Análisis Comparativo de Lenguajes

Teoría 8: *Control de subprogramas y control de datos*

Departamento de Informática

Facultad de Cs. Fco. Matemáticas y Naturales

Universidad Nacional de San Luis

San Luis - Argentina

Control de Subprogramas

Analizaremos la interacción entre subprogramas y cómo se pasan datos entre ellos.

La estructura call-return (llamada-retorno simple) de subprogramas es común a la mayoría de los lenguajes de programación. Es una estructura de control que permite visualizar a los programas como jerarquías y que puede ser explicada a partir de la regla de la copia:

"el efecto de una sentencia call (llamada) a un subprograma, es equivalente al que se obtendría substituyendo esta sentencia por el cuerpo del subprograma (con la subst. apropiada de parámetros e ident. en conflicto)

Supuestos de la regla de la copia:

1. Los subprogramas no pueden ser recursivos.
2. Se requiere llamadas explícitas (los manejadores de excepciones no).
3. Los subprogramas deben ejecutarse completamente en cada llamada (las corutinas no).
4. Transferencia inmediata del control en el punto de llamada (subprogramas planificados la difieren).
5. Secuencia de ejecución única (no ocurre con tareas o subprogramas concurrentes).

La llamada y retorno simple en subprogramas

Nos concentramos por ahora en el control de secuencia entre subprogramas.

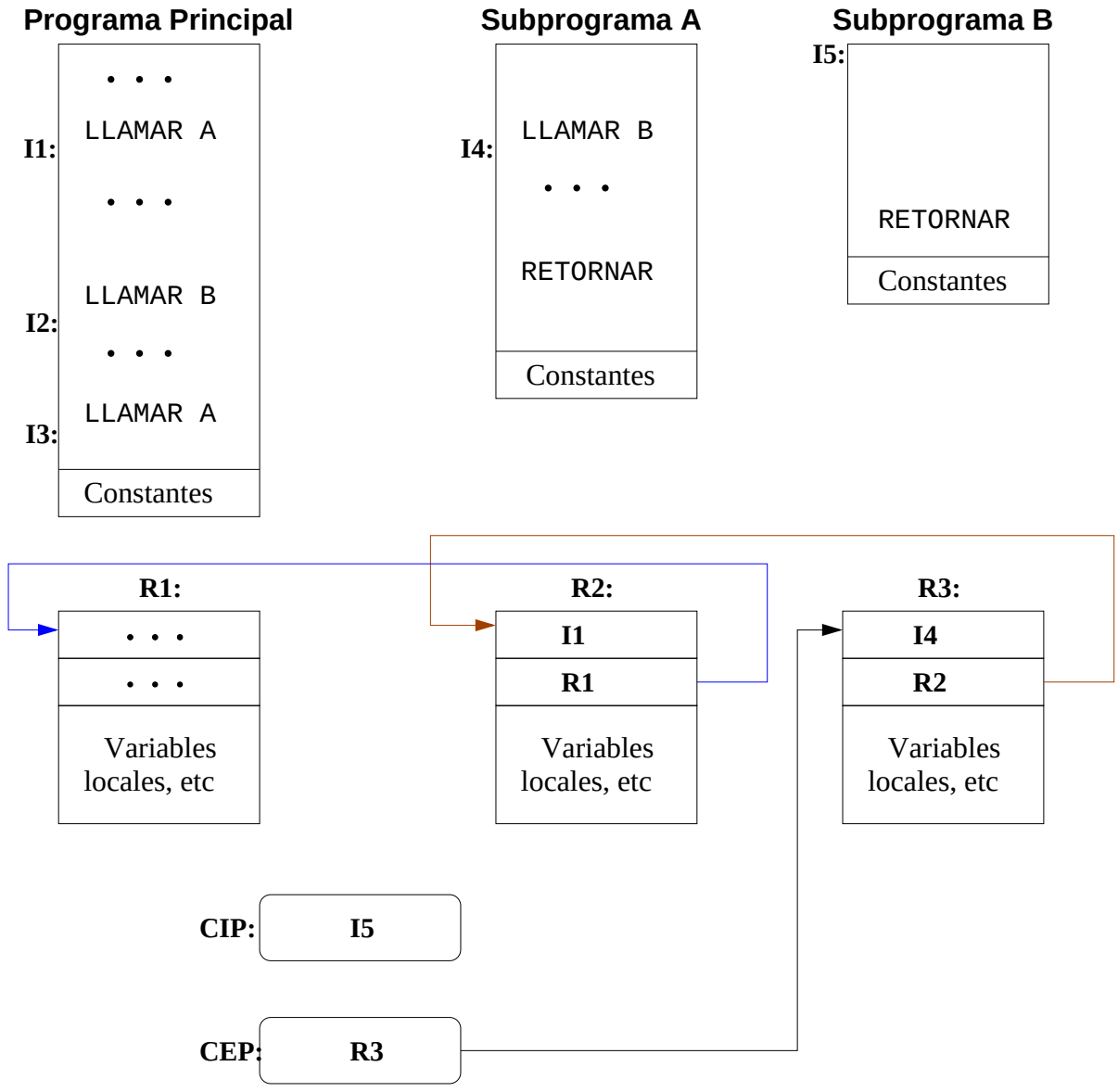
Para entender la implementación de esta estructura de control es bueno diferenciar:

- la definición del subprograma.
- la activación del subprograma.
  - el segmento de código
  - el registro de activación

Para mantener el punto en el que el programa se está ejecutando, se requieren dos variables punteros:

1. el CIP (Current Intruction Pointer): apunta a la instrucción que está siendo ejecutada (o la que va a ser ejecutada).
2. el CEP (Current Environment Pointer): apunta al registro de activación corriente.

Manteniendo la pista de ejecución



Subprogramas Recursivos

- Recursión es una de las estructuras de control de secuencia más importantes.
- Diferencia entre una llamada recursiva y una llamada común: la llamada recursiva crea una segunda activación del subprograma durante el tiempo de vida de la primera (extensible a k activaciones).
- Cada registro de activación contiene la pareja de punteros (pi,pe).
- El CEP y los valores de pe viejos forman una cadena, que une los R.A. del stack en el orden de su creación (cadena dinámica).
- El uso del stack es solo necesario si tenemos programas recursivos (se pueden plantear enfoques híbridos (tipo PL/I))

Control de datos (CD)

- Trata con la accesibilidad de los datos en diferentes puntos durante la ejecución del programa.
- El CD determina cómo acceder y proveer los datos a las operaciones y cómo guardar los resultados para poder ser usados como operandos en otra operación.
- El CD tiene que ver con el significado de los nombres de los operandos:

```
X:=Y+2
```

  - Y puede ser variable local o no local
  - Y puede ser parámetro formal
  - Y puede ser subprograma sin parámetros

Asociaciones y ambientes de referenciación

El CD trata con la ligadura de identificadores a objetos de datos y subprogramas, o asociación y se puede representar como un par $\langle id, OD \rangle$.

El ambiente de referenciación de un subprograma es el conjunto de asociaciones de identificadores que tiene disponibles para usar durante su ejecución.

Componentes del ambiente de referenciación de un subprograma

- *ambiente de referenciación local:* parámetros formales, variables locales, subprogramas definidos dentro del subprograma. Las asociaciones se crean al entrar al subprograma.
- *ambiente de referenciación no local:* conjunto de asociaciones que pueden usarse en un subprograma, pero no se crean a la entrada del mismo.
- *ambiente de referenciación global:* es parte del ambiente no local. Se crea al comienzo de la ejecución del programa.
- *ambiente de referenciación predefinido:* asociaciones definidas directamente en la definición del lenguaje, Ejemplo: `maxint`.

Ejemplo de ambientes de referenciación

```
program Principal
  var A, B, C: real;
  procedure Sub1 (A: real);
     var D: real;
     procedure Sub2 (C: real);
        var D: real;
        begin
        ...
        C := C + B;
        ...
        end;
  begin
  ...
  Sub2(B);
  ...
  end;
begin
...
Sub1(A);
...
end.
```

Ambiente de referenciación para Sub2

- Local: C, D
- No Local: A, Sub2 en Sub1
  B, Sub1 en Principal

Ambiente de referenciación para Sub1

- Local: A, D, Sub2
- No Local: B, C, Sub1 en Principal

Ambiente de referenciación para Principal

- Local: A, B, C, Sub1

Visibilidad y Alias

Visibilidad: una asociación para un identificador es visible dentro de un subprograma si es parte de su ambiente de referenciación. Una asociación que existe pero no forma parte del ambiente de referenciación del subprograma que se está ejecutando se dice que está oculta de ese subprograma.

Alias: Cuando un objeto de datos es visible a través de más de un nombre en el mismo ambiente de referenciación, cada uno de esos nombres se denomina *alias* del objeto de datos. Cuando un objeto de datos tiene múltiples nombres, pero un único nombre en cada ambiente de referenciación en el cual aparece, no existen problemas.

Problemas:

- para el programador: dos nombres distintos no significan objetos de datos distintos.
- para el implementador: puede interferir en el reordenamiento de operaciones o eliminación de operaciones innecesarias.

