Using C Libraries in Swift 3

Andrew Morrow

COMCAST
VIPER

- **Advantages of C**
- How to import C headers
- How to call simple C functions
- How to manually manage memory
- How to call complex C functions

Advantages of C

- Large existing ecosystem
  - OpenGL (ES), OpenCV, SQLite, Accelerate, etc.
- Cross-platform
- Extreme high-performance

- **Advantages of C**
- How to import C headers
- How to call simple C functions
- How to manually manage memory
- How to call complex C functions

- Advantages of C
- **How to import C headers**
- How to call simple C functions
- How to manually manage memory
- How to call complex C functions

Modules

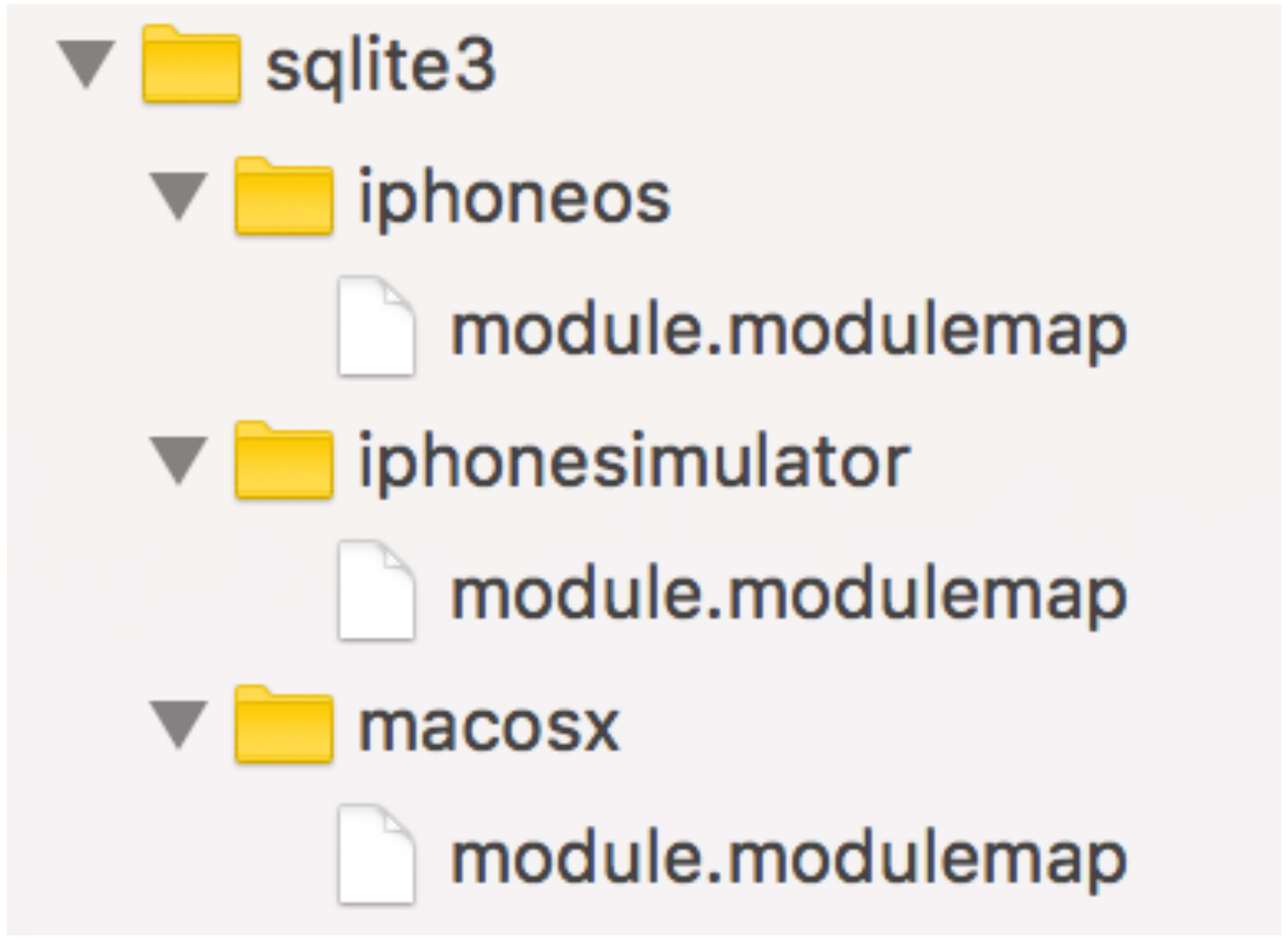
- How Swift “sees” C and Objective-C
- Binary representation of library API
- Defined by module maps

