X86 Assembly

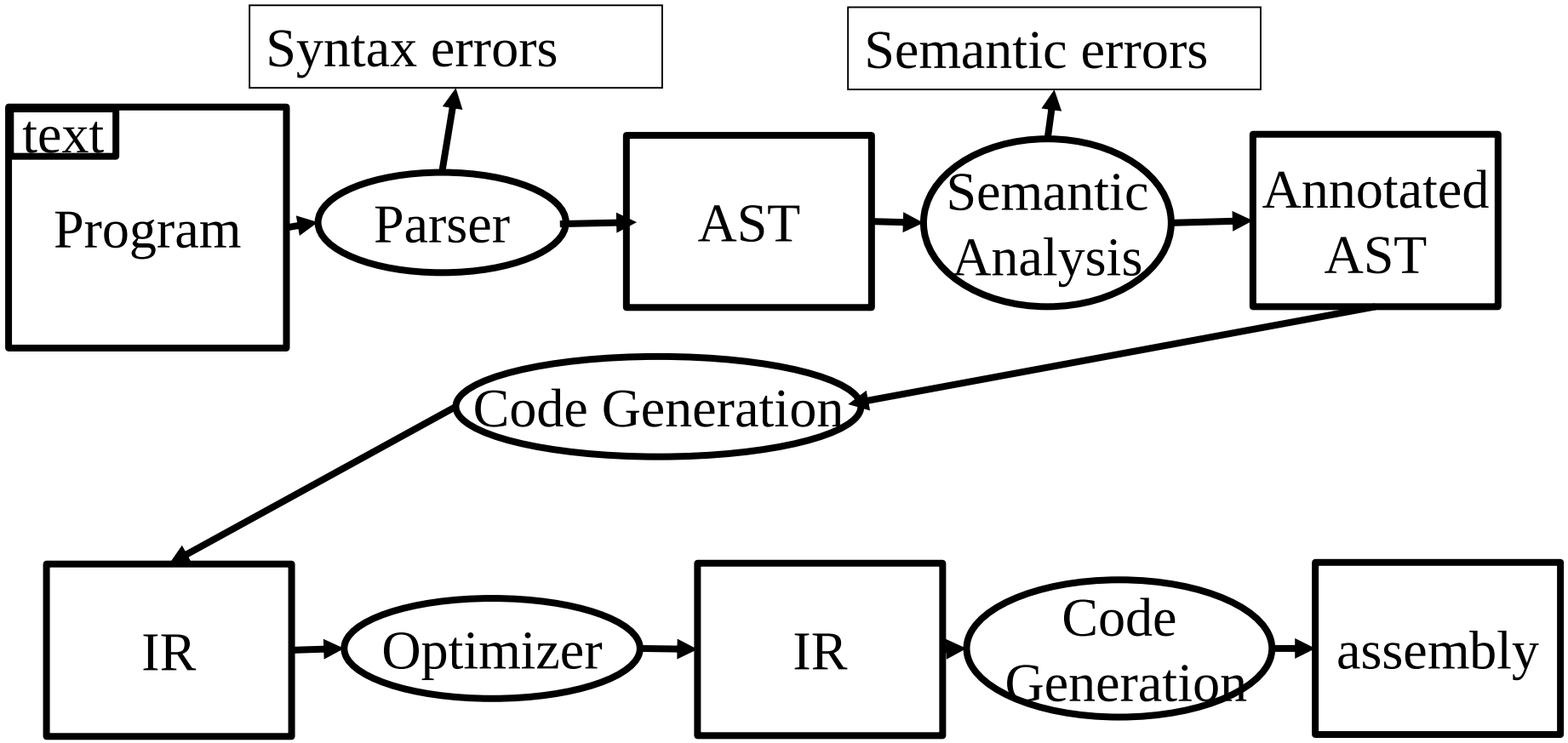
Mooly Sagiv

http://www.egr.unlv.edu/~ed/assembly64.pdf

https://godbolt.org/

https://www.cis.upenn.edu/~stevez/ CS341

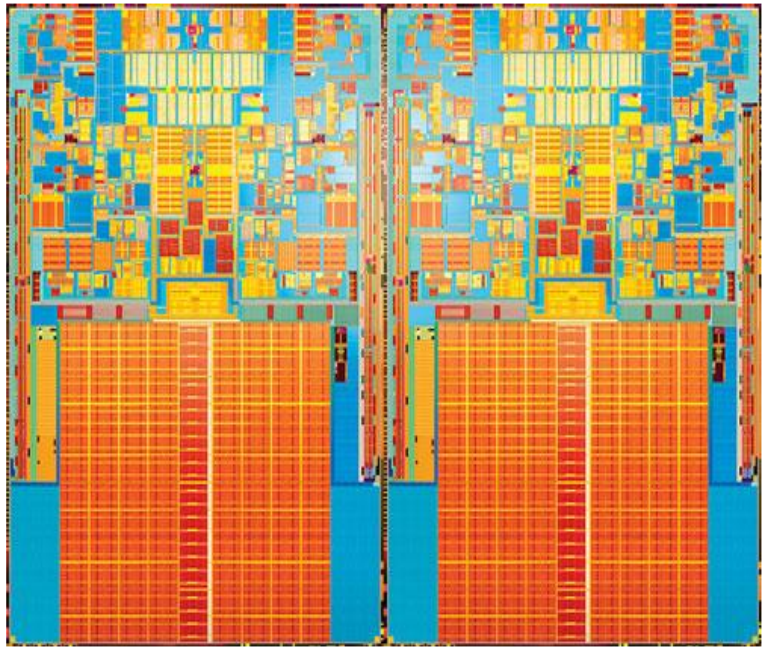
Compiler Phases

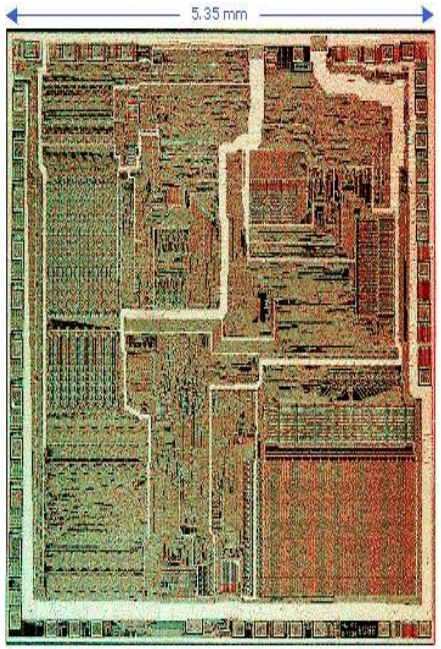


Outline

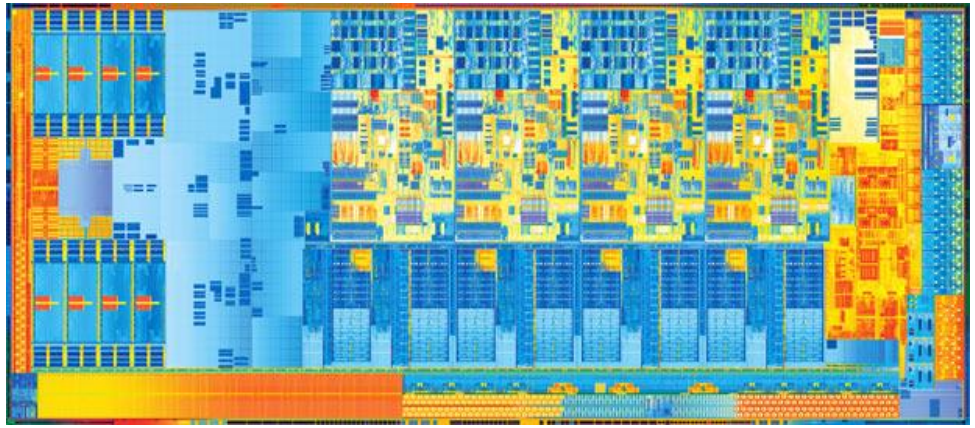
- X86 history
- Memory hierarchy
- Stack frames
- Compiling a simple example
- Running a simple example

Intel's X86 Architecture



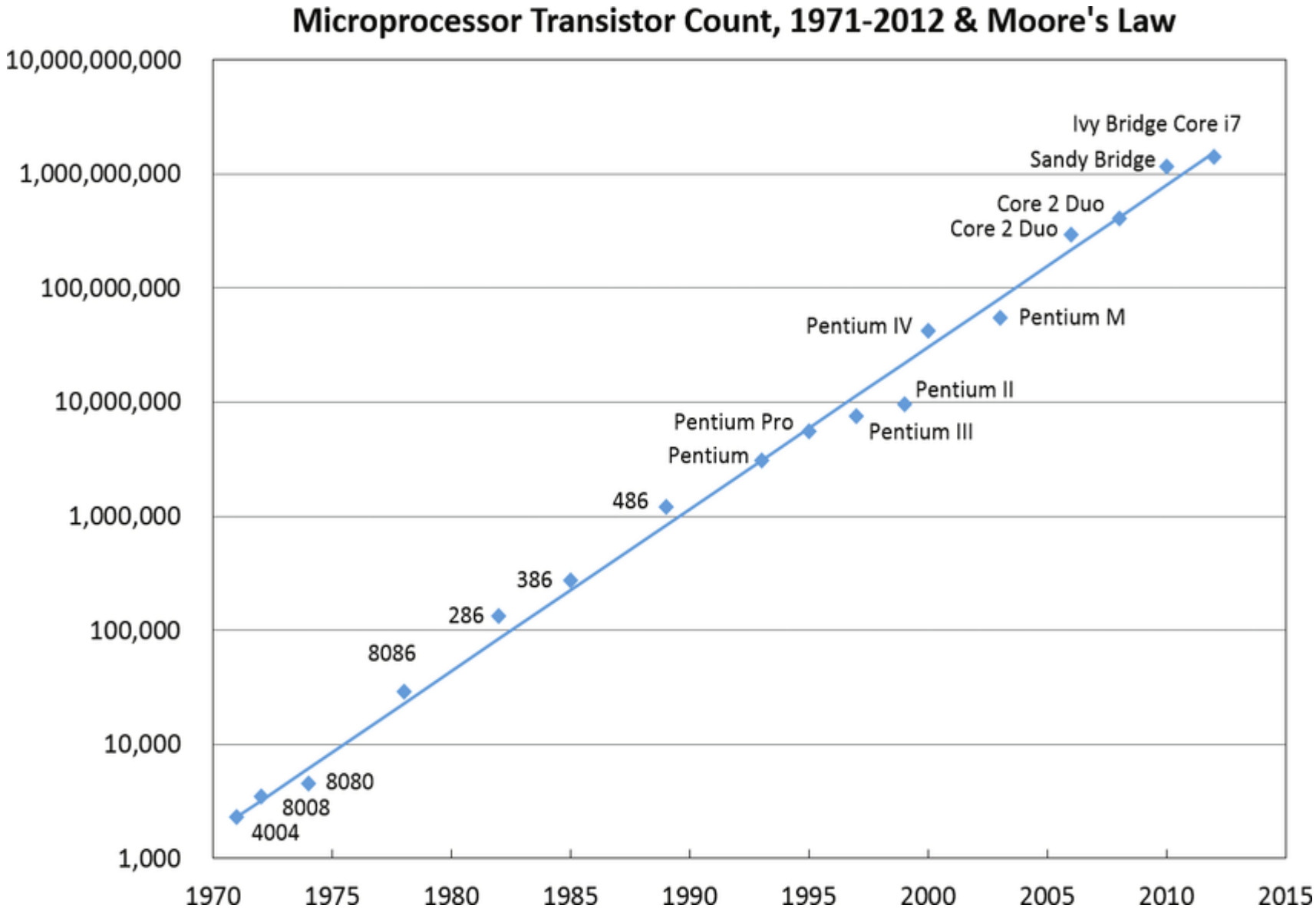


8086 internal structure

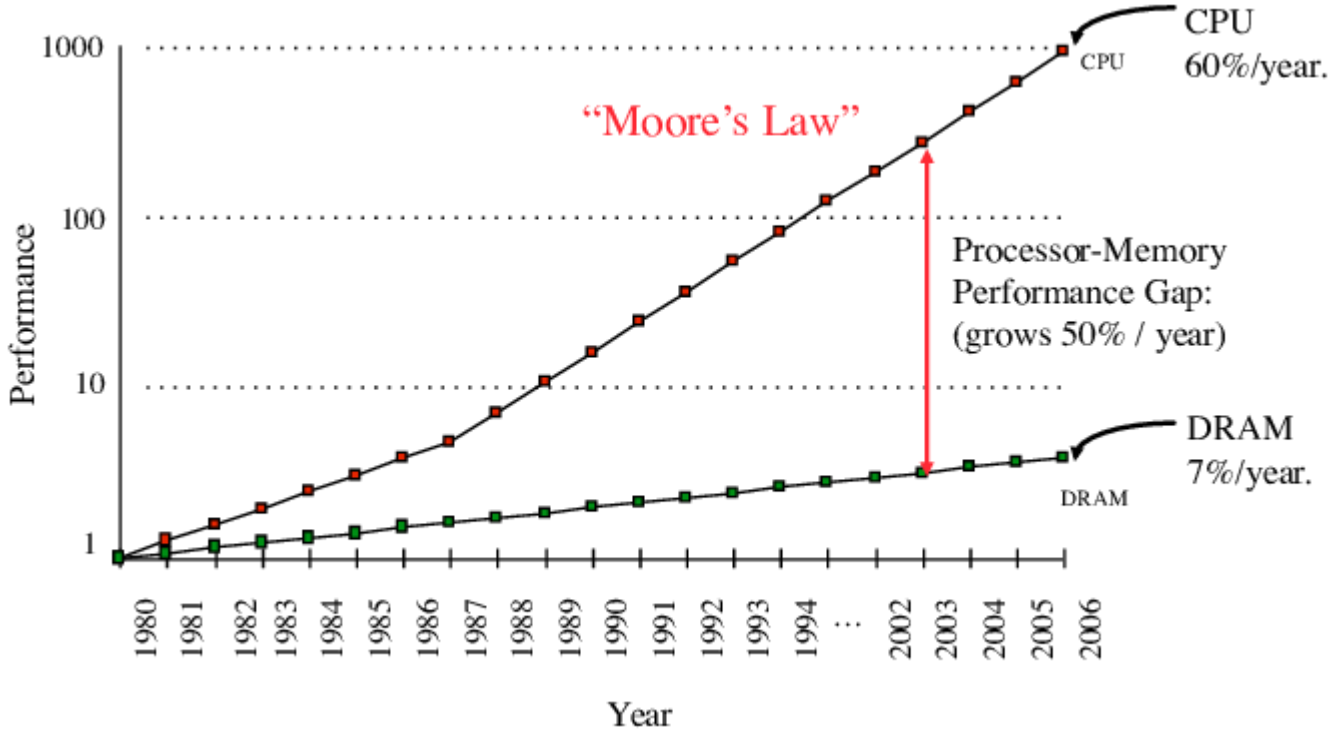


X86 history

Year	Invention
1978	Intel introduces 8086
1982	80186, 80286
1985	80386
1989	80486 (100MHz, 1µm)
1993	Pentium
1995	Pentium Pro
1997	Pentium II/III
2003	Pentium M(Banias), Intel Core
2006	Intel Core 2
2008	Intel Core i3/i5/i7
2011	SandyBridge / IvyBridge
2013	Haswell
2014	Broadwell
2015	Skylake (core i3/i5/i7/i9) (2.4GHz, 14nm)
2016	Xeon Phi


Microprocessor Transistor Count, 1971-2012 & Moore's Law
10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
Ivy Bridge Core i7
Sandy Bridge
Core 2 Duo
Core 2 Duo
Pentium IV
Pentium M
Pentium II
Pentium Pro
Pentium III
Pentium
486
386
286
8086
8080
8008
4004

Memory vs. CPU Speed

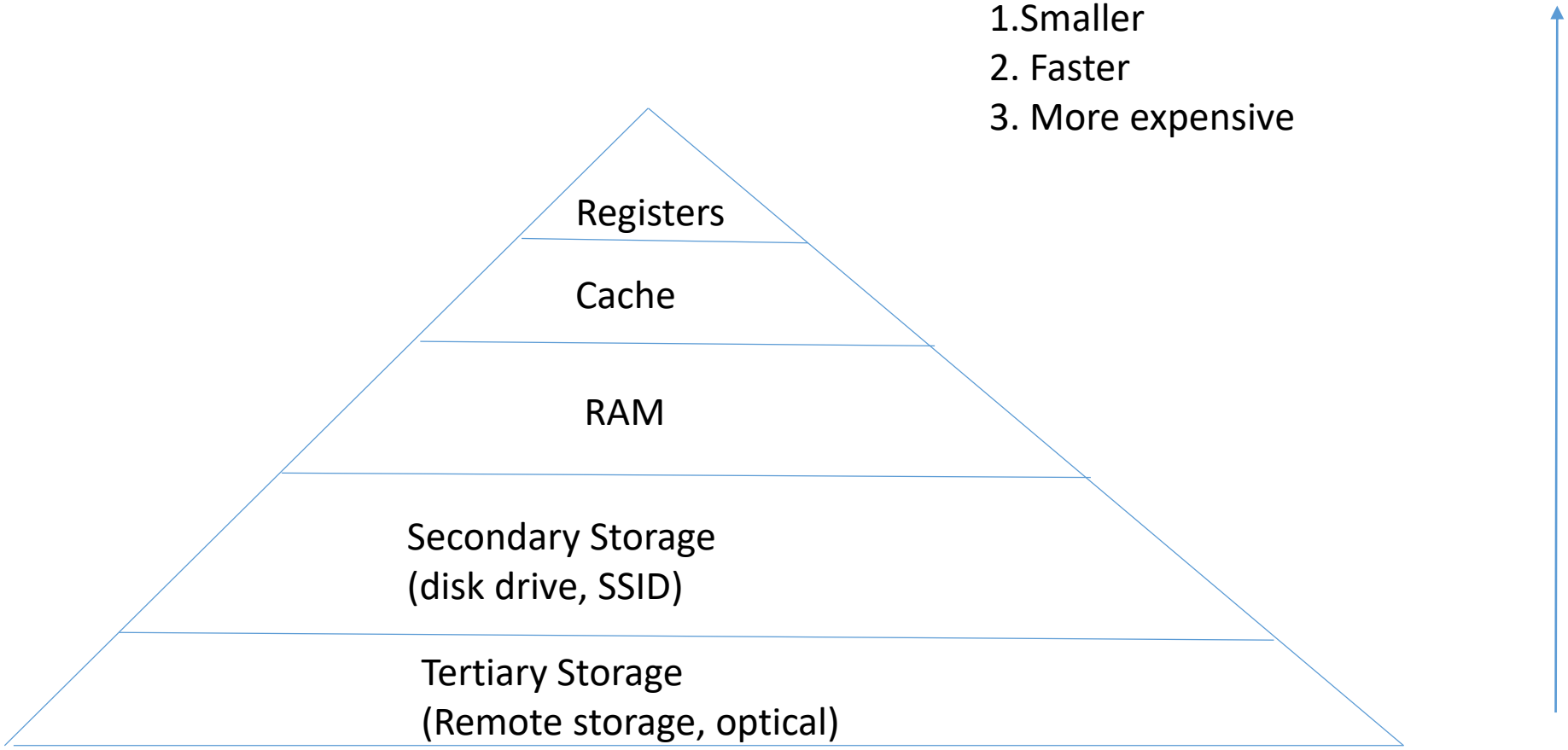


Solutions

- Architecture & Compiler
  - Machine registers utilized by compiler
- Explore locality

