Programming Language Concepts

Programming Language Theory

Topics

- Memory Management
  - Static Management
  - Dynamic Management w/ Stack
  - Dynamic Management w/ Heap
  - **Scope Rule Implementation**

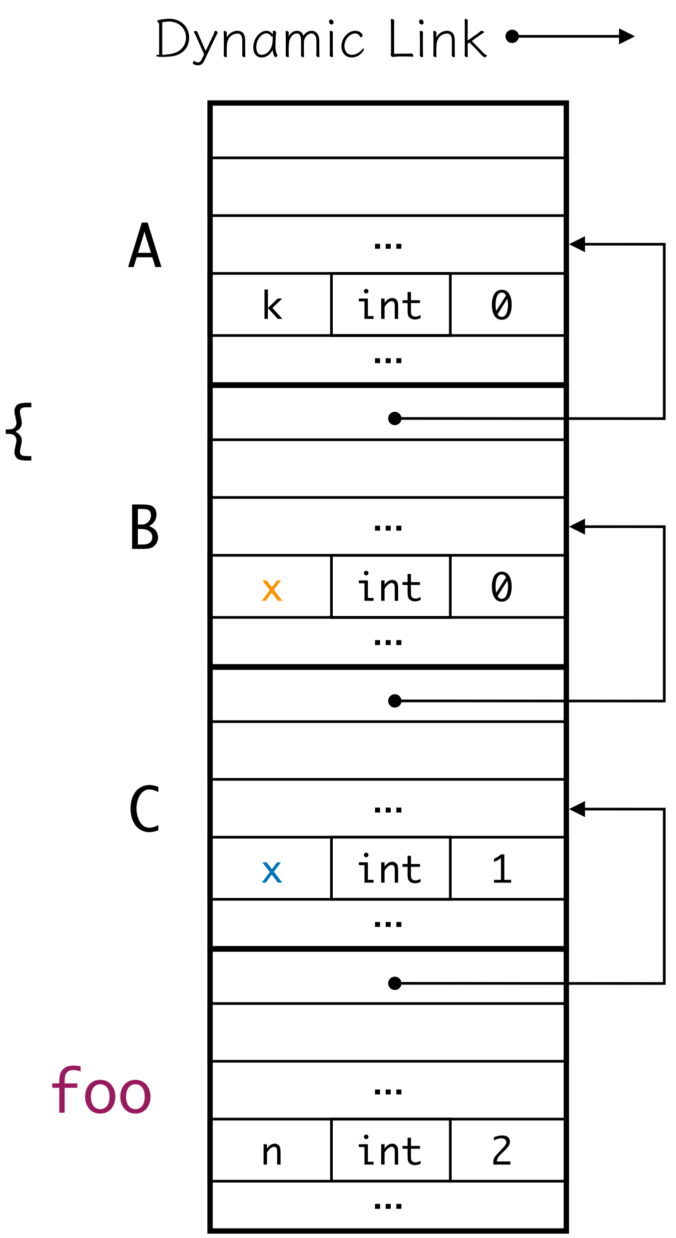
Scope Rule Implementation

- Static Scope Rule Implementation
  - Static Link
  - The Display
- Dynamic Scope Rule Implementation
  - Association Lists and CRT

Static Scope Rule Implementation

- Static scope rule implementation requires more management than simply using the stack.
- In procedure foo, variable x refers to x in Block B, not C.
- However, the activation record connected to foo with dynamic link is Block C.

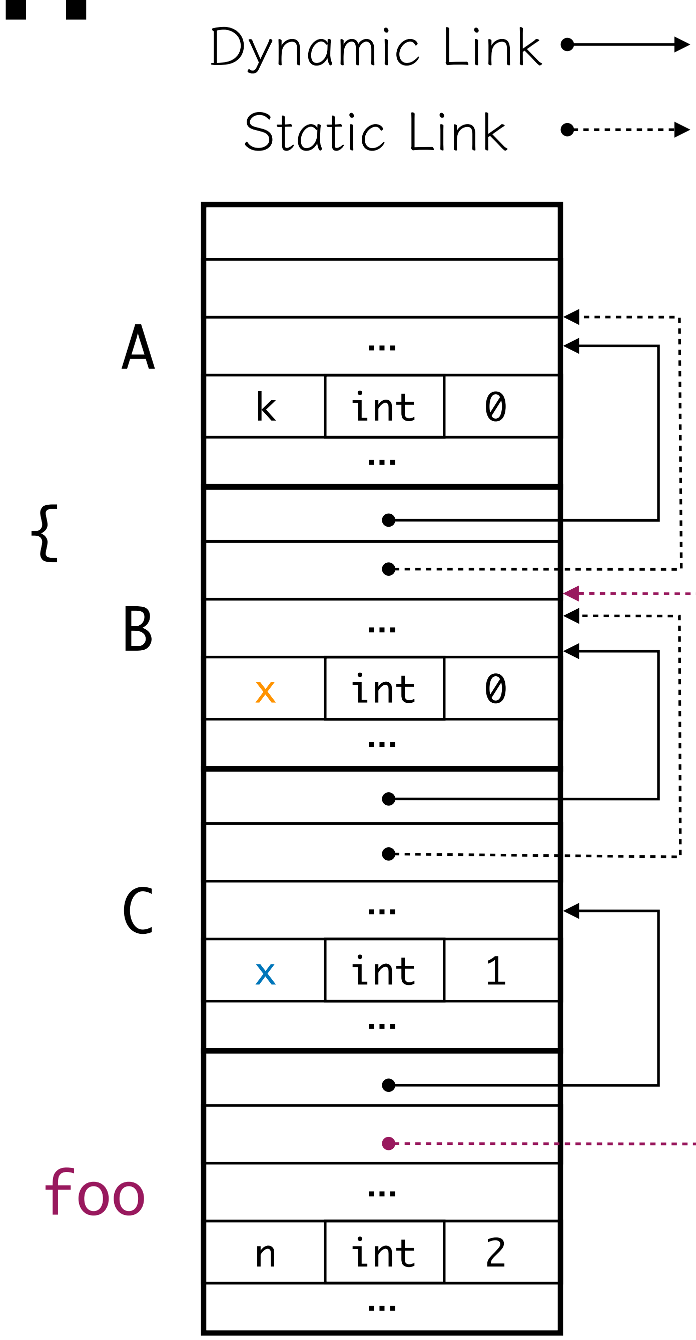
```
A:{ int k=0;
    B:{
        int x=0;
        void foo(int n) {
            x = n-1;
            k = n+1;
        }
        C:{
            int x=1;
            foo(2);
        }
    }
}
```



