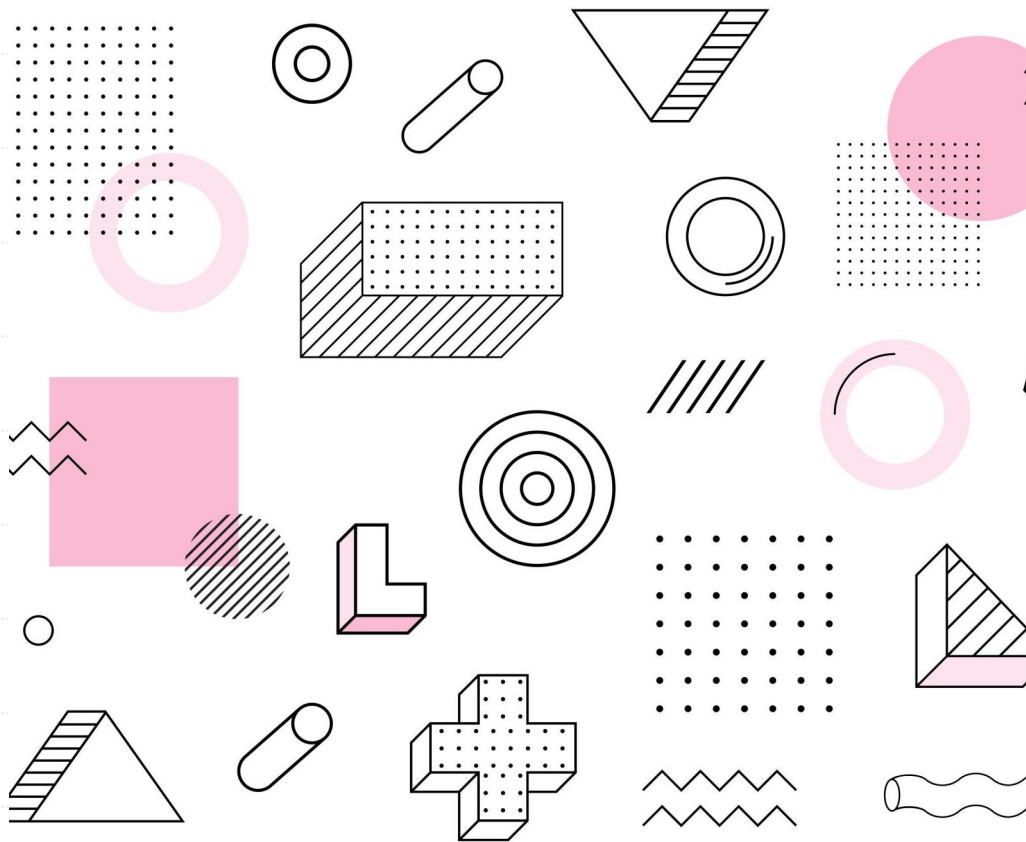


陳奇業 成功大學資訊工程系

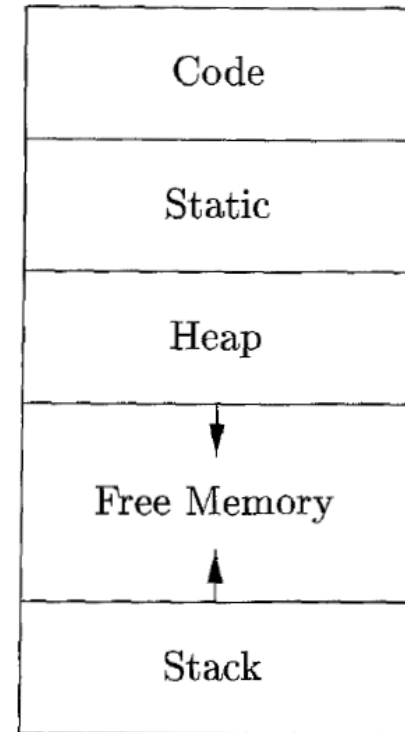
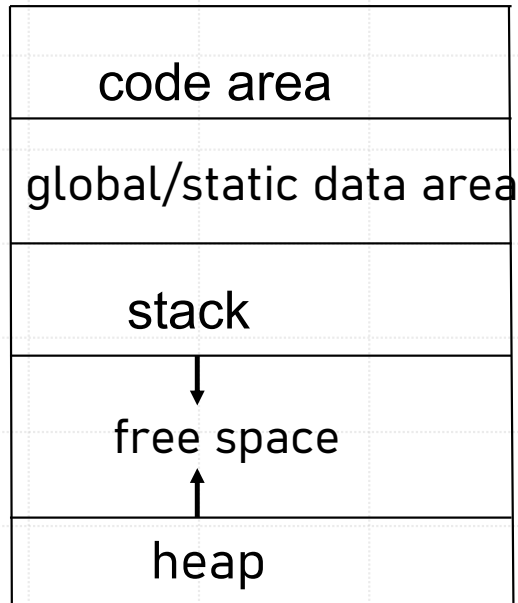




