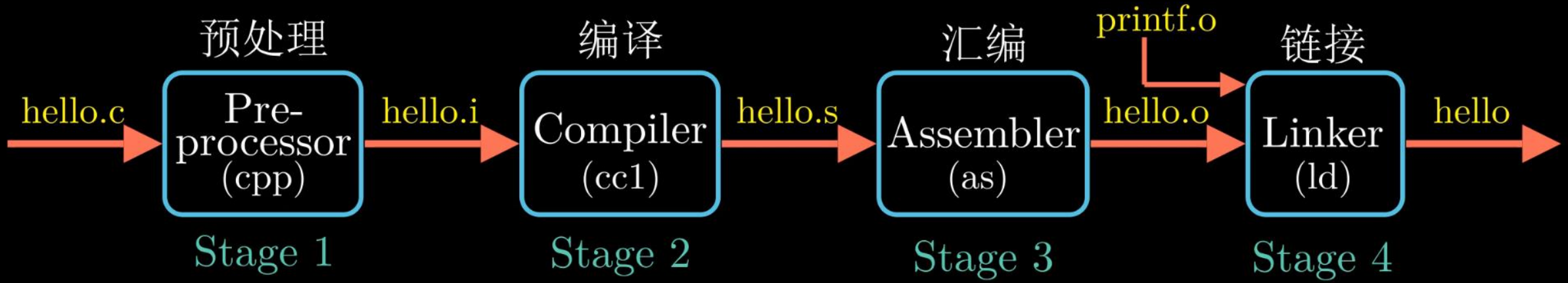
Computer Systems

A Programmer's Perspective

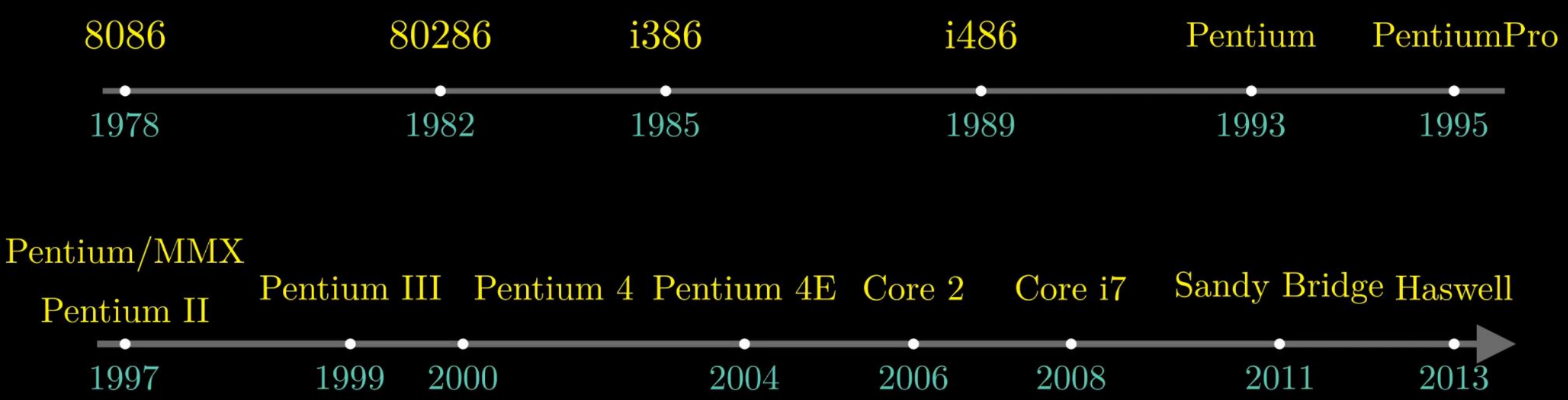
Chapter 3: Machine-Level Representation of Programs

(程序的机器级表示)

The Compilation System



Historical Perspective



Code File

main.c

```c
#include <stdio.h>

void mulstore(long, long, long *);

int main() {
    long d;
    multstore(2, 3, &d);
    printf("2 * 3 --> %ld\n", d);
    return 0;
}

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

mstore.c

```c
long mult2(long, long);

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

```
linux> gcc -Og -o prog main.c mstore.c
```

Code File

```
long mult2(long, long);

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

```
linux> gcc -Og -S mstore.c
```

Code File

```
long mult2(long, long);

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

```
linux> gcc -Og -S mstore.c
```

```
    .file "mstore.c"
    .text
    .globl multstore
    .type multstore, @function
multstore:
.LFB0:
    .cfi_startproc
    pushq   %rbx
    movq    %rdx, %rbx
    call    mult2
    movq    %rax, (%rbx)
    popq    %rbx
    ret
    ...
```

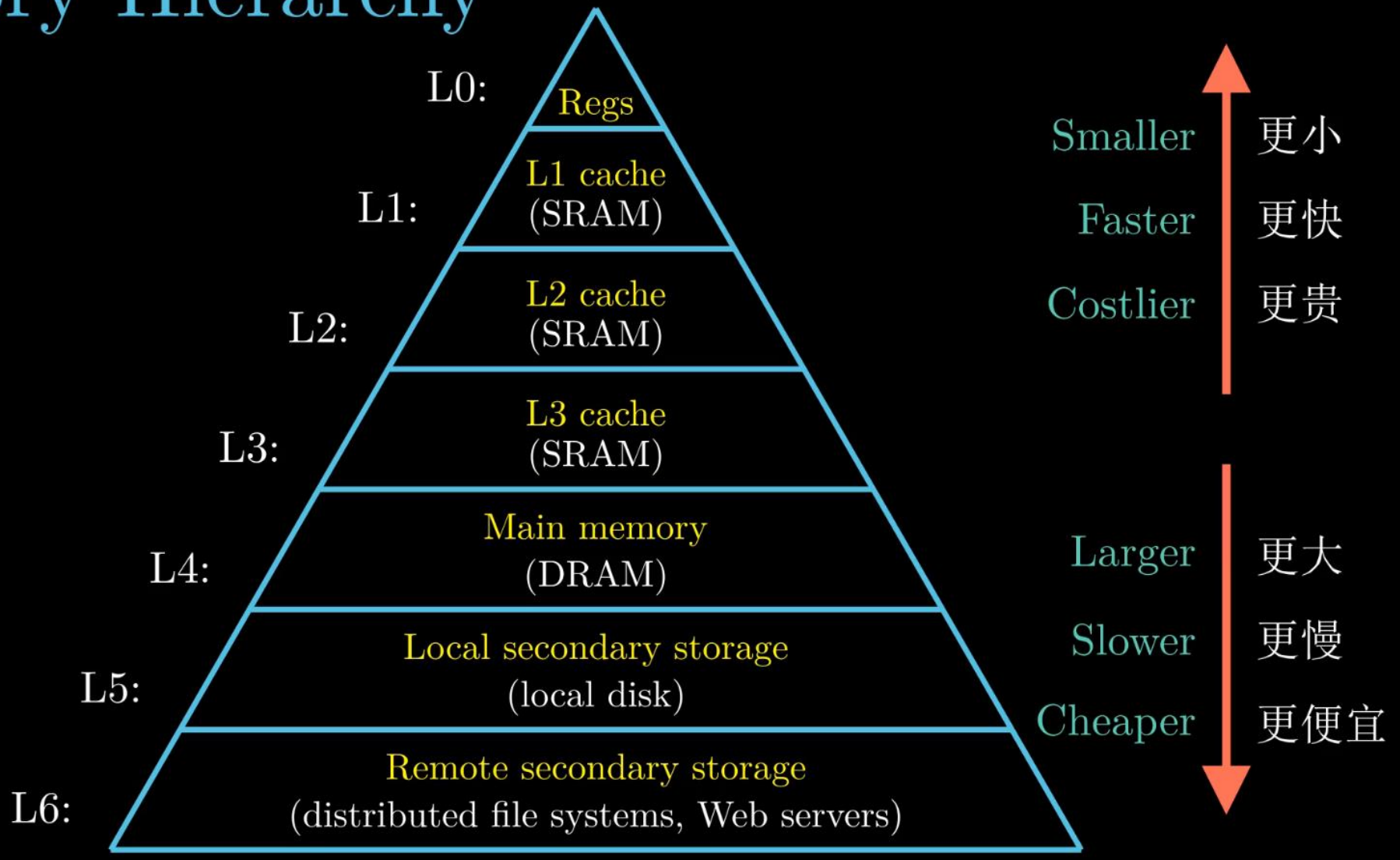
Code File

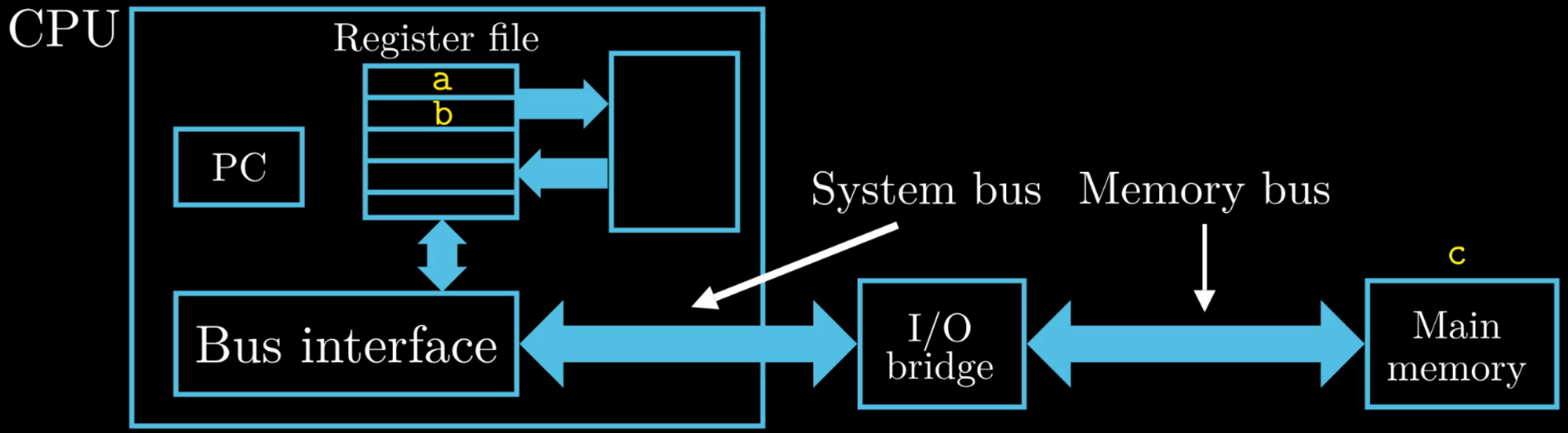
```
long mult2(long, long);

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

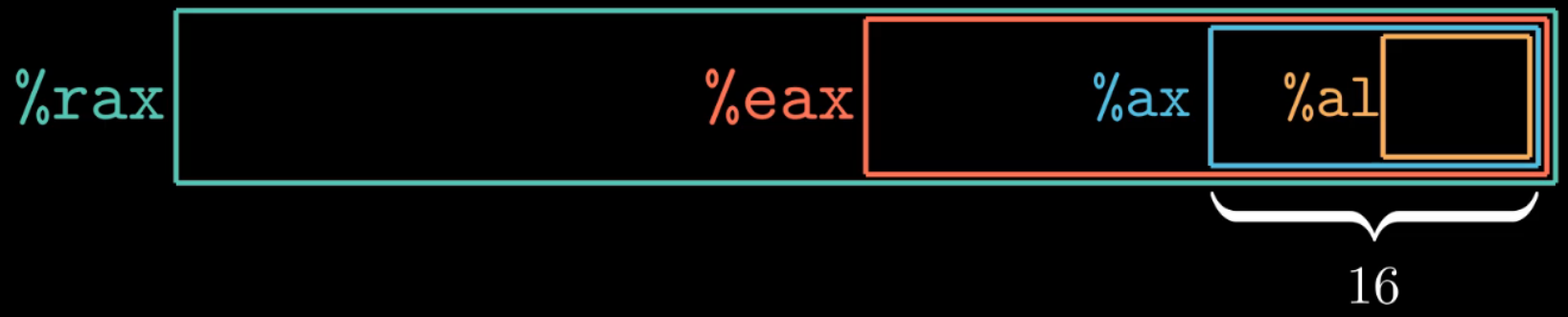
```
linux> gcc -Og -S mstore.c
```

```
multstore:
    pushq   %rbx
    movq    %rdx, %rbx
    call    mult2
    movq    %rax, (%rbx)
    popq    %rbx
    ret
```

Memory Hierarchy



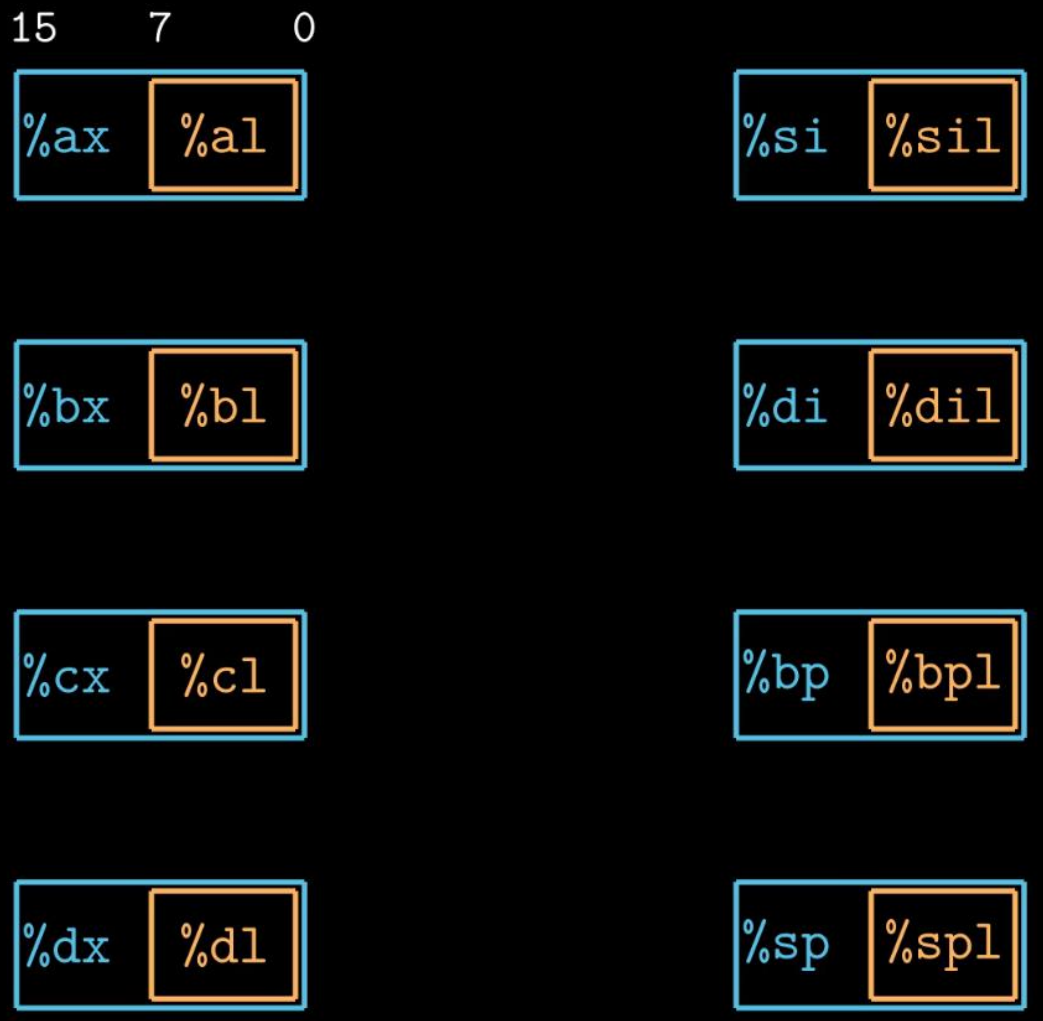

CPU
Register file
a
b
PC
System bus
Memory bus
c
Bus interface
I/O bridge
Main memory


%rax
%eax
%ax
%al
16

Integer Register

%rax	%rbx	%rcx	%rdx
%rsi	%rdi	%rbp	%rsp
%r8	%r9	%r10	%r11
%r12	%r13	%r14	%r15

Register



Register


31
15
7
0
%eax
%ax
%al
%esi
%si
%sil
%ebx
%bx
%bl
%edi
%di
%dil
%ecx
%cx
%cl
%ebp
%bp
%bpl
%edx
%dx
%dl
%esp
%sp
%spl

Register


63
31
15
7
0
%rax
%eax
%ax
%al
%rsi
%esi
%si
%sil
%rbx
%ebx
%bx
%bl
%rdi
%edi
%di
%dil
%rcx
%ecx
%cx
%cl
%rbp
%ebp
%bp
%bpl
%rdx
%edx
%dx
%dl
%rsp
%esp
%sp
%spl

Register

63 0

%rax	Return value-Caller saved
%rbx	Callee saved
%rcx	Argument #4 - Caller saved
%rdx	Argument #3 - Caller saved
%rsi	Argument #2 - Caller saved
%rdi	Argument #1 - Caller saved
%rbp	Callee saved
%rsp	Stack pointer

Register

63 0

%r8	Argument #5 -	Caller saved
%r9	Argument #6 -	Caller saved
%r10		Caller saved
%r11		Caller saved

%r12	Callee saved
%r13	Callee saved
%r14	Callee saved
%r15	Callee saved

Caller-saved / Callee-saved Register

```
func_A:
    ...
    movq   $123, %rbx
    call   func_B
    addq   %rbx, %rax
    ...
    ret
```

Caller

```
    ...
    addq   $456 %rbx
    ...
    ret
```

Callee

Caller-saved / Callee-saved Register

```
func_A:
    ...
    movq   $123, %rbx
    save register rbx
    call   func_B
    restore register rbx
    addq   %rbx, %rax
    ...
    ret
```

```
func_B:
    ...
    addq   $456  %rbx
    ...
    ret
```

Callee

Caller-saved / Callee-saved Register

```
func_A:
    ...
    movq   $123, %rbx
    save register rbx
    call   func_B
    restore register rbx
    addq   %rbx, %rax
    ...
    ret
```

```
func_B:
    ...
    save register rbx
    addq   $456  %rbx
    restore register rbx
    ...
    ret
```

Integer Register

Callee saved:	%rbx, %rbp, %r12, %r13, %r14, %15
Caller saved:	%r10, %r11
	%rax
	%rdi, %rsi, %rdx, %rcx, %r8, %r9

Instruction

Operation code (操作码)	Operands (操作数)
`movq`	`(%rdi), %rax`
`addq`	`$8, %rsx`
`subq`	`%rdi, %rax`
`xorq`	`%rsi, %rdi`
`ret`	

Instruction

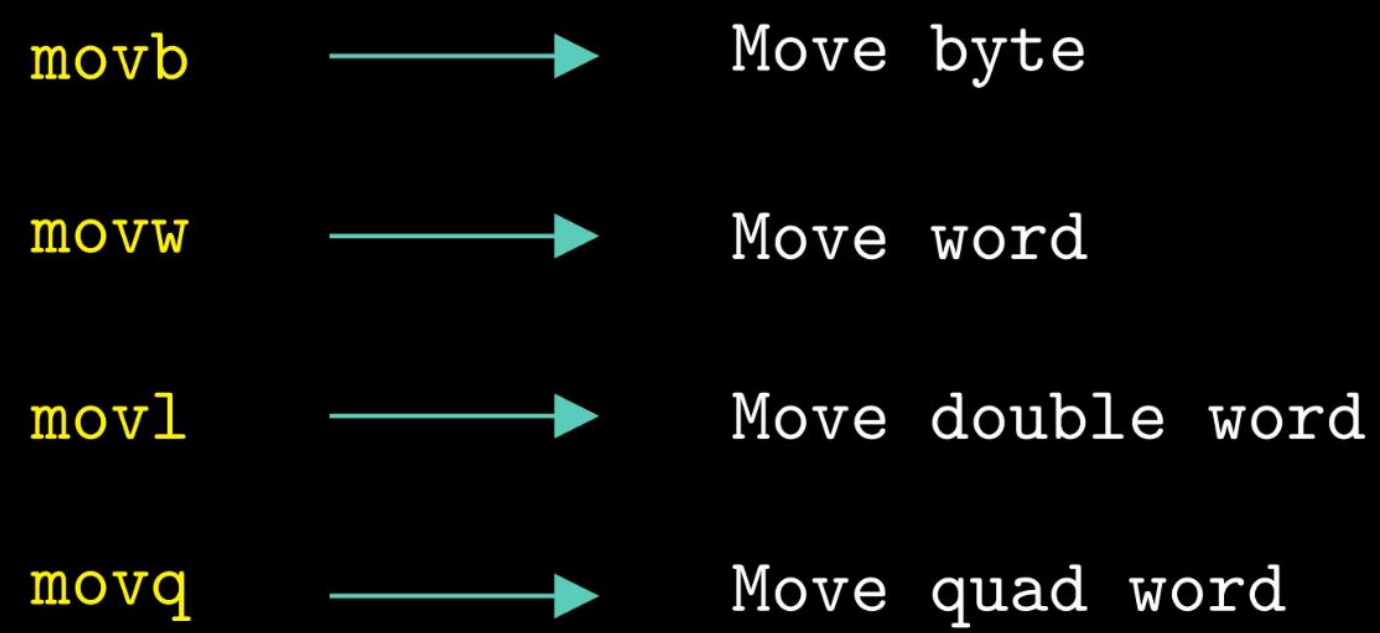
Operation code (操作码)	Operands (操作数)
`movq`	`(%rdi), %rax`
`addq`	`$8, %rsx`
`subq`	`%rdi, %rax`
`xorq`	`%rsi, %rdi`
`ret`	

Immediate (立即数)

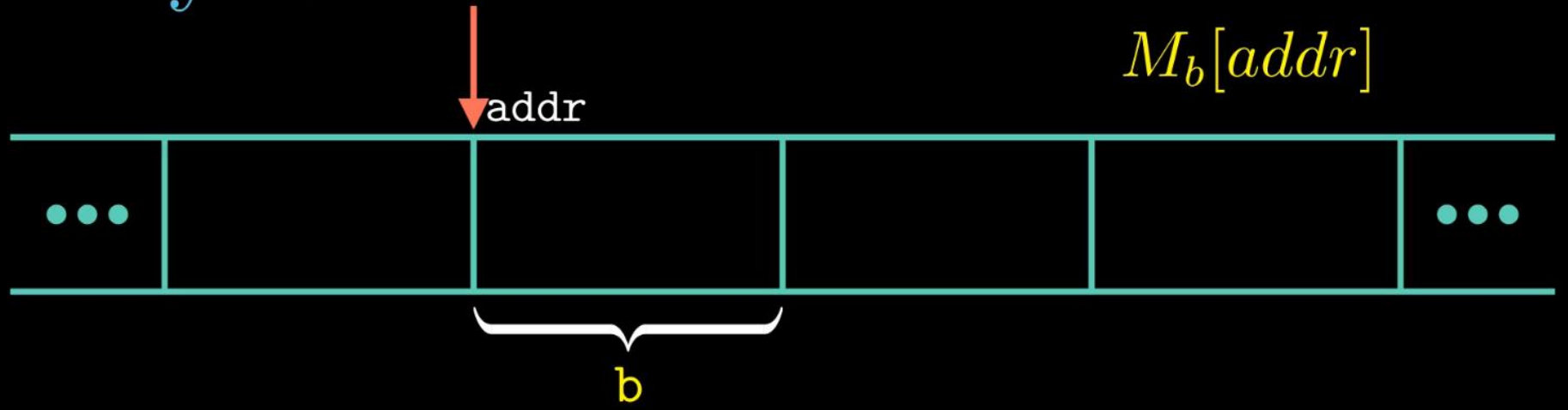
Register (寄存器)

Memory Reference (内存引用)

Data Movement Instructions



Memory Reference



$$Imm(r_b, r_i, s) \longrightarrow \text{Imm} + R[r_b] + R[r_i] \cdot s$$

Imm ⟶ Immediate (立即数)

r_b ⟶ Base Register (基址寄存器)

r_i ⟶ Index Register (变址寄存器)

s ⟶ Scale Factor (比例因子) [1, 2, 4, 8]

Size of Data Type in x86-64

C declaration	Intel data type	Assembly-code suffix	Size(bytes)
`char`	Byte	`b`	1
`short`	Word	`w`	2
`int`	Double word	`l`	4
`long`	Quad word	`q`	8
`char *`	Quad word	`q`	8
`float`	Single precision	`s`	4
`double`	Double precision	`l`	8

Data Movement Instructions

MOV Source operand (源操作数) Destination operand (目的操作数)

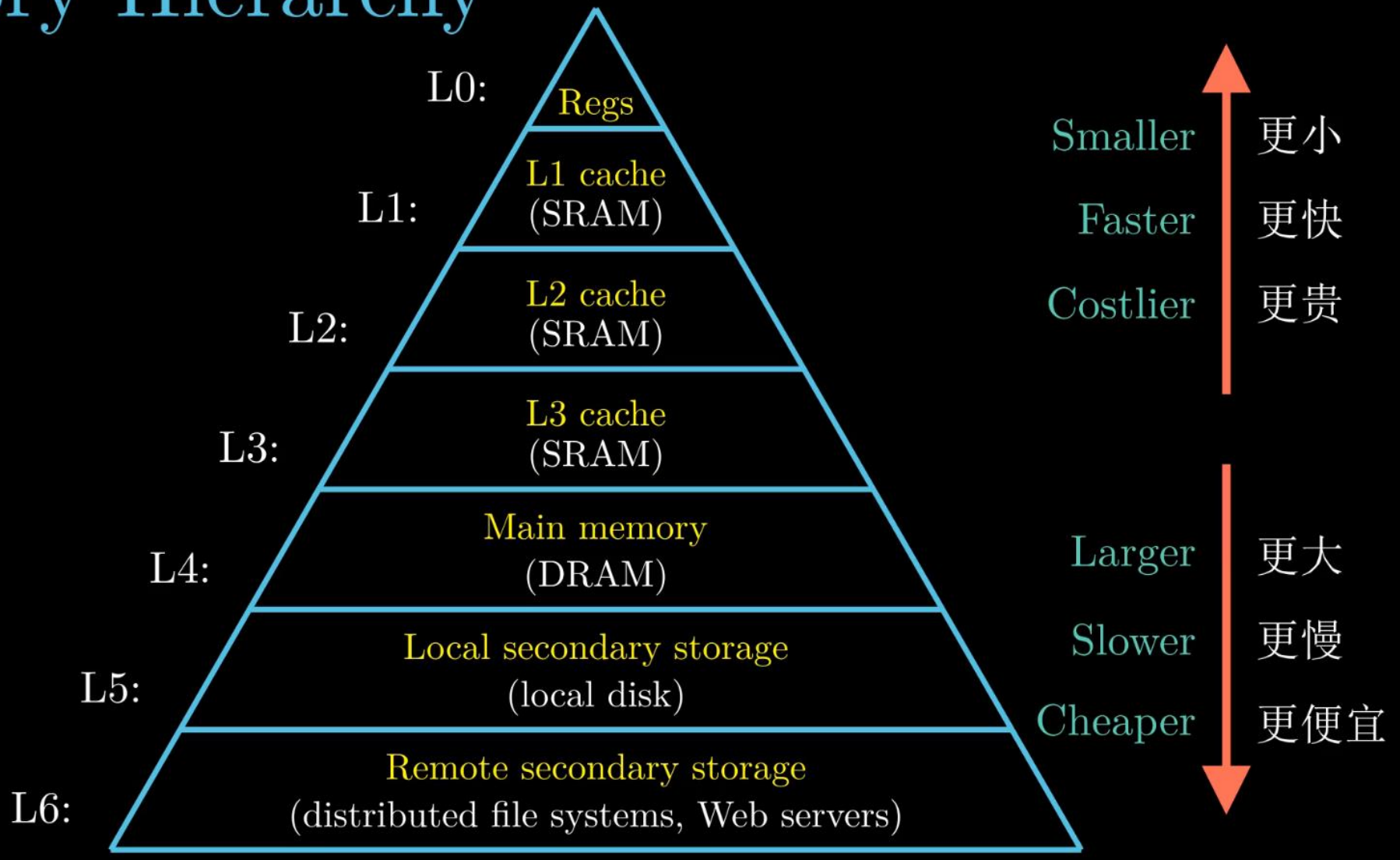
Data Movement Instructions

MOV	Source operand (源操作数)	Destination operand (目的操作数)
	Immediate	Register
	Register	Memory
	Memory	

Memory → Memory:

```
mov  memory, register
mov  register, memory
```

Memory Hierarchy



Data Movement Instructions

	Source	Destination	
`movl`	`$0x4050,`	`%eax`	Immediate → Register
`movw`	`%bp,`	`%sp`	Register → Register
`movb`	`(%rdi,%rcx),`	`%al`	Memory → Register
`movb`	`$-17,`	`(%rsp)`	Immediate → Memory
`movq`	`%rax,`	`-12(%rbp)`	Register → Memory

Data Movement Example

```
int main() {
    long a = 4;
    long b = exchange(&a, 3)
    printf("a = %ld, b = %ld\n", a, b);
    return 0;
}
```

```
long exchange(long *xp, long y) {
    long x = *xp;
    *xp = y
    return x;
}
```

```
linux> a = 3, b = 4
```

Data Movement Example

```
long exchange(long *xp, long y) {
    long x = *xp;
    *xp = y
    return x;
}
```

```
exchange:
  movq (%rdi), %rax
  movq %rsi, (%rdi)
  ret
```

Data Movement Example

```
long exchange(long *xp, long y) {
    long x = *xp;
    *xp = y
    return x;
}
```

```
exchange:
  xp in %rdi, y in %rsi
  movq (%rdi), %rax   Memory ⟶ Register
  movq %rsi, (%rdi)   Register ⟶ Memory
  ret
```

Zero-extending

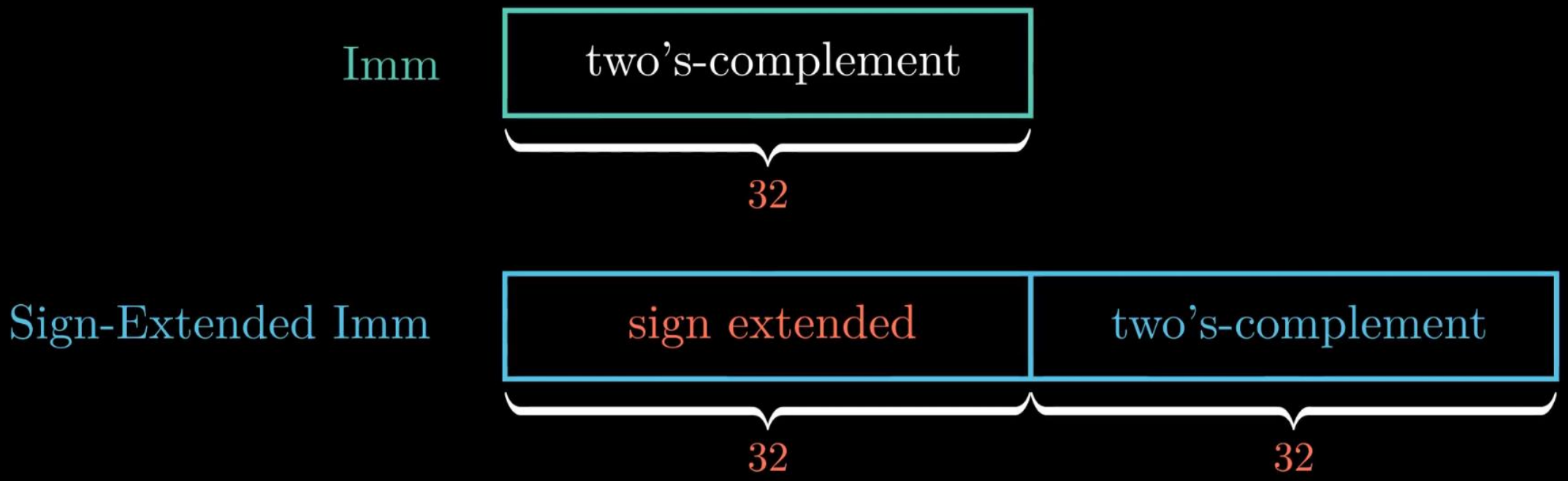
Instruction	Effect	Description
MOVZ S, R	$R \leftarrow$ ZeroExtend(S)	Move with zero extension
`movzbw`		Move Zero-extended byte to word
`movzbl`		Move Zero-extended byte to Double word
`movzwl`		Move Zero-extended word to Double word
`movzbq`		Move Zero-extended byte to Quad word
`movzwq`		Move Zero-extended word to Quad word

Sign-extending

Instruction	Effect	Description
MOVS S, R	$R \leftarrow$ SignExtend(S)	Move with sign extension
`movsbw`		Move Sign-extended byte to word
`movsbl`		Move Sign-extended byte to Double word
`movswl`		Move Sign-extended word to Double word
`movsbq`		Move Sign-extended byte to Quad word
`movswq`		Move Sign-extended word to Quad word
`movslq`		Move Sign-extended Double word to quad word
`cltq`	`movslq %eax, %rax`	

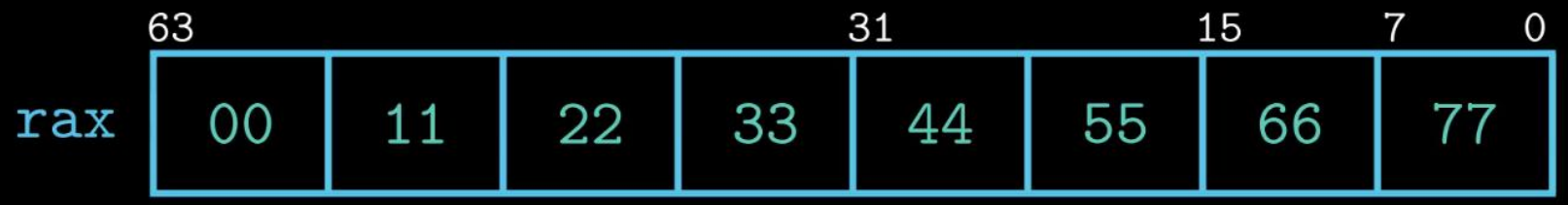
Special Situation

	S	D
`movq`	`$Imm,`	`%rax`



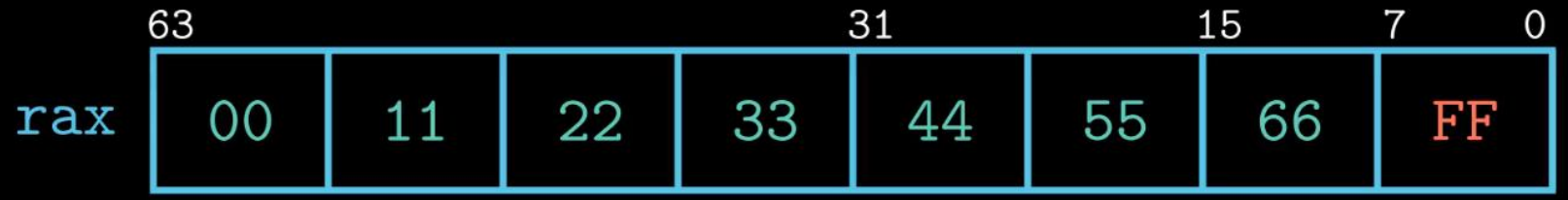
Change Destination Register

```
movabsq   $0x0011223344556677,   %rax
```



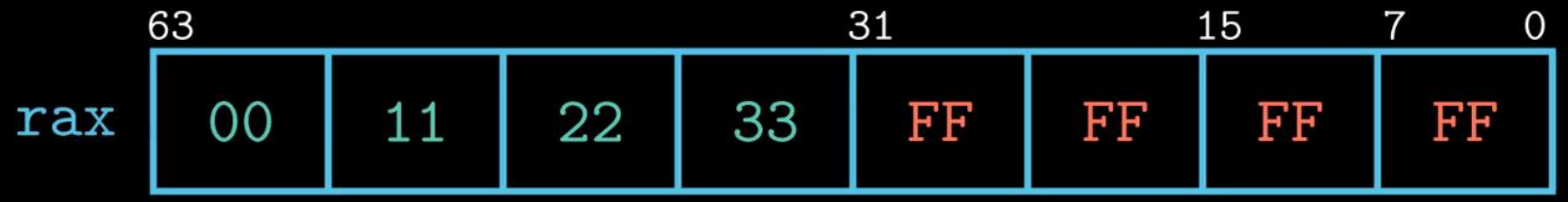
Change Destination Register

