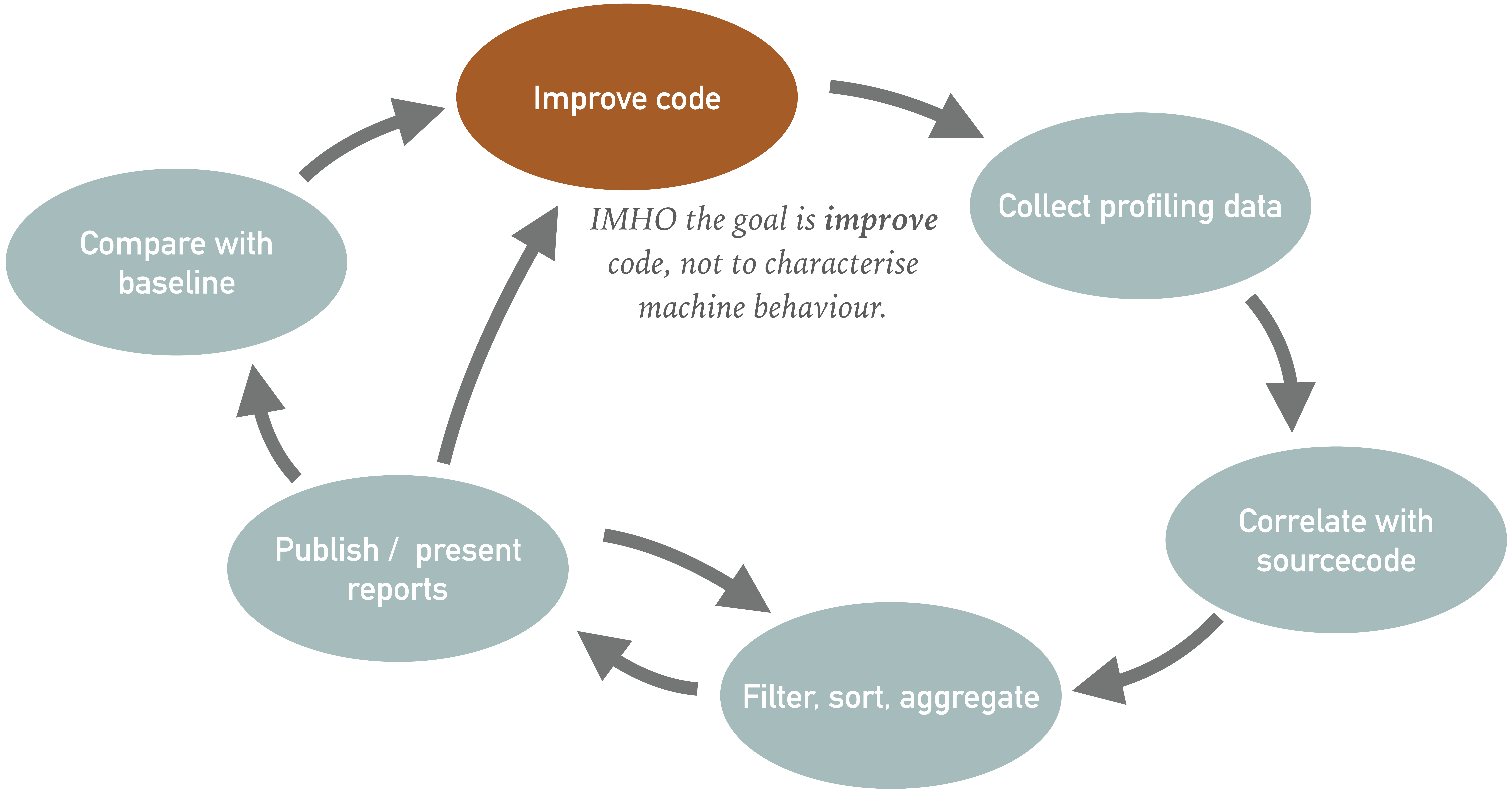
HSF & PROFILERS

Giulio Eulisse

SETTING

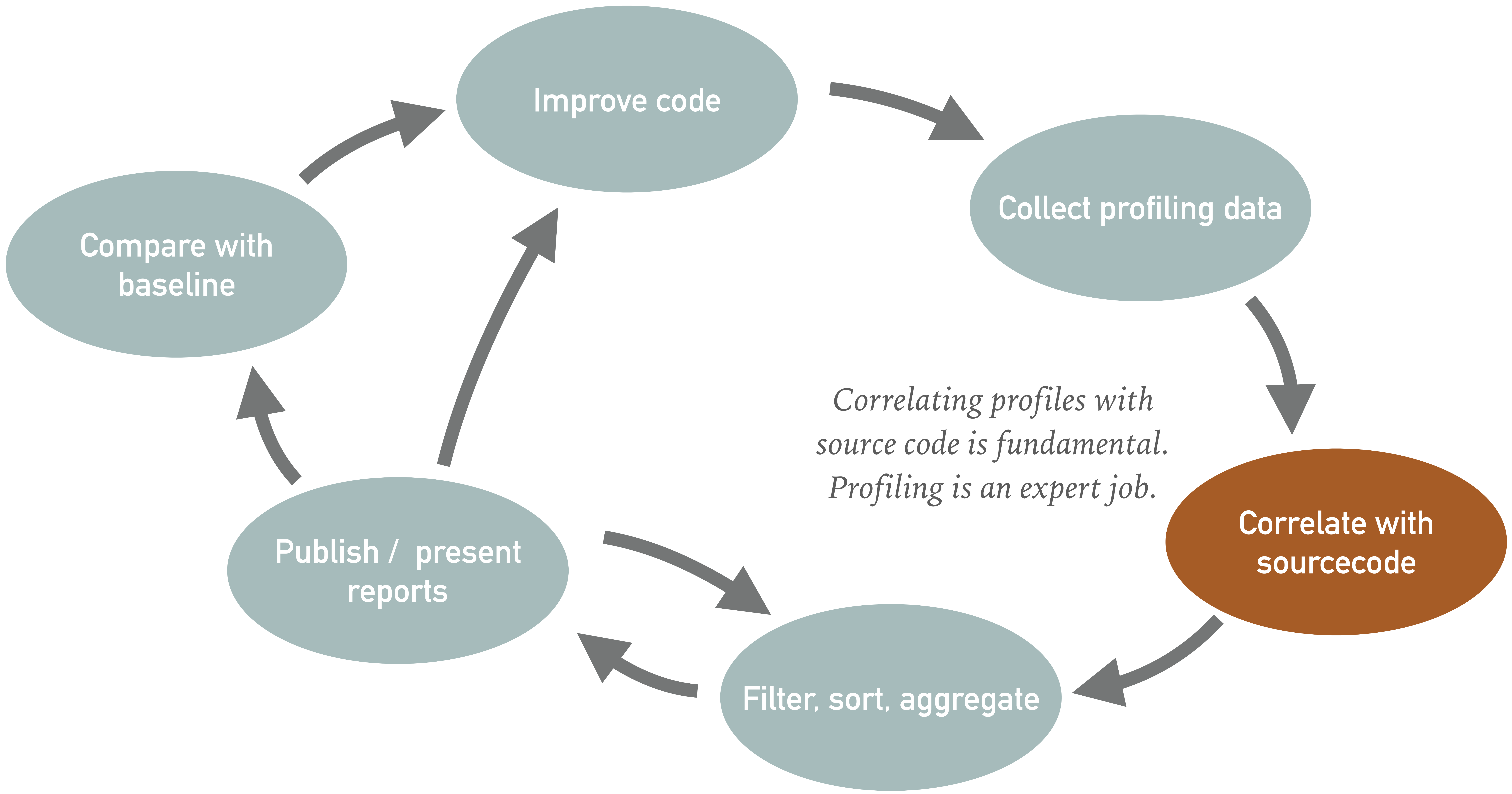
- *In Naples workshop we agreed that some common effort on code profilers would be good.*
- *Many tools, lot sparse expertise over very long time. Time to get together.*
- *This is to kickstart discussions, share ideas, define usecases and the way forward.*
- *Currently AFAIK there is 0 FTE associated to this and all the work is done on a voluntary basis or as part of our daily duty cycle.*

