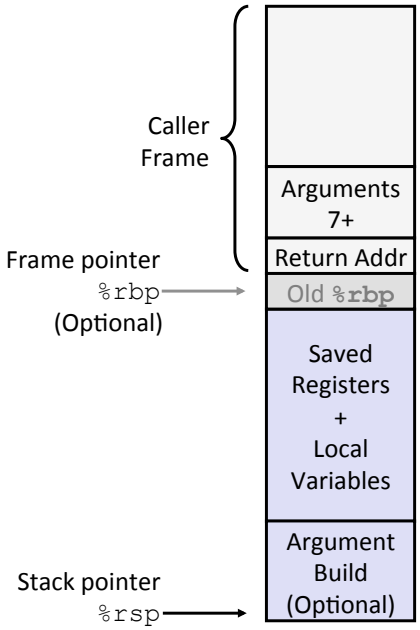
Stack Frame Components



Stack Example: incr

```
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
```

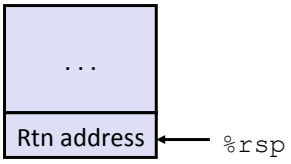
Register	Use(s)
%rdi	Argument p
%rsi	Argument val, y
%rax	x, Return value

Using the Stack (1)

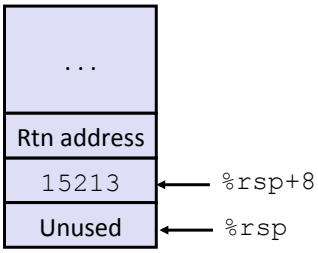
```
long call_incr() {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return v1+v2;
}
```

```
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
```

Initial Stack Structure



Resulting Stack Structure

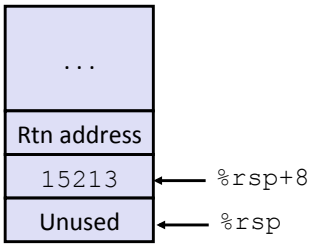


Using the Stack (2)

```
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



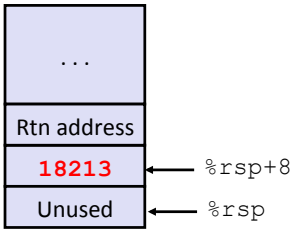
Register	Use(s)
%rdi	&v1
%rsi	3000

Using the Stack (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

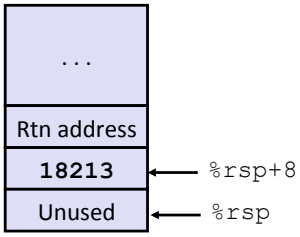
```
incr:
  movq    (%rdi), %rax
  addq    %rax, %rsi
  movq    %rsi, (%rdi)
  ret
```


Using the Stack (4)

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

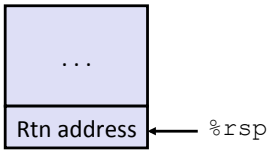
```
call_incr:
  subq    $16, %rsp
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    8(%rsp), %rax
  addq    $16, %rsp
  ret
```

Stack Structure



Register	Use(s)
%rax	Return value

Updated Stack Structure

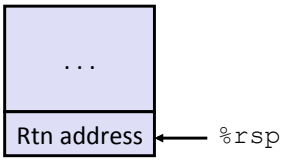


Using the Stack (5)

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

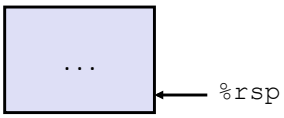
```
call_incr:
  subq    $16, %rsp
  movq    $15213, 8(%rsp)
  movl    $3000, %esi
  leaq    8(%rsp), %rdi
  call    incr
  addq    8(%rsp), %rax
  addq    $16, %rsp
  ret
```

Updated Stack Structure

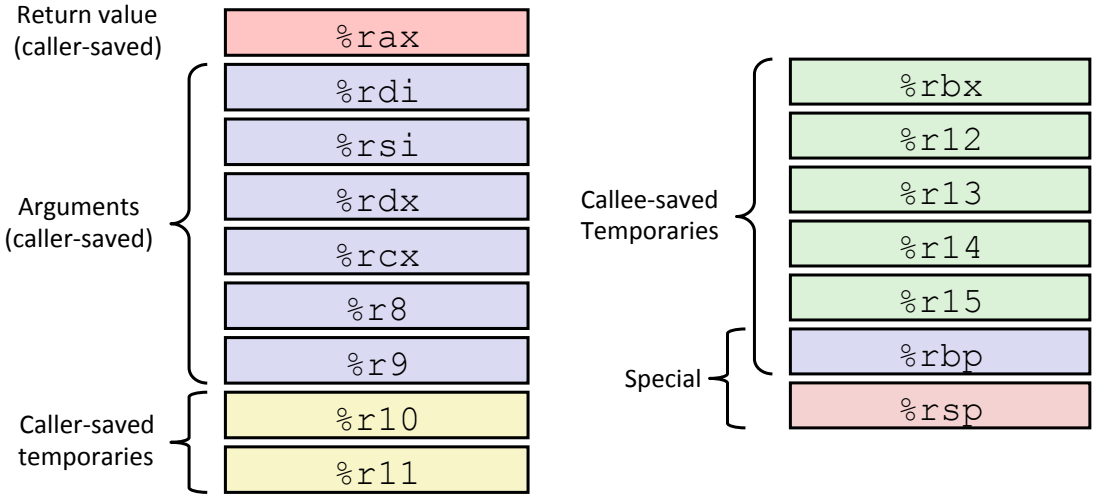


Register	Use(s)
%rax	Return value

Final Stack Structure



Register Conventions

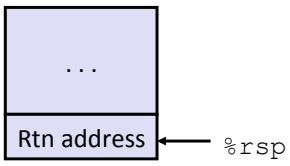


Callee-Saved Example (1)

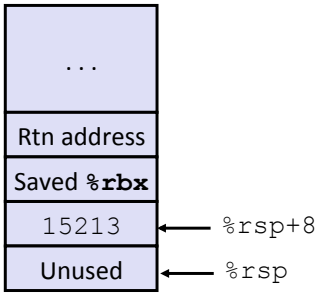
```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

```
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

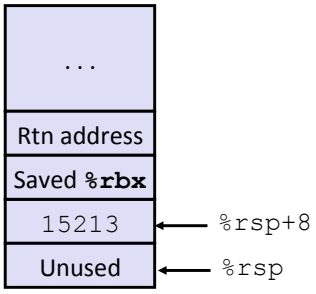


Callee-Saved Example (2)

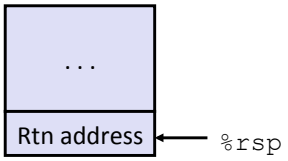
```
long call_incr2(long x) {
    long v1 = 15213;
    long v2 = incr(&v1, 3000);
    return x+v2;
}
```