```
movabsq   $0x0011223344556677,   %rax

movb      $-1   %al
```



Change Destination Register

```
movabsq   $0x0011223344556677,   %rax

movb      $-1   %al

movw      $-1   %ax

movl      $-1   %eax
```



Change Destination Register

```
movabsq   $0x0011223344556677,   %rax

movb      $-1   %al

movvw     $-1   %ax

movl      $-1   %eax
```

Computer Systems

A Programmer's Perspective

Chapter 3: Machine-Level Representation of Programs

(程序的机器级表示)

Register


63
31
15
7
0
%rax
%eax
%ax
%al
%rsi
%esi
%si
%sil
%rbx
%ebx
%bx
%bl
%rdi
%edi
%di
%dil
%rcx
%ecx
%cx
%cl
%rbp
%ebp
%bp
%bpl
%rdx
%edx
%dx
%dl
%rsp
%esp
%sp
%spl

Register

63 0

%rax	Return value-Caller saved
%rbx	Callee saved
%rcx	Argument #4 - Caller saved
%rdx	Argument #3 - Caller saved
%rsi	Argument #2 - Caller saved
%rdi	Argument #1 - Caller saved
%rbp	Callee saved
%rsp	Stack pointer

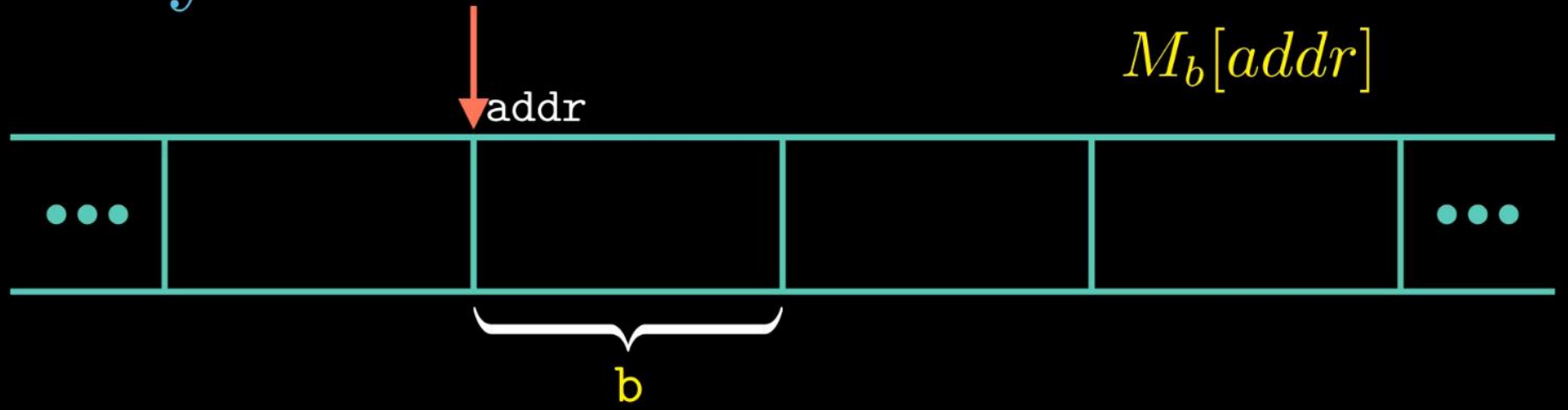
Register

63 0

%r8	Argument #5 -	Caller saved
%r9	Argument #6 -	Caller saved
%r10		Caller saved
%r11		Caller saved

%r12	Callee saved
%r13	Callee saved
%r14	Callee saved
%r15	Callee saved

Memory Reference



$$Imm(r_b, r_i, s) \longrightarrow \text{Imm} + R[r_b] + R[r_i] \cdot s$$

Imm ⟶ Immediate (立即数)

r_b ⟶ Base Register (基址寄存器)

r_i ⟶ Index Register (变址寄存器)

s ⟶ Scale Factor (比例因子) [1, 2, 4, 8]

Memory Reference

Type	Form	Operand value	Name
Memory	$Imm(r_b, r_i, s)$	$M[Imm + R[r_b] + R[r_i] \cdot s]$	Scaled indexed
Memory	Imm	$M[Imm]$	Absolute
Memory	(r_a)	$M[R[r_a]]$	Indirect
Memory	$Imm(r_b)$	$M[Imm + R[r_b]]$	Base + displacement
Memory	(r_b, r_i)	$M[R[r_b] + R[r_i]]$	Indexed
Memory	$Imm(r_b, r_i)$	$M[Imm + R[r_b] + R[r_i]]$	Indexed
Memory	$(, r_i, s)$	$M[R[r_i] \cdot s]$	Scale indexed
Memory	$Imm(, r_i, s)$	$M[Imm + R[r_i] \cdot s]$	Scale indexed
Memory	(r_b, r_i, s)	$M[R[r_b] + R[r_i] \cdot s]$	Scale indexed

Data Movement Instructions

MOV Source operand (源操作数) Destination operand (目的操作数)

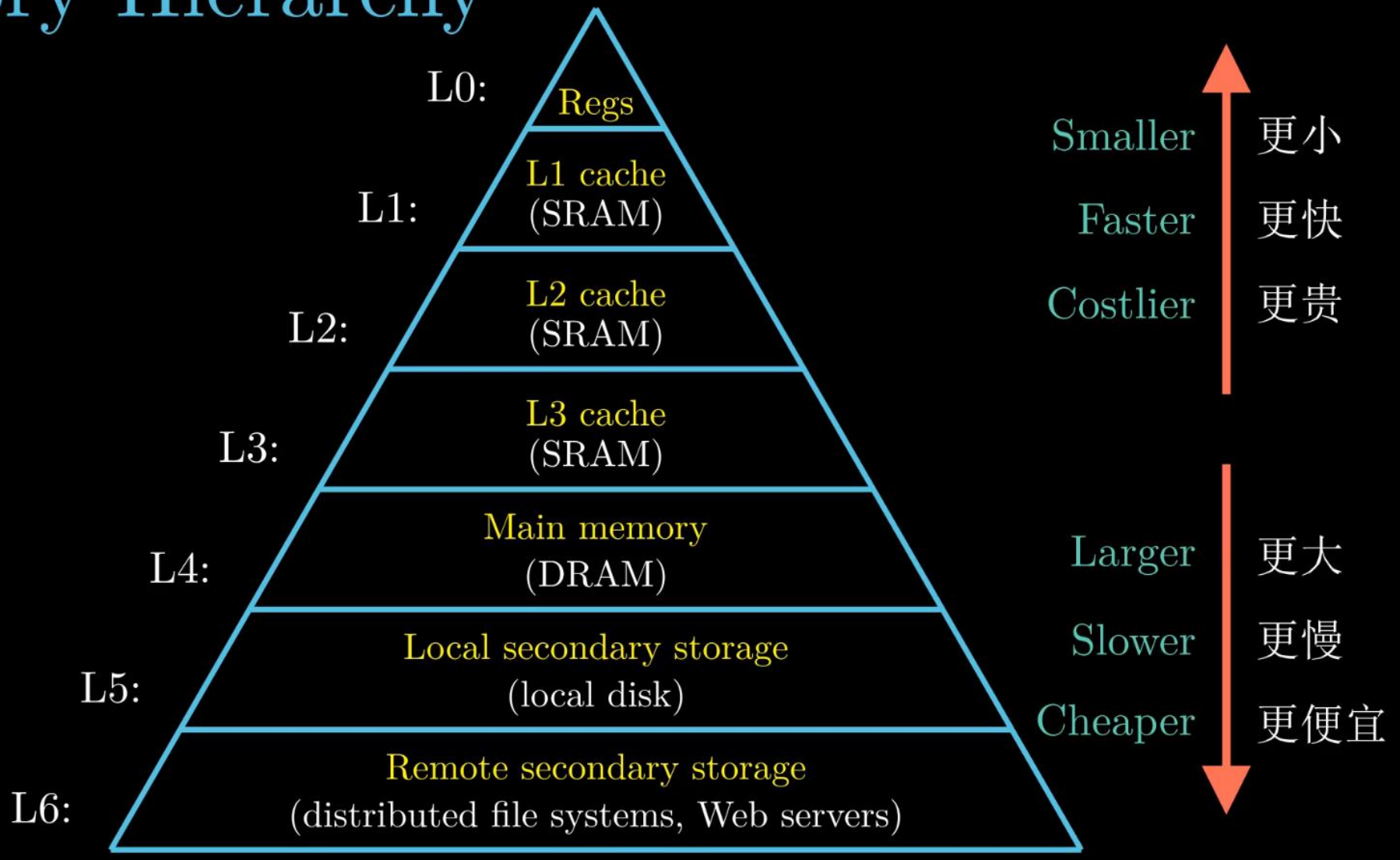
Data Movement Instructions

MOV	Source operand (源操作数)	Destination operand (目的操作数)
	Immediate	Register
	Register	Memory
	Memory	

Memory → Memory:

```
mov  memory, register
mov  register, memory
```

Memory Hierarchy



Data Movement Instructions

	Source	Destination	
`movl`	`$0x4050,`	`%eax`	Immediate → Register
`movw`	`%bp,`	`%sp`	Register → Register
`movb`	`(%rdi,%rcx),`	`%al`	Memory → Register
`movb`	`$-17,`	`(%rsp)`	Immediate → Memory
`movq`	`%rax,`	`-12(%rbp)`	Register → Memory

Data Movement Example

```
int main() {
    long a = 4;
    long b = exchange(&a, 3)
    printf("a = %ld, b = %ld\n", a, b);
    return 0;
}
```

```
long exchange(long *xp, long y) {
    long x = *xp;
    *xp = y
    return x;
}
```

```
linux> a = 3, b = 4
```

Data Movement Example

```
long exchange(long *xp, long y) {
    long x = *xp;
    *xp = y
    return x;
}
```

```
exchange:
  xp in %rdi, y in %rsi
  movq (%rdi), %rax   Memory ⟶ Register
  movq %rsi, (%rdi)   Register ⟶ Memory
  ret
```

Register

63 0

%rax	Return value-Caller saved
%rbx	Callee saved
%rcx	Argument #4 - Caller saved
%rdx	Argument #3 - Caller saved
%rsi	Argument #2 - Caller saved
%rdi	Argument #1 - Caller saved
%rbp	Callee saved
%rsp	Stack pointer

Zero-extending

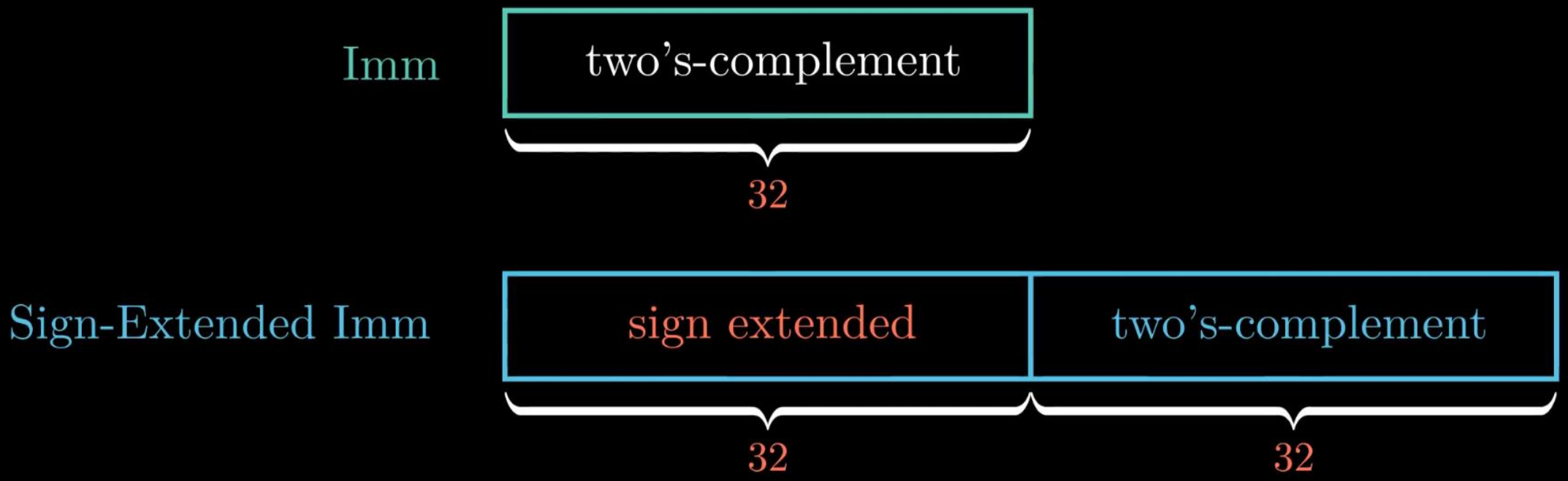
Instruction	Effect	Description
MOVZ S, R	$R \leftarrow$ ZeroExtend(S)	Move with zero extension
`movzbw`		Move Zero-extended byte to word
`movzbl`		Move Zero-extended byte to Double word
`movzwl`		Move Zero-extended word to Double word
`movzbq`		Move Zero-extended byte to Quad word
`movzwq`		Move Zero-extended word to Quad word

Sign-extending

Instruction	Effect	Description
MOVS S, R	$R \leftarrow$ SignExtend(S)	Move with sign extension
`movsbw`		Move Sign-extended byte to word
`movsbl`		Move Sign-extended byte to Double word
`movswl`		Move Sign-extended word to Double word
`movsbq`		Move Sign-extended byte to Quad word
`movswq`		Move Sign-extended word to Quad word
`movslq`		Move Sign-extended Double word to quad word
`cltq`	`movslq %eax, %rax`	

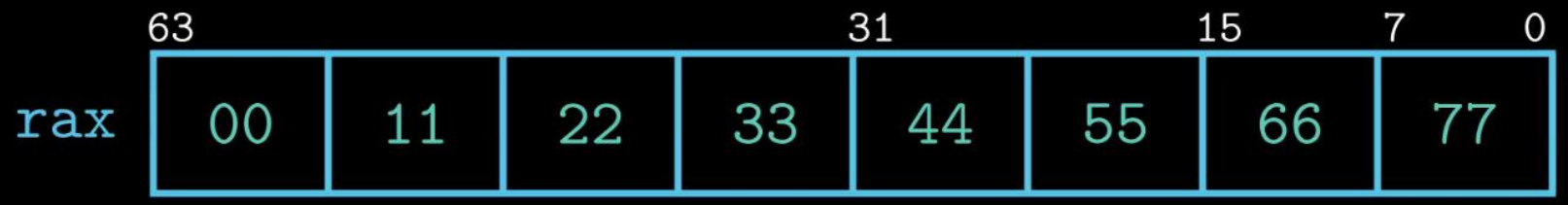
Special Situation

	S	D
`movq`	`$Imm,`	`%rax`



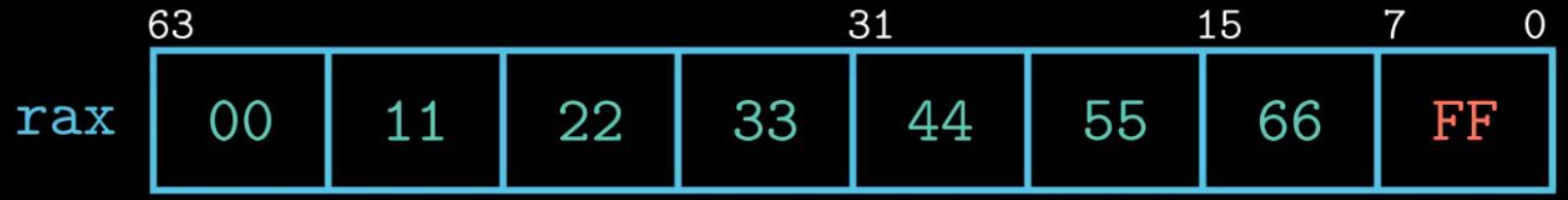
Change Destination Register

```
movabsq   $0x0011223344556677,   %rax
```



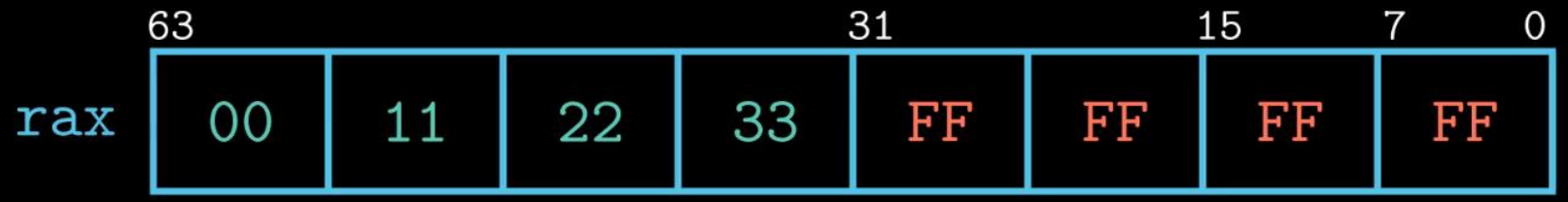
Change Destination Register

```
movabsq   $0x0011223344556677,   %rax

movb      $-1   %al
```



Change Destination Register

```
movabsq   $0x0011223344556677,   %rax

movb      $-1   %al

movw      $-1   %ax

movl      $-1   %eax
```

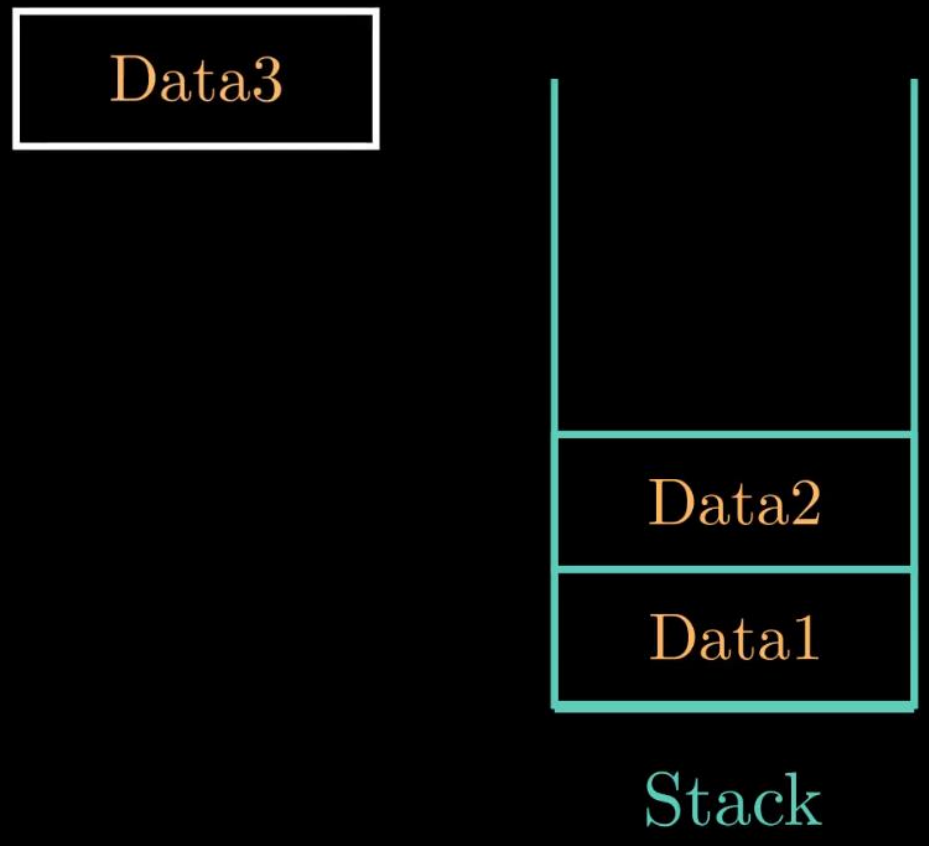


Change Destination Register

```
movabsq   $0x0011223344556677,   %rax

movb      $-1   %al

movvw     $-1   %ax

movl      $-1   %eax
```

Stack

Push



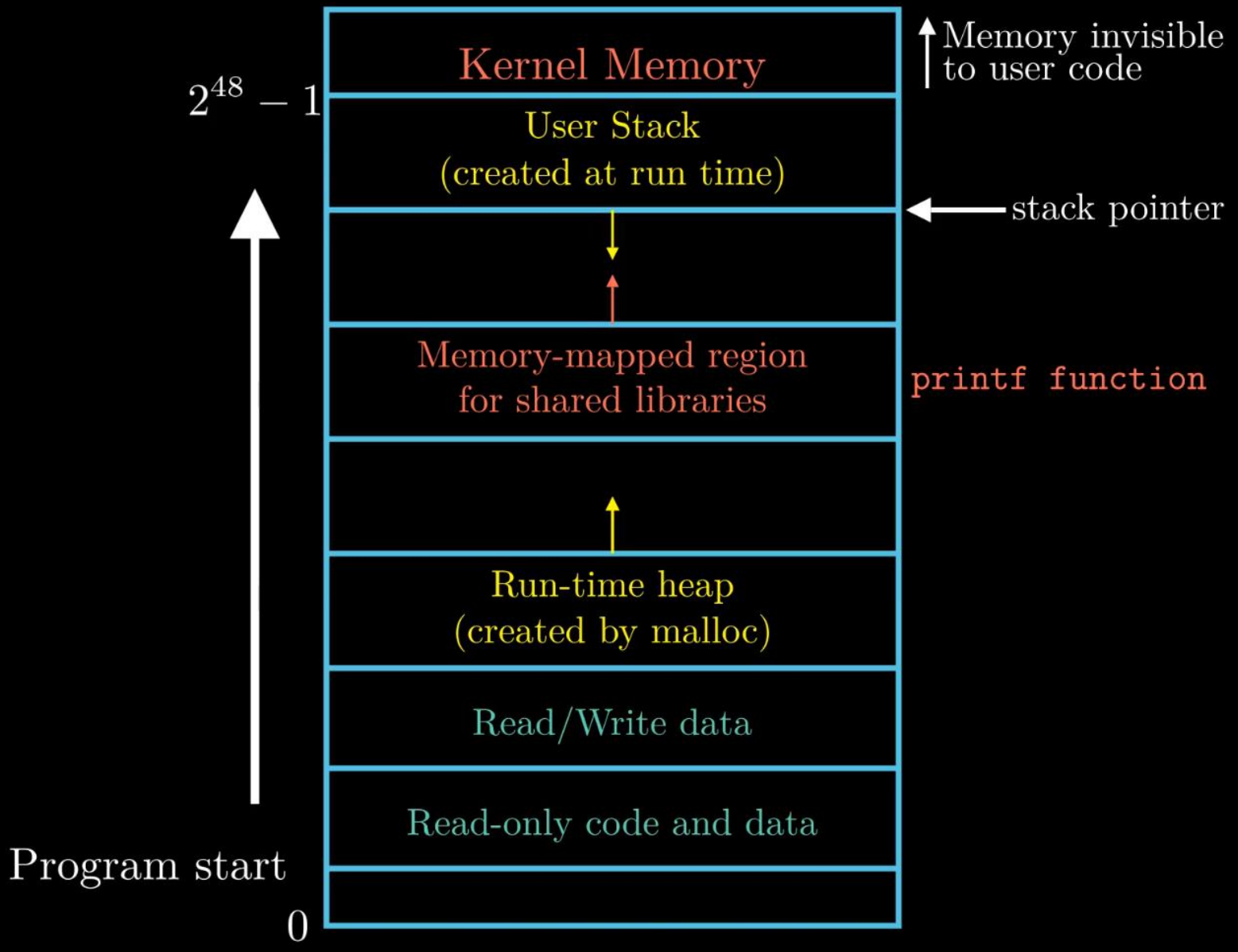
Stack

Data3

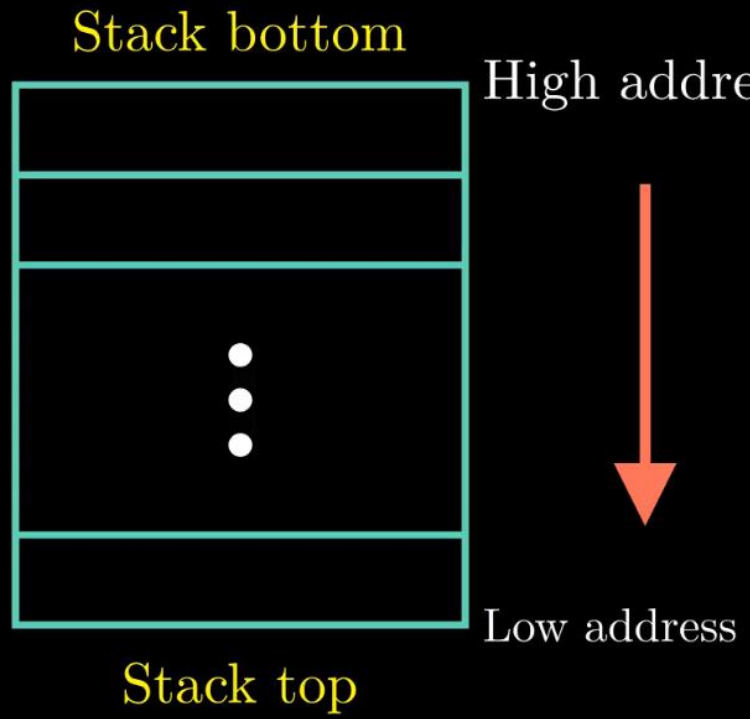
Pop

Stack

Stack


$2^{48} - 1$
Kernel Memory
User Stack
(created at run time)
Memory-mapped region
for shared libraries
Run-time heap
(created by malloc)
Read/Write data
Read-only code and data
0
Program start
Memory invisible
to user code
stack pointer
printf function

Stack

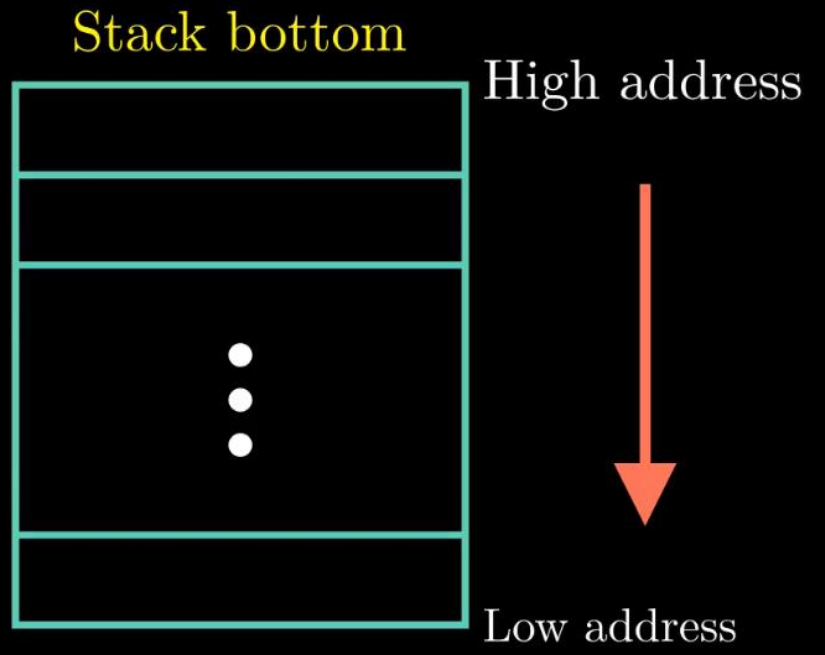


Stack

%rax 0x123

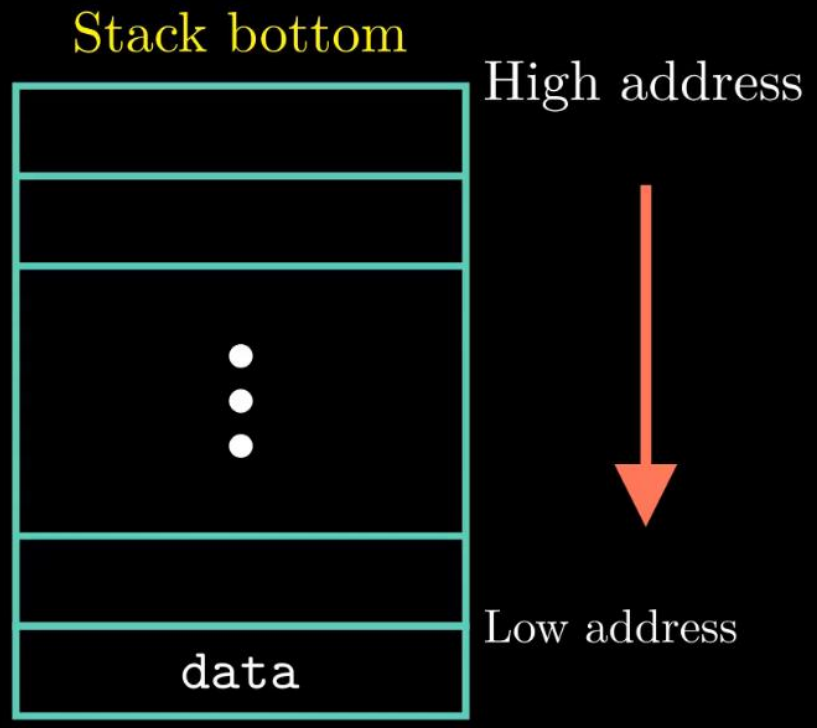
pushq %rax

data



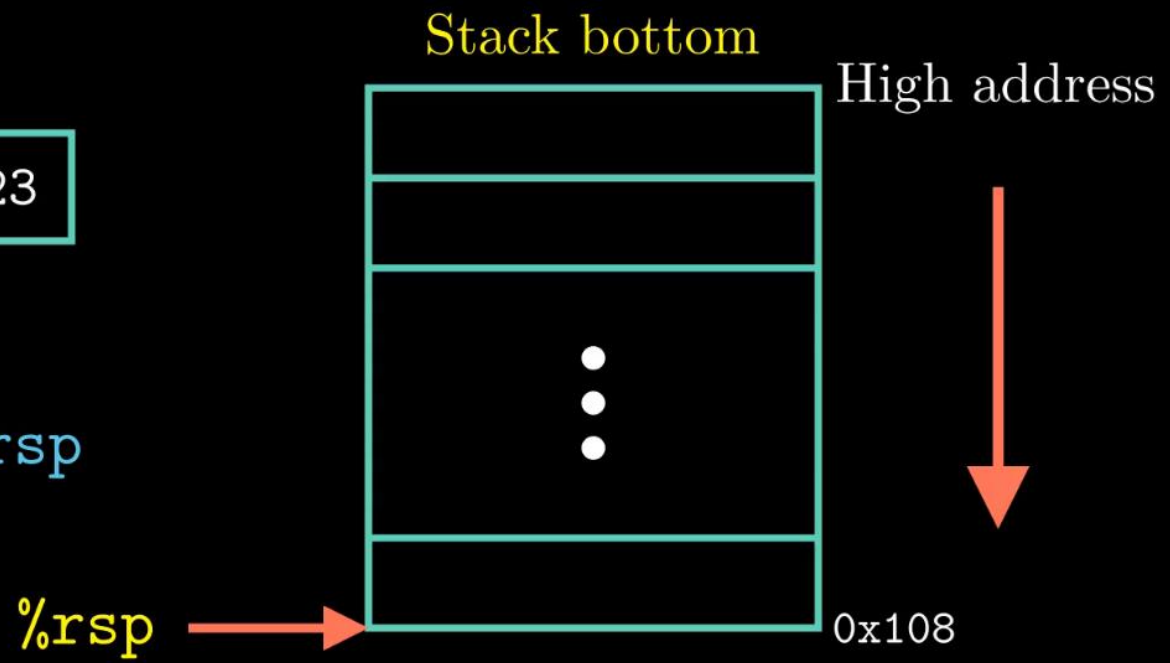
Stack

```
%rax    0x123
pushq   %rax
```



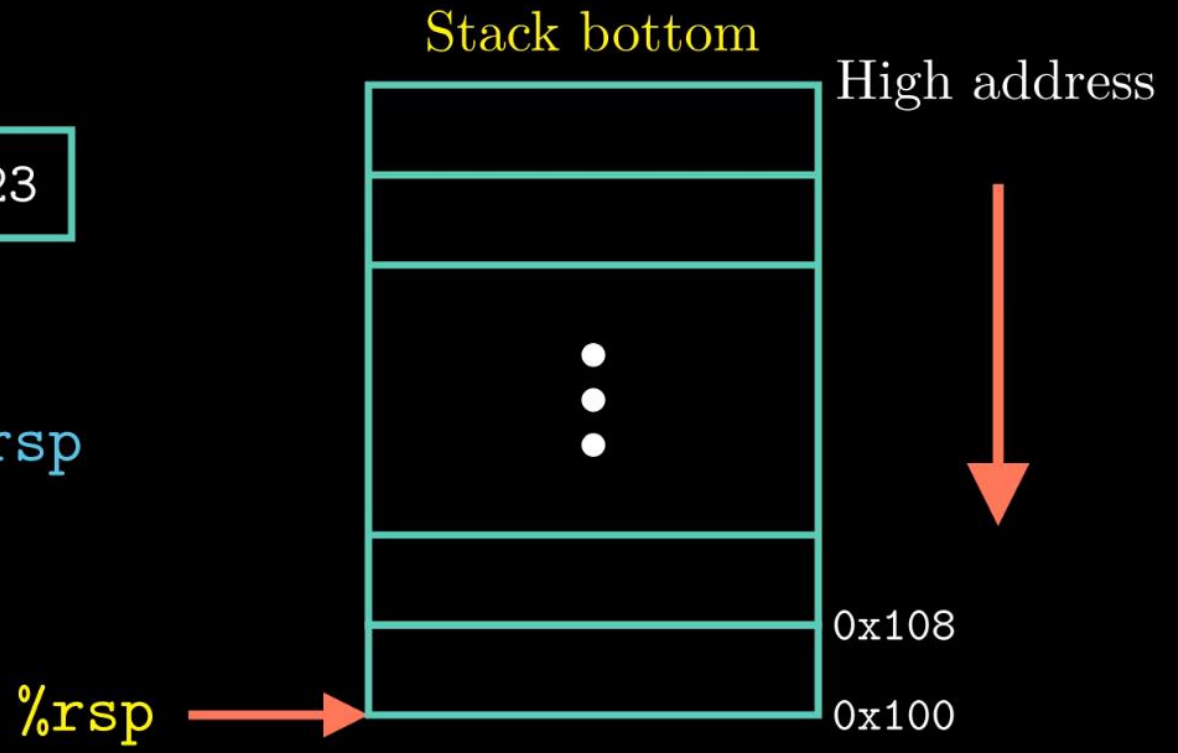
Stack

```
%rax      0x123
pushq   %rax
subq   $8, %rsp
```



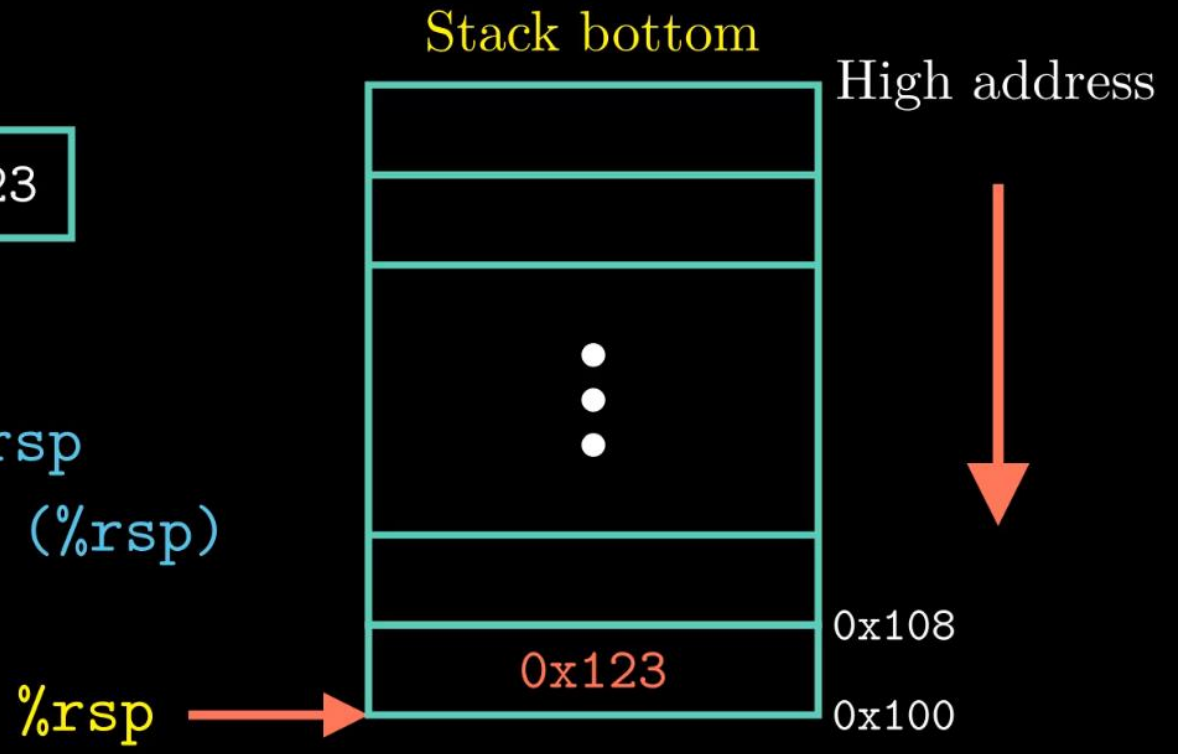
Stack

```
%rax    0x123
pushq   %rax
subq    $8, %rsp
```

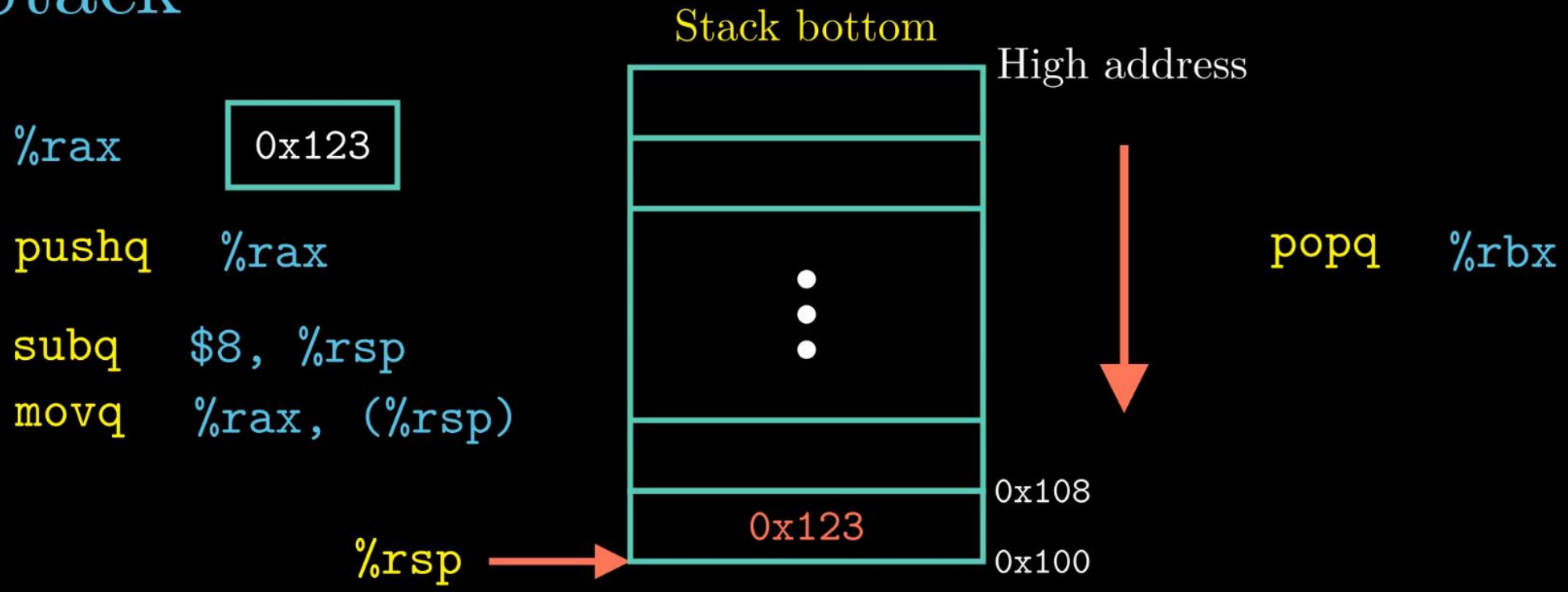


Stack

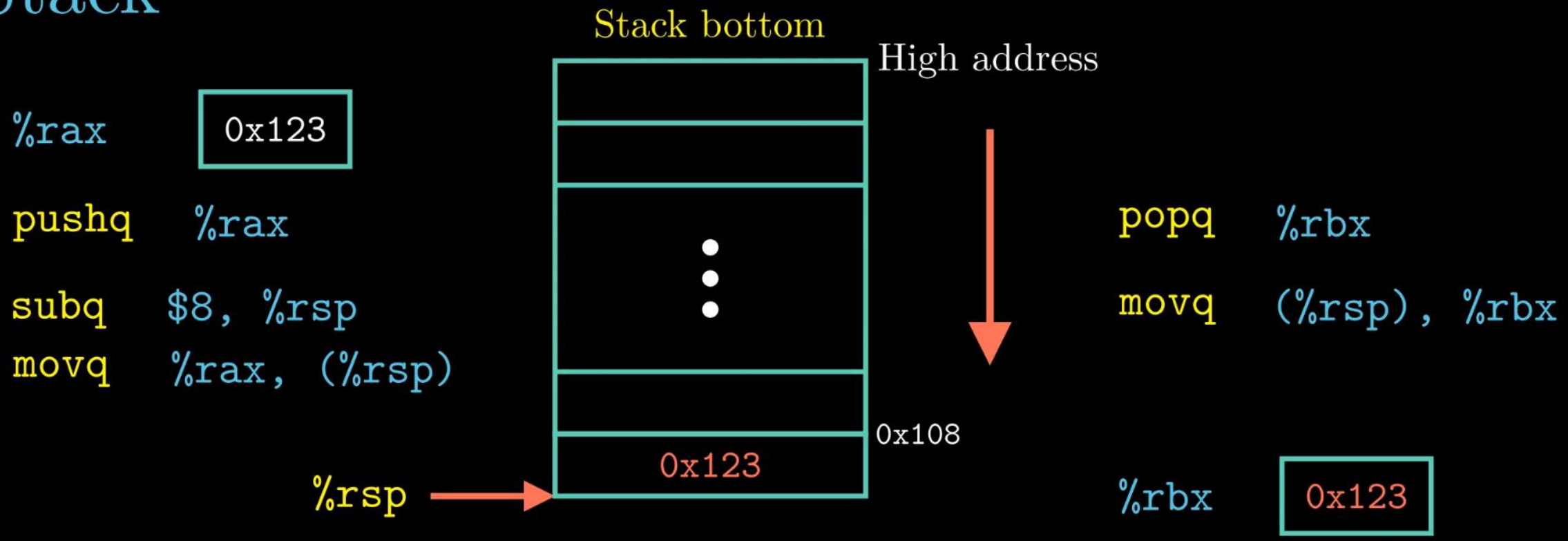
```
%rax      0x123

pushq   %rax

subq    $8, %rsp
movq    %rax, (%rsp)
```



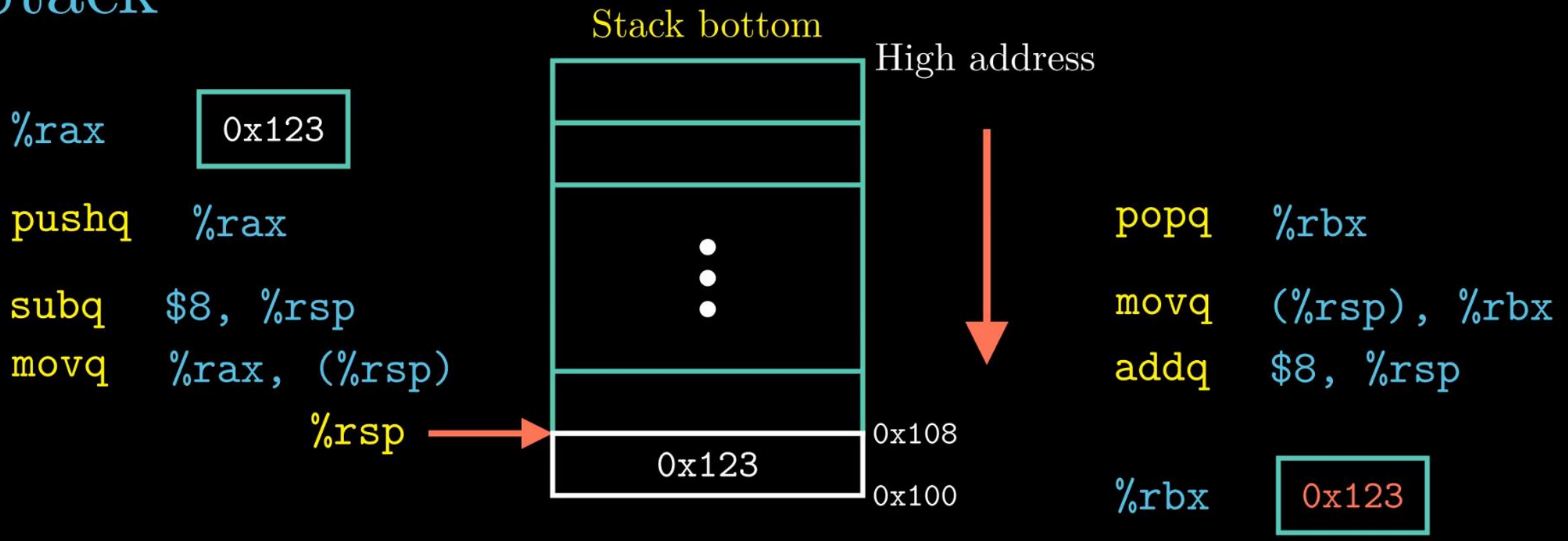
Stack


%rax
0x123
pushq %rax
subq $8, %rsp
movq %rax, (%rsp)
Stack bottom
High address
popq %rbx
0x108
0x123
0x100
%rsp

Stack


%rax
0x123
pushq %rax
subq $8, %rsp
movq %rax, (%rsp)
Stack bottom
High address
0x108
0x123
%rsp
popq %rbx
movq (%rsp), %rbx
%rbx
0x123

Stack


%rax
0x123
pushq %rax
subq $8, %rsp
movq %rax, (%rsp)
%rsp
Stack bottom
High address
0x123
0x108
0x100
popq %rbx
movq (%rsp), %rbx
addq $8, %rsp
%rbx
0x123

LEAQ

`leaq S, D` → Load Effective Address

LEAQ

`leaq S, D` → Load Effective Address

