Organización de la memoria en tiempo de ejecución

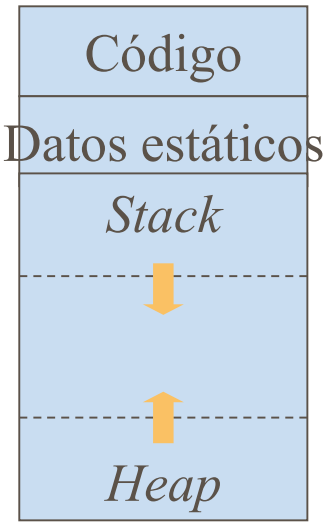
Eva Lucrecia Gibaja Galindo
Dpto. Informática y Análisis Numérico

Administración de la memoria en tiempo de ejecución

- Al ejecutar un programa, la memoria se divide (de manera lógica, no física) en:
  - **Código ejecutable**
    - Código máquina, constantes y literales (sólo lectura)
      - Ocupa un espacio fijo, determinado en tiempo de compilación
      - Se puede situar en una zona estática de memoria
  - **Datos**
    - Variables globales y estáticas
      - Su tamaño es conocido en tiempo de compilación
  - La **pila o *stack*** de control
    - Variables locales y resultados intermedios
    - **Pila de control** que controla las activaciones de los procedimientos
      - Cuando se produce una llamada, se interrumpe la ejecución de una activación y se guarda en la pila la información sobre el estado de la máquina
  - **Montículo o *heap***
    - Guarda el resto de la información generada en tiempo de ejecución
    - Bloques de memoria direccionados por punteros

Administración de la memoria en tiempo de ejecución

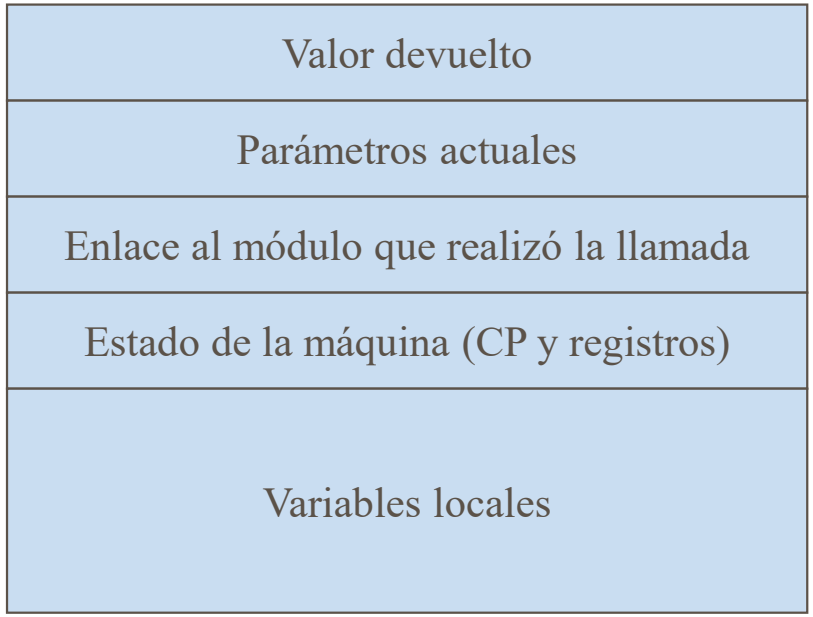
tamaño conocido en tiempo de compilación



La magnitud del *stack* y el *heap* puede variar durante la ejecución del programa y, como la memoria reservada para ambos es fija, si para una se necesita mucho espacio, se reduce el espacio para la otra y viceversa