Ejemplo de Alias

```
program principal
var X:integer;
  procedure P (var Y,Z:integer);
    begin
     Y:=Y+1;
     Z:=X*2;       X,Y,Z son alias en este A.R.
    end
begin
...
P(X,X);
...
end.
```

Datos Compartidos en Subprogramas

Un objeto de datos que es local es usado por operaciones sólo dentro del ambiente de referenciación local. Sin embargo, los objetos de datos suelen compartirse entre varios subprogramas para que las operaciones de cada uno de los subprogramas puedan utilizar esos datos.

Un objeto de datos se puede transmitir como un **parámetro** o a través de **ambientes no locales**, la cual es una alternativa importante al uso de compartición directa a través del uso de parámetros.

Enfoques básicos para ambientes no locales:

1. Ambientes comunes explícitos.
2. Ambientes no locales implícitos basados en:

   Alcance Dinámico.

   Alcance Estático.
3. Herencia.

Parámetros y Transmisión de Parámetros

Los parámetros y resultados transmitidos de manera explícita constituyen el método alternativo principal para compartir objetos de datos entre subprogramas.

Parámetro Formal: es una clase particular de objetos de datos local dentro de un subprograma. La definición del subprograma enumera generalmente los nombres y declaraciones para los parámetros formales como parte de la sección de especificación.

Parámetro Actual: es un objeto de datos que se comparte con el subprograma de llamada. Un parámetro actual se representa en el punto de llamada del subprograma por medio de una expresión, llamada expresión de parámetro actual, que comúnmente tiene la misma forma que cualquier otra expresión del lenguaje.

Correspondencia parámetro formal con parámetro actual

Posicional: la correspondencia se establece apareando parámetros actuales y formales con respecto a sus posiciones respectivas en las listas de parámetros actuales y formales.

Por *nombre explícito*: el parámetro formal que se va a aparear con cada parámetro actual se puede nombrar de manera explícita en la sentencia de llamada. Argumentos de palabra clave en LISP.

Métodos para transmitir parámetros

Cuando un subprograma transfiere el control a otro subprograma, debe haber una asociación del parámetro actual del subprograma que llama con el parámetro formal del subprograma llamado.

Existen varios métodos para transmitir parámetros, de los cuales los más usados son:

Por valor: el valor del parámetro actual se pasa al parámetro formal.

Por dirección: el parámetro formal y el parámetro actual referencian a la misma dirección.

Por resultado: un parámetro transmitido por resultado se usa sólo para transmitir un resultado al regresar del subprograma invocado.

Por valor-resultado: el valor del parámetro actual se copia en el objeto de datos del parámetro formal en el momento de la llamada. Cuando el subprograma termina, el contenido final del objeto de datos del parámetro formal se copia en el parámetro actual.

Por nombre: los parámetros son transmitidos sin evaluar. Se reevaluan cada vez que se usan dentro del subprograma.

Métodos para transmitir parámetros

Ejemplo: Diferencias cuando X e Y se pasan por dirección y por nombre

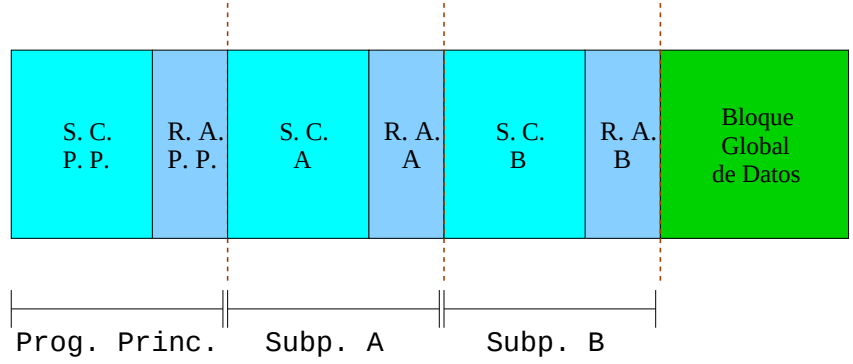
```
var a: array [1..3] of integer;
    i: integer;
procedure P (X,Y);
begin
  X := X + 1;
  Y := Y + 1;
end;
begin
  ...
  i := 2;
  P(i,a[i]);
end.
```

Ambientes comunes explícitos

Ambiente común: es un bloque de almacenamiento separado con un nombre que contiene un conjunto de O.D.s que se van a compartir entre un grupo de subprogramas.

Similar al ambiente local de un subprograma, pero sin formar parte de ningún subprograma (es una declaración externa a cualquier subprograma).

El acceso a los datos del ambiente común es similar al de un registro y el área de almacenamiento asignado para el mismo tiene existencia más allá de la ejecución de cualquier subprograma.



Variables explícitamente compartidas: forman parte del ambiente local de un subprograma, es decir que tienen un dueño, y él las hace visibles a otros subprogramas.

Ambientes no locales implícitos

Otra alternativa al uso de ambientes comunes explícitos (y parámetros) para compartir datos entre subprogramas es mediante el ambiente no local de un subprograma.

Idea: cuando referencio a una variable `X` en un subprograma `P`, y no lo encuentro en el ambiente local de `P`, busco la asociación correspondiente a `X` en el ambiente de referenciación no local de `P`.

Pregunta: ¿Cómo se determina el ambiente no local de un subprograma?

Respuesta: depende del tipo de regla de alcance utilizada por el lenguaje para sus referenciaciones:

- Dinámico
- Estático

Alcance estático y dinámico

Algunos conceptos fundamentales para determinar el ambiente no local:

- Alcance Dinámico de *una asociación* para un identificador: conjunto de activaciones de subprogramas en el cual la asociación es visible.
- Alcance Estático de *una declaración*: es la parte del texto del programa donde un uso del identificador es una referencia a esa declaración particular del identificador.
- Regla de Alcance Dinámico: define el alcance dinámico de cada asociación en términos del curso dinámico de la ejecución del programa (relaciona referencias con asociaciones en ejecución).
- Regla de Alcance Estático: determina el alcance estático de las declaraciones (relaciona referencias con declaraciones de nombres en el texto del programa).

IMPORTANTE: ambas reglas deben ser congruentes.

Ambiente no local y reglas de Alcance

Volviendo al problema de determinar el ambiente no local de un subprograma, podemos decir que si el lenguaje está basado en el alcance estático (como es el caso de los lenguajes estructurados en bloques) las reglas de alcance estático determinarán el ambiente no local implícito.

Los lenguajes que carecen de esta estructura de anidamiento suelen basarse en el alcance dinámico (como en el caso de LISP) y determinan el ambiente no local de un subprograma mediante alguna regla de alcance dinámico

Alcance Dinámico

Si en el subprograma P existe una referencia a una variable foránea (no local) X se busca la asociación considerando la cadena dinámica de activaciones que conducen a P.

La regla de referenciación más conocida basada en el alcance dinámico es la regla de la asociación más reciente (RAMR):

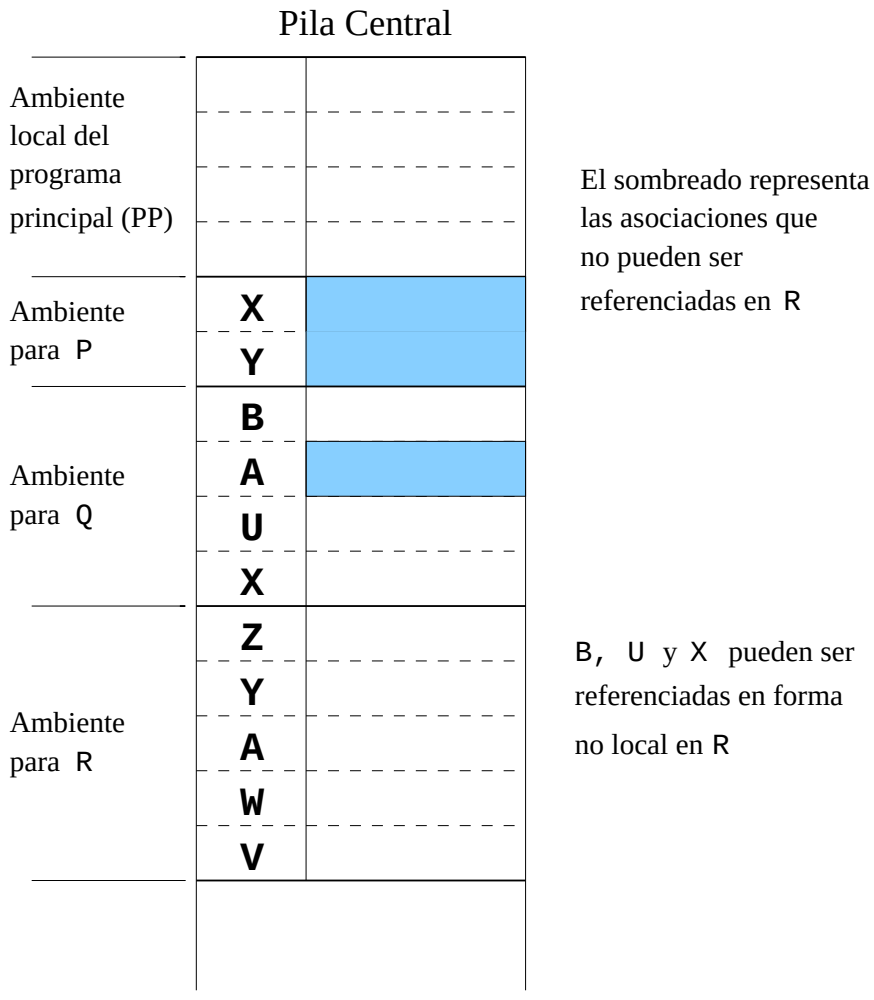
"la referencia a una variable X en un subpograma P corresponde a la asociación local de X en P. Si no existe tal asociación, corresponde a la del subprograma que llamó a P. Si tampoco ésta existe será la del suprograma que llamó a éste, y así sucesivamente hasta alcanzar la asociación más recientemente creada para X".