PROFILING: THE GIULIO DIAGRAM



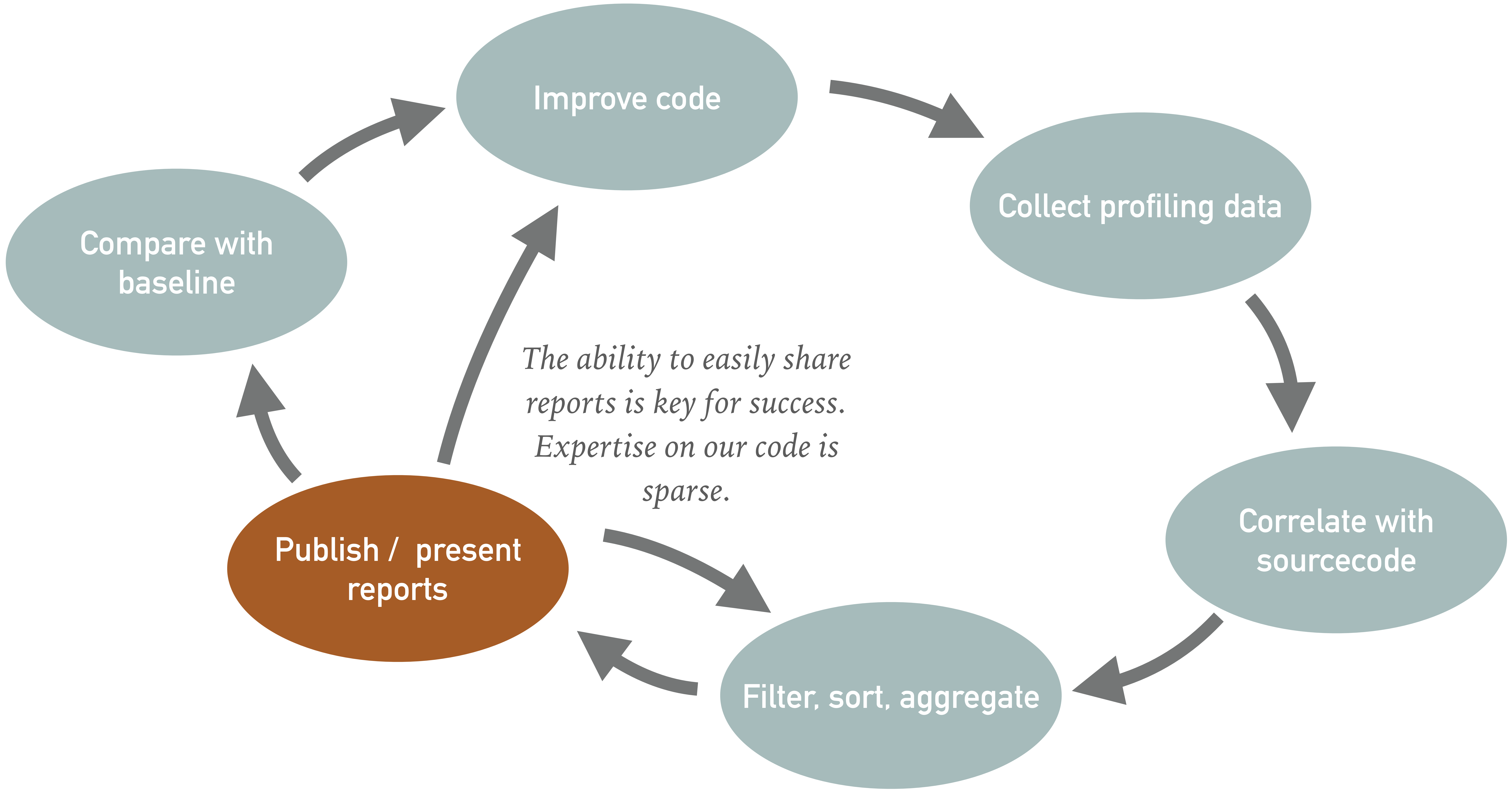
IMHO the goal is ***improve*** *code, not to characterise machine behaviour.*

PROFILING: THE GIULIO DIAGRAM



Correlating profiles with source code is fundamental. Profiling is an expert job.

