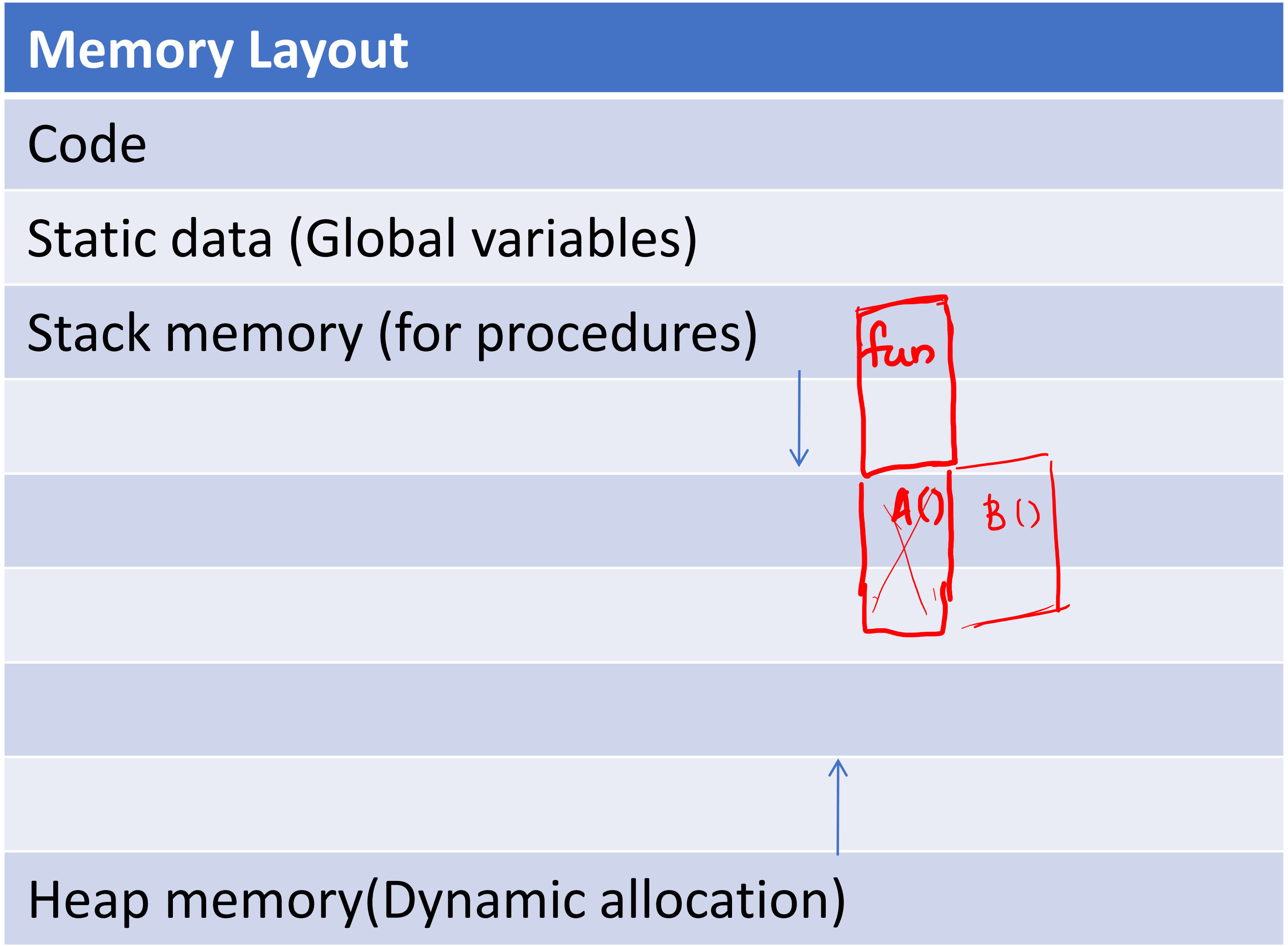
Run-time storage management

Run-time storage management

- We need to use memory to store:
  - code
  - static data (global variables)
  - dynamic data objects
    - data that are used when executing a certain procedure.
    - Dynamically allocated objects (malloc, free).

