Today's Topics

Last Time

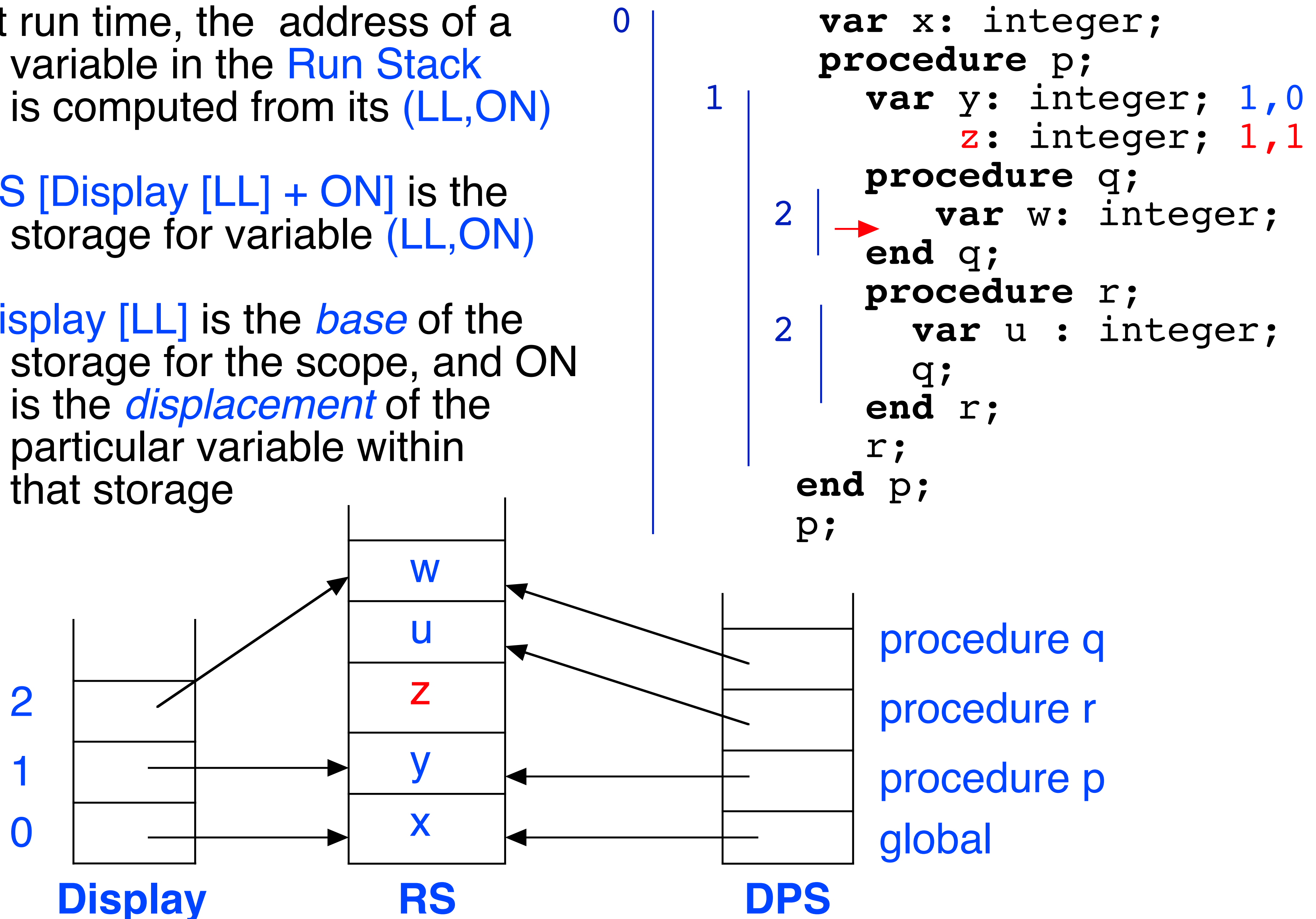
- Modelling Scopes and Visibility
- The Run Stack, the Dynamic Pointer Stack, the Display
- (LL,ON) addressing

