M2 – Instruction Set Architecture

Module Outline

- Addressing modes. Instruction classes.
- MIPS-I ISA.
- Translating and starting a program.
- High level languages, Assembly languages and object code.
- Subroutine and subroutine call. Use of stack for handling subroutine call and return.

Subroutine Calls

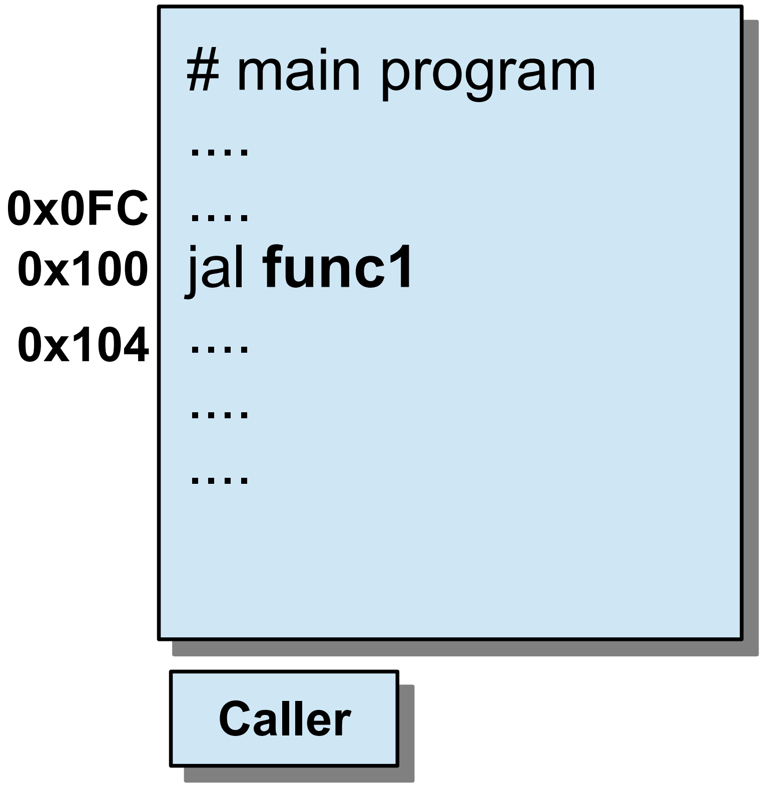
Subroutines in MIPS

- Subroutine Call – **jal *subname***
  - Saves return address in R31 ($ra) and jumps to subroutine entry label subname

Subroutines in MIPS

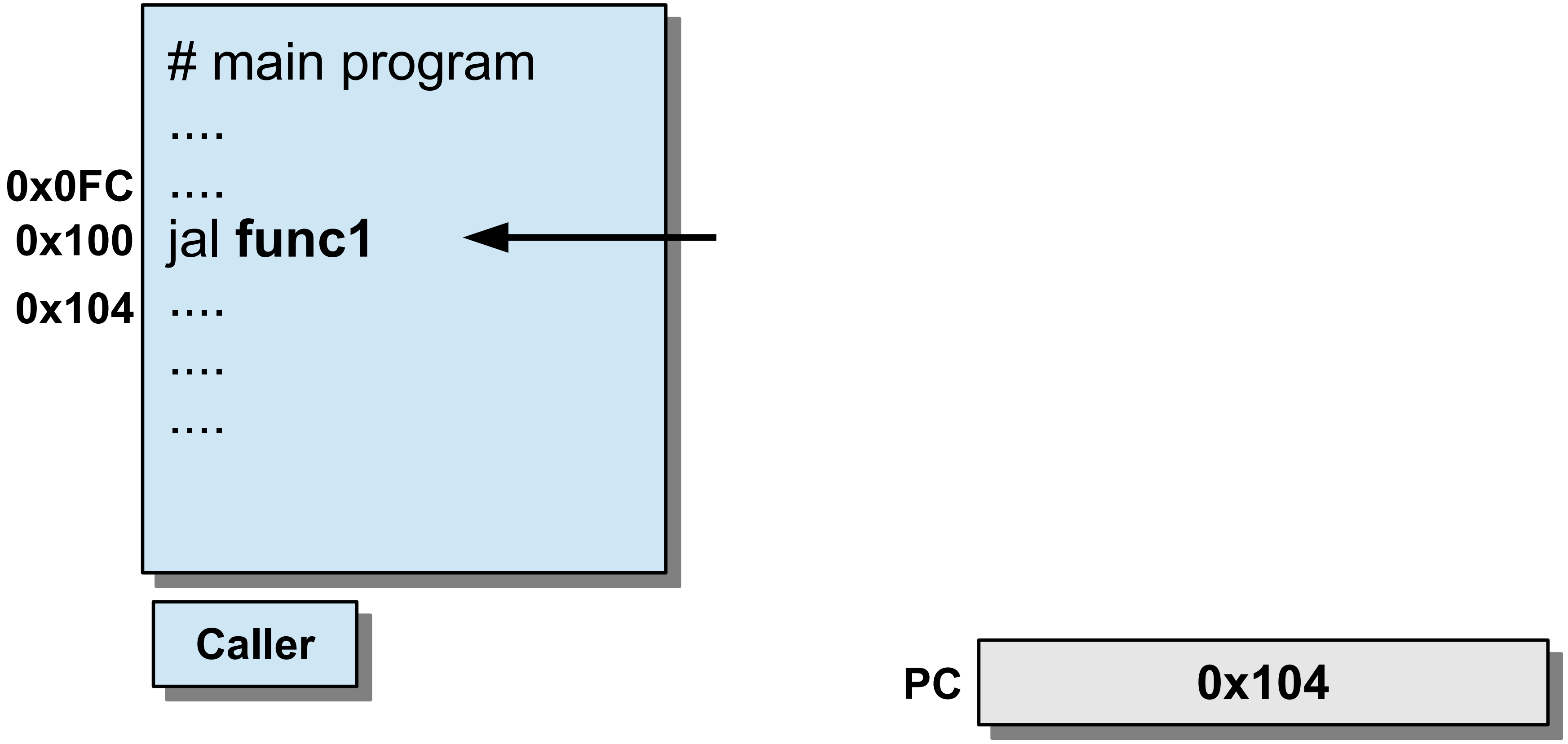
- Subroutine Call – **jal *subname***
  - Saves return address in R31 ($ra) and jumps to subroutine entry label subname
- Subroutine Return – **jr $31**
  - Loads PC with return address in $31

Subroutines in MIPS



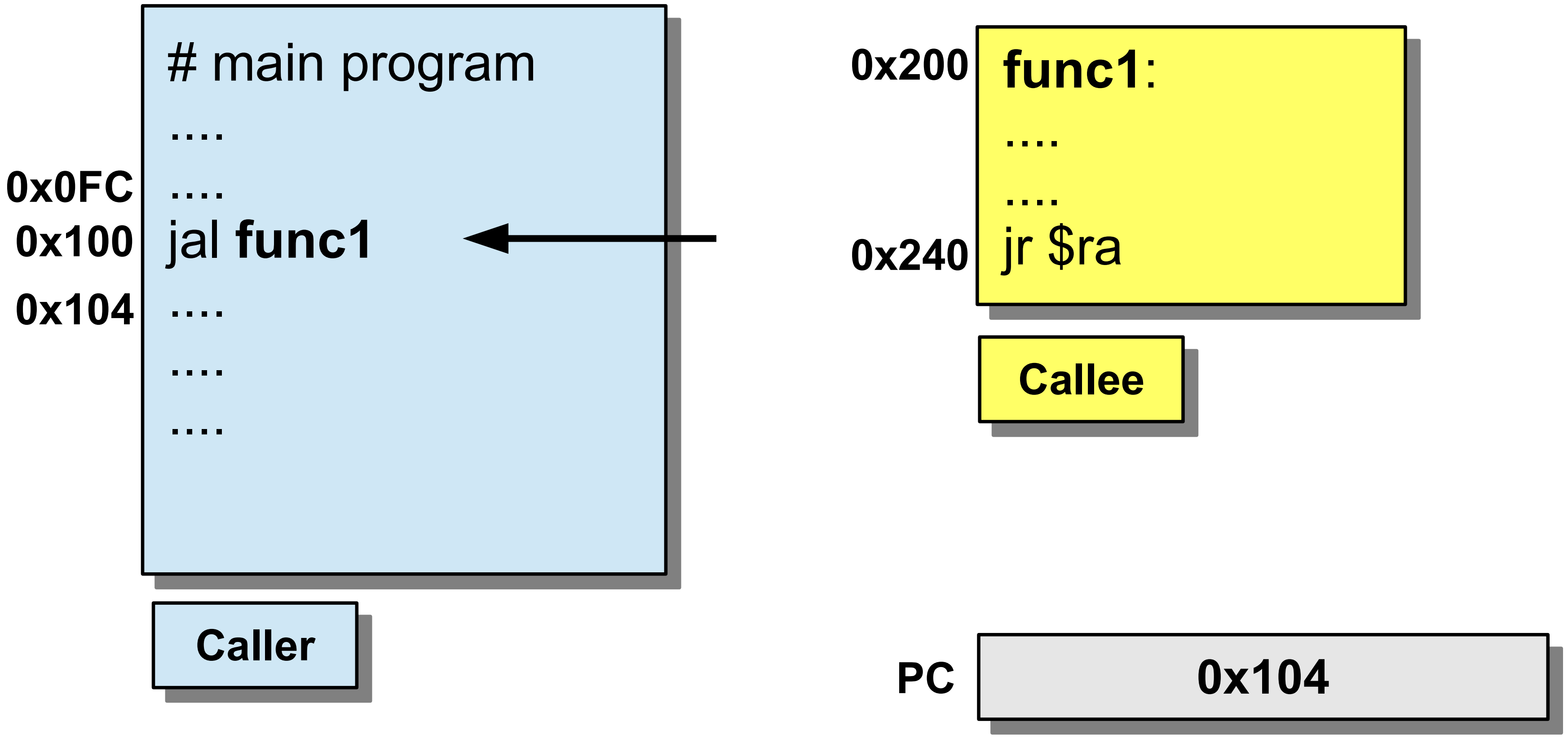
```
       # main program
       ....
0x0FC  ....
0x100  jal func1
0x104  ....
       ....
       ....
```

Subroutines in MIPS

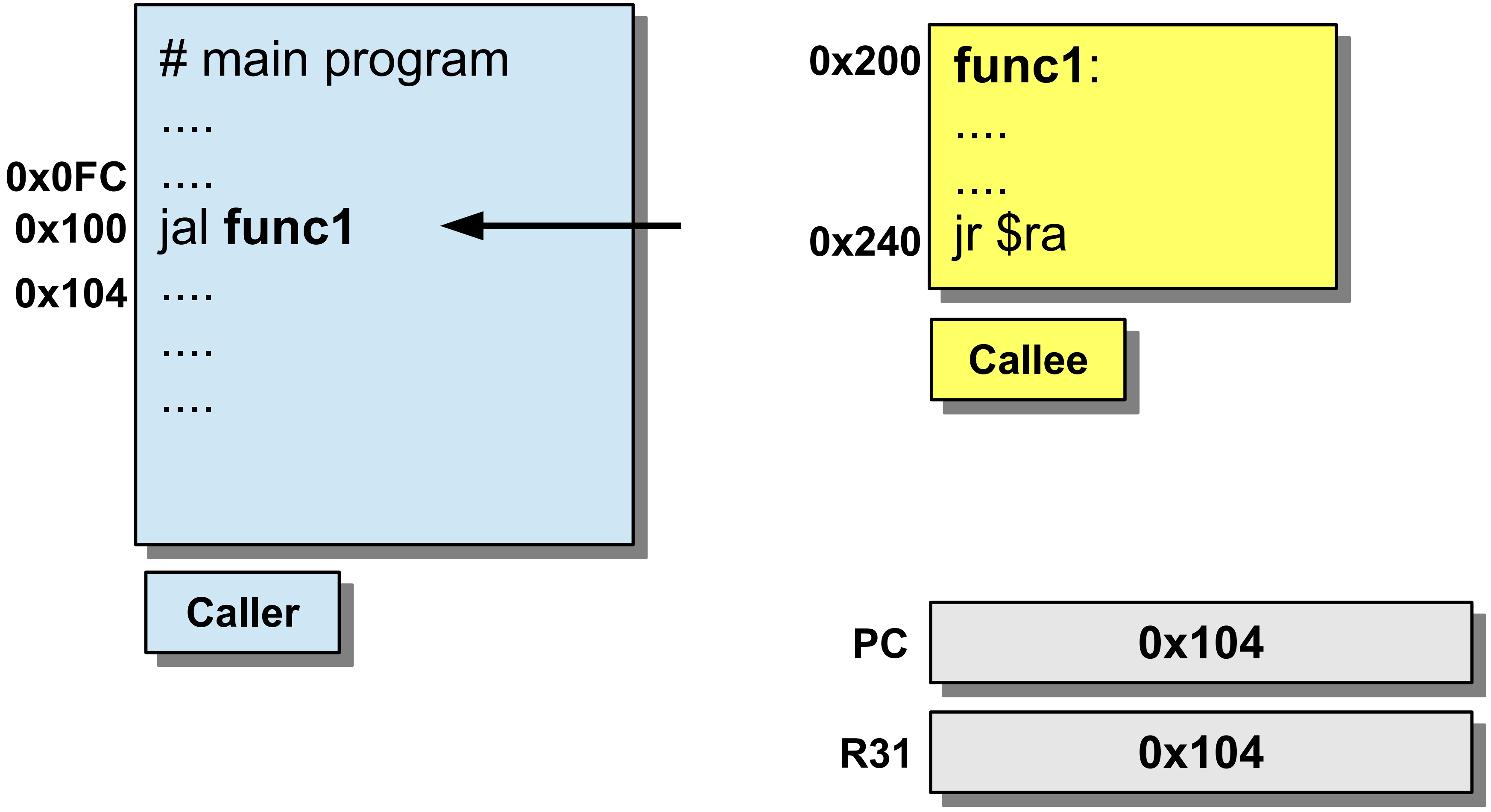


```
      # main program
      ....
0x0FC ....
0x100 jal func1
0x104 ....
      ....
      ....
```

