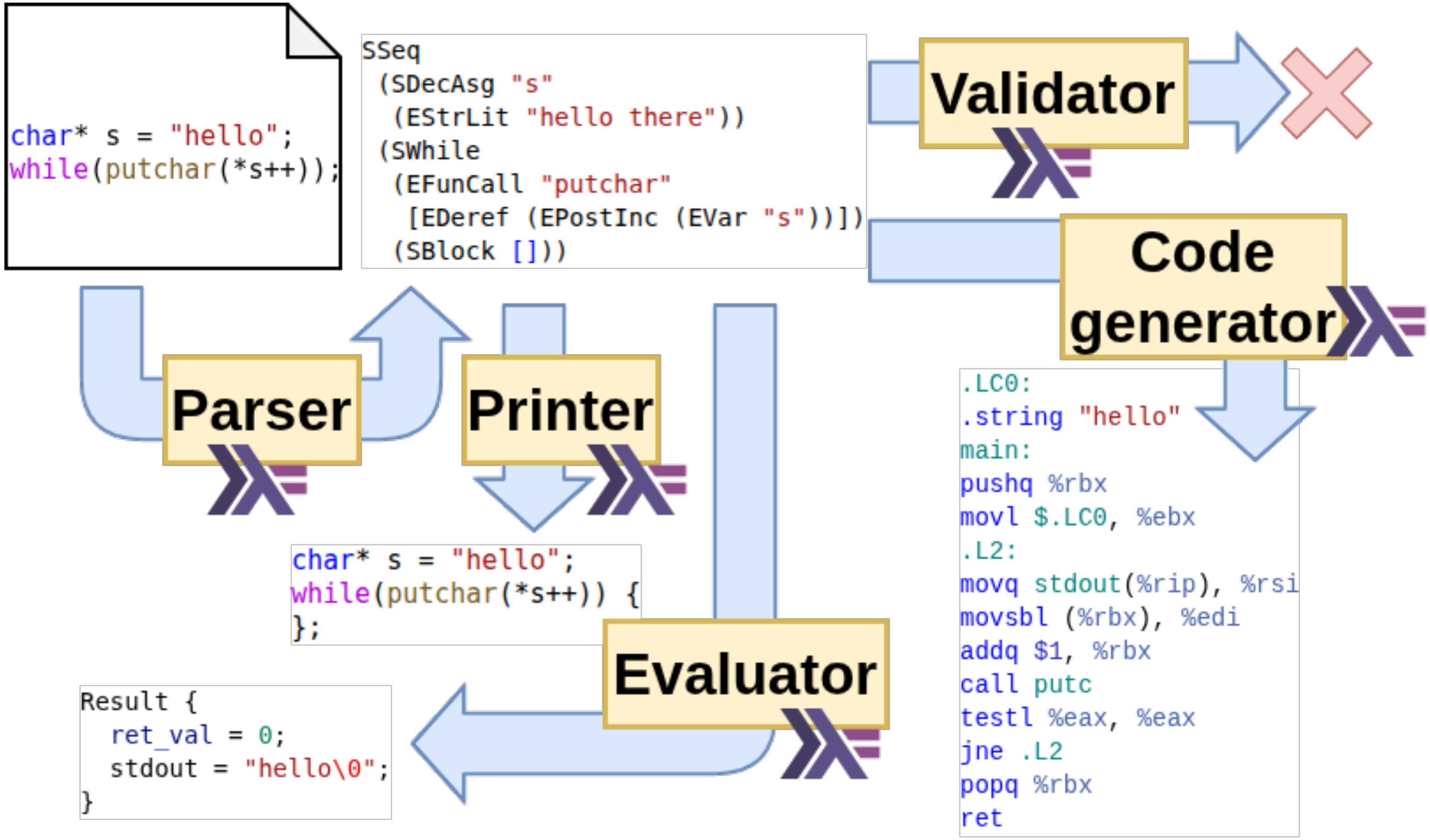
Compiler Architecture



```
char* s = "hello";
while(putchar(*s++));
```

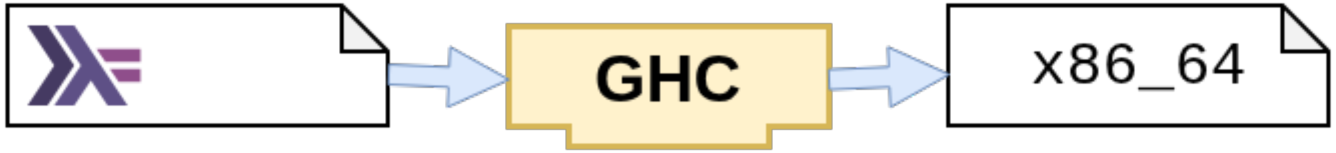
```
SSeq
 (SDecAsg "s"
  (EStrLit "hello there"))
 (SWhile
  (EFunCall "putchar"
   [EDeref (EPostInc (EVar "s"))])
  (SBlock []))
```

```
char* s = "hello";
while(putchar(*s++)) {
};
```

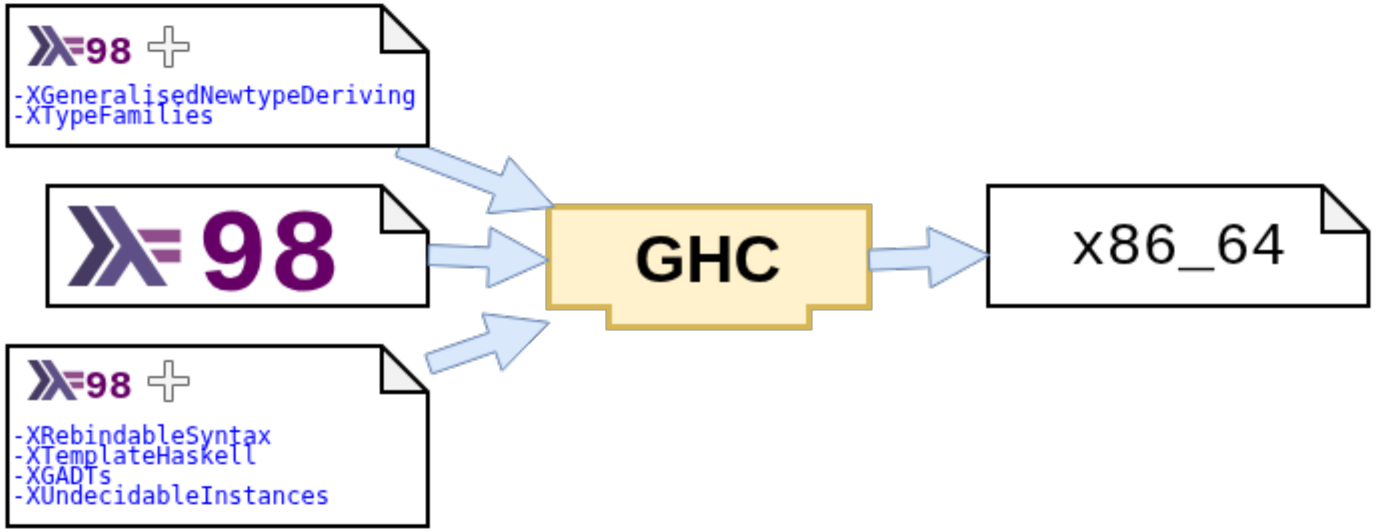
```
Result {
  ret_val = 0;
  stdout = "hello\0";
}
```

```
.LC0:
.string "hello"
main:
pushq %rbx
movl $.LC0, %ebx
.L2:
movq stdout(%rip), %rsi
movsbl (%rbx), %edi
addq $1, %rbx
call putc
testl %eax, %eax
jne .L2
popq %rbx
ret
```

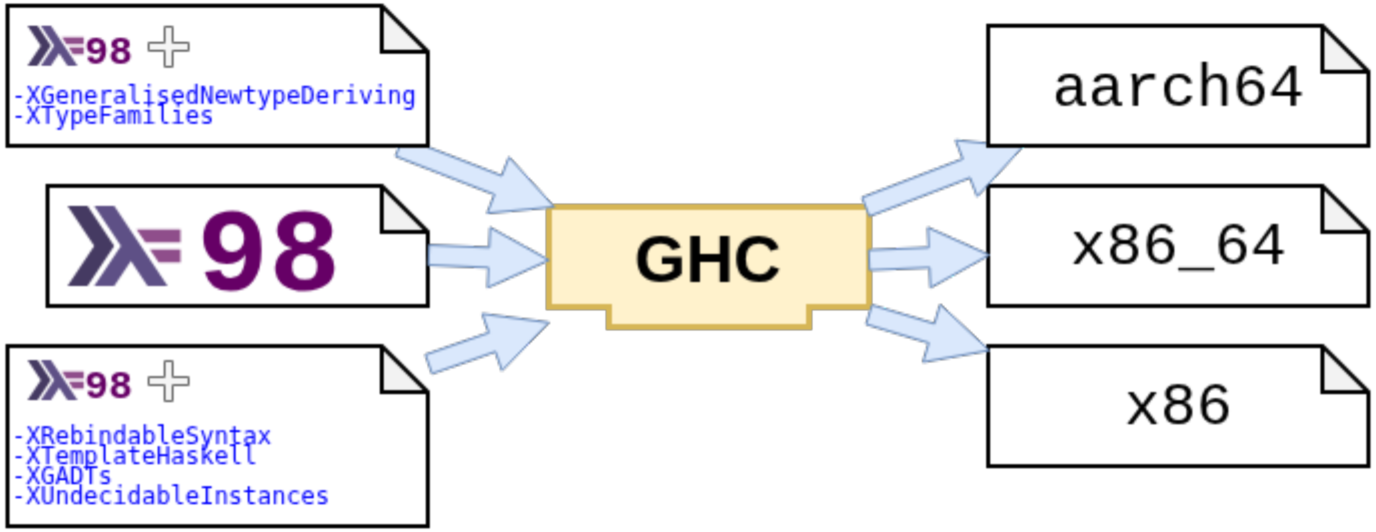
How hard can it be?



