Procedure Basics

P. J. Denning

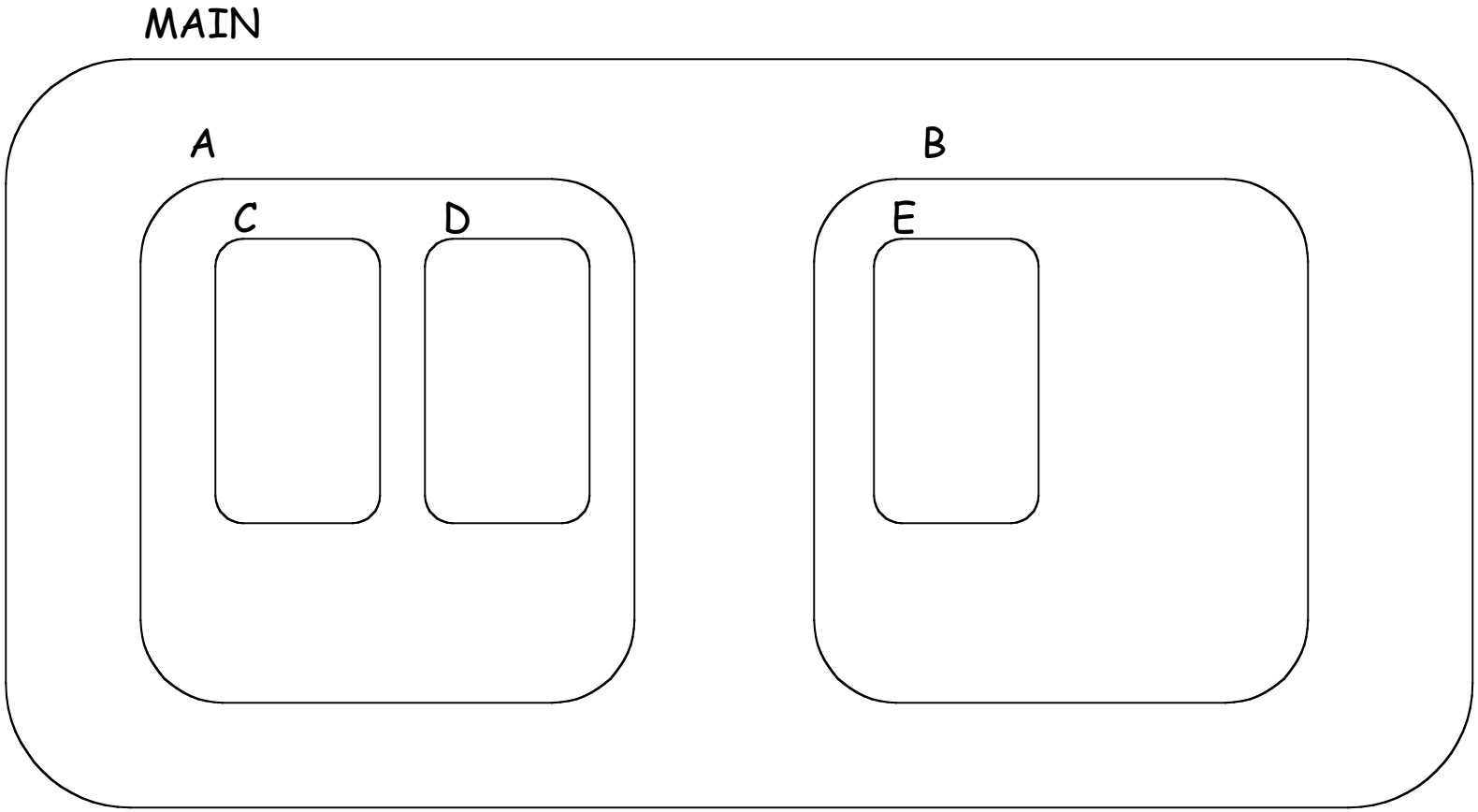
For CS471/571

Closed Sub-Programs

- Idea dates back to 1949 (Atlas @ U Manchester)
- Motivations for sub-programs:
  - Modularity
  - Information hiding
  - Reuse
- Sub-program invocation is a function call that appears as a single step to the caller.
- Function call takes parameters (inputs) and produces a result value (output)

