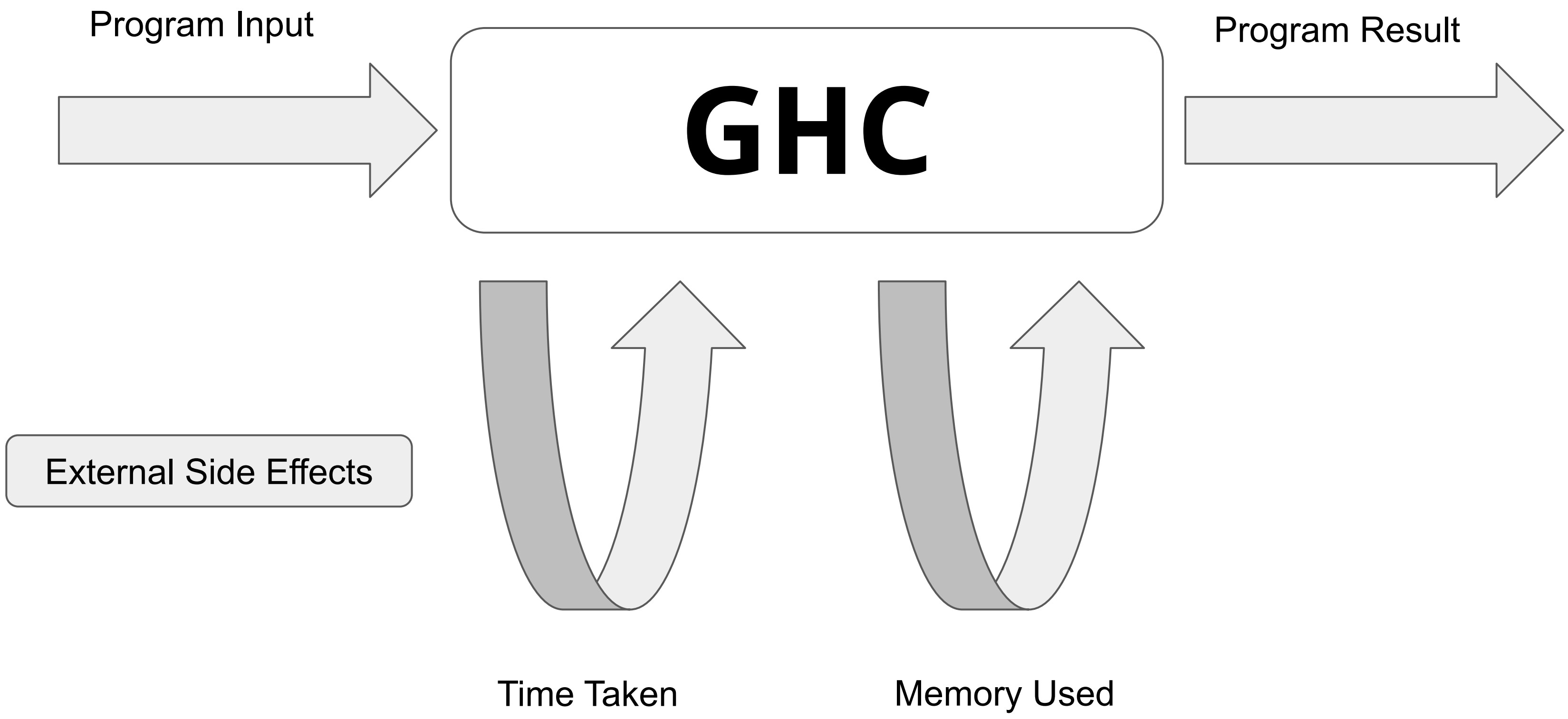
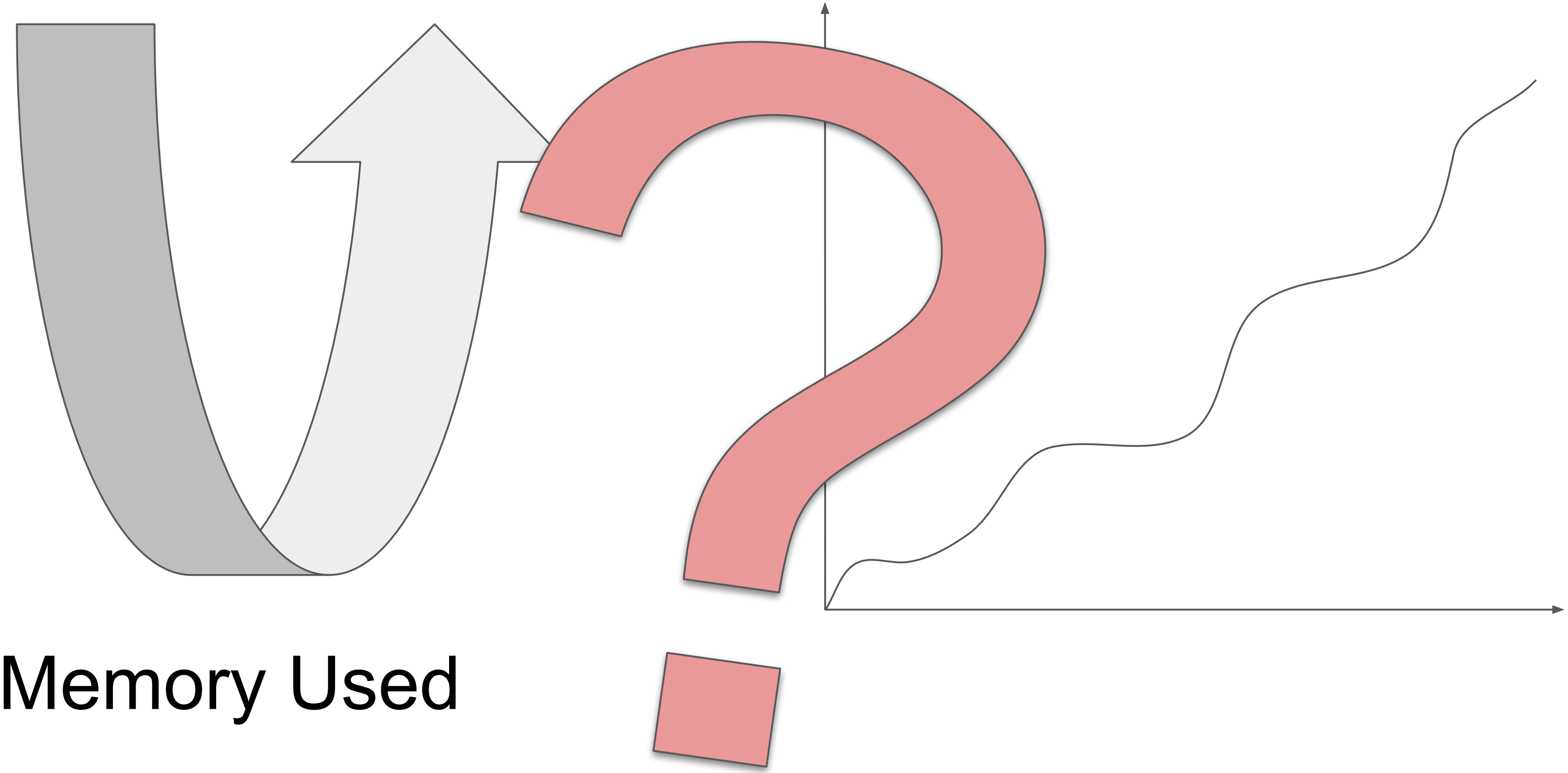
Gazing into the void