```
leaq 7(%rdx, %rdx, 4), %rax
```

$Imm(r_b, r_i, s) \longrightarrow Imm + R[r_b] + R[r_i] \cdot s$

LEAQ

`leaq S, D` ⟶ Load Effective Address

```
leaq 7(%rdx, %rdx, 4), %rax
```

$Imm(r_b, r_i, s) \longrightarrow Imm + R[r_b] + R[r_i] \cdot s$

`%rdx` ⟵ x

Effective Address: `7 + %rdx + %rdx * 4` $= 5x + 7$

LEAQ Instruction

```
long scale(long x, long y long z) {
    long t = x + 4 * y + 12 * z;
    return t;
}
```

```
scale:
  leaq (%rdi, %rsi, 4), %rax
  leaq (%rdx, %rdx, 2), %rdx
  leaq (%rax, %rdx, 4), %rax
  ret
```

LEAQ Instruction

```
t = x + 4 * y + 12 * z;
```

```
scale:
  x in %rdi, y in %rsi, z in %rdx
  leaq (%rdi, %rsi, 4), %rax
  leaq (%rdx, %rdx, 2), %rdx
  leaq (%rax, %rdx, 4), %rax
  ret
```

Register

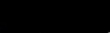
63 0

%rax	Return value-Caller saved
%rbx	Callee saved
%rcx	Argument #4 - Caller saved
%rdx	Argument #3 - Caller saved
%rsi	Argument #2 - Caller saved
%rdi	Argument #1 - Caller saved
%rbp	Callee saved
%rsp	Stack pointer

LEAQ Instruction

```
t = x + 4 * y + 12 * z;
```

```
scale:
  x in %rdi, y in %rsi, z in %rdx
  leaq (%rdi, %rsi, 4), %rax
  leaq (%rdx, %rdx, 2), %rdx
  leaq (%rax, %rdx, 4), %rax
  ret
```



```
%rdi + 4*%rsi = x + 4*y
```

LEAQ Instruction

```
t = x + 4 * y + 12 * z;
```

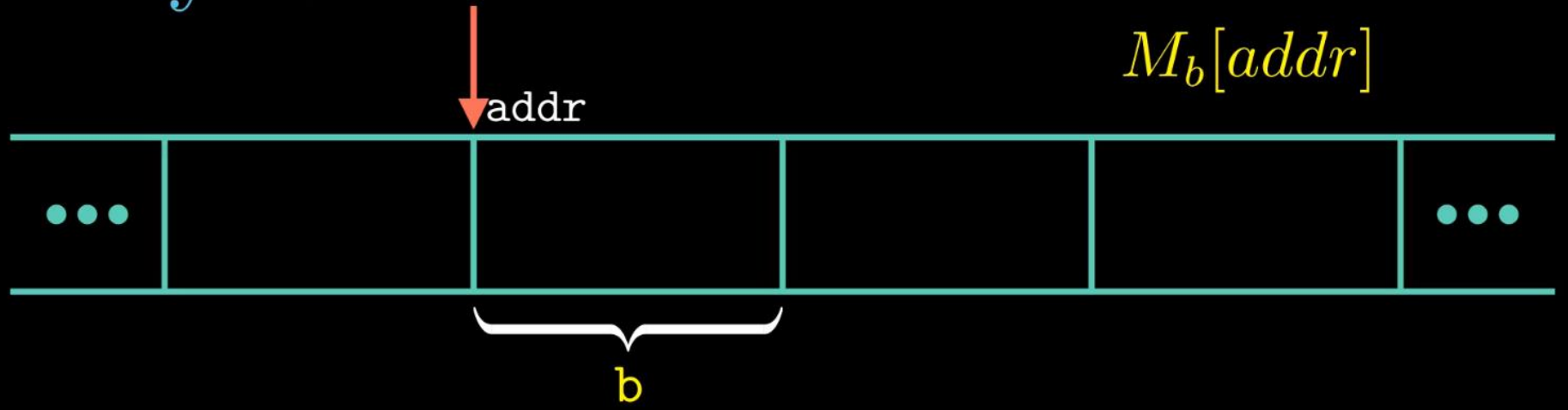
```
scale:
  x in %rdi, y in %rsi, z in %rdx
  leaq (%rdi, %rsi, 4), %rax  ⟶  %rdi + 4*%rsi = x + 4*y
  leaq (%rdx, %rdx, 2), %rdx  ⟶  %rdx + 2*%rdx = z + 2*z
  leaq (%rax, %rdx, 4), %rax  ⟶  %rax + 4*%rdx
  ret                                = (x + 4*y) + 4*(3*z)
```

LEAQ Instruction

```
t = x + 4 * y + 12 * z;
```

```
scale:
  x in %rdi, y in %rsi, z in %rdx
  leaq (%rdi, %rsi, 4), %rax  ⟶  %rdi + 4*%rsi = x + 4*y
  leaq (%rdx, %rdx, 2), %rdx  ⟶  %rdx + 2*%rdx = z + 2*z
  leaq (%rax, %rdx, 4), %rax  ⟶  %rax + 4*%rdx
  ret                                = (x + 4*y) + 4*(3*z)

  leaq (%rax, %rdx, 12), %rax ⟶  %rax + 12*%rdx
                                     = (x + 4*y) + 12*z
```

Memory Reference



$$Imm(r_b, r_i, s) \longrightarrow \text{Imm} + R[r_b] + R[r_i] \cdot s$$

Imm ⟶ Immediate (立即数)

r_b ⟶ Base Register (基址寄存器)

r_i ⟶ Index Register (变址寄存器)

s ⟶ Scale Factor (比例因子) [1, 2, 4, 8]

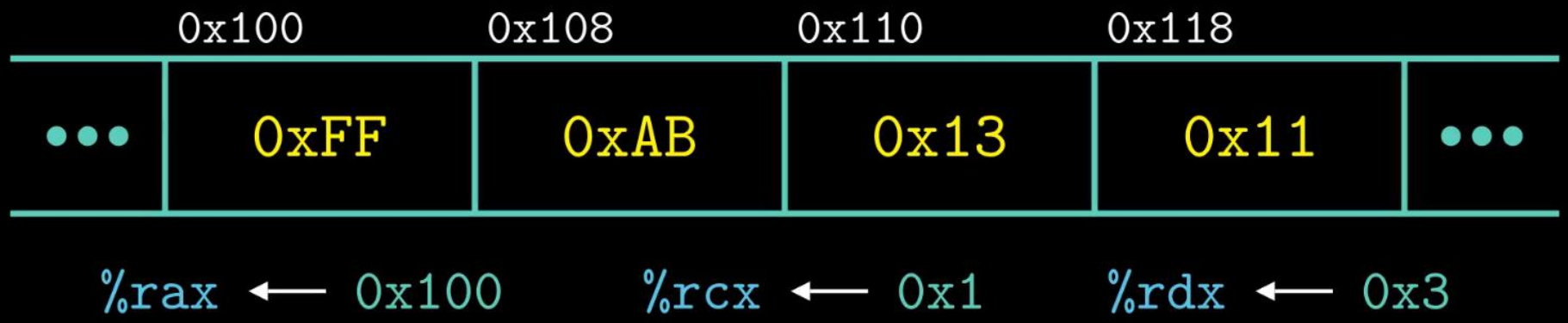
Unary Operations(一元操作)

Instruction	Effect	Description
INC D	D ⟵ D + 1	Increment(加 1)
DEC D	D ⟵ D - 1	Decrement(减 1)
NEG D	D ⟵ -D	Negate(取负)
NOT D	D ⟵ ~D	Complement(取补)

Binary Operations(二元操作)

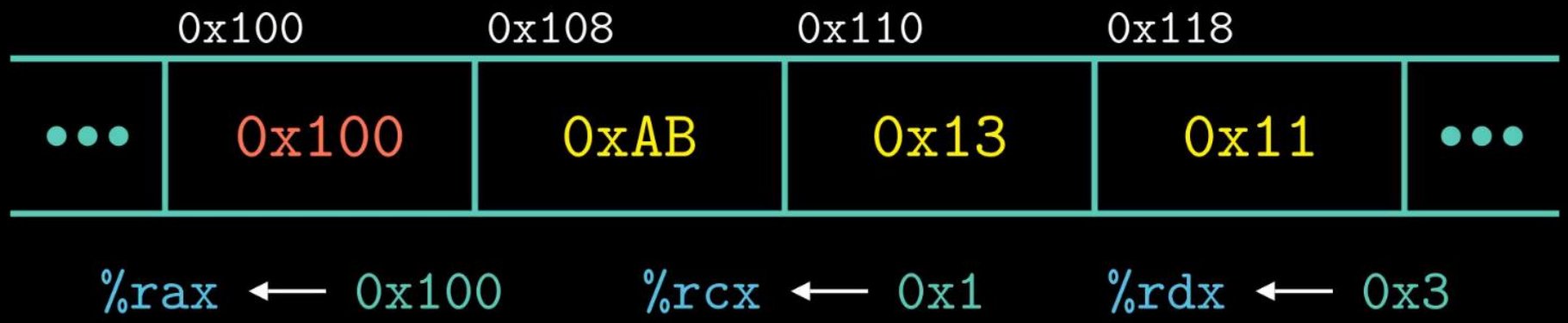
Instruction	Effect	Description
`ADD S,D`	`D ⟵ D + S`	ADD(加)
`SUB S,D`	`D ⟵ D - S`	Subtract(减)
`IMUL S,D`	`D ⟵ D * S`	Multiply(乘)
`XOR S,D`	`D ⟵ D ^ S`	Exclusive-or(异或)
`OR S,D`	`D ⟵ D | S`	Or(或)
`AND S,D`	`D ⟵ D & S`	And(与)

Instructions



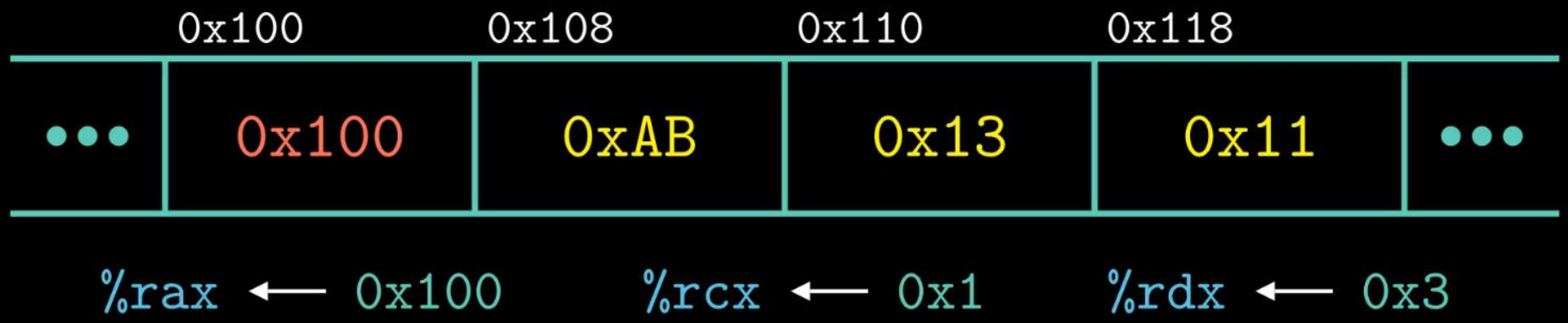

```
addq %rcx, (%rax)
```

Instructions



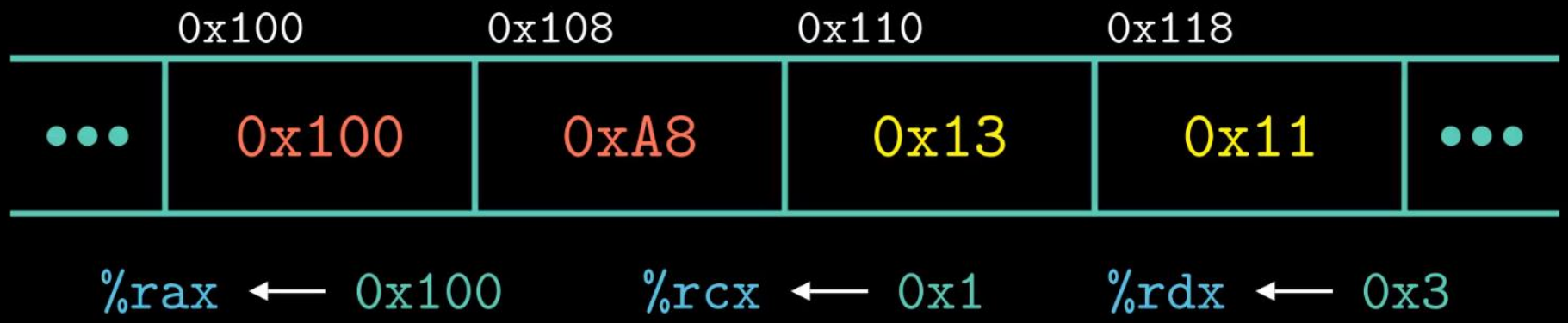
`addq %rcx, (%rax)` ⟶ `Mem[0x100] = Mem[0x100] + R[%rcx]`

Instructions




```
addq %rcx, (%rax)  ⟶  Mem[0x100] = Mem[0x100] + R[%rcx]
subq %rdx, 8(%rax)
```

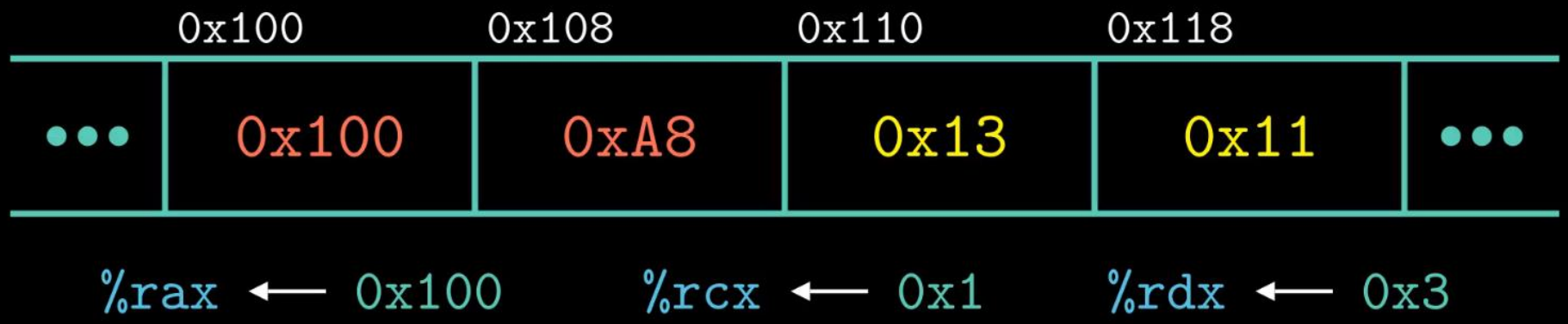
Instructions



`addq %rcx, (%rax)` ⟶ `Mem[0x100] = Mem[0x100] + R[%rcx]`

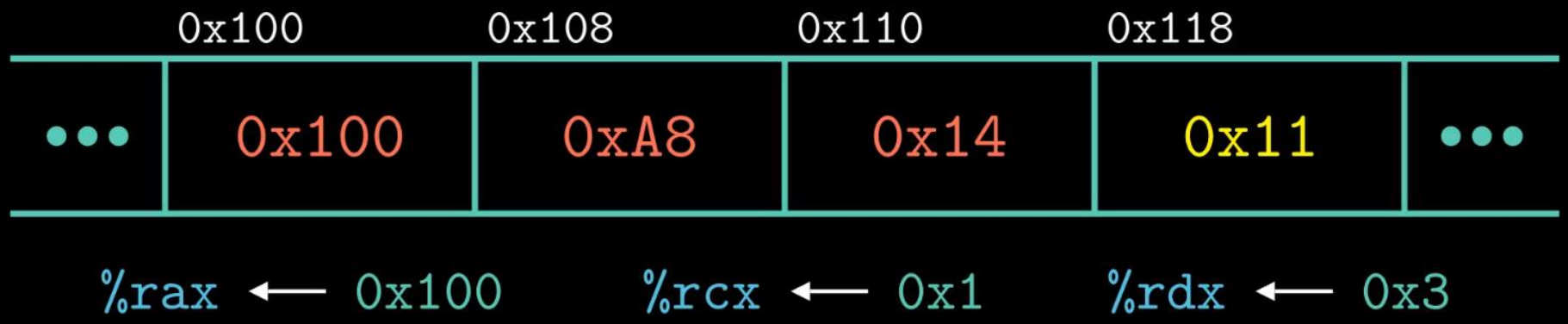
`subq %rdx, 8(%rax)` ⟶ `Mem[0x108] = Mem[0x108] - R[%rdx]`

Instructions



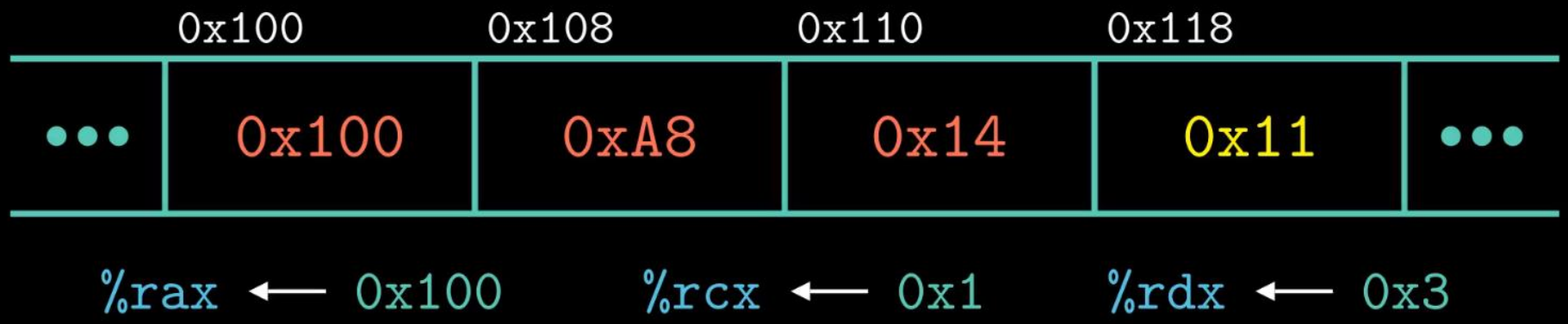

```
addq %rcx, (%rax)  ⟶  Mem[0x100] = Mem[0x100] + R[%rcx]
subq %rdx, 8(%rax) ⟶  Mem[0x108] = Mem[0x108] - R[%rdx]
incq 16(%rax)
```

Instructions



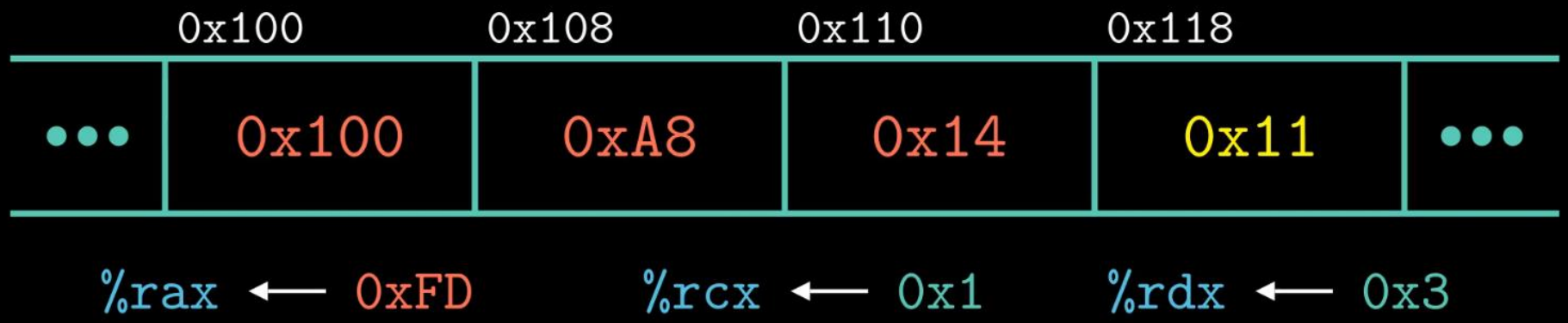

```
addq %rcx, (%rax) ⟶ Mem[0x100] = Mem[0x100] + R[%rcx]
subq %rdx, 8(%rax) ⟶ Mem[0x108] = Mem[0x108] - R[%rdx]
incq 16(%rax), ⟶ Mem[0x110] = Mem[0x110] + 1
```

Instructions




```
addq %rcx, (%rax)  ⟶  Mem[0x100] = Mem[0x100] + R[%rcx]
subq %rdx, 8(%rax)  ⟶  Mem[0x108] = Mem[0x108] - R[%rdx]
incq 16(%rax),  ⟶  Mem[0x110] = Mem[0x110] + 1
subq %rdx, %rax
```

Instructions

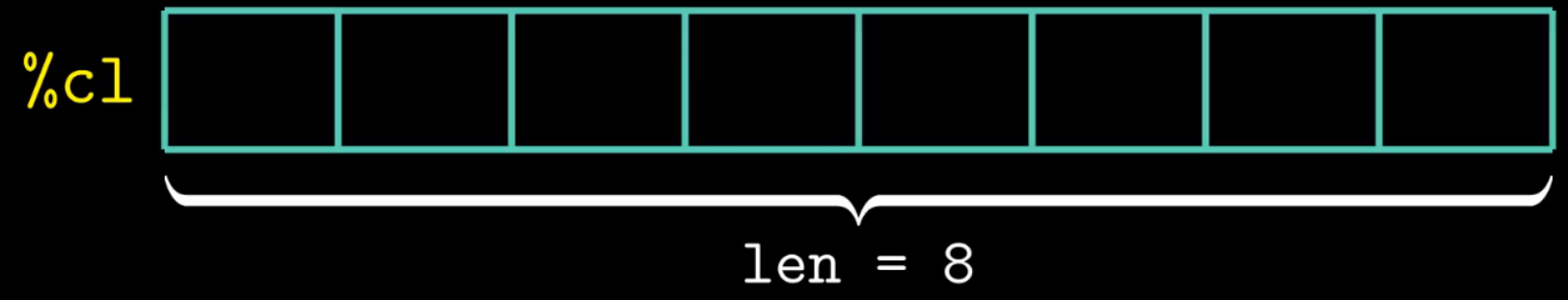



```
addq %rcx, (%rax)  ⟶  Mem[0x100] = Mem[0x100] + R[%rcx]
subq %rdx, 8(%rax)  ⟶  Mem[0x108] = Mem[0x108] - R[%rdx]
incq 16(%rax),  ⟶  Mem[0x110] = Mem[0x110] + 1
subq %rdx, %rax  ⟶  R[%rax] = R[%rax] - R[%rdx]
```

Shift Operations

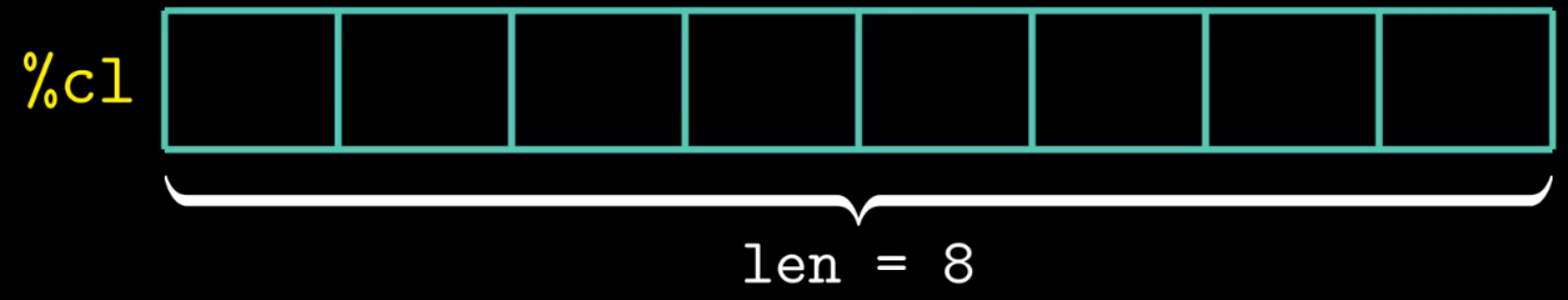
Instruction	Effect	Description
`SAL k, D`	`D` $\longleftarrow$ `D << k`	`Left shift`(左移)
`SHL k, D`	`D` $\longleftarrow$ `D << k`	`Left shift`(左移, 等同于 SAL)
`SAR k, D`	`D` $\longleftarrow >>_A k$	`Arithmetic right shift`(算术右移)
`SHR k, D`	`D` $\longleftarrow >>_L$ `k`	`Logical right shift`(逻辑右移)

Shift Amount(移位量)



For data length w: $2^m = w$

Shift Amount(移位量)



For data length w: $2^m = w$

Code

```
long arith(long x, long y long z) {
    long t1 = x^y;
    long t2 = z * 48;
    long t3 = t1 & 0x0F0F0F0F;
    long t4 = t2 - t3;
    return t4;
}
```

```
arith:
  xorq  %rsi, %rdi
  leaq (%rdx, %rdx, 2), %rax
  salq $4, %rax
  andl $252645135, %edi
  subq %rdi, %rax
  ret
```

Code

```
long t2 = z * 48;
```

```
leaq (%rdx, %rdx, 2), %rax
```

R[%rdx] + R[%rdx]*2 = 3 * z ⟶ %rax

```
salq $4, %rax
```

Code

```
long t2 = z * 48;
```

```
leaq (%rdx, %rdx, 2), %rax
```

R[%rdx] + R[%rdx]*2 = 3 * z ⟶ %rax

```
salq $4, %rax
```

2^4 * R[%rax] = 16 * (3*z) = 48 * z

Special Arithmetic Operations

Instruction	Description
imulq S	Signed full multiply(有符号全乘法)
mulq S	Unsigned full multiply(无符号全乘法)
cqto	Convert to oct word(转换为八字)
idivq S	Signed divide(有符号除法)
divq S	Unsigned divide(无符号除法)

Computer Systems

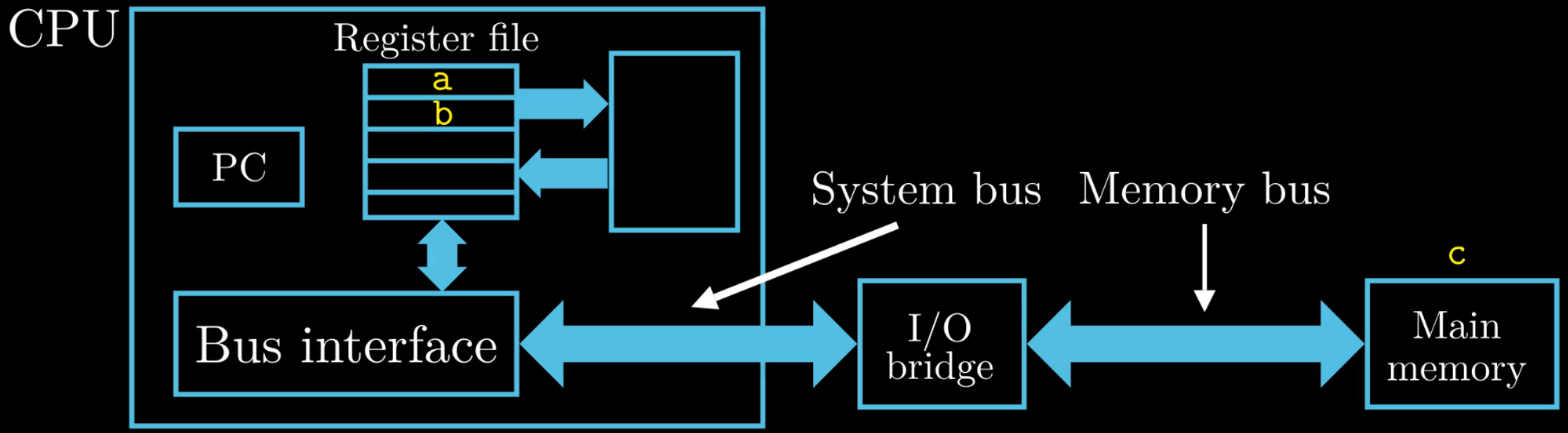
A Programmer's Perspective

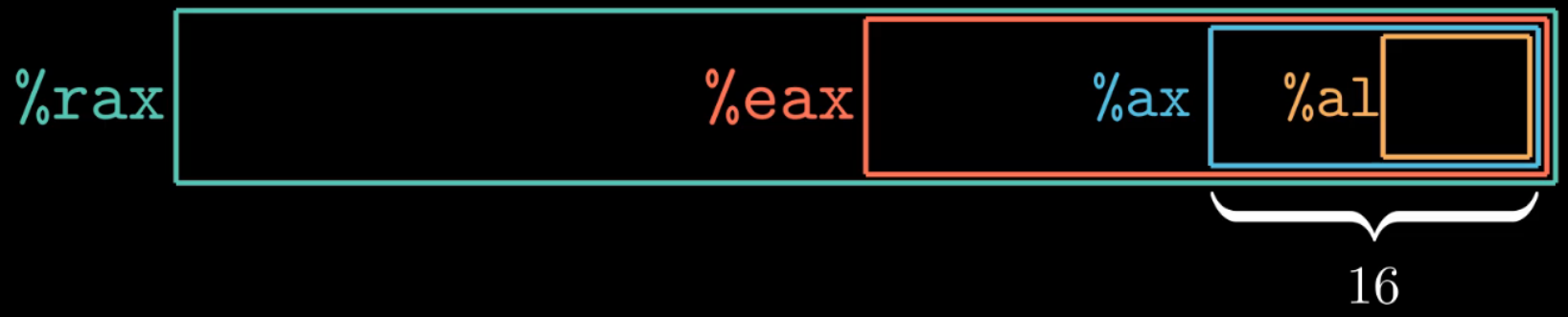
Chapter 3: Machine-Level Representation of Programs

(程序的机器级表示)

Register


63
31
15
7
0
%rax
%eax
%ax
%al
%rsi
%esi
%si
%sil
%rbx
%ebx
%bx
%bl
%rdi
%edi
%di
%dil
%rcx
%ecx
%cx
%cl
%rbp
%ebp
%bp
%bpl
%rdx
%edx
%dx
%dl
%rsp
%esp
%sp
%spl


CPU
Register file
a
b
PC
System bus
Memory bus
c
Bus interface
I/O bridge
Main memory


%rax
%eax
%ax
%al
16

Register

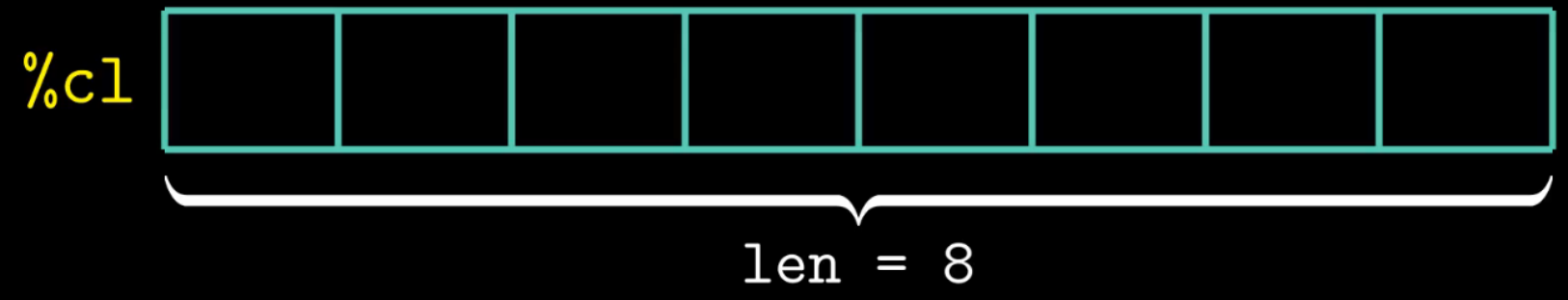
63 0

%rax	Return value-Caller saved
%rbx	Callee saved
%rcx	Argument #4 - Caller saved
%rdx	Argument #3 - Caller saved
%rsi	Argument #2 - Caller saved
%rdi	Argument #1 - Caller saved
%rbp	Callee saved
%rsp	Stack pointer

Shift Operations

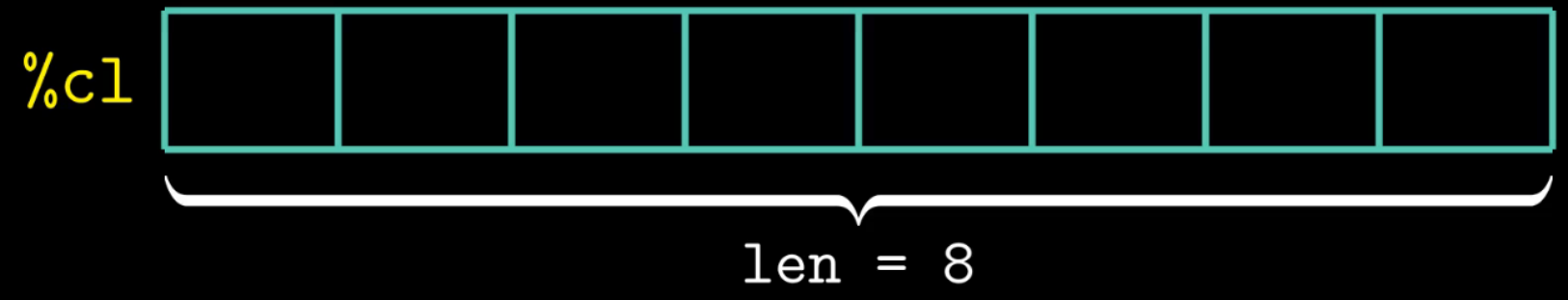
Instruction	Effect	Description
`SAL k, D`	`D` $\longleftarrow$ `D << k`	`Left shift`(左移)
`SHL k, D`	`D` $\longleftarrow$ `D << k`	`Left shift`(左移, 等同于 SAL)
`SAR k, D`	`D` $\longleftarrow >>_A k$	`Arithmetic right shift`(算术右移)
`SHR k, D`	`D` $\longleftarrow >>_L$ `k`	`Logical right shift`(逻辑右移)

Shift Amount(移位量)



For data length w: $2^m = w$

Shift Amount(移位量)



For data length w: $2^m = w$

Code

```
long arith(long x, long y long z) {
    long t1 = x^y;
    long t2 = z * 48;
    long t3 = t1 & 0x0F0F0F0F;
    long t4 = t2 - t3;
    return t4;
}
```

```
arith:
  xorq  %rsi, %rdi
  leaq (%rdx, %rdx, 2), %rax
  salq $4, %rax
  andl $252645135, %edi
  subq %rdi, %rax
  ret
```

Code

