Chương 5: Thực hiện chương trình con

Giảng viên: Ph.D Nguyễn Văn Hòa

Khoa KT-CN-MT – ĐH An Giang

Định nghĩa

- Trong NNLT, tác vụ gọi “call” và trả về (return) của chương trình con được gọi chung là liên kết chương trình con “*subprogram linkage*”

Nội dung chính của chương

- Giới thiệu chung về ngữ nghĩa của Call và Return
- Thực hiện chương trình con đơn giản
- Thực hiện chương trình con với biến cục bộ động Stack
- Chương trình con lòng nhau (nested Subprograms)
- Khối (Blocks)
- Cài đặt phạm vi động

Ngữ nghĩa của việc gọi (call) và trả về (return)

- Một số tác vụ cần thiết cho việc gọi chương trình con
  - Cơ chế truyền các tham số (truyền tham trị, truyền quy chiếu, truyền kết quả, ...)
  - Các biến cục bộ là ***static*** hay ***not static***
  - Lưu lại trạng thái hiện hành (execution status) của chương trình gọi CTC
  - Chuyển quyền điều khiển cho CTC
  - Cung cấp các truy xuất đến các biến không cục bộ

Thực hiện CTC đơn giản: ***Call***

- Chương trình con đơn giản “simple”
  - Không lòng nhau và các biến là tĩnh (***static)***
- Các tác vụ có cần thiết
  - Lưu hiện trạng thực thị của chương trình gọi “caller”
  - Thực hiện tiến trình truyền tham số
  - Chuyển địa chỉ trả về cho chương trình con “callee”
  - Chuyển quyền điều khiển cho chương trình con “callee”

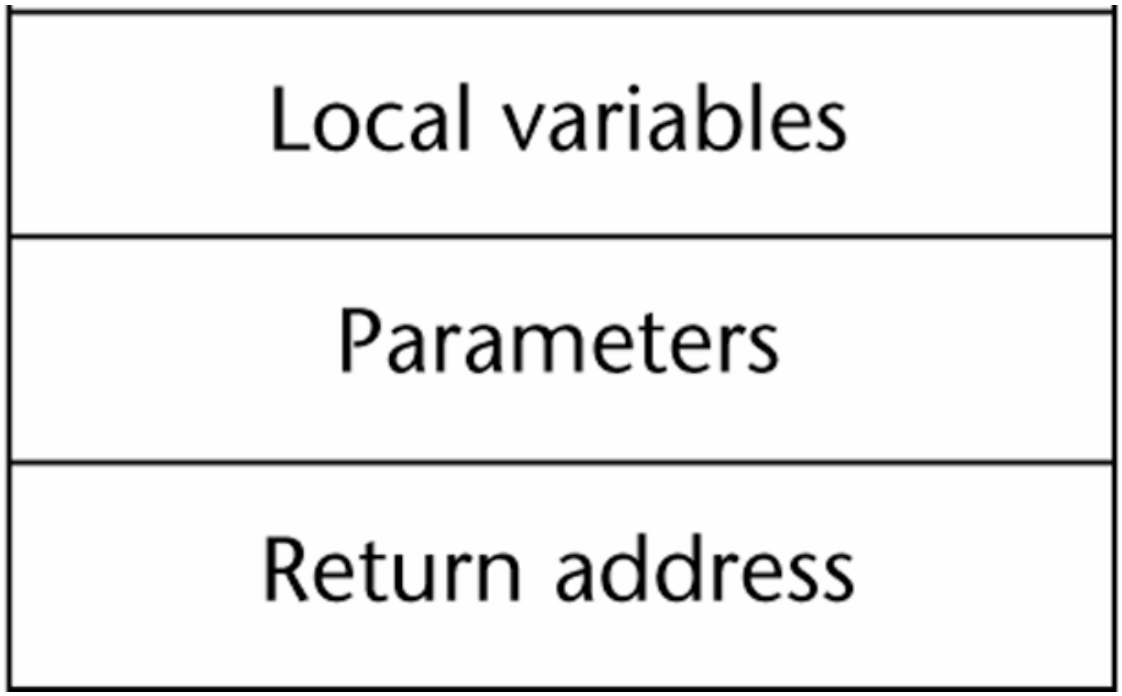
Thực hiện CTC đơn giản: ***Return***

- Nếu sử dụng truyền trị-kết quả, thì di chuyển giá trị hiện hành của các tham số hình thức đến từng tham số thực tương ứng
- Nếu là hàm, di chuyển giá trị của hàm đến vị trí mà caller có thể lấy được
- Phục hồi lại trạng thái thực thi của caller
- Trả quyền điều khiển lại cho caller