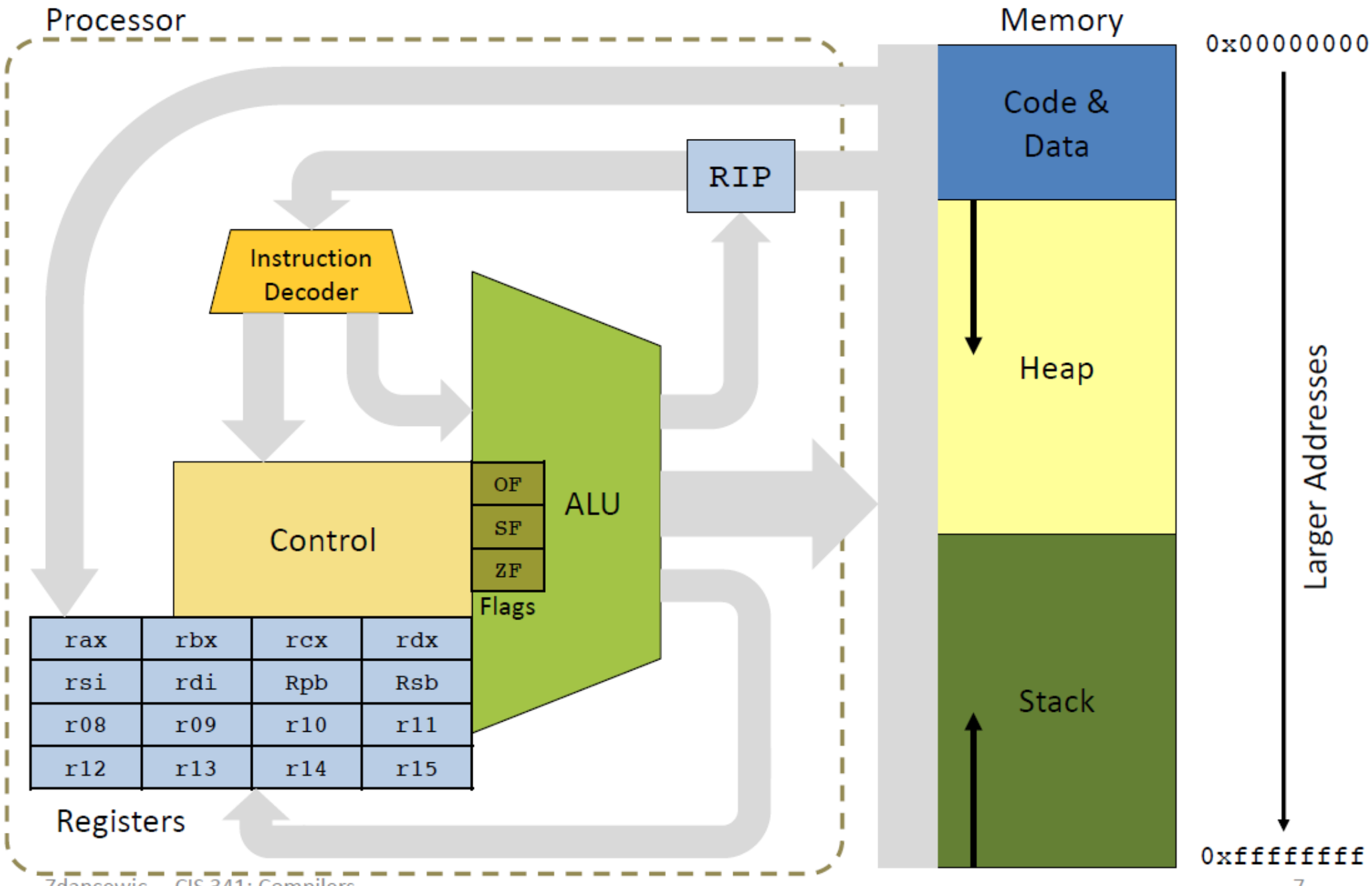
Memory Hierarchy

1.Smaller
2. Faster
3. More expensive

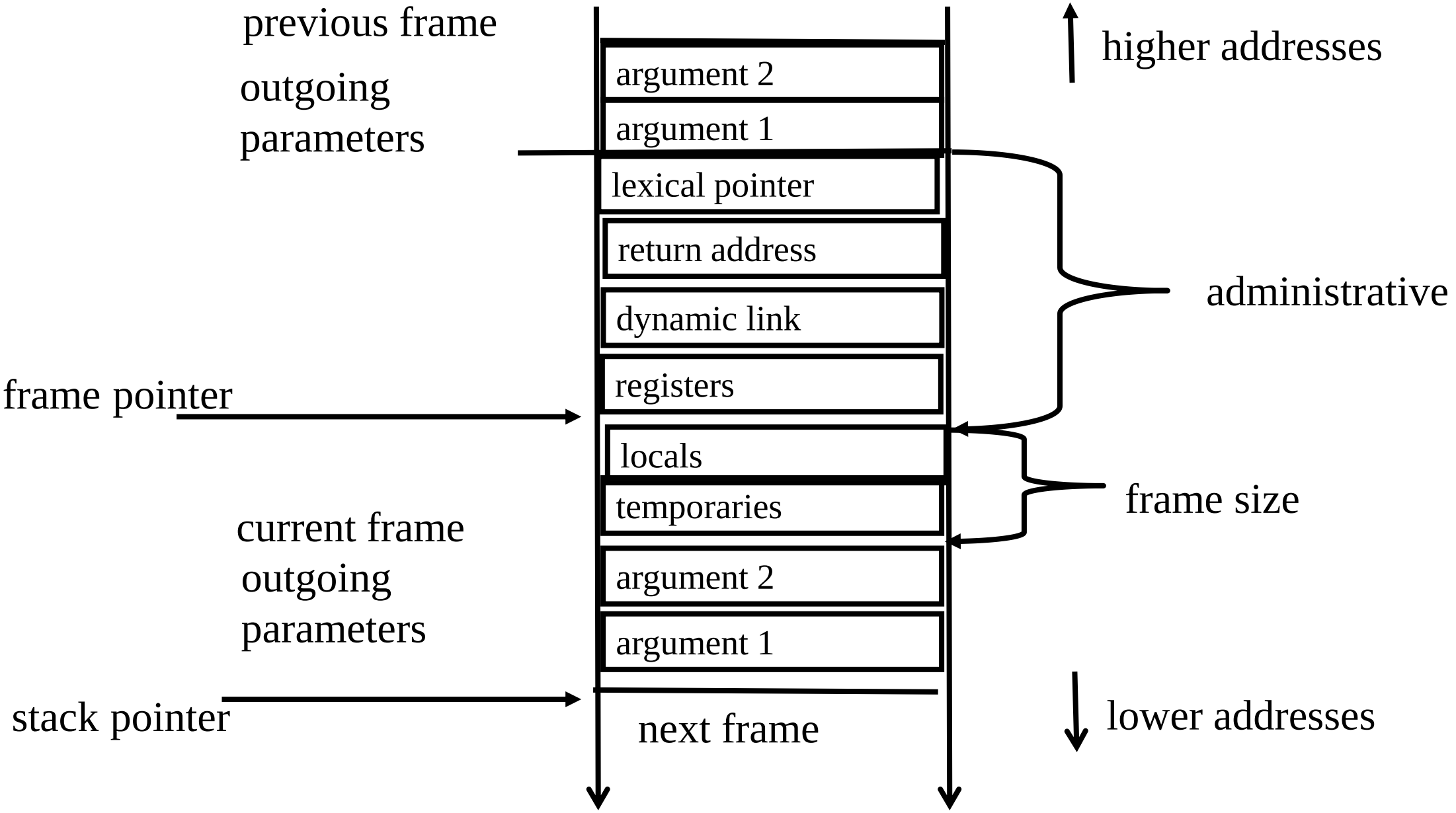


Stack Frames

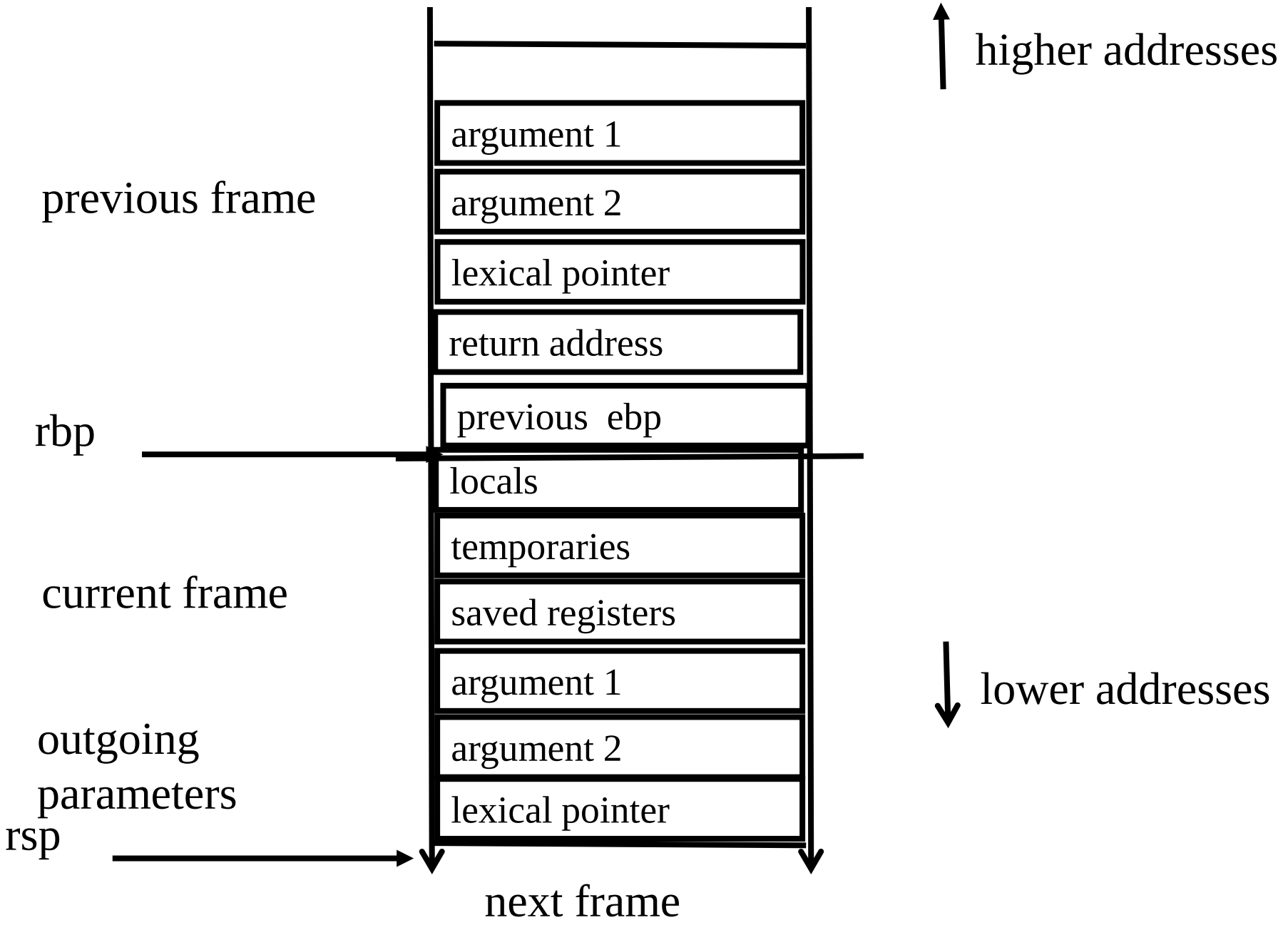
- Allocate a separate space for every procedure incarnation
- Relative addresses
- Provide a simple mean to achieve modularity
- Supports separate code generation of procedures
- Naturally supports recursion
- Efficient memory allocation policy
  - Low overhead
  - Hardware support may be available
- LIFO policy
- Not a pure stack
  - Non local references
  - Updated using arithmetic


Processor
Memory
0x00000000
Code & Data
RIP
Instruction Decoder
Heap
Control
OF
SF
ZF
Flags
ALU
Larger Addresses
rax
rbx
rcx
rdx
rsi
rdi
Rpb
Rsb
r08
r09
r10
r11
r12
r13
r14
r15
Stack
Registers
0xffffffff

A Typical Stack Frame



Pascal 80386 Frame



Compiling a simple example

```
#include <stdio.h>
main() {
   printf("factorial(2)=%d", factorial(2));
}
```

```
.LC0:
        .string "factorial(2)=%d"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, 2
        call    factorial(int)
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

Compiling factorial

```
int factorial(int num) {
   if (num == 1) return 1 ;
   else return num * factorial(num -1 );
}
```

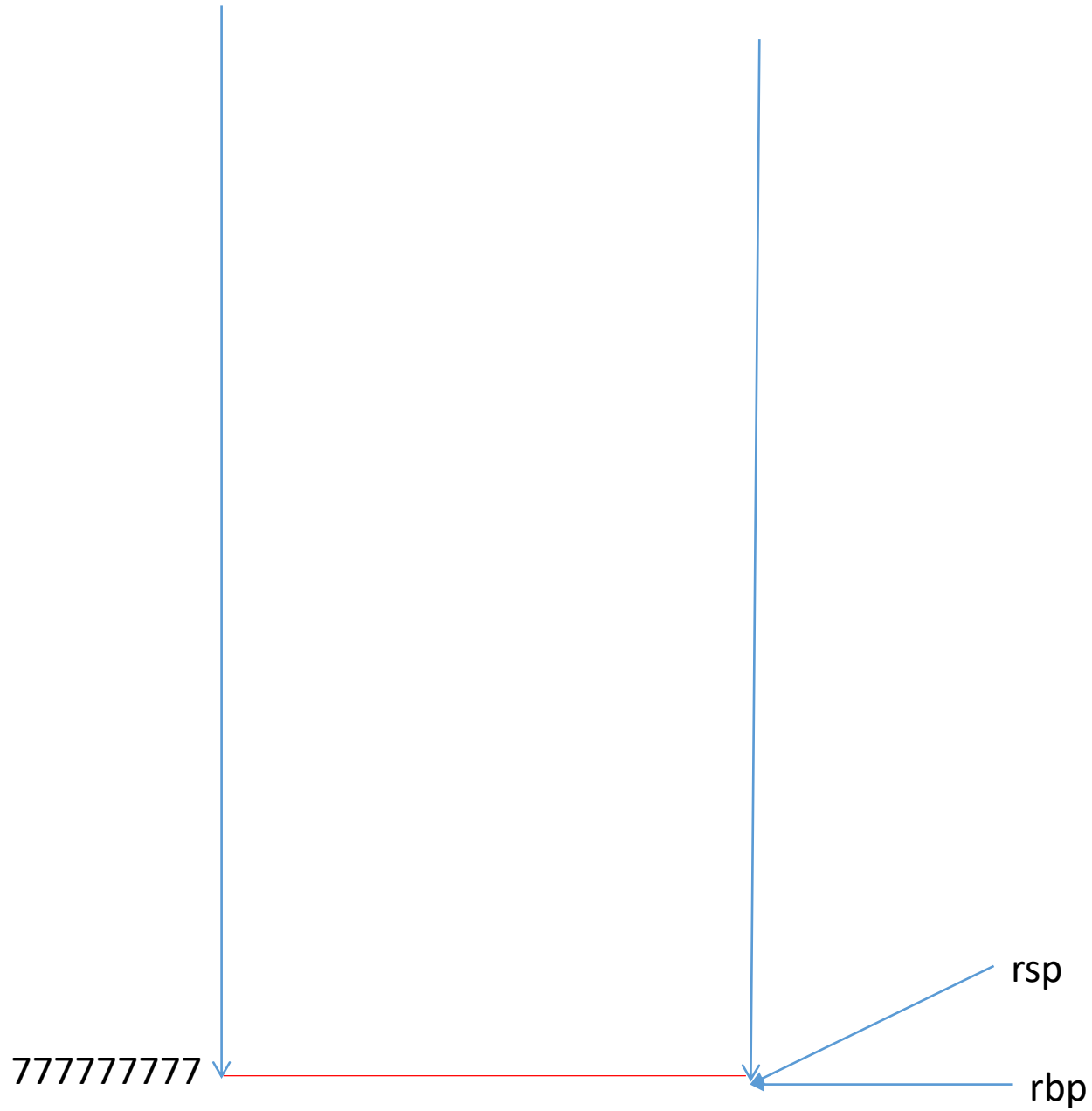
```
. factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
        imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
```

Code/Data

```
factorial(int):
        …

.LC0:
        .string "factorial(2)=%d"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi,2
        call    factorial(int)
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

