



From OCaml 4 to 5 and from Parmap to Effects: A legacy code transition story

Nat Mote & Nathan Taylor
September 2025



Part 1: OCaml 4 to 5



What is Semgrep?

- Static analysis and code search tool written mostly in OCaml
- Parses code and patterns into Abstract Syntax Trees (ASTs) and matches patterns against code
- Taint analysis built atop pattern matching helps identify additional issues, particularly security issues.

```
rules:  
- id: test-rule  
  languages:  
  - ocaml  
  severity: ERROR  
  message: found match  
  pattern: print_endline("...")
```

```
1 let f x =  
2 | print_endline(x);  
3 | print_endline("this is  
  a literal string")
```



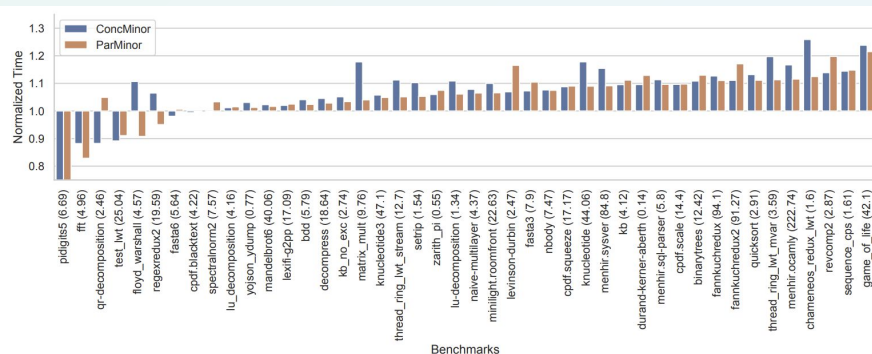
Why Upgrade to OCaml 5?

- Multicore (see part 2)!
- Algebraic effects (backbone of eio)
- Fixes for Windows
- Risk of falling behind the ecosystem
 - May not be able to use latest package versions
- BUT: We needed to upgrade without exposing our users to significant negative effects!

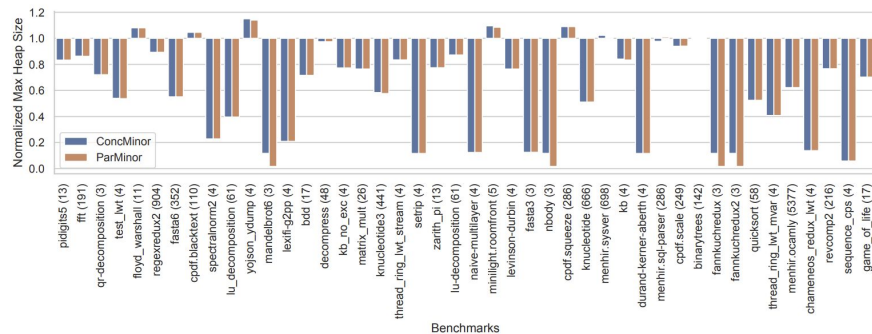
Background on OCaml Runtime Changes

- Introducing shared-memory parallelism (multicore) required re-architecting the garbage collector and memory allocator
- Maintainers ran benchmarks with positive results

[Retrofitting parallelism onto OCaml](#)
(Sivaramakrishnan et al.)



(a) Normalized runtime. Baseline is Stock OCaml whose running time in seconds is given in parenthesis.



(b) Normalized maximum major heap size. Baseline is Stock OCaml whose maximum major heap size in MB is given in parenthesis.

Initial Attempt in 2023



feat!: OCaml 5.0 #8194

Merged aryx merged 56 commits into `develop` from `ocaml5.0` on Oct 4, 2023

Conversation 33

Commits 56

Checks 0

Files



aryx commented on Sep 29, 2023

ALL CHECKS ARE PASSING!!!! IS IT TRUE!!! WE CAN SWITCH TO 5.0??



A small circular profile picture of a man with glasses and a blue shirt.
brandonspark commented on Jun 30, 2023 · edited ▾

What:

This PR updates our version of OCaml to 5.0.

Revert to OCaml 4 #8940

Merged aryx merged 2 commits into `develop` from `iago/revert-8194` on Oct 10, 2023

Conversation 20

Commits 2

Checks 0

Files changed 20



iagoAbal commented on Oct 9, 2023

Member ...

OCaml 5 has serious regressions regarding memory usage, running p/default-v2 on 46 repos from stress-test-monorepo we observed a 45% average increase in memory usage, in some cases Semgrep used almost twice as much memory as with OCaml 4.



What Went Wrong?

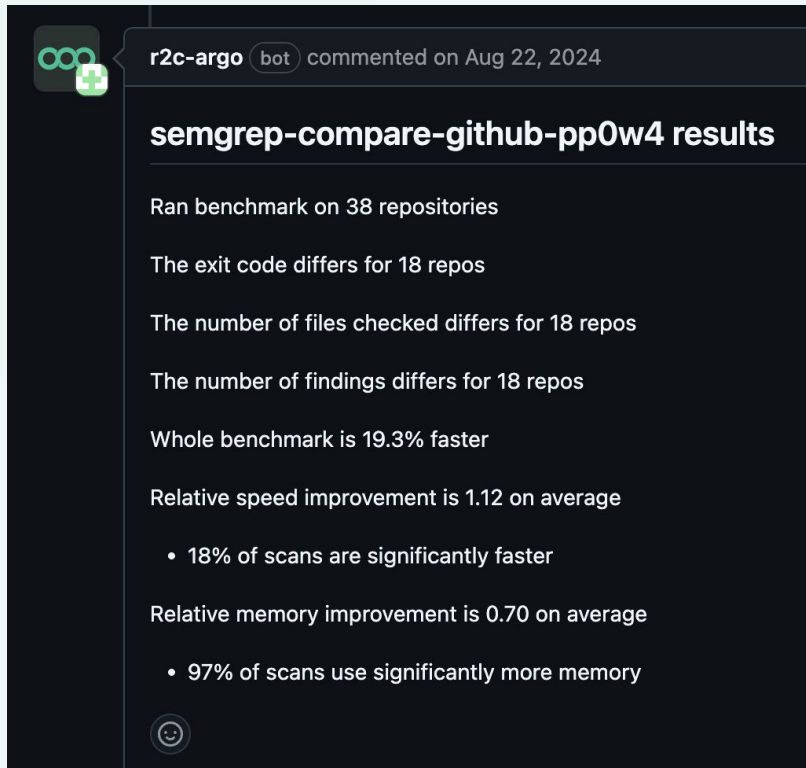
- OCaml 5.0 and 5.1 don't have compaction! Though this turned out not to be the issue
- [Infer ran into this](#)
- We assumed the lack of compaction was causing our problems too
- Decided to wait for 5.2 which has compaction (though must be explicitly called)

Moving to OCaml 5 we see a significant increase in memory pressure for our workload due to lack of compaction (=~ not releasing memory back to the OS).