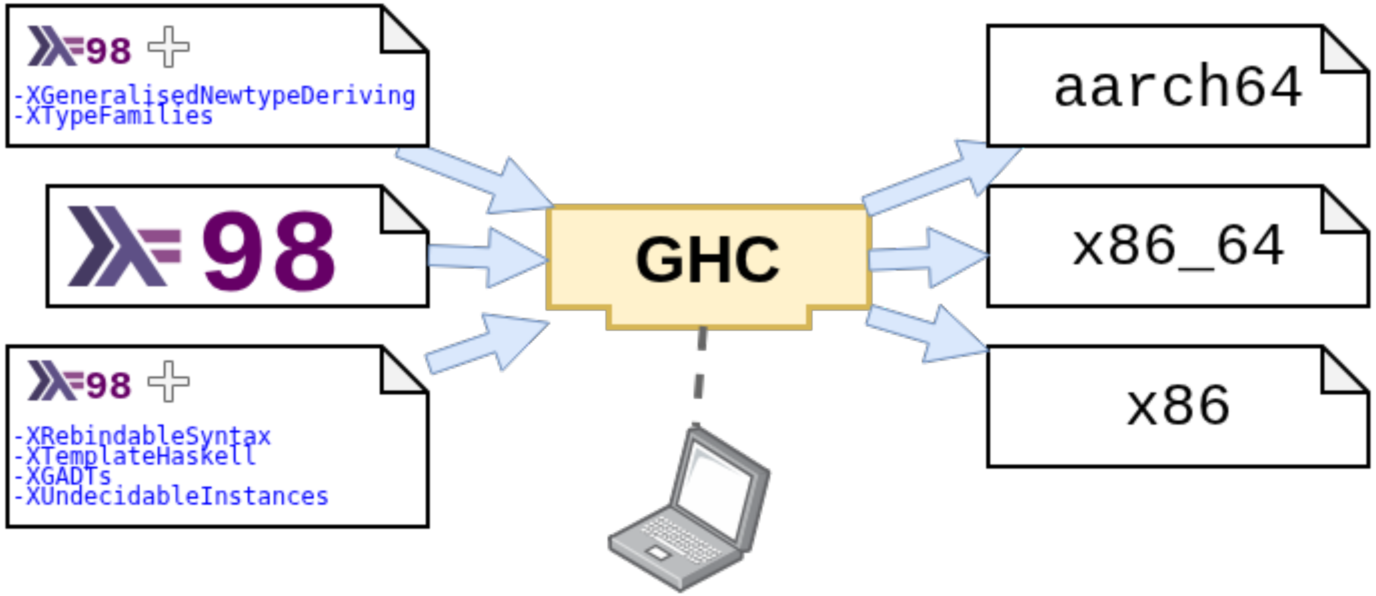
How hard can it be?



How hard can it be?



How hard can it be?



98

-XGeneralisedNewtypeDeriving
-XTypeFamilies

98

98

-XRebindableSyntax
-XTemplateHaskell
-XGADTs
-XUndecidableInstances

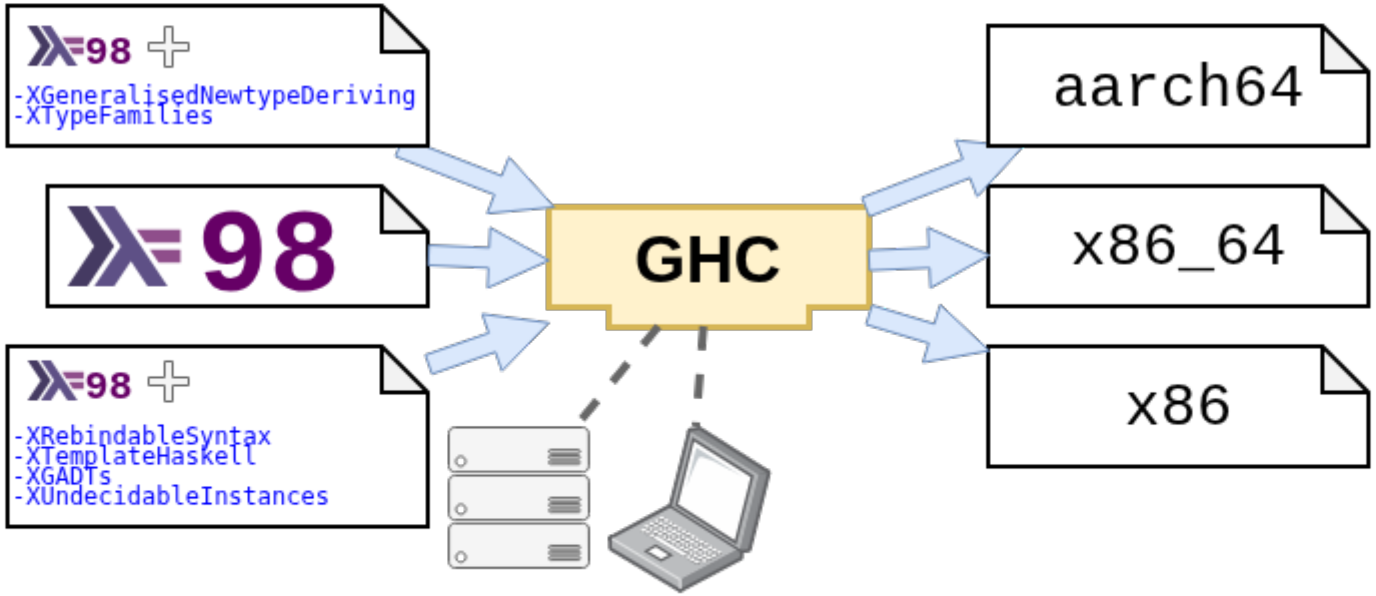
GHC

aarch64

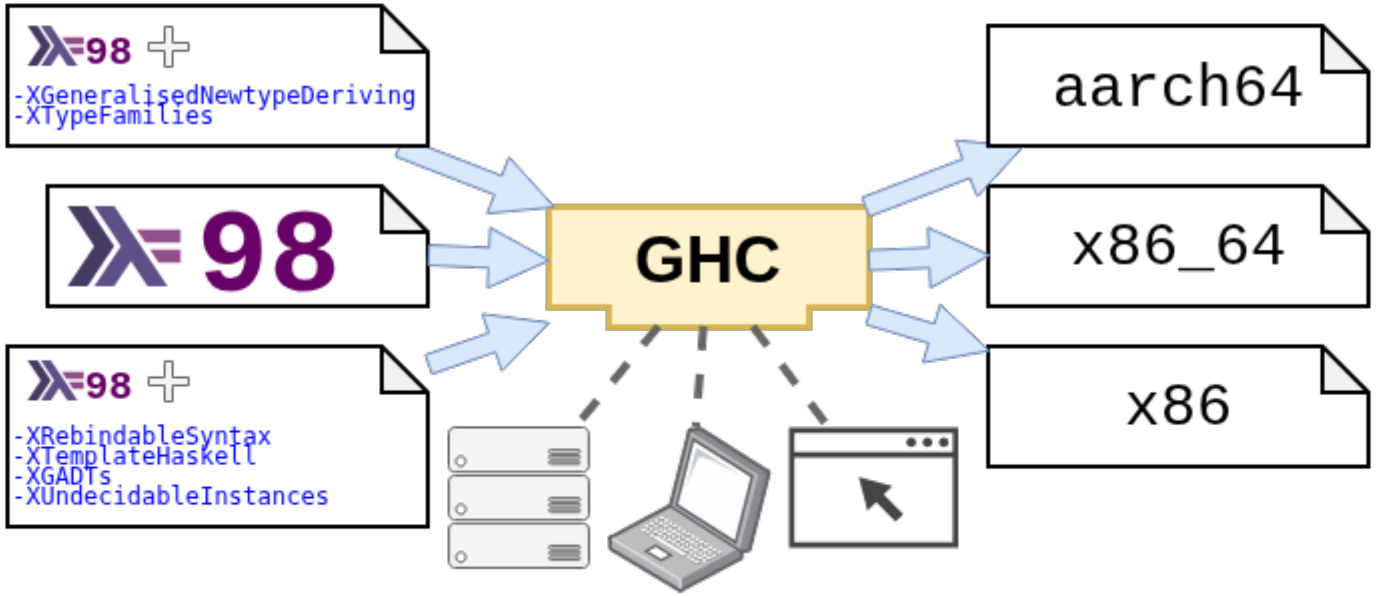
x86_64

x86

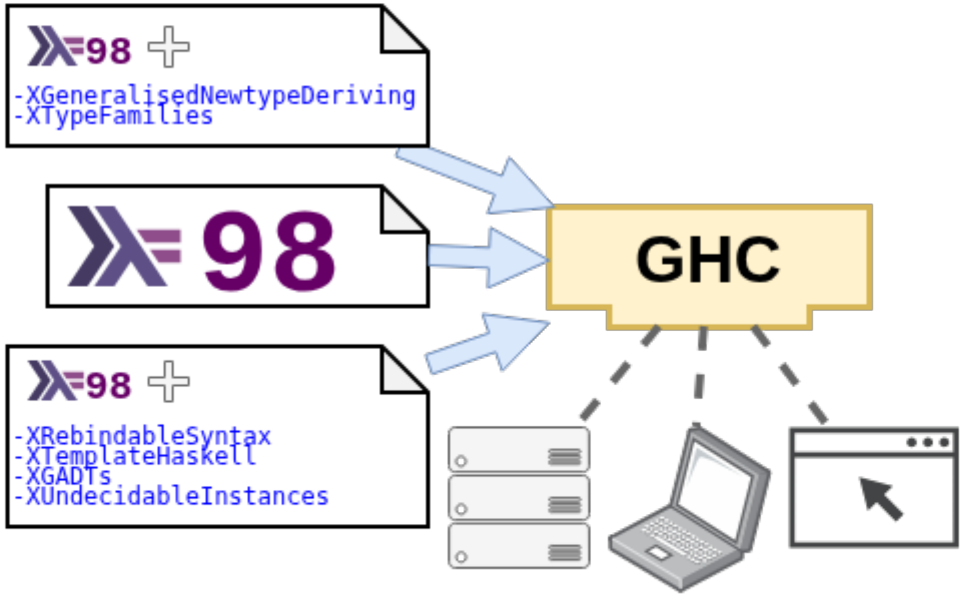
How hard can it be?



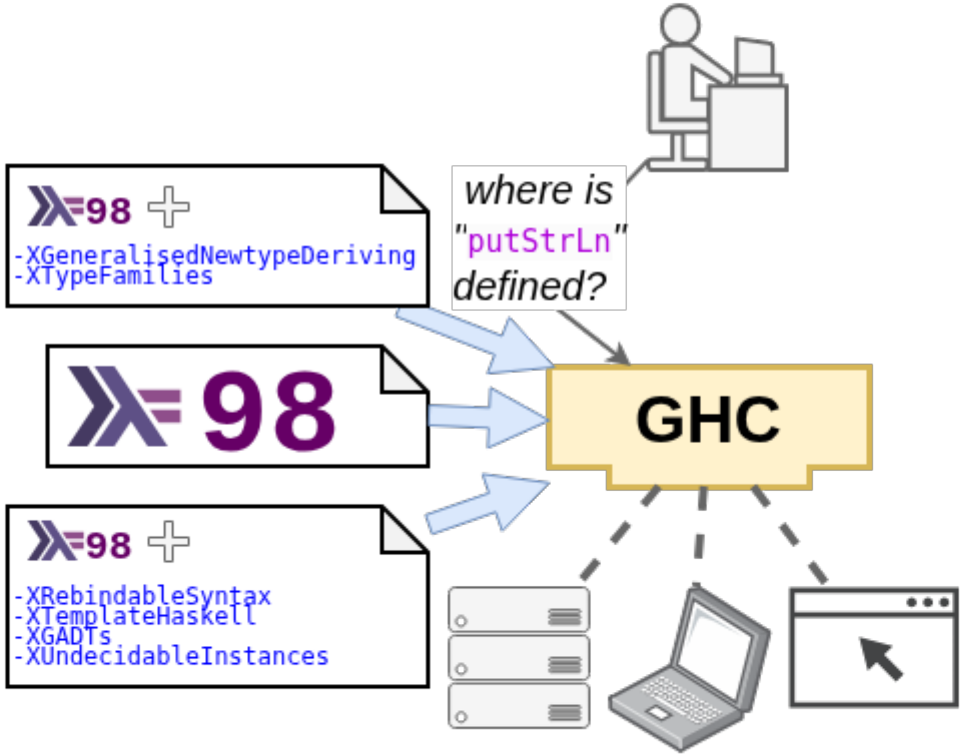
How hard can it be?



