Topic V0B

Procedure Calls, Stack, and Memory Layout

Readings: (Section 2.8)

A word about naming…

Callable Unit

Procedure

Function ← “Pure functions” have no side effect

See “functional programming languages”

Routine

Subroutine

Subprogram

Method ← Methods are usually associated with objects

See “object-oriented programming languages”

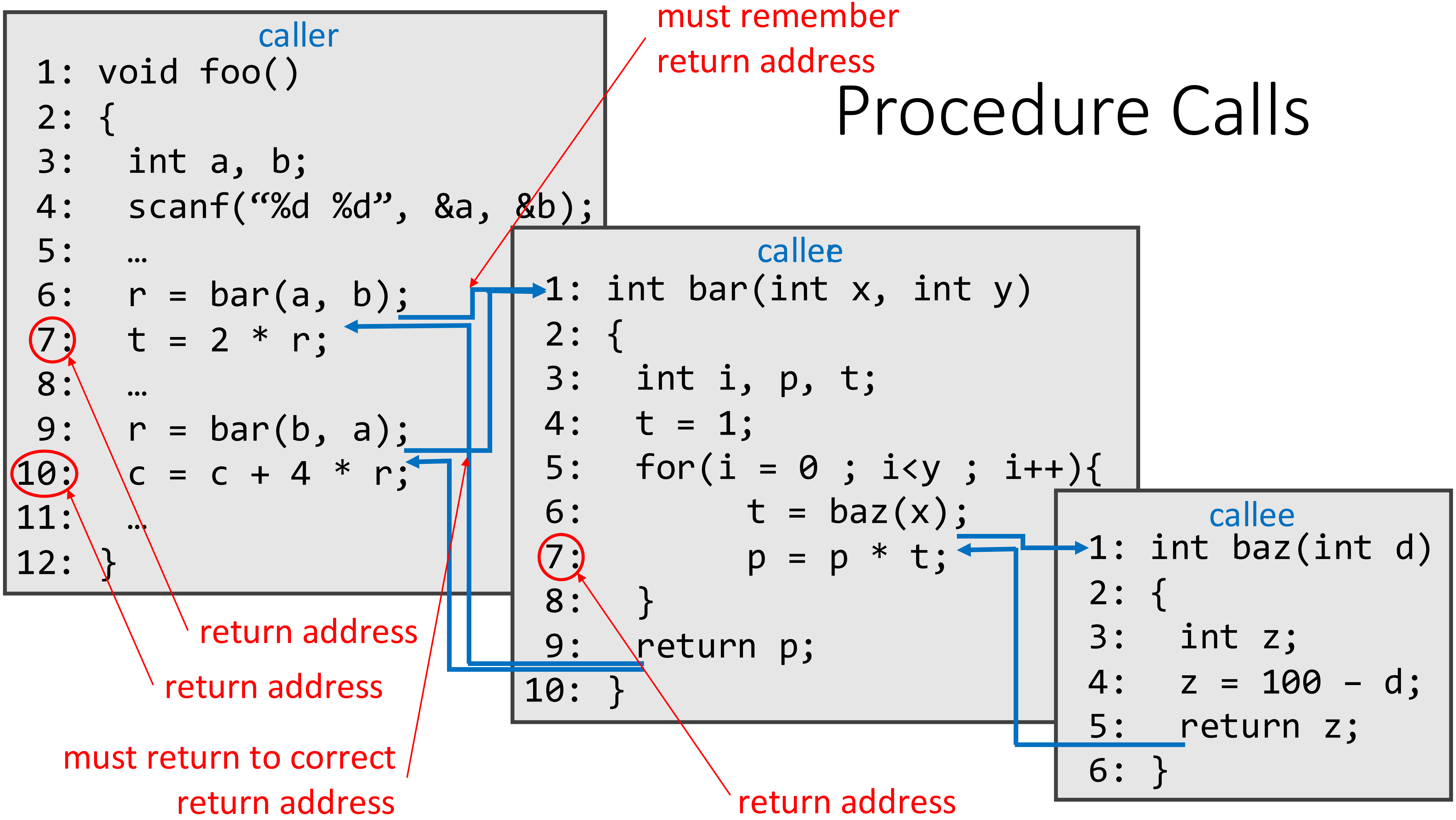
Procedure Calls

```
void foo()
{
  int a, b;
  scanf(“%d %d”, &a, &b);
  …
  r = bar(a, b);
  t = 2 * r;
  …
  r = bar(b, a);
  c = c + 4 * r;
  …
}
```

```
int bar(int x, int y)
{
  int i, p, t;
  t = 1;
  for(i = 0 ; i<y ; i++){
         t = baz(x);
         p = p * t;
  }
  return p;
}
```

```
int baz(int d)
{
  int z;
  z = 100 – d;
  return z;
}
```

