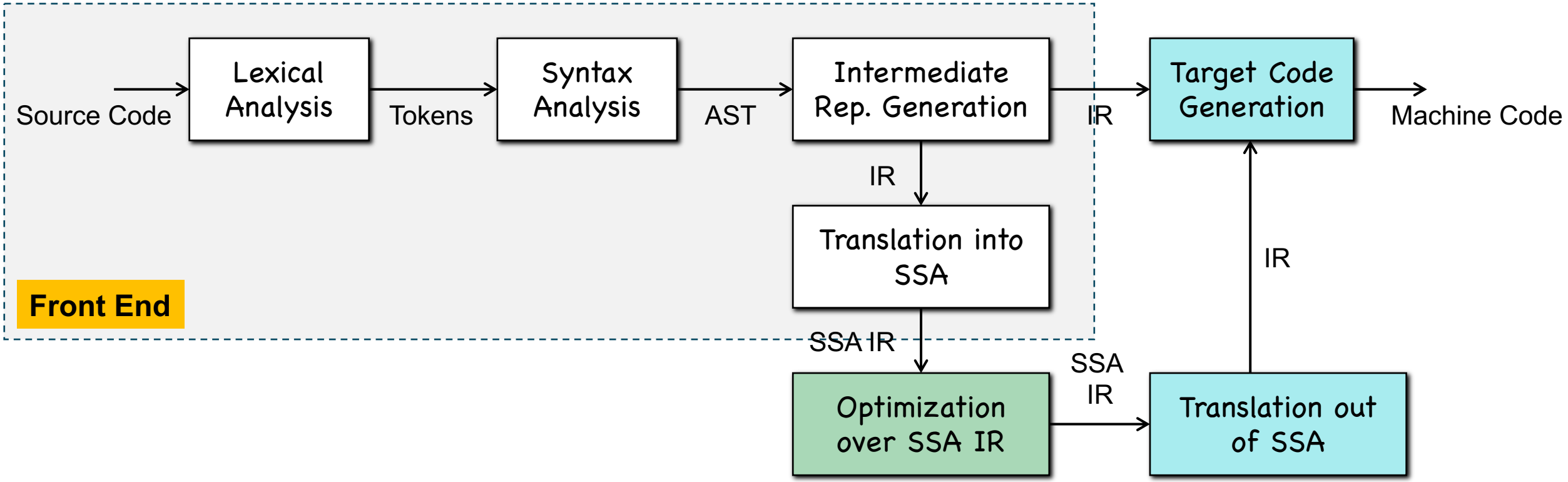
Made by Qingkai Shi
qingkaishi@nju.edu.cn

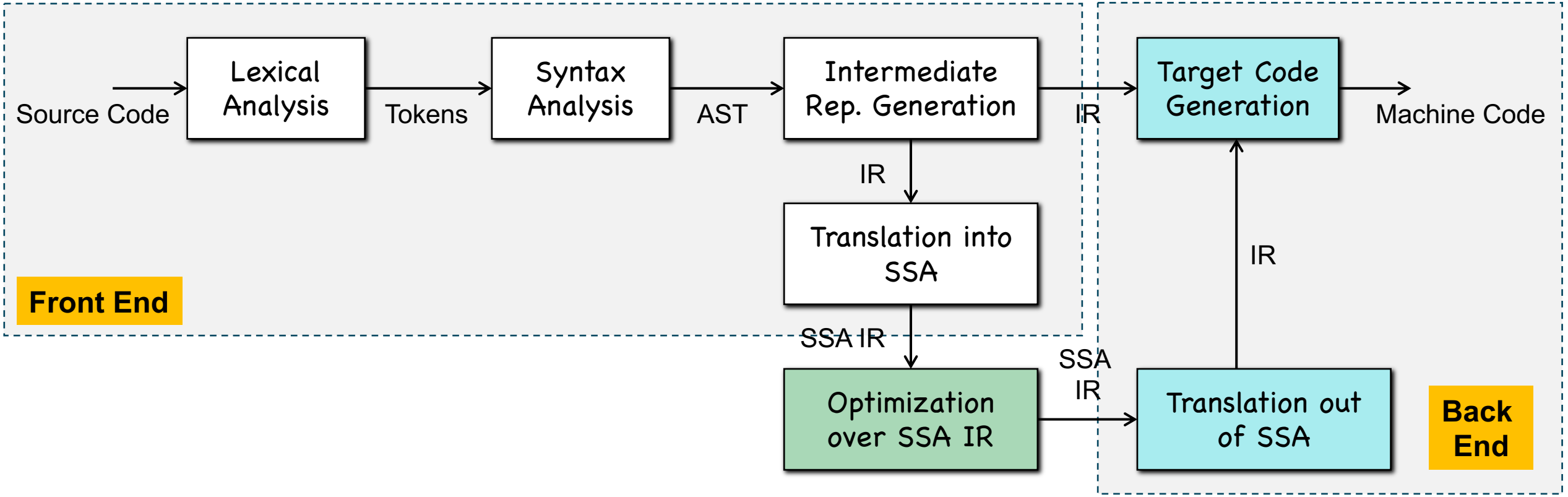
南京大学编译技术
及软件安全研究组

Chapter 8-1
Target Code Generation

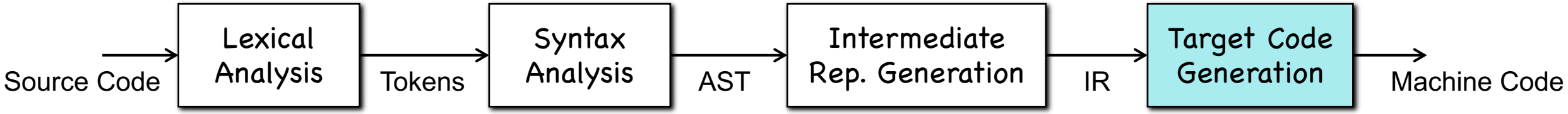
Target Code Generation



Target Code Generation

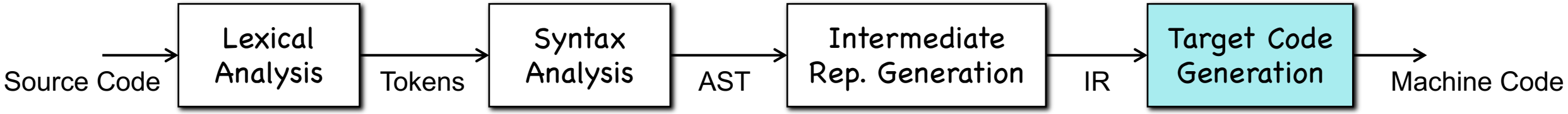


Target Code Generation



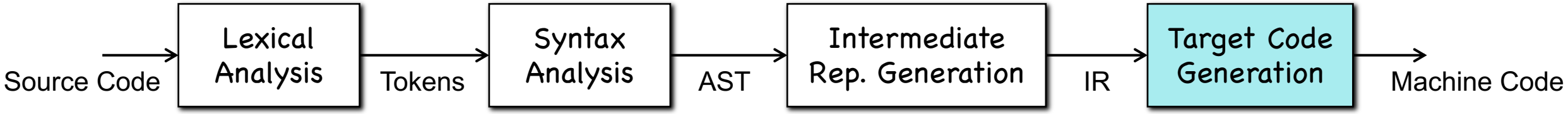
- **Code Generation**/Target Code Model /Memory Allocation

Target Code Generation



- **Code Generation**/Target Code Model
  /Memory Allocation
- **Gen Better Code**/In-Block Optimization
  /Peephole Optimization

Target Code Generation

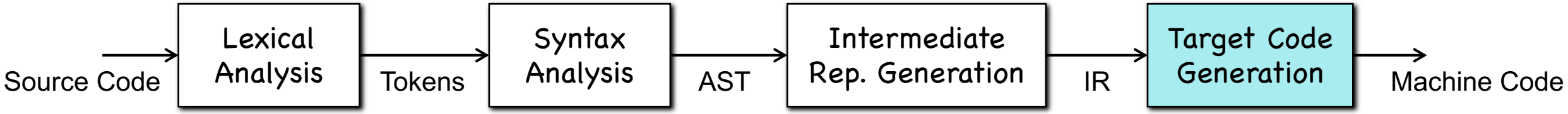


- **Code Generation**/Target Code Model /Memory Allocation
- **Gen Better Code**/In-Block Optimization /Peephole Optimization

```
LD   R0, a        // R0 = a
ADD  R0, R0, #1   // R0 = R0 + 1
ST   a, R0        // a = R0
```

Possible target code for a = a + 1

Target Code Generation



- **Code Generation**/Target Code Model /Memory Allocation
- **Gen Better Code**/In-Block Optimization /Peephole Optimization

```
LD   R0, a        // R0 = a
ADD  R0, R0, #1   // R0 = R0 + 1
ST   a, R0        // a = R0
```

Possible target code for a = a + 1

How about "INC a" ??

PART I: Target Code Generation

Recap: Three-Address Code

- What is three-address code? What does “address” mean?

Recap: Three-Address Code

- What is three-address code? What does “address” mean?

```
do i = i + 1; while (a[i + 2] < v);
```

```
L:   t1 = i + 1
     i = t1
     t2 = i + 2
     t3 = a [ t2 ]
     if t3 < v goto L
```

Symbolic Labels

```
100:  t1 = i + 1
101:  i = t1
102:  t2 = i + 2
103:  t3 = a [ t2 ]
104:  if t3 < v goto 100
```

Numeric Labels

Target Machine

- Memory (Heap/Stack/…). Each byte has an address.
- n Registers: R0, R1, …, Rn-1. Each four bytes.

Target Machine

- Memory (Heap/Stack/…). Each byte has an address.
- n Registers: R0, R1, …, Rn-1. Each four bytes.
- Load/Store/Calculation/Jump/… (Like x86 assembly)

Target Machine

- Memory (Heap/Stack/…). Each byte has an address.
- n Registers: R0, R1, …, Rn-1. Each four bytes.
- Load/Store/Calculation/Jump/… (Like x86 assembly)

- Load/Store
  - LD *R0*, *addr*
  - LD *R0*, *R1*
  - LD *R0*, #500
  - ST *addr*, *R0* **(Each LD/ST loads/stores a 4-byte integer)**

Target Machine

- Memory (Heap/Stack/…). Each byte has an address.
- n Registers: R0, R1, …, Rn-1. Each four bytes.
- Load/Store/Calculation/Jump/… (Like x86 assembly)

- Load/Store
  - LD *R0*, *addr*
  - LD *R0*, *R1*
  - LD *R0*, #500
  - ST *addr*, *R0*
- Calculation
  - OP *dst*, src_1, src_2
  - *e.g.*, SUB *R0*, *R1*, *R2*

(Each LD/ST loads/stores a 4-byte integer)

Target Machine

- Memory (Heap/Stack/…). Each byte has an address.
- n Registers: R0, R1, …, Rn-1. Each four bytes.
- Load/Store/Calculation/Jump/… (Like x86 assembly)

- Load/Store
  - LD *R0*, *addr*
  - LD *R0*, *R1*
  - LD *R0*, #500
  - ST *addr*, *R0*
- Calculation
  - OP *dst*, src_1, src_2
  - *e.g.*, SUB *R0*, *R1*, *R2*
- Jump
  - BR *L/addr*
  - Bcond *R*, *L/addr*
  - *e.g.*, BLTZ *R*, *L*

(Each LD/ST loads/stores a 4-byte integer)

Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y              // R1 = y
LD   R2, z              // R2 = z
SUB  R1, R1, R2         // R1 = R1 - R2
ST   x, R1              // x = R1
```

x = y - z

Target Machine

- Load/Store/Calculation/Jump/…

```
LD  R1, y          // R1 = y
LD  R2, z          // R2 = z
SUB R1, R1, R2     // R1 = R1 - R2
ST  x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

Target Machine

- Load/Store/Calculation/Jump/…

```
LD  R1, y          // R1 = y
LD  R2, z          // R2 = z
SUB R1, R1, R2     // R1 = R1 - R2
ST  x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

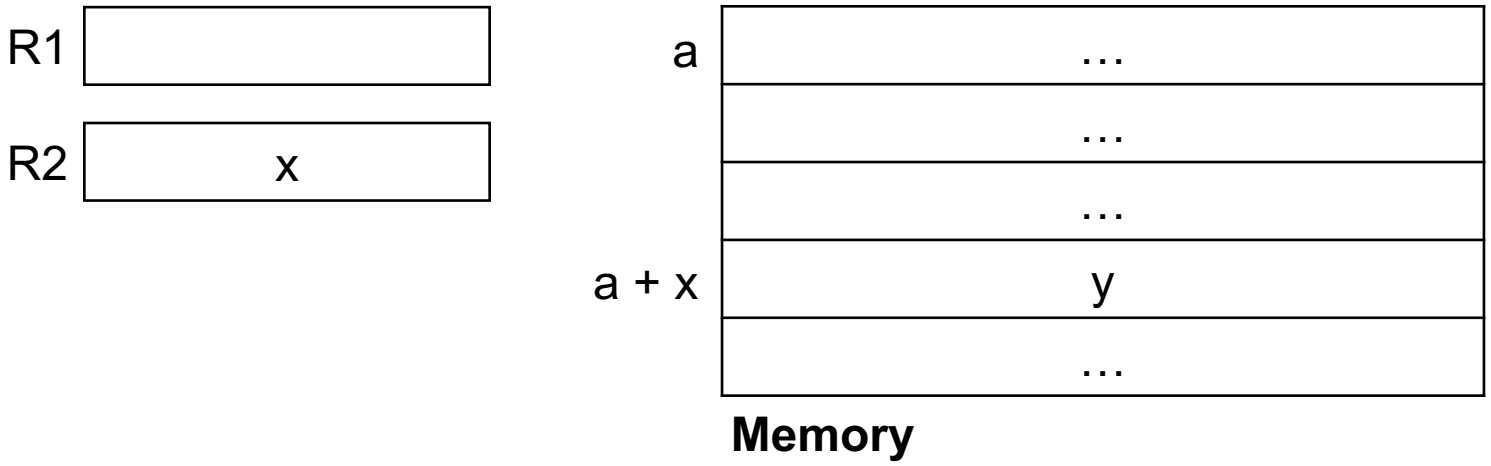
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**



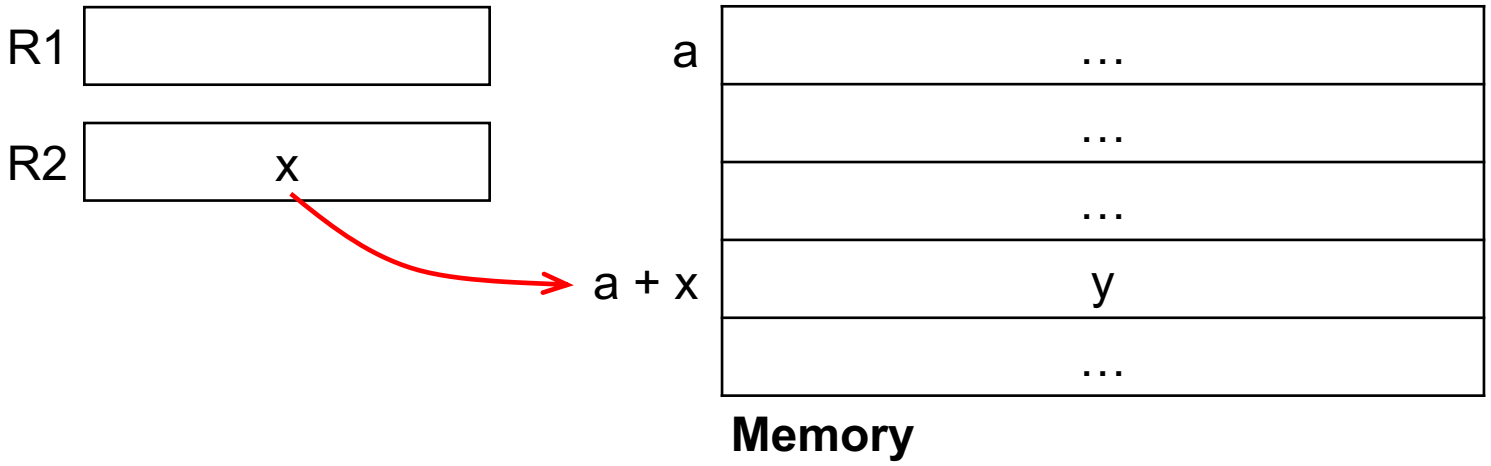
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**



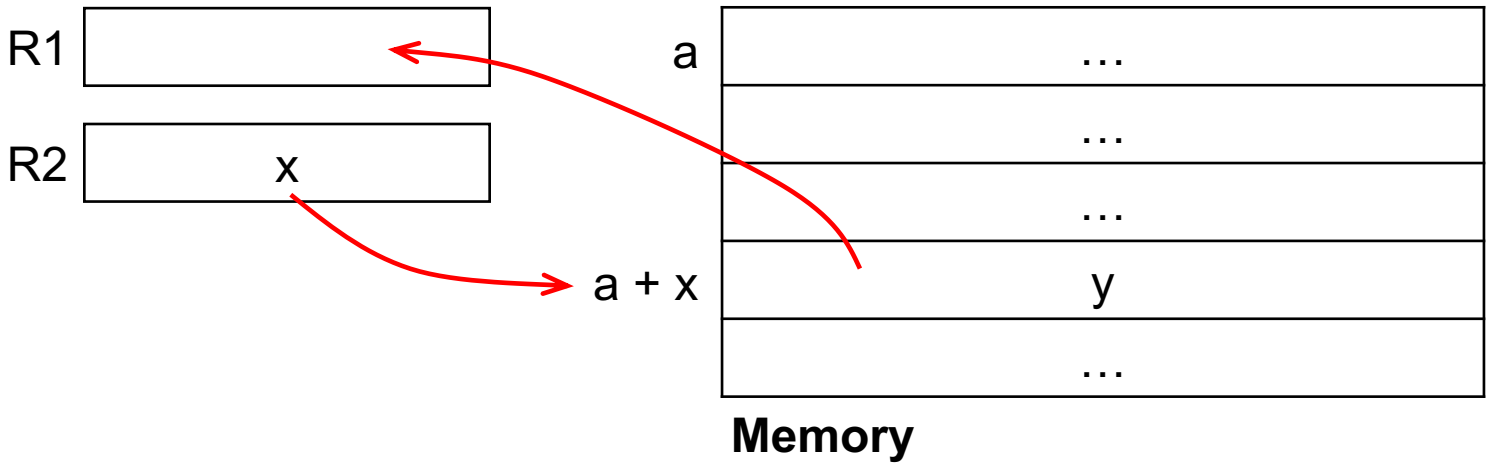
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

x = y - z



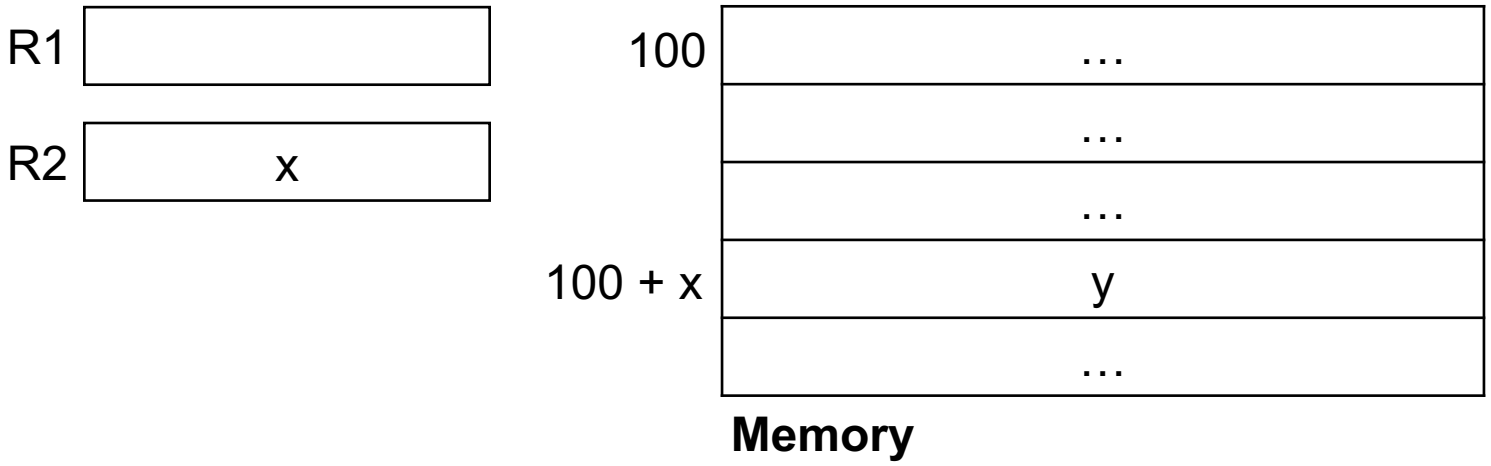
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**



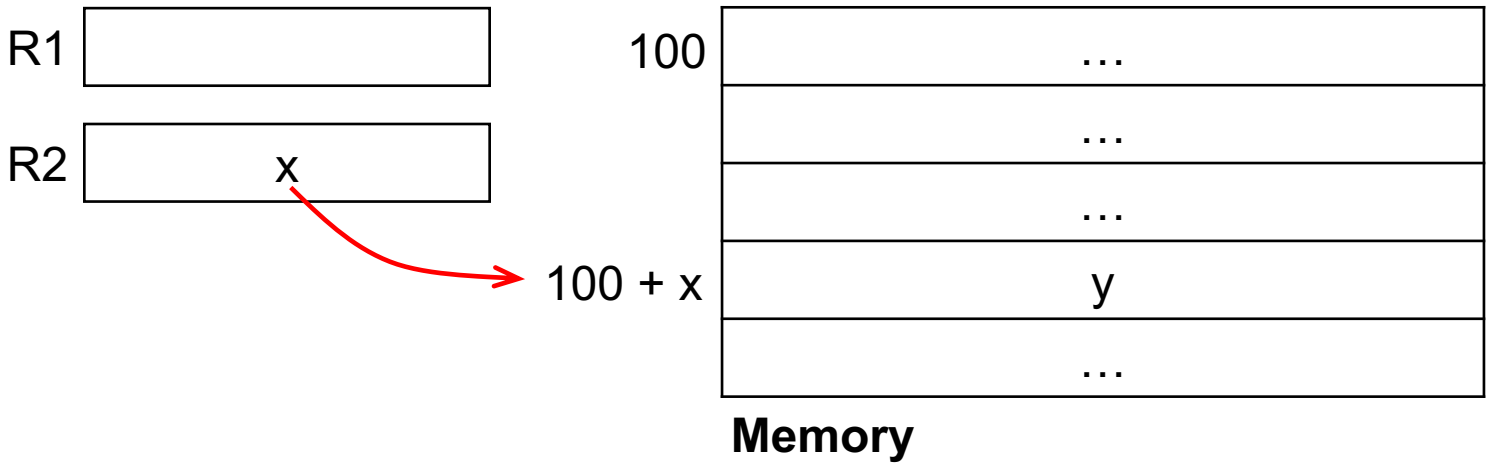
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**



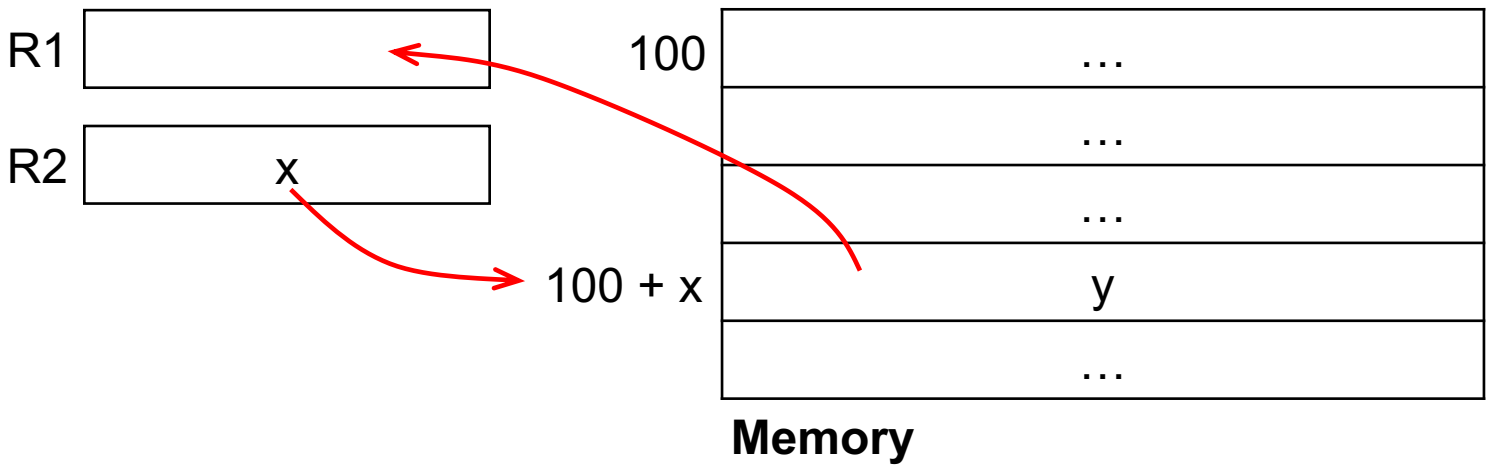
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

x = y - z



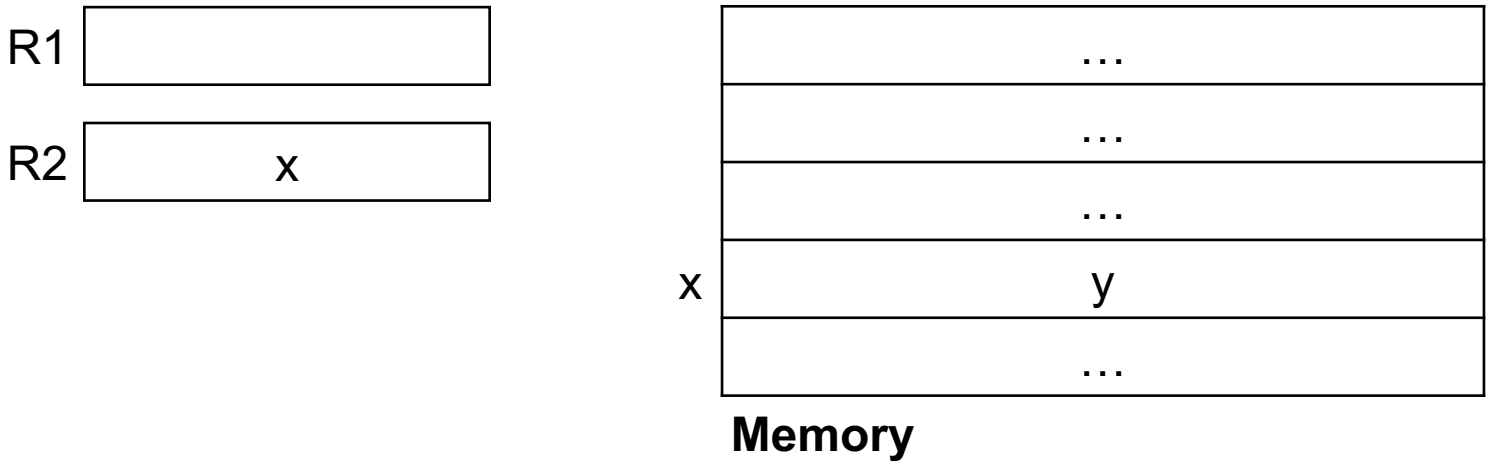
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**



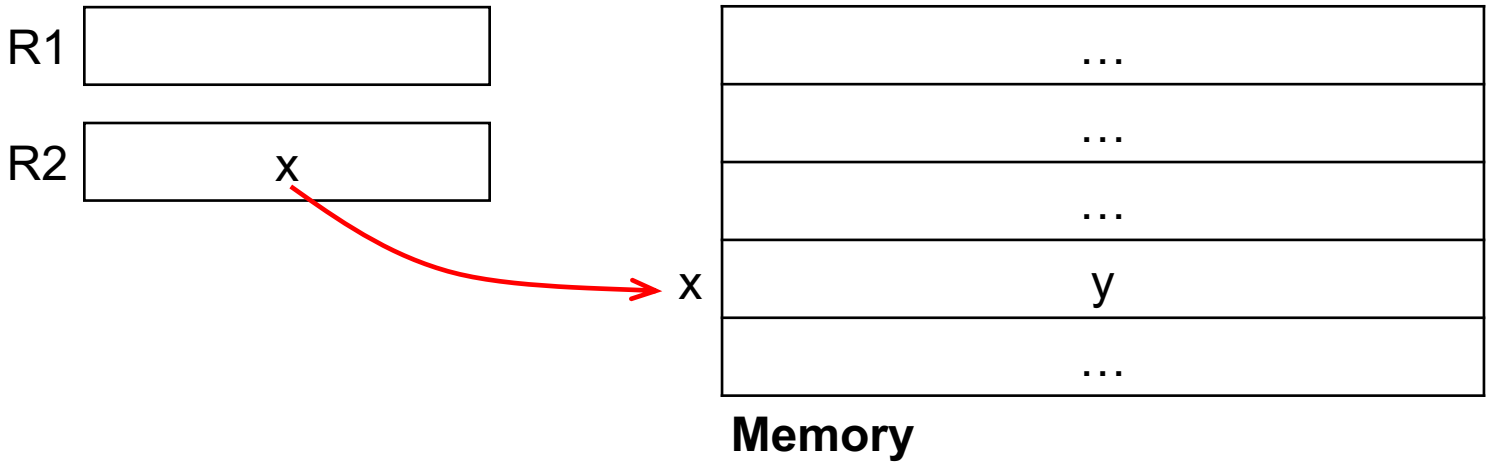
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**



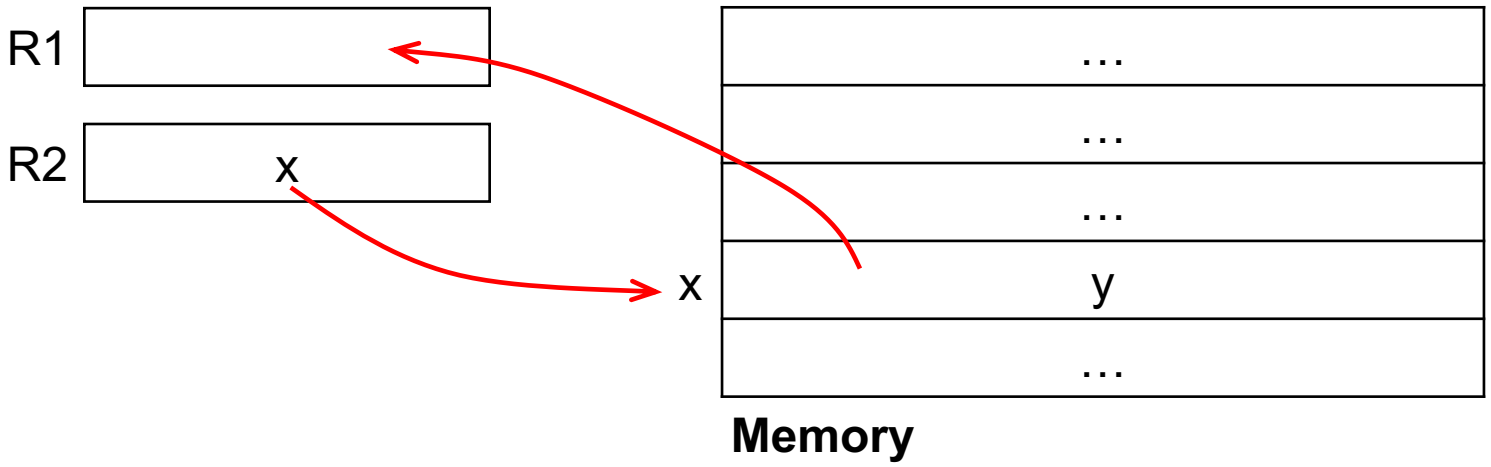
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**



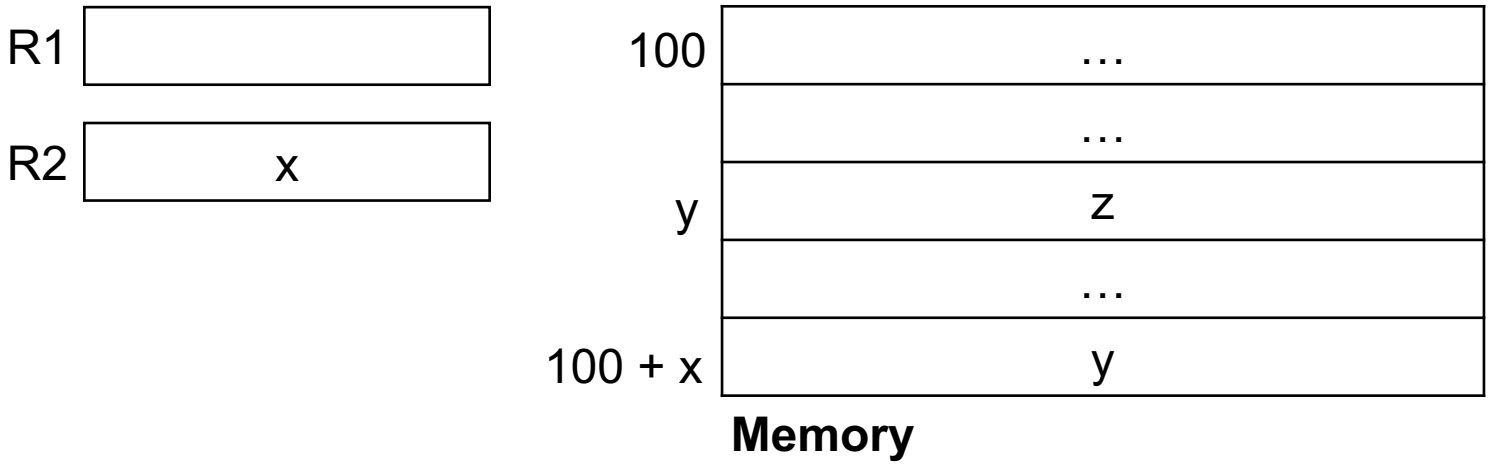
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**



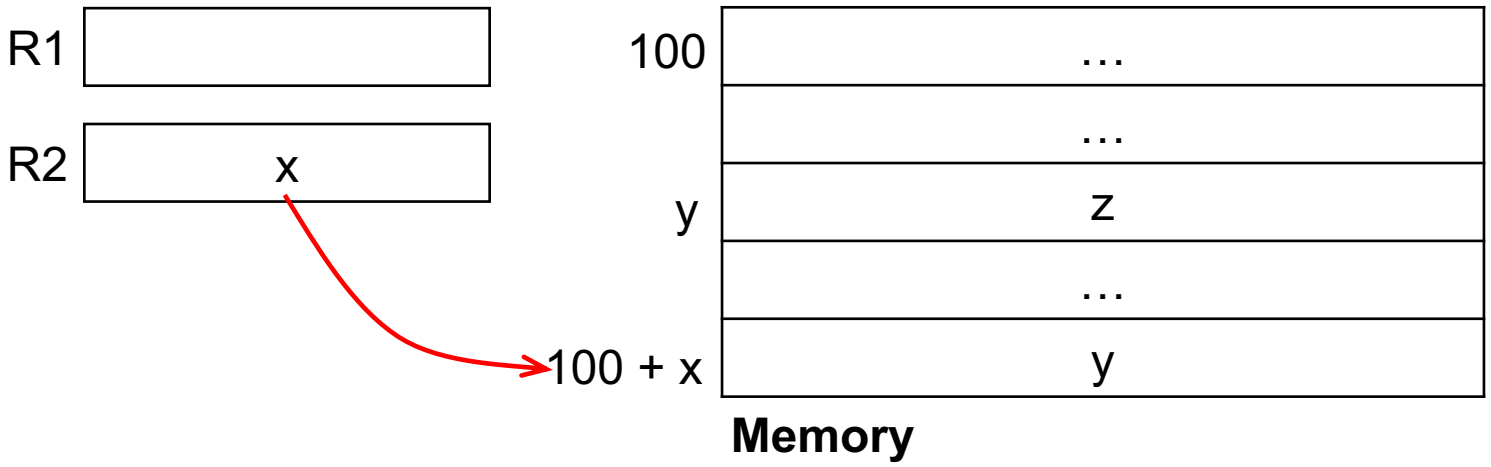
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**



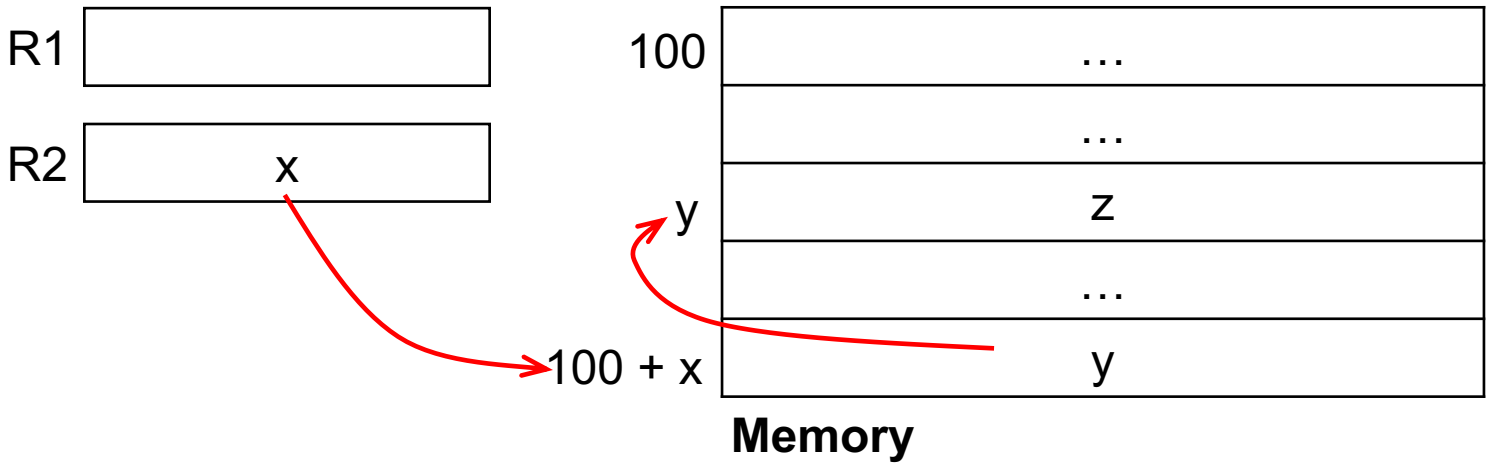
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

x = y - z



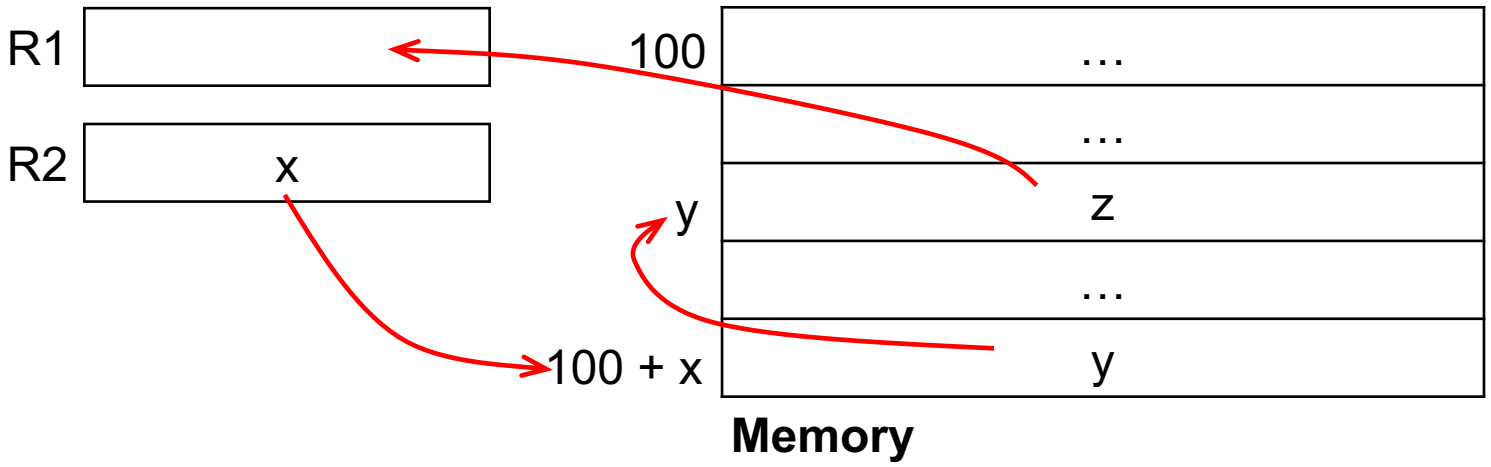
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

x = y - z



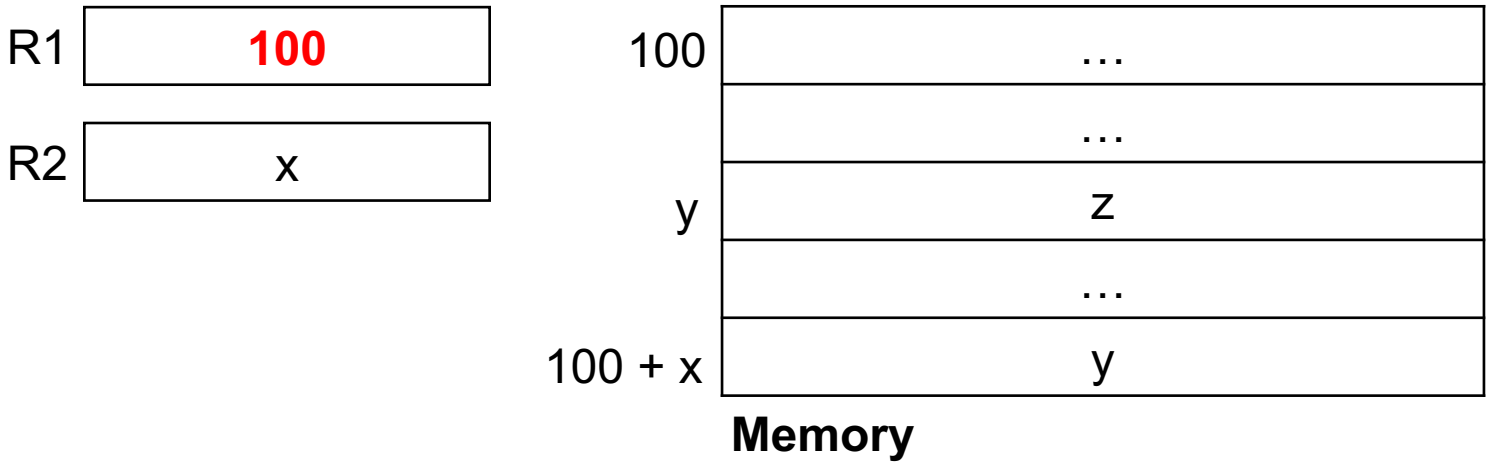
Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**



Target Machine

- Load/Store/Calculation/Jump/…

