

Evolving the OCaml programming language

KC Sivaramakrishnan
3rd September 2025
kcsrk.info

IIT
MADRAS
IIT MADRAS

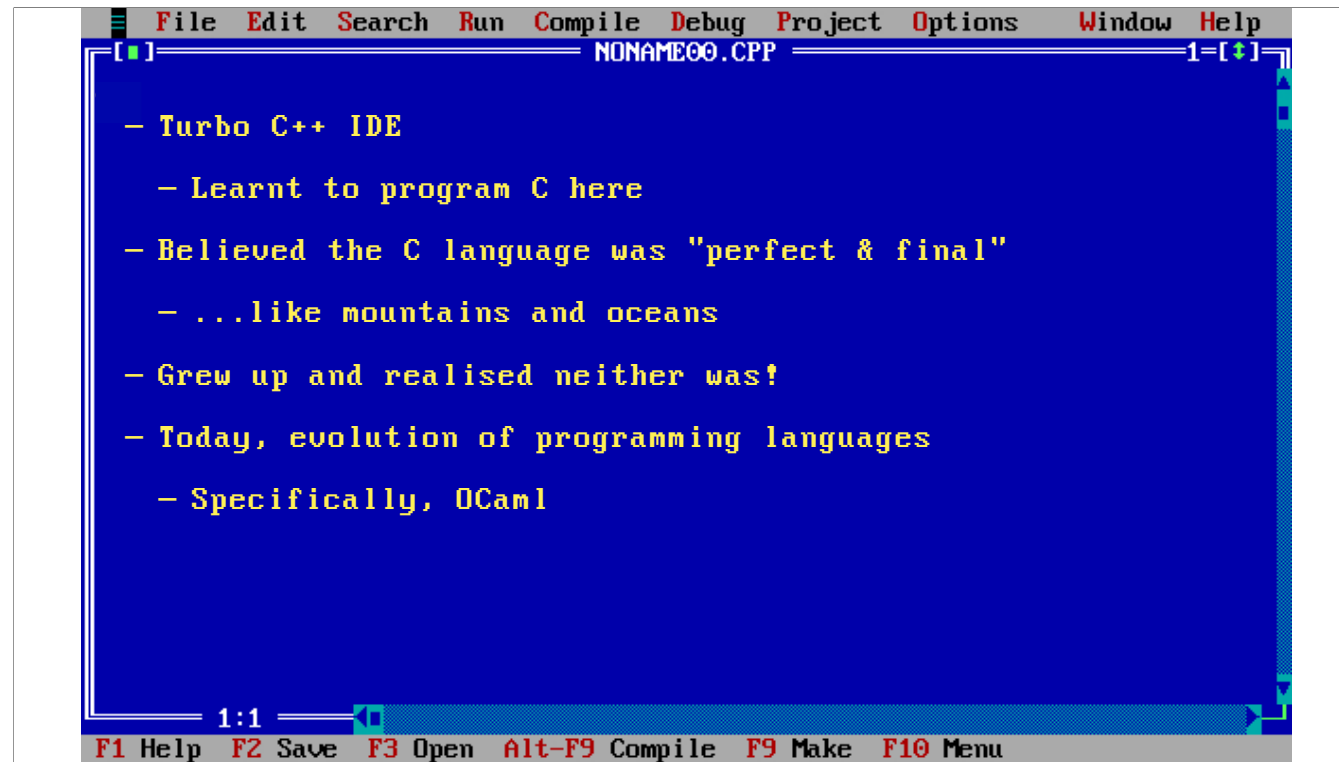


Tarides

Who am I — KC Sivaramakrishnan

- CS Prof at IIT Madras
 - Programming languages, formal verification and systems
- A core maintainer of the *OCaml* programming language
- CTO at Tarides, an open-source software shop
 - Building functional systems using *OCaml*
 - Maintainers of the OCaml compiler and platform tools

I am a Prof at IIT Madras at the Computer Science and Engineering Department. I am also a core maintainer of the OCaml programming language. I'm also the CTO of Tarides, which is an open-source company building functional systems using OCaml. We maintain the OCaml compiler and many of the platform tools in the OCaml ecosystem.



I learnt to program in this wonderful IDE.

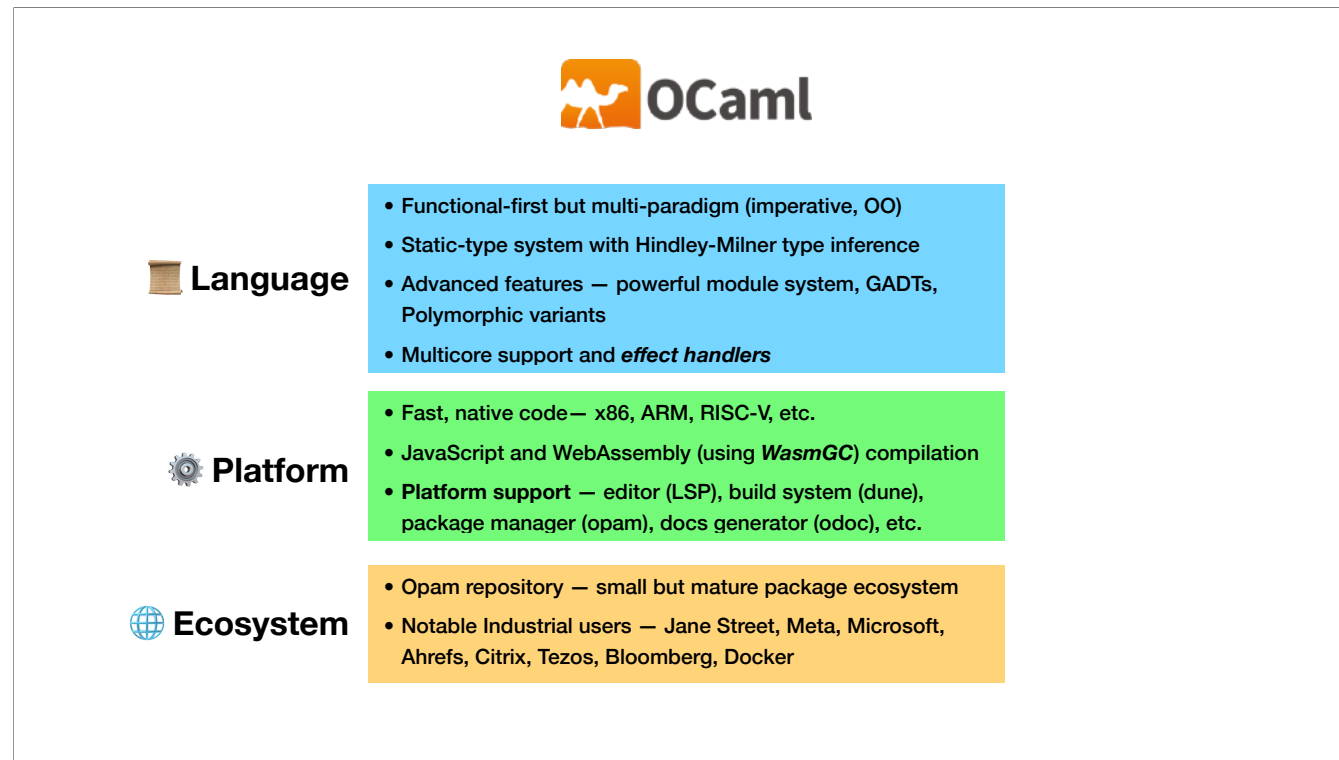
This is the turbo C++ IDE, where I learnt to program the C language.

