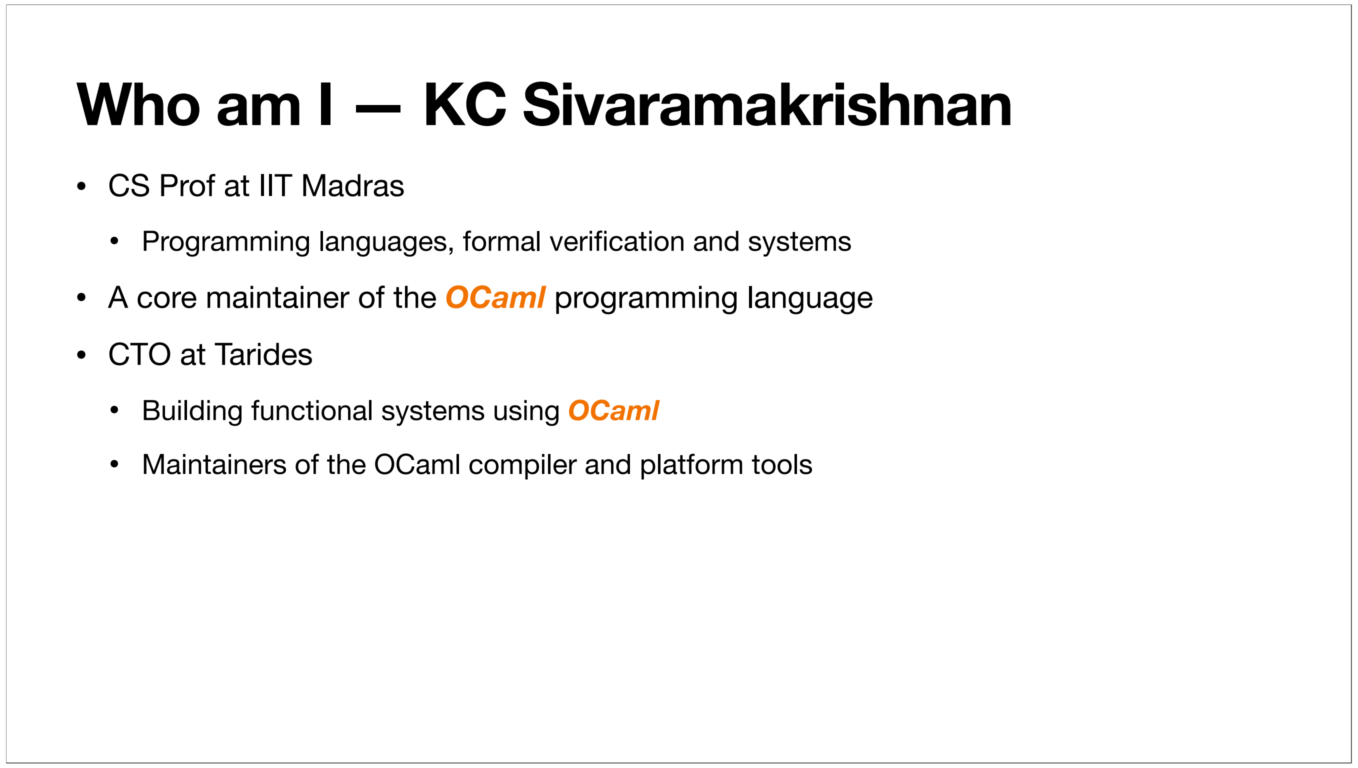
Evolving the OCaml programming language

KC Sivaramakrishnan
kcsrk.info

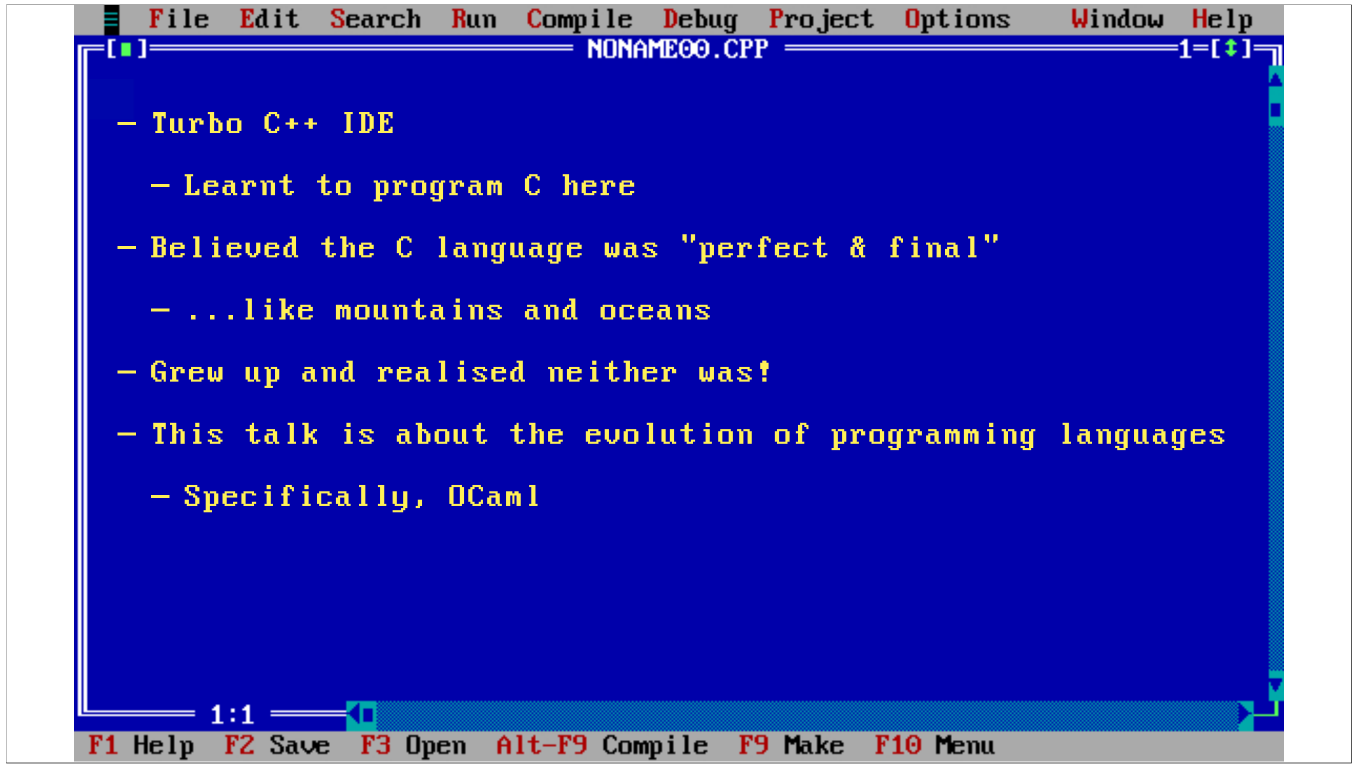
IICT
27th September 2025



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
    - 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.

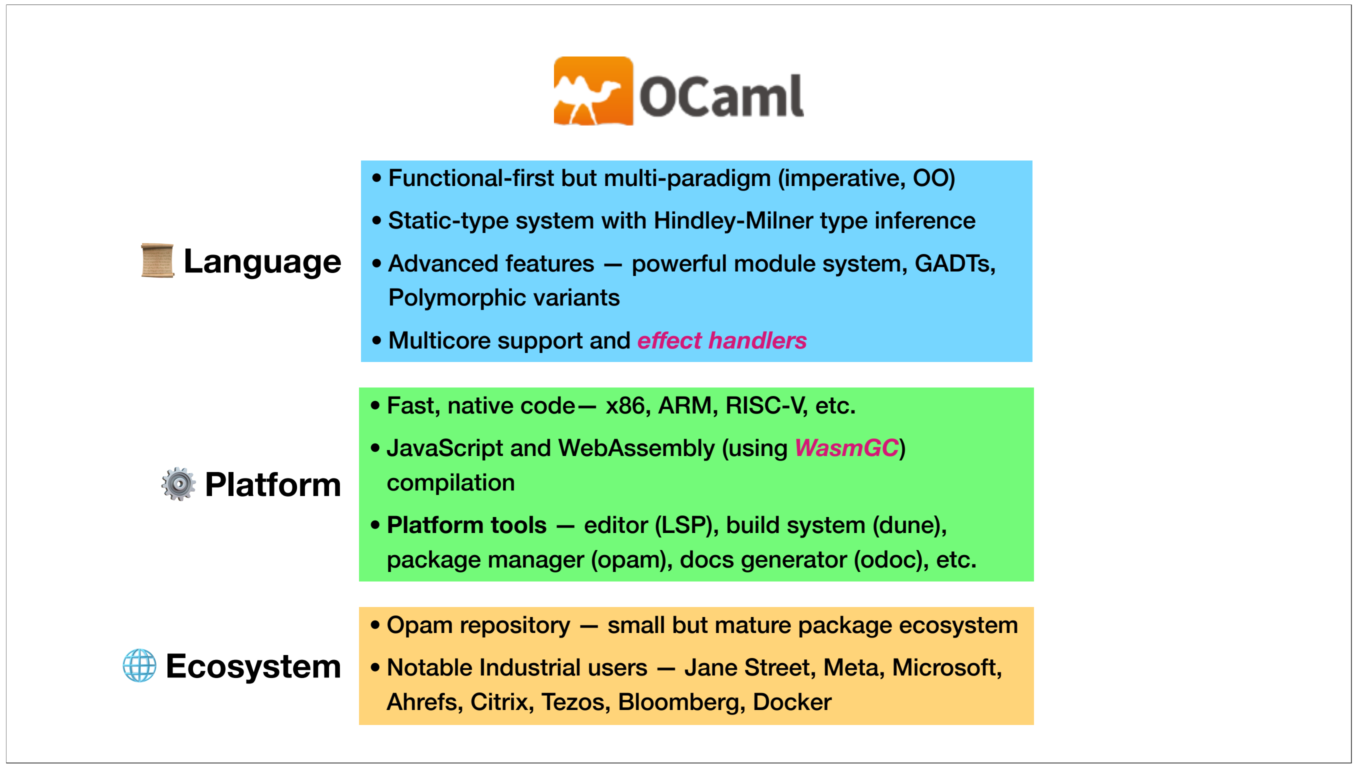

File Edit Search Run Compile Debug Project Options Window Help
NONAME00.CPP
1:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

- Turbo C++ IDE
  - Learnt to program C here
- Believed the C language was "perfect & final"
  - ...like mountains and oceans
- Grew up and realised neither was!
- This talk is about the evolution of programming languages
  - Specifically, OCaml

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. Likewise Programming Languages also change, but it is 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

Language

- Functional-first but multi-paradigm (imperative, OO)
- Static-type system with Hindley-Milner type inference
- Advanced features — powerful module system, GADTs, Polymorphic variants
- Multicore support and *effect handlers*

Platform

- Fast, native code— x86, ARM, RISC-V, etc.
- JavaScript and WebAssembly (using *WasmGC*) compilation
- **Platform tools** — editor (LSP), build system (dune), package manager (opam), docs generator (odoc), etc.

