Procedure Detail

CSE 410 - Computer Systems

October 10, 2001

Readings and References

- Reading
- Other References
  - D. Sweetman, See MIPS Run, Morgan Kauffman, Publishers
    - Chapter 10, C Programming on MIPS

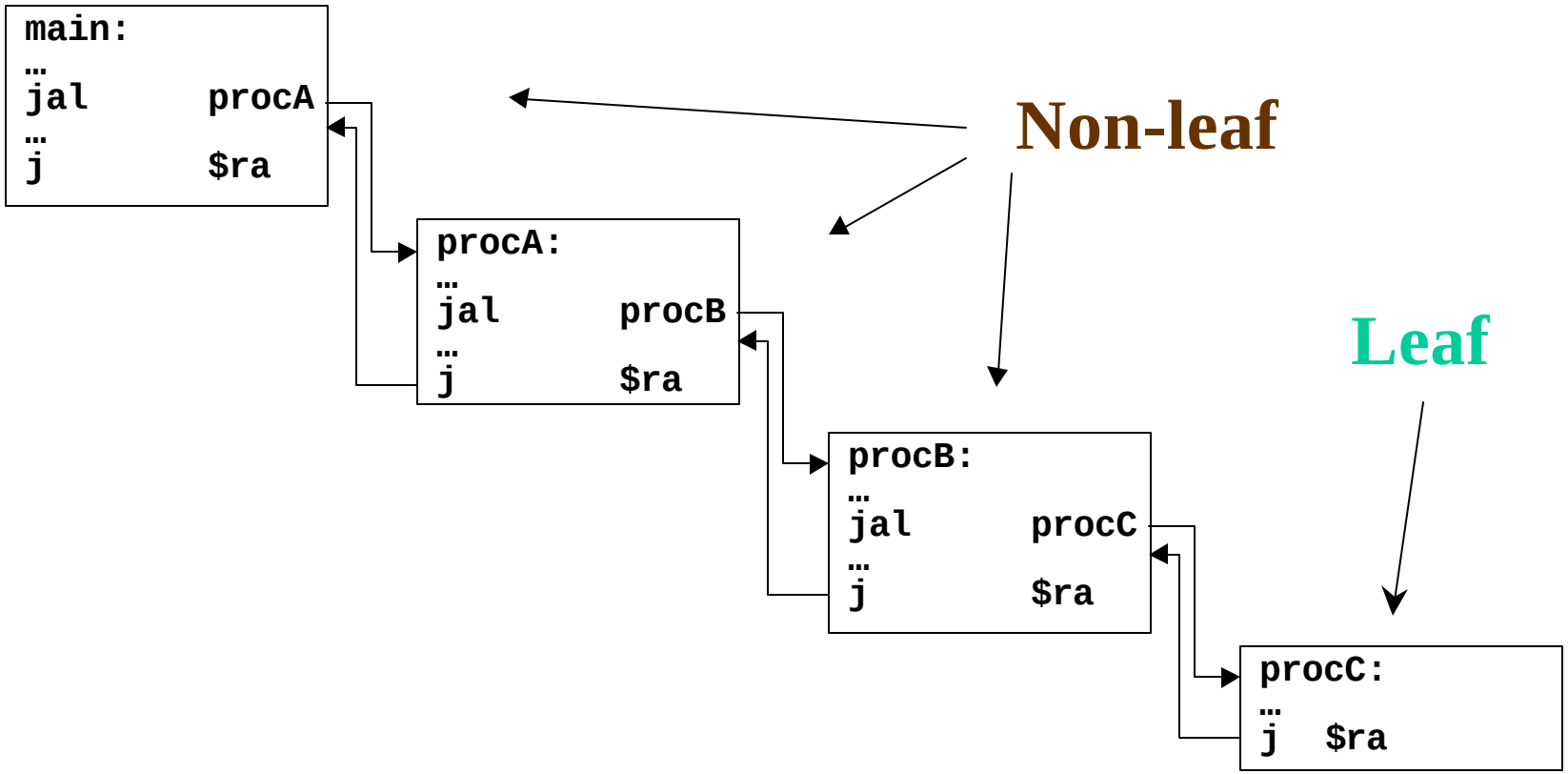
Leaf procedures

- A leaf procedure is one that does not call another procedure
- Relatively simple register usage since the procedure doesn’t call anyone else
- Little or no memory access requirements because you are not saving and restoring as many registers from the stack

