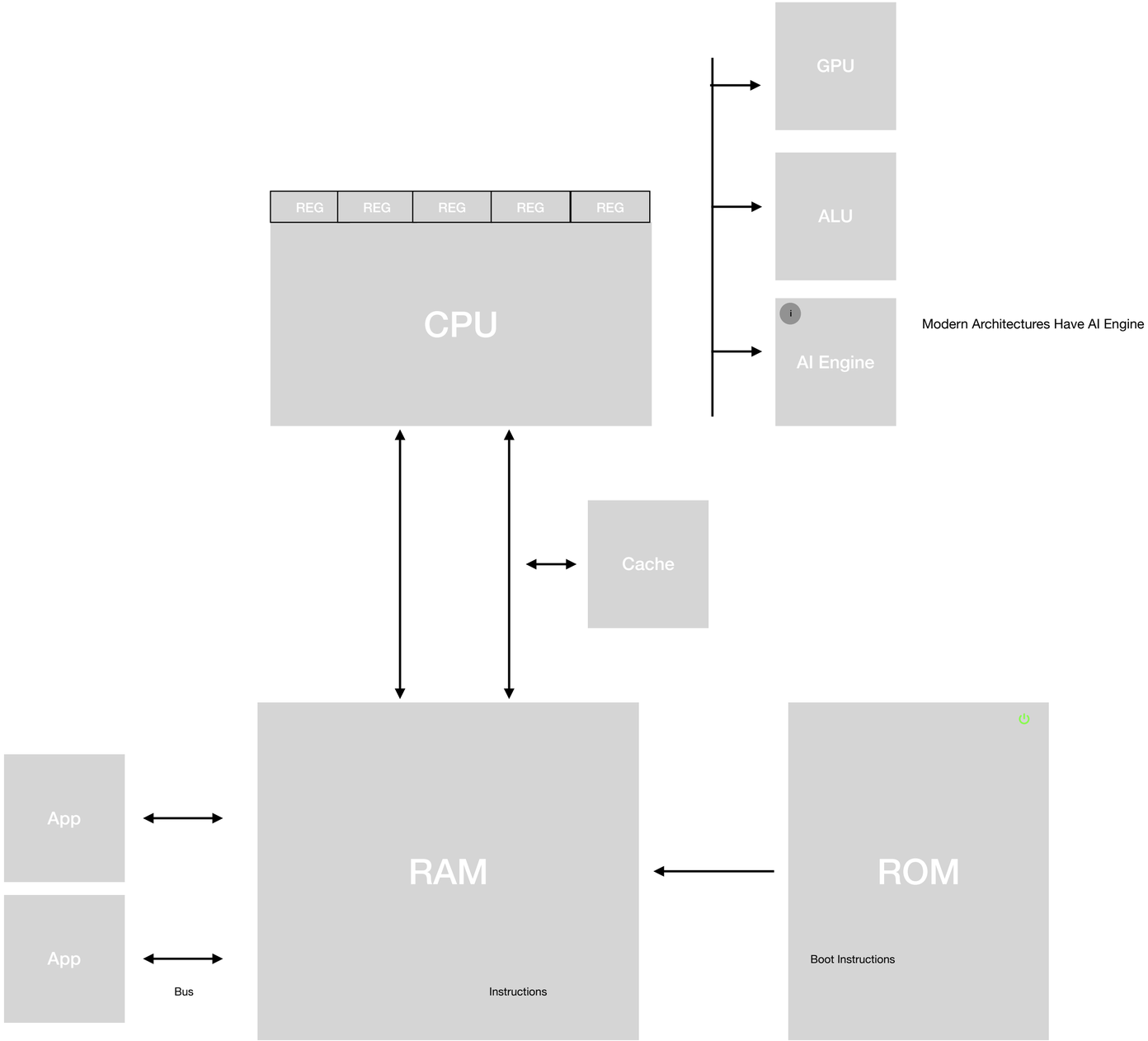
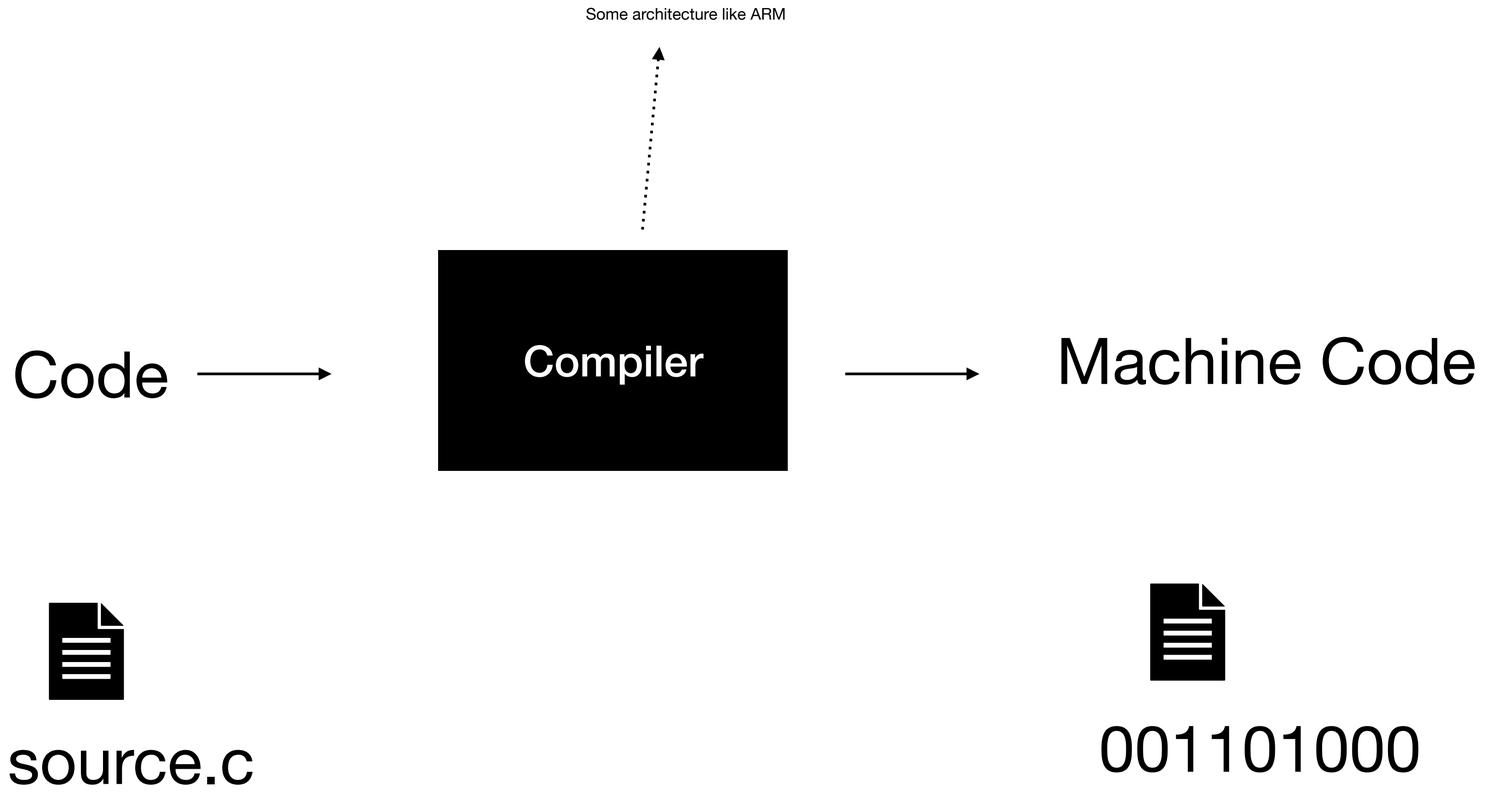
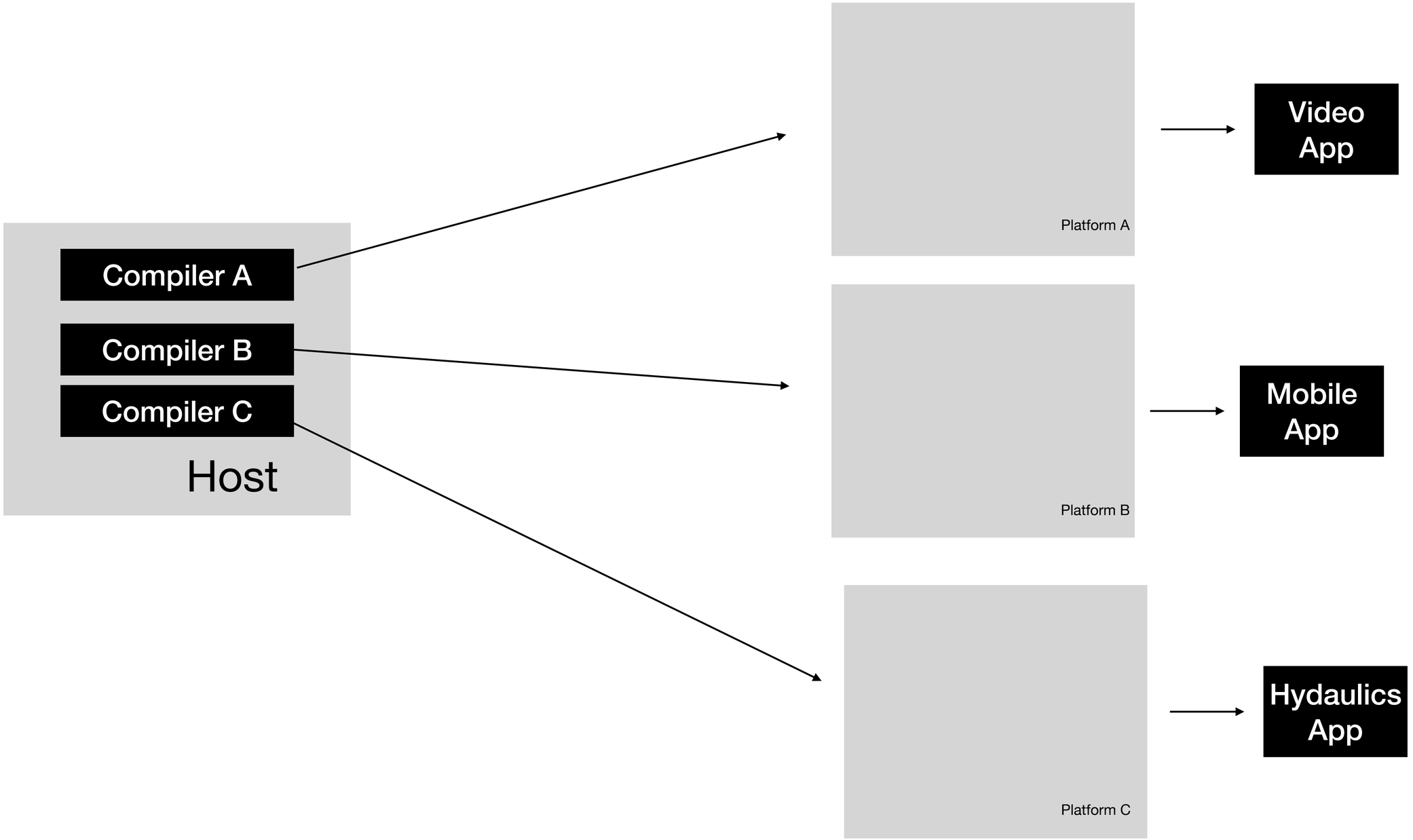
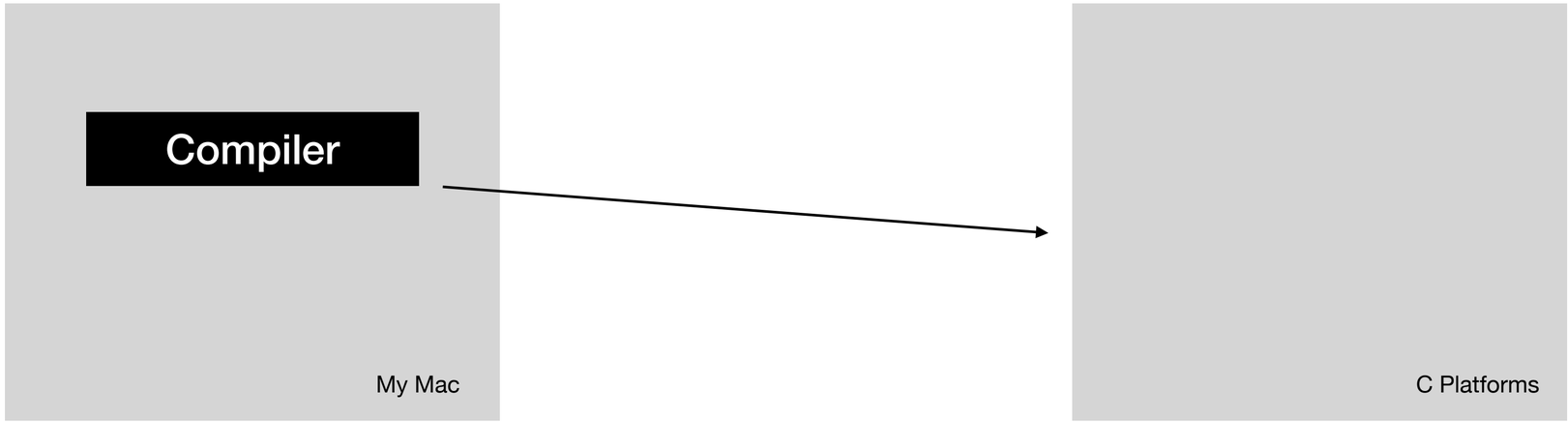
C