Ecosystem

- Opam repository — small but mature package ecosystem
- Notable Industrial users — Jane Street, Meta, Microsoft, Ahrefs, Citrix, Tezos, Bloomberg, Docker

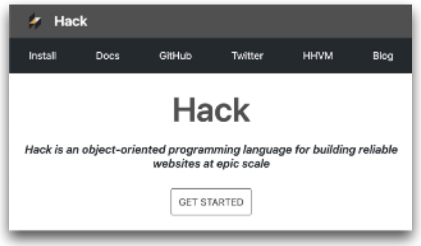
OCaml is a functional first programming language, but it also has first-class support for imperative and object-oriented programming. It is a statically typed language, with Hindley-Milner type inference. It has a number of advanced features such as a powerful module system, which makes it suitable for 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 target 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



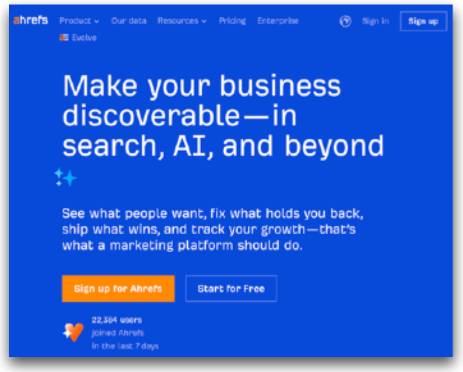
Hack

Hack is an object-oriented programming language for building reliable websites at epic scale

The Rust Programming Language

WA

Compilers



Make your business discoverable—in search, AI, and beyond

See what people want, fix what holds you back, ship what wins, and track your growth—that's what a marketing platform should do.

Web Frontend

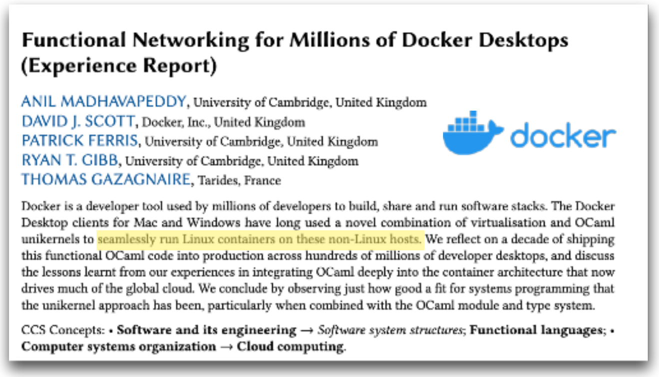
One of the reasons OCaml is favoured by its users is its high-dynamic range. OCaml is generally good at many things. From scripts to scalable systems, to research prototypes to production infrastructure. Here are a few examples.

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



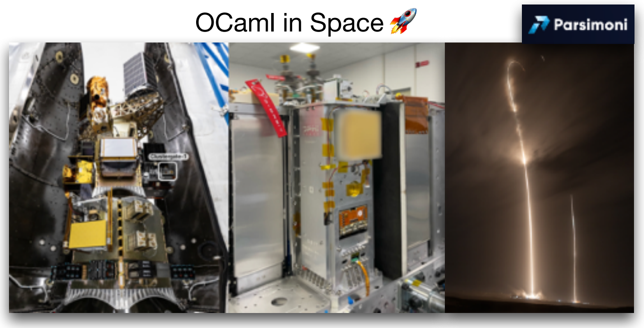
Functional Networking for Millions of Docker Desktops (Experience Report)

ANIL MADHAVAPEDDY, University of Cambridge, United Kingdom
DAVID J. SCOTT, Docker, Inc., United Kingdom
PATRICK FERRIS, University of Cambridge, United Kingdom
RYAN T. GIBB, University of Cambridge, United Kingdom
THOMAS GAZAGNAIRE, Tarides, France

Docker is a developer tool used by millions of developers to build, share and run software stacks. The Docker Desktop clients for Mac and Windows have long used a novel combination of virtualisation and OCaml unikernels to seamlessly run Linux containers on these non-Linux hosts. We reflect on a decade of shipping this functional OCaml code into production across hundreds of millions of developer desktops, and discuss the lessons learnt from our experiences in integrating OCaml deeply into the container architecture that now drives much of the global cloud. We conclude by observing just how good a fit for systems programming that the unikernel approach has been, particularly when combined with the OCaml module and type system.

CCS Concepts: • **Software and its engineering** → *Software system structures*; **Functional languages**; • **Computer systems organization** → **Cloud computing**.

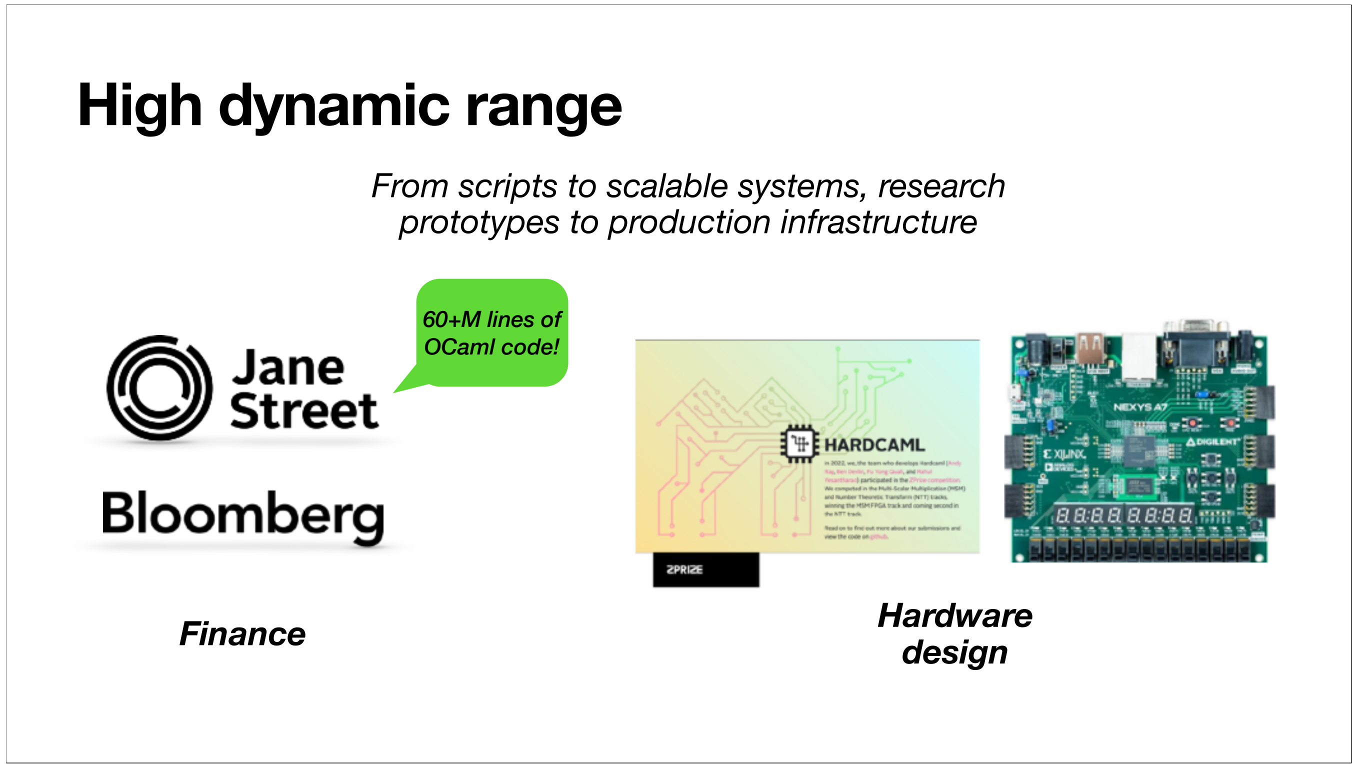
OCaml in Space 🚀



Virtualisation and Networking

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

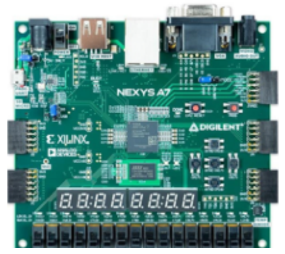
From scripts to scalable systems, research prototypes to production infrastructure

Jane Street

60+M lines of OCaml code!

