From C to Assembly

(And back...)

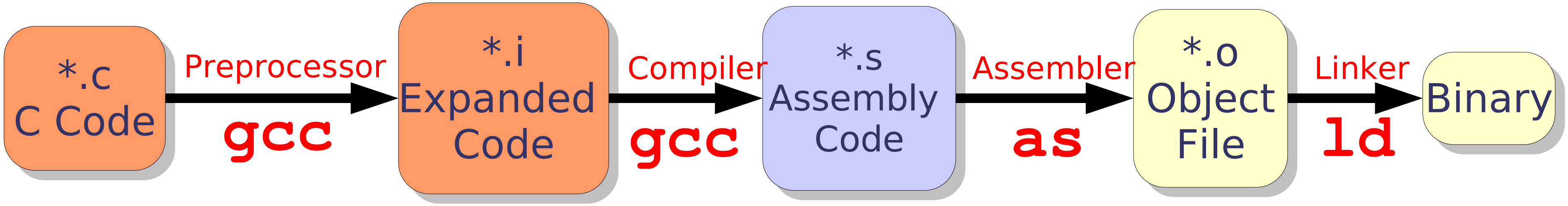
Emmanuel Fleury
LaBRI, Office 261
<emmanuel.fleury@labri.fr>

Outline

- From C to Assembly (gcc)
- From bin to Assembly (objdump)
- Variable Location in Memory
- Function Calls
- Doug Lea malloc
- ptmalloc
- C++ Heap Discipline

From C to Assembly (gcc)

Compilation Process



- Preprocessing (to expand macros)
- Compilation (from source code to assembly language)
- Assembly (from assembly language to machine code)
- Linking (to create the final executable)

Preprocessing

```
#include <stdio.h>

#define MESSAGE “Hello, world!\n”

int main () {
  printf (MESSAGE);
  return 0;
}
```

```
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command line>"
# 1 "hello.c"

... [snip] ...

extern void funlockfile(FILE *__stream);
# 831 "/usr/include/stdio.h" 3 4

# 2 "hello.c" 2

int main () {
  printf (“Hello, world!\n”);
  return 0;
}
```

Command: `gcc -E hello.c > hello.i`

Compilation

```
#include <stdio.h>

#define MESSAGE “Hello, world!\n”

int main () {
  printf (MESSAGE);
  return 0;
}
```

```
        .file   "hello.c"
        .section        .rodata
.LC0:
        .string "Hello world!\n"
        .text
.globl main
        .type   main, @function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        andl    $-16, %esp
        movl    $0, %eax
        subl    %eax, %esp
        movl    $.LC0, (%esp)
        call    printf
        movl    $0, %eax
        leave
        ret

        .size   main, .-main
        .section .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.3.5
```

Command: `gcc -S hello.c`

Assembly

```
        .file   "hello.c"
        .section        .rodata
.LC0:
        .string "Hello world!\n"
        .text
.globl main
        .type   main, @function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        andl    $-16, %esp
        movl    $0, %eax
        subl    %eax, %esp
        movl    $.LC0, (%esp)
        call    printf
        movl    $0, %eax
        leave
        ret
        .size   main, .-main
        .section
.note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.3.5
(Debian 1:3.3.5-1)"
```

Command: **`gcc -o hello hello.s`**

```
00000000: 7f45 4c46 0101 0100 0000 0000 0000 0000  .ELF............
00000010: 0100 0300 0100 0000 0000 0000 0000 0000  ................
00000020: dc00 0000 0000 0000 3400 0000 0000 2800  ........4.....(.
00000030: 0b00 0800 5589 e583 ec08 83e4 f0b8 0000  ....U...........
00000040: 0000 29c4 c704 2400 0000 00e8 fcff ffff  ..)...$.........
00000050: b800 0000 00c9 c300 4865 6c6c 6f20 776f  ........Hello wo
00000060: 726c 6421 0a00 0047 4343 3a20 2847 4e55  rld!...GCC: (GNU
00000070: 2920 332e 332e 3520 2844 6562 6961 6e20  ) 3.3.5 (Debian 
00000080: 313a 332e 332e 352d 3129 0000 2e73 796d  1:3.3.5-1)...sym
00000090: 7461 6200 2e73 7472 7461 6200 2e73 6873  tab..strtab..shs
000000a0: 7472 7461 6200 2e72 656c 2e74 6578 7400  trtab..rel.text.
000000b0: 2e64 6174 6100 2e62 7373 002e 726f 6461  .data..bss..roda
000000c0: 7461 002e 6e6f 7465 2e47 4e55 2d73 7461  ta..note.GNU-sta
000000d0: 636b 002e 636f 6d6d 656e 7400 0000 0000  ck..comment.....
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 1f00 0000 0100 0000 0600 0000  ................
00000110: 0000 0000 3400 0000 2300 0000 0000 0000  ....4...#.......
00000120: 0000 0000 0400 0000 0000 0000 1b00 0000  ................
00000130: 0900 0000 0000 0000 0000 0000 4c03 0000  ............L...
00000140: 1000 0000 0900 0000 0100 0000 0400 0000  ................
00000150: 0800 0000 2500 0000 0100 0000 0300 0000  ....%...........
00000160: 0000 0000 5800 0000 0000 0000 0000 0000  ....X...........
00000170: 0000 0000 0400 0000 0000 0000 2b00 0000  ............+...
00000180: 0800 0000 0300 0000 0000 0000 5800 0000  ............X...
00000190: 0000 0000 0000 0000 0000 0000 0400 0000  ................
000001a0: 0000 0000 3000 0000 0100 0000 0200 0000  ....0...........
000001b0: 0000 0000 5800 0000 0e00 0000 0000 0000  ....X...........
000001c0: 0000 0000 0100 0000 0000 0000 3800 0000  ............8...
000001d0: 0100 0000 0000 0000 0000 0000 6600 0000  ............f...
000001e0: 0000 0000 0000 0000 0000 0000 0100 0000  ................
000001f0: 0000 0000 4800 0000 0100 0000 0000 0000  ....H...........
00000200: 0000 0000 6600 0000 2500 0000 0000 0000  ....f...%.......
00000210: 0000 0000 0100 0000 0000 0000 1100 0000  ................
00000220: 0300 0000 0000 0000 0000 0000 8b00 0000  ................
00000230: 5100 0000 0000 0000 0000 0000 0100 0000  Q...............
00000240: 0000 0000 0100 0000 0200 0000 0000 0000  ................
00000250: 0000 0000 9402 0000 a000 0000 0a00 0000  ................
00000260: 0800 0000 0400 0000 1000 0000 0900 0000  ................
00000270: 0300 0000 0000 0000 0000 0000 3403 0000  ............4...
00000280: 1500 0000 0000 0000 0000 0000 0100 0000  ................
00000290: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000002a0: 0000 0000 0100 0000 0000 0000 0000 0000  ................
000002b0: 0400 f1ff 0000 0000 0000 0000 0000 0000  ................
000002c0: 0300 0100 0000 0000 0000 0000 0000 0000  ................
000002d0: 0300 0300 0000 0000 0000 0000 0000 0000  ................
000002e0: 0300 0400 0000 0000 0000 0000 0000 0000  ................
000002f0: 0300 0500 0000 0000 0000 0000 0000 0000  ................
00000300: 0300 0600 0000 0000 0000 0000 0000 0000  ................
00000310: 0300 0700 0900 0000 0000 0000 2300 0000  ............#...
00000320: 1200 0100 0e00 0000 0000 0000 0000 0000  ................
00000330: 1000 0000 0068 656c 6c6f 2e63 006d 6169  .....hello.c.mai
00000340: 6e00 7072 696e 7466 0000 0000 1300 0000  n.printf........
00000350: 0105 0000 1800 0000 0209 0000            ............
```

