
# Dynamic Memroy Allocation. Strings

Alexandru Copoț (alex.mihai.c@gmail.com)

10 July 2013

ROSEdu Summer Workshops

# Overview

# Memory Management

- Hardware level
- Operating system level
- **Application level**

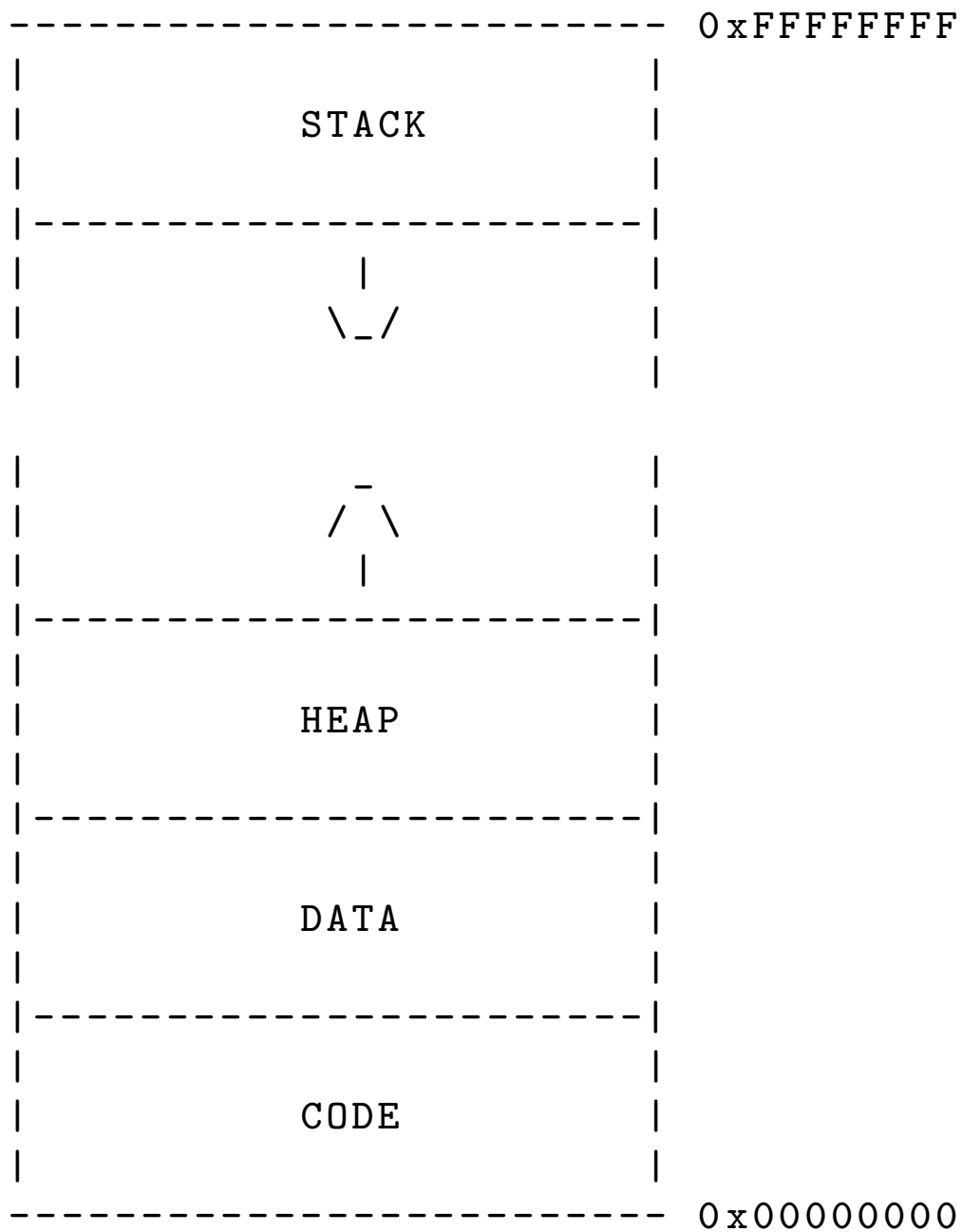
Common issues:

- Allocation
- Freeing
- Fragmentation

# Dynamic Memory Allocation in C

```
void *malloc(size_t sz);
void *calloc(size_t n, size_t sz);
void *realloc(void *ptr, size_t sz);
void free(void *ptr);
```

