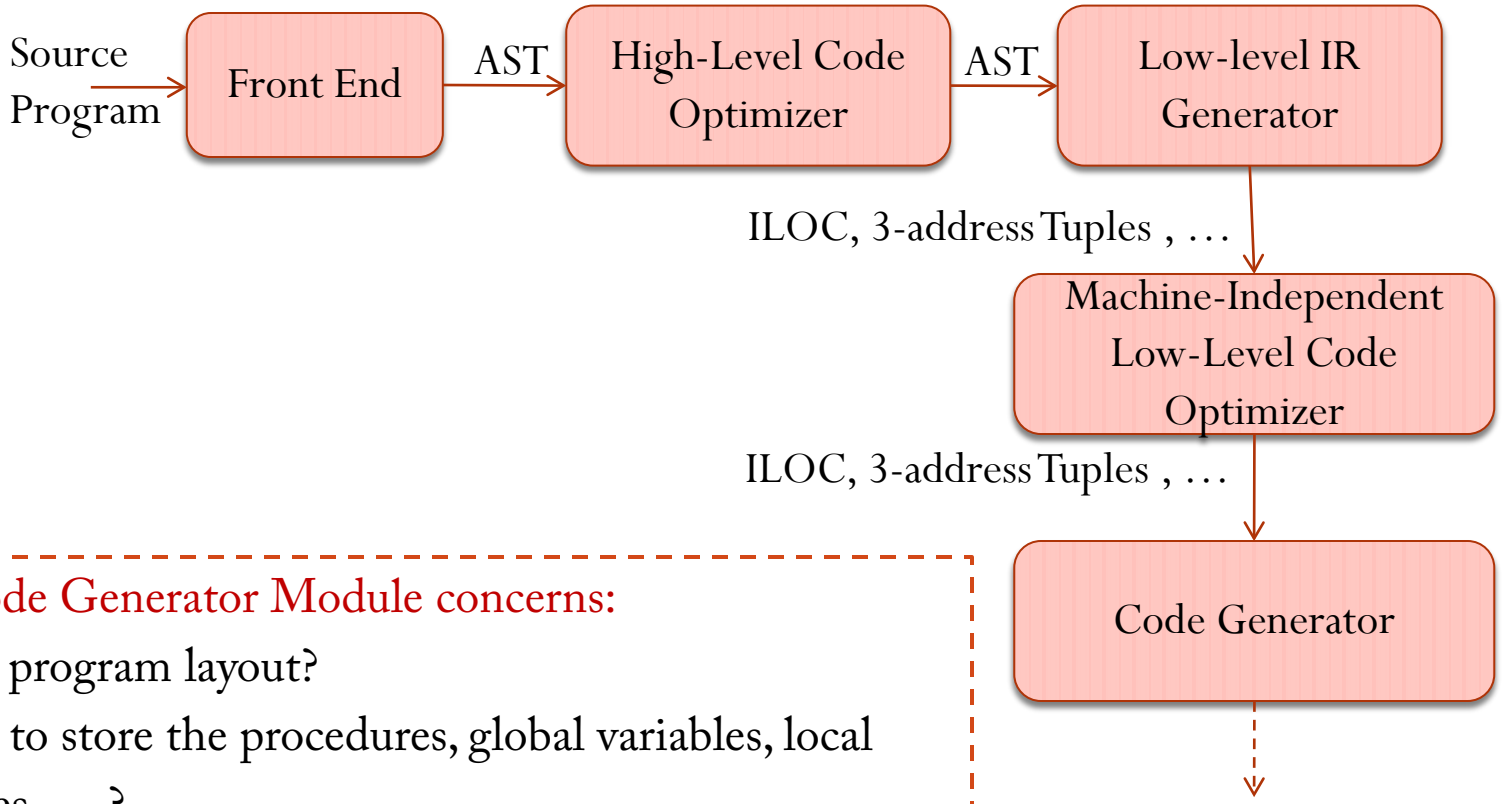
Compilers

Topic: Run Time Environment for C-like Languages

Monsoon 2011, IIIT-H, Suresh Purini

