Memory Management

Linking/Relocation
Loading
Starting
Heap

Baremetal Section

Processor and memory architecture

Peripherals: GPIO, timers, UART

Assembly language and machine code

From C to assembly language

Functions and stack frames

Serial communication and strings

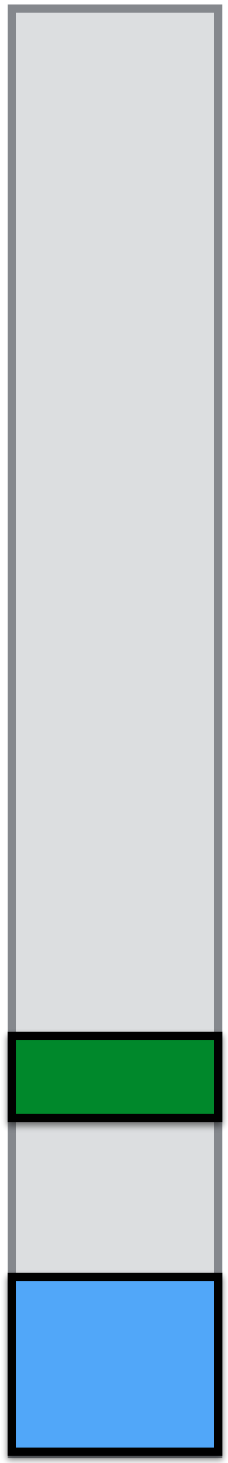
Modules and libraries: Linking

Memory Map: Linking, starting, and the heap

gpio
timer
uart
printf
malloc
keyboard
mailbox
fb
gl
console
monitor

The Memory Map

Memory Map



100000000_{16}

4 GB

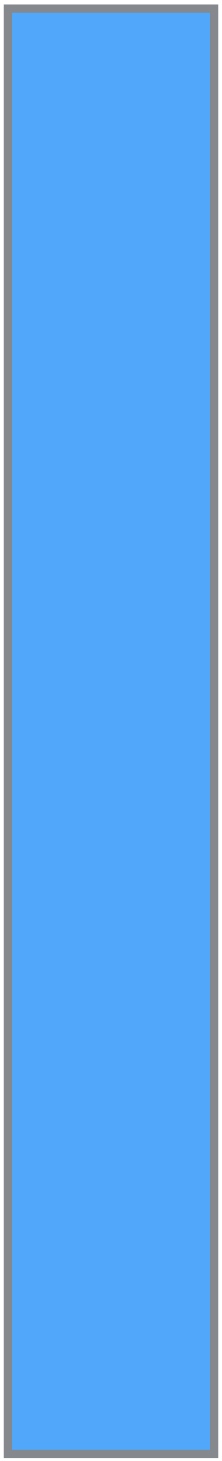
Peripheral Registers