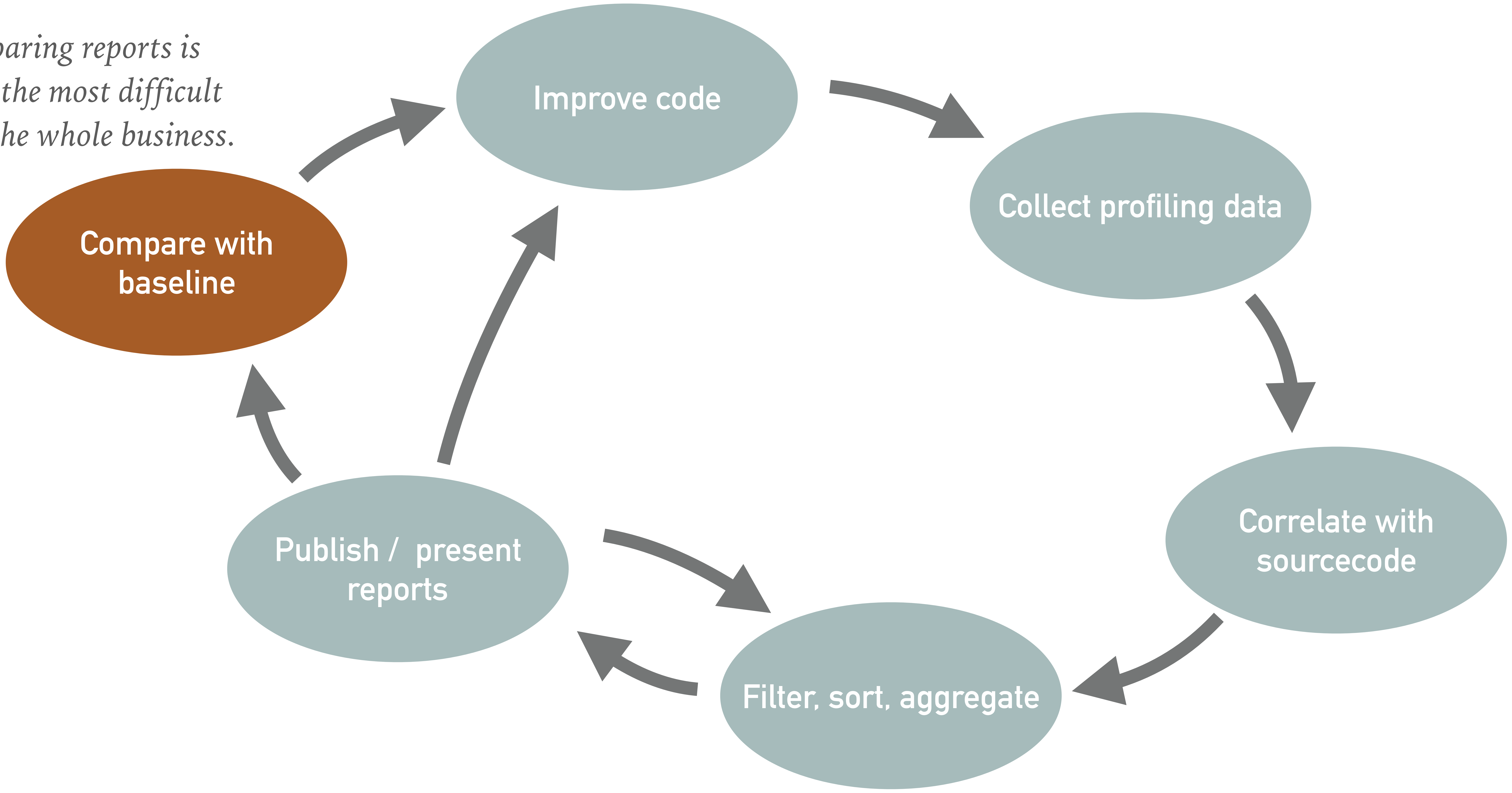
PROFILING: THE GIULIO DIAGRAM



The ability to easily share reports is key for success. Expertise on our code is sparse.

PROFILING: THE GIULIO DIAGRAM

Comparing reports is IMHO the most difficult part of the whole business.



COLLECTING PROFILE DATA

Key is diversity. *Many profiling tools, all with their strengths and usecases. It makes absolutely no sense to try to have one to rule them all (unless you want to be OSX only, then you use Instruments).*

- ***Hardware counters*** ⇒ *perf, oprofile, Instruments (OSX only), nvprof (GPU)*
- ***Sampling profilers*** ⇒ *Google Perf Tools, Cachegrind, IgProf, cProfile (Python), pprof (Go)*
- ***Memory profilers*** ⇒ *Valgrind, IgProf, Google Perf Tools, pprof*
- ***Instrumentation toolkits*** ⇒ *gprof, dtrace, IgTrace, Pin, go tool trace (Go)*
- ***All in one suites*** ⇒ *Instruments, Chrome Profiler (javascript), VTune*

Non exhaustive list: https://en.wikipedia.org/wiki/List_of_performance_analysis_tools

What can we do together here?

- *Document and present the various tools, their strengths and usecases. Make them easily accessible where not the case.*
- *Move everything to Mac and just adopt Instruments. ;-)*
- *Move everything to go ;-) (kudos to Sebastien for some of the links)*

STORING PROFILE DATA

Can we model profile data warehouse? *Storing profile data is a challenging task, however I personally think here it's possible to find a common description in terms of profile events.*

- ***Measure.*** *What and how much a given event counts.*
- ***Position inside the source code.*** *Where the event happens. In which line of code, in which library.*
- ***Stack-trace.*** *How we got there.*
- ***State.*** *E.g. arguments to each function in the stacktrace.*
- ***Time.*** *When a given event happened.*
- ***Metadata.*** *What user produced the profile, on what machine, for what workload.*

Questions:

- *Can we agree on a common way to model profile data?*
- *Is there already a viable backend available to store profile data?*
- *If not, can we get together and have one deployed "as a service"? Can we have it support "common" profile reports?*
- *Do we want to work together on it?*

ANALYSING AND VISUALISING PROFILE DATA

Analysis

All profilers provide ways to group and filter their data. Can we abstract those?

E.g.:

- *Group together all the contributions from libc.*
- *Split the report based on the value of a given function argument.*

Visualisation

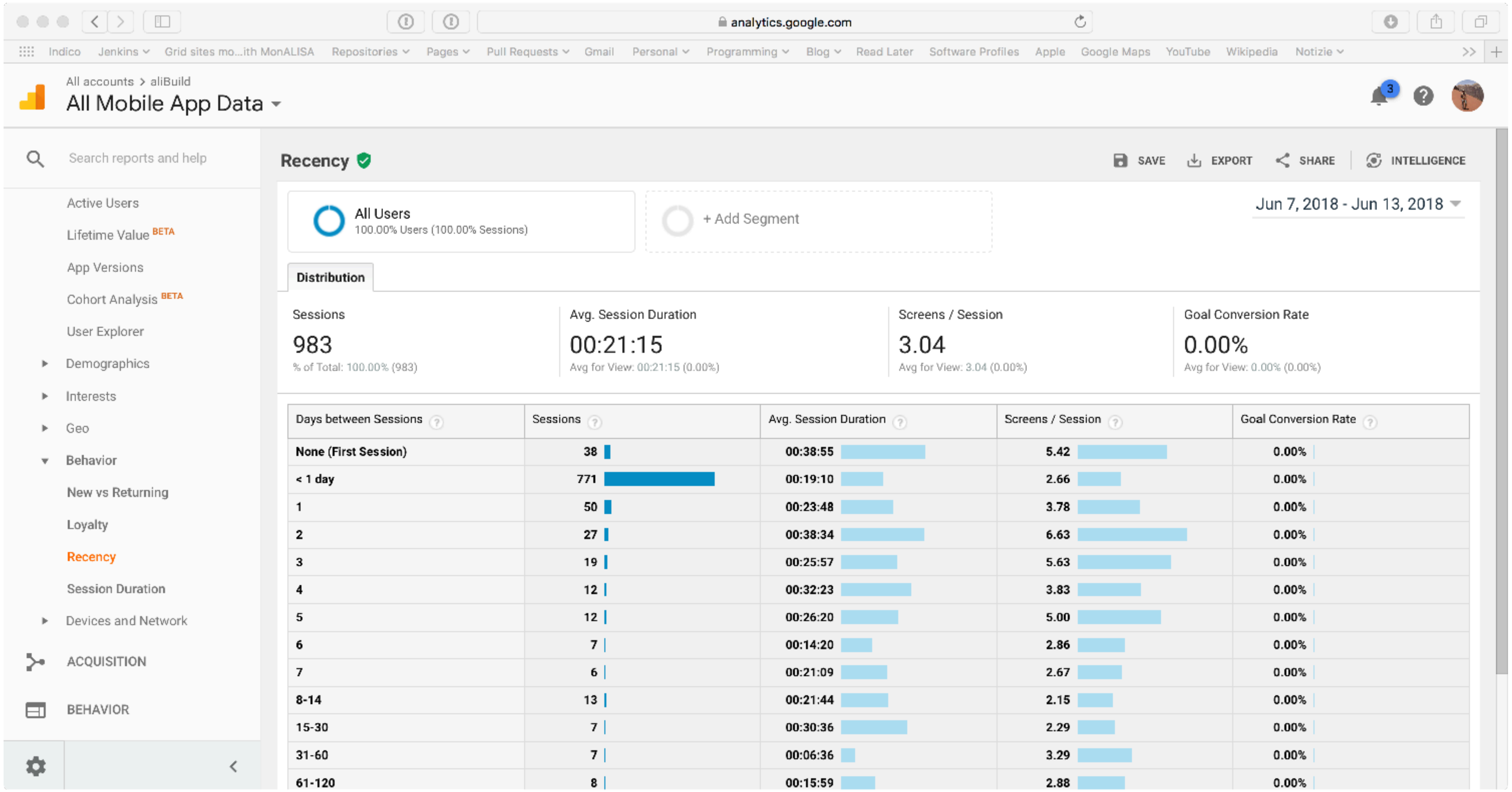
Very often profilers have powerful tools to visualise a single profile. Rarely they can be used to compare two reports. Even more rarely this is provided as web based multiuser service.