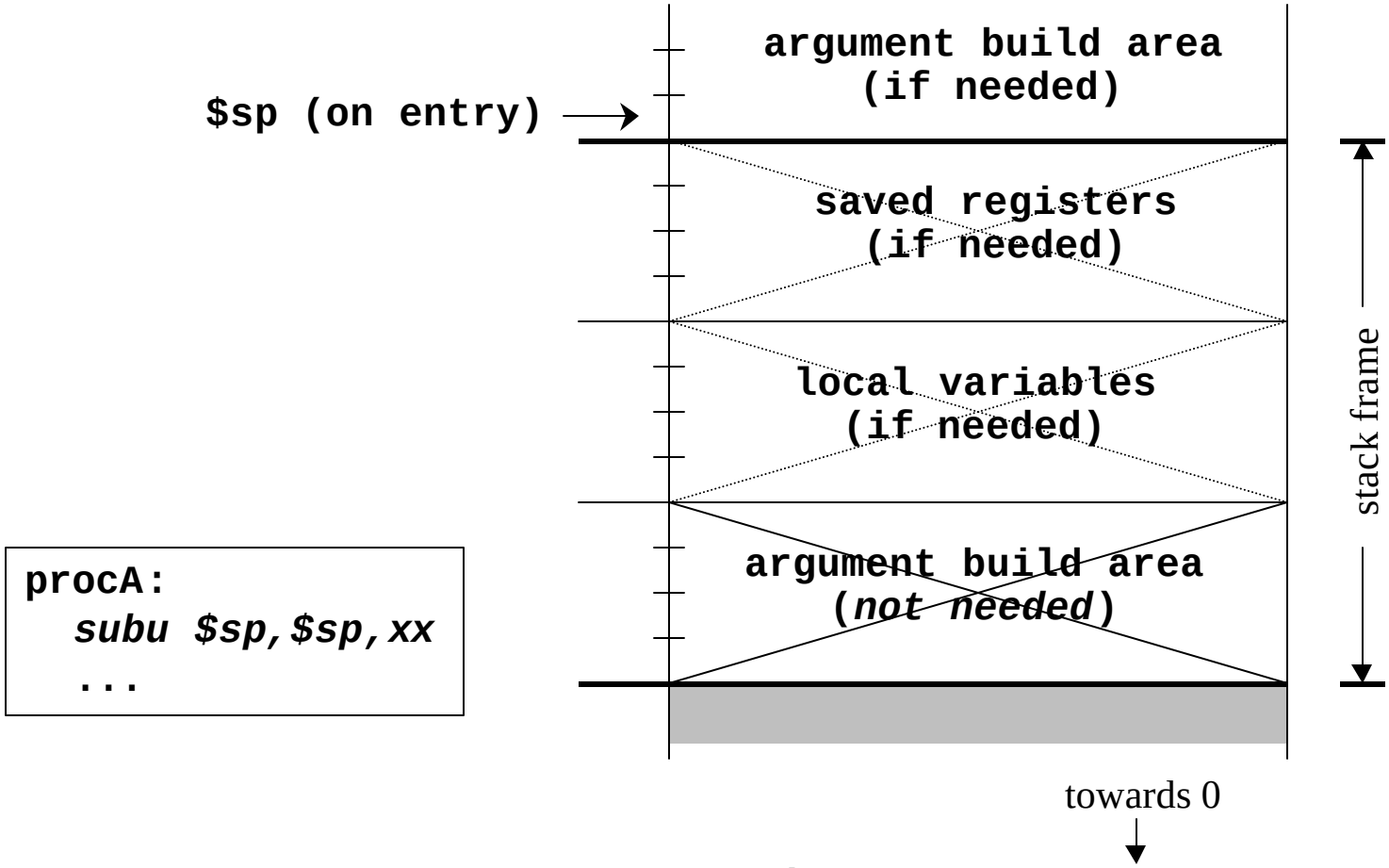
Non-leaf procedure

- A non-leaf procedure is one that calls another procedure
- You must save at least register $ra, since that register is overwritten by the jal when you call another procedure

