Memory and C Programming

Robbert van Renesse

What is Memory?

What is Memory?

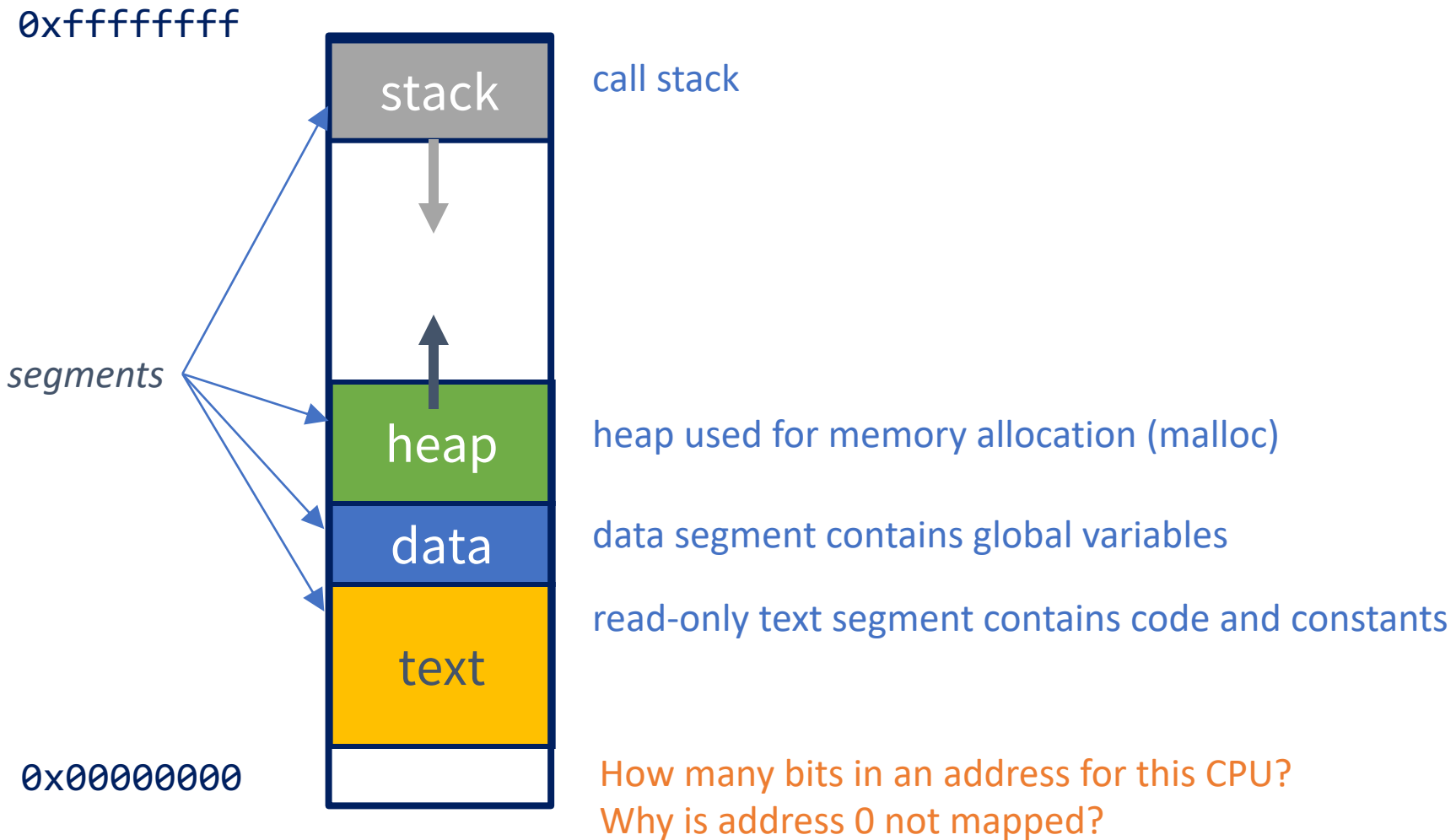- I can't recall…

I remember now:

- Memory is an array of bytes
- An index into this array is called an "address"
- A variable holding an address is called a "pointer"
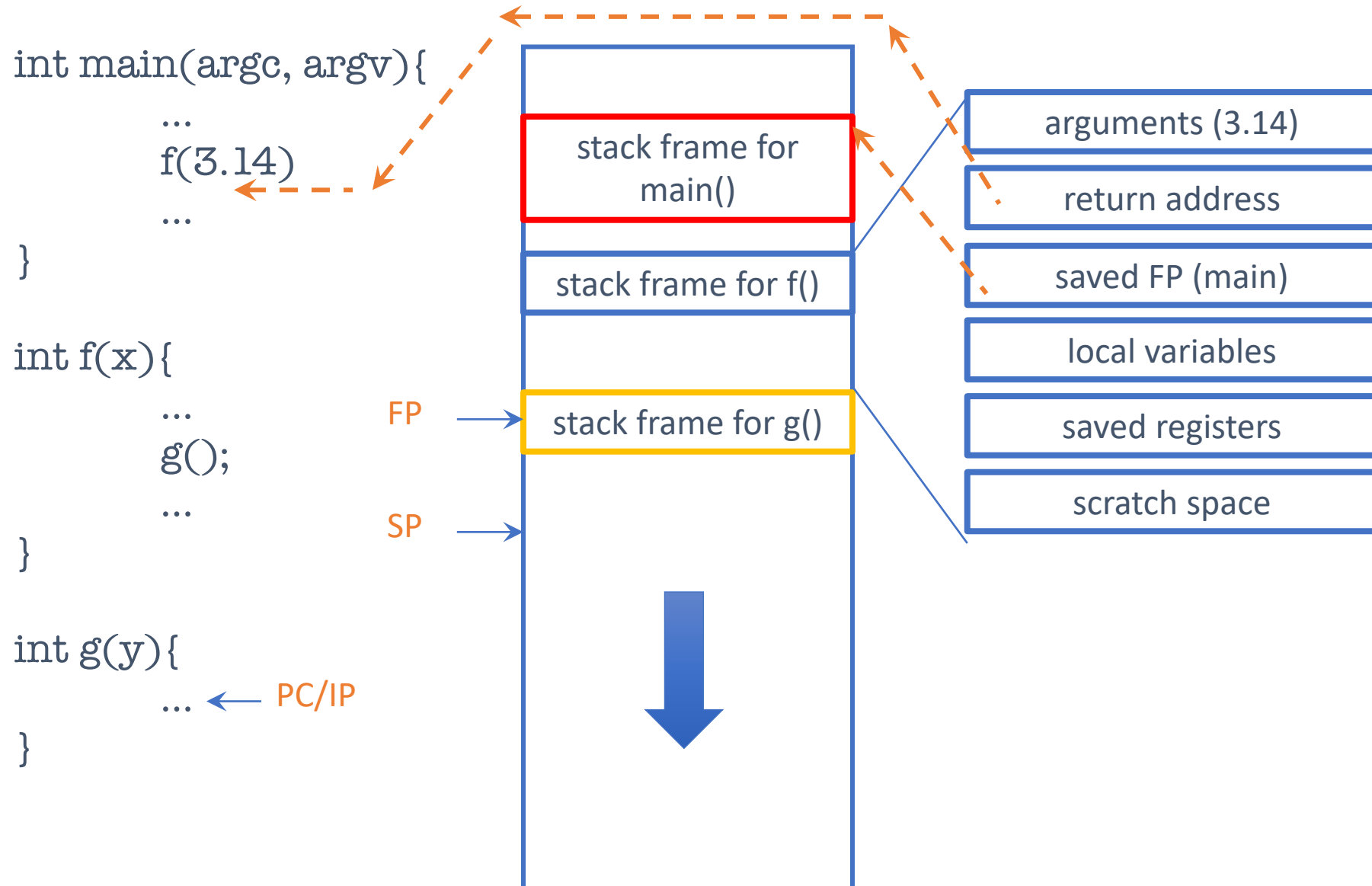
Types of memory

- Code: machine instructions (read-only)
- Read-only data (string constants etc.)
- Global variables
- Heap: dynamically allocated memory
- Stack

You can store your data in global variables, on the heap, or on the stack

Logical view of process memory

0xffffffff

stack — call stack

heap — heap used for memory allocation (malloc)

data — data segment contains global variables

text — read-only text segment contains code and constants

segments

0x00000000

How many bits in an address for this CPU?
Why is address 0 not mapped?

Review: stack (aka call stack)

```
int main(argc, argv){
        ...
        f(3.14)
        ...
}

int f(x){
        ...
        g();
        ...
}

int g(y){
        ...
}
```

PC/IP

Stack
stack frame for main()
stack frame for f()
stack frame for g()

FP → stack frame for g()

SP →

arguments (3.14)
return address
saved FP (main)
local variables
saved registers
scratch space

Review: heap

in use

free

"break"

NULL

"free list"

