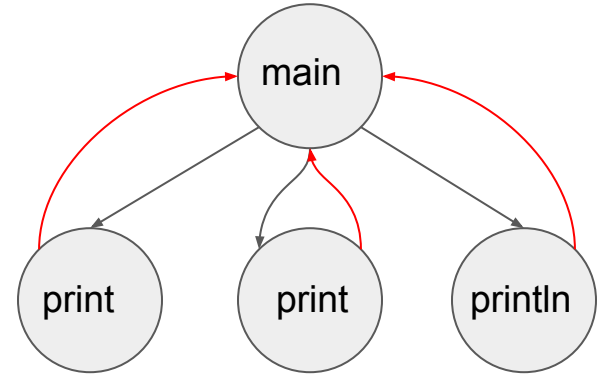



Call Graphs (recall) and Subroutine Construction


Call Graph

- a control flow graph depicting the relationships between subroutines
- Call Graph for the "Hello World" program

```
class HelloWorld
{
    public static void main(String args[])
    {
        System.out.print("Hello ");
        System.out.print("World");
        System.out.println("");
    }
}
```

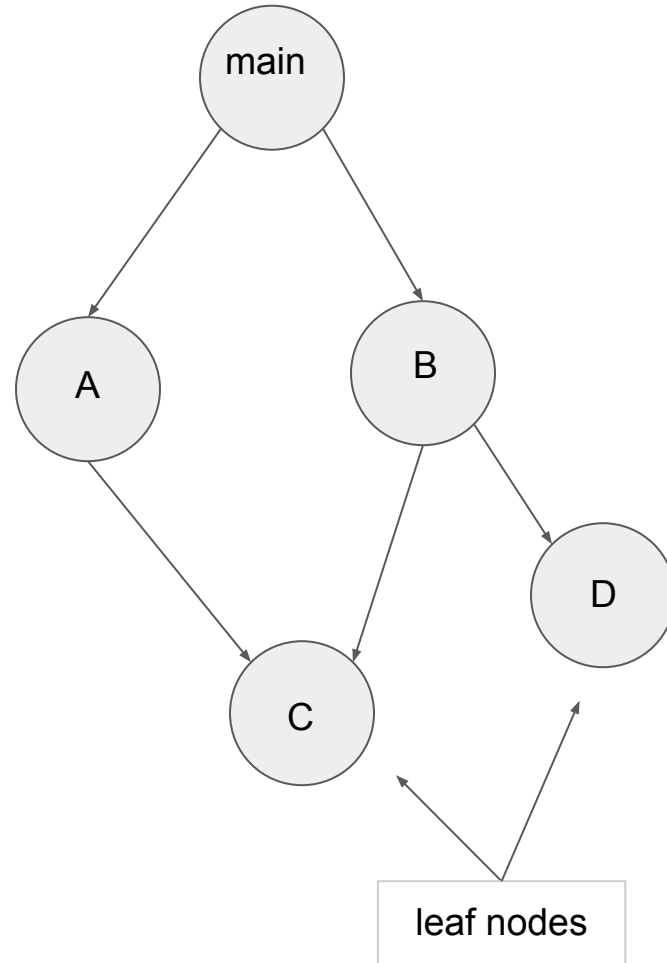


call: 

return: 

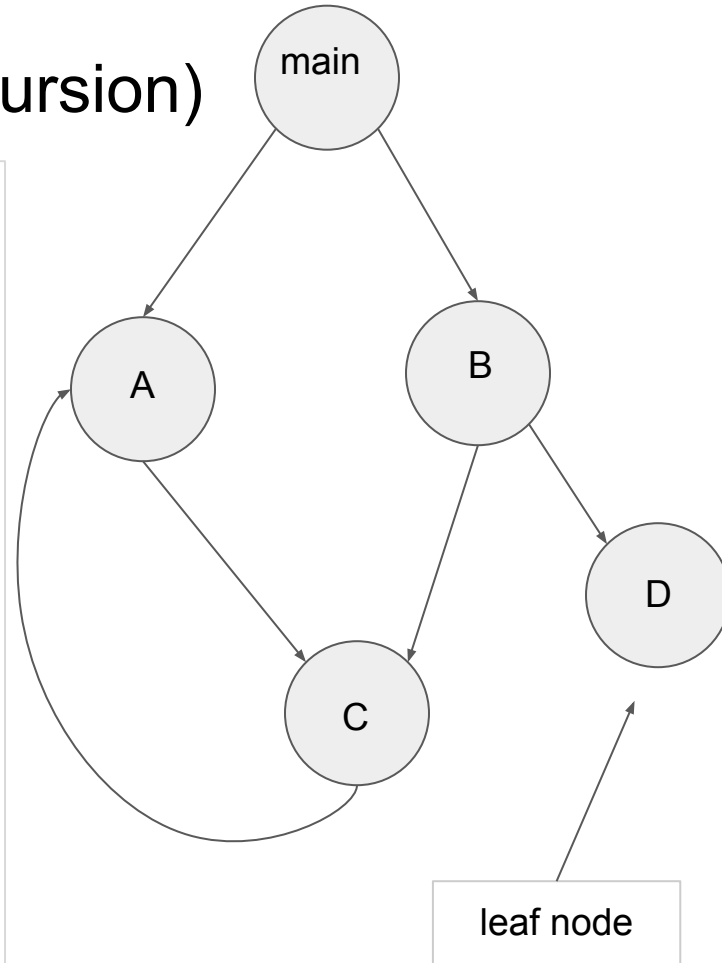
Call Graph II

```
public static void A(void) {  
    int x = 5;  
    C();  
}  
public static void B(void) {  
    C();  
    D();  
}  
public static void C(void) {  
    ;  
}  
public static void D(void) {  
    ;  
}  
  
public static void main(String args[])  
    {  
        A();  
        B();  
    }  
}
```



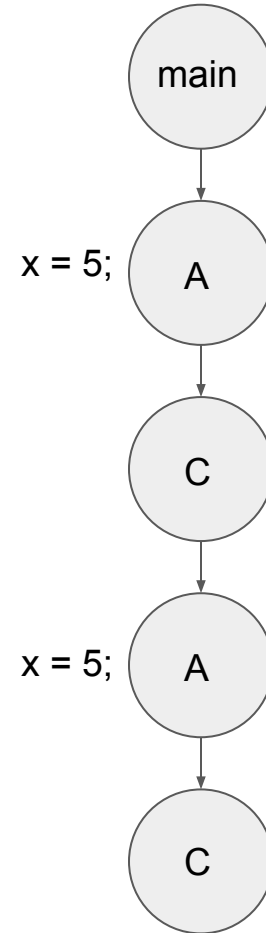
Call Graph with a Loop (Recursion)

```
public static void A(void) {  
    int x = 5;  
    C();  
}  
public static void B(void) {  
    C();  
    D();  
}  
public static void C(void) {  
    A();  
}  
public static void D(void) {  
    ;  
}  
  
public static void main(String args[])  
    {  
        A();  
        B();  
    }  
}
```