Question

If we really managed to agree on how to store the data, why don't we work together also on the presentation layer for it?

VISION

Like Google Analytics, but for profile data



Recency

All Users
100.00% Users (100.00% Sessions)

+ Add Segment

Jun 7, 2018 - Jun 13, 2018

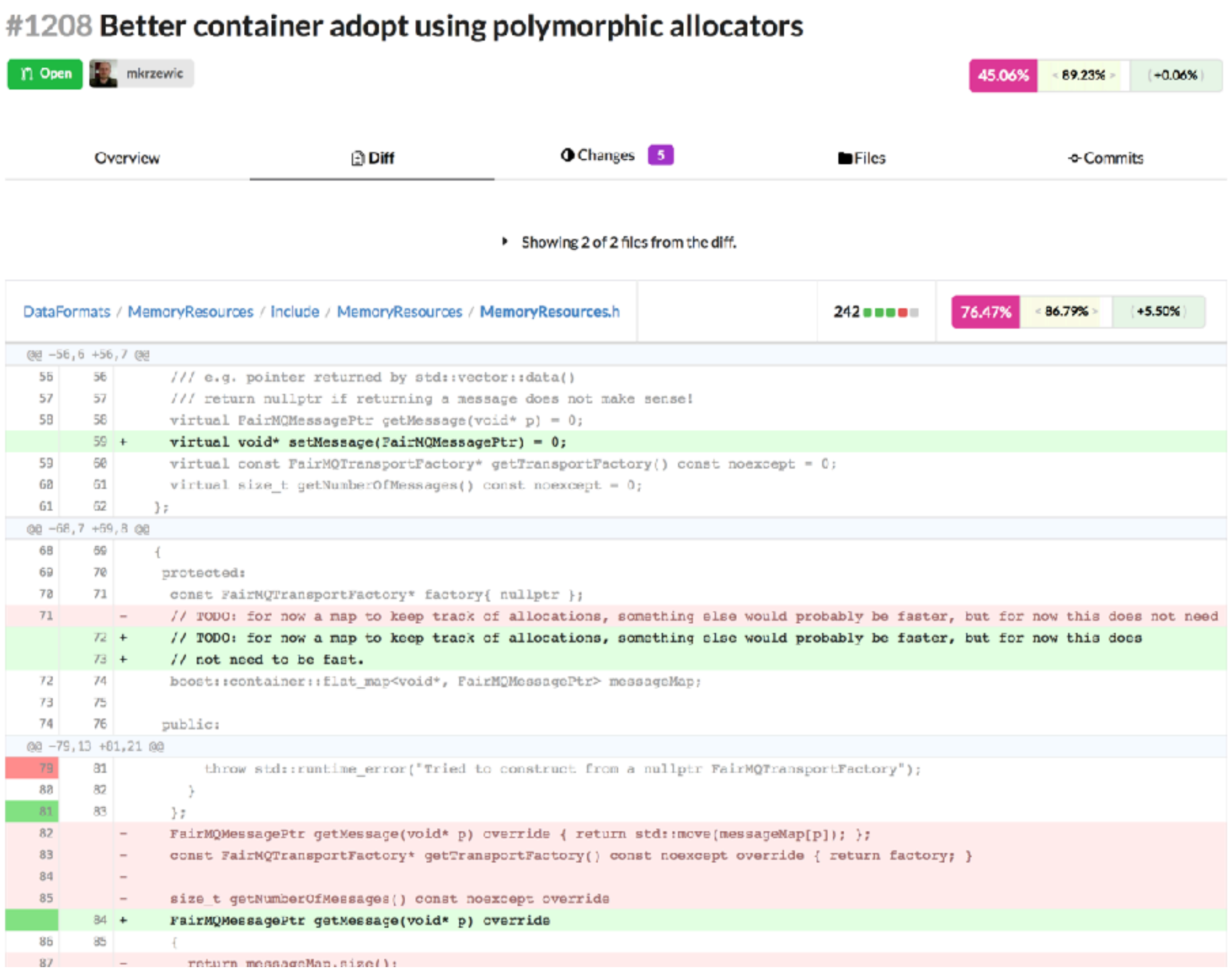
Distribution

Sessions	Avg. Session Duration	Screens / Session	Goal Conversion Rate
983	00:21:15	3.04	0.00%
% of Total: 100.00% (983)	Avg for View: 00:21:15 (0.00%)	Avg for View: 3.04 (0.00%)	Avg for View: 0.00% (0.00%)

Days between Sessions	Sessions	Avg. Session Duration	Screens / Session	Goal Conversion Rate
None (First Session)	38	00:38:55	5.42	0.00%
< 1 day	771	00:19:10	2.66	0.00%
1	50	00:23:48	3.78	0.00%
2	27	00:38:34	6.63	0.00%
3	19	00:25:57	5.63	0.00%
4	12	00:32:23	3.83	0.00%
5	12	00:26:20	5.00	0.00%
6	7	00:14:20	2.86	0.00%
7	6	00:21:09	2.67	0.00%
8-14	13	00:21:44	2.15	0.00%
15-30	7	00:30:36	2.29	0.00%
31-60	7	00:06:36	3.29	0.00%
61-120	8	00:15:59	2.88	0.00%

VISION

Like CodeCov, but for profile data



#1208 **Better container adopt using polymorphic allocators**

Showing 2 of 2 files from the diff.

