MachO linker in Zig

Linking in the era of Apple Silicon

Who is this guy?

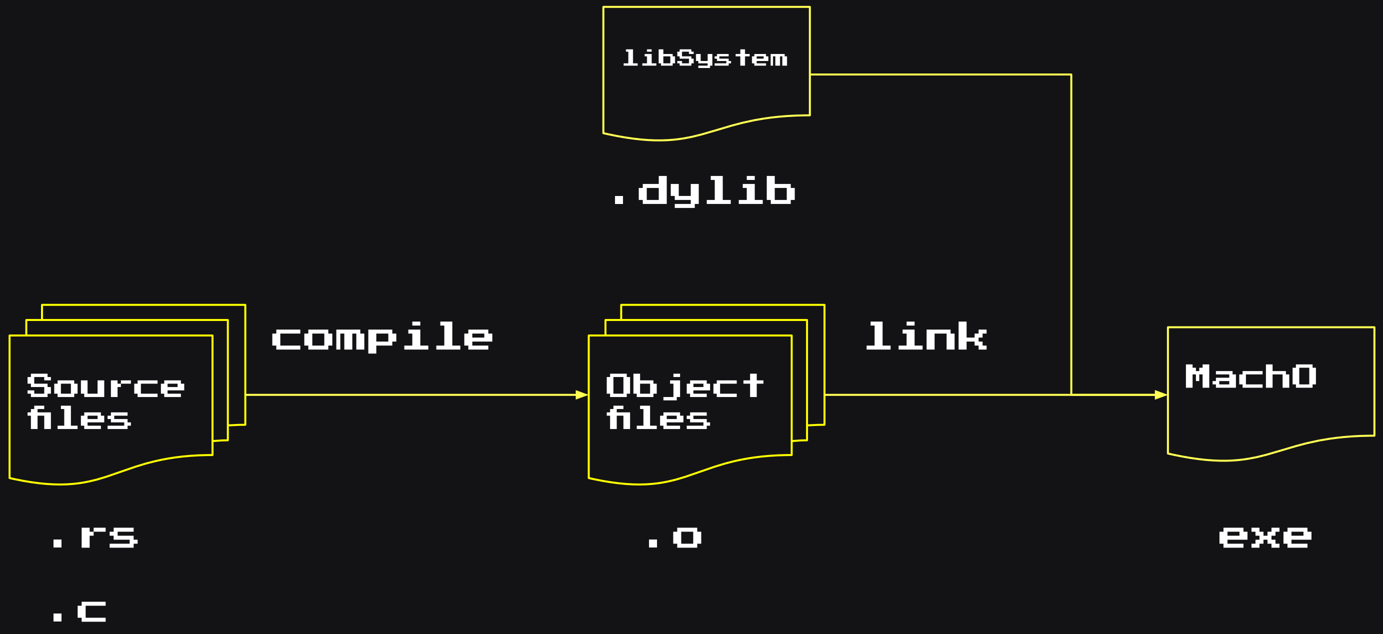
- → Name is Jakub “kubkon” Konka
- → Software engineer at Microsoft
- → Zig core team member

The agenda

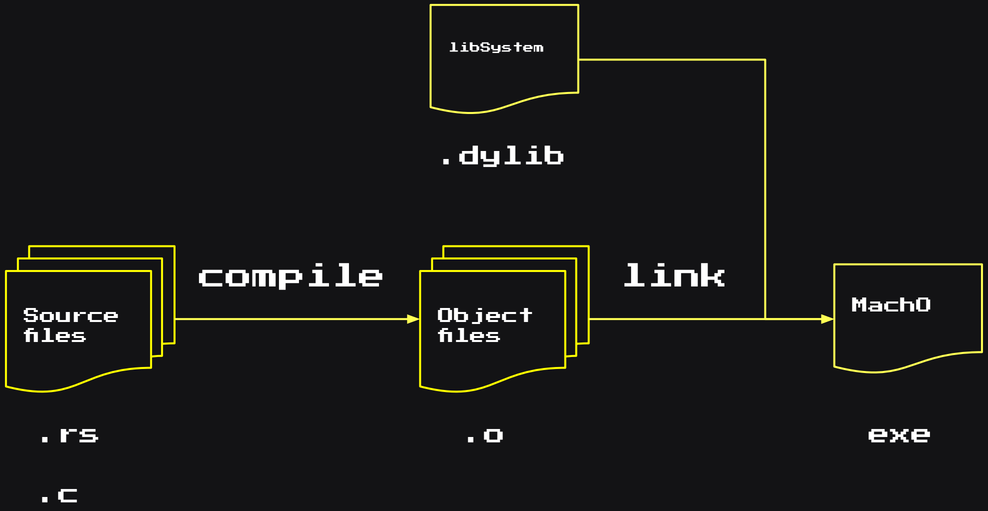
- → What is linking anyway?
- → Meet MachO!
- → Apple Silicon hard requirements
- → Incremental linking in Zig
- → **BONUS:** cross-compiling C to Apple Silicon with Zig

What is linking anyway?