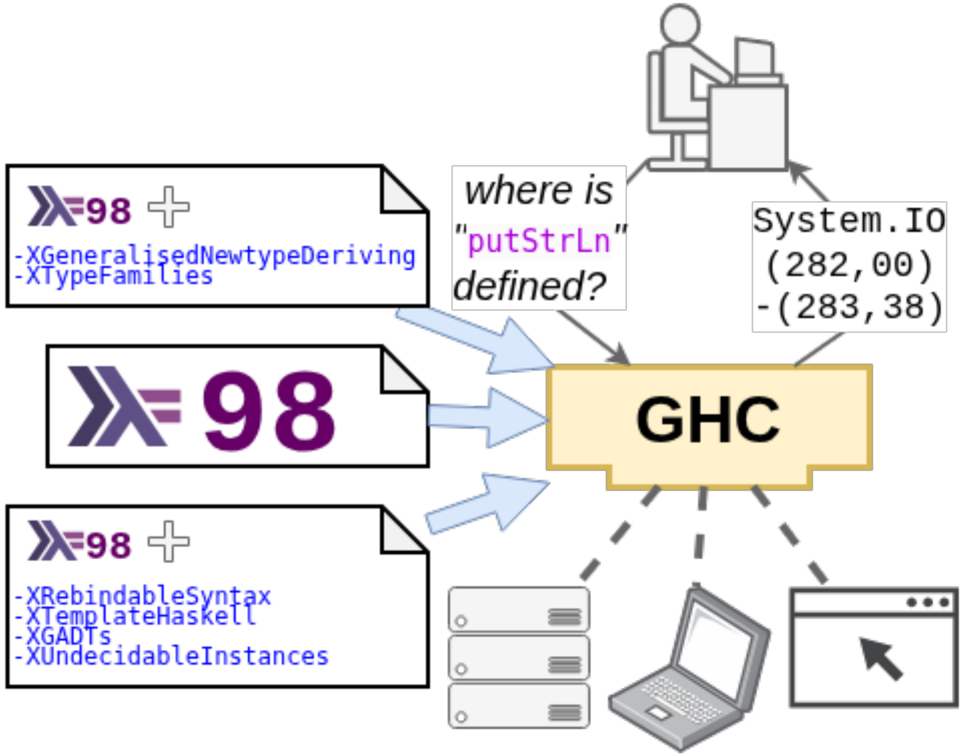
How hard can it be?



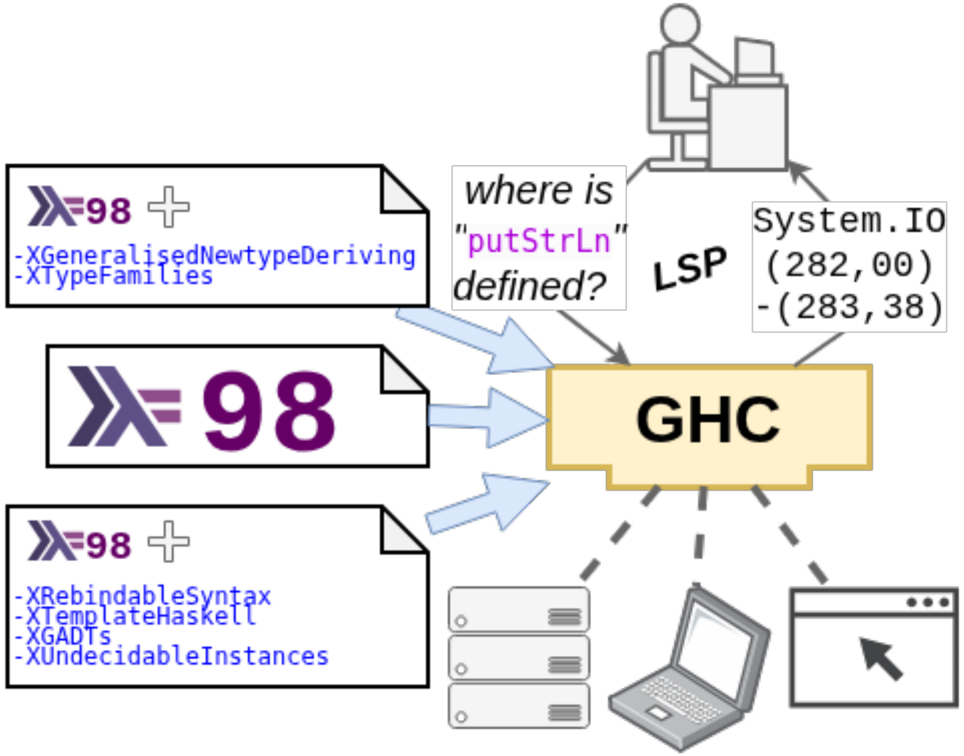
How hard can it be?



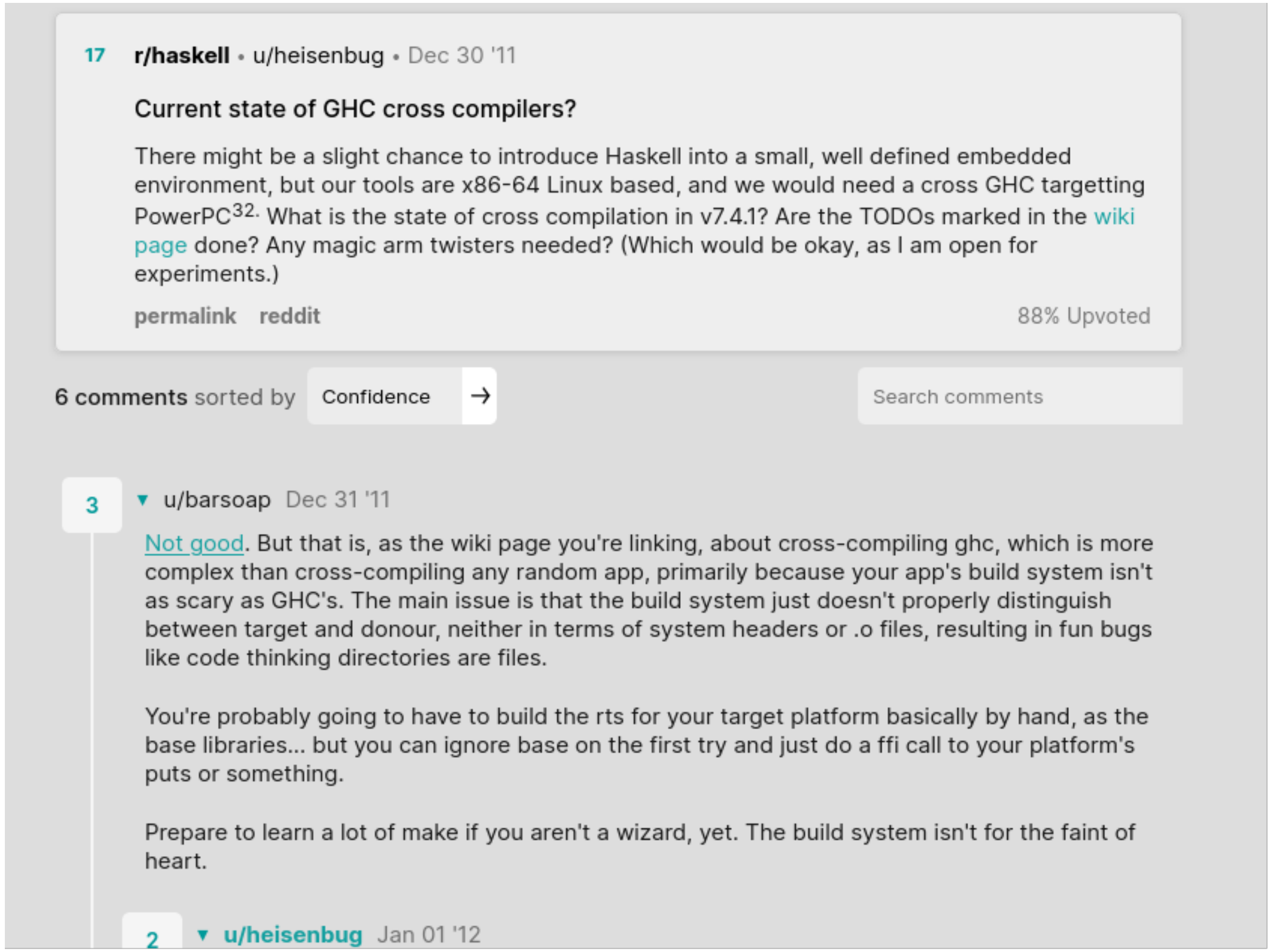
How hard can it be?



How hard can it be?



Hard even for experts



r/haskell • u/heisenbug • Dec 30 '11

Current state of GHC cross compilers?

There might be a slight chance to introduce Haskell into a small, well defined embedded environment, but our tools are x86-64 Linux based, and we would need a cross GHC targetting PowerPC$^{32.}$ What is the state of cross compilation in v7.4.1? Are the TODOs marked in the wiki page done? Any magic arm twisters needed? (Which would be okay, as I am open for experiments.)

u/barsoap Dec 31 '11

Not good. But that is, as the wiki page you're linking, about cross-compiling ghc, which is more complex than cross-compiling any random app, primarily because your app's build system isn't as scary as GHC's. The main issue is that the build system just doesn't properly distinguish between target and donour, neither in terms of system headers or .o files, resulting in fun bugs like code thinking directories are files.

You're probably going to have to build the rts for your target platform basically by hand, as the base libraries... but you can ignore base on the first try and just do a ffi call to your platform's puts or something.

