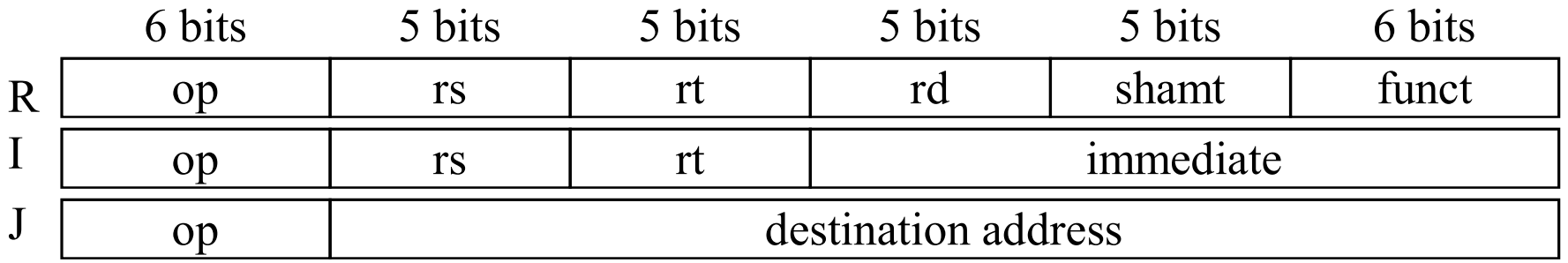
Procedure Support

Review

- 3 formats of MIPS instructions in binary:
  - Op field determines format

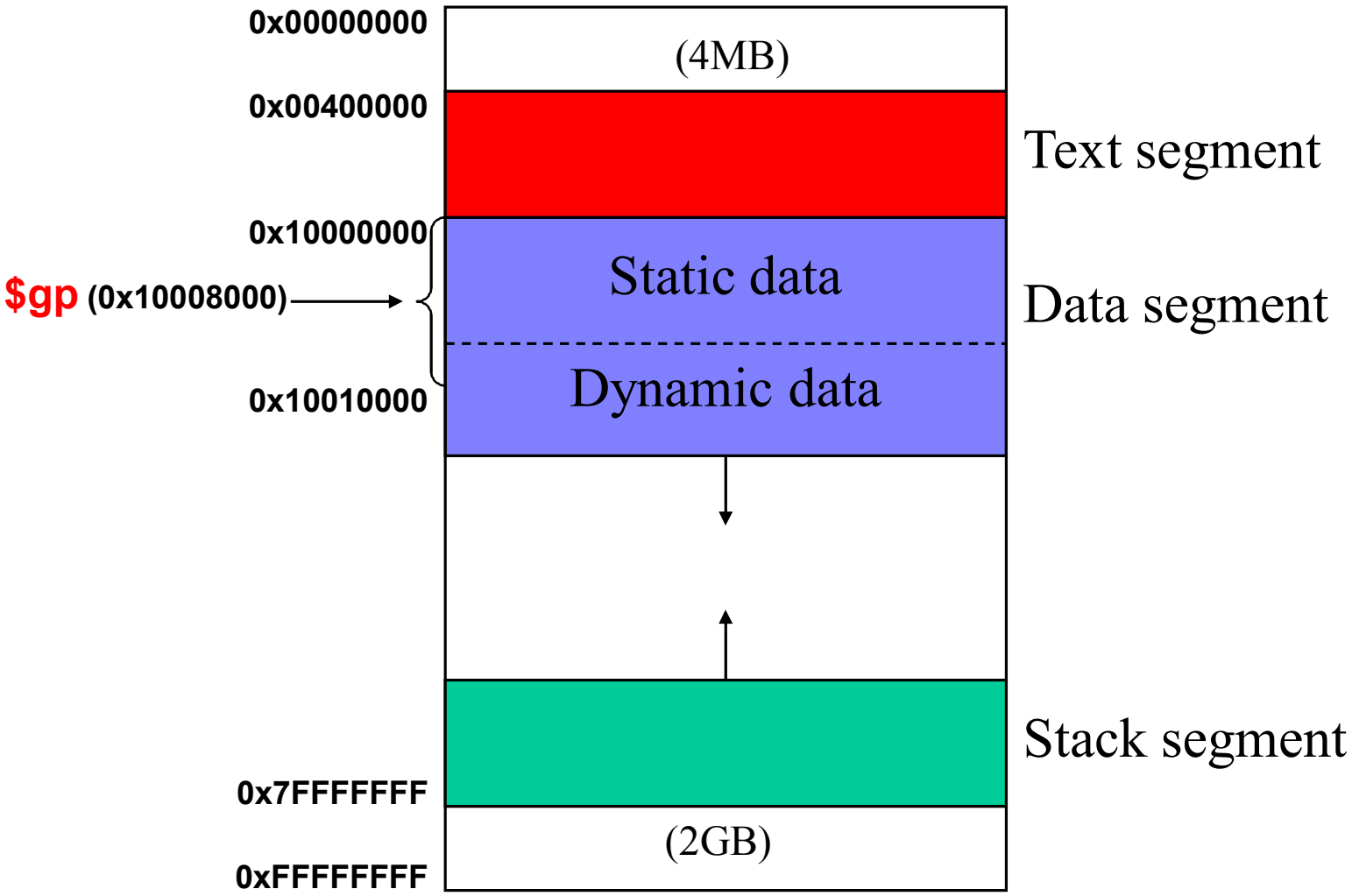


- Operands
  - Registers: $0 to $31 mapped onto: `$zero, $at, $v_, $a_, $s_, $t_, $gp, $sp, $fp, $ra`
  - Memory : `Mem[0], Mem[4], … , Mem[4294967292]`
    - Index is the “address” (Array index => Memory index)
- Stored program concept (instructions are numbers!)

