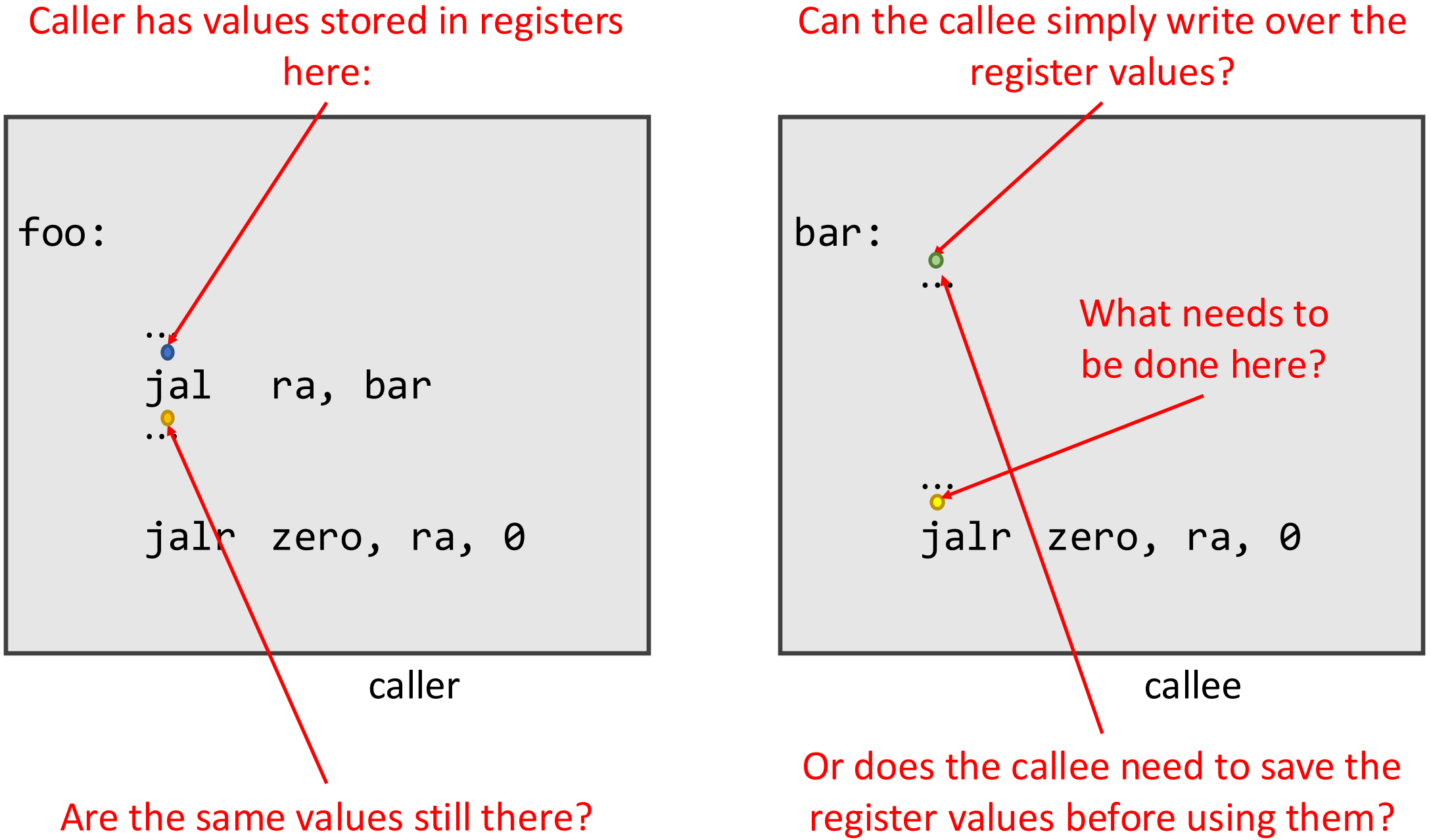
Topic V0C

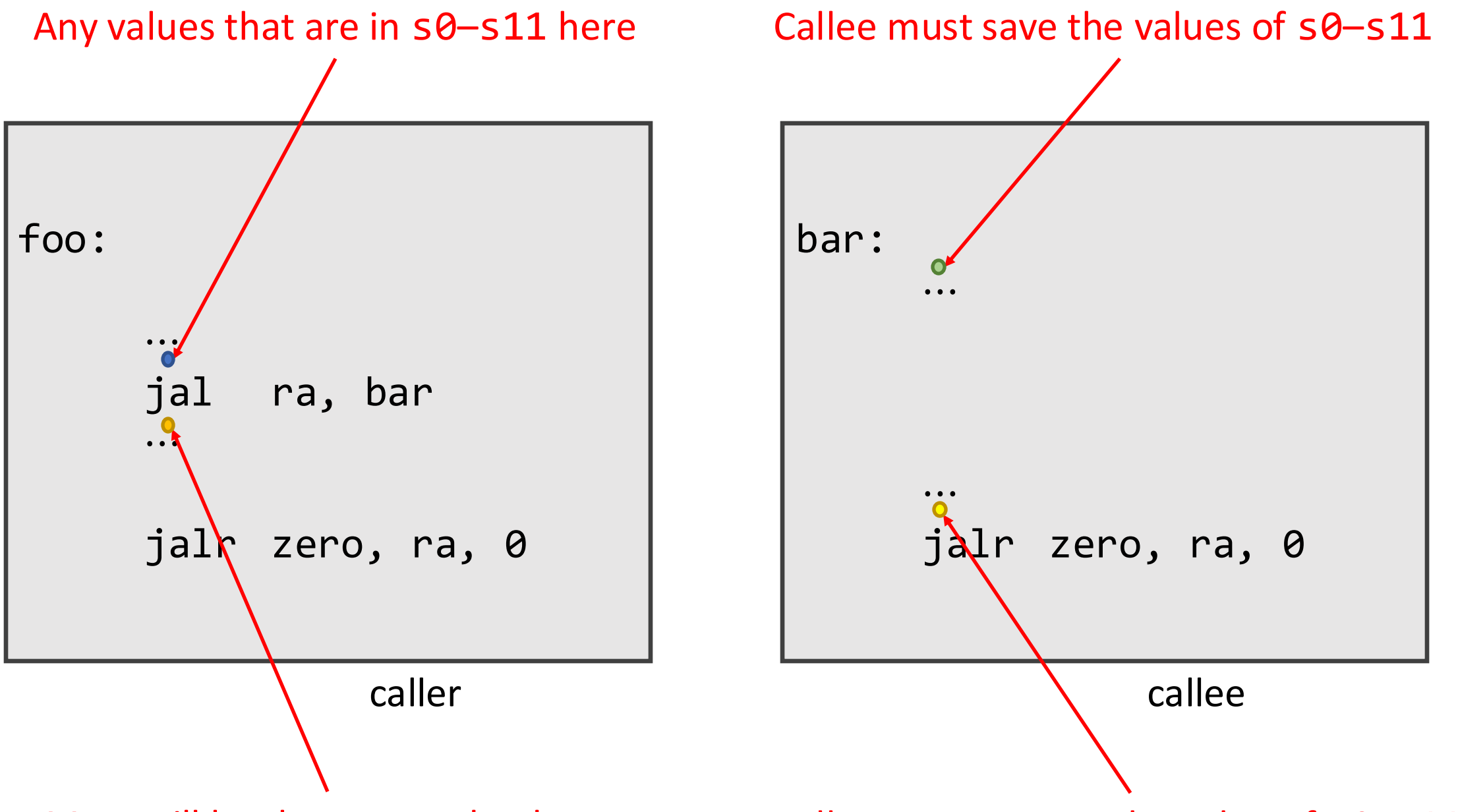
Register Calling Conventions

Readings: (Section 2.8)

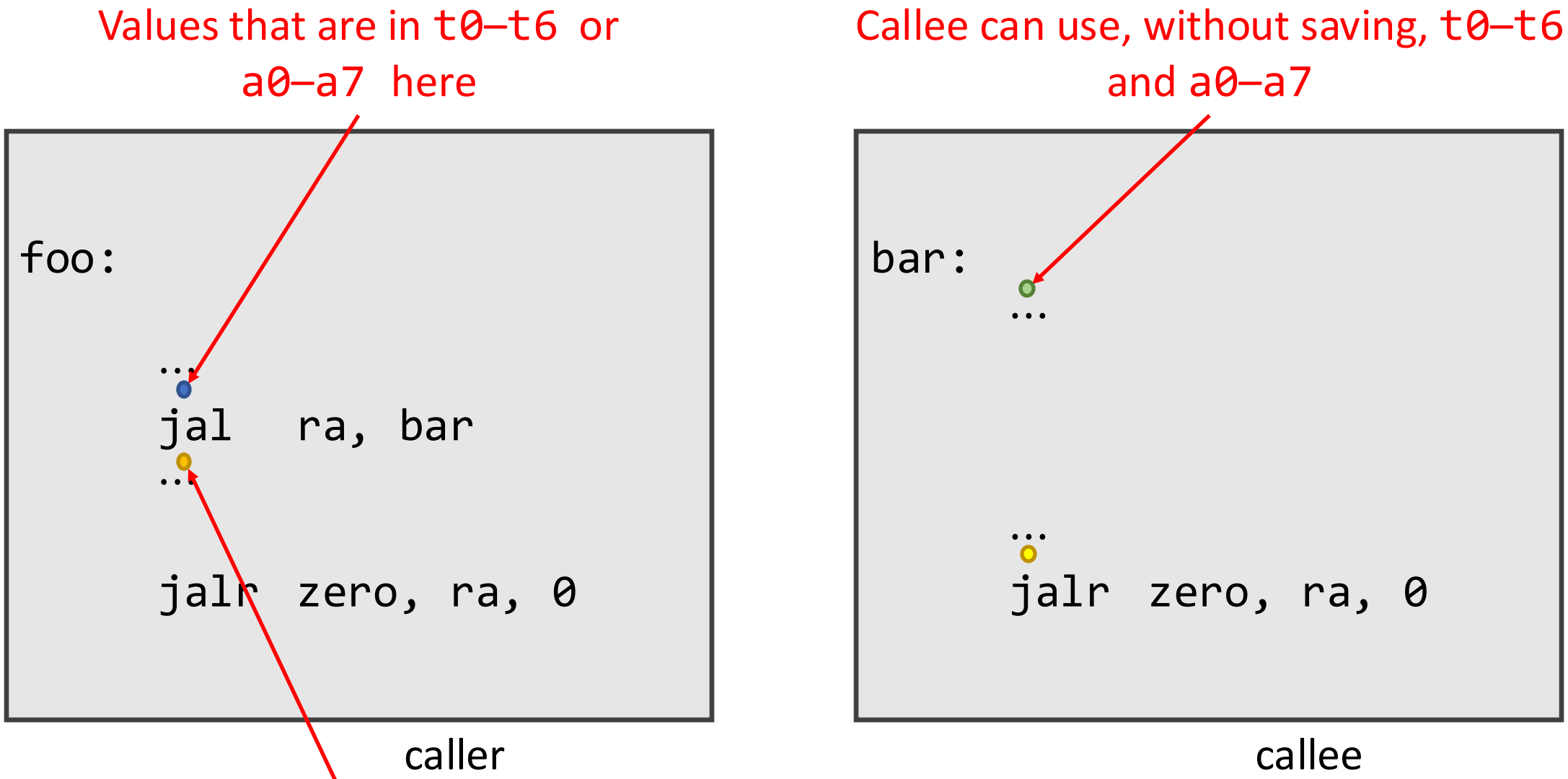
Register saving/restoring calling conventions


Caller has values stored in registers here:
foo:
…
jal ra, bar
…
jalr zero, ra, 0
caller
Are the same values still there?
Can the callee simply write over the register values?
bar:
…
What needs to be done here?
…
jalr zero, ra, 0
callee
Or does the callee need to save the register values before using them?

s0–s11 are saved registers



t0–t6 and a0–a7 are temporary registers



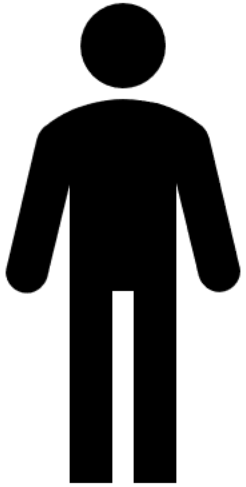
Registers Used for Procedure Calls

a0–a1: two argument registers in which to pass parameters;
also used to return values from a procedure;

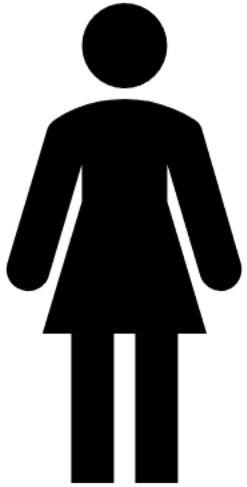
a2–a7: six argument registers in which to pass parameters;

ra: return-address register to return to the point after the call;

