Bend vs Mojo

Гурбанов Михаил

t.me/mygurbanov

Михаил Гурбанов

– работаю в Райфе
– старший разработчик
– развиваю сообщество
– выступаю в разных местах

Питону нужна замена?

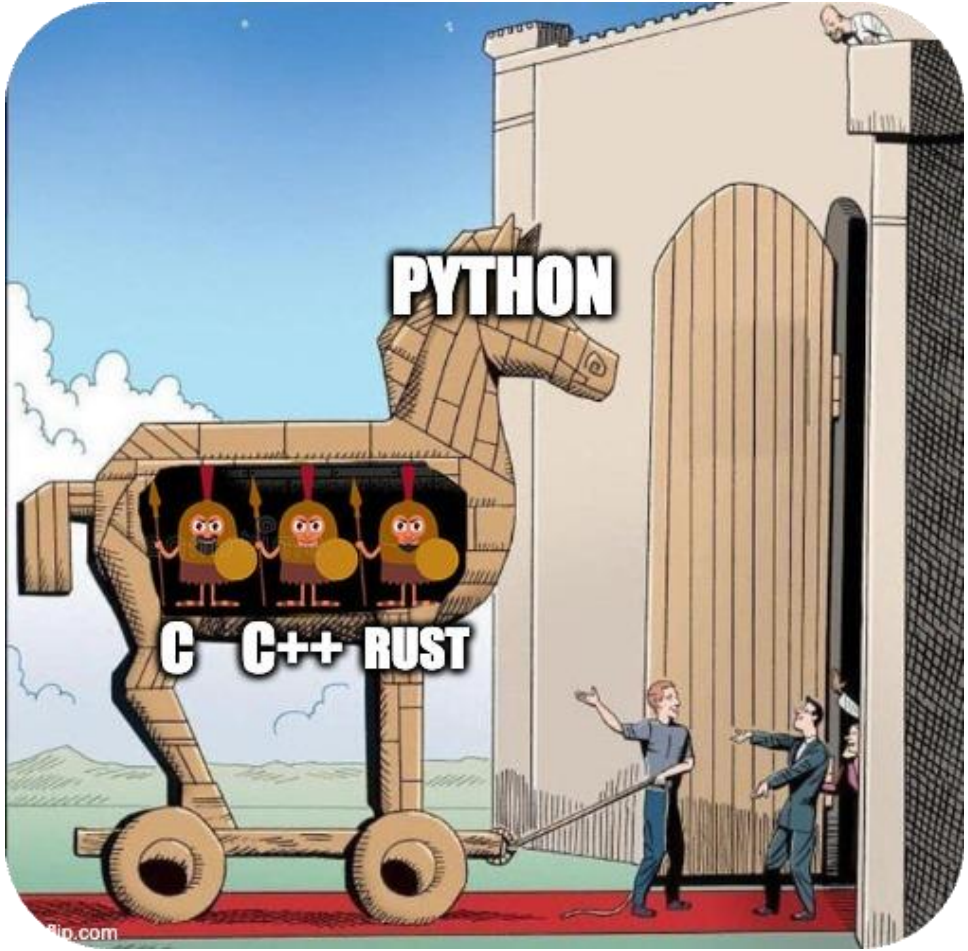
Scientific computing

- CPU/GPU bound
  - DS/ML - много матриц
  - Криптография - шифровка и дешифровка
  - Численные методы - еще больше матриц
  - Сжатие и распаковка

Параллелизм

```python
import multiprocessing

def main():
    numbers = [1, 2, 3, 4, 5]

    process = multiprocessing.Process(
      target=square_numbers, args=(numbers,)
    )

    process.start()

    process.join()
```

- В DS не засунешь
- Тяжелый код
- Python медленный

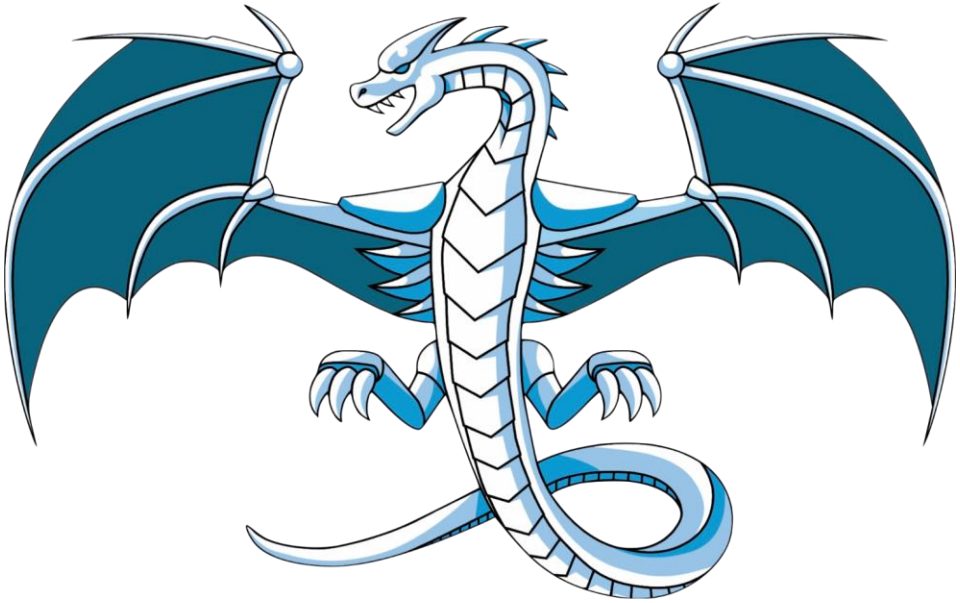

PYTHON
C C++ RUST

Как еще можно ускорить

- Купить еще больше железяк
- ~~Помолиться Гвидо~~ Ускорить CPython
- Использовать PyPy, Cython или другие
- Использовать либы на C/C++/Rust

Chris Lattner

Chris Lattner

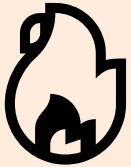
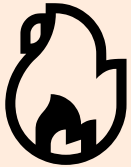
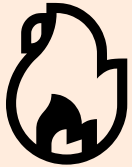


Chris Lattner

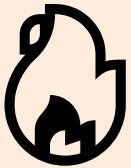
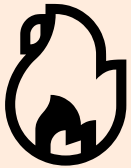
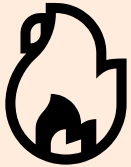
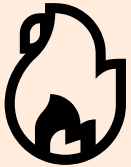
Chris Lattner

Modular

- Туса с ИИ платформой
- Максимальный performance
- Должно быть “хорошо”
- Не хватило C/C++/Rust

Mojo

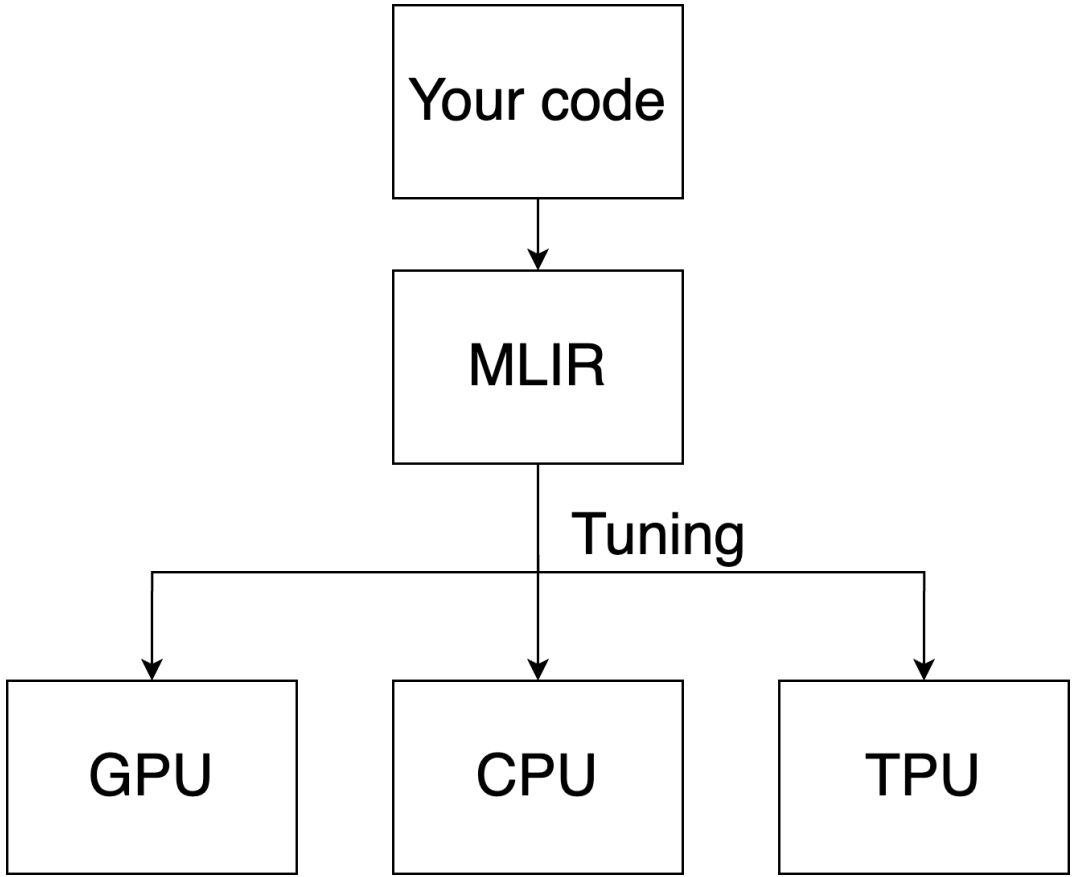
Mojo

LANGUAGES	TIME (S)*	SPEEDUP VS PYTHON
Python 3.10.9	1027s	1X
PYPY	46.1s	22x
Scalar C++	0.20s	5,000x
Mojo🔥	0.03s	68,000x

MLIR

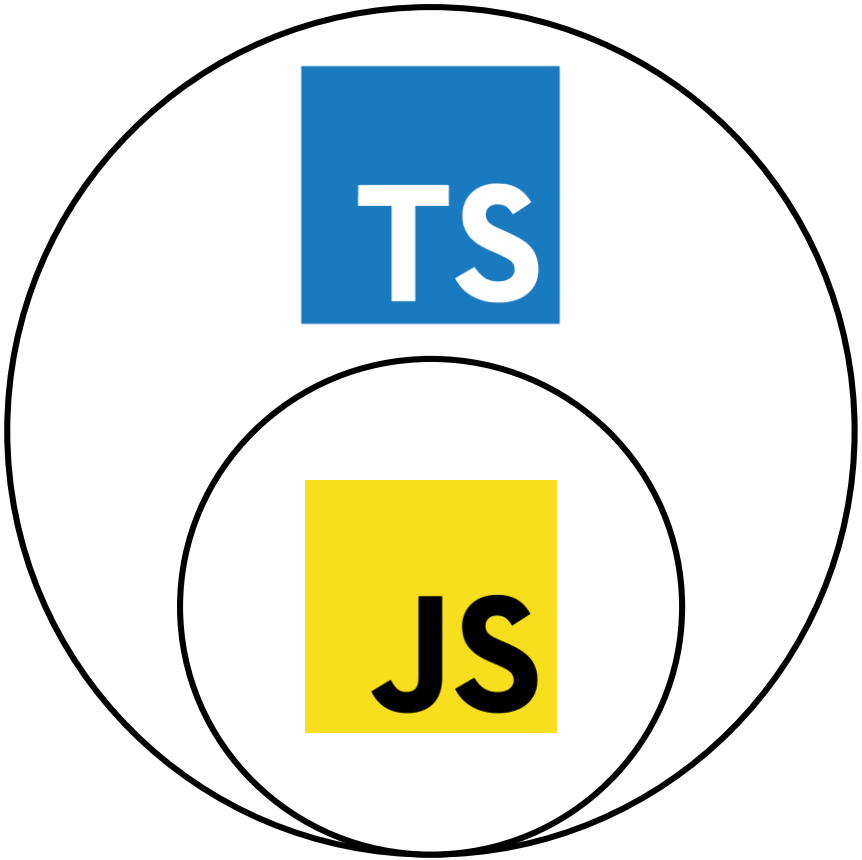
- Multi Level Intermediate Representation
- Фреймворк для создания оптимизационных компиляторов
- Можно компилировать код под любое железо

MLIR



Mojo

Mojo



Mojo

- Не совсем superset
- Вообще другой язык
- Компилируемый язык
- Статическая типизация
- Пока не совсем opensource

Mojo: Типы данных

Простые типы	Сложные типы
Int8..64	List
Uint8..64	Dict
Float16..64	Set
String	Optional
Boolean	
SIMD	

Mojo: Переменные

```
var number = 42
var string: String = "Mojo"

number = "some-string" # Error
```

Mojo: Переменные

```
number = 42
string = "Mojo"

number = "some-string" # Error
```

Mojo: Функции

```
def greet(name):
    greeting = "Hello, " + name + "!"
    return greeting
```

