

Run-Time Environments

Chapter 7

Procedure Activation and Lifetime

- A procedure is activated when called
- The *lifetime* of an activation of a procedure is the sequence of steps between the first and last steps in the execution of the procedure body
- A procedure is *recursive* if a new activation can begin before an earlier activation of the same procedure has ended

Procedure Activations

```

program sort(input, output)
  var a : array [0..10] of integer;
  procedure readarray;
    var i : integer;
    begin
      for i := 1 to 9 do read(a[i])
    end;
  function partition(y, z : integer) : integer
    var i, j, x, v : integer;
    begin ...
    end
  procedure quicksort(m, n : integer);
    var i : integer;
    begin
      if (n > m) then begin
        i := partition(m, n);
        quicksort(m, i - 1);
        quicksort(i + 1, n)
      end
    end;
begin
  a[0] := -9999; a[10] := 9999;
  readarray;
  quicksort(1, 9)
end.

```

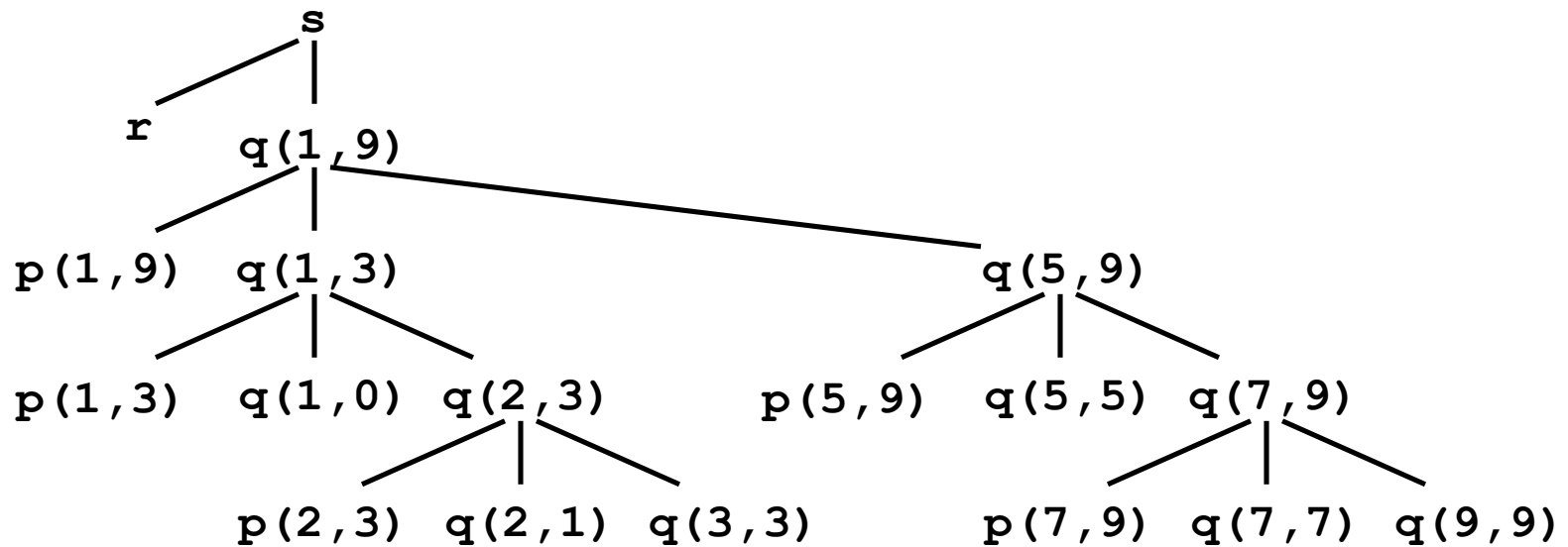
Activations:

```

begin sort
  enter readarray
  leave readarray
  enter quicksort(1, 9)
  enter partition(1, 9)
  leave partition(1, 9)
  enter quicksort(1, 3)
  ...
  leave quicksort(1, 3)
  enter quicksort(5, 9)
  ...
  leave quicksort(5, 9)
  leave quicksort(1, 9)
end sort.

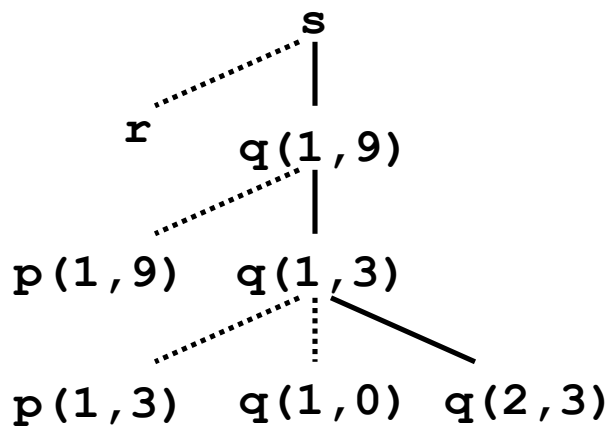
```