Static Scope Rule Implementation

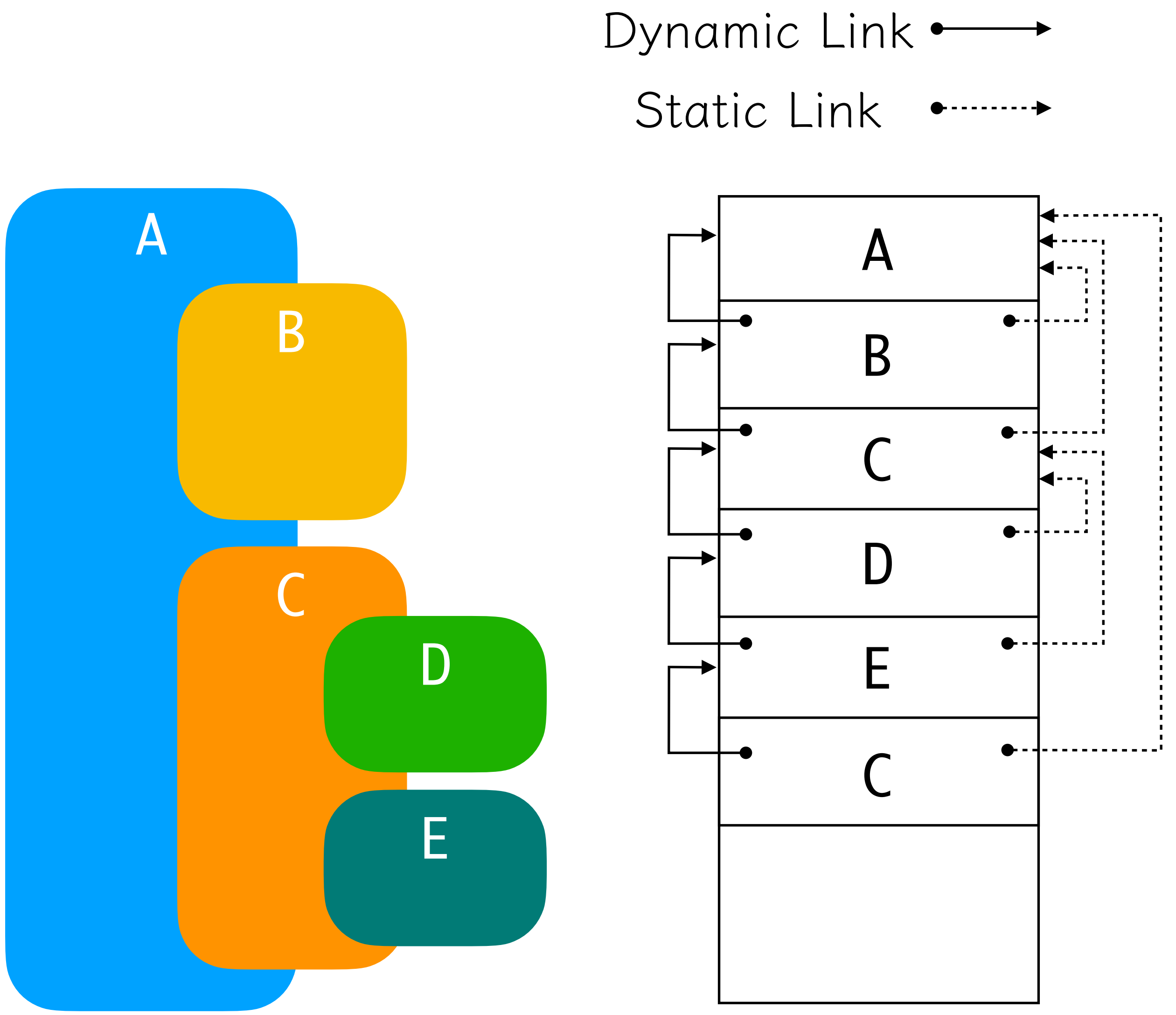
- So we need one more link to the block immediately enclosing the block.
- Using ***static link*** (dotted line) to point the enclosing block.
- When managing static scope, use static links instead of dynamic links.

```
A:{ int k=0;
  B:{
    int x=0;
    void foo(int n) {
      x = n-1;
      k = n+1;
    }
    C:{
      int x=1;
      foo(2);
    }
  }
}
```



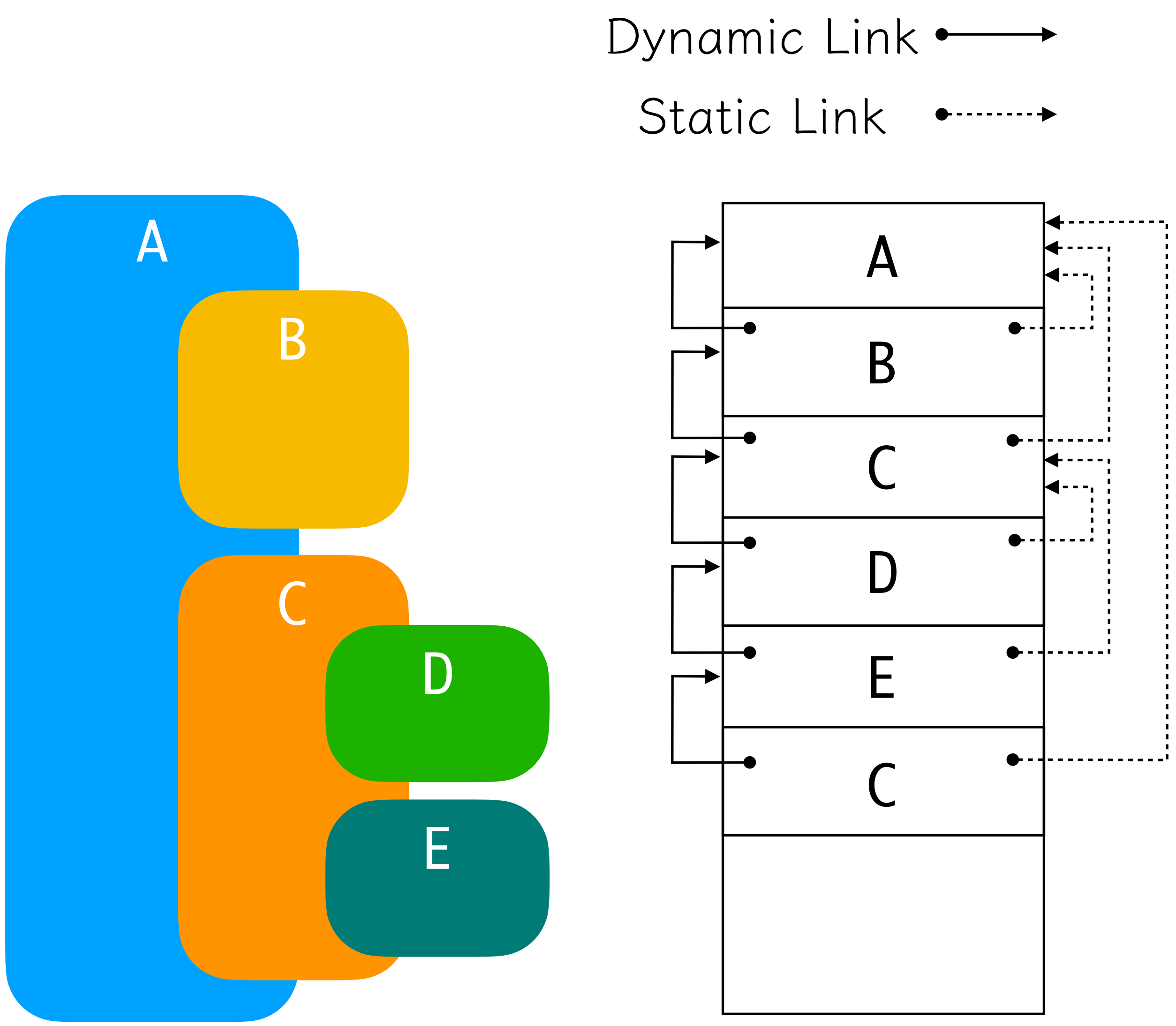
Static Link

- Consider an example with the following structures.
- Blocks B, C are inside Block A.
- Blocks D, E are enclosed in Block C.
- There is a sequence of calls,
    - A, B, C, D, E, C



