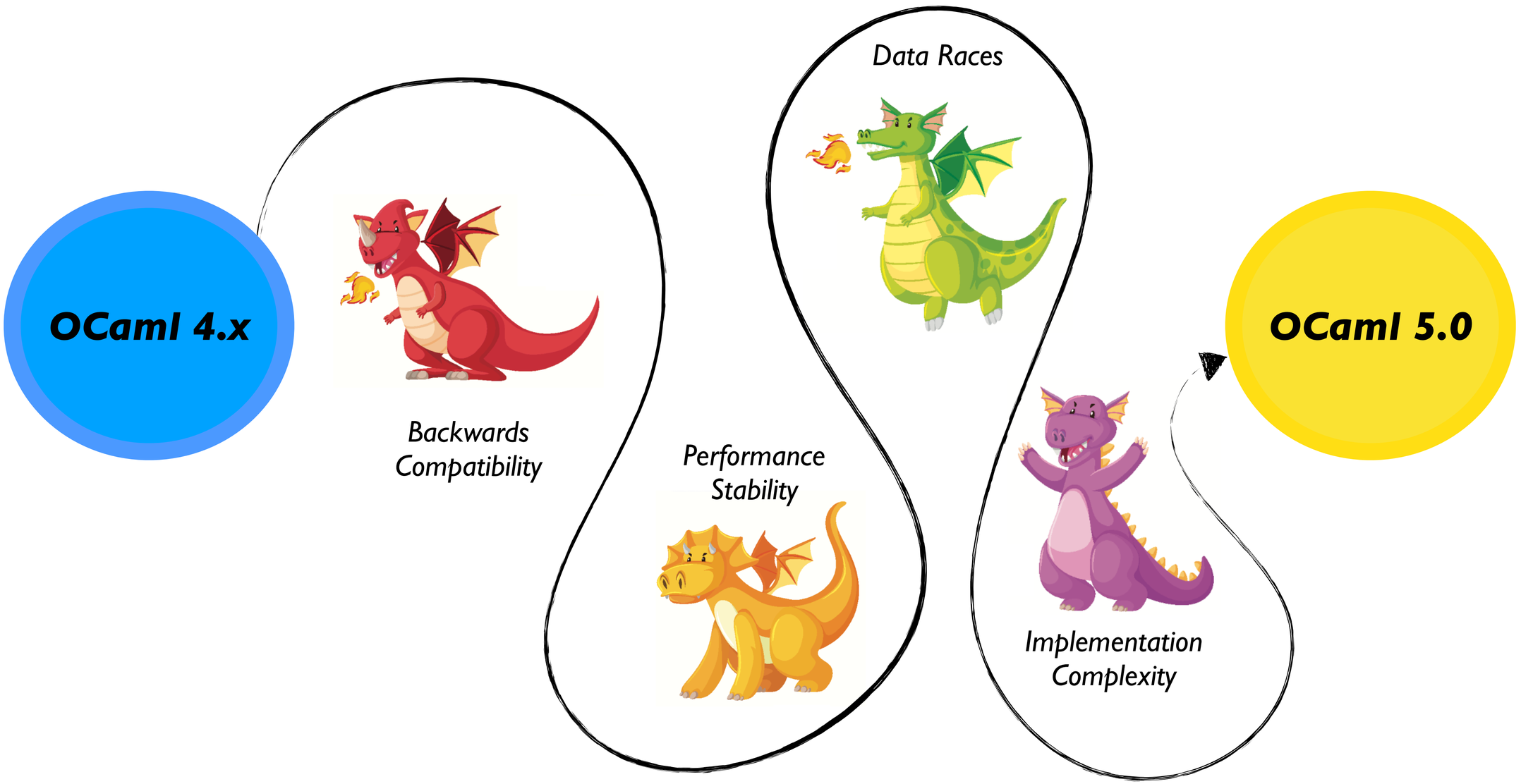
OCaml 5.0

"KC" Sivaramakrishnan

IIT MADRAS

Tarides

ICFP Keynote



This talk…

OCaml 5.0

What's in the can?

FAQs

Merge Process

Moving to OCaml 5.0

Concurrency and Parallelism

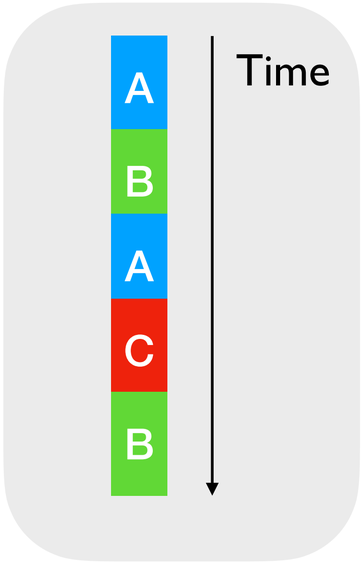
Concurrency and Parallelism

Concurrency

Parallelism

Concurrency and Parallelism

Concurrency



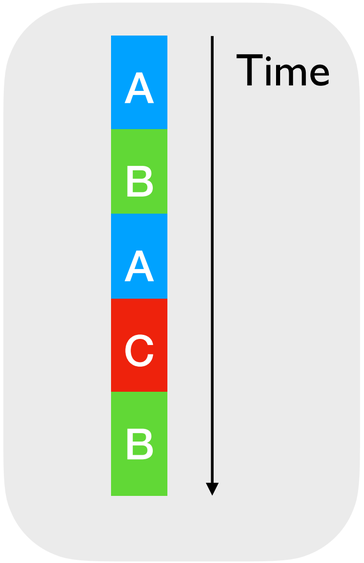
Overlapped execution

Effect Handlers

Parallelism

Concurrency and Parallelism

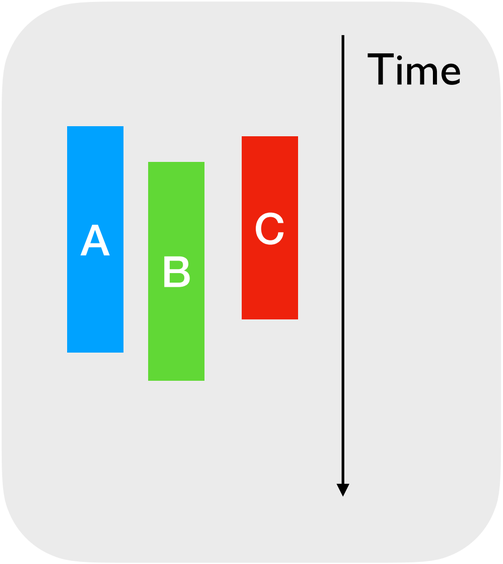
Concurrency



Overlapped execution

Effect Handlers

Parallelism

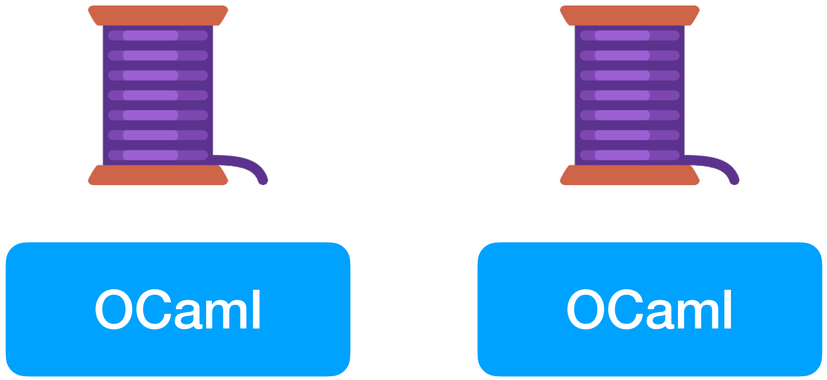


Simultaneous execution

Domains

Domains

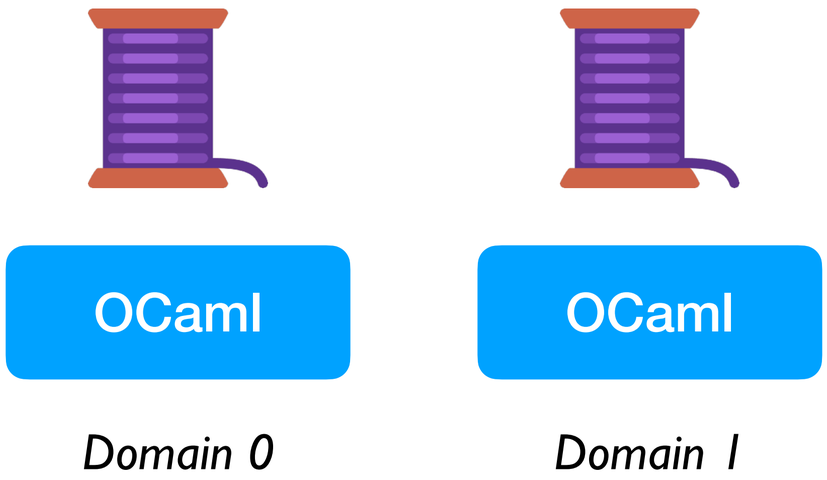
- Units of parallelism
- *Heavy-weight* entities
  - ✦ Recommended to have 1 domain per core



