Stacks, Frames, and Calling Conventions

Stack

- Calling Convention
  - An agreement, usually created by a system's designers, on how function calls should be implemented
- Stack
  - A portion of memory managed in a last-in, first-out (LIFO) fashion
- Function Call
  - A control transfer to a segment of code that ends with a return to the point in code immediately after where the call was made (the *return address*)

Stack Frame

- An object containing all the necessary data for a function
  - Values of parameters
  - Count of number of arguments
  - Return address
  - Return value
  - Value of $SP for Activation Record Below

Temporary Storage

- Caller-Saved
  - A piece of data (e.g., a register) that must be explicitly saved if it needs to be preserved across a function call
- Callee-Saved
  - A piece of data (e.g., a register) that must be saved by a called function before it is modified, and restored to its original value before the function returns

MIPS Calling Convention

- First 4 arguments $a0-$a3
  - Remainder put on stack
- Return values $v0-$v1
- $t0-$t9 are caller-saved temporaries
- $s0-$s9 are callee-saved

x86 Calling Convention

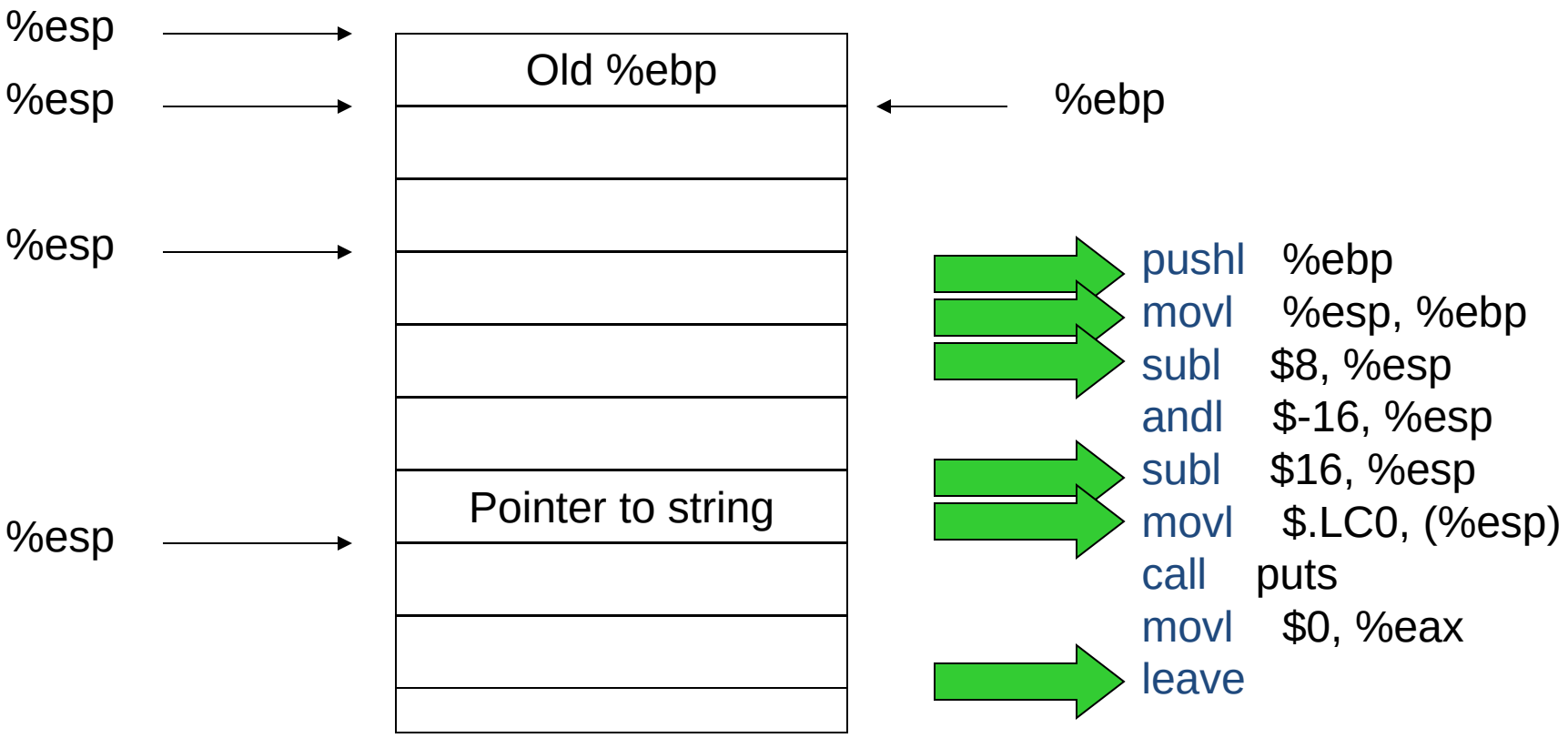
- %eax, %ecx, and %edx are generally caller-saved
- Three registers are probably insufficient
  - Most registers are “spilled” onto the stack

