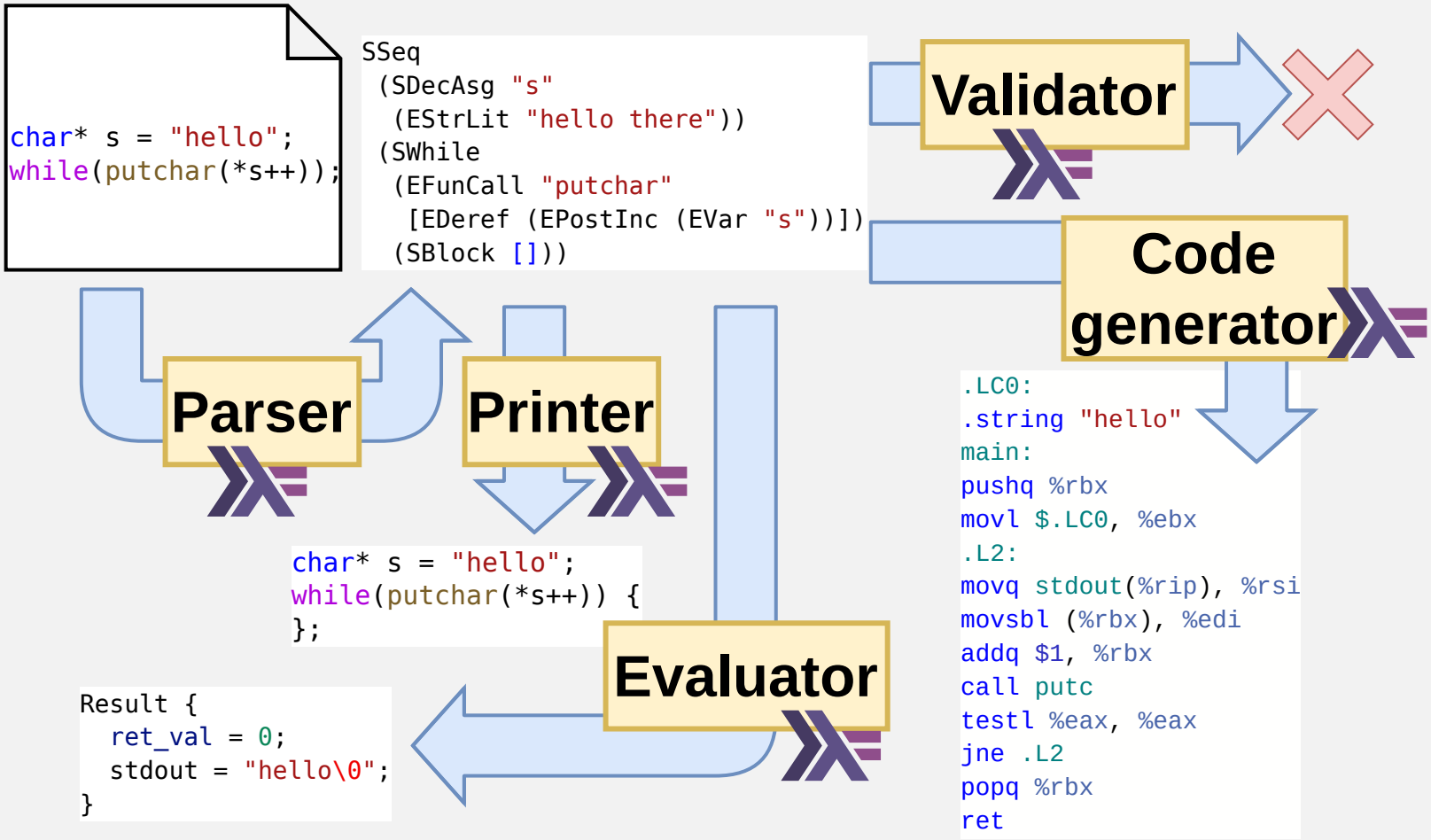
Lecture 13: Nanopass Compilation

Talen en Compilers 2023-2024, period 2

Lawrence Chonavel

Department of Information and Computing Sciences, Utrecht University

Compiler Architecture

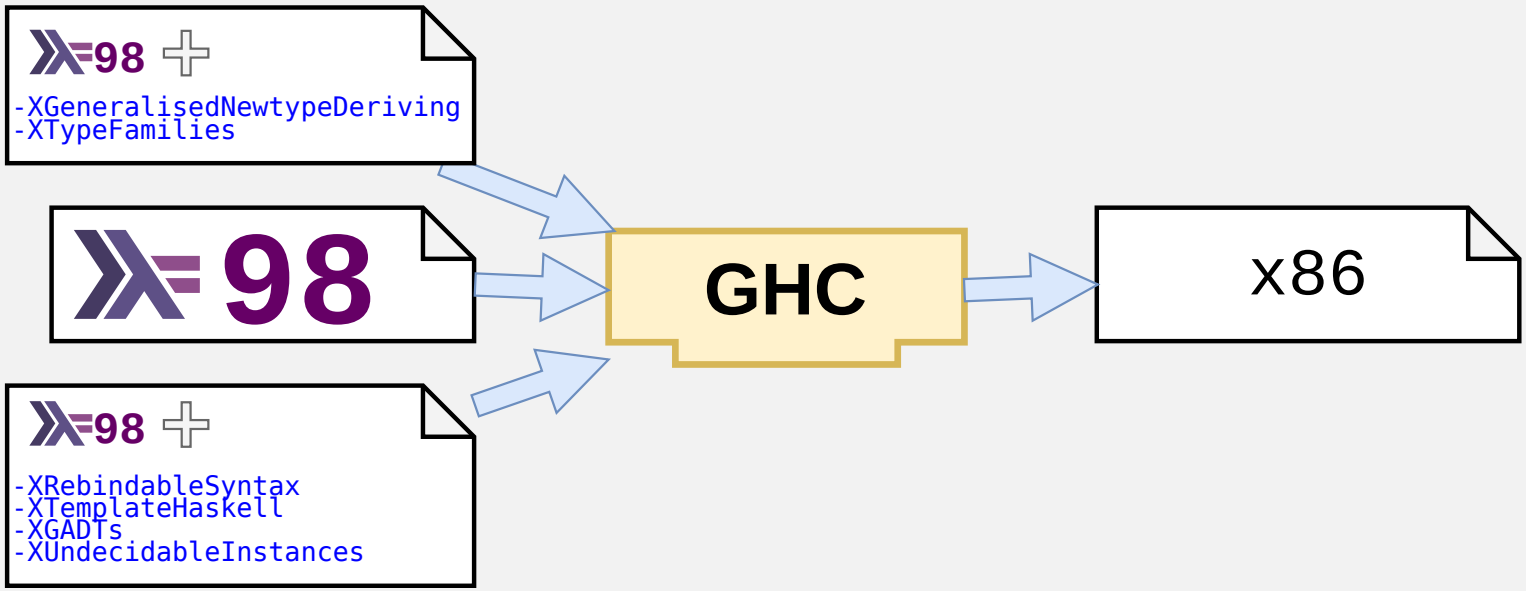


Compiler Architecture affects scalability

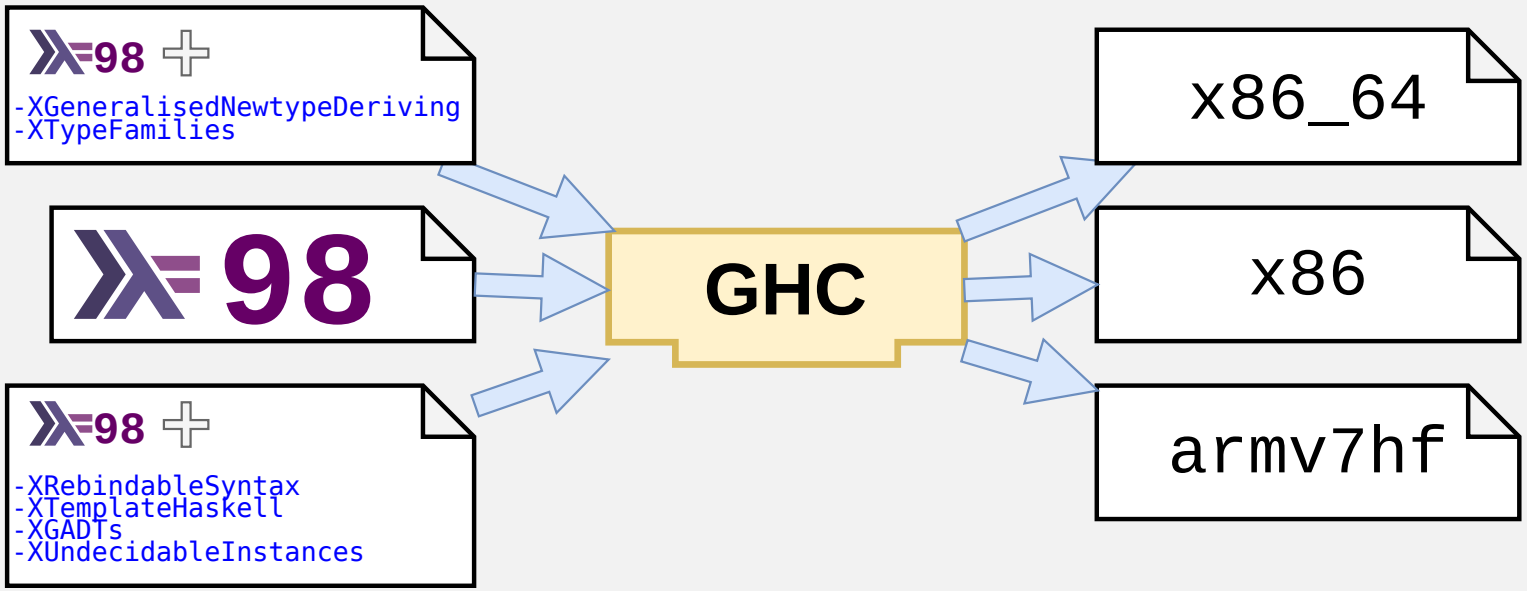
Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

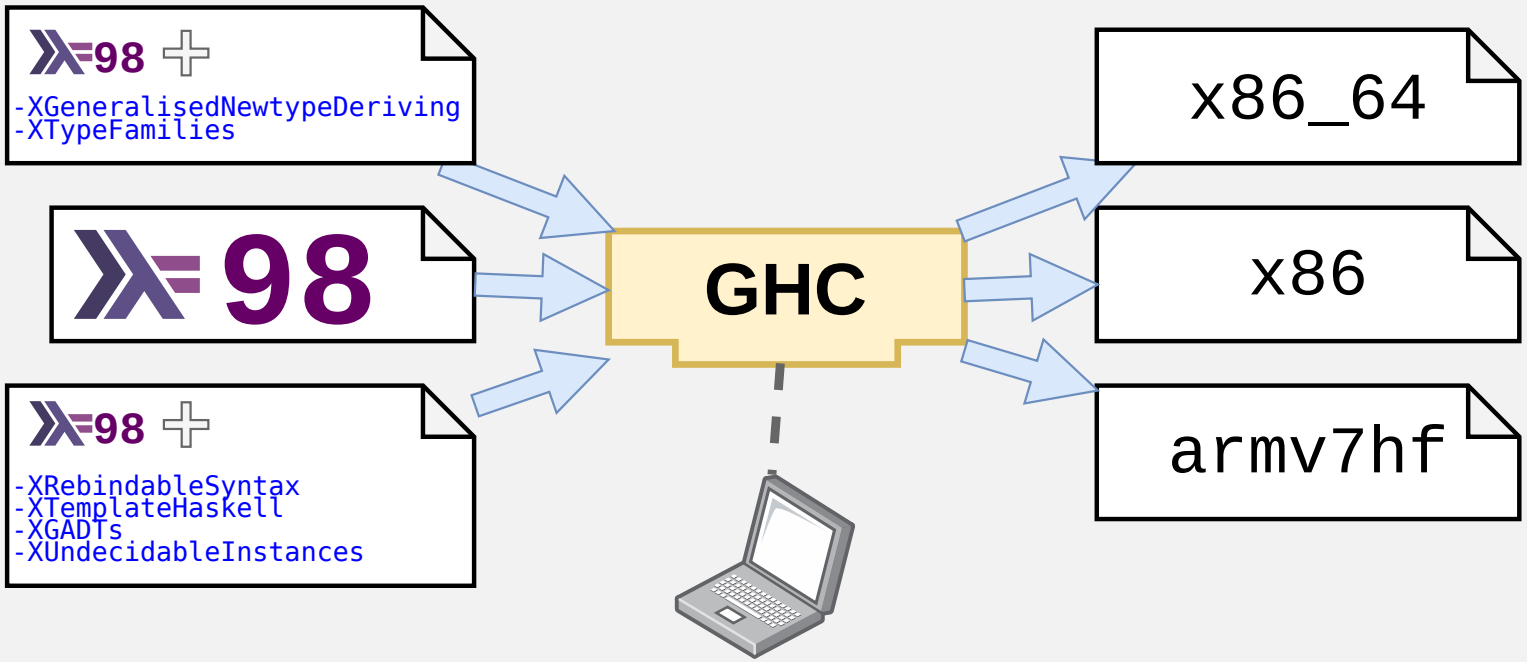
Compiler Architecture affects scalability



