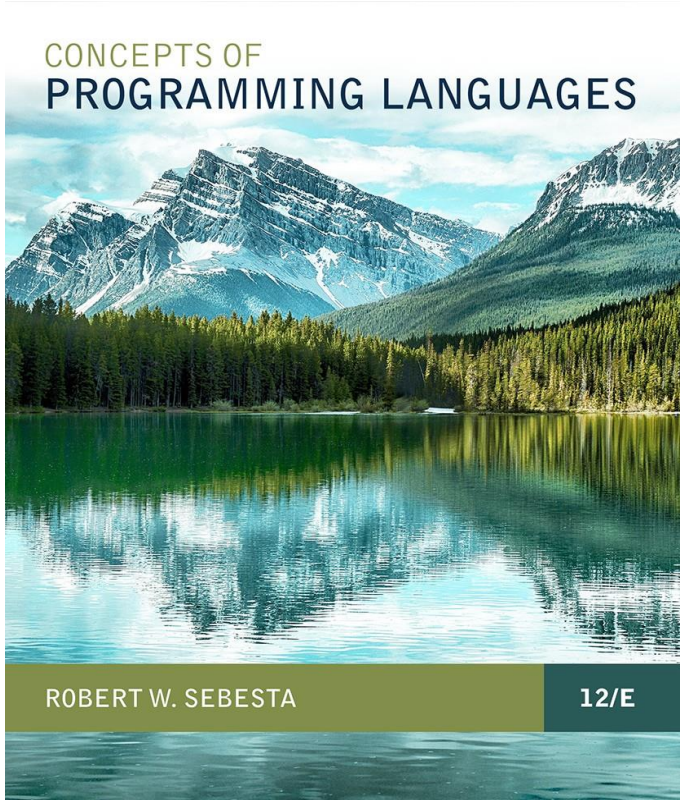
CAPÍTULO 10: IMPLEMENTACIÓN DE SUBPROGRAMAS

Estructura de los lenguajes

Dr. Christian von Lücken

Tópicos

- Semántica General de Invocaciones y Retornos
- Implementando Subprogramas simples
- Implementando Subprogramas con variables locales Stack-Dynamic
- Subprogramas anidados
- Bloques
- Implementando alcance Dinámico



AJUSTADO A LAS PRESENTACIONES DEL LIBRO "CONCEPTS OF PROGRAMMING LANGUAGES", ROBERT SEBESTA, 12/E. PEARSON. 2018. ISBN 0-321-49362-1

Semántica General de Invocaciones y Retornos

La invocación de subprograma y las operaciones de retorno de un lenguaje son conjuntamente denominadas el enlazado del subprograma *(subprogram linkage)*

Una invocación de subprograma posee varias acciones asociadas:

- Métodos de paso de párametros
- Asignación de variables stack-dynamic
- Guardar el estado de ejecución del programa invocador
- Transferencia del control de ejecución y organizar el retorno
- Si el subprograma es anidado, el acceso a las variables no locales debe ser manejado
- Si es una función, se debe colocar el valor de retorno de la función en un lugar accesible para el invocador

Semántica General de Invocaciones y Retornos

Semántica general del retorno:

- En modo in e inout los parámetros deben tener sus valores retornados
- Desasignación de locales stack-dynamic
- Restaurar el estatus de ejecución
- Retornar el control al llamador
-

Implementación de subprogramas “Simples”

Semántica de llamadas:

1. Guardar el estado de ejecución del llamador
2. Pasar los parámetros
3. Pasar la dirección de retorno al llamado
4. Transferir el control al llamado

Implementing "Simple" Subprograms (continued)

Return Semantics:

- If pass-by-value-result or out mode parameters are used, move the current values of those parameters to their corresponding actual parameters
- If it is a function, move the functional value to a place the caller can get it
- Restore the execution status of the caller
- Transfer control back to the caller

Required storage:

- Status information, parameters, return address, return value for functions, temporaries