Procedure Calls



must remember return address

caller

```
1: void foo()
2: {
3:   int a, b;
4:   scanf(“%d %d”, &a, &b);
5:   …
6:   r = bar(a, b);
7:   t = 2 * r;
8:   …
9:   r = bar(b, a);
10:  c = c + 4 * r;
11:  …
12: }
```

callee

```
1: int bar(int x, int y)
2: {
3:   int i, p, t;
4:   t = 1;
5:   for(i = 0 ; i<y ; i++){
6:        t = baz(x);
7:        p = p * t;
8:   }
9:   return p;
10: }
```

callee

```
1: int baz(int d)
2: {
3:   int z;
4:   z = 100 – d;
5:   return z;
6: }
```

return address

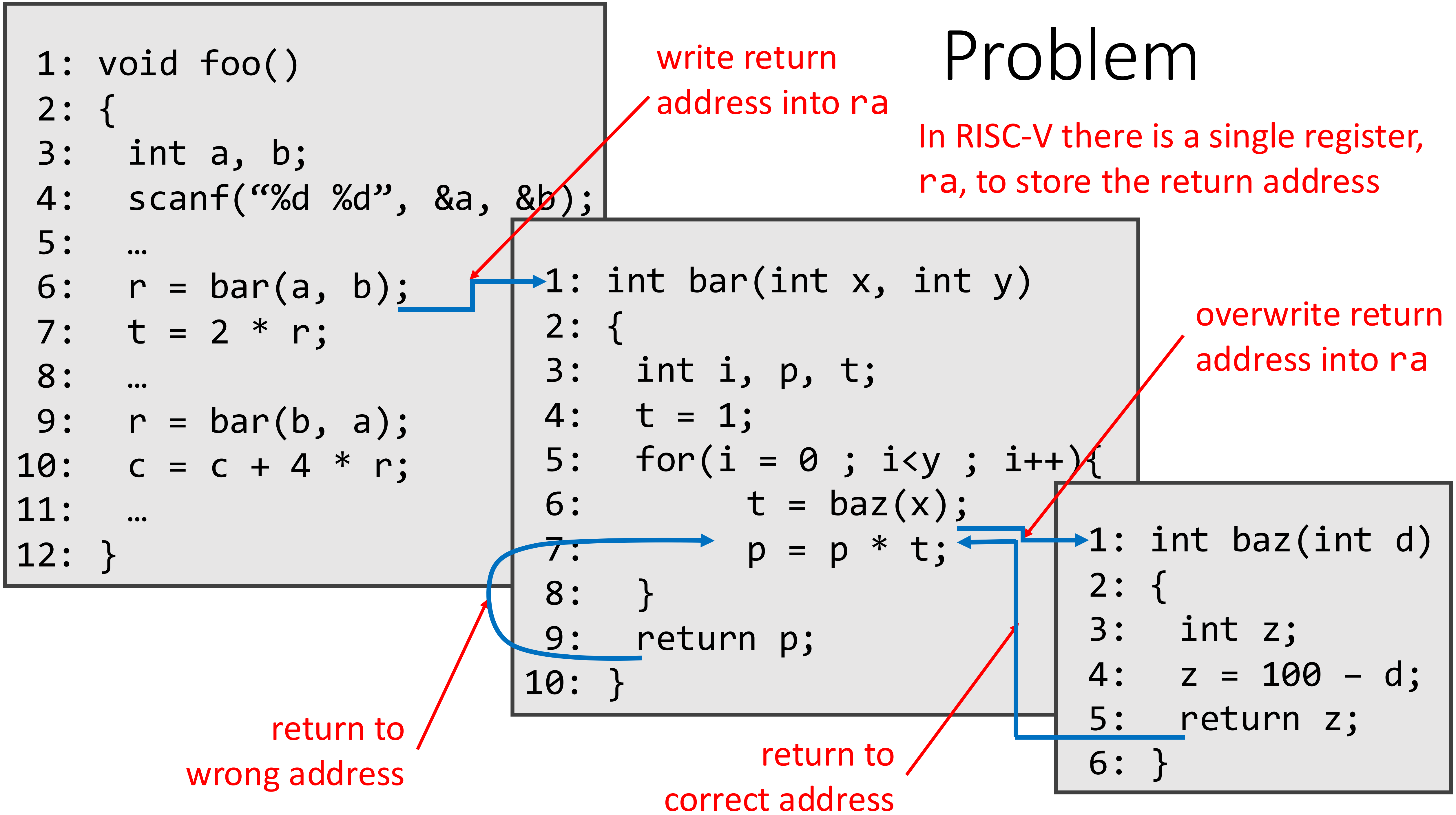
return address

must return to correct return address

return address

Problem

In RISC-V there is a single register, ra, to store the return address



```
void foo()
{
  int a, b;
  scanf(“%d %d”, &a, &b);
  …
  r = bar(a, b);
  t = 2 * r;
  …
  r = bar(b, a);
  c = c + 4 * r;
  …
}
```

```
int bar(int x, int y)
{
  int i, p, t;
  t = 1;
  for(i = 0 ; i<y ; i++){
        t = baz(x);
        p = p * t;
  }
  return p;
}
```

```
int baz(int d)
{
  int z;
  z = 100 – d;
  return z;
}
```

write return address into ra
overwrite return address into ra
return to wrong address
return to correct address