Bloomberg

Finance



Hardware design

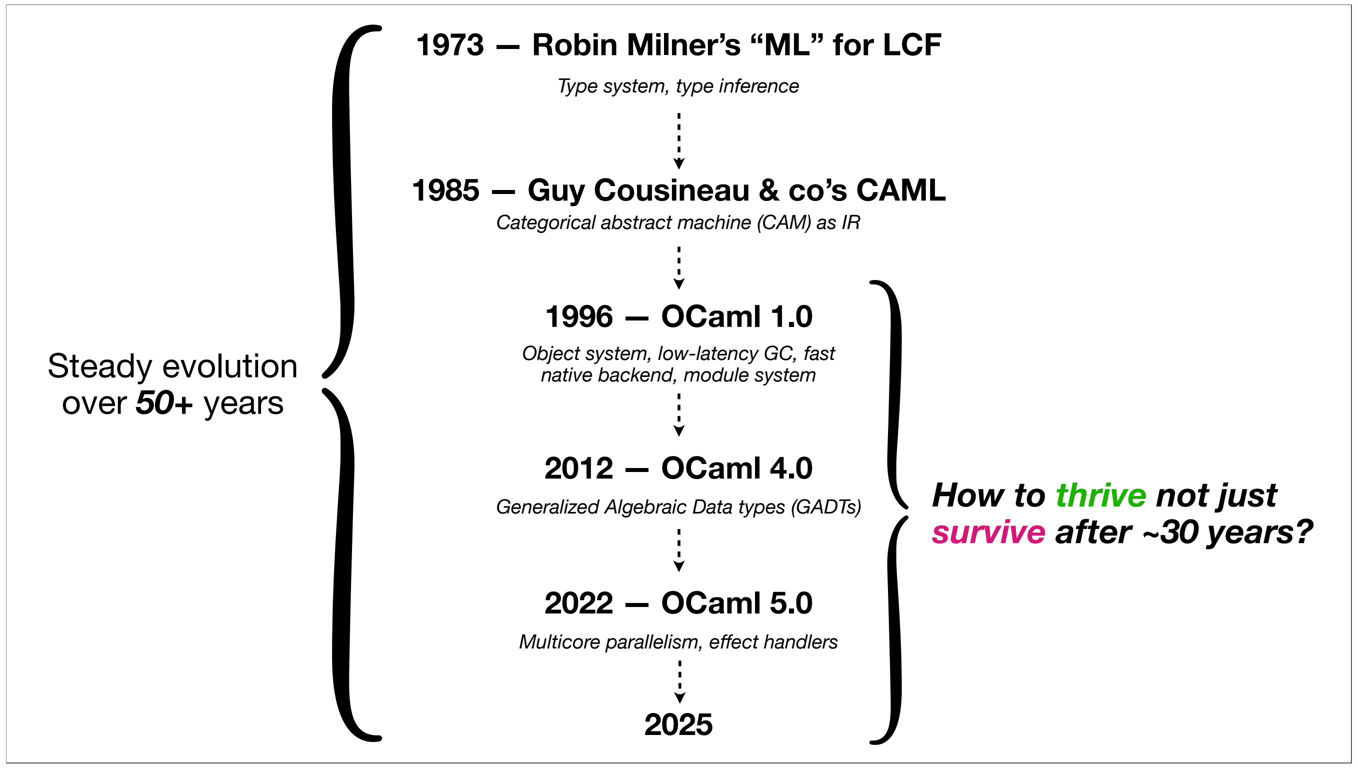
OCaml is widely used in Finance. Jane Street, a quant firm, is the largest user of OCaml in the industry with more than 60 million lines of code. OCaml is used to write all the software from trading systems to UIs for traders.

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.

OCaml

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.

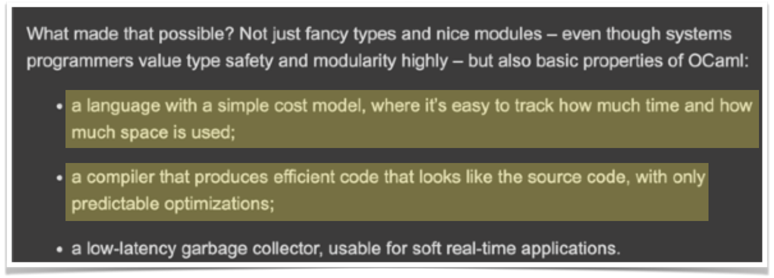

1973 — Robin Milner's "ML" for LCF
Type system, type inference
1985 — Guy Cousineau & co's CAML
Categorical abstract machine (CAM) as IR
1996 — OCaml 1.0
Object system, low-latency GC, fast native backend, module system
2012 — OCaml 4.0
Generalized Algebraic Data types (GADTs)
2022 — OCaml 5.0
Multicore parallelism, effect handlers
2025
Steady evolution over 50+ years
How to thrive not just survive after ~30 years?

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:

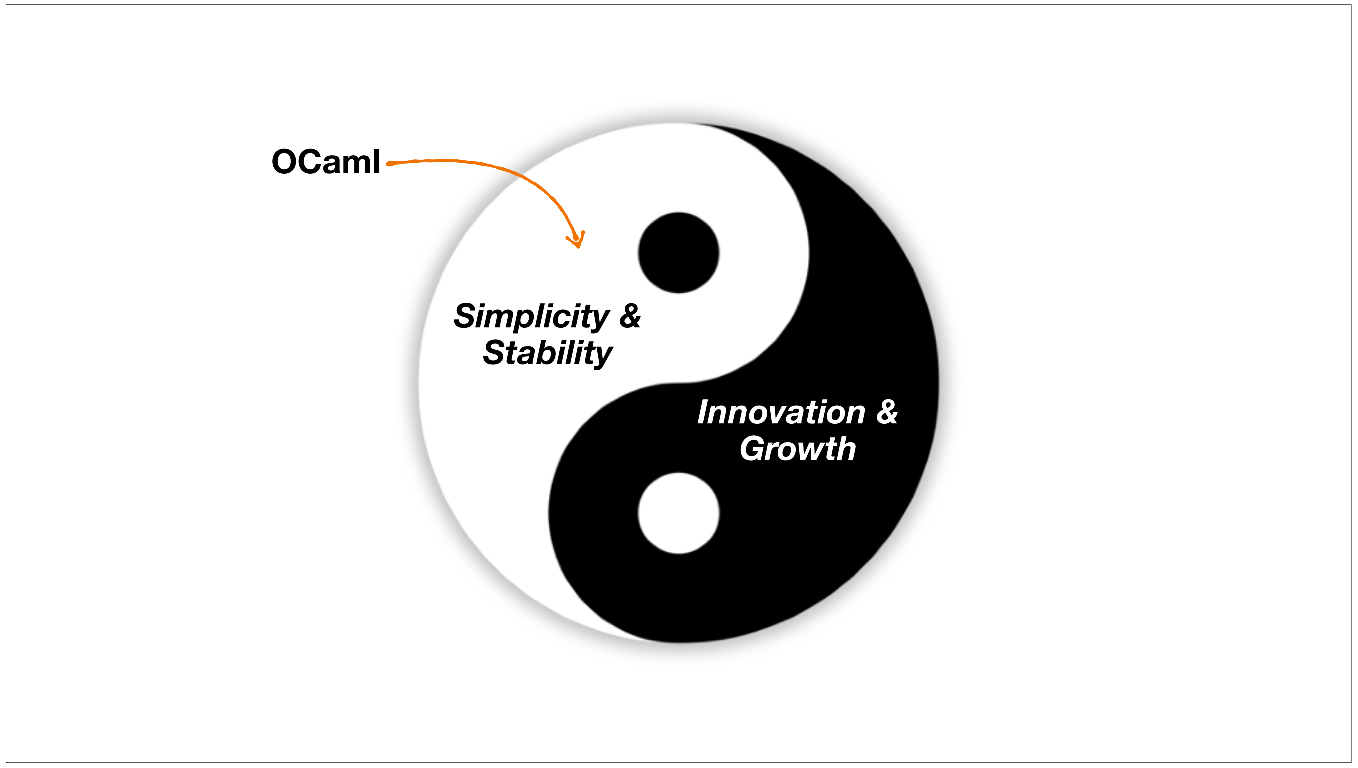
- a language with a simple cost model, where it's easy to track how much time and how much space is used;
- a compiler that produces efficient code that looks like the source code, with only predictable optimizations;
- a low-latency garbage collector, usable for soft real-time applications.

- 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!

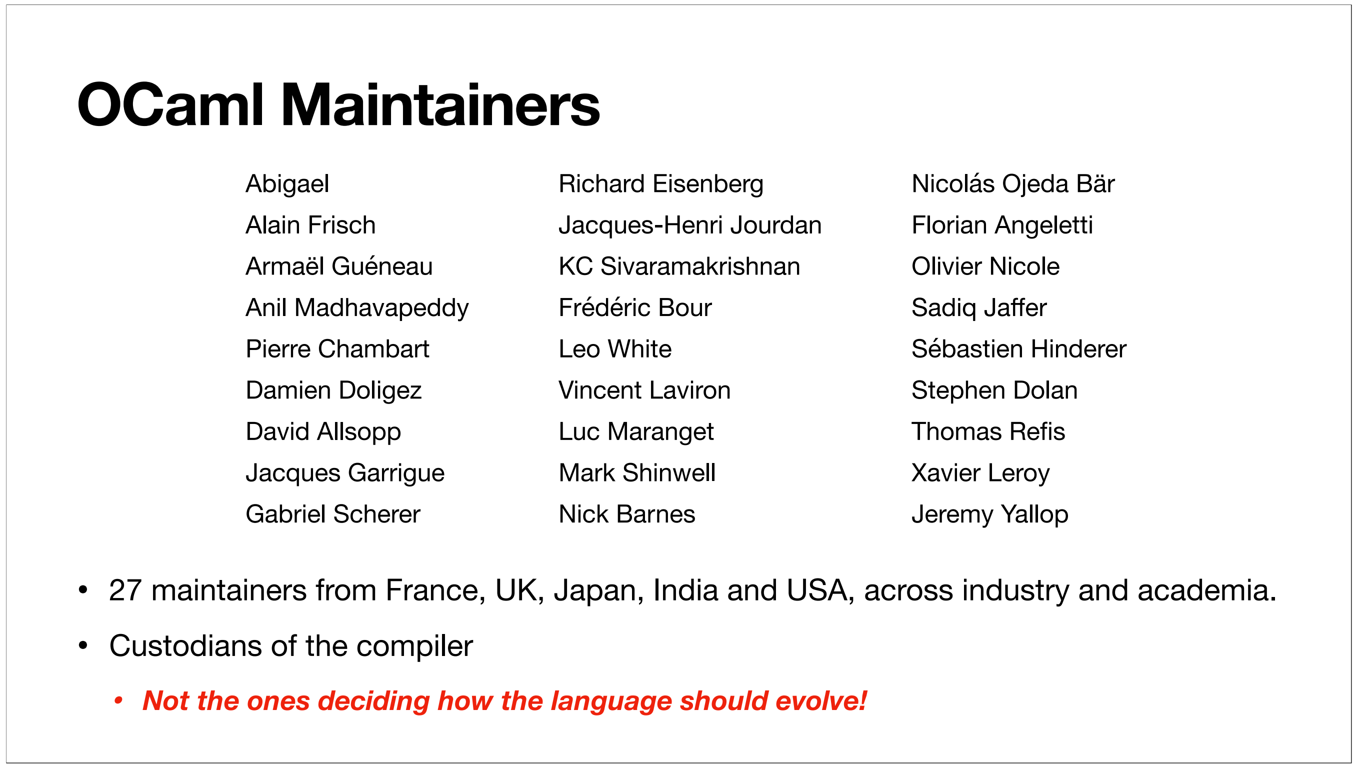
The secret sauce in the language is “simplicity” and “stability”

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.