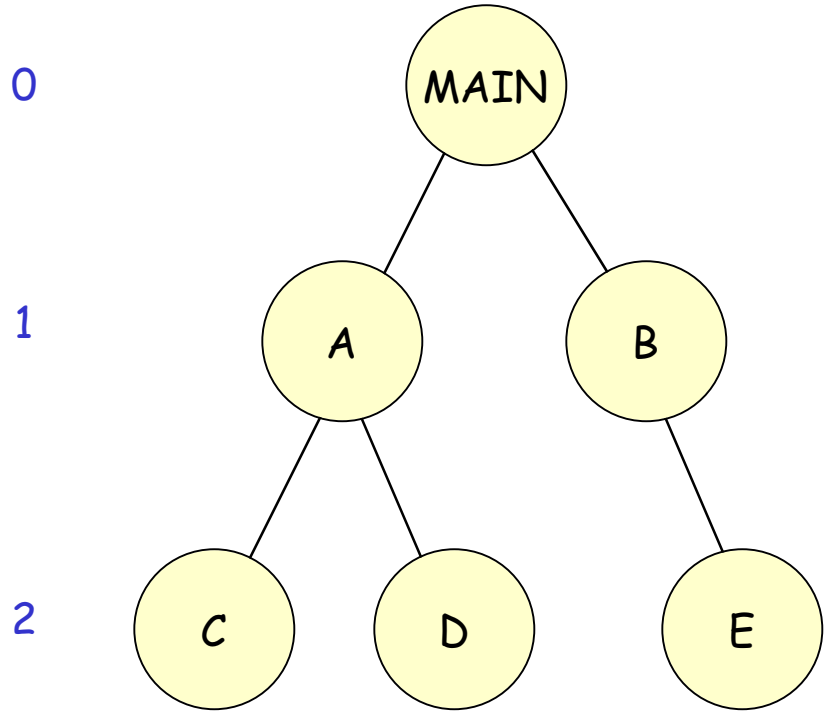
Contour Model

- Contour: an executable sub-program unit with instructions, parameters, local variables, and private working store.
- Name comes from a diagramming method in which sub-programs depicted as contours on a topological map.
- Contour model originated around 1970 (Johnston)



MAIN program declares two sub-programs A and B.
A declares two more sub-programs C and D.
B declares one more sub-program E.

LEVELS



Contour diagram corresponds to tree of enclosures.

ENVIRONMENT of a contour = path from the node back to the root

e.g.,

E(C) = (C, A, MAIN)
E(A) = (A, MAIN)

Block level = distance from root

- Contours are a model for block structure in some programming languages
- Contours are a model for type inheritance in object-oriented languages

Block Levels

- Block level of a contour or a variable is the level at which the name is declared.
  - Level of MAIN: 0
  - Level of A and B: 1
  - Level of C, D, and E: 2
  - Level of variable x of MAIN: 0
  - Level of variable y of A: 1

Execution Sequences

- (C, …) denotes period of invocation of contour C
- (C is the moment of call,) the return
- Two contour calls are either independent or one embedded in the other
  - (C1, …) … (C2, …)
  - (C1, … (C2, …) …)