020000000_{16}

010000000_{16}

Ref: BCM2835-ARM-Peripherals.pdf

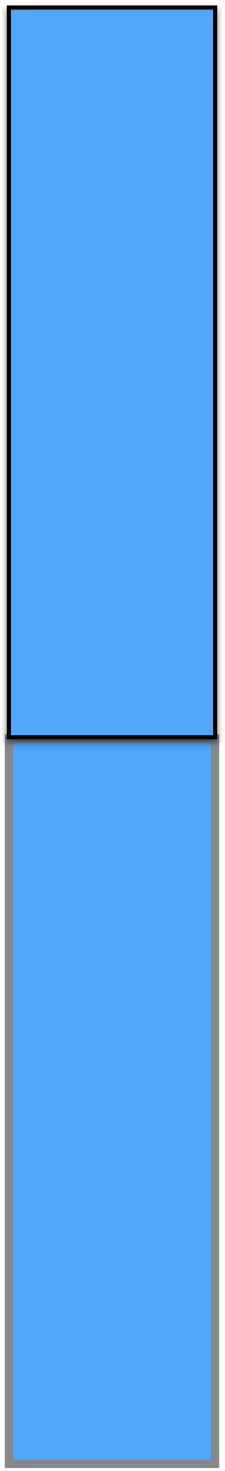
Memory Map



01000000_{16}

256 MB

Memory Map



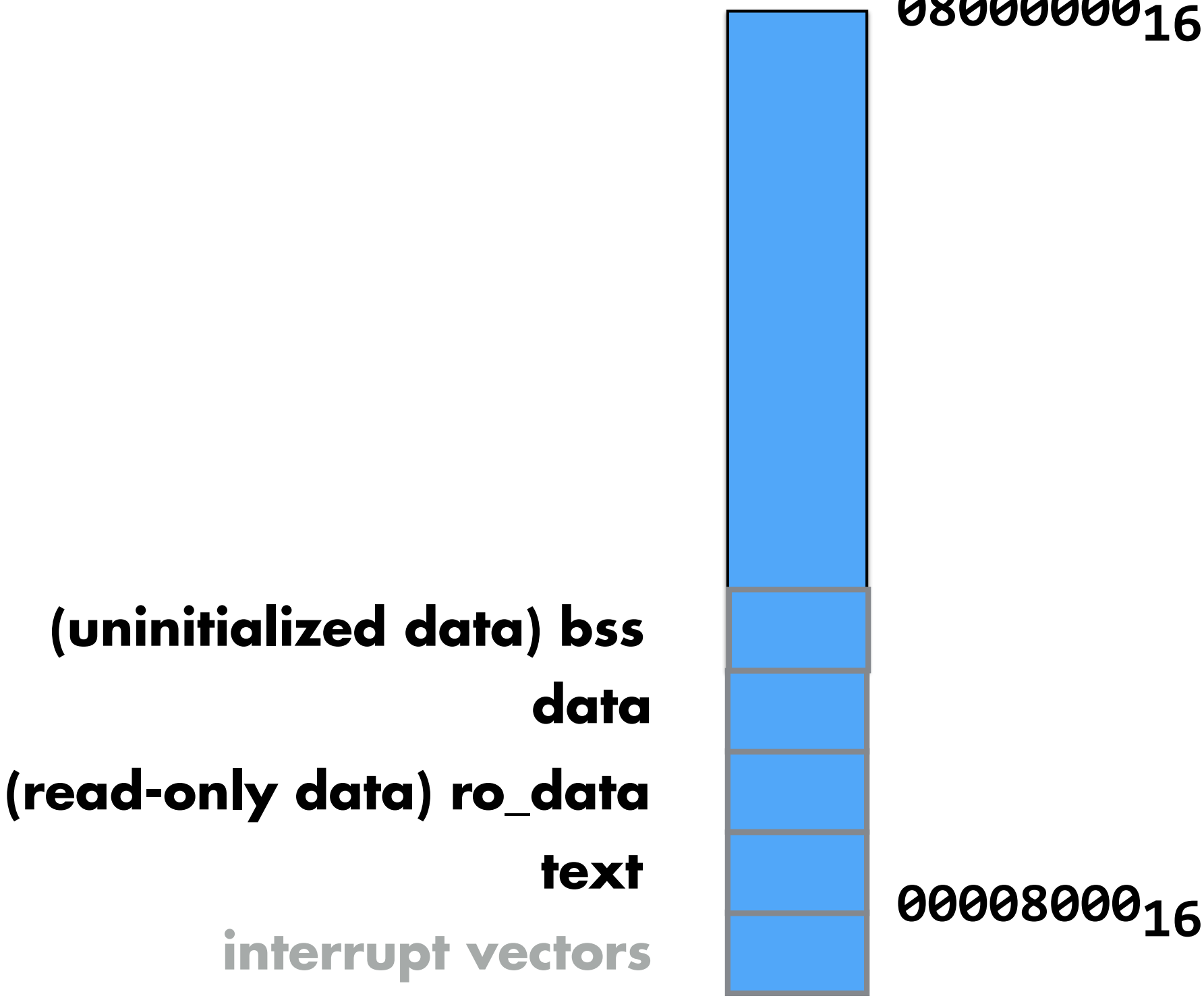
10000000_{16}

256 MB

GPU

08000000_{16}

CPU


08000000_{16}
(uninitialized data) bss
data
(read-only data) ro_data
text
interrupt vectors
00008000_{16}

common

Symbols

Types

- extern (undefined)
- global vs local (static)
- const vs non-const
- initialized vs uninitialized

Guide to Symbols

T/t - text

D/d - read-write data

R/r - read-only data

B/b - bss (*Block Started by Symbol*)