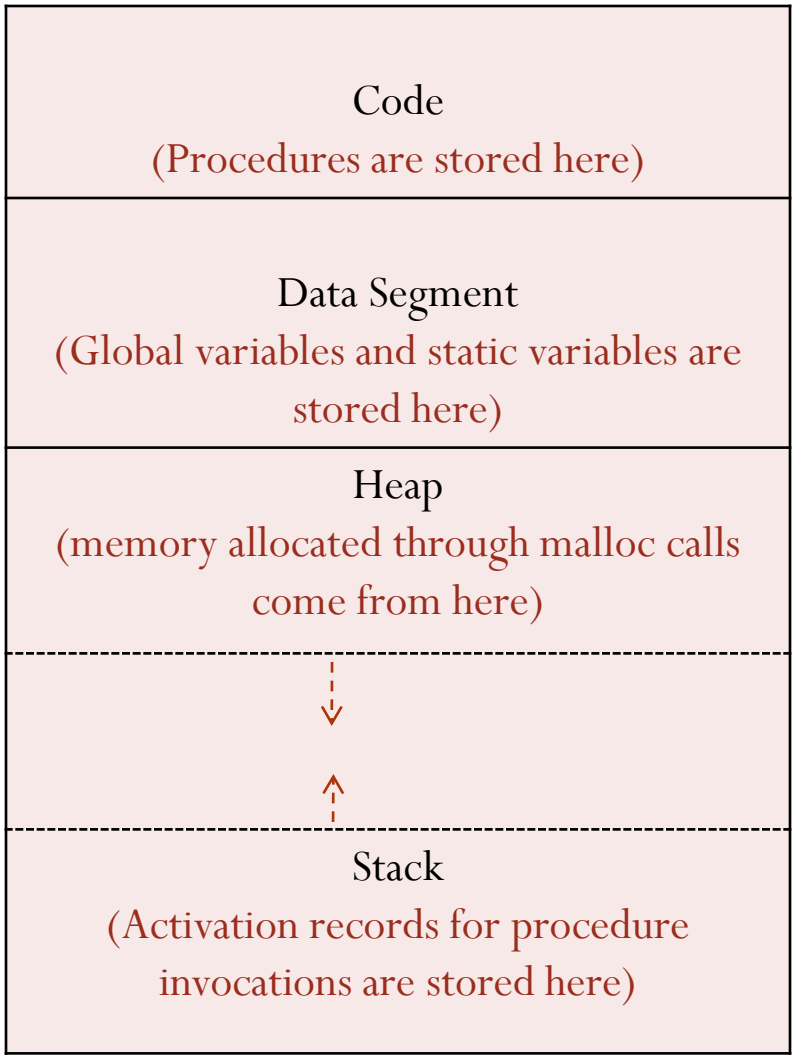
High-level Compiler Architecture



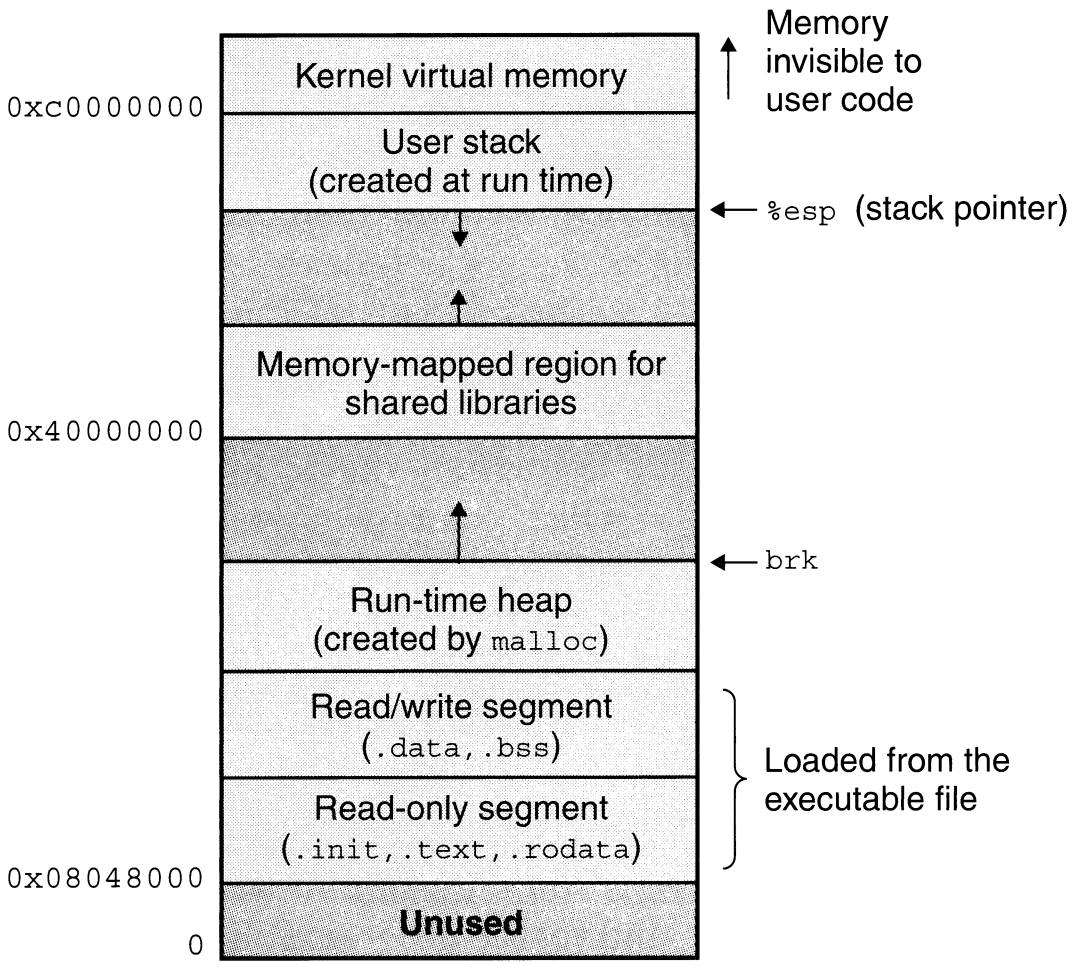
Some of the Code Generator Module concerns:

1. What is the program layout?
    1. Where to store the procedures, global variables, local variables, ... ?
2. How to handle procedure calls? Hey, by now other high-level language constructs like if, for, while etc. are flattened out!
3. More questions need to be answered for Object Oriented Languages like C++, Java.

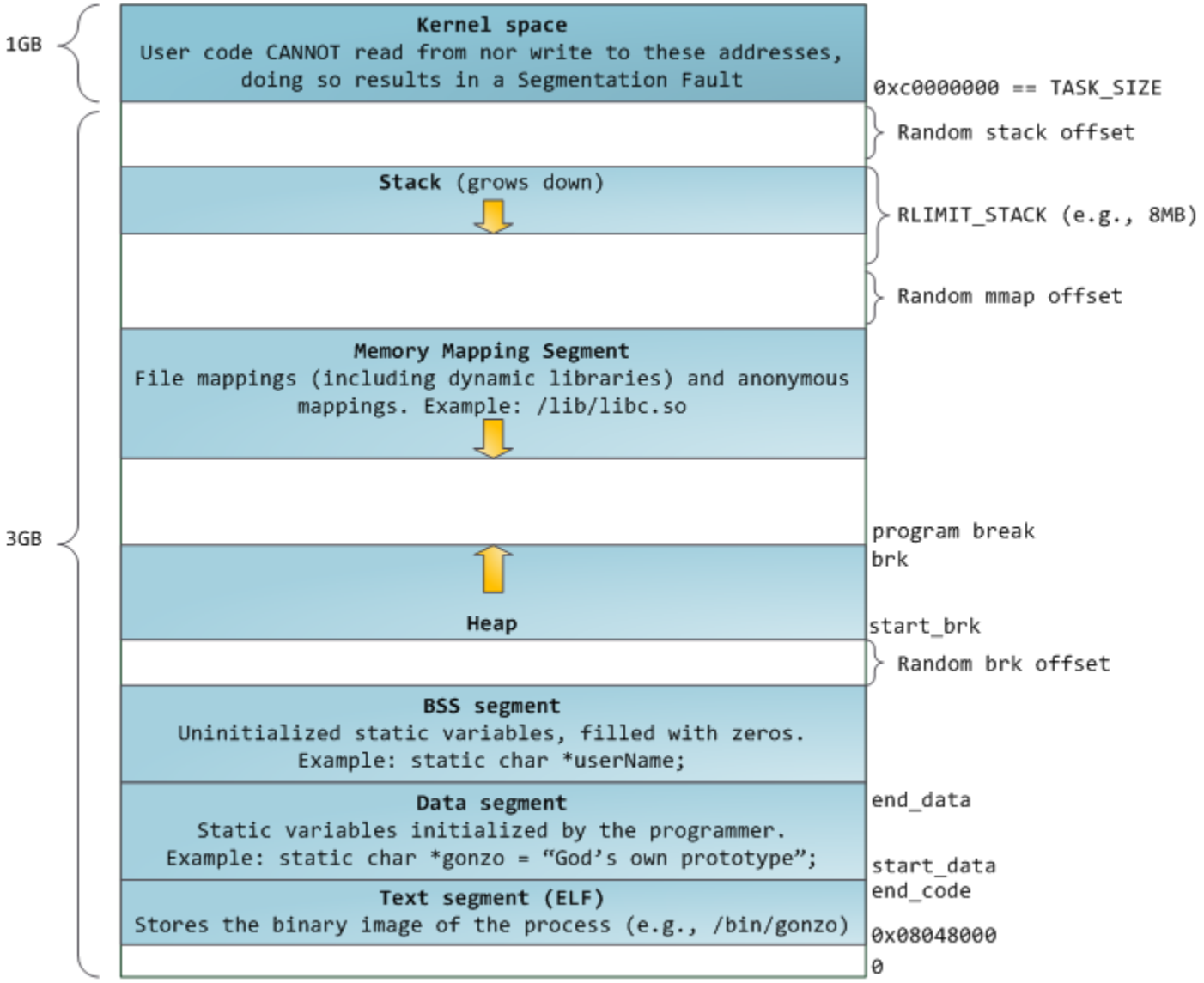