Today

- Maintaining the Display
- Kinds of parameters and parameter passing

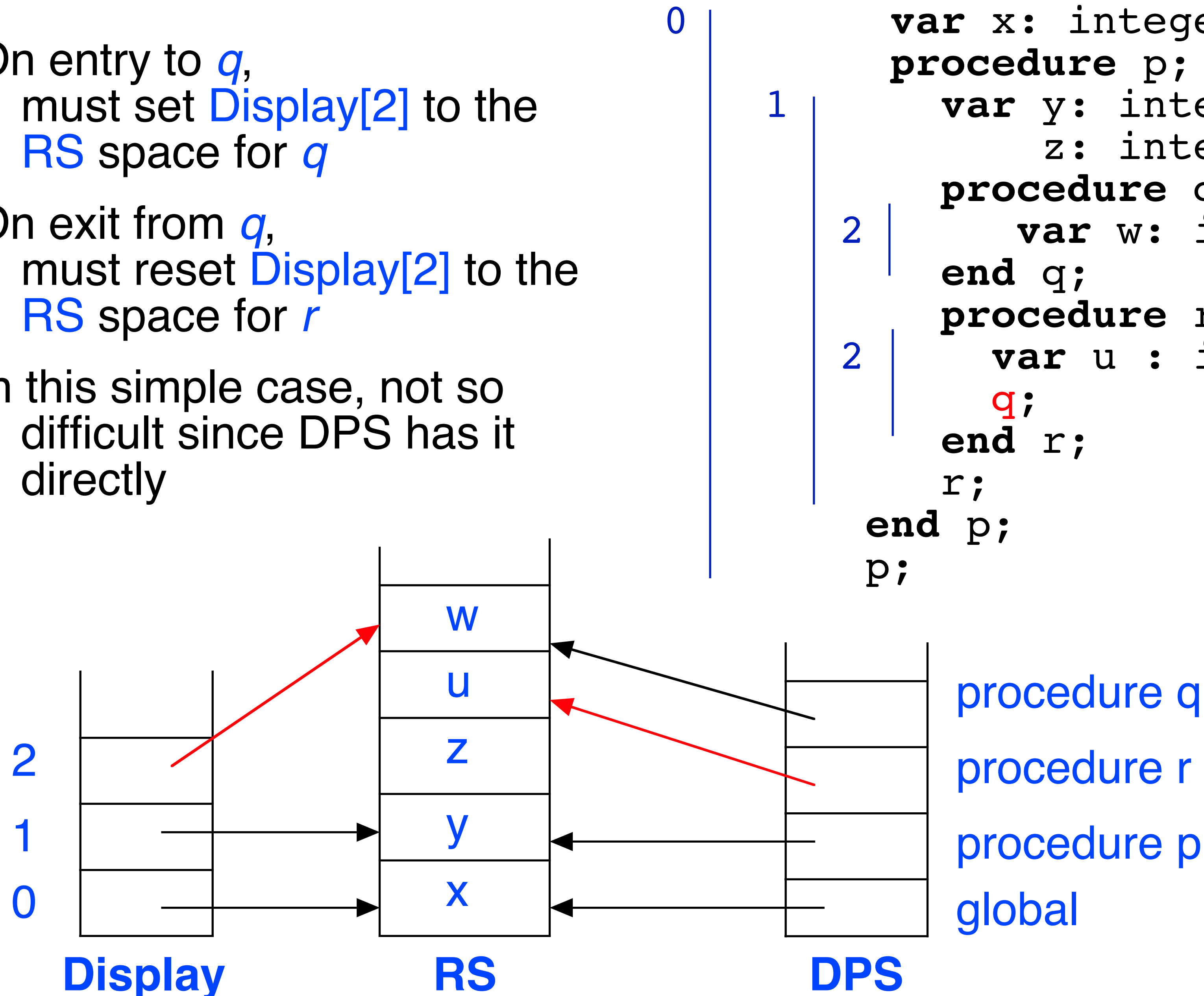
© 2019 J.R. Cordy

(LL,ON) Address Calculation



Maintaining the Display

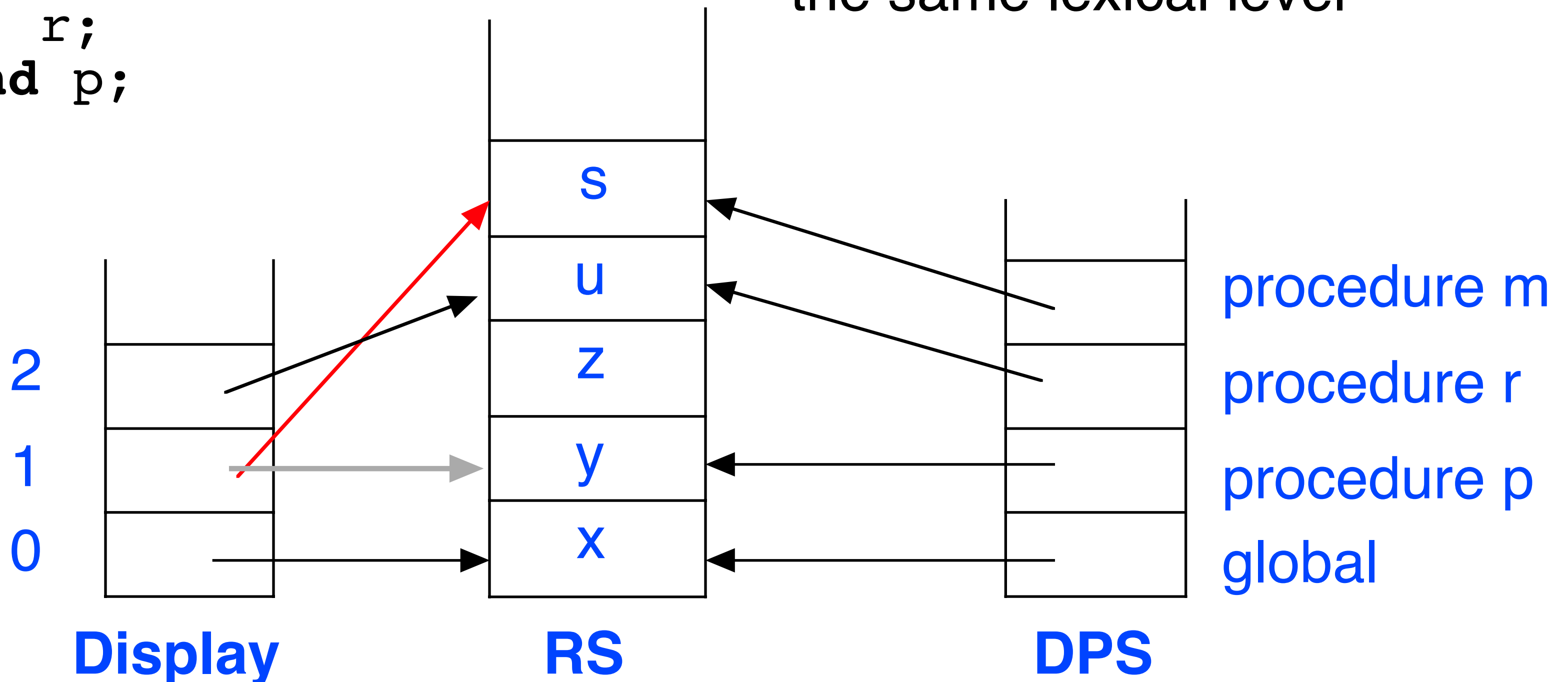
- On entry to *q*, must set Display[2] to the RS space for *q*
- On exit from *q*, must reset Display[2] to the RS space for *r*
- In this simple case, not so difficult since DPS has it directly



Maintaining the Display

```
0       var x: integer;
        procedure m;
1          var s;
        end m;
        procedure p;
1          var y: integer;
               z: integer;
           procedure r;
2             var u : integer;
              m;
           end r;
           r;
       end p;
       p;
```

- Not always so easy - when calls are made *"uplevel"*, restoring the Display may not be so simple
- Example: When we leave *m*, we must restore Display[1] to point to the frame for *p*
- Somehow we must keep track of which is the previous scope at the same lexical level