Prepare to learn a lot of make if you aren't a wizard, yet. The build system isn't for the faint of heart.

u/heisenbug Jan 01 '12

Hard even for experts

17 r/haskell

Current

There mi
environm
PowerPC
page dor
experime

permalink

6 comments sor

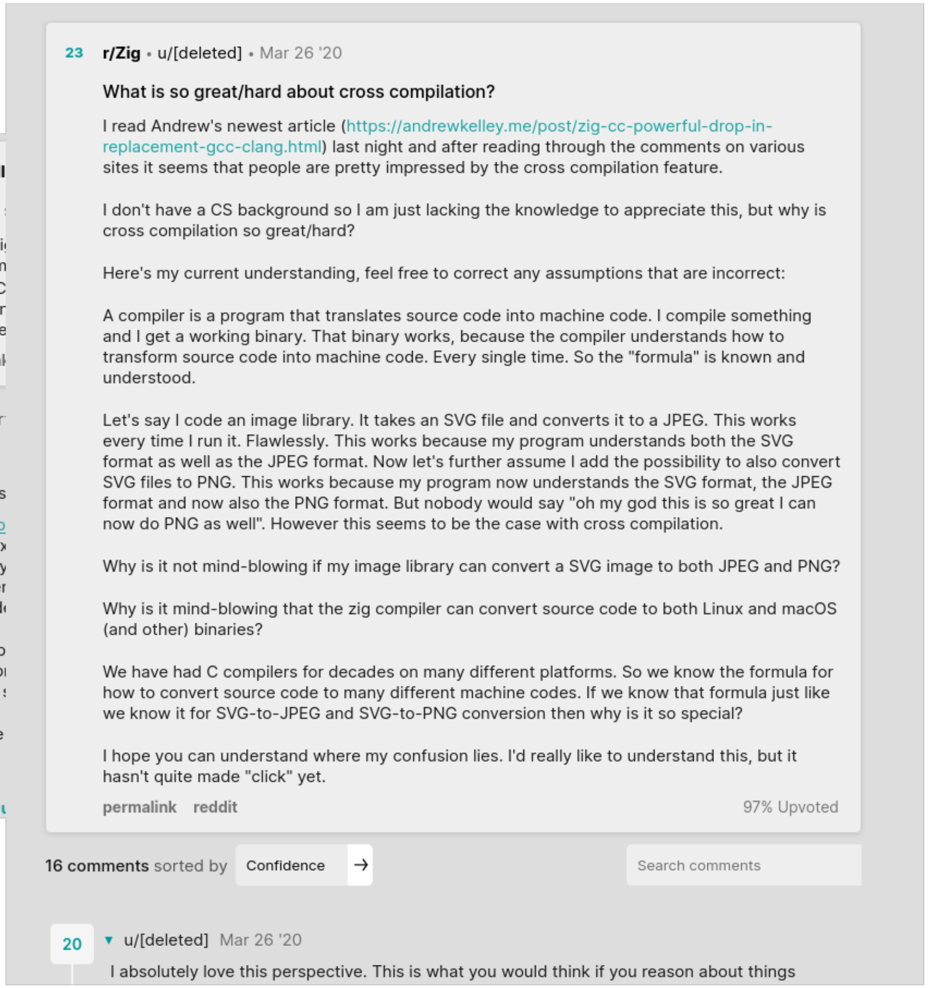
3 u/bars

Not goo
complex
as scary
betweer
like code

You're p
base libr
puts or

Prepare
heart.

2



23 r/Zig • u/[deleted] • Mar 26 '20

What is so great/hard about cross compilation?

I read Andrew's newest article (https://andrewkelley.me/post/zig-cc-powerful-drop-in-replacement-gcc-clang.html) last night and after reading through the comments on various sites it seems that people are pretty impressed by the cross compilation feature.

I don't have a CS background so I am just lacking the knowledge to appreciate this, but why is cross compilation so great/hard?

Here's my current understanding, feel free to correct any assumptions that are incorrect:

A compiler is a program that translates source code into machine code. I compile something and I get a working binary. That binary works, because the compiler understands how to transform source code into machine code. Every single time. So the "formula" is known and understood.

Let's say I code an image library. It takes an SVG file and converts it to a JPEG. This works every time I run it. Flawlessly. This works because my program understands both the SVG format as well as the JPEG format. Now let's further assume I add the possibility to also convert SVG files to PNG. This works because my program now understands the SVG format, the JPEG format and now also the PNG format. But nobody would say "oh my god this is so great I can now do PNG as well". However this seems to be the case with cross compilation.

Why is it not mind-blowing if my image library can convert a SVG image to both JPEG and PNG?

Why is it mind-blowing that the zig compiler can convert source code to both Linux and macOS (and other) binaries?

We have had C compilers for decades on many different platforms. So we know the formula for how to convert source code to many different machine codes. If we know that formula just like we know it for SVG-to-JPEG and SVG-to-PNG conversion then why is it so special?

I hope you can understand where my confusion lies. I'd really like to understand this, but it hasn't quite made "click" yet.

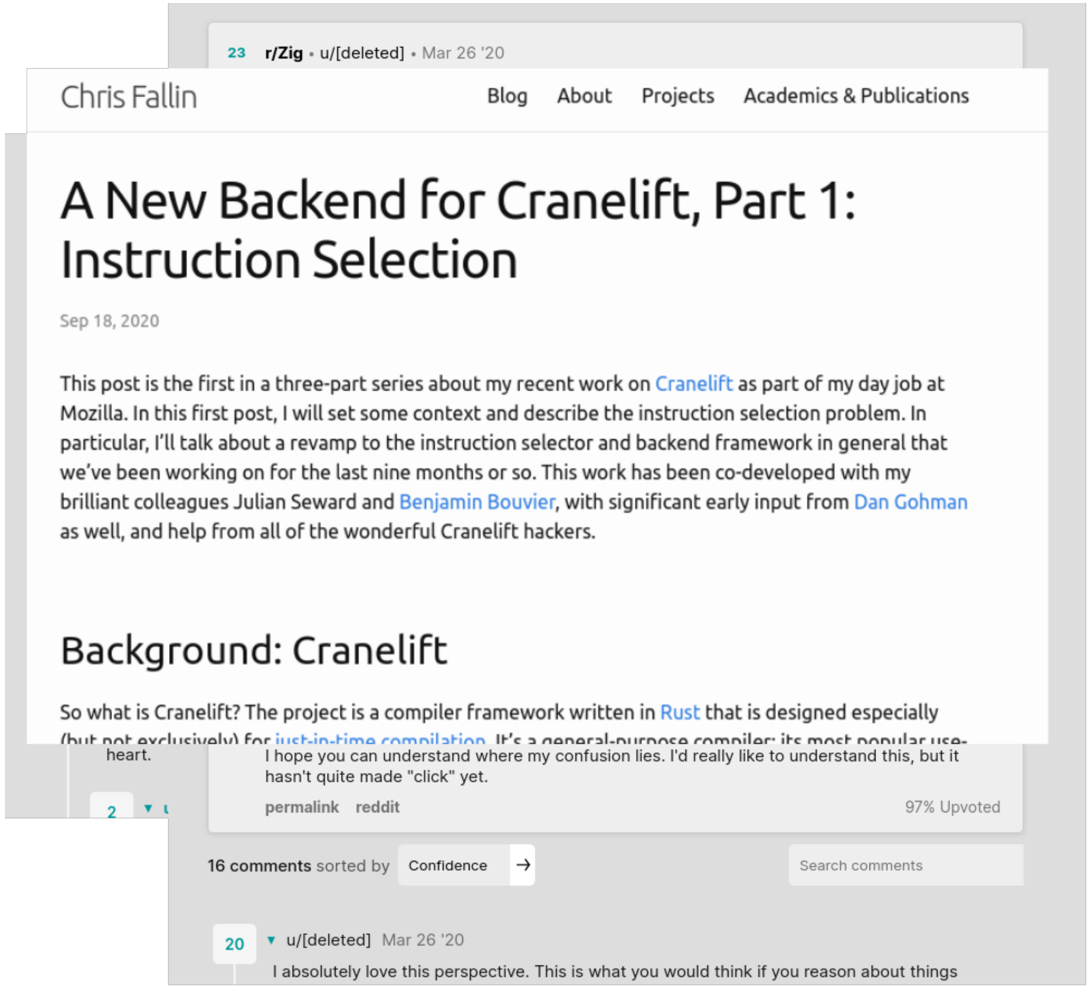
permalink reddit 97% Upvoted

16 comments sorted by Confidence →

20 u/[deleted] Mar 26 '20

I absolutely love this perspective. This is what you would think if you reason about things

Hard even for experts



23 r/Zig • u/[deleted] • Mar 26 '20

Chris Fallin

Blog About Projects Academics & Publications

A New Backend for Cranelift, Part 1: Instruction Selection

Sep 18, 2020

This post is the first in a three-part series about my recent work on Cranelift as part of my day job at Mozilla. In this first post, I will set some context and describe the instruction selection problem. In particular, I'll talk about a revamp to the instruction selector and backend framework in general that we've been working on for the last nine months or so. This work has been co-developed with my brilliant colleagues Julian Seward and Benjamin Bouvier, with significant early input from Dan Gohman as well, and help from all of the wonderful Cranelift hackers.

Background: Cranelift

So what is Cranelift? The project is a compiler framework written in Rust that is designed especially (but not exclusively) for just-in-time compilation. It's a general-purpose compiler; its most popular use-

heart.

2

I hope you can understand where my confusion lies. I'd really like to understand this, but it hasn't quite made "click" yet.

permalink reddit 97% Upvoted

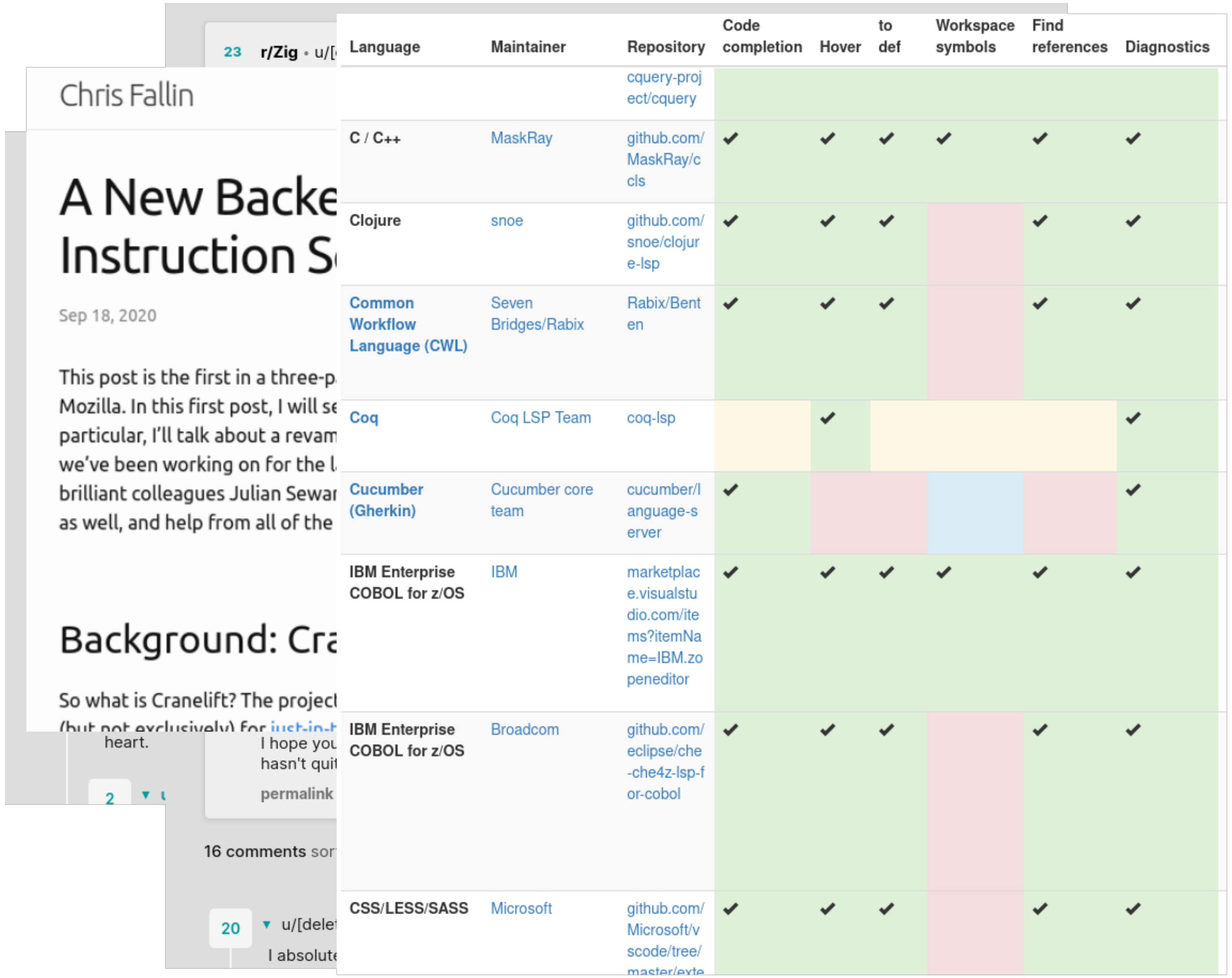
16 comments sorted by Confidence →

Search comments

20 ▾ u/[deleted] Mar 26 '20

I absolutely love this perspective. This is what you would think if you reason about things

Hard even for experts



23 r/Zig • u/[

Chris Fallin

A New Backe
Instruction S

Sep 18, 2020

This post is the first in a three-p
Mozilla. In this first post, I will se
particular, I'll talk about a revam
we've been working on for the l
brilliant colleagues Julian Sewar
as well, and help from all of the

Background: Cra

So what is Cranelift? The project
(but not exclusively) for just-in-t

heart.

