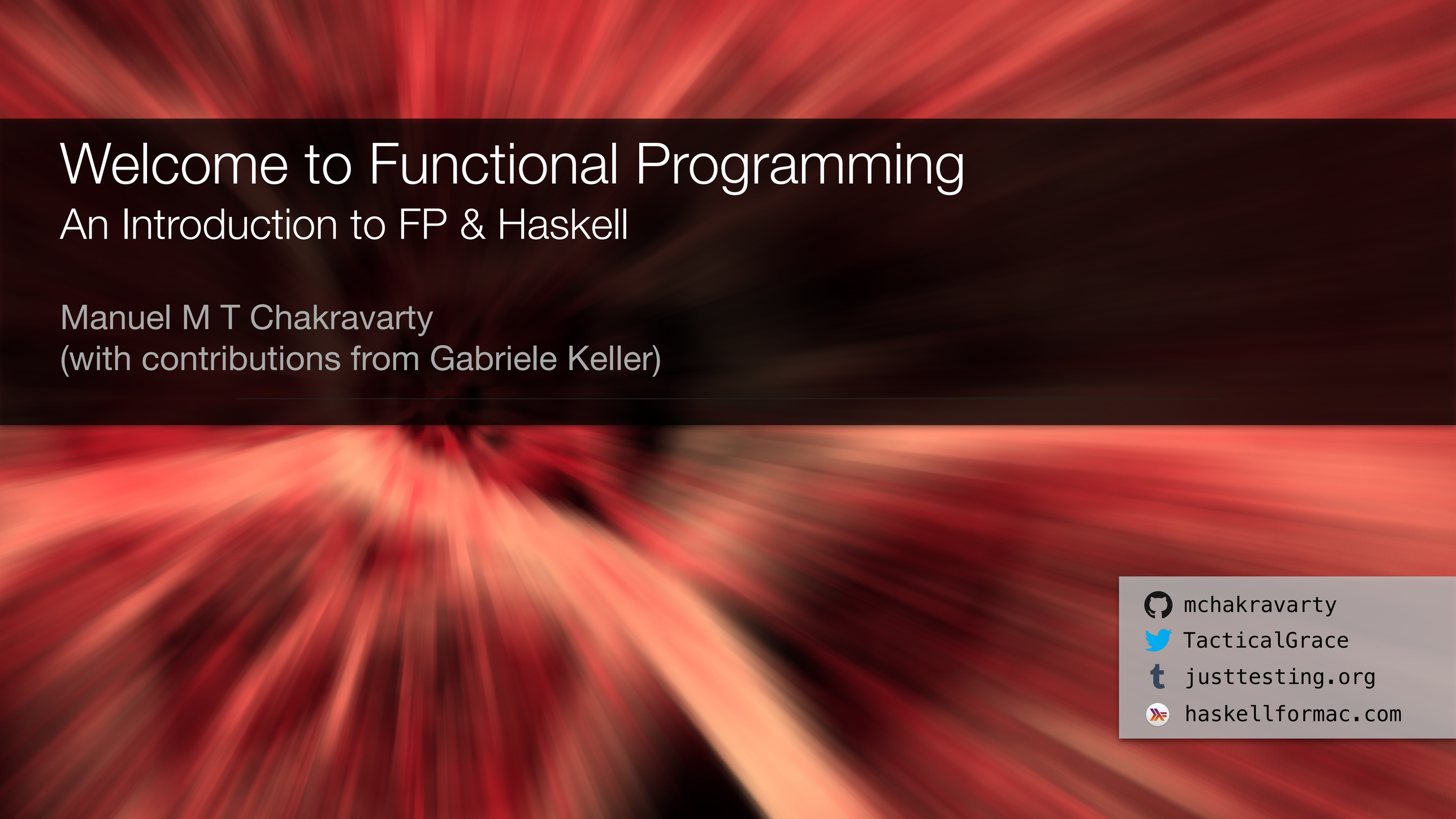
Welcome to Functional Programming

An Introduction to FP & Haskell

Manuel M T Chakravarty
(with contributions from Gabriele Keller)



Who am I?

Who am I?

Formerly

UNSW SYDNEY

Associated Professor

Who am I?

Formerly

UNSW SYDNEY

Associated Professor

Haskell for Mac

Indie Developer

Who am I?

Formerly

UNSW SYDNEY

Associated Professor

Haskell for Mac

Indie Developer

tweag.io

Functional Programming
Evangelist

Who am I?

Formerly

UNSW SYDNEY

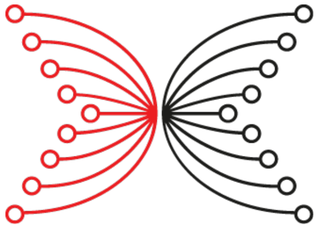
Associated Professor

Haskell for Mac

Indie Developer

tweag.io

Functional Programming Evangelist



INPUT | OUTPUT

Blockchain Language Architect

What are we going to do?

What are we going to do?

Overview over functional programming

Taxonomy of FP languages & features

What are we going to do?

Overview over functional programming

Taxonomy of FP languages & features

Fundamentals of Haskell & functional programming

Core concepts of writing code in an FP language

What are we going to do?

Overview over functional programming

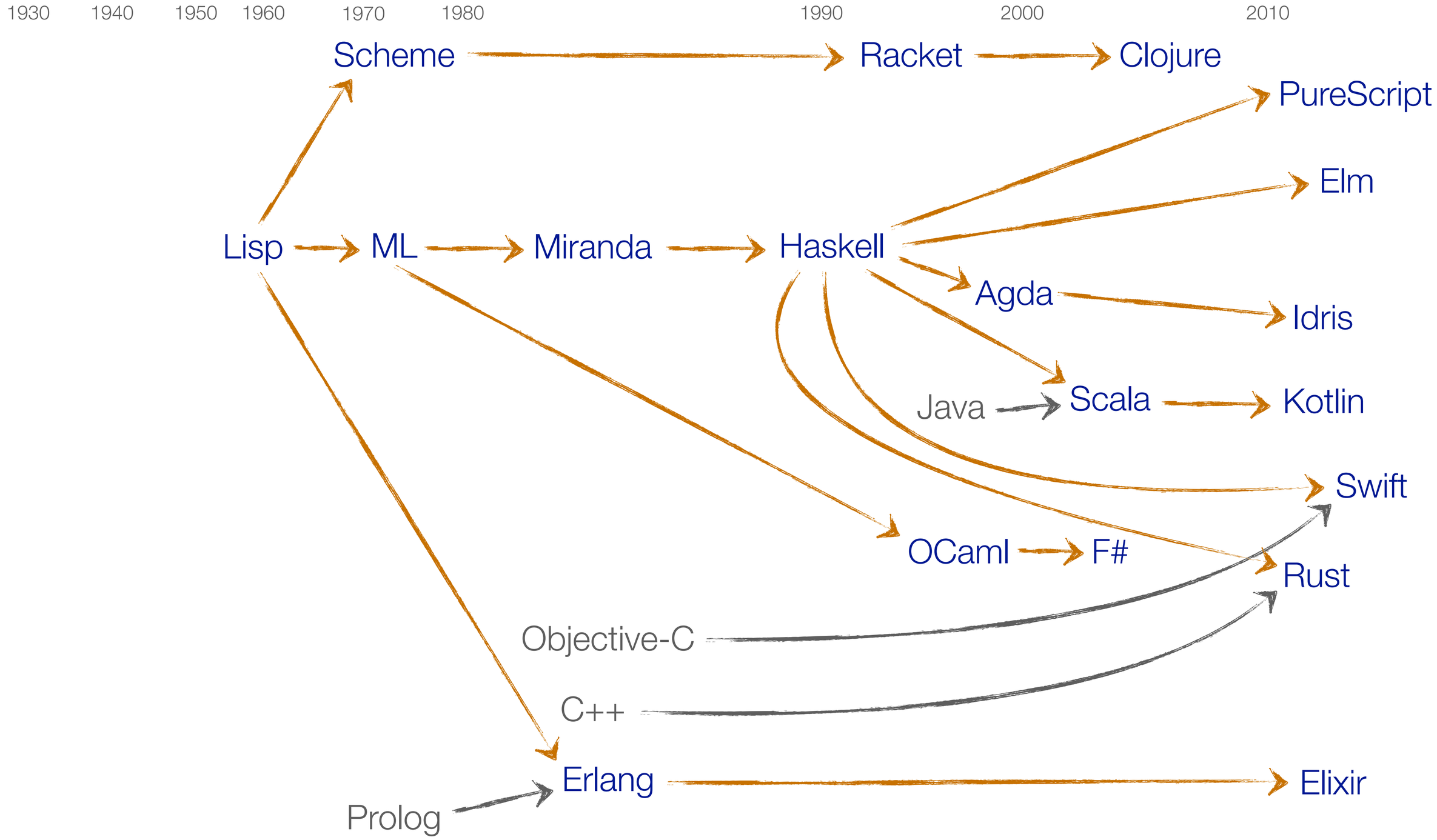
Taxonomy of FP languages & features

Fundamentals of Haskell & functional programming

Core concepts of writing code in an FP language

Actually use Haskell

Get a taste for writing programs in Haskell


1930
1940
1950
1960
1970
1980
1990
2000
2010
Scheme
Racket
Clojure
PureScript
Elm
Lisp
ML
Miranda
Haskell
Agda
Idris
Java
Scala
Kotlin
Swift
OCaml
F#
Rust
Objective-C
C++
Erlang
Elixir
Prolog