While I was learning C many years ago, I assumed C was perfect and final, just like how mountains and oceans are. Unchanging, remaining the same from long ago in the past and far into the future.

Then I grew up and realised that mountains and oceans do change over millions of years. It is hard to see the change.

Likewise Programming Languages also change, and this is also hard to see.

In this talk, I'd like to talk about the evolution of programming languages. Specifically, the how the OCaml programming language evolves.



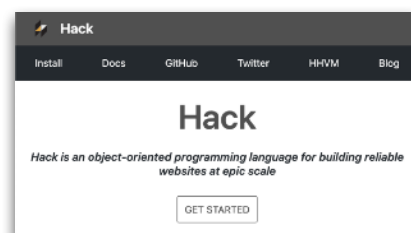
OCaml is a functional first programming language, but it also has first-class support for imperative and object-oriented programming. It is a statically typed language, with Hindley-Milner type inference. It has a number of advanced features such as a power module system, which makes it suitable for industrial users. It has recently gained shared-memory parallelism support and effect handlers for concurrency. OCaml is the first industrial-strength language to support effect handlers.

OCaml can generate fast-native code for a variety of backends including x86, ARM and RISC-V. OCaml code can also be compiled efficiently to JavaScript and WebAssembly. Even here, OCaml is a pioneer. OCaml is the first industrial-strength language to use Wasm Garbage Collection extension. OCaml has good platform tooling support with an LSP, a fast build system, a strong package manager, a solid documentation generator, etc.

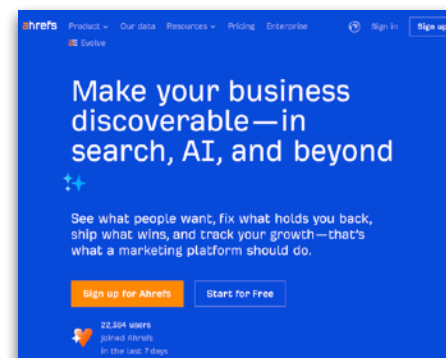
In terms of the ecosystem, OCaml has a small but mature package ecosystem. A number of notable industrial users use OCaml, including Jane Street, Meta, Microsoft and Bloomberg.

High dynamic range

From scripts to scalable systems, research prototypes to production infrastructure



Compilers



Web Frontend

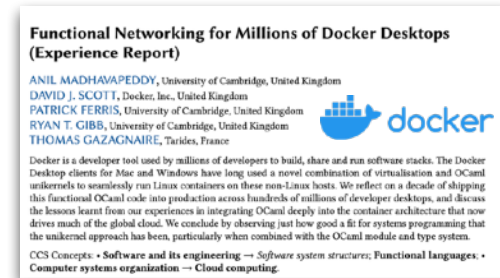
One of the reasons OCaml is favoured by its users is its high-dynamic range. OCaml is generally good at many things. From scripts to scalable systems, to research prototypes to production infrastructure. Here are a few examples.

OCaml is great for writing compilers. The first version of the Rust compiler was written using OCaml. WebAssembly reference interpreter is written in OCaml. Hack, the programming language with which Facebook is written is written in OCaml.

OCaml is also used for developing serious front-end applications. All the frontend of Ahrefs, one of the largest SEO tools company, is written in OCaml and compiled to JavaScript.

High dynamic range

From scripts to scalable systems, research prototypes to production infrastructure



OCaml in Space 🚀🌌

Parsimoni



Systems Programming

OCaml is also great for “systems programming”

Docker for Mac and Windows use OCaml to seamlessly run Linux containers on non-Linux hosts.

Early this year, Parsimoni, a spinoff from Tarides, launched OCaml into space. The payload contains a virtualisation manager that is written in OCaml.

High dynamic range

From scripts to scalable systems, research prototypes to production infrastructure



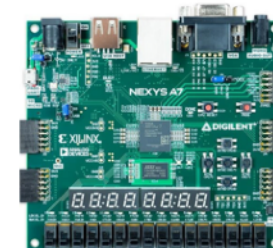
Bloomberg

Finance

60+M lines of
OCaml code!



*Hardware
design*



OCaml is widely used in Finance. Jane Street, a quant firm, is the largest user of OCaml in the industry with more than 60 million lines of code. OCaml is used to write all the software from trading systems to UIs for traders.