```
long t2 = z * 48;
```

```
leaq (%rdx, %rdx, 2), %rax
```

R[%rdx] + R[%rdx]*2 = 3 * z ⟶ %rax

```
salq $4, %rax
```

Code

```
long t2 = z * 48;
```

```
leaq (%rdx, %rdx, 2), %rax
```

R[%rdx] + R[%rdx]*2 = 3 * z ⟶ %rax

```
salq $4, %rax
```

2^4 * R[%rax] = 16 * (3*z) = 48 * z

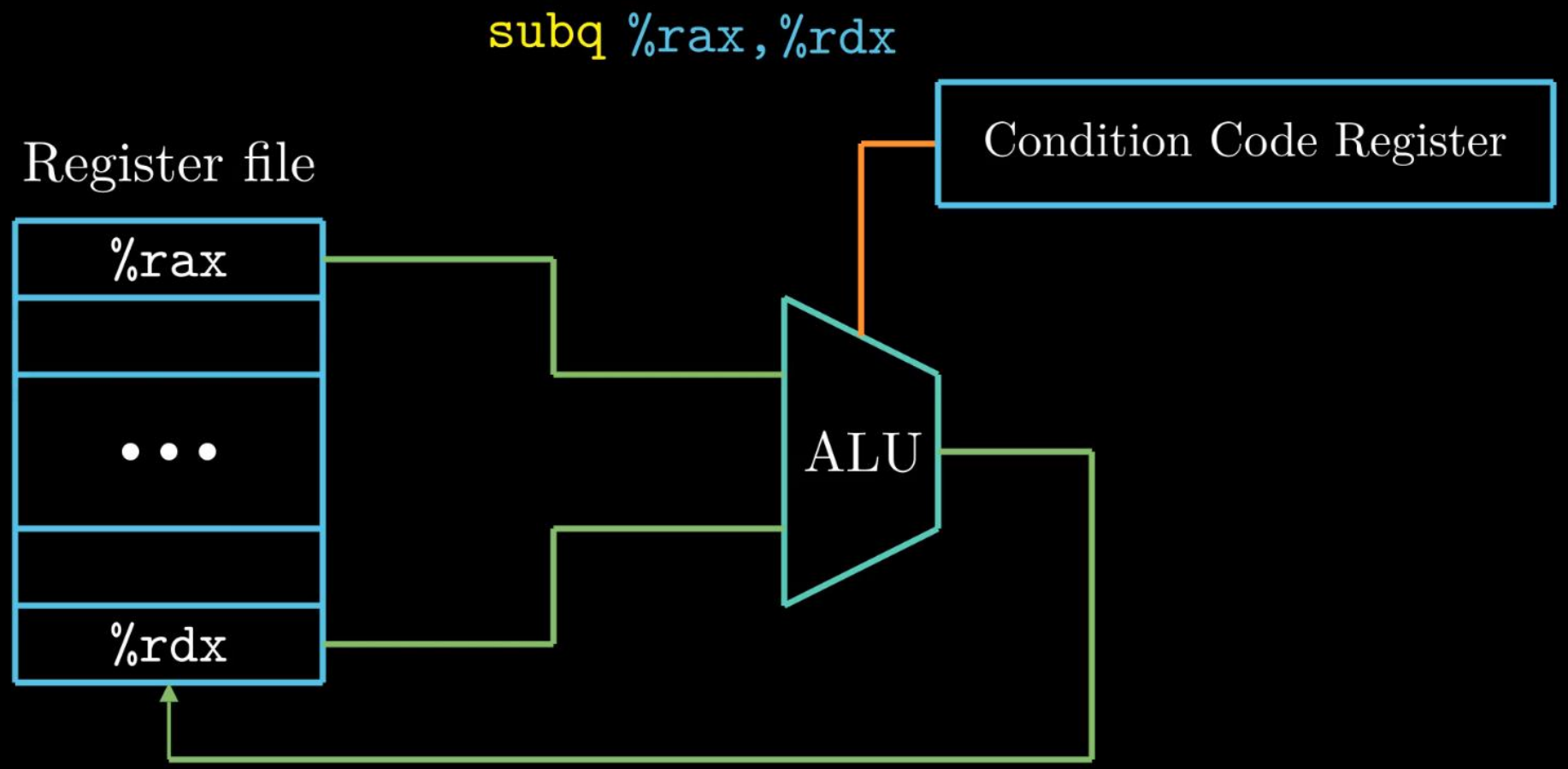
Special Arithmetic Operations

Instruction	Description
imulq S	Signed full multiply(有符号全乘法)
mulq S	Unsigned full multiply(无符号全乘法)
cqto	Convert to oct word(转换为八字)
idivq S	Signed divide(有符号除法)
divq S	Unsigned divide(无符号除法)

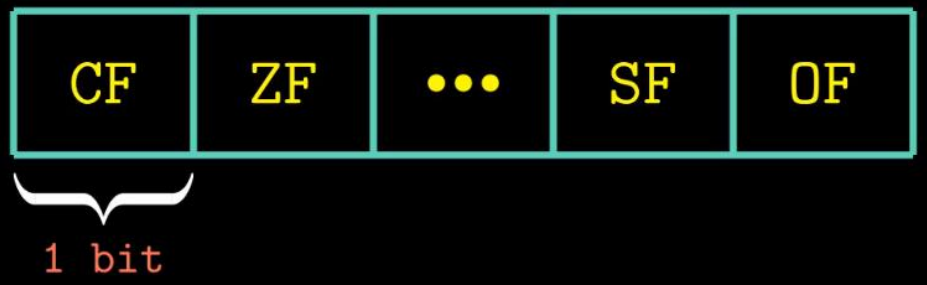
Control

```
if(test-expr)
   then-statement
else
   else-statement

init-expr;
while(test-expr) {
   body-statement
   update-expr;
 }
```

Control


subq %rax,%rdx
Condition Code Register
Register file
%rax
• • •
%rdx
ALU

Condition Code Register



Condition Code Register

t1: `addq %rax, %rbx`

t2: `subq %rcx, %rdx`

Condition Code Register

CF —— Carry Flag(进位标志)

```
unsigned char a = 255;
unsigned char b = 1;
unsigned char t = a + b;
```

⟶ CF = 1

Condition Code Register

CF —— Carry Flag(进位标志)

ZF —— Zero Flag(零标志)

```
int a = 1;
int b = -1;
int t = a + b;
```

⟶ ZF = 1

Condition Code Register

CF —— Carry Flag(进位标志)

ZF —— Zero Flag(零标志)

SF —— Sign Flag(符号标志)

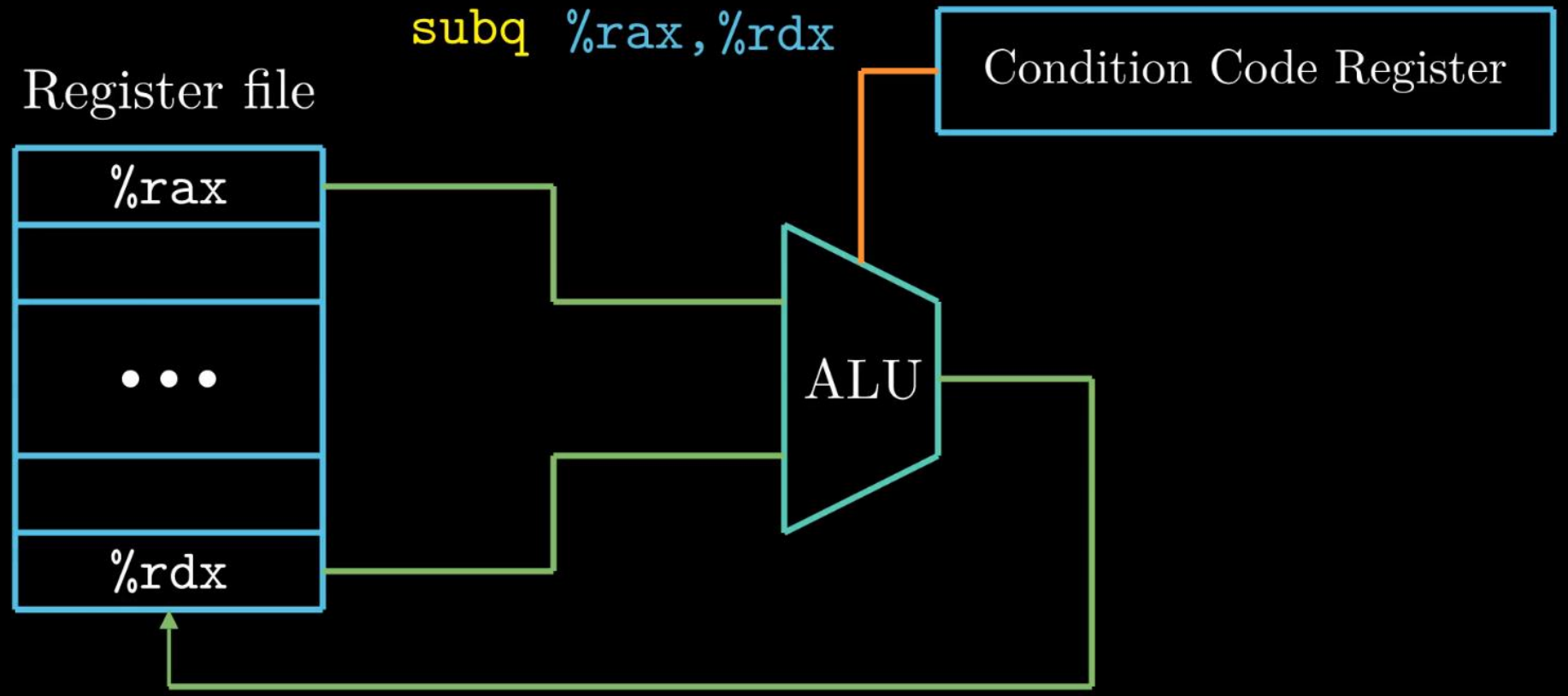
OF —— Overflow Flag(溢出标志)

Integer Arithmetic Operation

Unary Operations	Binary Operations	Shift Operations
INC D } OF,ZF	ADD S,D	SAL k,D
DEC D } OF,ZF	SUB S,D	SHL k,D
NEG D	IMUL S,D	SAR k,D
NOT D	OR S,D	SHR k,D
	XOR S,D → CF=0,OF=0	
	AND S,D	

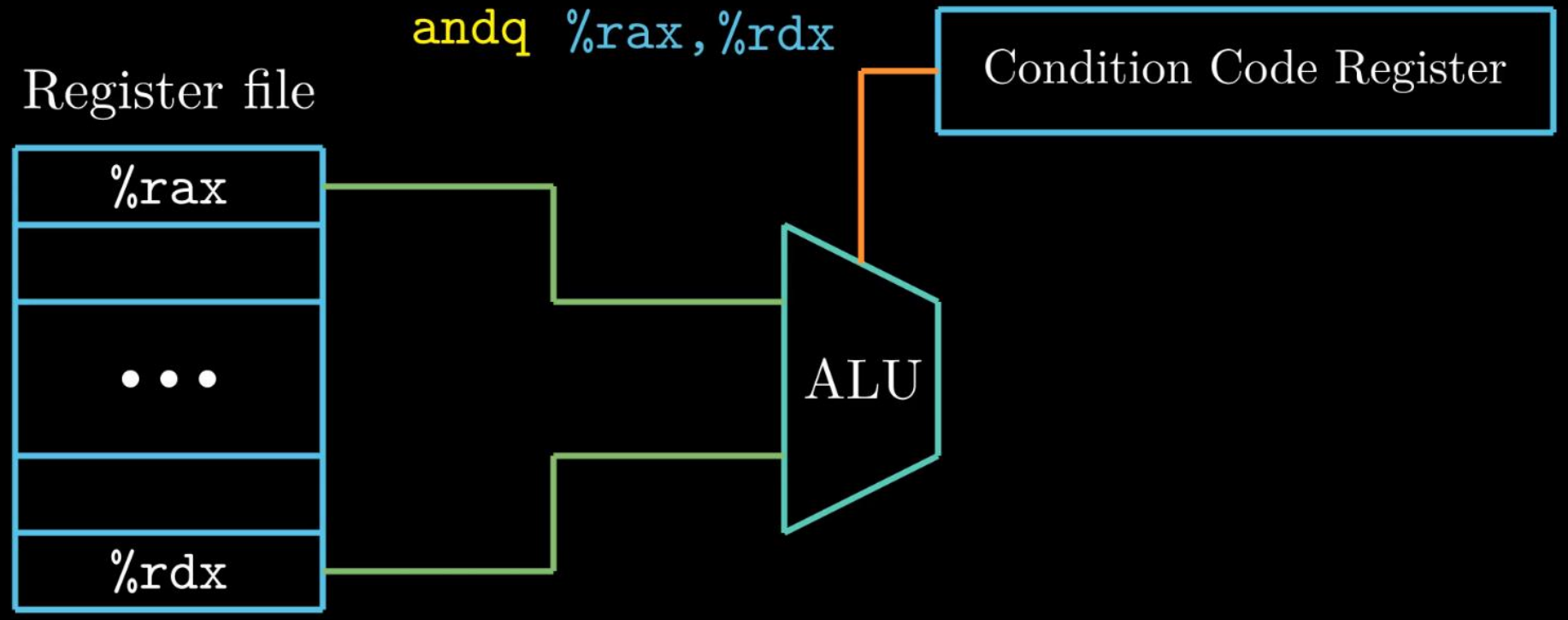
CMP and TEST

```
cmpq %rax,%rdx
subq %rax,%rdx
```



CMP and TEST

```
testq %rax,%rdx
andq %rax,%rdx
```



Condition Code

```
int comp(long a, long b) {
    return (a == b);
}
```

Condition Code

```
int comp(long a, long b) {
    return (a == b);
}
```

```
cmpq:
  a in %rdi, b in %rsi
  cmpq %rsi, %rdi
  sete %al
  movzbl %al, %eax
  ret
```

Condition Code


```
int comp(long a, long b) {
    return (a == b);
}
```

```
cmpq:
    a in %rdi, b in %rsi
    cmpq %rsi, %rdi  ⟶  a - b
    sete %al
    movzbl %al, %eax
    ret
```

Condition Code


```
int comp(long a, long b) {
    return (a == b);
}
```

```
cmpq:
    a in %rdi, b in %rsi
    cmpq %rsi, %rdi  ⟶  a - b
    sete %al
    if ZF=1, %al=1
    if ZF=0, %al=0
    movzbl %al, %eax
    ret
```

Condition Code

```
int comp(char a, char b) {
    return (a < b);
}
```

```
comp:
  cmpq %sil, %dil
  setl %al   l is short for less
  movzbl %al, %eax
  ret
```

Set Instruction

$$a < b \longrightarrow SF \hat{} OF$$

$$t = a - b$$

Case 1: $a < b$ $t < 0$ $SF = 1$ $SF \hat{} OF = 1$

Case 2: $a > b$ $t > 0$ $SF = 0$ $SF \hat{} OF = 0$

Set Instruction

$$a < b \longrightarrow SF \hat{} OF$$

$$t = a - b$$

Case 1: a < b t < 0 SF = 1 SF ^ OF =1

Case 2: a > b t > 0 SF = 0 SF ^ OF =0

Case 3: a < b a = -2 b = 127

t = 127 > 0, SF = 0 OF = 1, SF ^ OF =1

Set Instruction

$$a < b \longrightarrow SF \hat{} OF$$

$$t = a - b$$

Case 1: a < b t < 0 SF = 1 SF ^ OF =1

Case 2: a > b t > 0 SF = 0 SF ^ OF =0

Case 3: a < b a = -2 b = 127

t = 127 > 0, SF = 0 OF = 1, SF ^ OF =1

Case 4: a > b a = 1 b = -128

t = -127 < 0, SF = 1 OF = 1, SF ^ OF =0

Set Instructionis

Instruction	Effect	Description
setg D	D ⟵ ~(SF ^ OF) & ~ZF	Greater(signed >)
setge D	D ⟵ ~(SF ^ OF)	Greater or equal(signed >=)
setl D	D ⟵ SF ^ OF	Less(signed <)
setle D	D ⟵ (SF ^ OF) | ZF	Less or equal(signed <=)

Set Instructionis

Instruction	Effect	Description
`seta D`	`D ⟵ ~CF & ~ZF`	Above(unsigned >)
`setae D`	`D ⟵ ~CF`	Above or equal(unsigned >=)
`setb D`	`D ⟵ CF`	Below(unsigned <)
`setbe D`	`D ⟵ CF|ZF`	Below or equal(unsigned <=)

Jump Instructions

```
long absdiff_se(long x, long y)
{
    long result;
    if (x < y) {
        result = y - x;
    }
    else {
        result = x - y;
    }
    return result;
}
```

Jump Instructions

```
long absdiff_se(long x, long y)
{
    long result;
    if (x < y) {
        result = y - x;
    }
    else {
        result = x - y;
    }
    return result;
}
```

```
absdiff_se:
x in %rdi, y in %rsi
    cmpq %rsi, %rdi  set SF,OF
    jl .L4
    movq %rdi, %rax
    subq %rsi, %rax
    ret
.L4:
    movq %rsi, %rax
    subq %rdi, %rax
    ret
```

Jump Instructions

Instruction	Jump Condition	Description
`jg Label`	`~(SF^OF)&~ZF`	Greater(signed >)
`jge Label`	`~(SF^OF)`	Greater or equal(signed >=)
`jl Label`	`SF^OF`	Less(signed <)
`jle Label`	`(SF^OF)|ZF`	Less or equal(signed <=)

Conditional Control

```
long absdiff_se(long x, long y)
{
    long result;
    if (x < y) {
        result = y - x;
    }
    else {
        result = x - y;
    }
    return result;
}
```

```
long cmovdiff_se(long x, long y)
{
    long rval = y - x;
    long eval = x - y;
    long ntest = x >= y;
    if (ntest)
        rval = eval
    return rval;
}
```

Conditional Control

```
long cmovdiff_se(long x, long y)
{
    long rval = y - x;
    long eval = x - y;
    long ntest = x >= y;
    if (ntest)
        rval = eval
    return rval;
}
```

```
cmovdiff_se:
    x in %rdi, y in %rsi
    movq %rsi, %rdx
    subq %rdi, %rdx  ⟶ rval = y - x
    movq %rdi, %rax
    subq %rsi, %rax  ⟶ eval = x - y
    cmpq %rsi, %rdi
    cmovge %rdx, %rax
    ret
```

Conditional Control

```
cmpq %rsi, %rdi   ⟶  compare x:y
cmovge %rdx, %rax ⟶  ~(SF^OF)
```

Conditional Move Instructions

Instruction	Move Condition	Description
`cmovg S,R`	`~(SF^OF)&~ZF`	Greater(signed >)
`cmovge S,R`	`~(SF^OF)`	Greater or equal(signed >=)
`cmovl S,R`	`SF^OF`	Less(signed <)
`cmovle S,R`	`(SF^OF)|ZF`	Less or equal(signed <=)

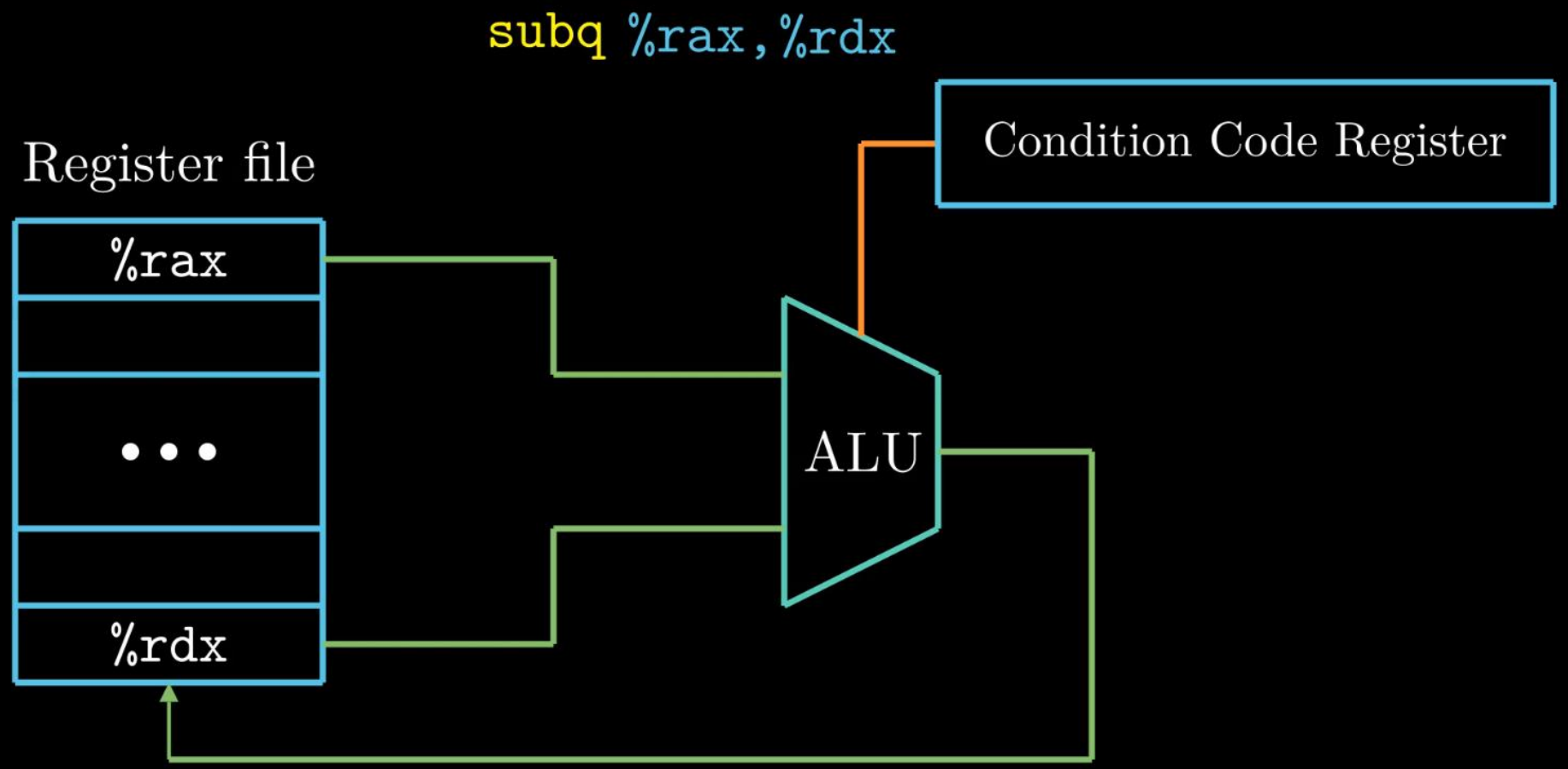
Computer Systems

A Programmer's Perspective

Chapter 3: Machine-Level Representation of Programs

(程序的机器级表示)

Control


subq %rax,%rdx
Condition Code Register
Register file
%rax
• • •
%rdx
ALU

Condition Code Register

CF —— Carry Flag(进位标志)

ZF —— Zero Flag(零标志)

SF —— Sign Flag(符号标志)

OF —— Overflow Flag(溢出标志)

Set Instructionis

Instruction	Effect	Description
setg D	D ⟵ ~(SF ^ OF) & ~ZF	Greater(signed >)
setge D	D ⟵ ~(SF ^ OF)	Greater or equal(signed >=)
setl D	D ⟵ SF ^ OF	Less(signed <)
setle D	D ⟵ (SF ^ OF) | ZF	Less or equal(signed <=)

Set Instructionis

Instruction	Effect	Description
`seta D`	`D ⟵ ~CF & ~ZF`	Above(unsigned >)
`setae D`	`D ⟵ ~CF`	Above or equal(unsigned >=)
`setb D`	`D ⟵ CF`	Below(unsigned <)
`setbe D`	`D ⟵ CF|ZF`	Below or equal(unsigned <=)

Jump Instructions

```
long absdiff_se(long x, long y)
{
    long result;
    if (x < y) {
        result = y - x;
    }
    else {
        result = x - y;
    }
    return result;
}
```

Jump Instructions

```
long absdiff_se(long x, long y)
{
    long result;
    if (x < y) {
        result = y - x;
    }
    else {
        result = x - y;
    }
    return result;
}
```

```
absdiff_se:
x in %rdi, y in %rsi
    cmpq %rsi, %rdi  set SF,OF
    jl .L4
    movq %rdi, %rax
    subq %rsi, %rax
    ret
.L4:
    movq %rsi, %rax
    subq %rdi, %rax
    ret
```

Jump Instructions

Instruction	Jump Condition	Description
`jg Label`	`~(SF^OF)&~ZF`	Greater(signed >)
`jge Label`	`~(SF^OF)`	Greater or equal(signed >=)
`jl Label`	`SF^OF`	Less(signed <)
`jle Label`	`(SF^OF)|ZF`	Less or equal(signed <=)

Conditional Control

```
long absdiff_se(long x, long y)
{
    long result;
    if (x < y) {
        result = y - x;
    }
    else {
        result = x - y;
    }
    return result;
}
```

```
long cmovdiff_se(long x, long y)
{
    long rval = y - x;
    long eval = x - y;
    long ntest = x >= y;
    if (ntest)
        rval = eval
    return rval;
}
```

Conditional Control

```
long cmovdiff_se(long x, long y)
{
    long rval = y - x;
    long eval = x - y;
    long ntest = x >= y;
    if (ntest)
        rval = eval
    return rval;
}
```

```
cmovdiff_se:
    x in %rdi, y in %rsi
    movq %rsi, %rdx
    subq %rdi, %rdx  ⟶ rval = y - x
    movq %rdi, %rax
    subq %rsi, %rax  ⟶ eval = x - y
    cmpq %rsi, %rdi
    cmovge %rdx, %rax
    ret
```

Conditional Control

```
cmpq %rsi, %rdi   ⟶  compare x:y
cmovge %rdx, %rax ⟶  ~(SF^OF)
```

Conditional Move Instructions

Instruction	Move Condition	Description
`cmovg S,R`	`~(SF^OF)&~ZF`	Greater(signed >)
`cmovge S,R`	`~(SF^OF)`	Greater or equal(signed >=)
`cmovl S,R`	`SF^OF`	Less(signed <)
`cmovle S,R`	`(SF^OF)|ZF`	Less or equal(signed <=)

Loops

```
do
    body-statement
while(test-expr);
```

```
while(test-expr)
    body-statement
```

```
for(init-expr; test-expr; update-expr)
        body-statement
```

Loops

```
long fact_do(long n)
{
    long result = 1;
    do {
        result *= n;
        n = n - 1;
    } while (n > 1);
    return result;
}
```

```
fact_do:
    n in %rdi
    movl $1, %eax
.L2:
    imulq %rdi, %rax
    subq $1, %rdi
    cmpq $1, %rdi
    jg .L2
    rep ret
```

Jump Instructions

Instruction	Jump Condition	Description
`jg Label`	`~(SF^OF)&~ZF`	Greater(signed >)
`jge Label`	`~(SF^OF)`	Greater or equal(signed >=)
`jl Label`	`SF^OF`	Less(signed <)
`jle Label`	`(SF^OF)|ZF`	Less or equal(signed <=)

Loops

```
long fact_do(long n)
{
      long result = 1;
      do {
          result *= n;
          n = n - 1;
      } while (n > 1);
      return result;
}
```

```
long fact_while(long n)
{
    long result = 1;
    while (n > 1) {
        result *= n;
        n = n - 1;
    }
    return result;
}
```

For Loops

```
for(init-expr; test-expr; update-expr)
                body-statement


init-expr;
while(test-expr) {
    body-statement
    update-expr
}
```

Loops

```
long fact_for(long n)
{
        long i;
        long result = 1;
        for (i = 2; i <= n; i++)
            result *= i;
        return result;
}
```

Loops

```
long fact_for(long n)
{
      long i;
      long result = 1;
      for (i = 2; i <= n; i++)
          result *= i;
      return result;
}
```

```
long fact_for_while(long n)
{
   long i = 2;
   long result = 1;
   while (i <= n) {
       result *= i;
       i++;
    }
   return result;
}
```

Switch Code

```
void switch_eg(long x, long n,
               long *dest)
{
    long val = x;

    switch (n) {
    case 0:
        val *= 13;
        break;
    case 2:
        val += 10;
    case 3:
        val += 11;
        break;
    case 4:
    case 6:
        val += 11;
        break;
    default:
        val = 0;
    }
    *dest = val
}
```

Switch Code

```
void switch_eg(long x, long n,
               long *dest)
{
    long val = x;

    switch (n) {
    case 0:
        val *= 13;
        break;
    case 2:
        val += 10;
```

```
switch_eg:
  n in %rsi
   cmpq $6, %rsi
   ja .L8
   leaq .L4(%rip), %rcx
   movslq (%rcx,%rsi,4), %rax
   addq %rcx, %rax
   jmp *%rax
```

Switch Code

```
void switch_eg(long x, long n,
               long *dest)
{
    long val = x;

    switch (n) {
    case 0:
        val *= 13;
        break;
    case 2:
        val += 10;
```

```
.L4:
  .long: .L3-.L4 ⟶ Case 0
  .long: .L8-.L4 ⟶ Case 1
  .long: .L5-.L4 ⟶ Case 2
  .long: .L6-.L4 ⟶ Case 3
  .long: .L7-.L4 ⟶ Case 4
  .long: .L8-.L4 ⟶ Case 5
  .long: .L7-.L4 ⟶ Case 6
```

Computer Systems: A Programmer's Perspective, 3/E (CS:APP3e)

Randal E. Bryant and David R. O'Hallaron, Carnegie Mellon University

Lab Assignments

This page contains a complete set of turnkey labs for the CS:APP3e text. The labs all share some common features. Each lab is distributed in a self-contained tar file. You will need a CS:APP account to download the code. To untar `foo.tar`, type "`tar xvf foo.tar`" to the Unix shell. This will create a directory called "`foo`" that contains all of the material for the lab.

Handout directories for each lab (without solutions) are available to students who are using the book for self-study and who want to work on the labs. Solutions are provided only to instructors.

- *Data Lab* ***[Updated 12/16/19]*** (README, Writeup, Release Notes, Self-Study Handout) NEW

  Students implement simple logical, two's complement, and floating point functions, but using a highly restricted subset of C. For example, they might be asked to compute the absolute value of a number using only bit-level operations and straightline code. This lab helps students understand the bit-level representations of C data types and the bit-level behavior of the operations on data.

- *Bomb Lab [Updated 1/12/16]* (README, Writeup, Release Notes, Self-Study Handout)

  A "binary bomb" is a program provided to students as an object code file. When run, it prompts the user to type in 6 different strings. If any of these is incorrect, the bomb "explodes," printing an error message and logging the event on a grading server. Students must "defuse" their own unique bomb by disassembling and reverse engineering the program to determine what the 6 strings should be. The lab teaches students to understand assembly language, and also forces them to learn how to use a debugger. It's also great fun. A legendary lab among the CMU undergrads.

  Here's a Linux/x86-64 binary bomb that you can try out for yourself. The feature

RECENT ARTICLES FROM THE CS:APP BLOG

```
csapp@jiuqulangan:~/lab$ ls
bomb.tar
csapp@jiuqulangan:~/lab$ tar -xvf bomb.tar
bomb/
bomb/bomb
bomb/bomb.c
bomb/README
csapp@jiuqulangan:~/lab$ cd bomb/
csapp@jiuqulangan:~/lab/bomb$ ls
bomb  bomb.c  README
csapp@jiuqulangan:~/lab/bomb$
```

```
55          printf("%s: Error: Couldn't open %s\n", argv[0], argv[1]);
56          exit(8);
57      }
58      }
59
60      /* You can't call the bomb with more than 1 command line argument. */
61      else {
62      printf("Usage: %s [<input_file>]\n", argv[0]);
63      exit(8);
64      }
65
66      /* Do all sorts of secret stuff that makes the bomb harder to defuse. */
67      initialize_bomb();
68
69      printf("Welcome to my fiendish little bomb. You have 6 phases with\n");
70      printf("which to blow yourself up. Have a nice day!\n");
71
72      /* Hmm...  Six phases must be more secure than one phase! */
73      input = read_line();             /* Get input                   */
74      phase_1(input);                  /* Run the phase               */
75      phase_defused();                 /* Drat!  They figured it out!
76                       * Let me know how they did it. */
77      printf("Phase 1 defused. How about the next one?\n");
78
79      /* The second phase is harder.  No one will ever figure out
80       * how to defuse this... */
81      input = read_line();
82      phase_2(input);
83      phase_defused();
84      printf("That's number 2.  Keep going!\n");
```

```
csapp@jiuqulangan:~/lab$ ls
bomb.tar
csapp@jiuqulangan:~/lab$ tar -xvf bomb.tar
bomb/
bomb/bomb
bomb/bomb.c
bomb/README
csapp@jiuqulangan:~/lab$ cd bomb/
csapp@jiuqulangan:~/lab/bomb$ ls
bomb  bomb.c  README
csapp@jiuqulangan:~/lab/bomb$ vim bomb.c
csapp@jiuqulangan:~/lab/bomb$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
qwer

BOOM!!!
The bomb has blown up.
csapp@jiuqulangan:~/lab/bomb$
```

```
csapp@jiuqulangan:~/lab$ ls
bomb.tar
csapp@jiuqulangan:~/lab$ tar -xvf bomb.tar
bomb/
bomb/bomb
bomb/bomb.c
bomb/README
csapp@jiuqulangan:~/lab$ cd bomb/
csapp@jiuqulangan:~/lab/bomb$ ls
bomb  bomb.c  README
csapp@jiuqulangan:~/lab/bomb$ vim bomb.c
csapp@jiuqulangan:~/lab/bomb$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
qwer

BOOM!!!
The bomb has blown up.
csapp@jiuqulangan:~/lab/bomb$ objdump -d bomb > bomb.s
csapp@jiuqulangan:~/lab/bomb$ ls
bomb  bomb.c  bomb.s  README
csapp@jiuqulangan:~/lab/bomb$
```

```
59
60      /* You can't call the bomb with more than 1 command line argument. */
61      else {
62      printf("Usage: %s [<input_file>]\n", argv[0]);
63      exit(8);
64      }
65
66      /* Do all sorts of secret stuff that makes the bomb harder to defuse. */
67      initialize_bomb();
68
69      printf("Welcome to my fiendish little bomb. You have 6 phases with\n");
70      printf("which to blow yourself up. Have a nice day!\n");
71
72      /* Hmm...  Six phases must be more secure than one phase! */
73      input = read_line();             /* Get input                   */
74      phase_1(input);                  /* Run the phase               */
75      phase_defused();                 /* Drat!  They figured it out!
76                       * Let me know how they did it. */
77      printf("Phase 1 defused. How about the next one?\n");
78
79      /* The second phase is harder.  No one will ever figure out
80       * how to defuse this... */
81      input = read_line();
82      phase_2(input);
83      phase_defused();
84      printf("That's number 2.  Keep going!\n");
85
86      /* I guess this is too easy so far.  Some more complex code will
87       * confuse people. */
88      input = read_line();
```

```
400d7a:   b8 00 00 00 00          mov    $0x0,%eax
400d7f:   48 85 c0                test   %rax,%rax
400d82:   74 14                   je     400d98 <frame_dummy+0x28>
400d84:   55                      push   %rbp
400d85:   bf 08 2e 60 00          mov    $0x602e08,%edi
400d8a:   48 89 e5                mov    %rsp,%rbp
400d8d:   ff d0                   callq  *%rax
400d8f:   5d                      pop    %rbp
400d90:   e9 7b ff ff ff          jmpq   400d10 <register_tm_clones>
400d95:   0f 1f 00                nopl   (%rax)
400d98:   e9 73 ff ff ff          jmpq   400d10 <register_tm_clones>
400d9d:   90                      nop
400d9e:   90                      nop
400d9f:   90                      nop

0000000000400da0 <main>:
400da0:   53                      push   %rbx
400da1:   83 ff 01                cmp    $0x1,%edi
400da4:   75 10                   jne    400db6 <main+0x16>
400da6:   48 8b 05 9b 29 20 00    mov    0x20299b(%rip),%rax        # 603748 <stdin@@GLIBC_2.2.5>
400dad:   48 89 05 b4 29 20 00    mov    %rax,0x2029b4(%rip)        # 603768 <infile>
400db4:   eb 63                   jmp    400e19 <main+0x79>
400db6:   48 89 f3                mov    %rsi,%rbx
400db9:   83 ff 02                cmp    $0x2,%edi
400dbc:   75 3a                   jne    400df8 <main+0x58>
400dbe:   48 8b 7e 08             mov    0x8(%rsi),%rdi
400dc2:   be b4 22 40 00          mov    $0x4022b4,%esi
400dc7:   e8 44 fe ff ff          callq  400c10 <fopen@plt>
400dcc:   48 89 05 95 29 20 00    mov    %rax,0x202995(%rip)        # 603768 <infile>
400dd3:   48 85 c0                test   %rax,%rax
```

```
400df3:   e8 28 fe ff ff          callq  400c20 <exit@plt>
400df8:   48 8b 16                mov    (%rsi),%rdx
400dfb:   be d3 22 40 00          mov    $0x4022d3,%esi
400e00:   bf 01 00 00 00          mov    $0x1,%edi
400e05:   b8 00 00 00 00          mov    $0x0,%eax
400e0a:   e8 f1 fd ff ff          callq  400c00 <__printf_chk@plt>
400e0f:   bf 08 00 00 00          mov    $0x8,%edi
400e14:   e8 07 fe ff ff          callq  400c20 <exit@plt>
400e19:   e8 84 05 00 00          callq  4013a2 <initialize_bomb>
400e1e:   bf 38 23 40 00          mov    $0x402338,%edi
400e23:   e8 e8 fc ff ff          callq  400b10 <puts@plt>
400e28:   bf 78 23 40 00          mov    $0x402378,%edi
400e2d:   e8 de fc ff ff          callq  400b10 <puts@plt>
400e32:   e8 67 06 00 00          callq  40149e <read_line>
400e37:   48 89 c7                mov    %rax,%rdi
400e3a:   e8 a1 00 00 00          callq  400ee0 <phase_1>
400e3f:   e8 80 07 00 00          callq  4015c4 <phase_defused>
400e44:   bf a8 23 40 00          mov    $0x4023a8,%edi
400e49:   e8 c2 fc ff ff          callq  400b10 <puts@plt>
400e4e:   e8 4b 06 00 00          callq  40149e <read_line>
400e53:   48 89 c7                mov    %rax,%rdi
400e56:   e8 a1 00 00 00          callq  400efc <phase_2>
400e5b:   e8 64 07 00 00          callq  4015c4 <phase_defused>
400e60:   bf ed 22 40 00          mov    $0x4022ed,%edi
400e65:   e8 a6 fc ff ff          callq  400b10 <puts@plt>
400e6a:   e8 2f 06 00 00          callq  40149e <read_line>
400e6f:   48 89 c7                mov    %rax,%rdi
400e72:   e8 cc 00 00 00          callq  400f43 <phase_3>
400e77:   e8 48 07 00 00          callq  4015c4 <phase_defused>
400e7c:   bf 0b 23 40 00          mov    $0x40230b,%edi
400e81:   e8 8a fc ff ff          callq  400b10 <puts@plt>
```

```
400ec6:   e8 29 02 00 00          callq  4010f4 <phase_6>
400ecb:   e8 f4 06 00 00          callq  4015c4 <phase_defused>
400ed0:   b8 00 00 00 00          mov    $0x0,%eax
400ed5:   5b                      pop    %rbx
400ed6:   c3                      retq
400ed7:   90                      nop
400ed8:   90                      nop
400ed9:   90                      nop
400eda:   90                      nop
400edb:   90                      nop
400edc:   90                      nop
400edd:   90                      nop
400ede:   90                      nop
400edf:   90                      nop

0000000000400ee0 <phase_1>:
400ee0:   48 83 ec 08             sub    $0x8,%rsp
400ee4:   be 00 24 40 00          mov    $0x402400,%esi
400ee9:   e8 4a 04 00 00          callq  401338 <strings_not_equal>
400eee:   85 c0                   test   %eax,%eax
400ef0:   74 05                   je     400ef7 <phase_1+0x17>
400ef2:   e8 43 05 00 00          callq  40143a <explode_bomb>
400ef7:   48 83 c4 08             add    $0x8,%rsp
400efb:   c3                      retq

0000000000400efc <phase_2>:
400efc:   55                      push   %rbp
400efd:   53                      push   %rbx
400efe:   48 83 ec 28             sub    $0x28,%rsp
400f02:   48 89 e6                mov    %rsp,%rsi
```

Phase_1

```
0000000000400ee0 <phase_1>:
  400ee0:    sub  $0x8, %rsp
  400ee4:    mov  $0x402400, %esi
  400ee9:    callq  401338 <strings_not_equal>
  400eee:    test  %eax, %eax
  400ef0:    je  400ef7
  400ef2:    callq  40143a <explode_bomb>
  400ef7:    add  $0x8, %rsp
  400efb:    retq
```

Phase_1

```
0000000000400ee0 <phase_1>:
  400ee0:    sub  $0x8, %rsp
  400ee4:    mov  $0x402400, %esi
  400ee9:    callq  401338 <strings_not_equal>
  400eee:    test  %eax, %eax
  400ef0:    je  400ef7
```

```
int strings_not_equal(char *input, char *key)
if input == key, return 0    eax = 0
if input != key, return !0   eax != 0
```

Phase_1

```
0000000000400ee0 <phase_1>:
   400ee0:    sub  $0x8, %rsp
   400ee4:    mov  $0x402400, %esi
   400ee9:    callq  401338 <strings_not_equal>
   400eee:    test  %eax, %eax
   400ef0:    je  400ef7
```