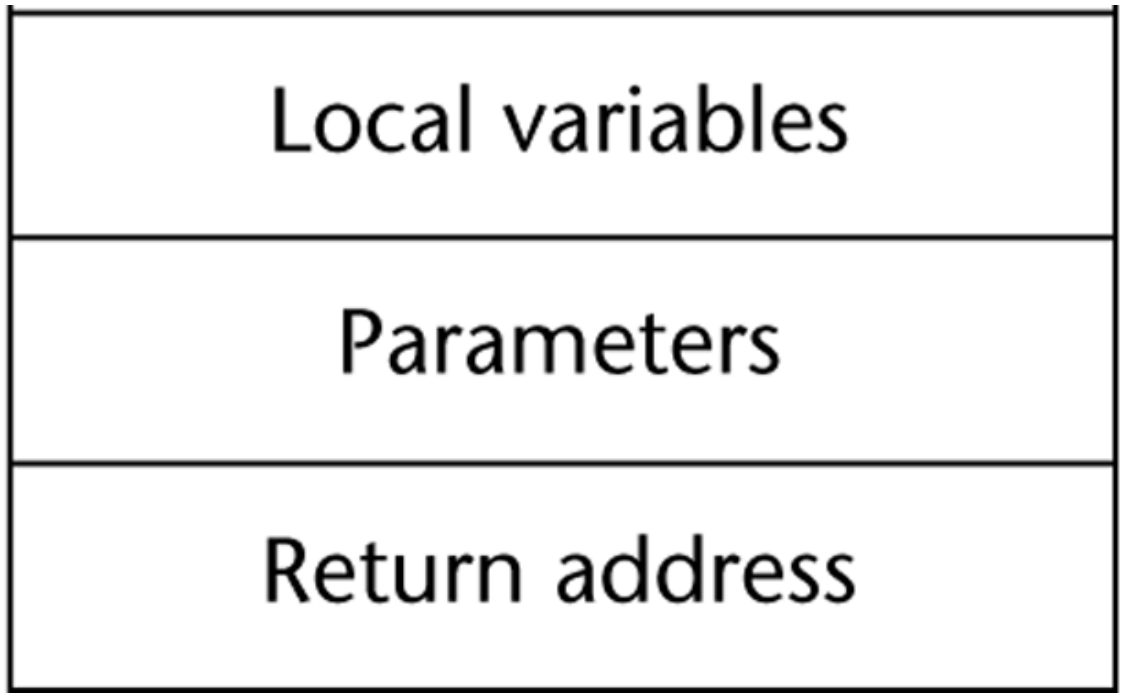
Implementación de subprogramas "Simples"

Partes Separadas: *él código actual y la parte no-código* (variables locales y datos que pueden ser modificados)

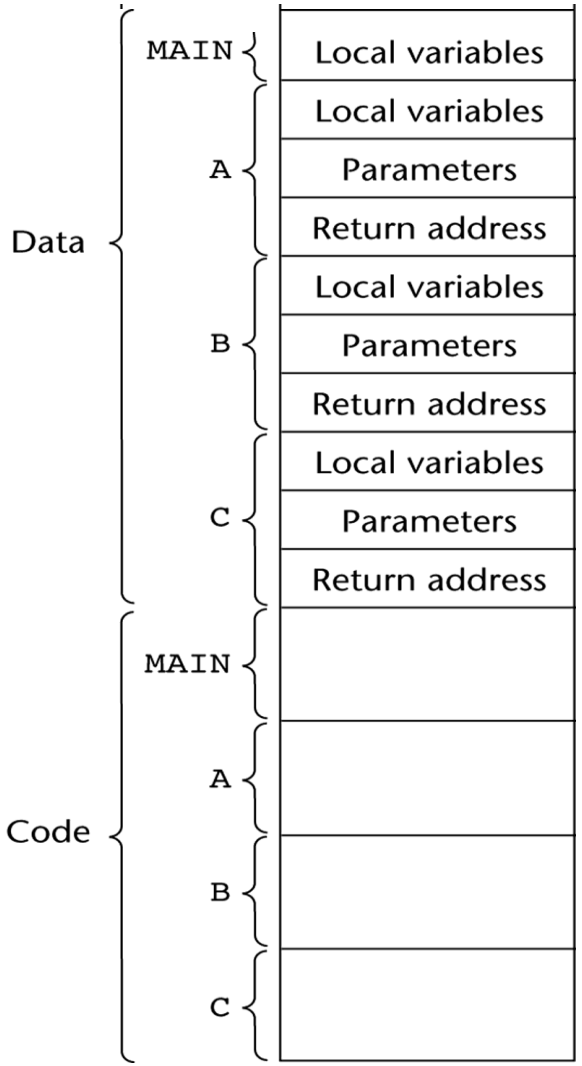
El formato, o diseño, de la parte no-código de un subprograma en ejecución se denomina ***registro de activación***

Una ***instancia de registro de activación*** es un ejemplo de un registro de activación (la colección de datos para una activación particular de un subprograma)

Resgistro de Activación para subprogramas simples



Código y Registros de Activaciones de un Programa con Subprogramas simples



Implementando Subprogramas con Variables Locales Dinámicas mediante pila: Registro de activación

El formato del registro de activación es estático, pero su tamaño puede ser dinámico

El enlace dinámico (*dynamic link*) apunta al tope de una instancia del registro de activación del llamador

