

semántica

Paradigmas de la Programación

FaMAF 2021

programa: sintaxis y semántica

- un programa es la descripción de un proceso dinámico
 - **sintaxis**: texto del programa
 - **semántica**: cosas que hace
- una definición precisa del significado de un programa correcto sintácticamente y a nivel de tipos

lenguaje objeto y metalenguaje

- el **metalenguaje** es el que usamos para hablar de un **lenguaje objeto**
- necesitamos un lenguaje para hablar de la semántica de los lenguajes de programación

semántica

delimitación de la semántica de los lenguajes de programación

algunas observaciones sobre computabilidad
(capítulo 2 de Mitchell, no es necesario leerlo
pero puede resultarles interesante y es cortito)

- los programas pueden definir **funciones parciales**
 - algunos de sus valores pueden **indefinidos** (p.ej., si no terminan)
 - algunos de sus valores pueden ser **errores**

delimitación de la semántica de los lenguajes de programación

- intuitivamente, una función es computable si hay algún programa que la computa
 - problema: definición dependiente de la implementación de un lenguaje de programación concreto, con sus limitaciones y particularidades
- queremos una definición independiente (libre!) de lenguaje

cómo definir la clase de funciones computables?

- una clase de funciones matemáticas: las **funciones recursivas parciales** (Church)
- las que se pueden computar con una máquina idealizada, abstracta: la máquina de Turing
 - cinta infinita, dividida en celdas
 - un cabezal de lectura – escritura
 - un controlador de estado finito
- si se puede expresar en **lambda cálculo**