Lisp

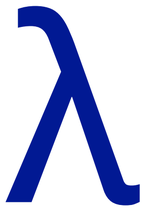
Lambda calculus

Lambda calculus

Church (1930s)

λ

Lambda calculus



Church (1930s)

exp →	var	— variables
|	(λvar. exp)	— abstraction
|	(exp_1 exp_2)	— application

Lambda calculus

λ

Church (1930s)

exp → var — variables

| (λvar. exp) — abstraction

| (exp_1 exp_2) — application

(λvar. exp_1) exp_2

Lambda calculus

Church (1930s)

$$
\begin{array}{lll}
\text{exp} \rightarrow \text{var} & & \text{— variables} \\
\quad \mid \ (\lambda \text{var}.\ \text{exp}) & & \text{— abstraction} \\
\quad \mid \ (\text{exp}_1\ \ \text{exp}_2) & & \text{— application}
\end{array}
$$

replace every var in exp_1 by exp_2

$$
(\lambda \text{var}.\ \text{exp}_1)\ \ \text{exp}_2 \xrightarrow{\beta} [\text{exp}_2/\text{var}]\ \text{exp}_1
$$

— execution step

Lambda calculus

Church (1930s)

$$
\begin{aligned}
\text{exp} &\rightarrow \text{var} && \text{— variables}\\
&\mid (\lambda \text{var}.\ \text{exp}) && \text{— abstraction}\\
&\mid (\text{exp}_1\ \ \text{exp}_2) && \text{— application}
\end{aligned}
$$

replace every var in exp_1 by exp_2

$$(\lambda \text{var}.\ \text{exp}_1)\ \ \text{exp}_2 \xrightarrow{\beta} [\text{exp}_2/\text{var}]\ \text{exp}_1$$

— execution step

Turing-complete

Lambda calculus

λ

Church (1930s)

exp → var — variables

| (λvar. exp) — abstraction

| (exp_1 exp_2) — application

replace every var in exp_1 by exp_2

$(\lambda var.\ exp_1)\ exp_2 \xrightarrow{\beta} [exp_2/var]\ exp_1$ — execution step

Turing-complete

Foundations of mathematics

Lambda calculus

λ

Church (1930s)

exp → var — variables

| (λvar. exp) — abstraction

| (exp_1 exp_2) — application

replace every var in exp_1 by exp_2

$(\lambda var.\ exp_1)\ exp_2 \xrightarrow{\beta} [exp_2/var]\ exp_1$ — execution step

Typed lambda calculus in 1940!

Turing-complete

Foundations of mathematics

Lisp

Lisp

McCarthy (1958)

Lisp

McCarthy (1958)

```
(lambda (arg) (+ arg 1))
```

Lisp

McCarthy (1958)

```
(lambda (arg) (+ arg 1))
```

```
(defun factorial (n)
   (if (= n 0) 1
       (* n (factorial (- n 1)))))
```

Lisp

Lisp

McCarthy (1958)

Lisp

McCarthy (1958)

Lambdas

Lisp

McCarthy (1958)

Lambdas

Recursion

Lisp

McCarthy (1958)

Lambdas

Tree structures

Recursion

Lisp

McCarthy (1958)

Lambdas

Tree structures

Recursion

Garbage collection

Lisp

McCarthy (1958)

Lambdas

Meta programming (macros)

Tree structures

Recursion

Garbage collection

Lisp

McCarthy (1958)

Lambdas

Meta programming (macros)

Tree structures

Recursion

Read-eval-print loop (REPL)

Garbage collection

Lisp

McCarthy (1958)

Lambdas

Meta programming (macros)

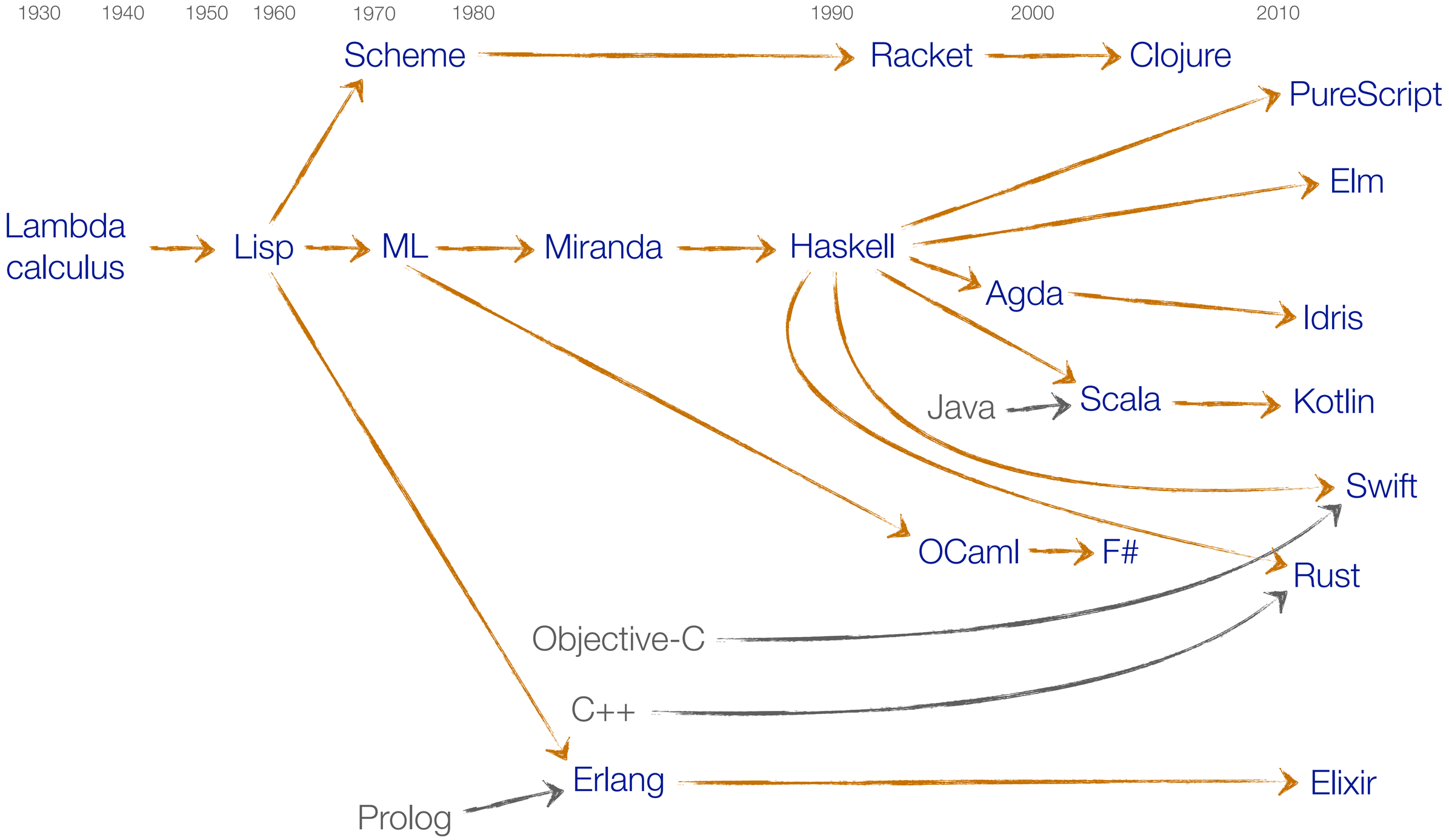
Tree structures

Recursion

Read-eval-print loop (REPL)

