Runtime System- 2

- Lexical scoping - how to manage with stack
  - Use of display
- How to handle dynamic scope?
- Heap allocation

Managing Lexical Scoping

- **Nested procedure definitions or nested begin/end blocks or let expressions**
- **Conceptually can treat *let* expression as an unnamed procedure with its own frame**
- ***Display* - an invention used with Algol60 to help with lookup in nested lexical scopes**
  - **Display array has pointers into runtime stack for each lexically encompassing environment**
  - ***Display_top* pointer keeps track of current scope**

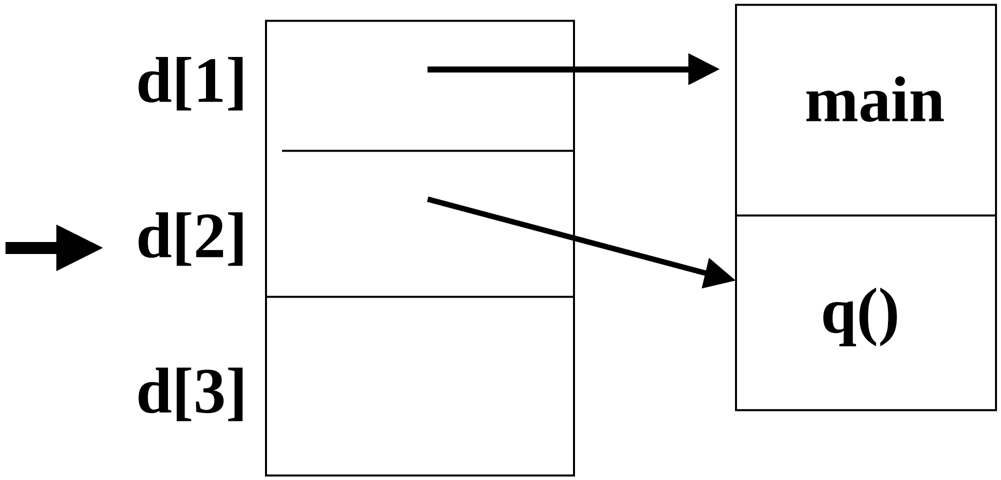
Display

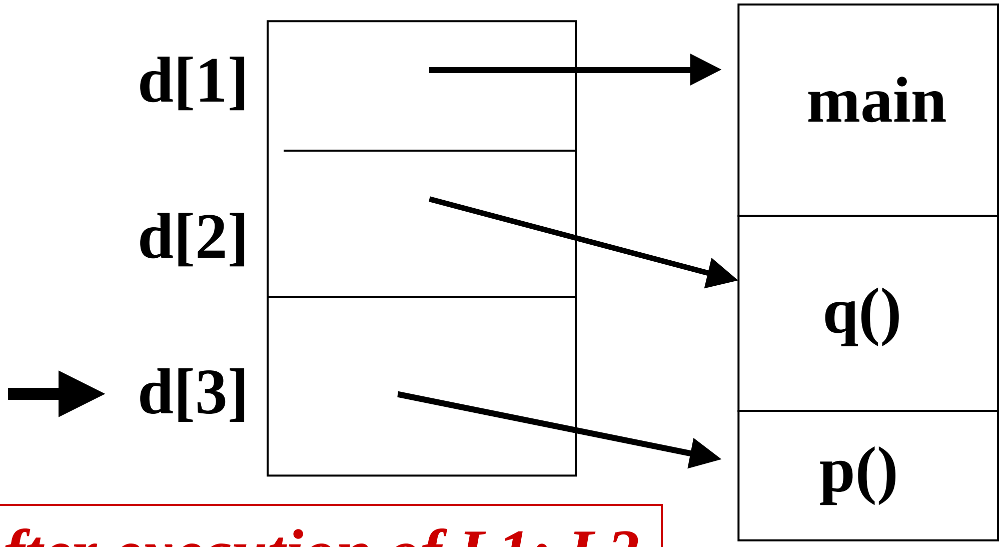
- **An array such that d[j] points to the frame of procedure at nesting depth j, where d[1] points to *main*'s frame**
- **How to maintain?**
  - **When procedure p's frame is put on runtime stack and p's declaration is nested at level j, then save value of d[j] in the new frame and make d[j] point to the new frame**
  - **When p's frame is popped from the stack restore the value of d[j]**

