Names, Scopes, Bindings

CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

If a subroutine is passed as a parameter, when are dynamic/static rules apply?

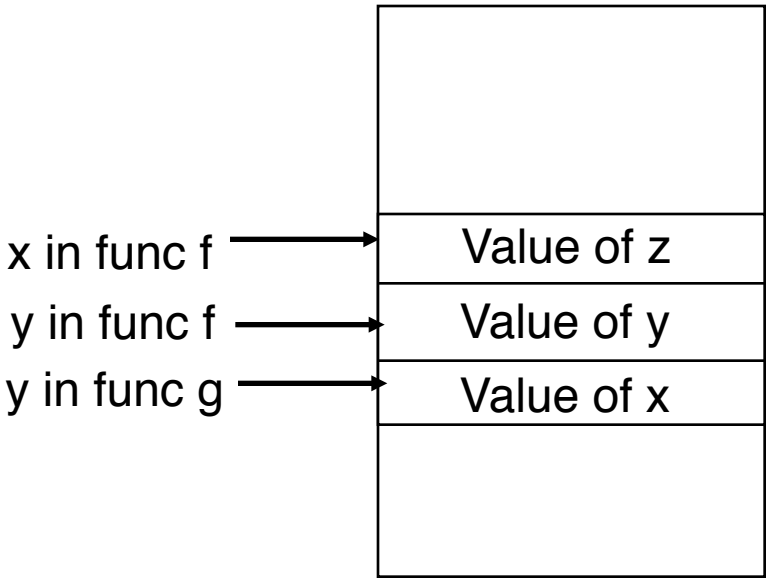
```
function F(int x) {
    function G(fx) {
        int x = 13;
        fx();
    };
    function H() {
        print x;
    };
    G(H);
};
```

Shallow Binding: when the subroutine is called

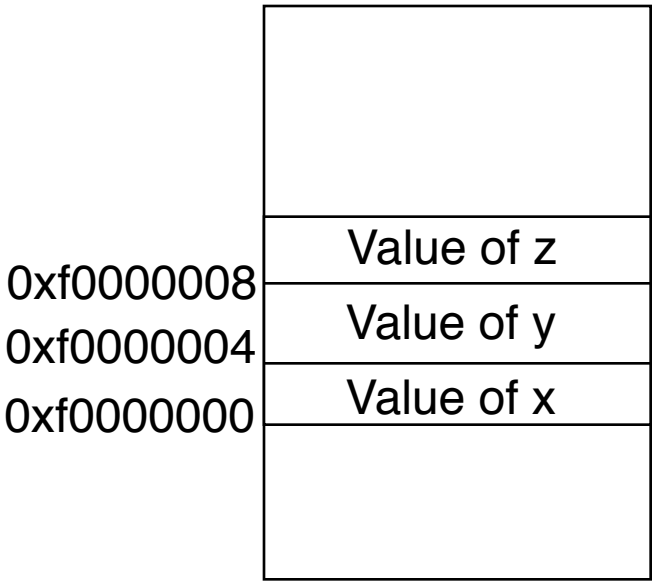
Deep Binding: when reference is created

Establishing Addressability

How does the compiled code locate a variable used in a function at run time?



Memory abstraction in PL



Real Memory

Establishing Addressability

How does the compiled code locate a variable used in a function at run time?

- What is the start address?
  - Start address = base address (of a scope) + offset (in the scope)