Module maps

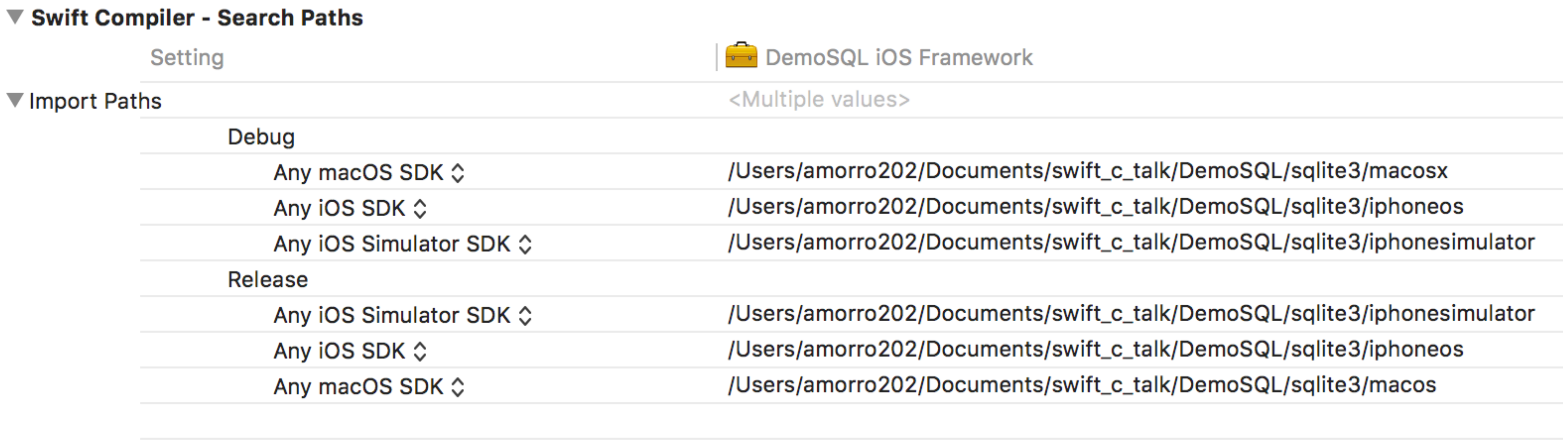
- Explicit list of headers and symbols
- Paired with existing header files
- Provided for *most* system libraries

```
module sqlite3 {
    header "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/
Developer/SDKs/MacOSX.sdk/usr/include/sqlite3.h"
    export *
}
```

One module map per SDK



```
sqlite3
    iphoneos
        module.modulemap
    iphonesimulator
        module.modulemap
    macosx
        module.modulemap
```



Swift Compiler - Search Paths

Setting	DemoSQL iOS Framework
Import Paths	<Multiple values>
Debug	
Any macOS SDK	/Users/amorro202/Documents/swift_c_talk/DemoSQL/sqlite3/macosx
Any iOS SDK	/Users/amorro202/Documents/swift_c_talk/DemoSQL/sqlite3/iphoneos
Any iOS Simulator SDK	/Users/amorro202/Documents/swift_c_talk/DemoSQL/sqlite3/iphonesimulator
Release	
Any iOS Simulator SDK	/Users/amorro202/Documents/swift_c_talk/DemoSQL/sqlite3/iphonesimulator
Any iOS SDK	/Users/amorro202/Documents/swift_c_talk/DemoSQL/sqlite3/iphoneos
Any macOS SDK	/Users/amorro202/Documents/swift_c_talk/DemoSQL/sqlite3/macos

No expectation of privacy

- All imported modules are re-exported publicly
- Library consumers can see and use imported libraries