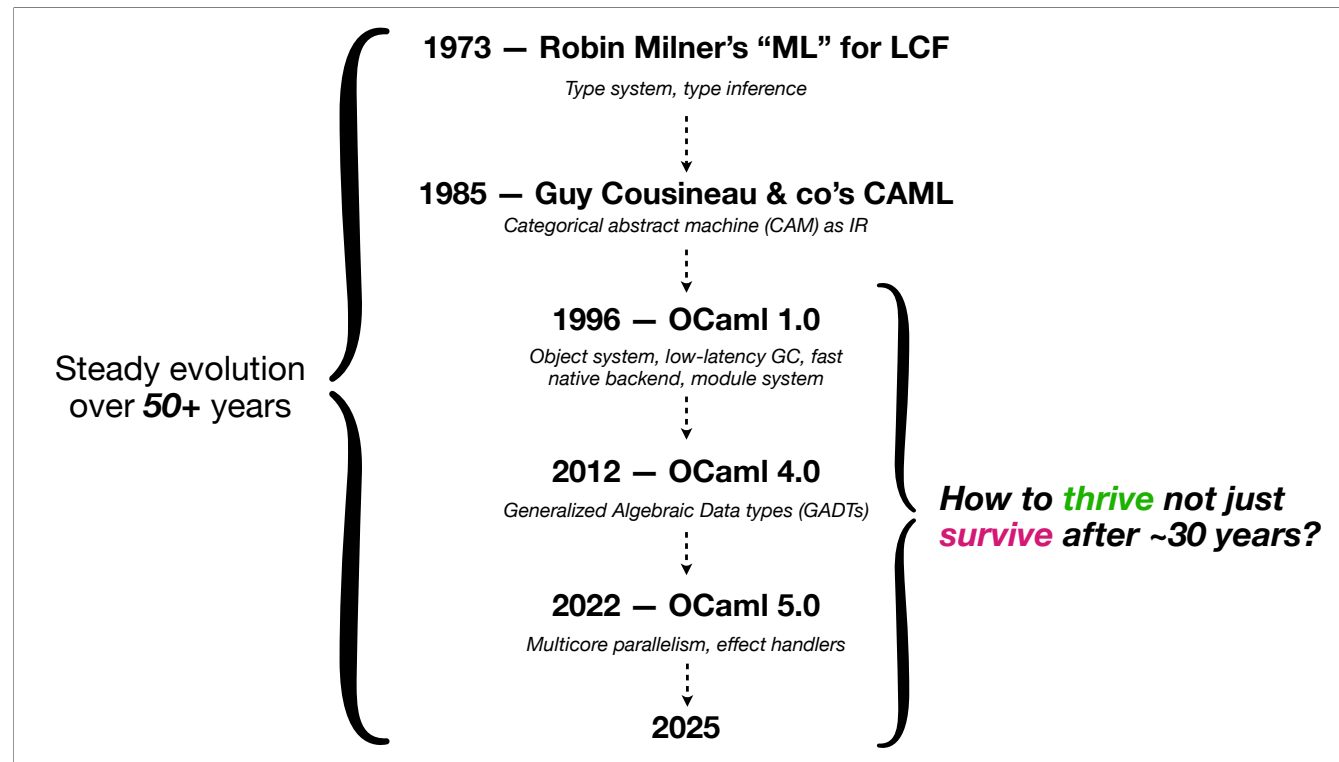
You can also program FPGAs using the OCaml library called HardCaml. HardCaml has been used to develop award-winning solutions to the ZPrize competition for Zero-Knowledge Cryptography.



29 years old!



Given these developments, you might be surprised to hear that OCaml is 29 years old. It is likely that OCaml is older than many of you in the audience. Just to drive the point home, when OCaml was released, this movie was still running in theatres and this was the most selling cell phone that year.



But even when OCaml was released, it already had a bunch of features it is loved for today including the object system, low-latency GC and the module system. In fact, OCaml draws from research done decades earlier.

Its type system and type inference draws from Robin Milner's LCF theorem prover which introduced the Meta-Language (ML), which is the source of all ML family languages.

OCaml's execution builds on top of Guy Cousineau (coo-zee-noo)'s Categorical Abstract Machine IR which gives rise to CAM in OCaml.

OCaml has continued to evolve with getting Generalised Algebraic Data Types in 2012 in OCaml 4.0 and Multicore Parallelism and Effect Handlers in 2022 with OCaml 5.0.

Observe that the language has developed steadily over the past 50+ years! What's the secret sauce to, not just survive, but thrive after almost 30 years.

Simplicity and *stability*

Xavier Leroy, 2023 SIGPLAN programming languages software award! 🏆

What made that possible? Not just fancy types and nice modules – even though systems programmers value type safety and modularity highly – but also basic properties of OCaml:

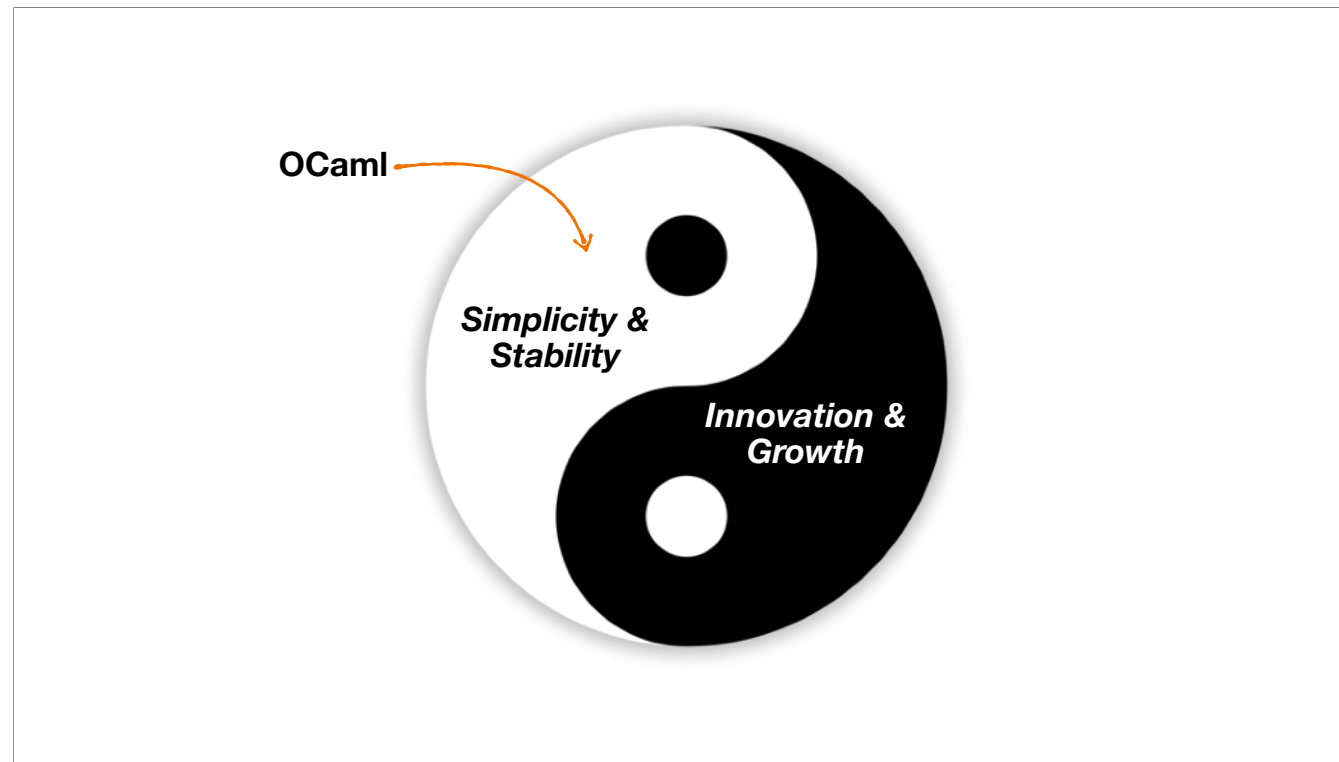
- a language with a simple cost model, where it's easy to track how much time and how much space is used;
- a compiler that produces efficient code that looks like the source code, with only predictable optimizations;
- a low-latency garbage collector, usable for soft real-time applications.