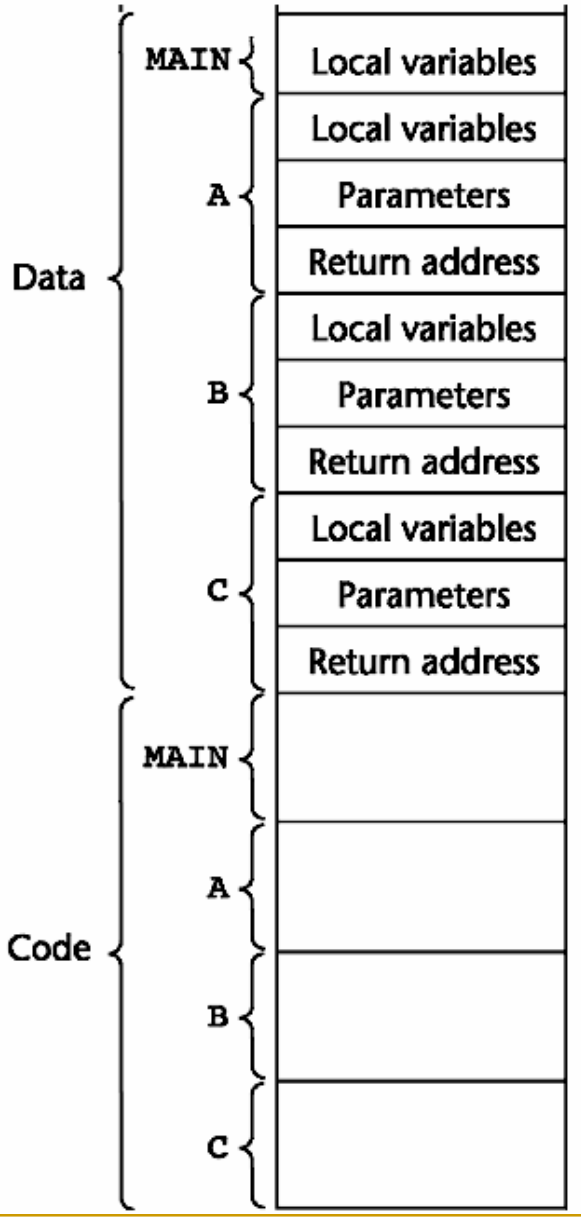
Thực hiện CTC đơn giản: ***Parts***

- Có 2 phần phân biệt: phần code thực và phần noncode (biến cục bộ và dữ liệu có thể bị thay đổi)
- Định dạng, hoặc layout, của phần noncode của một chương trình thực thi con được gọi là bản hoạt động (*activation)*
- Một thể hiện (instance) bản hoạt động là mẫu cụ thể của bản hoạt động (bao gồm dữ liệu hoạt động của chương trình con)

Bản hoạt động của chương trình con đơn giản



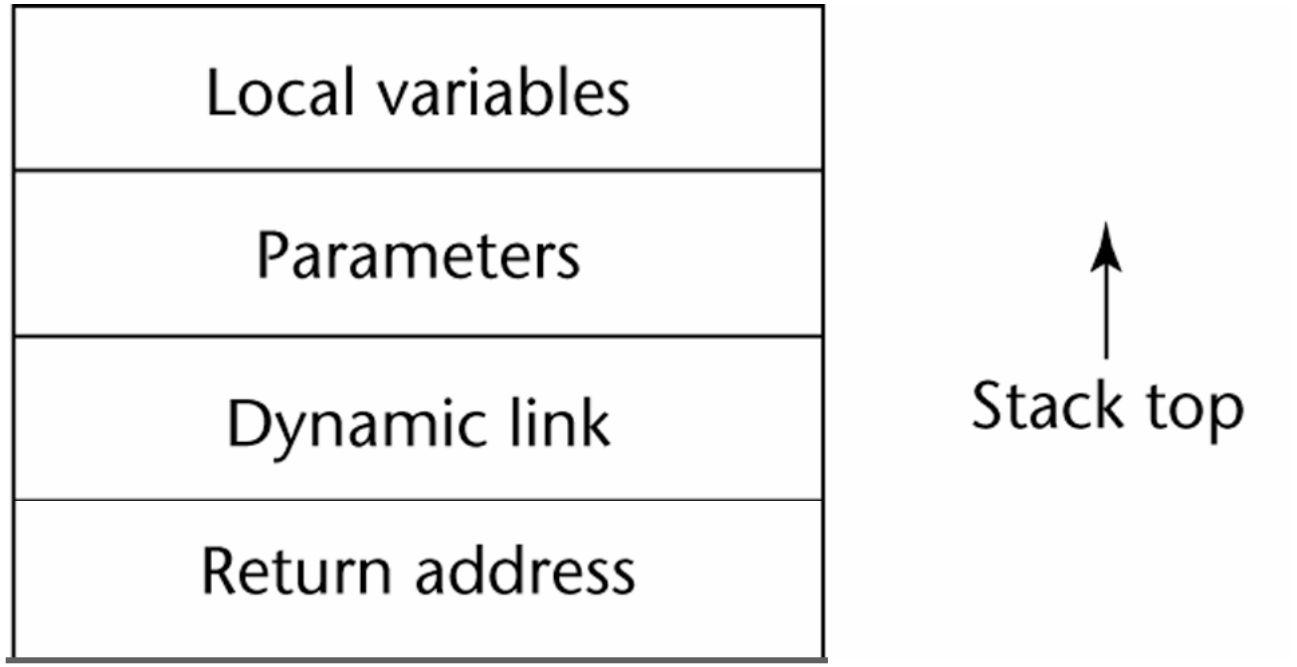
Code và bản hoạt động của chương trình với chương trình con đơn giản



Cài đặt CTC với biến cục bộ động Stack

- Bản hoạt động phức tạp
  - Trình biên dịch phải sinh code để cấp phát và giải phóng một cách tường minh cho các tham số cục bộ
  - Phải hỗ trợ đệ qui “recursion” (tạo đồng thời nhiều thể hiện bản hoạt động của một chương trình con)
  - Đệ qui yêu cầu nhiều thể hiện của bản hoạt động, mỗi thể hiện của bản hoạt động tương ứng với 1 một lần gọi đệ qui
  - Mỗi thể hiện cầu 1 bản copy các tham số hình thức, các biến cục bộ cấp phát động và địa chỉ trả về

