



Welcome to the massively parallel future of computing

Victor Taelin & friends

Problem

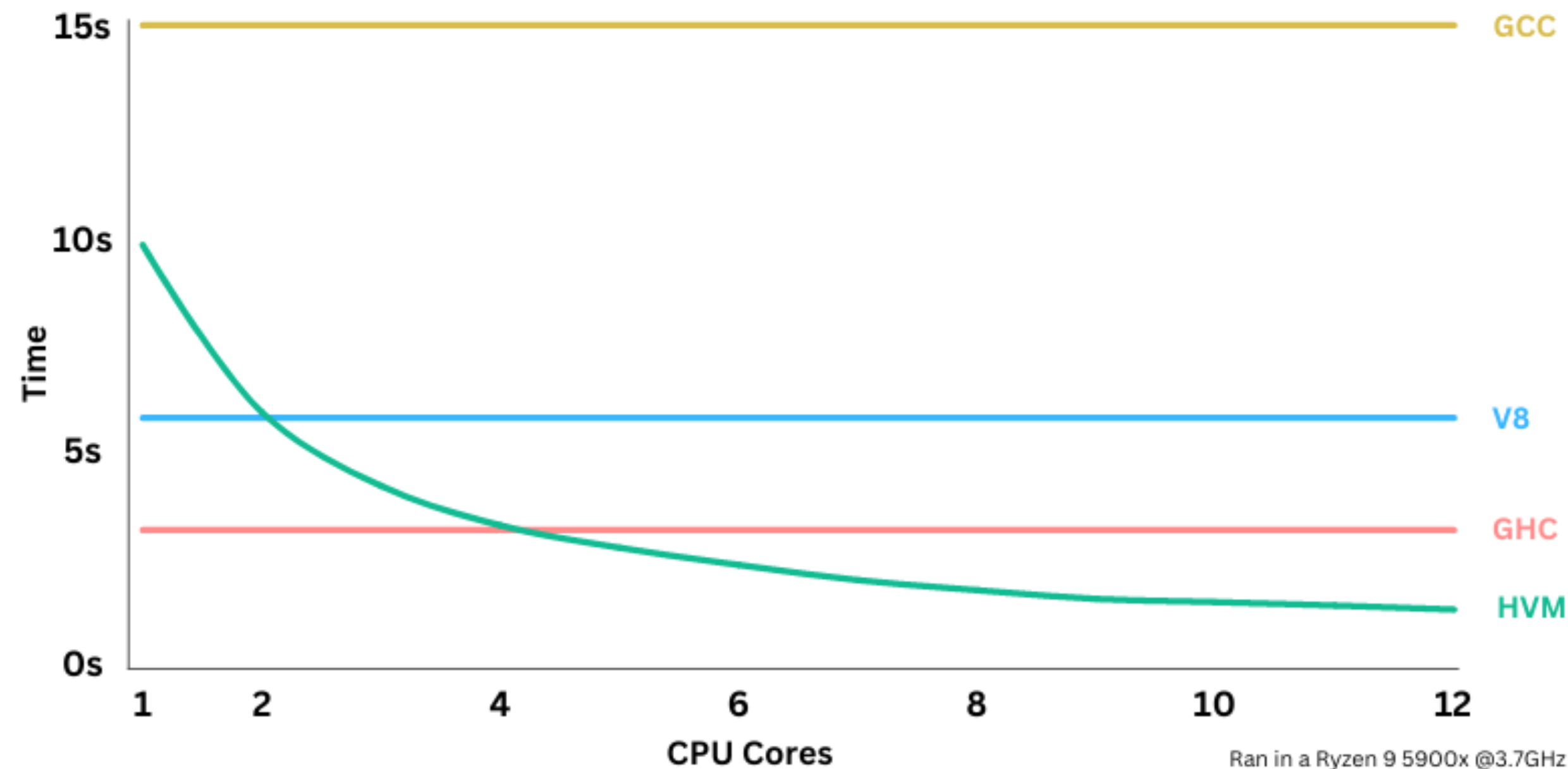
Software is not ready for parallel hardware

- CPUs with **increasingly more cores** build pressure to parallelize software
- Most modern programming languages are **single threaded** by default
- Parallel programming is **very expensive**, because:
 1. **concurrency errors** are complex (race conditions, deadlocks, etc.)
 2. **non-deterministic** behavior is very hard to debug
 3. **parallelism overhead** can actually reduce performance

Solution

HVM: a massively parallel runtime

- A runtime capable of automatic parallelism with near-ideal speedups
- Allows high-level programs to scale horizontally with available cores



To illustrate, we implemented a tree radix sort and compared running it on established runtimes vs HVM. Only on HVM, the more cores you have, the faster it runs! This is not a cherry picked example, but a general rule that is seen in most tests.