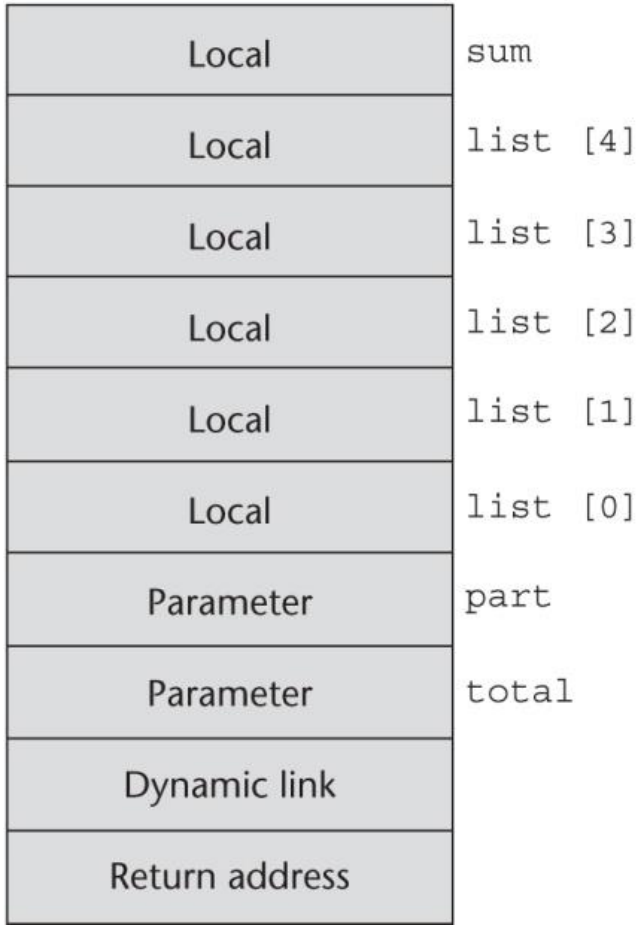
Una instancia del registro de activación se crea dinámicamente cuando se llama a un subprograma

Las instancias de los registro de activación residen en el run-time stack

El *Environment Pointer* (EP) debe ser mantenido por el sistema de tiempo de ejecución. Este siempre apunta a la base de la instancia del registro de activación de la unidad de programa actualmente en ejecución

Un ejemplo: Función de C

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



Revisión de las acciones de llamada/retorno

Acciones del llamador:

- Crear una instancia del registro de activación
- Guardar el estatus de ejecución de la unidad de programa actual
- Computar y pasar los parámetros
- Pasar la dirección de retorno al llamado
- Transferir el control al llamado

Acciones finales del llamado:

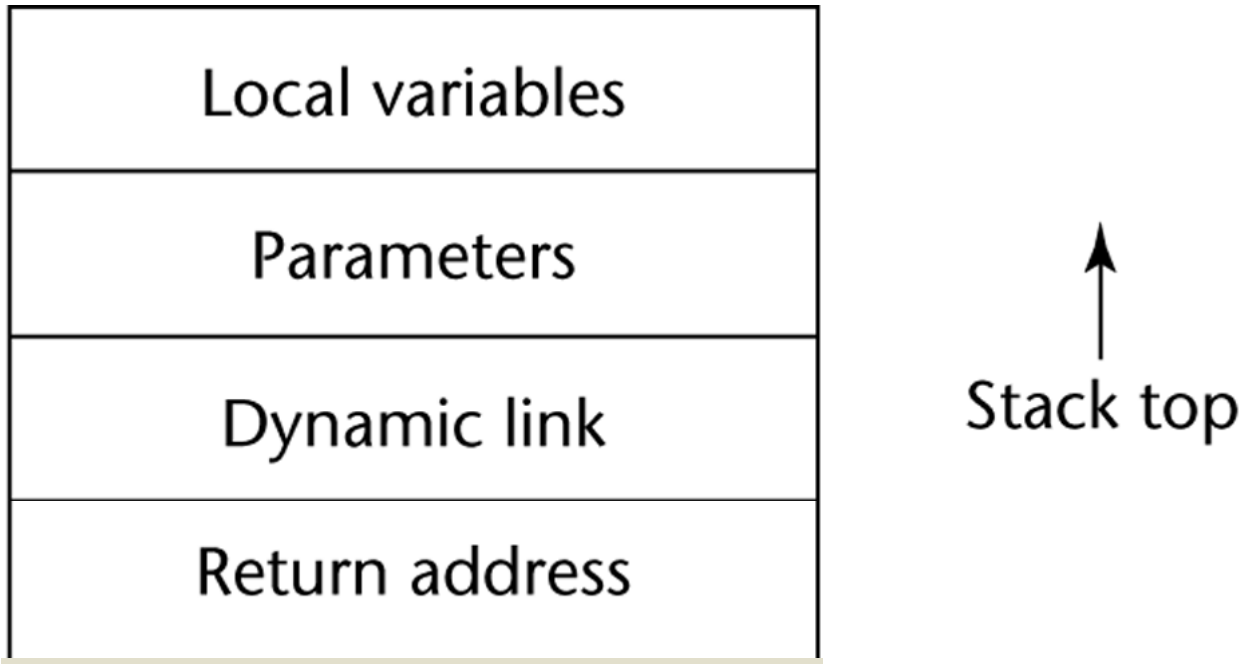
- Guardar el Viejo EP en el stack como el dynamic link y crear un nuevo valor
- Ubicar las variables locales

Revisión de las acciones de llamada/retorno (continuación)

Acciones finales del llamado:

- Si existen parámetros pass-by-value-result o out-mode, los valores actuales de estos parámetros se mueven a los parámetros actuales correspondientes
- Si el subprograma es una función, su valor se mueve a un lugar accesible por el llamador
- Restaurar el stack pointer asignándole el valor del EP-1 actual y hacer el EP al viejo dynamic link
- Restaurar el estatus de ejecución del llamador
- Transferir de nuevo el control al llamador

