Function Calls and Calling Conventions 2

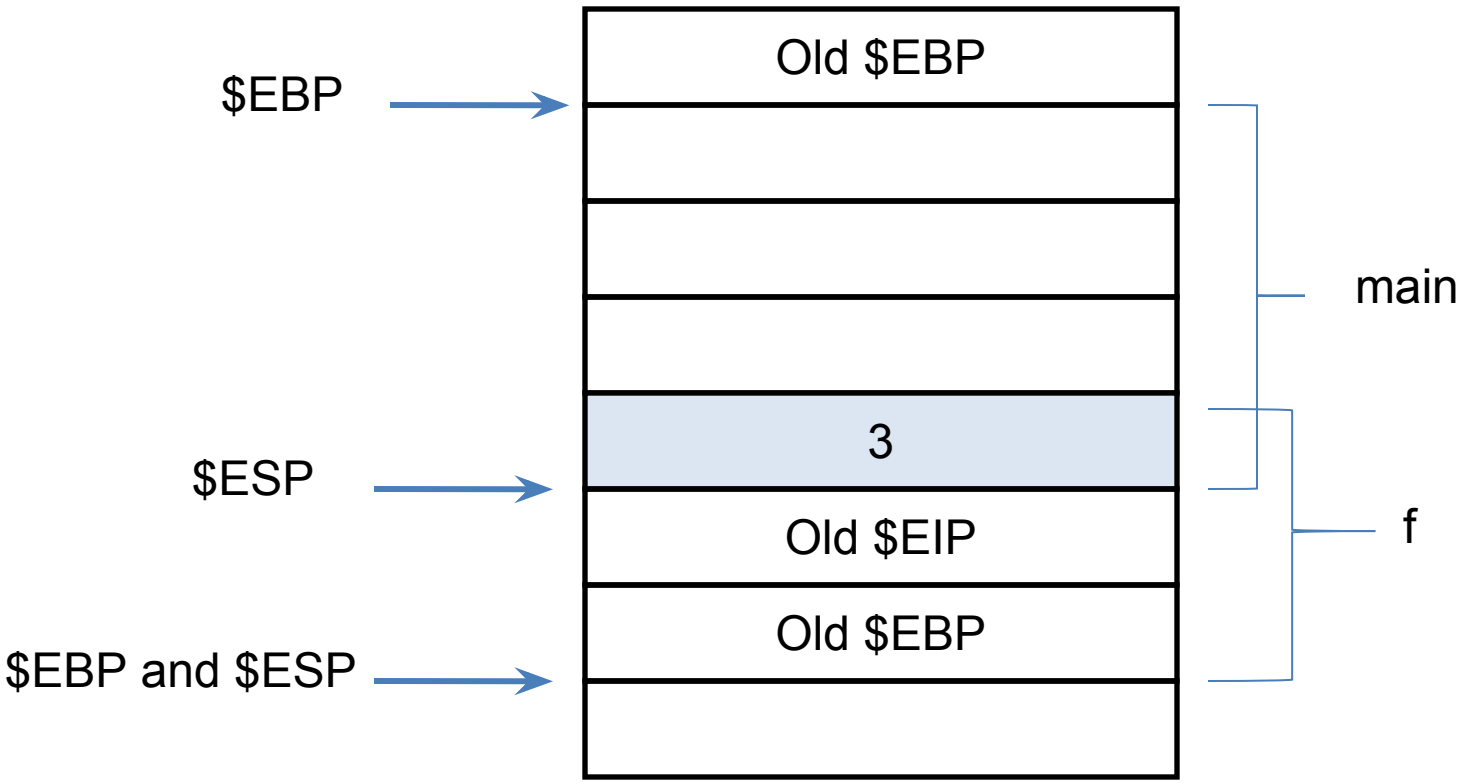
CS449 Spring 2016

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

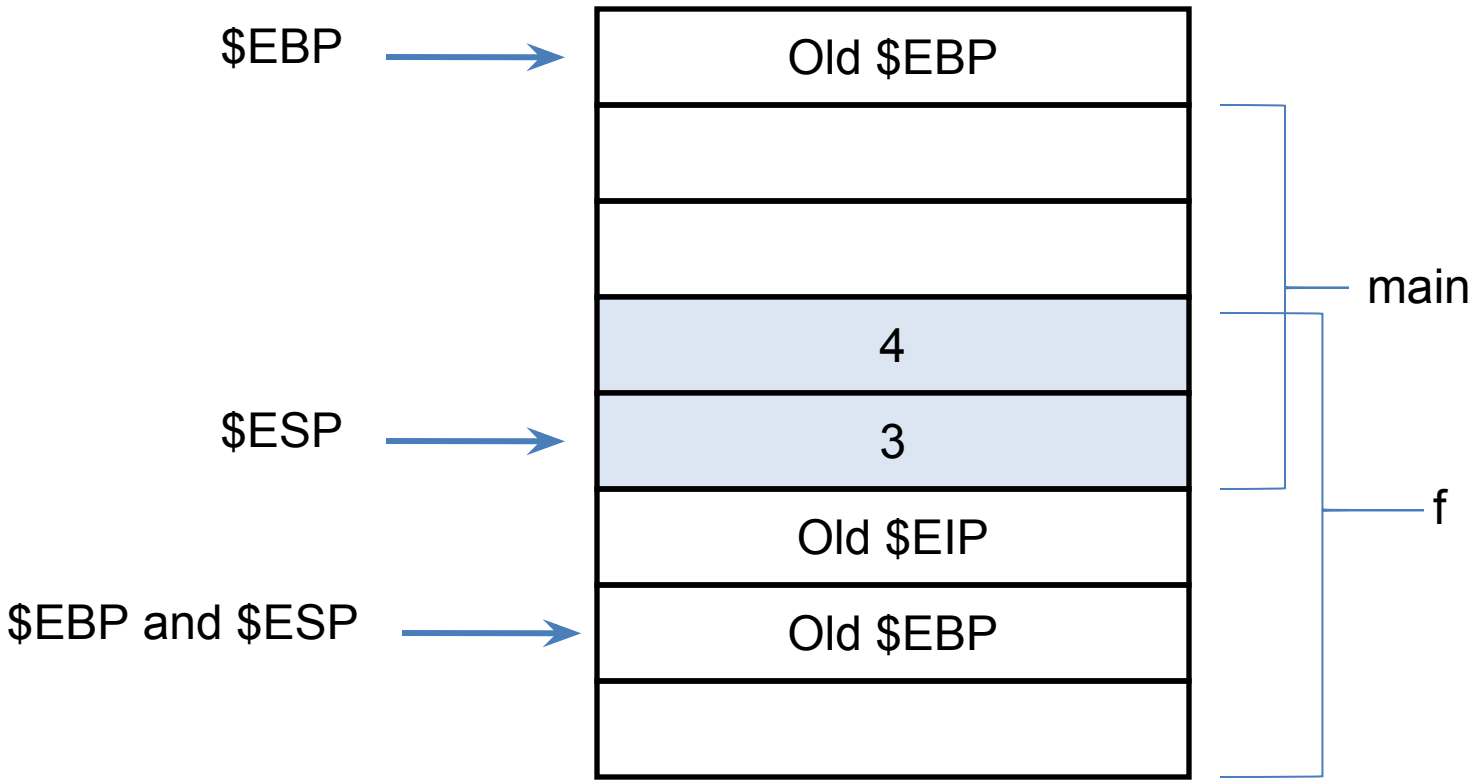


Function Call, 2 params

```
#include <stdio.h>

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

int main()
{
        int y;

        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 *format,...);
```

- "…" means variable number of arguments
- *format* must be pushed last
  - printf must first parse *format* to discover the number of arguments
  - Pushing *format* last fixes its location relative to EBP (base pointer)

#include <stdarg.h>

```
int *makearray(int a, ...) {
    va_list ap;
    int *array = (int *)malloc(MAXSIZE * sizeof
   (int));
    int argno = 0;
    va_start(ap, a);
    while (a > 0 && argno < MAXSIZE) {
        array[argno++] = a;
        a = va_arg(ap, int);
    }
    array[argno] = -1;
    va_end(ap);
    return array;
 }
```

Variable Arguments Usage

```
int main()
{
        int *p;
        int i;
        p = makearray(1,2,3,4,-1);

        for(i=0;i<5;i++)
                printf("%d\n", p[i]);

        return 0;
}
```

Other Notes

- Also called a *Variadic* function
- Java:

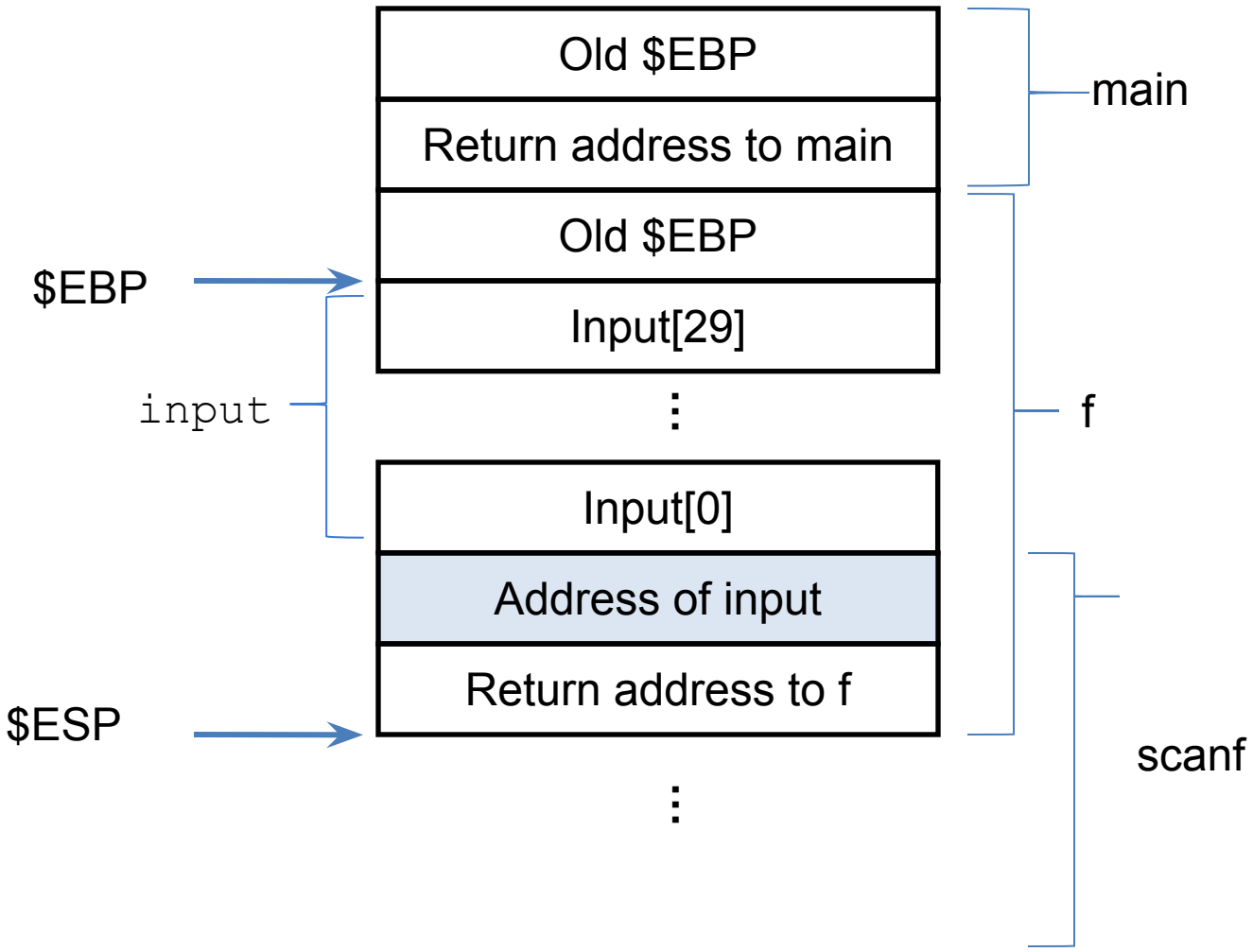
```
public static void printArray(Object... objects) {
    for (Object o : objects)
        System.out.println(o);
}

printArray(3, 4, “abc”);
```

