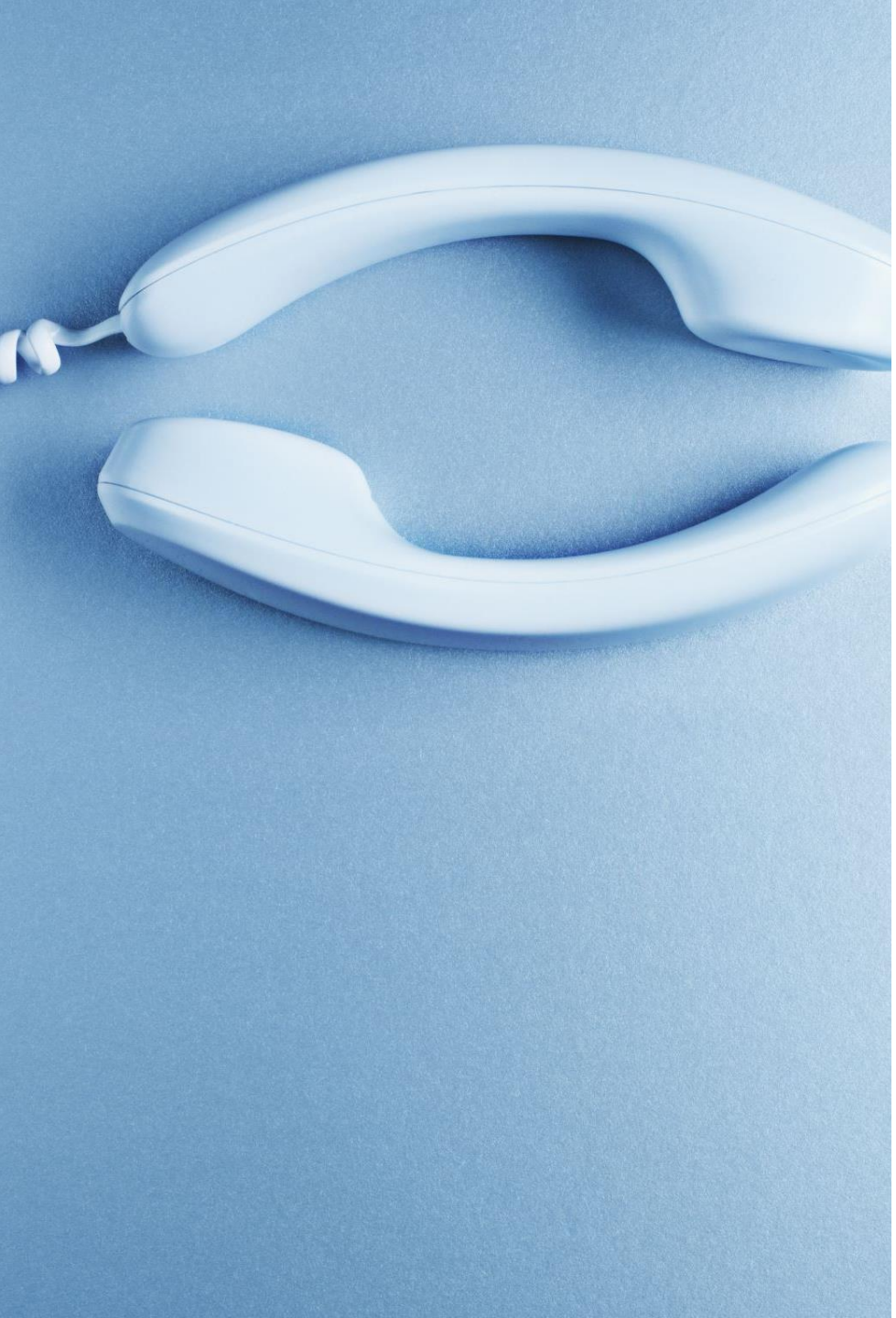
CS230: Digital Logic Design and Computer Architecture

Lecture 8: MIPSInstructions contd…

https://www.cse.iitb.ac.in/~biswa/courses/CS230/main.html



Phones on Silence

If you are busy,

Then you may not consider making others busy ☺

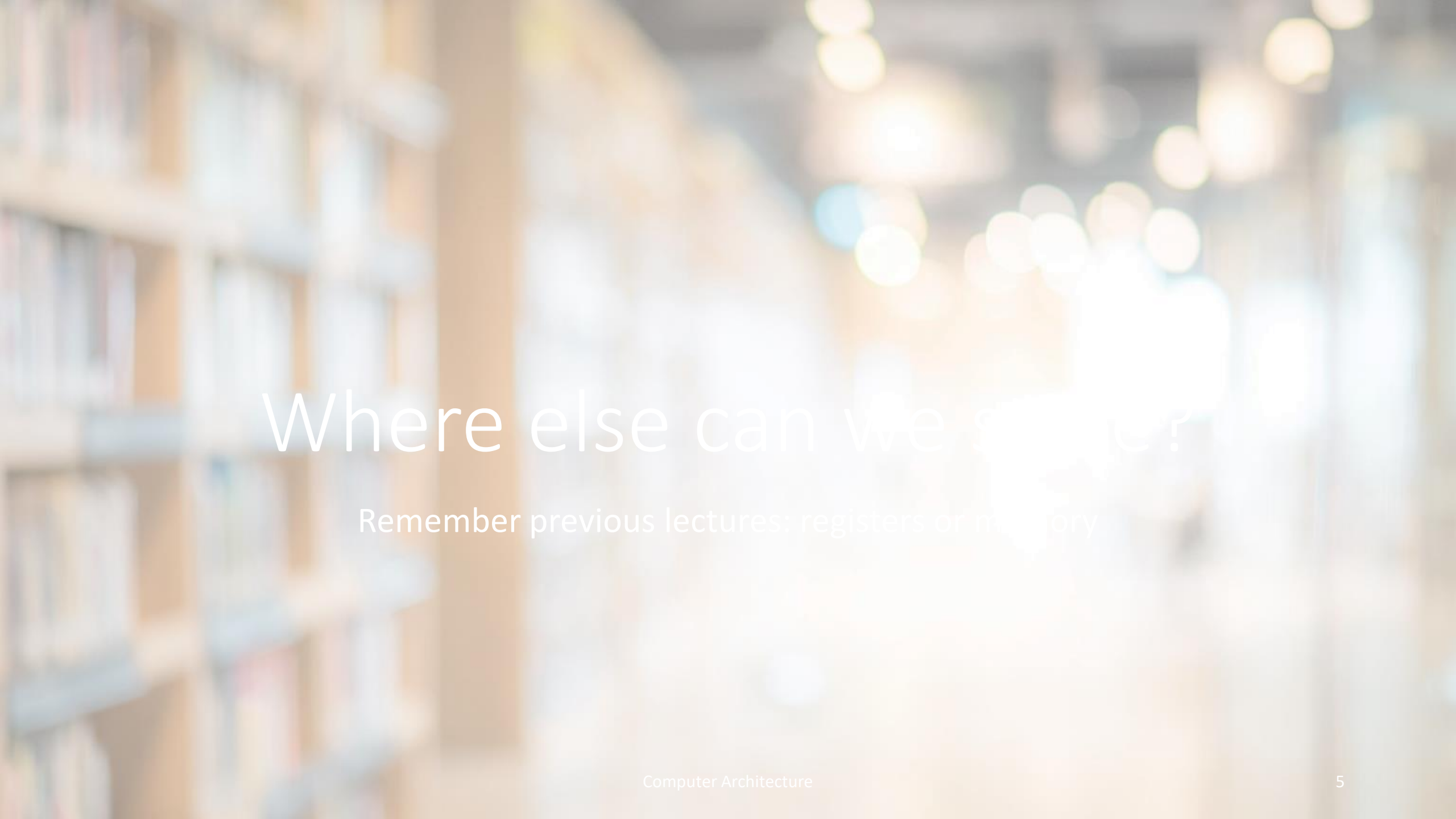
Do not forget 32 MIPS registers only Register spilling ☹