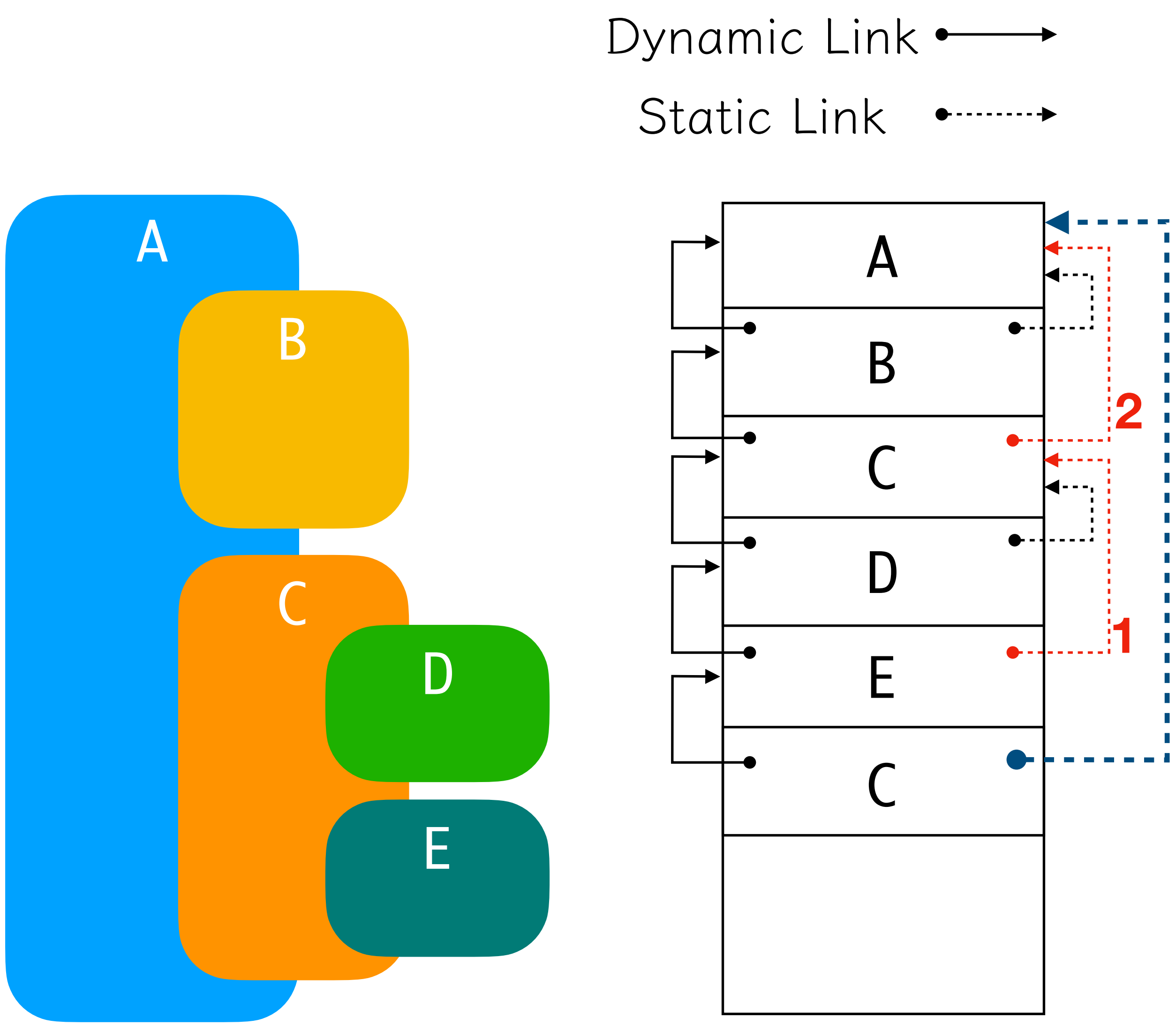
Static Link

- When activation records are pushed to the stack, it is necessary to decide the address for static link.
- In most common approach, the caller calculates the link and passes it to the callee.
- There are two possible cases.



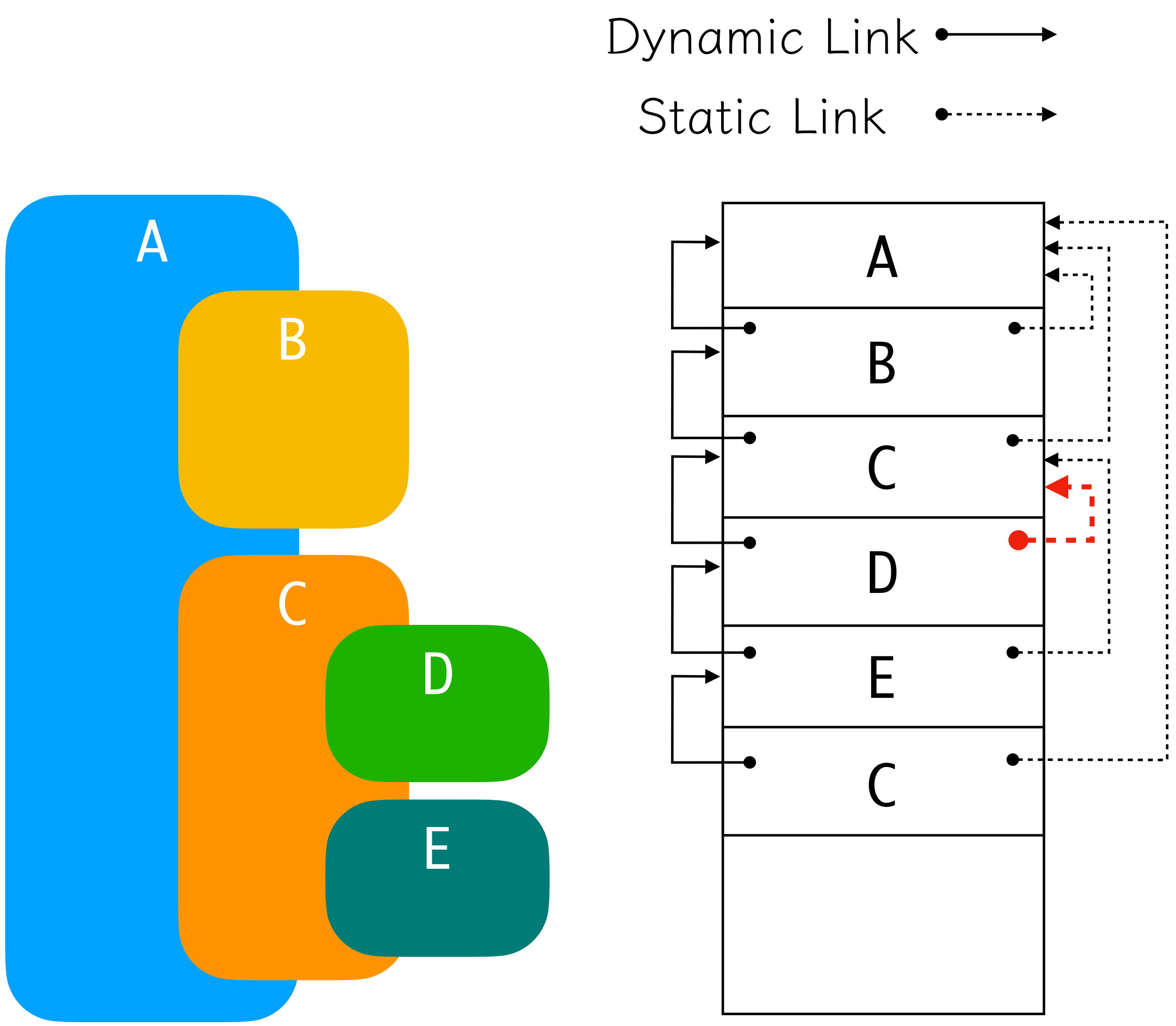
Static Link

- **Case 1**: Callee is outside of the caller (e.g. E calls C).
    - Callee must be in an outer block of the caller based on visibility rules.
    - Hence the activation record of the callee must be in the stack already.
    - So we can backtrace the static links to find a new link for the new block.



Static Link

- **Case 2:** Callee is inside the caller (e.g. C calls D).
  - Visibility rules guarantee that the callee is declared in the same block which the call is occurred.
  - Hence we can simply use a static link to the caller.