Stack

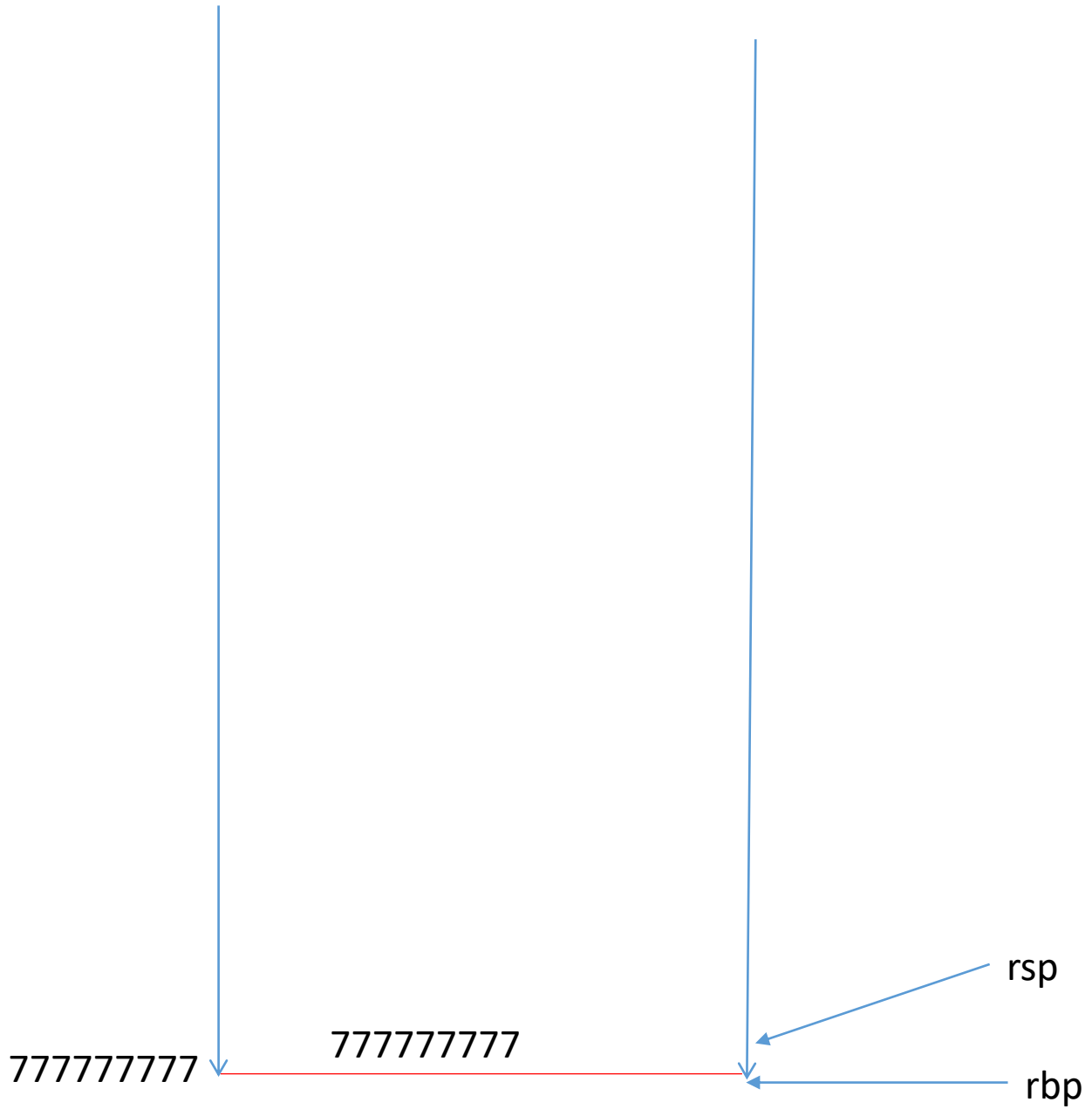


Code/Data

```
factorial(int):
        …

.LC0:
        .string "factorial(2)=%d"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, 2
        call    factorial(int)
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

Stack

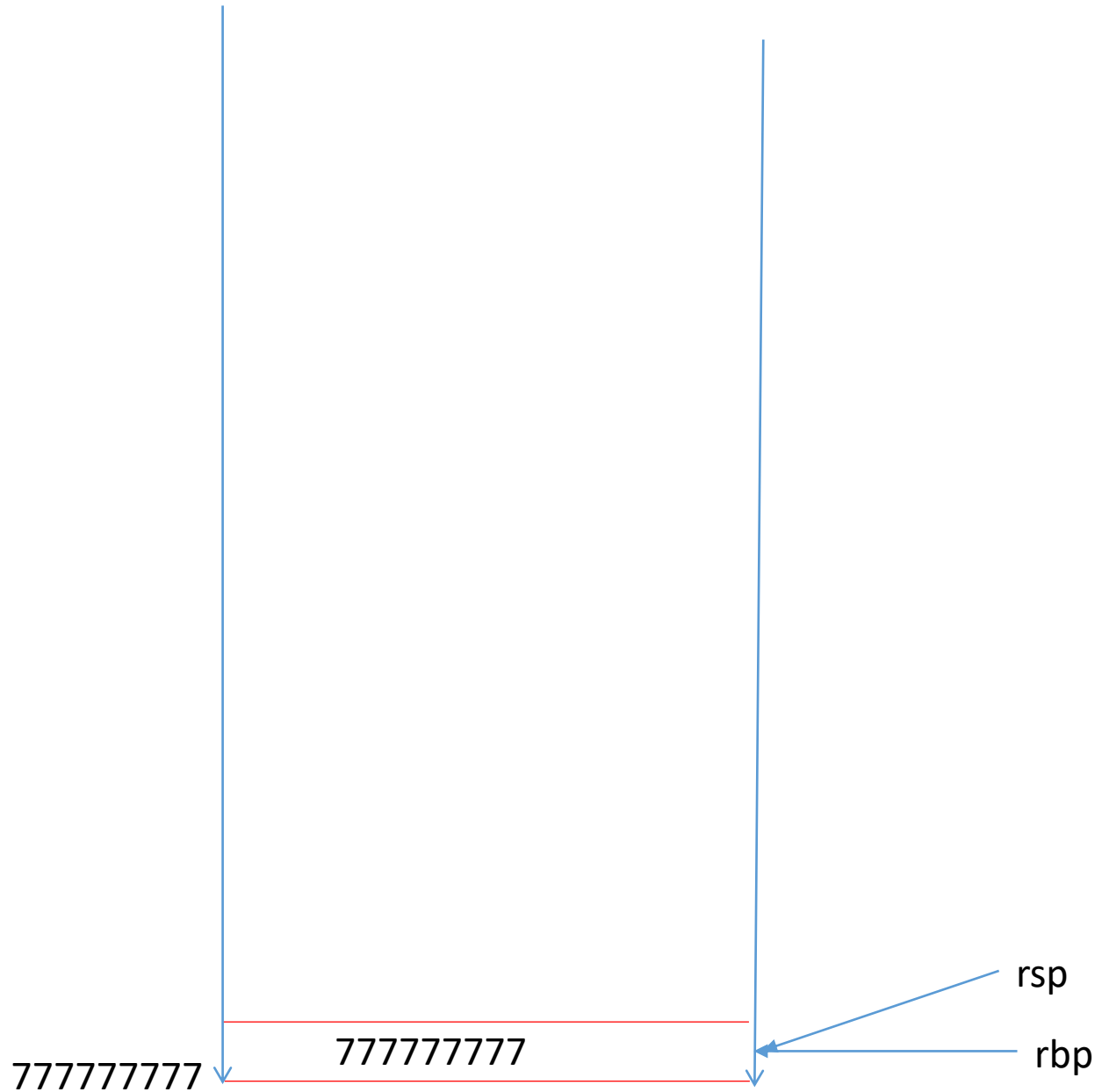


Code/Data

```
factorial(int):
        …

.LC0:
        .string "factorial(2)=%d"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, 2
        call    factorial(int)
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

Stack

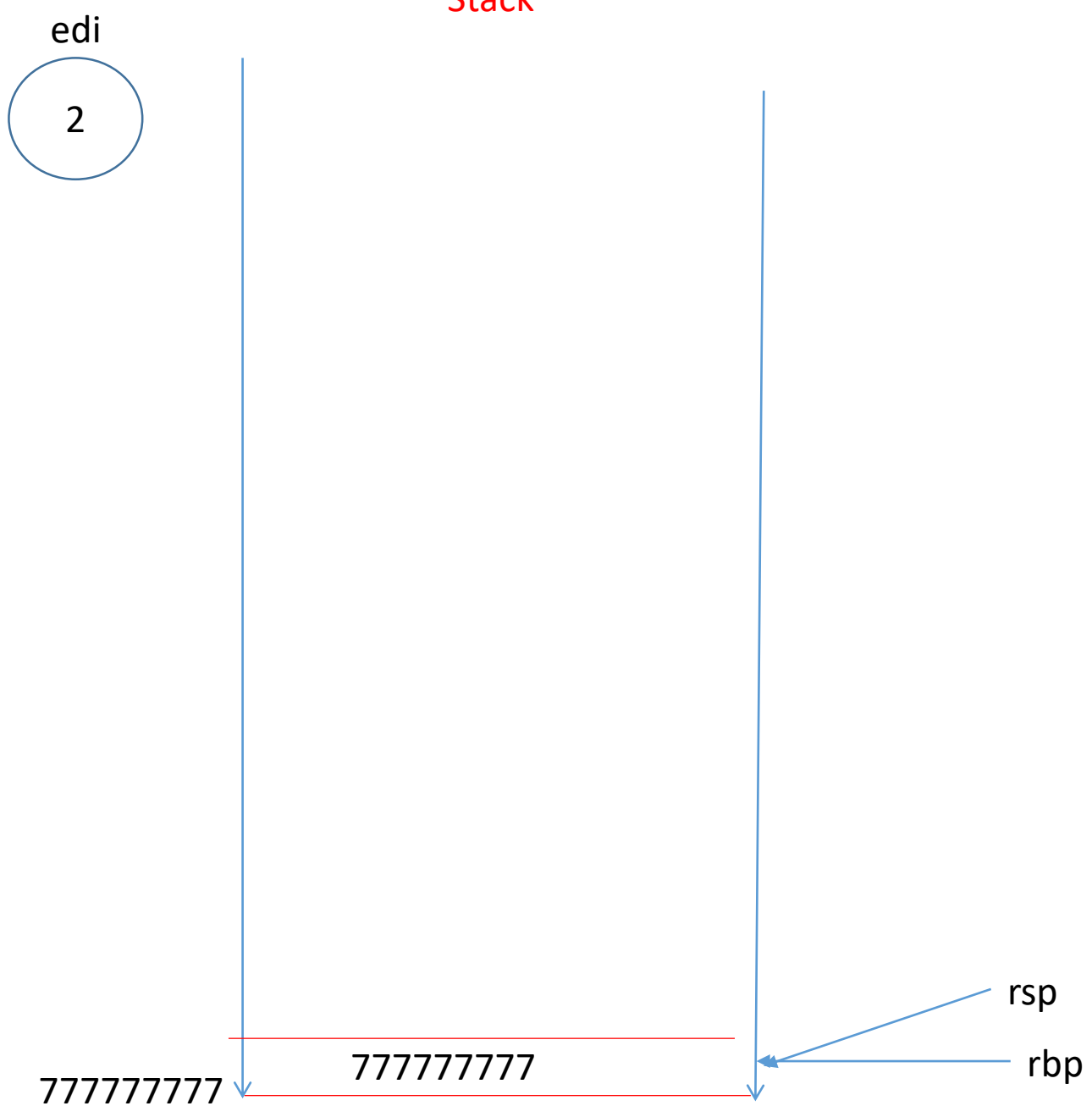


Code/Data

```
factorial(int):
        …

.LC0:
        .string "factorial(2)=%d"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, 2
        call    factorial(int)
L4:     mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

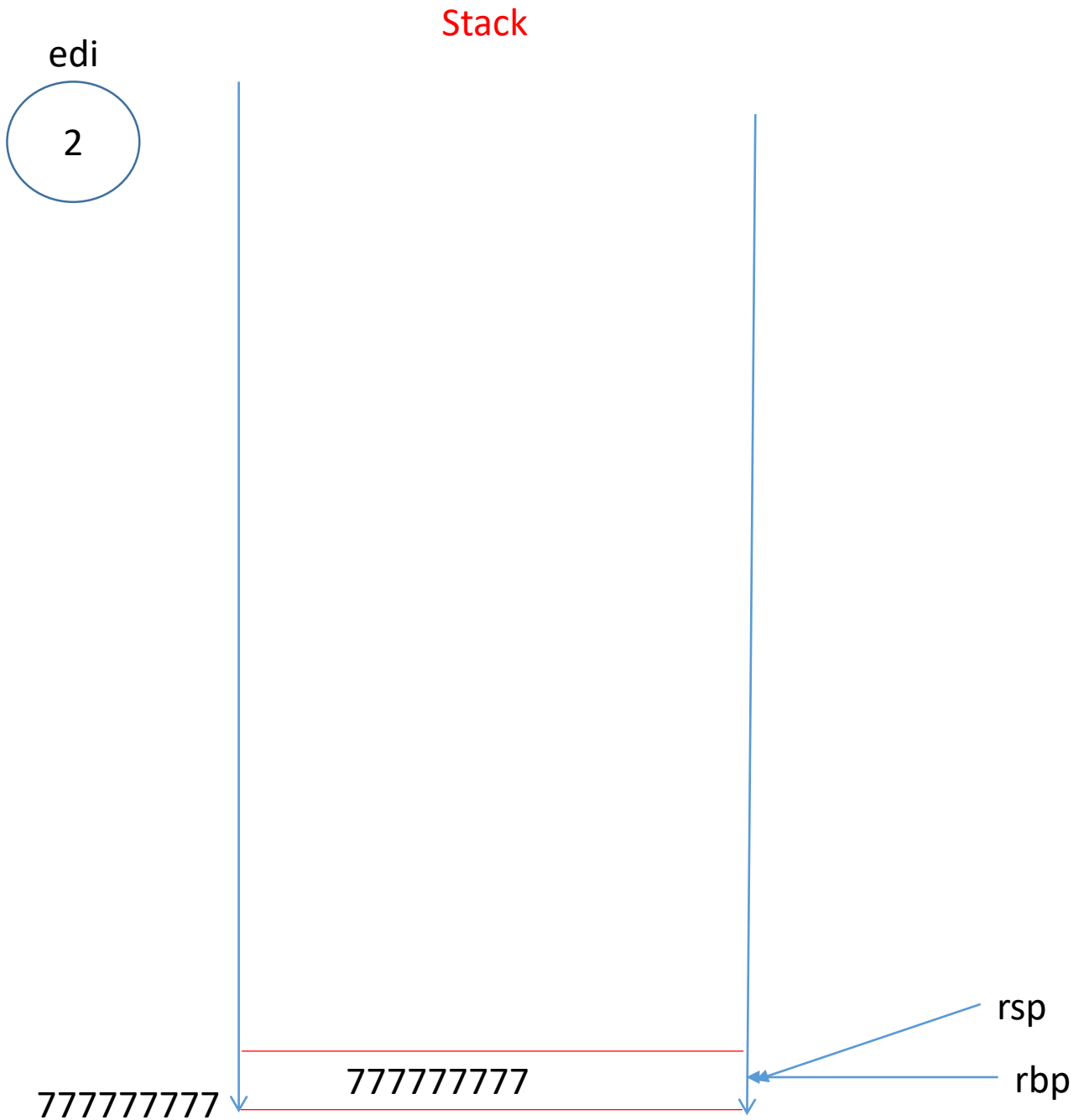
Stack



Code/Data

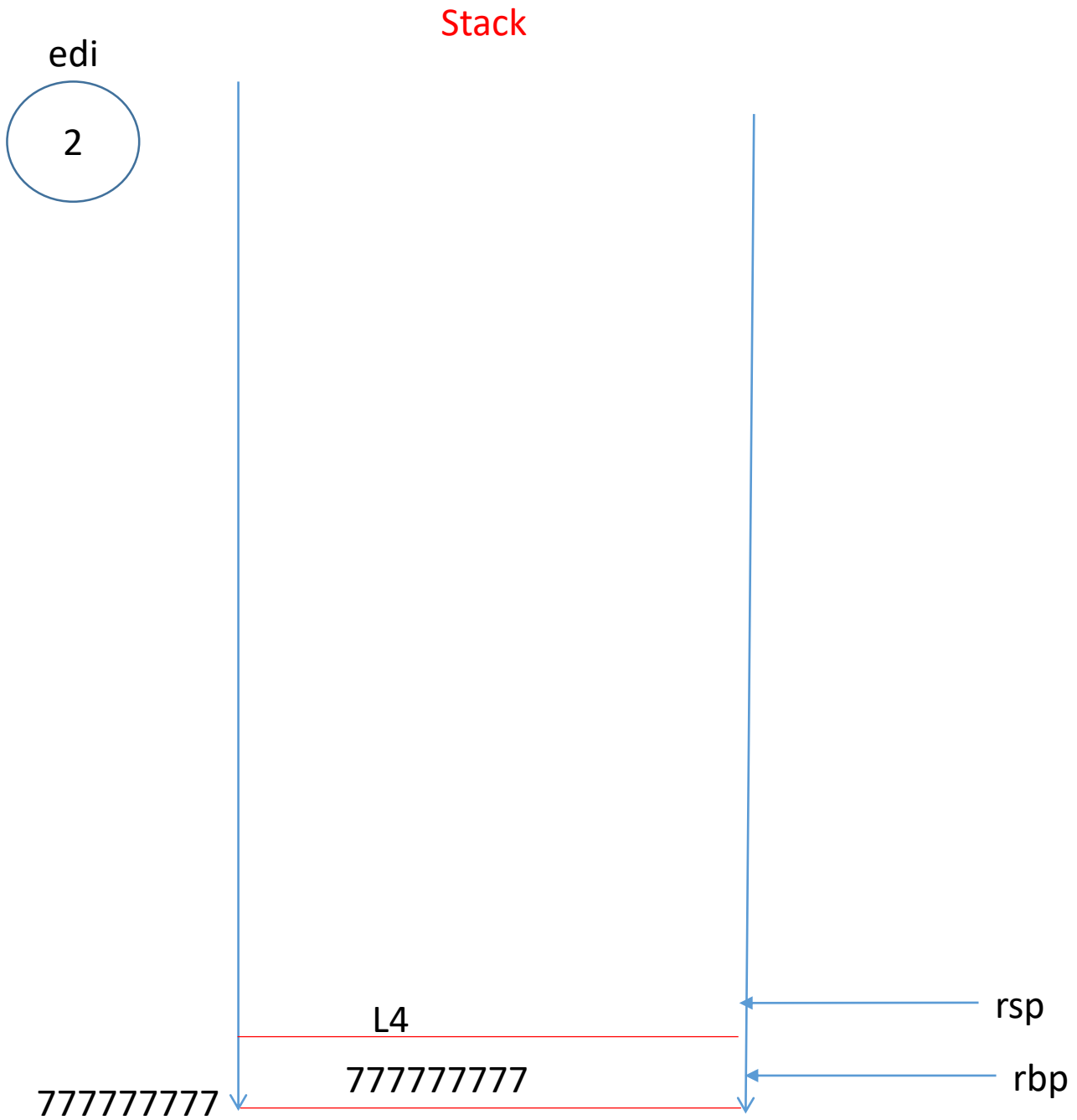
```
factorial(int):
        …

.LC0:
        .string "factorial(2)=%d"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, 2
        call    factorial(int)
L2:     mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```



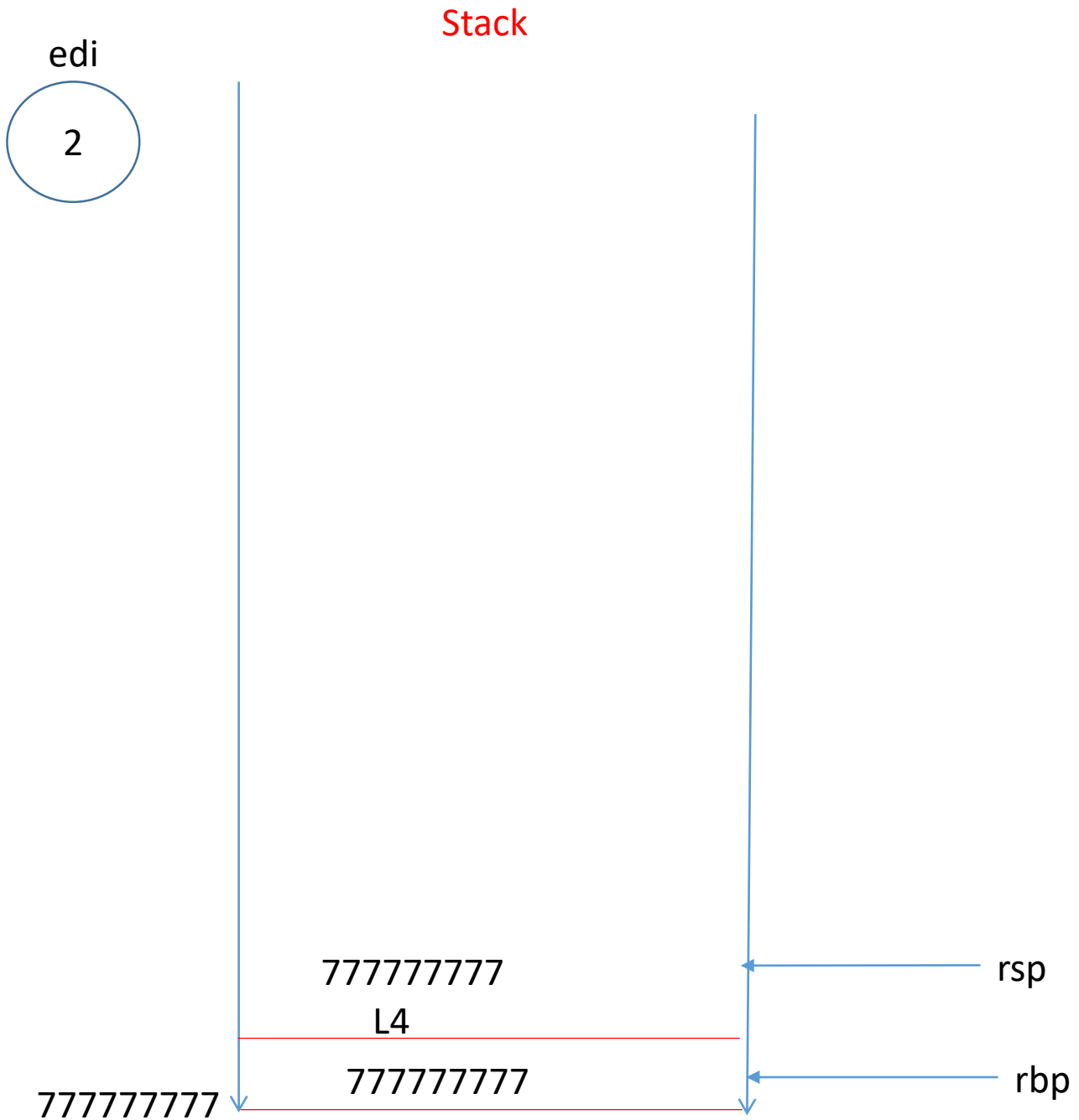
Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
        imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```