Registros de activación

- La magnitud de estos campos se puede determinar en tiempo de compilación

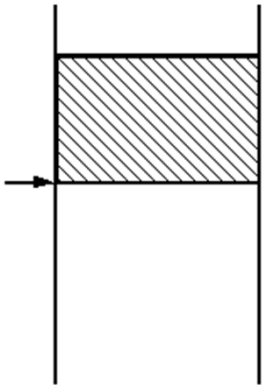


Estrategias de asignación de memoria

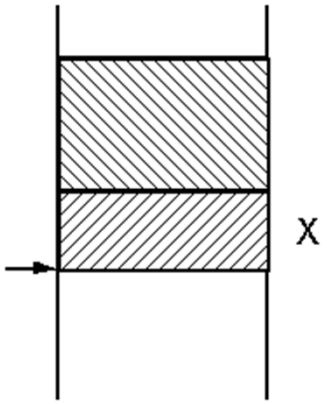
- **Asignación estática**. Dispone la memoria para todos los datos durante la compilación
- **Asignación por medio de una pila**. Trata la memoria en ejecución como una pila
- **Asignación por medio de montículo** y desasigna la memoria conforme se necesita, durante la ejecución

Asignación estática de memoria

- El tamaño de toda la memoria que va a ser reservada es conocido en tiempo de compilación
- Los objetos están vigentes desde que comienza la ejecución del programa hasta que termina
- La memoria se reserva de forma **consecutiva**
- La dirección de memoria de los objetos se puede conocer en tiempo de compilación
  - Si la primera variable está en la dirección x y ocupa n_1 unidades de memoria, la segunda variable estará en la dirección $x+n_1$, la siguiente en $x+n_1+n_2$ etc.



Alojamiento en memoria de un objeto x



Asignación estática de memoria

- Ventaja
  - La implementación de esta estrategia es simple
  - Conocer las direcciones de las variables al generar el código objeto, mejora el tiempo de ejecución de los programas
  - Se suele utilizar en asignaciones estáticas para datos globales
- Inconvenientes
  - El tamaño del objeto ha de ser conocido en tiempo de compilación
  - Al no asignar memoria en tiempo de ejecución:
    - No permite procedimientos recursivos puesto que todas las llamadas recursivas utilizarían las mismas posiciones de memoria, guardando solo un entorno de la función recursiva
    - Las estructuras de datos no se pueden crear dinámicamente (listas, árboles)

Asignación mediante pila (*stack*)

- El número de llamadas recursivas (→número de registros de activación) no es conocido en tiempo de compilación
- Para implementar la recursividad se usan los registros de activación de forma que:
  - Al llamar a un procedimiento, se coloca un nuevo registro en la pila
  - Al finalizar el mismo, se extrae
- Entre la activación de un bloque y su terminación pueden ser invocados otros bloques

Asignación mediante montículo

- La estrategia anterior no es suficiente para manejar estructuras de datos cuya magnitud puede cambiar durante la ejecución de un programa y no es conocida en tiempo de compilación sino en tiempo de ejecución
- Para gestionar esta memoria se reserva un gran bloque contiguo de memoria llamado *montículo*, *montón* o *heap*
- Cuando se realiza una petición adicional de memoria, un controlador de memoria localiza espacio en el *heap* en tiempo de ejecución
- Cuando un especio ha dejado de utilizarse, esta memoria ha de ser liberada. Existen tres técnicas de liberar la memoria:
  - **No liberar**. Cuando se acaba la memoria se para la ejecución del programa
  - **Liberación explícita**. Con lo cual se deja al programador la responsabilidad de liberar la memoria
  - **Liberación implícita**. Recuperación automática del espació en el *heap* sin utilizar. Este proceso se denomina a menudo **recolección de basura**