Quick recap

Register spilling, 32 MIPS registers, nested functions,

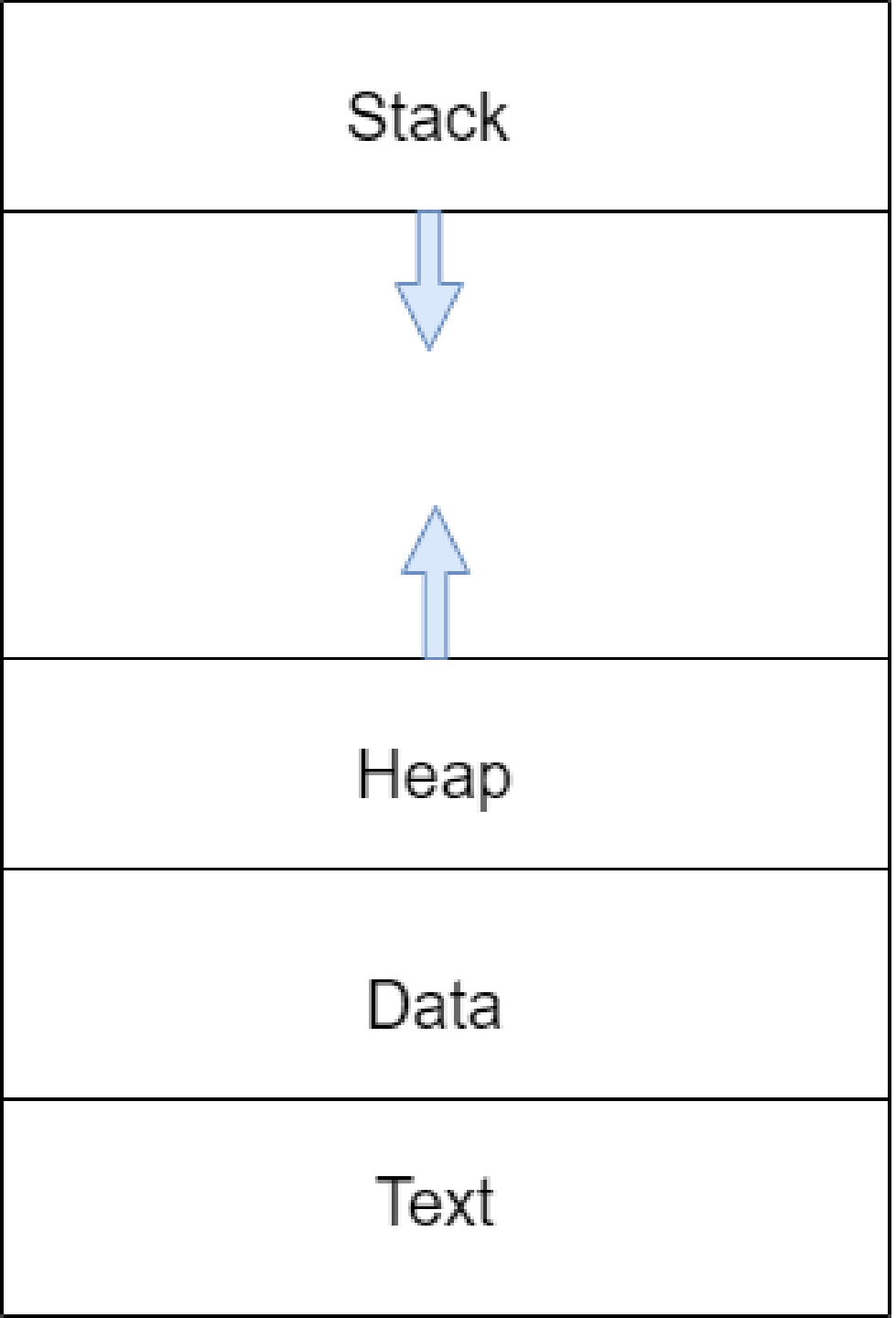
oh no!

Spilled registers: Where else can we store?



Where else can we store?