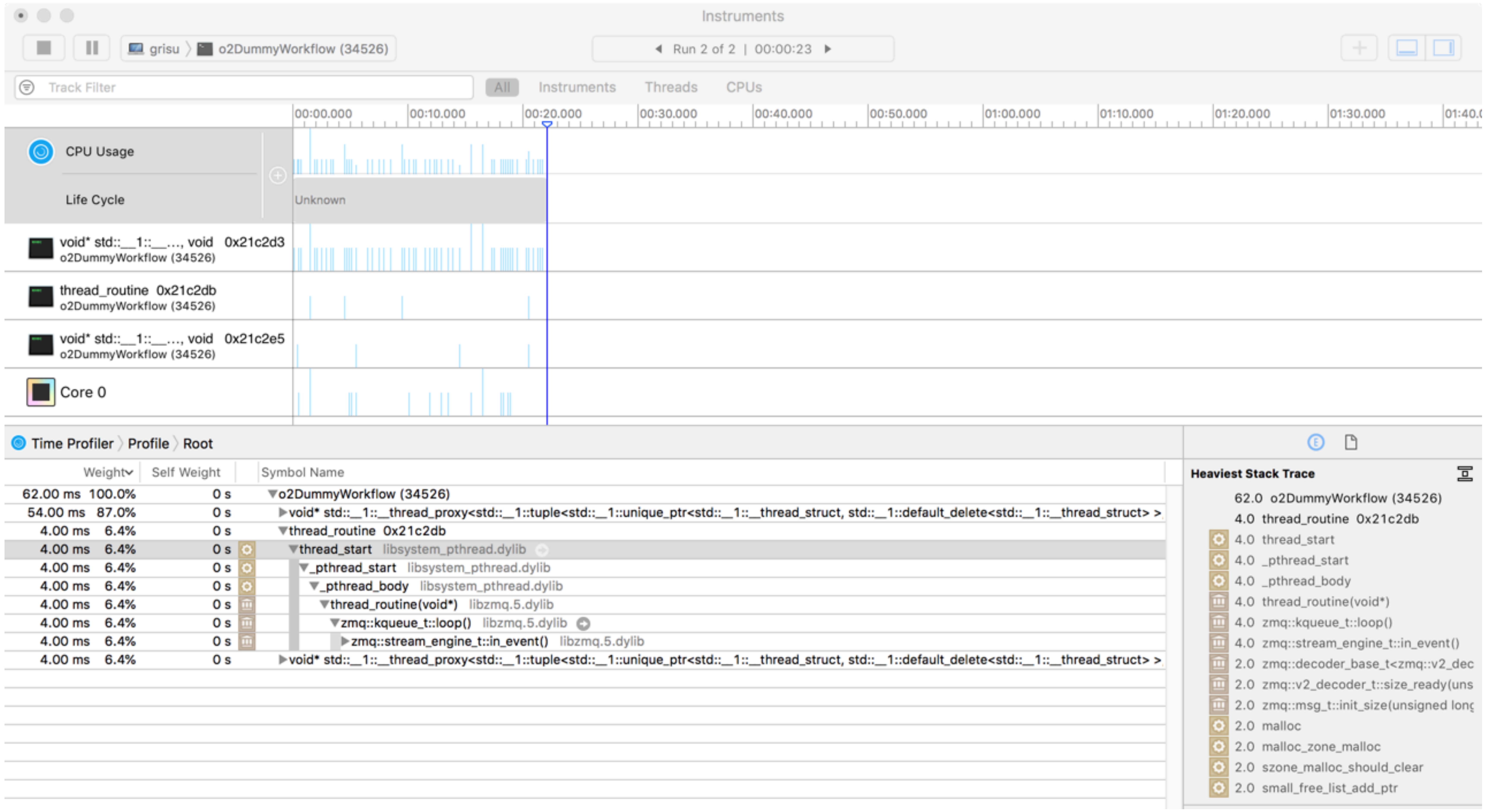
DataFormats / MemoryResources / Include / MemoryResources / MemoryResources.h

```
@@ -56,6 +56,7 @@
      /// e.g. pointer returned by std::vector::data()
      /// return nullptr if returning a message does not make sense!
      virtual FairMQMessagePtr getMessage(void* p) = 0;
+     virtual void* setMessage(FairMQMessagePtr) = 0;
      virtual const FairMQTransportFactory* getTransportFactory() const noexcept = 0;
      virtual size_t getNumberOfMessages() const noexcept = 0;
    };
@@ -68,7 +69,8 @@
    {
     protected:
      const FairMQTransportFactory* factory{ nullptr };
-     // TODO: for now a map to keep track of allocations, something else would probably be faster, but for now this does not need
+     // TODO: for now a map to keep track of allocations, something else would probably be faster, but for now this does
+     // not need to be fast.
      boost::container::flat_map<void*, FairMQMessagePtr> messageMap;

     public:
@@ -79,13 +81,21 @@
          throw std::runtime_error("Tried to construct from a nullptr FairMQTransportFactory");
        }
      };
-     FairMQMessagePtr getMessage(void* p) override { return std::move(messageMap[p]); };
-     const FairMQTransportFactory* getTransportFactory() const noexcept override { return factory; }
-
-     size_t getNumberOfMessages() const noexcept override
+     FairMQMessagePtr getMessage(void* p) override
      {
-       return messageMap.size();
```

VISION

Like Instruments, but for the web, OpenSource and not limited to OSX.



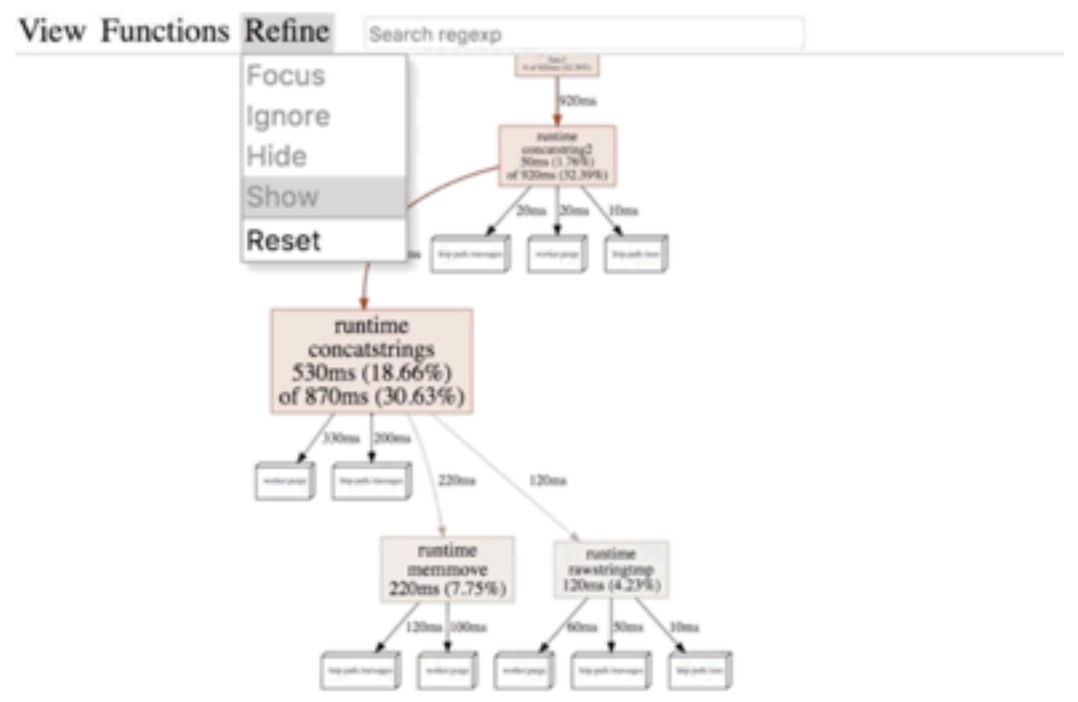
```
   Weight        Self Weight   Symbol Name
62.00 ms  100.0%     0 s      ▼o2DummyWorkflow (34526)
54.00 ms   87.0%     0 s       ▶void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct> >
 4.00 ms    6.4%     0 s       ▼thread_routine 0x21c2db
 4.00 ms    6.4%     0 s        ▼thread_start  libsystem_pthread.dylib
 4.00 ms    6.4%     0 s         ▼_pthread_start  libsystem_pthread.dylib
 4.00 ms    6.4%     0 s          ▼_pthread_body  libsystem_pthread.dylib
 4.00 ms    6.4%     0 s           ▼thread_routine(void*)  libzmq.5.dylib
 4.00 ms    6.4%     0 s            ▼zmq::kqueue_t::loop()  libzmq.5.dylib
 4.00 ms    6.4%     0 s             ▶zmq::stream_engine_t::in_event()  libzmq.5.dylib
 4.00 ms    6.4%     0 s       ▶void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct> >
```

