Programming Languages

Produced by:

Eamonn de Leastar (edeleastar@wit.ie)

Dr. Siobhán Drohan (sdrohan@wit.ie)

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

Programming Languages

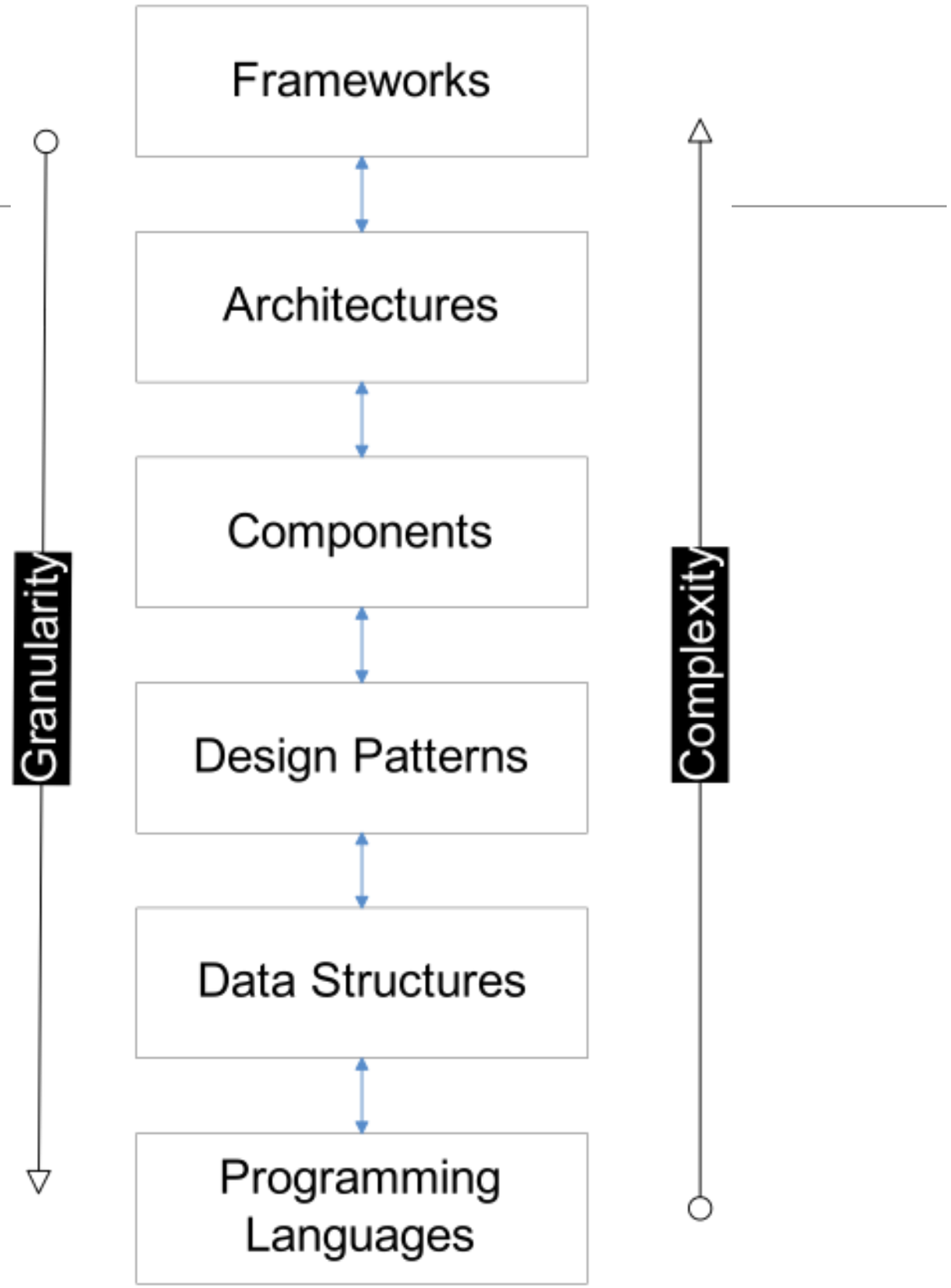
- Context
- Family Trees
- Characteristics
- Typing Spectrum
- Static vs Dynamic Typing Example
- Static to Inferred

