?

DISCLAIMER: Simplifications!

FizzBuzz: from IDE to microcode

Yuri Khabarov

FizzBuzz

(Well-known problem)

1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, Fizz Buzz

FizzBuzz

BLIND CAREERS >

Tech Industry

Google Asked FizzBuzz

Had to wait to change my name and post, because it would've led to a doxx otherwise. I interviewed for an entry/mid-level Software…

New / Eng · butt fizz

1,297 12 23 Feb 16

Tech Industry

Google interview asked Fizzbuzz: Update

Reposting since the original post was removed, because Googlers were big mad in the comments. This is a follow up for this post:…

New / Eng · butt fizz

1,607 14 31 Feb 21

Software Engineering Career

Interviewers, have you seen people actually fail FizzBuzz?

https://www.google.com/amp/s/blog.codinghorror.com/why-cant-programmers-program/amp/ This is an older article, but I still find…

New / Eng · tau4life

727 2 18 Aug 3, 2021

Tech Industry

Are you a real software engineer if you can't solve Fizzbuzz?

Poll

MongoDB · no.sql

411 Like 20 Aug 5, 2022

Comments

```
ct_3 = 1
ct_5 = 1

for num in range(1,N):
    if ct_3 == 3:
        print('fizz')
        ct_3 = 0

    if ct_5 == 5:
        print('buzz')
        ct_5 = 0

        ct_3 += 1
        ct_5 += 1
```

"LMAO man, division is so hard, use counters and get your offer" ©

Time to experiment!

```
class FizzBuzzNaive {

    public static void main(String[] args){

        long startTime = System.currentTimeMillis();

        int max = 15_000_000;

        for (int i=1; i<=max; i++) {
            if (i % 3 == 0) {
                if (i % 5 == 0) {
                    System.out.println("FizzBuzz");
                } else {
                    System.out.println("Fizz");
                }
            } else if (i % 5 == 0) {
                System.out.println("Buzz");
            } else {
                System.out.println(i);
            }
        }

        long endTime = System.currentTimeMillis();

        System.out.println("Time in millis: " + (endTime - startTime));

    }

}
```

Time to experiment!

```
class FizzBuzzNaive {

    public static void main(String[] args){

        long startTime = System.currentTimeMillis();

        int max = 15_000_000;

        for (int i=1; i<=max; i++) {
            if (i % 3 == 0) {
                if (i % 5 == 0) {
                    System.out.println("FizzBuzz");
                } else {
                    System.out.println("Fizz");
                }
            } else if (i % 5 == 0) {
                System.out.println("Buzz");
            } else {
                System.out.println(i);
            }
        }

        long endTime = System.currentTimeMillis();

        System.out.println("Time in millis: " + (endTime - startTime));

    }

}
```

java FizzBuzzNaive > out.txt

64 sec!

Time to experiment!

```
class FizzBuzzCounters {

    public static void main(String[] args){

        long startTime = System.currentTimeMillis();

        int max = 15_000_000;

        short tree = 1;
        short five = 1;

        for (int i=1; i<=max; i++) {

            if (tree == 3) {
                System.out.print("Fizz");
                tree = 0;
            }
            if (five == 5) {
                System.out.print("Buzz");
                five = 0;
            }
            if (five == 0 || tree == 0) {
                System.out.print("\n");
            } else {
                System.out.println(i);
            }
            tree++;
            five++;

        }

        long endTime = System.currentTimeMillis();

        System.out.println("Time in millis: " + (endTime - startTime));

    }
}
```

java FizzBuzzCounters > out.txt

68 sec!

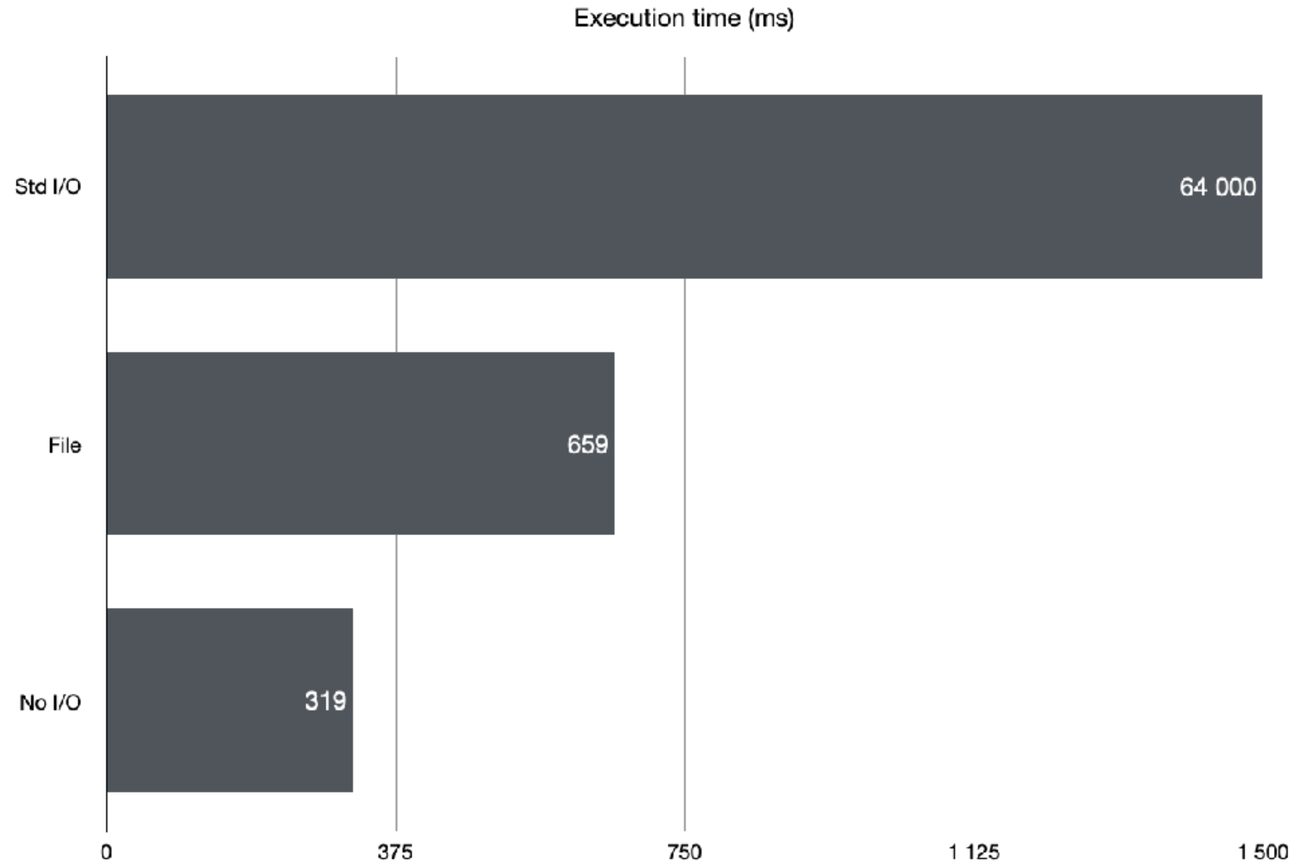
Let’s optimize
(what else we can do, huh?)

- System.out.println(); -> FileWriter?
  - Still slow, let’s don’t write to file at all
- MAYBE WE CAN CHANGE ALGORITHM?
  - (Spoiler: epic fail)
- We will not perform pattern hacks
  - We want to measure arithmetic ops

**Now we count up to 100M*

Write to file?

(We can be faster)



Eeeehm, algorithm?

```
for (int i=1; i<=max; i++) {
          if (modulo3(i)) {
              if (modulo5(i)) {
                  result[i] = "FizzBuzz";
              } else {
                  result[i] = "Fizz";
              }
          } else if (modulo5(i)) {
              result[i] = "Buzz";
          } else {
              result[i] = String.valueOf(i);
          }
      }
```

```
public static boolean modulo3(int x) {
    int divider = 3;
    int diff = x;

    while (diff > 3) {

        while (divider < x) {
            divider = divider << 1;
        }
        divider = divider >> 1;

        diff = x - divider;
        x = diff;

        divider = 3;
    }

    return diff == 3;
}
```

Well, not so cool algorithm

```
101: lload_1
102: lsub
103: invokedynamic #10,  0             // InvokeDynamic #0:makeConcatWithConstants:(J)Ljava/lang/String;
108: invokevirtual #11                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
111: return
```

Original™ solution

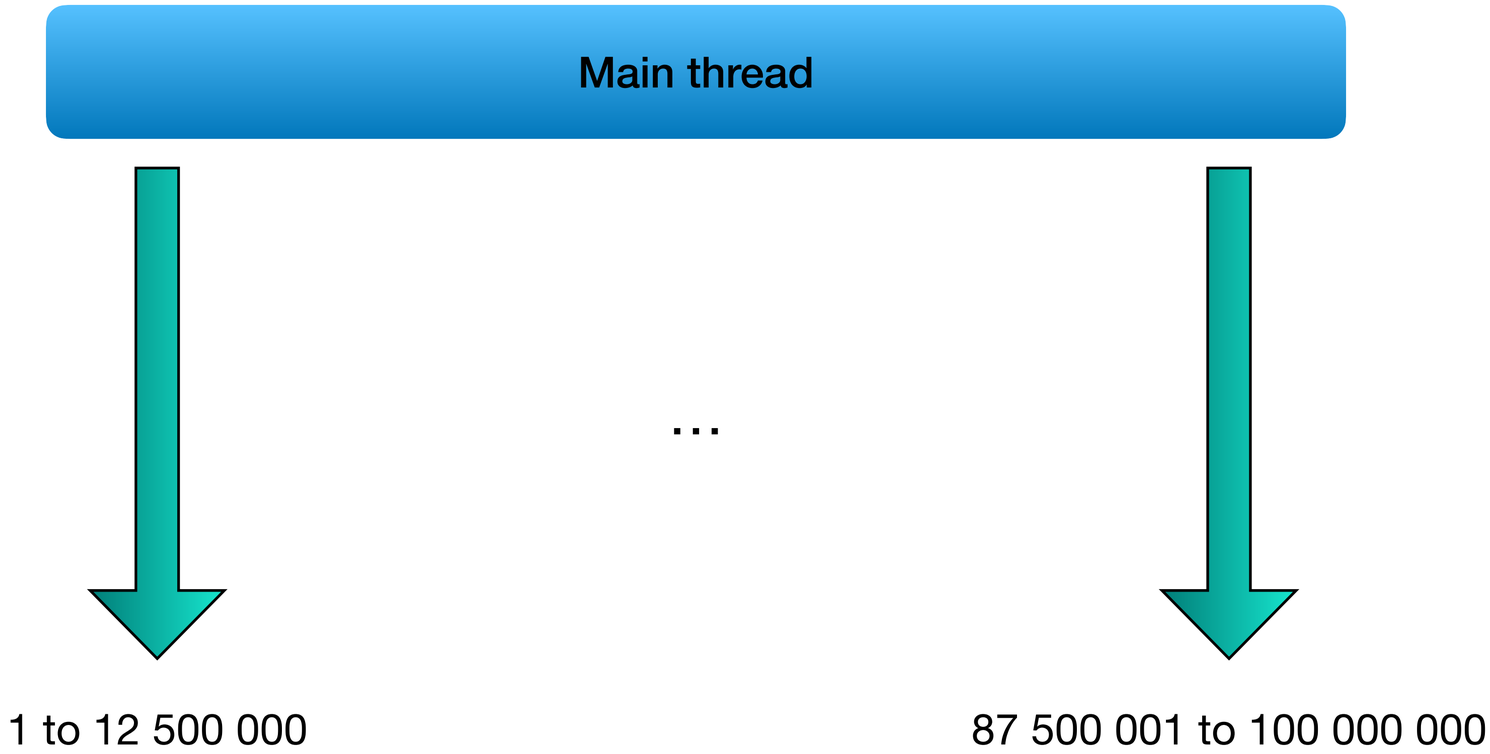
```
106: invokedynamic #12,  0             // InvokeDynamic #0:makeConcatWithConstants:(J)Ljava/lang/
String;
     111: invokevirtual #13                 // Method java/io/PrintStream.println:(Ljava/lang/
String;)V
     114: return

  public static boolean modulo3(int);
    Code:
       0: iconst_3
       1: istore_1
       2: iload_0
    . . .
      42: goto          46
      45: iconst_0
      46: ireturn
    . . .
```

"Algorithm"

Let’s optimise more
(At least we want to shoehorn)

- Let’s use C language?
  - Faster and native
- Let’s measure business-logic time?
- Get rid of some functions!
  - *sprintf( ) is* ***expensive***
    - Why not *itoa()* btw?
  - *malloc( ) is* ***expensive***
- Let’s use multithreading!
  - Finally good idea!

Multithreading



Using C language

```c
void fizzbuzz(int start, int end) {
    int i;
    for (i = start; i < end; i++) {
        if (i % 3 == 0 && i % 5 == 0) {
            printf("FizzBuzz");
        } else if (i % 3 == 0) {
            printf("Fizz");
        } else if (i % 5 == 0) {
            printf("Buzz");
        } else {
            printf("%d", i);
        }
    }
}
```

Avoiding stdout

Passing structure with data

```
void *fizzbuzz(void *arg) {
    struct thread_data *data = (struct thread_data *) arg;
    int i;
    for (i = data->start; i < data->end; i++) {
        if (i % 3 == 0 && i % 5 == 0) {
            char* buffer = malloc(sizeof(char) * 9);
            sprintf(buffer, "FizzBuzz");
            data->output_array[i-1] = buffer;
        } else if (i % 3 == 0) {
            char* buffer = malloc(sizeof(char) * 5);
            sprintf(buffer, "Fizz");
            data->output_array[i-1] = buffer;
        } else if (i % 5 == 0) {
            char* buffer = malloc(sizeof(char) * 5);
            sprintf(buffer, "Buzz");
            data->output_array[i-1] = buffer;
        } else {
            char* buffer = malloc(sizeof(char) * 10);
            sprintf(buffer, "%d", i);
            data->output_array[i-1] = buffer;
        }
    }
    pthread_exit(NULL);
}
```

Passing result to memory

Avoiding stdout

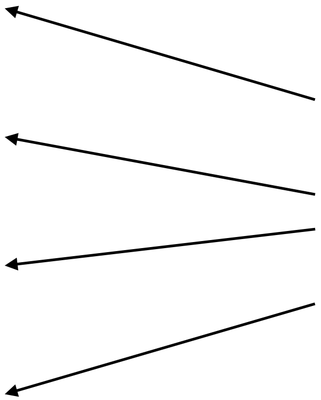
Passing structure with data

```
void *fizzbuzz(void *arg) {
    struct thread_data *data = (struct thread_data *) arg;
    int i;
    for (i = data->start; i < data->end; i++) {
        if (i % 3 == 0 && i % 5 == 0) {
            char* buffer = malloc(sizeof(char) * 9);
            sprintf(buffer, "FizzBuzz");
            data->output_array[i-1] = buffer;
        } else if (i % 3 == 0) {
            char* buffer = malloc(sizeof(char) * 5);
            sprintf(buffer, "Fizz");
            data->output_array[i-1] = buffer;
        } else if (i % 5 == 0) {
            char* buffer = malloc(sizeof(char) * 5);
            sprintf(buffer, "Buzz");
            data->output_array[i-1] = buffer;
        } else {
            char* buffer = malloc(sizeof(char) * 10);
            sprintf(buffer, "%d", i);
            data->output_array[i-1] = buffer;
        }
    }
    pthread_exit(NULL);
}
```

Passing result to memory

Get rid of malloc();