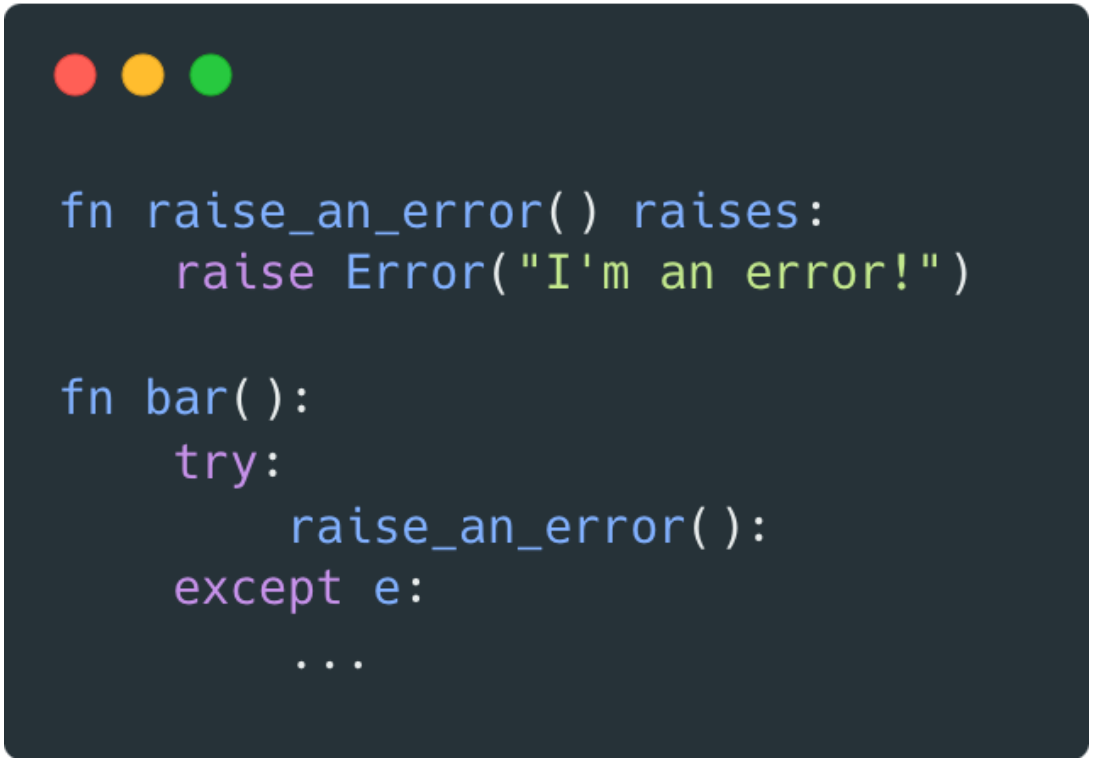
Mojo: Функции

```
fn  greet(name: String) -> String:
    greeting = "Hello, " + name + "!"
    return greeting
```

Mojo: Структуры

```
struct MyPair:
    var first: Int
    var second: Int

    fn __init__(inout self, first: Int, second: Int):
        self.first = first
        self.second = second
```

Mojo: Ошибки

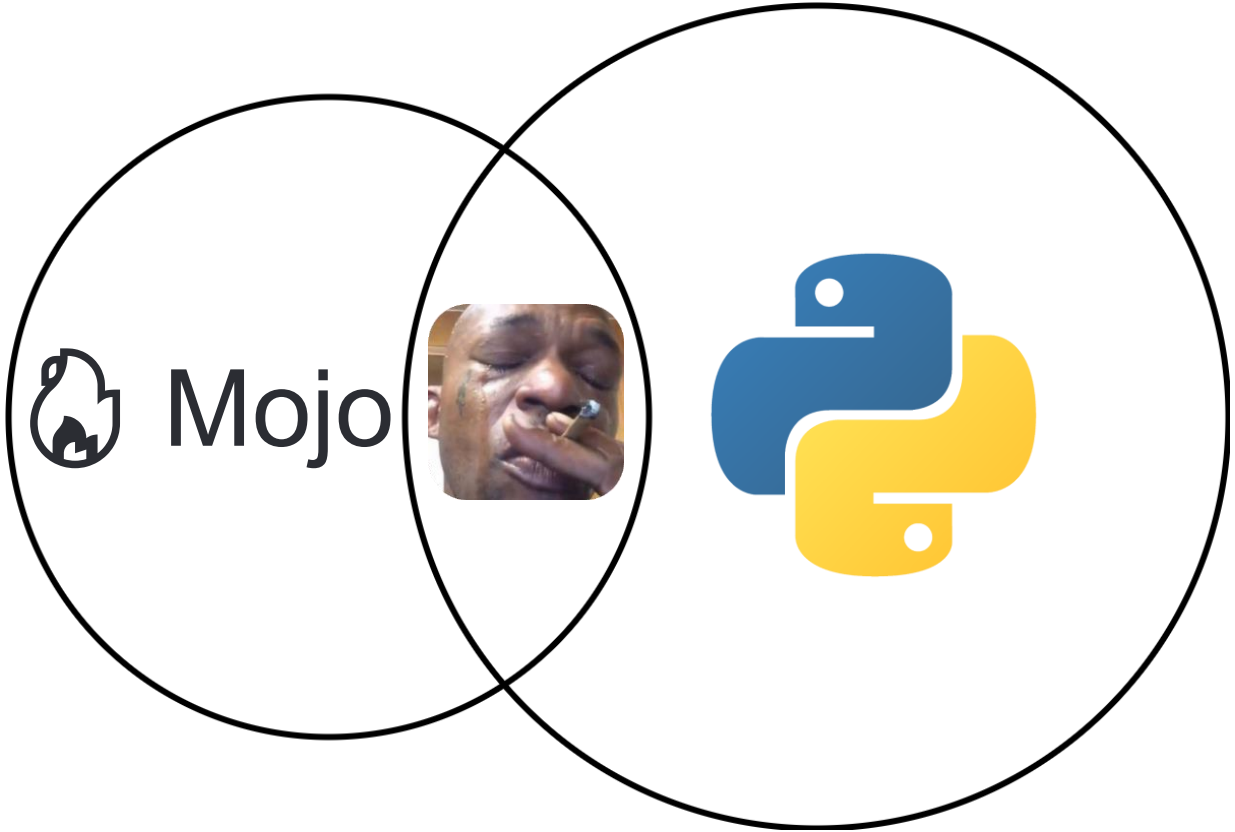


```
fn raise_an_error() raises:
    raise Error("I'm an error!")

fn bar():
    try:
        raise_an_error():
    except e:
        ...
```

Mojo: использование Python

```
from python import Python

fn use_array() raises:
    var np = Python.import_module("numpy")

    var array = np.array([1, 2, 3])
    print(array)
```

Mojo



Mojo: Бенчи

- Mac M1
- numpy.dot
- Mojo

$$C_{ij} = \sum_{n=1}^{k} A_{in} * B_{nj}$$

Mojo: Бенчи

```
struct Matrix[rows: Int, cols: Int]:
    ...

fn matmul_naive(C: Matrix, A: Matrix, B: Matrix):
    for m in range(C.rows):
        for k in range(A.cols):
            for n in range(C.cols):
                C[m, n] += A[m, k] * B[k, n]
```

Mojo: Бенчи

```
struct Matrix[rows: Int, cols: Int]:
    ...

fn matmul_naive(C: Matrix, A: Matrix, B: Matrix):
    for m in range(C.rows):
        for k in range(A.cols):
            for n in range(C.cols):
                C[m, n] += A[m, k] * B[k, n]
```

Mojo в 50 раз медленнее NumPy

Mojo: Бенчи

```
from algorithm import vectorize

fn matmul_vectorized_1(C: Matrix, A: Matrix, B: Matrix):
    for m in range(C.rows):
        for k in range(A.cols):
            @parameter
            fn dot[nelts: Int](n: Int):
                C.store(
                  m, n,
                  C.load[nelts](m, n) + A[m, k] * B.load[nelts](k, n)
                )
            vectorize[dot, nelts, size = C.cols]()
```

Mojo: Бенчи

```
from algorithm import vectorize

fn matmul_vectorized_1(C: Matrix, A: Matrix, B: Matrix):
    for m in range(C.rows):
        for k in range(A.cols):
            @parameter
            fn dot[nelts: Int](n: Int):
                C.store(
                  m, n,
                  C.load[nelts](m, n) + A[m, k] * B.load[nelts](k, n)
                )
            vectorize[dot, nelts, size = C.cols]()
```

🔥 Mojo в 10 раз медленнее NumPy

Mojo: Бенчи

```
from algorithm import parallelize

fn matmul_parallelized(C: Matrix, A: Matrix, B: Matrix):
    @parameter
    fn calc_row(m: Int):
        for k in range(A.cols):
            @parameter
            fn dot[nelts : Int](n : Int):
                C.store[nelts](
                  m, n,
                  C.load[nelts](m,n) + A[m,k] * B.load[nelts](k,n)
                )
            vectorize[dot, nelts, size = C.cols]()
    parallelize[calc_row](C.rows, C.rows)
```

Mojo: Бенчи

```
from algorithm import parallelize

fn matmul_parallelized(C: Matrix, A: Matrix, B: Matrix):
    @parameter
    fn calc_row(m: Int):
        for k in range(A.cols):
            @parameter
            fn dot[nelts : Int](n : Int):
                C.store[nelts](
                  m, n,
                  C.load[nelts](m,n) + A[m,k] * B.load[nelts](k,n)
                )
            vectorize[dot, nelts, size = C.cols]()
    parallelize[calc_row](C.rows, C.rows)
```

Mojo в 2.5 раз медленнее NumPy

Mojo: Бенчи

```
fn matmul_unrolled_parallelized(C: Matrix, A: Matrix, B: Matrix):
    @parameter
    fn calc_row(m: Int):
        @parameter
        fn calc_tile[tile_x: Int, tile_y: Int](x: Int, y: Int):
            for k in range(y, y + tile_y):
                @parameter
                fn dot[nelts: Int](n: Int):
                    C.store(
                      m, n + x,
                      C.load[nelts](m, n + x) + A[m, k] * B.load[nelts](k, n + x)
                    )

                # Vectorize by nelts and unroll by tile_x/nelts
                # Here unroll factor is 4
                alias unroll_factor = tile_x // nelts
                vectorize[dot, nelts, size=tile_x, unroll_factor=unroll_factor]()

    parallelize[calc_row](C.rows, C.rows)
```

Mojo: Бенчи

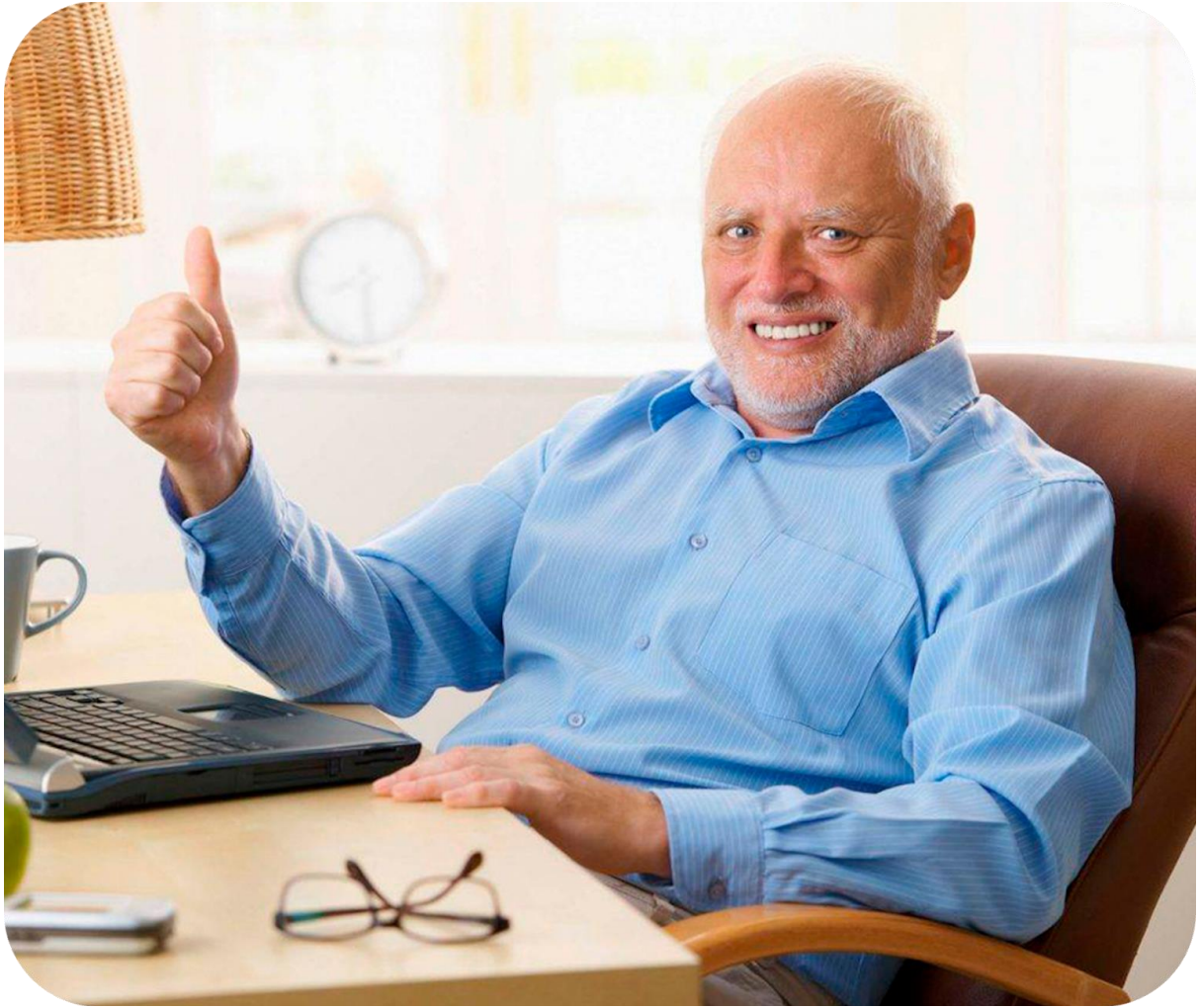
```
fn matmul_unrolled_parallelized(C: Matrix, A: Matrix, B: Matrix):
    @parameter
    fn calc_row(m: Int):
        @parameter
        fn calc_tile[tile_x: Int, tile_y: Int](x: Int, y: Int):
            for k in range(y, y + tile_y):
                @parameter
                fn dot[nelts: Int](n: Int):
                    C.store(
                      m, n + x,
                      C.load[nelts](m, n + x) + A[m, k] * B.load[nelts](k, n + x)
                    )

                # Vectorize by nelts and unroll by tile_x/nelts
                # Here unroll factor is 4
                alias unroll_factor = tile_x // nelts
                vectorize[dot, nelts, size=tile_x, unroll_factor=unroll_factor]()

    parallelize[calc_row](C.rows, C.rows)
```