Bản hoạt động của NN với biến cục bộ động Stack

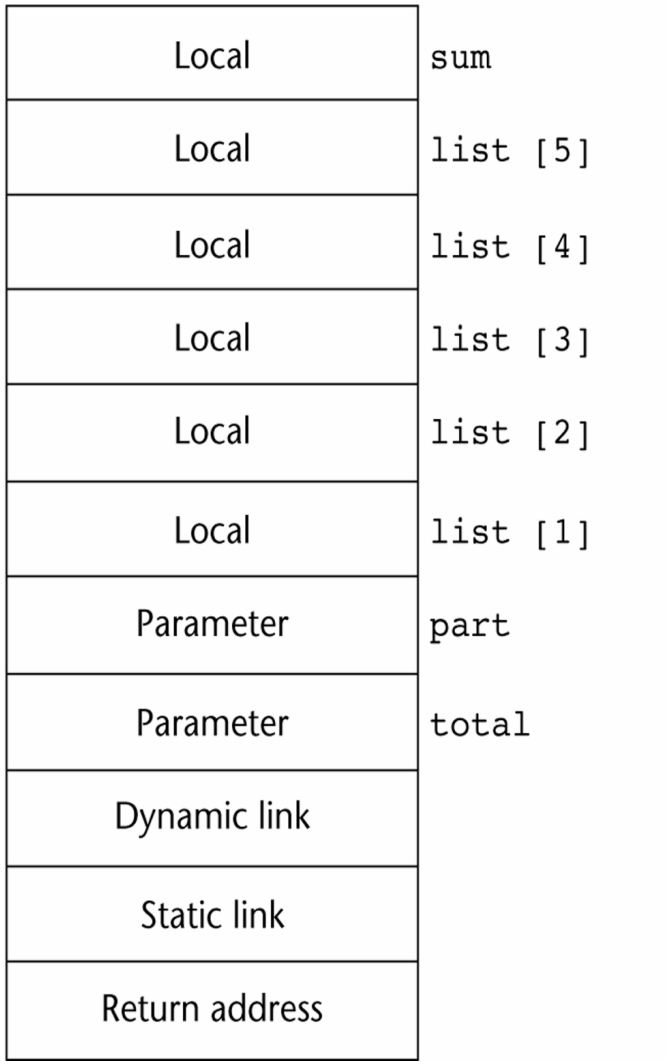


Cài đặt CTC với biến cục bộ động
Stack : Bản hoạt động

- Định dạng của bản hoạt động là tĩnh, nhưng kích thước của nó có thể động
- *Liên kết động (dynamic link)* trỏ đến đỉnh của bản hoạt động của caller
- Bản hoạt động được xác định ra một cách động khi gọi chương trình con
- Run-time stack

Ví dụ: Hàm C

```
void sub(float total, int part)
{
  int list[5];
  float sum;
  …
}
```



Ví dụ không đệ qui

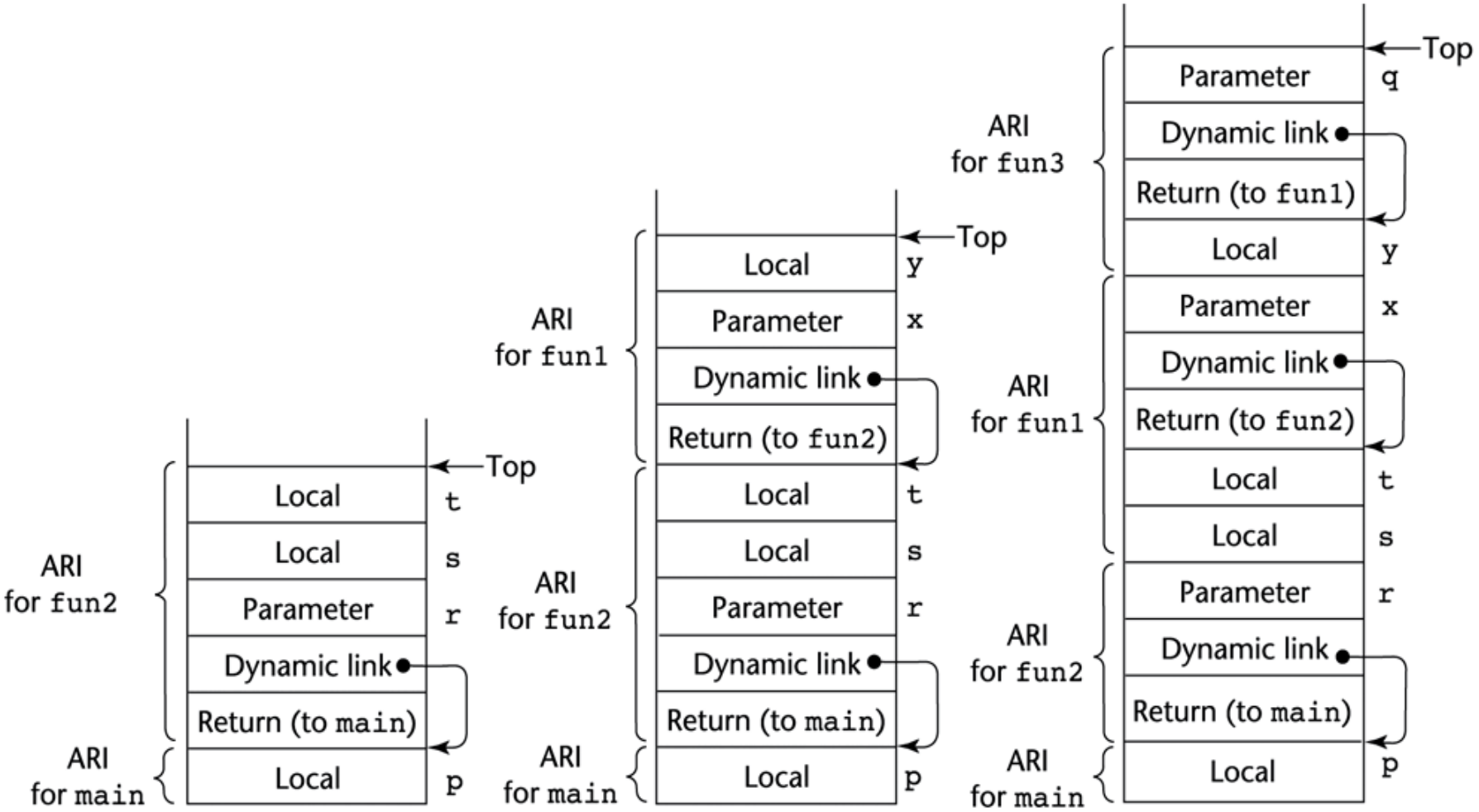
```
void fun1(int x) {
   int y;
   ...          ← 2
   fun3(y);
}
void fun2(float r) {
   int s, t;
   ...          ← 1
   fun1(s);
}
void fun3(int q) {
   ...          ← 3
}
void main() {
   float p;
   ...
   fun2(p);
}
```