Example

```
main
  proc q()
      proc x()
      L4: w();
      L5:...
      end x;
      proc p()
      L3: x();
      end p;
  L2: p();
  end q;
  proc w()
  end w;
L1: q();
end main
```

After execution of L1

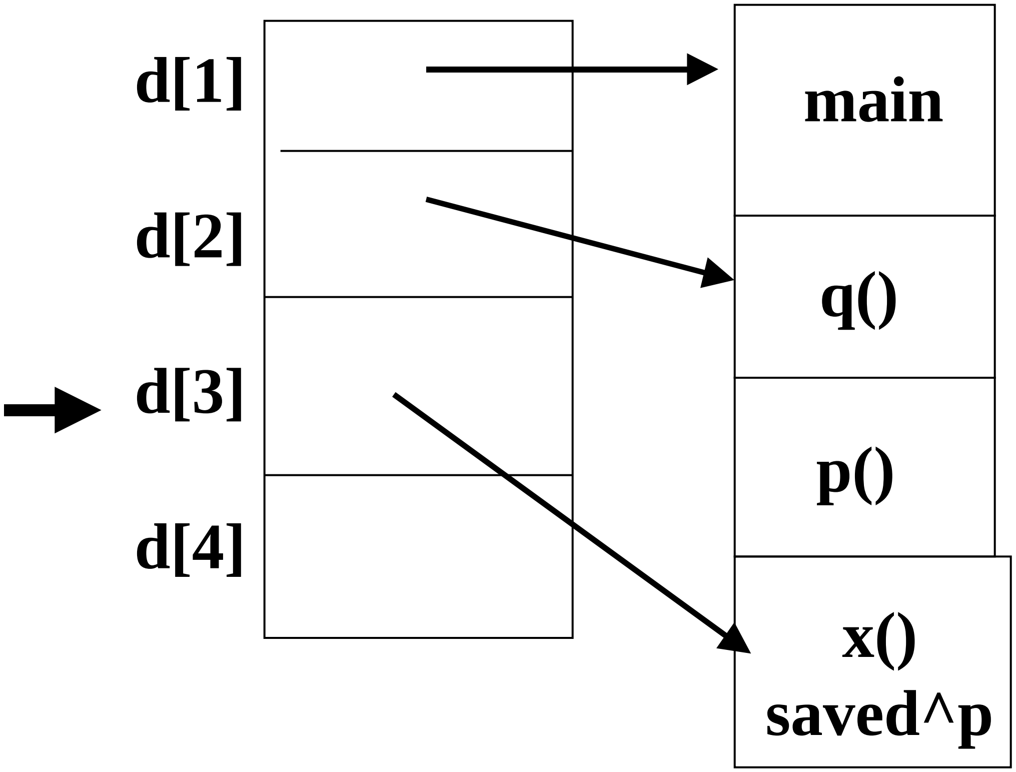




After execution of L1; L2

