Process Layout, Function Calls, and the Heap

CS 161 – Spring 2011

Prof. Vern Paxson

TAs: Devdatta Akhawe, Mobin Javed, Matthias Vallentin

January 19, 2011

ZDNet
News & Blogs | Reviews | Downloads | White Papers
Log In | Join | Site Assistance
US Edition
Companies | Hardware | Software | Mobile | Security | Research | Special Coverage

Zero Day

Ryan Naraine and Daniel Kennedy

Mobile
RSS
Email Alerts

7 Comments | Share | Print | Facebook | Twitter | Recommend | 0 Votes

Ad Info

Sponsored Links

Network Security
Compliance - Network
Scanning - Free Trial D
www.eEye.com/Network-S

Security Mgmt. I
Build Your Career in th
with a 100% Online De
www.AMU.APUS.edu/Secur

SQL Injection sc
Check for SQL injectio
Download Free Acuneti
www.acunetix.com/free-ed

Home / News & Blogs / Zero Day

Google pays $14,000 for high-risk Chrome security holes

By Ryan Naraine | January 14, 2011, 9:52am PST

Summary

Google has shelled out more than $14,000 in rewards for critical and high-risk vulnerabilities affecting its flagship Chrome web browser.

Topics

Google Inc., Team, Adobe PDF, CERT,

Google has shelled out more than $14,000 in rewards for critical and high-risk vulnerabilities affecting its flagship Chrome web browser.

The latest Google Chrome 8.0.552.237, available for all platforms, patches a total of 16 documented vulnerabilties, including one critical bug for which Google paid the first elite $3133.7 award to researcher Sergey Glazunov.

"Critical bugs are harder to come by in Chrome, but Sergey has done it," says Google's Jerome Kersey. "Sergey also collects a $1337 reward and several other rewards at the same time, so congratulations Sergey!," he added.

The best of ZDNet, delive

Become a ZDNet member today. C and get a FREE ebook ($69 value)

ZDNet Newsletters

Get the best of ZDNet delivere inbox

Your Email

ZDNet's White Paper

Outline

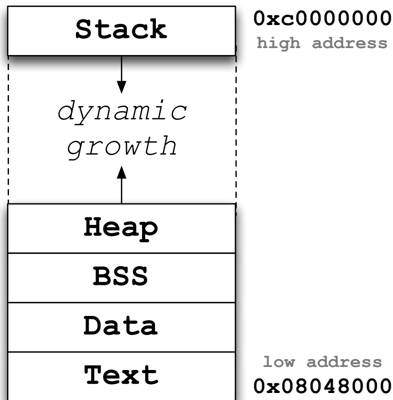
Process Layout

Function Calls

The Heap

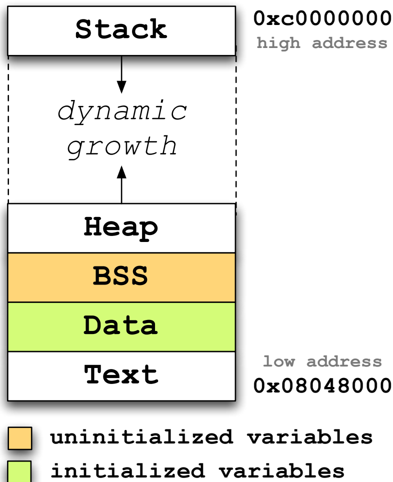
Process Layout in Memory

- **Stack**
  - grows towards *decreasing* addresses.
  - is initialized at *run-time*.
- **Heap** and **BSS** sections
  - grow towards *increasing* addresses.
  - are initialized at *run-time*.
- **Data** section
  - is initialized at *compile-time*.
- **Text** section
  - holds the program instructions (read-only).



Process Layout in Memory

- **Stack**
  - grows towards *decreasing* addresses.
  - is initialized at *run-time*.
- **Heap** and **BSS** sections
  - grow towards *increasing* addresses.
  - are initialized at *run-time*.
- **Data** section
  - is initialized at *compile-time*.
- **Text** section
  - holds the program instructions (read-only).