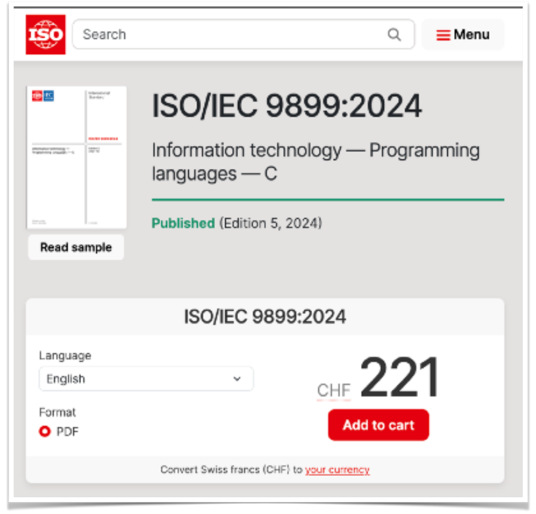
Who decides how OCaml evolves?

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
- **Language** and **compiler** aren't distinct
  - OCaml compiler implementation ***IS*** the language.
    - Unlike C, Wasm, JavaScript
  - *Bar is lower to change the language*



ISO/IEC 9899:2024

Information technology — Programming languages — C

Published (Edition 5, 2024)

ISO/IEC 9899:2024

Language: English

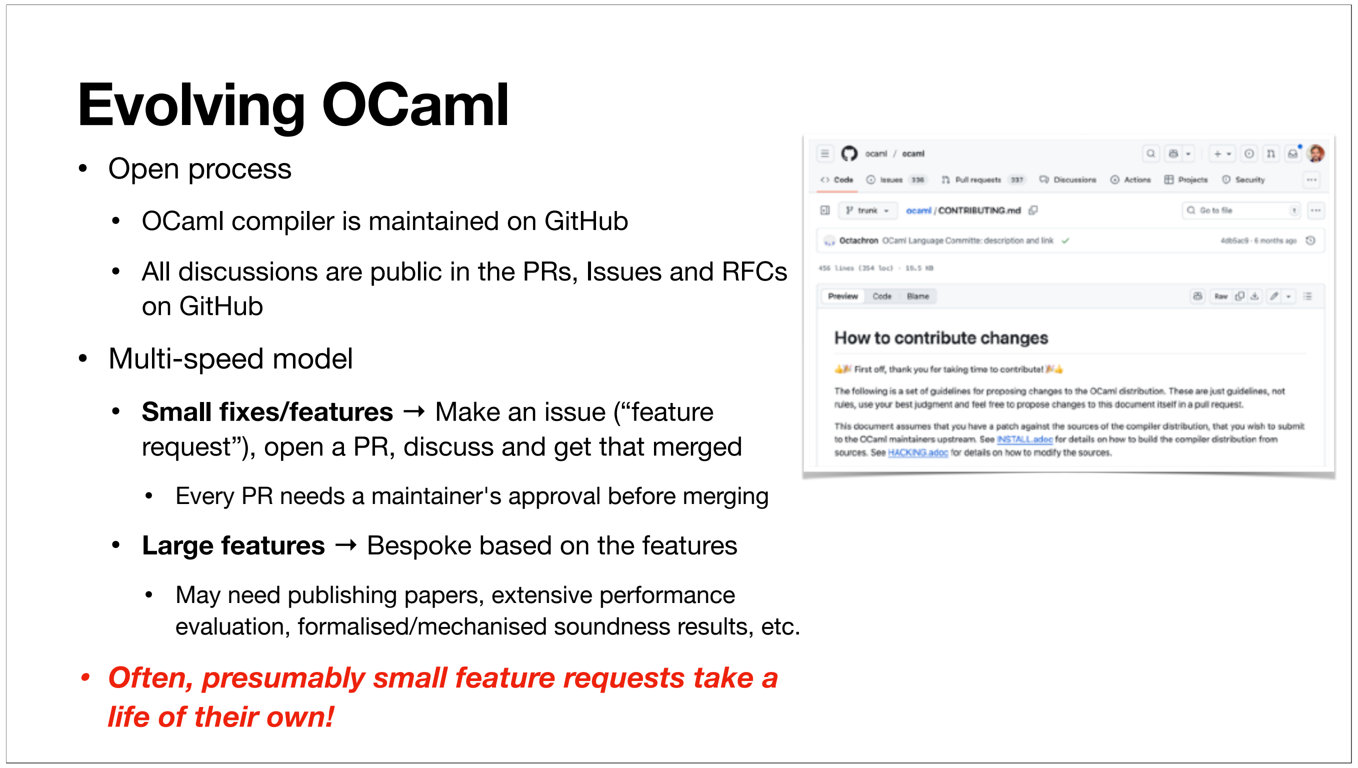
Format: PDF

CHF 221

Convert Swiss francs (CHF) to your currency

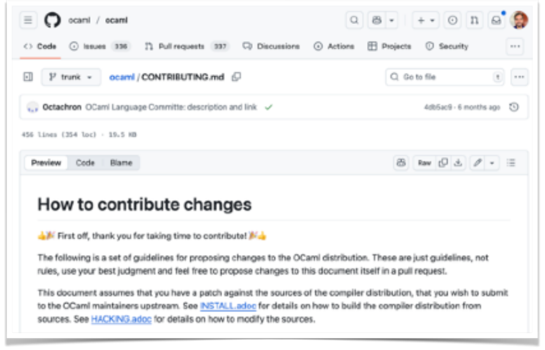
As a users of the language, OCaml users can make feature requests or contribute PRs that add features to the language. This is unlike say how C/C++ evolves with strict standardised committees driven by international standards organisation. WebAssembly follows a democratic process, but the big browser 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 / CONTRIBUTING.md

Octachron OCaml Language Committe: description and link ✓ 4db5ac9 · 6 months ago

456 lines (354 loc) · 19.5 KB

How to contribute changes

👍🎉 First off, thank you for taking time to contribute! 🎉👍

The following is a set of guidelines for proposing changes to the OCaml distribution. These are just guidelines, not rules, use your best judgment and feel free to propose changes to this document itself in a pull request.

This document assumes that you have a patch against the sources of the compiler distribution, that you wish to submit to the OCaml maintainers upstream. See INSTALL.adoc for details on how to build the compiler distribution from sources. See HACKING.adoc for details on how to modify the sources.

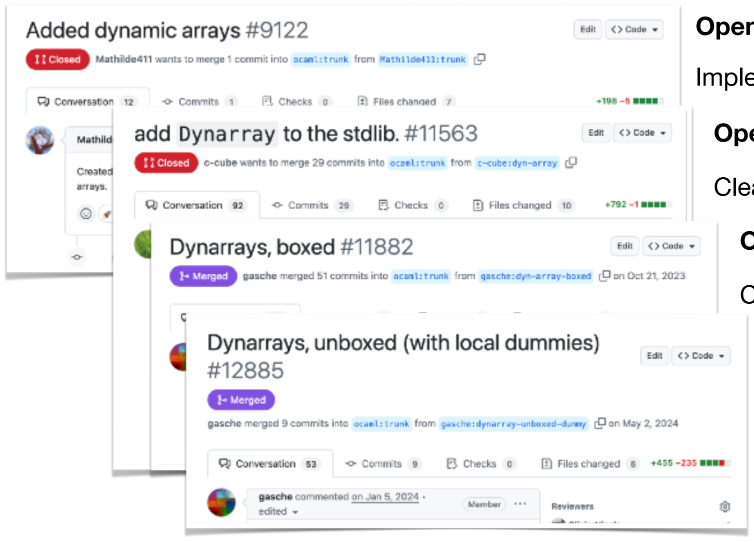
OCaml follows an open development model. The compiler source is maintained on GitHub. All the discussions in the PRs, issues and RFCs are public. For contributions, OCaml follows a multi-speed model.

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


Added dynamic arrays #9122

Closed Mathilde411 wants to merge 1 commit into `ocaml:trunk` from `Mathilde411:trunk`

Conversation 12 · Commits 1 · Checks 0 · Files changed 7 · +198 −5

add `Dynarray` to the stdlib. #11563

Closed c-cube wants to merge 29 commits into `ocaml:trunk` from `c-cube:dyn-array`

Conversation 92 · Commits 29 · Checks 0 · Files changed 10 · +792 −1

Dynarrays, boxed #11882

Merged gasche merged 51 commits into `ocaml:trunk` from `gasche:dyn-array-boxed` on Oct 21, 2023

Dynarrays, unboxed (with local dummies) #12885

Merged gasche merged 9 commits into `ocaml:trunk` from `gasche:dynarray-unboxed-dummy` on May 2, 2024

Conversation 53 · Commits 9 · Checks 0 · Files changed 6 · +455 −235

gasche commented on Jan 5, 2024 · edited · Member

Reviewers

Opened: Nov 15, 2019, **Closed:** Nov 15 2019

Implementation rather naive, room for improvements

Opened: Sep 25, 2022, **Closed:** Jan 18, 2023

Clean API, ***but*** multicore safety, performance

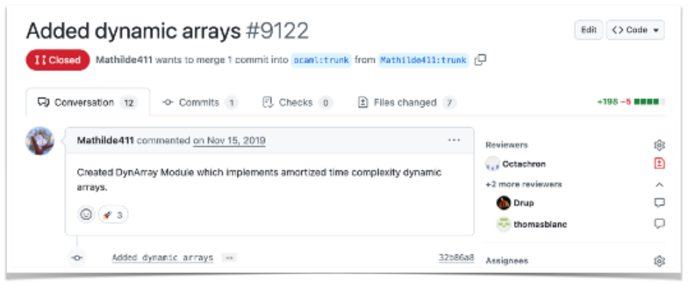
Opened: Jan 11, 2023, **Merged:** Oct 21, 2023

Clean API ***and*** simple implementation

Opened: Jan 5, 2024, **Merged:** May 2, 2024

Clean API and ***optimised*** implementation

Dynamic Arrays


Added dynamic arrays #9122

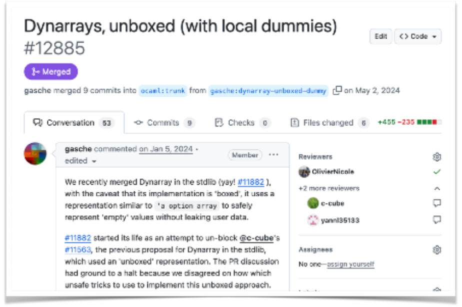
Closed Mathilde411 wants to merge 1 commit into ocaml:trunk from Mathilde411:trunk

Conversation 12 Commits 1 Checks 0 Files changed 7 +198 −5

Mathilde411 commented on Nov 15, 2019

Created DynArray Module which implements amortized time complexity dynamic arrays.

