Profiling the Memory Usage of OCaml Applications without Changing their Behavior

OCaml 2013

Çagdas Bozman[1,2,3]
Thomas Gazagnaire[1]
Fabrice Le Fessant[2]
Michel Mauny[3]

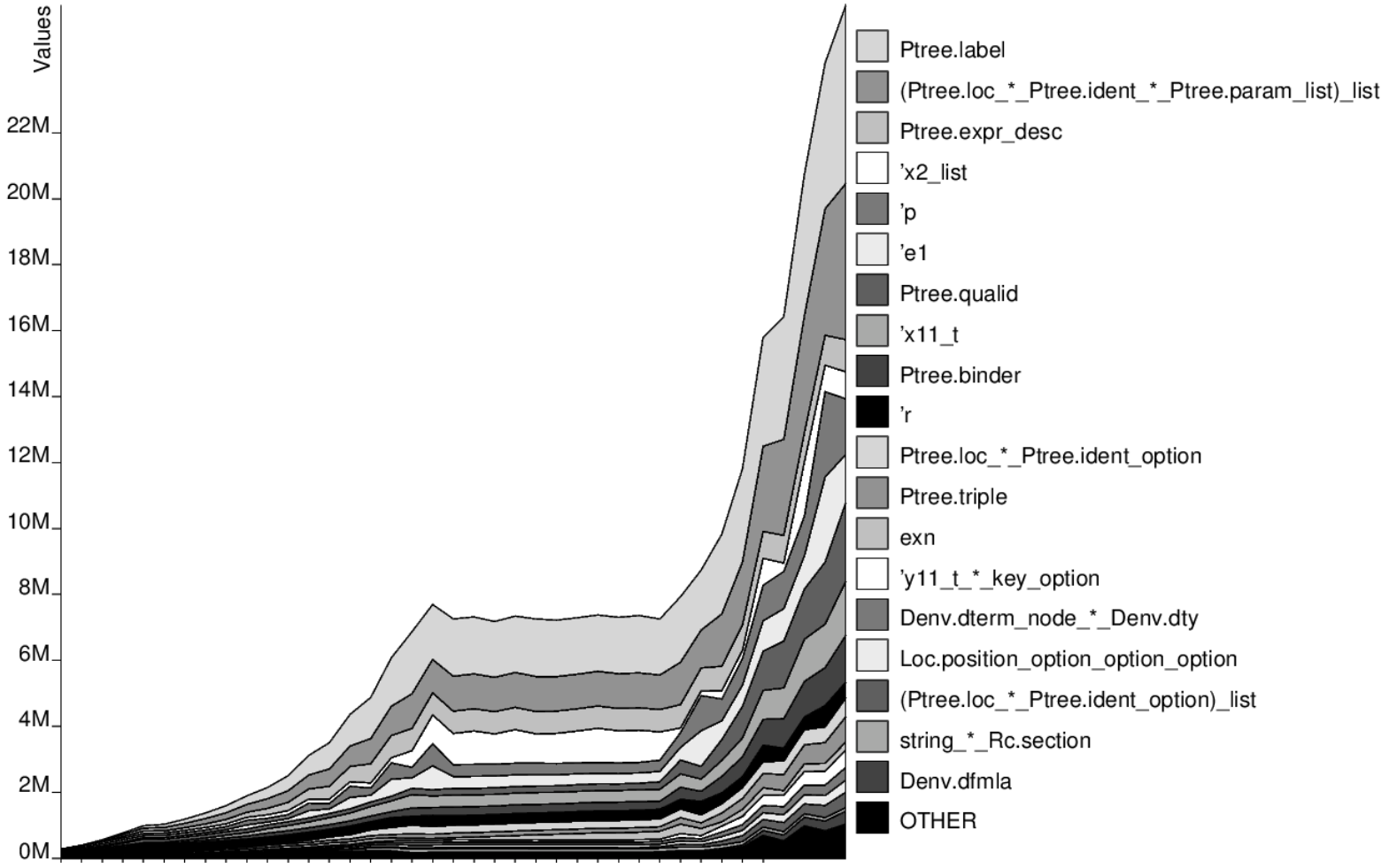
OCamlPro[1] | INRIA[2] | ENSTA-ParisTech[3]

