



# Softwareprojekt Übersetzerbau SoSe 2014

## Milestone 1

Haskellgruppe

Rail-LLVM-Compiler

# Anforderungen an den 1. Meilenstein

1. Rollenverteilung
2. Gruppenorganisation
3. Pipeline-Schnittstellen
4. Hello-World-Compiler
5. AST-Interface
6. Syntax-Highlighting



# Rollenverteilung

## Chef:

- Christopher Pockrandt

## Interface-Master:

- Nicolas Lehmann
- Christopher Pockrandt

## Haskell-Master:

- Maximilian Claus

## Rail-Master:

- Christian Hofmann
- Tobias Kranz

## Target-Master:

- Sascha Zinke
- Tudor Soroceanu

## Git-Master:

- Sascha Zinke
- Tilman Blumenbach
- Marcus Hoffmann

## GUI-Design:

- Christoph Gräbnitz
- Michel Ajchman

## Dokumentation:

- Philipp Borgers
- Benjamin Koderer

## Test-Master:

- Kristin Knorr
- Nicolas Lehmann

# Gruppenorganisation

## Framework:

- Nicolas Lehmann

## Preprocessor:

- Nicolas Lehmann
- Christopher Prockrandt

## Lexer:

- Christian Hofmann
- Tilman Blumenbach

## Syntactical Analysis:

- Kristin Knorr
- Tobias Kranz
- Marcus Hoffmann

## Semantical Analysis:

- entfällt

## Intermediate Code:

- Philipp Borgers
- Lyudmila Vaseva
- Michel Ajchman

## Code Optimization:

- entfällt

## Backend:

- Tudor Soroceanu
- Tilman Blumenbach
- Maximilian Claus
- Sascha Zinke

## Editor / Syntax HL:

- Christoph Gräbnitz
- Kelvin Glaß
- Benjamin Koderä



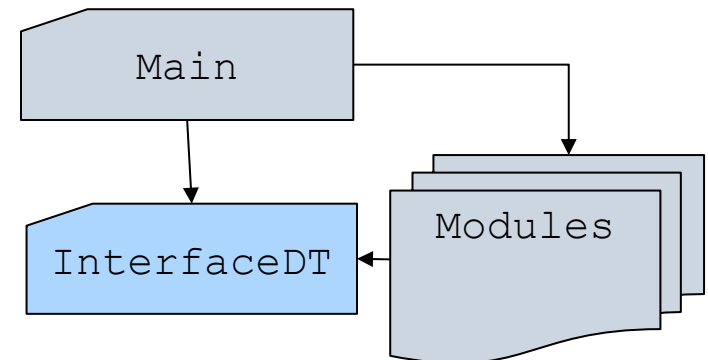
# Team: Framework

## ✓ 1. Entwurf: IoC Typ 3

- Aufgabe der Idee nach ca. 4 Tagen
- Idee verschoben in einen späteren Meilenstein

## ✓ 2. Entwurf:

- *InterfaceDT* regelt die Datentypübergabe zwischen den Modulen global



# Team: Preprozessor

## ✓ Idee:

- Für jede Rail-Funktion wird ein eigenes *2D-Grid* (2-dimensionale Matrize) erzeugt.
- Führende Zeilen ohne das \$-Symbol werden entfernt.

# Team: Lexer

## ✓ Idee:

### ▪ Zwei Datentypen

- *Instruction Pointer* (IP) mit Koordinaten und Ausrichtung
- *LexNode* der einen Knoten im Graphen darstellt als (Id, Lexeme, Nachfolger)

### ▪ Hauptabfolge

- *Instruction Pointer* bewegen
- Zeichen unter IP verarbeiten
  - bei Mehr-Zeichen-Lexemen geradeaus bis zur schließenden Klammer lesen

## ✓ Geplant:

- rekursiver Unteraufruf bei Junction
- Update des Nachfolgers des vorherigen Knotens

# Team: Syntaktische Analyse

## ✓ Idee:

- Ein Startknoten entspricht einem möglichen Startknoten für Pfade mit Ingrad  $>1$  oder einer Abzweigung die durch eine Weiche genommen wird
- Pfad mit Tiefensuche erstellen
- solange suchen, bis man wieder auf möglichen Startknoten treffen



# Team: Semantische Analyse

✓ **getestet UND bewiesen!**


$$(\lambda x.x) y = y$$



*„Awesome proof made by Iron Man!“*  
2014/05/14 Daily Bugle

# Team: Zwischencode

- ✓ **Stack mit fester Größe in LLVM implementiert**
  - ✓ **unterstützt zur Zeit nur Pointer auf Strings**
  - ✓ **Push() und Top()**
- ✓ **LLVM Assembly-Code Erzeugung mit**
  - ✓ **llvm-general**
  - ✓ **llvm-general-pure**

# Team: Codeoptimierung

- ✓ **Entfällt vorerst!!!** (geplant für 2. Meilenstein)



# Team: Backend

## ✓ Idee:

- verwendet LLVM-Bibliothek um aus dem LLVM-AST einen Assemblycode zu erstellen

## ✓ In Arbeit:

- Implementierung einer Runtime
  - read from input
  - eof
  - ...zur Benutzung im LLVM-Code

# Hello-World-Compiler

*siehe und staune...*



# AST-Interface

## ✓ Beispiel:

```
//Beginn Beispiel AST-Serialisierung für Hello World
```

```
[main]
```

```
1; [HelloWorld] ;2
```

```
2;o;3
```

```
3;#;
```

```
//Ende Beispiel AST-Serialisierung für Hello World
```

# Syntax Highlighting

*siehe und staune...*





# Softwareprojekt Übersetzerbau SoSe 2014

**Danke für Eure Aufmerksamkeit!**

**Fragen? Fragen!**