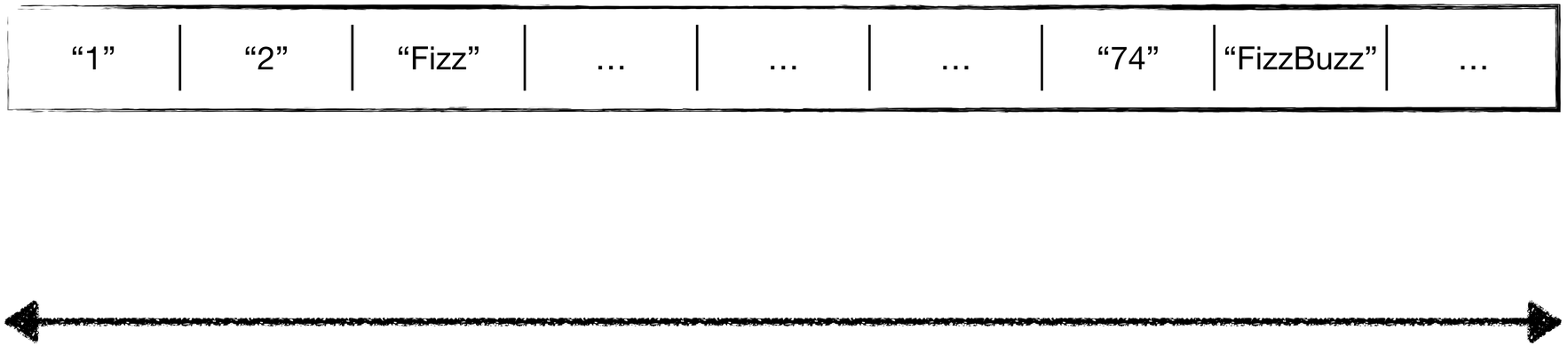
```c
void *fizzbuzz(void *arg) {
    struct thread_data *data = (struct thread_data *) arg;
    int i;
    for (i = data->start; i < data->end; i++) {
        if (i % 3 == 0 && i % 5 == 0) {
            char* buffer = malloc(sizeof(char) * 9);
            sprintf(buffer, "FizzBuzz");
            data->output_array[i-1] = buffer;
        } else if (i % 3 == 0) {
            char* buffer = malloc(sizeof(char) * 5);
            sprintf(buffer, "Fizz");
            data->output_array[i-1] = buffer;
        } else if (i % 5 == 0) {
            char* buffer = malloc(sizeof(char) * 5);
            sprintf(buffer, "Buzz");
            data->output_array[i-1] = buffer;
        } else {
            char* buffer = malloc(sizeof(char) * 10);
            sprintf(buffer, "%d", i);
            data->output_array[i-1] = buffer;
        }
    }
    pthread_exit(NULL);
}
```



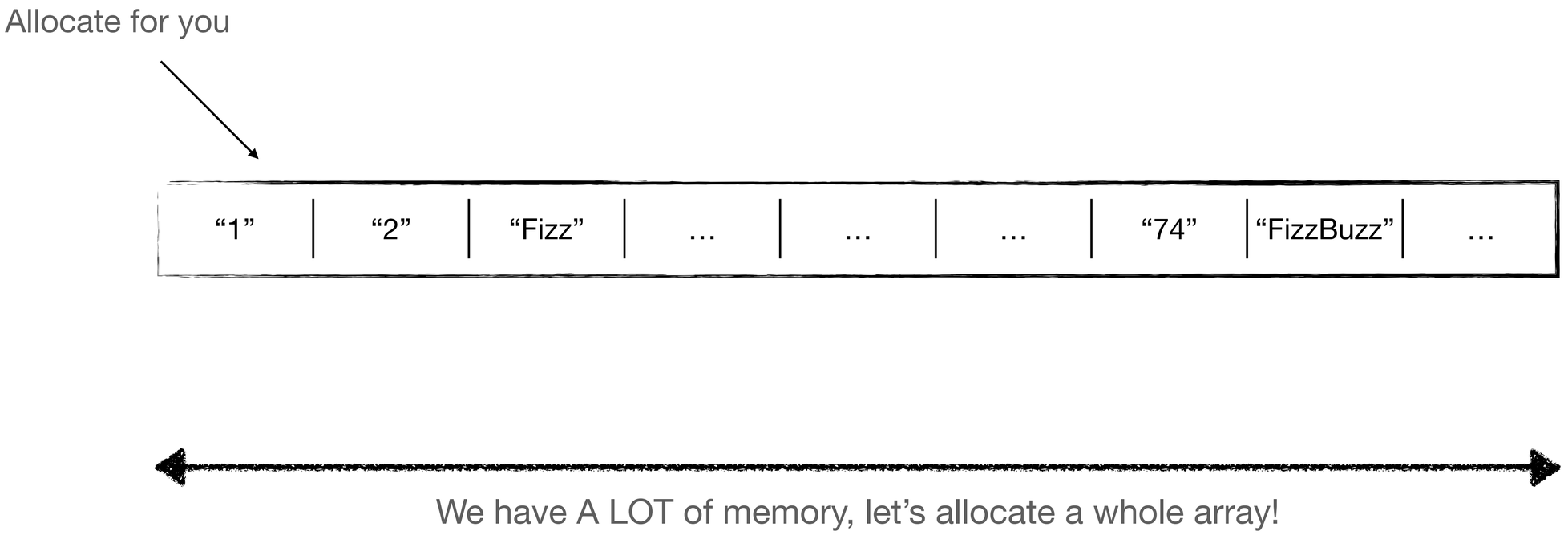
Too much allocations

Get rid of malloc();

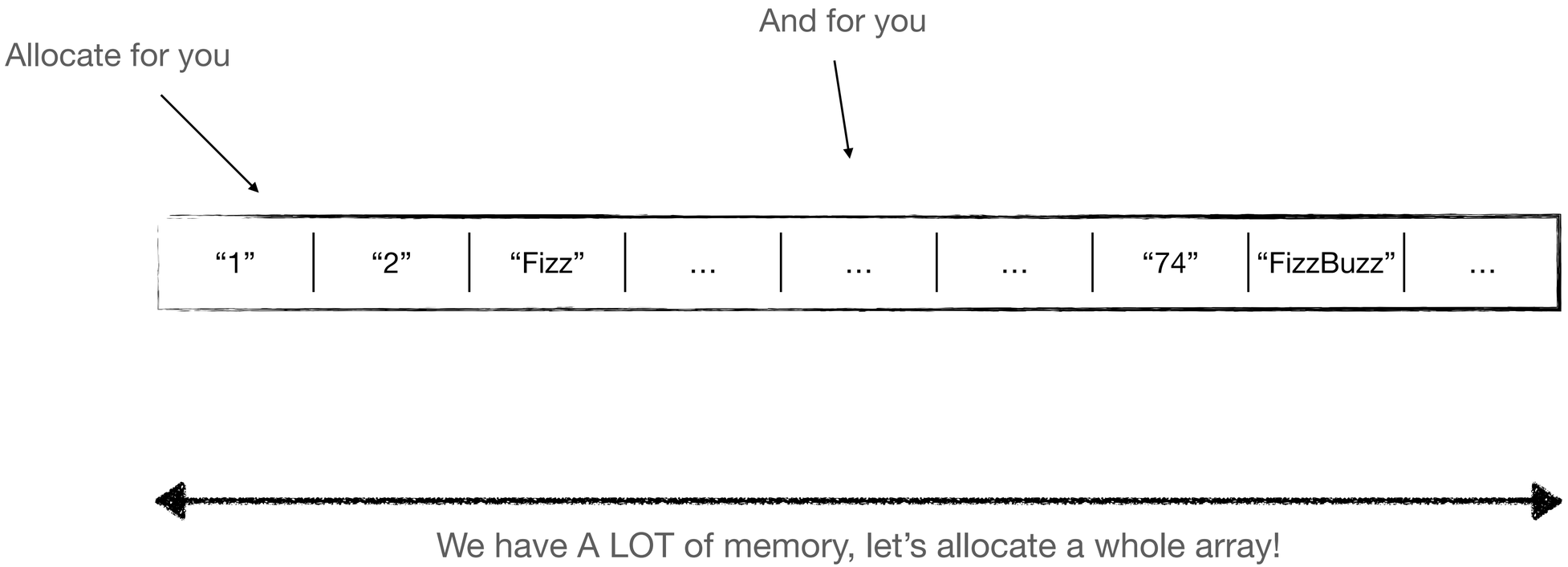
Get rid of malloc();



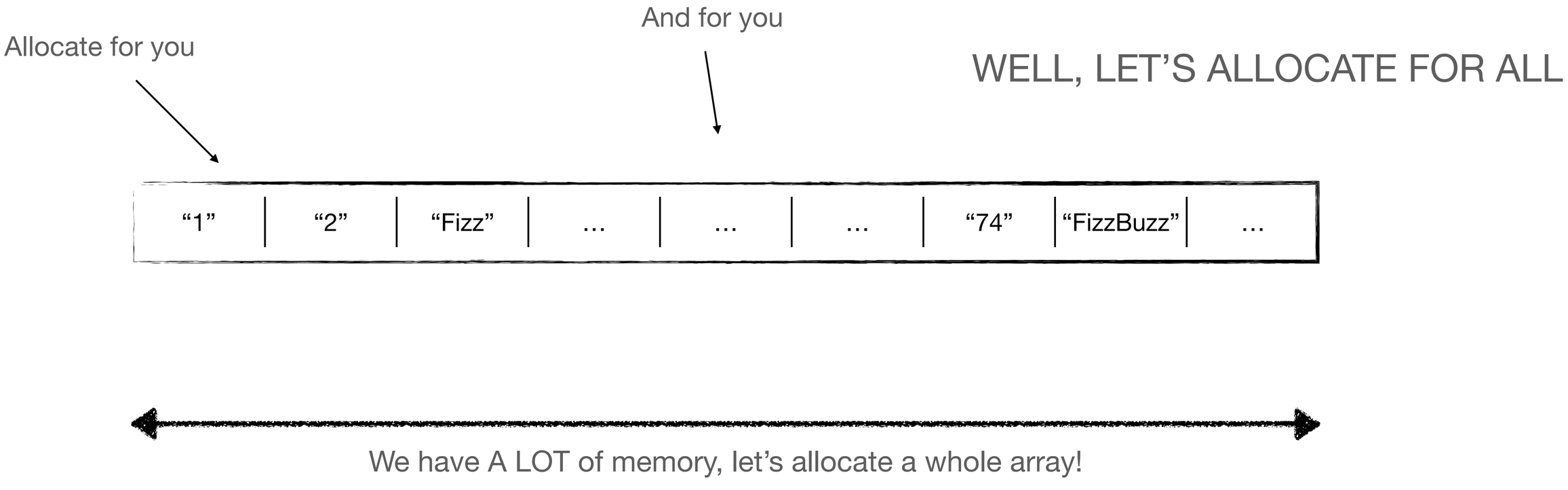
Get rid of malloc();



Get rid of malloc();



Get rid of malloc();



Get rid of sprintf();

integer 420

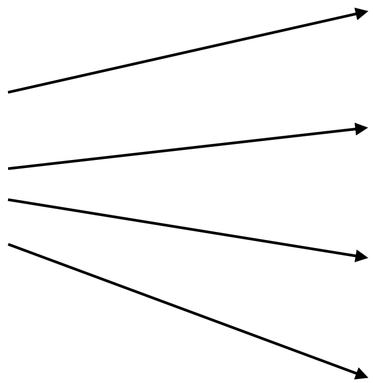
```
sprintf(buffer, "%d", i);
```

String “420”

Get rid of sprintf();

- Transform number to string is expensive
- It’s easier to replace any number by some constant string

Too much calls



```c
void *fizzbuzz(void *arg) {
    struct thread_data *data = (struct thread_data *) arg;
    int i;
    for (i = data->start; i < data->end; i++) {
        if (i % 3 == 0 && i % 5 == 0) {
            char* buffer = malloc(sizeof(char) * 9);
            sprintf(buffer, "FizzBuzz");
            data->output_array[i-1] = buffer;
        } else if (i % 3 == 0) {
            char* buffer = malloc(sizeof(char) * 5);
            sprintf(buffer, "Fizz");
            data->output_array[i-1] = buffer;
        } else if (i % 5 == 0) {
            char* buffer = malloc(sizeof(char) * 5);
            sprintf(buffer, "Buzz");
            data->output_array[i-1] = buffer;
        } else {
            char* buffer = malloc(sizeof(char) * 10);
            sprintf(buffer, "%d", i);
            data->output_array[i-1] = buffer;
        }
    }
    pthread_exit(NULL);
}
```

Get rid of sprintf();

```
char fizzBuzzArray[9];
char fizzArray[8];
char buzzArray[8];
char xArr[2];
```



```
strcpy(fizzBuzzArray, "FizzBuzz");
strcpy(fizzArray, "Fizz");
strcpy(buzzArray, "Buzz");
strcpy(xArr, "x");
```



```
for (i = data->start; i < data->end; i++) {
        if (i % 3 == 0 && i % 5 == 0) {
            data->output_array[i - 1] = fizzBuzzArray;
        } else if (i % 3 == 0) {
            data->output_array[i - 1] = fizzArray;
        } else if (i % 5 == 0) {
            data->output_array[i - 1] = buzzArray;
        } else {
            data->output_array[i-1] = xArr;
        }
    }
```

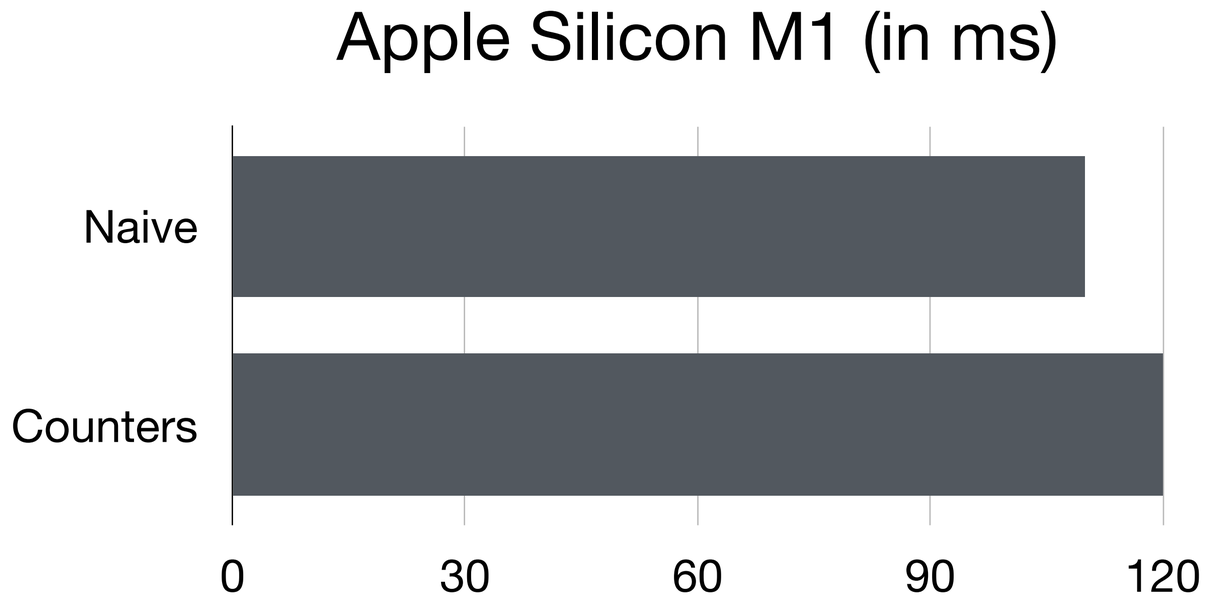
Looks like a String Pool in Java, heh?

Result

```
void *fizzbuzz(void *arg) {
    struct thread_data *data = (struct thread_data *) arg;
    int i;

    for (i = data->start; i < data->end; i++) {
        if (i % 3 == 0 && i % 5 == 0) {
            data->output_array[i - 1] = fizzBuzzArray;
        } else if (i % 3 == 0) {
            data->output_array[i - 1] = fizzArray;
        } else if (i % 5 == 0) {
            data->output_array[i - 1] = buzzArray;
        } else {
            data->output_array[i-1] = xArr;
        }
    }
    pthread_exit(NULL);
}
```

Let’s use function pointer

After optimisation



Hmm maybe change CPU?

That’s M1

👉

That’s Intel

Hmm maybe change CPU?