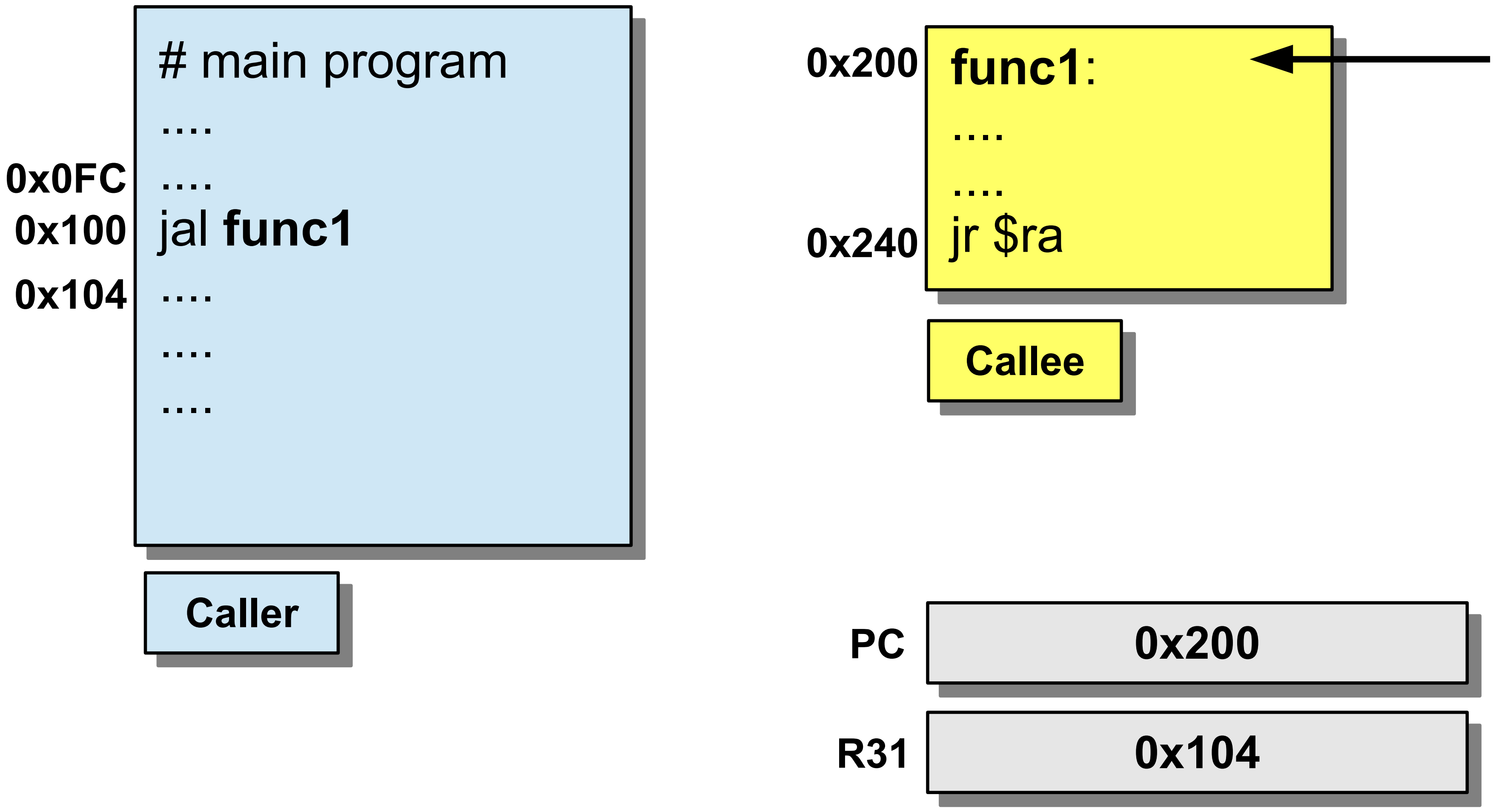
Subroutines in MIPS



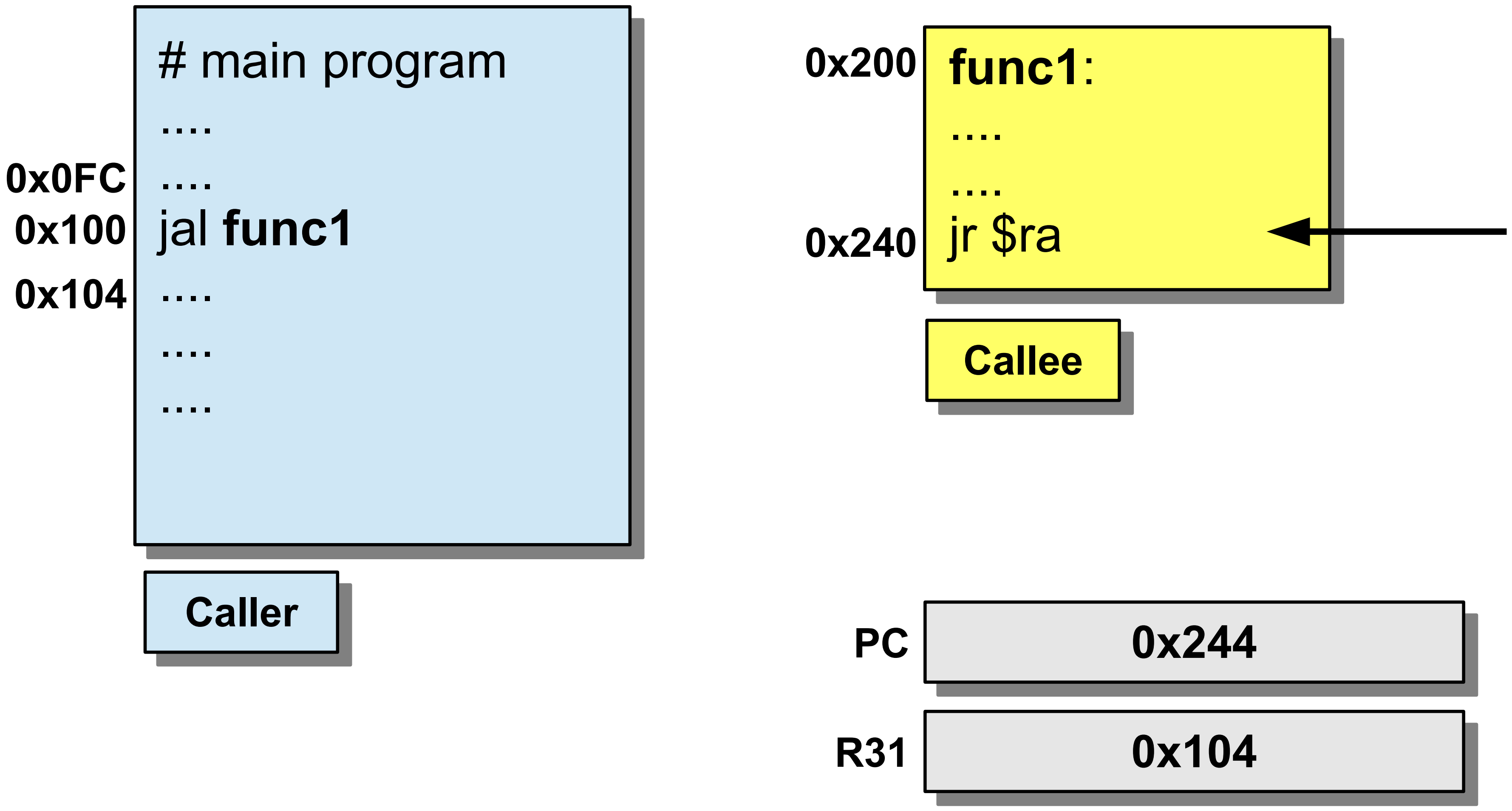
Subroutines in MIPS



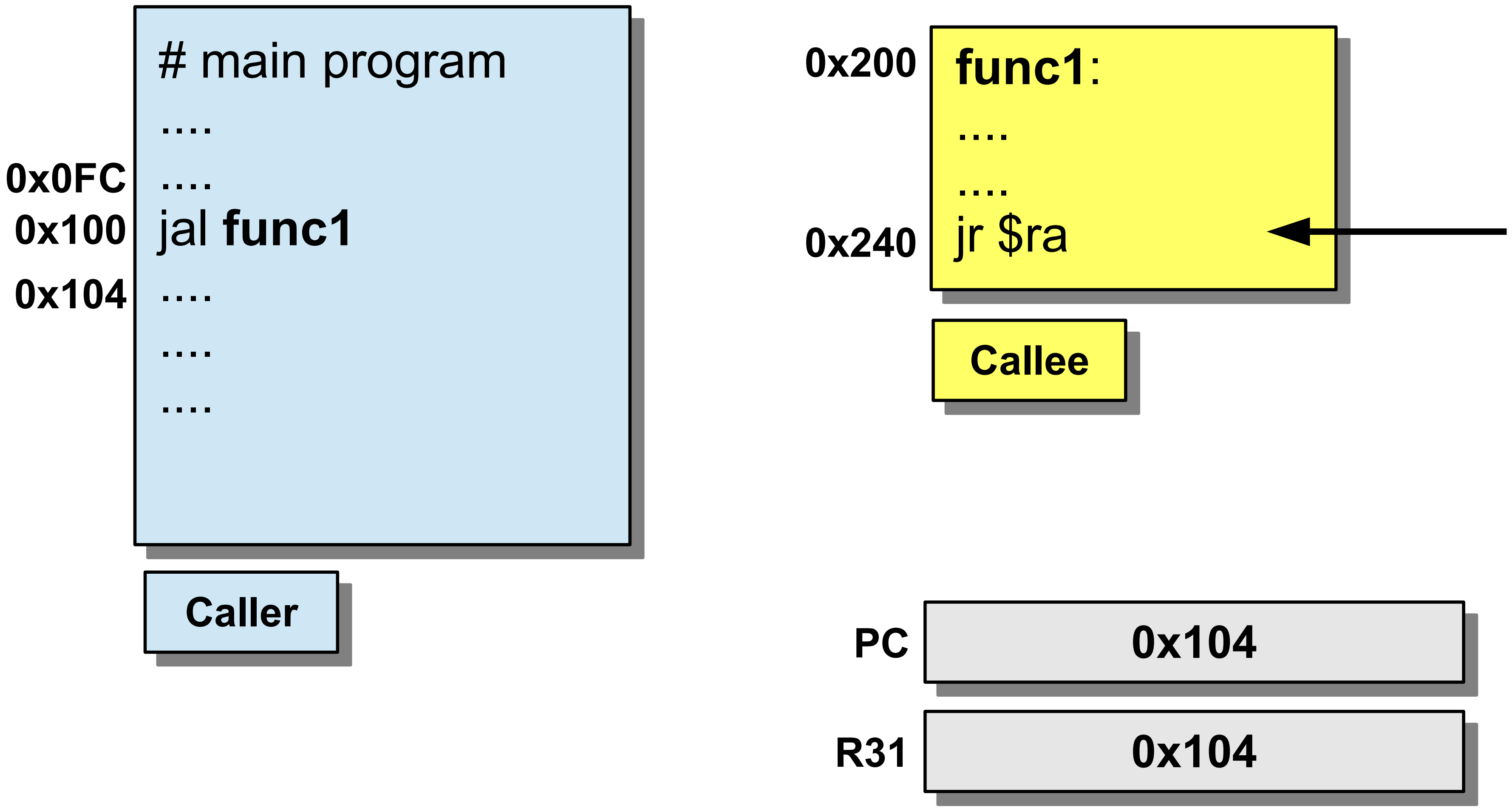
Subroutines in MIPS



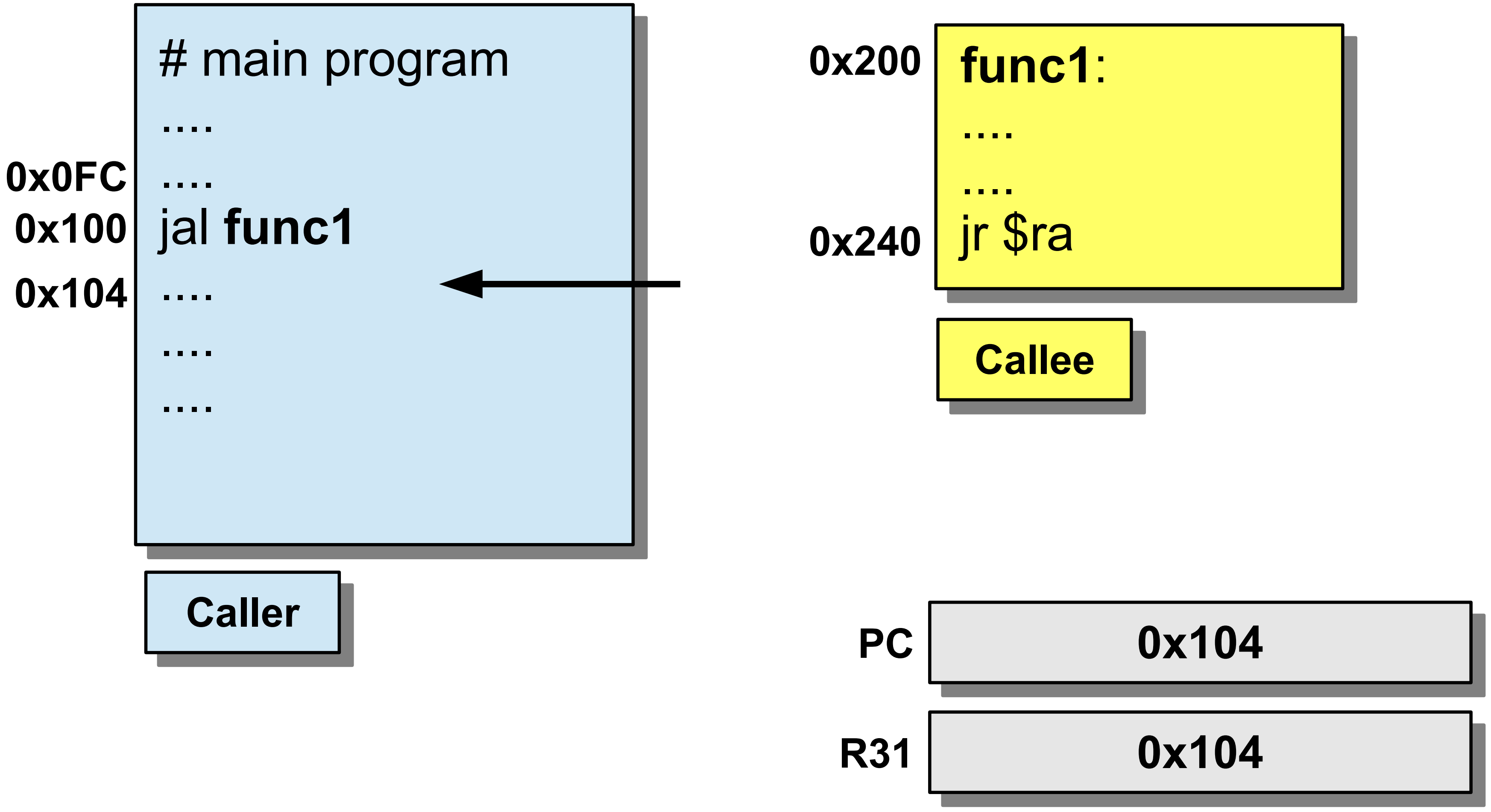
Subroutines in MIPS



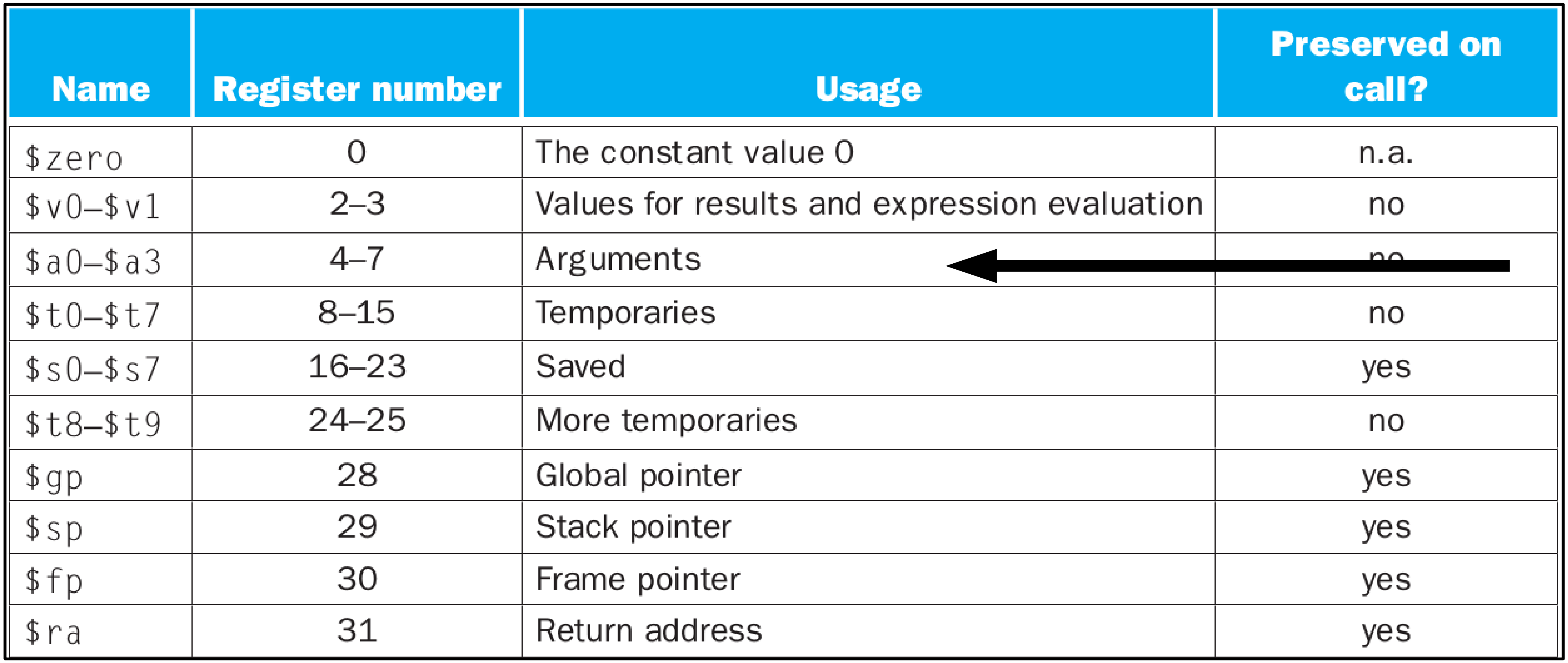
Subroutines in MIPS



Subroutines in MIPS



Registers Usage Convention



Name	Register number	Usage	Preserved on call?
$zero	0	The constant value 0	n.a.
$v0–$v1	2–3	Values for results and expression evaluation	no
$a0–$a3	4–7	Arguments	no
$t0–$t7	8–15	Temporaries	no
$s0–$s7	16–23	Saved	yes
$t8–$t9	24–25	More temporaries	no
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

Subroutines – Parameter Passing

```
# main program
....
....
add R4, R0, R16
add R5, R0, R17
jal func1
....
....
....
```

```
0x200  func1:
       ....
       ....
0x240  jr $ra
```

Subroutines – Parameter Passing

```
# main program
....
....
add $a0, $zero, $s0
add $a1, $zero, $s1
jal accArray
print $v0
....
....
....
```

```
accArray:
add $v0, $zero, $zero
loop:
beq $a0, $zero, done
lw $t0, 0($a1)
add $v0, $v0, $t0
addiu $a1, $a1, 4
addi $a0, $a0, -1
j loop
done:
jr $ra
```

Subroutines – Parameter Passing

- Caller saves parameters in $a0 - $a3

Subroutines – Parameter Passing

- Caller saves parameters in $a0 - $a3
- Callee stores results in $v0, $v1.

Subroutines – Parameter Passing

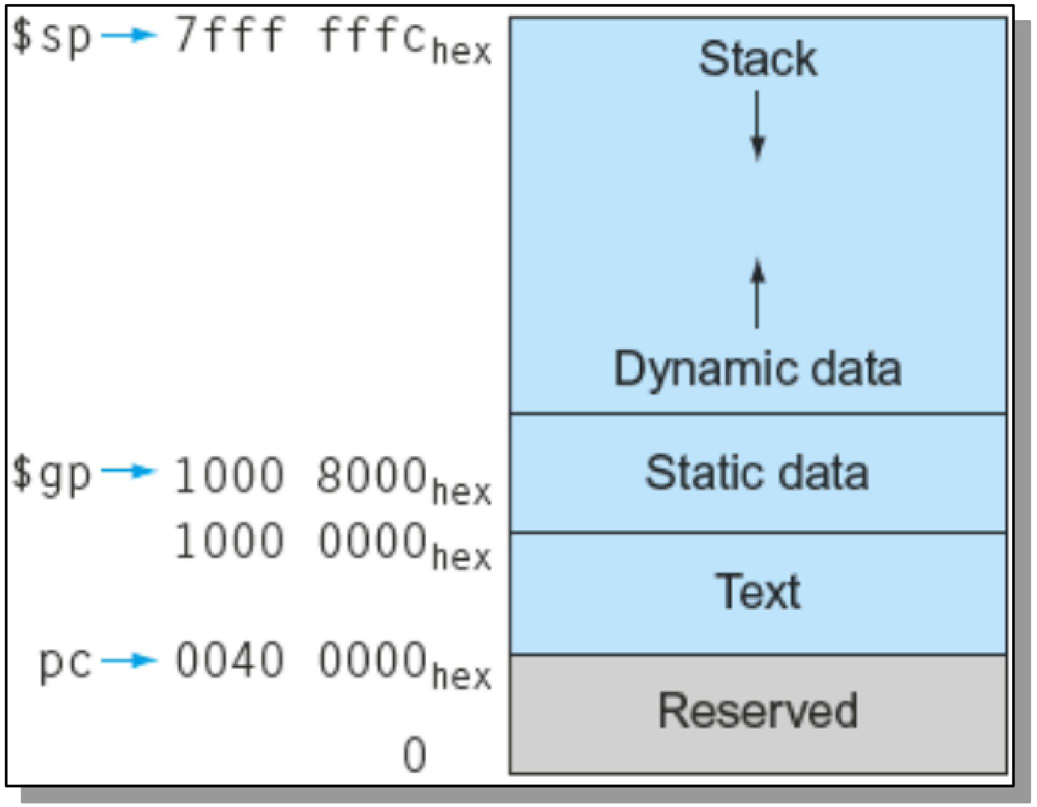
- Caller saves parameters in \$a0 - \$a3
- Callee stores results in \$v0, \$v1.
- How does the caller pass more than 4 parameters to the callee?

Subroutines – Parameter Passing

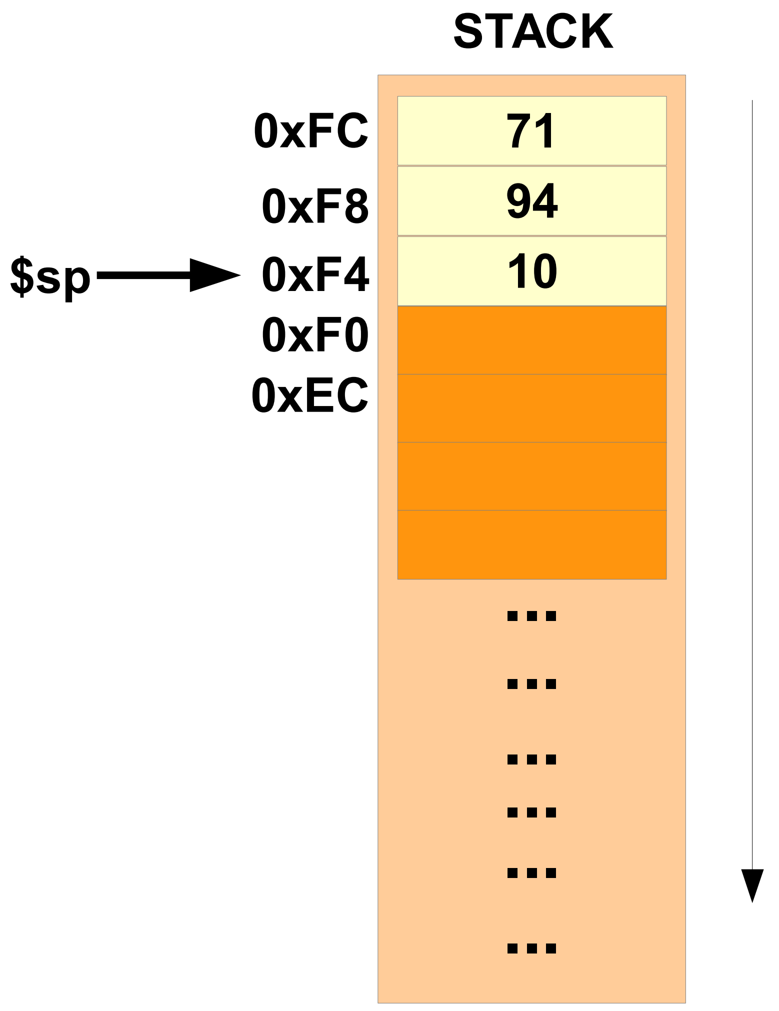
- Caller saves parameters in $a0 - $a3
- Callee stores results in $v0, $v1.
- How does the caller pass more than 4 parameters to the callee?
- Program stack

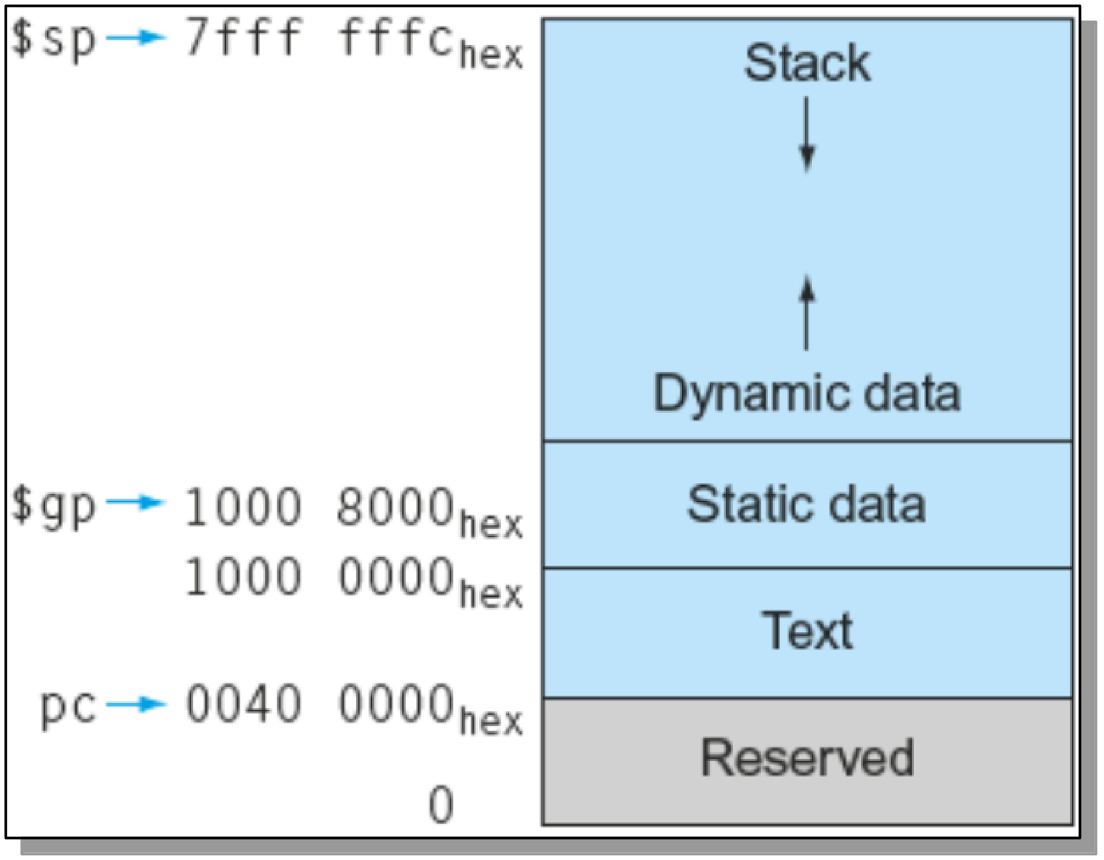
Subroutines – Parameter Passing

- Caller saves parameters in \$a0 - \$a3
- Callee stores results in \$v0, \$v1.
- How does the caller pass more than 4 parameters to the callee?
- Program stack