Run-time memory



Memory Layout

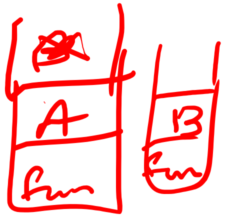
Code

Static data (Global variables)

Stack memory (for procedures)

Heap memory(Dynamic allocation)

```
fun ( )
{
  A ( )
  → B ( )
}
```

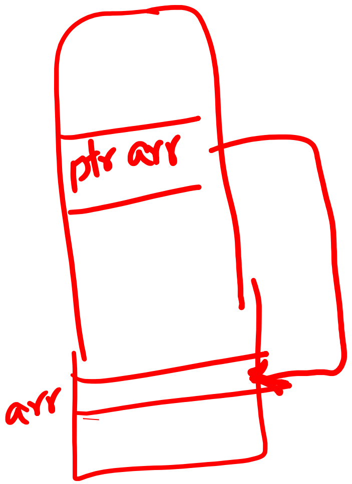


Activation Record

- also called frames
- Information needed by a single execution of a procedure
- A general activation record has seven fields

Activation record fields

```
{
  int a=10;   (1)
   int c=5;   (1)
  {
    int a=5;  (2)
    int b=2;  (2)
        c
  }
  printf(a,b,c);
}
```



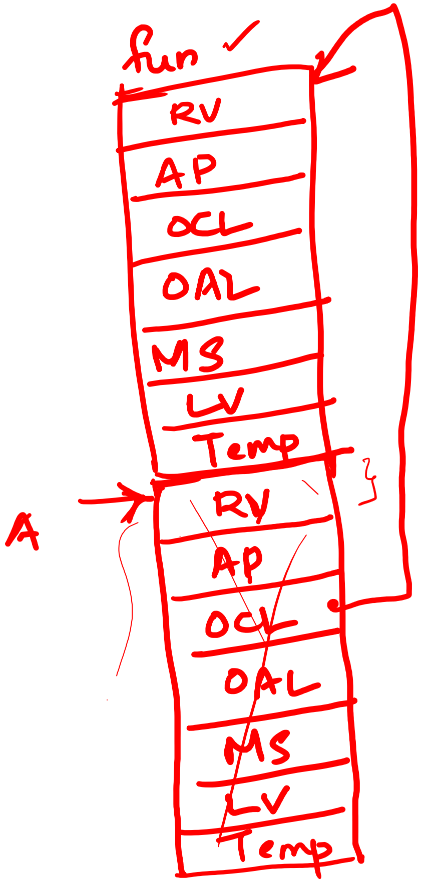
Activation record
Return value
Actual parameters
Optional control link
Optional access link
Machine status PC
Local variables
Temporaries

i ;

caller

calle

```
fun ( )
{
  c=A (b);
}
```



RO

```
A ( — )
{
  int r,a,n;
  int arr[n];
}
```

n=10;

Storage – memory

- Static allocation
- Stack allocation
- Heap allocation

Static allocation

- Lays out storage for all data objects at compile time.
    - Constraints:
        - size of object must be known and alignment requirements must be known at compile time.
        - No recursion.
        - No dynamic data structure

Stack allocation

- Stack allocation manages the run time storage as a stack
    - The activation record is pushed on as a function is entered.
    - The activation record is popped off as a function exits.
    - Constraints:
        - values of locals cannot be retained when an activation ends.
        - A called activation cannot outlive a caller.

