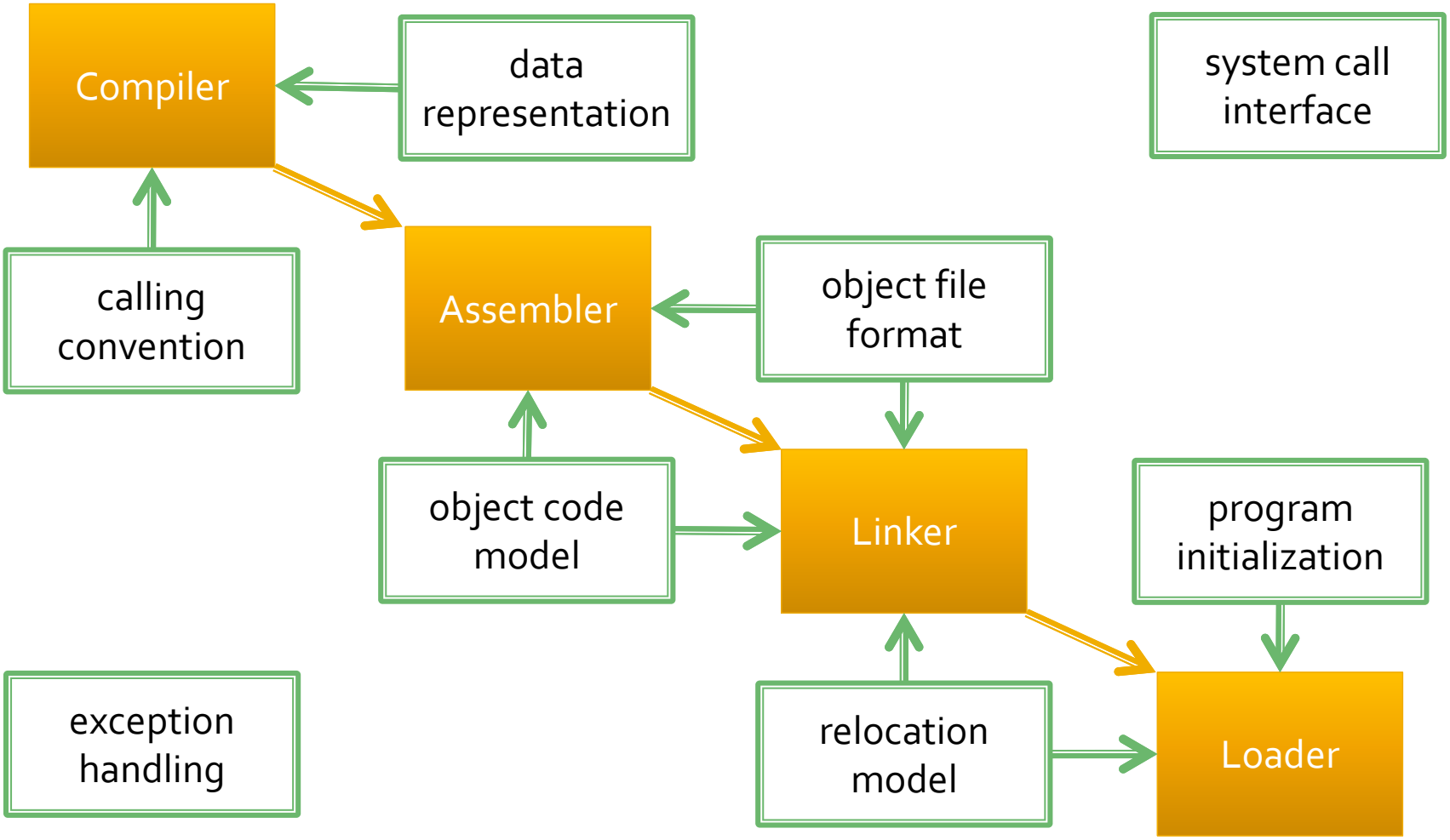
CPEG421/621 Tutorial

# Memory Maps, ABIs

# Application Binary Interface



# Intel x86 Architecture

| ia32 | ia64 |
| --- | --- |
| 8 x 32bit General Purpose Registers | 16 x 64bit General Purpose Registers |
| 6 x 16bit Segment Registers | 6 x 16bit Segment Registers |
| 8 x 80bit Floating-Point Registers | 8 x 80bit Floating-Point Registers |
| 8 x 64bit MMX Registers | 8 x 64bit MMX Registers |
| 8 x 128bit XMM Registers | 16 x 128bit XMM Registers |
| 32bit address space | 64bit address space |

# Compiler

- Data representation
  Specifies the size and alignment of the different data types
- Calling Convention
  Specifies how to pass arguments to a function
- Register Convention
  Specifies who has to save and restore the registers

# Data Representation (Linux)

| | 32 bit | | 64 bit | |
|---|---|---|---|---|
| Type | Size | Alignment | Size | Alignment |
| *_Bool* | 1 | 1 | 1 | 1 |
| char | 1 | 1 | 1 | 1 |
| Short | 2 | 2 | 2 | 2 |
| int | 4 | 4 | 4 | 4 |
| enum | 4 | 4 | 4 | 4 |
| long | 4 | 4 | 8 | 8 |
| *long long* | 8 | 4 | 8 | 8 |
| pointer | 4 | 4 | 8 | 8 |
| float | 4 | 4 | 4 | 4 |
| double | 8 | 4 | 8 | 8 |
| *long double* | 12 | 4 | 16 | 16 |

# Structures, Unions and Bitfields

```
struct {
    char c;
};
```


Size: 1 byte
Alignment: 1 byte

# Structures, Unions and Bitfields

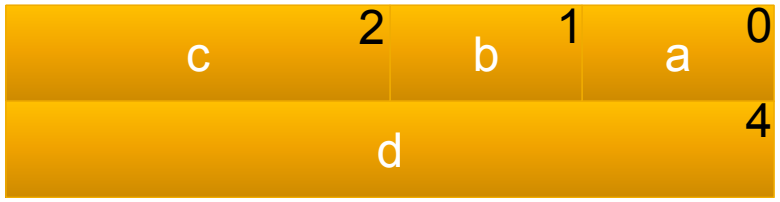
```
struct {
    char a;
    short b;
};
```