Domain 0

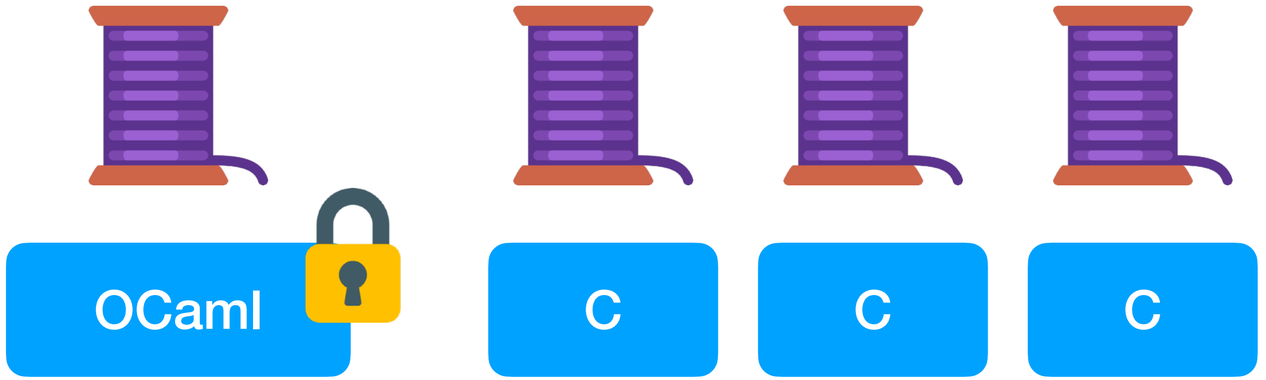
Domain 1

Domains

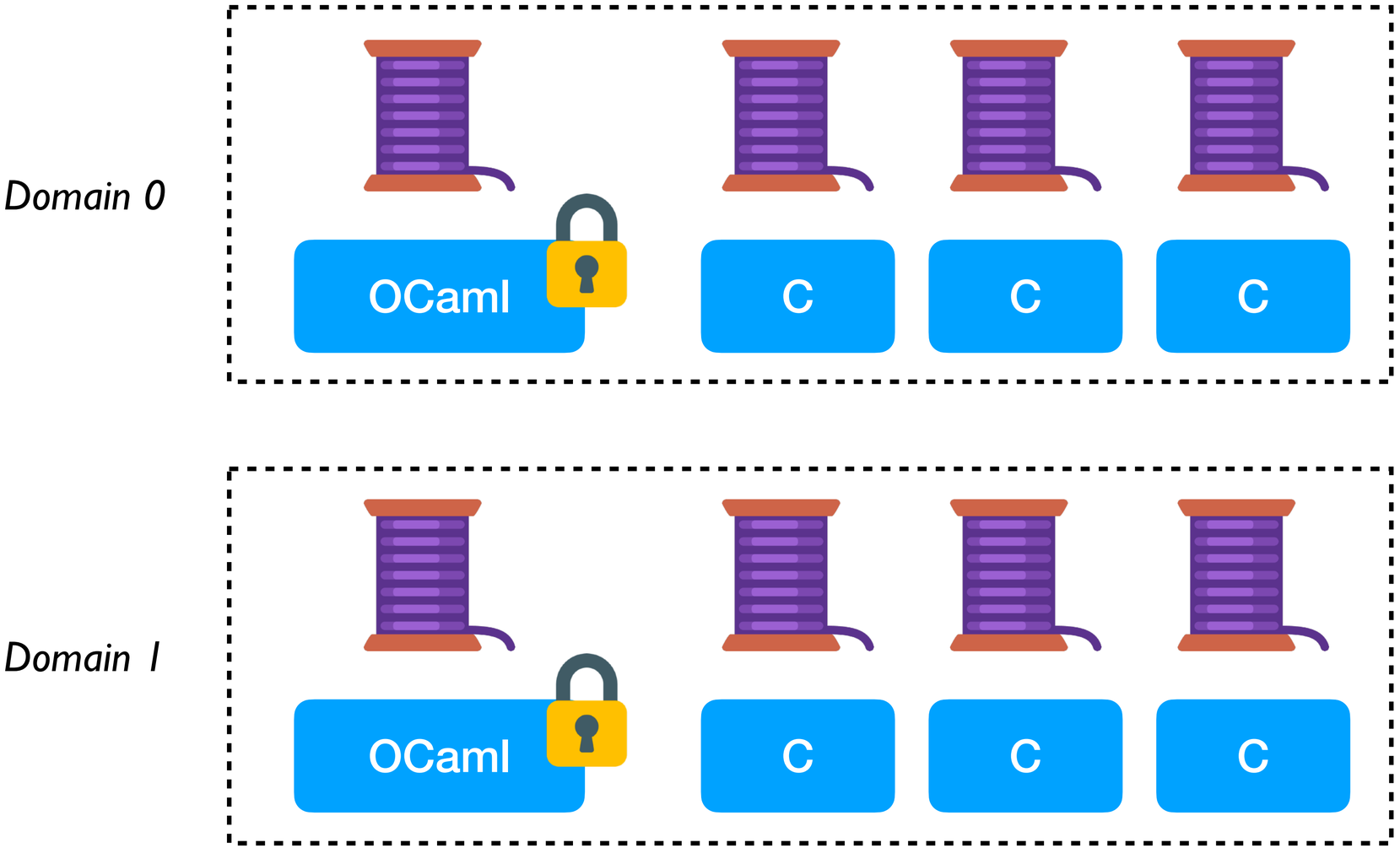


- Units of parallelism
- *Heavy-weight* entities
  - ✦ Recommended to have 1 domain per core
- API
  - ✦ *Create and destroy* — Spawn and Join
  - ✦ *Blocking synchronisation* — Mutex, Condition and Semaphore
  - ✦ *Non-blocking synchronisation* — Atomic
  - ✦ Domain-local state

Threads



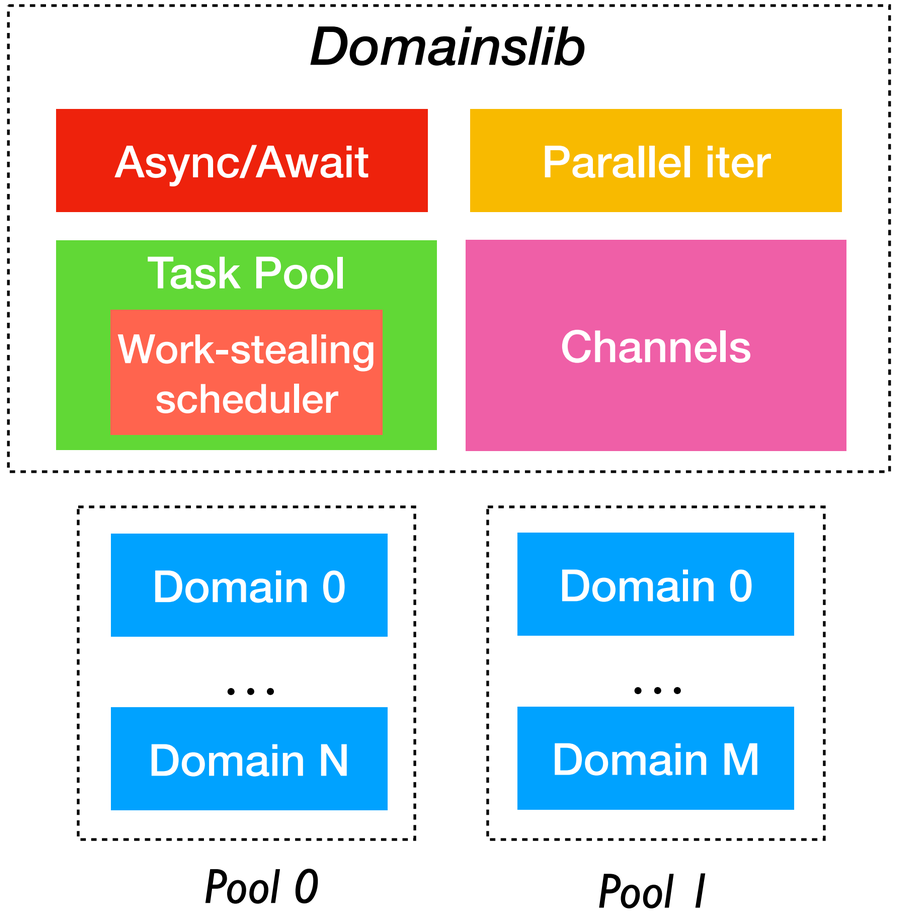
Domains with Threads



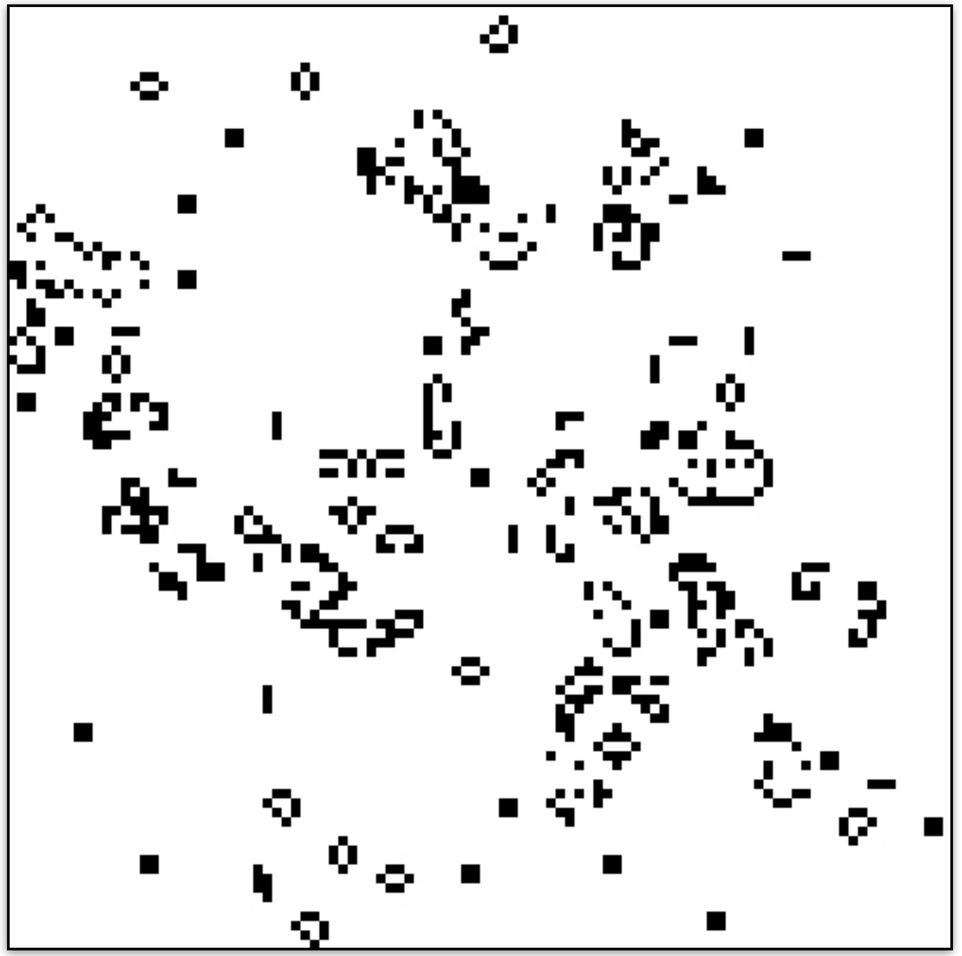
Blocking and non-blocking synchronisation works uniformly across threads and domains