Posibles implementaciones

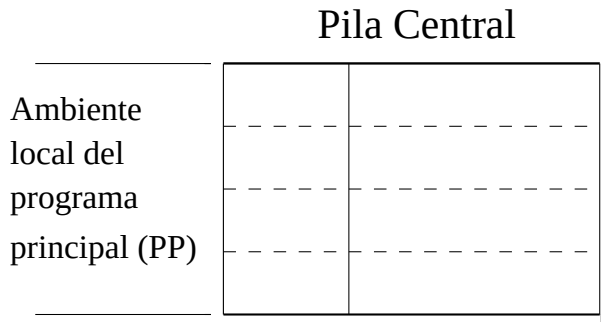
- Búsqueda directa en la pila central de ejecución
- Acceso mediante una tabla central y pila oculta

Ejemplo

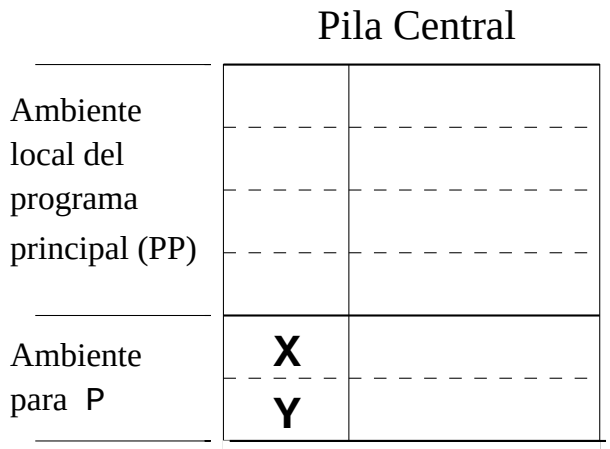
Considere el siguiente caso de tres subprogramas `P`, `Q` y `R`, donde `P` llama a `Q` y `Q` llama a `R`:



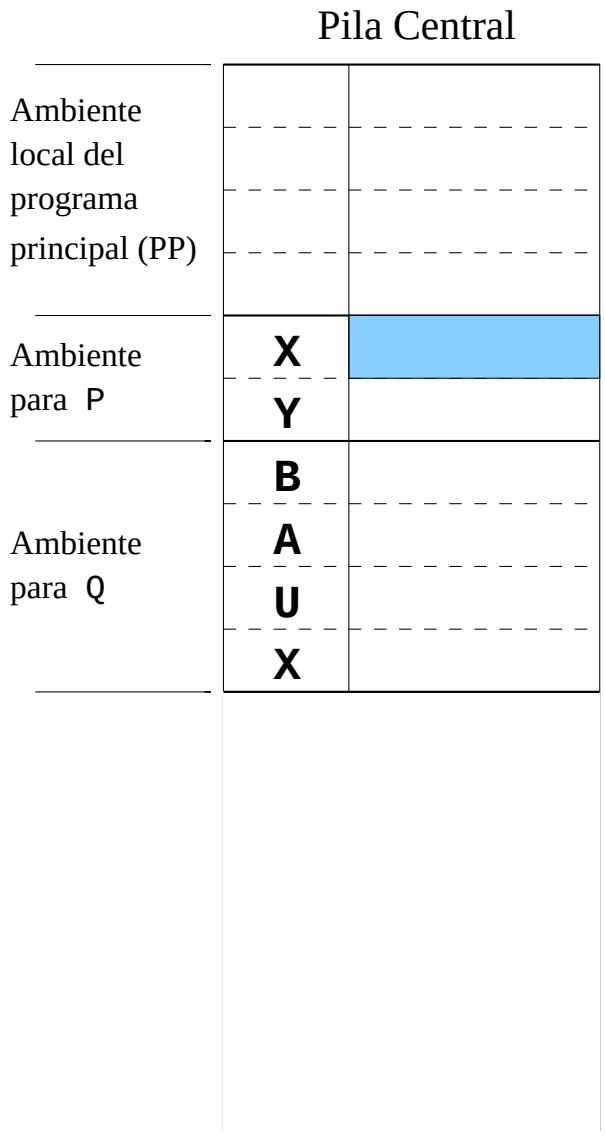
Búsqueda directa en la pila central (1)



Búsqueda directa en la pila central (2)

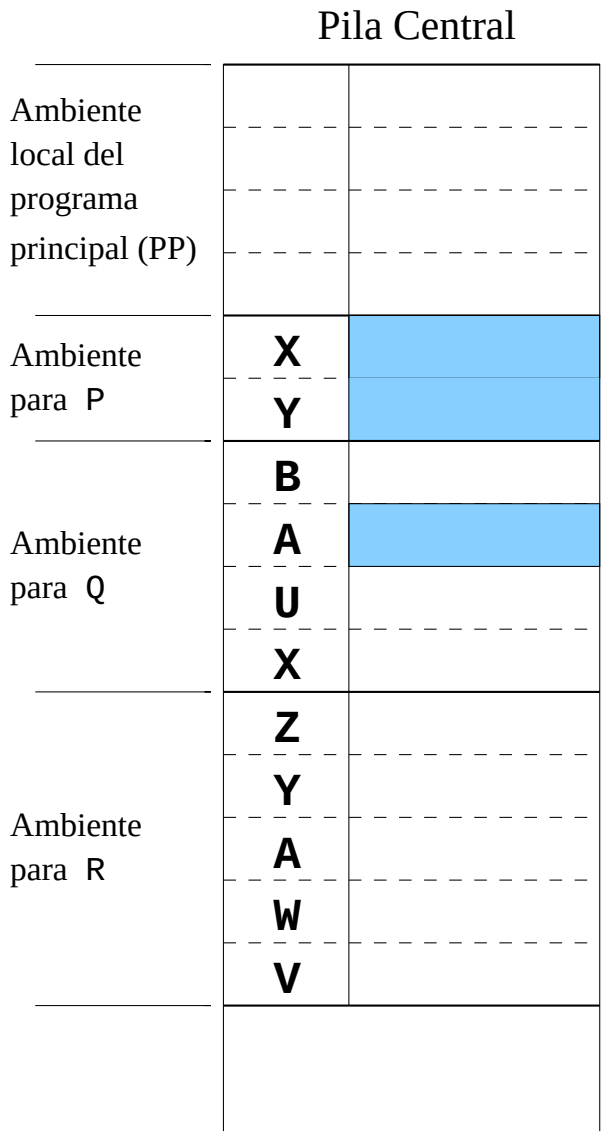


Búsqueda directa en la pila central (3)