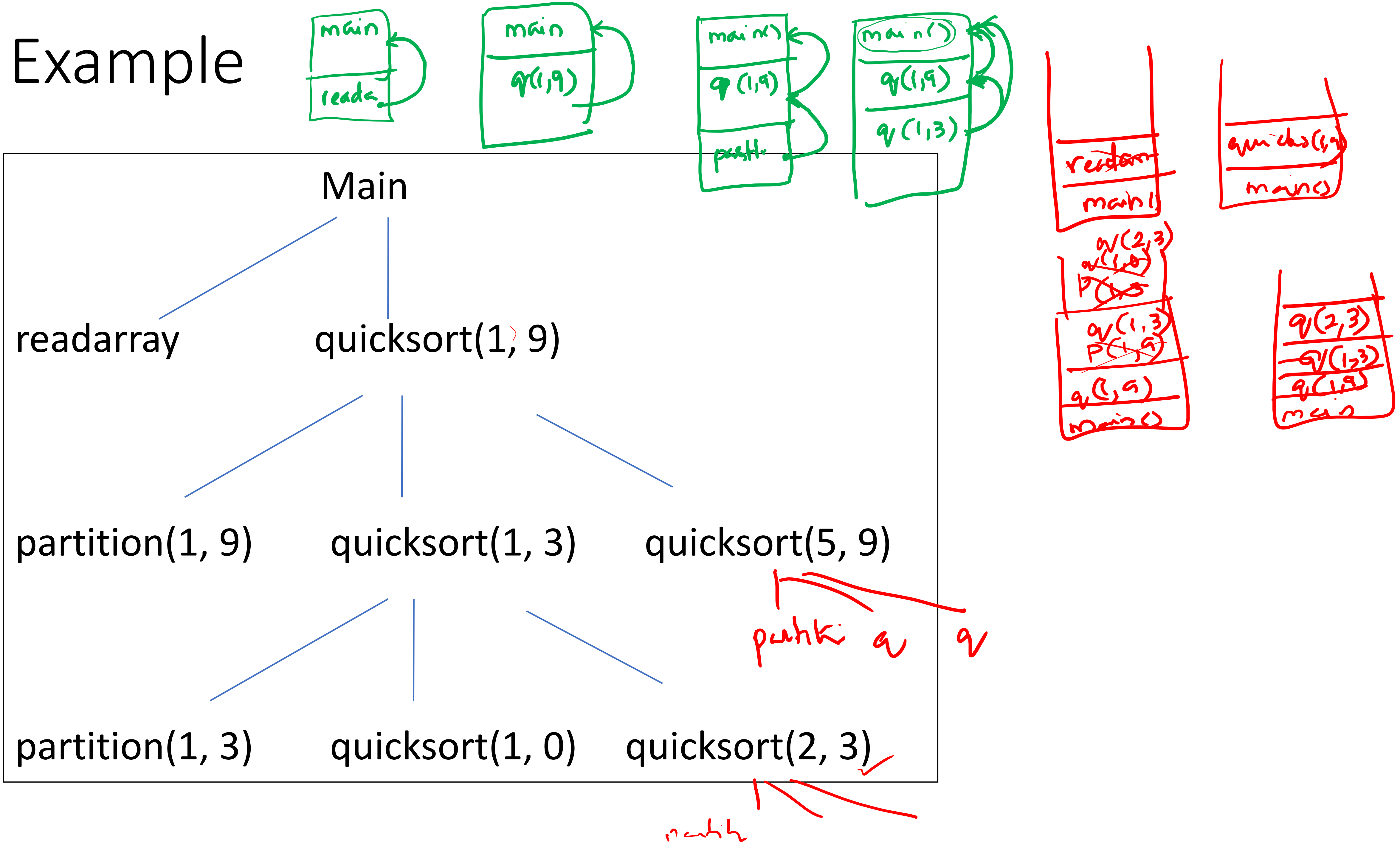
Heap allocation

- Heap allocation -- allocates and deallocates storage as needed at runtime from a data area called as heap.
  - Does not require the activation of procedures to be LIFO.
  - Requires true dynamic memory management.

Example

```
Program sort
var
   procedure readarray;
   ….
   function partition(…)
   ….
   procedure quicksort(…)
      ……
      partition   ✓   __ 4 __
      quicksort
      quicksort
      ….
    Begin
       ……
       readarray
       quicksort
    end
```

Example



Main

readarray

quicksort(1, 9)

partition(1, 9)

quicksort(1, 3)

quicksort(5, 9)

partition(1, 3)

quicksort(1, 0)

quicksort(2, 3)

Example

- How is stack memory managed?
    - Everything must be done by the compiler.
    - What makes this happen is known as **calling sequence** (how to implement a procedure call).
        - A calling sequence allocates an activation record and enters information into its fields (push the activation record).
    - On the opposite of the calling sequence is the **return sequence.**
        - Return sequence restores the state of the machine so that the calling procedure can continue execution.

Calling sequence

- The caller evaluates actuals and push the actuals on the stack
- The caller saves return address(pc) the old value of sp into the stack
- The caller increments the sp
- The callee saves registers and other status information
- The callee initializes local variables & begin execution.