Activation Trees



Control Stack

Activation tree:



Control
stack:

s
q(1, 9)
q(1, 3)
q(2, 3)

Activations:

```

begin sort
enter readarray
leave readarray
enter quicksort(1, 9)
enter partition(1, 9)
leave partition(1, 9)
enter quicksort(1, 3)
enter partition(1, 3)
leave partition(1, 3)
enter quicksort(1, 0)
leave quicksort(1, 0)
enter quicksort(2, 3)

```

...

Scope Rules

- *Environment* determines name-to-object bindings: which objects are in scope?

```

program prg;
  var y : real;
  function x(a : real) : real;
    begin ... end;
  procedure p;
    var x : integer;
    begin
      x := 1;
      ...
    end;
  begin
    y := x(0.0);
    ...
  end.

```

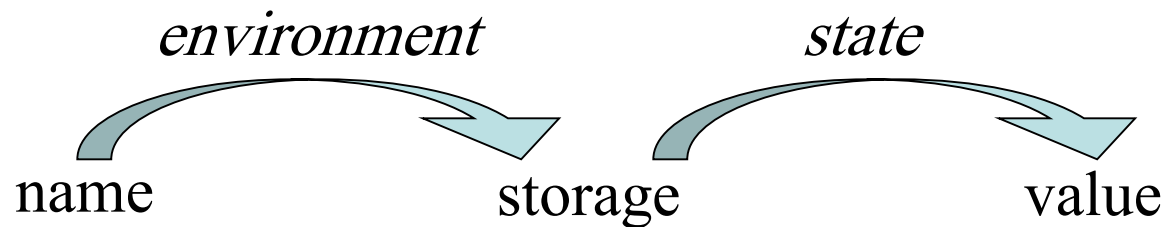
Variable **x** locally declared in **p**