Allowable References

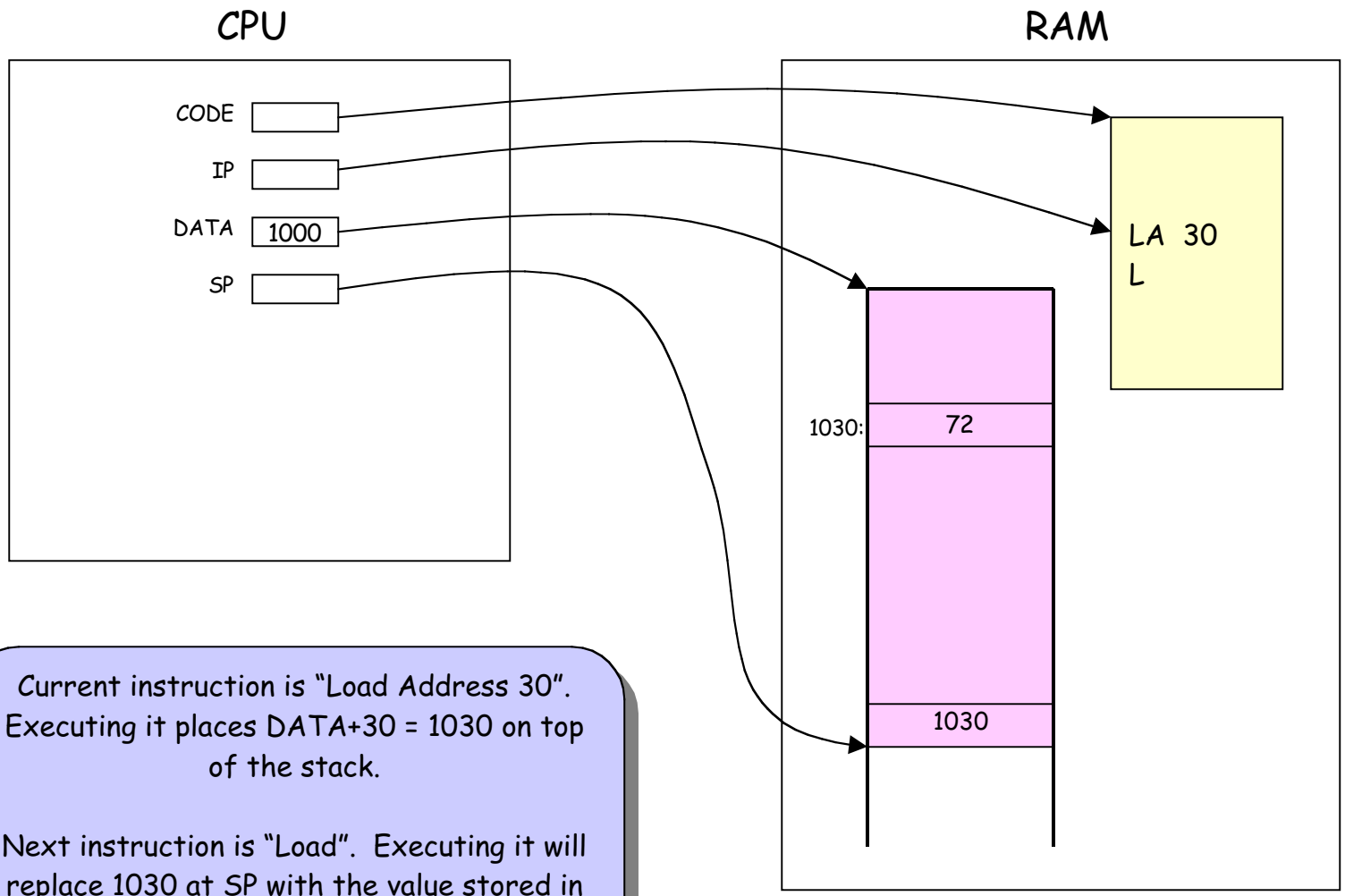
- Contour can refer to any name defined in any contour of its environment. In case two contours duplicate a name, only the closest is visible.
  - Example: MAIN contains variable x; all embedded contours (A,B,C,D,E) can refer to x.
  - Example: A contains variable y; contours C and D can refer to y, but not B, E, or MAIN.
  - Example: A contains variable named x also; C and D can refer to A's x but cannot see MAIN's x.
- Contour cannot see any name at a level deeper than its own.

Allowable Calls

- A contour's name is visible within its defining contour; that name is also visible from all contours embedded in the defining contour. Examples:
  - C can call B or D
  - B can call A or E
  - B cannot call C or D
  - MAIN can call A or B; but not C, D, or E
- Any contour can call itself (recursion)

Code Implementation

- Contour (procedure) code stored in a separate, executable code segment.
- Base of that segment is in a standard CPU register.
- Next instruction within that segment is addressed by IP (instruction pointer) register in CPU.
- All data used by the code come from a data segment, usually stored as a stack. (Sometimes called the "call stack" because space is allocated on procedure calls and released on procedure returns.)