Asignación mediante montículo

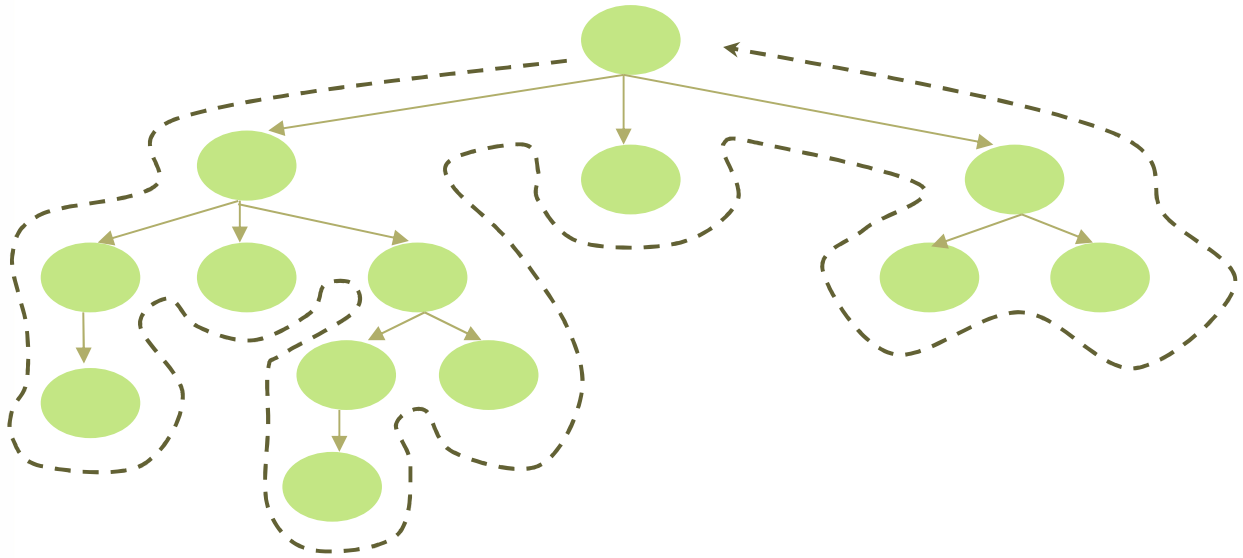
- Estrategias de desasignación **implícita:**
  - **Desocupación sobre la marcha** (*free-as-you-go*)
    - Desalojar (implícitamente) los bloques del montón tan pronto como se detecta que no están referenciados por ningún puntero
    - Es necesario que el sistema lleve internamente un contador del número de punteros que se dirigen hacia cada uno de los bloques del montón
  - **Método de *Marcar y Barrer*** (*mark and sweep*)
    - Activar un procedimiento especial cuando se detectan ciertas condiciones en el uso de la memoria (p. e. cuando queda poca memoria libre, finaliza un módulo, etc.)

El árbol de activación

- Un **árbol** está compuesto por un conjunto de **nodos** y por un conjunto de **arcos** que conectan a los nodos. Nodos y/o arcos pueden tener etiquetas que los identifiquen
- Los nodos se clasifican en tres clases:
  - **Nodo raíz**: no posee ningún arco entrante
  - **Nodo hoja**: no posee ningún arco saliente
  - **Nodo interno**: Posee un arco entrante y uno o más arcos saliente
- Las condiciones que debe cumplir un árbol son:
  - Ha de tener una y sólo una raíz
  - Un nodo sólo puede tener un arco entrante (un único padre)
- Si un nodo A posee n arcos salientes hacia los nodos con etiquetas P_1, ..., P_{n-1} y P_n, entonces se dice que A es el padre de P_1, ..., P_{n-1} y P_n y éstos son, a su vez, los hijos de A