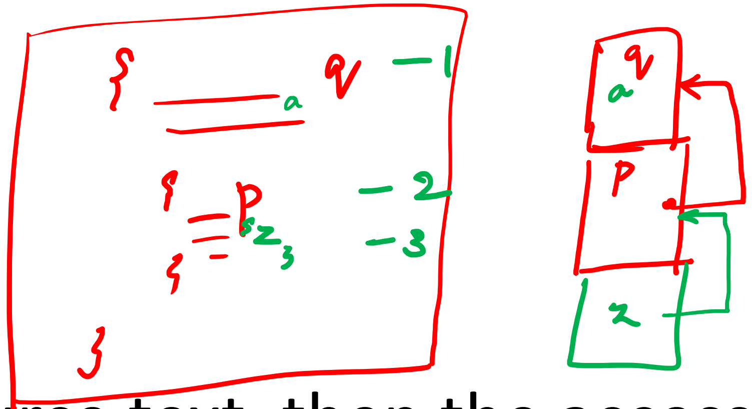
Return sequence

- The callee places a return value next to the activation record of the caller.
- The callee restores other registers and sp and return (jump to pc).
- The caller copies the return value to its activation record.

Access to non-local variables

- Nonlocal variables in C (without nested procedures):
  - Still have nested scopes (blocks).
  - Solution:
    - All data declared outside procedures are static.
    - Other names must be at the activation record at the top of the stack, can be accessed from sp.
      - Treat a block as a parameter-less procedure
      - Allocates space for all blocks in a procedure.

Access to non-local variables



- If p is nested immediately within q in the source text, then the access link in an activation record for p points to the access link in the record for the most recent activation of q.
- A procedure p at nesting depth n_p accesses a nonlocal a at nesting depth n_a: (1) following n_p – n_a links and (2) using the relative offset in the activation record.

Parameter Passing

- The method to associate actual parameters with formal parameters.
- The parameter passing method will effect the code generated.

Techniques

- Call by value
- Call by reference
- Call by copy-restore
- Call by name

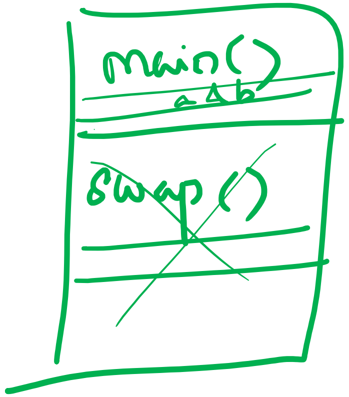
$a = \boxed{10};$

Call by value

- The actual parameters are evaluated and their r-values are passed to the called procedure.
- Implementation:
  - a formal parameter is treated like a local name, so the storage for the formals is in the activation record of the called procedure.
  - The caller evaluates the actual parameters and places their r-values in the storage for the formals.

Example – C & Pascal

```
Swap(int a, int b)
{  int temp;
   temp = a; a = b; b = temp;
}
Void main()
{int a = 1, b = 2;
Swap(a, b); printf("%d \t %d", a, b);
}
```

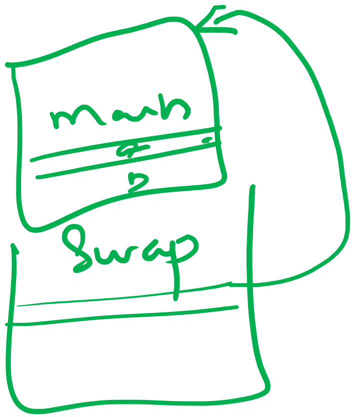


Call by reference

- Referred as call-by address or call-by-location.
- The caller passes to the called procedure a pointer to the storage address of each actual parameter.
    - Actual parameter must have an address -- only variables make sense, an expression will not

Example C++

```
Swap(int * a, int * b)
{  int temp;
   temp = *a; *a = *b; *b = temp;
}
void main()
{
int a = 1, b = 2;
Swap(&a, &b); printf(“%d \t %d”, a, b);
}
```



Call by Copy-Restore

- A hybrid between call-by-value and call-by-reference.
  - The actual parameters are evaluated and its r-values are passed to the called procedure as in call-by-value.
  - When the control returns, the r-value of the formal parameters are copied back into the l-value of the actuals.

a , b

Example

- Swap(i, a[i]) works correctly using copy-restore
- Location of a[i] is computed and preserved by the calling program before initiating the call
- Used by Fortran

Call by name

- Defined by the copy-rule of Algol
  - Procedure is considered as if it were a macro. The actual parameters are literally substituted with the formal as a macro-expansion
  - Local names of called procedures are kept distinct. May be renamed
  - Actual parameters are surrounded by parentheses to preserve integrity

Example

- x : = f(A) + f(B)
- A , B are expressions
- Substitution of expressions A and B in the formal parameter leads to call by name

Summary

- Run-time storage management
- Static, Stack, Heap allocation
- Parameter Passing techniques