The need for a convention

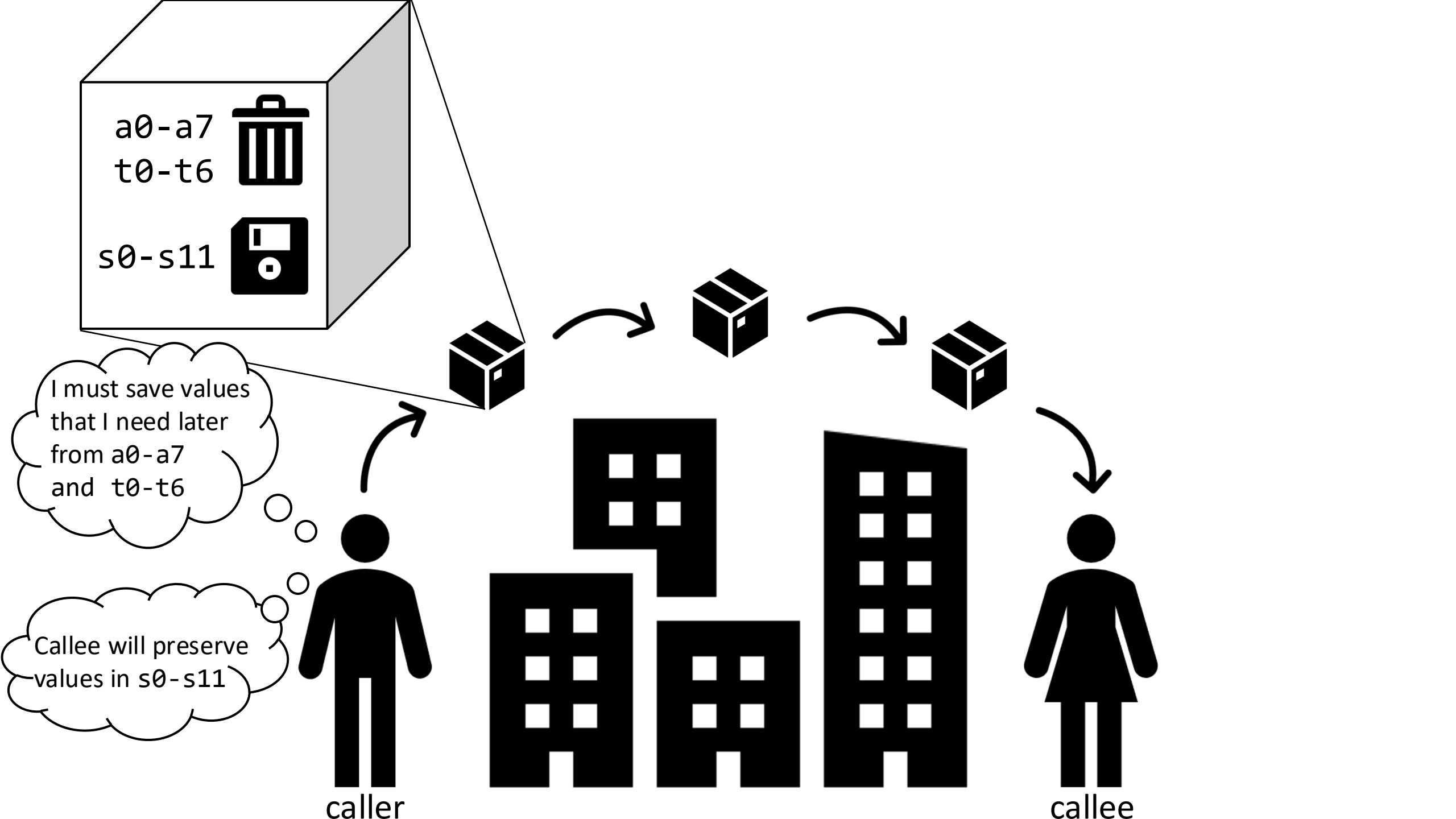


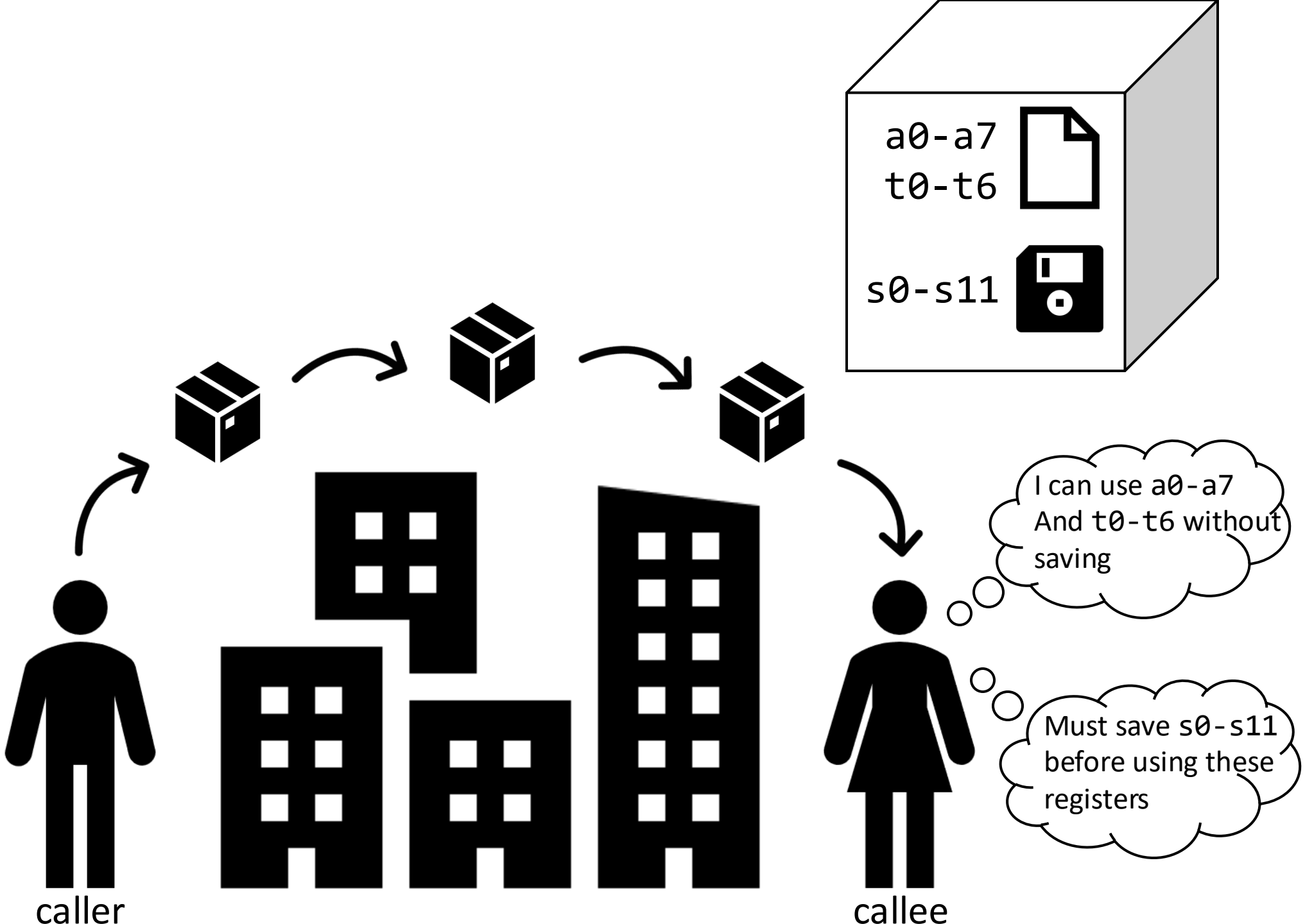
Hooman



Hamidat

An illustration of the RISC-V register calling convention


a0-a7
t0-t6
s0-s11
I must save values that I need later from a0-a7 and t0-t6
Callee will preserve values in s0-s11
caller
callee