Remember previous lectures: registers or memory

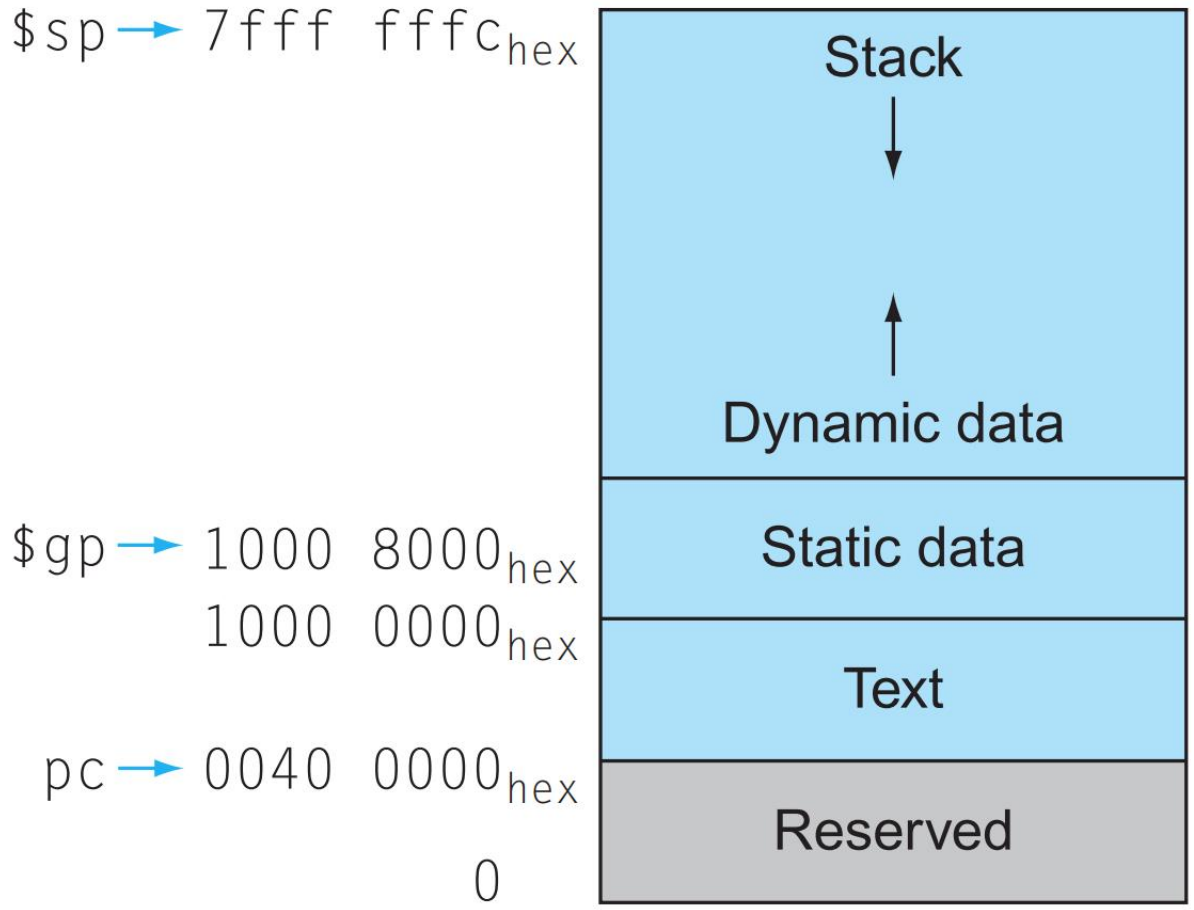


The loaded program

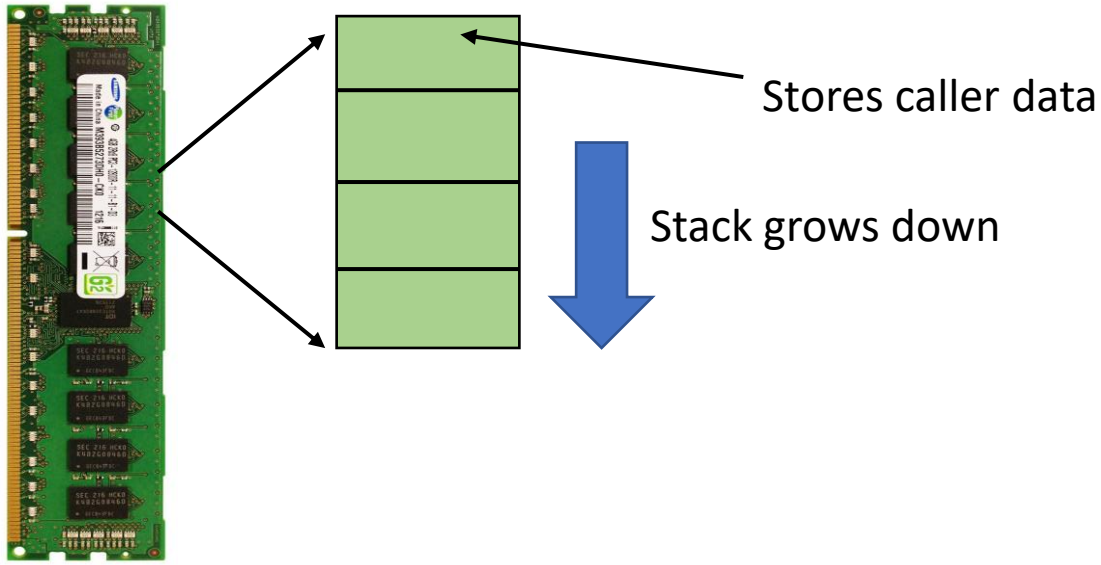
System program that loads the executable into the memory.