Basic Architecture


GPU
ALU
AI Engine
Modern Architectures Have AI Engine
REG
REG
REG
REG
REG
CPU
Cache
App
App
Bus
RAM
Instructions
ROM
Boot Instructions


Some architecture like ARM
Code
Compiler
Machine Code
source.c
001101000

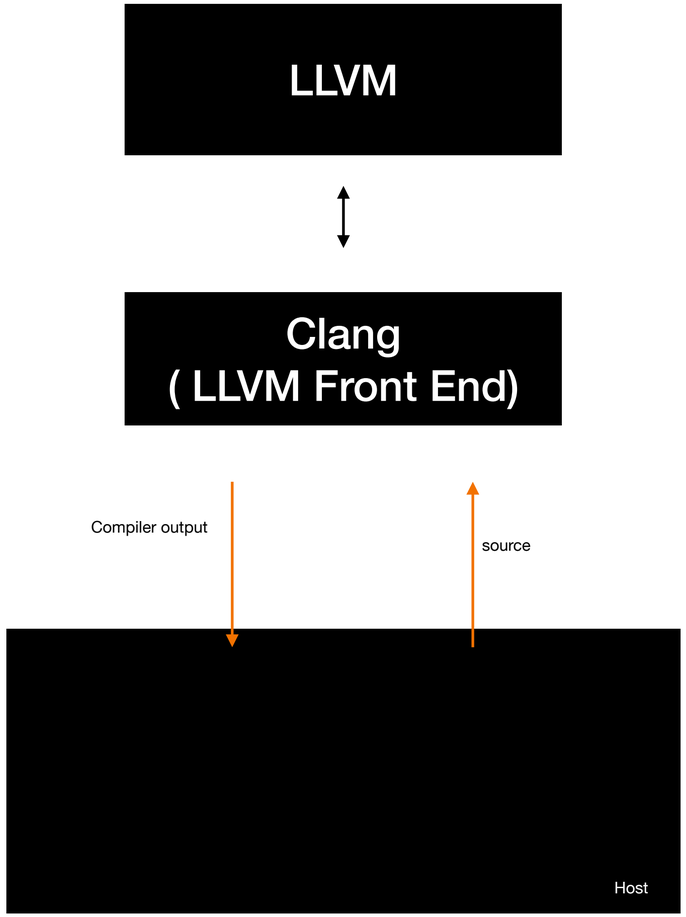

Compiler A
Compiler B
Compiler C
Host
Platform A
Platform B
Platform C
Video App
Mobile App
Hydaulics App

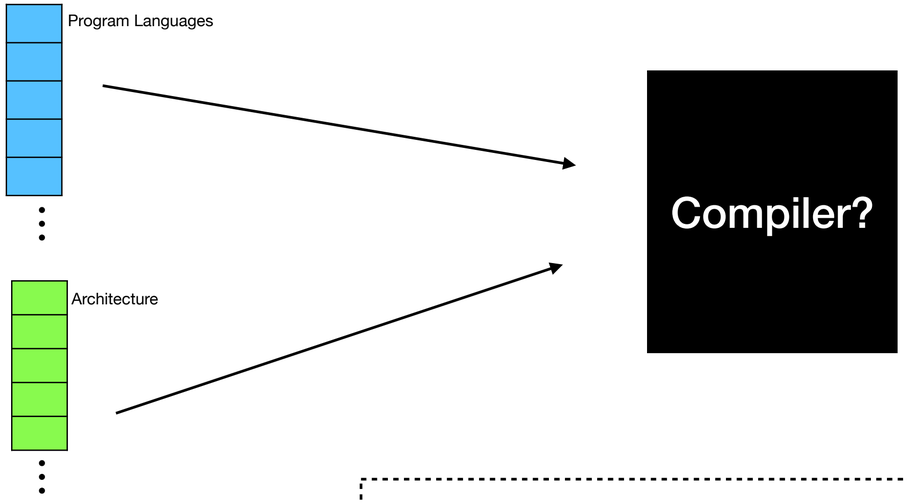


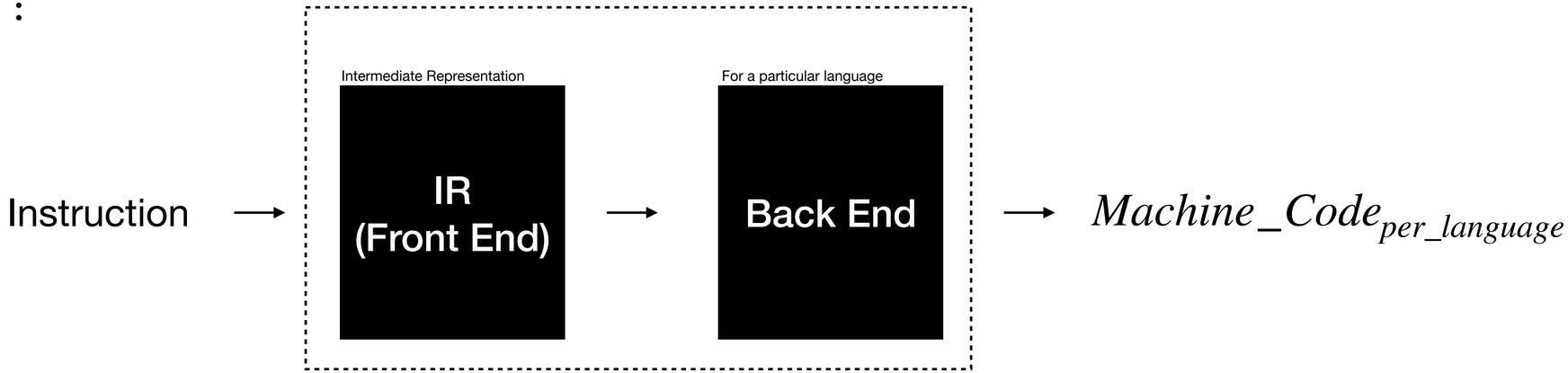
Compiles to machine code format for any platform that executes C

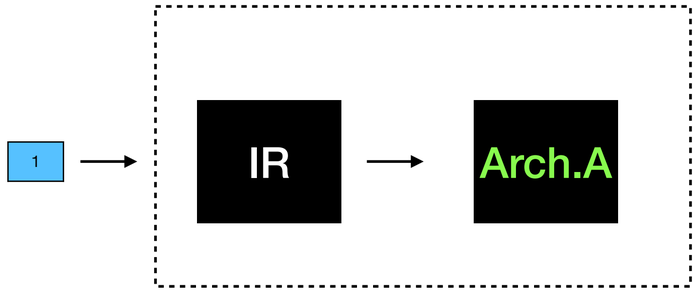
Compiler

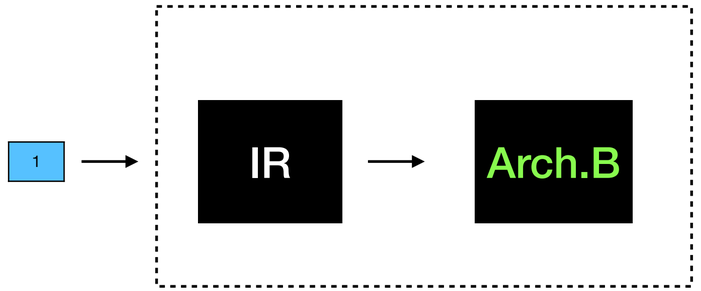
LLVM collection of modular and reusable compiler and toolchain technologies