- If you take OCaml from 20 years ago, the code will likely *continue to work!*
- No recent releases for some popular packages
 - They are *good enough*, and continue to be so.
 - Nothing to be done to keep it working!

The secret sauce in the language is “simplicity” and “stability”

In 2023, OCaml won the SIGPLAN programming languages software award. In the award acceptance speech, Xavier Leroy, the creator of the OCaml programming language mentioned a few reasons why OCaml has been successful with industrial users. First, it is a language with a simple cost model, where it is easy to track how much time and space is used by the program. Secondly, its compiler produces efficient code that looks like the source code, with only predictable optimisations. You can change the source code and get appropriate change in the compiled code.

OCaml is also an incredibly stable language. You can take code from 20 years ago and compile it with the latest compiler and it is very likely that the code will continue to work! People coming from other language ecosystems come look at the OPAM repository and see that many packages haven't been updated for a couple of years. They assume that the ecosystem is dead. In fact it is the opposite. The code is good enough and will continue to be so. There's nothing to be done to keep it working!!!



There is indeed a tension between simplicity and stability on one side and innovation and growth on the other side. OCaml squarely is on the simplicity and stability side of things. OCaml may strike people as a conservative language.

OCaml Maintainers

Abigael	Richard Eisenberg	Nicolás Ojeda Bär
Alain Frisch	Jacques-Henri Jourdan	Florian Angeletti
Armaël Guéneau	KC Sivaramakrishnan	Olivier Nicole
Anil Madhavapeddy	Frédéric Bour	Sadiq Jaffer
Pierre Chambart	Leo White	Sébastien Hinderer
Damien Doligez	Vincent Laviron	Stephen Dolan
David Allsopp	Luc Maranget	Thomas Refis
Jacques Garrigue	Mark Shinwell	Xavier Leroy
Gabriel Scherer	Nick Barnes	Jeremy Yallop

- 27 maintainers from France, UK, Japan, India and USA, across industry and academia.
- Custodians of the compiler
 - ***Not the ones deciding how the language should evolve!***

Before we look at how OCaml evolves, let's look at how the development community is organised.

There are 27 maintainers in OCaml today from all over the world, across industry and academia. None of these people are paid to work on OCaml. Some of them, like me, are in jobs that allow them to spend some of their time on OCaml.

An important thing to note is that maintainers are only the custodians of the compiler. They are not the ones deciding how the language should evolve.

Who decides how OCaml evolves?

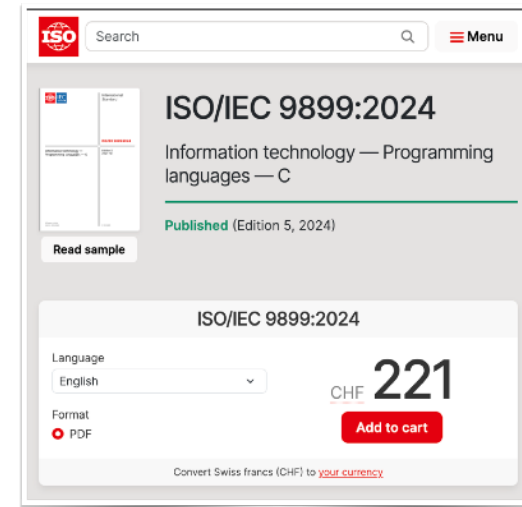


Then, who does?

I'd like to convince you that "YOU CAN"!

Who decides how OCaml evolves?

- Evolution
 - **User-driven:** OCaml, Python
 - **Committee-driven:** ISO/IEC evolving C and C++
 - **Vendor-driven consensus:** WebAssembly
- No separate “OCaml language” standard, separate from implementation (unlike C)
 - OCaml compiler implementation *IS* the language.

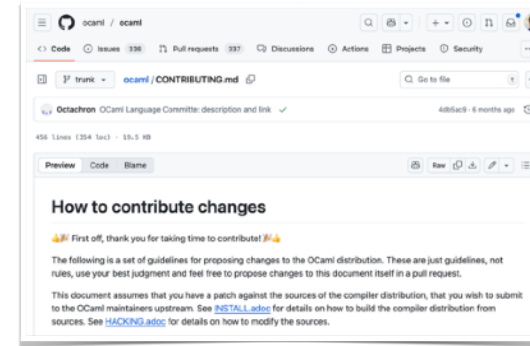


As a users of the language, OCaml users can make feature requests or contribute PRs that add features to the language. This is unlike say how C/C++ evolves with strict standardised committees driven by international standards organisation. WebAssembly follows a democratic process, but the big browser vendors essentially hold a lot of power in that they can choose to veto a feature. Only if all the vendors agree do the features get in.

Another point to highlight here is that unlike say C or C++, OCaml does not have a distinction between language and the compiler. You can buy the C language standard from ISO paying 221 Swiss francs. Whereas in OCaml, whatever the compiler accepts in a particular release is the OCaml language. The bar is somewhat lower for OCaml.

Evolving OCaml

- Open process
 - OCaml compiler is maintained on GitHub
 - All discussions are public in the PRs, Issues and RFCs on GitHub
- Multi-speed model
 - **Small fixes/features** → Make an issue (“feature request”), open a PR, discuss and get that merged
 - Every PR needs a maintainer's approval before merging
 - **Large features**
 - Bespoke based on the features
 - May need publishing papers, extensive performance evaluation, formalised/mechanised soundness results, etc.
- ***Often, presumably small feature requests take a life of their own!***



OCaml follows an open development model. The compiler source is maintained on GitHub. All the discussions in the PRs, issues and RFCs are public. For contributions, OCaml follows a multi-speed model.

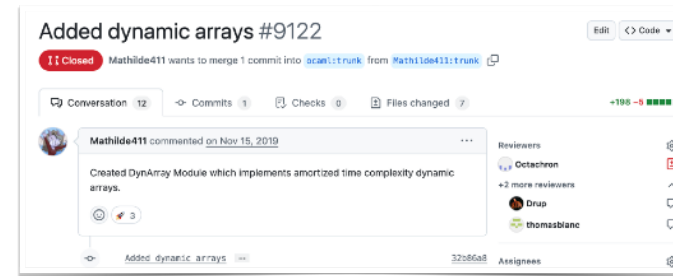
For small changes, users can make an issue requesting a feature or make a PR. The features get discussed on the thread and get merged. Every PR does need a maintainer's approval before being merged.

For larger features, the process depends on the nature of the feature. OCaml has strong roots in academia and many of the maintainers are part of the French Computer Science Research lab INRIA. Hence, we value peer-reviewed academic papers for complex changes. Some features may also require extensive performance evaluation and formalised and mechanised soundness results.

Often, presumably small feature requests may take a life of their own.