Hàm main gọi fun2

Hàm fun2 gọi fun1

Hàm fun1 gọi fun3

Ví dụ không đệ qui



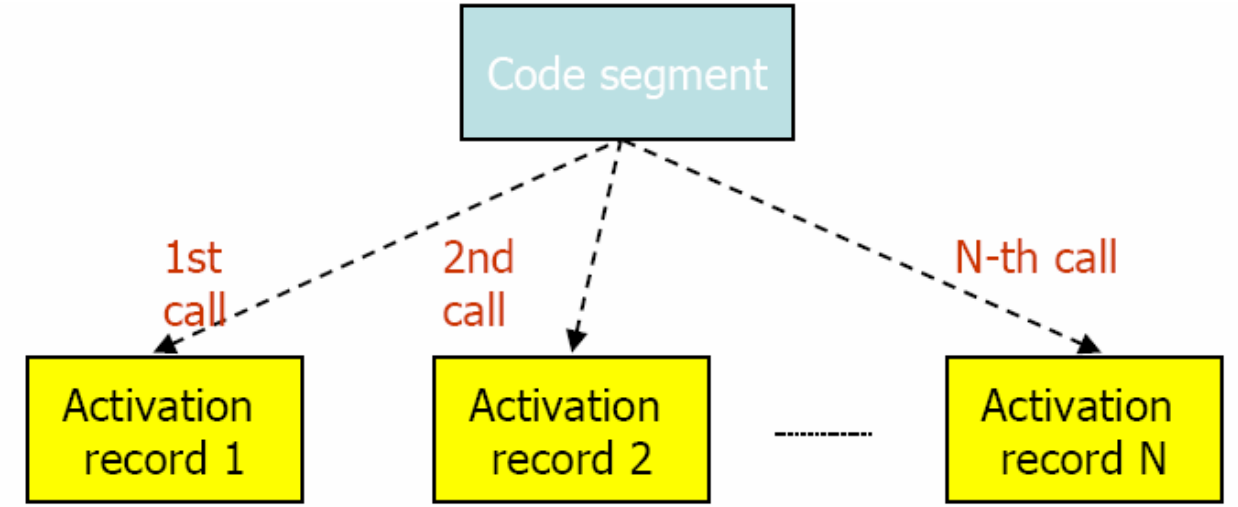
ARI = activation record instance

Dynamic Chain và Local Offset

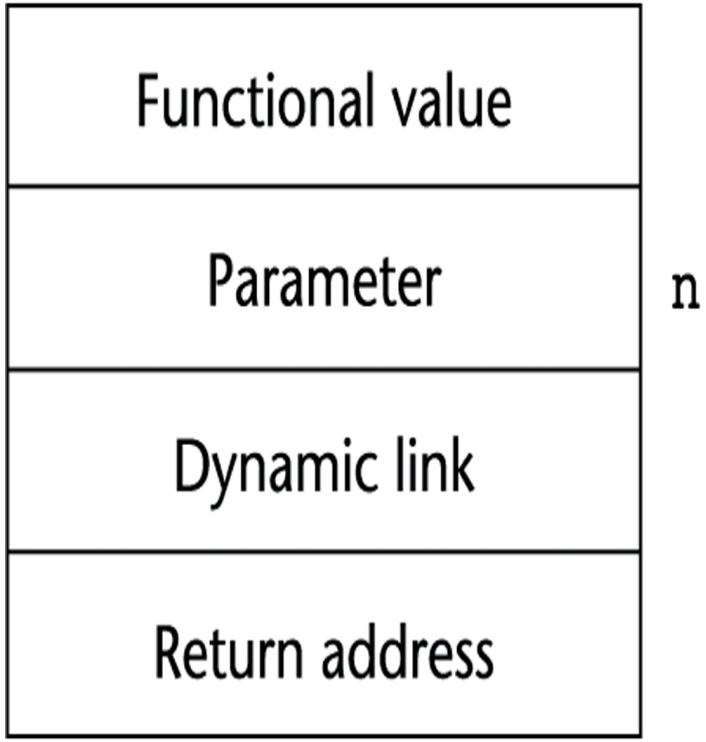
- Tập hợp các *dynamic links* trong stack tại một thời điểm nào đó được gọi là *dynamic chain*, hoặc *call chain*
- Các biến cục bộ có thể được truy xuất thông qua các offset của nó trong bản hoạt động. *Offset* này được gọi là *local_offset*
- *Local_offset* của một biến cục bộ có thể được xác định bởi trình biên dịch ngay thời điểm dịch hoặc thời điểm thực thi