a0-a7
t0-t6
s0-s11
I can use a0-a7
And t0-t6 without
saving
Must save s0-s11
before using these
registers
caller
callee

Register Usage Conventions

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--

The Thread Pointer to Thread Local Storage (TLS) facilitates switching between threads in multi-threaded programming.

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x28-x31	t3-t6	Temporaries	Caller

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x28-x31	t3-t6	Temporaries	Caller

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function arguments/return values	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function arguments/return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function arguments/return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller

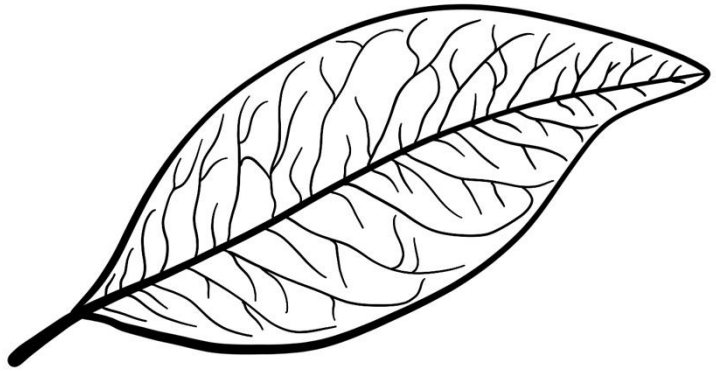
How is a function executed?

A function cannot be executed unless it is called by another function.

⇒ every function is a callee

⇒ every function must preserve the original value of the **s** registers.

Even the main function is called by a runtime system function.
⇒ main is a callee
without a call main would never execute.

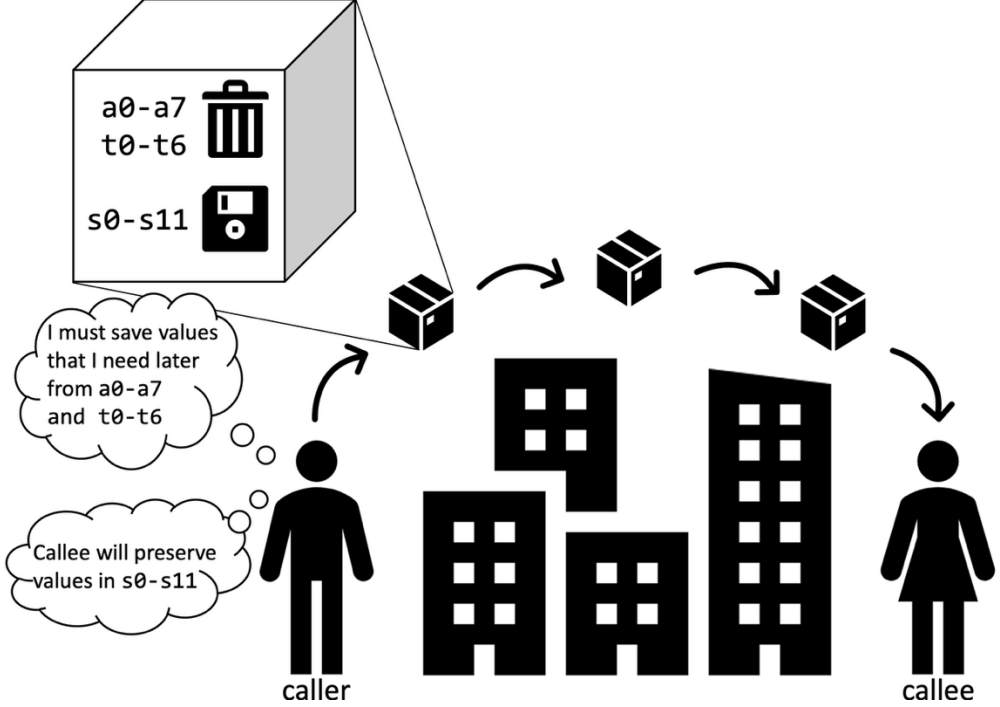


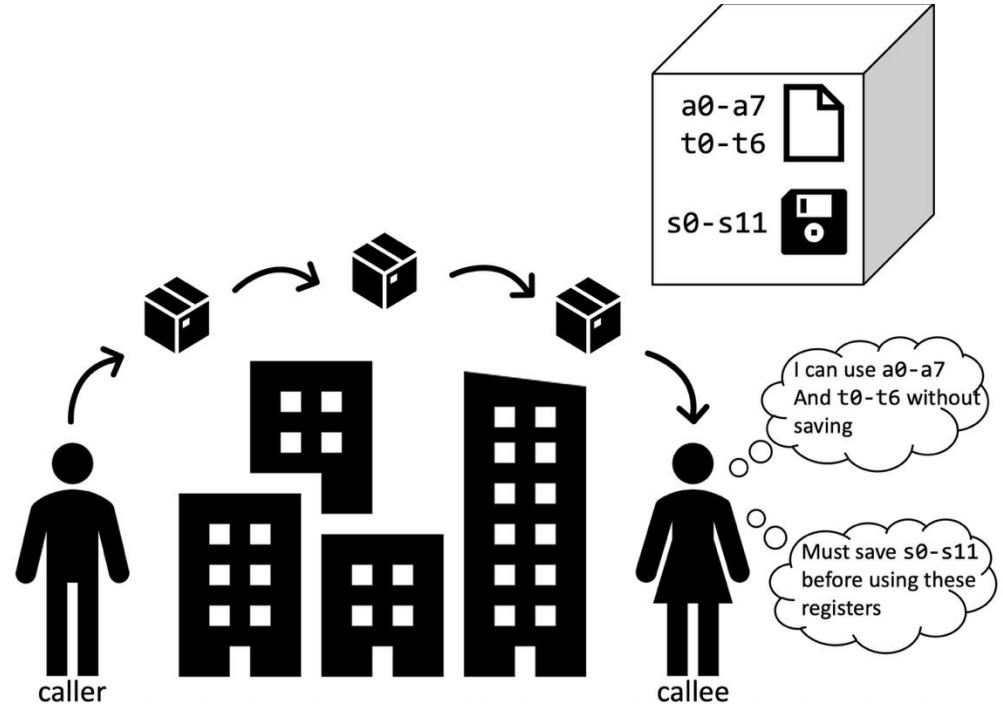
Leaf functions do not call another function.
Leaf functions are not callers.