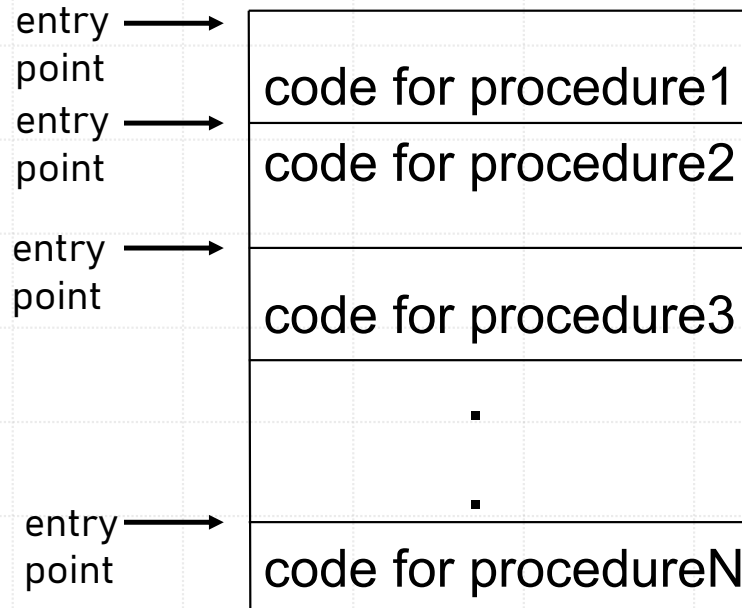
# Memory of a computer

- Register
- RAM
  - Code Area
  - Data Area

# General Organization of Runtime Storage

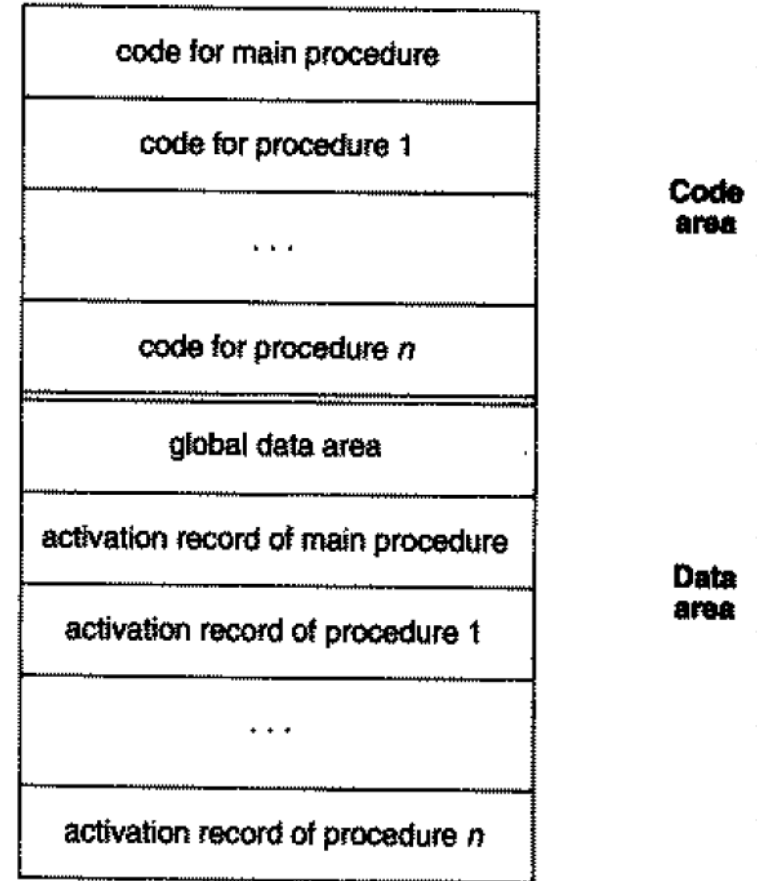



# Code Area



# Fully Static Runtime Environments

- The simplest kind of a runtime environment is that in which all state is static, remaining fixed in memory for the duration of program execution.
- Such an environment can be used to implement a language in which there are no pointers or dynamic allocation, and in which procedures may not be called recursively.