Outline

Process Layout

Function Calls

The Heap

IA-32

Registers

- **EAX** Accumulator for operands and results data
- **EBX** Pointer to data in the DS segment
- **ECX** Counter for string and loop operations
- **EDX** I/O pointer
- **ESI** Source pointer for string operations
- **EDI** Destination pointer for string operations
- **EBP** Frame pointer
- **ESP** Stack pointer

Terminology

- **SFP** **saved frame pointer**: saved `%ebp` on the stack
- **OFP** **old frame pointer**: old `%ebp` from the previous stack frame
- **RIP** **return instruction pointer**: return address on the stack

IA-32

Registers

- **EAX** Accumulator for operands and results data
- **EBX** Pointer to data in the DS segment
- **ECX** Counter for string and loop operations
- **EDX** I/O pointer
- **ESI** Source pointer for string operations
- **EDI** Destination pointer for string operations
- **EBP** Frame pointer
- **ESP** Stack pointer

Terminology

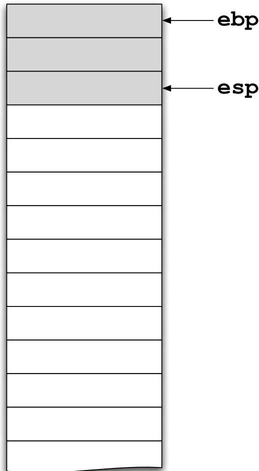
- **SFP** **saved frame pointer**: saved `%ebp` on the stack
- **OFP** **old frame pointer**: old `%ebp` from the previous stack frame
- **RIP** **return instruction pointer**: return address on the stack

Function Calls

```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];

    bar[0] = 'A';
    qux[0] = 0x2a;
}

int main(void)
{
    int i = 1;
    foo(1, 2, 3);

    return 0;
}
```

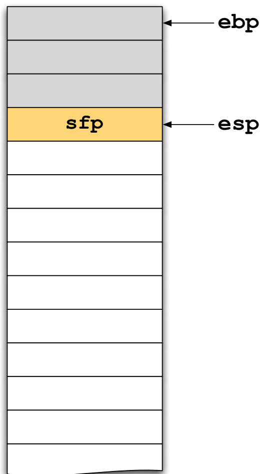
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



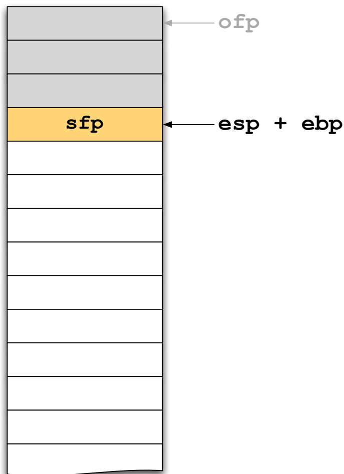
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



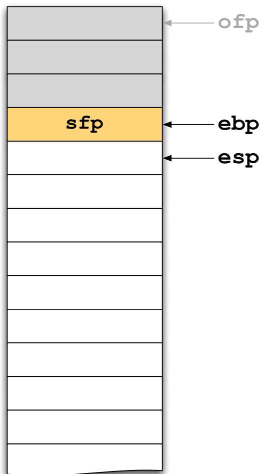
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



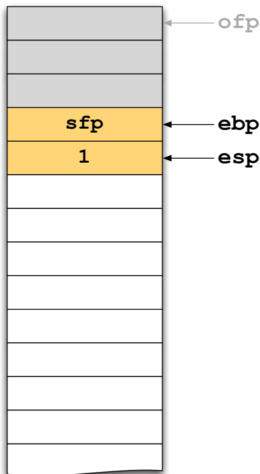
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