Garbage collection

Metacircular interpreter


1930
1940
1950
1960
1970
1980
1990
2000
2010
Scheme
Racket
Clojure
PureScript
Elm
Lambda calculus
Lisp
ML
Miranda
Haskell
Agda
Idris
Java
Scala
Kotlin
Swift
OCaml
F#
Rust
Objective-C
C++
Erlang
Elixir
Prolog

Scheme Racket Clojure

Scheme

λ

Steele/Sussman
(1975)

Racket

PLT Inc
(1994)

Clojure

Hickey
(2007)

Scheme

λ

Steele/Sussman
(1975)

Racket

PLT Inc
(1994)

Clojure

Hickey
(2007)

Lexical scoping

Scheme

λ

Steele/Sussman
(1975)

Racket

PLT Inc
(1994)

Clojure

Hickey
(2007)

Lexical scoping

Tail call optimisation

Scheme

λ

Steele/Sussman
(1975)

Racket

PLT Inc
(1994)

Clojure

Hickey
(2007)

Lexical scoping

Tail call optimisation

```
(defun factorial (n)
   (if (= n 0) 1
       (* n (factorial (- n 1)))))
```

Scheme

λ

Steele/Sussman
(1975)

Racket

PLT Inc
(1994)

Clojure

Hickey
(2007)

Lexical scoping

Tail call optimisation

```
(defun factorial (n)
   (if (= n 0) 1
       (* n (factorial (- n 1)))))
```

```
(defun factorial (n &optional (acc 1))
```

Scheme

λ

Steele/Sussman
(1975)

Racket

PLT Inc
(1994)

Clojure

Hickey
(2007)

Lexical scoping

Tail call optimisation

```
(defun factorial (n)
   (if (= n 0) 1
       (* n (factorial (- n 1)))))
```

```
(defun factorial (n &optional (acc 1))
  (if (= n 0) acc
```

Scheme

λ

Steele/Sussman
(1975)

Racket

PLT Inc
(1994)

Clojure

Hickey
(2007)

Lexical scoping

Tail call optimisation

```
(defun factorial (n)
   (if (= n 0) 1
       (* n (factorial (- n 1)))))
```

```
(defun factorial (n &optional (acc 1))
  (if (= n 0) acc
      (factorial (- n 1) (* acc n))))
```

Scheme

λ

Steele/Sussman
(1975)

Racket

PLT Inc
(1994)

Clojure

Hickey
(2007)

Lexical scoping

Tail call optimisation

First-class continuations

Scheme

λ

Steele/Sussman
(1975)

Racket

PLT Inc
(1994)

Clojure

Hickey
(2007)

Lexical scoping

Education

Tail call optimisation

First-class continuations

Scheme

λ

Steele/Sussman
(1975)

Lexical scoping

Tail call optimisation

First-class continuations

Racket

PLT Inc
(1994)

Education

Domain-specific
languages

Clojure

Hickey
(2007)

Scheme

λ

Steele/Sussman
(1975)

Lexical scoping

Tail call optimisation

First-class continuations

Racket

PLT Inc
(1994)

Education

Domain-specific
languages

Clojure

Hickey
(2007)

JVM

Scheme

λ

Steele/Sussman
(1975)

Lexical scoping

Tail call optimisation

First-class continuations

Racket

PLT Inc
(1994)

Education

Domain-specific
languages

Clojure

Hickey
(2007)

JVM

ClojureScript

Scheme

λ

Steele/Sussman
(1975)

Lexical scoping

Tail call optimisation

First-class continuations

Racket

PLT Inc
(1994)

Education

Domain-specific
languages

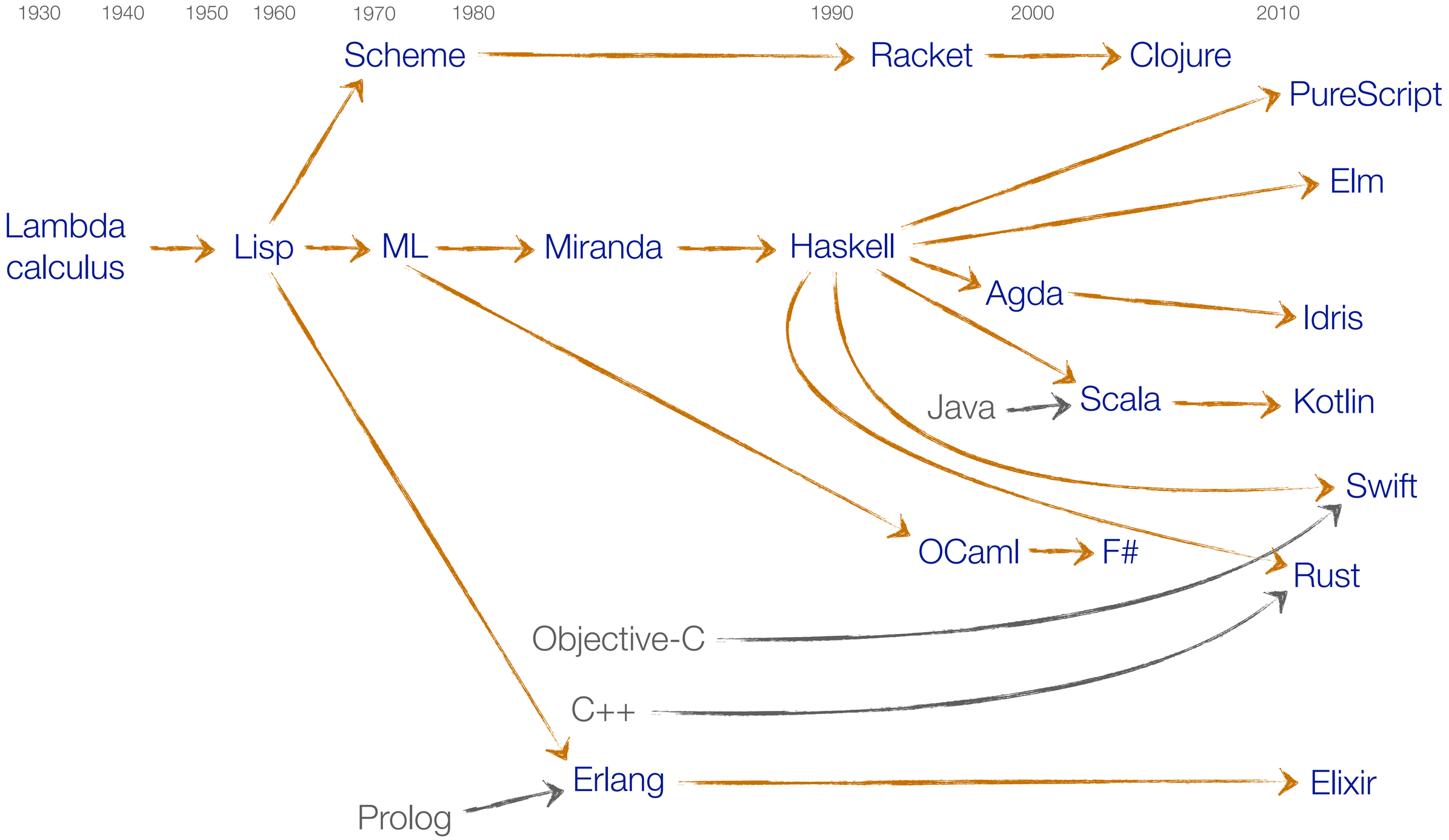
Clojure

Hickey
(2007)

JVM

ClojureScript

Immutability


1930
1940
1950
1960
1970
1980
1990
2000
2010
Scheme
Racket
Clojure
PureScript
Elm
Lambda calculus
Lisp
ML
Miranda
Haskell
Agda
Idris
Java
Scala
Kotlin
Swift
OCaml
F#
Rust
Objective-C
C++
Erlang
Elixir
Prolog

Erlang

Elixir

Erlang

Armstrong et al.
(1986)

Elixir

Valim
(2011)

Erlang

Armstrong et al.
(1986)

Elixir

Valim
(2011)

Concurrent

Erlang

Armstrong et al.
(1986)

Elixir

Valim
(2011)

Concurrent

Distributed

Erlang

Armstrong et al.
(1986)

Elixir

Valim
(2011)

Concurrent

Distributed

Fault-tolerant

Erlang

Armstrong et al.
(1986)

Concurrent

Distributed

Fault-tolerant

Hot swapping

Elixir

Valim
(2011)

Erlang

Armstrong et al.
(1986)