# Address Space



# Overview

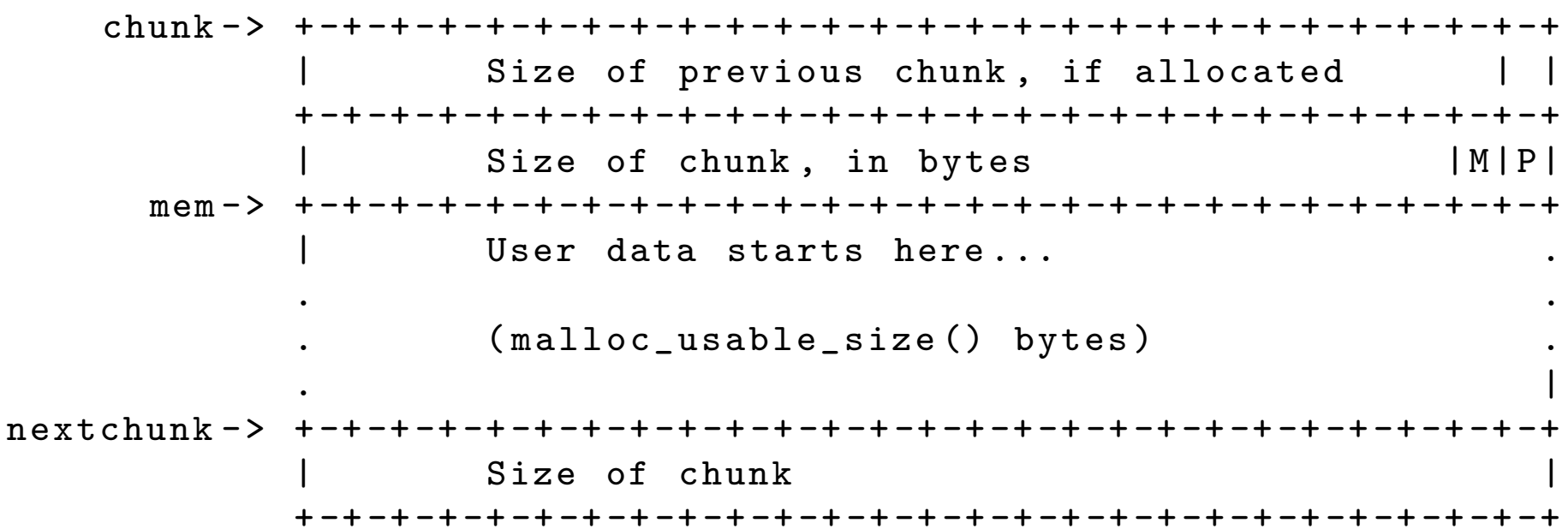
# Initialization

- malloc() does not initialize the memory
- Strings. Bugs with strcpy(), strcat()
- Information leaks

# Incorrect Use of the API

- malloc(-1)
- malloc(0)
- realloc(p, 0)
- malloc(a * b)
- ptr = realloc(ptr, newsize)