Second Attempt

- Tried with 5.2
- Saw the same issues (unsurprising, since compaction has to be called explicitly)
- Tried adding a few calls to `Gc.compact` explicitly
- Read a bunch of runtime code
- Read about (and misunderstood) the new allocation code
- Looked into fragmentation related to the use of `malloc`
- Spent some time with [Memtrace](#)
 - At least made a few memory optimizations.



r2c-argo bot commented on Aug 22, 2024

semgrep-compare-github-pp0w4 results

Ran benchmark on 38 repositories

The exit code differs for 18 repos

The number of files checked differs for 18 repos

The number of findings differs for 18 repos

Whole benchmark is 19.3% faster

Relative speed improvement is 1.12 on average

- 18% of scans are significantly faster

Relative memory improvement is 0.70 on average

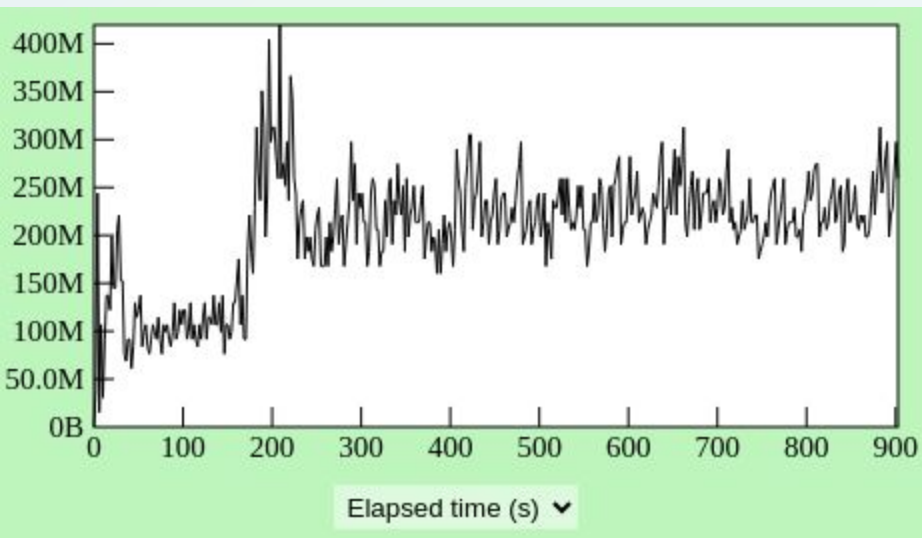
- 97% of scans use significantly more memory

🕒

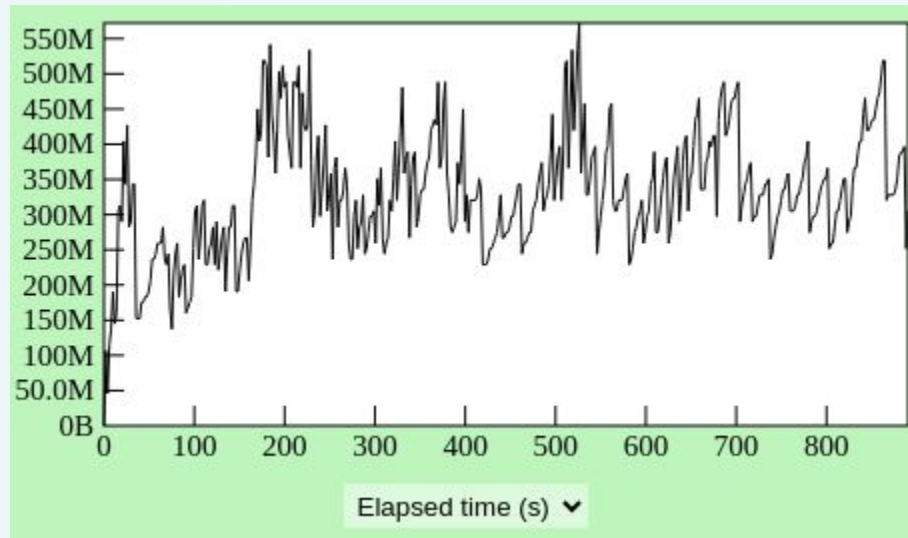


Memtrace Investigation

- Directly compared memtrace results on an interfile analysis of [Juice Shop](#)
- Shows increase is at least in part due to major GC behavior changes
- Note difference in scale