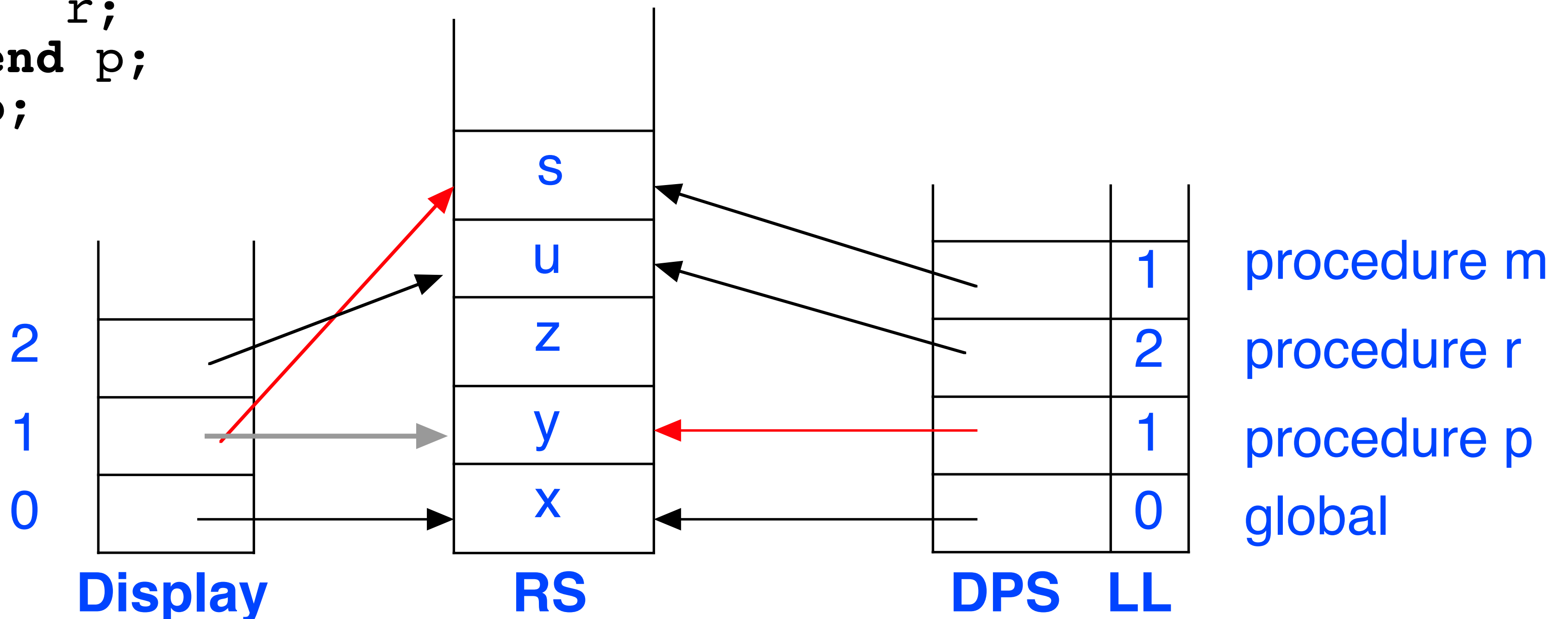


© 2019 J.R. Cordy

Maintaining the Display

```
0  var x: integer;
   procedure m;
1     var s;
→  end m;
   procedure p;
1     var y: integer;
          z: integer;
      procedure r;
2        var u : integer;
         m;
      end r;
      r;
   end p;
   p;
```

