NEW YORK UNIVERSITY

CSCI-GA.2130-001

Compiler Construction

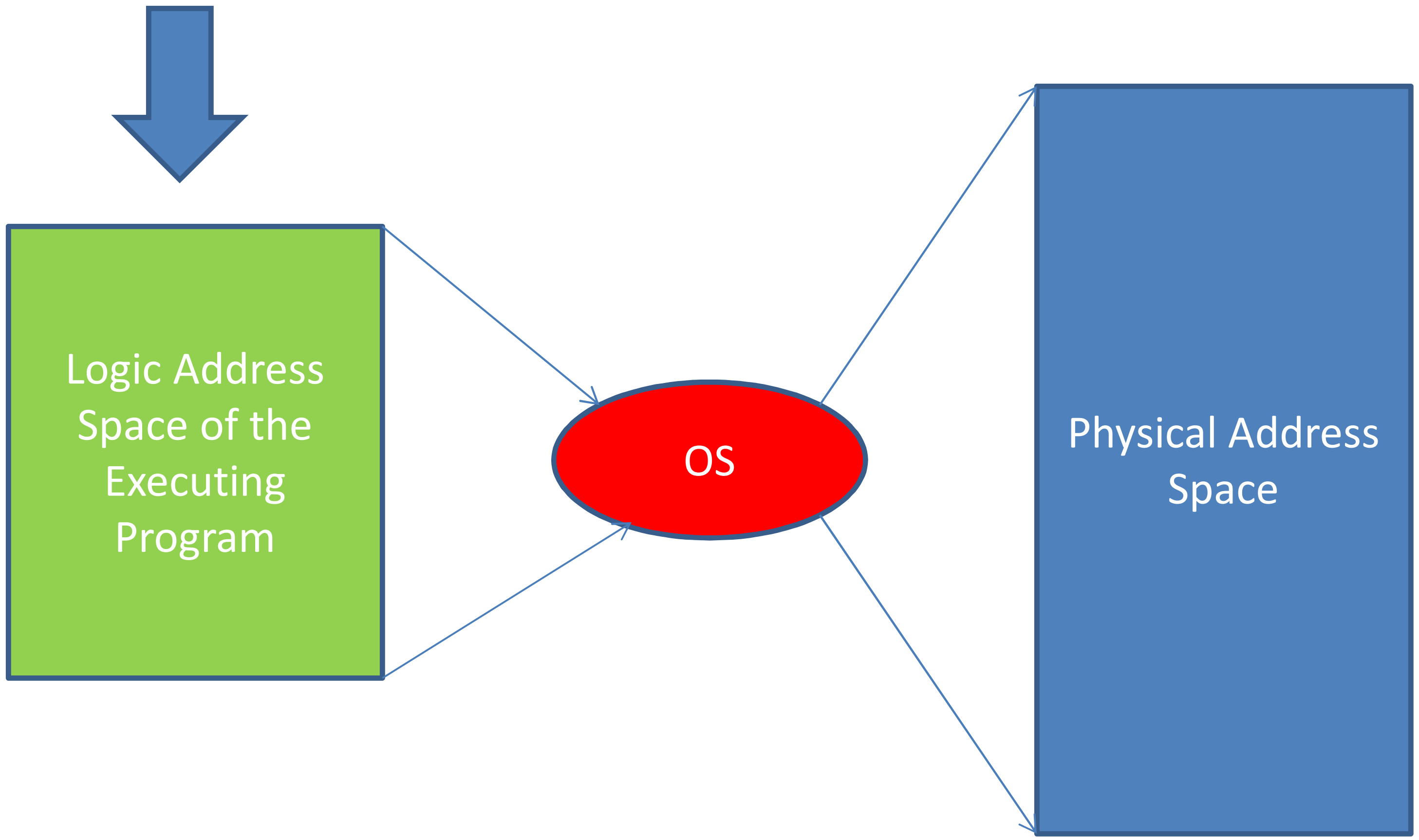
Lecture 11: Run-Time Environment

Hubertus Franke
frankeh@cs.nyu.edu

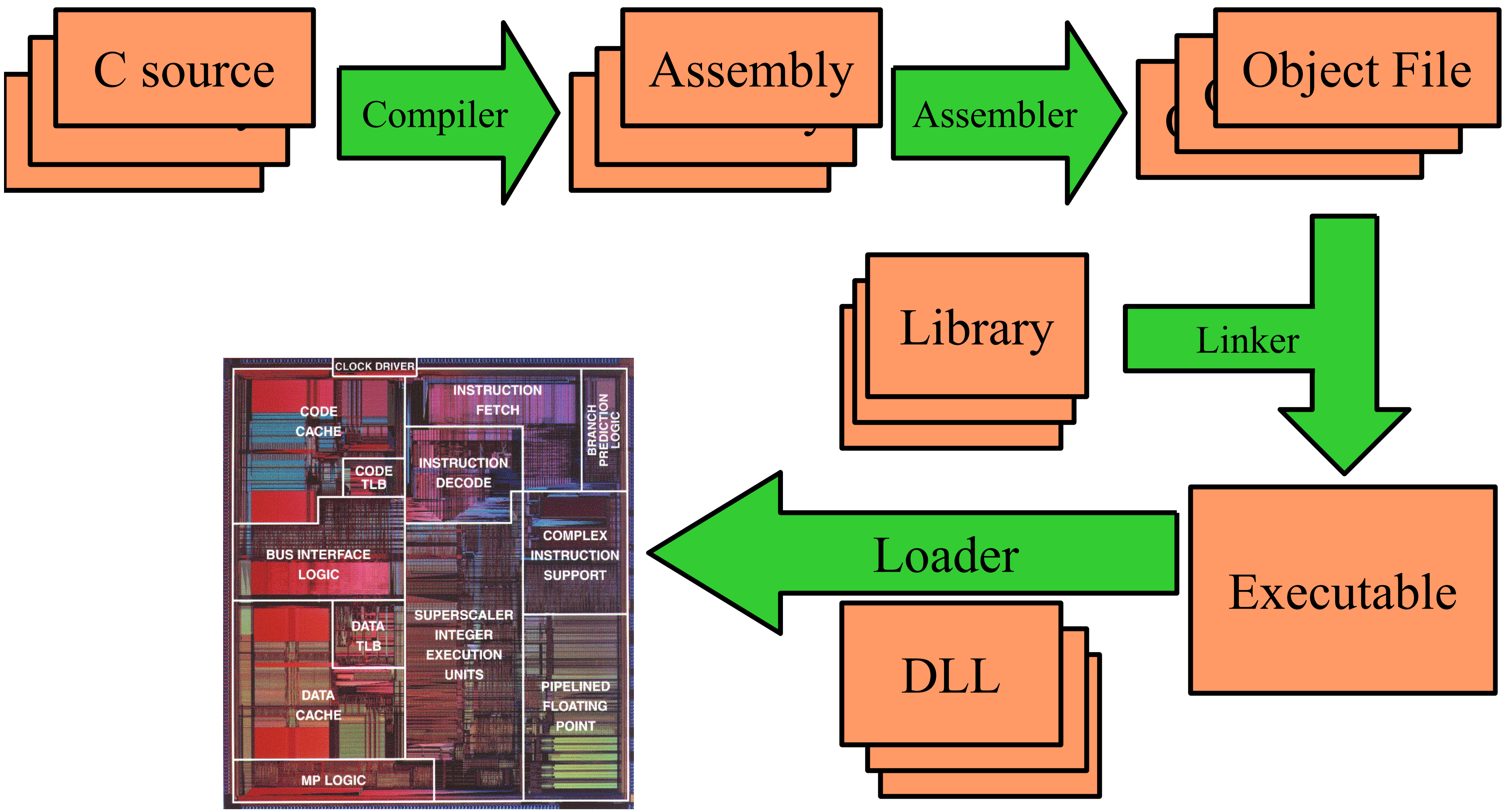
What Are We Talking About Here?

- How do your code and data look like during execution?
- Interaction among compiler, OS, and target machine
- The main two themes:
  - Allocation of storage locations
  - Access to variables and data