Concurrent

Distributed

Fault-tolerant

Hot swapping

Elixir

Valim
(2011)

Metaprogramming

Erlang

Armstrong et al.
(1986)

Concurrent

Distributed

Fault-tolerant

Hot swapping

Elixir

Valim
(2011)

Metaprogramming

Protocols

Erlang

Armstrong et al.
(1986)

Concurrent

Distributed

Fault-tolerant

Hot swapping

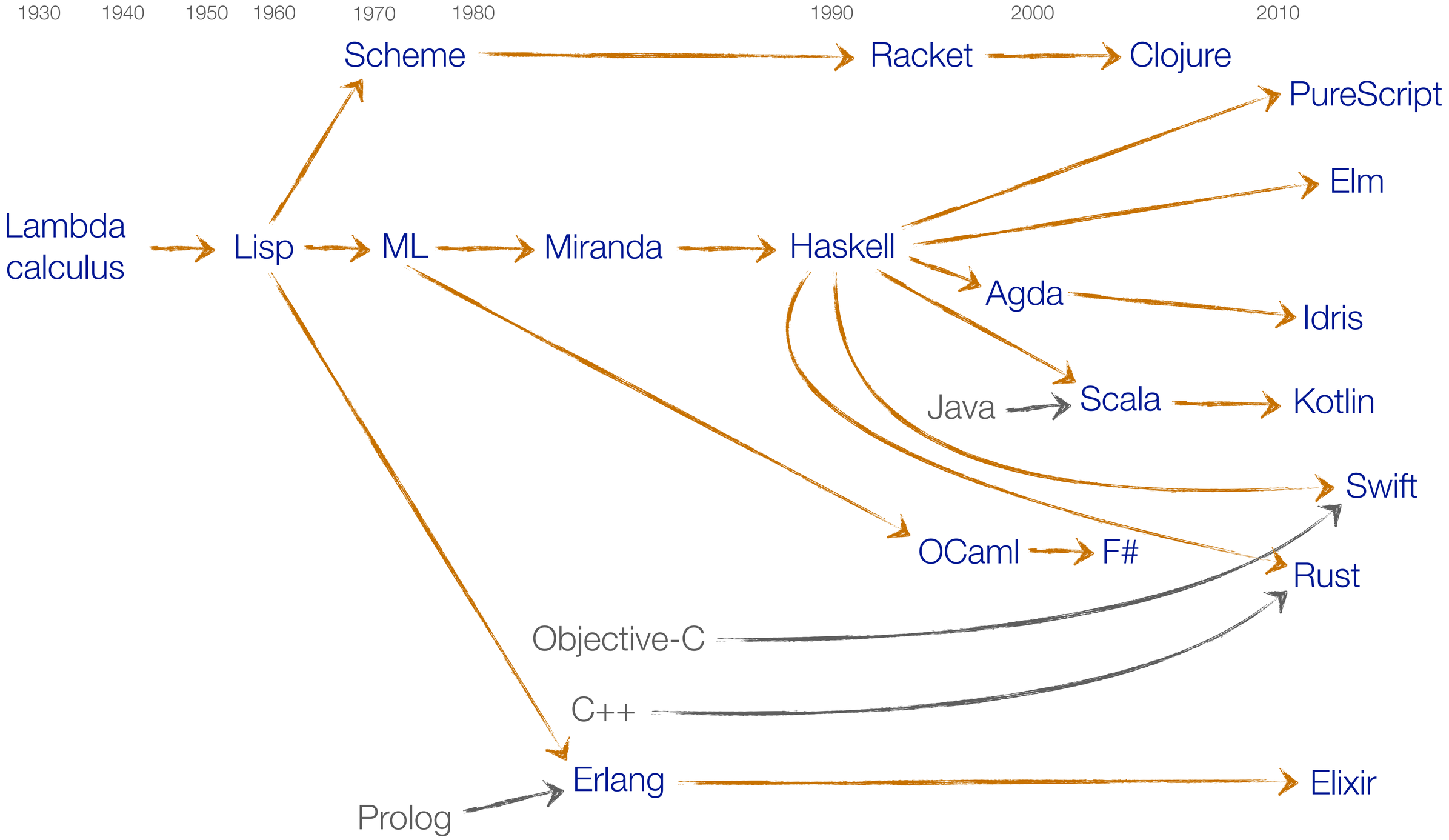
Elixir

Valim
(2011)

Metaprogramming

Protocols

Tooling


1930
1940
1950
1960
1970
1980
1990
2000
2010
Scheme
Racket
Clojure
PureScript
Elm
Lambda calculus
Lisp
ML
Miranda
Haskell
Agda
Idris
Java
Scala
Kotlin
Swift
OCaml
F#
Rust
Objective-C
C++
Erlang
Elixir
Prolog

(Standard) ML

(Standard) ML

Miller/Tofte/Harper (1973)

(Standard) ML

Miller/Tofte/Harper (1973)

```
fun fact (0 : int) : int = 1
  | fact (n : int) : int = n * fact (n - 1)
```

(Standard) ML

Miller/Tofte/Harper (1973)

```
fun fact (0 : int) : int = 1
  | fact (n : int) : int = n * fact (n - 1)
```

Algebraic data types

(Standard) ML

Miller/Tofte/Harper (1973)

```
fun fact (0 : int) : int = 1
  | fact (n : int) : int = n * fact (n - 1)
```

Algebraic data types

Pattern matching

(Standard) ML

Miller/Tofte/Harper (1973)

```
fun fact (0 : int) : int = 1
  | fact (n : int) : int = n * fact (n - 1)
```

Algebraic data types

Pattern matching

Parametric polymorphism
(aka generics)

(Standard) ML

Miller/Tofte/Harper (1973)

```
fun fact 0 = 1
  | fact n = n * fact (n - 1)
```

Algebraic data types

Type inference

Pattern matching

Parametric polymorphism
(aka generics)