El sombreado representa las asociaciones que no pueden ser referenciadas en `Q`

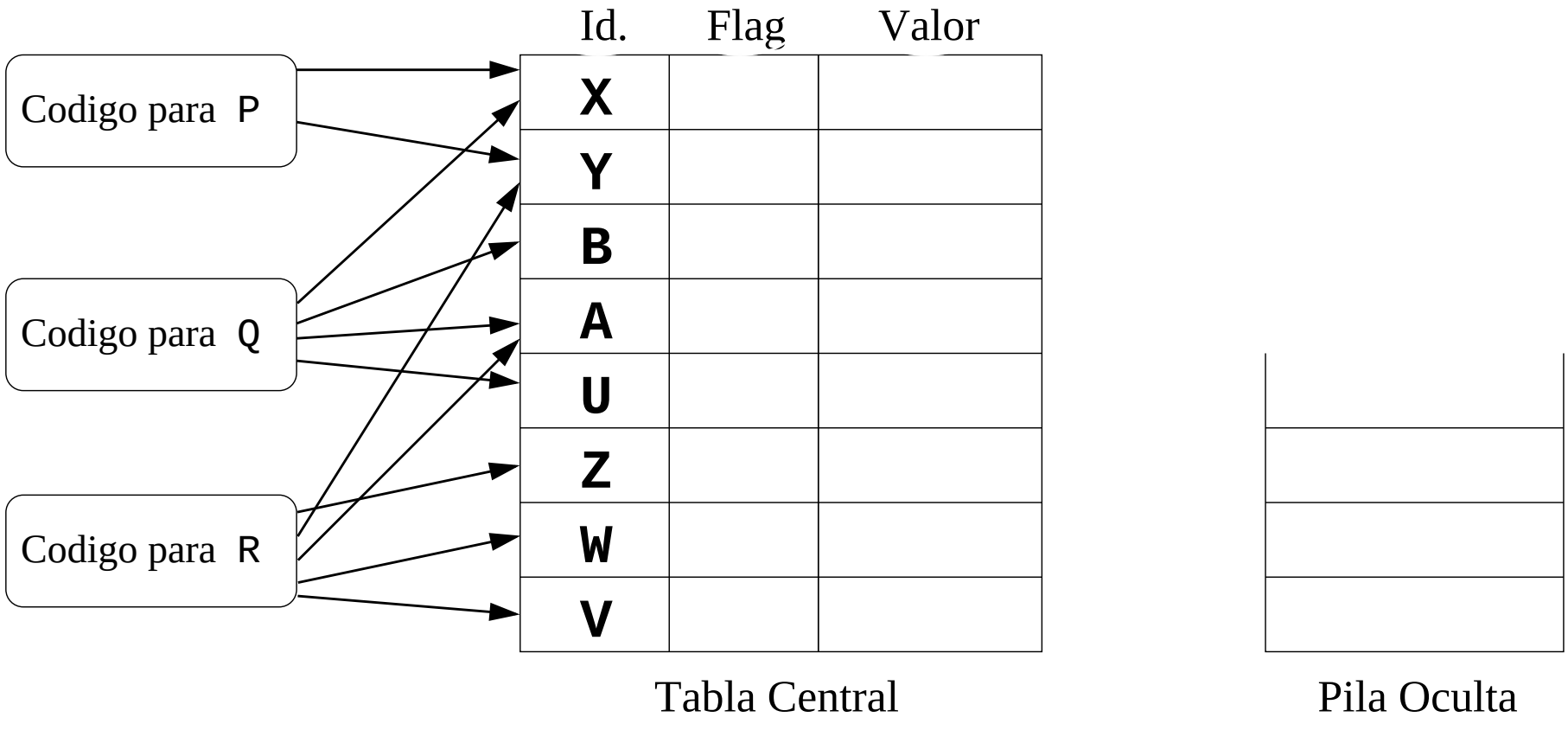
Búsqueda directa en la pila central (4)



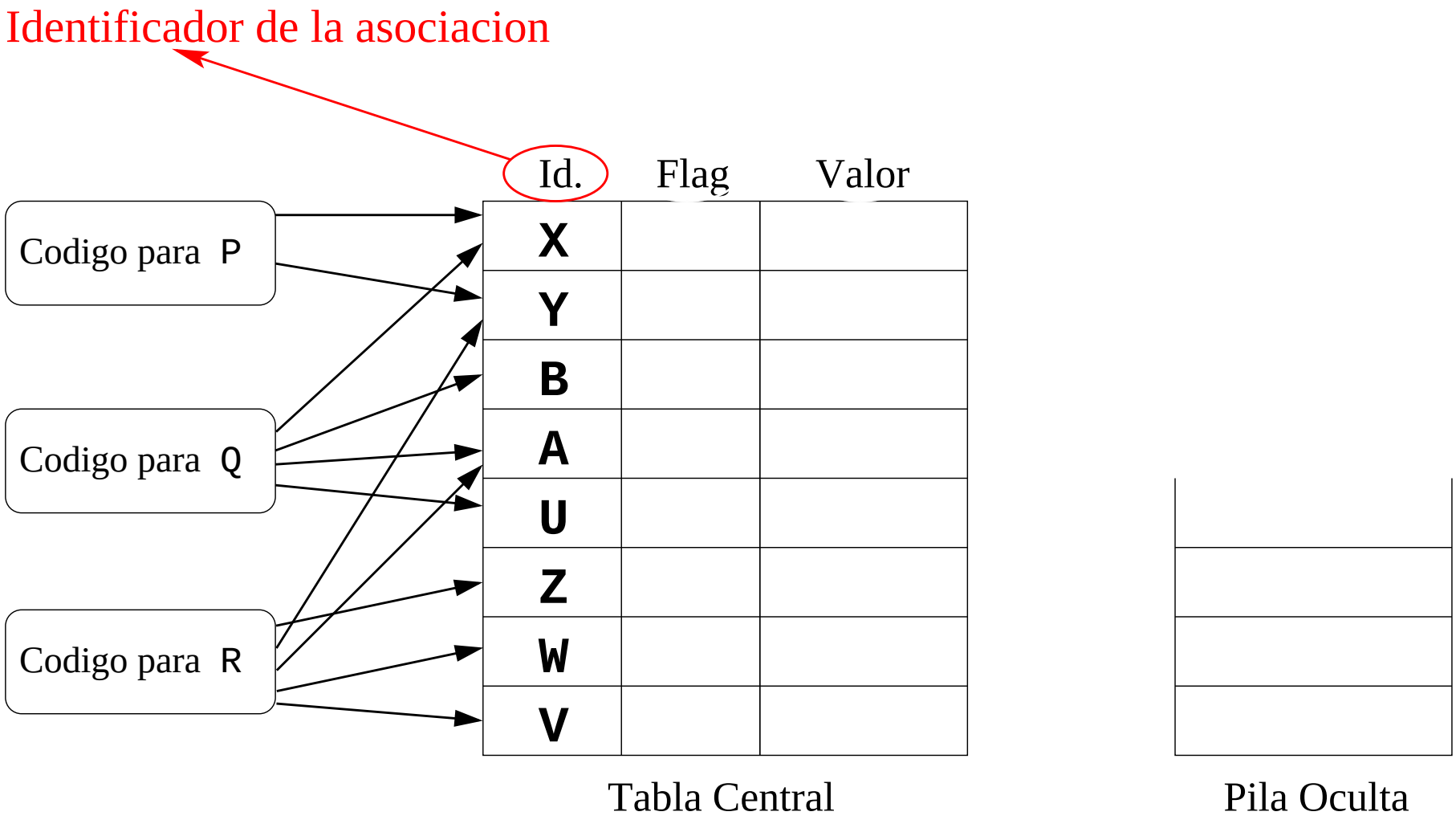
El sombreado representa las asociaciones que no pueden ser referenciadas en `R`