Domainslib

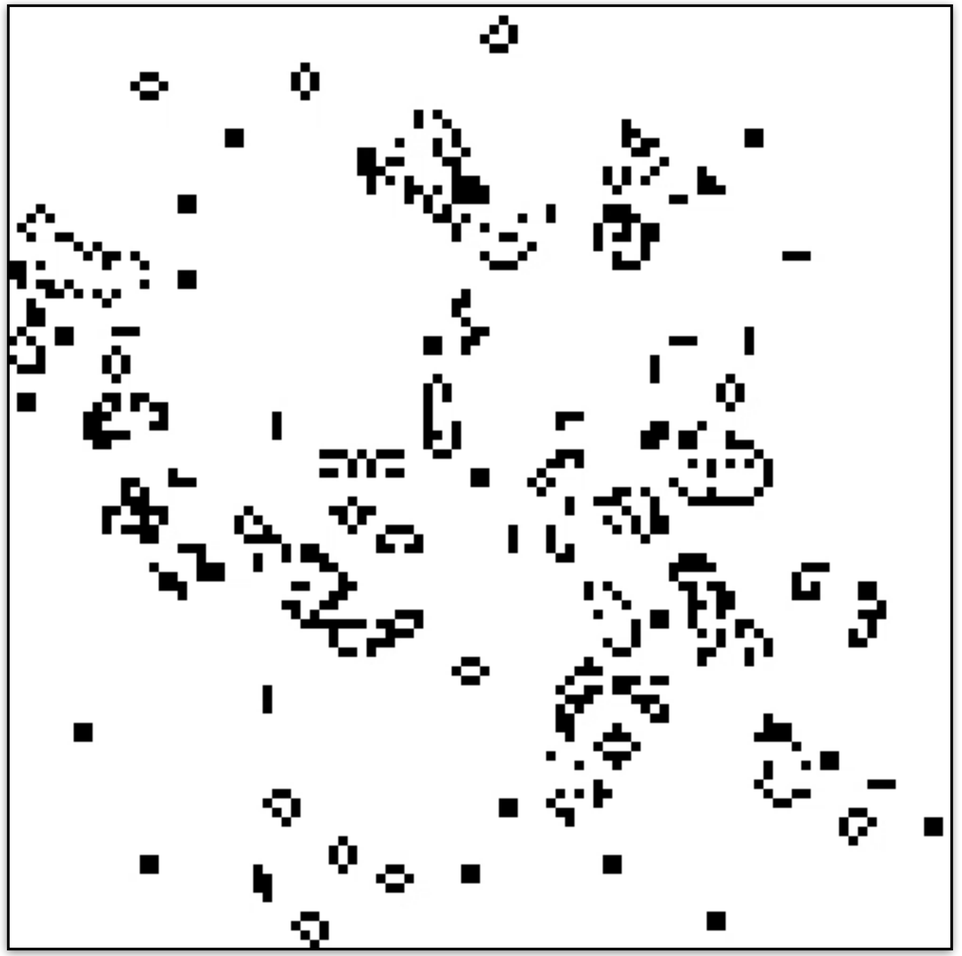
- A library for nested-parallel programming (OpenMP, Cilk, NESL,…)



Conway's Game of Life



Conway's Game of Life



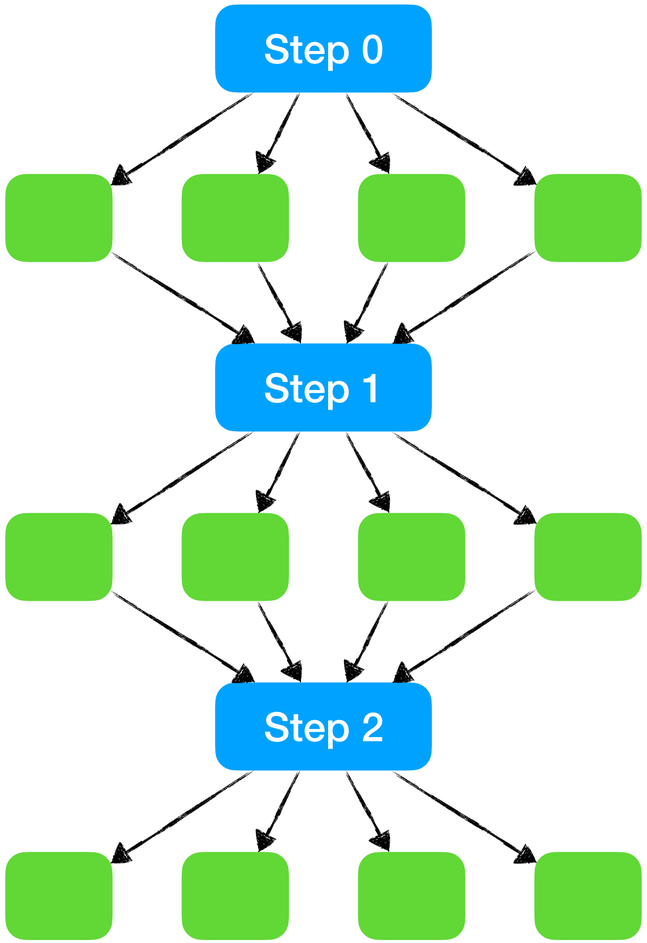
Conway's Game of Life

```
let next () =
  ...
  for x = 0 to board_size - 1 do
    for y = 0 to board_size - 1 do
      next_board.(x).(y) <- next_cell cur_board x y
    done
  done;
  ...
```

Conway's Game of Life

```
let next () =
  ...
  for x = 0 to board_size - 1 do
    for y = 0 to board_size - 1 do
      next_board.(x).(y) <- next_cell cur_board x y
    done
  done;
  ...
```

```
let next () =
  ...
  T.parallel_for pool ~start:0 ~finish:(board_size - 1)
    ~body:(fun x ->
       for y = 0 to board_size - 1 do
         next_board.(x).(y) <- next_cell cur_board x y
       done);
  ...
```