C - common (instead of B)

lower-case letter means `static`

Sections

Instructions go in `.text`

Data goes in `.data`

`const` data (read-only) goes in `.ro_data`

Uninitialized data goes in `.bss`

+ other information about the program

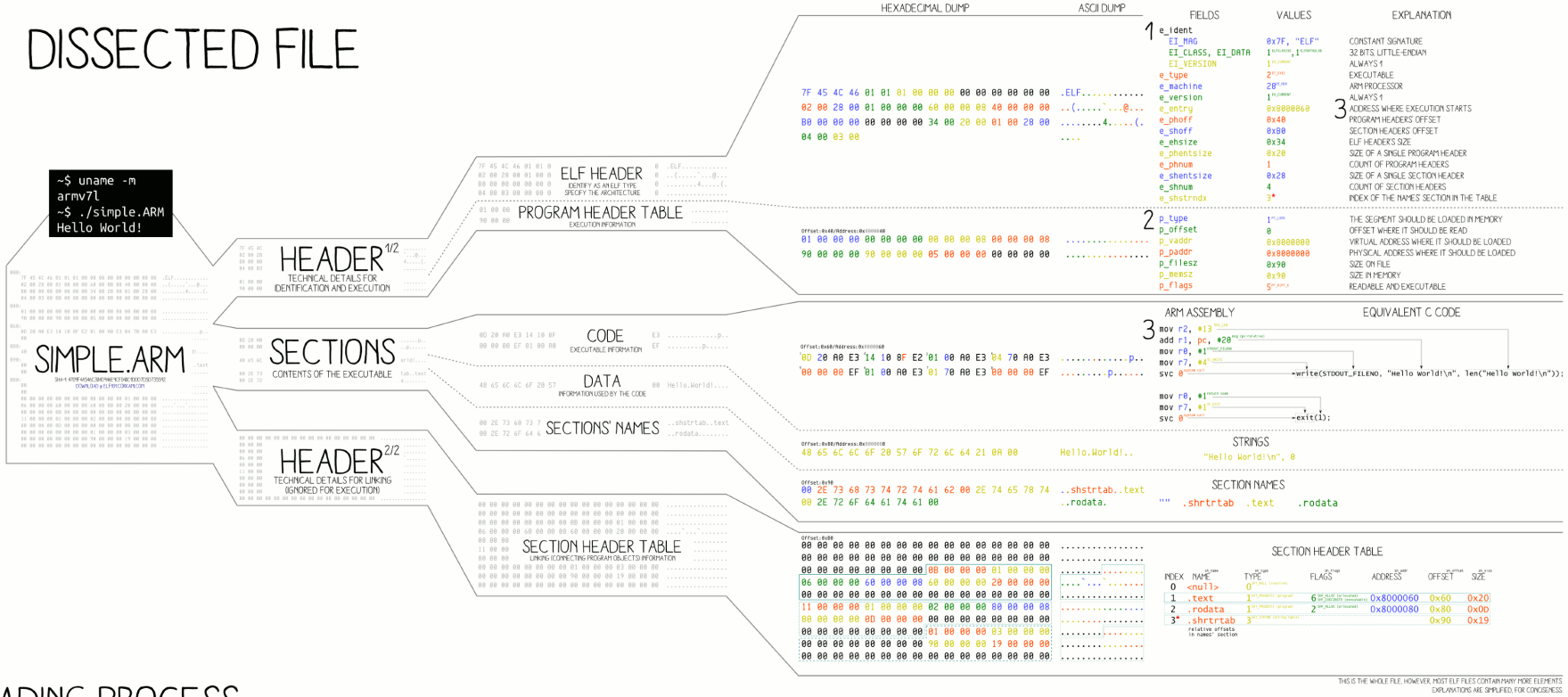
- **symbols, relocation, debugging, ...**

memmap

ELF[101] a Linux executable walk-through

ANGE ALBERTINI
CORKAMI.COM

DISSECTED FILE



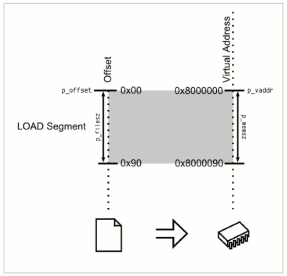
LOADING PROCESS

1 HEADER

THE ELF HEADER IS PARSED
THE PROGRAM HEADER IS PARSED
(SECTIONS ARE NOT USED)

2 MAPPING

THE FILE IS MAPPED IN MEMORY
ACCORDING TO ITS SEGMENT(S)