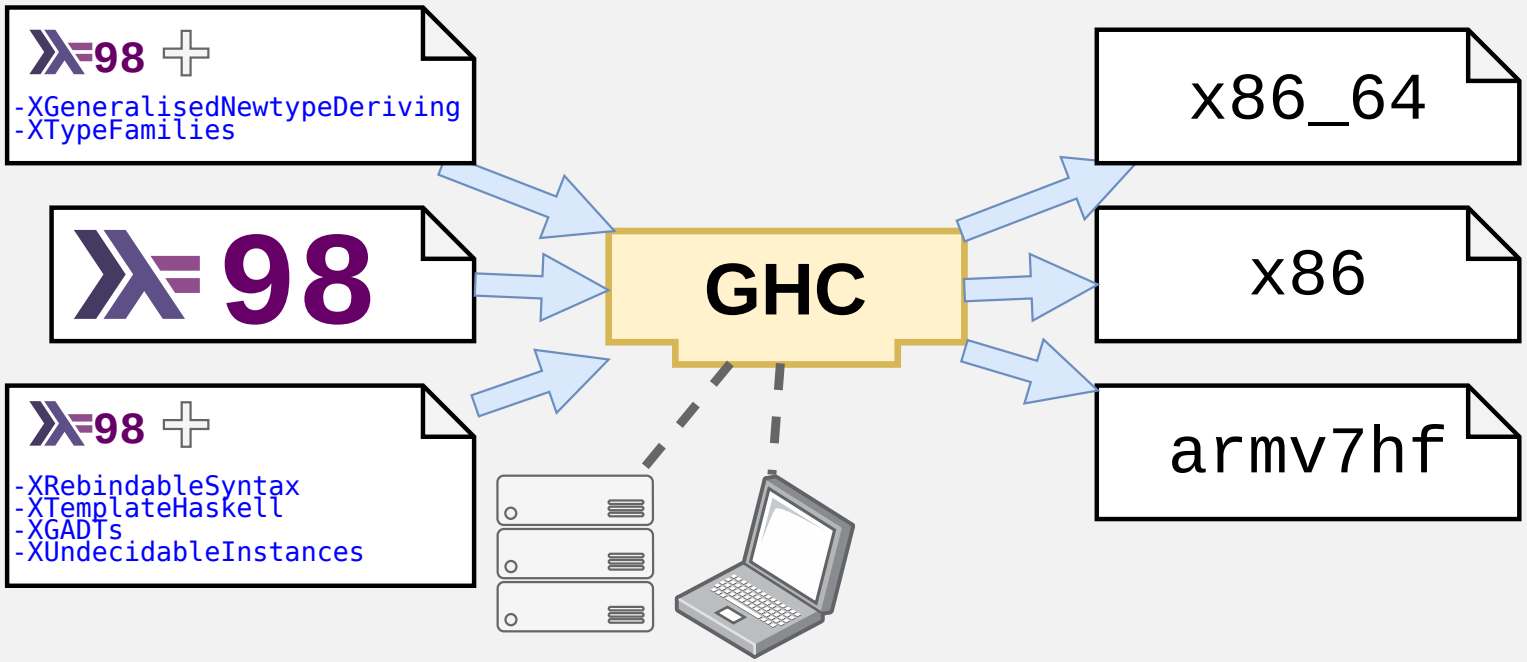
Compiler Architecture affects scalability



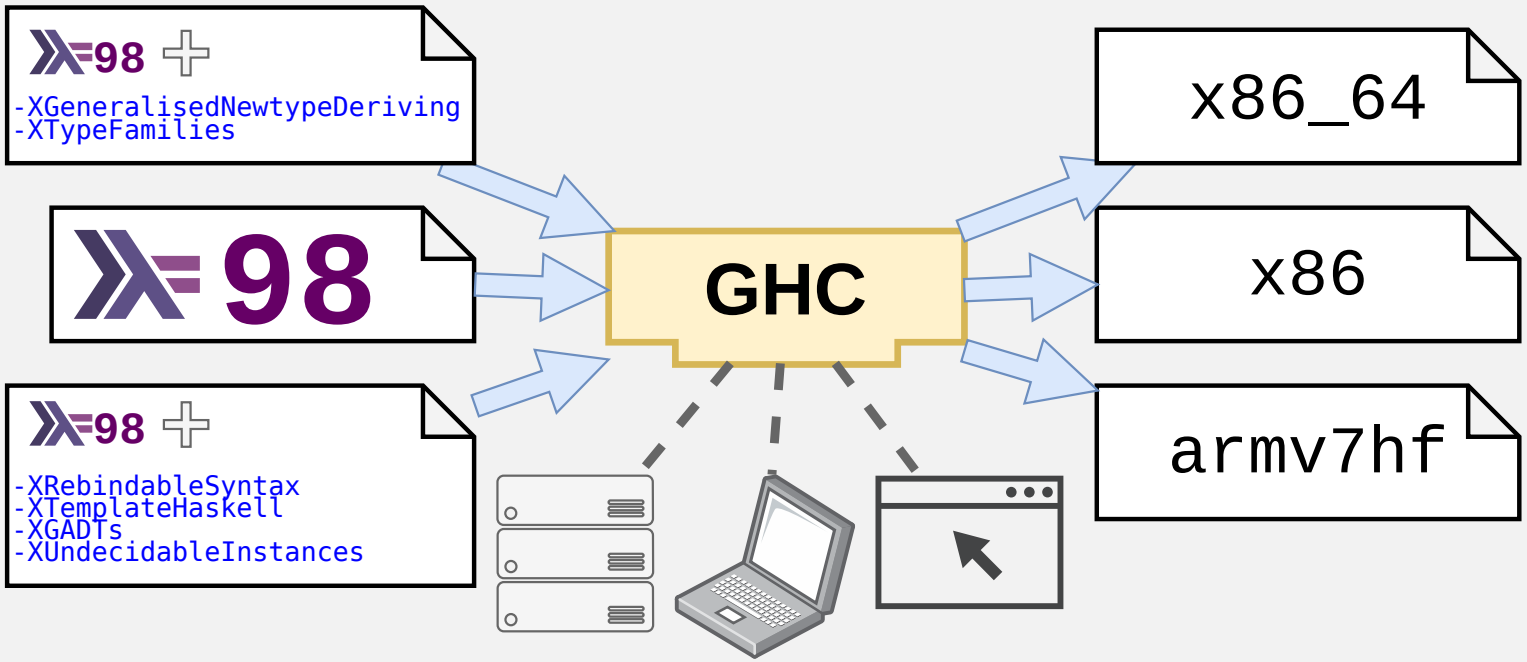
Compiler Architecture affects scalability



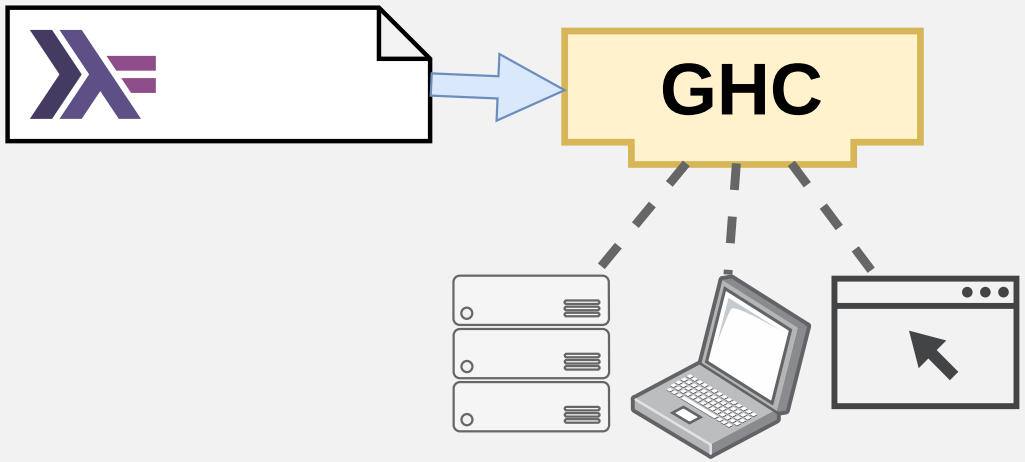
Compiler Architecture affects scalability



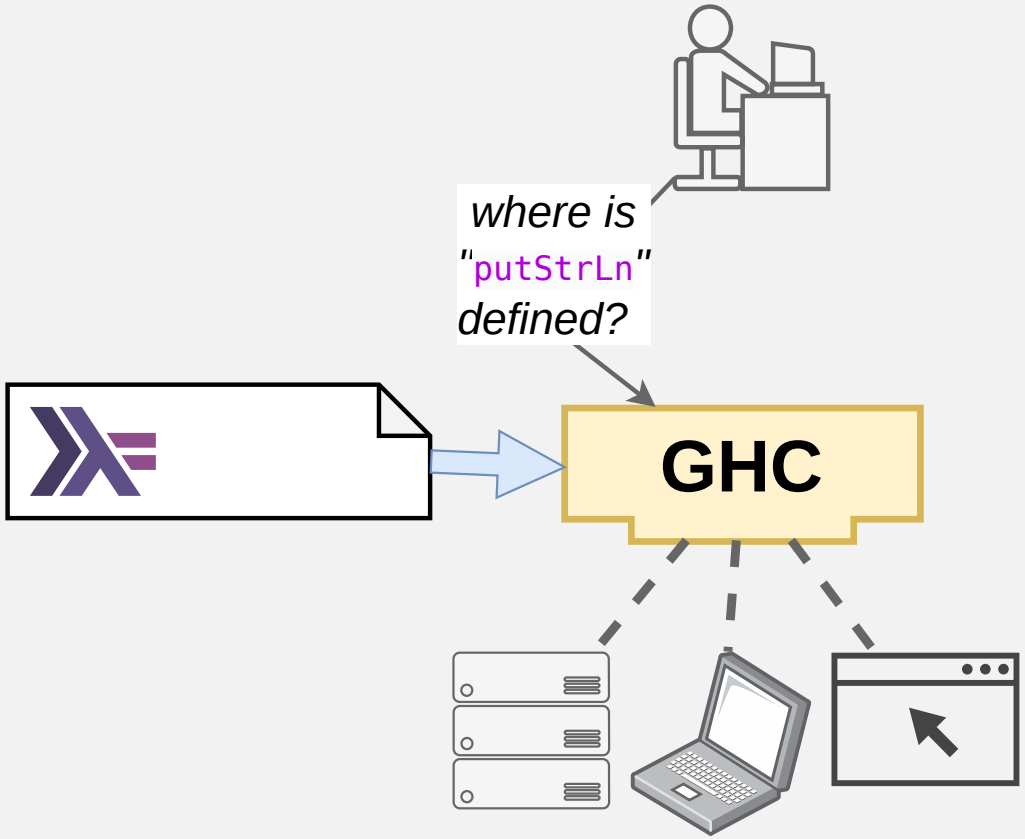
Compiler Architecture affects scalability



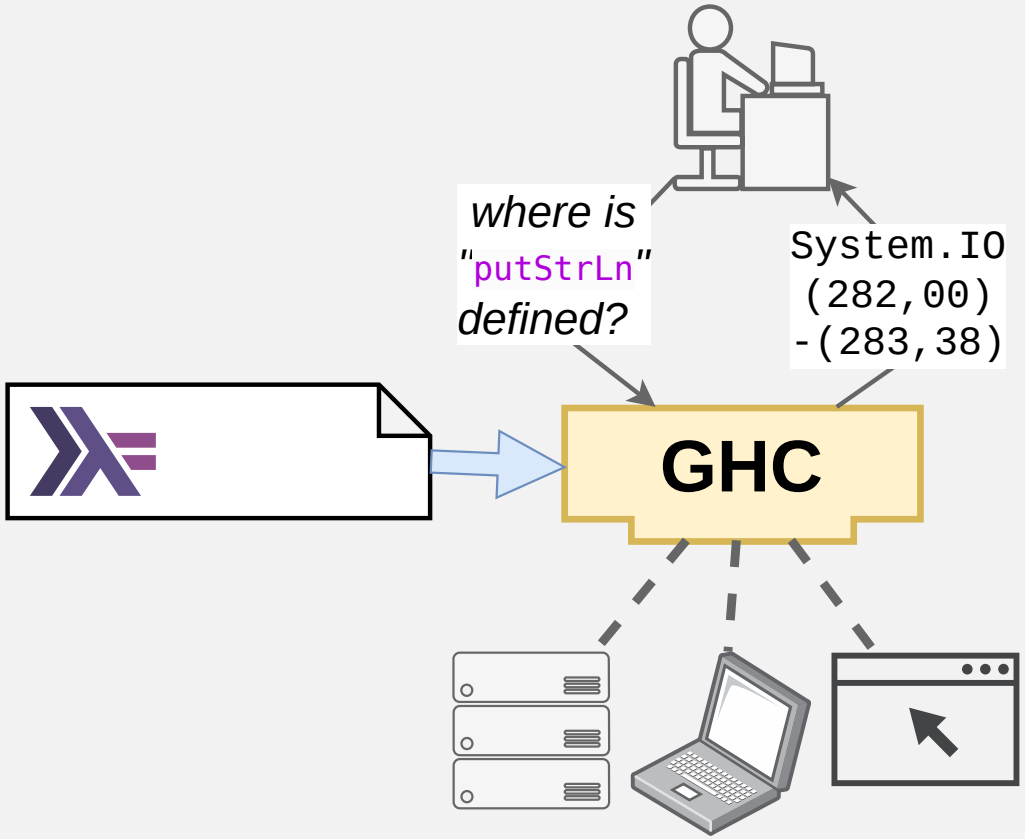
Compiler Architecture affects scalability