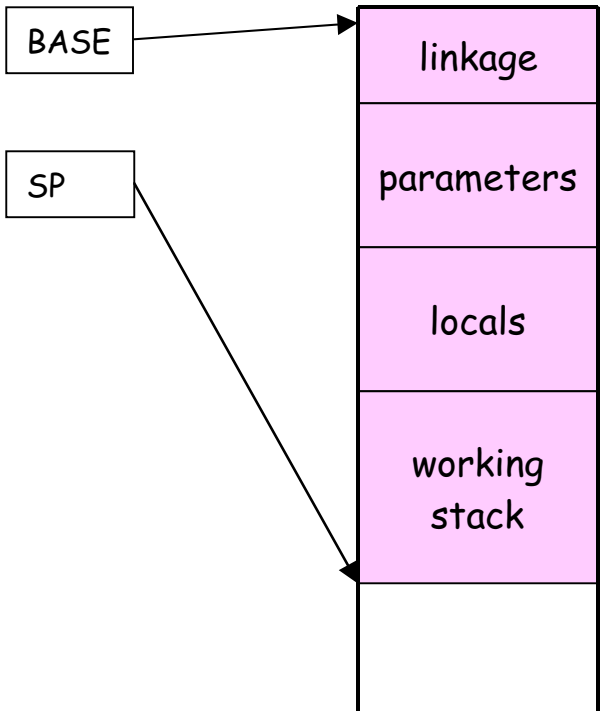


Current instruction is "Load Address 30". Executing it places DATA+30 = 1030 on top of the stack.

Next instruction is "Load". Executing it will replace 1030 at SP with the value stored in location 1030, i.e., with 72.

Data Implementation

- Invocation of contour is stored in an activation record, or frame, created on call, deleted on return.
- Frame contains space for return linkage information, parameters, local variables, and working store.
- Frames linked in LIFO order; therefore, frames can be stored on a stack.
- New (called) frame is constructed on top of the working stack of the caller. Its base is the caller's SP just before the call.



FRAME STRUCTURE

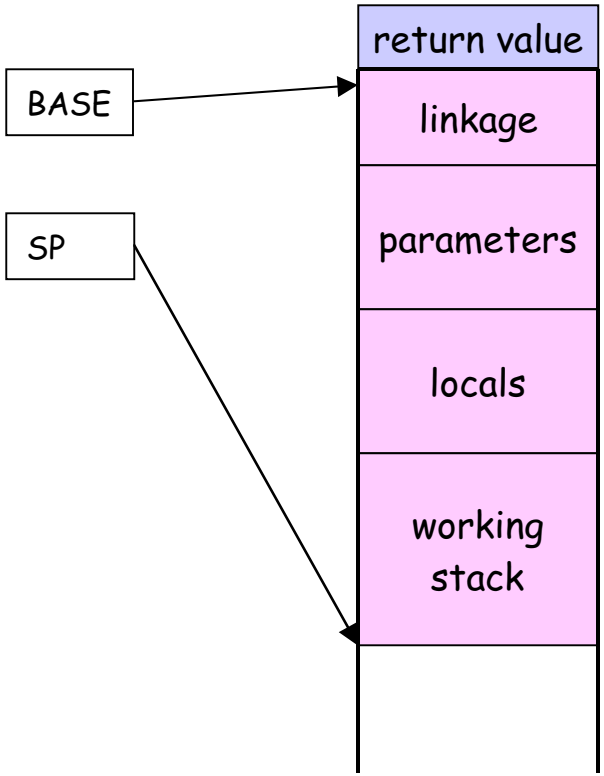
Linkage area contains return state to restore caller's environment.

Parameter area contains parameters of the call.

Locals area contains local variables of the procedure.

Working stack contains the temporary store, managed as stack. E.g., to compute X+Y,

```
LA  x      BASE+x on stack
L          value of X on the stack
LA  y      BASE+y on stack
L          value of Y on the stack
A          sum of X+Y on the stack
```



RETURNING A VALUE

Frame element BASE-1 is the target for the return value. That element will be the top of stack (SP) after the return.

