

ИНФОРМАТИКА



```
~/polytech $ date
```

```
осень 2019
```

```
~/polytech $ name
```

```
константин кориков
```

```
~/polytech $ topic
```

```
бонус: программирование, it и рок-н-ролл 🎸
```

ГДЕ МЫ?



Программирование, IT и рок-н-ролл 🎸

Рок-н-ролл в широком смысле 🎸

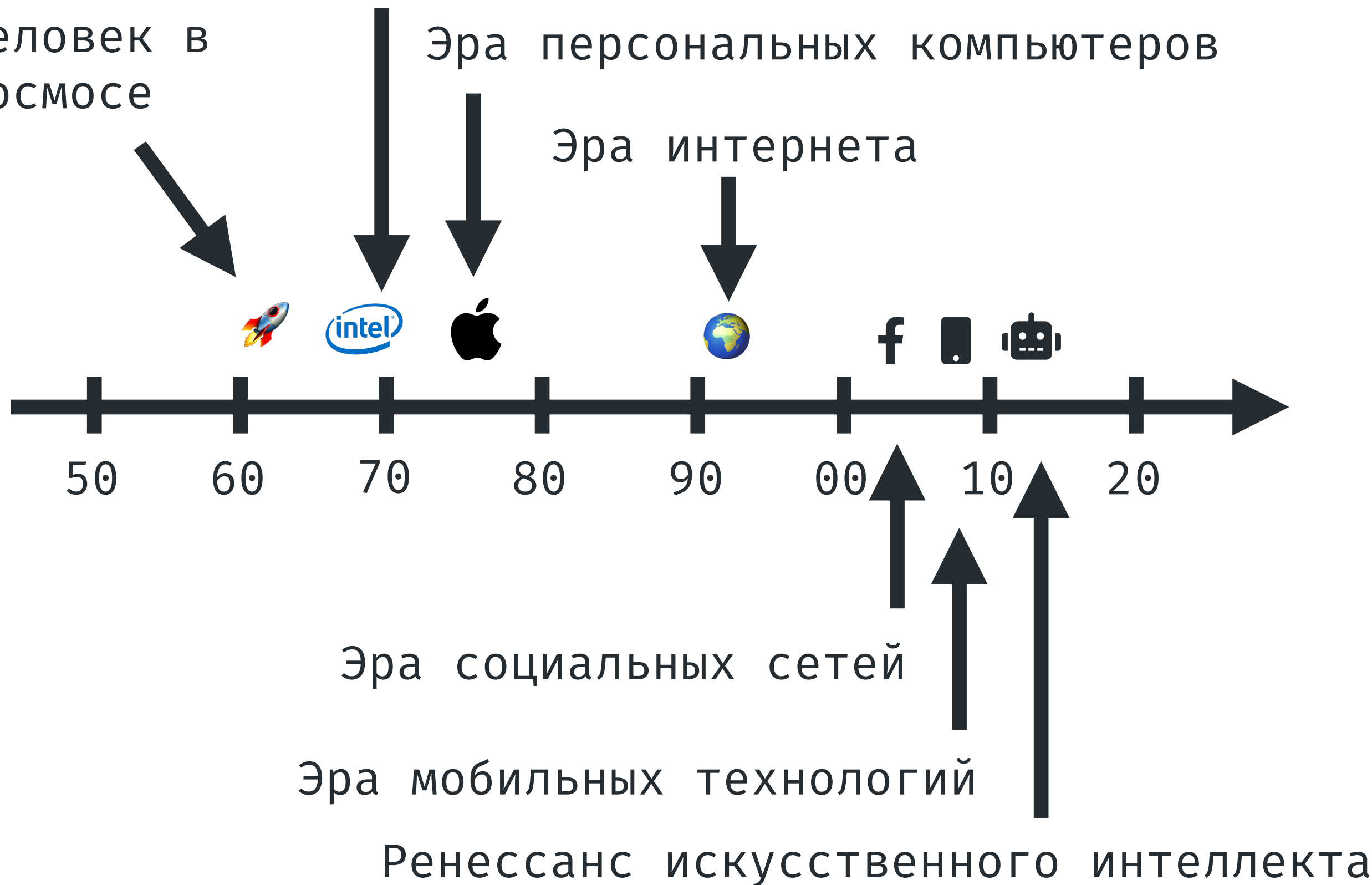


Эра микропроцессоров

Человек в
космосе

Эра персональных компьютеров

Эра интернета



Первый программист



Ада Байрон (графиня Лавлейс)

Ada[®]

Н. ДЖЕХАНИ

Язык Ада

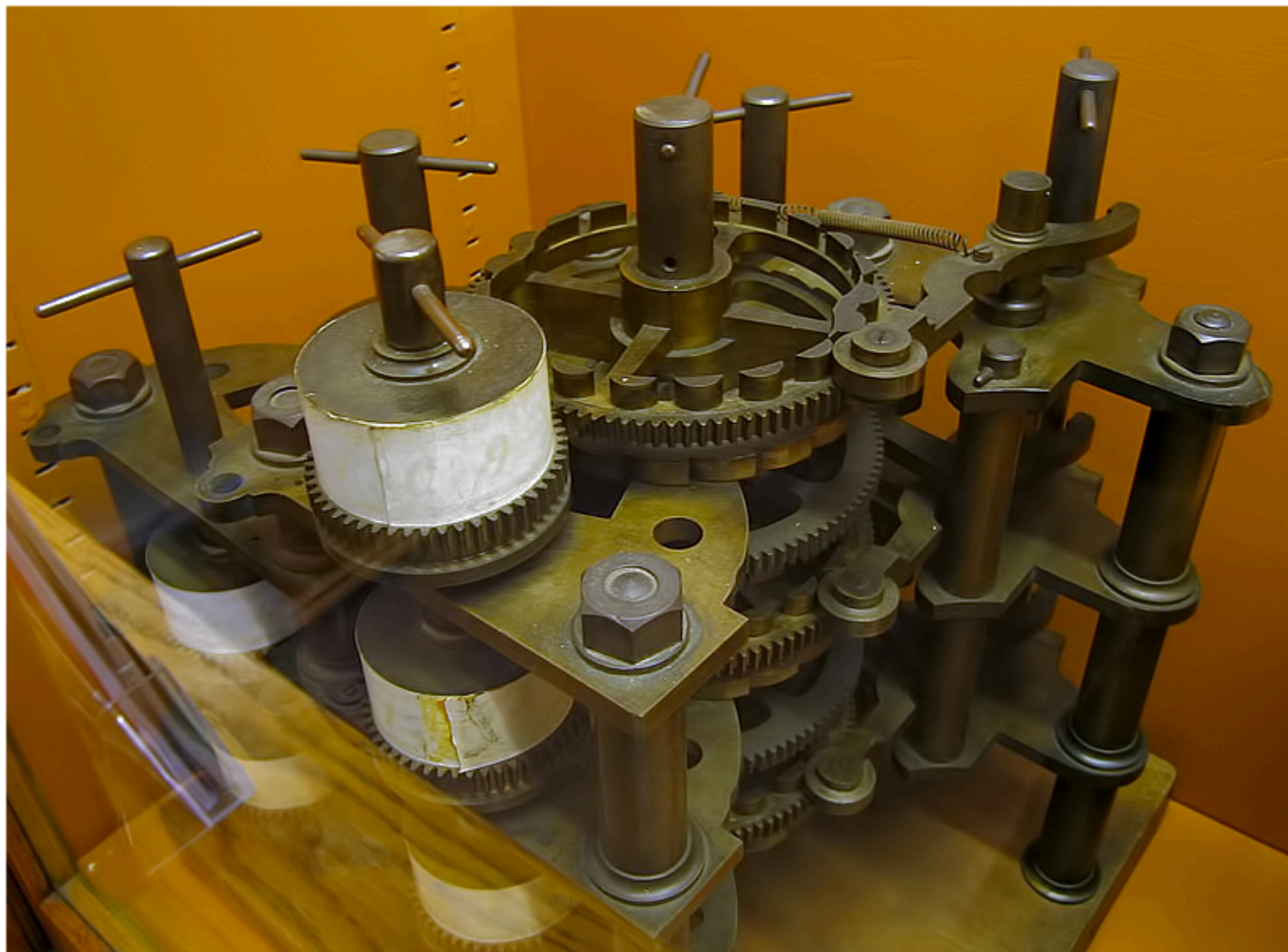
Ada[®]

Ada[®]

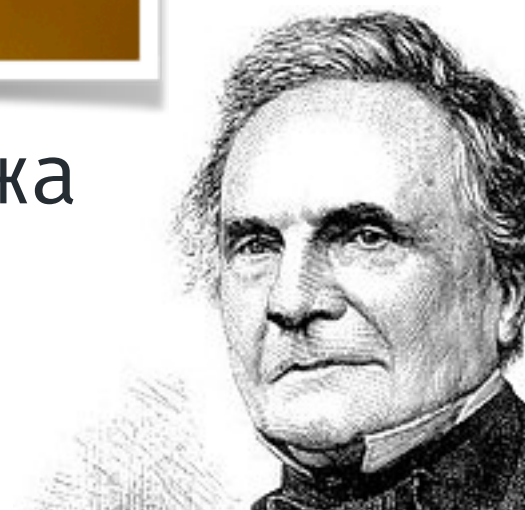
Ada[®]

ИЗДАТЕЛЬСТВО «МИР»





Часть паровой машины Чарлза Бэббиджа



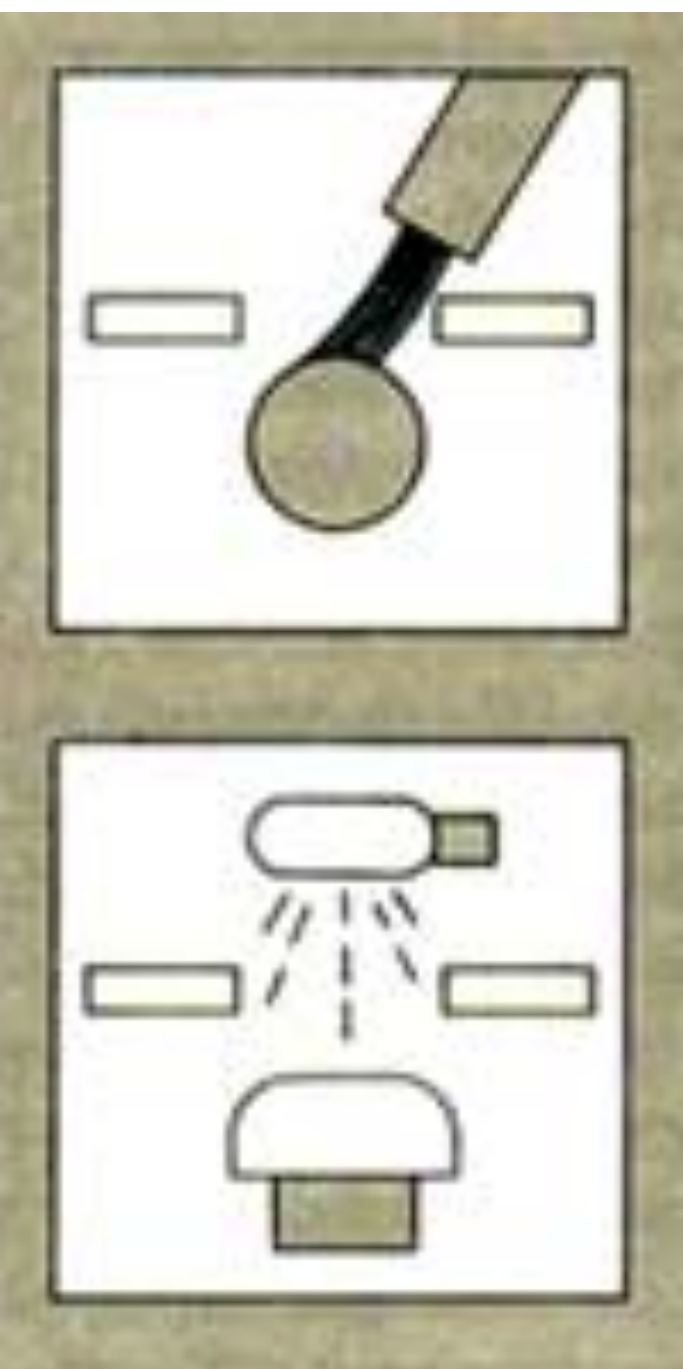


Табулятор Германа Холлерита (1890)



1	1	3	0	2	4	10	On	S	A	C	E	a	c	e	g		EB	SB	Ch	Sy	U	Sh	Hk	Br	Rm
2	2	4	1	3	E	15	Off	IS	B	D	F	b	d	f	h		SY	X	Fp	Cn	R	X	Al	Cg	Kg
3	0	0	0	0	W	20		0	0	0	0	0	0	0	0	0	●	0	0	0	0	0	0	0	0
A	1	1	1	1	0	25	A	1	1	1	1	1	1	1	1	1	1	●	1	1	1	1	1	1	1
B	2	2	2	2	5	30	B	2	2	●	2	2	2	2	2	2	2	2	●	2	2	2	2	2	2
C	3	3	3	3	0	3	C	3	3	3	●	3	3	3	3	3	3	3	3	●	3	3	3	3	3
D	4	4	4	4	1	4	D	4	4	4	4	●	4	4	4	4	4	4	4	4	●	4	4	4	4
E	5	5	5	5	2	C	E	5	5	5	5	5	●	5	5	5	5	5	5	5	5	●	5	5	5
F	6	6	6	6	A	D	F	6	6	6	6	6	6	●	6	6	6	6	6	6	6	6	●	6	6
G	7	7	7	7	B	E	G	7	7	7	7	7	7	7	●	7	7	7	7	7	7	7	7	●	7
H	8	8	8	8	a	F	H	8	8	8	8	8	8	8	8	●	8	8	8	8	8	8	8	8	●
I	9	9	9	9	b	c	I	9	9	9	9	9	9	9	9	9	●	9	9	9	9	9	9	9	9

Перфокарта



1	1	3	0	2	4	10	On	S	A	C	E	a	c	e	g		EB	SB	Ch	Sy	U	Sh	Hk	Br	Rm
2	2	4	1	3	E	15	Off	IS	B	D	F	b	d	f	h		SY	X	Fp	Cn	R	X	Al	Cg	Kg
3	0	0	0	0	W	20		0	0	0	0	0	0	0	0	0	●	0	0	0	0	0	0	0	0
A	1	1	1	1	0	25	A	1	■	1	1	1	1	1	1	1	1	●	1	1	1	1	1	1	1
B	2	2	2	2	5	30	B	2	■	2	2	2	2	2	2	2	2	●	2	2	2	2	2	2	2
C	3	3	3	3	0	3	C	3	3	3	●	3	3	3	3	3	3	3	3	●	3	3	3	3	3
D	4	4	4	4	1	4	D	4	4	4	●	4	4	4	4	4	4	4	4	●	4	4	4	4	4
E	5	5	5	5	2		C	E	5	5	5	5	●	5	5	5	5	5	5	5	5	●	5	5	5
F	6	6	6	6	A	D	F	6	6	6	6	6	6	●	6	6	6	6	6	6	6	6	●	6	6
G	7	7	7	7	B	E	G	7	7	7	7	7	7	7	●	7	7	7	7	7	7	7	7	●	7
H	8	8	8	8	a	F	H	8	8	8	8	8	8	8	8	●	8	8	8	8	8	8	8	8	●
I	9	9	9	9	b	c	I	9	9	9	9	9	9	9	9	9	●	9	9	9	9	9	9	9	9

Заплата (англ. patch) = патч

#42

30.08.2004 в 16:00

<Sashok> -- Здравствуйте, это канал об аниме? -- Да. -- Как мне пропатчить KDE2 под FreeBSD?

