ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

HACKING SCHOOL

Buffer overflow explained

Gabriel Maciá Fernández

UGRCyS

GENIL
GRANADA EXCELLENCE NETWORK OF INNOVATION LABORATORIES

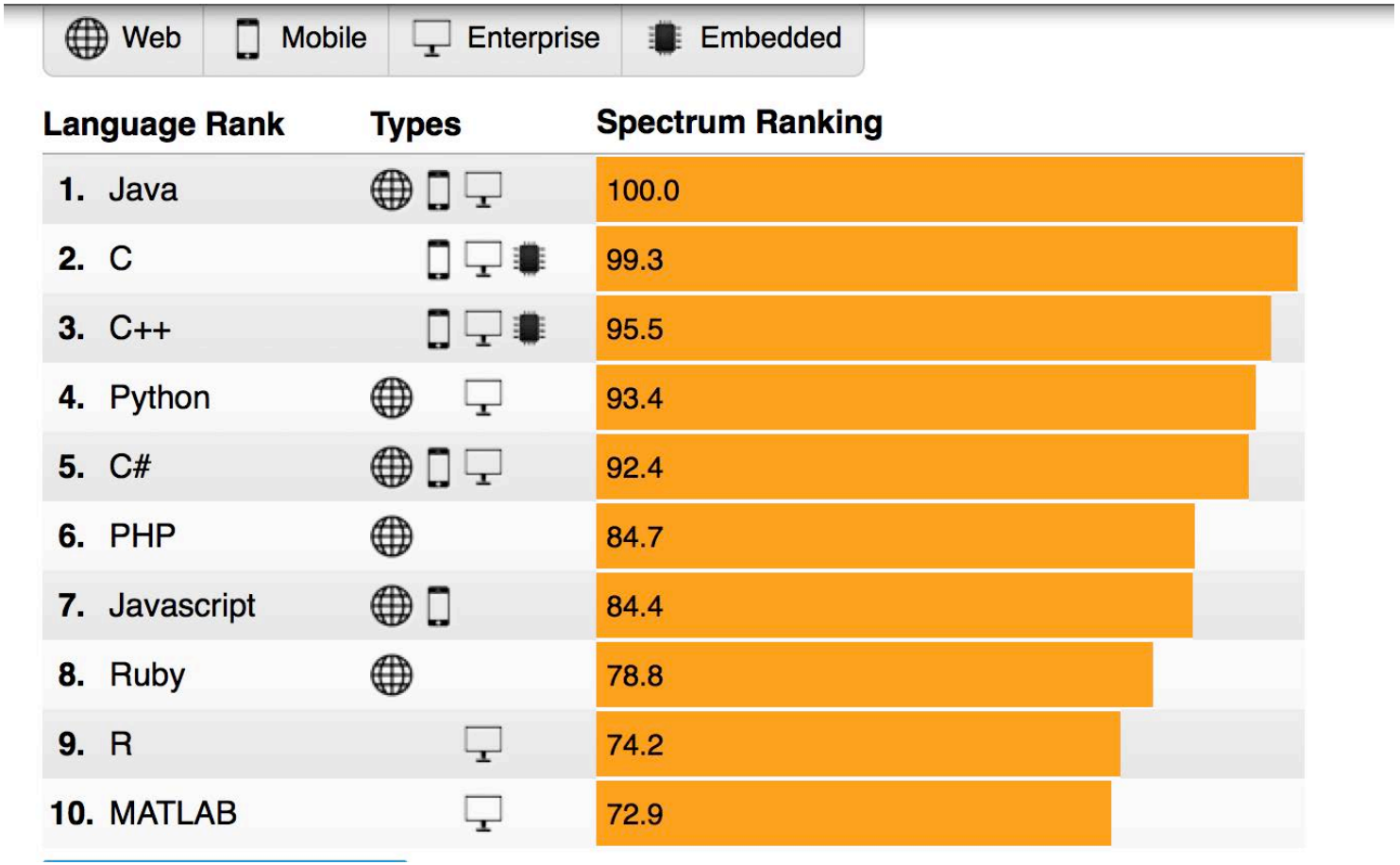
Fundamentals

- Buffer overflow definition:

