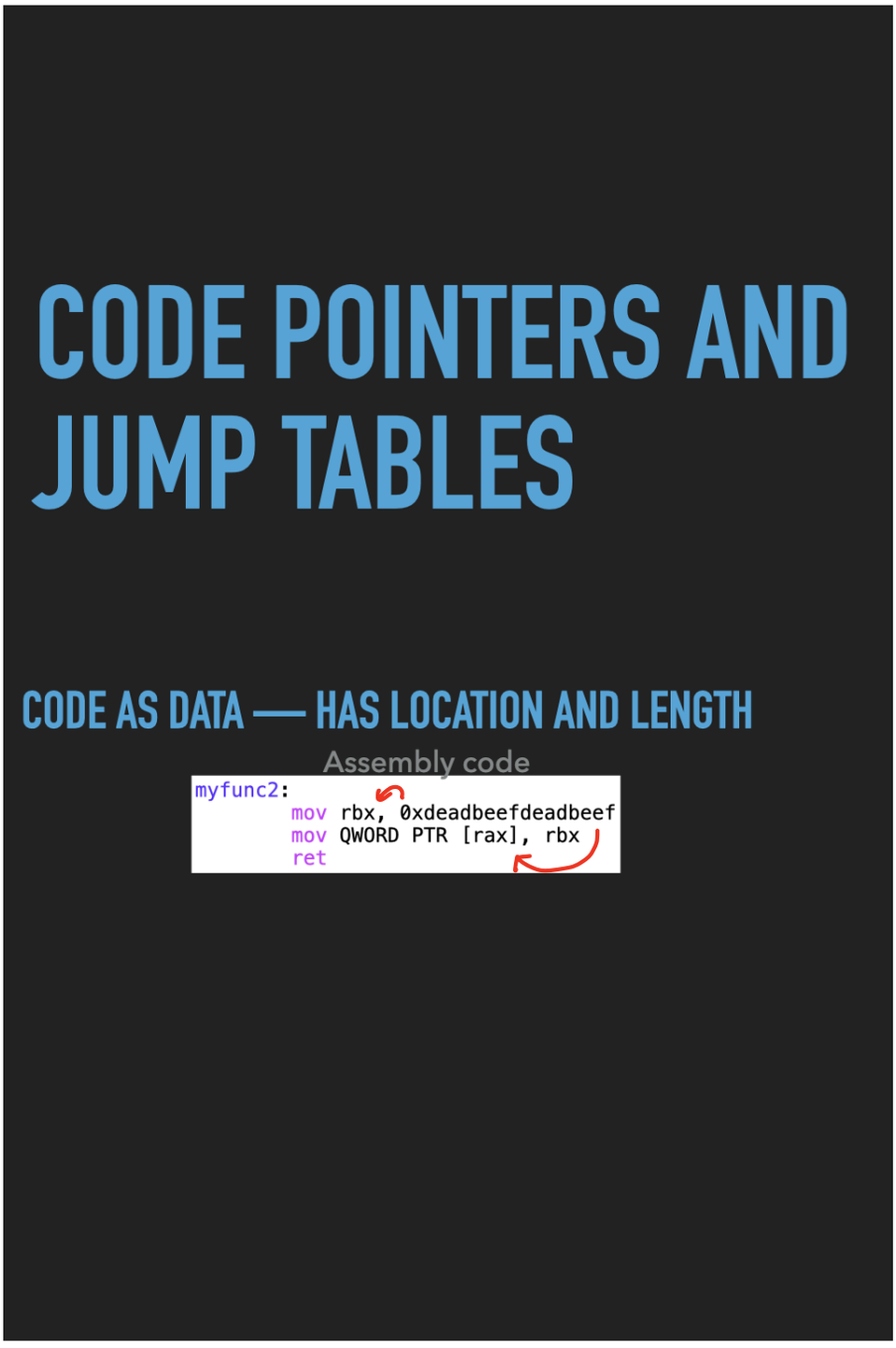
SLS Lecture 12 : Program Anatomy III : Code as Data

- create a directory `mkdir codedata; cd codedata`
- copy examples
- add a `Makefile` to automate assembling and linking
    - we are going run the commands by hand this time to highlight the details
- add our `setup.gdb` and `codeptrs.gdb` and `selfmodify.gdb` to make working in gdb easier
- normally you would want to track everything in git