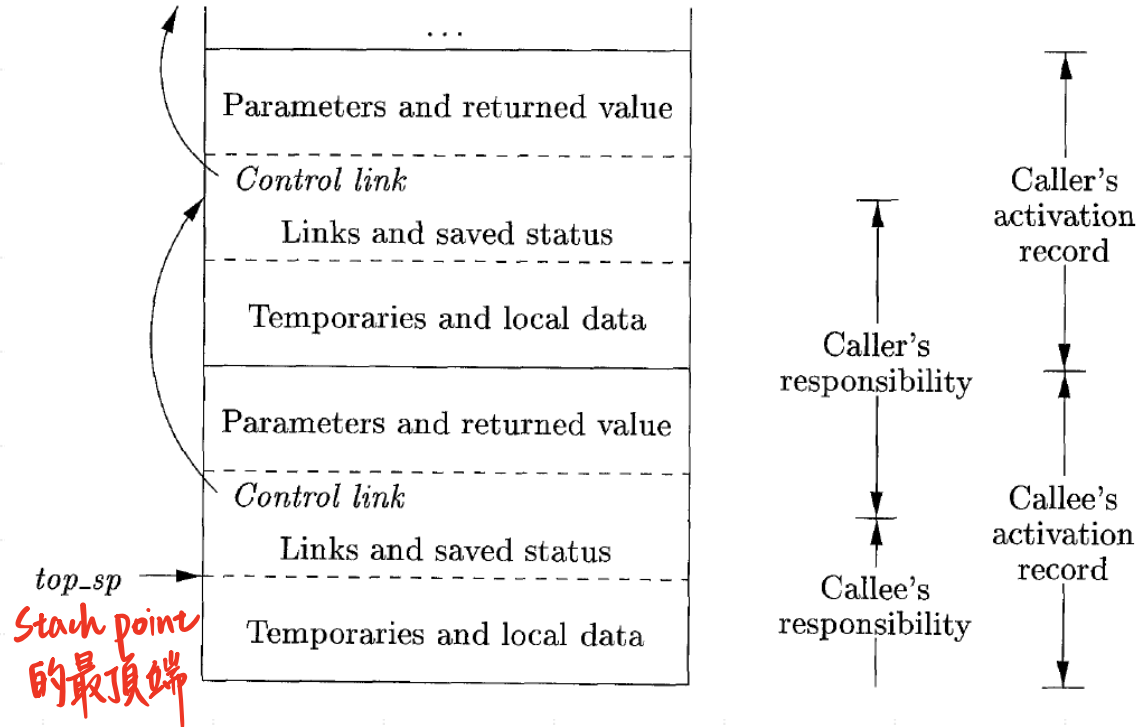
- 
- Each execution of a procedure is referred to as an activation of the procedure. If the procedure is recursive, several of its activation may be alive at the same time.
  - Activation Record
    - a contiguous block of storage recording the state of an activation of a function, not all the compilers set the same fields.