```
LD  R1, y          // R1 = y
LD  R2, z          // R2 = z
SUB R1, R1, R2     // R1 = R1 - R2
ST  x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

Addressing Modes: How we compute the addr

- LD R, ***addr***
- ST ***addr***, R
- BR ***addr***

Target Machine

- Load/Store/Calculation/Jump/…

```
LD  R1, y           // R1 = y
LD  R2, z           // R2 = z
SUB R1, R1, R2      // R1 = R1 - R2
ST  x, R1           // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

```
LD  R1, i           // R1 = i
MUL R1, R1, 4       // R1 = R1 * 4
LD  R2, a(R1)       // R2 = contents(a + contents(R1))
ST  b, R2           // b = R2
```

b = a[i] (a is an int array)

Target Machine

- Load/Store/Calculation/Jump/…

```
LD  R1, y          // R1 = y
LD  R2, z          // R2 = z
SUB R1, R1, R2     // R1 = R1 - R2
ST  x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

```
LD  R1, i          // R1 = i
MUL R1, R1, 4      // R1 = R1 * 4
LD  R2, a(R1)      // R2 = contents(a + contents(R1))
ST  b, R2          // b = R2
```

b = a[i] (a is an int array)

Target Machine

- Load/Store/Calculation/Jump/…

```
LD  R1, y           // R1 = y
LD  R2, z           // R2 = z
SUB R1, R1, R2      // R1 = R1 - R2
ST  x, R1           // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

```
LD  R1, i           // R1 = i
MUL R1, R1, 4       // R1 = R1 * 4
LD  R2, a(R1)       // R2 = contents(a + contents(R1))
ST  b, R2           // b = R2
```

b = a[i] (a is an int array)

Target Machine

- Load/Store/Calculation/Jump/…

```
LD  R1, y          // R1 = y
LD  R2, z          // R2 = z
SUB R1, R1, R2     // R1 = R1 - R2
ST  x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

```
LD  R1, i          // R1 = i
MUL R1, R1, 4      // R1 = R1 * 4
LD  R2, a(R1)      // R2 = contents(a + contents(R1))
ST  b, R2          // b = R2
```

b = a[i] (a is an int array)

Target Machine

- Load/Store/Calculation/Jump/…

```
LD  R1, y          // R1 = y
LD  R2, z          // R2 = z
SUB R1, R1, R2     // R1 = R1 - R2
ST  x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

```
LD  R1, i          // R1 = i
MUL R1, R1, 4      // R1 = R1 * 4
LD  R2, a(R1)      // R2 = contents(a + contents(R1))
ST  b, R2          // b = R2
```

b = a[i] (a is an int array)

Target Machine

- Load/Store/Calculation/Jump/…

```
LD  R1, y          // R1 = y
LD  R2, z          // R2 = z
SUB R1, R1, R2     // R1 = R1 - R2
ST  x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

```
LD   R1, x         // R1 = x
LD   R2, y         // R2 = y
SUB  R1, R1, R2    // R1 = R1 - R2
BLTZ R1, M         // if R1 < 0 jump to M
```

if x < y goto M

Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

```
LD   R1, x          // R1 = x
LD   R2, y          // R2 = y
SUB  R1, R1, R2     // R1 = R1 - R2
BLTZ R1, M          // if R1 < 0 jump to M
```

if x < y goto M

Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

```
LD   R1, x          // R1 = x
LD   R2, y          // R2 = y
SUB  R1, R1, R2     // R1 = R1 - R2
BLTZ R1, M          // if R1 < 0 jump to M
```

if x < y goto M

Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

```
LD   R1, x          // R1 = x
LD   R2, y          // R2 = y
SUB  R1, R1, R2     // R1 = R1 - R2
BLTZ R1, M          // if R1 < 0 jump to M
```

if x < y goto M

Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

```
LD   R1, x          // R1 = x
LD   R2, y          // R2 = y
SUB  R1, R1, R2     // R1 = R1 - R2
BLTZ R1, M          // if R1 < 0 jump to M
```

if x < y goto M

Target Machine

- Load/Store/Calculation/Jump/…

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

- Addressing Modes
  - LD R1, **a(R2)**
  - LD R1, **100(R2)**
  - LD R1, ***R2**
  - LD R1, ***100(R2)**
  - LD R1, **#100**

```
y = *q
q = q + 4
*p = y
p = p + 4
```

Try!

Address Representation

Q: How does a machine understand x, y, z, and M?

A: Each variable or label corresponds to a memory address

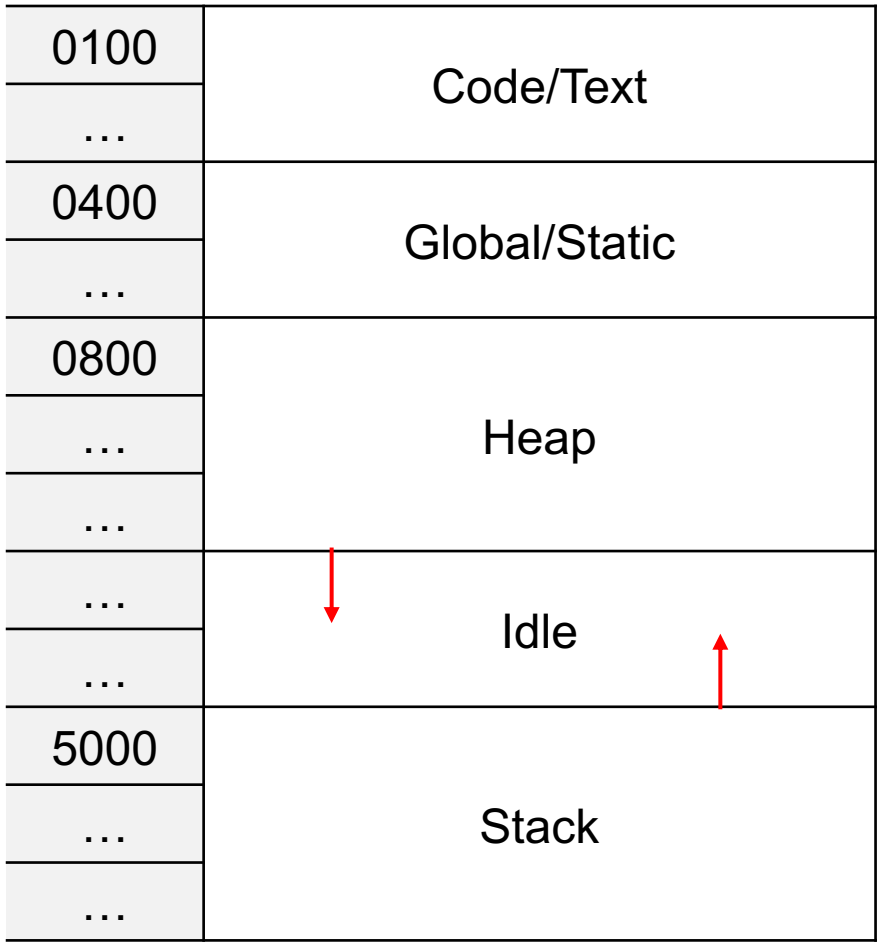
```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

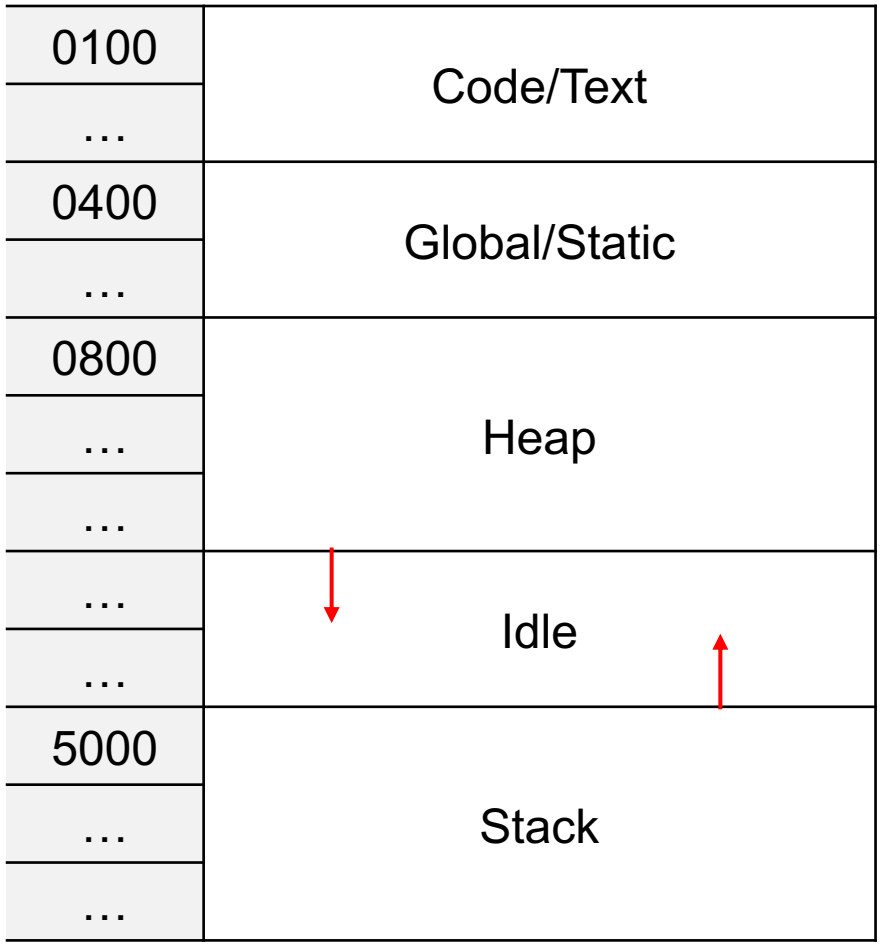
```
LD    R1, x         // R1 = x
LD    R2, y         // R2 = y
SUB   R1, R1, R2    // R1 = R1 - R2
BLTZ  R1, M         // if R1 < 0 jump to M
```

if x < y goto M

Memory Structure



Memory Structure



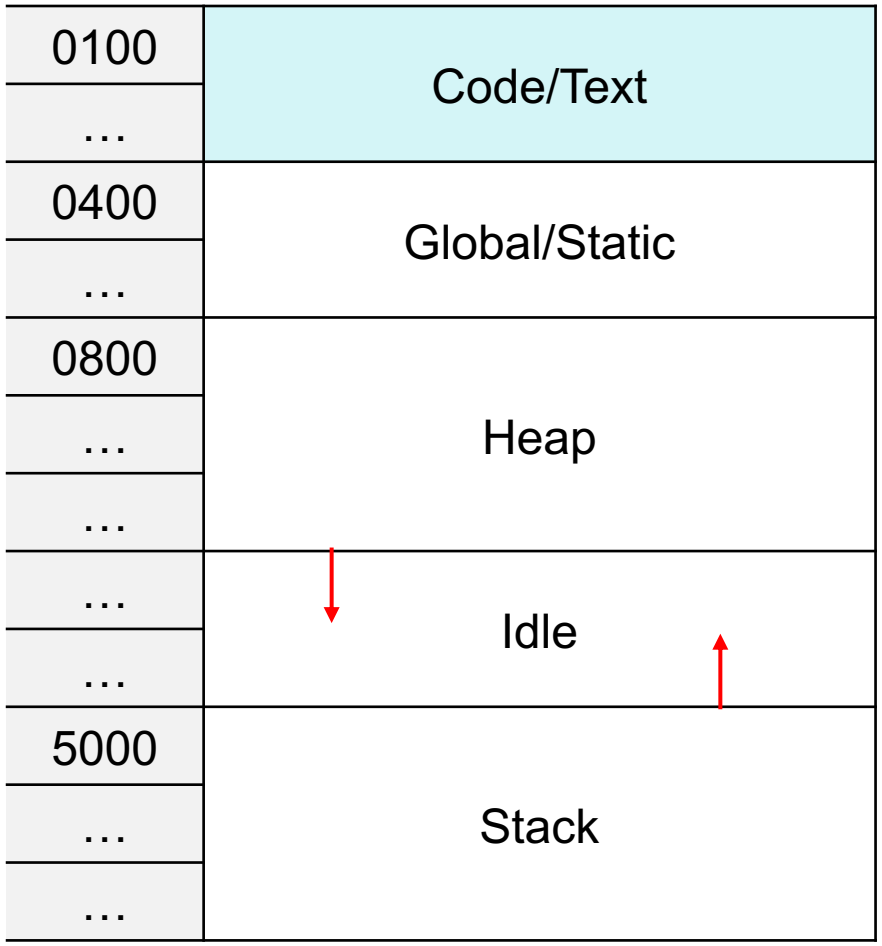

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

```
LD   R1, x          // R1 = x
LD   R2, y          // R2 = y
SUB  R1, R1, R2     // R1 = R1 - R2
BLTZ R1, M          // if R1 < 0 jump to M
```

if x < y goto M

Text Memory




```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

```
LD   R1, x          // R1 = x
LD   R2, y          // R2 = y
SUB  R1, R1, R2     // R1 = R1 - R2
BLTZ R1, M          // if R1 < 0 jump to M
```

if x < y goto M

Text Memory

0100	LD R1 x
0104	LD R2 y
0108	SUB R1 R1 R2
0112	BLTZ R1 **#0124**
0116	…
0120	…
0124	…
…	…
	Static
	Heap
	Idle
	Stack

Load to memory

```
LD   R1, x          // R1 = x
LD   R2, y          // R2 = y
SUB  R1, R1, R2     // R1 = R1 - R2
BLTZ R1, M          // if R1 < 0 jump to M
```

if x < y goto M

Text Memory

0100	LD R1 x
0104	LD R2 y
0108	SUB R1 R1 R2
0112	BLTZ R1 **#0124**
0116	…
0120	…
0124	…
…	…
	Static
	Heap
	Idle
	Stack

M is translated to a memory address

```
LD   R1, x          // R1 = x
LD   R2, y          // R2 = y
SUB  R1, R1, R2     // R1 = R1 - R2
BLTZ R1, M          // if R1 < 0 jump to M
```

if x < y goto M

Text Memory

0100	LD R1 x
0104	LD R2 y
0108	SUB R1 R1 R2
0112	BLTZ R1 **#0124**
0116	…
0120	…
0124	…
…	…
	Static
	Heap
	Idle
	Stack

jump

M is translated to a memory address

```
LD   R1, x          // R1 = x
LD   R2, y          // R2 = y
SUB  R1, R1, R2     // R1 = R1 - R2
BLTZ R1, M          // if R1 < 0 jump to M
```

if x < y goto M

Jump in X86

```
int x = 2;
int y = 4;

int main (int argc, char** argv) {
      if (argc > 0) {  return  x; }
      else return y;
}
```

- clang -c hello.c –o hello.o
- objdump –D hello.o

Jump

```
int x = 2;
int y = 4;

int main (int
      if (arg
      else re
}
```

- clang -c hello.
- objdump –D h

```
Disassembly of section __TEXT,__text:

0000000000000000 <_main>:
       0: 55                            pushq   %rbp
       1: 48 89 e5                      movq    %rsp, %rbp
       4: c7 45 fc 00 00 00 00          movl    $0, -4(%rbp)
       b: 89 7d f8                      movl    %edi, -8(%rbp)
       e: 48 89 75 f0                   movq    %rsi, -16(%rbp)
      12: 83 7d f8 00                   cmpl    $0, -8(%rbp)
      16: 0f 8e 0e 00 00 00             jle     0x2a <_main+0x2a>
      1c: 8b 05 00 00 00 00             movl    (%rip), %eax
      22: 89 45 fc                      movl    %eax, -4(%rbp)
      25: e9 09 00 00 00                jmp     0x33 <_main+0x33>
      2a: 8b 05 00 00 00 00             movl    (%rip), %eax
      30: 89 45 fc                      movl    %eax, -4(%rbp)
      33: 8b 45 fc                      movl    -4(%rbp), %eax
      36: 5d                            popq    %rbp
      37: c3                            retq
```

Jump

```
int x = 2;
int y = 4;

int main (int
      if (arg
      else re
}
```

- clang -c hello.
- objdump –D h

```
Disassembly of section __TEXT,__text:

0000000000000000 <_main>:
       0: 55                                pushq   %rbp
       1: 48 89 e5                          movq    %rsp, %rbp
       4: c7 45 fc 00 00 00 00              movl    $0, -4(%rbp)
       b: 89 7d f8                          movl    %edi, -8(%rbp)
       e: 48 89 75 f0                       movq    %rsi, -16(%rbp)
      12: 83 7d f8 00                       cmpl    $0, -8(%rbp)
      16: 0f 8e 0e 00 00 00                 jle     0x2a <_main+0x2a>
      1c: 8b 05 00 00 00 00                 movl    (%rip), %eax
      22: 89 45 fc                          movl    %eax, -4(%rbp)
      25: e9 09 00 00 00                    jmp     0x33 <_main+0x33>
      2a: 8b 05 00 00 00 00                 movl    (%rip), %eax
      30: 89 45 fc                          movl    %eax, -4(%rbp)
      33: 8b 45 fc                          movl    -4(%rbp), %eax
      36: 5d                                popq    %rbp
      37: c3                                retq
```

Jump

```
int x = 2;
int y = 4;

int main (int
      if (arg
      else re
}
```

- clang -c hello.
- objdump –D h

```
Disassembly of section __TEXT,__text:

0000000000000000 <_main>:
       0: 55                    pushq   %rbp
       1: 48 89 e5              movq    %rsp, %rbp
       4: c7 45 fc 00 00 00 00  movl    $0, -4(%rbp)
       b: 89 7d f8              movl    %edi, -8(%rbp)
       e: 48 89 75 f0           movq    %rsi, -16(%rbp)
      12: 83 7d f8 00           cmpl    $0, -8(%rbp)
      16: 0f 8e 0e 00 00 00     jle     0x2a <_main+0x2a>
      1c: 8b 05 00 00 00 00     movl    (%rip), %eax
      22: 89 45 fc              movl    %eax, -4(%rbp)
      25: e9 09 00 00 00        jmp     0x33 <_main+0x33>
      2a: 8b 05 00 00 00 00     movl    (%rip), %eax
      30: 89 45 fc              movl    %eax, -4(%rbp)
      33: 8b 45 fc              movl    -4(%rbp), %eax
      36: 5d                    popq    %rbp
      37: c3                    retq
```

Jump

```
int x = 2;
int y = 4;

int main (int
      if (arg
      else re
}
```

- clang -c hello.
- objdump –D h

```
Disassembly of section __TEXT,__text:

0000000000000000 <_main>:
       0: 55                    pushq   %rbp
       1: 48 89 e5              movq    %rsp, %rbp
       4: c7 45 fc 00 00 00 00  movl    $0, -4(%rbp)
       b: 89 7d f8              movl    %edi, -8(%rbp)
       e: 48 89 75 f0           movq    %rsi, -16(%rbp)
      12: 83 7d f8 00           cmpl    $0, -8(%rbp)
      16: 0f 8e 0e 00 00 00     jle     0x2a <_main+0x2a>
      1c: 8b 05 00 00 00 00     movl    (%rip), %eax
      22: 89 45 fc              movl    %eax, -4(%rbp)
      25: e9 09 00 00 00        jmp     0x33 <_main+0x33>
      2a: 8b 05 00 00 00 00     movl    (%rip), %eax
      30: 89 45 fc              movl    %eax, -4(%rbp)
      33: 8b 45 fc              movl    -4(%rbp), %eax
      36: 5d                    popq    %rbp
      37: c3                    retq
```

Jump

```
int x = 2;
int y = 4;

int main (int
      if (arg
      else re
}
```

- clang -c hello.
- objdump –D h

```
Disassembly of section __TEXT,__text:

0000000000000000 <_main>:
       0: 55                           pushq   %rbp
       1: 48 89 e5                     movq    %rsp, %rbp
       4: c7 45 fc 00 00 00 00         movl    $0, -4(%rbp)
       b: 89 7d f8                     movl    %edi, -8(%rbp)
       e: 48 89 75 f0                  movq    %rsi, -16(%rbp)
      12: 83 7d f8 00                  cmpl    $0, -8(%rbp)
      16: 0f 8e 0e 00 00 00            jle     0x2a <_main+0x2a>
      1c: 8b 05 00 00 00 00            movl    (%rip), %eax
      22: 89 45 fc                     movl    %eax, -4(%rbp)
      25: e9 09 00 00 00               jmp     0x33 <_main+0x33>
      2a: 8b 05 00 00 00 00            movl    (%rip), %eax
      30: 89 45 fc                     movl    %eax, -4(%rbp)
      33: 8b 45 fc                     movl    -4(%rbp), %eax
      36: 5d                           popq    %rbp
      37: c3                           retq
```

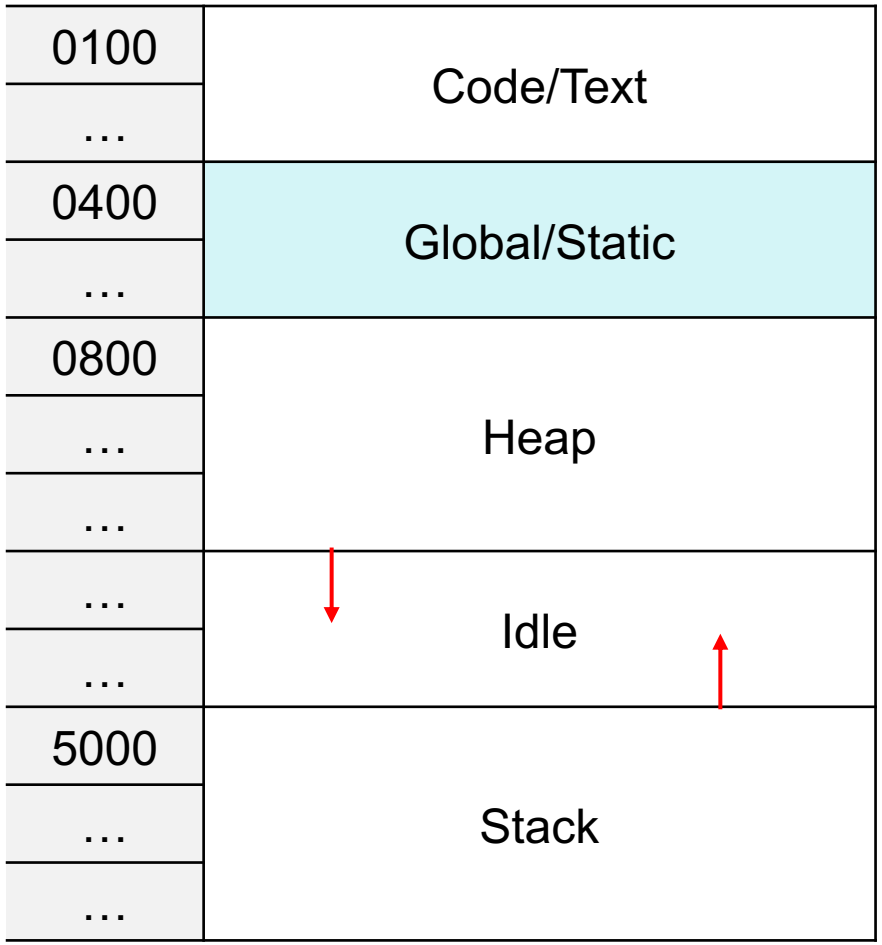
Jump

```
int x = 2;
int y = 4;

int main (int
      if (arg
      else re
}
```

- clang -c hello.
- objdump –D h

```
Disassembly of section __TEXT,__text:

0000000000000000 <_main>:
       0: 55                                pushq   %rbp
       1: 48 89 e5                          movq    %rsp, %rbp
       4: c7 45 fc 00 00 00 00              movl    $0, -4(%rbp)
       b: 89 7d f8                          movl    %edi, -8(%rbp)
       e: 48 89 75 f0                       movq    %rsi, -16(%rbp)
      12: 83 7d f8 00                       cmpl    $0, -8(%rbp)
      16: 0f 8e 0e 00 00 00                 jle     0x2a <_main+0x2a>
      1c: 8b 05 00 00 00 00                 movl    (%rip), %eax
      22: 89 45 fc                          movl    %eax, -4(%rbp)
      25: e9 09 00 00 00                    jmp     0x33 <_main+0x33>
      2a: 8b 05 00 00 00 00                 movl    (%rip), %eax
      30: 89 45 fc                          movl    %eax, -4(%rbp)
      33: 8b 45 fc                          movl    -4(%rbp), %eax
      36: 5d                                popq    %rbp
      37: c3                                retq
```

Global Memory



Global Memory

0100	…	Text
0104	…	
0108	…	
0112	…	
…	…	
0400	2	Global
0404	3	
0408	…	
…	…	
…	Heap	
…	Idle	
…	Stack	

```
int y = 2;
int z = 3;

void foo ()  {
    int x;
    x = y - z;
}
```

Global Memory

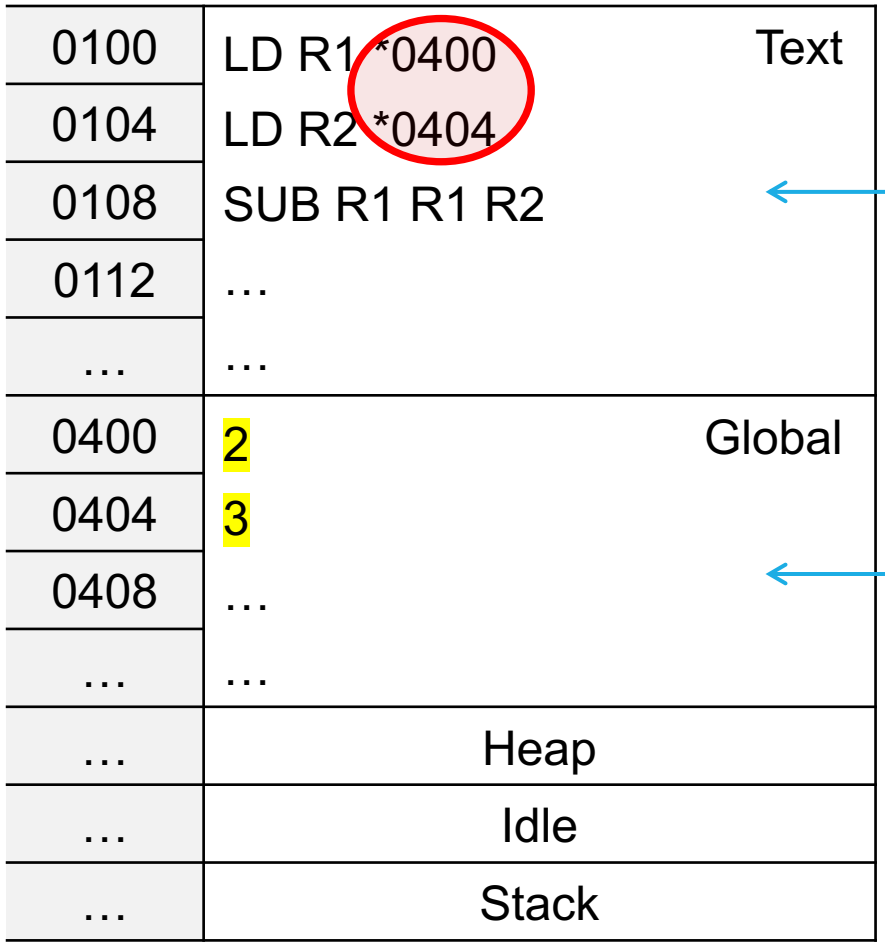
0100	LD R1 *0400	Text
0104	LD R2 *0404	
0108	SUB R1 R1 R2	
0112	…	
…	…	
0400	2	Global
0404	3	
0408	…	
…	…	
…	Heap	
…	Idle	
…	Stack	

```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