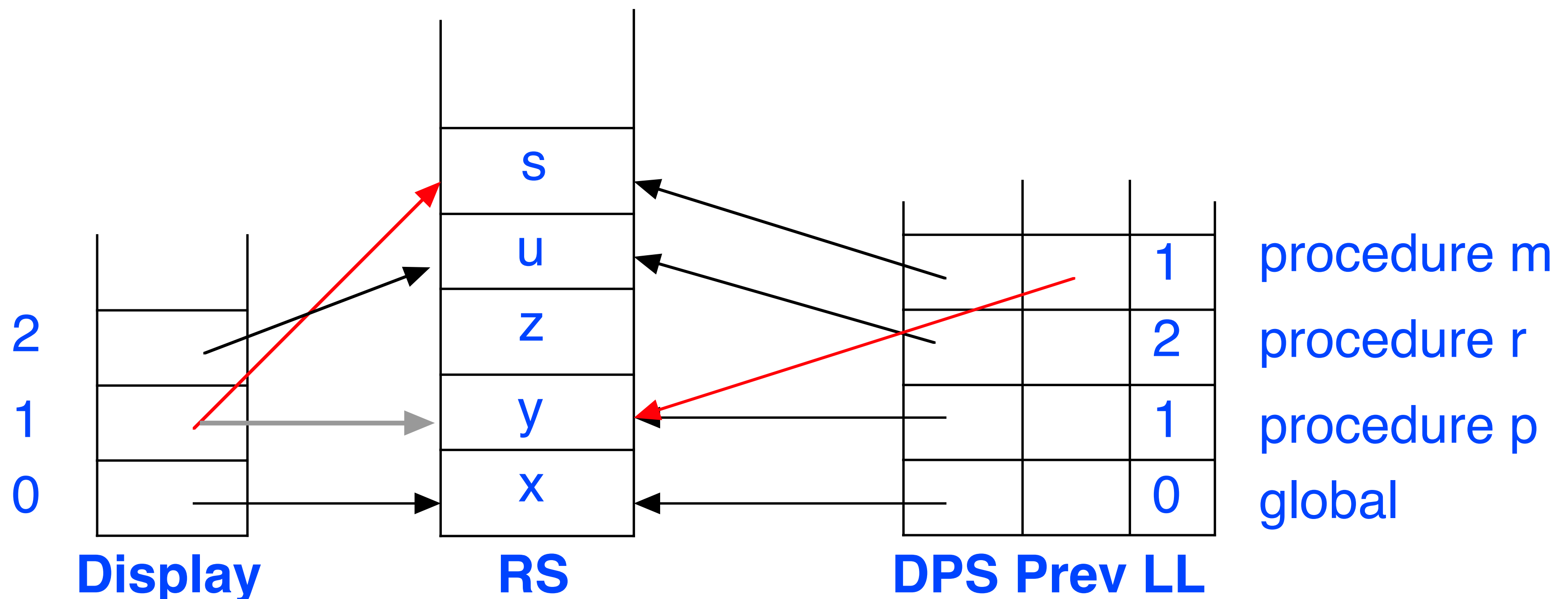
- There are many ways we could keep track - one of the simplest is to store the lexical level for each scope in the DPS
- Now when we leave *m*, we look down the DPS for the next entry with the same lexical level - in this case it is the scope of procedure *p*



© 2019 J.R. Cordy

Maintaining the Display

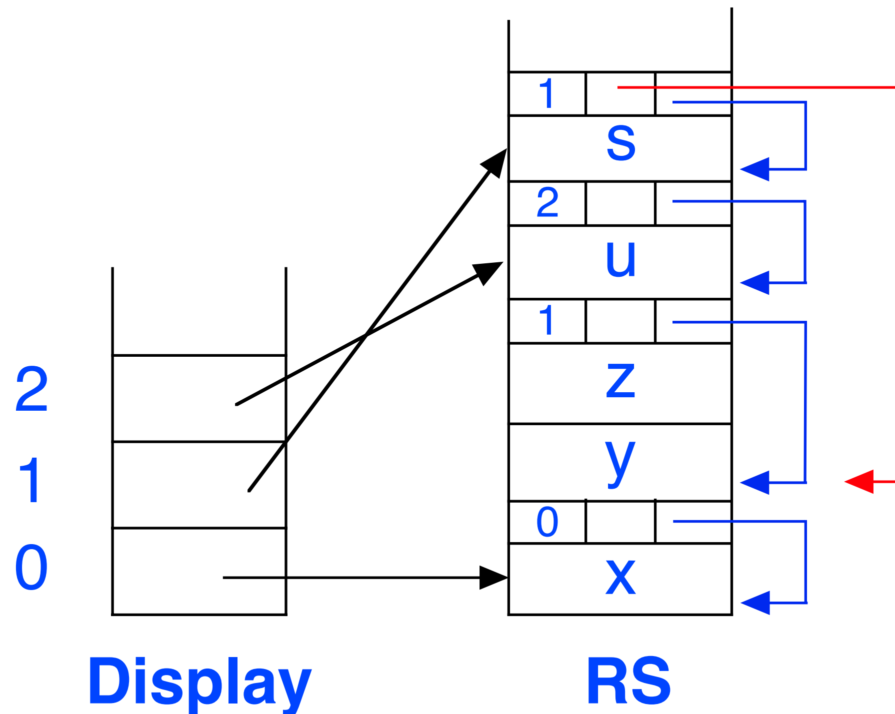
- In practical solutions, the previous Display entry at the lexical level is stored in the DPS, eliminating the search
- On entry to a scope at LL, the previous Display[LL] pointer is stored in the DPS before setting Display[LL] to the new scope base
- On exit from a scope at LL, Display[LL] is simply set to the previous Display[LL] entry stored in the DPS



