UPDF
WWW.UPDF.COM

Machine-Level Programming III: Procedures

COMP400727: Introduction to Computer Systems

Danfeng Shan
Xi'an Jiaotong University

UPDF
WWW.UPDF.COM

Mechanisms in Procedures

Passing control 控制权转移

- To beginning of procedure code
- Back to return point

Passing data 传参

- Procedure arguments
- Return value

Memory management 内存分配

- Allocate during procedure execution
- Deallocate upon return

Mechanisms all implemented with machine instructions

x86-64 implementation of a procedure uses only those mechanisms required

```
P(…) {
   •
   •
   y = Q(x);
   print(y)
   •
}
```

```
int Q(int i)
{
   int t = 3*i;
   int v[10];
   •
   •
   return v[t];
}
```

UPDF
WWW.UPDF.COM

Mechanisms in Procedures

Passing control

- To beginning of procedure code
- Back to return point

Passing data

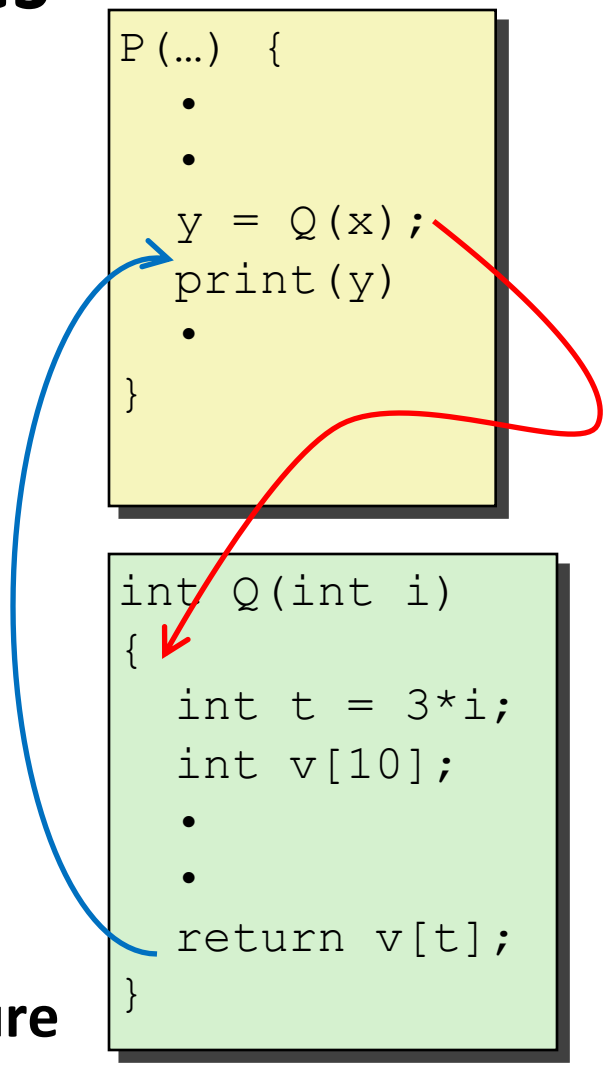
- Procedure arguments
- Return value

Memory management

- Allocate during procedure execution
- Deallocate upon return

Mechanisms all implemented with machine instructions

x86-64 implementation of a procedure uses only those mechanisms required



UPDF
WWW.UPDF.COM

Mechanisms in Procedures

Passing control

- To beginning of procedure code
- Back to return point

Passing data

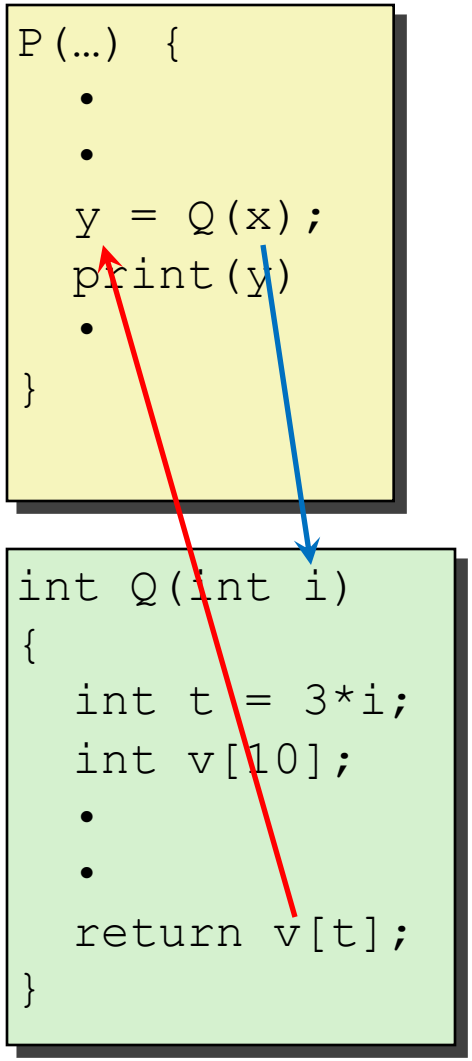
- Procedure arguments
- Return value

Memory management

- Allocate during procedure execution
- Deallocate upon return

Mechanisms all implemented with machine instructions

x86-64 implementation of a procedure uses only those mechanisms required



UPDF
WWW.UPDF.COM

Mechanisms in Procedures

Passing control

- To beginning of procedure code
- Back to return point

Passing data

- Procedure arguments
- Return value

Memory management

- Allocate during procedure execution
- Deallocate upon return

Mechanisms all implemented with machine instructions

x86-64 implementation of a procedure uses only those mechanisms required

```
P(…) {
   •
   •
   y = Q(x);
   print(y)
   •
}
```

```
int Q(int i)
{
   int t = 3*i;
   int v[10];
   •
   •
   return v[t];
}
```

UPDF
WWW.UPDF.COM

Mechanisms in Procedures

Machine instructions implement the mechanisms, but the choices are determined by designers. These choices make up the **Application Binary Interface (ABI)**.

Deallocate upon return

Mechanisms all implemented with machine instructions

x86-64 implementation of a procedure uses only those mechanisms required

```
P(…) {
  int v[10];
  •
  •
  return v[t];
}
```

UPDF
WWW.UPDF.COM

Today

Procedures

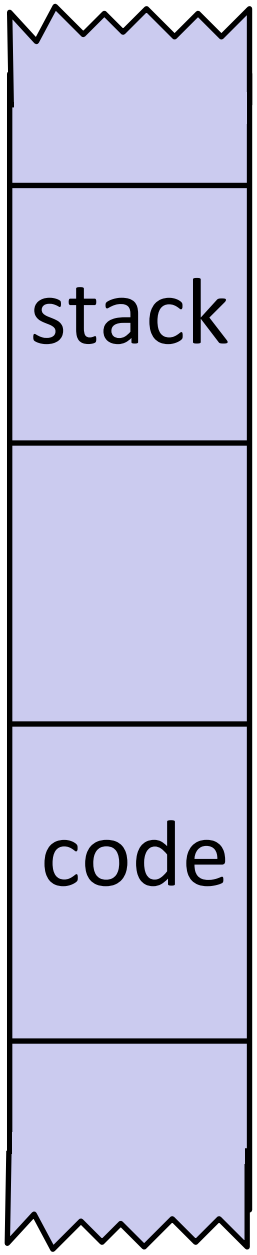
- **Stack Structure**
- Calling Conventions
  - Passing control
  - Passing data
  - Managing local data
- If we have time: illustration of recursion

UPDF
WWW.UPDF.COM

x86-64 Stack

Region of memory managed with stack discipline

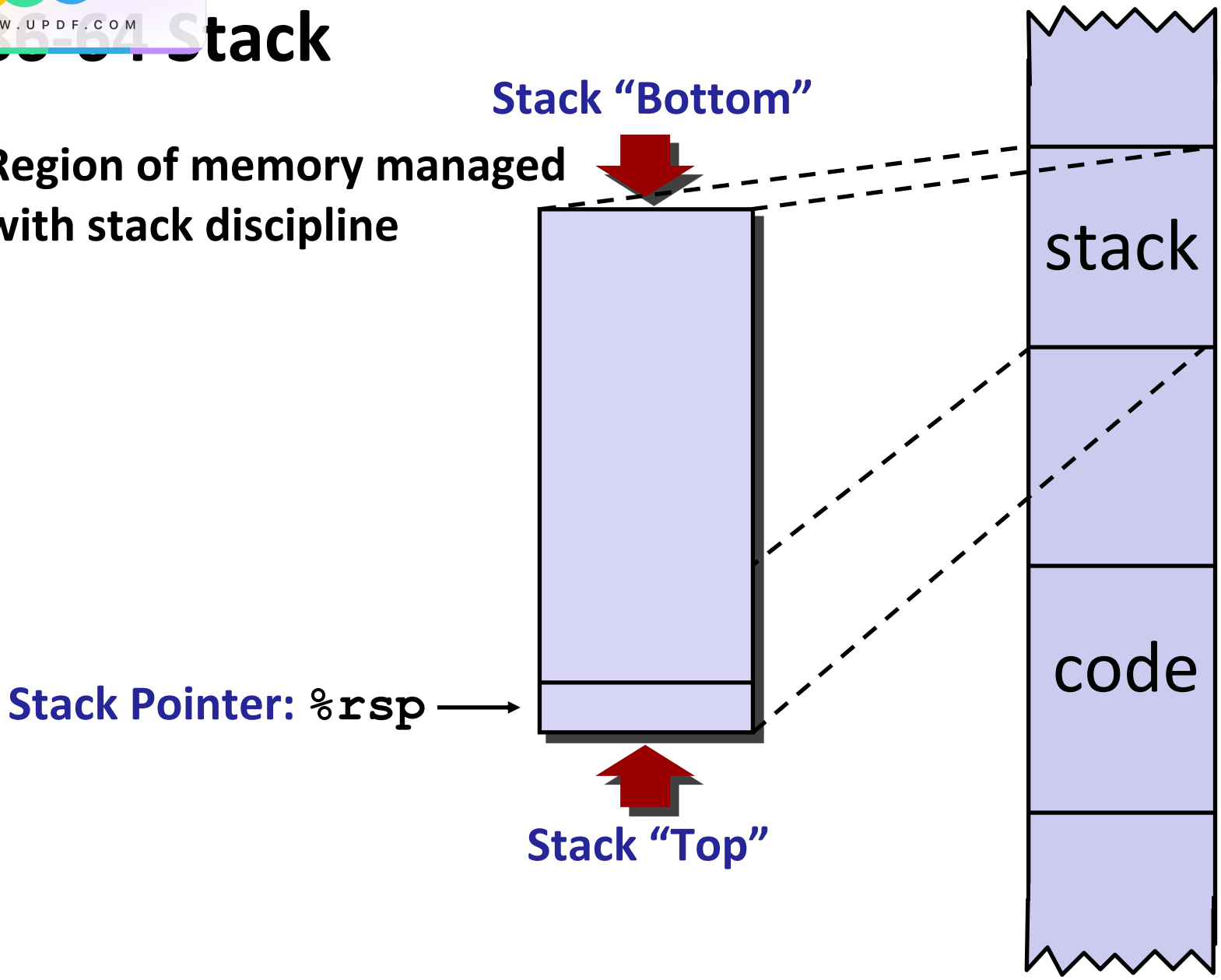
- Memory viewed as array of bytes.
- Different regions have different purposes.
- (Like ABI, a policy decision)