Global variables

Local variables

Variables defined in surrounding scopes

Typical Code and Data Layout

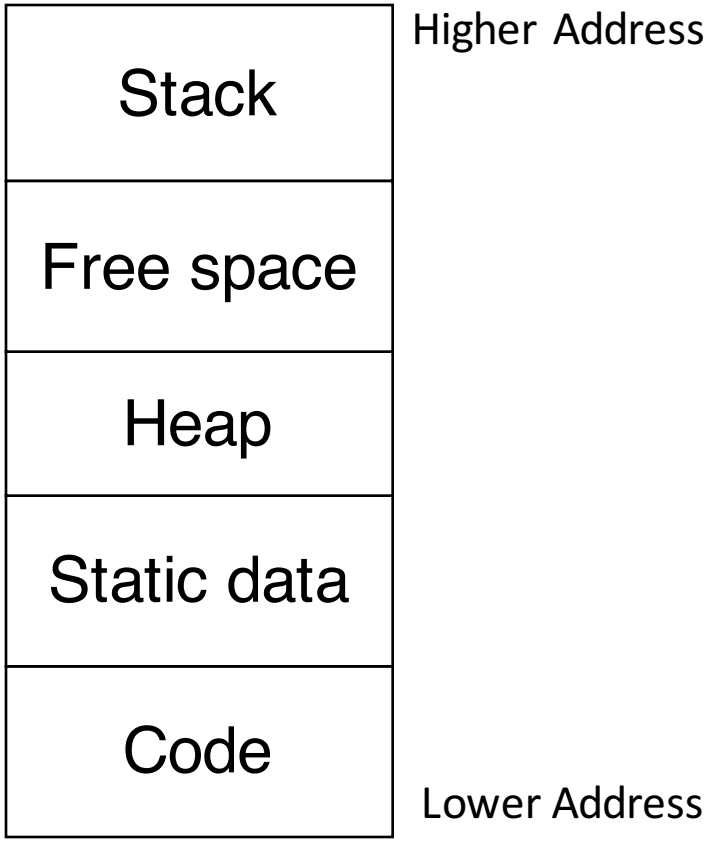
When executed, a program occupies memory

Code section

- Stores source code
- Read-only

Data section

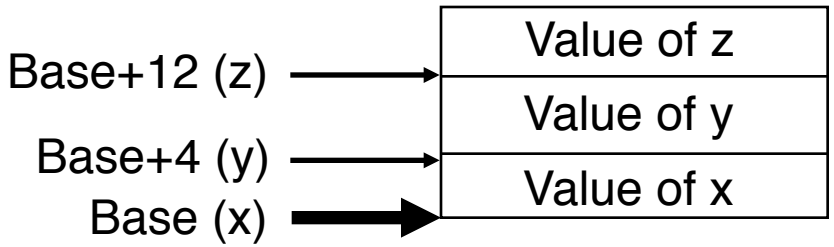
- Stack
- Heap



Global & Static Variables

Base Address: fixed throughout the run

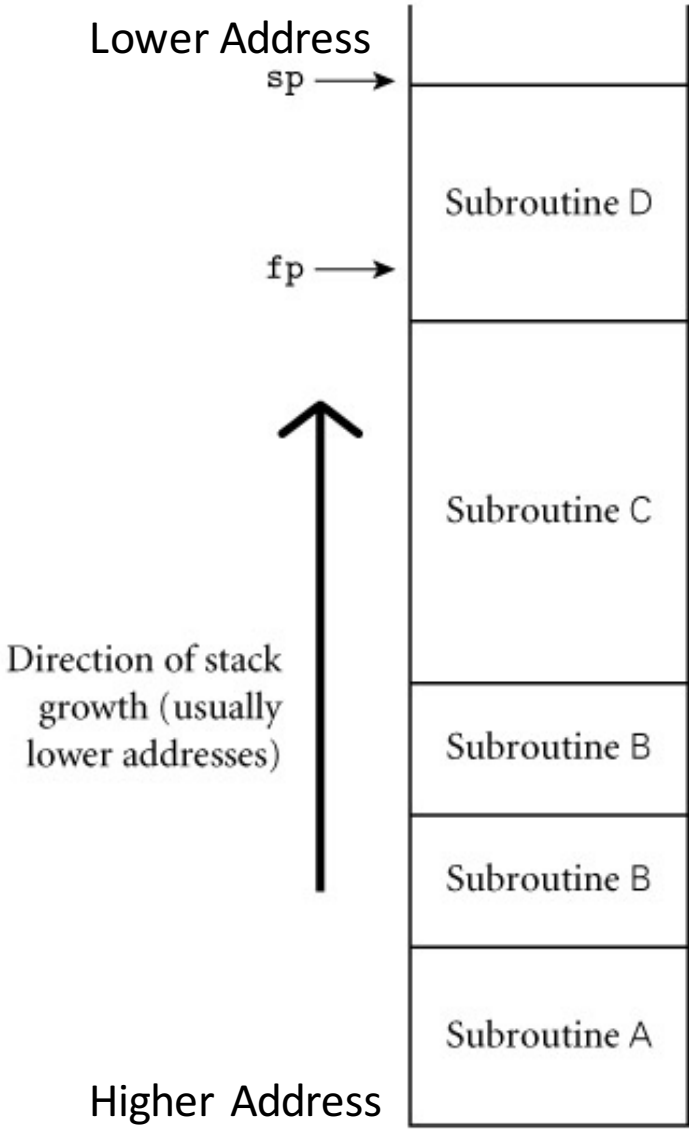
Offset: known at compile time (length of each variable is determined by its type)