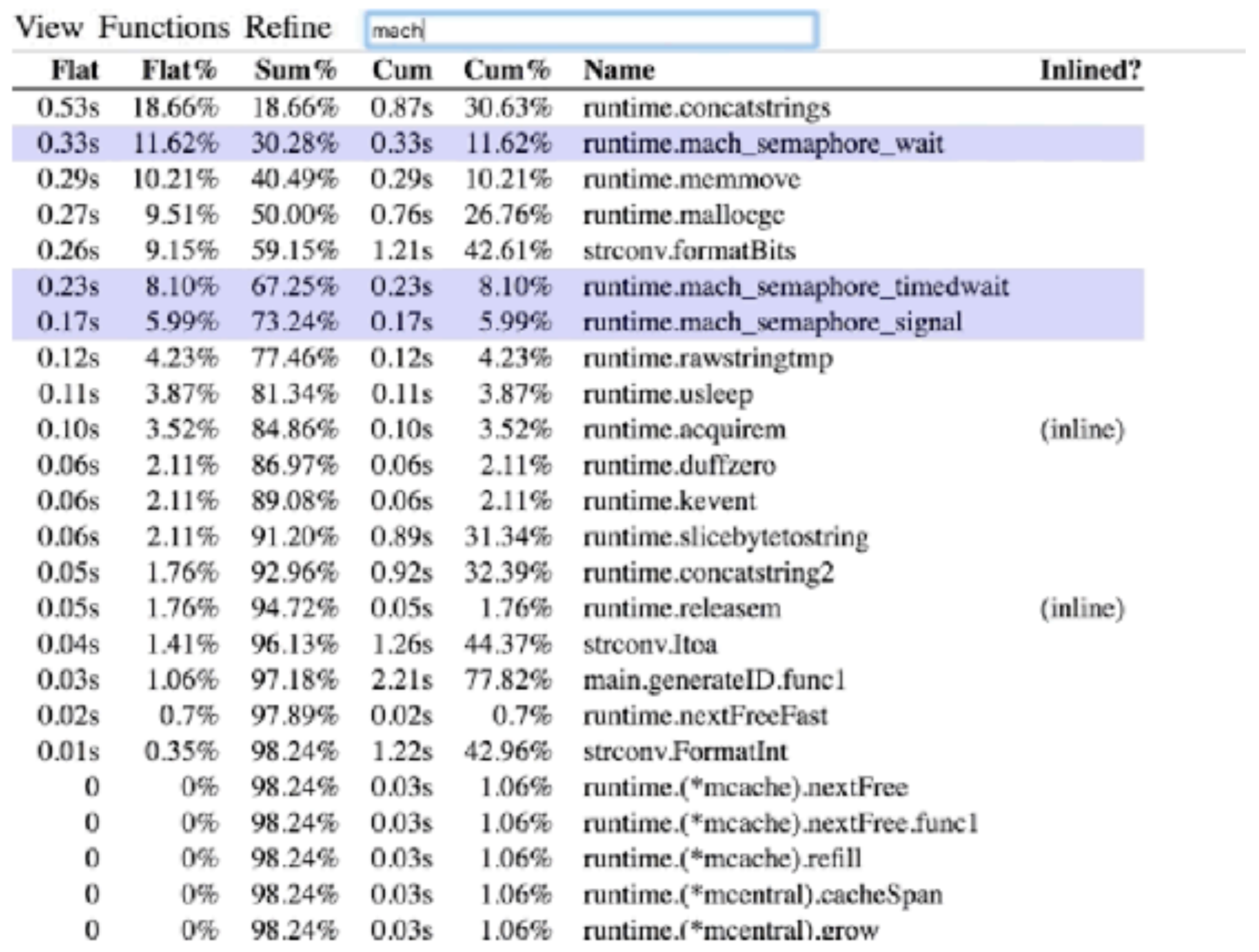
Heaviest Stack Trace

```
62.0  o2DummyWorkflow (34526)
 4.0  thread_routine  0x21c2db
 4.0  thread_start
 4.0  _pthread_start
 4.0  _pthread_body
 4.0  thread_routine(void*)
 4.0  zmq::kqueue_t::loop()
 4.0  zmq::stream_engine_t::in_event()
 2.0  zmq::decoder_base_t<zmq::v2_dec
 2.0  zmq::v2_decoder_t::size_ready(uns
 2.0  zmq::msg_t::init_size(unsigned lonc
 2.0  malloc
 2.0  malloc_zone_malloc
 2.0  szone_malloc_should_clear
 2.0  small_free_list_add_ptr
```