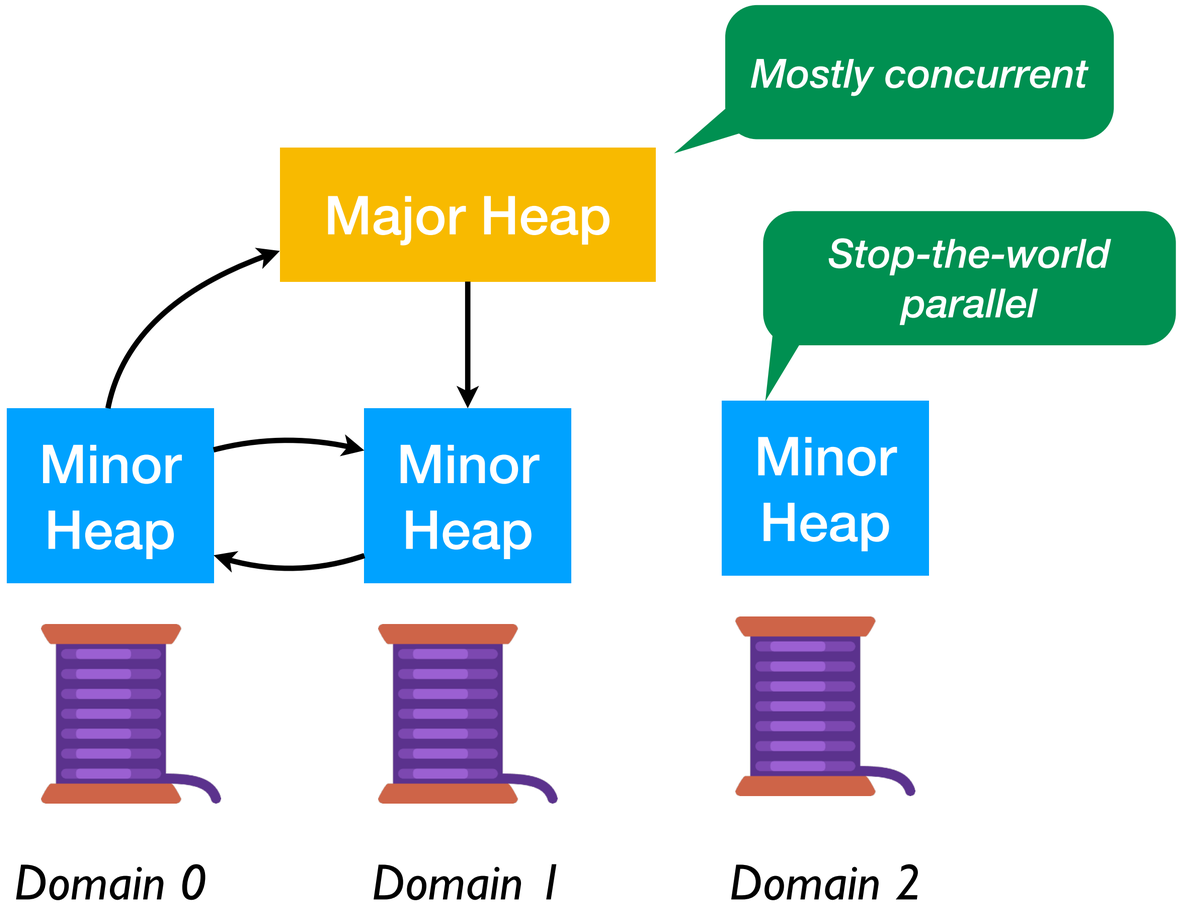


Performance: Game of Life

Board size = 1024, Iterations = 512

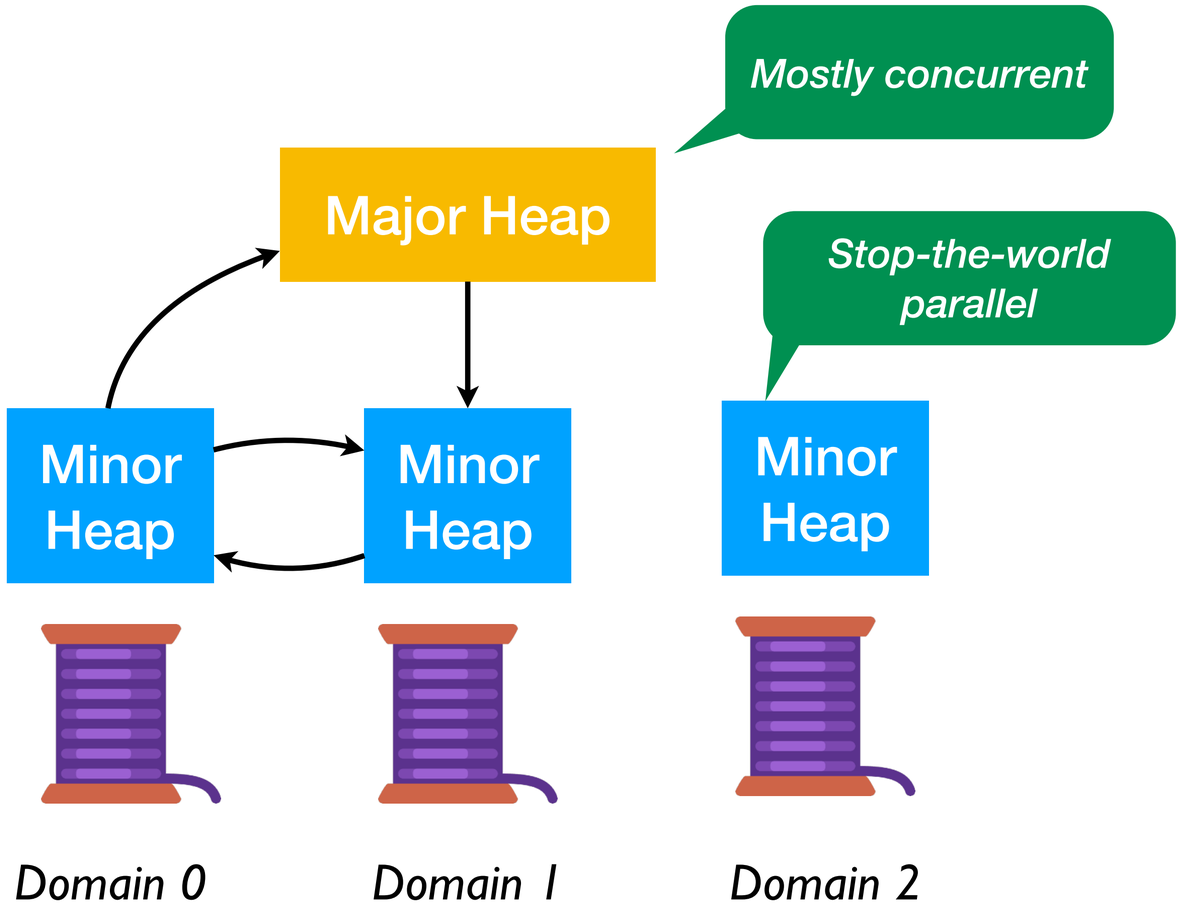
Cores	Time (Seconds)	Vs Serial
1	24.326	1
2	12.290	1.980
4	6.260	3.890
8	3.238	7.51
16	1.726	14.09
24	1.212	**20.07**

Allocation and Collection



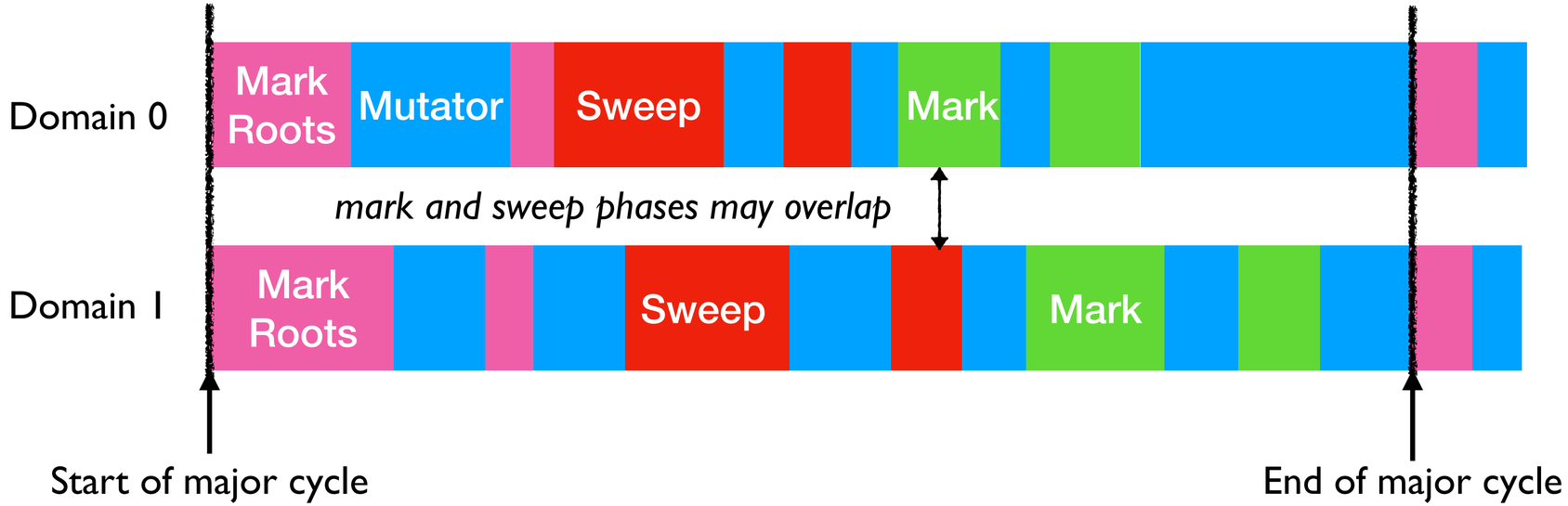
- Minor heap allocations require *no synchronization*
- Major heap allocator is
  - **Small:** Thread-local, size-segmented free list
  - **Large:** malloc

Allocation and Collection

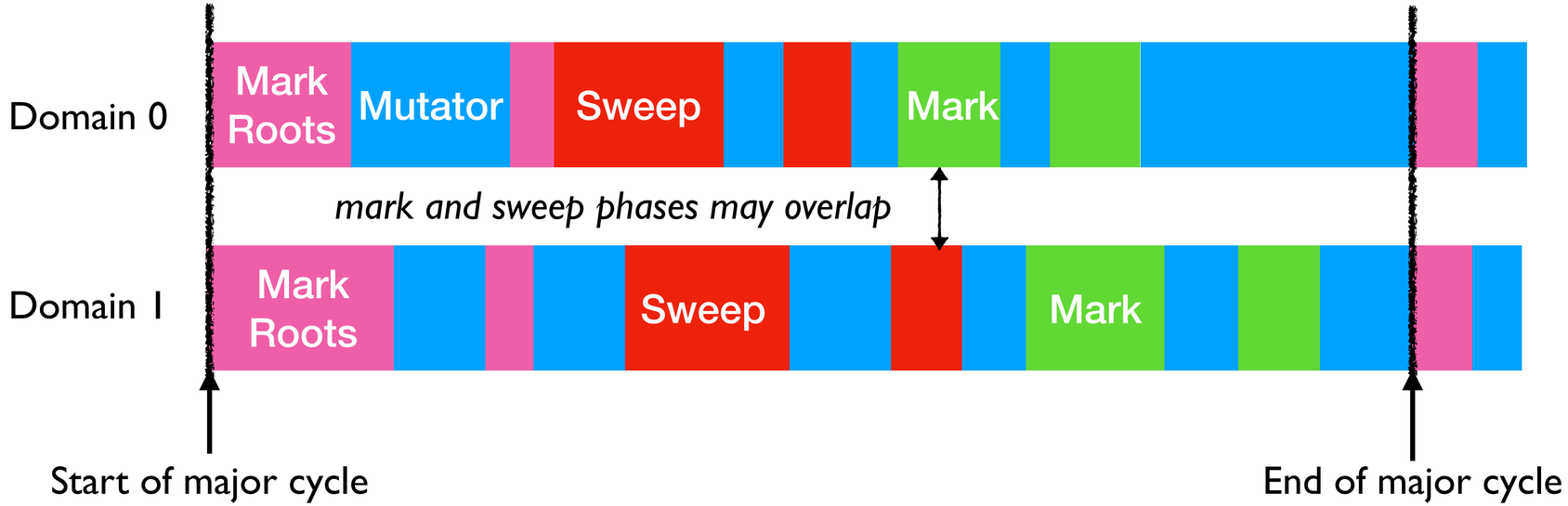


- Minor heap allocations require *no synchronization*
- Major heap allocator is
  - **Small:** Thread-local, size-segmented free list
  - **Large:** malloc
- Goal is to match *best-fit* for sequential programs
  - If we're slower than best-fit, then it is a *performance regression*