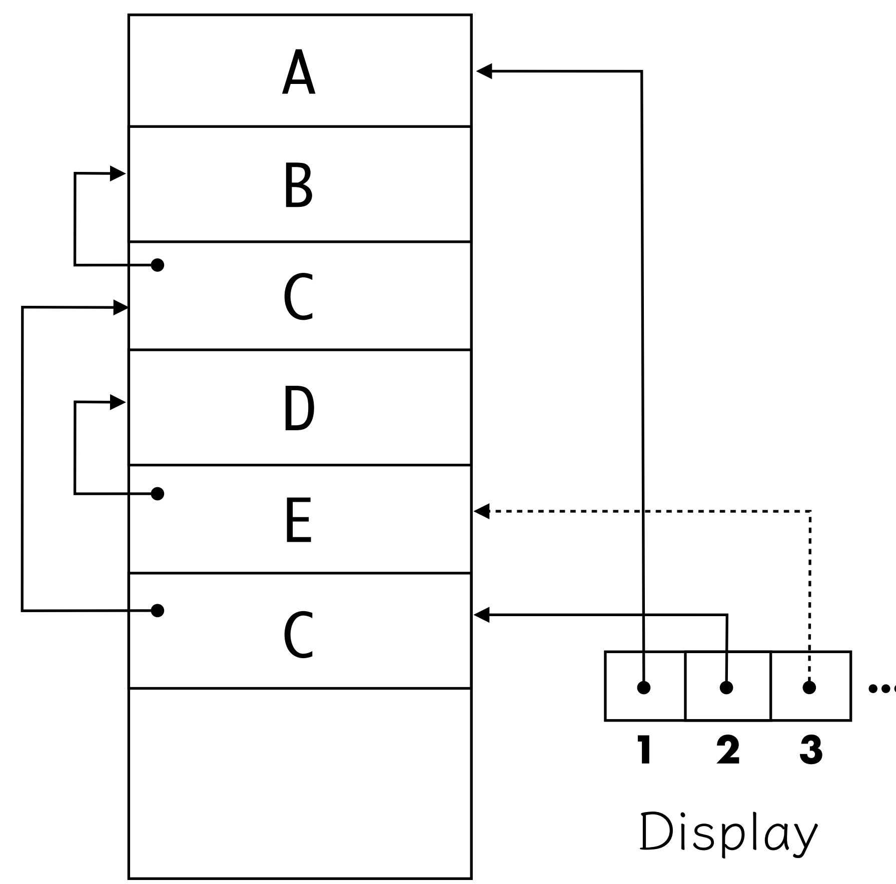
The Display

- Static Link requires several memory access for each procedure call.
- If a non-local name is declared *k* levels of block away, we need *k* memory accesses to follow the static links.
- Although we usually don’t have too much nesting (i.e. k is not big), we can do better.
- The Display technique only requires ***constant memory accesses (twice)***.

The Display

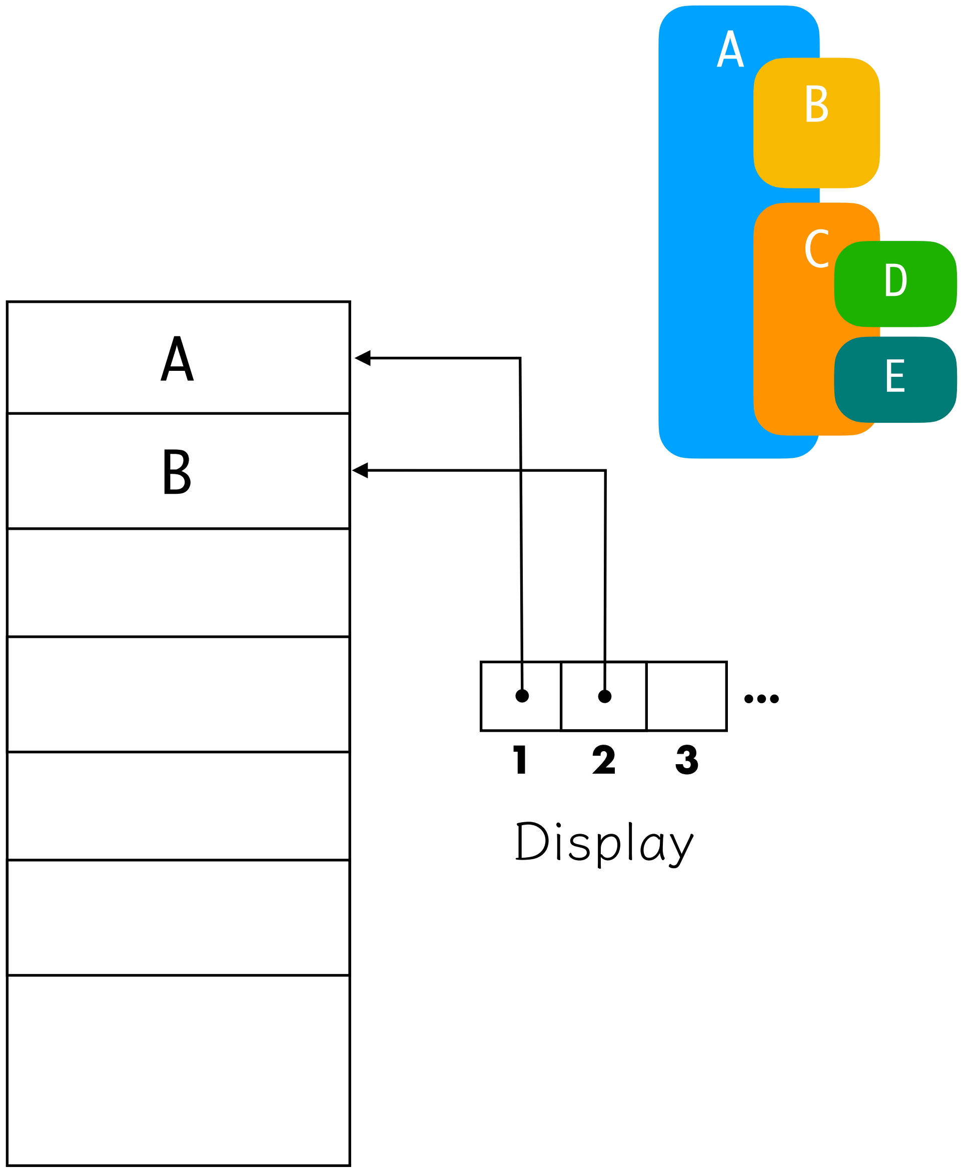
- The technique employs a vector called ***display***, whose k-th element contains the pointer to the current activation record of k-th nesting level.

- To find a non-local name declared in a block at level n, we can follow the pointer at element n, then use local offset to find the name.



The Display

- Display processing is simple.
- When a procedure is called at level k, display's k-th element is updated as the pointer to the activation record of callee.
- Let's consider the sequence of calls A, B, C, D, E, C again.