- %eax is the return value
- Everything else is on the stack

Hello World

```
 .file   "asm.c"
        .section        .rodata.str1.1,"aMS",@progbits,1
.LC0:
        .string "hello world!"
        .text
.globl main
        .type   main, @function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        andl    $-16, %esp          ;1111 1111 1111 0000
        subl    $16, %esp
        movl    $.LC0, (%esp)
        call    puts
        movl    $0, %eax
        leave
        ret
        .size   main, .-main
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.4.6 20060404 (Red Hat 3.4.6-8)"
```

Hello World Stack

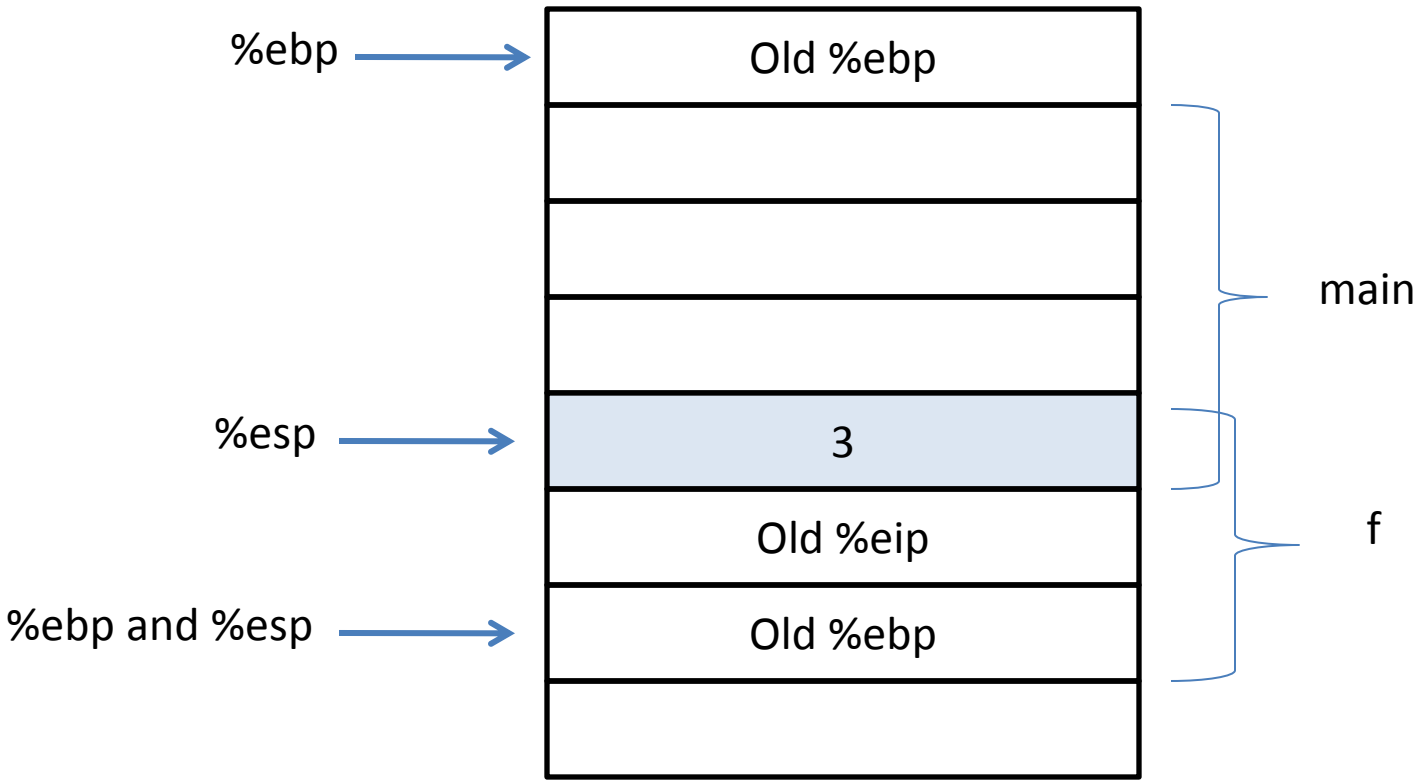


Function Call, 1 param

```
#include <stdio.h>

int f(int x)
{
        return x;
}

int main()
{
        int y;

        y = f(3);

        return 0;
}
```

```
f:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %eax
        leave
        ret

main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        andl    $-16, %esp
        subl    $16, %esp
        movl    $3, (%esp)
        call    f
        movl    %eax, -4(%ebp)
        movl    $0, %eax
        leave
        ret
```