Static data

Global variables:
int x;
float y;
int z;

Scope and Stack




```
procedure C
    D; E
procedure B
    if … then B else C
procedure A
    B
-- main program
    A
```

Each instance of a subroutine has a ***frame (activation record)*** at run time

- Compiler generates code that setup frame, call routine, and destroy frame

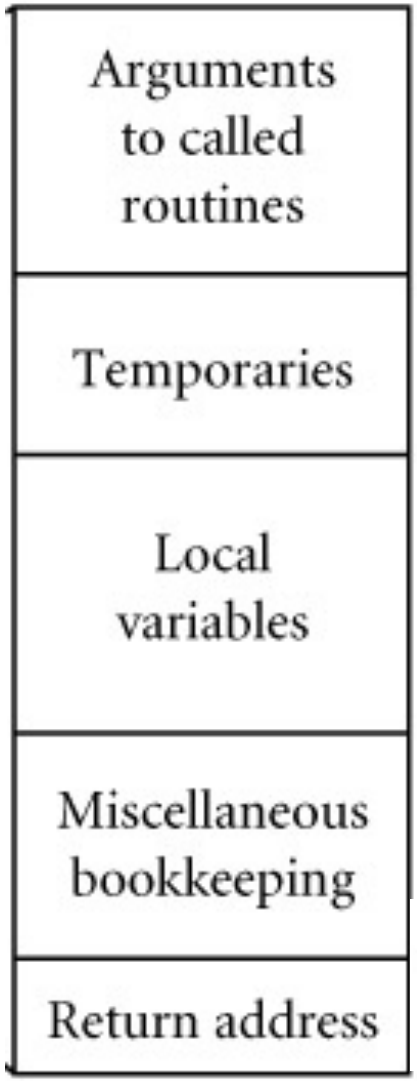
Frame pointer (fp): currently active frame
Stack pointer (sp): free space on stack

Hardware Support

X86 Registers and instructions:

- Stack pointer register: esp
- Frame pointer register: ebp
- Push instructions: push, pusha, etc.
- Pop instructions: pop, popa, etc
- Call instruction: call
- Return instruction: ret