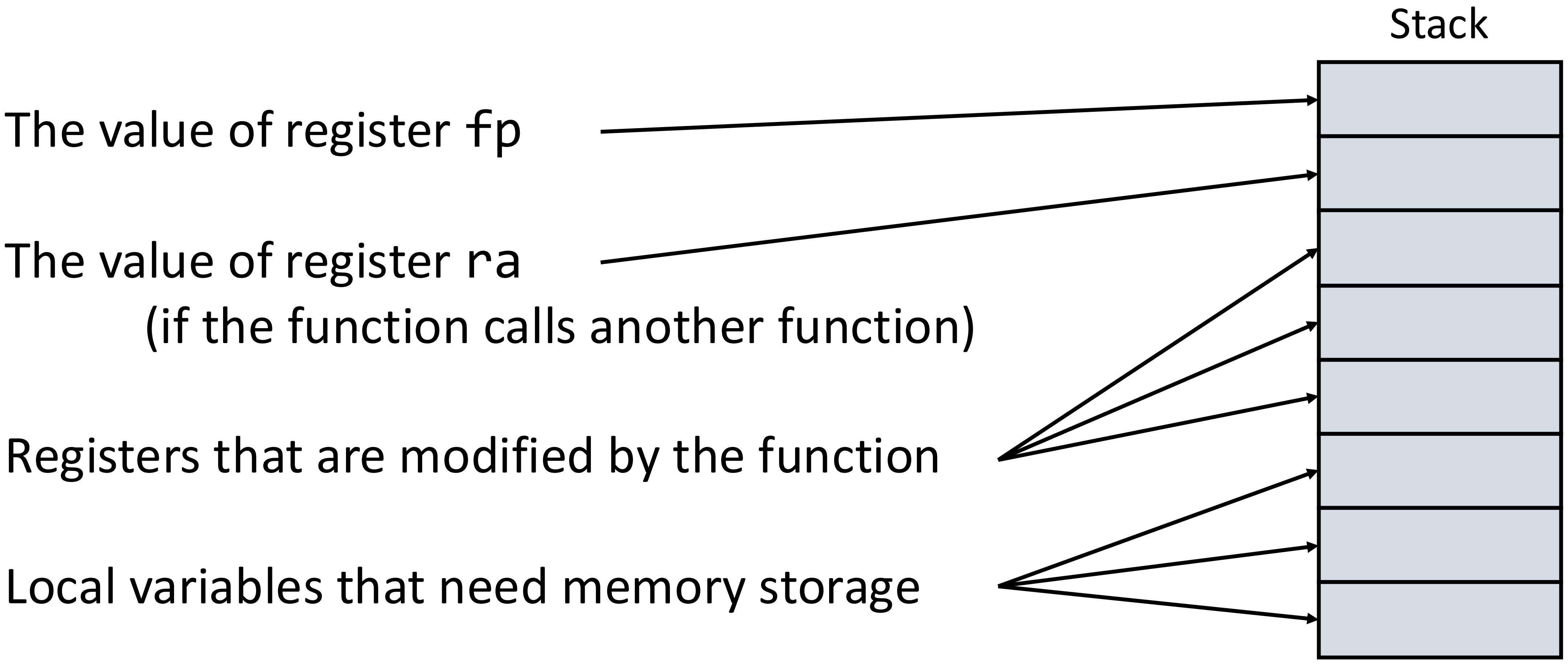
Solution

We need to save the value of `ra` somewhere

Where?