I hope you
hasn't quit

permalink

2 ▼ u

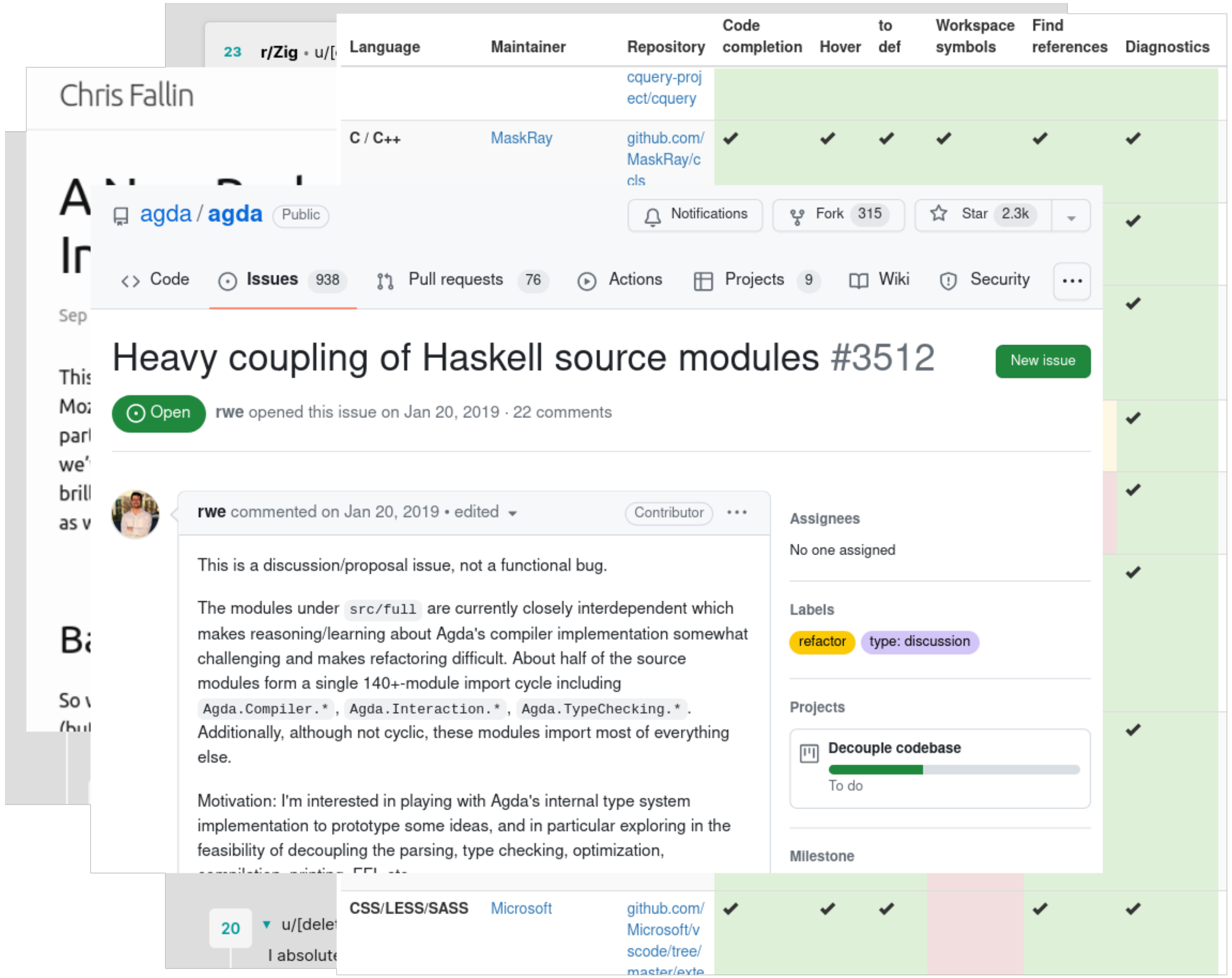
16 comments sor

20 ▼ u/[delet

I absolute

Language	Maintainer	Repository	Code completion	Hover	to def	Workspace symbols	Find references	Diagnostics
		cquery-project/cquery	🟩	🟩	🟩	🟩	🟩	🟩
C / C++	MaskRay	github.com/MaskRay/ccls	✔	✔	✔	✔	✔	✔
Clojure	snoe	github.com/snoe/clojure-lsp	✔	✔	✔	🟥	✔	✔
Common Workflow Language (CWL)	Seven Bridges/Rabix	Rabix/Benten	✔	✔	✔	🟥	✔	✔
Coq	Coq LSP Team	coq-lsp	🟨	✔	🟨	🟨	🟨	✔
Cucumber (Gherkin)	Cucumber core team	cucumber/language-server	✔	🟥	🟥	🟦	🟥	✔
IBM Enterprise COBOL for z/OS	IBM	marketplace.visualstudio.com/items?itemName=IBM.zopeneditor	✔	✔	✔	✔	✔	✔
IBM Enterprise COBOL for z/OS	Broadcom	github.com/eclipse/che-che4z-lsp-for-cobol	✔	✔	✔	🟥	✔	✔
CSS/LESS/SASS	Microsoft	github.com/Microsoft/vscode/tree/master/exte	✔	✔	✔	🟥	✔	✔

Hard even for experts



23 r/Zig • u/[

20 u/[dele

I absolute

Chris Fallin

A N P l

Ir

Sep

This
Mo
part
we'
brill
as v

Ba

So v
(bu

Language	Maintainer	Repository	Code completion	Hover	to def	Workspace symbols	Find references	Diagnostics
		cquery-project/cquery						
C / C++	MaskRay	github.com/MaskRay/ccls	✔	✔	✔	✔	✔	✔
CSS/LESS/SASS	Microsoft	github.com/Microsoft/vscode/tree/master/exte	✔	✔	✔		✔	✔

Heavy coupling of Haskell source modules #3512

Open **rwe** opened this issue on Jan 20, 2019 · 22 comments

rwe commented on Jan 20, 2019 • edited Contributor

This is a discussion/proposal issue, not a functional bug.

The modules under `src/full` are currently closely interdependent which makes reasoning/learning about Agda's compiler implementation somewhat challenging and makes refactoring difficult. About half of the source modules form a single 140+-module import cycle including `Agda.Compiler.*`, `Agda.Interaction.*`, `Agda.TypeChecking.*`. Additionally, although not cyclic, these modules import most of everything else.

Motivation: I'm interested in playing with Agda's internal type system implementation to prototype some ideas, and in particular exploring in the feasibility of decoupling the parsing, type checking, optimization,

Assignees

No one assigned

Labels

refactor type: discussion

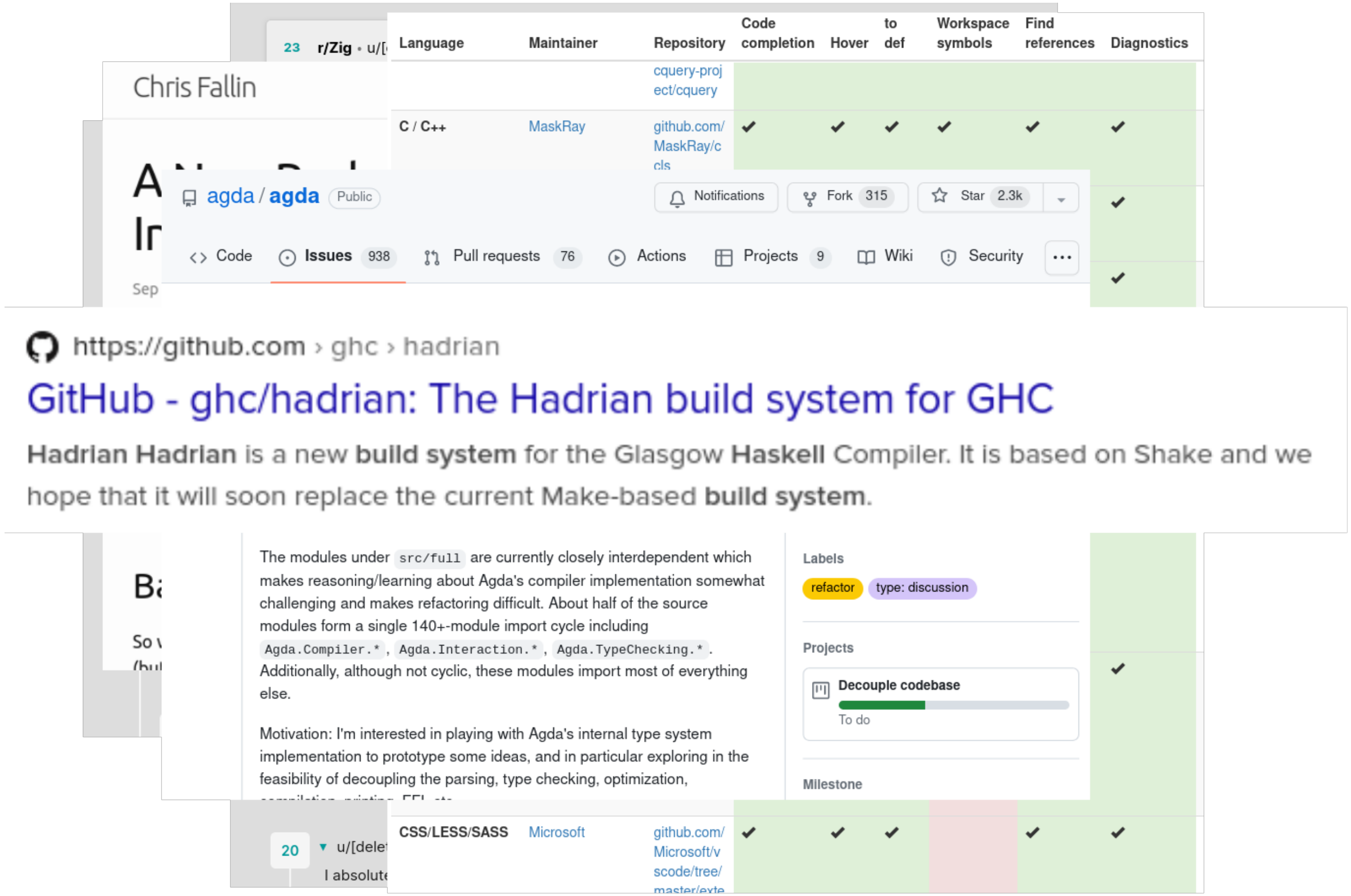
Projects

Decouple codebase

To do

Milestone

Hard even for experts



23 r/Zig • u/[

Chris Fallin

A N
Ir

Sep

Ba

So v
(bu

20 ▾ u/[delet
I absolute

Language	Maintainer	Repository	Code completion	Hover	to def	Workspace symbols	Find references	Diagnostics
		cquery-project/cquery						
C / C++	MaskRay	github.com/MaskRay/ccls	✓	✓	✓	✓	✓	✓
CSS/LESS/SASS	Microsoft	github.com/Microsoft/vscode/tree/master/exte	✓	✓	✓		✓	✓

agda / agda Public

Notifications · Fork 315 · Star 2.3k

Code · Issues 938 · Pull requests 76 · Actions · Projects 9 · Wiki · Security · ···

https://github.com › ghc › hadrian

GitHub - ghc/hadrian: The Hadrian build system for GHC

Hadrian Hadrian is a new **build system** for the Glasgow **Haskell** Compiler. It is based on Shake and we hope that it will soon replace the current Make-based **build system**.