Calling tree

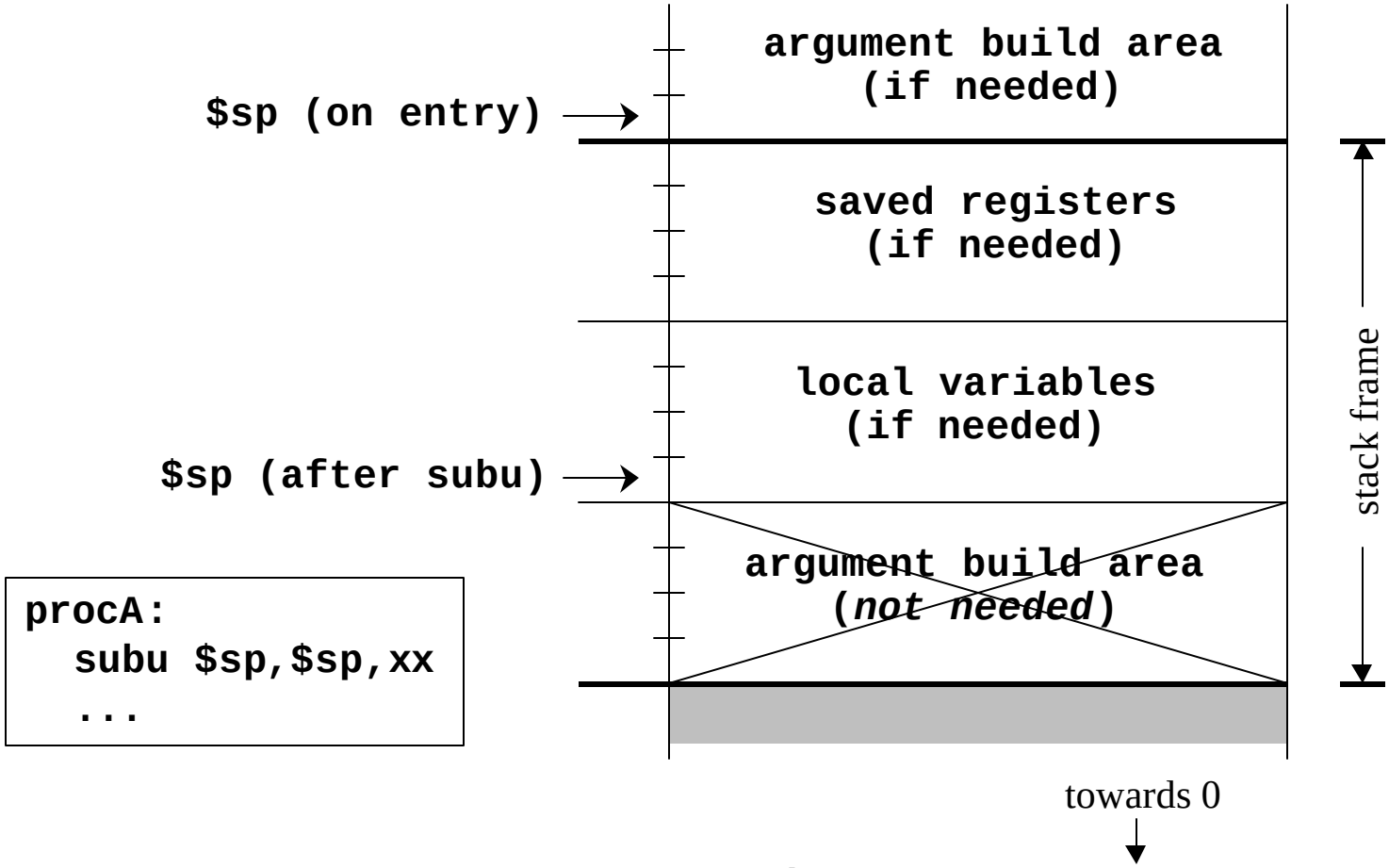


Layout of stack frame (little leaf)