OCaml 4 with no GC tuning

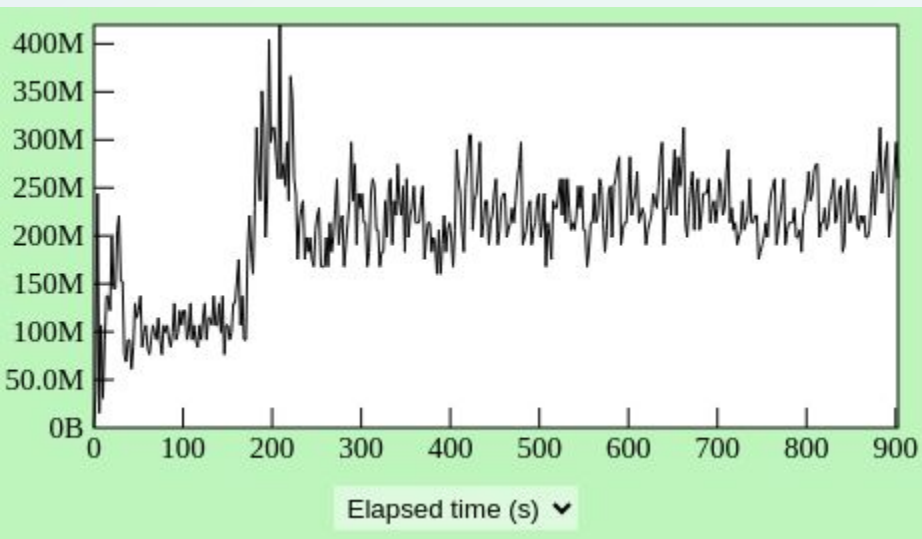


OCaml 5 with no GC tuning

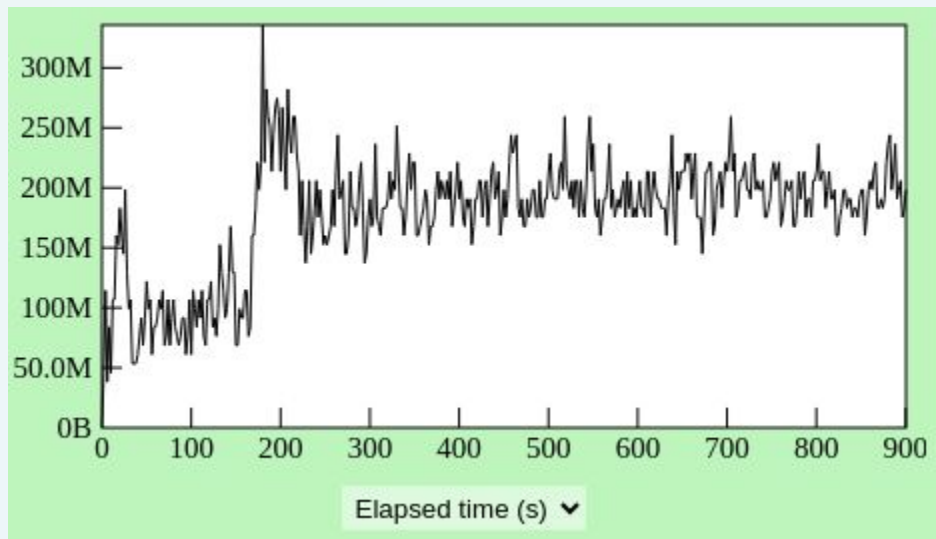


Tuning Garbage Collector

- Started experimenting with `space_overhead`
- Dropped it to 40 from default of 120 and got good spacetime results!
- But need to see if we also get good results on Resident Set Size (RSS).

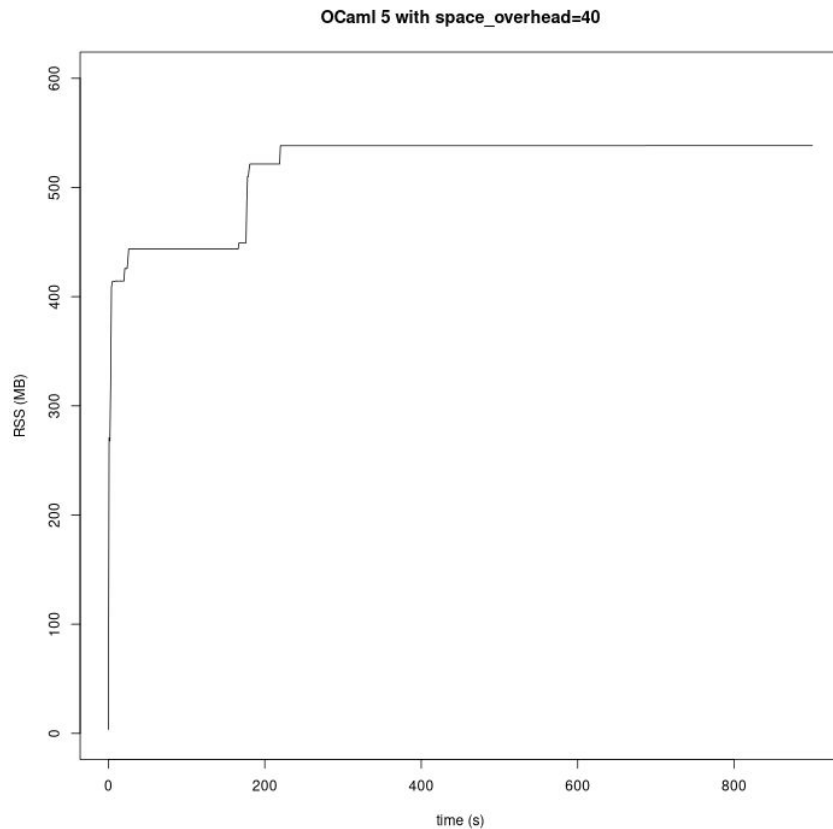
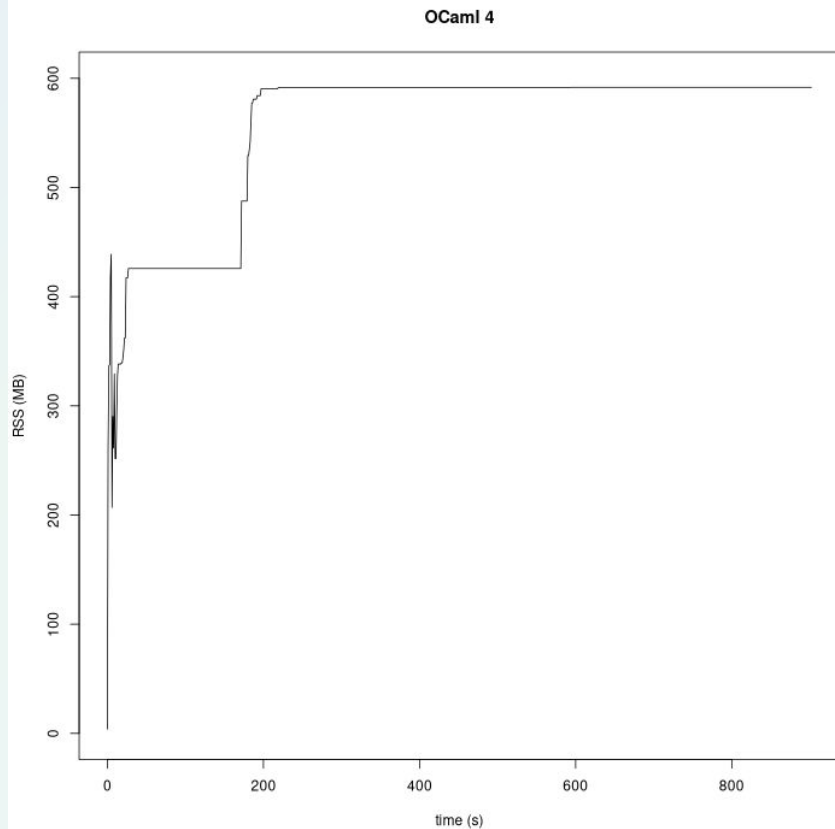