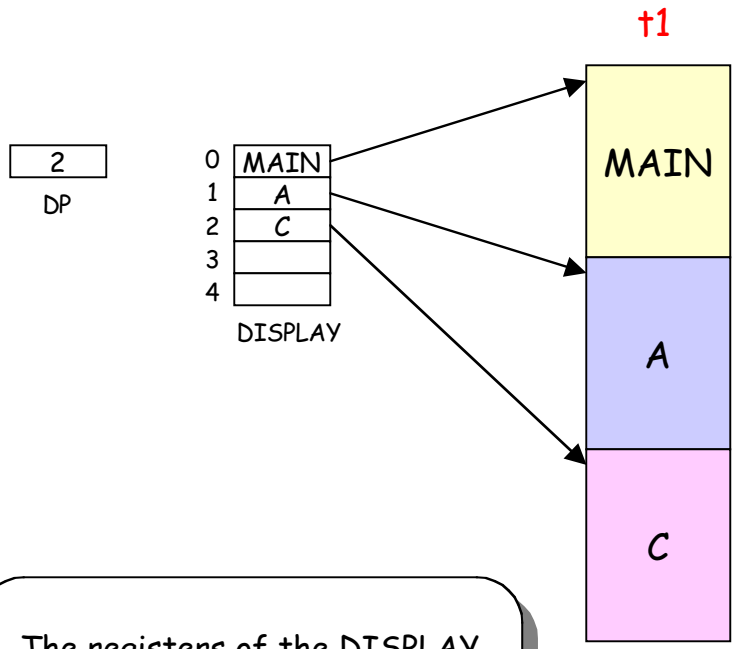
Before making the call, the caller must reserve an element on top of its working stack for the return value.

Contour procedure copies the return value from the top of its working stack to the return location with

```
LA   -1
"compute return value"
ST
```

(MAIN, … (A, … (C, … (B, …) …) …) …)

t1



The registers of the DISPLAY and the display pointer DP replace the single register DATA shown previously.

DISPLAY contains the current environment (at time t1).

DP contains the current block level in static tree.

DISPLAY[DP] is base of current frame.

Extend definition of LA to include the block level k of the contour in which an address x is to be interpreted:

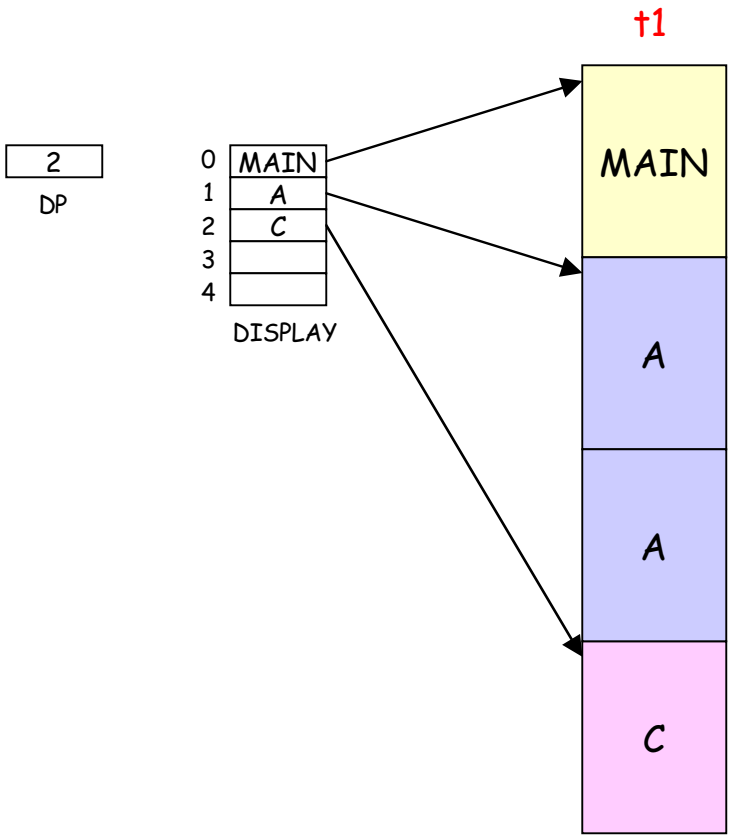
LA k,x

Puts DISPLAY[k]+x on stack.

If contour at level K contains this instruction, K≥k because no contour can see variables deeper in the tree.