Benchmark: <https://github.com/VictorTaelin/HOC/tree/master/bench>

Product

ThreadBender: make your code massively parallel

- Transpiles popular languages (Python, JavaScript, etc.) to HVM on the fly
- Low entry barrier: just **npm install** and **bend** which functions to parallelize!

```
// slow code...
bigdata = function(size) {
  if (size <= 1) {
    return 1;
  } else {
    return {
      x: bigdata(size / 2),
      y: bigdata(size / 2),
    };
  }
}
console.log(sum(bigdata(2 ** 26)));
```

Time to run: **2.8 seconds**
On V8, the default runtime

Product

ThreadBender: make your code massively parallel

- Transpiles popular languages (Python, JavaScript, etc.) to HVM on the fly
- Low entry barrier: just **npm install** and **bend** which functions to parallelize!

```
//      \ just bend it!  
bigdata = bend function(size) {  
  if (size <= 1) {  
    return 1;  
  } else {  
    return {  
      x: bigdata(size / 2),  
      y: bigdata(size / 2),  
    };  
  }  
}  
console.log(sum(bigdata(2 ** 26)));
```

Time to run: **0.4 seconds**

With ThreadBender + HVM

That's a **700%** speedup with **8 cores**

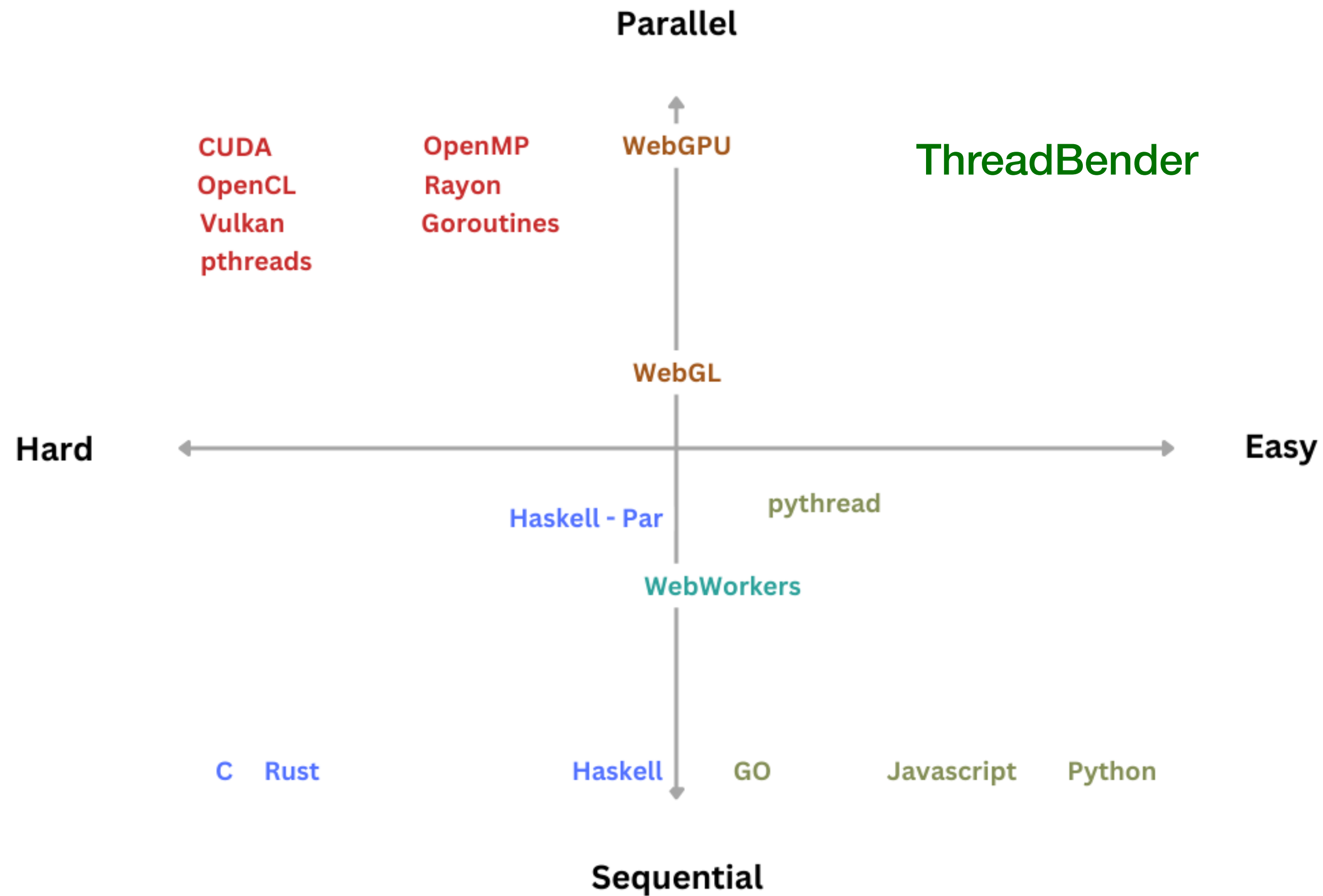
Benchmark: <https://github.com/VictorTaelin/HOC/tree/master/bench>