```
int y = 2;
int z = 3;

void foo ()  {
    int x;
    x = y - z;
}
```

Global Memory

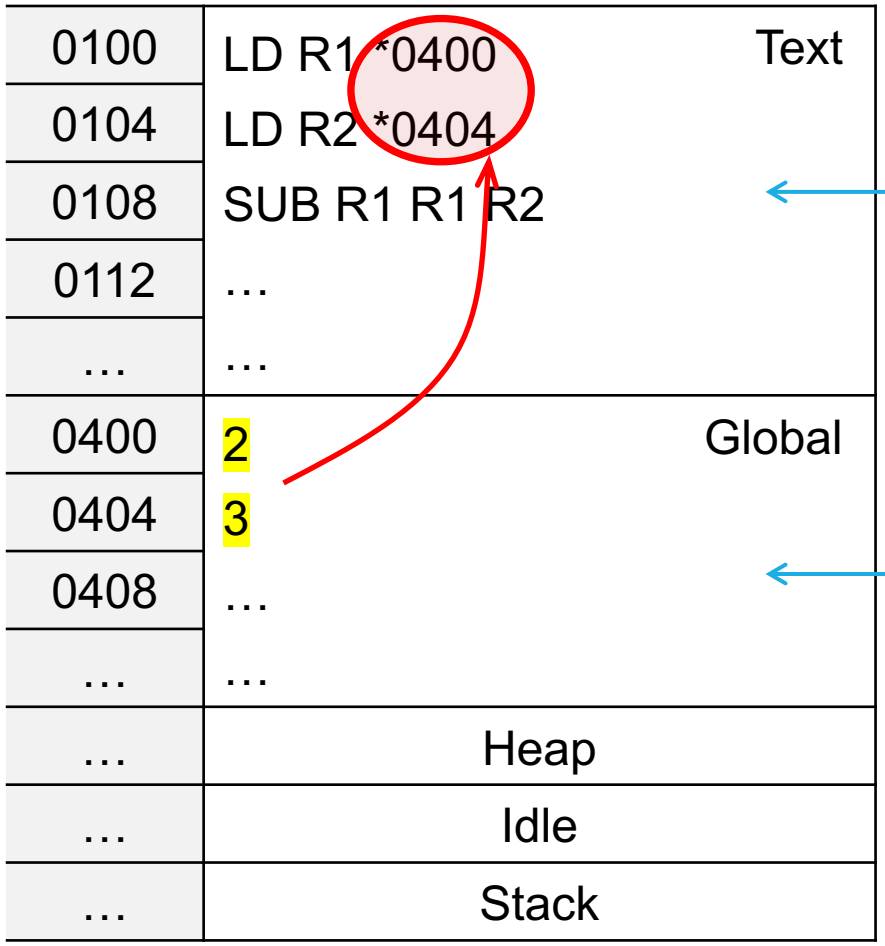



```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

```
int y = 2;
int z = 3;

void foo ()  {
    int x;
    x = y - z;
}
```

Global Memory




```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

```
int y = 2;
int z = 3;

void foo ()  {
    int x;
    x = y - z;
}
```

Globals in x86

```
int x = 2;
int y = 4;

int main (int argc, char** argv) {
      if (argc > 0) {  return  x; }
      else return y;
}
```

- clang -c hello.c –o hello.o
- objdump –D hello.o

Globals in x86

```
int x = 2;
int y = 4;

int main (int argc, char** argv) {
      if (argc > 0) {  return  x; }
      else return y;
}
```

- clang -c hello.c –o hello.o
- objdump –D hello.o

```
Disassembly of section __DATA,__data:

0000000000000038 <_x>:
      38: 02 00
      3a: 00 00

000000000000003c <_y>:
      3c: 04 00
      3e: 00 00
```

Globals in x86

```
int x = 2;
int y = 4;

int main (int argc, char** argv) {
      if (argc > 0) {  return  x; }
      else return y;
}
```

- clang -c hello.c –o hello.o
- objdump –D hello.o

```
Disassembly of section __DATA,__data:

0000000000000038 <_x>:
      38: 02 00
      3a: 00 00

000000000000003c <_y>:
      3c: 04 00
      3e: 00 00
```

Globals in x86

```
int x = 2;
int y = 4;

int main (int argc, char** argv) {
      if (argc > 0) {  return  x; }
      else return y;
}
```

- clang -c hello.c –o hello.o
- objdump –D hello.o

```
Disassembly of section __DATA,__data:

0000000000000038 <_x>:
      38: 02 00
      3a: 00 00

000000000000003c <_y>:
      3c: 04 00
      3e: 00 00
```

Take Away Message

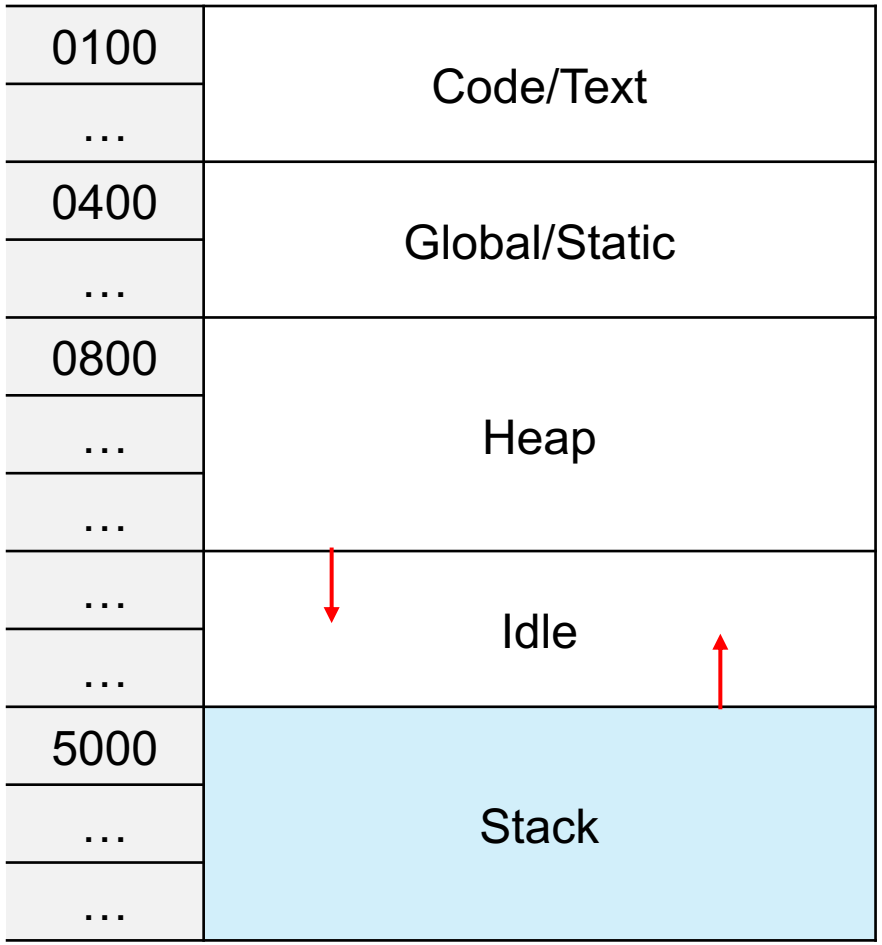
The memory locations of **the code** and **the globals** are determined by the compiler!

Take Away Message

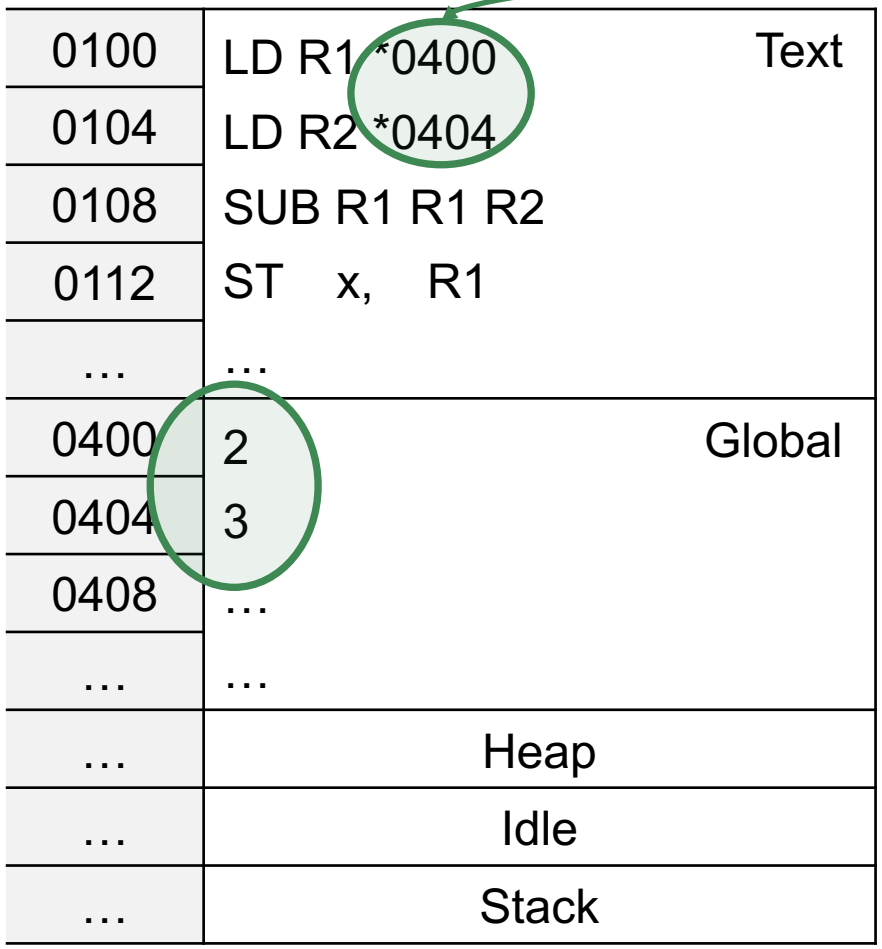
The memory locations of **the code** and **the globals** are determined by the compiler!

The memory locations of other variables, *e.g.*, **the locals**, are controlled by the code generated by the compiler and stored in heap or stack!

Stack Memory



Recap: Global Memory




```
LD   R1, y          // R1 = y
LD   R2, z          // R2 = z
SUB  R1, R1, R2     // R1 = R1 - R2
ST   x, R1          // x = R1
```

x = y - z

```
int y = 2;
int z = 3;
```

```
void foo ()  {
    int x;
    x = y - z;
}
```

Stack Memory

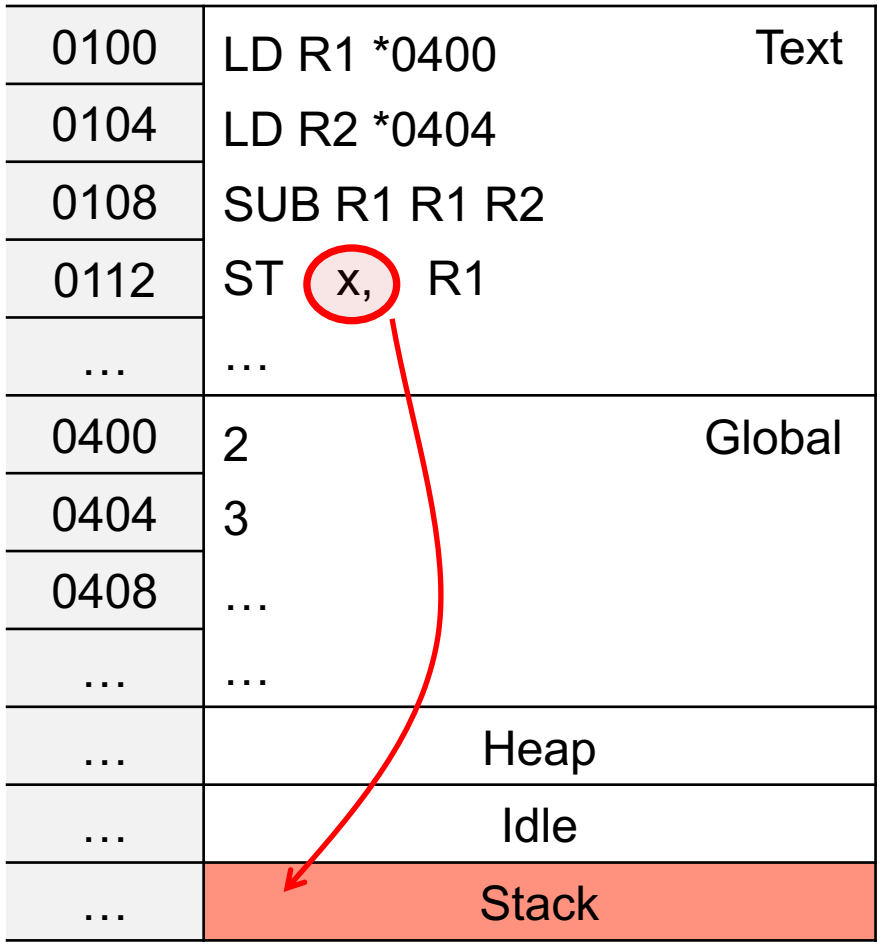
0100	LD R1 *0400	Text
0104	LD R2 *0404	
0108	SUB R1 R1 R2	
0112	ST x, R1	
…	…	
0400	2	Global
0404	3	
0408	…	
…	…	
…	Heap	
…	Idle	
…	Stack	

```
LD   R1, y            // R1 = y
LD   R2, z            // R2 = z
SUB  R1, R1, R2       // R1 = R1 - R2
ST   x, R1            // x = R1
```

x = y - z

```
int y = 2;
int z = 3;

void foo ()  {
    int x;
    x = y - z;
}
```

Stack Memory

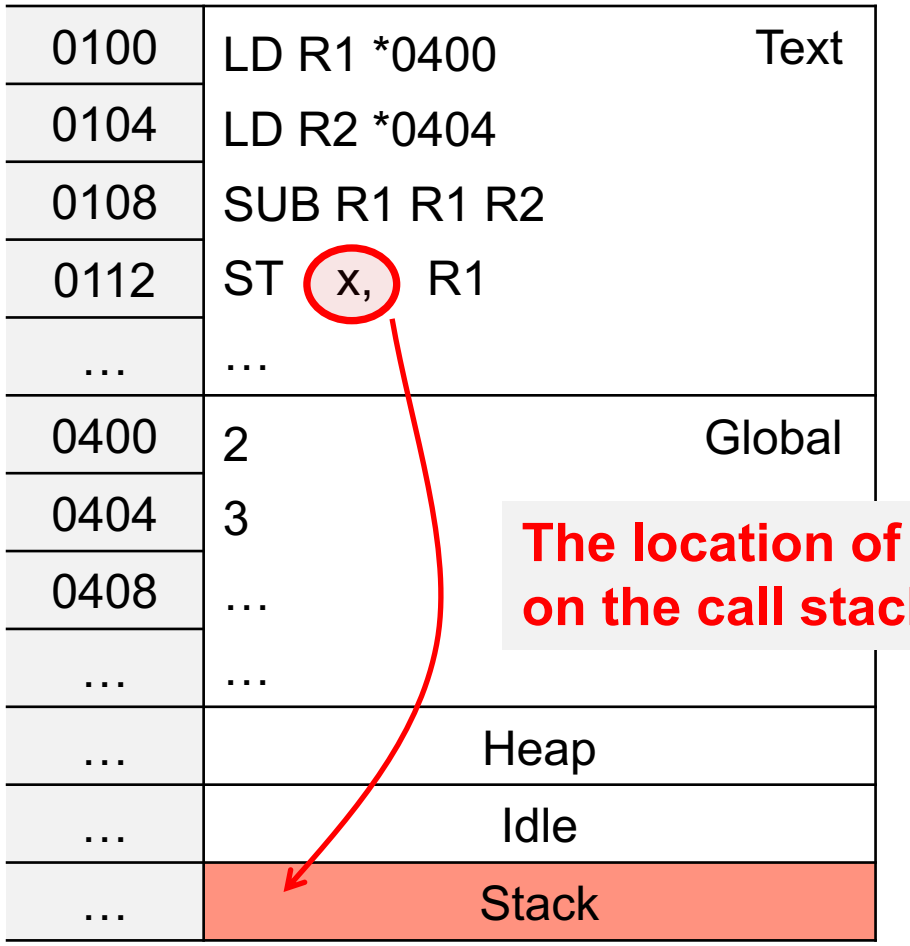



```
LD   R1, y            // R1 = y
LD   R2, z            // R2 = z
SUB  R1, R1, R2       // R1 = R1 - R2
ST   x, R1            // x = R1
```

x = y - z

```
int y = 2;
int z = 3;

void foo ()  {
    int x;
    x = y - z;
}
```

Stack Memory




```
LD   R1, y            // R1 = y
LD   R2, z            // R2 = z
SUB  R1, R1, R2       // R1 = R1 - R2
ST   x, R1            // x = R1
```

x = y - z

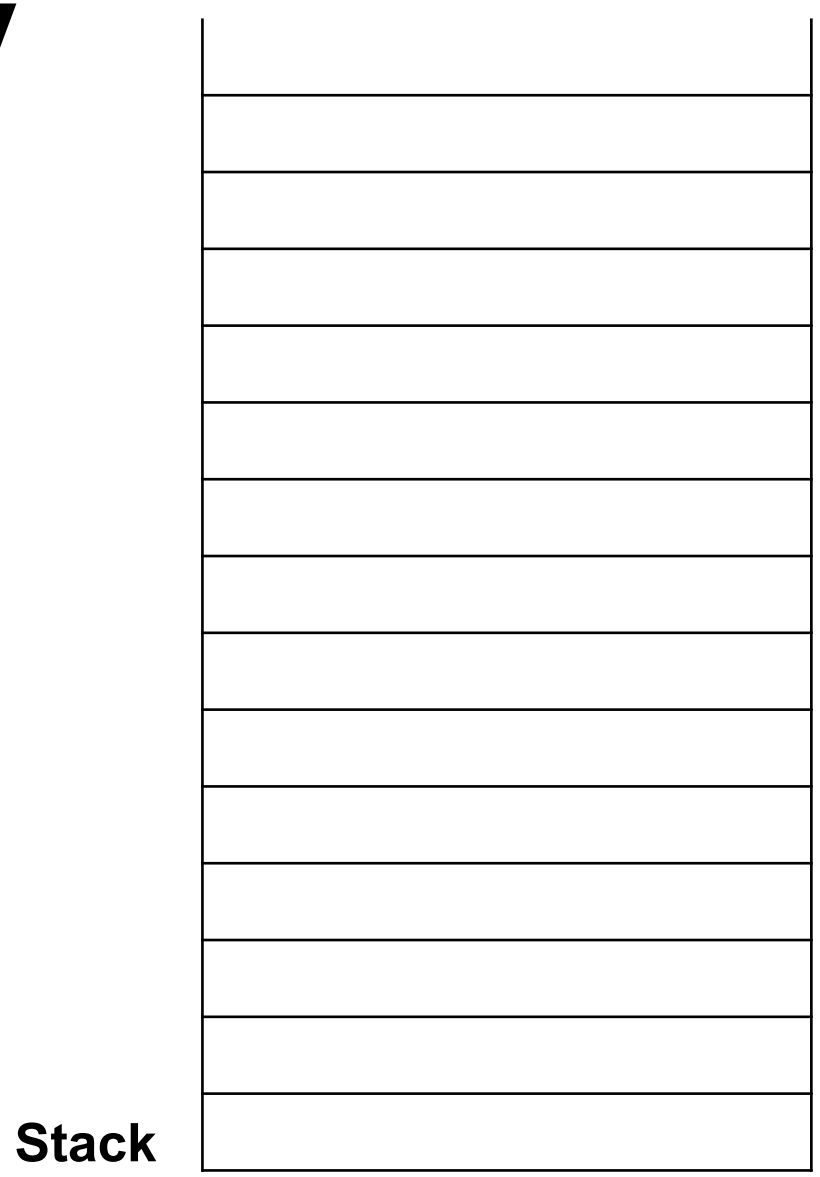
The location of x depends on the call stack!

```
= 2;
= 3;

void foo ()  {
    int x;
    x = y - z;
}
```

Stack Memory

- Call Stack

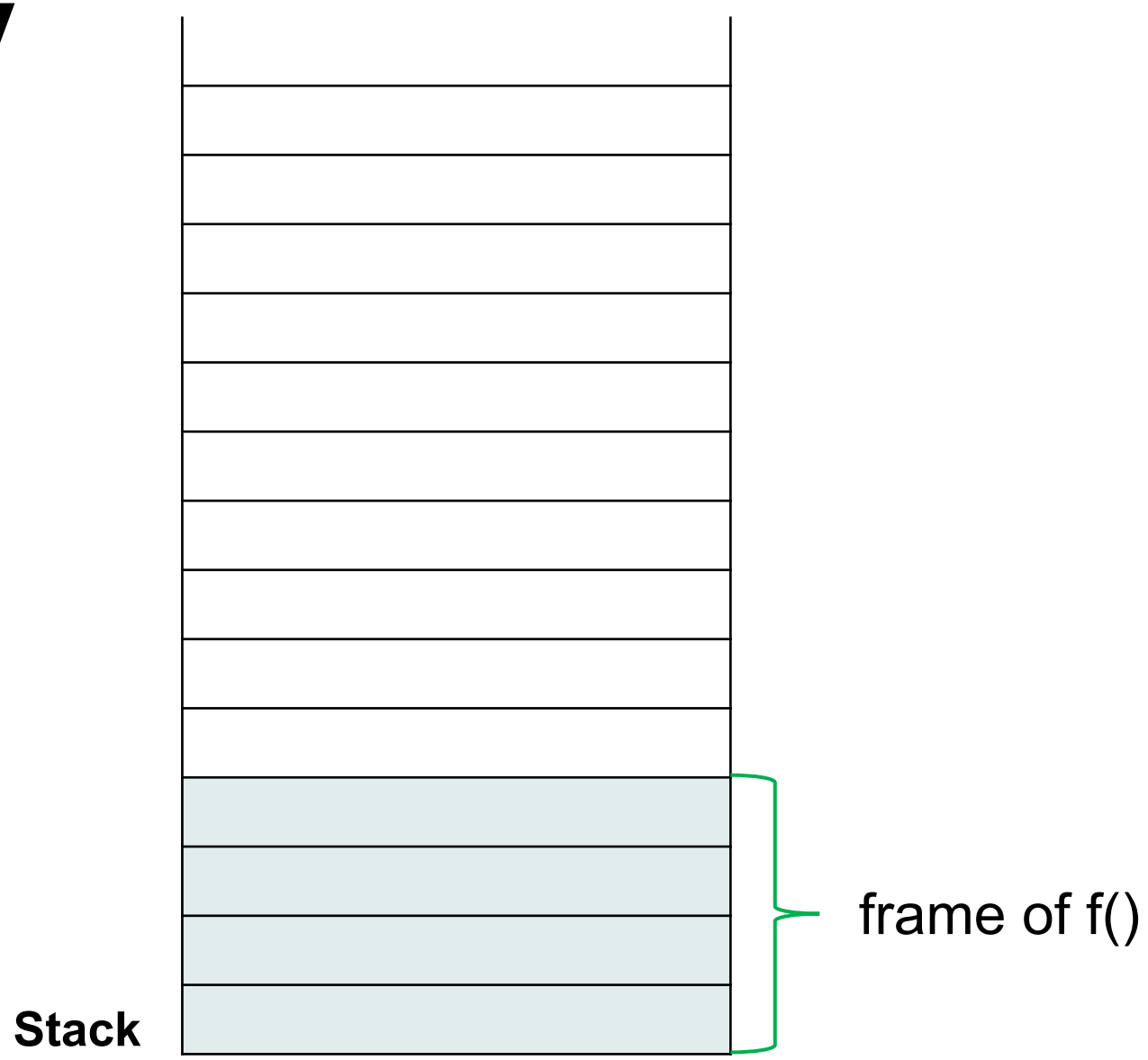
```
1.  void f() {
2.      …
3.      g();
4.      …
5.  }

6.  void g() {
7.      …
8.      h();
9.      …
10. }
```



Stack Memory

- Call Stack

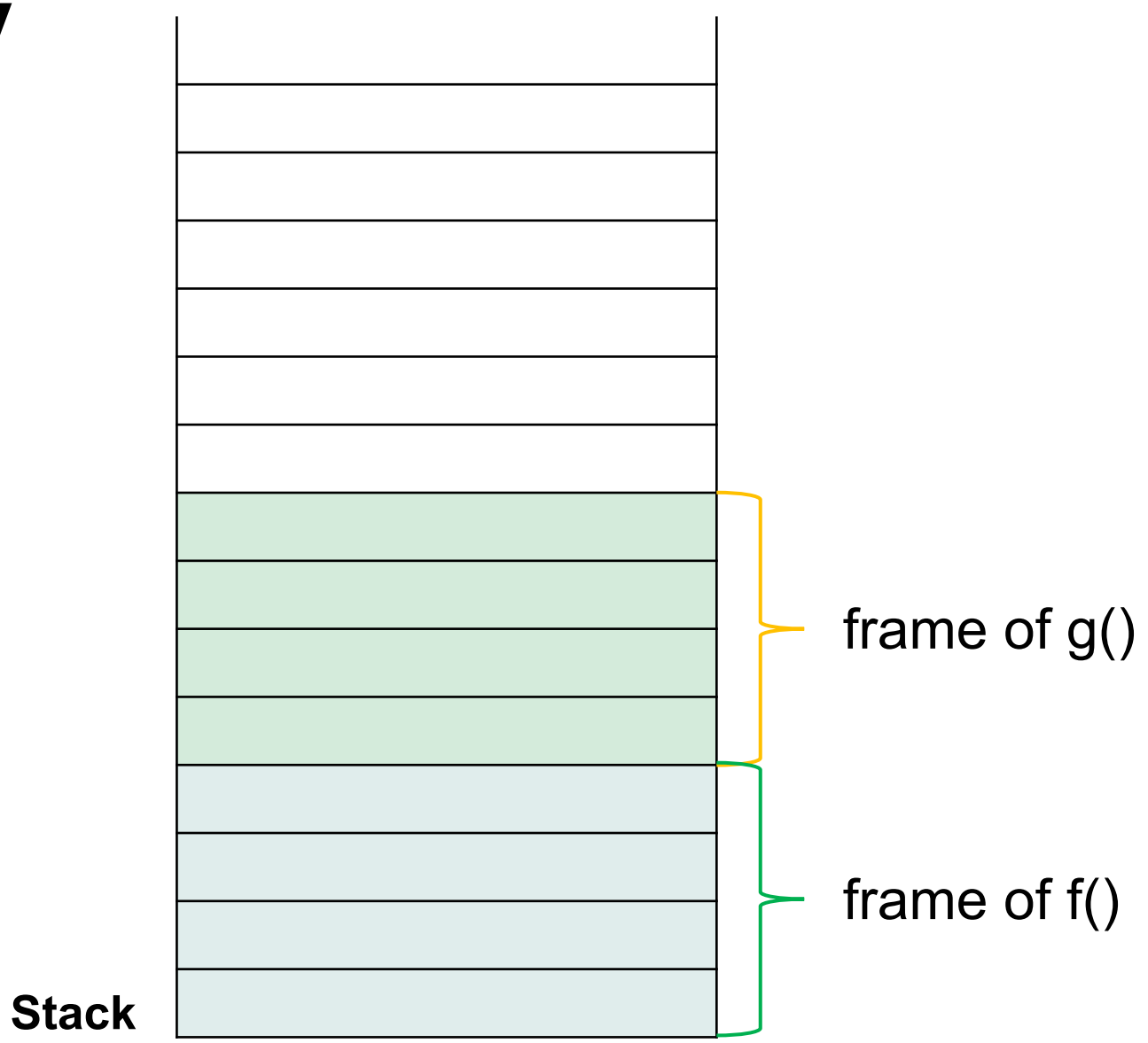
```
1.  void f() {
2.      …
3.      g();
4.      …
5.  }

6.  void g() {
7.      …
8.      h();
9.      …
10. }
```



Stack Memory

- Call Stack

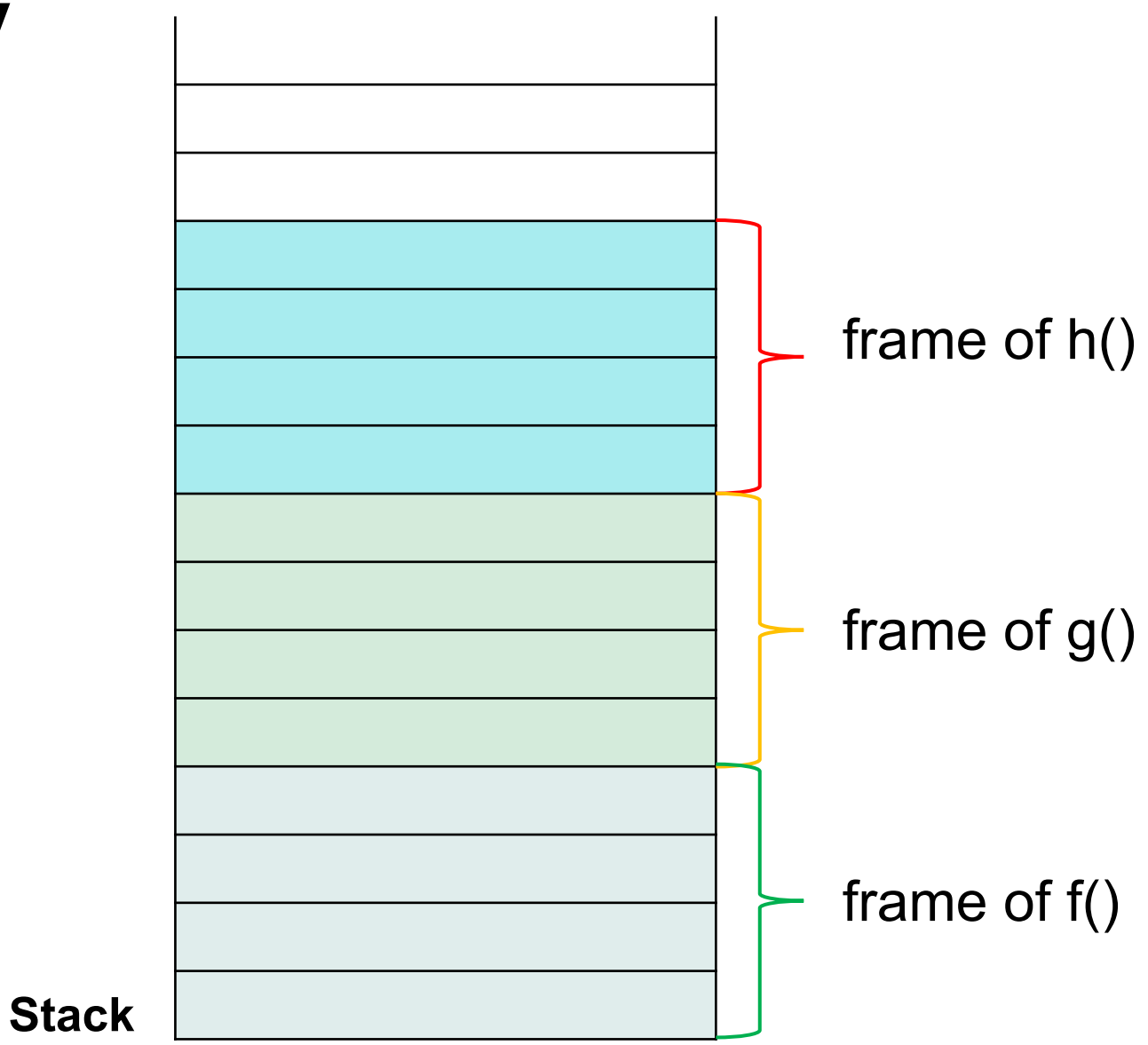
```
1.  void f() {
2.      …
3.      g();
4.      …
5.  }

6.  void g() {
7.      …
8.      h();
9.      …
10. }
```



Stack Memory

- Call Stack

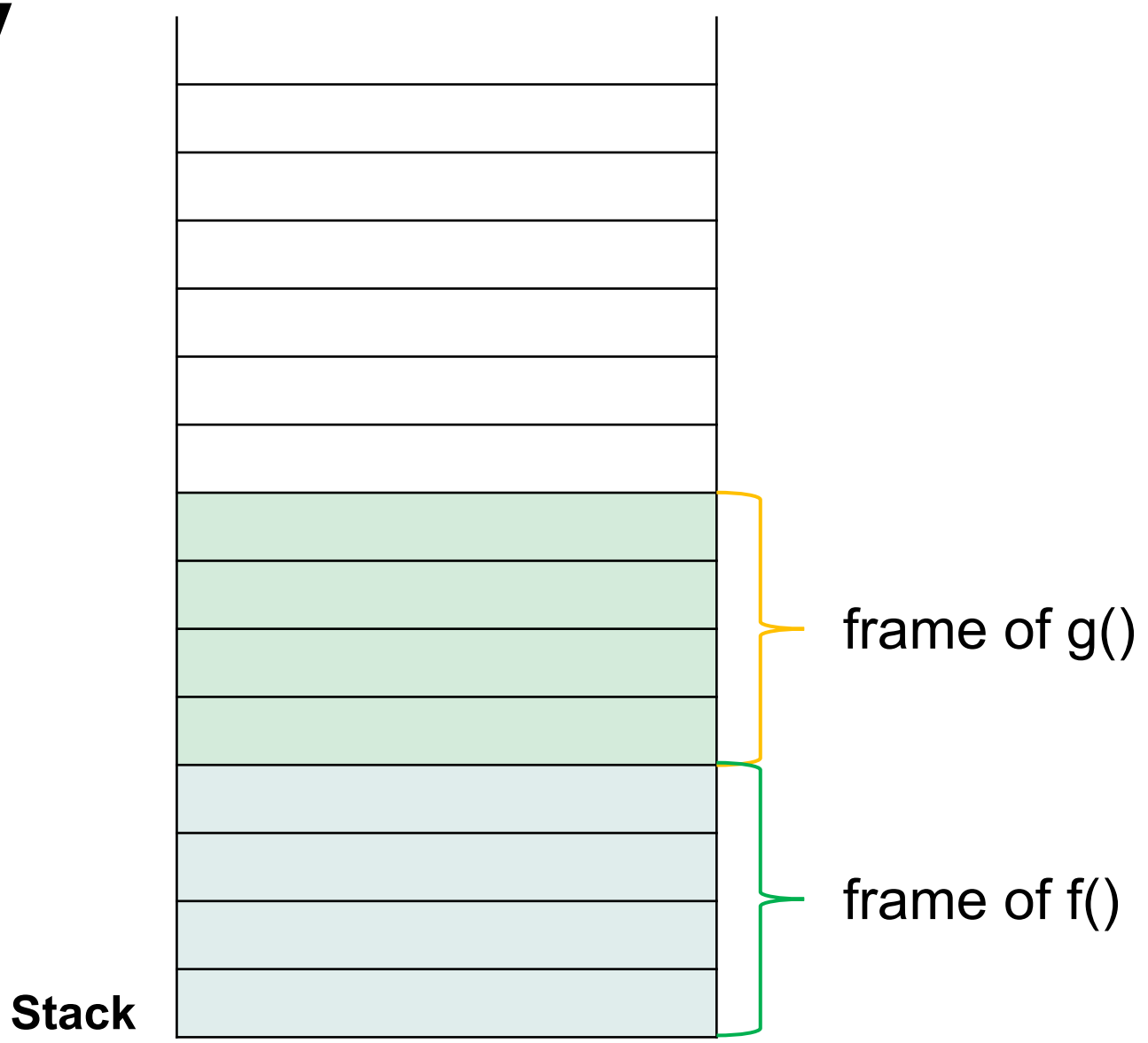
```
1.  void f() {
2.      …
3.      g();
4.      …
5.  }

