How we got here

The genesis of enlightened programming languages
@steshaw

Where is “here”?

What we’re talking about

- ML - OCaml / SML
- Haskell
- Agda
- Idris

The genesis

Lambda Calculus

- lambda calculus (LC). Church.
- Perhaps mention STLC here too or a little further on.
- Will have to research the dates/timeline.
- The idea is that the genesis of enlightened PLs come from logicians, mathematicians and philosophers.

Automated Theorem Proving

- Theorem provers and Martin-Löf.
- Genesis of ML as a tactics language for the LCF theorem prover.
- Mention some of the people and places such as Robin Milner and University of Edinburgh.

Lisp

- Perhaps mention of lisp as a simple external syntax for the LC. This is were lisp remains in the past. Essentially it's light goes out.

CAML

- The first implementation of CAML in Lisp. Mention some early INRIA researchers such a Gérard Huet.

ML and SML

- The work on ML/SML. One of the researchers went from the US -> Scotland or the other way around. Find that guy and mention him :).
- Modules in SML and OCaml. Perhaps mention applicative v generative module systems and the similarity with Haskell's type classes. Mention modular type classes by Dreyer, Harper and Chakravarty.

Lazy MLs

- the rise of the lazy functional PLs such as Hope, Miranda and LazyML. Some folks, faces and places.

Haskell

- Haskell, one of the few times a design by committee worked (someone should study that and learn how they did it).

Type Classes

- Type classes. Wadler.
- A generalisation of eqtype variables from Standard ML.

Proof Assistants

- Verified software and dependent types.
- Languages such as Agda, Coq, Idris.
- These influence the Glorious dialect of Haskell available in GHC.
- Here, we're full-circle back to theorem provers :D

Further Resources

Key references

- How to make ad-hoc polymorphism less ad hoc. Philip Wadler and Stephen Blott. 16'th Symposium on Principles of Programming Languages, ACM Press, Austin, Texas, January 1989.
- This paper presents type classes, a new approach to ad-hoc polymorphism. Type classes permit overloading of arithmetic operators such as multiplication, and generalise the "eqtype variables" of Standard ML. Type classes extend the Hindley-Milner polymorphic type system, and provide a new approach to issues that arise in object-oriented programming, bounded type quantification, and abstract data types. This paper provides an informal introduction to type classes, and defines them formally by means of type inference rules.

Further reading

- See `http://steshaw.org/how-we-got-here/resources.html`
- A curated resource for the interested student.
- Includes key papers & books.
- Links to programming languages. Haskell, ML, proof assistants,
- less well-known research PLs such as ATS, Ur/Web, F*.

Questions?