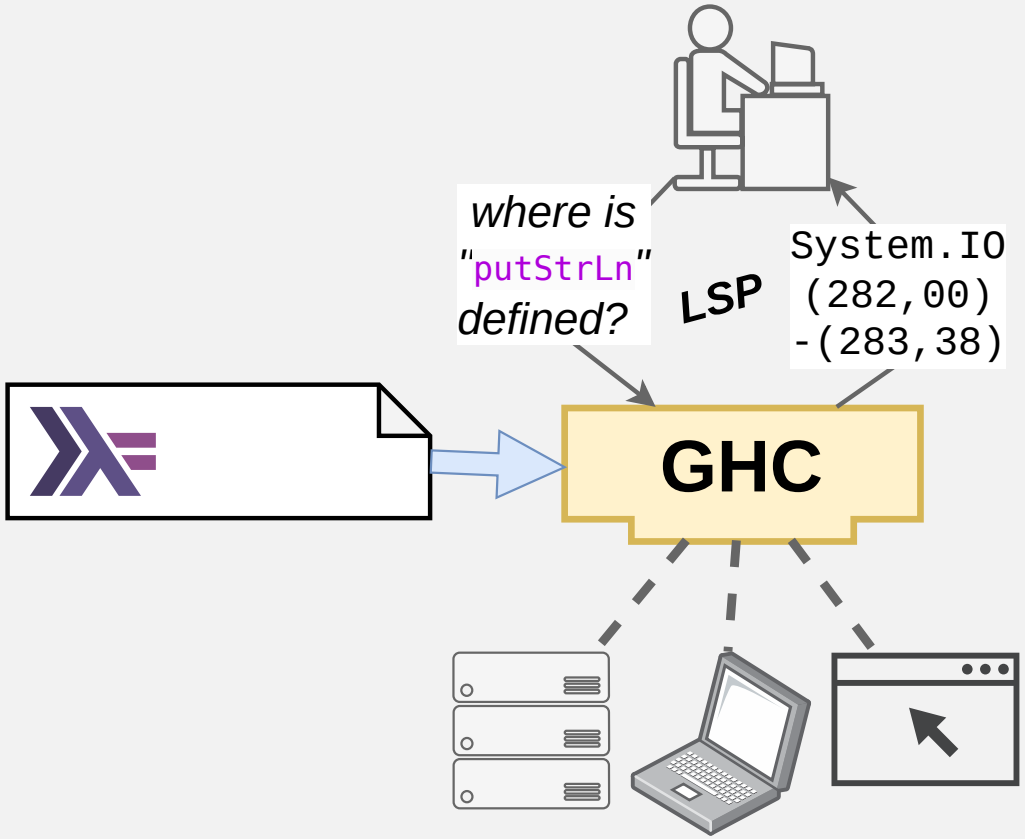
Compiler Architecture affects scalability



Compiler Architecture affects scalability



Compiler Architecture affects scalability



Compiler Architecture is hard

17 **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.)

permalink reddit 88% Upvoted

6 comments sorted by Confidence →

Search comments

3 ▼ 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.

2 ▼ u/heisenbug Jan 01 '12

Compiler Architecture is hard

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 →

Search comments

Universiteit

Compiler Architecture is hard

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

Universiteit

Compiler Architecture is hard

23 **r/Zig** • u/[deleted] • Mar 26 '20

Chris Fallin

Blog About Projects Academics & Publications

agda / agda Public

Notifications Fork 315 Star 2.3k

Code Issues 938 Pull requests 76 Actions Projects 9 Wiki Security

Heavy coupling of Haskell source modules #3512

New issue

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

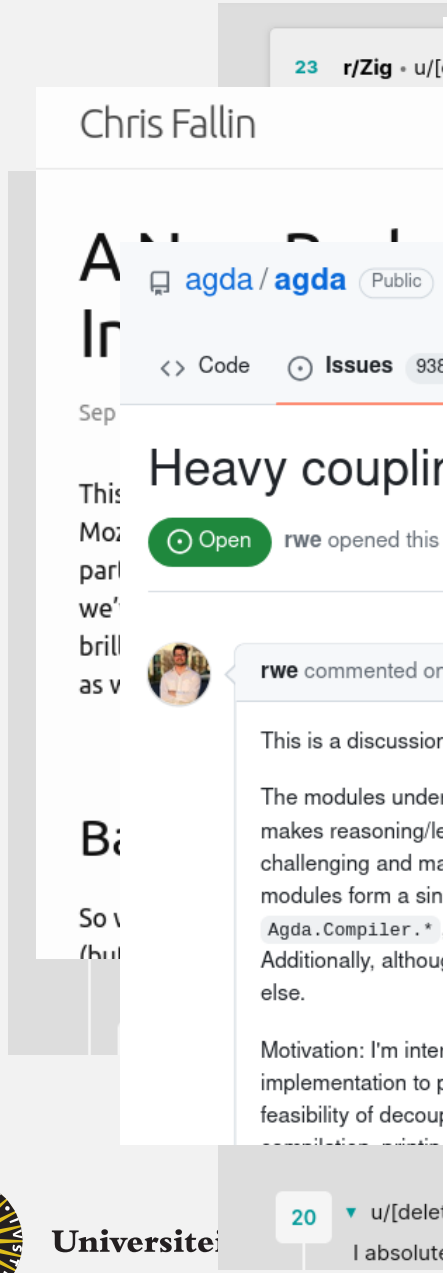
Decouple codebase

To do

Milestone

20 u/[deleted] Mar 26 '20

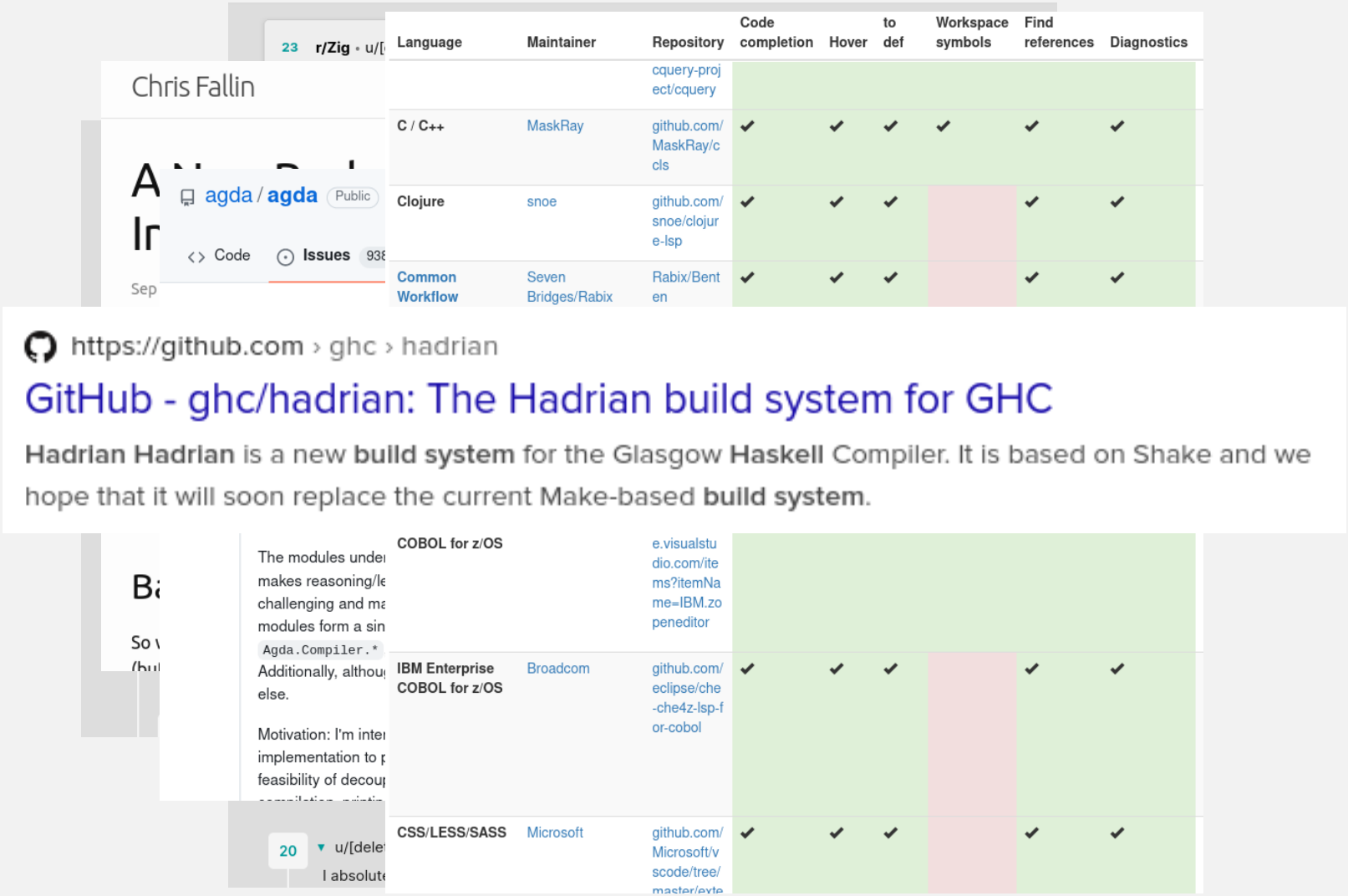
Compiler Architecture is hard



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	✔	✔	✔		✔	✔

Universite

Compiler Architecture is hard



23 r/Zig • u/[

Chris Fallin

agda / agda Public

Code Issues 938

Sep

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
makes reasoning/le
challenging and ma
modules form a sin
`Agda.Compiler.*`
Additionally, althou
else.

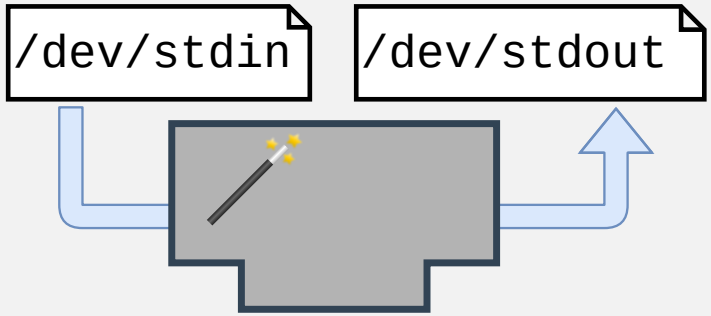
Motivation: I'm inter
implementation to p
feasibility of decoup

20 u/[dele

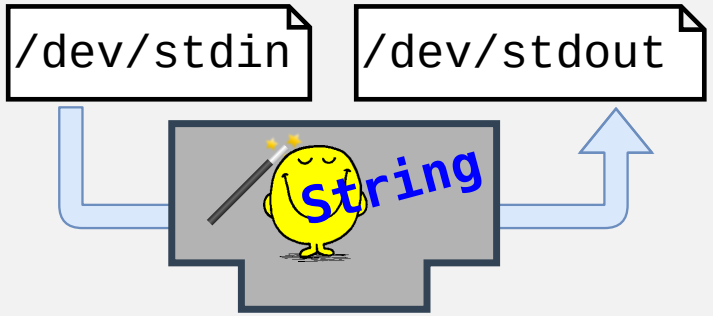
I absolute

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	Seven Bridges/Rabix	Rabix/Benen	✔	✔	✔		✔	✔
COBOL for z/OS		e.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	✔	✔	✔		✔	✔

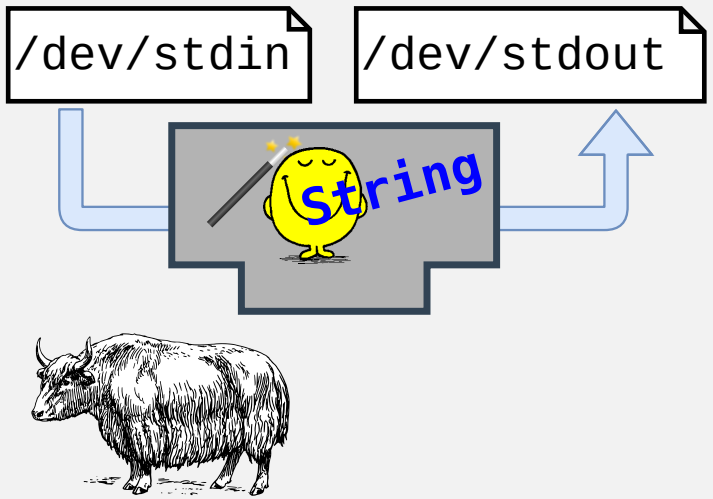
Compiler Architecture: monolith



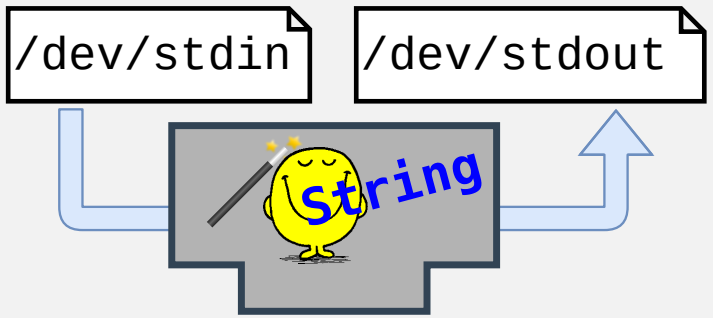
Compiler Architecture: monolith

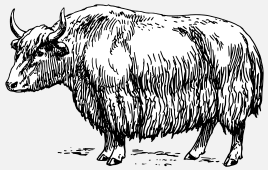


Compiler Architecture: monolith



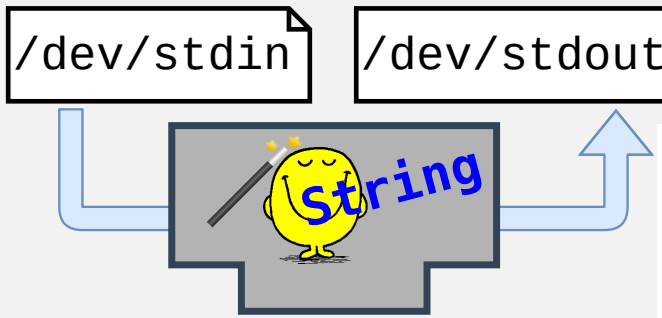
Compiler Architecture: monolith



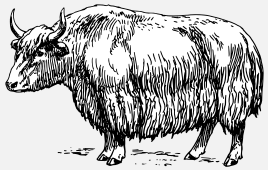


YACC

Compiler Architecture: monolith



LaTeX



YACC

5 Answers

Sorted by: Highest score (default)