Every executable has a text, heap/stack data segments

Stack in MIPS (Grows downwards, High to Low)

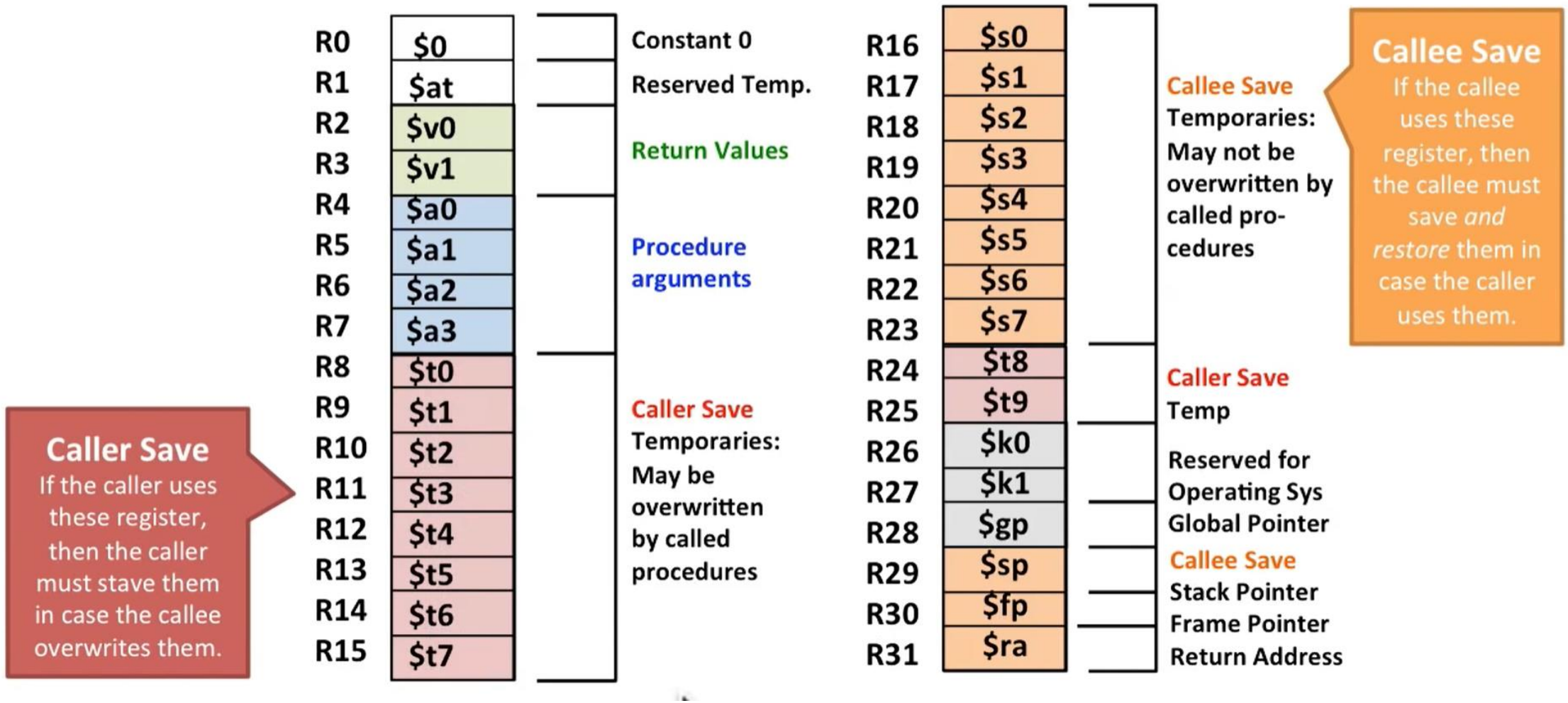


MIPS way of handling it:
The Stack (part of DRAM, for each function call)

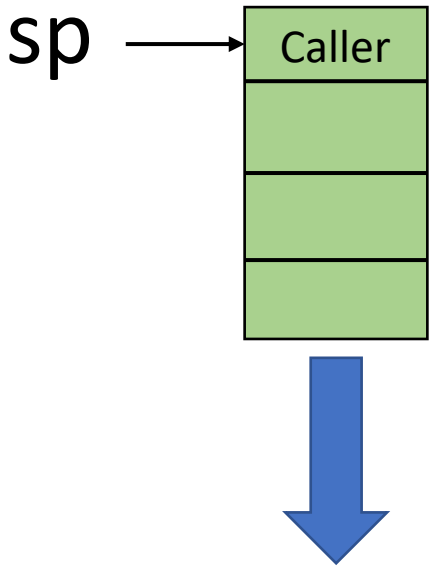


$sp (stack pointer) points to the address where stack ends

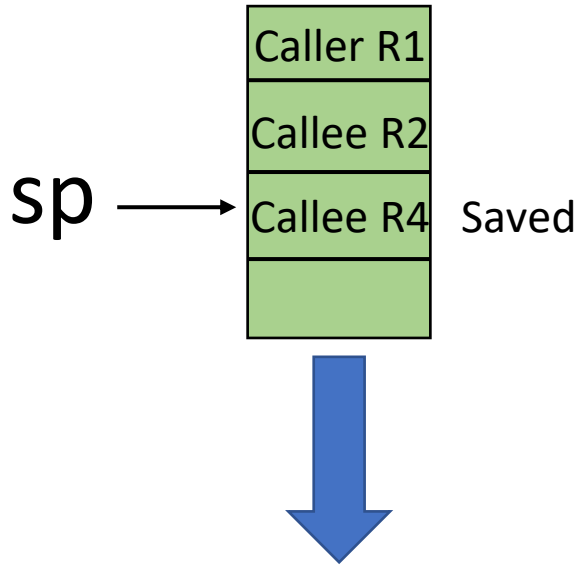
One per function, private memory area, else the same problem ☹


R0 $0
R1 $at
R2 $v0
R3 $v1
R4 $a0
R5 $a1
R6 $a2
R7 $a3
R8 $t0
R9 $t1
R10 $t2
R11 $t3
R12 $t4
R13 $t5
R14 $t6
R15 $t7
Constant 0
Reserved Temp.
Return Values
Procedure arguments
Caller Save
Temporaries:
May be overwritten by called procedures
Caller Save
If the caller uses these register, then the caller must stave them in case the callee overwrites them.
R16 $s0
R17 $s1
R18 $s2
R19 $s3
R20 $s4
R21 $s5
R22 $s6
R23 $s7
R24 $t8
R25 $t9
R26 $k0
R27 $k1
R28 $gp
R29 $sp
R30 $fp
R31 $ra
Callee Save
Temporaries:
May not be overwritten by called pro-cedures
Caller Save
Temp
Reserved for Operating Sys
Global Pointer
Callee Save
Stack Pointer
Frame Pointer
Return Address
Callee Save
If the callee uses these register, then the callee must save and restore them in case the caller uses them.