Concurrent GC

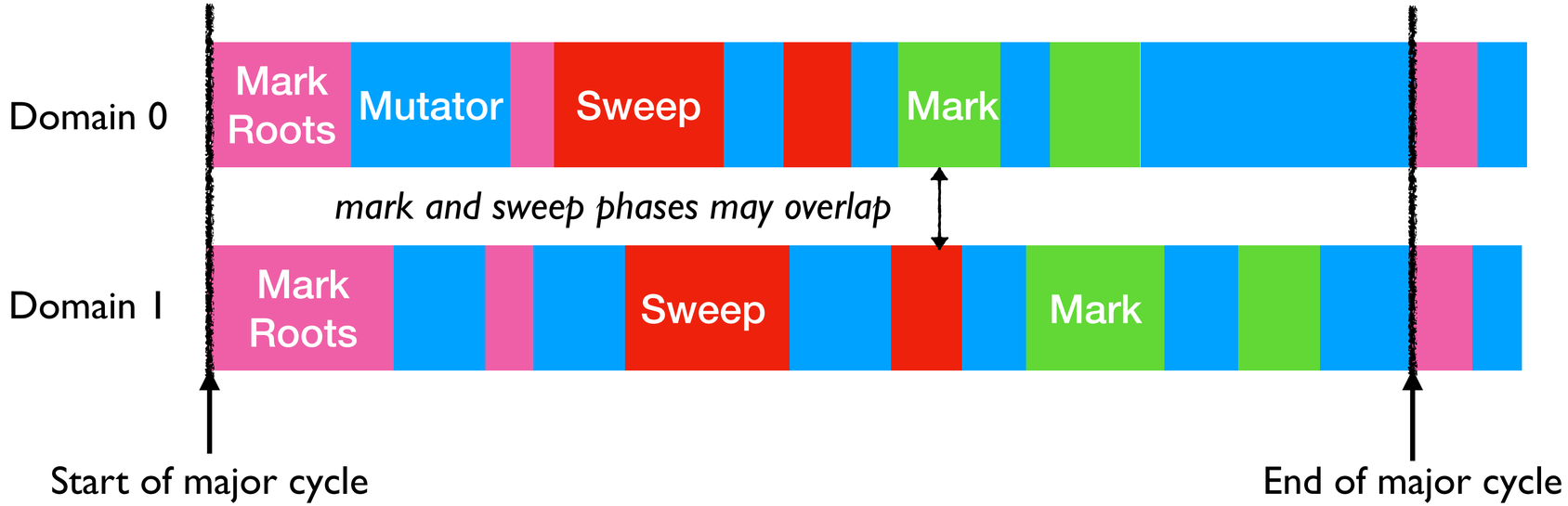


Concurrent GC



- Stop-the-world parallel *minor* GC + non-moving *major* GC
  - ✦ *Objects don't move while the mutator is running!*

Concurrent GC



- Stop-the-world parallel *minor* GC + non-moving *major* GC
  - ✦ *Objects don't move while the mutator is running!*
- **No additional rules for the C FFI in OCaml 5.0**
  - ✦ Same rules as OCaml 4.x hold even for parallel programs!

OCaml memory model

- Simple (*comprehensible!*) operational memory model
  - ✦ Only atomic and non-atomic locations
  - ✦ DRF-SC
  - ✦ No “out of thin air” values
  - ✦ Squeeze at most perf ⇒ write that module in C, C++ or Rust.

OCaml memory model

- Simple (*comprehensible!*) operational memory model
  - ✦ Only atomic and non-atomic locations
  - ✦ DRF-SC
  - ✦ No “out of thin air” values
  - ✦ Squeeze at most perf ⇒ write that module in C, C++ or Rust.
- **Key innovation:** *Local data race freedom*
  - ✦ Permits compositional reasoning

OCaml memory model

- Simple (*comprehensible!*) operational memory model
  - ✦ Only atomic and non-atomic locations
  - ✦ DRF-SC
  - ✦ No “out of thin air” values
  - ✦ Squeeze at most perf ⇒ write that module in C, C++ or Rust.
- **Key innovation:** *Local data race freedom*
  - ✦ Permits compositional reasoning
- Performance impact
  - ✦ Free on x86 and < 1% on ARM

OCaml memory model

- Simple (*comprehensible!*) operational memory model

Bounding Data Races in Space and Time

(Extended version, with appendices)

PLDI '18

Stephen Dolan
University of Cambridge, UK

KC Sivaramakrishnan
University of Cambridge, UK

Anil Madhavapeddy
University of Cambridge, UK

Abstract

We propose a new semantics for shared-memory parallel programs that gives strong guarantees even in the presence of data races. Our *local data race freedom* property guarantees that all data-race-free portions of programs exhibit

The primary reasoning tools provided to programmers by these models are the *data-race-freedom (DRF) theorems*. Programmers are required to mark as *atomic* all variables used for synchronisation between threads, and to avoid *data races*, which are concurrent accesses (except concurrent reads) to

- Performance impact
  - ✦ Free on x86 and < 1% on ARM

OCaml memory model

- PLDI '18 paper only formalised compilation to *hardware memory models*
  - ✦ Omitted object initialisation

OCaml memory model

- PLDI '18 paper only formalised compilation to *hardware memory models*
  - ✦ Omitted object initialisation
- OCaml 5.0 extended the work to cover
  - ✦ Object initialisation
  - ✦ Compilation to C11 memory model

OCaml memory model

- PLDI '18 paper only formalised compilation to *hardware memory models*
  - ✦ Omitted object initialisation
- OCaml 5.0 extended the work to cover
  - ✦ Object initialisation
  - ✦ Compilation to C11 memory model
- C FFI has been made stronger (by making the access `volatile`)

```
#define Field(x, i) (((volatile value *)(x)) [I])

void caml_modify (volatile value *, value);

void caml_initialize (volatile value *, value);
```

  - ✦ Assumes Linux Kernel Memory Model (LKMM)
  - ✦ Does not break code

OCaml memory model

- ***C FFI also respects LDRF!***

OCaml memory model

- ***C FFI also respects LDRF!***

```
let msg = ref 0
let flag = Atomic.make false

let t1 =
  msg := 1;
  Atomic.set flag true

let t2 =
  let rf = Atomic.get flag in
  let rm = !msg in
  assert (not (rf = true && rm = 0))
```

OCaml memory model

- ***C FFI also respects LDRF!***

```
let msg = ref 0
let flag = Atomic.make false

let t1 =
  msg := 1;
  Atomic.set flag true

let t2 =
  let rf = Atomic.get flag in
  let rm = !msg in
  assert (not (rf = true && rm = 0))
```

```
/* t1 implemented in C */
void t1 (value msg, value flag) {
  caml_modify (&Field(msg,0), Val_int(1));
  caml_atomic_exchange (flag, Val_true);
}
```

ThreadSanitizer

```
1  type t = { mutable x : int }
2
3  let v = { x = 0 }
4
5  let () =
6    let t1 = Domain.spawn (fun () -> v.x <- 10) in
7    let t2 = Domain.spawn (fun () -> Unix.sleep v.x) in
8    Domain.join t1;
9    Domain.join t2
```

ThreadSanitizer

```
1  type t = { mutable x : int }
2
3  let v = { x = 0 }
4
5  let () =
6    let t1 = Domain.spawn (fun () -> v.x <- 10) in
7    let t2 = Domain.spawn (fun () -> Unix.sleep v.x) in
8    Domain.join t1;
9    Domain.join t2
```

```
WARNING: ThreadSanitizer: data race (pid=502344)
  Read of size 8 at 0x7fc0b15fe458 by thread T4 (mutexes: write M0):
    #0 camlDune__exe__Simple_race__fun_600 /workspace_root/simple_race.ml:7 (simple_race.exe+0x51e9b1)
    #1 caml_callback ??:? (simple_race.exe+0x5777f0)
    #2 domain_thread_func domain.c:? (simple_race.exe+0x57b8fc)

  Previous write of size 8 at 0x7fc0b15fe458 by thread T1 (mutexes: write M1):
    #0 camlDune__exe__Simple_race__fun_596 /workspace_root/simple_race.ml:6 (simple_race.exe+0x51e971)
    #1 caml_callback ??:? (simple_race.exe+0x5777f0)
    #2 domain_thread_func domain.c:? (simple_race.exe+0x57b8fc)
```

Effect handlers

- Structured programming with delimited continuations
- No effect system, no dedicated syntax
- Provides both *deep* and *shallow* handlers

Effect handlers

- Structured programming with delimited continuations
- No effect system, no dedicated syntax
- Provides both *deep* and *shallow* handlers

Example prints “0 1 2 3 4”

```
1  open Effect
2  open Effect.Deep
3
4  type _ Effect.t += E : string t
5
6  let comp () =
7    print_string "0 ";
8    print_string (perform E);
9    print_string "3 "
10
11 let main () =
12   try_with comp ()
13   { effc = fun (type a) (e : a t) ->
14       match e with
15       | E -> Some (fun (k : (a,_) continuation) ->
16           print_string "1 ";
17           continue k "2 ";
18           print_string "4 ")
19       | e -> None }
```