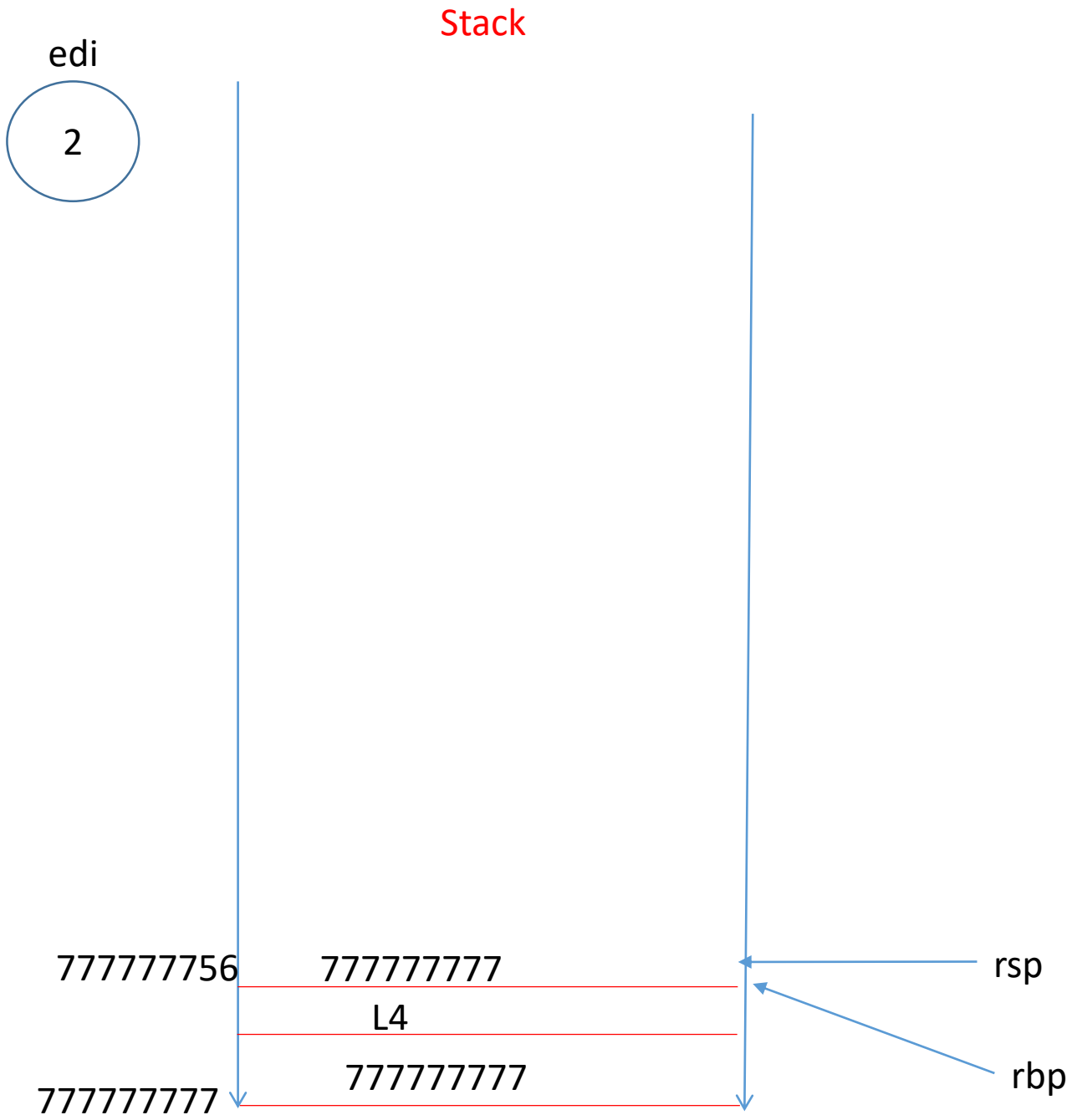
Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
        imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```



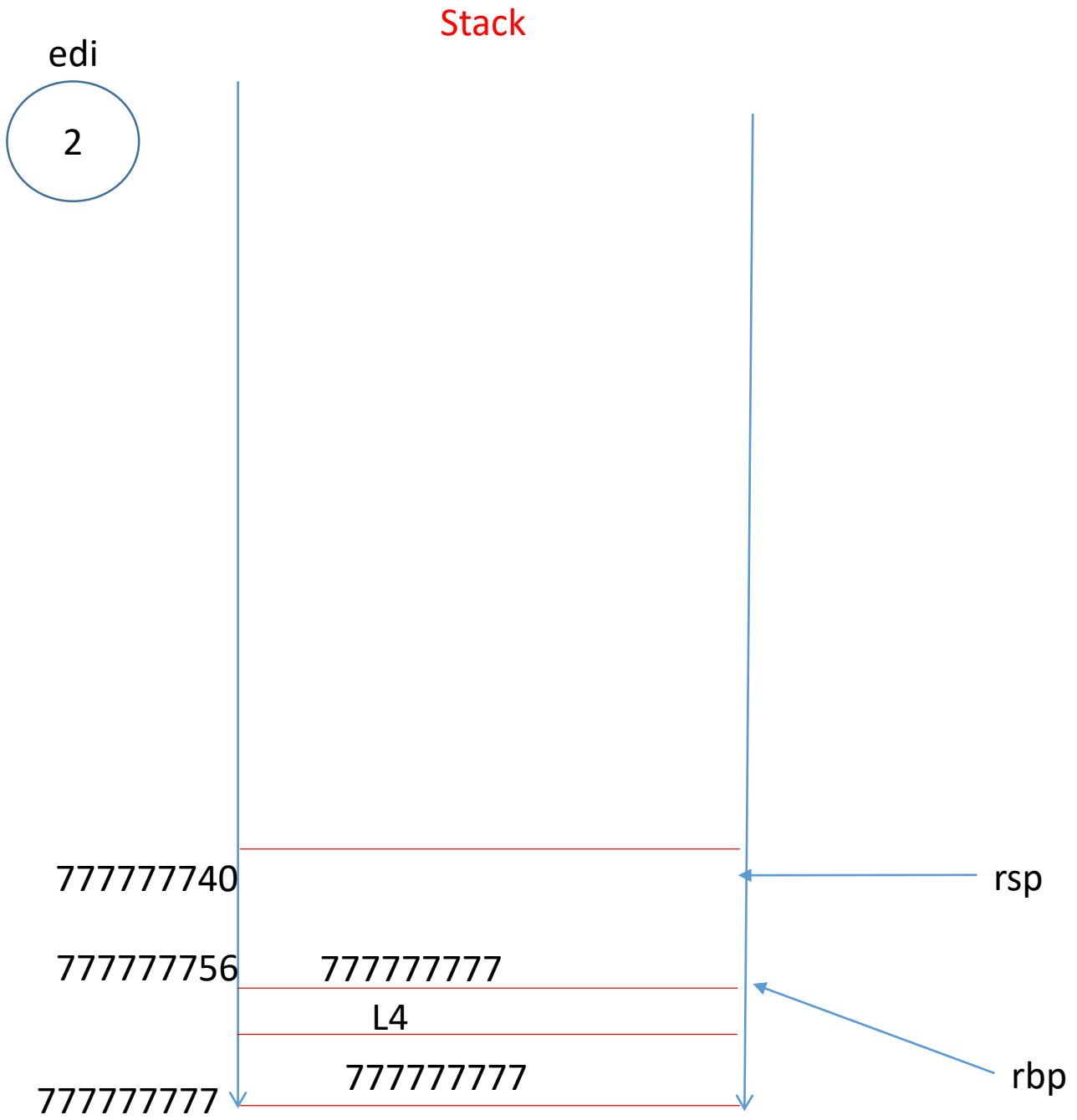
Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
        imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```



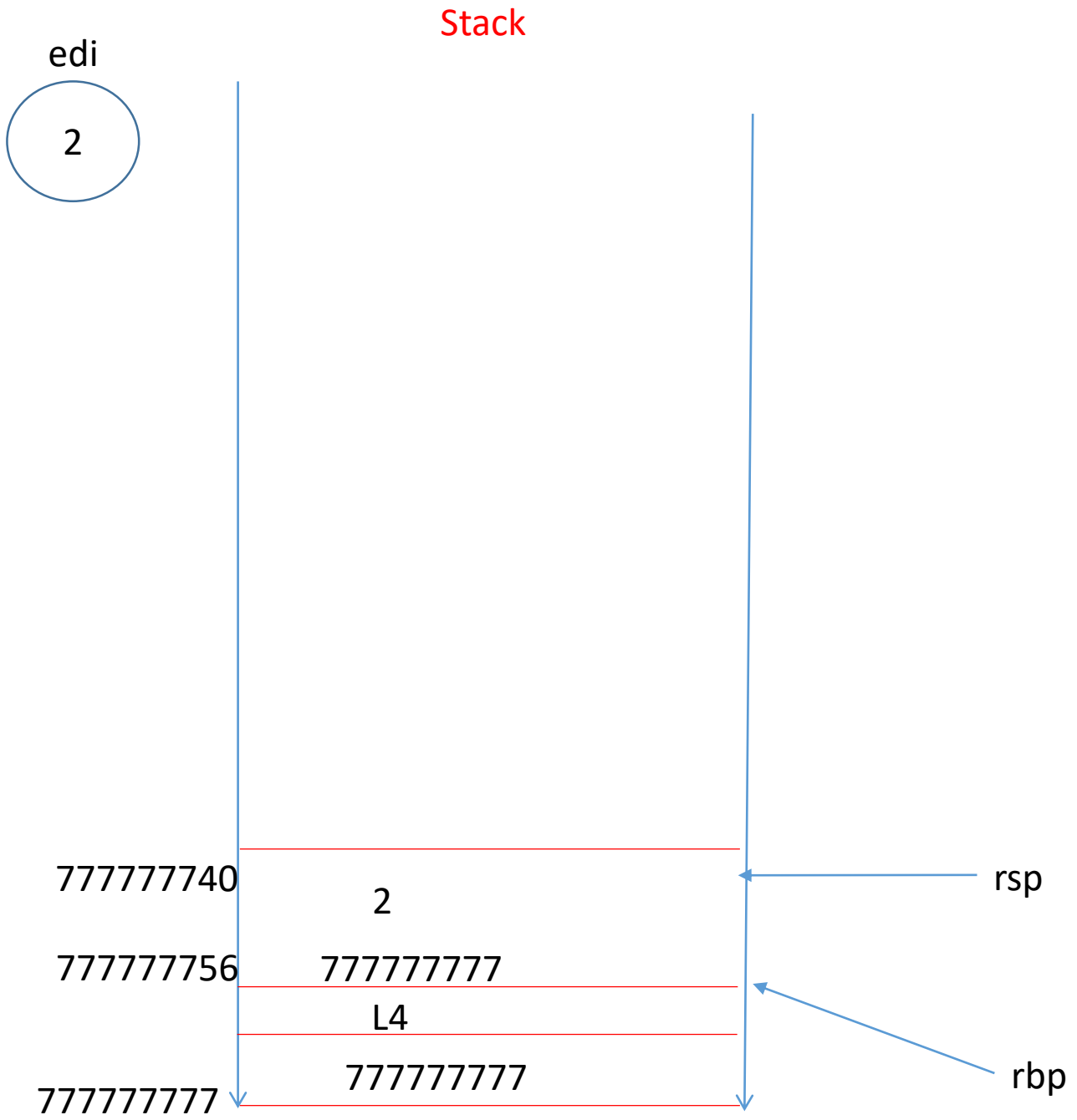
Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
        imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```



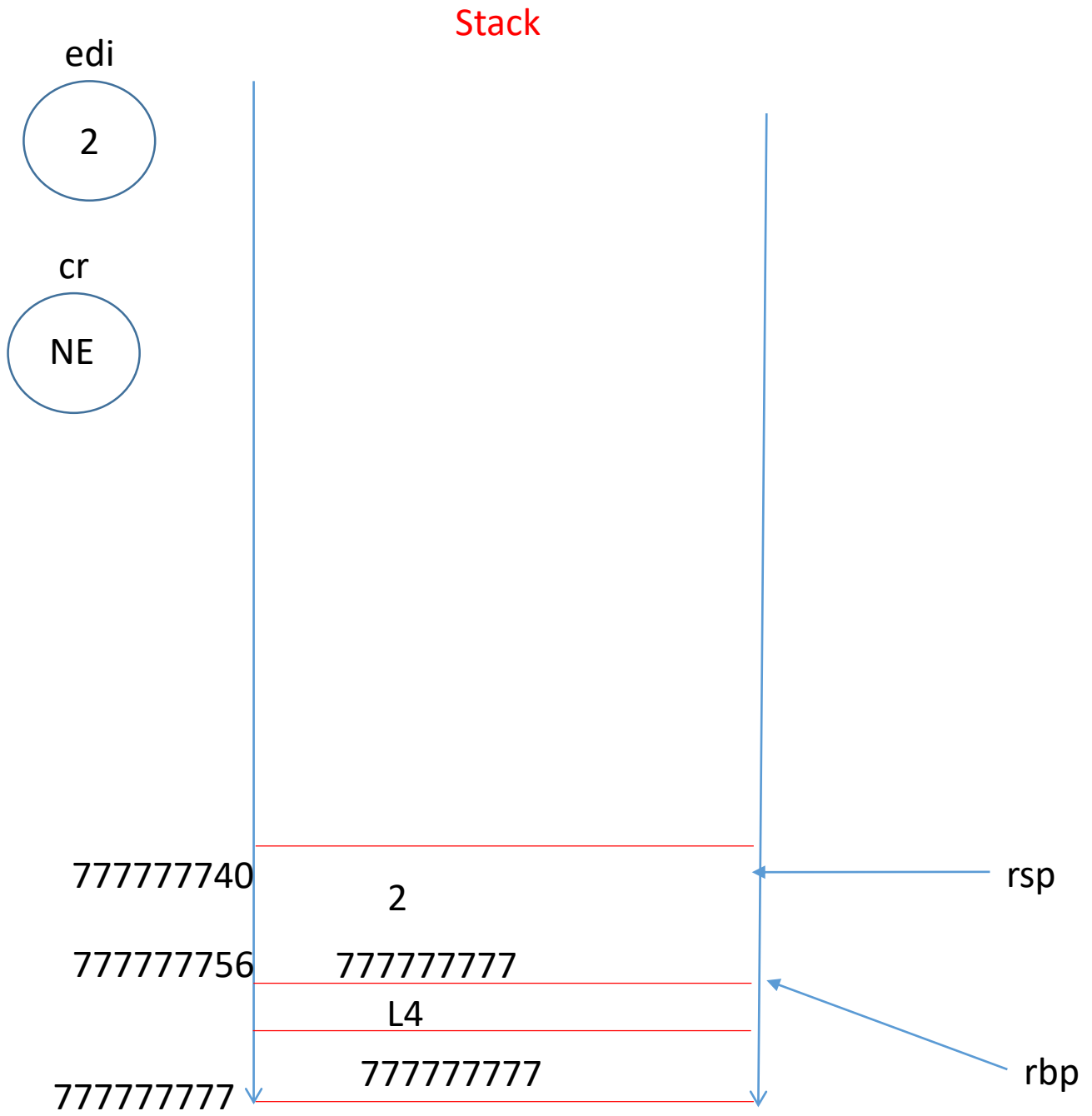
Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
        imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```



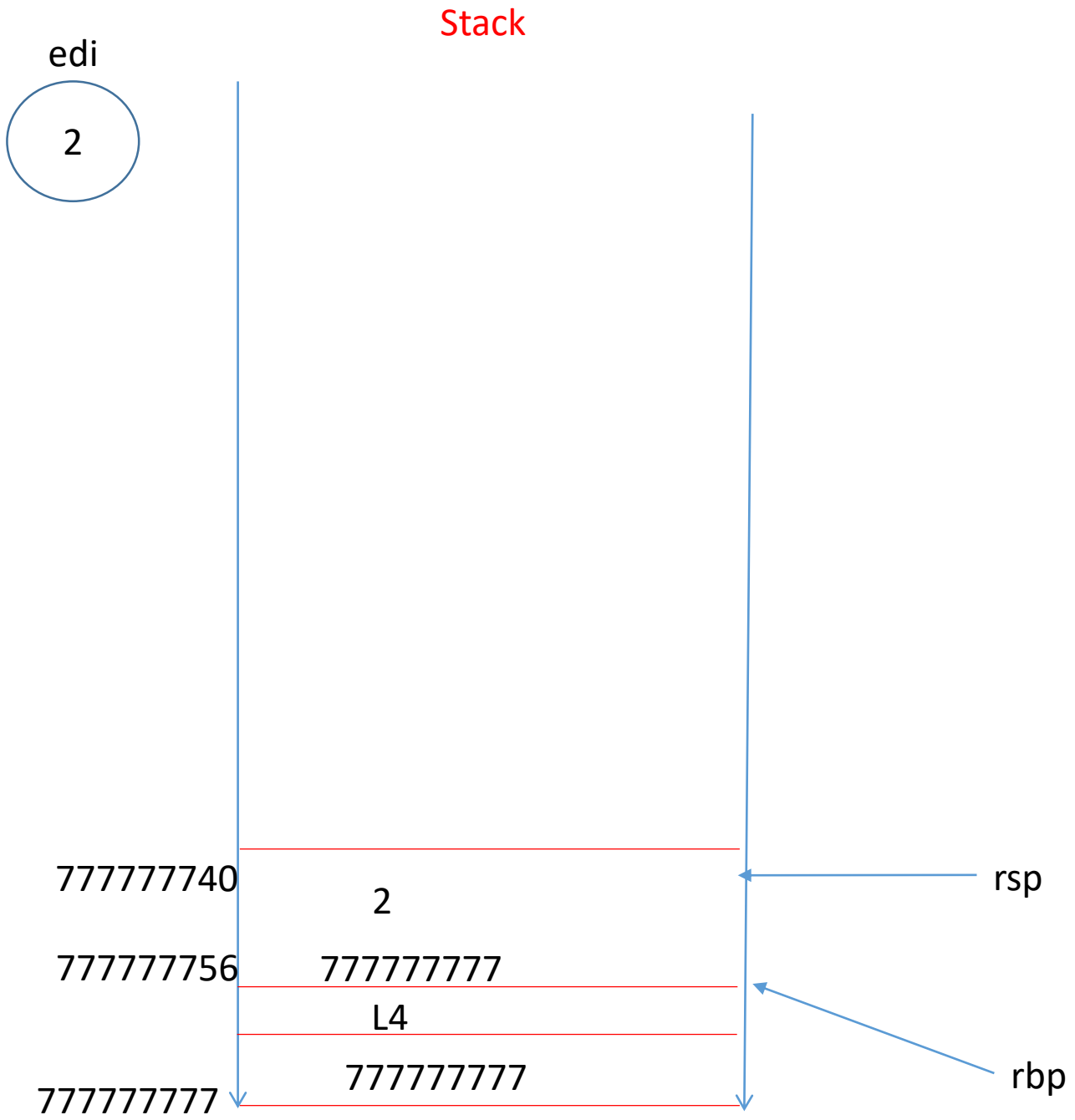
Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
        imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```