VISION

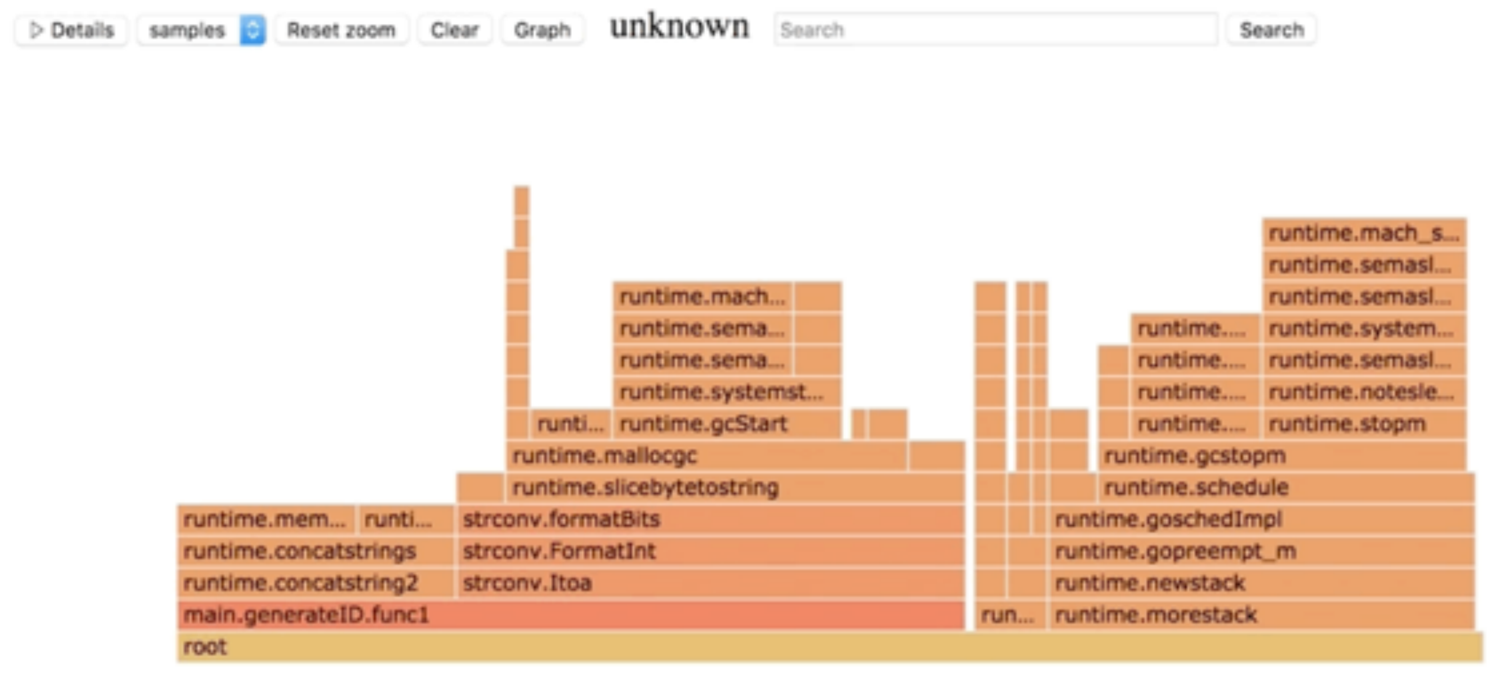
Maybe contribute to existing projects?


View Functions Refine
Search regexp
Focus
Ignore
Hide
Show
Reset
runtime concatstrings 530ms (18.66%) of 870ms (30.63%)
runtime memmove 220ms (7.75%)
runtime rawstringtmp 120ms (4.23%)



View Functions Refine mach

Flat	Flat%	Sum%	Cum	Cum%	Name	Inlined?
0.53s	18.66%	18.66%	0.87s	30.63%	runtime.concatstrings	
0.33s	11.62%	30.28%	0.33s	11.62%	runtime.mach_semaphore_wait	
0.29s	10.21%	40.49%	0.29s	10.21%	runtime.memmove	
0.27s	9.51%	50.00%	0.76s	26.76%	runtime.mallocgc	
0.26s	9.15%	59.15%	1.21s	42.61%	strconv.formatBits	
0.23s	8.10%	67.25%	0.23s	8.10%	runtime.mach_semaphore_timedwait	
0.17s	5.99%	73.24%	0.17s	5.99%	runtime.mach_semaphore_signal	
0.12s	4.23%	77.46%	0.12s	4.23%	runtime.rawstringtmp	
0.11s	3.87%	81.34%	0.11s	3.87%	runtime.usleep	
0.10s	3.52%	84.86%	0.10s	3.52%	runtime.acquirem	(inline)
0.06s	2.11%	86.97%	0.06s	2.11%	runtime.duffzero	
0.06s	2.11%	89.08%	0.06s	2.11%	runtime.kevent	
0.06s	2.11%	91.20%	0.89s	31.34%	runtime.slicebytetostring	
0.05s	1.76%	92.96%	0.92s	32.39%	runtime.concatstring2	
0.05s	1.76%	94.72%	0.05s	1.76%	runtime.releasem	(inline)
0.04s	1.41%	96.13%	1.26s	44.37%	strconv.Itoa	
0.03s	1.06%	97.18%	2.21s	77.82%	main.generateID.func1	
0.02s	0.7%	97.89%	0.02s	0.7%	runtime.nextFreeFast	
0.01s	0.35%	98.24%	1.22s	42.96%	strconv.FormatInt	
0	0%	98.24%	0.03s	1.06%	runtime.(*mcache).nextFree	
0	0%	98.24%	0.03s	1.06%	runtime.(*mcache).nextFree.func1	
0	0%	98.24%	0.03s	1.06%	runtime.(*mcache).refill	
0	0%	98.24%	0.03s	1.06%	runtime.(*mcentral).cacheSpan	
0	0%	98.24%	0.03s	1.06%	runtime.(*mcentral).grow	