Function **x**



Mapping Names to Values



```
var i;  
...  
i := 0;  
...  
i := i + 1;
```

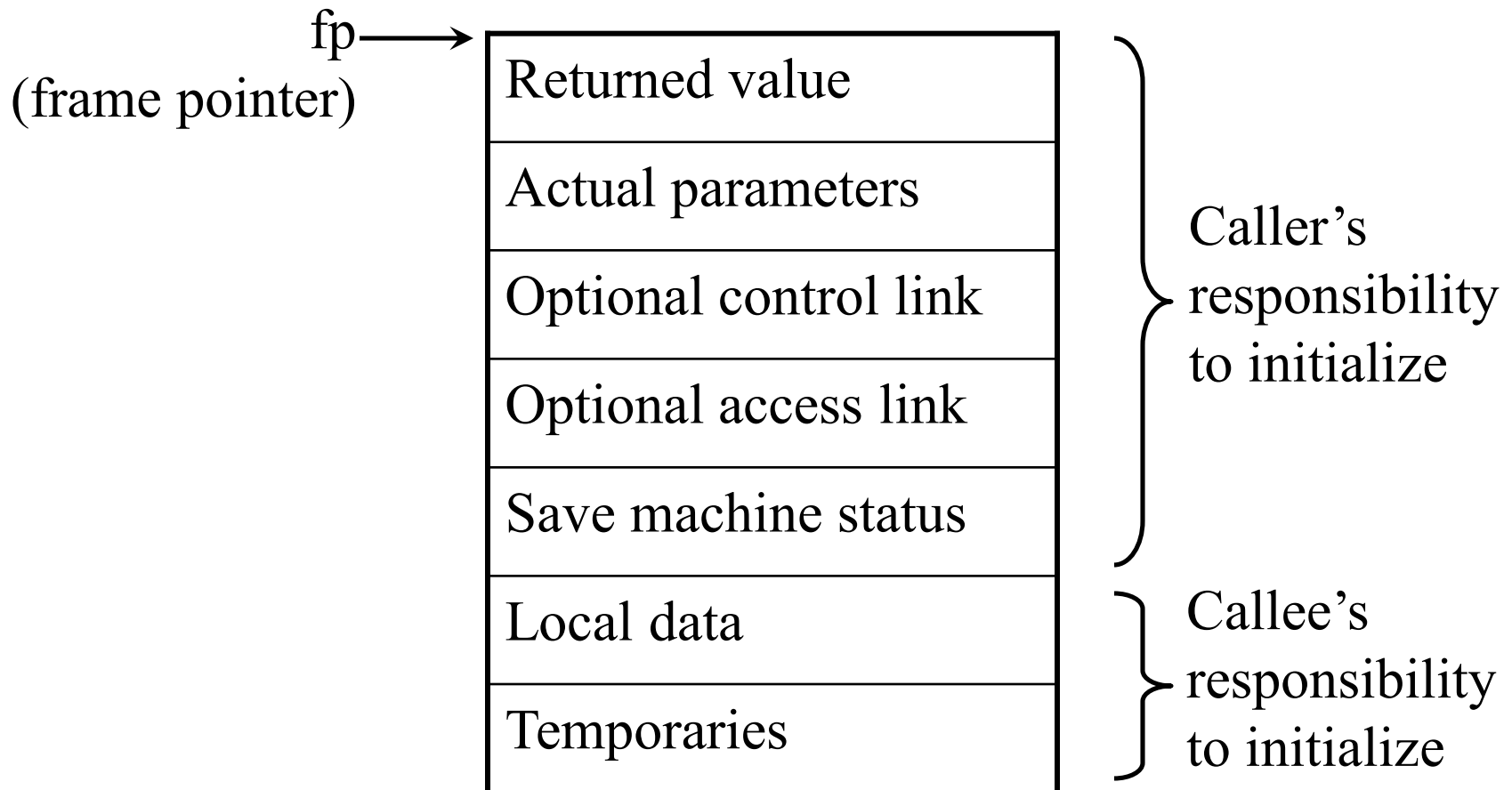
Static and Dynamic Notions of Bindings

<i>Static Notion</i>	<i>Dynamic Notion</i>
Definition of a procedure	Activations of the procedure
Declaration of a name	Bindings of the name
Scope of a declaration	Lifetime of a binding