Programming Languages

- Context
- Family Trees
- Characteristics
- Typing Spectrum
- Static vs Dynamic Typing Example
- Static to Inferred

Context

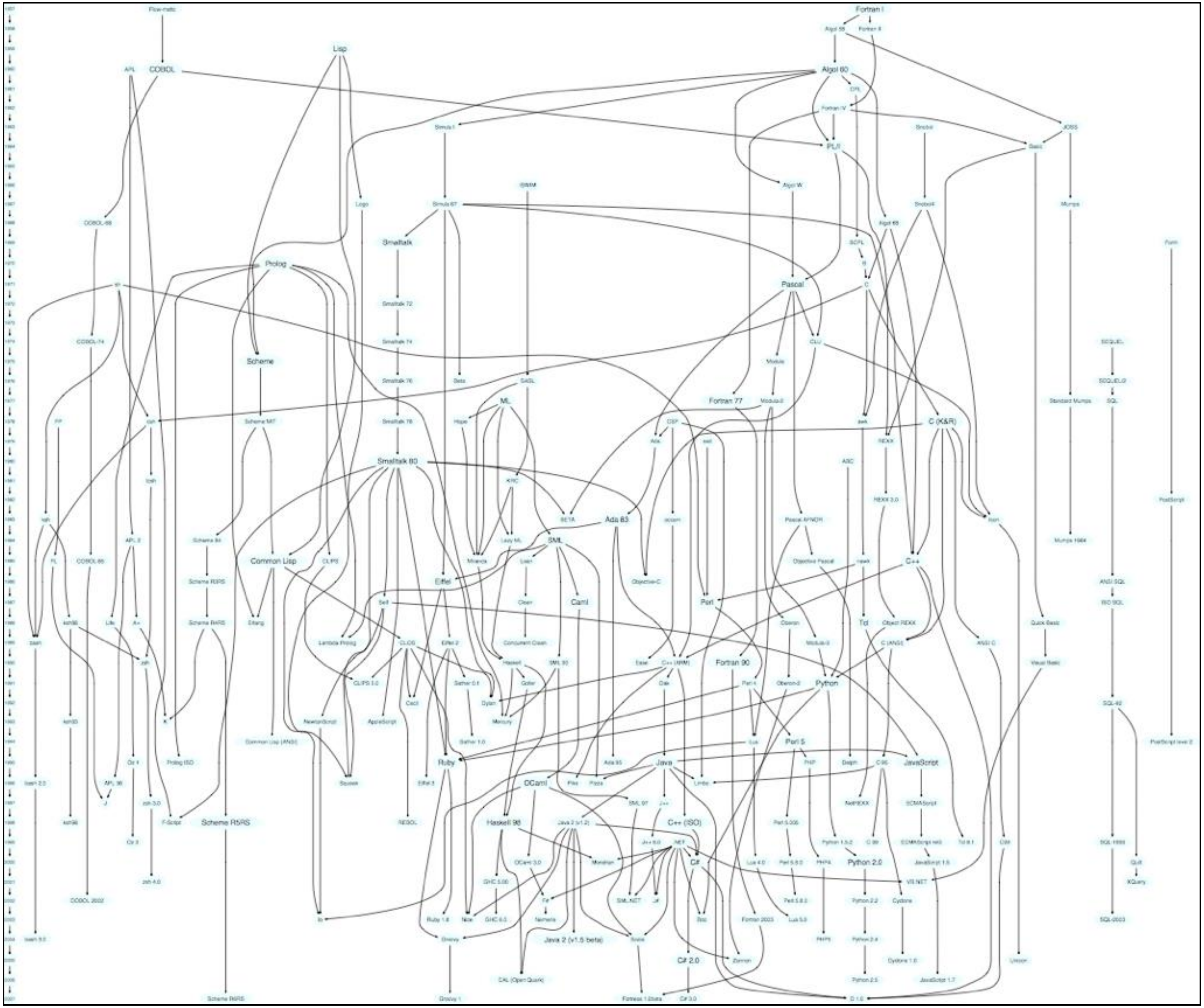
- Choice of programming language can have profound effect on accessibly of knowledge at higher layers.