OCaml 4 with no GC tuning



OCaml 5 with GC tuning

Measuring RSS





Generalizing to Other Repositories

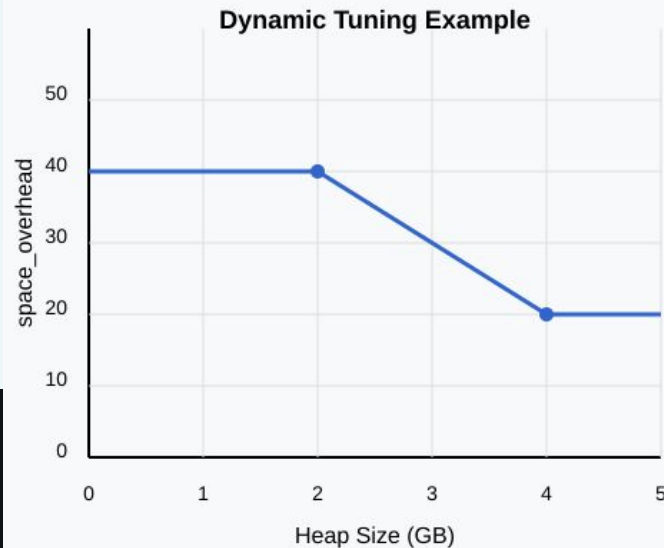
- Experimented on [Blaze Persistence](#) and found that we needed `space_overhead=20`.
- But that slowed down smaller repositories too much!
- Ran with automated benchmarks and found some even larger repos that needed an even smaller value!
- `space_overhead=15` seemed to work.



Dynamic GC Tuning

- No need for a single static value for **space_overhead**!
- Utility to adjust it based on heap size
- Tried it with the values below and it worked well
- Now [open source](#)
- Only used for interfile. Single-file scans do fine with a single static value.

```
(if USys.ocaml_release.major = 5 then
  DynamicGc.(
    setup_dynamic_tuning
    {
      min_space_overhead = 15;
      max_space_overhead = 40;
      heap_start_worrying_mb = 2_048;
      heap_really_worry_mb = 8_192;
    }));
```





Validation and rollout

- Did a/b testing with dry runs alongside real customer scans on our infrastructure
 - Showed no significant changes between OCaml 4 and OCaml 5 with GC tuning!
- Rolled out with a plan for a quick rollback if needed.
- Turned out entirely uneventful!
- Semgrep is now fully on OCaml 5 and we have begun to make use of some new features!



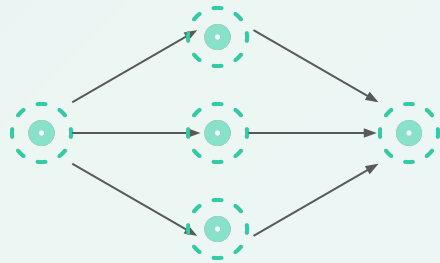
Upcoming Runtime Improvements

- I [reported an issue](#) on the OCaml repository and joined an OCaml runtime meeting to discuss my experiences.
- The maintainers identified two pull requests which they believe may make our GC tuning unnecessary:
 - <https://github.com/ocaml/ocaml/pull/13580>
 - <https://github.com/ocaml/ocaml/pull/13736>
- Both are now merged into mainline OCaml
- Additional fixes are available in [OxCaml](#)