Into a stack

What Goes Into the Stack



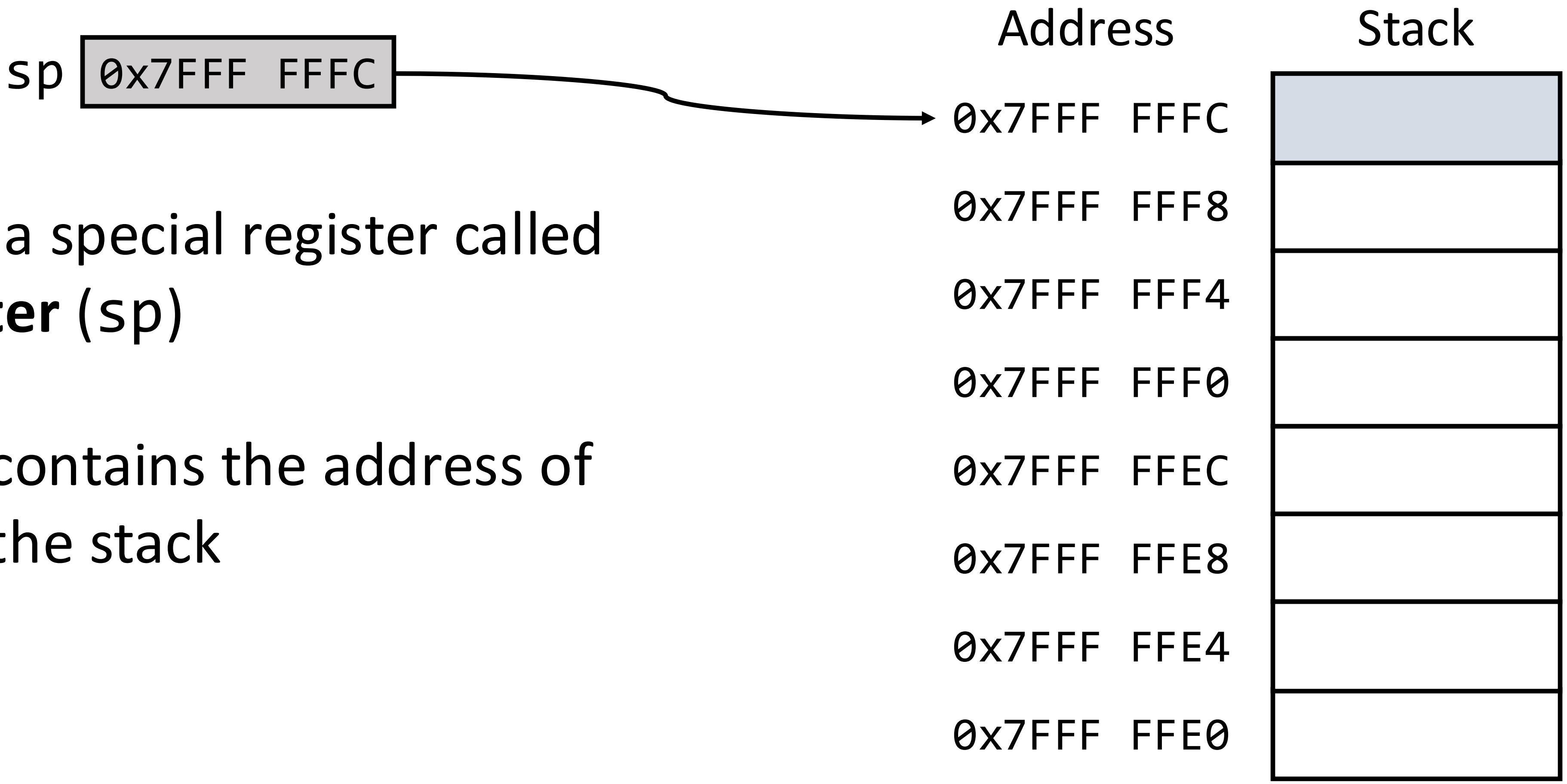
The value of register `fp`

The value of register `ra`
(if the function calls another function)

Registers that are modified by the function

Local variables that need memory storage

Where is the Top of the Stack?



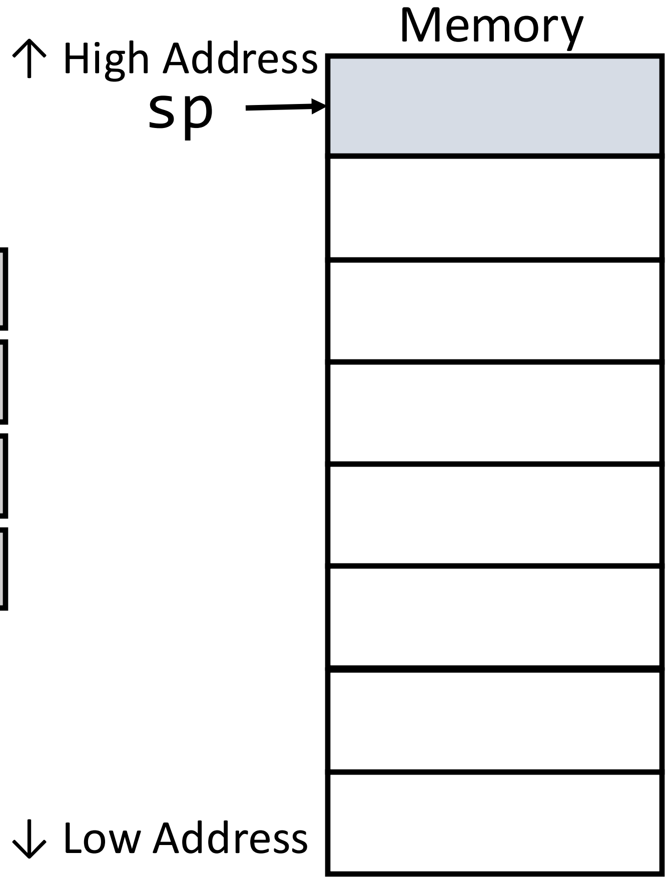
Address	Stack
0x7FFF FFFC	
0x7FFF FFF8	
0x7FFF FFF4	
0x7FFF FFF0	
0x7FFF FFEC	
0x7FFF FFE8	
0x7FFF FFE4	
0x7FFF FFE0	

RISC-V has a special register called **stack pointer** (`sp`)