```
int strings_not_equal(char *input, char *key)
```

arg1 -> %edi, arg2 -> %esi

input -> %edi, key -> %esi

```
csapp@jiuqulangan:~/lab/bomb$ ls
bomb  bomb.c  bomb.s  README
csapp@jiuqulangan:~/lab/bomb$ gdb bomb
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...done.
(gdb) b phase_1
Breakpoint 1 at 0x400ee0
(gdb) r
Starting program: /home/csapp/lab/bomb/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
qwert

Breakpoint 1, 0x0000000000400ee0 in phase_1 ()
(gdb) x/s 0x402400
0x402400:       "Border relations with Canada have never been better."
(gdb)
```

```
csapp@jiuqulangan:~/lab/bomb$ ls
bomb  bomb.c  bomb.s  README
csapp@jiuqulangan:~/lab/bomb$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
```

Computer Systems

A Programmer's Perspective

Chapter 3: Machine-Level Representation of Programs

(程序的机器级表示)

Register


63
31
15
7
0
%rax
%eax
%ax
%al
%rsi
%esi
%si
%sil
%rbx
%ebx
%bx
%bl
%rdi
%edi
%di
%dil
%rcx
%ecx
%cx
%cl
%rbp
%ebp
%bp
%bpl
%rdx
%edx
%dx
%dl
%rsp
%esp
%sp
%spl

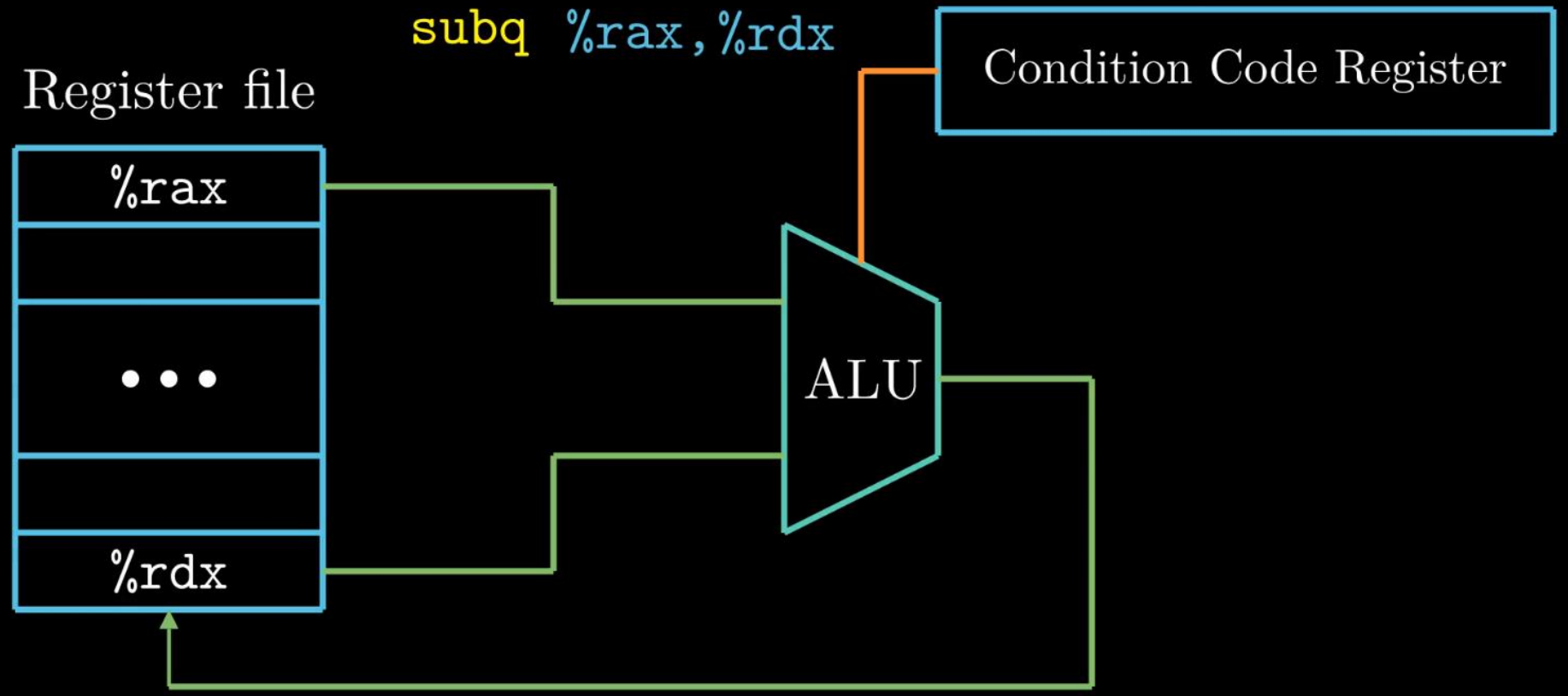
Register

63 0

%rax	Return value-Caller saved
%rbx	Callee saved
%rcx	Argument #4 - Caller saved
%rdx	Argument #3 - Caller saved
%rsi	Argument #2 - Caller saved
%rdi	Argument #1 - Caller saved
%rbp	Callee saved
%rsp	Stack pointer

CMP and TEST

```
cmpq %rax,%rdx
subq %rax,%rdx
```



Condition Code Register

CF —— Carry Flag(进位标志)

ZF —— Zero Flag(零标志)

SF —— Sign Flag(符号标志)

OF —— Overflow Flag(溢出标志)

Procedures(过程)

Procedures(过程)

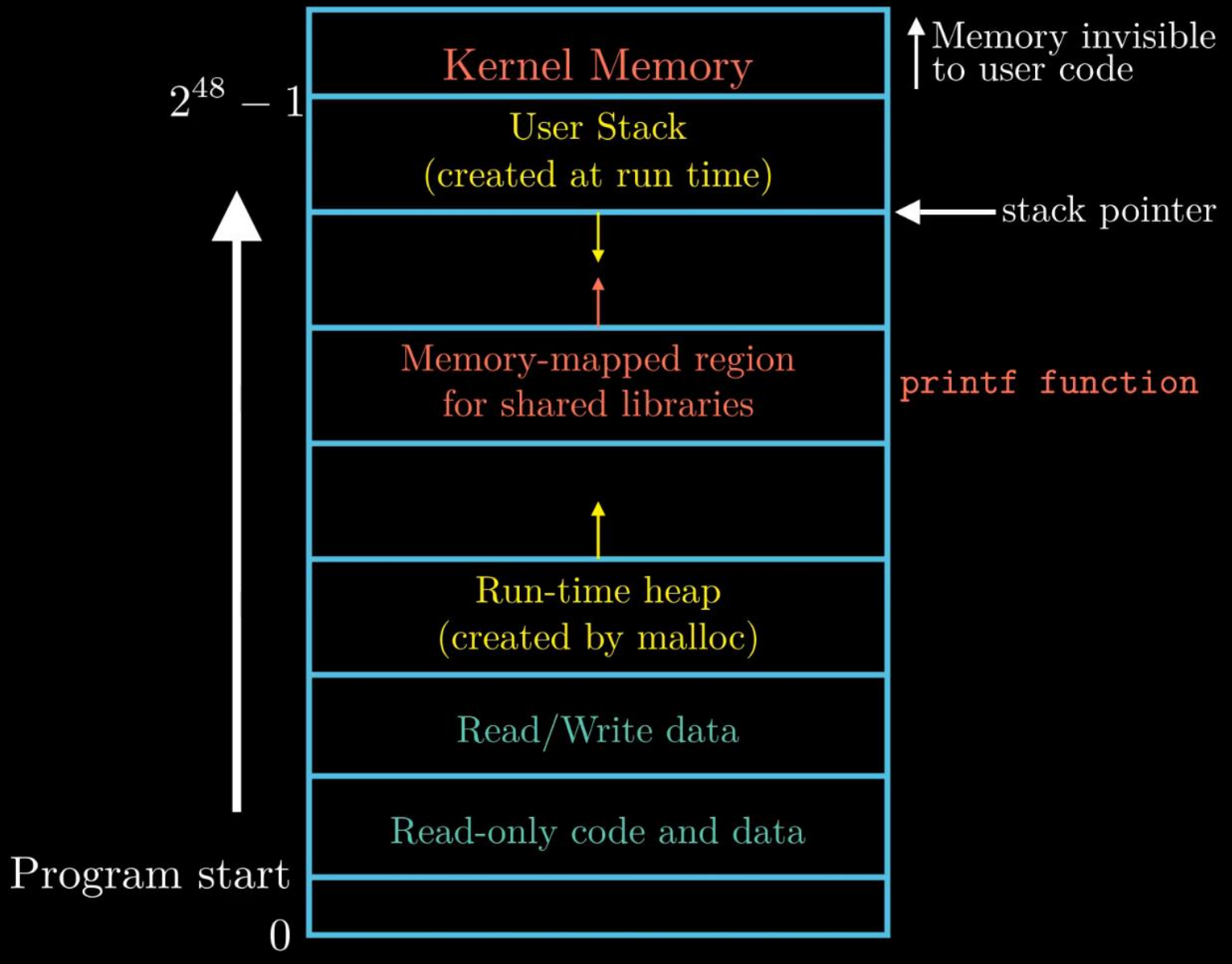
C Programming Language: Function(函数)

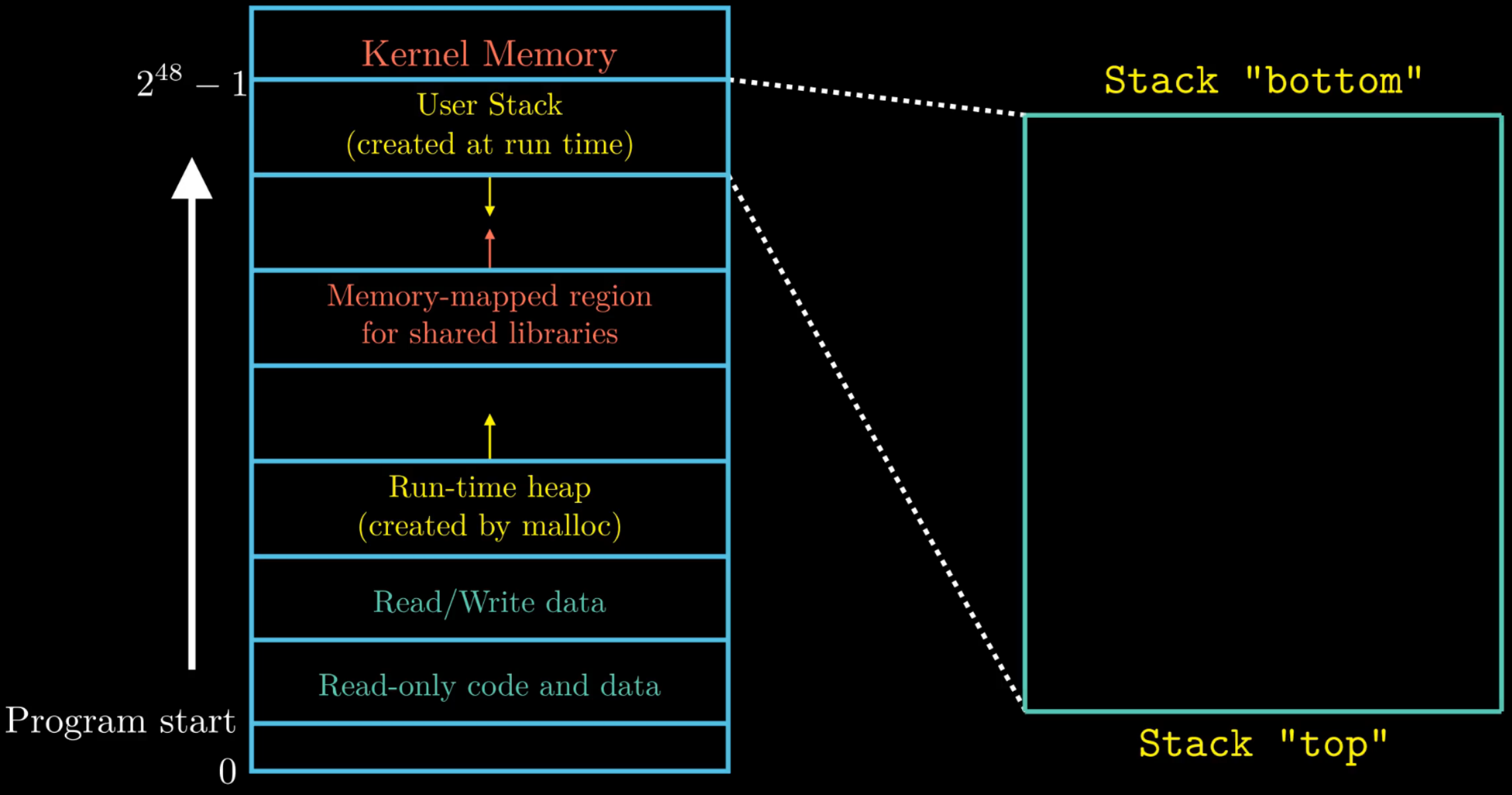
Java Programming Language: Method(方法)

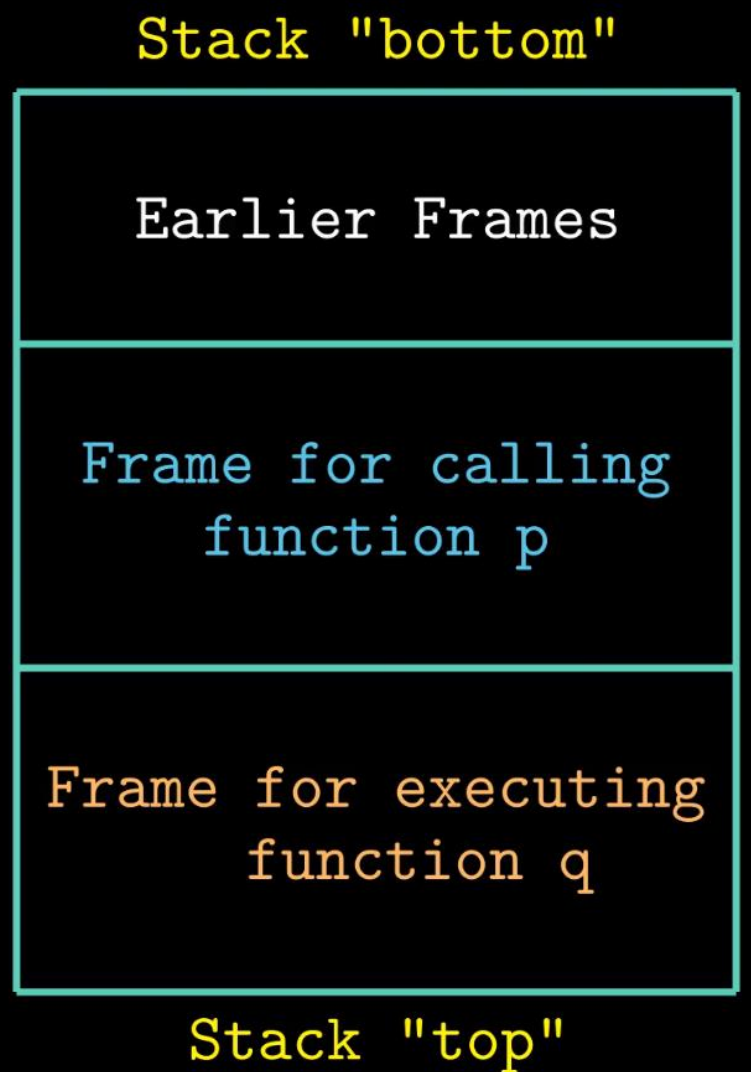
Procedures

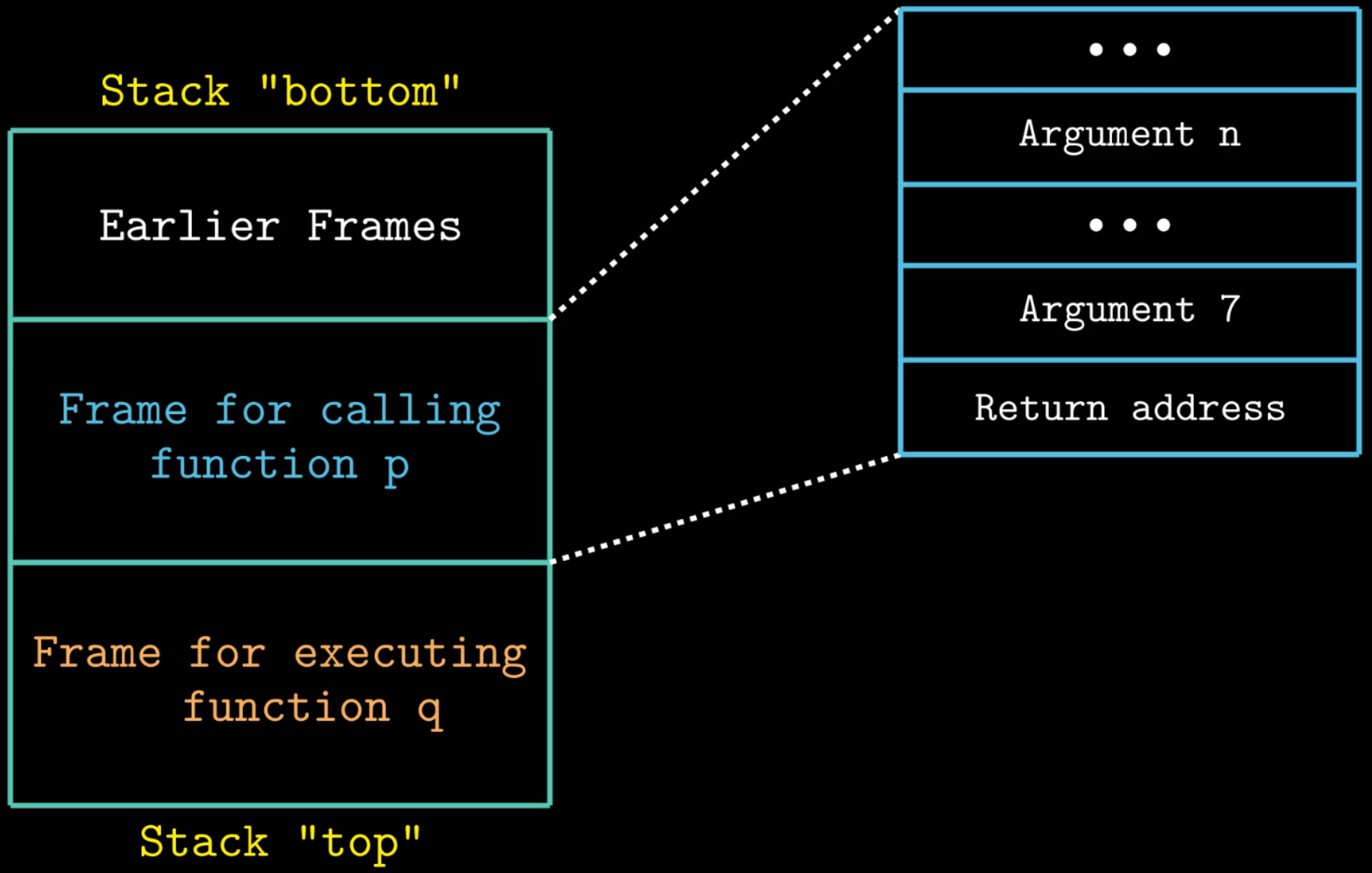
```
long P()
{
    ...
}
long Q()
{
    ...
}
```

- Passing control (传递控制)
- Passing data (传递数据)
- Allocating and deallocating memory (分配和释放内存)

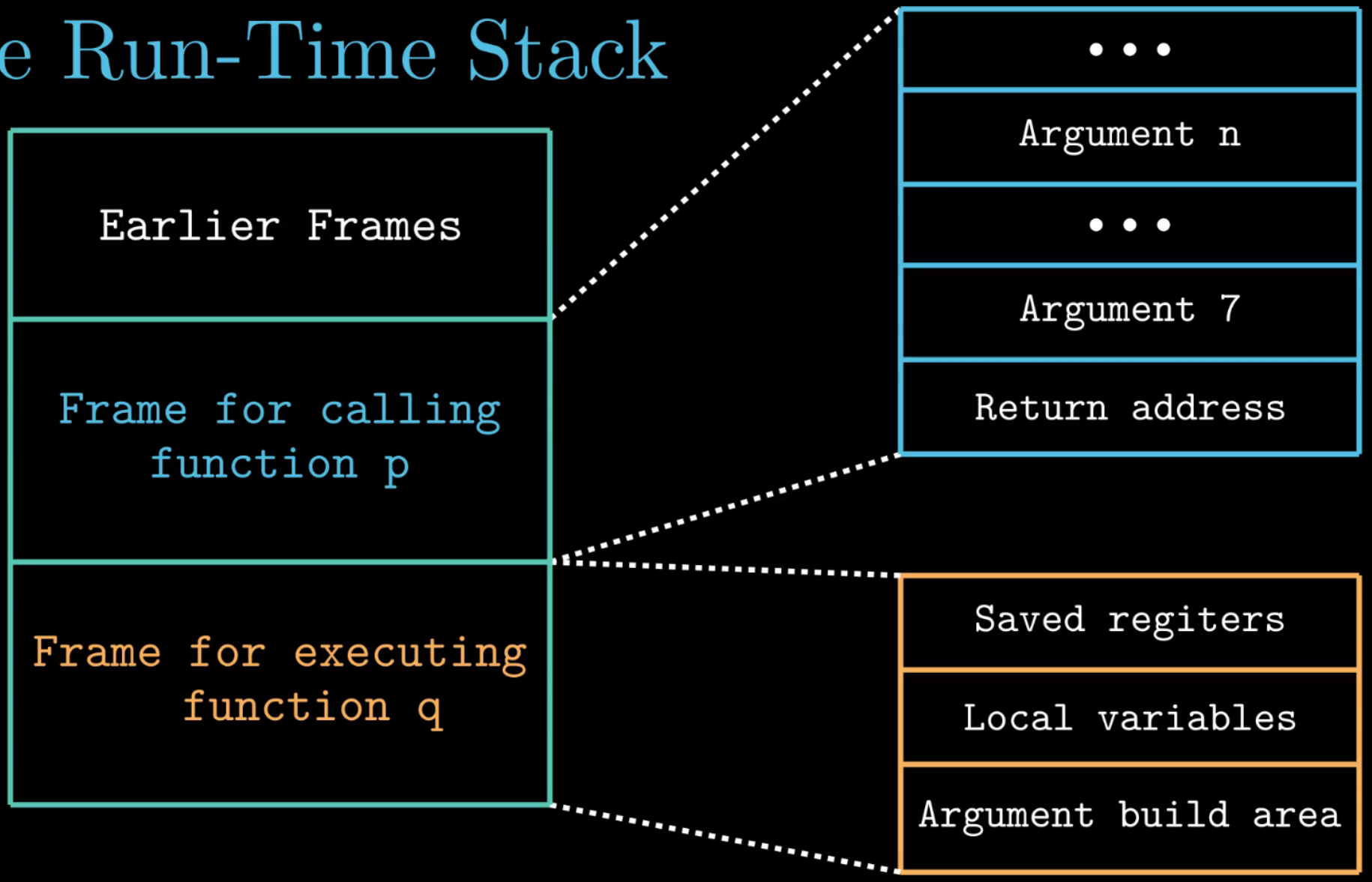

$2^{48} - 1$
Kernel Memory
Memory invisible to user code
User Stack
(created at run time)
stack pointer
Memory-mapped region
for shared libraries
printf function
Run-time heap
(created by malloc)
Read/Write data
Read-only code and data
Program start
0


Kernel Memory
$2^{48} - 1$
User Stack
(created at run time)
Memory-mapped region
for shared libraries
Run-time heap
(created by malloc)
Read/Write data
Read-only code and data
Program start
0
Stack "bottom"
Stack "top"


Stack "bottom"
Earlier Frames
Frame for calling function p
Frame for executing function q
Stack "top"


Stack "bottom"
Earlier Frames
Frame for calling function p
Frame for executing function q
Stack "top"
• • •
Argument n
• • •
Argument 7
Return address

The Run-Time Stack



Code File

main.c

```c
#include <stdio.h>

void multstore(long, long, long *);

int main() {
    long d;
    multstore(2, 3, &d);
    printf("2 * 3 --> %ld\n", d);
    return 0;
}

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

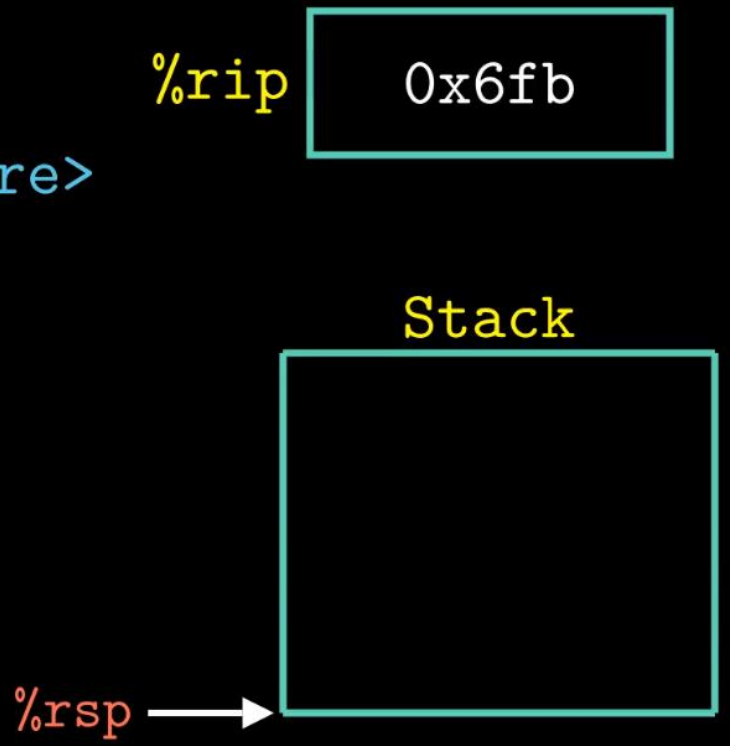
mstore.c

```c
long mult2(long, long);

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

```
linux> gcc -Og -o prog main.c mstore.c
linux> objdump -d proc
```

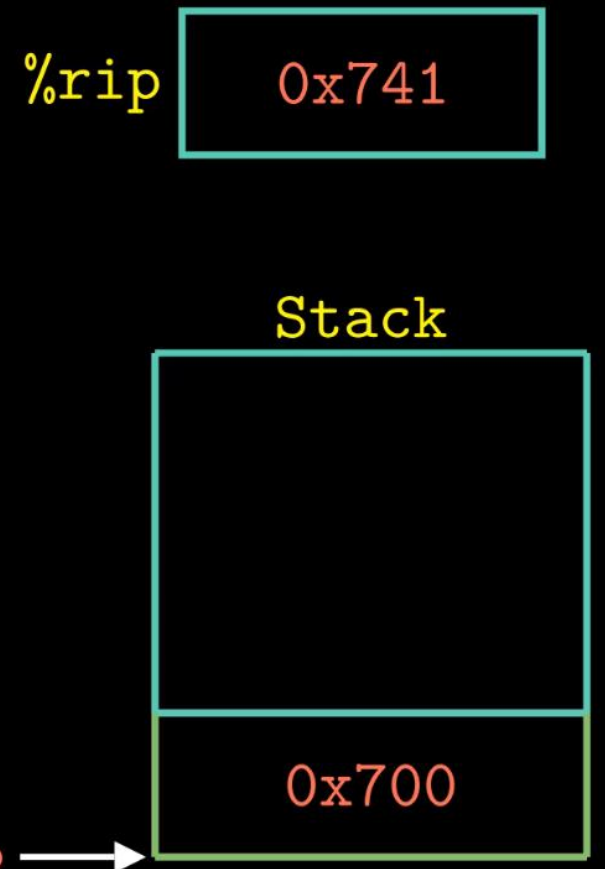
Code File

```
0000000000006da <main>:
  •••
→ 6fb: e8 41 00 00 00     callq  741 <multstore>
  700: 48 8b 14 24        mov  (%rsp), %rdx

0000000000000741 <multstore>:
  •••
  741: 53                 push  %rbx
  742: 48 89 d3           mov  %rdx, %rbx
```



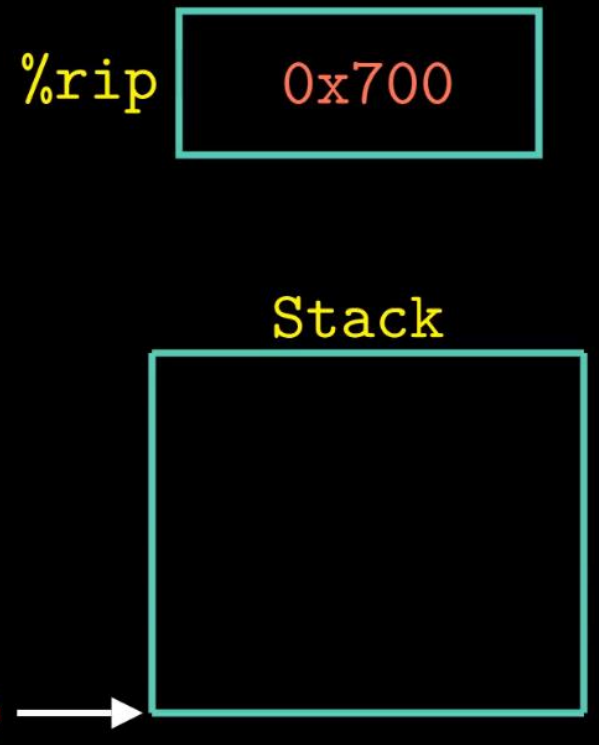
Code File

```
0000000000006da <main>:
  •••
6fb: e8 41 00 00 00     callq  741 <multstore>
700: 48 8b 14 24        mov (%rsp), %rdx

0000000000000741 <multstore>:
  •••
741: 53                 push %rbx
742: 48 89 d3           mov %rdx, %rbx
```

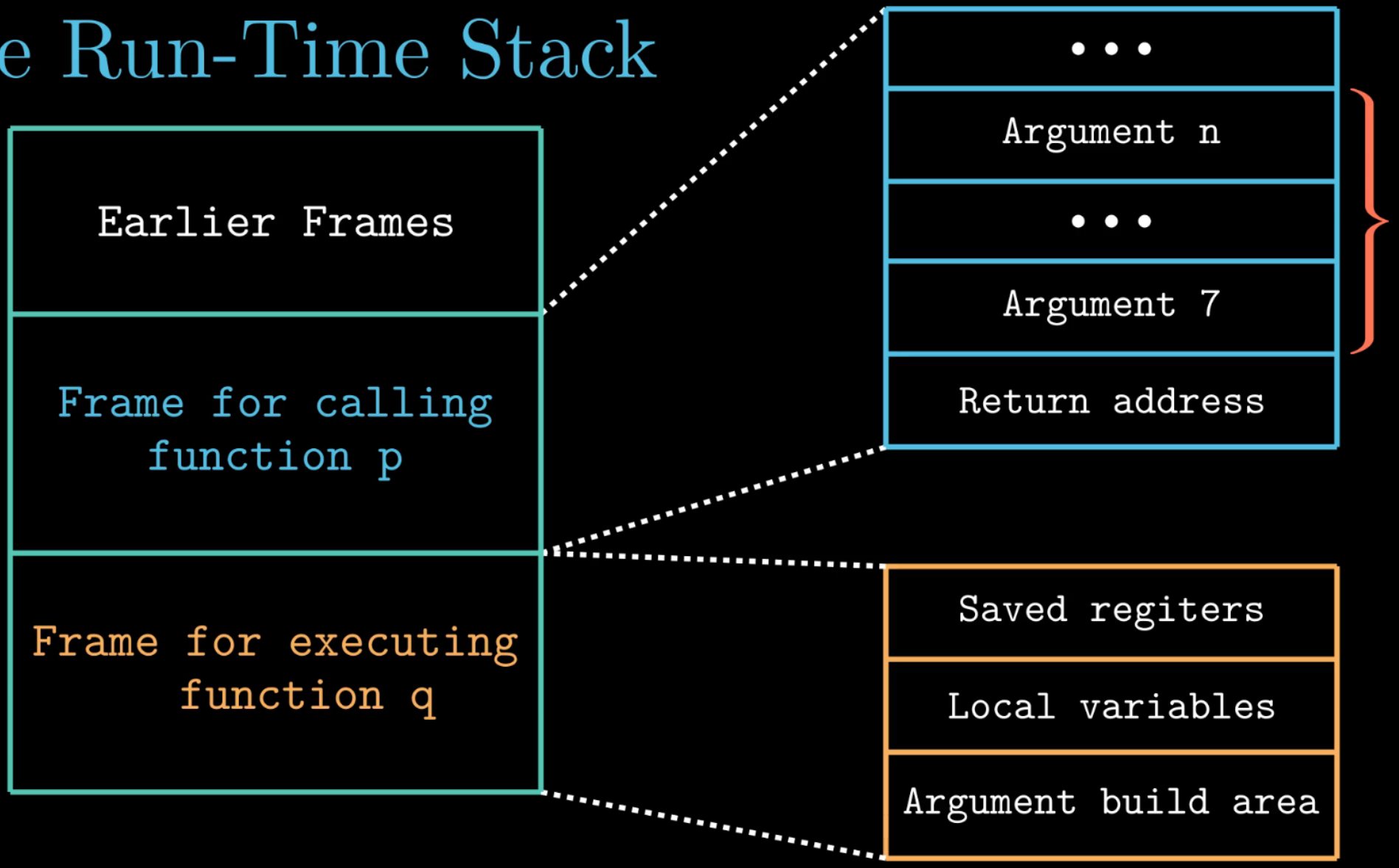


Code File

```
0000000000006da <main>:
  •••
  6fb: e8 41 00 00 00     callq  741 <multstore>
→ 700: 48 8b 14 24        mov  (%rsp), %rdx

0000000000000741 <multstore>:
  •••
  741: 53                 push  %rbx
  742: 48 89 d3           mov  %rdx, %rbx
```



The Run-Time Stack



Data Transfer

Operand size(bits)	Argument number					
	1	2	3	4	5	6
64	%rdi	%rsi	%rdx	%rcx	%r8	%r9
32	%edi	%esi	%edx	%ecx	%r8d	%r9d
16	%di	%si	%dx	%cx	%r8w	%r9w
8	%dil	%sil	%dl	%cl	%r8b	%r9b

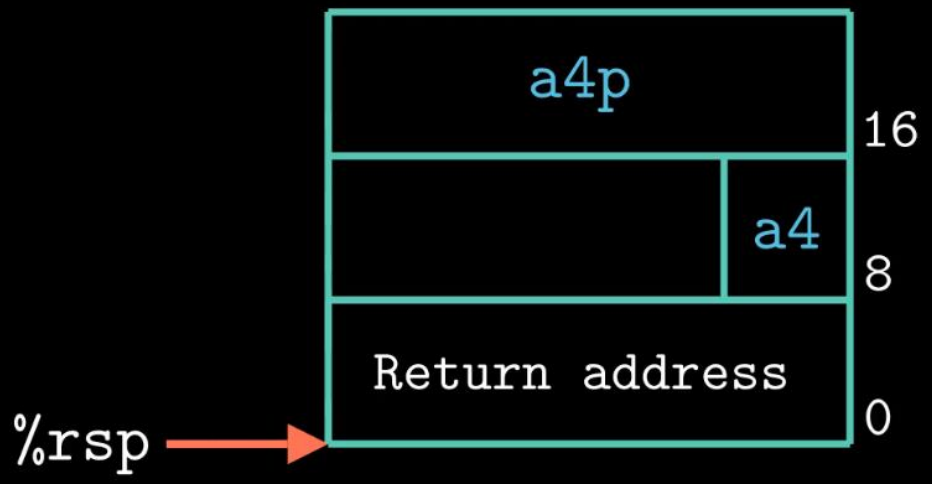
Data Transfer

```
void proc(long a1, long *a1p,
          int a2, long *a2p,
          short a3, long *a3p,
          char a4, long *a4p)
{
    a1p += a1;
    a2p += a2;
    a3p += a3;
    a4p += a4;
}
```

Data Transfer

```
void proc(long a1, long *a1p,
          int a2, long *a2p,
          short a3,long *a3p,
          char a4, long *a4p)
{
    a1p += a1;
    a2p += a2;
    a3p += a3;
    a4p += a4;
}
```

a1 → %rdi a1p → %rsi
a2 → %edx a2p → %rcx
a3 → %r8w a3p → %r9
a4 → %rsp + 8
a4p → %rsp + 16



Local Storage

```
long caller()
{
    long arg1 = 534;
    long arg2 = 1057;
    long sum = swap(&arg1, &arg2);
    long diff = arg1 - arg2;
    return sum * diff;
}
```

```
long swap(long *xp, long *yp)
{
    long x = *xp;
    long y = *yp;
    *xp = y
    *yp = x
    return x + y;
}
```

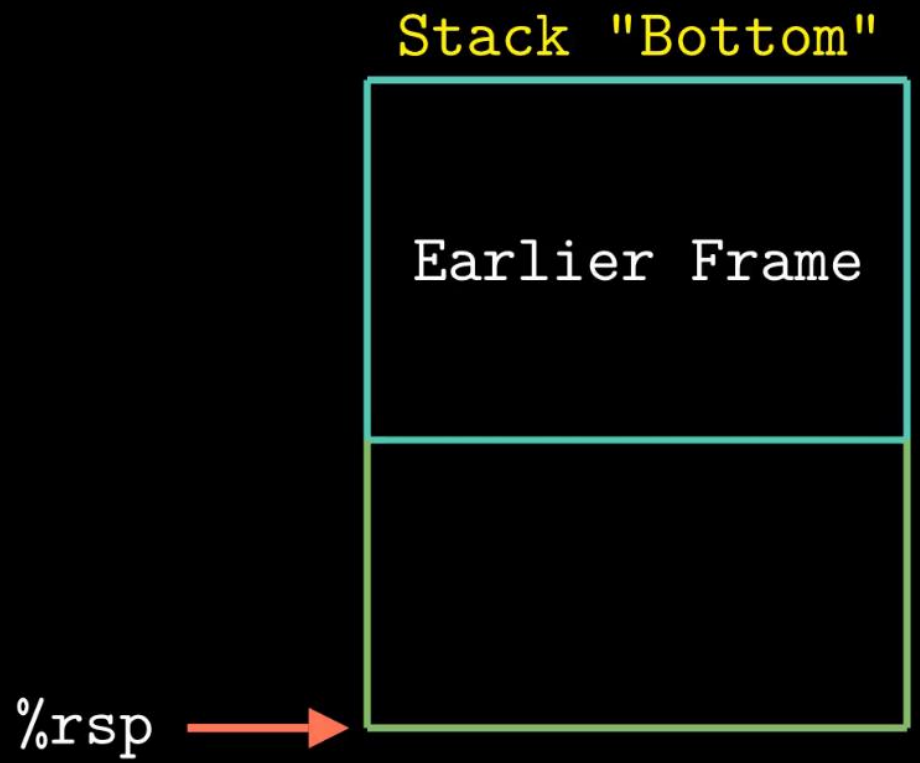
Local Storage

```
long caller()
{
    long arg1 = 534;
    long arg2 = 1057;
    long sum = swap(&arg1, &arg2);
    long diff = arg1 - arg2;
    return sum * diff;
}
```