Example

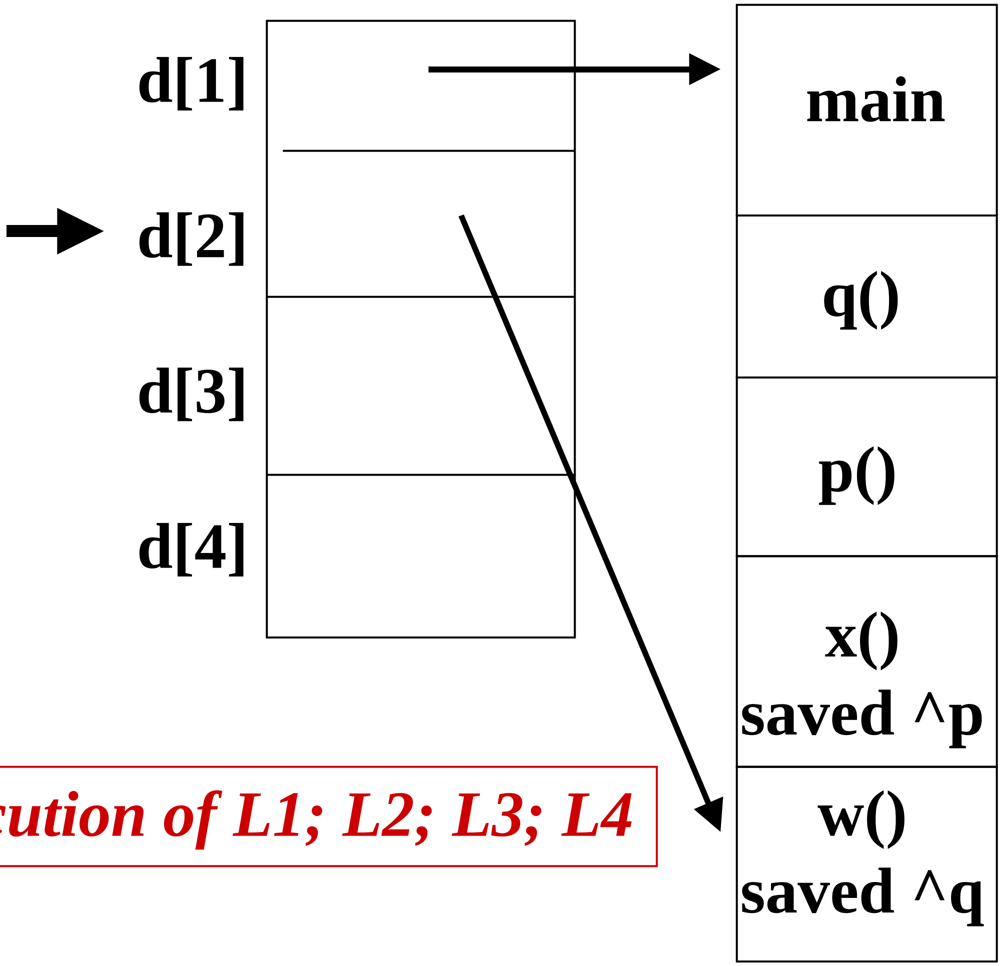
```
main
  proc q()
       proc x()
       L4: w();
       L5: ...
       end x;
       proc p()
       L3: x();
       end p;
  L2: p();
  end q;
  proc w()
  end w;
L1: q();
end main
```



After execution of L1; L2; L3;