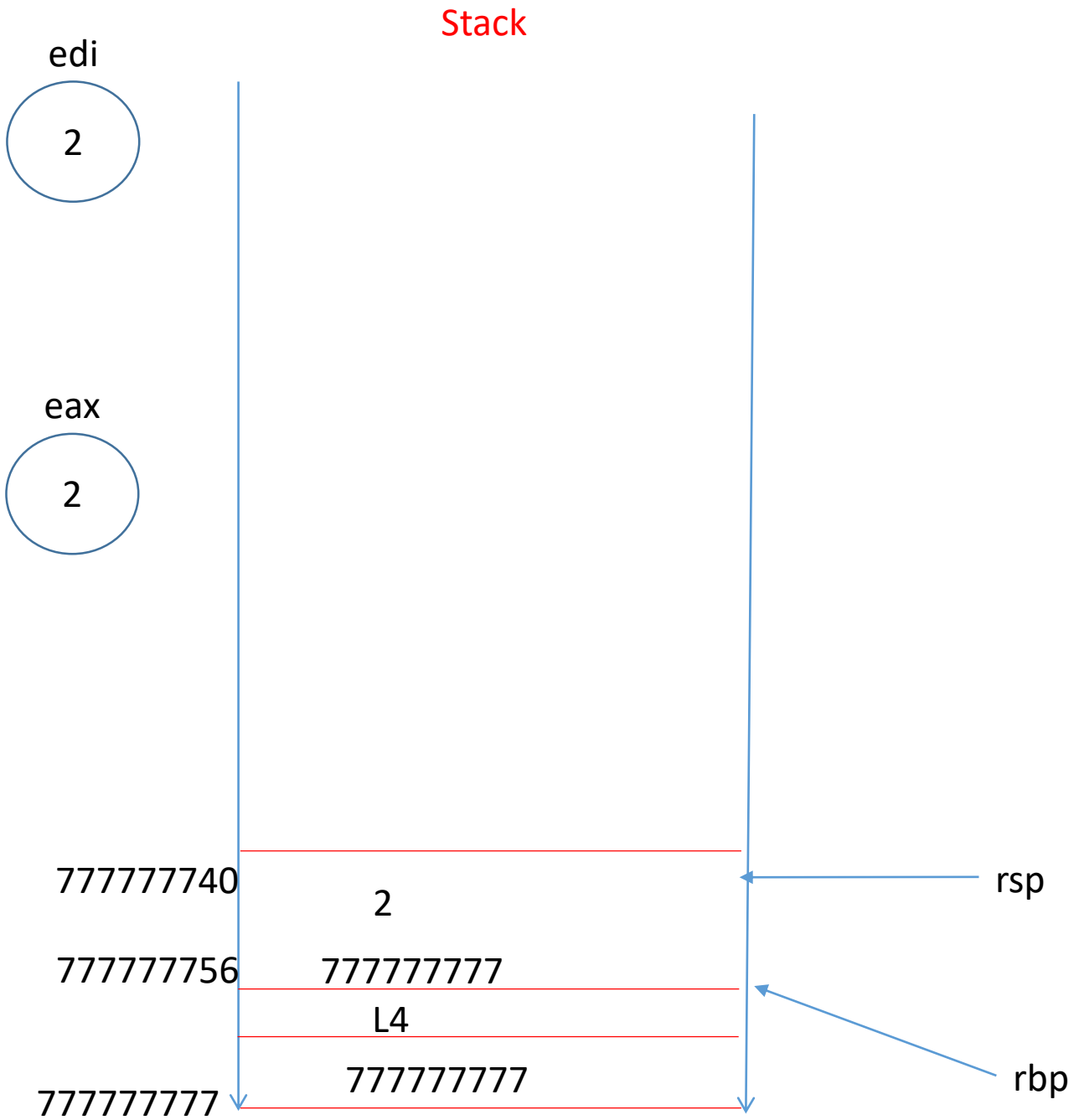
Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
        imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```



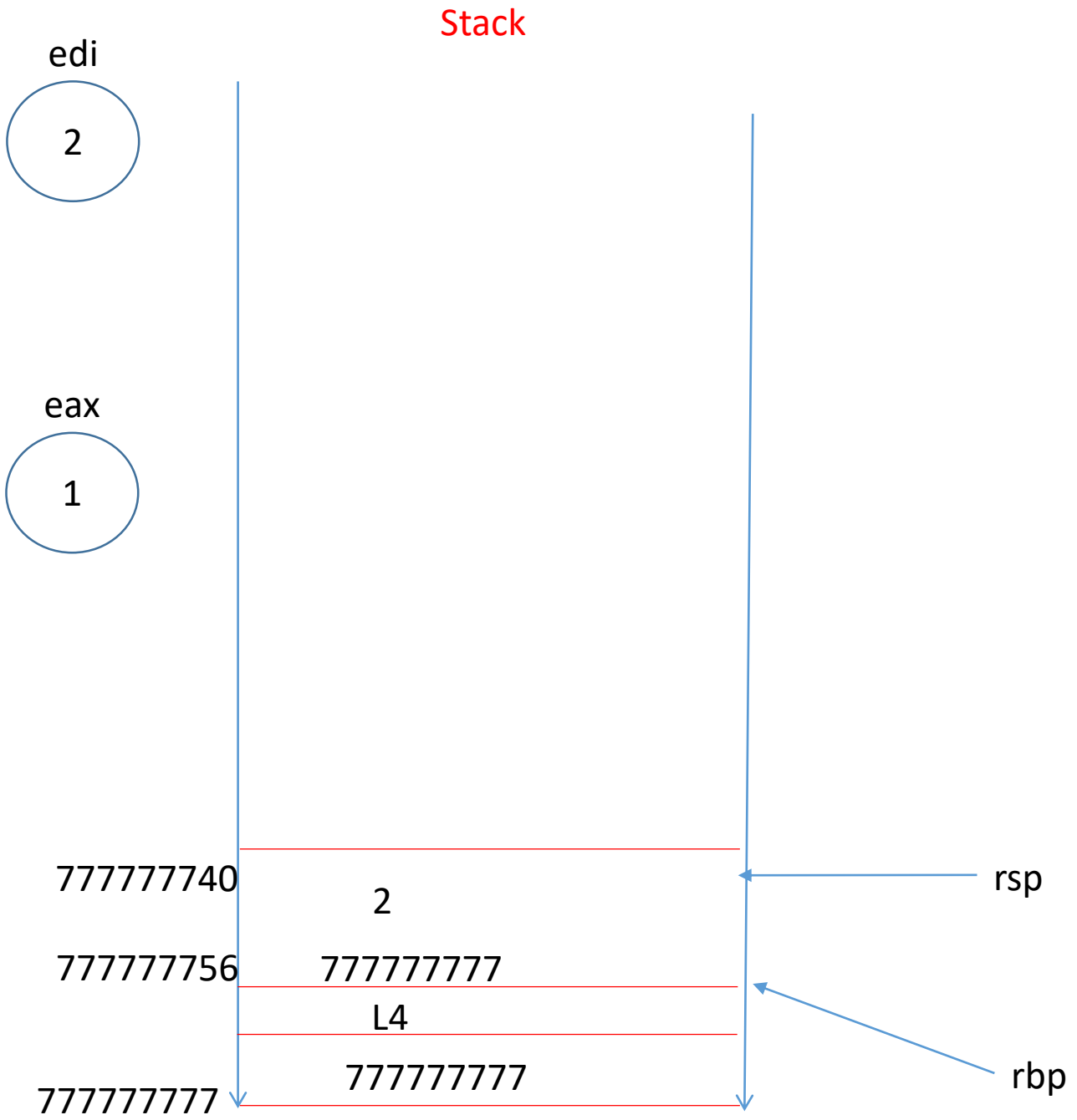
Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
        imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```



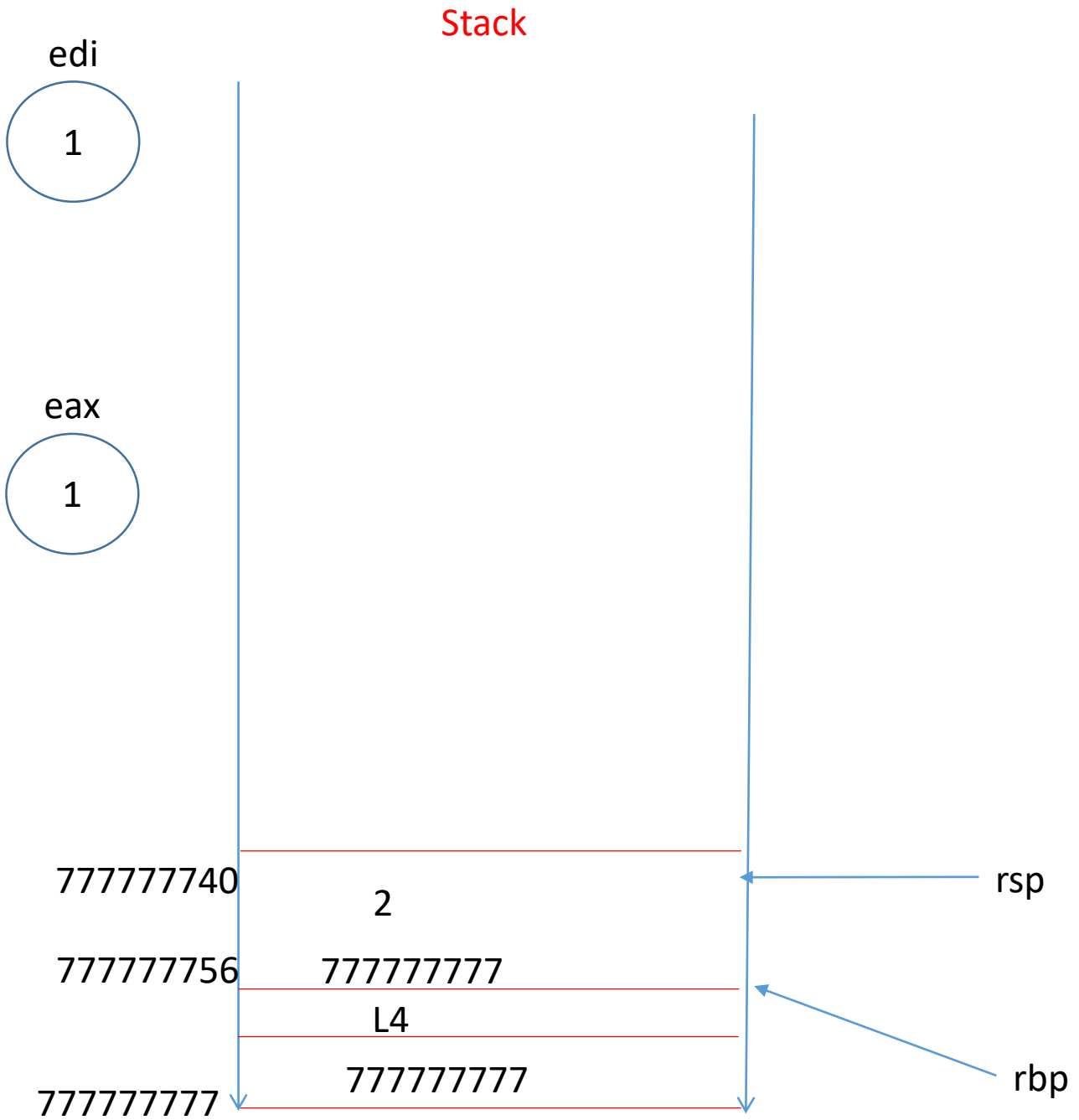
Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
        imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```