```
caller:
    subq $16, %rsp
    movq $534, (%rsp)
    movq $1057, 8(%rsp)
    leaq 8(%rsp), %rsi
    movq %rsp, %rdi
    call swap
    movq (%rsp), %rdx
    subq 8(%rsp), %rdx
    imulq %rdx, %rax
    addq $16, %rsp
    ret
```

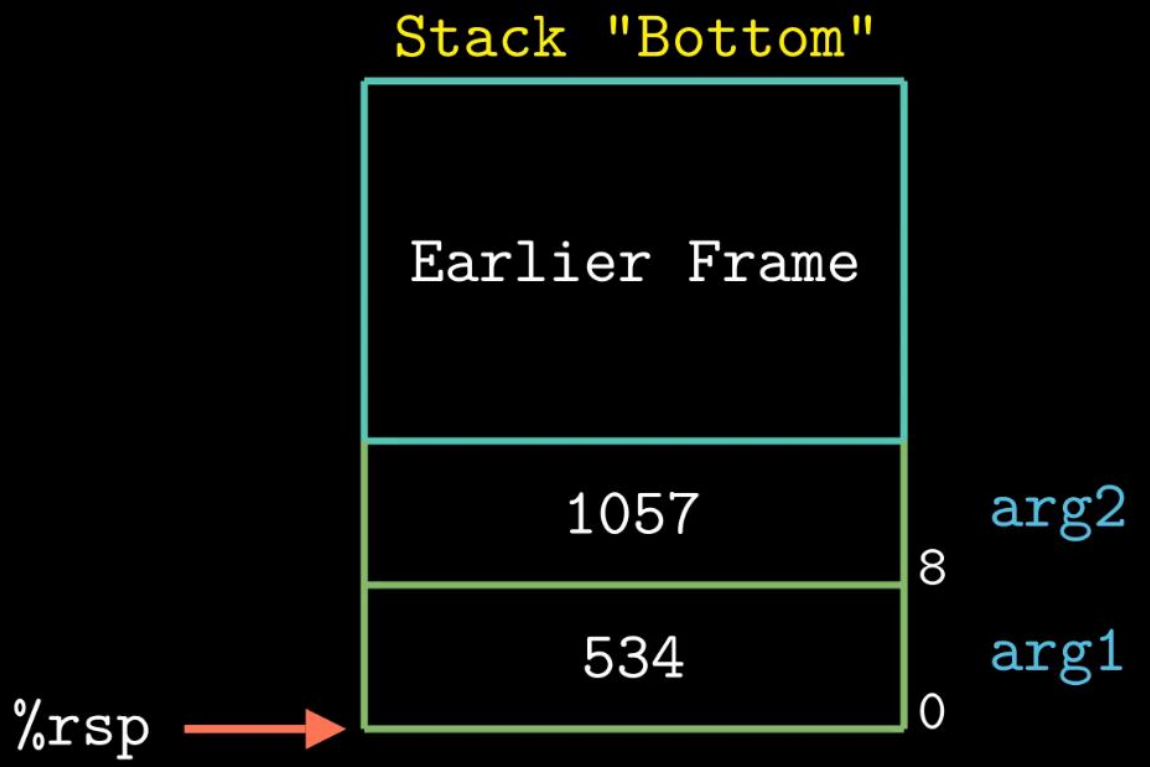
Local Storage

```
caller:
    subq $16, %rsp
    movq $534, (%rsp)
    movq $1057, 8(%rsp)
    leaq 8(%rsp), %rsi
    movq %rsp, %rdi
    call swap
    movq (%rsp), %rdx
    subq 8(%rsp), %rdx
    imulq %rdx, %rax
    addq $16, %rsp
    ret
```



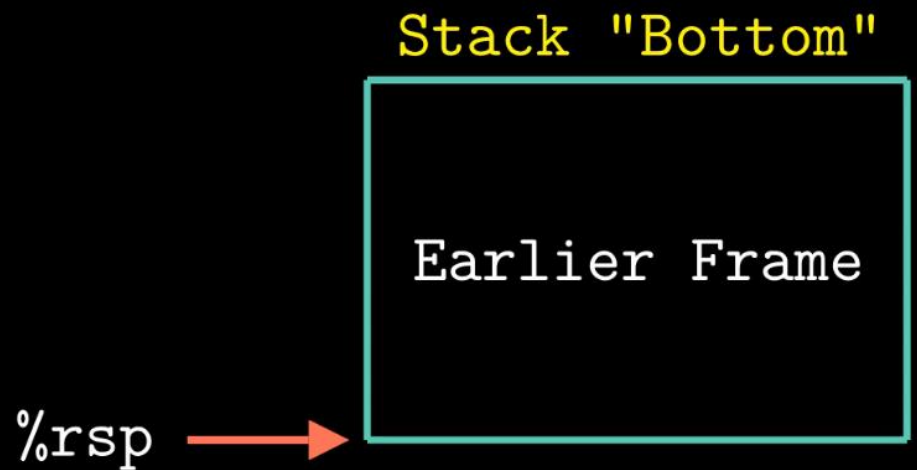
Local Storage

```
caller:
    subq $16, %rsp
    movq $534, (%rsp)
    movq $1057, 8(%rsp)
    leaq 8(%rsp), %rsi
    movq %rsp, %rdi
    call swap
    movq (%rsp), %rdx
    subq 8(%rsp), %rdx
    imulq %rdx, %rax
    addq $16, %rsp
    ret
```



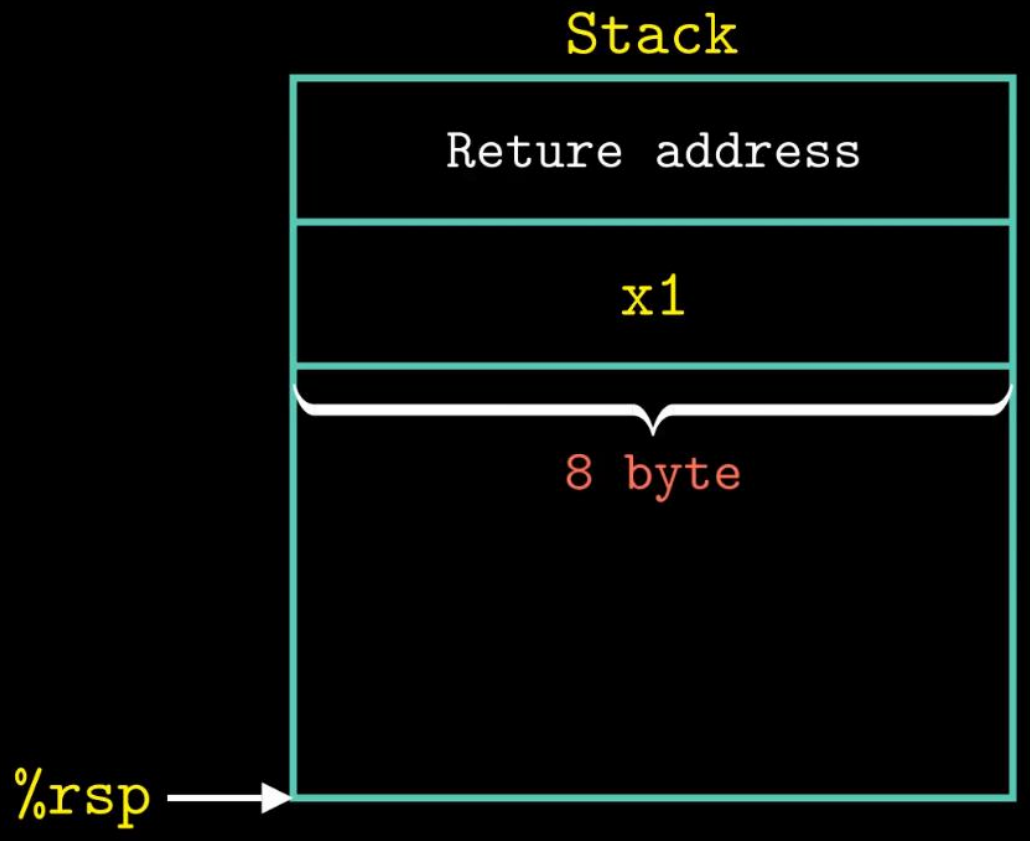
Local Storage

```
caller:
    subq $16, %rsp
    movq $534, (%rsp)
    movq $1057, 8(%rsp)
    leaq 8(%rsp), %rsi
    movq %rsp, %rdi
    call swap
    movq (%rsp), %rdx
    subq 8(%rsp), %rdx
    imulq %rdx, %rax
    addq $16, %rsp
    ret
```



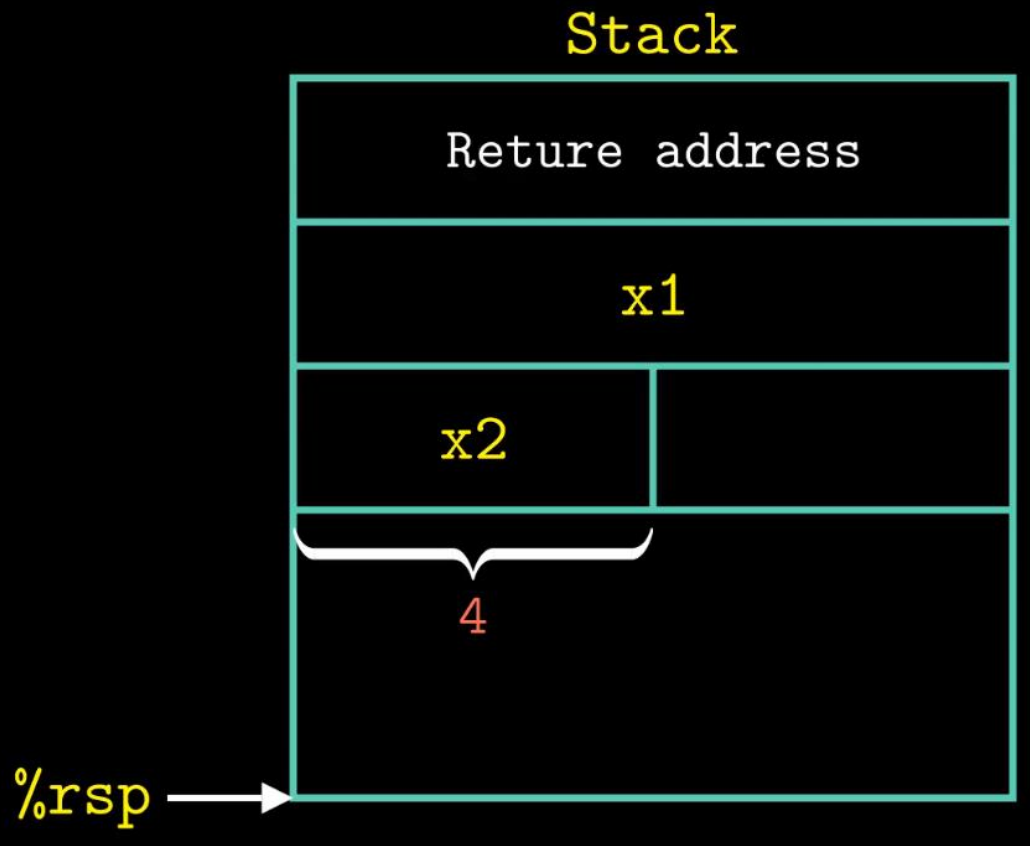
Local Storage

```
long call_proc()
{
    long  x1 = 1;
    int   x2 = 2;
    short x3 = 3;
    char  x4 = 4;
    proc(x1, &x1, x2, &x2,
         x3, &x3, x4, &x4);
    return (x1+x2)*(x3-x4);
}
```



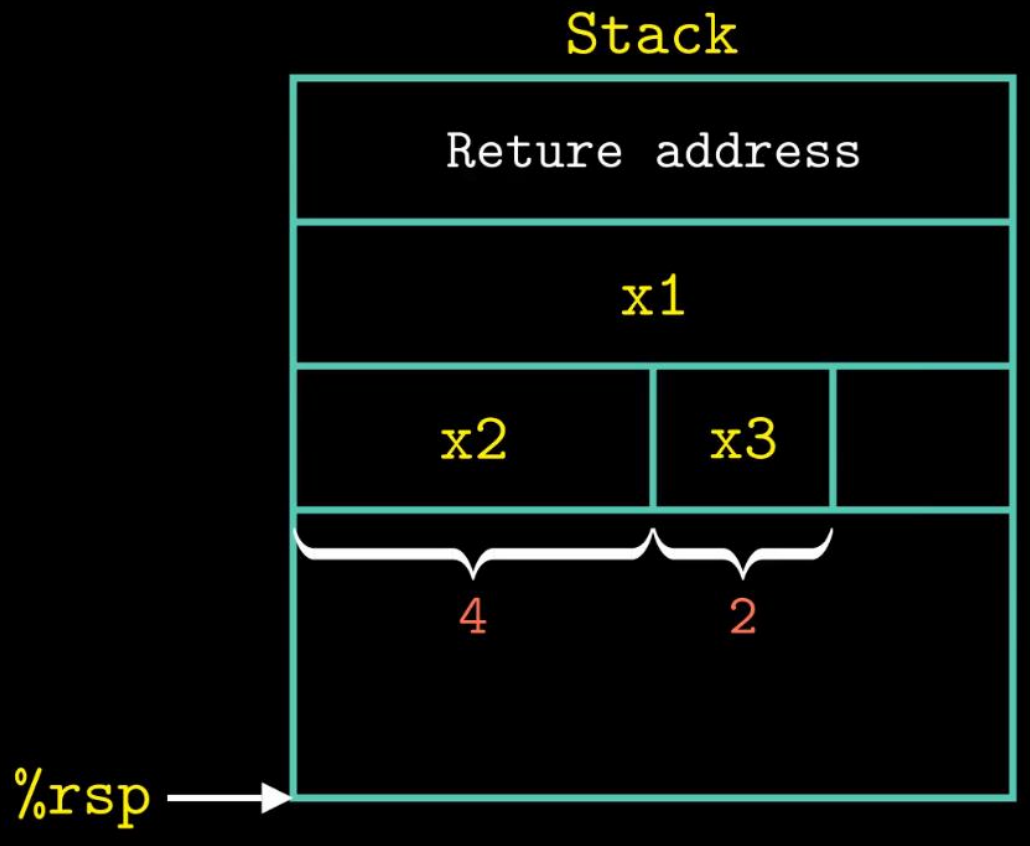
Local Storage

```
long call_proc()
{
    long  x1 = 1;
    int   x2 = 2;
    short x3 = 3;
    char  x4 = 4;
    proc(x1, &x1, x2, &x2,
         x3, &x3, x4, &x4);
    return (x1+x2)*(x3-x4);
}
```



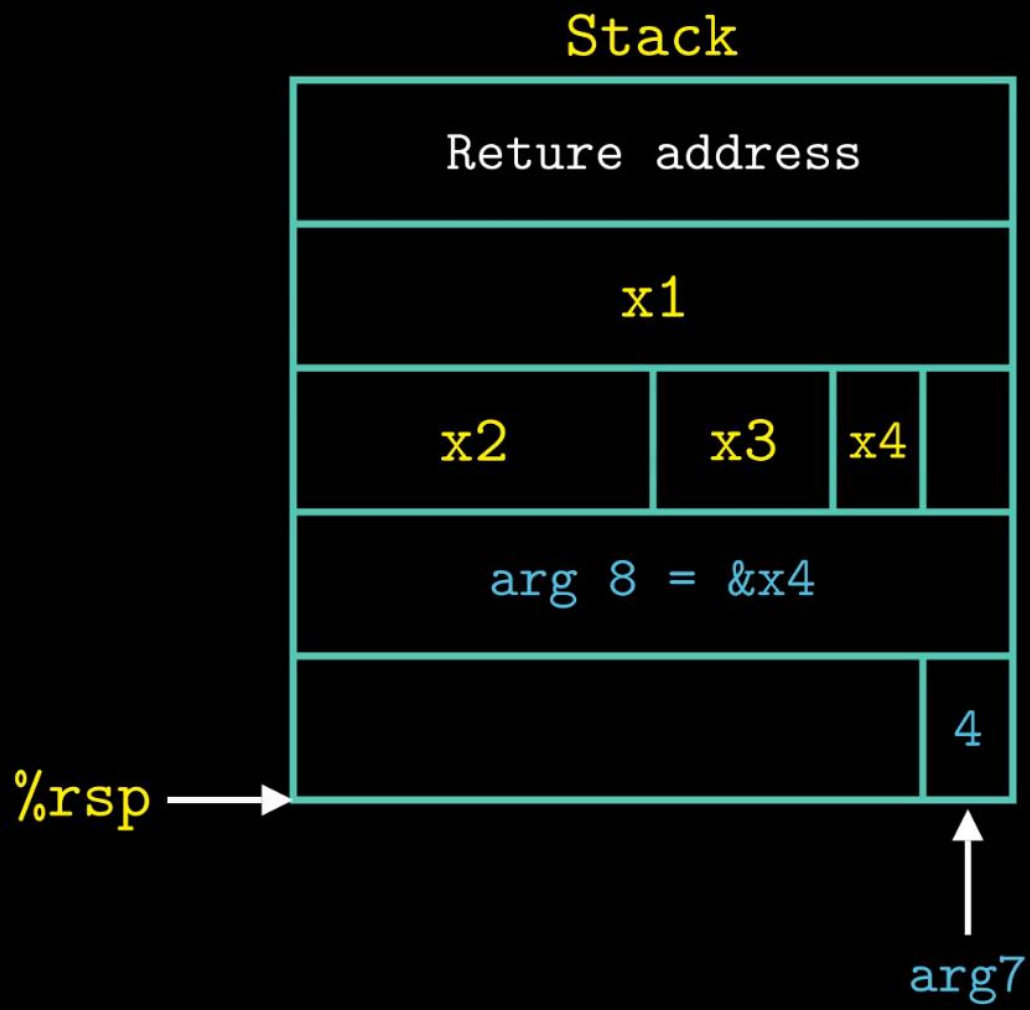
Local Storage

```
long call_proc()
{
    long  x1 = 1;
    int   x2 = 2;
    short x3 = 3;
    char  x4 = 4;
    proc(x1, &x1, x2, &x2,
         x3, &x3, x4, &x4);
    return (x1+x2)*(x3-x4);
}
```



Local Storage

```
long call_proc()
{
    long  x1 = 1;
    int   x2 = 2;
    short x3 = 3;
    char  x4 = 4;
    proc(x1, &x1, x2, &x2,
         x3, &x3, x4, &x4);
    return (x1+x2)*(x3-x4);
}
```



Local Storage in Register

```
void P(long x, long y)
{
      long u = Q (y);
      long v = Q (x);
      return u + v ;
}
```

```
P:
  x in %rdi, y in %rsi
   pushq %rbp
   pushq %rbx
   subq $8, %rsp
   movq %rdi, %rbp ⟶ Save x
   movq %rsi, %rdi
   call Q
   •••
   popq %rbx
   popq %rbp
   ret
```

Local Storage in Register

```
void P(long x, long y)
{
      long u = Q (y);
      long v = Q (x);
      return u + v ;
}
```

```
P:
  x in %rdi, y in %rsi
   pushq %rbp
   pushq %rbx
   subq $8, %rsp
   movq %rdi, %rbp ⟶ Save x
   movq %rsi, %rdi
   call Q
   •••
   popq %rbx
   popq %rbp
   ret
```

Recursive Procedures

```
long rfact(long n)
{
    long result;
    if (n <= 1) {
        result = 1;
    }
    else {
        result = n * rfact(n - 1)
    }
    return result;
}
```

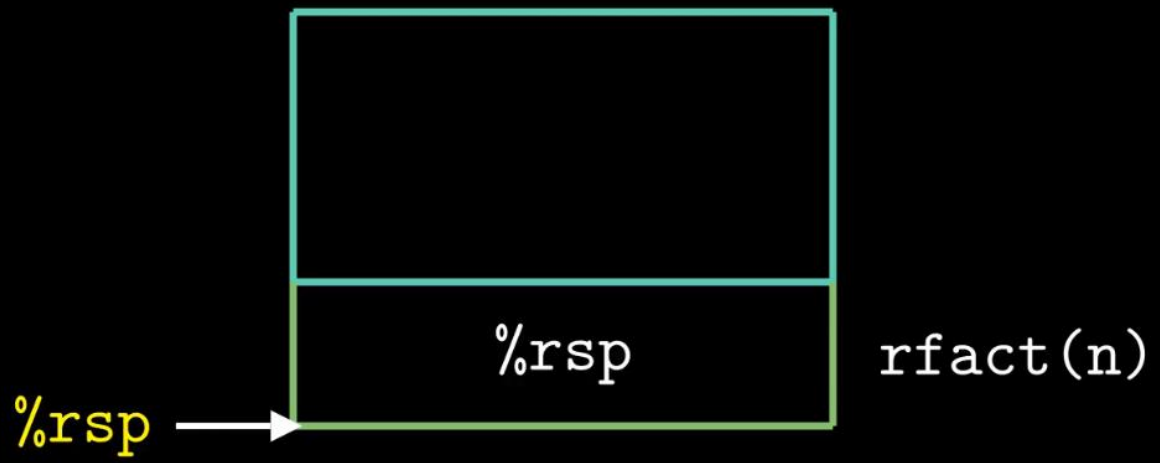
Recursive Procedures

```
long rfact(long n)
{
    long result;
    if (n <= 1) {
        result = 1;
    }
    else {
        result = n * rfact(n - 1)
    }
    return result;
}
```

```
rfact:
    pushq %rbx
    movq %rdi, %rbx
    movl $1, %eax
    cmpq $1, %rdi
    jle .L35
    leaq -1(%rdi), %rdi
    call rfact
    imulq %rbx, %rax
.L35:
    popq %rbx
    ret
```

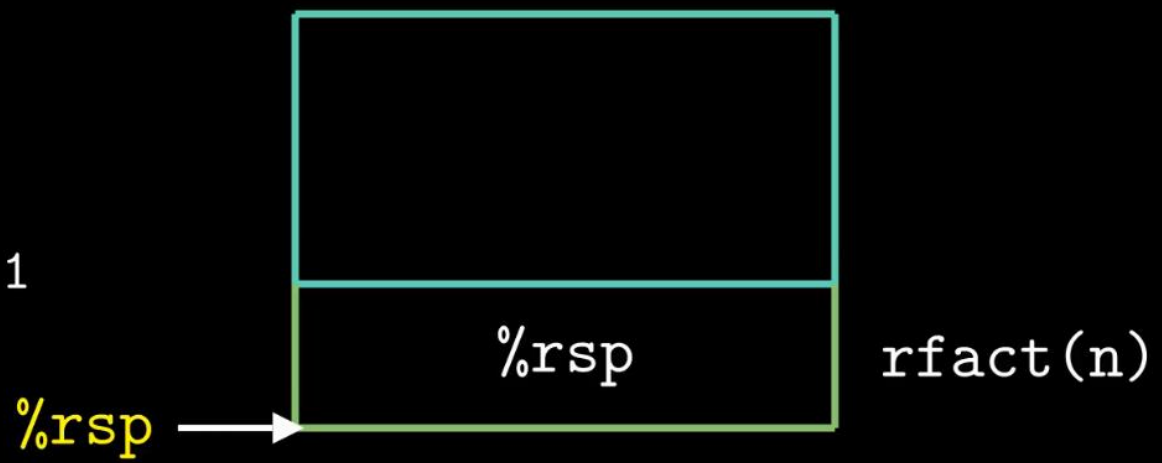
Recursive Procedures

```
rfact:
    pushq %rbx      → Save %rbx
    movq %rdi, %rbx
    movl $1, %eax
    cmpq $1, %rdi
    jle .L35
    leaq -1(%rdi), %rdi
    call rfact
    imulq %rbx, %rax
.L35:
    popq %rbx
    ret
```



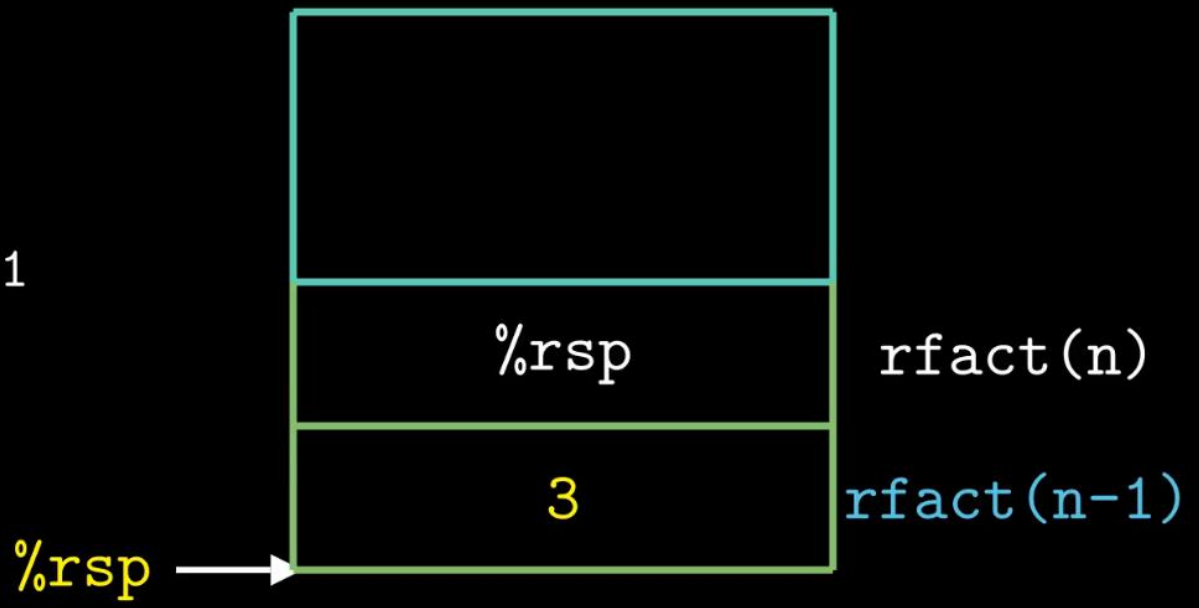
Recursive Procedures

```
rfact:
    pushq %rbx      → Save %rbx
    movq %rdi, %rbx
    movl $1, %eax
→   cmpq $1, %rdi  → compare n:1
    jle .L35
    leaq -1(%rdi), %rdi
    call rfact
    imulq %rbx, %rax
.L35:
    popq %rbx
    ret
```



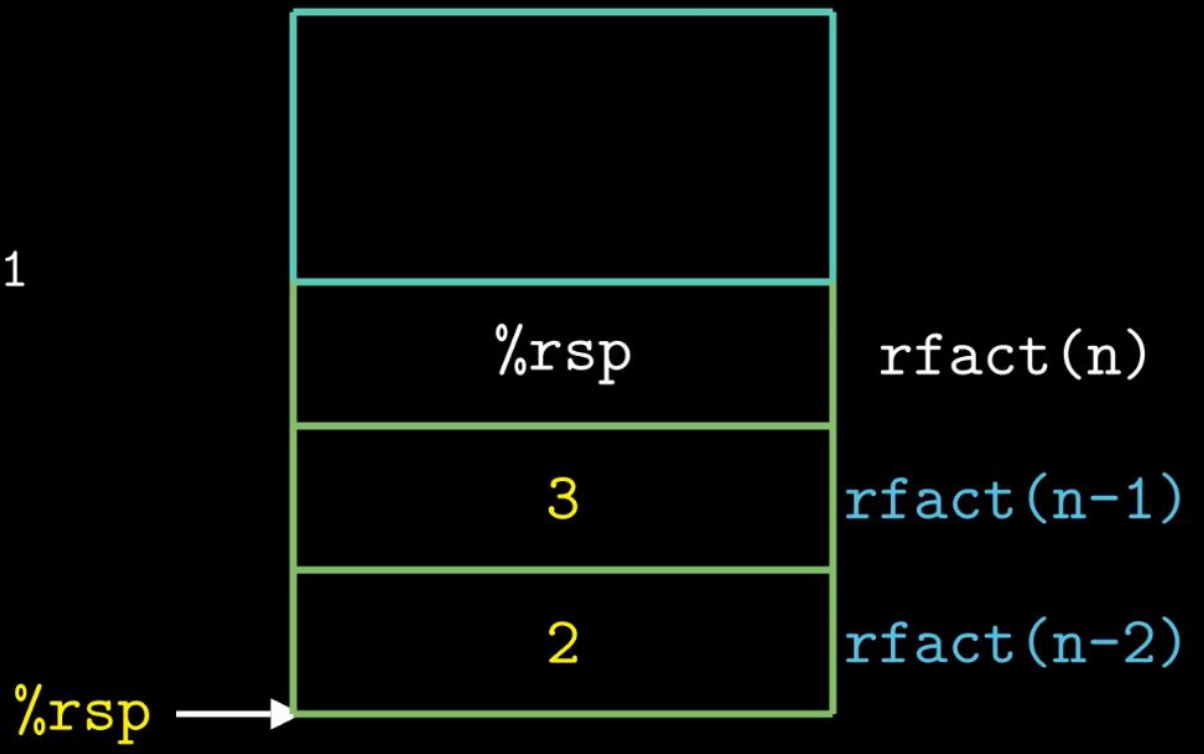
Recursive Procedures

```
rfact:
    pushq %rbx      → %rbx = 3
    movq %rdi, %rbx
    movl $1, %eax
    cmpq $1, %rdi   → compare n:1
    jle .L35
    leaq -1(%rdi), %rdi
    call rfact
    imulq %rbx, %rax
.L35:
    popq %rbx
    ret
```



Recursive Procedures

```
rfact:
    pushq %rbx  ⟶  %rbx = 3
    movq %rdi, %rbx
    movl $1, %eax
    cmpq $1, %rdi  ⟶  compare n:1
    jle .L35
    leaq -1(%rdi), %rdi
    call rfact
    imulq %rbx, %rax
.L35:
⟶  popq %rbx
    ret
```



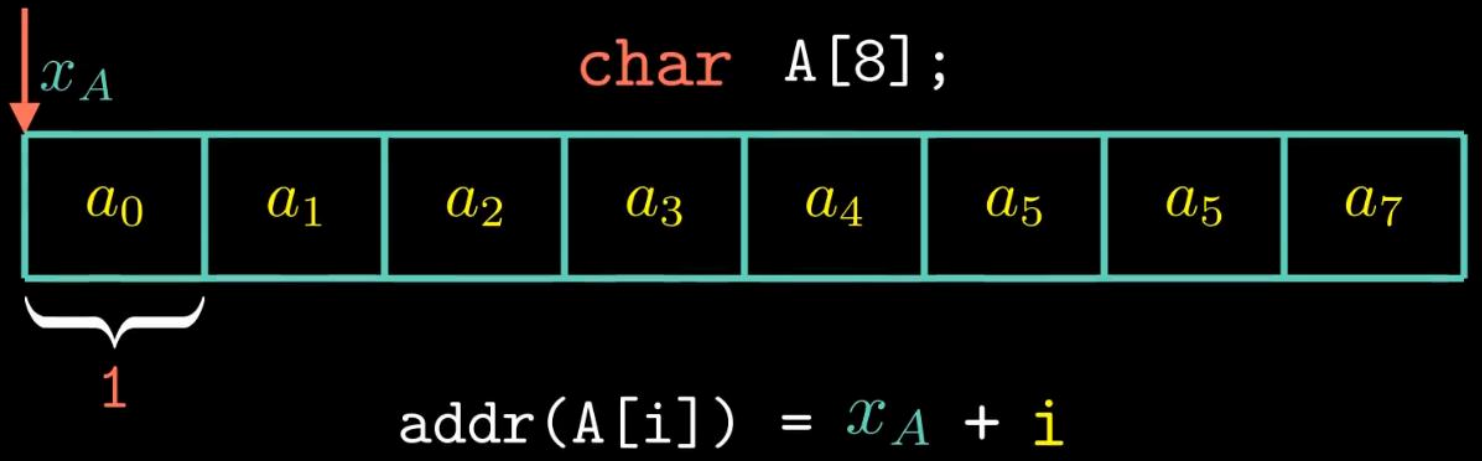
Computer Systems

A Programmer's Perspective

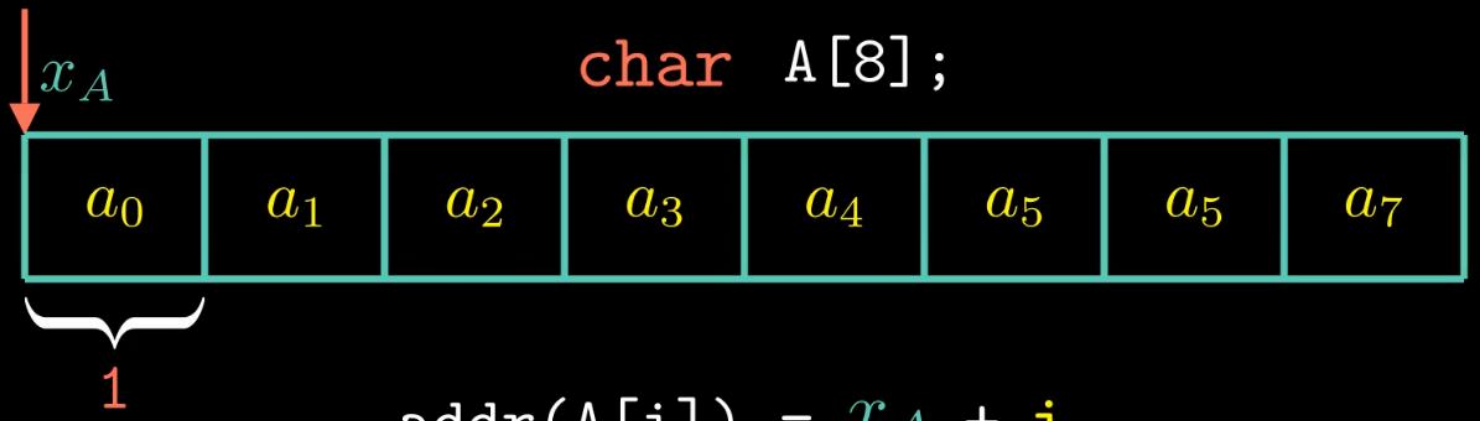
Chapter 3: Machine-Level Representation of Programs

(程序的机器级表示)

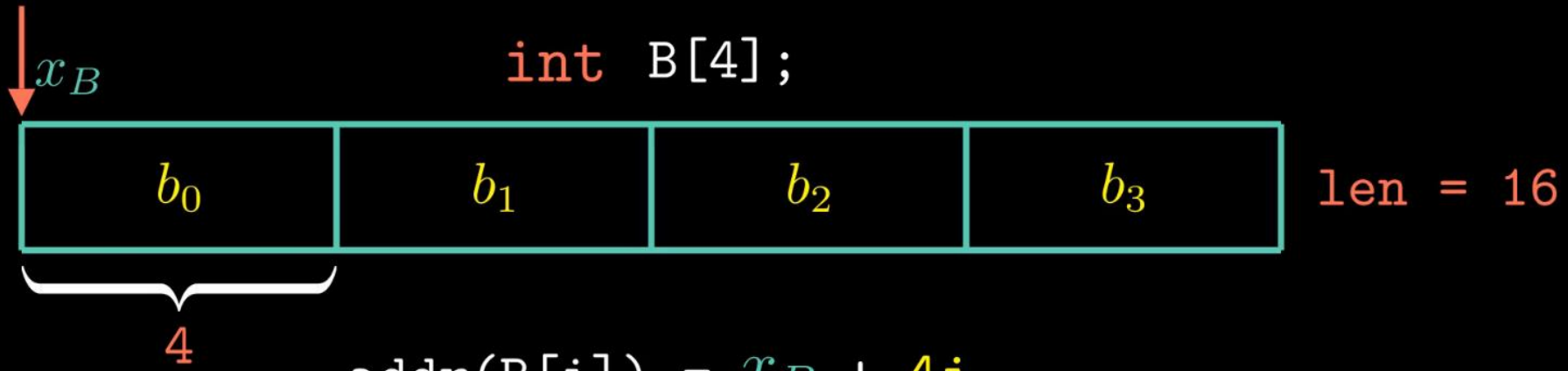
Basic Principles



Basic Principles



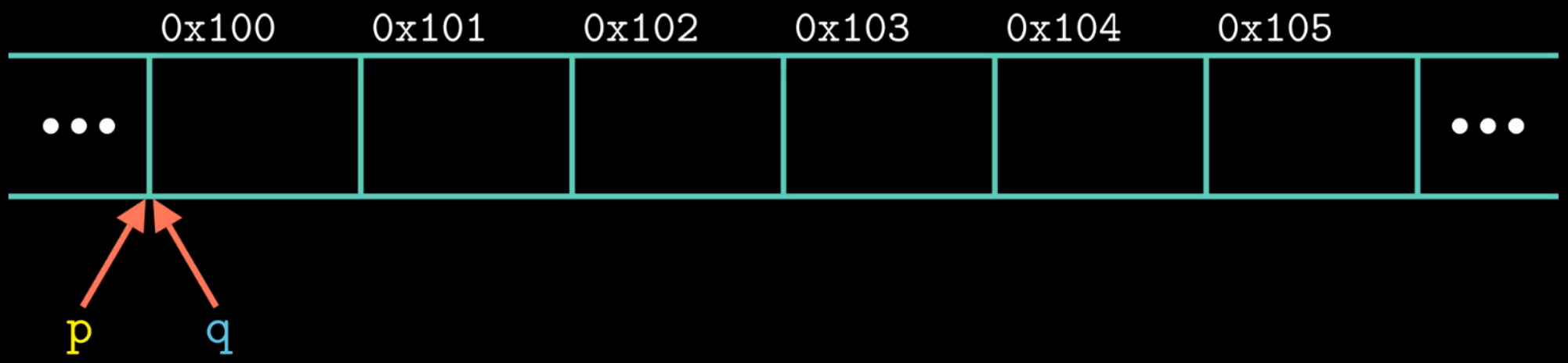
addr(A[i]) = x_A + i



addr(B[i]) = x_B + 4i

Pointer Arithmetic

```
char *p;
int *q;
```



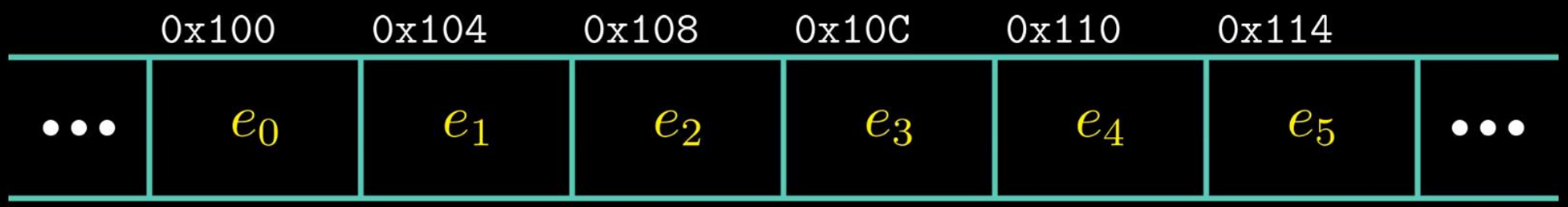
Pointer Arithmetic