3 EXECUTION

ENTRY IS CALLED
SYSCALLS[GLOSS] ARE ACCESSED VIA:
- SYSCALL NUMBER IN THE R7 REGISTER[ARM]
- CALLING INSTRUCTION SVC

TRIVIA

THE ELF WAS FIRST SPECIFIED BY U.S.L. AND U.I.[SYSTEM INTERNATIONAL]
FOR UNIX SYSTEM V, IN 1989

THE ELF IS USED, AMONG OTHERS, IN:
- LINUX, ANDROID, *BSD, SOLARIS, BEOS
- PSP, PLAYSTATION 2-4, DREAMCAST, GAMECUBE, WII
- VARIOUS OSES MADE BY SAMSUNG, ERICSSON, NOKIA,
- MICROCONTROLLERS FROM ATMEL, TEXAS INSTRUMENTS

VERSION 1.0A
2013/12/06

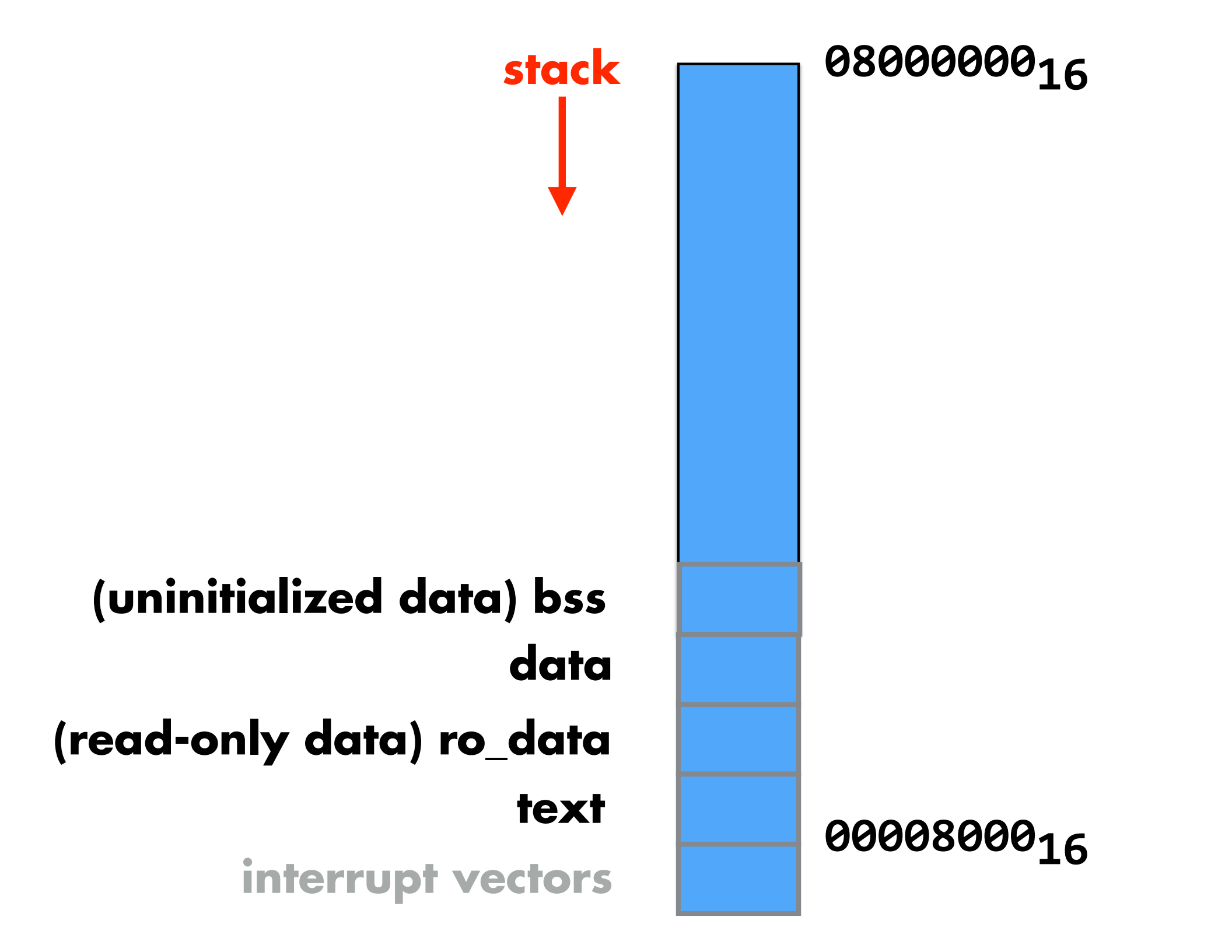
Relocation

```
// start.s

.globl _start
start:
    mov sp, #0x8000
    bl cstart
hang:
    b hang
```

```
// Disassembly of start.o (start.list)

00000000 <_start>:
   0:   mov sp, #32768  ; 0x8000
   4:   bl  0 <main>

00000008 <hang>:
   8:   b   8 <hang>

// Note: the address of main is 0
// Why? It doesn't know where main is!
```

```
// Disassembly of blink.exe.list

00008000 <_start>:
    8000:   mov sp, #32768  ; 0x8000
    8004:   bl  800c <main>

00008008 <hang>:
    8008:   b   8008 <hang>

0000800c <main>:
    800c:   push    {r3, lr}

// Note: the address of main is #800c
// Now it knows where main is!
```