Understanding Space (Leaks)

Program Input
Category Theory
GHC
Program Result
()



Program Input
GHC
Program Result
External Side Effects
Time Taken
Memory Used



Memory Used

1. General space profiles to get an idea for the problem.
2. Use precise techniques to fix the problem.

1

General space profiles to get an idea for the problem.

2

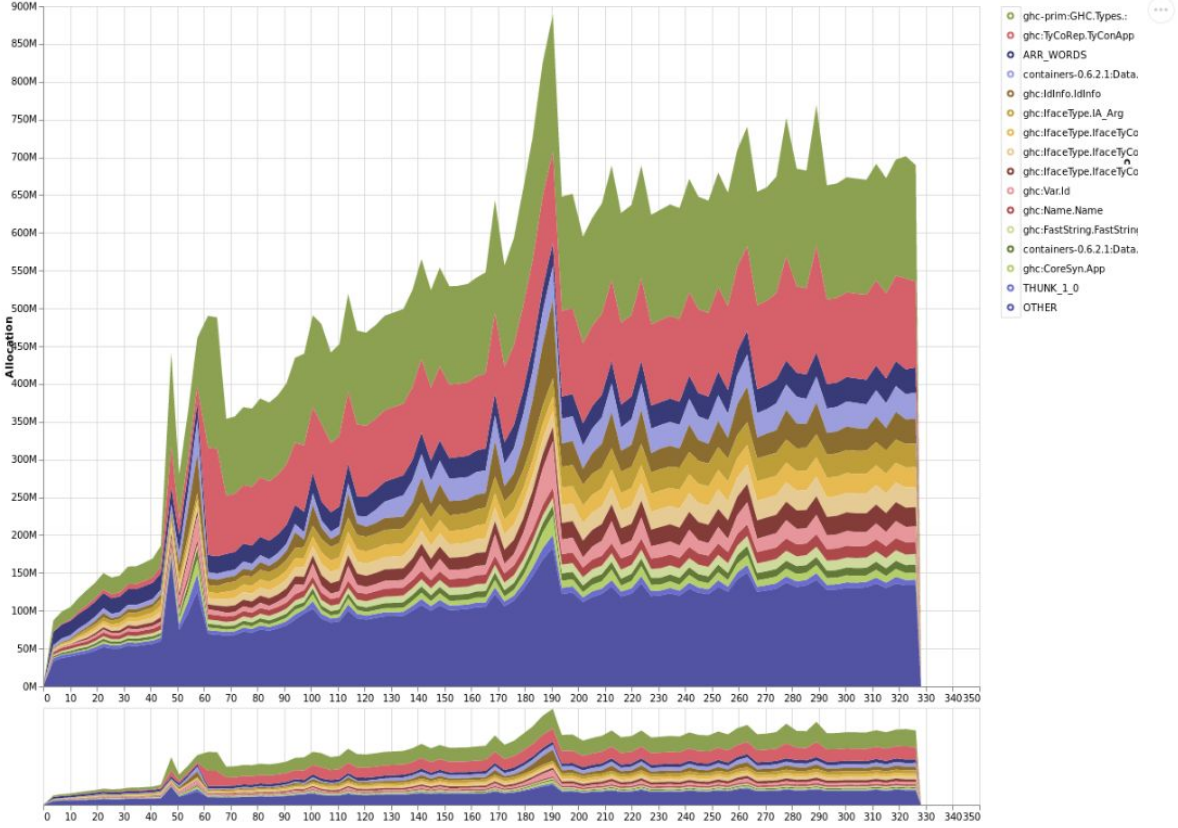
Use precise techniques to fix the problem.