Typical Frame Layout

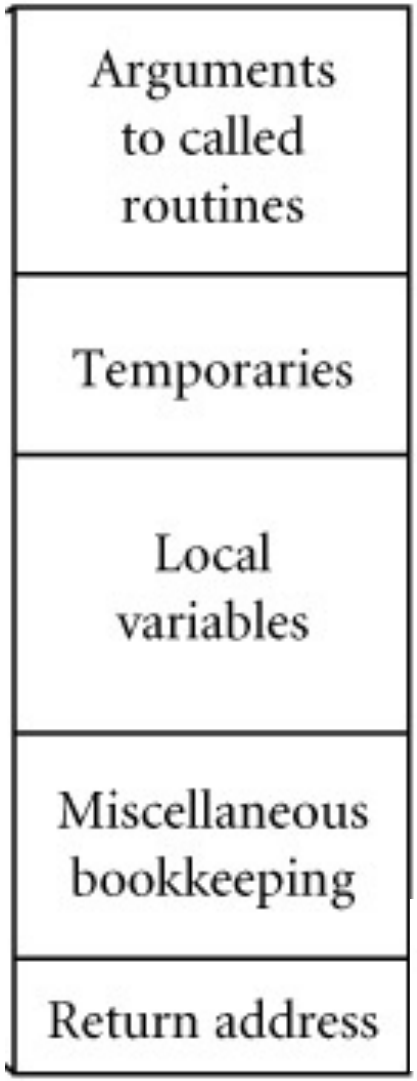


Temporaries.: register spill area, language-specific exception-handling context, and so on (not covered)

Bookkeeping info.: a reference to the stack frame of the caller (also called the ***dynamic link***) and so on

← `fp` (when the frame is active)

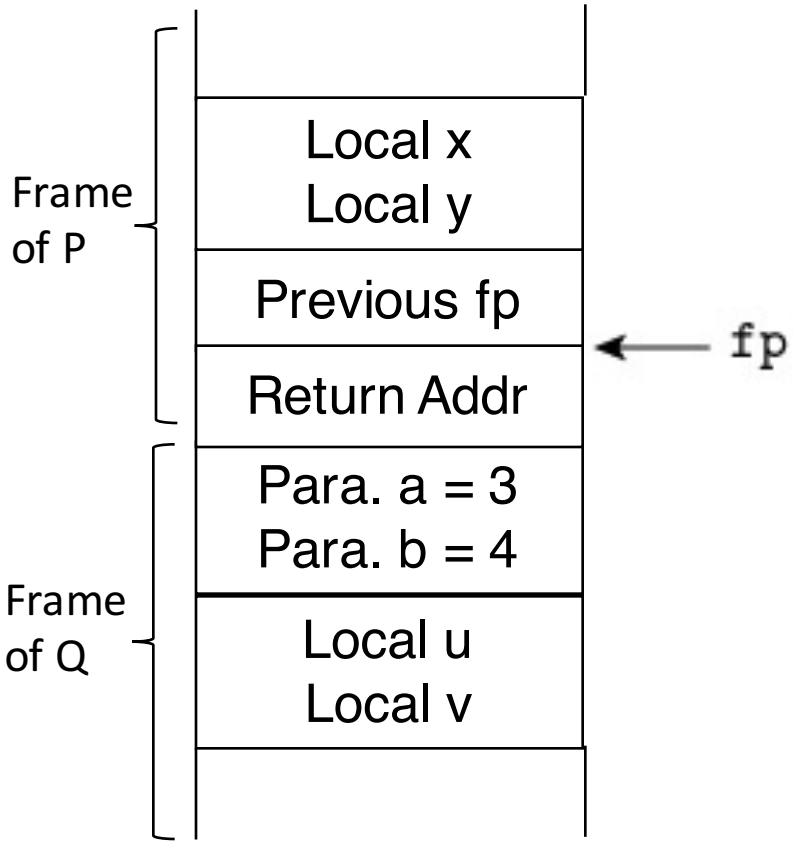
Local Variables



The offsets of objects *within* a frame usually can be statically determined

```
void P (int a, int b) {
    int x, y;
    …
}

void Q () {
    int u, v;
    P (3, 4);
}
```



Nested Scopes and Static Link

Languages with nested functions (Pascal):