CODE AS DATA — HAS LOCATION AND LENGTH

Assembly code

```
myfunc2:
        mov rbx, 0xdeadbeefdeadbeef
        mov QWORD PTR [rax], rbx
        ret
```

after assembler, linked and loading : loaded opcodes – aka machine code : 14 bytes located at 0x401024

```
(gdb) print /x  &myfunc2
$2 = 0x401024
(gdb) disass myfunc2
Dump of assembler code for function myfunc2:
   0x0000000000401024 <+0>:   movabs rbx,0xdeadbeefdeadbeef
   0x000000000040102e <+10>:  mov    QWORD PTR [rax],rbx
   0x0000000000401031 <+13>:  ret
End of assembler dump.
(gdb) x/14bx &myfunc2
0x401024 <myfunc2>:   0x48  0xbb  0xef  0xbe  0xad  0xde  0xef  0xbe
0x40102c <myfunc2+8>: 0xad  0xde  0x48  0x89  0x18  0xc3
```


CODE AS DATA — HAS LOCATION AND LENGTH

Assembly code

```
myfunc2:
        mov rbx, 0xdeadbeefdeadbeef
        mov QWORD PTR [rax], rbx
        ret
```

after assembler, linked and loading : loaded opcodes – aka machine code : 14 bytes located at 0x401024

```
(gdb) print /x  &myfunc2
$2 = 0x401024
(gdb) disass myfunc2
Dump of assembler code for function myfunc2:
   0x0000000000401024 <+0>:   movabs rbx,0xdeadbeefdeadbeef
   0x000000000040102e <+10>:  mov    QWORD PTR [rax],rbx
   0x0000000000401031 <+13>:  ret
End of assembler dump.
(gdb) x/14bx &myfunc2
0x401024 <myfunc2>:   0x48  0xbb  0xef  0xbe  0xad  0xde  0xef  0xbe
0x40102c <myfunc2+8>: 0xad  0xde  0x48  0x89  0x18  0xc3
```


MYFUNC2 @ 0X401024, LEN=14

CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

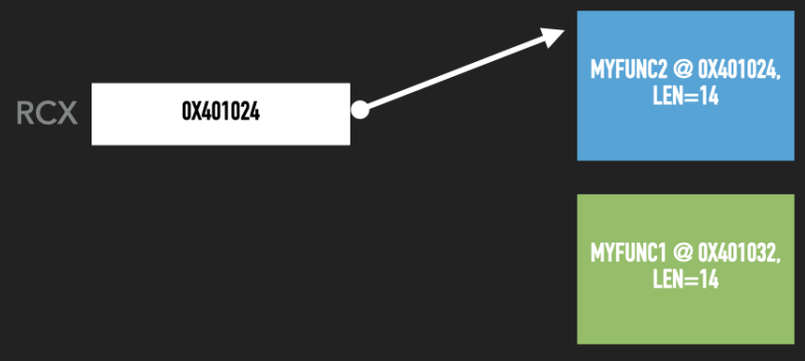
MYFUNC2 @ 0X401024, LEN=14

MYFUNC1 @ 0X401032, LEN=14

CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

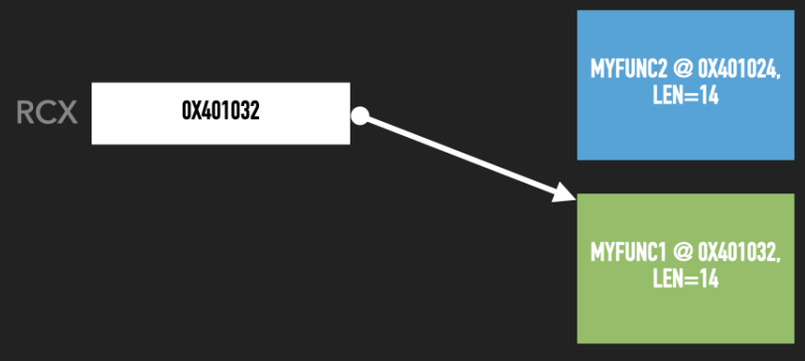
CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

Code "Pointers": We can think of RCX pointing to MYFUNC2



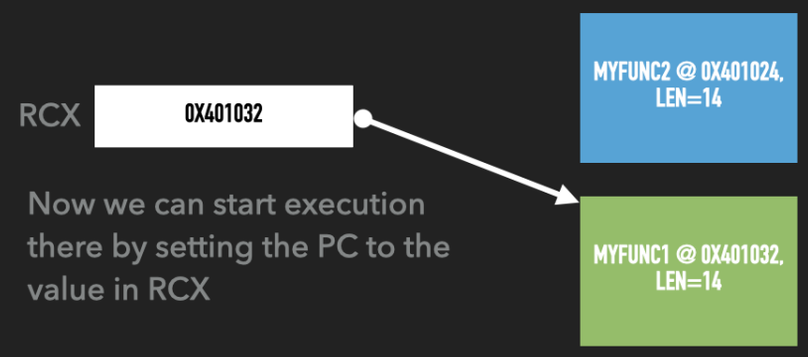
CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