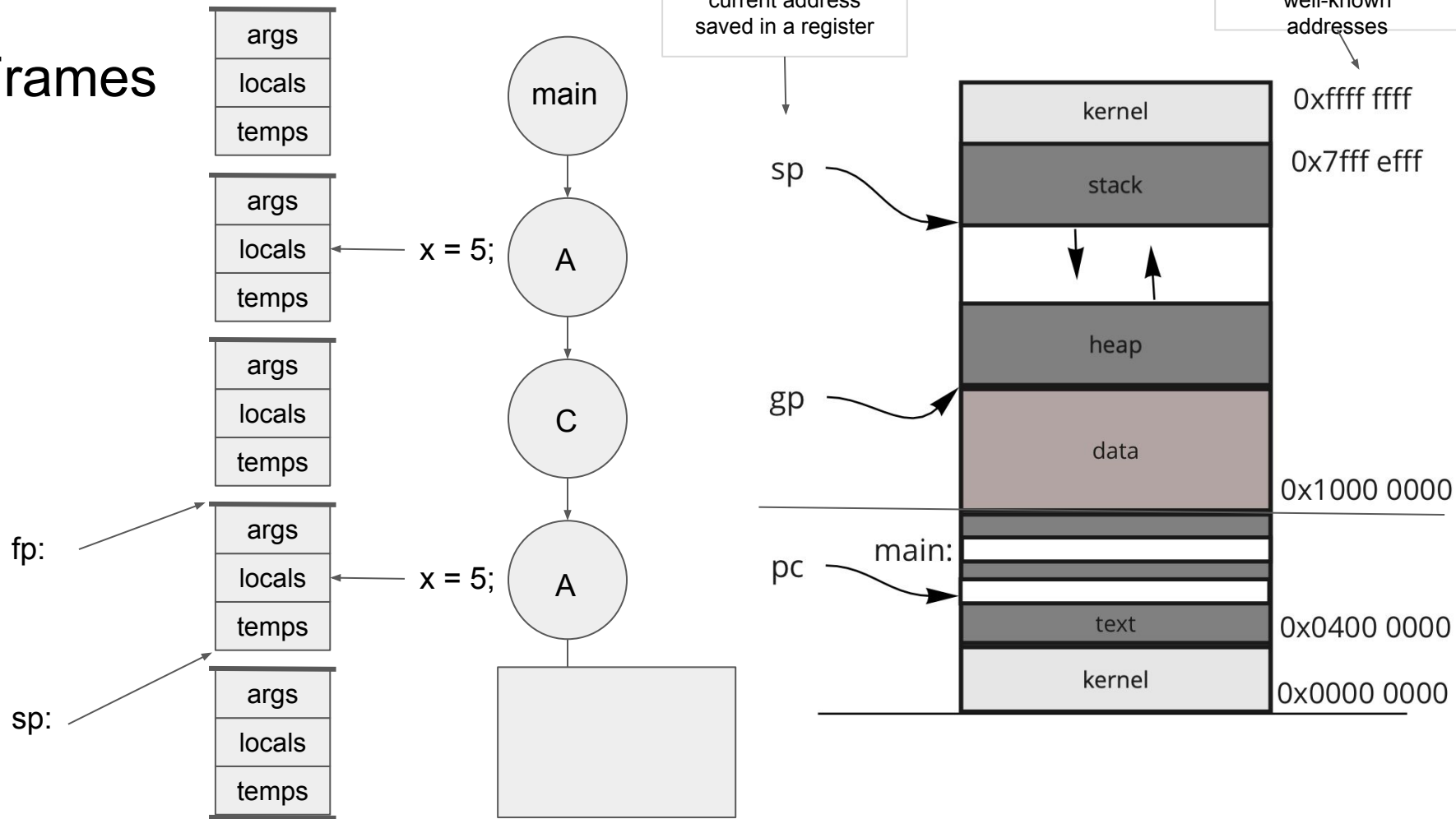


Dynamic Call Graph (Runtime)

```
public static void A(void) {  
    static int x = 5;  
    C();  
}  
public static void B(void) {  
    C();  
    D();  
}  
public static void C(void) {  
    A();  
}  
public static void D(void) {  
    ;  
}  
  
public static void main(String args[])  
    {  
        A();  
        B();  
    }  
}
```



Frames



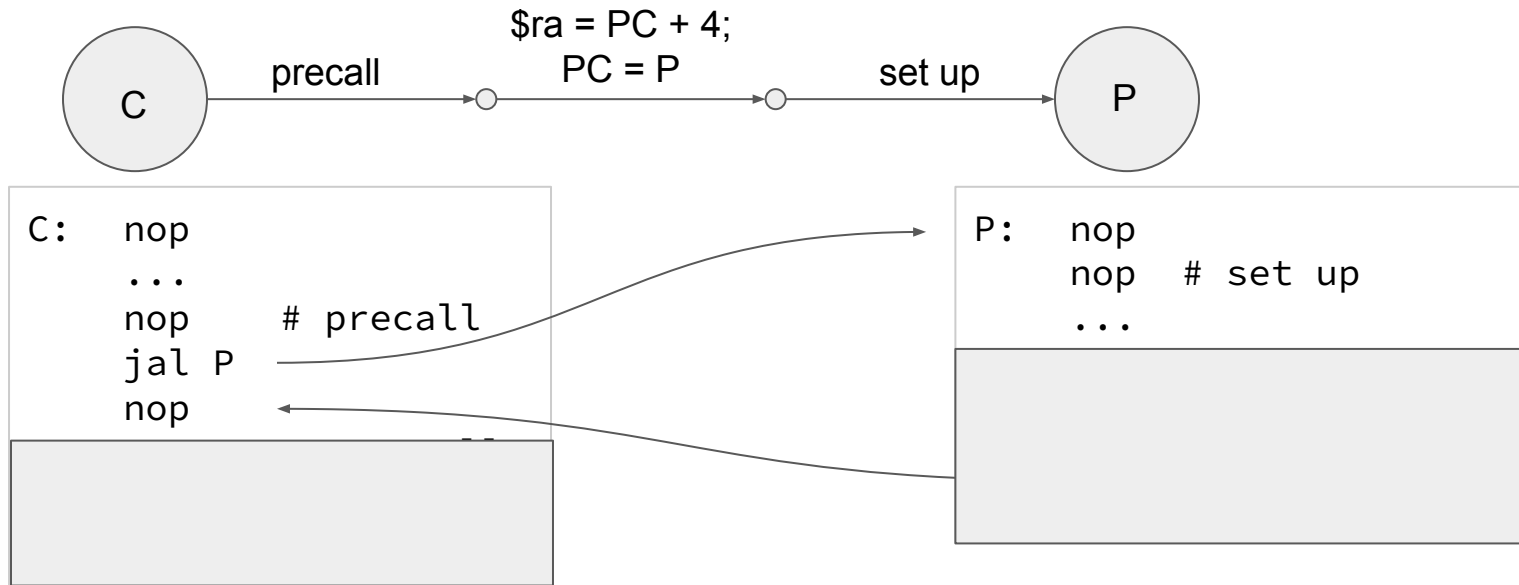
Subroutine Transition: Calling a Subroutine

1. The Client (C) needs to:

- Place actual args into the Frame
- Precall (preparation for the call)
- Transition

