Why Frame Pointers?

C Code

```
f(1);
...;
g(2)
```

Assembly

```
addiu $29,$29,-16  # space for 4 args
li $4,1            # argument setup for f
jal f              # call f
addiu $29,$29,16   # pop stack
...                # code for ...
addiu $29,$29,-16  # space for 4 args
li $4,2            # argument setup for g
jal g              # call g
addiu $29,$29,16   # pop stack
```

Why Frame Pointers?

Assembly

```
addiu $29,$29,-16   # space for 4 args
li $4,1             # argument setup for f
jal f               # call f
# addiu $29,$29,16  no more pop stack
...                 # code for ...
addiu $29,$29,-16   # space for 4 args
li $4,2             # argument setup for g
jal g               # call g
# addiu $29,$29,16  no more pop stack
```

Save initial `$sp` in the frame pointer

Optimization

Calculate max stack adjustment, change $sp once

Assembly

```
entry:  addiu $29,$29,-24 # allocate space once
        sw $31,20($29)    # save return address
        ...
        li $4,1           # argument setup for f
        jal f             # call f
        ...               # code for ...
        li $4,2           # argument setup for g
        jal g             # call g
```

Stack pointer must be double-word aligned.

More Argument Passing

What if I want to pass a half-word (`short`) or a byte (`char`)?

- Use a full word (least significant bits)

If the argument is stored on the stack (argument 5 and greater):

- Memory is big-endian
- Therefore uses *higher addresses*

Example: reading fifth argument of type char:

```
    lbu $8,19($29)
```

More Stacks...

Stack frame:

- Region of stack allocated by function

If a function doesn't call another one:

- Called a "leaf function"
- Doesn't save return address on stack
- Stack only used for local variables

Leaf Function With Local Storage

C Code

```
void storage (int i)
{
  int x[16];
  ...
  return;
}
```

Large array, need to save in memory.

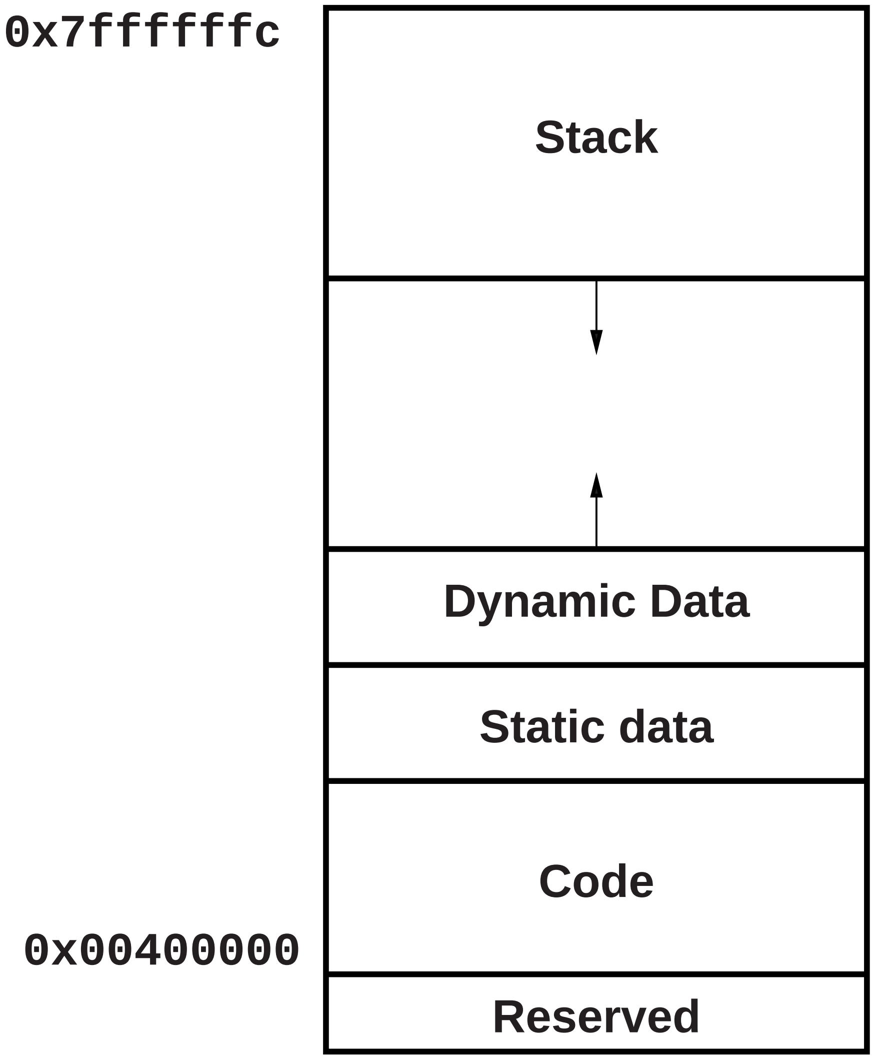
Leaf Function With Local Storage

```
          .ent storage      # assembler directives
          .globl storage    # global function
storage:  addiu $29,$29,-64 # allocate space for x
          ...               # x begins at address
                            # $29
          addiu $29,$29,64  # pop stack
          jr $31            # return
          .end storage
```

Stacks: Summary

- Stack pointer points to top of stack
- Stack grows downward in memory
- Stack pointer is double-word aligned
- Stack frame is at least 24 bytes
- Registers 16–23 are saved by callee
- Argument passing: register 4–7, space on stack
- Return values in register 2–3
- Local variables, saved registers, return address saved on stack

Memory Layout



Simulator Init SP:

`0x7fffae50`