Traditional approach

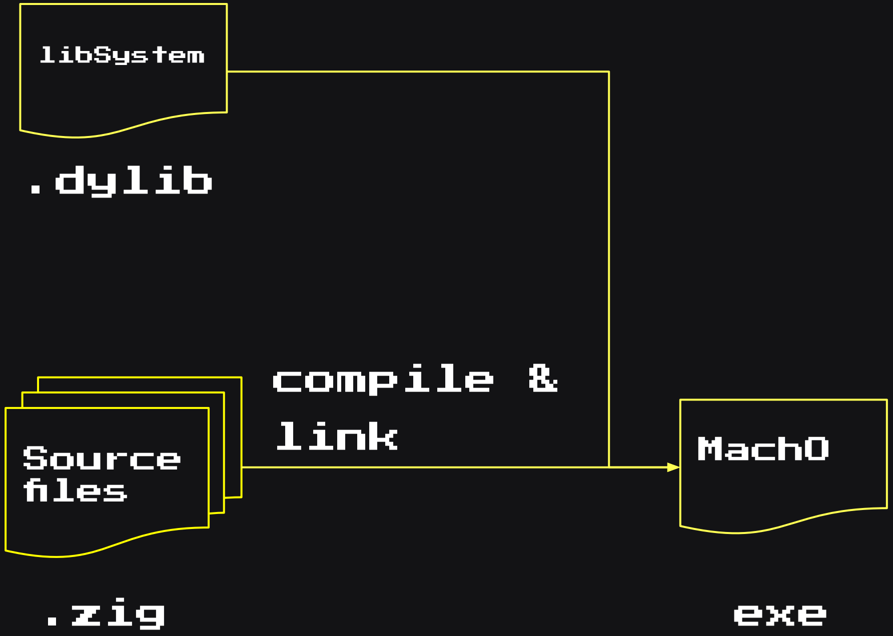


Traditional approach

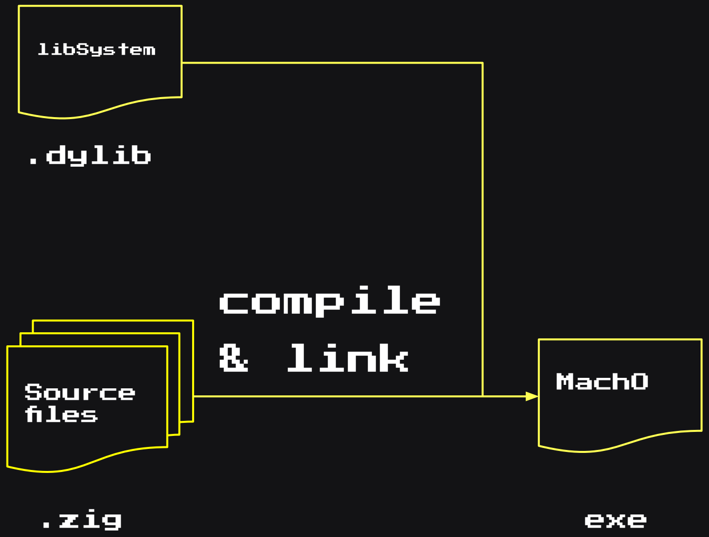


- → Compilation separate from linking
- → Source changes always lead to full relinking
- → LLVM-favoured

Zig's self-hosted approach



Zig's self-hosted approach



- → Compilation coupled with linking
- → Incremental linking means editing final exe in-place
- → Shorter turnaround time

Incremental linking

- → Only in Debug mode
- → Shorter compilation times
- → Means quicker prototyping
- → Write, compile, change, recompile...
- → Larger artifacts
- → No LTO

Meet MachO!

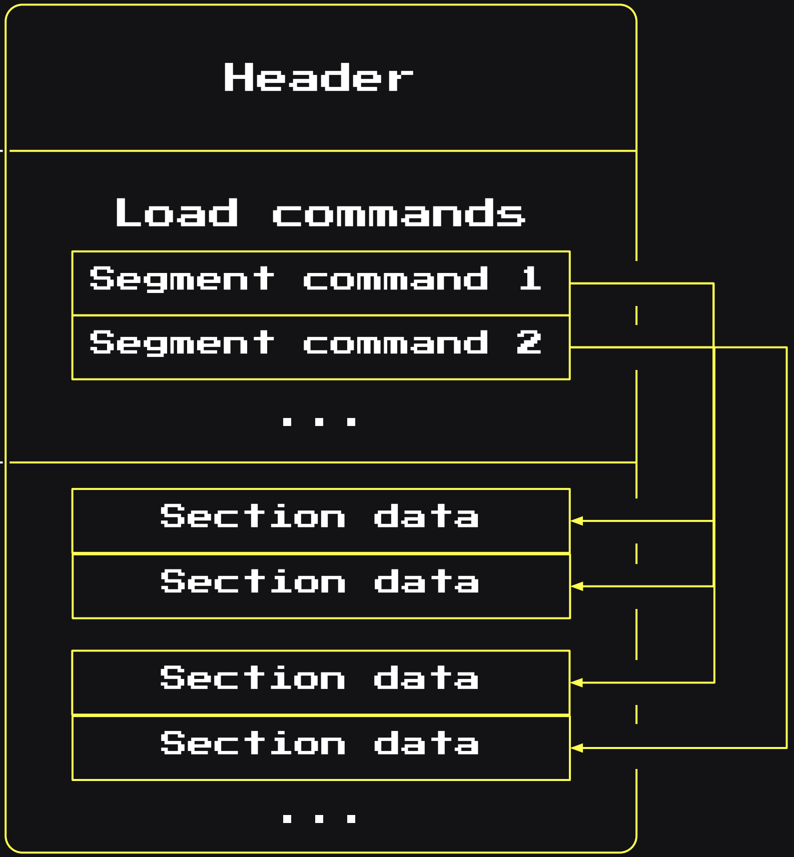
What is MachO?

- → MachO, or Mach object file format
- → Used by all Apple OSes
- → File structure is straightforward
- → No up-to-date docs
- → Incremental linking is difficult

- Magic number
- Architecture
- File type

- Description of segments in file
- Mapping into virtual memory

- Actual data mapped into memory
- Compiled machine code
- Stubs trampolines
- Loader info



Mach header

```
> otool -h hello

hello:
Mach header
magic          ... filetype     ncmds sizeofcmds flags
0xfeedfacf         MH_EXECUTE   15    1256       MH_PIE
```