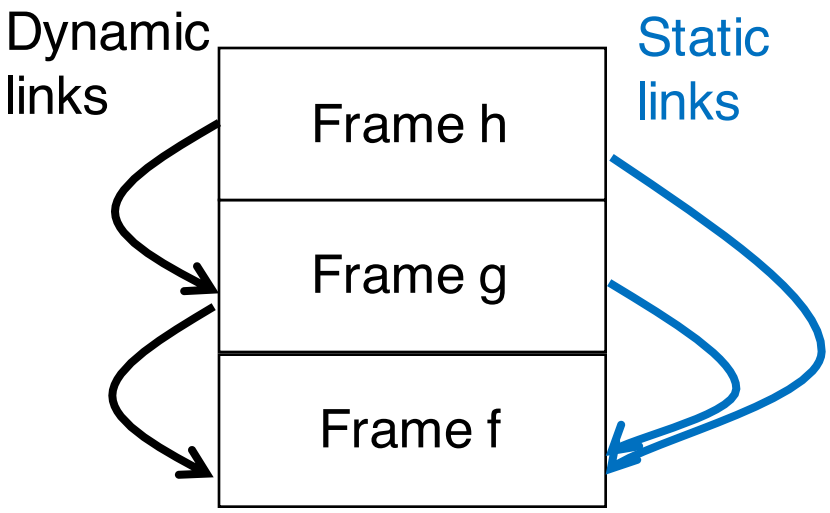
How do we access local variables from other frames?

Static link: a pointer to the frame of enclosing function

Previous fp = **dynamic link**, i.e., pointer to the previous frame in the current execution (dynamic)

Static Link and Dynamic Link

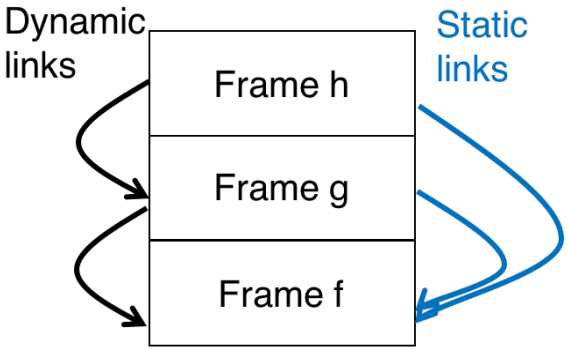
```
void f (int i) {
    int a;
    void h (int j) {
      a = j;
    }
    void g (int k) {
      h(k);
    }
    g(i+2);
}
```

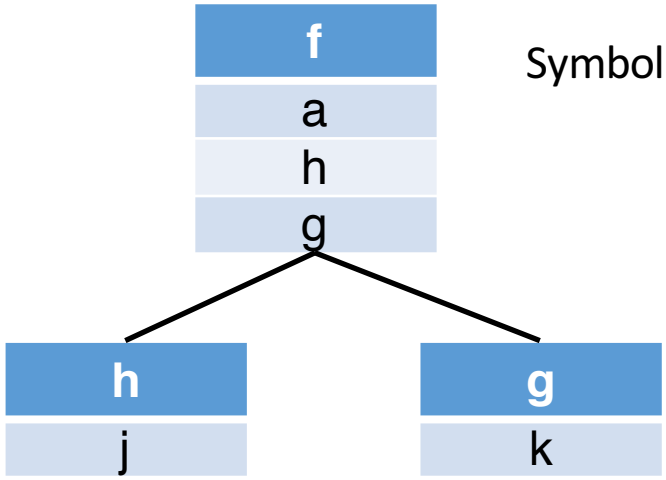


Where is variable a in h?

Static Scoping

```
void f (int i) {
    int a;
    void h (int j) {
      a = j;
    }
    void g (int k) {
      h(k);
    }
    g(i+2);
}
```





Symbol tables

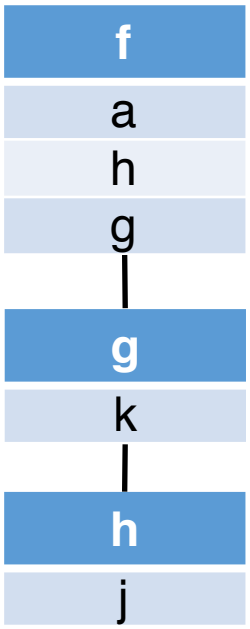
Where is a? From frame of h, go up one hop following static links

Where is a? From symbol table of h, go up one hop

Where is variable a in h?