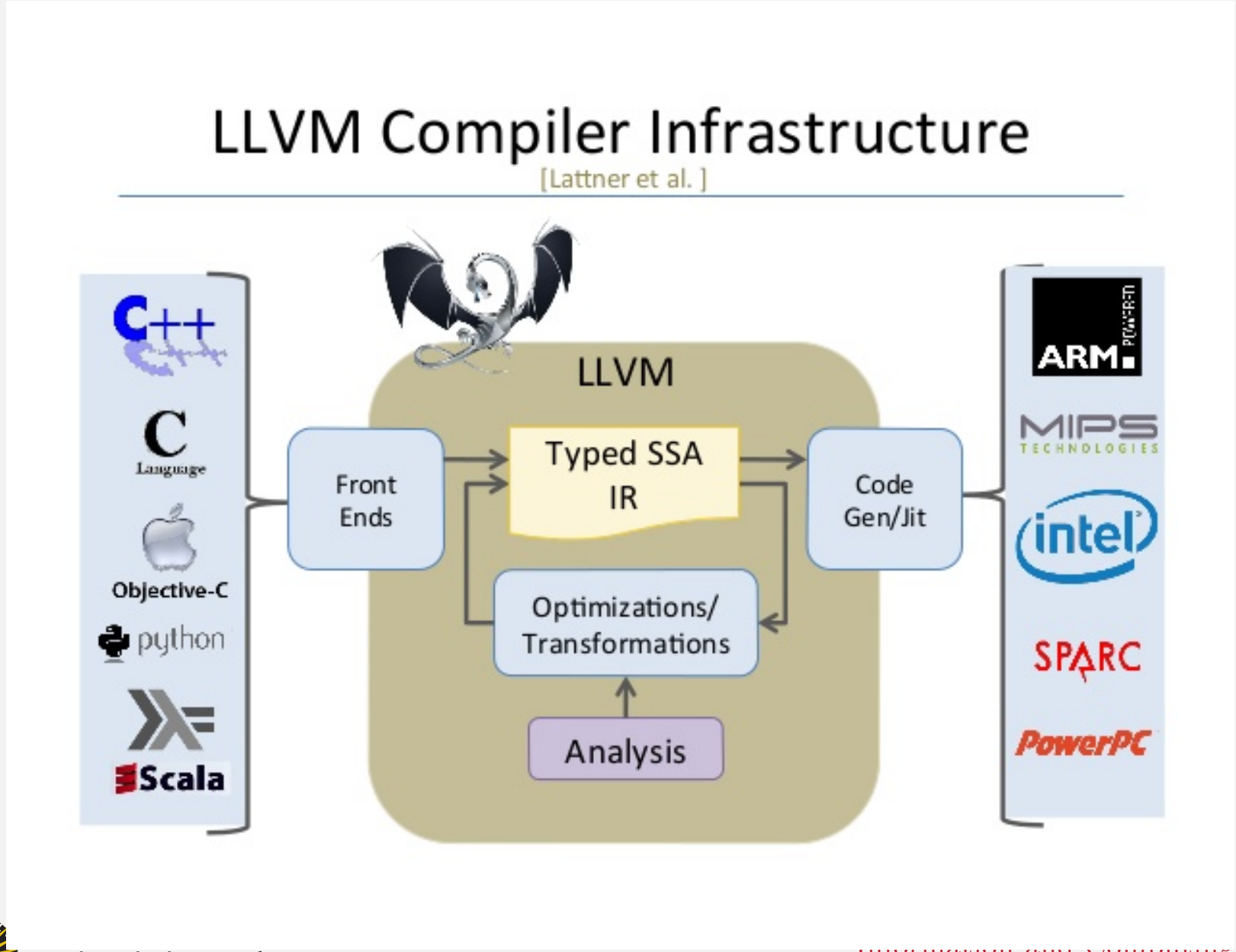
Parsing TeX is Turing complete

164

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 $f : \mathbb{Z} \to \mathbb{Z}$:

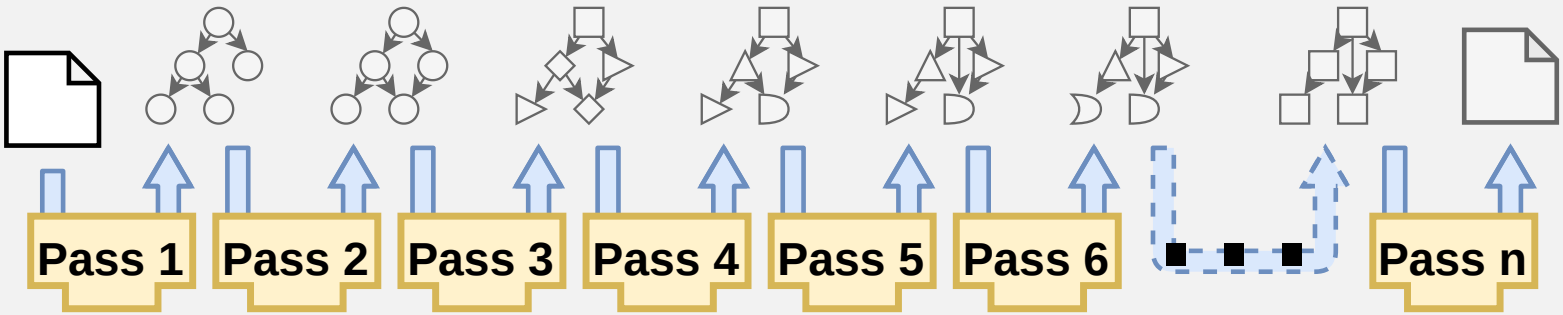
```
\def\TuringCompleteness#1#2{%
  \edef\Output{#1{#2}}%
  \ifnum\Output=0
```

Compiler Architecture: multi-pass



By P. Teoh on SO, https://stackoverflow.com/a/49081640

Compiler Architecture: nanopass

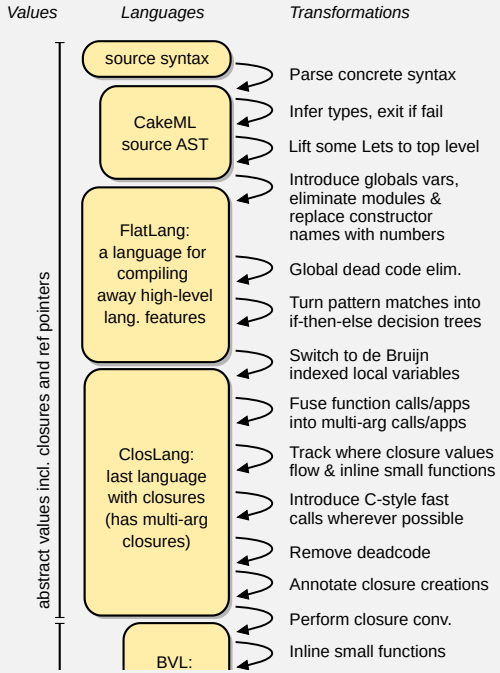


Compiler Architecture: nanopass

https://cakeml.org/

Compiler Architecture: nanopass

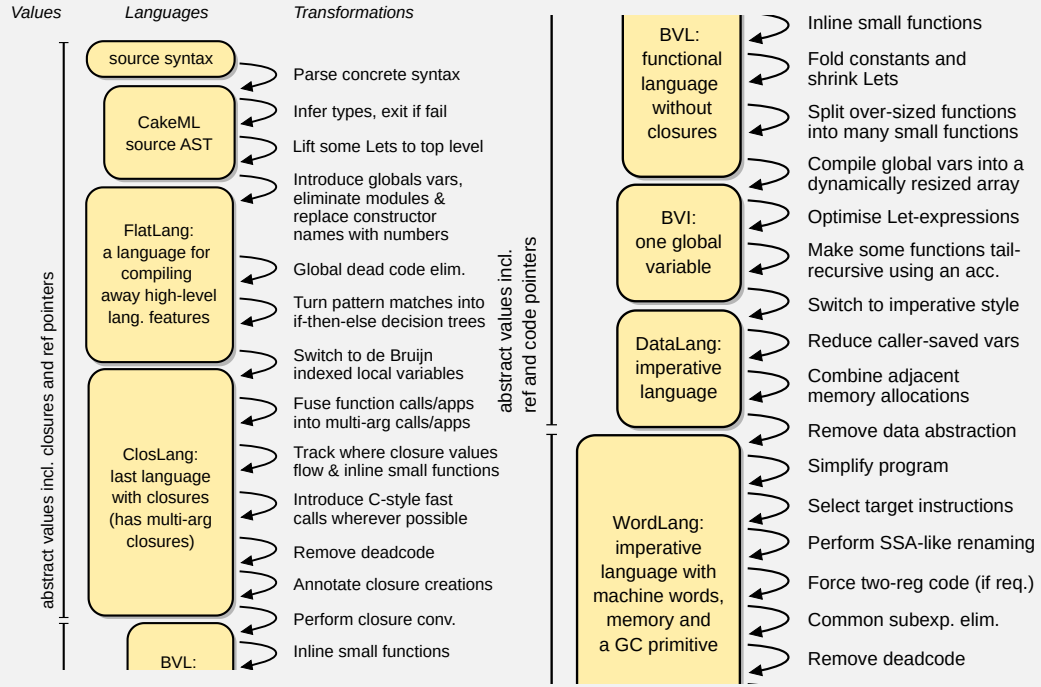
https://cakeml.org/



Compiler Architecture: nanopass

CAKEML
A Verified Implementation of ML

https://cakeml.org/



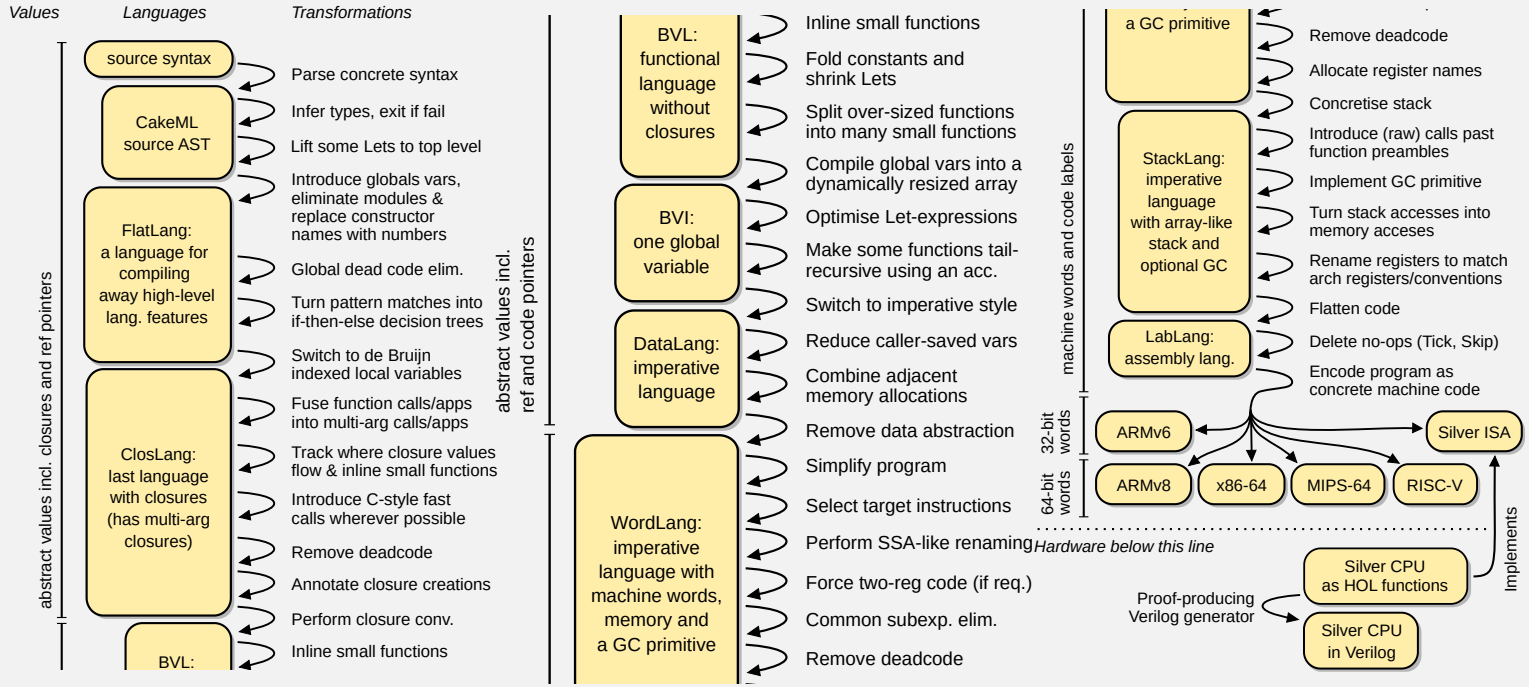
Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

Compiler Architecture: nanopass

CAKEML
A Verified Implementation of ML

https://cakeml.org/



Nanopass: Parse

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

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

Nanopass: Type-Check

```
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
real_solns a b c =
  let d = b**2 - 4*a*c :: Flo
  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++;
}
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Nanopass: for → while

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

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

```
for2while :: AstF → AstW
for2while (For (i,c,n) b) =
    i `Seq` While c (b `Seq` n)
for2while (Call f)     = Call f
for2while (Var i)      = Var i
for2while (Add e1 e2) = Add e1 e2
for2while (Seq e2 e2) = Seq e2 e2
for2while ...
```

Nanopass: for → while

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

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

```
for2while' = AstFAlg {
  for = λ (i,c,n) b →
    i `Seq` While c (e `Seq` n);
  var = λ i     → Var i;
  add = λ e1 e2 → Add e1 e2;
  seq = λ e1 e2 → Seq e1 e2;
  ...}
```

Nanopass: for → while

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

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