`sp` always contains the address of the top of the stack

Saving Registers Into the Stack

s1	0x0000 0003
fp	0x7FFF FFFC
sp	0x7FFF FFF8
ra	0x0040 0008



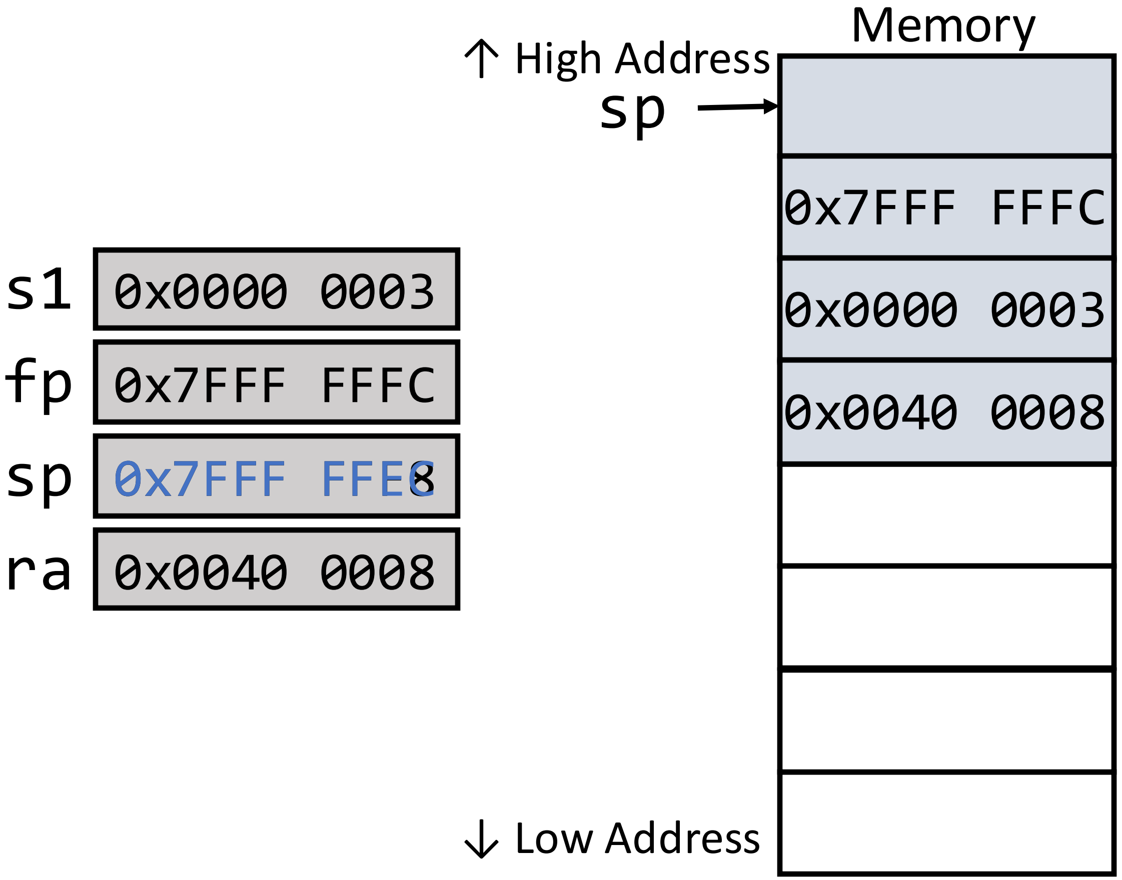
The stack grows towards lower memory addresses

Situation: a procedure will write on `s1` and call another subroutine

Must save values of `s1`, `fp`, and `ra` into the stack

How do we make room in the stack for these three registers?

Saving Registers Into the Stack



We need to decrement the value of sp by 12

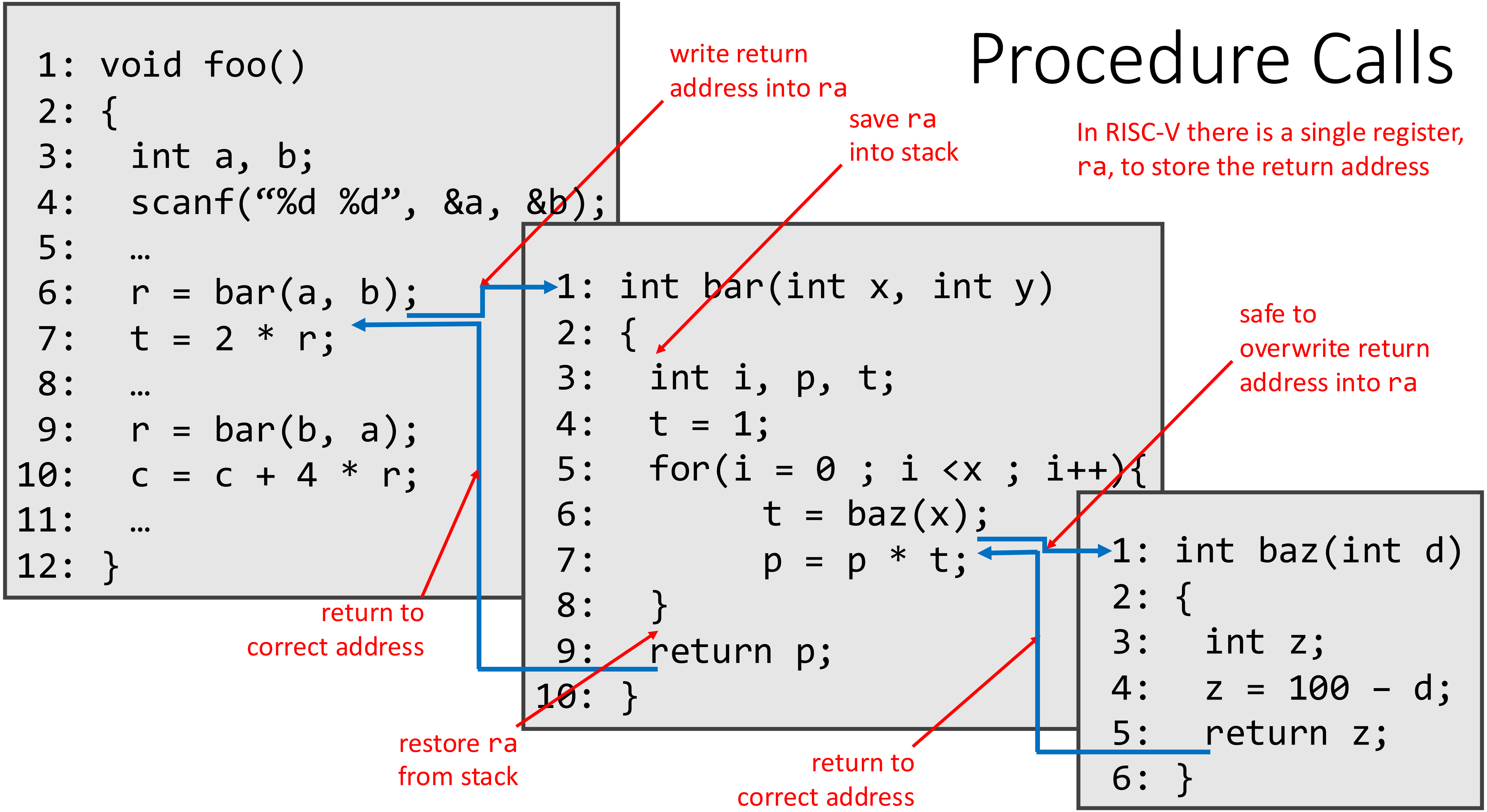
How do we actually ‘save’ `s1`, `fp`, and `ra`?