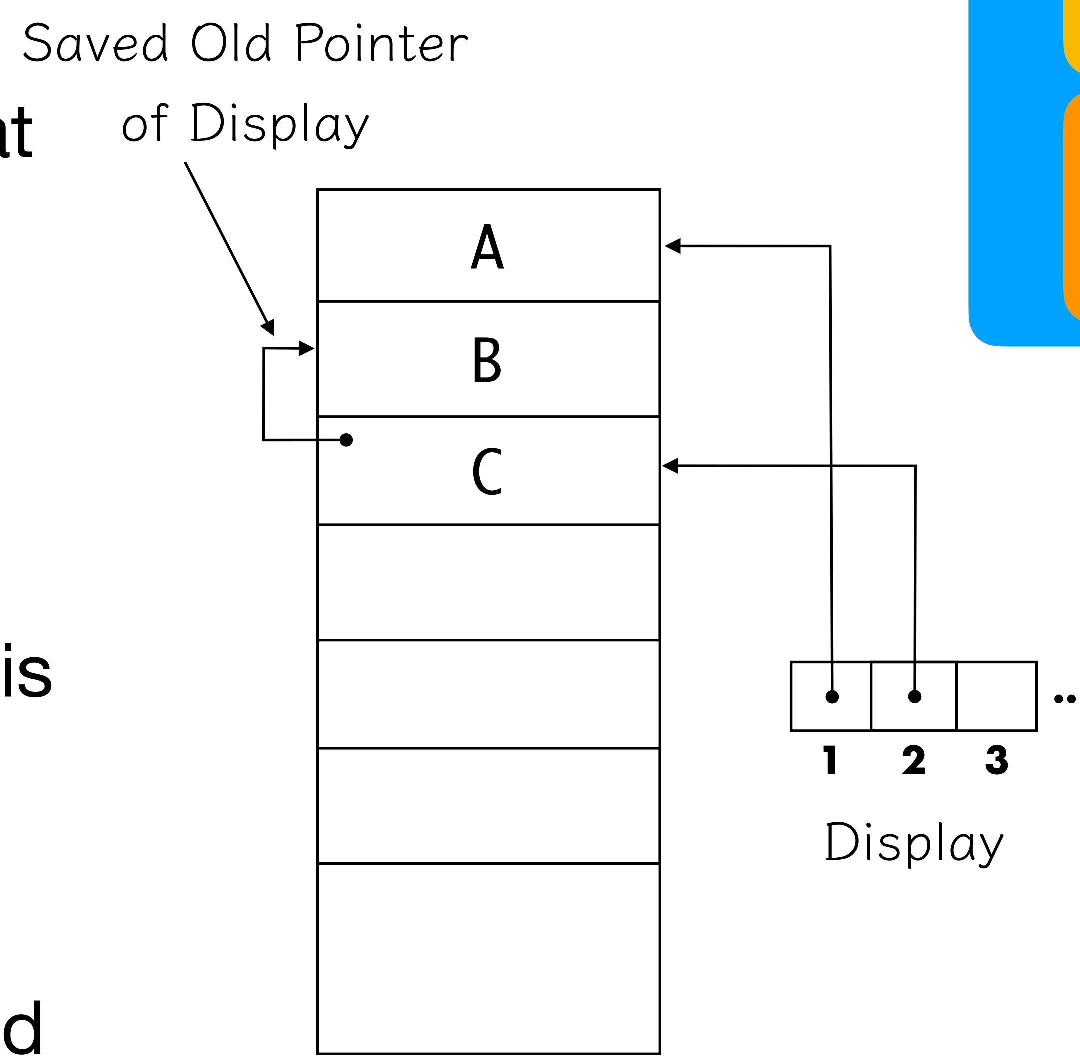


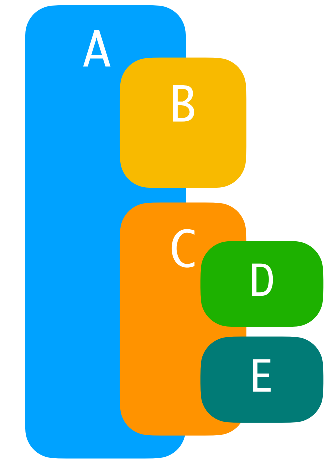
The Display

- When a procedure is called at the same level, ***it is necessary to store the old value*** in display.

- Since Block C is also at level 2, the old pointer to Block B is saved as a link from Block C to B.

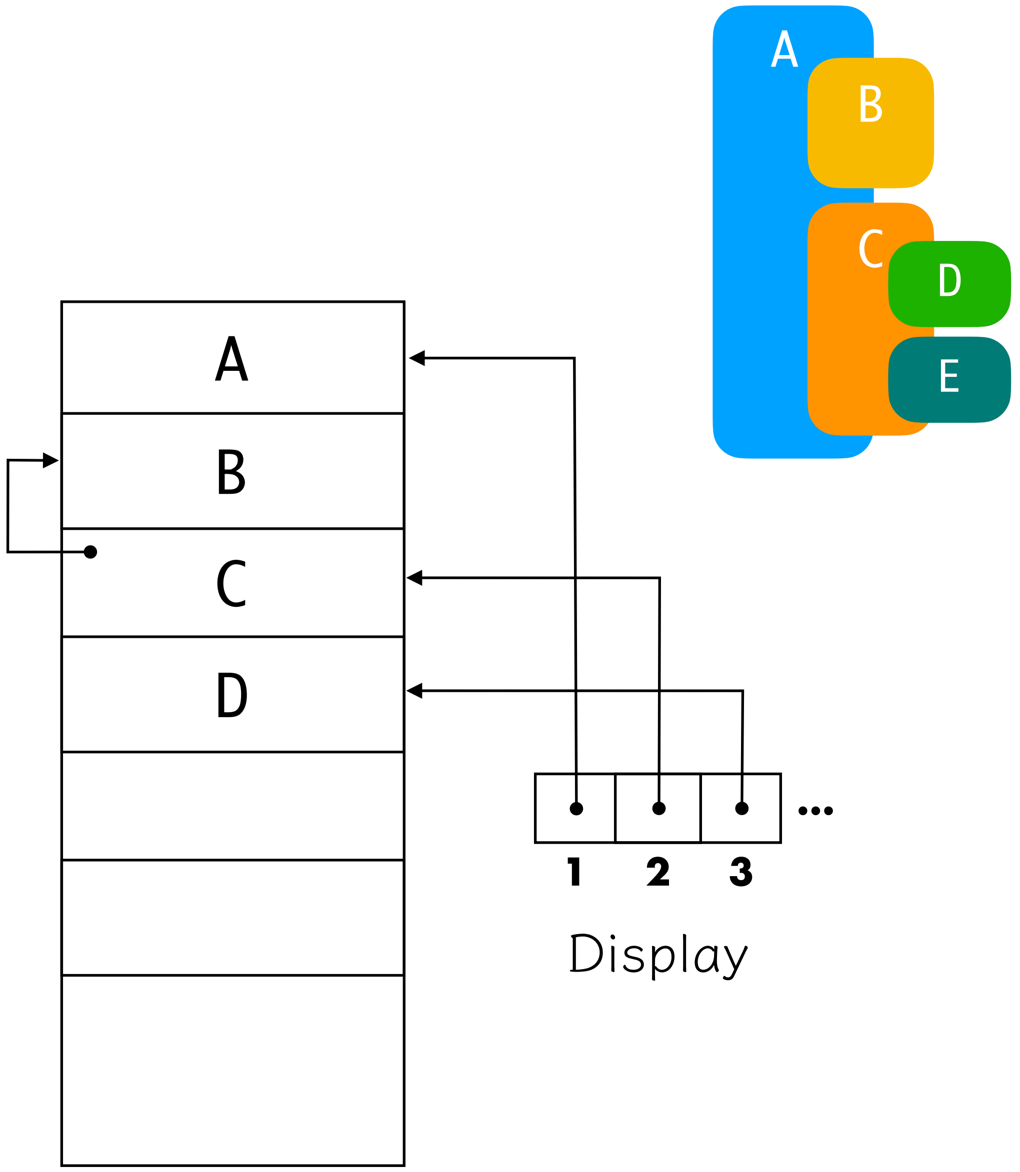
- This old value can be restored after C is out of the stack.