(Standard) ML

Miller/Tofte/Harper (1973)

```
fun fact 0 = 1
  | fact n = n * fact (n - 1)
```

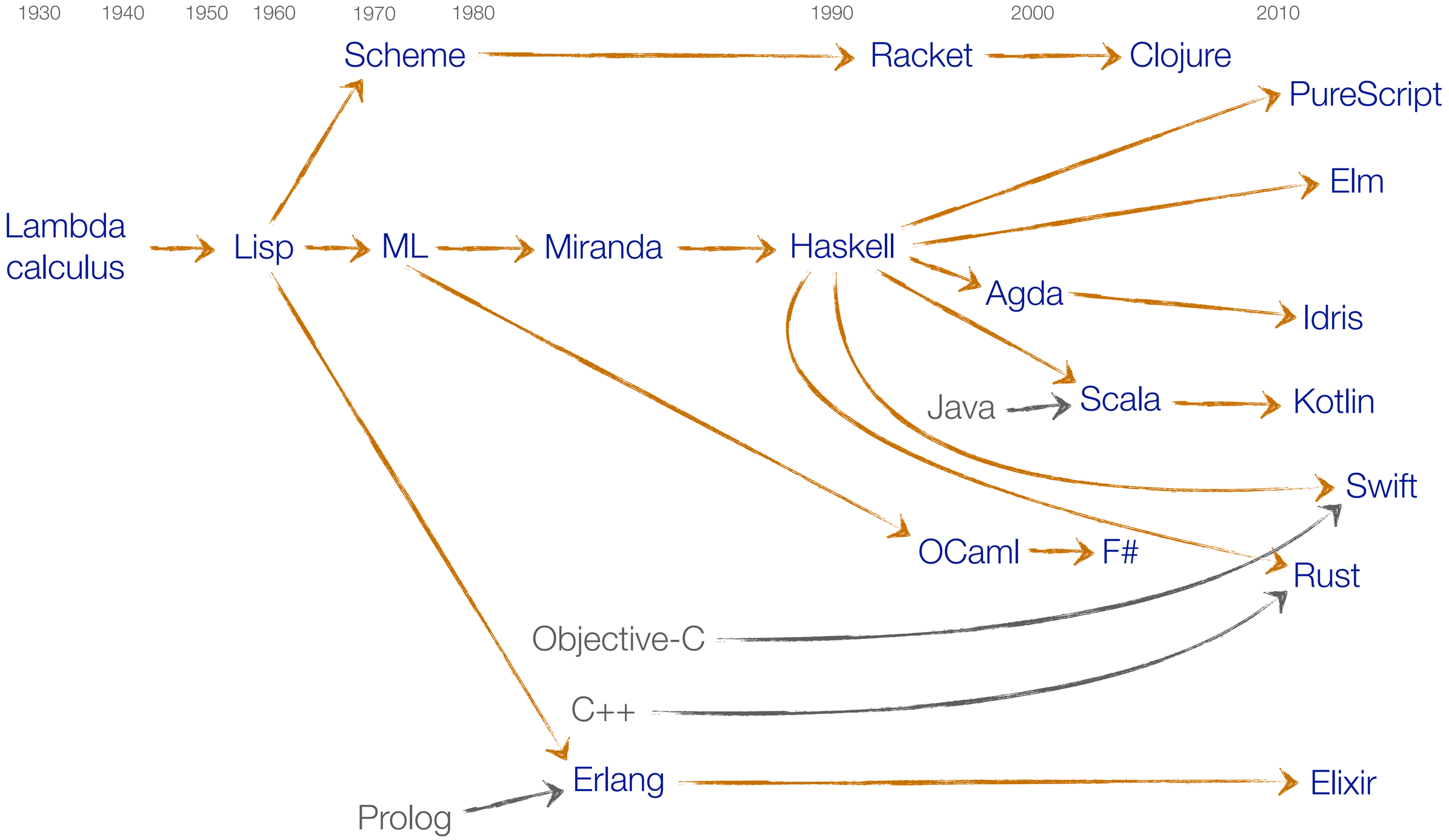
Algebraic data types

Type inference

Pattern matching

Parametric polymorphism
(aka generics)

Higher-order
module system


1930
1940
1950
1960
1970
1980
1990
2000
2010
Scheme
Racket
Clojure
PureScript
Elm
Lambda calculus
Lisp
ML
Miranda
Haskell
Agda
Idris
Java
Scala
Kotlin
Swift
OCaml
F#
Rust
Objective-C
C++
Erlang
Elixir
Prolog

OCaml F#

OCaml

Leroy et al.
(1996)

F#

Syme
(2005)

OCaml

Leroy et al.
(1996)

F#

Syme
(2005)

Object system

OCaml

Leroy et al.
(1996)

F#

Syme
(2005)

Object system

Structural subtyping

OCaml

Leroy et al.
(1996)

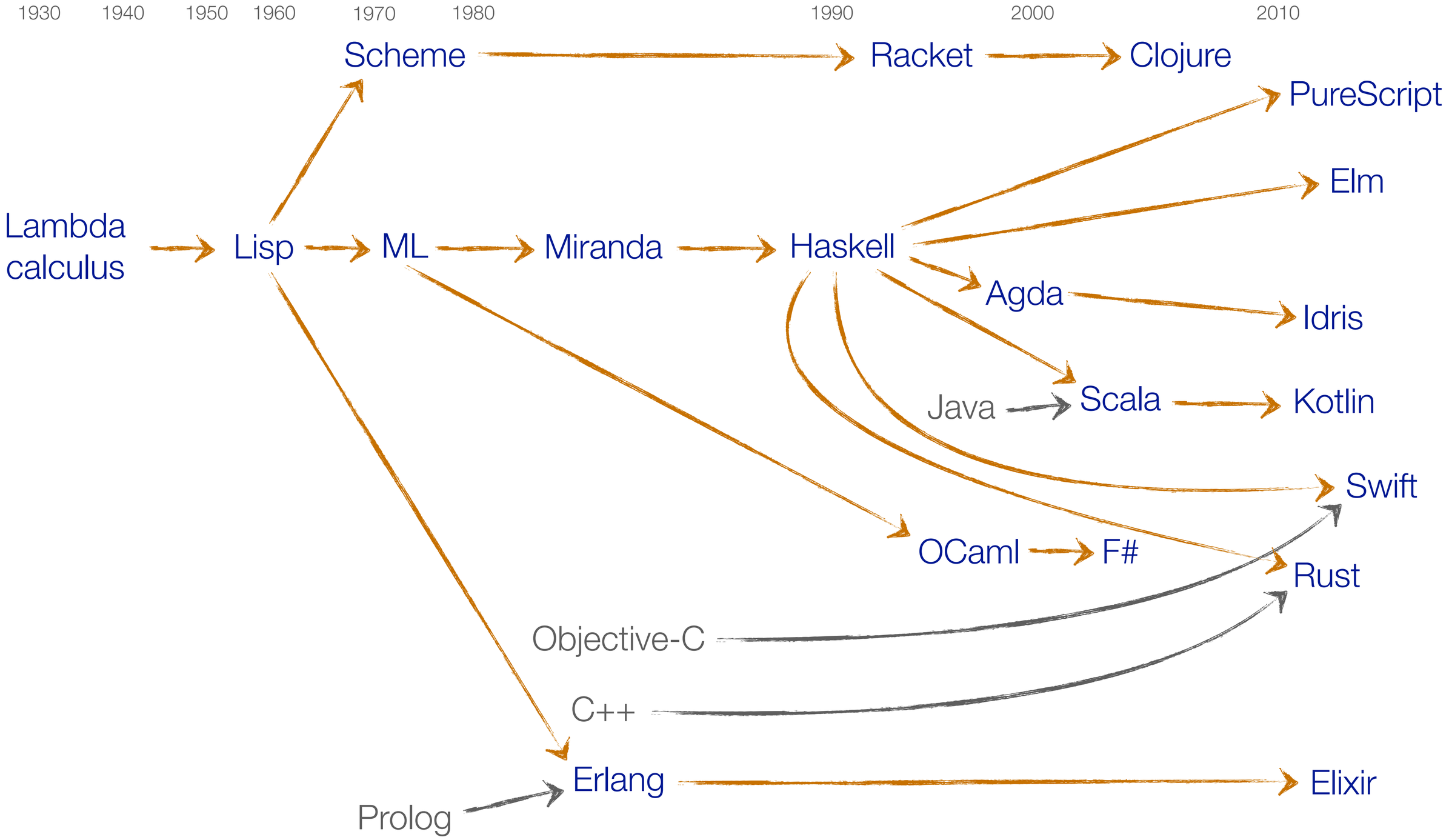
F#

Syme
(2005)

Object system

.NET CLR

Structural subtyping


1930
1940
1950
1960
1970
1980
1990
2000
2010
Scheme
Racket
Clojure
PureScript
Elm
Lambda calculus
Lisp
ML
Miranda
Haskell
Agda
Idris
Java
Scala
Kotlin
Swift
OCaml
F#
Rust
Objective-C
C++
Erlang
Elixir
Prolog

Miranda

Miranda

Turner
(1985)

Miranda

Turner
(1985)

```
fact :: num -> num    || optional
fact 0      = 1
fact (n+1) = (n+1) * fact n
```

Miranda

Turner
(1985)

```
fact :: num -> num    || optional
fact 0      = 1
fact (n+1) = (n+1) * fact n
```

Laziness

Miranda

Turner
(1985)

```
fact :: num -> num    || optional
fact 0      = 1
fact (n+1) = (n+1) * fact n
```