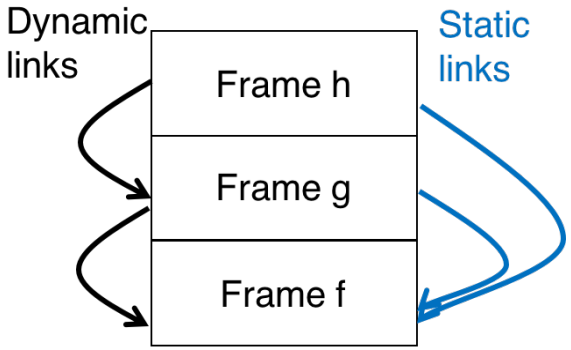
Dynamic Scoping

```
void f (int i) {
   int a;
   void h (int j) {
     a = j;
   }
   void g (int k) {
     h(k);
   }
   g(i+2);
}
```



Symbol tables

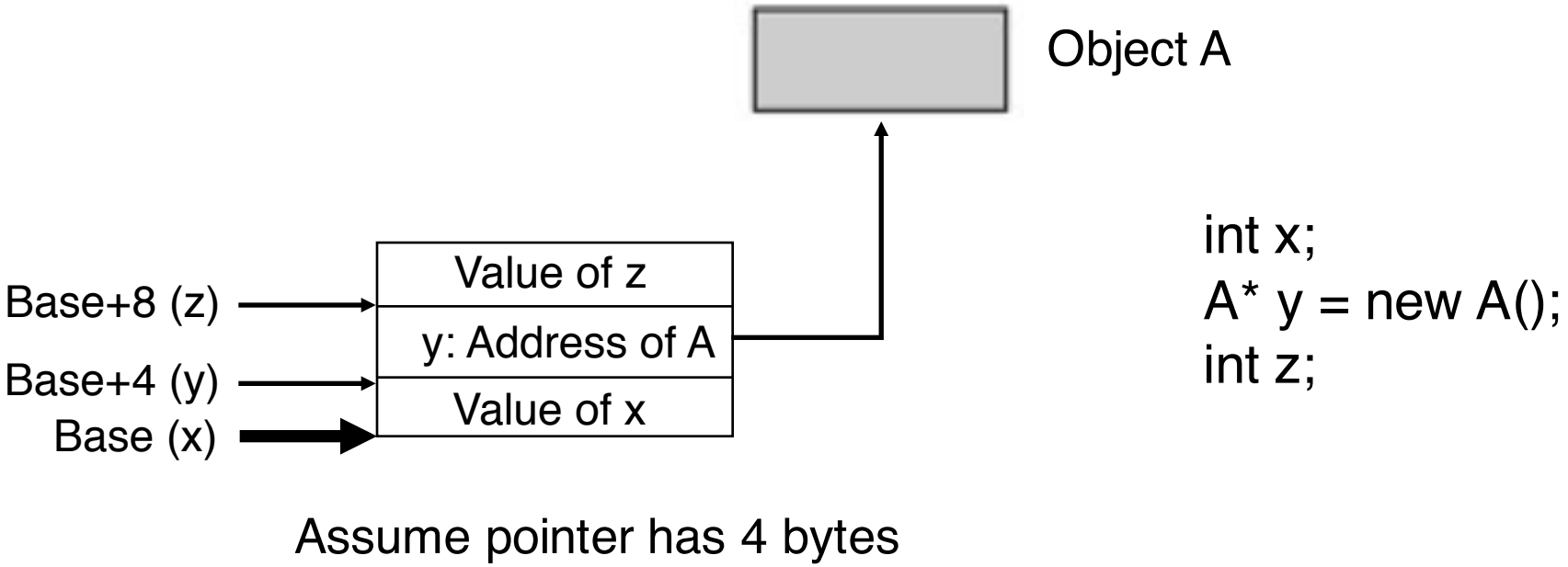
Where is a? From symbol table of h, go up two hops



Where is a? From frame of h, go up two hops following dynamic links

Heap-Based Variable

Similar to a normal variable, except that the memory cell stores the address of heap space



Big Picture: Data Memory Layout