Storage Lay-out of a Program



Virtual Address Space Layout of a Linux Process

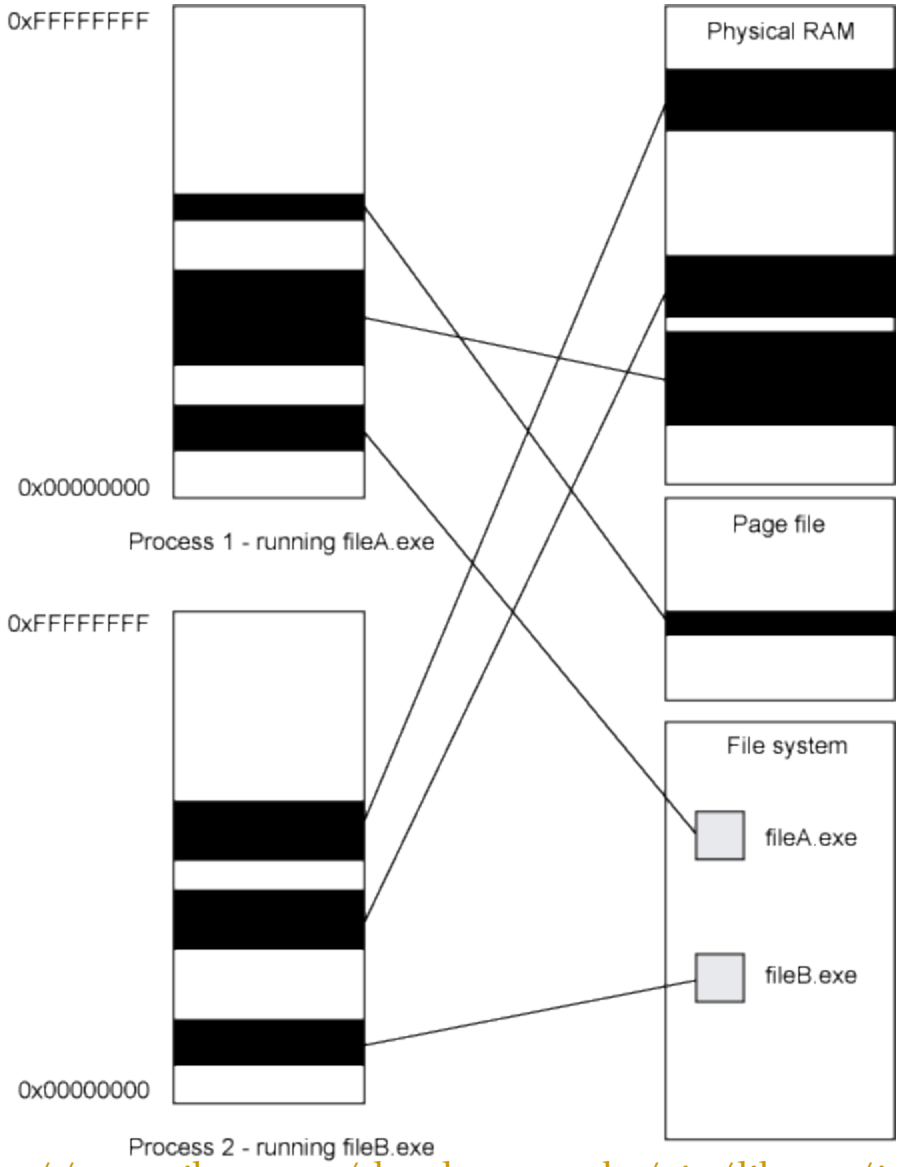


Virtual Address Space Layout of a Linux Process



Taken from: http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory

Virtual Address Space versus Physical Address Space



Taken From: http://www.ibm.com/developerworks/aix/library/j-nativememory-aix/index.html