Register Usage Conventions

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function arguments/return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller

Recap





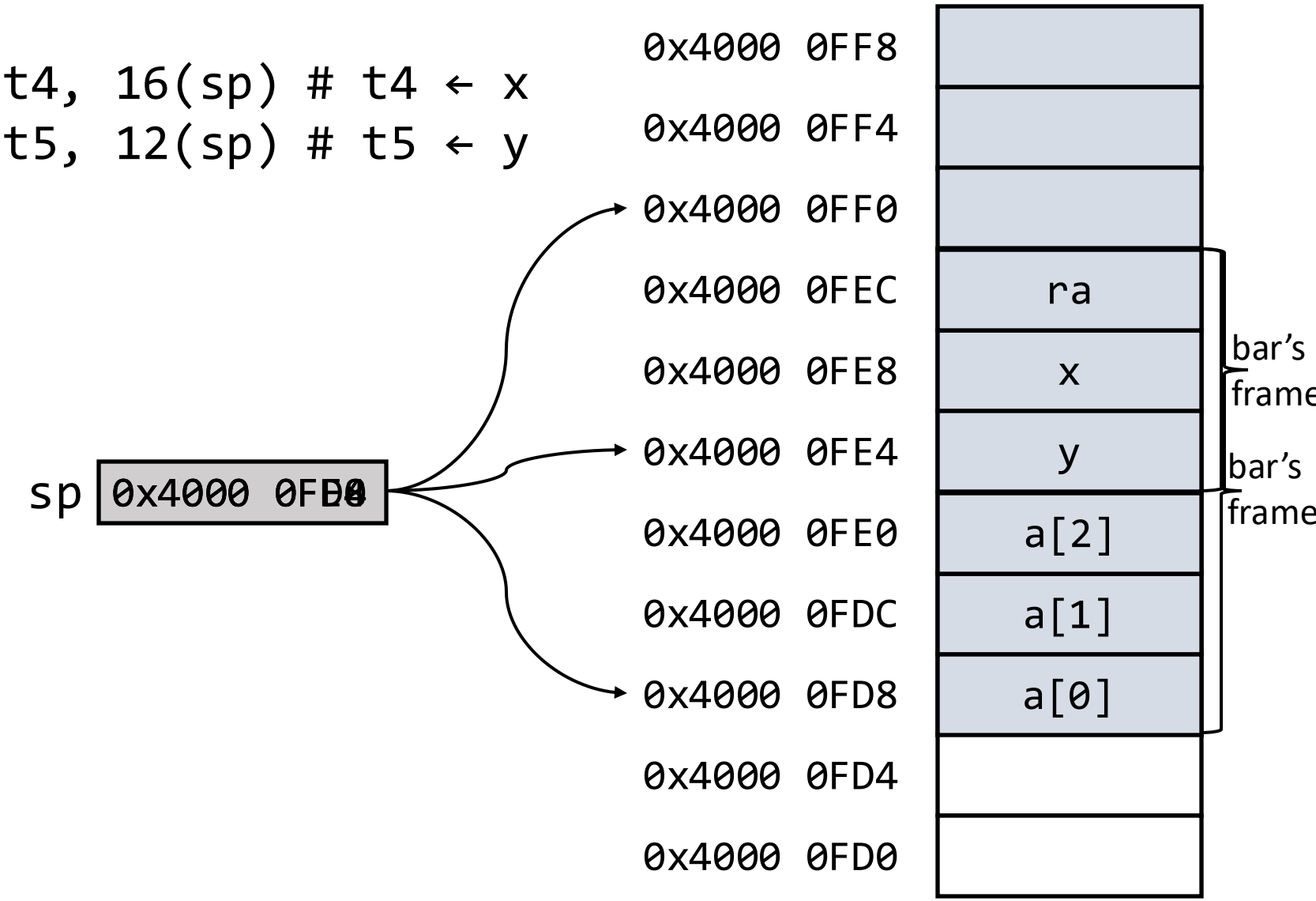
Why do we need a frame pointer?

An example without **fp**

```
void foo(…){
    …
    bar();
    …
}
```

```
lw    t0, 4(sp)  # t0 ← x
lw    t1, 0(sp)  # t1 ← y
…
lw    t4, 16(sp) # t4 ← x
lw    t5, 12(sp) # t5 ← y
```

```
int bar(…){
    int x, y;
    …
    x = x + y;
    if(…){
        int a[3];
        y = x + y;
        …
    }
    x = x + y;
}
```