Linking

Command:

```
ld -o hello -dynamic-linker
/lib/ld-linux.so.2 /usr/lib/crt1.o
/usr/lib/crti.o
/usr/lib/gcc-lib/i686/3.3.1/crtbegin.o
-L/usr/lib/gcc-lib/i686/3.3.1 hello.o
-lgcc -lgcc_eh -lc -lgcc -lgcc_eh
/usr/lib/gcc-lib/i686/3.3.1/crtend.o
/usr/lib/crtn.o
```

Alternate Command: `gcc -o hello hello.o`

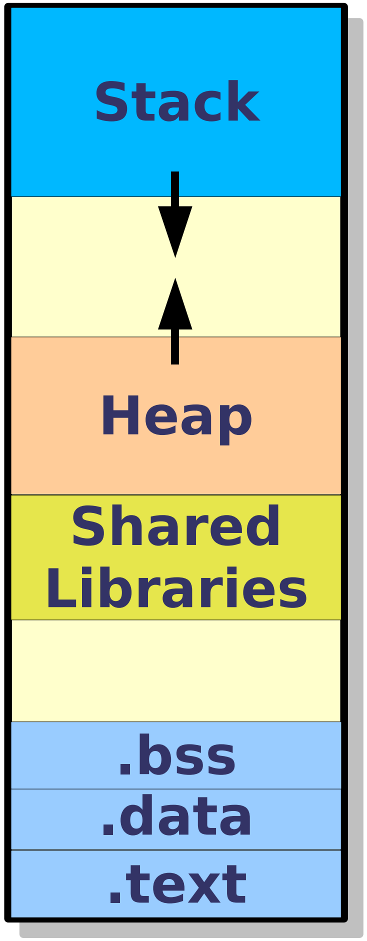
From bin to Assembly (objdump)

Main Options

-a, --archive-headers	Display archive header information
-f, --file-headers	Display the contents of the overall file header
-p, --private-headers	Display object format specific file header contents
-h, --[section-]headers	Display the contents of the section headers
-x, --all-headers	Display the contents of all headers
-d, --disassemble	Display assembler contents of executable sections
-D, --disassemble-all	Display assembler contents of all sections
-S, --source	Intermix source code with disassembly
-s, --full-contents	Display the full contents of all sections requested
-g, --debugging	Display debug information in object file
-e, --debugging-tags	Display debug information using ctags style
-G, --stabs	Display (in raw form) any STABS info in the file
-W, --dwarf	Display DWARF info in the file
-t, --syms	Display the contents of the symbol table(s)
-T, --dynamic-syms	Display the contents of the dynamic symbol table
-r, --reloc	Display the relocation entries in the file
-R, --dynamic-reloc	Display the dynamic relocation entries in the file
-i, --info	List object formats and architectures supported

Variable Location in Memory

Variables Location in Memory



```
/* Global variables */
int var1;                /* .bss */
char var2[]=”buff”;      /* .data */

main()
{
   /* Local/Automatic variables */
   int var3;                      /* stack */
   static int var4;               /* .bss  */
   static char var5[]=”buff”;     /* .data */
   char *var6;                    /* stack */
   var6 = malloc(512);            /* heap  */
}
```

`static`: Preserves variable value to survive after its scope ends. static function or data element are only known within the scope of the current compile. If `static` keyword is used with a variable that is local to a function, it allows the last value of the variable to be preserved between successive calls to that function.

Automatic/Static/Global Variables

Automatic Variables:

- Initialized at runtime
- Stored in the stack

Static Variables:

- Initialized at compile time
- Stored in .data or .bss

Global Variables:

- Initialized at compile time
- Stored in .data or .bss

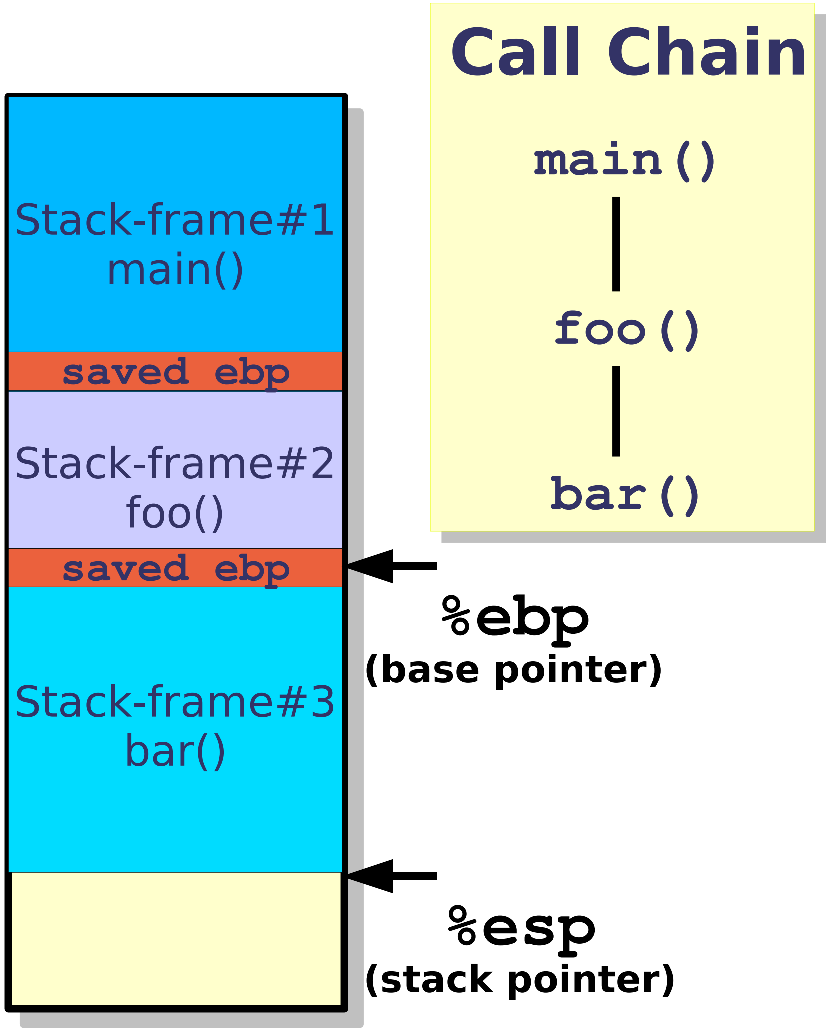
Where are stored “const” data ?