Write their values, using the `sw` instruction, to the stack:

```
      sw    s1, 4(sp)
```

Procedure Calls

In RISC-V there is a single register, ra, to store the return address



```
void foo()
{
  int a, b;
  scanf(“%d %d”, &a, &b);
  …
  r = bar(a, b);
  t = 2 * r;
  …
  r = bar(b, a);
  c = c + 4 * r;
  …
}
```

```
int bar(int x, int y)
{
  int i, p, t;
  t = 1;
  for(i = 0 ; i <x ; i++){
        t = baz(x);
        p = p * t;
  }
  return p;
}
```

```
int baz(int d)
{
  int z;
  z = 100 – d;
  return z;
}
```

write return address into ra

save ra into stack

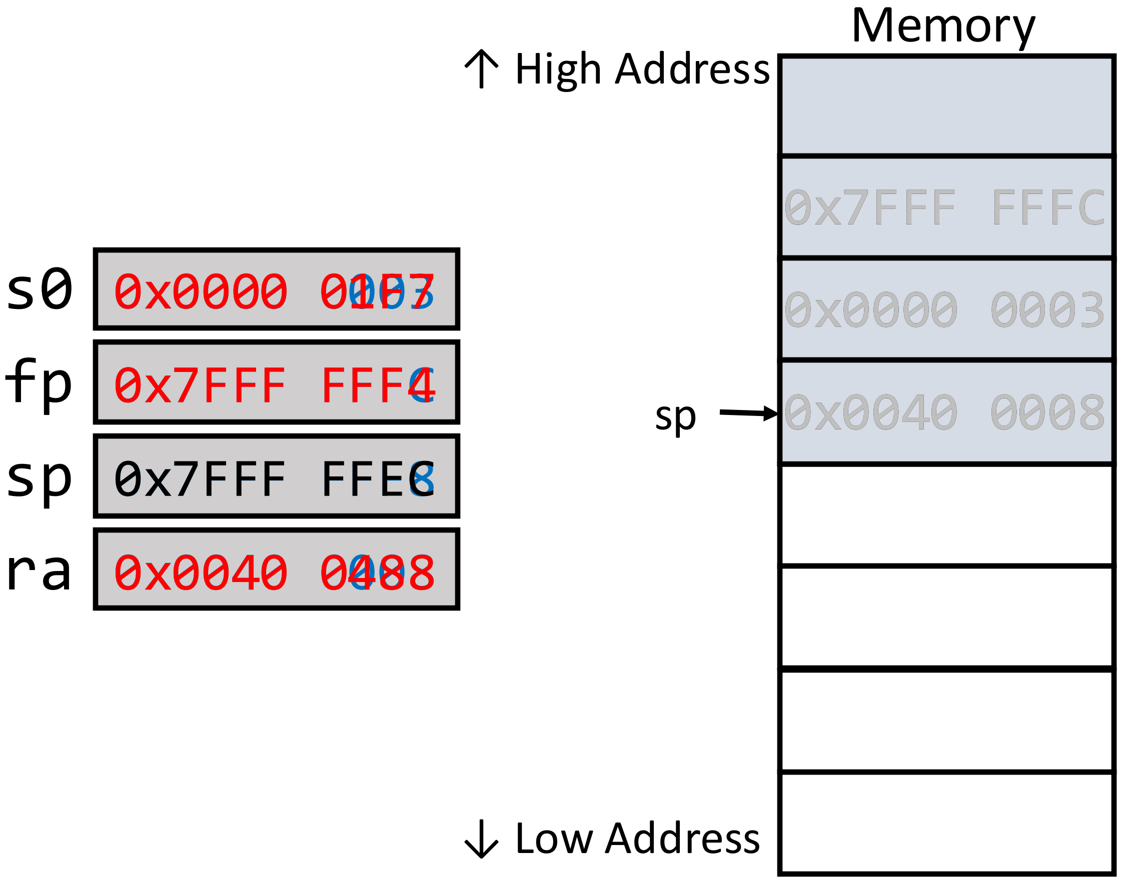
safe to overwrite return address into ra

return to correct address

restore ra from stack

return to correct address

Saving Registers Into the Stack



Now that the procedure has completed its execution it is time to return to the caller

How do we 'restore' the value of `s0`, `fp`, and `ra`?

Read their values, using the `lw` instruction, from the stack

Do we need to do anything to the value of `sp`?

