CS 315 – Programming Languages
Implementing Subprograms

Subprograms (functions) are implemented via the use of the *function call stack*.

When a function is called, an *activation record instance* (*ARI*) is placed on the stack. And when the function returns, the ARI is removed. An *activation record* (*AR*) is a data structure that stores the necessary context information for a function call. This includes:

- space for the return value
- space for local variables
- space for parameters
- a dynamic link (linking to the activation record that comes before this one)
- return address (to return back to the caller location (PC before the call)
- auxiliary space for execution status of the caller (such as registers)

The semantics of a simple subprogram call requires the following actions:

- Save the execution status of the caller
- Compute and pass parameters
- Save the return address
- Transfer control to the called

The call return actions include the following:

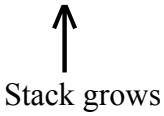
- Return value is moved to a place accessible to the caller
- The execution status of the caller is resstored
- Control is transferred back to the caller

There are a few special registers that have to be maintained:

- PC (program counter)
  - Points to the currently executing instruction.
  - Set to the called function's address when the call is made, restored from the return address stored in the ARI when the call returns.
- FP (frame pointer):
  - Points to the base of the stack
  - Used for accessing local variables and parameters that are at fixed offsets from the base of the stack
  - Used for traversing the call chain (think exception stack traces)
  - Restored from the dynamic link stored in the ARI
- SP (stack pointer)
  - Points to the top of the stack

Generic format of an AR:

Return value
Local variables
Parameters
Dynamic link
Return address



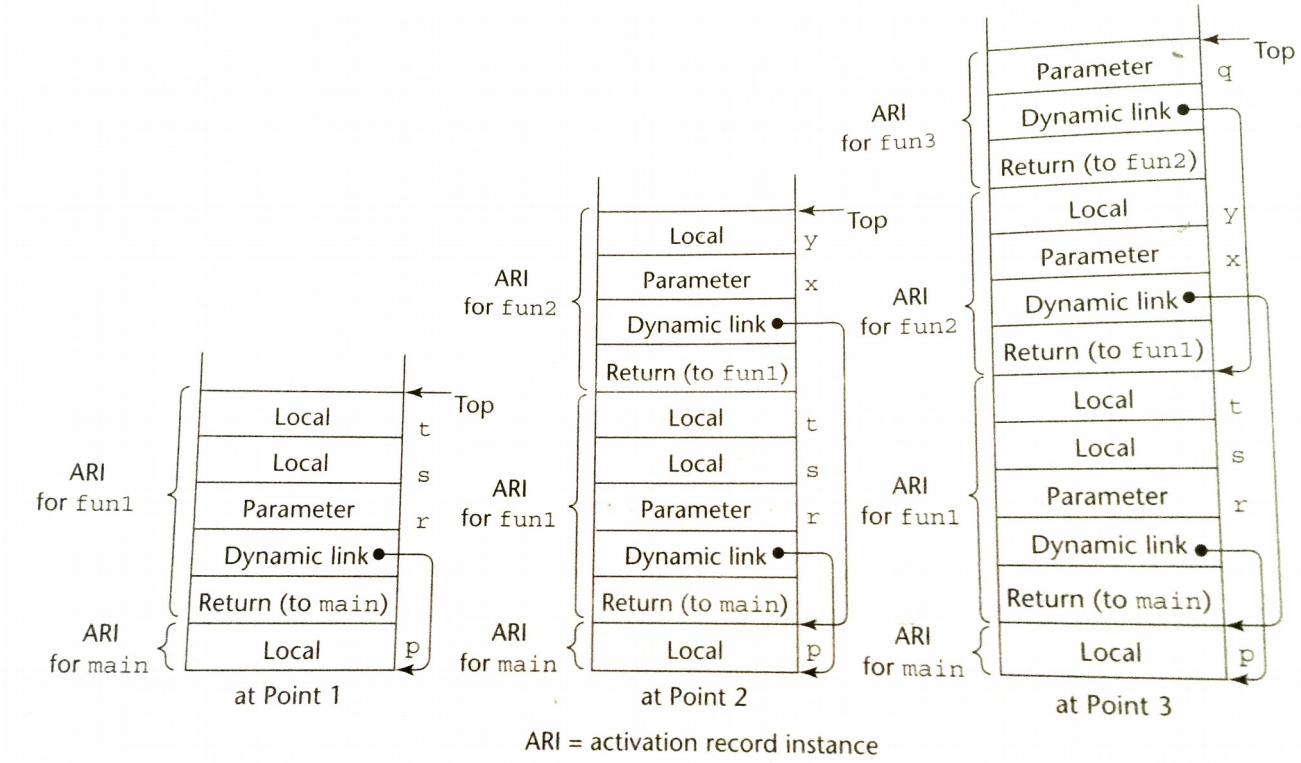
E.g:

```
void sub(float total, int part)
{
        int list[3];
        float sum;
        …
}
```