Registro de activación para un lenguaje con variables locales dinámicas

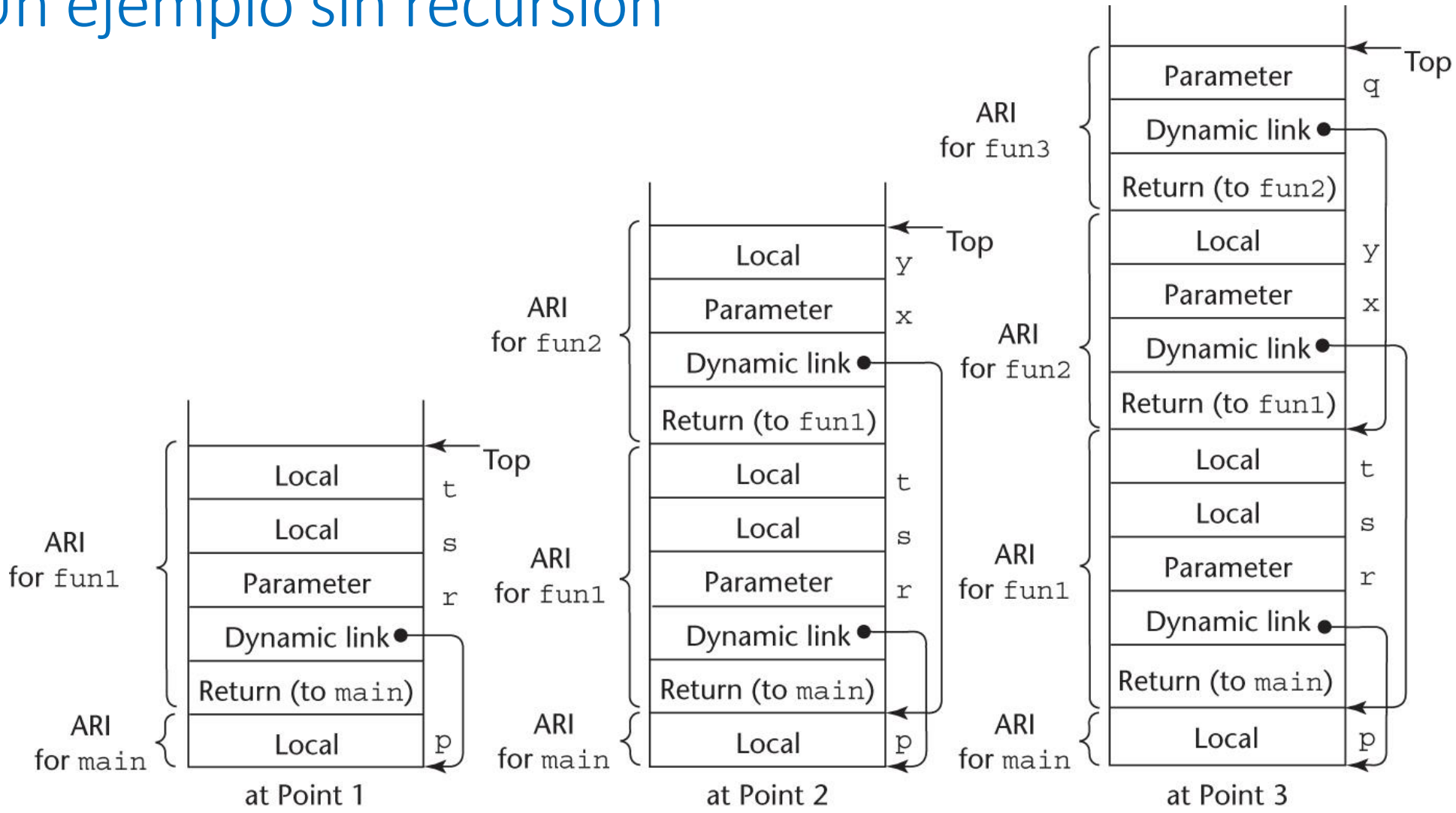


Un ejemplo sin recursión

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

`main` invoca `fun1`
`fun1` invoca `fun2`
`fun2` invoca `fun3`

Un ejemplo sin recursión



ARI = activation record instance

Encadenamiento dinámico y Desplazamiento Local (Dynamic Chain and Local Offset)