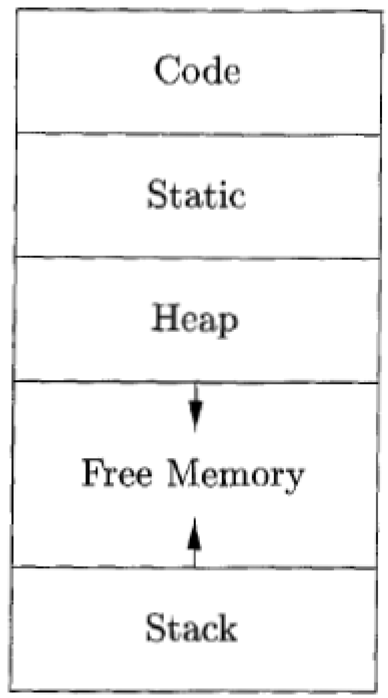
Compiler-writer Perspective



Source Code to Execution



Typical Memory Subdivision



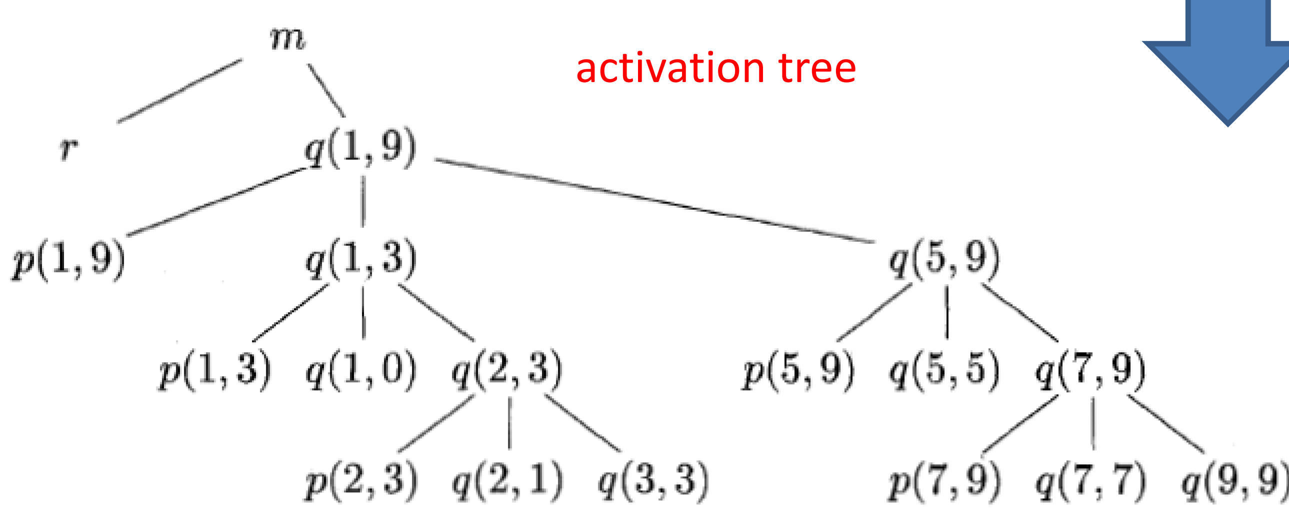
Stack Allocation

- For managing procedure calls
- Stack grows with each call and shrinks with each procedure return/terminate
- Each procedure call *pushes* an activation record into the stack

```
int a[11];
void readArray() { /* Reads 9 integers into a[1], ..., a[9]. */
    int i;
    ...
}
int partition(int m, int n) {
    /* Picks a separator value v, and partitions a[m..n] so that
       a[m..p-1] are less than v, a[p] = v, and a[p+1..n] are
       equal to or greater than v. Returns p. */
    ...
}
void quicksort(int m, int n) {
    int i;
    if (n > m) {
        i = partition(m, n);
        quicksort(m, i-1);
        quicksort(i+1, n);
    }
}
main() {
    readArray();
    a[0] = -9999;
    a[10] = 9999;
    quicksort
}
```

```
enter main()
    enter readArray()
    leave readArray()
    enter quicksort(1,9)
        enter partition(1,9)
        leave partition(1,9)
        enter quicksort(1,3)
            ...
        leave quicksort(1,3)
        enter quicksort(5,9)
            ...
        leave quicksort(5,9)
    leave quicksort(1,9)
leave main()
```

activation tree



Activation Tree

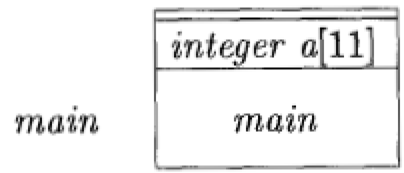
- Models procedure activations
- The *main* is the root
- Children of the same parent are executed in sequence from left to right
- Sequence of procedure calls -> preorder traversal of activation tree
- Sequence of procedure returns -> postorder traversal of activation tree

Activation Records

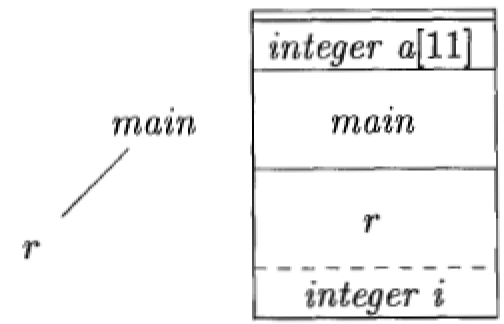
- What is pushed into the stack for each procedure activation
- Contents vary with the language being implemented

General Activation Record

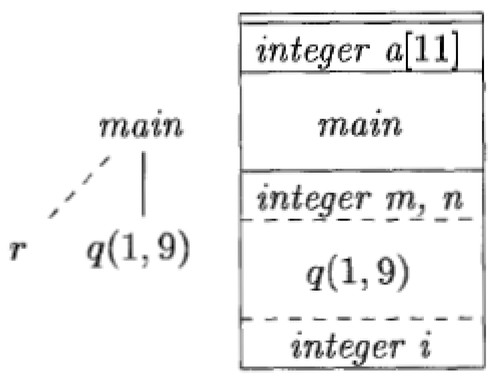
Actual parameters
Returned values
Control link
Access link
Saved machine status
Local data
Temporaries