Segment command

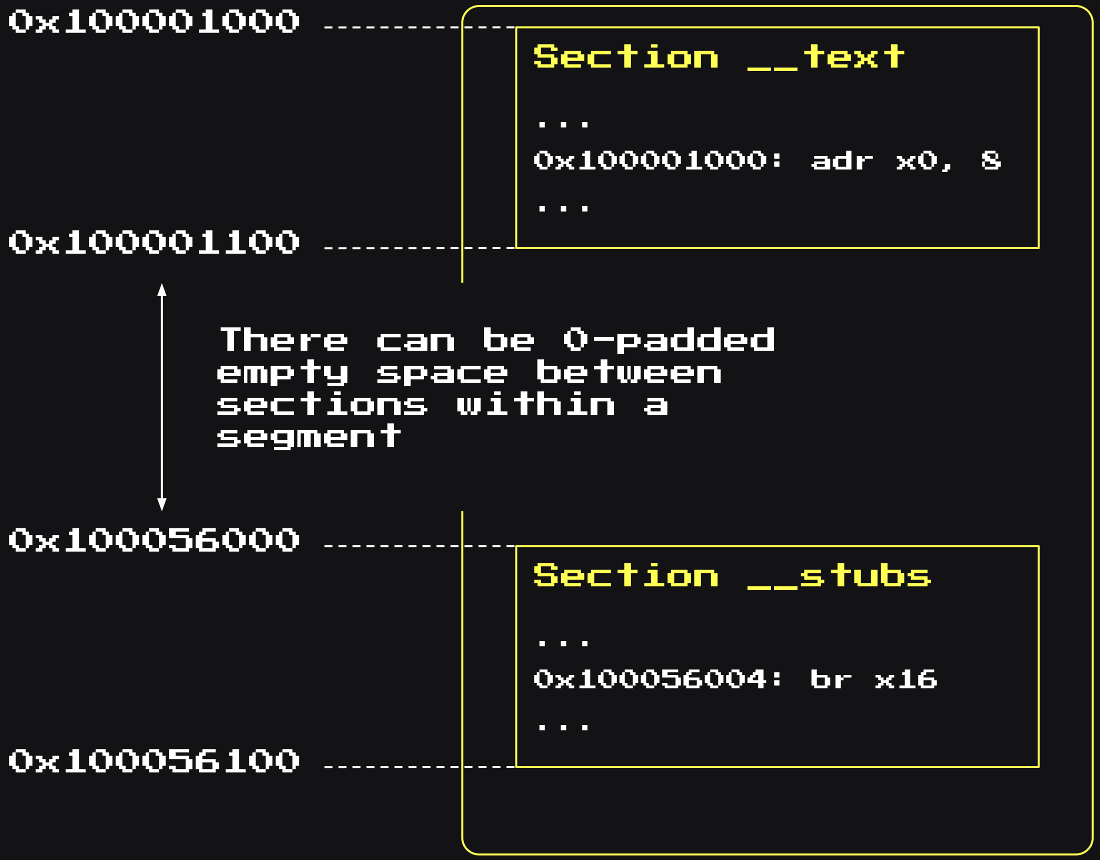
```
> otool -lv hello

hello:
Load command 1
      cmd   LC_SEGMENT_64
  cmdsize   392
  segname   __TEXT
   vmaddr   0x100000000
   vmsize   0x58000
  fileoff   0
 filesize   0x58000
  maxprot   rwx
 initprot   r-x
   nsects   4
```

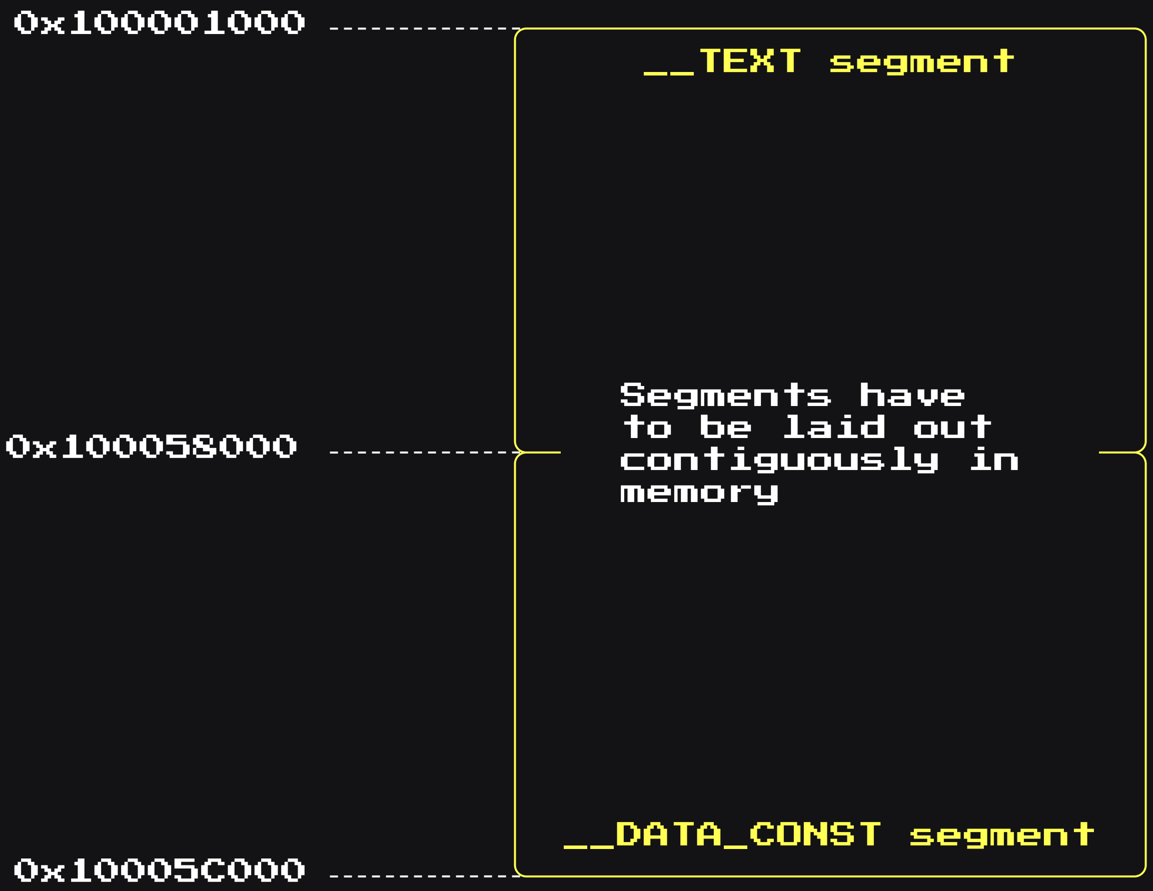
Segment command

```
> otool -lv hello

hello:
Load command 1
      cmd   LC_SEGMENT_64
  cmdsize   392                          0x8 aligned
  segname   __TEXT
   vmaddr   0x100000000
   vmsize   0x58000                     page aligned
  fileoff   0
 filesize   0x58000                     page aligned
  maxprot   rwx
 initprot   r-x                    segment dependent
   nsects   4
```