Size: 4 byte
Alignment: 2 byte

# Structures, Unions and Bitfields

```
struct {
    char a;
    char b;
    short c;
    long d;
};
```

ia32 ABI

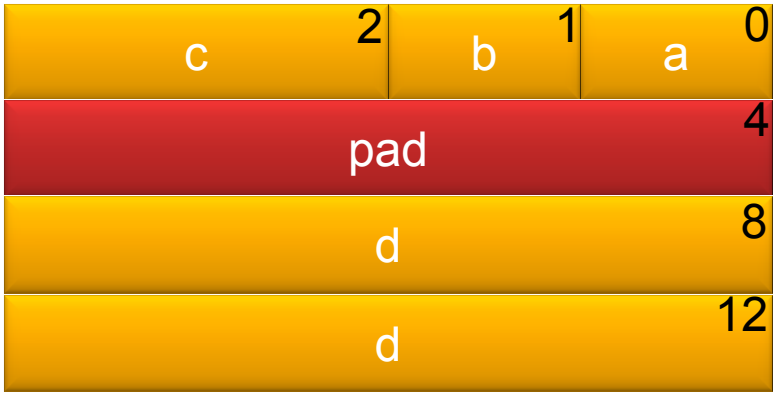



Size: 8 byte
Alignment: 4 byte

# Structures, Unions and Bitfields

```
struct {
    char a;
    char b;
    short c;
    long d;
};
```

ia64 ABI

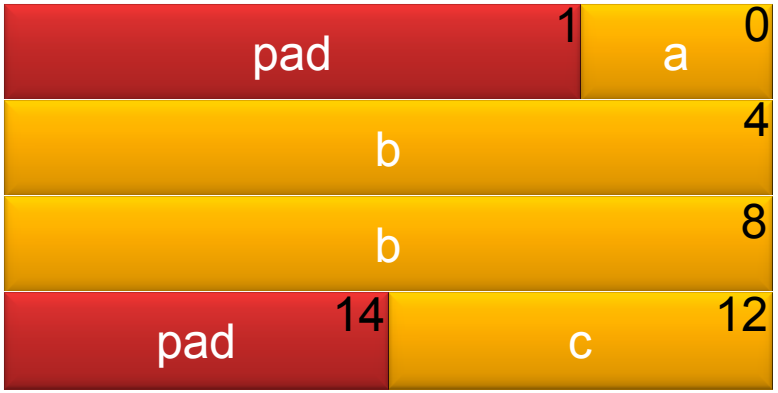



Size: 16 byte
Alignment: 8 byte

# Structures, Unions and Bitfields

```
struct {
    char a;
    double
b;
    short c;
};
```

ia32 ABI

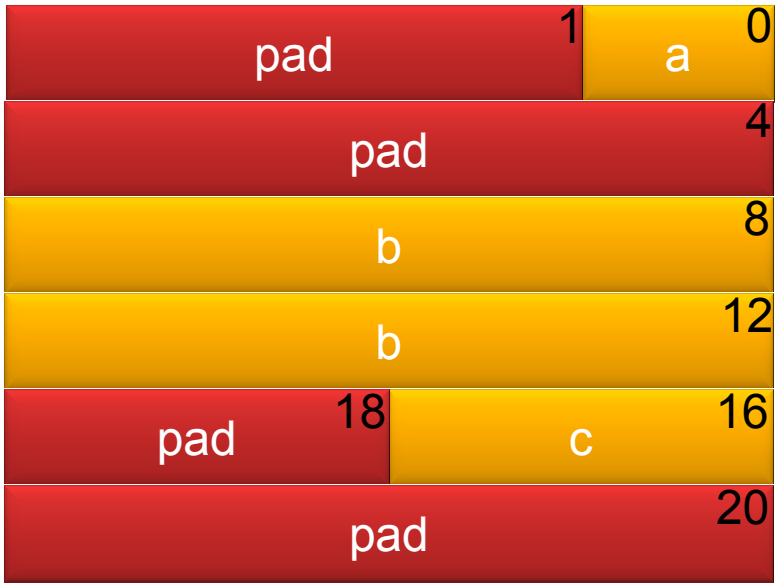



Size: 16 byte
Alignment: 4 byte

# Structures, Unions and Bitfields

```
struct {
    char a;
    double b;
    short c;
};
```

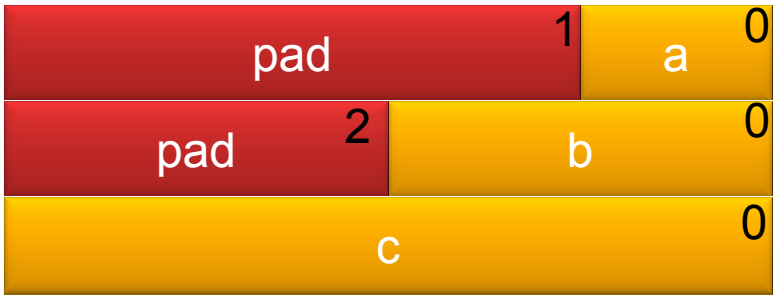
ia64 ABI




Size: 24 byte
Alignment: 8 byte

# Structures, Unions and Bitfields

```
union {
    char a;
    short b;
    int c;
};
```




Size: 4 byte
Alignment: 4 byte

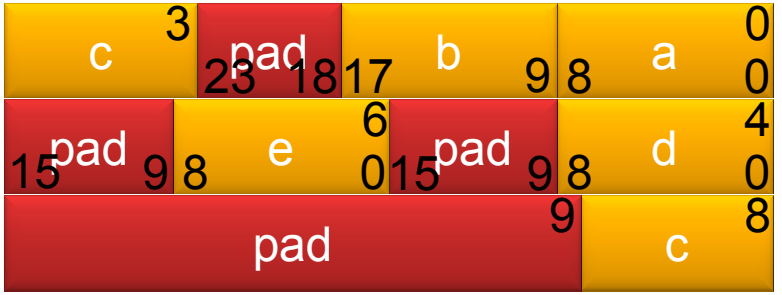
# Structures, Unions and Bitfields

```
struct {
    short
a:5;
    int b:6;
    int c:7;
};
```


Size: 4 byte
Alignment: 4 byte

# Structures, Unions and Bitfields

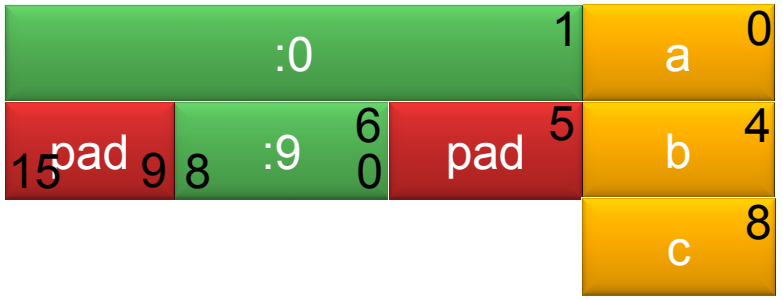
```
struct {
    short a:9;
    int b:9;
    char c;
    short d:9;
    short e:9;
    char f;
};
```




Size: 12 byte
Alignment: 4 byte

# Structures, Unions and Bitfields

```
struct {
    char a;
    int :0;
    char b;
    short :9;
    char c;
    char :0;
};
```




Size: 9 byte
Alignment: 1 byte

# Definitions

- Caller
  The calling function
- Callee
  The called function
- Caller Saved Registers
  Registers that are not preserved during a function call and need to be saved by the caller
- Callee Saved Registers
  Registers that are preserved during a function call and don't need to be saved by the caller. The callee needs to save and restore them.