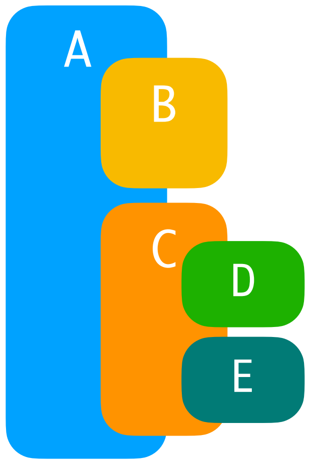
The Display

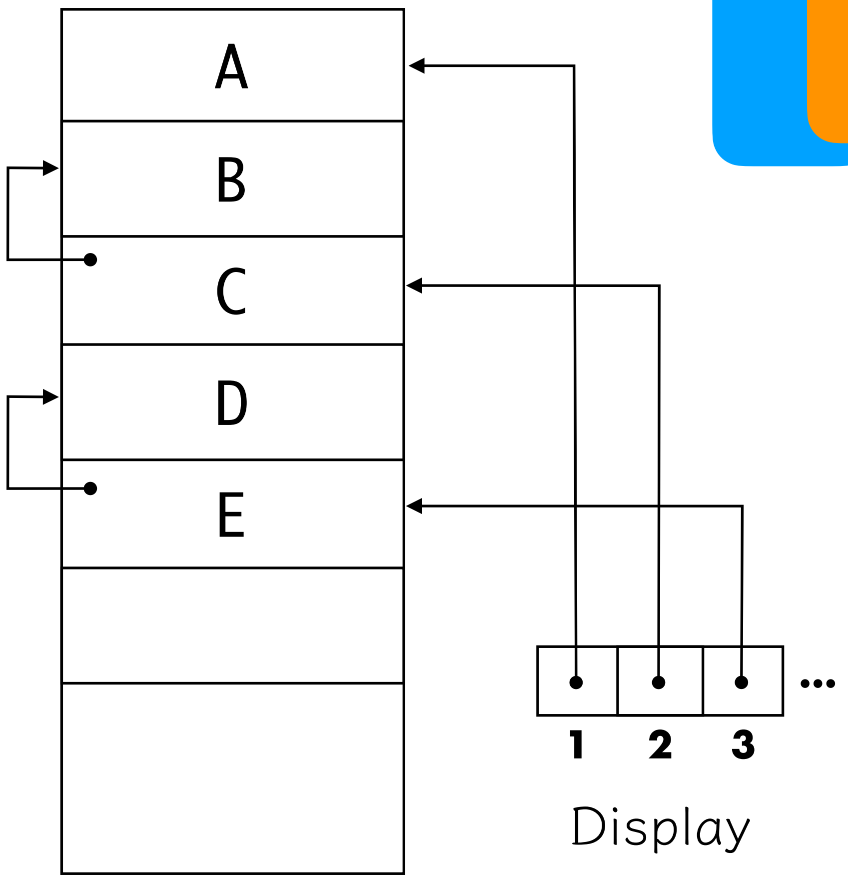
- If a callee (Block D) is inside the caller (Block C), we can increase the display length.
    - then we can put the pointer in the next element (3rd element).



The Display

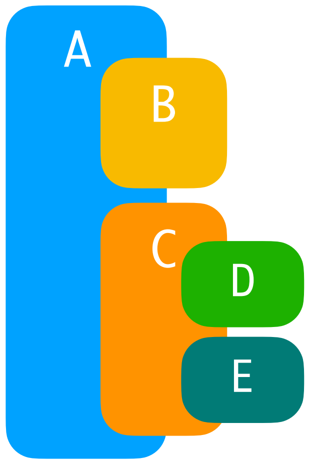
- As Block E is called, old value of element 3 is stored.
- Then the pointer is updated for Block E.
- Suppose variable x is declared in Block C and used in Block E.
  - We know that x is declared in C at compile time (static scope).
  - Block C is at level 2, hence check the display element 2.
  - Then use the local offset to find the binding of x.