Local	sum
Local	list[2]
Local	list[1]
Local	list[0]
Parameter	part
Parameter	total
Dynamic link	
Return address	

Example without recursion:

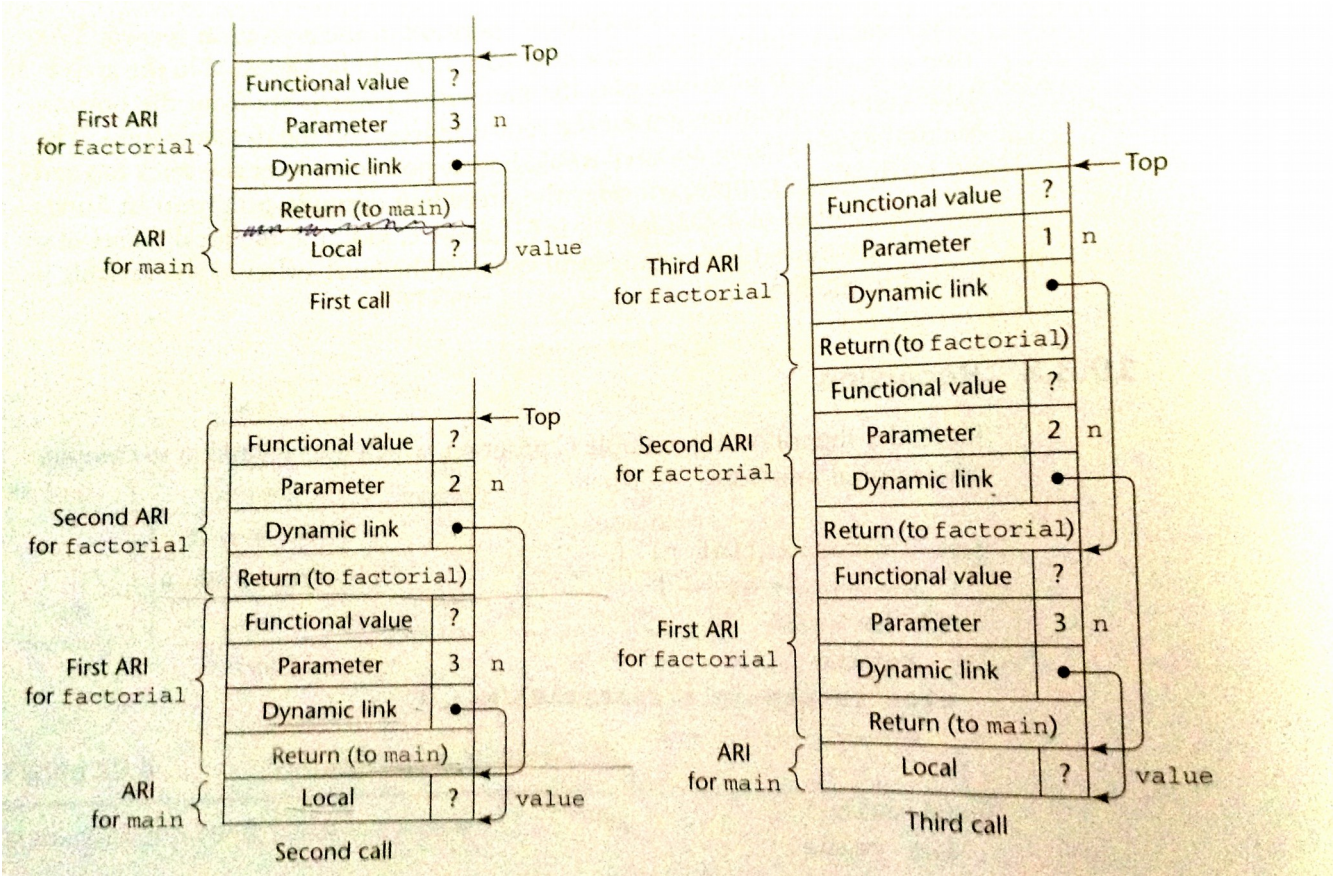
```
void fun1(float r) {
        int s, t;
                                <---------- location 1
        …
        fun2(s);
        …
}
void fun2(int x) {
        int y;
                                <---------- location 2
        …
        fun3(y);
        …
}
void fun3(int q)  {
                                <---------- location 3
        …
}
void main() {
        float p;
        …
        fun1(p);
        …
}
```