Code/Data

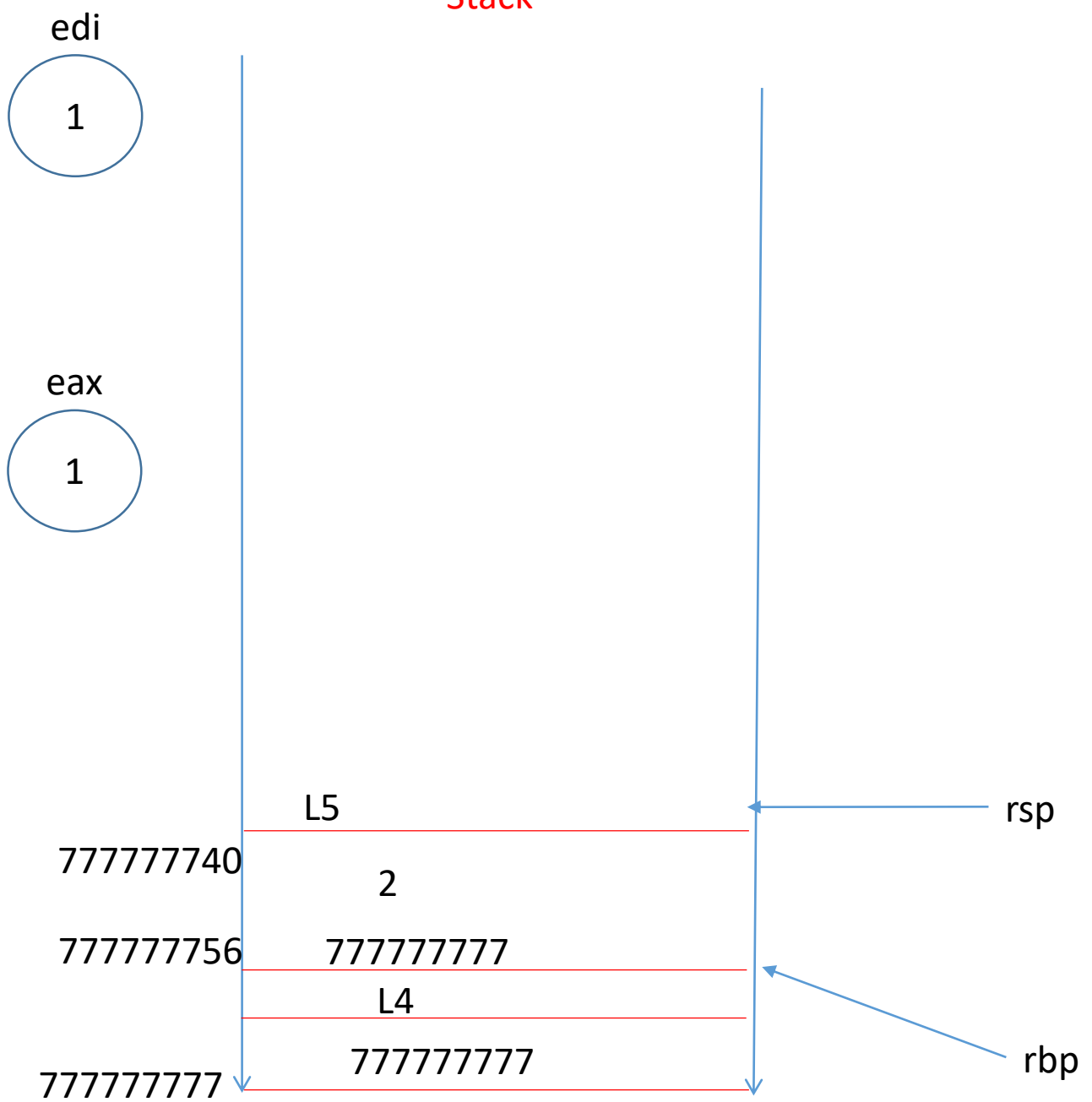
```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
        imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```



Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

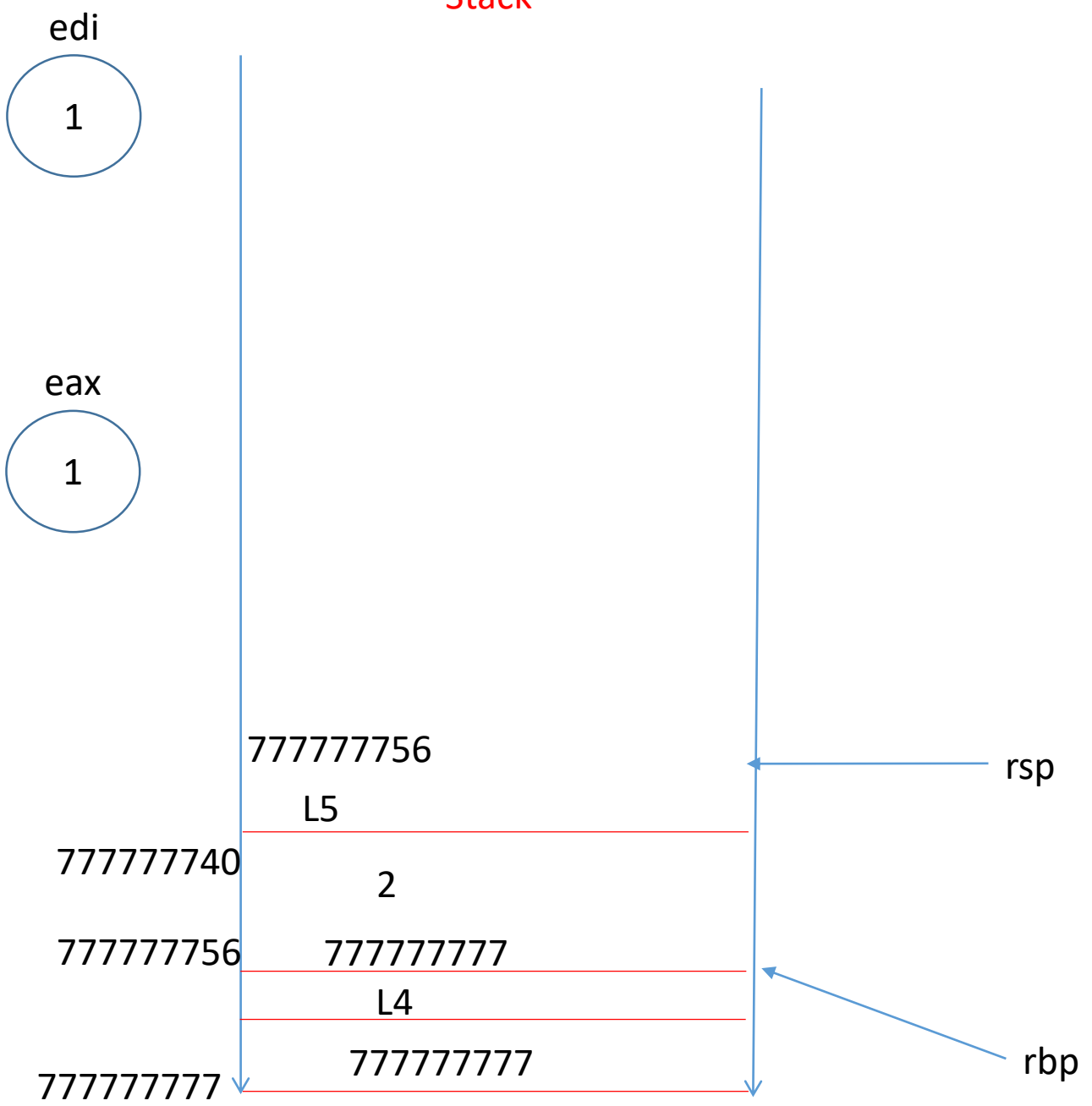
Stack



Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

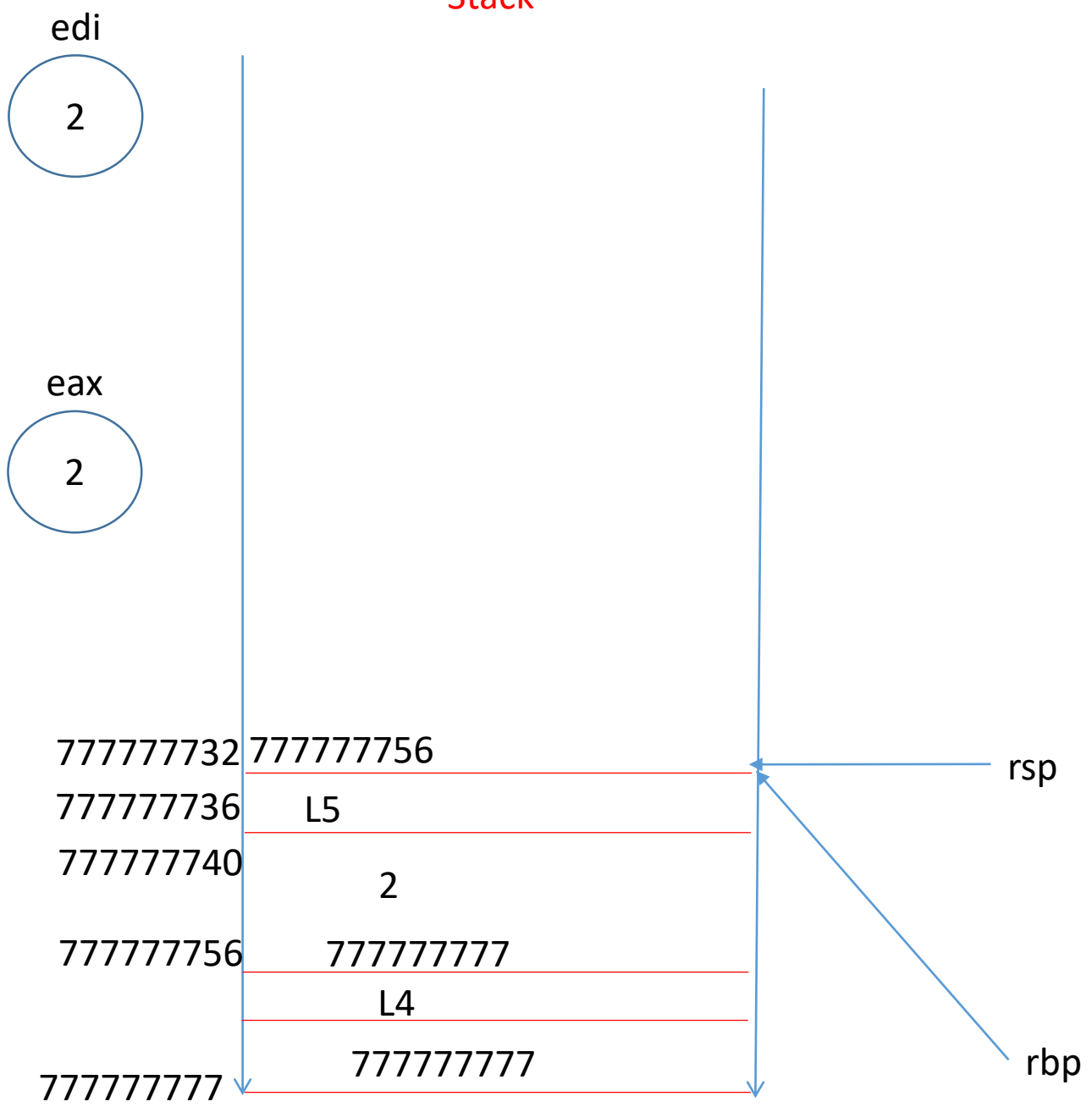
Stack



Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

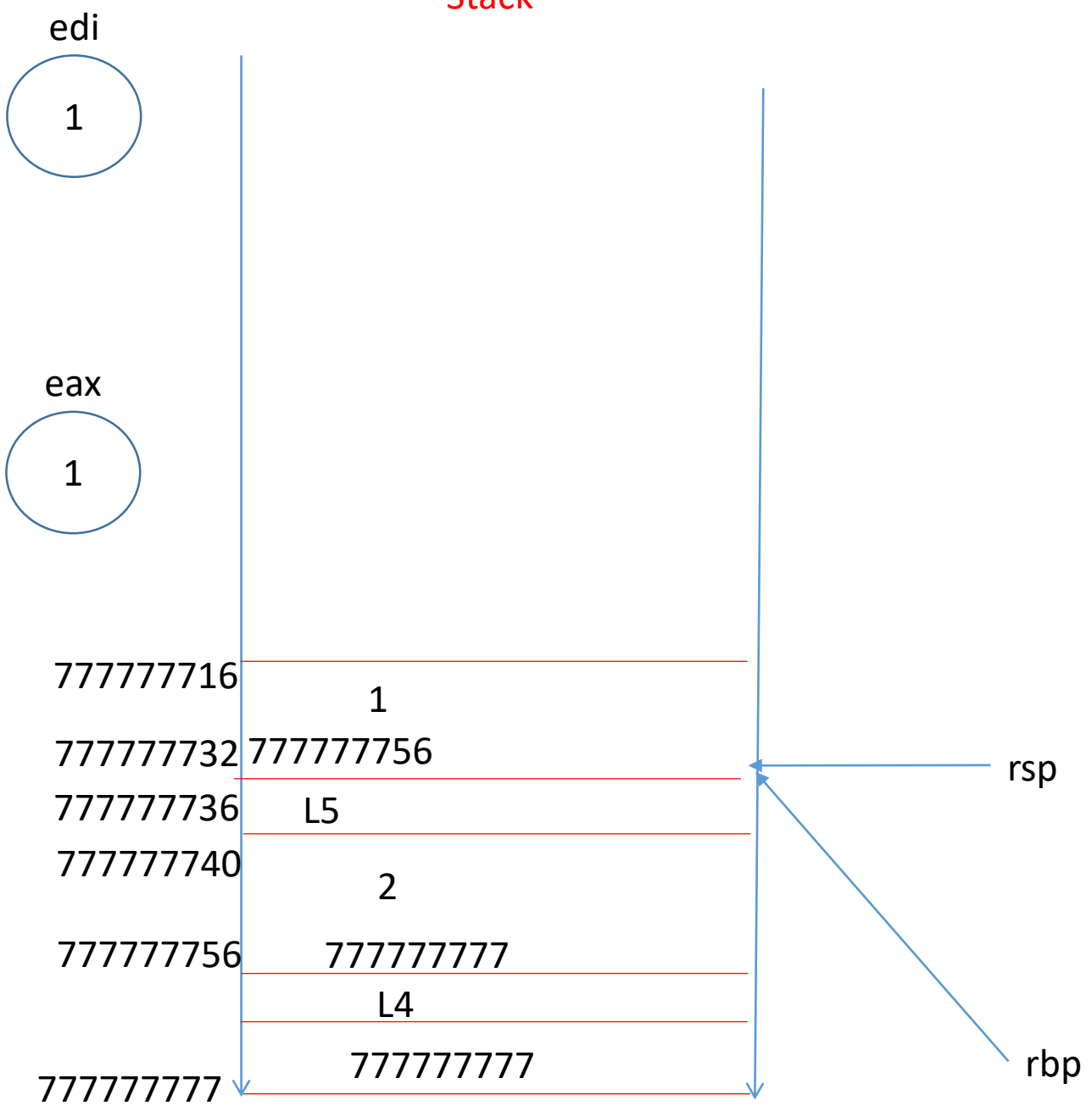
Stack



Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

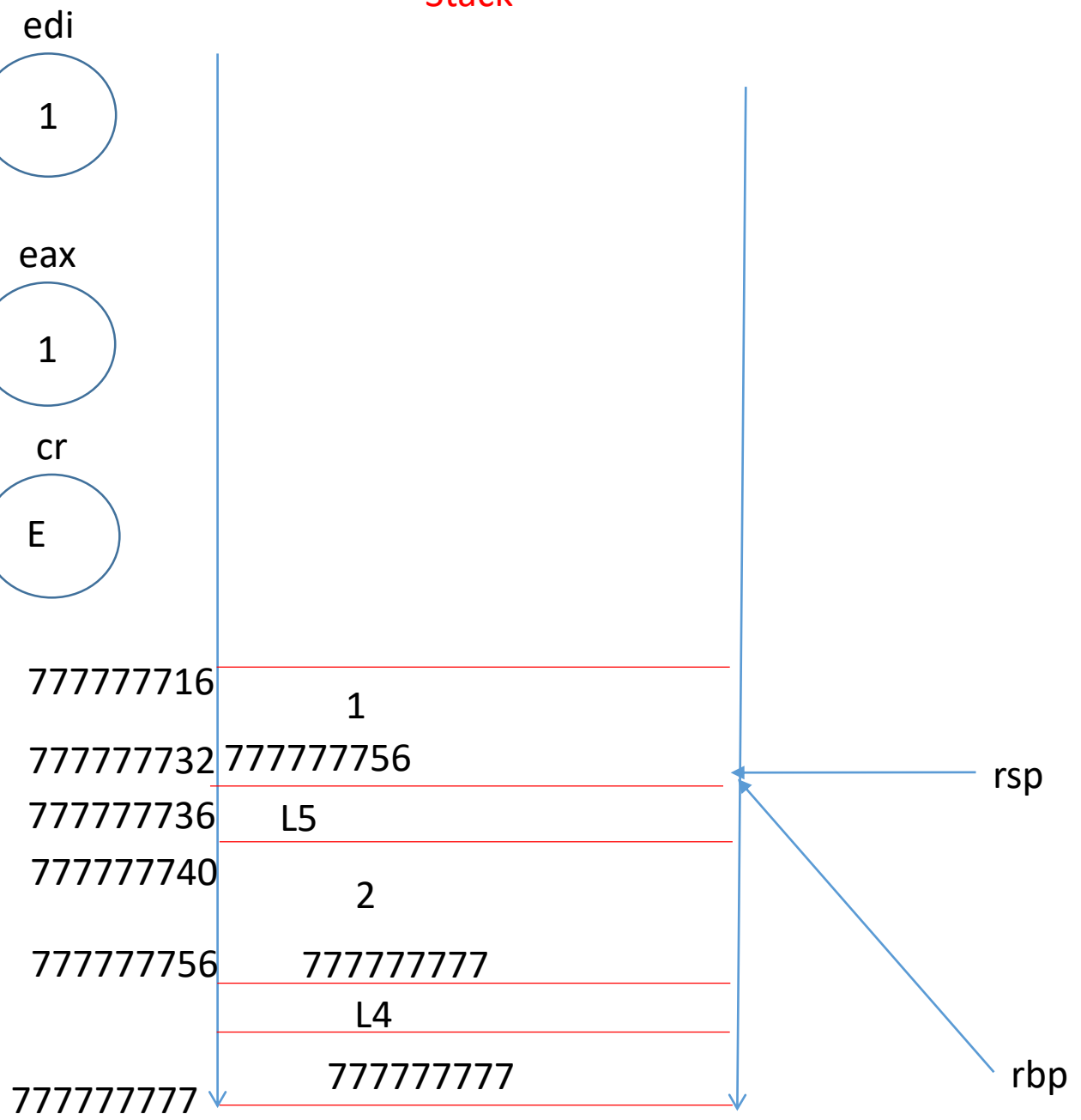
Stack



Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