```
for2while' = AstFAlg {
  for = λ (i,c,n) b →
    i `Seq` While c (e `Seq` n);
  var = Var; add = Add; seq = Seq;
  ...}
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Nanopass: $\lambda \rightarrow$ class

```
int[] squares (int[] l) {

 return
  sum( map( (x => x*x)
          , l
    ));
}
```

```
int[] squares (int[] l) {

 return
  sum( map( new Lam43()
          , l
    ));
}

class Lam43 : Runnable {
 object run (object x) {
  return x*x;
 }

}
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Nanopass: $\lambda \rightarrow$ class

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

```
int[] squares (int[] l) {

 return
  sum( map( new Lam43()
          , l
     ));
}

class Lam43 : Runnable {
 object run (object x) {
  return x*x;
 }

}
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Nanopass: $\lambda \rightarrow$ 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 {
 object run (object x) {
  return q.log(x*x);
 }
 Logger q;
}
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Nanopass: class → struct

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

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

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

Nanopass: Insert Reference-Counting code

```
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);
}
```

Nanopass: Insert Reference-Counting code

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

Nanopass: Constant folding

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

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

Nanopass: Constant folding

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

```
float circle_area(float r){
  return 3.13159 * r * r;
}
```

Nanopass: Constant folding

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

```
float circle_area(float r){
  return 3.13159 * r * r;
}
```

- Not essential

Nanopass: Constant folding

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

```
float circle_area(float r){
  return 3.13159 * r * r;
}
```

- Not essential
- Might improve the code

Nanopass: Constant folding

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

```
float circle_area(float r){
  return 3.13159 * r * r;
}
```

- Not essential
- Might improve the code
- "Optimisation"

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Nanopass: Constant folding

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

```
float circle_area(float r){
  return 3.13159 * r * r;
}
```

- Not essential
- Might improve the code
- "Optimisation"
- More on Tuesday

Nanopass: if,while,... → goto

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

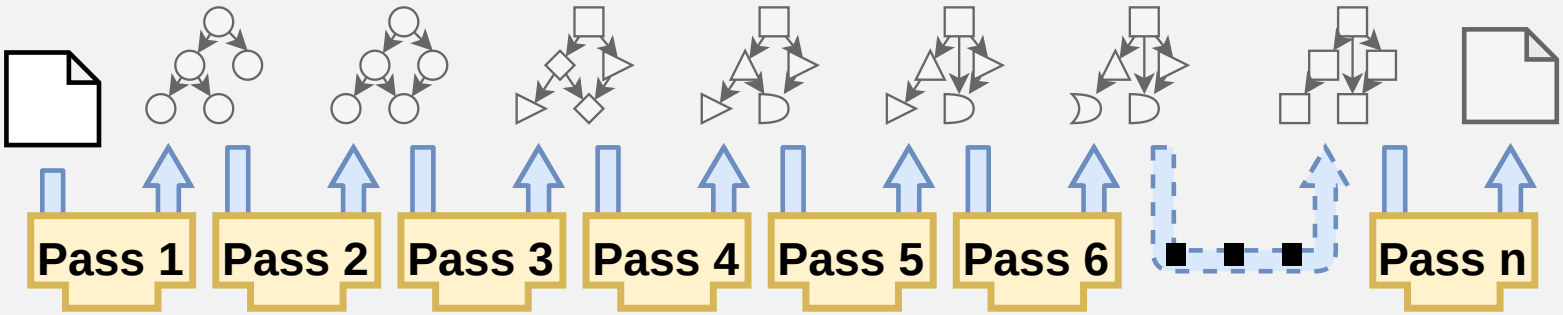
```
.L0:
  l.length > 7
  branch .L1 .L2
.L1:
  u = insertion_sort(l)
  goto .L3
.L2:
  u = quick_sort(l)
  goto .L3
.L3:
```

Nanopass: SSM instructions → x86_64 instructions

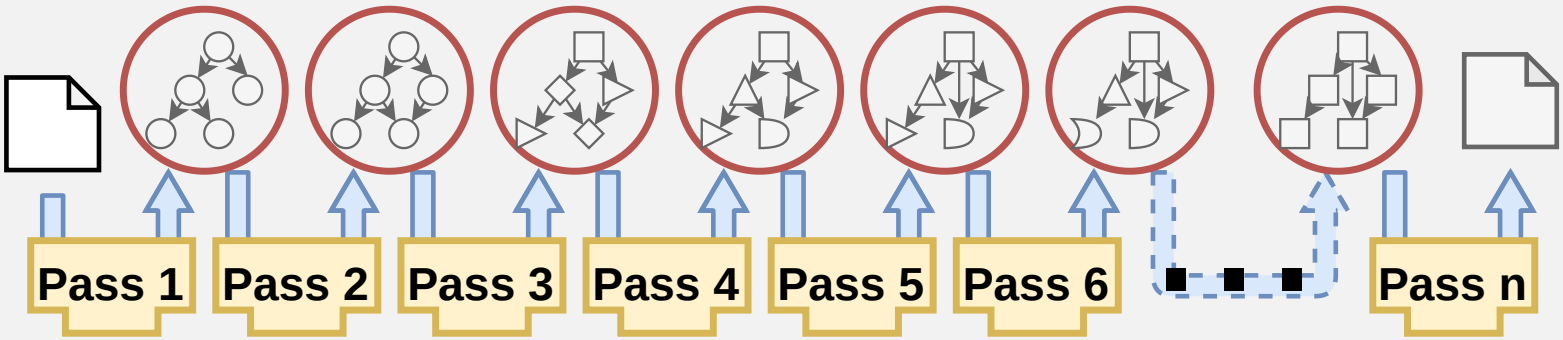
```
global.get  __stack_pointer      sub   rsp, 88
local.set   3                    mov   qword ptr [rsp + 8], rdi
i32.const   32                   mov   qword ptr [rsp + 16], rs
local.set   4                    mov   qword ptr [rsp + 24], rd
local.get   3                    mov   qword ptr [rsp + 32], rd
local.get   4                    cmp   rdx, 1
i32.sub                          ja    .LBB0_2
local.set   5                    mov   rax, qword ptr [rsp + 32
local.get   5                    mov   rcx, qword ptr [rsp + 24
global.set  __stack_pointer      mov   rdx, qword ptr [rsp + 8]
i32.const   1                    mov   rsi, qword ptr [rsp + 16
local.set   6                    mov   qword ptr [rcx], rsi
local.get   2                    mov   qword ptr [rcx + 8], rdi
local.set   7                    mov   rsi, qword ptr [rip + .L
local.get   6                    mov   rdx, qword ptr [rip + .L
local.set   8                    mov   qword ptr [rcx + 32], rs
local.get   7                    mov   qword ptr [rcx + 40], rd
local.get   8                    lea   rdx, [rip + .L__unnamed
```

rcx,
ing
esi]

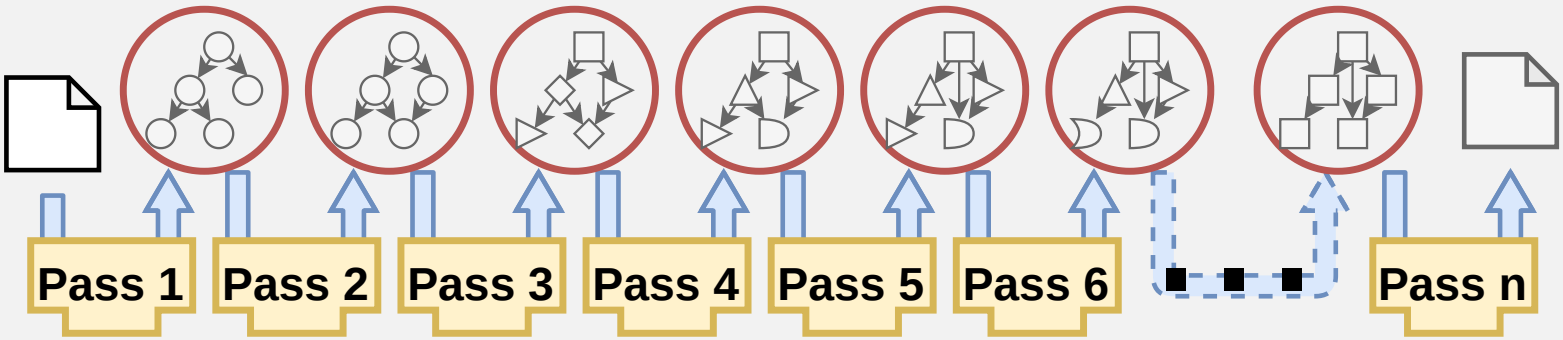
Nanopass AST(s)?



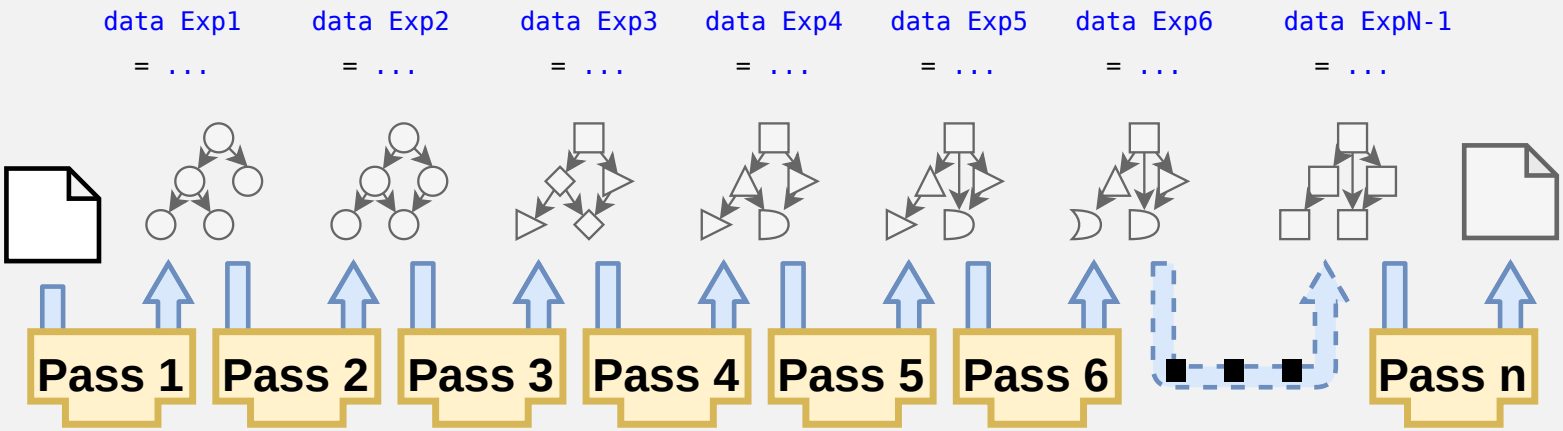
Nanopass AST(s)?



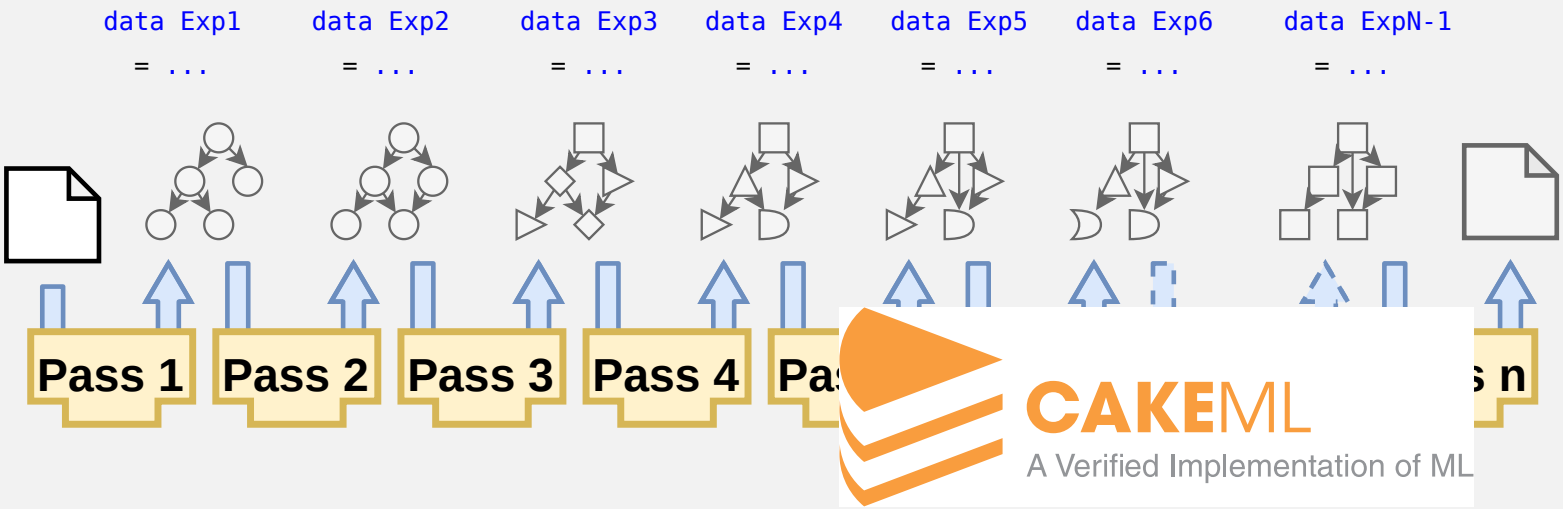
Design 1: Many ASTs



Design 1: Many ASTs



Design 1: Many ASTs



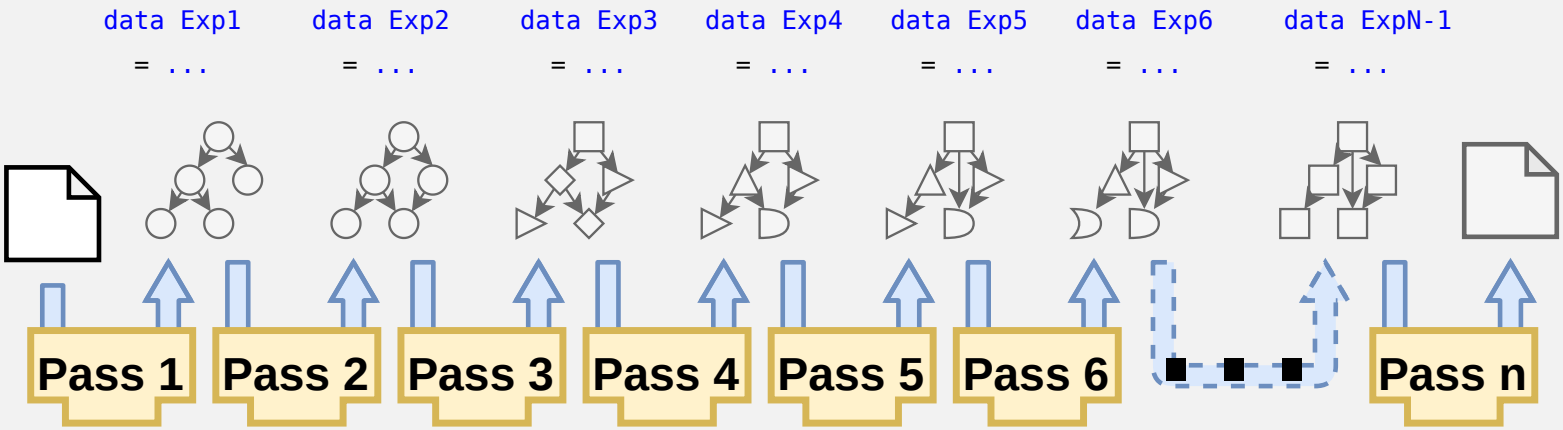
Design 1: Many ASTs ⇒ 💩 Repetition

```
data Exp3
  = For (Exp3,Exp3,Exp3) Exp3
  | While Exp3 Exp3
  | Call Func
  | Var Var
  | Add Exp3 Exp3
  | Seq Exp3 Exp3
  | ...
```