```
char *p;
int *q;
```

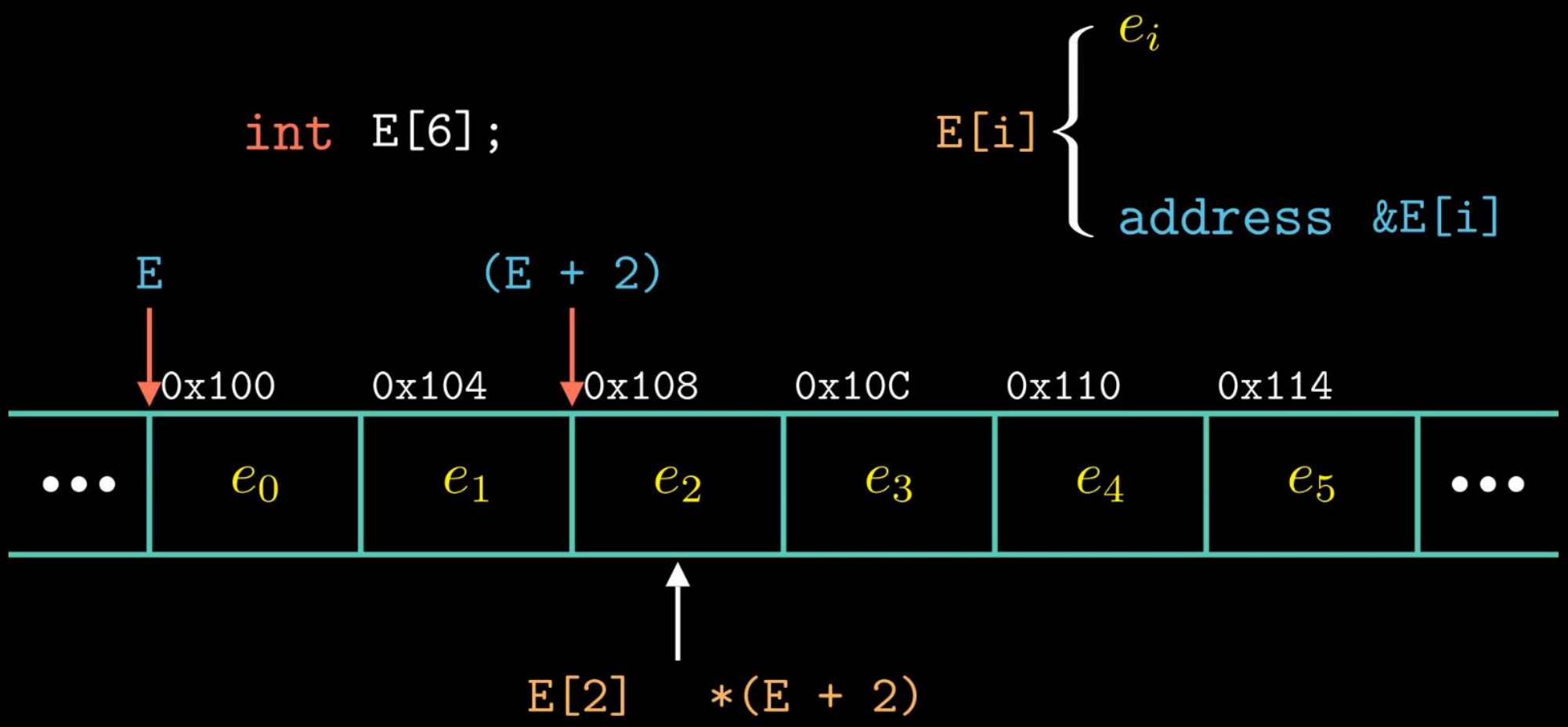
Pointer Arithmetic

```
int E[6];
```

E[i] { e_i ; address &E[i] }



Pointer Arithmetic



Nested Arrays

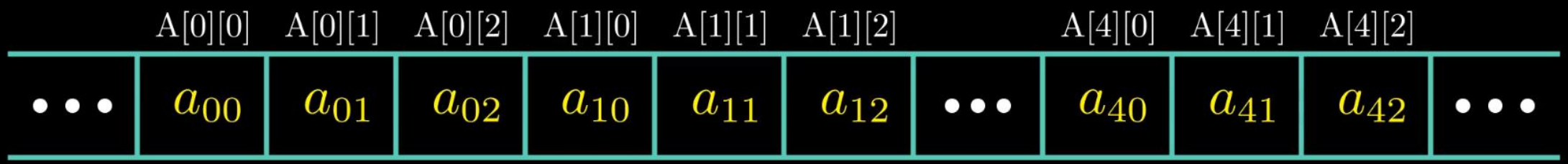
```
int A[5][3];
```

$$\mathrm{A} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \\ a_{30} & a_{31} & a_{32} \\ a_{40} & a_{41} & a_{42} \end{bmatrix}$$

Nested Arrays

```
int A[5][3];
```

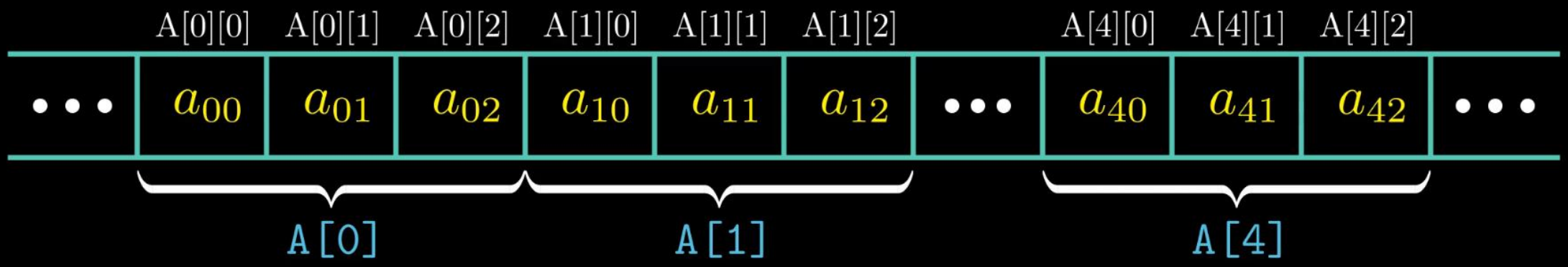
$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \\ a_{30} & a_{31} & a_{32} \\ a_{40} & a_{41} & a_{42} \end{bmatrix}$$



Nested Arrays

```
int A[5][3];
```

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \\ a_{30} & a_{31} & a_{32} \\ a_{40} & a_{41} & a_{42} \end{bmatrix}$$



Nested Arrays

```
T D[R][C];
```

$$\&D[i][j] = x_D + L(C \cdot i + j)$$

```
int A[5][3];
```

$$\&A[i][j] = x_A + 4(3 \cdot i + j)$$

Nested Arrays

T D[R][C];

&D[i][j] = x_D + L($C \cdot i$ + j)

int A[5][3];

&A[i][j] = x_A + 4($3 \cdot i$ + j)

x_A in %rdi, i in %rsi, j in %rdx

```
leaq (%rsi, %rsi, 2), %rax   compute 3*i
leaq (%rdi, %rax, 4), %rax   compute x_A +12*i
movl (%rax, %rdx, 4), %eax   Read from M[x_A +12i+4j]
```

Fixed-Size Arrays

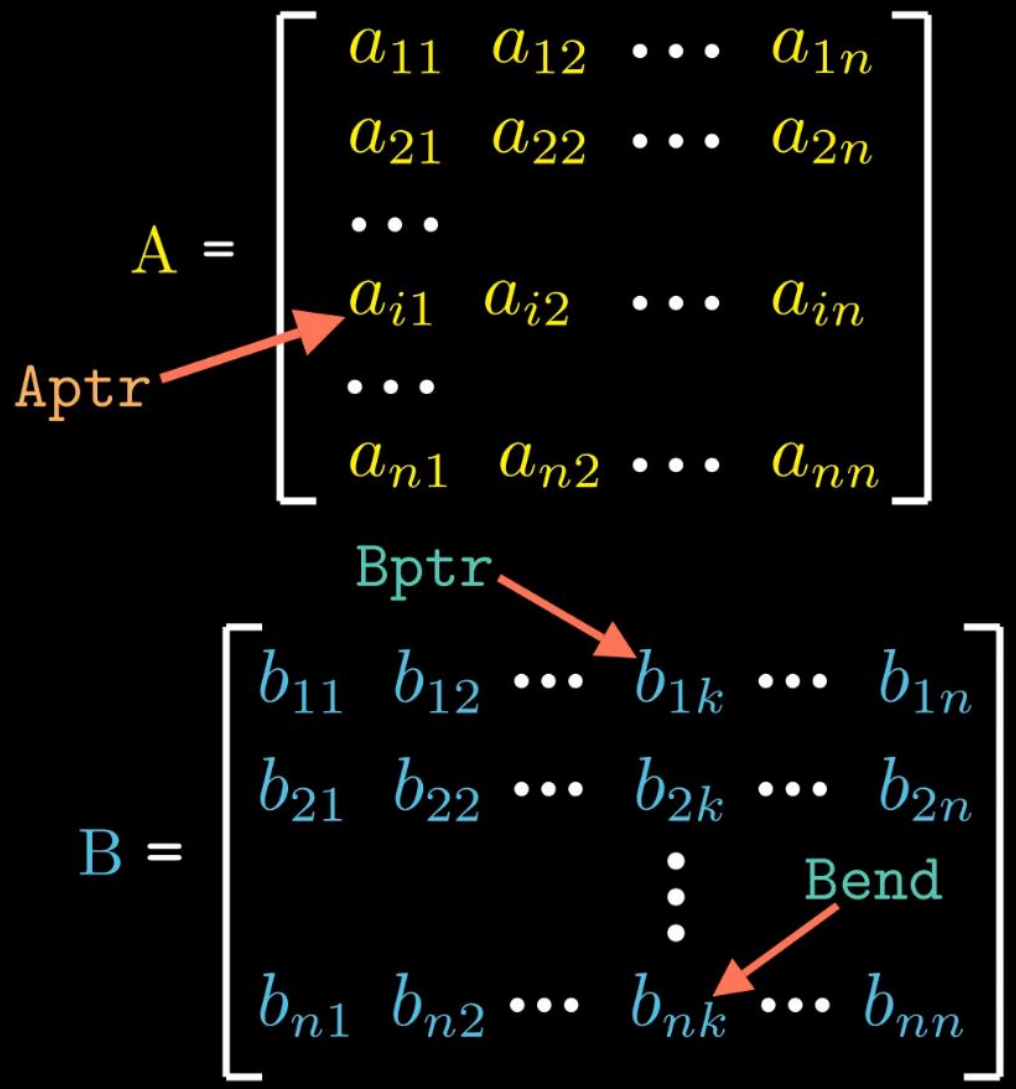
```
#define N 16
typedef int fix_matrix[N][N];

int matrix(fix_matrix A, fix_matrix B,
           long i, long k)
{
    long j;
    int result = 0;

    for (j = 0; j < N; j++)
        result += A[i][j] * B[j][k];
    return result;
}
```

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & & & \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \cdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2k} & \cdots & b_{2n} \\ & & & \vdots & & \\ b_{n1} & b_{n2} & \cdots & b_{nk} & \cdots & b_{nn} \end{bmatrix}$$

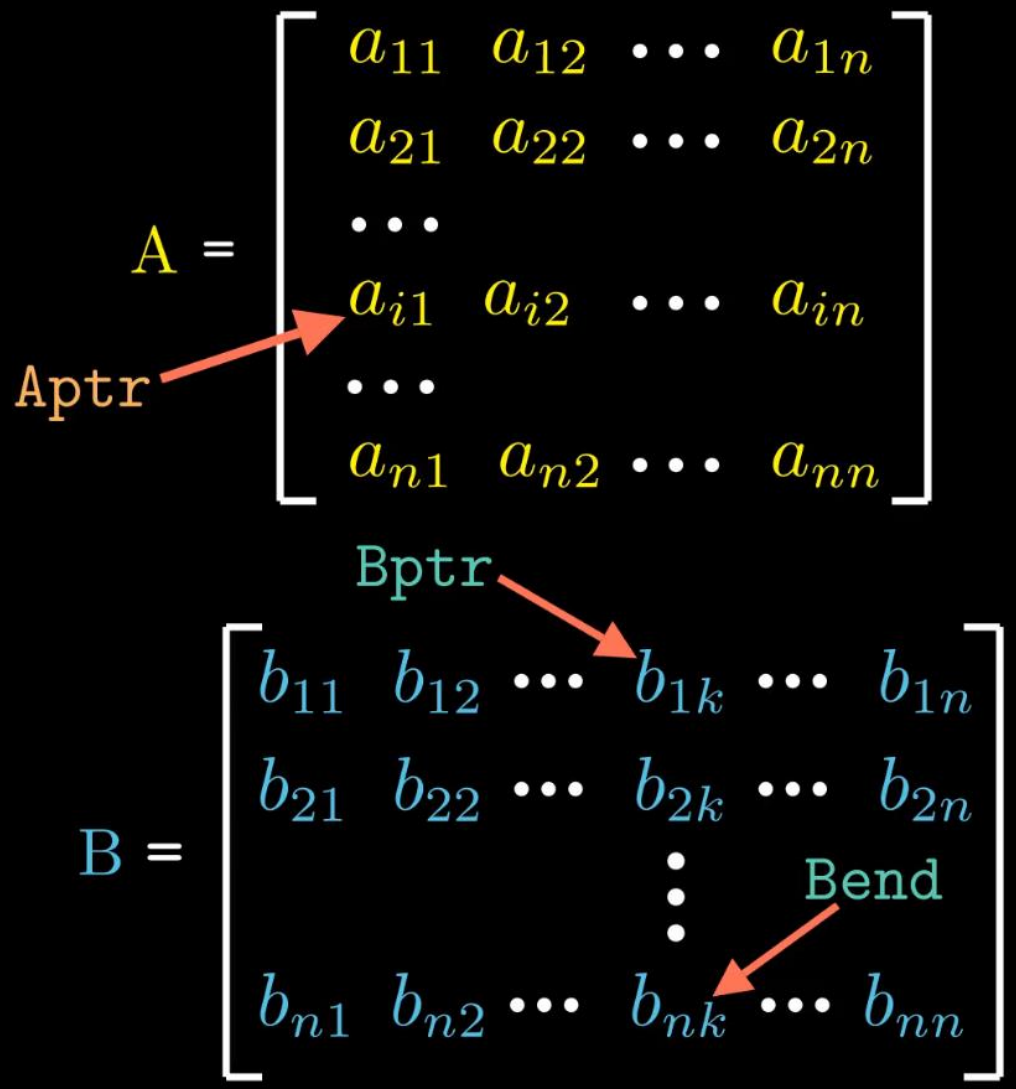
Fixed-Size Arrays

```
matrix:
    salq $6, %rdx
    addq %rdx, %rdi
    leaq (%rsi, %rcx, 4), %rcx
    leaq 1024(%rcx), %rsi
    movl $0, %eax
.L7:
    ...
```



Fixed-Size Arrays

```
.L7:
    movl (%rdi), %edx
    imull (%rcx), %edx
    addl %edx, %eax
    addq $4, %rdi
    addq $64, %rcx
    cmpq (%rsi), %rcx
    jne .L7
    rep; ret
```



Fixed-Size Arrays

```
.L7:
    movl (%rdi), %edx    read *Aptr
    imull (%rcx), %edx   multiply by *Bptr
    addl %edx, %eax      add to result
    addq $4, %rdi
    addq $64, %rcx
    cmpq (%rsi), %rcx
    jne .L7
    rep; ret
```

$$A = \overbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & & & \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \cdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}}^{4}$$

Aptr

$$B = \underbrace{\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2k} & \cdots & b_{2n} \\ & & & \vdots & & \\ b_{n1} & b_{n2} & \cdots & b_{nk} & \cdots & b_{nn} \end{bmatrix}}_{64}$$

Bptr

Bend

Fixed-Size Arrays

```
.L7:
    movl (%rdi), %edx    read *Aptr
    imull (%rcx), %edx   multiply by *Bptr
    addl %edx, %eax      add to result
    addq $4, %rdi
    addq $64, %rcx
    cmpq (%rsi), %rcx
    jne .L7
    rep; ret
```

$$A = \overbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & & & \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \cdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}}^{4}$$

Aptr

$$B = \underbrace{\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2k} & \cdots & b_{2n} \\ & \text{Bptr} & & \vdots & & \text{Bend} \\ b_{n1} & b_{n2} & \cdots & b_{nk} & \cdots & b_{nn} \end{bmatrix}}_{64}$$

Variable-Size Arrays

```
int A[expr1][expr2];

long var_ele(long n, int A[n][n], long i, long k)
{
    return A[i][j];
}
```

Variable-Size Arrays

```
int A[expr1][expr2];
```

```
long var_ele( long n, int A[n][n], long i, long k)
{
    return A[i][j];
}
```

n in %rdi, A in %rsi, i in %rdx, j in %rcx

```
var_ele:
  imulq %rdx, %rdi              compute n*i
  leaq (%rsi, %rdi, 4), %rax    compute x_A +4(n*i)
  movl (%rax, %rcx, 4), %eax    Read from M[x_A +4(n*i)+4j]
  ret
```

Variable-Size Arrays

```
int var_mat( long n, int A[n][n], int B[n][n],
             long i,long k)
{
    long j;
    int result = 0;

    for (j = 0; j < N; j++)
        result += A[i][j] * B[j][k];
    return result;
}
```

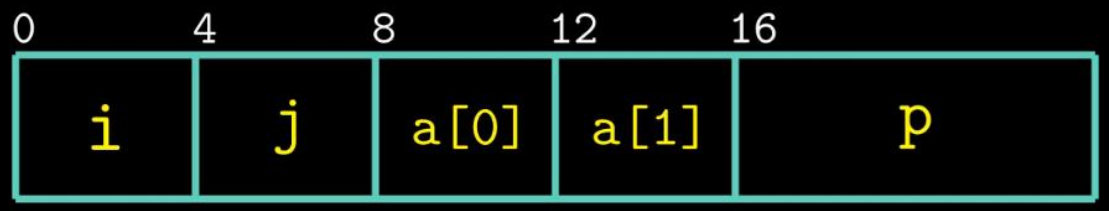
Variable-Size Arrays

```
.L7:(fixed)
    movl (%rdi), %edx
    imull (%rcx), %edx
    addl %edx, %eax
    addq $4, %rdi
    addq $64, %rcx
    cmpq (%rsi), %rcx
    jne .L7
    rep; ret
```

```
.L24:(variable)
    movl (%rsi, %rdx, 4), %r8d
    imull (%rcx), %r8d
    addl %r8d, %eax
    addq $1, %rdx
    addq %r9, %rcx
    cmpq %rdi, %rdx
    jne .L24
```

Structures

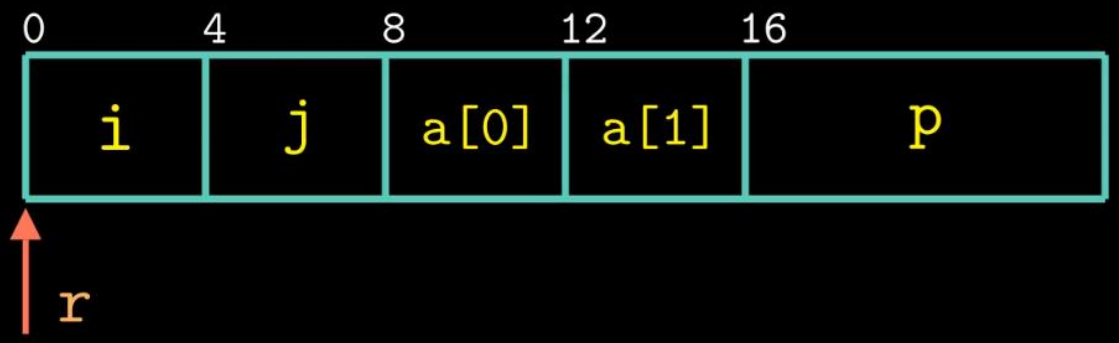
```
struct rec {
    int     i;
    int     j;
    int     a[2];
    int     *p;
};
```



Structures

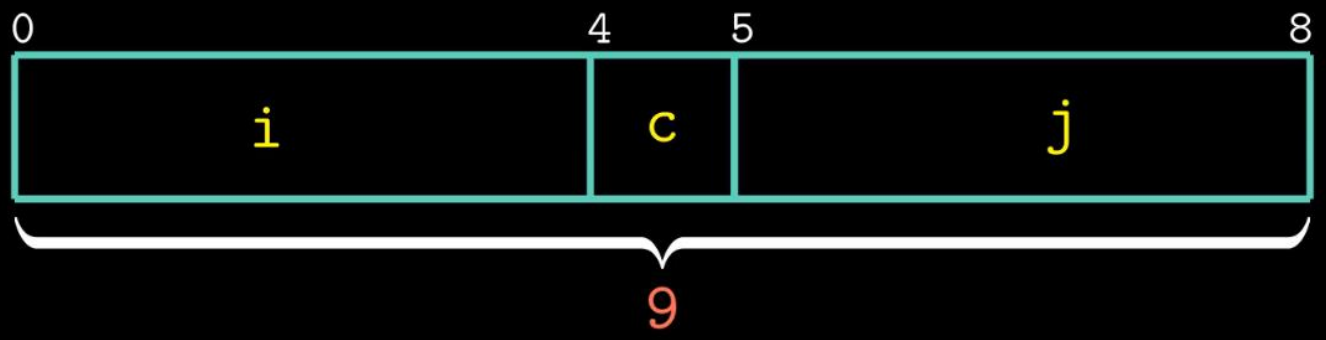
```
struct rec {
    int      i;
    int      j;
    int      a[2];
    int      *p;
};
```

```
r in %rdi
  movl  (%rdi), %eax    Get r->i
  movl  %eax, 4(%rdi)   Store in r->j
r in %rdi, i in %rsi
  leaq  8(%rdi,%rsi,4), %rax
```



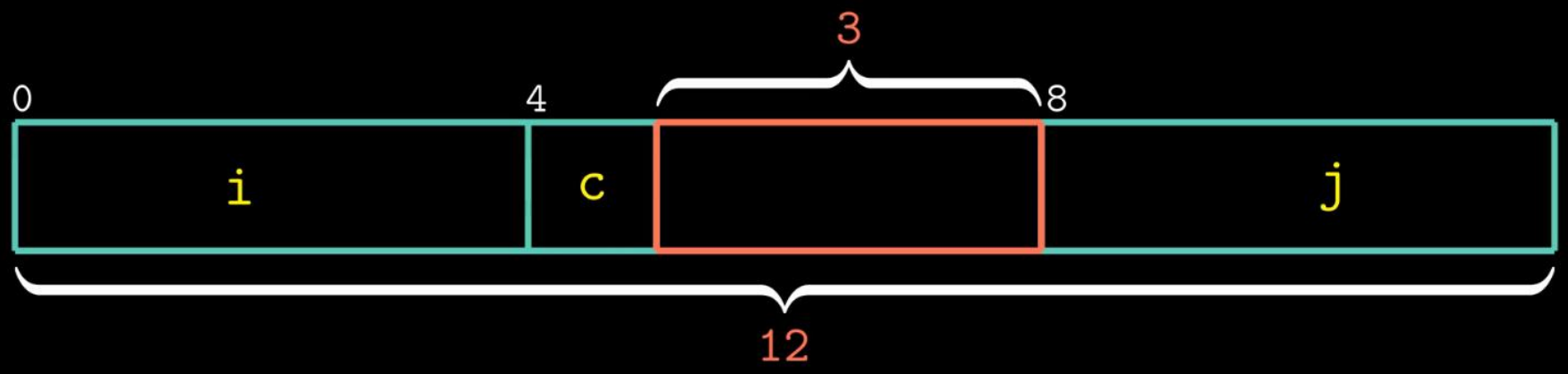
Data Alignment

```
struct S1 {
    int     i;
    char    c;
    int     j;
};
```



Data Alignment

```
struct S1 {
    int     i;
    char    c;
    int     j;
};
```



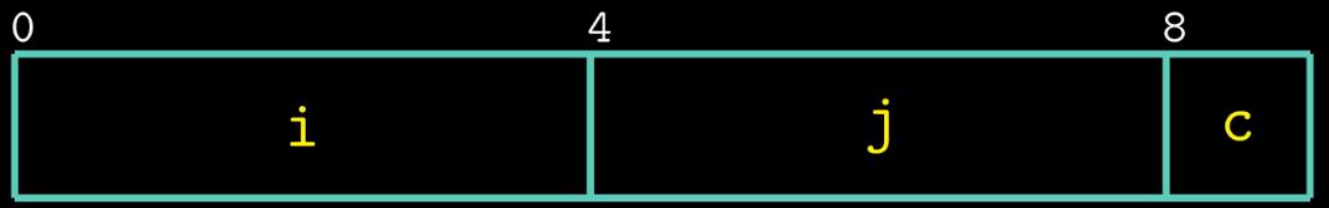
Data Alignment

K	Types
1	char
2	short
4	int, float
8	long, double, char*

Data Alignment

```
struct S1 {
    int     i;
    char    c;
    int     j;
};
```

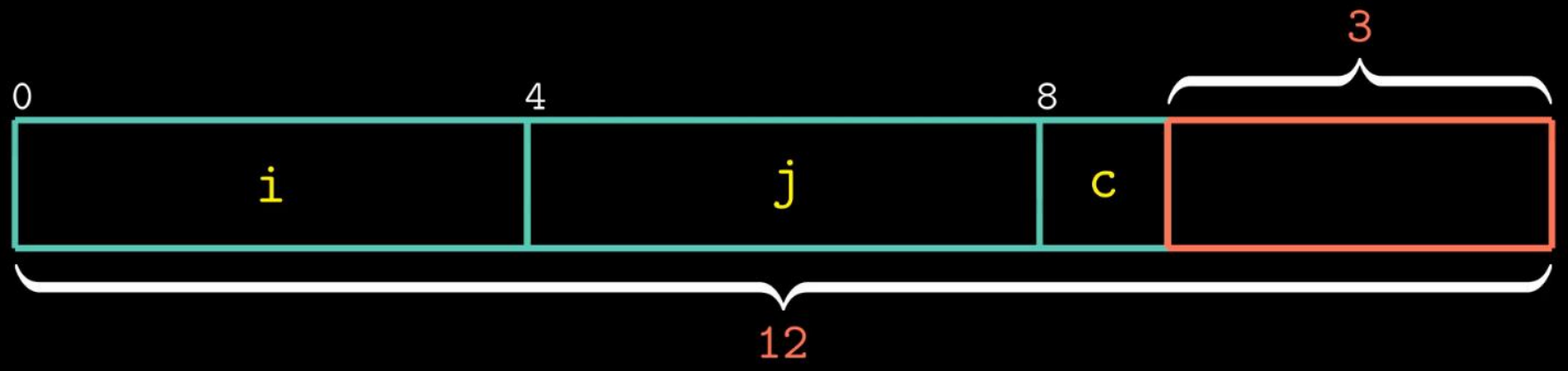
```
struct S2 {
    int     i;
    int     j;
    char    c;
};
```



Data Alignment

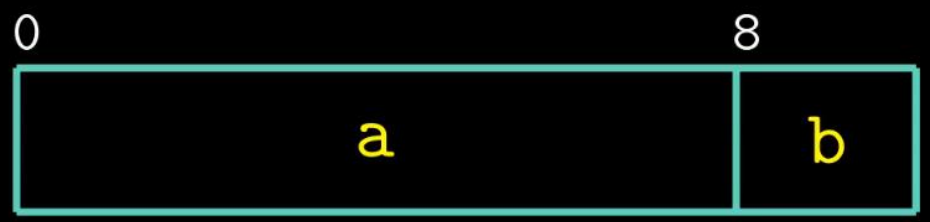
```
struct S1 {
    int     i;
    char    c;
    int     j;
};
```

```
struct S2 {
    int     i;
    int     j;
    char    c;
};
```



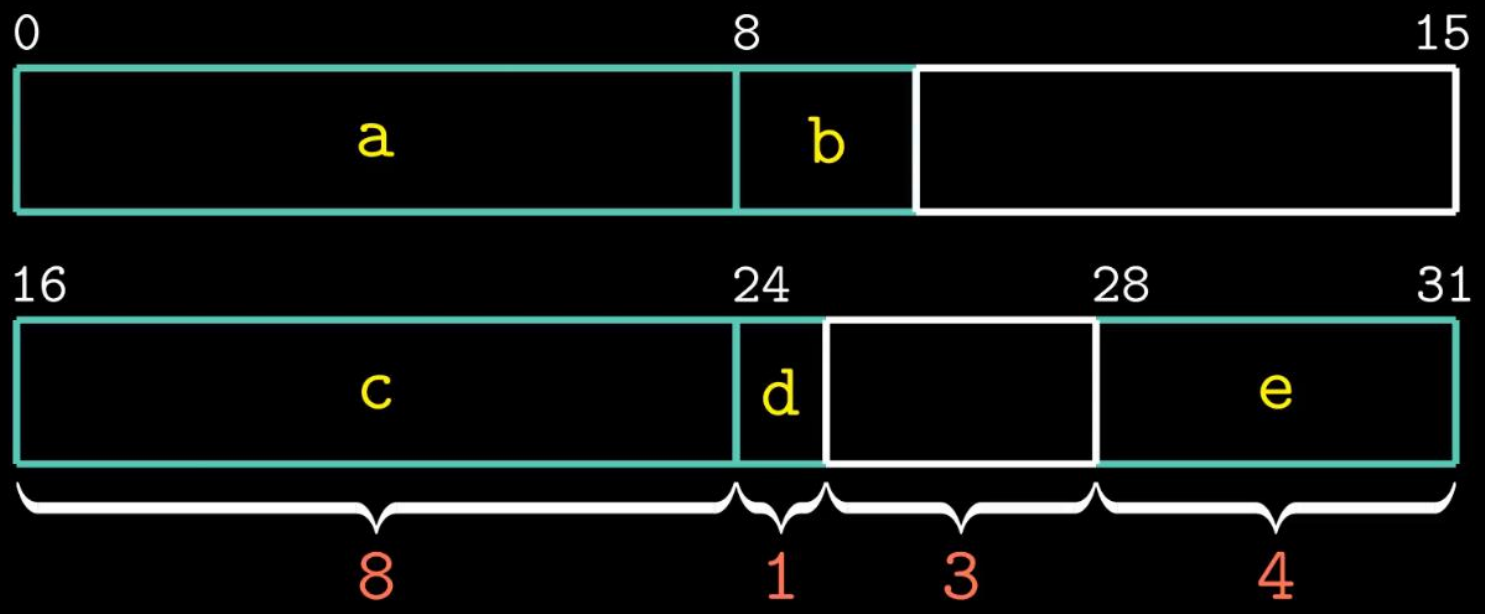
Data Alignment

```
struct {
    char      *a;
    short     b;
    double    c;
    char      d;
    float     e;
    char      f;
    long      g;
    int       h;
} rec;
```



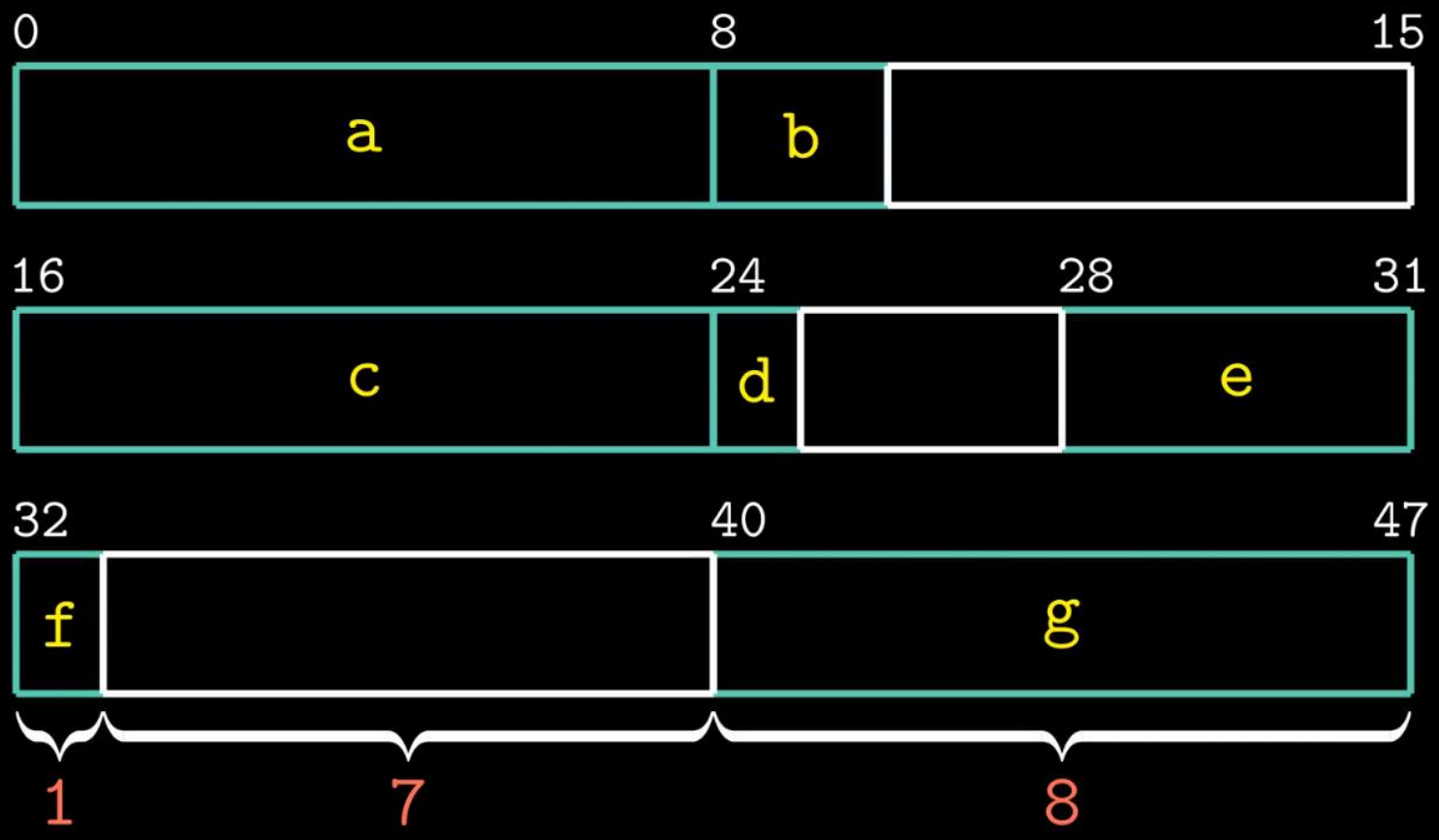
Data Alignment

```
struct {
    char      *a;
    short     b;
    double    c;
    char      d;
    float     e;
    char      f;
    long      g;
    int       h;
} rec;
```



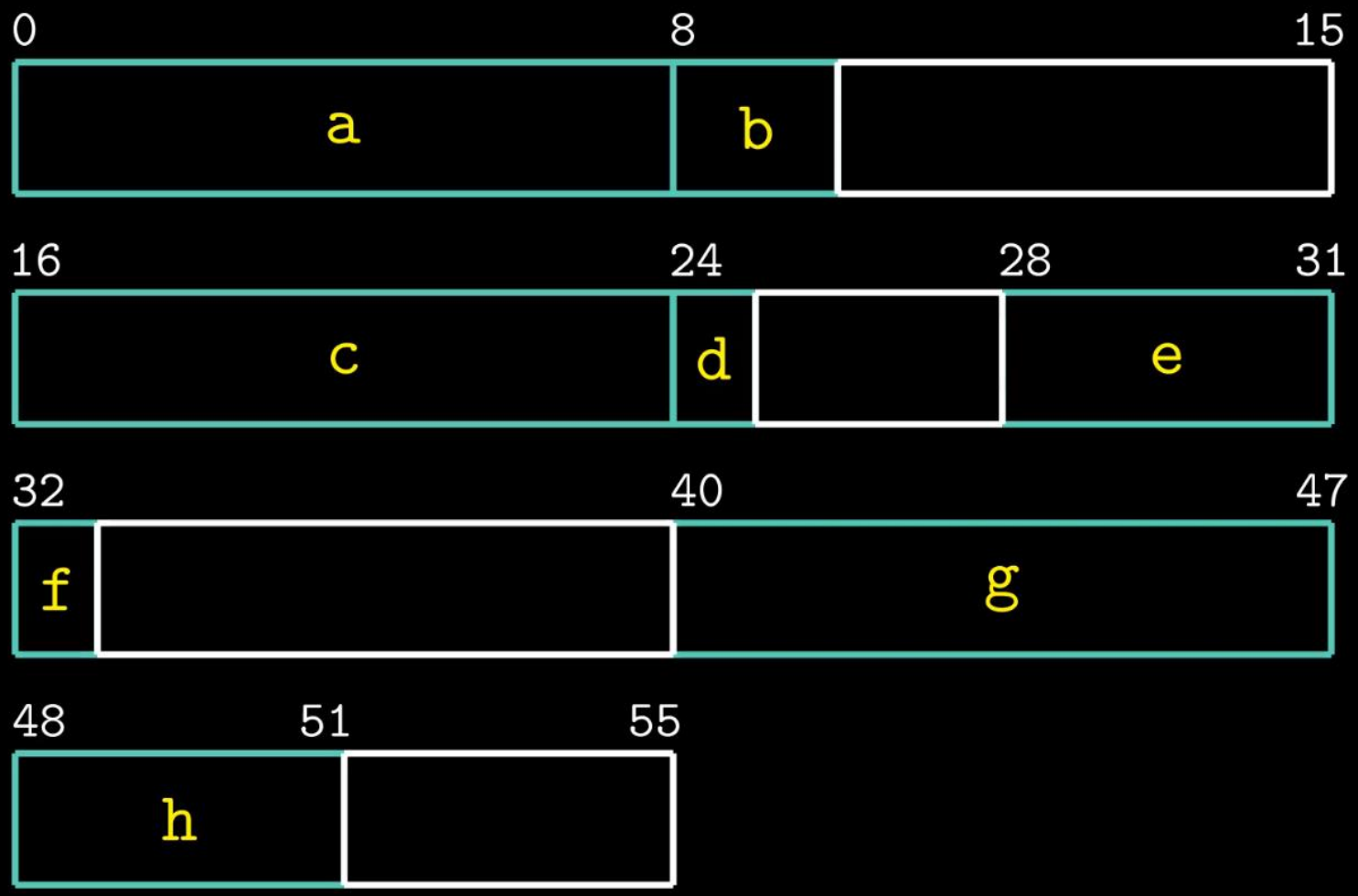
Data Alignment

```
struct {
    char      *a;
    short     b;
    double    c;
    char      d;
    float     e;
    char      f;
    long      g;
    int       h;
} rec;
```



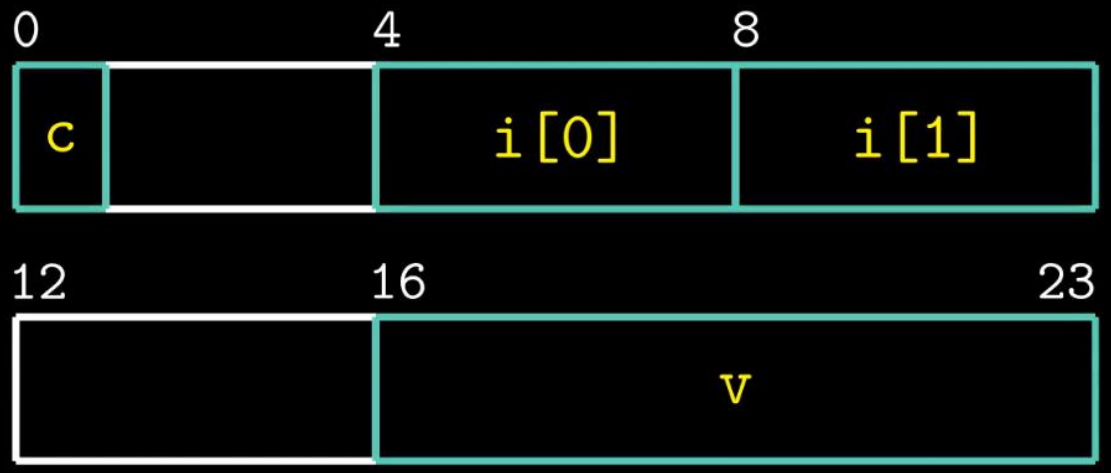
Data Alignment