UPDF
WWW.UPDF.COM

x86-64 Stack

Region of memory managed with stack discipline



UPDF
WWW.UPDF.COM

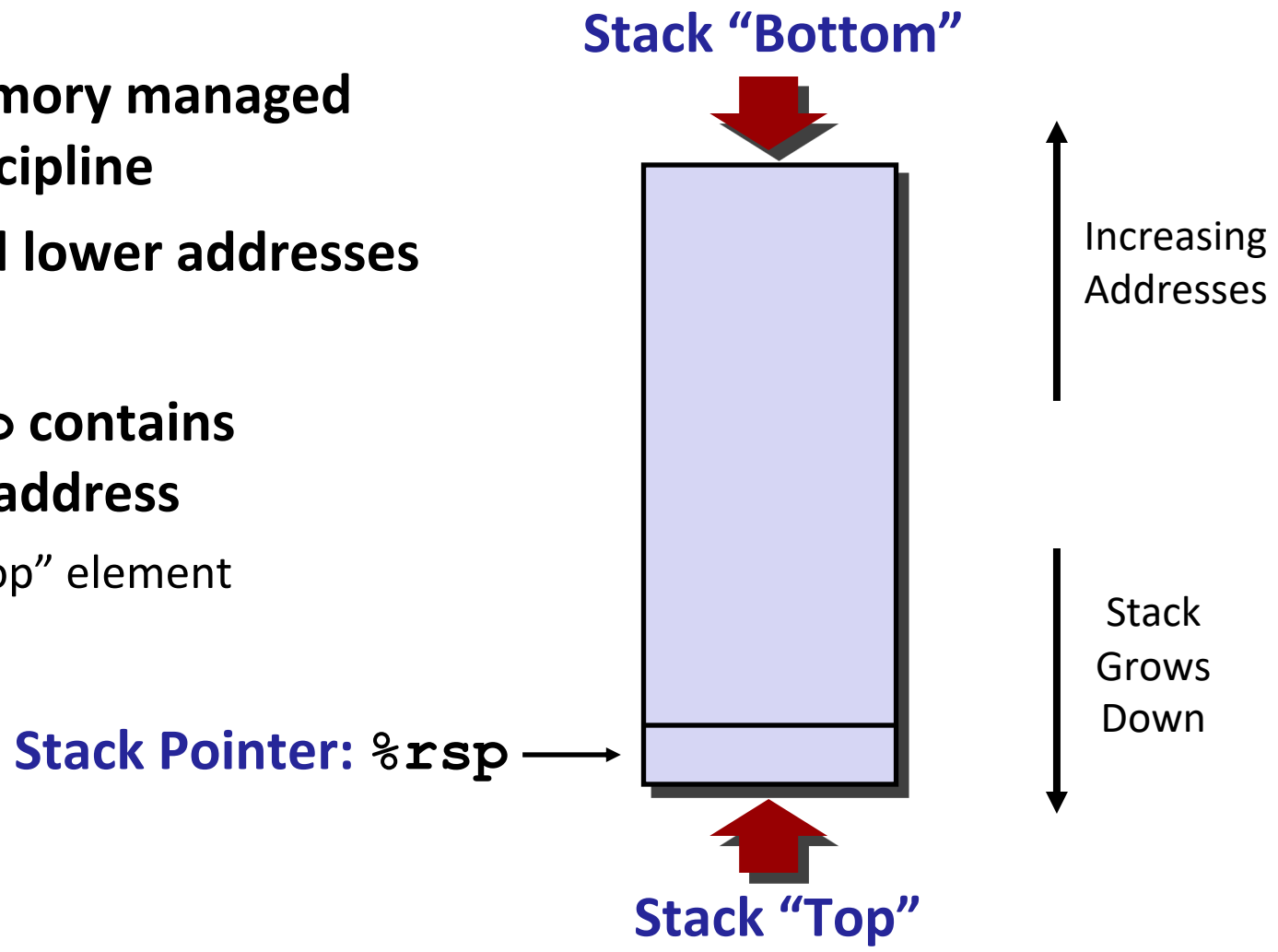
x86-64 Stack

Region of memory managed with stack discipline

Grows toward lower addresses

Register `%rsp` contains lowest stack address

address of "top" element



UPDF
WWW.UPDF.COM

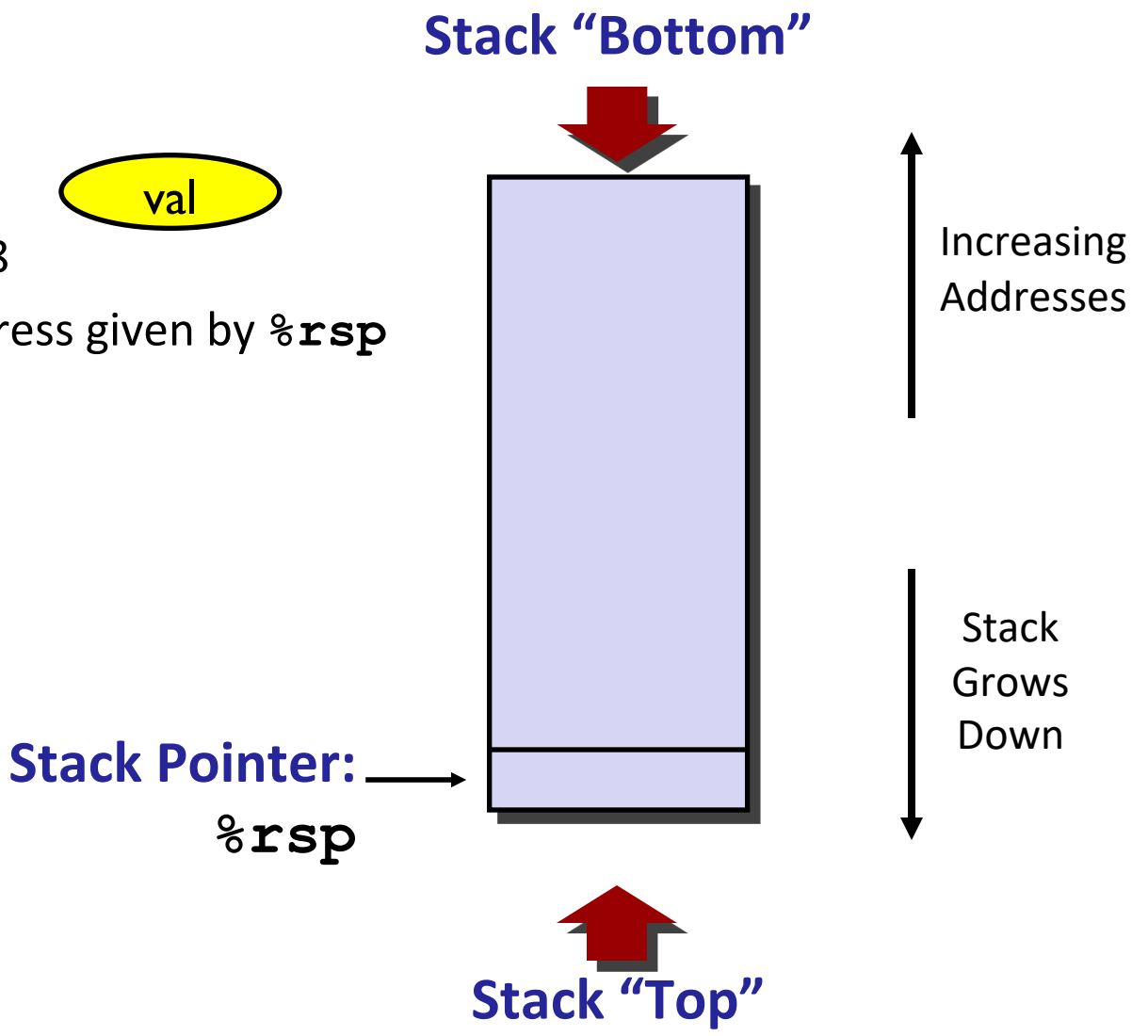
x86-64 Stack: Push

`pushq` *Src*

Fetch operand at *Src*

Decrement **`%rsp`** by 8

Write operand at address given by **`%rsp`**



UPDF
WWW.UPDF.COM

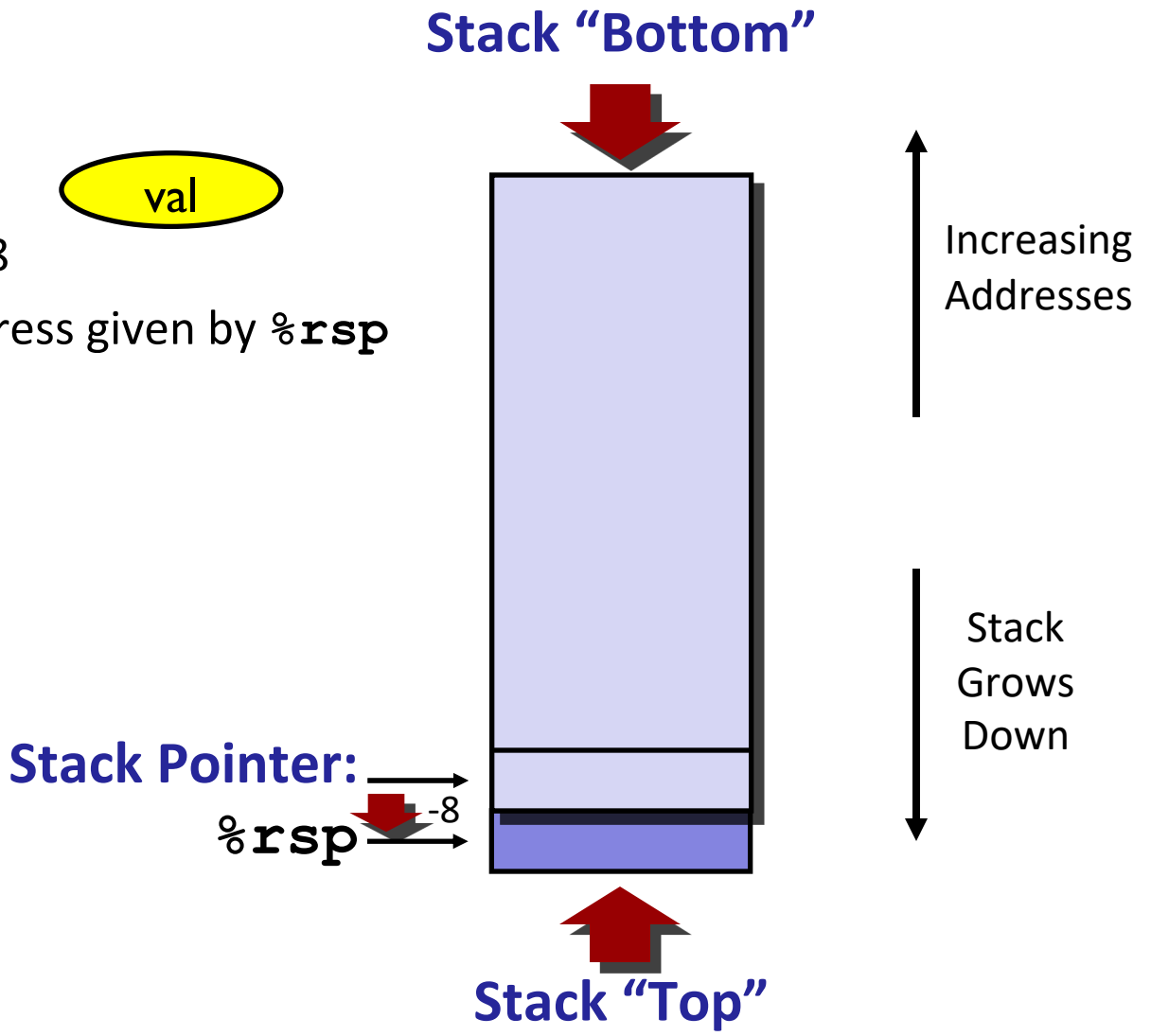
x86-64 Stack: Push

`pushq` *Src*

Fetch operand at *Src*

Decrement **`%rsp`** by 8

Write operand at address given by **`%rsp`**



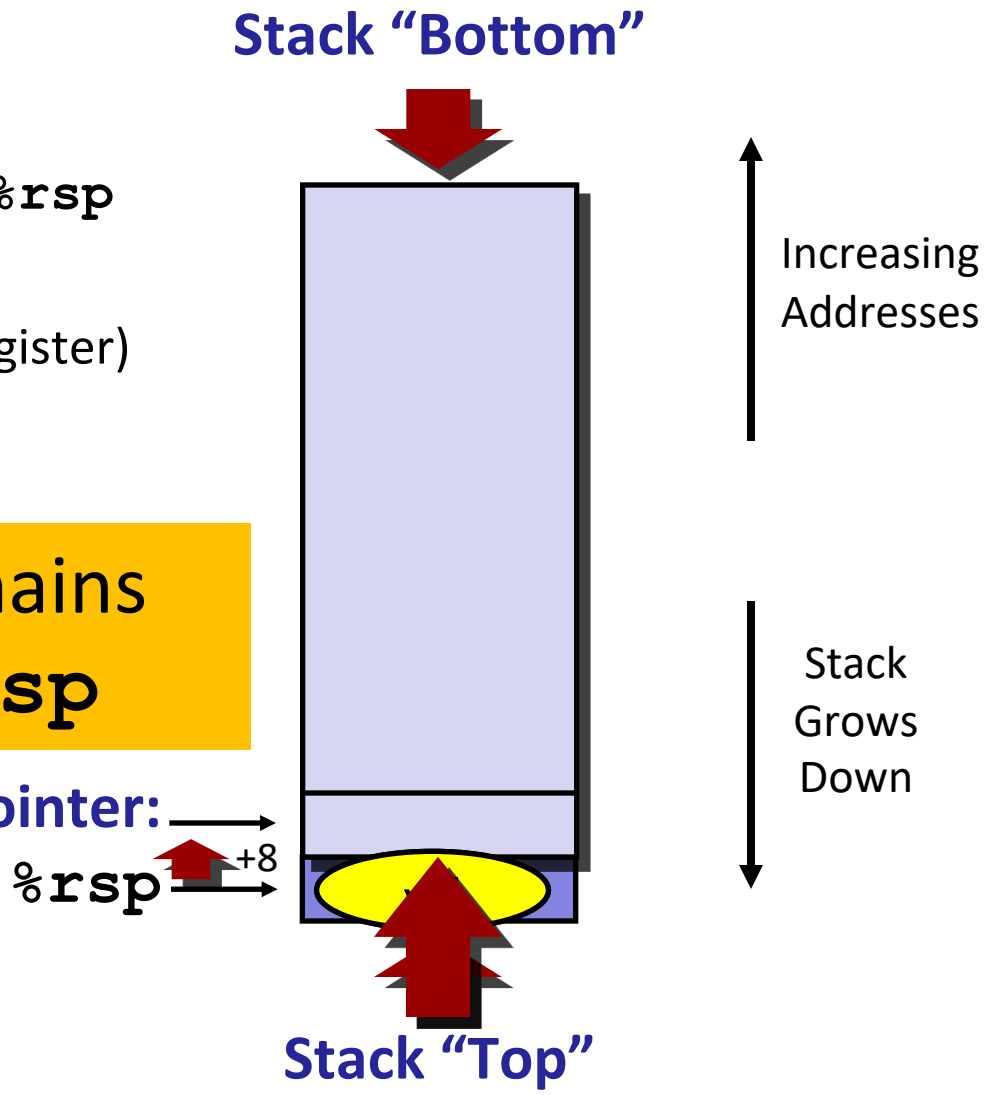
UPDF
WWW.UPDF.COM

x86-64 Stack: Pop

`popq` ***Dest***

- Read value at address given by **`%rsp`**
- Increment **`%rsp`** by 8
- Store value at Dest (usually a register)

Value is **copied**; it remains in memory at old **`%rsp`**



UPDF
WWW.UPDF.COM

Today

Procedures

- Stack Structure
- Calling Conventions
  - **Passing control**
  - Passing data
  - Managing local data
- If we have time: illustration of recursion

UPDF
WWW.UPDF.COM

Code Examples

```
void multstore(long x, long y, long *dest)
{
    long t = mult2(x, y);
    *dest = t;
}
```

```
0000000000400540 <multstore>:
  400540: push   %rbx                  # Save %rbx
  400541: mov    %rdx,%rbx             # Save dest
  400544: call   400550 <mult2>        # mult2(x,y)
  400549: mov    %rax,(%rbx)           # Save at dest
  40054c: pop    %rbx                  # Restore %rbx
  40054d: ret                          # Return
```

```
long mult2(long a, long b)
{
  long s = a * b;
  return s;
}
```

```
0000000000400550 <mult2>:
  400550:  mov    %rdi,%rax            # a
  400553:  imul   %rsi,%rax            # a * b
  400557:  ret                         # Return
```

UPDF
WWW.UPDF.COM

Procedure Control Flow

Use stack to support procedure call and return

Procedure call: `call label`

- Push return address on stack
- Jump to ***label***

Return address:

- Address of the next instruction right after call
- Example from disassembly

Procedure return: `ret`

- Pop address from stack
- Jump to address

These instructions are sometimes printed with a `q` suffix

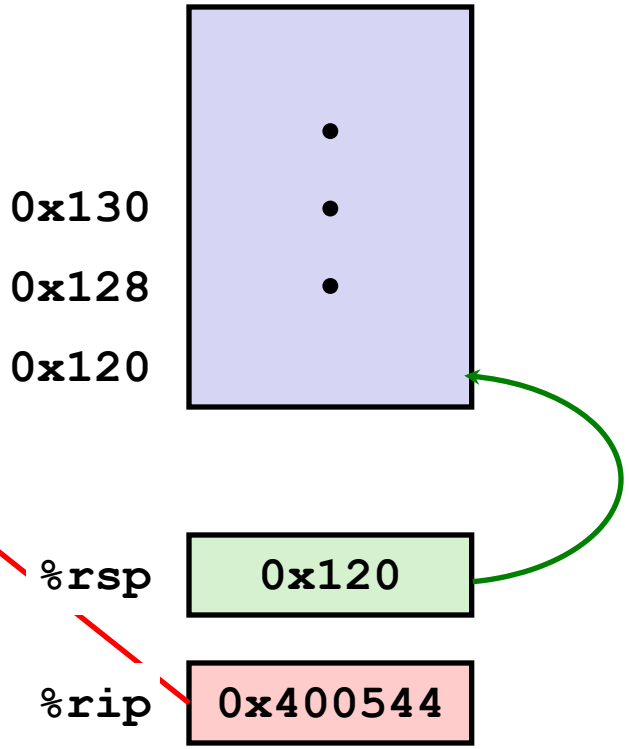
- This is just to remind you that you're looking at 64-bit code

UPDF
WWW.UPDF.COM

Control Flow Example #1

```
0000000000400540 <multstore>:
  •
  •
  400544: call   400550 <mult2>
  400549: mov    %rax,(%rbx)
  •
  •
```

```
0000000000400550 <mult2>:
  400550:  mov    %rdi,%rax
  •
  •
  400557:  ret
```

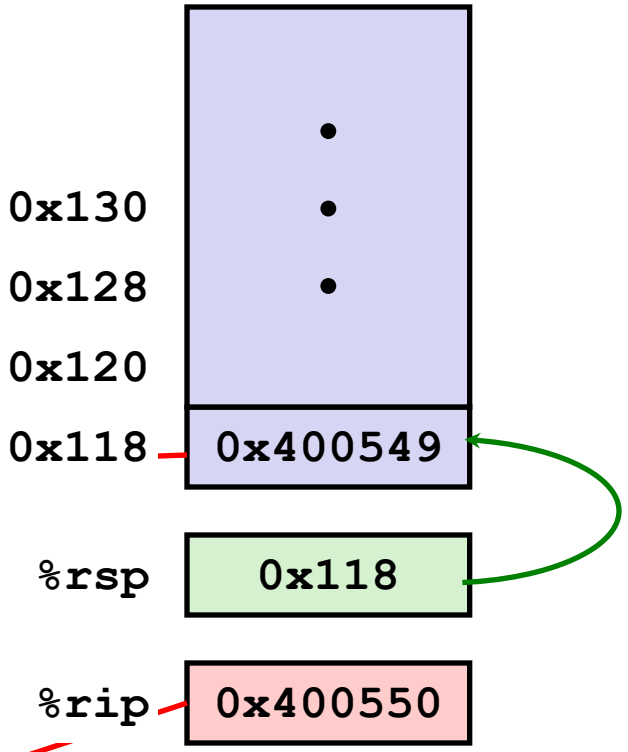


UPDF
WWW.UPDF.COM

Control Flow Example #2

```
0000000000400540 <multstore>:
  •
  •
  400544: call   400550 <mult2>
  400549: mov    %rax,(%rbx)
  •
  •
```

```
0000000000400550 <mult2>:
  400550:  mov    %rdi,%rax
  •
  •
  400557:  ret
```

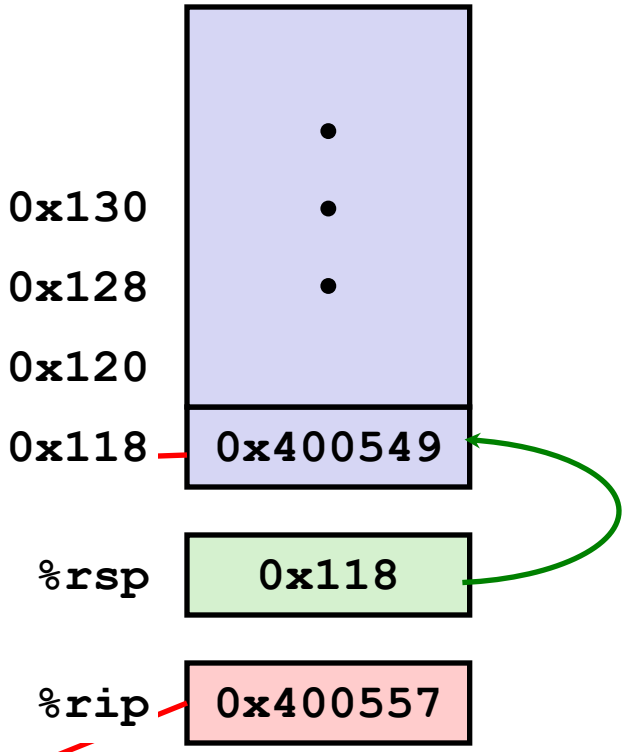


UPDF
WWW.UPDF.COM

Control Flow Example #3

```
0000000000400540 <multstore>:
  •
  •
  400544: call   400550 <mult2>
  400549: mov    %rax,(%rbx)
  •
  •
```

```
0000000000400550 <mult2>:
  400550:  mov    %rdi,%rax
  •
  •
  400557:  ret
```

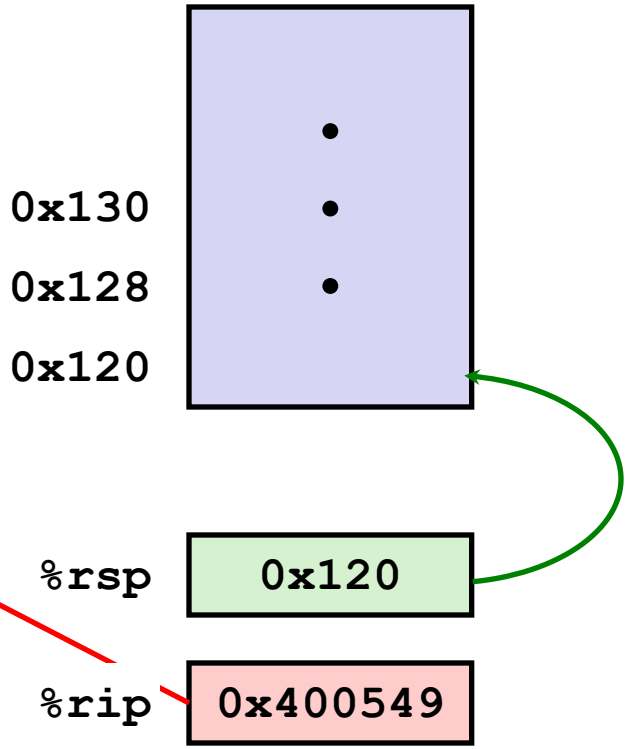


UPDF
WWW.UPDF.COM

Control Flow Example #4

```
0000000000400540 <multstore>:
  •
  •
  400544: call   400550 <mult2>
  400549: mov    %rax,(%rbx)
  •
  •
```

```
0000000000400550 <mult2>:
  400550:  mov    %rdi,%rax
  •
  •
  400557:  ret
```



UPDF
WWW.UPDF.COM

Today

Procedures

- Stack Structure
- Calling Conventions
  - Passing control
  - **Passing data**
  - Managing local data
- If we have time: illustration of recursion

UPDF
WWW.UPDF.COM

Procedure Data Flow

Registers

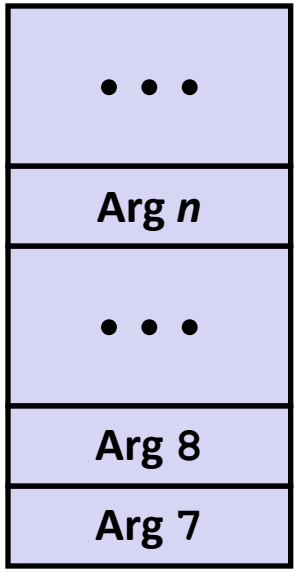
First 6 arguments

%rdi
%rsi
%rdx
%rcx
%r8
%r9

Return value

%rax

Stack



Only allocate stack space when needed

UPDF
WWW.UPDF.COM

Data Flow Examples

```
void multstore
 (long x, long y, long *dest)
{
    long t = mult2(x, y);
    *dest = t;
}
```

```
0000000000400540 <multstore>:
  # x in %rdi, y in %rsi, dest in %rdx
  • • •
  400541: mov    %rdx,%rbx          # Save dest
  400544: call   400550 <mult2>     # mult2(x,y)
  # t in %rax
  400549: mov    %rax,(%rbx)        # Save at dest
  • • •
```

```
long mult2
  (long a, long b)
{
  long s = a * b;
  return s;
}
```

```
0000000000400550 <mult2>:
  # a in %rdi, b in %rsi
  400550:  mov    %rdi,%rax         # a
  400553:  imul   %rsi,%rax         # a * b
  # s in %rax
  400557:  ret                      # Return
```

UPDF
WWW.UPDF.COM

Today

Procedures

- Stack Structure
- Calling Conventions
  - Passing control
  - Passing data
  - **Managing local data**
- If we have time: illustration of recursion

UPDF
WWW.UPDF.COM

Stack-Based Languages

Languages that support recursion

- e.g., C, Pascal, Java
- Code must be "*Reentrant*"
  - Multiple simultaneous instantiations of single procedure
- Need some place to store state of each instantiation
  - Arguments
  - Local variables
  - Return pointer

Stack discipline

- State for given procedure needed for limited time
  - From when called to when return
- Callee returns before caller does

Stack allocated in *Frames*

- state for single procedure instantiation

UPDF
WWW.UPDF.COM

Call Chain Example

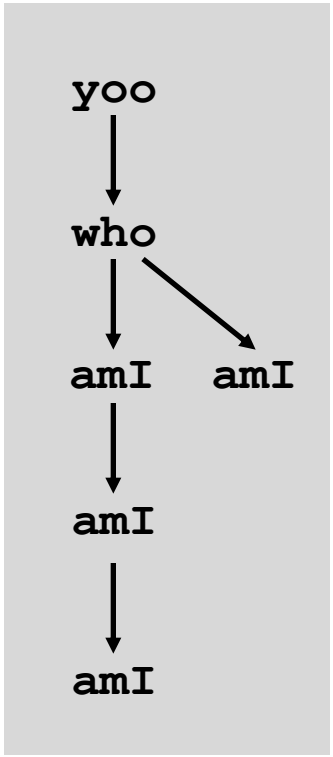
```
yoo(…)
{
   •
   •
   who();
   •
   •
}
```

```
who(…)
{
   • • •
   amI();
   • • •
   amI();
   • • •
}
```

```
amI(…)
{
   •
   •
   amI();
   •
   •
}
```

Procedure `amI()` is recursive

Example Call Chain



UPDF
WWW.UPDF.COM

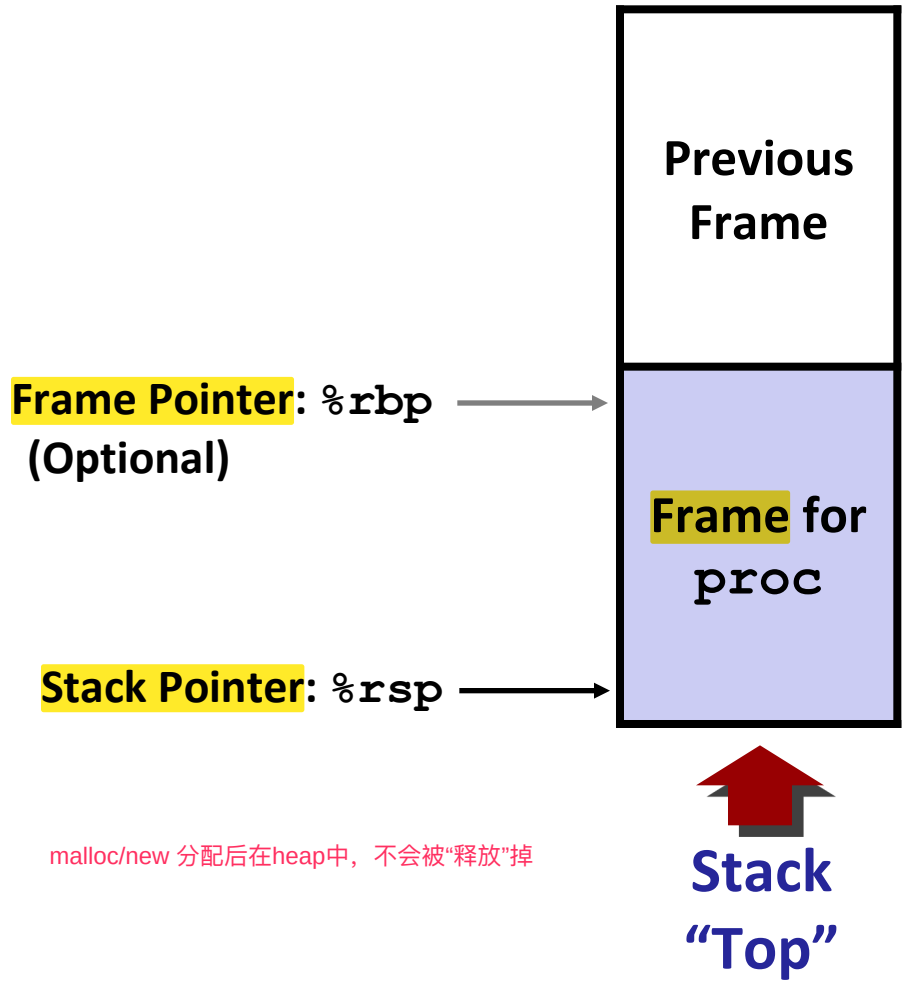
Stack Frames

Contents

Return information

Local storage (if needed)

Temporary space (if needed)