eventlog2html

Options: `./Profile +RTS -hT -l -i2`

Created at: `2019-09-30, 15:03 UTC`

Area Chart | Normalized | Streamgraph | Linechart



Using ghc

1. `ghc … -prof`

2. `./Main +RTS -h`

Using cabal

1. profiling: True

2. cabal new-run exe -- +RTS -h

Profiling GHC, using a simple GHC API application

```haskell
module Main where
initGhcM :: [String] -> Ghc ()
initGhcM xs = do
    df1 <- getSessionDynFlags
    let cmdOpts = ["-fforce-recomp"] ++ xs
    (df2, leftovers, warns) <- G.parseDynamicFlags df1 (map G.noLoc cmdOpts)
    setSessionDynFlags df2
    ts <- mapM (flip G.guessTarget Nothing) $ map unLoc leftovers

    setTargets ts

    void $ G.load LoadAllTargets

main :: IO ()
main = do
    xs <- words <$> readFile "args"
    let libdir = "/home/matt/ghc/root-prof-mode/stage1/lib"
    runGhc (Just libdir) $ initGhcM xs
```

Set up environment

Compile project

```
cabal v2-repl -w /path/to/ghc Cabal -v3
```

- Building a simple application is much easier than building a profiled GHC executable from scratch
- `cabal` deals with building dependencies with profiling

Go to other slide deck

-hT	CLOSURE TYPE	General Idea
-hy	HASKELL TYPE	Bigger Buckets
-hb	LIFECYLE	Used, or not?
-hc	ALLOCATION SITE	Where?