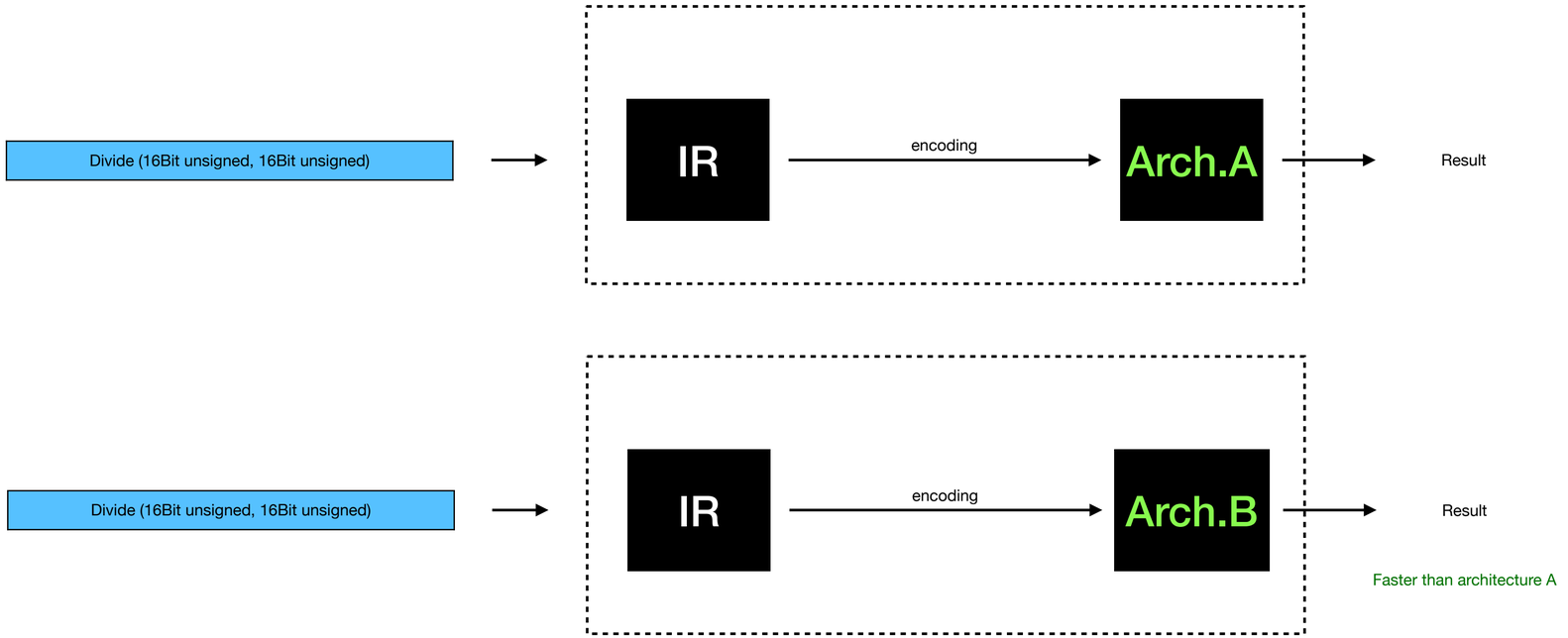


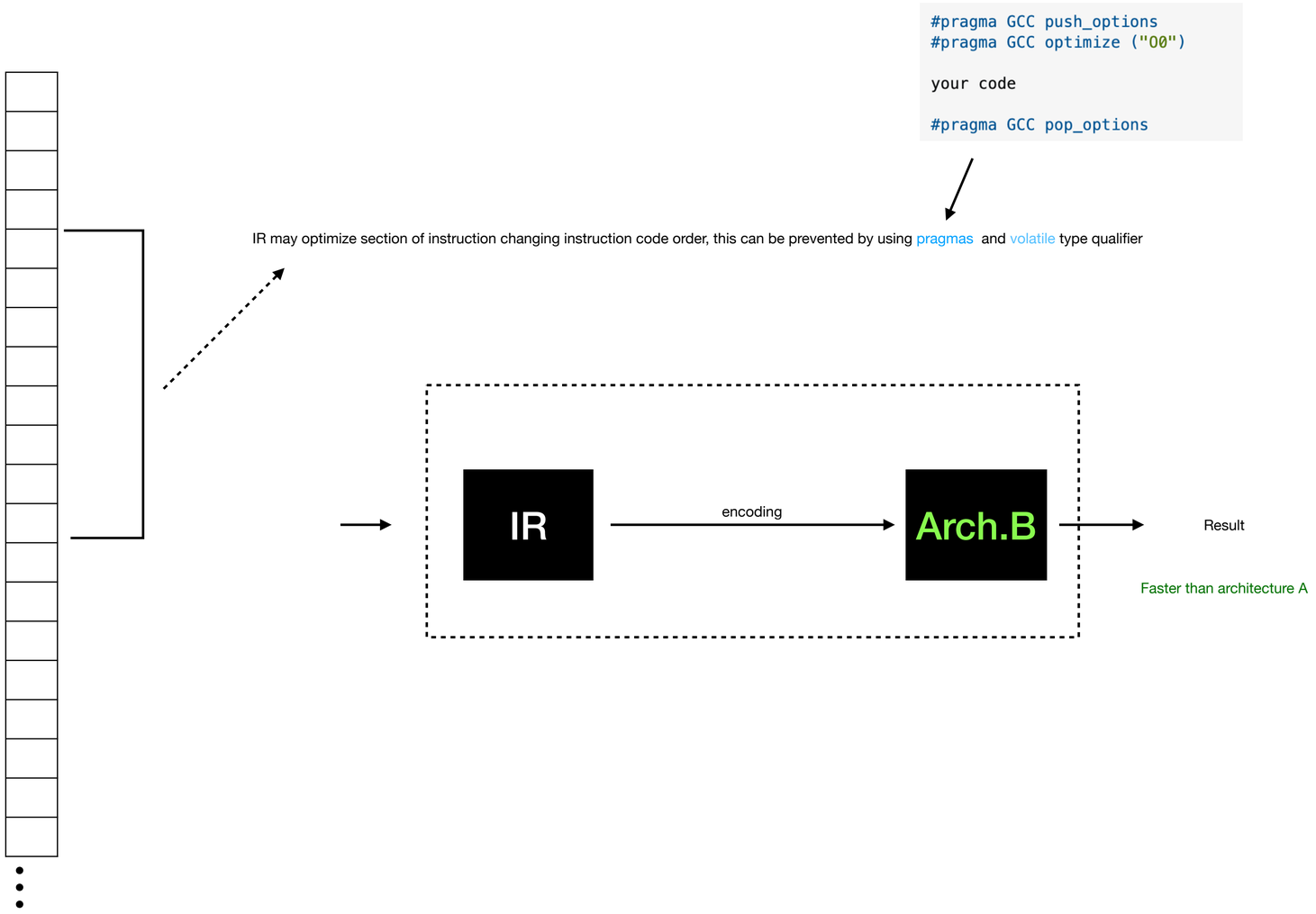

Program Languages
Architecture
Compiler?


Intermediate Representation
For a particular language
Instruction
IR
(Front End)
Back End
$Machine\_Code_{per\_language}$

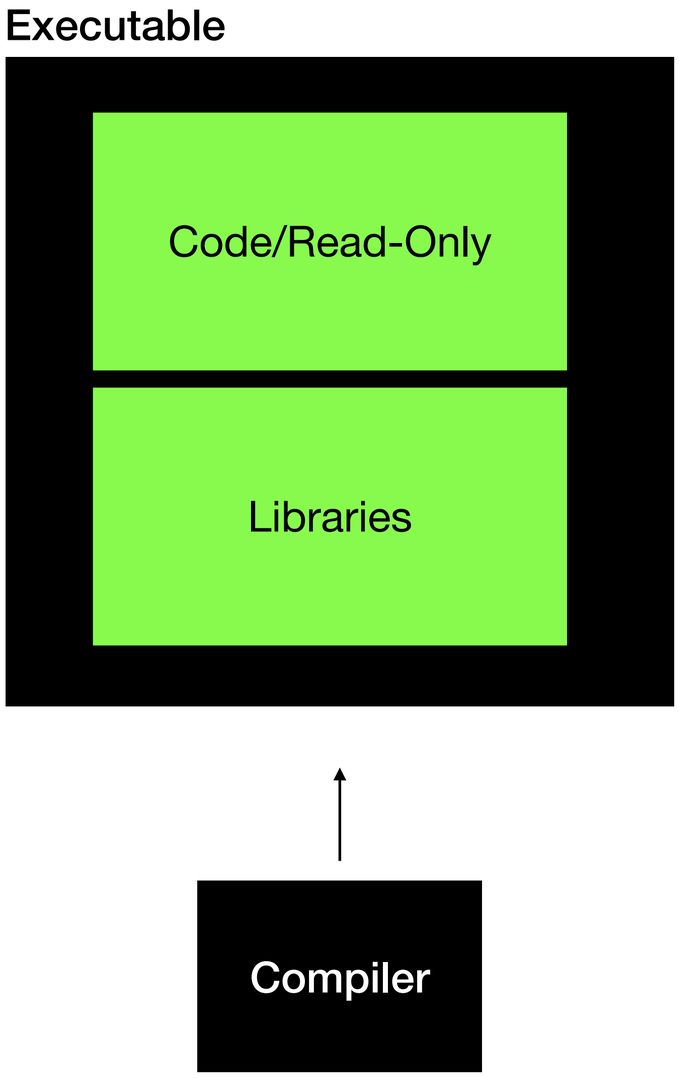

1
IR
Arch.A

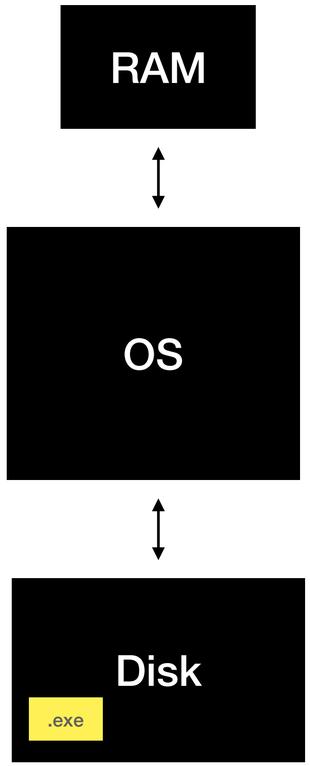

1
IR
Arch.B


Divide (16Bit unsigned, 16Bit unsigned)
IR
encoding
Arch.A
Result
Divide (16Bit unsigned, 16Bit unsigned)
IR
encoding
Arch.B
Result
Faster than architecture A


#pragma GCC push_options
#pragma GCC optimize ("O0")
your code
#pragma GCC pop_options
IR may optimize section of instruction changing instruction code order, this can be prevented by using pragmas and volatile type qualifier
IR
encoding
Arch.B
Result
Faster than architecture A

Static Linking



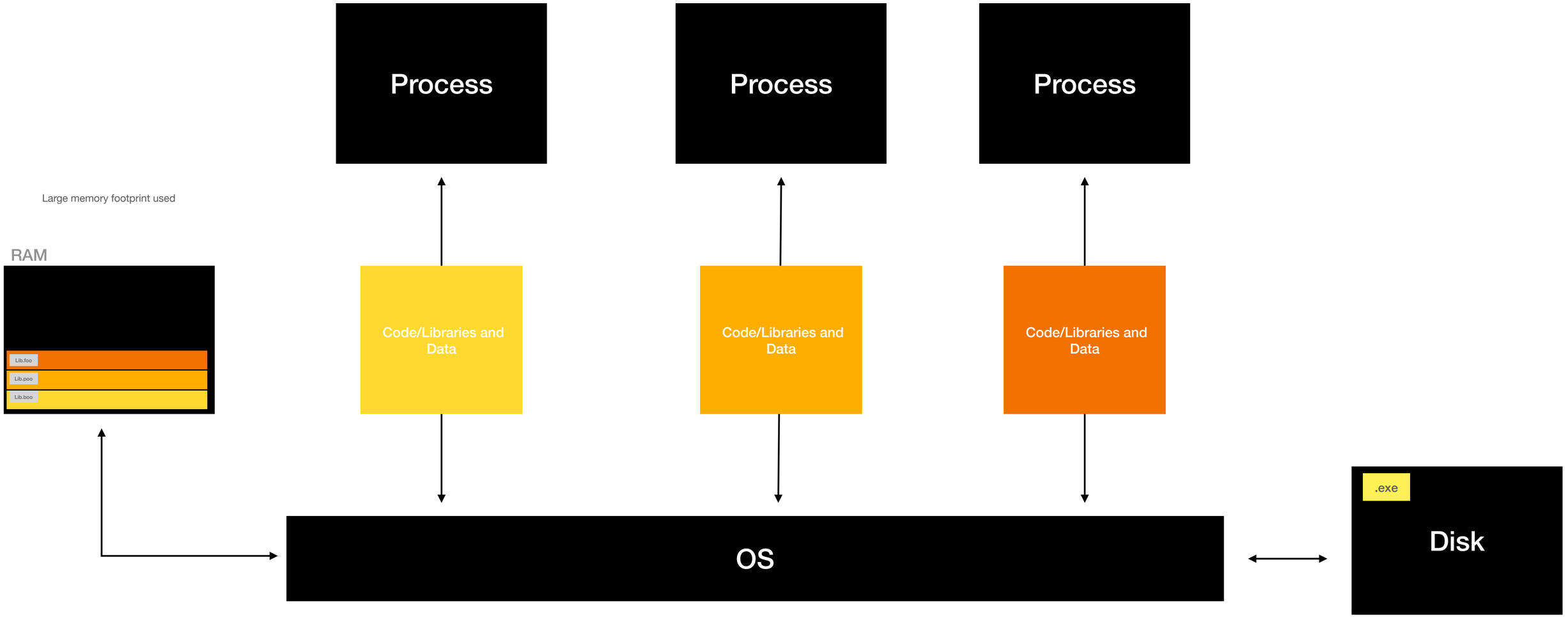


Faster execution, no symbols to resolve during context change or processes change.