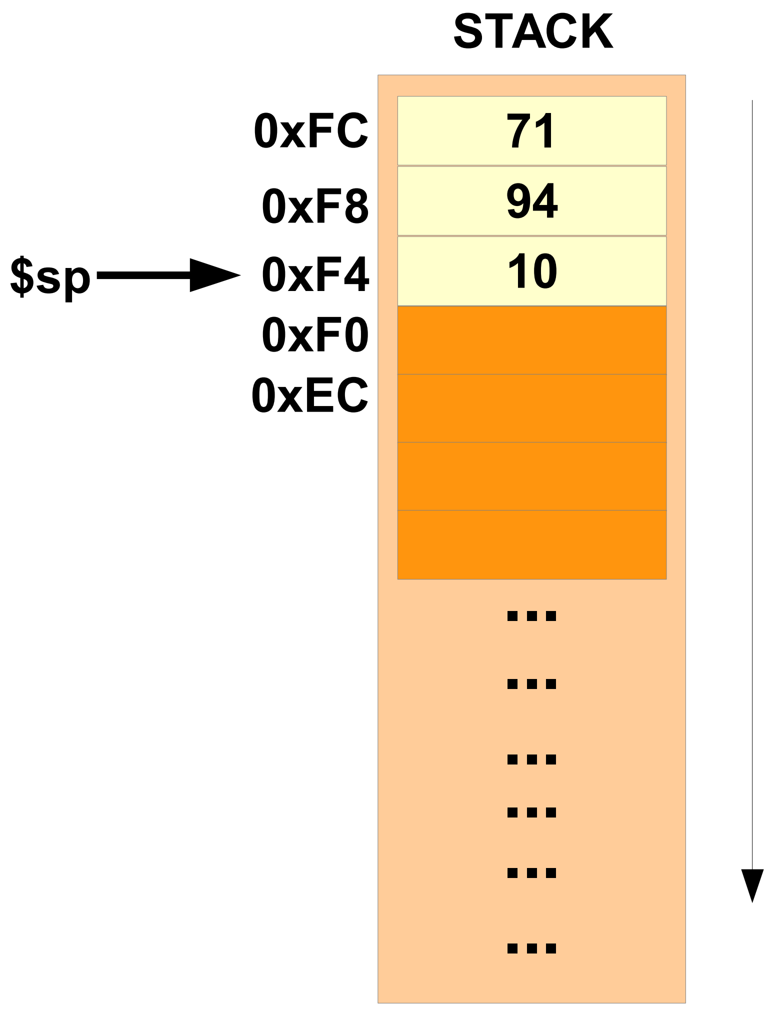
The MIPS Stack



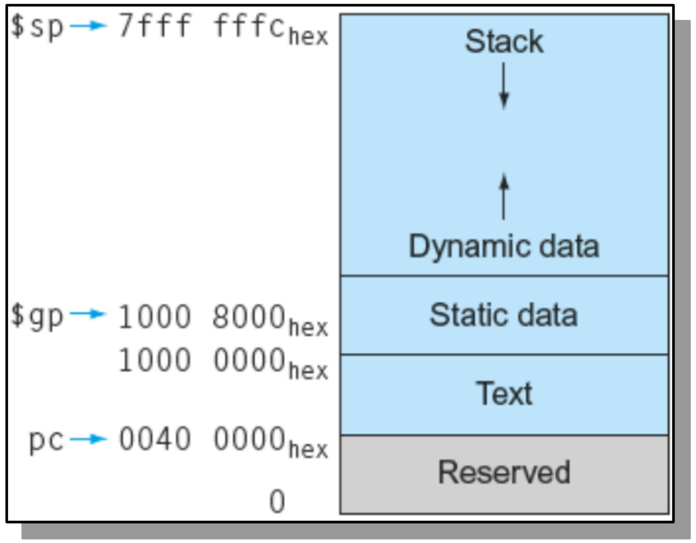


R29 = $sp 0x7FFF FFF4

The MIPS Stack

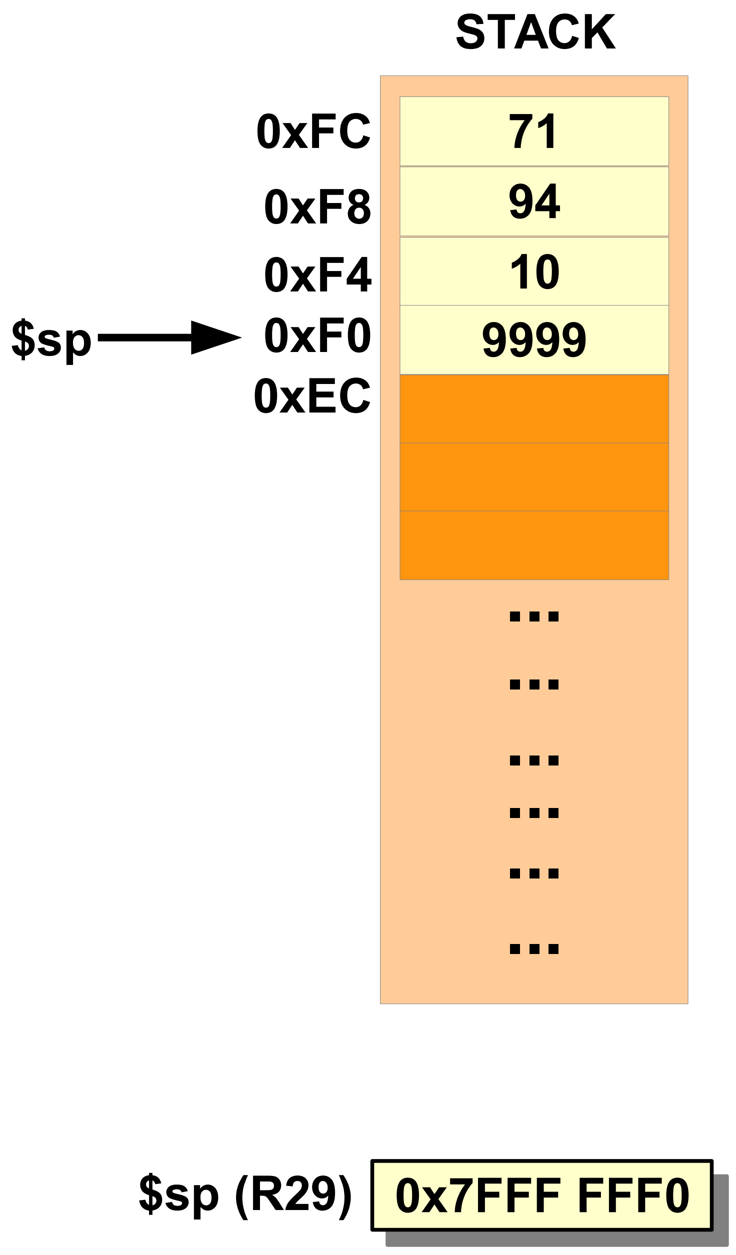


R29 = $sp 0x7FFF FFF4



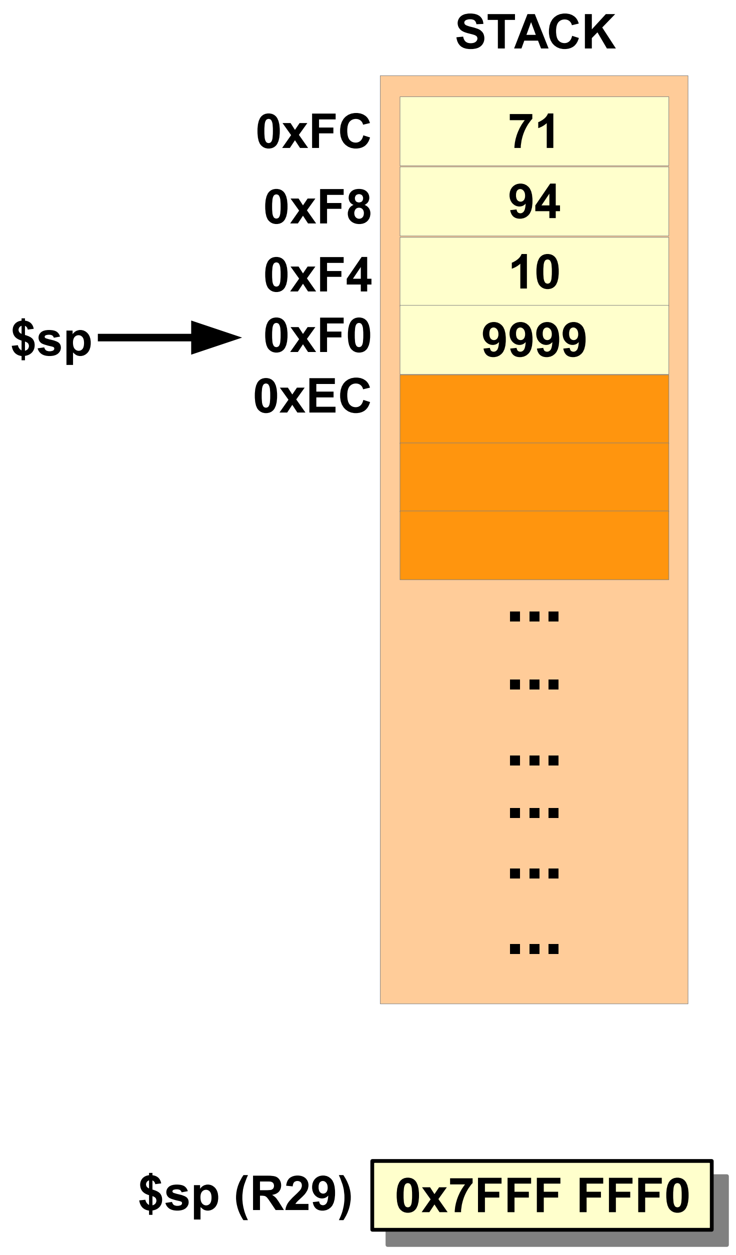
- Push the value in $t0 on the stack

The MIPS Stack



- Push the value in $t0 on the stack

The MIPS Stack

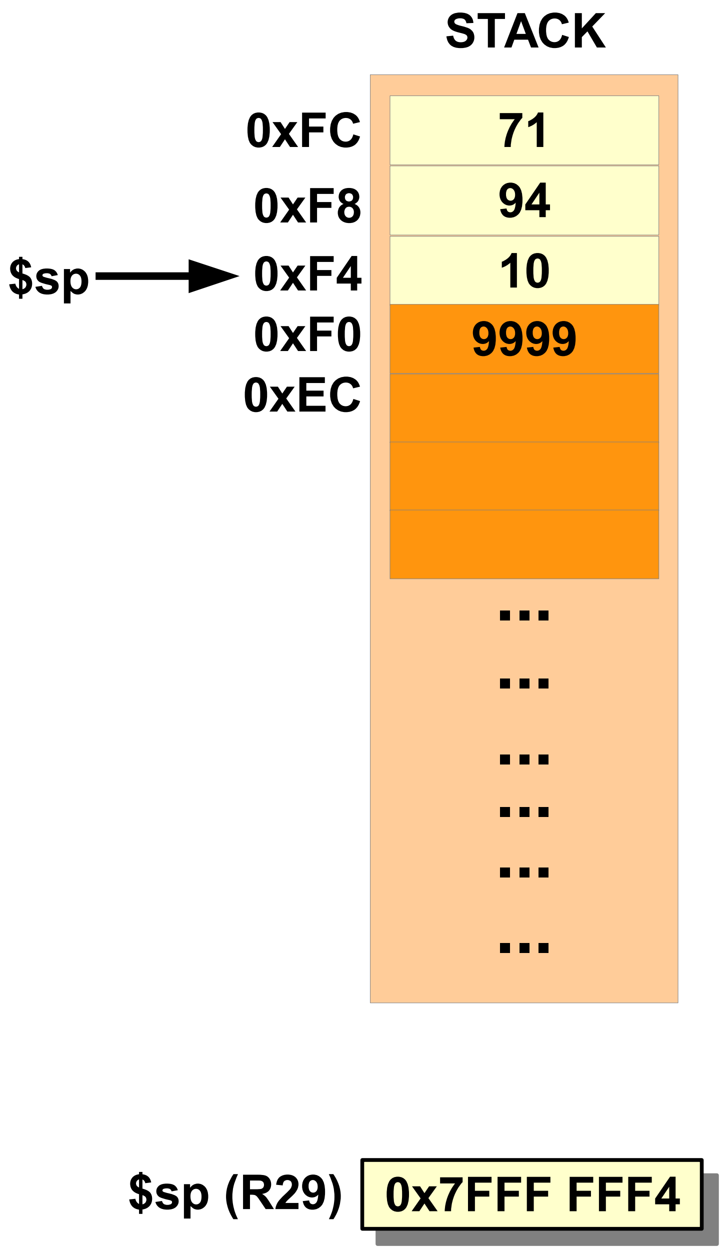


- Push the value in $t0 on the stack

Push

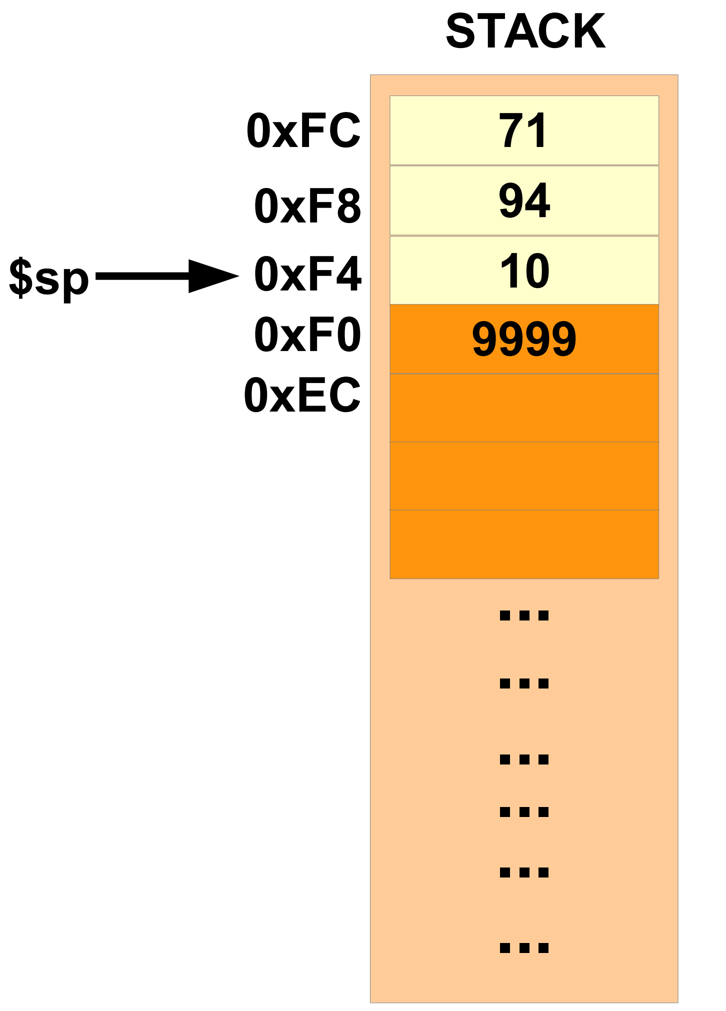
```
addi $sp, $sp, -4
sw $t0, 0($sp)
```

The MIPS Stack



- Pop into $t1

The MIPS Stack



- Pop into $t1

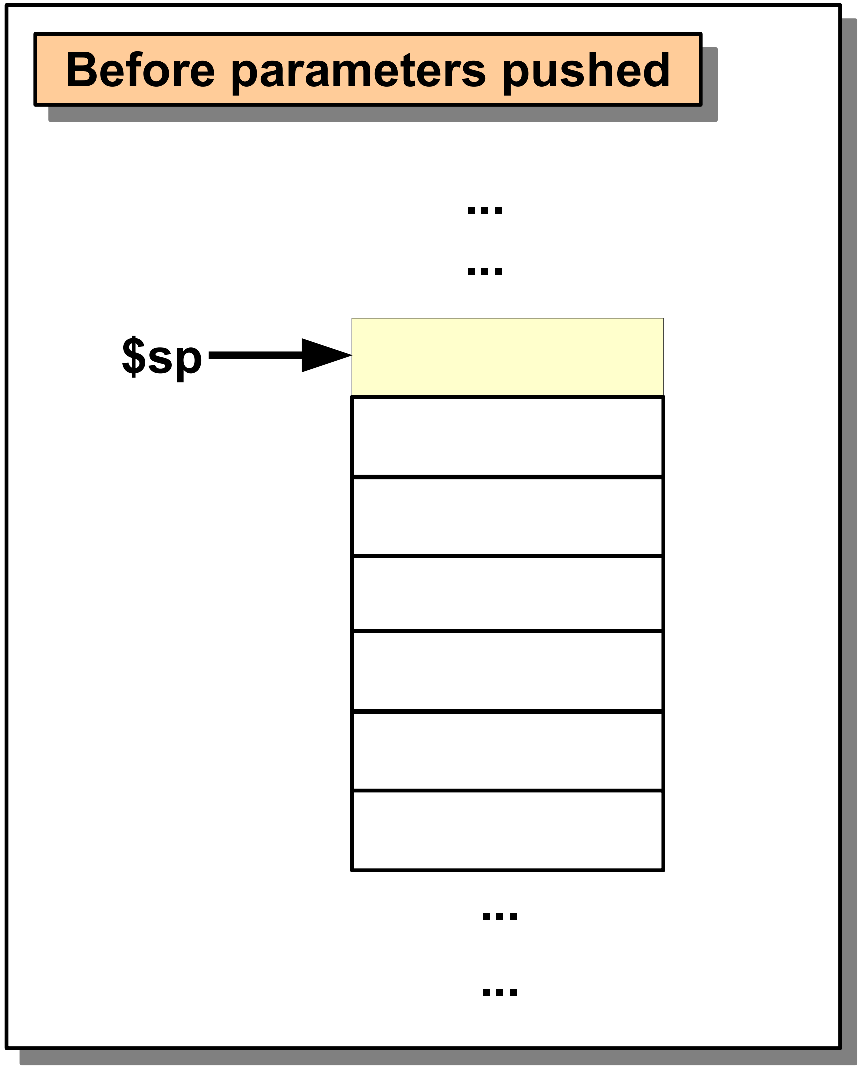
Pop

```
lw $t1, 0($sp)
addi $sp, $sp, +4
```

$sp (R29) **0x7FFF FFF4**

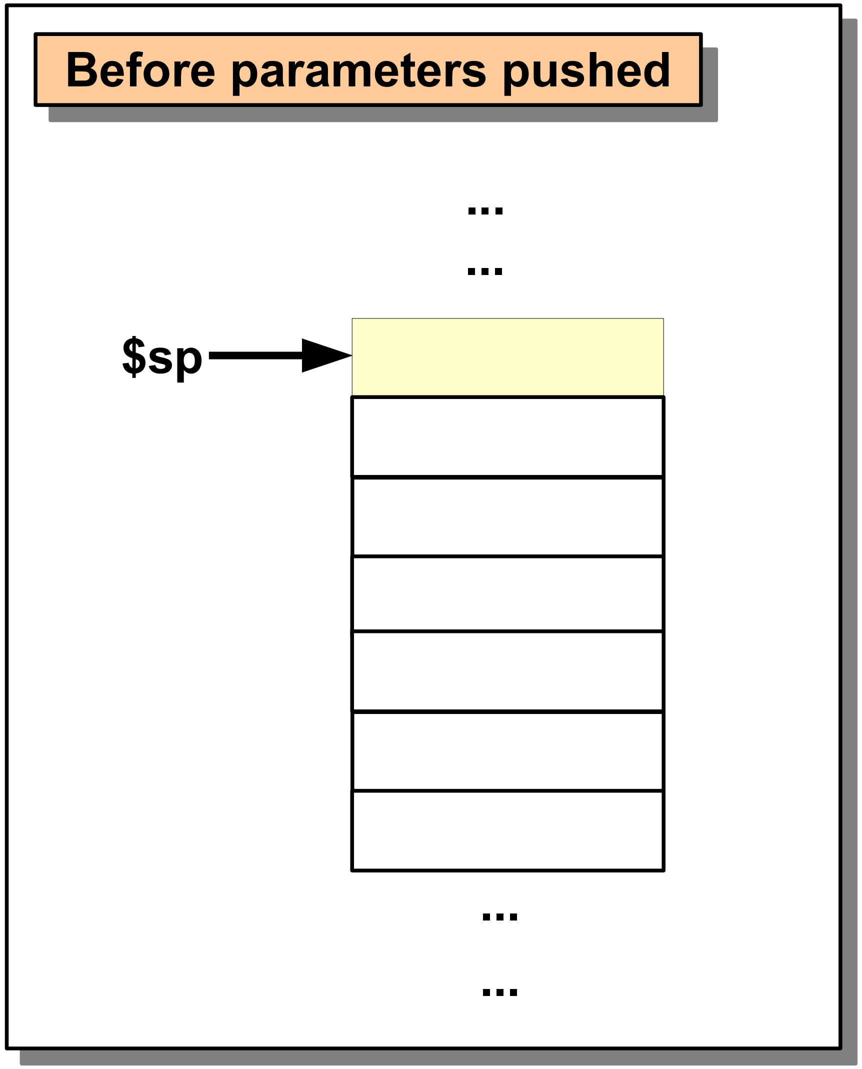
Subroutines – Parameter Passing

```
# main program
# 6 parameters to func1
....
....
...
jal func1
....
....
....
```



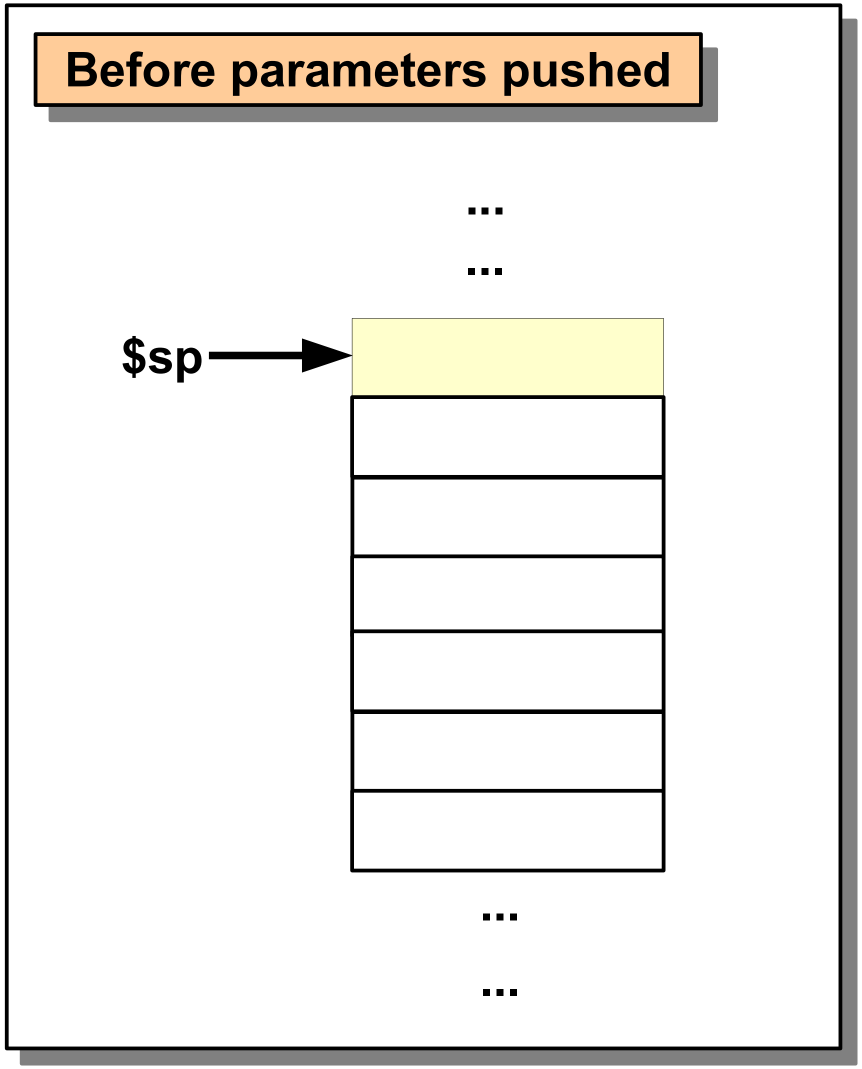
Subroutines – Parameter Passing

```
# main program
# 6 parameters to func1
....
....
# 4 args are in $a0 - $a3
...
# push 2 on stack
...
...
jal func1
....
....
....
```



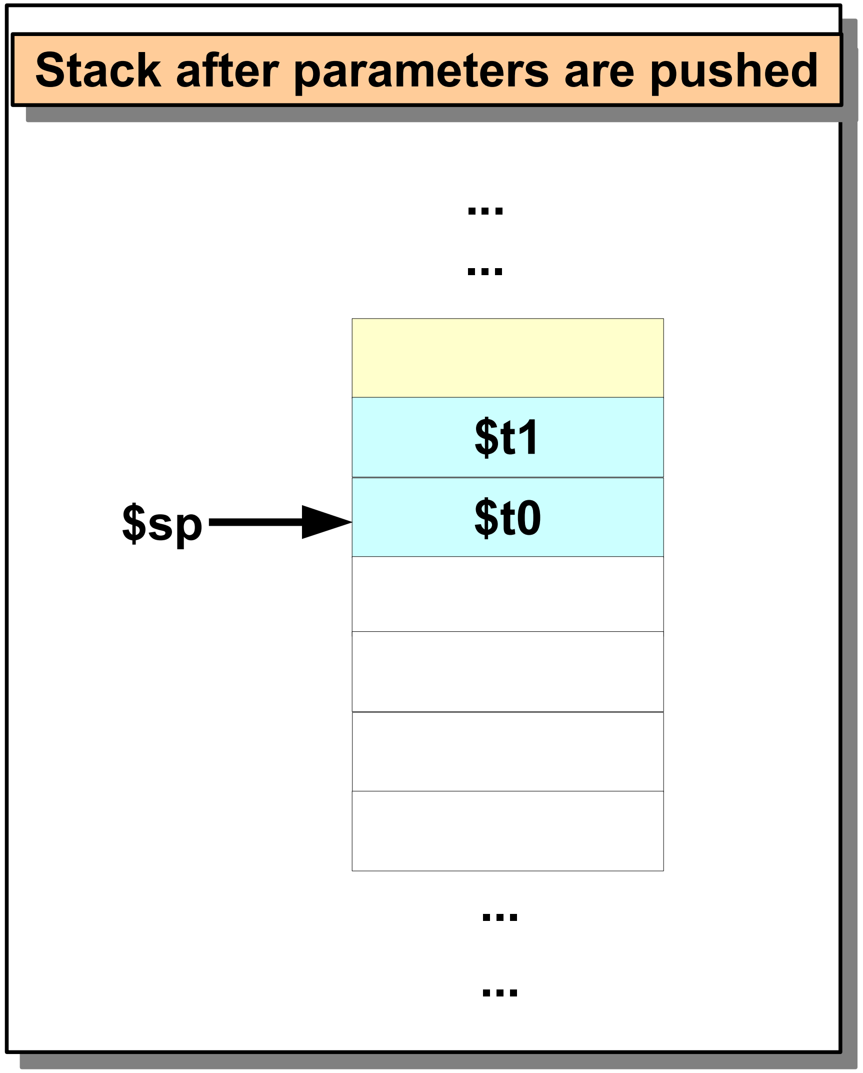
Subroutines – Parameter Passing

```
# main program
# 6 parameters to func1
....
....
# 4 args are in $a0 - $a3
...
# push 2 on stack
addi $sp, $sp, -8
sw $t0, 0($sp)
sw $t1, -4($sp)
jal func1
....
....
....
```



Subroutines – Parameter Passing

```
# main program
# 6 parameters to func1
....
....
# 4 args are in $a0 - $a3
...
# push 2 on stack
addi $sp, $sp, -8
sw $t0, 0($sp)
sw $t1, -4($sp)
jal func1
....
....
....
```

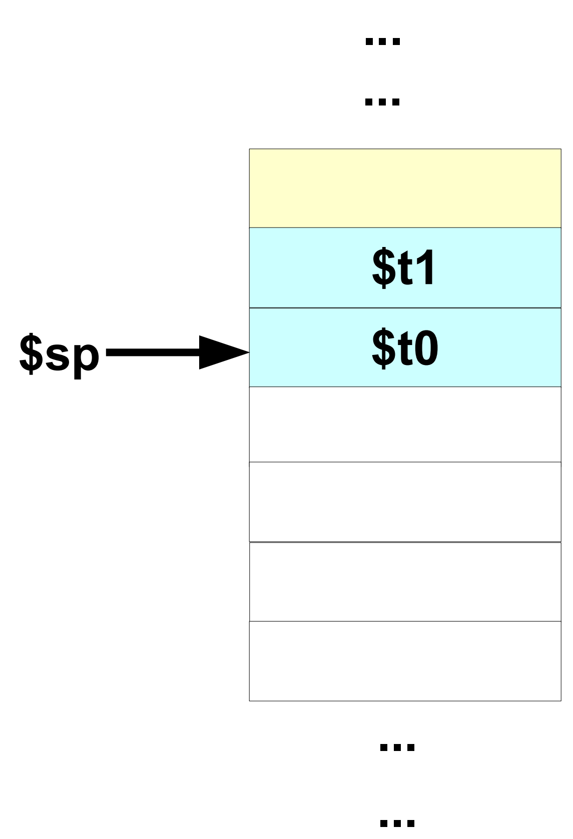


Subroutines – Parameter Passing

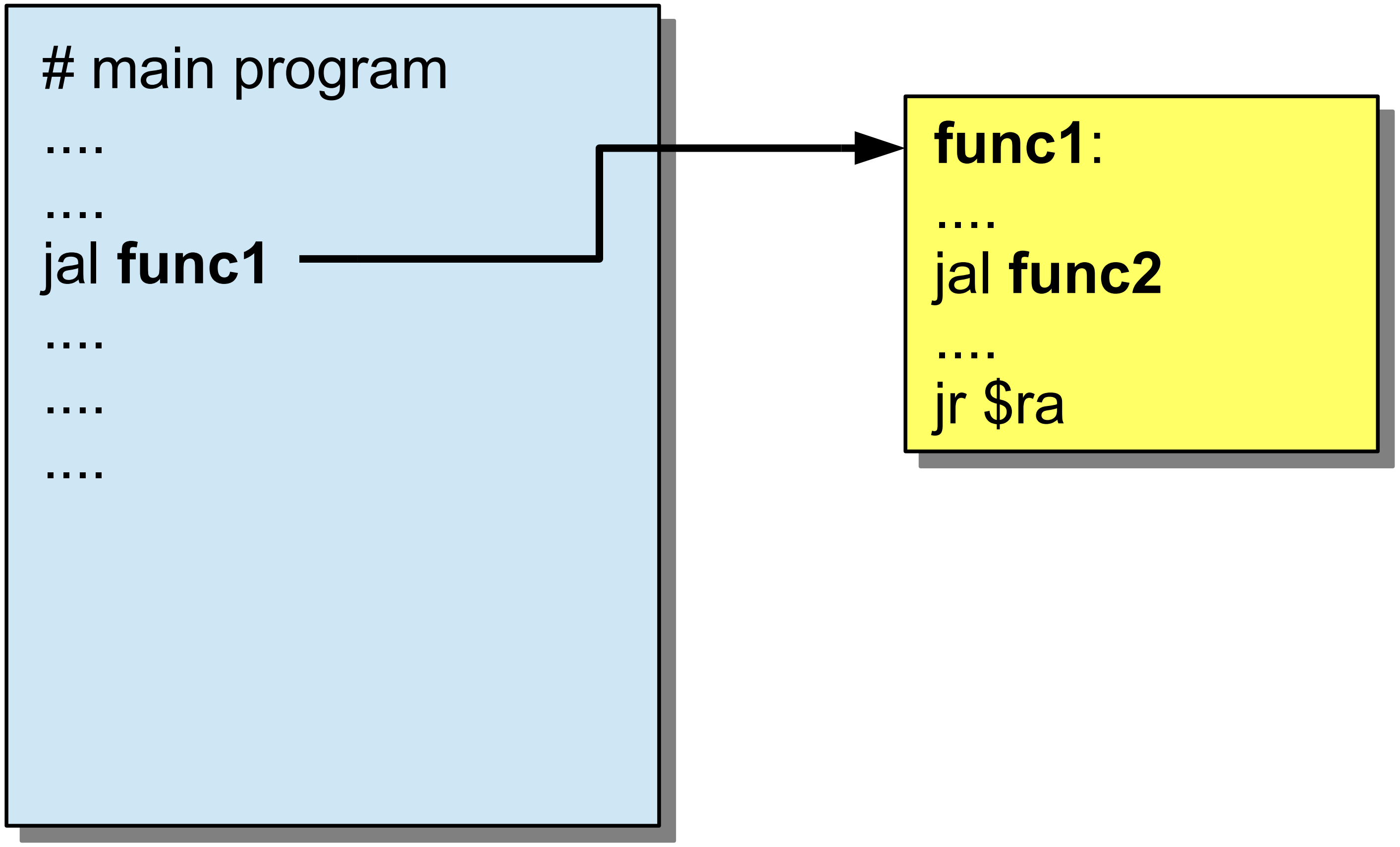
```
# main program
# 6 parameters to func1
....
....
# 4 args are in $a0 - $a3
...
# push 2 on stack
addi $sp, $sp, -8
sw $t0, 0($sp)
sw $t1, -4($sp)
jal func1
....
....
....
```

```
func1:
....
lw $t4, 0($sp)
lw $t5, -4($sp)
....
....
```