Business Model

Monetize on ThreadBender licenses, support and services

	Individual	Company	Enterprise
HVM (always free, open-source)	✓	✓	✓
ThreadBender (freemium, paid licenses)	✓	✓	✓
Consulting Services		✓	✓
Email Support		✓	✓
24/7 Support			✓
	starting at \$0	starting at \$??	starting at \$??

Alternatives



There are several tools and languages used for parallelism, but they are either limited in scope, or require expensive development, due to very strict limitations and hard-to-debug issues.

As for automatic parallelism, this isn't a new idea. There is a wide body of research trying to achieve it, but, until now, success was limited.

HVM auto-parallelizes high-level features like allocation, matching, recursion, lambdas. That's why **ThreadBender** is so easy: no need to deal with atomics, locks, mutexes. ***Just bend it.***

Technology

How we solve automatic parallelism

We use a new model of computation, the **Interaction Calculus**, which *completes* the **Lambda Calculus** with **Interaction Net** semantics. Looks complex, but the key insights are simple:

- 1. Make **everything pure** (like Haskell) - *no side effects*
- 2. Make **everything linear** (like Rust) - *no shared references*
- 3. Add a **first-class lazy cloner** ("*fan nodes*") - makes it *Turing complete*
- 4. Keep a **thread pool** with a **work stealing queue** of **interaction rules**

This **new theoretical foundation** built on the shoulder of giants (Yves Lafont, Girard, Lamping, Aaron Stump...) let us create **HVM**, the first general-purpose, parallel runtime with near-linear speedup!


In-depth explanation: <https://github.com/Kindelia/HVM/blob/master/HOW.md>

Interaction Combinators: <https://www.semanticscholar.org/paper/Interaction-Combinators-Lafont/6cfe09aa6e5da6ce98077b7a048cb1badd78cc76>




Adoption

Our prototype already conquered developer's hearts!



▲ High-order Virtual Machine (HVM): Massively parallel, optimal functional runtime (github.com/kindelia)
493 points by Kinrany 10 months ago | [hide](#) | [past](#) | [favorite](#) | 151 comments



 **HVM** Public

A massively parallel, optimal functional runtime in Rust



 Rust  5.5k  174

YOU SENT AN INVITE TO JOIN A SERVER

 **Kindelia Community** 

 279 Online  1,311 Members






Joined

  ...
@delete_shitcoin

The smartest CS person alive is [@VictorTaelin](#), the symmetric interaction calculus is the ultimate foundational model of computation, it completes and supersedes the half-done false idol lambda calculus. You heard it here first. All other living PL theorists completely outclassed.

10:56 PM · Aug 17, 2022

2 Retweets 1 Quote Tweet 34 Likes

     Tip

Seed Round

We are raising 7 million to build our business

- In our seed round, **we'll offer 20% of HOC for a \$7m ask**
- These funds will be used to:
 1. Hire developers to make HVM production ready
 2. Develop and ship ThreadBender, our main product
 3. Cover the day to day operations and expenses
- We've accomplished a lot with very little:
 - We built a competitive compiler on a \$100k budget that outperforms GCC, GHC and V8 by 10x on real tasks
 - We also built a proof assistant (Kind) and a p2p computer (Kindelia) to explore HVM's applications
 - We hired unexperienced developers from our developer community and trained them to use our tech
 - We have extensive experience on the field and our tech has been able to draw attention on its own merit

Research

In the future, the entire world will run on higher-order machines

Kind-Lang

Towards the formalization of mathematics

- **Kind-Lang** is already the fastest proof assistant in the world, by far!
- We're now working on making it the best overall, period.