Комикс по мотивам цитаты



[:::]

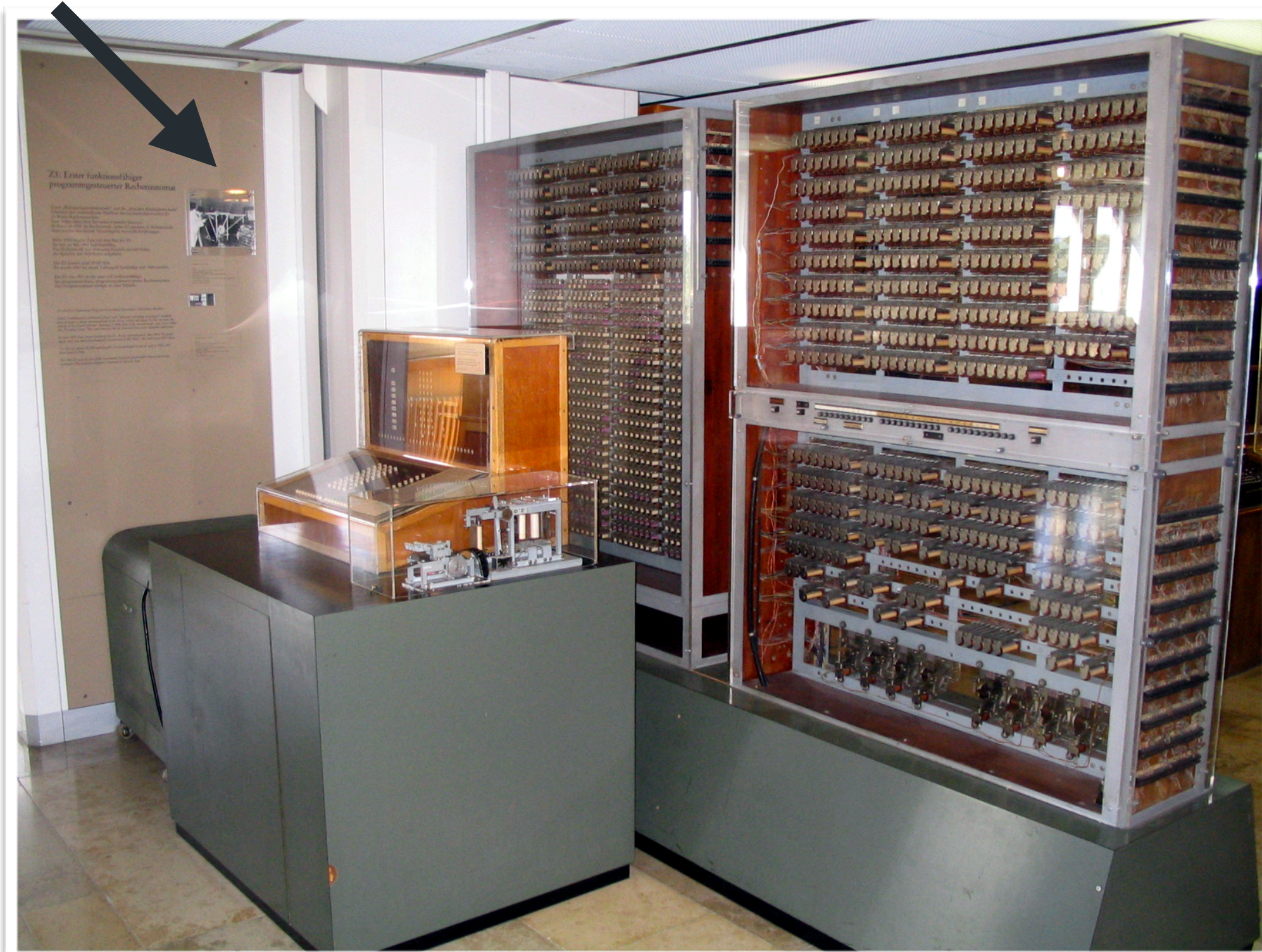
[-] 125779 [+]



THE TABULATING MACHINE COMPANY
GENERAL OFFICES - 270 BROADWAY
NEW YORK, N. Y.

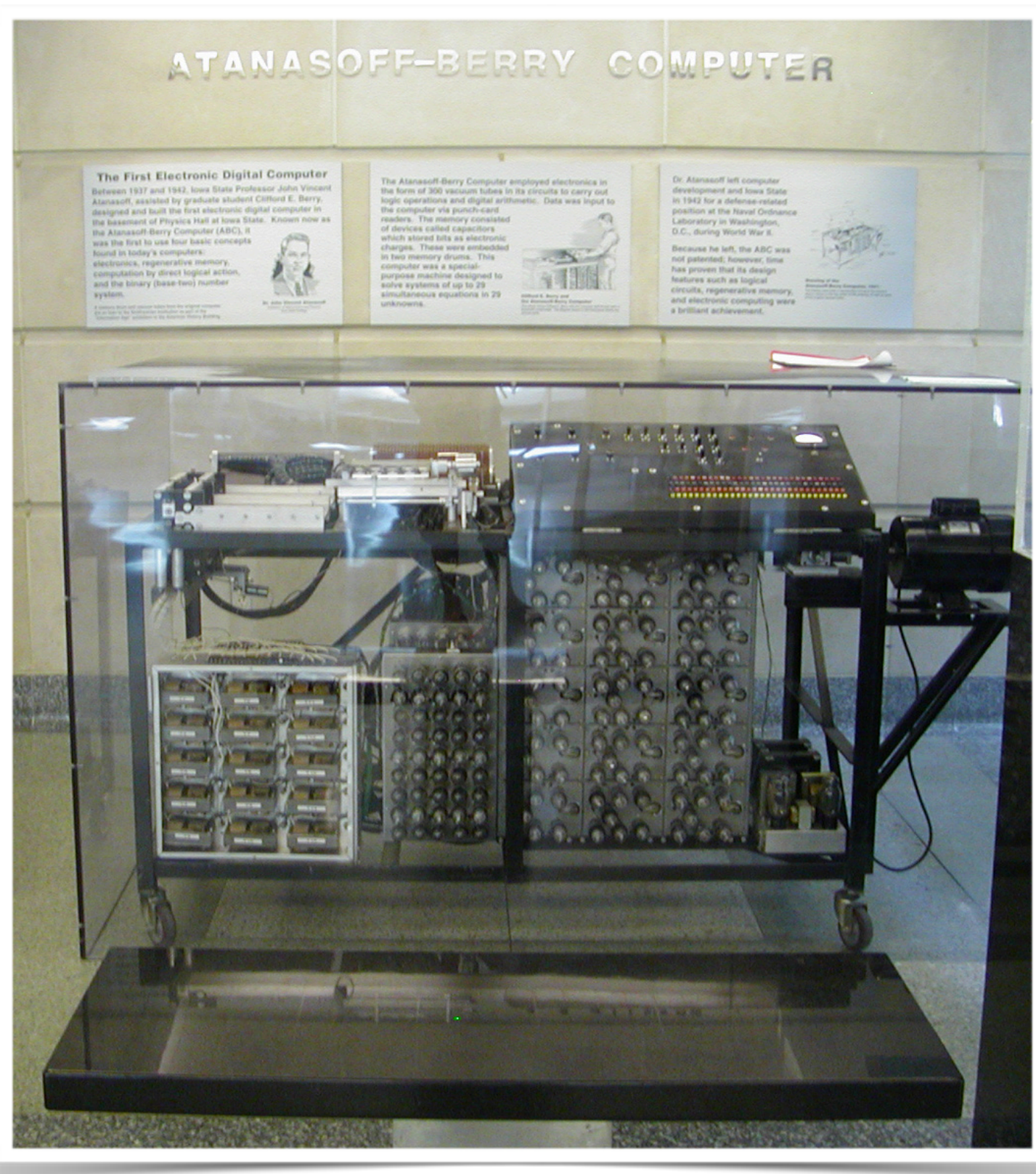


Автор: Конрад Цузе



Z3 (1941)



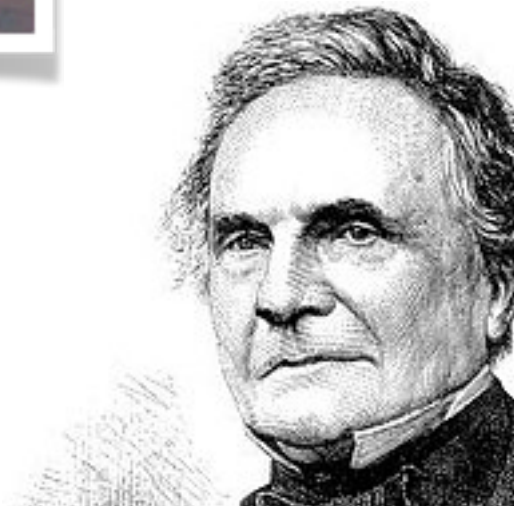


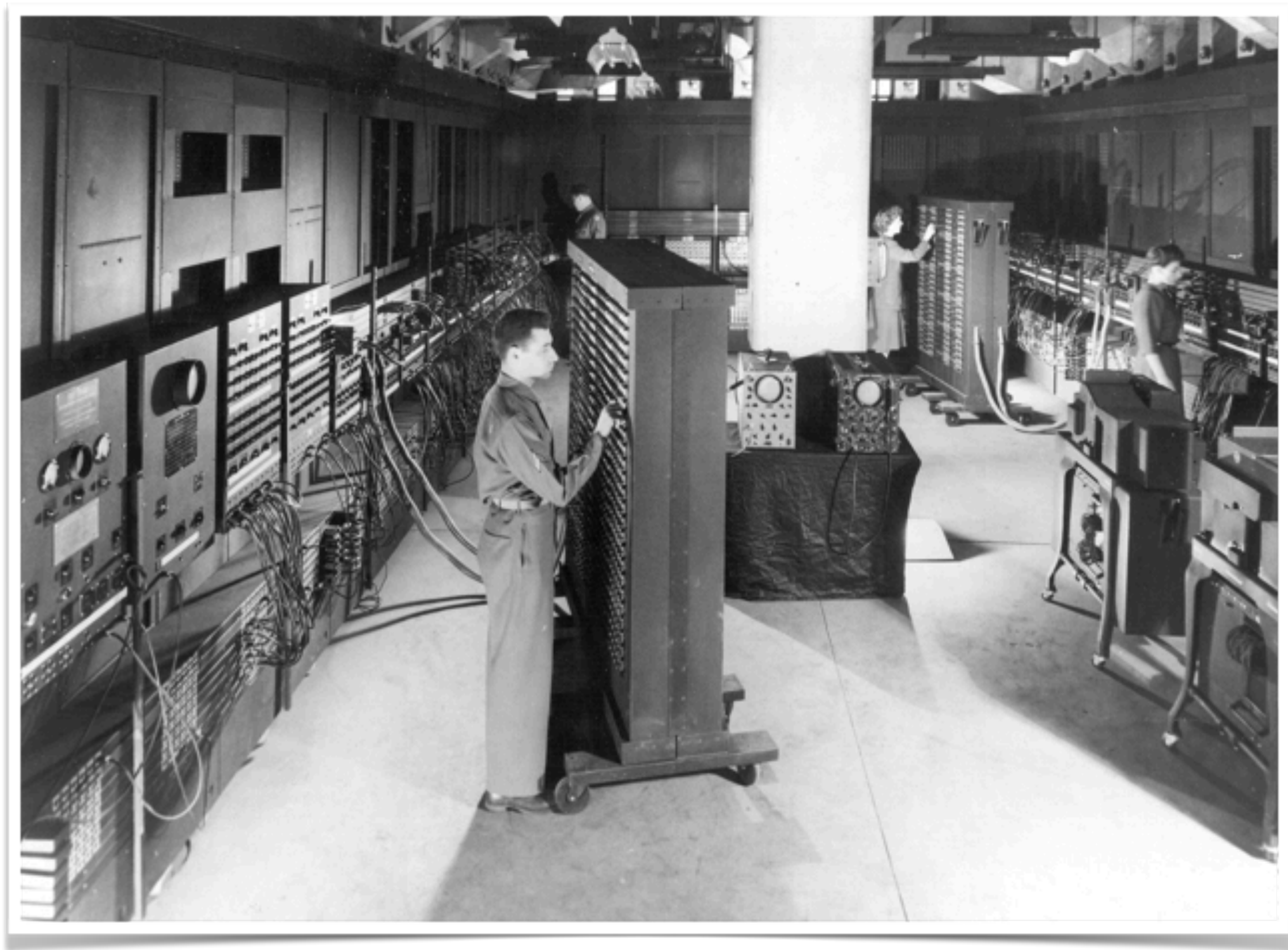
ABC (1944)





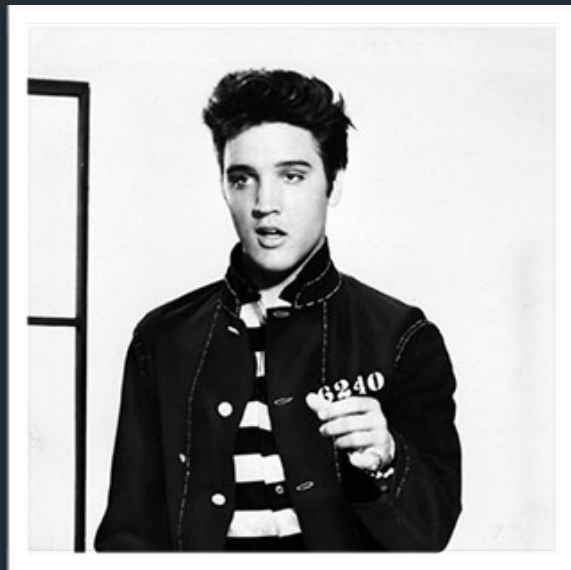
MARK I (1944)





ENIAC (1945)