Starting


stack
08000000_{16}
(uninitialized data) bss
data
(read-only data) ro_data
text
interrupt vectors
00008000_{16}

blink/start.s

blink/cstart.c

Memory Allocation and the Heap

Why Memory Allocation?

Compile-time memory allocation

Run-time memory allocation

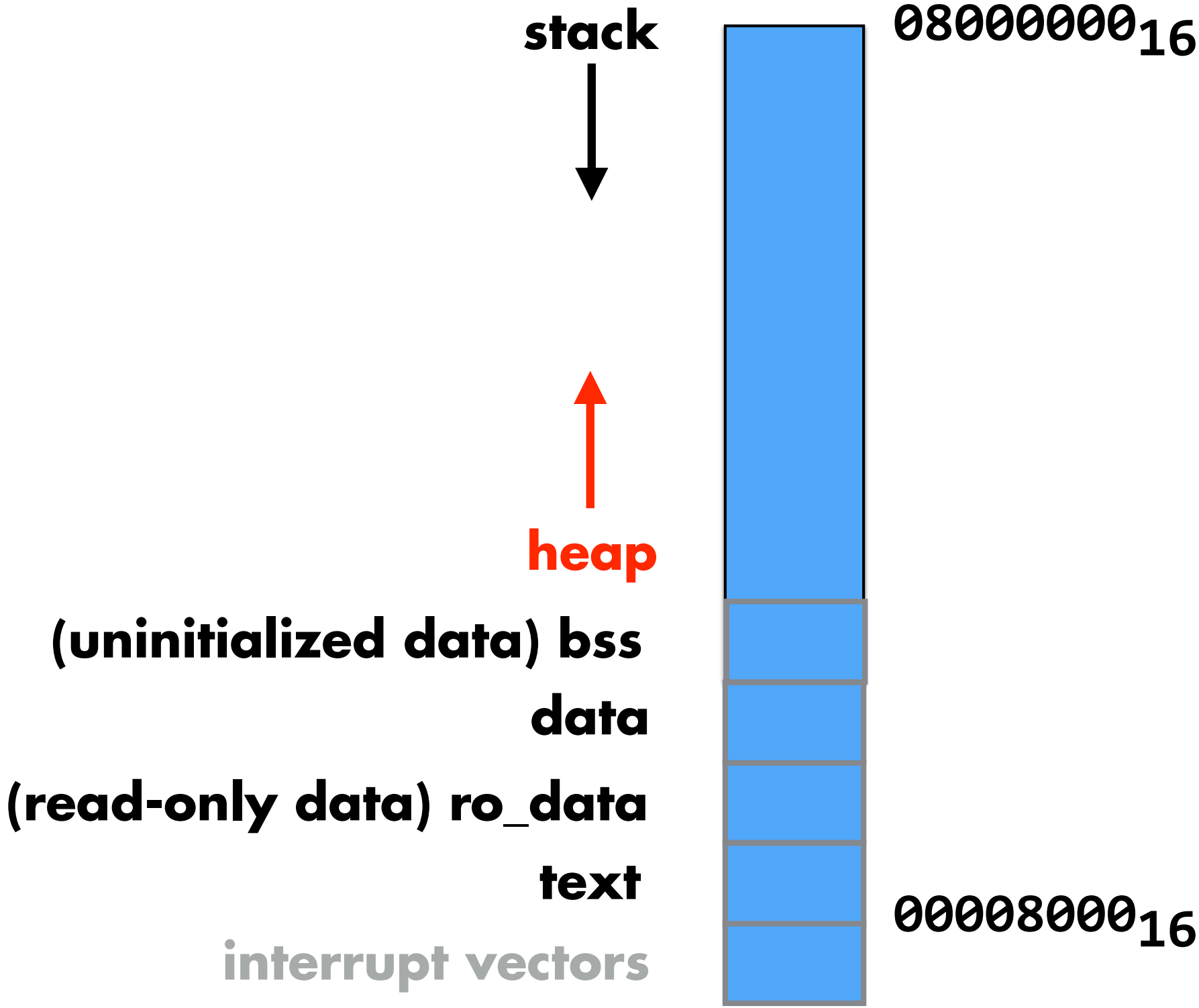
Why?