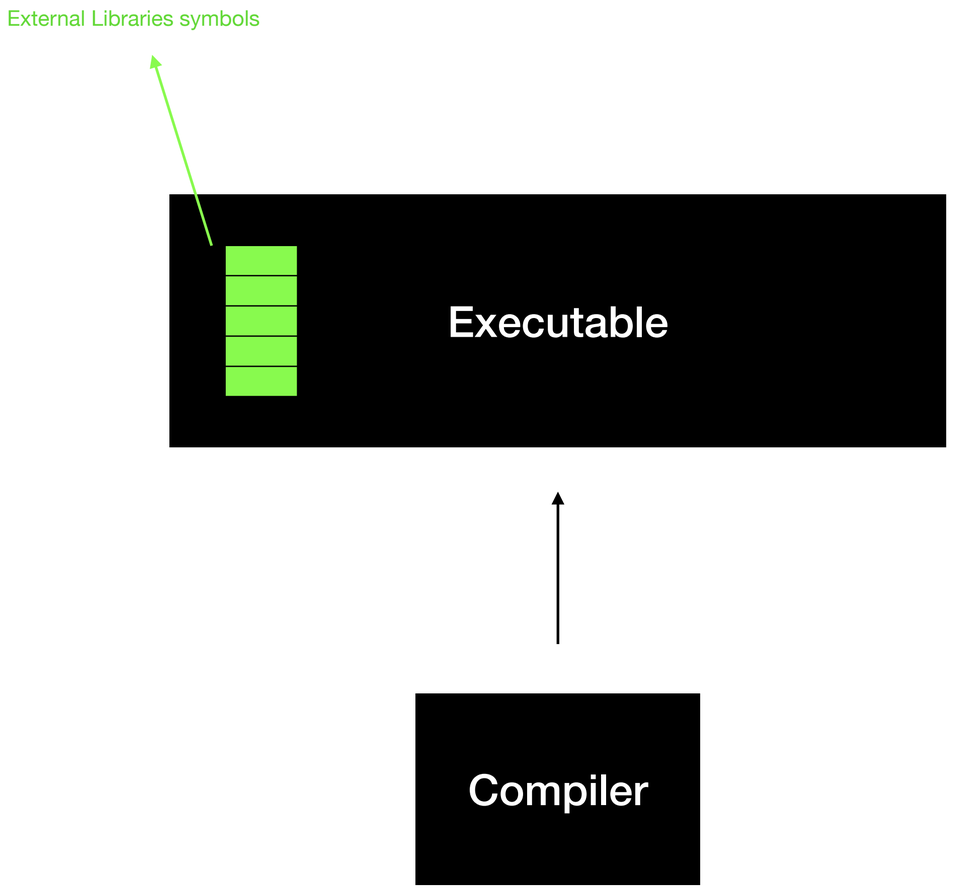
Larger file size

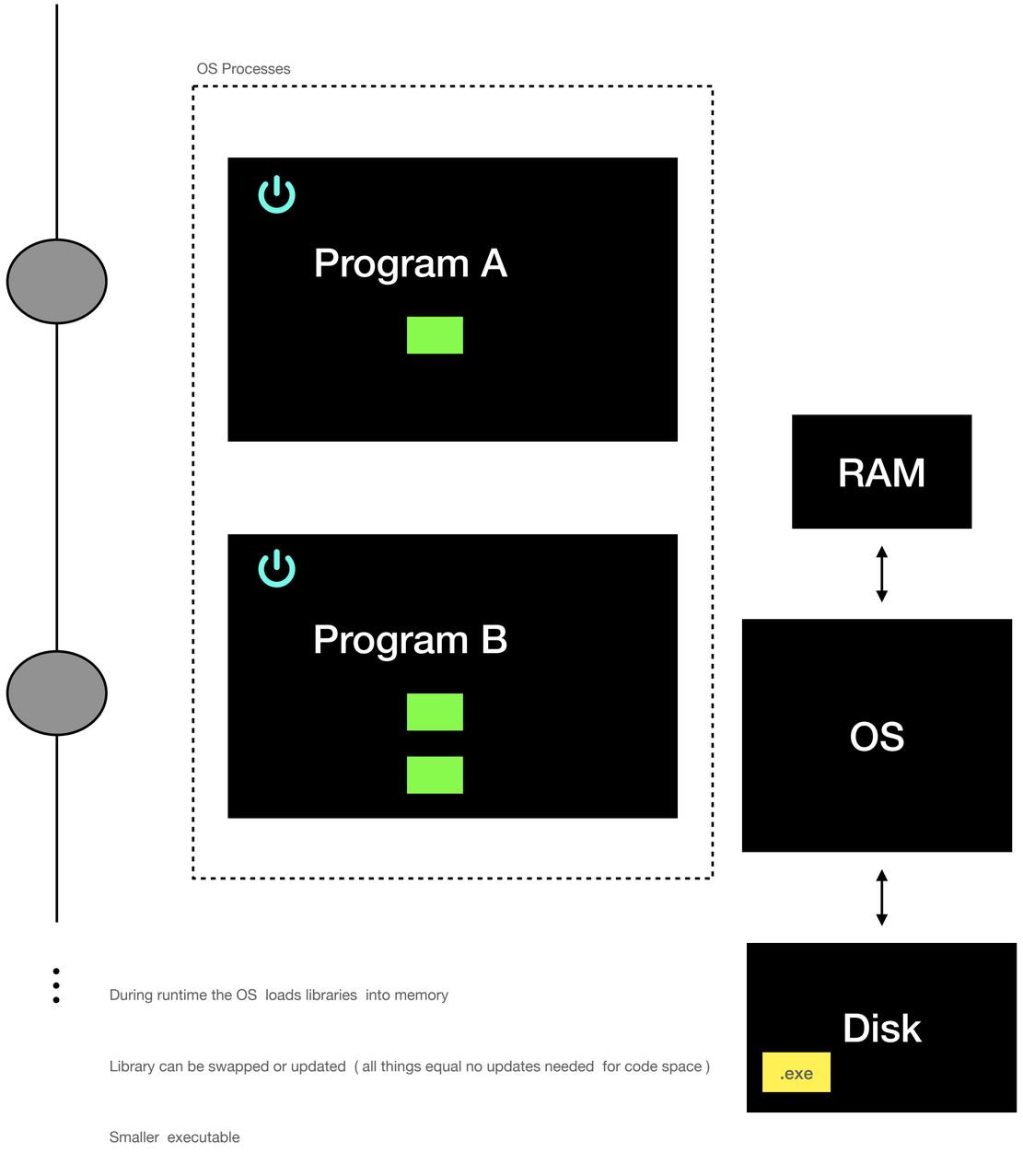
Change in libraries require recompilation (not good !)