(MAIN, … (A, … (A, … (C, … (B, …) …) …) …) …)

t1



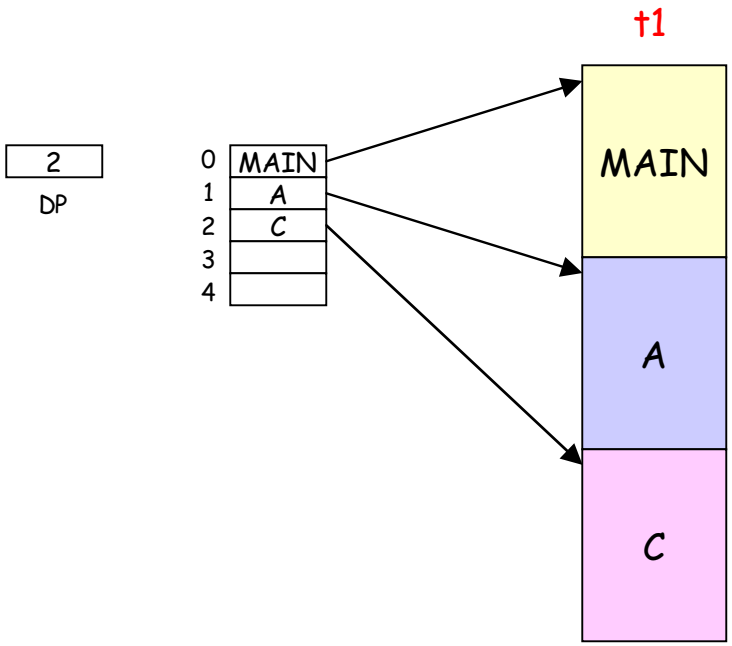
More frames may be on the stack than are indicated by the display -- because of a recursion or because of a call to block higher in the tree.

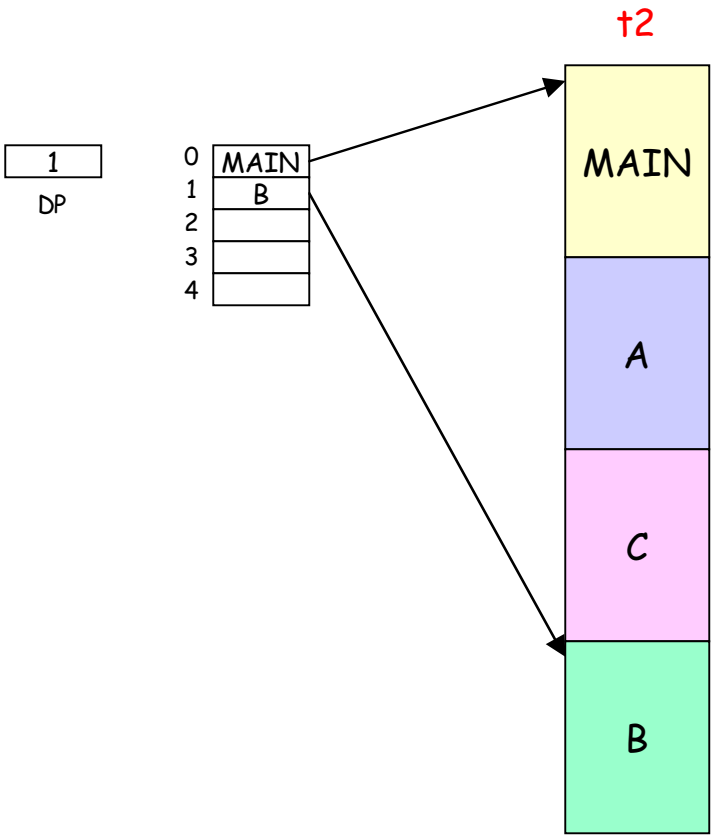
Diagram illustrates an extra frame inserted by a recursive call on A.

The DISPLAY permits referencing statically-enclosing contours no matter what the dynamics of contour calls have been.