Function Calls

Functions Basics

- If subroutine **`foo()`** calls function **`bar()`**:
  - **`foo()`** is the **caller**;
  - **`bar()`** is the **callee**.
- A variable used only inside the scope of the function is called a **local variable**;
- Data set by the caller for the callee before start are called **arguments** (or **parameters**);
- Data set by the callee for the caller at the end of execution of the callee are called **return code**.

Stack Frames



Call Chain

main()

foo()

bar()

Stack-frame#1 main()

saved ebp

Stack-frame#2 foo()

saved ebp

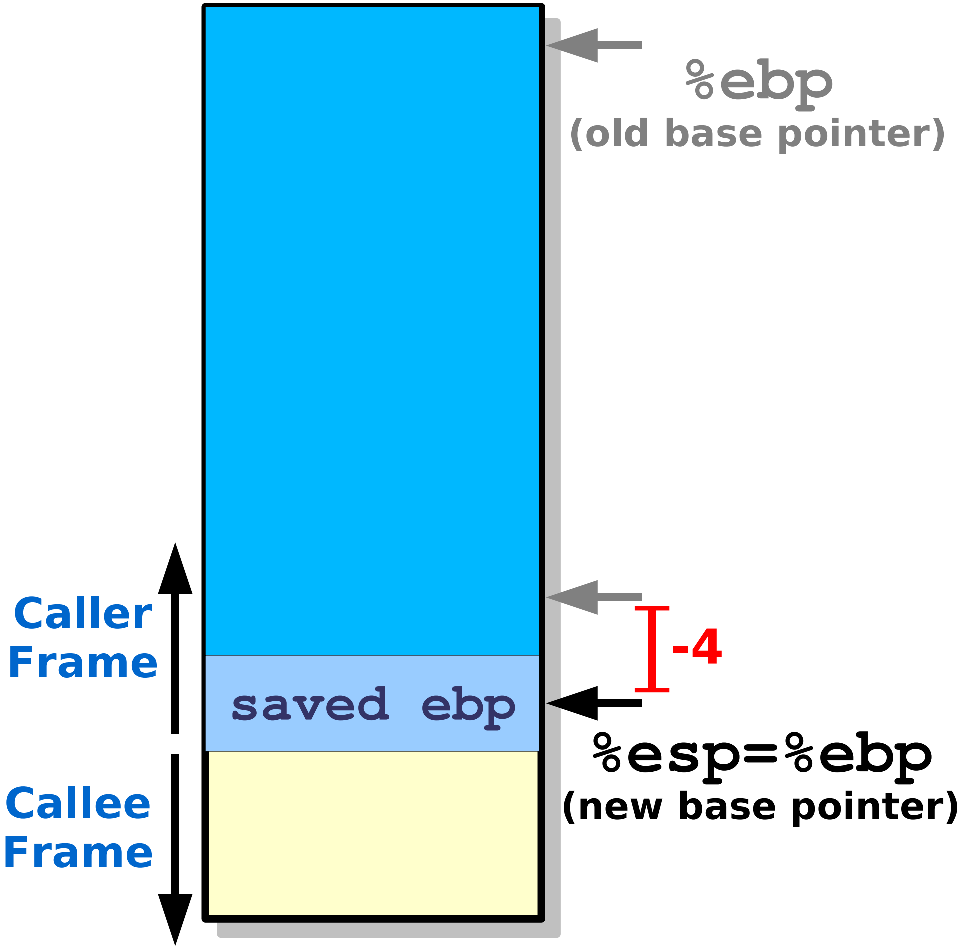
Stack-frame#3 bar()

%ebp (base pointer)

%esp (stack pointer)

- Contents
  - Local variables
  - Return information
  - Temporary space
- Management
  - Created on enter
  - Restored on leave
- Pointers
  - %esp: Stack pointer (stack top)
  - %ebp: Frame pointer (frame start)

enter
(create new frame)



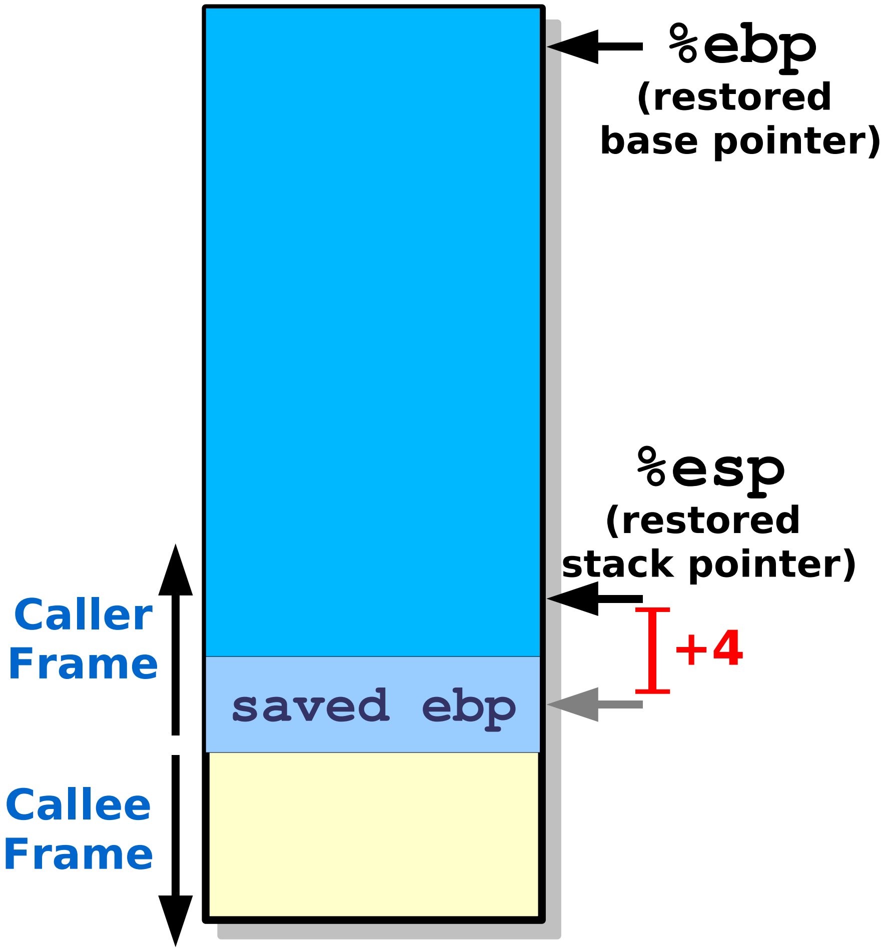
Create a new stack-frame

enter:

1. `pushl %ebp`
2. `movl %esp,%ebp`
3. Get back to execution

leave

(restore old frame)

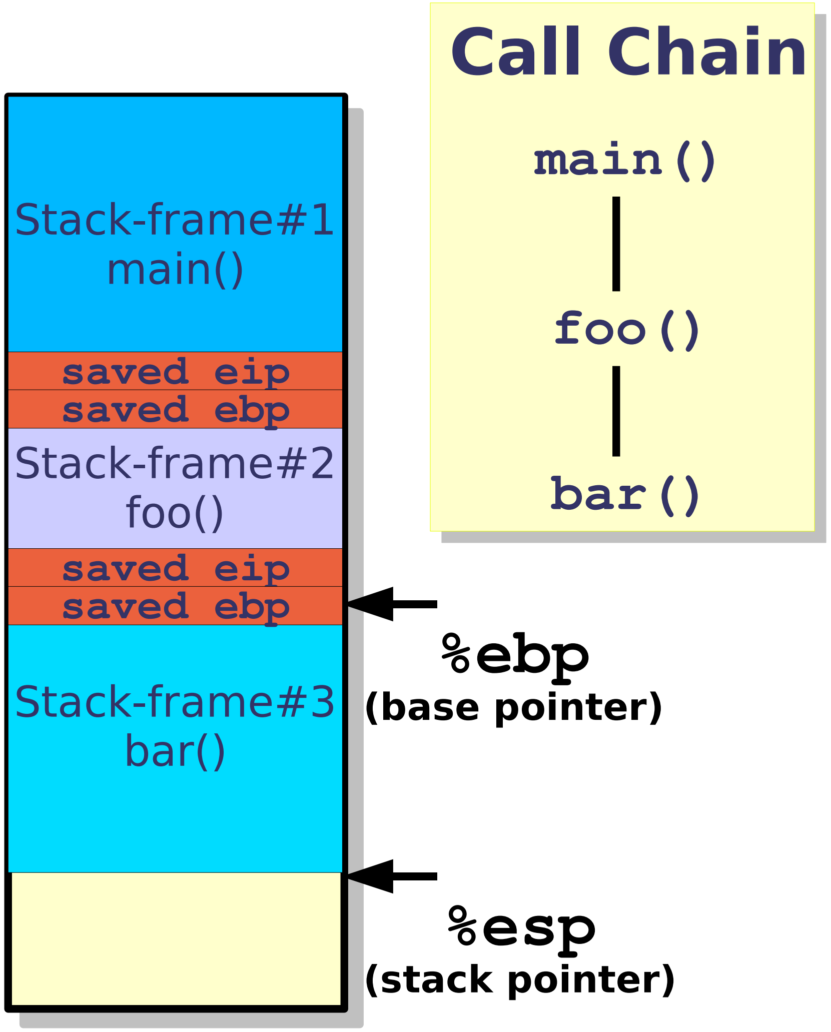


Restore the old stack-frame

`leave`:

1. `movl %ebp,%esp`
2. `popl %ebp`
3. Get back to execution

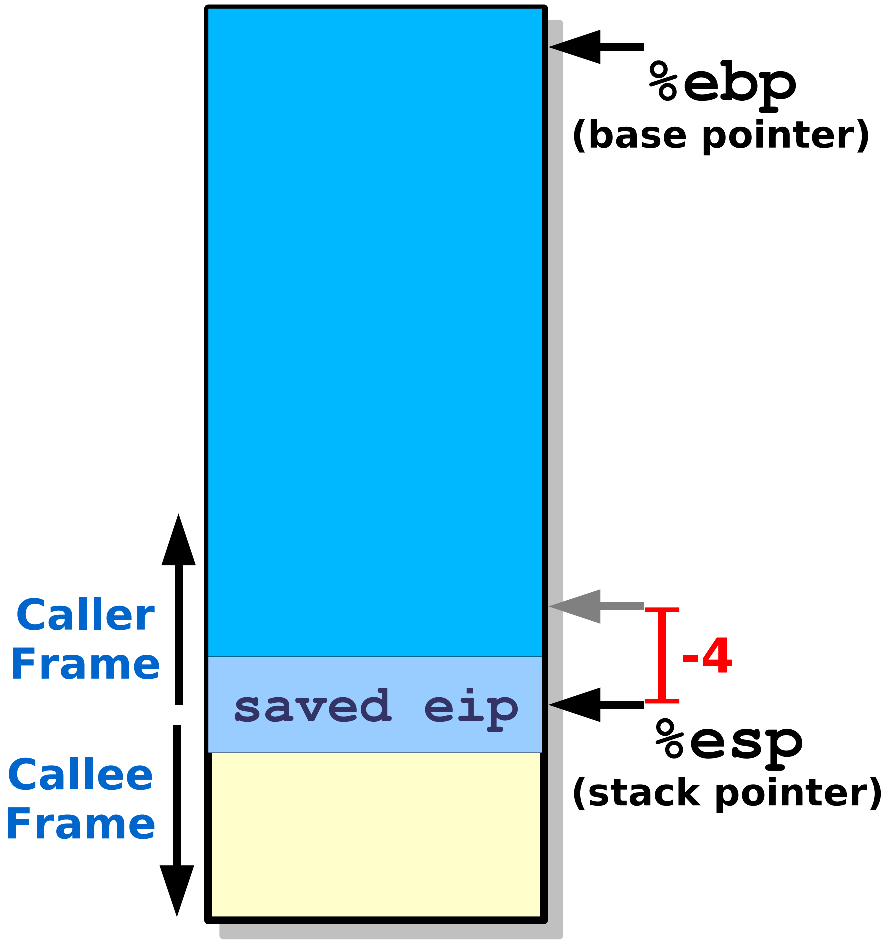
Execution Flow Management



- Contents
  - Store old instruction pointer on stack
  - Load new instruction pointer
- Management
  - Created on call
  - Restored on ret
- Pointers
  - %eip: Instruction pointer