It's a **bug** that affects low-level code, typically in **C and C++**, with significant **security implications**

Fundamentals

- C and C++ popularity



http://spectrum.ieee.org/static/interactive-the-top-programming-languages

Fundamentals

- Critical systems in C/C++:
  - Most OS kernels and utilities
  - Many high performance servers
    - Microsoft IIS, Apache httpd
    - Microsoft SQL Server, MySQL
  - Many embedded systems

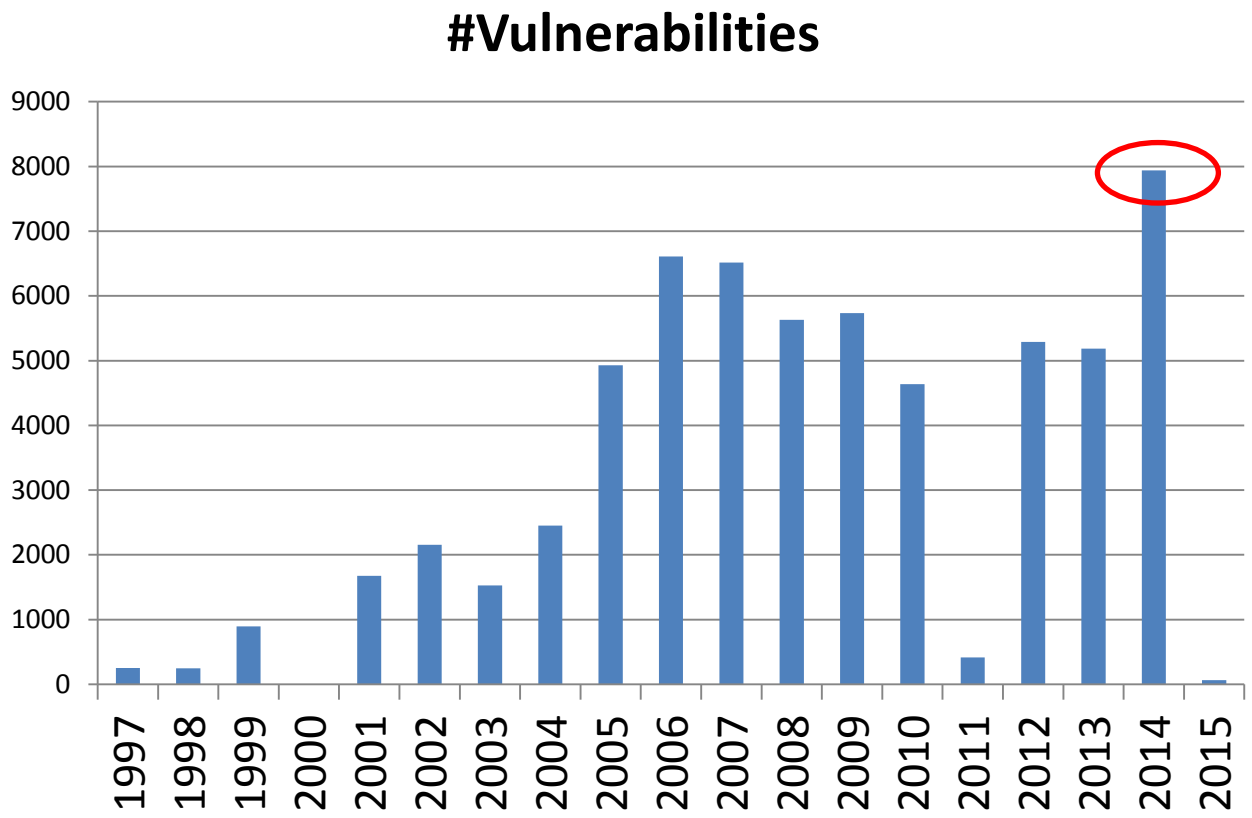
Fundamentals

- Brief history:
  - 1988: Morris worm (fingerd)
    - $10-100M damages
  - 2001: CodeRed (MS-IIS)
    - 300.000 machines infected in 14 hours
  - 2003: SQL Slammer (MS-SQL Server)
    - 75.000 machines infected in 10 minutes