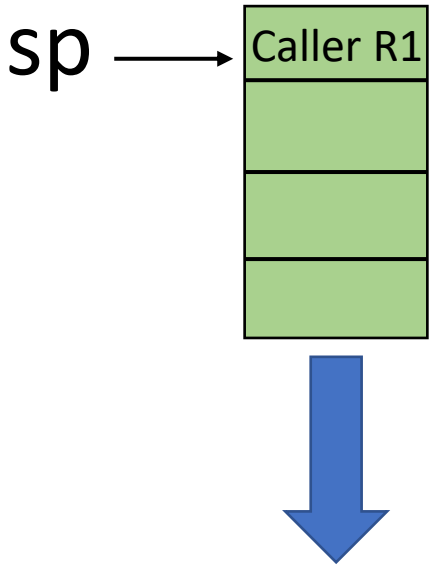
MIPS way of handling it: Before function call



MIPS way of handling it: Function call is ON



MIPS way of handling it: After the function call



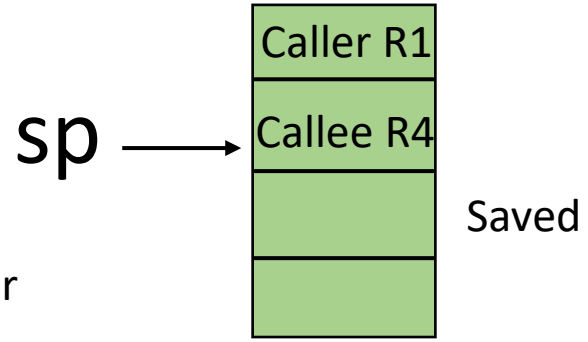
How to save and restore?

Save:

addi $sp, $sp, -4

sw R4, ($sp)

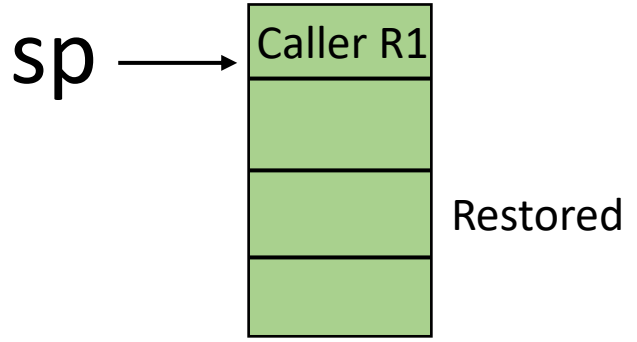
32 bit registers, 4 bytes, one word, remember



Restore:

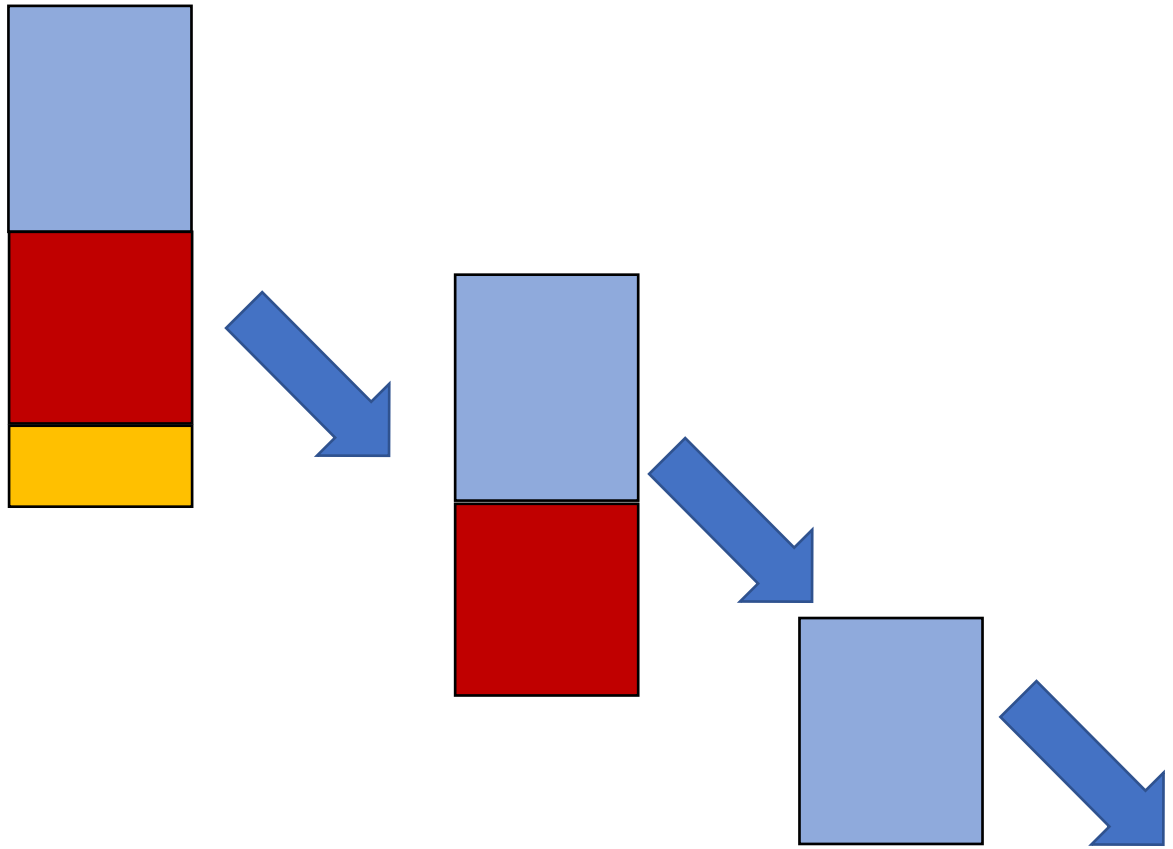
lw R4, ($sp)

addi $sp, $sp, 4



Nested Functions (Remember main() is a function too ☺)

```
CS230 // jal cs230
{
    CS330 // jal cs330
     {
        CS430 // jal cs430
         {
         } //jr
      } //jr
} // jr
```