`B`, `U` y `X` pueden ser referenciadas en forma no local en `R`

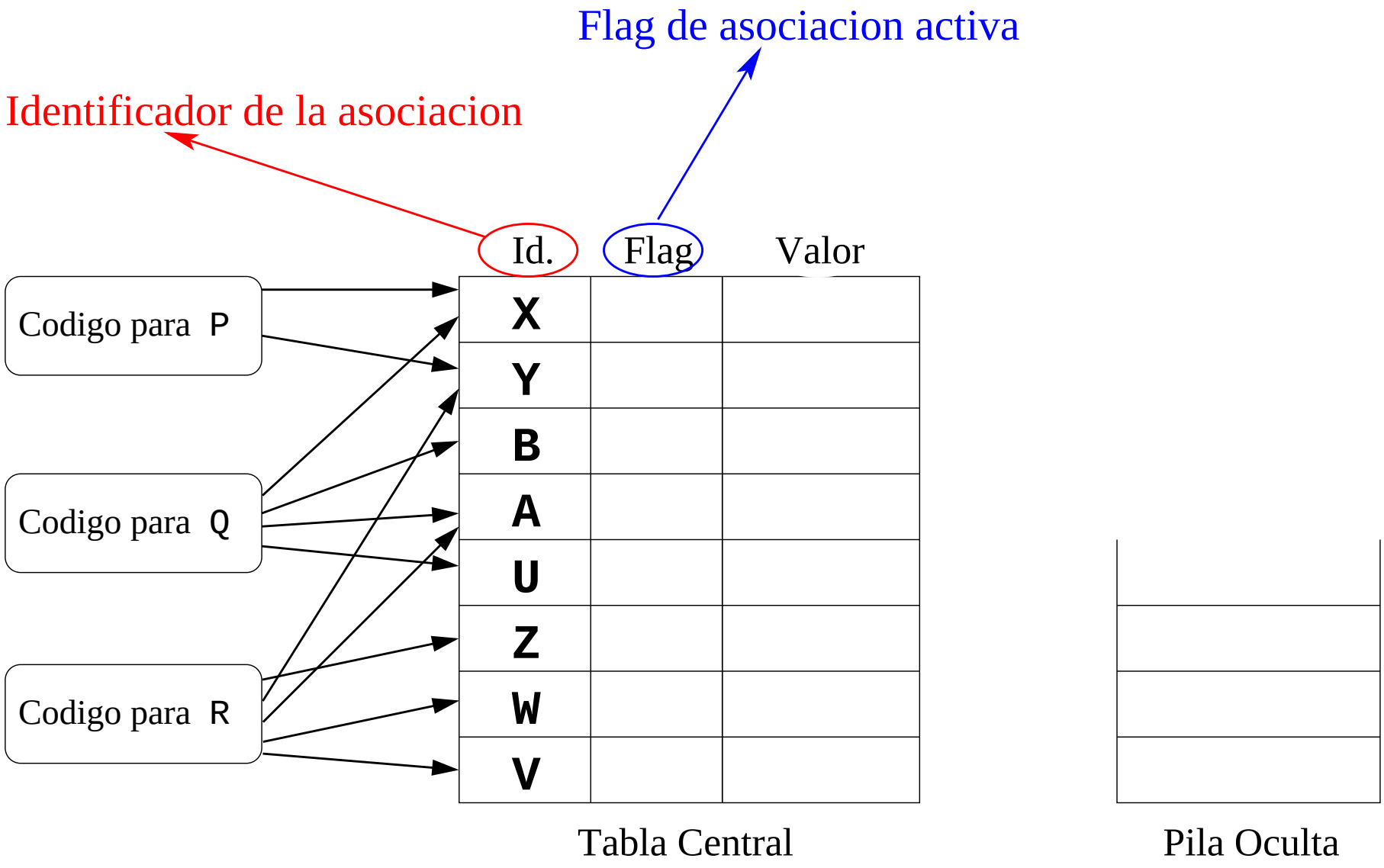
Acceso con tabla central y pila oculta



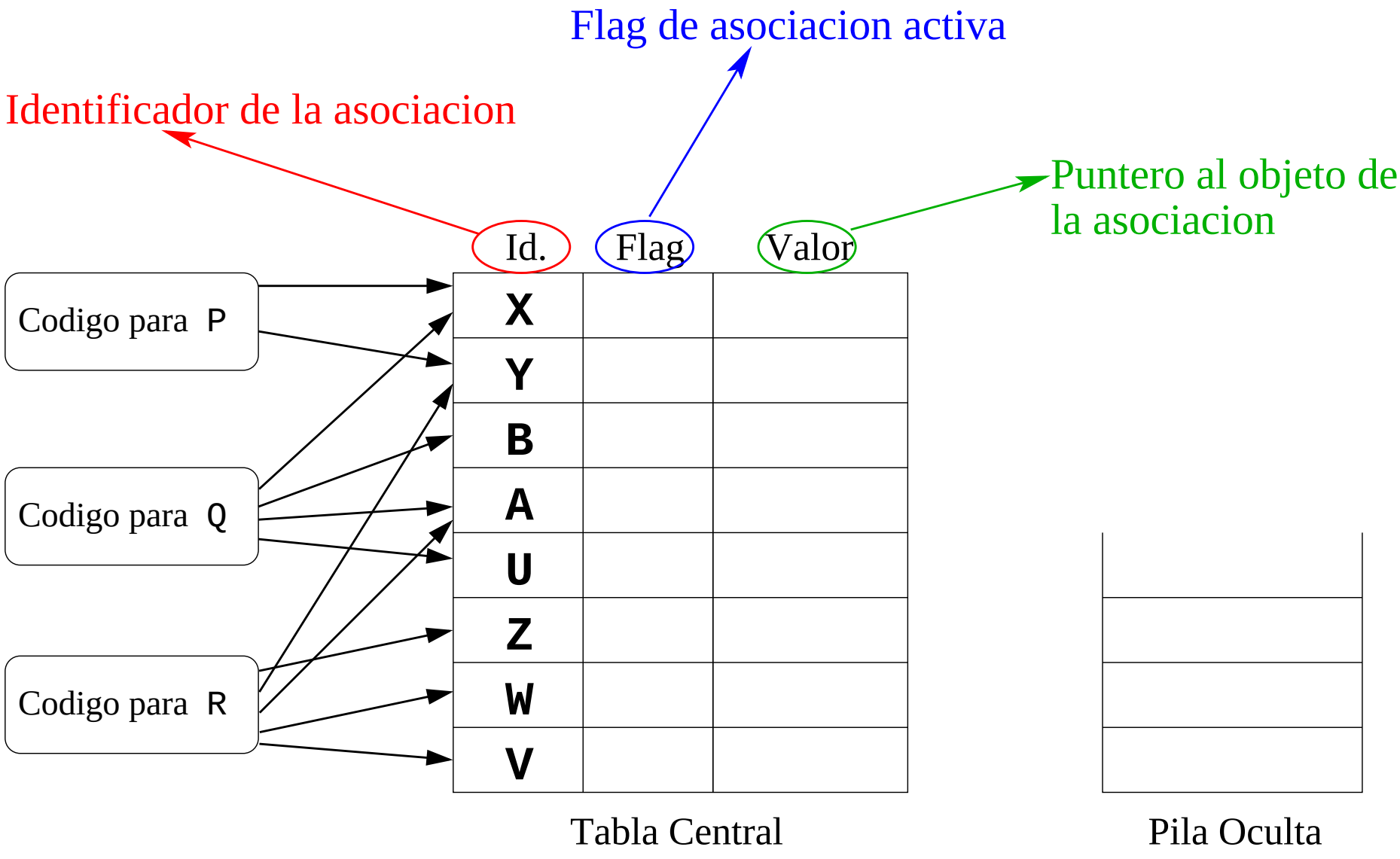
Acceso con tabla central y pila oculta



Acceso con tabla central y pila oculta



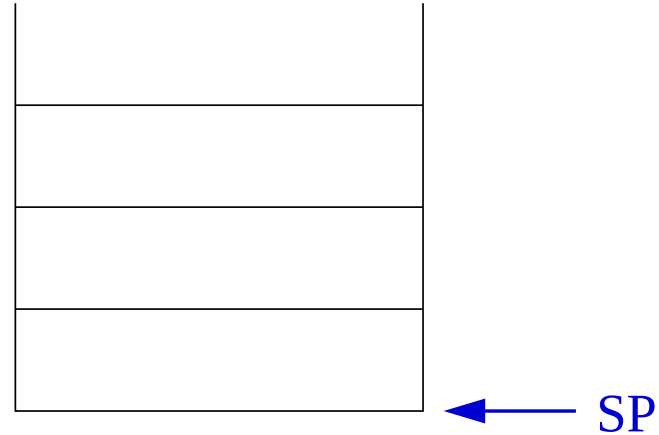
Acceso con tabla central y pila oculta



Situación inicial

Id.	Flag	Valor
X	0	———
Y	0	———
B	0	———
A	0	———
U	0	———
Z	0	———
W	0	———
V	0	———

Tabla Central

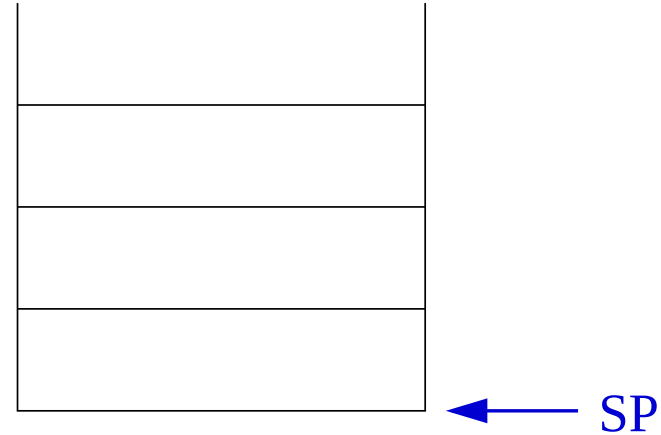