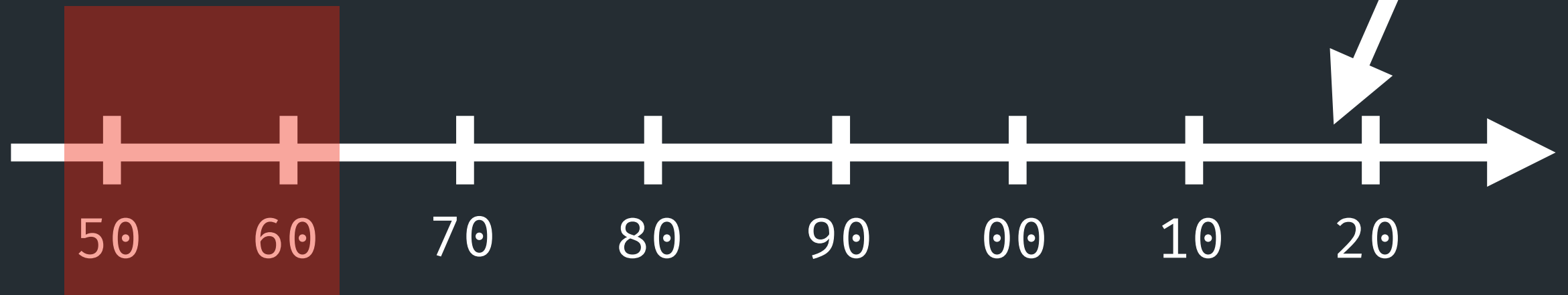




Элвис Пресли



Чак Берри



Мадди Уотерс



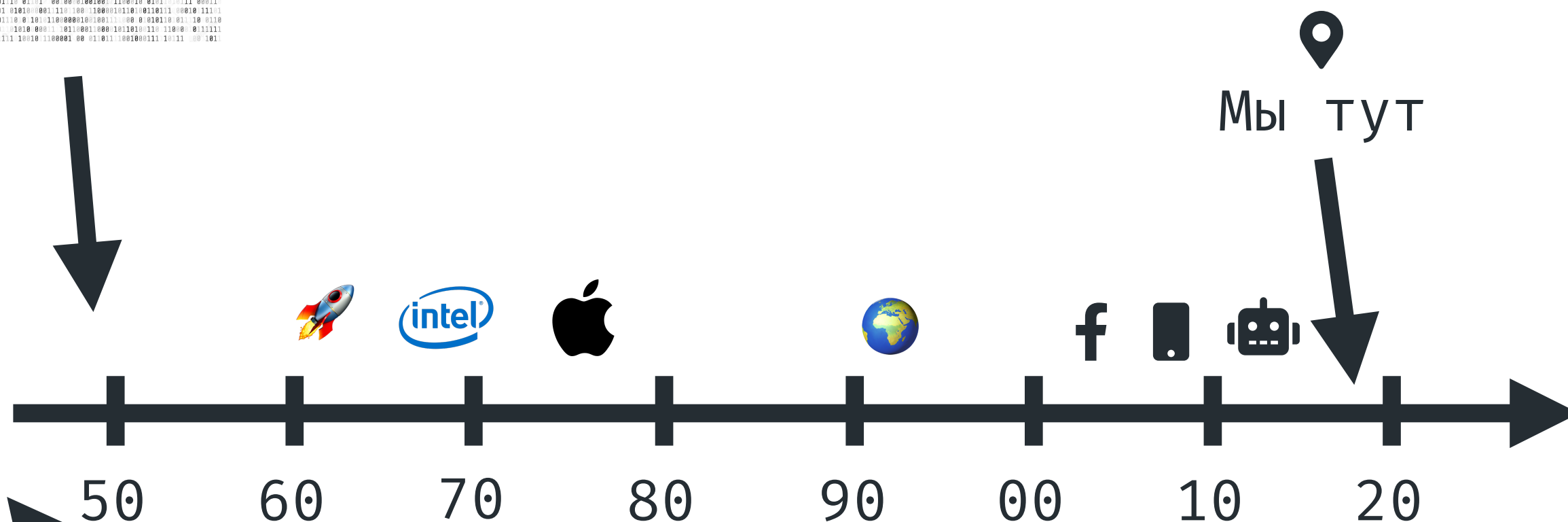
Джонни Кэш



Литтл Ричард

0100110110 1 110 1001100 010110110111 0100111101100011011
11 010 10 1101 110 10100100 101100100 1100 10 1 101000
0110 01100111 111110 1000001 010111 01 10 01101110 110
0001 1 10 101 1000 010 101 0001 0 0 101 0 011111 00001111 1
011000 0 11110011110 010 0001011100 011111 110 0011011
100100100000100110110101 10101011000 00100100000 0101
0010011100 100 11110 0010000011 11110 11 1000 11 1 01 0111
0011000 1101010 01010 1011 0 110000 0110 0 111101101 1 1 1
01 1 00 101 11 11010000101 0 1011100100111 000000 0
110 1 01000111 10 111 0100 111100010000100 10010 1100 11
11011 00010 011 0100 10 00 1111 010100011 0 1 1000100001010
010001000110 0110 011000100 0011100 101 101 010 011 00110
10 111000 010101000 011111000110 01011 10011011 0000101 11
0110000101111 010 11 000 00001101 0010001011 111 0 11011
01111100 00110111001101000000 1100 001111010100010111 00
1010 00000 0 0011010000010 1111 1011 001 0111010 1 0 0 0
0110101100011010110 1100100 010001001000 010101010111100 11
0 01 1 001010101000101 0 11000110000 0 010 1111000011101
0011000101111100 0 11100010 00 110111 01 0011110111100110 1
1111101 0 0001110000011010000 0 0000 0010 00001100 0110 01
00101100001 0001100100 001 001 1 0 01100111011101001000001
1110111000110 01 01 00 00 0100100 110010 011 0011 00011
0001100 101 01010 001 110 00 110001110 011011 00010 1111
0 10101 000110 0 10 110000010010011 000 0 010110 011 10 0110
0011100 101 1010 0001 101100011000 1011010 110 11000 0 111111
011000010111 10010 1100001 00 0110111001000111 10111 00 1011

Программирование в машинных кодах



Plankalkül
(~1945, Конрад Цузе)



Plankalkül (~1945)



```
1 P1 max3 ( V0[:8.0], V1[:8.0], V2[:8.0] ) → R0[:8.0]
2 max( V0[:8.0], V1[:8.0] ) → Z1[:8.0]
3 max( Z1[:8.0], V2[:8.0] ) → R0[:8.0]
4 END
5 P2 max ( V0[:8.0], V1[:8.0] ) → R0[:8.0]
6 V0[:8.0] → Z1[:8.0]
7 ( Z1[:8.0] < V1[:8.0] ) → V1[:8.0] → Z1[:8.0]
8 Z1[:8.0] → R0[:8.0]
9 END
```


Парадигма программирования —
способ программирования

Обычно **языки программирования**
поддерживают разные парадигмы
программирования

Императивное программирование (парадигма)

программирование с помощью
явно написанной
последовательности команд,
которые меняют состояние
системы

NB: Парадигмы помогают
контролировать сложность!

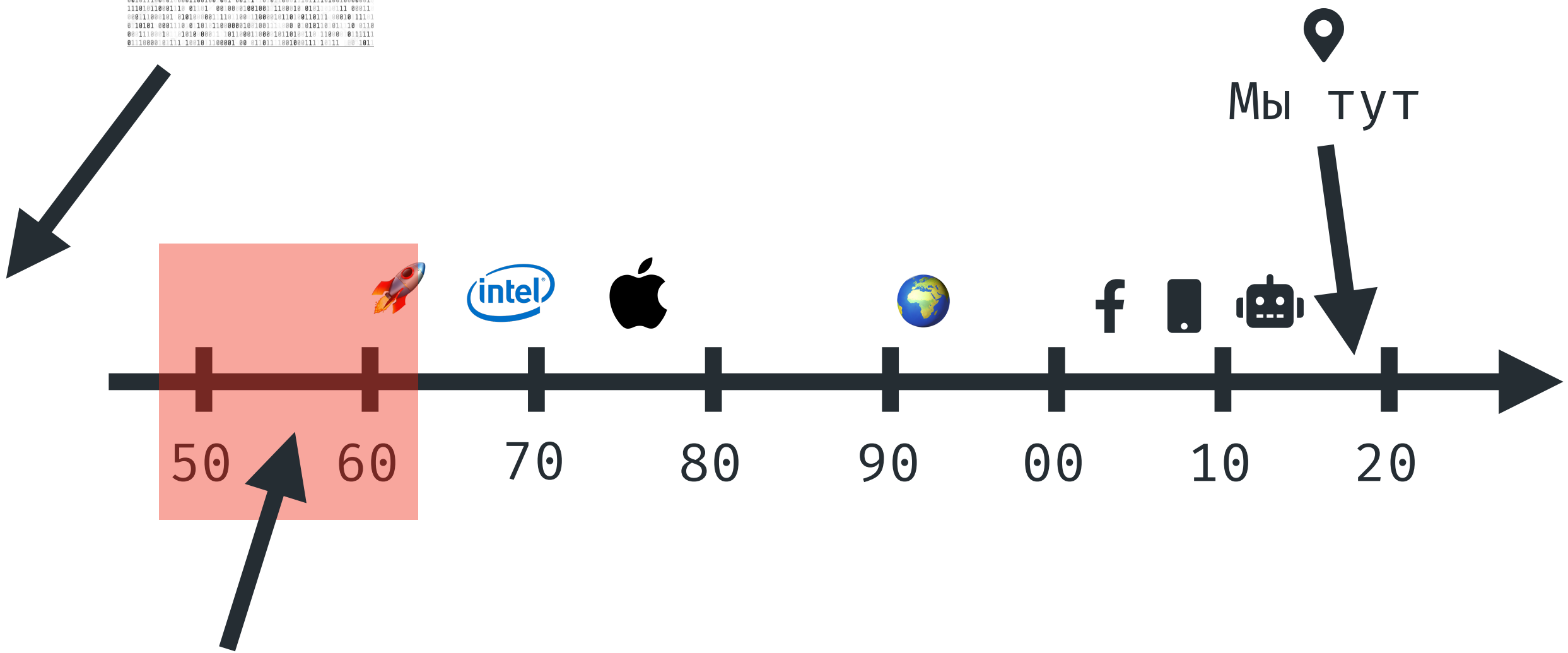


IBM 704 (1954)



```
010100110 1 1 10 00110 0101010111 010011110100011011
11 010 10 1101 110 0 1010100 10110100 1100 10 1 101000
0110 01100111 1011 10 10000 0 010111 001 10 010101110 010
0001 1 10 1011000 010 101 0001 0 0 01 0 001101 00001111
0111000 0 11110001110 010 0001011100 001111 110 0011011
11010010000001001101010101010101000 001001001 00000 0101
0010011100 100 1 110 0010000011 11110 11 1000 11 1 01 011
0011 000 1101010 0010 1011 0 110000 011010 11 110101 1 1 1
01 1 00 101 11 110100000101 0 10111100100111 000000 0
110 1 0100111 10 111 0100 11111000110000100 10010 1100 11
11011 00010 011 0100 10 00 1111 0101100011 011 101000000010
01001 000110 0110 0110001000 0111 000101 101 0101011 00110
10 11000 01010000 011111000110 010111001011 0000101 11
0111000101111 010 11 000 00 001101 0010001011 11 0 011011
01111100 00110110 011010000 11000 0011110010000 00111 00
1010 00000 01 00010010000010 1111 1011 001 0111010 1 0 0 0
0110 01100011010101 11001000 01001001000 01010101011100011
0101 1 01010101000101 1 011000110000 0 010 11 1110000101101
0011000010111100 011100010 00 110111 01 00111101111 00110 1
1111101 0 000110000011010000 0 0000 0010 00000100 01 0 01
001011100 01 000100100 001 001 1 0 01000110111010010000001
111011000110 01101 00 00 010000 110010 0101 0011 00011
00011000101 010100001110 100 110000101101 001011 00010 11101
010101 000110 0 1011100001001011000 0 01010 011 10 0110
001110010 1010 00011 0110001000 1011010110 11000 0111111
011100000111 10010 1100001 00 01011 001000111 1011 00 101
```

Программирование в машинных кодах



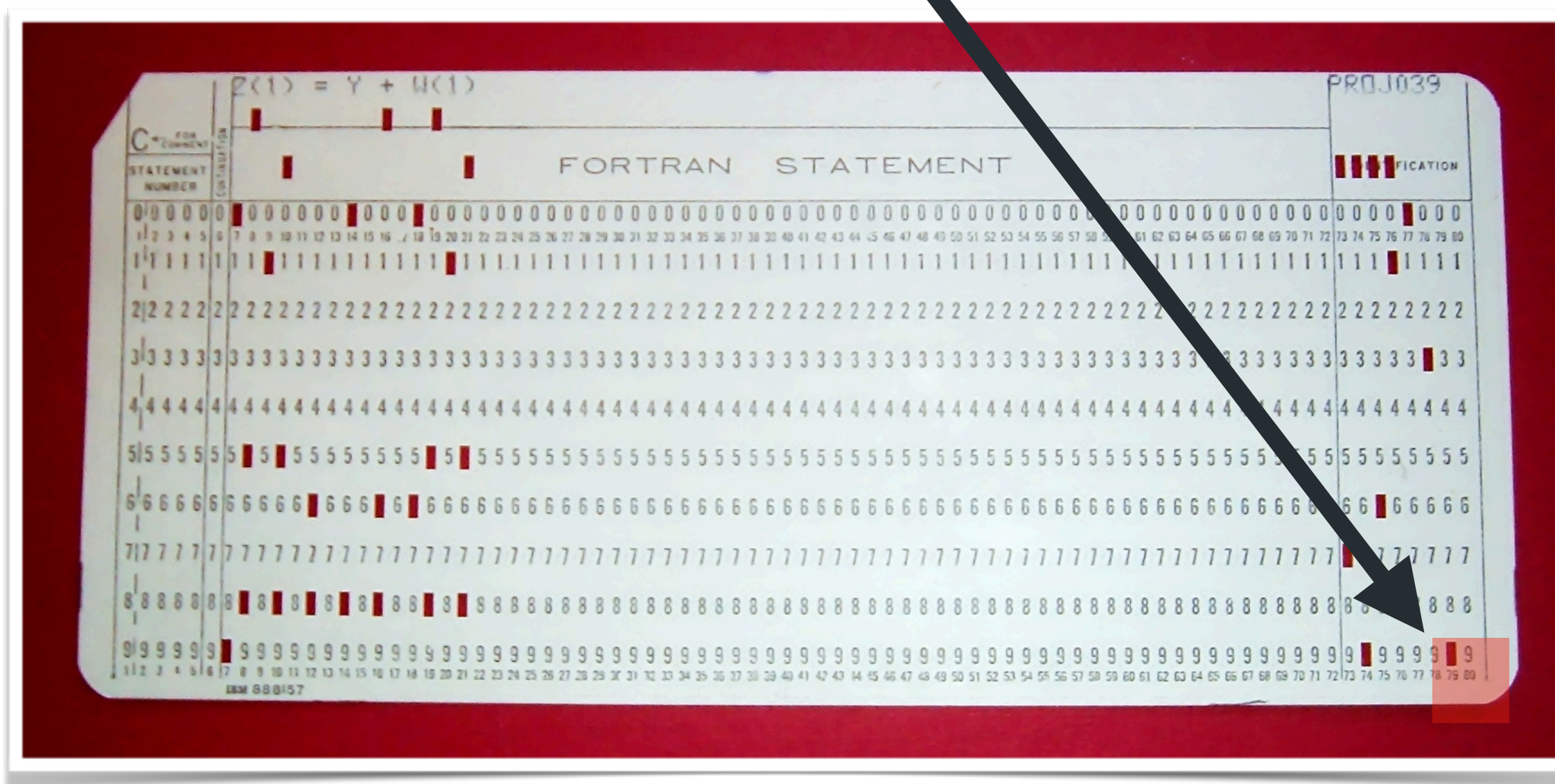
FORTRAN
(1957)



FORTRAN

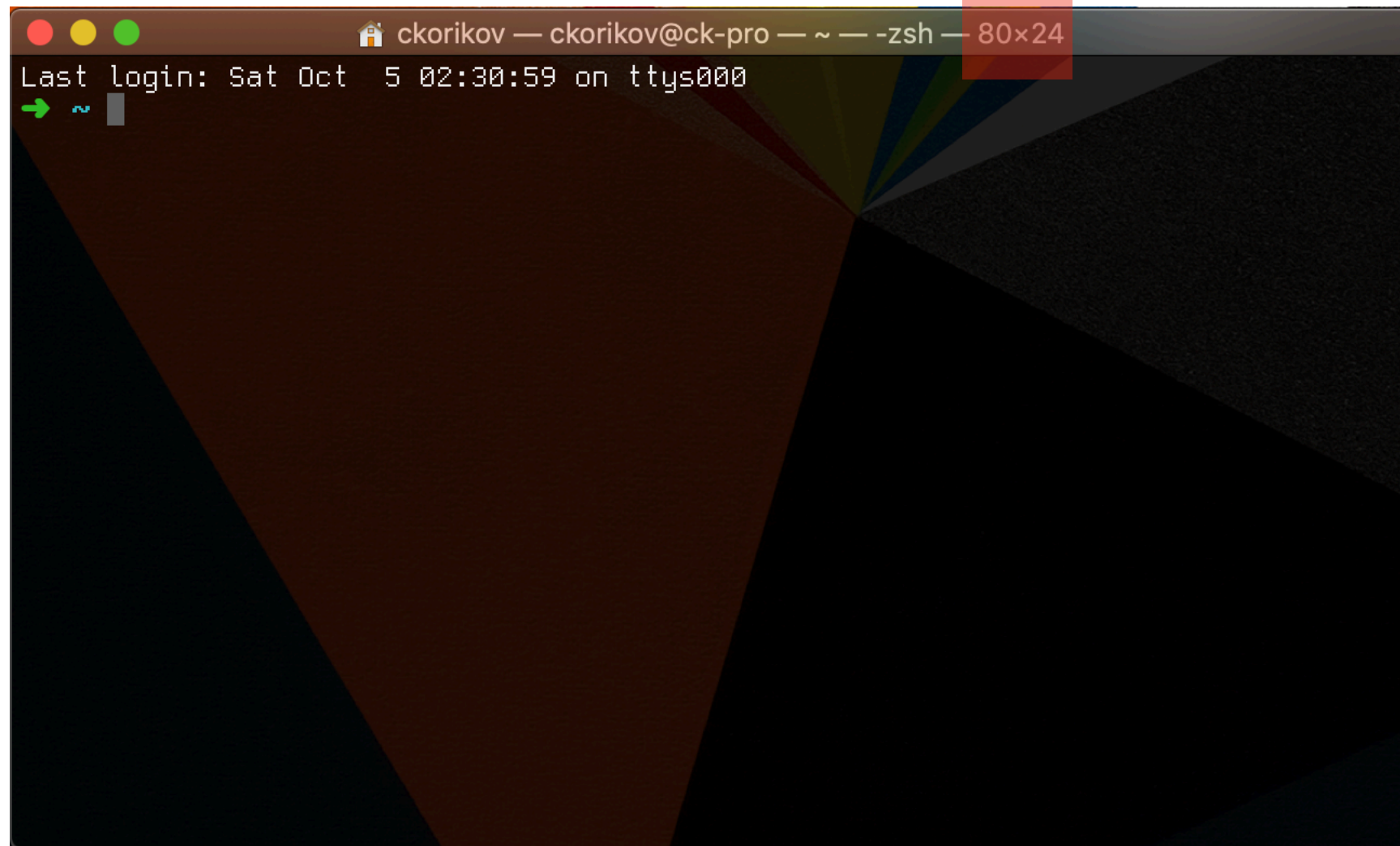
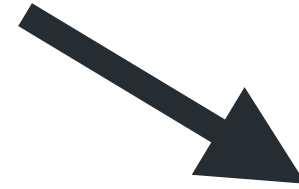
```
1 ! AREA OF A TRIANGLE - HERON'S FORMULA
2 ! INPUT - CARD READER UNIT 5, INTEGER INPUT
3 ! OUTPUT -
4 ! INTEGER VARIABLES START WITH I,J,K,L,M OR N
5     READ(5,501) IA,IB,IC
6 501 FORMAT(3I5)
7     IF(IA.EQ.0 .OR. IB.EQ.0 .OR. IC.EQ.0) STOP 1
8     S = (IA + IB + IC) / 2.0
9     AREA = SQRT( S * (S - IA) * (S - IB) * (S - IC) )
10    WRITE(6,601) IA,IB,IC,AREA
11 601 FORMAT(4H A= ,I5,5H B= ,I5,5H C= ,I5,8H AREA= ,F10.2,
12    $13H SQUARE UNITS)
13    STOP
14    END
```