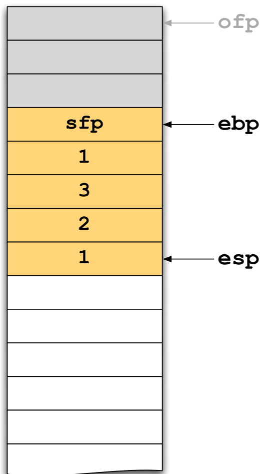
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



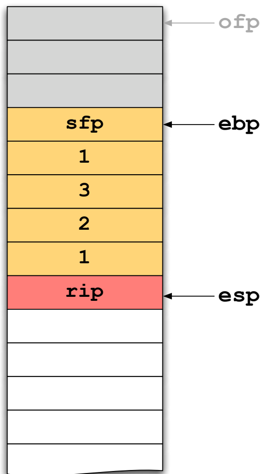
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



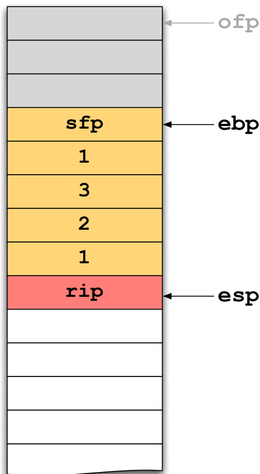
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