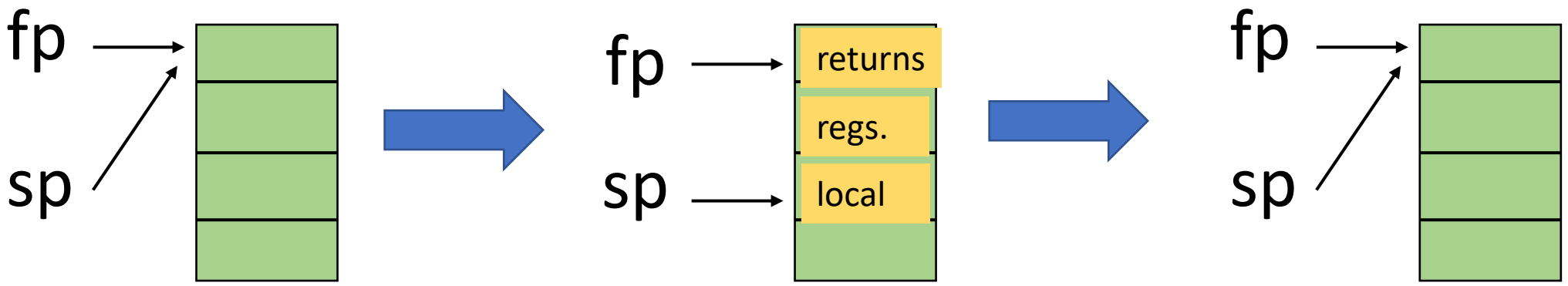
The final one: Frame pointer

Stack also stores local variables and data structures (local arrays and structures) for a function along with the return address(es).

Frame pointer will get incremented and decremented based on the local arguments used.

The final one: Frame pointer

Frame pointer: Points to local variables and saved registers. Points to the highest address in the procedure frame. Stays there throughout the procedure. Stack pointer, moves around.



Awesomeness: You can access any using fp/sp and an offset

Try This Out! Discuss on Piazza

Page no A-27 to A-29 P&H

Recursive function fact(n)

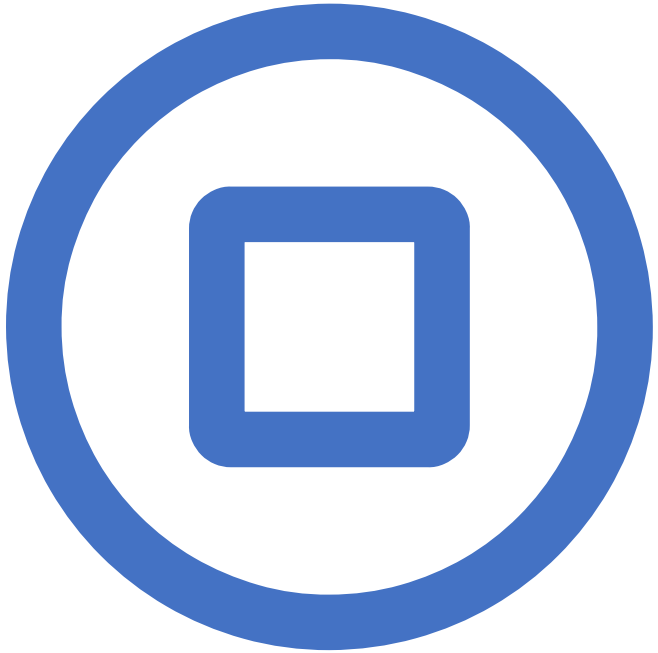
Look for sp, fp, ra, jal, and jr

For the Curious Ones (Beyond CS230)

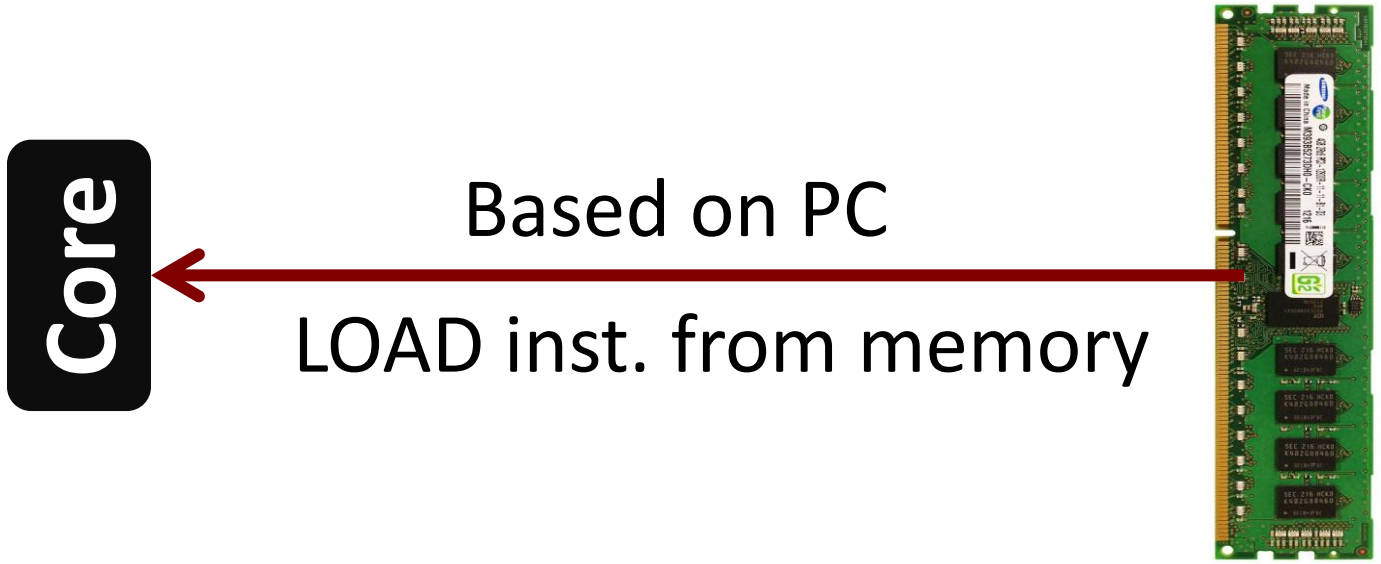
Stack buffer overflow - 101:
https://en.wikipedia.org/wiki/Stack_buffer_overflow

Nilabh ☺

How to know what is what?