- Advantages of C
- **How to import C headers**
- How to call simple C functions
- How to manually manage memory
- How to call complex C functions

- Advantages of C
- How to import C headers
- **How to call simple C functions**
- How to manually manage memory
- How to call complex C functions

Demo

C functions

```c
// sqlite3.h
SQLITE_API double sqlite3_column_double(sqlite3_stmt*, int iCol);
```

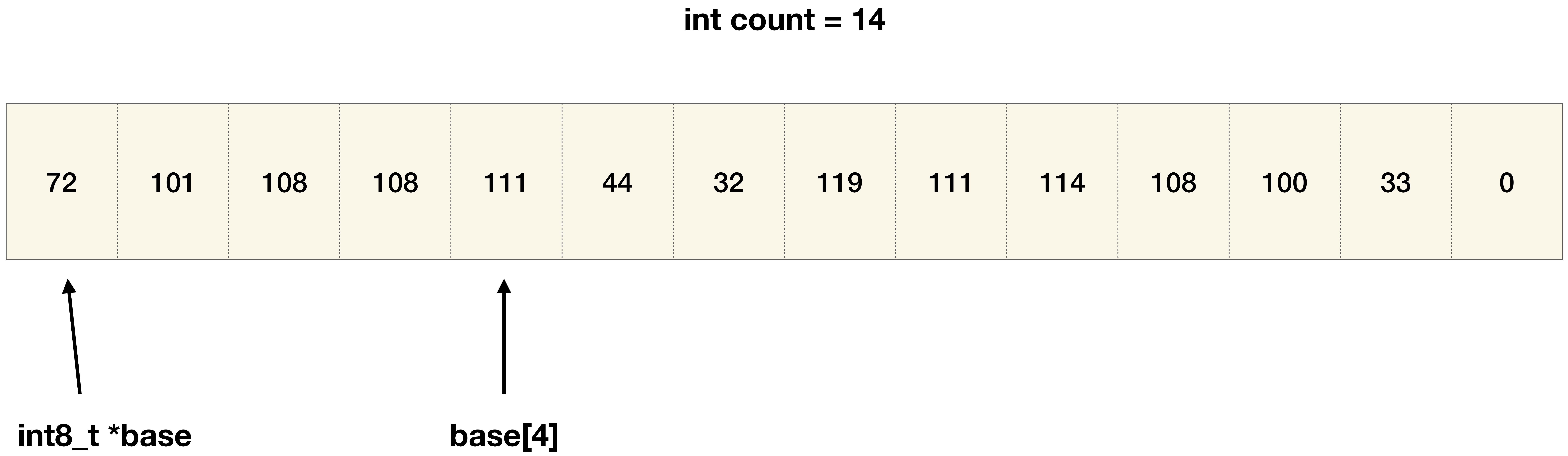
C functions

```
// Generated Swift interface
public func sqlite3_column_double(_: OpaquePointer!, _ iCol: Int32) -> Double
```