# Register Convention (Linux ia32)

| Register | Description | Save site |
|---|---|---|
| %eax | Return value | Caller |
| %edx | Dividend register | Caller |
| %ecx | Count register | Caller |
| %ebx | Global Offset Table Base Register / Local register variable | Callee |
| %ebp | Stack frame pointer (optional) | Callee |
| %esi | Local register variable | Callee |
| %edi | Local register variable | Callee |
| %esp | Stack pointer | Callee |
| %st(0) | Floating point stack - top | |
| … | | |
| %st(7) | Floating point stack - bottom | |

# Calling Convention (Linux ia32)

- All arguments are passed on the stack
- The stack is always aligned to 4 bytes
- Arguments are promoted to 4 bytes
- Arguments are passed from right to left on the stack

```
void foo(
char c, short s,
int i, long l,
float f, double d);

foo(1,2,3,4,5.0,6.0);
```

| Offset | Stack |
|---|---|
| 24 | 6.0 (higher 4 byte) |
| 20 | 6.0 (lower 4 byte) |
| 16 | 5.0 |
| 12 | 4 |
| 8 | 3 |
| 4 | 2 |
| 0 | 1 |

# Stack Frame

| Base | Offset | Content | |
|---|---|---|---|
| %ebp | 4n+8 | argument word n | High Address |
| | | … | |
| | 8 | argument word 0 | |
| | 4 | return address | |
| %ebp | 0 | caller's %ebp | |
| %ebp | -4 | x words local space local variables, etc | |
| %ebp | -4x | | |
| %esp | 8 | caller's %edi | |
| %esp | 4 | caller's %esi | |
| %esp | 0 | caller's %ebx | Low Address |