Sept. 24 2013 - Boston (MA)

Memory Problems

- What ?
  - Study the memory behavior of OCaml programs
  - Memory profiling tools
- Why ?
  - To decrease memory footprint
  - To fix memory leaks
  - To spend less time in memory management

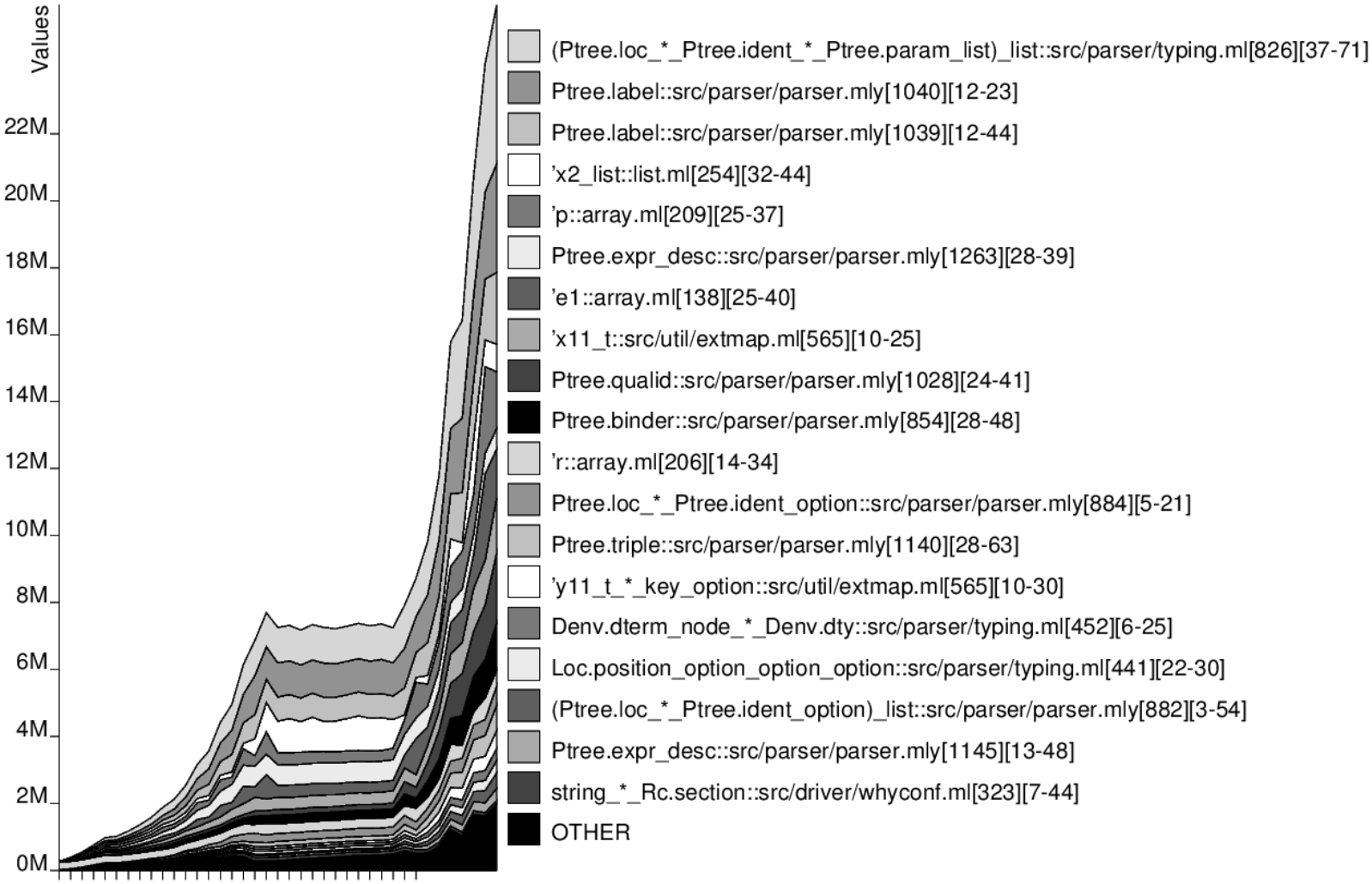
Real World Example – Why3[1] (1/2)