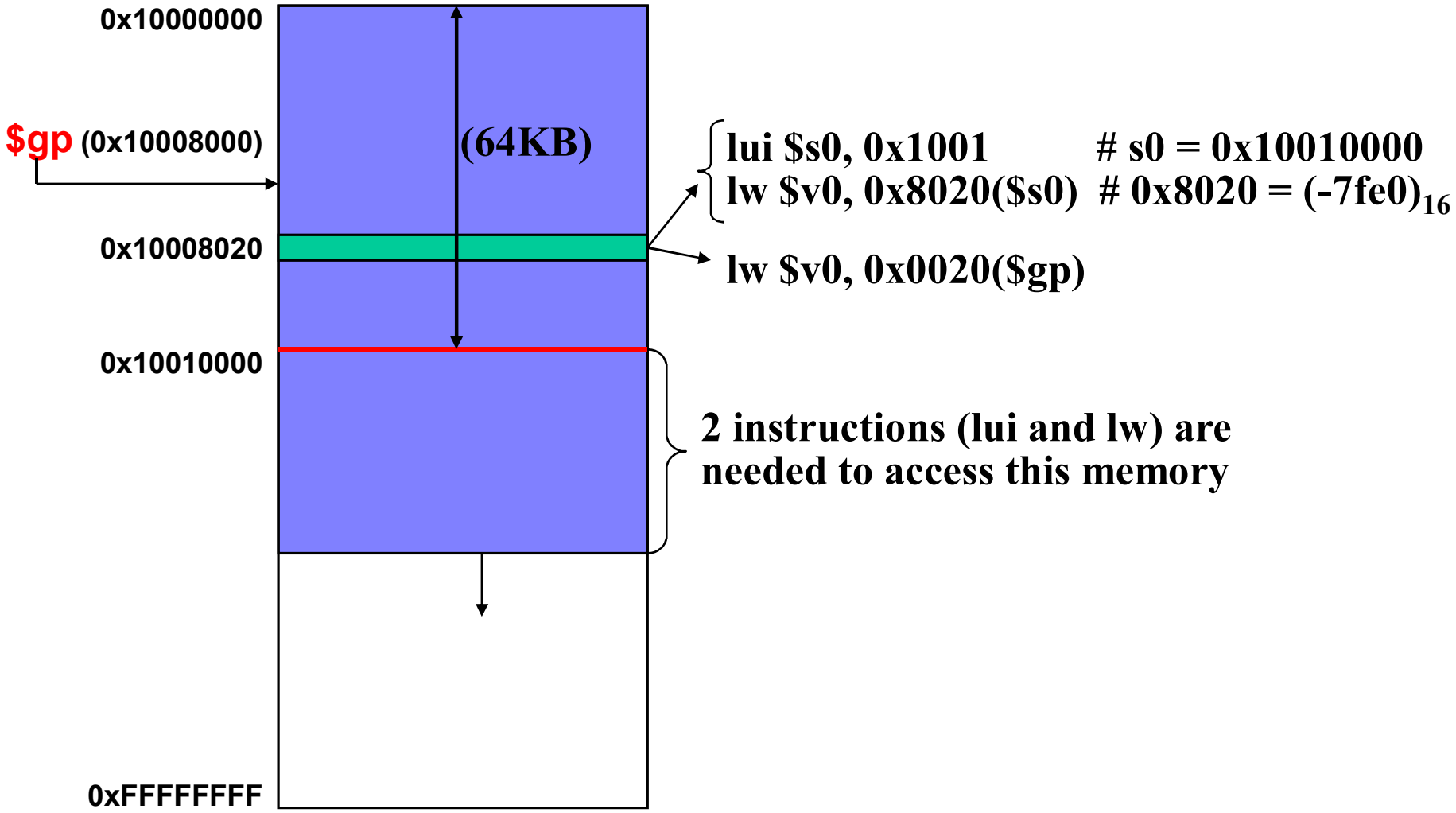
Overview

- Memory layout
- C functions
- MIPS support (instructions) for procedures
- The stack
- Procedure Conventions
- Manual Compilation