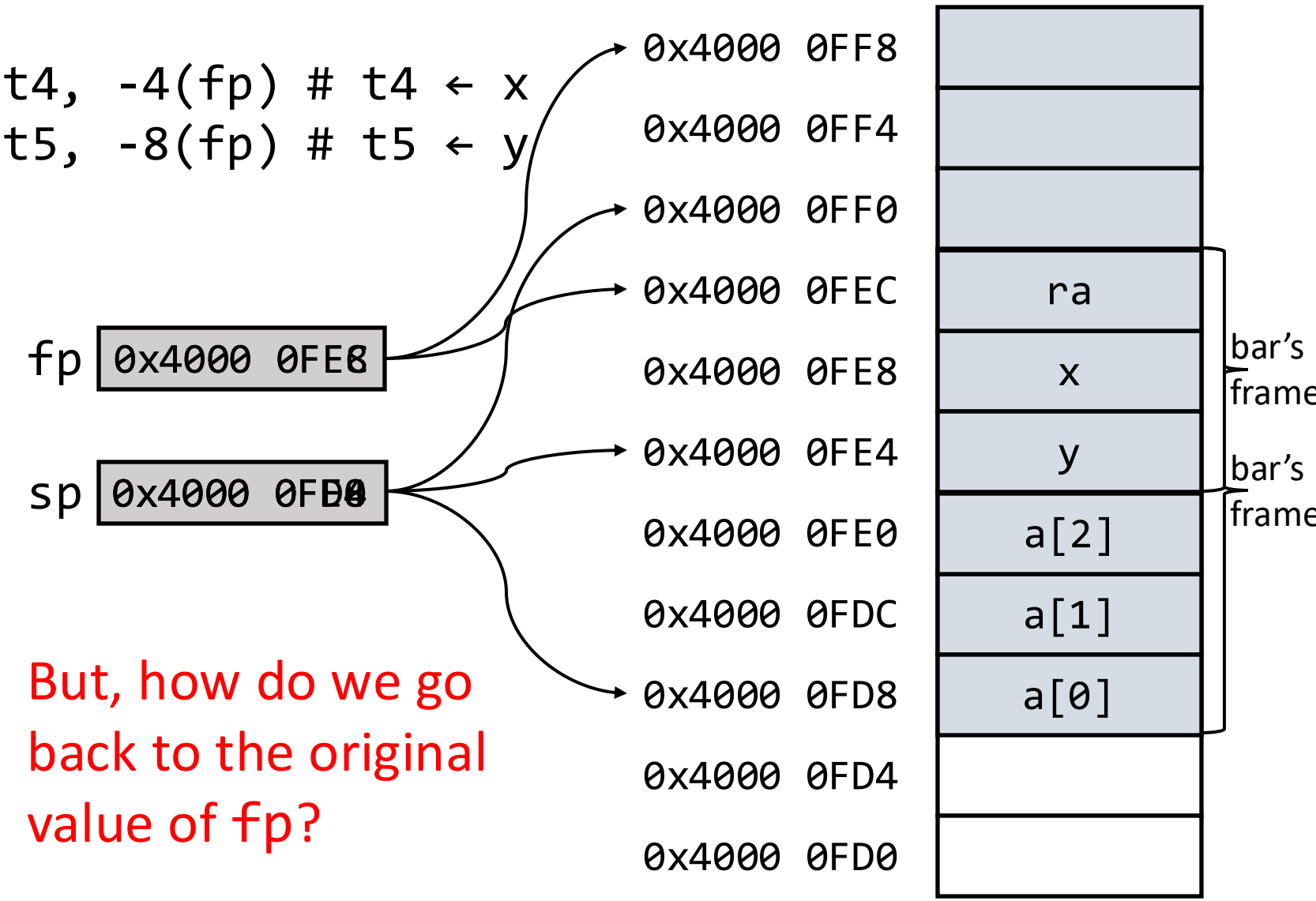
When a function declares local variables in the middle, the value of the **sp** changes.

Same example with **fp**

```
void foo(…){
    …
    bar();
    …
}
```

```
int bar(…){
    int x, y;
    …
    x = x + y;
    if(…){
        int a[3];
        y = x + y;
        …
    }
    x = x + y;
}
```

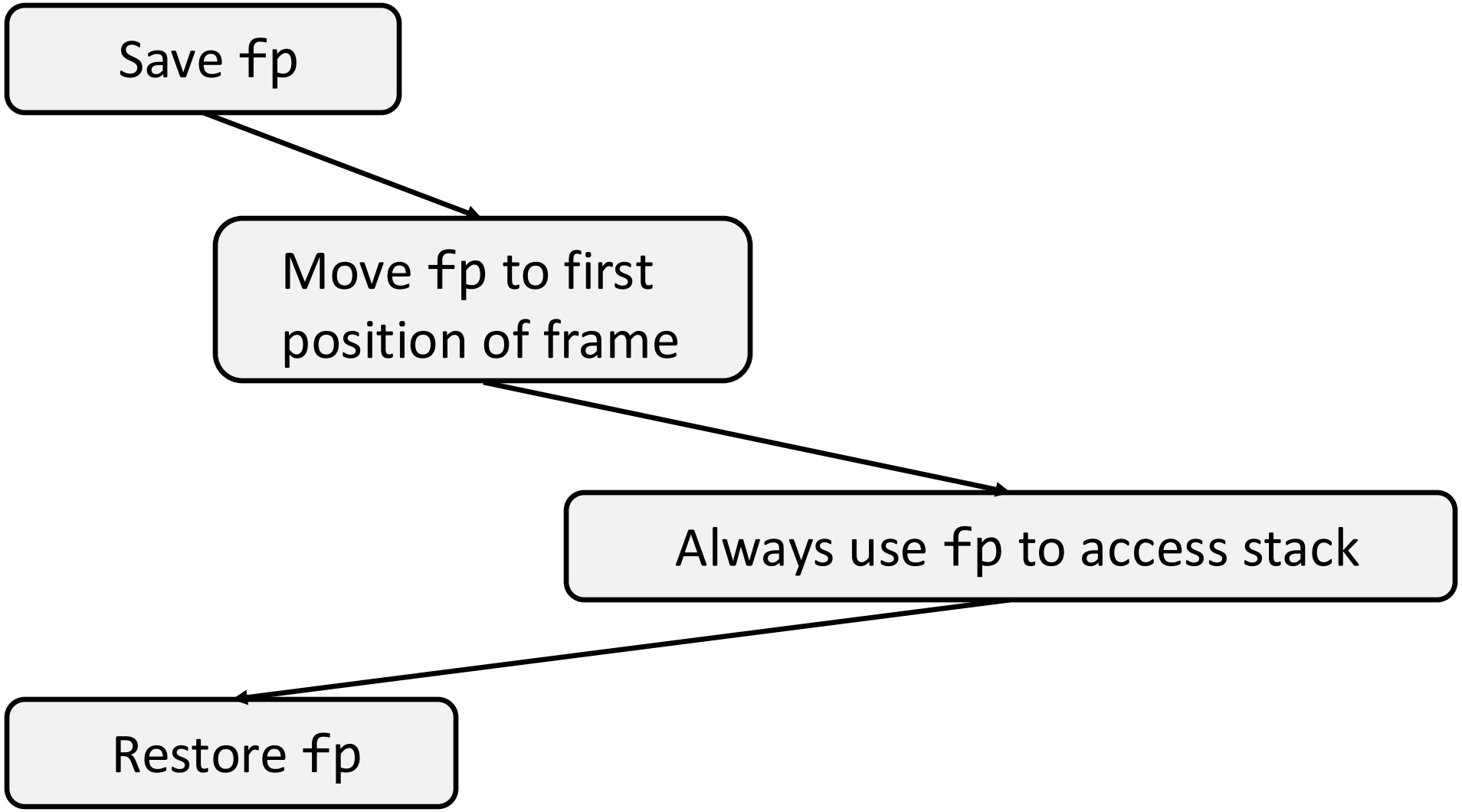
```
lw   t0, -4(fp) # t0 ← x
lw   t1, -8(fp) # t1 ← y
…
lw   t4, -4(fp) # t4 ← x
lw   t5, -8(fp) # t5 ← y
```