Laziness

List comprehensions

Miranda

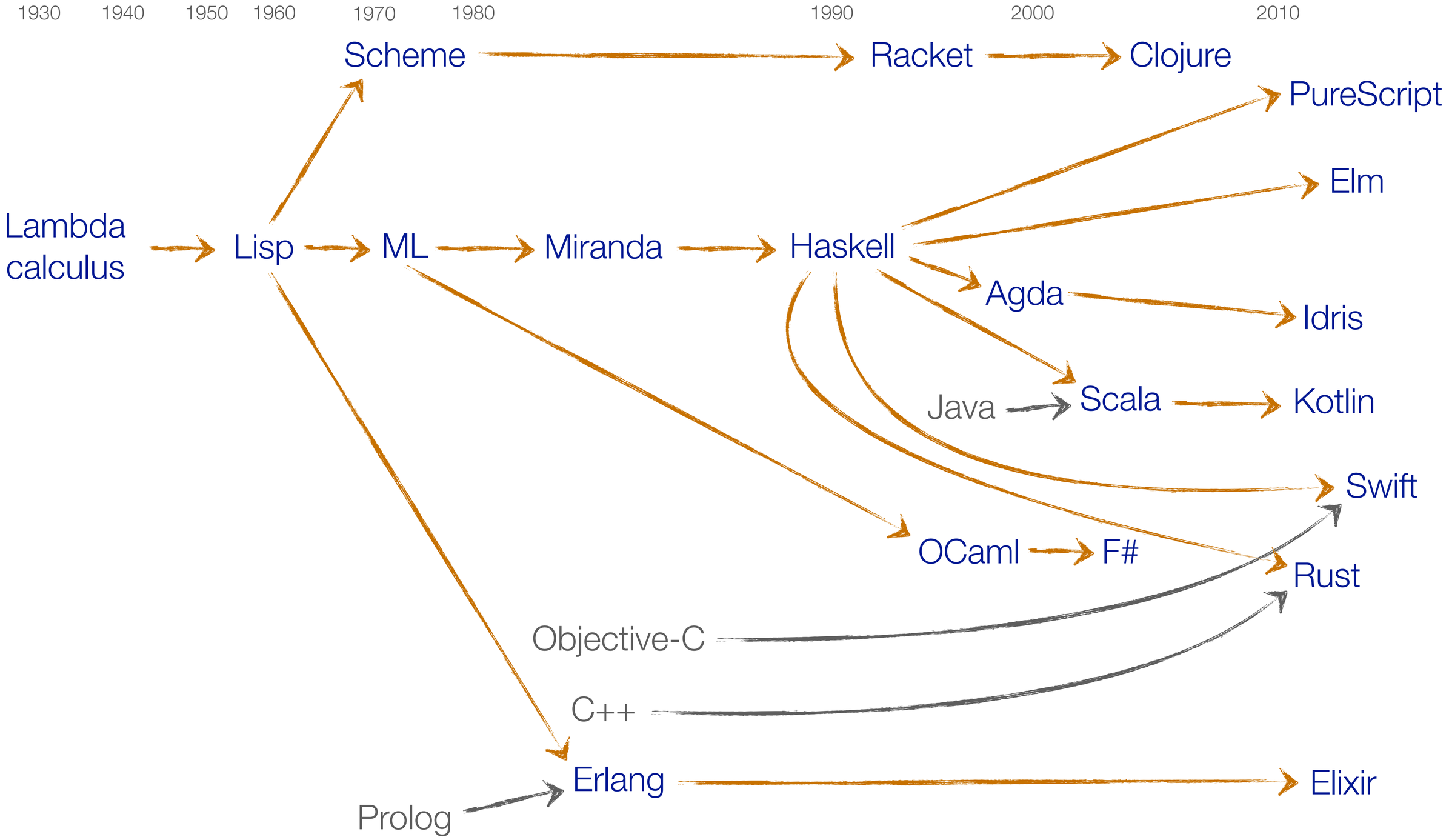
Turner
(1985)

```
fact :: num -> num    || optional
fact 0      = 1
fact (n+1) = (n+1) * fact n
```

Laziness

List comprehensions

```
primes       = sieve [2..]
sieve (p:x) = p : sieve [n | n <- x; n mod p ~= 0]
```


1930
1940
1950
1960
1970
1980
1990
2000
2010
Scheme
Racket
Clojure
PureScript
Elm
Lambda calculus
Lisp
ML
Miranda
Haskell
Agda
Idris
Java
Scala
Kotlin
Swift
OCaml
F#
Rust
Objective-C
C++
Erlang
Elixir
Prolog

Haskell

Haskell

Augustsson et al.

(1990)

Haskell

Augustsson et al.
(1990)

```haskell
fact :: Num a => a -> a      -- optional
fact 0 = 1
fact n = n * fact (n - 1)
```

Haskell

Augustsson et al.
(1990)

```
fact :: Num a => a -> a      -- optional
fact 0 = 1
fact n = n * fact (n - 1)
```

Type classes

Haskell

Augustsson et al.
(1990)

```haskell
fact :: Num a => a -> a      -- optional
fact 0 = 1
fact n = n * fact (n - 1)
```

Type classes

Monads
(and other categorial structures)

Haskell

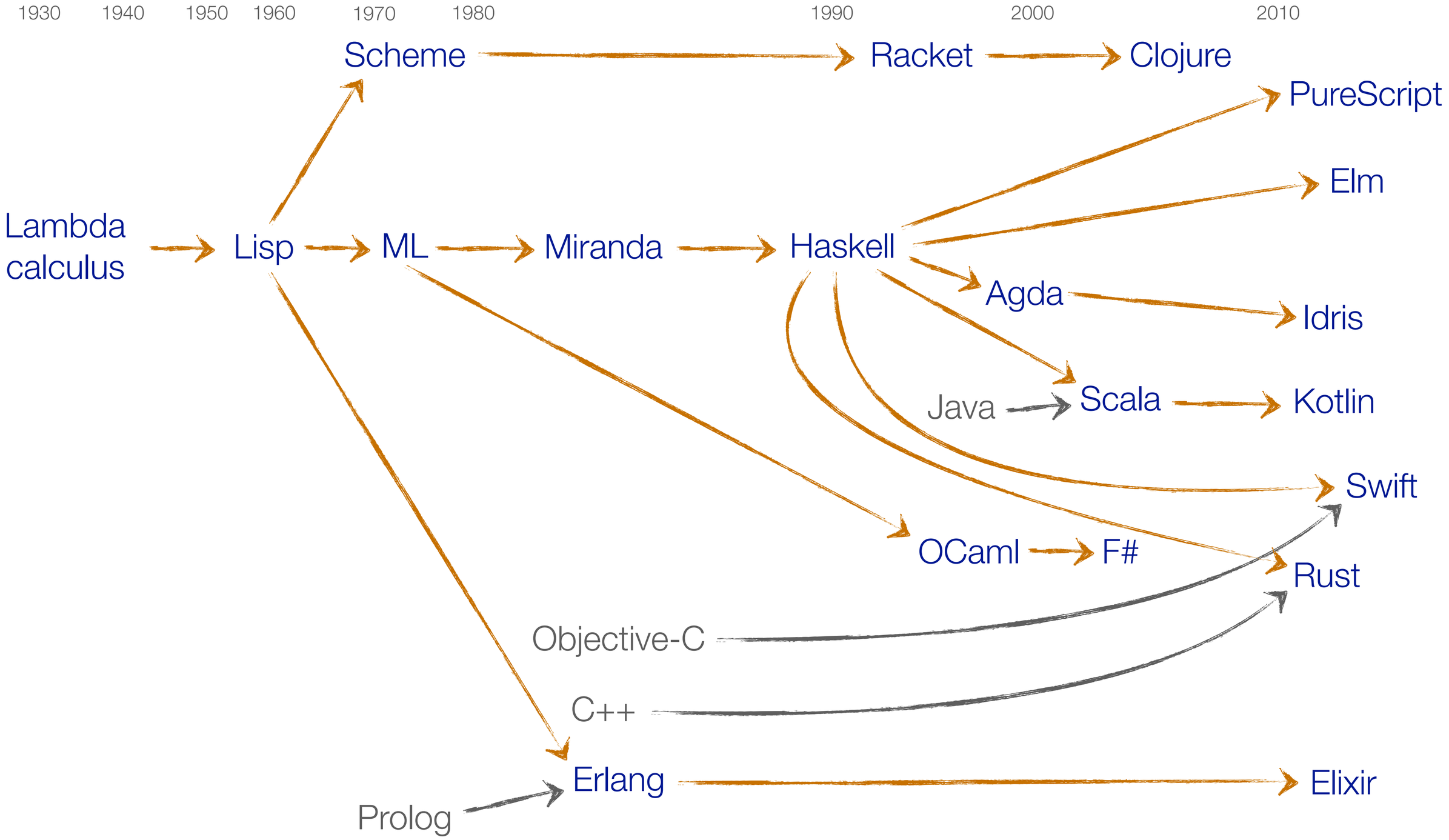
Augustsson et al.
(1990)

```haskell
fact :: Num a => a -> a      -- optional
fact 0 = 1
fact n = n * fact (n - 1)
```

Type classes

Monads
(and other categorial structures)

Type families


1930
1940
1950
1960
1970
1980
1990
2000
2010
Scheme
Racket
Clojure
PureScript
Elm
Lambda calculus
Lisp
ML
Miranda
Haskell
Agda
Idris
Java
Scala
Kotlin
Swift
OCaml
F#
Rust
Objective-C
C++
Erlang
Elixir
Prolog

Agda

Idris

Agda

Norell/Coquand
(1999)

Idris

Brady
(2011)

Agda

Norell/Coquand
(1999)

Idris

Brady
(2011)

Dependent types

Agda

Norell/Coquand
(1999)

Idris

Brady
(2011)

Dependent types

Termination checking

Agda

Norell/Coquand
(1999)

Idris

Brady
(2011)