A small(?) change — Dynamic Arrays

- Student request + PR (#9122)
 - **Opened:** Nov 15, 2019
 - Gained interest from various quarters
 - Used unsafe features of the language
 - PR closed the same day to discuss design elsewhere
 - **Closed:** Nov 15, 2019

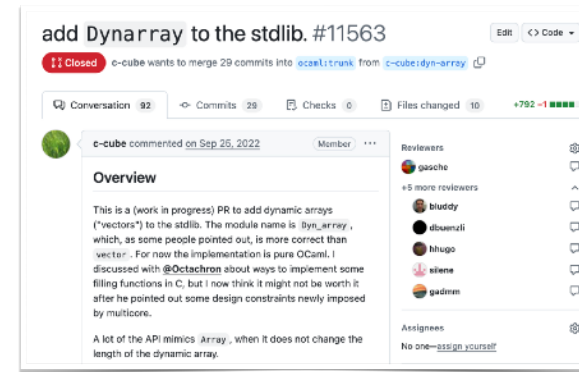


Let me talk about one such feature.

In 2019, a student observed that OCaml does not have dynamic arrays, the equivalent of C++ vectors. So they proposed a PR with that features. The feature immediately gathered a lot of interest from various users. But the implementation used several unsafe features in the language. It was quickly decided that the feature needs more discussions and the PR was closed the same day by an OCaml maintainer.

Dynamic Arrays

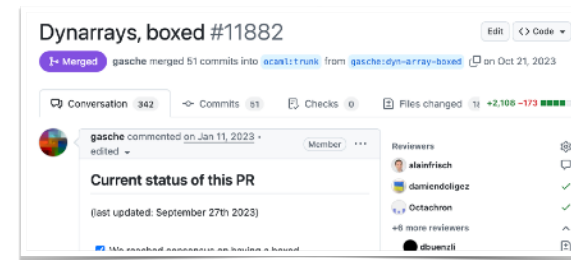
- PR #11563
 - **Opened:** Sep 25, 2022
 - Better-designed API from an expert contributor
 - Discussion on representation, performance and multicore-safety
 - **Closed:** Jan 18, 2023



A few years later, an expert OCaml user (who is not a maintainer) proposed the feature again, this time paying attention to the API. There was a long discussion on the representation of the data structure, its performance and multicore-safety. The PR was closed after a few months by the contributor.

Dynamic Arrays

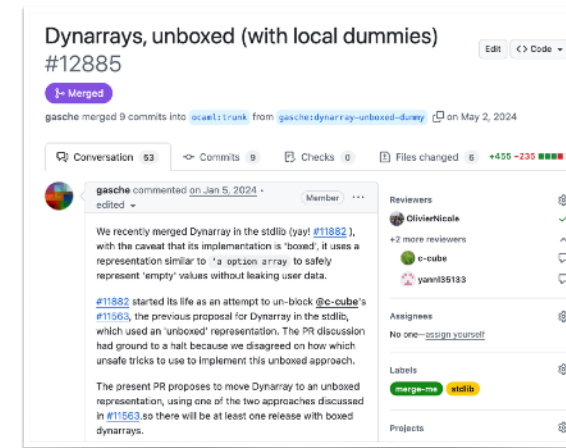
- PR #11563
 - **Opened:** Jan 11, 2023
 - Safer-API but less-optimised representation
 - **Merged:** Oct 21, 2023
 - **Released:** OCaml 5.2.0 (May 13, 2024)



In the mean time, a separate PR was opened by an OCaml maintainer, who went for a less ambitious solution. It used a safe API but a safer but less-optimised representation. This PR was discussed extensively, and was merged after 10 months. The feature was released in OCaml 5.2.

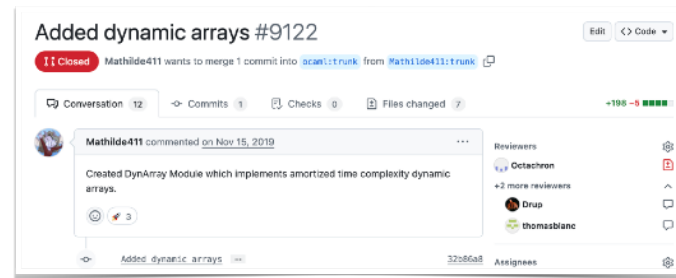
Dynamic Arrays

- PR #12885
 - **Opened:** Jan 5, 2024
 - Optimised (unboxed) and safe version
 - **Merged:** May 2, 2024
 - **Released:** OCaml 5.3.0 (Jan 8, 2025)



The same contributor then later opened a separate PR for an optimised version of the API. This time, it was much easier to discuss the feature because the API had been discussed extensively. This was merged after a few months and landed in OCaml 5.3.

Dynamic Arrays



- **Summary**
 - Proposed — Nov 2019, Merged — (PR#1) Jan 2024; (PR#2) May 2024
 - Initially — 198 loc, finally — ~2500 loc
 - 500+ comments in the various PRs
- Worth it?
 - **Yes!** Should work for a couple of decades.
 - Harder to undo changes after the release.

If you compare the original PR to the couple that landed, the original PR was proposed in 2019. The first of the PRs landed in Jan 2024. While the initial PR was 198 LOC the merged ones was 2500 lines of code in total. There were overall 500+ discussion comments.

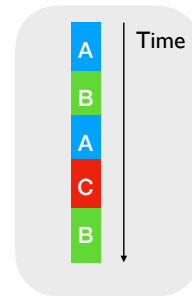
This may seem like an overkill. Is this discussion worth it?

The answer is a firm yes! We expect code that uses dynamic arrays to work for the next couple of decades. So it is important that we put in the effort to get this right. It is hard to undo changes once released.

A large change — Multicore OCaml

- Native support for concurrency and parallelism to OCaml

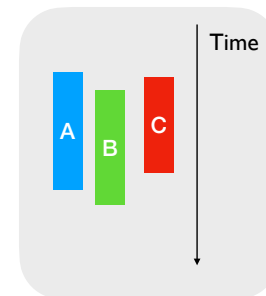
Concurrency



*Interleaved
execution*

Effect Handlers

Parallelism



*Simultaneous
execution*

Domains

<https://tarides.com/blog/2023-03-02-the-journey-to-ocaml-multicore-bringing-big-ideas-to-life/>

A big feature that we merged in the recent years is the Multicore OCaml project. Multicore OCaml aims to add native support for concurrency and parallelism to OCaml. This is done with the help of two independent features called the Effect Handlers and Domains.

Challenges

- A new multicore garbage collector and multicore runtime system
 - *Replacing a car engine with a new one!*
- Make the language itself thread-safe
 - OCaml is a safe language! (Unlike C/C++, Go)
- Maintain feature and performance backwards compatibility!
 - Most OCaml programs will continue to remain single-threaded

Build credibility by *publishing key results* and *rigorous evaluation*

<https://tarides.com/blog/2023-03-02-the-journey-to-ocaml-multicore-bringing-big-ideas-to-life/>

We had a number of challenges in getting multicore OCaml into OCaml.