Yes, we need to increment the value of `sp` by 12

Memory Layout

Text: program code (instructions)

Static data: global variables

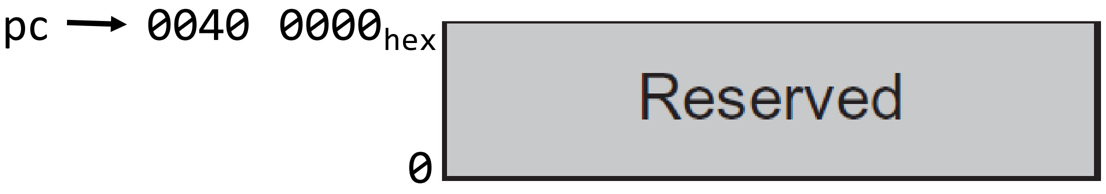
e.g. variables in C, constant arrays and strings

gp initialized to address allowing ± offsets into this segment

Dynamic data: heap

e.g. malloc in C, new in Java

Stack: automatic storage



Memory Layout

Text: program code (instructions)

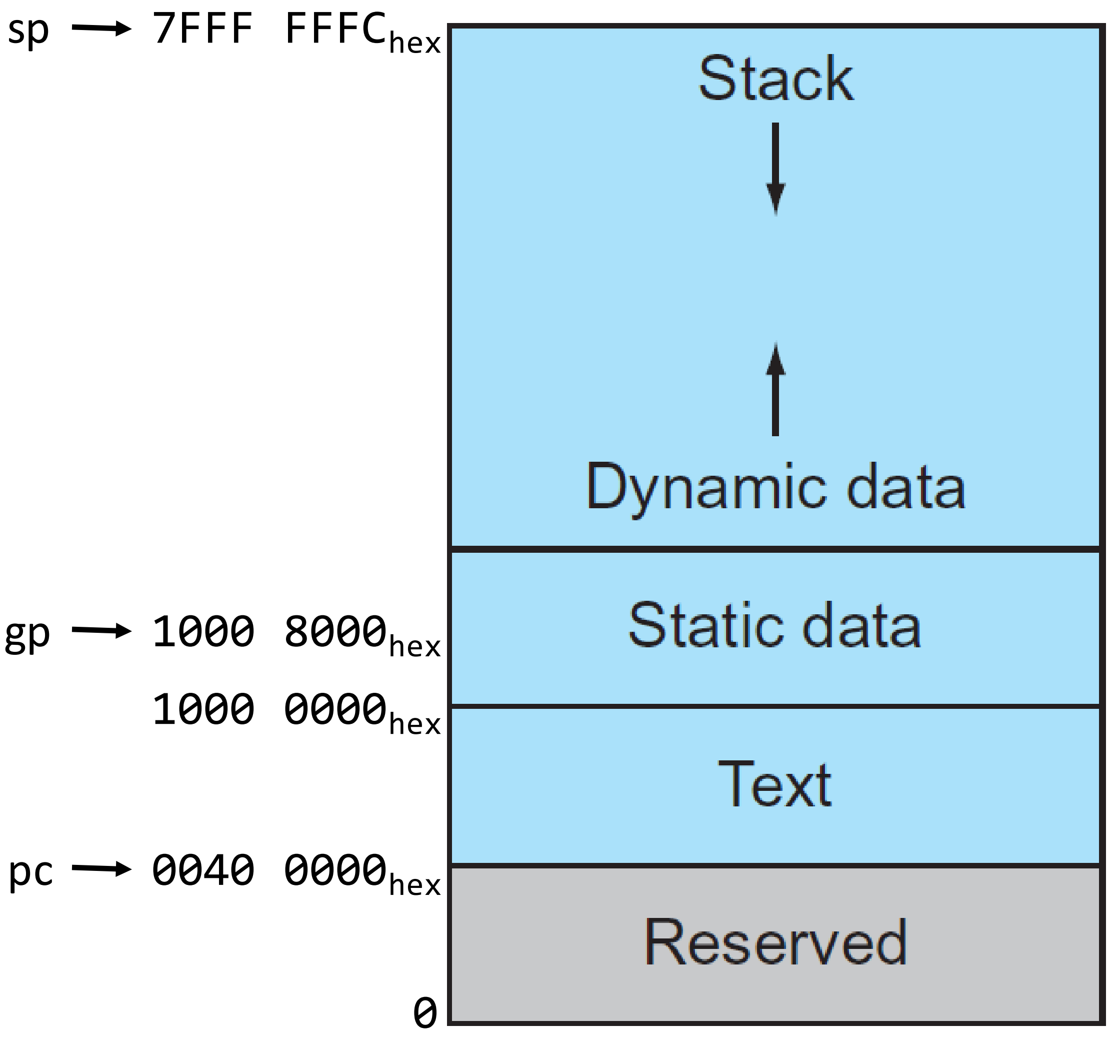
Static data: global variables

- e.g. variables in C, constant arrays and strings
- gp initialized to address allowing ± offsets into this segment