Dependent types

Termination checking

```
data _≤_ : ℕ → ℕ → Set where
  z≤n : {n : ℕ} → zero ≤ n
  s≤s : {n m : ℕ} → n ≤ m → suc n ≤ suc m
```

Agda

Norell/Coquand
(1999)

Dependent types

Termination checking

Idris

Brady
(2011)

Effect system

```
data _≤_ : ℕ → ℕ → Set where
  z≤n : {n : ℕ} → zero ≤ n
  s≤s : {n m : ℕ} → n ≤ m → suc n ≤ suc m
```

Agda

Norell/Coquand
(1999)

Dependent types

Termination checking

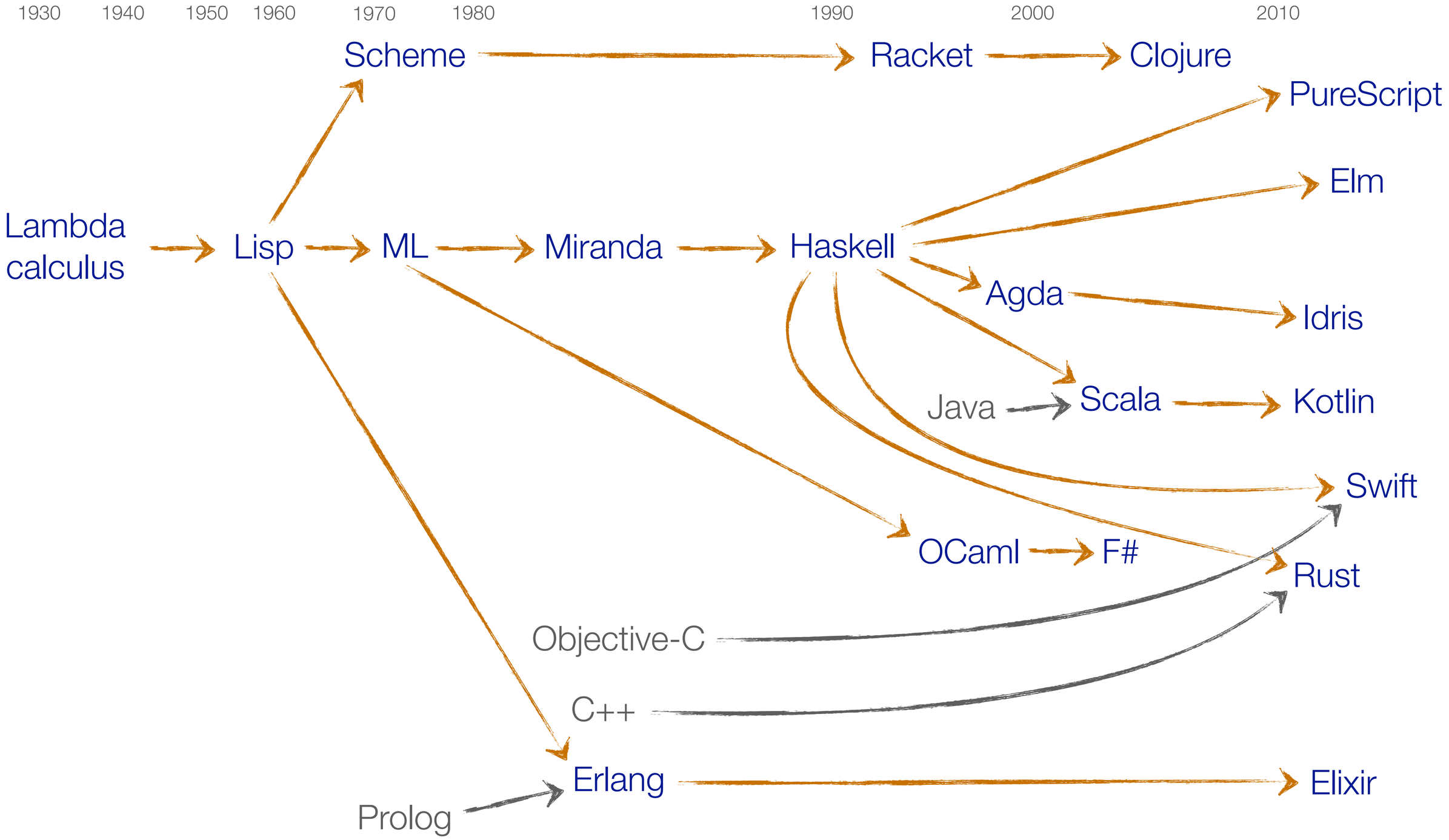
Idris

Brady
(2011)

Effect system

EDSL support

```
data _≤_ : ℕ → ℕ → Set where
  z≤n : {n : ℕ} → zero ≤ n
  s≤s : {n m : ℕ} → n ≤ m → suc n ≤ suc m
```


1930
1940
1950
1960
1970
1980
1990
2000
2010
Scheme
Racket
Clojure
PureScript
Elm
Lambda calculus
Lisp
ML
Miranda
Haskell
Agda
Idris
Java
Scala
Kotlin
Swift
OCaml
F#
Rust
Objective-C
C++
Erlang
Elixir
Prolog

Elm

PureScript

Elm

Czaplicki
(2012)

PureScript

Freeman
(2013)

Elm

Czaplicki
(2012)

Minimalistic
& approachable

PureScript

Freeman
(2013)

Elm

Czaplicki
(2012)

Minimalistic
& approachable

Elm Architecture

PureScript

Freeman
(2013)

Elm

Czaplicki
(2012)

Minimalistic
& approachable

Elm Architecture

Web frontends

PureScript

Freeman
(2013)

Elm

Czaplicki
(2012)

Minimalistic
& approachable

Elm Architecture

Web frontends

PureScript

Freeman
(2013)

Stays closer
to Haskell

Elm

Czaplicki
(2012)

Minimalistic
& approachable

Elm Architecture

Web frontends

PureScript

Freeman
(2013)

Stays closer
to Haskell

Extensible effects

Elm

Czaplicki
(2012)