Management

Space allocated when enter procedure

malloc/new 分配后在heap中，不会被"释放"掉

"Set-up" code

开启函数 => 需要内存时：栈顶指针向下移

Includes push by **`call`** instruction

Deallocated when return

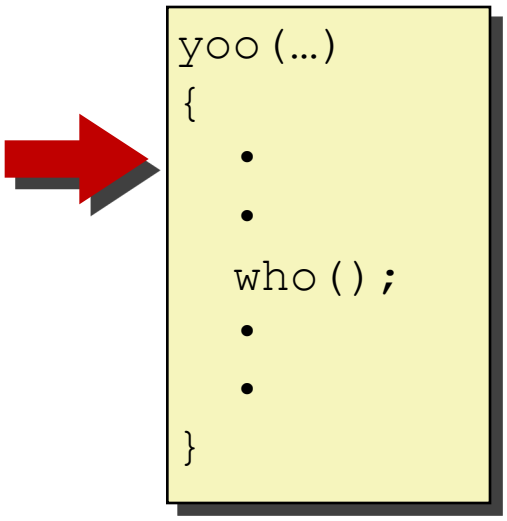
"Finish" code

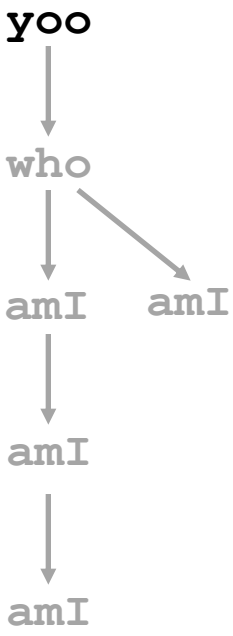
关闭函数 => 不需要内存时：栈顶指针向上移

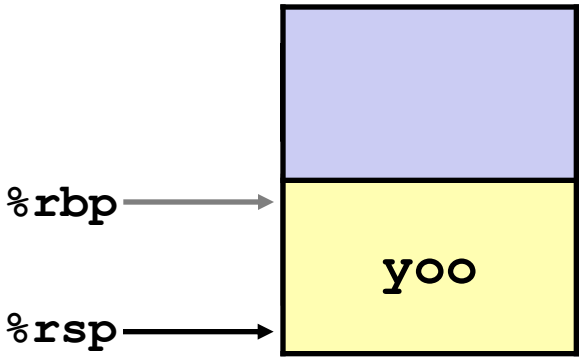
Includes pop by **`ret`** instruction

UPDF
WWW.UPDF.COM

Example

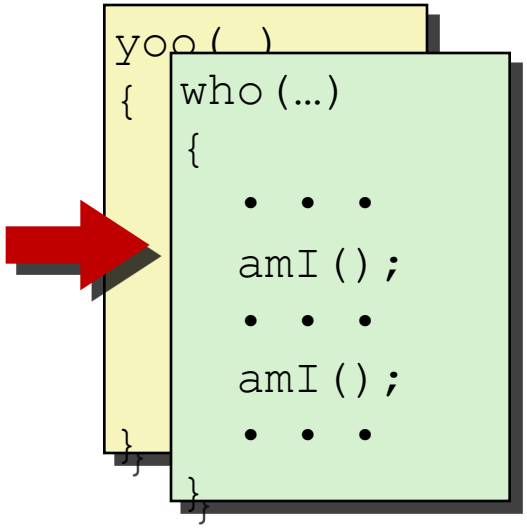


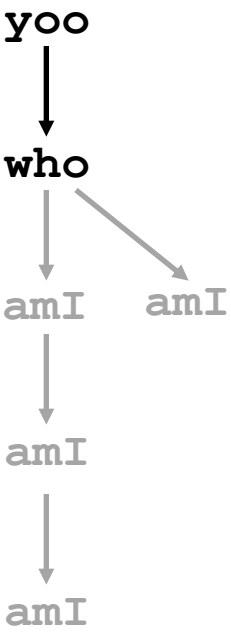


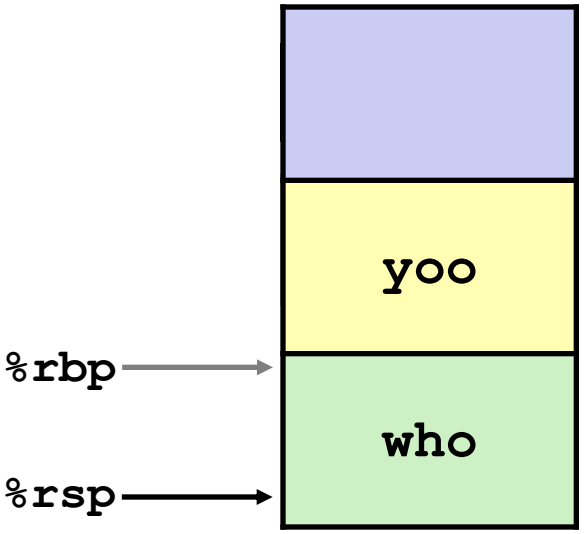


UPDF
WWW.UPDF.COM

Example

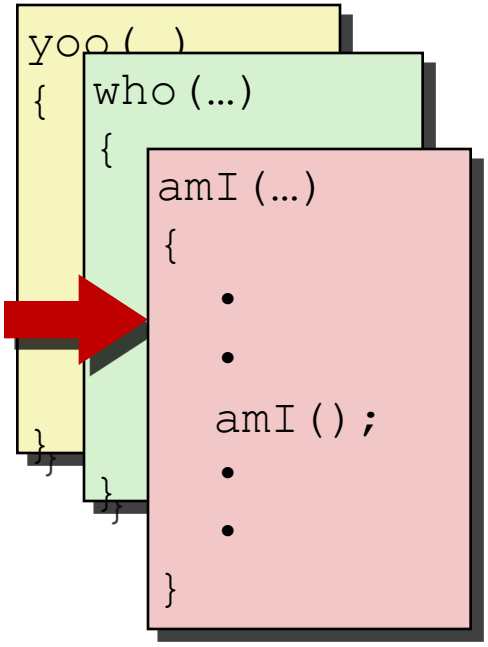


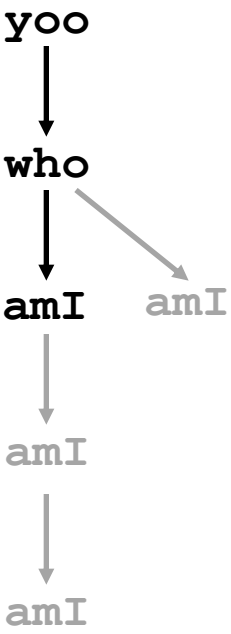


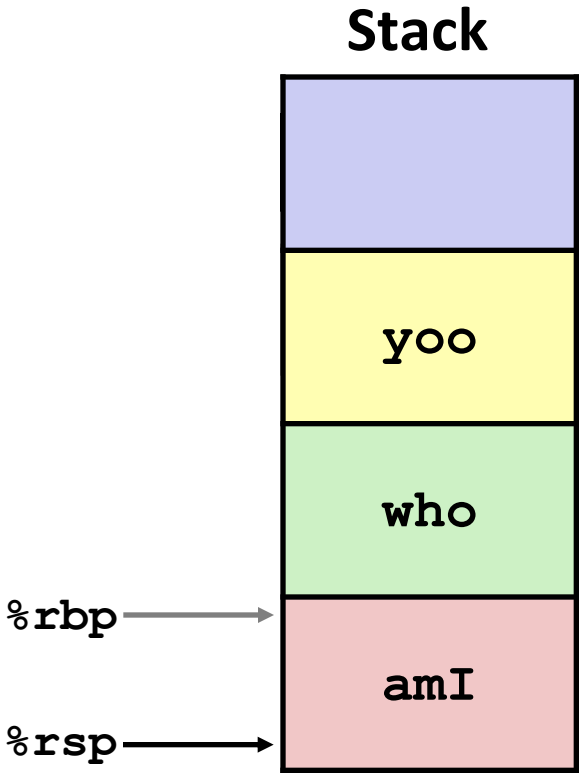


UPDF
WWW.UPDF.COM

Example

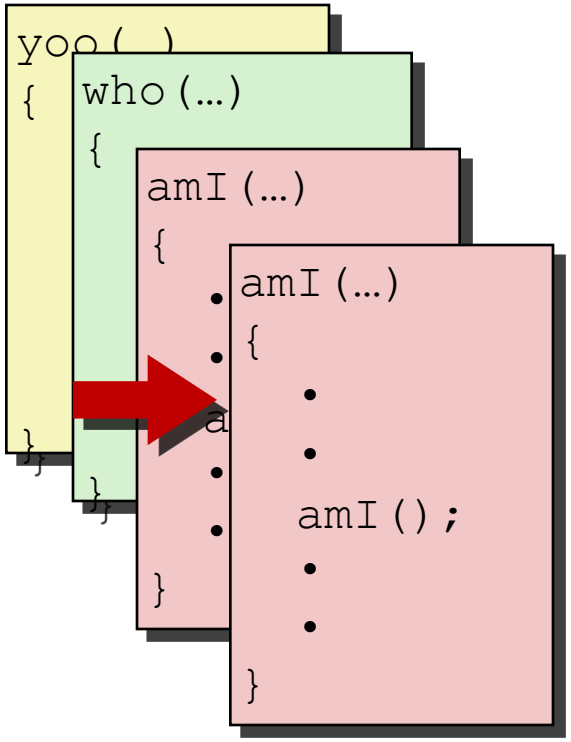


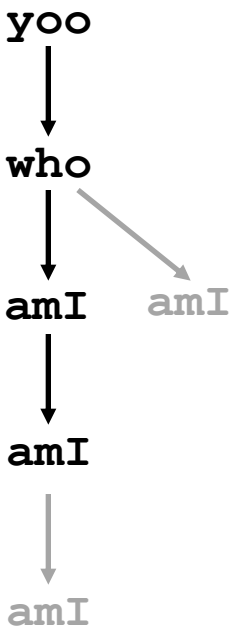


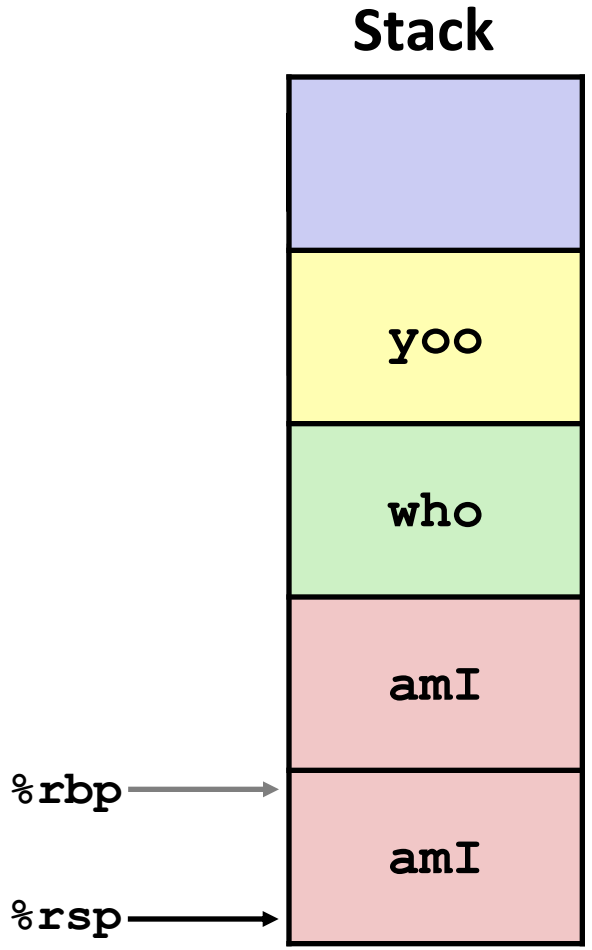


UPDF
WWW.UPDF.COM

Example

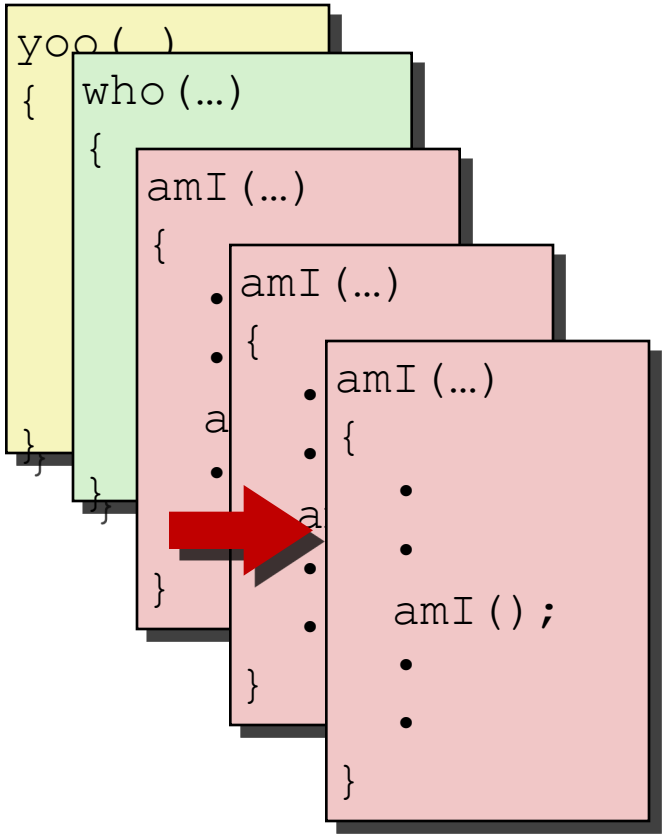


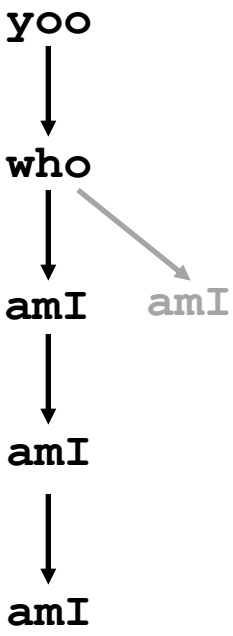