The modules under `src/full` are currently closely interdependent which makes reasoning/learning about Agda's compiler implementation somewhat challenging and makes refactoring difficult. About half of the source modules form a single 140+-module import cycle including `Agda.Compiler.*`, `Agda.Interaction.*`, `Agda.TypeChecking.*`. Additionally, although not cyclic, these modules import most of everything else.

Motivation: I'm interested in playing with Agda's internal type system implementation to prototype some ideas, and in particular exploring in the feasibility of decoupling the parsing, type checking, optimization, compilation, printing, FFI, etc.

Labels

refactor · type: discussion

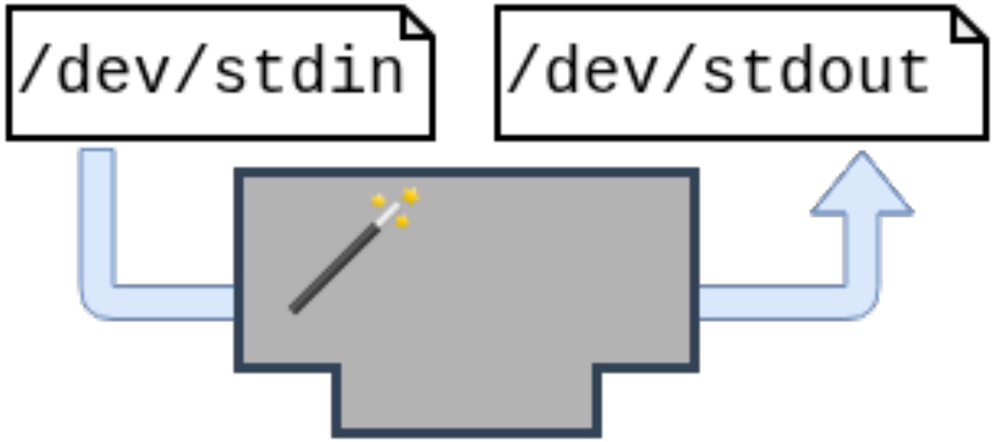
Projects

Decouple codebase

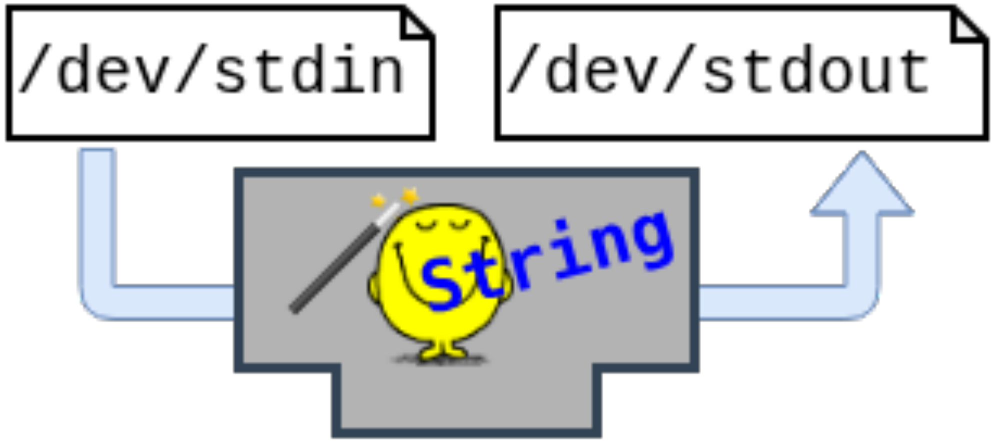
To do

Milestone

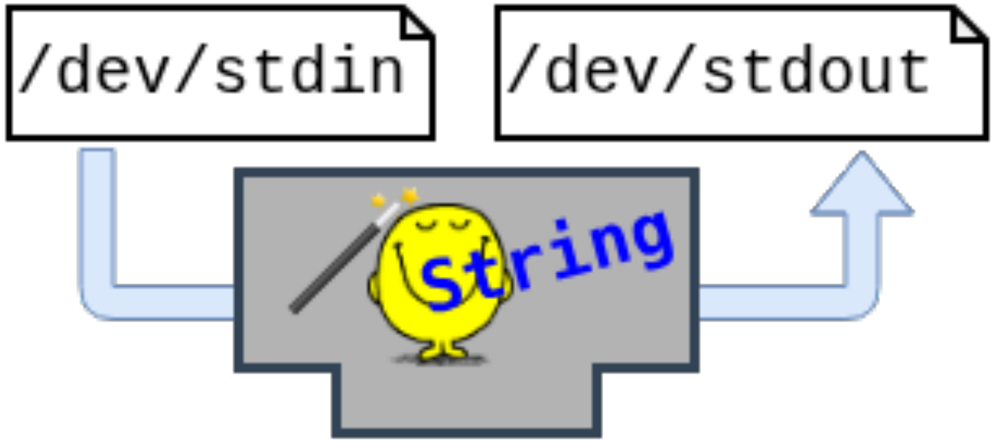
Compiler Architecture: 1-pass



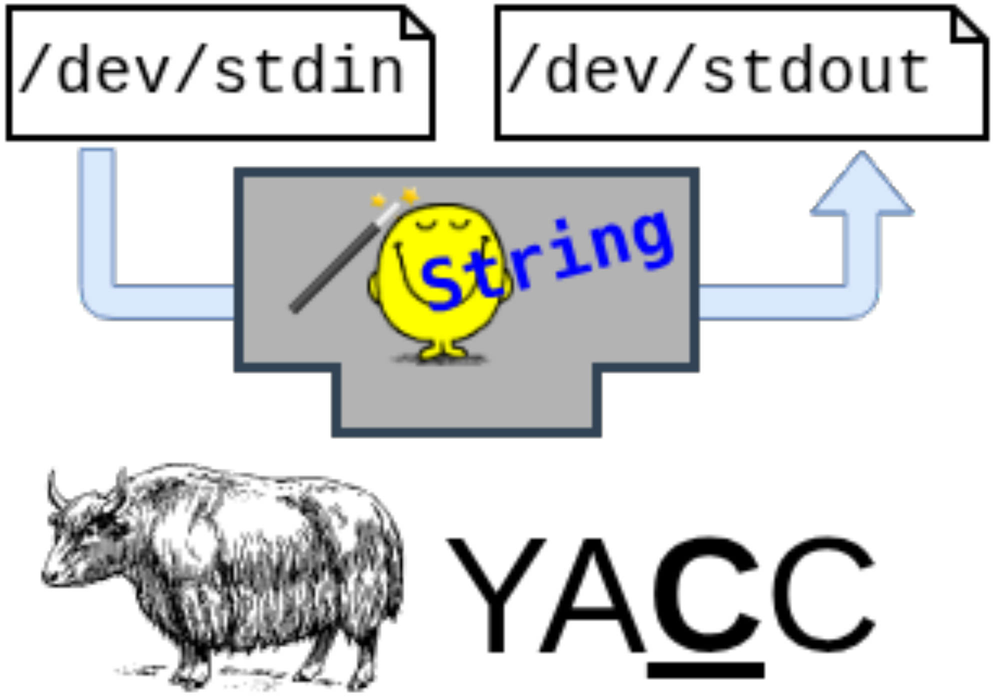
Compiler Architecture: 1-pass



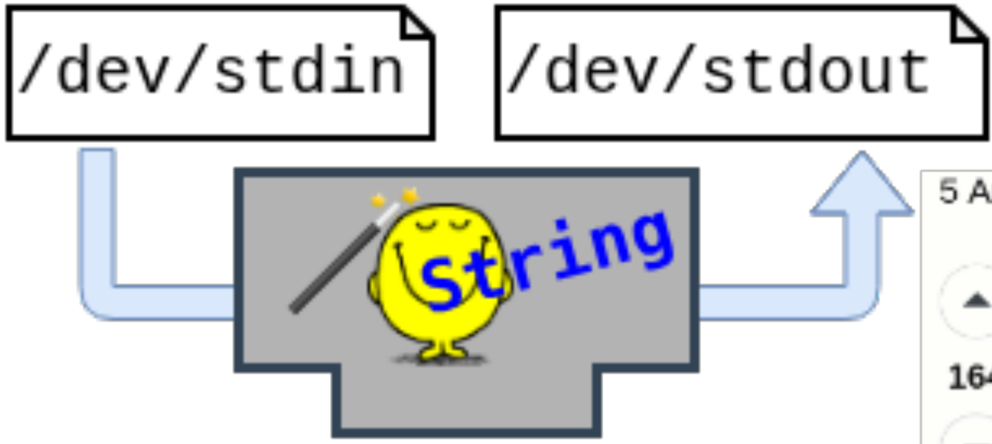
Compiler Architecture: 1-pass



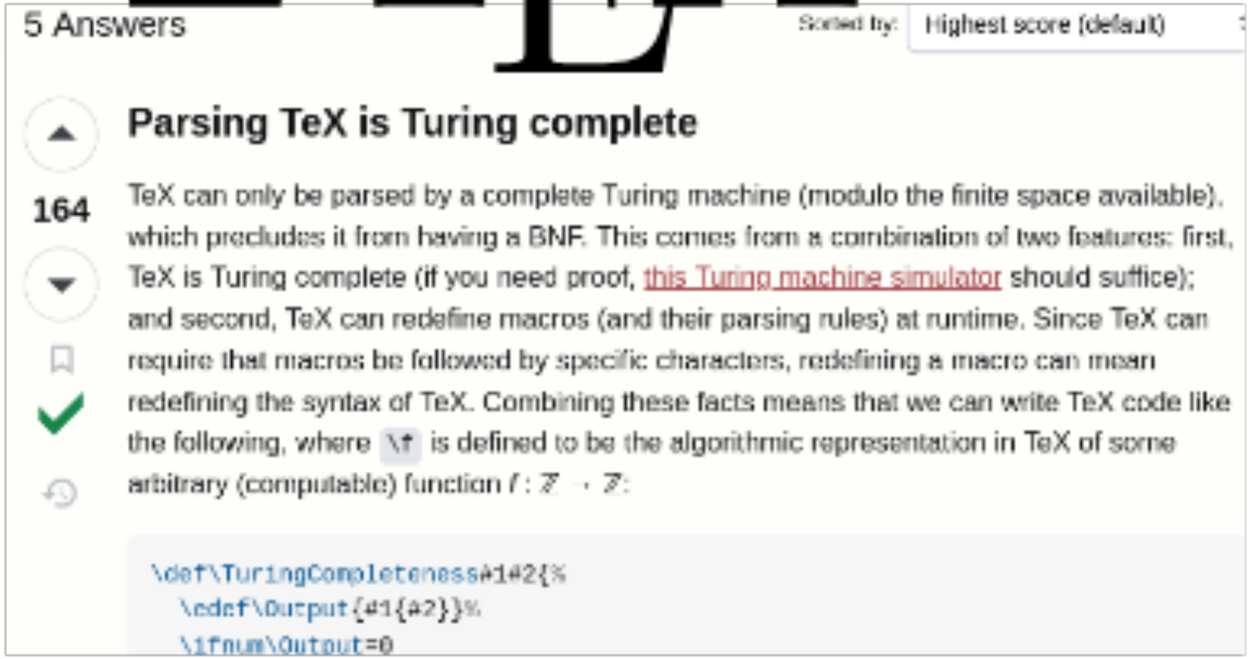
Compiler Architecture: 1-pass



Compiler Architecture: 1-pass



LaTeX

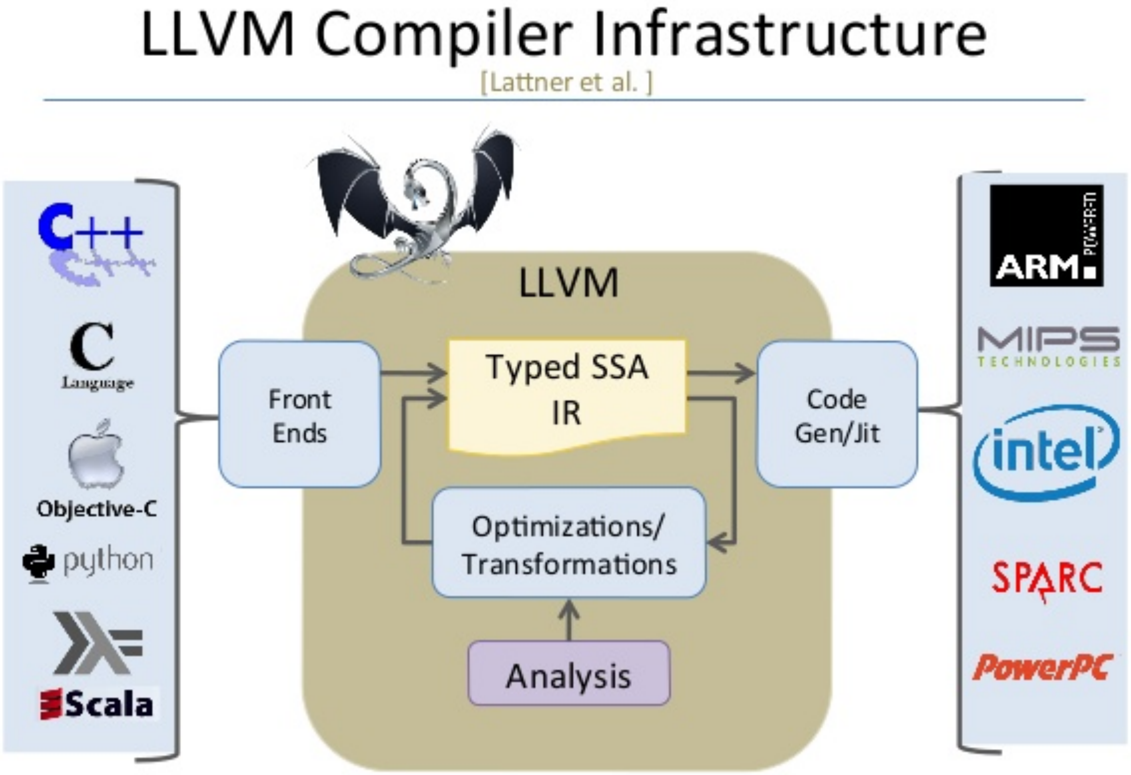


Parsing TeX is Turing complete