🔥 Mojo в 87 раз быстрее numpy

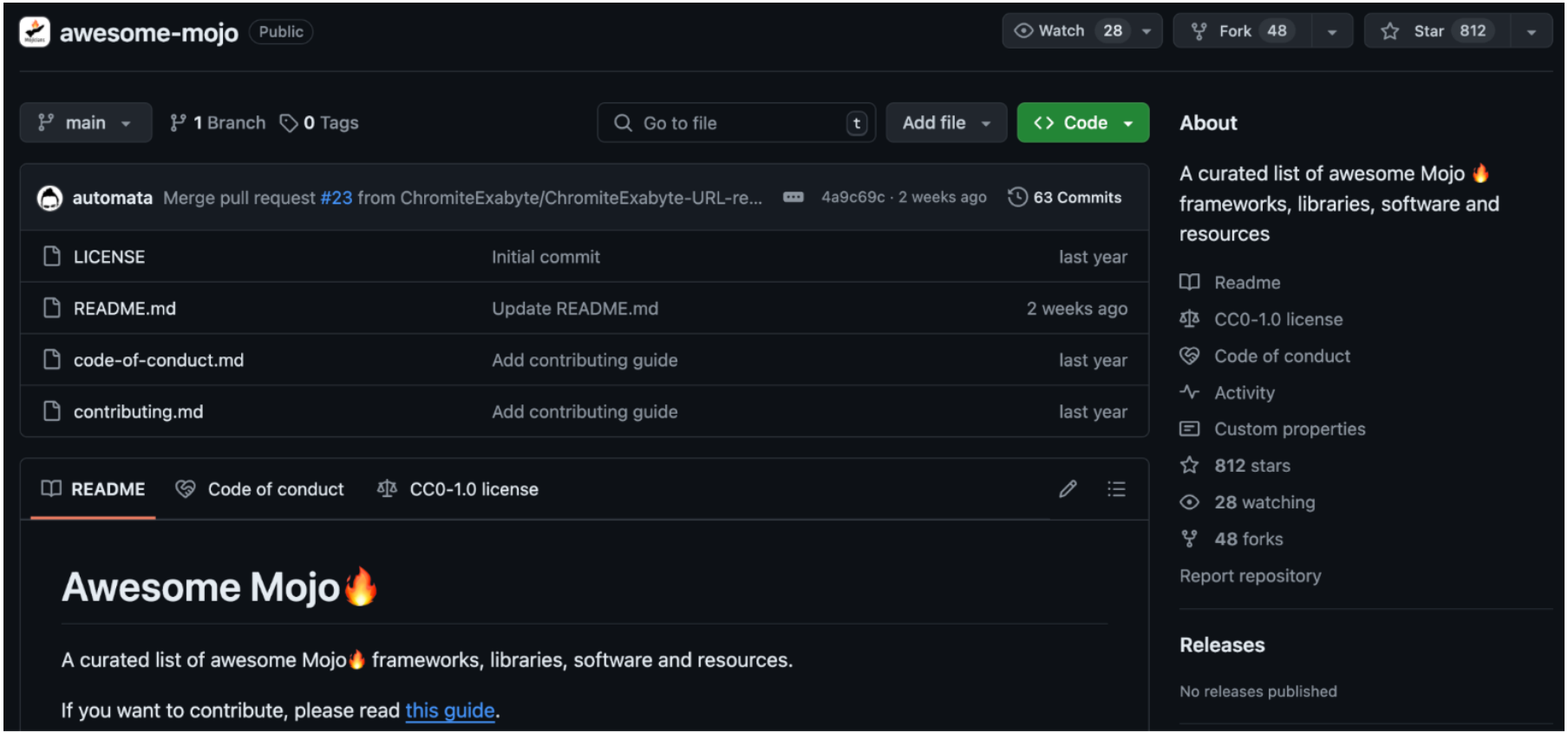
Mojo: Бенчи

```
fn matmul_unrolled_parallelized(C: Matrix, A: Matrix, B: Matrix):
    @parameter
    fn calc_row(m: Int):
        @parameter
        fn calc_tile[tile_x: Int, tile_y: Int](x: Int, y: Int):
            for k in range(y, y + tile_y):
                @parameter
                fn dot[nelts: Int](n: Int):
                    C.store(
                      m, n + x,
                      C.load[nelts](m, n + x) + A[m, k] * B.load[nelts](k, n + x)
                    )

                # Vectorize by nelts and unroll by tile_x/nelts
                # Here unroll factor is 4
                alias unroll_factor = tile_x // nelts
                vectorize[dot, nelts, size=tile_x, unroll_factor=unroll_factor]()

    parallelize[calc_row](C.rows, C.rows)
```

Mojo в 7 раз быстрее jax + jit



Mojo: Эко-система



Mojo: Сообщество

@modular_ai

company/modular_ai

@modularinc

modular

Mojo: Установка

```
$ curl -ssL https://magic.modular.com/069ba1af-3b4f-488b-b881-5566868898ef | bash

$ magic init my-project

$ magic run mojo --version
```

Mojo: Установка

```
$ curl -ssL https://magic.modular.com/069ba1af-3b4f-488b-b881-5566868898ef | bash

$ magic init my-project

$ magic run mojo --version

$ magic add max
```

MAX

A new framework for Gen AI, and the best way to deploy PyTorch

SERVING LIBRARY

INFERENCE RUNTIME

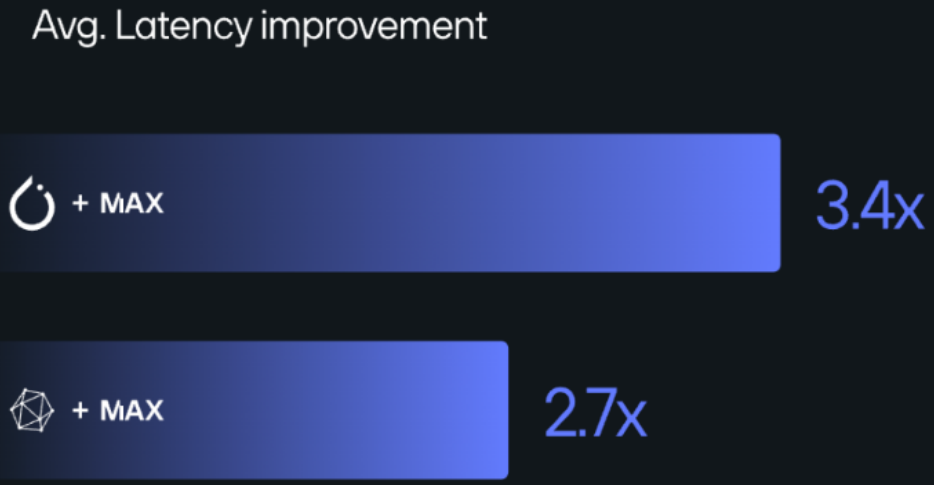
MODEL PIPELINE

CLI

EXTEND EASILY IN

SCALE IN THE CLOUD

The best way to deploy PyTorch



*CPU Benchmarks.

SOTA performance in just 3 lines of code

Drop in your PyTorch or ONNX models and get an instant boost in performance with our next generation inference runtime for CPUs and GPUs

See for yourself >

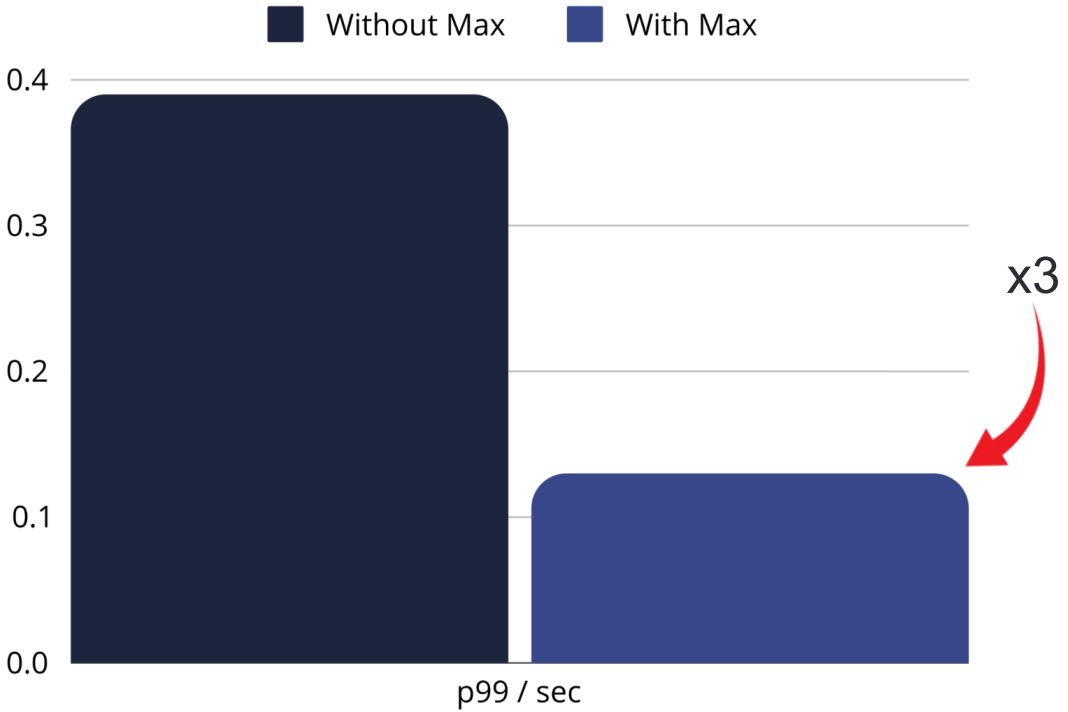
Mojo: Max

```
from max import engine

# Load your model:
session = engine.InferenceSession()
model = session.load(model_path)

# Prepare the inputs, then run an inference:
outputs = model.execute(**inputs)

# Process the output here.
```

Mojo: Max CPU benchmarks

- Mac M1
- torchscripted BERT
- Emotion dataset
- Python / Max engine

Mojo: Max CPU benchmarks



MAX on GPU waiting list

Be the first to get lightning fast inference speed on your GPUs. Be the envy of all your competitors and lower your compute spend.

First name*

First name ...

Last name*

Last name ...

Email*

Enter work email ...

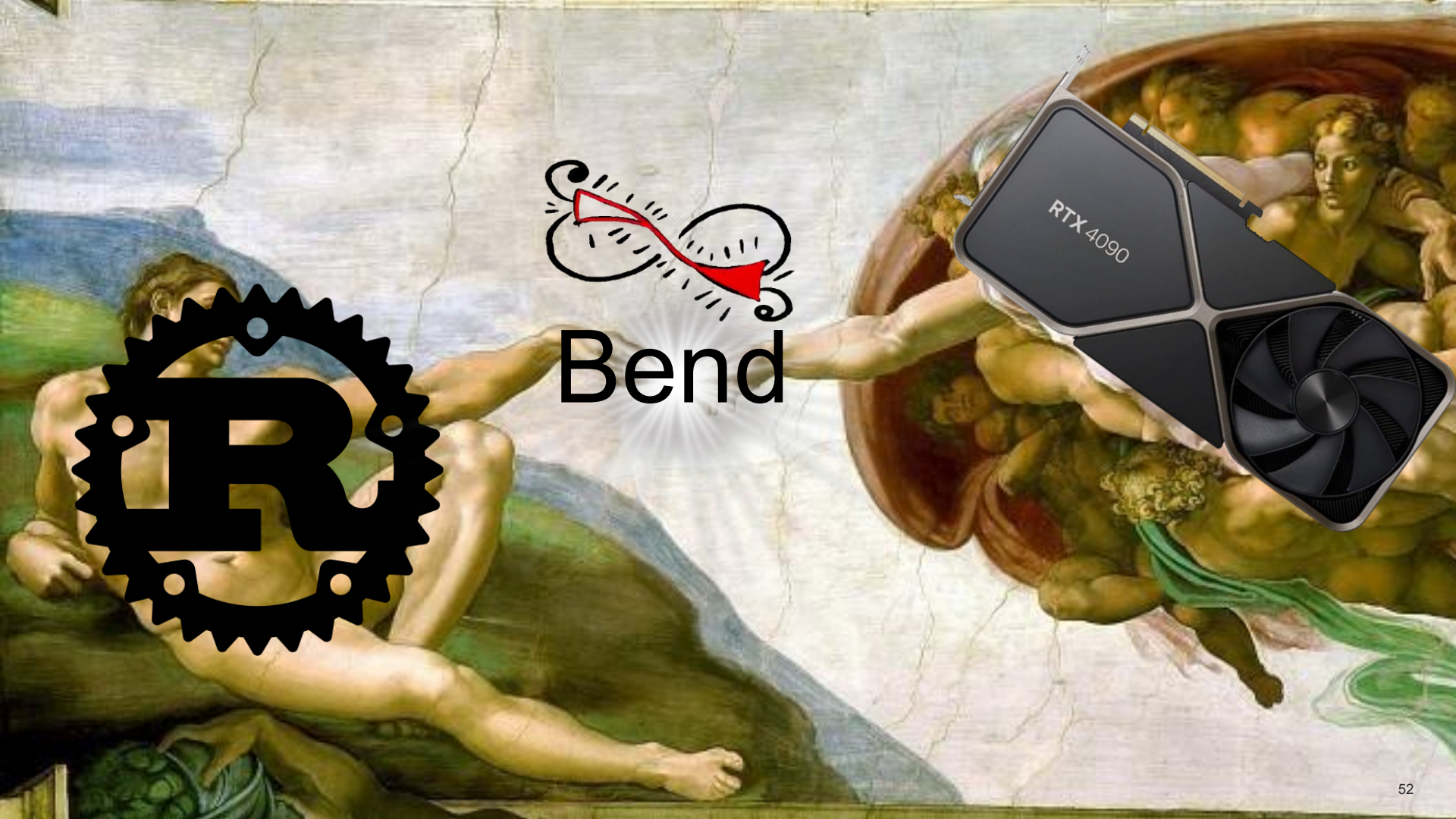
Job title*

Enter your job title ...