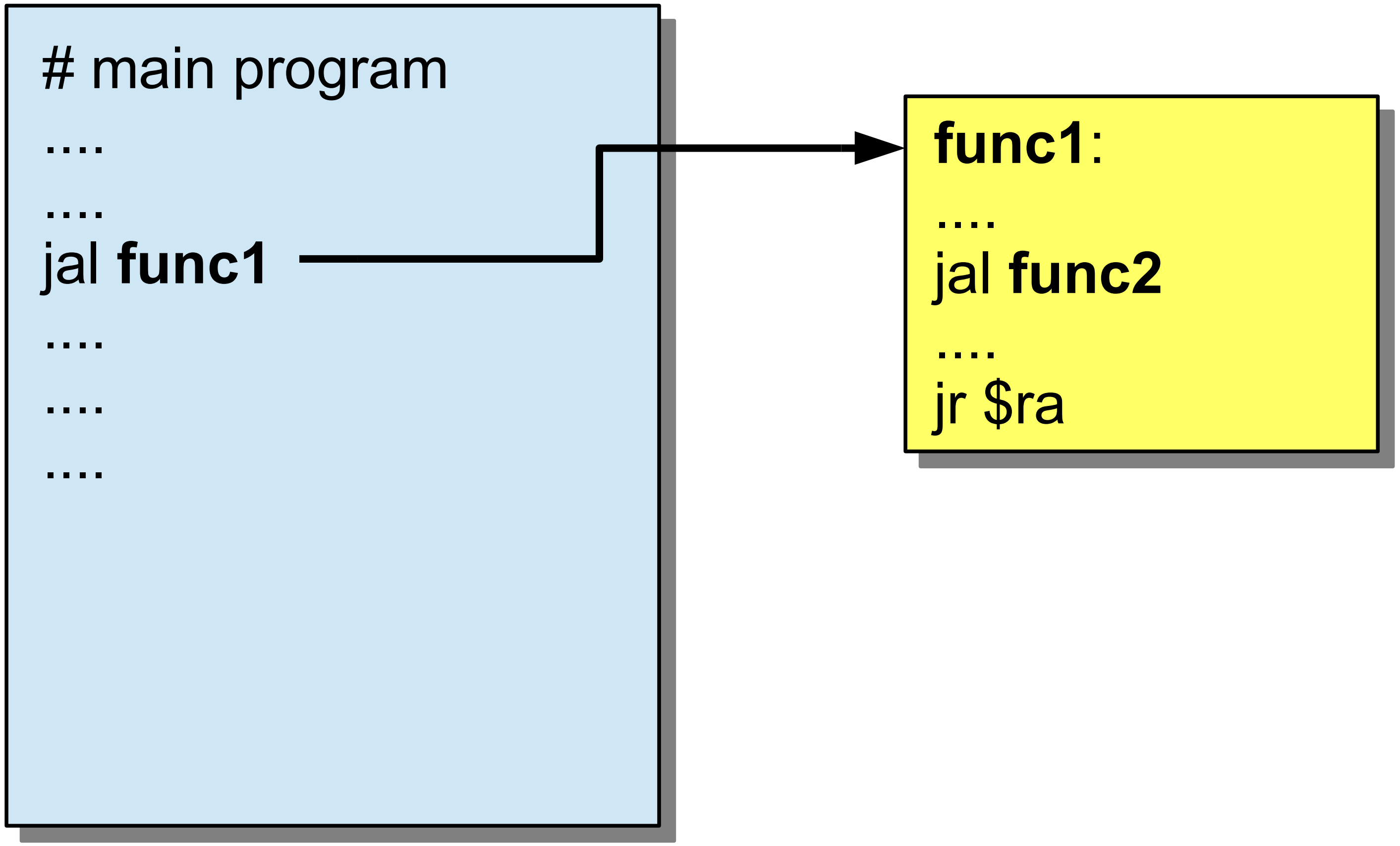
Stack after parameters are pushed



Nested Subroutines



Nested Subroutines

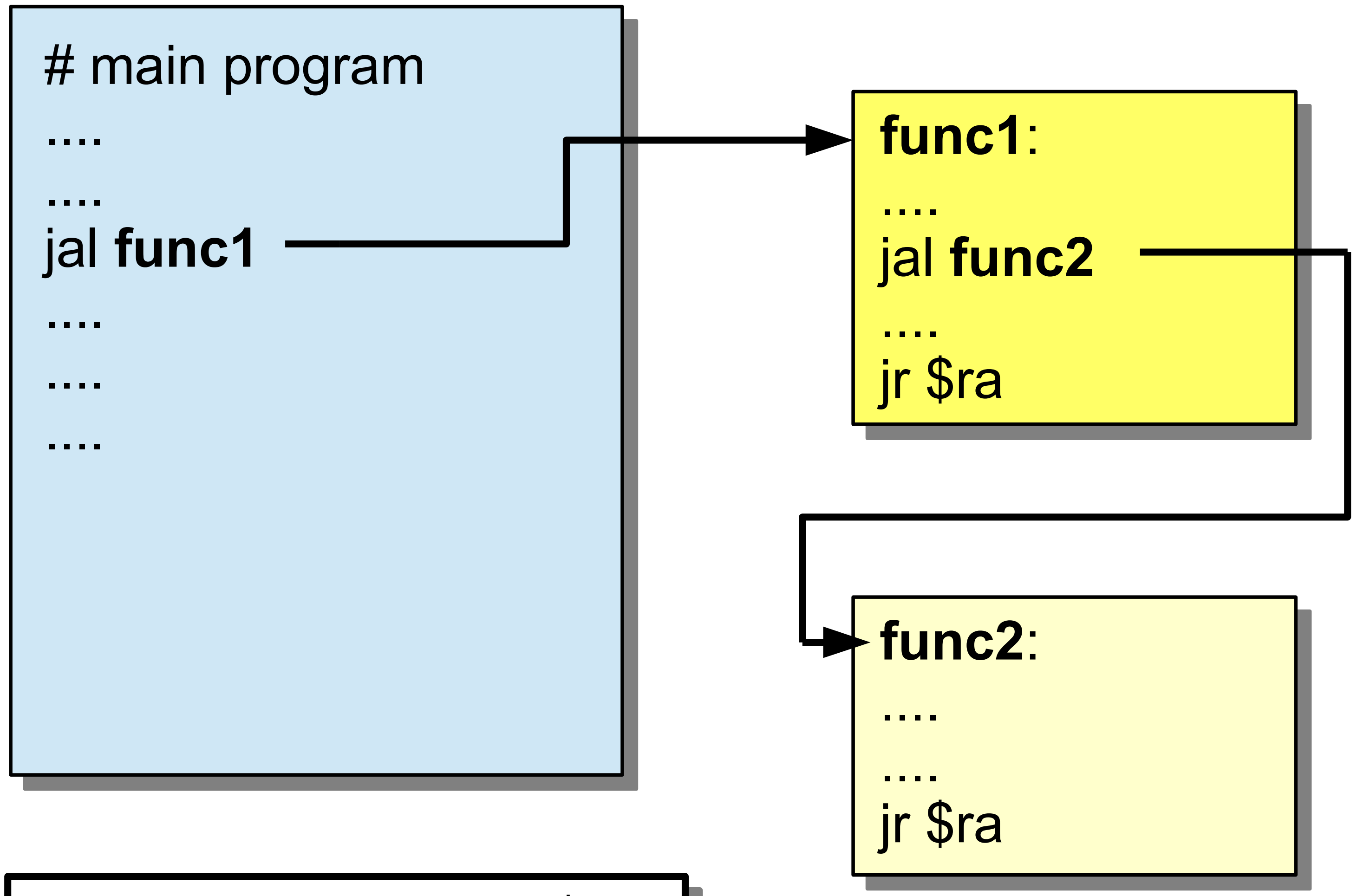


```
# main program
....
....
jal func1
....
....
....
```

```
func1:
....
jal func2
....
jr $ra
```

Stores return address in $ra

Nested Subroutines



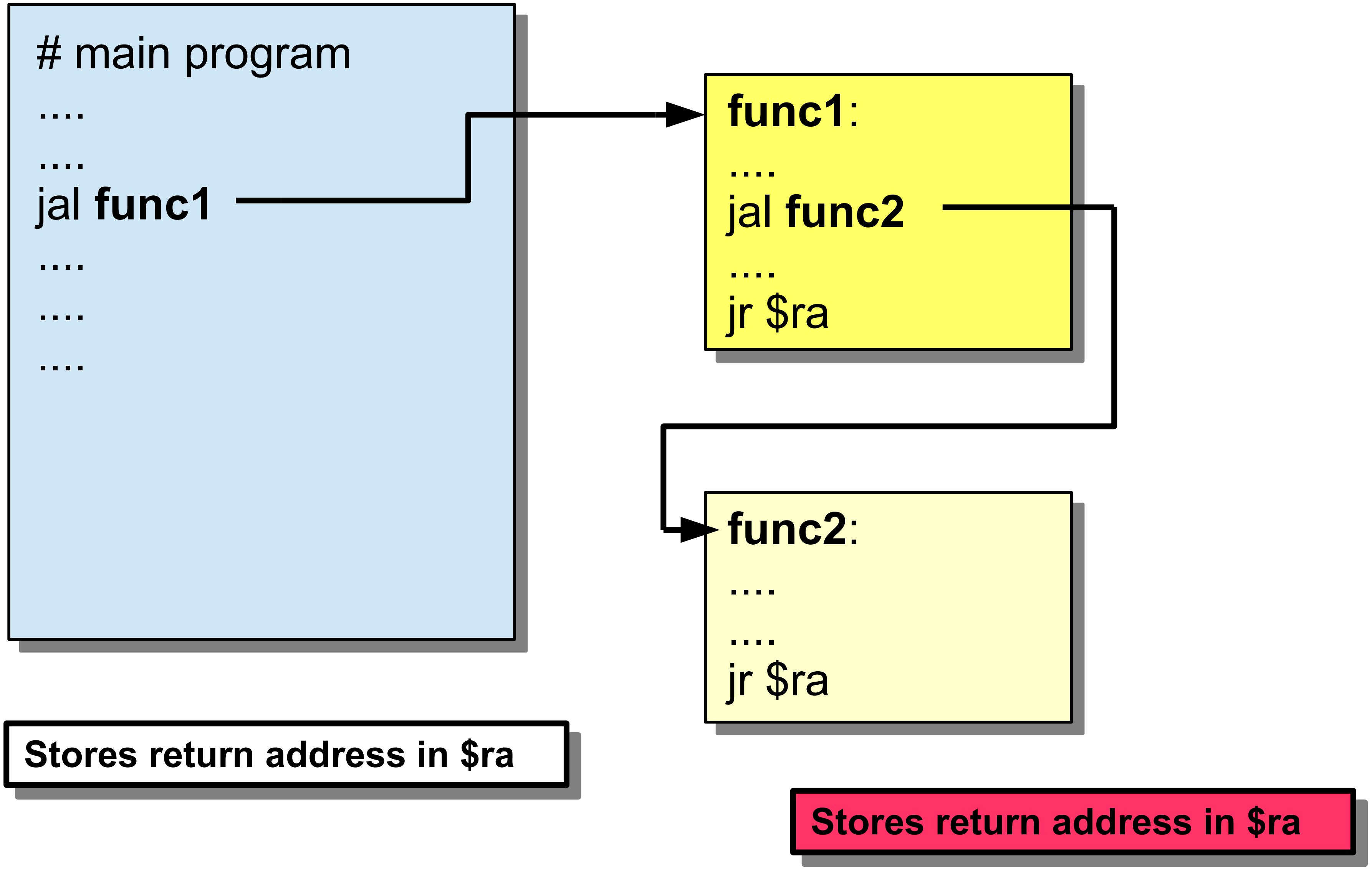
```
# main program
....
....
jal func1
....
....
....
```

```
func1:
....
jal func2
....
jr $ra
```

```
func2:
....
....
jr $ra
```

Stores return address in $ra

Nested Subroutines



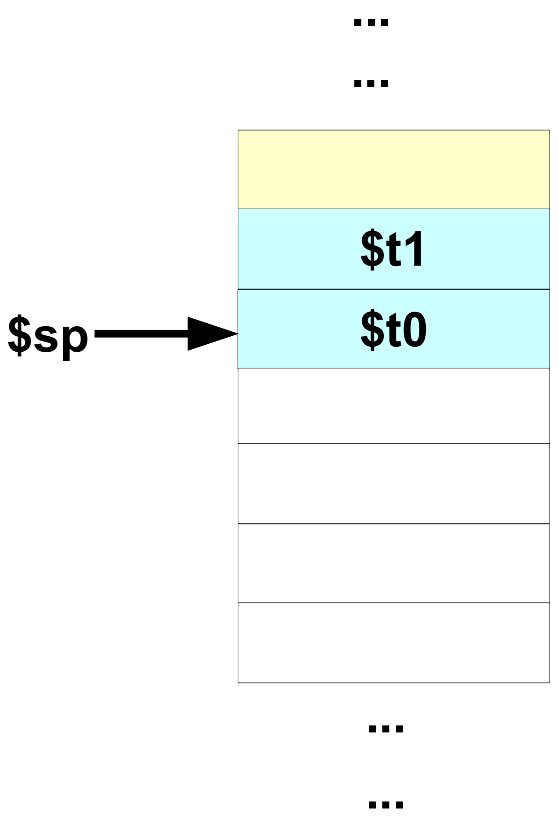
Nested Subroutines

- func1 overwrites return address in $ra (R31)
- Store the current return address in the program stack

Nested Subroutines

```
func1:
....
....
....
```

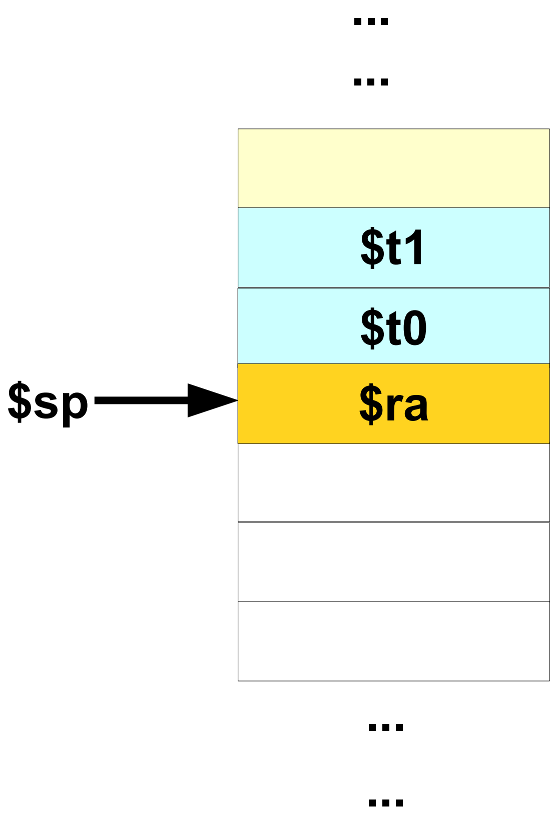
Stack before func1 is called



Nested Subroutines

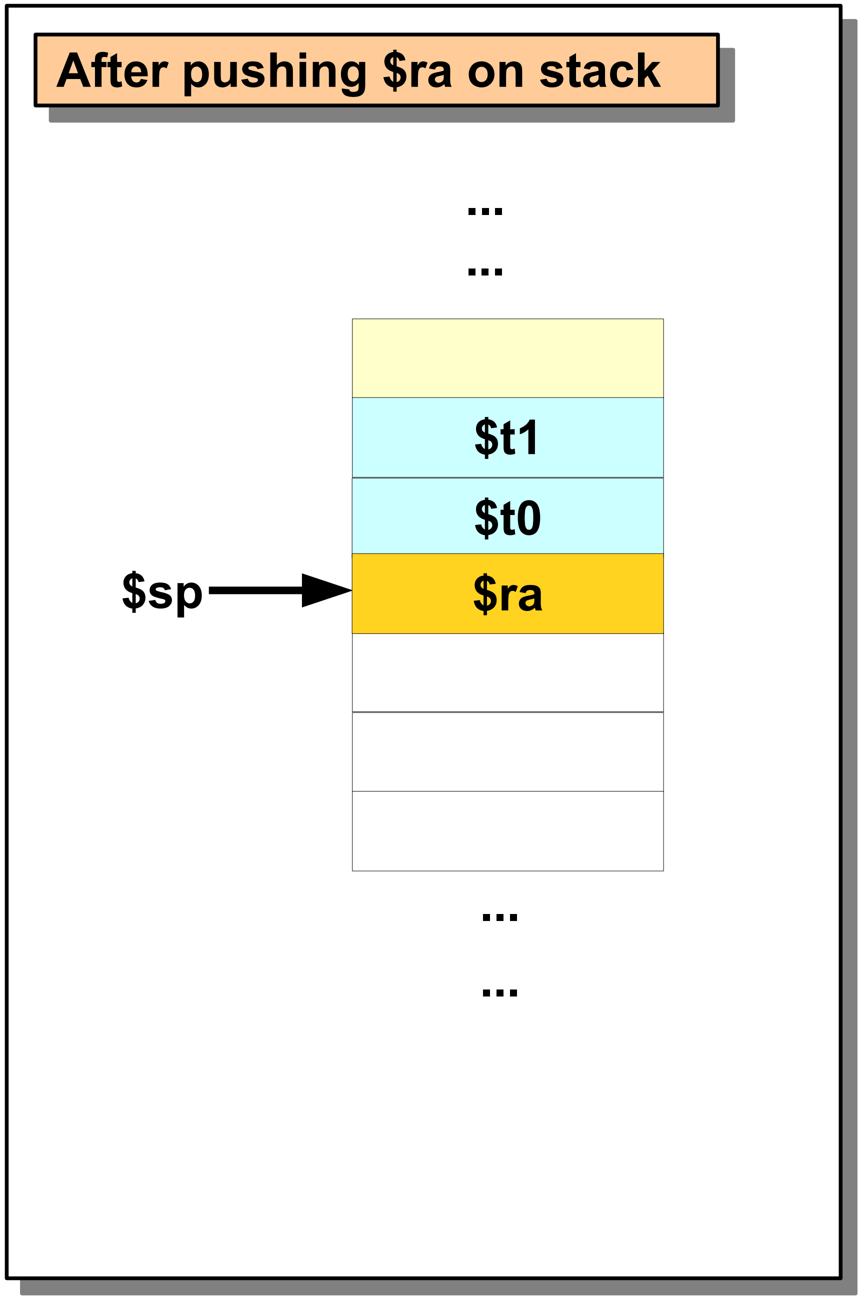
```
func1:
....
....
....
```

After pushing $ra on stack



Nested Subroutines

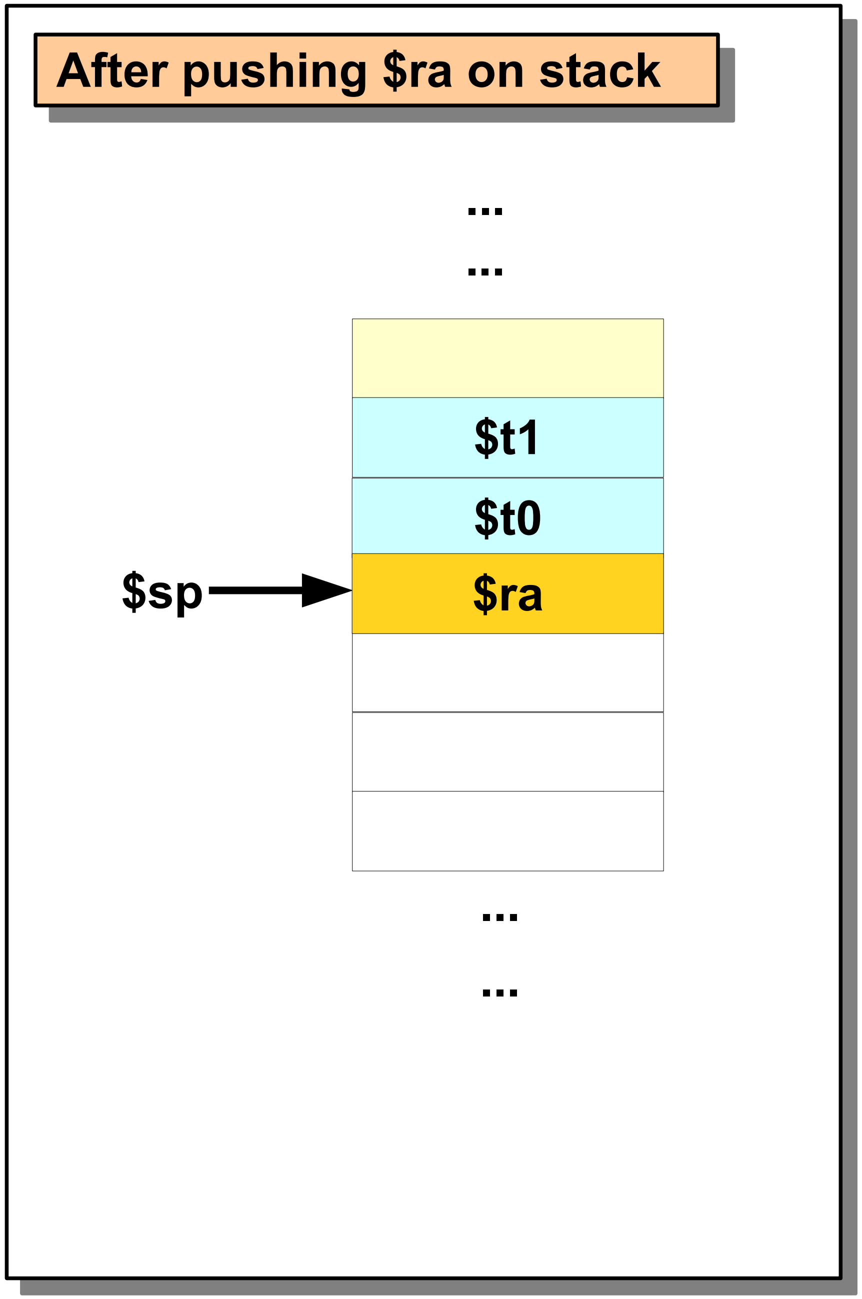
```
func1:
addi $sp, $sp, -4
sw $ra, 0($sp)
....
....
....
```



Nested Subroutines

```
func1:
addi $sp, $sp, -4
sw $ra, 0($sp)
....
....
....
```

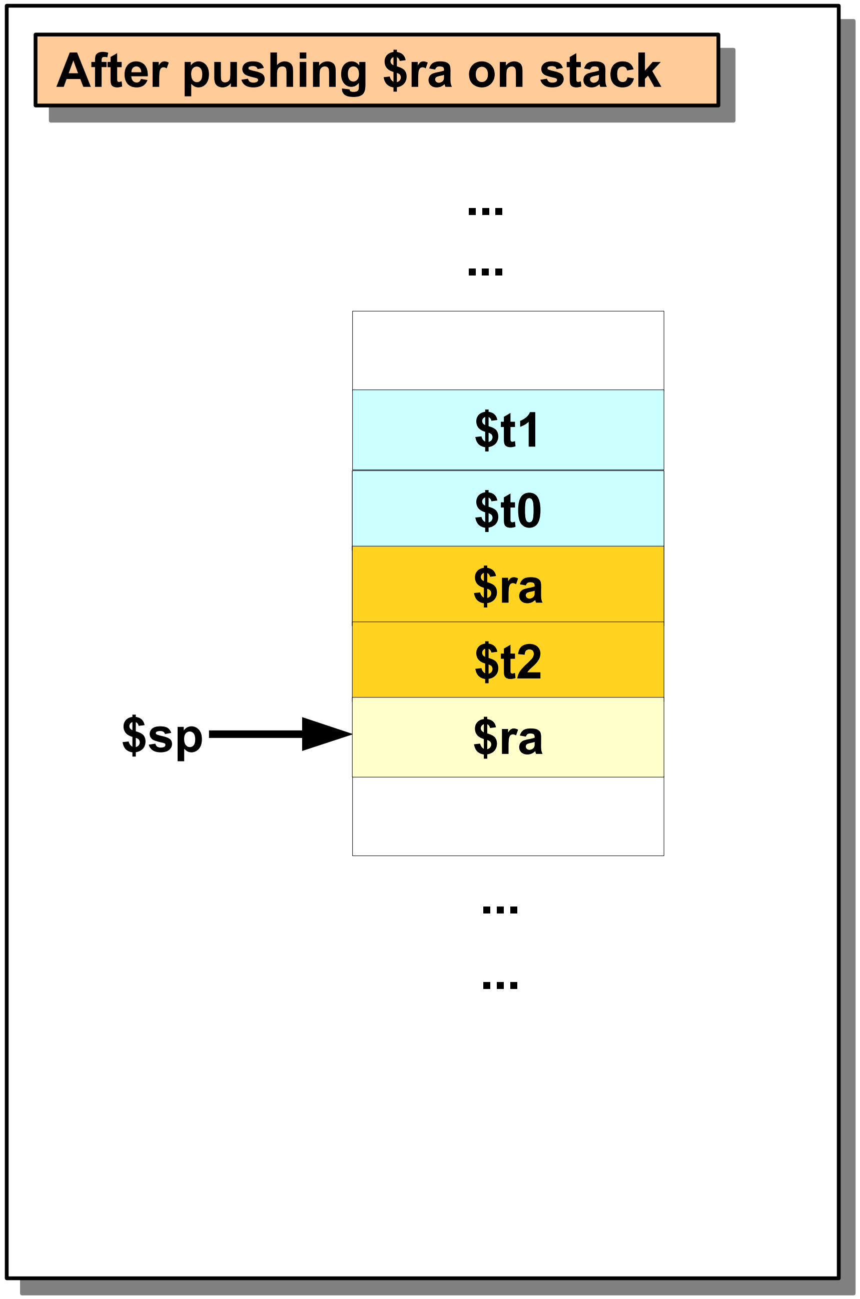
What does the stack look like after func1 passes contents of register $t2 as a parameter to func2 and calls func2? Show the code changes in func1 and func2.