Fundamentals

Fundamentals

- Trend



Source: http://web.nvd.nist.gov/view/vuln/statistics-results?adv_search=true&cves=on&cwe_id=CWE-119

Let’s go into more details

Brief review of C concepts

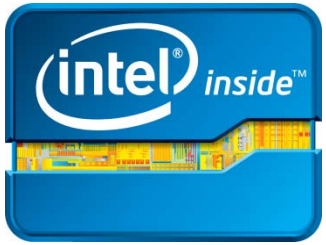
- Int: 32 bits
- Char: 8 bits
- Pointer: 32 bits

```
int *p;
```

- Reference:

```
int a = 3;
int *p = &a;
*p = 2;
```

The Intel 80x86 CPU

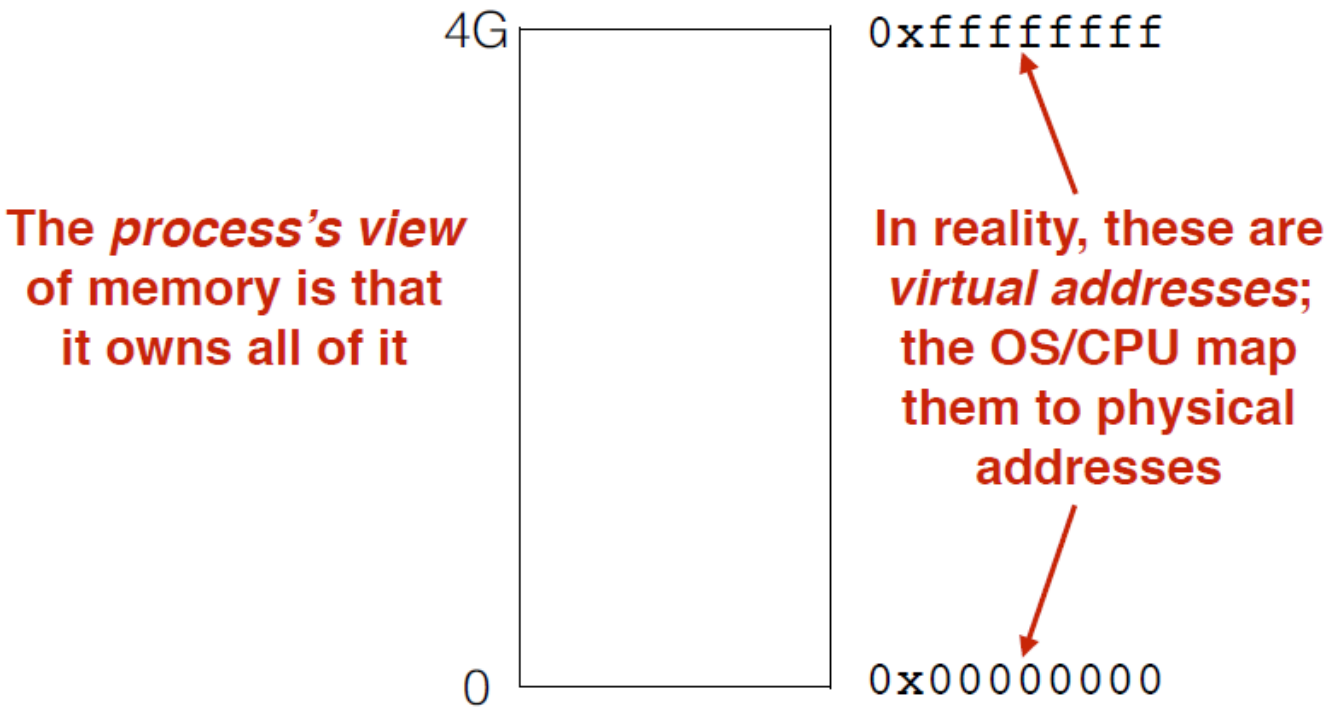


- Registers:
  - General purpose: %eax, %ebx, %ecx, %edx
  - (*Extended*) instruction pointer: %eip
  - (*Extended*) stack pointer: %esp
  - (*Extended*) frame pointer: %ebp
  - Flags: %eflags (ZF, SF, CF, …)