2. The Producer (P) needs to:

- Setup
- Do it's Thing



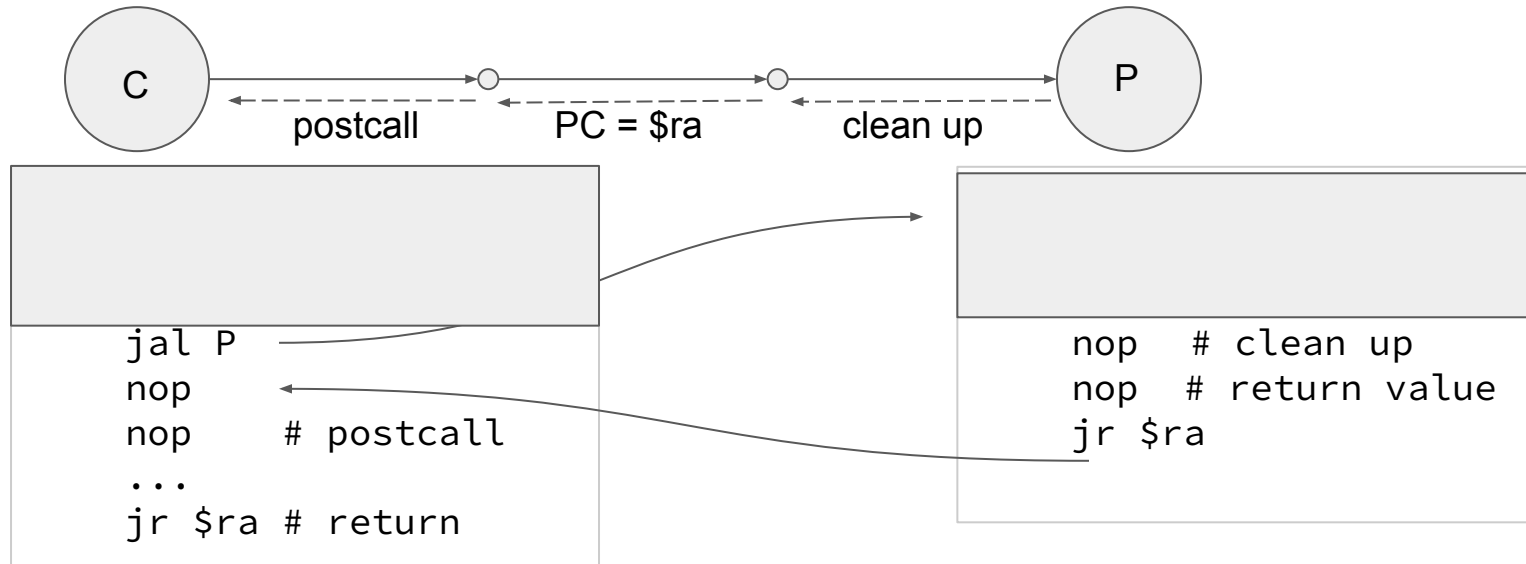
Subroutine Transition: Return from a Subroutine

2. The Client (C) needs to:

- Postcall
- Continue doing it's thing

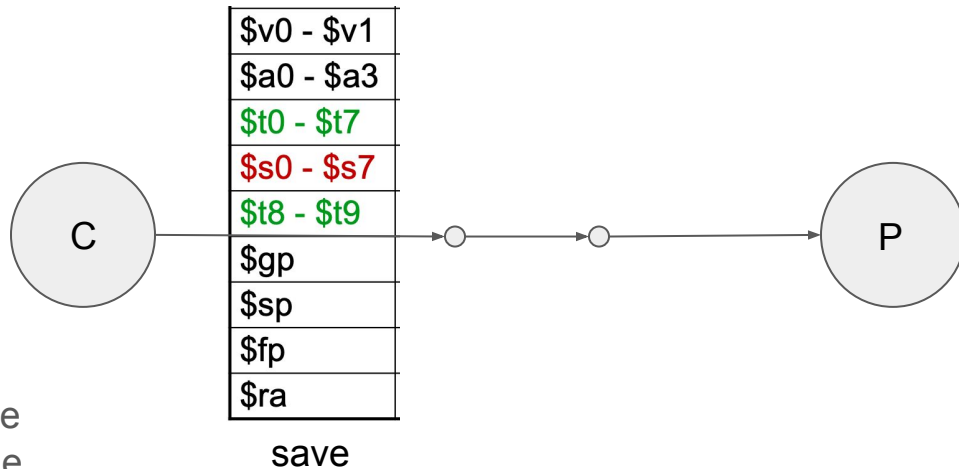
1. The Producer (P) needs to:

- Clean up
- Position the return value
- Transition back



Shared Resource: Registers

- You need to perform setup and cleanup routines for any shared resource!
- Precall:
 - Save what you need,
 - ~~Clear what you want private,~~
 - Leave alone what is passed along!
- Brute Force Approach:
 - ignore: \$zero, \$at, \$k1, \$k2
 - save all other registers
 - especially:
 - \$gp: might as well!
 - \$sp: this is the end of my frame
 - \$fp: this is the start of my frame
 - \$ra: this is my "return to" location

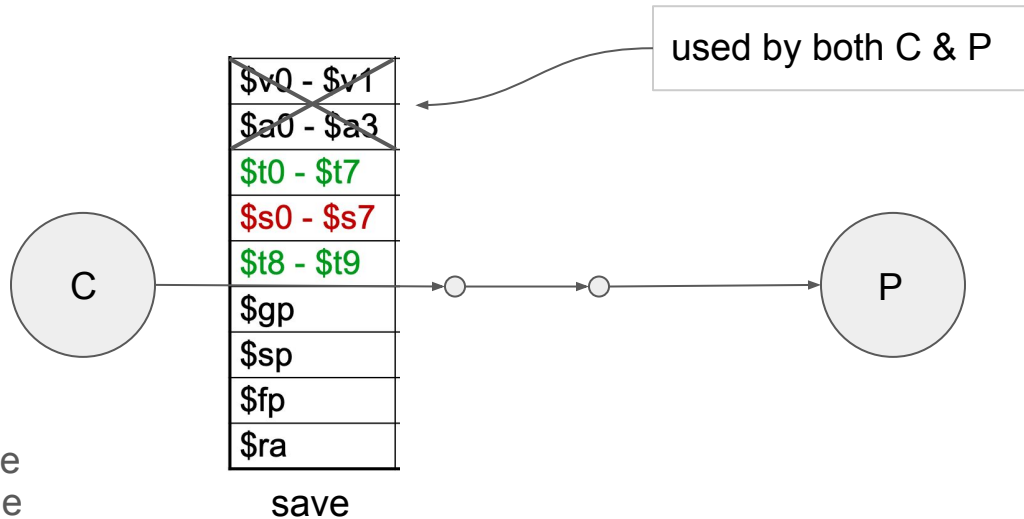


Shared Resource: Registers

- You need to perform setup and cleanup routines for any shared resource!

- Semi-Optimal

- ignore: \$zero, \$at, \$k1, \$k2
- save only registers in local use
- but always save:
 - \$gp: might as well!
 - \$sp: this is the end of my frame
 - \$fp: this is the start of my frame
 - \$ra: this is my "return to" location

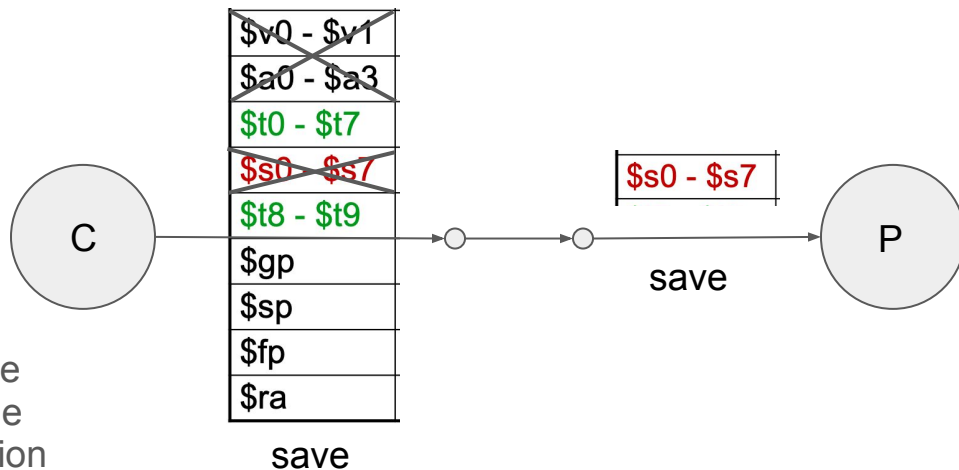


Shared Resource: Registers

- You need to perform setup and cleanup routines for any shared resource!