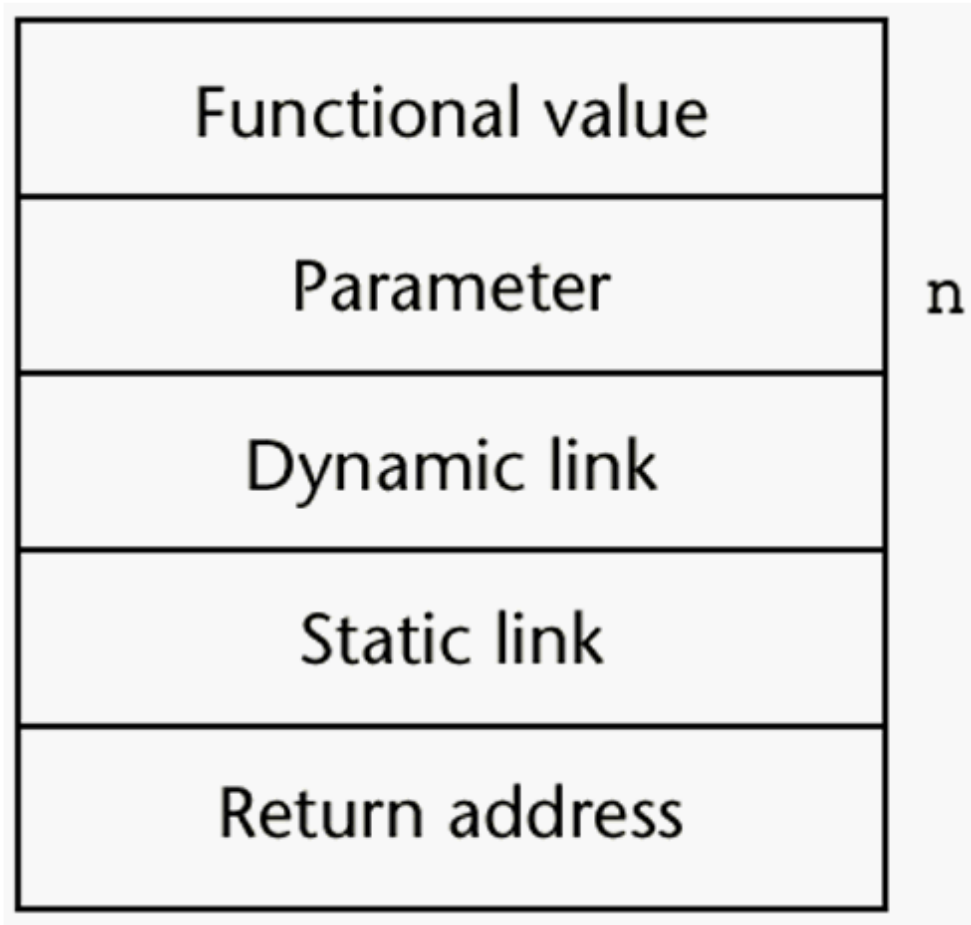
- La colección de enlaces dinámicos en la pila en cualquier momento se denomina encadenamiento dinámico (*dynamic chain*) o cadena de invocación
- Las variables locales pueden ser accesadas por su desplazamiento a partir del comienzo del registro de activación, cuya dirección esta en el EP. Este desplazamiento se denomina desplazamiento local (*local_offset*)