Static Linking

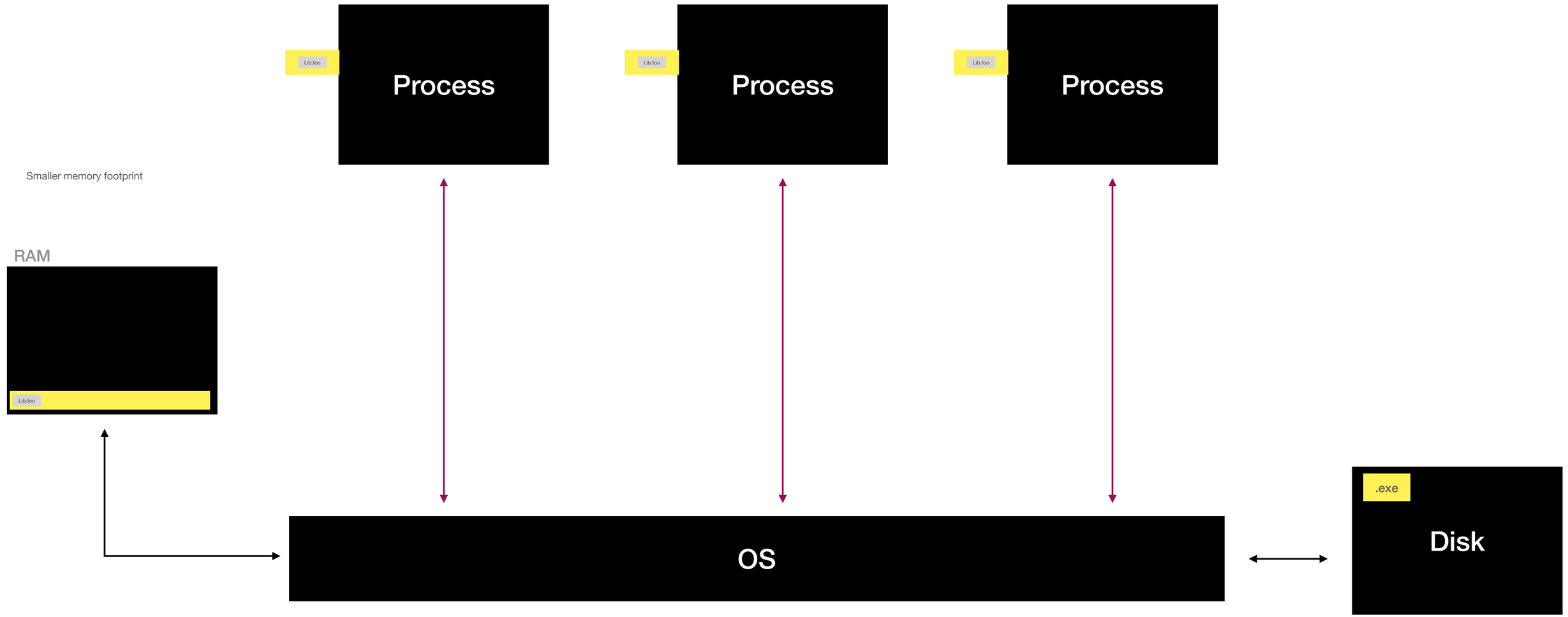


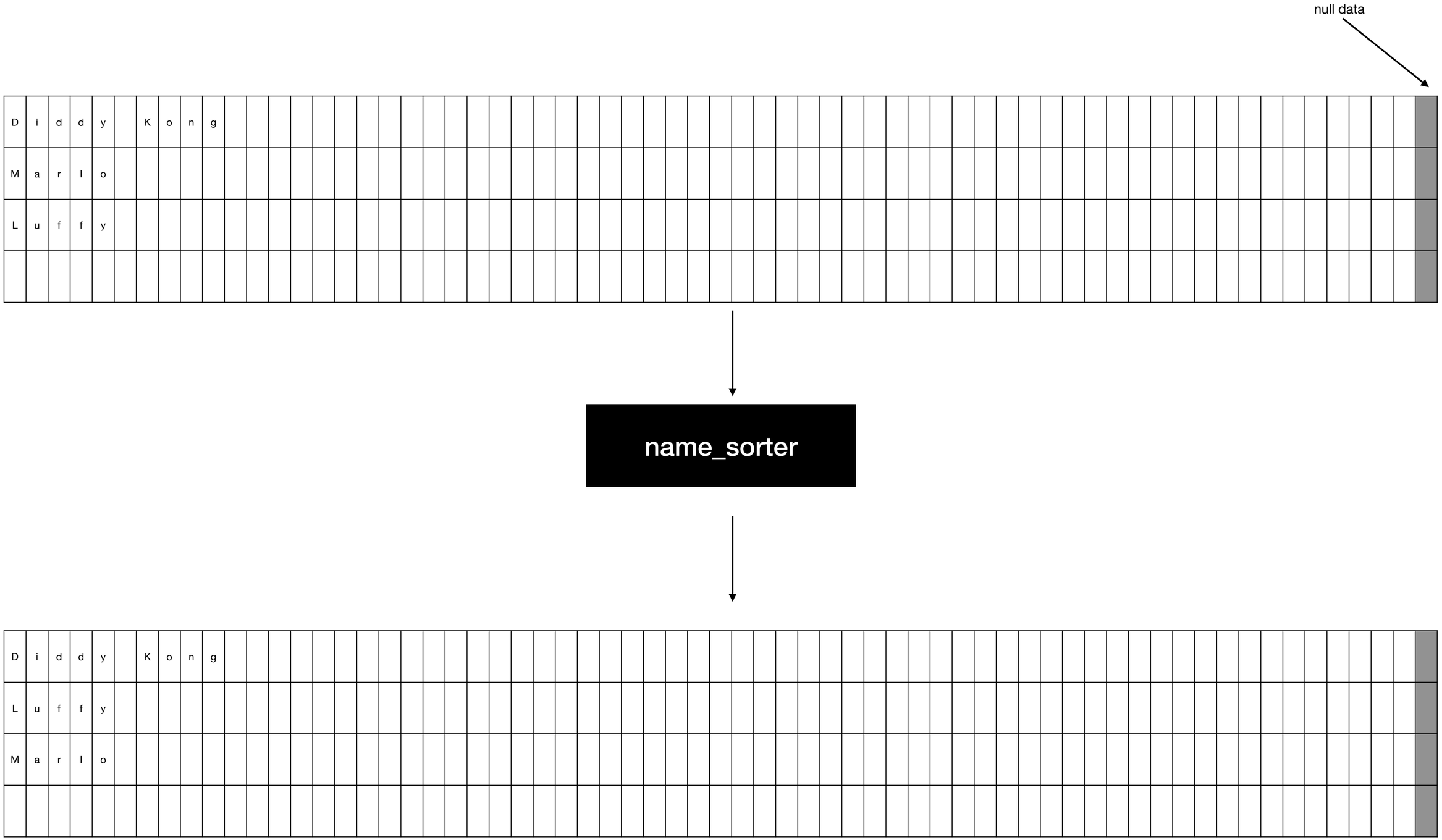
Dynamic Linking



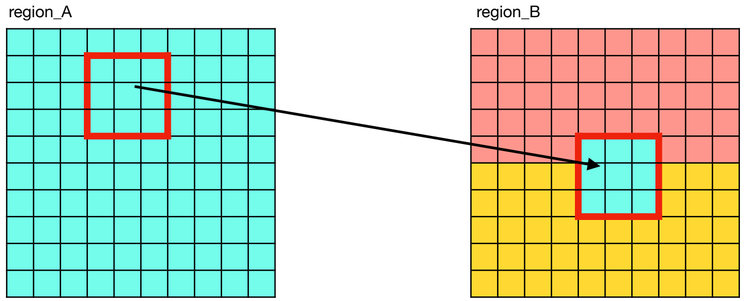


Static Linking

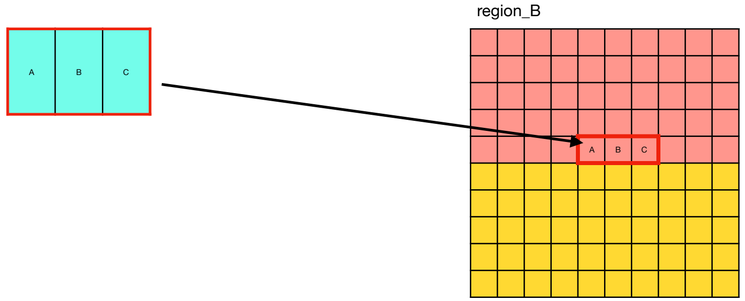



null data
Diddy Kong
Marlo
Luffy
name_sorter
Diddy Kong
Luffy
Marlo

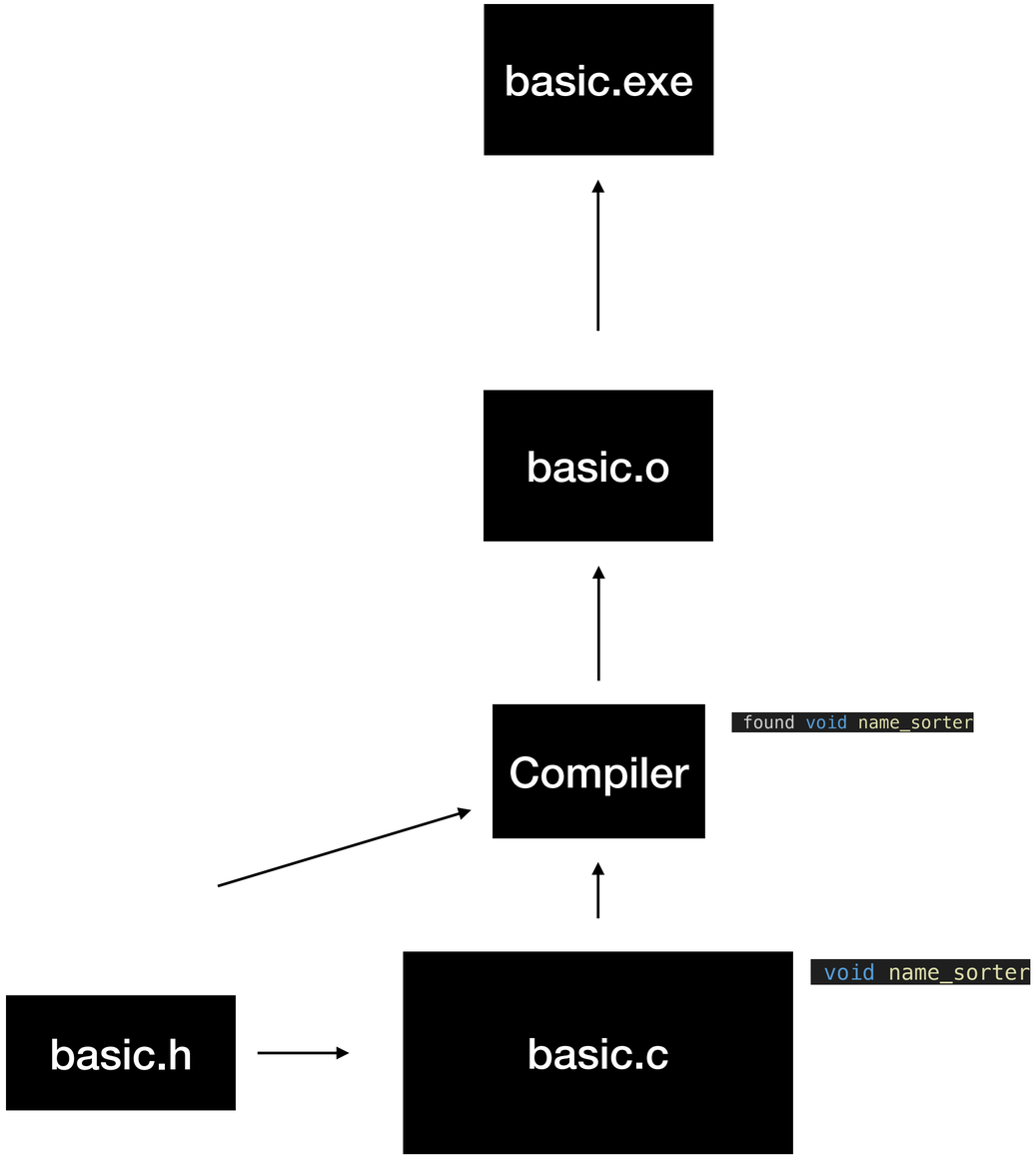
MEMCPY

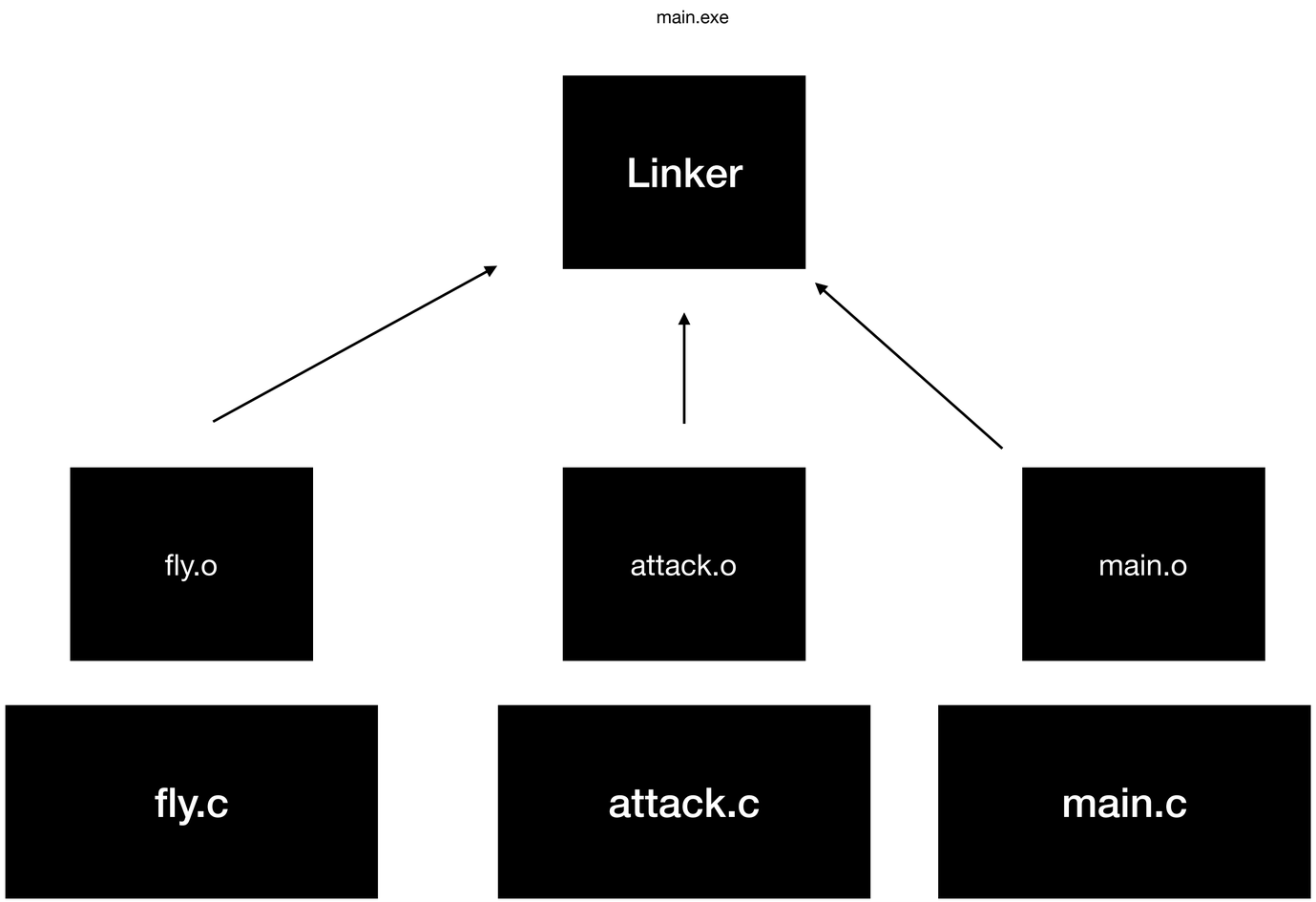


STRINGCPY/BYTE COPY



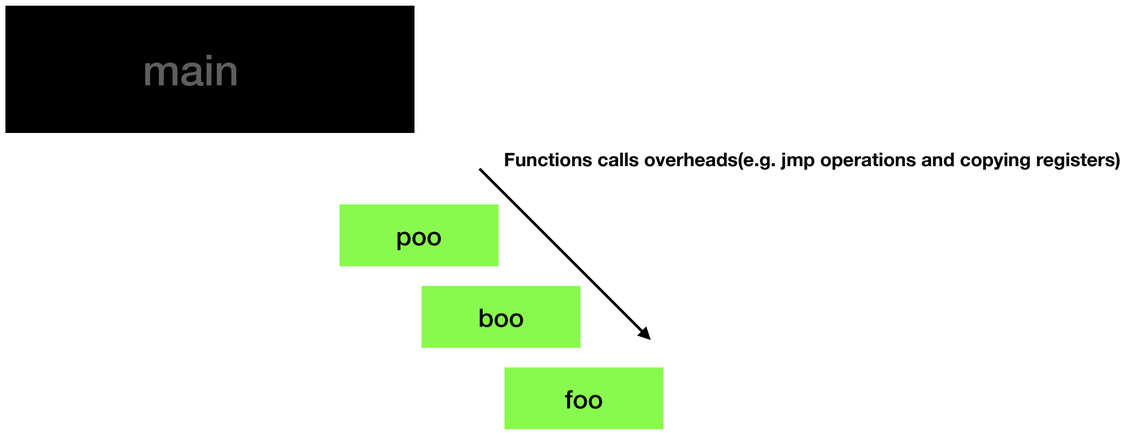
Code Gen


basic.exe
basic.o
Compiler
found void name_sorter
basic.h
basic.c
void name_sorter


main.exe
Linker
fly.o
attack.o
main.o
fly.c
attack.c
main.c

Inline Keyword

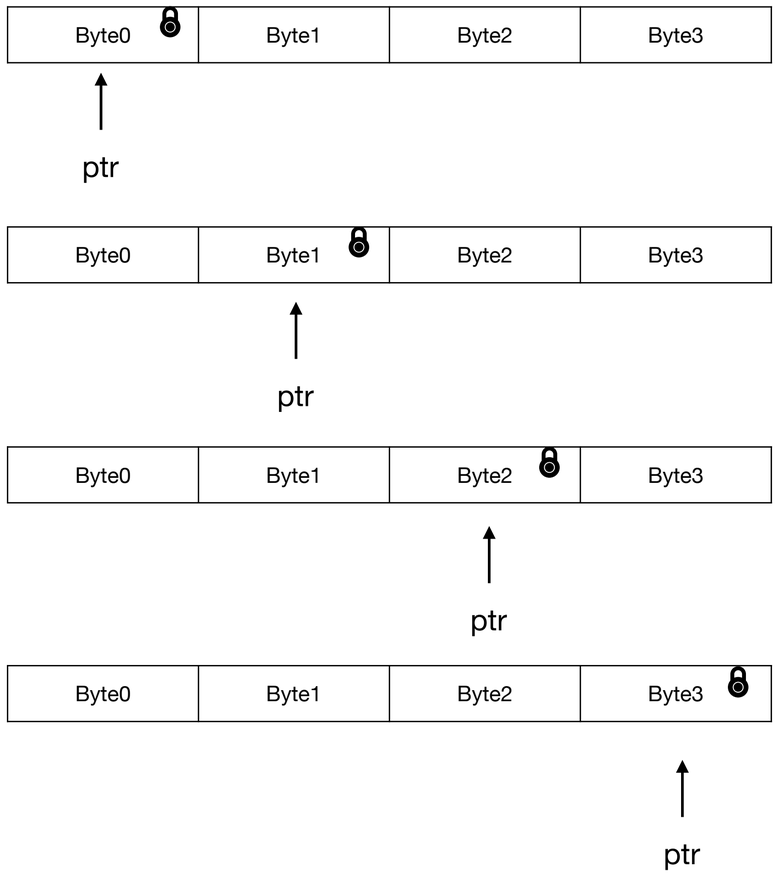
Compiler inserts code directly at call site to reduce call overhead