```
struct {
    char      *a;
    short     b;
    double    c;
    char      d;
    float     e;
    char      f;
    long      g;
    int       h;
} rec;
```

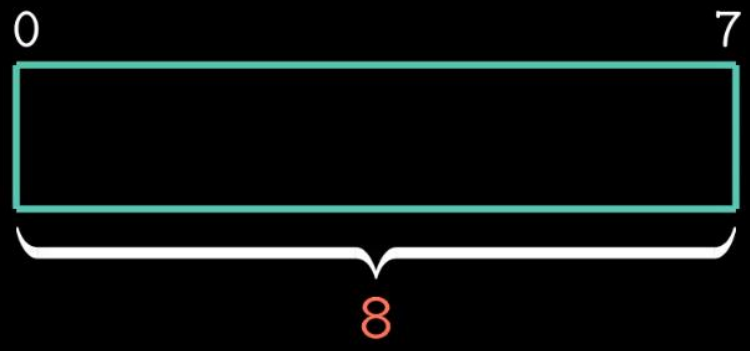


Unions

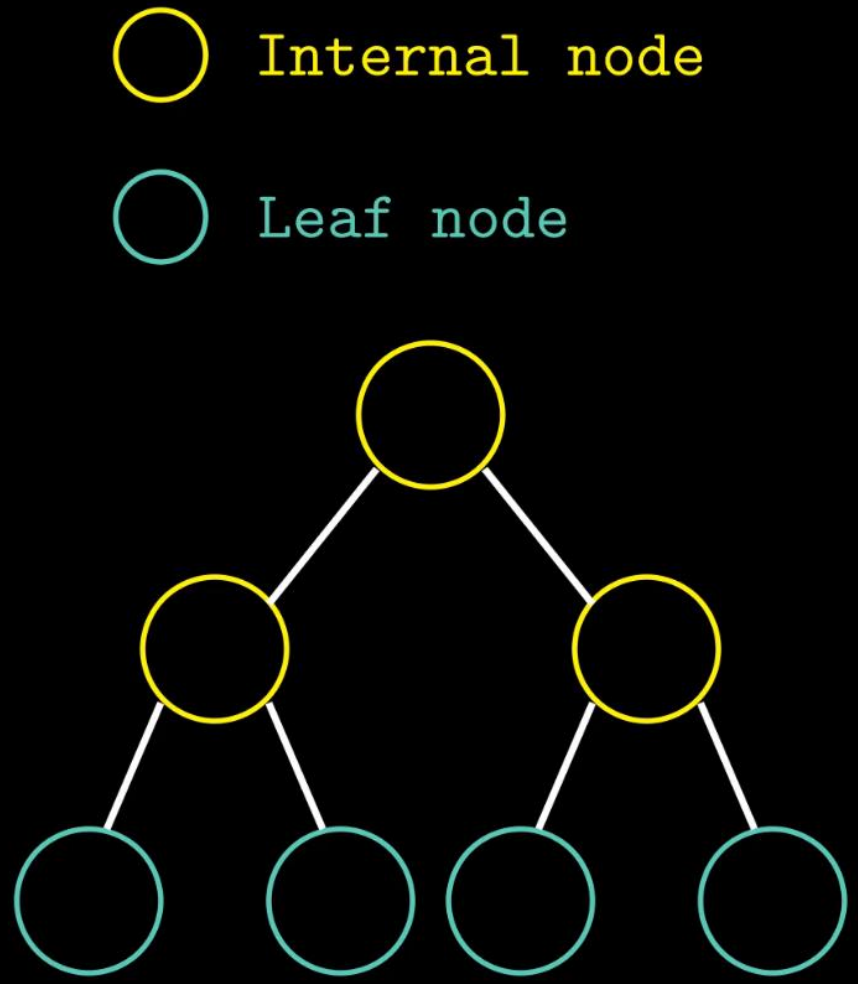
```
struct S3 {
    char    c;
    int     i[2];
    double  v;
};
```




```
union U3 {
    char    c;
    int     i[2];
    double  v;
};
```



Unions

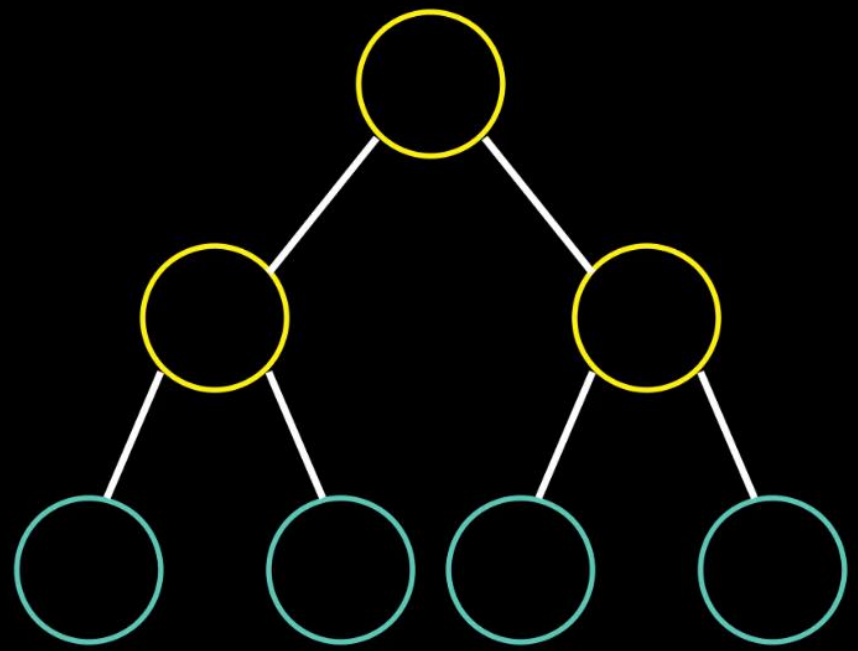



```
struct node_s {
    struct node_s *left;
    struct node_s *right;
    double data[2];
};

union node_u {
    struct {
        union node_u *left;
        union node_u *right;
    } internal;
    double data[2];
};
```

Unions

Internal node

Leaf node



```
typedef enum {N_LEAF,N_INTERNAL} nodetype_t;

struct node_t {
    nodetype_t type;
    union {
        struct {
            struct node_t *left;
            struct node_t *right;
        } internal;
        double data[2];
    } info;
};
```

Unions

```
unsigned long u = (unsigned long) d;

unsigned long double2bits (double d) {
    union {
        double d;
        unsigned long u;
    } temp;
    temp.d = d;
    return temp.u;
};
```

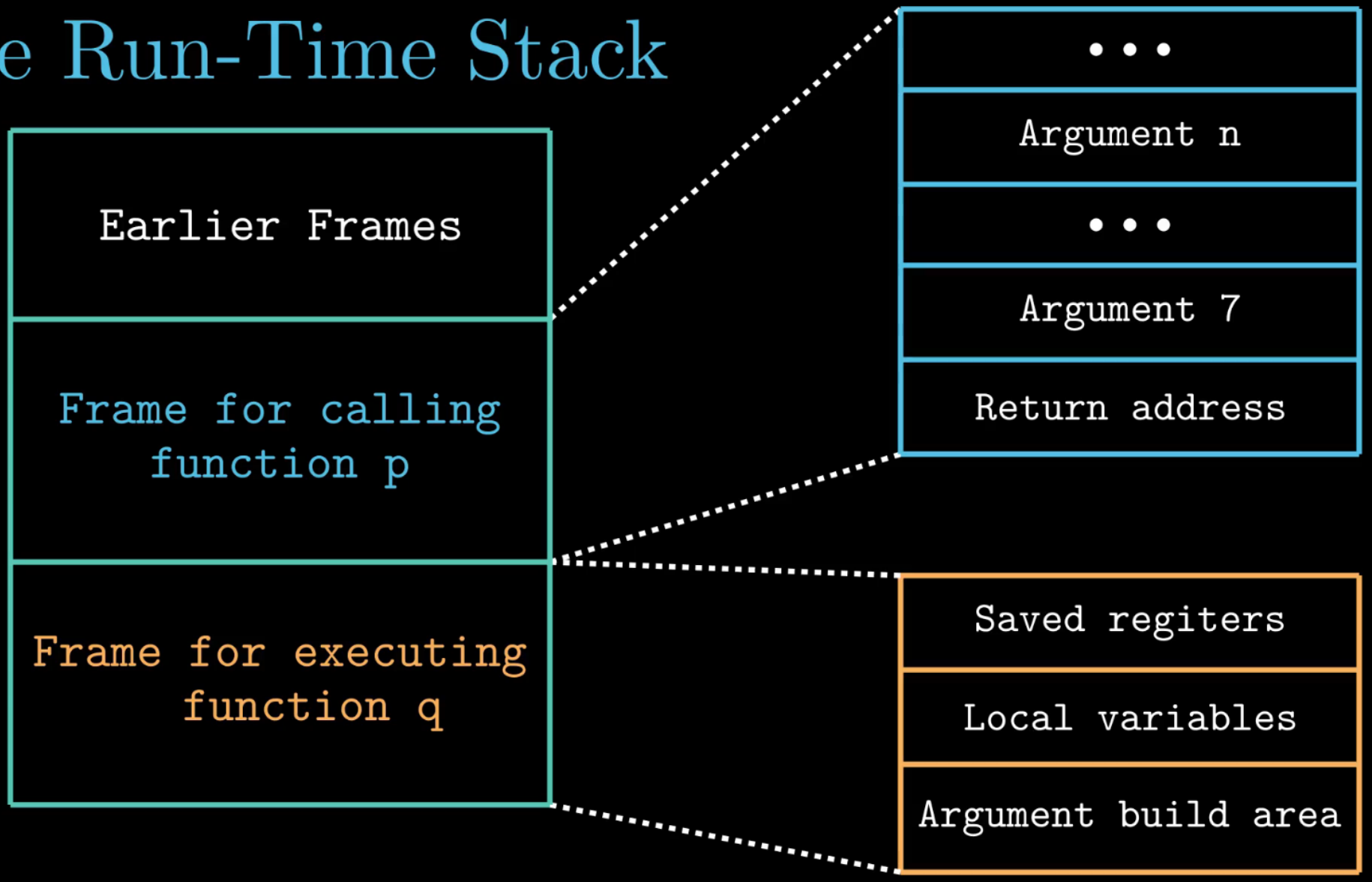
Computer Systems

A Programmer's Perspective

Chapter 3: Machine-Level Representation of Programs

(程序的机器级表示)

The Run-Time Stack



Buffer Overflow

```
void echo()
{
    char buf[8];
    gets (buf);
    puts (buf);
}
```

```
#include <stdio.h>

char *gets(char *buf);
```

Buffer Overflow

```
void echo()
{
    char buf[8];
    gets (buf);
    puts (buf);
}
```

```
echo:
    subq $24, %rsp
    movq %rsp, %rdi
    call gets
    movq %rsp, %rdi
    call puts
    addq $24, %rsp
    ret
```

Buffer Overflow

Thwarting Buffer Overflow Attacks
(对抗缓冲区溢出攻击)

- Stack Randomization (栈随机化)
- Stack Corruption Detection (栈破坏检测)
- Limiting Executable Code Regions (限制可执行代码区域)

Stack Randomization

```
int main()
{
    long local;
    printf("local at %p\n", &local);
    return 0;
}
```

64 Linux: 0x7fff0001b698 ~ 0x7fffffffaa4a8

Stack Randomization

Address-Space Layout Randomization, ASLR

(地址空间布局随机化)

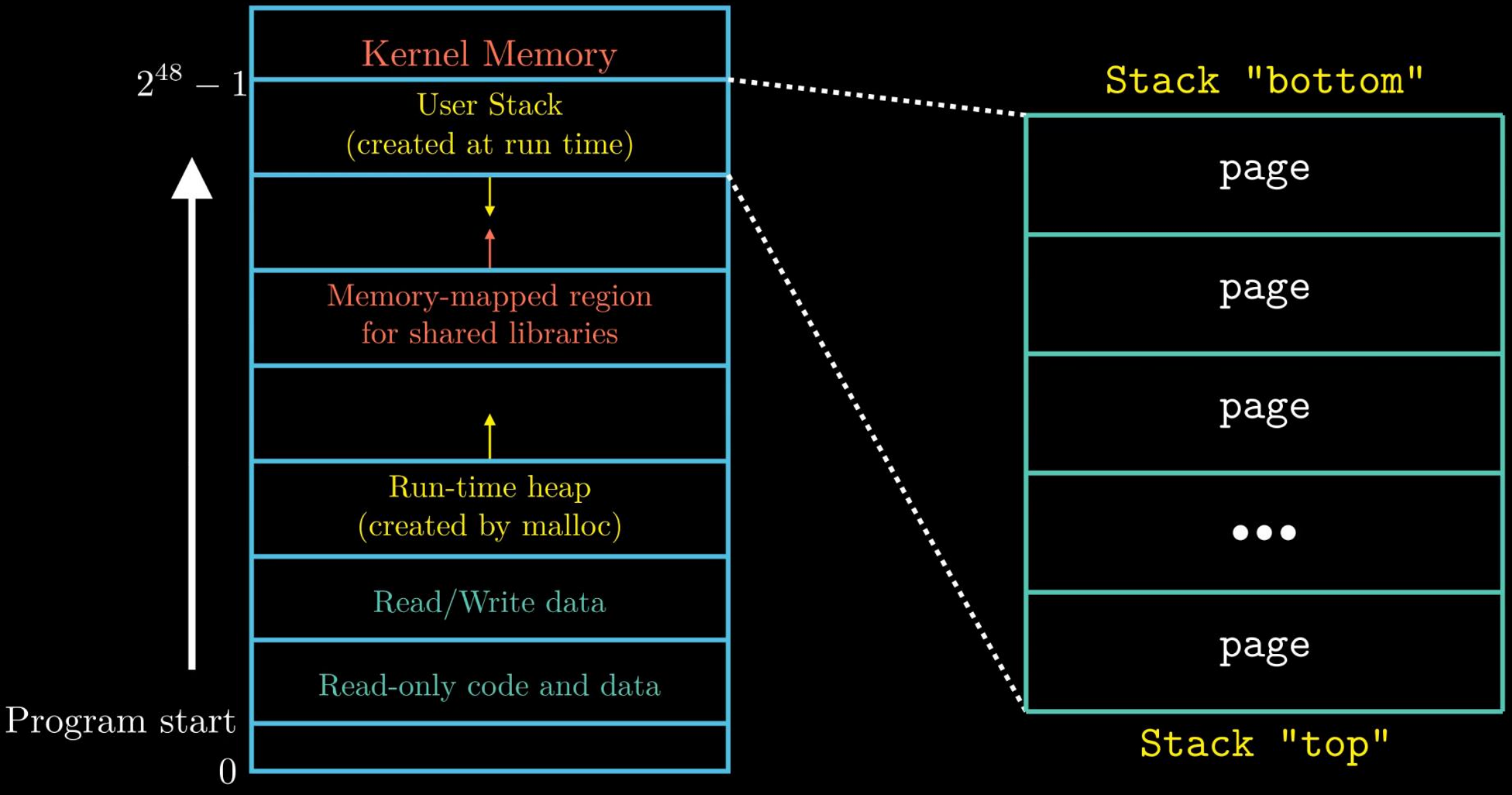
Stack Corruption Detection

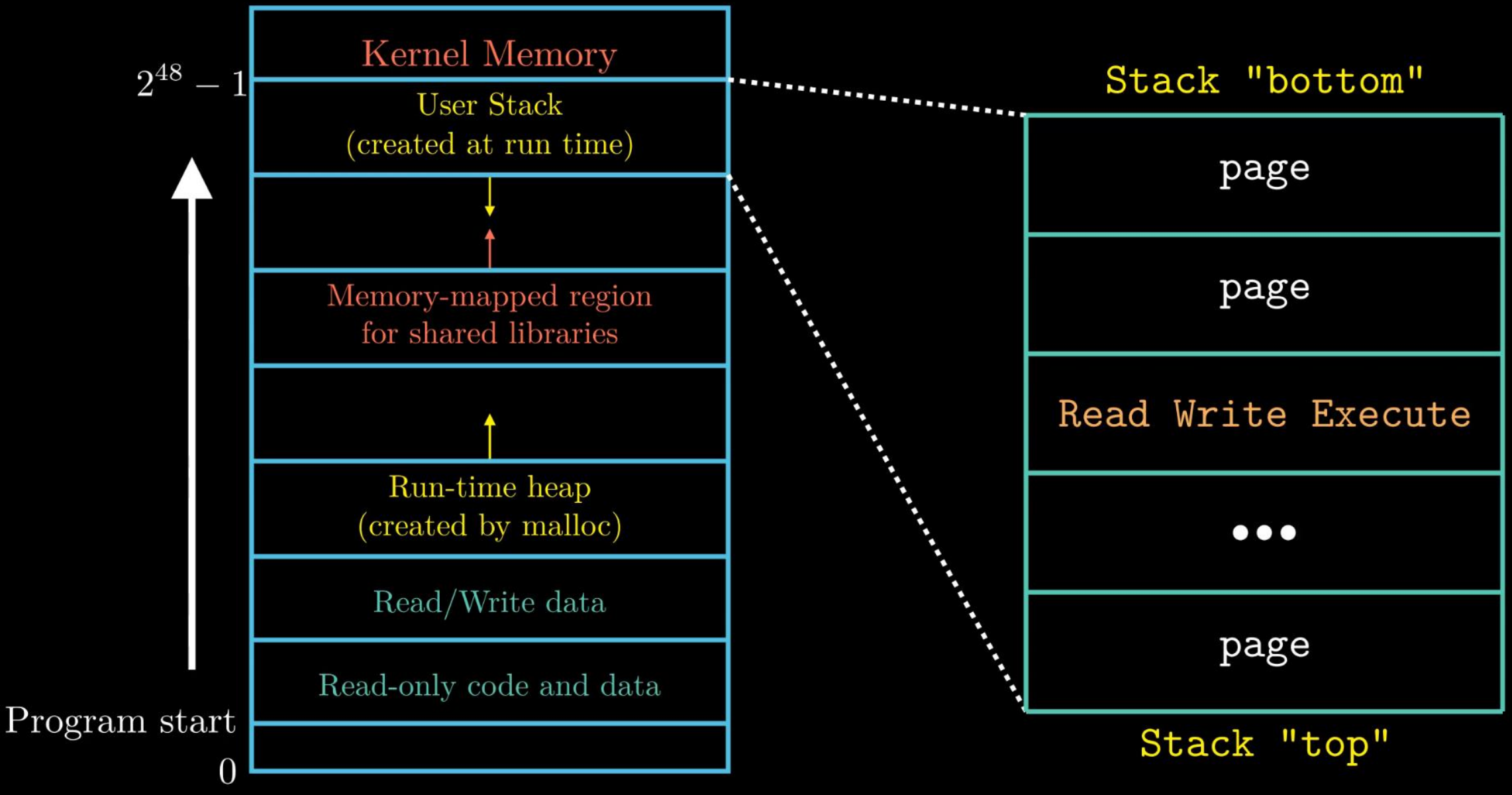
Stack Corruption Detection

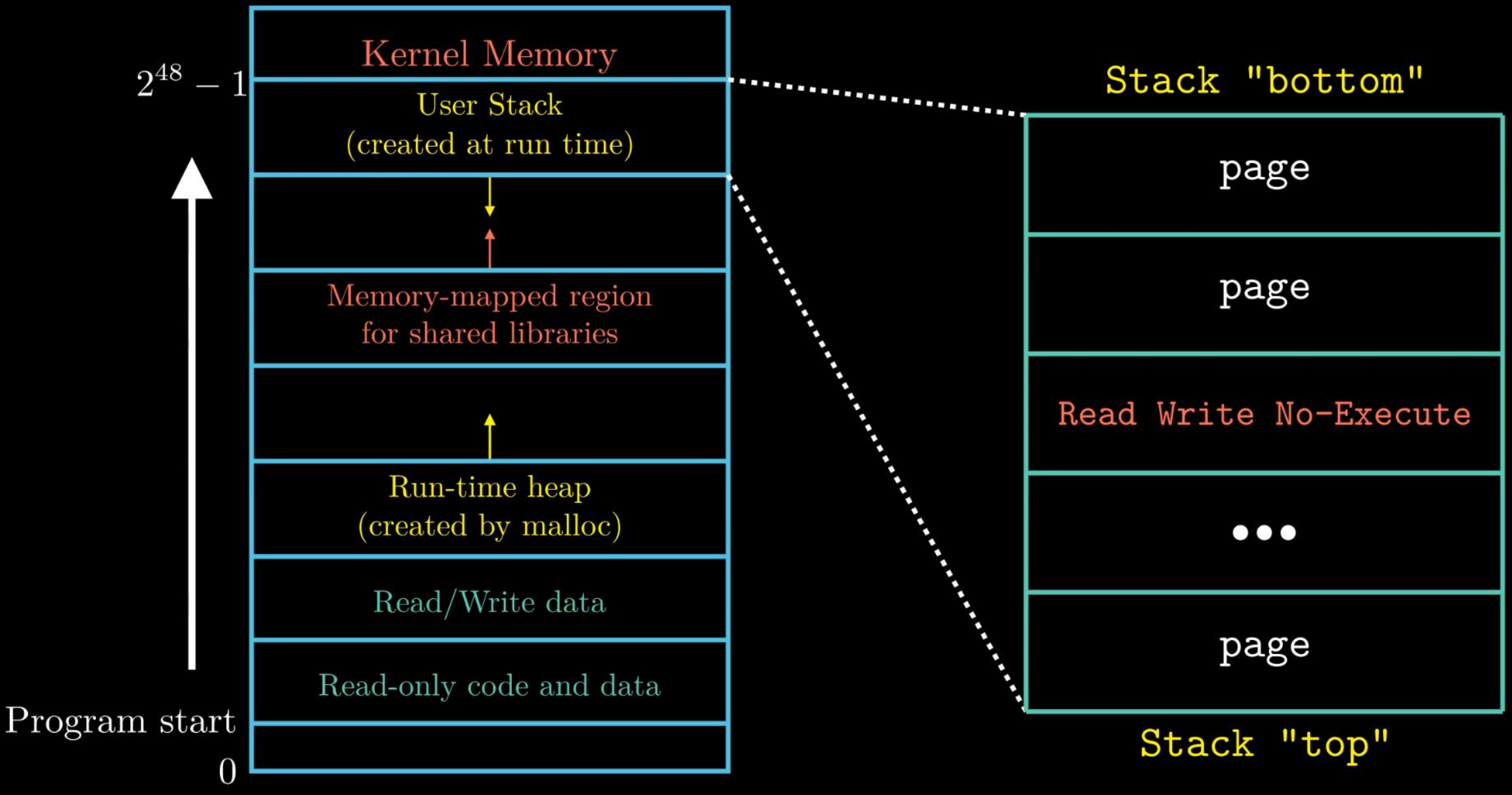
```
echo:
    subq $24, %rsp
    movq %fs:40, %rax
    movq %rax, 8(%rsp)
    xorl %eax, %eax
    movq %rsp, %rdi
    call gets
    movq %rsp, %rdi
    call puts
    movq 8(%rsp), %rax
    xorq %fs:40, %rax
    je .L9
    call __stack_chk_fail
.L9:
    addq $24, %rsp
    ret
```

Thwarting Buffer Overflow Attacks
(对抗缓冲区溢出攻击)

- Stack Randomization (栈随机化)
- Stack Corruption Detection (栈破坏检测)
- Limiting Executable Code Regions (限制可执行代码区域)


$2^{48} - 1$
Kernel Memory
User Stack
(created at run time)
Memory-mapped region
for shared libraries
Run-time heap
(created by malloc)
Read/Write data
Read-only code and data
Program start
0
Stack "bottom"
page
page
page
•••
page
Stack "top"


$2^{48} - 1$
Kernel Memory
User Stack
(created at run time)
Memory-mapped region
for shared libraries
Run-time heap
(created by malloc)
Read/Write data
Read-only code and data
Program start
0
Stack "bottom"
page
page
Read Write Execute
•••
page
Stack "top"


$2^{48} - 1$
Kernel Memory
User Stack
(created at run time)
Memory-mapped region
for shared libraries
Run-time heap
(created by malloc)
Read/Write data
Read-only code and data
Program start
0
Stack "bottom"
page
page
Read Write No-Execute
•••
page
Stack "top"

compute the absolute value of a number using only bit-level operations and straightline code. This lab helps students understand the bit-level representations of C data types and the bit-level behavior of the operations on data.

- *Bomb Lab [Updated 1/12/16]* (README, Writeup, Release Notes, Self-Study Handout)

  A "binary bomb" is a program provided to students as an object code file. When run, it prompts the user to type in 6 different strings. If any of these is incorrect, the bomb "explodes," printing an error message and logging the event on a grading server. Students must "defuse" their own unique bomb by disassembling and reverse engineering the program to determine what the 6 strings should be. The lab teaches students to understand assembly language, and also forces them to learn how to use a debugger. It's also great fun. A legendary lab among the CMU undergrads.

  Here's a Linux/x86-64 binary bomb that you can try out for yourself. The feature that notifies the grading server has been disabled, so feel free to explode this bomb with impunity. If you're an instructor with a CS:APP account, then you can download the solution.

- *Attack Lab* ***[Updated 1/11/16]*** (README, Writeup, Release Notes, Self-Study Handout)

  Note: This is the 64-bit successor to the 32-bit Buffer Lab. Students are given a pair of unique custom-generated x86-64 binary executables, called *targets*, that have buffer overflow bugs. One target is vulnerable to code injection attacks. The other is vulnerable to return-oriented programming attacks. Students are asked to modify the behavior of the targets by developing exploits based on either code injection or return-oriented programming. This lab teaches the students about the stack discipline and teaches them about the danger of writing code that is vulnerable to buffer overflow attacks.

  If you're a self-study student, here are a pair of Ubuntu 12.4 targets that you can try out for yourself. You'll need to run your targets using the **"-q"** option so that they don't try to contact a non-existent grading server. If you're an instructor with a CS:APP acount, you can download the solutions here.

- *Buffer Lab (IA32) [Updated 9/10/14]* (README, Writeup, Release Notes, Self-Study Handout)

  Note: This is the legacy 32-bit lab from CS:APP2e. It has been replaced by the

```
csapp@jiuqulangan:~/lab/target1$ ls
cookie.txt  ctarget  ctarget.s  farm.c  hex2raw  README.txt  rtarget
csapp@jiuqulangan:~/lab/target1$
```

`README.txt`: A file describing the contents of the directory

`ctarget`: An executable program vulnerable to *code-injection* attacks

`rtarget`: An executable program vulnerable to *return-oriented-programming* attacks

`cookie.txt`: An 8-digit hex code that you will use as a unique identifier in your attacks.

`farm.c`: The source code of your target's "gadget farm," which you will use in generating return-oriented programming attacks.

`hex2raw`: A utility to generate attack strings.

```
csapp@jiuqulangan:~/lab/target1$ ls
cookie.txt  ctarget  ctarget.s  farm.c  hex2raw  README.txt  rtarget
csapp@jiuqulangan:~/lab/target1$ ./ctarget
FAILED: Initialization error: Running on an illegal host [jiuqulangan]
csapp@jiuqulangan:~/lab/target1$
```

```
csapp@jiuqulangan:~/lab/target1$ ls
cookie.txt  ctarget  ctarget.s  farm.c  hex2raw  README.txt  rtarget
csapp@jiuqulangan:~/lab/target1$ ./ctarget
FAILED: Initialization error: Running on an illegal host [jiuqulangan]
csapp@jiuqulangan:~/lab/target1$ ./ctarget -q
Cookie: 0x59b997fa
Type string:
```

```
csapp@jiuqulangan:~/lab/target1$ ls
cookie.txt  ctarget  ctarget.s  farm.c  hex2raw  README.txt  rtarget
csapp@jiuqulangan:~/lab/target1$ ./ctarget
FAILED: Initialization error: Running on an illegal host [jiuqulangan]
csapp@jiuqulangan:~/lab/target1$ ./ctarget -q
Cookie: 0x59b997fa
Type string:123456dasdasd
No exploit.  Getbuf returned 0x1
Normal return
csapp@jiuqulangan:~/lab/target1$
```

```
csapp@jiuqulangan:~/lab/target1$ ls
cookie.txt  ctarget  ctarget.s  farm.c  hex2raw  README.txt  rtarget
csapp@jiuqulangan:~/lab/target1$ ./ctarget
FAILED: Initialization error: Running on an illegal host [jiuqulangan]
csapp@jiuqulangan:~/lab/target1$ ./ctarget -q
Cookie: 0x59b997fa
Type string:123456dasdasd
No exploit.  Getbuf returned 0x1
Normal return
csapp@jiuqulangan:~/lab/target1$
```

```
csapp@jiuqulangan:~/lab/target1$ ls
cookie.txt  ctarget  ctarget.s  farm.c  hex2raw  README.txt  rtarget
csapp@jiuqulangan:~/lab/target1$ ./ctarget
FAILED: Initialization error: Running on an illegal host [jiuqulangan]
csapp@jiuqulangan:~/lab/target1$ ./ctarget -q
Cookie: 0x59b997fa
Type string:123456dasdasd
No exploit.  Getbuf returned 0x1
Normal return
csapp@jiuqulangan:~/lab/target1$
```

```
1 void test()
2 {
3     int val;
4     val = getbuf();
5     printf("No exploit.  Getbuf returned 0x%x\n", val);
6 }
```

```
csapp@jiuqulangan:~/lab/target1$ ls
cookie.txt  ctarget  ctarget.s  farm.c  hex2raw  README.txt  rtarget
csapp@jiuqulangan:~/lab/target1$ ./ctarget
FAILED: Initialization error: Running on an illegal host [jiuqulangan]
csapp@jiuqulangan:~/lab/target1$ ./ctarget -q
Cookie: 0x59b997fa
Type string:00 00 00 00 00 00 00 11 11  11 1 1 1 1 1 11  11 1 1 1 1 1 1 1 11  11 1 1 1 1 1 1 1 1
Ouch!: You caused a segmentation fault!
Better luck next time
FAIL: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:FAIL:0xffffffff:ctarget:0:30 30 20 30 30 20 30 30 20 30 30 20 30 30 20 30 30 20 31 31 20 31 31 20 20 31
31 20 31 20 31 20 31 20 31 20 31 20 31 31 20 20 31 31 20 31 20 31 20 31 20 31 20 31 20 31 20 31 20 31 20 31 31 20 20 31 31 20 31 20
 31 20 31 20 31 20 31 20 31 20 31 20 31 20
csapp@jiuqulangan:~/lab/target1$
```

Buffer Overflow

Buffer Overflow

```
void echo()
{
    char buf[8];
    gets (buf);
    puts (buf);
}
```

```
echo:
    subq $24, %rsp
    movq %rsp, %rdi
    call gets
    movq %rsp, %rdi
    call puts
    addq $24, %rsp
    ret
```

```
csapp@jiuqulangan:~/lab/target1$ ls
cookie.txt  ctarget  farm.c  hex2raw  README.txt  rtarget
csapp@jiuqulangan:~/lab/target1$ objdump -d ctarget > ctarget.s
csapp@jiuqulangan:~/lab/target1$
```

```
ctarget:     file format elf64-x86-64


Disassembly of section .init:

0000000000400c48 <_init>:
  400c48:   48 83 ec 08             sub    $0x8,%rsp
  400c4c:   e8 6b 02 00 00          callq  400ebc <call_gmon_start>
  400c51:   48 83 c4 08             add    $0x8,%rsp
  400c55:   c3                      retq

Disassembly of section .plt:

0000000000400c60 <.plt>:
  400c60:   ff 35 8a 33 20 00       pushq  0x20338a(%rip)        # 603ff0 <_GLOBAL_OFFSET_TABLE_+0x8>
  400c66:   ff 25 8c 33 20 00       jmpq   *0x20338c(%rip)        # 603ff8 <_GLOBAL_OFFSET_TABLE_+0x10>
  400c6c:   0f 1f 40 00             nopl   0x0(%rax)

0000000000400c70 <strcasecmp@plt>:
  400c70:   ff 25 8a 33 20 00       jmpq   *0x20338a(%rip)        # 604000 <strcasecmp@GLIBC_2.2.5>
  400c76:   68 00 00 00 00          pushq  $0x0
  400c7b:   e9 e0 ff ff ff          jmpq   400c60 <.plt>

0000000000400c80 <__errno_location@plt>:
  400c80:   ff 25 82 33 20 00       jmpq   *0x203382(%rip)        # 604008 <__errno_location@GLIBC_2.2.5>
  400c86:   68 01 00 00 00          pushq  $0x1
  400c8b:   e9 d0 ff ff ff          jmpq   400c60 <.plt>

0000000000400c90 <srandom@plt>:
  400c90:   ff 25 7a 33 20 00       jmpq   *0x20337a(%rip)        # 604010 <srandom@GLIBC_2.2.5>
/getbuf
```

```
  401783:   89 44 24 ec             mov    %eax,-0x14(%rsp)
  401787:   ba 00 00 00 00          mov    $0x0,%edx
  40178c:   b8 00 00 00 00          mov    $0x0,%eax
  401791:   eb 0b                   jmp    40179e <scramble+0x416>
  401793:   89 d1                   mov    %edx,%ecx
  401795:   8b 4c 8c c8             mov    -0x38(%rsp,%rcx,4),%ecx
  401799:   01 c8                   add    %ecx,%eax
  40179b:   83 c2 01                add    $0x1,%edx
  40179e:   83 fa 09                cmp    $0x9,%edx
  4017a1:   76 f0                   jbe    401793 <scramble+0x40b>
  4017a3:   f3 c3                   repz retq
  4017a5:   90                      nop
  4017a6:   90                      nop
  4017a7:   90                      nop

00000000004017a8 <getbuf>:
  4017a8:   48 83 ec 28             sub    $0x28,%rsp
  4017ac:   48 89 e7                mov    %rsp,%rdi
  4017af:   e8 8c 02 00 00          callq  401a40 <Gets>
  4017b4:   b8 01 00 00 00          mov    $0x1,%eax
  4017b9:   48 83 c4 28             add    $0x28,%rsp
  4017bd:   c3                      retq
  4017be:   90                      nop
  4017bf:   90                      nop

00000000004017c0 <touch1>:
  4017c0:   48 83 ec 08             sub    $0x8,%rsp
  4017c4:   c7 05 0e 2d 20 00 01    movl   $0x1,0x202d0e(%rip)        # 6044dc <vlevel>
  4017cb:   00 00 00
  4017ce:   bf c5 30 40 00          mov    $0x4030c5,%edi
  4017d3:   e8 e8 f4 ff ff          callq  400cc0 <puts@plt>
```

```
1 00 00 00 00 00 00 00 00 00 00
2 00 00 00 00 00 00 00 00 00 00
3 00 00 00 00 00 00 00 00 00 00
4 00 00 00 00 00 00 00 00 00 00
5 c0 17 40 00 00 00 00 00
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"key-file-00" 5L, 148C
```

```
csapp@jiuqulangan:~/lab/target1$ ls
cookie.txt  ctarget  ctarget.s  farm.c  hex2raw  key-file-00  key-file-11  README.txt  rtarget
csapp@jiuqulangan:~/lab/target1$ ./hex2raw < key-file-00 | ./ctarget -q
Cookie: 0x59b997fa
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:1:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C0 17 40 00 00 00 00 00
csapp@jiuqulangan:~/lab/target1$
```

```
csapp@jiuqulangan:~/lab/target1$ ls
cookie.txt  ctarget  ctarget.s  farm.c  hex2raw  key-file-00  key-file-11  README.txt  rtarget
csapp@jiuqulangan:~/lab/target1$ ./hex2raw < key-file-00 | ./ctarget -q
Cookie: 0x59b997fa
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:1:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C0 17 40 00 00 00 00 00
csapp@jiuqulangan:~/lab/target1$ ./hex2raw < key-file-11 | ./ctarget -q
Cookie: 0x59b997fa
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:1:11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 C0 17 40 00 00 00 00 00
csapp@jiuqulangan:~/lab/target1$
```

```
11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11
c0 17 40 00 00 00 00 00
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"key-file-11" 5L, 144C
```

Register

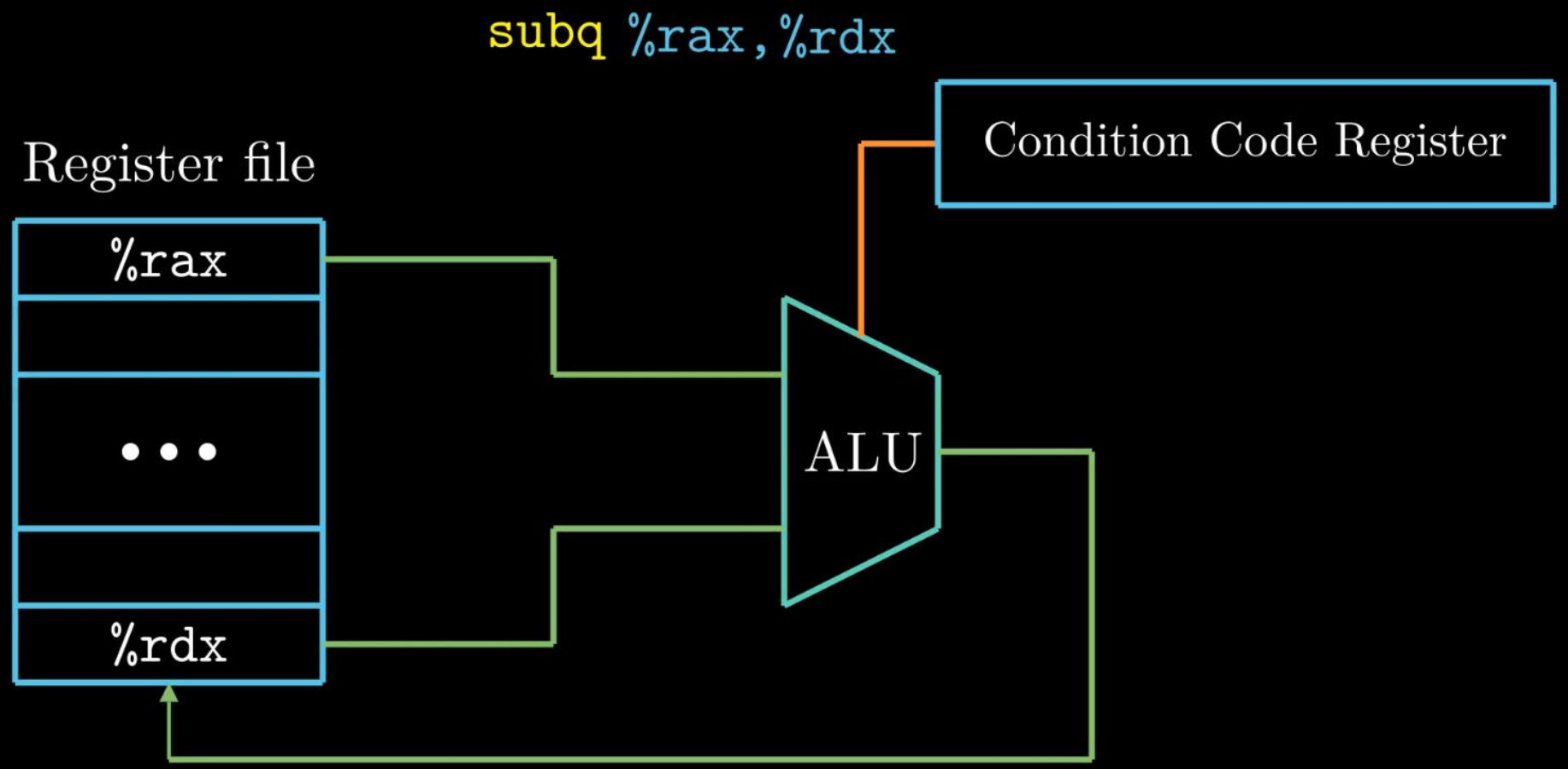

63
31
15
7
0
%rax
%eax
%ax
%al
%rsi
%esi
%si
%sil
%rbx
%ebx
%bx
%bl
%rdi
%edi
%di
%dil
%rcx
%ecx
%cx
%cl
%rbp
%ebp
%bp
%bpl
%rdx
%edx
%dx
%dl
%rsp
%esp
%sp
%spl

Register

63 0

%rax	Return value-Caller saved
%rbx	Callee saved
%rcx	Argument #4 - Caller saved
%rdx	Argument #3 - Caller saved
%rsi	Argument #2 - Caller saved
%rdi	Argument #1 - Caller saved
%rbp	Callee saved
%rsp	Stack pointer

Control


subq %rax,%rdx
Condition Code Register
Register file
%rax
• • •
%rdx
ALU

Condition Code Register

CF —— Carry Flag(进位标志)

ZF —— Zero Flag(零标志)

SF —— Sign Flag(符号标志)

OF —— Overflow Flag(溢出标志)

The Run-Time Stack