80 столбцов



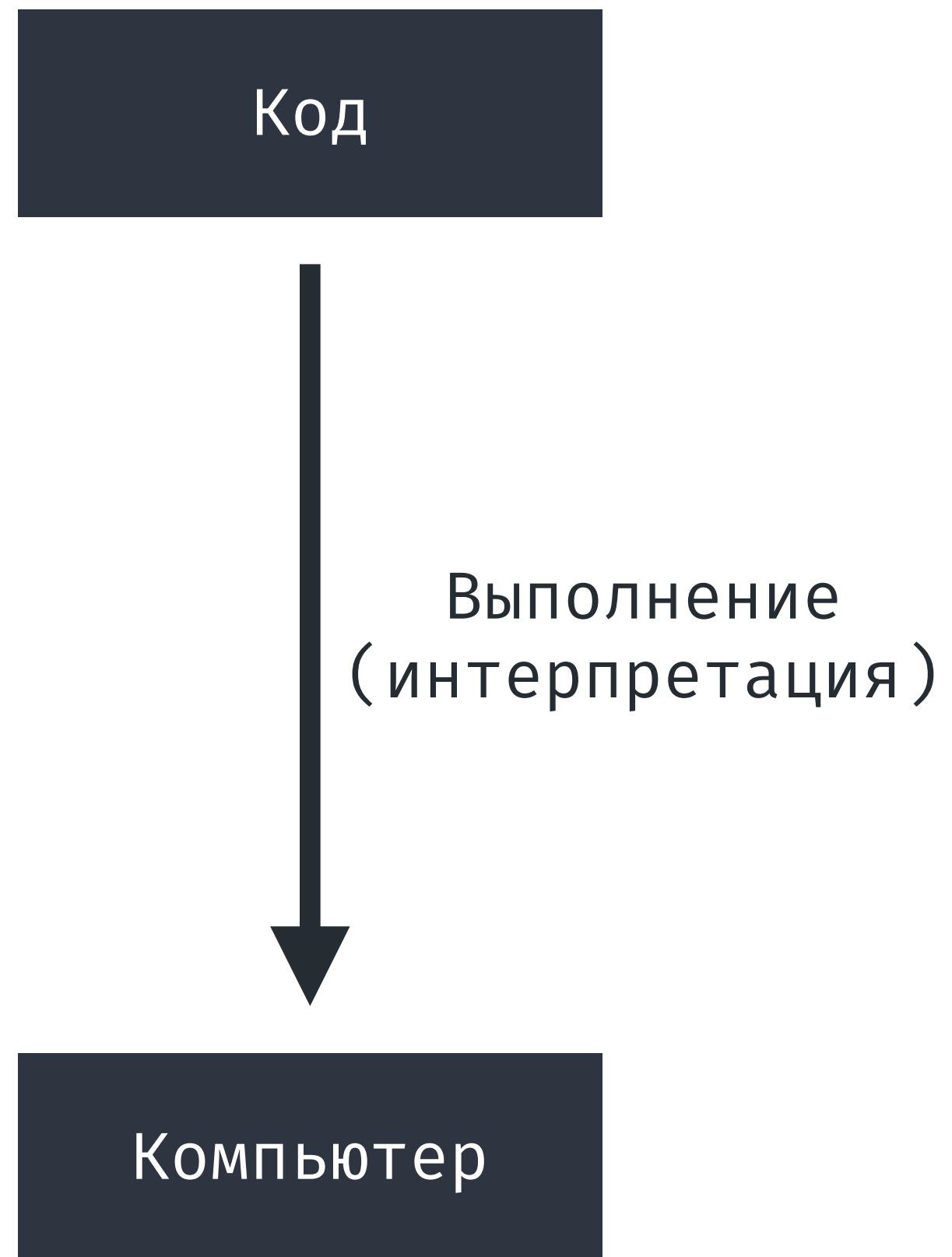
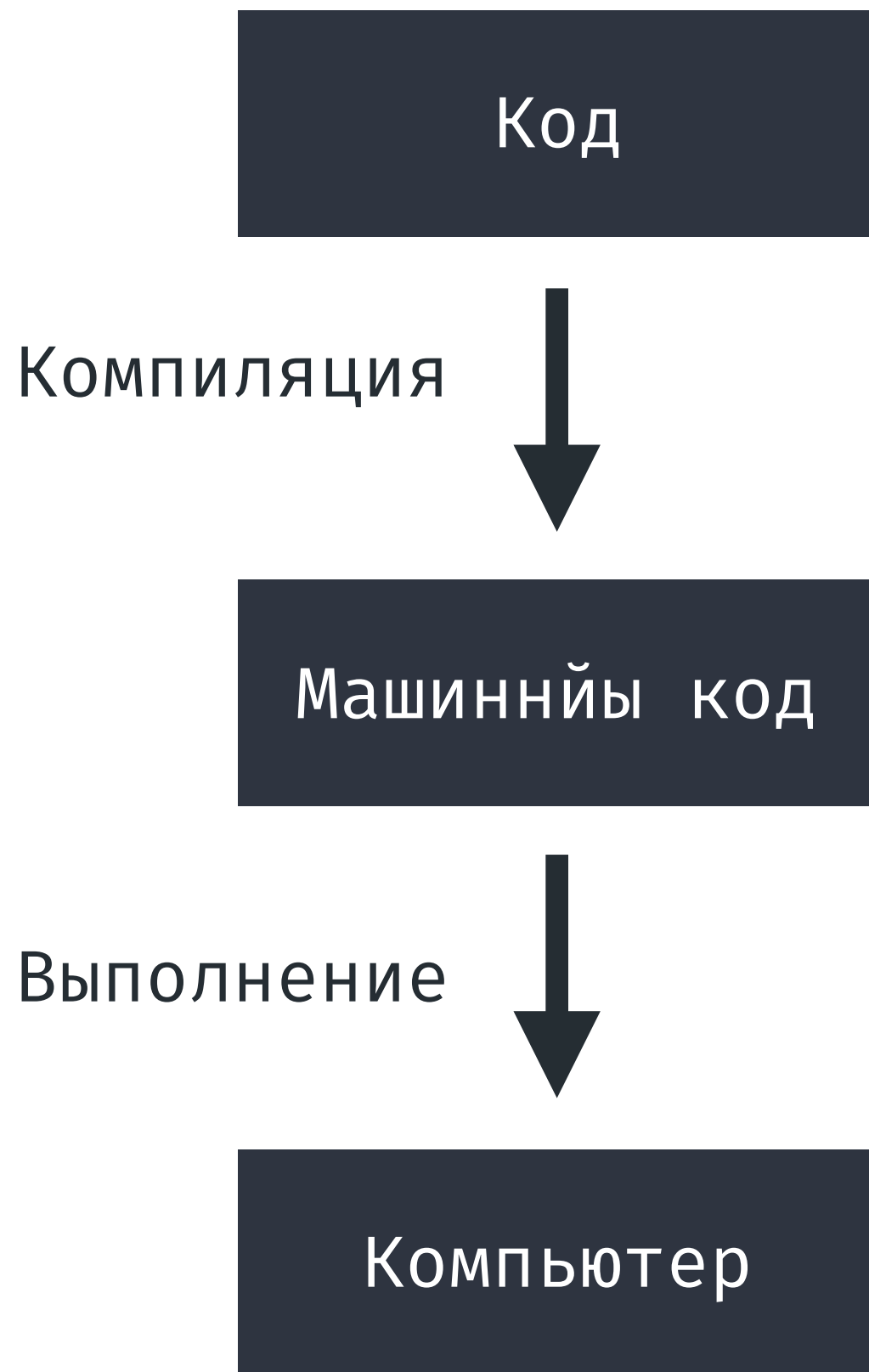
Строчка кода на Fortran на перфокарте

80 столбцов





Компилятор Fortran



Типизация

```
graph TD; A[Типизация] --> B[Статическая]; A --> C[Динамическая]
```

A diagram illustrating the classification of typing. At the top, a dark blue rectangular box contains the word "Типизация" in white. Two dark blue arrows originate from the bottom center of this box and point downwards and outwards to two separate text labels: "Статическая" on the left and "Динамическая" on the right.

Статическая

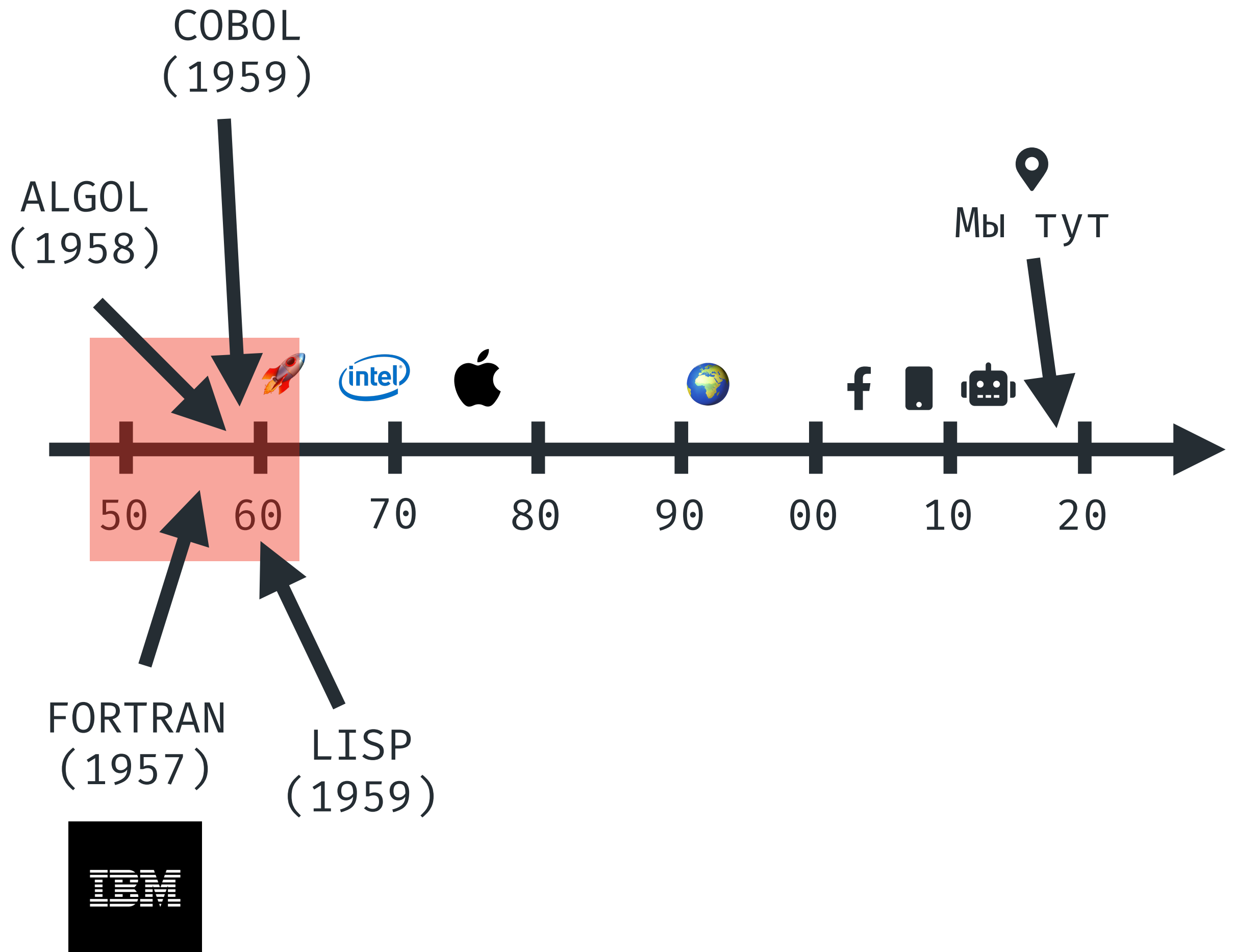
Динамическая

Типизация

Строгая



Слабая



LISP



```
1 (defun comb (m list fn)
2   (labels ((comb1 (l c m)
3             (when (>= (length l) m)
4               (if (zerop m) (return-from comb1 (funcall fn c))
5                 (comb1 (cdr l) c m)
6                 (comb1 (cdr l) (cons (first l) c) (1- m))))))
7   (comb1 list nil m)))
8
9 (comb 3 '(0 1 2 3 4 5) #'print)
```


Функциональное программирование (парадигма)

способ программирования с
помощью функций в
математическом смысле: нет
побочных эффектов

ALGOL



```
1 BEGIN
2 FILE F (KIND=REMOTE);
3 EBCDIC ARRAY E [0:11];
4 REPLACE E BY "HELLO WORLD!";
5 WHILE TRUE DO
6   BEGIN
7     WRITE (F, *, E);
8   END;
9 END.
```

COBOL

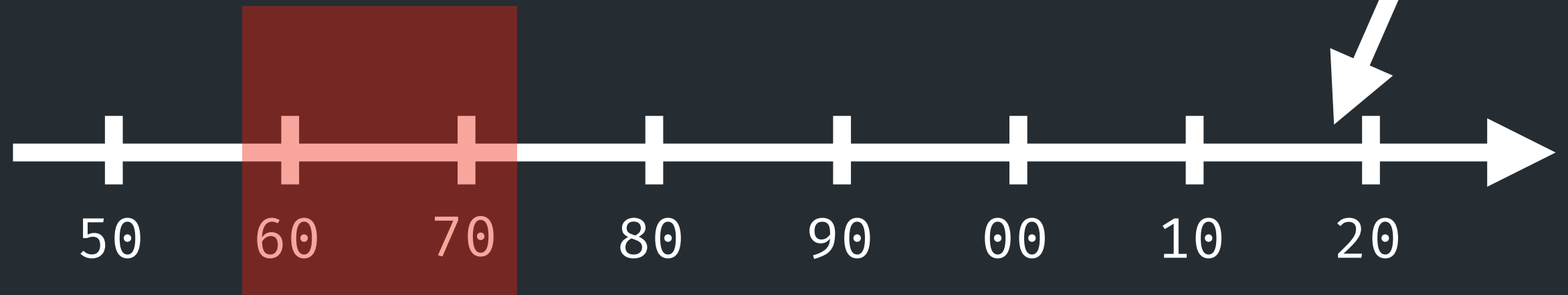
```
1  IDENTIFICATION DIVISION_.  
2  PROGRAM-ID_ A-Plus-B_.  
3  
4  DATA DIVISION_.  
5  WORKING-STORAGE SECTION_.  
6  01  A          PIC S9(5)_.  
7  01  B          PIC S9(5)_.  
8  
9  01  A-B-Sum    PIC S9(5)_.  
10  
11  PROCEDURE DIVISION_.  
12      ACCEPT A  
13      ACCEPT B  
14  
15      ADD A TO B GIVING A-B-Sum  
16  
17      DISPLAY A-B-Sum  
18  
19      GOBACK  
20      _.
```