Submit

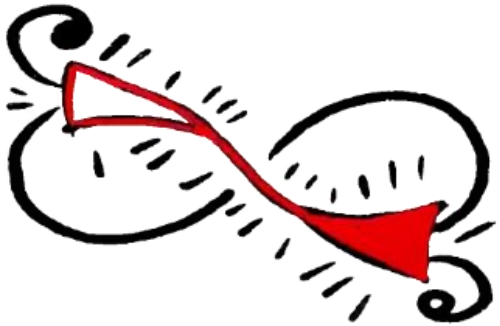
Mojo: Вывод

- Намного лучше, чем раньше
- Развивающееся сообщество
- Хотят внедриться в ядро pytorch
- В прод бы пока не тащил
- MAX выглядит очень круто


Bend
RTX 4090

Bend

- Совсем молодой
- Massively Parallel
- In Rust ⚡
- Can parallel - will parallel
- CPU / GPU



Interaction Combinators

Yves Lafont*

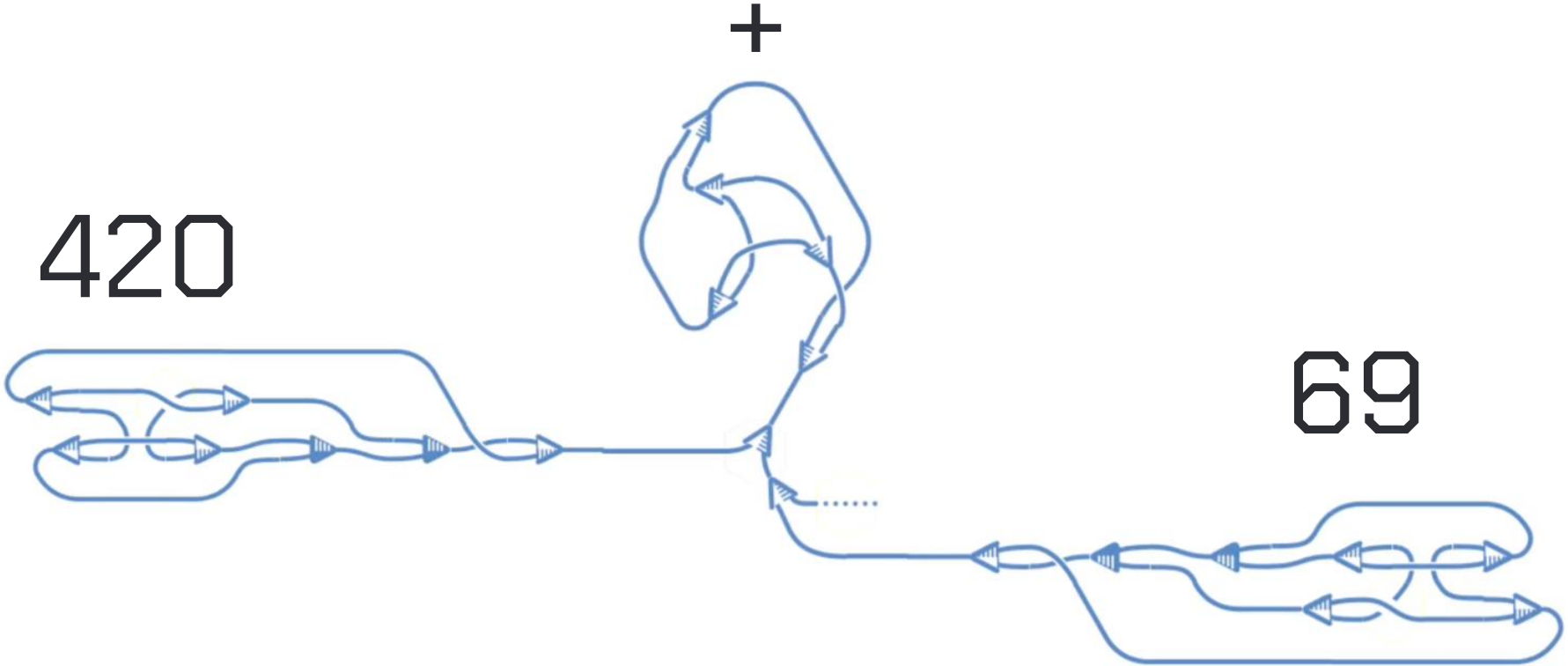
Institut de Mathématiques de Luminy, UPR 9016 du CNRS,
163 avenue de Luminy, case 930, 13288 Marseille Cedex 9, France
E-mail: lafont@iml.univ-mrs.fr

It is shown that a very simple system of *interaction combinators*, with only three symbols and six rules, is a universal model of distributed computation, in a sense that will be made precise. This paper is the continuation of the author's work on *interaction nets*, inspired by Girard's proof nets for *linear logic*, but no preliminary knowledge of these topics is required for its reading.