```
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
```

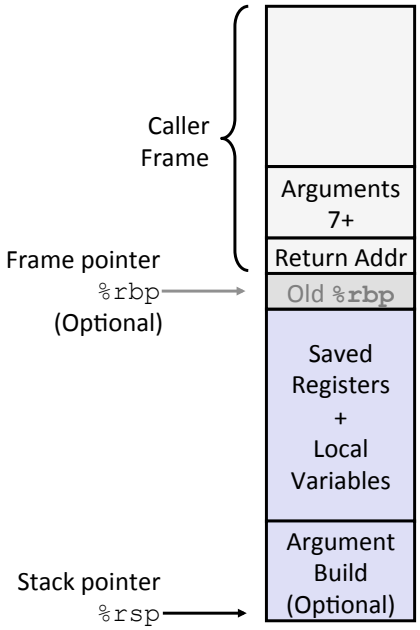
Resulting Stack Structure



Pre-return Stack Structure



Stack Frame Components



Recursion Example

```
/* 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
```

Recursion Base 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
```

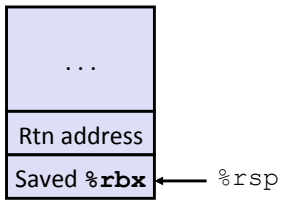
Register	Use(s)	Type
%rdi	x	Argument
%rax	Return value	Return value