1. Don't know the size of an array

2. Dynamic data structures such as lists and trees

API

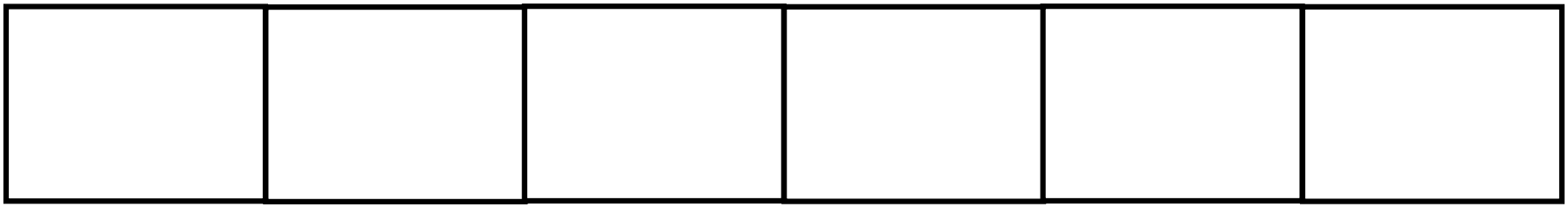
```
void *malloc( size_t size );
void free( void *pointer );
void *realloc( void *pointer,
               size_t size);

// Note that void* is a generic pointer
```