That’s M1

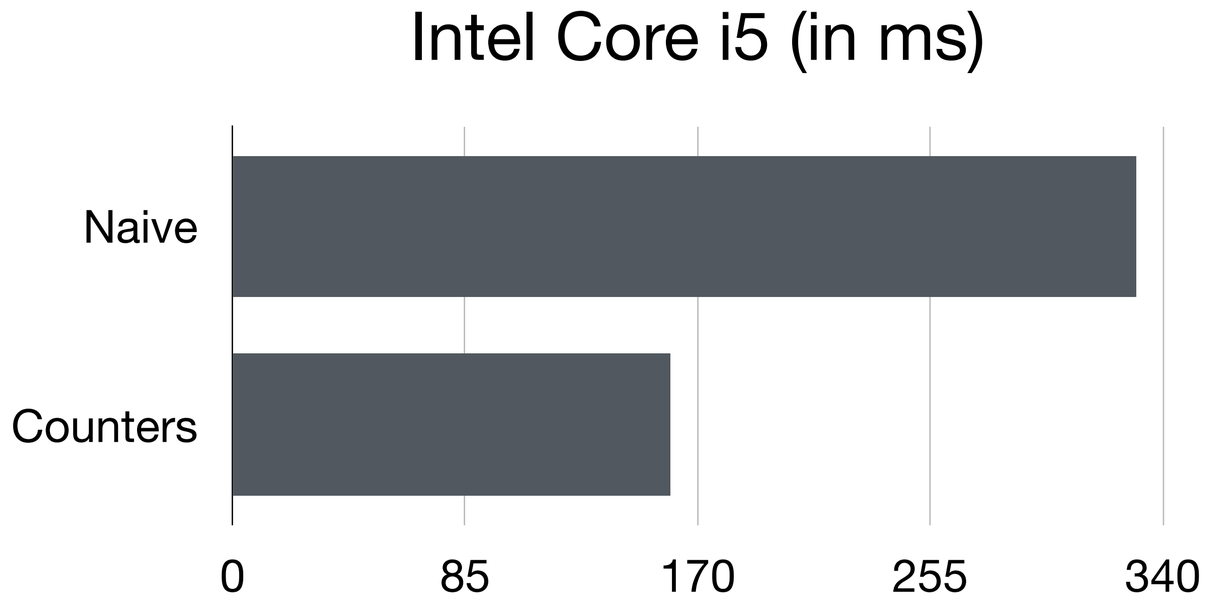
ARM

👉

That’s Intel

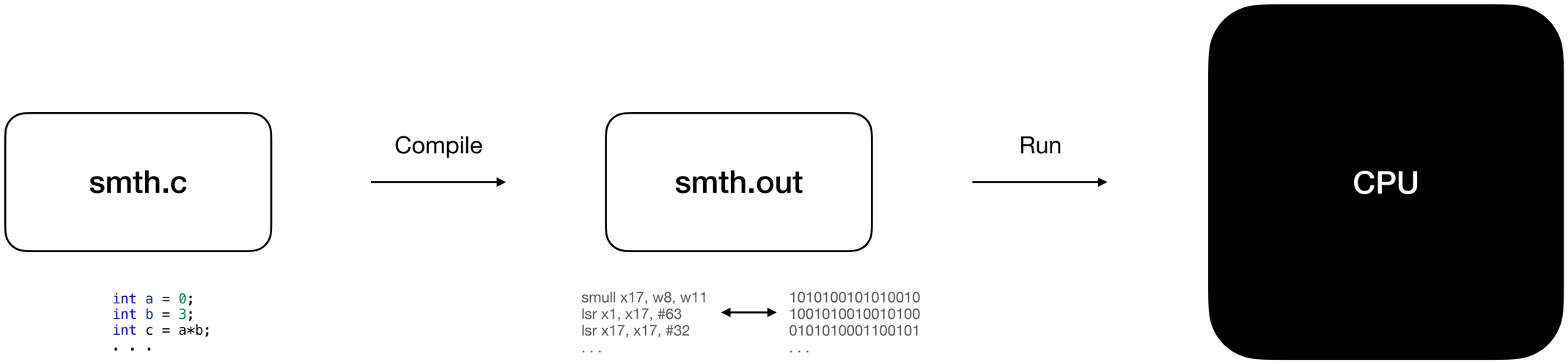
x86

Multithreading using Intel

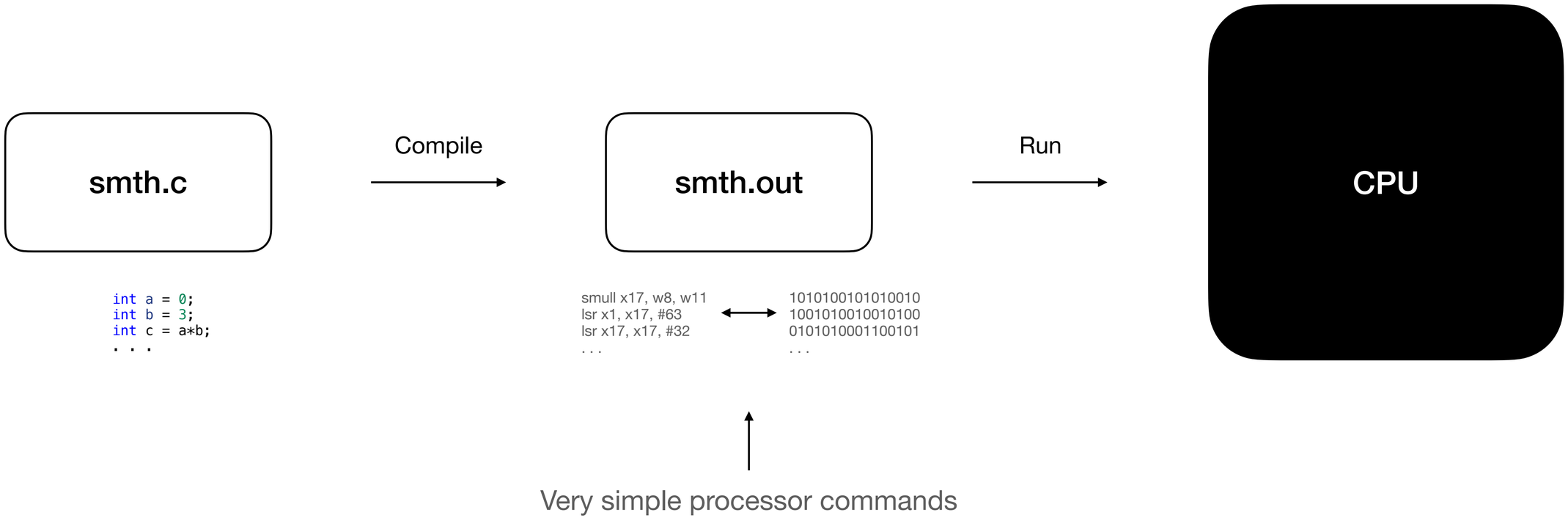


HOW. DOES. IT. WORK?