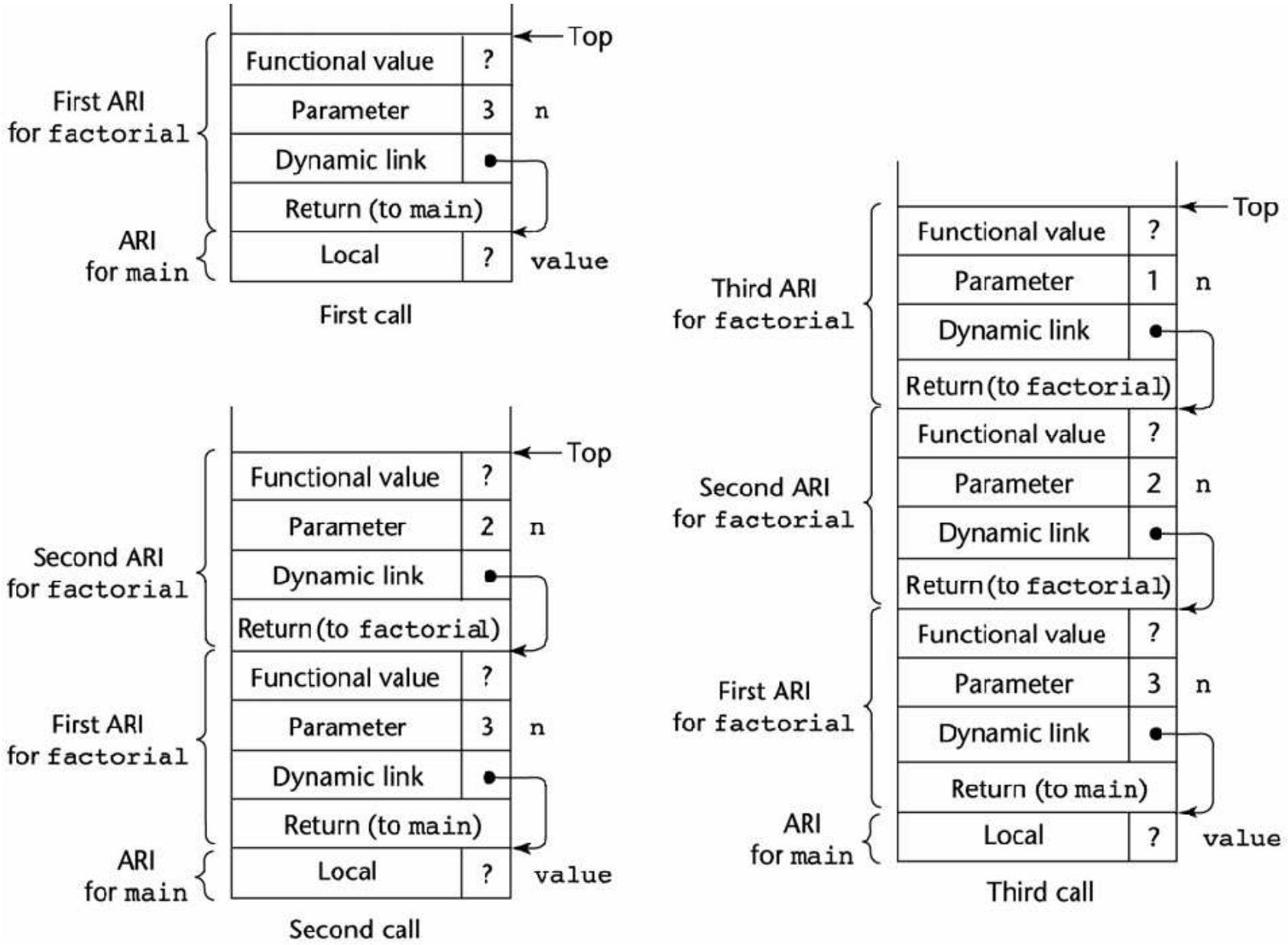
Ví dụ với đệ qui

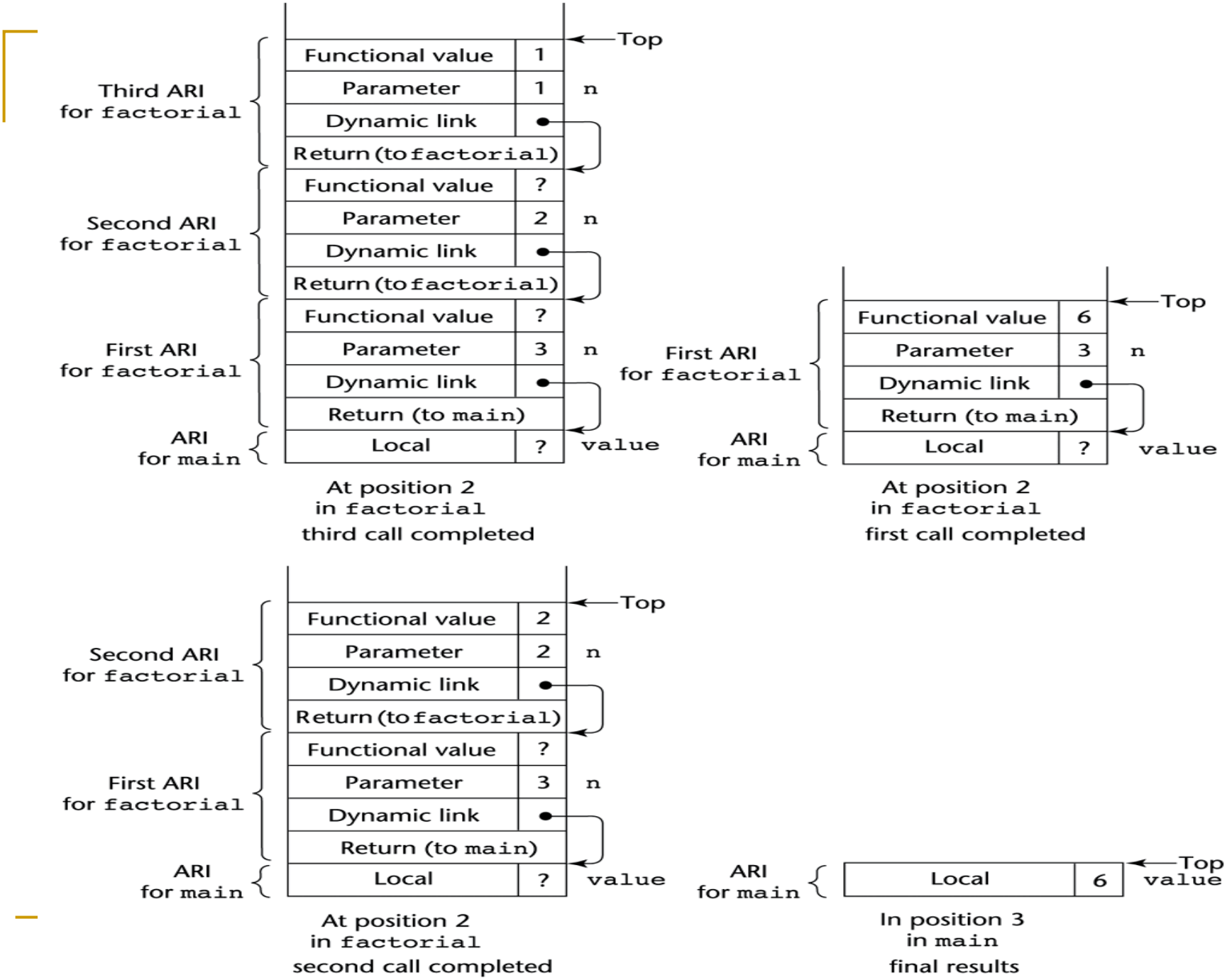
```
int factorial (int n) {
    <------------------------------1
  if (n <= 1) return 1;
  else return (n * factorial(n - 1));
    <------------------------------2
}
void main() {
  int value;
  value = factorial(3);
    <------------------------------3
}
```