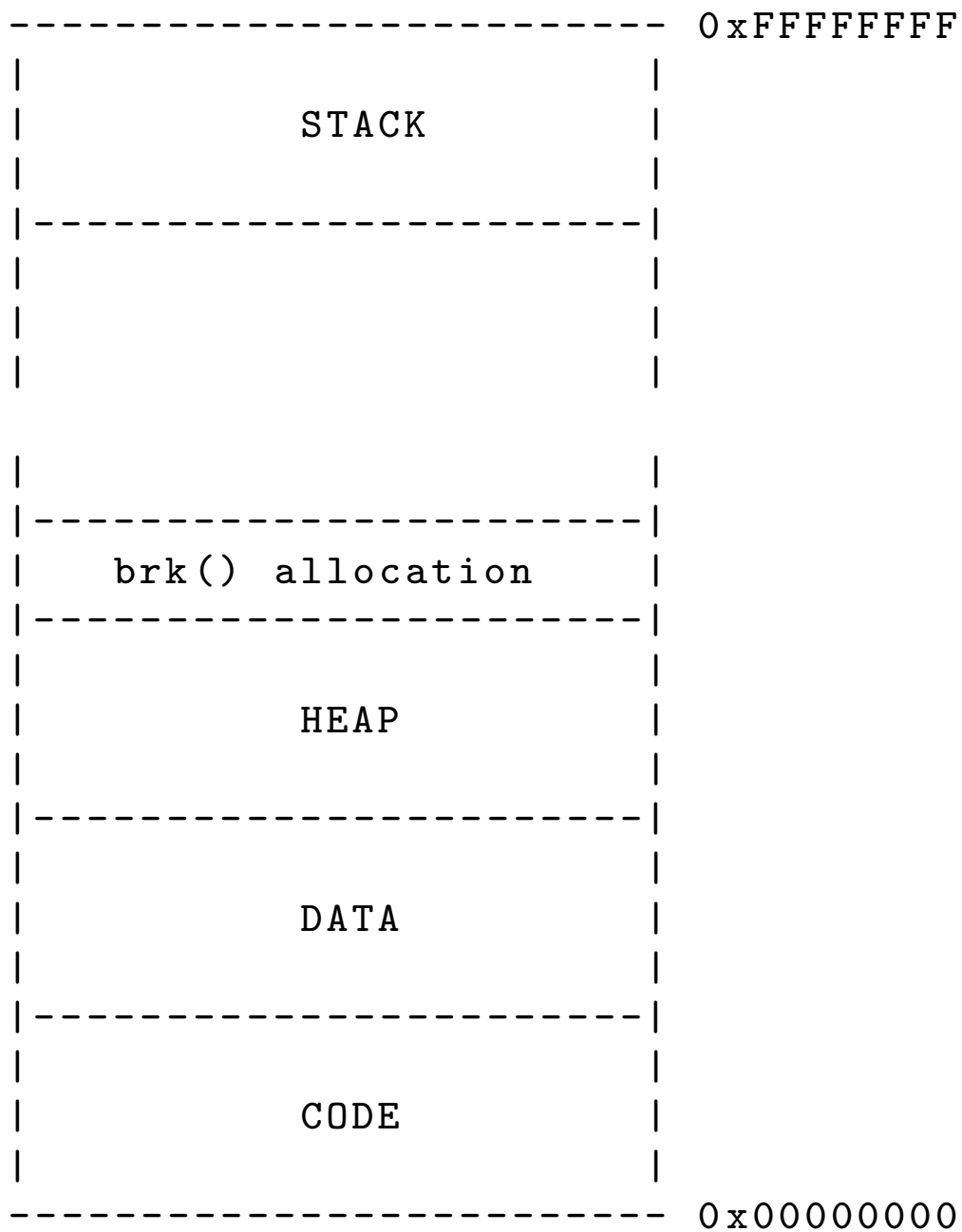
# Doug Lea's Allocator



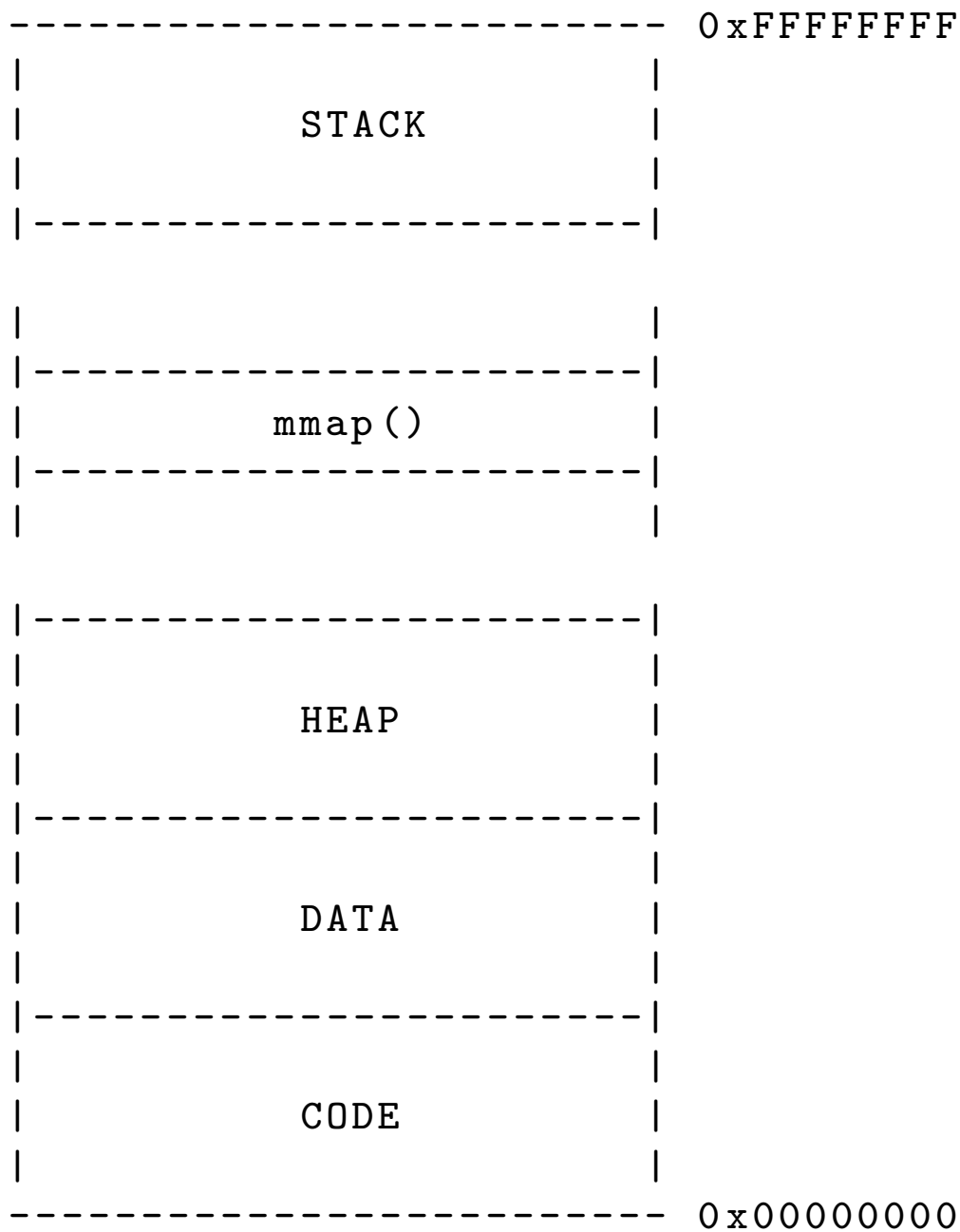
# Doug Lea's Allocator Implementation

- Linked list of free chunks - "bins"
- Small requests (<256K): use bins, call brk()
- Really large requests: mmap()
  - Page sized allocations (usually 4K)
  - Demand paging

# malloc() ->brk()


0xFFFFFFFF
STACK
brk() allocation
HEAP
DATA
CODE
0x00000000

# malloc() ->mmap()



# Use after free. Double free

- Reading from a freed area won't always produce a crash
- Writing might
- Corupting the state of the allocator

# Overview

# Strings

- Where is the memory allocated and where it is freed ?
- What's the allocated size ?
- What's the length of the string ?

# Memory Allocation Models

- Caller allocates, caller frees
  - Most functions in <string.h>
- Callee allocates, caller frees
  - `strdup()`
  - useful when you don't know how much to allocate
- Callee allocates, callee frees

# String API

- Use `strncpy(3)`, `strlcpy(3)`
- Always keep the allocated size
  - Don't `realloc()` with only `strlen()`
  - Don't `strcat()` over the allocated size
- Never use `gets()`
- Remeber to `free()` `strdup()`-ed strings

# Common Errors

- Off-by-one
- Invalid write
- Truncation
- Missing NULL terminator

# Off-by-one. Missing NULL Terminator

```
char *msg = malloc(10);
int i;

if (msg == NULL)
  return;

for (i = 0; i < 10; i++)
  msg[i] = 'a';
```

# Missing NULL Terminator

```
char *strncpy(char *dest, const char *src, size_t n)
...
    char msg[10];
    strcnpy(msg, "01233456789", 10);
...
```

- `strncpy` does not add NULL terminator after `n`

# Truncation

```
char *msg = malloc(10);
...
if (msg == NULL)
  return;
...
msg = realloc(msg, 8);
if (!msg)
  return
// use msg with string functions
```

- `realloc` truncates the string
- The NULL terminator might be gone
- If `realloc` fails, the initial memory is leaked