Segment command

```
> otool -lv hello

hello:
Load command 1
      cmd   LC_SEGMENT_64
  cmdsize   392                    0x8 aligned
  segname   __TEXT
   vmaddr   0x100000000
   vmsize   0x58000             0x4000 aligned (ARM64)
  fileoff   0
 filesize   0x58000             0x4000 aligned (ARM64)
  maxprot   rwx
 initprot   r-x            segment dependent
   nsects   4
```


0x100001000
Section __text
...
0x100001000: adr x0, 8
...
0x100001100
There can be 0-padded empty space between sections within a segment
0x100056000
Section __stubs
...
0x100056004: br x16
...
0x100056100


0x100001000
__TEXT segment
0x100058000
Segments have to be laid out contiguously in memory
__DATA_CONST segment
0x10005C000

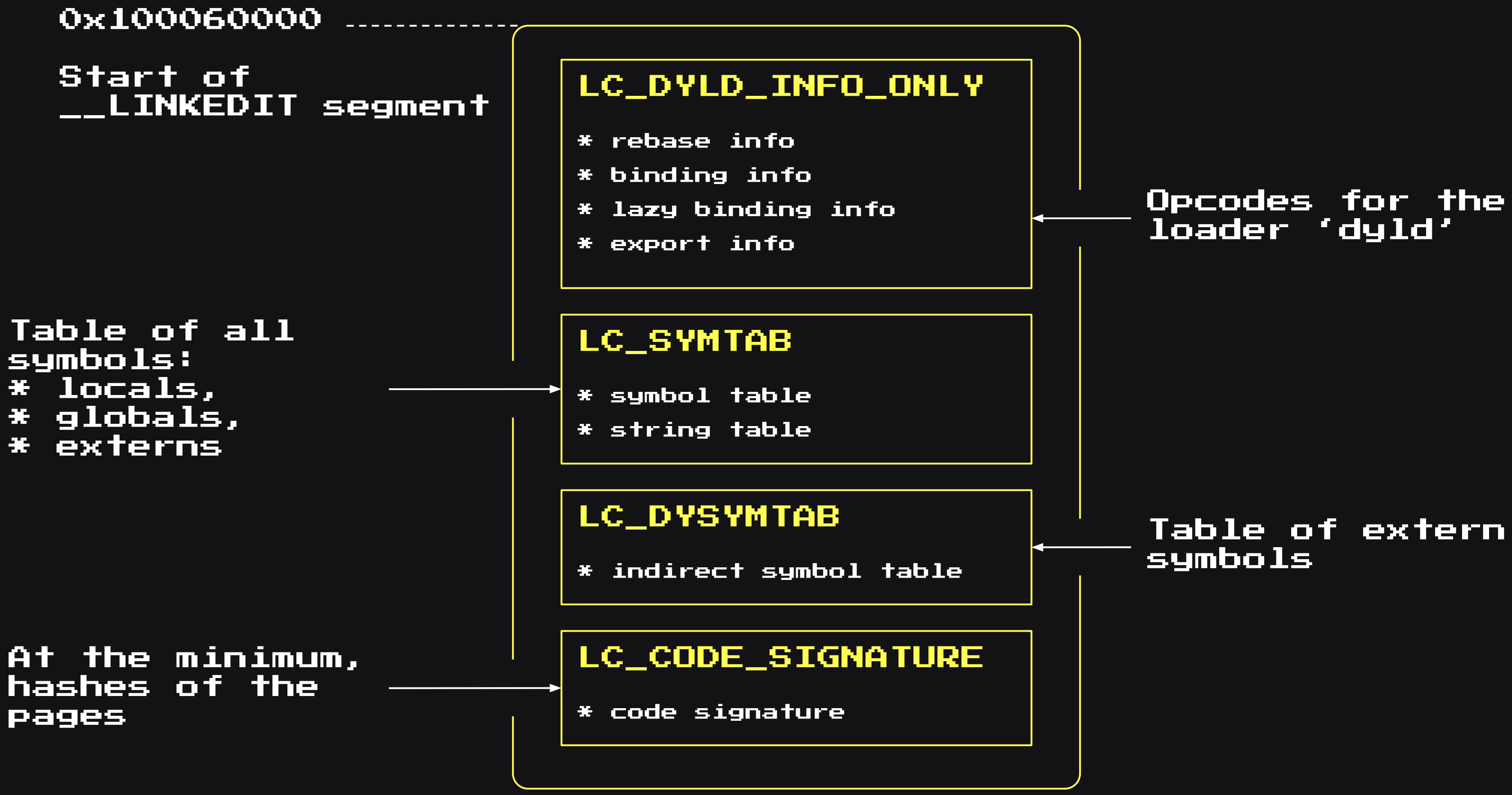
The __LINKEDIT segment

```
> otool -lv hello

hello:
Load command 4
      cmd  LC_SEGMENT_64
  cmdsize  72
  segname  __LINKEDIT
   vmaddr  0x100060000
   vmsize  0x4000
  fileoff  0
 filesize  0x10B2
 maxprot  rwx
initprot  r--
   nsects  0
```

The __LINKEDIT segment

```
> otool -lv hello

hello:
Load command 4
      cmd  LC_SEGMENT_64
  cmdsize  72
  segname  __LINKEDIT
   vmaddr  0x100060000
   vmsize  0x4000                  page aligned
  fileoff  0
 filesize  0x10B2                  not page aligned!
  maxprot  rwx
 initprot  r--
   nsects  0                       no sections?!
```

__LINKEDIT sections get their own load commands


0x100060000
Start of __LINKEDIT segment
LC_DYLD_INFO_ONLY
* rebase info
* binding info
* lazy binding info
* export info
Opcodes for the loader 'dyld'
Table of all symbols:
* locals,
* globals,
* externs
LC_SYMTAB
* symbol table
* string table
LC_DYSYMTAB
* indirect symbol table
Table of extern symbols
At the minimum, hashes of the pages
LC_CODE_SIGNATURE
* code signature

Apple Silicon hard requirements

Binary has to be...