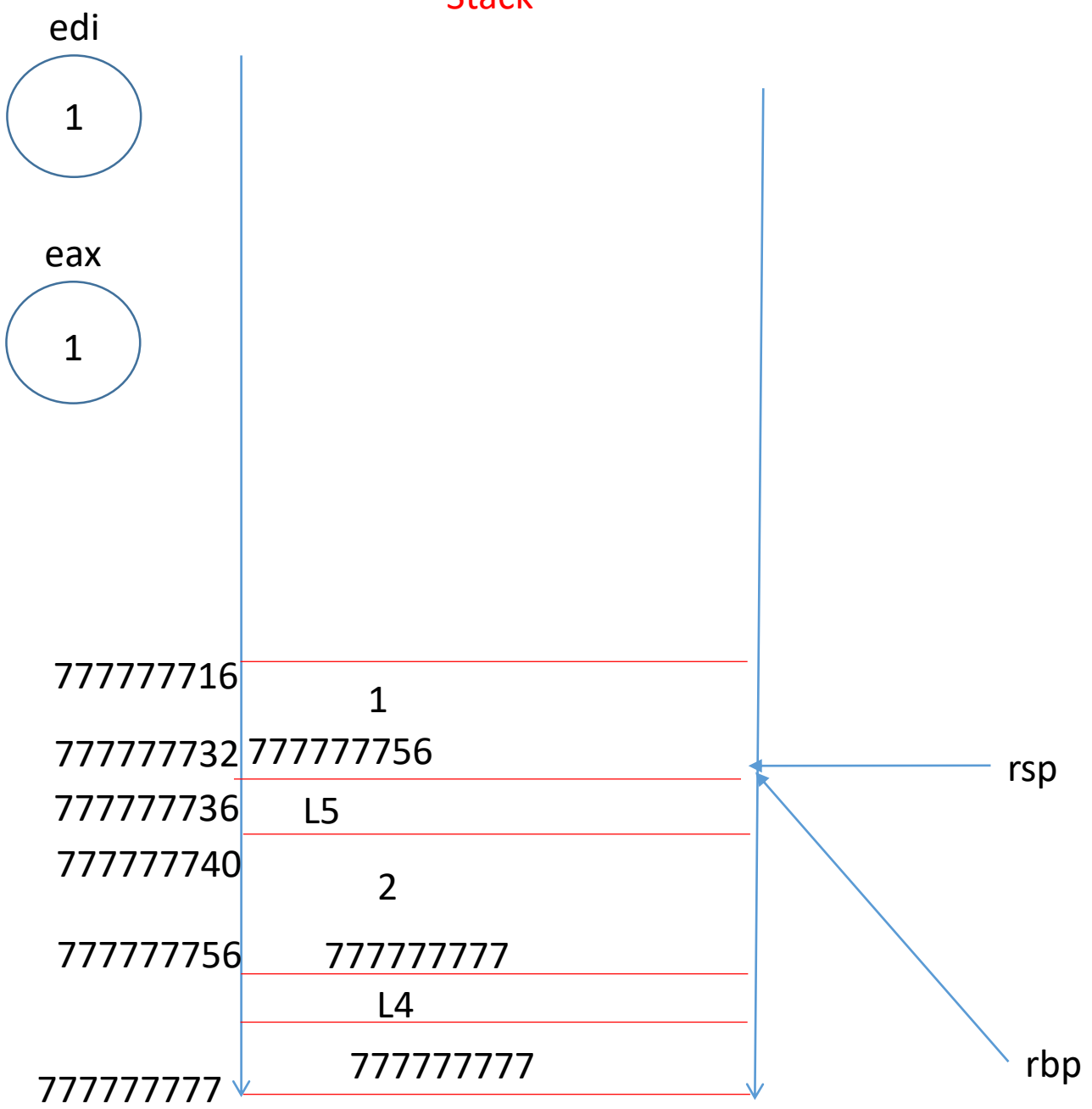
Stack



Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

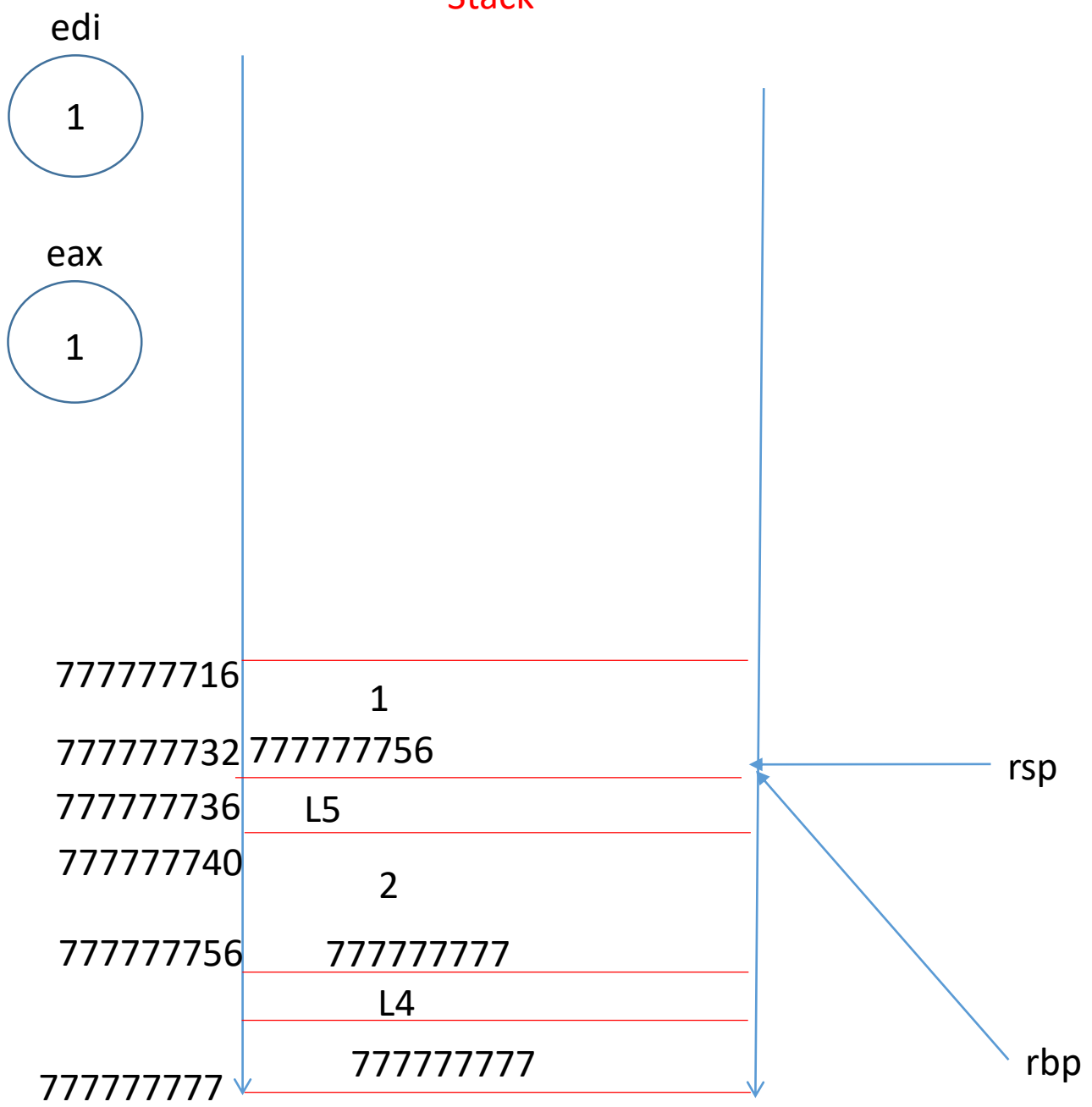
Stack



Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

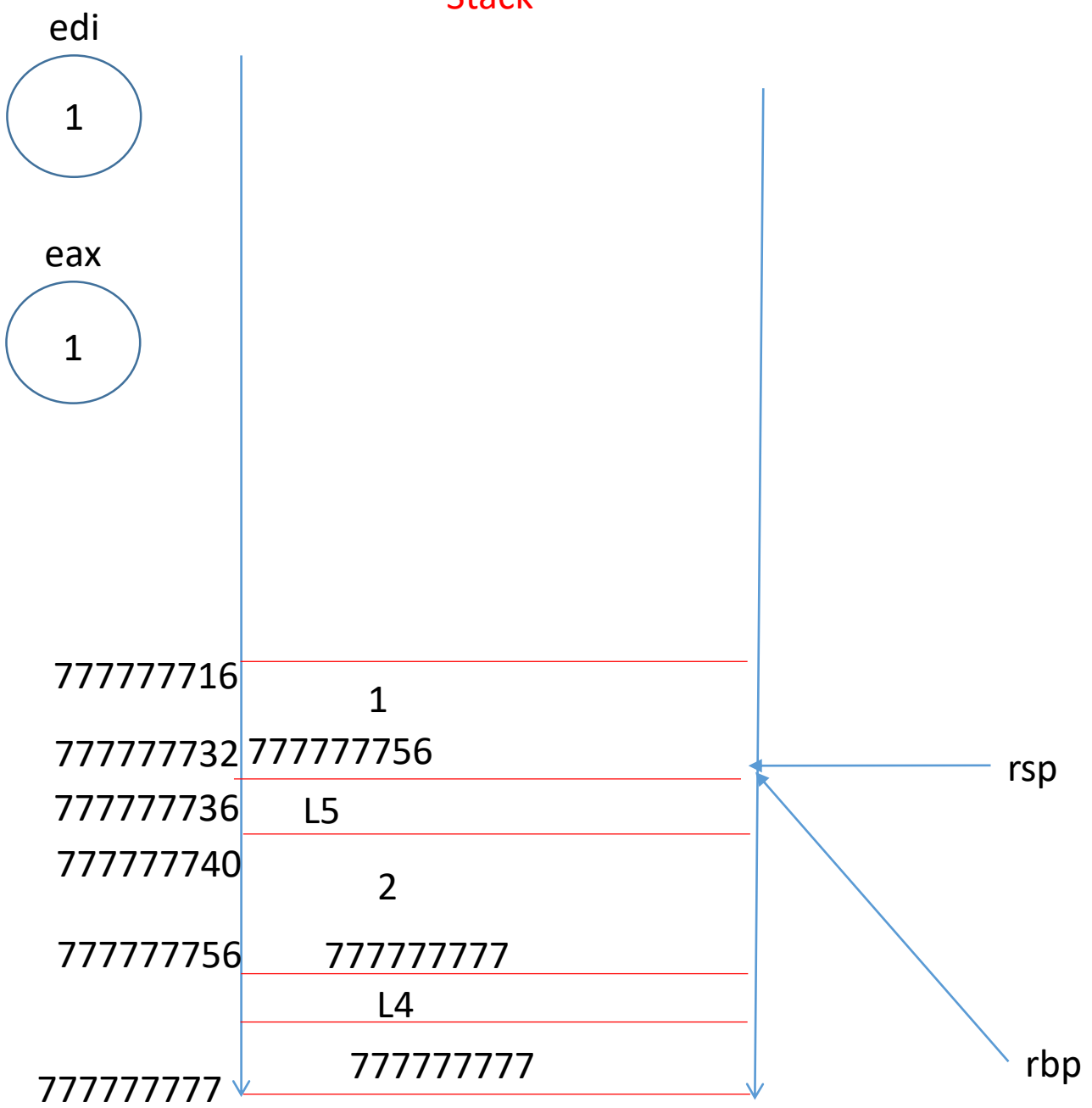
Stack



Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

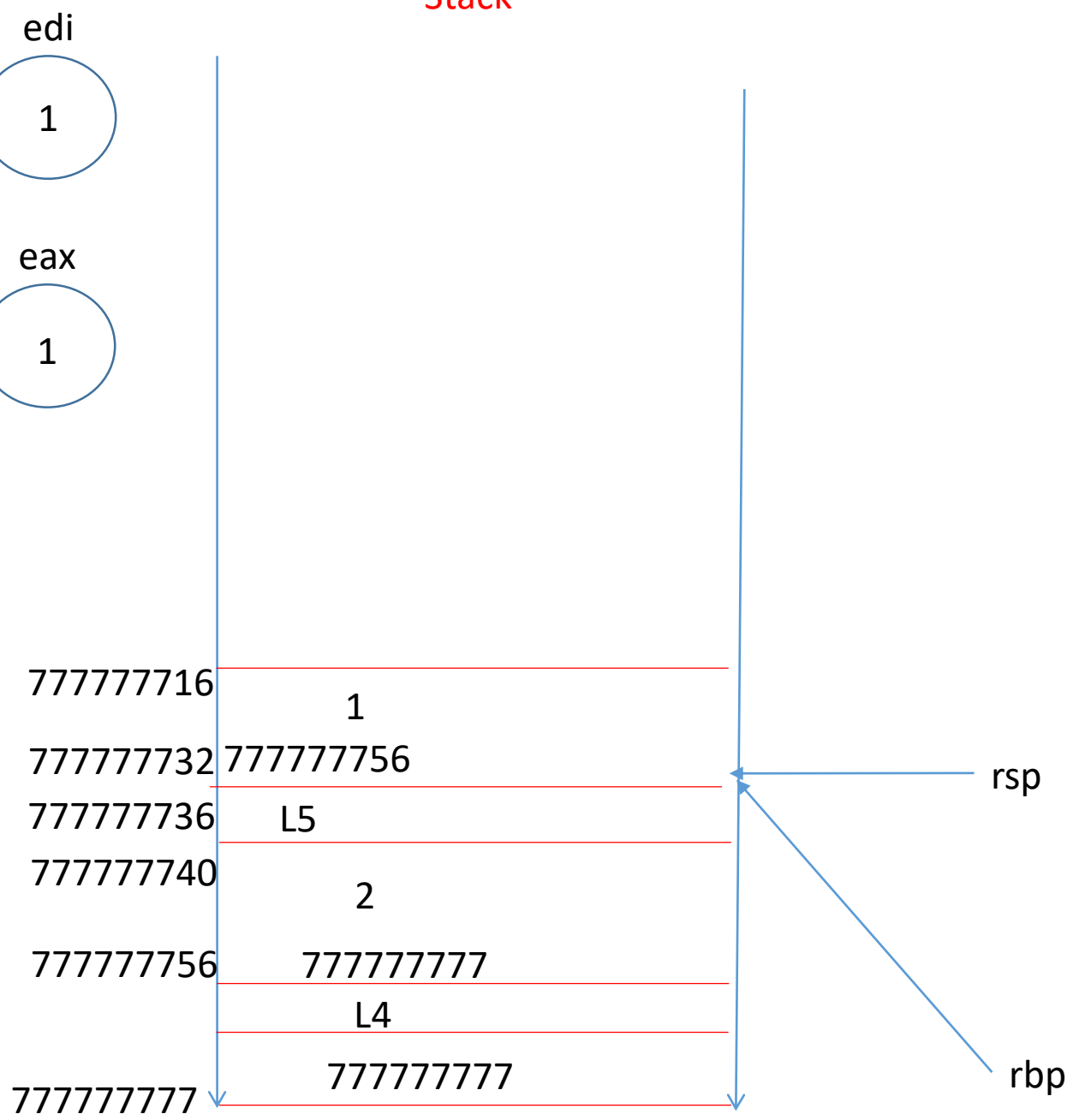
Stack



Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

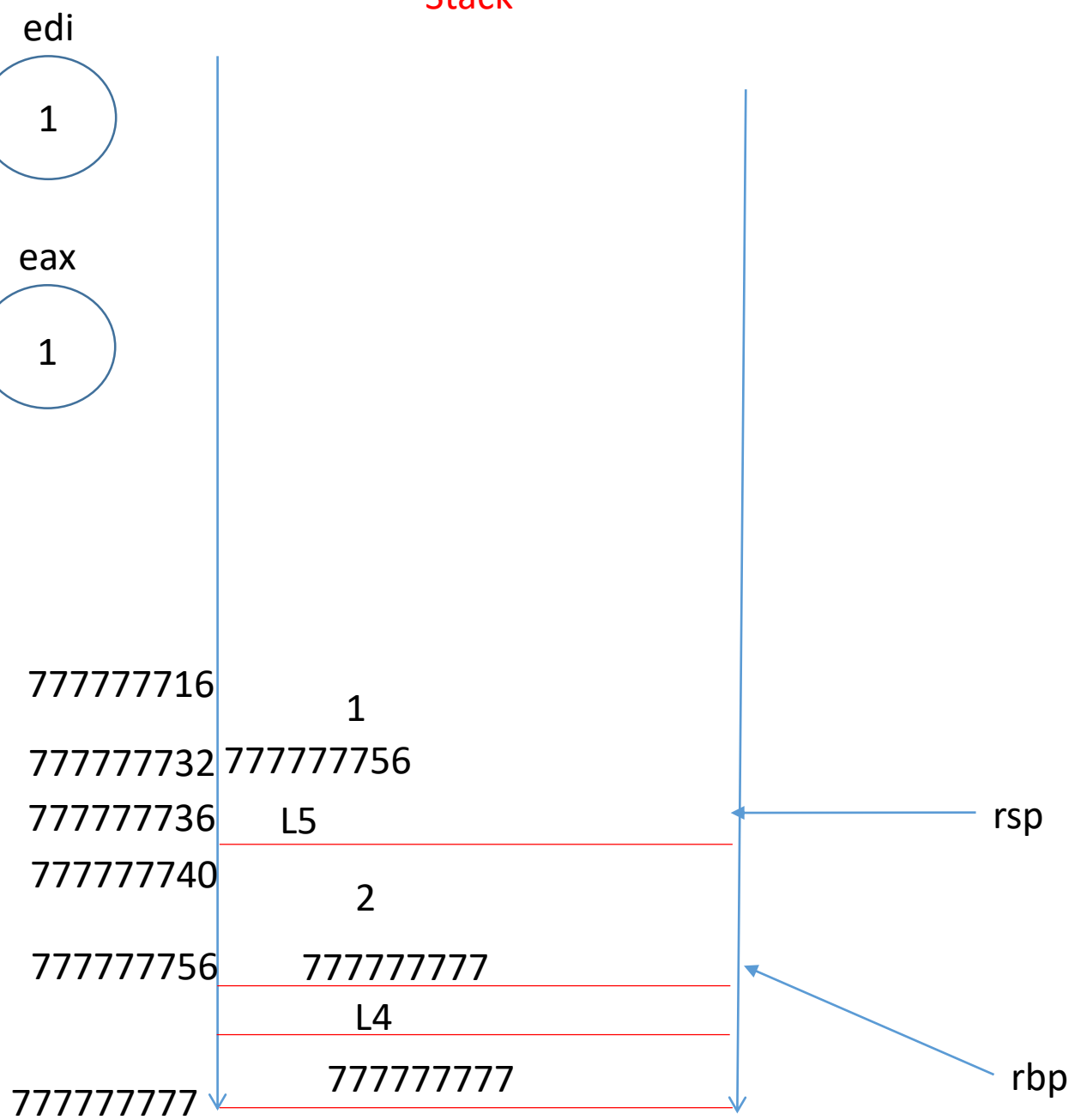
Stack



Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

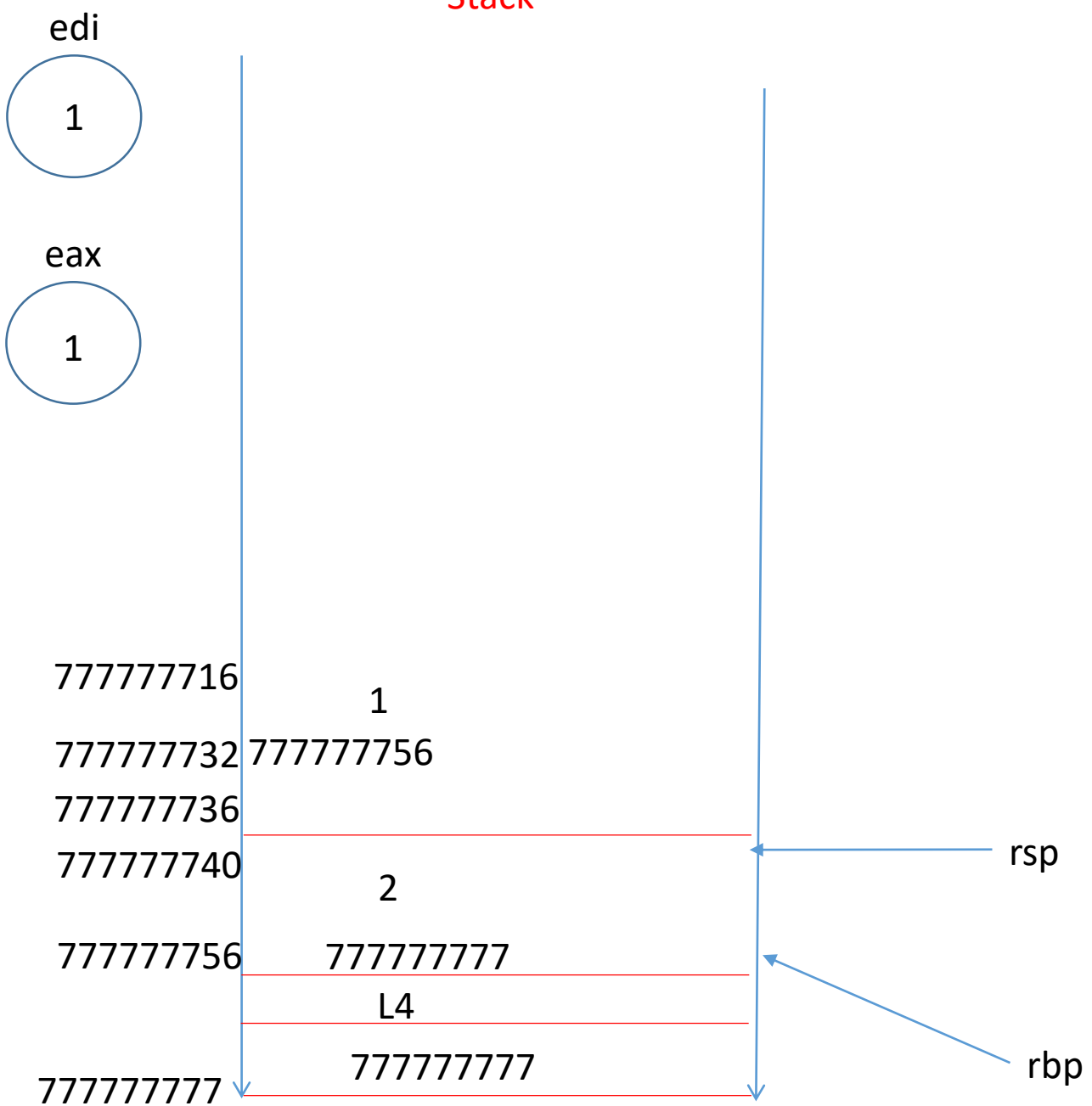
Stack



Code/Data

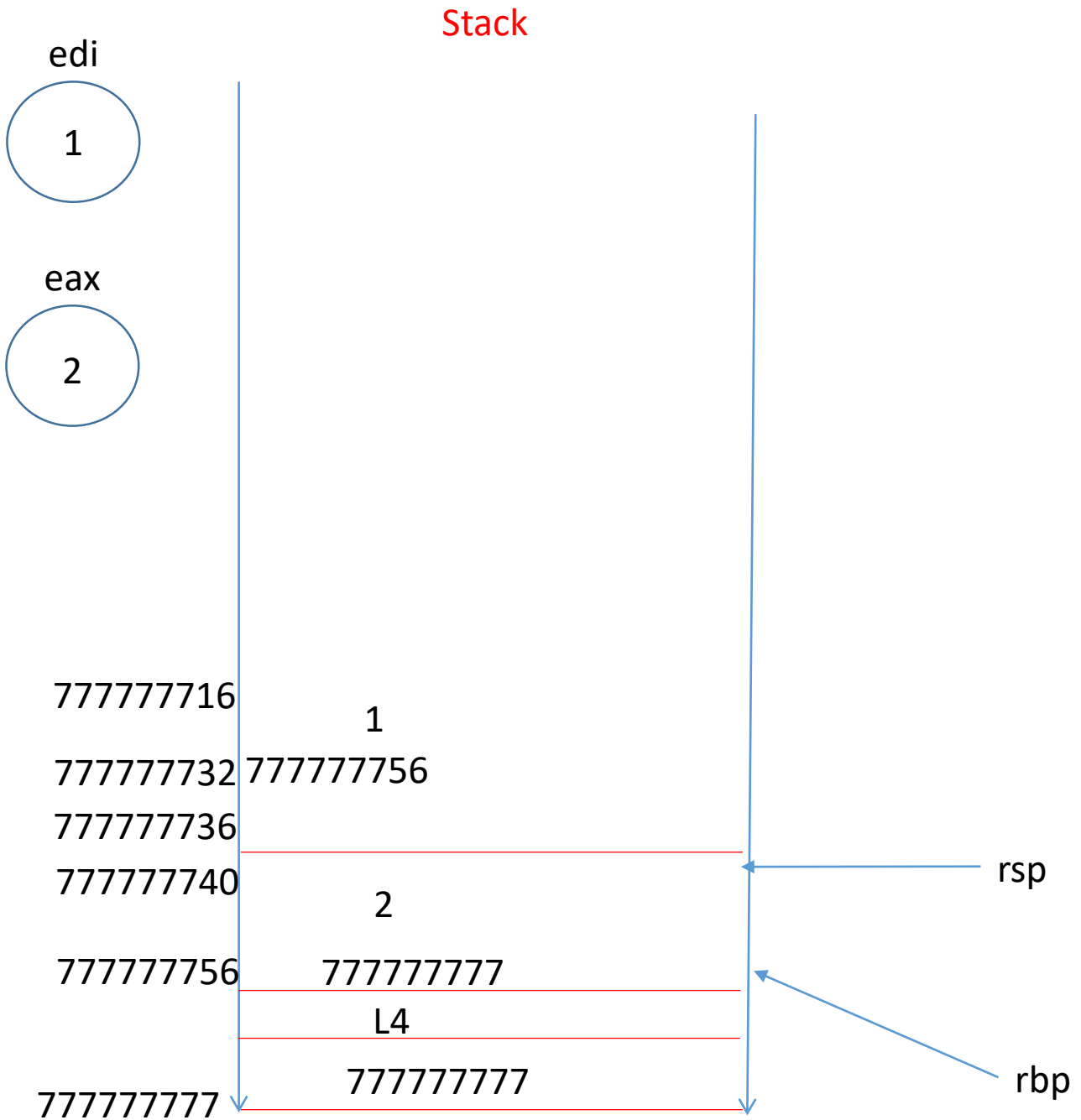
```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