Effect handlers

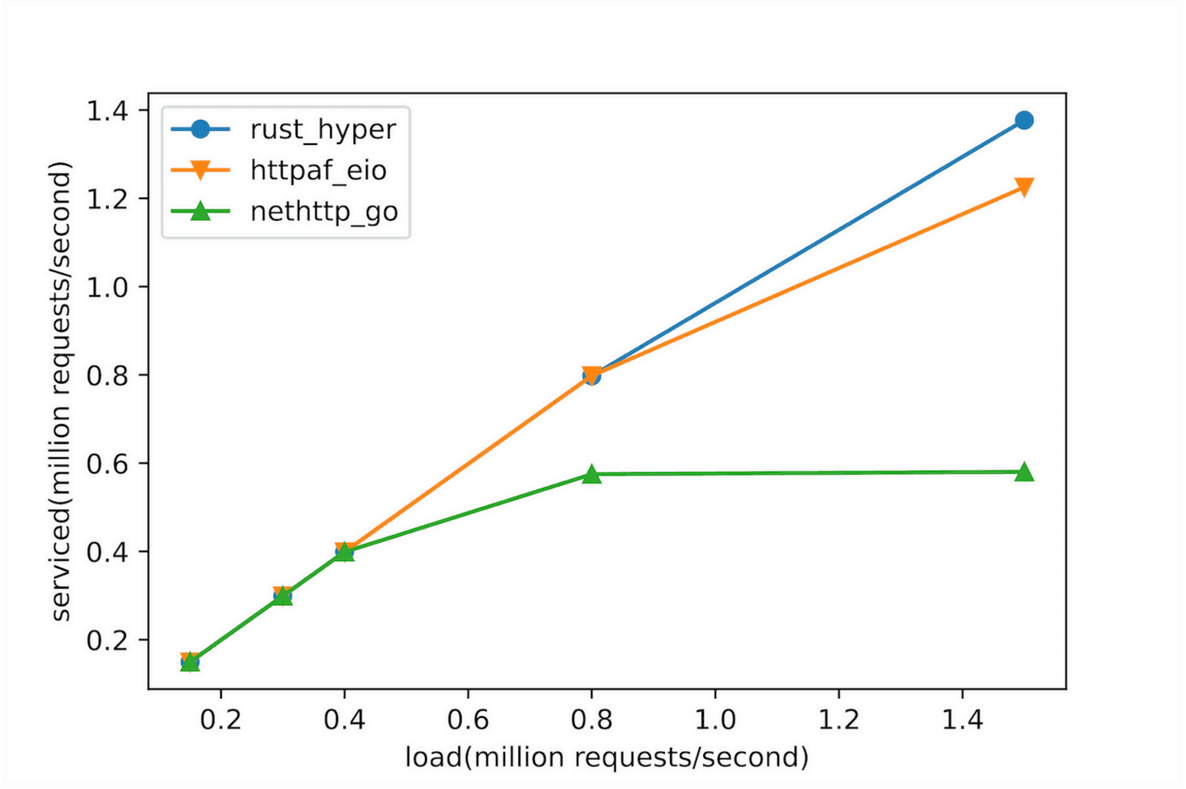
- Structured programming with delimited continuations
- No effect system, no dedicated syntax
- Provides both *deep* and *shallow* handlers

Example prints “0 1 2 3 4”

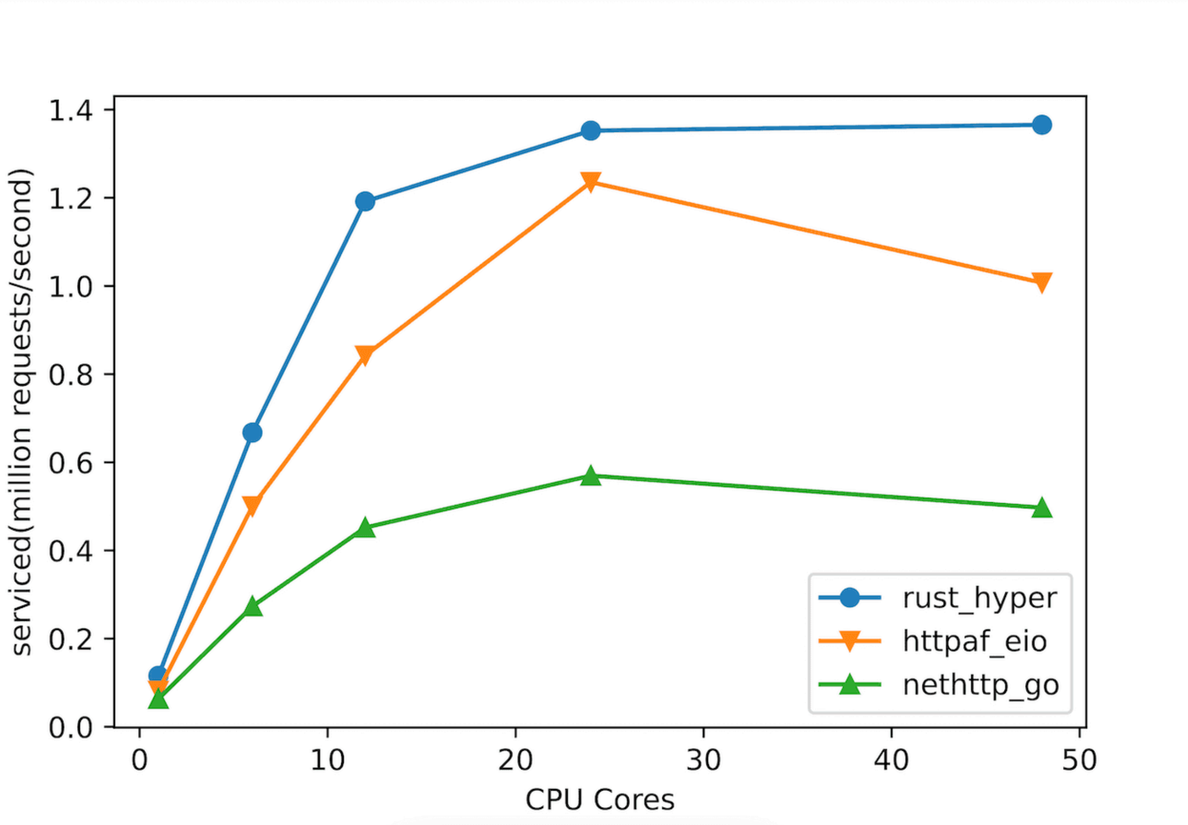
- *Same type safety* as the earlier syntactic version

```
1  open Effect
2  open Effect.Deep
3
4  type _ Effect.t += E : string t
5
6  let comp () =
7    print_string "0 ";
8    print_string (perform E);
9    print_string "3 "
10
11 let main () =
12   try_with comp ()
13   { effc = fun (type a) (e : a t) ->
14       match e with
15       | E -> Some (fun (k : (a,_) continuation) ->
16           print_string "1 ";
17           continue k "2 ";
18           print_string "4 ")
19       | e -> None }
```

Eio — Direct-style effect-based concurrency



HTTP server performance using 24 cores



HTTP server scaling maintaining a constant load of 1.5 million requests per second

Integration with Lwt & Async

- `Lwt_eio` allows running Lwt and Eio code together
  - Only sequential
  - Cancellation semantics is also integrated
  - Incrementally port Lwt applications to Eio

Integration with Lwt & Async

- `Lwt_eio` allows running Lwt and Eio code together
  - Only sequential
  - Cancellation semantics is also integrated
  - Incrementally port Lwt applications to Eio
- *Very experimental* `Async_eio` running Async and Eio code together
  - Required changes to Async

Merge Process

- Multicore OCaml was maintained as a separate *fork* of the compiler
  - ✦ Multiple *tricky* rebases to keep the fork up to date with trunk

Merge Process

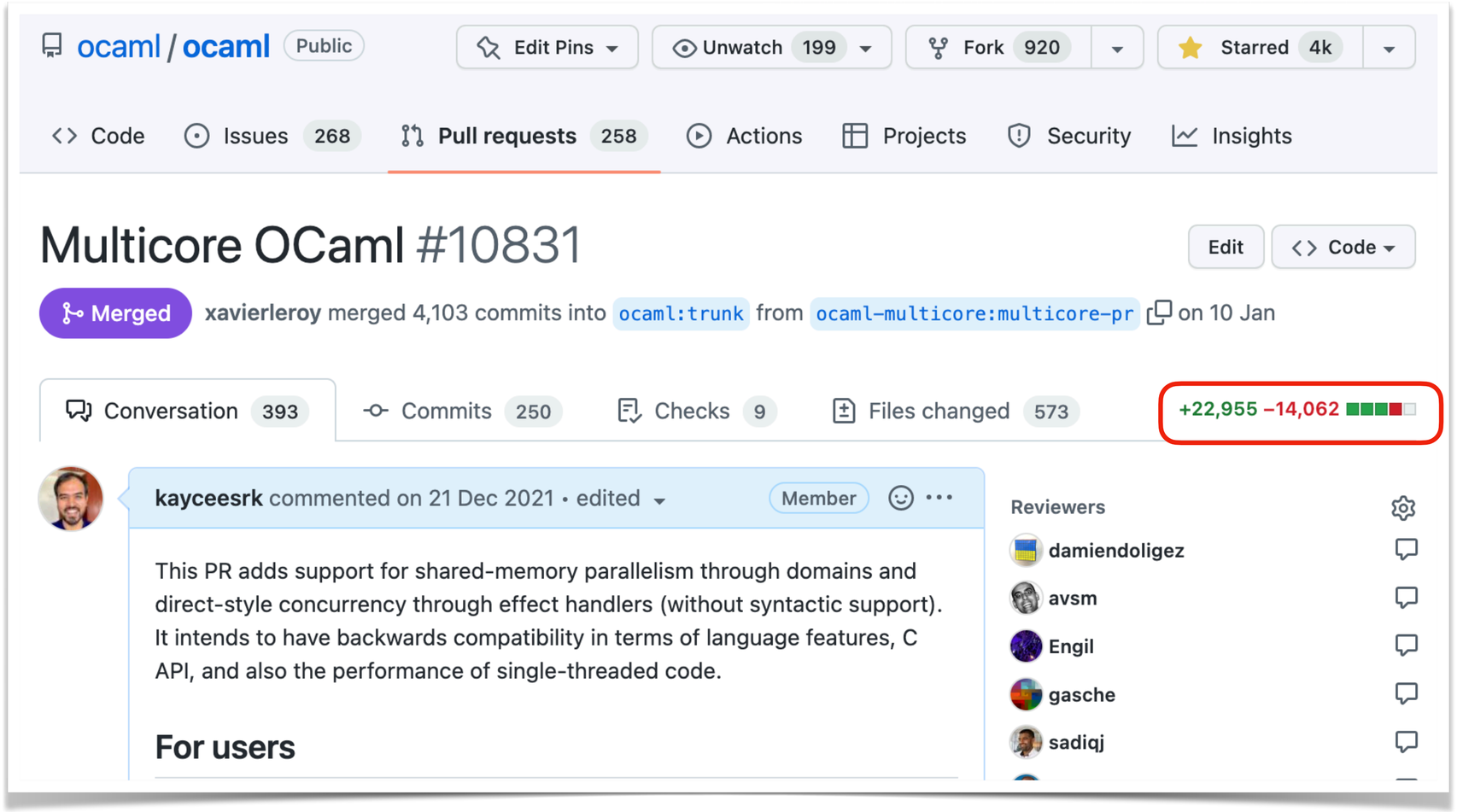
- Multicore OCaml was maintained as a separate *fork* of the compiler
  - ✦ Multiple *tricky* rebases to keep the fork up to date with trunk
- *Single PR* to merge multicore change
  - ✦ Not worth splitting into multiple PR — context loss