- El *local_offset* de una variable local puede ser determinado por el compilador en tiempo de compilación

Un ejemplo con Recursión

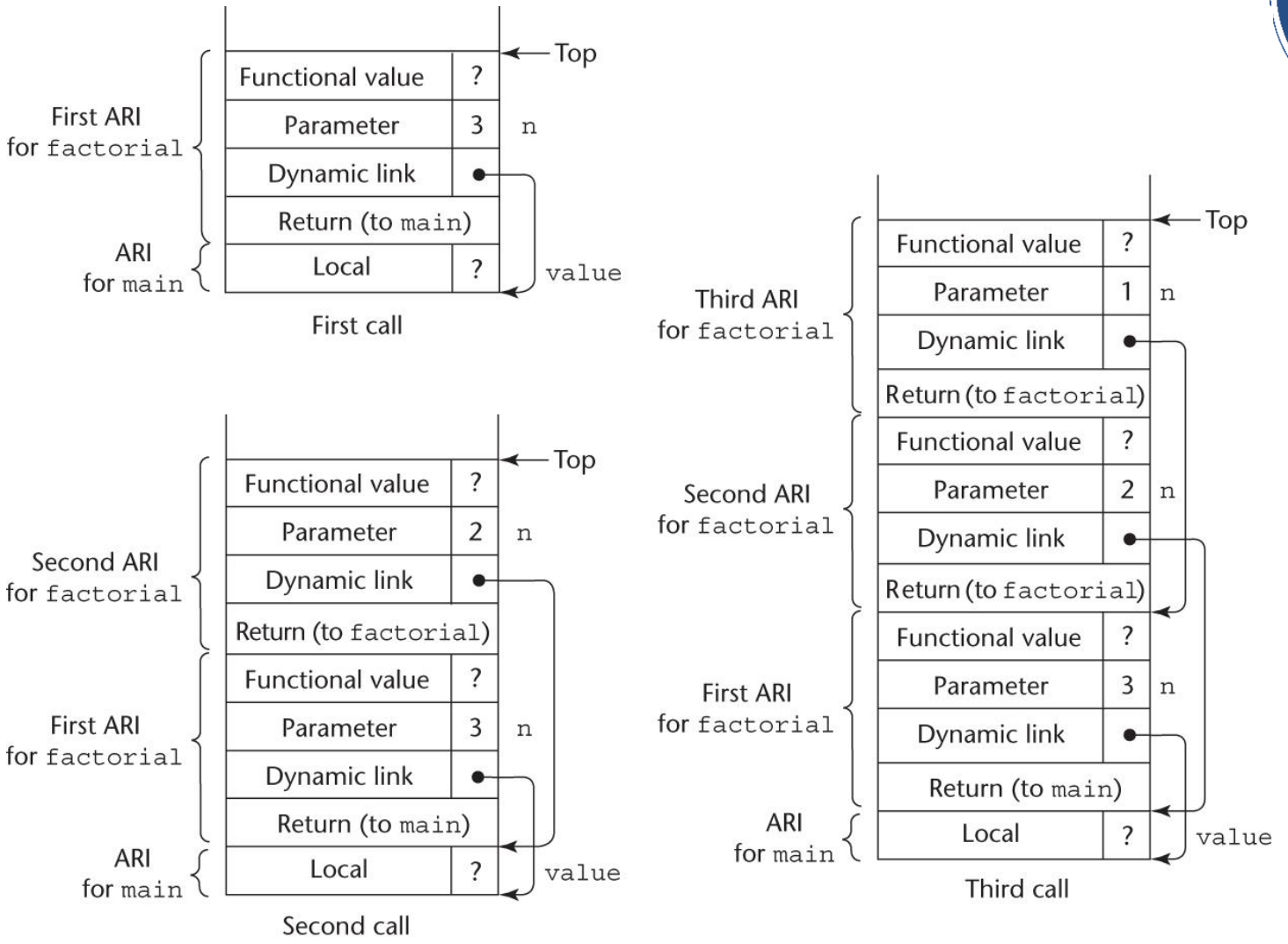
El registro de activación utilizado en el ejemplo anterior soporta recursión, por ejemplo

```
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
 }
```

