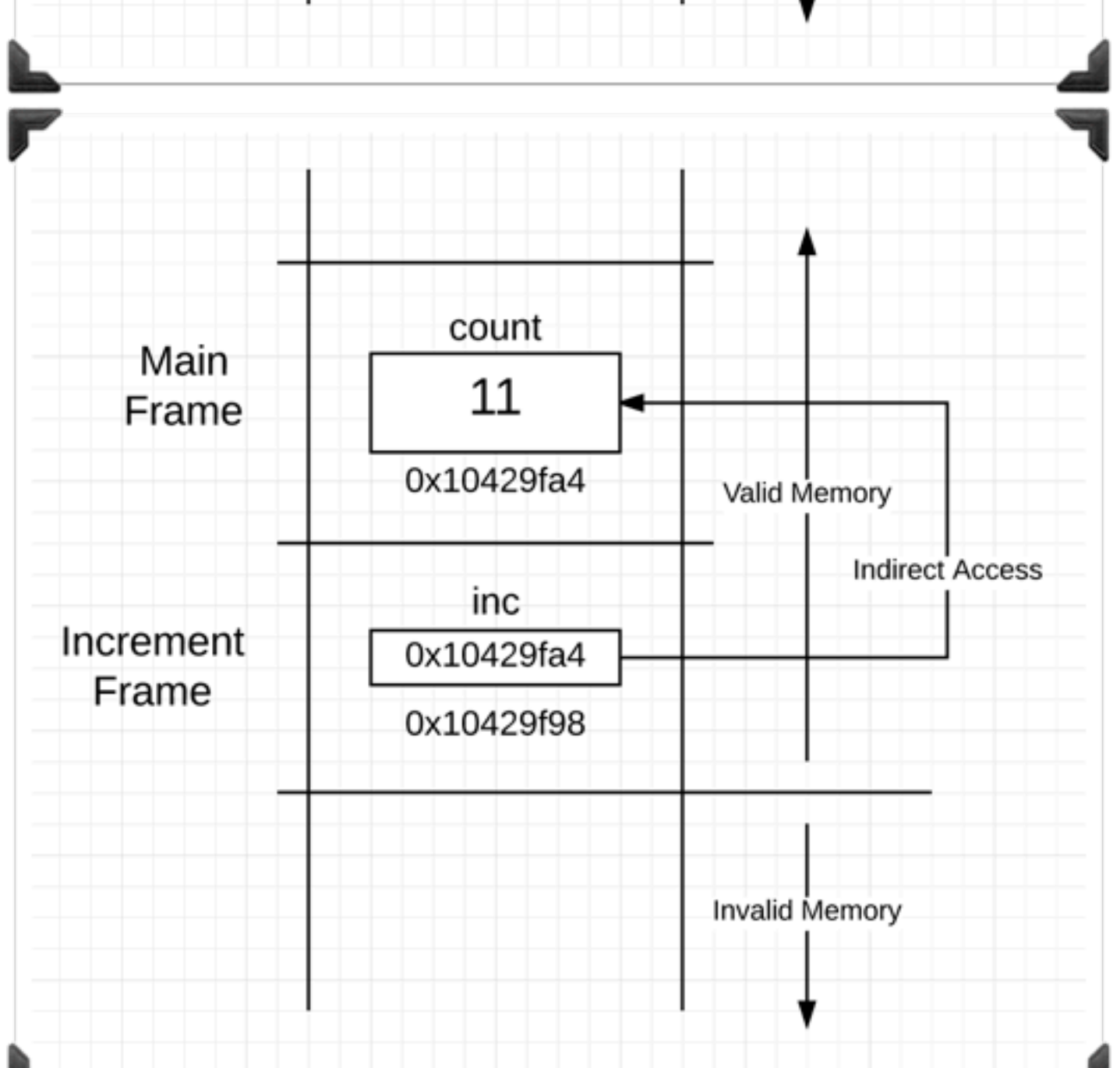






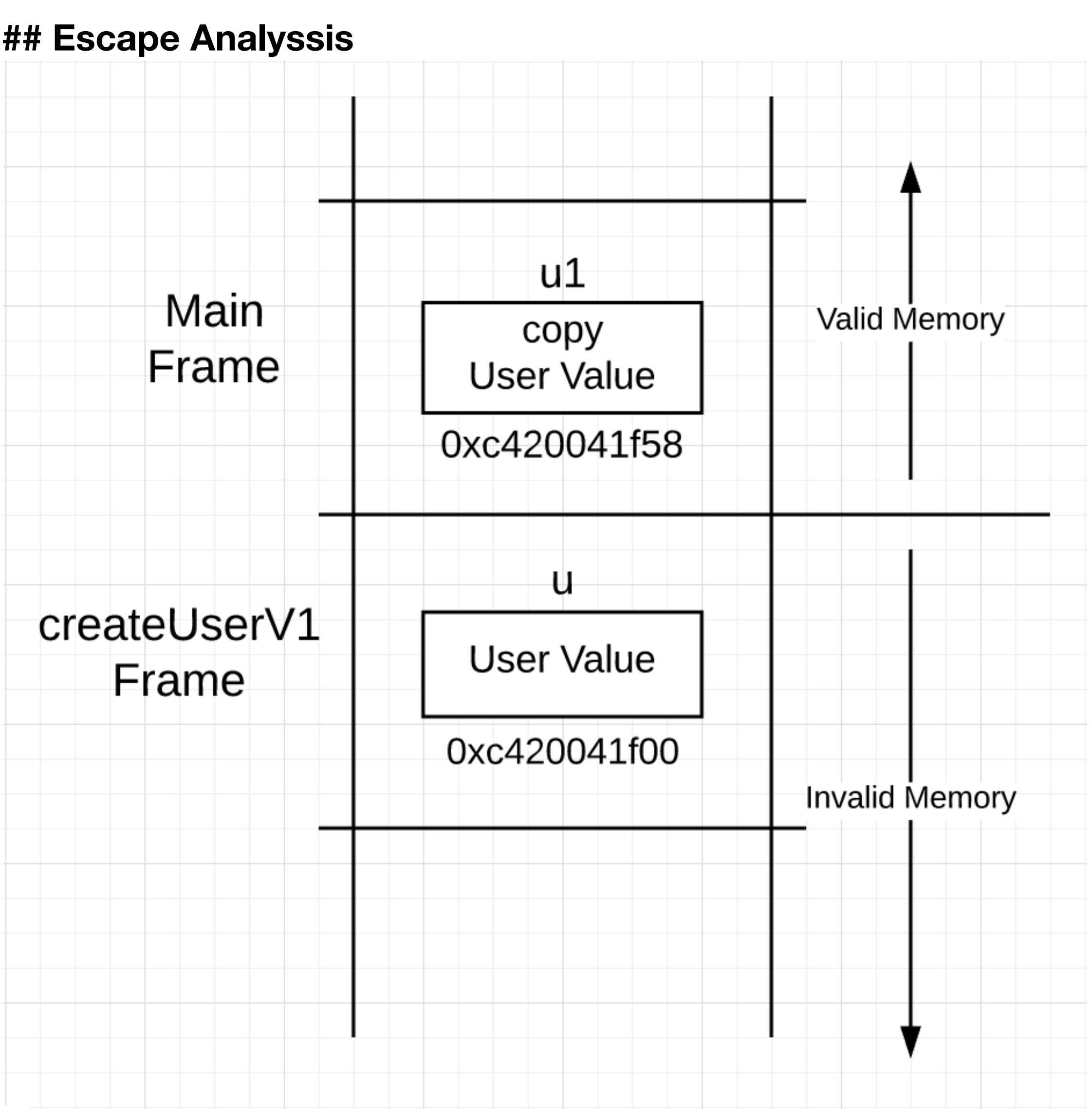
Sharing Data
*** => "The value that the pointer points to"**

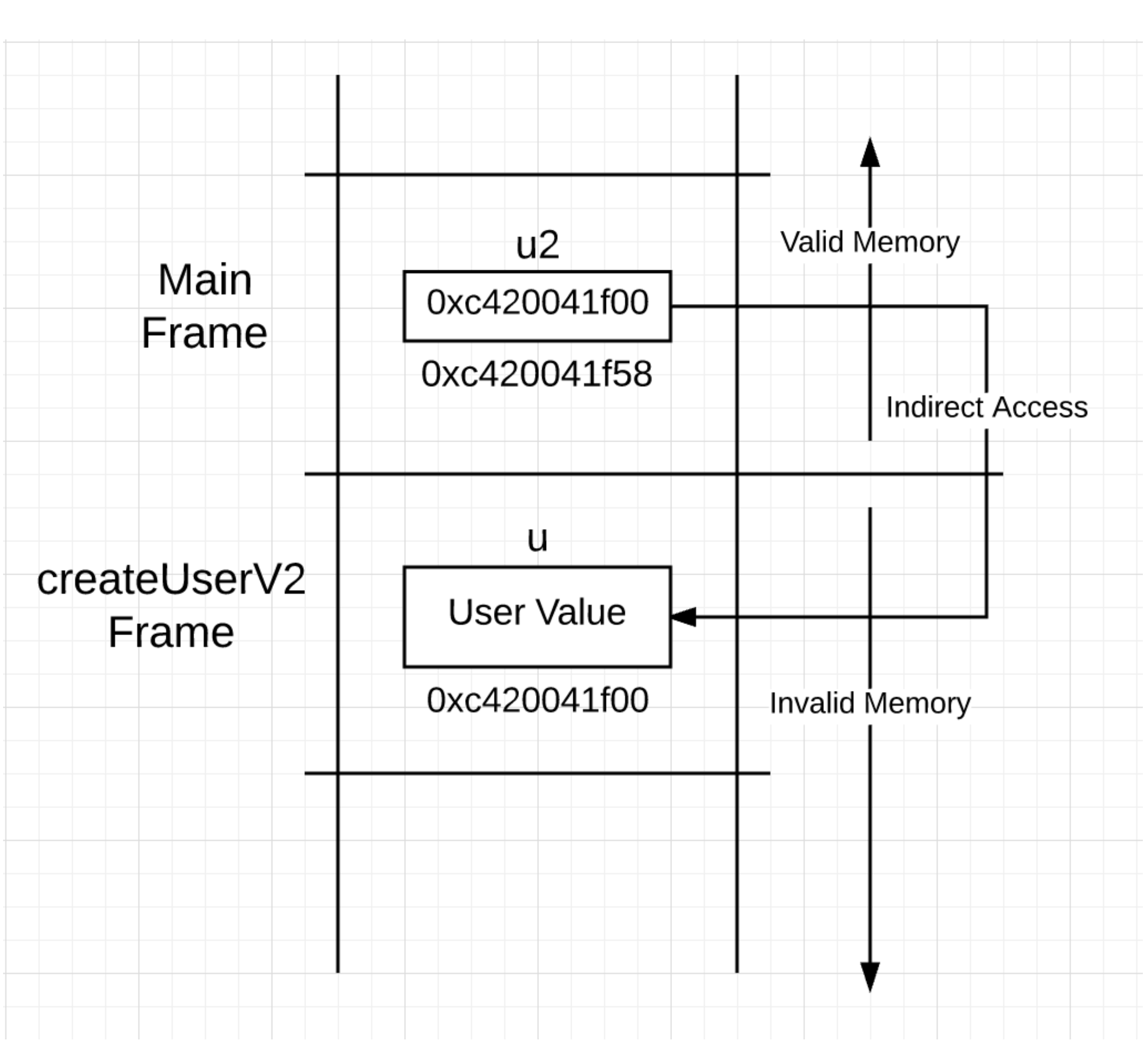


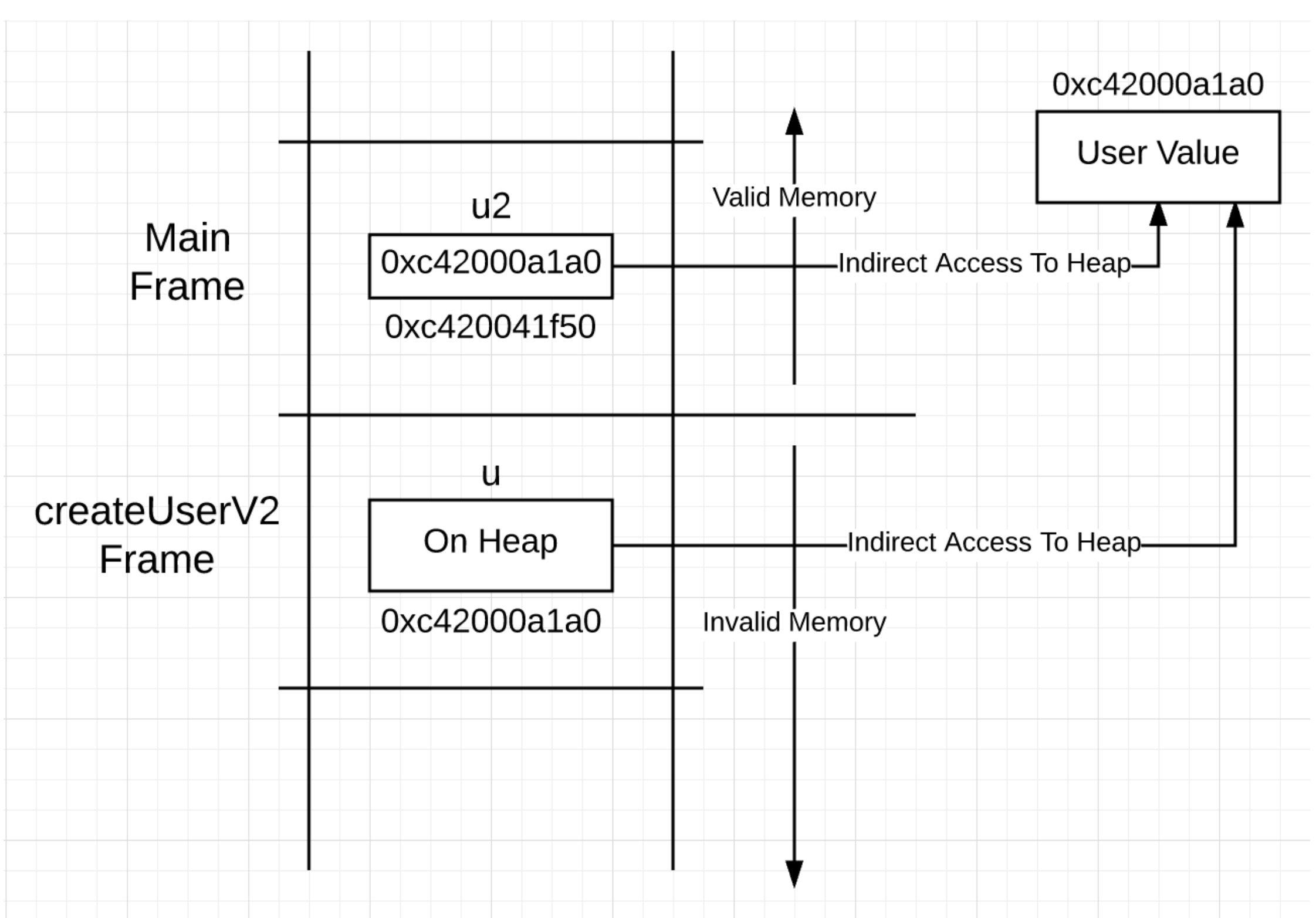


"Value semantics keep values on the stack, which reduces pressure on the Garbage Collector (GC). However, value semantics require various copies of any given value to be stored, tracked and maintained. Pointer semantics place values on the heap, which can put pressure on the GC. However, pointer semantics are efficient because only one value needs to be stored, tracked and maintained."

Escape Analyssis







Constants

```
## Constants

Constants are a way to create a named identifier whose value can never change. They also provide an incredible amount of flexibility to the language. The way constants are implemented in Go is very unique.
```