Nested Subroutines

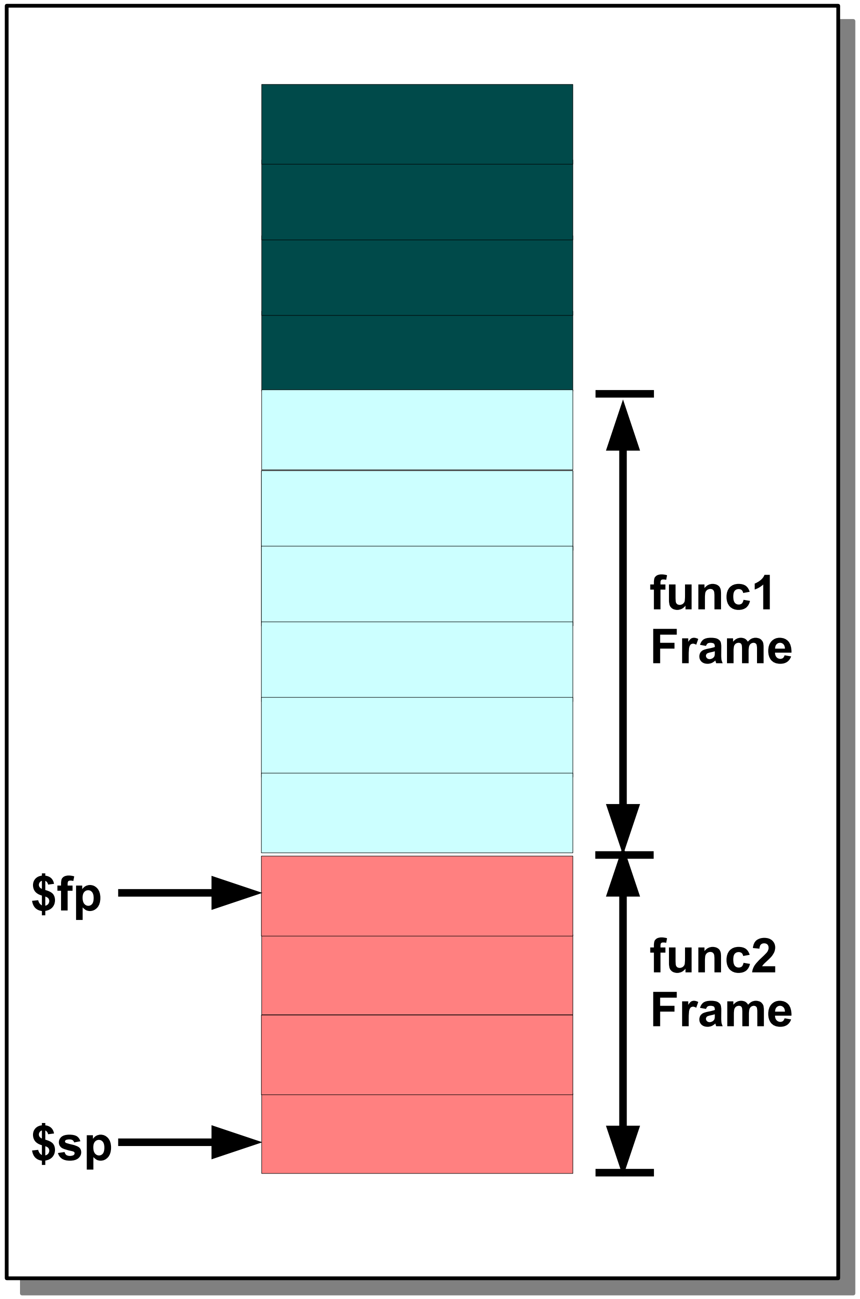
```
func1:
addi $sp, $sp, -4
sw $ra, 0($sp)
....
addi $sp, $sp, -4
sw $t2, 0($sp)
jal func2
....
....
```

```
func2:
addi $sp, $sp, -4
sw $ra, 0($sp)
....
```

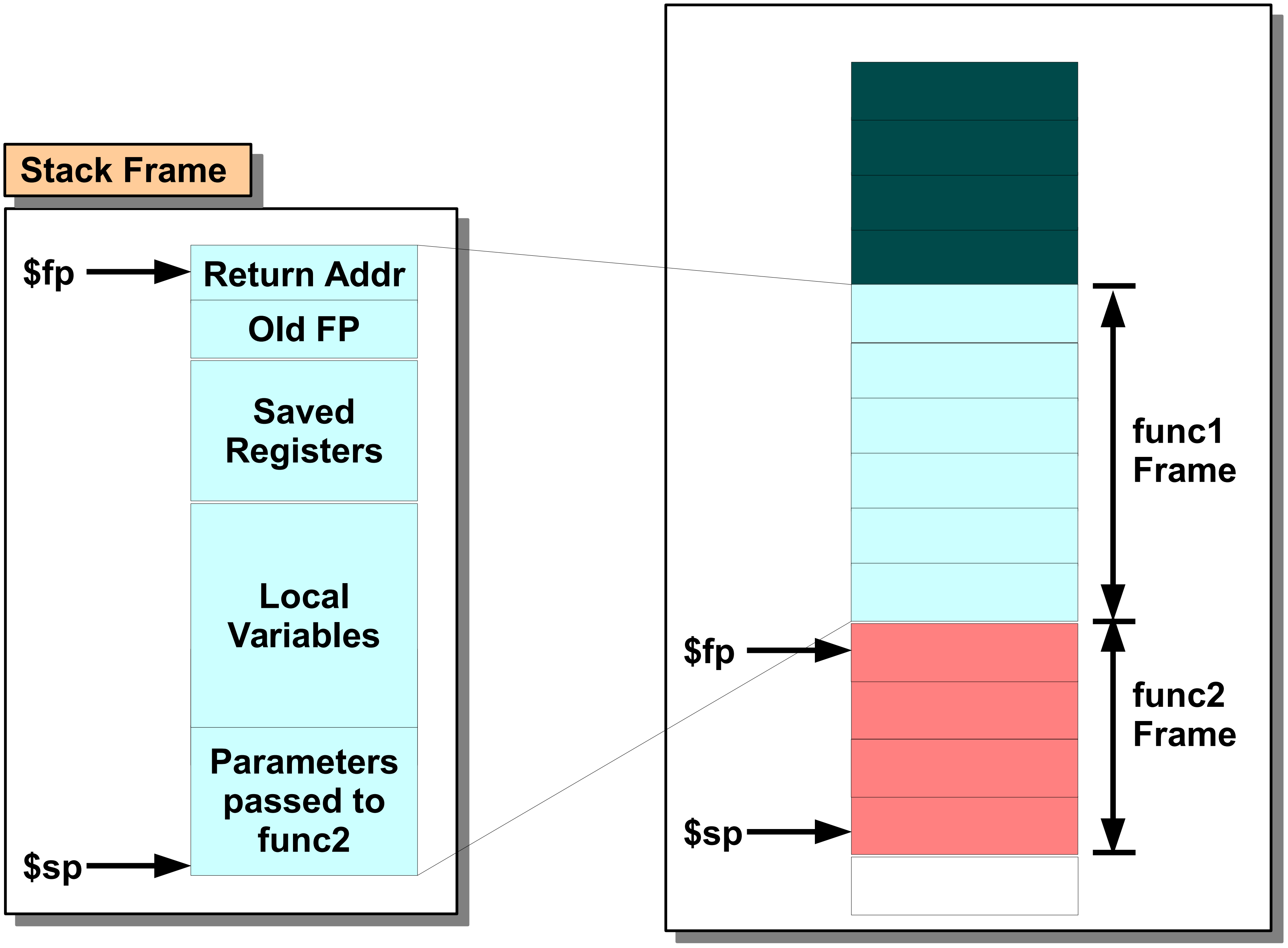


Stack Frame

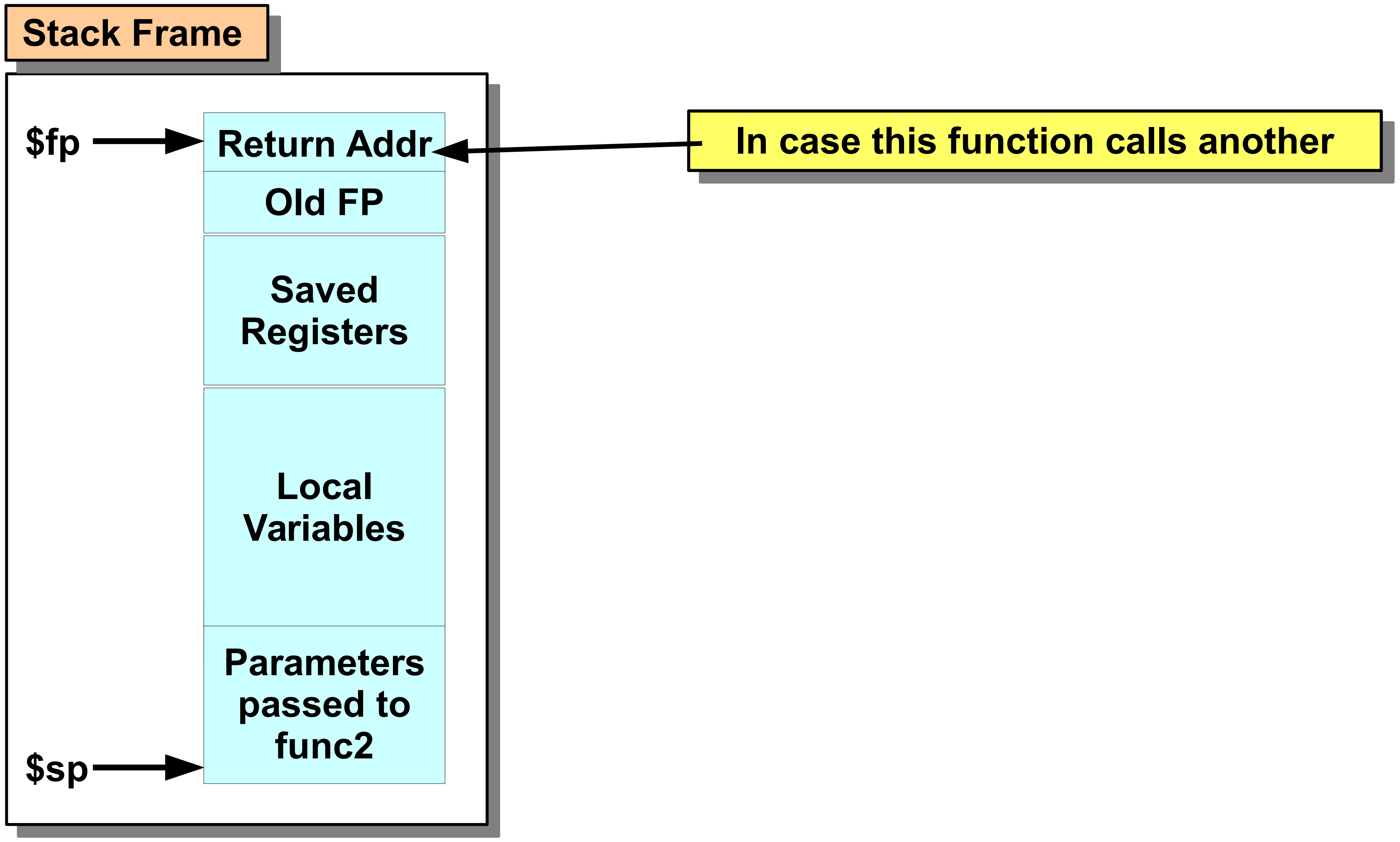
- Stack Frame: Private space for a subroutine allocated on entry and deallocated on exit
- Identified by a Frame Pointer ($fp (R30))



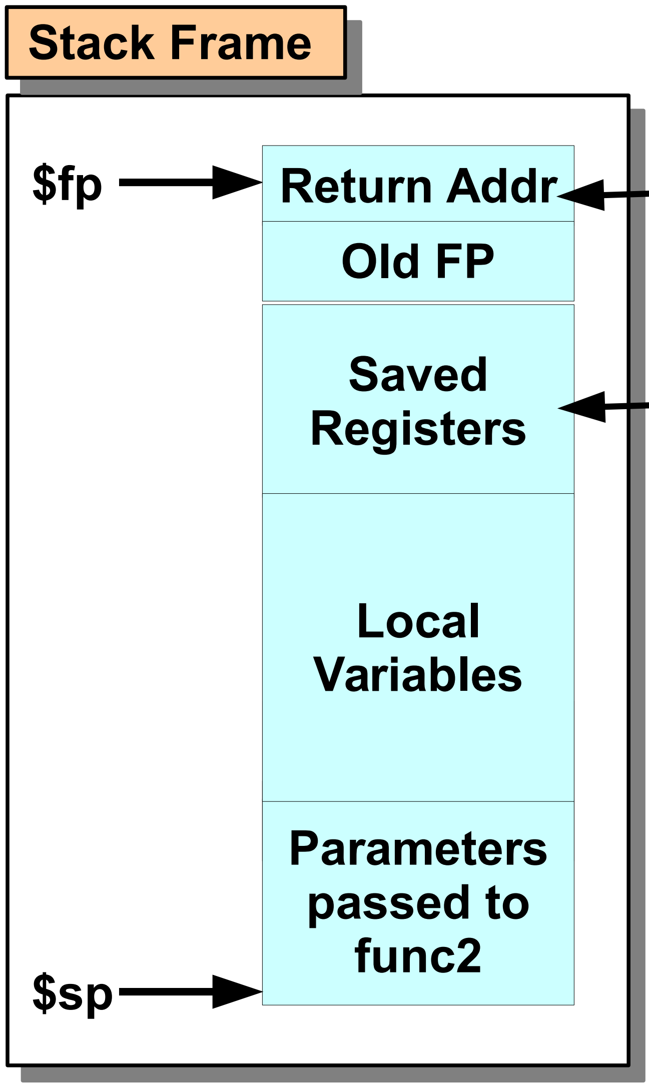
Stack Frame


Stack Frame
$fp
Return Addr
Old FP
Saved Registers
Local Variables
Parameters passed to func2
$sp
func1 Frame
$fp
func2 Frame
$sp

Stack Frame

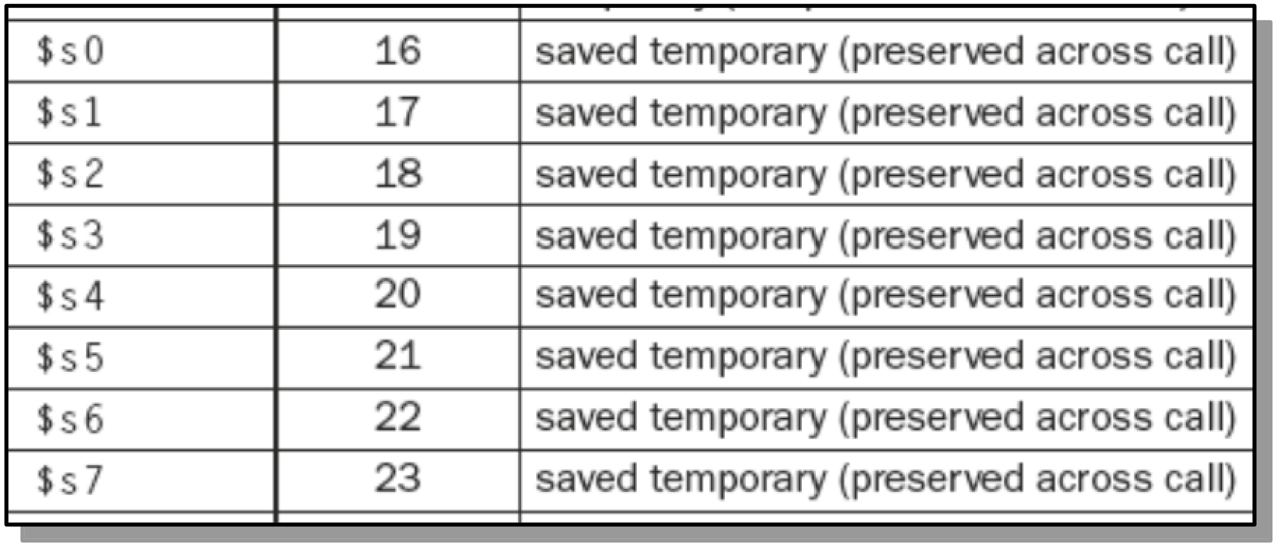

Stack Frame
$fp
Return Addr
Old FP
Saved Registers
Local Variables
Parameters passed to func2
$sp
In case this function calls another

Stack Frame



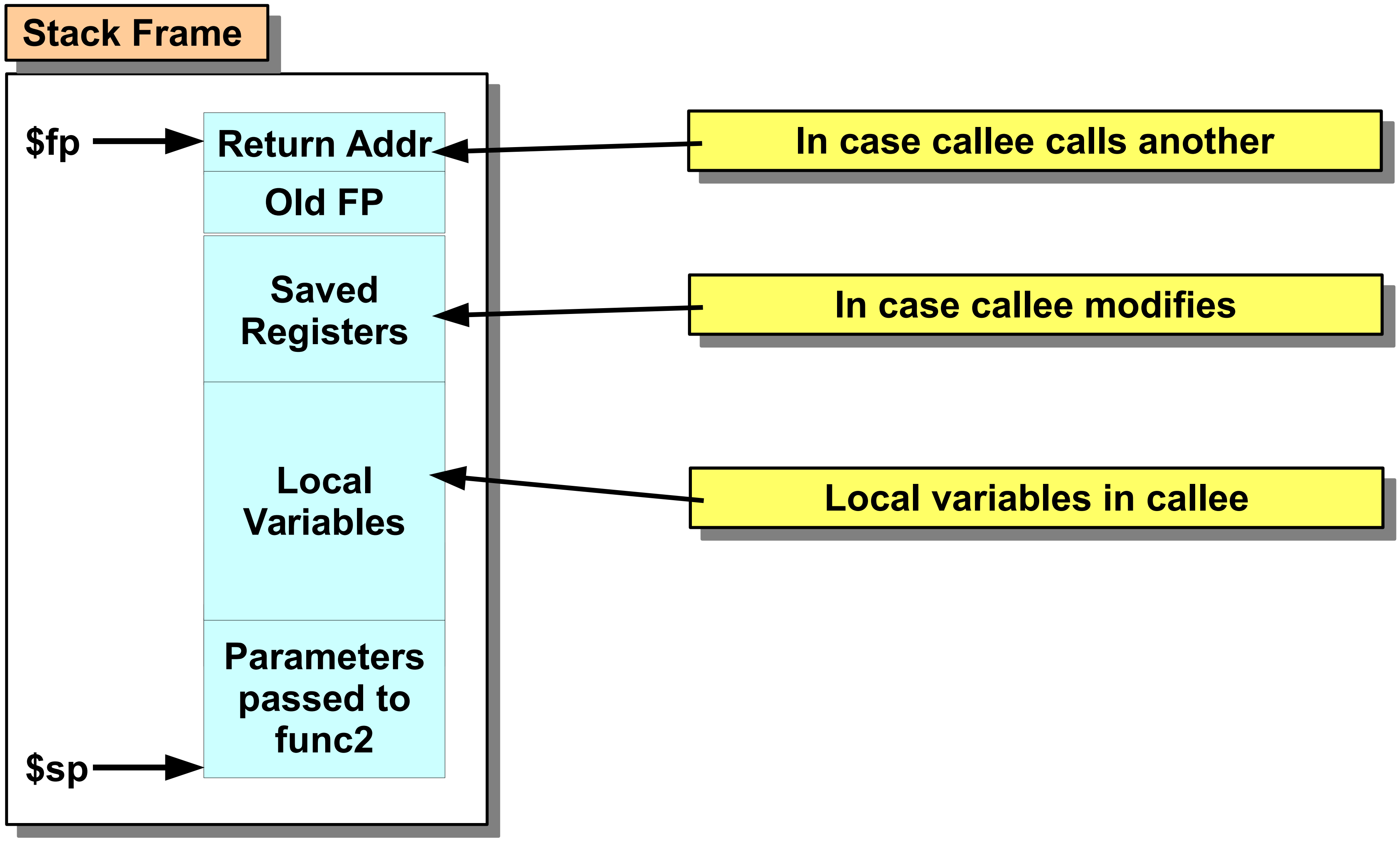
In case callee calls another

In case callee modifies

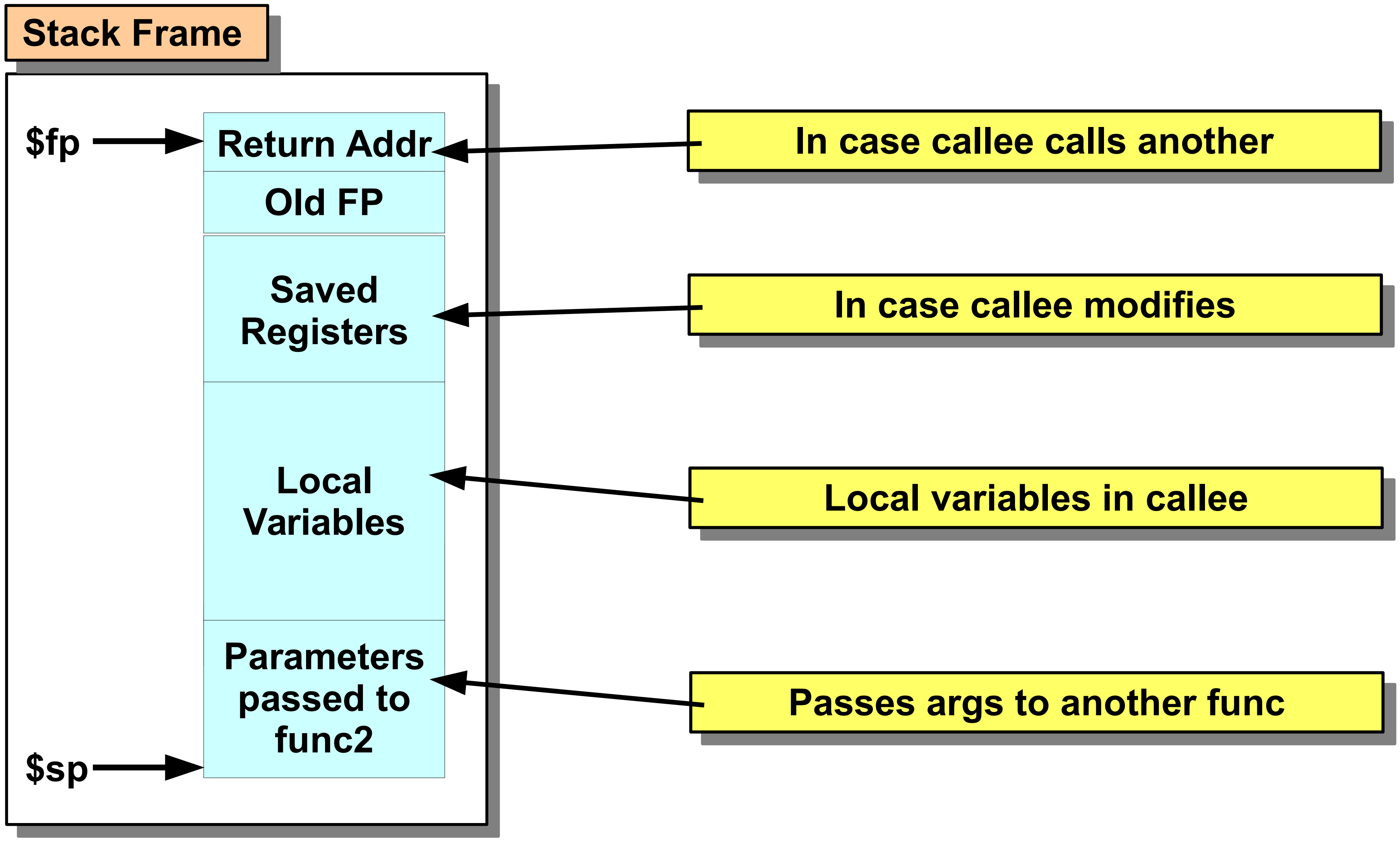


$s0	16	saved temporary (preserved across call)
$s1	17	saved temporary (preserved across call)
$s2	18	saved temporary (preserved across call)
$s3	19	saved temporary (preserved across call)
$s4	20	saved temporary (preserved across call)
$s5	21	saved temporary (preserved across call)
$s6	22	saved temporary (preserved across call)
$s7	23	saved temporary (preserved across call)

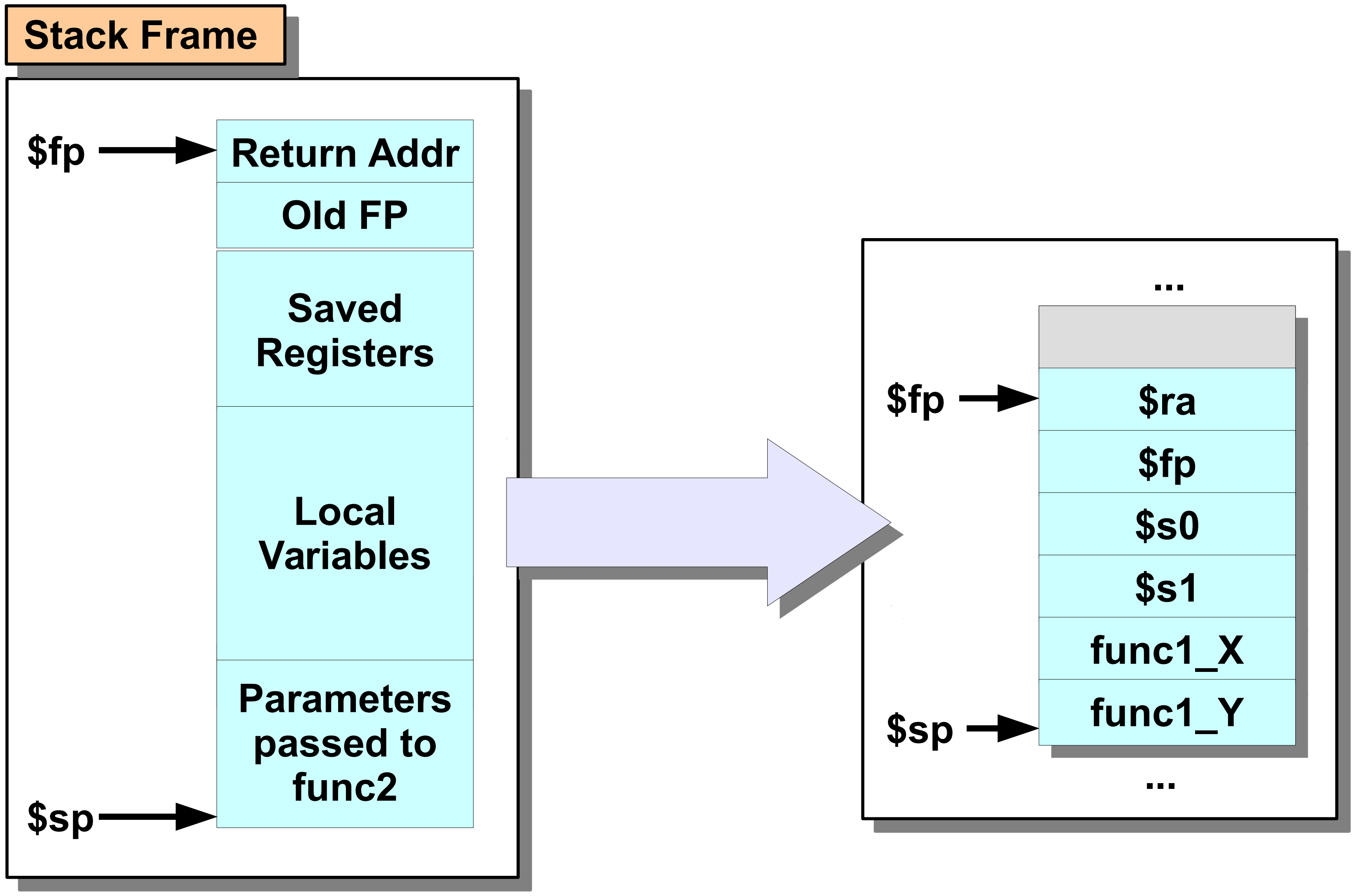
Stack Frame


Stack Frame
$fp
Return Addr
Old FP
Saved Registers
Local Variables
Parameters passed to func2
$sp
In case callee calls another
In case callee modifies
Local variables in callee

Stack Frame

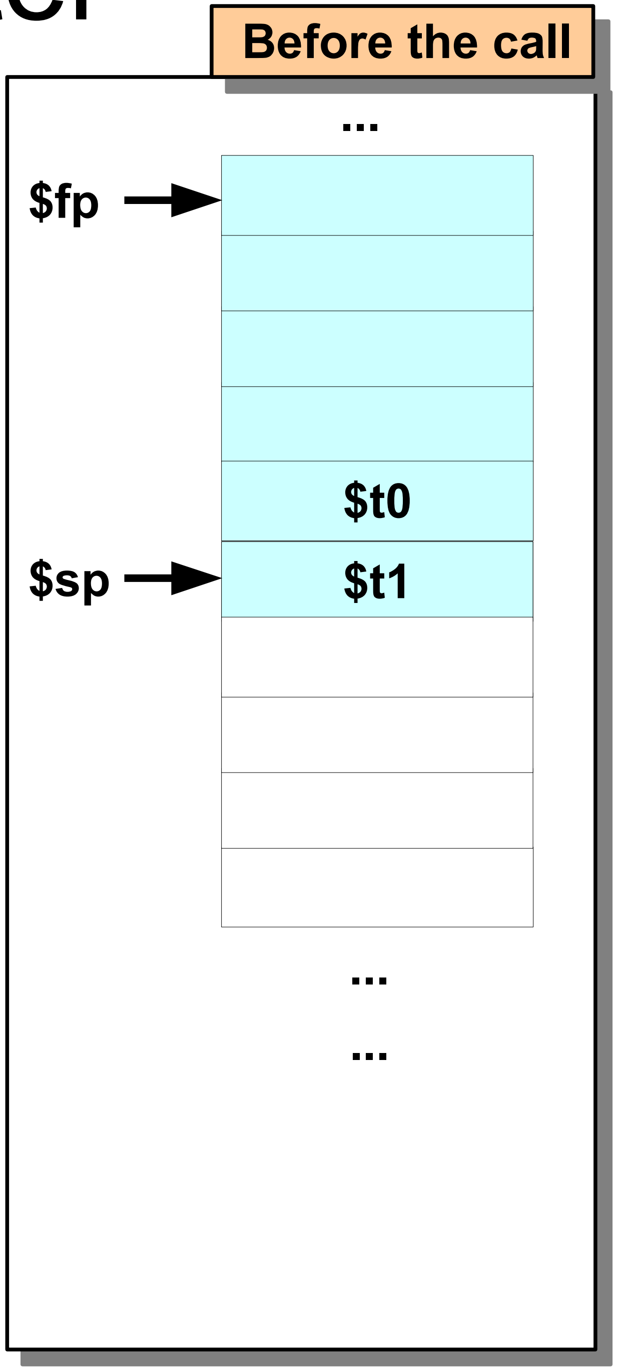

Stack Frame
$fp
Return Addr
Old FP
Saved Registers
Local Variables
Parameters passed to func2
$sp
In case callee calls another
In case callee modifies
Local variables in callee
Passes args to another func

Stack Frame


Stack Frame
$fp
Return Addr
Old FP
Saved Registers
Local Variables
Parameters passed to func2
$sp
...
$fp
$ra
$fp
$s0
$s1
func1_X
func1_Y
$sp
...