- Conclusion

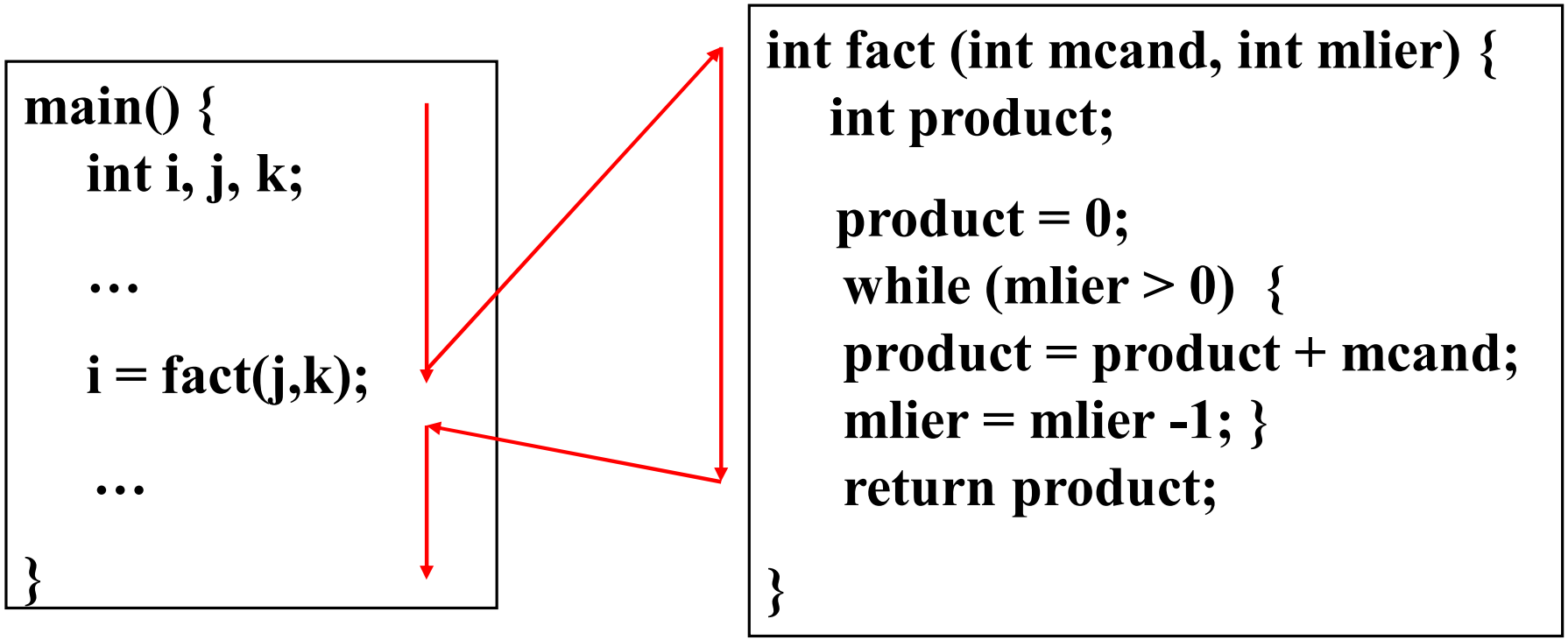
Memory Layout



Data Segment



C Functions / Procedures



What information must compiler keep track of?