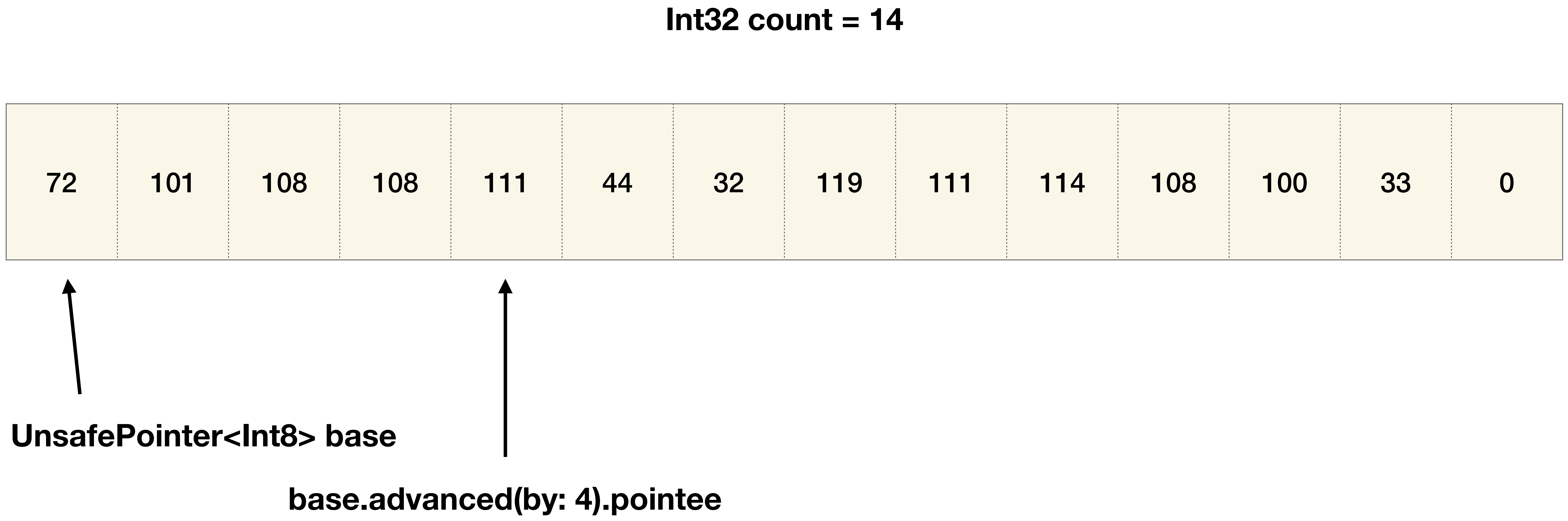
C Structs

```c
// Struct is opaque: no info about fields
typedef struct sqlite3_stmt sqlite3_stmt;

// Struct has known layout and size: all fields declared
typedef struct sqlite3_mem_methods sqlite3_mem_methods;
struct sqlite3_mem_methods {
  void *(*xMalloc)(int);
  void (*xFree)(void*);
  // ...truncated
};
```

C Arrays


int count = 14
72
101
108
108
111
44
32
119
111
114
108
100
33
0
int8_t *base
base[4]