Pure functions are defined via equations, as in [Haskell](#):

```
// Applies a function to every element of a list
map <a> <b> (list: List a) (f: a -> b) : List b
map a b Nil          f = Nil
map a b (Cons head tail) f = Cons (f head) (map tail f)
```

Side-effective programs are written via monads, resembling [Rust](#) and [TypeScript](#):

```
// Prints the double of every number up to a limit
Main : IO (Result () String) {
  ask limit = IO.prompt "Enter limit:"
  for x in (List.range limit) {
    IO.print "{} * 2 = {}".x (Nat.double x)
  }
  return Ok ()
}
```

Theorems can be proved inductively, as in [Agda](#) and [Idris](#):

```
// Black Friday Theorem. Proof that, for every Nat n: n * 2 / 2 == n.
black_friday_theorem (n: Nat) : Equal Nat (Nat.half (Nat.double n)) n
black_friday_theorem Nat.zero      = Equal.refl
black_friday_theorem (Nat.succ n) = Equal.apply (x => Nat.succ x) (black_friday_theorem n)
```

For more examples, check the [Wikind](#).



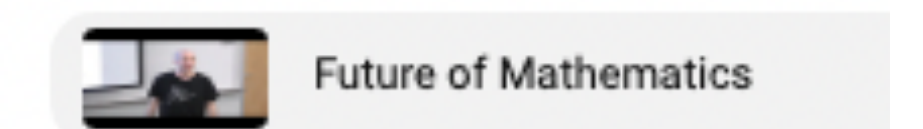
The Future of Mathematics?