Global Variables versus Local Variables

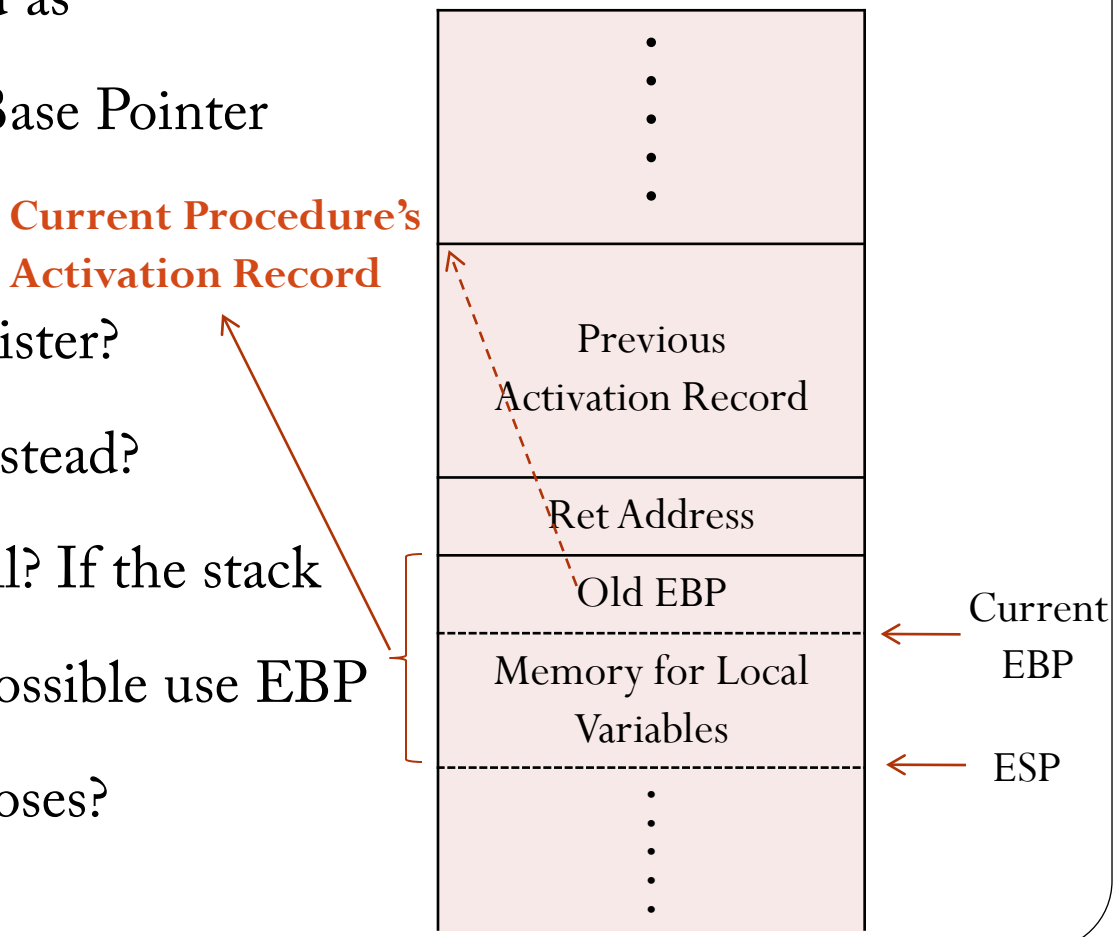
- numarray is allocated memory in .data section.
  - You can notice this using gdb or objdump
- How about the variables i, sum?
  - Do objdump –D –S a.out

```
int numarray[8] = {1, 2, 3, 4, 5, 6, 7, 8};


main()
{
  int i, sum = 0;
  for(i = 0; i <= 7; ++i)
      sum+=numarray[i];
}
```