[1]Why3 is a platform for deductive program verification (http://why3.lri.fr/)

Real World Example – Why3 (2/2)

With locations precision



How do we do that ?

```
$ opam switch 4.00.1+ocp-memprof
$ opam install why3

$ OCAMLRUNPARAM=m why3replayer.opt -C why3.conf p9_16
```
this step will generate a lot of snapshots of heap image

No need to change your code nor the compilation options.
No impact on execution time.

```
$ opam install ocp-memprof
$ ocp-memprof -loc -sizes PID
```
this step analyzes all these snapshots

Look at the graphs.

Snapshots

What is a snapshot ?

- Compressed version of the heap
- Location identifiers, graph with pointers, etc.
- Save globals (toplevel modules)

How do we obtain these snapshots ?

- Computed by a linear scan of all chunks[2] which contain sets of consecutive blocks.

[2] huge block of memory

Generate A Snapshot

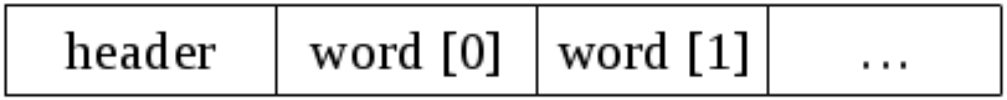
Two ways to trigger the generation of snapshots

- Use `OCAMLRUNPARAM=m` force a program to generate a snapshot after every GC
- Request explicitly the program to generate a snapshot
  - by sending a HUP signal (very useful for server-like application, cf mldonkey)
  - in module `GC`, use the following function

```
val dump_heap :  string -> unit
```

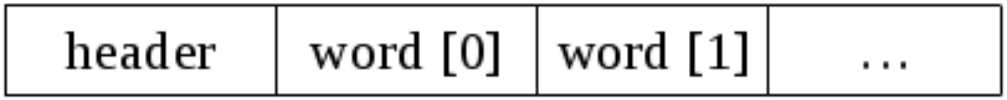
Patched Compiler (1/3)

OCaml memory block:



Patched Compiler (1/3)

