Run-Time Environements

Martin Sulzmann

Overview

Things to consider

- Static program code versus dynamic procedure activation.
  - First-order functions (C).
  - Nested procedure declarations (C++/Java).
  - Higher-order functions (OCaml/Haskell/Go).
- Allocation and deallocation of data objects.
  - Stack versus heap.

Static versus Dynamic

Classification

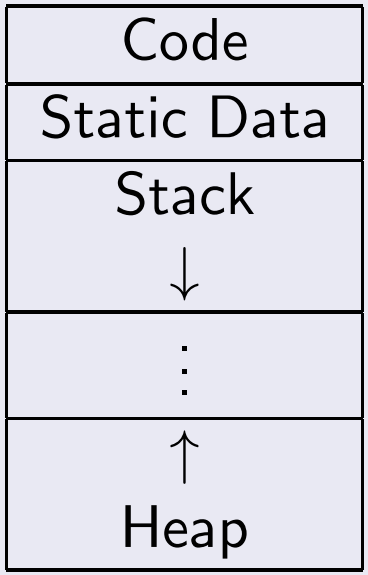
- Static = at compile time.
- Dynamic = at run-time.
- Stack maintains data local to (procedure call).
  - Layout statically known.
  - Managed automatically by generated code.
- Heap maintains (dynamic) data that survives between calls.
  - Layout statically not known.
  - Managed dynamically by garbage collector.

Storage Organization

Classification

- Code area: instructions.
- Static area: (global) constants.
- Heap: objects (records, arrays, ...).
- Run-time stack: (procedure) activation records/stack frames.

Memory Layout



Procedures

Example

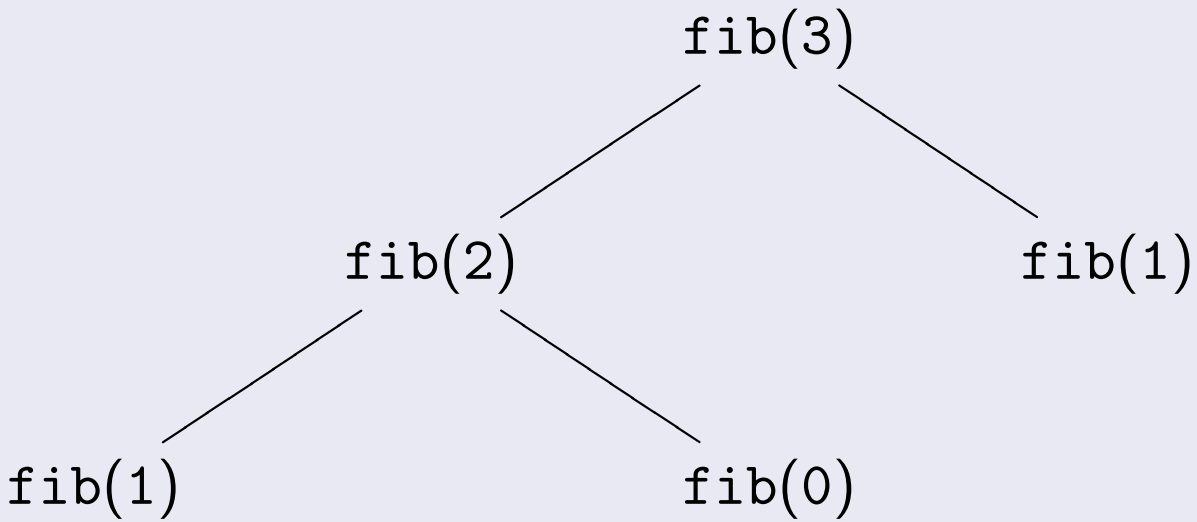
```
int fib(int n){
 int f1, f2;
 if (n<=1) return 1;
 f1=fib(n-1);
 f2=fib(n-2);
 return f1+f2;
}
```

We assume a procedural language with

- sequential control flow, and
- return to point after the call.