Stack does not contain function call overhead.
Compiler code added directly, which will increase code region

const char * vs char const * vs const char *const ptr

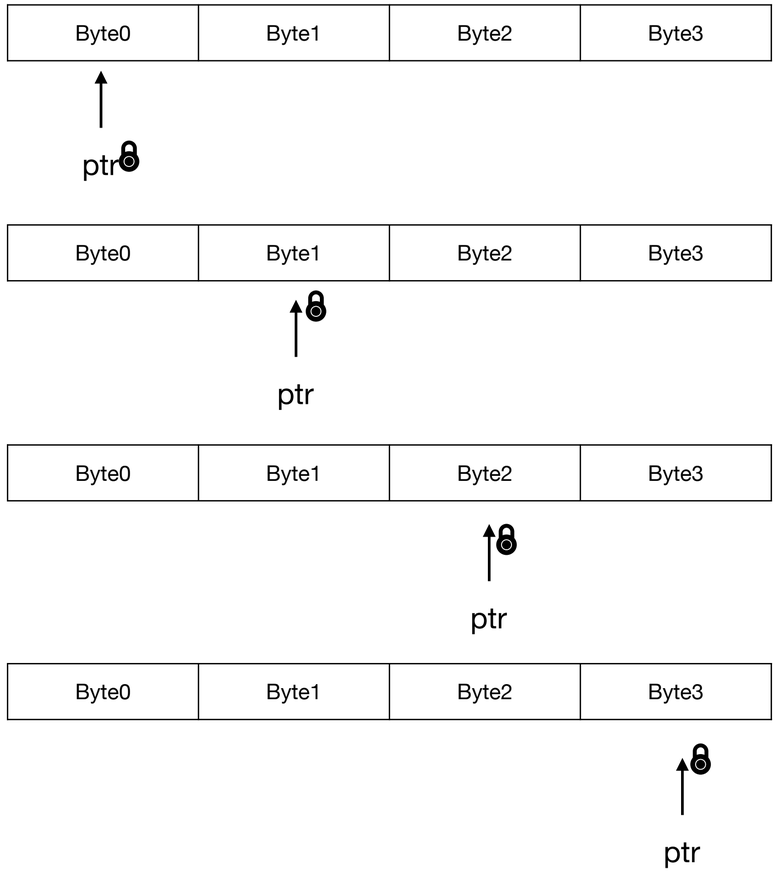
const char *ptr



Pointer can be modified, but data is constant

const char * vs char const * vs const char *const ptr

char const *ptr

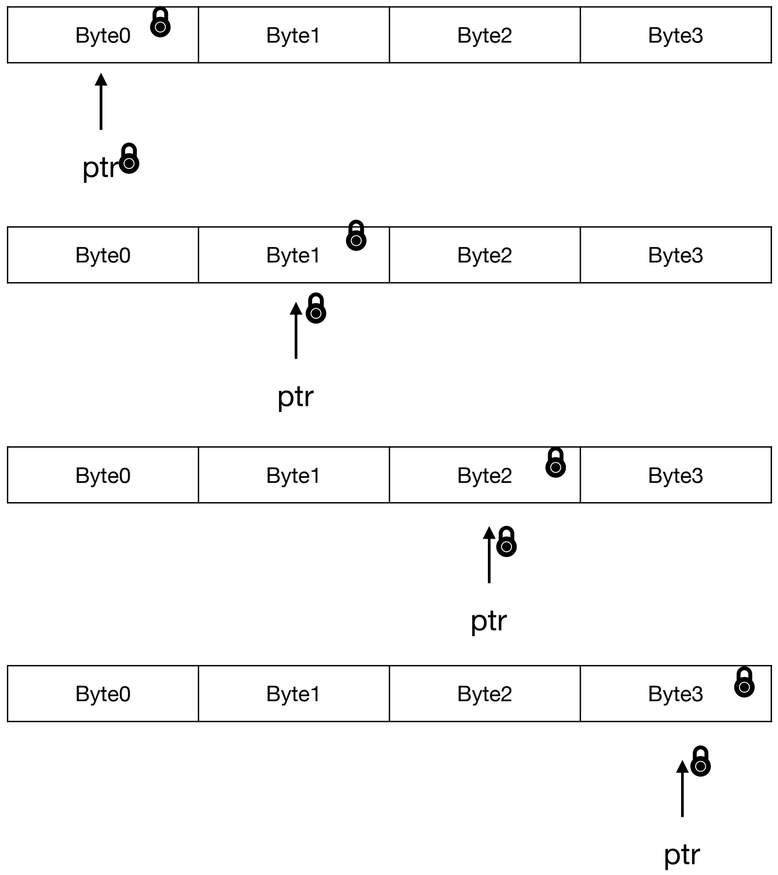


Pointer is constant, but data can be modified

const char * vs char const * vs const char *const ptr

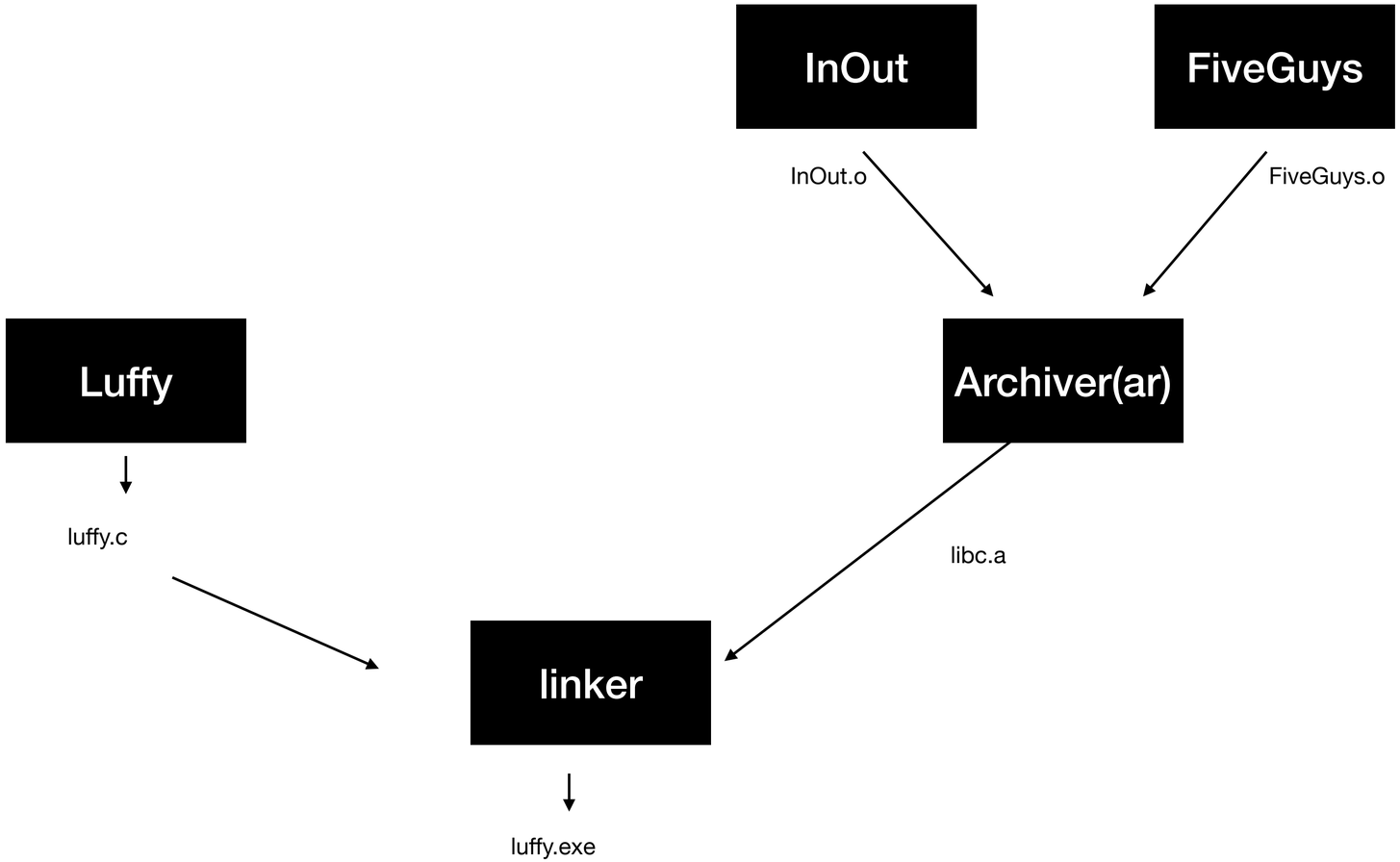
const char const *ptr

Pointer and data is constant

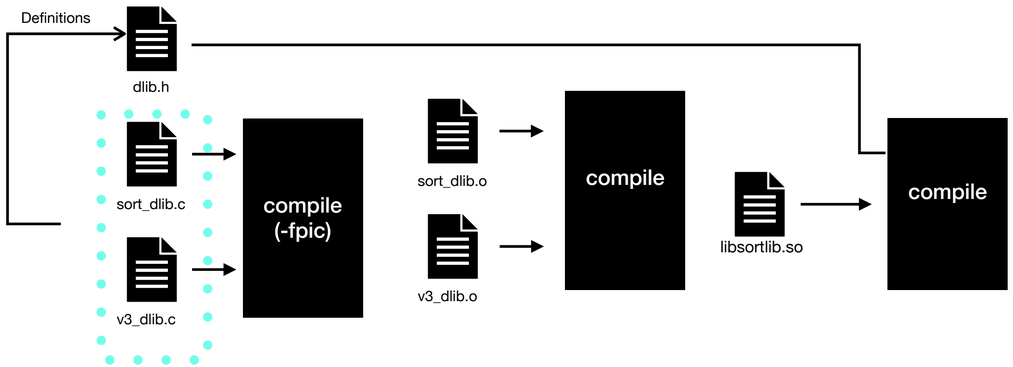


Static libraries

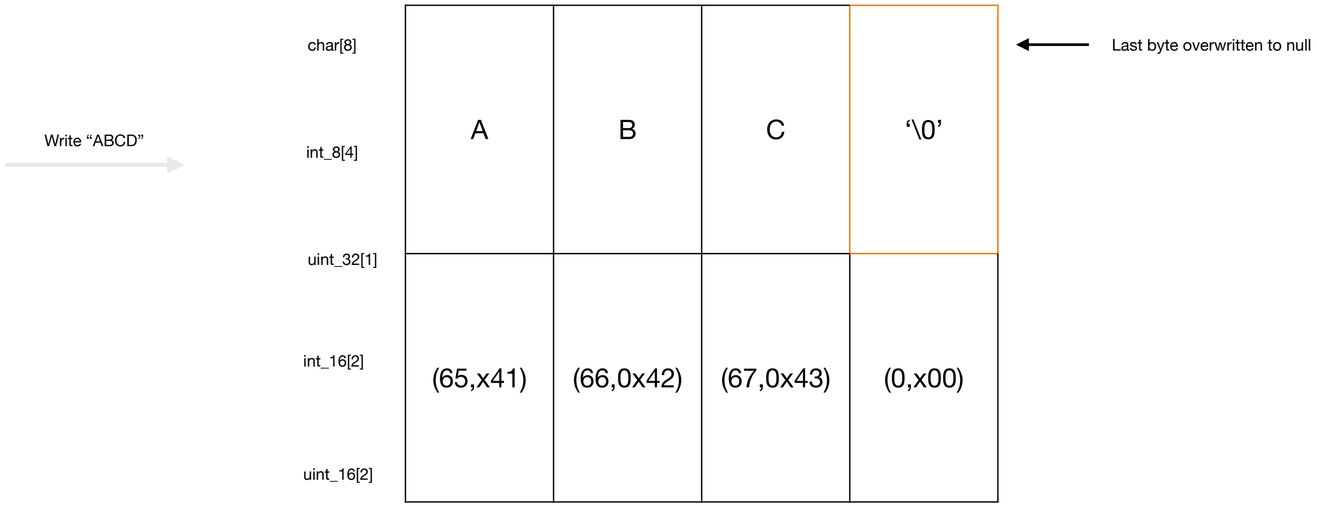
Generated by archive tool



Dynamic library

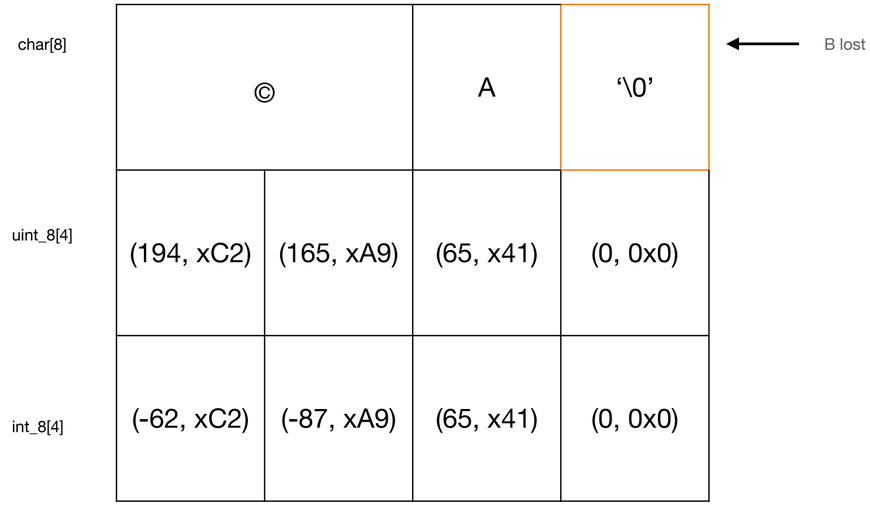


Union