Pila Oculta

Invocación de P

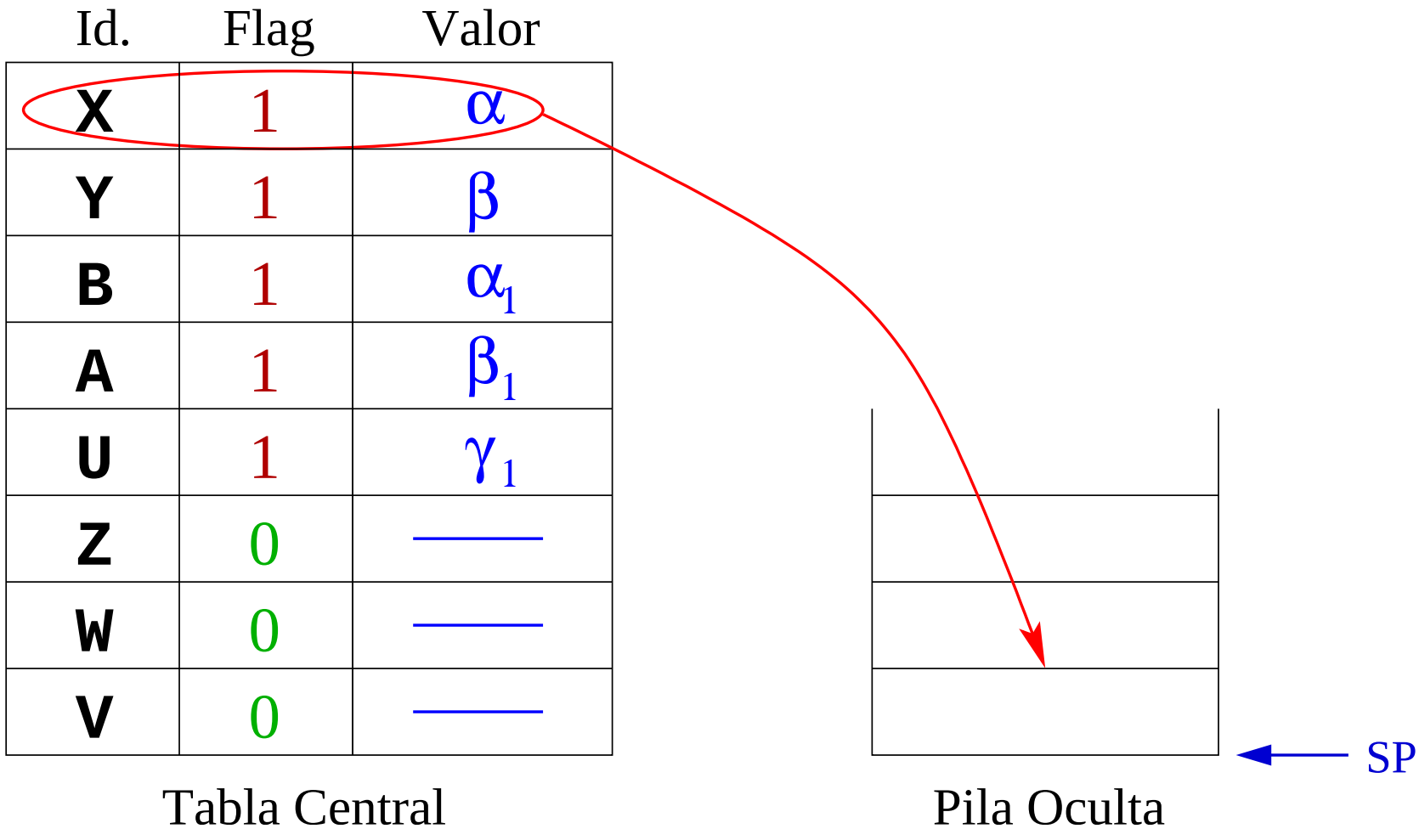
Id.	Flag	Valor
X	1	α
Y	1	β
B	0	———
A	0	———
U	0	———
Z	0	———
W	0	———
V	0	———

Tabla Central



Pila Oculta

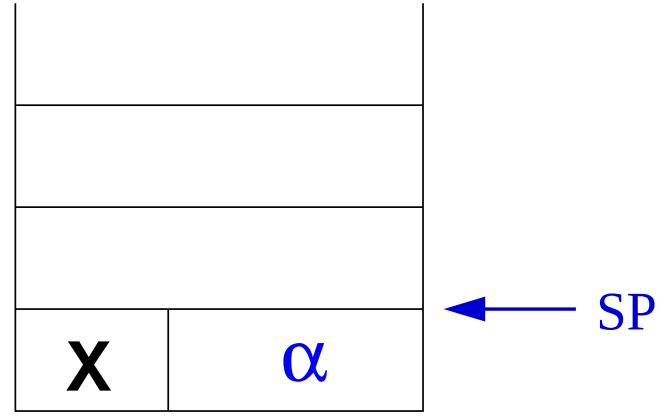
Invocación de Q (1)



Invocación de Q (2)

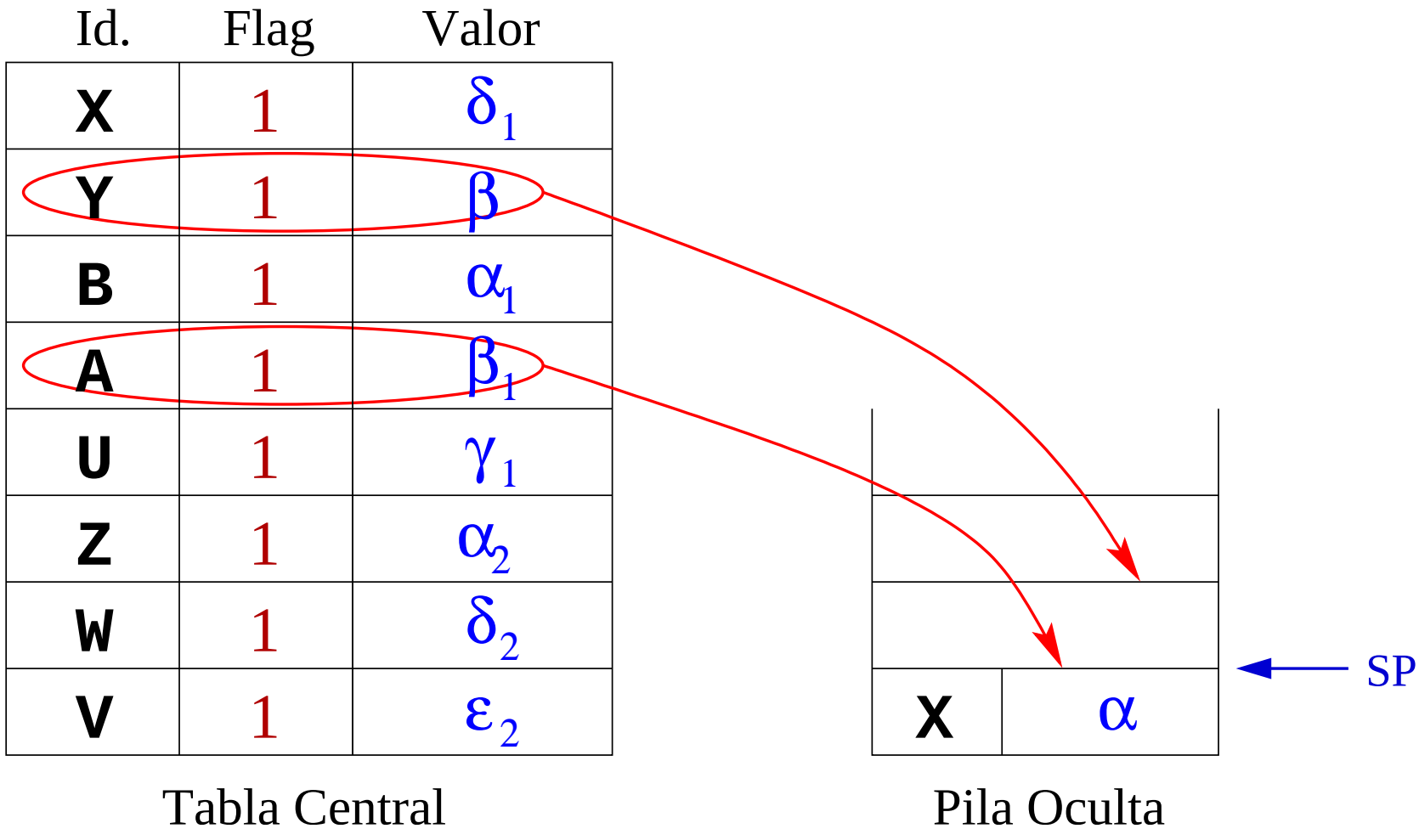
Id.	Flag	Valor
X	1	δ_1
Y	1	β
B	1	α_1
A	1	β_1
U	1	γ_1
Z	0	——
W	0	——
V	0	——

Tabla Central



Pila Oculta

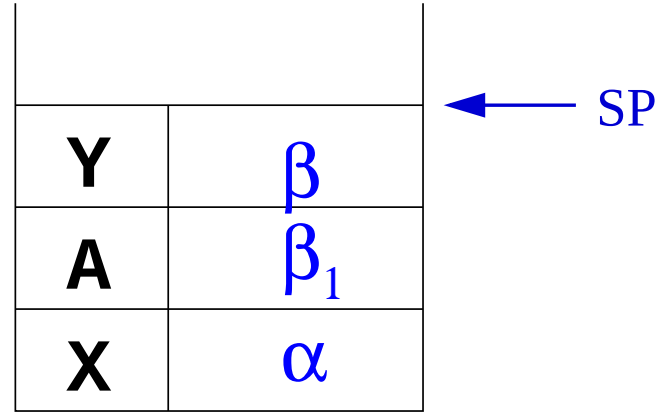
Invocación de R (1)



Invocación de R (2)