6.  void g() {
7.      …
8.      h();
9.      …
10. }
```



Stack Memory

- Call Stack

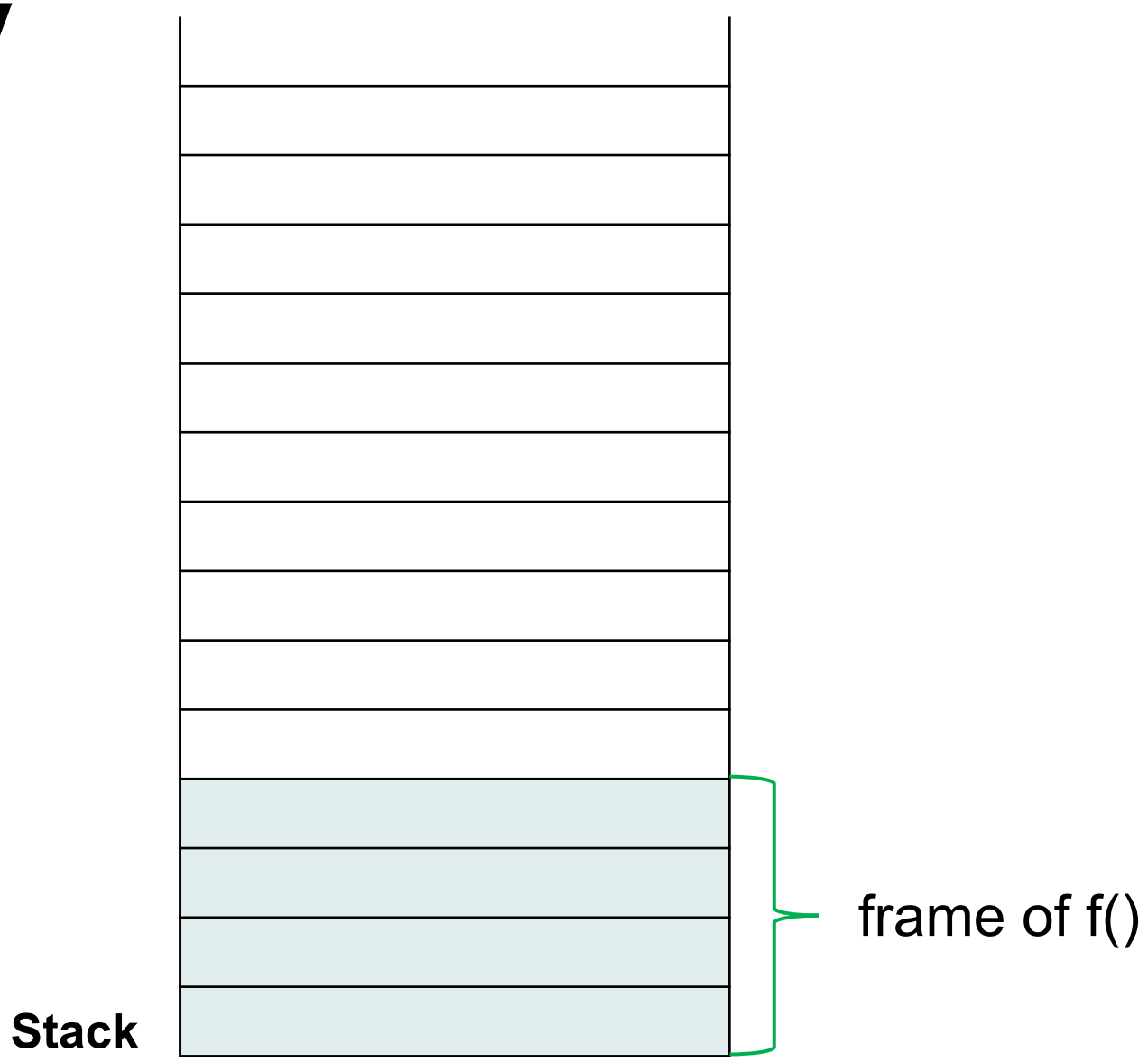
```
1.  void f() {
2.      …
3.      g();
4.      …
5.  }

6.  void g() {
7.      …
8.      h();
9.      …
10. }
```



Stack Memory

- Call Stack

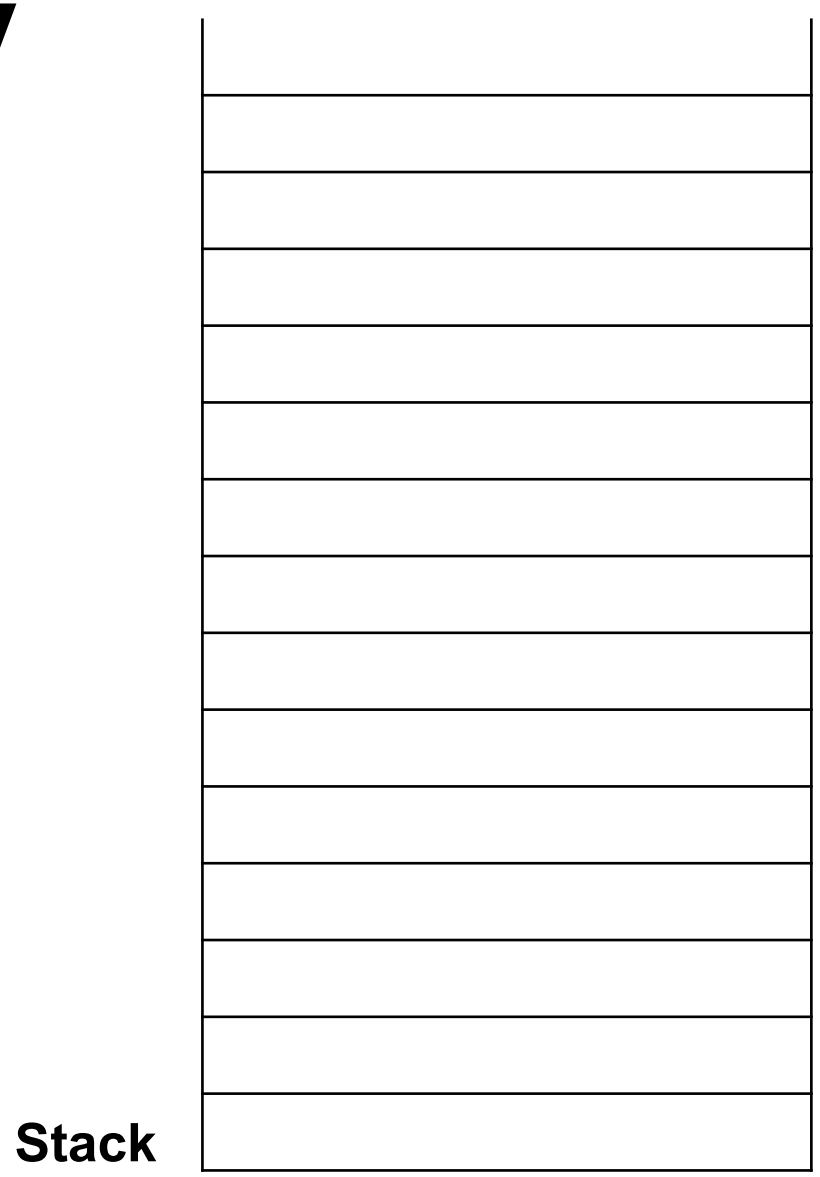
```
1.  void f() {
2.      …
3.      g();
4.      …
5.  }

6.  void g() {
7.      …
8.      h();
9.      …
10. }
```



Stack Memory

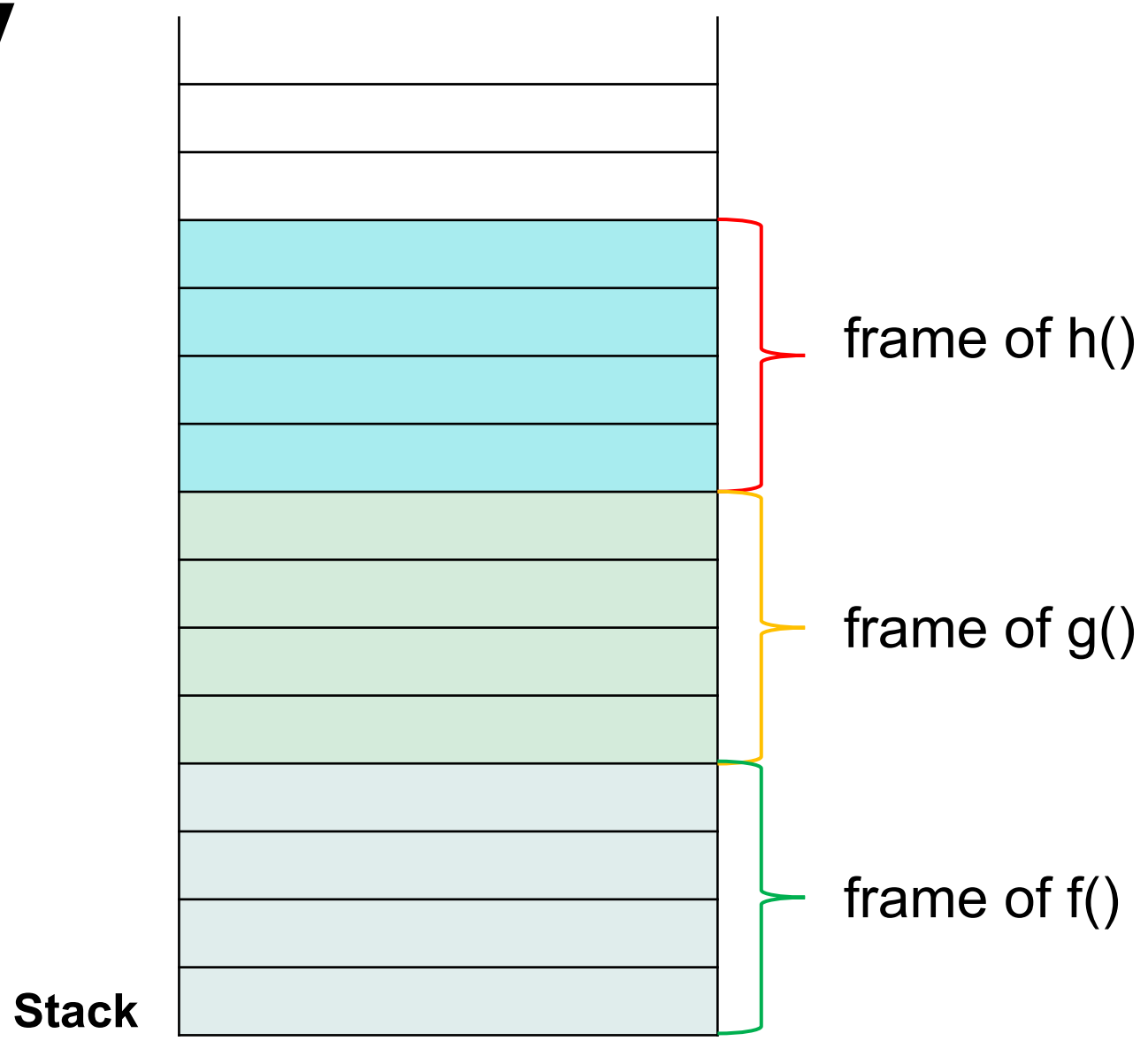
- Call Stack

```
1.  void f() {
2.      …
3.      g();
4.      …
5.  }

6.  void g() {
7.      …
8.      h();
9.      …
10. }
```



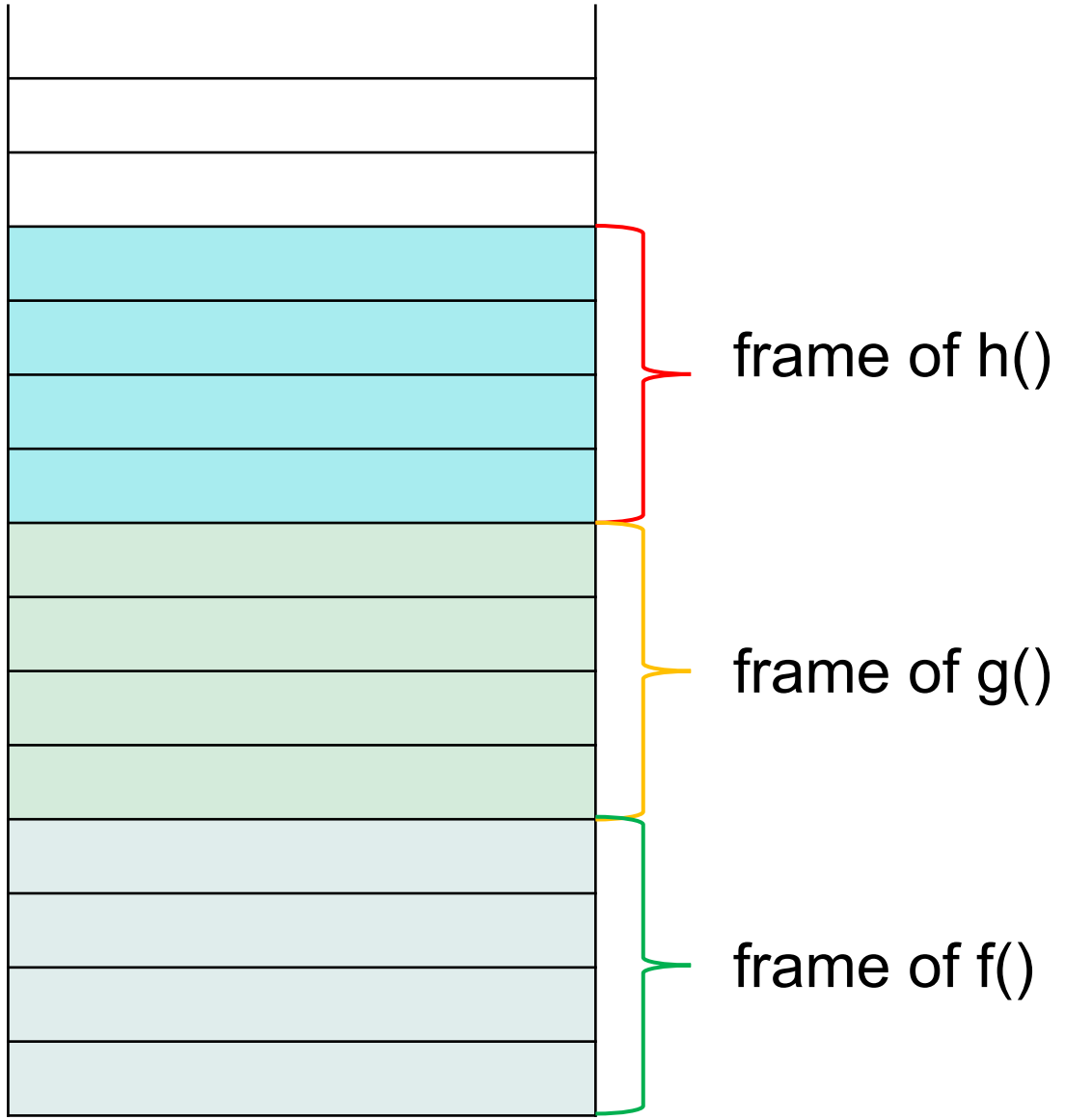
Stack Memory

- Call Stack
- What is in a Frame?
  - Local variables
  - Return address
  - Return value, Parameters
  - Data to recover
  - …



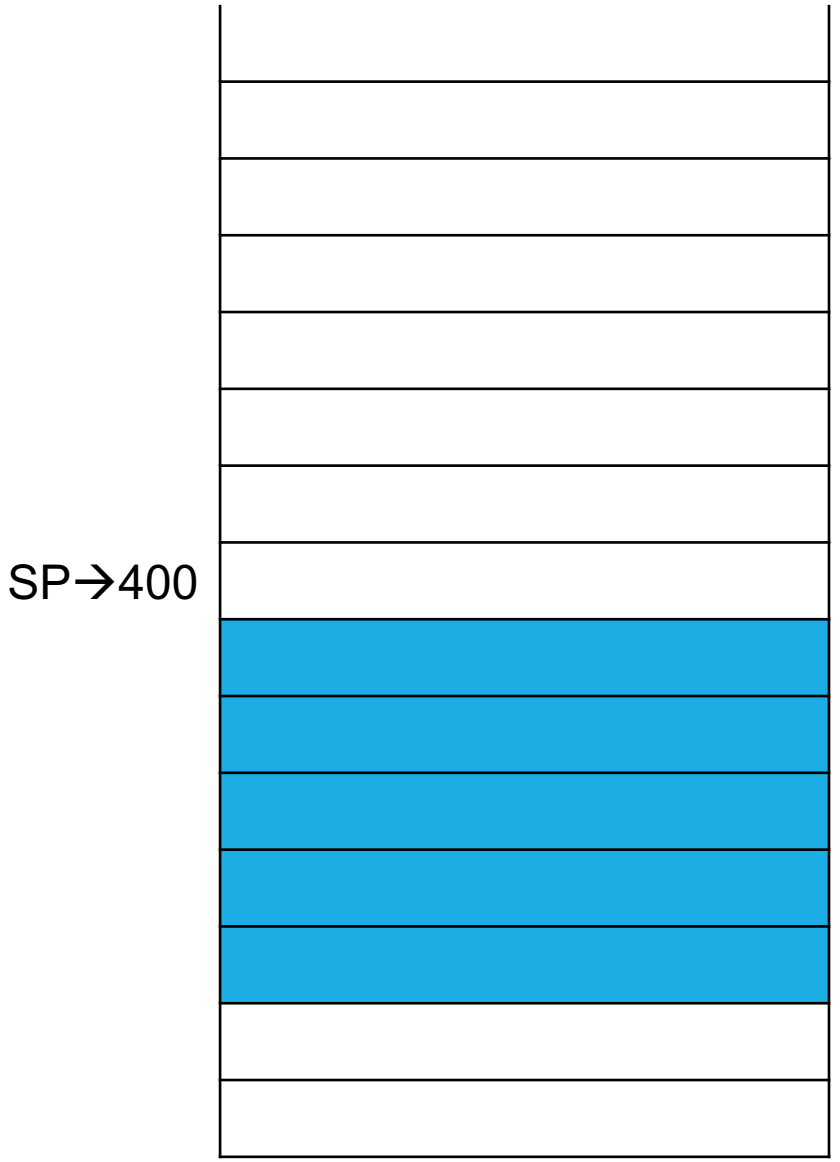
Stack Memory

- Call Stack
- What is in a Frame?
  - Local variables
  - Return address
  - Return value, Parameters
  - Data to recover
  - …
- **A special register**: SP
- **A convention** of using stack



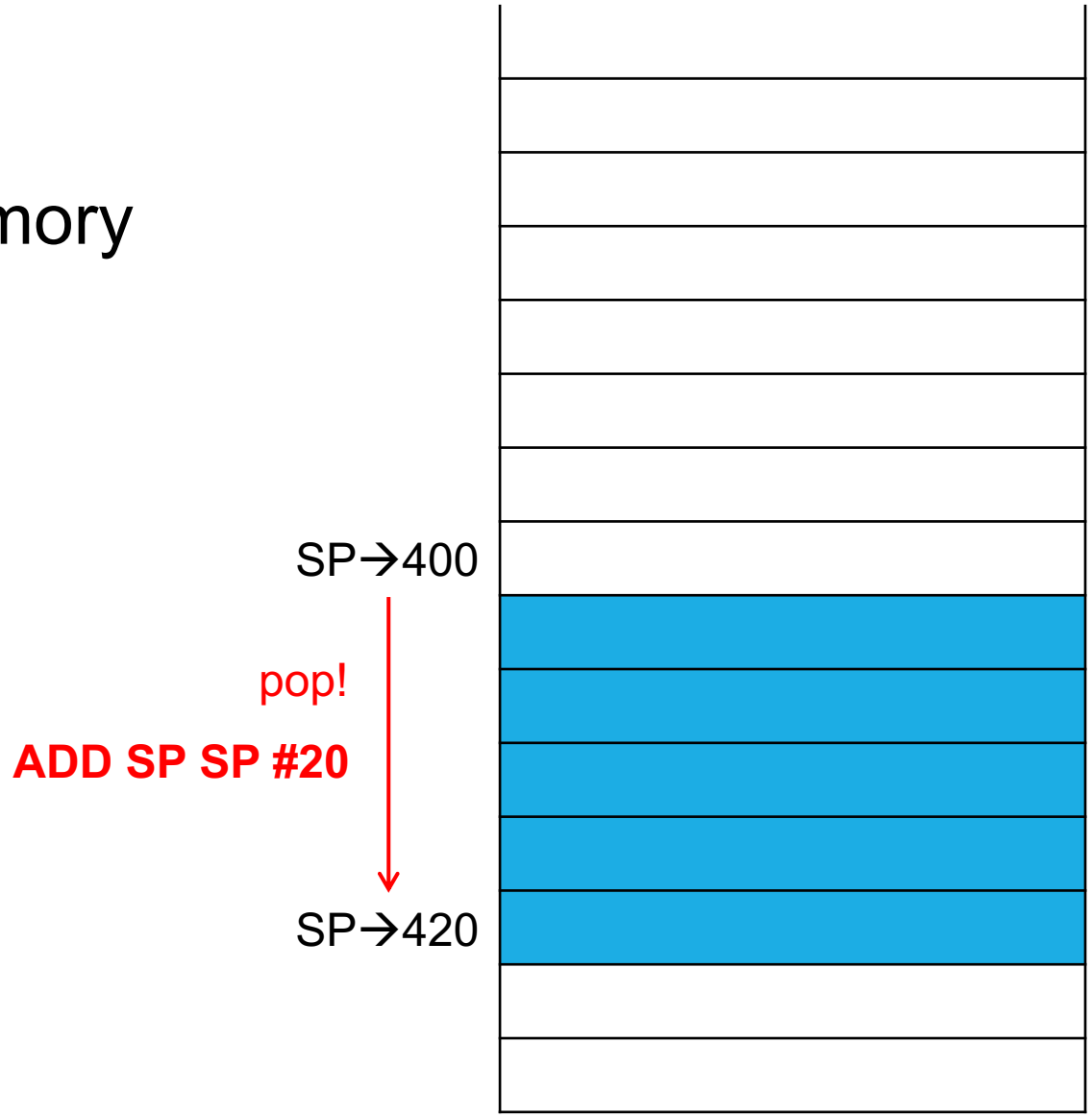
Stack Memory

- Pop or Release Stack Memory



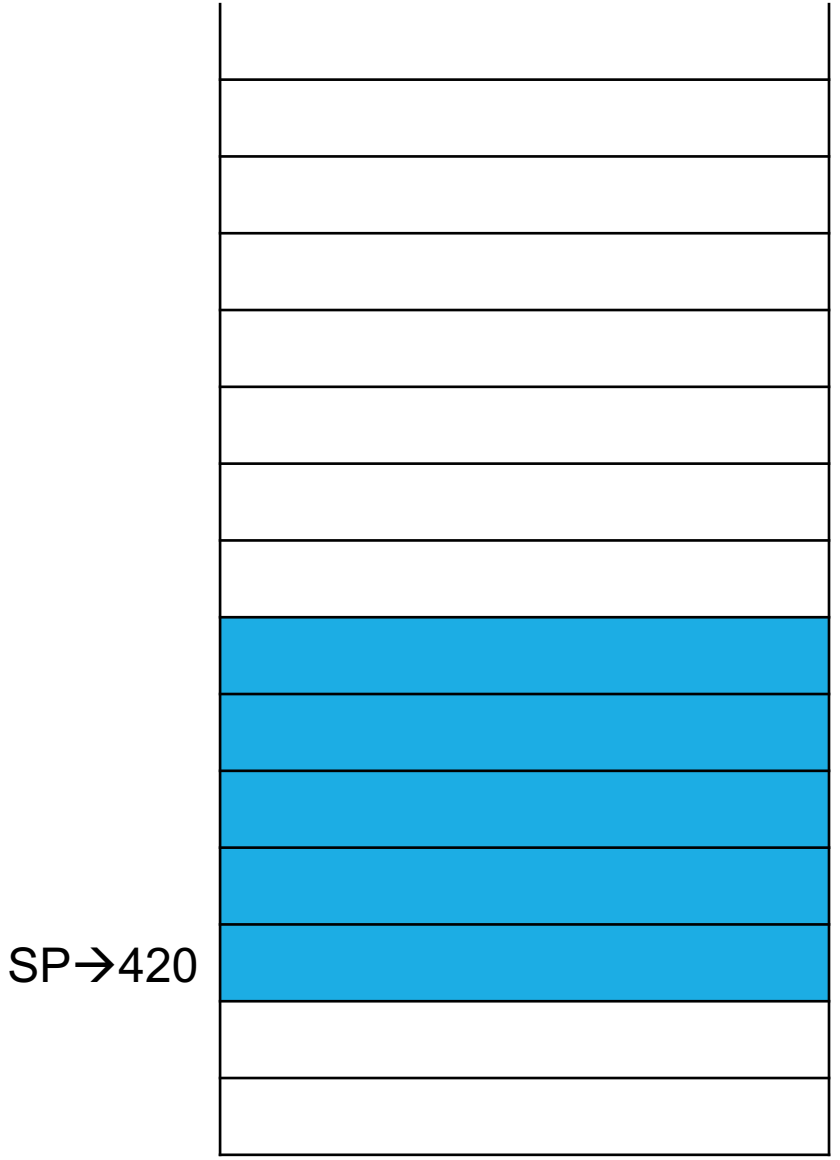
Stack Memory

- Pop or Release Stack Memory



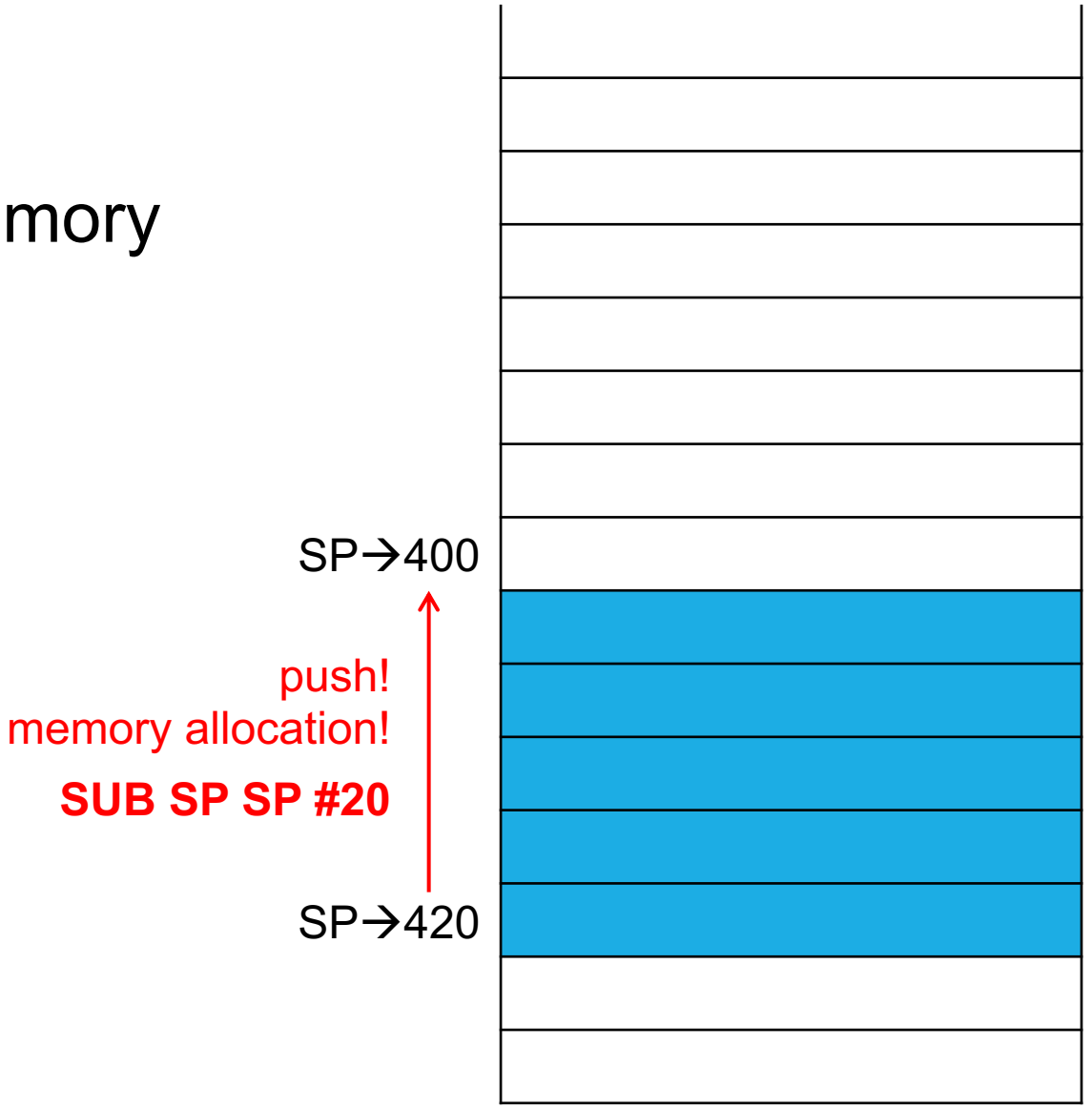
Stack Memory

- Push or Allocate Stack Memory



Stack Memory

- Push or Allocate Stack Memory



Stack Memory

- A convention of using stack

Stack Memory

- Before a call
  - If the procedure uses some registers, push them into the stack
  - Push (record) the **return address** & arguments to the stack
  - Jump to the code of callee

Stack Memory

- Before a call
  - If the procedure uses some registers, push them into the stack
  - Push (record) the **return address** & arguments to the stack
  - Jump to the code of callee
- During a call
  - Allocate memory spaces for locals; and actions based on the locals

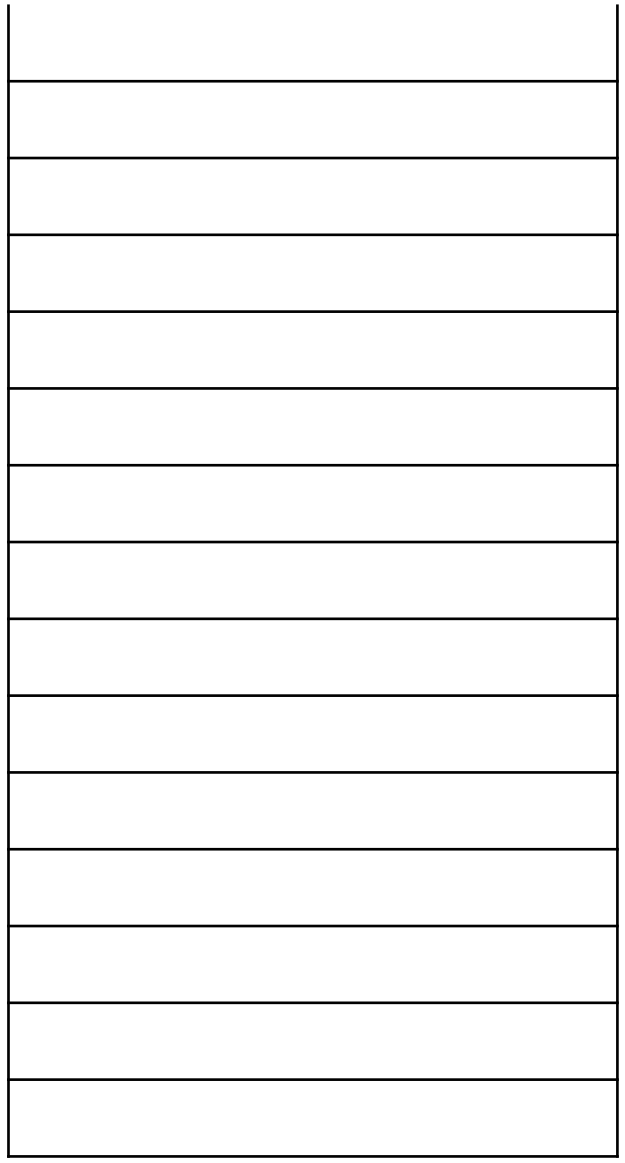
Stack Memory

- Before a call
  - If the procedure uses some registers, push them into the stack
  - Push (record) the **return address** & arguments to the stack
  - Jump to the code of callee
- During a call
  - Allocate memory spaces for locals; and actions based on the locals
- After a call
  - Pop the frame;
  - Return to the caller according to the **return address** in the stack
  - Recover data; Get the return value if the function has a return value

Stack Memory