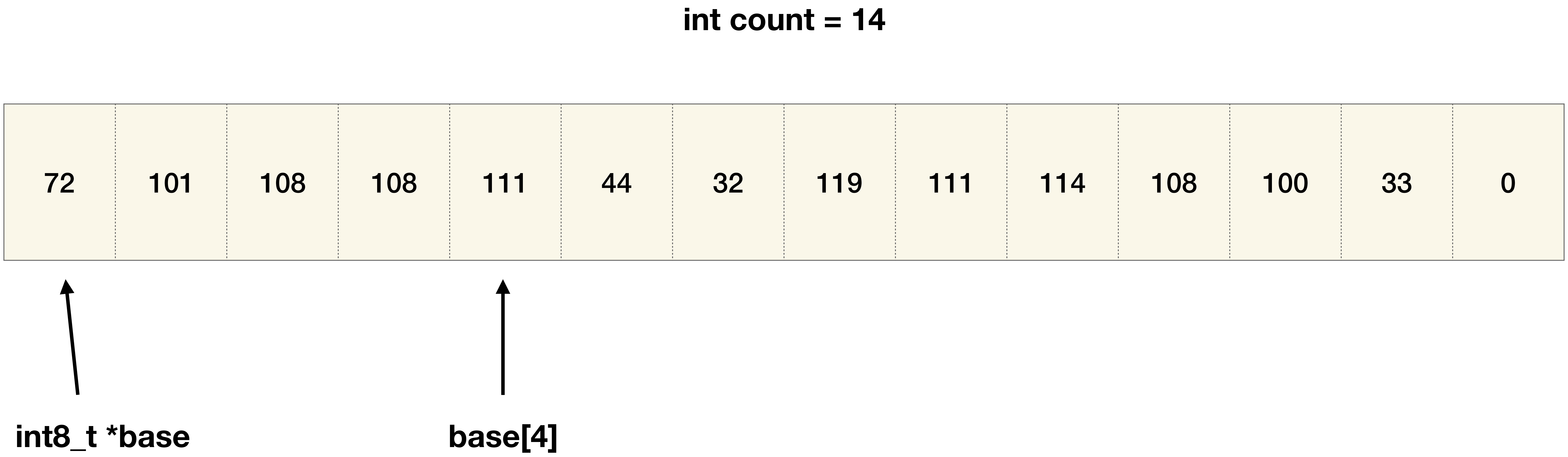
C Arrays in Swift



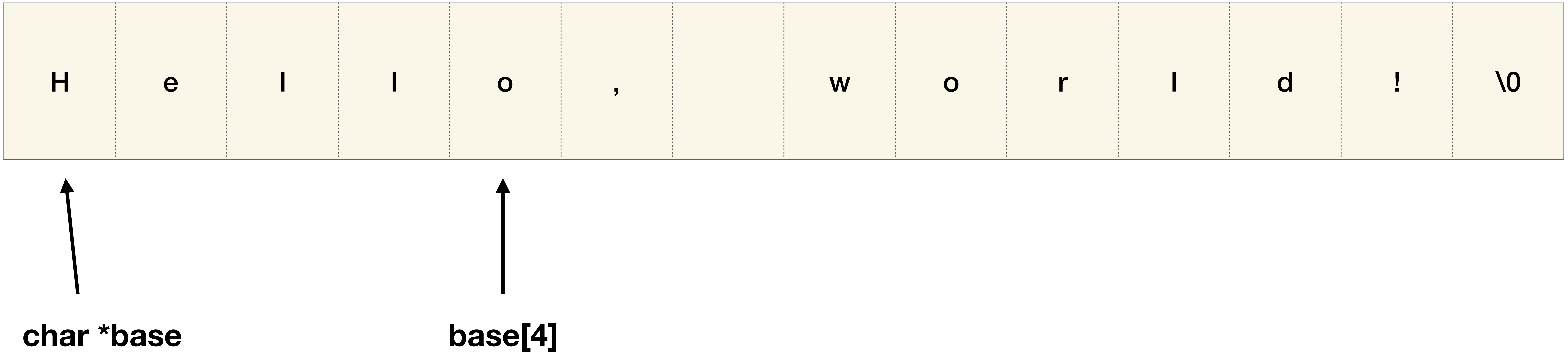
UnsafeBufferPointer

```swift
let base: UnsafePointer<Int8> = ... // from C function
let count: Int32 = ... // from C function

let buffer = UnsafeBufferPointer(start: base, count: Int(count))

// buffer is a Collection of Int8 (can index and slice)
let fifthElement = buffer[4]
// compute the sum using Collection methods
let sum: Int64 = buffer.reduce(0) { $0 + Int64($1) }

// copy the contents into Array<Int8>
let array = Array(buffer)
let array: [Int8] = [72, 101, 108, 108, 111, 44, 32, 119, 111, 114, 108,
100, 33, 0]

// Get a C array from a Swift Array
array.withUnsafeBufferPointer { (pointer: UnsafeBufferPointer<Int8>) in
    // Call a C function that takes an array
    let length = strlen(pointer.baseAddress!)
    print("The string is \(length) bytes long.")
}

// Compiler magic converts [T] to UnsafePointer<T>
let length = strlen(array)
print("The string is \(length) bytes long.”)
```

C Strings


int count = 14
72
101
108
108
111
44
32
119
111
114
108
100
33
0
int8_t *base
base[4]

C Strings



C Strings in Swift

```
let base: UnsafePointer<CChar> = ... // from C function
let string = String(cString: base, encoding: .utf8)
```

Swift strings

- `Swift.Character ≠ CChar`
- g + ◌̈ = g̈
- Must be encoded and null-terminated for C

```swift
let msg = "Hello, world!"

// Gets null-terminated [CChar], then borrows buffer pointer
msg.utf8CString.withUnsafeBufferPointer { buf in
    let len = strlen(buf.baseAddress)
}

// Borrows a pointer to a null-terminated UTF-8 string
msg.withCString { buf in
    let len = strlen(buf)
}

// Compiler automagic: String -> UnsafePointer<Int8>
let len = strlen(msg)
```