# Calling Convention (Leaf Function)

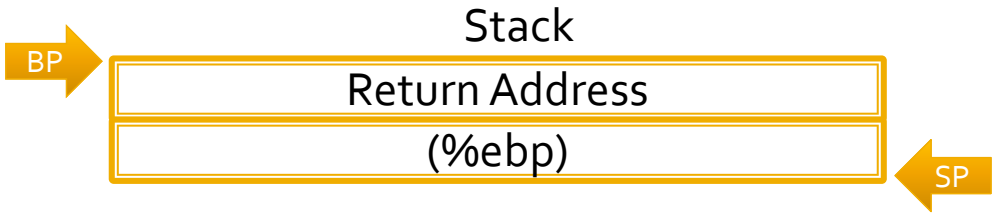
IP

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```

# Calling Convention (Leaf Function)

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```

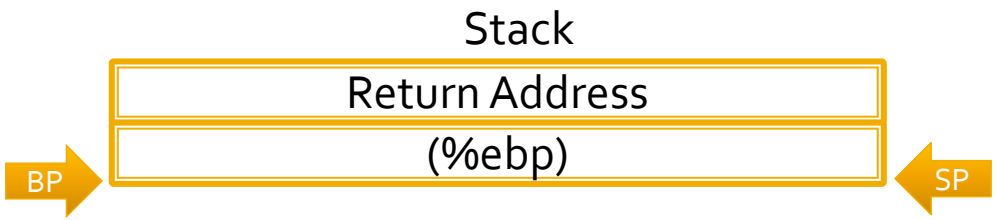
IP



# Calling Convention (Leaf Function)

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```

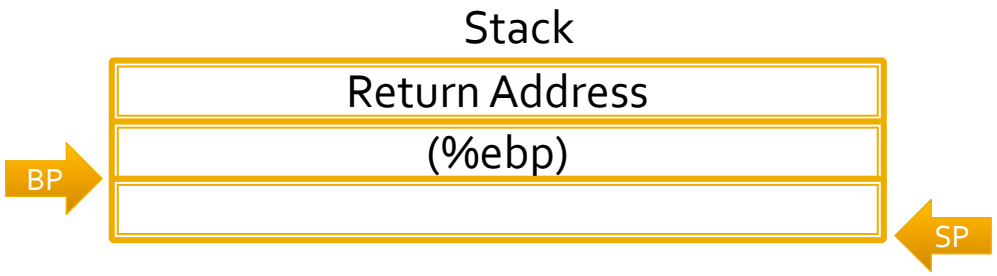
IP



# Calling Convention (Leaf Function)

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```

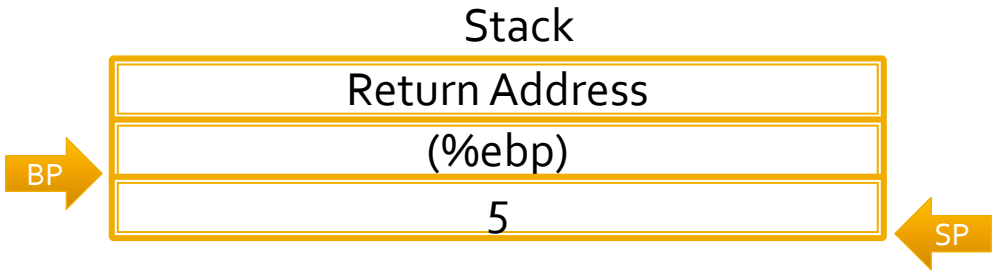
IP



# Calling Convention (Leaf Function)

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```

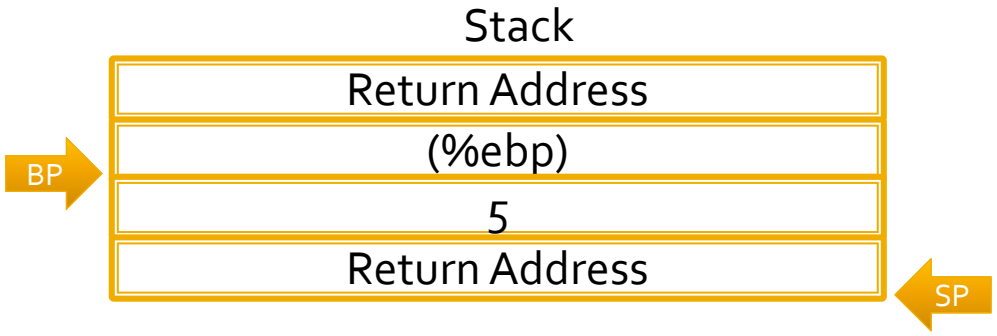
IP



# Calling Convention (Leaf Function)

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```

IP

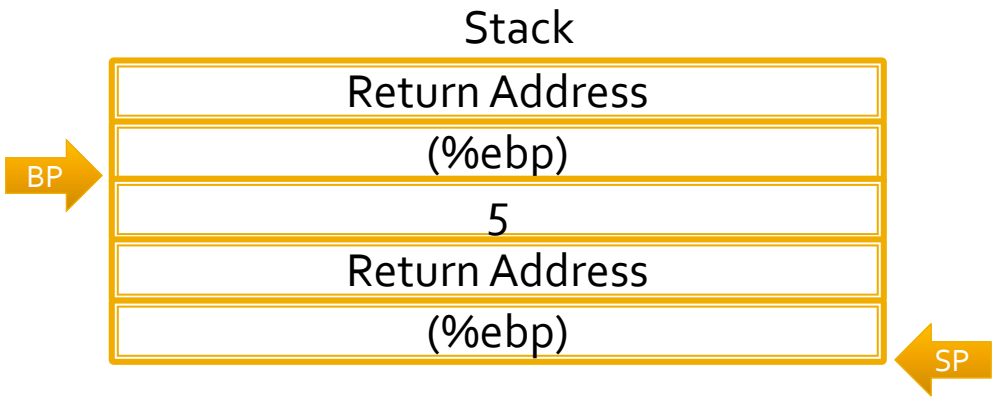


# Calling Convention (Leaf Function)

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
```

IP

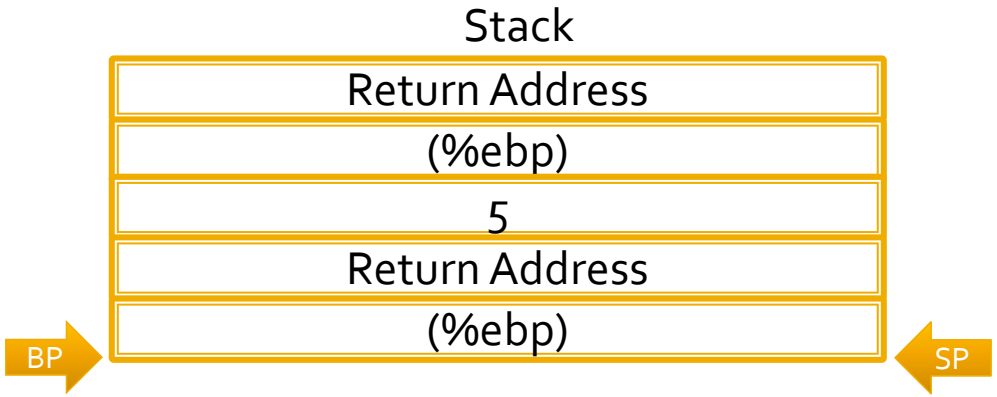
```
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```



# Calling Convention (Leaf Function)

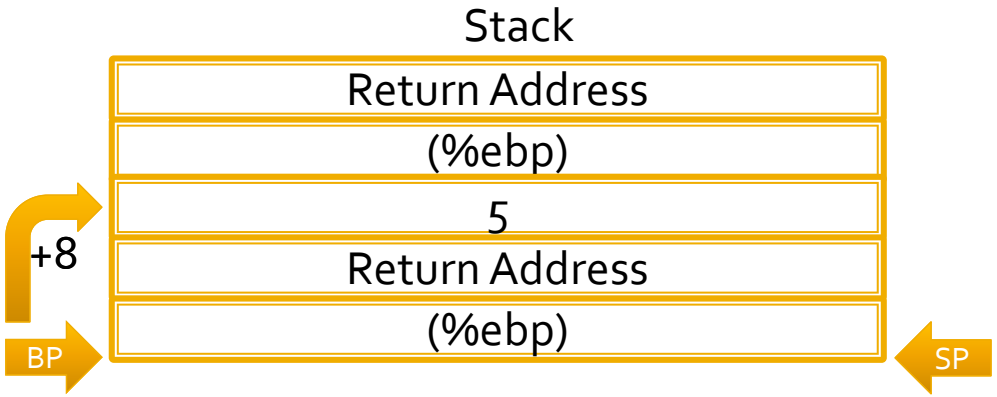
```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```

IP



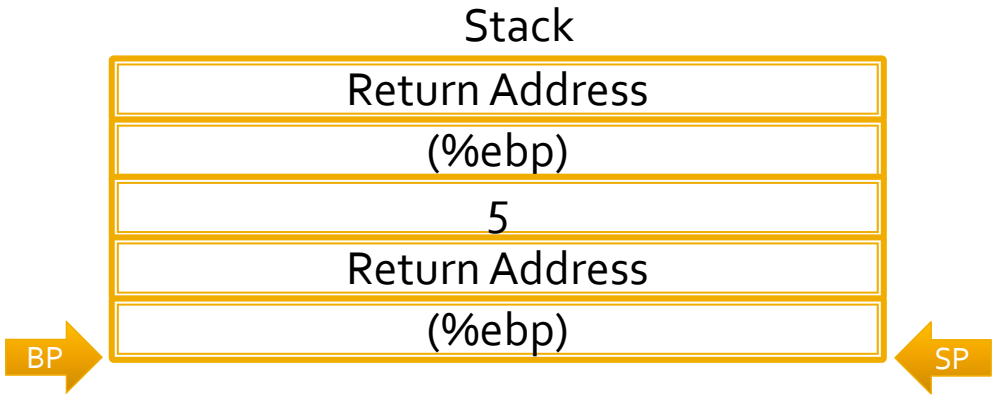
# Calling Convention (Leaf Function)

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
IP      movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```



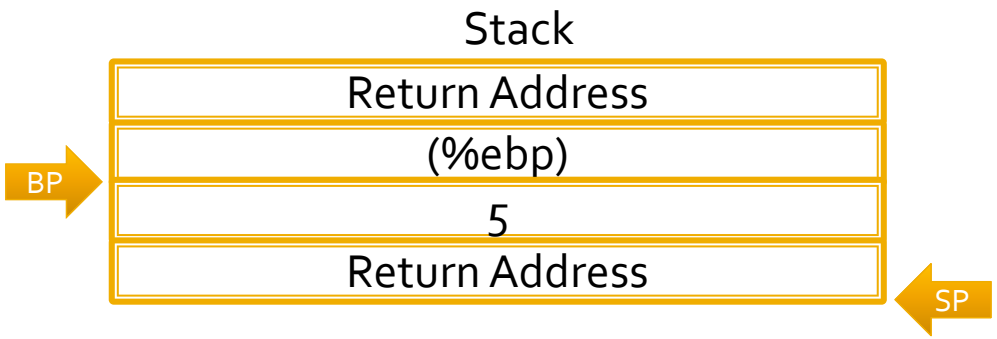
# Calling Convention (Leaf Function)

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
IP  %eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```



# Calling Convention (Leaf Function)

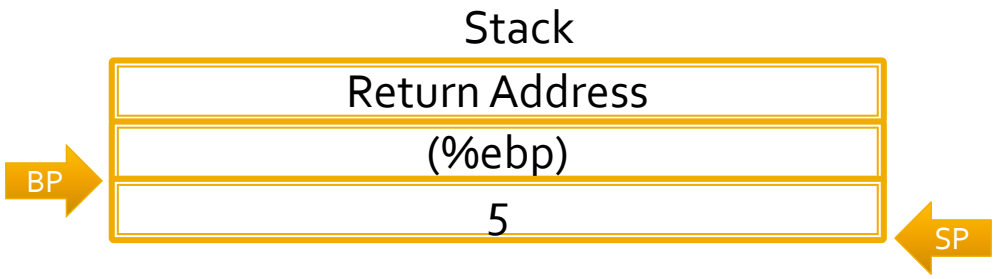
```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
IP      addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```