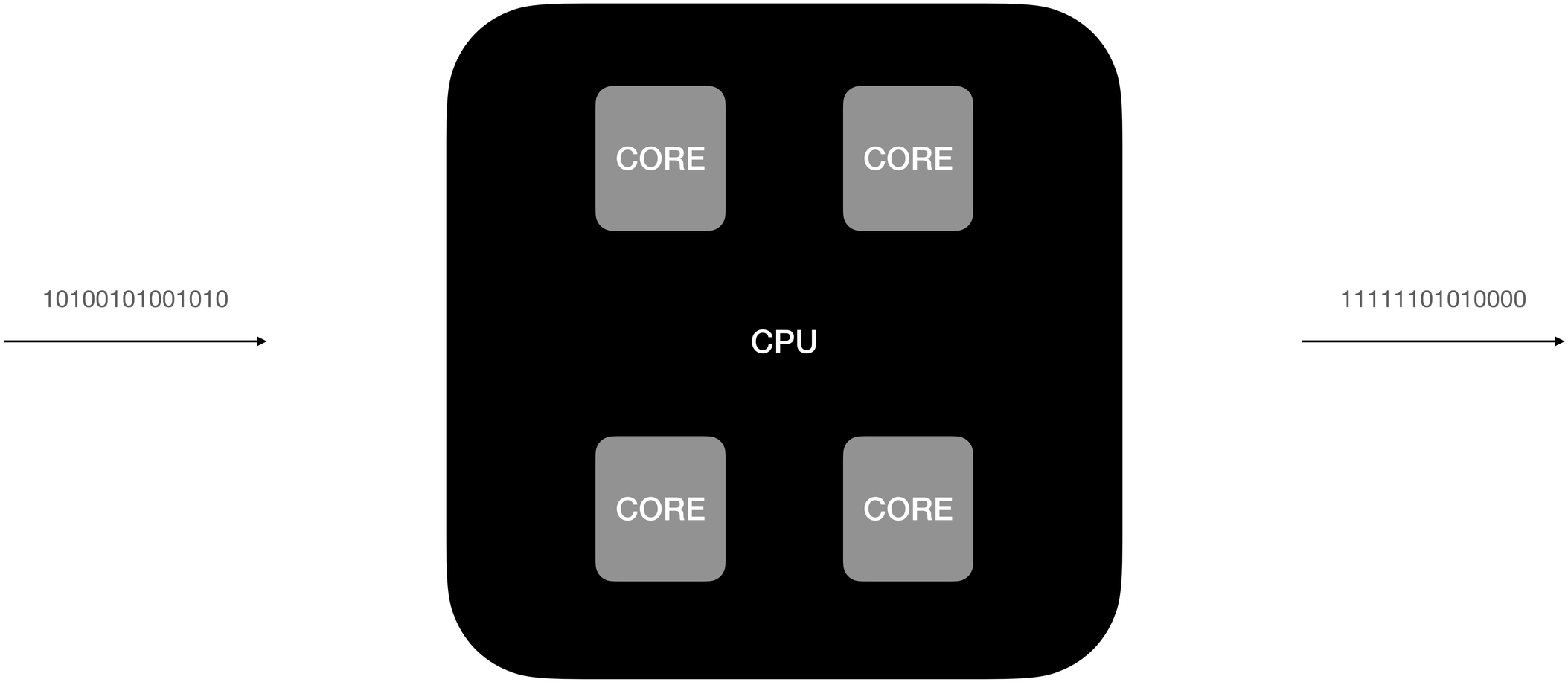
How program works



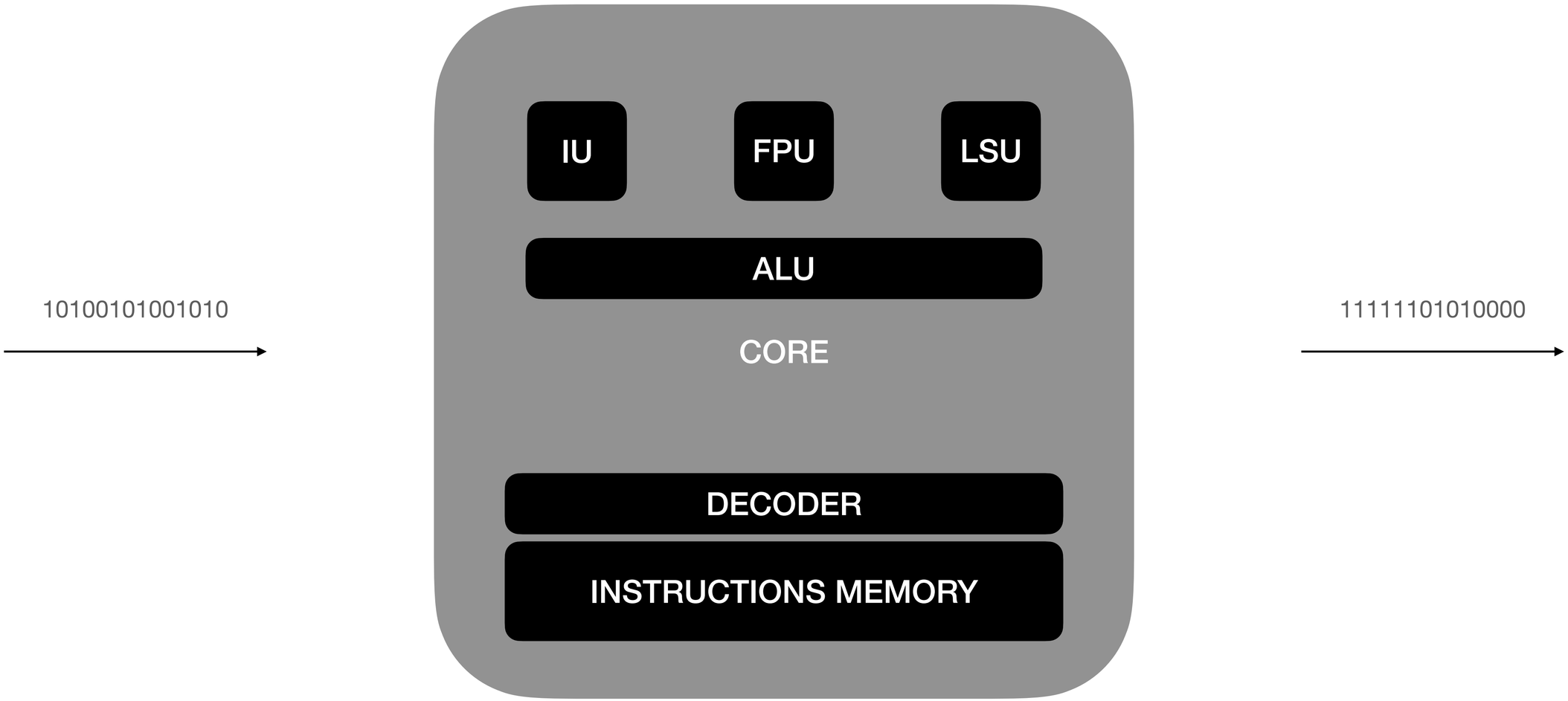
How program works



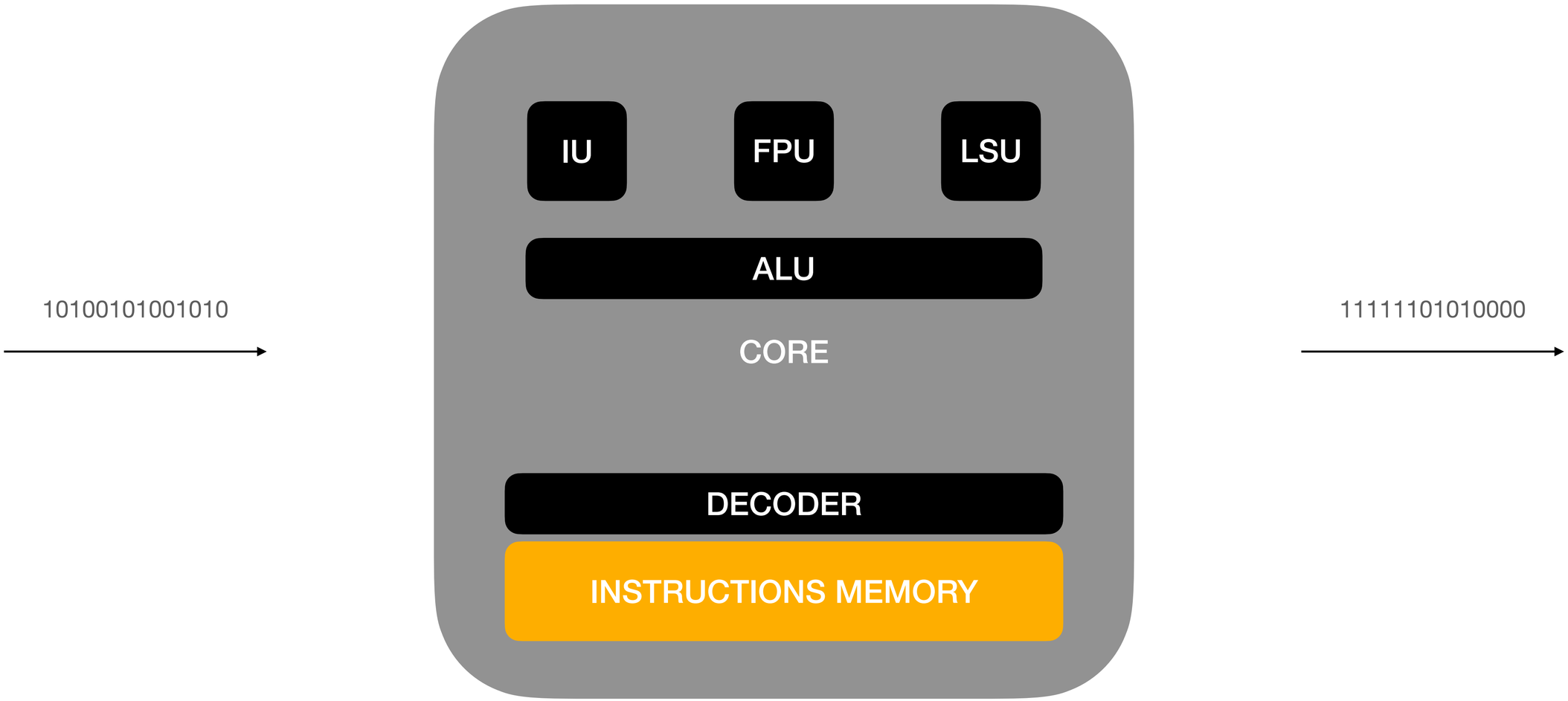
How program works



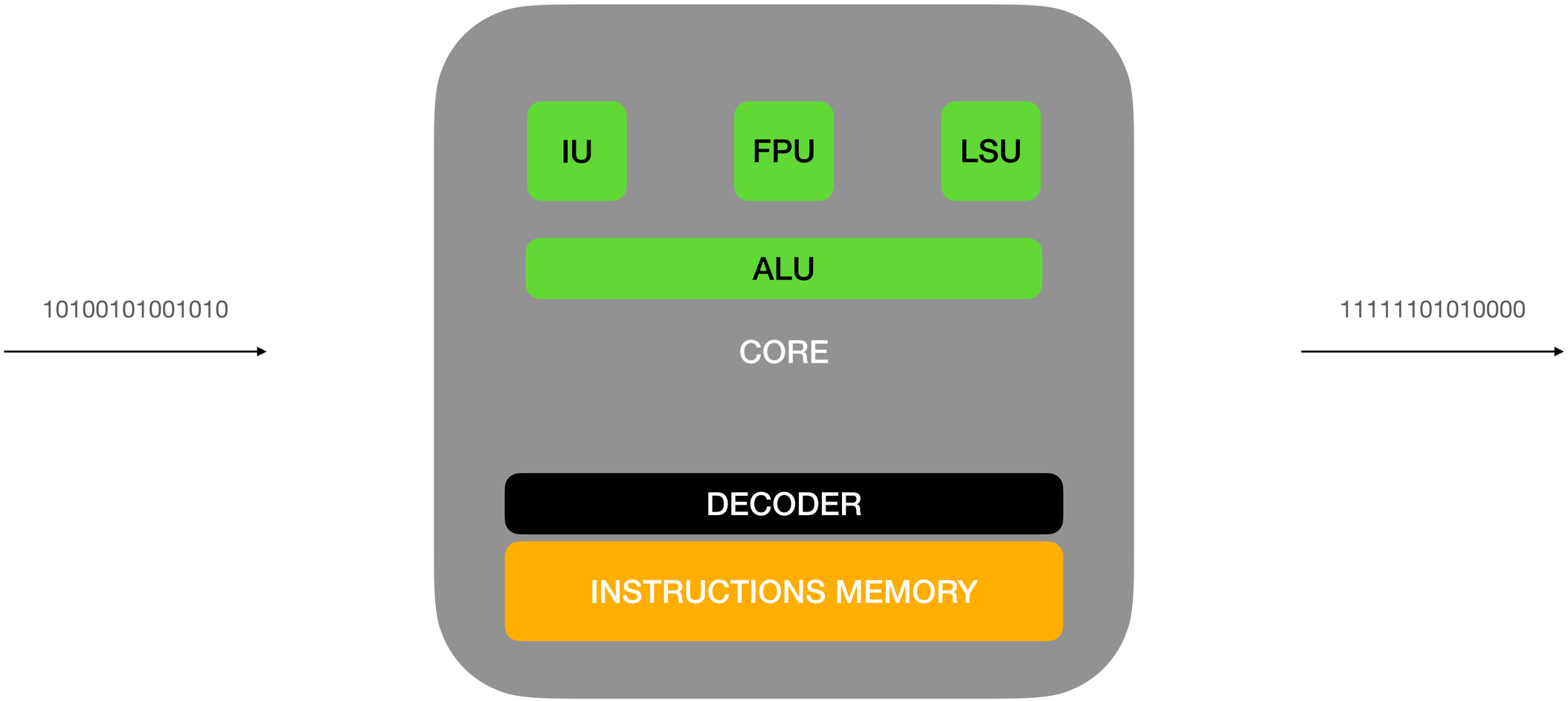
How program works



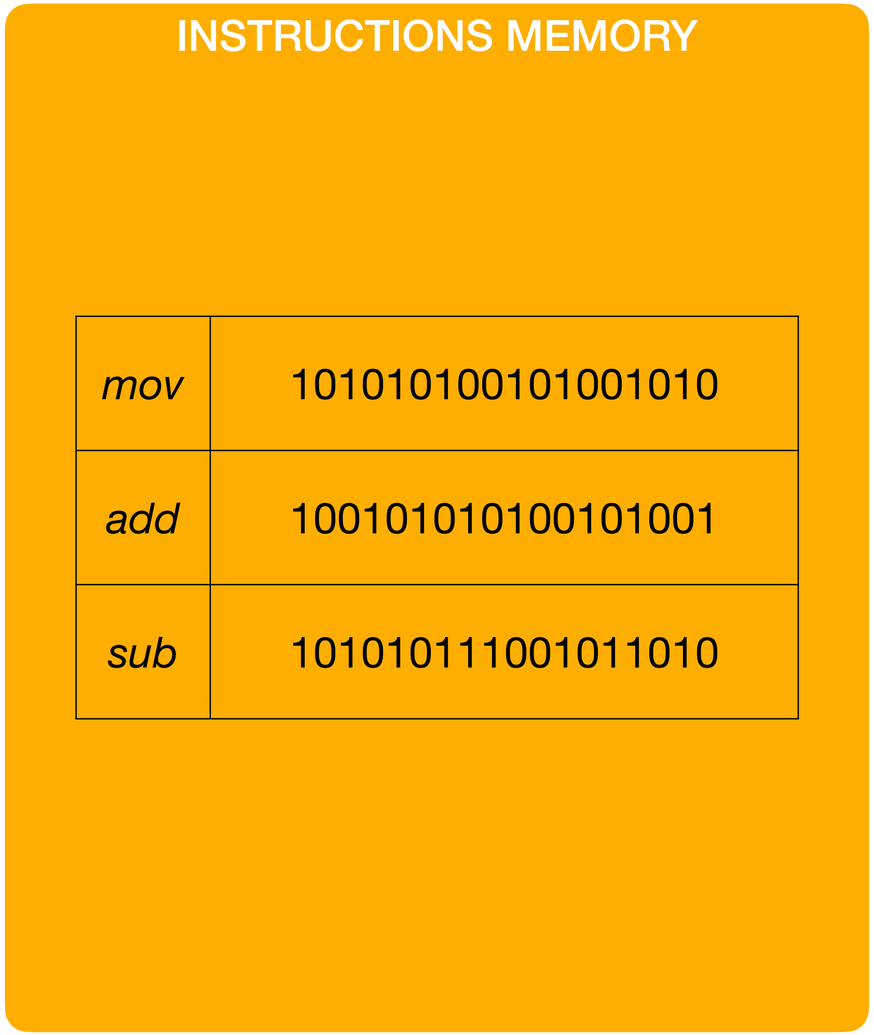
How program works



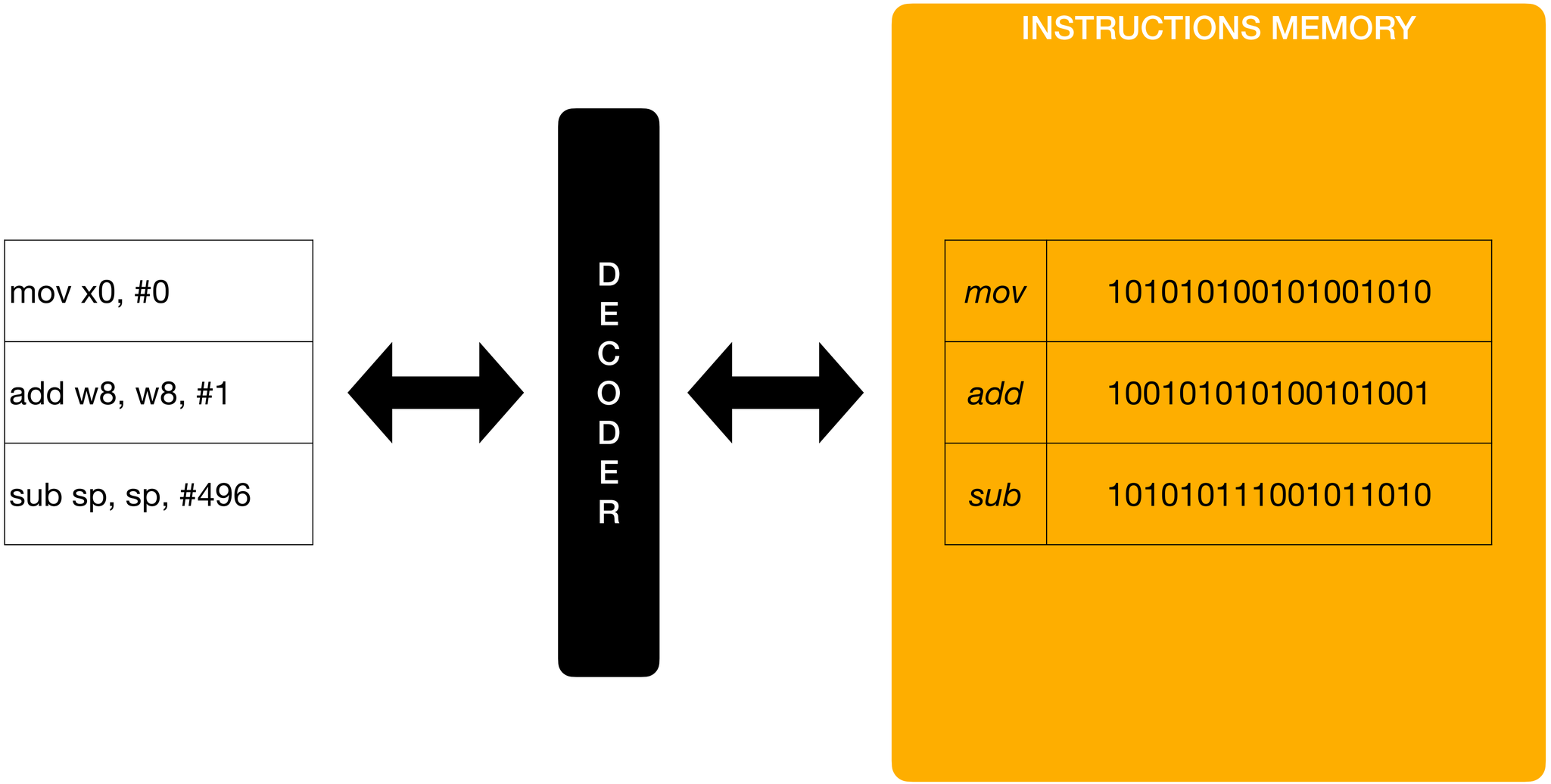
How program works



Microcode


INSTRUCTIONS MEMORY
mov
1010101001010010
add
1001010101001010 01
sub
1010101110010110 10

Microcode

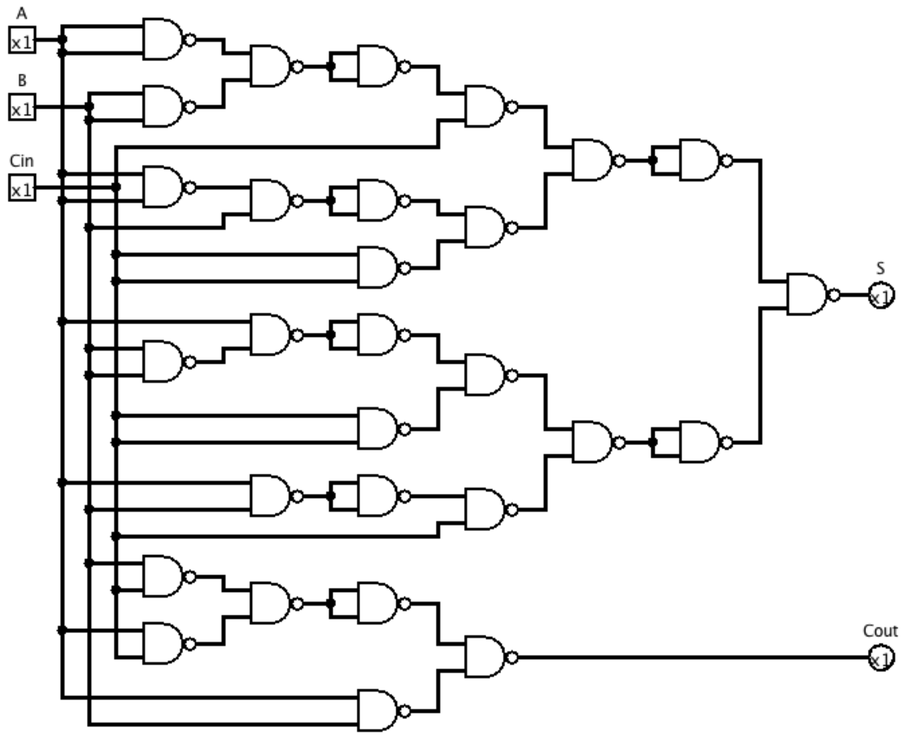

INSTRUCTIONS MEMORY
mov x0, #0
add w8, w8, #1
sub sp, sp, #496
DECODER
mov
1010101001010010
add
1001010101001010 01
sub
1010101110010110 10