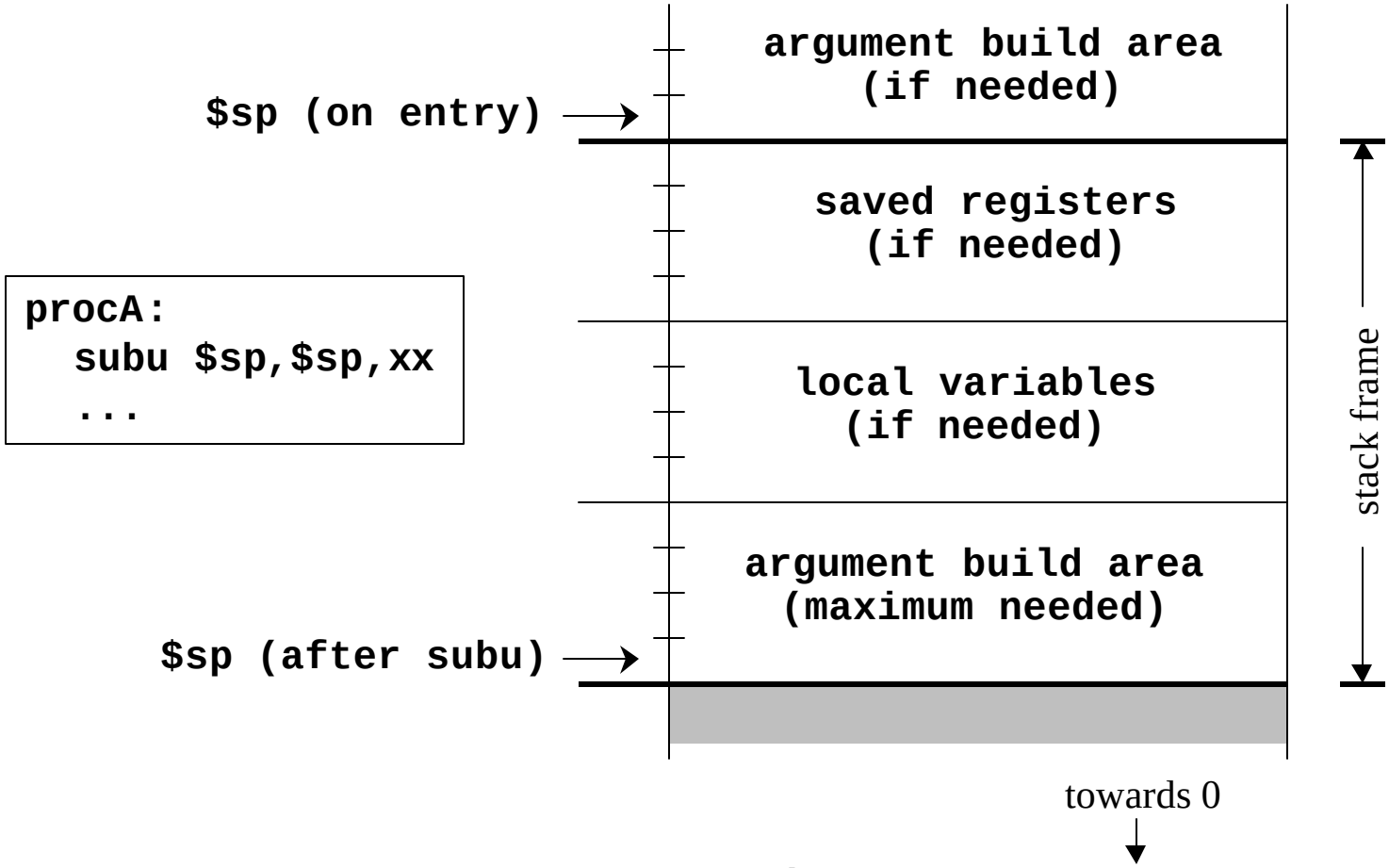

```
procA:
   subu $sp,$sp,xx
   ...
```

Layout of stack frame (big leaf)




```
procA:
  subu $sp,$sp,xx
  ...
```

Layout of stack frame (non-leaf)



Little leaf example - swap.c

```
/* Swap two integer array elements */

void swap(int a[], int i, int j)
{
        int T;
        T = a[i];
        a[i] = a[j];
        a[j] = T;
}
```

Little leaf example - swap.s

```
swap:
   sll     $a1,$a1,2          # $a1 = 4*i
   addu    $a1,$a1,$a0        # $a1 = addr(a[i])
   lw      $v1,0($a1)         # $v1 = a[i]
   sll     $a2,$a2,2          # $a2 = 4*j
   addu    $a2,$a2,$a0        # $a2 = addr(a[j])
   lw      $v0,0($a2)         # $v0 = a[j]
   sw      $v0,0($a1)         # a[i] = old a[j]
   sw      $v1,0($a2)         # a[j] = old a[i]
   j       $ra                # return
```