# Calling Convention (Leaf Function)

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```

IP



# Calling Convention (Leaf Function)

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
IP      ret
leave
ret
```

# Calling Convention (Leaf Function)

```
...
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    $5, (%esp)
call    foo
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp),
%eax
        addl    $1, %eax
        popl    %ebp
        ret
leave
ret
```

BP

SP

IP

Stack

# Calling Convention (Return of Structures)

- Caller needs to allocate memory for return value
- Passes address as hidden first argument to the function
- Callee needs to remove hidden argument from the stack upon completion
- Also returns address in %eax

| | After call |
|---|---|
| 4n+4(%esp) | argument word n |
| | … |
| 8(%esp) | argument word 1 |
| 4(%esp) | value address |
| 0(%esp) | return address |

| After return | |
|---|---|
| argument word n | 4n-4(%esp) |
| … | |
| argument word 1 | 0(%esp) |
| undefined | |
| | |

# Dynamic Stack Allocation

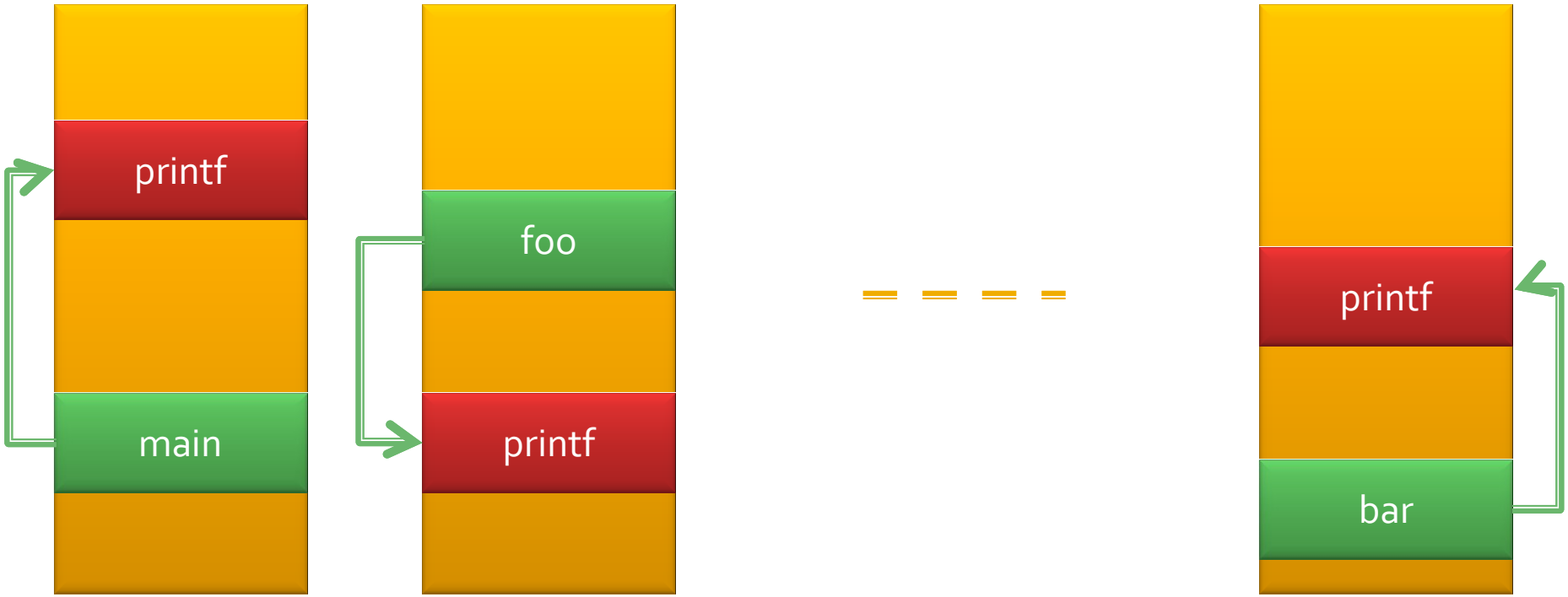