call

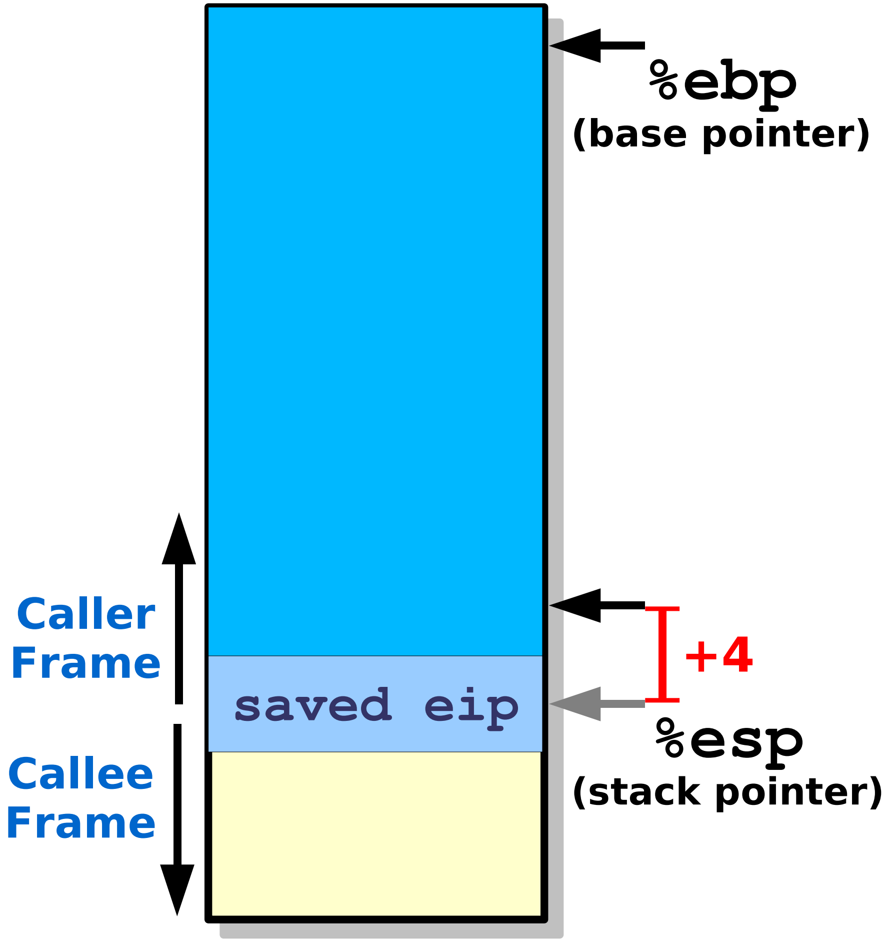
(start new execution)



call addr:

1. `pushl %eip`
2. `movl addr,%eip`
3. Get back to execution
   (needs to create a new stack-frame)

ret
(restart old execution)

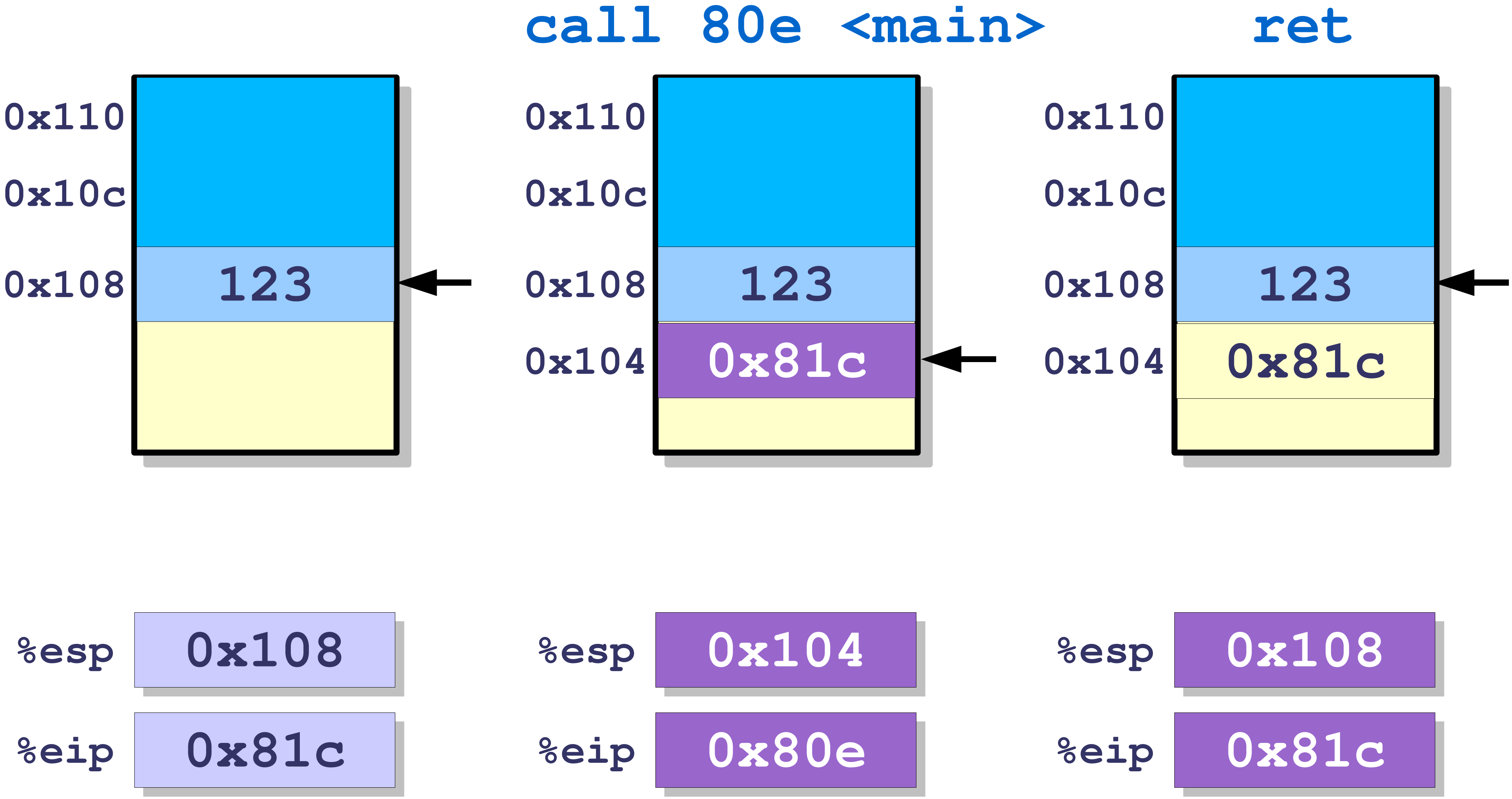


ret:

1. **popl %eip**
2. Get back to execution

Note that the stack-frame must have been restored **before** calling **ret**.