Process memory layout

- Memory addressing (80x86 family): 32 bit

All programs are stored in memory

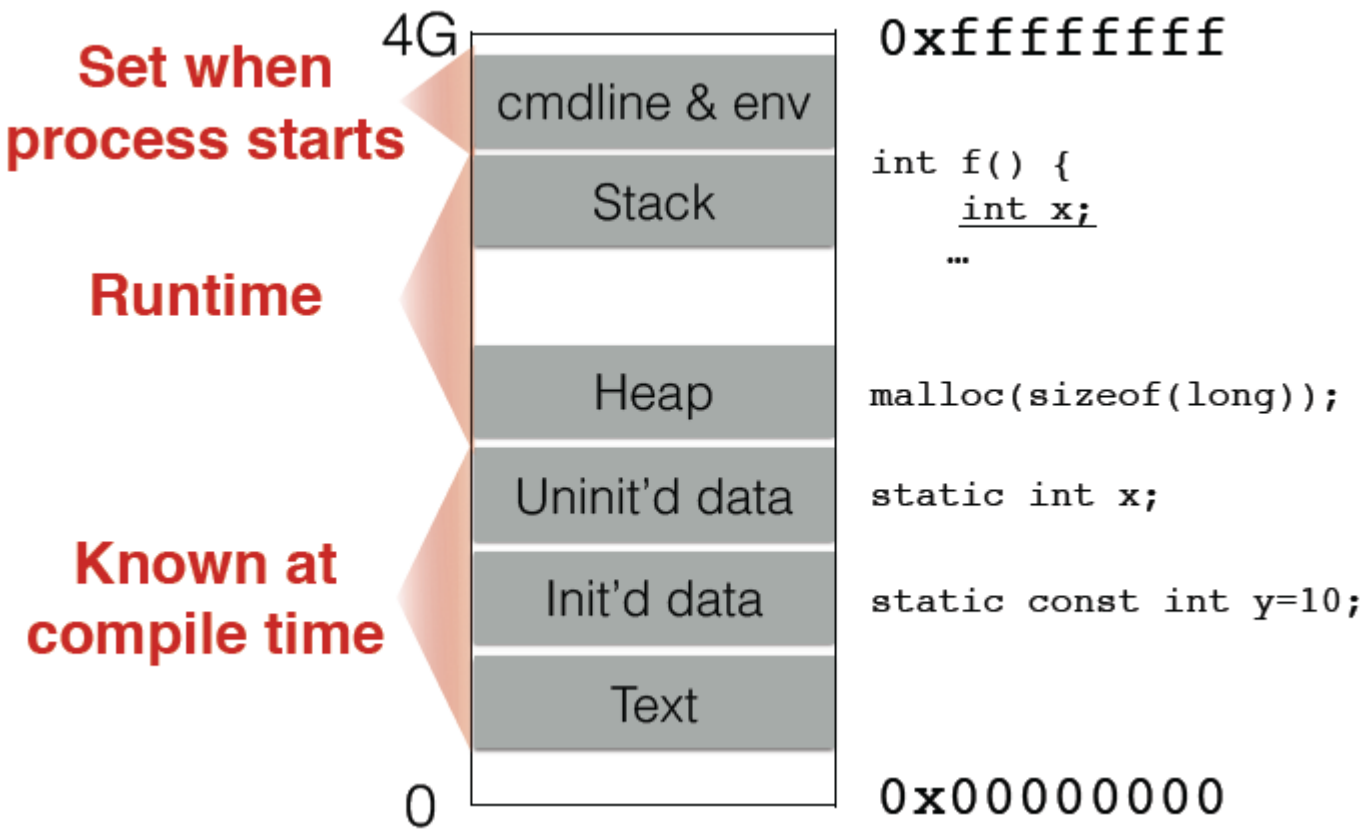


Process memory layout

- Intel uses **little endian** ordering
  - 0x03020100 starting at address 0x00F67B40