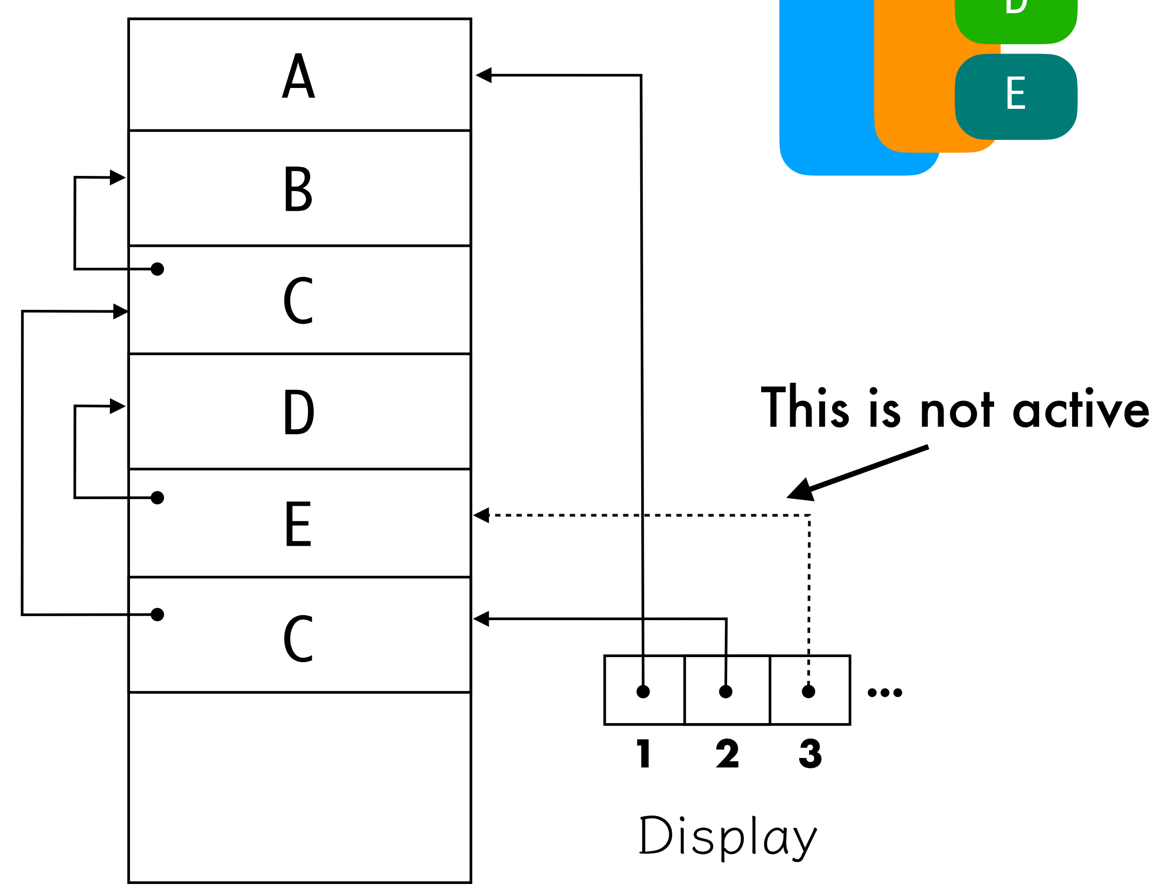




The Display

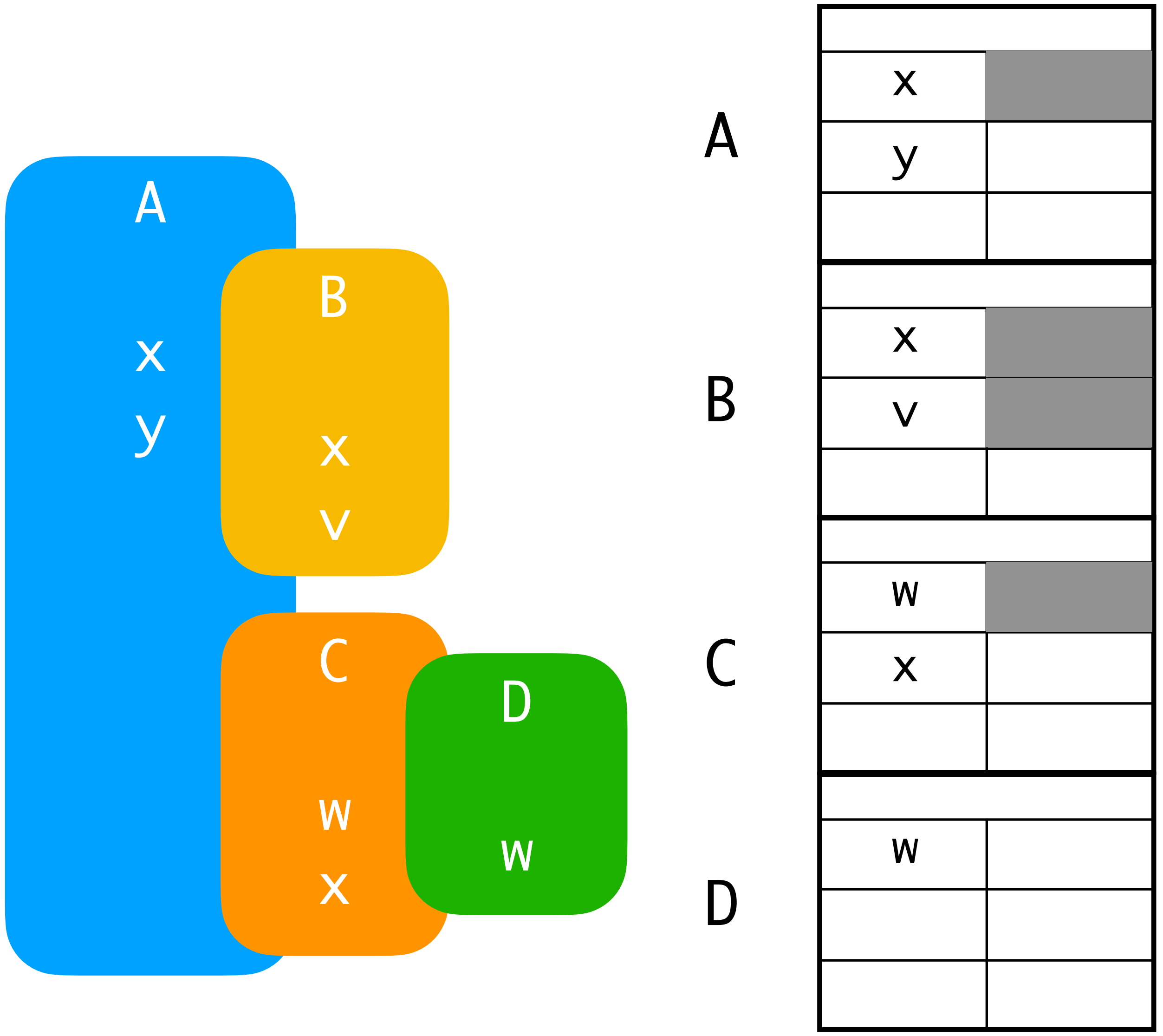
- Block C is called in Block E.
    - We cannot use local names declared in Block E in Block C.
    - Display contains the pointer to C at element 2, hence elements behind this are deactivated.





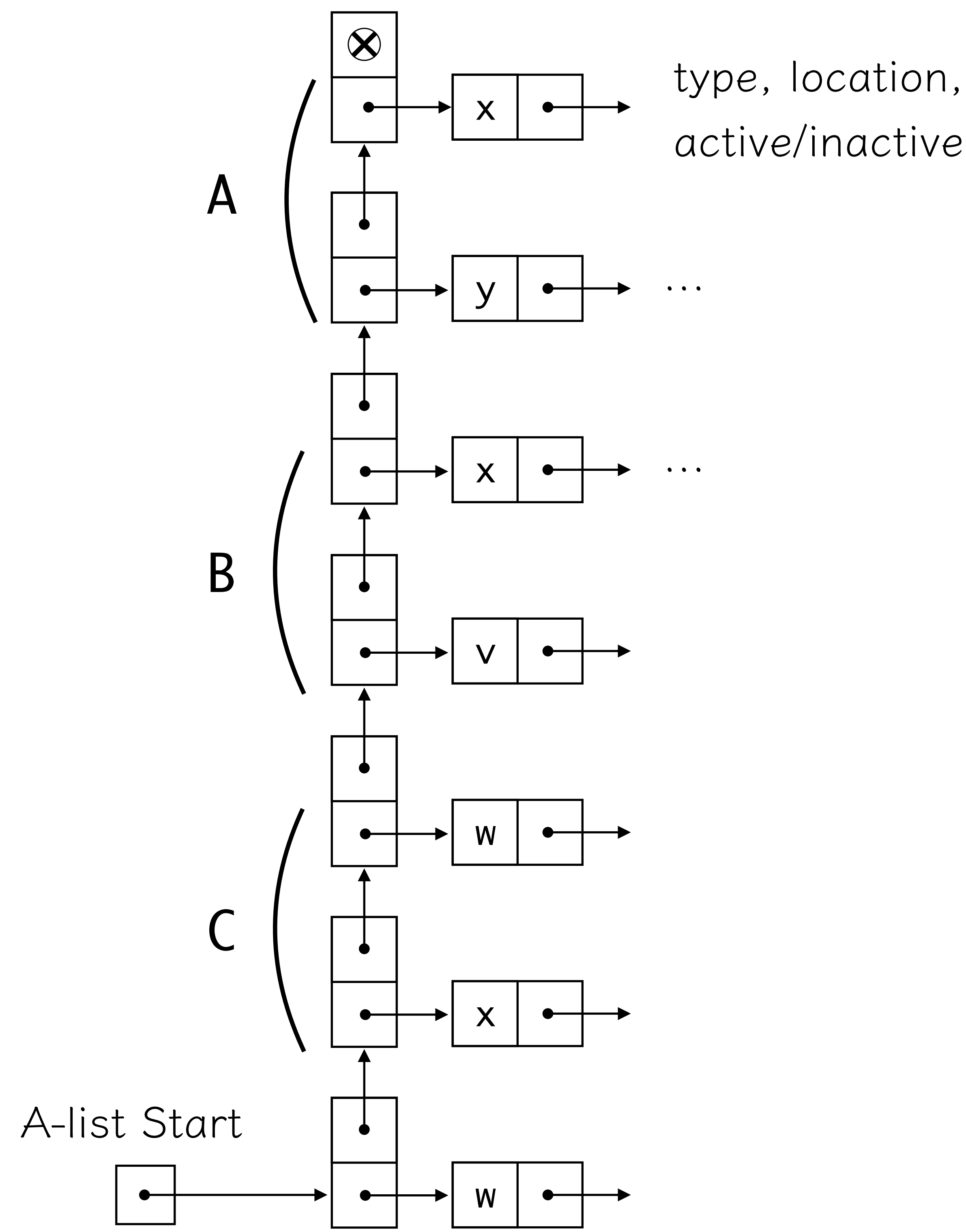
Dynamic Scope Rule Implementation

- In dynamic scope, non-local environments are considered in the order of their activations.
- Hence we need to go backward in the stack to find a proper binding.
- Let's consider calls A, B, C, D.
- Grey color means deactivated bindings.