TeX can only be parsed by a complete Turing machine (modulo the finite space available), which precludes it from having a BNF. This comes from a combination of two features: first, TeX is Turing complete (if you need proof, this Turing machine simulator should suffice); and second, TeX can redefine macros (and their parsing rules) at runtime. Since TeX can require that macros be followed by specific characters, redefining a macro can mean redefining the syntax of TeX. Combining these facts means that we can write TeX code like the following, where `\f` is defined to be the algorithmic representation in TeX of some arbitrary (computable) function $f: \mathbb{Z} \to \mathbb{Z}$:

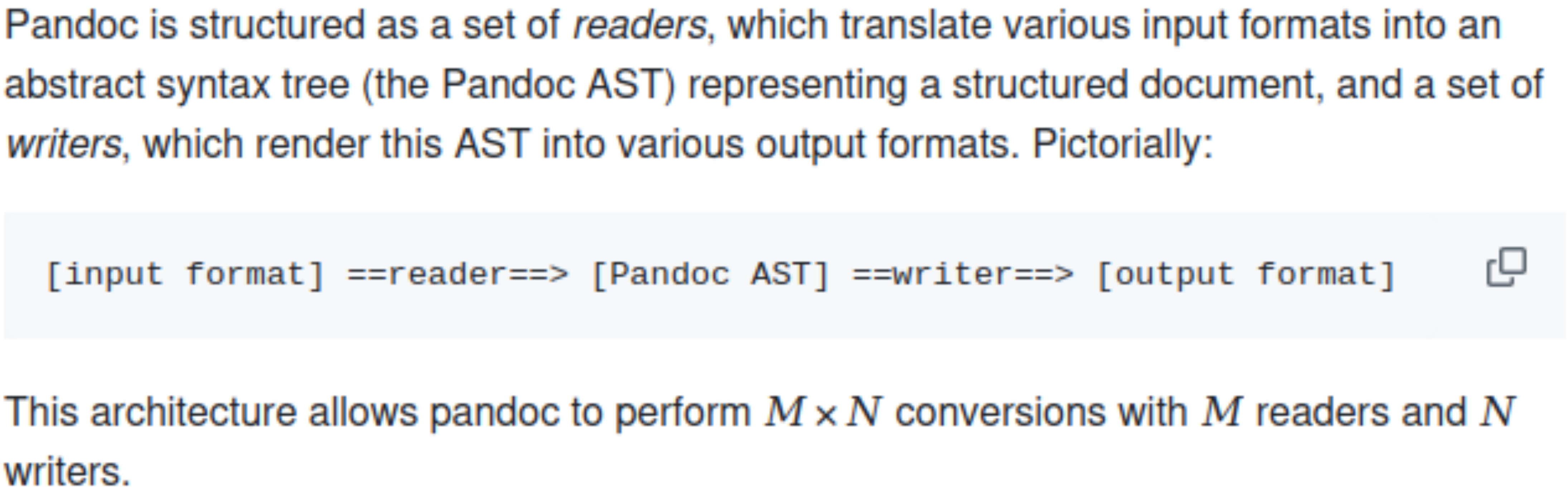
```
\def\TuringCompleteness#1#2{%
  \edef\Output{#1{#2}}%
  \ifnum\Output=0
```

Compiler Architecture: 3-pass



P. Teoh, Stackoverflow, https://stackoverflow.com/a/49081640

Compiler Architecture: 3-pass



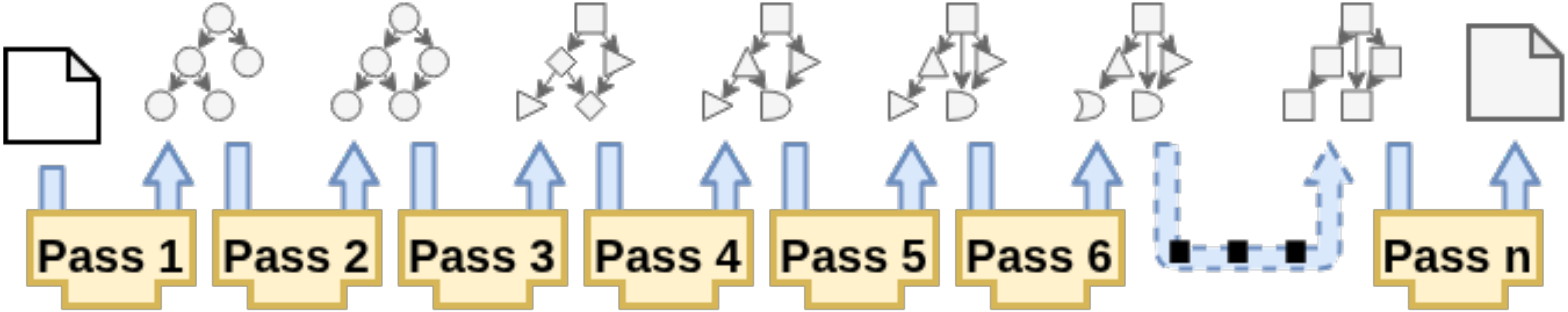
Pandoc is structured as a set of *readers*, which translate various input formats into an abstract syntax tree (the Pandoc AST) representing a structured document, and a set of *writers*, which render this AST into various output formats. Pictorially:

```
[input format] ==reader==> [Pandoc AST] ==writer==> [output format]
```

This architecture allows pandoc to perform $M \times N$ conversions with M readers and N writers.