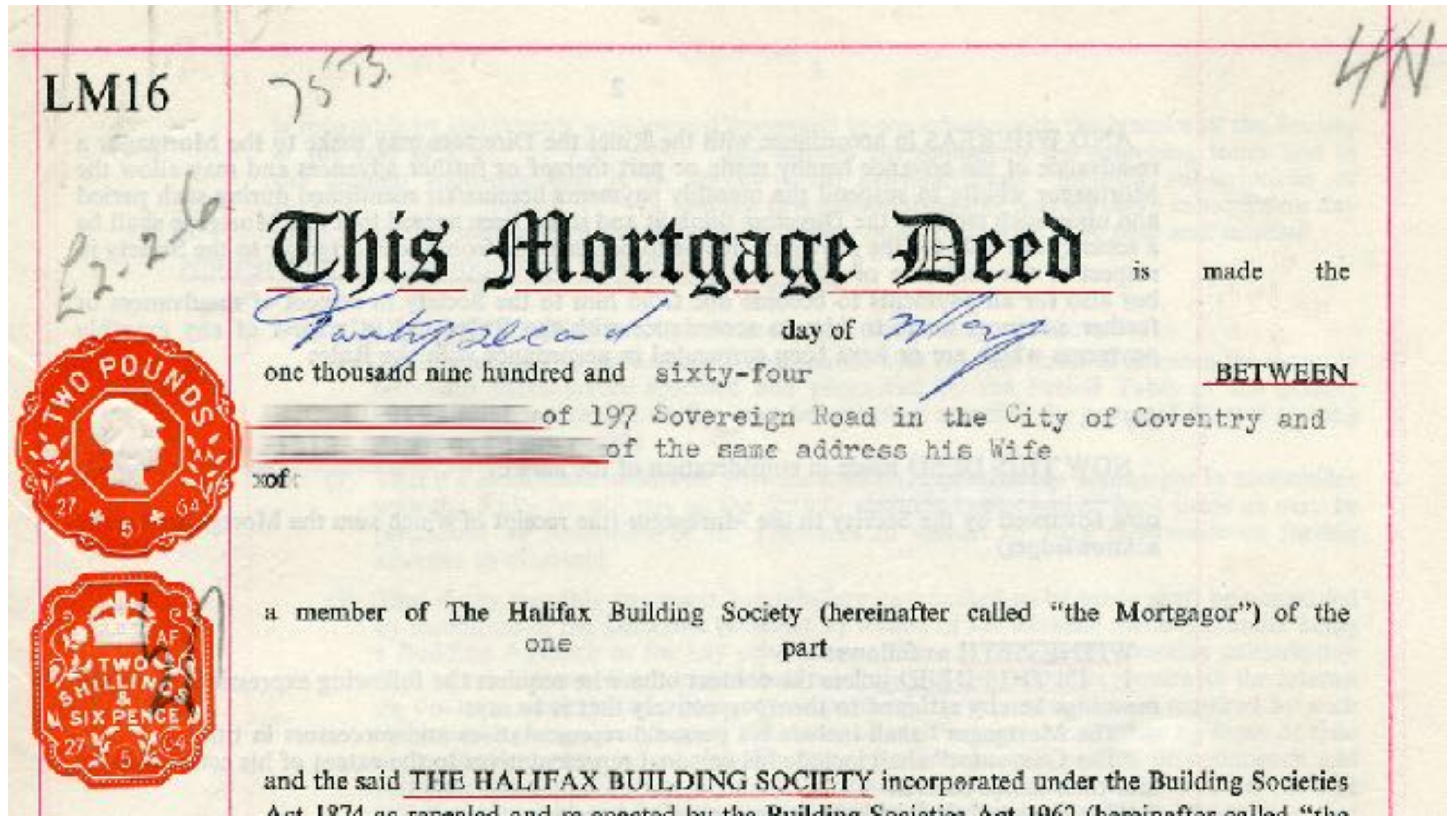
Demo

- Advantages of C
- How to import C headers
- **How to call simple C functions**
- How to manually manage memory
- How to call complex C functions

- Advantages of C
- How to import C headers
- How to call simple C functions
- **How to manually manage memory**
- How to call complex C functions

Ownership

- Memory is either owned or borrowed
  - `create / copy / alloc` are owned
- Owned memory must be freed or transferred
- Borrowed memory must not be freed
- Pointers must not be used once memory is freed