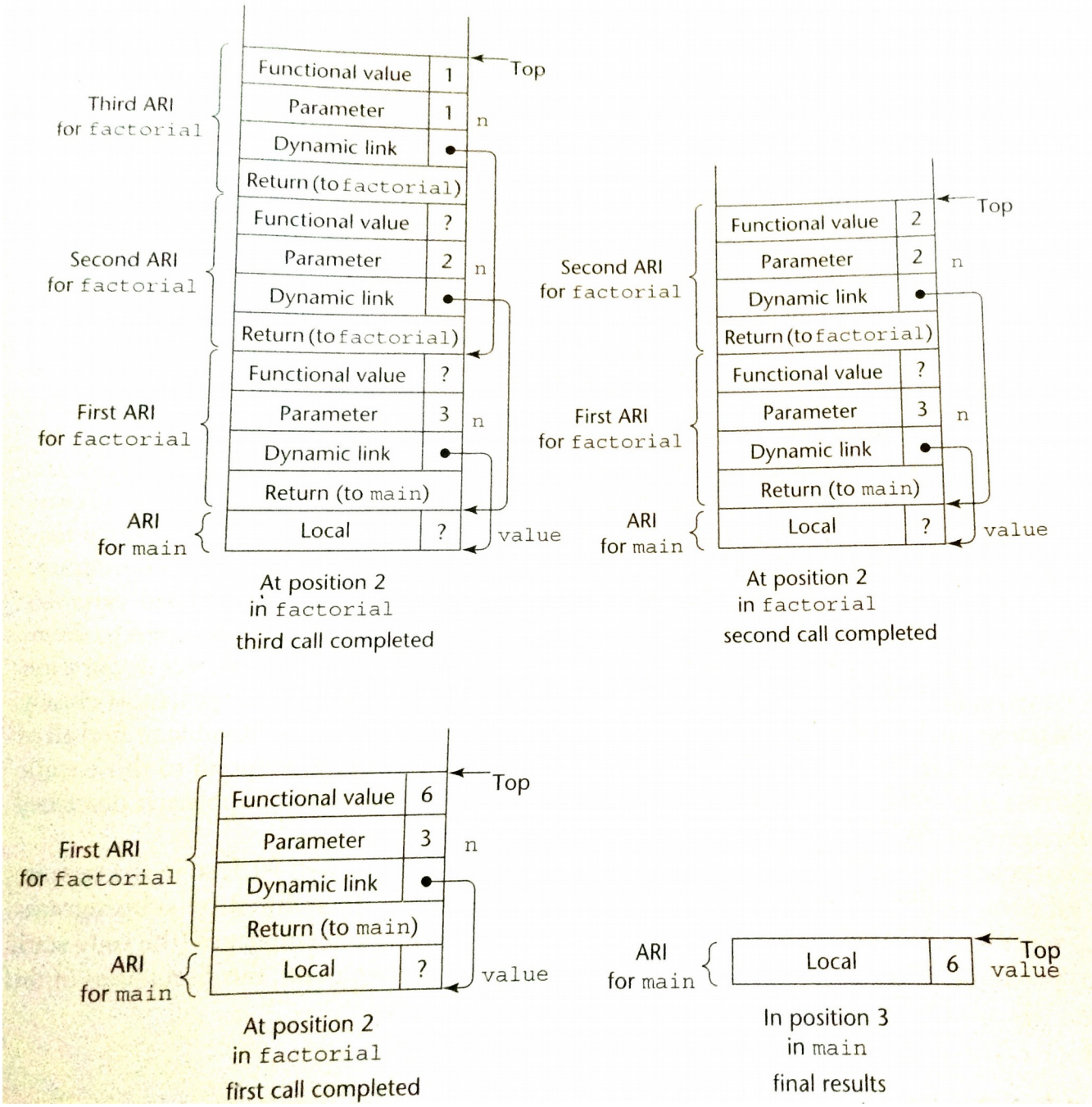


Example with recursion:

```
int factorial(int n) {
                                <---------- location 1 (visited 3 times)
        if (n<=1)
                return 1;
        else
                return (n * factorial(n-1));
                                <---------- location 2 (visited 3 times)
}

void main() {
        int value;
        value = factorial(3);
                                <---------- location 3 (visited 1 time)
}
```




Third ARI for factorial
Second ARI for factorial
First ARI for factorial
ARI for main
Functional value
Parameter
Dynamic link
Return (to factorial)
Local
Top
n
value
At position 2 in factorial third call completed
At position 2 in factorial second call completed
At position 2 in factorial first call completed
In position 3 in main final results

To implement nested functions, typically a *static link* field is added into the ARs. The static link always points to the base of the ARI of the function that is syntactically enclosing the current one.

Consider the following example:

```
procedure Main_2 is
        X : Integer;
        procedure Bigsub is
                A, B, C : Integer;
                procedure Sub1 is
                        A, D : Integer;
                begin
                        A := B + C;
                        …
                end;
                procedure Sub2(X: Integer) is
                        B, E : Integer;
                        procedure Sub3 is
                                C, E : Integer;
                        begin
                                …
                                Sub1;
                                …
                                E := B+A;
                        end;
                begin
                        …
                        Sub3;
                        …
                        A := D + E;
                end;
        begin
                …
                Sub2(7);
                …
        end;
begin
        …
        Bigsub;
        …
end;
```