| | Original | Intermediate | Final | |
|---|---|---|---|---|
| 0(%ebp)<br><br>12(%esp) | arguments and automatic variables | arguments and automatic variables | arguments and automatic variables | 0(%ebp) |
| 0(%esp) | save area<br>3 words | save area<br>3 words | old save area<br>3 words | |
| | undefined | new space | new space | 12(%esp) |
| | | undefined | new save area<br>3 words | 0(%esp) |

# Assembler/Linker

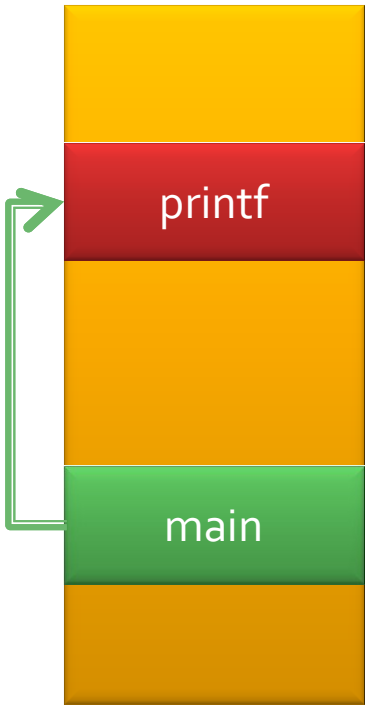
**Object Code Models:**

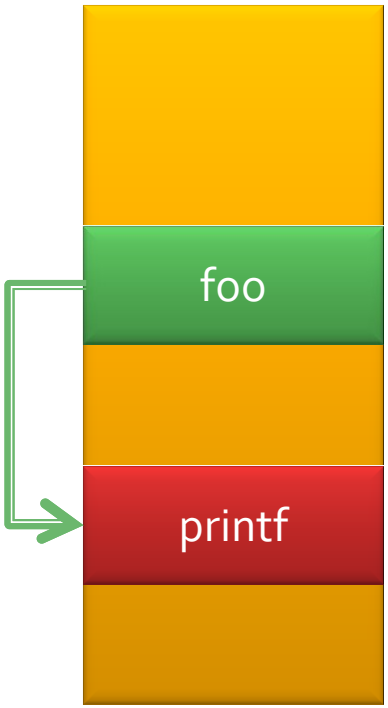
- Absolute Code
  Instructions hold absolute virtual addresses. The code has to be loaded at a specific address
- Position-independent Code
  All addresses are relative. Code can be loaded at any address.

# Object Code Models

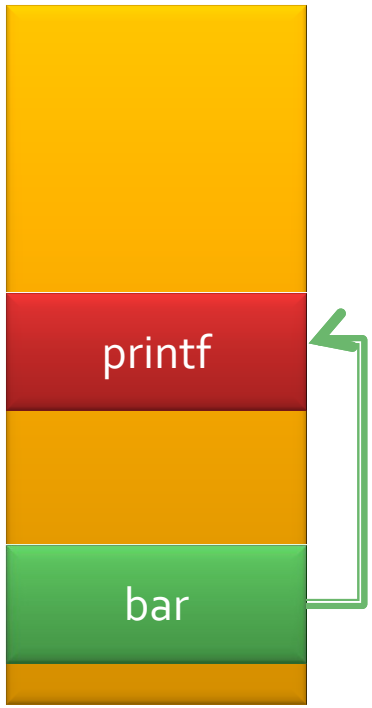


# Object Code Models

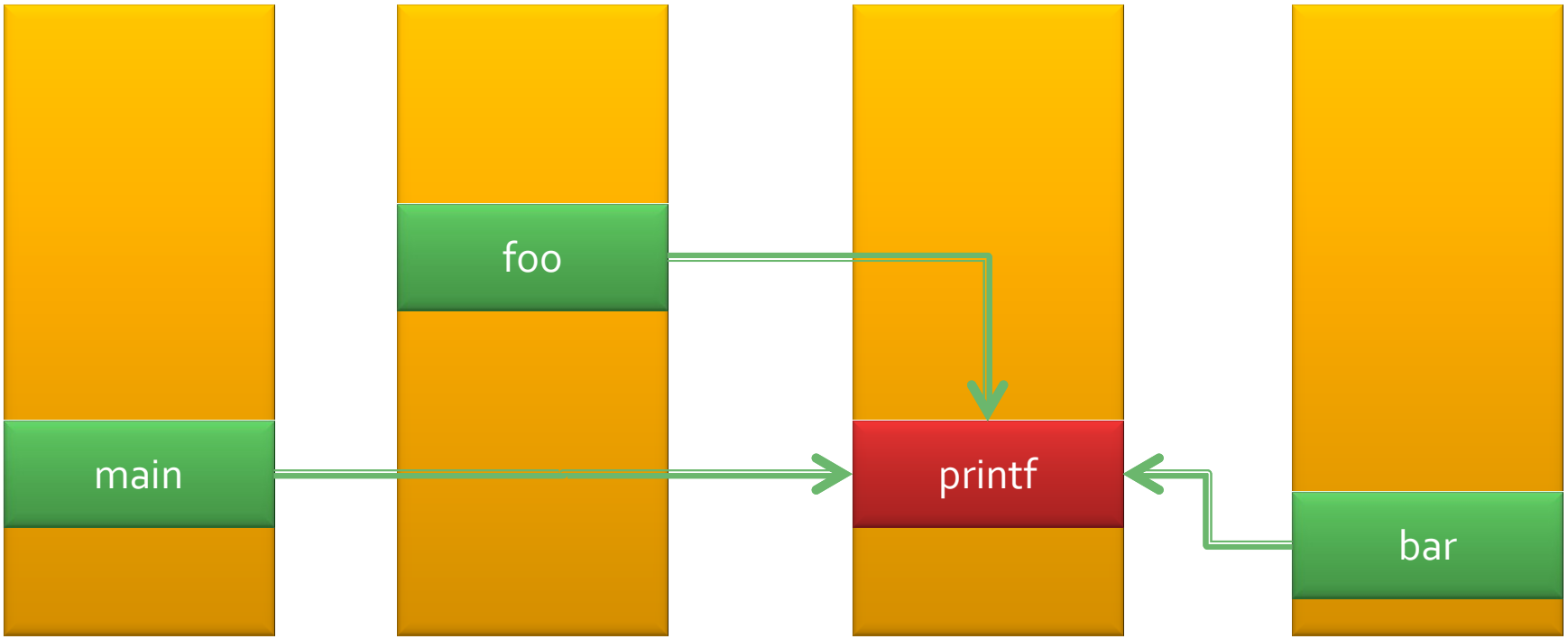


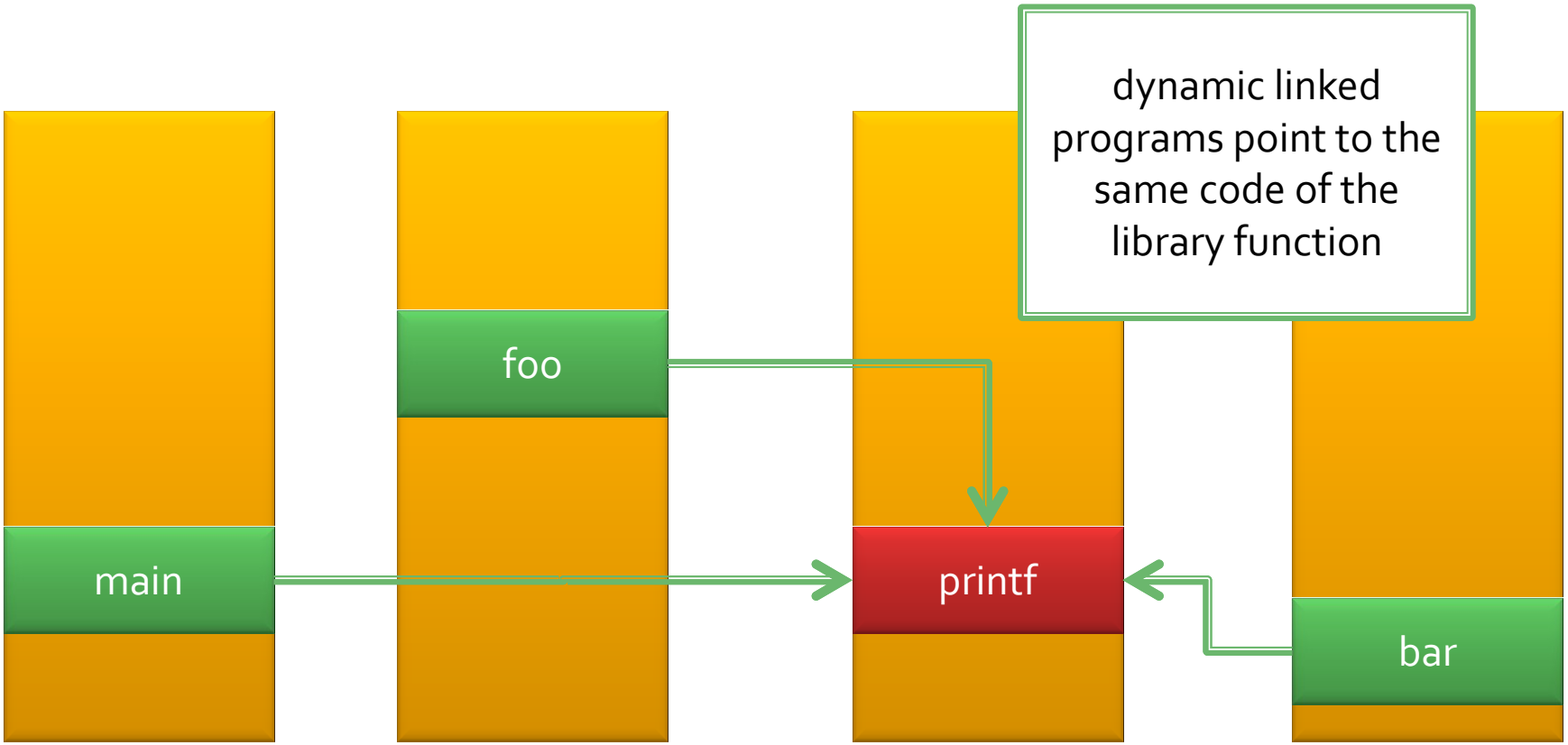

static linked programs have a copy of the library functions they use included in the binary



# Object Code Models



# Object Code Models



# Absolute vs Position-Independent Code (Global Variables)

**C CODE**