Stack Frame for a function (Activation)

- Associated with each function invocation or Activation an Activation Record (or frame) for the function will be created
- Local variables are addressed as (negative) offsets from the Base Pointer Register
- Why are we wasting one register? Can't we use ESP register instead?
- What if there is an alloca call? If the stack frame size is fixed we can possible use EBP register for some other purposes?



Local, Static Local and Global Variables

- Why can't we simply allocate memory to all the local variables in the data segement?
  - Handling recursion could be a problem?
- Where memory be allocated to Static Locals?
  - In the data segment. Why? We should preserve the value of a static local across different procedure activations.

Implementing Function/Procedure Abstraction

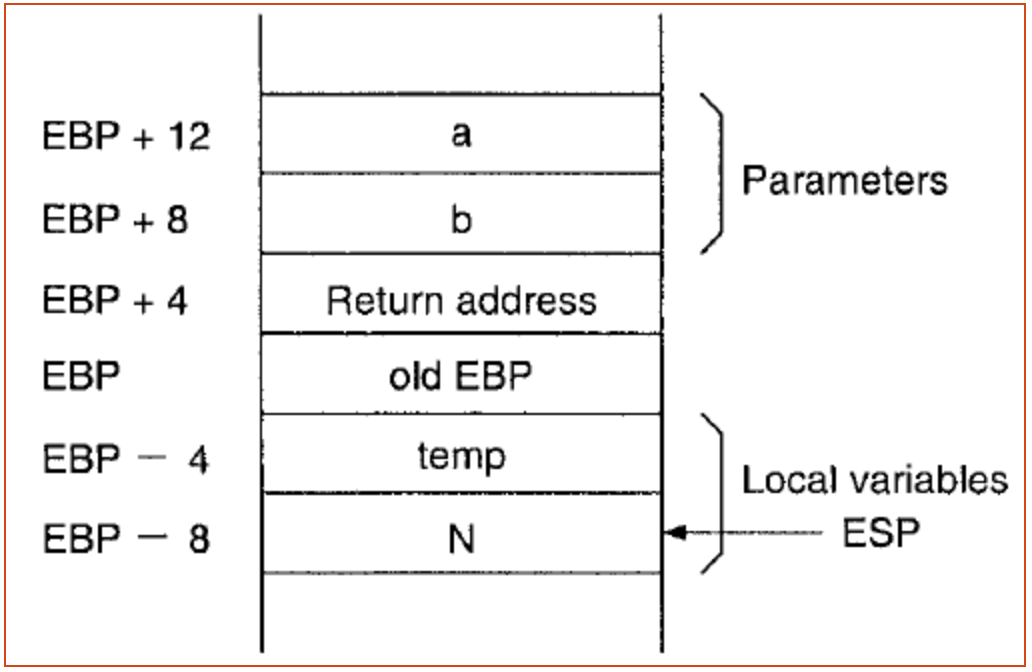
- Caller – Callee Pact
  - Caller – Passes Parameters
  - Callee – Returns a value
- Parameter Passing Mechanisms
  - Call-by-value
  - Call-by-reference
  - Call-by-name
- Our Focus: On implementing Call-by-Value mechanism