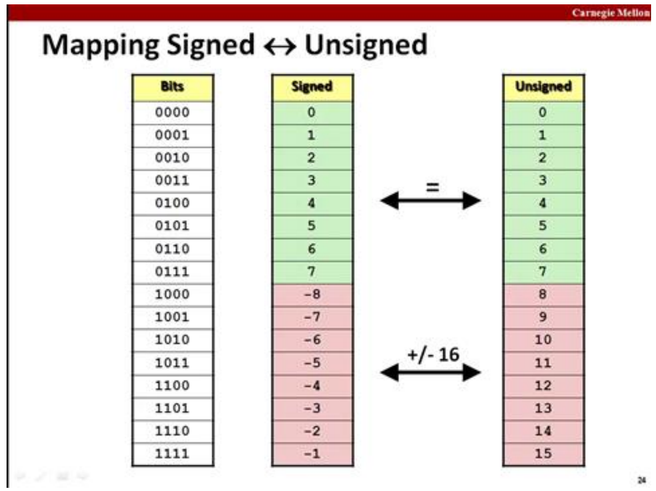

Write “ABCD”
char[8]
int_8[4]
uint_32[1]
int_16[2]
uint_16[2]
A
B
C
‘\0’
(65,x41)
(66,0x42)
(67,0x43)
(0,x00)
Last byte overwritten to null

Union

Write “©ABC”

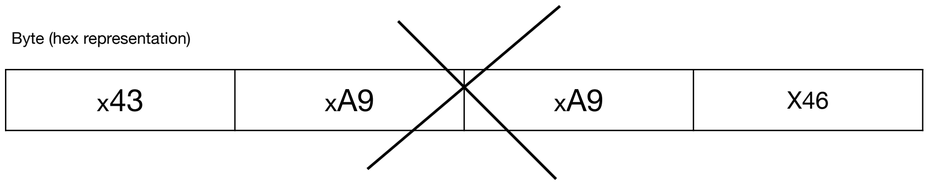


Signed, Unsigned, 2s-Compliment



Two's complement binary	Decimal
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

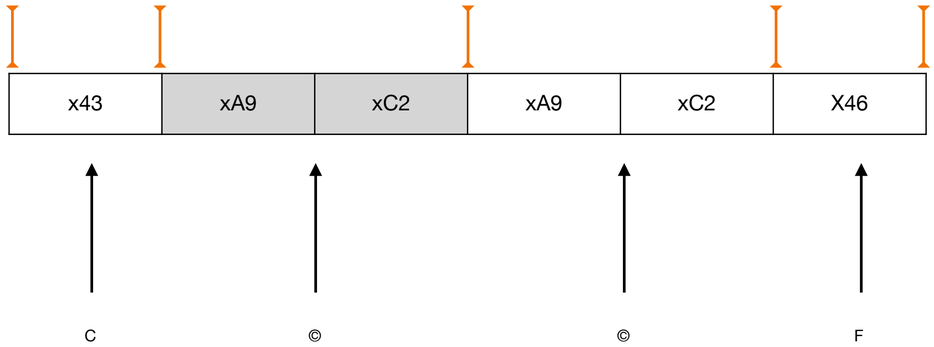
Struct packed



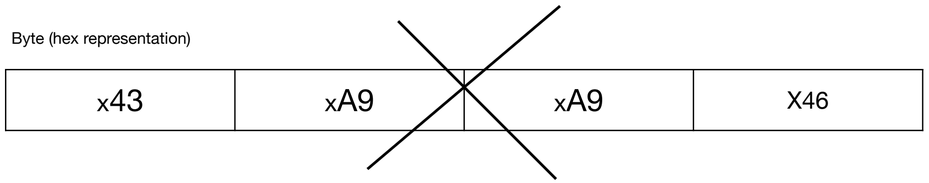
Write “C©©F”

Byte Aligned

Special characters require 2 bytes



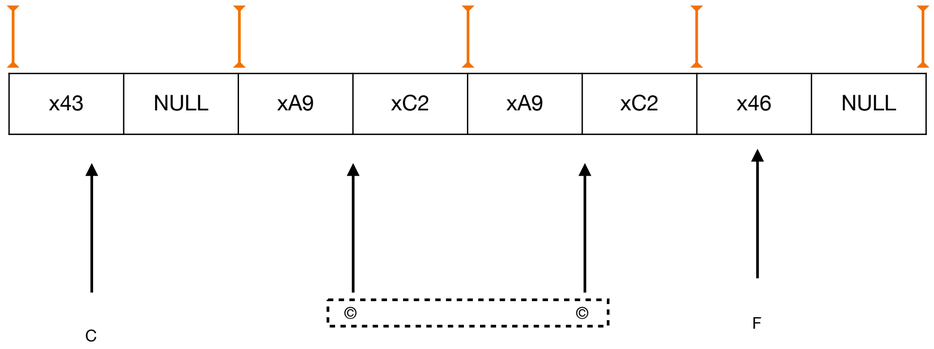
Struct packed



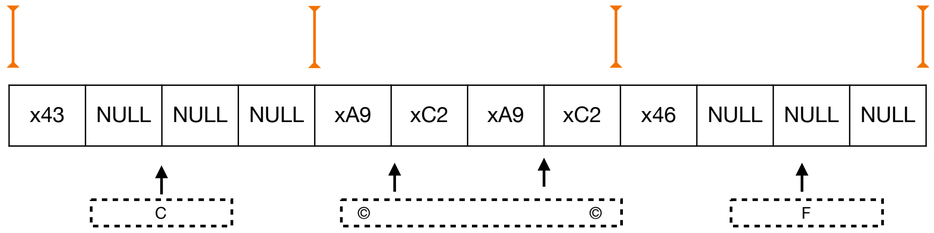
Special characters require 2 bytes

Write “C©©F”

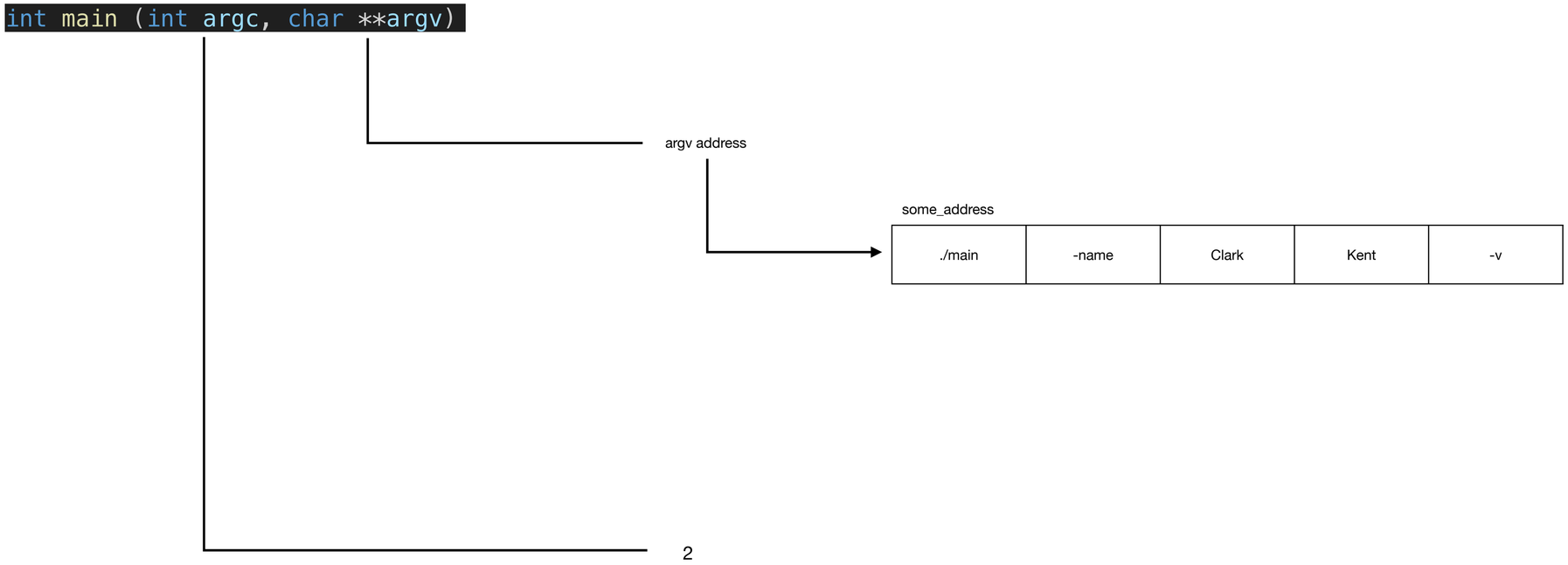
2 Byte Aligned



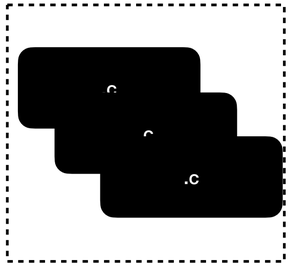
Unpacked
(System add padding)