Added dynamic arrays 32b86a8


Dynarrays, unboxed (with local dummies) #12885

Merged gasche merged 9 commits into ocaml:trunk from gasche:dynarray-unboxed-dummy on May 2, 2024

Conversation 53 Commits 9 Checks 0 Files changed 6 +455 −235

gasche commented on Jan 5, 2024 · edited

We recently merged Dynarray in the stdlib (yay! #11882), with the caveat that its implementation is 'boxed', it uses a representation similar to `'a option array` to safely represent 'empty' values without leaking user data.

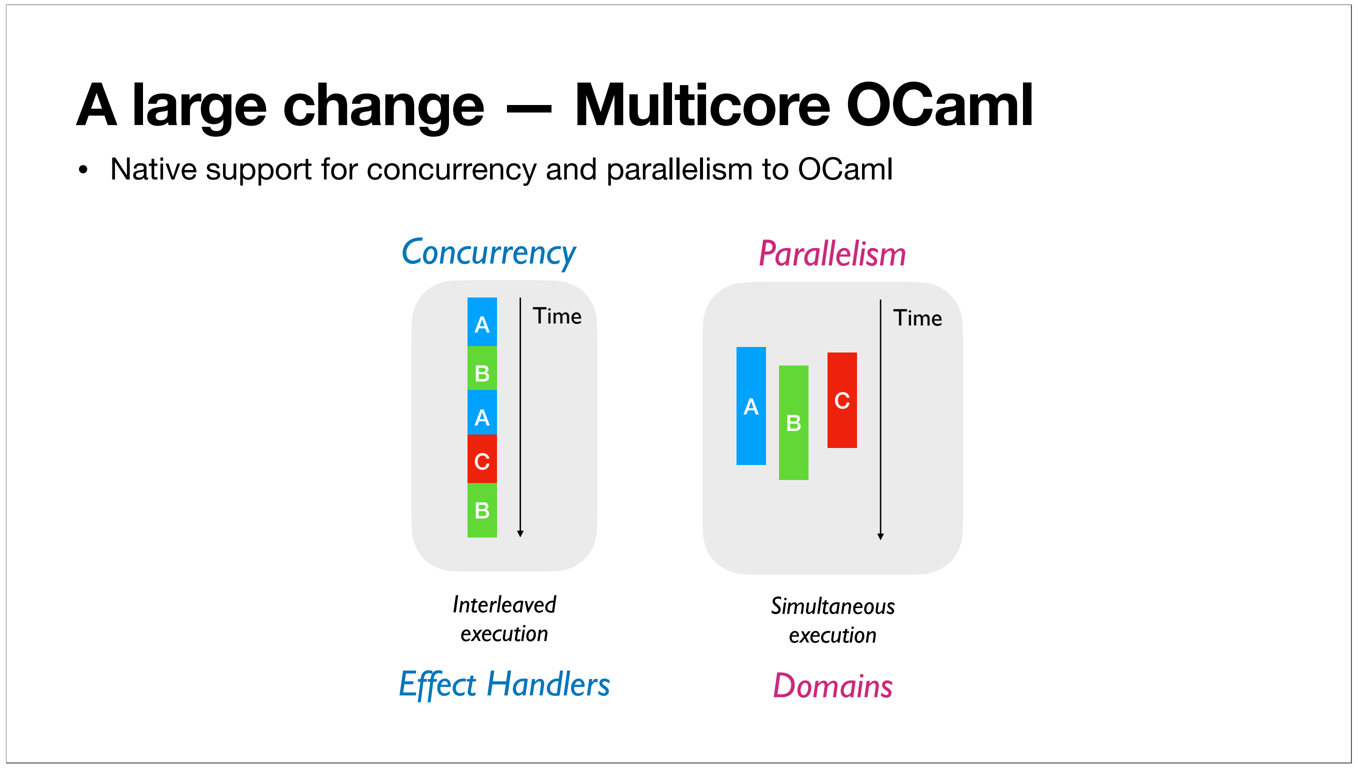
#11882 started its life as an attempt to un-block @c-cube's #11563, the previous proposal for Dynarray in the stdlib, which used an 'unboxed' representation. The PR discussion had ground to a halt because we disagreed on how which unsafe tricks to use to implement this unboxed approach.

- **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 the next 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.

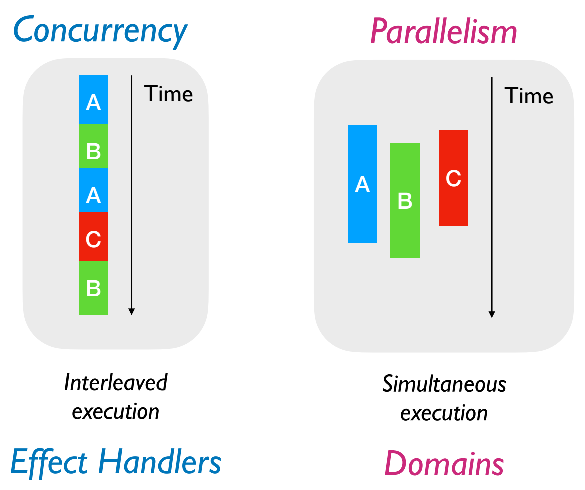
This may seem like an overkill. Is this discussion worth it?

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.



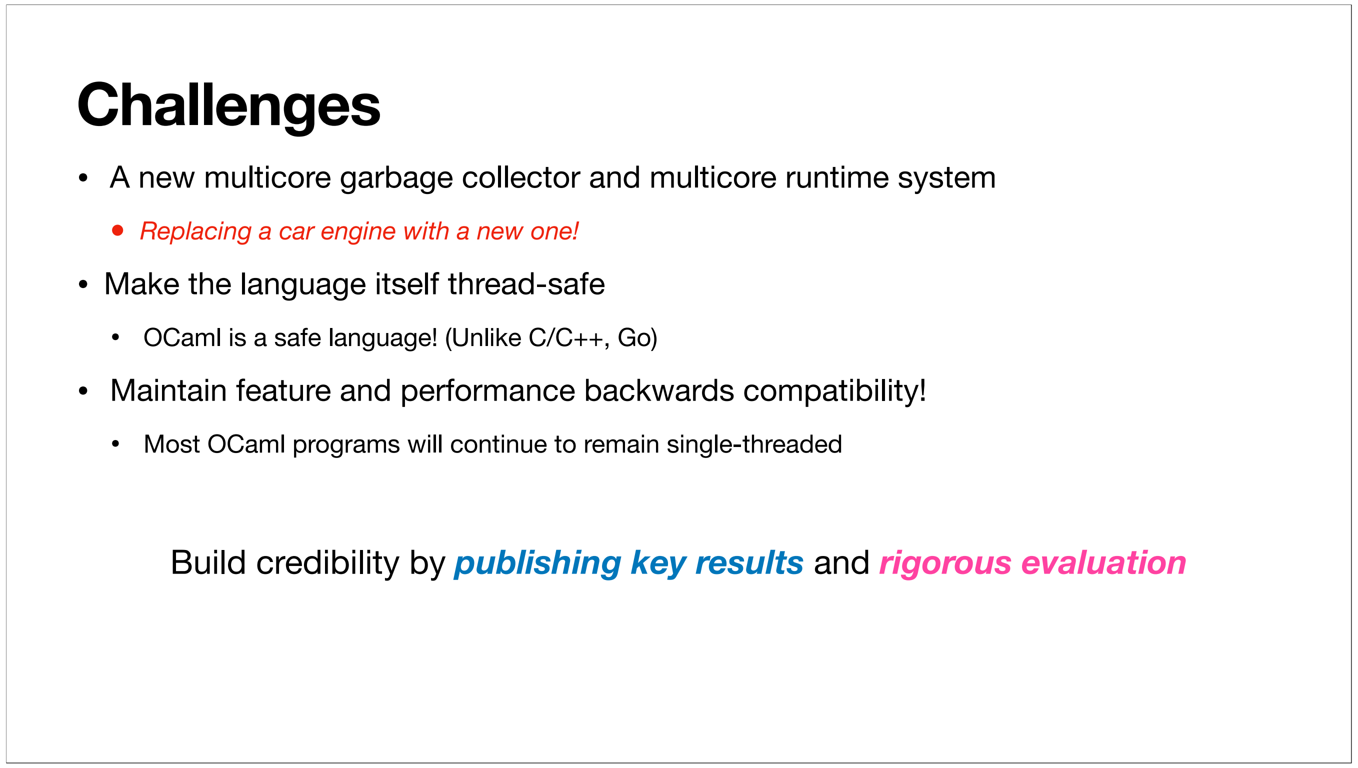
A large change — Multicore OCaml

- Native support for concurrency and parallelism to OCaml



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.

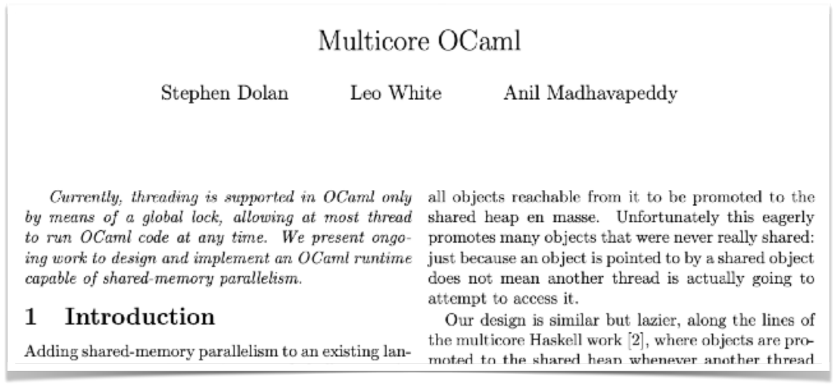
First, Multicore OCaml needed a new multicore capable garbage collector and a multicore runtime system. A good analogy is replacing the engine of a car with the new one.

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.