89K views • 3 years ago



As a professor of pure mathematics, my job

CC

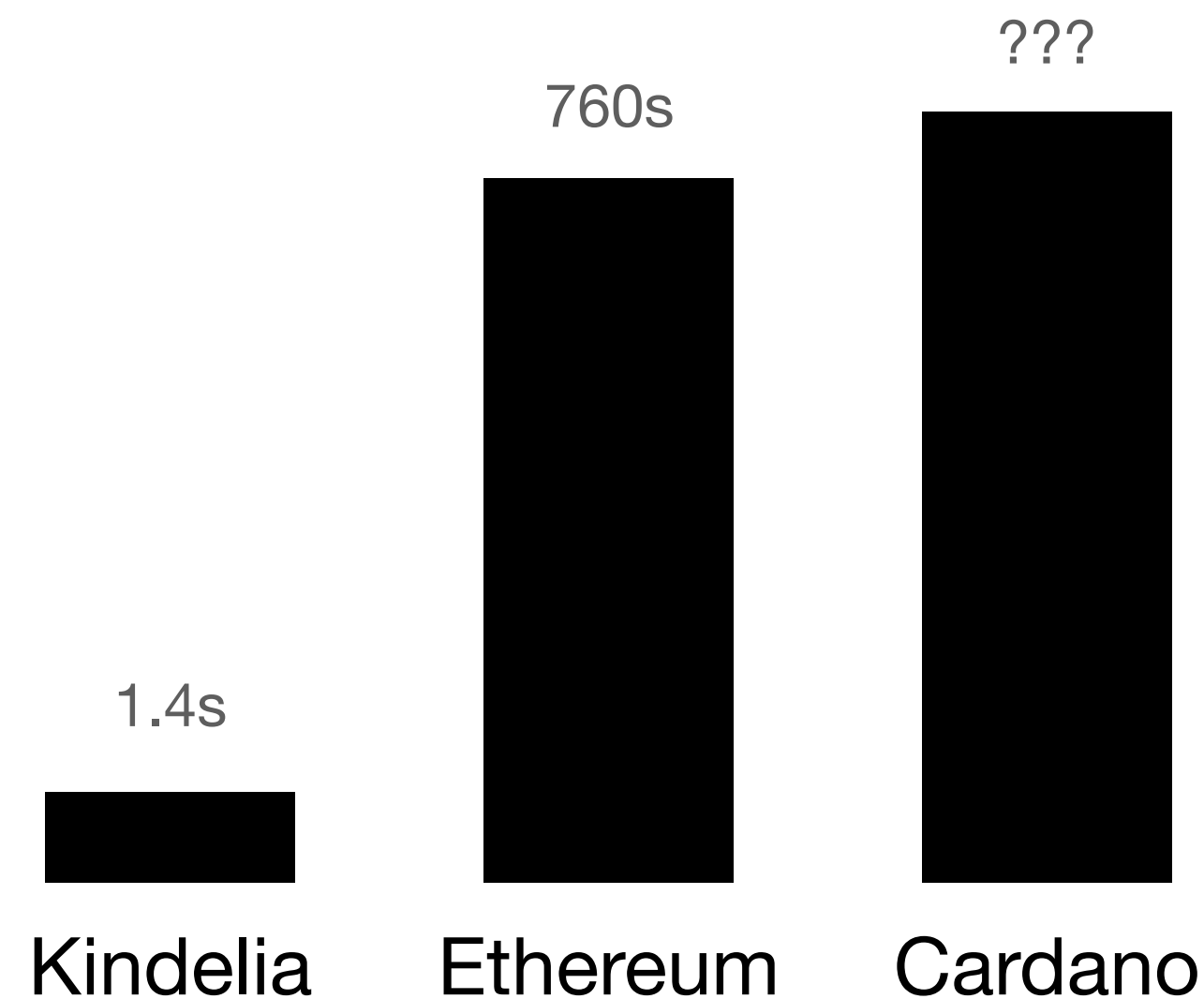


Benchmarks: <https://github.com/kindelia/functional-benchmarks>

Kindelia

Towards pure peer-to-peer computers

- **Kindelia** is a no-currency peer-to-peer computer where devs can deploy unstopable HVM apps.
- Thanks to HVM and Kind, it is faster than Ethereum, and more secure than Cardano.



Time to run `fib(42)` on-chain, in seconds

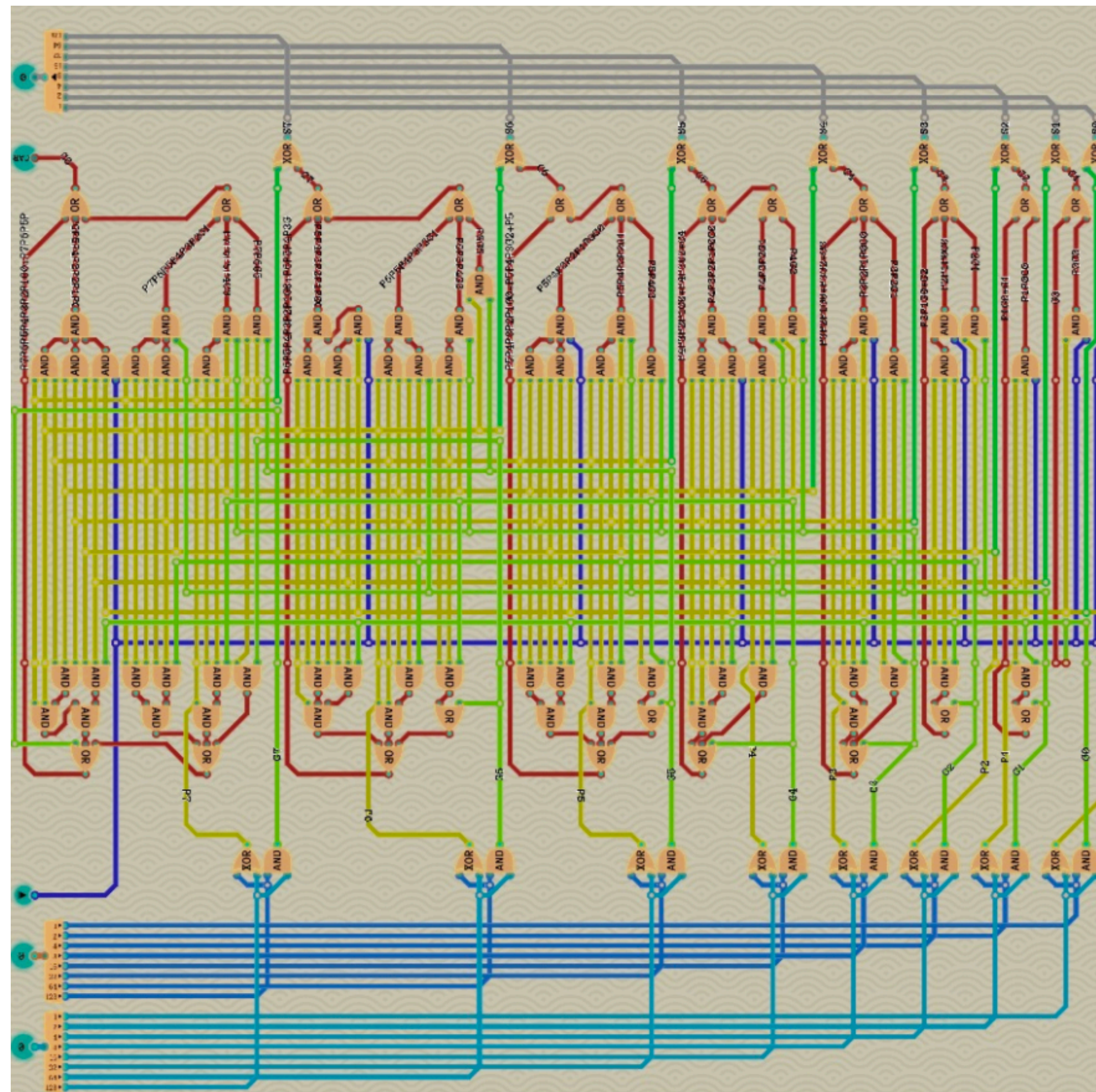
```
1 // Supply Conservation Theorem
2
3 MyToken.send_conserve_supply
4   (b : Map U60) // for any a balance map `b`
5   (s : Address) // for any source address `s`
6   (d : Address) // for any destination address `d`
7   (a : U60)     // for any amount `a`
8
9 : // Sending `a` from `s` to `d` conserves `b` supply
10 let old_supply = Map.sum b
11 let new_supply = Map.sum (MyToken.send b d s a)
12 Equal U60 old_sum new_sum
13
14 // Proof:
15 let aux_0 = MyToken.send_conserve_supply.aux_0 (MyT
16 let aux_1 = Equal.rewrite aux_0 (Equal.mirror (MyTok
17 let aux_2 = match aux_1 {
```