Bản hoạt động của **`factorial`**




Top
Functional value
?
Parameter
3
n
Dynamic link
Return (to main)
Local
?
value
First ARI for factorial
ARI for main
First call
Top
Functional value
?
Parameter
2
n
Dynamic link
Return (to factorial)
Functional value
?
Parameter
3
n
Dynamic link
Return (to main)
Local
?
value
Second ARI for factorial
First ARI for factorial
ARI for main
Second call
Top
Functional value
?
Parameter
1
n
Dynamic link
Return (to factorial)
Functional value
?
Parameter
2
n
Dynamic link
Return (to factorial)
Functional value
?
Parameter
3
n
Dynamic link
Return (to main)
Local
?
value
Third ARI for factorial
Second ARI for factorial
First ARI for factorial
ARI for main
Third call


Top
Functional value
1
Parameter
1
n
Dynamic link
Return (to factorial)
Functional value
?
Parameter
2
n
Dynamic link
Return (to factorial)
Functional value
?
Parameter
3
n
Dynamic link
Return (to main)
Local
?
value
Third ARI for factorial
Second ARI for factorial
First ARI for factorial
ARI for main
At position 2 in factorial third call completed
Top
Functional value
6
Parameter
3
n
Dynamic link
Return (to main)
Local
?
value
First ARI for factorial
ARI for main
At position 2 in factorial first call completed
Top
Functional value
2
Parameter
2
n
Dynamic link
Return (to factorial)
Functional value
?
Parameter
3
n
Dynamic link
Return (to main)
Local
?
value
Second ARI for factorial
First ARI for factorial
ARI for main
At position 2 in factorial second call completed
Top
Local
6
value
ARI for main
In position 3 in main final results
ARI = activation record instance

Chương trình con lòng nhau

- Vài NNLT với phạm vi tĩnh (e.g., Fortran 95, Ada, JavaScript) dùng các biến cục bộ động stack và cho phép các chương trình con lòng vào nhau
- Các biến bên trong các bản hoạt động trong stack có thể được truy xuất một cách không cục bộ
- Hai bước để tham chiếu các biến không cục bộ:
  - Tìm kiếm thể hiện bản hoạt động tương ứng
  - Xác định offset của biến trong thể hiện đó

Định vị các tham chiếu không cục bộ

- Tìm *Offset*: khá dễ dàng
- Tìm thể hiện bản hoạt động tương ứng
  - Để đảm bảo có thể tham chiếu đến các biến không cục bộ thì các biến này phải được cấp phát trong các thể hiện bản hoạt động trong stack

Phạm vi tĩnh

- *Static chain* là tập hợp các *static links* liên kết một vài bản hoạt động trong stack
- *Static chain* là cách cài đặt phổ biến trong các NNLT hỗ trợ CTC lòng nhau
- Liên kết tĩnh (*static link*) trong một thể hiện bản hoạt động nào đó của 1 CTC là 1 pointer trỏ đến phần dưới cùng của thể hiện bản hoạt động cha, liên kết này dùng để truy xuất các biến không cục bộ
- Chuỗi tĩnh (*static chain*) của một thể hiện bản hoạt động nào đó đều phải kết nối với thể hiện bản hoạt động của CTC tổ tiên (ancestors)