Procedure Call Bookkeeping

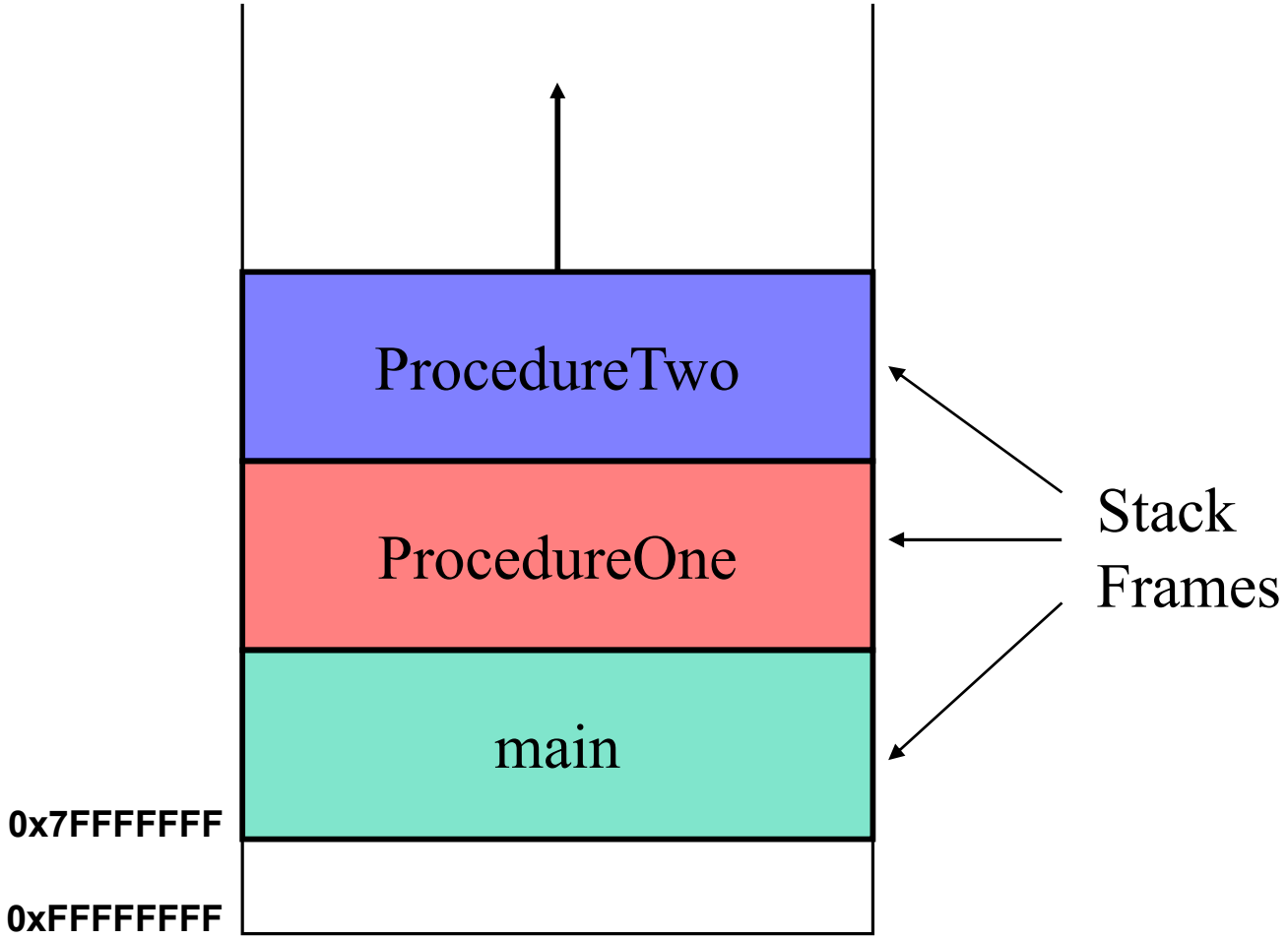
- Problems
  - Procedure address
  - Return address
  - Arguments
  - Return value
  - Local variables
- Register conventions
  - Labels
  - $ra
  - $a0, $a1, $a2, $a3
  - $v0, $v1
  - $s0, $s1, …, $s7
- Dynamic nature of procedures
  - Procedure call frames
    - Arguments, save registers, local variables