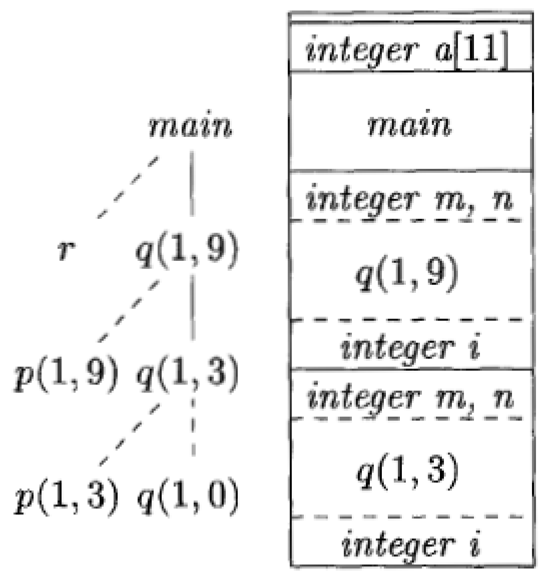
(a) Frame for *main*



(b) *r* is activated



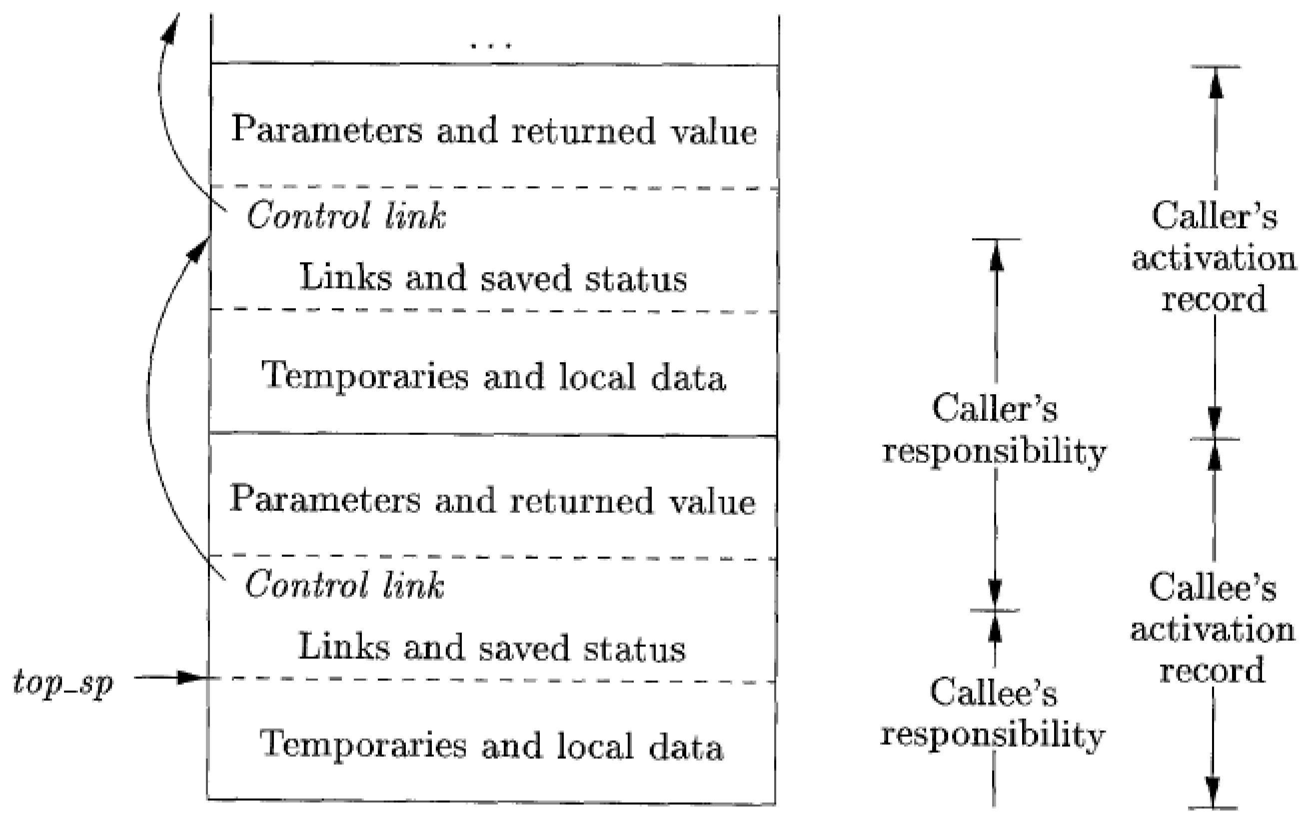
(c) *r* has been popped and $q(1,9)$ pushed



(d) Control returns to $q(1,3)$

Code Generation

- Calling sequence
  - Code that allocates activation record
  - Code for entering information in it
- Return sequence
  - Code to restore the state of the machine

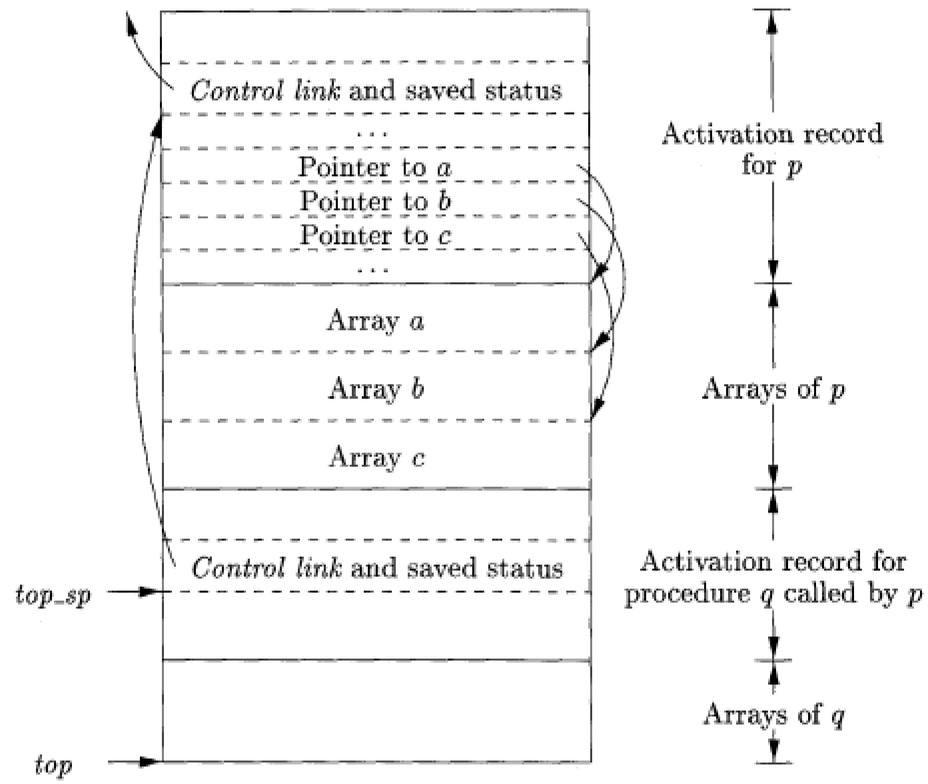

...
Parameters and returned value
Control link
Links and saved status
Temporaries and local data
Parameters and returned value
Control link
Links and saved status
top_sp
Temporaries and local data
Caller's responsibility
Callee's responsibility
Caller's activation record
Callee's activation record

Variable Length Data

- What if size of local array can not be determined at compile time?