Stack

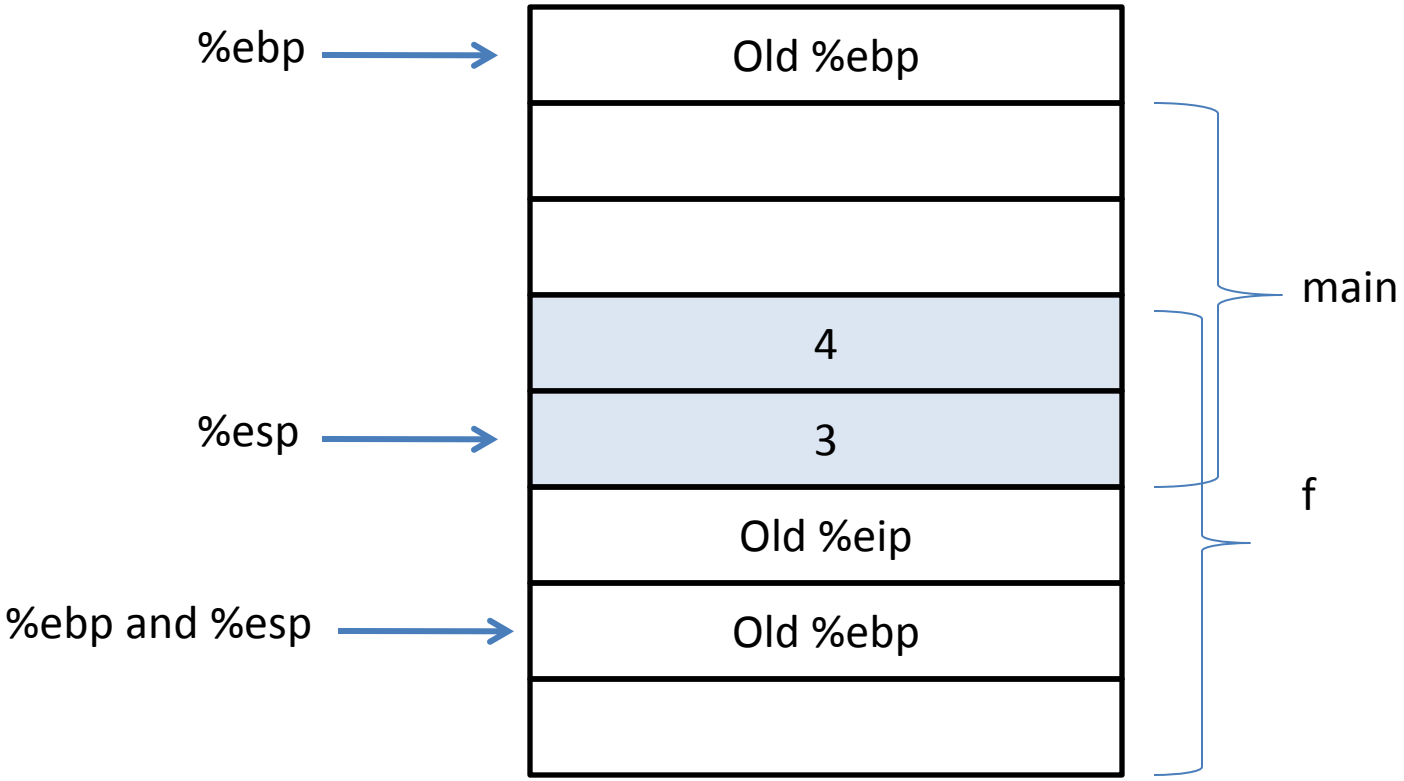


Function Call, 2 params

```
#include <stdio.h>

int f(int x, int y) {
        return x + y;
}

int main(int argc, char ** argv) {
        int y = 0;

        y = f(3, 4);

        return 0;
}
```

```
f:      pushl   %ebp
        movl    %esp, %ebp
        movl    12(%ebp), %eax
        addl    8(%ebp), %eax
        leave
        ret
main:   pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        andl    $-16, %esp
        subl    $16, %esp
        movl    $4, 4(%esp)
        movl    $3, (%esp)
        call    f
        movl    %eax, 4(%esp)
        movl    $0, %eax
        leave
        ret
```

Stack



Observation

- Parameters are pushed right to left onto the stack
- Why?

printf

```
int printf(const char * fmt, ...);
```

- "…" means variable number of arguments

#include <stdarg.h>

```
int * makearray(int cnt, ...) {
   va_list ap;
   int i = 0;
   int * array = (int *)malloc(cnt * sizeof(int));

   va_start(ap, cnt);
   for (i = 0; i < cnt, i++) {
       array[i] = va_arg(ap, int);
   }

   va_end(ap);
   return array;
 }
```

Variable Arguments Usage

```
int main(int argc, char ** argv) {
   int * p = NULL;
   int i = 0;
   p = makearray(4, 1,2,3,4);

   for (i = 0; i < 4; i++) {
       printf("%d\n", p[i]);
   }

   return 0;
}
```

Other Notes

- Also called a *Variadic* function
- Old header <varargs.h> no longer favored for use
- C99 Includes support for Variadic Macros
- Java:

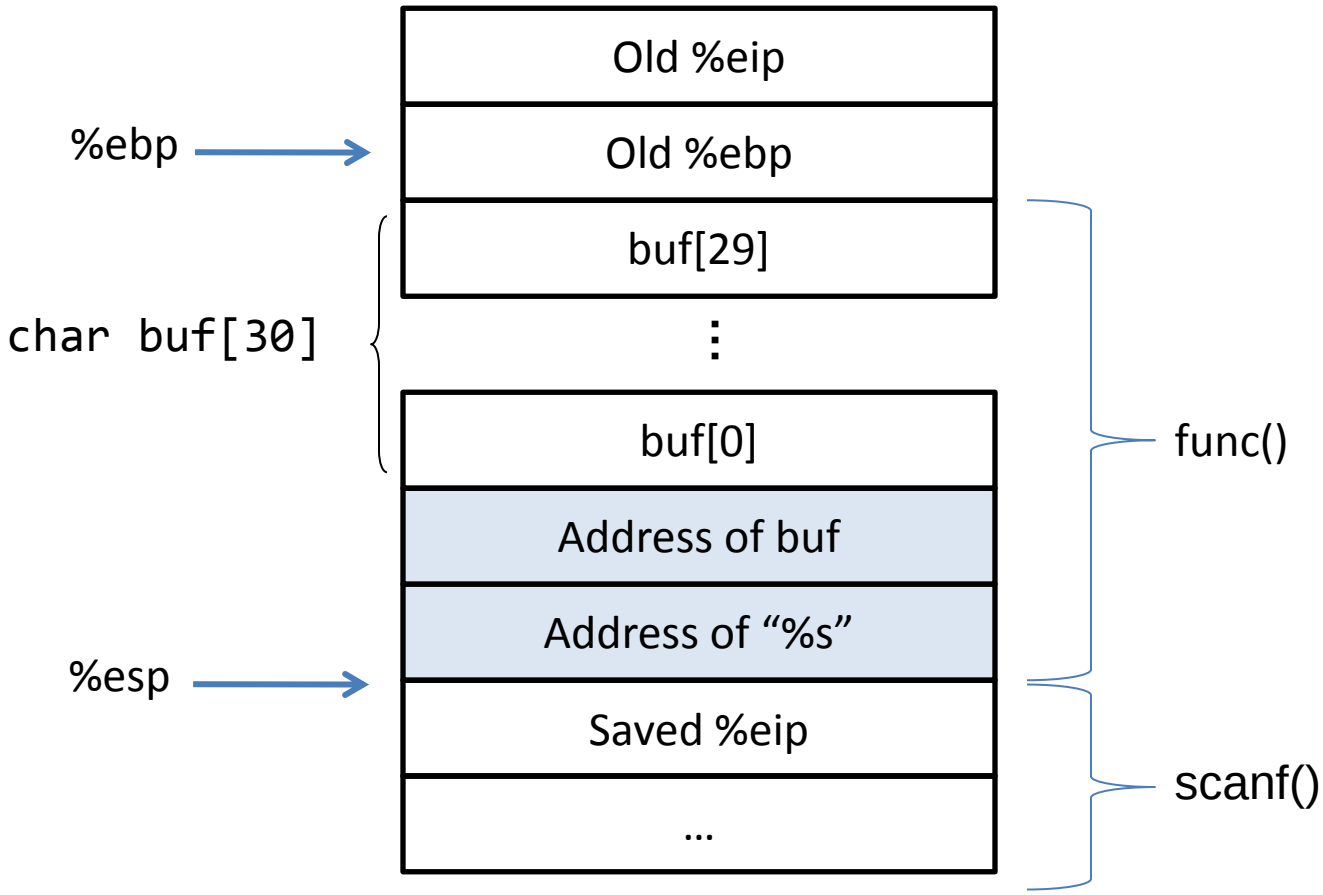
```
public static void printArray(Object... objects) {
   for (Object o : objects)
        System.out.println(o);
}

printArray(3, 4, “abc”);
```