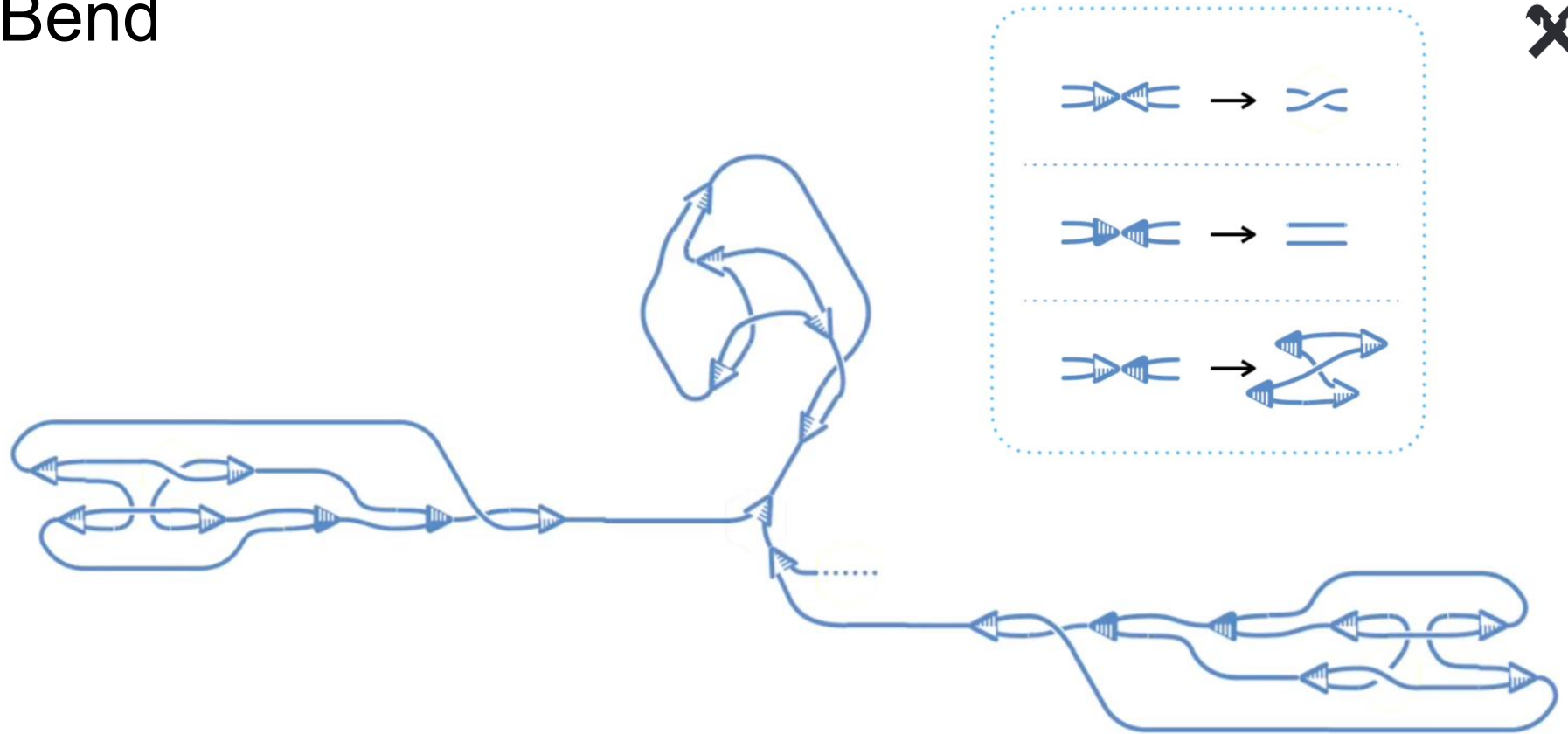
Bend

420 + 69

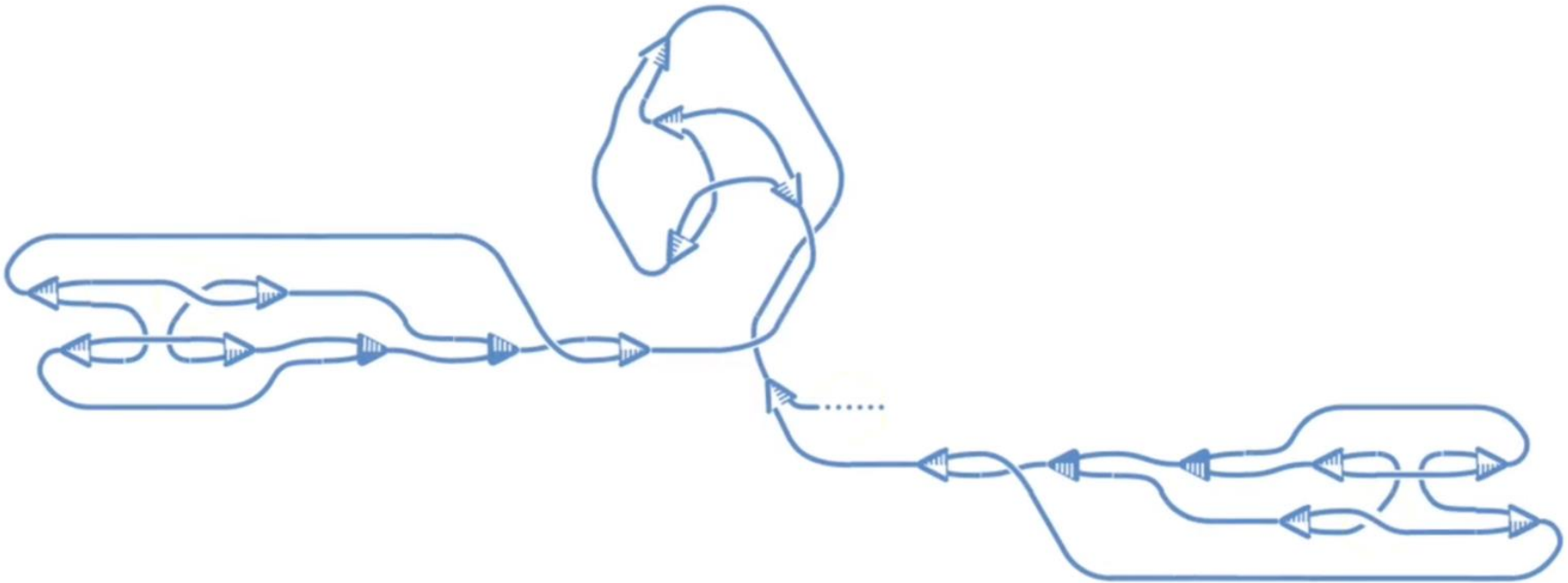
Bend



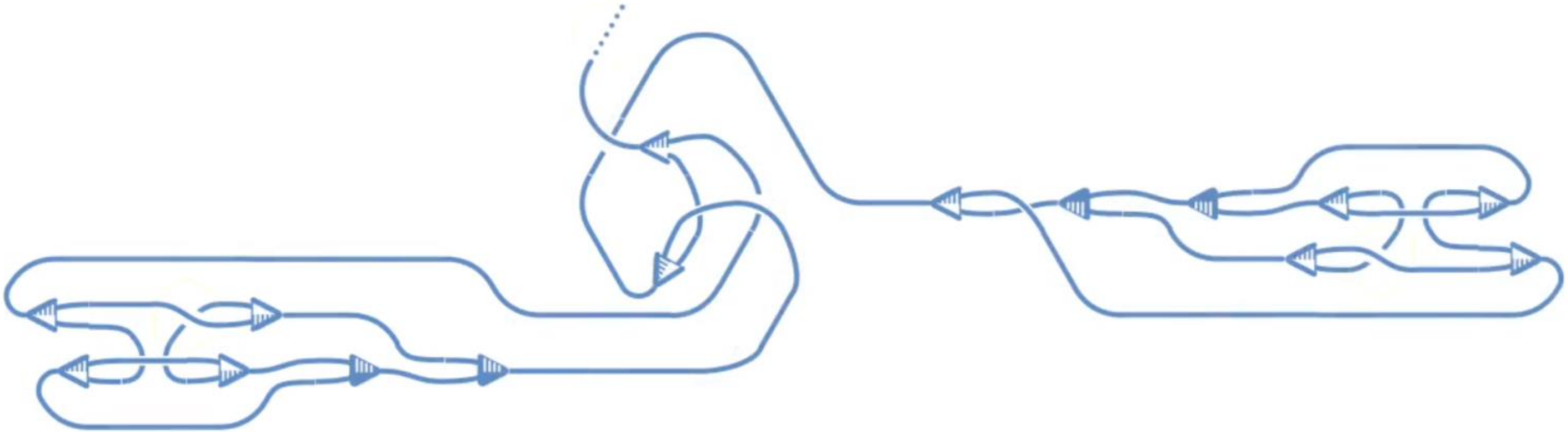
Bend



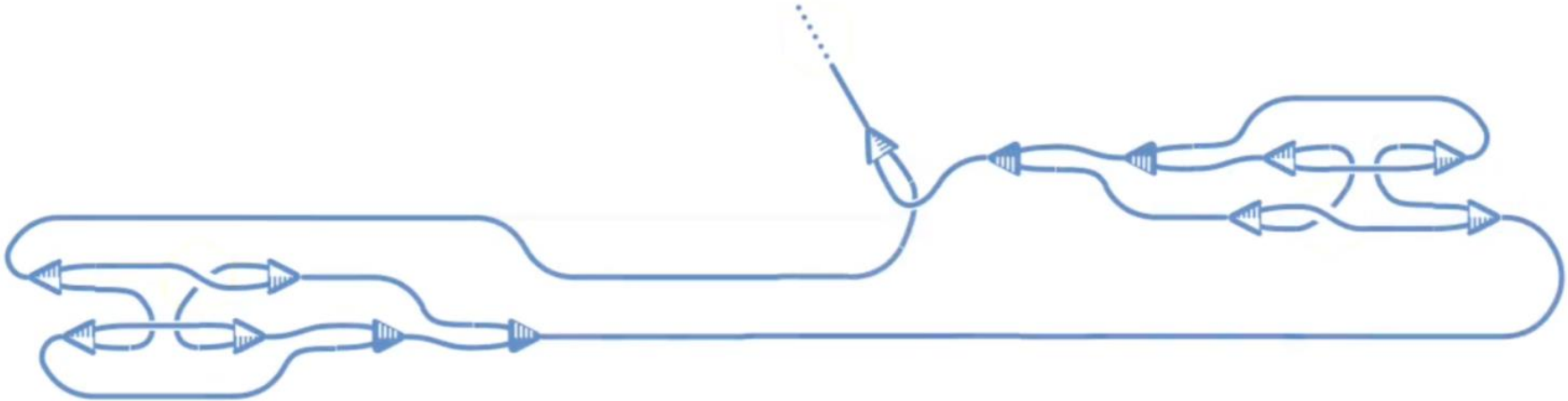
Bend



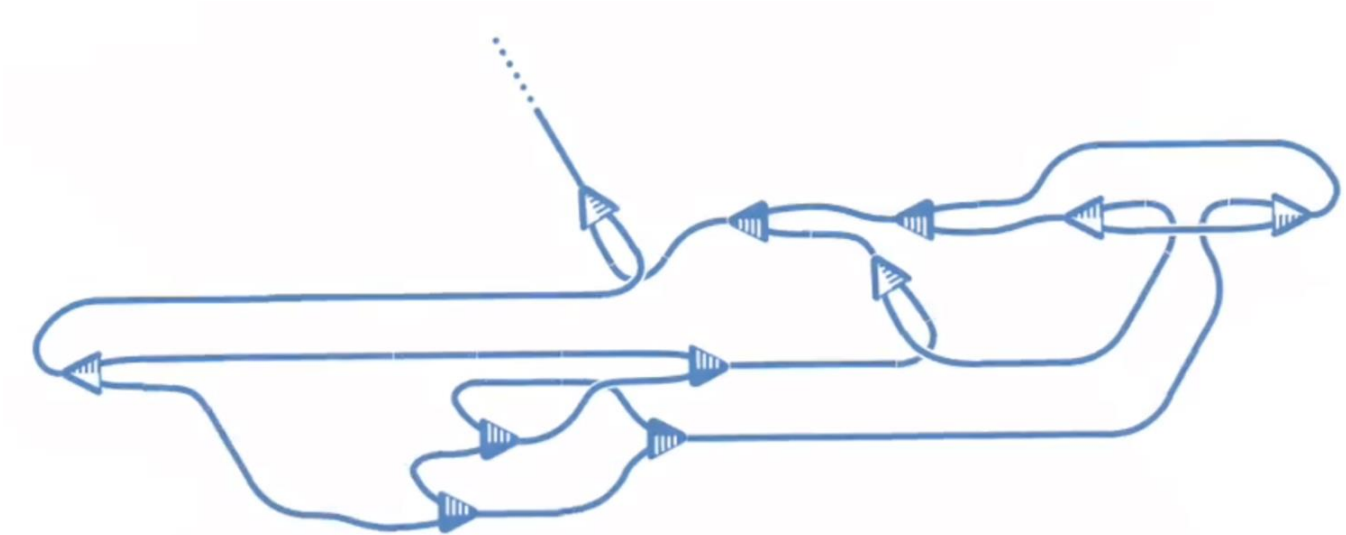
Bend



Bend



Bend



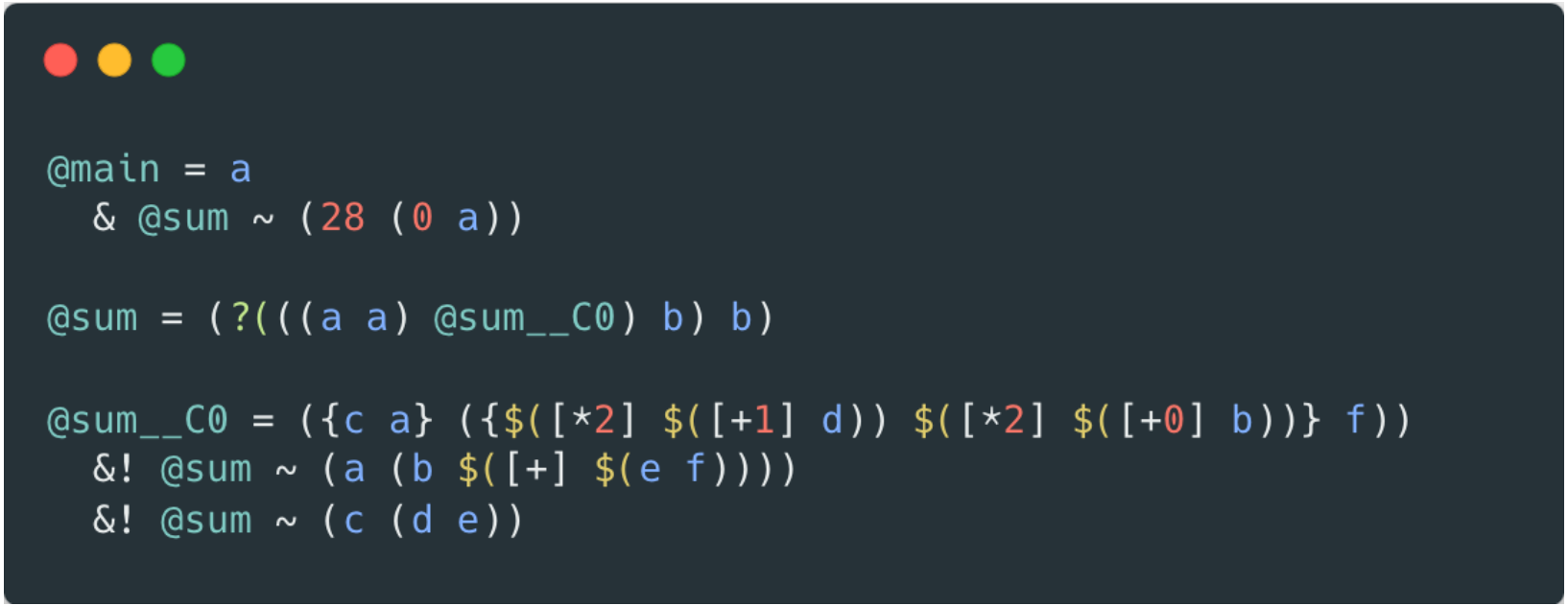
Bend

$$420 + 69 = 489$$

Bend: HVM

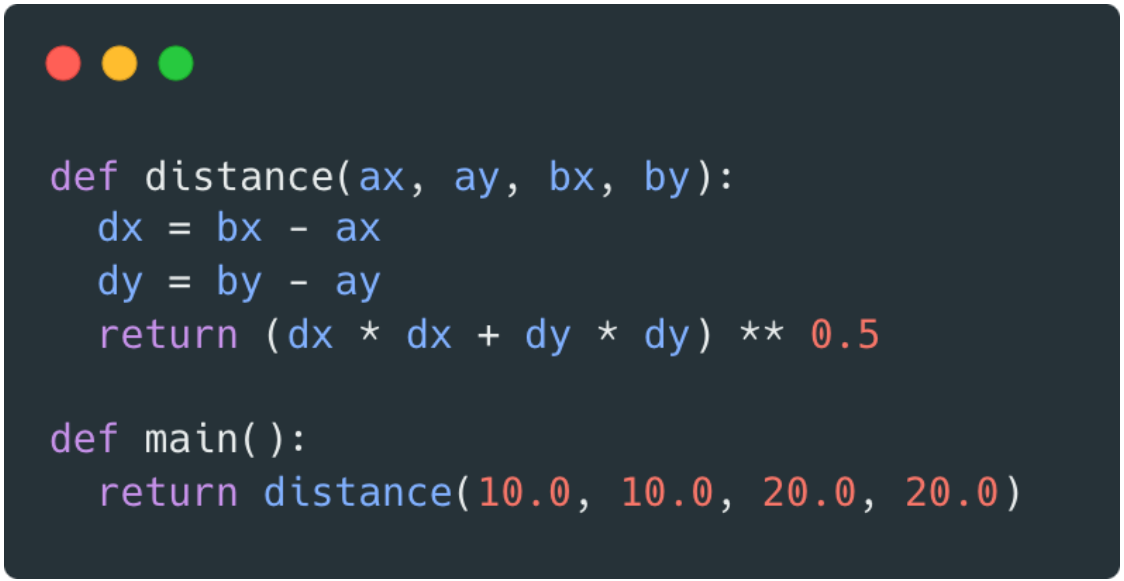
- Higher order Virtual Machine
- На основе Interaction Combinators
- Является параллельным рантаймом
- Для взаимодействия с ним и сделан Bend
- Можно обращаться из Bend напрямую

Bend: HVM



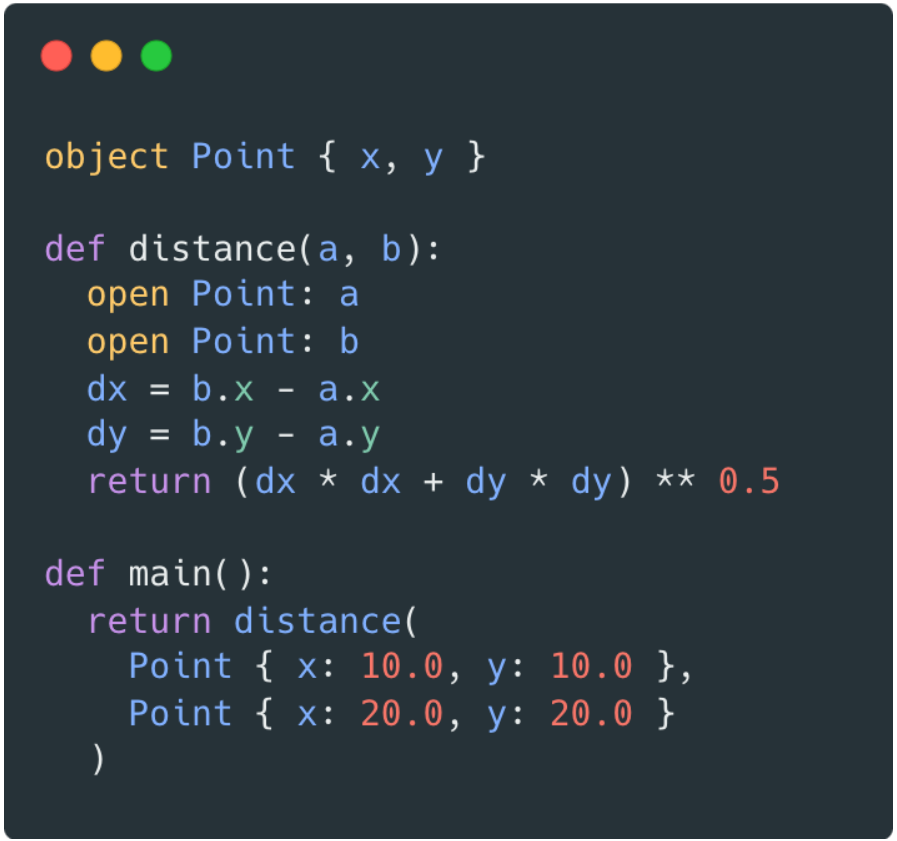
```
@main = a
  & @sum ~ (28 (0 a))

@sum = (?(((a a) @sum__C0) b) b)

@sum__C0 = ({c a} ({$([*2] $([+1] d)) $([*2] $([+0] b))} f))
  &! @sum ~ (a (b $([+] $(e f))))
  &! @sum ~ (c (d e))
```

Bend: Функции



```
def distance(ax, ay, bx, by):
  dx = bx - ax
  dy = by - ay
  return (dx * dx + dy * dy) ** 0.5

def main():
  return distance(10.0, 10.0, 20.0, 20.0)
```

Bend: Объекты



```
object Point { x, y }

def distance(a, b):
  open Point: a
  open Point: b
  dx = b.x - a.x
  dy = b.y - a.y
  return (dx * dx + dy * dy) ** 0.5

def main():
  return distance(
    Point { x: 10.0, y: 10.0 },
    Point { x: 20.0, y: 20.0 }
  )
```

Bend: Типы

```
type Shape:
  Circle { radius }
  Rectangle { width, height }

def area(shape):
  match shape:
    case Shape/Circle:
      return 3.14 * shape.radius ** 2.0
    case Shape/Rectangle:
      return shape.width * shape.height

def main:
  return area(Shape/Circle { radius: 10.0 })
```

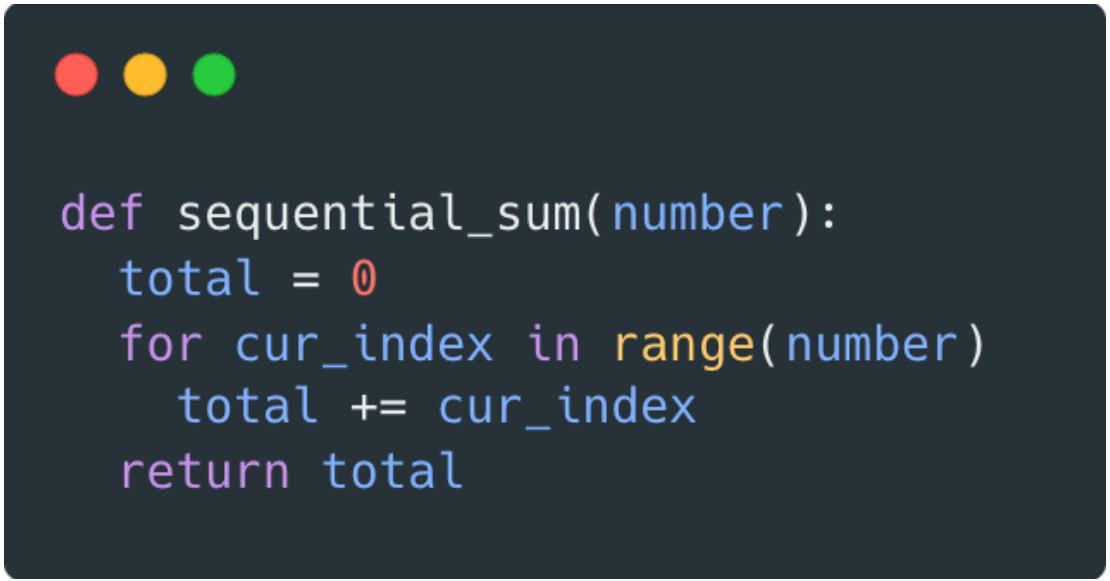
Bend: Неизменяемость

```
def parity(x):
  result = "odd"
  if x % 2 == 0:
    result = "even"
  return result
```