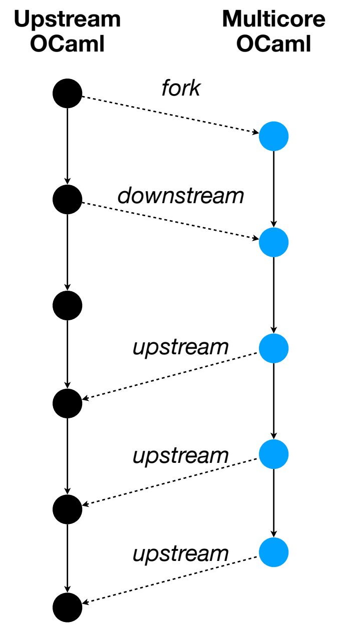
1 Introduction

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

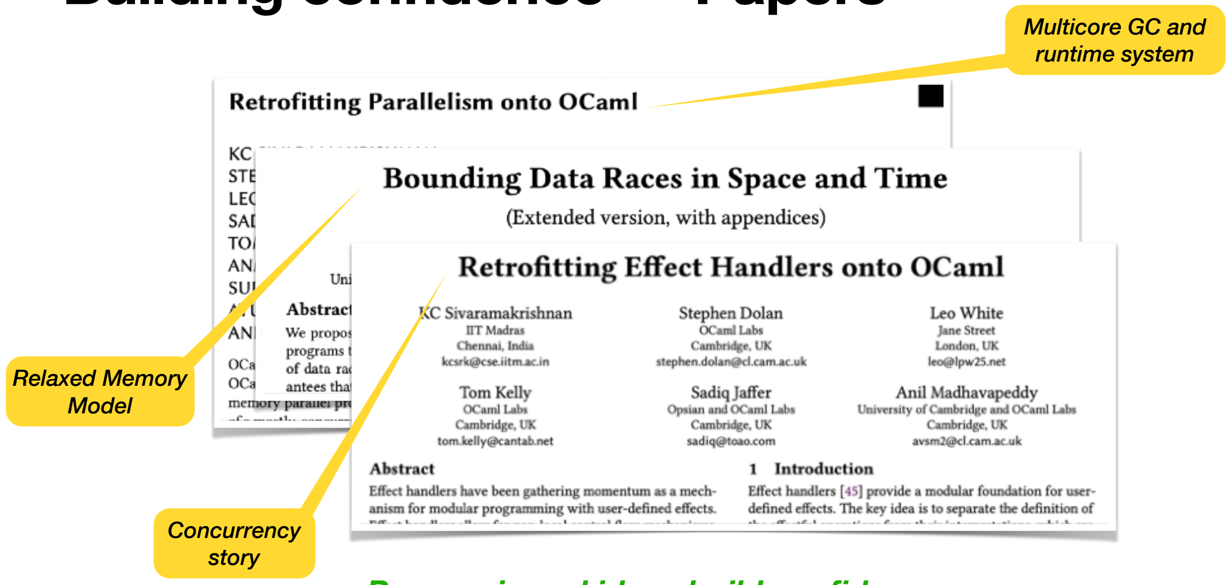
OCaml Workshop *2014*



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


Retrofitting Parallelism onto OCaml

Bounding Data Races in Space and Time

(Extended version, with appendices)

Retrofitting Effect Handlers onto OCaml

KC Sivaramakrishnan, IIT Madras, Chennai, India, kcsrk@cse.iitm.ac.in
Stephen Dolan, OCaml Labs, Cambridge, UK, stephen.dolan@cl.cam.ac.uk
Leo White, Jane Street, London, UK, leo@lpw25.net
Tom Kelly, OCaml Labs, Cambridge, UK, tom.kelly@cantab.net
Sadiq Jaffer, Opsian and OCaml Labs, Cambridge, UK, sadiq@toao.com
Anil Madhavapeddy, University of Cambridge and OCaml Labs, Cambridge, UK, avsm2@cl.cam.ac.uk

Abstract

Effect handlers have been gathering momentum as a mechanism for modular programming with user-defined effects.

1 Introduction

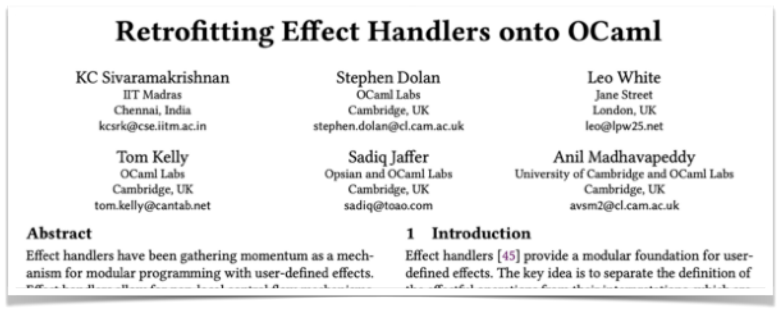
Effect handlers [45] provide a modular foundation for user-defined effects. The key idea is to separate the definition of

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.

Diving deeper — Concurrency



Retrofitting Effect Handlers onto OCaml

KC Sivaramakrishnan
IIT Madras
Chennai, India
kcsrk@cse.iitm.ac.in

Stephen Dolan
OCaml Labs
Cambridge, UK
stephen.dolan@cl.cam.ac.uk

Leo White
Jane Street
London, UK
leo@lpw25.net

Tom Kelly
OCaml Labs
Cambridge, UK
tom.kelly@cantab.net

Sadiq Jaffer
Opsian and OCaml Labs
Cambridge, UK
sadiq@toao.com

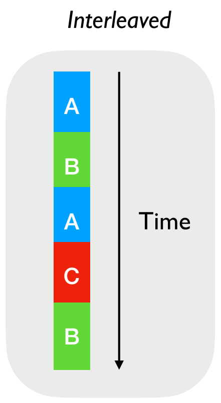
Anil Madhavapeddy
University of Cambridge and OCaml Labs
Cambridge, UK
avsm2@cl.cam.ac.uk

Abstract

Effect handlers have been gathering momentum as a mechanism for modular programming with user-defined effects.

1 Introduction

Effect handlers [45] provide a modular foundation for user-defined effects. The key idea is to separate the definition of



Concurrent Programming

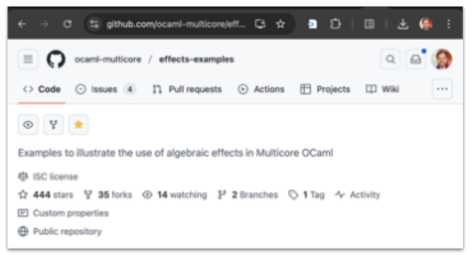
- Computations may be *suspended* and *resumed* later
- Many languages provide concurrent programming mechanisms as *primitives*
  - **async/await** — JavaScript, Python, Rust, C# 5.0, F#, Swift, …
  - **generators** — Python, Javascript, …
  - **coroutines** — C++, Kotlin, Lua, …
  - **futures & promises** — 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 ~= *first-class, restartable exceptions*
  - ✦ Structured programming with *delimited continuations*


github.com/ocaml-multicore/eff...

ocaml-multicore / effects-examples

Code · Issues 4 · Pull requests · Actions · Projects · Wiki

Examples to illustrate the use of algebraic effects in Multicore OCaml

ISC license

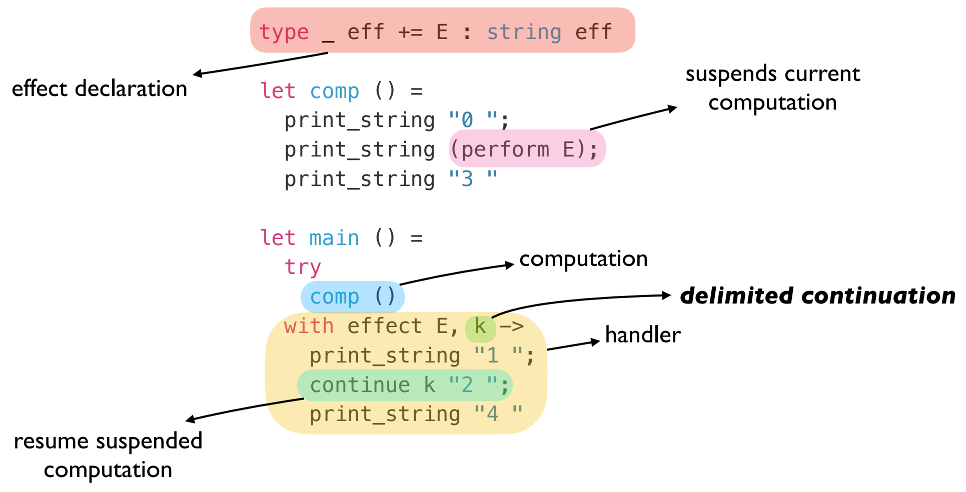
444 stars · 35 forks · 14 watching · 2 Branches · 1 Tag · Activity