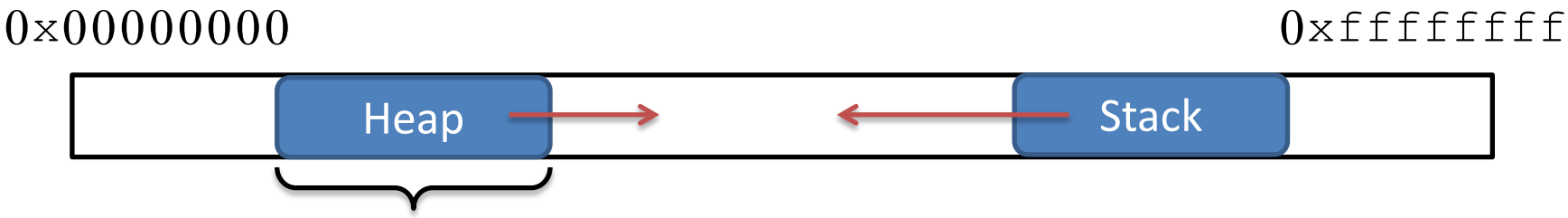
```
0x00F67B40   00
0x00F67B41   01
0x00F67B42   02
0x00F67B43   03
```

Process memory layout



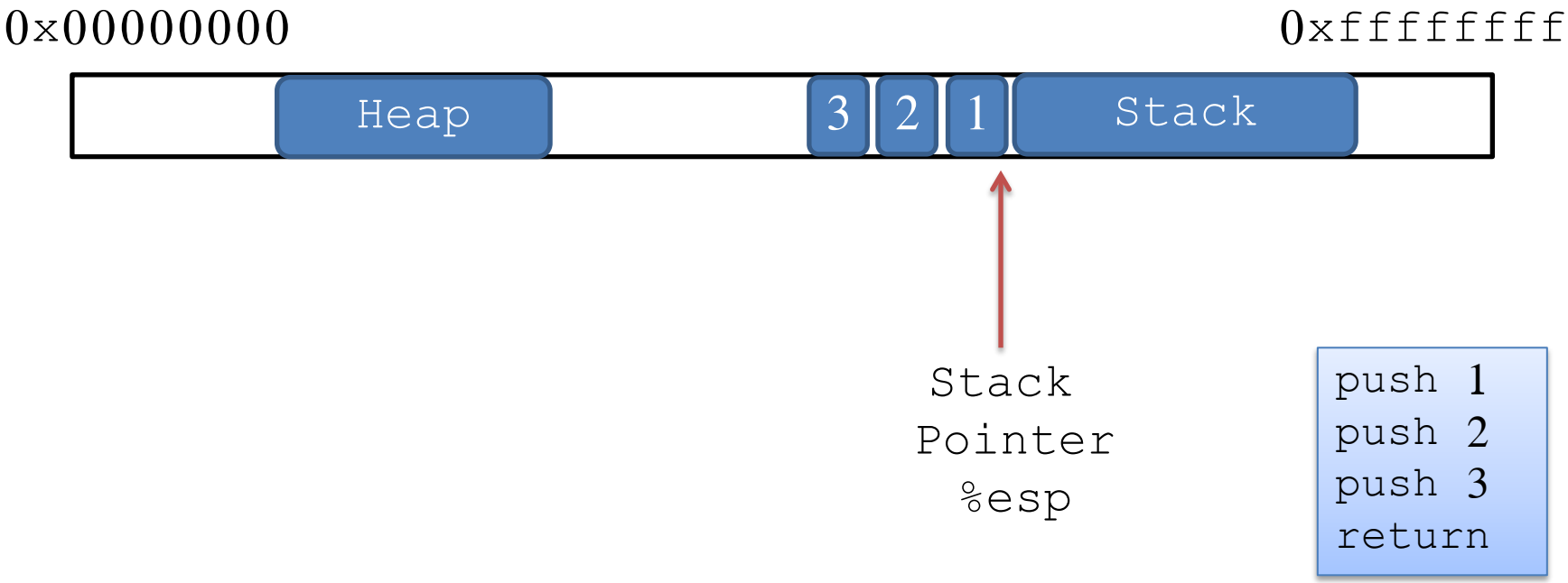
Stack and heap

- Stack and heap grow in opposite directions



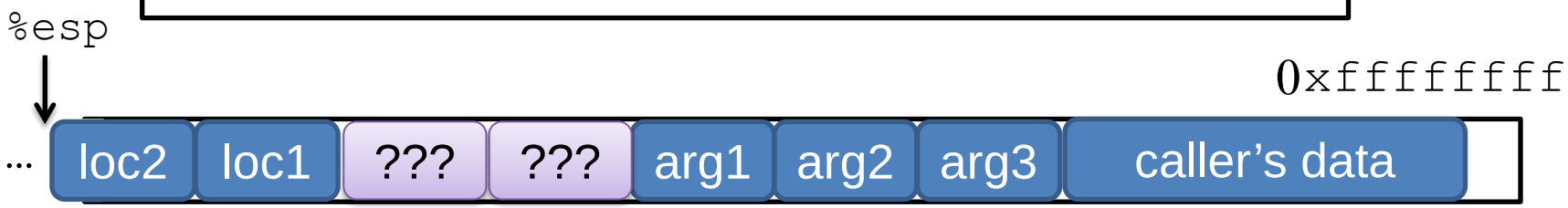
Apportioned by the OS;
managed in-process
by **malloc**

Stack and heap



Basic stack layout

```
void func(char *arg1, int arg2, int arg3)
{
        char loc1[4];
        int loc2;
        ...
}
```



Local variables are pushed in the same order