```
extern int src;
extern int dst;
extern int *ptr;
ptr = &dst;
*ptr = src;
```

**ABSOLUTE CODE**

```
.globl src, dst, ptr

movl $dst, ptr

movl ptr, %eax
movl src, %edx
movl %edx, (%eax
```

# Absolute vs Position-Independent Code (Global Variables)

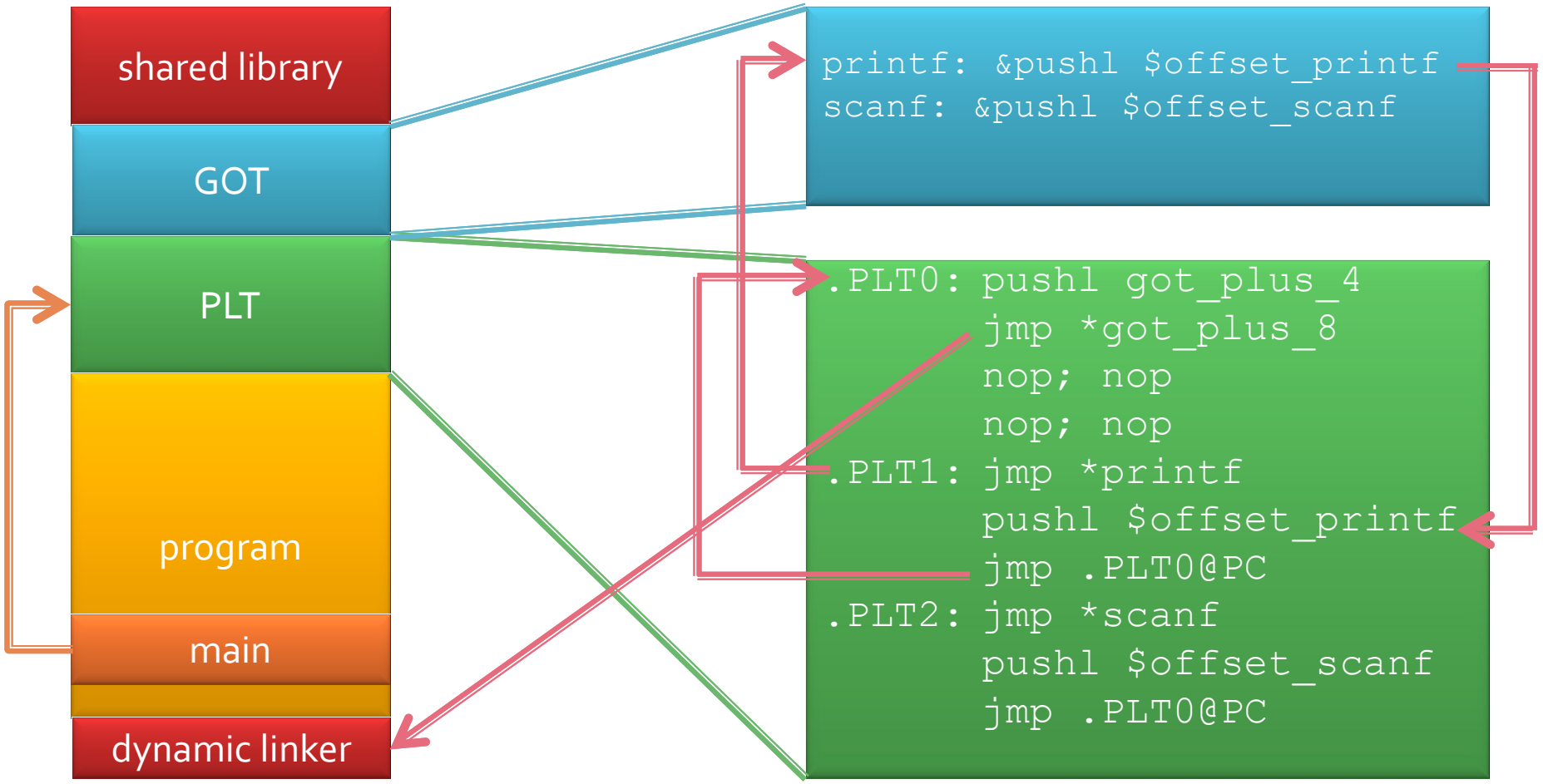
**C CODE**