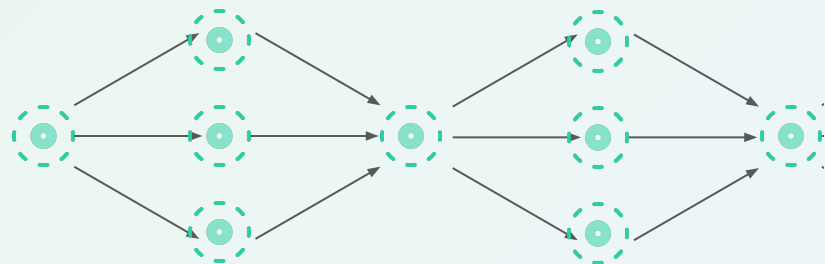
Part 2: Multicore





Parsing

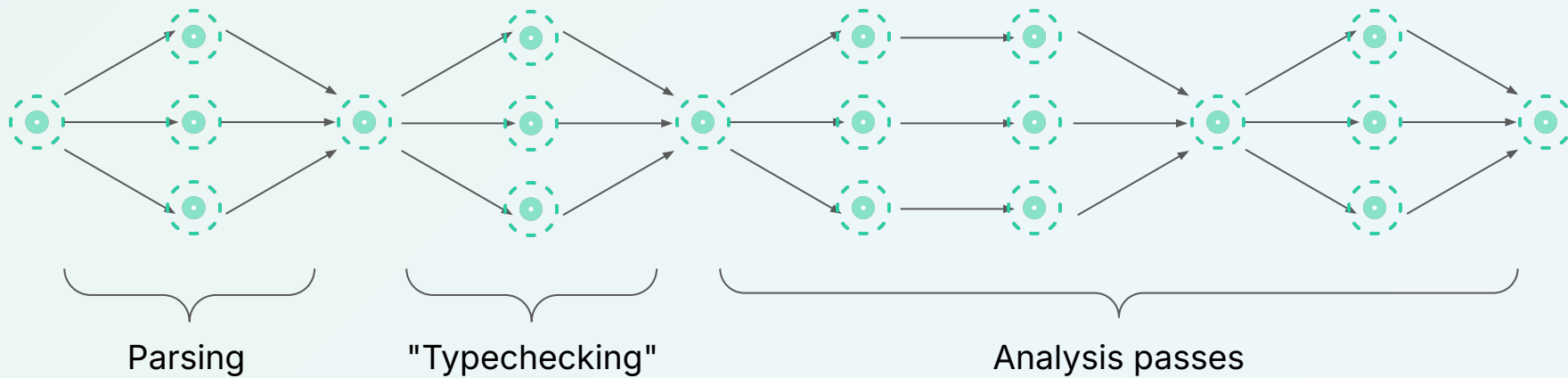


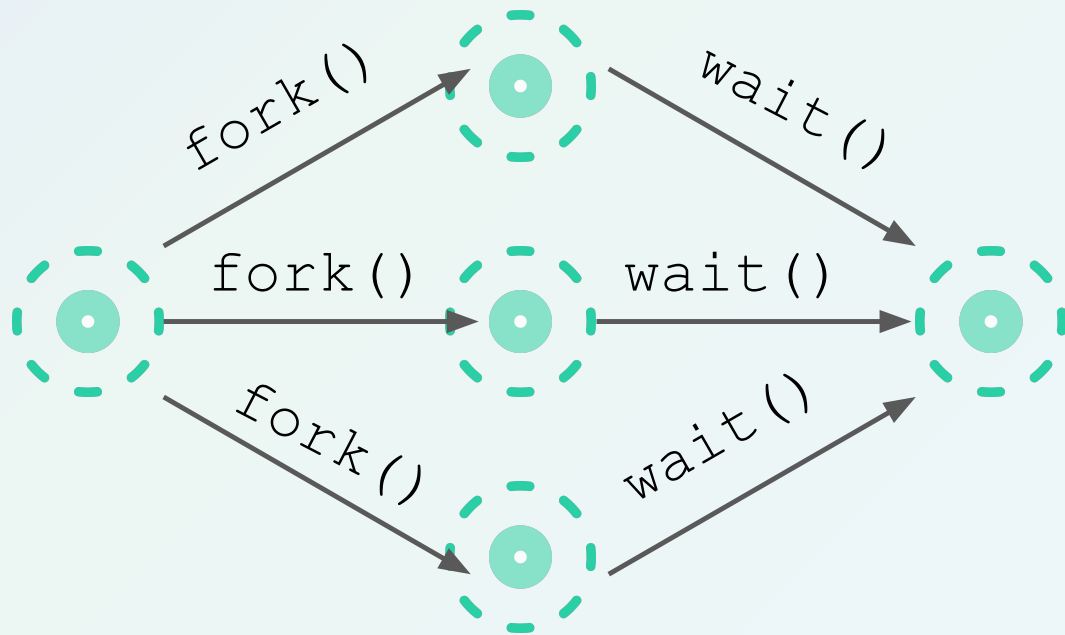





Parsing



"Typechecking"