LM16 7573 4N

£2.2.6

TWO POUNDS 27 5 64

AF TWO SHILLINGS & SIX PENCE 27 5 64

This Mortgage Deed is made the Twenty second day of May one thousand nine hundred and sixty-four BETWEEN [redacted] of 197 Sovereign Road in the City of Coventry and [redacted] of the same address his Wife ~~of~~

a member of The Halifax Building Society (hereinafter called "the Mortgagor") of the one part

and the said THE HALIFAX BUILDING SOCIETY incorporated under the Building Societies Act 1874 as repealed and re-enacted by the Building Societies Act 1962 (hereinafter called "the

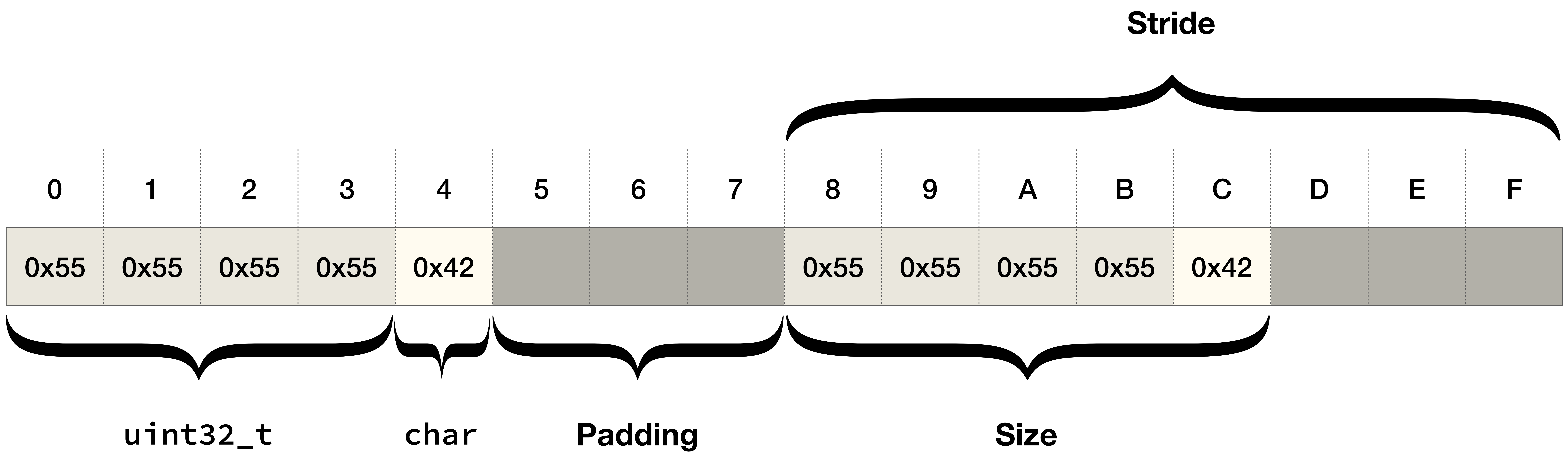
Re-binding memory

$$\begin{tikzcd}
 & \text{0x42} \arrow[dl] \arrow[dr] & \\
66 & & \text{B} \\
\text{uint} & & \text{char}
\end{tikzcd}$$

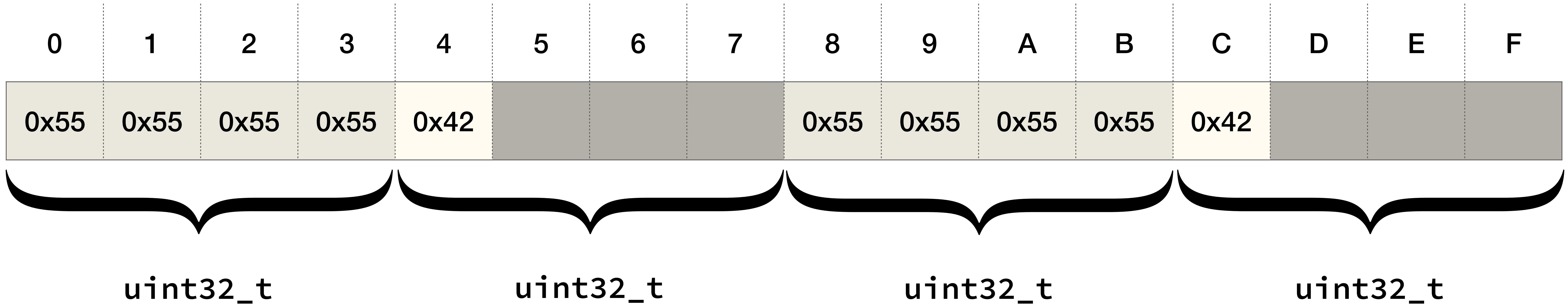
Memory layout

```
typedef struct {
        uint32_t number;
        char letter;
} NumberAndLetter;
```