```
extern int src;
extern int dst;
extern int *ptr;
ptr = &dst;
*ptr = src;
```

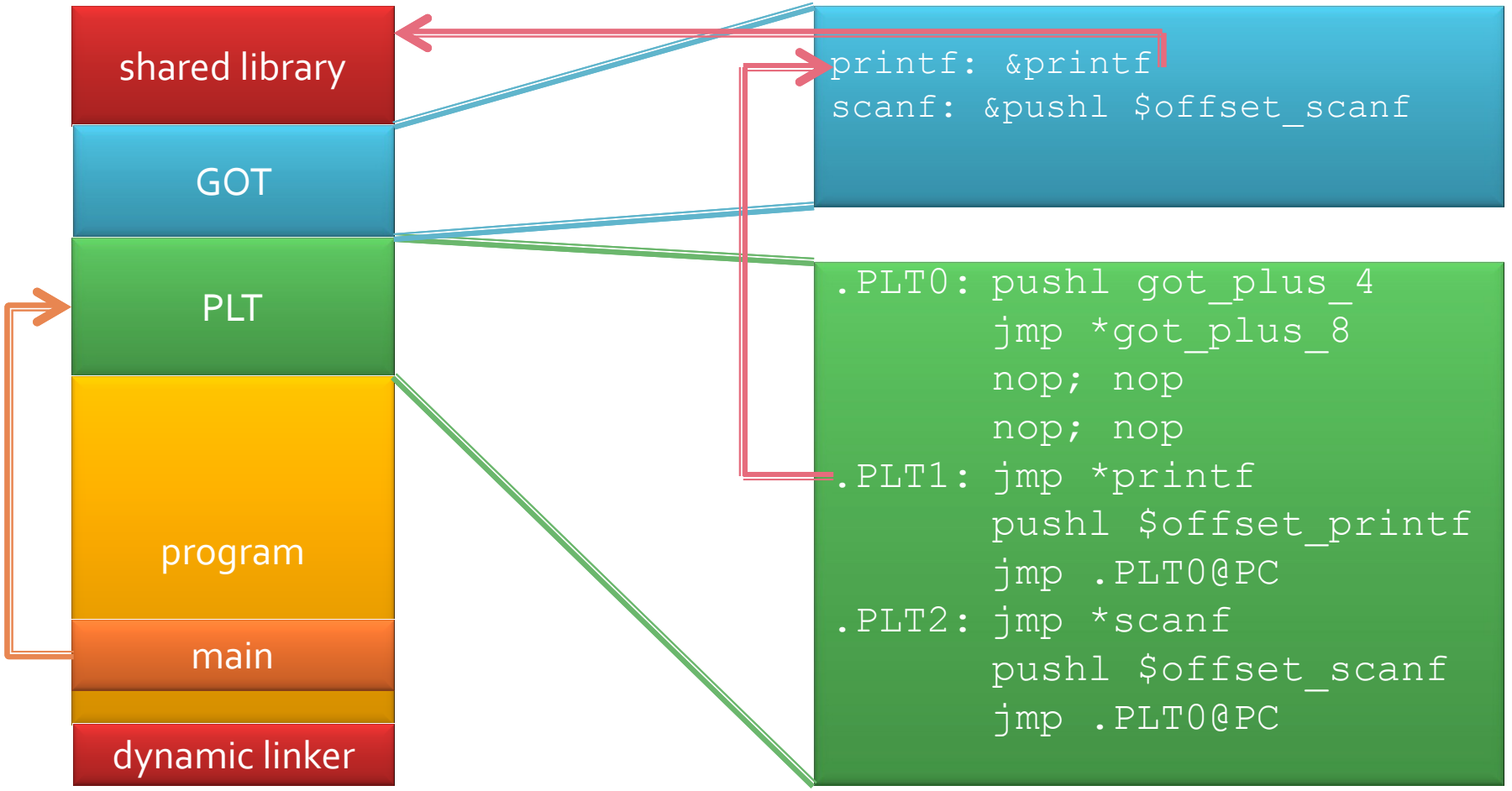
**POSITION-INDEPENDENT CODE**

```
.globl src, dst, ptr

movl ptr@GOT(%ebx), %eax
movl dst@GOT(%ebx), %edx
movl %edx, (%eax)

movl ptr@GOT(%ebx), %eax
movl (%eax), %eax
movl src@GOT(%ebx), %edx
movl (%edx), %edx
movl %edx, (%eax)
```

# Calling Dynamic Linked Library Functions



# Calling Dynamic Linked Library Functions



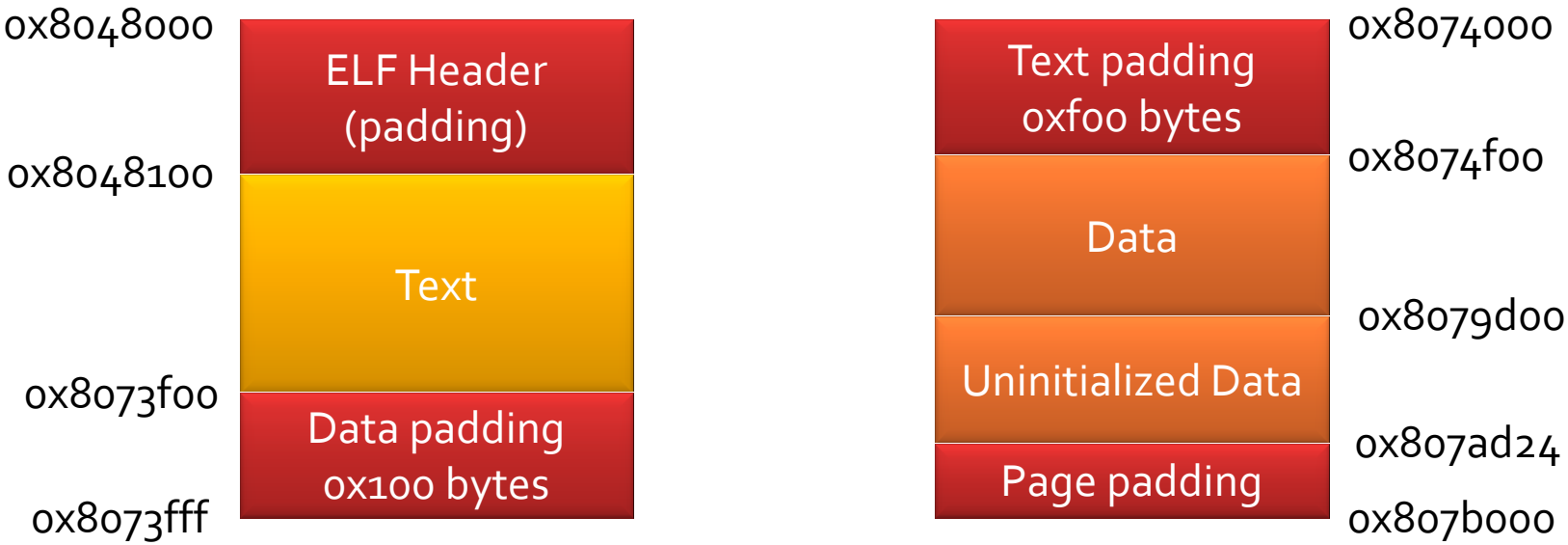
# Loader

| File Offset | File | Virtual Address |
|---|---|---|
| 0 | ELF Header Table | |
| | Pogramm Header Table | |
| 0x100 | Text Segment<br>...<br>0x2be00 bytes | 0x8048100<br><br>0x8073eff |
| 0x2bf00 | Data Segment<br>...<br>0x4e00 bytes | 0x8074f00<br><br>0x8079cff |
| 0x30d00 | Section Header Table | |

# Loader

| | Text | Data |
|---|---|---|
| Type | Load | Load |
| Offset | 0x100 | 0x2bf00 |
| Virtual Address | 0x8048100 | 0x8074f00 |
| File Size | 0x2be00 | 0x4e00 |
| Memory Size | 0x2be00 | 0x5e24 |
| Alignment | 0x1000 | 0x1000 |

# Loader


0x8048000
ELF Header (padding)
0x8048100
Text
0x8073f00
Data padding 0x100 bytes
0x8073fff
Text padding oxfoo bytes
0x8074000
0x8074f00
Data
0x8079d00
Uninitialized Data
0x807ad24
Page padding
0x807b000

# Program Memory Map

| | |
|---|---|
| 0xffffffff<br><br>0x80000000 | Dynamic Segment |
| 0x7fffffff | Data Segment |
| <br><br>0x8048000 | Text Segment |
| 0x8047fff<br><br>0x0000000 | Stack Segment |

# More details …

- Linkers & Loaders
  free draft at http://www.iecc.com/linker/
- SYSTEM V APPLICATION BINARY INTERFACE Intel386 Architecture Processor Supplement, Fourth Edition
- System V Application Binary Interface AMD64 Architecture Processor Supplement, Draft Version 0.99.5