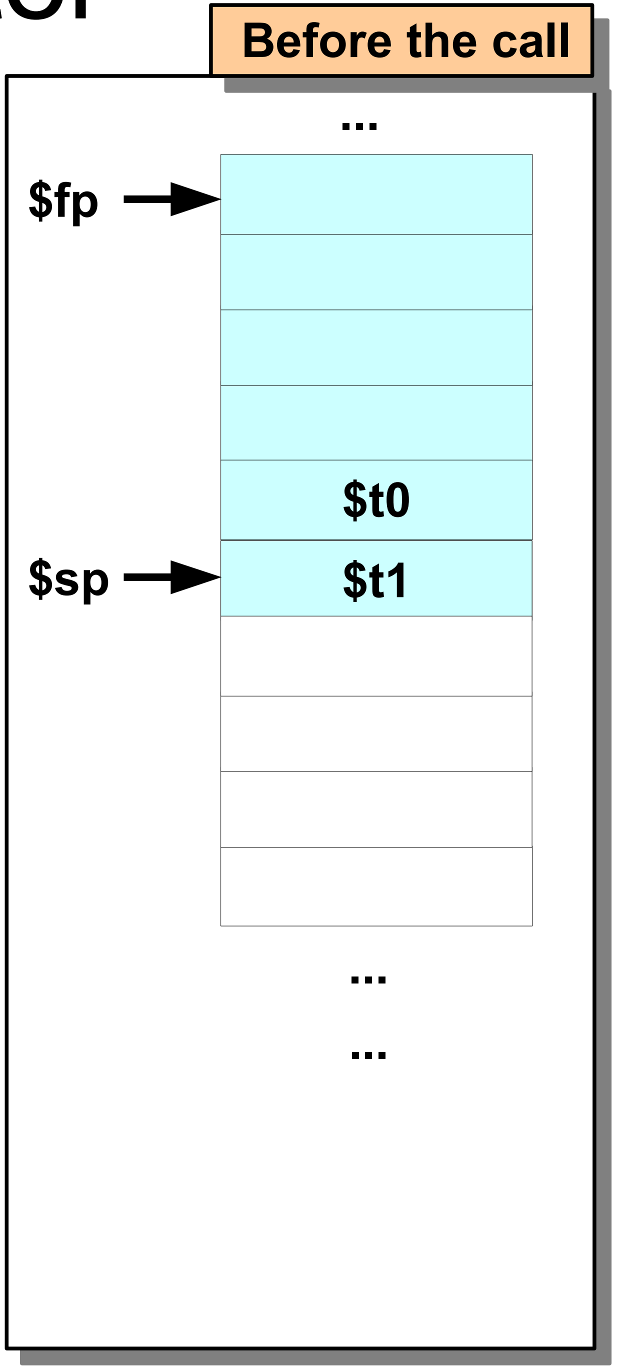
Frame Pointer

- After entry into a subroutine:



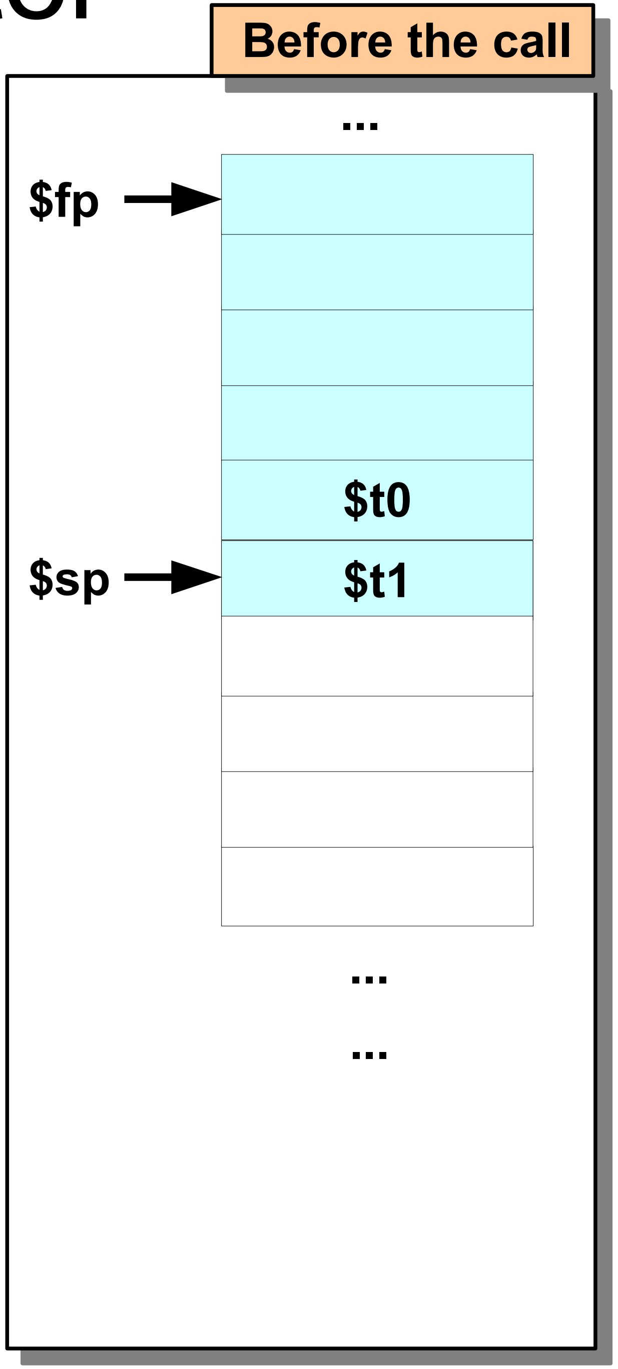
Frame Pointer

- After entry into a subroutine:
  - Save return address



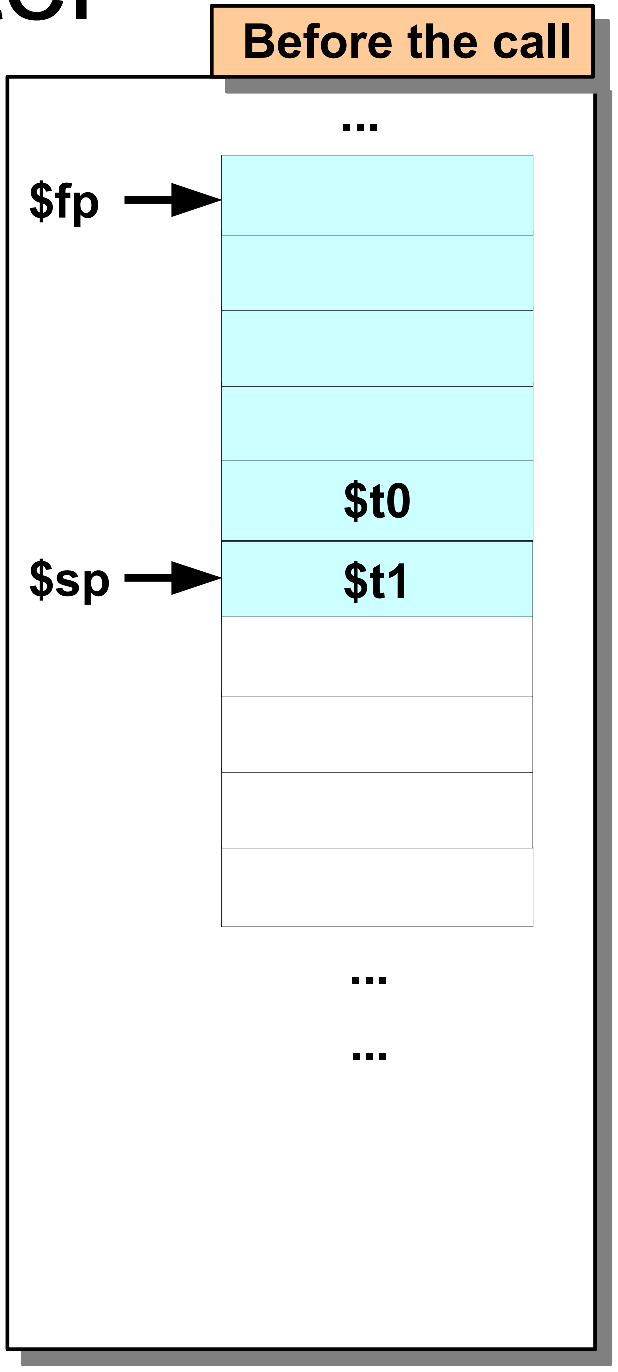
Frame Pointer

- After entry into a subroutine:
  - Save return address
  - Save frame pointer of the caller function



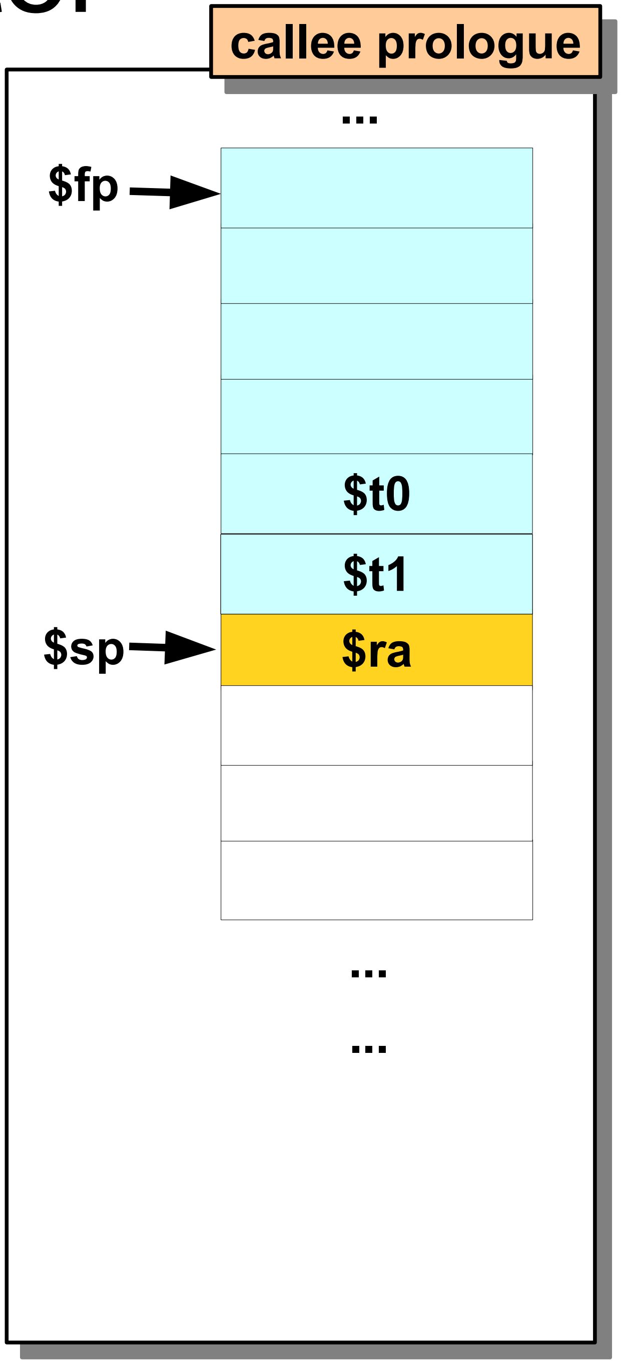
Frame Pointer

- After entry into a subroutine:
  - Save return address
  - Save frame pointer of the caller function
  - Point the frame pointer to the first location of stack frame of the current subroutine



Frame Pointer

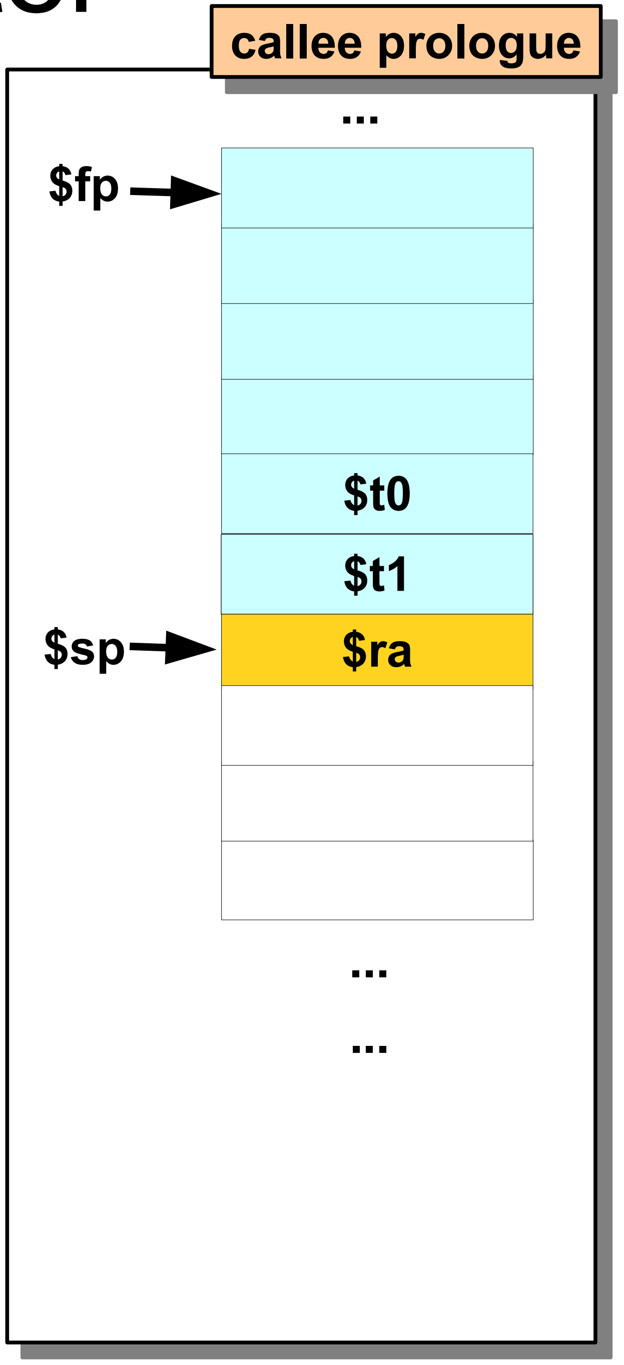
- After entry into a subroutine:
  - **Save return address**



Frame Pointer

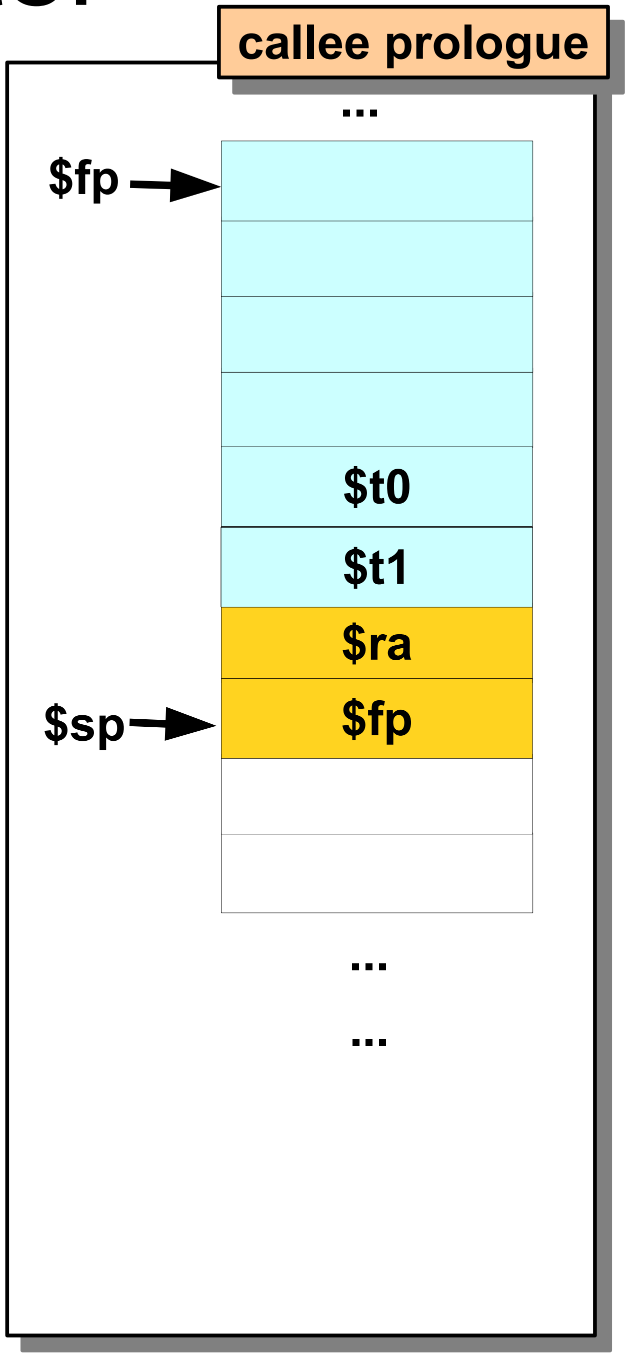
- After entry into a subroutine:
  - **Save return address**

```
addi $sp, $sp, -4
sw $ra, 0($sp)
```



Frame Pointer

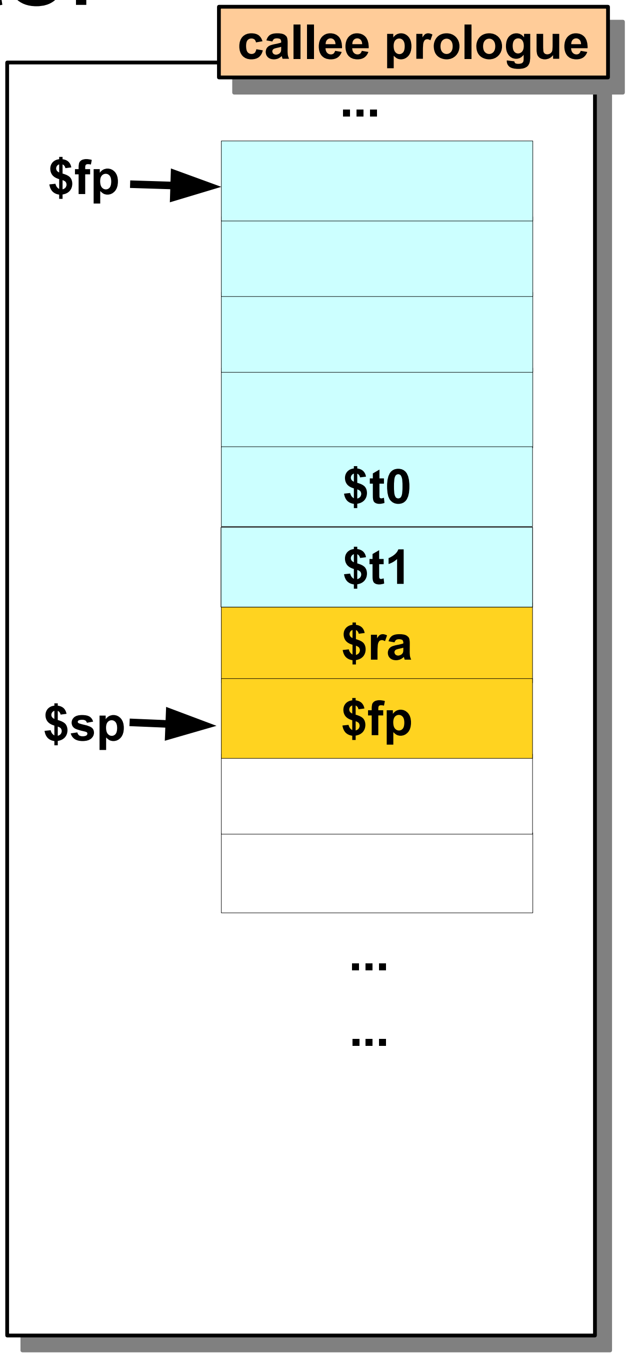
- After entry into a subroutine:
  - Save return address
  - **Save frame pointer of the caller function**



Frame Pointer

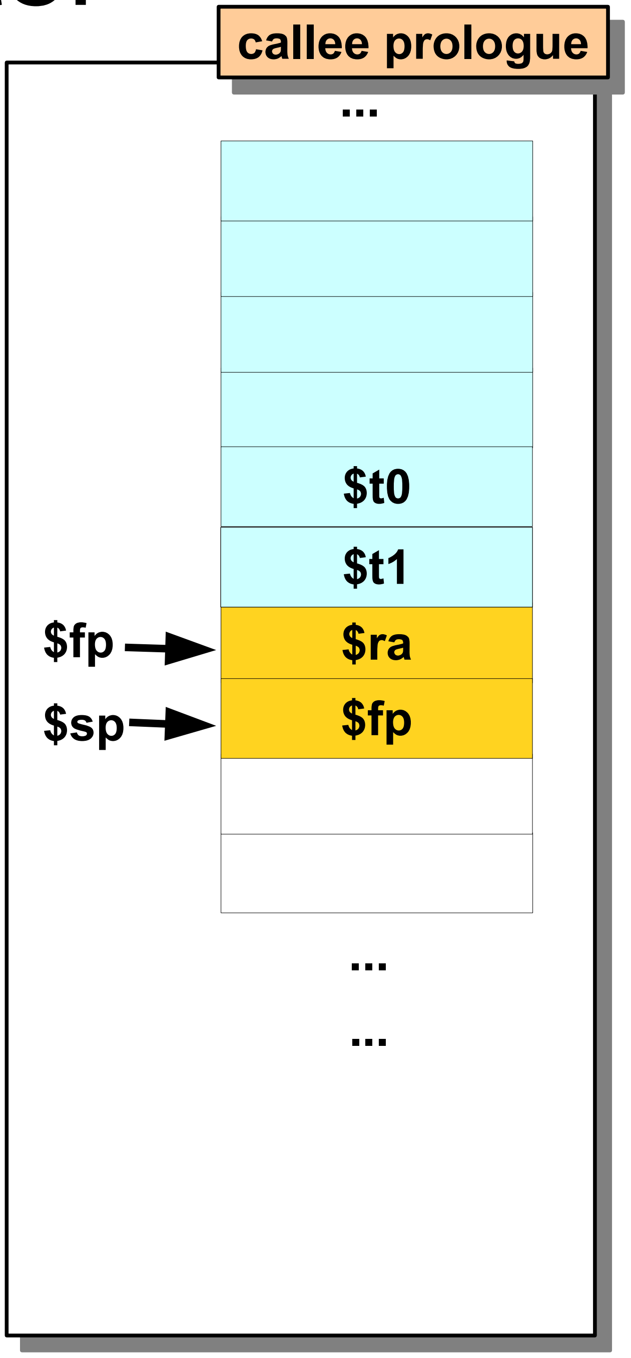
- After entry into a subroutine:
  - Save return address
  - **Save frame pointer of the caller function**

```
addi $sp, $sp, -4
sw $fp, 0($sp)
```