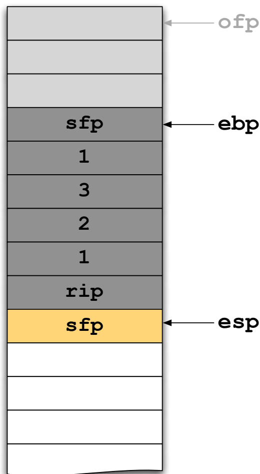
Function Calls in Assembler

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



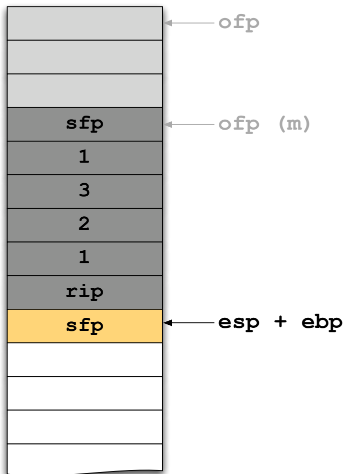
Function Calls in Assembler

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



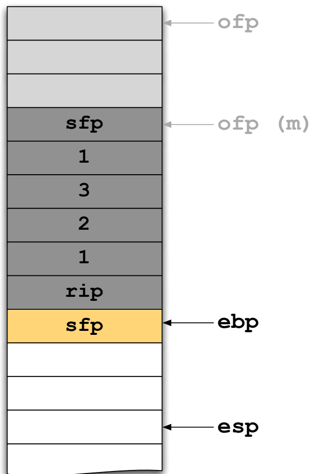
Function Calls in Assembler

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



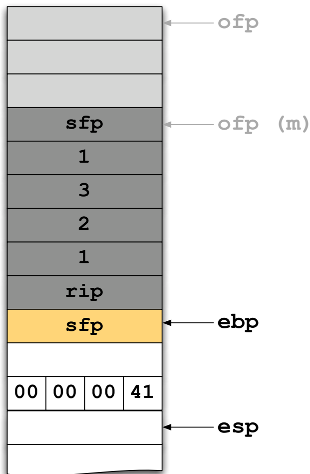
Function Calls in Assembler

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



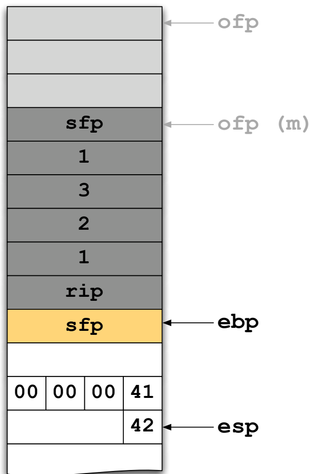
Function Calls in Assembler

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



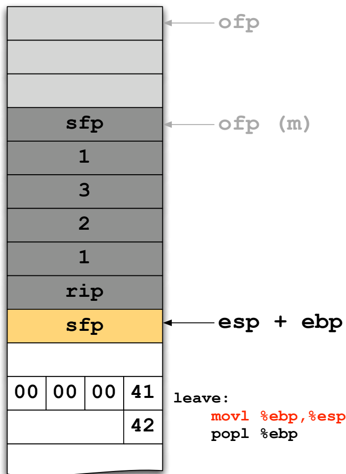
Function Calls in Assembler

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



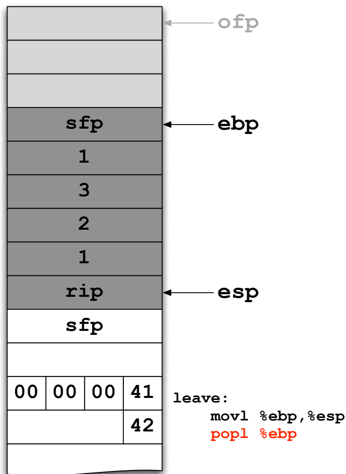
Function Calls in Assembler

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



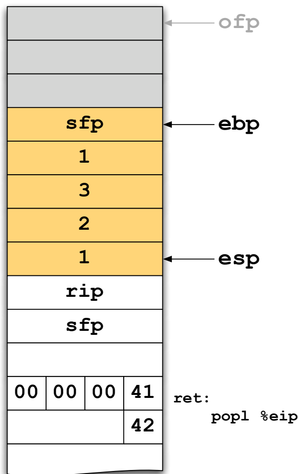
Function Calls in Assembler

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



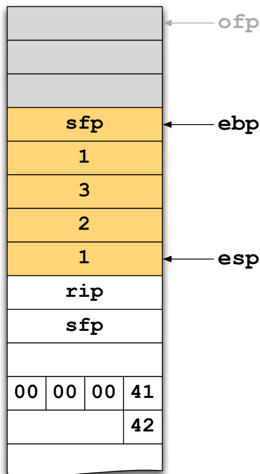
Function Calls in Assembler

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



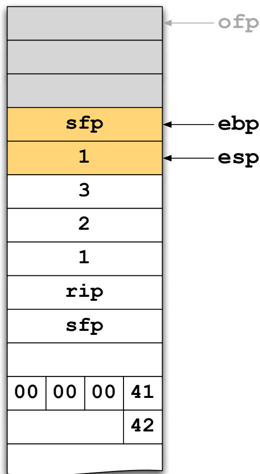
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



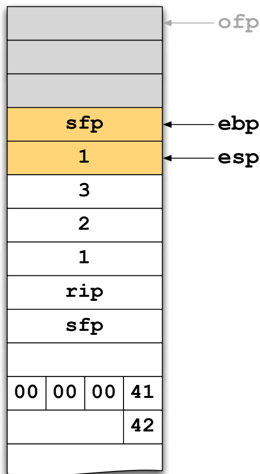
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



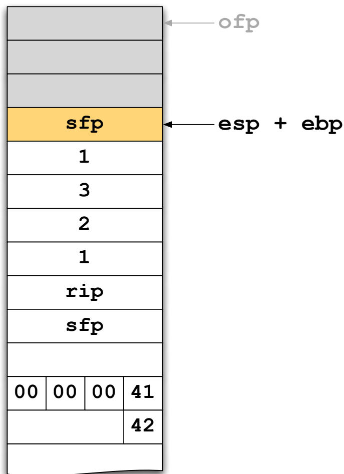
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



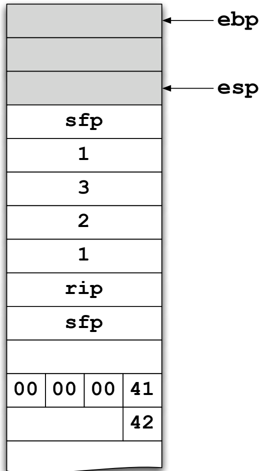
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