- MIPS Conventions

- Client (C): saves:
 - all T registers
 - \$gp: might as well!
 - \$sp: this is the end of my frame
 - \$fp: this is the start of my frame
 - \$ra: this is my "return to" location
- Provider (P): saves:
 - all S registers



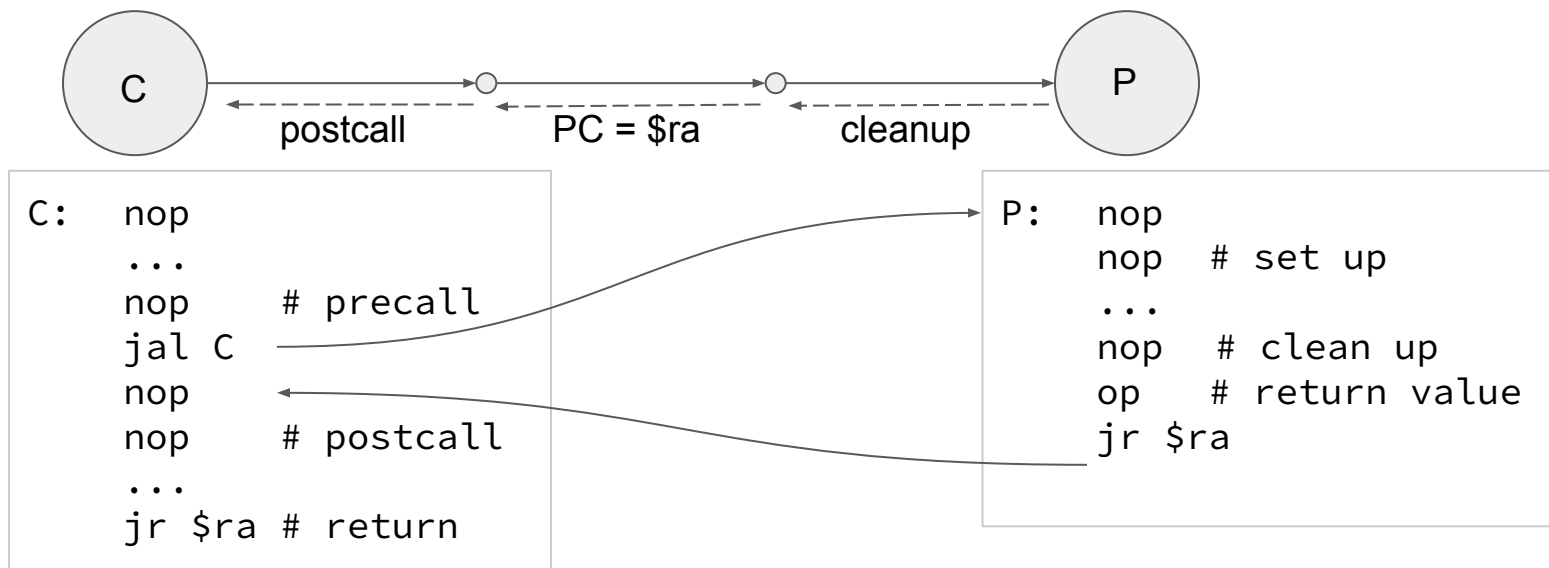
MIPS: Return from a Subroutine

2. The Client (C) needs to:

- Postcall ← Restore saved registers
- Do it's Thing

1. The Producer (P) needs to:

- Cleanup ← Restore S registers
- Position the return value
- Transition back

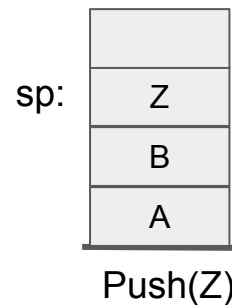
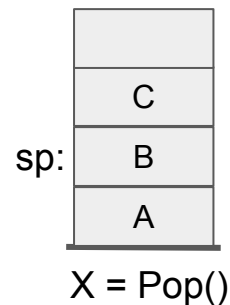
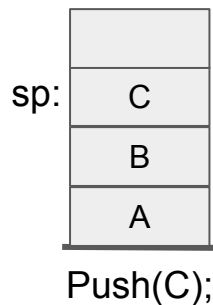
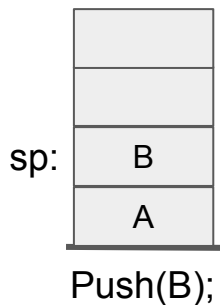
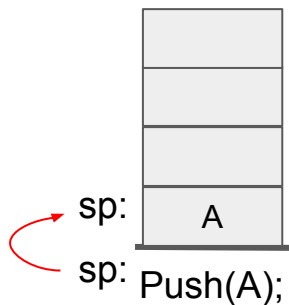


Stack Operations

- Stack is an abstract data structure
- Operations:
 - Push: Push(A), Push(B), Push(C)
 - Pop: X = Pop();
 - Push: Push(Z);

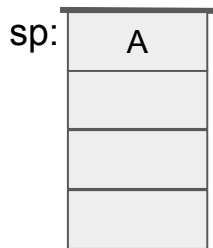
Push(a) \Leftrightarrow
 $sp = sp + 1$
 $sp[0] = a$

x = Pop() \Leftrightarrow
 $x = sp[0]$
 $sp = sp - 1$

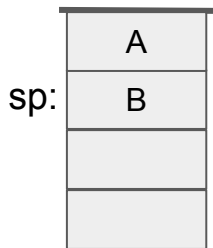


But the MIPS Way

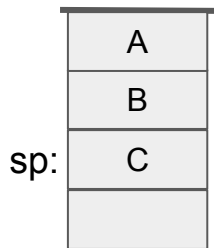
- Stack is an abstract data structure
- Operations:
 - Push: Push(A), Push(B), Push(C)
 - Pop: X = Pop();



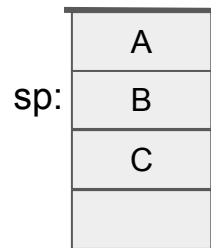
Push(A);



Push(B);



Push(C);



X = Pop();