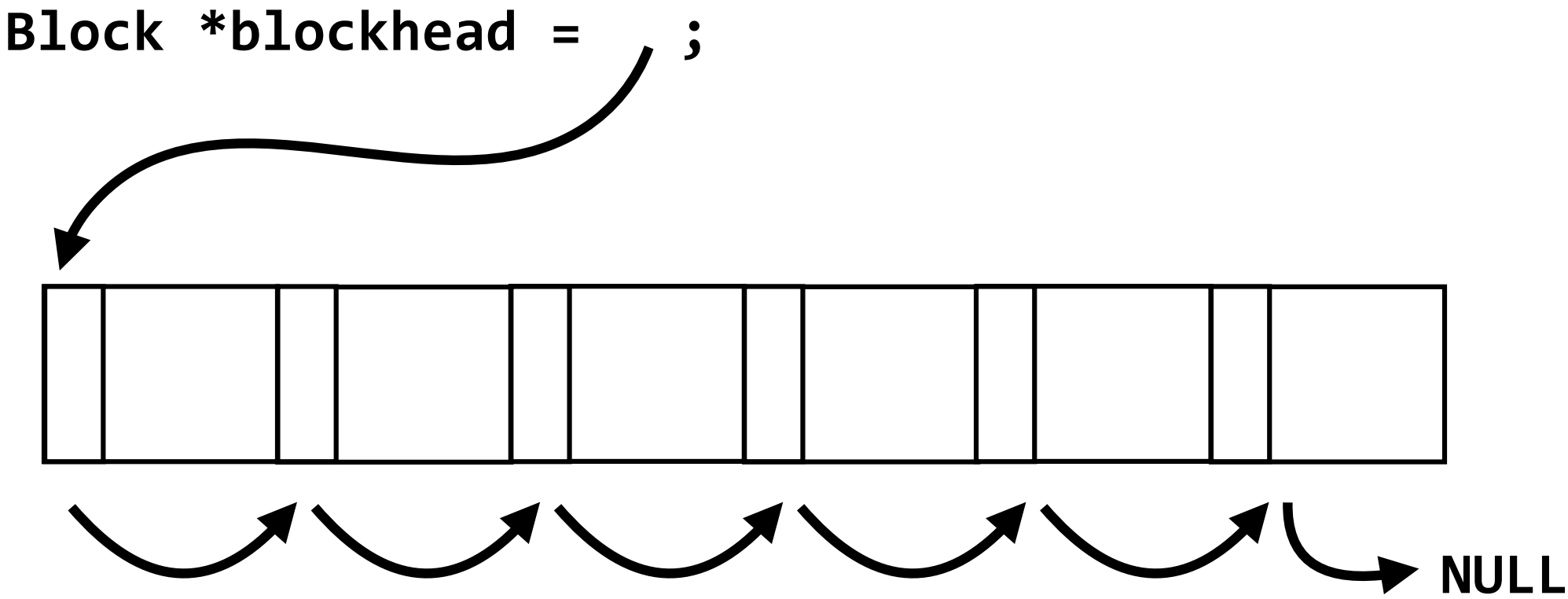

stack
08000000_{16}
heap
(uninitialized data) bss
data
(read-only data) ro_data
text
interrupt vectors
00008000_{16}