Merge Process

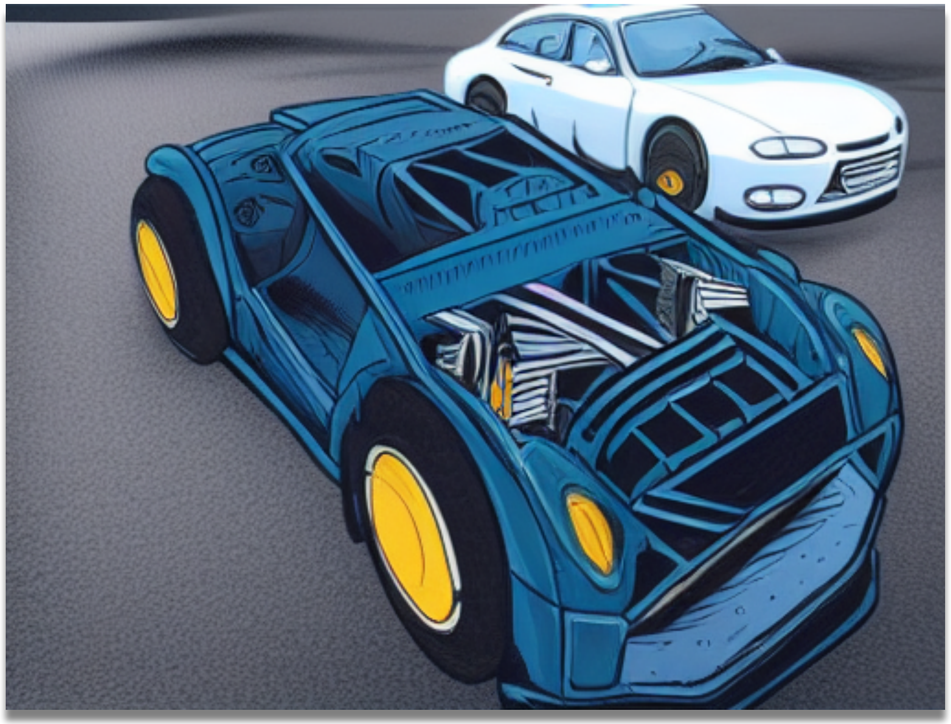
- Multicore OCaml was maintained as a separate *fork* of the compiler
  - ✦ Multiple *tricky* rebases to keep the fork up to date with trunk
- *Single PR* to merge multicore change
  - ✦ Not worth splitting into multiple PR — context loss
- Asynchronous & Synchronous review phases (Nov 2021)

Working Group	Multicore Lead	Dev Team Lead
GC	Tom Kelly & Sadiq Jaffer	Damien Doligez
Domains	Tom Kelly	Luc Maranget
Runtime multi-domain safety	Enguerrand Decorne	Xavier Leroy
Stdlib changes	KC Sivaramakrishnan	Florian Angeletti (and Gabriel Scherer)
Fibers	KC Sivaramakrishnan	Damien Doligez (and Xavier Leroy)

Merge Process

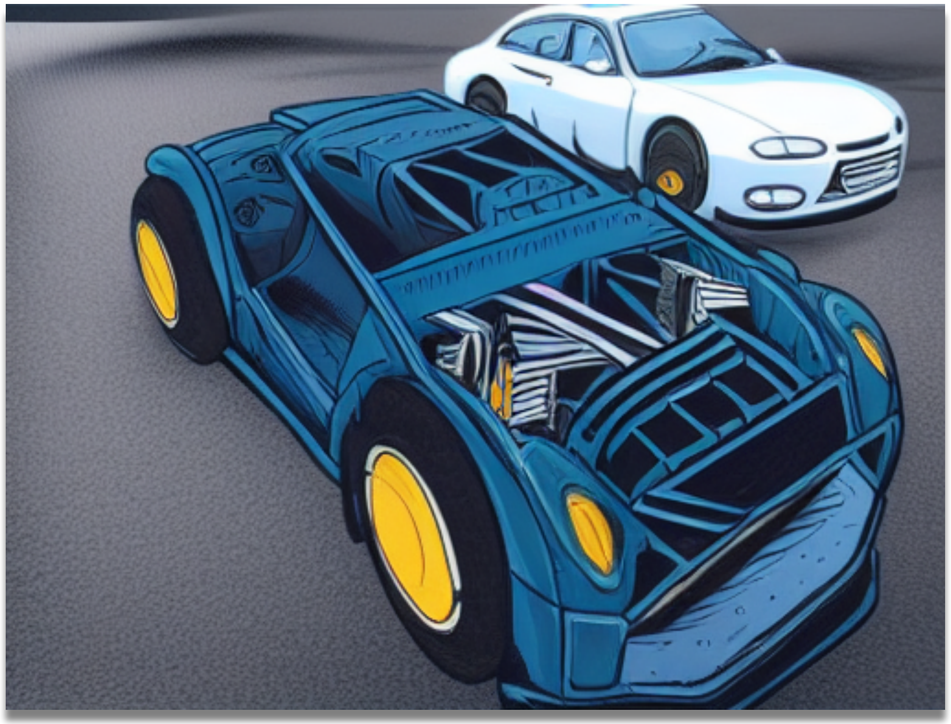


OCaml 5.0 — an MVP release



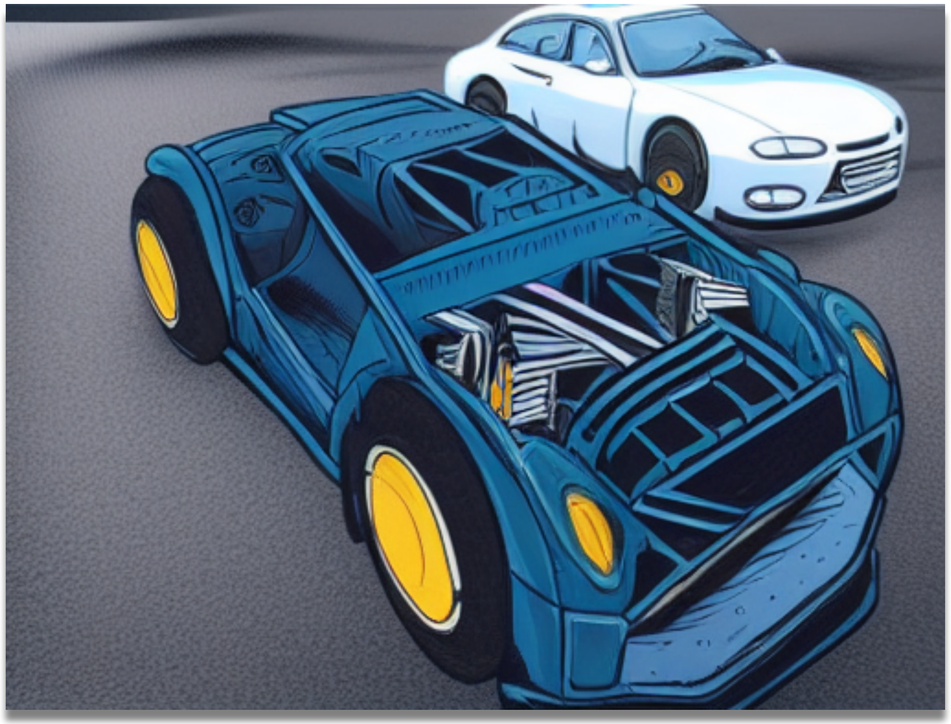
- Many features broken and are being added back
  - ✦ This will continue after 5.0 gets released