Knowledge Gained From Profiling

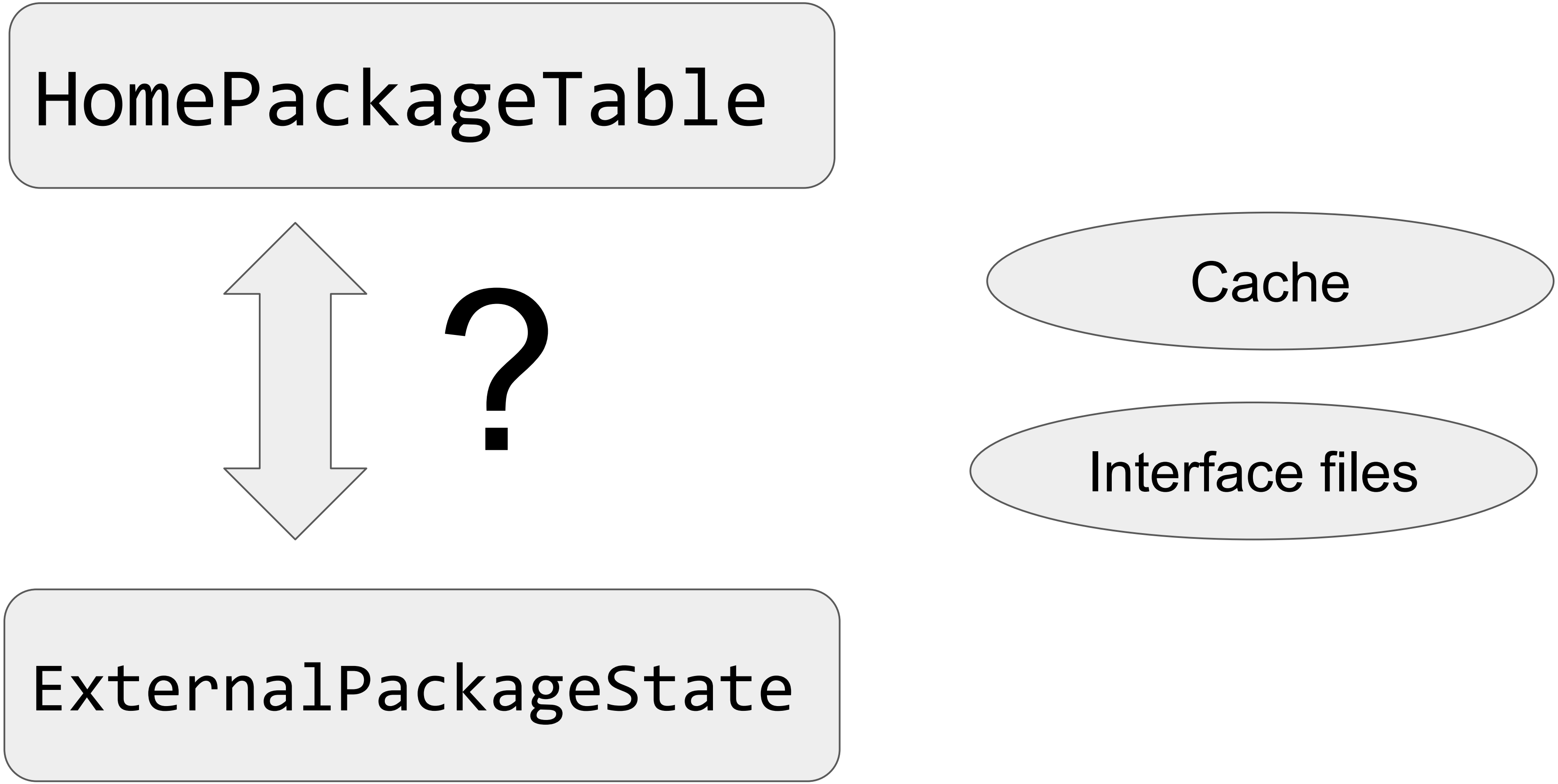
-hy

Most Type allocations are TyConApp

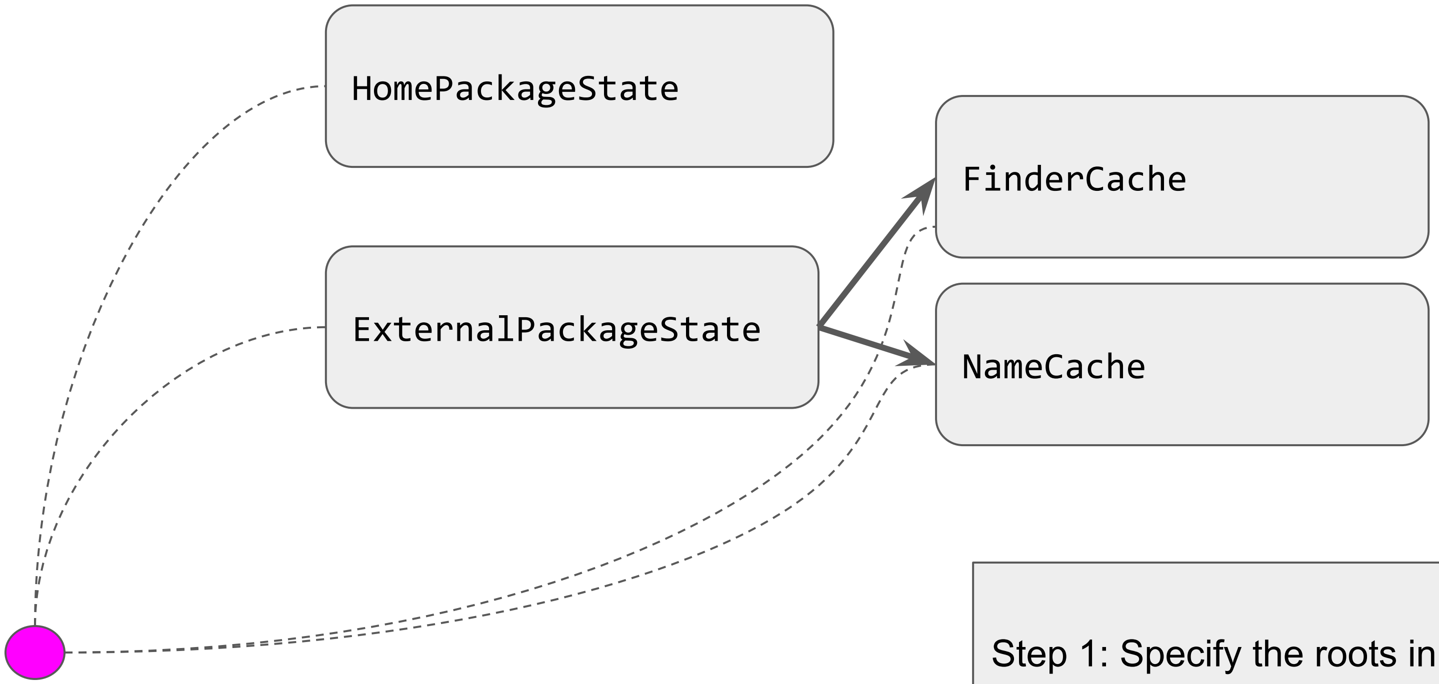
Domain Knowledge

-hc

Unused allocation comes from interface creation functions


HomePackageTable
?
ExternalPackageState
Cache
Interface files

Profiling by user specified roots

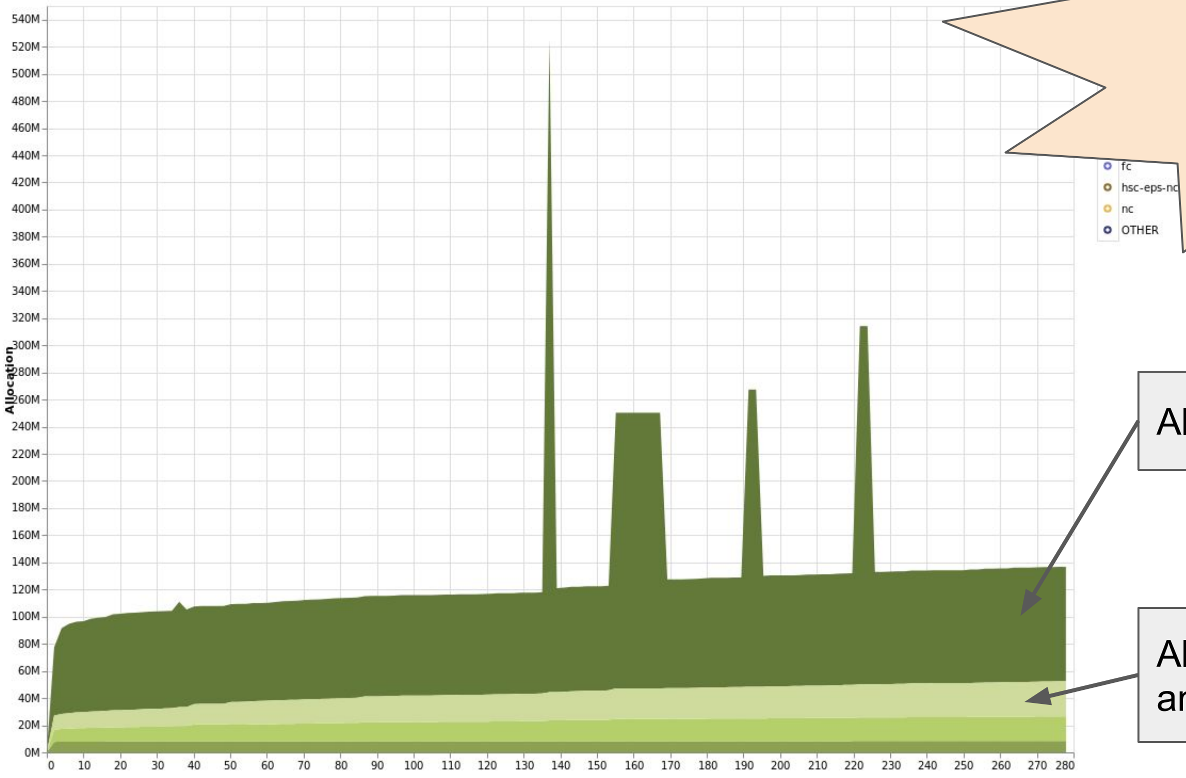


Step 1: Specify the roots in your program

Step 2: Run the profiler

+RTS -ho

Coming in 8.10?