First, Multicore OCaml needed a new multicore capable garbage collector and a multicore runtime system. A good analogy is replacing the engine of a car with the new one.

Secondly, for the 25+ years OCaml had been in existence at this point, all the code was sequential and there was millions of lines of code out there already. All of this code was written without concurrency in mind. Making the language multicore means that we needed to think about thread safety. Unlike Go and C++, which may crash when you have concurrency bugs, OCaml is a safe language, which MUST NOT crash if the program compiles. This required us to give strict semantics for thread-safety.

Finally, even when the language becomes multicore, it is expected that majority of programs will still continue to be single threaded. We wanted to make sure that those programs continued to work and just as well in the multicore version of the language.

Our approach was to build credibility for this work by publishing key results in academic conference and perform rigorous, continuous evaluation.

Starting out

Multicore OCaml

Stephen Dolan Leo White Anil Madhavapeddy

Currently, threading is supported in OCaml only by means of a global lock, allowing at most thread to run OCaml code at any time. We present ongoing work to design and implement an OCaml runtime capable of shared-memory parallelism.

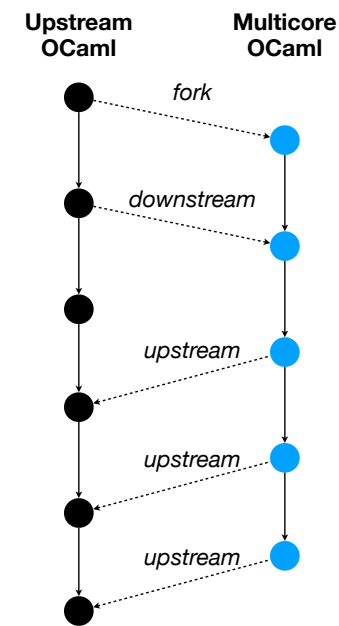
1 Introduction

Adding shared-memory parallelism to an existing lan-

all objects reachable from it to be promoted to the shared heap en masse. Unfortunately this eagerly promotes many objects that were never really shared: just because an object is pointed to by a shared object does not mean another thread is actually going to attempt to access it.

Our design is similar but lazier, along the lines of the multicore Haskell work [2], where objects are promoted to the shared heap whenever another thread

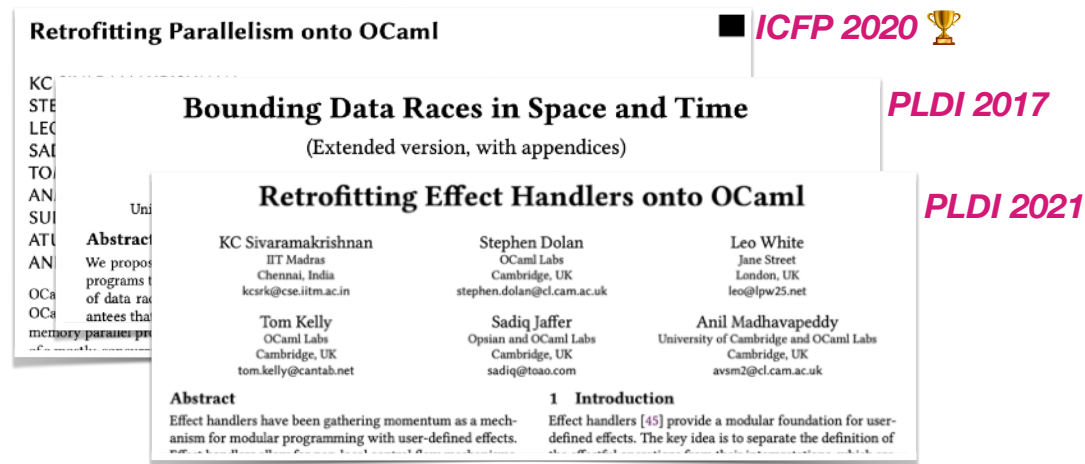
OCaml Workshop 2014



Multicore OCaml started out in 2014.

Our aim was to fork the compiler into a separate project. The two projects would continue to evolve with Multicore OCaml down streaming changes from the upstream compiler. Finally, when we were confident about the compiler, our aim was to upstream features piecewise to OCaml.

Building confidence — Papers

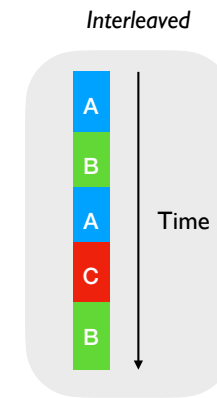


Peer-reviewed ideas build confidence

We published a number of papers around the design on top academic venues.

The point of publishing the papers is to build the confidence into these ideas with the help of peer-review. OCaml started at INRIA, which is a French computer science research lab. And hence, the language values academic rigour.

Concurrency



Concurrent Programming

- Computations may be *suspended* and *resumed* later
- Many languages provide concurrent programming mechanisms as *primitives*
 - ✦ **async/await** — JavaScript, Python, Rust, C# 5.0, F#, Swift, ...
 - ✦ **generators** — Python, Javascript, ...
 - ✦ **coroutines** — C++, Kotlin, Lua, ...
 - ✦ **futures & promises** — JavaScript, Swift, ...
 - ✦ **Lightweight threads/processes** — Haskell, Go, Erlang
- *Often include many different primitives in the same language!*
 - ✦ JavaScript has async/await, generators, promises, and callbacks

Don't want a **zoo** of primitives but
want ***expressivity***

What's the ***smallest*** primitive that
expresses ***many*** concurrency patterns?

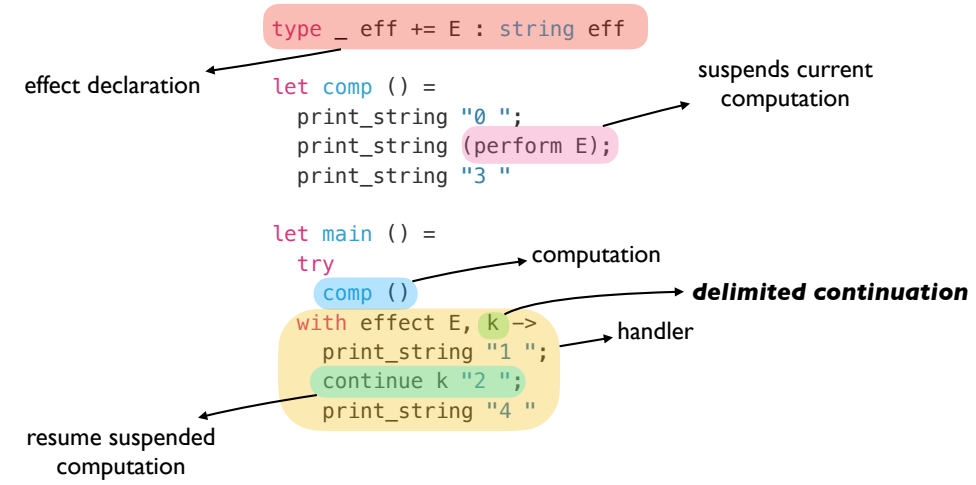
Effect handlers

- A mechanism for programming with *user-defined effects*
- *Modular* and *composable* basis of non-local control-flow mechanisms
 - ✦ Exceptions, generators, lightweight threads, promises, asynchronous IO, coroutines as *libraries*
- Effect handlers \sim *first-class, restartable exceptions*
 - ✦ Structured programming with *delimited continuations*