pointer to next free chunk

start of heap segment
end of data segment

8
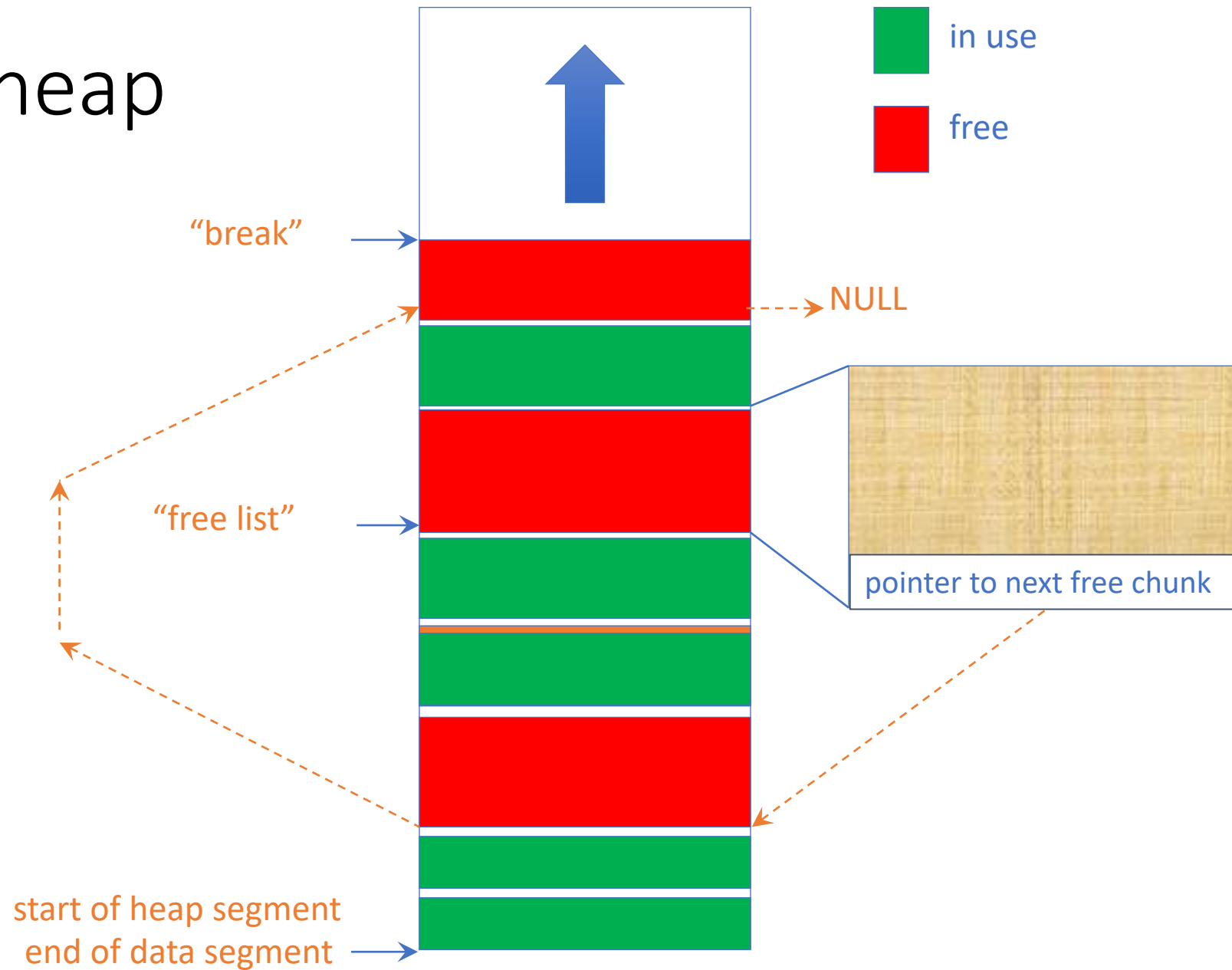
Three types of data memory

	Global	Heap	Stack
allocated	at start of process	using malloc()	at start of function call
initial state	as specified or 0 otherwise	junk	as specified or junk otherwise
released	at end of process	using free()	at end of function call

C Programming

- Like Java programming, but
  - no garbage collection
  - no type safety
  - no object-orientation, polymorphism, container types, …
- Instead:
  - "structs" (instead of classes)
  - pointers (instead of references)
  - malloc/free (instead of **new** and garbage collection)

Hello World

```c
#include <stdio.h>

int main()
{
    printf("Hello World\n");
    return 0;
}
```

Structs

```
struct square
{
    int width, height;
};


typedef struct square square_t;
```

Pointers

```
void f()
{
        square_t sq1, sq2;        // on the stack!
        square_t* ptr = &sq1;   // a pointer

        ptr->width = 300;

        ...
```

malloc/free

```
void f()
{
        square_t* ptr = malloc( sizeof(square_t) );

        ptr->width = 300;

        ...
        free(ptr);
        ...
```

Project P0

- Implement a queue *and* a test program

- Has to be done by each student individually
  - by Wednesday February 8, so you have almost two weeks
  - but start today if only to find out how hard it is to get started on this!
    - you don't want to find this out too late

- Tar file with instructions (README file) on CMS

On Testing

- Testing is at least as important as implementing the data structure itself
- Don't be satisfied with a simple "it compiles and I can do a simple access"
  - think about corner case, and check those
    - dequeue from an empty queue
    - removing the first entry of a queue
    - removing the last entry of a queue
    - …

Linux…

- All projects should be done in a Linux / x86 environment
- Learn to use
  - C compiler: cc (or gcc or clang)
  - Debugging tools: valgrind, gdb
  - Code project tools: make
  - Code repo: git
    - use private github repo on github.coecis.cornell.edu

- Running Linux
  - Can install Linux on your laptop
    - dual boot or virtual machine (Vmware, VirtualBox, Hyper/V, …)
  - Remote access
    - CSUGlab machines