Activation Record for **`factorial`**

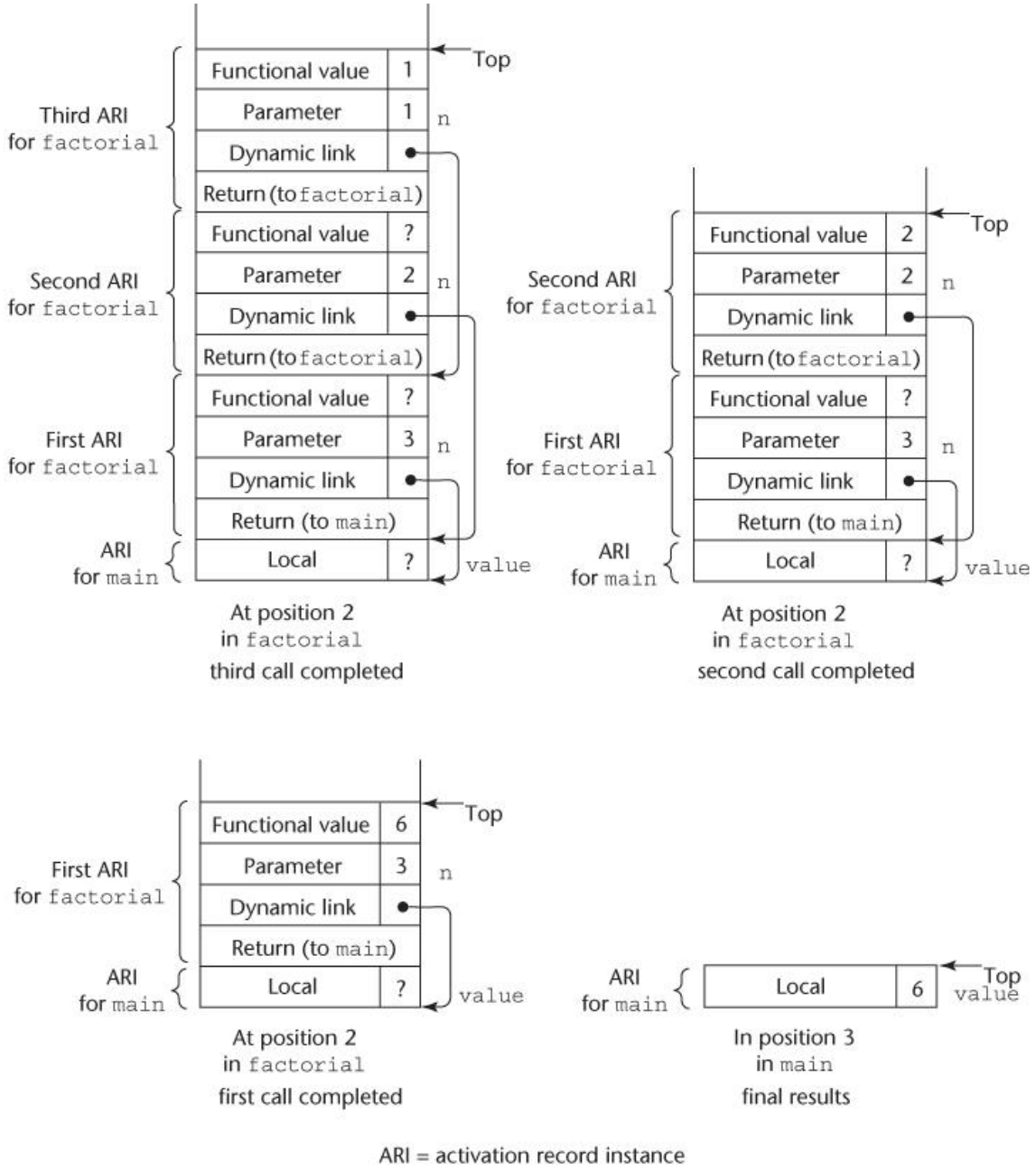


Stacks para las llamadas a `factorial`



ARI = activation record instance

Stacks para las llamadas a `factorial`



Subprogramas Anidados

- Algunos lenguajes no basados en C de alcances estáticos (como Fortran 95+, Ada, Python, JavaScript, Ruby, y Swift) utilizan variables locales dinámicas y permiten anidamiento de subprogramas
- Todas las variables que pueden ser de acceso no-local residen en alguna instancia de registro de activación en la pila
- Proceso de búsqueda de la referencia no local:
  1. Encontrar el registro de activación correcto
  2. Determinar el desplazamiento correcto dentro de la instancia del registro de activación

Encontrando referencias no locales

Encontrar el desplazamiento, es simple

Encontrar la instancia correcta de registro de activación

- Las reglas semánticas estáticas garantizan que todas las variables no locales que puedan ser referenciadas fueron alojadas en alguna instancia de registro de activación que debe estar en la pila cuando se realiza la referencia

Alcances estáticos (Static Scoping)

Un encadenamiento estático (*static chain*) es una cadena de enlaces estáticos que conectan ciertas instancias de registros de activación

El enlace estático en una instancia de registro de activación de un subprograma A apunta a la instancia del registro de activación del padre estático de A

La cadena estática de una instancia de registro de activación conecta a la instancia con todos sus ancestros estáticos

El acceso a variables no locales a más de 1 salto se implementa recorriendo la cadena

El *Static_depth* es un entero asociado con un alcance estático cuyo valor es la profundidad de anidamiento de tal alcance

Static Scoping (continuación)

El *chain_offset* o *nesting_depth* de una referencia no-local es la diferencia entre el static_depth de la referencia y la del scope donde este es declarado

Una referencia a una variable puede ser representada por el par:

(chain_offset, local_offset),

donde local_offset es el offset en el registro de activación de la variable que está siendo referenciada

Ejemplo en un programa JavaScript

```
function main(){
  var x;
  function bigsub() {
    var a, b, c;
    function sub1 {
      var a, d;
      function main(){
  var x;
  function bigsub() {
    var a, b, c;
    function sub1 {
      var a, d;
       a = b + c; <----------------------------1
      ...
    }  // end of sub1
    function sub2(x) {
      var b, e;
```

```
    function sub3() {
      var c, e;
      ...
       sub1();
       ...
       e = b + a; <----------------------------2
     } // end of sub3 ...
     sub3();
     ...
      a = d + e; <----------------------------3
    } // end of sub2
    ...
    sub2(7);
    ...
  } // end of bigsub
  ...
  bigsub();
  ...
} // end of main
```