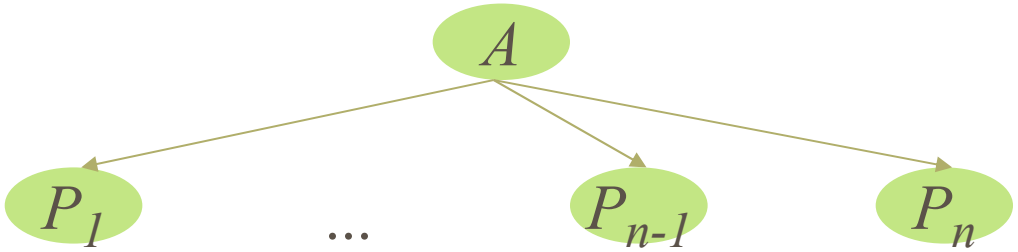
El árbol de activación

- Para representar las llamadas, se suele utilizar un árbol, llamado **árbol de activación,** una representación gráfica que muestra en qué orden se ejecutan un programa y los subprogramas que contiene
- Se utiliza el método del **recorrido en profundidad** para recorrer los nodos de un árbol:
  - Se comienza en la raíz
  - Un nodo será visitado antes que sus hijos
  - Los hijos de un nodo se visitan de manera recursiva y de izquierda a derecha

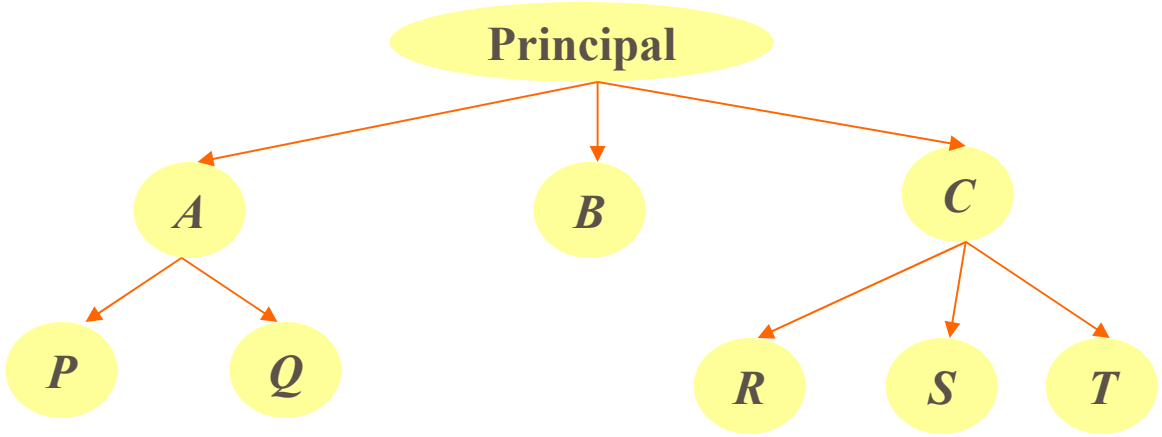


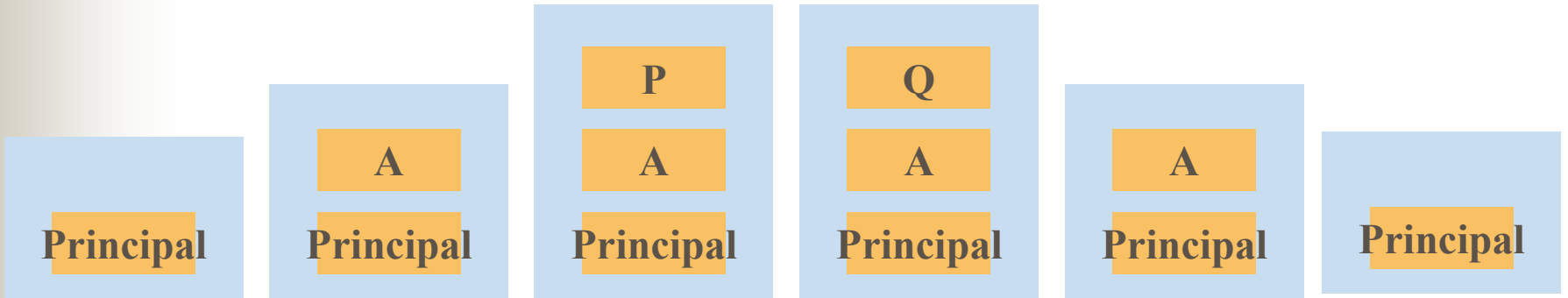
El árbol de activación

- Construcción del árbol de activación:
  - El nodo raíz representa la activación del programa principal
  - Si un subprograma A invoca (activa) a los subprogramas P_1, .., P_{n-1}, y P_n, entonces el nodo asociado a la activación de A posee n nodos hijos que se corresponden con las activaciones de P_1, ..., P_{n-1} y P_n
  - Un nodo a estará a la izquierda de otro b si a se activa antes que b
- Para controlar las activaciones, se utiliza la **pila de control**. Se introduce un **registro de activación** cada vez que se produce la activación de un procedimiento y se saca de la misma cuando termina