Activation Tree

Example

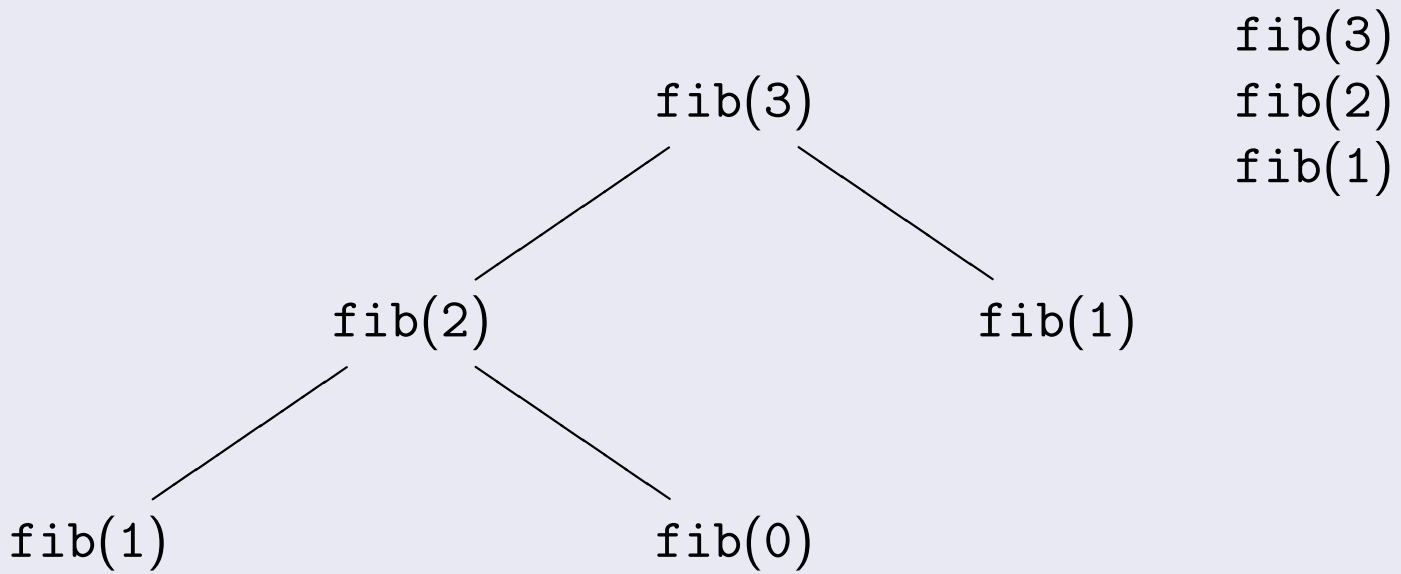


Node: procedure activation
Child: activation from inside parent
Order: Left child finishes before right child begins
Life-time: From call to exit

Control Stack

Stack for procedure activation

Depth-first left-to-right visit of activation tree.

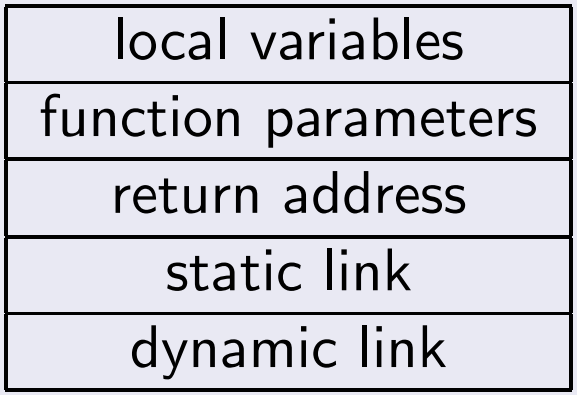


Where do we store (local) data, scope of declarations, binding of names?

Activation Records

Layout

Item on (control) stack. Contains all information for an activation of a procedure.



dynamic link	=	caller's activation record (where to return after procedure exit)
static link	=	non-local data on stack (how to access variables from some outer scope)

Variable Access

Things to consider

- Variables found in some activation record (ignore heap allocated data).
- Local variable easy (within current activation record).
- Non-local variable? Must find proper activation record.