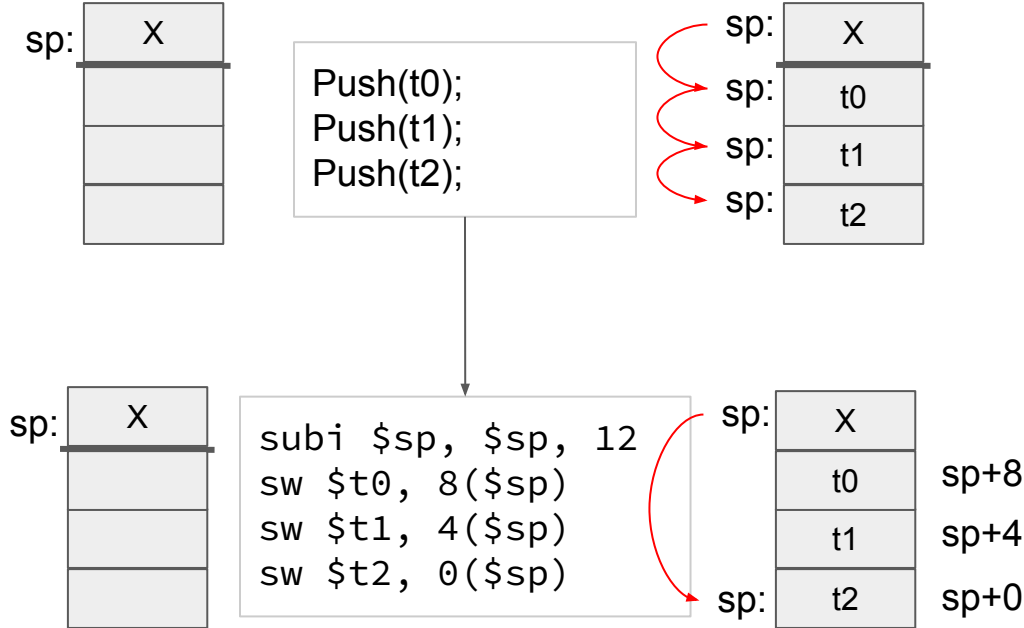
Push(a) \Leftrightarrow
 $sp = sp - 1$
 $sp[0] = a$

x = Pop() \Leftrightarrow
 $x = sp[0]$
 $sp = sp + 1$

Push(a) \Leftrightarrow
 $subi \$sp, \$sp, 4$
 $sw \$a0, 0(\$sp)$

x = Pop() \Leftrightarrow
 $lw \$v0, 0(\$sp)$
 $addi $sp, $sp, 4$

Multiple Pushes / Pops



Push(a) \Leftrightarrow
 $sp = sp - 1$
 $sp[0] = a$

x = Pop() \Leftrightarrow
 $x = sp[0]$
 $sp = sp + 1$

Push(a) \Leftrightarrow
 $subi \$sp, \$sp, 4$
 $sw \$a0, 0(\$sp)$

x = Pop() \Leftrightarrow
 $lw \$v0, 0(\$sp)$
 $addi \$sp, \$sp, 4$

t0 = Pop();
 t1 = Pop();
 t2 = Pop();

lw \$t0, 8(\$sp)
 lw \$t1, 4(\$sp)
 lw \$t2, 0(\$sp)
 addi \$sp, \$sp, 12

Args

Frames in Detail

fp:	X
	Y
	Z
	j
	k
	t0
	...
	s0
	...
	gp
	sp
	fp
sp:	ra

- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - $but\ sp = fp + 1$
 - position the return value: ($\$fp$), $\$v0$
 - jr $\$ra$ # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - $-4(\$sp)$ or $\$v0$

```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

    j = sub(1, k, 3);
    ;
    return j;
}
```

X	
Y	
Z	
j	
k	
t0	
-	
t9	
s0	
s7	
gp	
sp	
fp	
ra	

Calling "sub"



Client

➡ Precall steps before "sub"

- push args
- save registers
- jal sub # jump and link

● Steps to set up

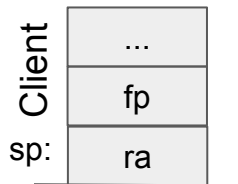
- build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
- save S registers

● Steps to clean up

- restore S registers
- delete the frame (no need to!)
 - but $sp = fp + 1$
- position the return value: ($\$fp$), $\$v0$
- jr $\$ra$ # jump register

● Postcall steps after "sub"

- restore registers
- move return value?
 - $-4(\$sp)$ or $\$v0$



```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

    j = sub(1, k, 3);
    ;
    return j;
}
```

X	↔	0(\$fp)
Y	↔	-4(\$fp)
Z	↔	-8(\$fp)
j	↔	-12(\$fp)
k	↔	-16(\$fp)
t0	↔	-20(\$fp)
...		
t9	↔	-56(\$fp)
s0	↔	-60(\$fp)
s7	↔	-88(\$fp)
gp	↔	-92(\$fp)
sp	↔	-96(\$fp)
fp	↔	-100(\$fp)
ra	↔	-104(\$fp)

Calling "sub"

Client	fp:	X
		Y
		Z
		j
		k
		t0
		...
		s0
		...
		gp
		sp
		fp
		ra

- Precall steps before "sub"

- push args
- save registers
- jal sub # jump and link

- Steps to set up

- build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
- save S registers

- Steps to clean up

- restore S registers
- delete the frame (no need to!)
 - but $sp = fp + 1$
- position the return value: ($\$fp$), $\$v0$
- jr $\$ra$ # jump register

- Postcall steps after "sub"

- restore registers
- move return value?
 - $-4(\$sp)$ or $\$v0$

Client		...
	fp	
	ra	
sp:		1
		k
		3
args		

```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

    j = sub(1, k, 3);
    ;
    return j;
}
```

X	↔	0(\$fp)
Y	↔	-4(\$fp)
Z	↔	-8(\$fp)
j	↔	-12(\$fp)
k	↔	-16(\$fp)
t0	↔	-20(\$fp)
...		
t9	↔	-56(\$fp)
s0	↔	-60(\$fp)
s7	↔	-88(\$fp)
gp	↔	-92(\$fp)
sp	↔	-96(\$fp)
fp	↔	-100(\$fp)
ra	↔	-104(\$fp)

Calling "sub"



- Precall steps before "sub"

- push args
- save registers
- jal sub # jump and link

- Steps to set up

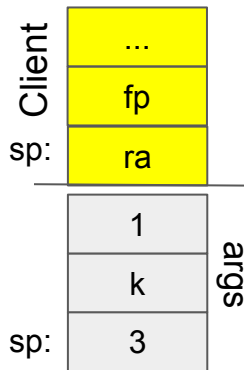
- build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
- save S registers

- Steps to clean up

- restore S registers
- delete the frame (no need to!)
 - but $sp = fp + 1$
- position the return value: (\$fp), \$v0
- jr \$ra # jump register

- Postcall steps after "sub"

- restore registers
- move return value?
 - -4(\$sp) or \$v0



```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

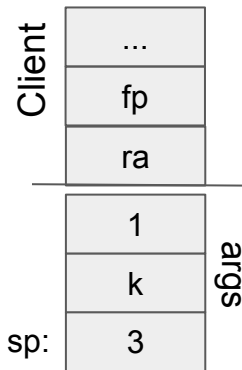
    j = sub(1, k, 3);
    ;
    return j;
}
```

X	⇔	0(\$fp)
Y	⇔	-4(\$fp)
Z	⇔	-8(\$fp)
j	⇔	-12(\$fp)
k	⇔	-16(\$fp)
t0	⇔	-20(\$fp)
...		
t9	⇔	-56(\$fp)
s0	⇔	-60(\$fp)
s7	⇔	-88(\$fp)
gp	⇔	-92(\$fp)
sp	⇔	-96(\$fp)
fp	⇔	-100(\$fp)
ra	⇔	-104(\$fp)

Transition to "sub"



- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but $sp = fp + 1$
 - position the return value: (\$fp), \$v0
 - jr \$ra # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - -4(\$sp) or \$v0



```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

    j = sub(1, k, 3);
    ;
    return j;
}
```