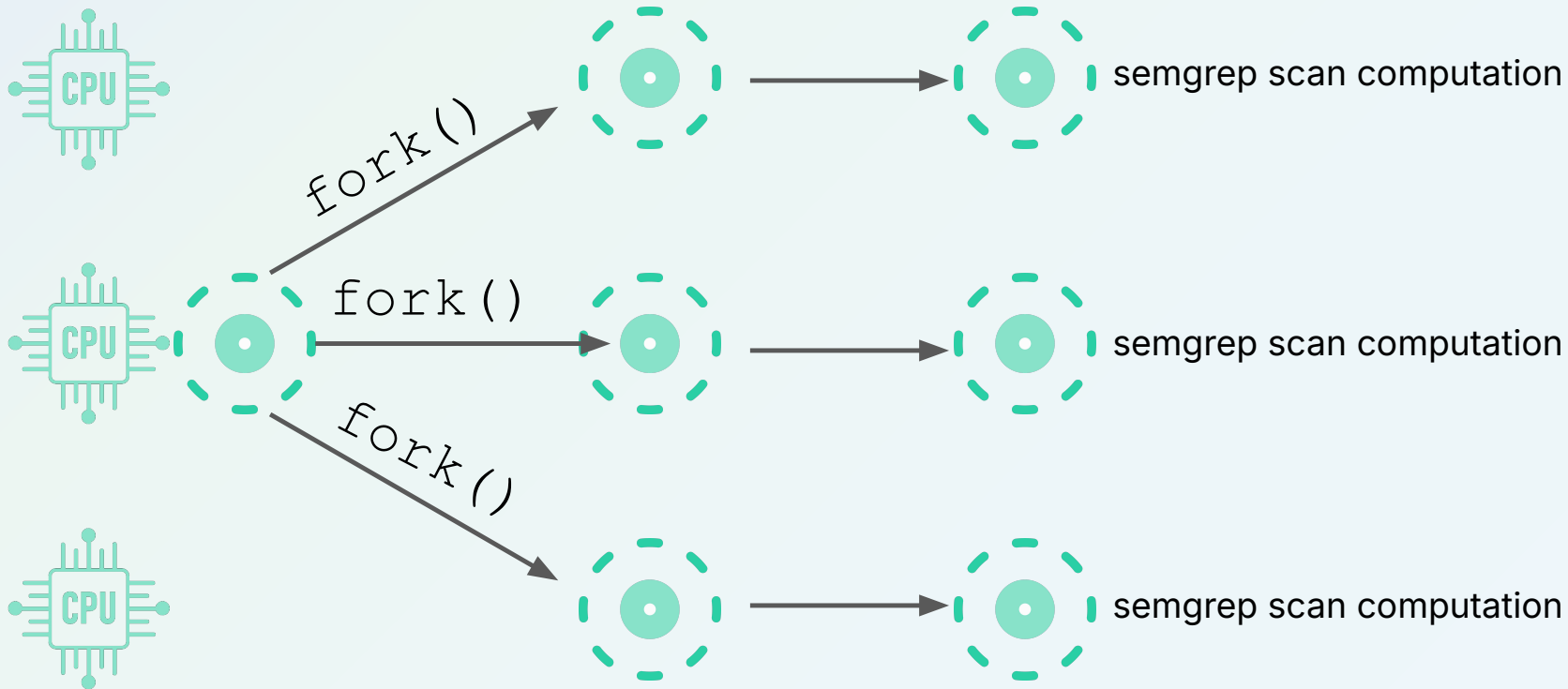


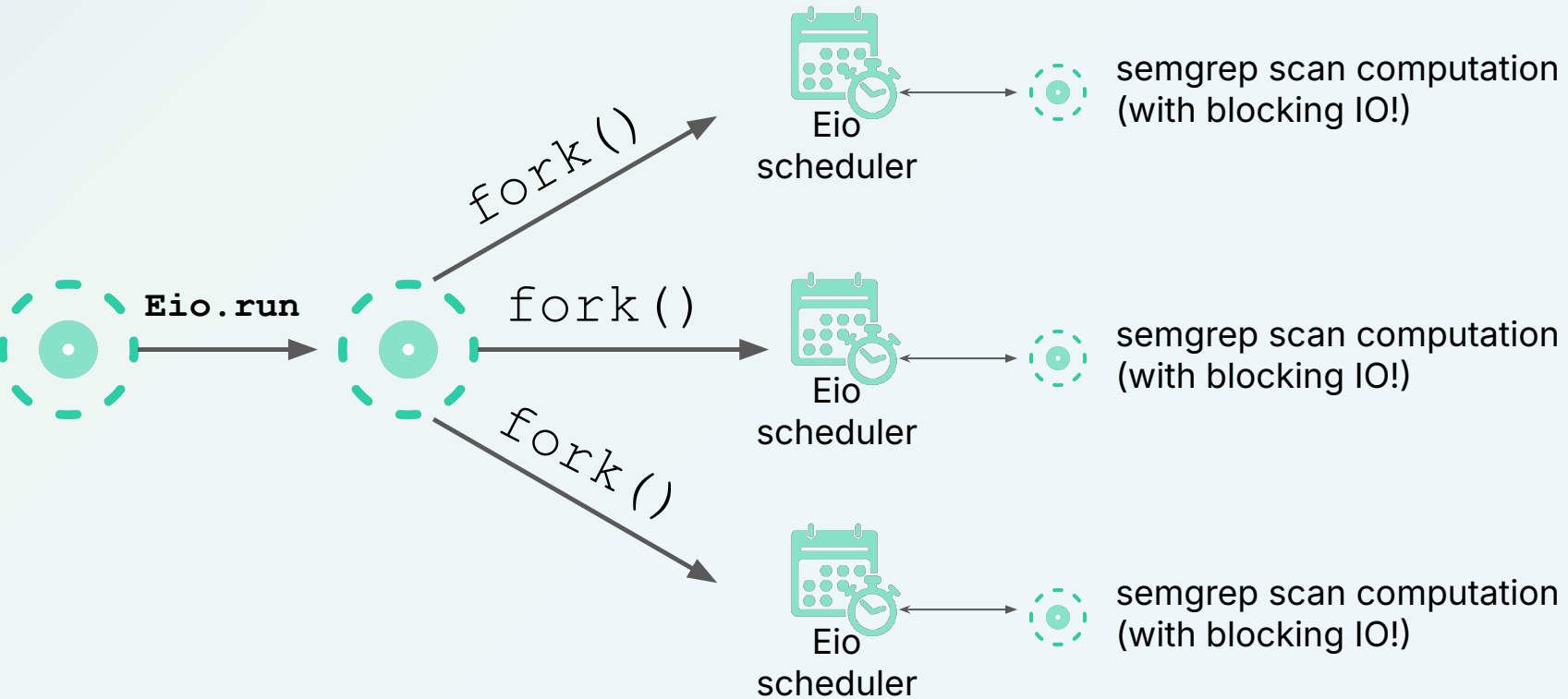
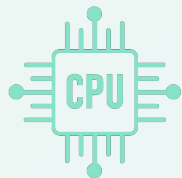
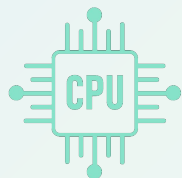
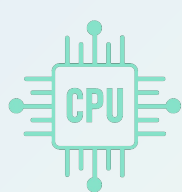