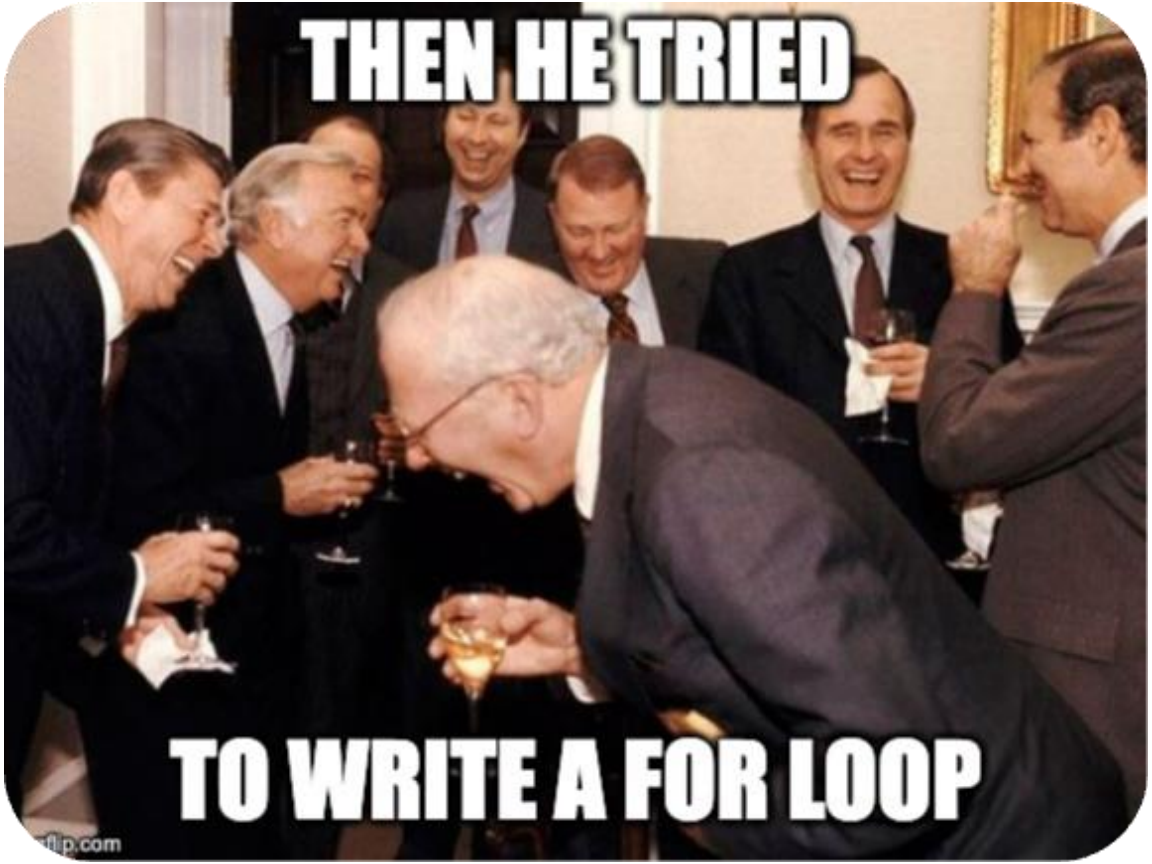
Bend: Неизменяемость

```
def partity(x):
  if x % 2 == 0:
    return "even"
  else:
    return "odd"
```

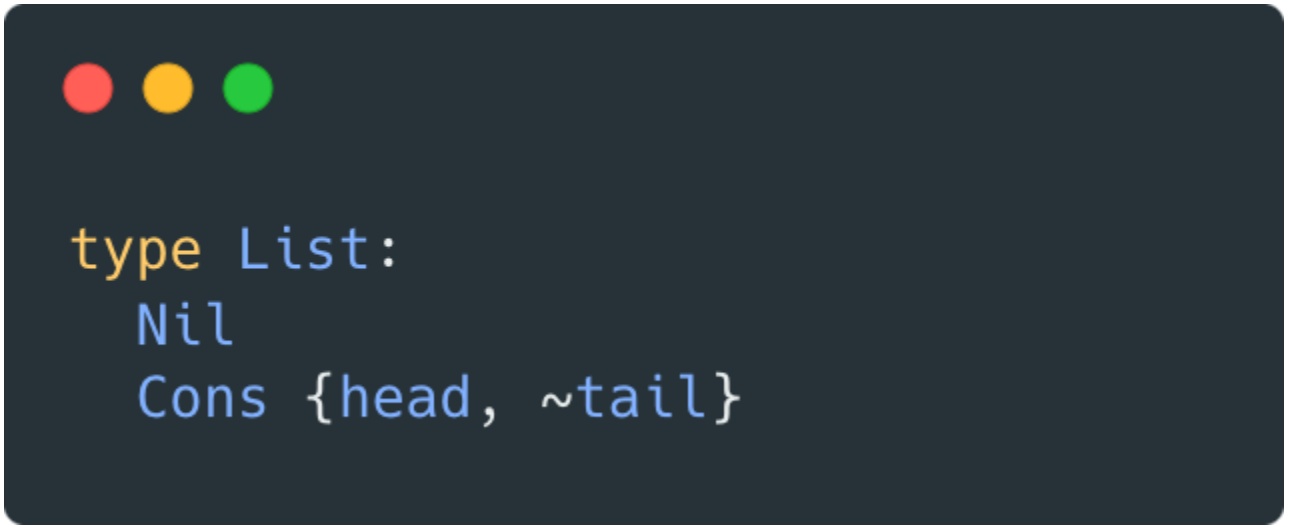
Bend: Неизменяемость



```
def sequential_sum(number):
  total = 0
  for cur_index in range(number)
    total += cur_index
  return total
```


THEN HE TRIED
TO WRITE A FOR LOOP

Bend: List

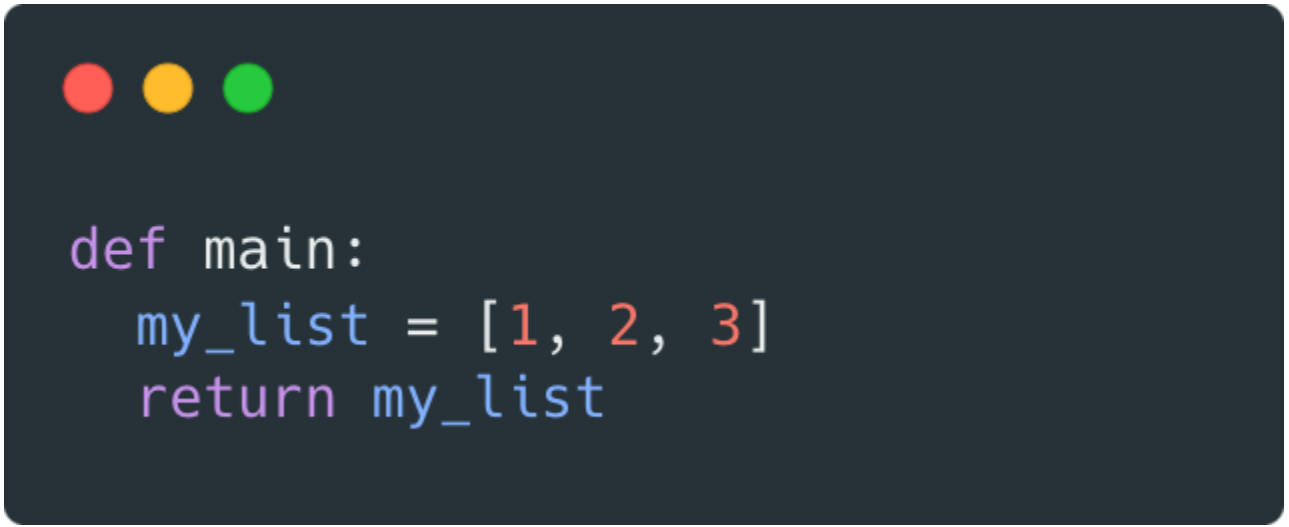


```
type List:
  Nil
  Cons {head, ~tail}
```

Bend: List

```
def main:
  my_list = List/Cons {
       head: 1, tail: List/Cons {
           head: 2, tail: List/Cons {
               head: 3, tail: List/Nil
           }
       }
   }
  return my_list
```

Bend: List



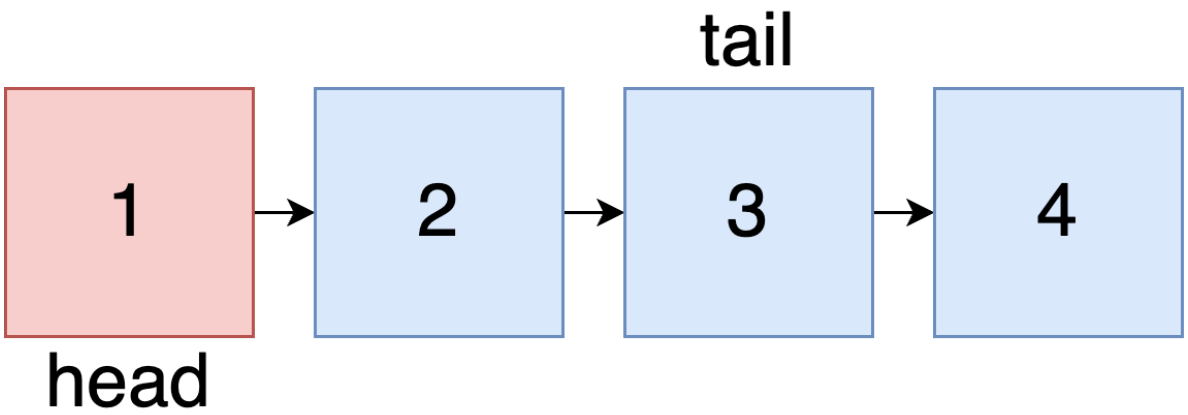
```
def main:
  my_list = [1, 2, 3]
  return my_list
```

Bend: Fold

```
def list_sum(input_list):
  fold input_list:
    case List/Cons:
      return input_list.head + input_list.tail
    case List/Nil:
      return 0
```

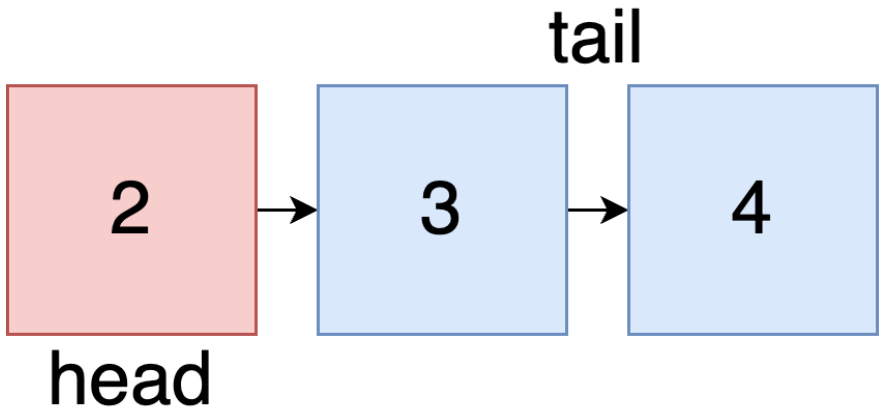
Bend: Fold

1) (1 + tail)



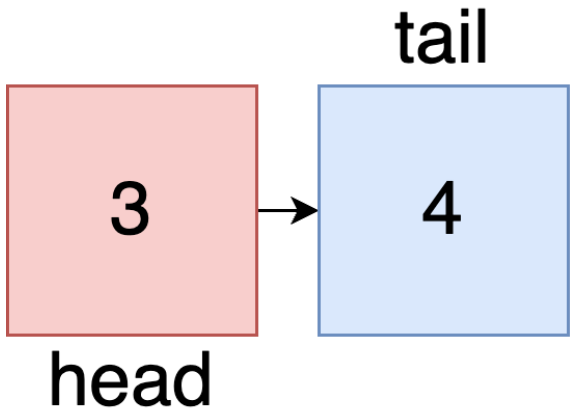
Bend: Fold

1) (1 + tail)
2) (1 + (2 + tail))



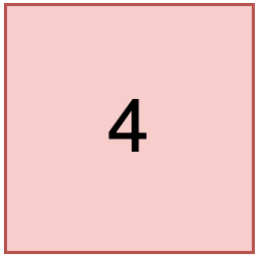
Bend: Fold

1) (1 + tail)
2) (1 + (2 + tail))
3) (1 + (2 + (3 + tail)))