Frame Pointer

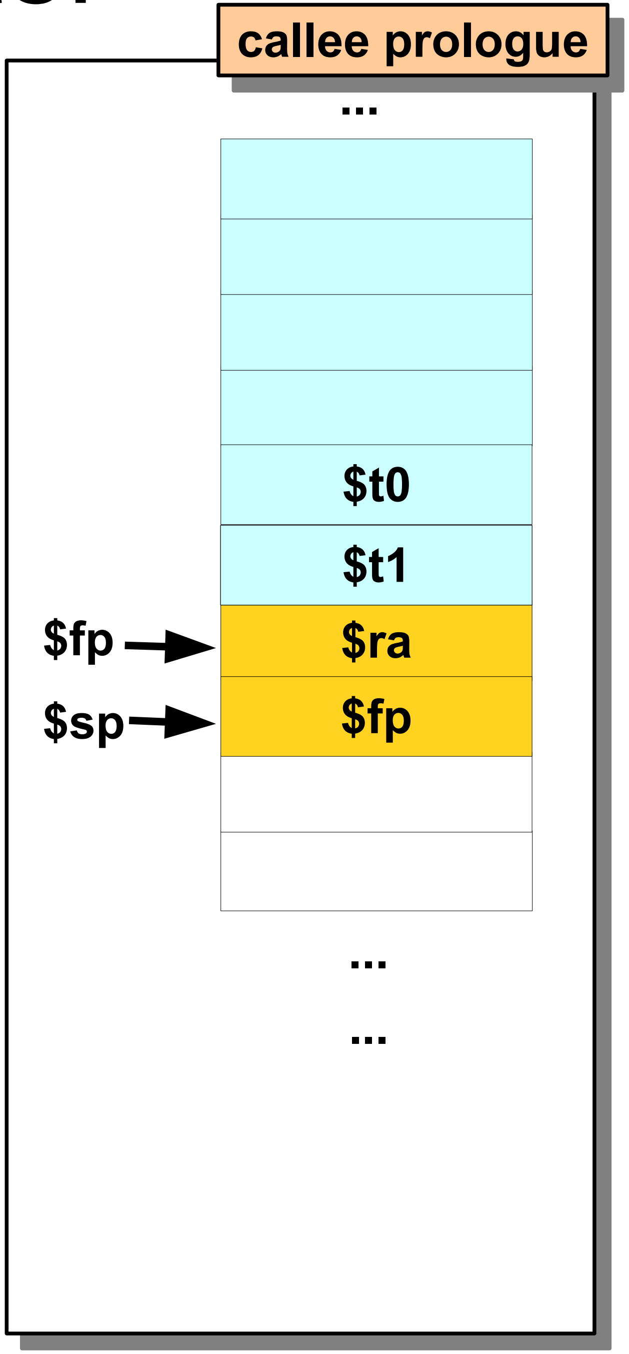
- After entry into a subroutine:
  - Save return address
  - Save frame pointer of the caller function
  - **Point the frame pointer to the first location of stack frame of the current subroutine**



Frame Pointer

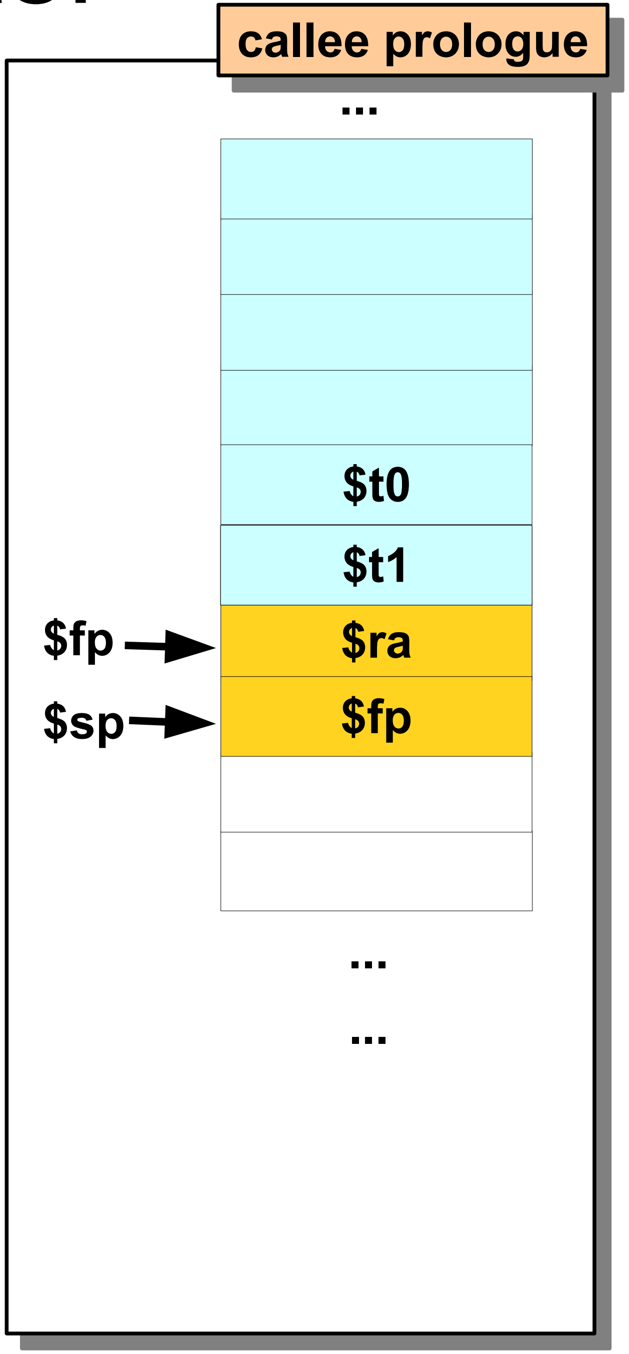
- After entry into a subroutine:
  - Save return address
  - Save frame pointer of the caller function
  - **Point the frame pointer to the first location of stack frame of the current subroutine**

```
addi $fp, $sp, 4
```



Frame Pointer

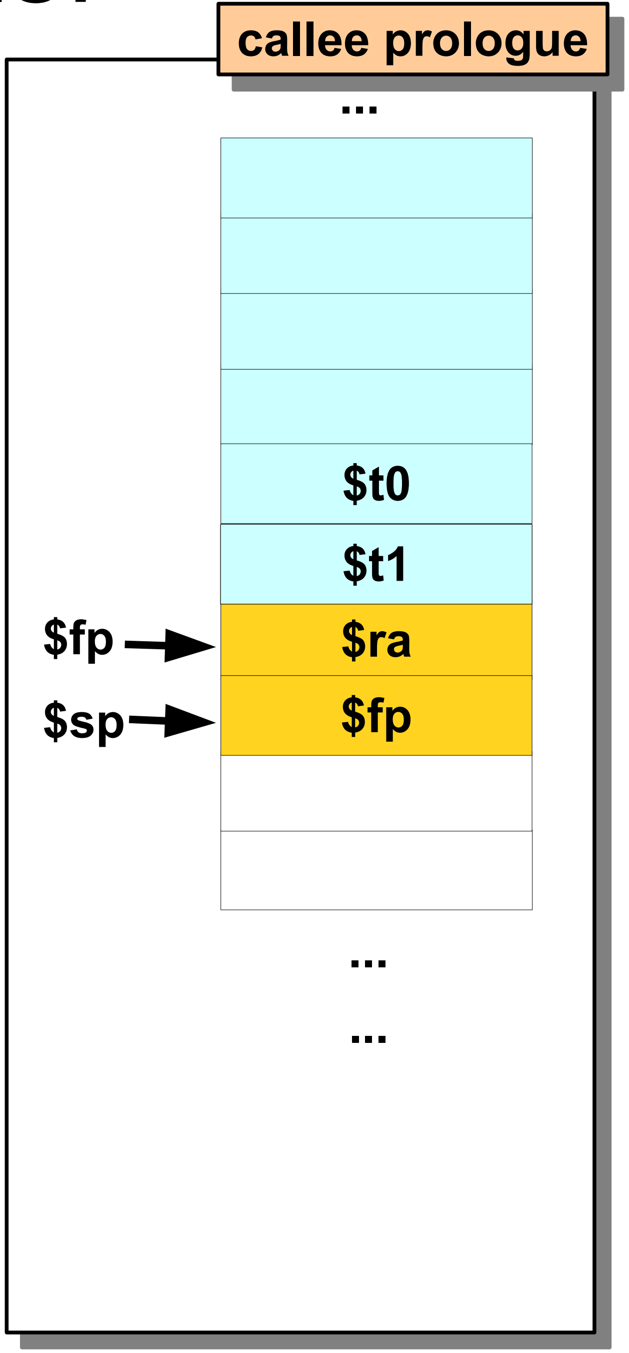
```
func1:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $fp, 0($sp)
addi $fp, $sp, 4
....
....
....
```



Frame Pointer

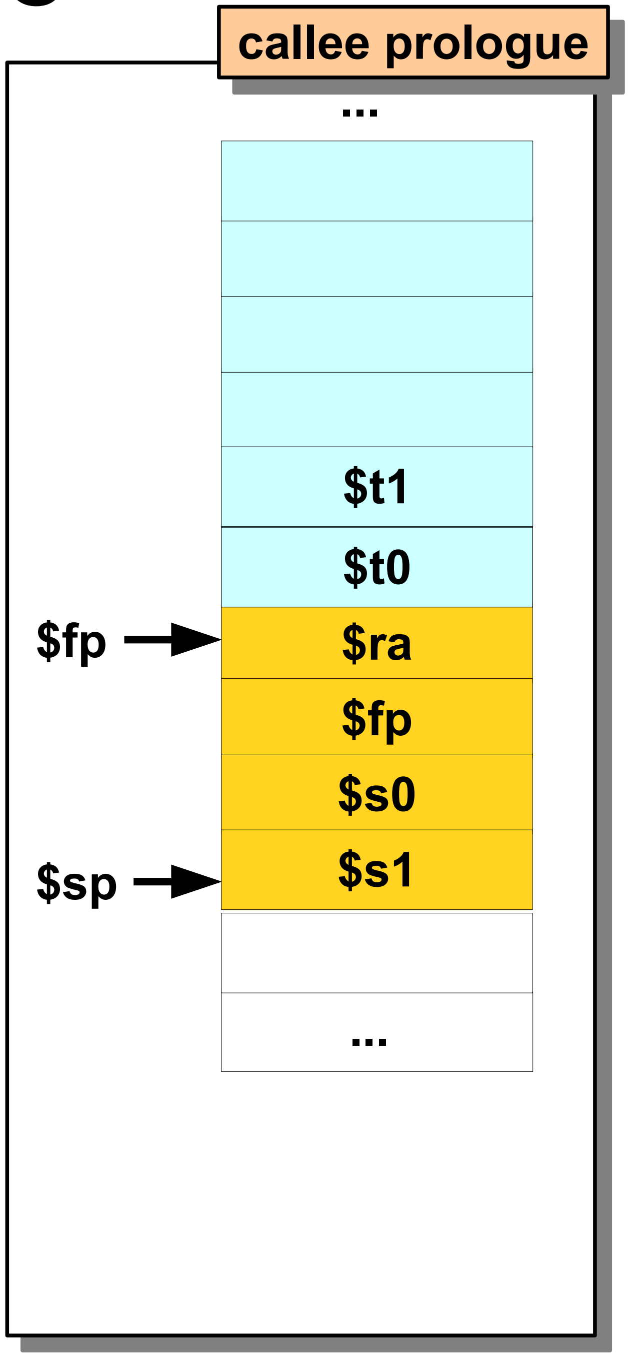
```
func1:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $fp, 0($sp)
addi $fp, $sp, 4
....
....
....
```

Parameters can be accessed in the callee function: 4($fp), 8($fp)



Stack Frame

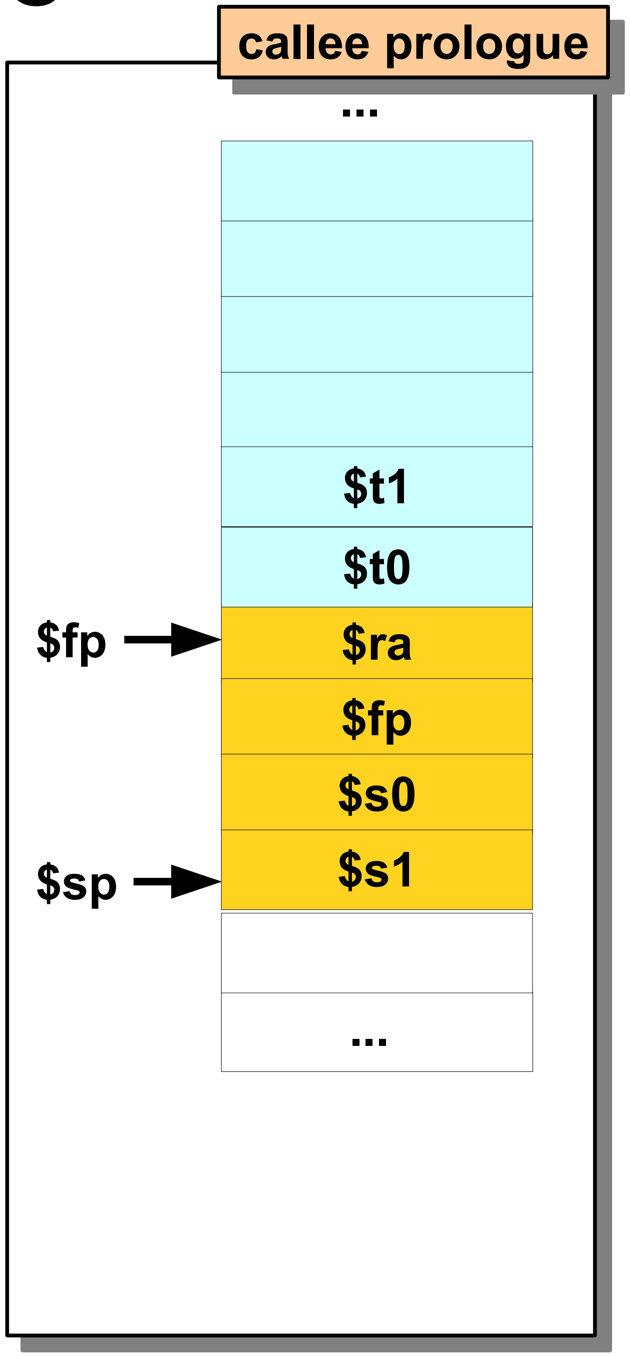
- **Save change registers**



Stack Frame

- **Save change registers**

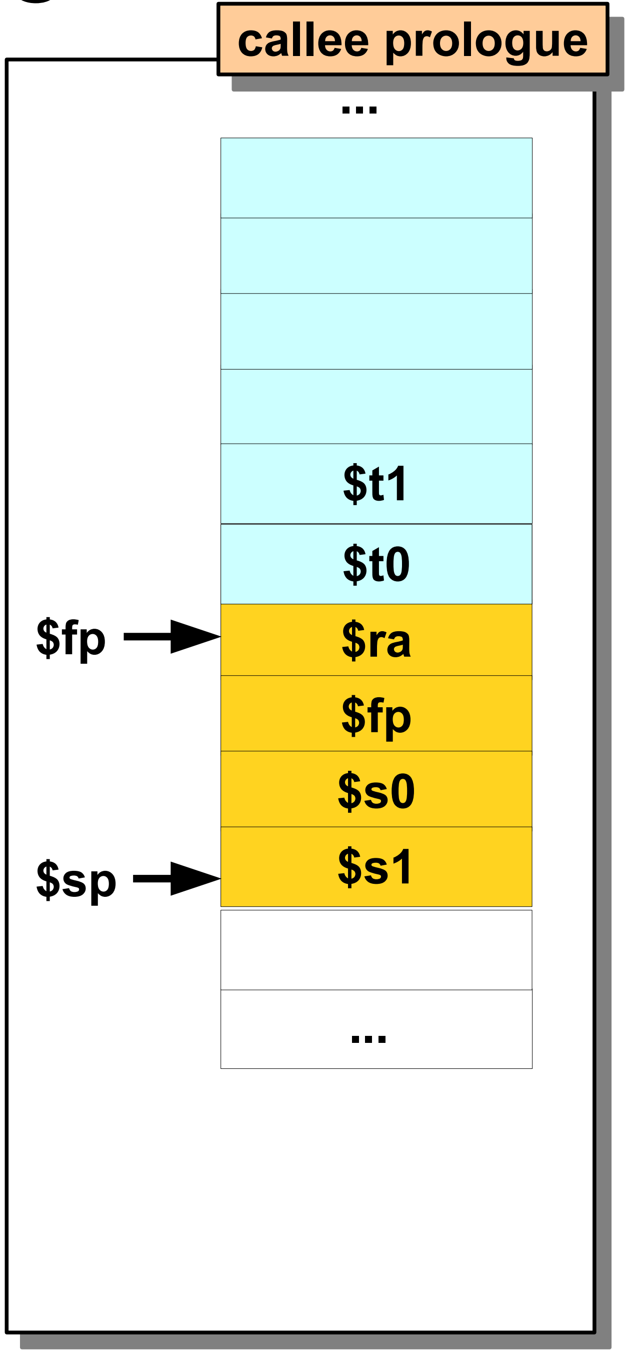
```
addi $sp, $sp, -8
sw $s0, 4($sp)
sw $s1, 0($sp)
```



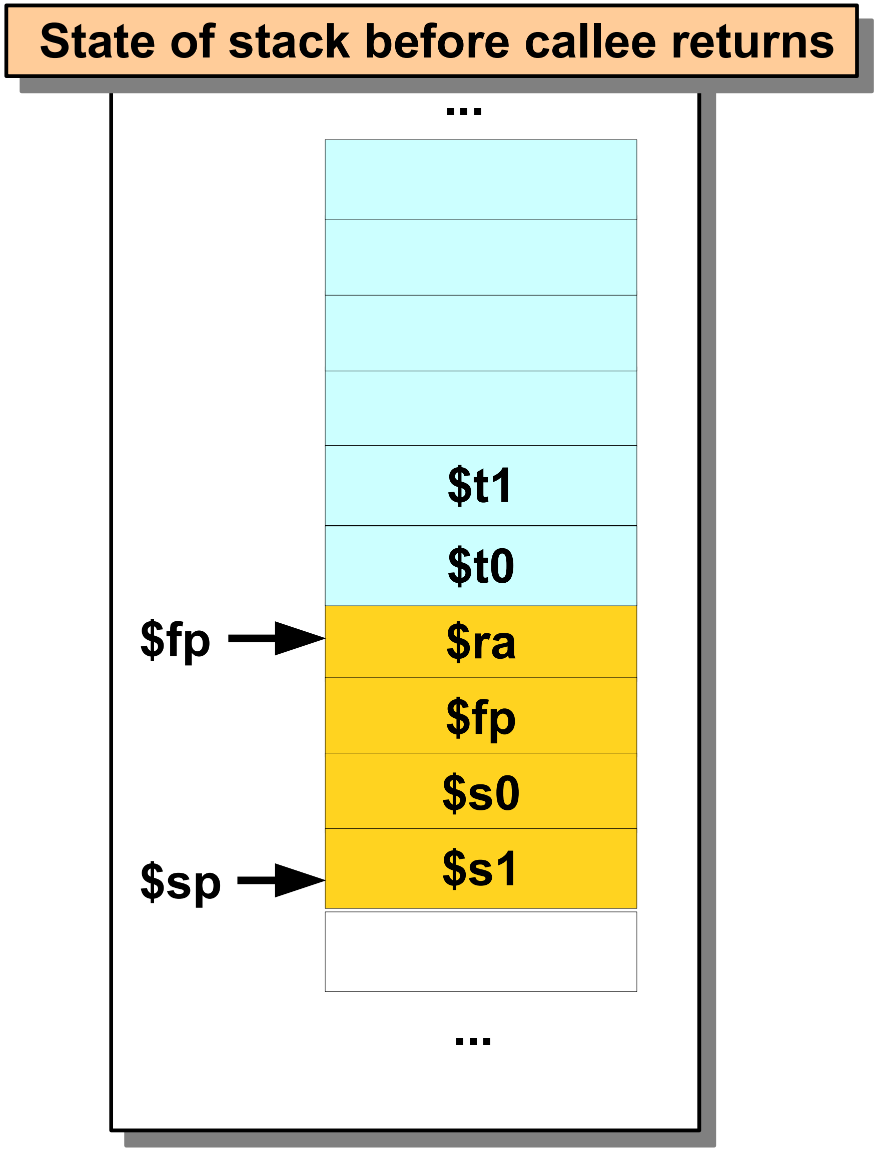
Stack Frame

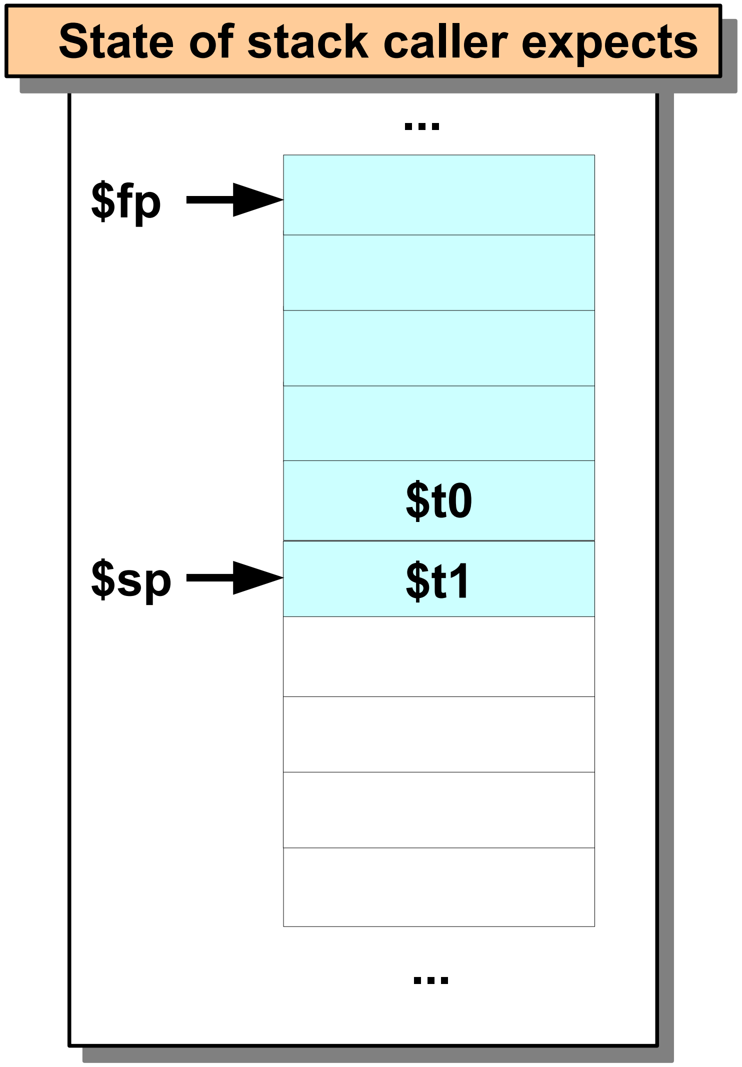
```
func1:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $fp, 0($sp)
addi $fp, $sp, 4
addi $sp, $sp, -8
sw $s0, 4($sp)
sw $s1, 0($sp)

# func1 code
```



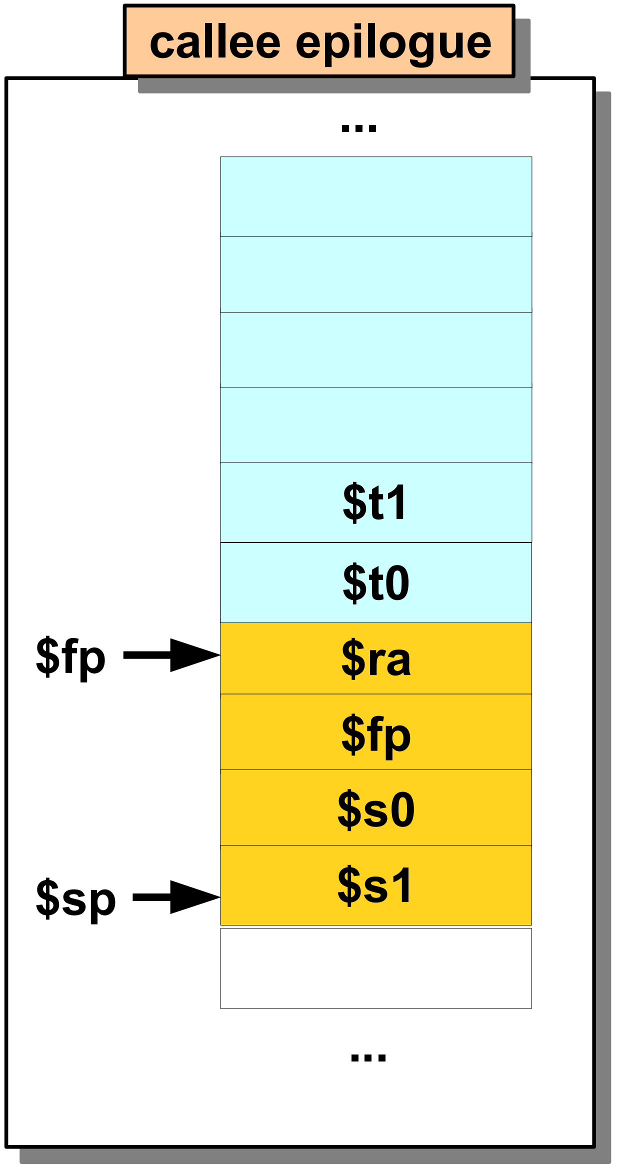
Stack Frame


State of stack before callee returns
...
$t1
$t0
$fp
$ra
$fp
$s0
$sp
$s1
...


State of stack caller expects
...
$fp
$t0
$sp
$t1
...