Secuencia de llamadas para `main`

`main` llama `bigsub`

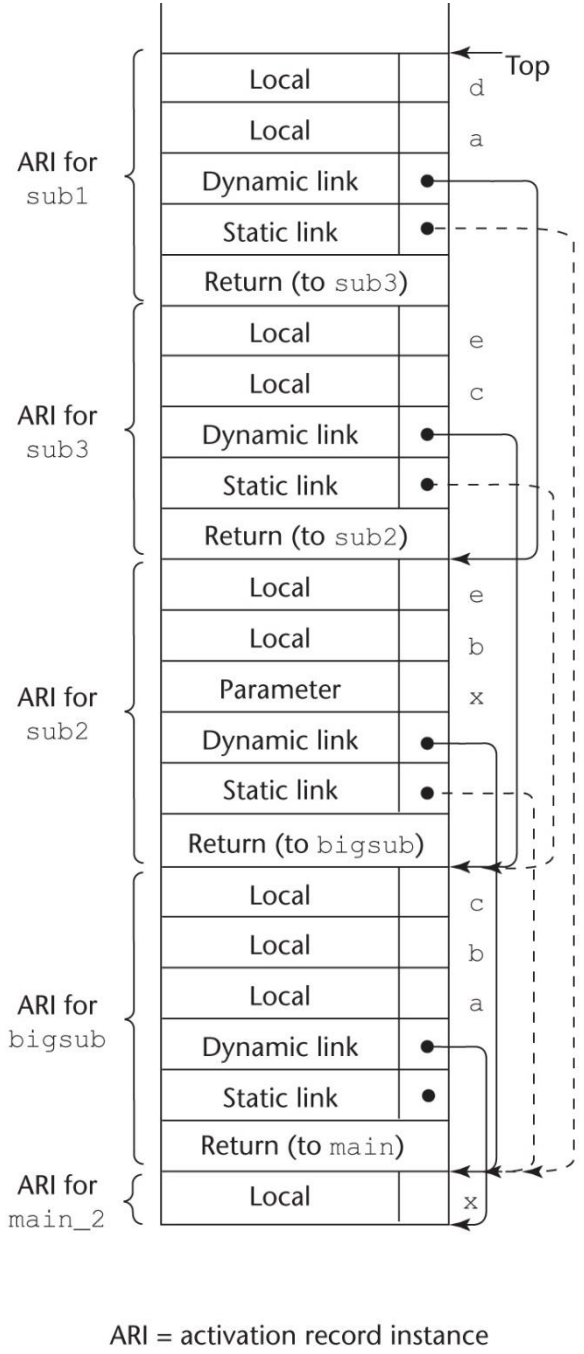
`bigsub` llama `sub2`

`sub2` llama `sub3`

`sub3` llama `sub1`

Contenido del Stack en la posición 1

```
function main(){
 var x;
 function bigsub() {
  var a, b, c;
  function sub1 {
   var a, d;
   function main(){
var x;
function bigsub() {
  var a, b, c;
  function sub1 {
   var a, d;
    a = b + c; <----------------------------1
   ...
  }  // end of sub1
  function sub2(x) {
   var b, e;
```



ARI = activation record instance

Mantenimiento de las cadenas estáticas

En la llamada,
- Debe construirse la instancia del registro de activación
- El dynamic link es solo el viejo puntero al tope de la pila
- El static link debe apuntar al ARI más reciente del padre estático

- Dos métodos :
    1. Buscar en la cadena dinámica
    2. Tratar cada llamada a subprograma y definiciones como referencias variables y definiciones

Evaluación de las cadenas estáticas

Problemas:
1. Una referencia no-local es lenta si es que el nivel de anidamiento es grande

2. Código de tiempo critico es difícil:
    a. Costos de las referencias no-locales son difíciles de determinar

    b. Los cambios en el código pueden cambiar la profundidad de anidamiento y por tanto el costo

Displays

Alternativa a encadenamiento estático

Los enlaces estáticos son almacenados en un array denominado *display*

El contenido del display en cualquier tiempo es la lista de direcciones de las instancias de registros de activación accesibles en el órden de invocaciones

Cualquier acceso a variable no local se encuentra a 1 salto en la cadena

Existe un costo de mantenimiento del display

Bloques

Los bloques son ambientes locales especificados por el usuario para ciertas variables

Un ejemplo en C

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

El tiempo de vida de `temp` en el ejemplo comienza cuando el control ingresa al bloque y termina al finalizar el bloque

Una ventaja es que usando bloques el nombre de la variable `temp` no interfiere con otra variable del mismo nombre

Implementando Bloques

Métodos:

1. Tratar bloques como subprogramas sin parámetros que son invocados siempre desde un mismo lugar.

   Cada bloque posee un registro de activación, una instancia se crea cada vez que se ejecuta el bloque

2. Puesto que el almacenamiento máximo requerido por el bloque puede ser determinado estáticamente, entonces ese espacio de almacenamiento puede ser colocado luego de las variables locales en el registro de activación

Implementando Ambientes Dinámicos

Deep Access: Las referencias no locales se encuentran buscando la instancia del registro de activación en la cadena de enlaces dinámicos:

La longitud de la cadena no puede determinarse estáticamente

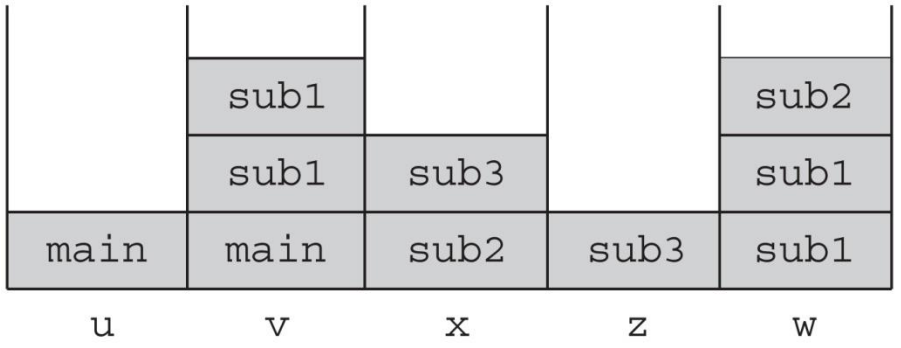
Cada instancia de registro de activación deber tener nombres de variables

Shallow Access : Coloca las variables en un repositorio central

- Una pila por cada nombre de variable
- Tabla central con una entrada para cada nombre de variable

Usando Shallow Access para implementar alcance dinámico

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



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

Resumen

La semántica de enlace de subprogramas requiere varias acciones para su implementación

Los subprogramas simples tienen acciones relativamente básicas

Lenguajes dinámicos mediante pila son más complejos (enlace dinámico)

Subprogramas con variables dinámicas mediante pila y subprogramas anidados poseen 2 componentes:

- Código Actual
- Registro de Activación

Resumen (cont.)

Las instancias de registro de activación contienen parámetros formales, variables locales, enlaces estático y dinámico, y la referencia de retorno, entre otros.

El encadenamiento estático es el método primario para implementar acceso a variables no locales en lenguajes de ambiente estático con subprogramas anidados, otra alternativa puede ser el display.

El acceso a variables no locales en lenguajes con ambiente dinámico puede ser implementado utilizando encadenamiento dinámico o a través del método de tabla central de variables.