# What is the contents of the activation record?

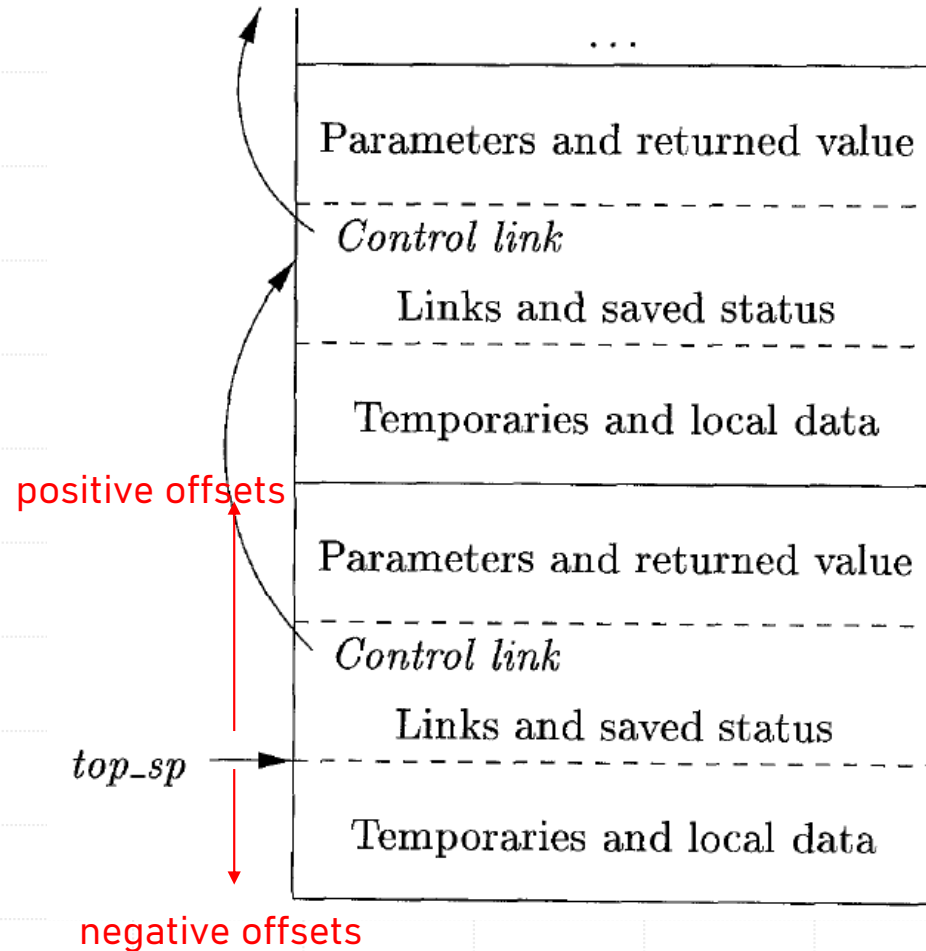
- temporary values
- local data
- saved machine status
- optional access link to refer to non-local data held in other activation records
- optional control link to refer to the activation
- actual parameters
- returned value