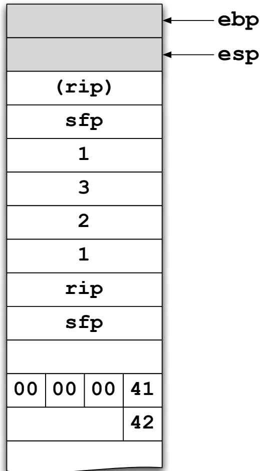
Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Outline

Process Layout

Function Calls

The Heap

The Heap

The **heap** is *"[...] a pool of memory available for the allocation and deallocation of arbitrary-sized blocks of memory in arbitrary order."* [WJN+95]

The Heap

The **heap** is *"[...] a pool of memory available for the allocation and deallocation of arbitrary-sized blocks of memory in arbitrary order."* [WJN+95]

- ANSI-C functions `malloc()` and friends are used to manage the heap (`glibc` uses `ptmalloc`).

The Heap

The **heap** is *"[...] a pool of memory available for the allocation and deallocation of arbitrary-sized blocks of memory in arbitrary order."* [WJN+95]

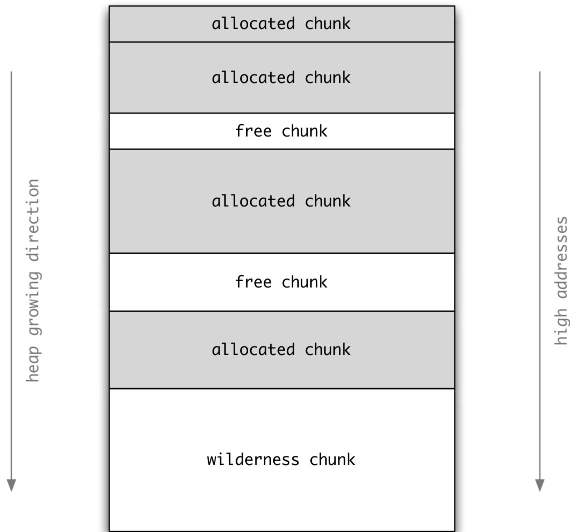
- ANSI-C functions `malloc()` and friends are used to manage the heap (`glibc` uses `ptmalloc`).
- Heap memory is organized in **chunks** that can be allocated, freed, merged, etc.

The Heap

The **heap** is *"[...] a pool of memory available for the allocation and deallocation of arbitrary-sized blocks of memory in arbitrary order."* [WJN+95]

- ANSI-C functions `malloc()` and friends are used to manage the heap (`glibc` uses `ptmalloc`).
- Heap memory is organized in **chunks** that can be allocated, freed, merged, etc.
- **Boundary Tags** contain meta information about chunks (size, previous/next pointer, etc.)
  - stored both in the front and end of each chunk.
  - $\rightarrow$ makes consolidating fragmented chunks into bigger chunks very fast.