Argc Argv


int main (int argc, char **argv)
argv address
some_address
./main
-name
Clark
Kent
-v
2

lldb



Reminder Add -g when compiling object files

frame pointer - 0xf3f

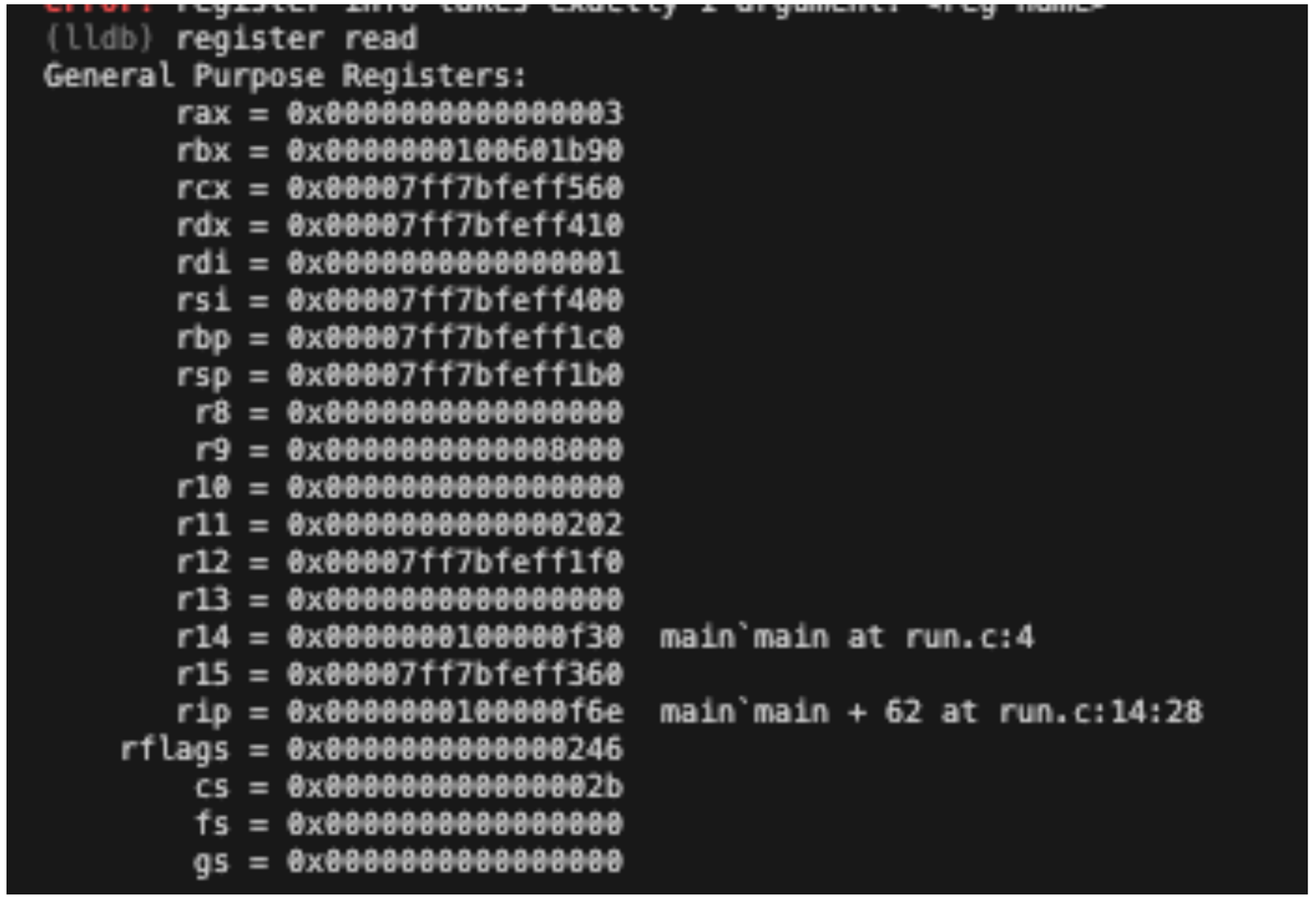
```
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
    frame #0: 0x0000000100000f3f main`main at run.c:6:7
   3
   4    int main () {
   5
-> 6      int count = 0;
   7
   8      for(int i = 0; i < 3; i++) {
   9
Target 0: (main) stopped.
```

next

frame pointer - 0xf46

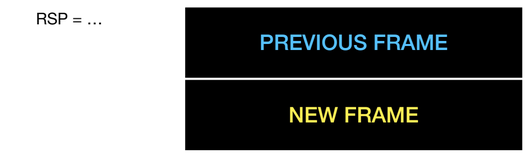
```
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
    frame #0: 0x0000000100000f46 main`main at run.c:8:11
   5
   6      int count = 0;
   7
-> 8      for(int i = 0; i < 3; i++) {
   9
   10       count += 0;
   11
Target 0: (main) stopped.
```

GBR

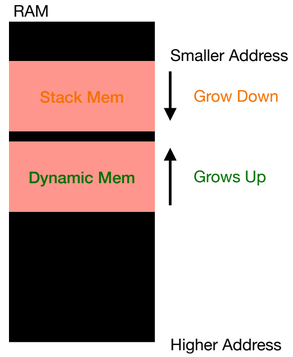

(lldb) register read
General Purpose Registers:
rax = 0x0000000000000003
rbx = 0x0000000100601b90
rcx = 0x00007ff7bfeff560
rdx = 0x00007ff7bfeff410
rdi = 0x0000000000000001
rsi = 0x00007ff7bfeff400
rbp = 0x00007ff7bfeff1c0
rsp = 0x00007ff7bfeff1b0
r8 = 0x0000000000000000
r9 = 0x0000000000008000
r10 = 0x0000000000000000
r11 = 0x0000000000000202
r12 = 0x00007ff7bfeff1f0
r13 = 0x0000000000000000
r14 = 0x0000000100000f30 main`main at run.c:4
r15 = 0x00007ff7bfeff360
rip = 0x0000000100000f6e main`main + 62 at run.c:14:28
rflags = 0x0000000000000246
cs = 0x000000000000002b
fs = 0x0000000000000000
gs = 0x0000000000000000

64 Bit

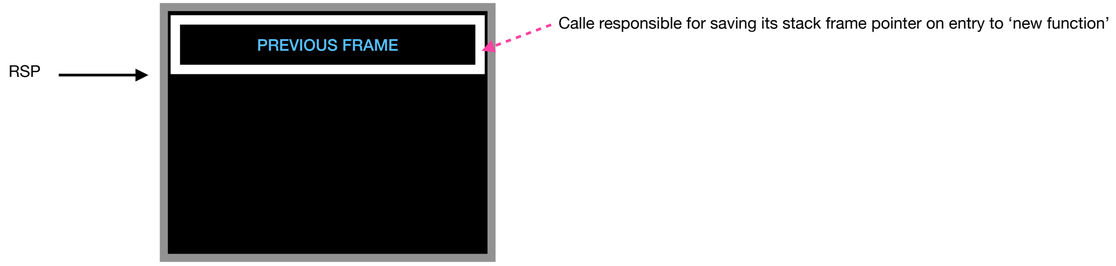
RAX	scratchpad register	Used as accumulator
RBX	Base Register	
RCX	Cycle Counter Register	Used as the cycle counter by loop instruction
RDX	Data Register	Holds data for arithmetic operation (i.e multiplication/division)
RDI	Destination Index	Used in repetitive instructions copying bytes from source to destination (e.g. MOVSB instruction)
RSI	Source Index	
RBP	Frame Pointer	Points to current stack frame (I.e. region of stack functions uses)
RSP	Stack Pointer	Points to top of the stack
R8	Additional Registers	
R9		
R10		
R11		
R12		
R13		
R14		
R15		
RIP	Instruction Pointer	Stores the address of the next instruction to be executed in memory (Contains offset within code segment of the memory)
RFLAGS	Status Register	current state ofcpu (carry flag, parity flag)
CS	Code Segment	Used to address code segment of the memory (location in memory where code is stored)
FS	Segment Registers	Commonly used by OS kernels to access thread-specific memory
GS	Segment Registers	



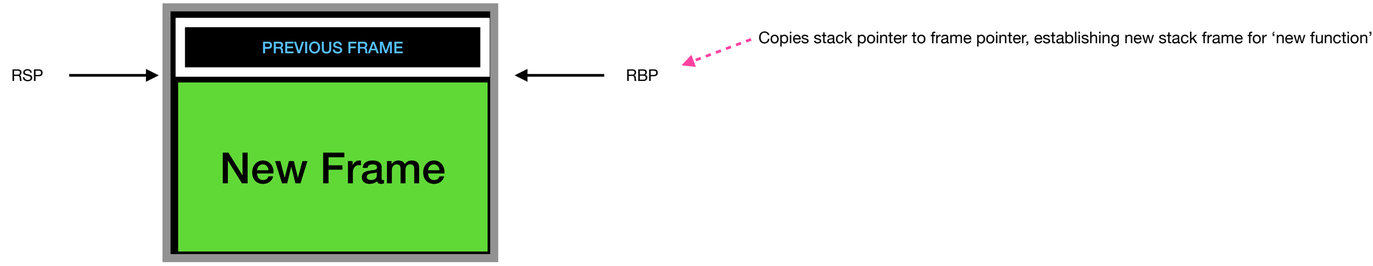

```
1   pushq   %rbp        ; Save address of previous stack frame
2   movq    %rsp, %rbp  ; Address of current stack frame
3   subq    $16, %rsp   ; Reserve 16 bytes for local variables

    ; ... function ...

    movq    %rbp, %rsp  ; \ equivalent to the
    popq    %rbp        ; / 'leave' instruction
    ret
```



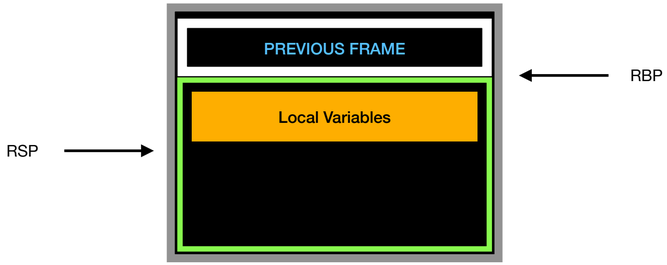
1



2

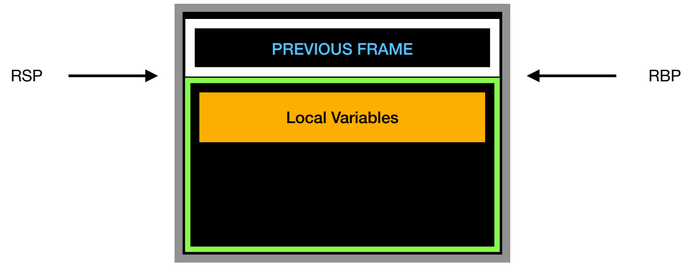


3

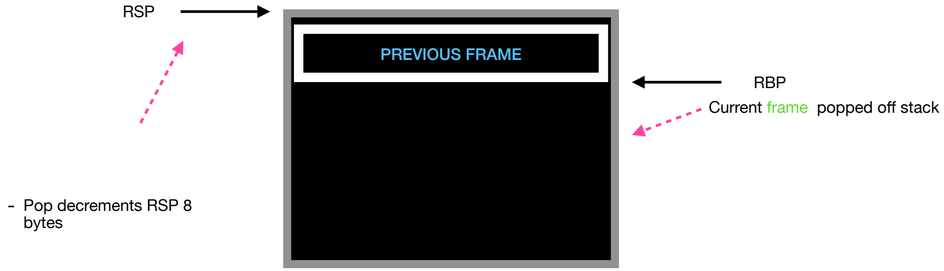


```
1 →   pushq   %rbp        ; Save address of previous stack frame
2 →   movq    %rsp, %rbp  ; Address of current stack frame
3 →   subq    $16, %rsp   ; Reserve 16 bytes for local variables

      ; ... function ...

4 →   movq    %rbp, %rsp  ; \ equivalent to the
5 →   popq    %rbp        ; / 'leave' instruction
6 →   ret
```

4

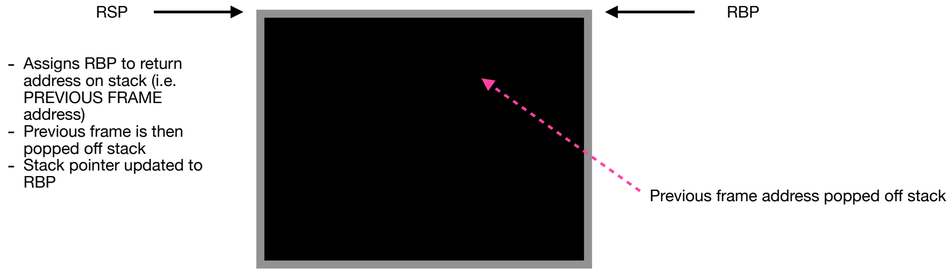


5



- Pop decrements RSP 8 bytes

6



- Assigns RBP to return address on stack (i.e. PREVIOUS FRAME address)
- Previous frame is then popped off stack
- Stack pointer updated to RBP