```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

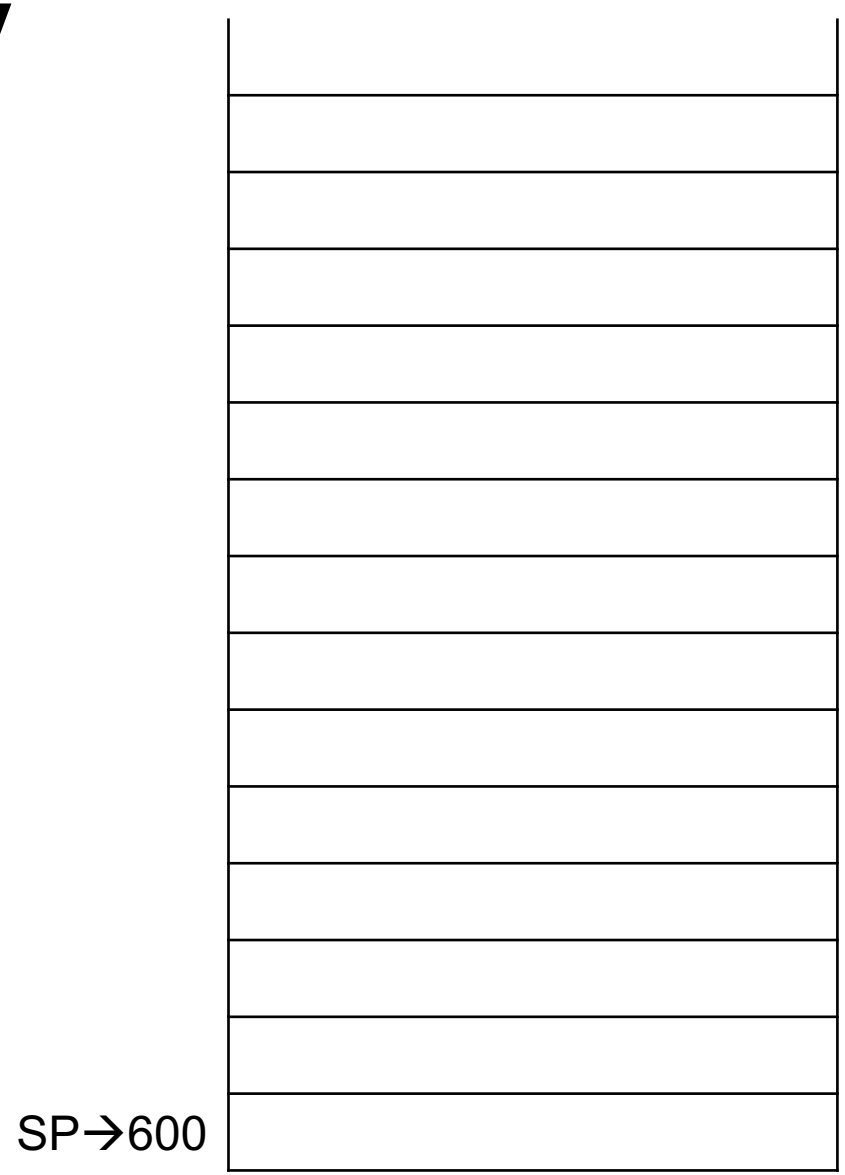
Stack Memory



```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack



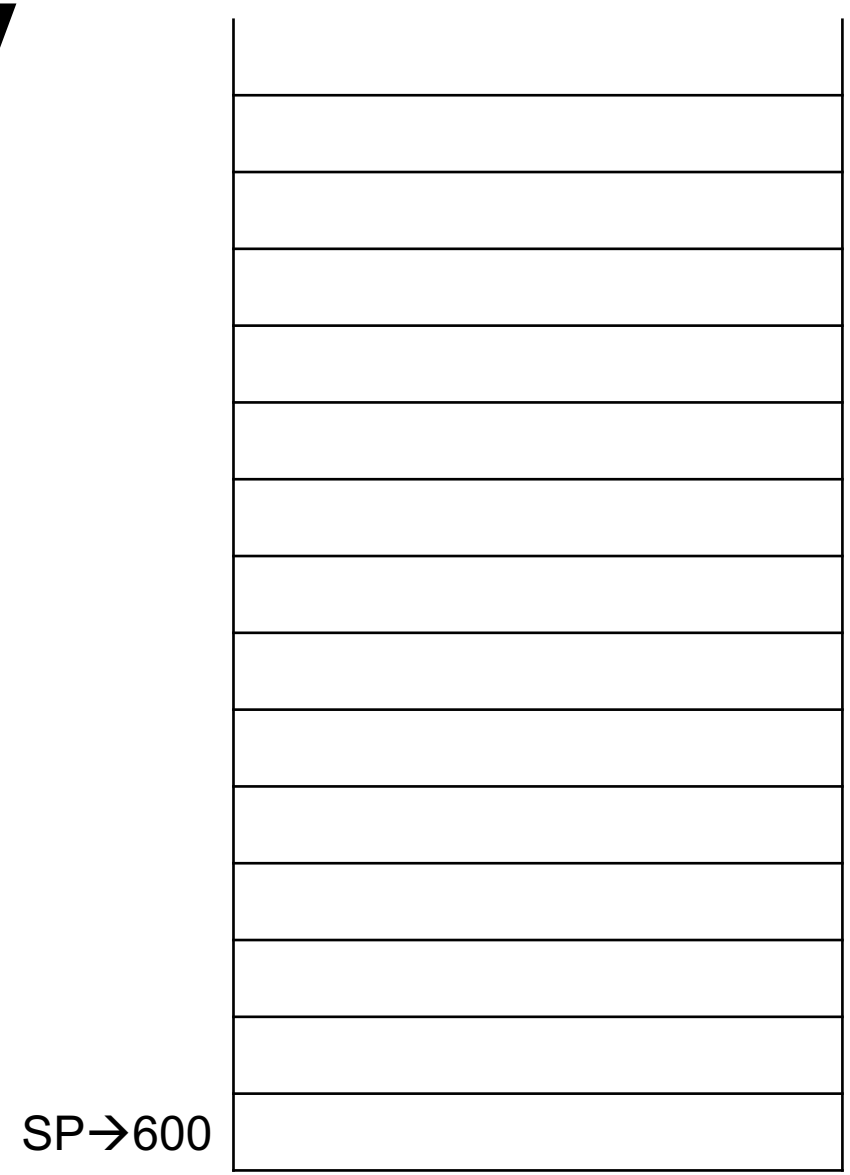

```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

foo




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

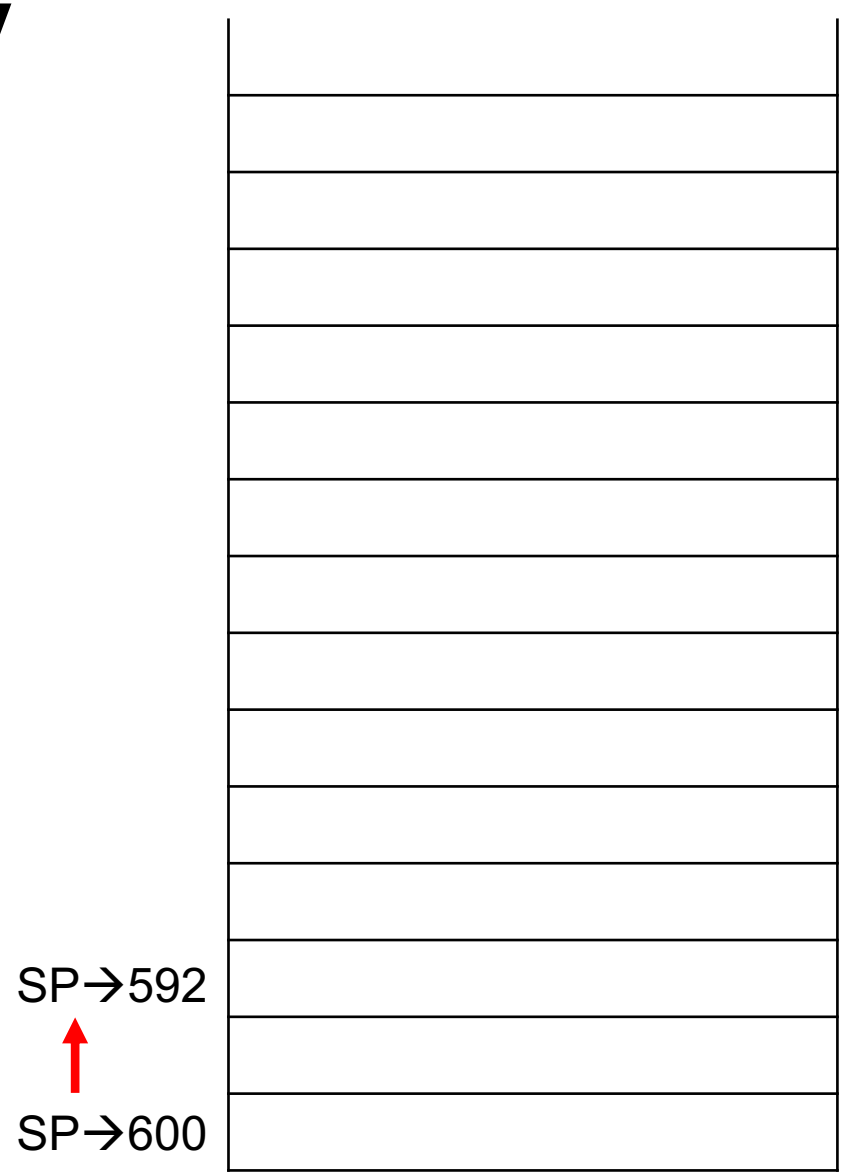
Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

foo




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

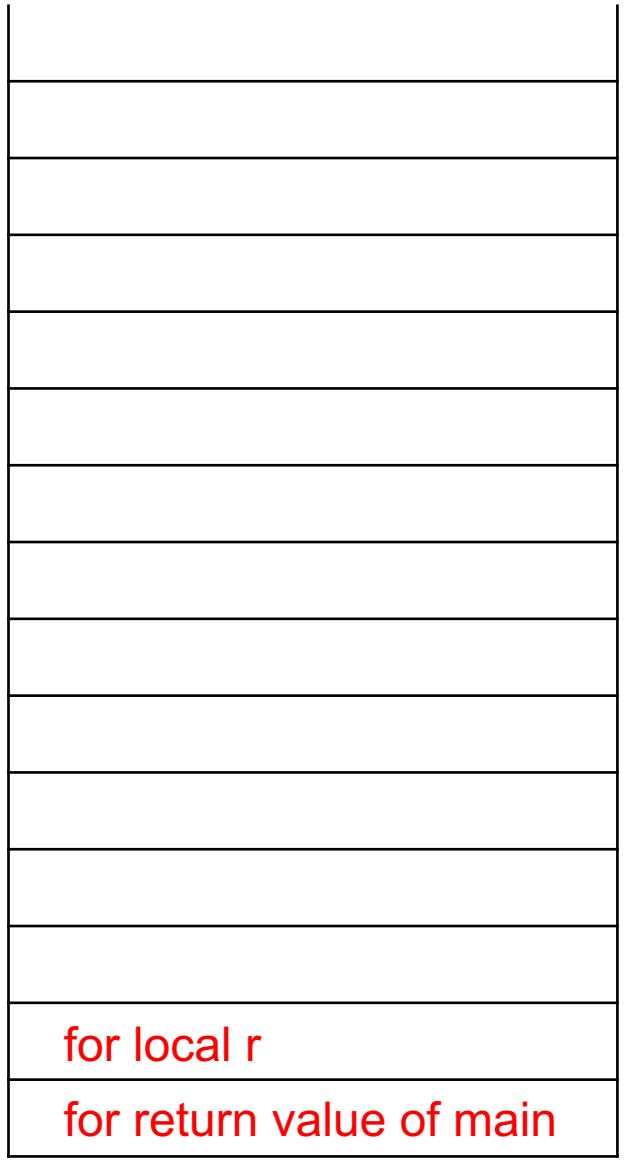
LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

foo

SP→592




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

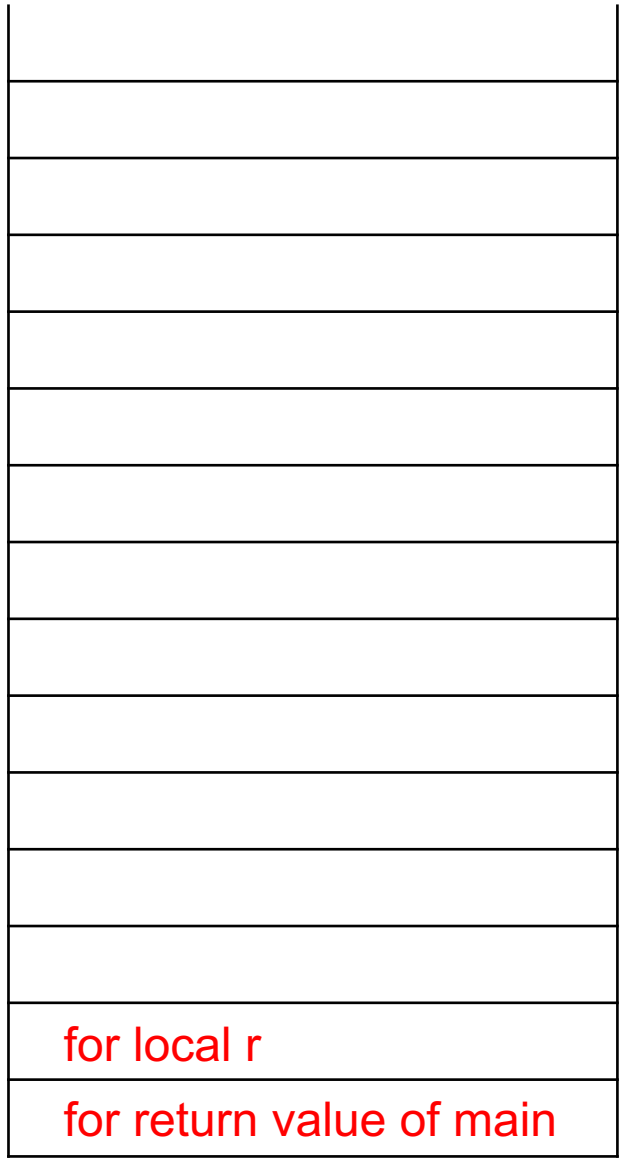
main

SUB SP, SP, #8 // allocate space for return and locals

.....

foo

SP→592




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

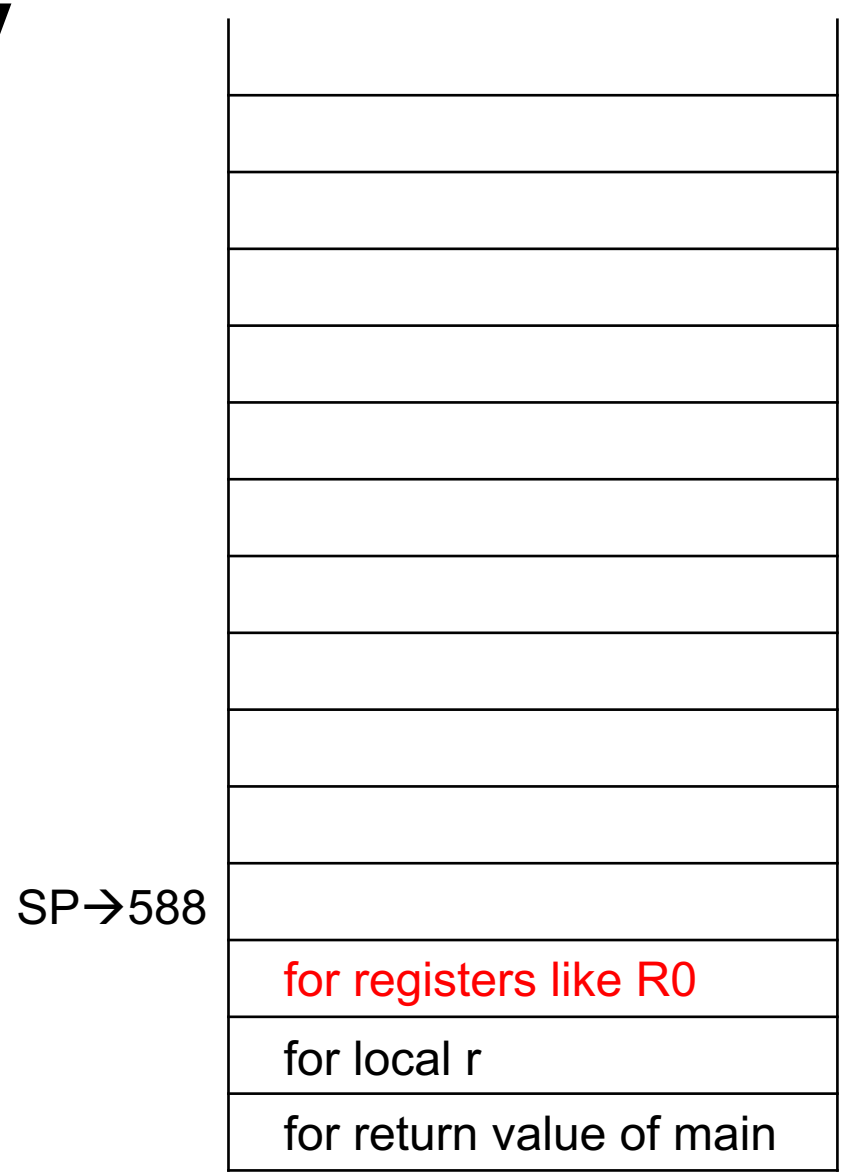
SUB SP, SP, #8 // allocate space for return and locals

……

SUB SP, SP, #4 // record registers

ST 4(SP), R0

foo




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

……

SUB SP, SP, #4 // record registers

ST 4(SP), R0

SUB SP, SP, #4 // record return addr

ST 4(SP), L_1

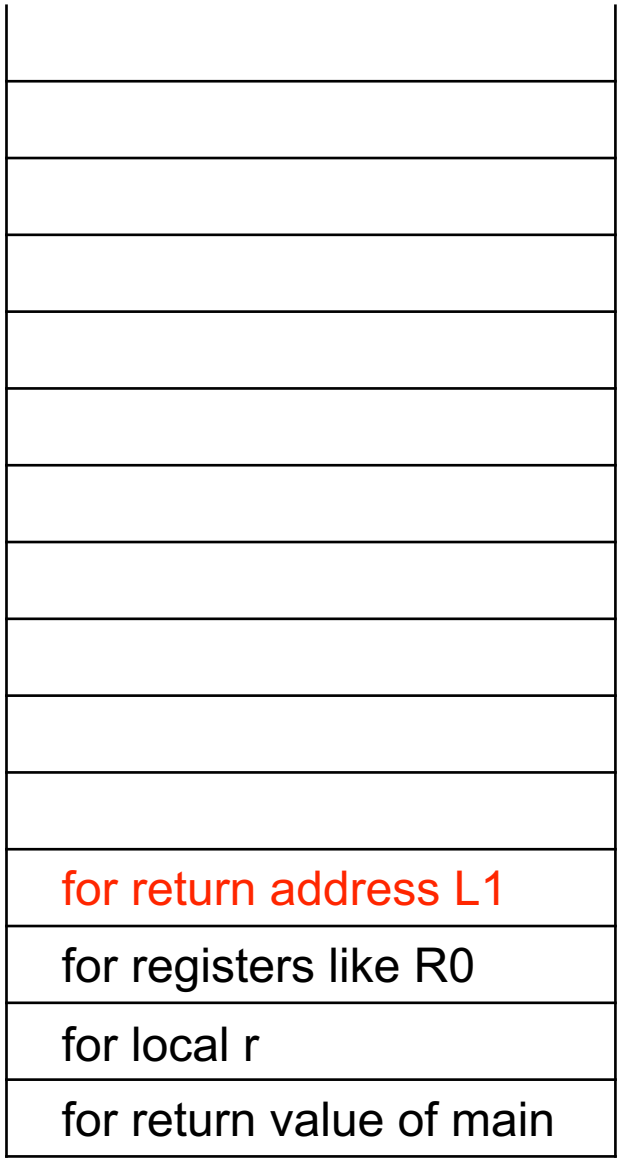
BR L_2

L_1 …….

L_2 ……. foo

BR ……

SP→584




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

……

SUB SP, SP, #4 // record registers

ST 4(SP), R0

SUB SP, SP, #4 // record return addr

ST 4(SP), L_1

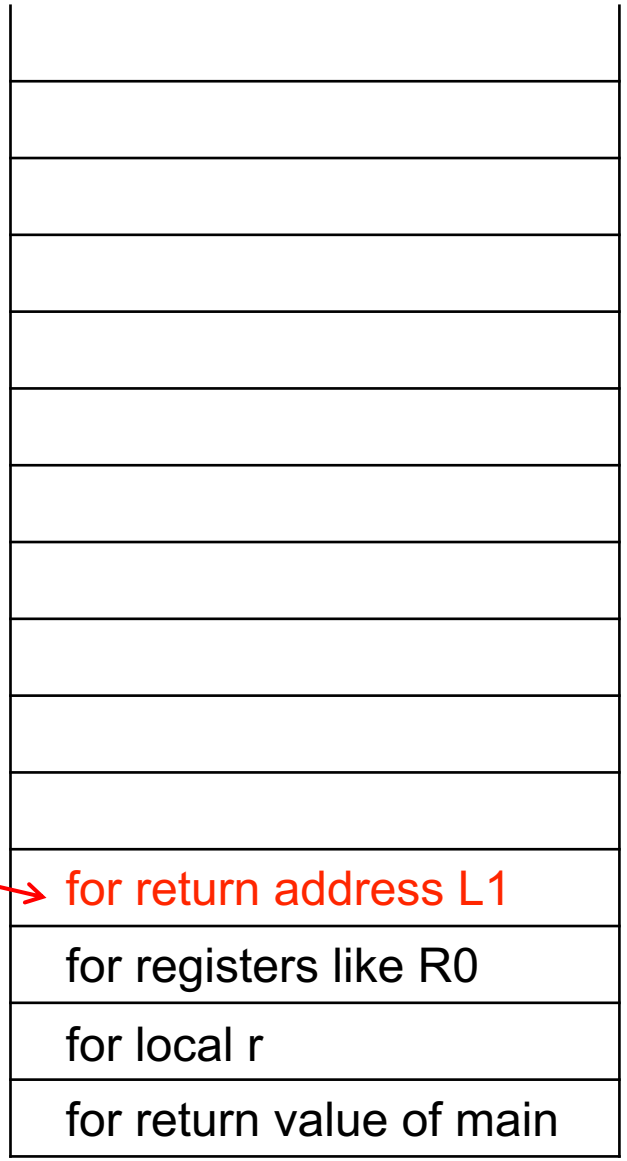
BR L_2

L_1 …….

L_2 ……. foo

BR ……

SP→584




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

……

SUB SP, SP, #4 // record registers

ST 4(SP), R0

SUB SP, SP, #4 // record return addr

ST 4(SP), L_1

BR L_2

L_1 …….

L_2 ……. foo

BR ……

SP→584

for return address L1
for registers like R0
for local r
for return value of main

```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

……

SUB SP, SP, #4 // record registers

ST 4(SP), R0

SUB SP, SP, #4 // record return addr

ST 4(SP), L_1

BR L_2

L_1 …….

L_2 ……. foo

BR ……

SP→584

for return address L1
for registers like R0
for local r
for return value of main

```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

……

SUB SP, SP, #4 // record registers

ST 4(SP), R0

SUB SP, SP, #4 // record return addr

ST 4(SP), L_1

BR L_2

L_1 …….

L_2 ……. foo

BR ……

SP→584

for return address L1
for registers like R0
for local r
for return value of main

```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

……

SUB SP, SP, #4 // record registers
ST 4(SP), R0

SUB SP, SP, #4 // record return addr
ST 4(SP), L_1
BR L_2

L_1 ……

L_2

BR ……

SP→584

for return address L1
for registers like R0
for local r
for return value of main

```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

……

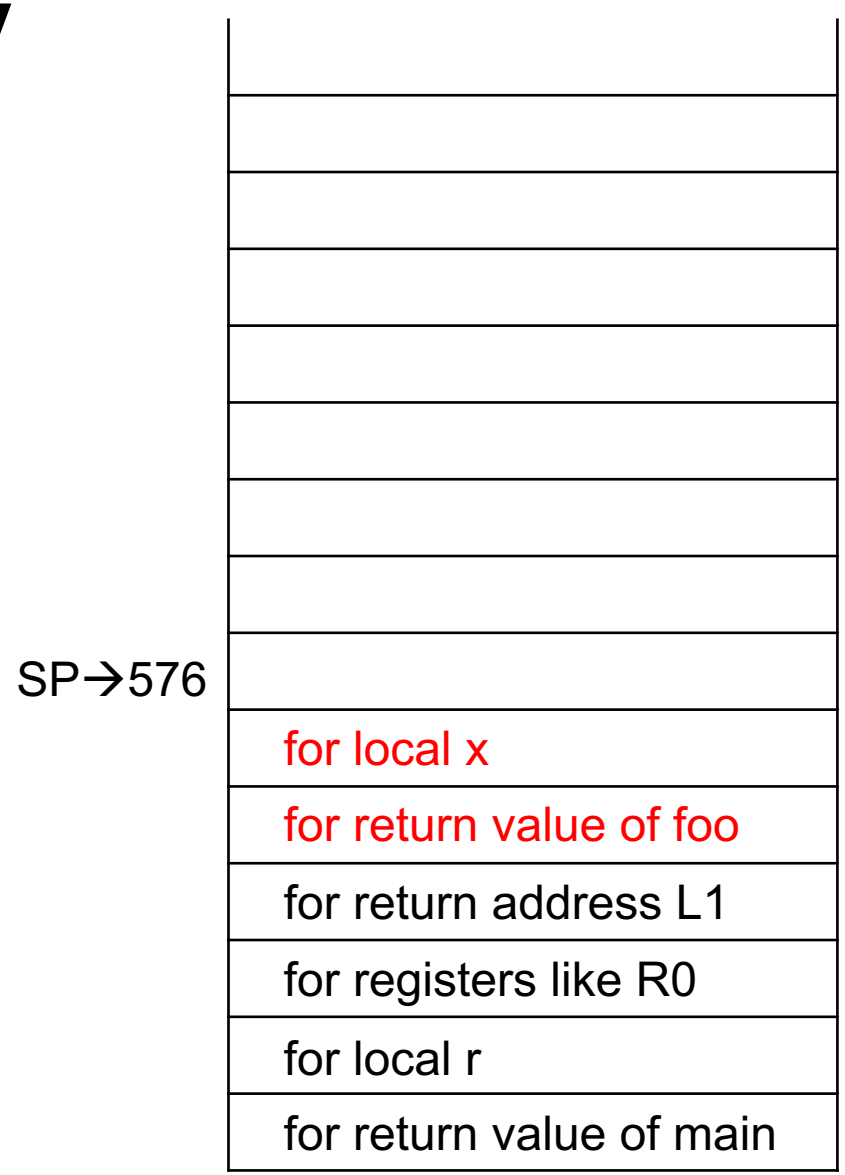
SUB SP, SP, #4 // record registers
ST 4(SP), R0

SUB SP, SP, #4 // record return addr
ST 4(SP), L_1
BR L_2

L_1 ……

L_2 **SUB SP, SP, #8**
…… // x = y - z

BR ……




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

……

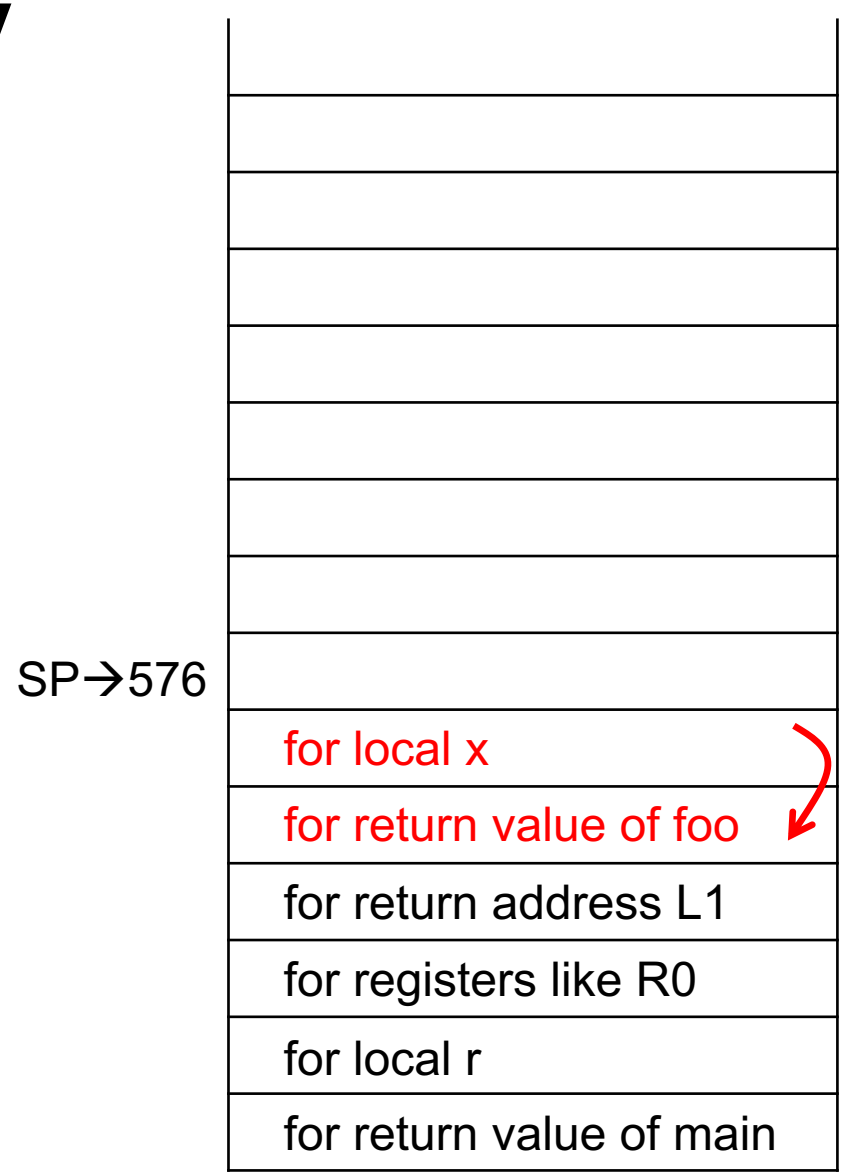
SUB SP, SP, #4 // record registers
ST 4(SP), R0

SUB SP, SP, #4 // record return addr
ST 4(SP), L_1
BR L_2

L_1 …….

L_2

```
SUB SP, SP, #8
……
LD R0, 4(SP)    ST 8(SP), R0

BR ……
```




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

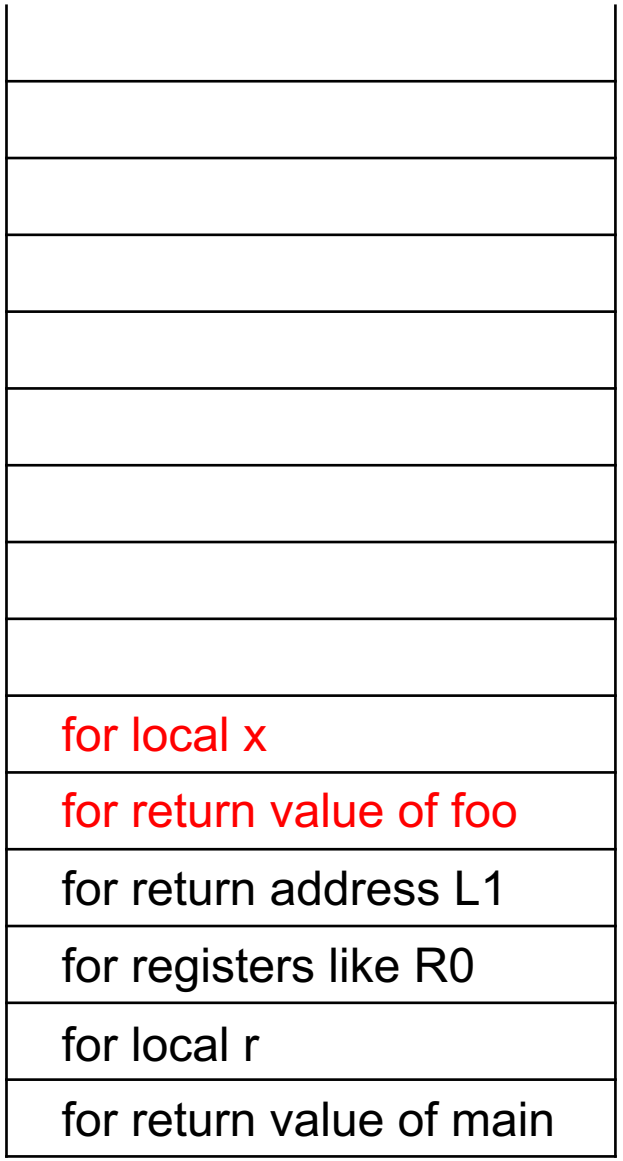
……

SUB SP, SP, #4 // record registers
ST 4(SP), R0

SUB SP, SP, #4 // record return addr
ST 4(SP), L_1
BR L_2

L_1 ……

L_2
SUB SP, SP, #8
……
LD R0, 4(SP) ST 8(SP), R0
ADD SP, SP, #8
BR ……




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

……

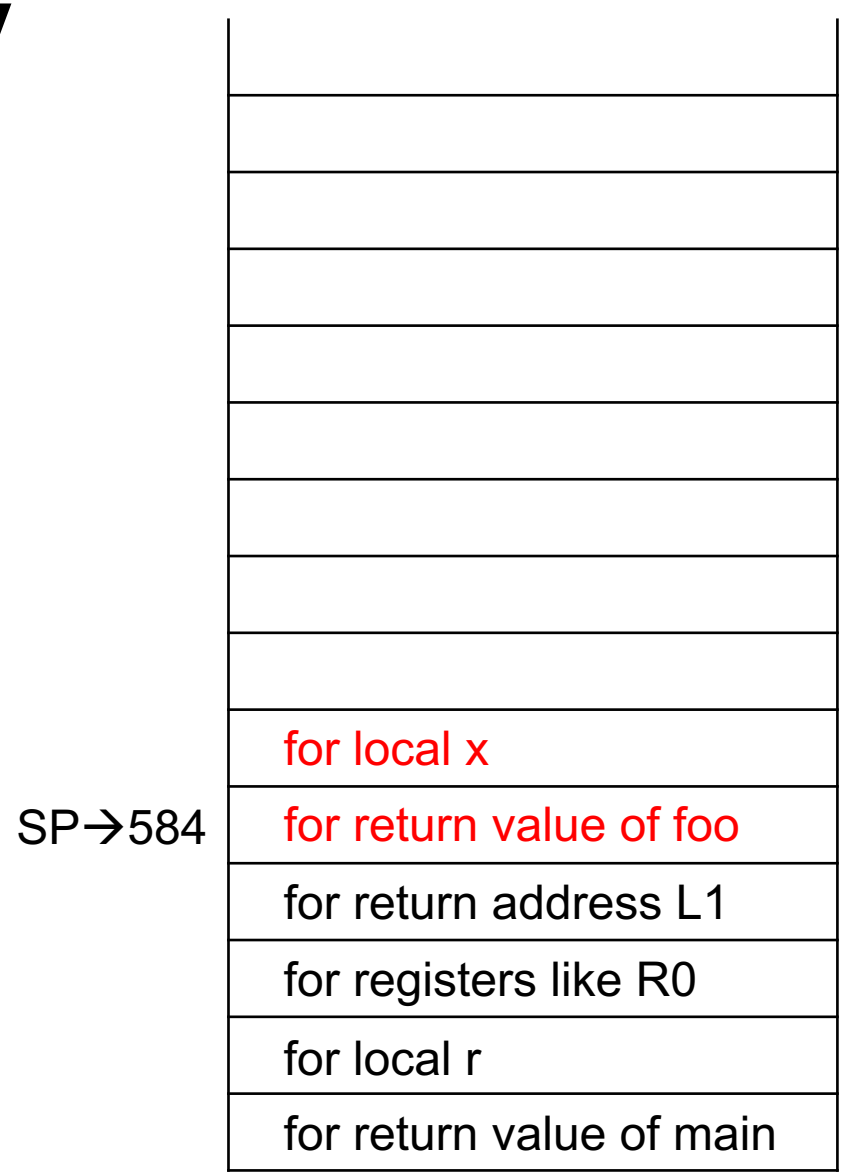
SUB SP, SP, #4 // record registers
ST 4(SP), R0