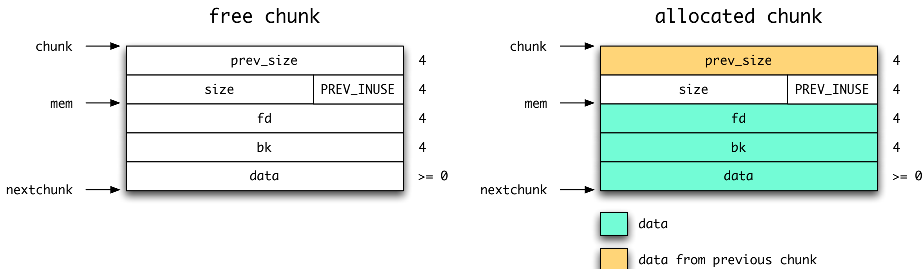
Chunks in Memory



Understanding Heap Management

Boundary Tags

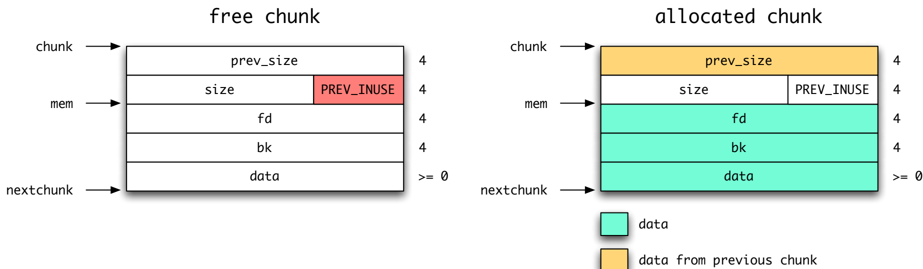
- **prev_size**: size of previous chunk (if free).
- **size**: size in bytes, including overhead.
- **PREV_INUSE**: Status bit; set if previous chunk is allocated.
- **fd/bk**: *forward/backward pointer* for double links (if free).



Understanding Heap Management

Boundary Tags

- **prev_size**: size of previous chunk (if free).
- **size**: size in bytes, including overhead.
- **PREV_INUSE**: Status bit; set if previous chunk is allocated.
- **fd/bk**: *forward/backward pointer* for double links (if free).



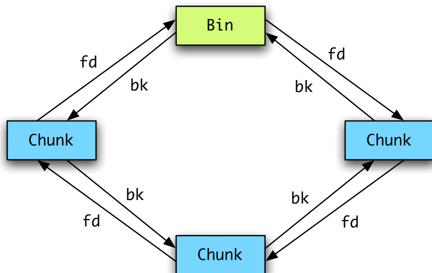
Understanding Heap Management

Boundary Tags

- **prev_size**: size of previous chunk (if free).
- **size**: size in bytes, including overhead.
- **PREV_INUSE**: Status bit; set if previous chunk is allocated.
- **fd/bk**: *forward/backward pointer* for double links (if free).

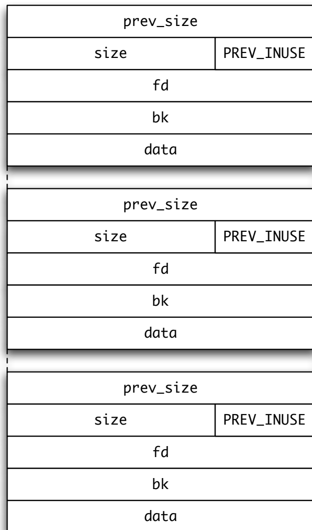
Managing Free Chunks

- Free chunks of similar size are grouped into **bins**.
- `fd/bk` pointers to navigate through double links.



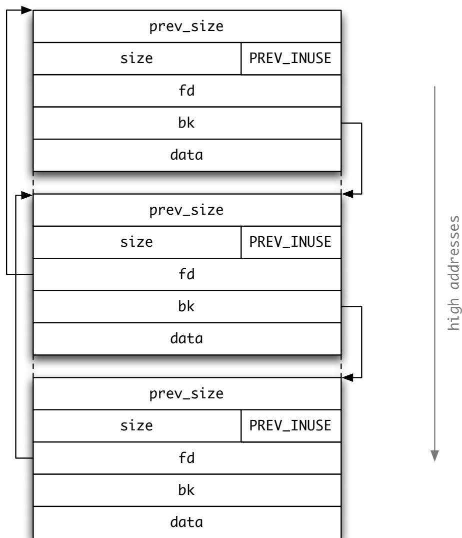
Removing Chunks from a Bin: `unlink()`

```
#define unlink(P, BK, FD)
{
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}
```