- Allocate <ptr>
- Allocate array[] at runtime (grow stack at runtime)
- ptr = array

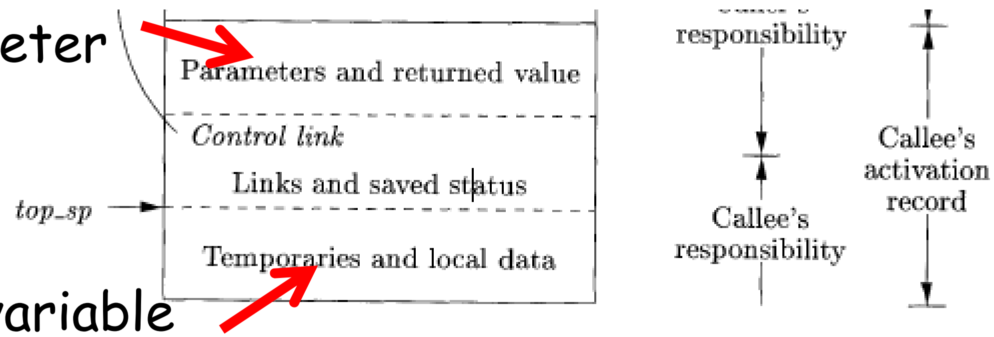
- alloca() is an example in C



Data Access (non-nested)

- Simple distinction between local and global
- Access method for <u>local</u> variables:
- Stack relative: variable is synonymous with relative location of the activation record (+/- offset to stack)

  - ldw r4, sp(-16)   % parameter
  - ldw r3, sp(8)   % local variable



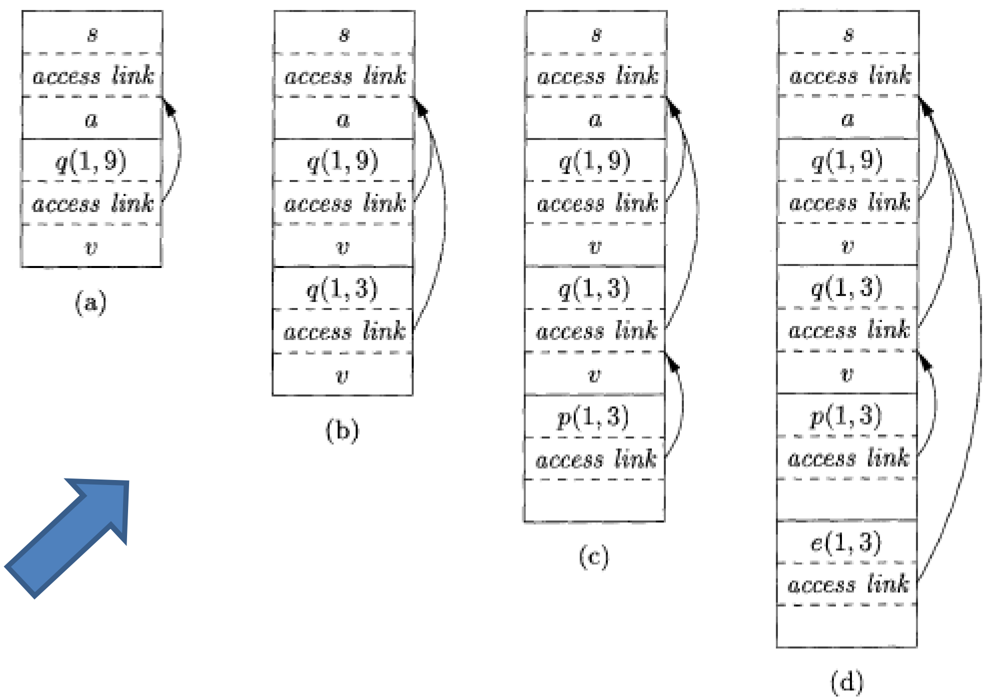
Data Access (non-nested)

- Access method for global variables:
- Compiler determines relative address of variable wrt to module/file into .data/.bss segment
- Linker merges all segments into a single segment and changes the offsets -> leads to global address

- Special handling of dynamically loaded modules

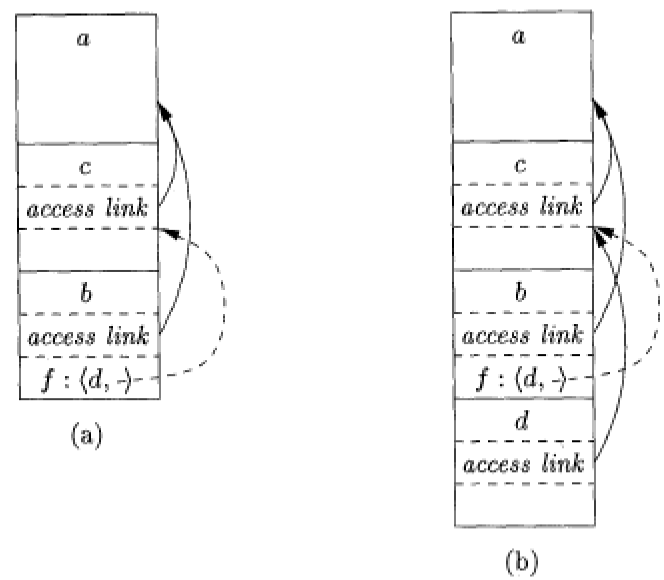
Non-Local Data Access (nested procedures)

```
 1) fun sort(inputFile, outputFile) =
        let
 2)         val a = array(11,0);
 3)         fun readArray(inputFile) = ··· ;
 4)                ··· a ··· ;
 5)         fun exchange(i,j) =
 6)                ··· a ··· ;
 7)         fun quicksort(m,n) =
                let
 8)                 val v = ··· ;
 9)                 fun partition(y,z) =
10)                        ··· a ··· v ··· exchang
                in
11)                 ··· a ··· v ··· partition ··· quicksort
                end
        in
12)         ··· a ··· readArray ··· quicksort ···
        end;
```



Nested Procedure with functions as parameters

```
fun a(x) =
    let
        fun b(f) =
            ... f ... ;
        fun c(y) =
            let
                fun d(z) = ...
            in
                ... b(d) ...
            end
    in
        ... c(1) ...
    end;
```