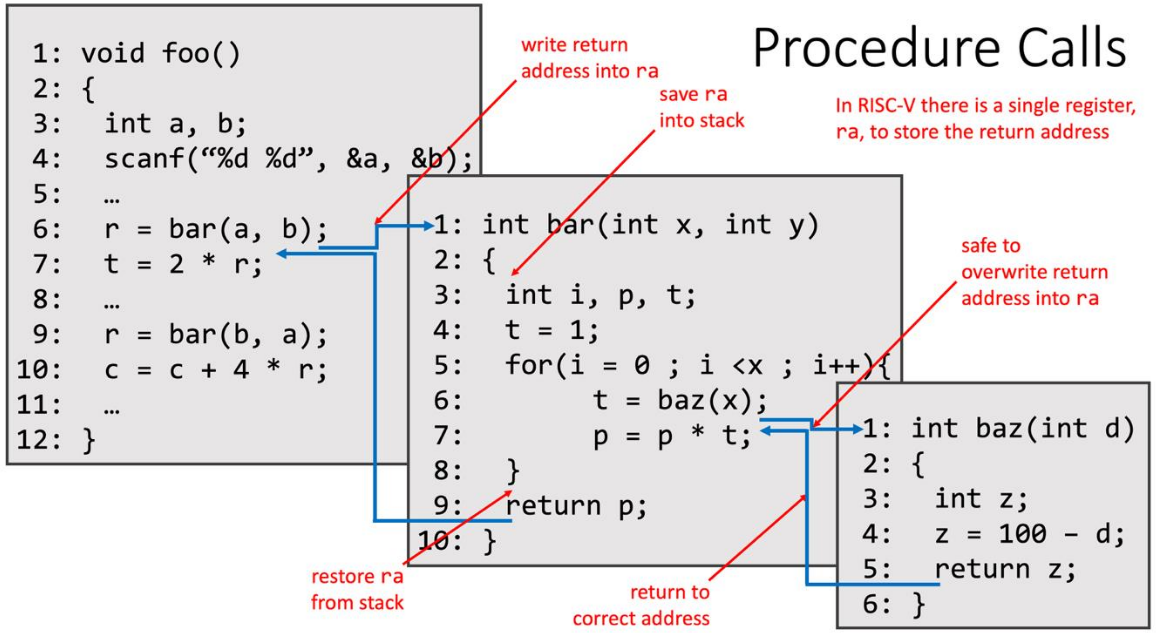
Dynamic data: heap

- e.g. malloc in C, new in Java

Stack: automatic storage



Procedure Calls



```
void foo()
{
  int a, b;
  scanf(“%d %d”, &a, &b);
  …
  r = bar(a, b);
  t = 2 * r;
  …
  r = bar(b, a);
  c = c + 4 * r;
  …
}
```

```
int bar(int x, int y)
{
  int i, p, t;
  t = 1;
  for(i = 0 ; i <x ; i++){
        t = baz(x);
        p = p * t;
  }
  return p;
}
```

```
int baz(int d)
{
  int z;
  z = 100 - d;
  return z;
}
```

write return address into ra

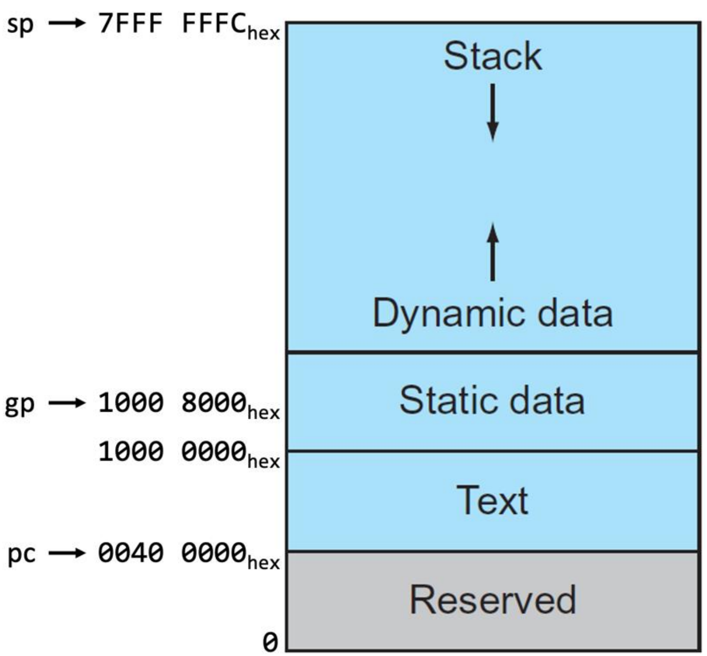
save ra into stack

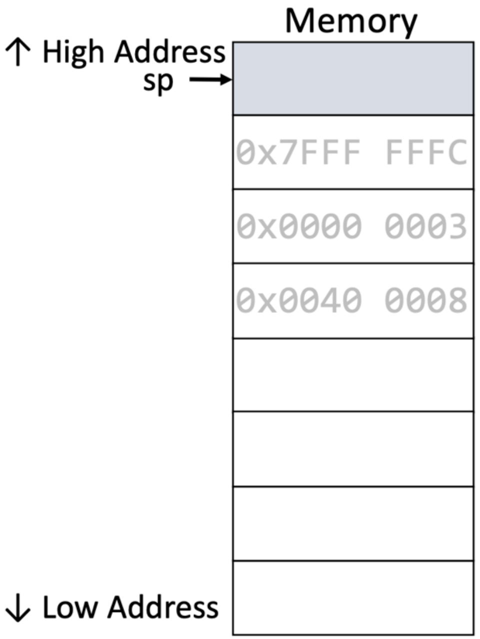
In RISC-V there is a single register, ra, to store the return address

safe to overwrite return address into ra

restore ra from stack

return to correct address





Memory
0x7FFF FFFC
0x0000 0003
0x0040 0008

Recap