# General Organization of an Activation Record



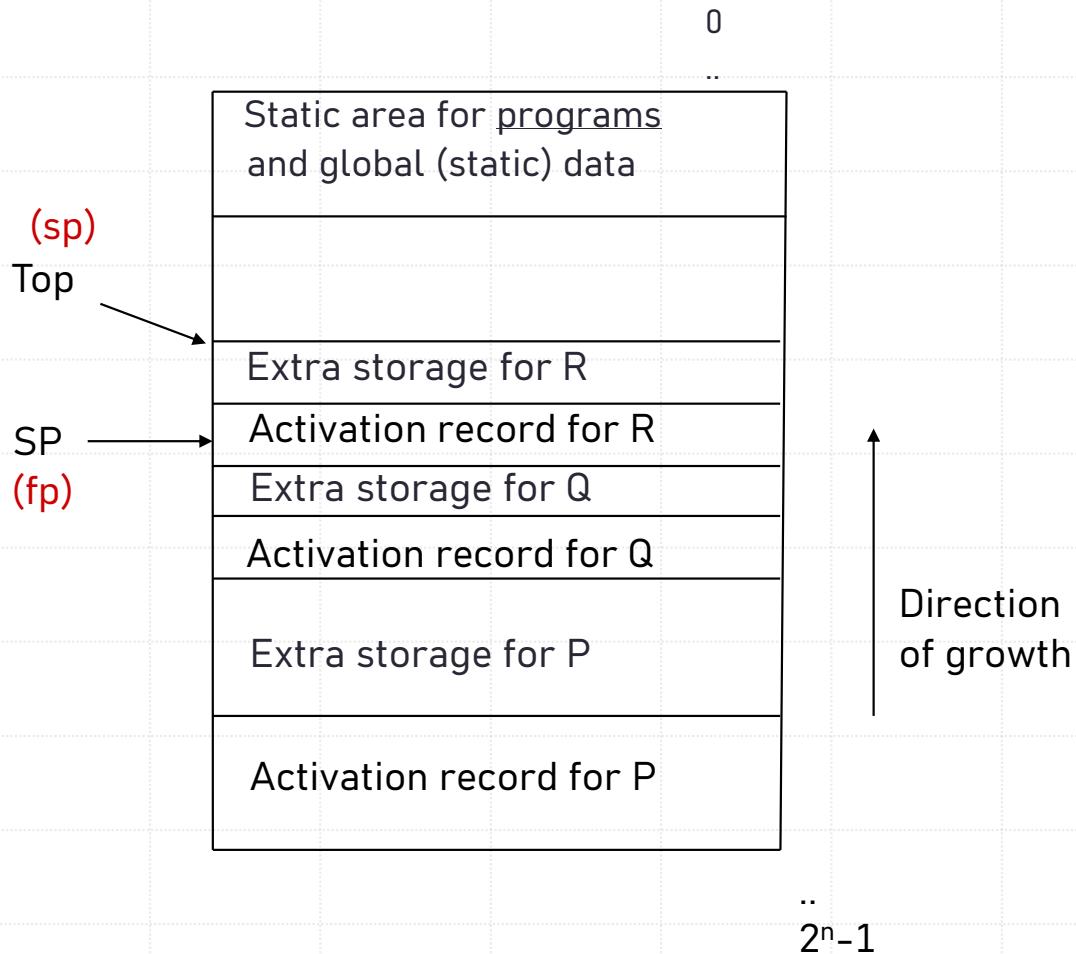



- So, local data are accessed by negative offsets from SP (stack pointer, a register holding a particular position of currently activated procedure).
- A local name  $X$  can be referenced by  $X[SP]$  ( $[SP]$  means the address of SP), where  $X$  is the (negative) offset for the name from the location pointed to by SP.
- Parameter can be referred to by a positive offset from SP. That is, the  $i$ th parameter can be referenced by  $(4+n-i)[SP]$  (assume each entry takes one unit of space)



- Suppose the Memory structure for C program is organized as below, where function P calls Q and Q calls R

Procedure  
的最頂端



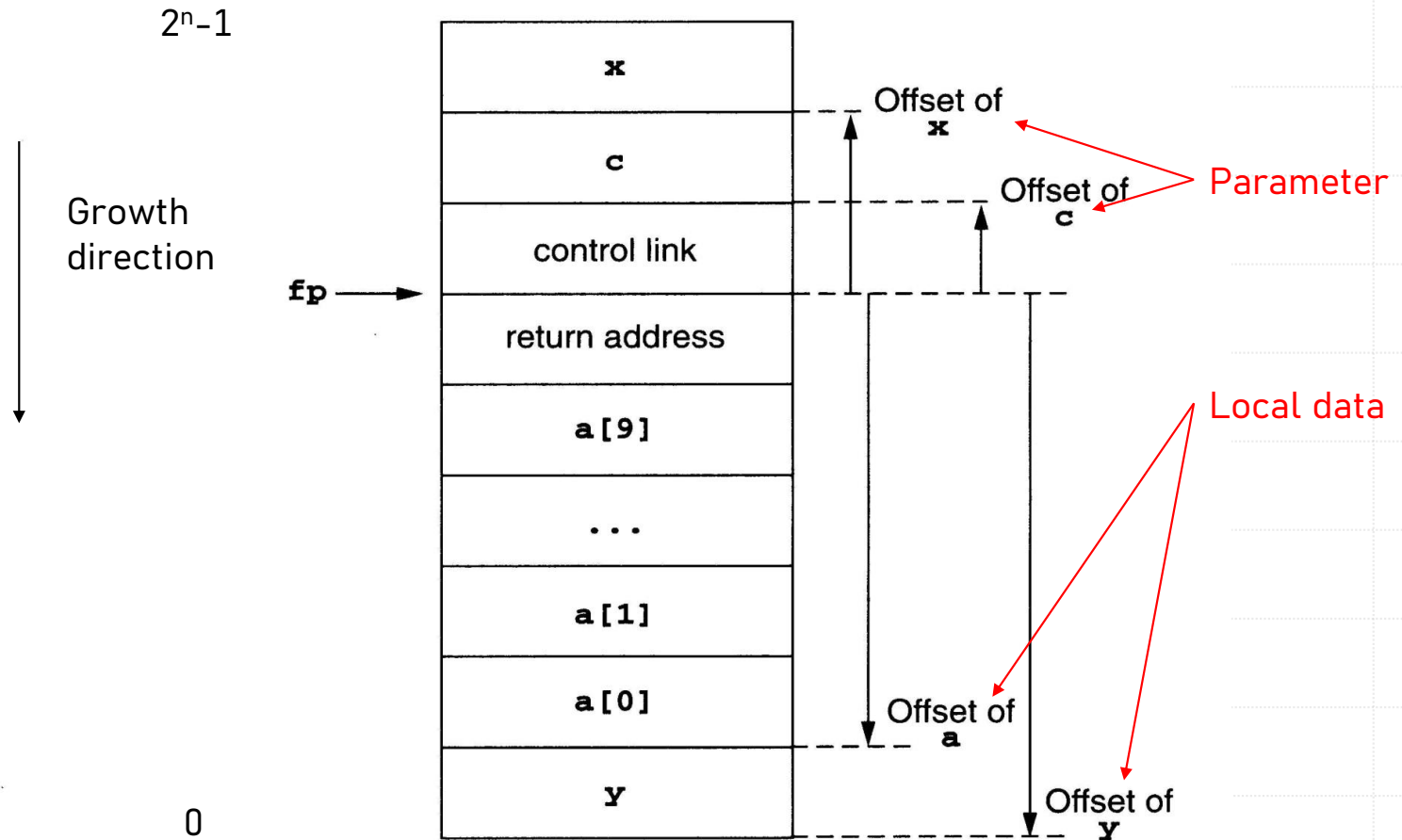
- 
- TOP is a register holding the top address of the stack. SP is also a permanently allocated register (stack pointer) holding a specific address in the activation record of the currently active procedure. We use SP to indirectly refer to the local variables, parameters, etc.
  - Extra storage is used for storing pointers on the display, variable-length data such as dynamically allocated arrays, etc.