© 2019 J.R. Cordy

Maintaining the Display

- Optimized solutions do even better, storing the previous DPS pointer and Display pointers for a new scope directly in the RS itself, eliminating the DPS entirely
- This is called *dynamic chaining* (for the DPS pointers) and *static chaining* (for the Display pointers)



© 2019 J.R. Cordy

Parameters

- Languages have several kinds of parameter passing:

- *Pass by Value* – A copy of the argument value is passed
  - Changes to parameter do not affect the original argument variable
  - Expressions may be used as arguments - value is computed and passed as the value of the parameter
  - Java, default parameter in Pascal

- *Pass by Reference* – A reference (pointer) to the original argument variable is passed
  - Inside the procedure, the pointer is implicitly dereferenced, changes to the parameter change the original argument variable
  - Argument must be a variable, expressions are not allowed
  - **var** parameter in Pascal, Turing, PL/I

© 2019 J.R. Cordy

Parameters (cont'd)

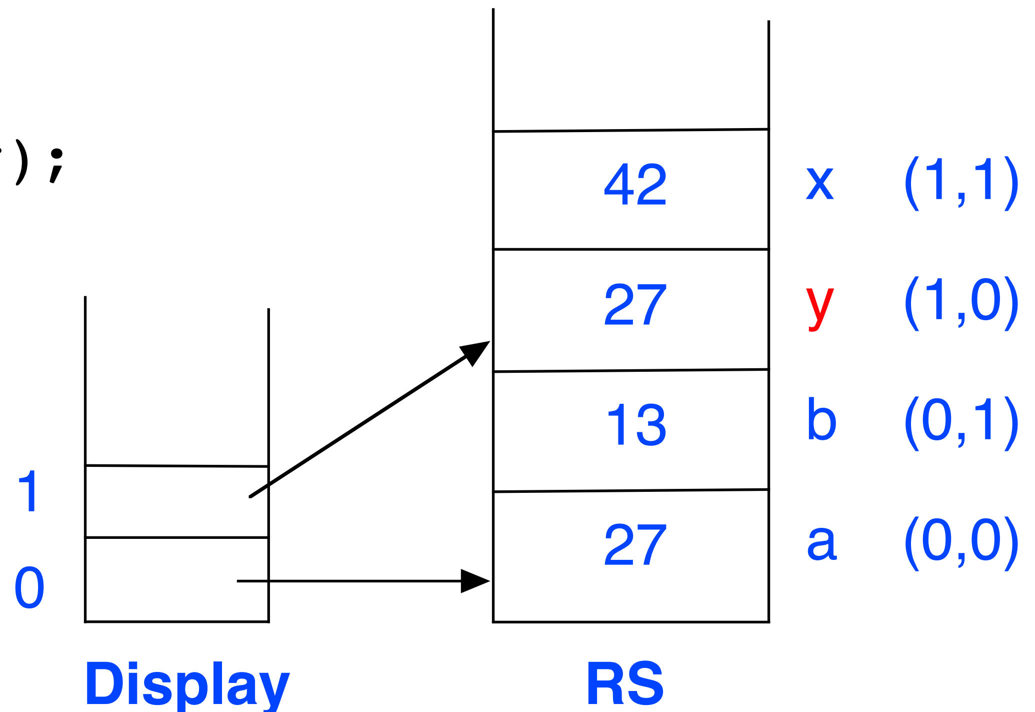
- Languages have several kinds of parameter passing:
- *Pass by Name* – The actual source text of the argument expression is passed
  - Substituted and evaluated in the context of its use inside the procedure
  - Can be useful, but confusing and not used in recent languages
  - Available in Algol 60, Algol 68 - not in Pascal
- *Pass by Value-Result* – Similar to pass by value, but at the end of the procedure the final parameter value is assigned back to the original argument variable
  - Algol W, Euclid, Ada