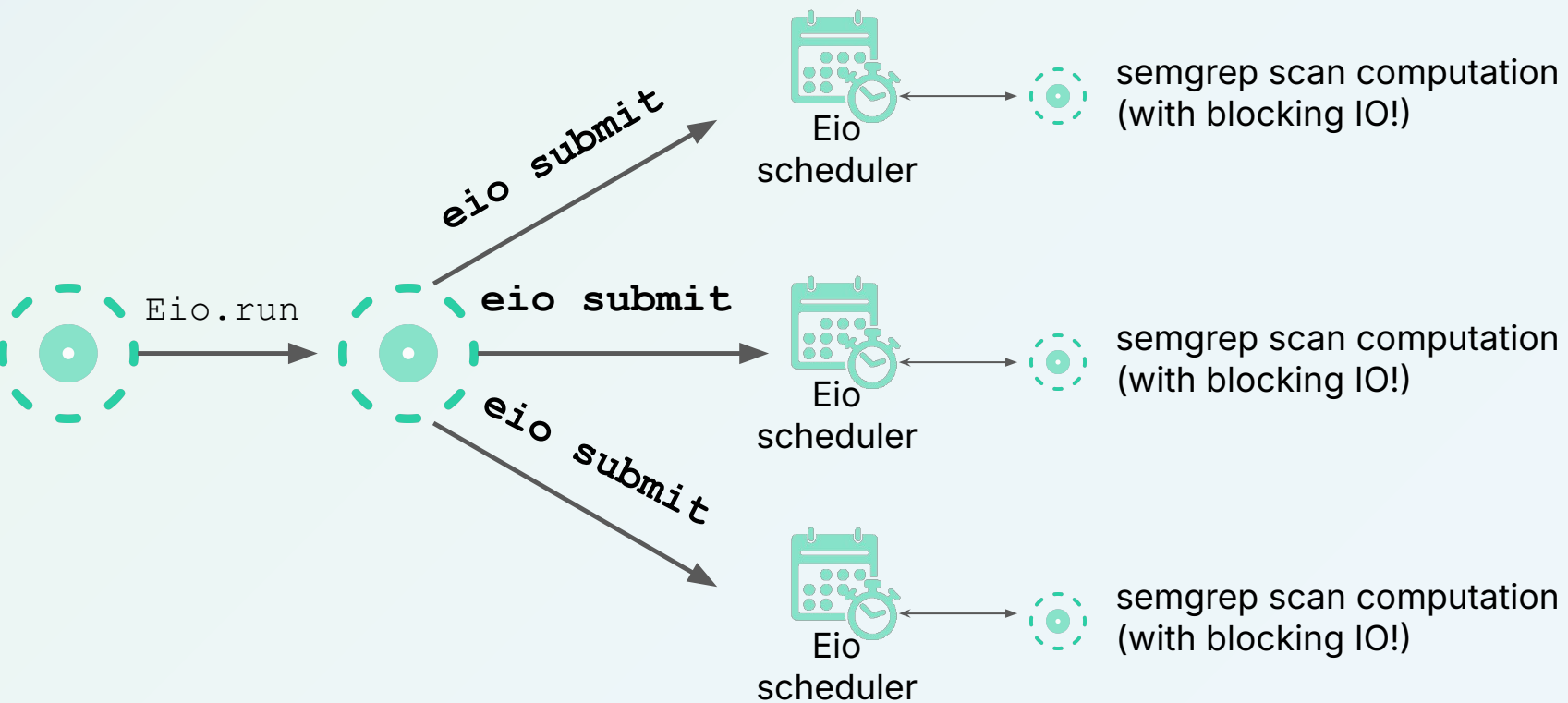
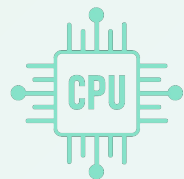
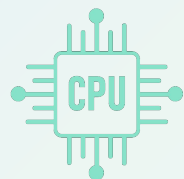
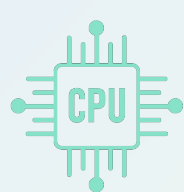
-  No shared memory, so we are free of data races
-  Programming model like calling a pure function
-  Each child has its own address space - copying and compacting the heap leads to linear memory overhead

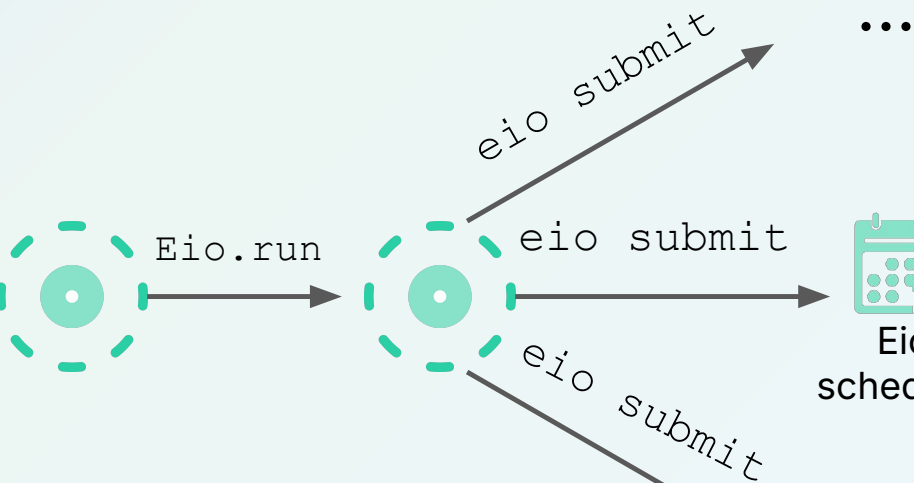
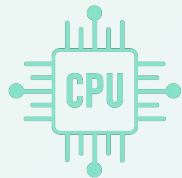
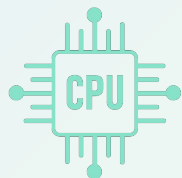
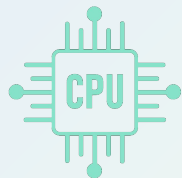


Our **goal**: Migrate *incrementally* to OCaml 5's newly-supported **shared memory parallelism**









...



async IO fiber



semgrep scan computation



timer fiber

...

Tracking down sources of mutable state







Tracking down sources of mutable state



Dynamic analysis

Workflow: "run the program with a race detector; observe and fix races; repeat!"

-  Sound!: A reported race is real.
-  Incomplete: will miss infrequently executed races, and initially very noisy
-  Finds the symptom, not the root cause
-  Operates at the OS thread level, so will not find inter-fiber races, nor non memory data races (e.g. with temp files)

Race in `Parse_typescript_tree_sitter.guess_dialect`

Yojison/Attdgen runtime is not threadsafe

UTmp hashtable is not threadsafe

Data races inside LWT's worker loop

`Parse_js` is not threadsafe

Menhir parser we use to parse patterns; see TSan output

```
=====
WARNING: ThreadSanitizer: data race (pid=92842)
  Write of size 8 at 0x00012553a338 by thread T6 (mutexes: write M0):
    #0 caml_modify memory.c:220 (Main.exe:arm64+0x103acc1a8)
    #1 camlStack_.push_364 <null> (Main.exe:arm64+0x102da837c)
    #2 camlLexer_js._ocaml_lex_initial_rec_705 <null> (Main.exe:arm64+0x102d9b000)
    #3 camlParse_js.token_1101 <null> (Main.exe:arm64+0x102687d3c)
    #4 camlParsing_helpers.tokens_aux_625 <null> (Main.exe:arm64+0x102c00000)
    #5 camlUFile.fun_2602 <null> (Main.exe:arm64+0x102da9bf0)
    #6 camlCommon.unwind_protect_723 <null> (Main.exe:arm64+0x102da45c8)
    #7 camlParse_json.parse_program_404 <null> (Main.exe:arm64+0x10263ff0)
    #8 camlParse_target.fun_2517 <null> (Main.exe:arm64+0x1025a6330)
    #9 camlPfff_or_tree_sitter.fun_1892 <null> (Main.exe:arm64+0x1025a3000)
```

Tracking down sources of mutable state



Example: what do you think of "Never shall we have a 'a ref value within the body of `Domain.spawn`'?"