Allocation in EPS

Allocation in NameCache and FinderCache

Potential Solution: Unload `ExternalPackageState`

`ExternalPackageState`

`ExternalPackageState`

Compact Region

- Not traversed by GC
- Can be serialised and deserialised quickly

Problem with Profiles

Good to get a general idea of what's happening

Hard to know what precisely is happening

Have to keep rerunning the program to learn different information

1

General space profiles to get an idea for the problem.

2

Use precise techniques to fix the problem.


@mpickering_ghcdev
The Heap
G
D
B

Our Precise Question

Why are there so many TyConApp constructors allocated?

1. `info functions TyConApp_con_info` — Get the address of TyConApp
2. `python lcs=listcl(<addr>)` — Find all TyConApp closures
3. `python res=payload_args(lcs)` — Get address of first argument

Result is a list of pointers to TyCons

1

```
info functions TyConApp_con_info
```

2

```
python lcs=listcl(<addr>)
```

3

```
python res=payload_args(lcs)
```

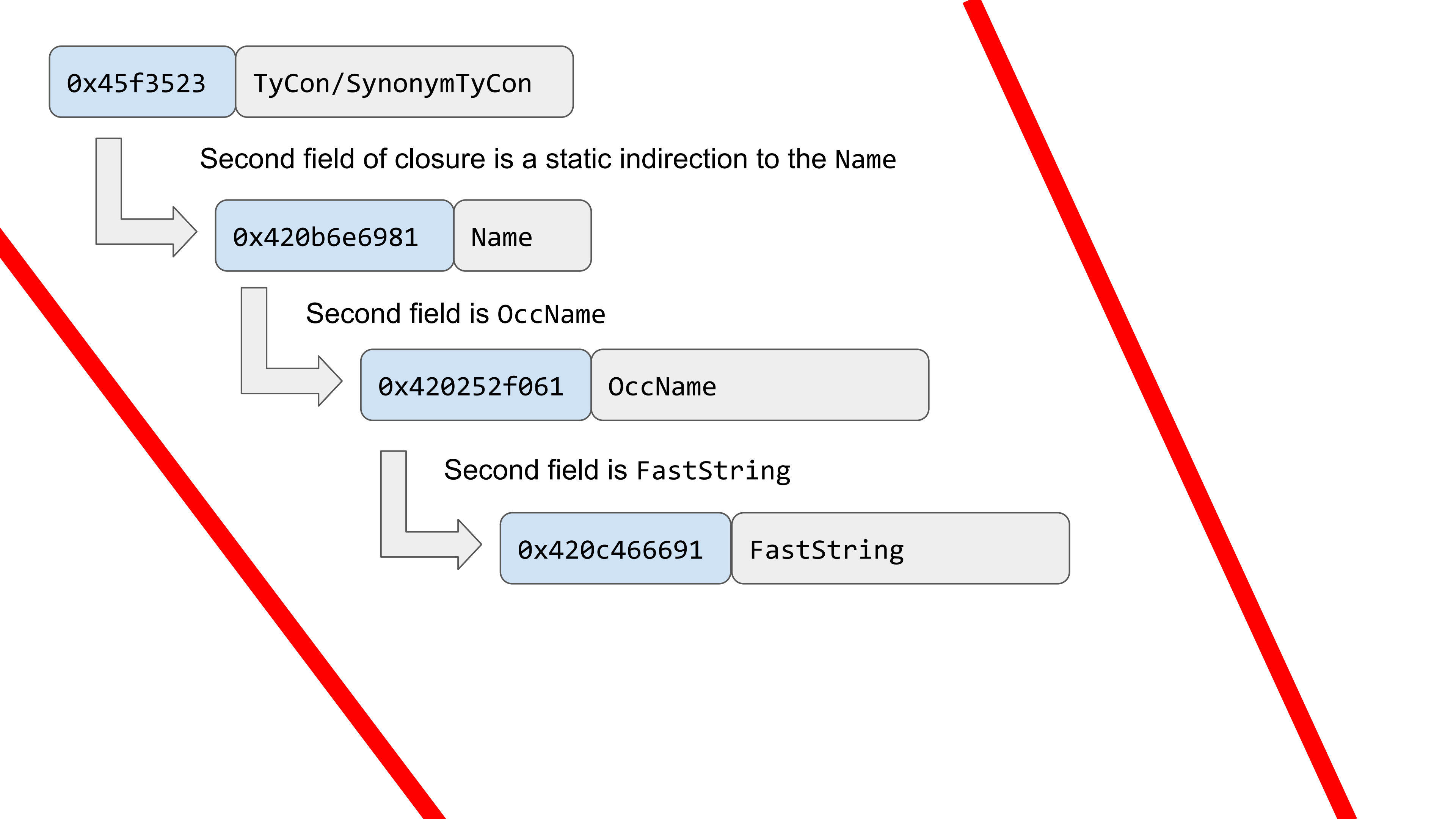
https://gitlab.haskell.org/bgamari/ghc-utils/tree/master/gdb

Result is a list of pointers to TyCons

Address of TyCon	Number of TyConApp
0x45f3523	28732
0x420b840702	9629
0x42055b7e46	9596
0x420559b582	9511
0x420bb15a1e	9509
0x420b86c6ba	9501
0x42055bac1e	9496
0x45e68fd	538
…..	