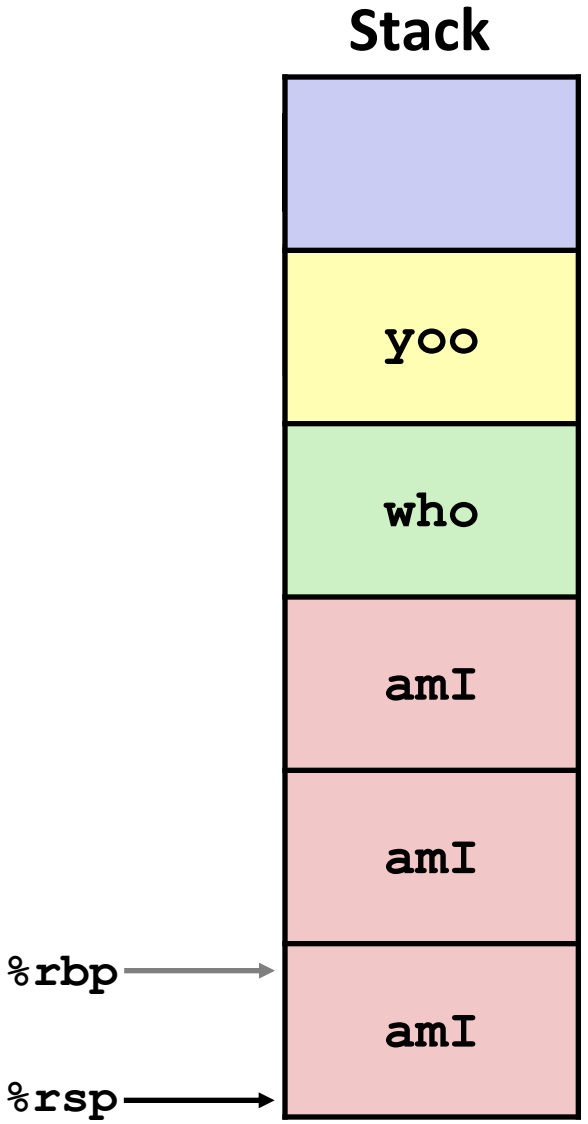


UPDF
WWW.UPDF.COM

Example

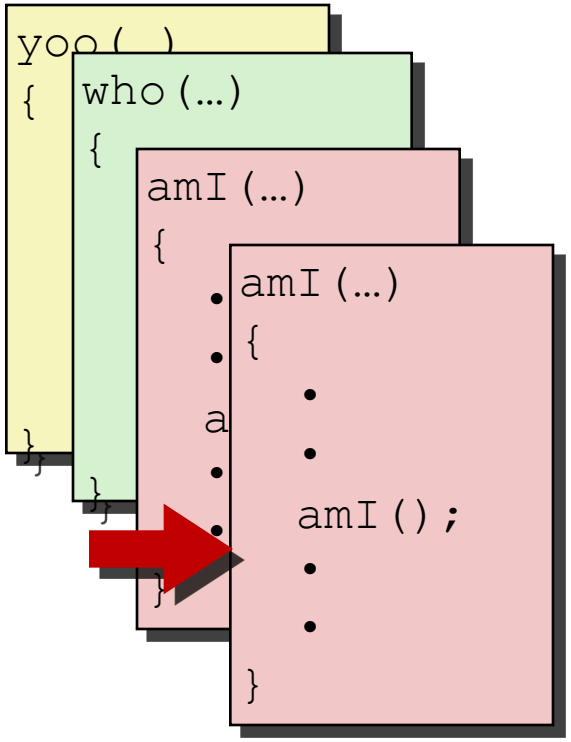


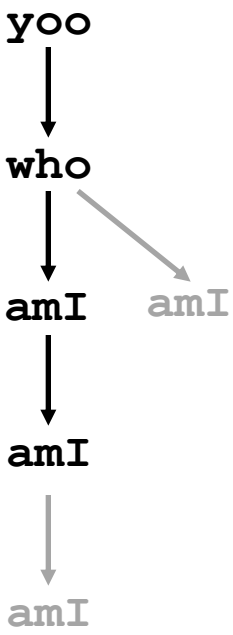


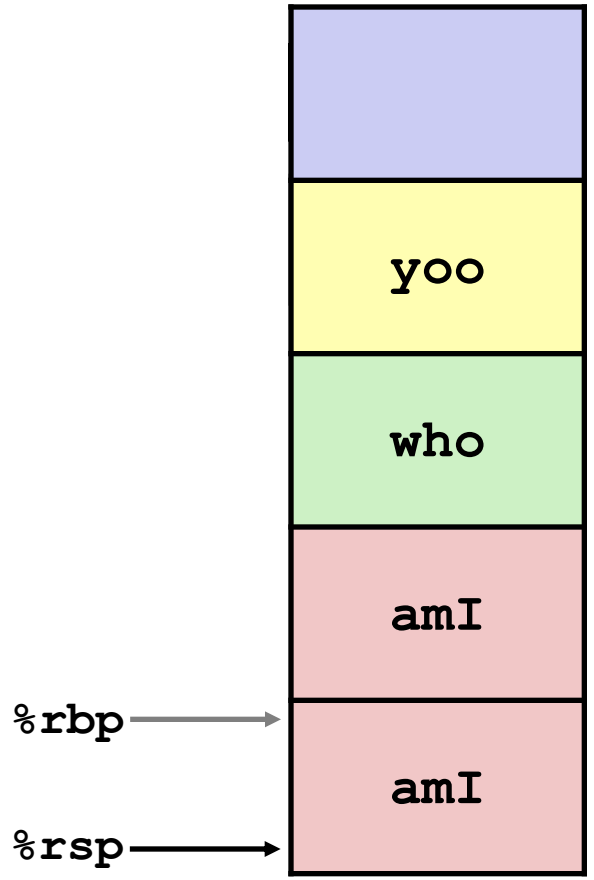


UPDF
WWW.UPDF.COM

Example

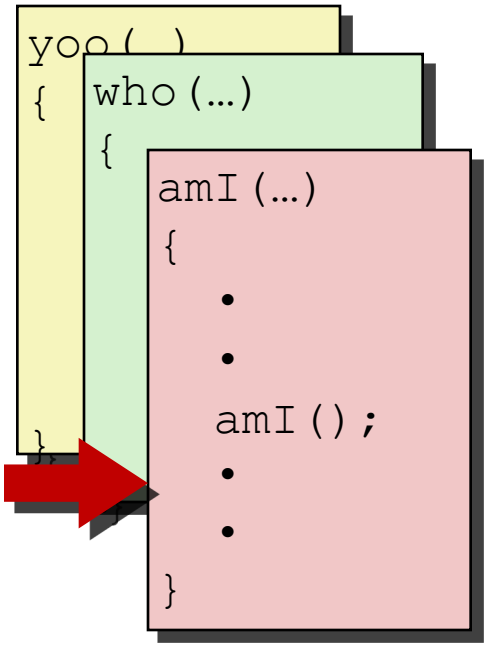


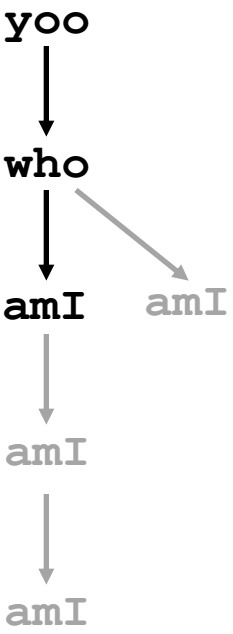


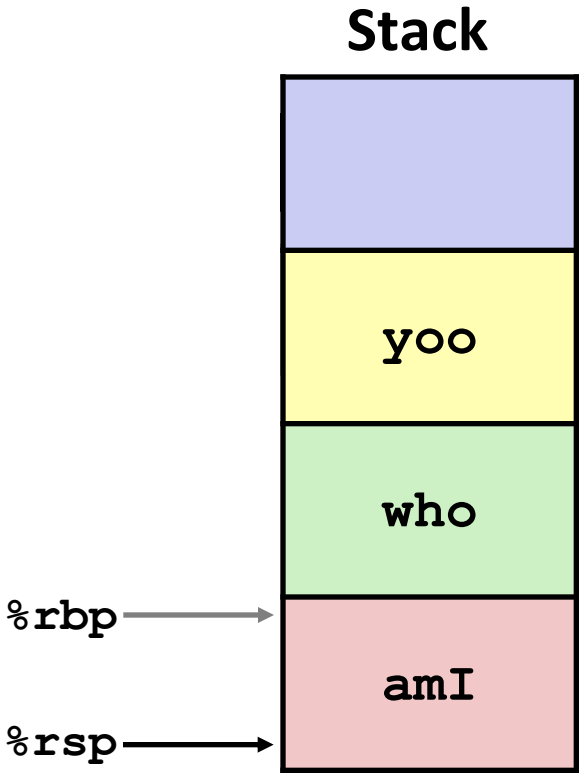


UPDF
WWW.UPDF.COM

Example

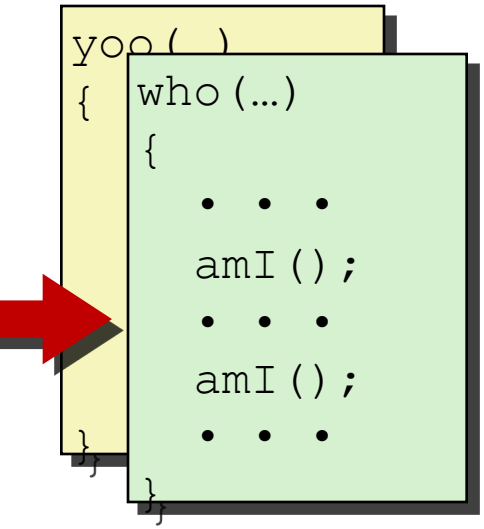


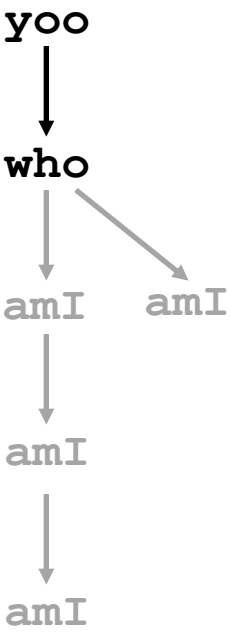


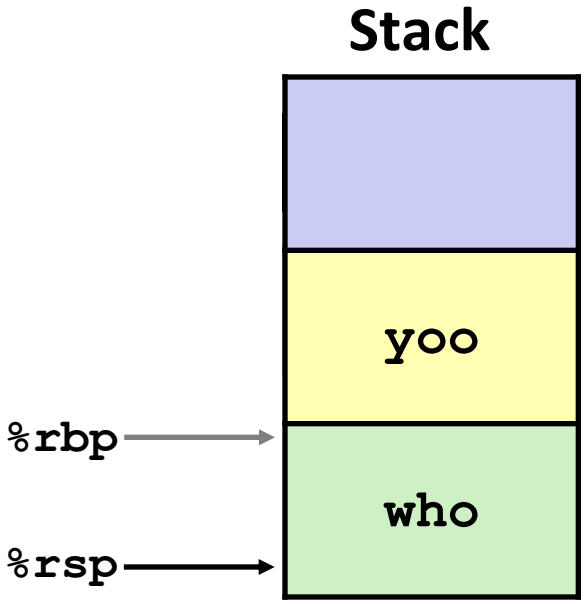


UPDF
WWW.UPDF.COM

Example

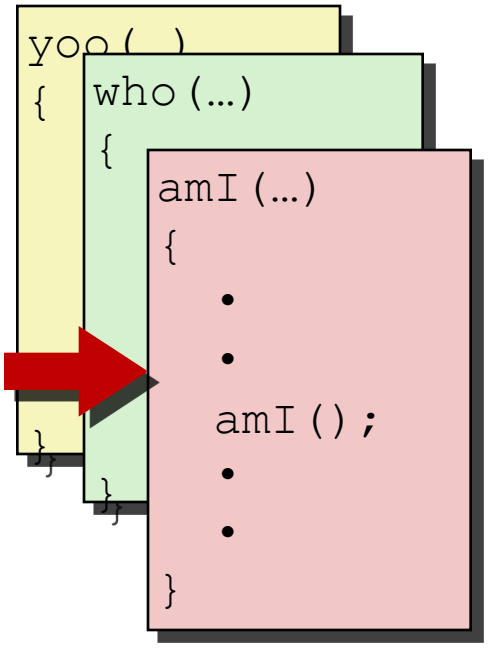


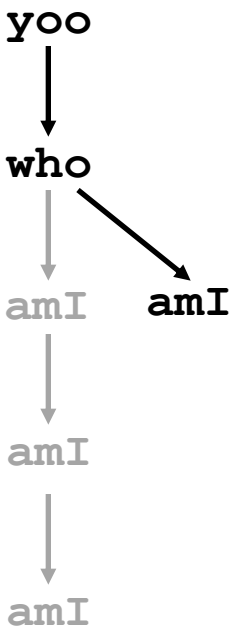


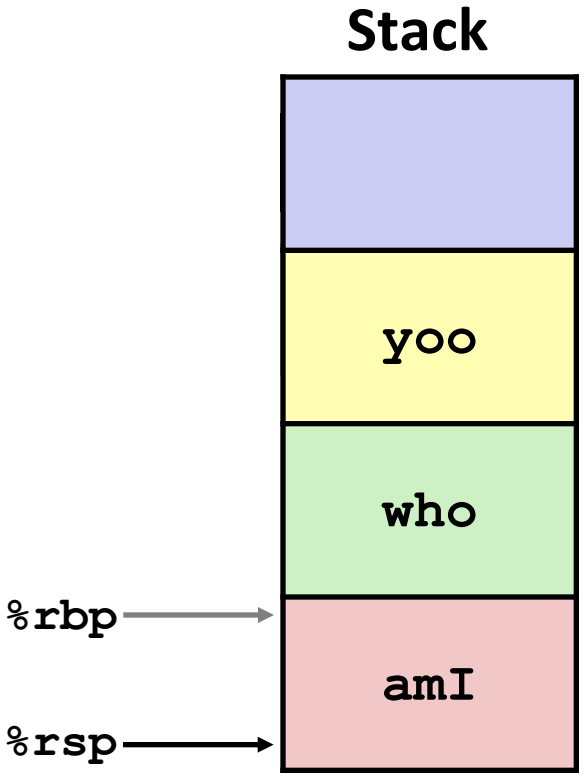


UPDF
WWW.UPDF.COM

Example



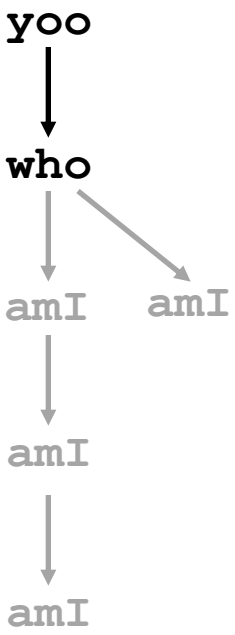


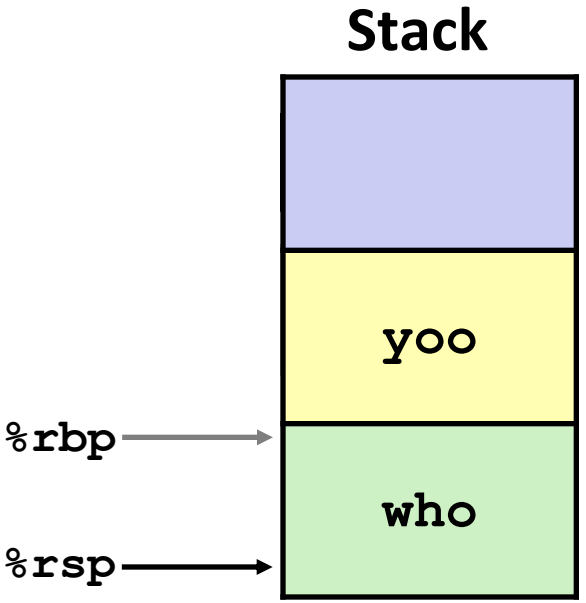


UPDF
WWW.UPDF.COM

Example

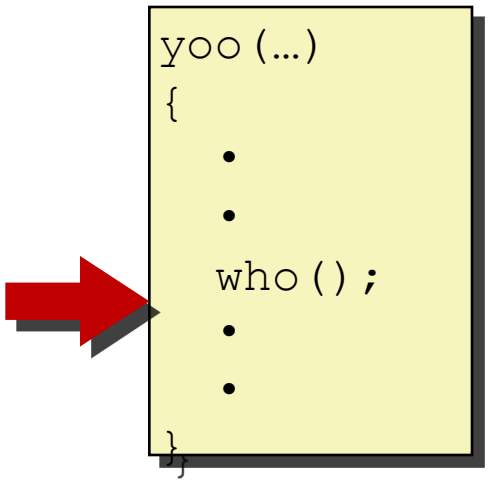
```
yoo( )
{
    who(…)
    {
       • • •
       amI();
       • • •
       amI();
       • • •
    }
```