Microcode

INSTRUCTIONS MEMORY

mov	1010101001010010 10
add	1001010101001010 01
sub	1010101110010110 10

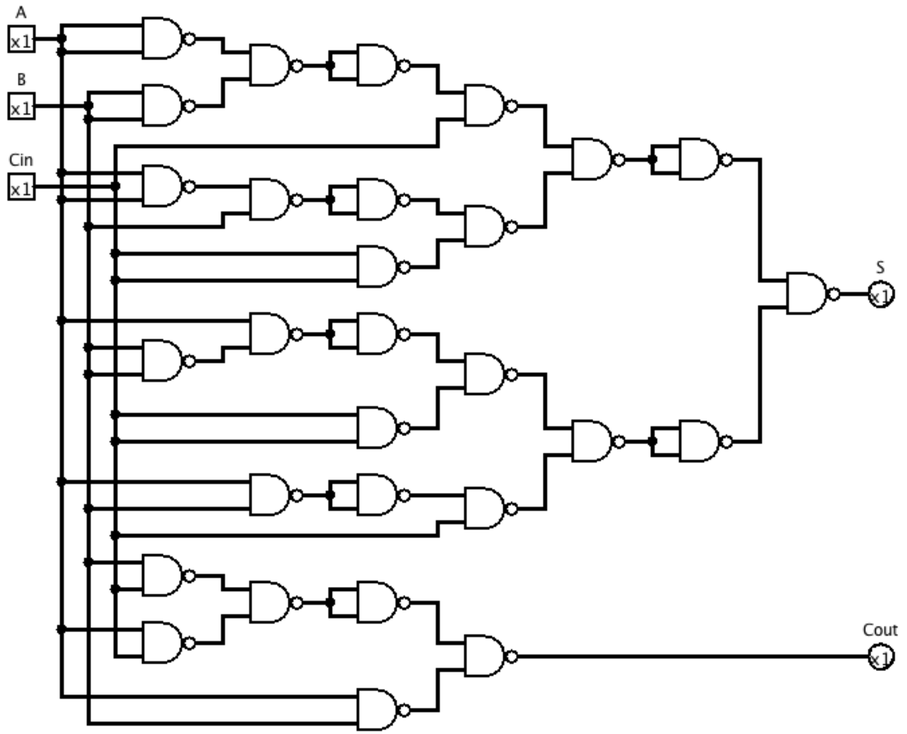
FETCH
DECODE
EXECUTE



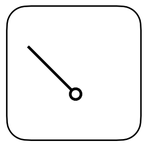
Clock generator

tick-tick-tick

INSTRUCTIONS MEMORY

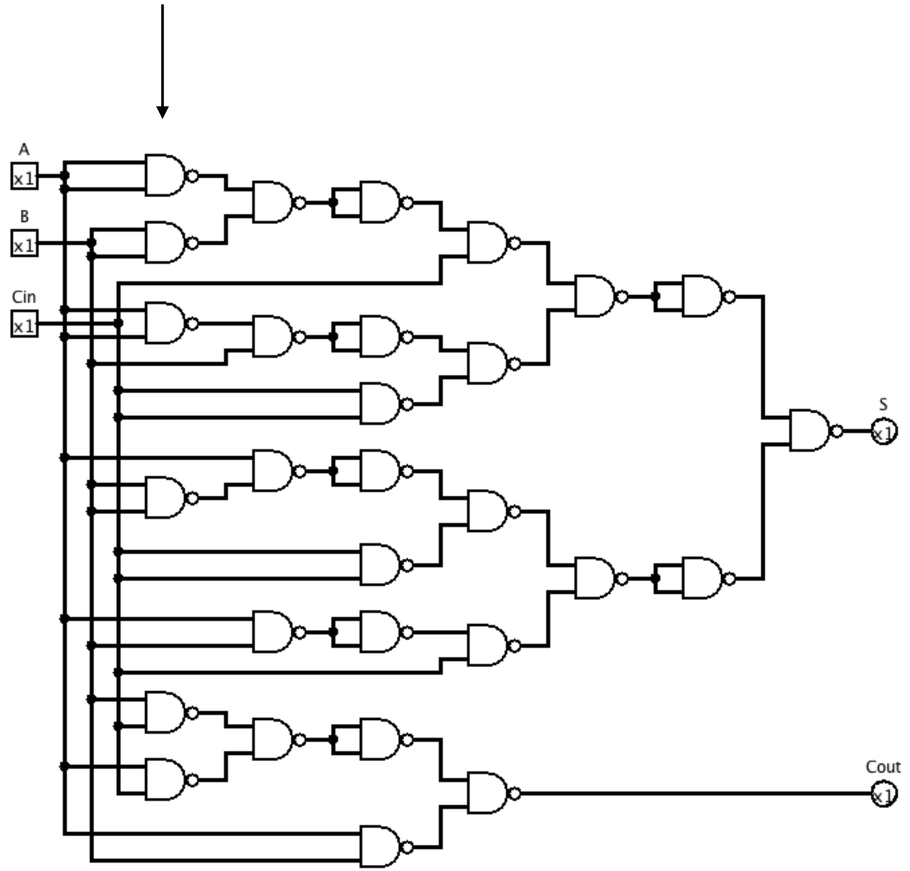


Clock generator

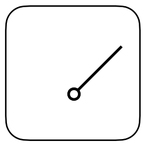


tick-tick-tick

INSTRUCTIONS MEMORY

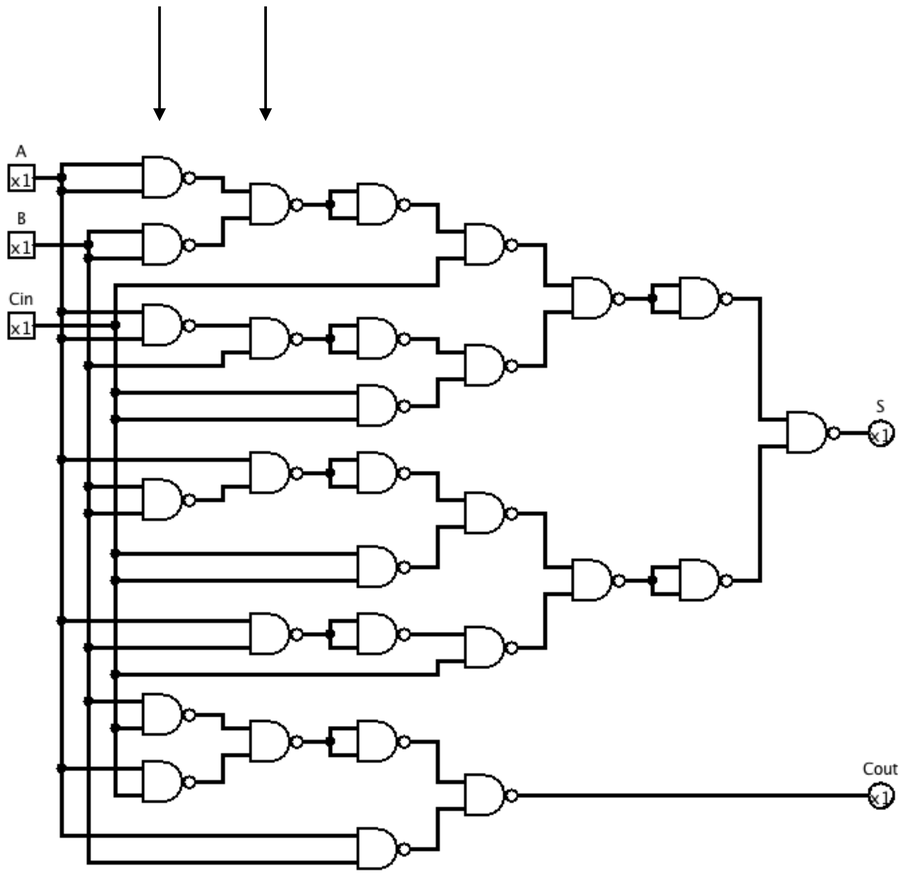


Clock generator



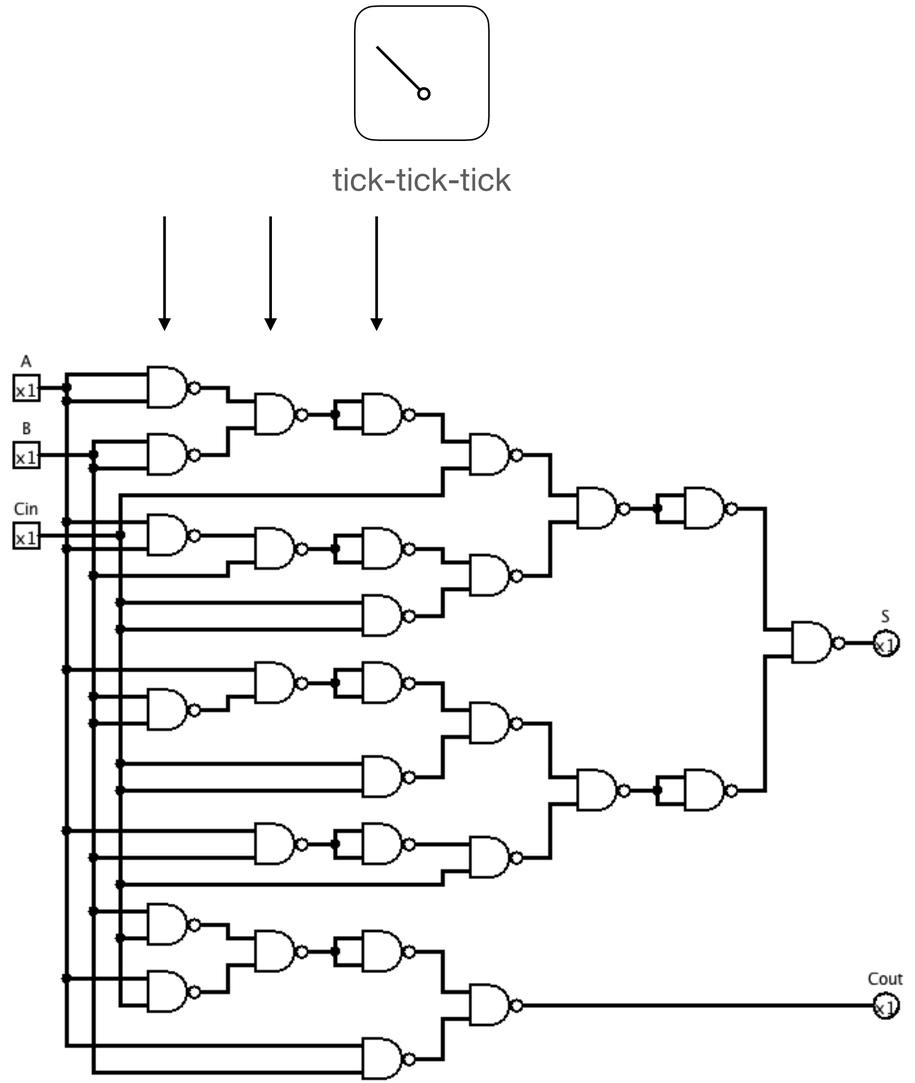
tick-tick-tick

INSTRUCTIONS MEMORY

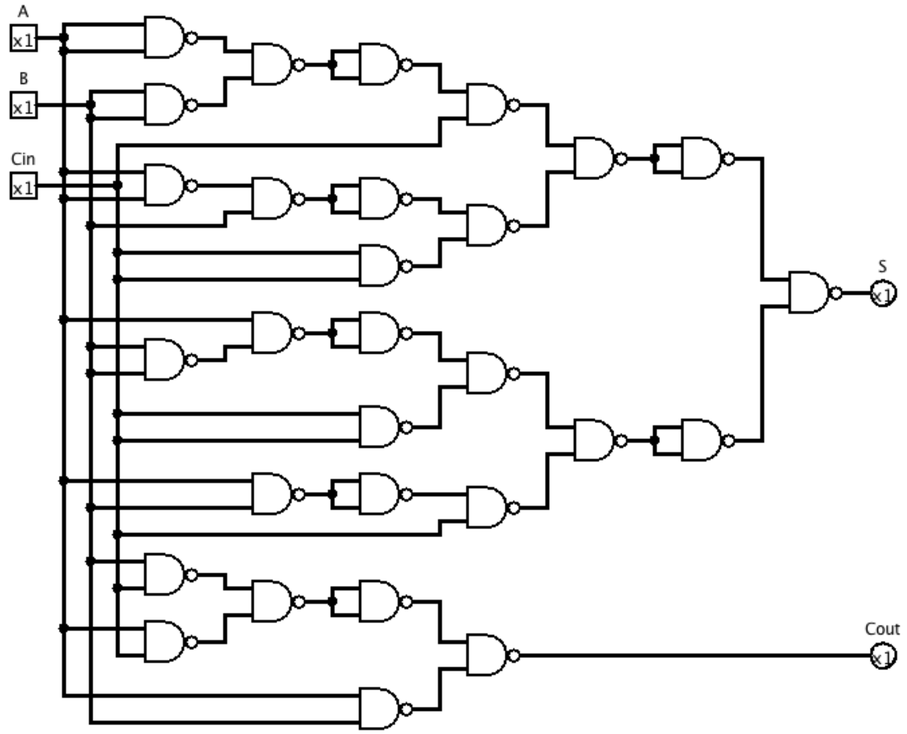


Clock generator

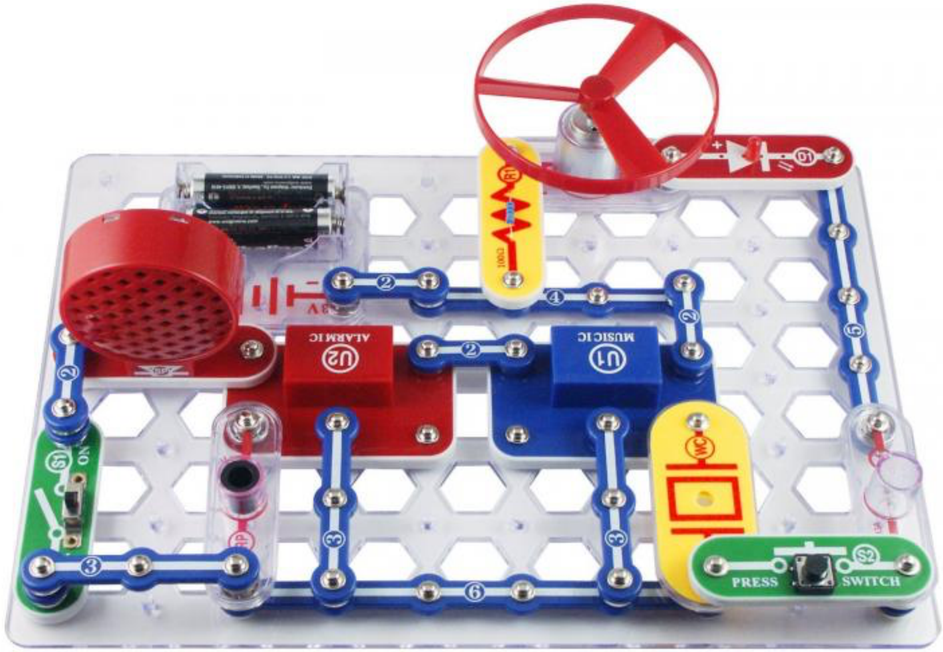
INSTRUCTIONS MEMORY



(let’s say) everything is a circuit



==



What computer does?

How computer divides?

- Digit reccurence
  - Restoring
  - Non-Restoring
  - SRT

[4] Review of Basic Classes of Dividers Based on Division Algorithm

SRT-division

```
a >> 2
b = b - a
...
```

Table with dividend and divisor

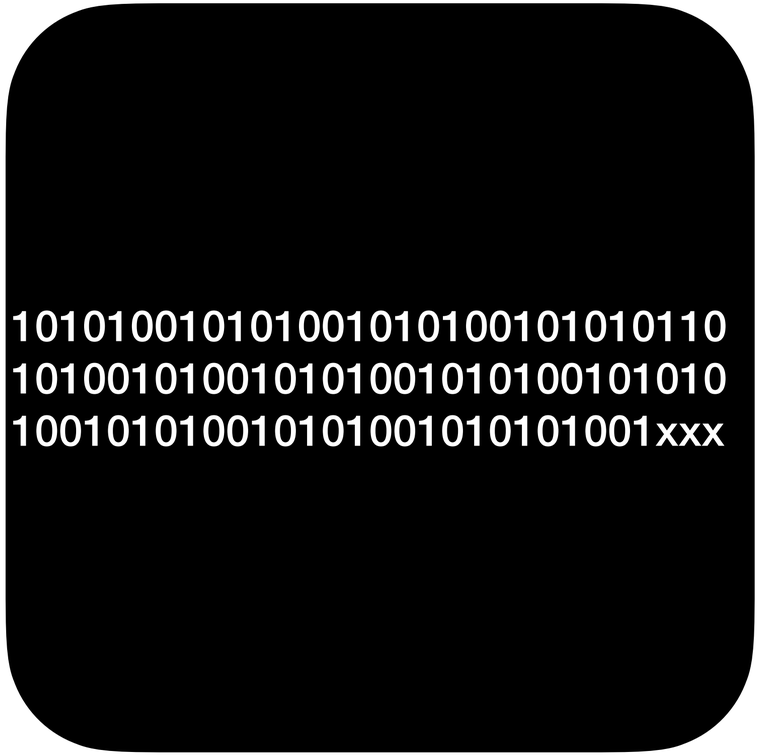
The best description of SRT-division

SRT-division

(funny stuff)

4195835.0/3145727.0 = 1.333 820 449 136 241 002 ✅

4195835.0/3145727.0 = 1.333 739 068 902 037 589 ❌



Lost 5 numbers of 1066

Table for SRT

[5] Vaughan Pratt, Anatomy of the Pentium Bug, 1995

Okay, any other division methods?

Sometimes, without division

- Core can doesn’t have a division block
- Multiplication is opposite for division
- Compiler optimisations

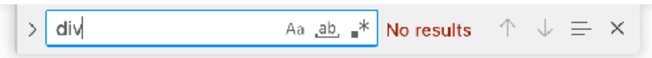
[6] Why are computers horrible at division? https://www.youtube.com/watch?v=ssDBqQ5f5_0

How computer divides?

Compiler optimisation

preall_mult_opt.asm

```
    b.ne    LBB2_5
; %bb.6:
    mov w8, #45613
    movk    w8, #61341, lsl #16
    smull   x8, w22, w8
    lsr x9, x8, #63
    asr x8, x8, #38
    add w20, w8, w9
    add x0, sp, #16
    mov x1, #0
    bl  _gettimeofday
    ldr x8, [sp, #16]
    ldrsw   x9, [sp, #24]
    mov w10, #19923
    movk    w10, #4194, lsl #16
    mul x9, x9, x10
    lsr x10, x9, #63
    asr x9, x9, #38
    sub x8, x8, x21
    mov w11, #1000
    mul x8, x8, x11
    add w9, w9, w10
    add w9, w9, w20
    add x8, x8, w9, sxtw
    str x8, [sp]
```



No division!

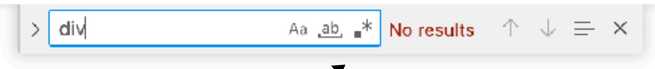
```
gcc -Os -S FizzBuzzNaivePreallocatedMult.c -o preall_mult_opt.asm
```

How computer divides?

Compiler optimisation

preall_mult_opt.asm

```
    b.ne    LBB2_5
; %bb.6:
    mov w8, #45613
    movk    w8, #61341, lsl #16
    smull   x8, w22, w8
    lsr x9, x8, #63
    asr x8, x8, #38
    add w20, w8, w9
    add x0, sp, #16
    mov x1, #0
    bl  _gettimeofday
    ldr x8, [sp, #16]
    ldrsw   x9, [sp, #24]
    mov w10, #19923
    movk    w10, #4194, lsl #16
    mul x9, x9, x10
    lsr x10, x9, #63
    asr x9, x9, #38
    sub x8, x8, x21
    mov w11, #1000
    mul x8, x8, x11
    add w9, w9, w10
    add w9, w9, w20
    add x8, x8, w9, sxtw
    str x8, [sp]
```



Multiplication

No division!

```
gcc -Os -S FizzBuzzNaivePreallocatedMult.c -o preall_mult_opt.asm
```

How computer divides?