## Example 7.4

Consider the C procedure

```
void f(int x, char c)
{ int a[10];
  double y;
  ...
}
```

The activation record for a call to **f** would appear as



and, assuming two bytes for integers, four bytes for addresses, one byte for characters, and eight bytes for double-precision floating point, we would have the following offset values (again assuming a negative direction of growth for the stack), which are all computable at compile time:

Name	Offset
<b>x</b>	+5
<b>c</b>	+4
<b>a</b>	-24
<b>y</b>	-32

Now an access of, say, **a[i]** would require the computation of the address

$$(-24 + 2 * i) \text{ (fp)}$$


(here the factor of 2 in the product **2\*i** is the **scale factor** resulting from the assumption that integer values occupy two bytes). Such a memory access, depending on the location of **i** and the architecture, might only need a single instruction. §

### At run-time

e.g.  $p(T_1, T_2, \dots, T_n) \Rightarrow$

param T1	$\Rightarrow \text{push}(T_1)$
param T2	$\Rightarrow \text{push}(T_2)$
...	..
...	..
param Tn	$\Rightarrow \text{push}(T_n)$
call p, n	$\Rightarrow \text{push}(n)$
	$\Rightarrow \text{push}(l_1)$
	$\Rightarrow \text{push}()$
	$\Rightarrow \text{push}(SP)$
	$\Rightarrow \text{goto } l_2$

$l_1$  denotes the return address;  $l_2$  denotes the address of the first statement of the called procedure.



Temporary variable

e.g.,  $p(A+B*C, D) \Rightarrow$

$T1 = B * C$

$T2 = A + T1$

Param T2

Param D

Call p, 2



## At run-time

- \* Assume TOP points to the lowest-numbered used location on the stack and the memory locations are counted by words.

'param T' is translated into 'push(T)' which stands for

TOP = TOP - 1; /\* now TOP points to an available entry \*/

\*TOP = T; /\* save T into the memory

'call p, n' is translated into the following instructions:

push (n) /\*store the argument count \*/

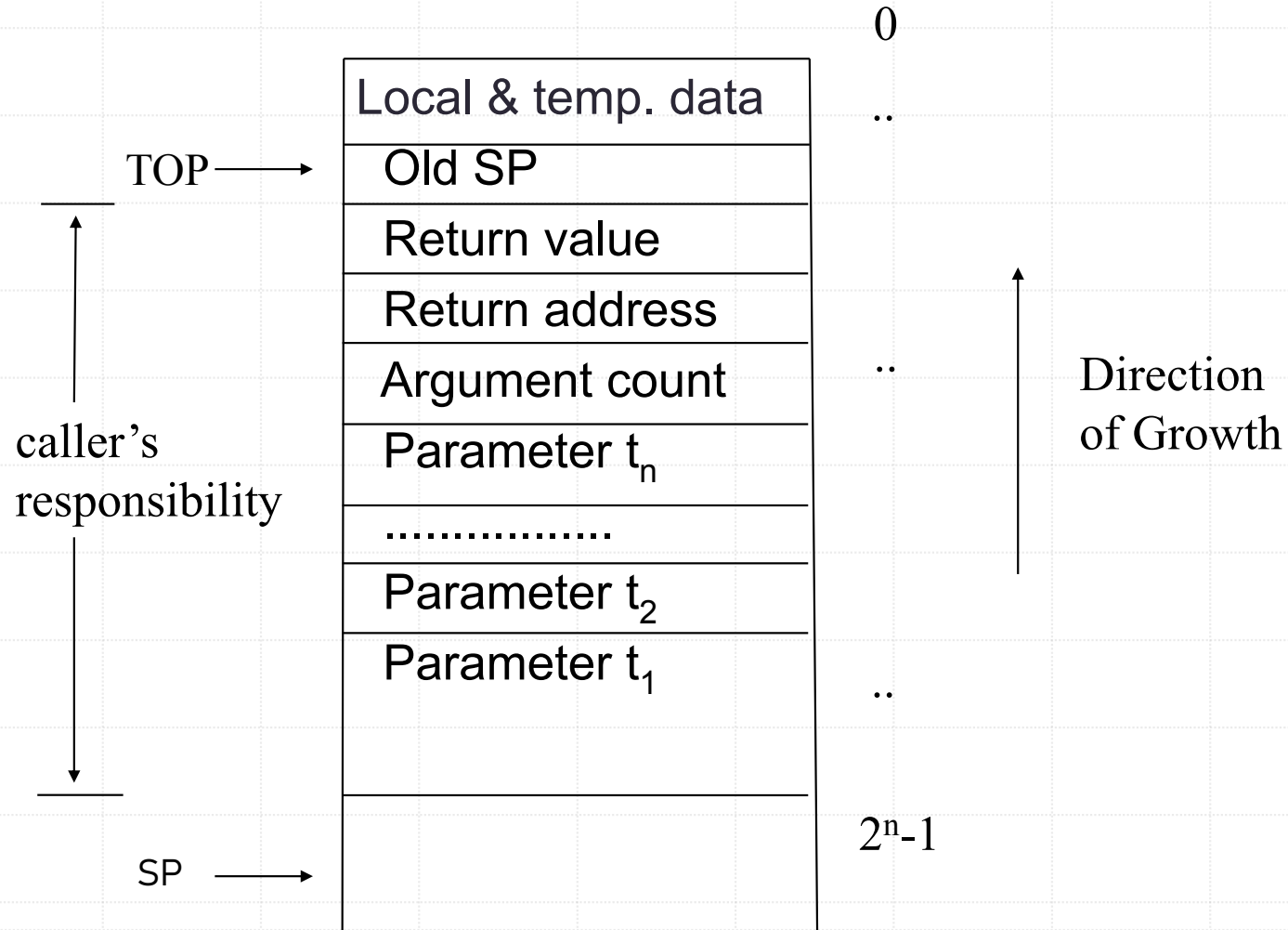
push ( $l_1$ ) /\*  $l_1$  is the return address \*/