Bend: Fold

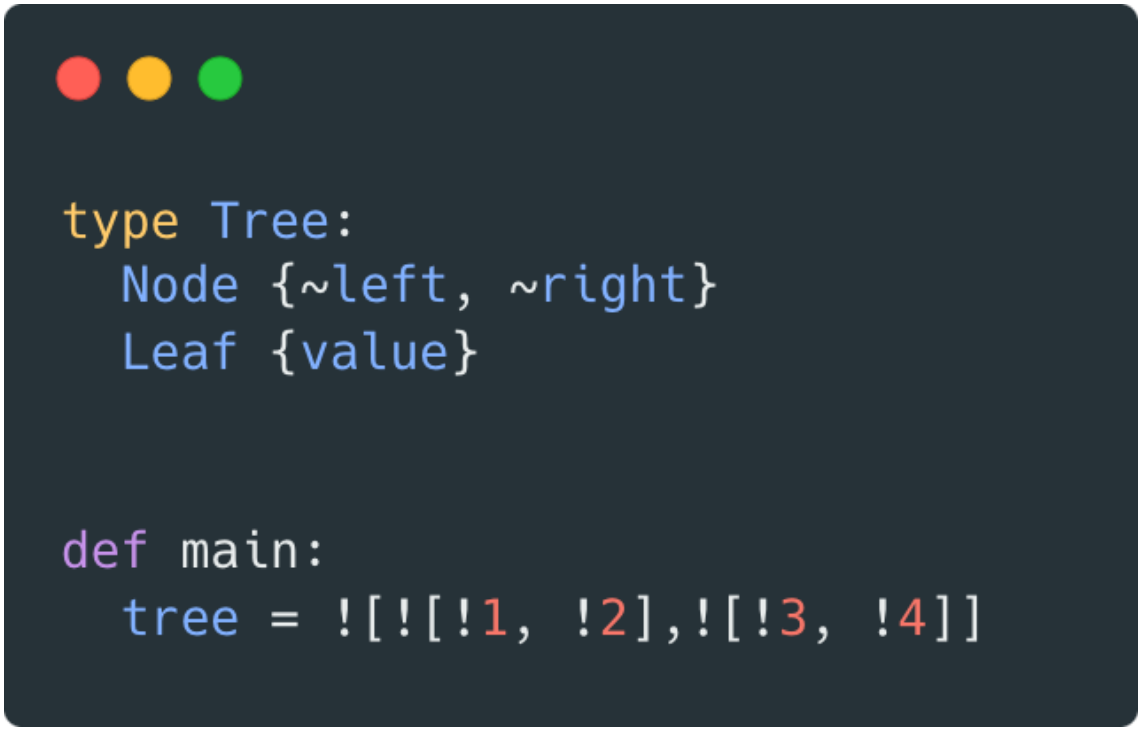
1) (1 + tail)
2) (1 + (2 + tail))
3) (1 + (2 + (3 + tail)))
4) (1 + (2 + (3 + (4 + 0))))



Bend: Fold

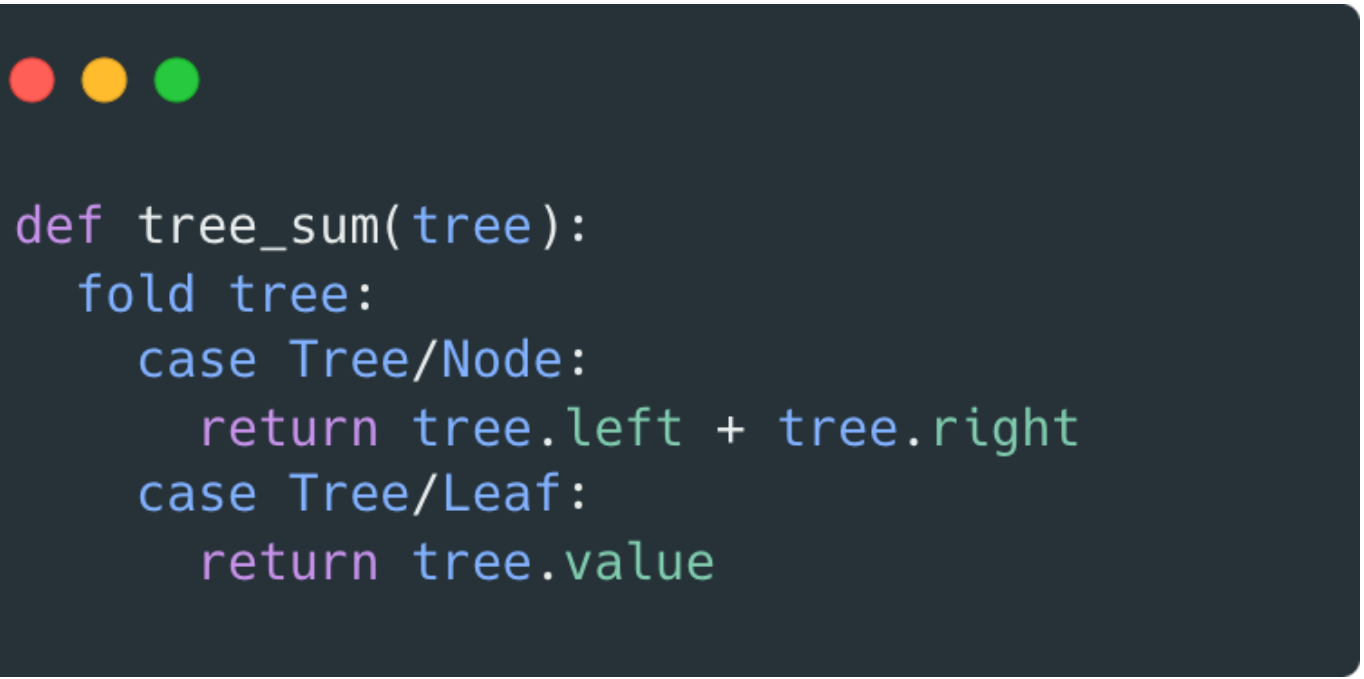
1) (1 + tail)
2) (1 + (2 + tail))
3) (1 + (2 + (3 + tail)))
4) (1 + (2 + (3 + (4 + 0))))

Bend: Tree



```
type Tree:
  Node {~left, ~right}
  Leaf {value}


def main:
  tree = ![![!1, !2],![!3, !4]]
```

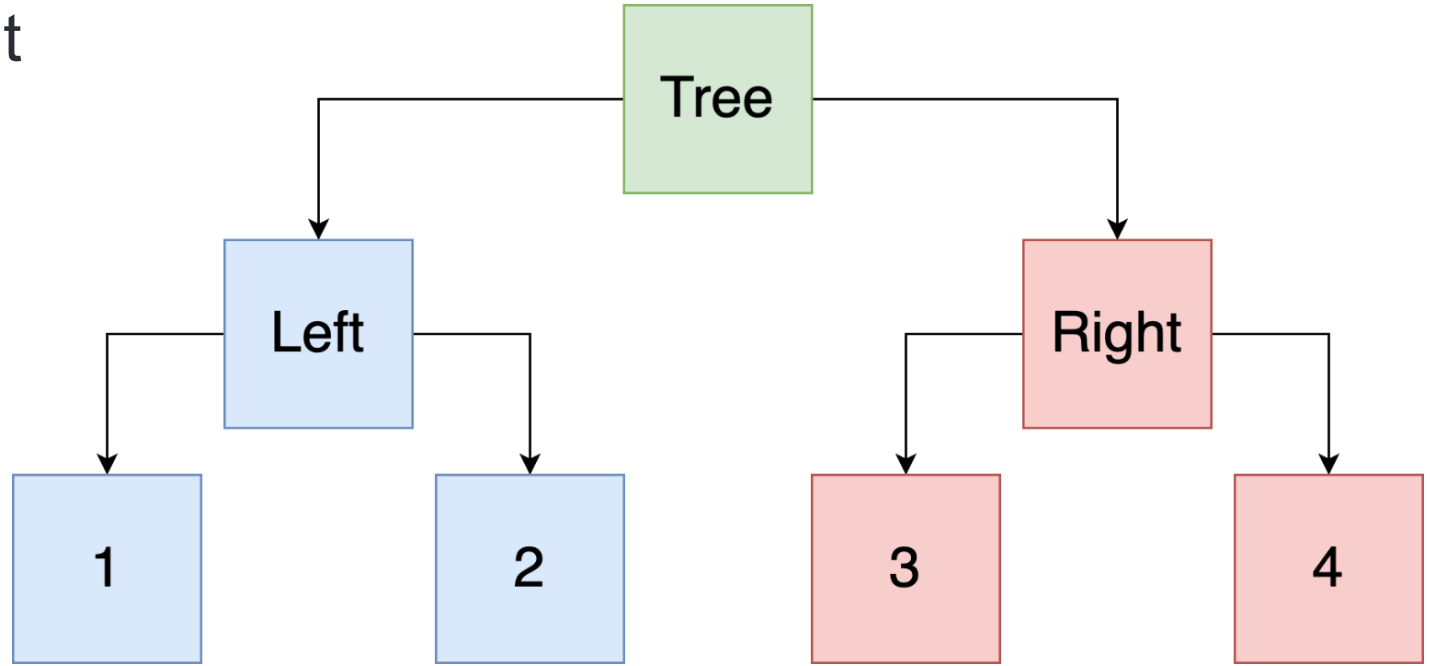
Bend: Fold



```
def tree_sum(tree):
  fold tree:
    case Tree/Node:
      return tree.left + tree.right
    case Tree/Leaf:
      return tree.value
```

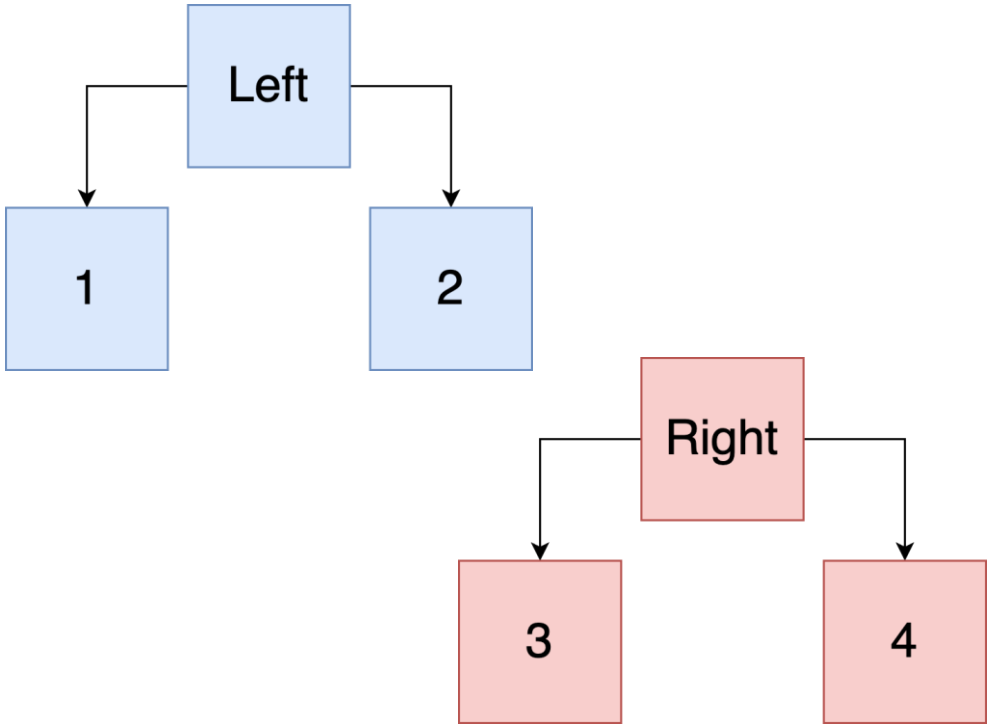
Bend: Fold

1) left + right



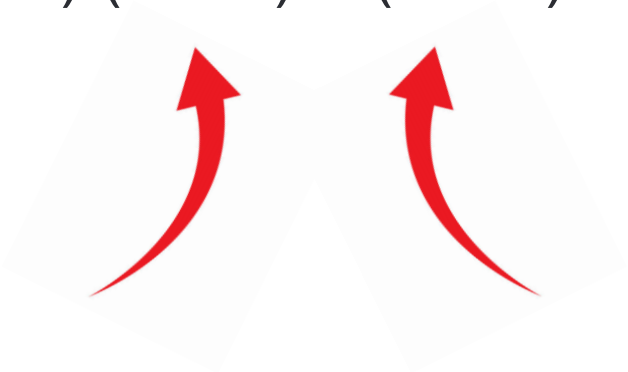
Bend: Fold

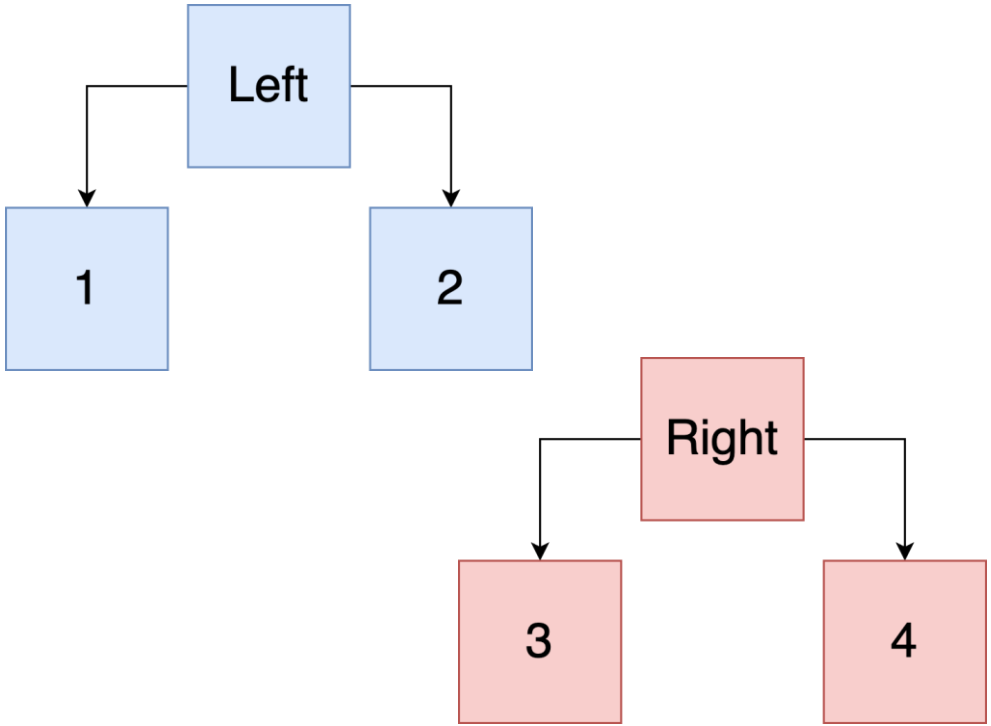
1) left + right
2) (1 + 2) + (3 + 4)