Method

- Location of a variable represented by (level, relpos).
- level = level of the lexical scope.
- relpos = relative position in the activation record.
- Need protocol to properly access variables based on the information provided.

Example

Mini-Go

```
{
  x := 1;
  y := true
}
```

Activation record

```
----------------------
| y                  |  1 <-- relative position
----------------------
| x                  |  0
----------------------
| NULL               |   <-- return address
----------------------
| NULL               |   <-- static link (SL)
----------------------
| NULL               |   <-- dynamic link (DL)
----------------------
```

Nested Procedure Declarations

Example

```
main () {
  int i;
  int c() { return i };
  int a(int i) {
    i = 1;
    int b( ) {
       int d( ) { return c( ); };
       d();
    };
    b();
  };
  i = 2;
  print a(i);
}
```

Nested Procedure Declarations (2)

Level information

Procedure	level
main	1
c	2
a	2
b	3
d	4

Things to consider

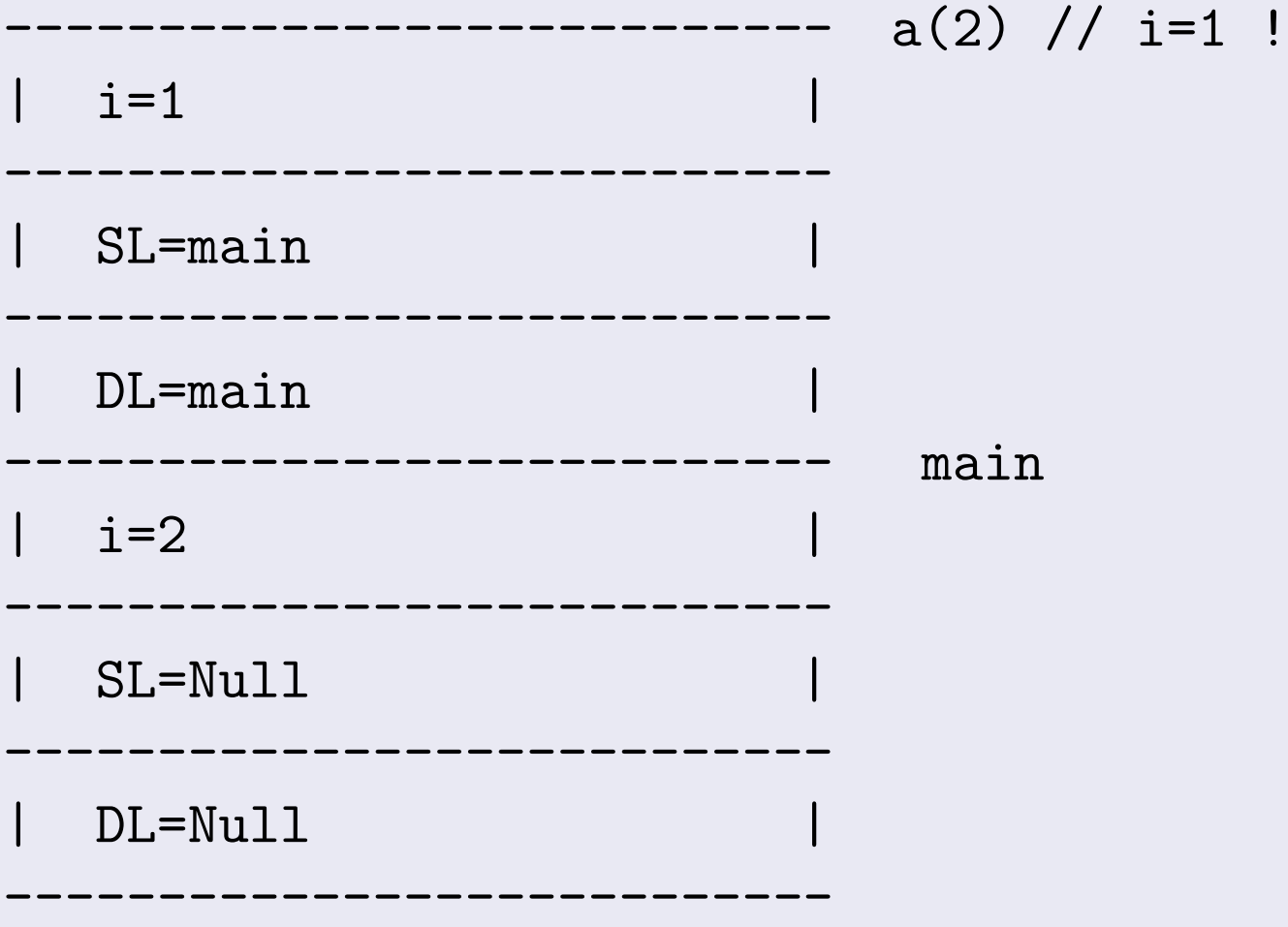
- Call sequence: a(2) $\rightarrow$ b() $\rightarrow$ d() $\rightarrow$ c()
- Procedure c has access to variable `i`. Which `i`? Where do we find `i`?