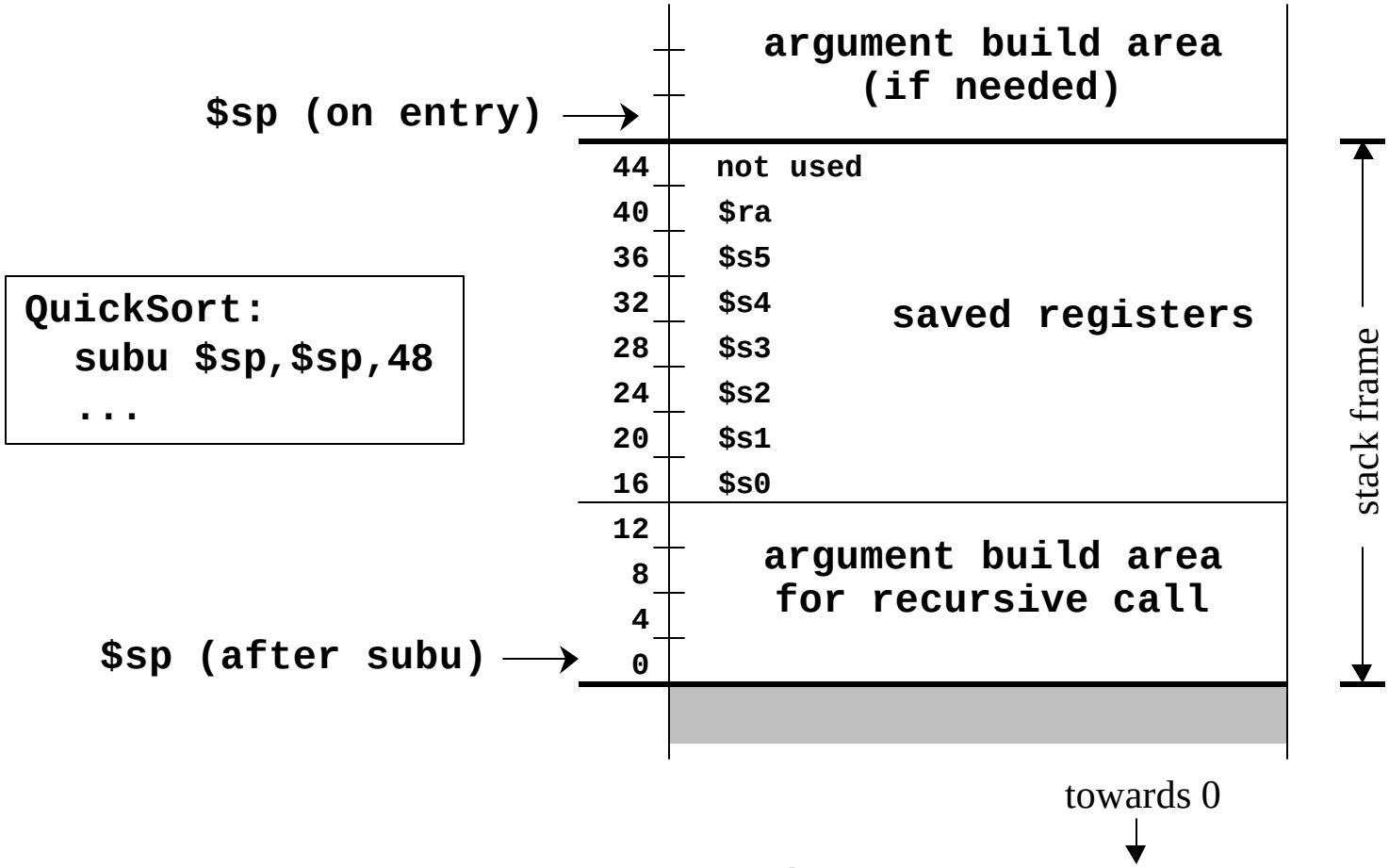
Non-leaf example - QuickSort.c

```
void QuickSort(int a[], int lo0, int hi0)
{
        int lo = lo0;
        int hi = hi0;
        int mid;

        if ( hi0 > lo0)
        {
        ...
```

Non-leaf example - QuickSort.s

```
QuickSort:
  subu    $sp,$sp,48          # create stack frame
  sw      $ra,40($sp)         #
  sw      $s5,36($sp)         #
  sw      $s4,32($sp)         #
  sw      $s3,28($sp)         #
  sw      $s2,24($sp)         #
  sw      $s1,20($sp)         #
  sw      $s0,16($sp)         #
  move    $s3,$a0             # $s3 = address(a)
  move    $s5,$a1             # $s5 = lo0
  ...
```