call/ret Example



call 80e <main>

ret

0x110

0x10c

0x108 123

0x110

0x10c

0x108 123

0x104 0x81c

0x110

0x10c

0x108 123

0x104 0x81c

%esp 0x108

%eip 0x81c

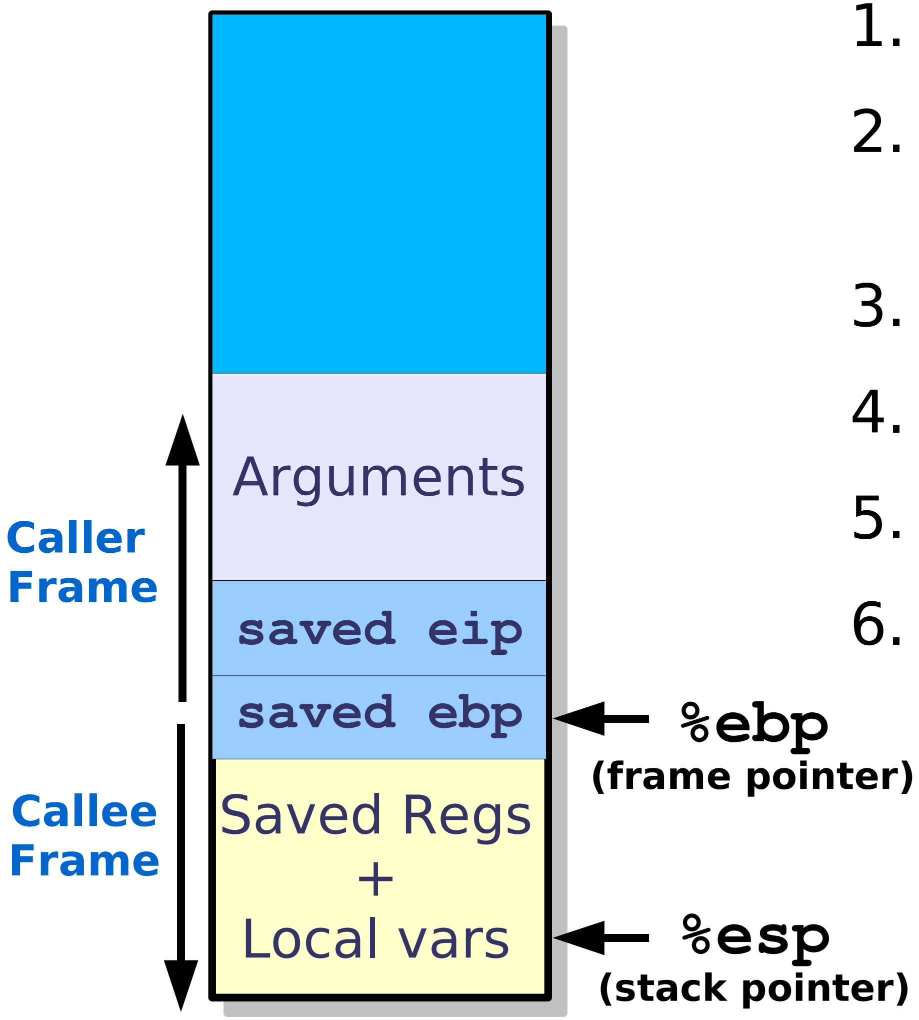
%esp 0x104

%eip 0x80e

%esp 0x108

%eip 0x81c

Passing Arguments



1. Push arguments on the stack;
2. Save `%eip` and jump to subroutine (`call`);
3. Create a new frame (`enter`);
4. Execution of the subroutine;
5. Restore old frame (`leave`);
6. Restore `%eip` and return (`ret`).

Return Code

Tell about struct returning...

```
#include <stdio.h>
#define LENGTH 10

typedef struct {
  char str[LENGTH];
  int value; } record_t;

record_t read_record() {
   record_t record;
   scanf("%s", record.str);
   return record;
}
int main() {
   record_t record;
   record = read_record();
   printf(record.str);
   return 0;
}
```

Return code is usually stored in the data registers **`%eax`**.

Calling a Subroutine

```
.globl main

main:
     movl    $12, %ebx
     pushl   %ebx
     call    sqr
     addl    $4, %esp       # Restore old esp
                            # before “push”
     ret

sqr:
     movl    4(%esp), %eax
     imull   %eax, %eax     # eax^2
     ret
```

Calling a C function (printf)

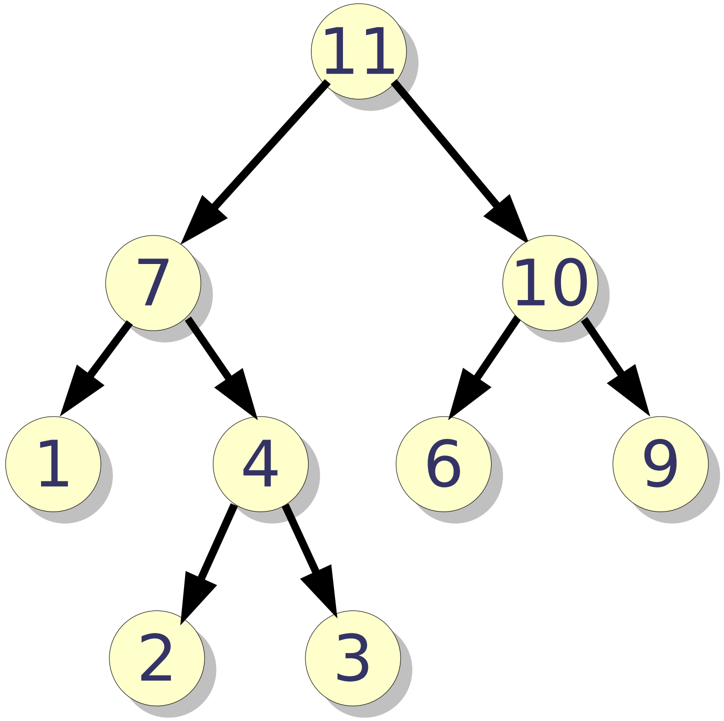
```
.globl main

main:
        movl    $12, %ebx
        pushl   %ebx
        call    sqr
        addl    $4, %esp        # Restore old esp
                                # before "push"
        ret

sqr:
        movl    4(%esp), %eax
        imull   %eax, %eax      # eax²
        ret
```

Doug Lea malloc

What is a (binary) Heap ?