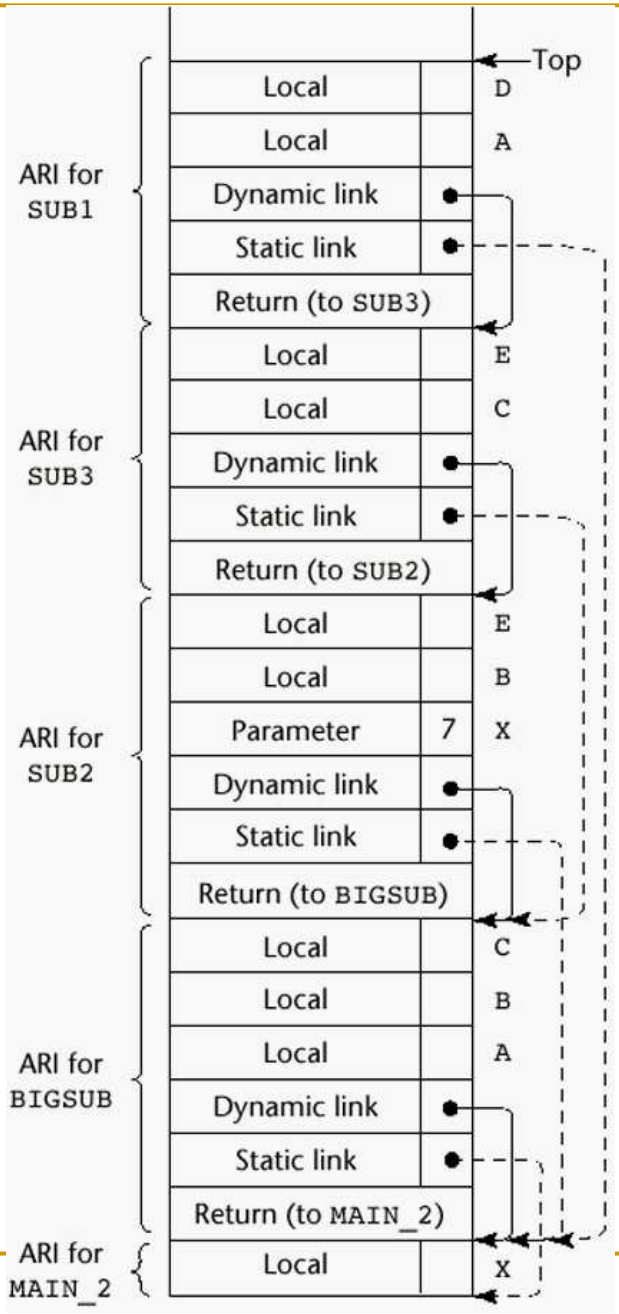
VD chương trình Pascal

```
program MAIN_2;
  var X : integer;
  procedure BIGSUB;
    var A, B, C : integer;
    procedure SUB1;
      var A, D : integer;
      begin { SUB1 }
      A := B + C;  <------------------------1
      end;  { SUB1 }
    procedure SUB2(X : integer);
      var B, E : integer;
      procedure SUB3;
        var C, E : integer;
        begin { SUB3 }
        SUB1;
        E := B + A:   <---------------------2
        end; { SUB3 }
      begin { SUB2 }
      SUB3;
      A := D + E;  <------------------------3
      end; { SUB2 }
    begin { BIGSUB }
    SUB2(7);
    end; { BIGSUB }
  begin
  BIGSUB;
  end; { MAIN_2 }
```

Stack ở vi trí 1

Thứ tự gọi các hàm như sau:

- MAIN_2 calls BIGSUB
- BIGSUB calls SUB2
- SUB2 calls SUB3
- SUB3 calls SUB1



Khối (blocks)

- Khối là vùng hay phạm vi hoạt động các biến do người dùng chỉ định
- VD chương trình C

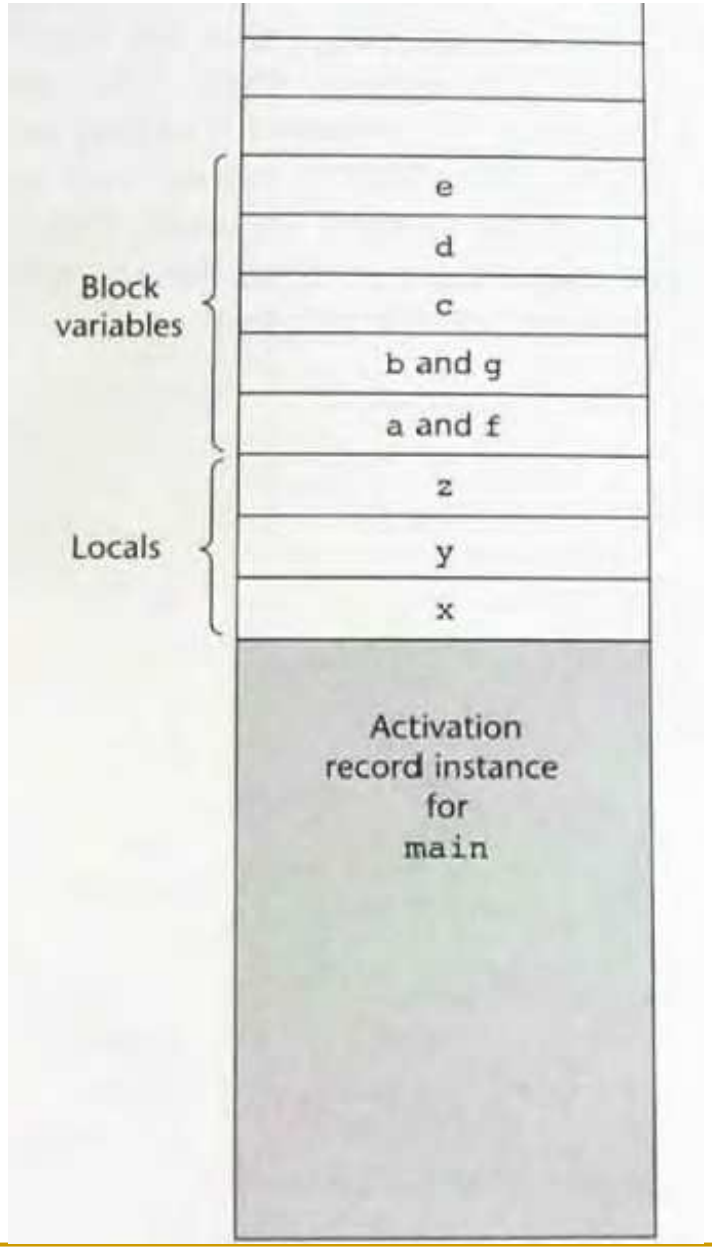
```
{int temp;
 temp = list [upper];
 list [upper] = list [lower];
 list [lower] = temp
}
```

- Thời gian tồn tại của `temp` trong VD trên bắt đầu ngay khi chương trình bắt đầu tiến vào khối
- Ưu điểm của việc dùng biến cục bộ như `temp` là biến temps không trở ngại nào nếu như có 1 biến khác cùng tên

Cài đặt khối

- Có hai cách:
  - Khối được xem như là các tham số của chương trình con mà nó được gọi tại ngay vị trí của nó
    - Mỗi khối có 1 thể hiện bản hoạt động, nó được tạo ra ngay thời điểm khối được thực thi
  - Vì kích thước lưu trữ cho 1 khối có thể xác định trong suốt thời gian thực hiện khối, kích thước này có thể được cấp phát sau các biến cục bộ trong bản hoạt động

```
void main() {
  int x, y, z;
  while ( ... ) {
    int a, b, c;
    ...
    while ( ... ) {
      int d, e;
      ...
    }
  }
  while ( ... ) {
    int f, g;
    ...
  }
  ...
}
```