Compiler optimisation

```
int rems3(int n) {
   unsigned r;
   static char table[62] = {0,1,2, 0,1,2, 0,1,2, 0,1,2,
       0,1,2, 0,1,2, 0,1,2, 0,1,2, 0,1,2, 0,1,2, 0,1,2,
       0,1,2, 0,1,2, 0,1,2, 0,1,2, 0,1,2, 0,1,2, 0,1,2,
       0,1,2, 0,1,2, 0,1};
    r = n;
    r = (r >> 16) + (r & 0xFFFF); // Max 0x1FFFE
    r=(r>> 8)+(r&0x00FF); //Max0x2FD
    r=(r>> 4)+(r&0x000F); //Max0x3D
    r = table[r];
    return r - (((unsigned)n >> 31) << (r & 2));
}
```

Magic numbers

[7] Hank Warren, Hacker's Delight, 2012

How computer divides?

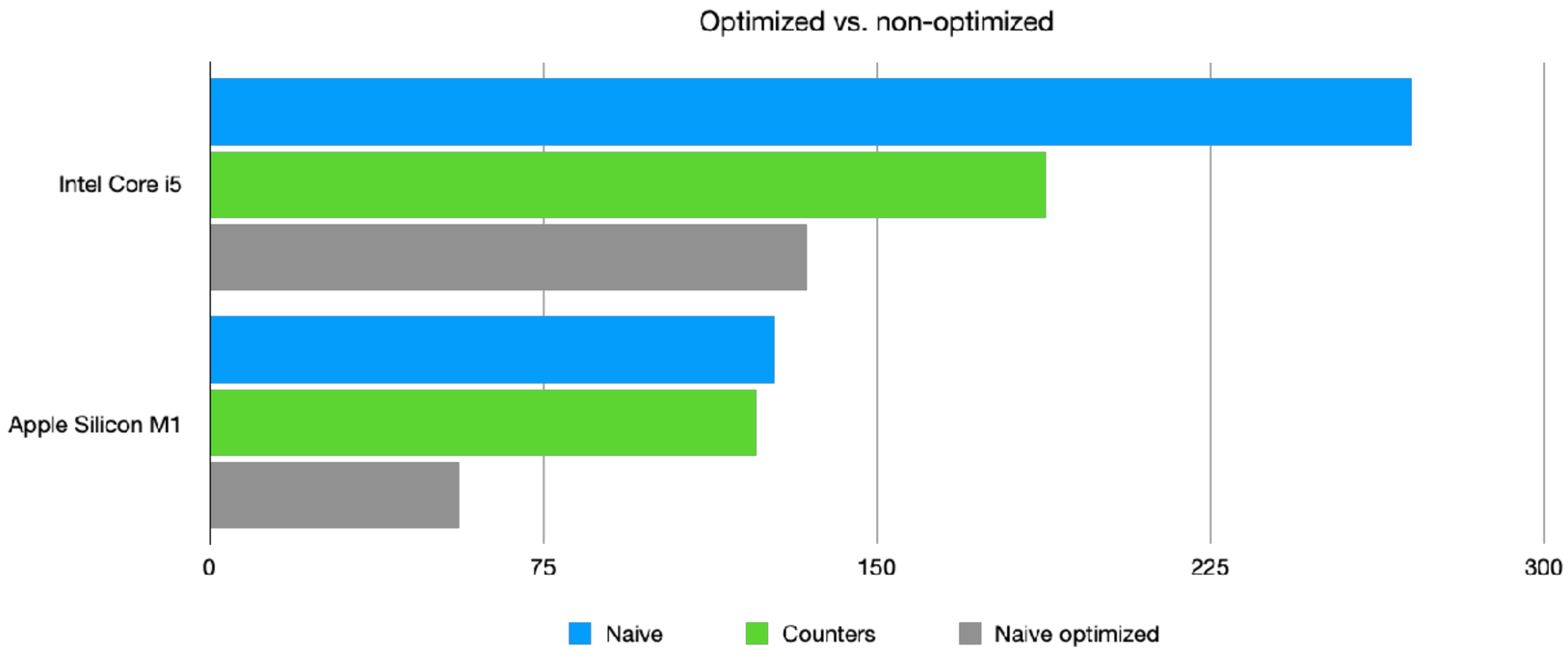
Compiler optimisation

13.2.4 Replace 128-bit Integer Division with 128-bit Multiplication

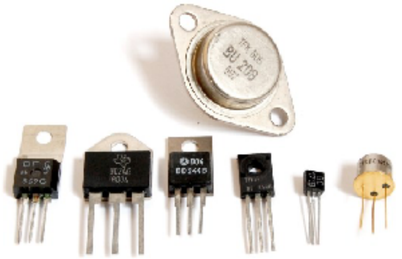
Modern compilers can transform expressions of integer division in high-level language code with a constant divisor into assembly sequences that use IMUL/MUL to replace IDIV/DIV instructions. Typically, compilers will replace a divisor value that is within the range of 32-bits if the divisor value is known at compile time. If the divisor value is not known at compile time or the divisor is greater than those represented by 32-bits, DIV or IDIV will be generated.

[8] Intel 64 and IA-32 Architectures Optimization Reference Manual

So, does it really help?