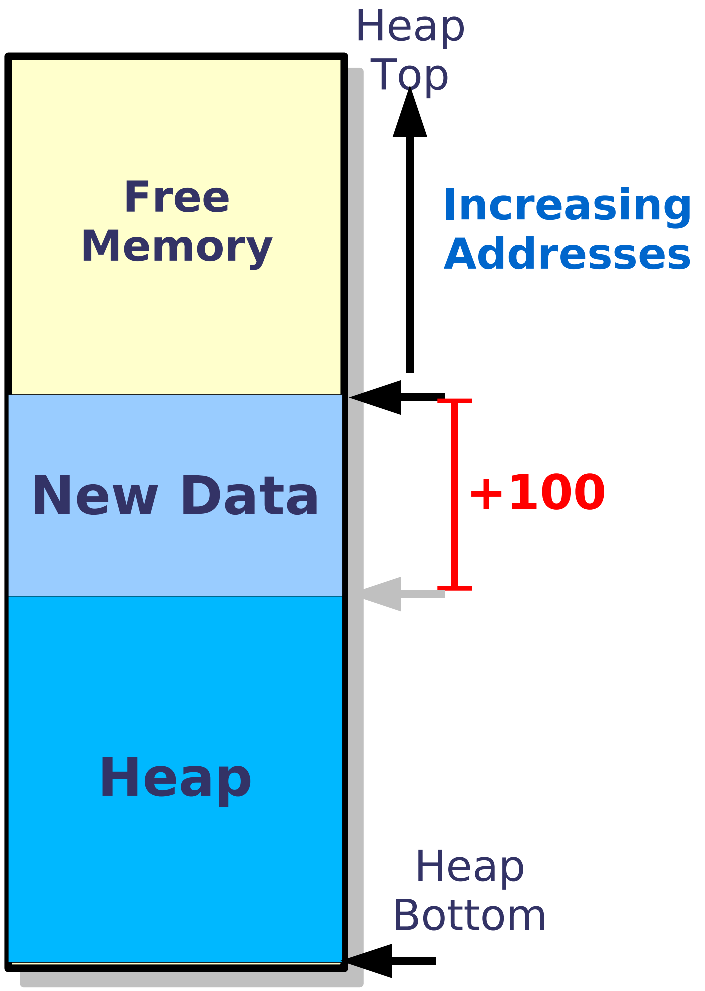


A heap is a tree structure such that, if A and B are nodes of a heap and B is a child of A, then:

key(A)≥key(B)

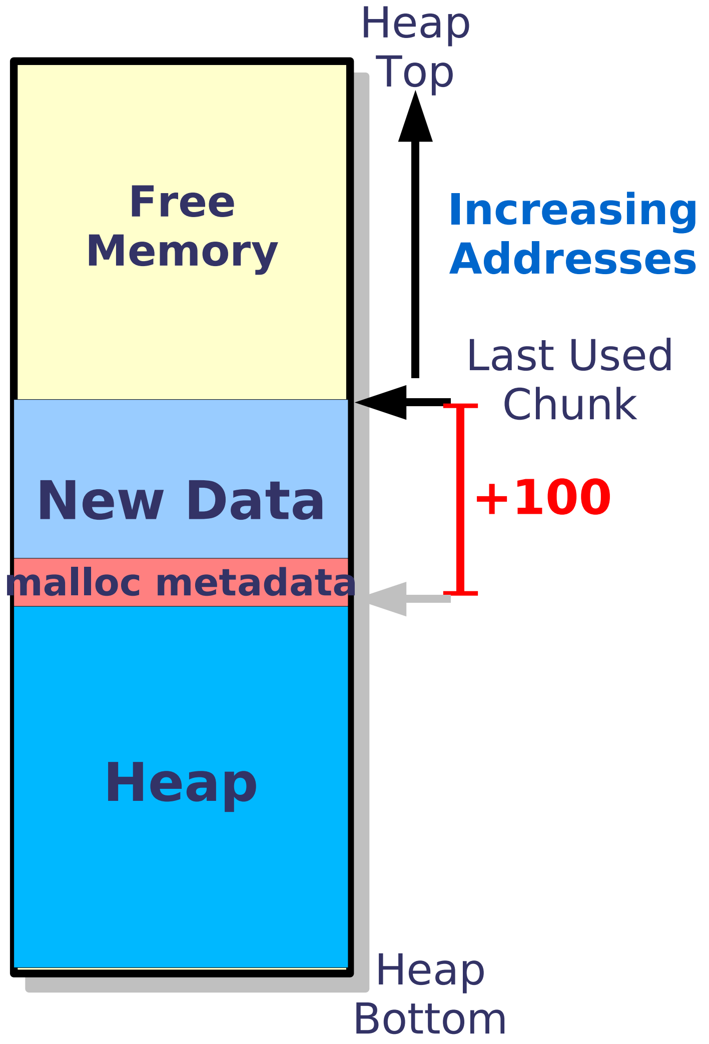
- Implemented via:
  - Arrays
    (a[i] has two children a[2i+1],a[2i+2])
  - Trees
- Applications:
  Quick access to data
  (databases)
- Groups of Data:
  In Doug Lea malloc (dlmalloc) memory chunks are classified by size (bytes).

IA-32 Heap



- Memory zone managed with "heap *discipline*"
- Grows toward higher addresses
- From programmer point of view:
  Managed through a language dependent interface (C, C++,...).
- From the system point of view:
  Managed through specific system calls
  (**`mmap()`**, **`brk()`**).

IA-32 Heap (Doug Lea malloc)

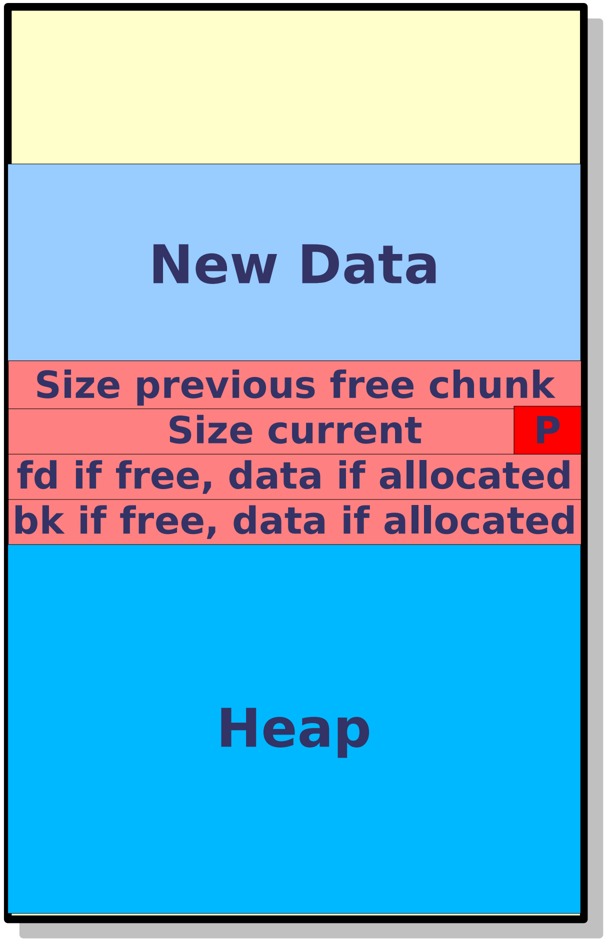


The `dlmalloc` library:

- **`void *malloc(size)`**:
  Allocate a memory chunk of size **`size`**.
- **`void *calloc(nb,size)`**:
  Allocate an array of **`nb`** cells where each cell has size **`size`**.
- **`void *realloc(*ptr,size)`**:
  Change the size of the memory block pointed by **`ptr`** to **`size`**.
- **`void free(*ptr)`**:
  Free the chunk pointed by **`ptr`**.