Stack Allocated Array

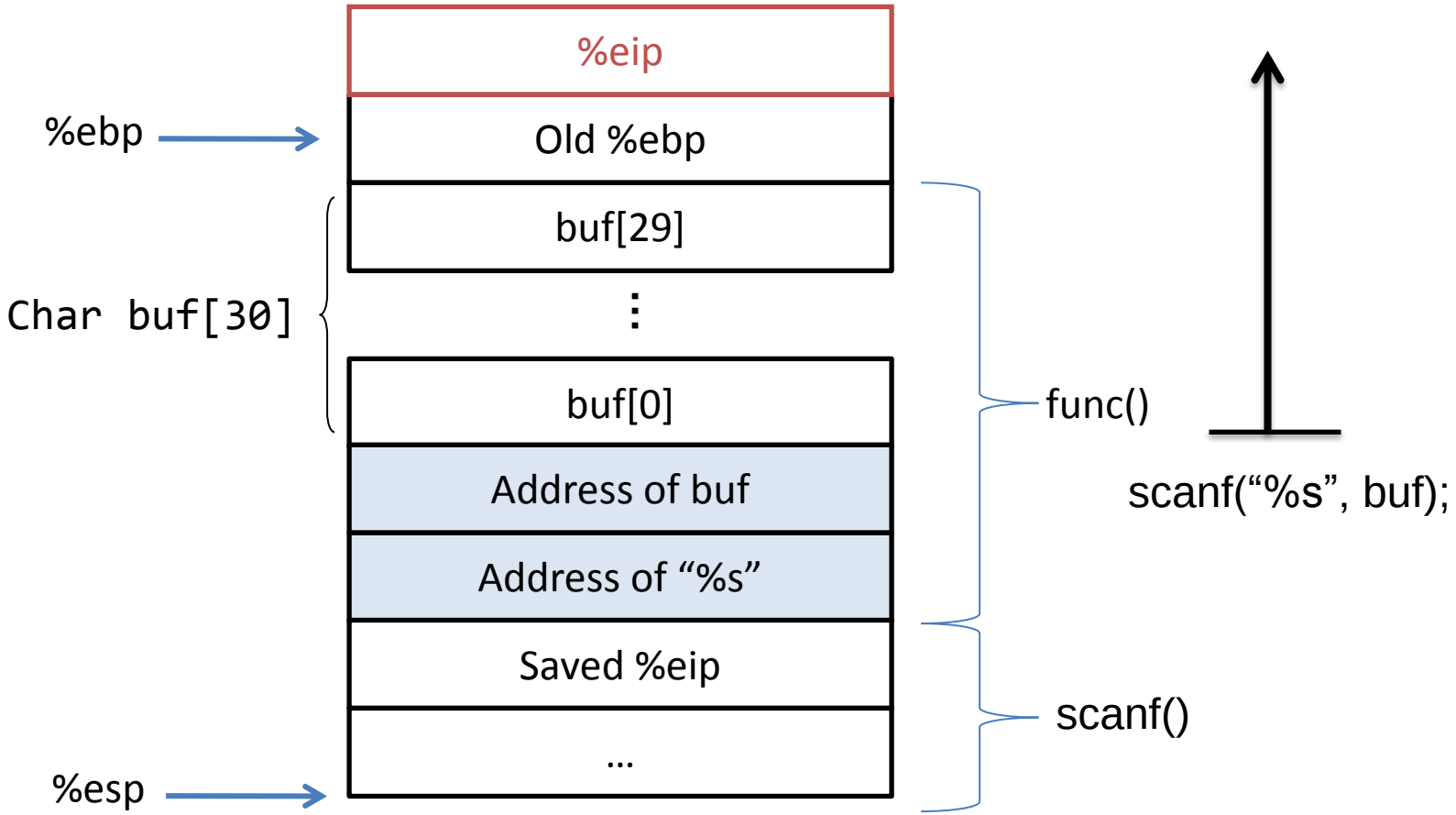
```
int func() {
   char buf[30];

   scanf("%s", buf);

   return 0;
}
```

```
func:
   pushl   %ebp
   movl    %esp, %ebp
   andl    $-16, %esp
   subl    $48, %esp
   movl    $.LC0, %eax
   leal    18(%esp), %edx
   movl    %edx, 4(%esp)
   movl    %eax, (%esp)
   call    scanf
   leave
   ret
```

Stack


Old %eip
%ebp
Old %ebp
buf[29]
char buf[30]
buf[0]
Address of buf
Address of “%s”
%esp
Saved %eip
…
func()
scanf()

Buffer Overrun



Buffer Overrun Vulnerability