X ⇔ 0(\$fp)
 Y ⇔ -4(\$fp)
 Z ⇔ -8(\$fp)
 j ⇔ -12(\$fp)
 k ⇔ -16(\$fp)
 t0 ⇔ -20(\$fp)
 ...
 t9 ⇔ -56(\$fp)
 s0 ⇔ -60(\$fp)
 s7 ⇔ -88(\$fp)
 gp ⇔ -92(\$fp)
 sp ⇔ -96(\$fp)
 fp ⇔ -100(\$fp)
 ra ⇔ -104(\$fp)

Producer: The set up



- Precall steps before "sub"

- push args
- save registers
- jal sub # jump and link

➡ Steps to set up

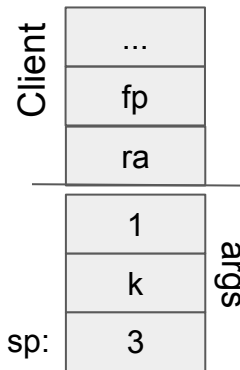
- build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
- save S registers

- Steps to clean up

- restore S registers
- delete the frame (no need to!)
 - but $sp = fp + 1$
- position the return value: ($\$fp$), $\$v0$
- jr $\$ra$ # jump register

- Postcall steps after "sub"

- restore registers
- move return value?
 - $-4(\$sp)$ or $\$v0$



```
int sub(int X, int Y, int Z) {
    ➡ int j;
      int k = Y + Z

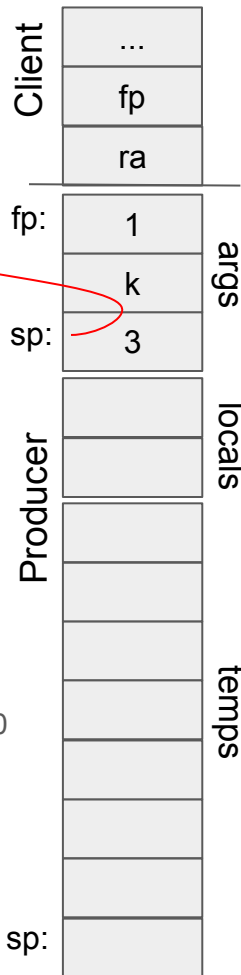
      j = sub(1, k, 3);
      ;
      return j;
}
```

X	↔	0(\$fp)
Y	↔	-4(\$fp)
Z	↔	-8(\$fp)
j	↔	-12(\$fp)
k	↔	-16(\$fp)
t0	↔	-20(\$fp)
...		
t9	↔	-56(\$fp)
s0	↔	-60(\$fp)
s7	↔	-88(\$fp)
gp	↔	-92(\$fp)
sp	↔	-96(\$fp)
fp	↔	-100(\$fp)
ra	↔	-104(\$fp)

Client

X
Y
Z
j
k
t0
...
s0
...
gp
sp
fp
ra

- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - ➡ ○ build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but $sp = fp + 1$
 - position the return value: (\$fp), \$v0
 - jr \$ra # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - $-4(\$sp)$ or \$v0



```
int sub(int X, int Y, int Z) {  
    int j;  
    int k = Y + Z  
  
    j = sub(1, k, 3);  
    ;  
    return j;  
}
```

```
X ⇔ 0($fp)
Y ⇔ -4($fp)
Z ⇔ -8($fp)
j ⇔ -12($fp)
k ⇔ -16($fp)
t0 ⇔ -20($fp)
...
t9 ⇔ -56($fp)
s0 ⇔ -60($fp)
s7 ⇔ -88($fp)
gp ⇔ -92($fp)
sp ⇔ -96($fp)
fp ⇔ -100($fp)
ra ⇔ -104($fp)
```

Client

X
Y
Z
j
k
t0
...
s0
...
gp
sp
fp
ra

- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - ➡ ○ save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but $sp = fp + 1$
 - position the return value: (\$fp), \$v0
 - jr \$ra # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - $-4(\$sp)$ or \$v0

Client	...	args
	fp	
	ra	
Producer	1	locals
	k	
	3	
		temps
	s0	
	...	
sp:		

```
int sub(int X, int Y, int Z) {  
    int j;  
    int k = Y + Z  
  
    j = sub(1, k, 3);  
    ;  
    return j;  
}
```

```
X ⇔ 0($fp)
Y ⇔ -4($fp)
Z ⇔ -8($fp)
j ⇔ -12($fp)
k ⇔ -16($fp)
t0 ⇔ -20($fp)
...
t9 ⇔ -56($fp)
s0 ⇔ -60($fp)
s7 ⇔ -88($fp)
gp ⇔ -92($fp)
sp ⇔ -96($fp)
fp ⇔ -100($fp)
ra ⇔ -104($fp)
```

Frame-variable mapping

Client	X
	Y
	Z
	j
	k
	t0
	...
	s0
	...
	gp
	sp
	fp
	ra

- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but $sp = fp + 1$
 - position the return value: ($\$fp$), $\$v0$
 - jr $\$ra$ # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - $-4(\$sp)$ or $\$v0$

Client	...
	fp
	ra
<hr/>	
fp:	1
	k
	3
Producer	j
	k
	t0
	...
	s0
	...
	gp
	sp
	fp
	ra
sp:	

args

locals

temps

```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

    j = sub(1, k, 3);
    ;
    return j;
}
```

```
X ⇔ 0($fp)
Y ⇔ -4($fp)
Z ⇔ -8($fp)
j ⇔ -12($fp)
k ⇔ -16($fp)
t0 ⇔ -20($fp)
...
t9 ⇔ -56($fp)
s0 ⇔ -60($fp)
s7 ⇔ -88($fp)
gp ⇔ -92($fp)
sp ⇔ -96($fp)
fp ⇔ -100($fp)
ra ⇔ -104($fp)
```


Executing "sub"

Client	X
	Y
	Z
	j
	k
	t0
	...
	s0
	...
	gp
	sp
	fp
	ra

- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but $sp = fp + 1$
 - position the return value: ($\$fp$), $\$v0$
 - jr $\$ra$ # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - $-4(\$sp)$ or $\$v0$

Client	...
	fp
	ra
<hr/>	
fp:	1
	k
	3
Producer	j
	k
	t0
	...
	s0
	...
	gp
	sp
	fp
	ra
sp:	

args

locals

temps

```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z
    {
        j = sub(1, k, 3);
    }
    return j;
}
```

X ⇔ 0(\$fp)
 Y ⇔ -4(\$fp)
 Z ⇔ -8(\$fp)
 j ⇔ -12(\$fp)
 k ⇔ -16(\$fp)
 t0 ⇔ -20(\$fp)
 ...
 t9 ⇔ -56(\$fp)
 s0 ⇔ -60(\$fp)
 s7 ⇔ -88(\$fp)
 gp ⇔ -92(\$fp)
 sp ⇔ -96(\$fp)
 fp ⇔ -100(\$fp)
 ra ⇔ -104(\$fp)

Returning from "sub"

Client

X
Y
Z
j
k
t0
...
s0
...
gp
sp
fp
ra

- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but $sp = fp + 1$
 - position the return value: (\$fp), \$v0
 - jr \$ra # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - $-4(\$sp)$ or $\$v0$

Client

fp:

Producer

sp:

...
fp
ra
1
k
3
j
k
t0
...
s0
...
gp
sp
fp
ra

args

locals

temps

```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

    j = sub(1, k, 3);
    ;
    → return j;
}
```