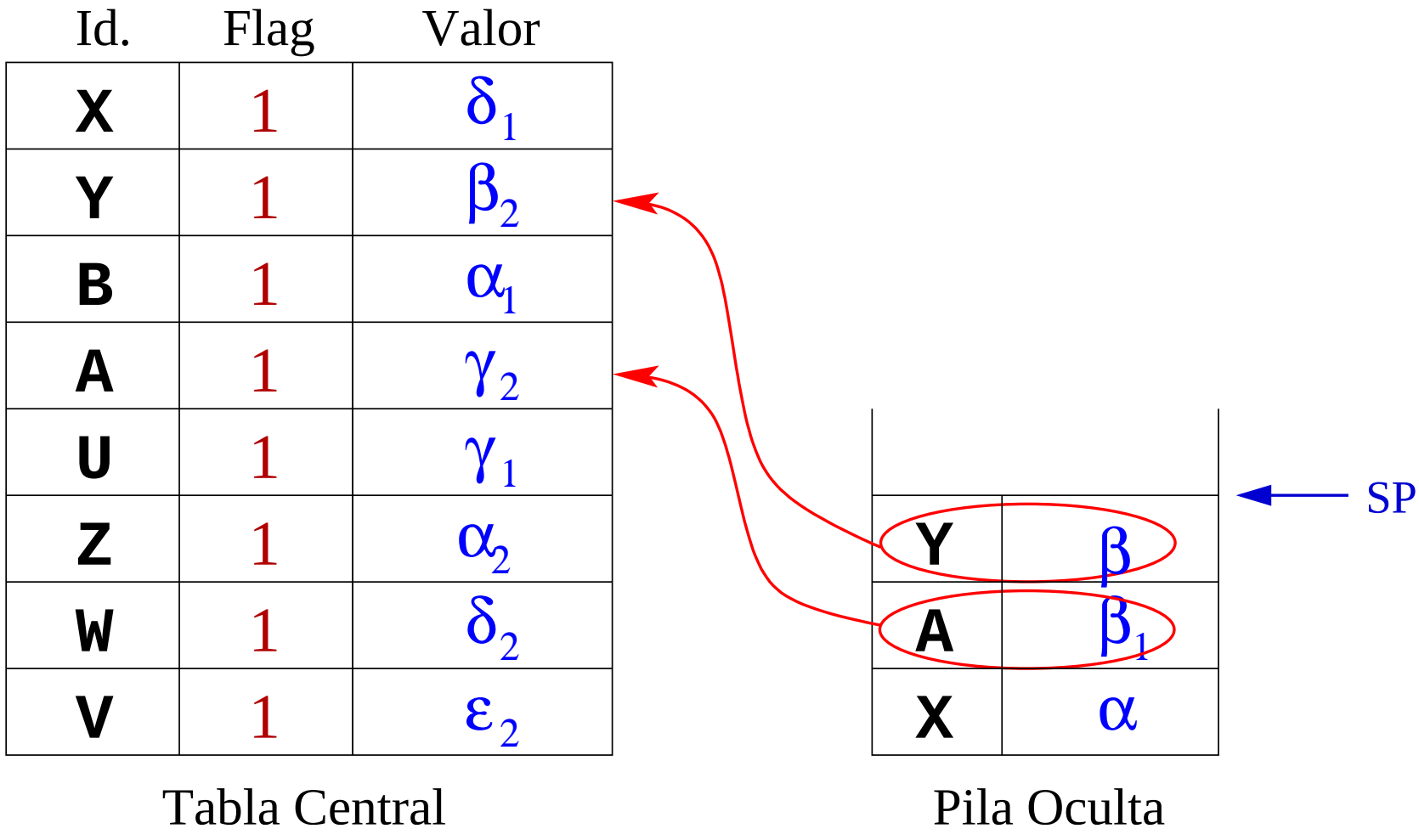
Id.	Flag	Valor
X	1	δ_1
Y	1	β_2
B	1	α_1
A	1	γ_2
U	1	γ_1
Z	1	α_2
W	1	δ_2
V	1	ε_2

Tabla Central



Pila Oculta

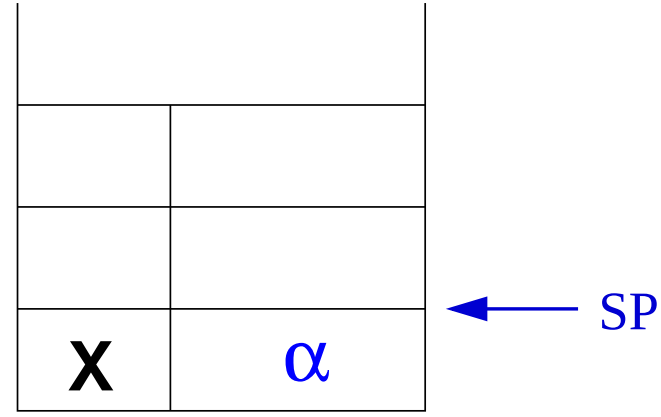
Retorno desde R (1)



Retorno desde R (2)

Id.	Flag	Valor
X	1	δ_1
Y	1	β
B	1	α_1
A	1	β_1
U	1	γ_1
Z	0	α_2
W	0	δ_2
V	0	ε_2

Tabla Central



Pila Oculta

Alcance Estático

El ambiente de referenciación no local de cada subprograma durante ejecución es determinado por las reglas de alcance estático usadas en traducción.

Problema: se debe mantener información en ejecución (sobre la estructura de anidamiento detectada en traducción) para hacer congruentes las reglas de alcance dinámico con las de alcance estático.

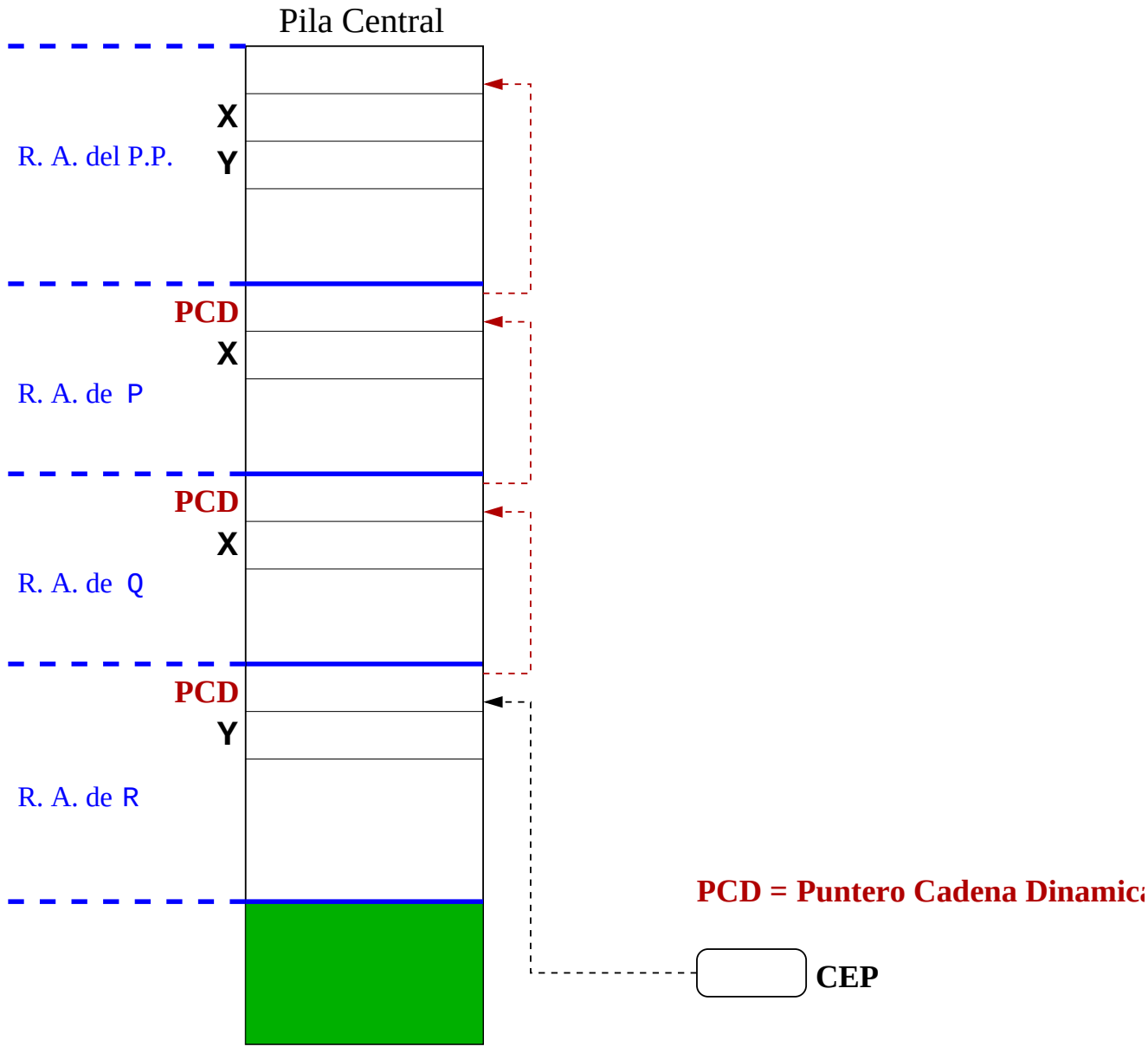
Posibles implementaciones:

1. Uso de la cadena estática
2. Uso de un display (visualizador) o vector separado

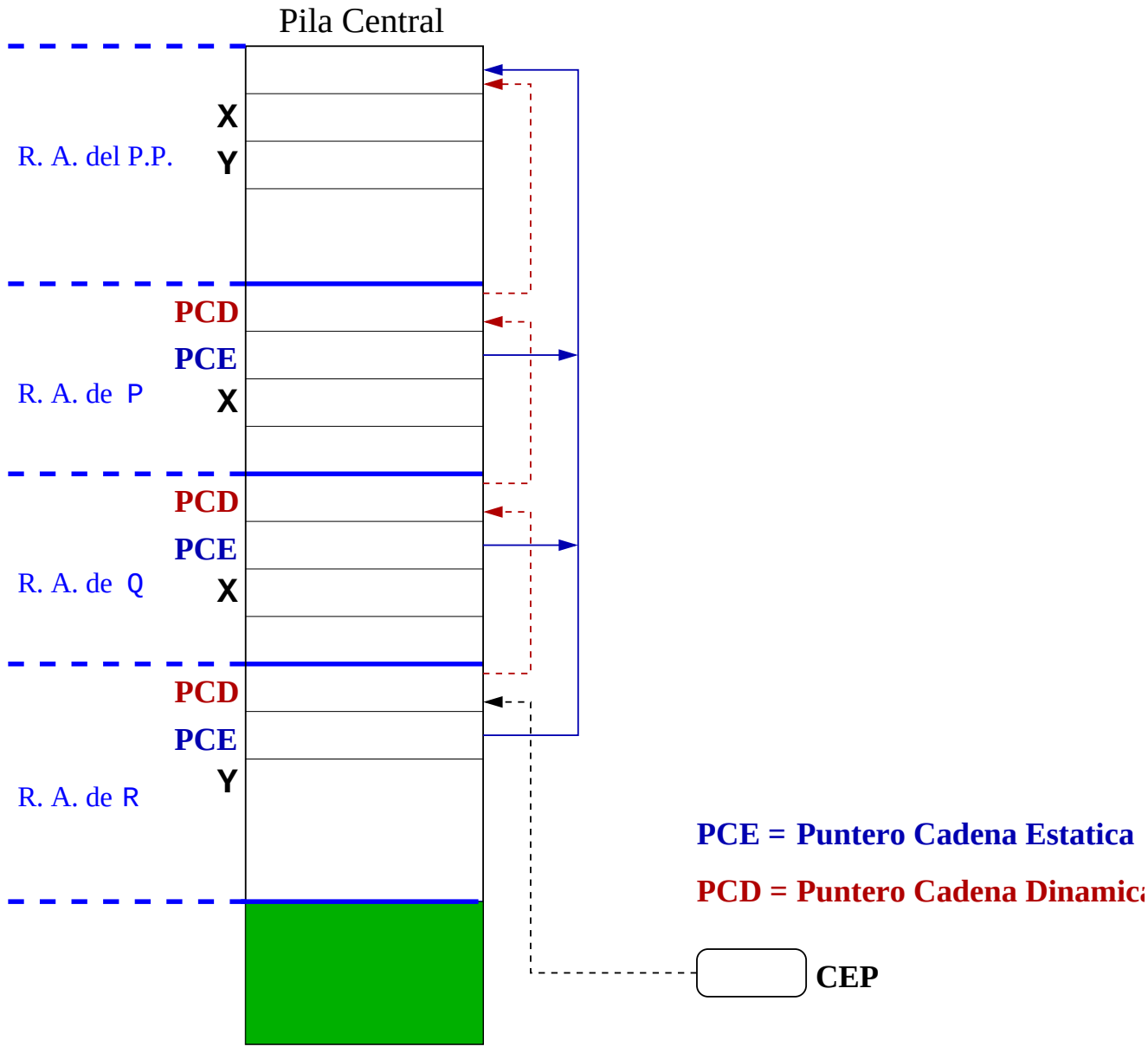
Ejemplo de un programa estructurado en bloques

```
program Principal
  var X, Y: integer;
  procedure R;
      var Y: real;
      begin
      ...
      X := X + 1; /* Referencia no local a X */
      ...
      end R;
  procedure Q;
      var X: real;
      begin
      ...
      R; /* llamar procedimiento R */
      ...
      end Q;
  procedure P;
      var X: boolean;
      begin
      ...
      Q; /* llamar procedimiento Q */
      ...
      end P;
begin /* comenzar Principal */
...
P; /* llamar procedimiento P */
...
end.
```

Información insuficiente para determinar alcance estático



Uso de la cadena *estática*



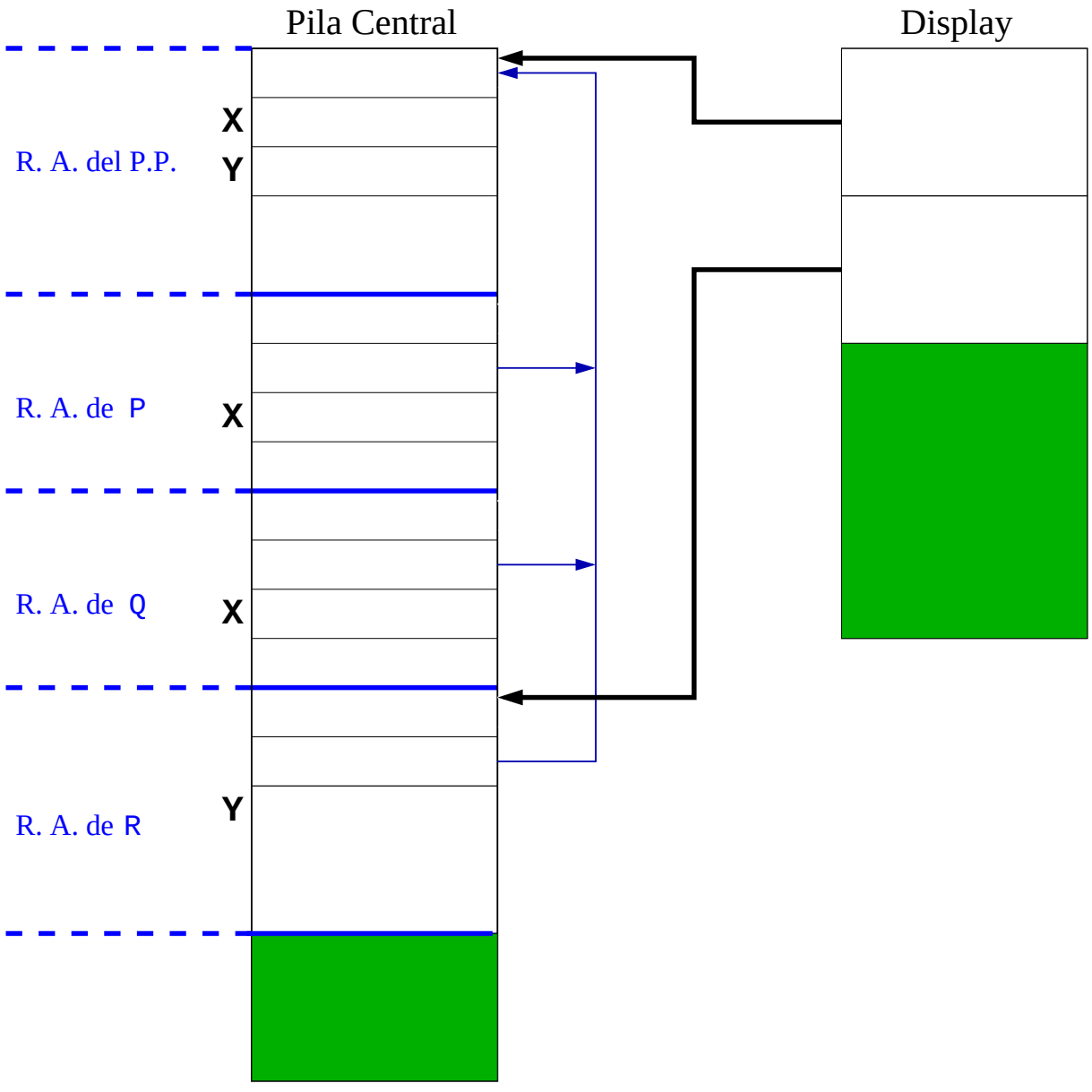
Uso de la cadena *estática*

Idea: se incorpora un puntero en cada registro, de manera de mantener el anidamiento estático de definiciones de subprogramas.

El acceso a una asociación no local puede involucrar varios accesos por la cadena estática pero no constituyen una búsqueda (no se almacena por lo tanto el identificador de la asociación).

Una referencia a una variable `X` es traducida por el compilador a un par `(numac, desp)` donde `numac` es el número de accesos necesarios por la cadena estática (0 para variables locales) y `desp` es el desplazamiento dentro del reg. de act. que contiene a `X`.

Uso de un *display (visualizador)*



Uso de un *display (visualizador)*

La cadena estática del subprograma actualmente en ejecución es mantenida en un vector separado llamado display (visualizador).

Cada entrada en el display representa el nivel que cada subprograma (o programa principal) tiene en el texto del programa. Es el método que se usa habitualmente.

Una referencia a una variable `X` es traducida por el compilador a un par `(i , desp)` donde `i` es el índice a usar en el display y `desp` es el desplazamiento dentro del reg. de act. que contiene a `X`.
La ubicación de `X` se calcula como `display[i] + desp`

El mantenimiento de la cadena estática actualmente activa hace eficiente el acceso pero encarece la entrada y salida de subprogramas por actualización del display.