The Beatles



Джими Хендрикс



Led Zeppelin



The Rolling Stones



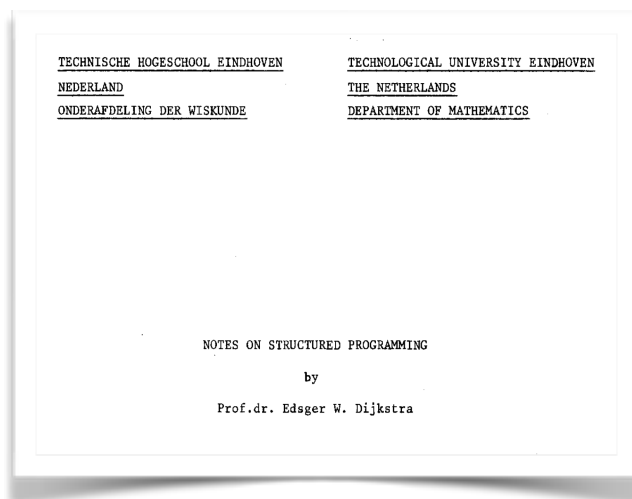
The Stooges



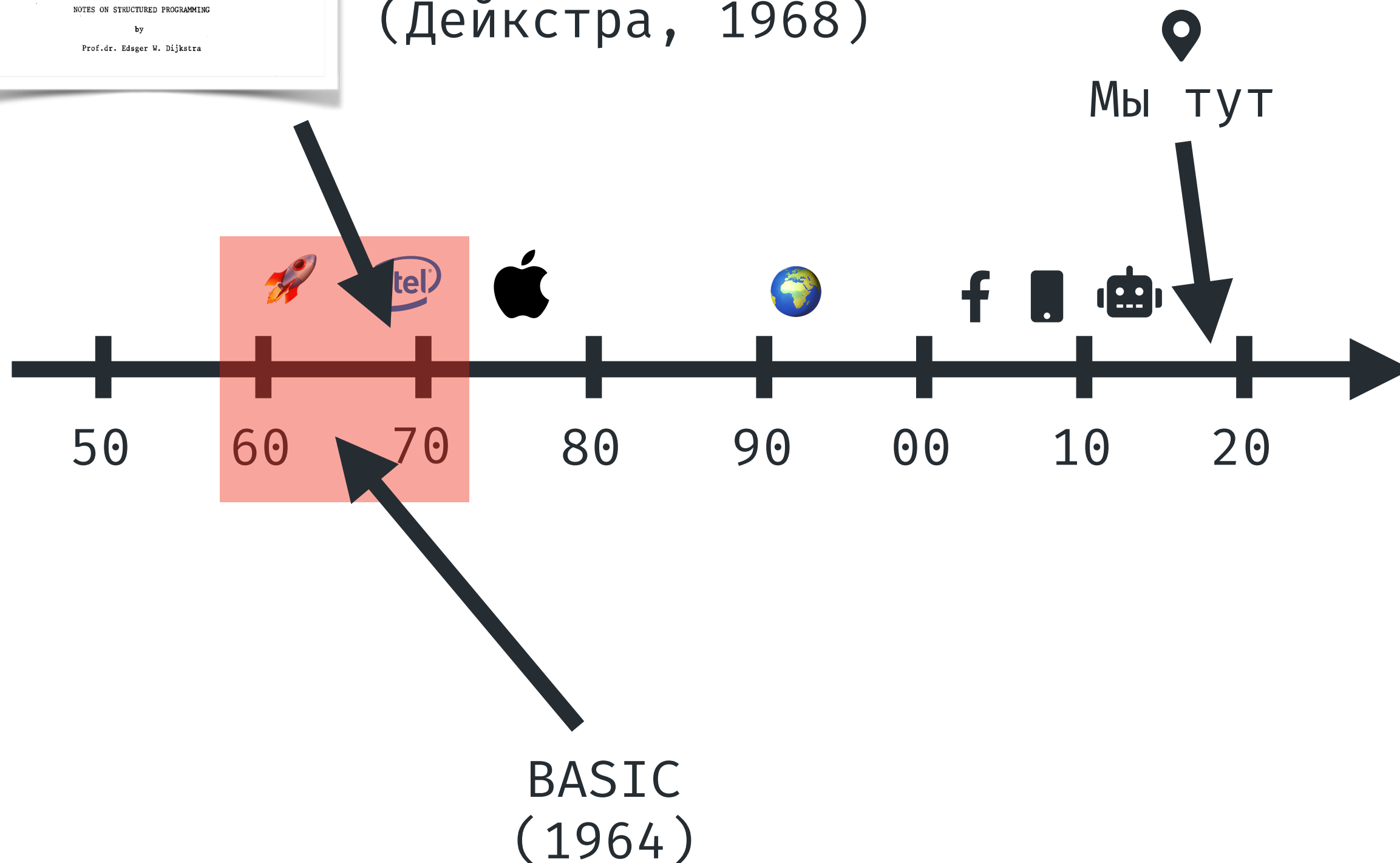
IBM System/360 (1964)



NCR 315 (1965)



Структурное программирование (Дейкстра, 1968)



Basic

```
1 input "N = ",f
2 limit = 500          # set upper limit - can be changed, removed
3 f = int(f)
4 if f > limit then f = limit
5 a = 0 : b = 1 : c = 0 : n = 0    # initial values
6
7
8 while n < f
9     print n + chr(9) + c    # chr(9) = tab
10    a = b
11    b = c
12    c = a + b
13    n += 1
14
15 end while
16
17 print " "
18 print n + chr(9) + c
```

Структурное программирование (парадигма)

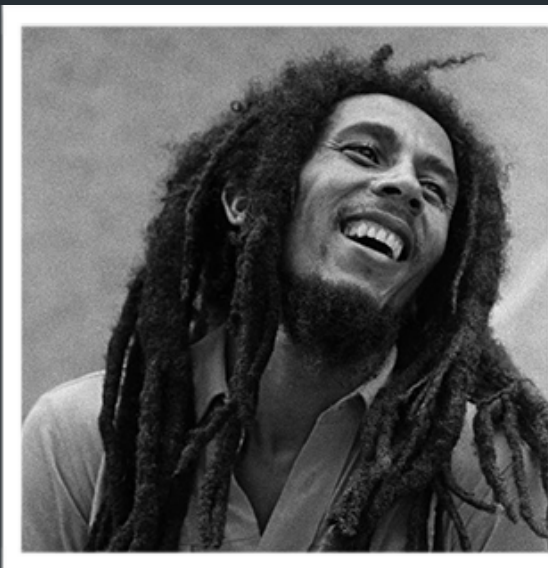
императивное программирование
в виде иерархии структурных
блоков на ветвлениях и циклах
(без goto)

Процедурное программирование (парадигма)

структурное программирование
с разбиением программы на
процедуры (функции)



Black Sabbath



Боб Марли



Neu!



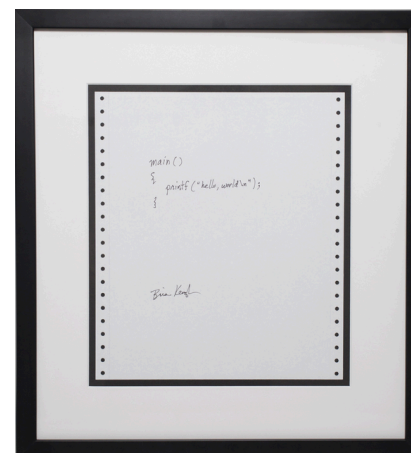
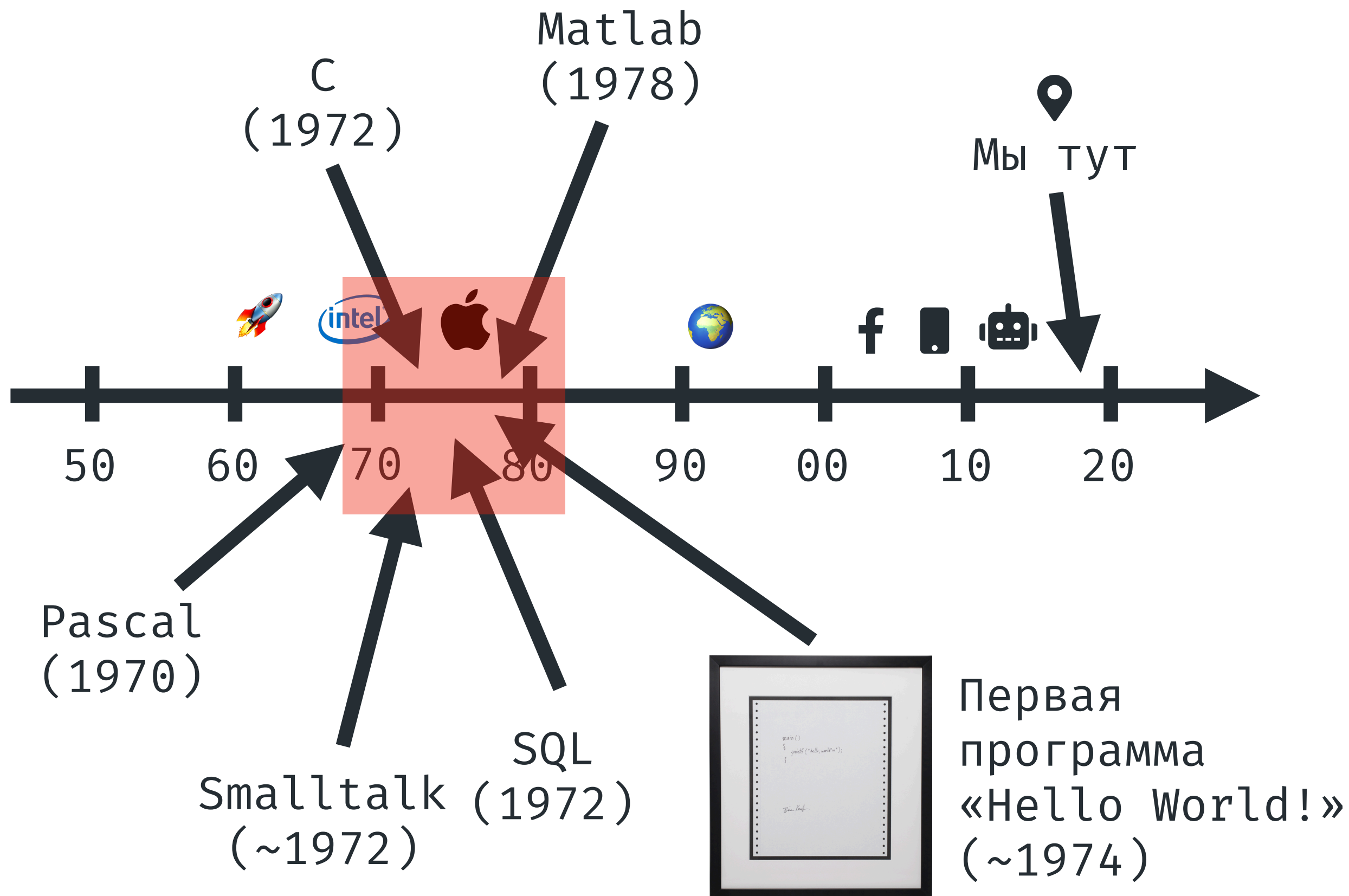
The Clash



Kraftwerk



IBM System/32 aka IBM 5320 (1975)



Первая
программа
«Hello World!»
(~1974)

Декларативное программирование (парадигма)

способ программирования, когда
указывается результат работы
без явного указания шагов
решения

SQL



```
1 SELECT LAT_N, CITY, TEMP_F
2 FROM STATS, STATION
3 WHERE MONTH = 7
4 AND STATS.ID = STATION.ID
5 ORDER BY TEMP_F;
```

C



```
1 long long int fibb(int n) {  
2     int fnow = 0, fnext = 1, tempf;  
3     while(--n>0){  
4         tempf = fnow + fnext;  
5         fnow = fnext;  
6         fnext = tempf;  
7     }  
8     return fnext;  
9 }
```

Matlab

```
1 function F = fibonacci(n)
2
3     Fn = [1 0]; %Fn(1) is F_{n-2}, Fn(2) is F_{n-1}
4     F = 0; %F is F_n
5
6     for i = (1:abs(n))
7         Fn(2) = F;
8         F = sum(Fn);
9         Fn(1) = Fn(2);
10    end
11
12    if n < 0
13        F = F*((-1)^(n+1));
14    end
15
16 end
```


Pascal



```
1 function fib(n: integer): integer;  
2 begin  
3     if (n = 0) or (n = 1)  
4     then  
5         fib := n  
6     else  
7         fib := fib(n-1) + fib(n-2)  
8 end;
```

Smalltalk

```
1 |fibo|
2 fibo := [ :i |
3     |ac t|
4     ac := Array new: 2.
5     ac at: 1 put: 0 ; at: 2 put: 1.
6     ( i < 2 )
7     ifTrue: [ ac at: (i+1) ]
8     ifFalse: [
9         2 to: i do: [ :l |
10             t := (ac at: 2).
11             ac at: 2 put: ( (ac at: 1) + (ac at: 2) ).
12             ac at: 1 put: t
13         ].
14         ac at: 2.
15     ]
16 ].
17
18 0 to: 10 do: [ :i |
19     (fibo value: i) displayNl
20 ]
```

Объектно-ориентированное программирование (парадигма)

способ программирования, в
котором моделируется задача
объектами



Joy Division



Metallica



Sonic Youth



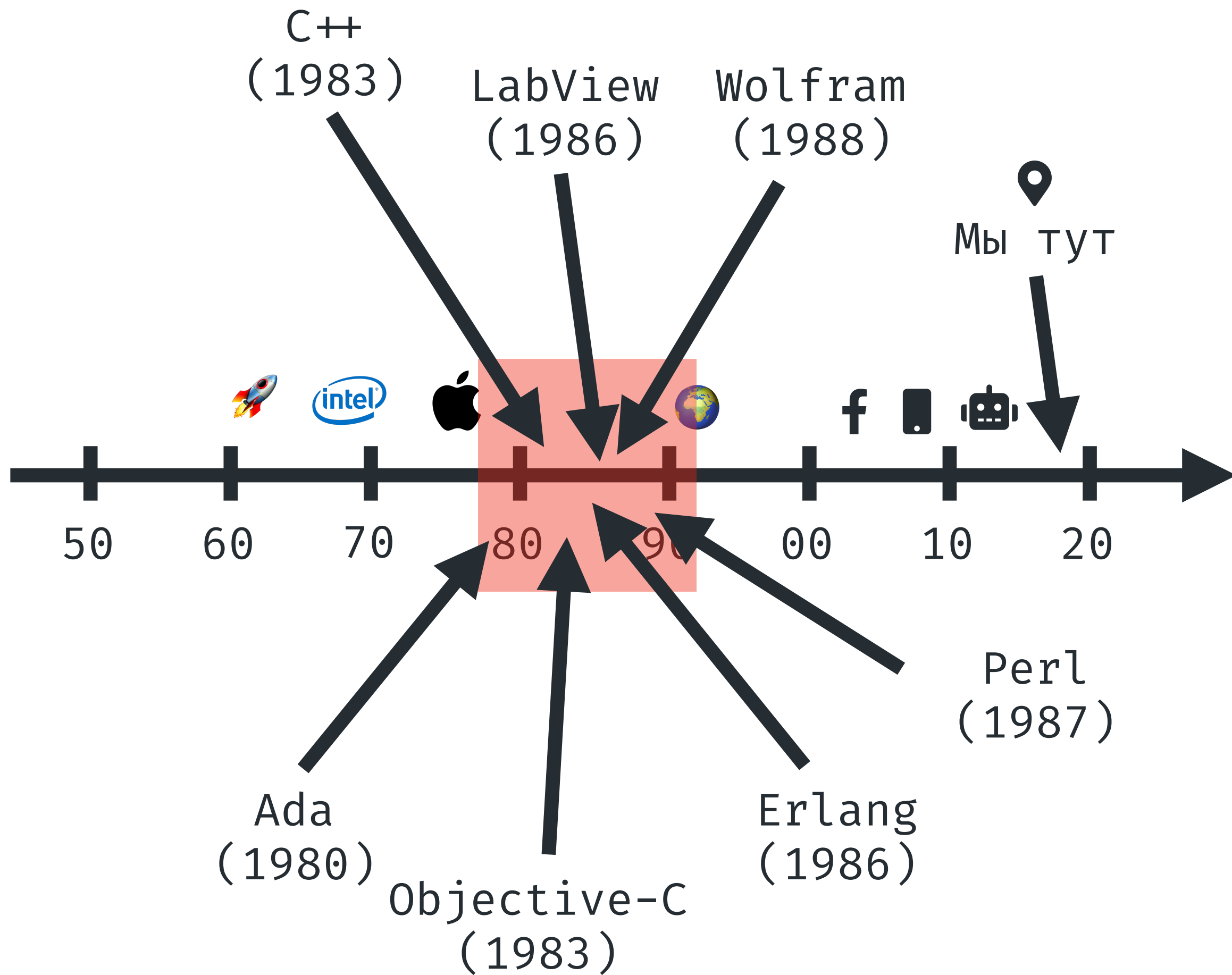
Public Enemy



Pixies



IBM 5150 PC (1981)