- → Adhoc code signed
- → A Position Independent Executable (PIE)
- → Moved to a new inode if modified in-place

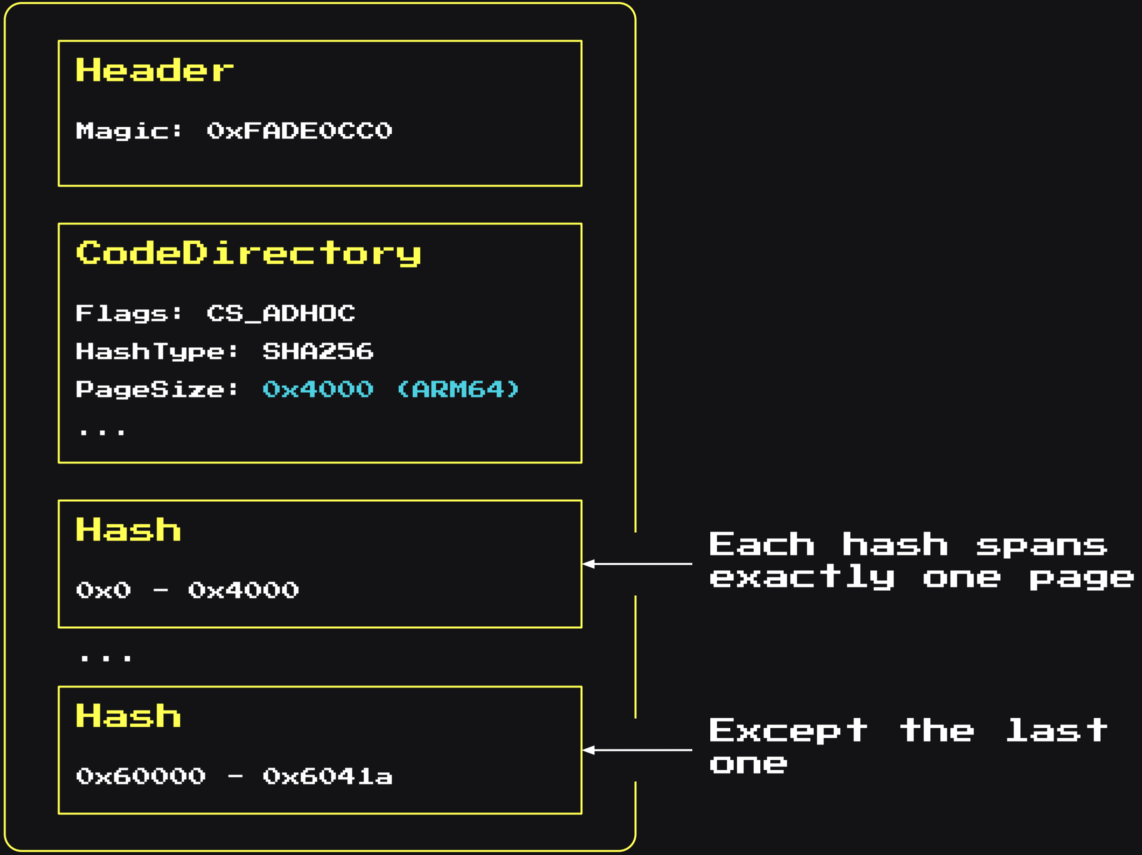
Adhoc code signature

→ Structure embedded as the last __LINKEDIT section

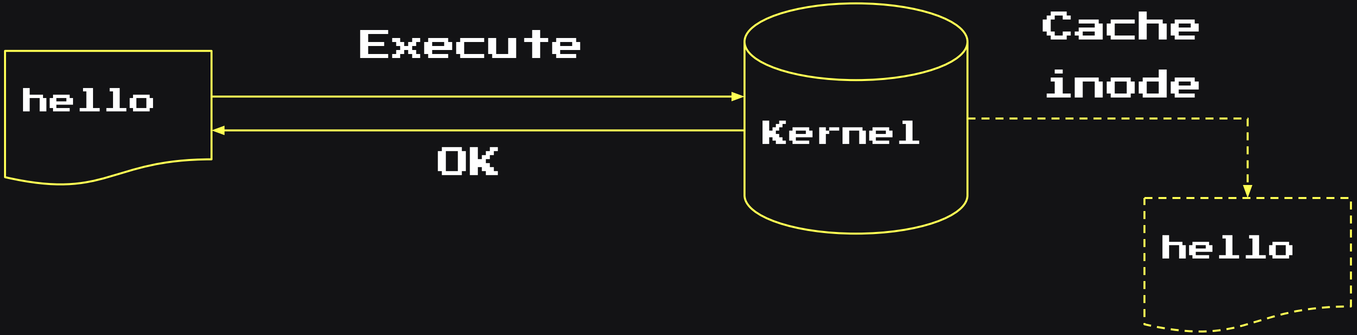
→ Pointed to by LC_CODE_SIGNATURE load command