Passing Parameters

We can pass parameters either through

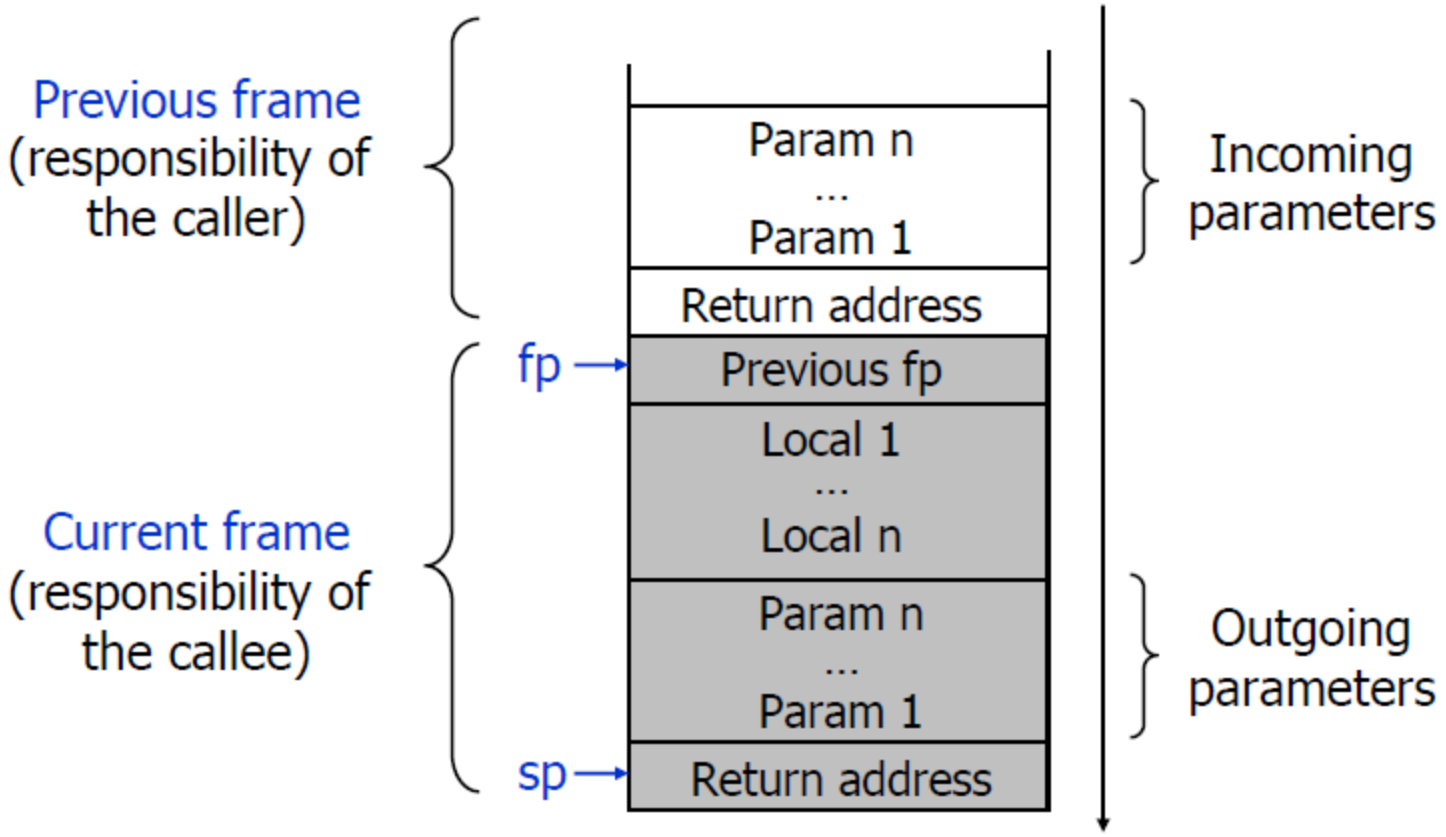
1. Registers
2. or through the Stack
3. or through the both

```
int compute(int a, int b)
{
    int    temp, N;
           . . .
           . . .
}
```



Question: How do we pass the return values?

Anatomy of a Stack Frame



ACK: This slide is taken from Tim Teitelbaum's Compiler's Course at Cornell University.