Cài đặt phạm vi động

- *Truy xuất sâu (deep access)*: Các tham chiếu không cục bộ có thể được tìm thấy trong các bản hoạt động hiện hữu trong các liên kết động (dynamic chain)
- *Truy xuất cạn (shallow access)*: tập trung các biến cục bộ vào một nơi
  - Mỗi biến có 1 stack
  - Central table xác định các biến và chương trình con

VD chương trình

```
void sub3(){
  int x,z;
  x=u+z;
  …
}
void sub2(){
  int w,x;
  …
}
void sub1(){
  int v,w;
  …
}
void main(){
  int v,u;
  …
}
```

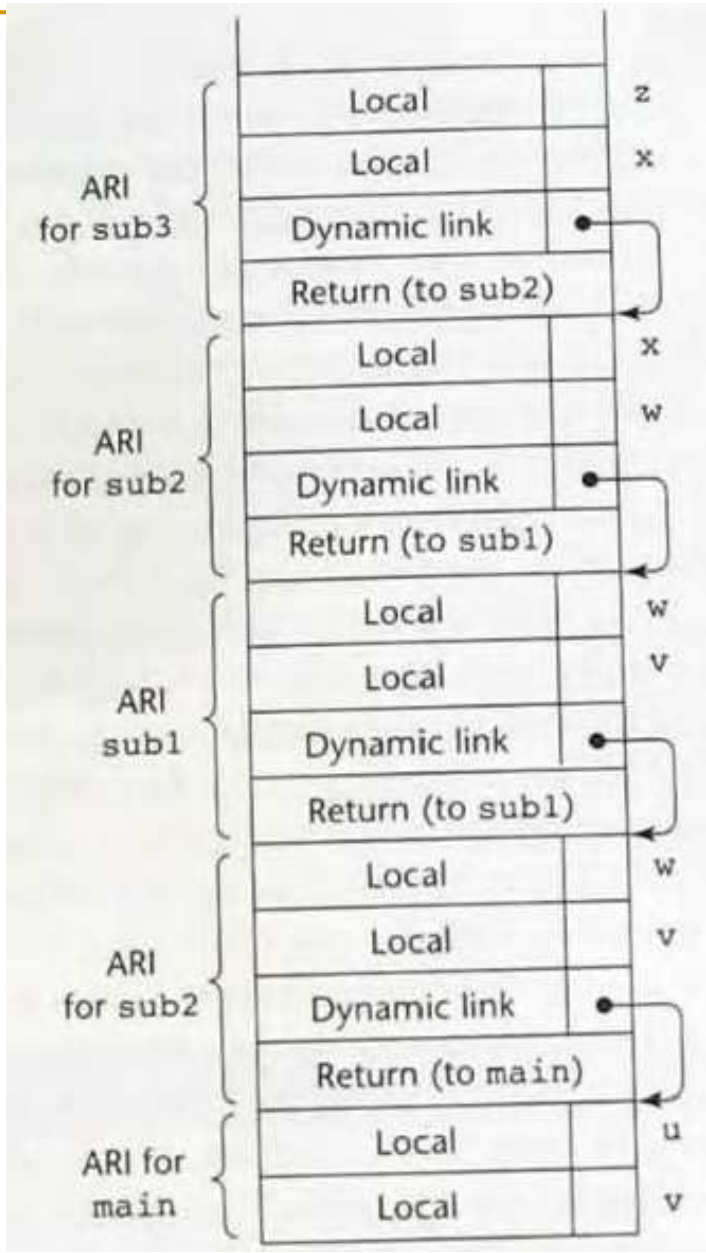
Minh họa Deep access

main gọi sub2

sub2 gọi sub1

sub1 gọi sub2

sub2 gọi sub3



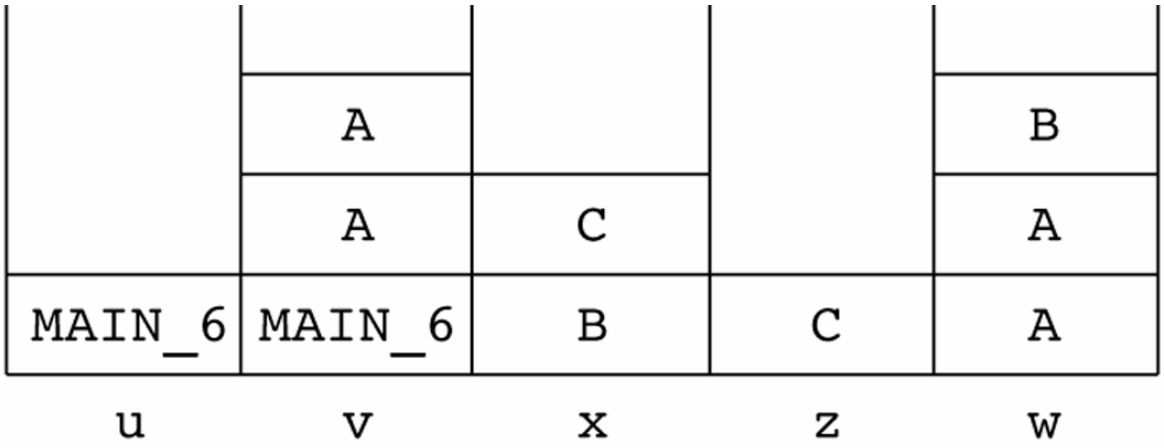
Minh họa Shallow Access

main gọi sub1 (A)

sub1 gọi sub1

sub1 gọi sub2 (B)

sub2 gọi sub3 (C)



(The names in the stack cells indicate the program units of the variable declaration.)