and now to MYFUNC1



CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

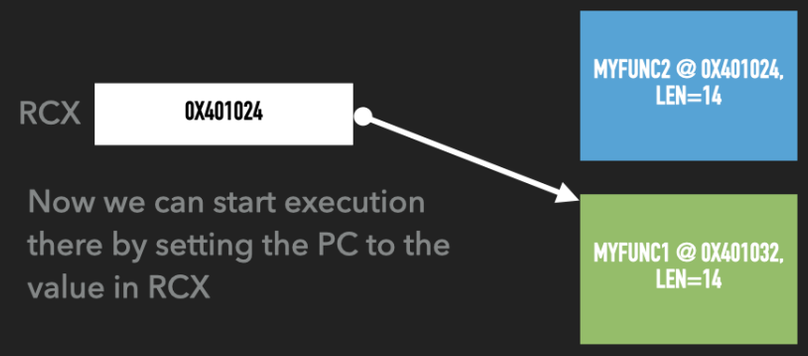
and now to MYFUNC1



Now we can start execution there by setting the PC to the value in RCX

CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

and now to MYFUNC1



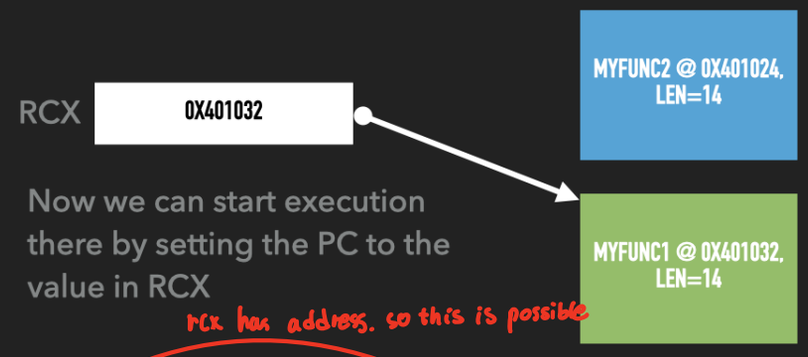
Now we can start execution there by setting the PC to the value in RCX

```
jmp rcx
```

jmp if it is just a simple "BASIC BLOCK"

CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

and now to MYFUNC1



Now we can start execution there by setting the PC to the value in RCX

rcx has address, so this is possible

```
jmp rcx
```

jmp if it is just a simple "BASIC BLOCK"

```
call rcx
```

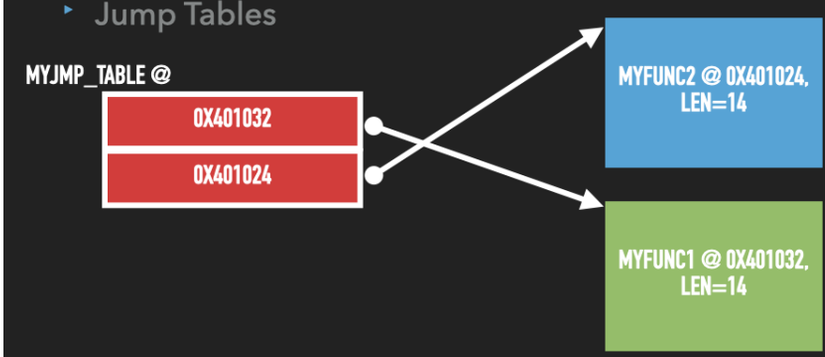
call if it is a function "eg has a ret at the end"

CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

- We can pass "pointers" to code around … eg tell one function to invoke another function at runtime – not hardcode
- What makes a value a code pointer?

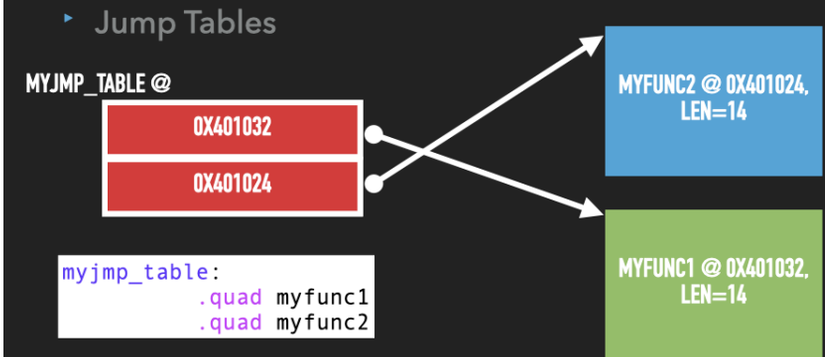
CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