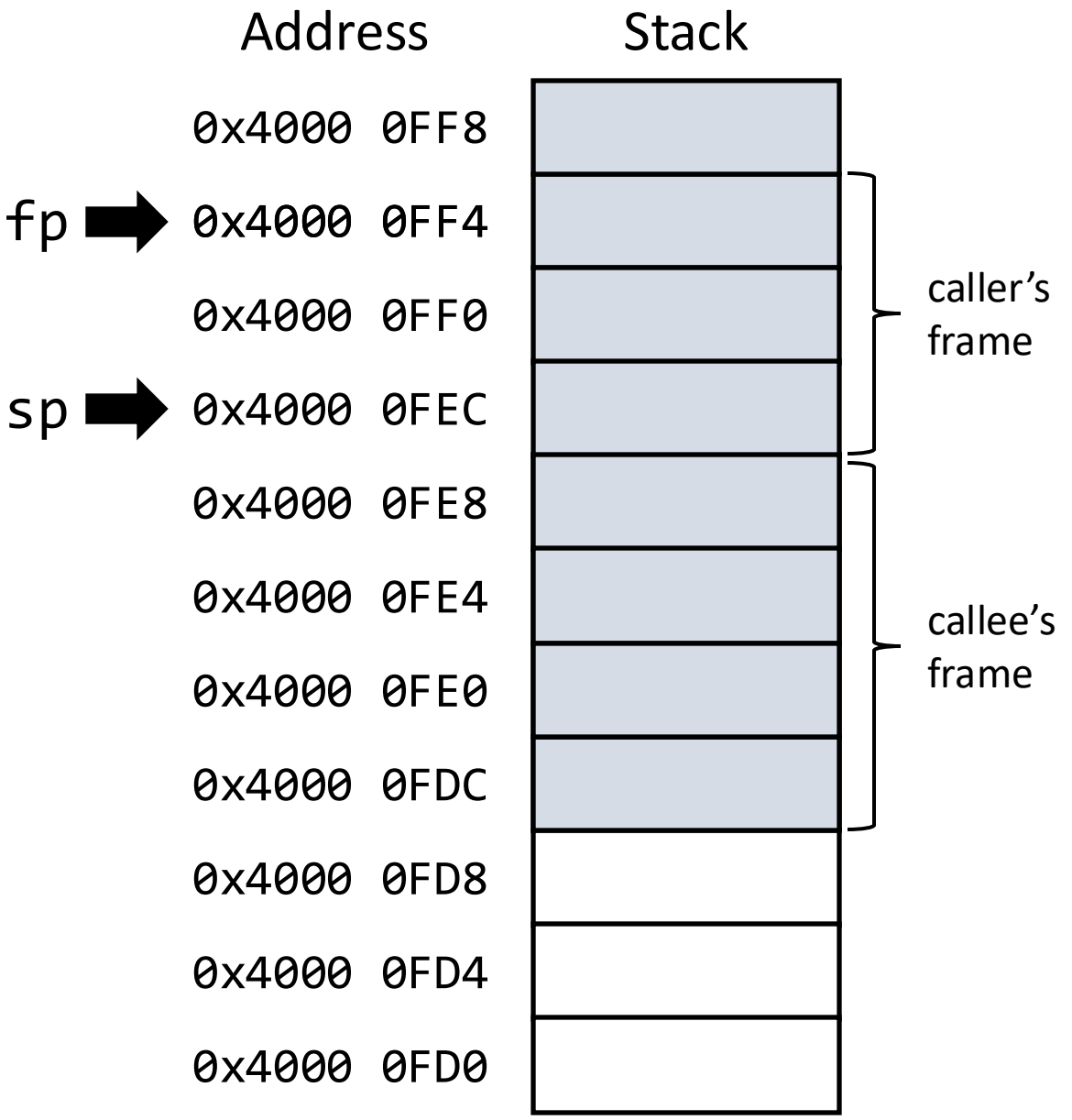
But, how do we go back to the original value of fp?

Frame Pointer

Some procedures **may** change sp during its execution, then:

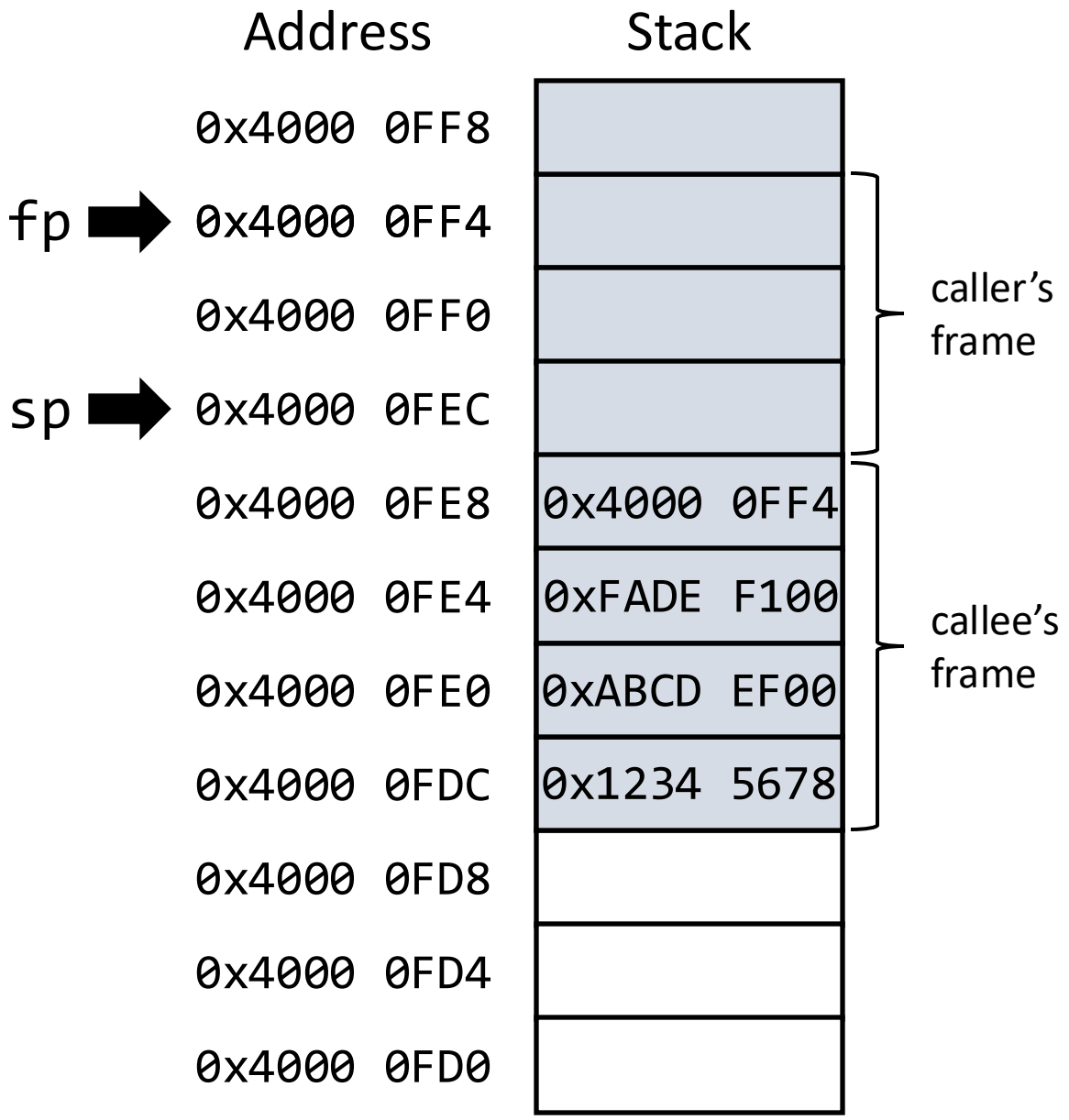


Frame Pointer (example)



Frame Pointer (example – at beginning)

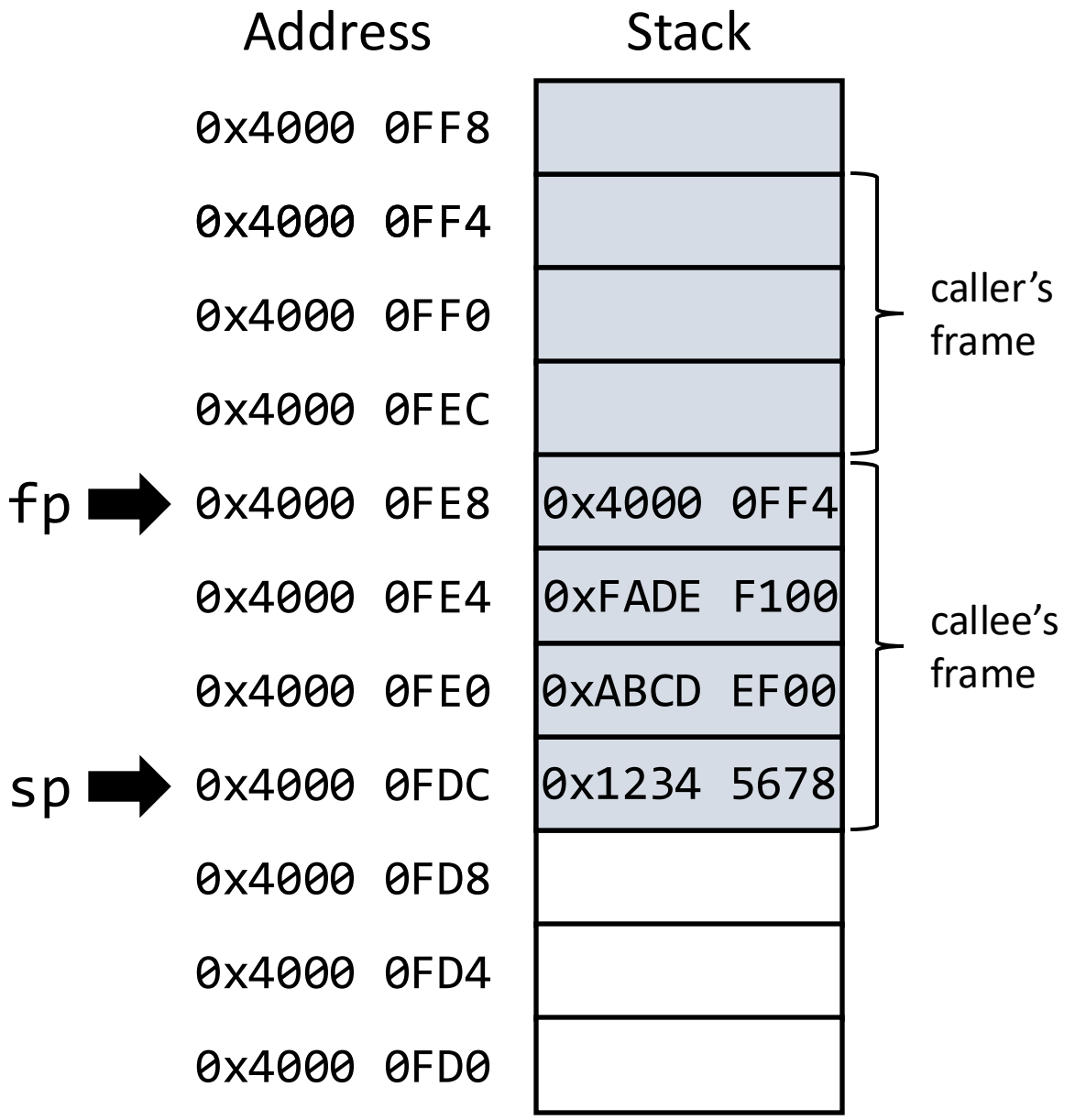
```
addi        sp, sp, -4
sw          fp, 0(sp)
mv          fp, sp
addi        sp, sp, -12
sw          s1, -4(fp)
…
```



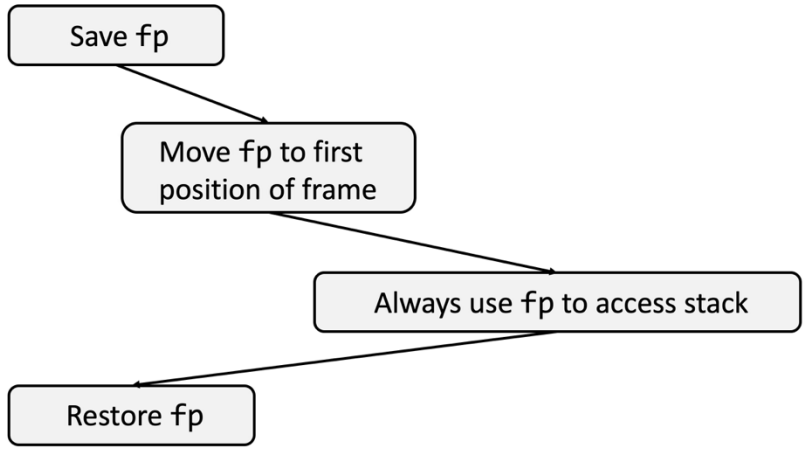
Frame Pointer (example – at end)

```
…
lw          s1, -4(fp)
lw          fp, 0(fp)
addi        sp, sp, 16

jalr        zero, ra, 0
```



```
int bar(…){
     int x, y;
     …
     x = x + y;
     if(…){
          int a[3];
          y = x + y;
          …
     }
     x = x + y;
}
```



Recap