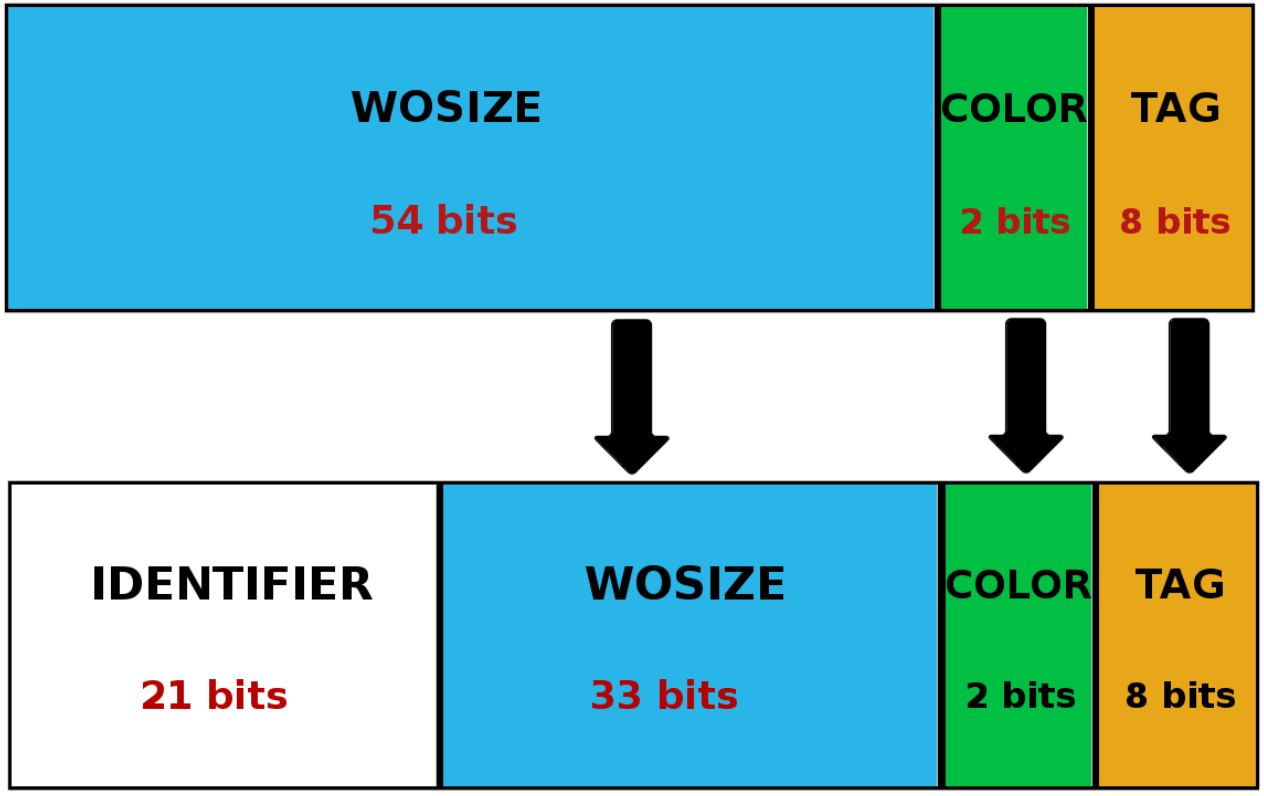
OCaml memory block:



OCaml block's header (one word) on 64-bit machines:

Patched Compiler (2/3)

Header after our modification:



Patched Compiler (3/3)

- Minimal impact on performance (only when generating snapshots)

Patched Compiler (3/3)

- Minimal impact on performance (only when generating snapshots)
- No space overhead

Patched Compiler (3/3)

- Minimal impact on performance (only when generating snapshots)
- No space overhead
- No impact on GC (its behavior is not changed)

Patched Compiler (3/3)

⊕

- Minimal impact on performance (only when generating snapshots)
- No space overhead
- No impact on GC (its behavior is not changed)

⊖

- Only on 64-bit platforms

Patched Compiler (3/3)

⊕

- Minimal impact on performance (only when generating snapshots)
- No space overhead
- No impact on GC (its behavior is not changed)

⊖

- Only on 64-bit platforms
- Location identifiers are limited ($2^{21} \sim 2$ million allocation sites)

Patched Compiler (3/3)