<https://github.com/ocaml-multicore/effects-examples>

- Direct-style asynchronous I/O
- Generators
- Resumable parsers
- Probabilistic Programming
- Reactive UIs
-

Effect handlers



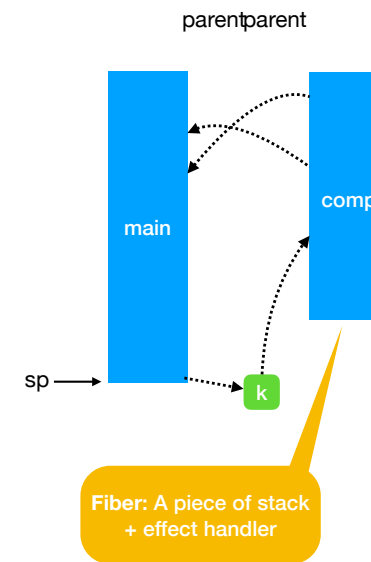
Stepping through the example

```
type 'a eff += E : string eff

let comp () =
  print_string "0 ";
  print_string (perform E);
  print_string "3 "

let main () =
  pc → try
    comp ()
  with effect E, k ->
    print_string "1 ";
    continue k "2 ";
    print_string "4 "
```

0 1 2 3 4



Lightweight threading

```
type _ eff += Fork : (unit -> unit) -> unit eff
           | Yield : unit eff

let run main =
  ... (* assume queue of continuations *)
  let run_next () =
    match dequeue () with
    | Some k -> continue k ()
    | None -> ()
  in
  let rec spawn f =
    match f () with
    | () -> run_next () (* value case *)
    | effect Yield, k -> enqueue k; run_next ()
    | effect (Fork f), k -> enqueue k; spawn f
  in
  spawn main

let fork f = perform (Fork f)
let yield () = perform Yield
```

Effect Handler {

Lightweight threading

```
let main () =  
  fork (fun _ ->  
    print_endline "1.a";  
    yield ();  
    print_endline "1.b");  
  fork (fun _ ->  
    print_endline "2.a";  
    yield ();  
    print_endline "2.b")  
;;  
run main
```

```
1.a  
2.a  
1.b  
2.b
```


Lightweight threading

```
let main () =  
  fork (fun _ ->  
    print_endline "1.a";  
    yield ();  
    print_endline "1.b");  
  fork (fun _ ->  
    print_endline "2.a";  
    yield ();  
    print_endline "2.b")  
;;  
run main
```

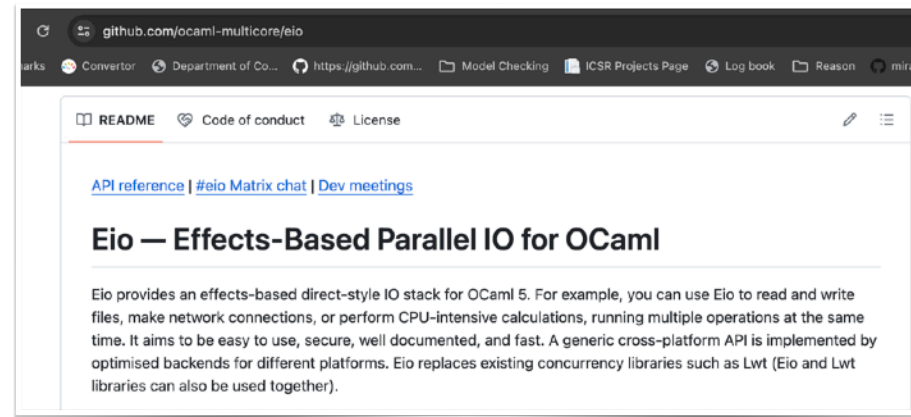
Ability to specialise
scheduler
unlike GHC Haskell / Go

User-code need not be
aware of effects

1.a
2.a
1.b
2.b

Industrial-strength concurrency

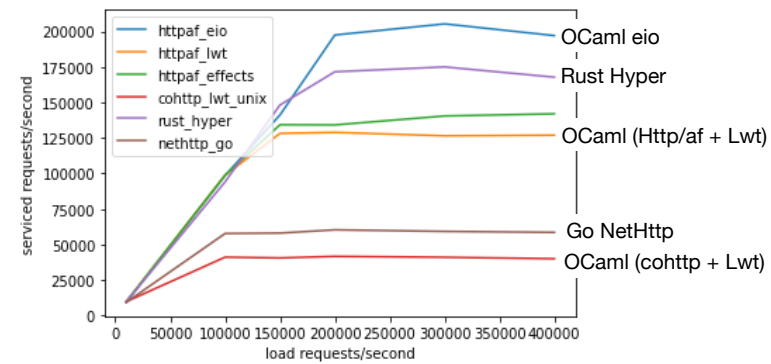
- **eio**: effects-based direct-style I/O
 - ✦ Multiple backends — epoll, select, ***io_uring*** (*new async io in Linux kernel*)



<https://github.com/ocaml-multicore/eio>

Industrial-strength concurrency

- **eio**: effects-based direct-style I/O
 - ✦ Multiple backends — epoll, select, ***io_uring*** (*new async io in Linux kernel*)

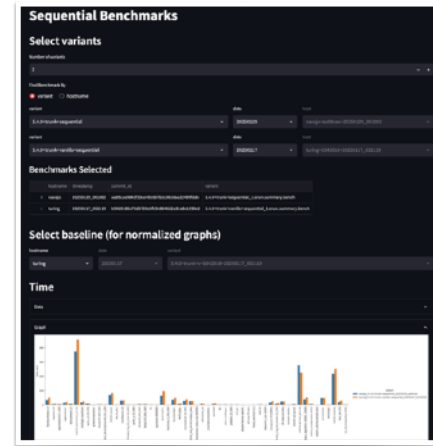


100 open connections, 60 seconds w/ io_uring

<https://github.com/ocaml-multicore/eio>

Building confidence — Benchmarking

- **Rigorous**, **continuous** benchmarking on **real-world programs**
- sandmark.tarides.com — Benchmark suite, Infra and runners



Apart from papers, we also spent significant engineering time on rigorous evaluation.