Initial

Stack of activation records

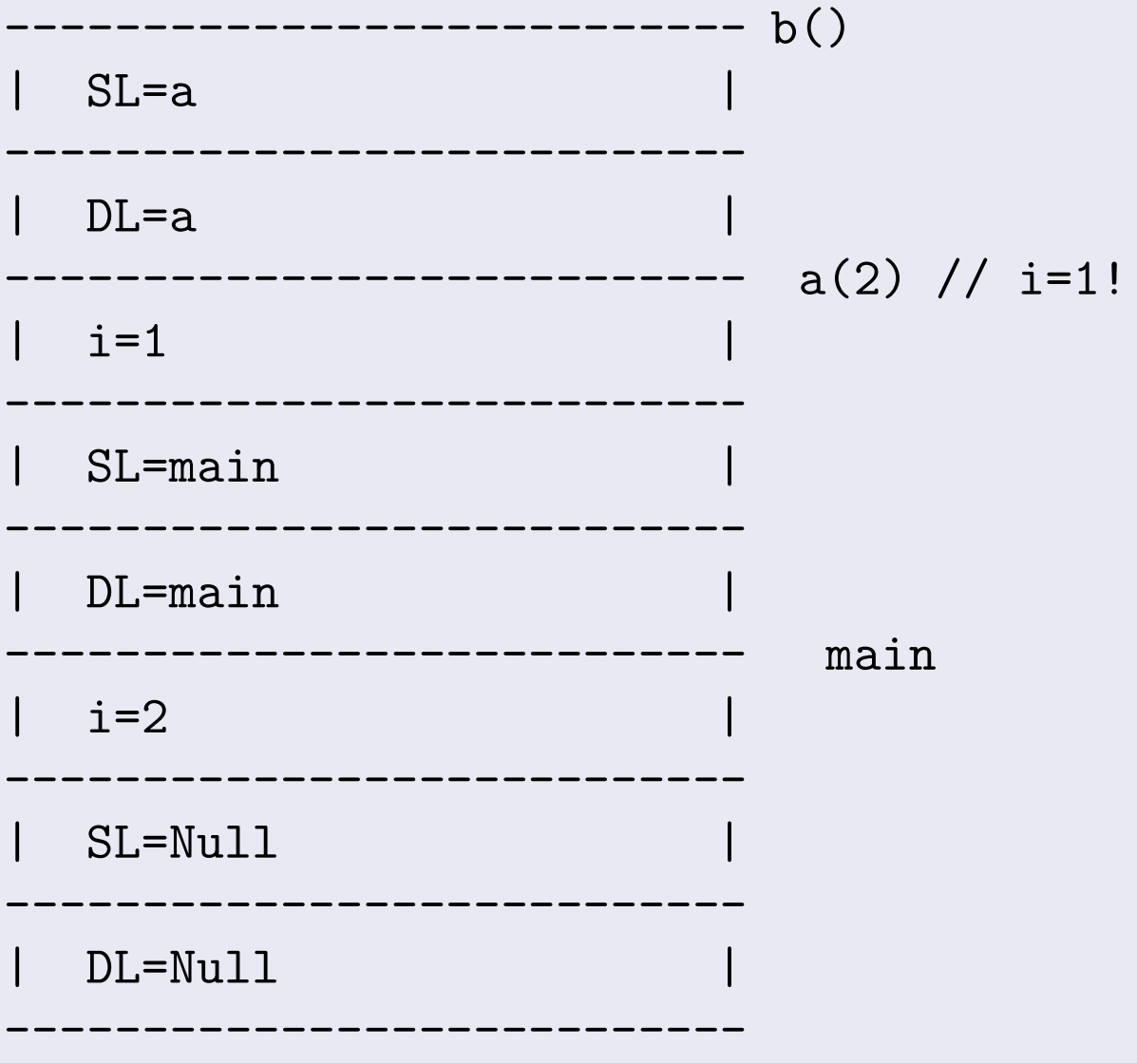
Call a(2)