Bend: Fold

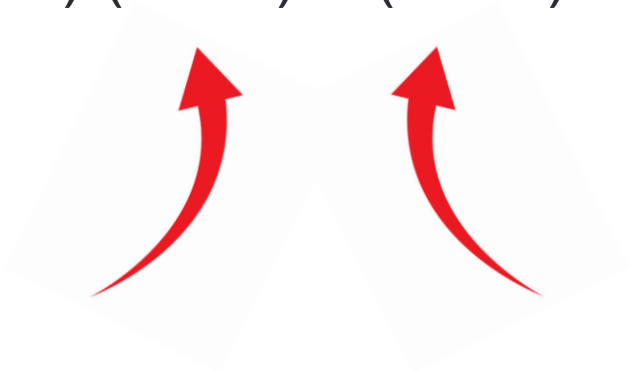
1) left + right
2) (1 + 2) + (3 + 4)





Bend: Fold

1) left + right
2) (1 + 2) + (3 + 4)



Bend: Установка

```
$ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

$ brew install gcc

$ cargo install hvm bend-lang
```

Bend: Запуск

```
# Rust interpreter (sequential)
$ bend run main.bend -s

# C interpreter (paralell)
$ bend run-c main.bend -s

# CUDA (paralell)
$ bend run-cu main.bend -s
```

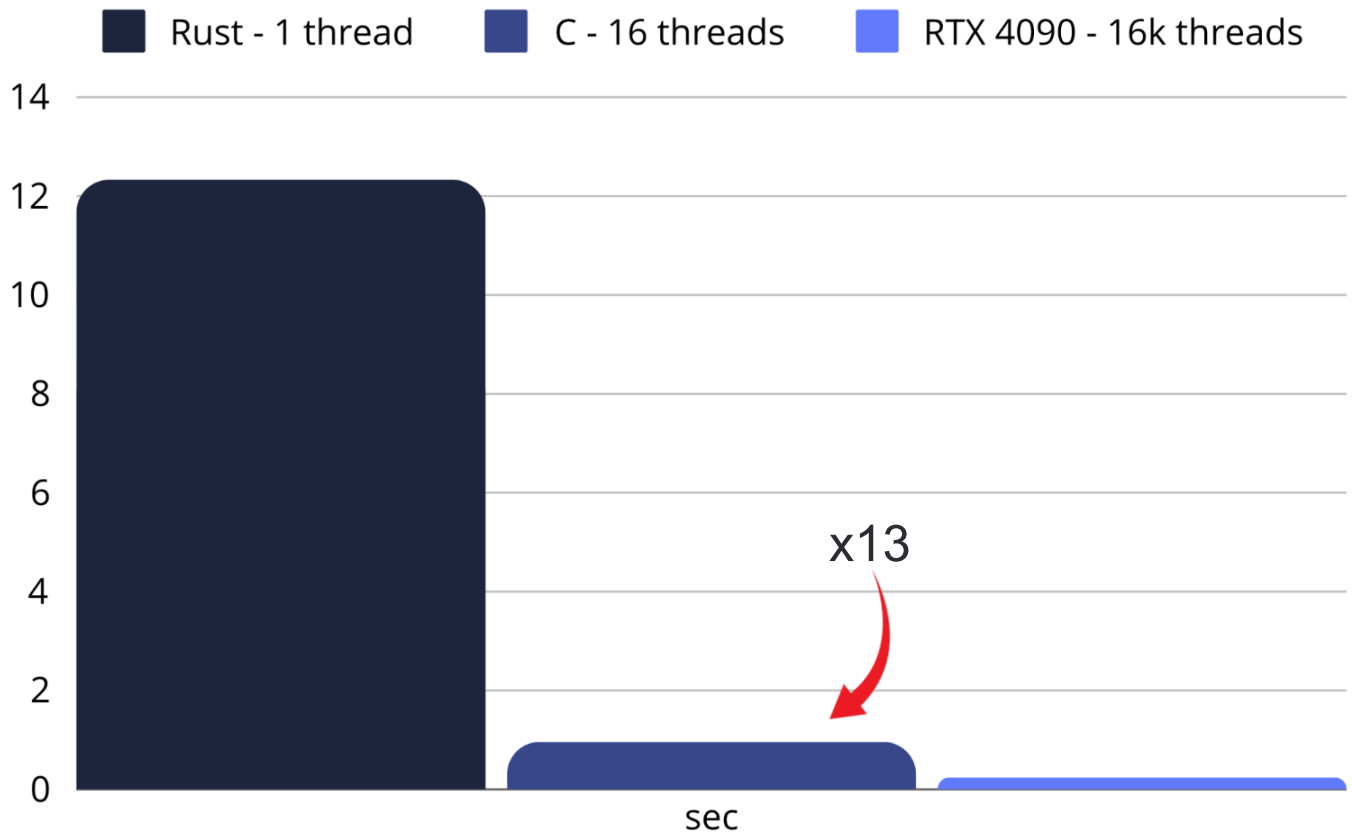
Bend: Бенчи

- Язык сырой
- Поддержаны только 24-х битные числа
- Библиотек нет
- Бенчить можно только что-то *простое

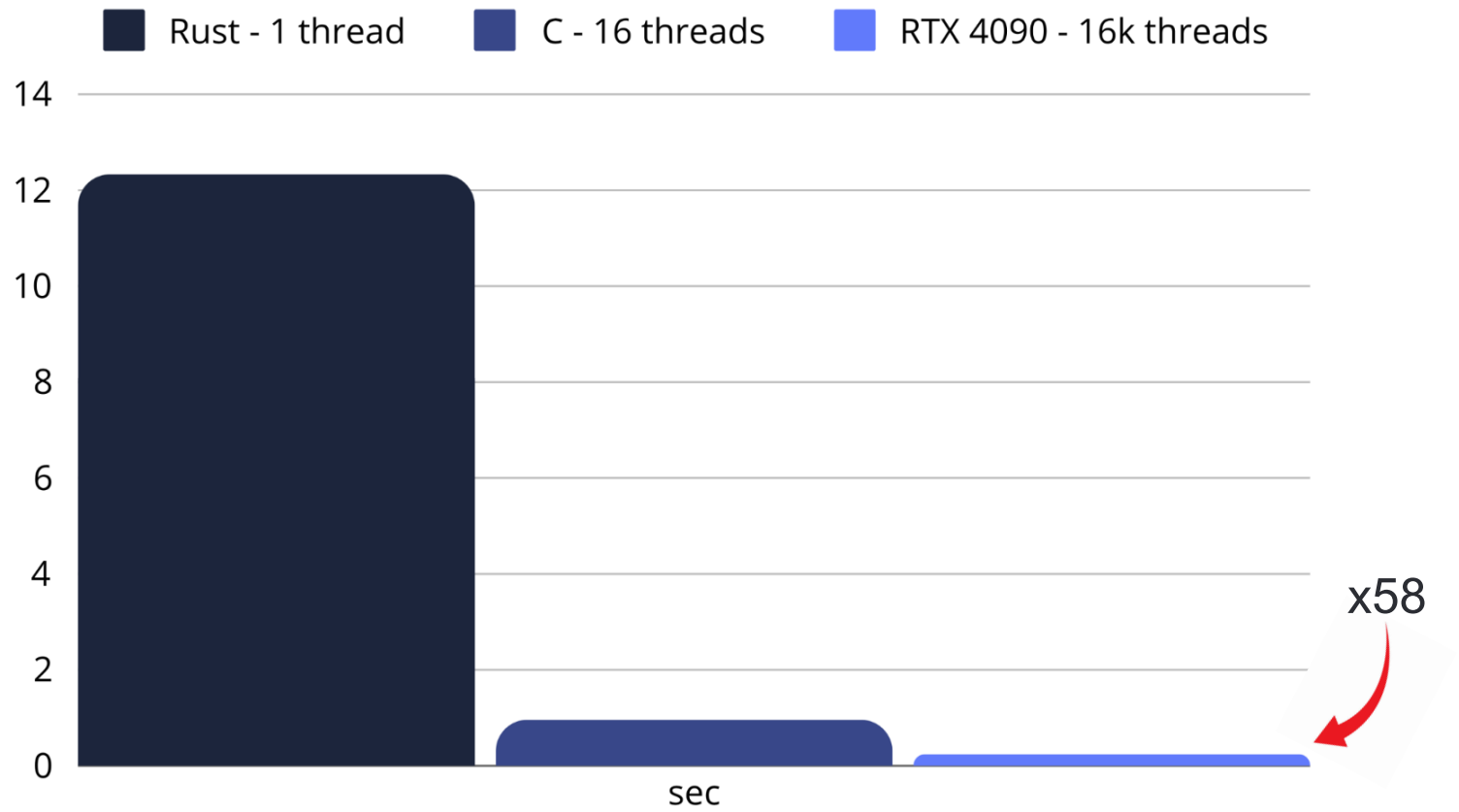
Bend: Bitonic sort

- Параллельный алгоритм
- Бьет массив на части
- Выполняется рекурсивно
- Подходит для GPU
- Apple M3 / RTX 4090

Bend: Бенчи


Rust - 1 thread
C - 16 threads
RTX 4090 - 16k threads
14
12
10
8
6
4
2
0
x13
sec

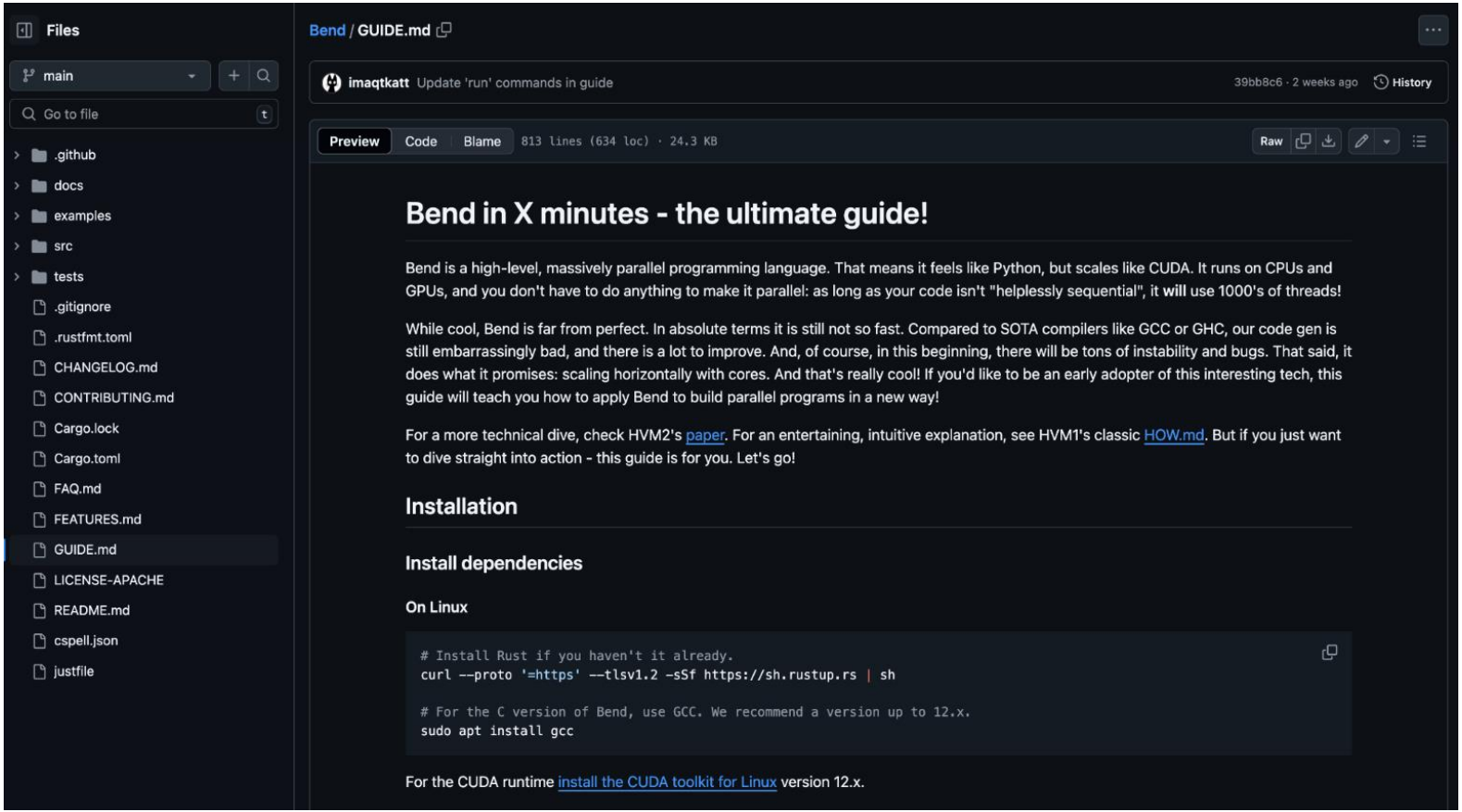
Bend: Бенчи


Rust - 1 thread
C - 16 threads
RTX 4090 - 16k threads
14
12
10
8
6
4
2
0
sec
x58

Bend: Использование Python

- Пока что нельзя
- Но есть FFI
- Несовершенный компилятор
- Собрать будет сложно

Bend: Эко-система



Mojo: Сообщество

@HigherOrderComp

higherorderco

Bend: Итоги

- Пока использовать нельзя
- Очень многообещающе
- Может улучшить питон
- Хочется еще побенчить

Итоги

	Mojo	Bend
Концепция	☑	☑ ☑ ☑
Опыт команды	☑	☑ / ☐
Скорость	☑	⁉
Удобство	☐	☐
Сообщество	☐	✕

Mojo

Bend

+ Max

Спасибо