SUB SP, SP, #4 // record return addr
ST 4(SP), L_1
BR L_2

L_1 ……

L_2

```
SUB SP, SP, #8
……
LD R0, 4(SP)    ST 8(SP), R0
ADD SP, SP, #8
BR 4(SP)
```




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

……

SUB SP, SP, #4 // record registers
ST 4(SP), R0

SUB SP, SP, #4 // record return addr
ST 4(SP), L_1
BR L_2

L_1 ……

L_2
```
SUB SP, SP, #8
……
LD R0, 4(SP)    ST 8(SP), R0
ADD SP, SP, #8
BR 4(SP)
```

SP→584

for local x
for return value of foo
for return address L1
for registers like R0
for local r
for return value of main

```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

LD SP, #600 // initialize the stack

main

SUB SP, SP, #8 // allocate space for return and locals

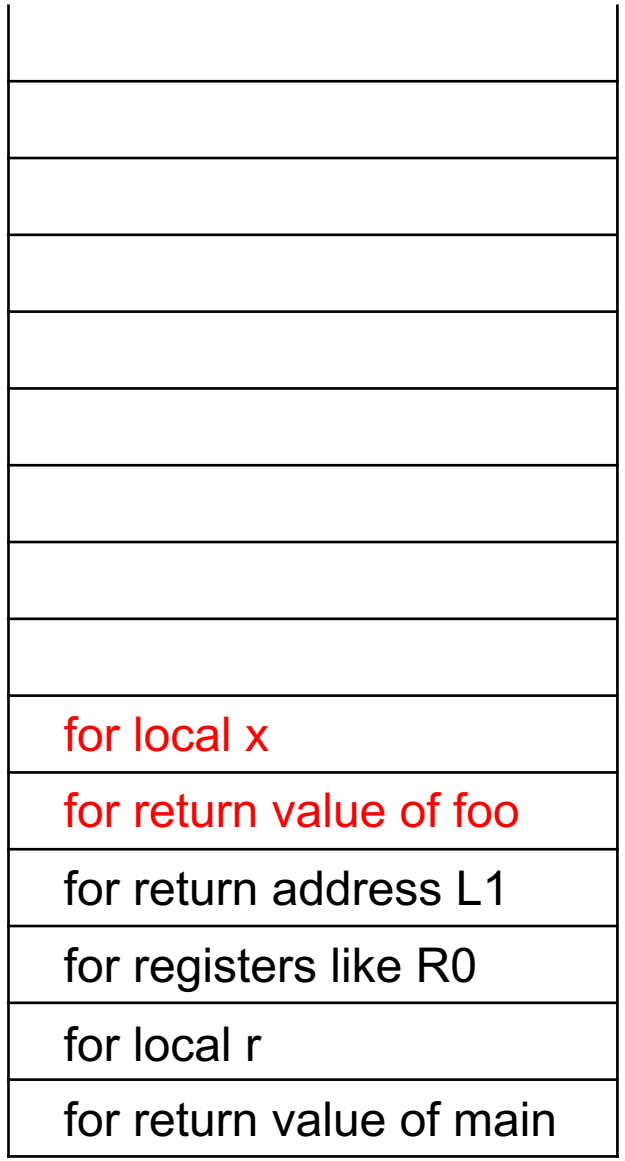
……

SUB SP, SP, #4 // record registers
ST 4(SP), R0

SUB SP, SP, #4 // record return addr
ST 4(SP), L_1
BR L_2

L_1 ……

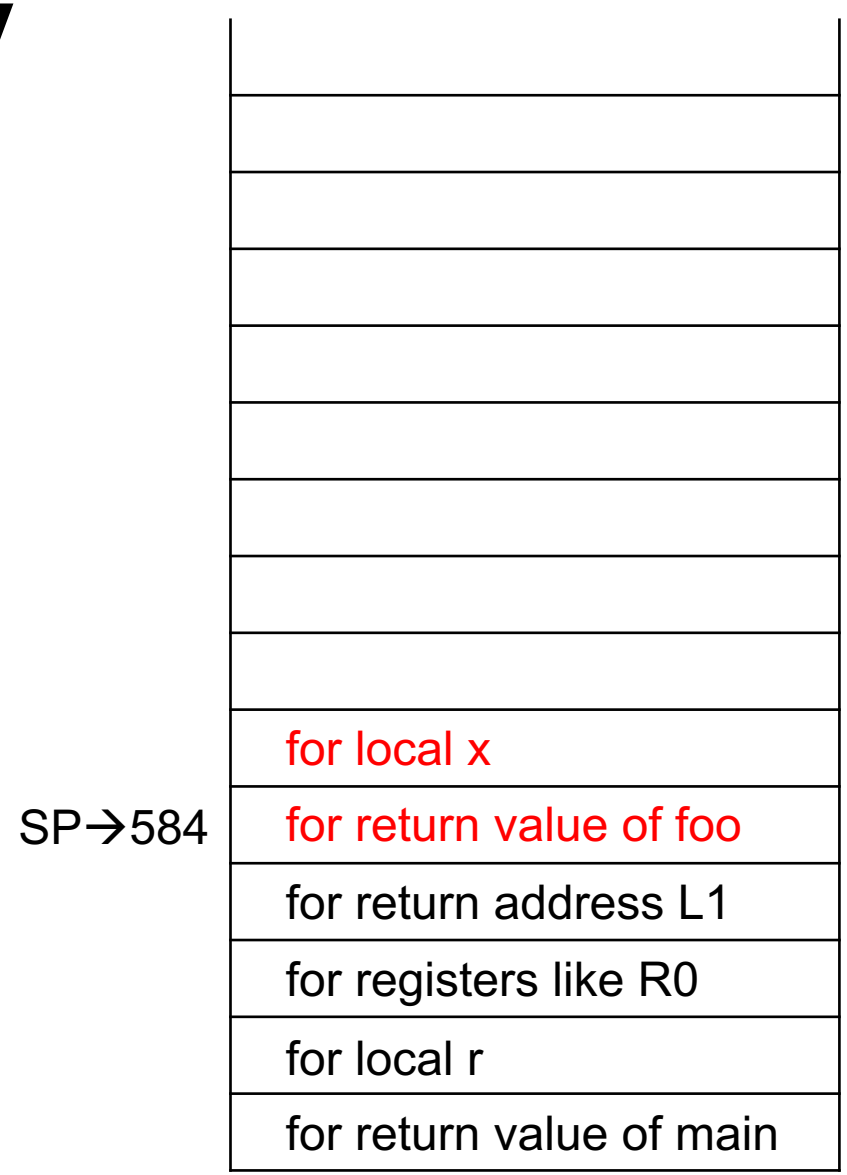
L_2 SUB SP, SP, #8
……
LD R0, 4(SP) ST 8(SP), R0
ADD SP, SP, #8
BR 4(SP)




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

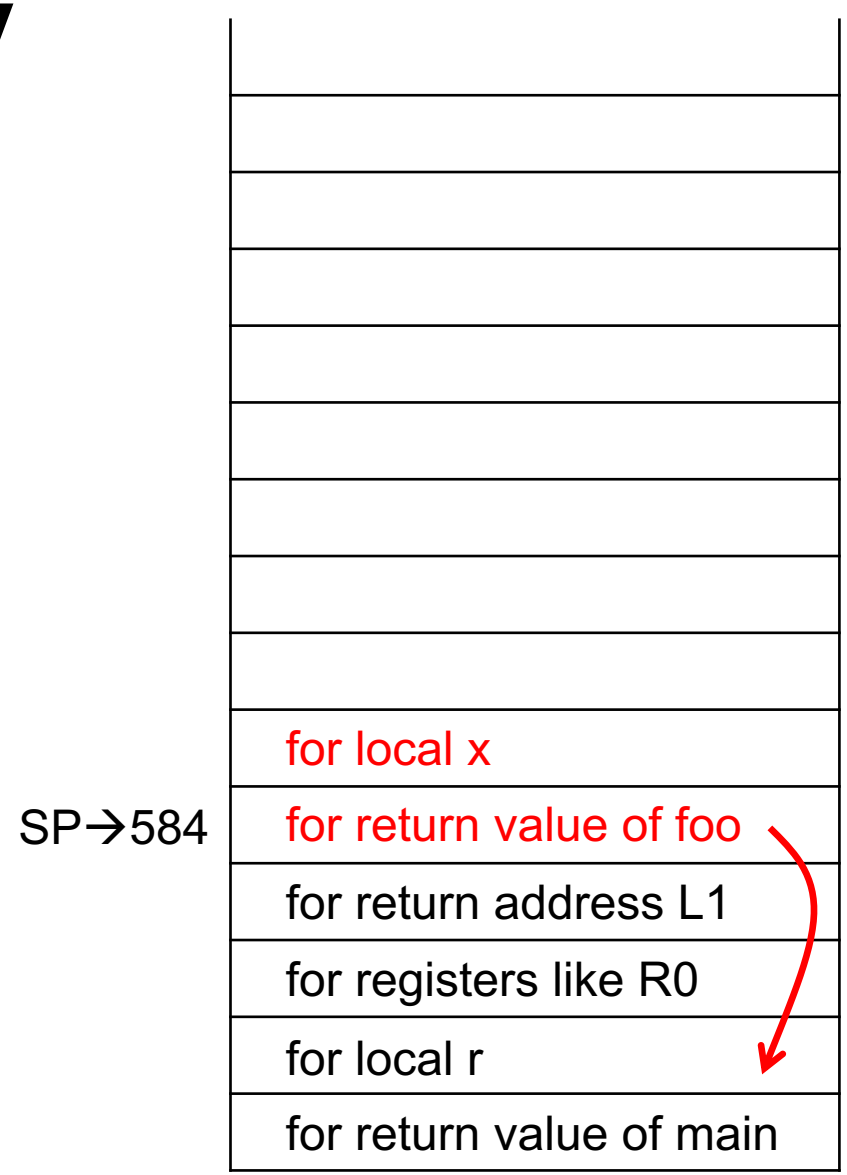
```
LD SP, #600          // initialize the stack

                                    main
SUB SP, SP, #8       // allocate space
                        for return and locals
……
SUB SP, SP, #4       // record registers
ST 4(SP), R0

SUB SP, SP, #4       // record return addr
ST 4(SP), L1
BR L2

L1  LD R0, 8(SP)     // recover R0
```




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

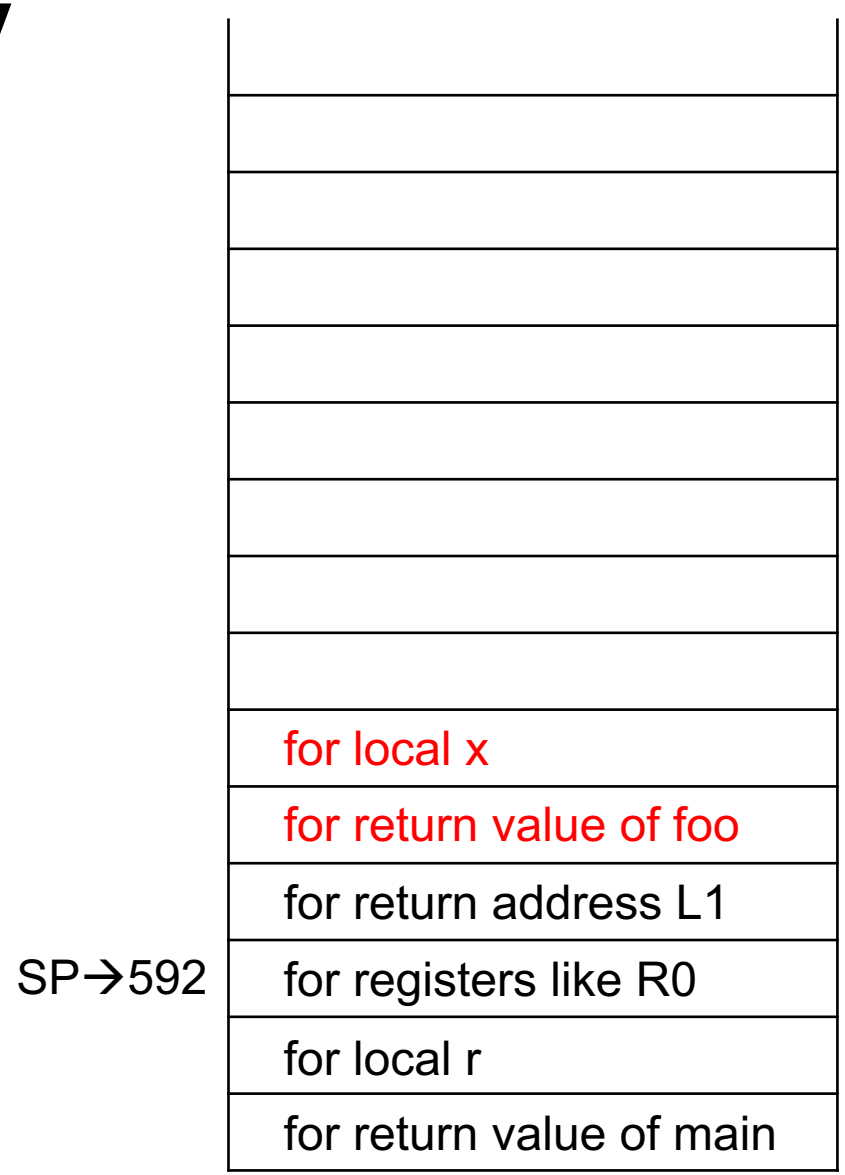
```
LD SP, #600          // initialize the stack

                                    main
SUB SP, SP, #8       // allocate space
                        for return and locals
……
SUB SP, SP, #4       // record registers
ST 4(SP), R0

SUB SP, SP, #4       // record return addr
ST 4(SP), L1
BR L2

L1  LD R0, 8(SP)     // recover R0
    LD R1, 0(SP)     ST  12(SP), R1
```




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

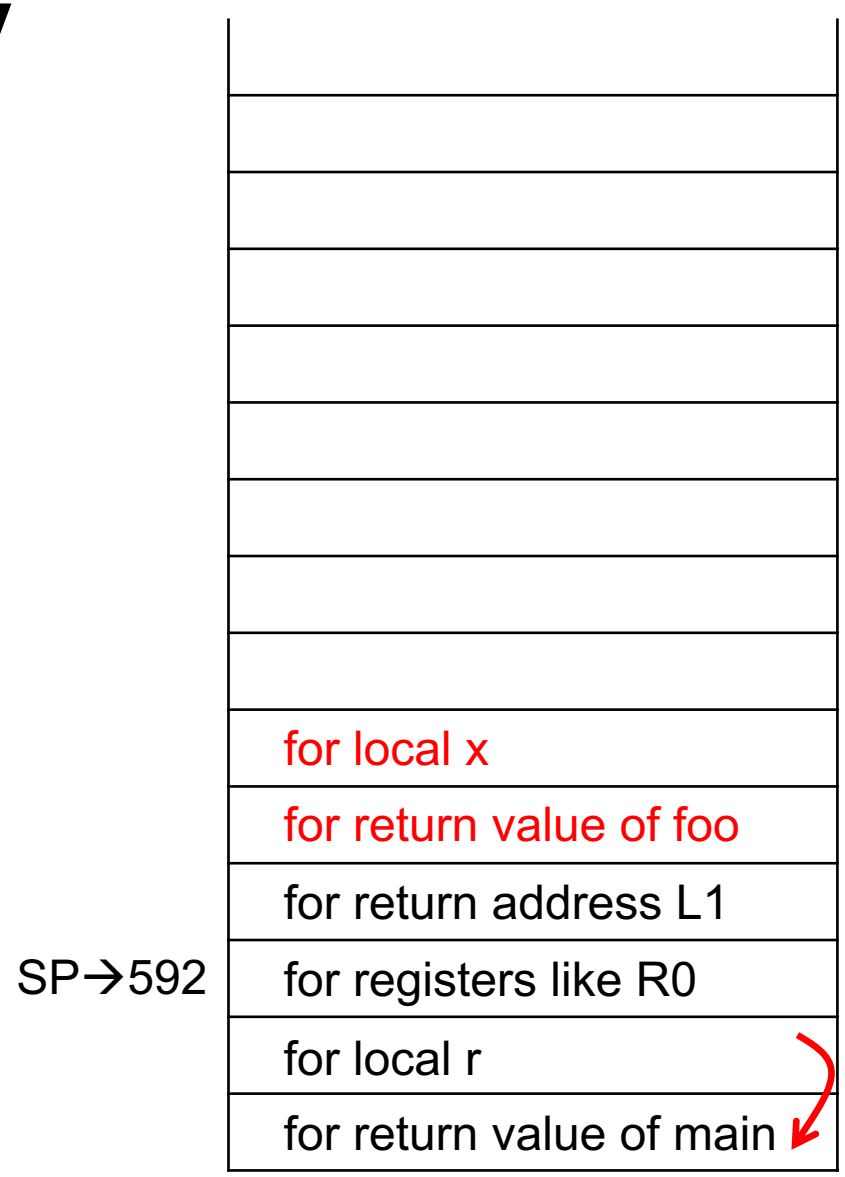
```
LD SP, #600          // initialize the stack

                                    main
SUB SP, SP, #8       // allocate space
                        for return and locals
……
SUB SP, SP, #4       // record registers
ST 4(SP), R0

SUB SP, SP, #4       // record return addr
ST 4(SP), L1
BR L2

L1  LD R0, 8(SP)     // recover R0
    LD R1, 0(SP)     ST  12(SP), R1
    ADD SP, SP, #8   // pop the space
```




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

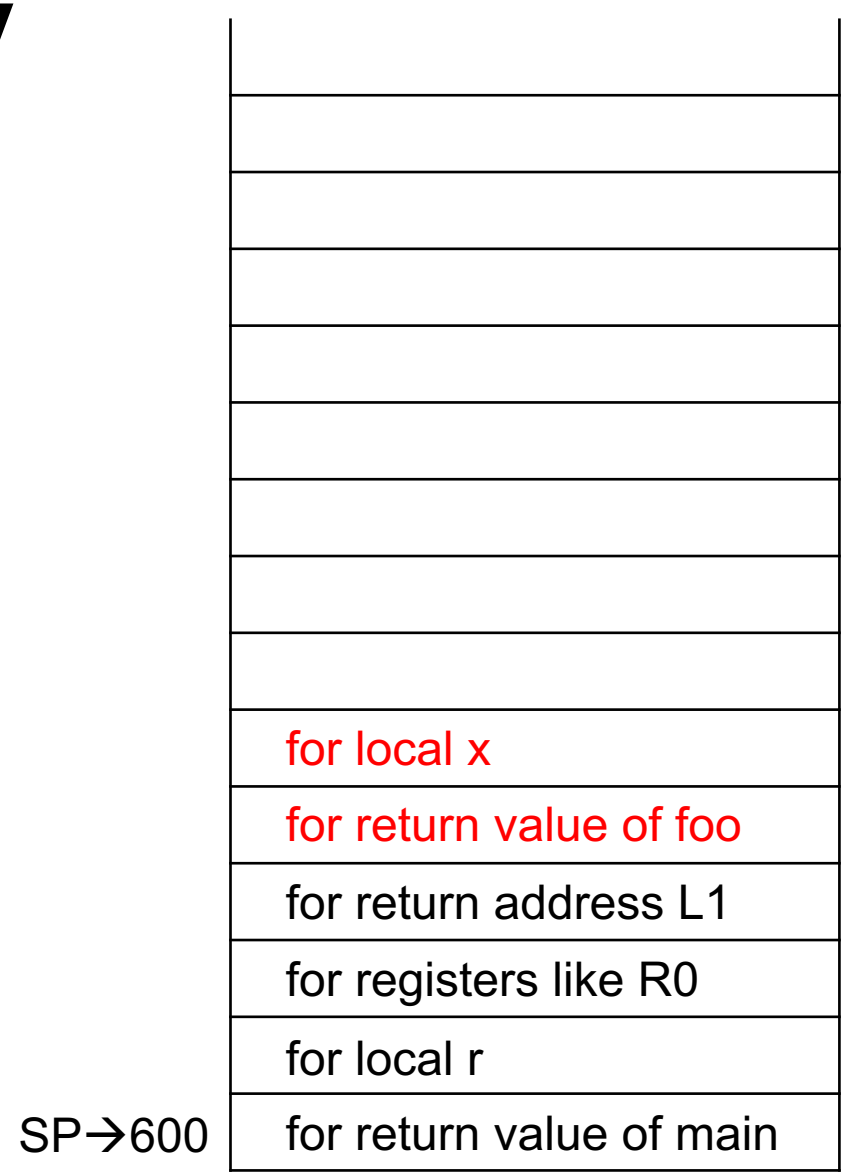
```
LD SP, #600          // initialize the stack

                                        main
SUB SP, SP, #8       // allocate space
                        for return and locals
……
SUB SP, SP, #4       // record registers
ST 4(SP), R0

SUB SP, SP, #4       // record return addr
ST 4(SP), L1
BR L2

L1  LD R0, 8(SP)     // recover R0
    LD R1, 0(SP)     ST  12(SP), R1
    ADD SP, SP, #8   // pop the space
    LD R0, 4(SP)     ST 8(SP), R0
```




```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Stack Memory

```
LD SP, #600          // initialize the stack

                                        main
SUB SP, SP, #8       // allocate space
                        for return and locals
……
SUB SP, SP, #4       // record registers
ST 4(SP), R0

SUB SP, SP, #4       // record return addr
ST 4(SP), L1
BR L2

L1  LD R0, 8(SP)     // recover R0
    LD R1, 0(SP)     ST  12(SP), R1
    ADD SP, SP, #8   // pop the space
    LD R0, 4(SP)     ST 8(SP), R0

    ADD SP, SP, #8
```

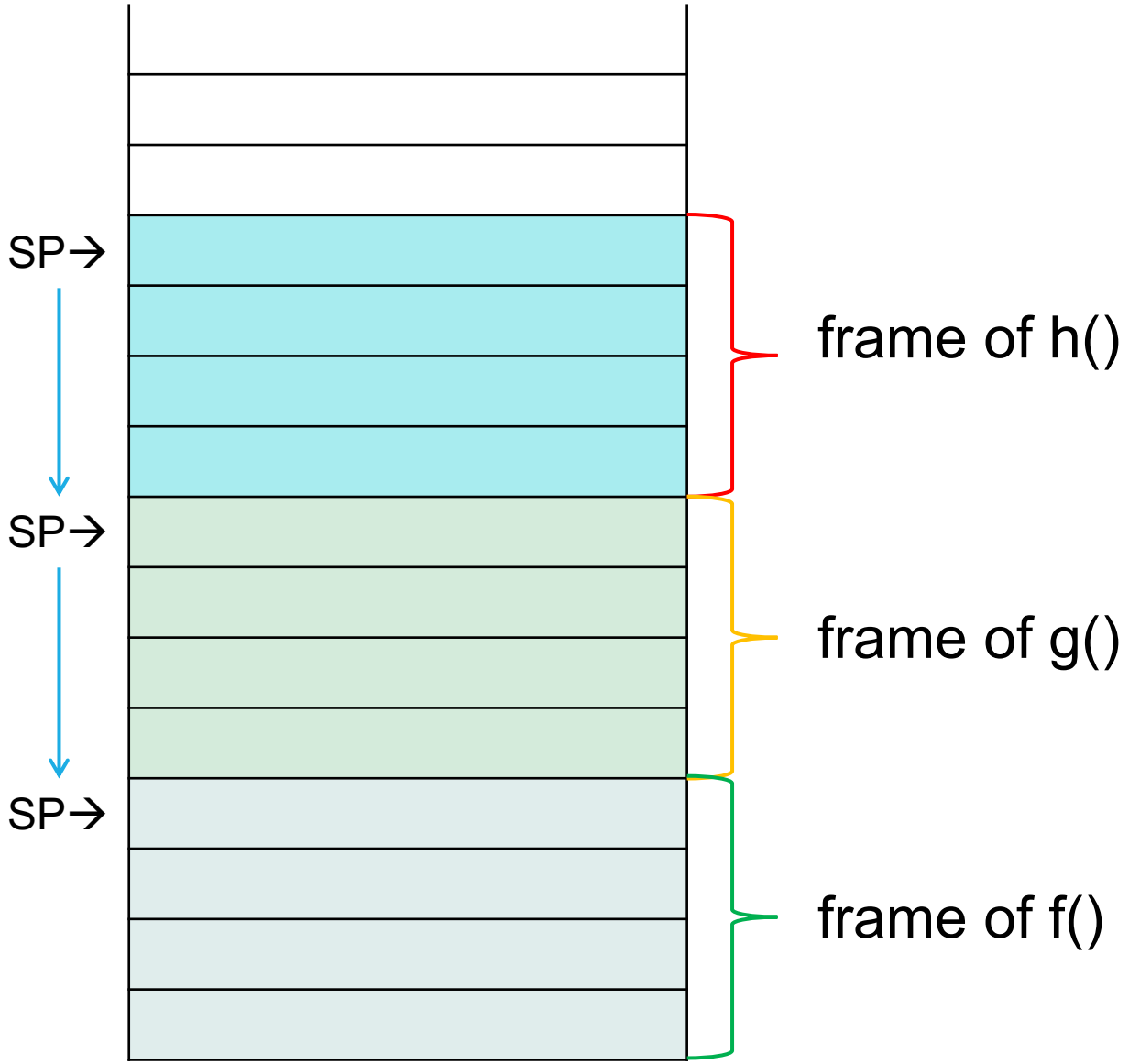



```
1.  int y, z;

2.  int main() {
3.      int r;
4.      ...
5.      r = foo();
6.      ...
7.      return r;
8.  }

9.  int foo() {
10.     int x;
11.     x = y - z;
12.     return x;
13. }
```

Jump via Return

- Call Stack

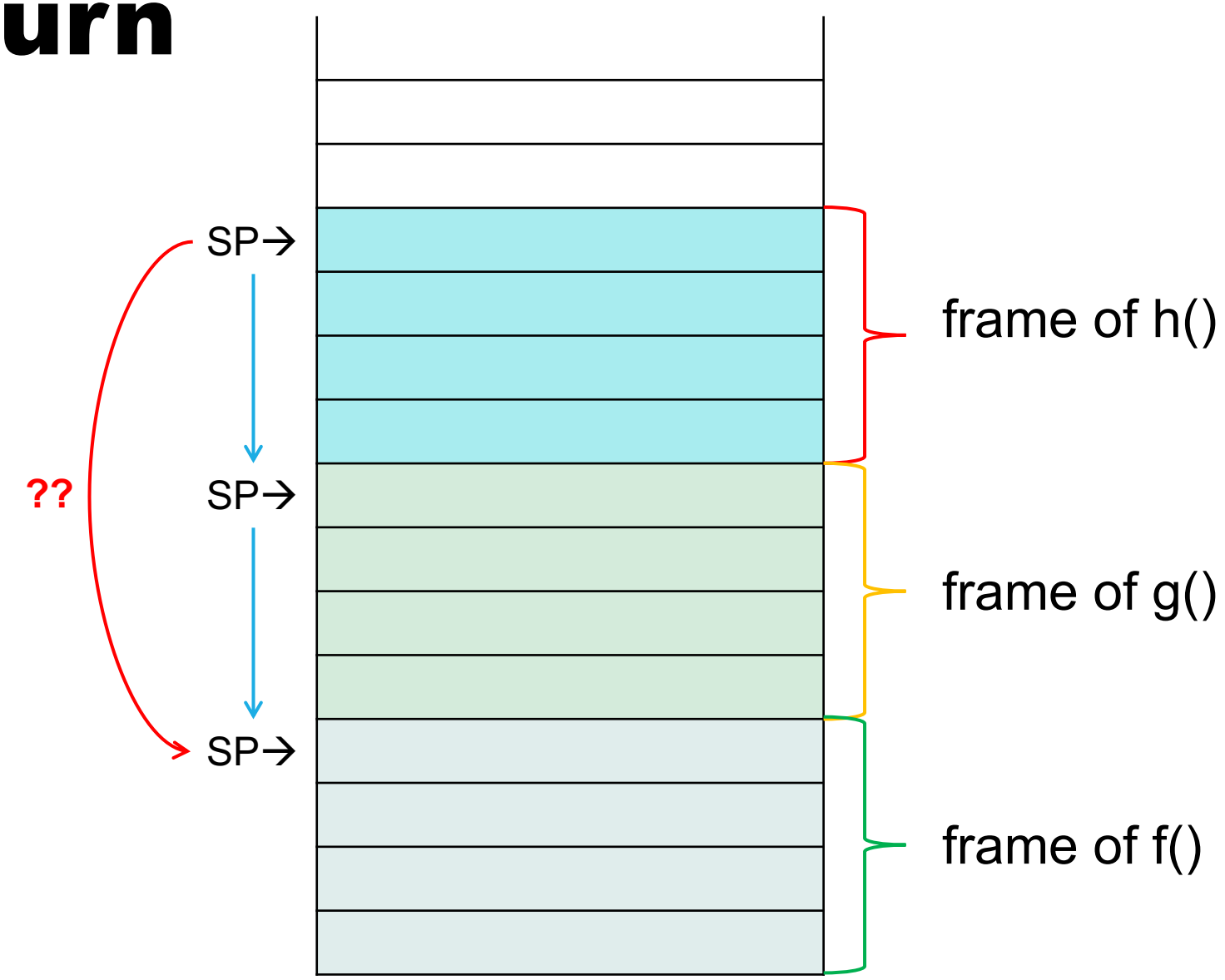
```
1.  void f() {
2.      …
3.      g();
4.      …
5.  }

6.  void g() {
7.      …
8.      h();
9.      …
10. }
```



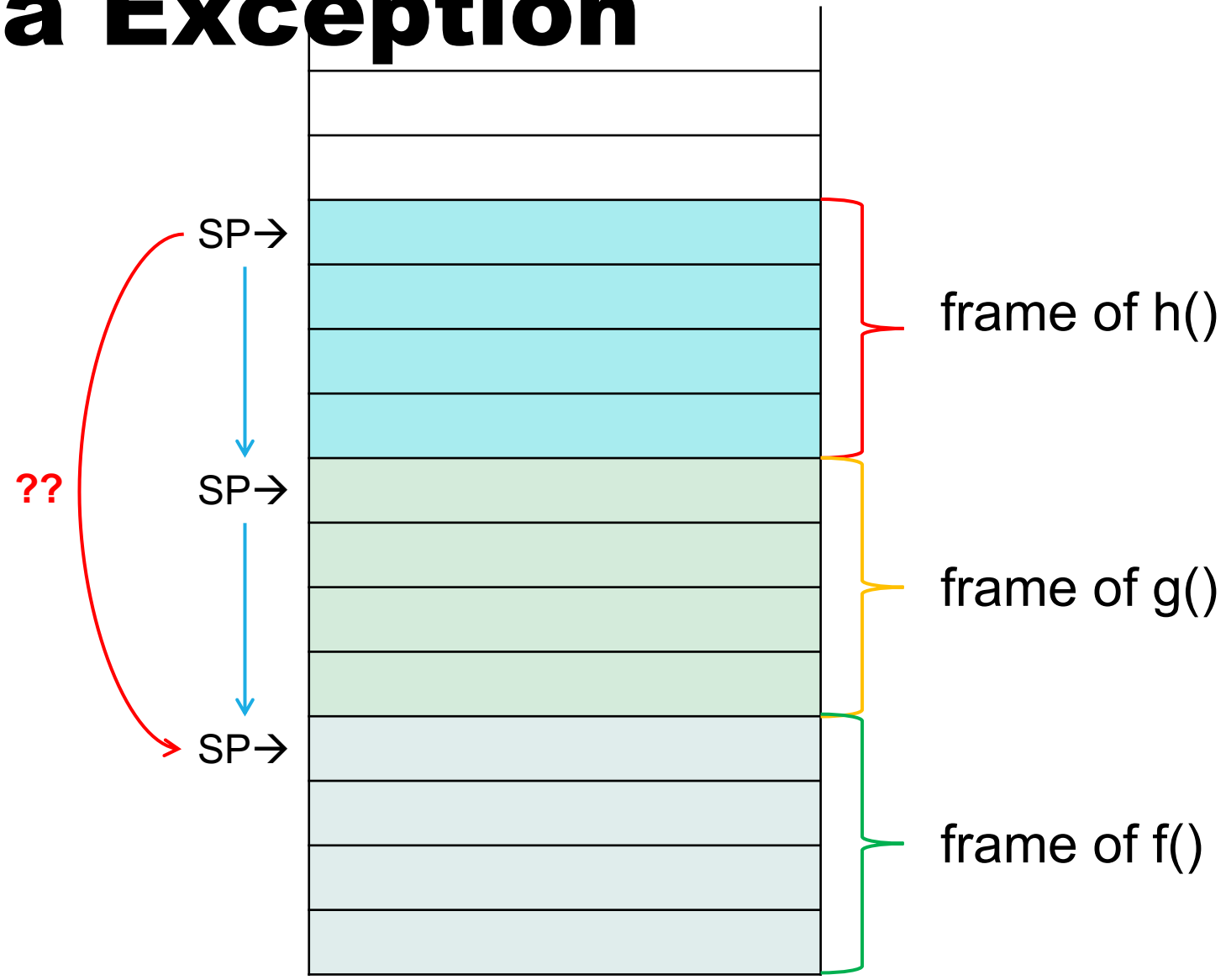
Jump via Return

- Call Stack

```
1.  void f() {
2.      …
3.      g();
4.      …
5.  }

6.  void g() {
7.      …
8.      h();
9.      …
10. }
```