(MAIN, … (A, … (C, … (B, …) …) …) …)

t1 t2





- Data to be saved in the linkage section of the called contour:
  - Old DP
  - Old DISPLAY[k], where k = level of called contour
  - Old SP
  - Old (IP,S,M), where S = supervisor state, M = interrupt mask
- The CALL instruction saves these data, sets DP=k and DISPLAY[k]=x, and sets (IP,S,M) = (ENTRY(x), S(x), M(x))
- The instruction RETURN restores the old values (and thereby discards the called frame from the stack).

In the example …

- The call at time t2 changes the environment from E(C) = (C, A, MAIN) to E(B) = (B, MAIN)
- The calling contour (C) is at block level 2 and the called contour (B) at block level 1.
- Environment is in DISPLAY, current block level in the display pointer (DP).
- The call instruction specifies k, the block level, and x, the code entry point, of the called contour.
- The call just prior to time t2 is for (k,x) = (1,B).
- All the call needs to do is change DP to 1 and DISPLAY[1] to B. The other display registers can be left in place because B cannot reference deeper and will not use them.

Procedure Entry

- The procedure code of Contour C is embedded in the code segment with an entry point ENTRY(C).
- If C is an OS procedure, it may operate in supervisor mode and with some interrupts masked.
- Define C's procedure handle as (ENTRY(C), S(C), M(C)), where S(C) is the supervisor/user mode setting and M(C) is the interrupt mask setting.
- Put procedure handle on stack, invoke it with CALL instruction.

CALL Sequence

- Objective: call procedure whose handle is h and is a block level k
- Reserve a stack element for the return value.
- MARK instruction pushes the SP value on a temporary register stack PC of pending calls; marks the base of the new frame.
- Compute parameters and push on the stack.
- Push initial values of locals on the stack.
- Push k and h on top of the stack.
- CALL instruction:
  - Saves environment state in linkage area (marked by PC)
  - Invokes procedure designated by procedure handle h
  - Pops and discards the top two elements of stack
  - Pops and discards the top element of PC

EXAMPLE

Let uppercase denote a variable and lowercase its location in its frame. Let X, Y be local variables of a procedure at block level j. Function F is defined at level k < j. Function invocation Y = F(X) is compiled as shown, constructing the new frame on top of the stack.

```
LA  j,y        address of Y on stack
R   1          reserve 1 blank element for return value
MARK           mark base of new frame
R   4          reserve 4 blank elements for linkage
LA  j,x        address of parameter X on stack
L              value of parameter X on stack
"locals"       set up local variables on stack
L  "k"         block level of function F on stack
LA  j,f        address of procedure handle of F on stack
L              value  procedure handle of F on stack
CALL           invoke procedure
ST             store result in Y (address on top of stack)
```

EXAMPLE

Same basic conditions as previous example. Function invocation Y = F(X+Z) is compiled as:

```
LA  j,y        address of Y on stack
R   1          reserve 1 blank element for return value
MARK           mark base of new frame
R   4          reserve 4 blank elements for linkage
LA  j,x        address of parameter X on stack
L              value of parameter X on stack
LA  j,z        address of parameter Z on stack
L              value of parameter Z on stack
A              compute sum
"locals"       set up local variables on stack
L  "k"         block level of function F on stack
LA  j,f        address of procedure handle of F on stack
L              value  procedure handle of F on stack
CALL           invoke procedure
ST             store result in Y (address on top of stack)
```

EXAMPLE

Same basic conditions as previous example. Function G is defined at level m. Invocation Y = F(G(X)) is compiled as:

```
LA  j,y        address of Y on stack
R   1          reserve 1 blank element for return value
MARK           mark base of new frame (F)
R   4          reserve 4 blank elements for linkage (F)
R   1          reserve 1 blank element for value of G
MARK           mark base of new frame (G)
R   4          reserve 4 blank elements for linkage
LA  j,x        address of parameter X on stack
L              value of parameter X on stack
"locals"       set up local variables for G on stack
L  "m"         block level of G on stack
LA  j,g        address of G handle on stack
L              value of G handle on stack
CALL           call G
"locals"       set up local variables for F on stack
L  "k"         block level of function F on stack
LA  j,f        address of procedure handle of F on stack
L              value  procedure handle of F on stack
CALL           invoke procedure
ST             store result in Y (address on top of stack)
```