Layout of QuickSort stack frame



$ra - Return Address

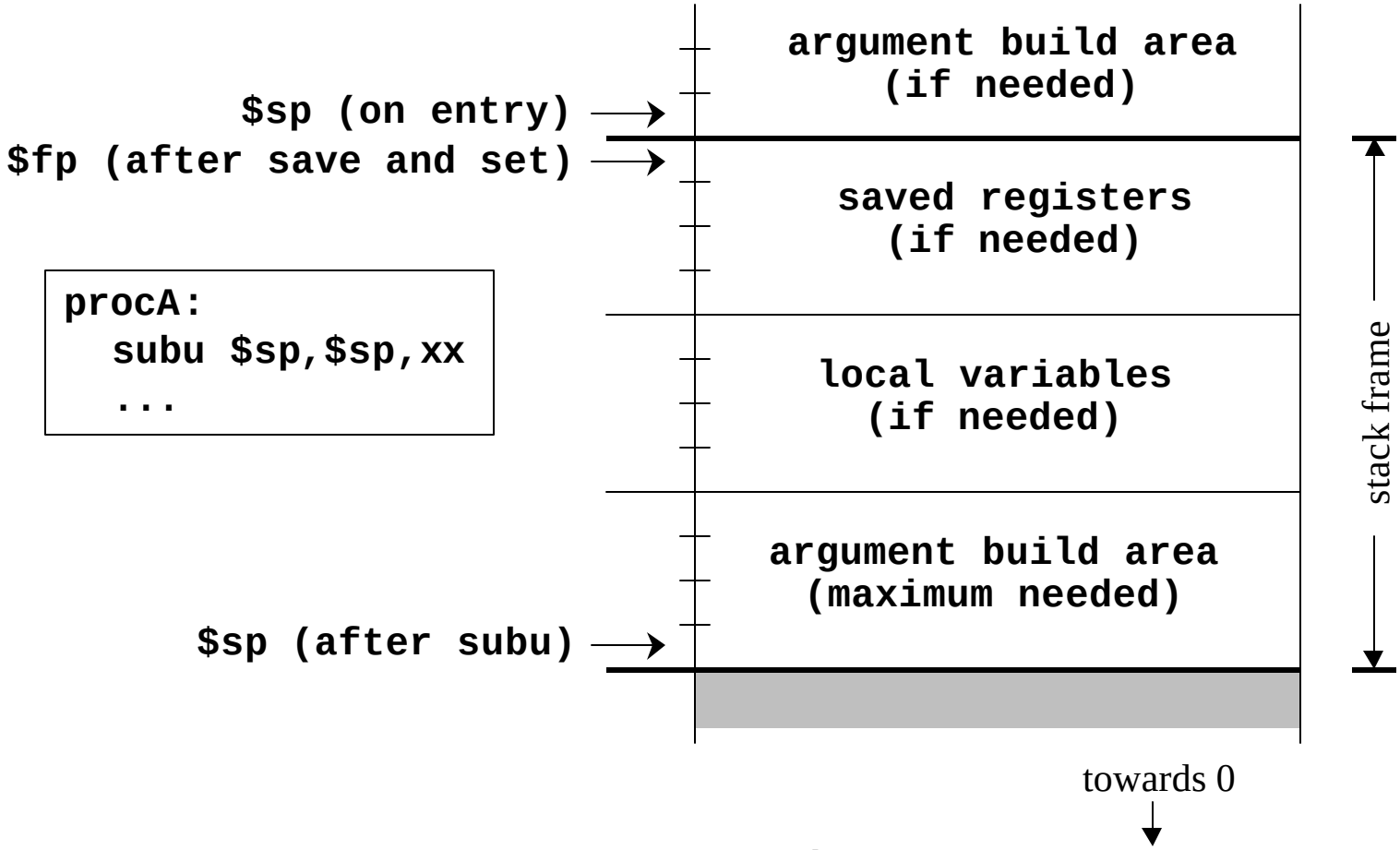
- Return address register
  - written with jal, jalr instructions
  - must be saved if procedure calls another

```
QuickSort:
  subu    $sp,$sp,48    # create stack frame
  sw      $ra,40($sp)   #
  . . .
  lw      $ra,40($sp)   # restore from stack …
  addu    $sp,$sp,48    #
  j       $ra           # return
```

$fp - Frame Pointer

- Frame pointer points to the largest address in the stack frame
- Stack pointer points to the smallest address in the stack frame
  - no advantage to $fp if $sp does not change during procedure's execution
- Consider $fp to be $s8
  - save and restore required if you use it

Layout of stack frame (with $fp)