Stack



Code/Data

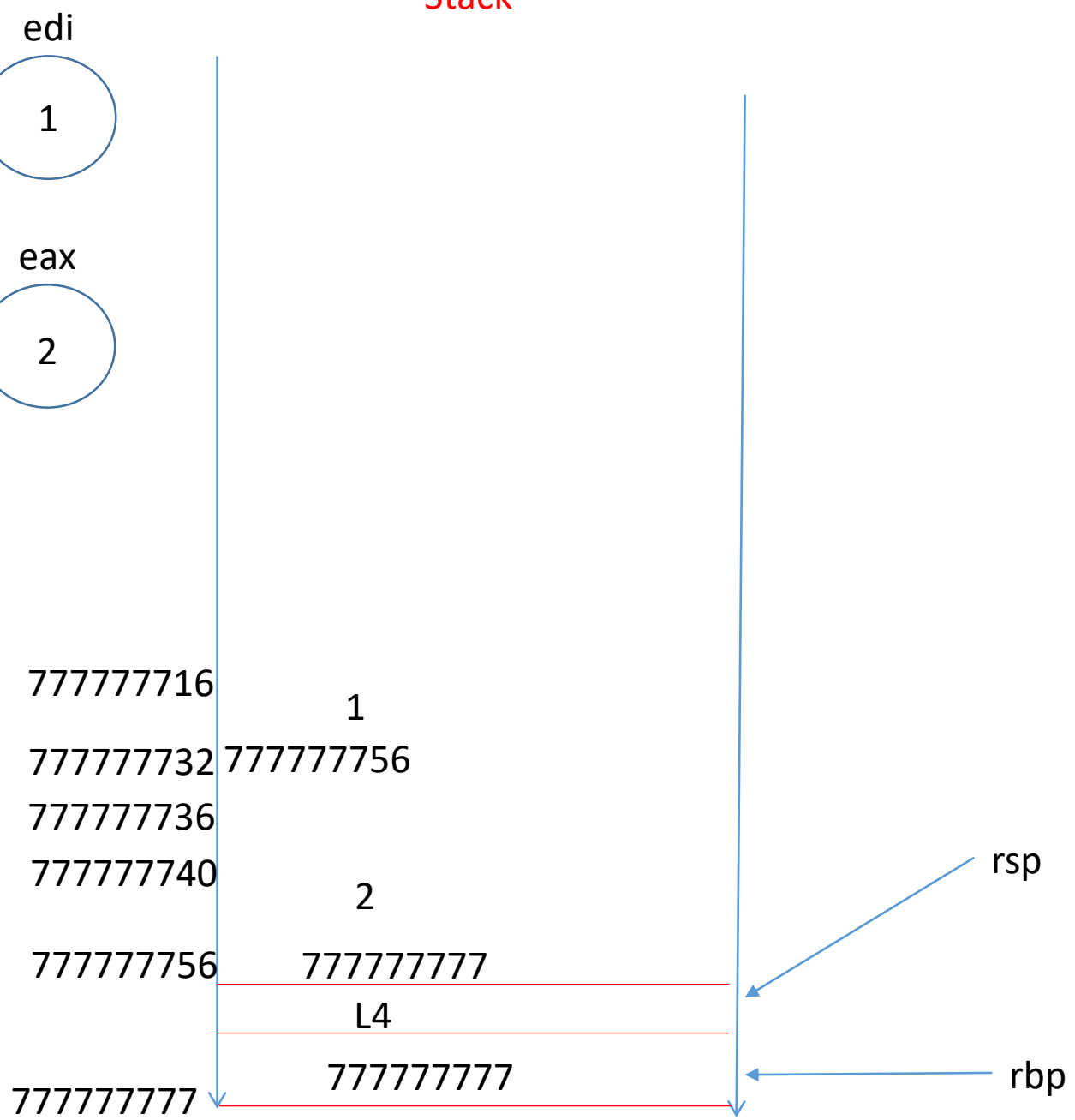
```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```



Code/Data

```
factorial(int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L2
        mov     eax, 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        sub     eax, 1
        mov     edi, eax
        call    factorial(int)
 L5:    imul    eax, DWORD PTR [rbp-4]
.L3:
        leave
        ret
…
```

Stack

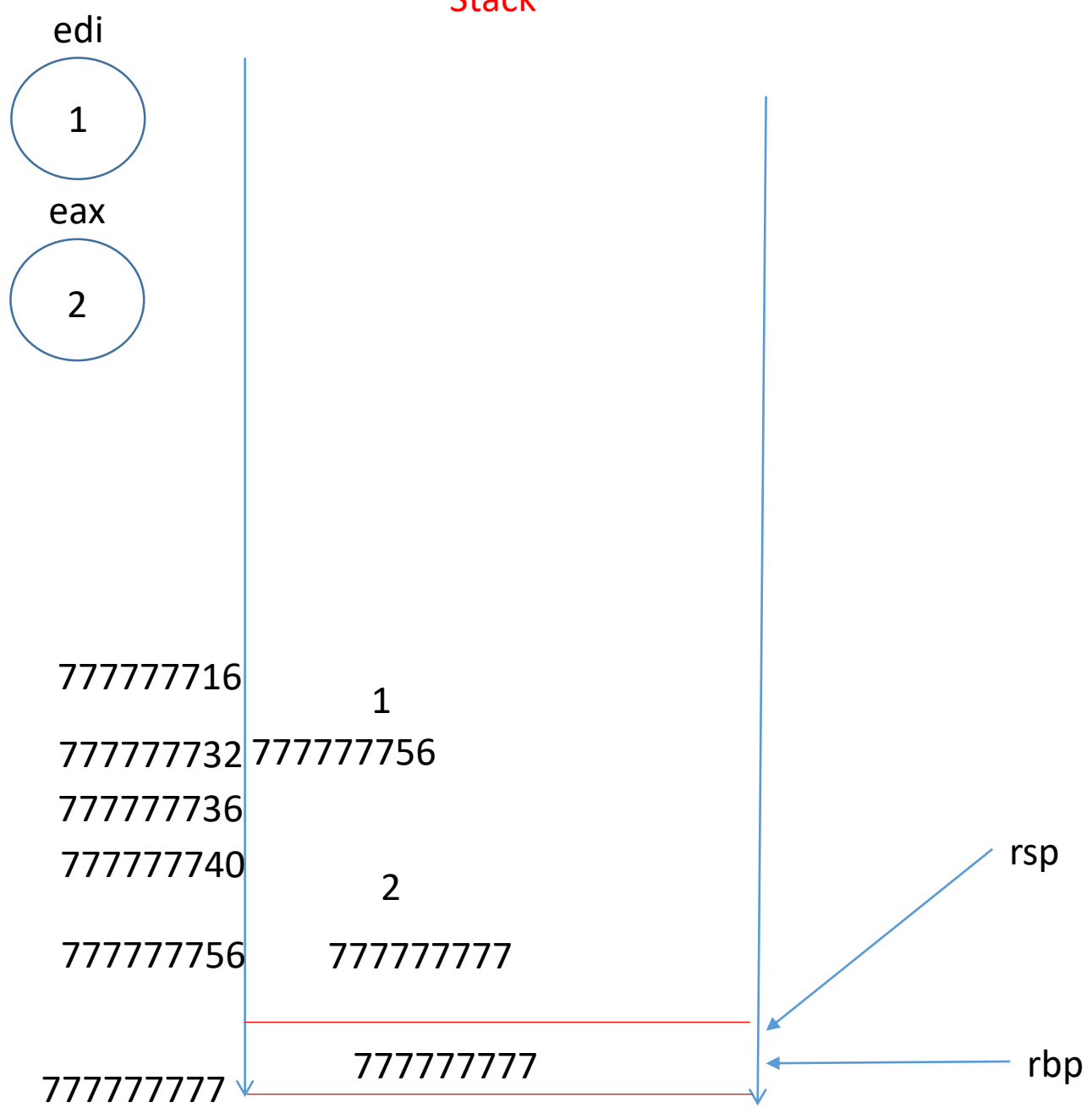


Code/Data

```
factorial(int):
        …

.LC0:
        .string "factorial(3)=%d"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, 3
        call    factorial(int)
L4:     mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

Stack

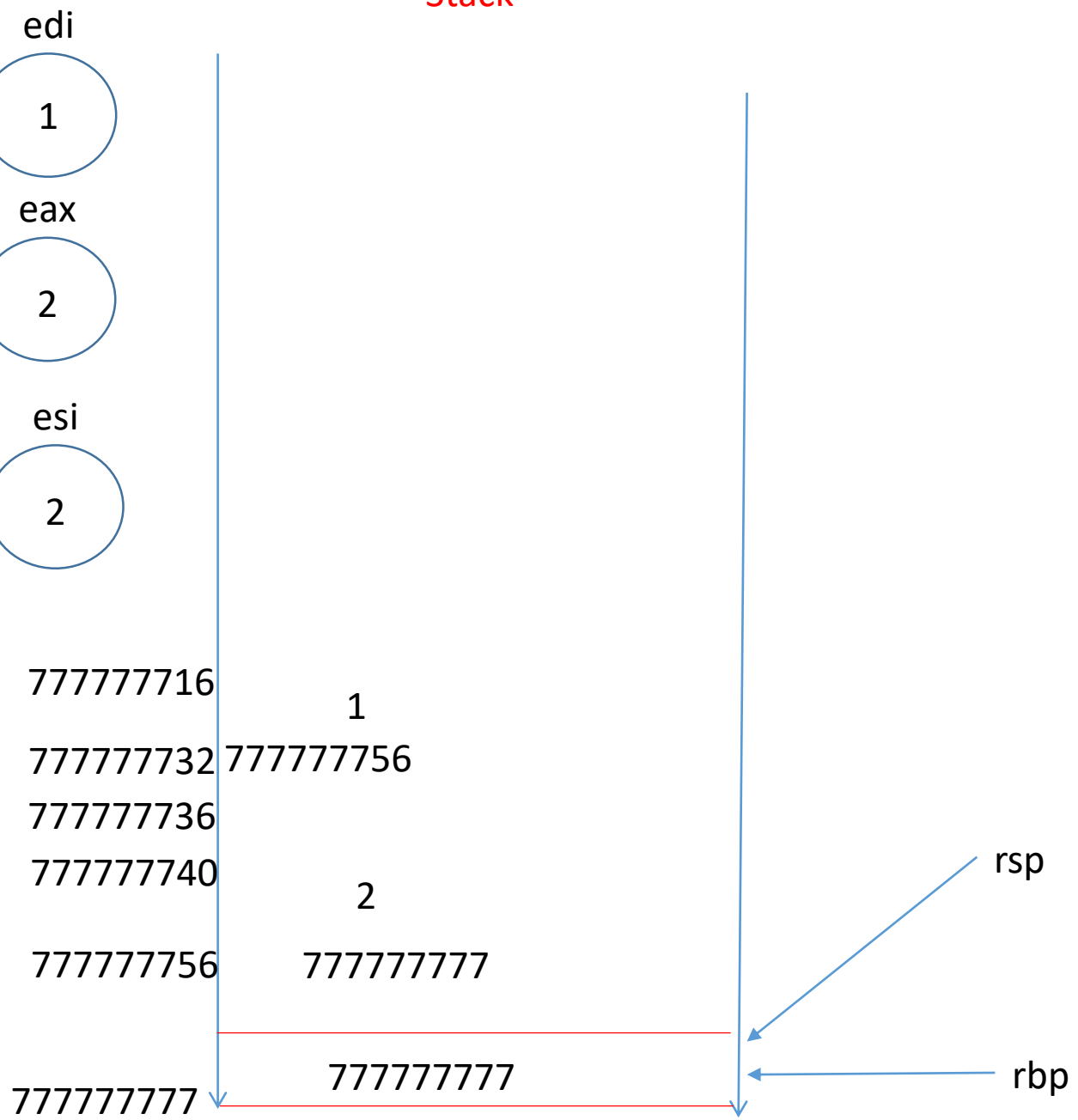


Code/Data

```
factorial(int):
        …

.LC0:
        .string "factorial(2)=%d"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, 2
        call    factorial(int)
L4:     mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

Stack

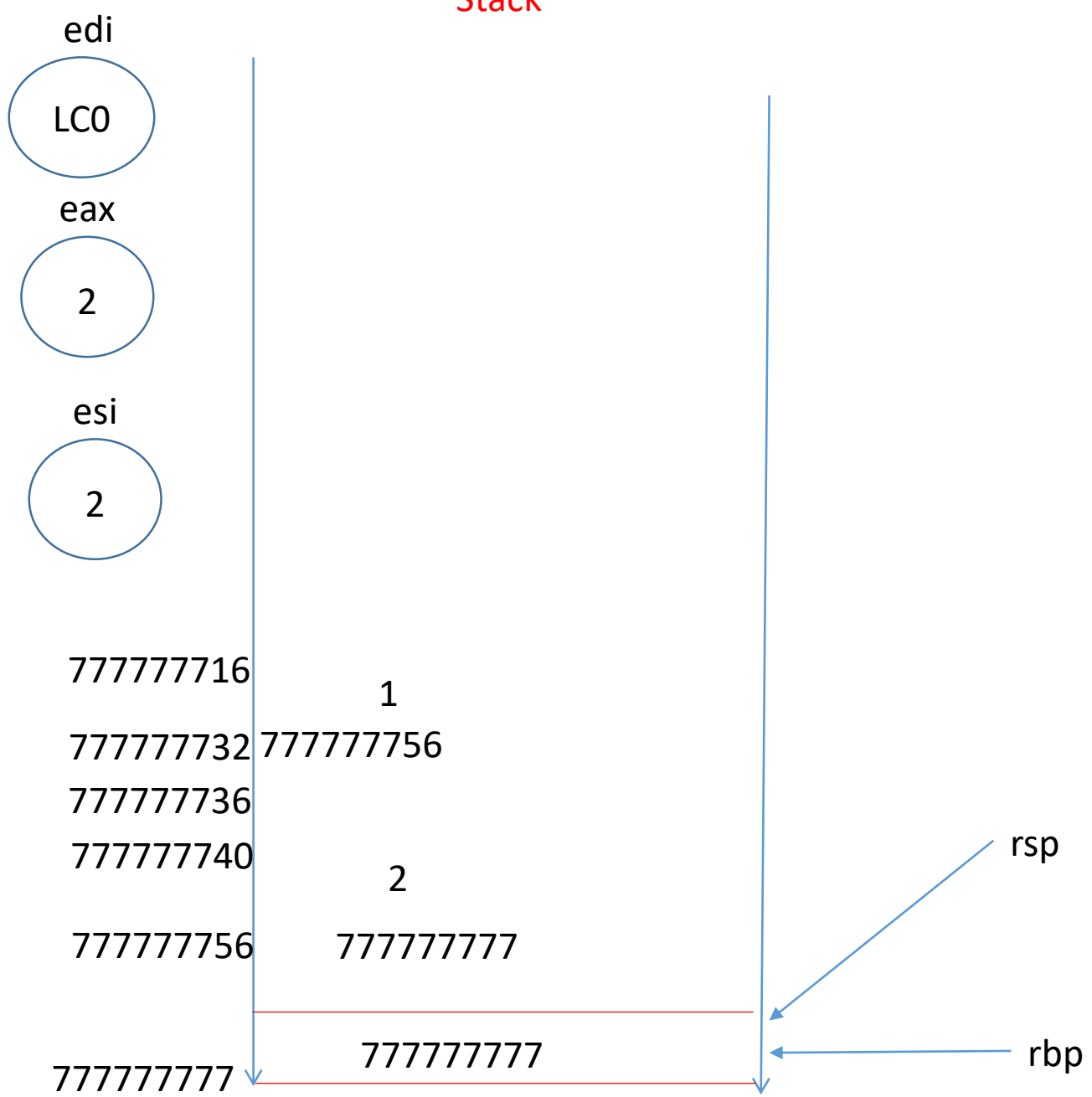


Code/Data

```
factorial(int):
        …

.LC0:
        .string "factorial(3)=%d"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, 3
        call    factorial(int)
L4:     mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

Stack

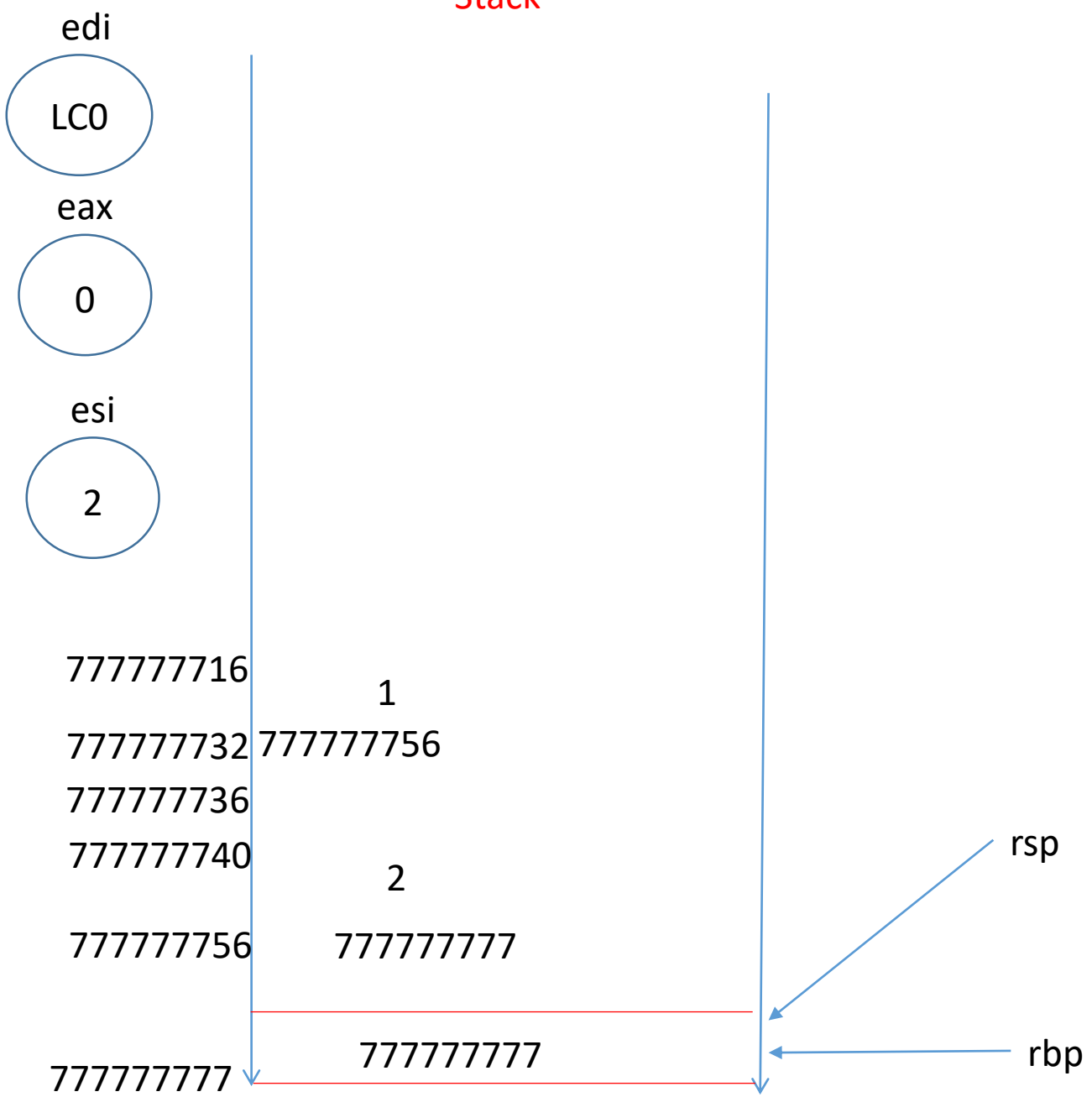


Code/Data

```
factorial(int):
        …

.LC0:
        .string "factorial(3)=%d"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, 3
        call    factorial(int)
L4:     mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

Stack



Summary

- Pentium X86 is a powerful CISC architecture
- Stack frames provide memory locality
  - Simple allocation/deallocation
  - Efficient even for recursive calls
  - Architecture support may help
- Understanding compiler generated code is not easy