`malloc()` is a memory allocator on the Heap.
When **`malloc()`** lack of space, it requires more and enlarge the Heap with **`mmap()`**.

IA-32 Heap
(dlmalloc metadata)



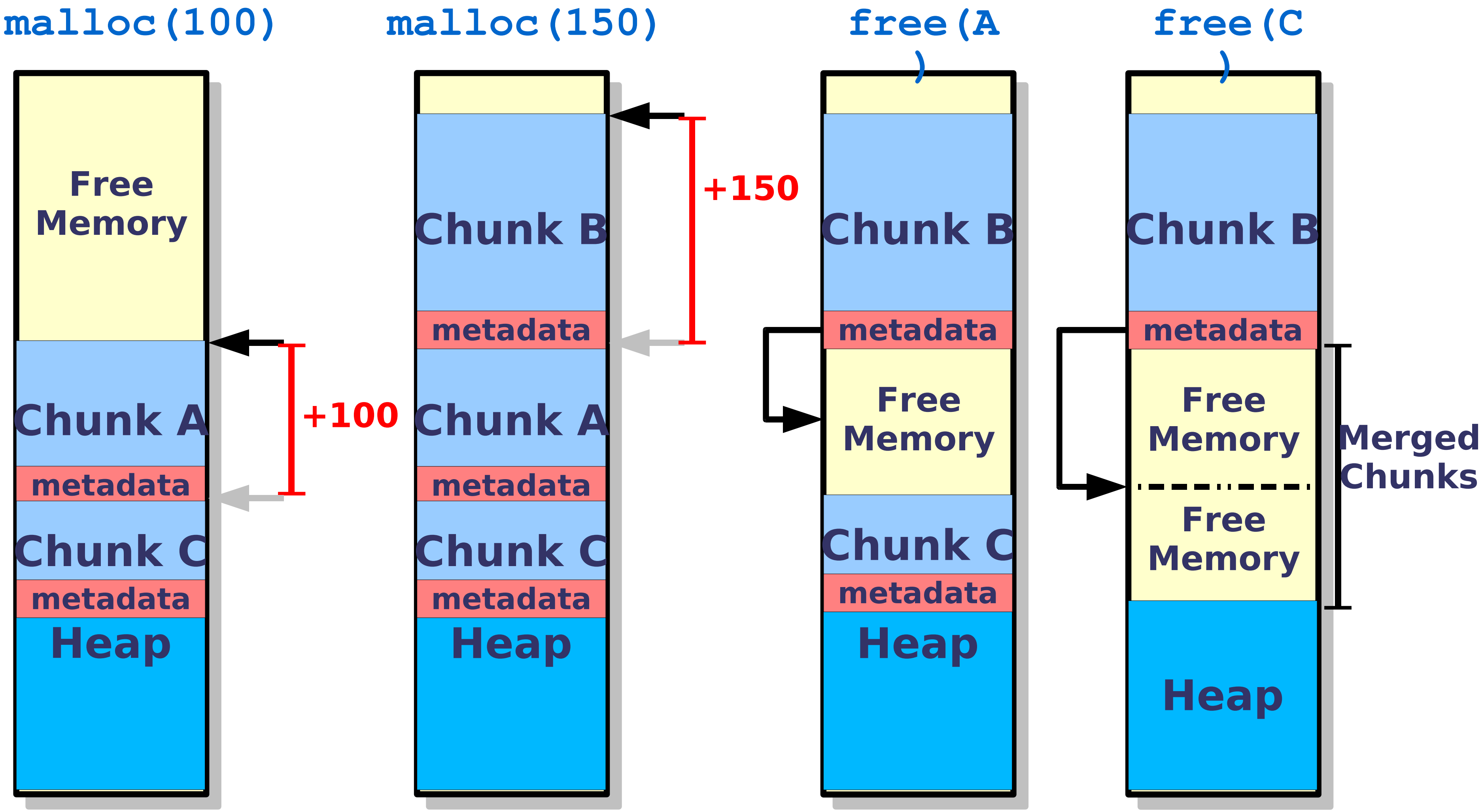
Metadata (information about):

- The size of the previous free chunk (if there is one);
- The size of the current chunk (including metadata);
- A pointer to the next free chunk;
- A pointer to the previous free chunk;
- P (**`PREV_INUSE`** bit) indicates if the previous chunk is free or not.

```
struct malloc_chunk {
  size_t prev_size; /* Size previous chunk */
  size_t size;/*Size current including metadata*/

  /* Double links - used only if free */
  struct malloc_chunk fd; /* Forward chunk  */
  struct malloc_chunk bk; /* Backward chunk */
};
```

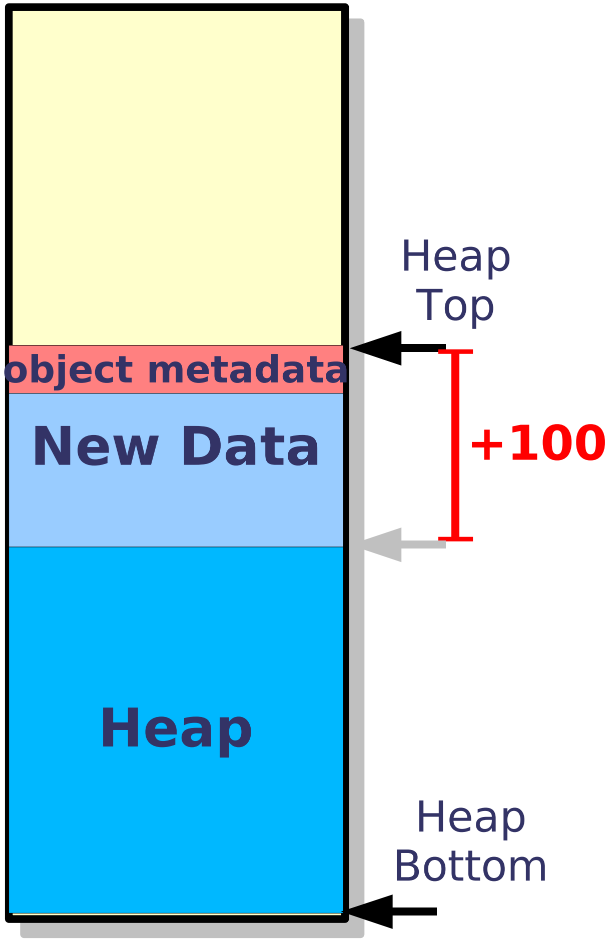
IA-32 Heap (fragmentation avoidance)



ptmalloc

C++ Heap Discipline

IA-32 Heap (C++)



The C++ memory library:

- new
- free
- ...

Next Time

Hacking Tools