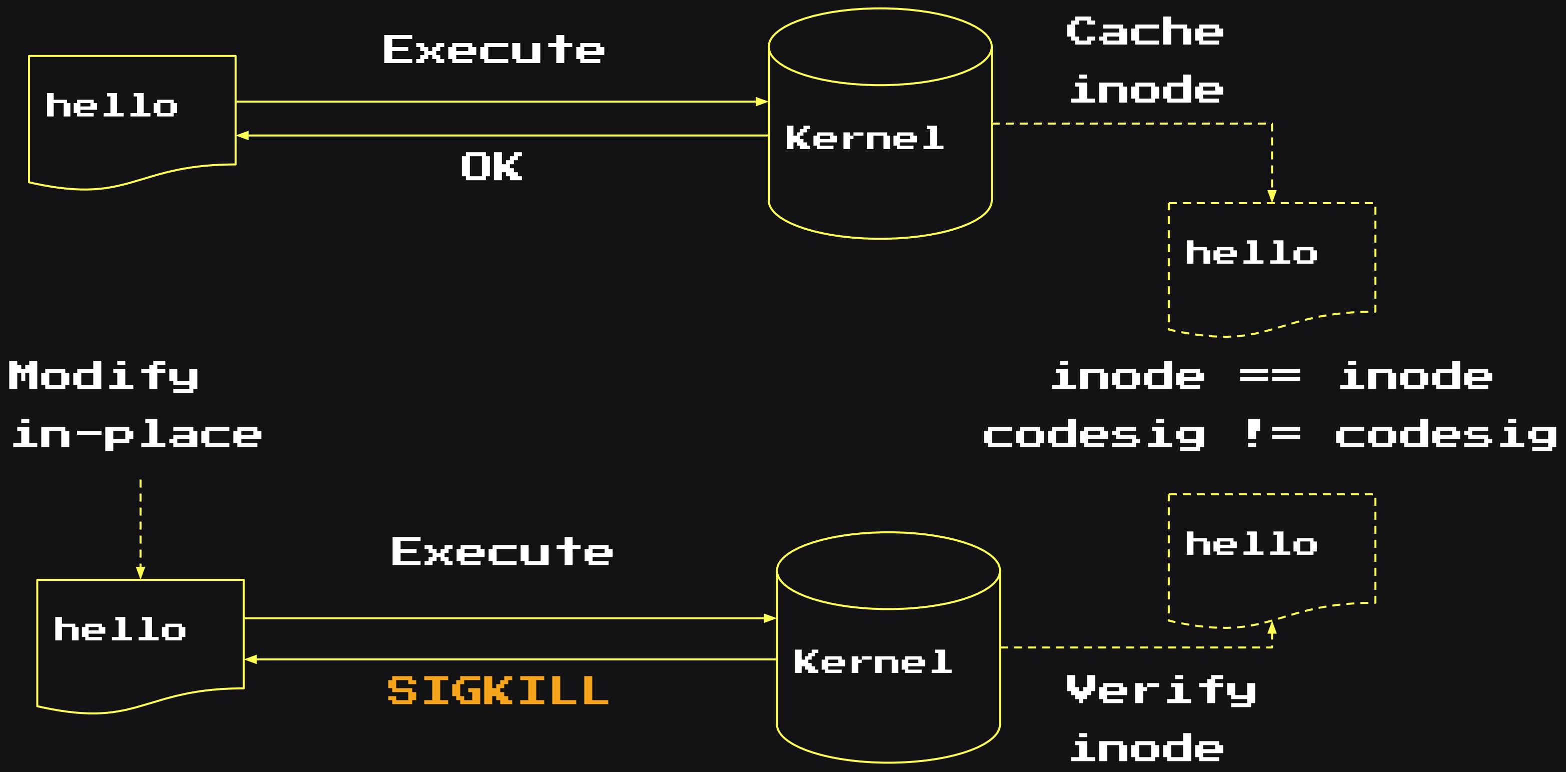
→ Contains hashes of the entire binary

→ In Big endian order!


Header
Magic: 0xFADE0CC0
CodeDirectory
Flags: CS_ADHOC
HashType: SHA256
PageSize: 0x4000 (ARM64)
...
Hash
0x0 - 0x4000
...
Hash
0x60000 - 0x6041a
Each hash spans exactly one page
Except the last one

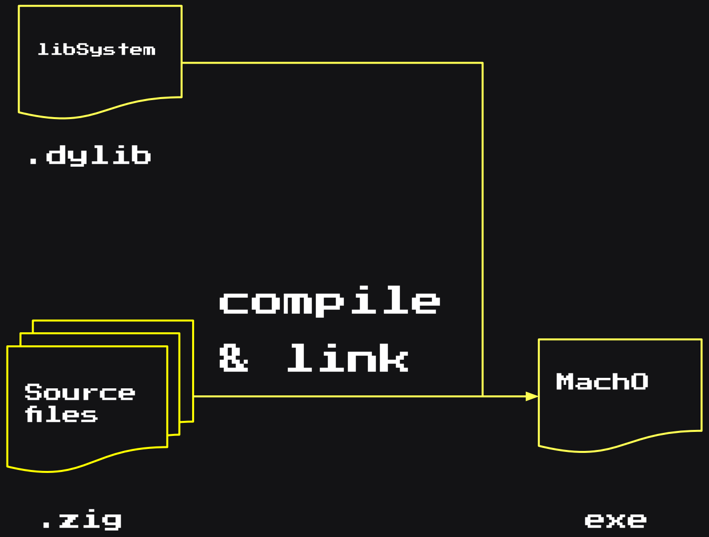
Since macOS 11, on Apple Silicon, kernel caches inodes