Can we reduce any of these numbers?

Address of TyCon	Number of TyConApp	Name of TyCon?
0x45f3523	28732	???
0x420b840702	9629	???
0x42055b7e46	9596	???
0x420559b582	9511	???
0x420bb15a1e	9509	???
0x420b86c6ba	9501	???
0x42055bac1e	9496	???
0x45e68fd	538	???
…..		


0x45f3523
TyCon/SynonymTyCon
Second field of closure is a static indirection to the Name
0x420b6e6981
Name
Second field is OccName
0x420252f061
OccName
Second field is FastString
0x420c466691
FastString

Second field is `FastString`

`0x420c466691` `FastString`

Unpacked ByteString

Info	PlainPtr	FastZString	Unique	Length	Addr#	Offset	Length
-	-	-	-	-	0x420011 a010	0x354b	0x4

`Calculate Offset` `0x420011a010 + 0x354b = 0x420011d558`

`Contents 0x420011d558` `0x65707954040000`

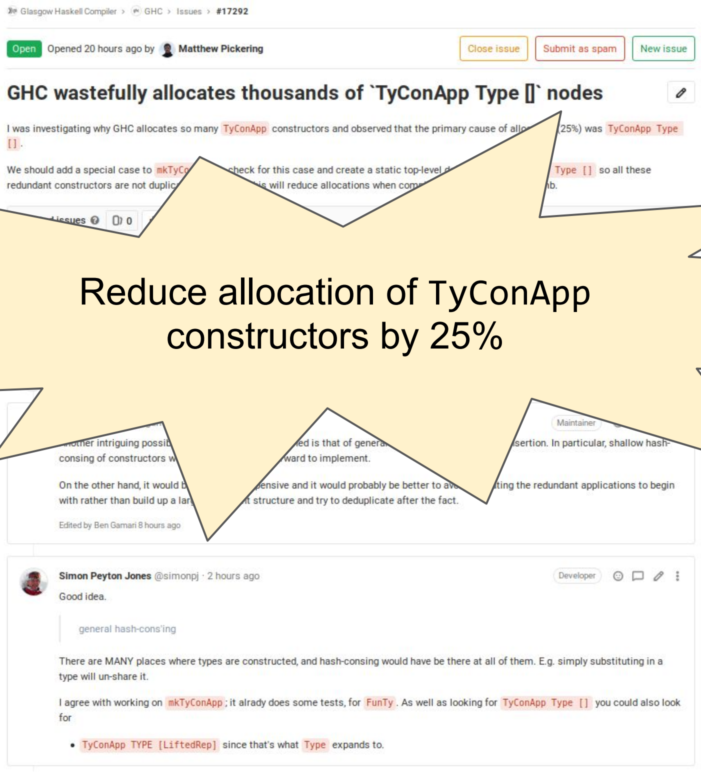
-	-	-	-	-	0x420011a010	0x354b	0x4

Calculate Offset	0x420011a010 + 0x354b = 0x420011d558
Contents 0x420011d558	0x6570795404000

```
type Type = TYPE LiftedKind
```

```
type Type = TYPE LiftedKind
```

No arguments so all occurrences are identical


Glasgow Haskell Compiler > GHC > Issues > #17292

Open Opened 20 hours ago by Matthew Pickering Close issue Submit as spam New issue

GHC wastefully allocates thousands of `TyConApp Type []` nodes

I was investigating why GHC allocates so many TyConApp constructors and observed that the primary cause of allo… (25%) was TyConApp Type [] .

We should add a special case to mkTyCo… check for this case and create a static top-level … Type [] so all these redundant constructors are not duplic… is will reduce allocations when comp… b.

…issues 0

Maintainer

…other intriguing possib… ed is that of genera… sertion. In particular, shallow hash-consing of constructors w… ward to implement.

On the other hand, it would b… ensive and it would probably be better to av… ting the redundant applications to begin with rather than build up a lar… t structure and try to deduplicate after the fact.

Edited by Ben Gamari 8 hours ago

Simon Peyton Jones @simonpj · 2 hours ago Developer

Good idea.