Saving Registers

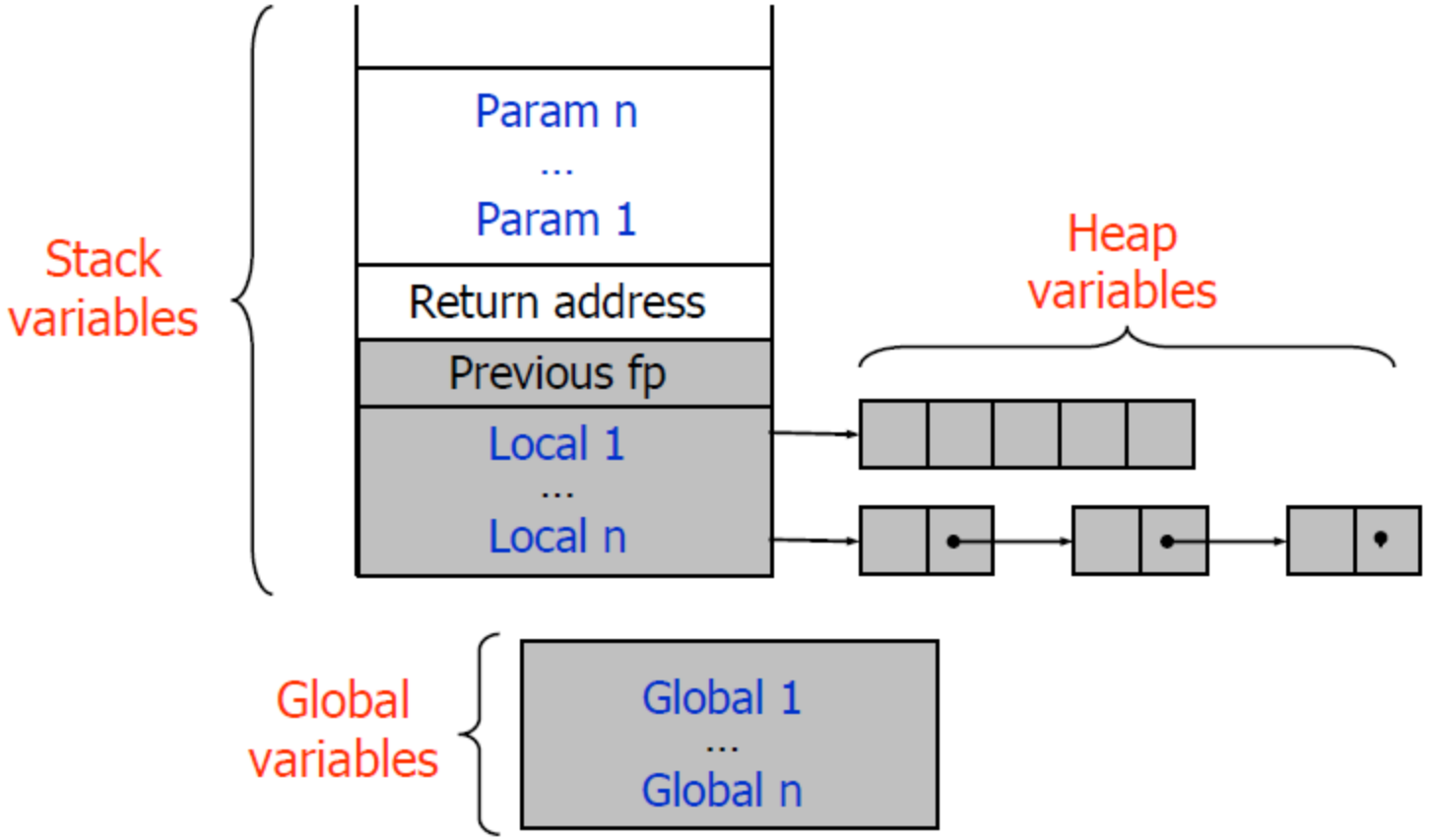
- Problem: When a function foo calls a function fun, foo wants the registers it is using to be preserved across the function call to fun.
- Solution 1: foo saves all the registers to be preserved across the procedure call to fun on the stack and restores them later.
- Solution 2: foo just calls fun. fun saves the registers it is using on the stack and restores them later.
- Solution 3: Divide the Register Bank into 2 sets
  - Caller Saved Registers
  - Callee Saved Registers

Idea: if fun is a leaf procedure it tries to get its job done using the Caller Saved Registers. Why? How about non-leaf functions?

Important Advisory

- You need not worry about procedure calling conventions if you are not interfacing with external C Libraies
- If your Assembly Language Program is interfacing with C Libraries you need to worry about things like..
  - How to pass parameters?
  - Order of passing the parameters
  - How are the return values passed back?
  - What are the Caller Saved Registers?
  - What are the Callee Saved Registers?
  - …..
- You have to consult the Application Binary Interface (ABI) Manual

Big Picture: Memory Layout



ACK: This slide is taken from Tim Teitelbaum's Compiler's Course at Cornell University.

Topics not Discussed

- Run-time Environment for languages with Nested Procedures like Pascal and Algol.
- Run-time Environment for Object-Oriented Languages.
- Run-time Environment for Scheme like languages.
- Garbage Collection