Pandoc manual, https://github.com/jgm/pandoc/blob/main/doc/using-the-pandoc-api.md

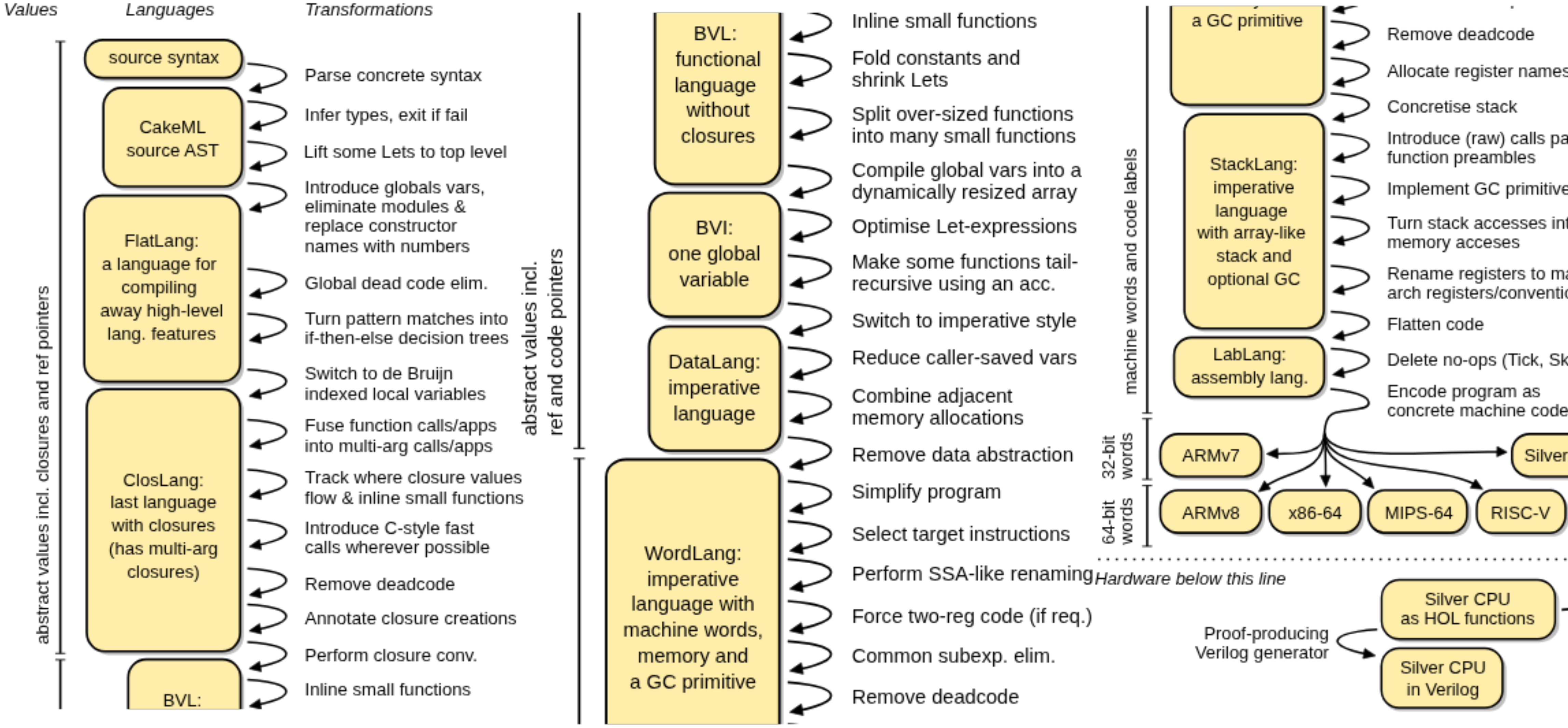
Compiler Architecture: nanopass



CAKEML

A Verified Implementation of ML

https://cakeml.org/



Values

Languages

Transformations

abstract values incl. closures and ref pointers

source syntax

Parse concrete syntax

CakeML source AST

Infer types, exit if fail

Lift some Lets to top level

Introduce globals vars, eliminate modules & replace constructor names with numbers

FlatLang: a language for compiling away high-level lang. features

Global dead code elim.

Turn pattern matches into if-then-else decision trees

Switch to de Bruijn indexed local variables

ClosLang: last language with closures (has multi-arg closures)

Fuse function calls/apps into multi-arg calls/apps

Track where closure values flow & inline small functions

Introduce C-style fast calls wherever possible

Remove deadcode

Annotate closure creations

Perform closure conv.

Inline small functions

BVL:

abstract values incl. ref and code pointers

BVL: functional language without closures

Inline small functions

Fold constants and shrink Lets

Split over-sized functions into many small functions

Compile global vars into a dynamically resized array

BVI: one global variable

Optimise Let-expressions

Make some functions tail-recursive using an acc.

Switch to imperative style

DataLang: imperative language

Reduce caller-saved vars

Combine adjacent memory allocations

Remove data abstraction

WordLang: imperative language with machine words, memory and a GC primitive

Simplify program

Select target instructions

Perform SSA-like renaming

Force two-reg code (if req.)

Common subexp. elim.

Remove deadcode

machine words and code labels

a GC primitive

Remove deadcode

Allocate register names

Concretise stack

Introduce (raw) calls pa function preambles

StackLang: imperative language with array-like stack and optional GC

Implement GC primitive

Turn stack accesses int memory acceses

Rename registers to ma arch registers/conventio

Flatten code

LabLang: assembly lang.

Delete no-ops (Tick, Sk

Encode program as concrete machine code

32-bit words

ARMv7

Silver

64-bit words

ARMv8

x86-64

MIPS-64

RISC-V

Hardware below this line

Silver CPU as HOL functions

Proof-producing Verilog generator

Silver CPU in Verilog

Nanopass: Parse

```
char* s = "hello";
while (
  putchar(*s++)
 );
```

```
char* s = "hello";
while (
  putchar(*s++)
 );
```

Nanopass: Infer Types

```
real_solns :: _ → _
          → _
real_solns a b c =
  let d = b**2 - 4*a*c
          in
  if d ≥ 0 then
    [(-b + sqrt d) /
          (2*a)
    ,(-b - sqrt d) /
          (2*a) ]
  else []
```

```
real_solns :: Float → Float
          → Float
real_solns a b c =
  let d = b**2 - 4*a*c ::
          Float in
  if d ≥ (0 :: Float) then
    [(-b + sqrt d) / (2*a)
    ,(-b - sqrt d) / (2*a) ]
  else []
```

Nanopass: for → while

```
for(int i = 0;
    i < l.length;
    i++) {
  do_stuff();
}
```

```
int i = 0;
while(i < l.length) {
  do_stuff();
  i++;
}
```

Nanopass: λ → class

```
int[] squares (int[] l) {
 Logger q = get_logger();
 return
  sum( map( (x => q.log(x*x))
          , l ));
}
```

```
int[] squares (int[] l) {
 Logger q = get_logger();
 return
  sum( map( new Lam43(q)
          , l ));
}
class Lam43 : Runnable {
 Logger q;
 object run (object x) {
  return q.log(x*x); }}
```

Nanopass: class → struct

```
class Player {
 uint coins;
 int hiscore;

 void again(){
  if(coins-- > 0) {
   int score = play();
   hiscore = max
     ( score
     , hiscore);
  }
```

Nanopass: reference counting

```
void test() {
  int[] xs =
    list(1,1000000);
  int[] ys =
    map(xs, inc);

  print(ys);

}
```

```
void test() {
  int[] xs =
    list(1,1000000);
  int[] ys =
    map(xs, inc);
  _drop(xs);
  print(ys);
  _drop(ys);
}
```

https://www.microsoft.com/en-us/research/uploads/prod/2020/11/perceus-tr-v1.pdf

Nanopass: Constant folding

```
float sphere_area(float r){
  float pi = calc_pi(5);
  return 4 * pi * r * r;
}
```

```
float sphere_area(float r){
  float pi = calc_pi(5);
  return 4 * pi * r * r;
}
```

Nanopass: Constant folding

```
float sphere_area(float r){
  float pi = calc_pi(5);
  return 4 * pi * r * r;
}
```

```
float sphere_area(float r){
  float pi = 3.13159;
  return 4 * pi * r * r;
}
```

Nanopass: Constant folding

```
float sphere_area(float r){
  float pi = calc_pi(5);
  return 4 * pi * r * r;
}
```

```
float sphere_area(float r){

  return 4 * 3.13159* r * r;
}
```

Nanopass: Constant folding

```
float sphere_area(float r){
  float pi = calc_pi(5);
  return 4 * pi * r * r;
}
```

```
float sphere_area(float r){

  return 12.52636 * r * r;
}
```

Nanopass: Constant folding

```
float sphere_area(float r){
  float pi = calc_pi(5);
  return 4 * pi * r * r;
}
```

```
float sphere_area(float r){

  return 12.52636 * r * r;
}
```

- Not essential
- For ‘performance’
- “Optimization” vs “Lowering”

Nanopass: if,while,… → goto

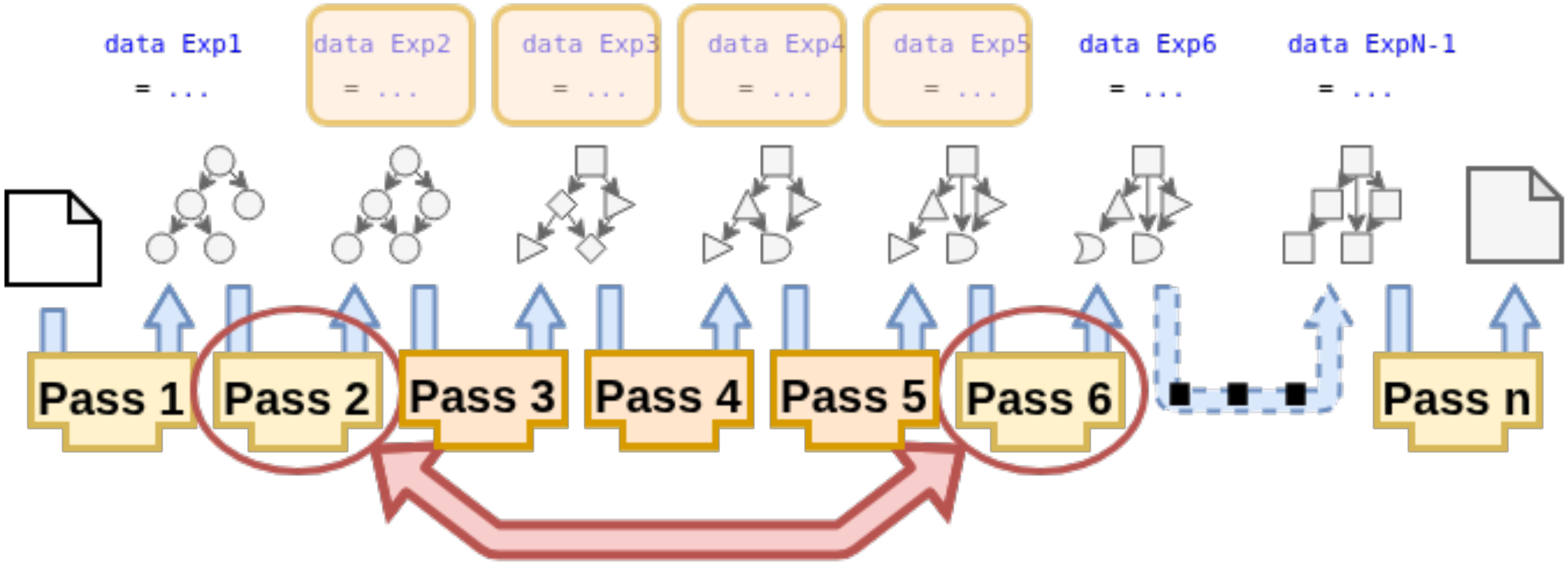
```
if {
  l.length > 7
}
then {
  u = insertion_sort(l)
}
else {
  u = quick_sort(l)
}
```

```
.L0:
  l.length > 7
  branch .L1 .L2
.L1:
  u = insertion_sort(l)
  goto .L3
.L2:
  u = quick_sort(l)
  goto .L3
.L3:
```

Skills

- Recognize common nanopasses
- Implement easy nanopasses
- Place nanopasses in compiler

Nanopass order hard to change



- Research: AST design for nanopass
- Meantime: design right order, early!