Stack of activation records



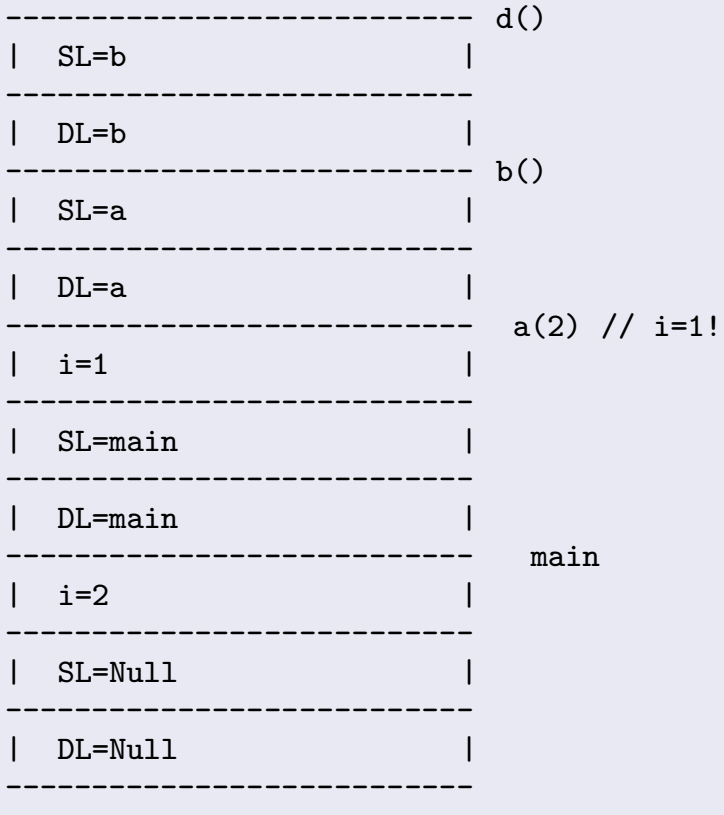
Call b()

Stack of activation records



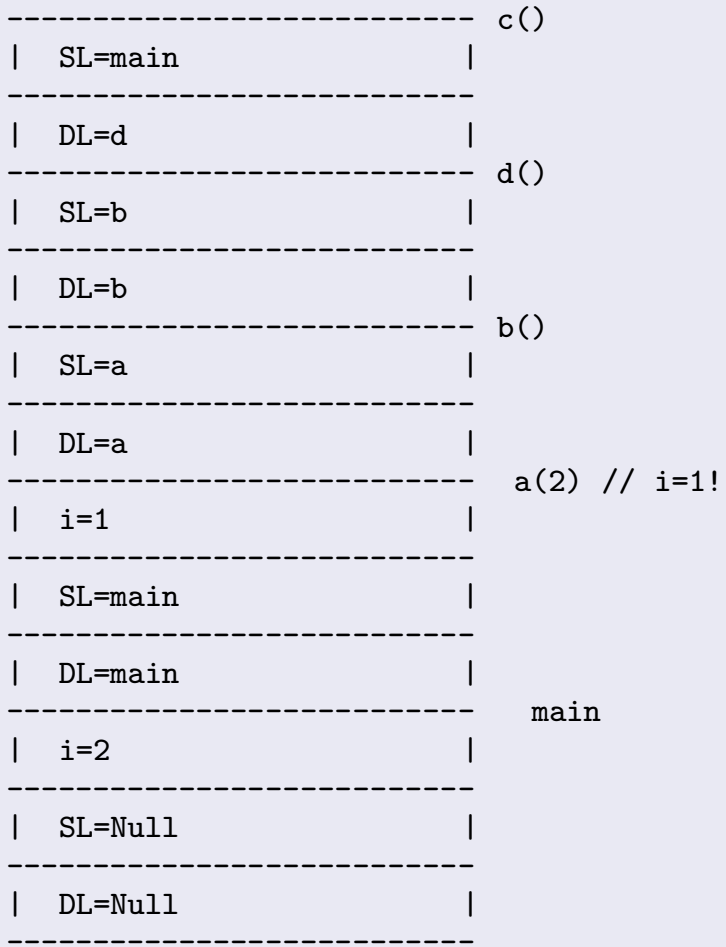
Call d()