Recursion 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



Recursion 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

Recursive 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	

Recursion 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
```

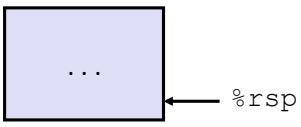
Register	Use(s)	Type
%rbx	x & 1	Callee-saved
%rax	Return value	

Recursion Result

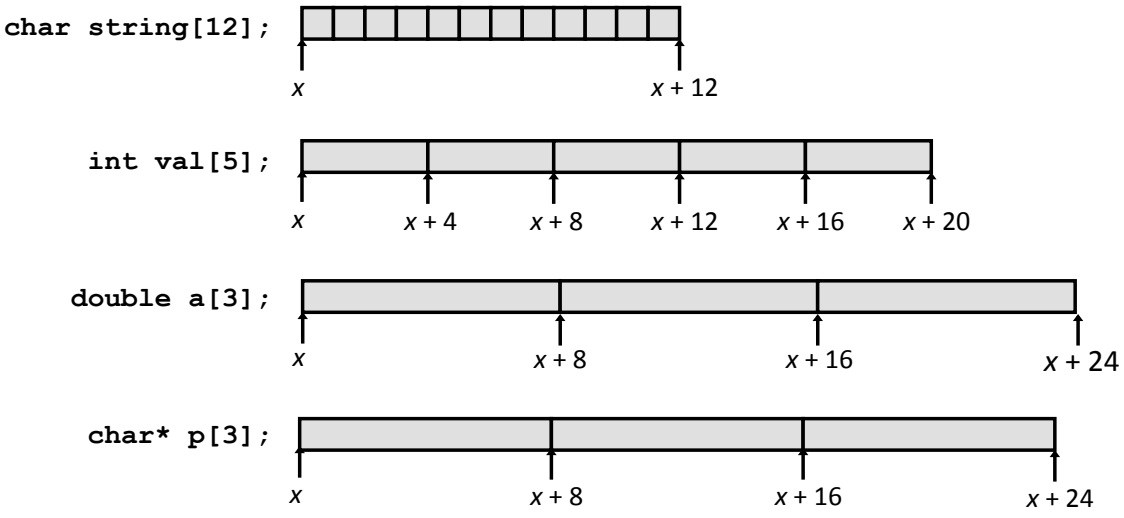
```
/* Recursive popcount */
long pcount_r(unsigned long x) {
  if (x == 0)
    return 0;
  else
    return (x & 1)
           + pcount_r(x >> 1);
}
```

```
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
```

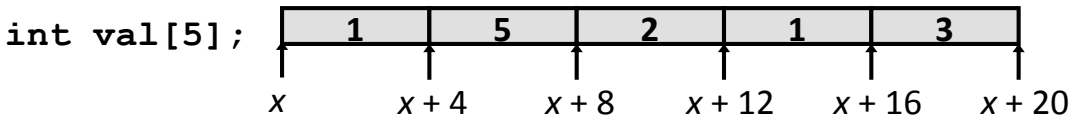
Register	Use(s)	Type
%rax	Return value	Return value



Array Allocation



Arrays as Pointers



Array Access

```
int get_val(int a[], int i) {
  return a[i];
}
```

```
  # %rdi = a
  # %rsi = i
movl (%rdi,%rsi,4), %eax  # a[i]
```

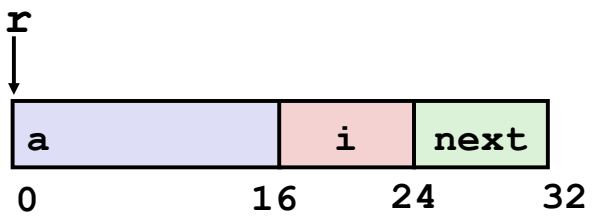

Array Looping

```
void inc5(int a[]) {
  size_t i;
  for (i = 0; i < 5; i++)
    a[i]++;
}
```

```
  # %rdi = a
  movl    $0, %eax          #   i = 0
  jmp     .L3               #   goto middle
.L4:                        # loop:
  addl    $1, (%rdi,%rax,4) #   a[i]++
  addq    $1, %rax          #   i++
.L3:                        # middle
  cmpq    $4, %rax          #   i:4
  jbe     .L4               #   if <=, goto loop
  rep; ret
```

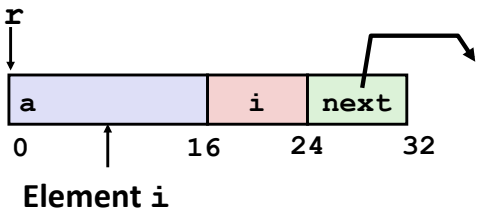
Structures

```
struct rec {
    int a[4];
    size_t i;
    struct rec* next;
};
```



Linked List Example

```
struct rec {
    int a[4];
    int i;
    struct rec* next;
};
```



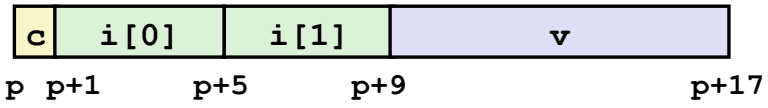

```
void set_val
  (struct rec* r, int val)
{
  while (r) {
    int i = r->i;
    r->a[i] = val;
    r = r->next;
  }
}
```

Register	Value
%rdi	r
%rsi	val

```
.L11:                           # loop:
  movslq  16(%rdi), %rax        #   i = M[r+16]
  movl    %esi, (%rdi,%rax,4)   #   M[r+4*i] = val
  movq    24(%rdi), %rdi        #   r = M[r+24]
  testq   %rdi, %rdi            #   Test r
  jne     .L11                  #   if !=0 goto loop
```

Data Alignment

Unaligned Data




```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```

Aligned Data