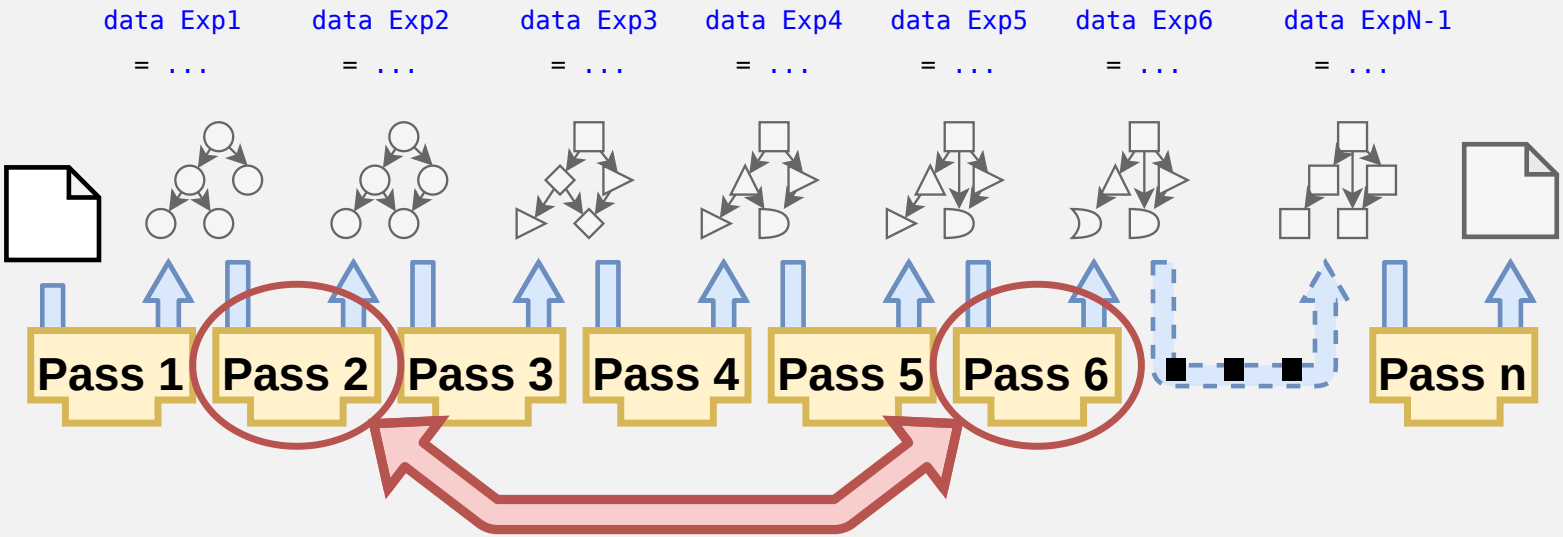
```
data Exp4

  = While Exp4 Exp4
  | Call Func
  | Var Var
  | Add Exp4 Exp4
  | Seq Exp4 Exp4
  | ...
```

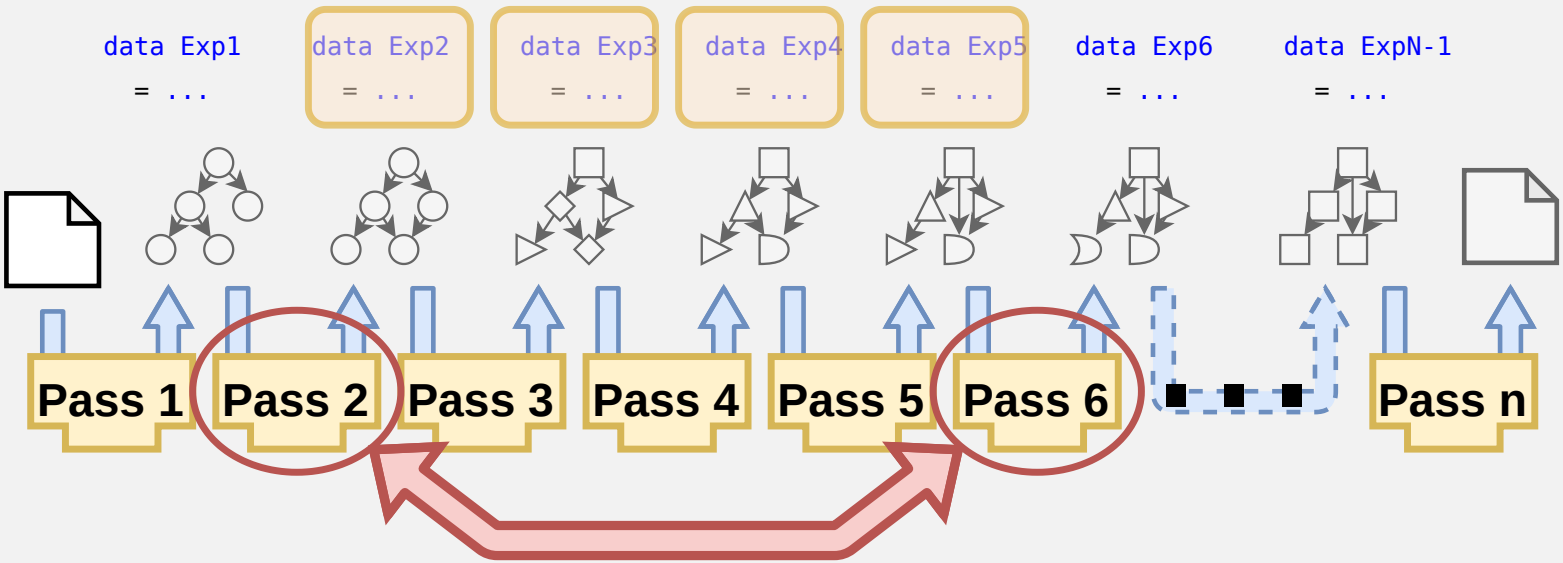
Design 1: Many ASTs ⇒ 💩 Brittle pass order



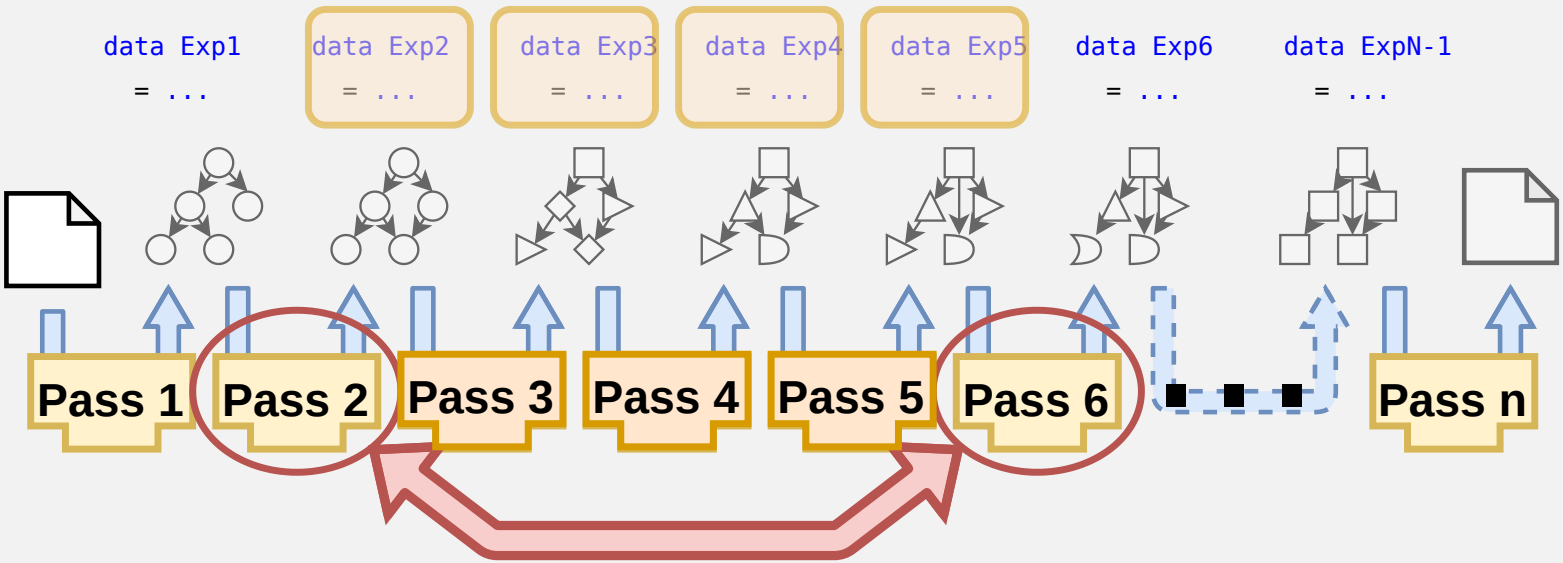
Design 1: Many ASTs ⇒ 💩 Brittle pass order



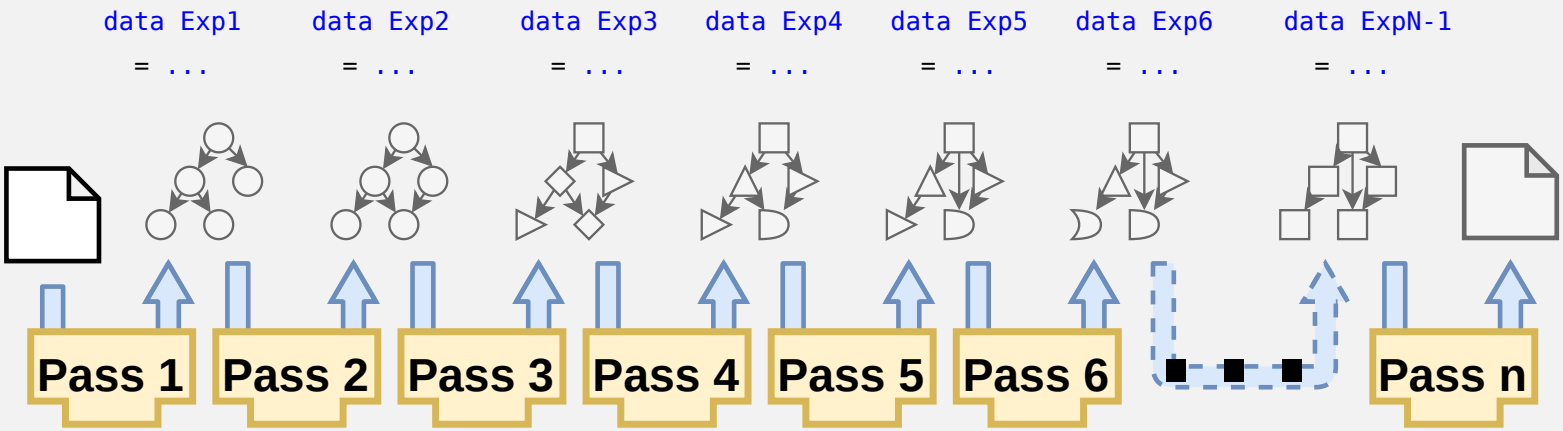
Design 1: Many ASTs ⇒ 💩 Brittle pass order