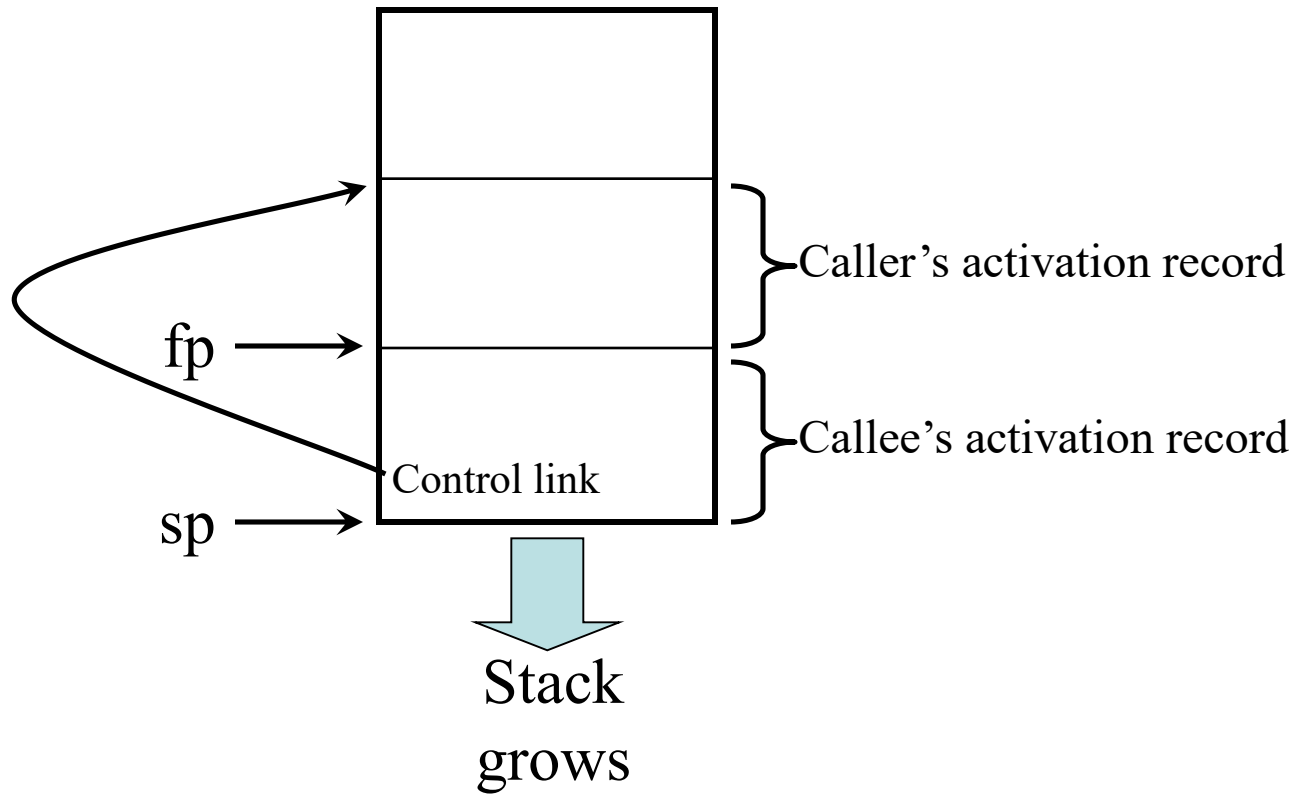
Stack Allocation

- *Activation records* (subroutine frames) on the run-time stack hold the state of a subroutine
- *Calling sequences* are code statements to create activations records on the stack and enter data in them
 - Caller's calling sequence enters actual arguments, control link, access link, and saved machine state
 - Callee's calling sequence initializes local data
 - Callee's return sequence enters return value
 - Caller's return sequence removes activation record

Activation Records (Subroutine Frames)