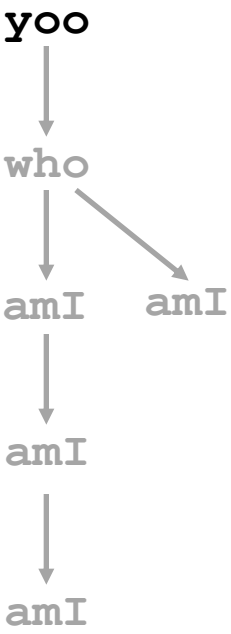


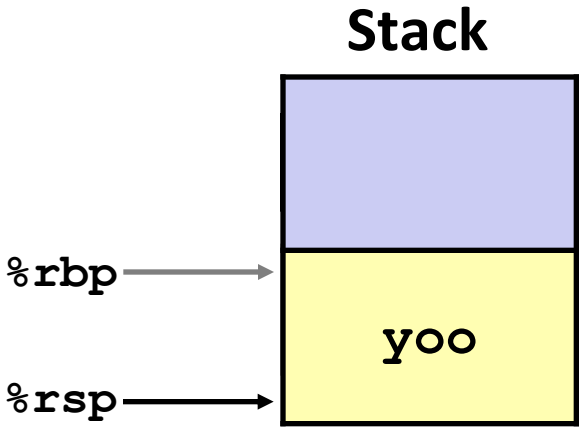


UPDF
WWW.UPDF.COM

Example







UPDF
WWW.UPDF.COM

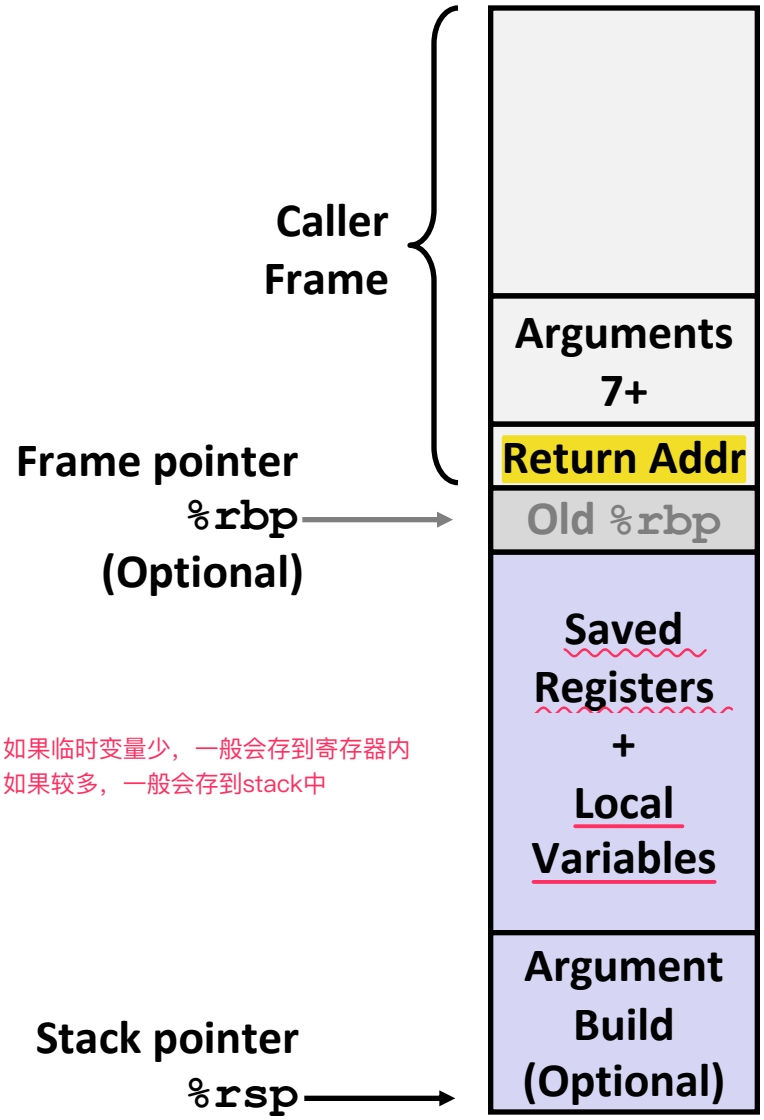
x86-64 Linux Stack Frame

Caller Stack Frame

- Arguments for this call
- Return address
  - Pushed by `call` instruction

Current Stack Frame

- Old frame pointer (optional)
- Saved register context
- Local variables
  If can't keep in registers
- "Argument build:"
  Parameters for function about to call



UPDF
WWW.UPDF.COM

Example: incr

%rdi

```
long incr(long *p, long val) {
    long x = *p;
    long y = x + val;
    *p = y;
    return x;
}
```

```
incr:
  movq    (%rdi), %rax
  addq    %rax, %rsi
  movq    %rsi, (%rdi)
  ret
```

Register	Use(s)
%rdi	Argument p
%rsi	Argument val, y
%rax	x, Return value

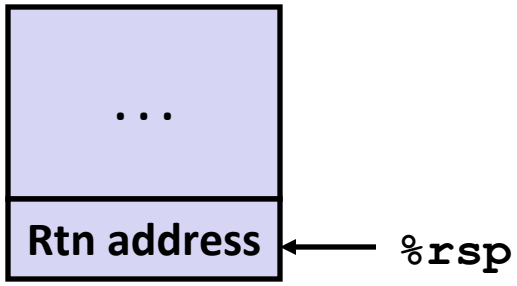
UPDF
WWW.UPDF.COM

Example: Calling `incr` #1

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return v1+v2;
}
```

此处v1不能存在寄存器中，因为这样没法取址

Initial Stack Structure




```
call_incr:
    subq    $16, %rsp
    movq    $15213, 8(%rsp)
    movl    $3000, %esi
    leaq    8(%rsp), %rdi
    call    incr
    addq    8(%rsp), %rax
    addq    $16, %rsp
    ret
```

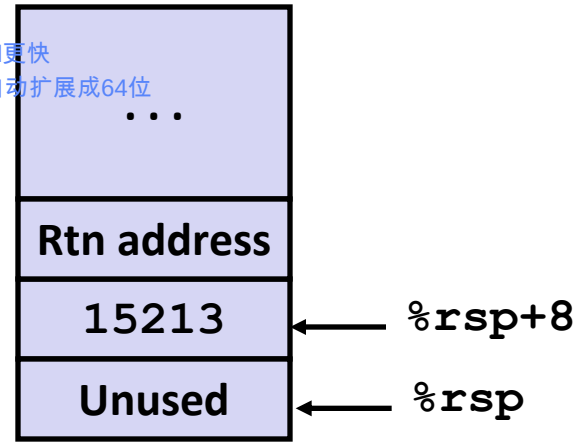
调用函数

内存对齐

32bytes已经够用
1）同等情况下，使用movl更快
2）32位会在64位机器中自动扩展成64位

结束函数调用

Resulting Stack Structure



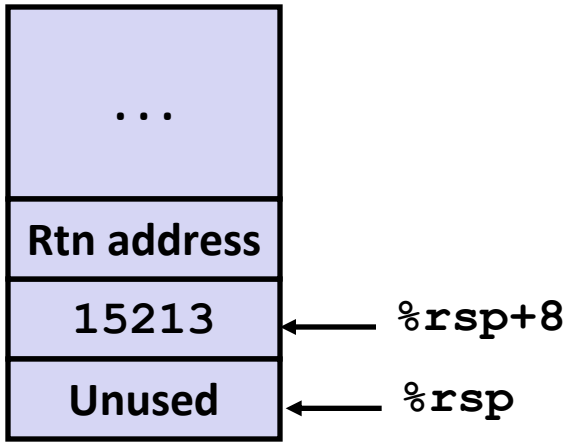
UPDF
WWW.UPDF.COM

Example: Calling `incr` #2

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return v1+v2;
}
```

```
call_incr:
  subq    $16, %rsp
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    8(%rsp), %rax
  addq    $16, %rsp
  ret
```

Stack Structure



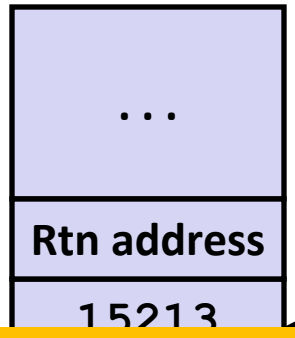
Register	Use(s)
%rdi	&v1
%rsi	3000

UPDF
WWW.UPDF.COM

Example: Calling `incr` #2

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return v1+v2;
}
```

Stack Structure



Aside 1: **`movl $3000, %esi`**

- Remember, movl -> %exx zeros out high order 32 bits.
- Why use movl instead of movq? 1 byte shorter.

```
ca
    movl    $3000, %esi
    leaq    8(%rsp), %rdi
    call    incr
    addq    8(%rsp), %rax
    addq    $16, %rsp
    ret
```

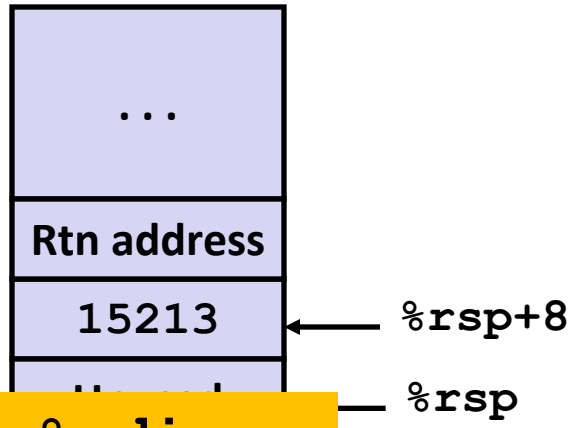
%rdi	&v1
%rsi	3000

UPDF
WWW.UPDF.COM

Example: Calling `incr` #2

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return v1+v2;
}
```

Stack Structure




```
ca
   $
   r
   r
   leaq    8(%rsp), %rdi
   call    incr
   addq    8(%rsp), %rax
   addq    $16, %rsp
   ret
```

Aside 2: `leaq 8(%rsp), %rdi`

- Computes %rsp+8
- Actually, used for what it is meant!

se(s)	
	v1
%rsi	3000

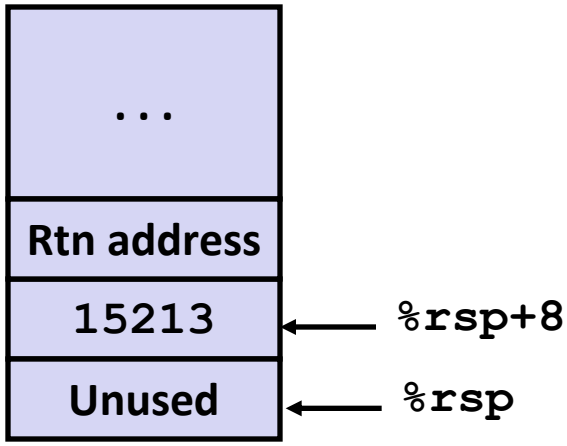
UPDF
WWW.UPDF.COM

Example: Calling `incr` #2

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return v1+v2;
}
```

```
call_incr:
  subq    $16, %rsp
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    8(%rsp), %rax
  addq    $16, %rsp
  ret
```

Stack Structure



Register	Use(s)
%rdi	&v1
%rsi	3000

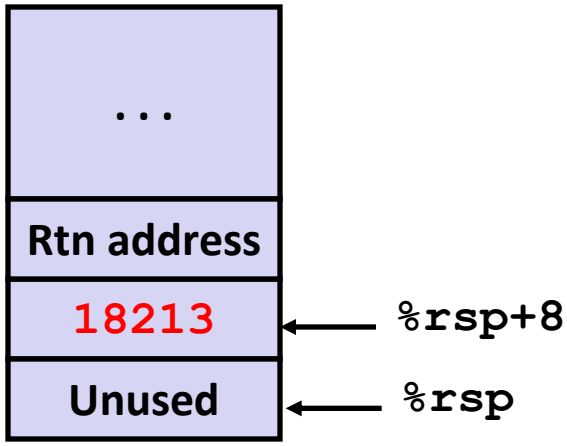
UPDF
WWW.UPDF.COM

Example: Calling `incr` #3

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return v1+v2;
}
```

```
call_incr:
  subq    $16, %rsp
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    8(%rsp), %rax
  addq    $16, %rsp
  ret
```