hello
Execute
OK
Kernel
Cache
inode
hello

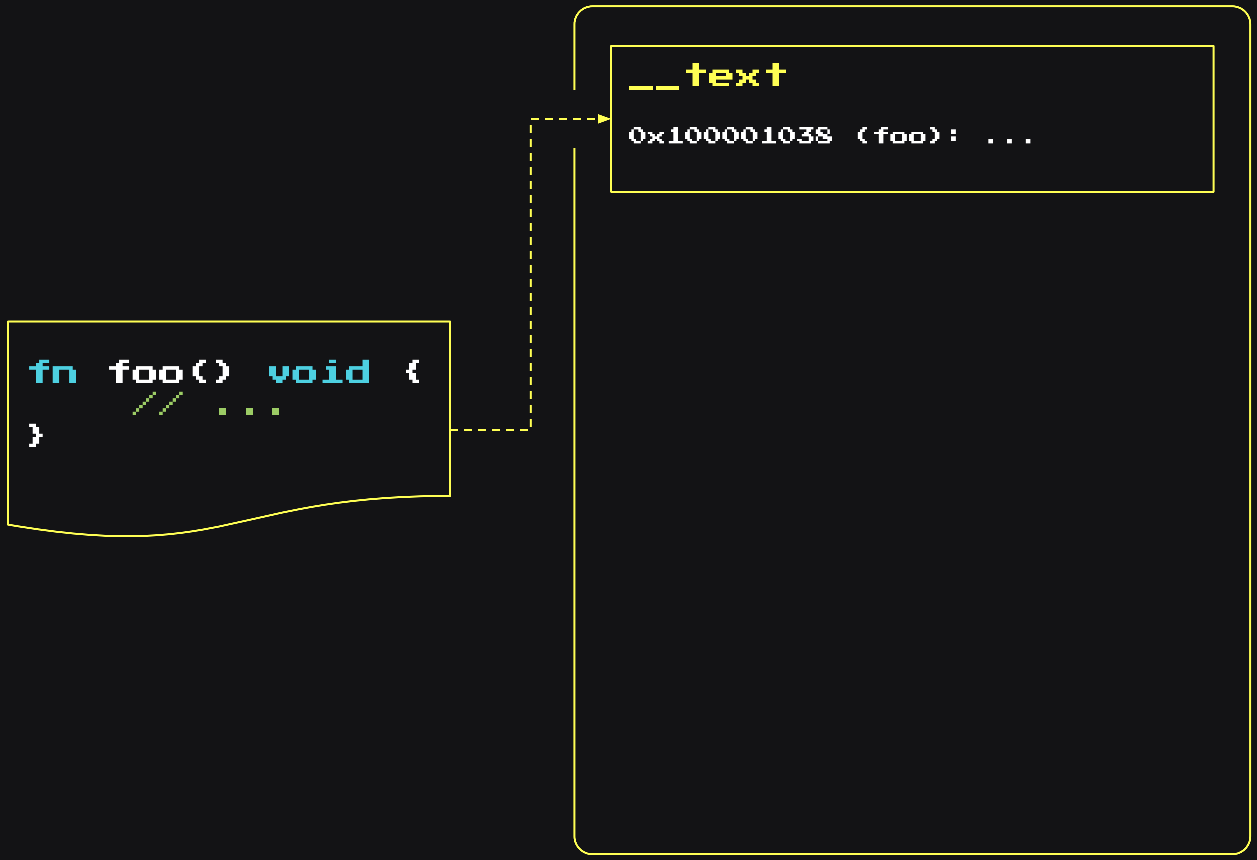

hello
Execute
OK
Kernel
Cache
inode
hello
Modify
in-place
inode == inode
codesig != codesig
hello
Execute
SIGKILL
Kernel
hello
Verify
inode

Incremental linking in Zig

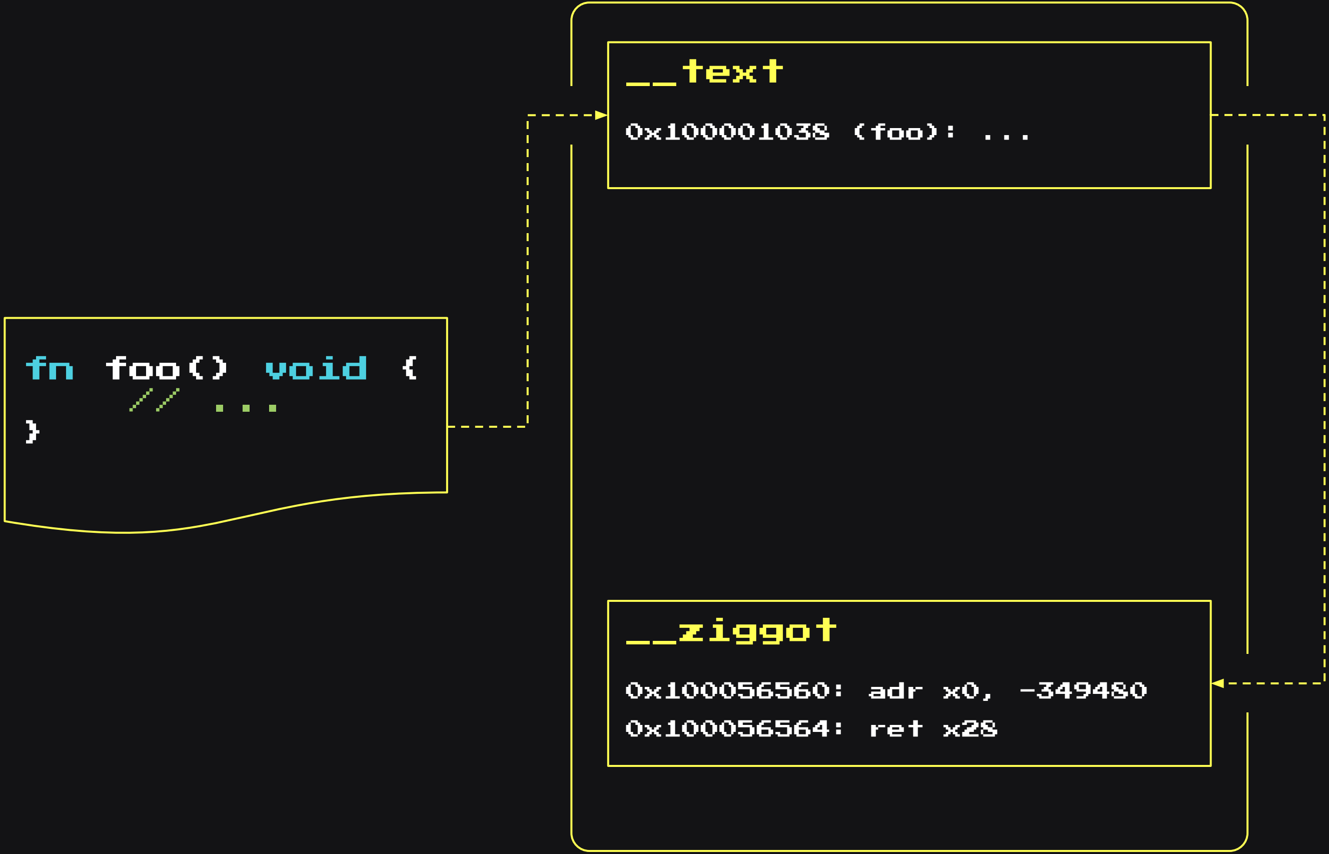
Zig's self-hosted approach



- Compilation coupled with linking
- Incremental linking means editing final exe in-place
- Shorter turnaround time