diferentes aproximaciones a la semántica

- lambda cálculo
- semántica denotacional
- semántica operacional

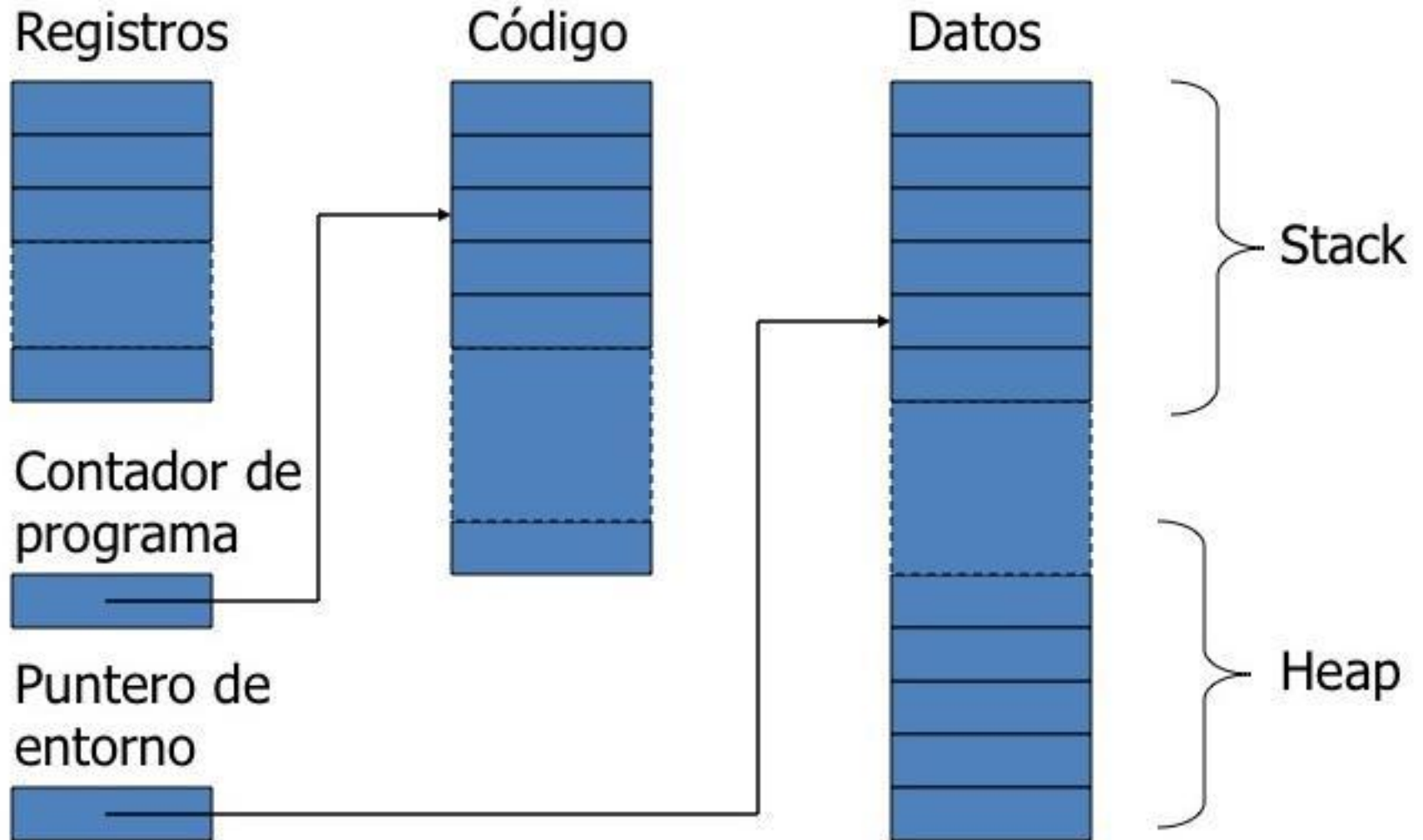
semántica operacional

- apartado 7.2 de Mitchell
- capítulo 5. de la guía de lectura

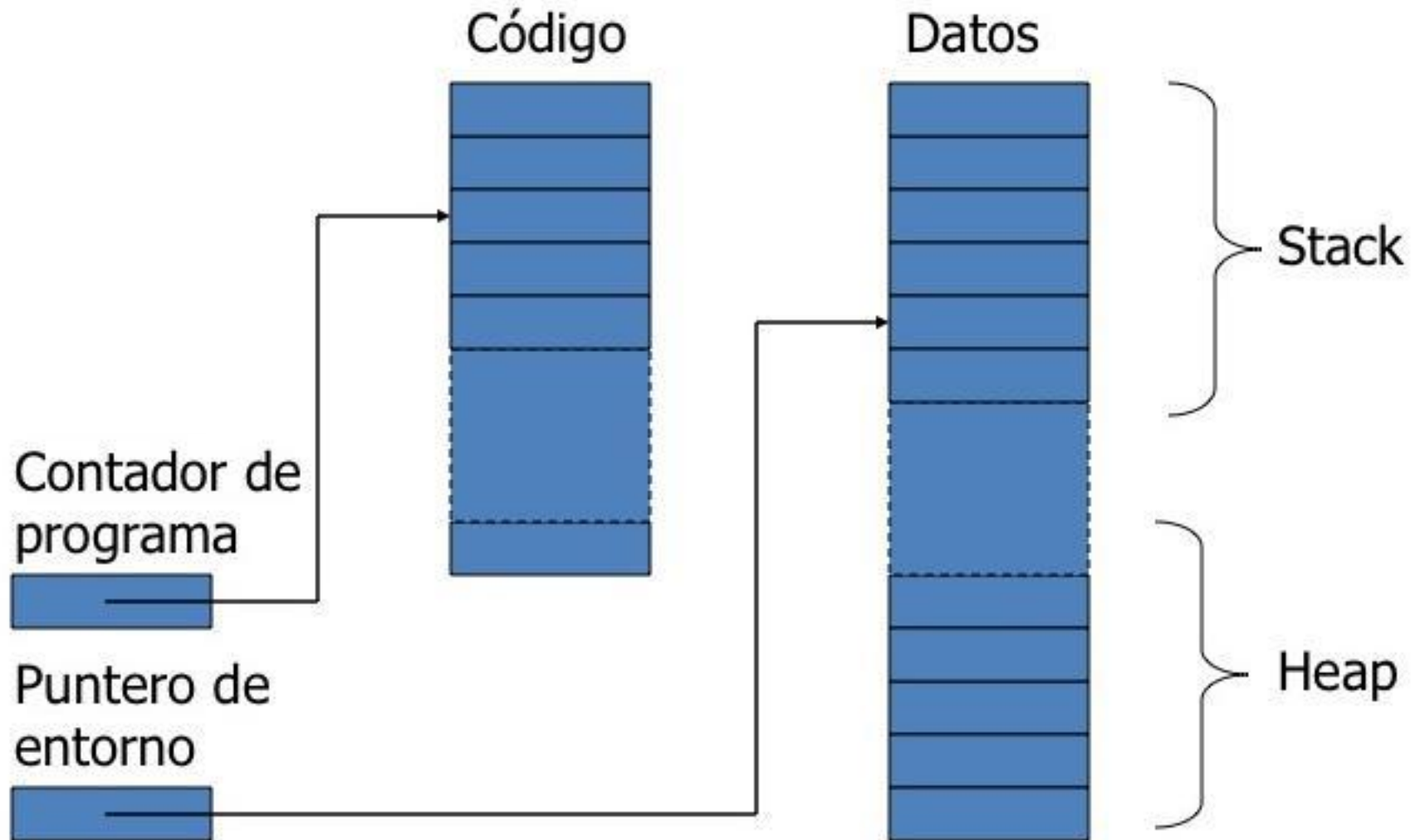
semántica operacional

- una representación abstracta de la ejecución de un programa, como secuencia de transiciones entre estados (en una máquina abstracta)
- los estados son una descripción abstracta de la memoria y estructuras de datos
- las transiciones siguen la estructura de la sintaxis