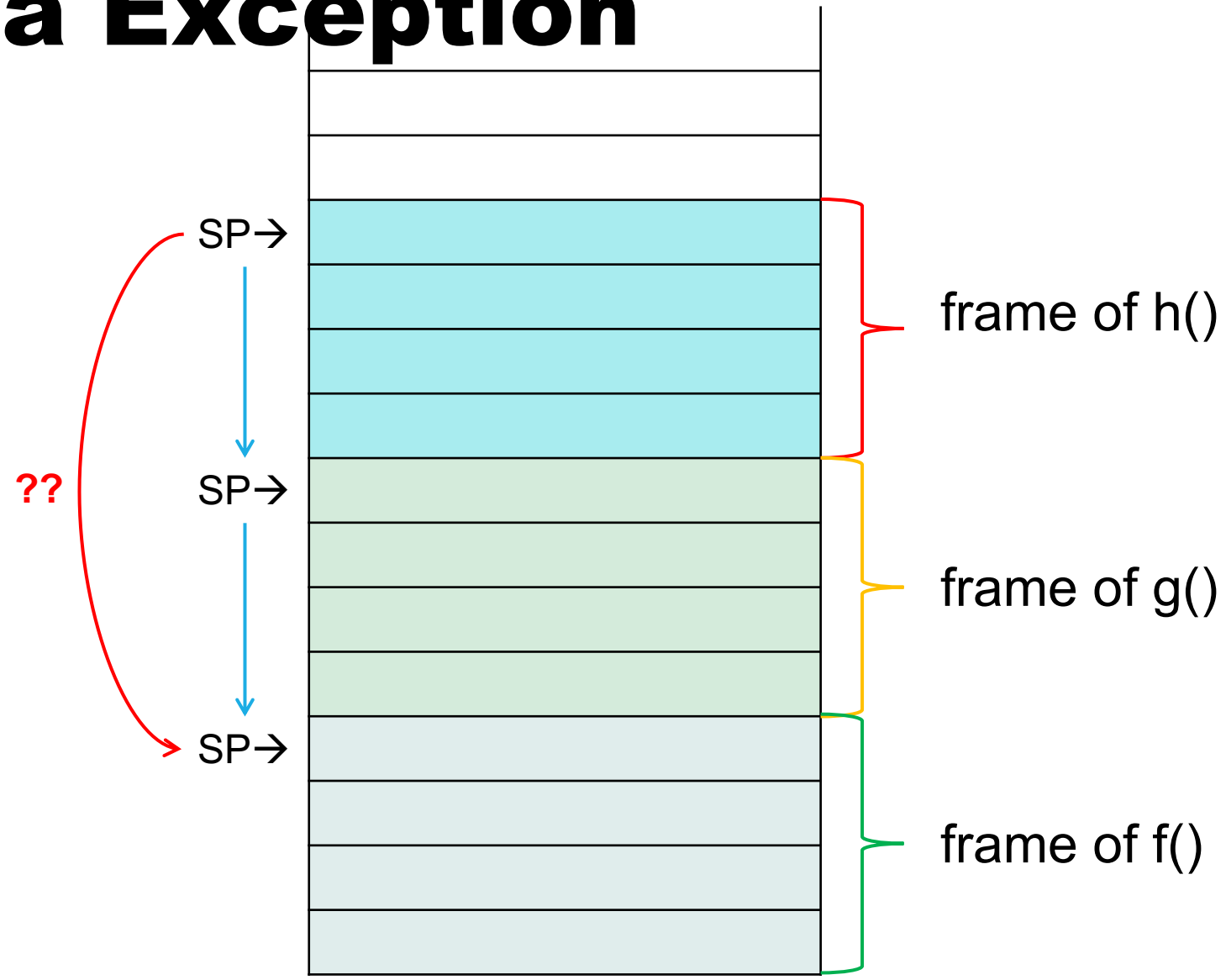


Long Jump via Exception



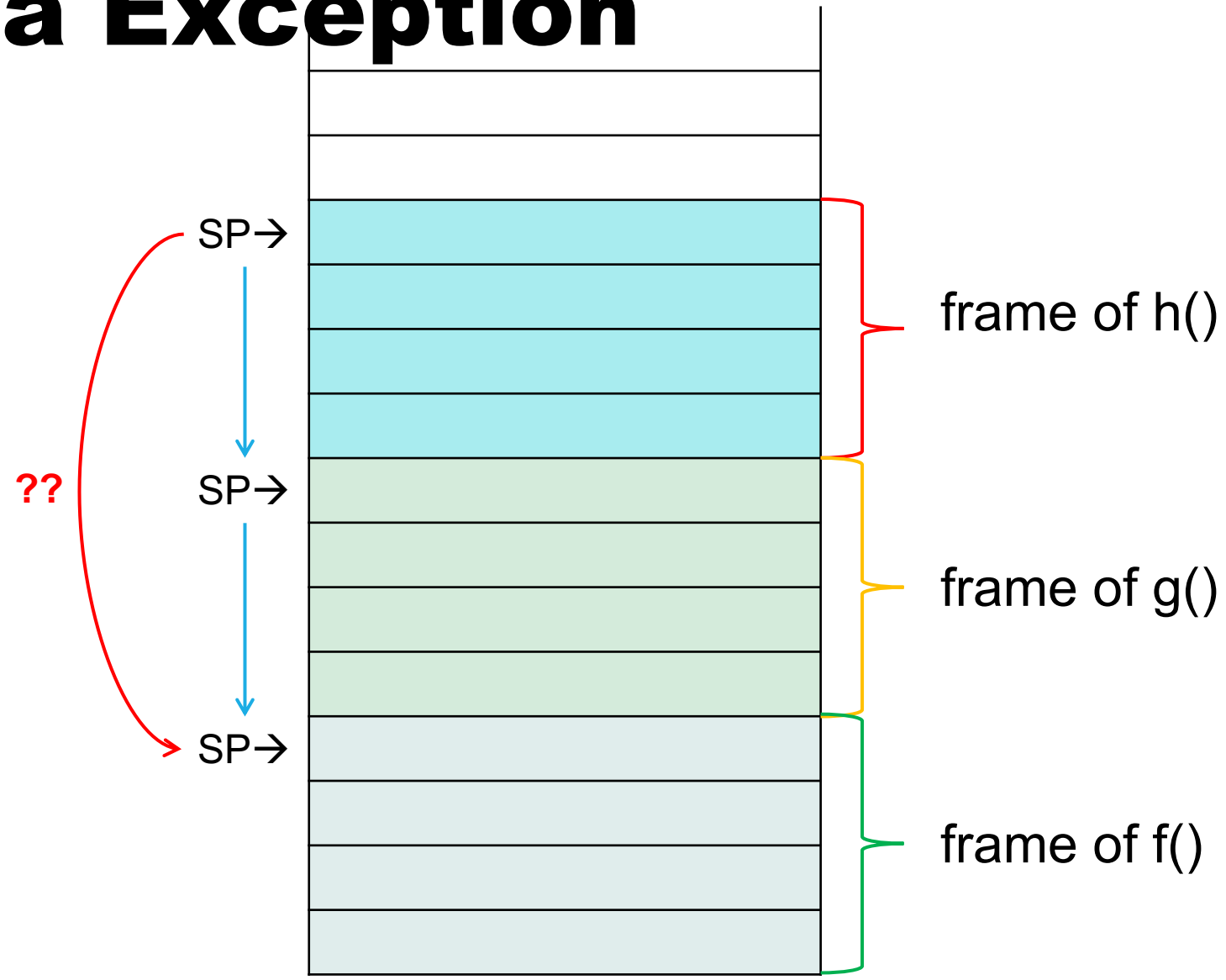
Long Jump via Exception

```
void h() throws Exception {
    throw new Exception();
}

void g () throws Exception {
    h();
}

void f() {
    try { g(); }
    catch (…) { … }
}
```



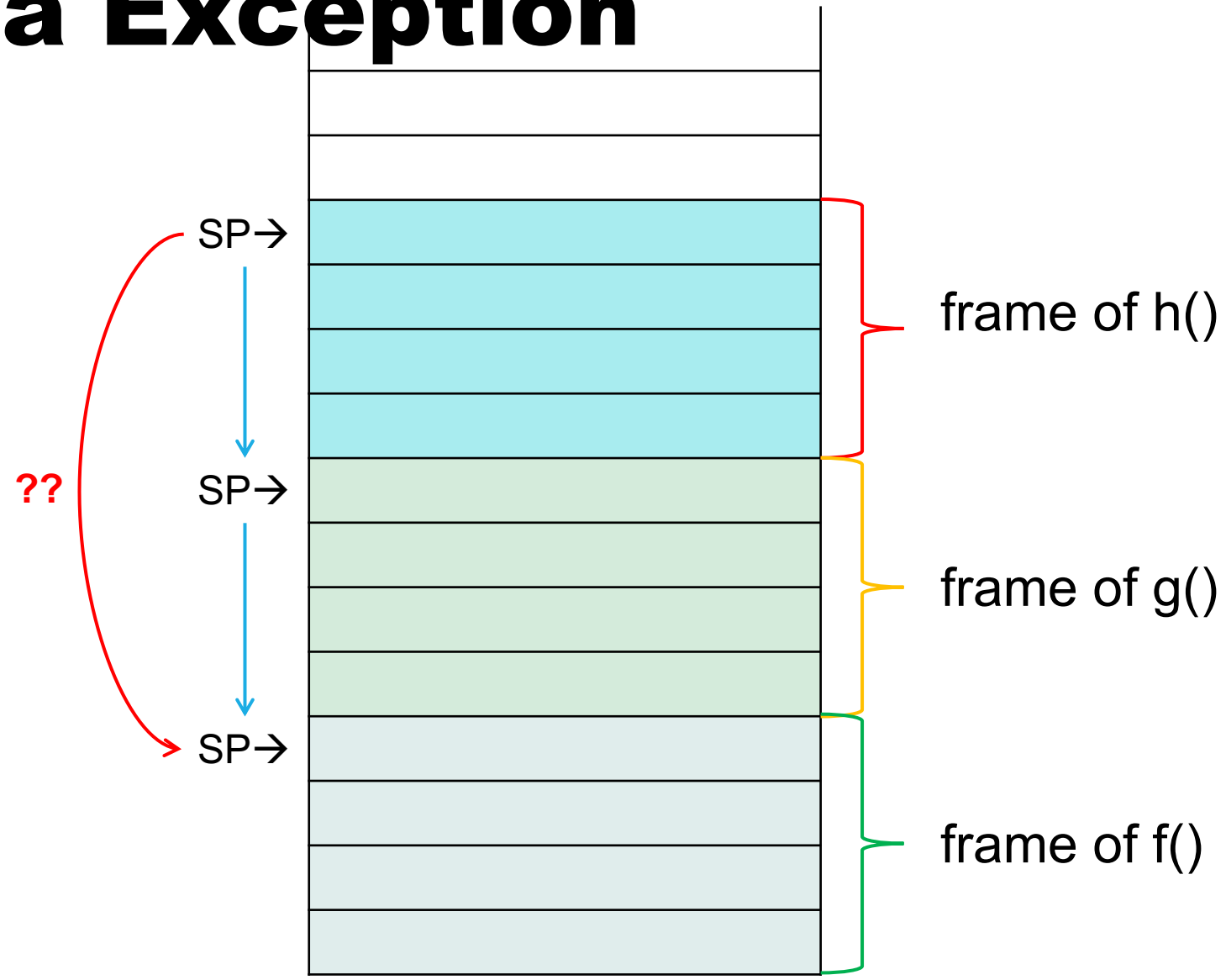
Long Jump via Exception

```
void h() throws Exception {
    throw new Exception();
}

void g () throws Exception {
    h();
}

void f() {
    try { g(); }
    catch (…) { … }
}
```



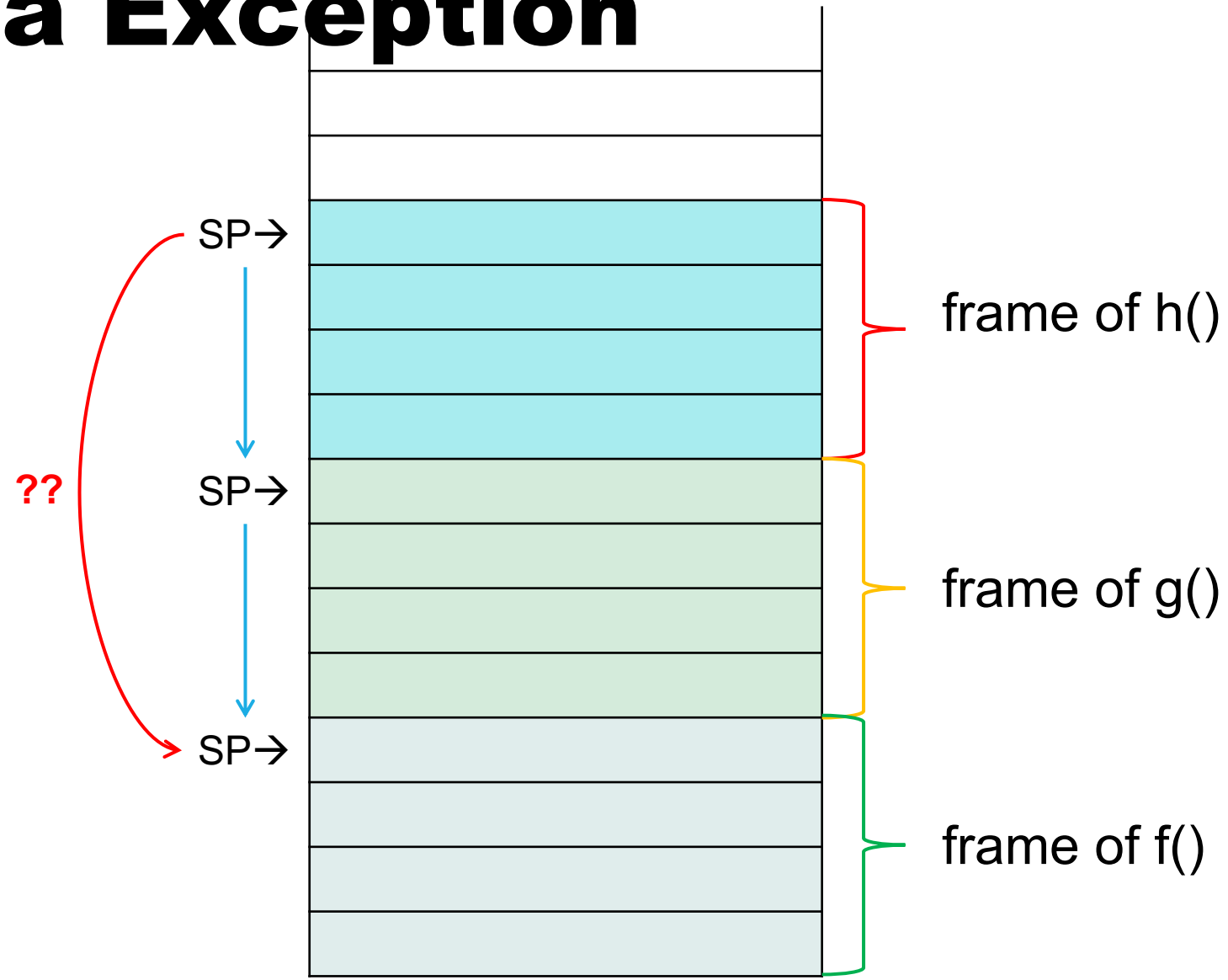
Long Jump via Exception

```
void h() throws Exception {
    throw new Exception();
}

void g () throws Exception {
    h();
}

void f() {
    try { g(); }
    catch (…) { … }
}
```



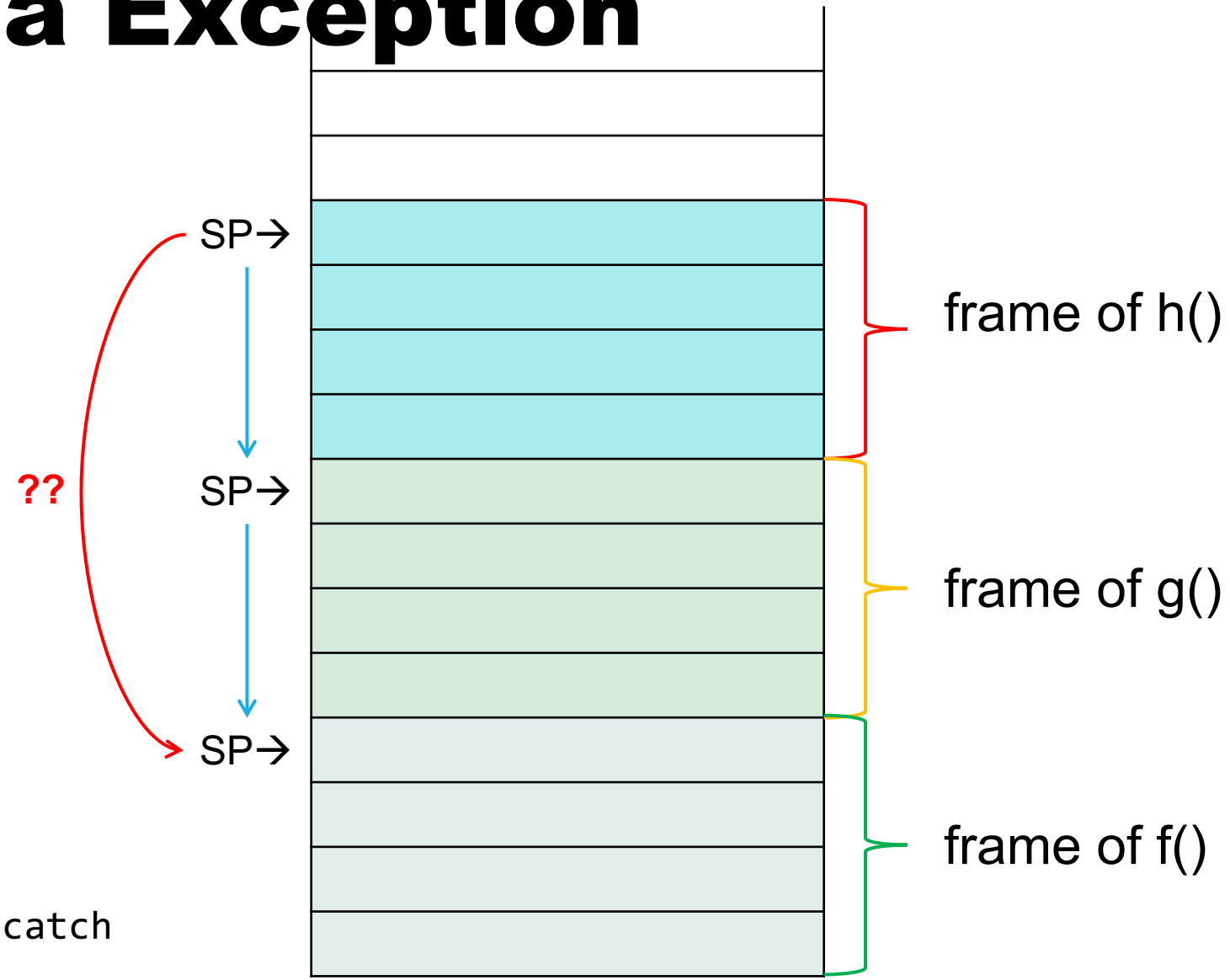
Long Jump via Exception

```
void h() throws Exception {
    throw new Exception();
}

void g () throws Exception {
    h();
}

void f() {
    try { g(); }
    catch (…) { …}
}
```



Long Jump via Exception

```
void h() throws Exception {
    throw new Exception();
}

void g () throws Exception {
    h();
}

void f() {
    try { g(); }
    catch (…) { …}
}
```



Code generation:

- **try:** record the address of catch
- **throw:** jump to the address of catch

Heap Memory

- Dynamically allocate memory in heap
  - C: **void *malloc(size_t)**
  - Java/C++: the **new** operator

Heap Memory

- Dynamically allocate memory in heap
  - C: **void *malloc(size_t)**
  - Java/C++: the **new** operator
- Focus on the efficiency and minimizing segmentation

Heap Memory

- Dynamically allocate memory in heap
  - C: **void *malloc(size_t)**
  - Java/C++: the **new** operator
- Focus on the efficiency and minimizing segmentation
- Manual management (C/C++) or garbage collection (Java)

Heap Memory

- Dynamically allocate memory in heap
  - C: **void *malloc(size_t)**
  - Java/C++: the **new** operator
- Focus on the efficiency and minimizing segmentation
- Manual management (C/C++) or garbage collection (Java)

- ***Refer to Chapter 7, the Dragon book.***

Summary

- Generate target code from three-address code containing
  - basic instructions
  - global variables
  - local variables
  - functions
  - function calls
  - …

PART II-1: In-Block Optimization

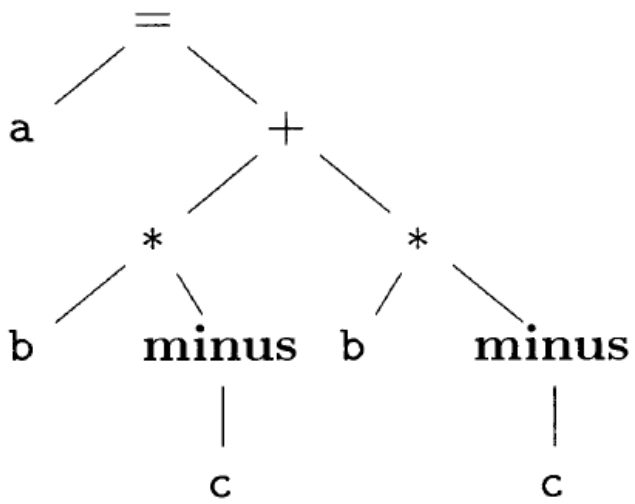
Recap: Syntax Tree and DAG

- What is the syntax tree and DAG of a = b * (-c) + b * (-c)

Try!

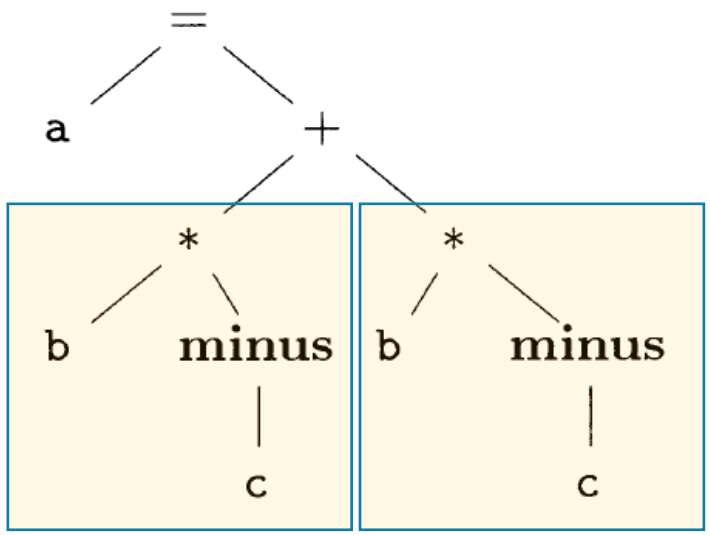
Recap: Syntax Tree and DAG

- What is the syntax tree and DAG of a = b * (-c) + b * (-c)



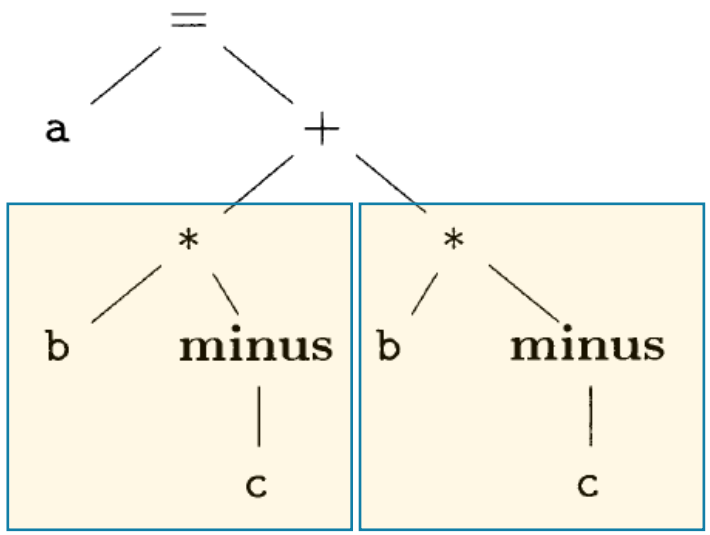
Recap: Syntax Tree and DAG

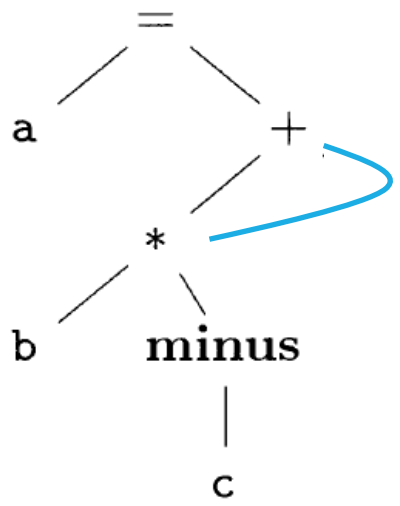
- What is the syntax tree and DAG of a = b * (-c) + b * (-c)



Recap: Syntax Tree and DAG

- What is the syntax tree and DAG of a = b * (-c) + b * (-c)



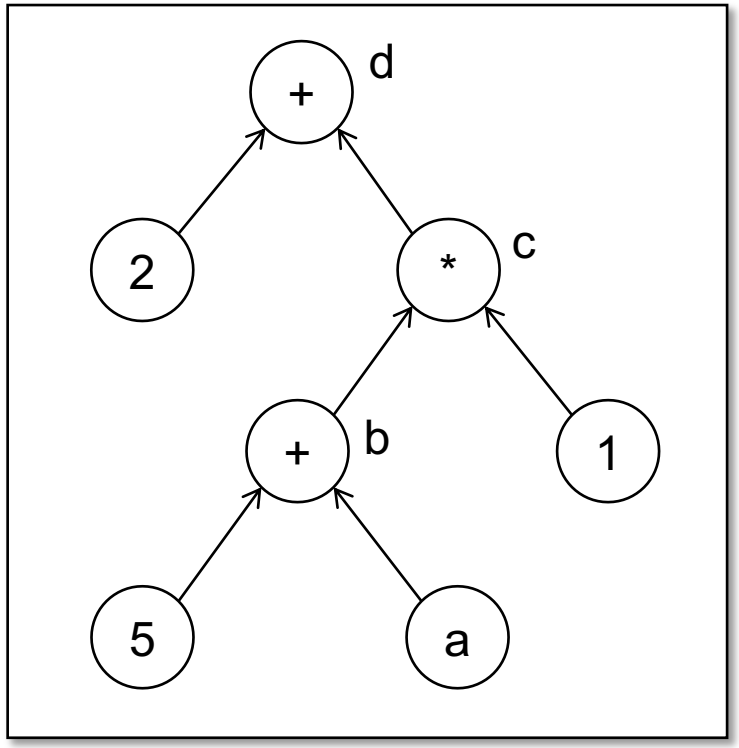


Tree of a Basic Block

```
b = a + 5
c = b * 1
d = 2 + c
```

Tree of a Basic Block

```
b = a + 5
c = b * 1
d = 2 + c
```



Tree-Based Instruction Selection

- Define a series of rules for selecting instructions
  - Pattern → Instructions

Tree-Based Instruction Selection

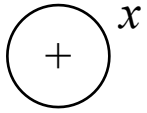
- Define a series of rules for selecting instructions
  - Pattern → Instructions

Patterns	Instructions
 a leaf node	`LD R, x // R is a new register`

Tree-Based Instruction Selection

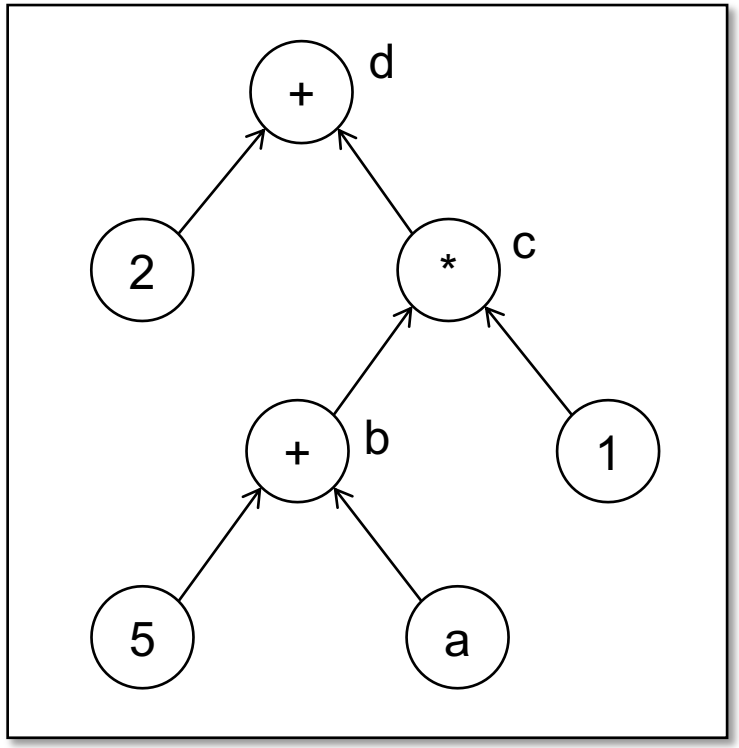
- Define a series of rules for selecting instructions
  - Pattern → Instructions

Patterns	Instructions
 a leaf node	`LD  R,  x  // R is a new register`

 an operator node	`ADD  Rn,  Ro1,  Ro2`
	`ST  x,  Rn`
	`// Rn is a new register;`
	`// Ro are previous registers`

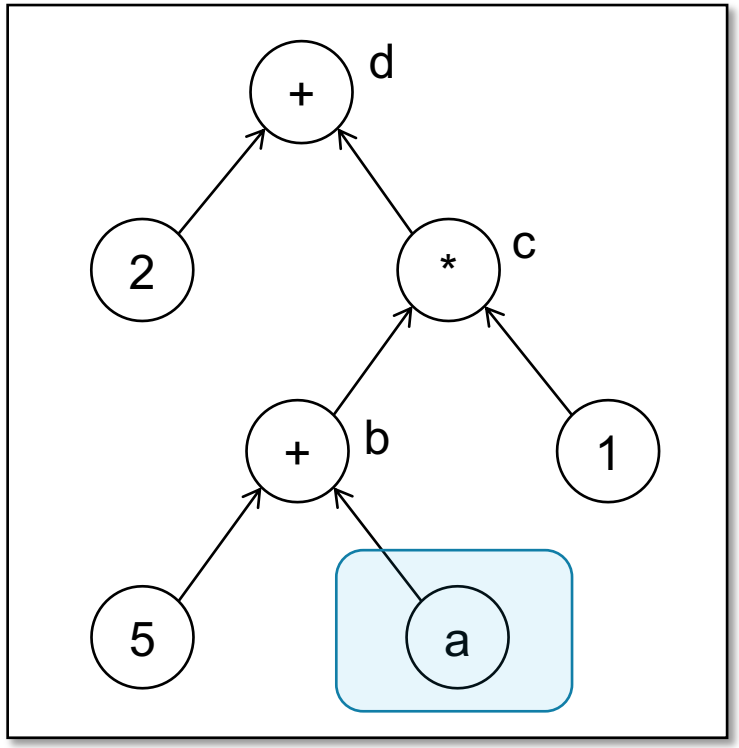
Tree-Based Instruction Selection

```
b = a + 5
c = b * 1
d = 2 + c
```



Tree-Based Instruction Selection

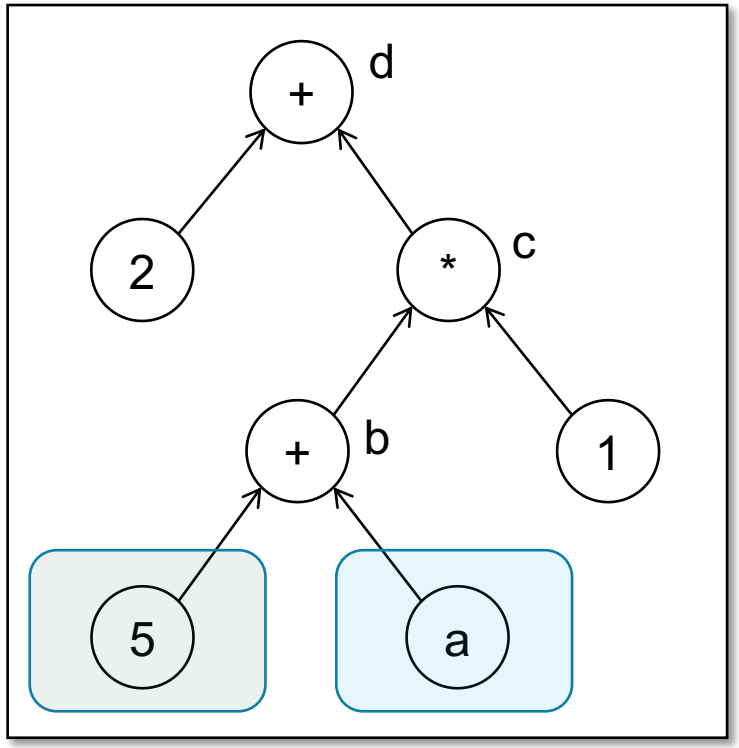
```
b = a + 5
c = b * 1
d = 2 + c
```




```
LD    R0,   a
```

Tree-Based Instruction Selection

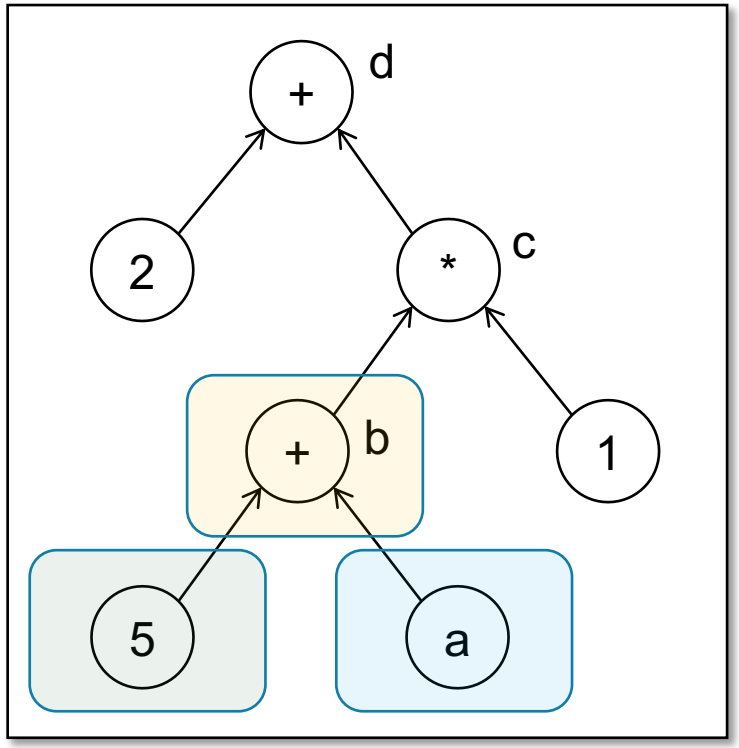
```
b = a + 5
c = b * 1
d = 2 + c
```




```
LD    R0,   a
LD    R1,   5
```

Tree-Based Instruction Selection

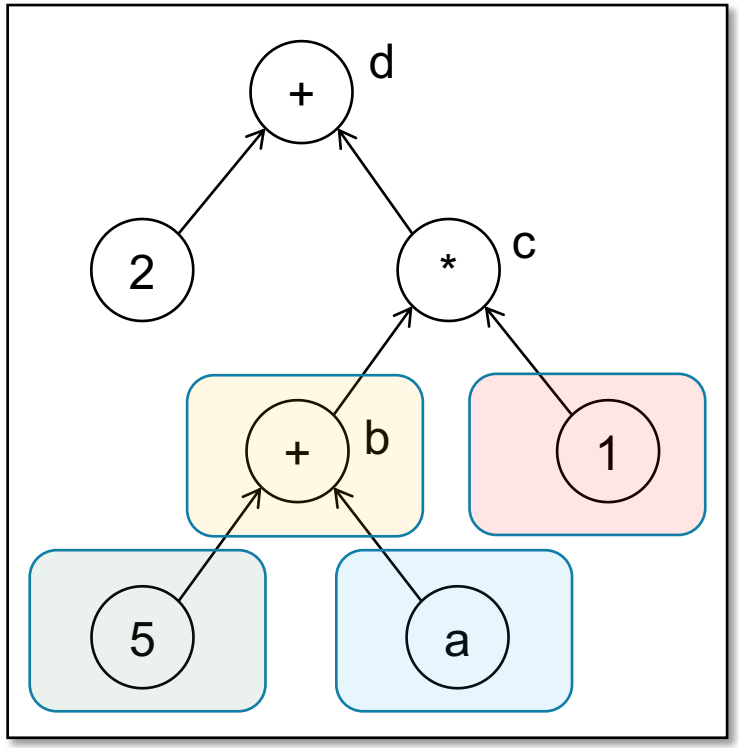
```
b = a + 5
c = b * 1
d = 2 + c
```




```
LD    R0,   a
LD    R1,   5
ADD   R2,   R0,   R1
ST    b     R2
```

Tree-Based Instruction Selection

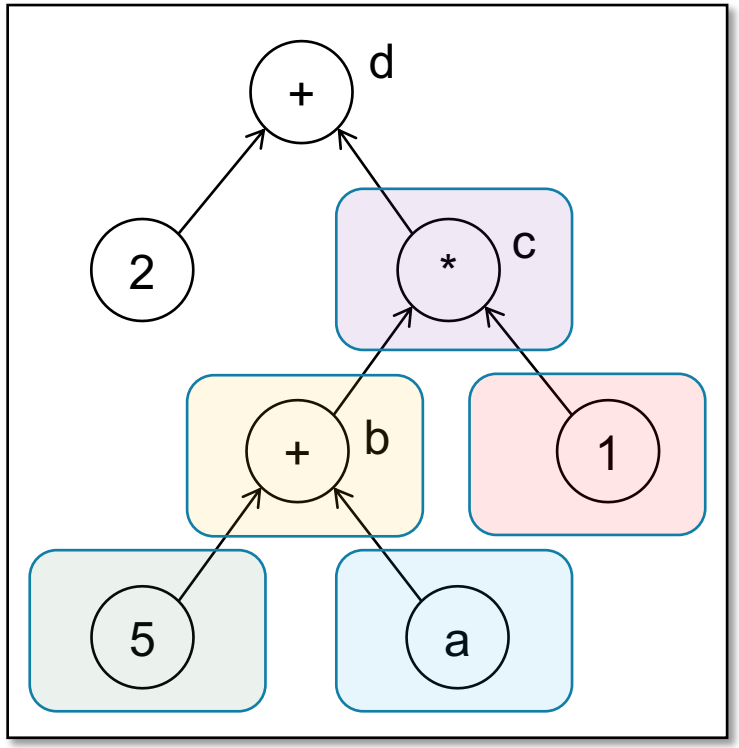
```
b = a + 5
c = b * 1
d = 2 + c
```




```
LD    R0,  a
LD    R1,  5
ADD   R2,  R0,  R1
ST    b    R2
LD    R3,  1
```