Stack Frame – Callee Epilogue

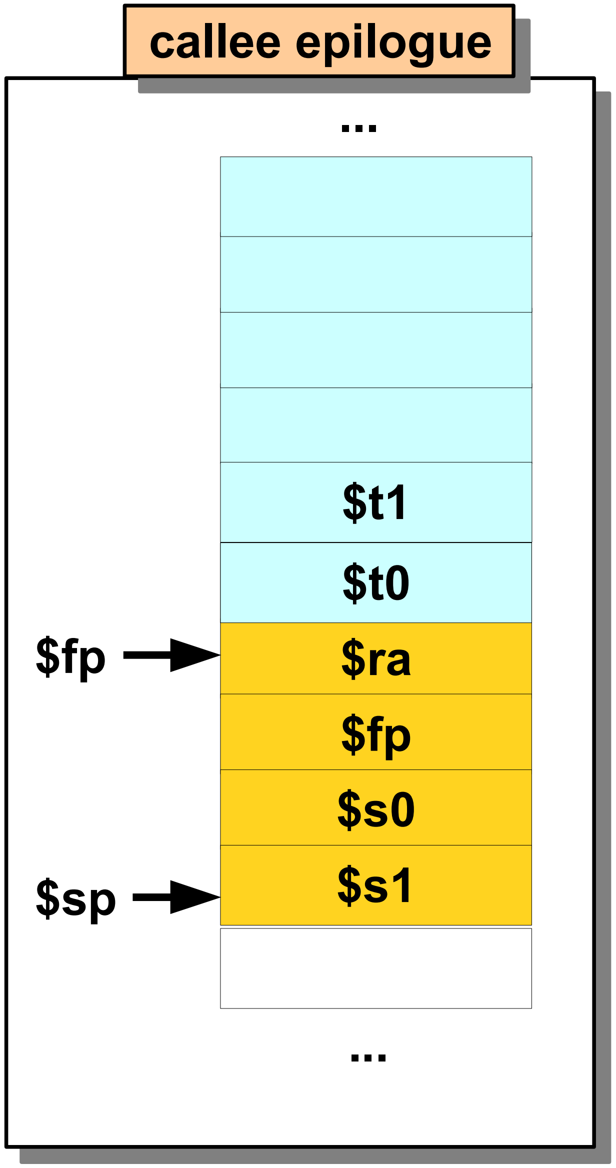
- Before return from callee subroutine:
  - Restore saved regs
  - Restore frame pointer of the caller function
  - Restore return address
  - Return



Stack Frame – Callee Epilogue

- Before return from callee subroutine:
  - Restore saved regs
  - Restore frame pointer of the caller function
  - Restore return address
  - Return

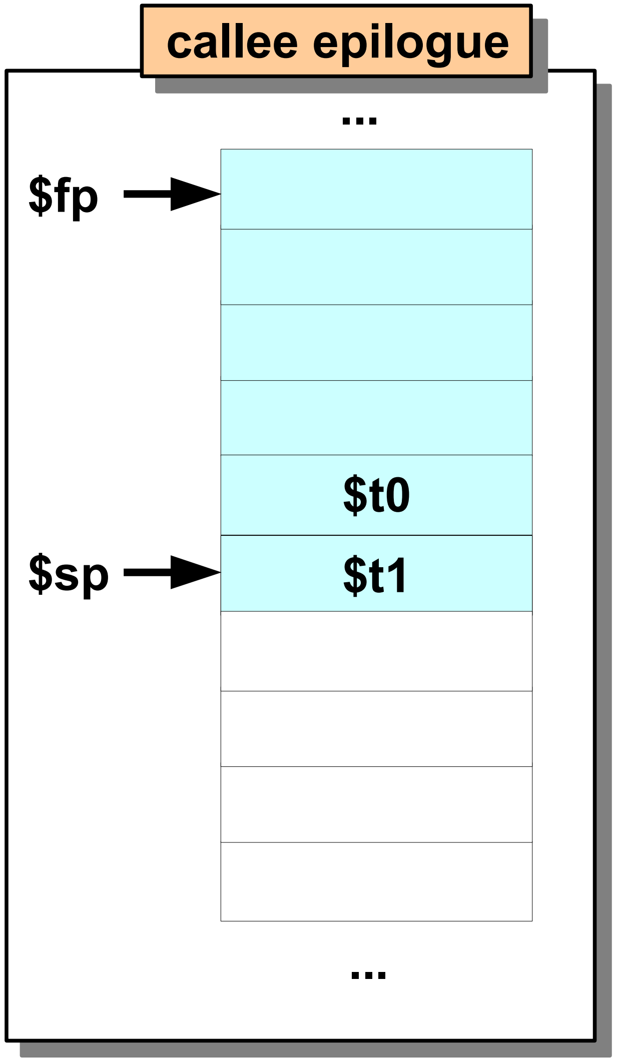
```
....
lw $s1, 0($sp)
lw $s0, 4($sp)
lw $fp, 8($sp)
lw $ra, 12($sp)
addi $sp, $sp, 16
jr $ra
```



Stack Frame – Callee Epilogue

- Before return from callee subroutine:
  - Restore saved regs
  - Restore frame pointer of the caller function
  - Restore return address
  - Return

```
....
lw $s1, 0($sp)
lw $s0, 4($sp)
lw $fp, 8($sp)
lw $ra, 12($sp)
addi $sp, $sp, 16
jr $ra
```

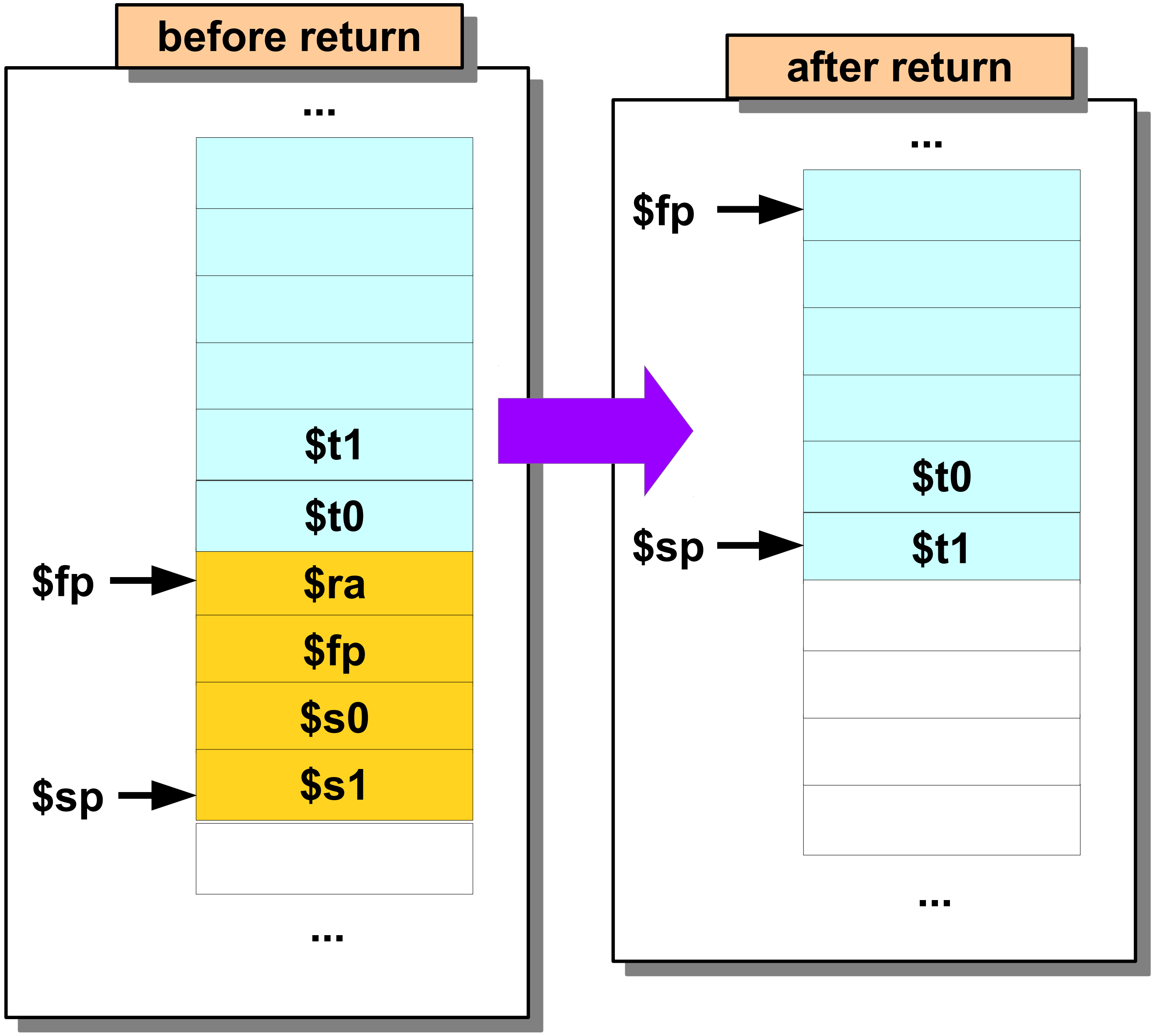


Stack Frame

```
func1:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $fp, 0($sp)
addi $fp, $sp, 4
addi $sp, $sp, -8
sw $s0, 4($sp)
sw $s1, 0($sp)

# func1 code

lw $s1, 0($sp)
lw $s0, 4($sp)
lw $fp, 8($sp)
lw $ra, 12($sp)
addi $sp, $sp, 16
jr $ra
```

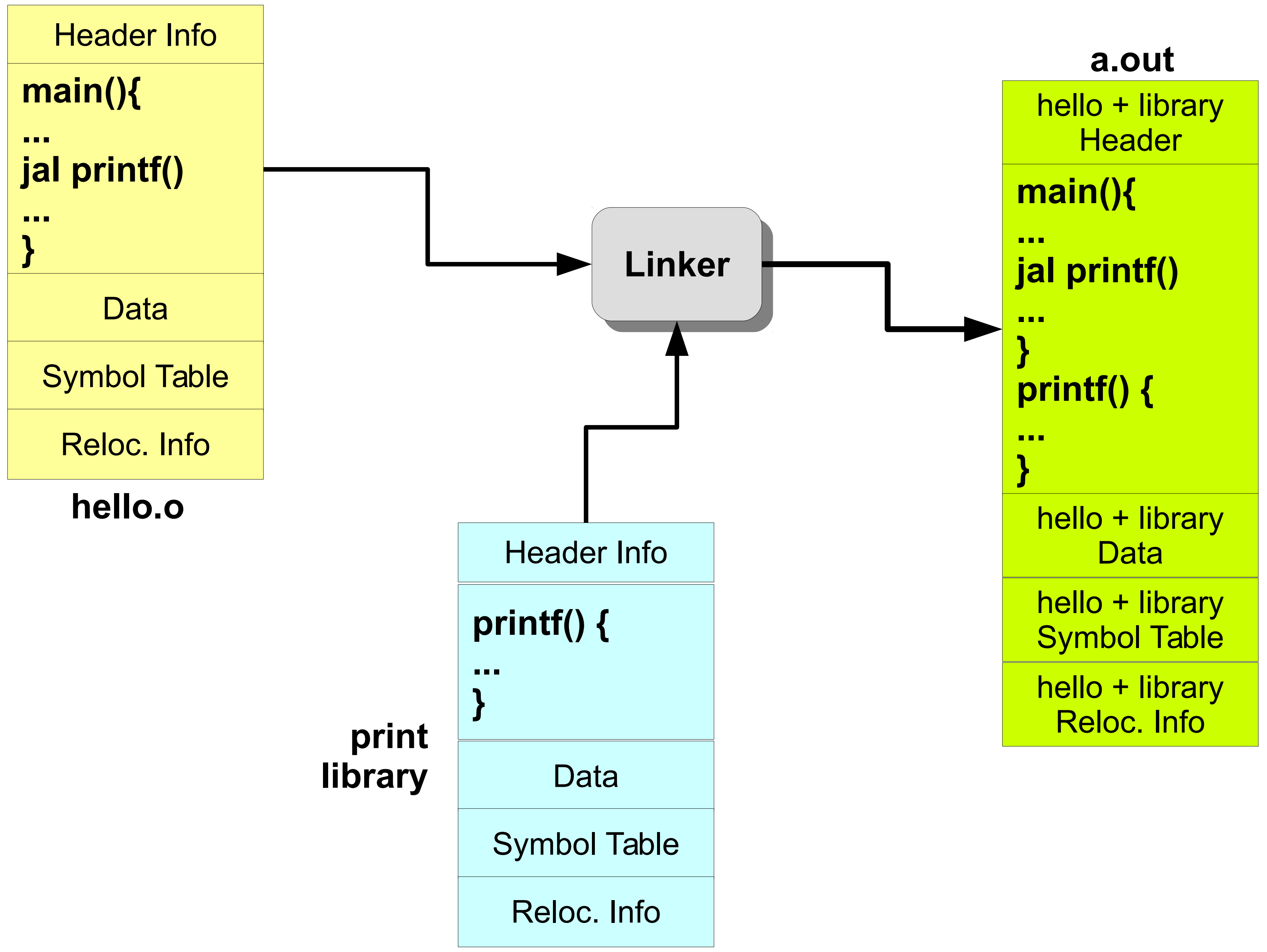


Module Outline

- Addressing modes. Instruction classes.
- MIPS-I ISA.
- Translating and starting a program.
- High level languages, Assembly languages and object code.
- Subroutine and subroutine call. Use of stack for handling subroutine call and return.

Backup

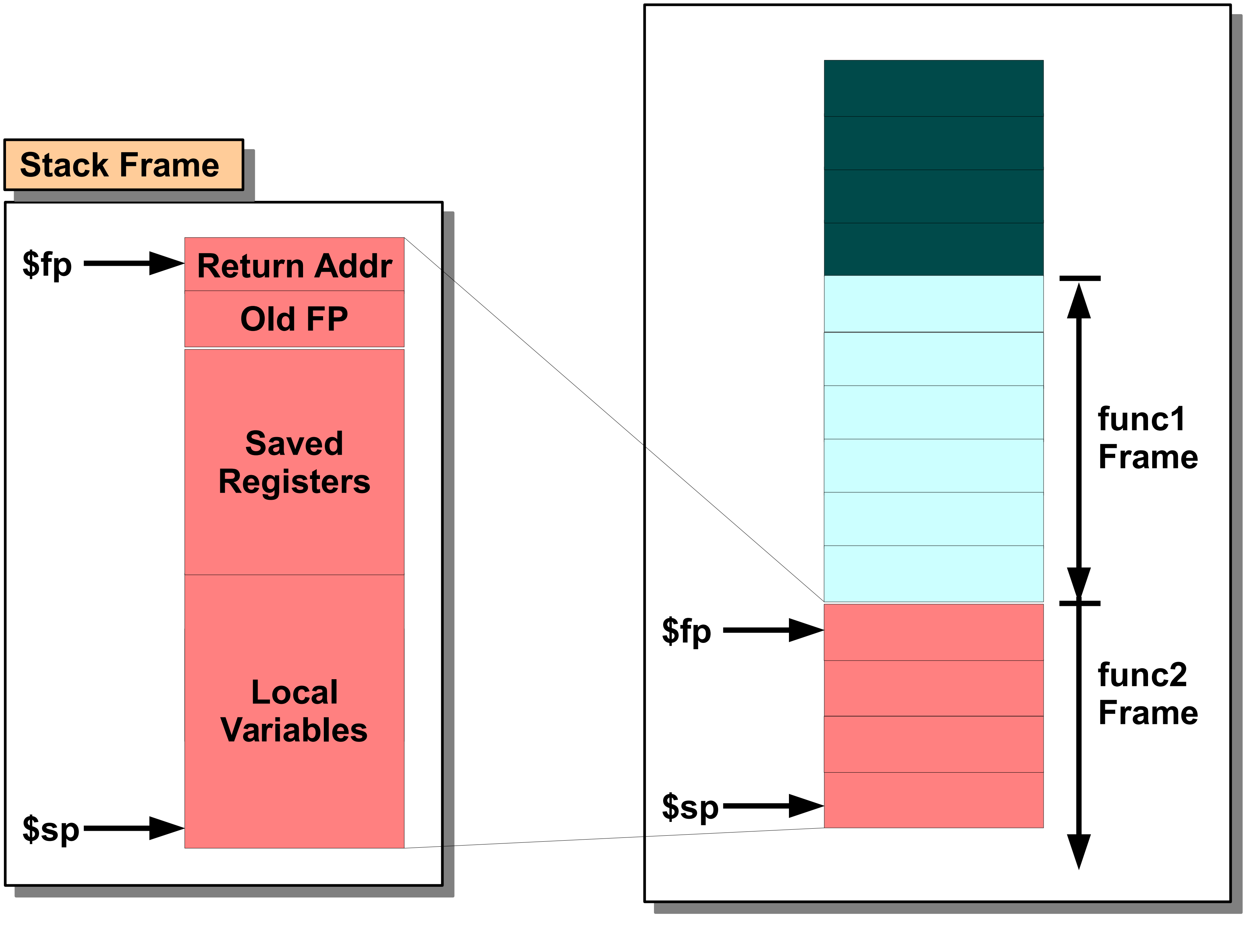
Linking Multiple Modules



The a.out executable

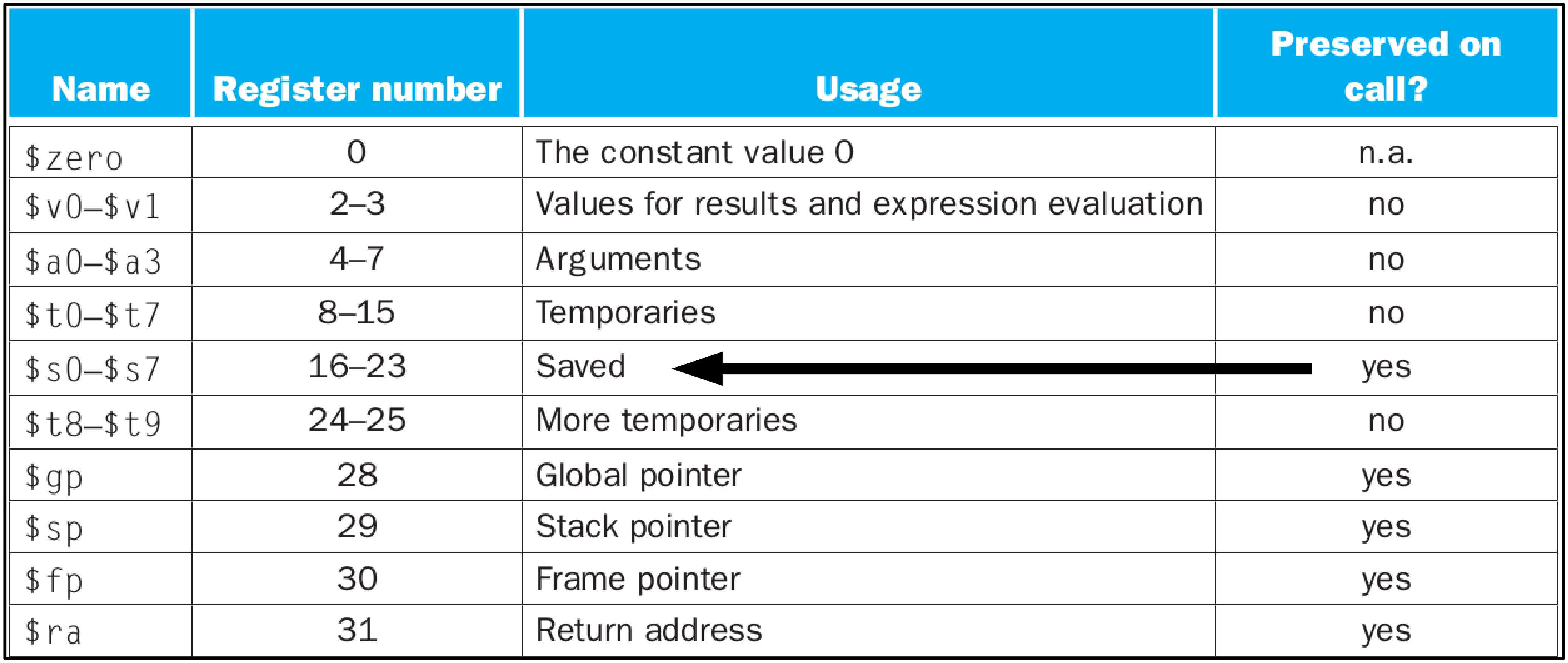
- What does the a.out file contain?
  - Program “code” (machine instructions)
  - Data values (values, size of arrays)
- Other information that is needed for
  - execution
  - debugging
    - Debugging: The stage in program development where mistakes (“bugs”) in the program are identified

Stack Frame – Recall



Saved Registers

- Registers 16 – 23 are saved across function calls



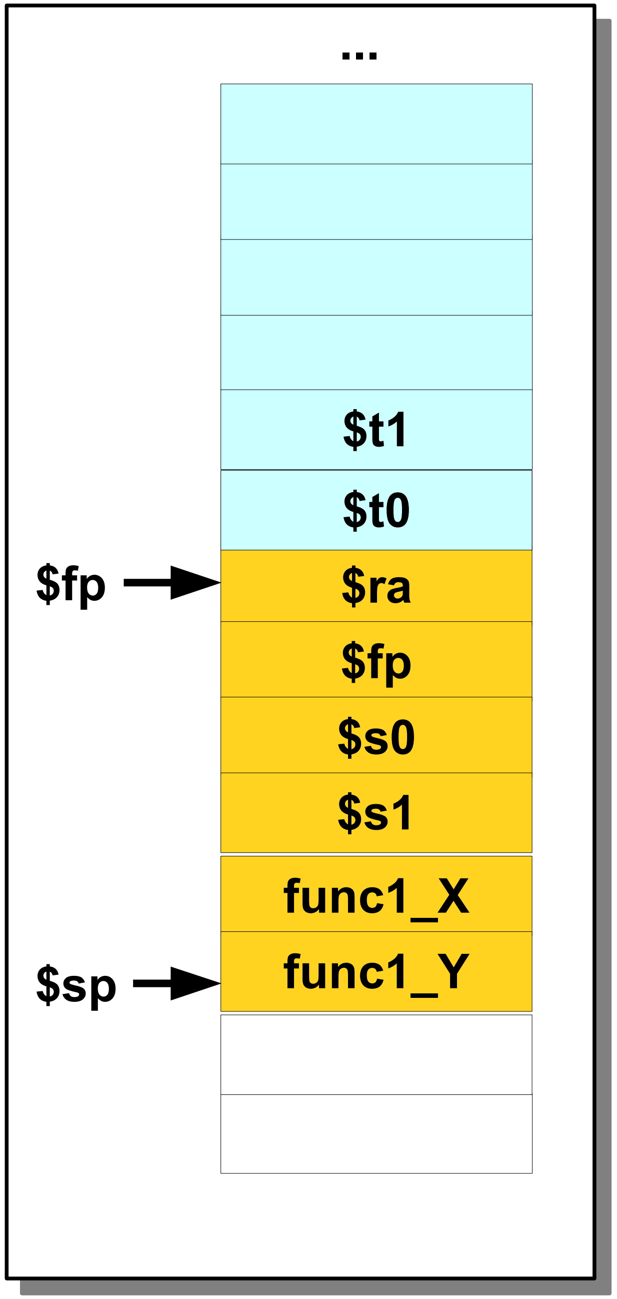
Name	Register number	Usage	Preserved on call?
$zero	0	The constant value 0	n.a.
$v0–$v1	2–3	Values for results and expression evaluation	no
$a0–$a3	4–7	Arguments	no
$t0–$t7	8–15	Temporaries	no
$s0–$s7	16–23	Saved	yes
$t8–$t9	24–25	More temporaries	no
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

Saved Registers

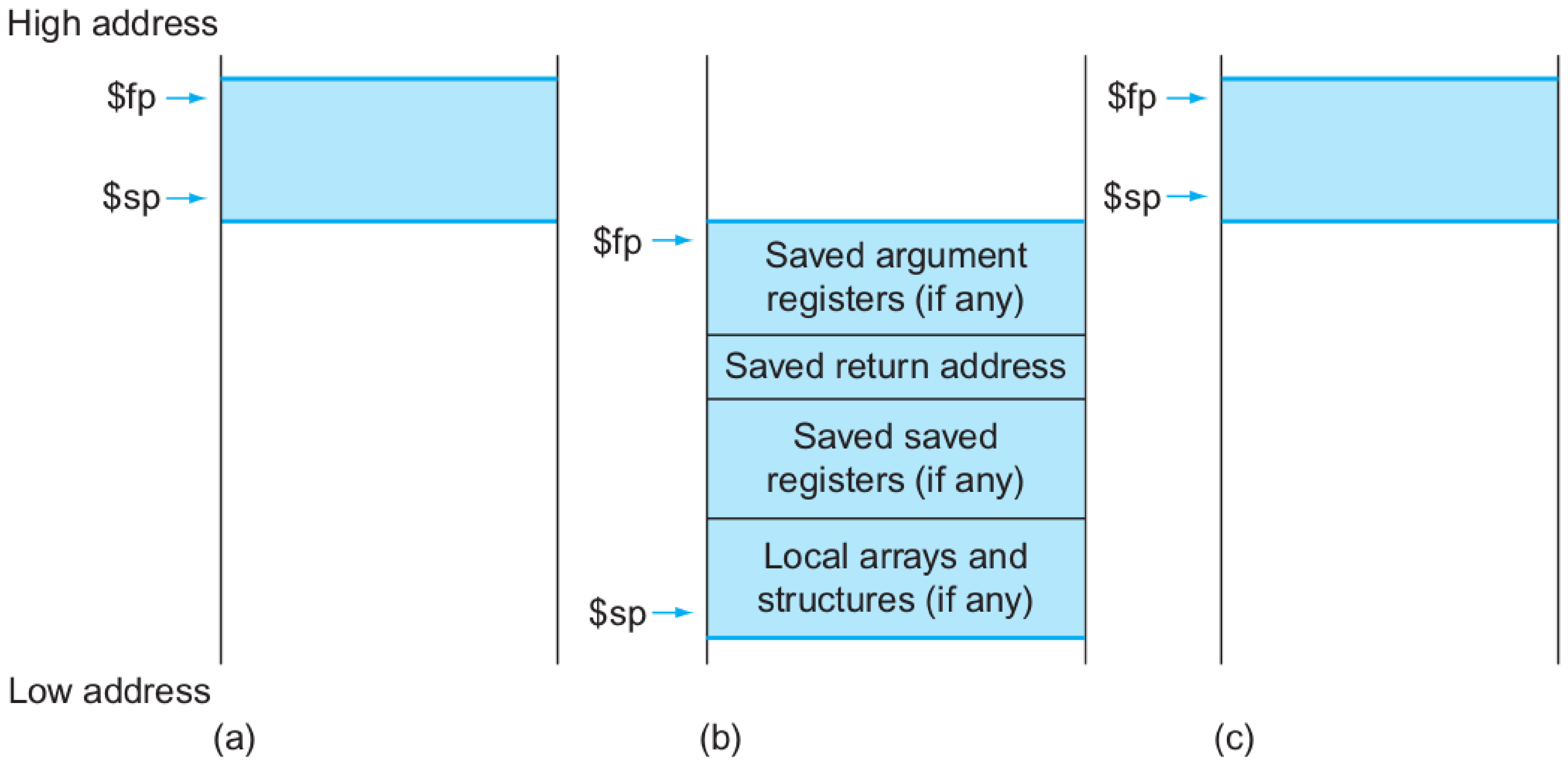
- Registers 16 – 23 are saved across function calls
- Save registers \$s0 - \$s7 if used by the callee
- Example: \$s0, \$s1 are saved

Stack Frame

- Local variables are allocated on the stack after the saved registers



Stack Frame


High address
$fp
$sp
$fp
Saved argument registers (if any)
Saved return address
Saved saved registers (if any)
Local arrays and structures (if any)
$sp
$fp
$sp
Low address
(a)
(b)
(c)