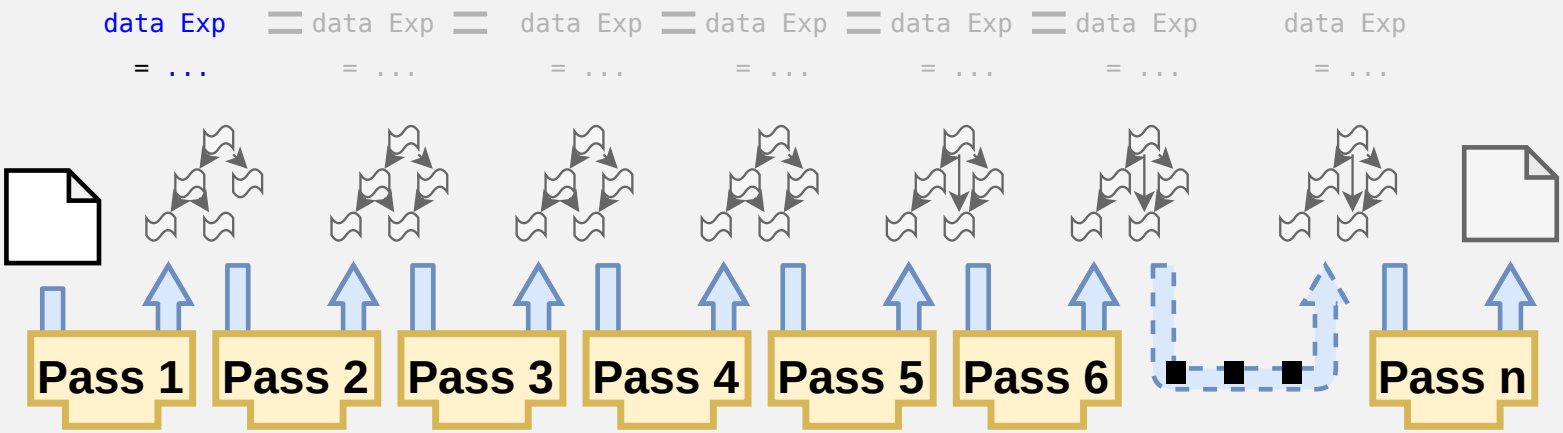
Design 1: Many ASTs ⇒ 💩 Brittle pass order



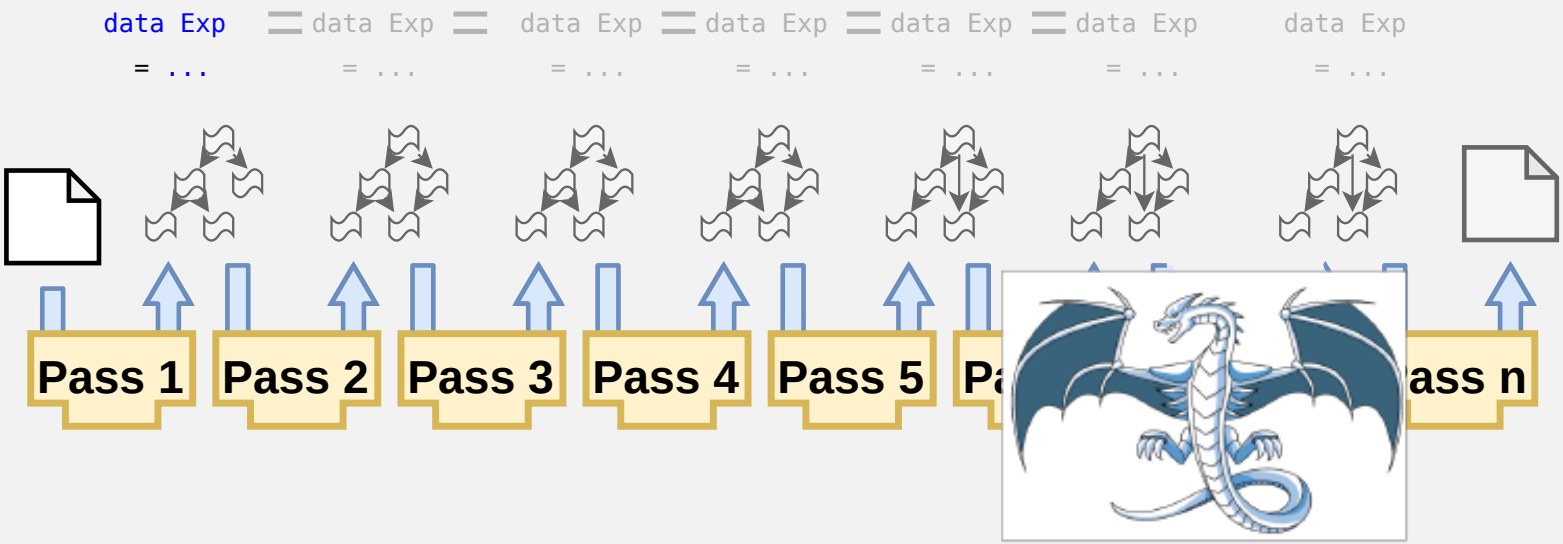
Design 2: One AST



Design 2: One AST



Design 2: One AST



Design 2: One AST

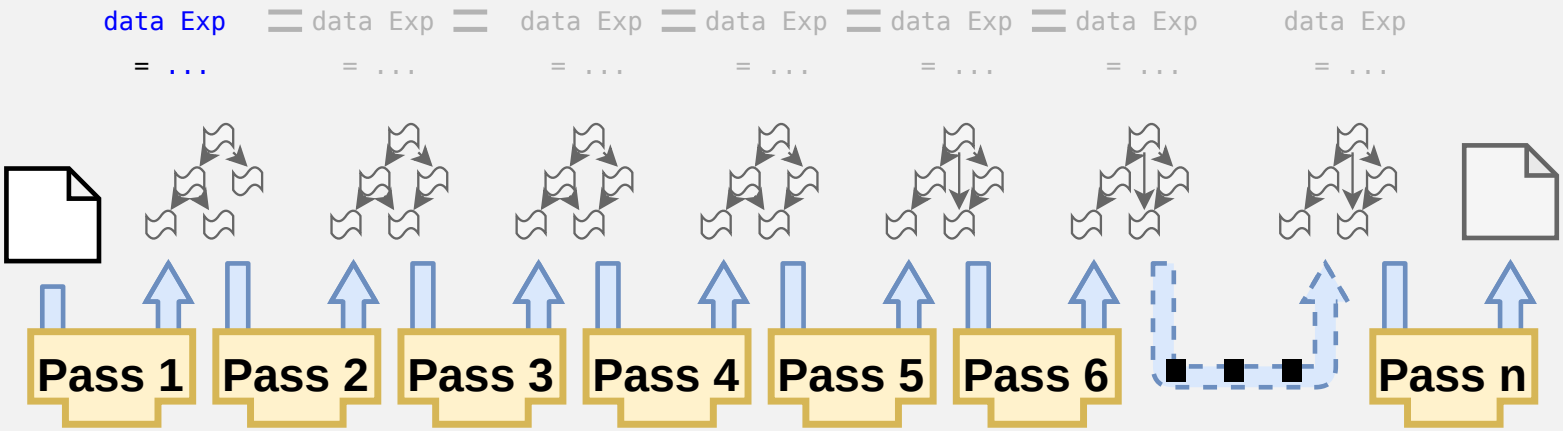
```
data Exp
  = Raw String
  | If Exp Exp Exp
  | Goto Label
  | Instr SSM.Instr
  | Typed Type Exp
  | For (Exp,Exp,Exp) Exp
  | While Exp Exp
  | ...
```

Design 2: One AST ⇒ 💩 No type safety

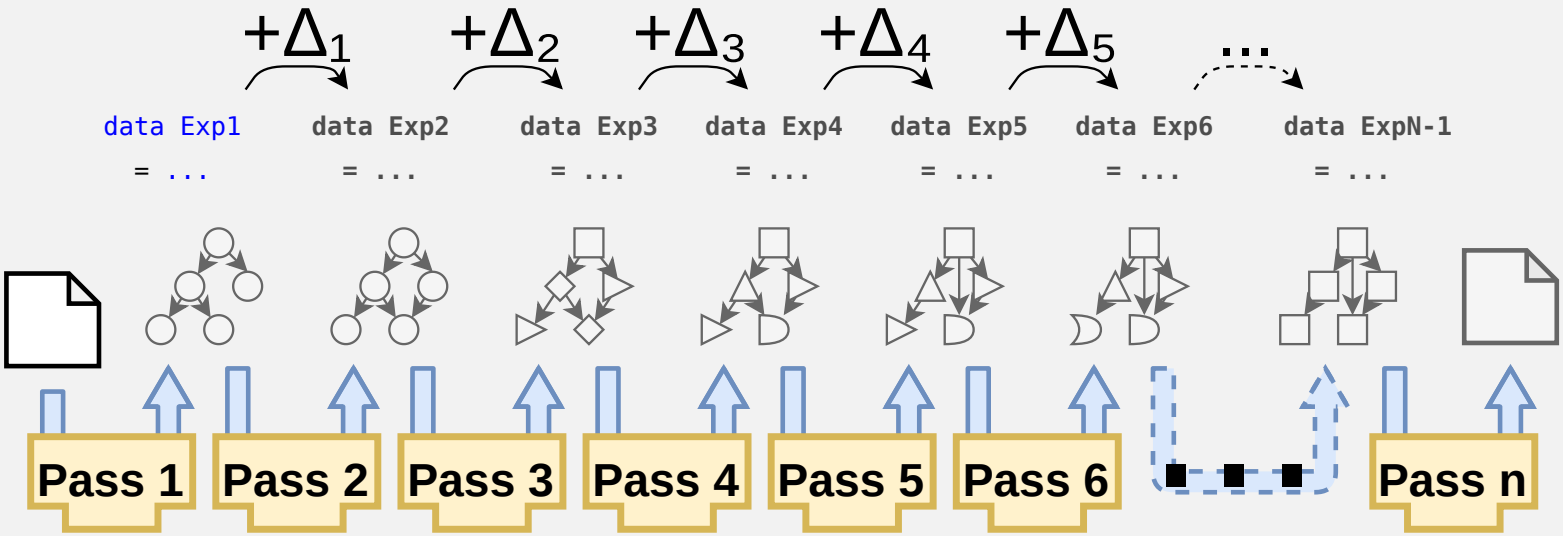
```
for2while :: Exp → Exp
for2while (For (i,c,n) b) =
    i `Seq` While c (b `Seq` n)
for2while (Call f)     = Call f
for2while (Var i)      = Var i
for2while (Add e1 e2) = Add e1 e2
for2while (Seq e2 e2) = Seq e2 e2
for2while ...
```

```
<interactive>:10:18: warning: [-Wincomplete-patterns]
    Pattern match(es) are non-exhaustive
    In an equation for `for2while':
        Patterns of type `Exp' not matched:
          rawCode :: String -> Exp
          goto    :: Label -> Exp
          instr   :: SSM.Instr -> Exp
          ...
```

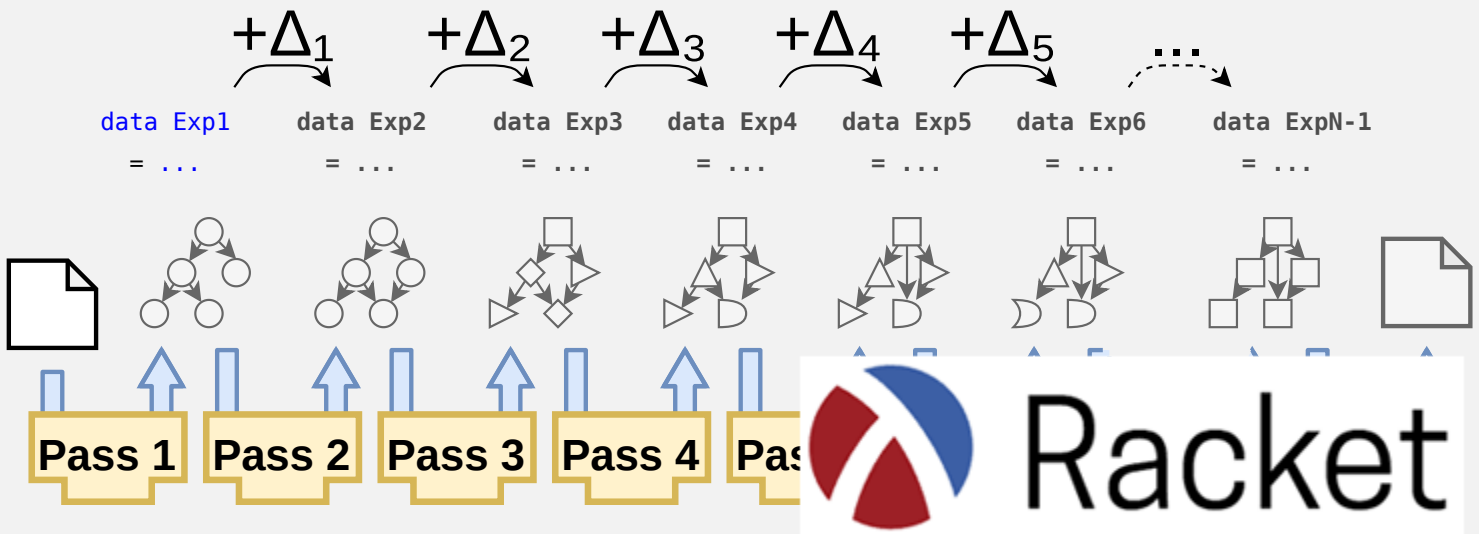
Design 3: Generics



Design 3: Generics



Design 3: Generics



Design 3: Generics ⇒ 💩 Not Haskell

```
{-# LANGUAGE TemplateHaskell #-}
import Vaporware.Generics.Library
```

Design 3: Generics ⇒ 💩 Not Haskell

```
{-# LANGUAGE TemplateHaskell #-}
import Vaporware.Generics.Library
```

```
patch4 :: ΔData
patch4 = \exp ->
  [ RemoveConstructor "For" [(exp,exp,exp),exp]
  , AddConstructor "While" [exp, exp]
  ]

data Exp4 = $(patch_datatype Exp3 patch4)
```