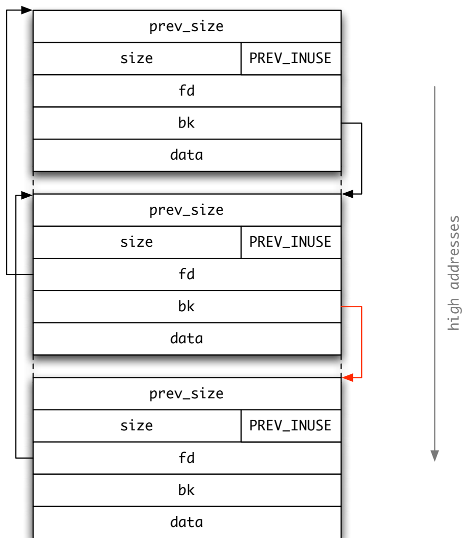
Removing Chunks from a Bin: unlink()

```
#define unlink(P, BK, FD)
{
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}
```



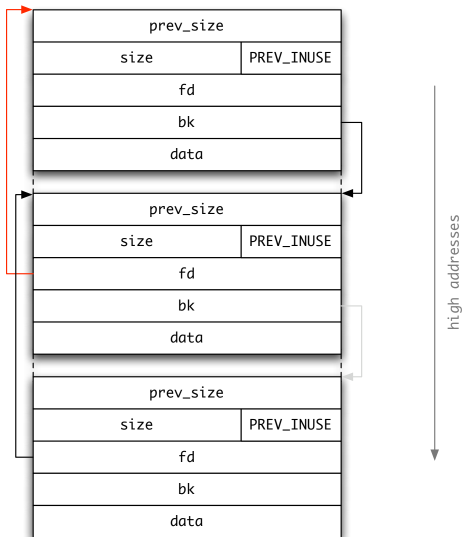
Removing Chunks from a Bin: unlink()

```
#define unlink(P, BK, FD)
{
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}
```



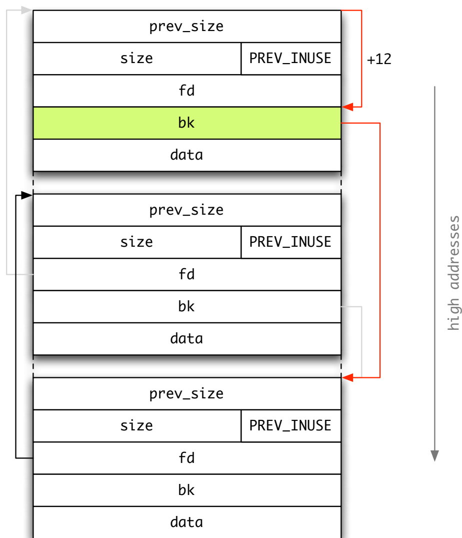
Removing Chunks from a Bin: unlink()

```
#define unlink(P, BK, FD)
{
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}
```



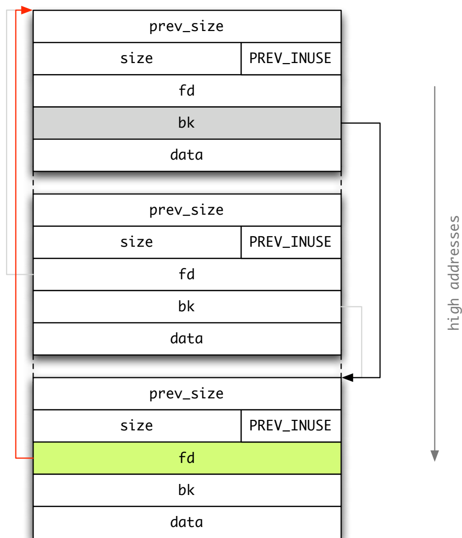
Removing Chunks from a Bin: `unlink()`

```
#define unlink(P, BK, FD)
{
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}

FD + 12 = BK
```