Arguments are pushed in reverse order of code

Basic buffer overflow

- Buffer:
  - Contiguous memory associated with a variable or field
  - Common in C
    - All strings are (NULL-terminated) arrays of chars
- Overflow:
  - Put more into the buffer that it can hold
- How?:
  - Bugs. E.g. Use of strcpy function

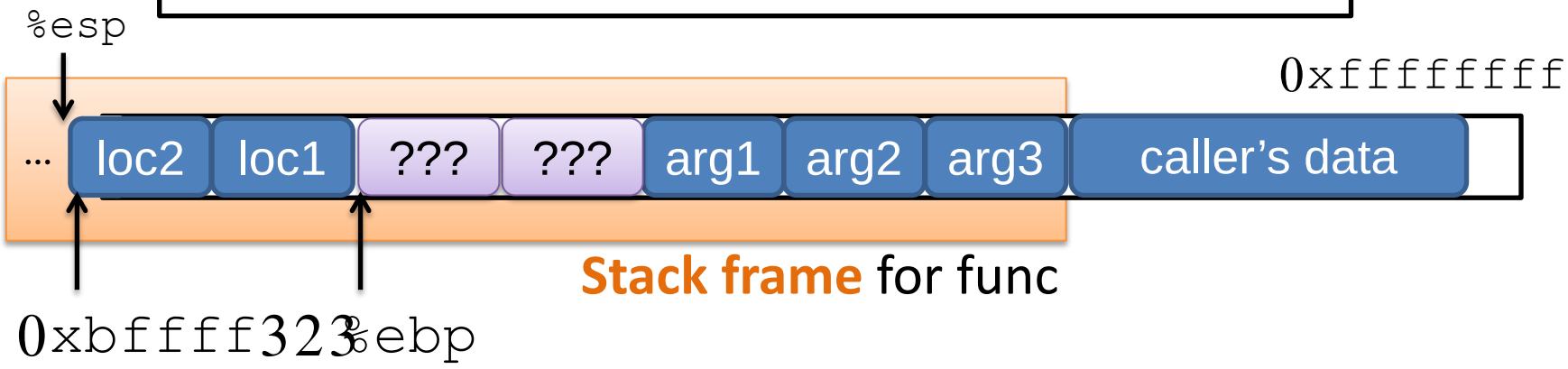
```
char *strcpy (char *dest, char *src)
```

- Let's go for an example: overflow_example

Accessing variables

```
void func(char *arg1, int arg2, int arg3)
{
        ...
        loc2++;
        ...
}
```

Where is loc2?