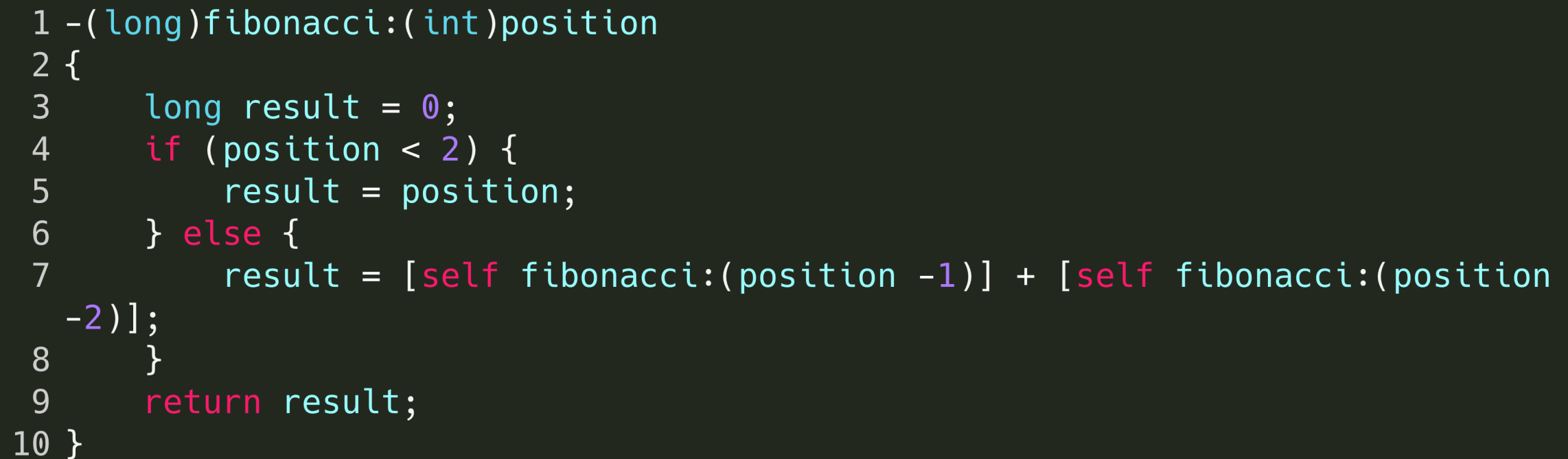
Ada



```
1 with Ada.Text_IO; use Ada.Text_IO;  
2 procedure Hello is  
3 begin  
4   Put_Line ("Hello, world!");  
5 end Hello;
```



Objective-C



```
1 -(long)fibonacci:(int)position
2 {
3     long result = 0;
4     if (position < 2) {
5         result = position;
6     } else {
7         result = [self fibonacci:(position - 1)] + [self fibonacci:(position
8         - 2)];
9     }
10    return result;
11 }
```


C++

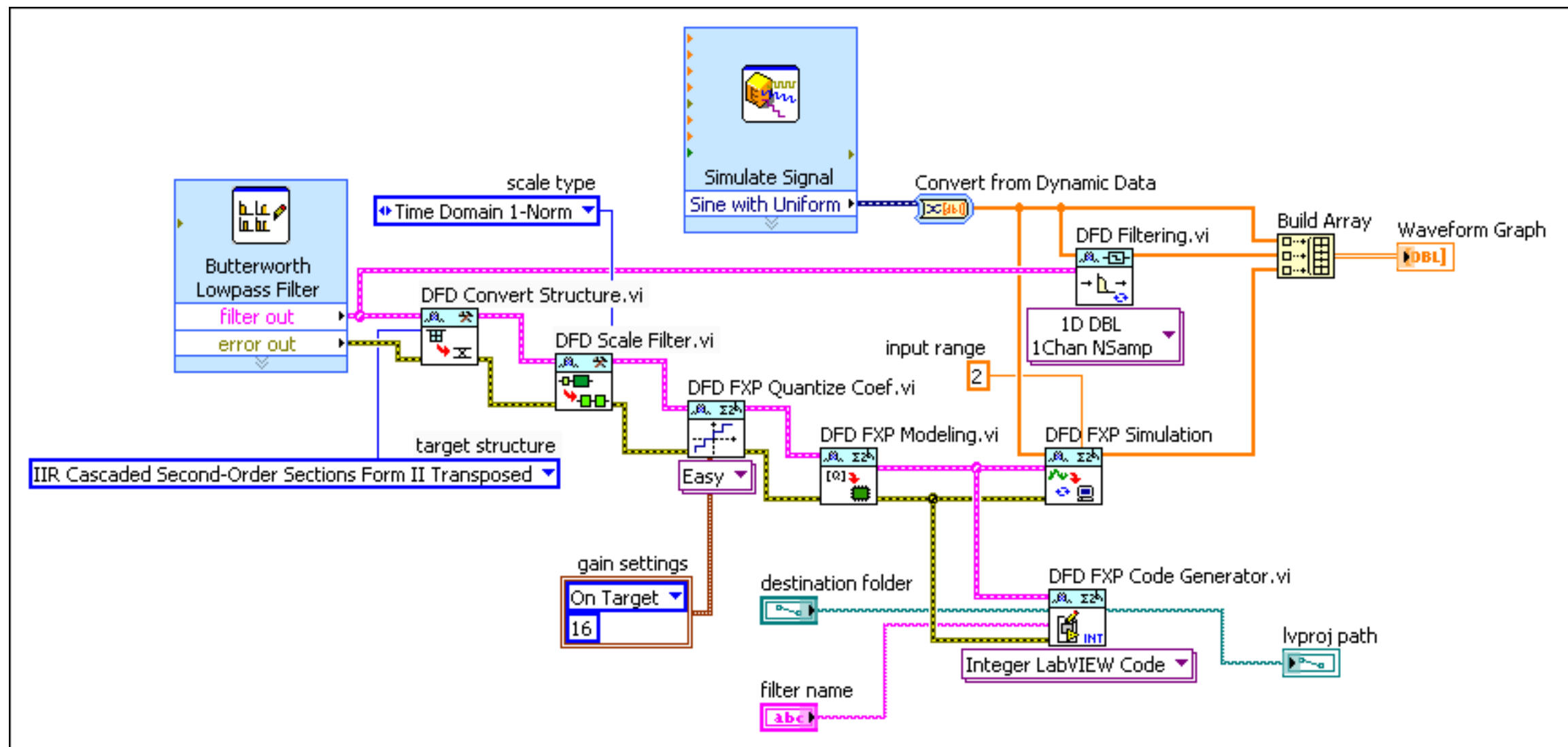
```
1 #include <iostream>
2
3 int main( )
4 {
5     unsigned int a = 1, b = 1;
6     unsigned int target = 48;
7     for(unsigned int n = 3; n <= target; ++n)
8     {
9         unsigned int fib = a + b;
10        std::cout << "F( " << n << " ) = " << fib << std::endl;
11        a = b;
12        b = fib;
13    }
14
15    return 0;
16 }
```

Erlang



```
1 -module(comb).
2 -compile(export_all).
3
4 comb(0,_) ->
5     [[]];
6 comb(_,[]) ->
7     [];
8 comb(N,[H|T]) ->
9     [[H|L] || L <- comb(N-1,T)] ++ comb(N,T).
```

LabView (1986)



Perl



```
1 sub fib (Int $n --> Int) {  
2     $n > 1 or return $n;  
3     my ($prev, $this) = 0, 1;  
4     ($prev, $this) = $this, $this + $prev for 1 .. ^ $n;  
5     return $this;  
6 }
```


Wolfram

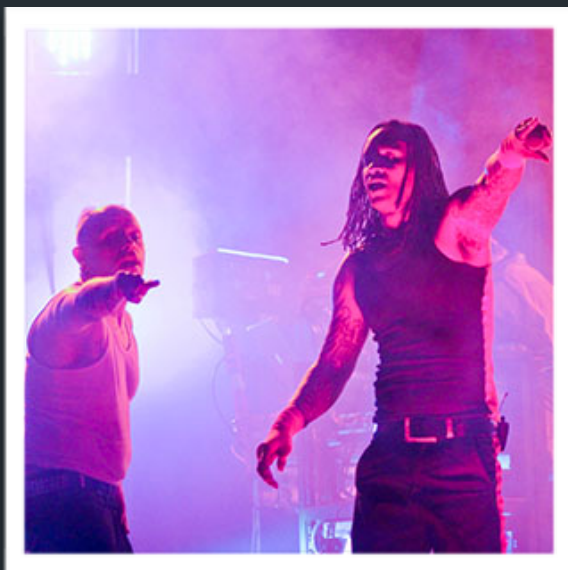
```
1 BinarySearch[x_List, val_] := Module[{lo = 1, hi = Length@x, mid},
2   While[lo <= hi,
3     mid = lo + Round@((hi - lo)/2);
4     Which[x[[mid]] > val, hi = mid - 1,
5           x[[mid]] < val, lo = mid + 1,
6           True, Return[mid]
7   ];
8 ];
9 Return[-1];
10 ]
```



Nirvana



Massive Attack



The Prodigy



Oasis

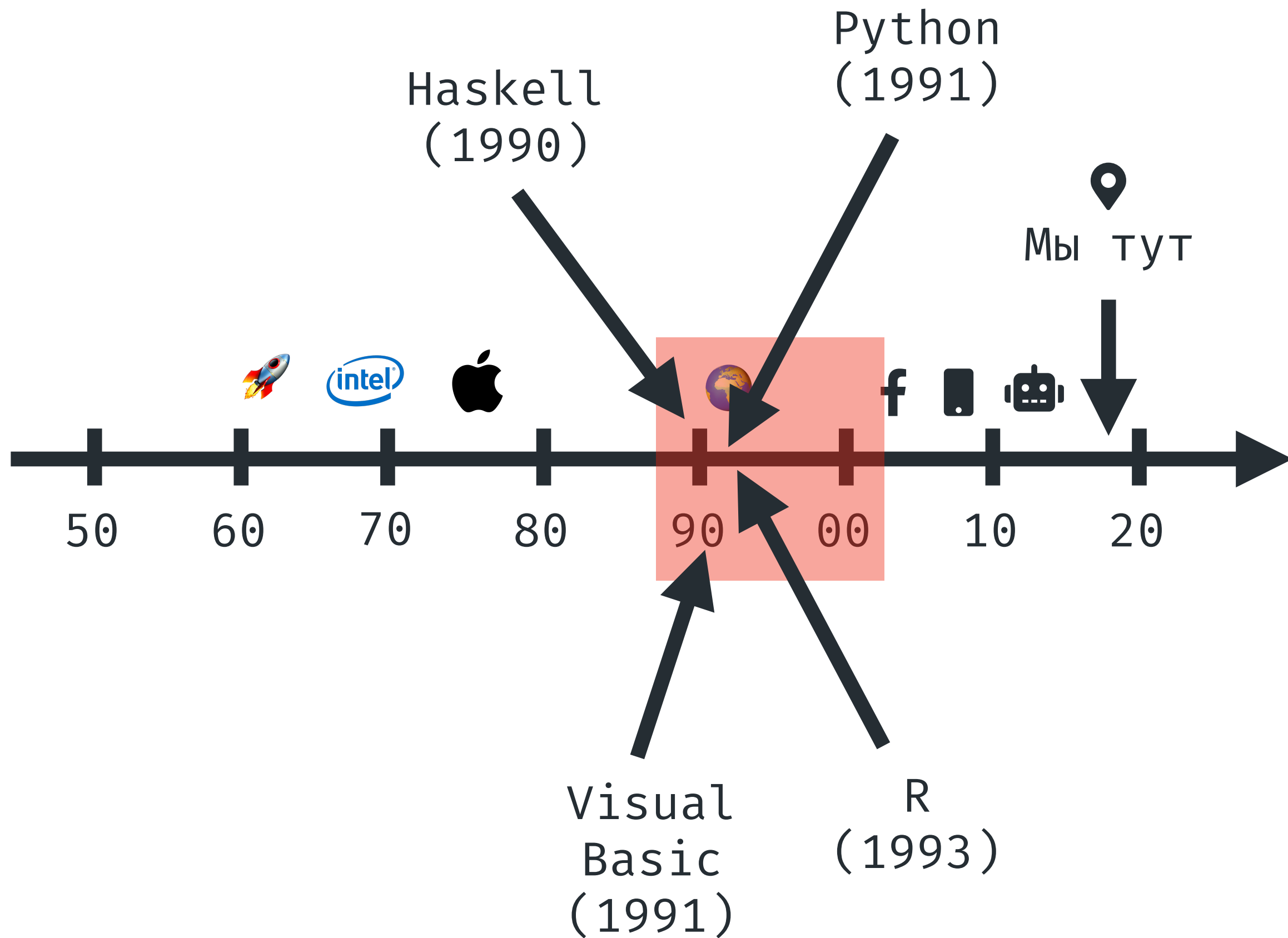


Radiohead



Apple The PowerBook 540c (1994)





Haskell



```
1 fib x =  
2   if x < 1  
3     then 0  
4     else if x < 2  
5           then 1  
6           else fib (x - 1) + fib (x - 2)
```

Python



```
1 def fibIter(n):  
2     if n < 2:  
3         return n  
4     fibPrev = 1  
5     fib = 1  
6     for num in xrange(2, n):  
7         fibPrev, fib = fib, fib + fibPrev  
8     return fib
```

VisualBasic

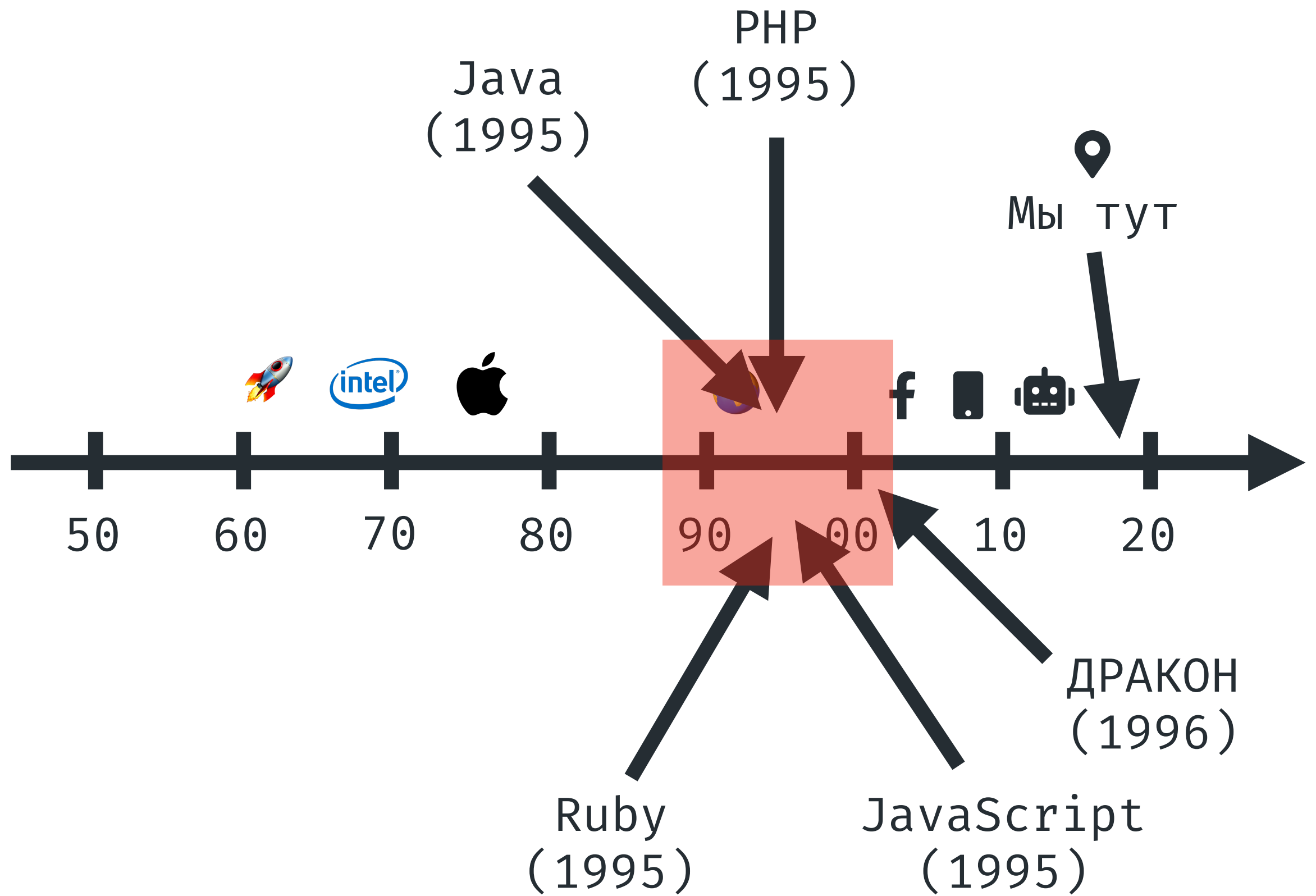


```
1 Sub fibonacci()  
2     Const n = 139  
3     Dim i As Integer  
4     Dim f1 As Variant, f2 As Variant, f3 As Variant 'for Decimal  
5     f1 = CDec(0): f2 = CDec(1) 'for Decimal setting  
6     Debug.Print "fibonacci(0; 0; )="; f1  
7     Debug.Print "fibonacci(0; 1; )="; f2  
8     For i = 2 To n  
9         f3 = f1 + f2  
10        Debug.Print "fibonacci(0; i; )="; f3  
11        f1 = f2  
12        f2 = f3  
13    Next i  
14 End Sub 'fibonacci
```