OCaml 5.0 — an MVP release



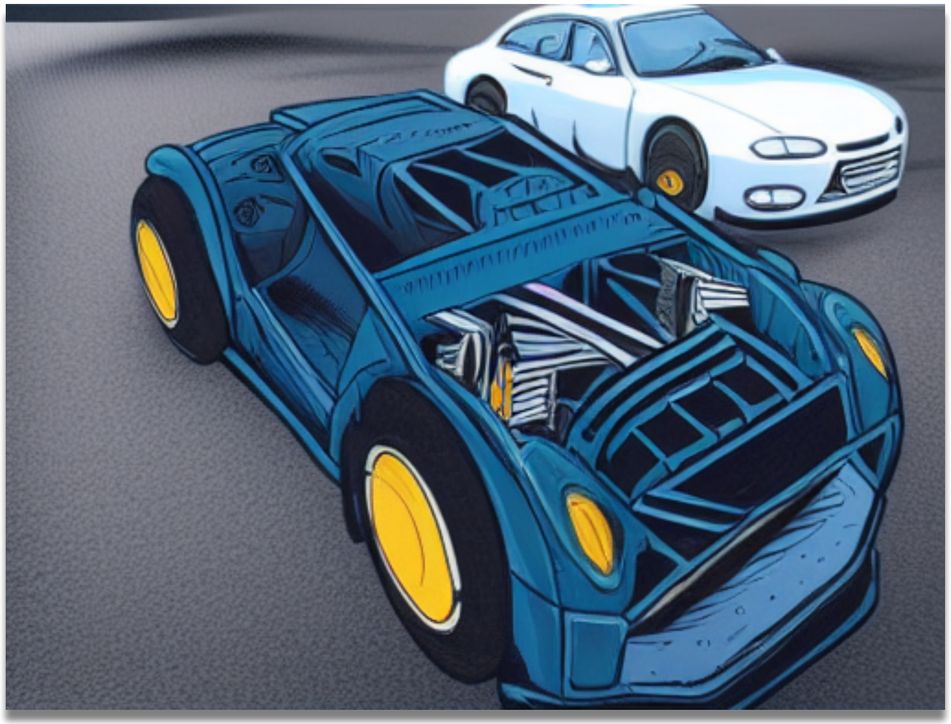
- Many features broken and are being added back
  - ✦ This will continue after 5.0 gets released
- Platform support
  - ✦ 32-bit will be *bytecode only*
  - ✦ On 64-bit,
    - ✤ x86-64 + Linux, macOS, Windows, OpenBSD, FreeBSD
    - ✤ Arm64 + Linux, macOS (*Apple Silicon*)
    - ✤ RISC-V (*PR open*)
  - ✦ JavaScript (jsoo) — effect handlers are not supported yet!

OCaml 5.0 — an MVP release



- GC performance improvements TBD
  - ✦ Decoupling major slice from minor GC
  - ✦ Mark stack prefetching
  - ✦ Best-fit vs multicore allocator

OCaml 5.0 — an MVP release



- GC performance improvements TBD
  - ✦ Decoupling major slice from minor GC
  - ✦ Mark stack prefetching
  - ✦ Best-fit vs multicore allocator
- Statmemprof
  - ✦ Work in progress for reinstating asynchronous action safety

Tidying

- We tidied up accumulated deprecations
  - `String.uppercase, lowercase, capitalize, uncapitalize`
  - `Stream, Genlex ~> camlp-streams`
  - `Pervasives, ThreadUnix` modules deleted
- Major version jump to make good changes
  - C function names are all prefixed uniformly
  - Additional libraries `Unix, Str` installed as findlib packages

OPAM Health Check

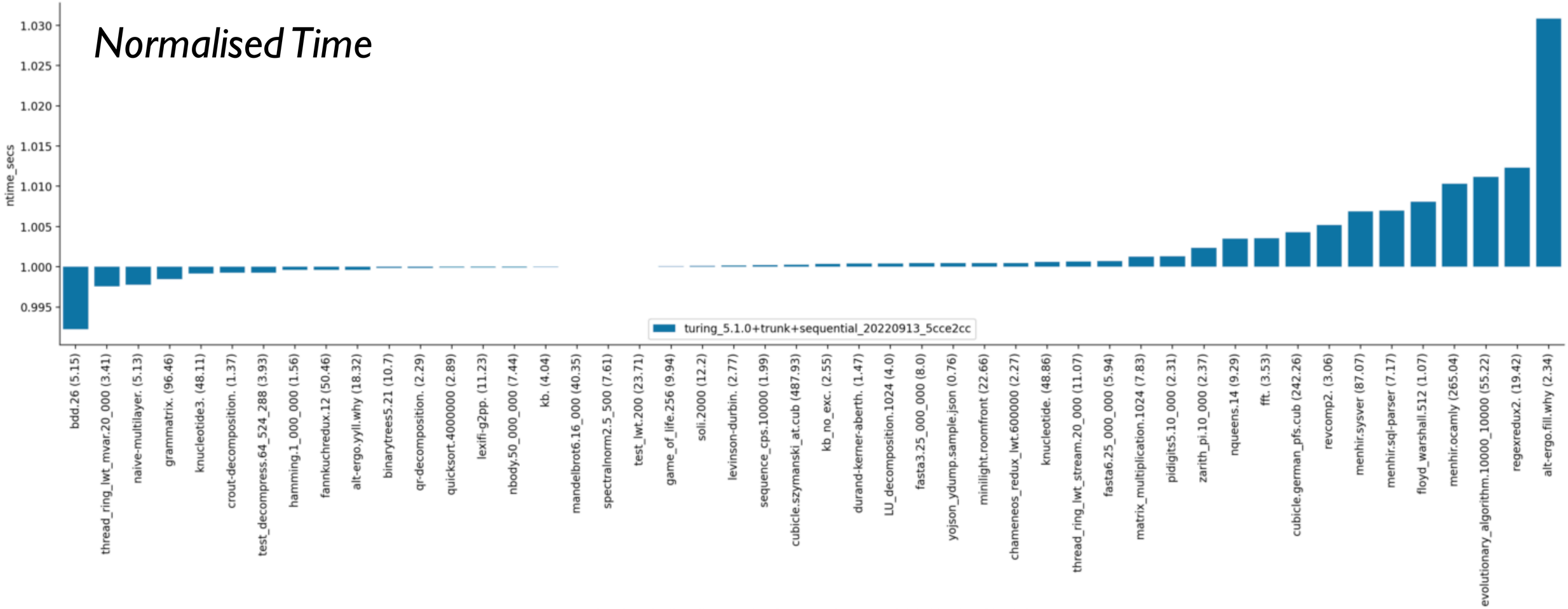
OPAM Health Check

	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

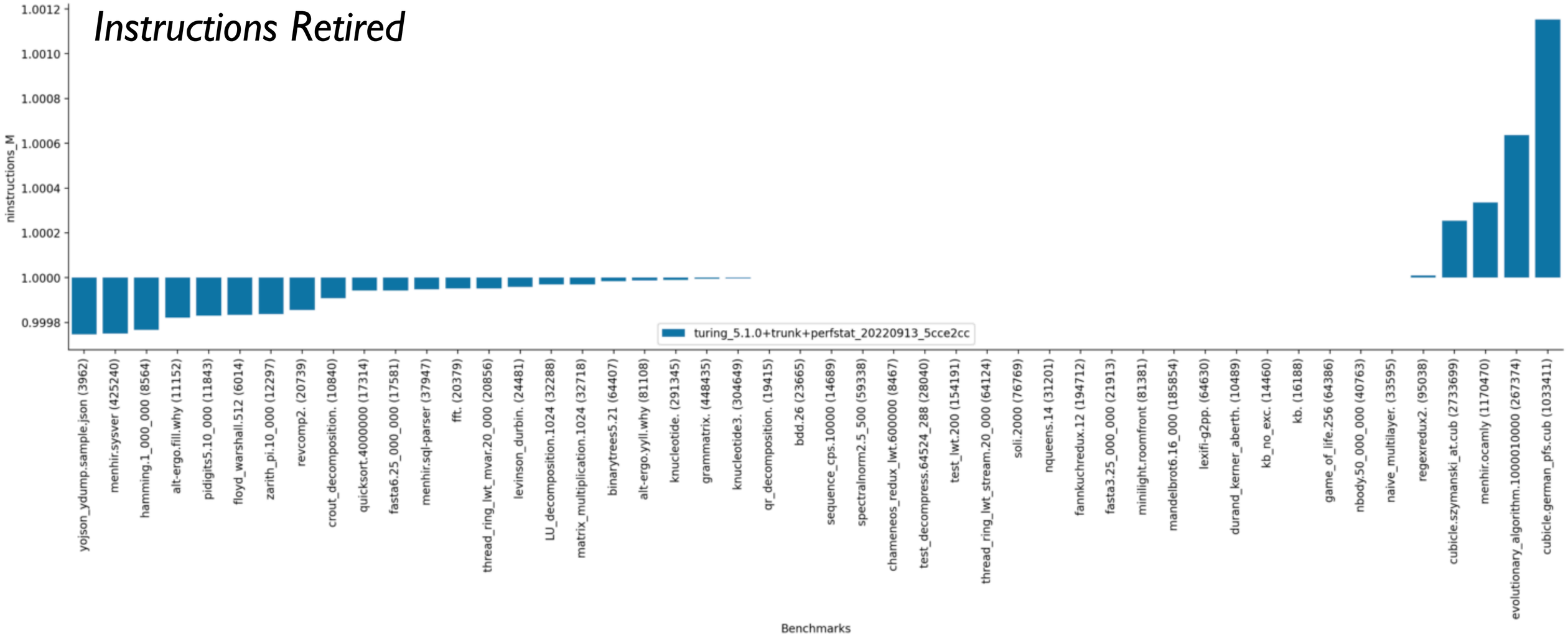
http://check.ocamllabs.io/

Sandmark Nightly Service

sandmark.tarides.com



Sandmark Nightly Service



OCaml 5.0 needs you!

- OCaml 4 will have longer term support than usual
- Even if you don't plan to use concurrency and parallelism, *switch to OCaml 5.0*
  - ✦ Only then can we move away from OCaml 4.x

OCaml 5.0 needs you!

- OCaml 4 will have longer term support than usual
- Even if you don't plan to use concurrency and parallelism, *switch to OCaml 5.0*
  - ✦ Only then can we move away from OCaml 4.x
- Sequential programs *must* work with same perf on 5.0
  - ✦ Test, deploy, evaluate, benchmark sequential programs in 5.0
  - ✦ Report bugs & performance regressions

OCaml 5.0 needs you!

- OCaml 4 will have longer term support than usual
- Even if you don't plan to use concurrency and parallelism, *switch to OCaml 5.0*
  - ✦ Only then can we move away from OCaml 4.x
- Sequential programs *must* work with same perf on 5.0
  - ✦ Test, deploy, evaluate, benchmark sequential programs in 5.0
  - ✦ Report bugs & performance regressions
- **What is stopping you from switching to OCaml 5.0?**
  - ✦ Let us know so that we can work on it!