Static analysis

Workflow: "scan the program with a static analyzer and an specification describing what a data race might be"

- ✓ Doesn't require running the code!
- 🤔 Overapproximates: may flag a violation even if in practice it is impossible to trigger
- 🤔 Writing a useful and correct specification can be really difficult
 - Can you think of ways my specification is poor? (I can think of at least three...)

```
File: bin/main.ml
1  let fetch_and_inc r = r := (!r + 1)
2
3  let () =
4    let forty_two: int ref = ref 42 in
5
6    (* safe !*)
7    let _ = fetch_and_inc forty_two in
8
9    (* unsafe! *)
10   let _ = Domain.spawn(fun () → fetch_and_inc forty_two) in
11
12   ()

1 Code Finding

bin/main.ml
>>> rules.ref-in-domain-spawn
Found a ref in a Domain.spawn()!

10! let _ = Domain.spawn(fun () → fetch_and_inc forty_two) in
```

Data representation of mutable state






```
1 (* Previously, each child process would get their
2  * own private copy of a top-level mutable value.
3  *
4  * Of course, in a shared-memory parallel world,
5  * this is no longer safe. *)
6 let mutable_ref = ref 42
7
8
9
10
11
12
13
14
15
```

Data representation of mutable state





```
1 (* Previously, each child process would get their
2  * own private copy of a top-level mutable value.
3  *
4  * Of course, in a shared-memory parallel world,
5  * this is no longer safe. *)
6 let mutable_ref = ref 42
7
8
9
10
11
12 (* An ['a Eio.Fiber.key] is similar but is
13  * scoped to an Eio fiber. *)
14 let mutable_fls: int Eio.Fiber.Key =
15     Eio.Fiber.create_key ()
```

-  The right abstraction given we are using Eio for concurrency!
-  Heavyweight implementation (stored in a per-fiber hash table)
-  Non-obvious semantics: FLS values inherited when a fiber forks, *unless* the fiber is forked across domains!

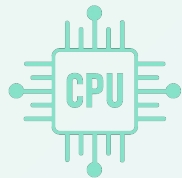
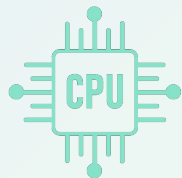
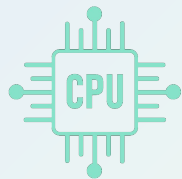
Data representation of mutable state



```
1 (* Previously, each child process would get their
2  * own private copy of a top-level mutable value.
3  *
4  * Of course, in a shared-memory parallel world,
5  * this is no longer safe. *)
6 let mutable_ref = ref 42
7
8 (* [Domain.DLS] behaves similarly to, in C, a
9  * value of type `pthread_key_t`. *)
10 let mutable_dls = Domain.DLS.create (const 42)
11
12
13
14
15
```

-  Cheap to access
-  Fatal flaw: Racey if two fibers on the same domain mutate the same DLS value....!

Data representation of mutable state



`Eio.run`

`eio submit`

`eio submit`



semgrep scan computation



async IO



semgrep scan computation



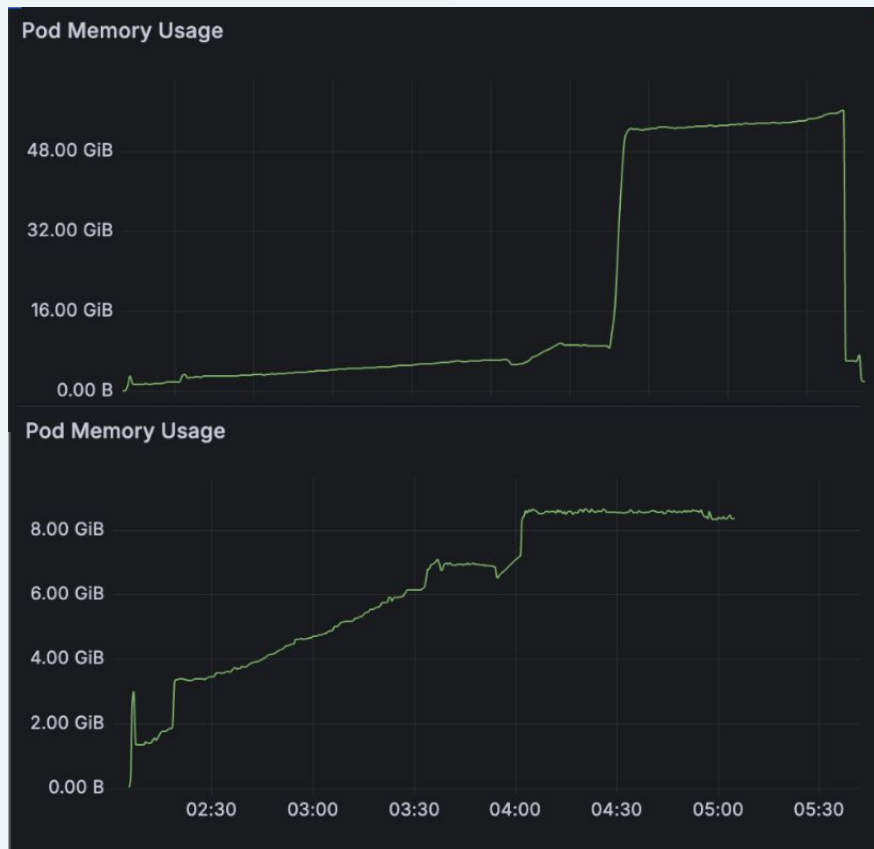
another semgrep scan computation





Conclusion

- **Memory usage** for large parallel scans reduced from
(baseline * number of CPUs)
to
(baseline + epsilon)!
- **Scan times** improved by ~10-15%!
- *All thanks to the hard work of the **OCaml** and **Multicore OCaml** project maintainers!*





Questions