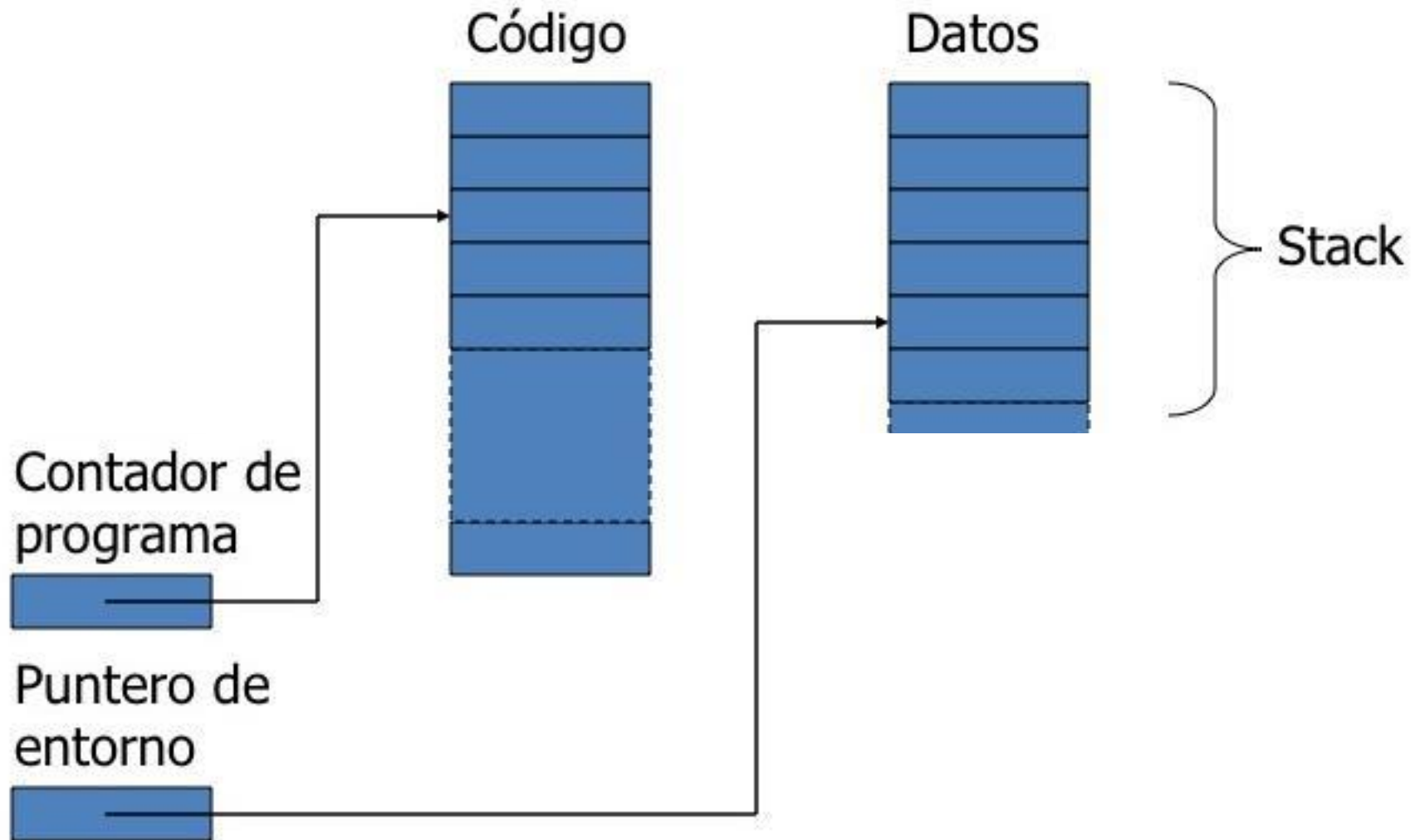
una máquina abstracta



una máquina abstracta



una máquina abstracta



información en la máquina abstracta

- separa memoria de código y de datos
 - **contador de programa:** dirección de memoria con la instrucción que se está ejecutando
 - **puntero de entorno:** valores de las variables en una parte del código
- **lenguajes no estructurados por bloques:** la memoria de datos es no estructurada, los valores de las variables son visibles desde todo el código

información en la máquina abstracta

- **lenguajes estructurados por bloques: pila de ejecución o stack.** Cuando el programa entra en un nuevo bloque, se agrega a la pila un **activation record** con espacio para las variables locales del bloque, y el puntero de entorno apunta al nuevo activation record. Cuando el programa sale del bloque, se retira el activation record de la pila y el puntero de entorno se restablece a su ubicación anterior.