Custom properties

Public repository

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

Effect handlers

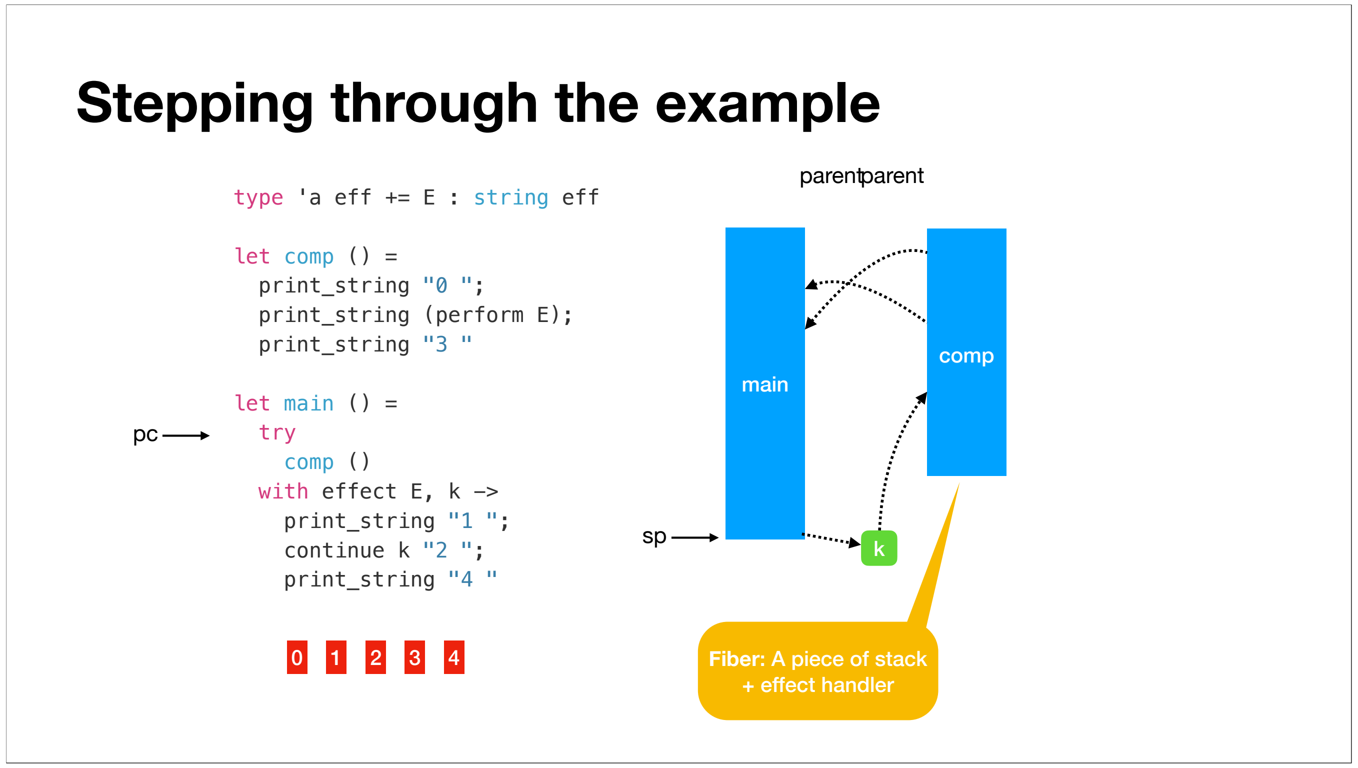


```
type _ eff += E : string eff

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

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

effect declaration
suspends current computation
computation
delimited continuation
handler
resume suspended computation

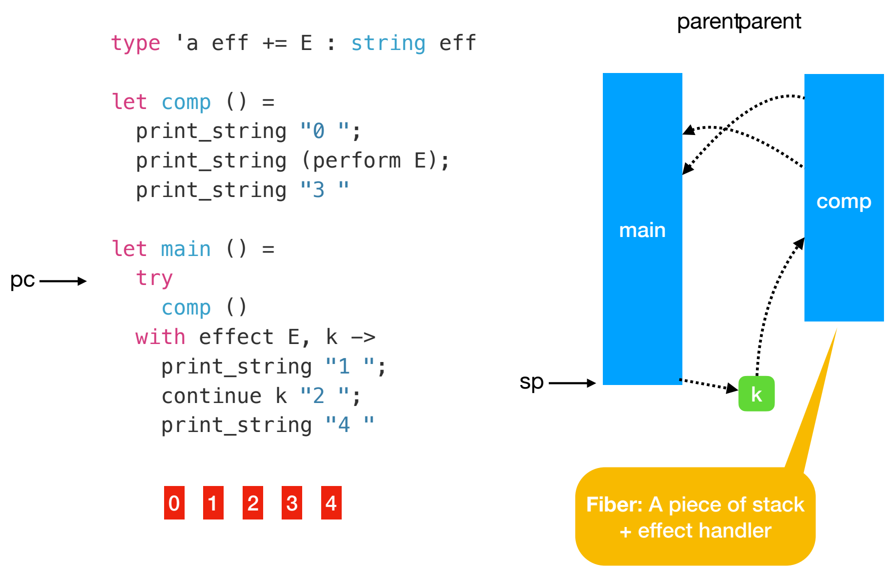


Stepping through the example

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

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

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

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
```

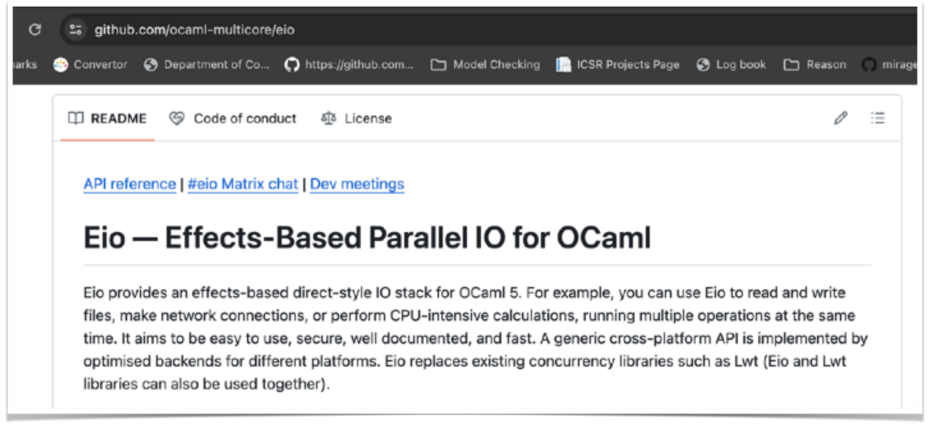
Ability to specialise scheduler unlike GHC Haskell / Go

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)*



API reference | #eio Matrix chat | Dev meetings

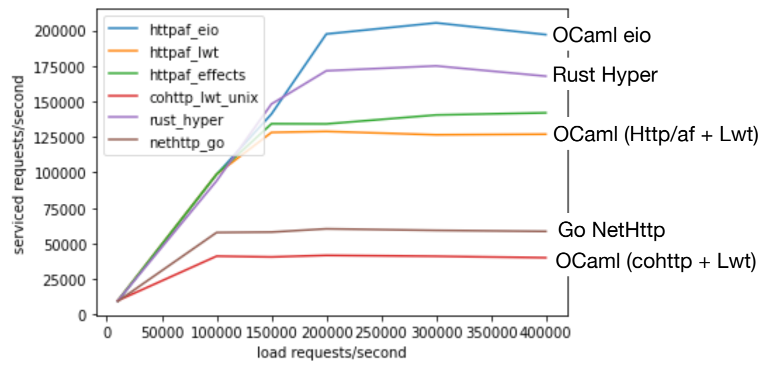
Eio — Effects-Based Parallel IO for OCaml

Eio provides an effects-based direct-style IO stack for OCaml 5. For example, you can use Eio to read and write files, make network connections, or perform CPU-intensive calculations, running multiple operations at the same time. It aims to be easy to use, secure, well documented, and fast. A generic cross-platform API is implemented by optimised backends for different platforms. Eio replaces existing concurrency libraries such as Lwt (Eio and Lwt libraries can also be used together).

`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)*

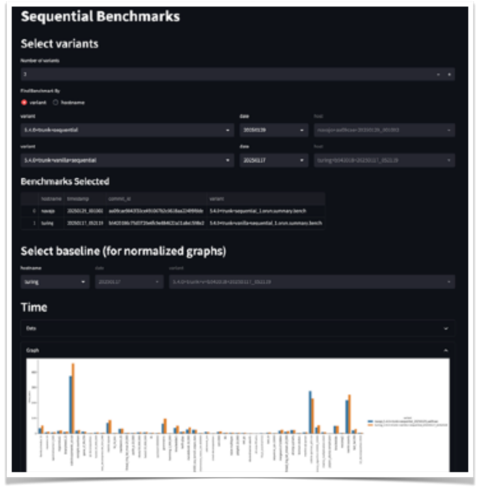


100 open connections, 60 seconds w/ io_uring

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

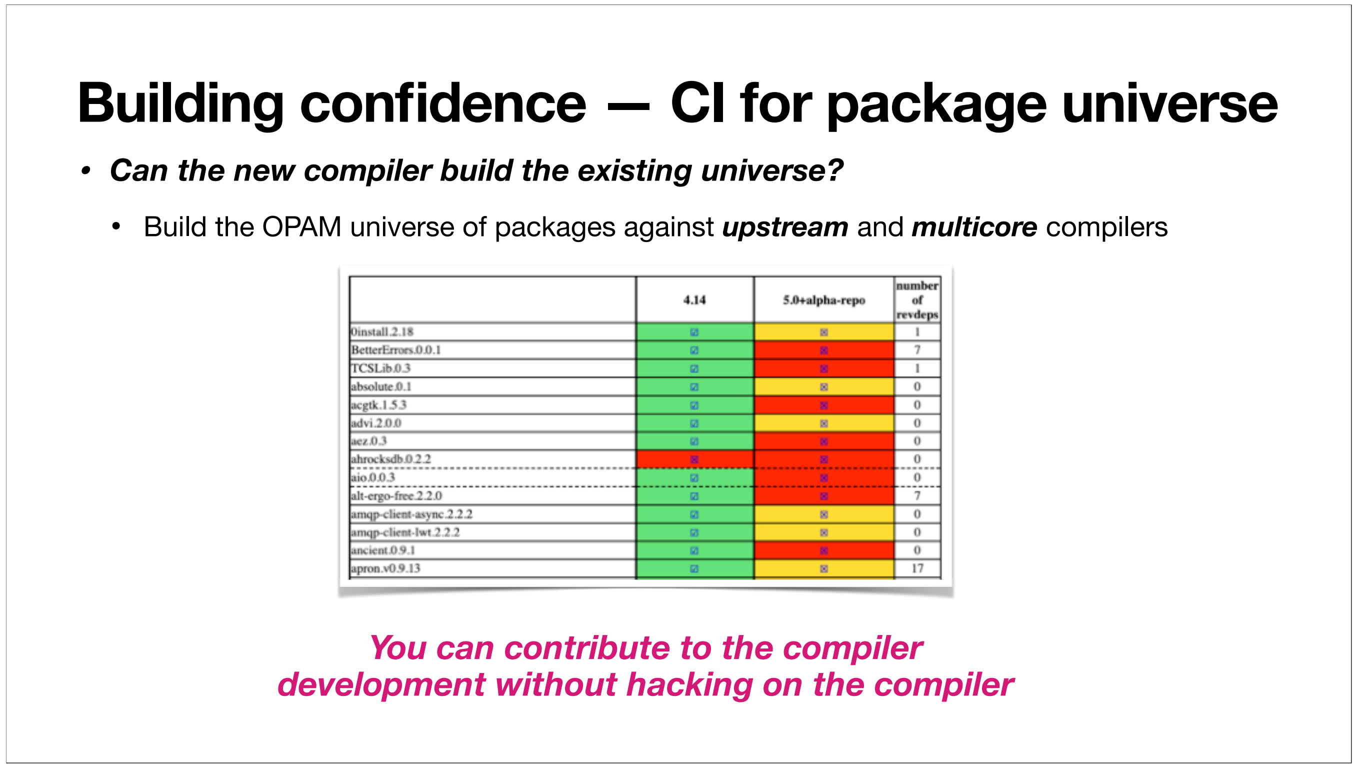
Building confidence — Benchmarking

- *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
advi.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

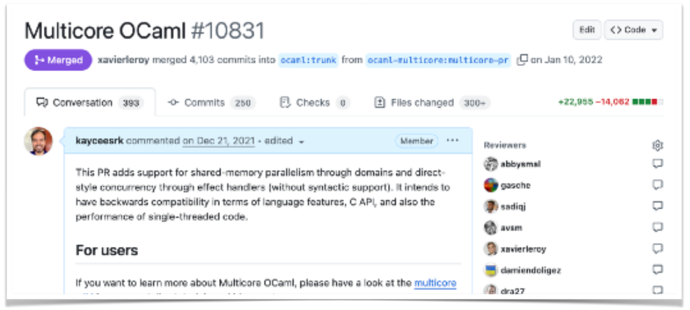
You can contribute to the compiler development without hacking on the compiler

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.