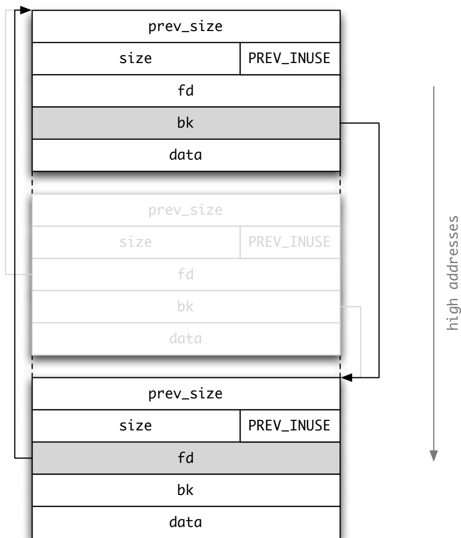
Removing Chunks from a Bin: `unlink()`

```
#define unlink(P, BK, FD)
{
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}
```



Removing Chunks from a Bin: `unlink()`

```
#define unlink(P, BK, FD)
{
    BK = P->bk;
    FD = P->fd;
    FD->bk = BK;
    BK->fd = FD;
}
```



References

Paul R. Wilson and Mark S. Johnstone and Michael Neely and David Boles.
Dynamic Storage Allocation: A Survey and Critical Review.
International Workshop on Memory Management, 1995.

IA-32 Reference

IA32 Instructions

`movl`	*Src,Dest*	*Dest* = *Src*
`addl`	*Src,Dest*	*Dest* = *Dest* + *Src*
`subl`	*Src,Dest*	*Dest* = *Dest* - *Src*
`imull`	*Src,Dest*	*Dest* = *Dest* * *Src*
`sall`	*Src,Dest*	*Dest* = *Dest* << *Src*
`sarl`	*Src,Dest*	*Dest* = *Dest* >> *Src*
`shrl`	*Src,Dest*	*Dest* = *Dest* >> *Src*
`xorl`	*Src,Dest*	*Dest* = *Dest* ^ *Src*
`andl`	*Src,Dest*	*Dest* = *Dest* & *Src*
`orl`	*Src,Dest*	*Dest* = *Dest* | *Src*
`incl`	*Dest*	*Dest* = *Dest* + 1
`decl`	*Dest*	*Dest* = *Dest* - 1
`negl`	*Dest*	*Dest* = - *Dest*
`notl`	*Dest*	*Dest* = ~ *Dest*
`leal`	*Src,Dest*	*Dest* = address of *Src*
`cmpl`	*Src2,Src1*	*Sets CCs Src1 – Src2*
`testl`	*Src2,Src1*	*Sets CCs Src1 & Src2*
`jmp`	*label*	*jump*
`je`	*label*	*jump equal*
`jne`	*label*	*jump not equal*
`js`	*label*	*jump negative*
`jns`	*label*	*jump non-negative*
`jg`	*label*	*jump greater (signed)*
`jge`	*label*	*jump greater or equal (signed)*
`jl`	*label*	*jump less (signed)*
`jle`	*label*	*jump less or equal (signed)*
`ja`	*label*	*jump above (unsigned)*
`jb`	*label*	*jump below (unsigned)*

Addressing Modes

Immediate $*val* *Val*

Normal (R) Mem[Reg[R]]

- Register R specifies memory address

```
movl (%ecx),%eax
```

Displacement D(R) Mem[Reg[R]+D]

- Register R specifies start of memory region
- Constant displacement D specifies offset

```
movl 8(%ebp),%edx
```

Indexed D(Rb,Ri,S) Mem[Reg[Rb]+S*Reg[Ri]+ D]

- D: Constant “displacement” 1, 2, or 4 bytes
- Rb: Base register: Any of 8 integer registers
- Ri: Index register:
- S: Scale: 1, 2, 4, or 8

Condition Codes

CF	Carry Flag
ZF	Zero Flag
SF	Sign Flag
OF	Overflow Flag

Registers: `%eax`, `%edx`, `%ecx`, `%ebx`, `%esi`, `%edi`, `%esp`, `%ebp`