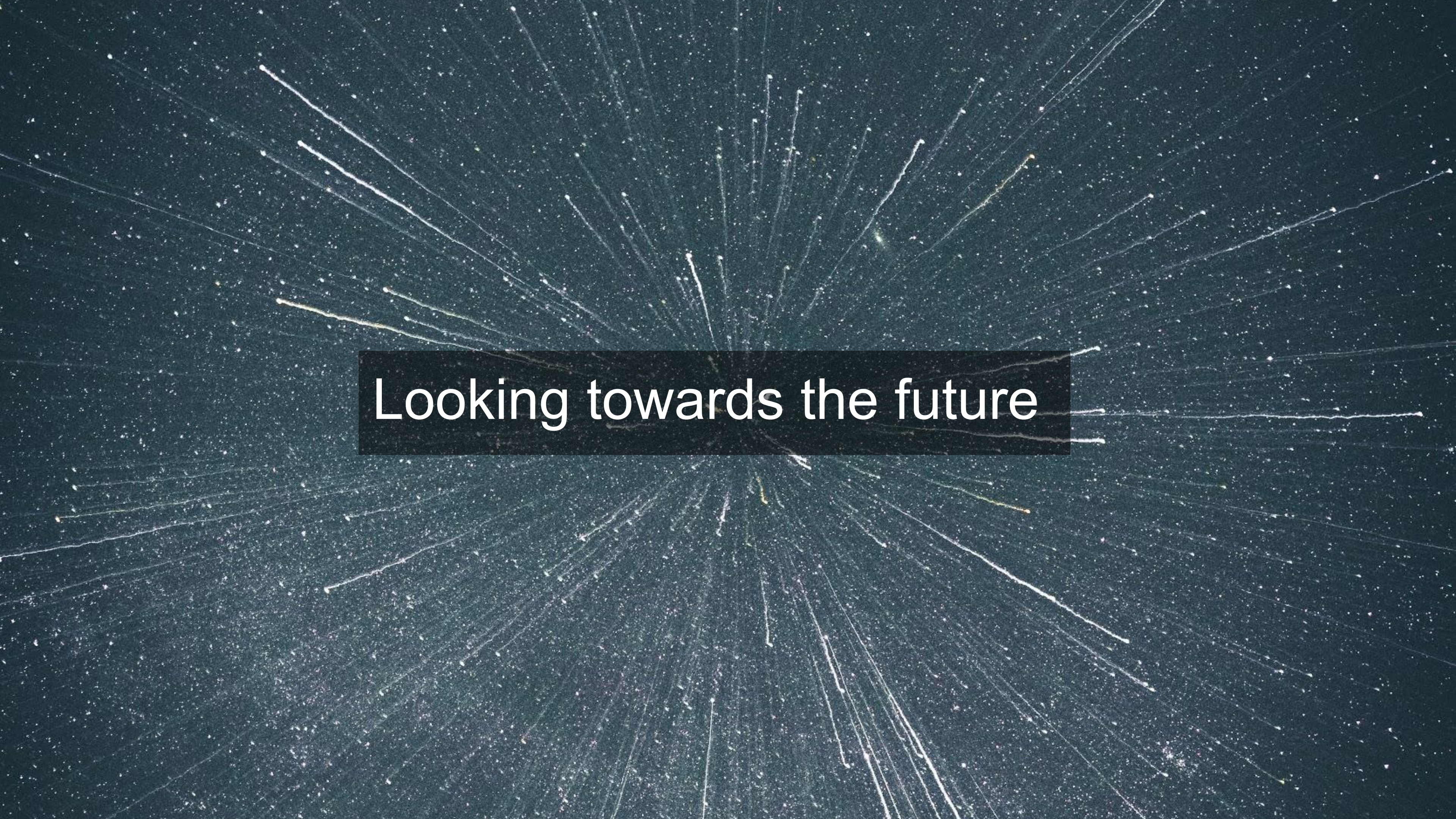
> general hash-cons'ing

There are MANY places where types are constructed, and hash-consing would have be there at all of them. E.g. simply substituting in a type will un-share it.

I agree with working on mkTyConApp ; it alrady does some tests, for FunTy . As well as looking for TyConApp Type [] you could also look for

- TyConApp TYPE [LiftedRep] since that's what Type expands to.