Stack Structure



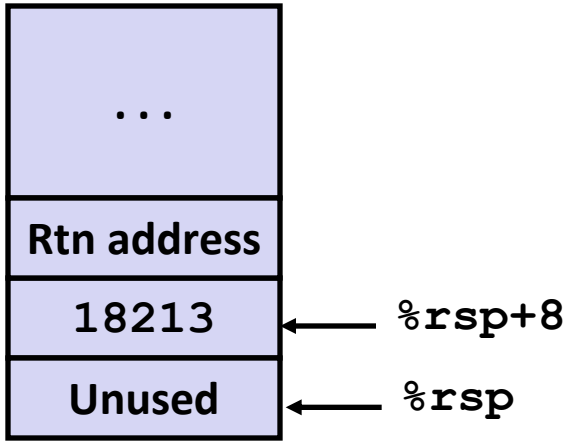
Register	Use(s)
%rdi	&v1
%rsi	3000

UPDF
WWW.UPDF.COM

Example: Calling `incr` #4

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return v1+v2;
}
```

Stack Structure




```
call_incr:
  subq    $16, %rsp
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    8(%rsp), %rax
  addq    $16, %rsp
  ret
```

Register	Use(s)
%rax	Return value

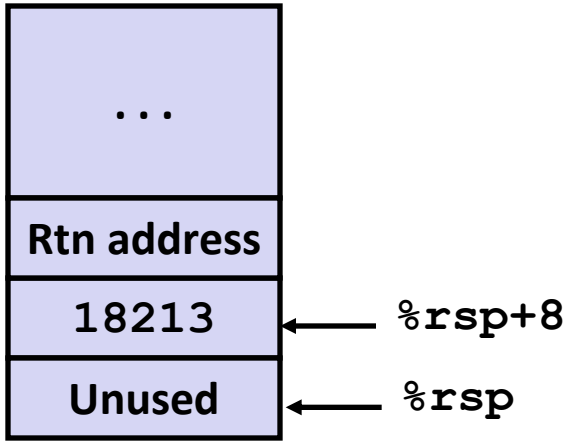
UPDF
WWW.UPDF.COM

Example: Calling `incr` #5a

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return v1+v2;
}
```

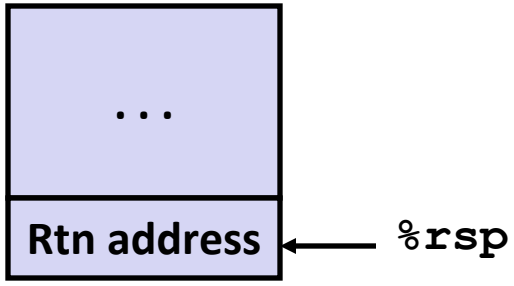
```
call_incr:
  subq    $16, %rsp
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    8(%rsp), %rax
  addq    $16, %rsp
  ret
```

Stack Structure



Register	Use(s)
%rax	Return value

Updated Stack Structure



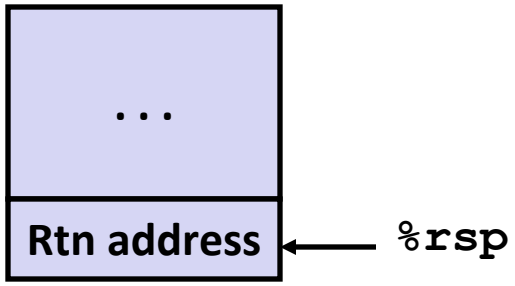
UPDF
WWW.UPDF.COM

Example: Calling `incr` #5b

```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return v1+v2;
}
```

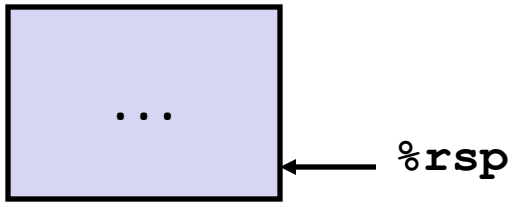
```
call_incr:
  subq    $16, %rsp
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    8(%rsp), %rax
  addq    $16, %rsp
  ret
```

Updated Stack Structure



Register	Use(s)
%rax	Return value

Final Stack Structure



UPDF
WWW.UPDF.COM

Register Saving Conventions

When procedure `yoo` calls `who`:

`yoo` is the ***caller*** 调用函数者

`who` is the ***callee*** 被调用函数者

Can register be used for temporary storage?

```
yoo:
        • • •
    movq $15213, %rdx
    call who
    addq %rdx, %rax
        • • •
    ret
```

```
who:
        • • •
        subq $18213, %rdx
        • • •
    ret
```

Contents of register **`%rdx`** overwritten by **`who`**

This could be trouble → something should be done!

Need some coordination

UPDF
WWW.UPDF.COM

Register Saving Conventions

When procedure `yoo` calls `who`:

`yoo` is the ***caller***

`who` is the ***callee***

Can register be used for temporary storage?

Conventions

"Caller Saved" (aka "Call-Clobbered")

Caller saves temporary values in its frame before the call

"Callee Saved" (aka "Call-Preserved")

Callee saves temporary values in its frame before using

Callee restores them before returning to caller

UPDF
WWW.UPDF.COM

x86-64 Linux Register Usage #1

%rax

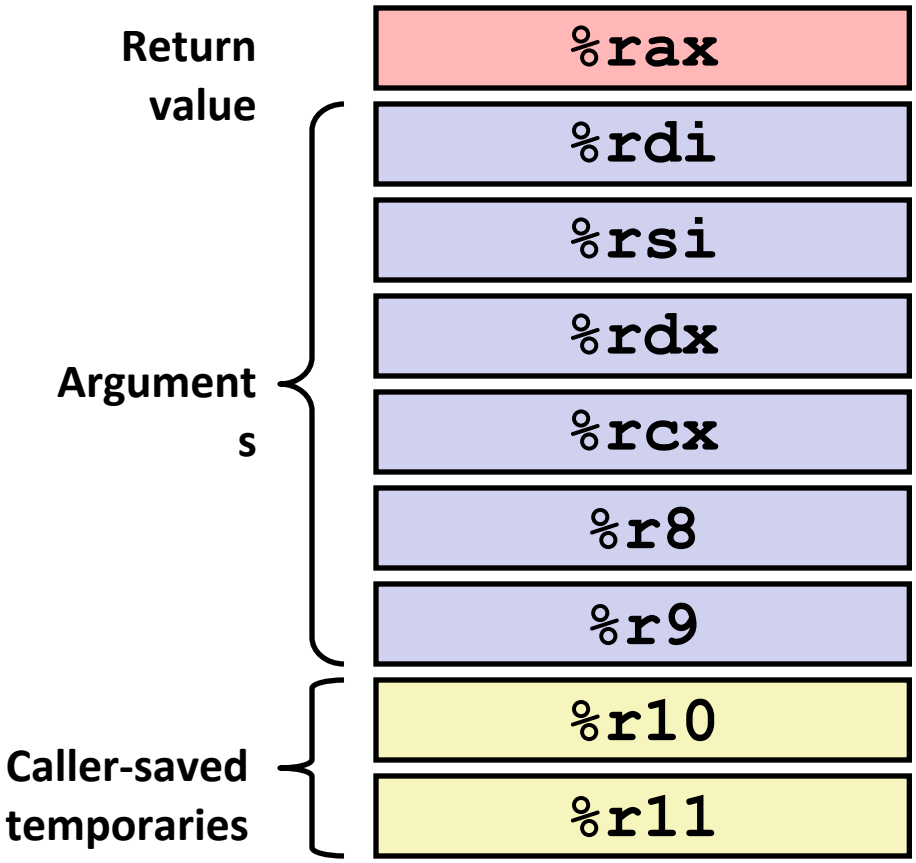
- Return value
- Also caller-saved
- Can be modified by procedure

%rdi, …, %r9

- Arguments
- Also caller-saved
- Can be modified by procedure

%r10, %r11

- Caller-saved
- Can be modified by procedure



UPDF
WWW.UPDF.COM

x86-64 Linux Register Usage #2

%rbx, %r12, %r13, %r14

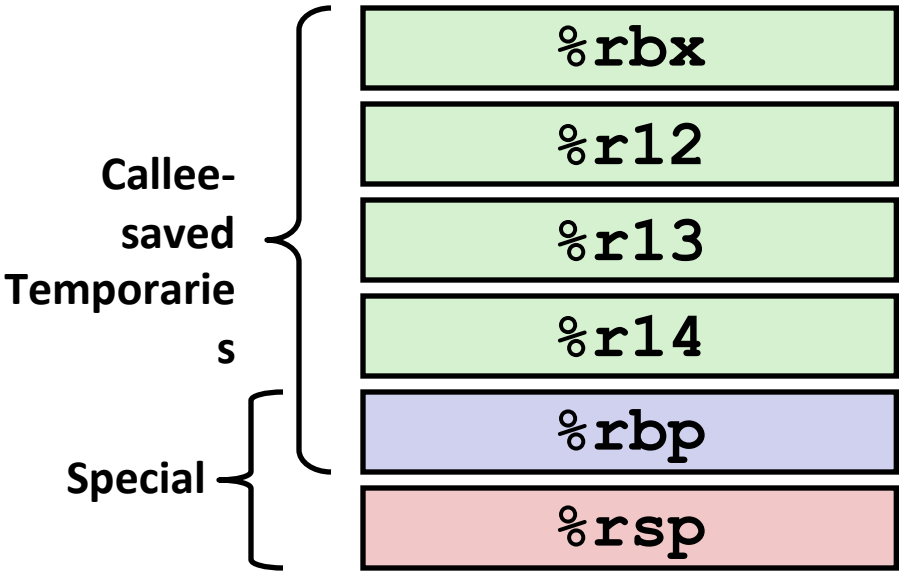
- Callee-saved
- Callee must save & restore

%rbp

- Callee-saved
- Callee must save & restore
- May be used as frame pointer
- Can mix & match

%rsp

- Special form of callee save
- Restored to original value upon exit from procedure

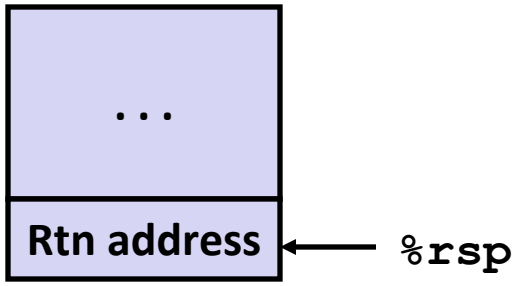


UPDF
WWW.UPDF.COM

Callee-Saved Example #1

```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

Initial Stack Structure



- X comes in register **%rdi**.
- We need **%rdi** for the call to incr.
- Where should be put x, so we can use it after the call to incr?

UPDF
WWW.UPDF.COM

Callee-Saved Example #2

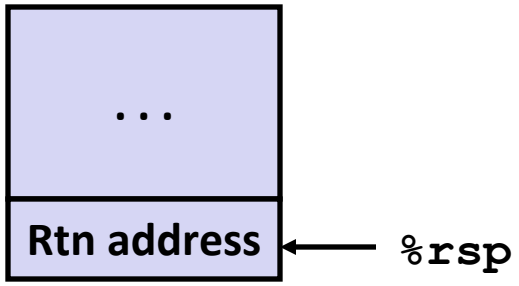
```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

```
call_incr2:
   pushq    %rbx
   subq     $16, %rsp
   movq     %rdi, %rbx
   movq     $15213, 8(%rsp)
   movl     $3000, %esi
   leaq     8(%rsp), %rdi
   call     incr
   addq     %rbx, %rax
   addq     $16, %rsp
   popq     %rbx
   ret
```

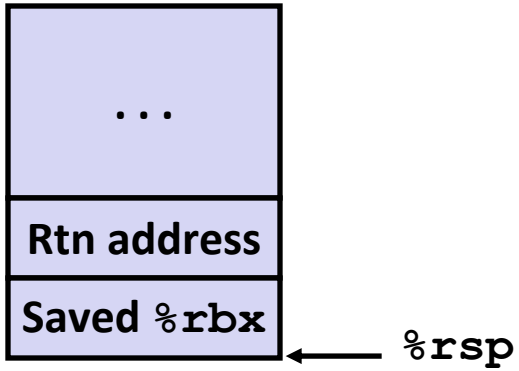
事实上是寄存器内原值的push，而不是“寄存器”的物理push

跟pushq同理

Initial Stack Structure



Resulting Stack Structure



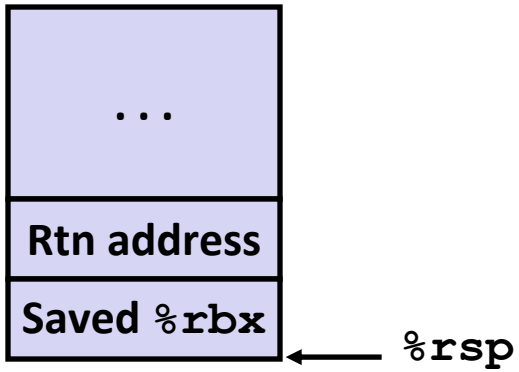
UPDF
WWW.UPDF.COM

Callee-Saved Example #3

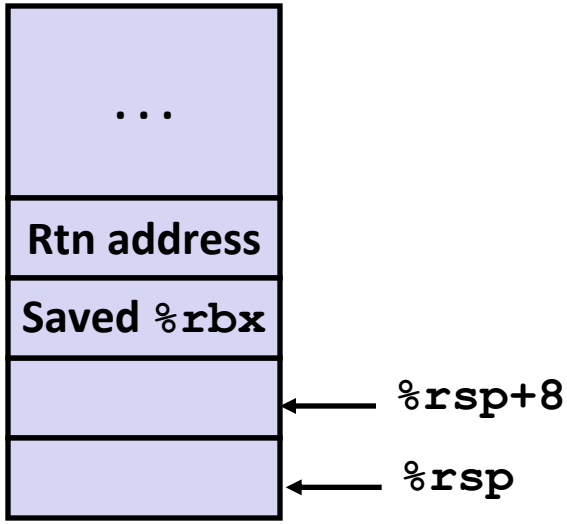
```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

```
call_incr2:
  pushq   %rbx
  subq    $16, %rsp
  movq    %rdi, %rbx
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    %rbx, %rax
  addq    $16, %rsp
  popq    %rbx
  ret
```

Initial Stack Structure



Resulting Stack Structure



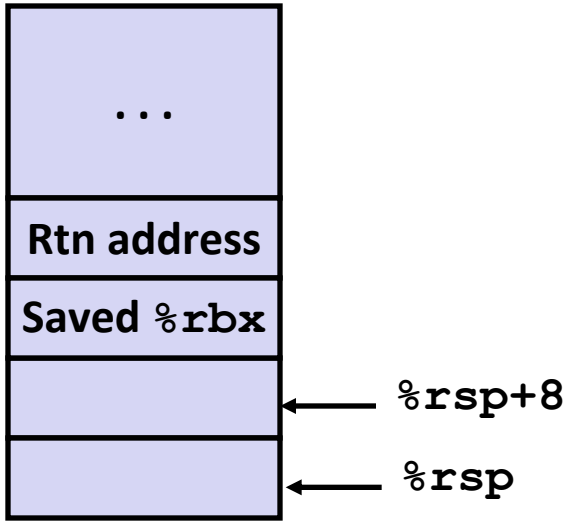
UPDF
WWW.UPDF.COM

Callee-Saved Example #4

```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

```
call_incr2:
  pushq   %rbx
  subq    $16, %rsp
  movq    %rdi, %rbx
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    %rbx, %rax
  addq    $16, %rsp
  popq    %rbx
  ret
```

Stack Structure



- X saved in **%rbx**.
- A callee saved register.