Why instruction decoding?

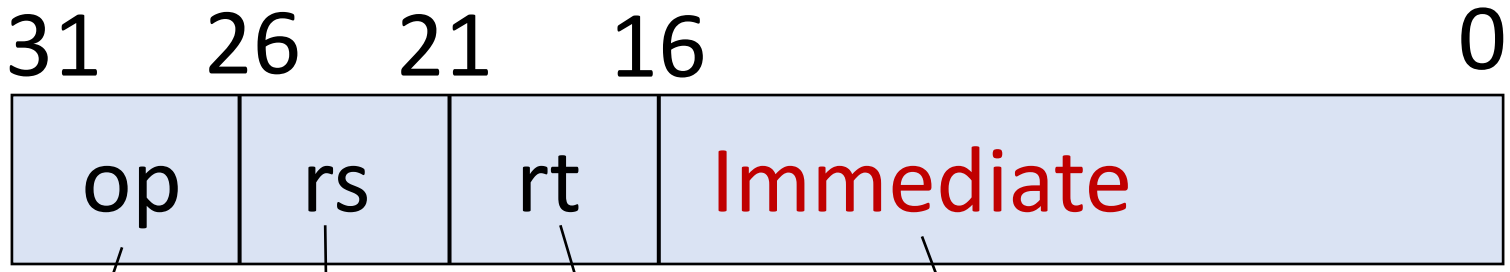


Instruction received then what?

Remember instructions are of 32-bit size (in MIPS), so PC+4

How will the processor know what to infer from these 32 bits?
Simple: Have a decoder ☺

Instruction Decoding



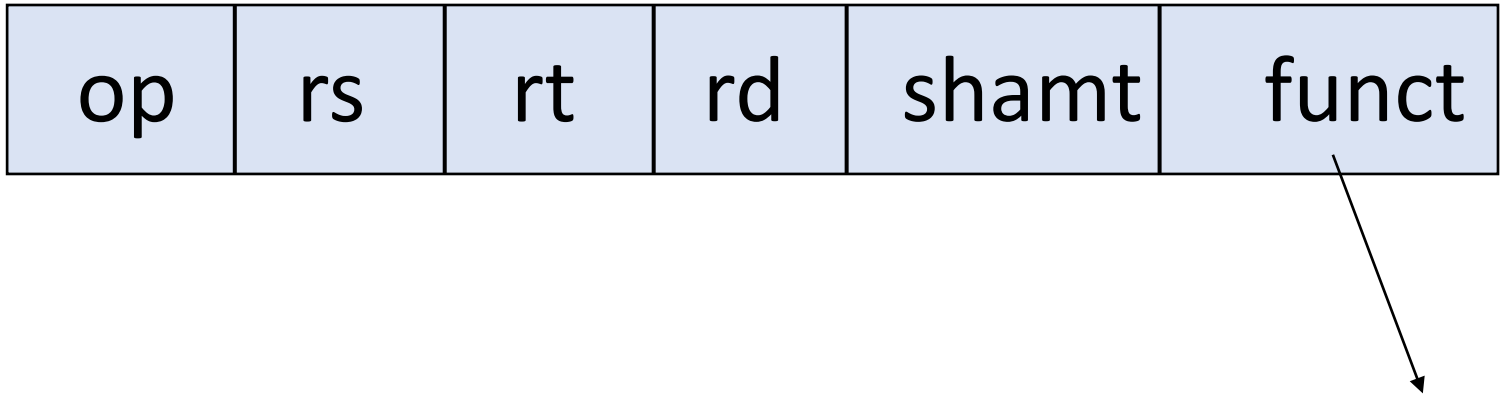
6 bits, maximum
64 operations
add
sub
mul
lw
bne …….

5 bits, 32 registers
(R0 to R31)

5 bits, 32 registers (R0 to R31)