Minimalistic
& approachable

Elm Architecture

Web frontends

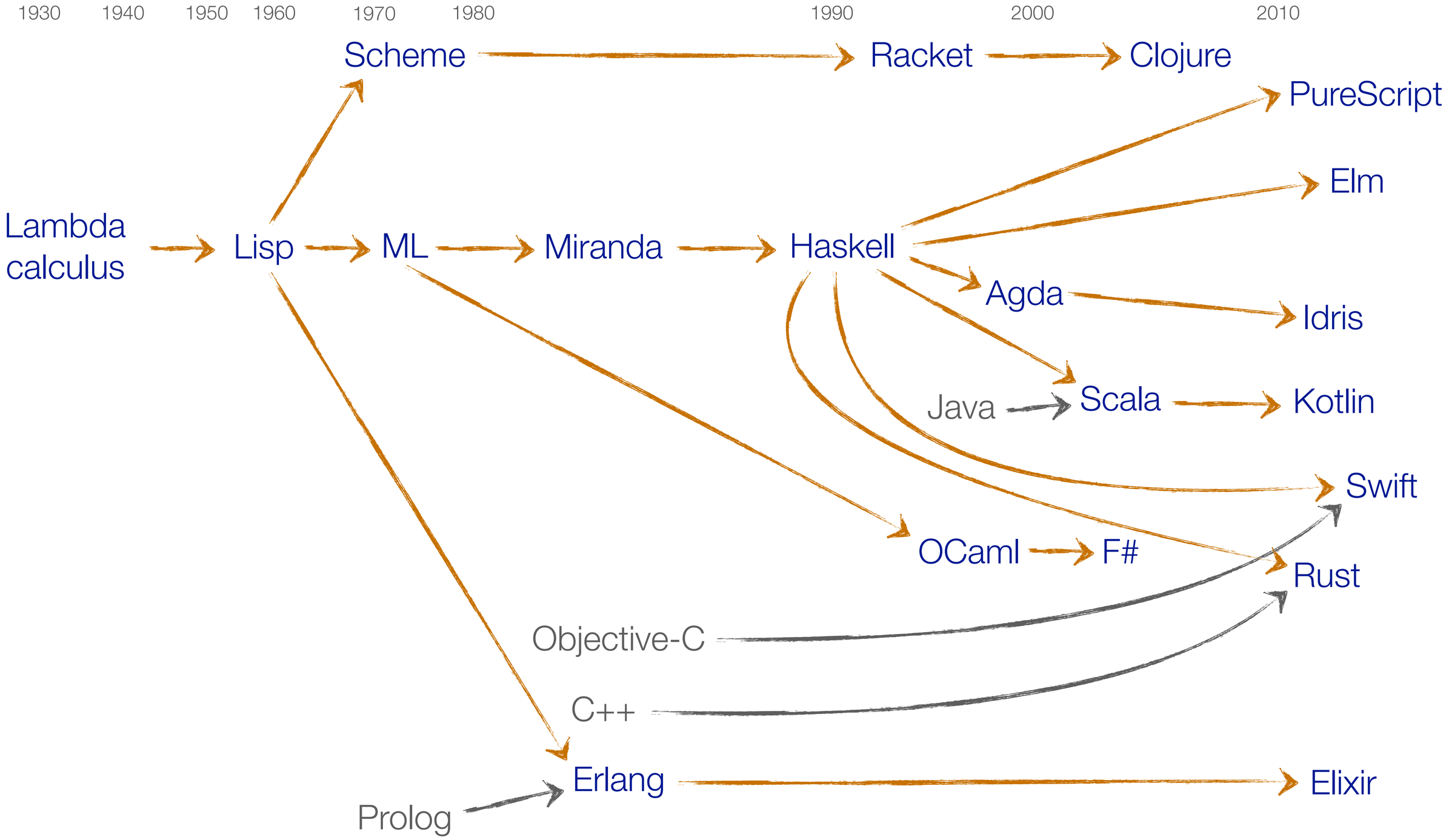
PureScript

Freeman
(2013)

Stays closer
to Haskell

Extensible effects

Row-polymorphic records


1930
1940
1950
1960
1970
1980
1990
2000
2010
Scheme
Racket
Clojure
PureScript
Elm
Lambda calculus
Lisp
ML
Miranda
Haskell
Agda
Idris
Java
Scala
Kotlin
Swift
OCaml
F#
Rust
Objective-C
C++
Erlang
Elixir
Prolog

Scala Kotlin Swift Rust

Scala	Kotlin	Swift	Rust
			
Odersky (2004)	JetBrains (2011)	Lattner (2014)	G. Hoare (2010)

Scala

Odersky
(2004)

JVM

Kotlin

JetBrains
(2011)

JVM

Swift

Lattner
(2014)

Rust

G. Hoare
(2010)

Scala	Kotlin	Swift	Rust
			
Odersky (2004)	JetBrains (2011)	Lattner (2014)	G. Hoare (2010)
JVM	JVM		

Java interoperability

Scala	Kotlin	Swift	Rust
			
Odersky (2004)	JetBrains (2011)	Lattner (2014)	G. Hoare (2010)
JVM	JVM		

Java interoperability

Hybrid, but objects first

Scala

Odersky
(2004)

JVM

Kotlin

JetBrains
(2011)

JVM

Swift

Lattner
(2014)

Objective-C
object system

Rust

G. Hoare
(2010)

Java interoperability

Hybrid, but objects first

Scala

Odersky
(2004)

JVM

Java interoperability

Hybrid, but objects first

Kotlin

JetBrains
(2011)

JVM

Swift

Lattner
(2014)

Objective-C
object system

Value types

Rust

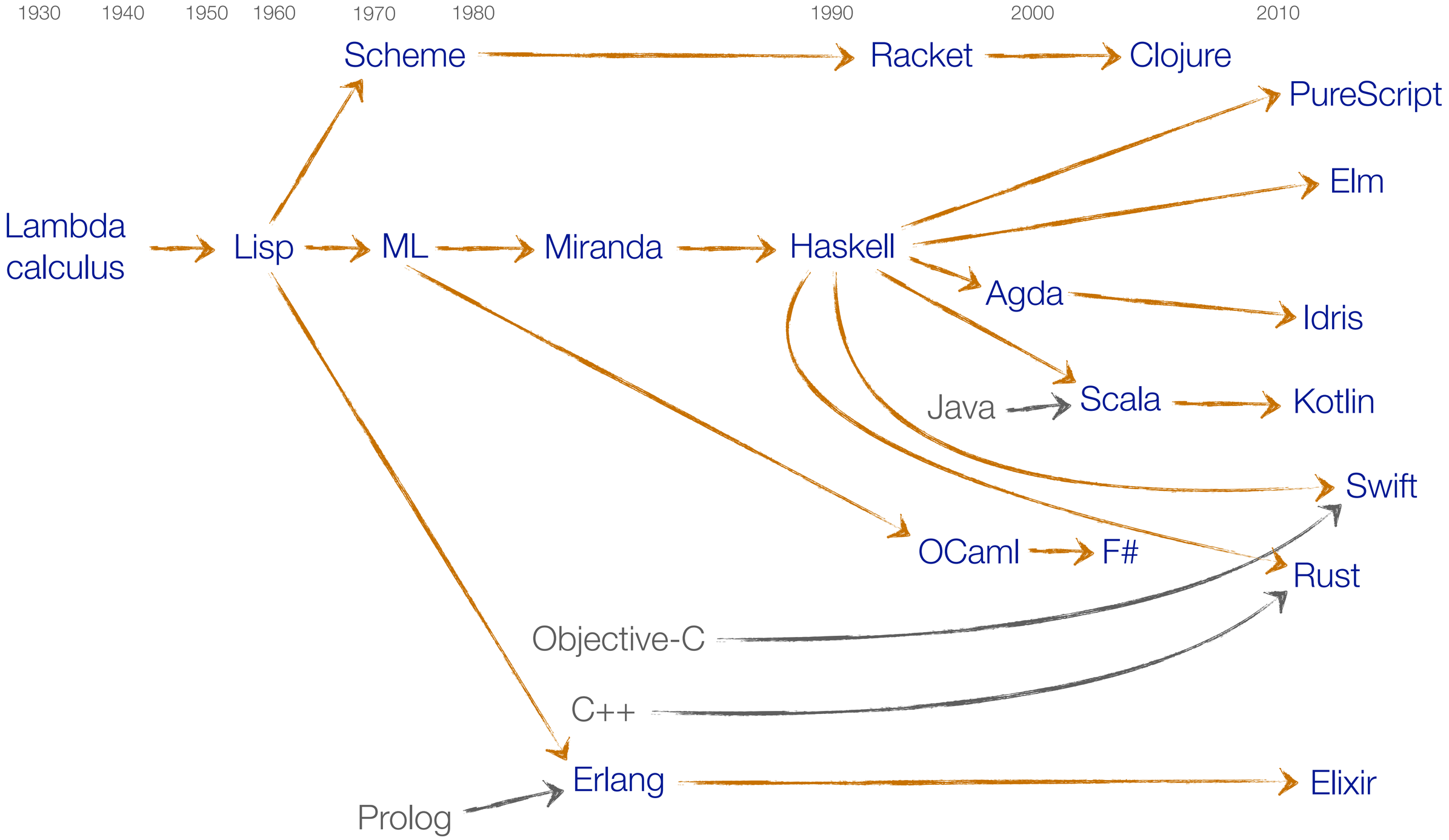
G. Hoare
(2010)

Scala	Kotlin	Swift	Rust
			
Odersky (2004)	JetBrains (2011)	Lattner (2014)	G. Hoare (2010)
JVM	JVM	Objective-C object system	
Java interoperability		Value types	
Hybrid, but objects first		Protocol-oriented	

Scala	Kotlin	Swift	Rust
			
Odersky (2004)	JetBrains (2011)	Lattner (2014)	G. Hoare (2010)
JVM	JVM	Objective-C object system	Affine types
Java interoperability		Value types	
Hybrid, but objects first		Protocol-oriented	

Scala	Kotlin	Swift	Rust
			
Odersky (2004)	JetBrains (2011)	Lattner (2014)	G. Hoare (2010)
JVM	JVM	Objective-C object system	Affine types
Java interoperability		Value types	Performance
Hybrid, but objects first		Protocol-oriented	

Scala	Kotlin	Swift	Rust
			
Odersky (2004)	JetBrains (2011)	Lattner (2014)	G. Hoare (2010)
JVM	JVM	Objective-C object system	Affine types
Java interoperability		Value types	Performance
Hybrid, but objects first		Protocol-oriented	Systems


1930
1940
1950
1960
1970
1980
1990
2000
2010
Scheme
Racket
Clojure
PureScript
Elm
Lambda calculus
Lisp
ML
Miranda
Haskell
Agda
Idris
Java
Scala
Kotlin
Swift
OCaml
F#
Rust
Objective-C
C++
Erlang
Elixir
Prolog

Clone

`https://github.com/mchakravarty/welcome-to-fp-workshop`

Haskell for Mac Licence Code

`chak@applicative.co`