Function Parameter must carry as "hidden parameter" the access link
Code must generated to install the link as part of the call

ABI: Application Binary Interface

- An Application Binary Interface (ABI) specifies an interface for compiled application programs to system software
- The "contract" that specifies
  – how functions are called
  – how parameters are passed
  – how the linkage is defined
  - what assumption can be made (and not)

PowerPC ABI

- Register Usage Convention

Table 3 - PowerPC EABI register usage

Register	Type	Used for:
R0	Volatile	Language Specific
R1	Dedicated	Stack Pointer (SP)
R2	Dedicated	Read-only small data area anchor
R3 - R4	Volatile	Parameter passing / return values
R5 - R10	Volatile	Parameter passing
R11 - R12	Volatile	
R13	Dedicated	Read-write small data area anchor
R14 - R31	Nonvolatile	
F0	Volatile	Language specific
F1	Volatile	Parameter passing / return values
F2 - F8	Volatile	Parameter passing
F9 - F13	Volatile	
F14 - F31	Nonvolatile	
Fields CR2 - CR4	Nonvolatile	
Other CR fields	Volatile	
Other registers	Volatile	

PowerPC ABI

- Datatypes

Table 1 - PowerPC scalar data types

Data type	Size (bytes)
Byte	1
Halfword	2
Word	4
Doubleword	8
Quadword	16

Table 2 - PowerPC ANSI C data types

ANSI C data type	PowerPC Data type	Size (bytes)
char	byte	1
short	halfword	2
int	word	4
long int	word	4
enum	word	4
pointer	word	4
float	word	4
double	doubleword	8
long double	quadword	16

- Function Call

Prologue

```
FuncX: mflr %r0              ; Get Link register
       stwu %r1,-88(%r1)     ; Save Back chain and move SP
       stw  %r0,+92(%r1)     ; Save Link register
       stmw %r28,+72(%r1)    ; Save 4 non-volatiles r28-r31
```

Epilogue

```
       lwz  %r0,+92(%r1)     ; Get saved Link register
       mtlr %r0              ; Restore Link register
       lmw  %r28,+72(%r1)    ; Restore non-volatiles
       addi %r1,%r1,88       ; Remove frame from stack
       blr                   ; Return to calling function
```

PowerPC ABI

- Stack Frame Convention

```
FuncX: mflr %r0                  ; Get Link register
       stwu %r1,-88(%r1)         ; Save Back chain and move SP
       stw  %r0,+92(%r1)         ; Save Link register
       stmw %r28,+72(%r1)        ; Save 4 non-volatiles r28-r31


       lwz  %r0,+92(%r1)         ; Get saved Link register
       mtlr %r0                  ; Restore Link register
       lmw  %r28,+72(%r1)        ; Restore non-volatiles
       addi %r1,%r1,88           ; Remove frame from stack
       blr                       ; Return to calling function
```

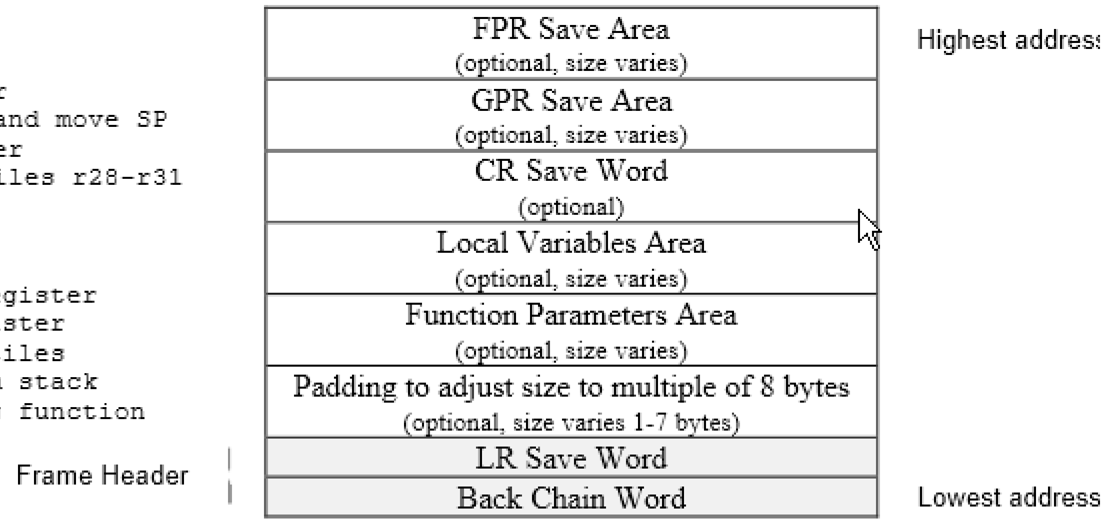
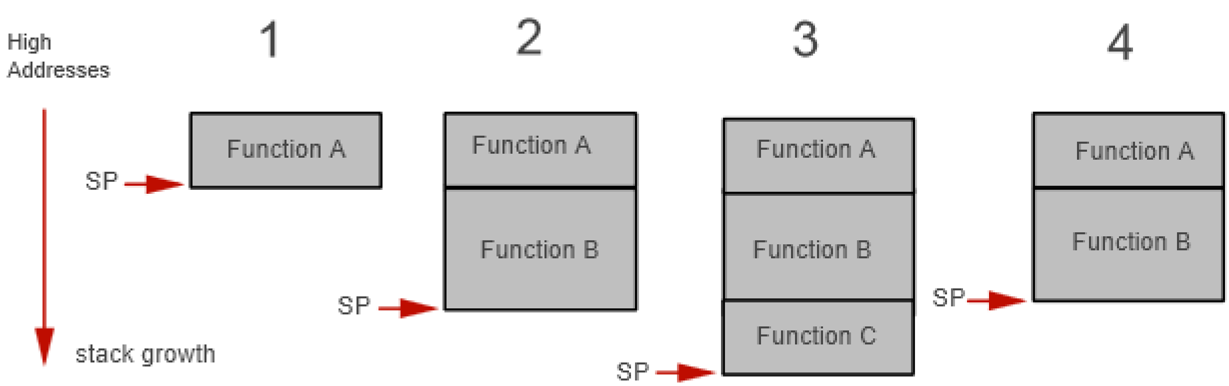