$s0-$s7 - Save and Restore

- These registers are available for unlimited use
- Must save immediately on procedure entry and restore just before procedure exit if you are going to use them
- As a result of this convention, the registers will have the same values after a procedure call as they had before

$t0-$t9 - Temporary registers

- Use however you like
- No save and restore required or expected
- As a result of this convention, the registers have no guaranteed values when you get back from calling another procedure

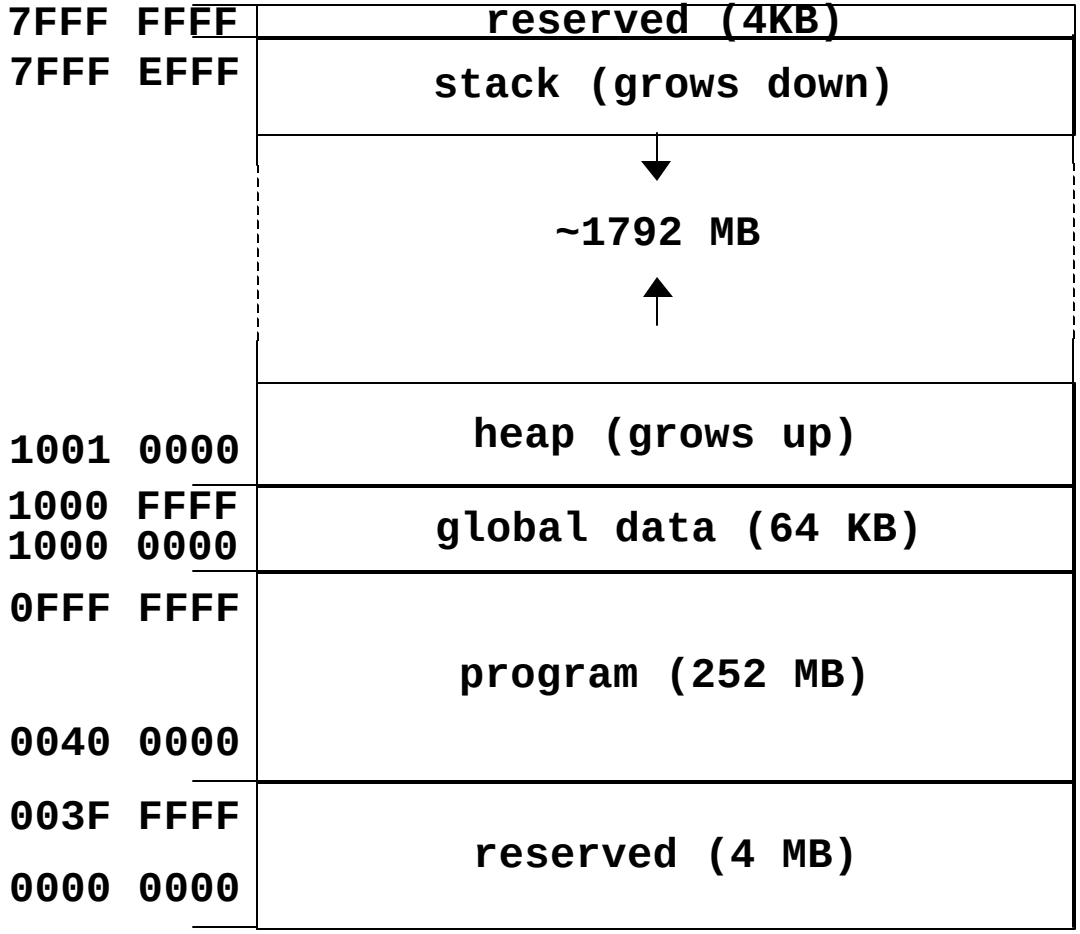
$a0-$a3 , $v0-$v1 - Args/Return

- The argument registers can be changed in a procedure without restriction
- No guarantee that they will be the same upon return from a called procedure
- The result registers will contain whatever the function prototype says they will
  - undefined value in $v1 if not used for return

$gp - Global Pointer

- Initialized so that it points to the middle of a 64KB section of the data segment
  - address **`0x10008000`**
- Variables placed in this section can be accessed without loading a 32-bit address
  - **`lw $t0,-32768($gp)`**
- Assembler directive
  - **`.extrn`** ***`symbol bytecount`***

Layout of program memory



Not to Scale!