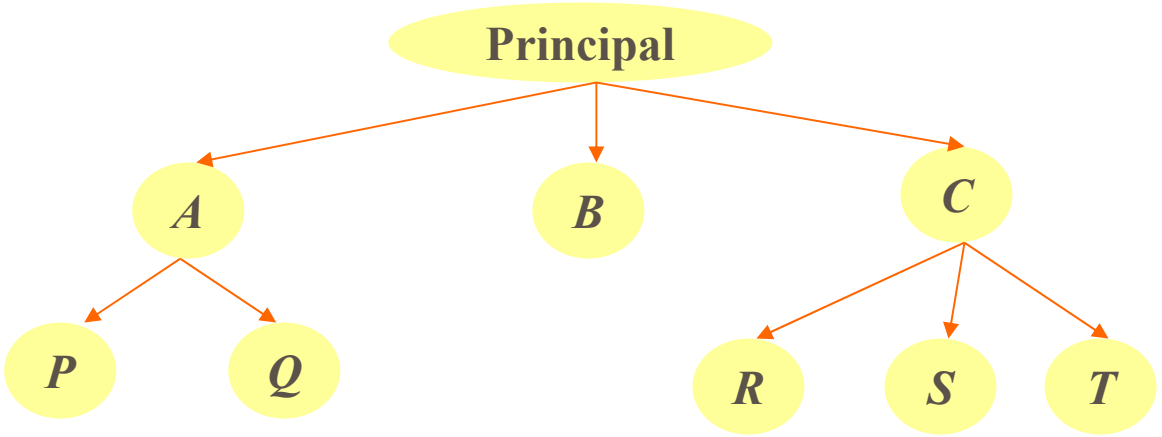


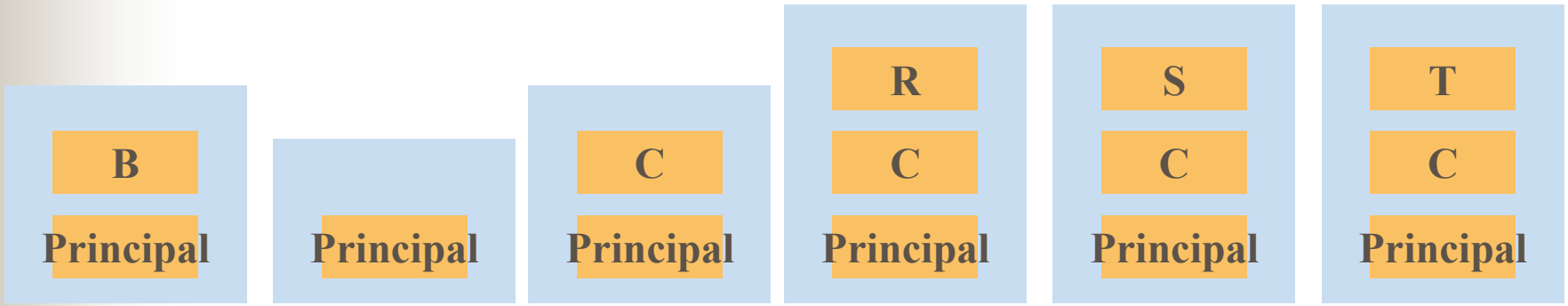
El árbol de activación





El árbol de activación






Cuando haces
pop ya no hay
Stack...