Figure 2 - EABI Stack Frame



X86-64 ABI

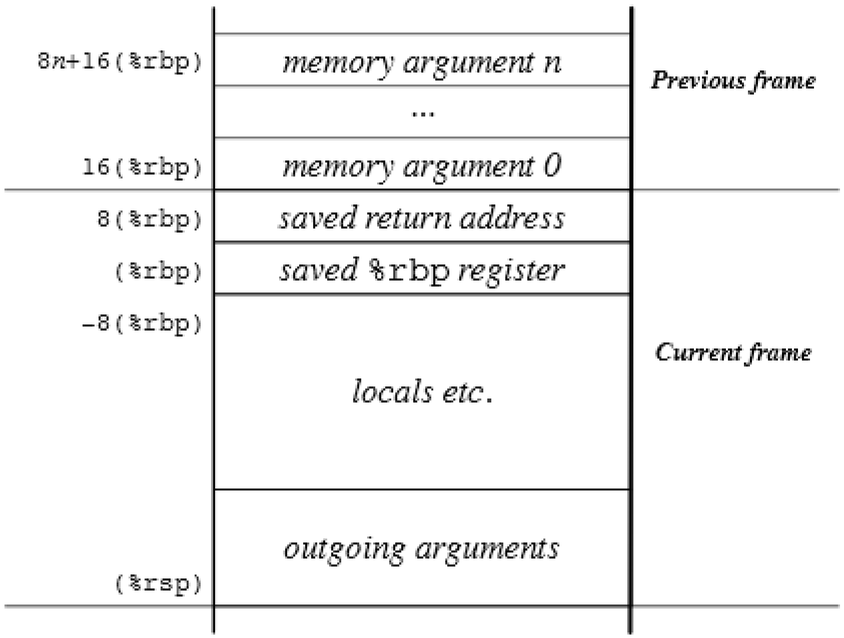
```c
int callee(int, int, int);

int caller(void)
{
        register int ret;

        ret = callee(1, 2, 3);
        ret += 5;
        return ret;
}
```

```
        .globl  caller
caller:
        pushl   %ebp
        movl    %esp,%ebp
        pushl   $3
        pushl   $2
        pushl   $1
        call    callee
        addl    $12,%esp
        addl    $5,%eax
        leave
        ret
```

- Stacks are aligned @ 16bytes

X86-64

Register	Callee Save	Description
%rax		result register; also used in idiv and imul instructions.
%rbx	yes	miscellaneous register
%rcx		fourth argument register
%rdx		third argument register; also used in idiv and imul instructions.
%rsp		stack pointer
%rbp	yes	frame pointer
%rsi		second argument register
%rdi		first argument register
%r8		fifth argument register
%r9		sixth argument register
%r10		miscellaneous register
%r11		miscellaneous register
%r12-%r15	yes	miscellaneous registers