X ⇔ 0(\$fp)
 Y ⇔ -4(\$fp)
 Z ⇔ -8(\$fp)
 j ⇔ -12(\$fp)
 k ⇔ -16(\$fp)
 t0 ⇔ -20(\$fp)
 ...
 t9 ⇔ -56(\$fp)
 s0 ⇔ -60(\$fp)
 s7 ⇔ -88(\$fp)
 gp ⇔ -92(\$fp)
 sp ⇔ -96(\$fp)
 fp ⇔ -100(\$fp)
 ra ⇔ -104(\$fp)

Returning from "sub"

Client	X
	Y
	Z
	j
	k
	t0
	...
	s0
	...
	gp
	sp
	fp
	ra

- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but $sp = fp + 1$
 - position the return value: ($\$fp$), $\$v0$
 - jr $\$ra$ # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - $-4(\$sp)$ or $\$v0$

Client	...
	fp
	ra
<hr/>	
fp:	1
	k
	3
Producer	j
	k
	t0
	...
	s0
	...
	gp
	sp
	fp
	ra
sp:	

args

locals

temps

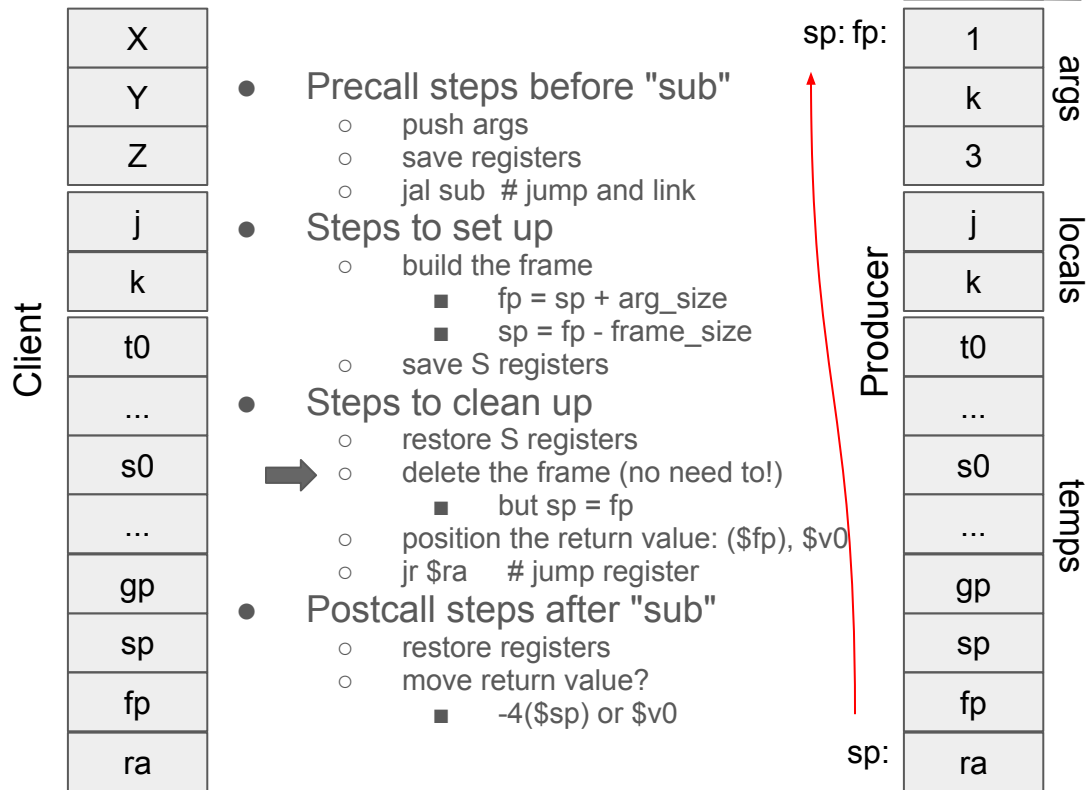
```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

    j = sub(1, k, 3);
    ;
    → return j;
}
```

X ⇔ 0(\$fp)
 Y ⇔ -4(\$fp)
 Z ⇔ -8(\$fp)
 j ⇔ -12(\$fp)
 k ⇔ -16(\$fp)
 t0 ⇔ -20(\$fp)
 ...
 t9 ⇔ -56(\$fp)
 s0 ⇔ -60(\$fp)
 s7 ⇔ -88(\$fp)
 gp ⇔ -92(\$fp)
 sp ⇔ -96(\$fp)
 fp ⇔ -100(\$fp)
 ra ⇔ -104(\$fp)

lw \$s0, -60(\$fp)
 lw \$s1, -64(\$fp)
 lw \$s2, -68(\$fp)
 ...
 lw \$s7, -88(\$fp)

Returning from "sub"



```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

    j = sub(1, k, 3);
    ;
    → return j;
}
```

X	↔	0(\$fp)
Y	↔	-4(\$fp)
Z	↔	-8(\$fp)
j	↔	-12(\$fp)
k	↔	-16(\$fp)
t0	↔	-20(\$fp)
...		
t9	↔	-56(\$fp)
s0	↔	-60(\$fp)
s7	↔	-88(\$fp)
gp	↔	-92(\$fp)
sp	↔	-96(\$fp)
fp	↔	-100(\$fp)
ra	↔	-104(\$fp)

Returning from "sub"

Client	X
	Y
	Z
	j
	k
	t0
	...
	s0
	...
	gp
	sp
	fp
	ra

- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but $sp = fp + 1$
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - $-4(\$sp)$ or $\$v0$

Client	...		
	fp		
	ra		
sp: fp:			
Produ	j	args	
	k		
	3		
	j	locals	
	k		
	cer	t0	temps
	...		
	s0		
	...		
	gp		
	sp		
	fp		
	ra		
\$v0			

```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

    j = sub(1, k, 3);
    ;
    → return j;
}
```

X	↔	0(\$fp)
Y	↔	-4(\$fp)
Z	↔	-8(\$fp)
j	↔	-12(\$fp)
k	↔	-16(\$fp)
t0	↔	-20(\$fp)
...		
t9	↔	-56(\$fp)
s0	↔	-60(\$fp)
s7	↔	-88(\$fp)
gp	↔	-92(\$fp)
sp	↔	-96(\$fp)
fp	↔	-100(\$fp)
ra	↔	-104(\$fp)

Transition back

Client	X
	Y
	Z
	j
	k
	t0
	...
	s0
	...
	gp
	sp
	fp
	ra

- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but $sp = fp + 1$
 - position the return value: (\$fp), \$v0
 - jr \$ra # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - $-4(\$sp)$ or \$v0

Client	...
	fp
	sp: ra
fp:	i
	k
	3
j	j
	k
	t0
...	...
	s0
	...
gp	gp
	sp
	fp
sp:	ra

```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

    j = sub(1, k, 3);
    ;
    → return j;
}
```

X	↔	0(\$fp)
Y	↔	-4(\$fp)
Z	↔	-8(\$fp)
j	↔	-12(\$fp)
k	↔	-16(\$fp)
t0	↔	-20(\$fp)
...		
t9	↔	-56(\$fp)
s0	↔	-60(\$fp)
s7	↔	-88(\$fp)
gp	↔	-92(\$fp)
sp	↔	-96(\$fp)
fp	↔	-100(\$fp)
ra	↔	-104(\$fp)

Client: The Postcall

Client	X
	Y
	Z
	j
	k
	t0
	...
	s0
	...
	gp
	sp
	fp
	ra

- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but $sp = fp + 1$
 - position the return value: ($\$fp$), $\$v0$
 - jr $\$ra$ # jump register



Postcall steps after "sub"