- Arrays of pointers to code
  - Jump Tables



CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

- Arrays of pointers to code
  - Jump Tables

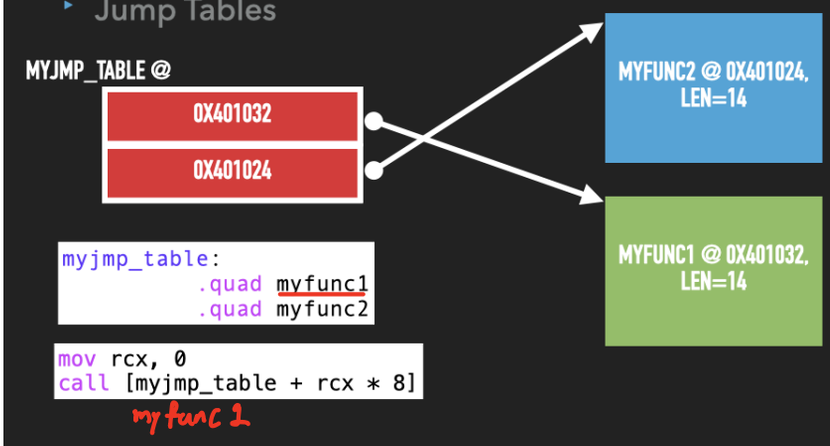



```
myjmp_table:
        .quad myfunc1
        .quad myfunc2
```

- myjmp_table[0] points to func1
- myjmp_table[1] points to func2
- an index can be used to say where to continue execution

CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

- Arrays of pointers to code
  - Jump Tables



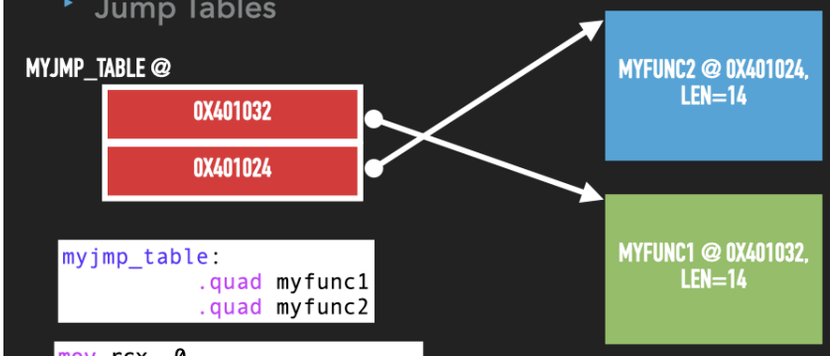

```
myjmp_table:
        .quad myfunc1
        .quad myfunc2
```

```
mov rcx, 0
call [myjmp_table + rcx * 8]
```

myfunc 2

CODE AS DATA — WHAT CAN WE DO WITH THIS FACT?

- Arrays of pointers to code
  - Jump Tables




```
myjmp_table:
        .quad myfunc1
        .quad myfunc2
```

```
mov rcx, 0
call [myjmp_table + rcx * 8]
```

Any thing we can do with arrays – index , iterate, update, etc.

WHEN YOU KNOW CODE IS JUST BYTES

SUPER ADVANCED: SELF MODIFYING CODE!!!!

- Since code is also just an array of bytes located at an address in memory.

SUPER ADVANCED: SELF MODIFYING CODE!!!!

- Since code is also just an array of bytes located at an address in memory.
  - We can generate code on the fly to execute – writes bytes that are valid opcodes to a location in memory and jump there

```
mov r8, OFFSET array
xor rdi, rdi        rdi: 0
mov al, 0xf3        al: f3
mov BYTE PTR [r8 + rdi], al
inc rdi             1
mov al, 0x48
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0x0f
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0xb8
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0xd8
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0xcc
mov BYTE PTR [r8 + rdi], al
jmp array
```

SUPER ADVANCED: SELF MODIFYING CODE!!!!

- Since code is also just an array of bytes located at an address in memory.
  - We can generate code on the fly to execute – writes bytes that are valid opcodes to a location in memory and jump there

```
mov r8, OFFSET array
xor rdi, rdi
mov al, 0xf3
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0x48
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0x0f
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0xb8
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0xd8
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0xcc
mov BYTE PTR [r8 + rdi], al
jmp array
```

```
(gdb) x/2i &array
   0x4000b7 <array>: popcnt rbx,rax
=> 0x4000bc <array+5>: int3
```

SUPER ADVANCED: SELF MODIFYING CODE!!!!

- Since code is also just an array of bytes located at an address in memory.
  - We can generate code on the fly to execute – writes bytes that are valid opcodes to a location in memory and jump there
  - We can modify the bytes of existing opcodes.

```
mov r8, OFFSET array
xor rdi, rdi
mov al, 0xf3
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0x48
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0x0f
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0xb8
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0xd8
mov BYTE PTR [r8 + rdi], al
inc rdi
mov al, 0xcc
mov BYTE PTR [r8 + rdi], al
jmp array
```

```
mov cl, 0xaf,
mov BYTE PTR [func2 + 4], cl
```

typo

MYFUNC2 @ 0X1000, LEN=14

By Jonathan Appavoo