R



```
1 fib=function(n,x=c(0,1)) {  
2   if (abs(n)>1) for (i in seq(abs(n)-1)) x=c(x[2],sum(x))  
3   if (n<0) return(x[2]*(-1)^(abs(n)-1)) else if (n) return(x[2]) else  
   return(0)  
4 }  
5  
6 sapply(seq(-31,31),fib)
```

Java



```
1 public static long itFibN(int n)
2 {
3     if (n < 2)
4         return n;
5     long ans = 0;
6     long n1 = 0;
7     long n2 = 1;
8     for(n--; n > 0; n--)
9     {
10         ans = n1 + n2;
11         n1 = n2;
12         n2 = ans;
13     }
14     return ans;
15 }
```

PHP



```
1 function fibIter($n) {  
2     if ($n < 2) {  
3         return $n;  
4     }  
5     $fibPrev = 0;  
6     $fib = 1;  
7     foreach (range(1, $n-1) as $i) {  
8         list($fibPrev, $fib) = array($fib, $fib + $fibPrev);  
9     }  
10    return $fib;  
11 }
```

Ruby



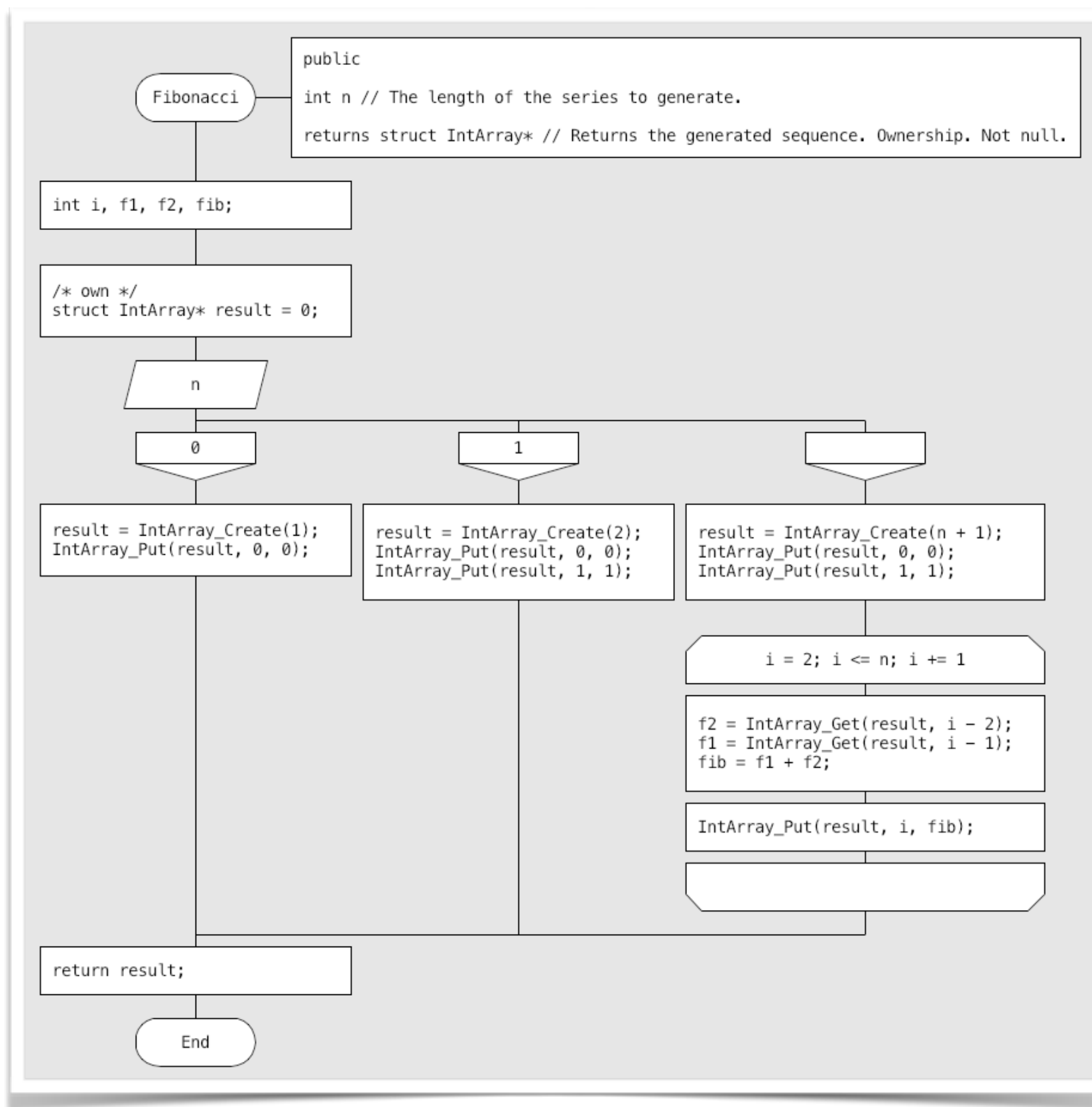
```
1 def fib(n, sequence=[1])
2   return sequence.last if n == 0
3
4   current_number, last_number = sequence.last(2)
5   sequence << current_number + (last_number or 0)
6
7   fib(n-1, sequence)
8 end
```


JavaScript



```
1 function fib(n) {  
2   var a = 0, b = 1, t;  
3   while (n-- > 0) {  
4     t = a;  
5     a = b;  
6     b += t;  
7     console.log(a);  
8   }  
9   return a;  
10 }
```

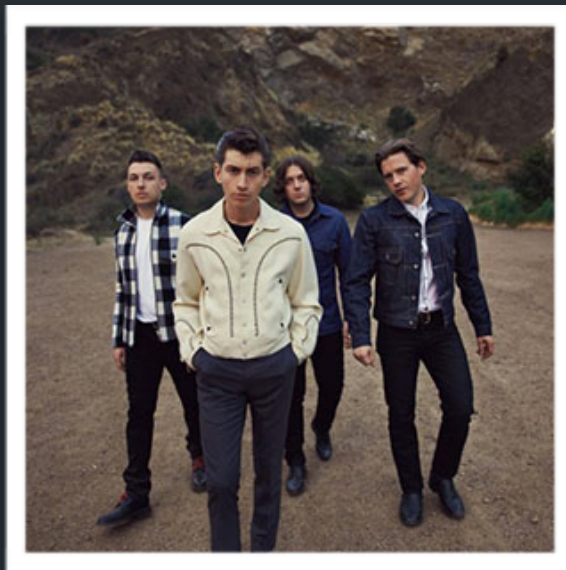
ДРАКОН



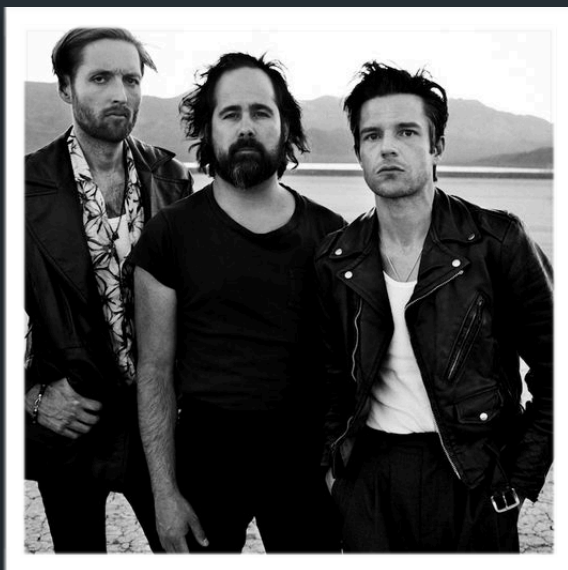
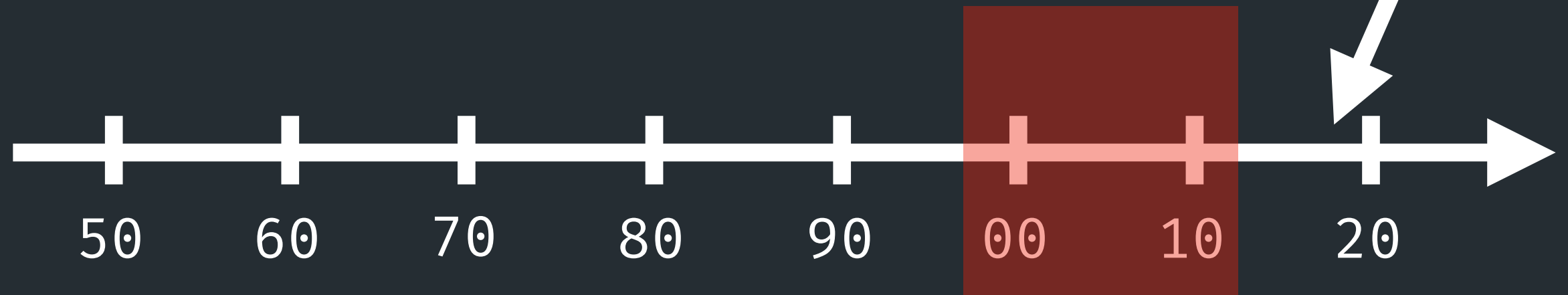
Дружелюбный русский алгоритмический язык, который обеспечивает наглядность



The Strokes



Arctic Monkeys



The Killers



Interpol

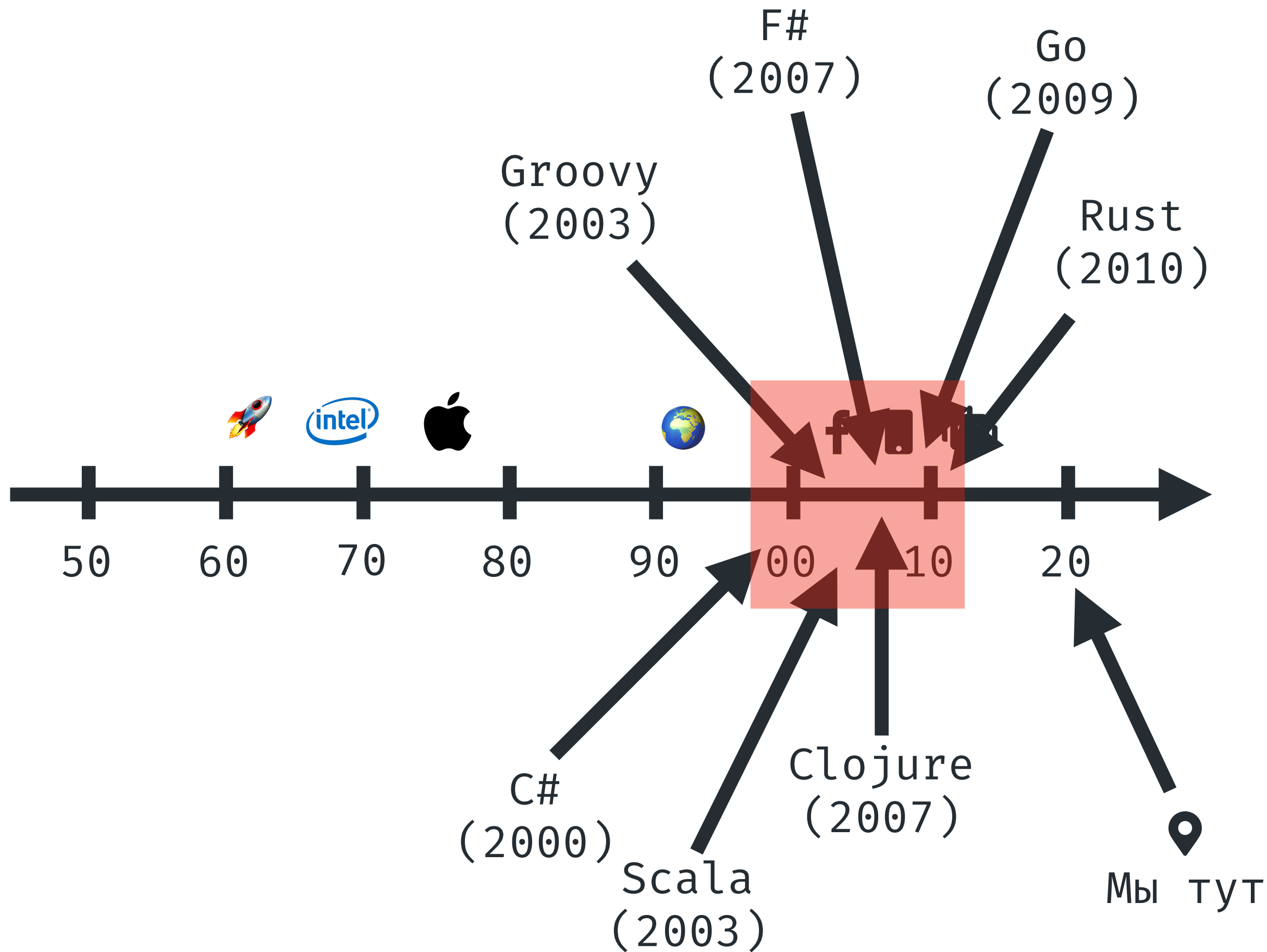


The White Stripes



Apple iPhone (2007)





C#



```
1 public static ulong Fib(uint x) {  
2     if (x == 0) return 0;  
3  
4     ulong prev = 0;  
5     ulong next = 1;  
6     for (int i = 1; i < x; i++)  
7     {  
8         ulong sum = prev + next;  
9         prev = next;  
10        next = sum;  
11    }  
12    return next;  
13 }
```

Groovy



```
1 def rFib
2 rFib = {
3     it == 0 ? 0
4     : it == 1 ? 1
5     : it > 1 ? rFib(it-1) + rFib(it-2)
6     /*it < 0*/: rFib(it+2) - rFib(it+1)
7
8 }
```

Scala



```
1 def fib(i:Int):Int = i match{  
2   case 0 => 0  
3   case 1 => 1  
4   case _ => fib(i-1) + fib(i-2)  
5 }
```

Clojure



```
1 (def fib (lazy-cat [0 1] (map + fib (rest fib))))
```


F#



```
1 let fibonacci n : bigint =  
2   let rec f a b n =  
3     match n with  
4       | 0 -> a  
5       | 1 -> b  
6       | n -> (f b (a + b) (n - 1))  
7   f (bigint 0) (bigint 1) n  
8 > fibonacci 100;;  
9 val it : bigint = 354224848179261915075I
```