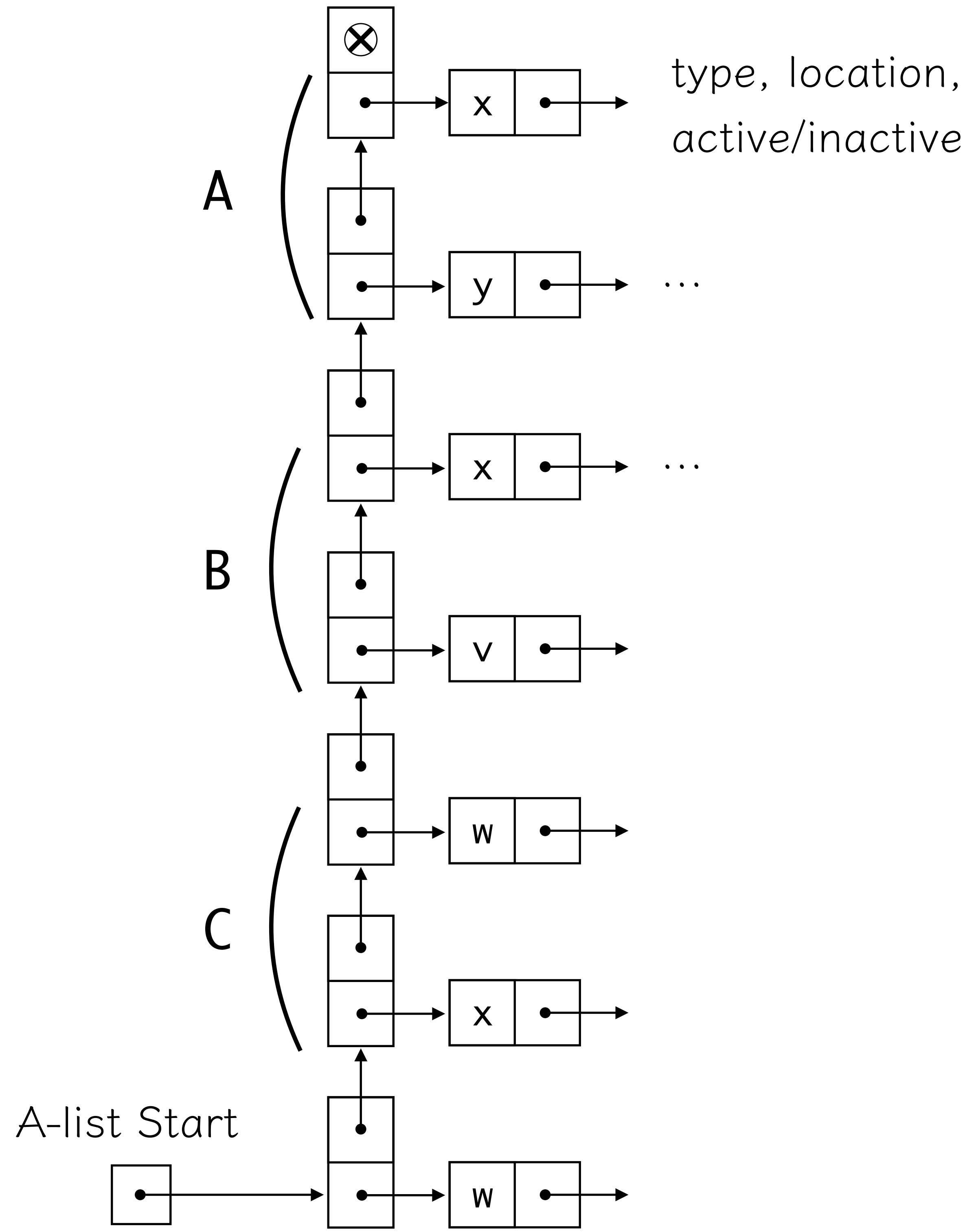
Association List

- Other than direct storage in activation records, bindings can be stored separately in ***an association list (A-list)***.
- A program enters a new environment, bindings are inserted to A-list, and removed when it exists the environment.



Association List

- Two disadvantages
  - *Names must be stored in structures at runtime.*
    - We cannot trace their locations at compile time.
  - *Runtime search of names is inefficient*.
    - We might need to check all the list.

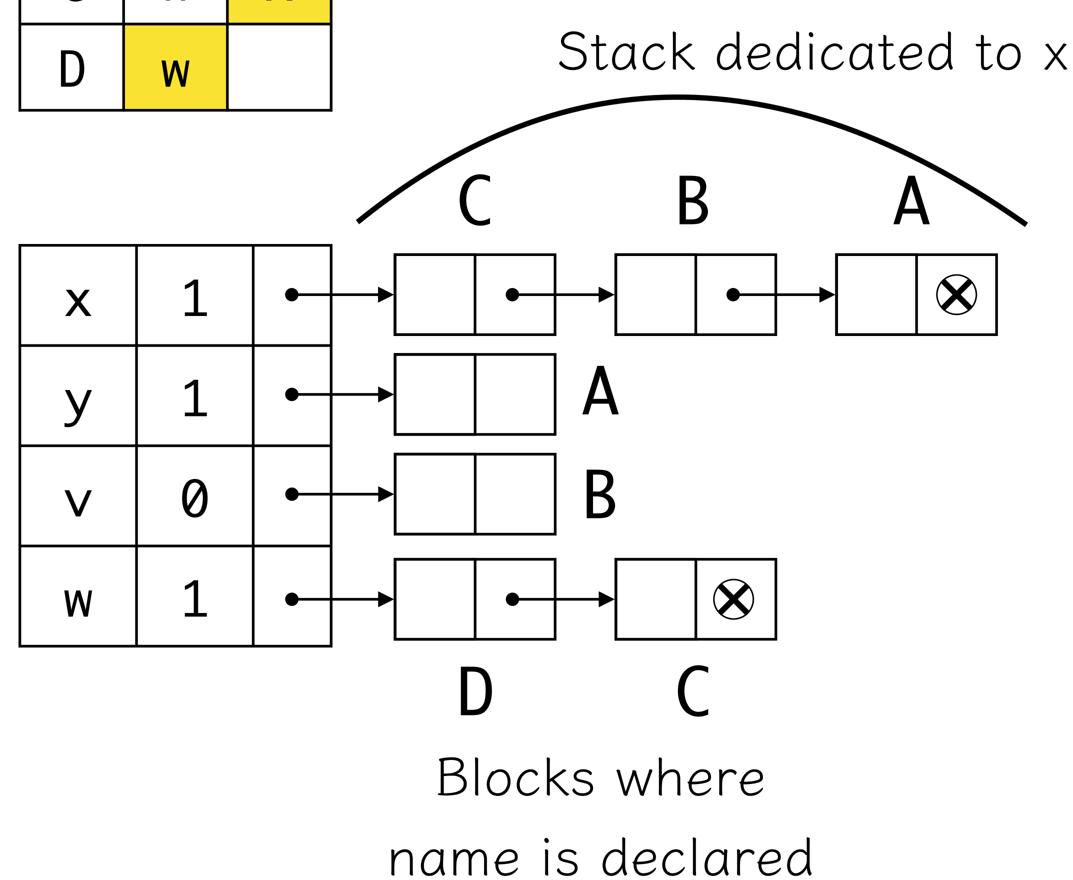


Central Referencing Table

- To address the disadvantages, we can employ ***Central Referencing Table (CRT)***.
- All the names used in a program are stored in CRT.
- For each name, there is a flag indicating whether it is active / inactive.
- Dedicated stack contains the valid binding for the name at the top, and deactivated bindings under it.

A	x	y
B	x	v
C	w	x
D	w	

Declared Names



Central Referencing Table

- We can also use a hidden stack to store all deactivated bindings.
- 3rd column contains the reference to the denotable object for the name.
- Deactivated bindings stored in the hidden stack, which will be restored when it becomes active again.

Declared Names

A	x	y
B	x	v
C	w	x
D	w	

reference to the object

A

x	1	a1
y	1	a2
v	0	
w	0	

AB

x	1	b1
y	1	a2
v	1	b2
w	0	

ABC

x	1	c1
y	1	a2
v	0	b2
w	1	c2

ABCD

x	1	c1
y	1	a2
v	0	b2
w	1	d1

hidden stack

x	a1

x	b1
x	a1

w	c2
x	b1
x	a1

Summary

- Scope Rule Implementation
  - Static Scope Implementation
    - Static Link
    - The Display
  - Dynamic Scope Implementation
    - A-list
    - CRT