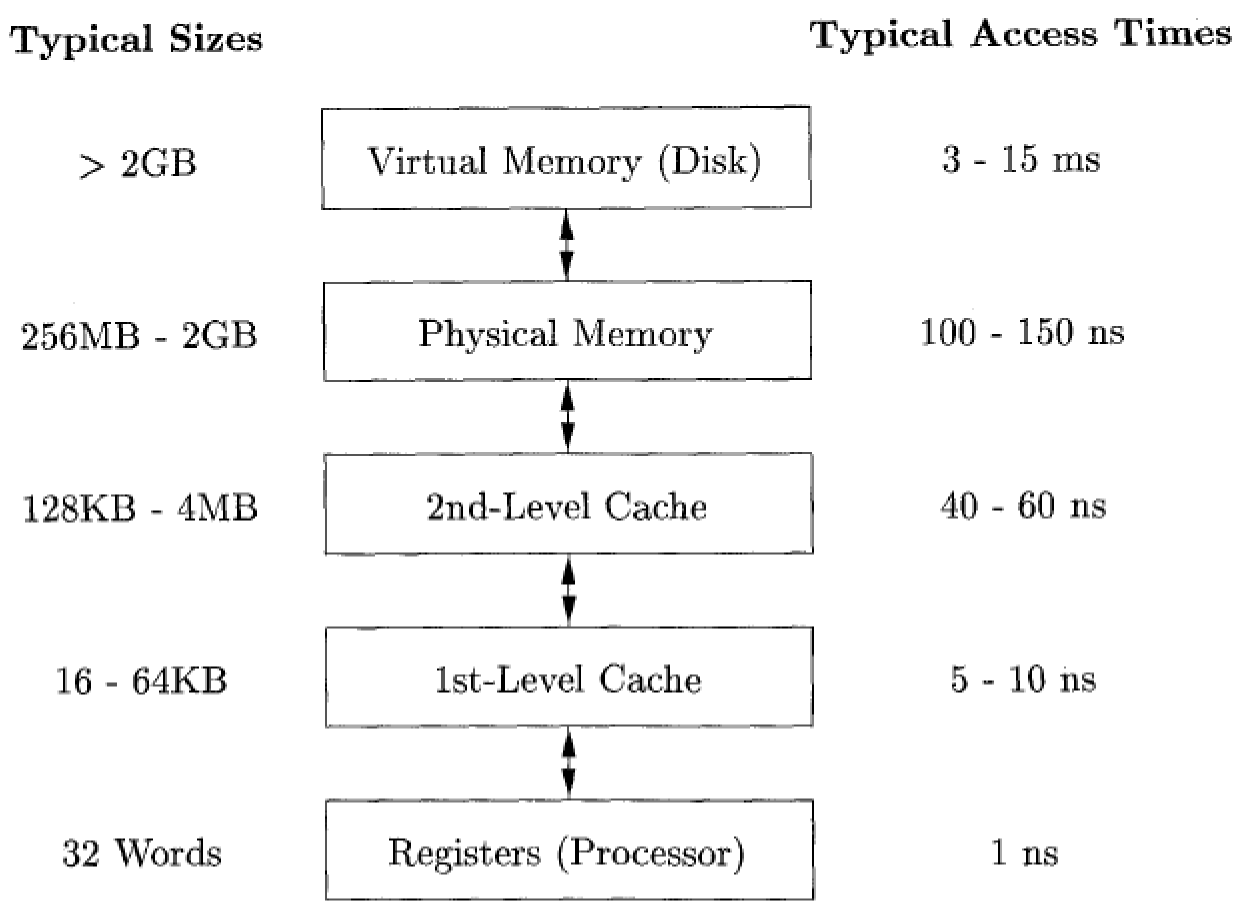


Heap Management

- Heap: portion of the store used for data that lives indefinitely
- Memory manager: subsystem responsible for (de)allocation of space within the heap
- Garbage collection: process of finding spaces within the heap that are no longer used and reallocate them to other data items

Memory Manager

- Keeps track of all the free space in heap at all time
- Allocation
  - Interaction with OS
- Deallocation
- Desired properties:
  - Space efficiency: minimize total heap space needed by programs
  - Program efficiency: making good use of memory subsystem
  - Low overhead: of (de)allocation processes


Typical Sizes
Typical Access Times
> 2GB
Virtual Memory (Disk)
3 - 15 ms
256MB - 2GB
Physical Memory
100 - 150 ns
128KB - 4MB
2nd-Level Cache
40 - 60 ns
16 - 64KB
1st-Level Cache
5 - 10 ns
32 Words
Registers (Processor)
1 ns

Heap Fragmentation

- Due to allocation/deallocation
- Why is it bad?
- How to deal with it?
  - Best fit
  - First fit
  - Next fit
  - Worst fit

Garbage Collection

- Garbage: data that cannot be referenced
- Garbage collection: reclamation of garbage from heap

Assumptions

- Objects have a type that can be determined by garbage collector at run-time.
- References to objects are always to the address of the beginning of the object.

Performance Metrics

- Overall execution time: garbage collection can be very slow
- Space usage: must avoid fragmentation
- Maximum pause time must be minimized
- Program locality

Reference-Counting Garbage Collection

- Every object must have a field for reference count
- This field counts the number of references to the object
- If count reaches zero, the object is deleted

Mark-and-Sweep

- Visit every object
- Mark object
- All unmarked objects are unreachable
    - » Can be deleted

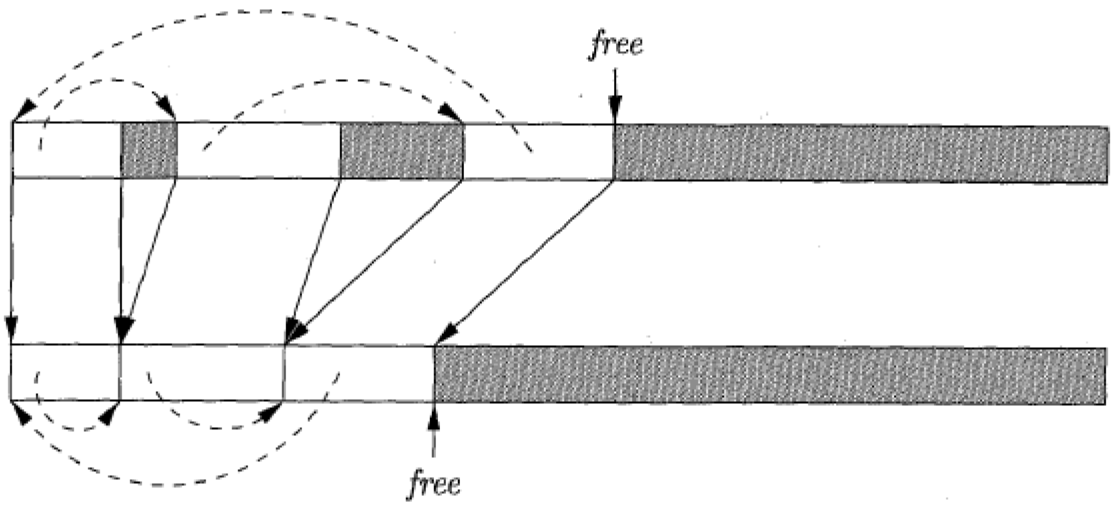
Mark-and-Compact

- Variation of Mark-and-Sweep
- Copy remaining objects into small contiguous area
- Why?

- In place compaction

Copying collectors

- Compacting at one end of the heap



Others

- Incremental Garbage Collectors

- Generational Garbage Collectors

So

- Skim: 7.3, 7.5.2, 7.6, 7.7, and 7.8
- Read: the rest of chp 7