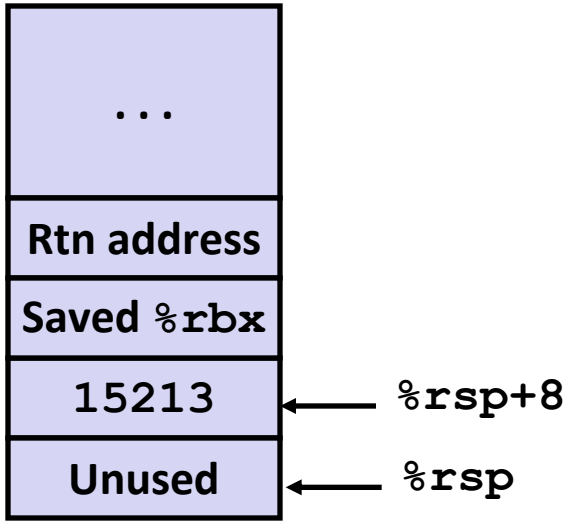
UPDF
WWW.UPDF.COM

Callee-Saved Example #5

```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

```
call_incr2:
  pushq   %rbx
  subq    $16, %rsp
  movq    %rdi, %rbx
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    %rbx, %rax
  addq    $16, %rsp
  popq    %rbx
  ret
```

Stack Structure



- X saved in **%rbx**.
- A callee saved register.

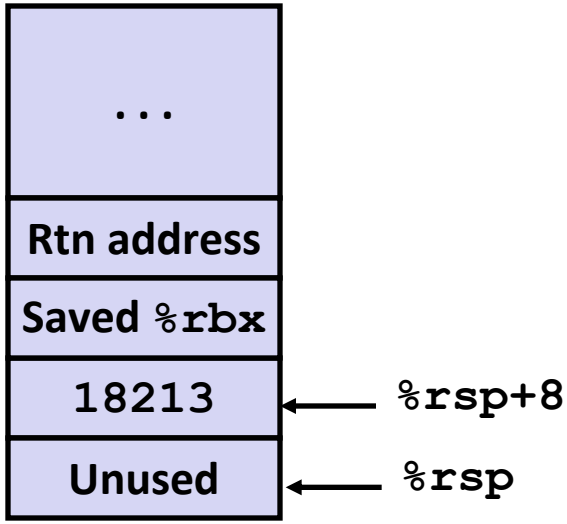
UPDF
WWW.UPDF.COM

Callee-Saved Example #6

```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

```
call_incr2:
  pushq   %rbx
  subq    $16, %rsp
  movq    %rdi, %rbx
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    %rbx, %rax
  addq    $16, %rsp
  popq    %rbx
  ret
```

Stack Structure



- X Is safe in **%rbx**
- Return result in **%rax**

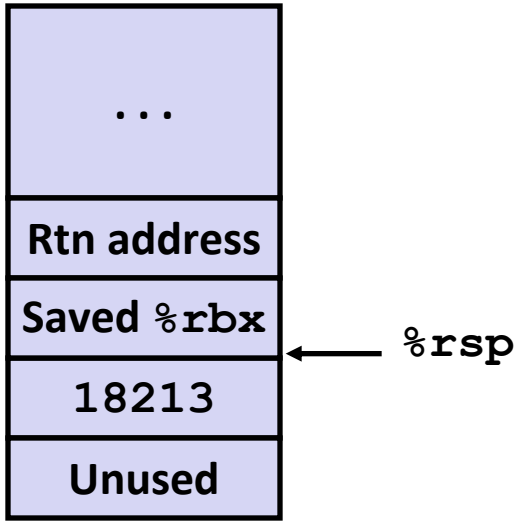
UPDF
WWW.UPDF.COM

Callee-Saved Example #7

```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

```
call_incr2:
  pushq   %rbx
  subq    $16, %rsp
  movq    %rdi, %rbx
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    %rbx, %rax
  addq    $16, %rsp
  popq    %rbx
  ret
```

Stack Structure



- Return result in **%rax**

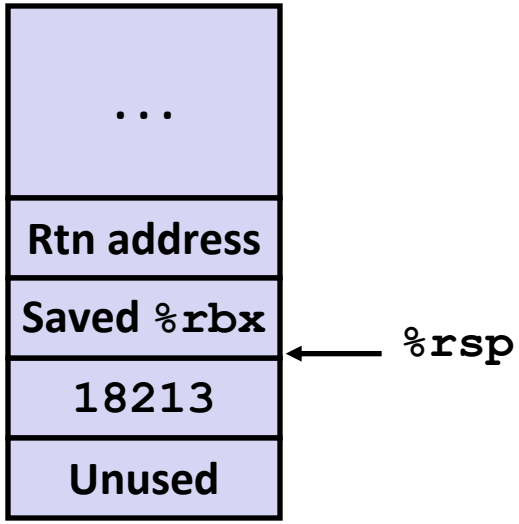
UPDF
WWW.UPDF.COM

Callee-Saved Example #8

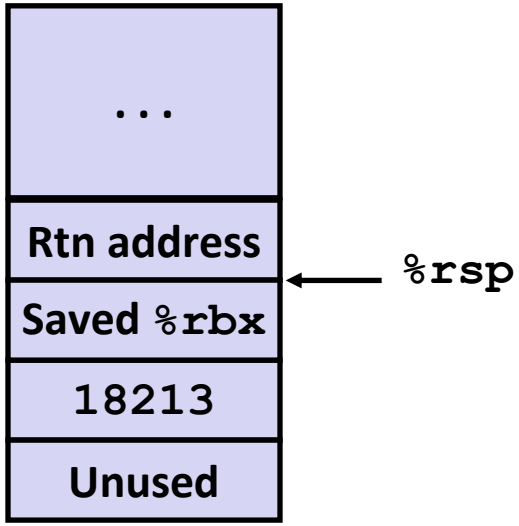
```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

```
call_incr2:
  pushq   %rbx
  subq    $16, %rsp
  movq    %rdi, %rbx
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    %rbx, %rax
  addq    $16, %rsp
  popq    %rbx
  ret
```

Initial Stack Structure



final Stack Structure



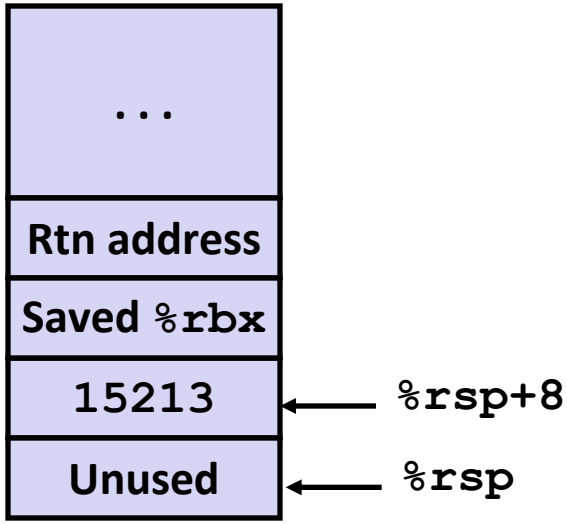
UPDF
WWW.UPDF.COM

Callee-Saved Example #2

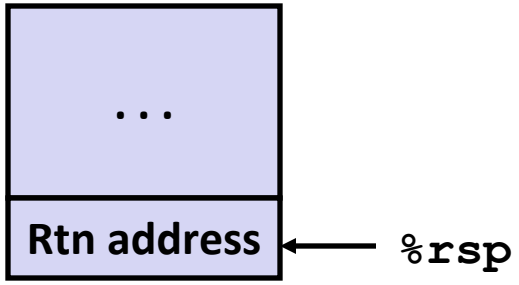
```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

```
call_incr2:
  pushq   %rbx
  subq    $16, %rsp
  movq    %rdi, %rbx
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    %rbx, %rax
  addq    $16, %rsp
  popq    %rbx
  ret
```

Resulting Stack Structure



Pre-return Stack Structure



UPDF
WWW.UPDF.COM

Today

Procedures

- Stack Structure
- Calling Conventions
  - Passing control
  - Passing data
  - Managing local data
- Illustration of Recursion

UPDF
WWW.UPDF.COM

Recursive Function

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0) 递归出口，否则会 stackoverflow
    return 0;
  else
    return (x & 1)
           + pcount_r(x >> 1);
}
```

LSB：The Least Significant Bit
MSB: The Most Significant Bit

```
pcount_r:
  movl    $0, %eax
  testq   %rdi, %rdi   test：X 本身
  je      .L6
  pushq   %rbx         callee-saved
  movq    %rdi, %rbx
  andl    $1, %ebx
  shrq    %rdi
  call    pcount_r
  addq    %rbx, %rax
  popq    %rbx
.L6:
  rep; ret
```

UPDF
WWW.UPDF.COM

Recursive Function Terminal Case

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
           + pcount_r(x >> 1);
}
```

```
pcount_r:
  movl    $0, %eax
  testq   %rdi, %rdi
  je      .L6
  pushq   %rbx
  movq    %rdi, %rbx
  andl    $1, %ebx
  shrq    %rdi
  call    pcount_r
  addq    %rbx, %rax
  popq    %rbx
.L6:
  rep; ret
```

Register	Use(s)	Type
%rdi	x	Argument
%rax	Return value	Return value

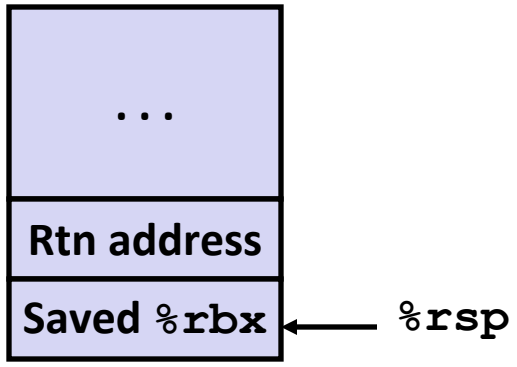
UPDF
WWW.UPDF.COM

Recursive Function Register Save

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
           + pcount_r(x >> 1);
}
```

```
pcount_r:
  movl    $0, %eax
  testq   %rdi, %rdi
  je      .L6
  pushq   %rbx
  movq    %rdi, %rbx
  andl    $1, %ebx
  shrq    %rdi
  call    pcount_r
  addq    %rbx, %rax
  popq    %rbx
.L6:
  rep; ret
```

Register	Use(s)	Type
%rdi	x	Argument



UPDF
WWW.UPDF.COM

Recursive Function Call Setup

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
           + pcount_r(x >> 1);
}
```

```
pcount_r:
  movl    $0, %eax
  testq   %rdi, %rdi
  je      .L6
  pushq   %rbx
  movq    %rdi, %rbx
  andl    $1, %ebx
  shrq    %rdi
  call    pcount_r
  addq    %rbx, %rax
  popq    %rbx
.L6:
  rep; ret
```

Register	Use(s)	Type
%rdi	x >> 1	Rec. argument
%rbx	x & 1	Callee-saved

UPDF
WWW.UPDF.COM

Recursive Function Call

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
           + pcount_r(x >> 1);
}
```

```
pcount_r:
  movl    $0, %eax
  testq   %rdi, %rdi
  je      .L6
  pushq   %rbx
  movq    %rdi, %rbx
  andl    $1, %ebx
  shrq    %rdi
  call    pcount_r
  addq    %rbx, %rax
  popq    %rbx
.L6:
  rep; ret
```

Register	Use(s)	Type
%rbx	x & 1	Callee-saved
%rax	Recursive call return value	

UPDF
WWW.UPDF.COM

Recursive Function Result

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
           + pcount_r(x >> 1);
}
```

```
pcount_r:
  movl    $0, %eax
  testq   %rdi, %rdi
  je      .L6
  pushq   %rbx
  movq    %rdi, %rbx
  andl    $1, %ebx
  shrq    %rdi
  call    pcount_r
  addq    %rbx, %rax
  popq    %rbx
.L6:
  rep; ret
```

Register	Use(s)	Type
%rbx	x & 1	Callee-saved
%rax	Return value	

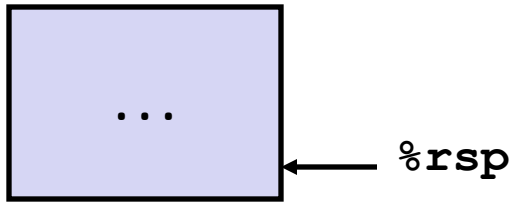
UPDF
WWW.UPDF.COM

Recursive Function Completion

```
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
           + pcount_r(x >> 1);
}
```

```
pcount_r:
   movl     $0, %eax
   testq    %rdi, %rdi
   je       .L6
   pushq    %rbx
   movq     %rdi, %rbx
   andl     $1, %ebx
   shrq     %rdi
   call     pcount_r
   addq     %rbx, %rax
   popq     %rbx
.L6:
   rep; ret
```

Register	Use(s)	Type
%rax	Return value	Return value



UPDF
WWW.UPDF.COM

Observations About Recursion

Handled Without Special Consideration

Stack frames mean that each function call has private storage

- Saved registers & local variables
- Saved return pointer

Register saving conventions prevent one function call from corrupting another's data called-saved & callee-saved

- Unless the C code explicitly does so (e.g., buffer overflow in Lecture 9)

Stack discipline follows call / return pattern

- If P calls Q, then Q returns before P
- Last-In, First-Out

Also works for mutual recursion

P calls Q; Q calls P

UPDF
WWW.UPDF.COM

x86-64: Procedure Summary

Important Points

Stack is the right data structure for procedure call/return

If P calls Q, then Q returns before P

Recursion (& mutual recursion) handled by normal calling conventions

Can safely store values in local stack frame and in callee-saved registers

Put function arguments at top of stack

Result return in `%rax`

Pointers are addresses of values

On stack or global