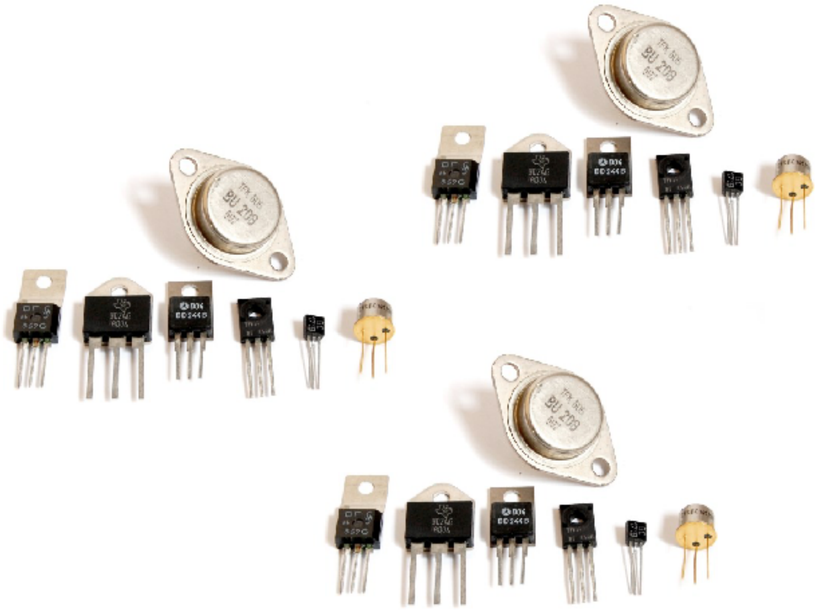


Blocks reusing



chip

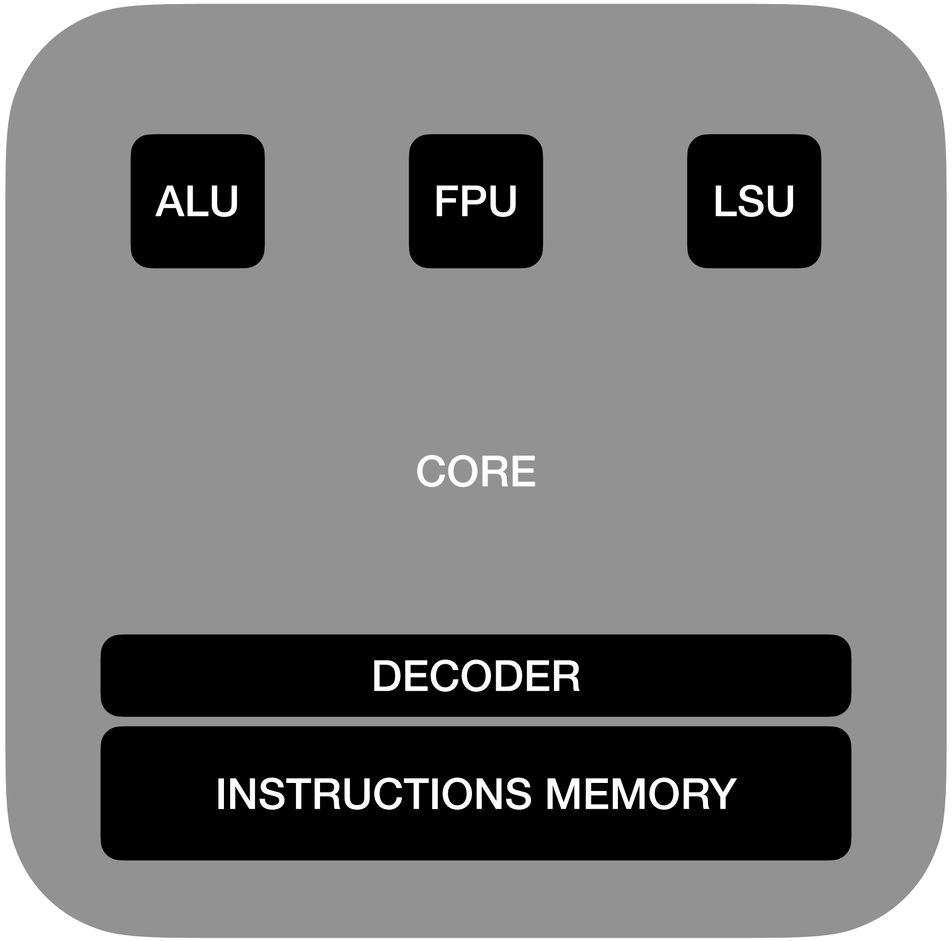
1980x



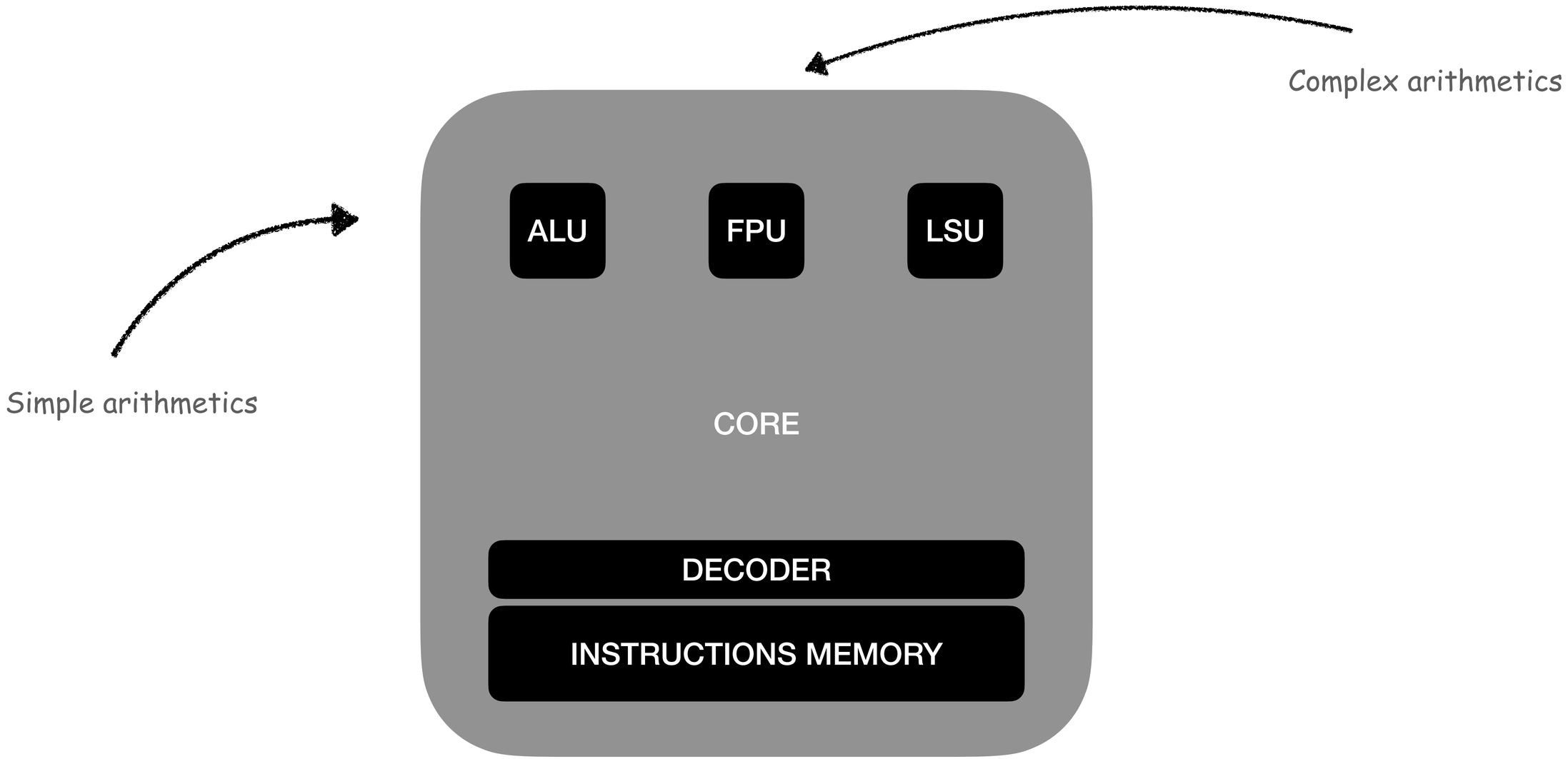
chip

2020x

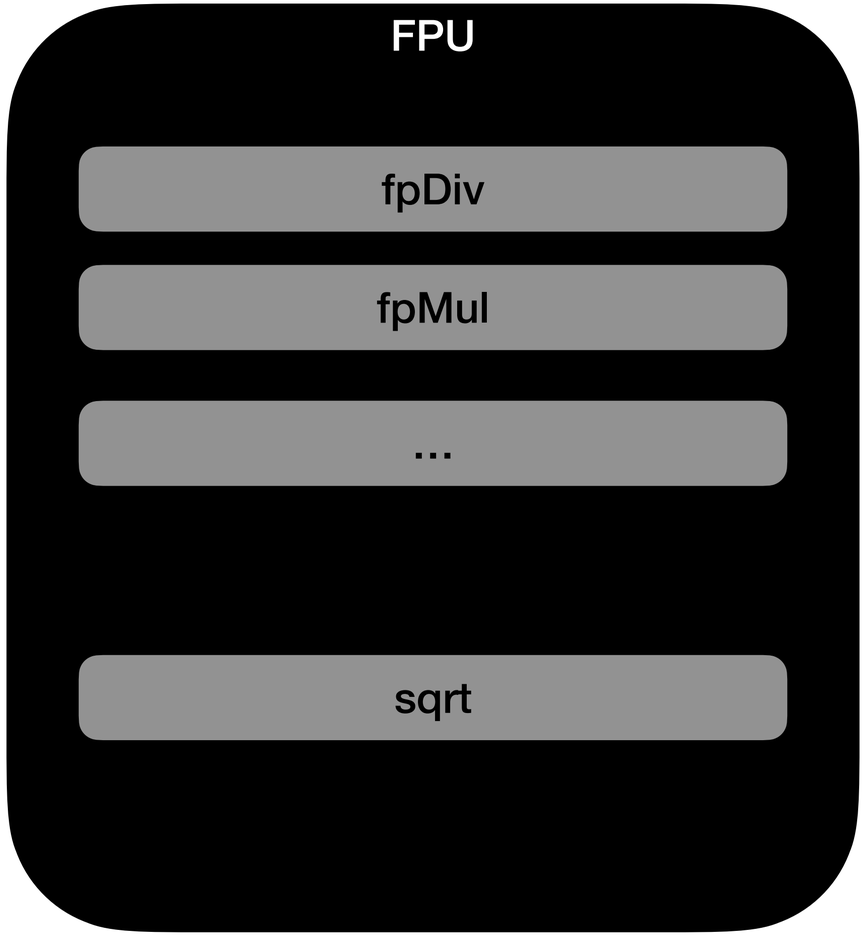
Blocks reusing



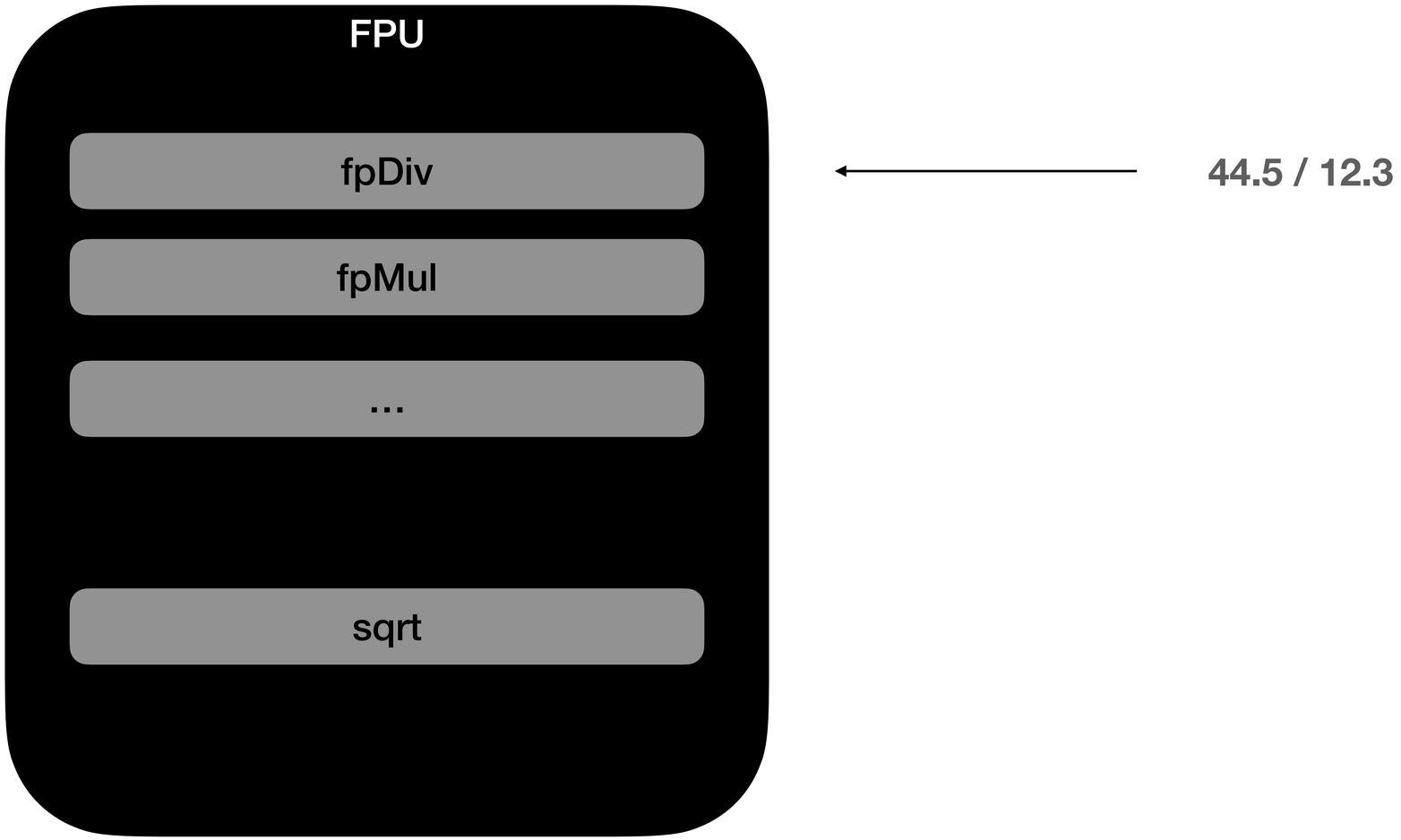
Blocks reusing



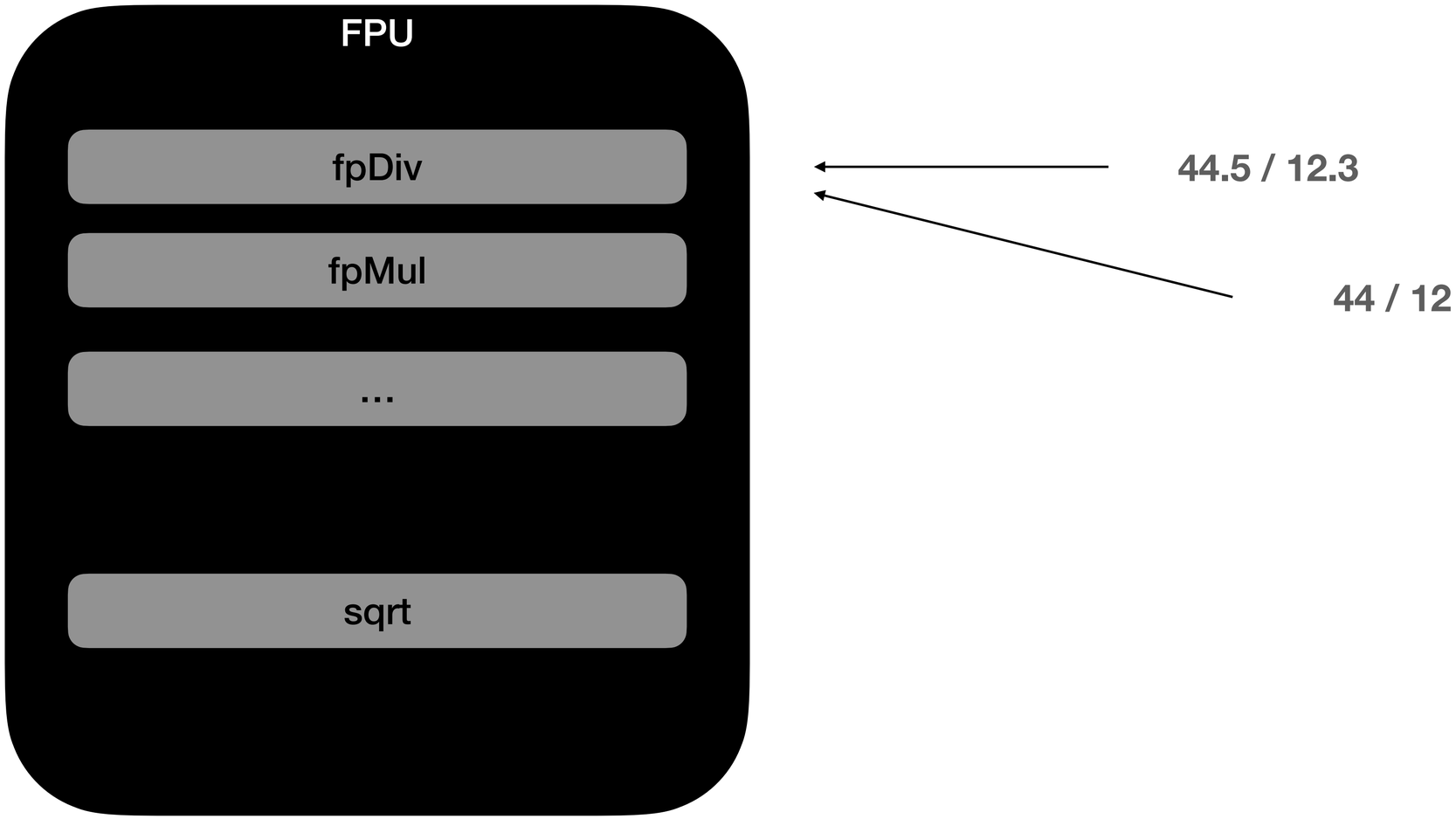
Blocks reusing



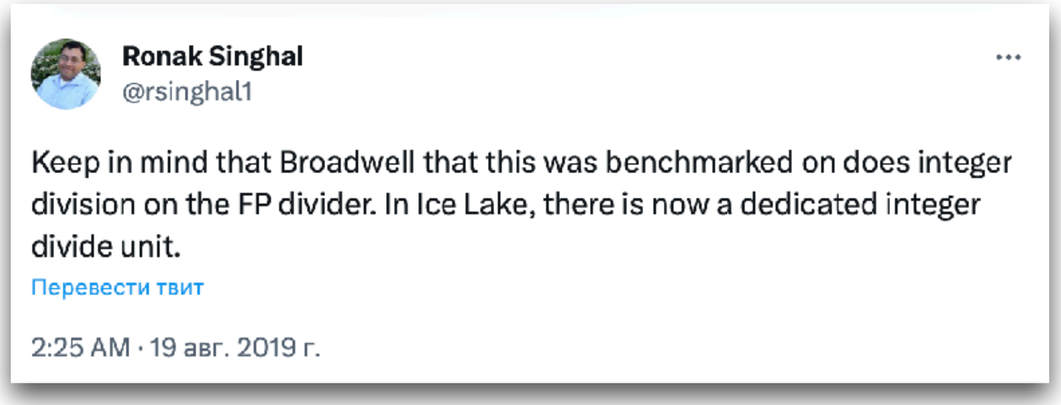
Blocks reusing



Blocks reusing



Blocks reusing



[8] Intel 64 and IA-32 Architectures Optimization Reference Manual
[9] https://twitter.com/rsinghal1/status/1163230580454019072

Blocks reusing

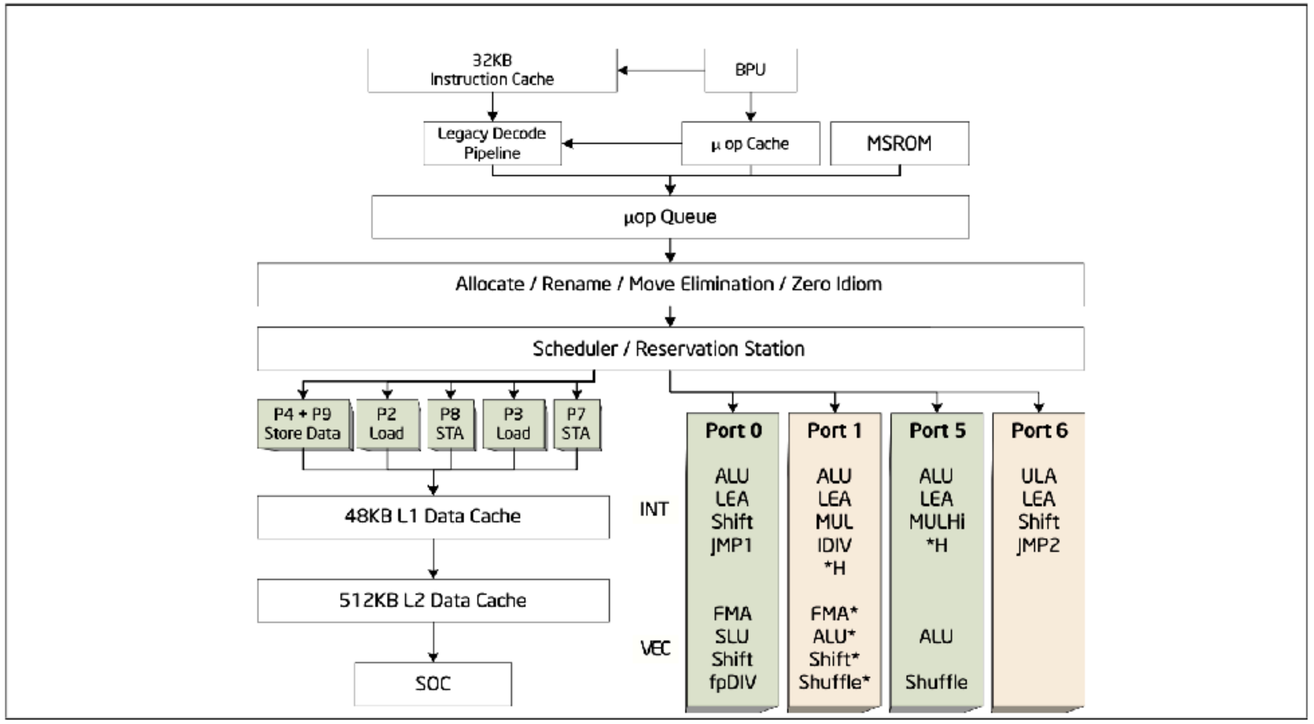


Figure 2-3. Processor Core Pipeline Functionality of the Ice Lake Client Microarchitecture[1]

[8] Intel 64 and IA-32 Architectures Optimization Reference Manual
[9] https://twitter.com/rsinghal1/status/1163230580454019072

Blocks reusing

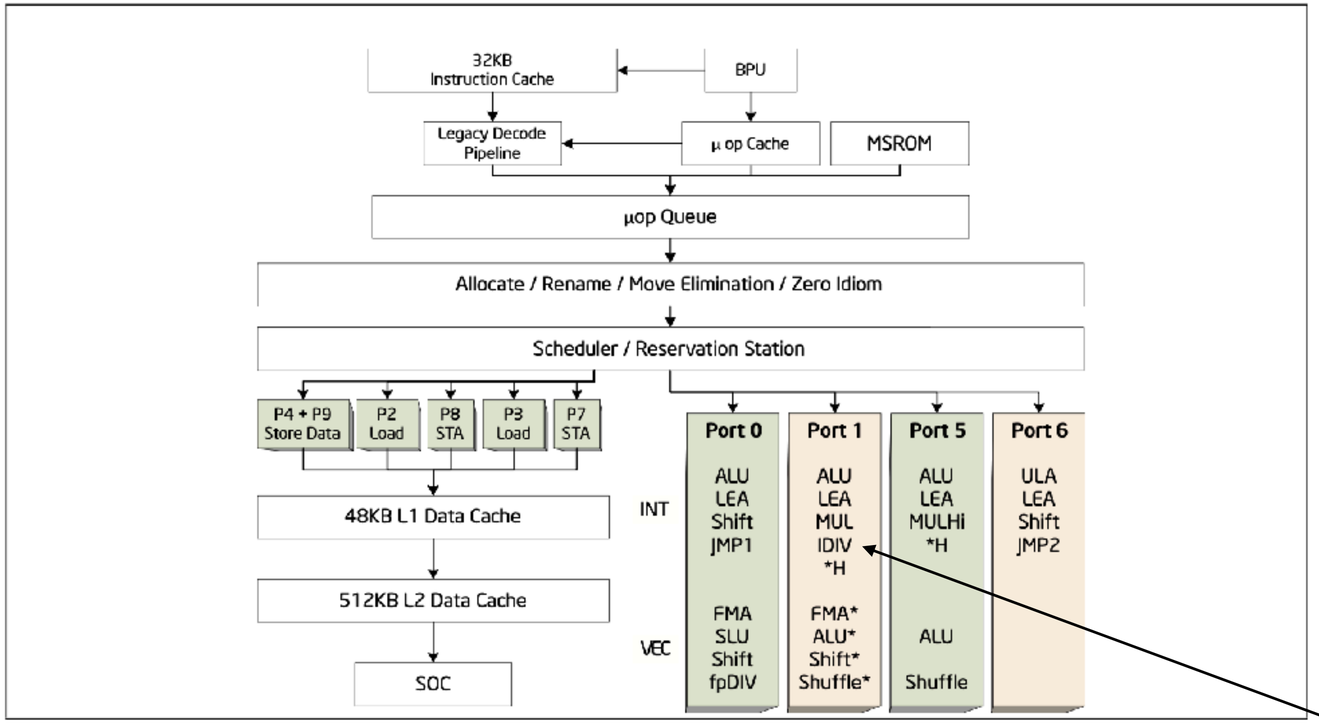


Figure 2-3. Processor Core Pipeline Functionality of the Ice Lake Client Microarchitecture[1]

IDIV

[8] Intel 64 and IA-32 Architectures Optimization Reference Manual
[9] https://twitter.com/rsinghal1/status/1163230580454019072

Well, what's the difference?

Well, what’s the difference?