Go



```
1 func fib(a int) int {  
2     if a < 2 {  
3         return a  
4     }  
5     return fib(a - 1) + fib(a - 2)  
6 }
```

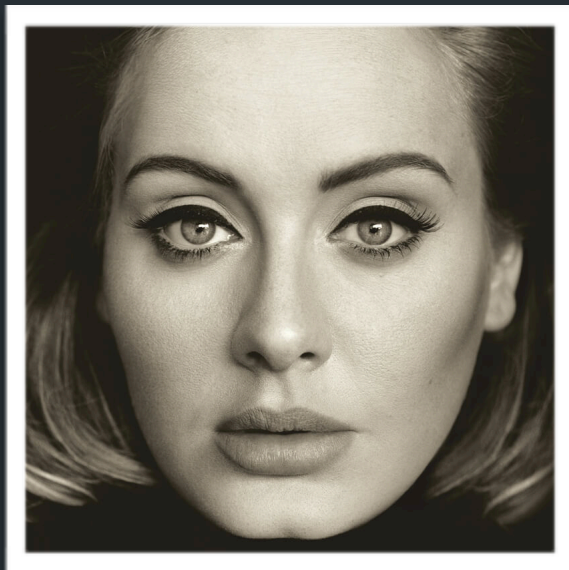
Rust



```
1 use std::mem;
2 fn main() {
3     let mut prev = 0;
4     // Rust needs this type hint for the checked_add method
5     let mut curr = usize;
6
7     while let Some(n) = curr.checked_add(prev) {
8         prev = curr;
9         curr = n;
10        println!("{}", n);
11    }
12 }
```



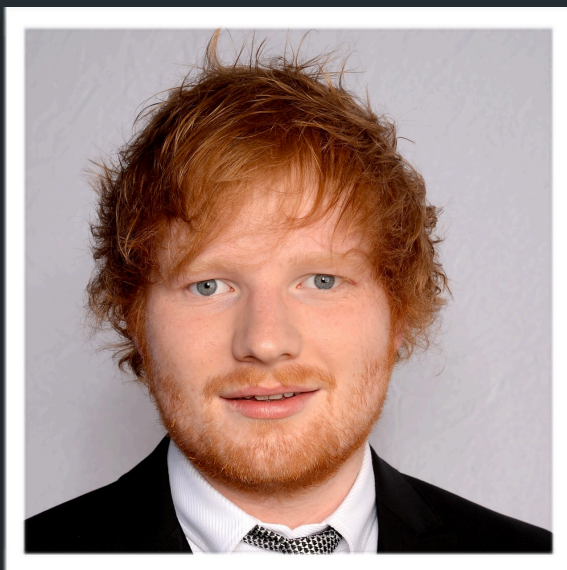
Kendrick Lamar



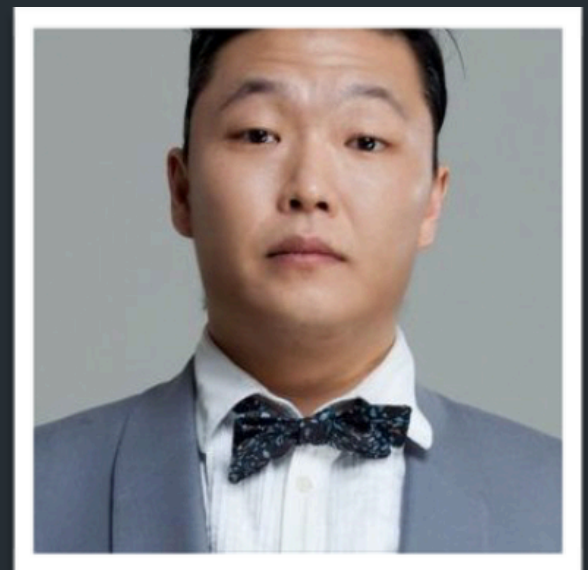
Adele



Drake



Ed Sheeran

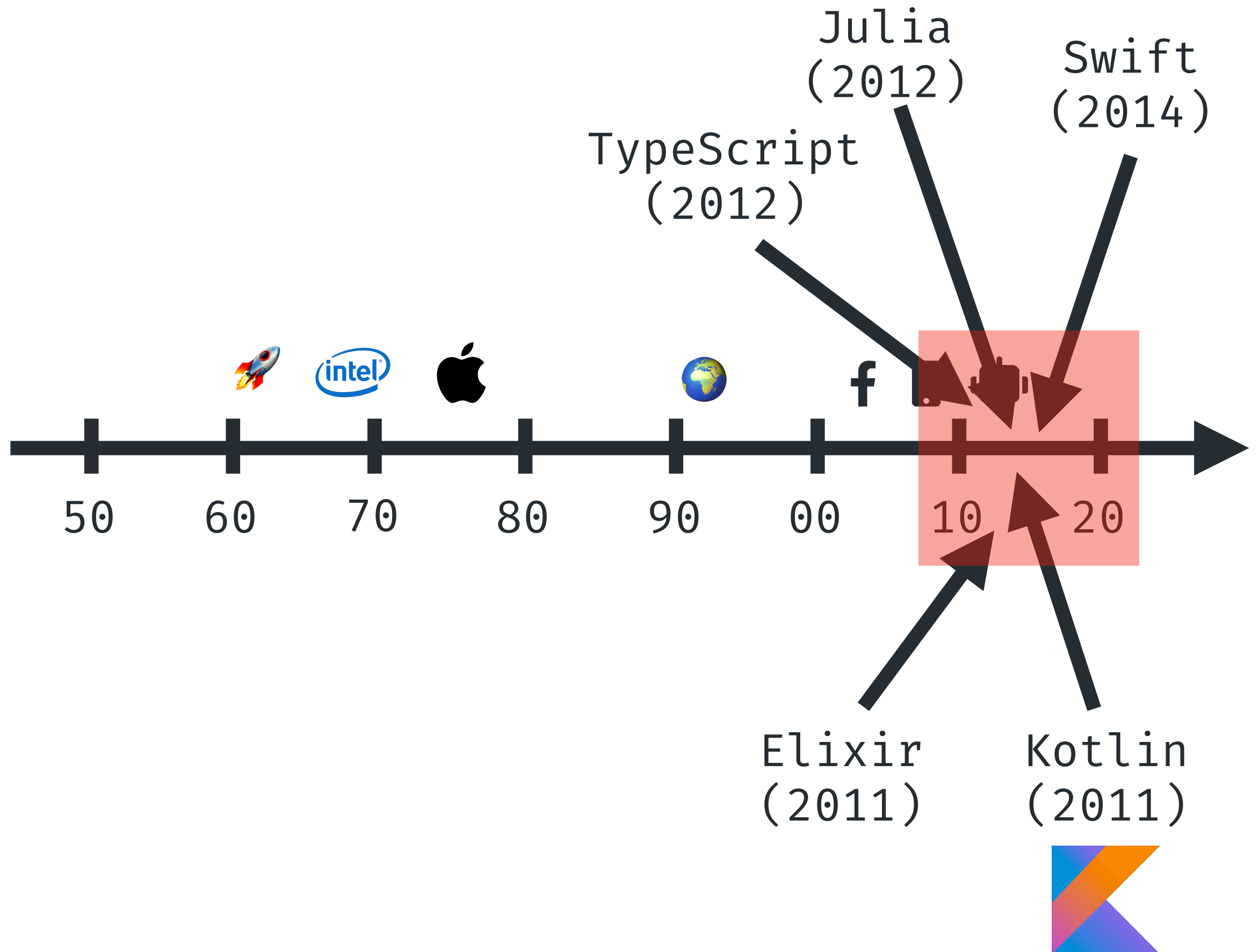


Psy



Apple MacBook Air (2019)





Elixir

```
1 defmodule Binary do
2   def search(list, value), do: search(List.to_tuple(list), value, 0,
    length(list)-1)
3
4   def search(_tuple, _value, low, high) when high < low, do: :not_found
5   def search(tuple, value, low, high) do
6     mid = div(low + high, 2)
7     midval = elem(tuple, mid)
8     cond do
9       value < midval -> search(tuple, value, low, mid-1)
10      value > midval -> search(tuple, value, mid+1, high)
11      value == midval -> mid
12    end
13  end
14 end
15
16 list =
17   [0,1,4,5,6,7,8,9,12,26,45,67,78,90,98,123,211,234,456,769,865,2345,3215,14345
18     ,24324]
19 Enum.each([0,42,45,24324,99999], fn val ->
20   case Binary.search(list, val) do
21     :not_found -> IO.puts "#{val} not found in list"
22     index      -> IO.puts "found #{val} at index #{index}"
23   end
24 end)
```

Kotlin

```
1 enum class Fibonacci {
2     ITERATIVE {
3         override fun invoke(n: Long) = if (n < 2) {
4             n
5         } else {
6             var n1 = 0L
7             var n2 = 1L
8             var i = n
9             do {
10                 val sum = n1 + n2
11                 n1 = n2
12                 n2 = sum
13             } while (i-- > 1)
14             n1
15         }
16     },
17     RECURSIVE {
18         override fun invoke(n: Long): Long = if (n < 2) n else this(n - 1) +
19         this(n - 2)
20     };
21     abstract operator fun invoke(n: Long): Long
22 }
23
24 fun main(a: Array<String>) {
25     val r = 0..30L
26     Fibonacci.values().forEach {
27         print("${it.name}: ")
28         r.forEach { i -> print(" " + it(i)) }
29         println()
30     }
31 }
```

TypeScript

```
1 function replace(input: string, key: number) : string {
2     return input.replace(/([a-z])/g,
3         ($1) => String.fromCharCode(($1.charCodeAt(0) + key + 26 - 97) % 26 +
4         97)
5         ).replace(/([A-Z])/g,
6         ($1) => String.fromCharCode(($1.charCodeAt(0) + key + 26 - 65) % 26 +
7         65));
8 }
9 // test
10 var str = 'The five boxing wizards jump quickly';
11 var encoded = replace(str, 3);
12 var decoded = replace(encoded, -3);
13 console.log('Enciphered: ' + encoded);
14 console.log('Deciphered: ' + decoded);
```

Julia

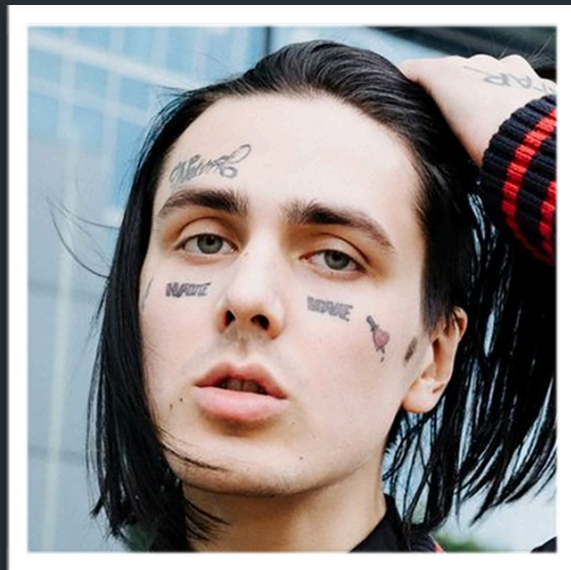


```
1 function binarysearch(lst::Vector{T}, val::T) where T
2     low = 1
3     high = length(lst)
4     while low ≤ high
5         mid = (low + high) ÷ 2
6         if lst[mid] > val
7             high = mid - 1
8         elseif lst[mid] < val
9             low = mid + 1
10        else
11            return mid
12        end
13    end
14    return 0
15 end
```


Swift



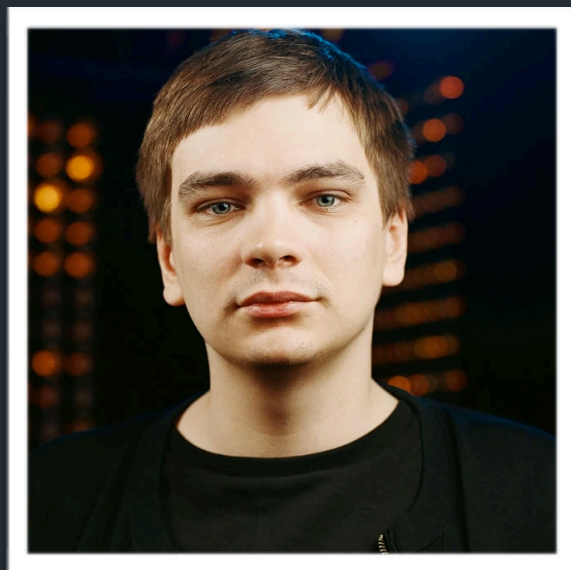
```
1 func binarySearch<T: Comparable>(xs: [T], x: T) -> Int? {
2     var recurse: ((Int, Int) -> Int?)!
3     recurse = {(low, high) in switch (low + high) / 2 {
4         case _ where high < low: return nil
5         case let mid where xs[mid] > x: return recurse(low, mid - 1)
6         case let mid where xs[mid] < x: return recurse(mid + 1, high)
7         case let mid: return mid
8     }}
9     return recurse(0, xs.count - 1)
10 }
```



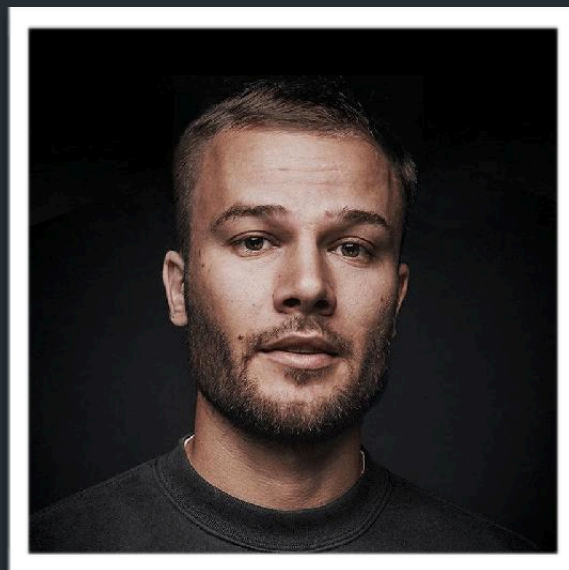
Face



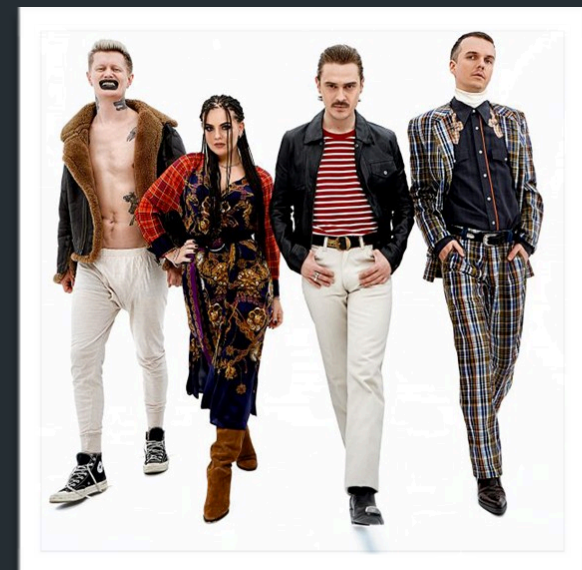
Pharaon



Слава КПСС



Макс Корж

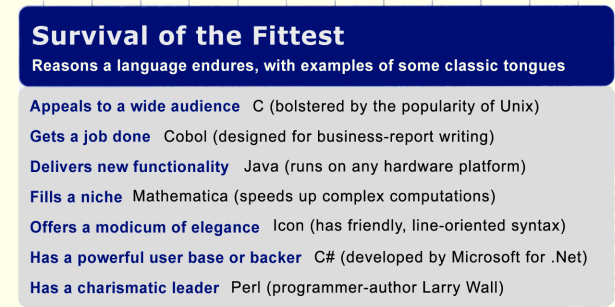


Little Big

Tracing the roots of computer languages through the ages

An ad hoc collection of engineers-electronic lexicographers, if you will-aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua francas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Year	Status	Description
1954	Year Introduced	
	Active	thousands of users
	Protected	taught at universities; compilers available
	Endangered	usage dropping off
	Extinct	no known active users or up-to-date compilers
	Lineage continues	



Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