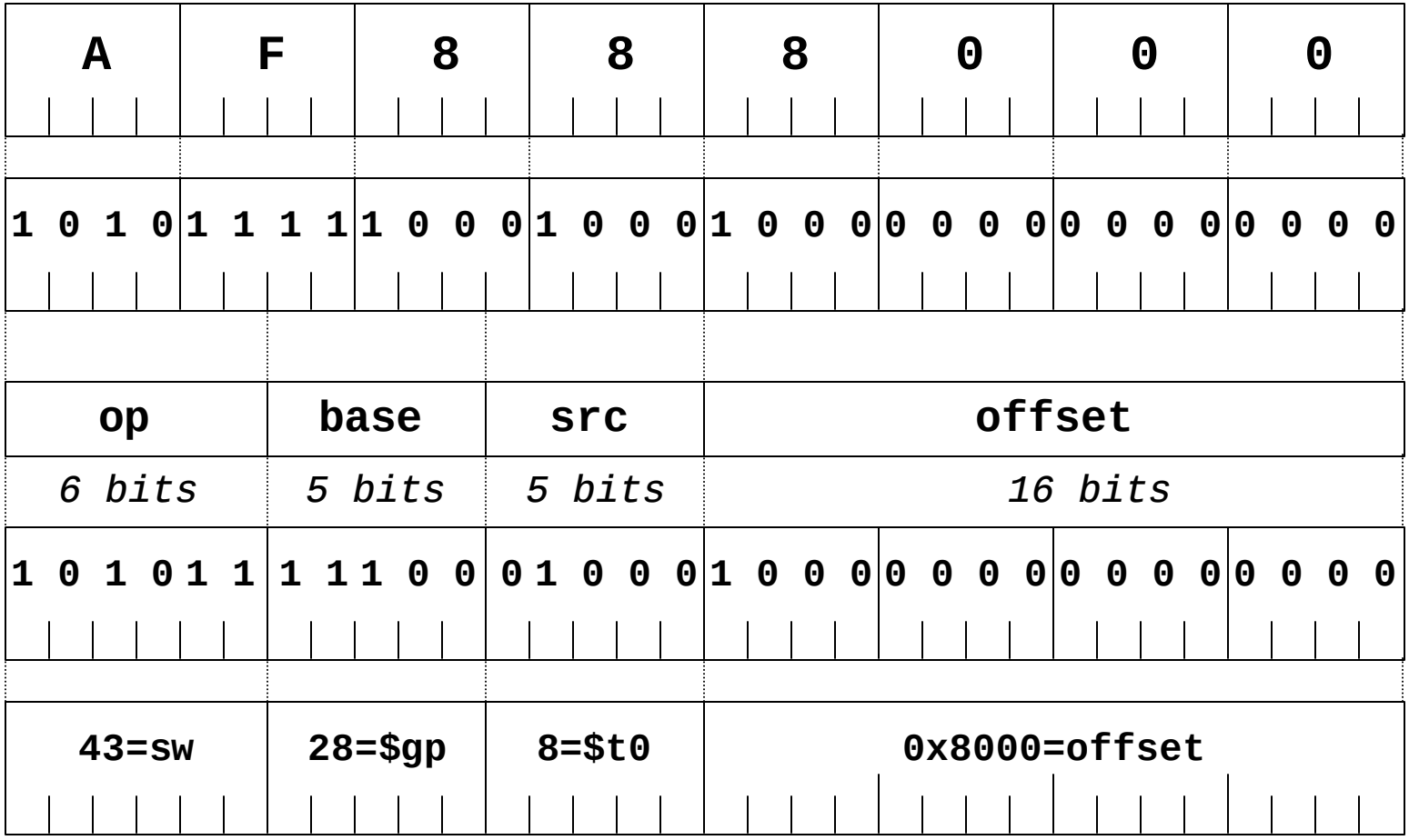
Using the global pointer - gp.s

```
   .extern common 4      # global area symbol
   .data
local:                   # non-global symbol
   .word   0xAAAA        # data value

   .text
main:
   lw        $t0,local   # load word
   sw        $t0,common  # store word
   j         $ra         # return
```

A reference through $gp

```
                # lw   $t0,local
0x3c011001        lui  $1, 4097
0x8c280000        lw   $8, 0($1)

                # sw   $t0,common
0xaf888000        sw   $8, -32768($28)
```

0xAF888000 <=> sw $t0,-32768($gp)



Pearls of wisdom from Sweetman

- These calling conventions can look very complex
  - but partly that's just appalling documentation
  - and the inclusion of debugging conventions
- Most functions that you may write in assembler for tuning reasons will be leaf functions
  - the declaration of such a function is very simple