Design 3: Generics ⇒ 💩 Not Haskell

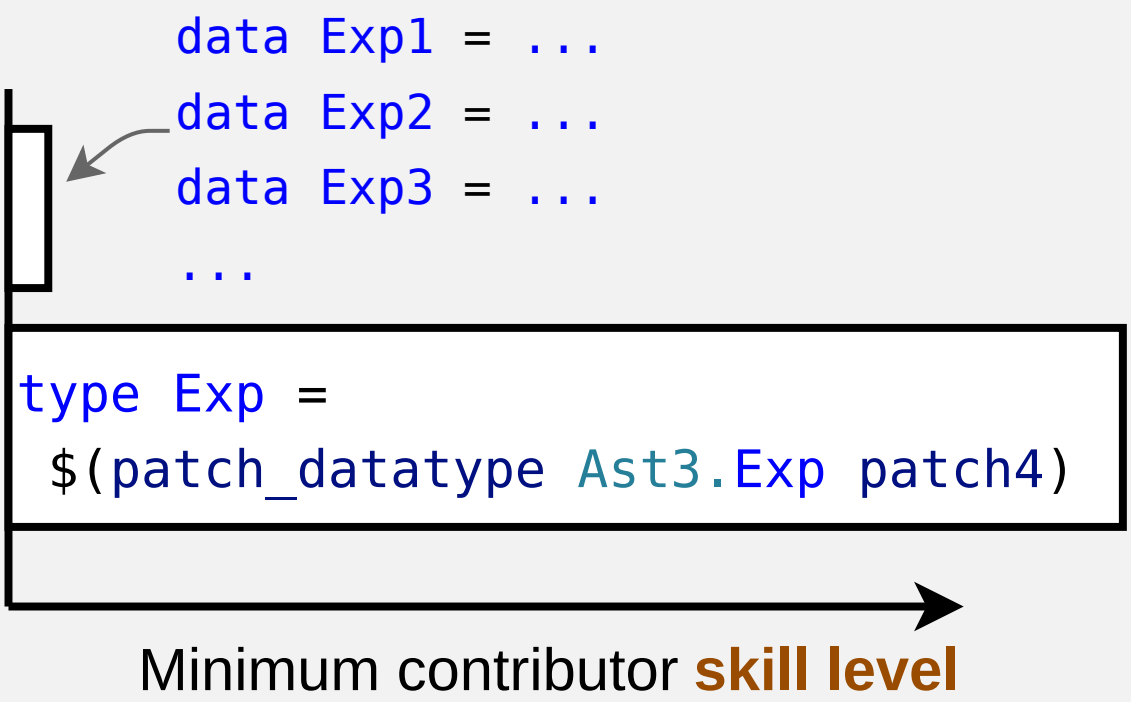
```
{-# LANGUAGE TemplateHaskell #-}
import Vaporware.Generics.Library
```

```
patch4 :: ΔData
patch4 = \exp ->
  [ RemoveConstructor "For" [(exp,exp,exp),exp]
  , AddConstructor "While" [exp, exp]
  ]

data Exp4 = $(patch_datatype Exp3 patch4)
```

```
for2while :: Ast3.Exp
for2while (For (i,c,n) b) =
    i `Seq` While c (b `Seq` n)
for2while _ = $(generate_fold_boilerplate)
```

Design 3: Generics ⇒ 💩 Complicated



Design 3: Generics ⇒ 💩 No Types?

- 👩‍🔬🔥 Research!

Design 3: Generics ⇒ 💩 No Types?

- 👰🔥 Research!
- 💍 Ornaments

Design 3: Generics ⇒ 💩 No Types?

- 👰🔥 Research!
- 💍 Ornaments
- 🎁 Data type description EDSLs

Design 3: Generics ⇒ 💩 No Types?

- 👰🔥 Research!
- 💍 Ornaments
- 🎁 Data type description EDSLs
- 📦 Container extension

Design 3: Generics ⇒ 💩 No Types?

- 👰🔥 Research!
- 💍 Ornaments
- 🎁 Data type description EDSLs
- 📦 Container extension
- https://personal.cis.strath.ac.uk/conor.mcbride/pub/OAAO/LitOrn.pdf

Design 3: Generics ⇒ 💩 No Types?

- 👰🔥 Research!
- 💍 Ornaments
- 🎁 Data type description EDSLs
- 📦 Container extension
- https://personal.cis.strath.ac.uk/conor.mcbride/pub/OAAO/LitOrn.pdf
- https://dl.acm.org/doi/pdf/10.1145/2502409.2502413

Design 3: Generics ⇒ 💩 No Types?

- 👰🔥 Research!
- 💍 Ornaments
- 🎁 Data type description EDSLs
- 📦 Container extension
- https://personal.cis.strath.ac.uk/conor.mcbride/pub/OAAO/LitOrn.pdf
- https://dl.acm.org/doi/pdf/10.1145/2502409.2502413
- https://doi.org/10.1017/S0956796814000069

Design 3: Generics ⇒ 💩 No Types?

- 👰🔥 Research!
- 💍 Ornaments
- 🎁 Data type description EDSLs
- 📦 Container extension
- https://personal.cis.strath.ac.uk/conor.mcbride/pub/OAAO/LitOrn.pdf
- https://dl.acm.org/doi/pdf/10.1145/2502409.2502413
- https://doi.org/10.1017/S0956796814000069
- https://dl.acm.org/doi/pdf/10.1145/2633628.2633631

Design 3: Generics ⇒ 💩 No Types?

- 👰🔥 Research!
- 💍 Ornaments
- 🎁 Data type description EDSLs
- 📦 Container extension
- https://personal.cis.strath.ac.uk/conor.mcbride/pub/OAAO/LitOrn.pdf
- https://dl.acm.org/doi/pdf/10.1145/2502409.2502413
- https://doi.org/10.1017/S0956796814000069
- https://dl.acm.org/doi/pdf/10.1145/2633628.2633631
- https://doi.org/10.1017/S0956796816000356

Design 3: Generics ⇒ 💩 No Types?

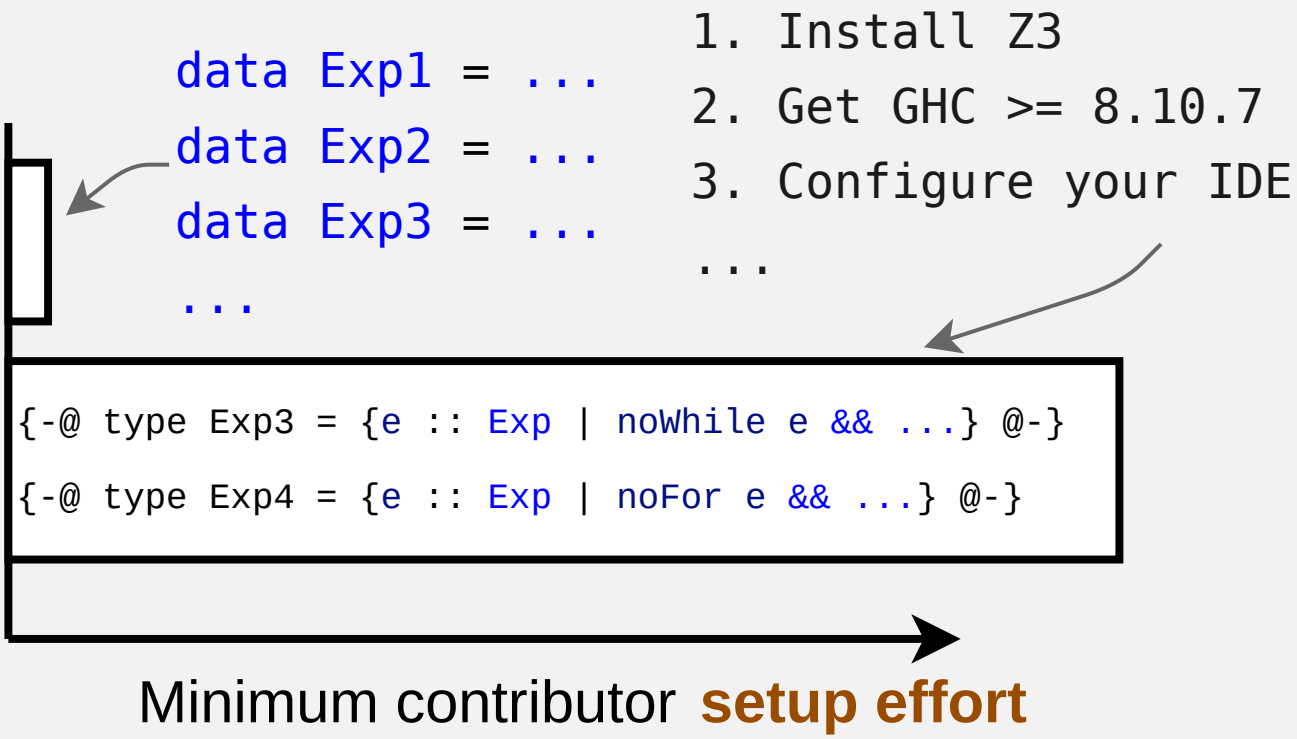
- 👰🔥 Research!
- 💍 Ornaments
- 🎁 Data type description EDSLs
- 📦 Container extension
- https://personal.cis.strath.ac.uk/conor.mcbride/pub/OAAO/LitOrn.pdf
- https://dl.acm.org/doi/pdf/10.1145/2502409.2502413
- https://doi.org/10.1017/S0956796814000069
- https://dl.acm.org/doi/pdf/10.1145/2633628.2633631
- https://doi.org/10.1017/S0956796816000356
- ...

Design 4: One AST, with refinements


```
{-@ type Exp3 = {e :: Exp | noWhile e && ...} @-}
{-@ type Exp4 = {e :: Exp | noFor e && ...} @-}

{-@ for2while :: Exp3 -> Exp4 @-}
for2while :: Exp -> Exp
```

Design 4: One AST, with refinements ⇒ 💩 Not Hs98



Design 5: One AST, with parameters

```
data Exp
  = Raw String
  | If Exp Exp Exp
  | Goto Label
  | Instr SSM.Instr
  | Typed Type Exp
  | For (Exp,Exp,Exp) Exp
  | While Exp Exp
  | ...
```

Design 5: One AST, with parameters

```
data Exp a b c d e f g h ... -- One param per ctr.
  = Raw a String
  | If b Exp Exp Exp
  | Goto c Label
  | Instr d SSM.Instr
  | Typed e Type Exp
  | For f (Exp,Exp,Exp) Exp
  | While g Exp Exp
  | ...
```

Design 5: One AST, with parameters

```
data Exp a b c d e f g h ... -- One param per ctr.
  = Raw a String
  | If b Exp Exp Exp
  | Goto c Label
  | Instr d SSM.Instr
  | Typed e Type Exp
  | For f (Exp,Exp,Exp) Exp
  | While g Exp Exp
  | ...
```

Design 5: One AST, with parameters

```
data Exp a b c d e f g h ... -- One param per ctr.
  = Raw a String
  | If b Exp Exp Exp
  | Goto c Label
  | Instr d SSM.Instr
  | Typed e Type Exp
  | For f (Exp,Exp,Exp) Exp
  | While g Exp Exp
  | ...
```

```
for2while :: Exp a b c d e for  while  ...
          -> Exp a b c d e Void ()     ...
```

Design 5: One AST, with parameters

```
data Exp a b c d e f g h ... -- One param per ctr.
  = Raw a String
  | If b Exp Exp Exp
  | Goto c Label
  | Instr d SSM.Instr
  | Typed e Type Exp
  | For f (Exp,Exp,Exp) Exp
  | While g Exp Exp
  | ...
```

```
for2while :: Exp a b c d e for  while  ...
          -> Exp a b c d e Void ()     ...
```

```
printSSMCode  :: Exp ⊥ ⊥ () () ⊥ ⊥ ... -> String
```

Design 5: One AST, with parameters

```
data Exp a b c d e f g h ... -- One param per ctr.
  = Raw a String
  | If b Exp Exp Exp
  | Goto c Label
  | Instr d SSM.Instr
  | Typed e Type Exp
  | For f (Exp,Exp,Exp) Exp
  | While g Exp Exp
  | ...
```

```
for2while :: Exp a b c d e for  while  ...
          -> Exp a b c d e Void ()     ...
```

```
printSSMCode  :: Exp ⊥ ⊥ () () ⊥ ⊥ ... -> String
```

✅ Pattern-checker friendly ✅ Easy re-ordering

Design 5: One AST, with parameters

```
data Exp a b c d e f g h ... -- One param per ctr.
  = Raw a String
  | If b Exp Exp Exp
  | Goto c Label
  | Instr d SSM.Instr
  | Typed e Type Exp
  | For f (Exp,Exp,Exp) Exp
  | While g Exp Exp
  | ...
```

```
for2while :: Exp a b c d e for  while  ...
          -> Exp a b c d e Void ()     ...
```

```
printSSMCode  :: Exp ⊥ ⊥ () () ⊥ ⊥ ... -> String
```

✅ Pattern-checker friendly ✅ Easy re-ordering

Big Types

Design 6: One AST, with parameter + type functions

```
data Exp ζ
  = Raw (XRaw ζ) String
  | If (XIf ζ) Exp Exp Exp
  | Goto (XGoto ζ) Label
  | Instr (XInstr ζ) SSM.Instr
  | Typed (XTyped ζ) Type Exp
  | For (XFor ζ) (Exp,Exp,Exp) E
  | While (XWhile ζ) Exp Exp
  | ...
```

```
-- One type per ctr.
type family XRaw ζ
type family XIf ζ
type family XGoto ζ
type family XInstr ζ
type family XTyped ζ
type family XFor ζ
type family XWhile ζ
```

- "Trees That Grow"
  - In GHC
  - https://gitlab.haskell.org/ghc/ghc/-/wikis/implementing-trees-that-grow

Summary

- "Divide & conquer" compiler passes
- Risk of O(№ctrs × №passes) repetition:

`Exp` Constructor	P1	P2	P3	P4	…	Pn
`Raw`	✅	❌	❌	❌	…	❌
`If`	❌	✅	✅	✅	…	❌
`Goto`	❌	❌	❌	❌	…	✅
`Instr`	❌	❌	❌	❌	…	✅
`Typed`	❌	❌	✅	✅	…	❌
`For`	❌	✅	✅	❌	…	❌
`While`	❌	✅	✅	✅	…	❌
⋮	⋮	⋮	⋮	⋮	⋱	⋮

- Motivates advanced data type design