Example

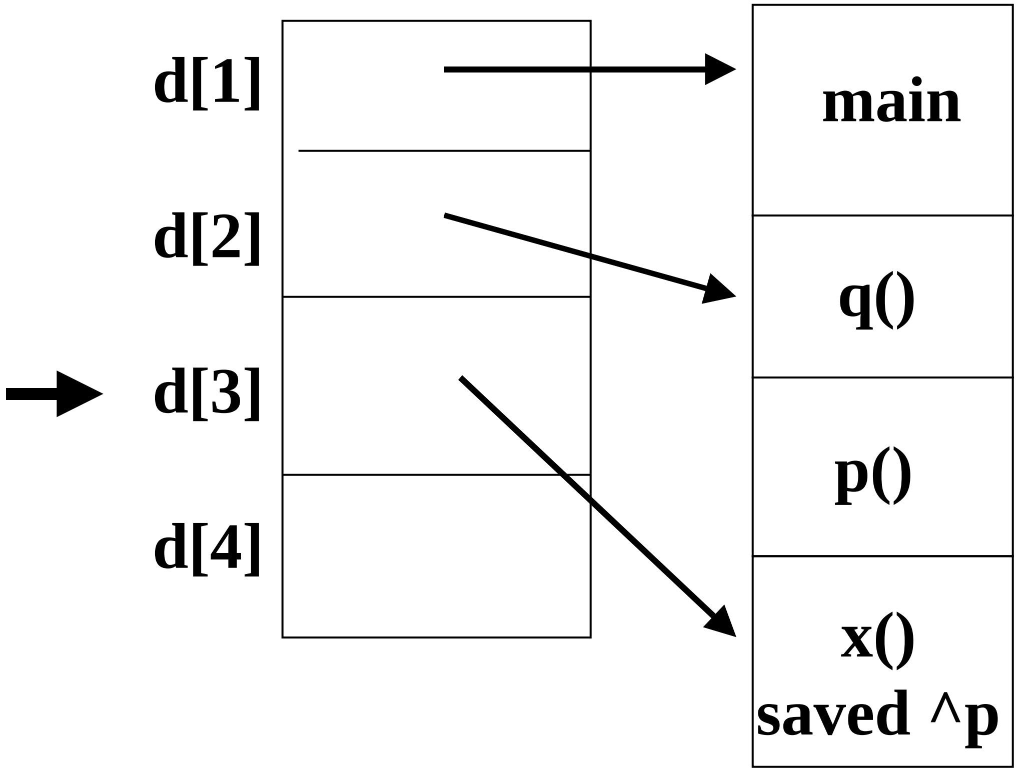
```
main
    proc q()
        proc x()
        L4: w();
        L5: ...
        end x;
        proc p()
        L3: x();
        end p;
   L2: p();
   end q;
   proc w()
   end w;
L1: q();
end main
```


d[1]
d[2]
d[3]
d[4]
main
q()
p()
x()
saved ^p
w()
saved ^q

After execution of L1; L2; L3; L4

Example

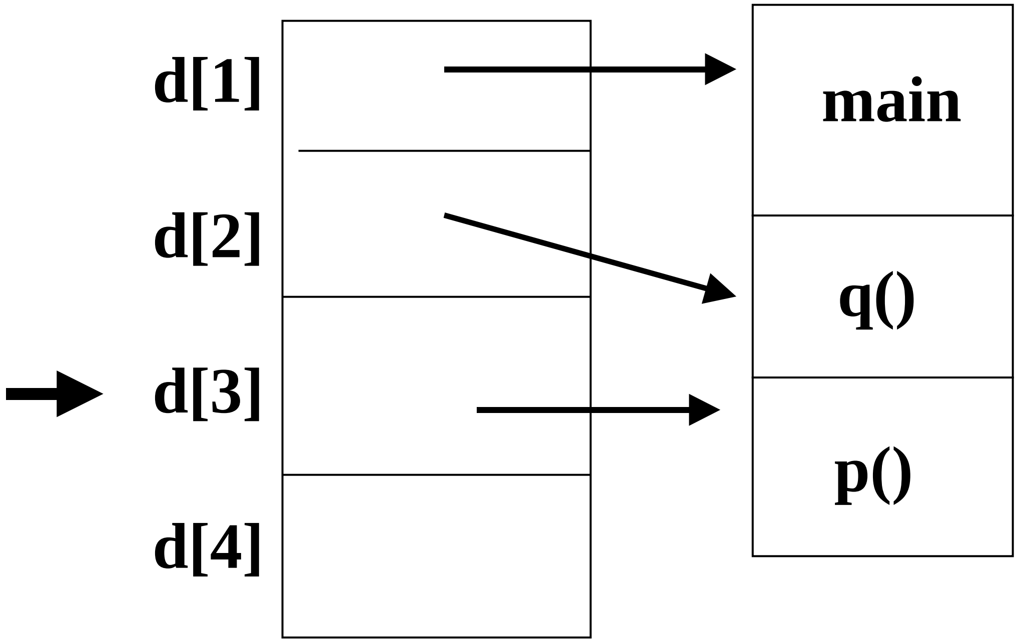
```
main
    proc q()
            proc x()
            L4: w();
            L5: ...
            end x;
            proc p()
            L3: x();
            end p;
    L2: p();
    end q;
    proc w()
    end w;
L1: q();
end main
```