push () /\* leave one space for the return value \*/

push (SP) /\* store the old stack pointer \*/

goto  $l_2$  /\*  $l_2$  is the first statement of the called procedure p \*/

# General Organization of an Activation Record



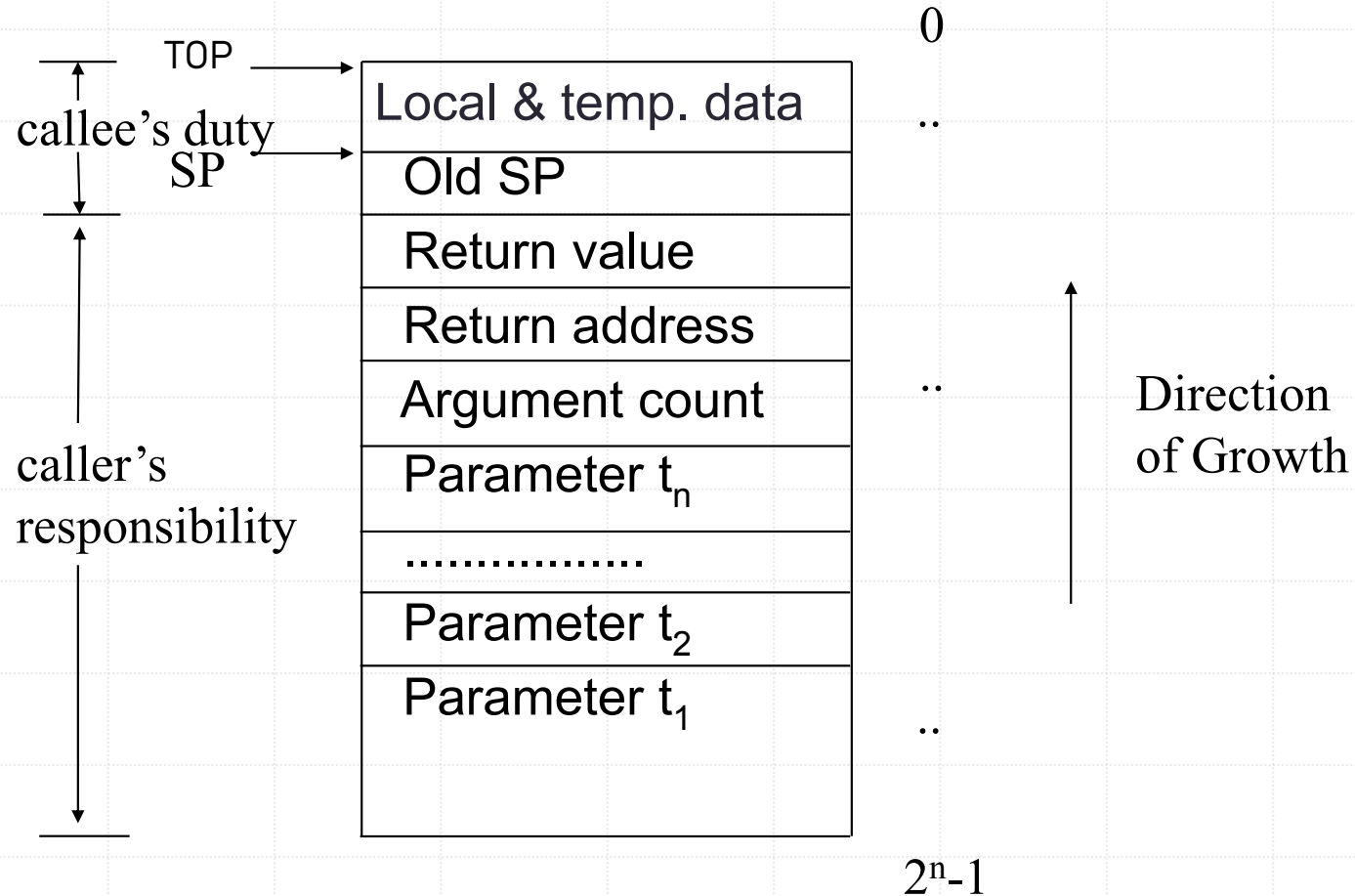
## The first statement of the called procedure

- The first statement of the called procedure must be a special three-address code 'procbegin', which (1) sets the stack pointer to the place holding the old SP and (2) sets TOP to the top of the activation record (or the stack), so 'procbegin' stands for:

SP = TOP; // now SP points to old SP value

TOP = SP + size\_of\_p; /\* size\_of\_p is the size of p,  
i.e., the number of words  
taken by the local data for  
p \*/

# General Organization of an Activation Record



## The 'return' statement

The *return statement* in c can have the form '**return (expression);**'

This statement can be implemented by three-address code to evaluate the expression into a temporary  $T$  followed by:

```
1[SP] = T      /* 1 is the offset for the location of the return value */
TOP = SP + 2    /* TOP now point to the return address */
SP = *SP       /* restore SP, SP now points to old location */
L = *TOP        /* the value of L is now the return address */
TOP = TOP + 1   /* TOP points to the argument count */
TOP = TOP + 1 + *TOP /* *TOP is the number of parameter of
                  P. We restore TOP to the top of extra
                  storage for the activation record below */
go to *L
```