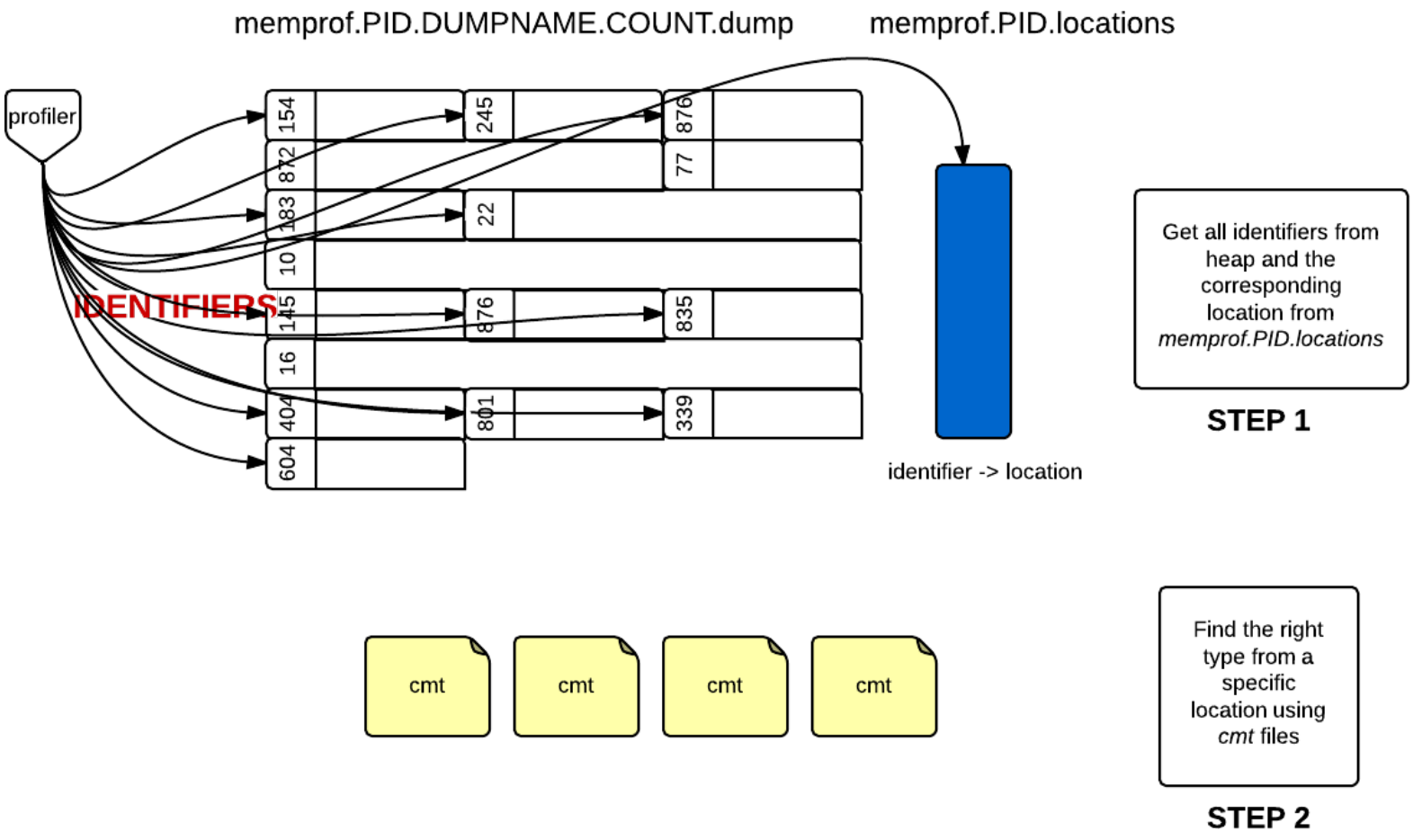
⊕

- Minimal impact on performance (only when generating snapshots)
- No space overhead
- No impact on GC (its behavior is not changed)

⊖

- Only on 64-bit platforms
- Location identifiers are limited ($2^{21} \sim 2$ million allocation sites)
- Maximum block size is now 64GB

One Tool Based On Identifiers



*A cmt file is a binary file containing the typed AST

Conclusion

Future Work:

- Improve the current framework
  - Aggregate information by type and location (work in progress)
  - Recover more types (e.g. using G.Henry's work)
  - Display life span of values (number of GC for example)
- More tools based to analyzed snapshots:
  - a graphical assistant to explore snapshots
  - a tool which use pointers to see which root retains some specific values

Questions ?