After having returned from L4;

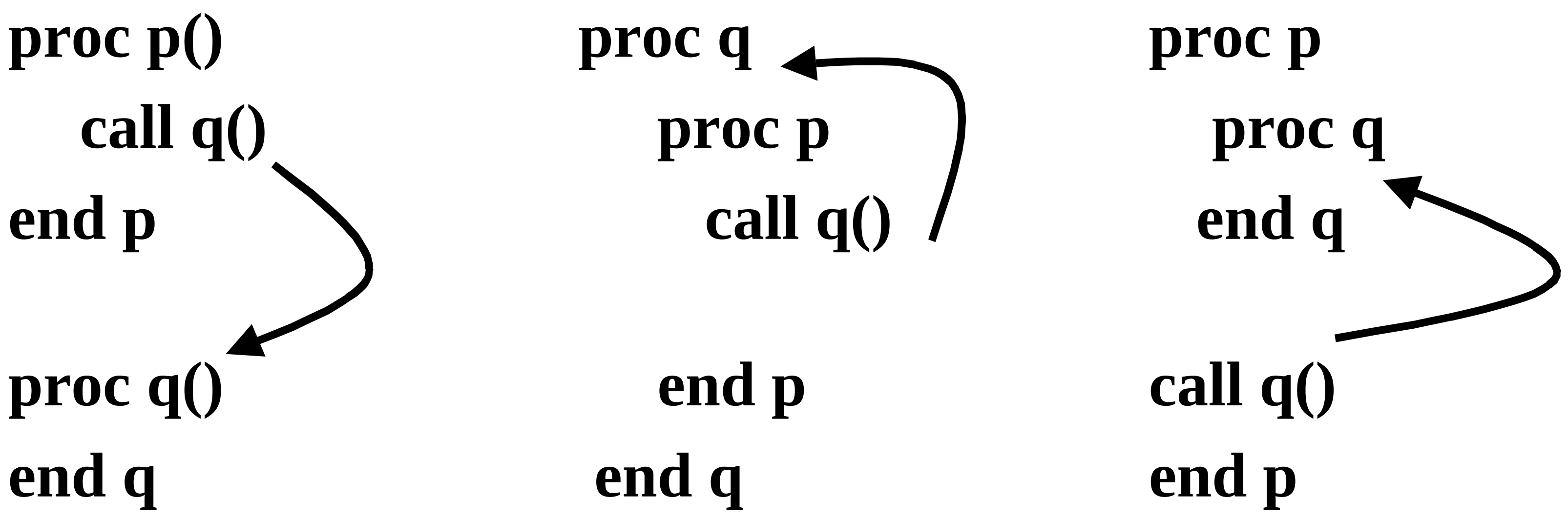
Example

```
main
  proc q()
    proc x()
    L4: w();
    L5: ...
    end x;
    proc p()
    L3: x();
    end p;
  L2: p();
  end q;
  proc w()
  end w;
L1: q();
end main
```



After having returned from L5;

Legal Nesting Patterns



```
proc p()
   call q()
end p

proc q()
end q
```

```
proc q
   proc p
      call q()

   end p
end q
```

```
proc p
   proc q
   end q

call q()
end p
```

PL's with nested procedure declarations permit these patterns of calls.

Dynamic Scope

- Nonlocal names are fetched from most recently executed scope
- Not a popular mechanism
  - Lisp used to use this and then changed to static scoping when Scheme was designed
  - Prolog still uses this
- Can implement using control link in the runtime stack

Heap Storage

- Problems
  - Dangling pointers and garbage
  - Storage fragmentation
- Modern languages offer user allocation and deallocation commands
  - Need for garbage collection techniques
    - Modern OOPL's have them: Java