16 bit constant

10K Feet View of MIPS encoding



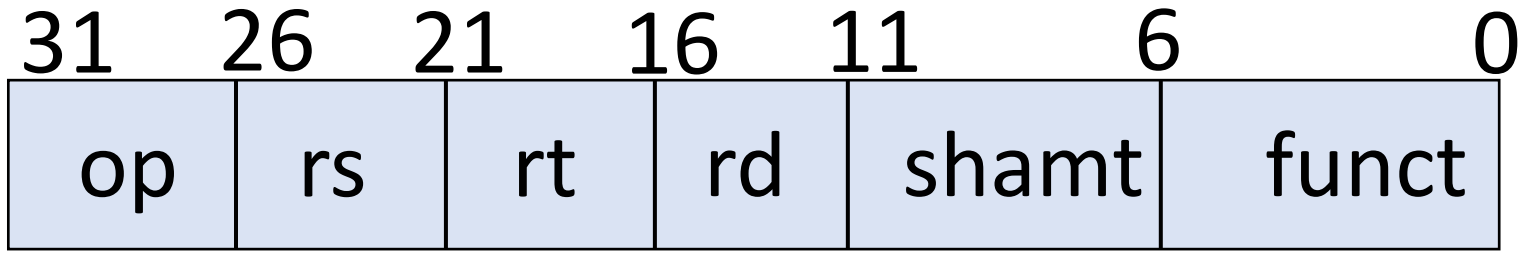
Why this field?
Wastage of space ☹

Good design demands good compromises

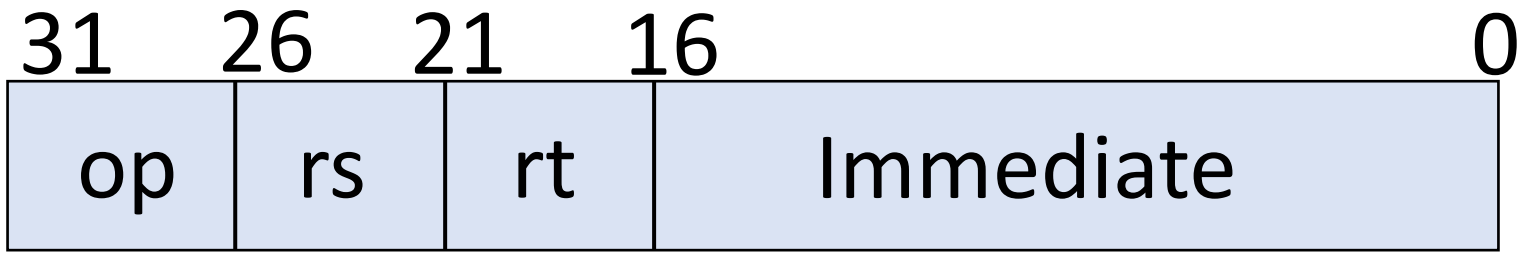
Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32	n.a.
sub	R	0	reg	reg	reg	0	34	n.a.
addi	I	8	reg	reg	n.a.	n.a.	n.a.	constant
lw	I	35	reg	reg	n.a.	n.a.	n.a.	address
sw	I	43	reg	reg	n.a.	n.a.	n.a.	address

tells how to treat the last set of fields:
three fields or one field, still why funct ☹

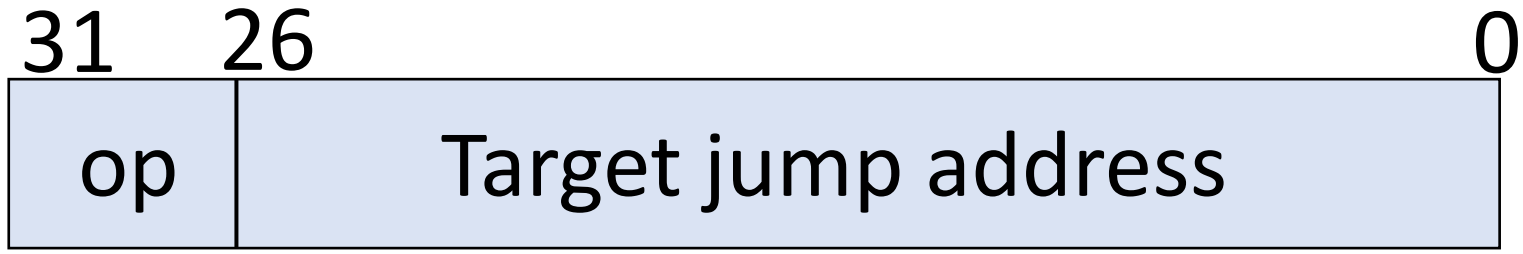
Let’s have a look



R-type

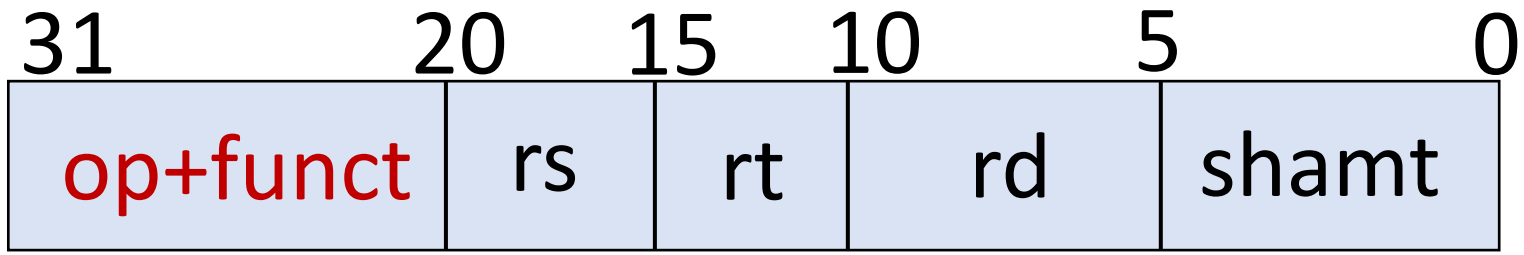


I-type

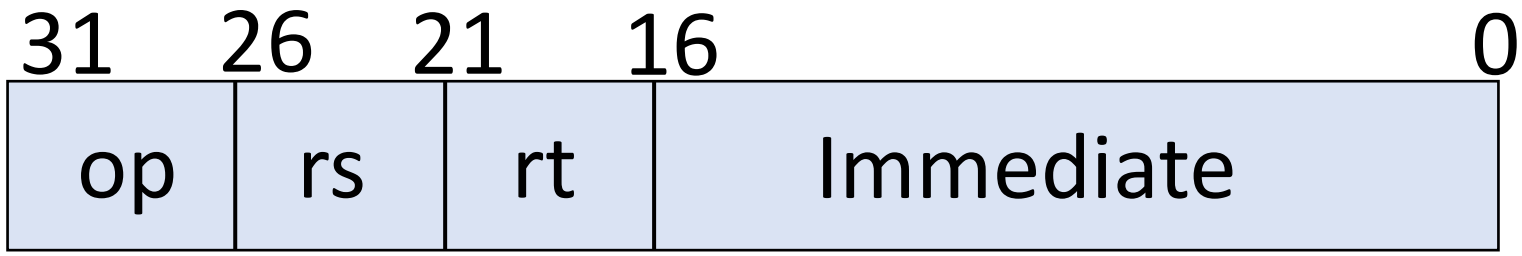


J-type

Why not?



R-type



I-type

J-type

What is a good compromise?

- Fixed length instructions ☺
  32-bit irrespective of ops
- Fields are at the *same* or almost same location
- All formats look *similar*


તમારો દિવસ શુભ રહે