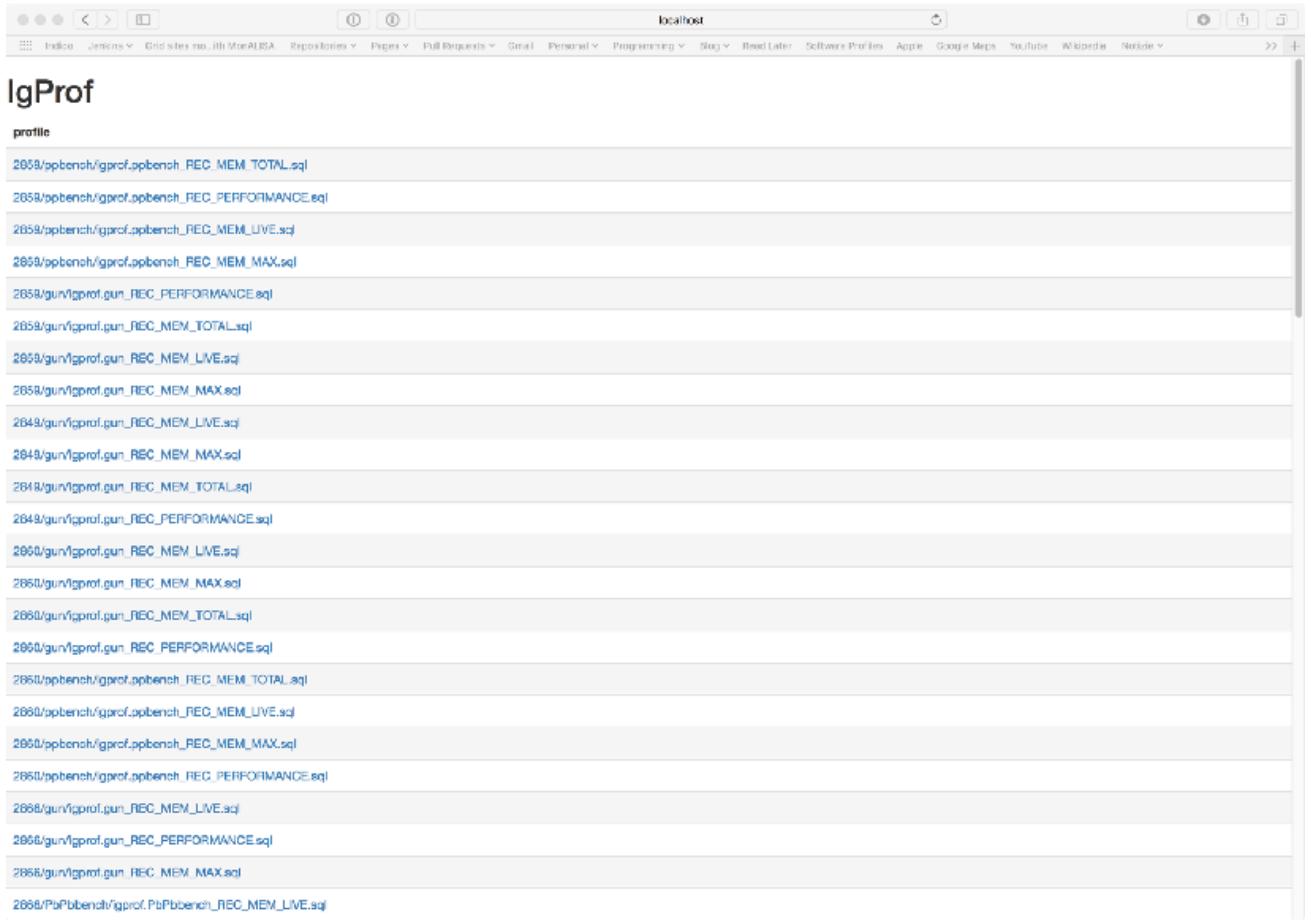


Of course as much as we like writing cool GUIs we should try to reuse opensource components.

E.g. catapult: https://github.com/catapult-project/catapult

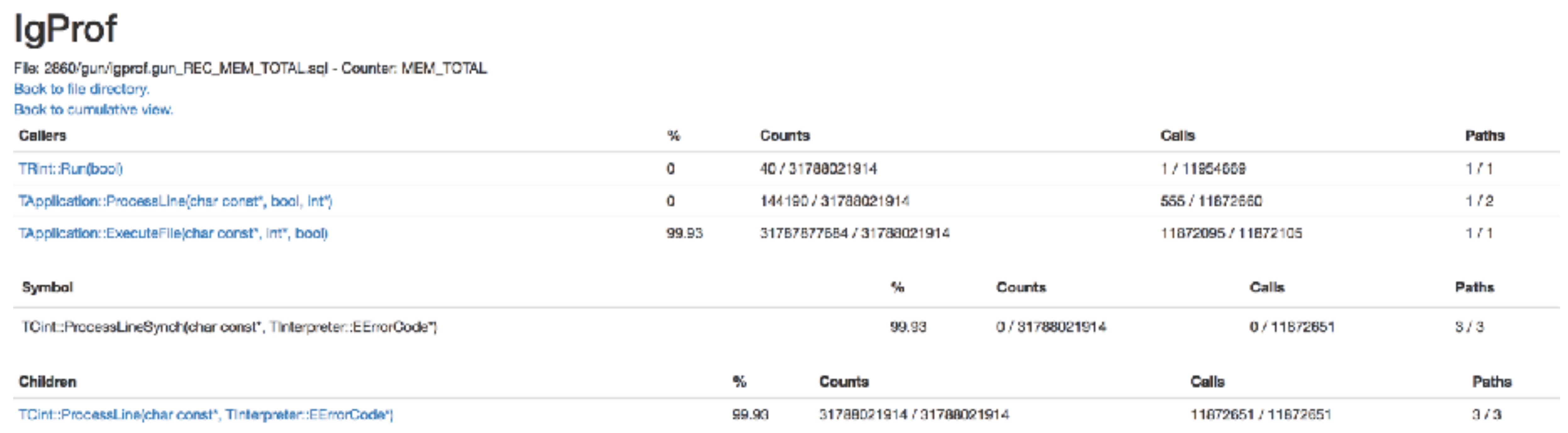
UNREALISED VISION

IgProf.io: nodejs + React rewrite of the IgProf GUI. Still very far from being a realised vision, but maybe a viable starting point?



IgProf

profile
2858/ppbench/igprof.ppbench_REC_MEM_TOTAL.sql
2858/ppbench/igprof.ppbench_REC_PERFORMANCE.sql
2858/ppbench/igprof.ppbench_REC_MEM_LIVE.sql
2858/ppbench/igprof.ppbench_REC_MEM_MAX.sql
2858/gun/igprof.gun_REC_PERFORMANCE.sql
2858/gun/igprof.gun_REC_MEM_TOTAL.sql
2858/gun/igprof.gun_REC_MEM_LIVE.sql
2858/gun/igprof.gun_REC_MEM_MAX.sql
2848/gun/igprof.gun_REC_MEM_LIVE.sql
2848/gun/igprof.gun_REC_MEM_MAX.sql
2848/gun/igprof.gun_REC_MEM_TOTAL.sql
2848/gun/igprof.gun_REC_PERFORMANCE.sql
2860/gun/igprof.gun_REC_MEM_LIVE.sql
2860/gun/igprof.gun_REC_MEM_MAX.sql
2860/gun/igprof.gun_REC_MEM_TOTAL.sql
2860/gun/igprof.gun_REC_PERFORMANCE.sql
2860/ppbench/igprof.ppbench_REC_MEM_TOTAL.sql
2860/ppbench/igprof.ppbench_REC_MEM_LIVE.sql
2860/ppbench/igprof.ppbench_REC_MEM_MAX.sql
2860/ppbench/igprof.ppbench_REC_PERFORMANCE.sql
2868/gun/igprof.gun_REC_MEM_LIVE.sql
2868/gun/igprof.gun_REC_PERFORMANCE.sql
2868/gun/igprof.gun_REC_MEM_MAX.sql
2868/PbPbbench/igprof.PbPbbench_REC_MEM_LIVE.sql



IgProf

File: 2860/gun/igprof.gun_REC_MEM_TOTAL.sql - Counter: MEM_TOTAL
Back to file directory.
Back to cumulative view.

Callers	%	Counts	Calls	Paths
TRint::Run(bool)	0	40 / 31788021914	1 / 11954609	1 / 1
TApplication::ProcessLine(char const*, bool, int*)	0	144190 / 31788021914	555 / 11872660	1 / 2
TApplication::ExecuteFile(char const*, int*, bool)	99.93	31757877584 / 31788021914	11872095 / 11872105	1 / 1

Symbol	%	Counts	Calls	Paths
TCint::ProcessLineSynch(char const*, TInterpreter::EErrorCode*)	99.93	0 / 31788021914	0 / 11872651	3 / 3

Children	%	Counts	Calls	Paths
TCint::ProcessLine(char const*, TInterpreter::EErrorCode*)	99.93	31788021914 / 31788021914	11872651 / 11872651	3 / 3

IgProf at FOSDEM 2015: https://archive.fosdem.org/2015/schedule/event/igprof_the_ignominous_profiler/