Memory layout



Memory layout



Prefer ARC!

- `Data.init(bytes:)`
- `Data.init(bytesNoCopy: count: deallocator:)`
- `String.init(bytes: encoding:)`
- `Array.init(_:)`

- Advantages of C
- How to import C headers
- How to call simple C functions
- **How to manually manage memory**
- How to call complex C functions

- Advantages of C
- How to import C headers
- How to call simple C functions
- How to manually manage memory
- **How to call complex C functions**

```c
// sqlite3.h
SQLITE_API int sqlite3_exec(
  sqlite3*,                                  /* An open database */
  const char *sql,                           /* SQL to be evaluated */
  int (*callback)(void*,int,char**,char**),  /* Callback function */
  void *,                                    /* 1st argument to callback */
  char **errmsg                              /* Error msg written here */
);
```

```
// Generated Swift interface
public func sqlite3_exec(
  _: OpaquePointer!,                /* An open database */
  _ sql: UnsafePointer<Int8>!,      /* SQL to be evaluated */
  _ callback: (@convention(c)       /* Callback function */
    (UnsafeMutableRawPointer?,
     Int32,
     UnsafeMutablePointer<UnsafeMutablePointer<Int8>?>?,
     UnsafeMutablePointer<UnsafeMutablePointer<Int8>?>?) -> Int32)!,
  _: UnsafeMutableRawPointer!,      /* 1st argument to callback */
  _ errmsg:                         /* Error msg written here */
     UnsafeMutablePointer<UnsafeMutablePointer<Int8>?>!
) -> Int32
```

Context pointers

- Opaque pointers passed through C APIs
- Provide mutable execution context for callbacks
- `Unmanaged` converts `AnyObject` to `UnsafeRawPointer`
- Manually managed memory

Demo

- Advantages of C
- How to import C headers
- How to call simple C functions
- How to manually manage memory
- **How to call complex C functions**

Q & A

Resources

- **Apple documentation**
  - https://developer.apple.com/library/content/documentation/Swift/Conceptual/BuildingCocoaApps/InteractingWithCAPIs.html
- **Umberto Raimo's reference**
  - https://www.uraimo.com/2016/04/07/swift-and-c-everything-you-need-to-know/
- **Chris Eidhof's function pointers tutorial**
  - http://chris.eidhof.nl/post/swift-c-interop/
- **SQLite.swift open-source project**
  - https://github.com/stephencelis/SQLite.swift
- **Clang modules documentation**
  - https://clang.llvm.org/docs/Modules.html

Thank you!

COMCAST VIPER

Bonus: Unsafe type casts

- Swift prohibits invalid type casts
- C allows all type casts
- `unsafeBitCast` to the rescue!

```
public func bindData(_ value: Data, at idx: Int) throws {
    try value.withUnsafeBytes { (bytes: UnsafePointer<UInt8>) -> Void in

        let sqliteTransient = unsafeBitCast(-1,
          to: (@convention(c) (UnsafeMutableRawPointer?) -> Void).self)

        let statusCode = sqlite3_bind_blob(statementHandle,
                                           Int32(idx),
                                           bytes,
                                           Int32(value.count),
                                           sqliteTransient)

        guard statusCode == SQLITE_OK else {
            throw DatabaseError(code: statusCode,
                                message: parentDatabase.errorMessage)
        }
    }
}
```