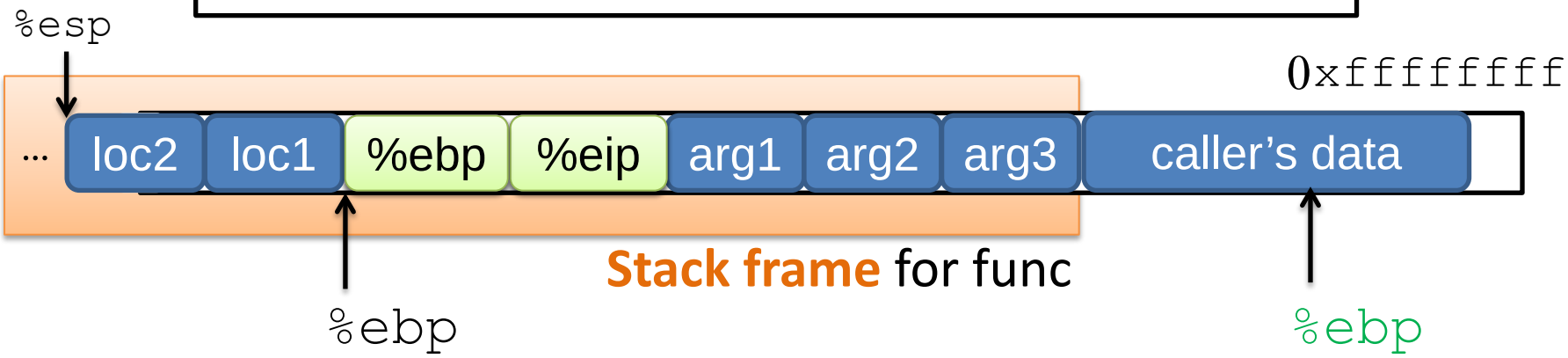
Can't guess absolute address at compile time

loc2 is at -8(%ebp)

But can know the **relative** address **loc2** is always 8B before ???s

Returning from functions

```
Int main()
{
        ...
        func ("Hello", 10, -3);
        ...
}
```



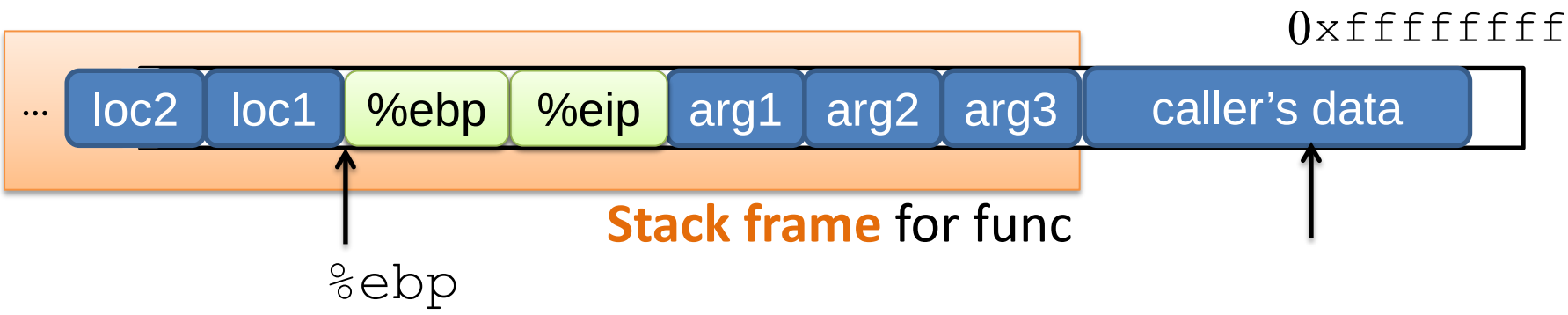
How do we restore %esp and %ebp?

1. Update `%esp`
2. Push `%ebp` before `locals`
   Set `%ebp` to current (`%ebp`)

How do we resume?

Push `%eip` before `call`
`Set %eip=4(%ebp)`

Let’s have fun with this



Defenses against buffer overflow

- Very quickly:
  - Stack canaries (StackGuard)
  - Non executable stack (NX)
  - Address Space Layout Randomization (ASLR)
- It is still possible to attack

Reto Buffer Overflow

- Programa a analizar: reto.c
- Instrucciones y pistas
- Pruebas offline
- Pruebas online
  - Instrucciones por email a inscritos en el reto
  - Periodo de estudio 1 semana
  - Periodo de ataque: 2 días
  - Aplicación en puertos TCP 5000 a 5015
    - Se restablece cada minuto en caso de crash

Thanks for your attention

- Thanks to:
  - Michael Hicks for its nice examples about overflow