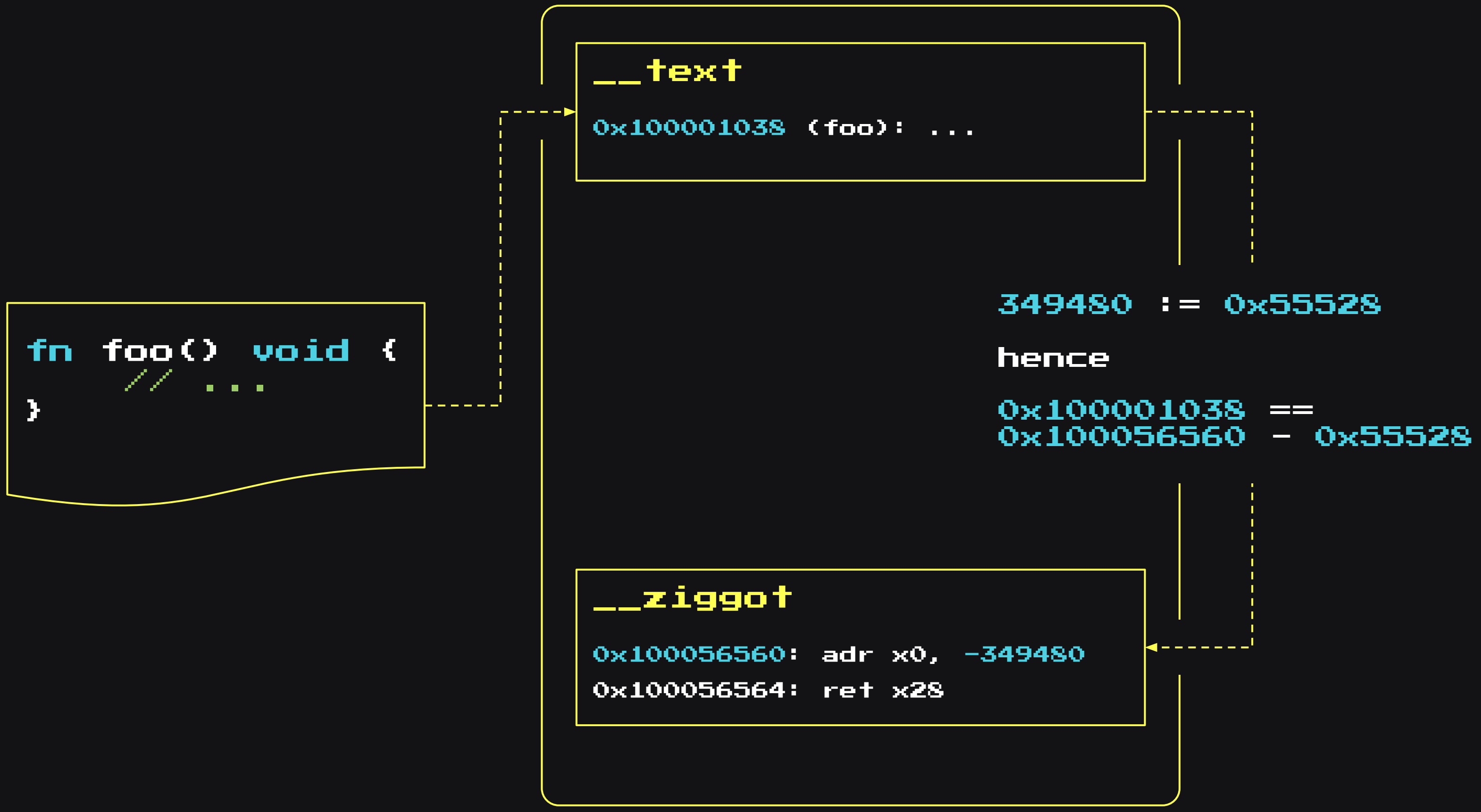
```
fn foo() void {
    // ...
}
```



```
fn foo() void {
    // ...
}
```

```
0x100001038 (foo): ...
```

```
0x100056560: adr x0, -349480
0x100056564: ret x28
```



```
fn foo() void {
    // ...
}
```

```
__text
0x100001038 (foo): ...
```

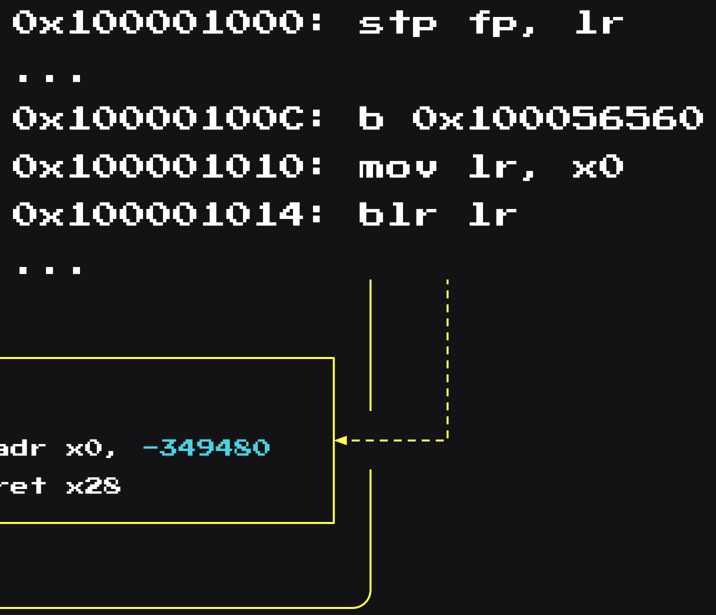
```
__ziggot
0x100056560: adr x0, -349480
0x100056564: ret x28
```

```
349480 := 0x55528
hence
0x100001038 ==
0x100056560 - 0x55528
```

```
export fn _start() void {
    foo();
}
```

```
0x100001000: stp fp, lr
...
0x10000100C: b 0x100056560
0x100001010: mov lr, x0
0x100001014: blr lr
...
```

```
export fn _start() void {
    foo();
}
```



```
0x100001000: stp fp, lr
...
0x10000100C: b 0x100056560
0x100001010: mov lr, x0
0x100001014: blr lr
...
```

```
0x100056560: adr x0, -349480
0x100056564: ret x28
```

Demo time!

BONUS: cross-compiling C to Apple Silicon with Zig

Thanks for watching!

Wanna get in touch?

GitHub, Twitter: @kubkon

Email: kubkon@jakubkonka.com