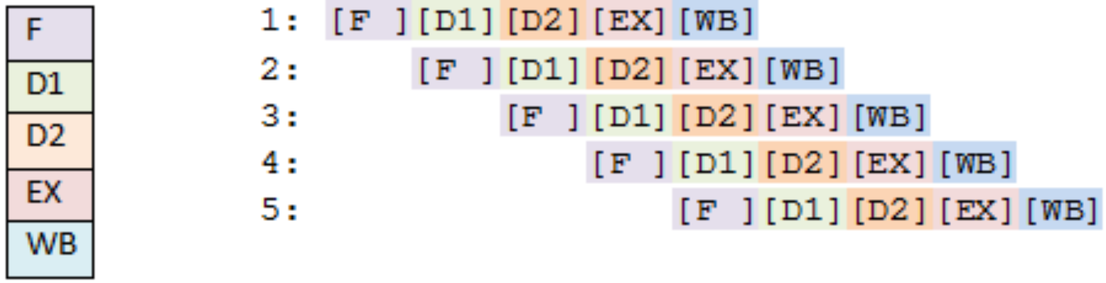
- Ticks quantity for instructions execution
- Conveyor and parallelism
- The problem is SIMPLE

Ticks quantity

- Apple Silicon M1 – 7 ticks (?)
- Intel – 14-47 ticks

[10] https://dougallj.github.io/applecpu/icestorm-int.html
[8] Intel 64 and IA-32 Architectures Optimization Reference Manual

Conveyor and parallelism



Ping-ping-ping one by one

Conveyor and parallelism

width	hardware	libdivide scalar	libdivide SSE2	libdivide AVX2	libdivide AVX512
u32	2 298	0.872 (-62%)	0.355 (-84%)	0.219 (-90%)	0.210 (-91%)
u64	6 998	0.891 (-87%)	1.058 (-85%)	0.574 (-92%)	0.492 (-93%)

Intel Xeon 3.0 GHz (8275CL)

width	hardware	libdivide scalar	libdivide NEON
u32	624	0.351 (-43%)	0.158 (-75%)
u64	623	0.315 (-49%)	0.555 (-11%)

Apple M1 3.2 GHz (Mac Mini)

[11] https://ridiculousfish.com/blog/posts/benchmarking-libdivide-m1-avx512.html

Conveyor and parallelism

width	hardware	libdivide scalar	libdivide SSE2	libdivide AVX2	libdivide AVX512
u32	2 298	0.872 (-62%)	0.355 (-84%)	0.219 (-90%)	0.210 (-91%)
u64	6 998	0.891 (-87%)	1.058 (-85%)	0.574 (-92%)	0.492 (-93%)

Intel Xeon 3.0 GHz (8275CL)

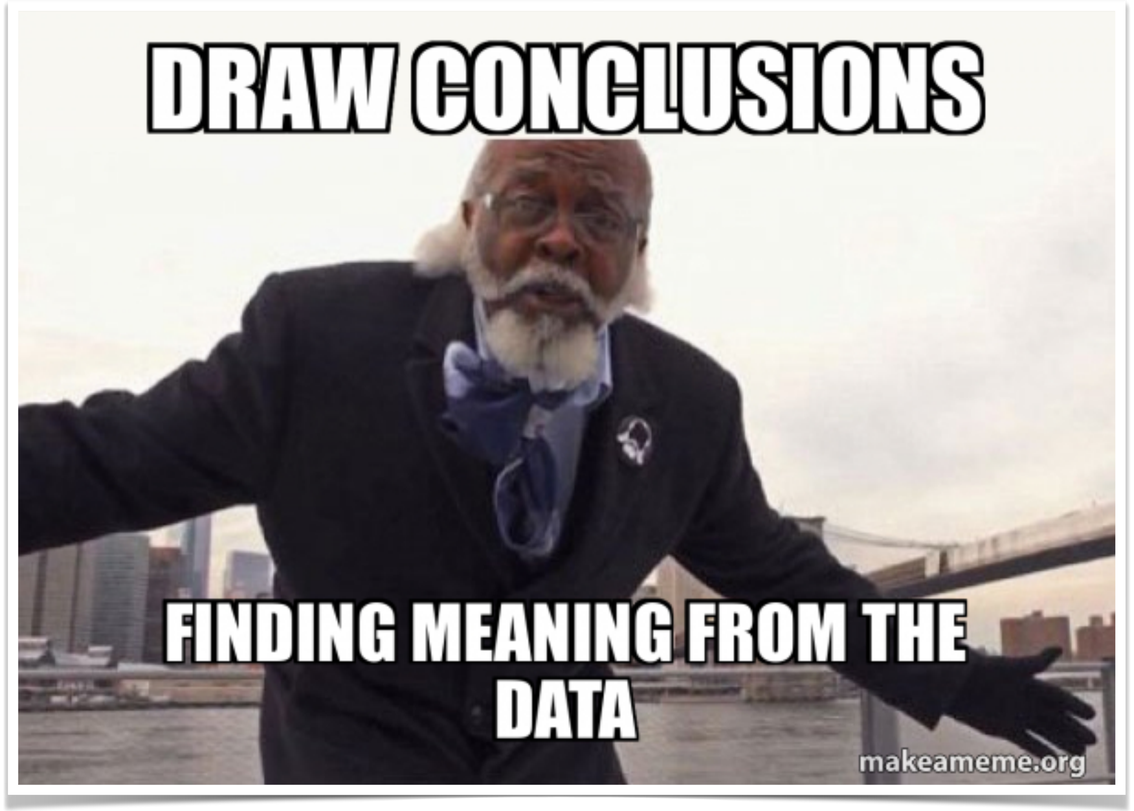
width	hardware	libdivide scalar	libdivide NEON
u32	624	0.351 (-43%)	0.158 (-75%)
u64	623	0.315 (-49%)	0.555 (-11%)

Apple M1 3.2 GHz (Mac Mini)

[11] https://ridiculousfish.com/blog/posts/benchmarking-libdivide-m1-avx512.html

The problem is SIMPLE

- 100 000 000 – 28 digits
- Complex instructions are not used (AVX?)


DRAW CONCLUSIONS
FINDING MEANING FROM THE DATA
makeameme.org

Conclusions!

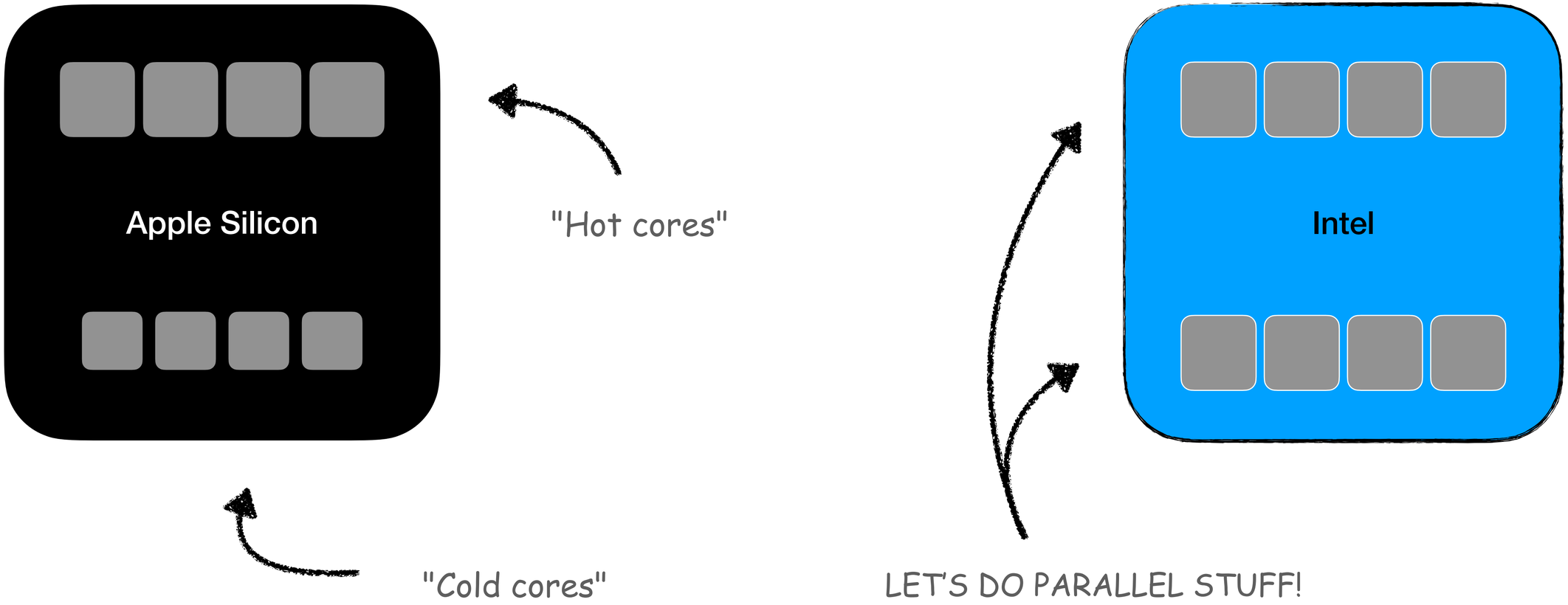
- Division is complex
- Now individual blocks in processors are being upgraded
- Math is cool, compiler optimizations are awesome
- The more general the solution, the more expensive it is

So it turns out that Intel is worse?

LMAO, of course not!

- Not so energy efficient
- HyperThreading, AVX and many other features

Different purposes



Easier example

width	hardware	libdivide scalar	libdivide SSE2	libdivide AVX2	libdivide AVX512
u32	2 298	0.872 (-62%)	0.355 (-84%)	0.219 (-90%)	0.210 (-91%)
u64	6 998	0.891 (-87%)	1.058 (-85%)	0.574 (-92%)	0.492 (-93%)

Intel Xeon 3.0 GHz (8275CL)

width	hardware	libdivide scalar	libdivide NEON
u32	624	0.351 (-43%)	0.158 (-75%)
u64	623	0.315 (-49%)	0.555 (-11%)

Apple M1 3.2 GHz (Mac Mini)

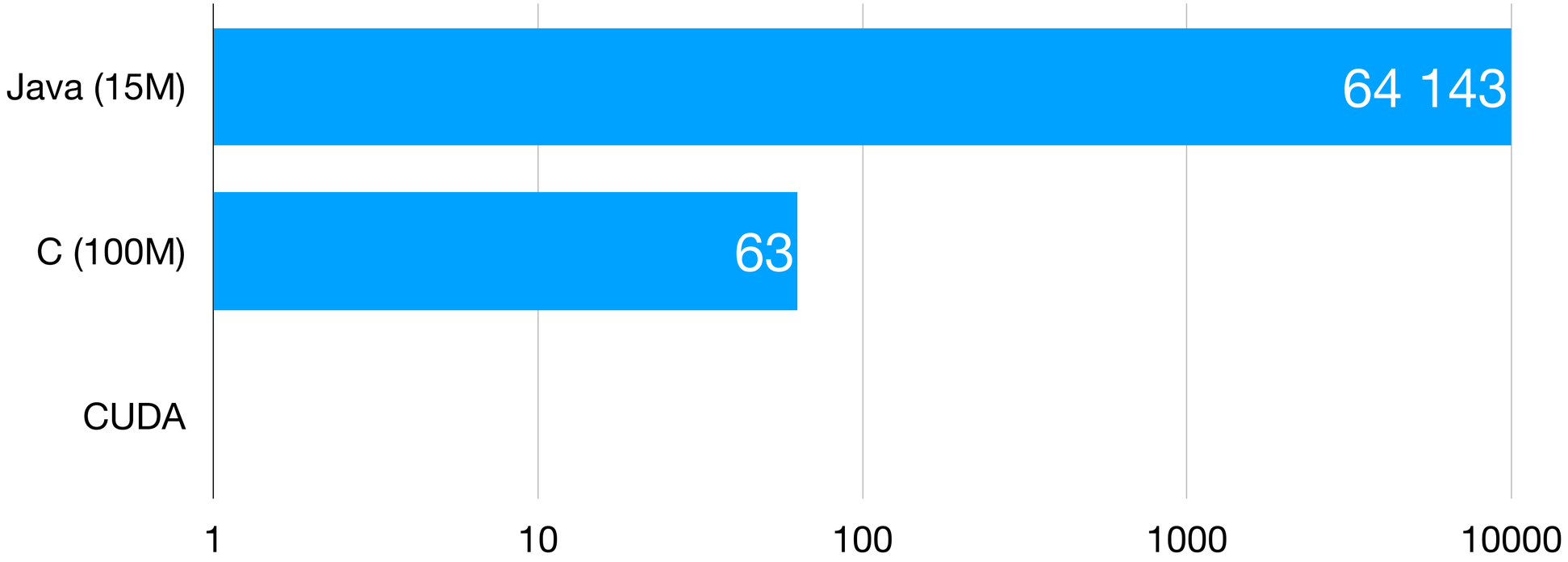
Somebody said… CUDA?

Somebody said… CUDA?

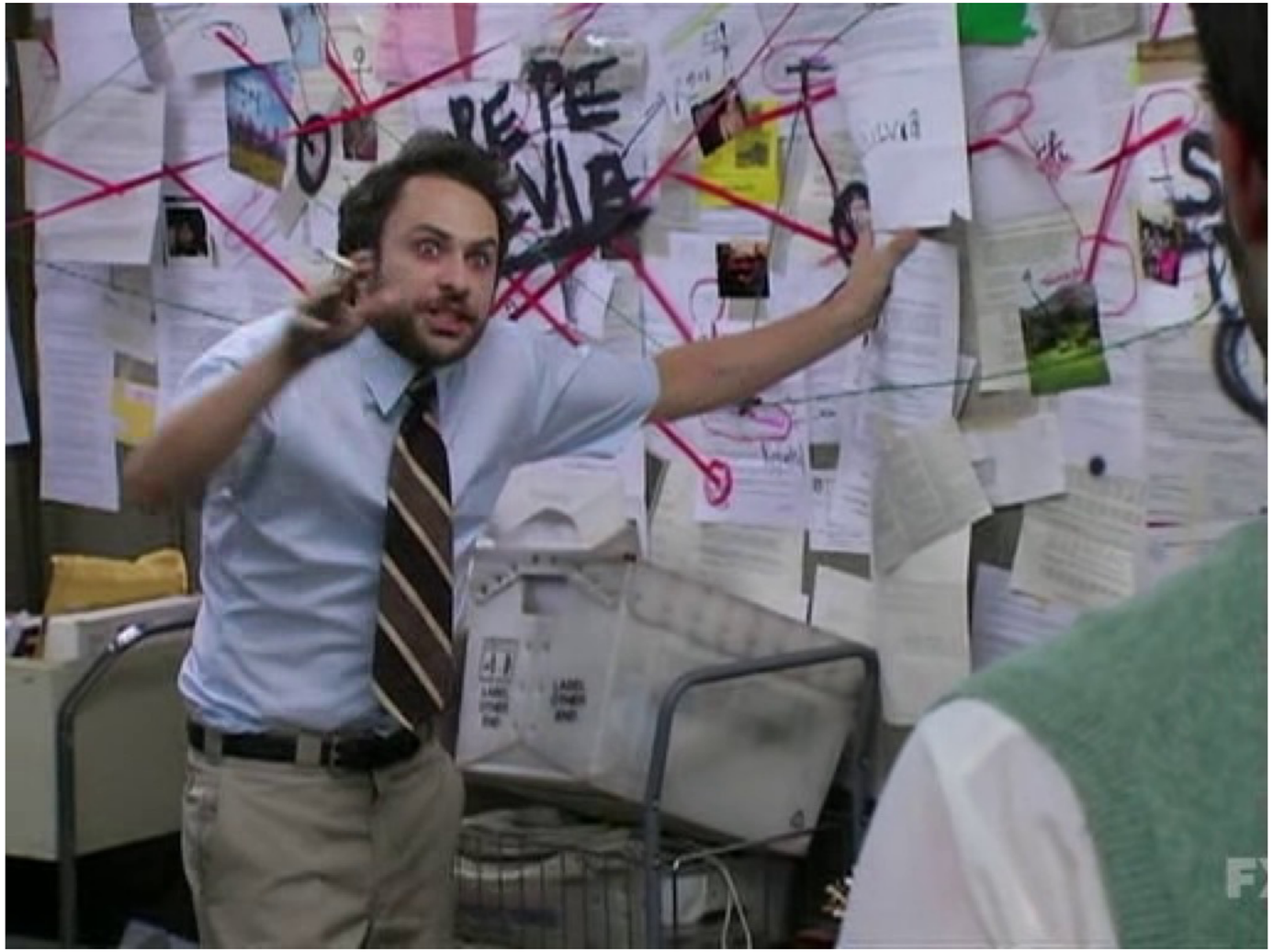
Makes math!

Makes math too!

Results


Java (15M)
C (100M)
CUDA
64 143
63
1
10
100
1000
10000
Milliseconds

Questions?



Links

https://github.com/b00blik/FizzBuzzTalk

http://b00blik.ru/pres/FizzBuzzTalk_en_FCC.pdf