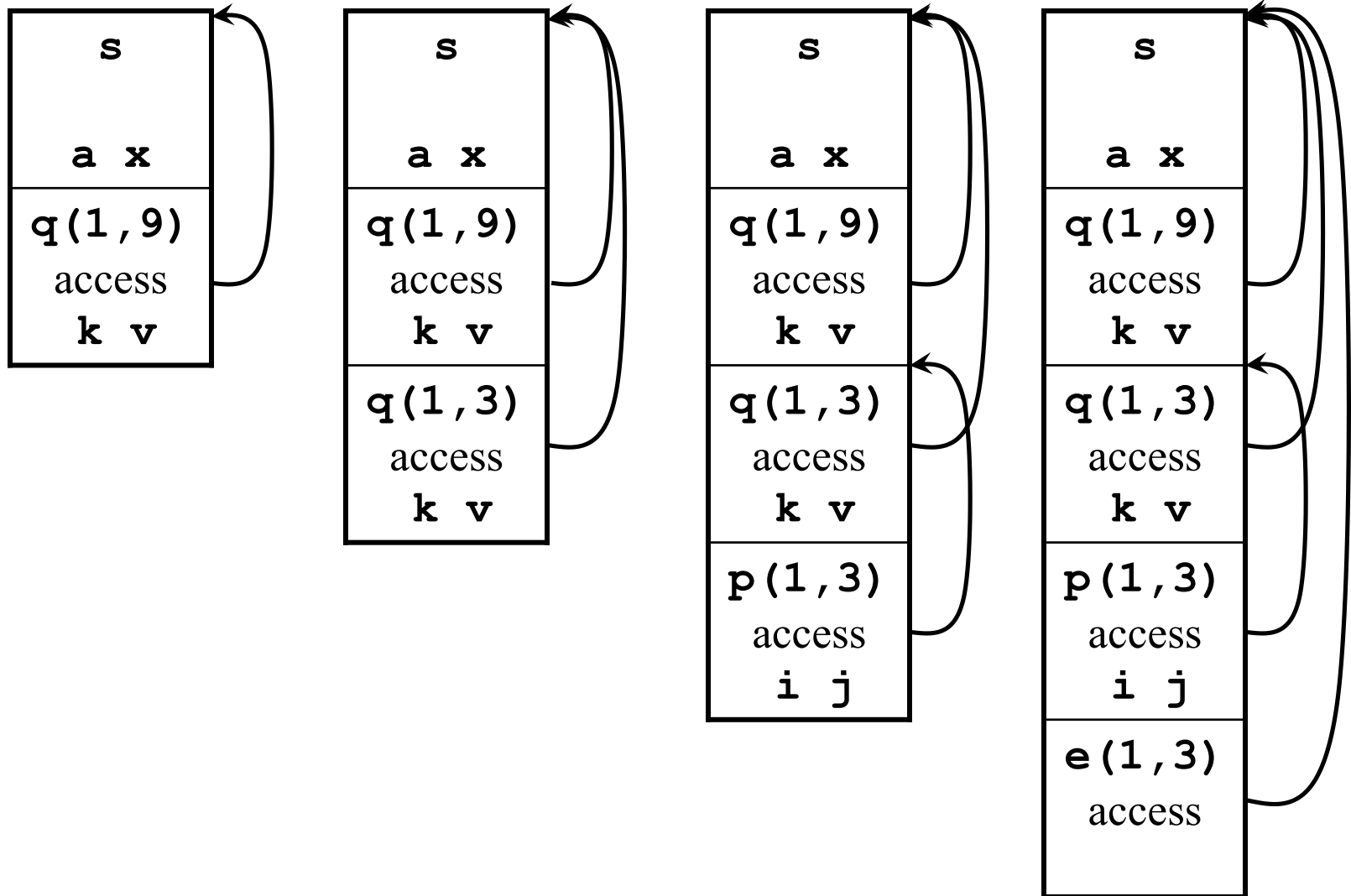
Control Links



Scope with Nested Procedures

```
program sort(input, output)
  var a : array [0..10] of integer;
      x : integer;
  procedure readarray;
    var i : integer;
    begin ... end;
  procedure exchange(i, j : integer);
    begin x := a[i]; a[i] := a[j]; a[j] := x end;
  procedure quicksort(m, n : integer);
    var k, v : integer;
    function partition(y, z : integer) : integer
      var i, j : integer;
      begin ... exchange(i, j) ... end
    begin
      if (n > m) then begin
        i := partition(m, n);
        quicksort(m, i - 1);
        quicksort(i + 1, n)
      end
    end;
  begin
    ...
    quicksort(1, 9)
  end.
```

Access Links



Accessing Nonlocal Data

- To implement access to nonlocal data a in procedure p , the compiler generates code to traverse $n_p - n_a$ access links to reach the activation record where a resides
 - n_p is the nesting depth of procedure p
 - n_a is the nesting depth of the procedure containing a

Parameter Passing

- Call-by-value: evaluate actual parameters and enter r-values in activation record
- Call-by-reference: enter pointer to the storage of the actual parameter
- Copy-restore (value-result): evaluate actual parameters and enter r-values, after the call copy r-values of formal parameters into actuals
- Call-by-name: use a form of in-line code expansion (*thunk*) to evaluate parameters