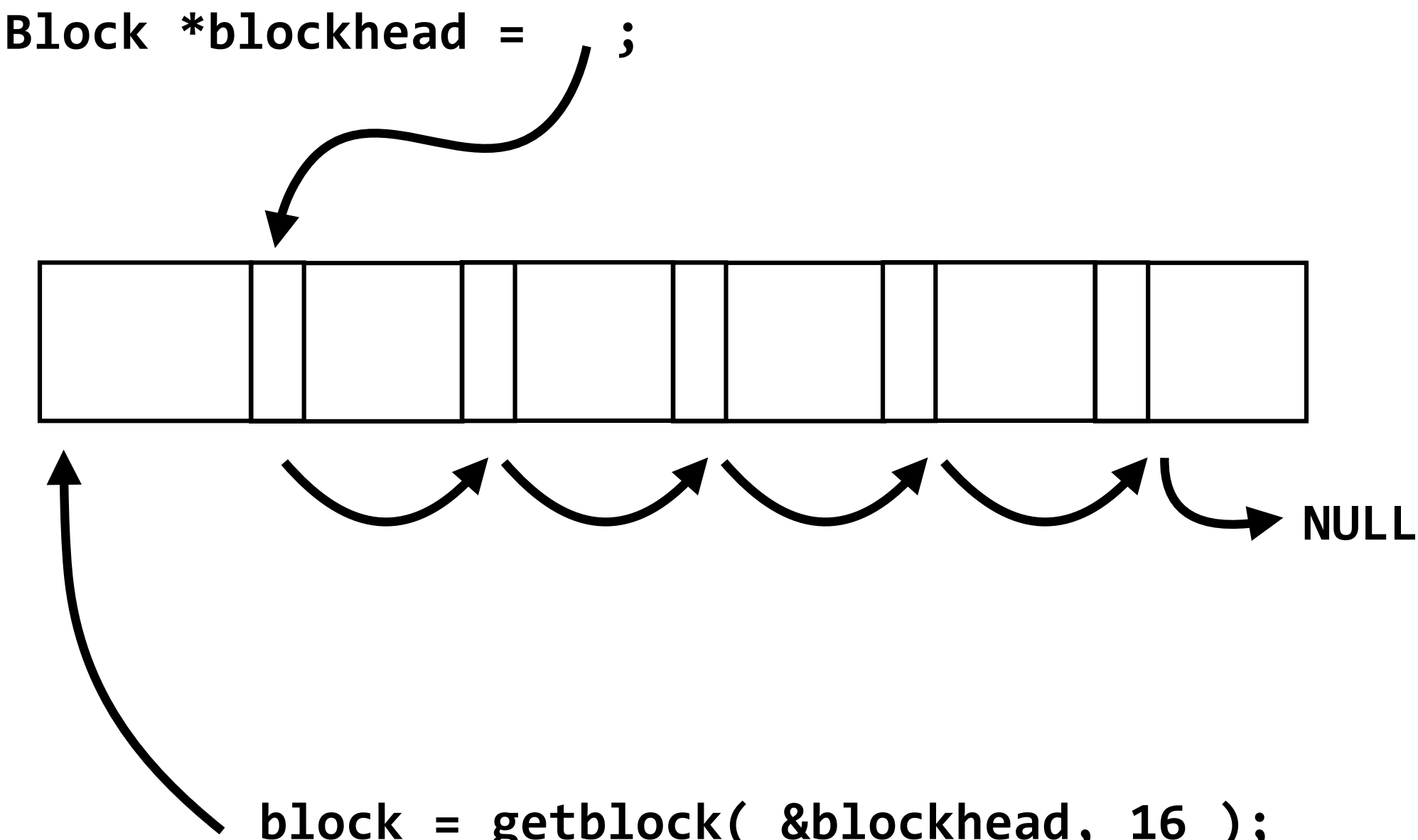
sbrk

block

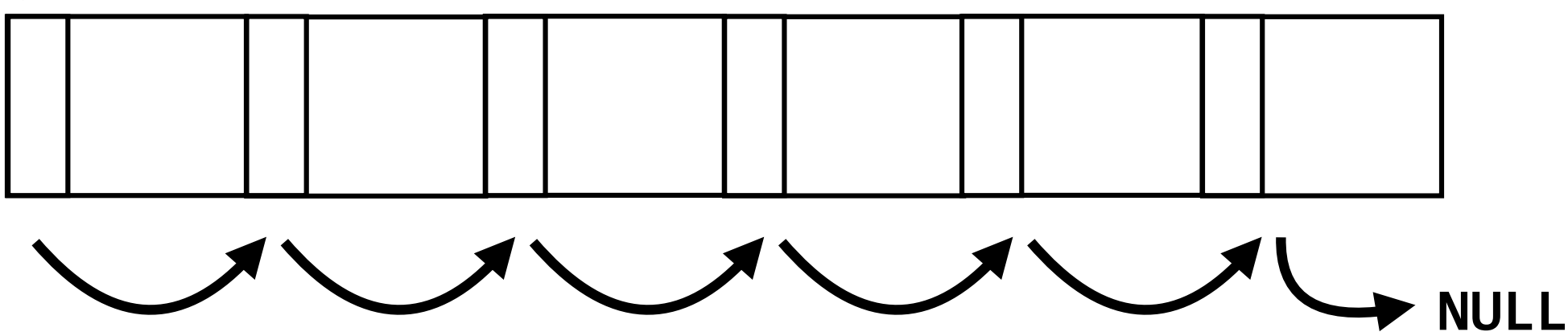
```
newblock( nelements=6, nsize=16 );
```




Block *blockhead = ;
NULL


Block *blockhead = ;
NULL
block = getblock(&blockhead, 16);

```
Block *blockhead =    ;
```




```
getblock( &blockhead, block );
```

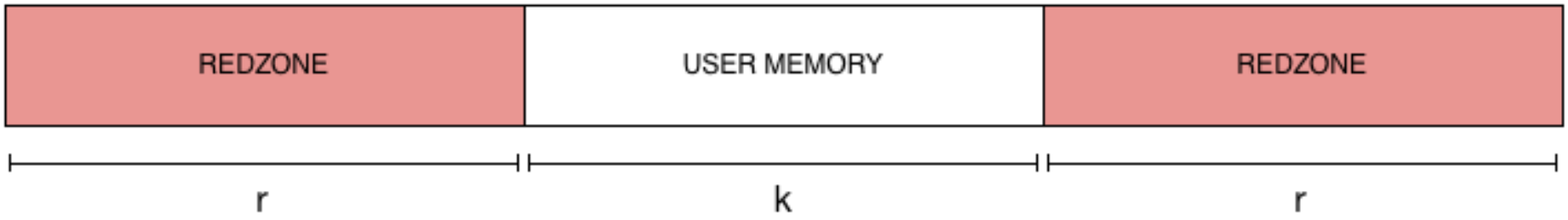
Variable Size `malloc/free`

just `malloc` is easy 🙂

`malloc` with `free` is hard 😕

- **`free` returns blocks that can be re-allocated**
- **`malloc` should search to see if there is a block of the sufficient size. Which block should it choose (best-fit, first-fit, largest)?**
- **`malloc` may use only some of the block. It splits the block into two sub-blocks of smaller sizes**
- **splitting blocks causes fragmentation**

Memory Corruption



Write special value (`#0xDEADBEEF`) to red zone
Look for unintended writes to the red zones

`red zone malloc`
`Assignment 4`