© 2019 J.R. Cordy

Value Parameters

- Parameters are modeled in the abstract machine as the first local variables in the procedure's scope in the RS
- We push them on to the Run Stack before calling the procedure, then set up the DPS and Display to point at them as the base of the scope (makes setting up Display and DPS slightly more complicated)
- Order Numbers (ON) of local variables start with the parameters of the procedure

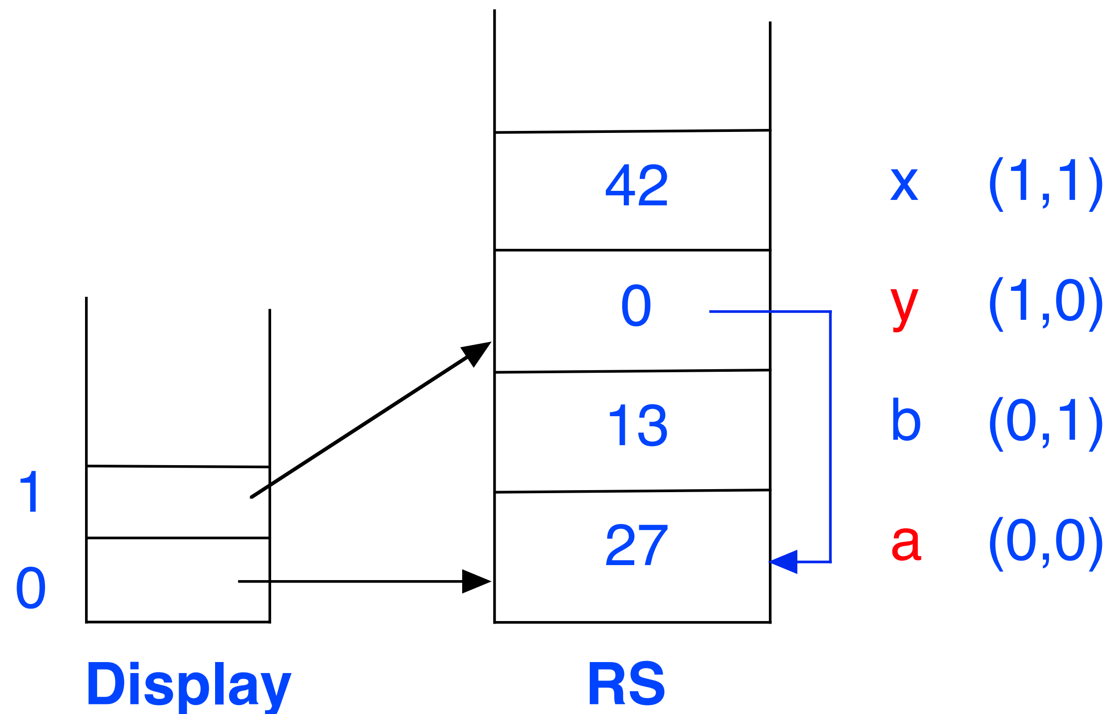
```
var a,b: integer;
procedure p (y: integer);
    var x: integer;
    x := 42;
end p;
a := 27;
b := 13;
p(a);
```



Reference Parameters

- Pass by reference means passing the address (RS stack index) of the argument variable as the value of the parameter
- Inside the procedure, we interpret the value of the parameter as a memory address (RS stack index) of the real parameter

```
var a,b: integer;
procedure p (var y: integer);
   var x: integer;
   x := 42;
end p;
a := 27;
b := 13;
p(a);
```



© 2019 J.R. Cordy

Parameter Passing

- Parameter passing requires some new instructions for our abstract machine model
- Two new instructions are needed:

`pushaddress (LL,ON)`	like *push*, but pushes the RS *address* of the variable onto the ES instead of its value
`passparameter`	pops the top value in the ES and pushes it onto the RS

- To pass variable *z* as a *value* parameter :

push (LL_z,ON_z)	push the *value* of *z* onto the ES
`passparameter`	pop *z's* value from ES and push it on RS

- To pass *z* as a *reference* parameter :

pushaddress (LL_z, ON_z)	push the RS index of *z* onto ES
`passparameter`	pop *z's* address from ES and push it on RS

© 2019 J.R. Cordy

Summary

Modelling Scopes and Visibility

- Maintaining the Display
- Kinds of parameters and parameter passing

Next

- Model for procedures and functions

© 2019 J.R. Cordy