Native formal verification

HPU

Towards higher-order processing units

- **HPU** is a hypothetical hardware capable of running higher-order computations on-chip.



A HVM interaction could use less space than a 8-bit adder!

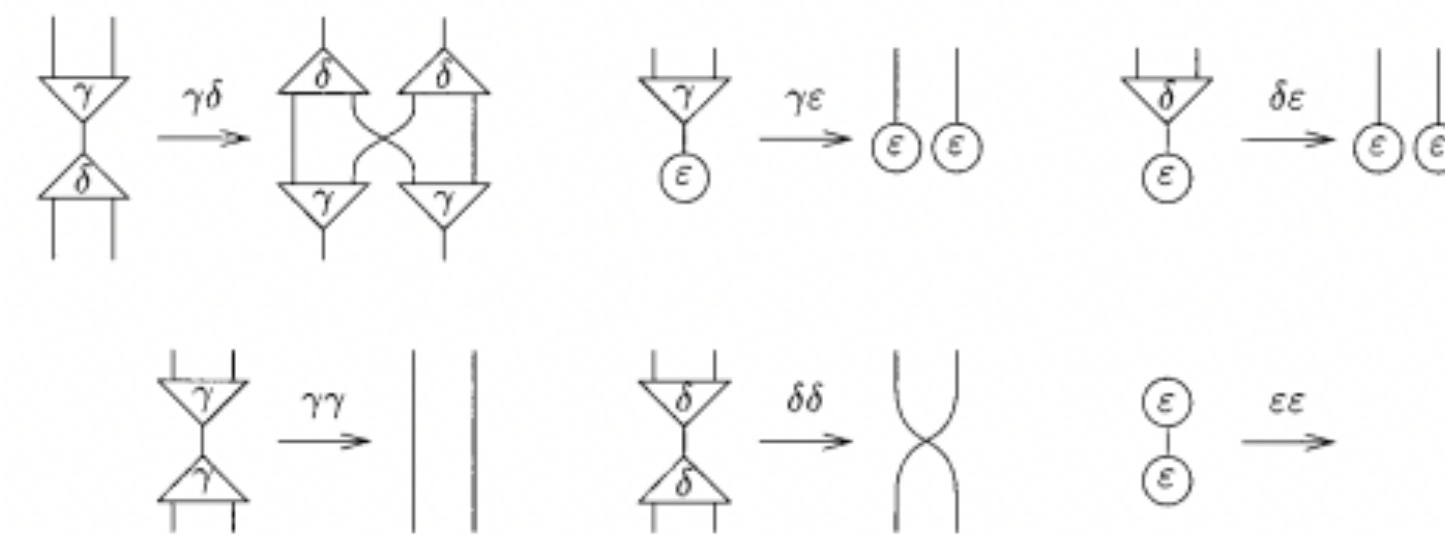
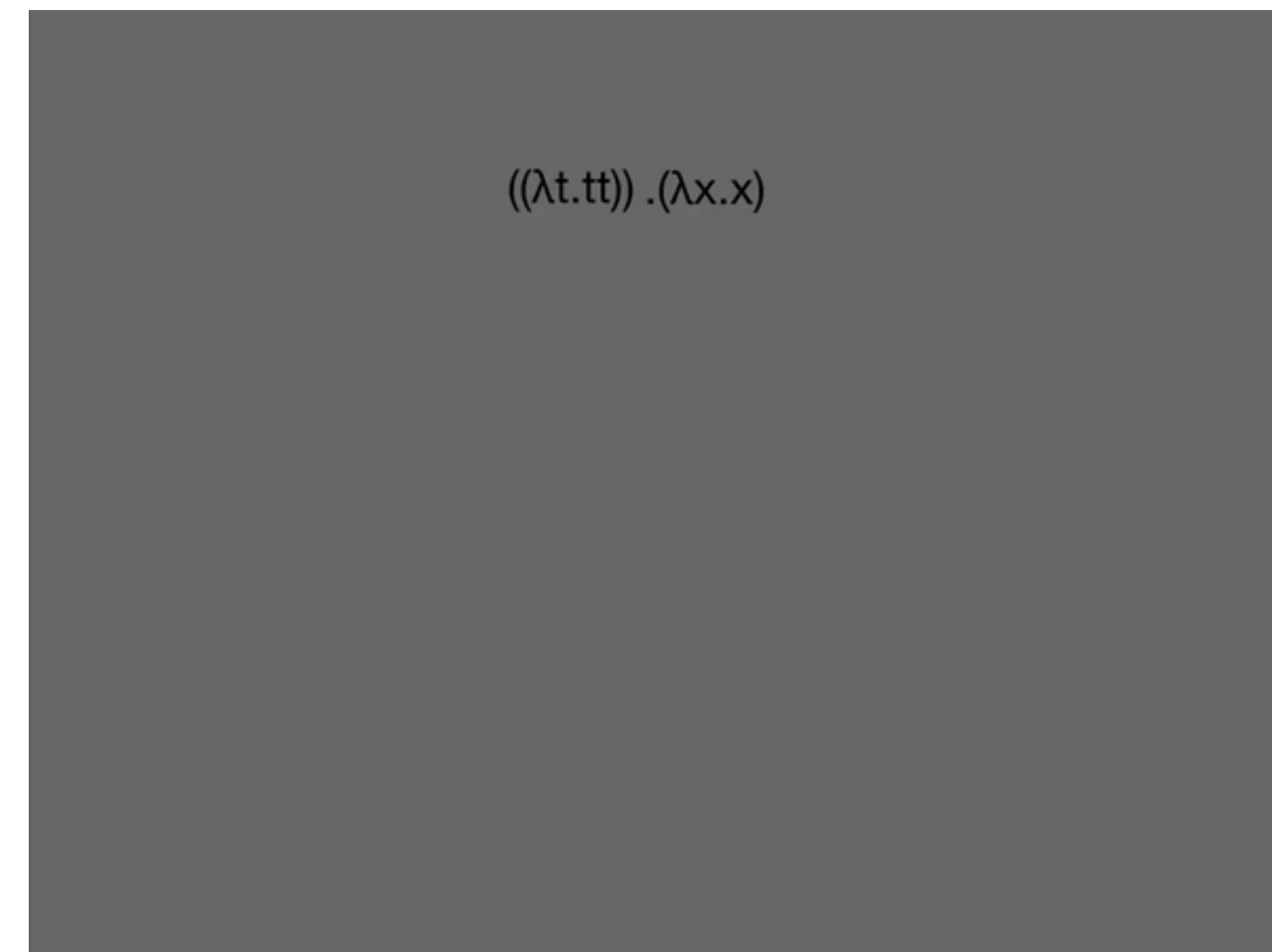