Tree-Based Instruction Selection

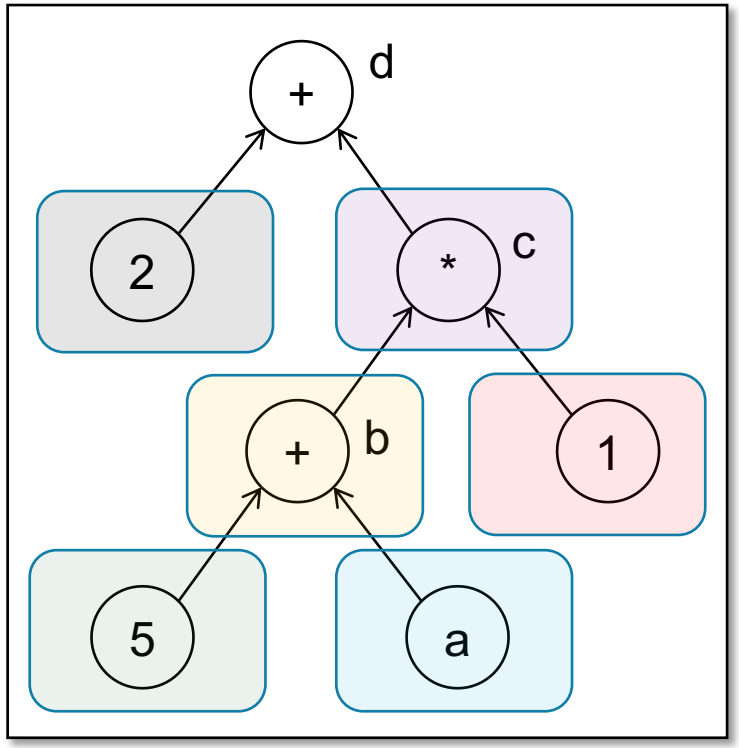
```
b = a + 5
c = b * 1
d = 2 + c
```




```
LD    R0,   a
LD    R1,   5
ADD   R2,   R0,   R1
ST    b     R2
LD    R3,   1
MUL   R4,   R2,   R3
ST    c     R4
```

Tree-Based Instruction Selection

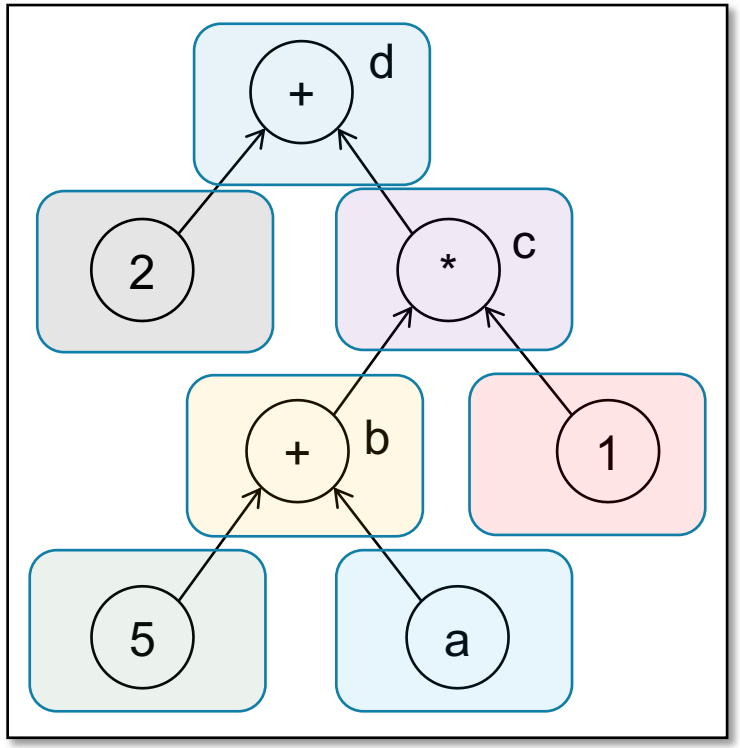
```
b = a + 5
c = b * 1
d = 2 + c
```




```
LD    R0,  a
LD    R1,  5
ADD   R2,  R0,  R1
ST    b    R2
LD    R3,  1
MUL   R4,  R2,  R3
ST    c    R4
LD    R5,  2
```

Tree-Based Instruction Selection

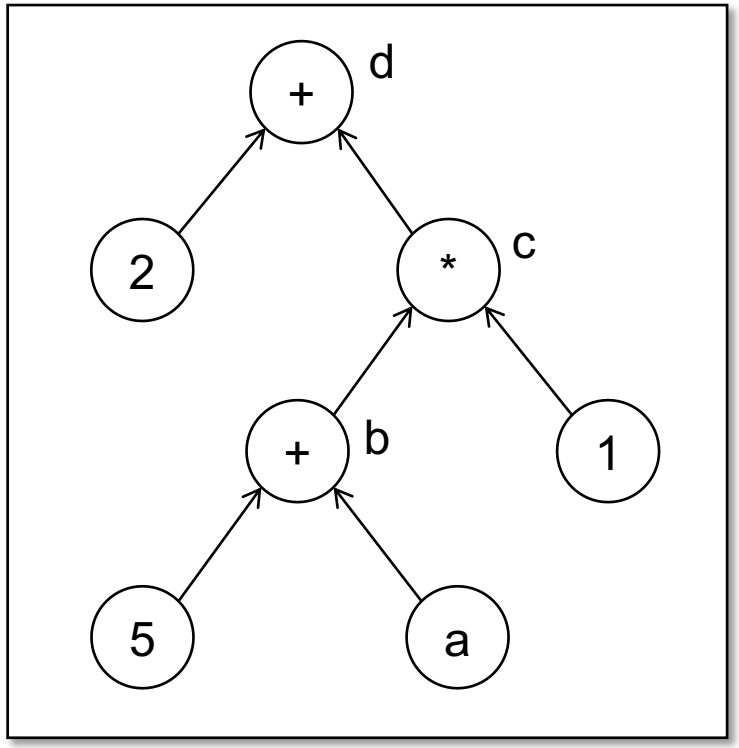
```
b = a + 5
c = b * 1
d = 2 + c
```




```
LD    R0,   a
LD    R1,   5
ADD   R2,   R0,   R1
ST    b     R2
LD    R3,   1
MUL   R4,   R2,   R3
ST    c     R4
LD    R5,   2
ADD   R6,   R4,   R5
ST    d,    R6
```

Tree-Based Instruction Selection

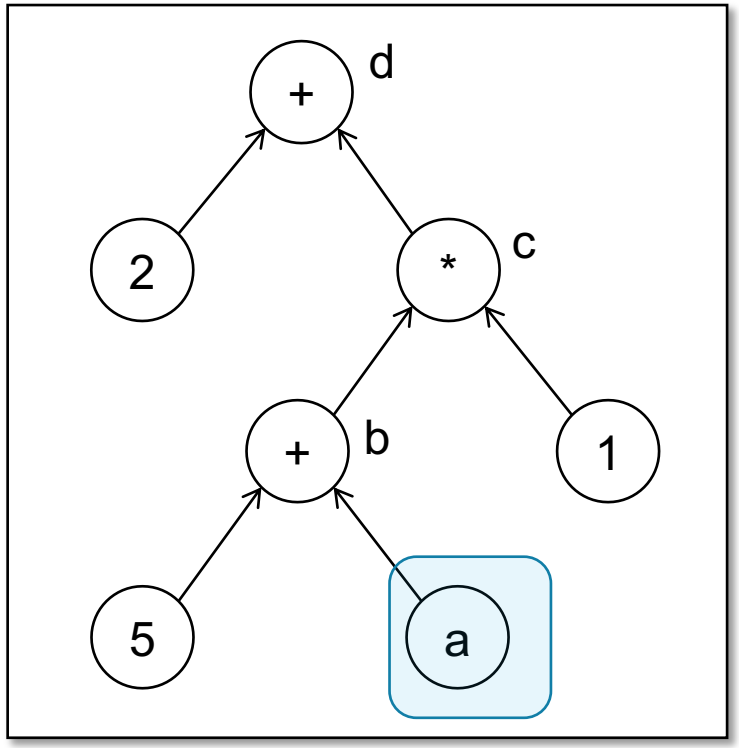
```
b = a + 5
c = b * 1
d = 2 + c
```




```
LD    R0,   a
LD    R1,   5
ADD   R2,   R0,   R1
ST    b     R2
LD    R3,   1
MUL   R4,   R2,   R3
ST    c     R4
LD    R5,   2
ADD   R6,   R4,   R5
ST    d,    R6
```

Tree-Based Instruction Selection

```
b = a + 5
c = b * 1
d = 2 + c
```

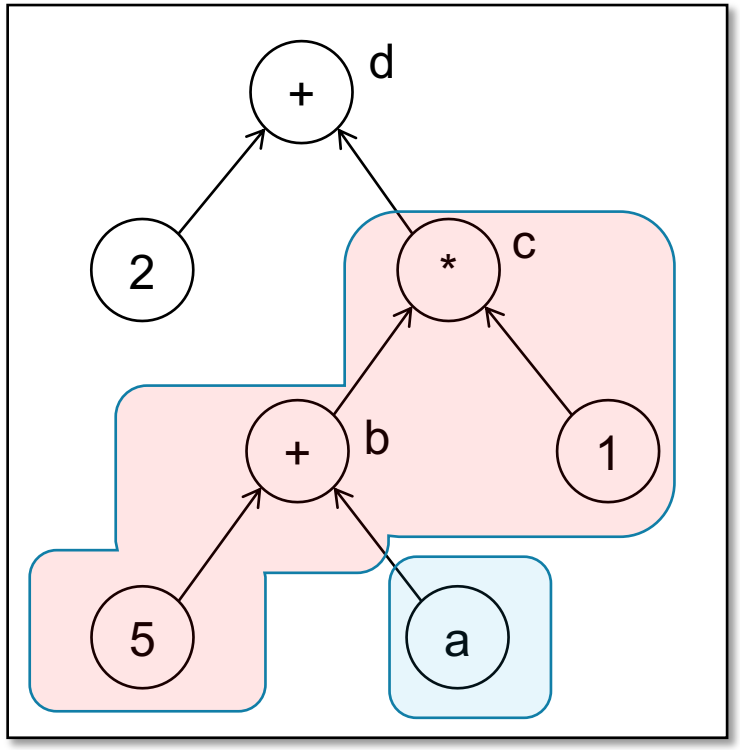



```
LD   R0,  a
LD   R1,  5
ADD  R2,  R0,  R1
ST   b    R2
LD   R3,  1
MUL  R4,  R2,  R3
ST   c    R4
LD   R5,  2
ADD  R6,  R4,  R5
ST   d,   R6
```

```
LD   R0,  a
```

Tree-Based Instruction Selection

```
b = a + 5
c = b * 1
d = 2 + c
```

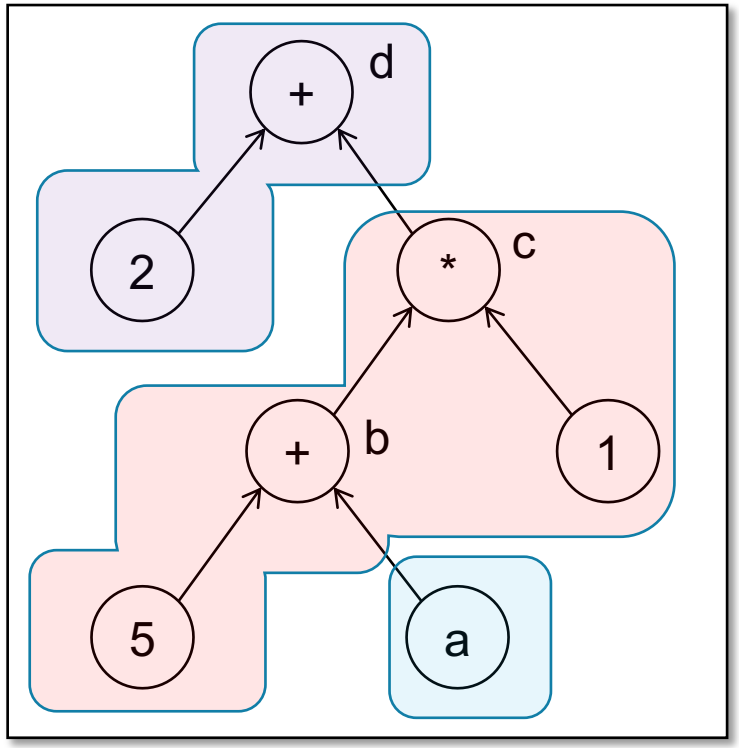



```
LD   R0,  a
LD   R1,  5
ADD  R2,  R0,  R1
ST   b    R2
LD   R3,  1
MUL  R4,  R2,  R3
ST   c    R4
LD   R5,  2
ADD  R6,  R4,  R5
ST   d,   R6
```

```
LD   R0,  a
ADD  R1,  R0,  5
ST   b,   R1
ST   c,   R1
```

Tree-Based Instruction Selection

```
b = a + 5
c = b * 1
d = 2 + c
```




```
LD    R0,  a
LD    R1,  5
ADD   R2,  R0,  R1
ST    b    R2
LD    R3,  1
MUL   R4,  R2,  R3
ST    c    R4
LD    R5,  2
ADD   R6,  R4,  R5
ST    d,   R6
```

```
LD    R0,  a
ADD   R1,  R0,  5
ST    b,   R1
ST    c,   R1
ADD   R2,  R1,  2
ST    d,   R2
```

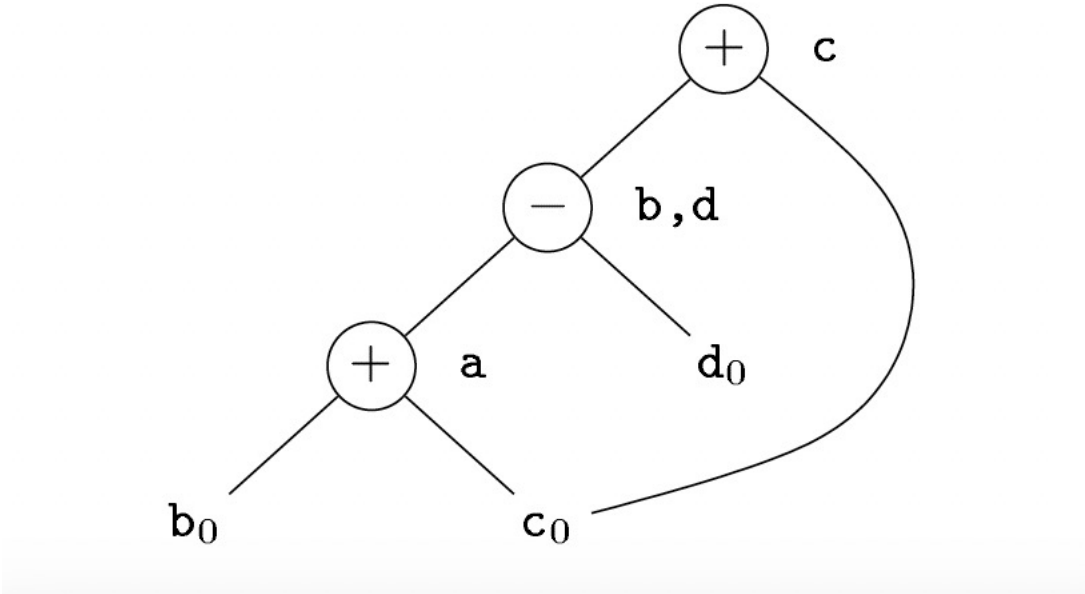
Tree-Based Instruction Selection

- Define a series of rules for selecting instructions
  - Pattern → Instructions

- **The effectiveness depends on the rules predefined.**

From Tree to DAG

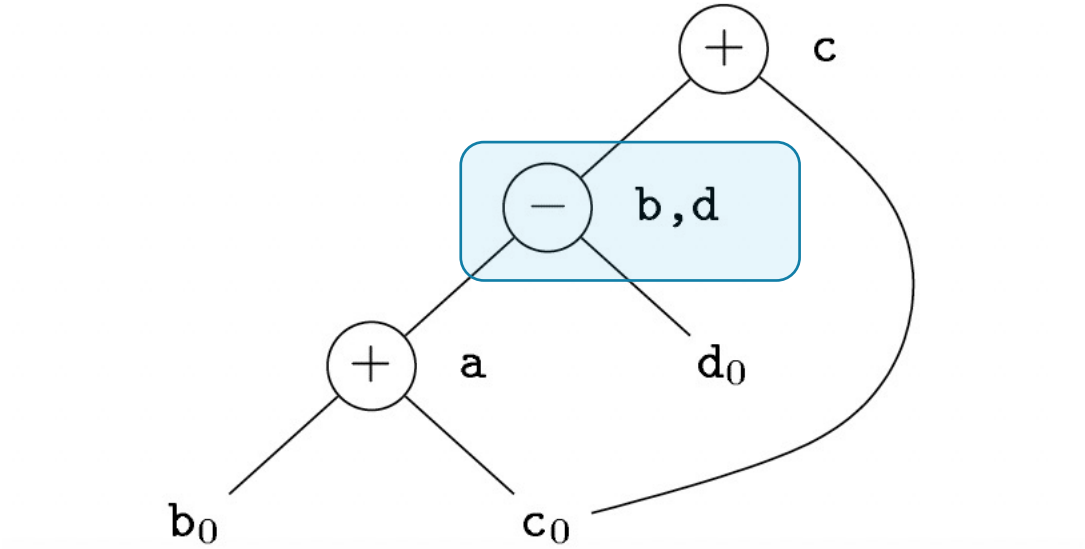
```
a = b + c
b = a - d
c = b + c
d = a - d
```



Local Common Sub-Expression

```
a = b + c
b = a - d
c = b + c
d = a - d
```

It can be replaced by d = b.



Local Common Sub-Expression

- a = x – y
- if x ≥ y goto L

Local Common Sub-Expression

- a = x – y
- if x ≥ y goto L
- ==========>
- a = x – y
- if x - y ≥ 0 goto L

Local Common Sub-Expression

- a = x – y
- if x ≥ y goto L
- ==========>
- a = x – y
- if x - y ≥ 0 goto L
- ==========>
- a = x – y
- if a ≥ 0 goto L

Local Common Sub-Expression

- a = x – y
- if x ≥ y goto L
- ==========>
- a = x – y
- if x - y ≥ 0 goto L
- ==========>
- a = x – y
- if a ≥ 0 goto L

Is it always correct?

Algebraic Identities

Algebraic Identities

$$x + 0 = 0 + x = x \qquad x - 0 = x$$
$$x \times 1 = 1 \times x = x \qquad x/1 = x$$

Algebraic Identities

$$x + 0 = 0 + x = x \qquad x - 0 = x$$
$$x \times 1 = 1 \times x = x \qquad x/1 = x$$

Expensive		Cheaper
x^2	$=$	$x \times x$
$2 \times x$	$=$	$x + x$
$x/2$	$=$	$x \times 0.5$

Algebraic Identities

$$x + 0 = 0 + x = x \qquad x - 0 = x$$
$$x \times 1 = 1 \times x = x \qquad x/1 = x$$

Expensive		Cheaper
x^2	$=$	$x \times x$
$2 \times x$	$=$	$x + x$
$x/2$	$=$	$x \times 0.5$

2 * 3.14 = 6.28

Machine Idioms

```
LD   R0, a        // R0 = a
ADD  R0, R0, #1   // R0 = R0 + 1
ST   a, R0        // a = R0
```

Possible target code for a = a + 1

How about "INC a" ??

Machine Idioms

```
LD  R0, a        // R0 = a
ADD R0, R0, #1   // R0 = R0 + 1
ST  a, R0        // a = R0
```

Possible target code for a = a + 1

How about "INC a" ??

Redundant Loads/Stores

```
LD R0, a
ST a, R0
```

PART II-2: Peephole Optimization

Peephole Optimization

```
1)   i = 1
2)   j = 1
3)   t1 = 10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

- Optimize the code in a small window

Peephole Optimization

```
 1)   i = 1
 2)   j = 1
 3)   t1 = 10 * i
 4)   t2 = t1 + j
 5)   t3 = 8 * t2
 6)   t4 = t3 - 88
 7)   a[t4] = 0.0
 8)   j = j + 1
 9)   if j <= 10 goto (3)
10)   i = i + 1
11)   if i <= 10 goto (2)
12)   i = 1
13)   t5 = i - 1
14)   t6 = 88 * t5
15)   a[t6] = 1.0
16)   i = i + 1
17)   if i <= 10 goto (13)
```

- Optimize the code in a small window

Peephole Optimization

```
1)   i = 1
2)   j = 1
3)   t1 = 10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

- Optimize the code in a small window

Peephole Optimization

```
1)   i = 1
2)   j = 1
3)   t1 = 10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

- Optimize the code in a small window

Peephole Optimization

```
1)   i = 1
2)   j = 1
3)   t1 = 10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

- Optimize the code in a small window

Peephole Optimization

```
1)   i = 1
2)   j = 1
3)   t1 = 10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

- Optimize the code in a small window

Peephole Optimization

```
1)   i = 1
2)   j = 1
3)   t1 = 10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

- Optimize the code in a small window

Peephole Optimization

```
1)   i = 1
2)   j = 1
3)   t1 = 10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

- Optimize the code in a small window

Peephole Optimization

```
1)   i = 1
2)   j = 1
3)   t1 = 10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

- Optimize the code in a small window

Peephole Optimization

```
1)   i = 1
2)   j = 1
3)   t1 = 10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

- Optimize the code in a small window

Peephole Optimization

```
1)   i = 1
2)   j = 1
3)   t1 = 10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

- Optimize the code in a small window
- Work on IR or the target code
- **Cross-block optimization**

Peephole Optimization

```
    if 0 != 1 goto L2
    print debugging information
L2:
```

Peephole Optimization

```
    if 0 != 1 goto L2
    print debugging information
L2:
```

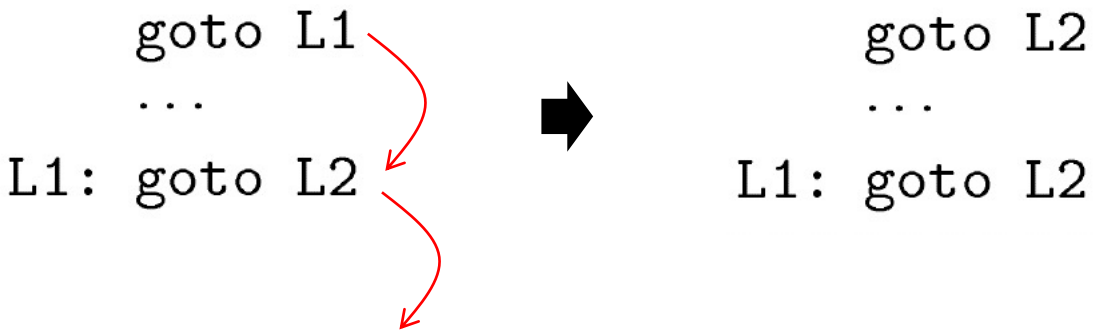
Peephole Optimization

```
    if 0 != 1 goto L2
    print debugging information
L2:
```

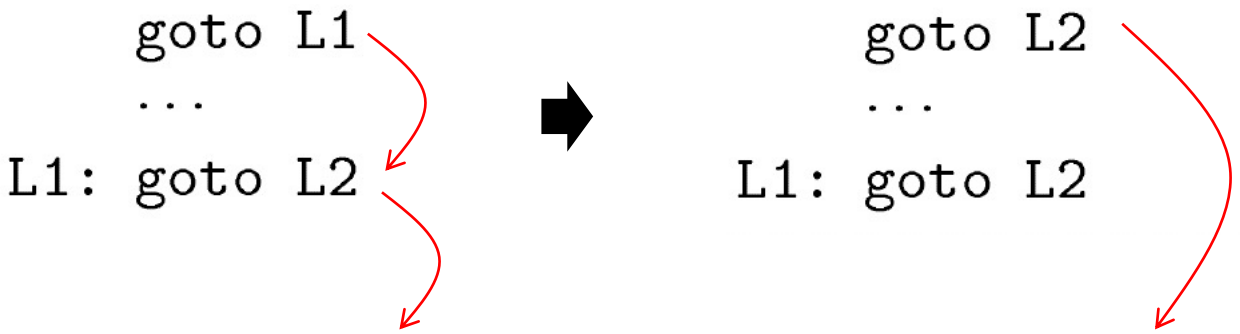
Peephole Optimization

```
    goto L1                    goto L2
    ...                        ...
L1: goto L2                L1: goto L2
```

Peephole Optimization



Peephole Optimization

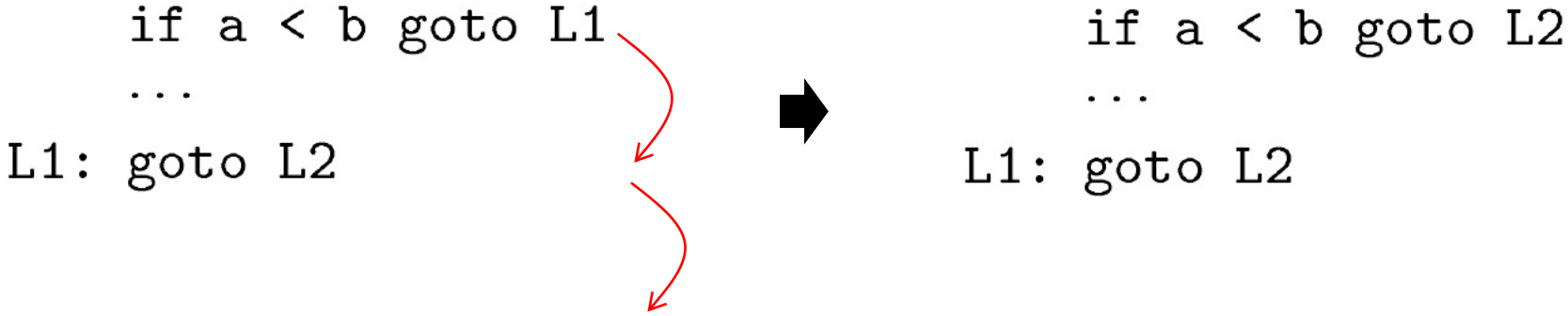


Peephole Optimization

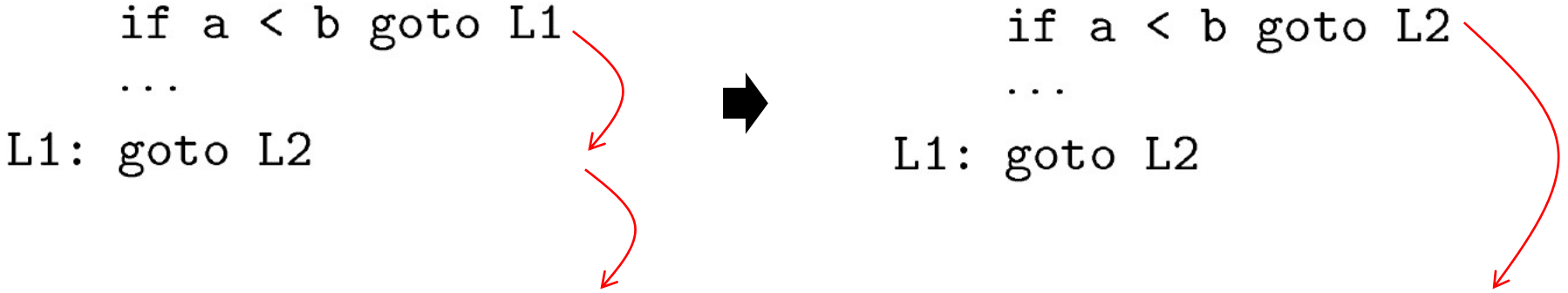
```
    if a < b goto L1
    ...
L1: goto L2
```

```
    if a < b goto L2
    ...
L1: goto L2
```

Peephole Optimization



Peephole Optimization



Peephole Optimization

```
    if debug == 1 goto L1
    goto L2
L1: print debugging information
L2:
```

```
    if debug != 1 goto L2
    print debugging information
L2:
```

Peephole Optimization

We can define many many such rules for peephole optimization!

Peephole Optimization

```
LD   R0, #2
LD   R1, x
LD   R2, #12
ADD  R3, R1, R2
LD   R4, *R3
MUL  R5, R0, R4
LD   R6, #16
ADD  R7, R1, R6
LD   R8, *R7
LD   R9, *R8
SUB  R10, R9, R5
ST   y, R10
```

Peephole Optimization

```
LD   R0, #2
LD   R1, x
LD   R2, #12
ADD  R3, R1, R2
LD   R4, *R3
MUL  R5, R0, R4
LD   R6, #16
ADD  R7, R1, R6
LD   R8, *R7
LD   R9, *R8
SUB  R10, R9, R5
ST   y, R10
```

- First window
- No optimization available
- Advance the window

Peephole Optimization

```
LD   R0, #2
LD   R1, x
LD   R2, #12
ADD  R3, R1, R2
LD   R4, *R3
MUL  R5, R0, R4
LD   R6, #16
ADD  R7, R1, R6
LD   R8, *R7
LD   R9, *R8
SUB  R10, R9, R5
ST   y, R10
```

Peephole Optimization

LD	R0,	#2	
LD	R1,	x	
LD	R2,	#12	
ADD	R3,	R1,	#12~~R2~~
LD	R4,	*R3	
MUL	R5,	R0,	R4
LD	R6,	#16	
ADD	R7,	R1,	R6
LD	R8,	*R7	
LD	R9,	*R8	
SUB	R10,	R9,	R5
ST	y,	R10	

Peephole Optimization

LD	R0,	#2	
LD	R1,	x	
~~LD~~	~~R2,~~	~~#12~~	
ADD	R3,	R1,	#12~~R2~~
LD	R4,	*R3	
MUL	R5,	R0,	R4
LD	R6,	#16	
ADD	R7,	R1,	R6
LD	R8,	*R7	
LD	R9,	*R8	
SUB	R10,	R9,	R5
ST	y,	R10	

Peephole Optimization

```
LD   R0, #2
LD   R1, x

ADD  R3, R1, #12
LD   R4, *R3
MUL  R5, R0, R4
LD   R6, #16
ADD  R7, R1, R6
LD   R8, *R7
LD   R9, *R8
SUB  R10, R9, R5
ST   y, R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
ADD  R3,  R1,  #12
LD   R4,  *R3
MUL  R5,  R0,  R4
LD   R6,  #16
ADD  R7,  R1,  R6
LD   R8,  *R7
LD   R9,  *R8
SUB  R10, R9,  R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
ADD  R3,  R1,  #12
LD   R4,  12(R1)*R3
MUL  R5,  R0,  R4
LD   R6,  #16
ADD  R7,  R1,  R6
LD   R8,  *R7
LD   R9,  *R8
SUB  R10, R9,  R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
ADD  R3,  R1,  #12
LD   R4,  12(R1)*R3
MUL  R5,  R0,  R4
LD   R6,  #16
ADD  R7,  R1,  R6
LD   R8,  *R7
LD   R9,  *R8
SUB  R10, R9,  R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,   R4
LD   R6,  #16
ADD  R7,  R1,   R6
LD   R8,  *R7
LD   R9,  *R8
SUB  R10, R9,   R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0, #2
LD   R1, x
LD   R4, 12(R1)
MUL  R5, R0,  R4
LD   R6, #16
ADD  R7, R1,  R6
LD   R8, *R7
LD   R9, *R8
SUB  R10, R9,  R5
ST   y,  R10
```

- No more optimization available
- Advance the window

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,   R4
LD   R6,  #16
ADD  R7,  R1,   R6
LD   R8,  *R7
LD   R9,  *R8
SUB  R10, R9,   R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,   R4
LD   R6,  #16
ADD  R7,  R1,   R6
LD   R8,  *R7
LD   R9,  *R8
SUB  R10, R9,   R5
ST   y,   R10
```

- No more optimization available
- Advance the window

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,  R4
LD   R6,  #16
ADD  R7,  R1,  R6
LD   R8,  *R7
LD   R9,  *R8
SUB  R10, R9,  R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,   R4
LD   R6,  #16
ADD  R7,  R1,   #16R6
LD   R8,  *R7
LD   R9,  *R8
SUB  R10, R9,   R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,  R4
LD   R6,  #16
ADD  R7,  R1,  #16R6
LD   R8,  *R7
LD   R9,  *R8
SUB  R10, R9,  R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,  R4
ADD  R7,  R1,  #16
LD   R8,  *R7
LD   R9,  *R8
SUB  R10, R9,  R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,   R4
ADD  R7,  R1,   #16
LD   R8,  *R716(R1)
LD   R9,  *R8
SUB  R10, R9,   R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,   R4
LD   R8,  16(R1)
LD   R9,  *R8
SUB  R10, R9,   R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,   R4
LD   R8,  16(R1)
LD   R9,  *16(R1)R8
SUB  R10, R9,   R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,   R4
LD   R9,  *16(R1)
SUB  R10, R9,   R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,  R4
LD   R9,  *16(R1)
SUB  R10, R9,  R5
ST   y,   R10
```

- No more optimization available
- Advance the window

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,   R4
LD   R9,  *16(R1)
SUB  R10, R9,   R5
ST   y,   R10
```

Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,   R4
LD   R9,  *16(R1)
SUB  R10, R9,   R5
ST   y,   R10
```

- No more optimization available
- Reach the end; Terminate

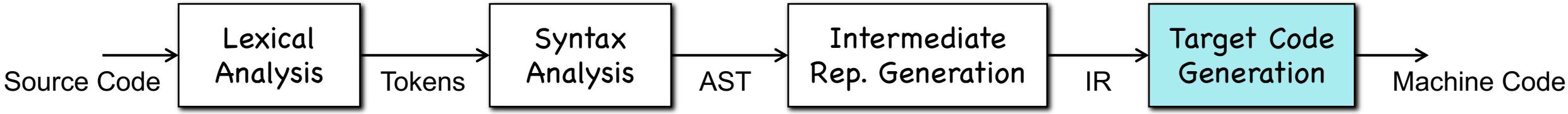
Peephole Optimization

```
LD   R0,  #2
LD   R1,  x
LD   R4,  12(R1)
MUL  R5,  R0,   R4
LD   R9,  *16(R1)
SUB  R10, R9,   R5
ST   y,   R10
```

- No more optimization available
- Reach the end; Terminate

- **In total:**
- Four unnecessary instructions are deleted via the peephole optimization

Summary



- **Code Generation**/Target Code Model
  /Memory Allocation
- **Gen Better Code**/In-Block Optimization
  /Peephole Optimization

THANKS!