Stack of activation records



Call c()

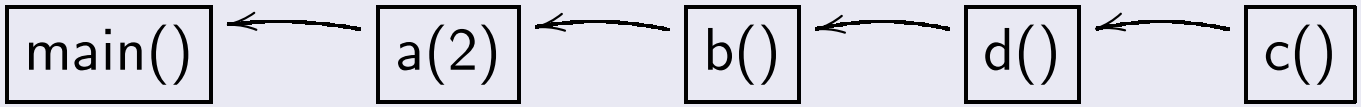
Stack of activation records



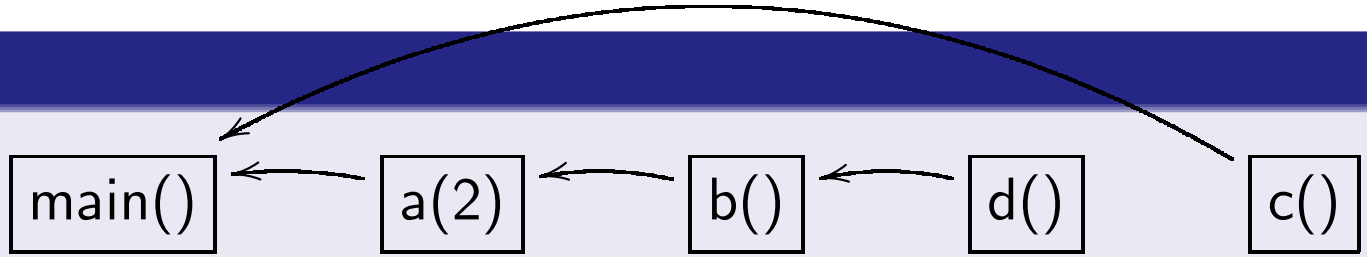
- c = level 2 and d = level 4.
- Hence k = 2.
- Follow SL twice.
- Hence, SL of c equals main.
- Access i: main (level = 1) and c (level = 2).
- Follow SL once. "Look-up" i.

Call sequence: a(2) → b() → d() → c()

Dynamic link



Static link



Managing Activation Records

Return sequence

- "Pop" current activation record.
- Follow DL to previous one.
- Jump to return address and continue.

Calling sequence

- "Push" new activation record.
- Set DL to previous one.
- If level of current block $>$ level of previous block then set SL $=$ SL of the kth previous block (=activation) where k= level of current block - level of previous block (see call c()).
- Otherwise, set SL $=$ SL of previous block.

Variable Access

Variable access protocol

Assume we are within a block at level i and access variable x whose location is represented by (k,relpos).

- If i=k (x local) then relpos refers to relative position of x in current activation record.
- If i>k (x global) then we need to follow the SL chain (i-k) times to get to x's activation record.
- Otherwise (i<k), can never happen.

Mini-Go

Observation

- No nested procedure declarations.
- But nested scope.
- Variables do not need to be declared at the beginning of a block.
- Introduce distinct variables (renaming!).
- Variables in main can then be managed by a single activation record.