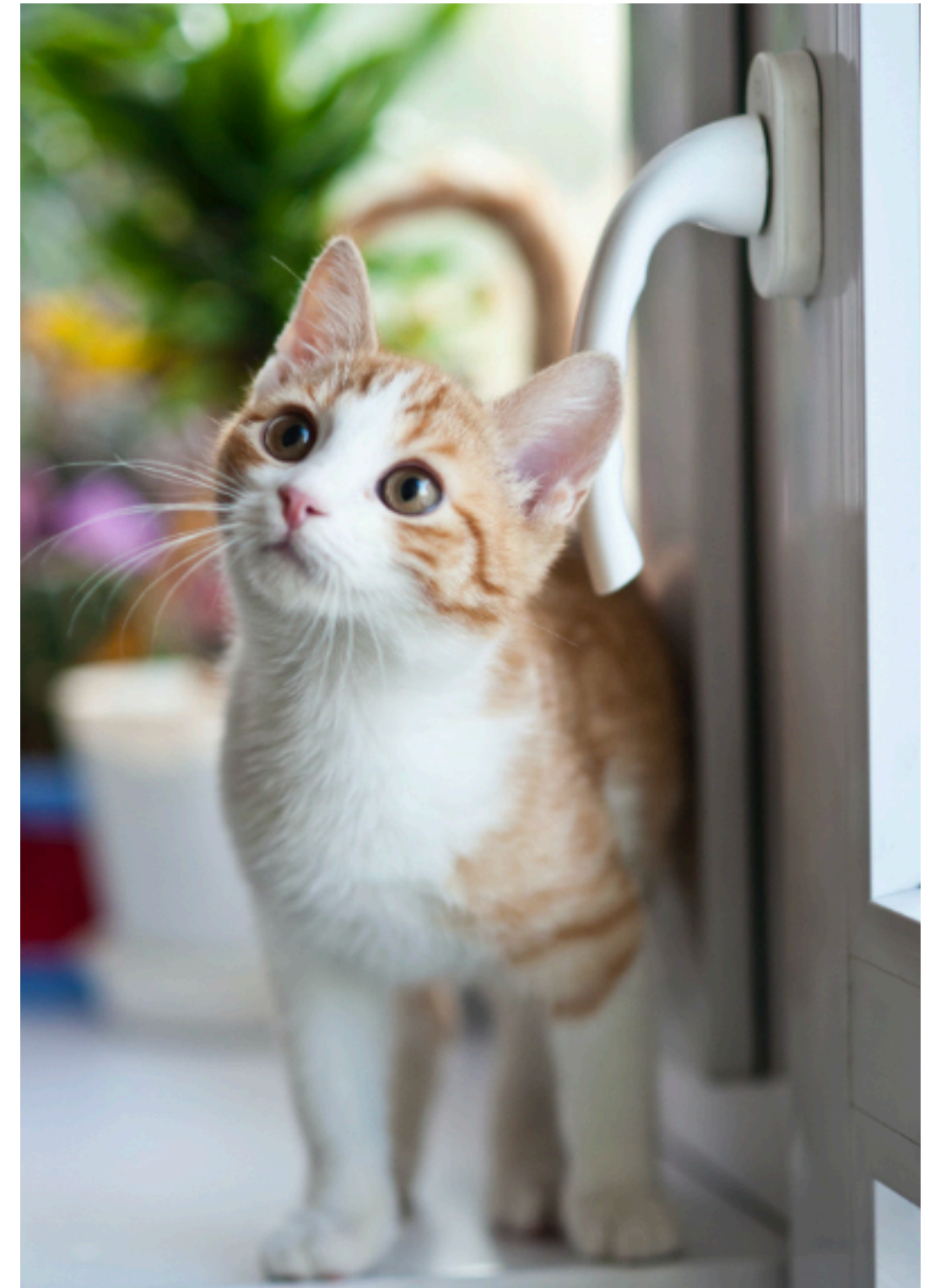
FIG. 2. Interaction rules for the combinators.



Higher order computation in action (click to play)

Cats

They're adorable



It is not everyday that...

- Someone builds a new compiler with a **100k budget**
- ... that outperforms GCC, GHC and V8 by **10x**
- Once we raise \$7m, *there is no going back*
- - we're not making a PDF converter
- - we're not competing with local restaurants
- + we're set to disrupt the entire tech industry
- + we're aiming for Google, nVidia, Intel, Apple
- **This is your best chance to own a big part of it!**