We developed Sandmark, a suite of real world benchmarks, infrastructure and runners to continuously run the benchmarks on the two different compilers. The benchmarks were run nightly and gave us the confidence that the compiler met the performance expectations.

Building confidence — CI for package universe

- *Can the new compiler build the existing universe?*
 - Build the OPAM universe of packages against **upstream** and **multicore** compilers

	4.14	5.0+alpha-repo	number of revdeps
0install.2.18	✅	✅	1
BetterErrors.0.0.1	✅	❌	7
TCSLib.0.3	✅	❌	1
absolute.0.1	✅	✅	0
acgtk.1.5.3	✅	❌	0
adv.2.0.0	✅	✅	0
aez.0.3	✅	❌	0
ahrocksdb.0.2.2	❌	❌	0
aio.0.0.3	✅	❌	0
alt-ergo-free.2.2.0	✅	❌	7
amqp-client-async.2.2.2	✅	✅	0
amqp-client-lwt.2.2.2	✅	✅	0
ancient.0.9.1	✅	❌	0
apron.v0.9.13	✅	✅	17

You can contribute to the compiler development without hacking on the compiler

We also developed a tool call OPAM health check. The idea here is that, with the two compilers, the upstream one and the multicore one, we will build the universe of packages from OPAM, and publicly produce the table that you see there. Every green box says that the package successfully builds with a compiler. OTOH, the red box says that the package fails to build with the new compiler. Yellow one represents that a dependency of this package fails to build.

We proactively went ahead and submitted PRs to various packages to fix this errors. Since we designed the language to mostly be compatible, the fixes were small, easy to review and merge.

Multicore OCaml — Process

- **Nov 2021** — Week-long design discussion meetings among maintainers
 -*A few months of iteration to fix design issues and bugs....*
- **Merged** — Jan 10 2022
 -*A few months of iteration to fix design issues and bugs....*
- **Released** — Dec 16 2022, as OCaml 5.0



In terms of process, in Nov 2021, we had a week long discussion amongst the developers to discuss the design. After which we worked on the design for a couple of months.

In January 2022, the multicore PR was merged upstream! Even after this, we continued working for a couple of months to fix outstanding issues and bugs.

Finally, the multicore OCaml features were released as OCaml 5.0.

Since then there has been a long tail of adding missing features, bug fixes and performance improvements over the series of releases. We're still hoping to fix a few more performance issues in future releases.

Multicore OCaml — Process

- **Nov 2021** — Week-long design discussion meetings among maintainers
 -A few months of iteration to fix design issues and bugs....
- **Merged** — Jan 10 2022
 -A few months of iteration to fix design issues and bugs....
- **Released** — Dec 16 2022, as OCaml 5.0
- **Long tail** of adding missing features, bug fixes and performance improvements
 - 5.1 — Sep 2023
 - 5.2 — May 2024
 - 5.3 — January 2025
 - 5.4 — Sep 2025



In terms of process, in Nov 2021, we had a week long discussion amongst the developers to discuss the design. After which we worked on the design for a couple of months.

In January 2022, the multicore PR was merged upstream! Even after this, we continued working for a couple of months to fix outstanding issues and bugs.

Finally, the multicore OCaml features were released as OCaml 5.0.

Since then there has been a long tail of adding missing features, bug fixes and performance improvements over the series of releases. We're still hoping to fix a few more performance issues in future releases.

What's next for OCaml?

- Bridging the performance and safety gap between OCaml and Rust
 - **Data-race-free parallelism** through **modes**
- Draws lessons from Multicore OCaml execution
 - In production at Jane Street
 - Valuable user-feedback-oriented design
- Several award-winning papers
 - POPL, ICFP, OOPSLA
- CI for the external universe
 - <https://oxcaml.check.ci.dev/>



<https://oxcaml.org>

What about the future of OCaml?

The big ongoing project now is OxCaml, which aims to bridge the gap between OCaml and Rust. 95% of programs that I write, I am happy with OCaml's use of GC and high-level features. For the 5%, I require rust like control over memory and safety. OxCaml is a project that aims to bridge this gap. This project is led by Jane Street.

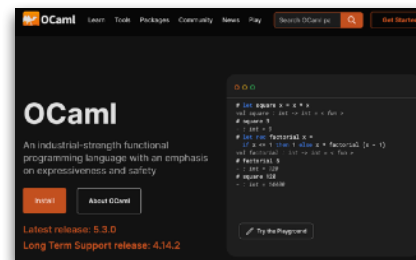
While it draws lessons from Multicore OCaml execution, there are a few things that are different. OxCaml is in production at Jane Street, which provides valuable feedback from real users. The features are much more experimental and the cost of change is far less compared to releasing the code for all OCaml users.

The OxCaml team has published several award-winning papers at top conferences. There is also a CI similar to OPAM health check for the ecosystem compatibility.

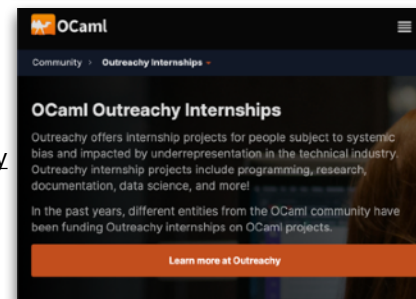
OxCaml features may land in OCaml in the next couple of years.

Get Involved!

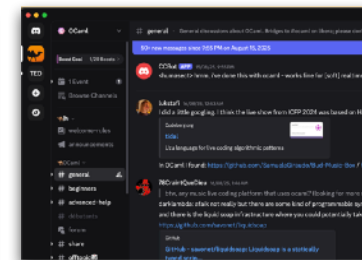
ocaml.org



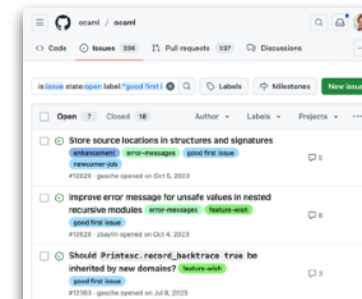
ocaml.org/outreachy



OCaml
Discord



github.com/ocaml



If all of this sounds interesting, please get involved.

The first stop should be OCaml.org, which is the home of the OCaml language and the ecosystem.

We have an active discord channel with lots of helpful users.

If you are looking to learn how to contribute, OCaml participates in Outreachy internships, which are targeted at underrepresented people.

Finally, you can just go to the OCaml compiler on Github and start looking at the issues and make contributions!