Stack Allocated Array

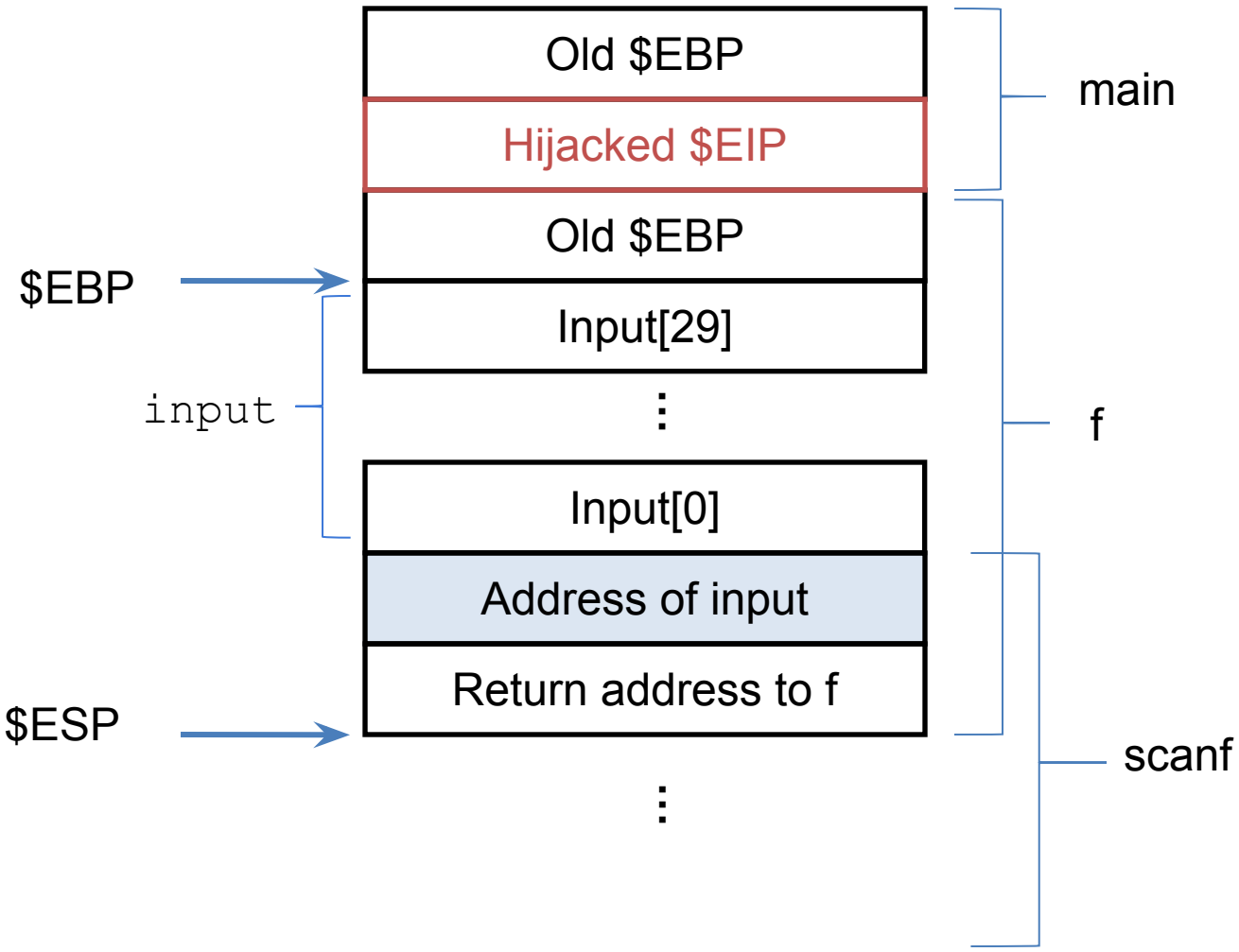
```
void f()
{
    char input[30];
    scanf("%s", input);
}

int main()
{
   f();
    return 0;
}
```

```
f:          pushl   %ebp
            movl    %esp, %ebp
            subl    $56, %esp
            leal    -38(%ebp), %eax
            movl    %eax, 4(%esp)
            movl    $.LC0, (%esp)
            call    scanf
            leave
            ret

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

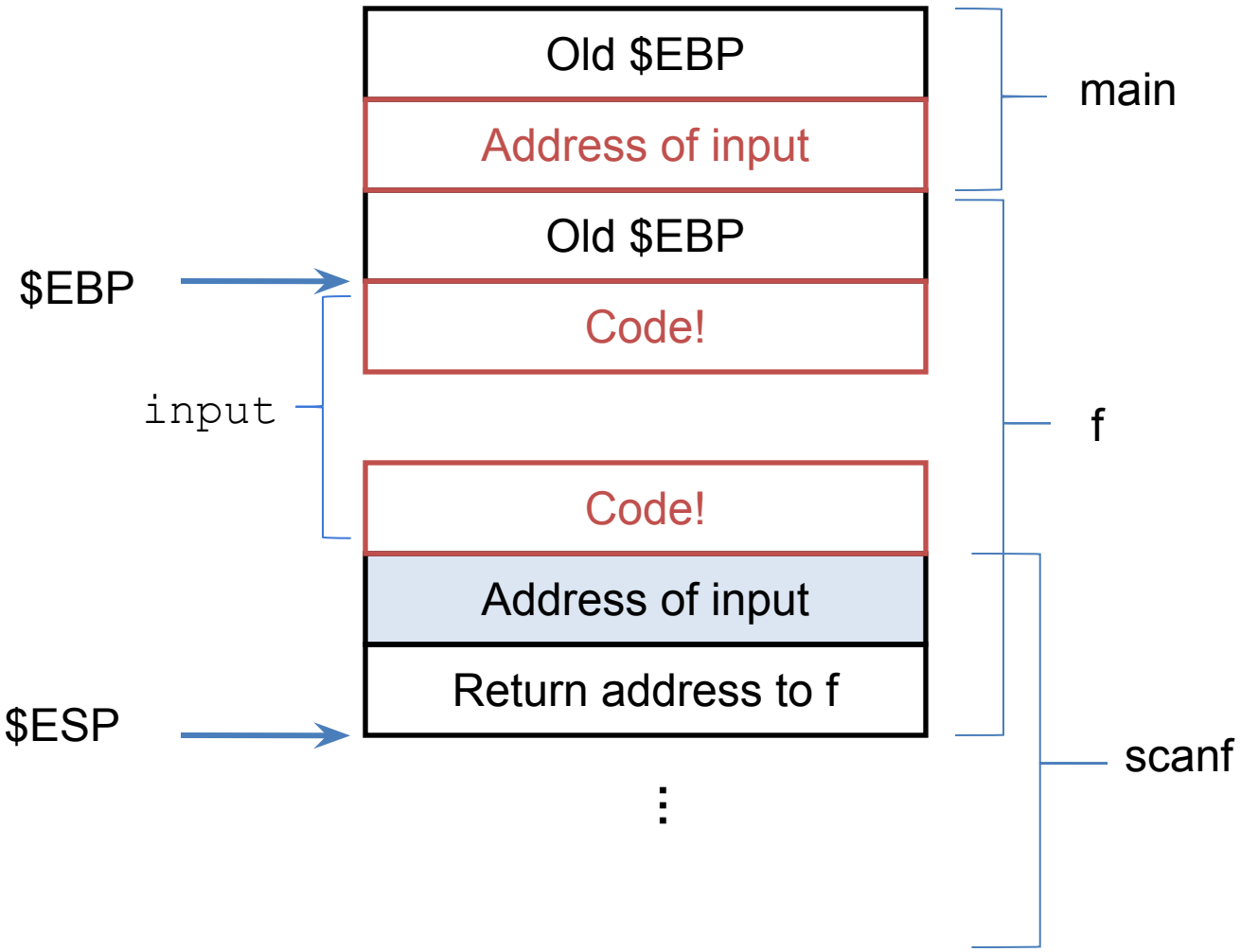
Stack


Old $EBP
Return address to main
Old $EBP
Input[29]
Input[0]
Address of input
Return address to f
$EBP
$ESP
input
main
f
scanf

Buffer Overrun



Buffer Overrun Vulnerability



Buffer Overrun Example

```
#include <stdio.h>
void bar() {
  printf("Hijacked!\n");
}
void foo() {
  char a[30];
  *(a + 34) = &bar;
}
int main() {
  foo();
  printf("Return\n");
  return 0;
}
```

```
>> gcc –m32 ./main.c
>> ./a.out
Hijacked!
Hijacked!
Segmentation fault (core dumped)
```

- Address (a+34) points to return address of foo()