Release and Long Tail

- **Opened —** Dec 2021, **Merged —** Jan 2022
    - *….A few months of iteration to fix design issues and bugs….*



Multicore OCaml #10831

Merged xavierleroy merged 4,103 commits into `ocaml:trunk` from `ocaml-multicore:multicore-pr` on Jan 10, 2022

kayceesrk commented on Dec 21, 2021 • edited Member

This PR adds support for shared-memory parallelism through domains and direct-style concurrency through effect handlers (without syntactic support). It intends to have backwards compatibility in terms of language features, C API, and also the performance of single-threaded code.

For users

If you want to learn more about Multicore OCaml, please have a look at the multicore

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.

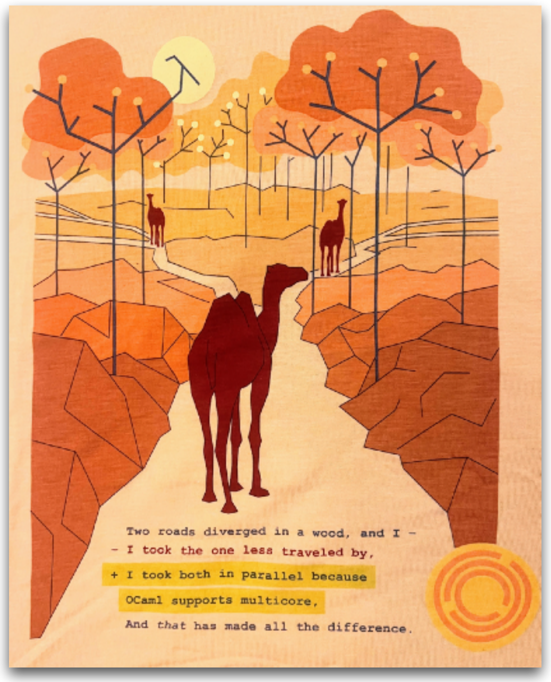
In January 2022, the multicore PR was merged upstream! Even after this, we continued working for a couple of months to fix outstanding issues and bugs.

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.

Release and Long Tail

- **Opened —** Dec 2021, **Merged —** Jan 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 — Jan 2025
  - 5.4 — Sep 2025

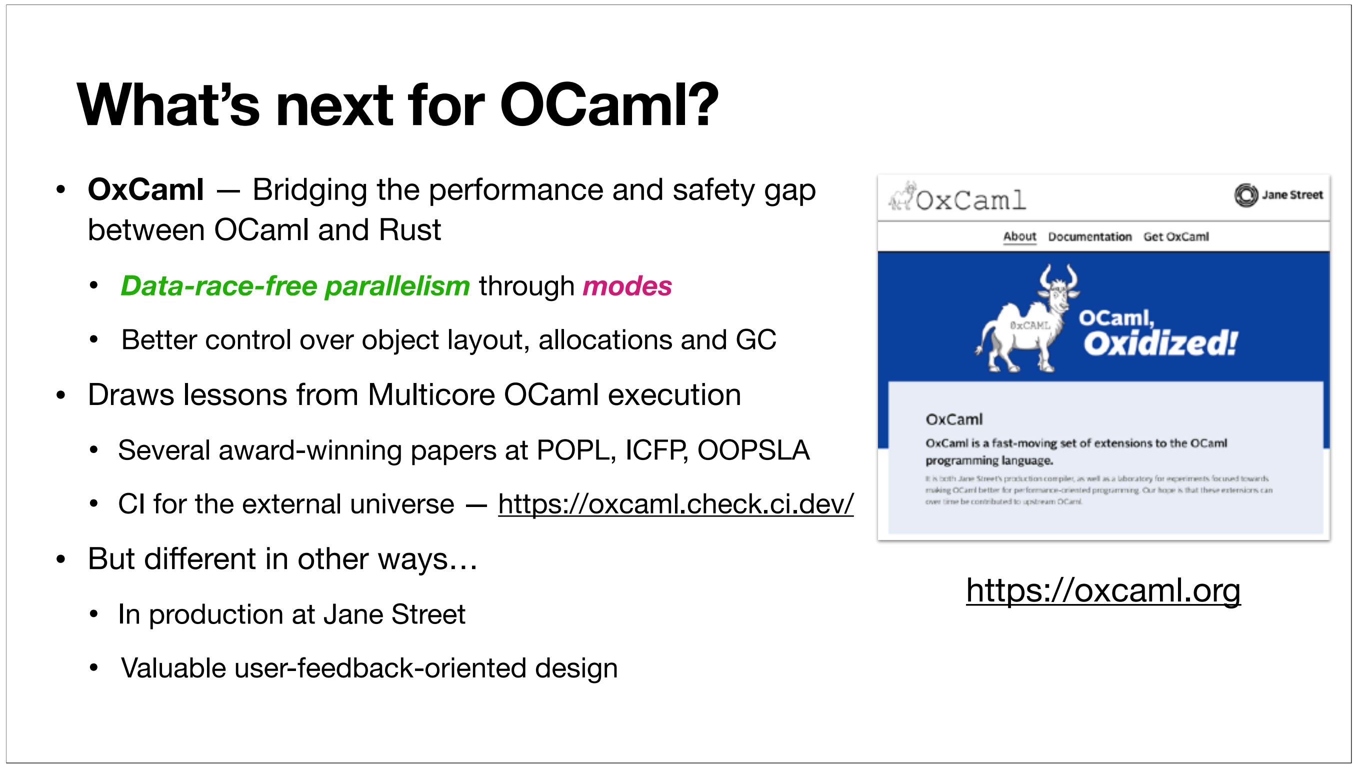

Two roads diverged in a wood, and I –
– I took the one less traveled by,
+ I took both in parallel because
OCaml supports multicore,
And *that* has made all the difference.

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?

- **OxCaml** — Bridging the performance and safety gap between OCaml and Rust
  - ***Data-race-free parallelism*** through ***modes***
  - Better control over object layout, allocations and GC
- Draws lessons from Multicore OCaml execution
  - Several award-winning papers at POPL, ICFP, OOPSLA
  - CI for the external universe — https://oxcaml.check.ci.dev/
- But different in other ways…
  - In production at Jane Street
  - Valuable user-feedback-oriented design

OxCaml

OxCaml is a fast-moving set of extensions to the OCaml programming language.

It is both Jane Street's production compiler, as well as a laboratory for experiments focused towards making OCaml better for performance-oriented programming. Our hope is that these extensions can over time be contributed to upstream OCaml.

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.

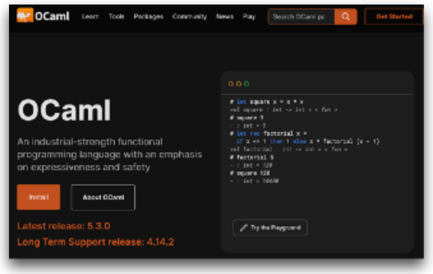
While it draws lessons from Multicore OCaml execution, there are a few things that are different. OxCaml is in production at Jane Street, which provides valuable feedback from real users. The features are much more experimental and the cost of change is far less compared to releasing the code for all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

OCaml

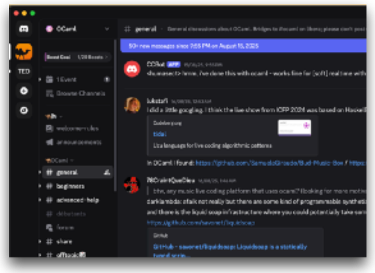
An industrial-strength functional programming language with an emphasis on expressiveness and safety

Install · About OCaml

Latest release: 5.3.0

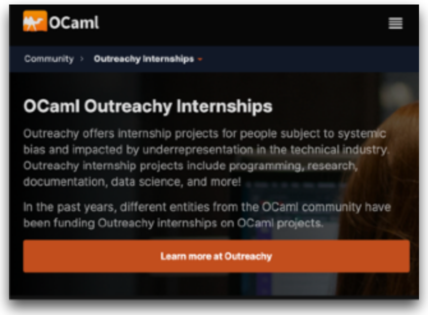
Long Term Support release: 4.14.2

```
# let square x = x * x
val square : int -> int = <fun>
# square 3
- : int = 9
# let rec factorial x =
  if x <= 1 then 1 else x * factorial (x - 1)
val factorial : int -> int = <fun>
# factorial 5
- : int = 120
# square 120
- : int = 14400
```



OCaml Discord

ocaml.org/outreachy

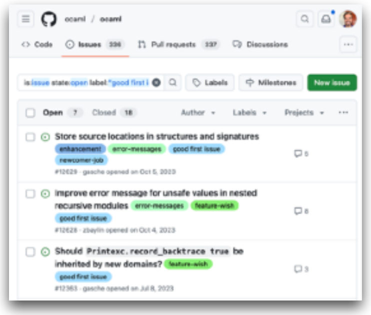


Community › Outreachy Internships

OCaml Outreachy Internships

Outreachy offers internship projects for people subject to systemic bias and impacted by underrepresentation in the technical industry. Outreachy internship projects include programming, research, documentation, data science, and more!

In the past years, different entities from the OCaml community have been funding Outreachy internships on OCaml projects.



ocaml / ocaml

Code · Issues 336 · Pull requests 337 · Discussions

is:issue state:open label:"good first i

Open 7 · Closed 18

- Store source locations in structures and signatures — enhancement, error-messages, good first issue, newcomer-job — #12629 · gasche opened on Oct 5, 2023 — 5
- Improve error message for unsafe values in nested recursive modules — error-messages, feature-wish, good first issue — #12628 · zbaylin opened on Oct 4, 2023 — 6
- Should `Printexc.record_backtrace true` be inherited by new domains? — feature-wish, good first issue — #12363 · gasche opened on Jul 8, 2023 — 3

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!