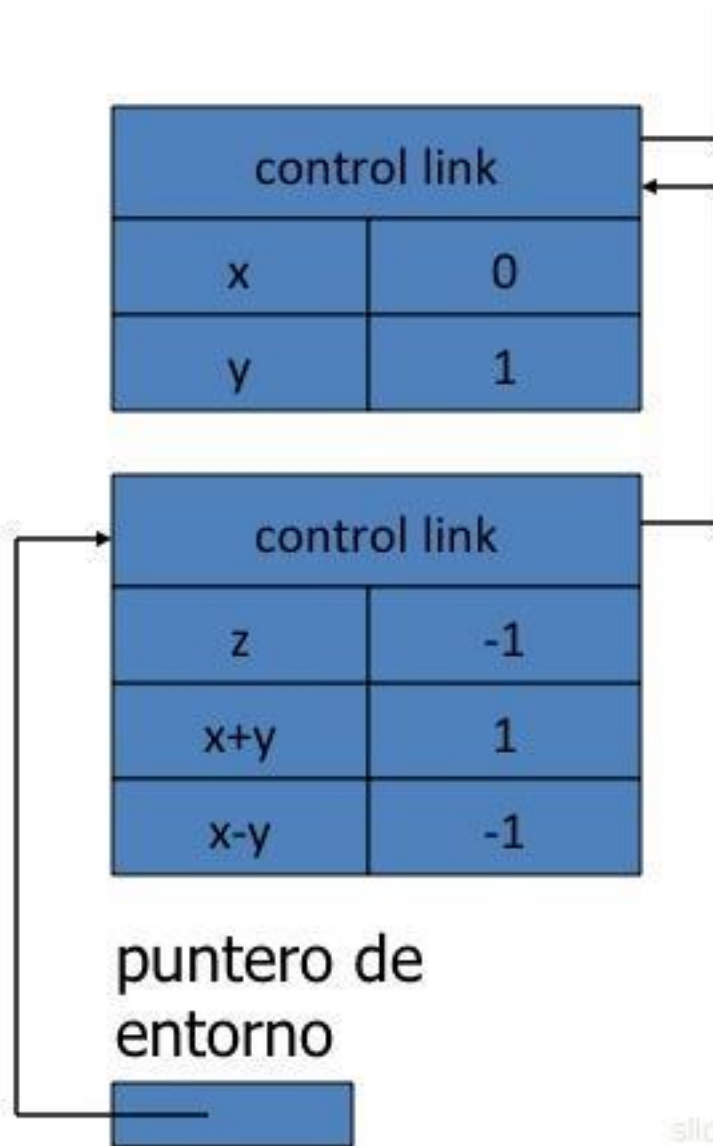
pila de ejecución

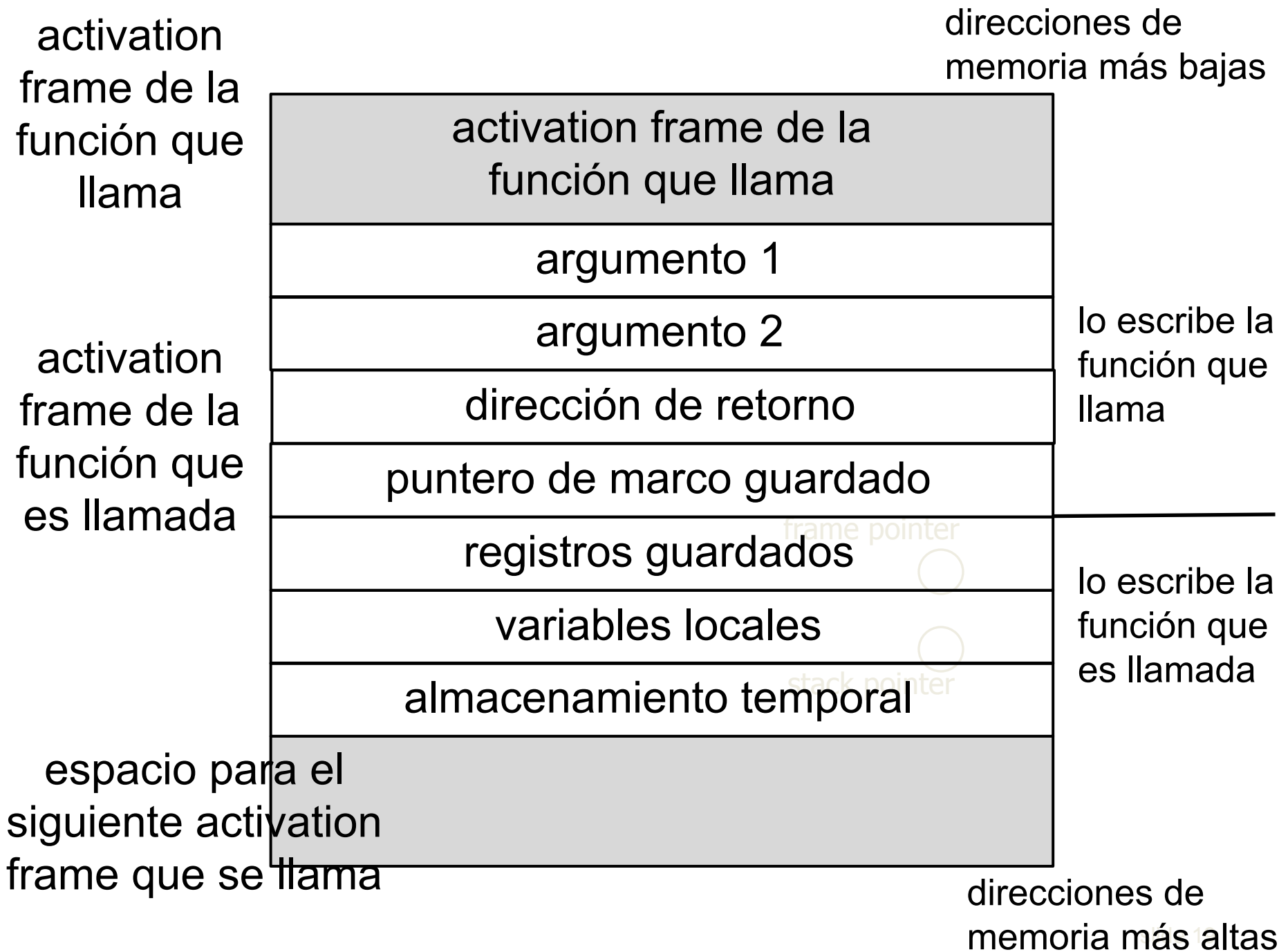
- los registros de activación se guardan en la pila
 - cada nuevo bloque apila (*push*) un nuevo registro de activación en la pila
 - cada vez que se termina un bloque se saca (*pop*) el registro de arriba de la pila
 - la pila tiene todos los registros que son activos en un determinado momento de la ejecución, con el que se usó más recientemente en la punta

registros de activación o marcos de pila (*Activation Records* o *stack frames*)

guardan la información de un bloque:

- variables locales
- control link al que ha llamado al activation record, para reubicar el puntero de entorno
- variables temporales y resultados intermedios
- entran y salen de la pila (*stack*), eso hace que puedan usarse llamados anidados, recursivas





pila de ejecución: ejemplo

espacio para
variables globales

espacio para
variables globales

espacio para
 x e y


espacio para
variables globales

espacio para
 x e y


espacio para
 z

espacio para
variables globales


espacio para
 x e y




```
{  
  int x=0;  
  int y=x+1;  
  {  
    int z=(x+y)*(x-y);  
  };  
}
```



```
{  
  int x=0;  
  int y=x+1;  
  {  
    int z=(x+y)*(x-y);  
  };  
}
```



```
{  
  int x=0;  
  int y=x+1;  
  {  
    int z=(x+y)*(x-y);  
  };  
}
```



```
{  
  int x=0;  
  int y=x+1;  
  {  
    int z=(x+y)*(x-y);  
  };  
}
```

pila de ejecución: otro ejemplo

fact(3)

1. apila un registro, llama a fact(2)
2. esta llamada apila otro registro, llama a fact(1)
3. esta llamada apila otro registro, de forma que hay tres registros en la pila
4. cuando se termina de ejecutar el bloque del registro más reciente, se saca ese registro de la pila
5. y así sucesivamente hasta que la pila queda vacía