Reduce allocation of `TyConApp` constructors by 25%



Looking towards the future

ghc-debug

DEBUGGER

Connection via socket

PROGRAM

Debug the heap by writing a Haskell program

Weak Pointer Techniques

Simon Marlow

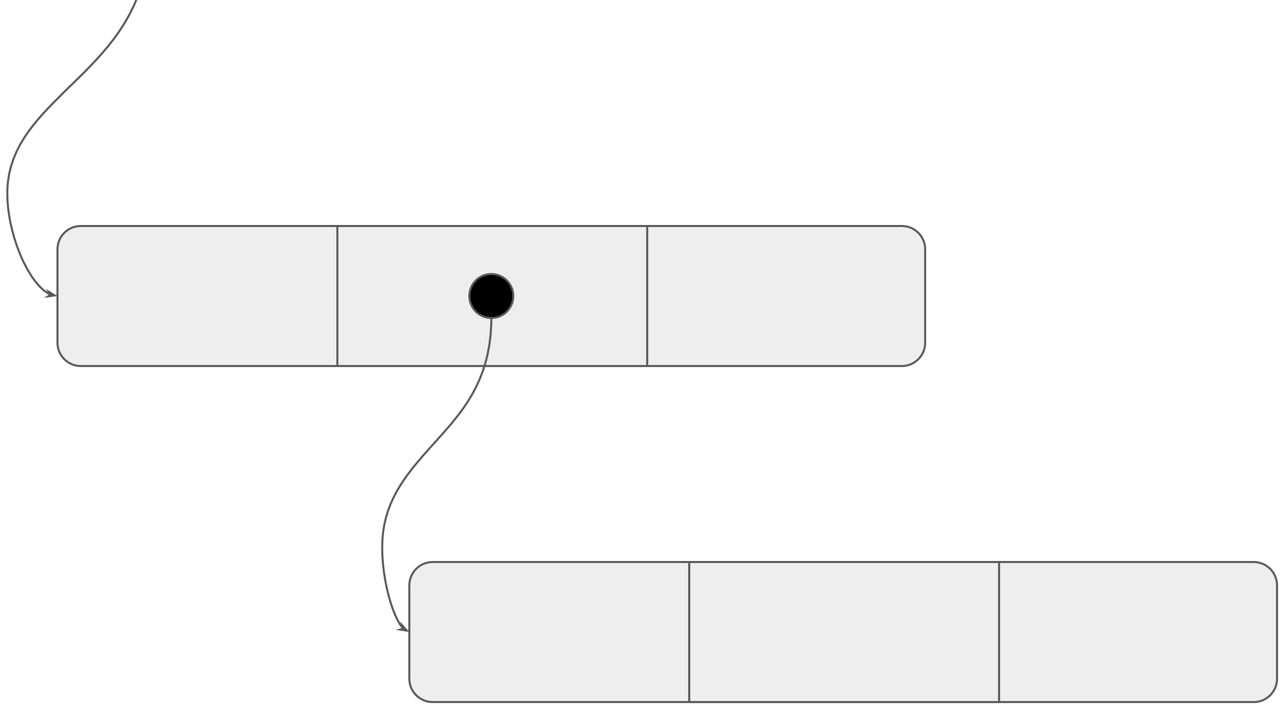
Blog | Book | Publications | Talks

Fixing 17 space leaks in GHCi, and keeping them fixed

June 20, 2018

In this post I want to tackle a couple of problems that have irritated me from time to time when working with Haskell.

- **GHC provides some powerful tools for debugging space leaks, but sometimes they're not enough**. The heap profiler shows you what's in the heap, but it doesn't provide detailed visibility into the chain of references that cause a particular data structure to be retained. Retainer profiling was supposed to help with this, but in practice it's pretty hard to extract the signal you need - retainer profiling will show you one relationship at a time, but you want to see the whole chain of references.
- **Once you've fixed a space leak, how can you write a regression test for it**? Sometimes you can make a test case that will use `O(n)` memory if it leaks instead of `O(1)`, and then it's straightforward. But what if your leak is only a constant factor?

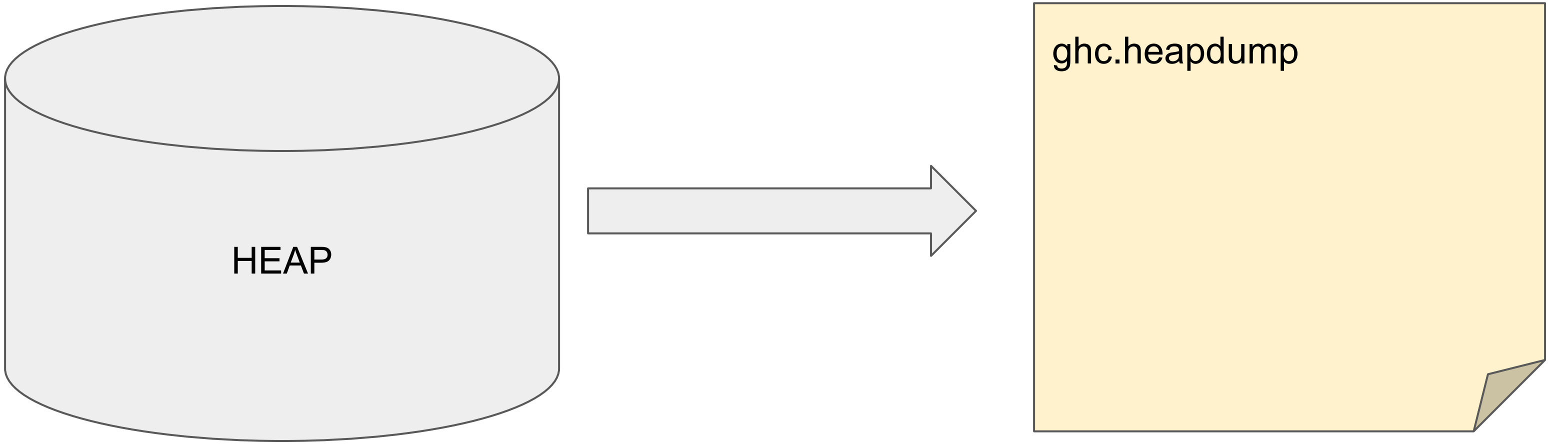


A weak pointer doesn’t act a GC root

WEAK

dyepack

Heap Snapshots



Offline precise analysis

BLeak: Automatically Debugging Memory Leaks in Web Applications

John Vilk
University of Massachusetts Amherst, USA
jvilk@cs.umass.edu

Emery D. Berger
University of Massachusetts Amherst, USA
emery@cs.umass.edu

Abstract

Despite the presence of garbage collection in managed languages like JavaScript, memory leaks remain a serious problem. In the context of web applications, these leaks are especially pervasive and difficult to debug. Web application memory leaks can take many forms, including failing to dispose of unneeded event listeners, repeatedly injecting iframes and CSS files, and failing to call cleanup routines in third-party libraries. Leaks degrade responsiveness by increasing GC frequency and overhead, and can even lead to browser tab crashes by exhausting available memory. Because previous leak detection approaches designed for conventional C, C++ or Java applications are ineffective in the browser environment, tracking down leaks currently requires intensive manual effort by web developers.

This paper introduces BLeak (**B**rowser **Leak** debugger), the first system for automatically debugging memory leaks in web applications. BLeak's algorithms leverage the observa-

CCS Concepts • **Software and its engineering → Software testing and debugging;**

Keywords Memory leaks, debugging, leak detection, web development, JavaScript

ACM Reference Format:
John Vilk and Emery D. Berger. 2018. BLeak: Automatically Debugging Memory Leaks in Web Applications. In *Proceedings of 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'18).* ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/3192366.3192376

1 Introduction

Browsers are one of the most popular applications on both smartphones and desktop platforms [3, 53]. They also have an established reputation for consuming significant amounts of memory [26, 38, 43]. To address this problem, browser vendors have spent considerable effort on shrinking their

John Vilk and Emery D. Berger. 2018. BLeak: automatically debugging memory leaks in web applications. PLDI 2018