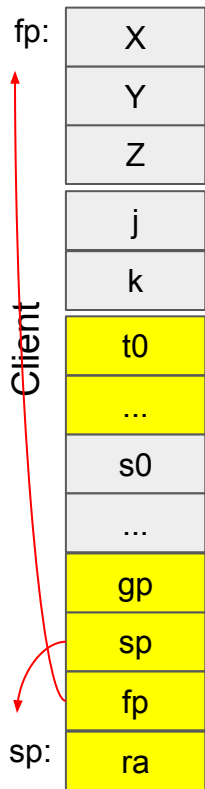
- restore registers
- move return value?
 - $-4(\$sp)$ or $\$v0$

Client	...
	fp
	sp: ra
args	fp: j
	k
	Z
locals	j
	k
	t0
temp	...
	s0
	...
temp	gp
	sp
	fp
sp:	ra

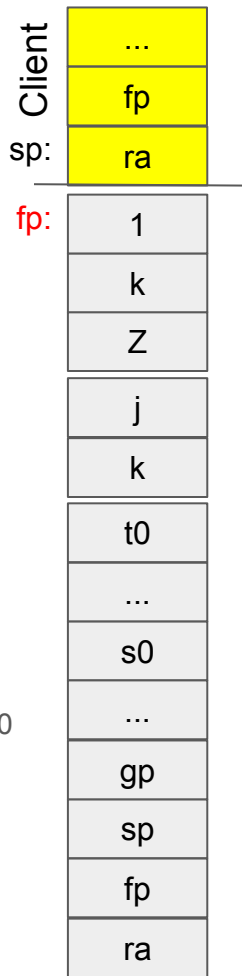
```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z
    j = sub(1, k, 3);
    return j;
}
```

X	↔	0(\$fp)
Y	↔	-4(\$fp)
Z	↔	-8(\$fp)
j	↔	-12(\$fp)
k	↔	-16(\$fp)
t0	↔	-20(\$fp)
...		
t9	↔	-56(\$fp)
s0	↔	-60(\$fp)
s7	↔	-88(\$fp)
gp	↔	-92(\$fp)
sp	↔	-96(\$fp)
fp	↔	-100(\$fp)
ra	↔	-104(\$fp)

Client: The set up



- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - $fp = sp + arg_size$
 - $sp = fp - frame_size$
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but $sp = fp + 1$
 - position the return value: ($\$fp$), $\$v0$
 - jr $\$ra$ # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - $-4(\$sp)$ or $\$v0$

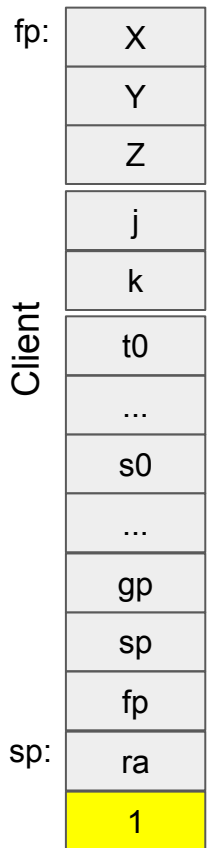


```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z
    j = sub(1, k, 3);
    return j;
}
```

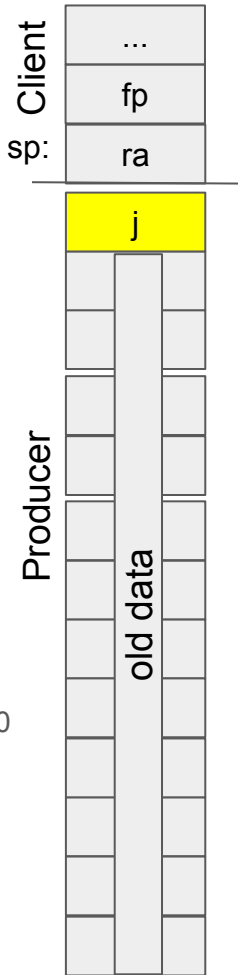
X \Leftrightarrow 0($\$fp$)
 Y \Leftrightarrow -4($\$fp$)
 Z \Leftrightarrow -8($\$fp$)
 j \Leftrightarrow -12($\$fp$)
 k \Leftrightarrow -16($\$fp$)
 t0 \Leftrightarrow -20($\$fp$)
 ...
 t9 \Leftrightarrow -56($\$fp$)
 s0 \Leftrightarrow -60($\$fp$)
 s7 \Leftrightarrow -88($\$fp$)
 gp \Leftrightarrow -92($\$fp$)
 sp \Leftrightarrow -96($\$fp$)
 fp \Leftrightarrow -100($\$fp$)
 ra \Leftrightarrow -104($\$fp$)

```
lw $sp, 4( $\$fp$ )
lw $fp, 4( $\$sp$ )
-----
lw $t0, -20( $\$fp$ )
lw $t1, -24( $\$fp$ )
...
lw $ra, -104( $\$fp$ )
```


Client: The Postcall



- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - fp = sp + arg_size
 - sp = fp - frame_size
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but sp = fp + 1
 - position the return value: (\$fp), \$v0
 - jr \$ra # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - -4(\$sp) or \$v0



```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z
    j = sub(1, k, 3);
    ;
    return j;
}
```

```
X ⇔ 0($fp)
Y ⇔ -4($fp)
Z ⇔ -8($fp)
j ⇔ -12($fp)
k ⇔ -16($fp)
t0 ⇔ -20($fp)
...
t9 ⇔ -56($fp)
s0 ⇔ -60($fp)
s7 ⇔ -88($fp)
gp ⇔ -92($fp)
sp ⇔ -96($fp)
fp ⇔ -100($fp)
ra ⇔ -104($fp)
```

The Next Instruction:



- Precall steps before "sub"
 - push args
 - save registers
 - jal sub # jump and link
- Steps to set up
 - build the frame
 - fp = sp + arg_size
 - sp = fp - frame_size
 - save S registers
- Steps to clean up
 - restore S registers
 - delete the frame (no need to!)
 - but sp = fp + 1
 - position the return value: (\$fp), \$v0
 - jr \$ra # jump register
- Postcall steps after "sub"
 - restore registers
 - move return value?
 - -4(\$sp) or \$v0

```
int sub(int X, int Y, int Z) {
    int j;
    int k = Y + Z

    j = sub(1, k, 3);
    → ;
    return j;
}
```

X	↔	0(\$fp)
Y	↔	-4(\$fp)
Z	↔	-8(\$fp)
j	↔	-12(\$fp)
k	↔	-16(\$fp)
t0	↔	-20(\$fp)
...		
t9	↔	-56(\$fp)
s0	↔	-60(\$fp)
s7	↔	-88(\$fp)
gp	↔	-92(\$fp)
sp	↔	-96(\$fp)
fp	↔	-100(\$fp)
ra	↔	-104(\$fp)

Client -- Producer Convention Caveats:

- Main Memory is slow:
 - first 4 arguments should not be passed via the stack but via: \$a0, \$a1, \$a2, \$a3
 - the 2 return values should not be passed via the stack but via: \$v0, \$v1
- Although there are 32 general purpose registers:
 - Can't use: \$zero, \$at, \$k1, \$k2
 - If you use: \$gp, \$sp, \$fp, \$ra
 - you must take steps to save--restore these registers at call boundaries
 - if you use: \$a0, \$a1, \$a2, \$a3, \$v0, \$v1
 - you must take steps to save--restore these registers at call boundaries
- A compiler **MUST** follow this convention,
 - but the assembly level programmer can "optimize" there code!