Procedure Call Convention

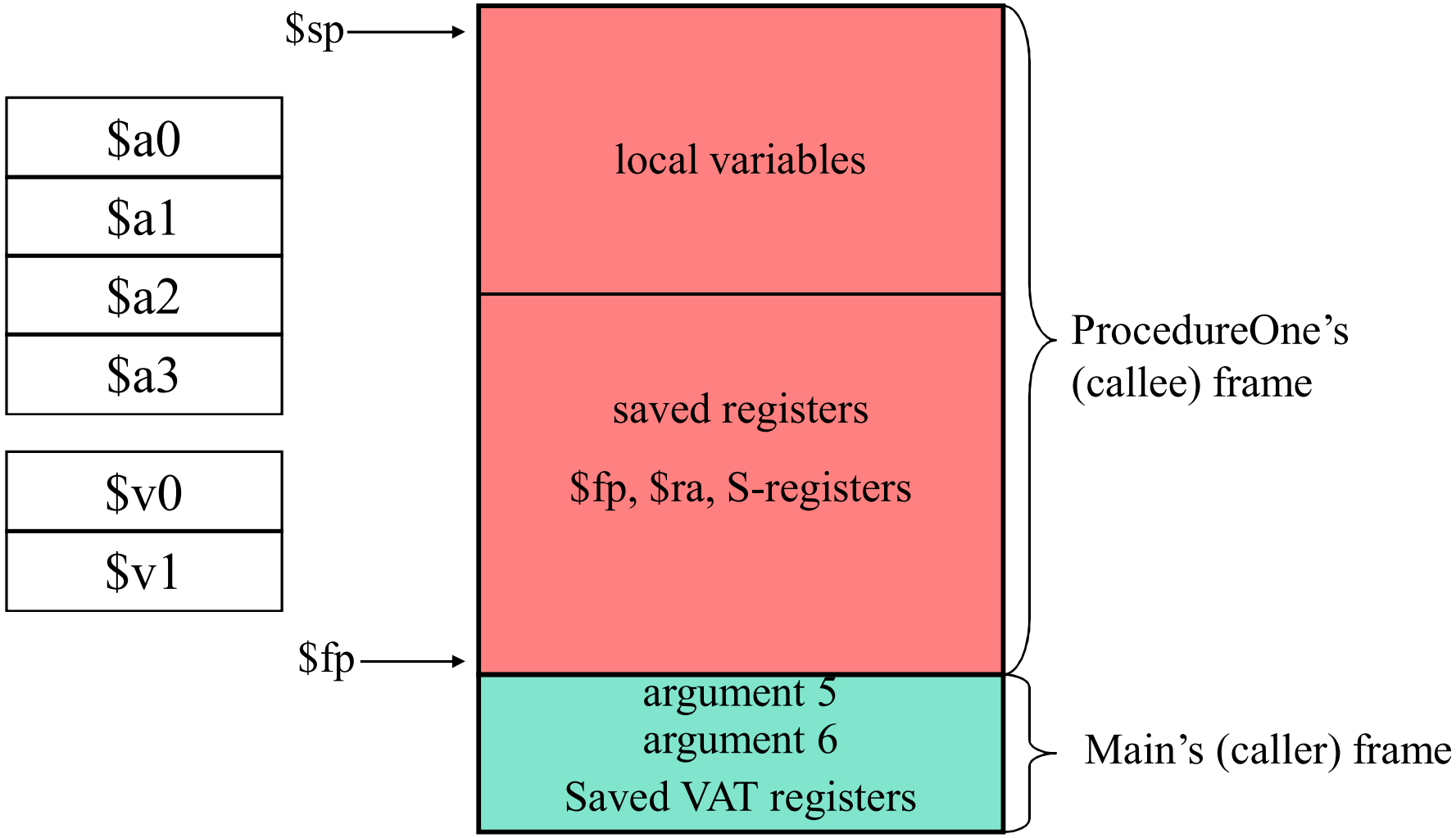
- **Software** rules for using registers

Name	Register Number	Usage	Preserved on call
$zero	0	the constant value 0	n.a.
$at	1	reserved for the assembler	n.a.
$v0-$v1	2-3	expr. evaluation and function result	no
$a0-$a3	4-7	arguments (procedures/functions)	yes
$t0-$t7	8-15	temporaries	no
$s0-$s7	16-23	saved	yes
$t8-$t9	24-25	more temporaries	no
$k0-$k1	26-27	reserved for the operating system	n.a.
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes

The Stack


ProcedureTwo
ProcedureOne
main
Stack
Frames
0x7FFFFFFF
0xFFFFFFFF

Stack Frames



Caller / Callee Conventions

- **Immediately before the caller invokes the callee**
  - Pass arguments ($a0 - $a3). Extra args: push on the stack
  - Save caller-saved registers ($a0 - $a3; $t0 - $t9)
  - Execute **jal** (jumps and links to callee; saves return addr)
- **Just before the callee starts executing**
  - Allocate memory for the frame ($sp = $sp – fsize)
  - Save callee-saved registers ($s0-$s7; $fp; $ra)
  - $fp = $sp + (fsize – 4)
- **Immediately before the callee returns to caller**
  - Place the returned value in register $v0
  - Restore all callee-saved registers
  - Pop the stack frame ($sp = $sp + fsize); restore $fp
  - Return by jumping to the address in $ra

Procedure Support

```
main( ) {
    …
    s = sum (a, b);
    …
}
```

```
int sum(int x, int y) {
    return x + y;
}
```

address

```
1000 add  $a0,$s0,$zero   # $a0 = x
1004 add  $a1,$s1,$zero   # $a1 = y
1008 addi $ra,$zero,1016  # $ra=1016
1012 j    sum             # jump to sum
1016 ...
2000 sum: add $v0,$a0,$a1
2004 jr   $ra             # jump to 1016
```

Jump and Link Instruction

- Single instruction to jump and save return address
  - **jal**: **jump and link (*Make the common case fast*)**
  - **J Format Instruction: jal label**
  - **Should be called *laj***
    1. **(link): save address of next instruction into $ra**
    2. **(jump): jump to label**

```
1000 add  $a0,$s0,$zero  # $a0 = x
1004 add  $a1,$s1,$zero  # $a1 = y
1008 jal  sum  # $ra = 1012; jump to sum
1012 ...
2000 sum: add $v0,$a0,$a1
2004 jr   $ra                # jump to 1012
```

Nested Procedures

```
int sumSquare(int x, int y) {
   return mult(x,x) + y;
}
```

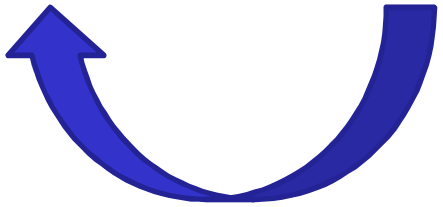
$sp = 0x7fffeffc

```
sumSquare:
        subi $sp,$sp,12      # space on stack
        sw $ra,$ 8($sp)      # save ret addr
        sw $a0,$ 0($sp)      # save x
        sw $a1,$ 4($sp)      # save y
        addi $a1,$a0,$zero   # mult(x,x)
        jal mult             # call mult
        lw $ra,$ 8($sp)      # get ret addr
        lw $a0,$ 0($sp)      # restore x
        lw $a1,$ 4($sp)      # restore y
        add $vo,$v0,$a1      # mult()+y
        addi $sp,$sp,12      # free stack space
        jr $ra
```

Example (1/2)

```
main( ) {
     int f;
     f = fact (10);
     printf (“Fact(10) = %d\n”, f);
}
```

```
int fact ( int n) {
    if (n < 1)
      return (1);
    else
       return (n * fact(n-1);
}
```



```
.text
main:   li      $a0, 10
        jal     fact
        li      $v0, 1
        addu    $a0, $v0, $zero
        syscall
        li      $v0, 10
        syscall
```

Example (2/2)

```
fact:   addi   $sp, $sp, -8        # adjust stack for 2 items
        sw     $ra, 4($sp)         # save the return address
        sw     $a0, 0($sp)         # save the argument n
        slti   $t0, $a0, 1         # test for n < 1
        beq    $t0, $zero, L1      # if n >= 1, go to L1
        addi   $v0, $zero, 1       # if  (n<1),  result is 1
        addi   $sp, $sp, 8         #   pop 2 items from stack
        jr     $ra                 #       return to caller
L1:     addi   $a0, $a0, -1        # else     n >= 1; agrument gets (n-1)
        jal    fact                #   call fact with (n-1)
        lw     $a0, 0($sp)         # return from jal: restore original n
        lw     $ra, 4($sp)         # restore the return address
        addi   $sp, $sp, 8         # adjust stack pointer to pop 2 items
        mul    $v0, $a0, $v0       # return n * fact (n-1)
        jr     $ra                 # and return to the caller
```

Summary

- Caller / callee conventions
  - Rights and responsibilities
  - Callee uses VAT registers freely
  - Caller uses S registers without fear of being overwritten
- Instruction support: jal *label* and jr $ra
- Use stack to save anything you need. *Remember to leave it the way you found it*
- Register conventions
  - Purpose and limits of the usage
  - Follow the rules even if you are writing all the code yourself