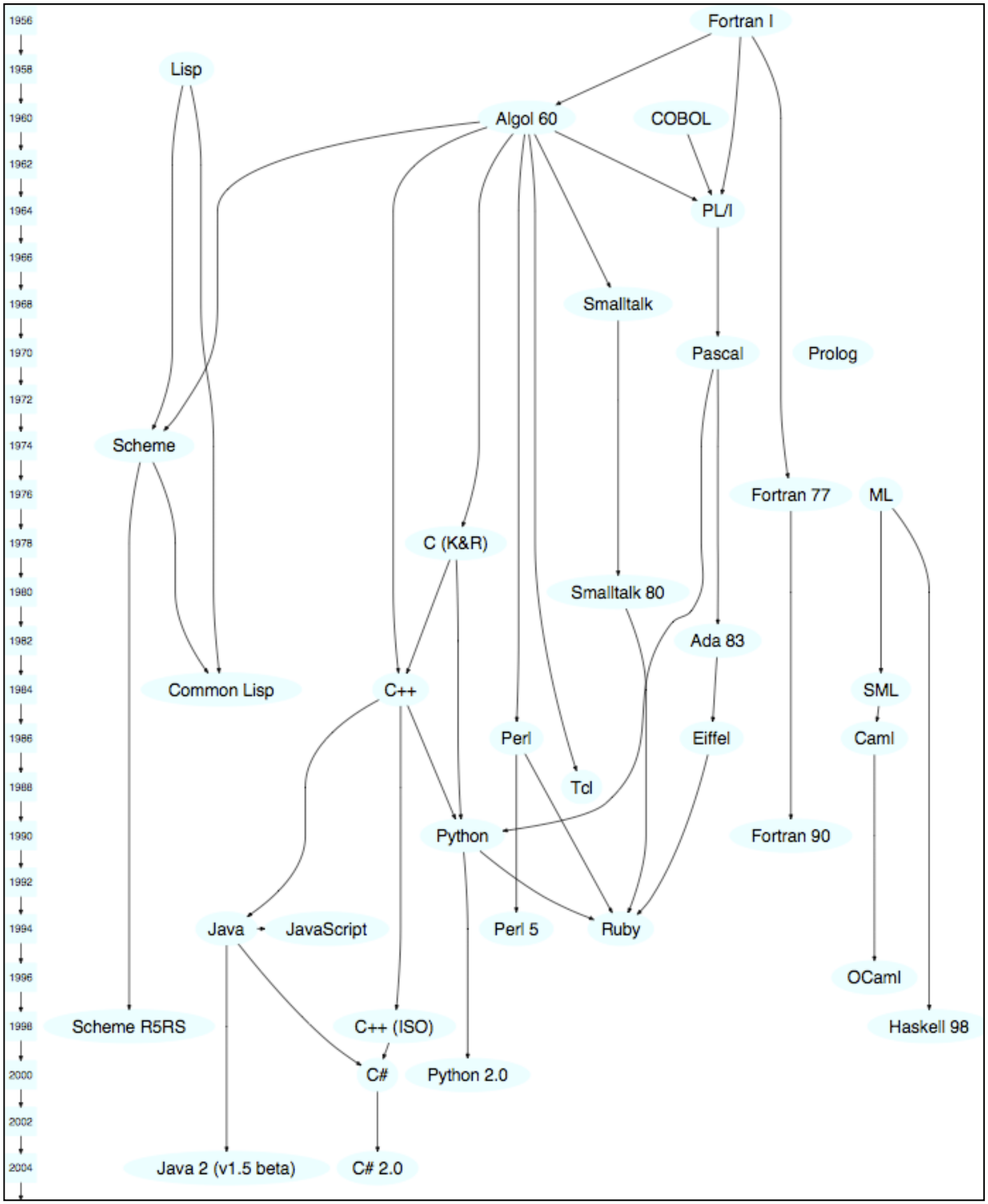
Programming Languages

- Context
- Family Trees
- Characteristics
- Typing Spectrum
- Static vs Dynamic Typing Example
- Static to Inferred

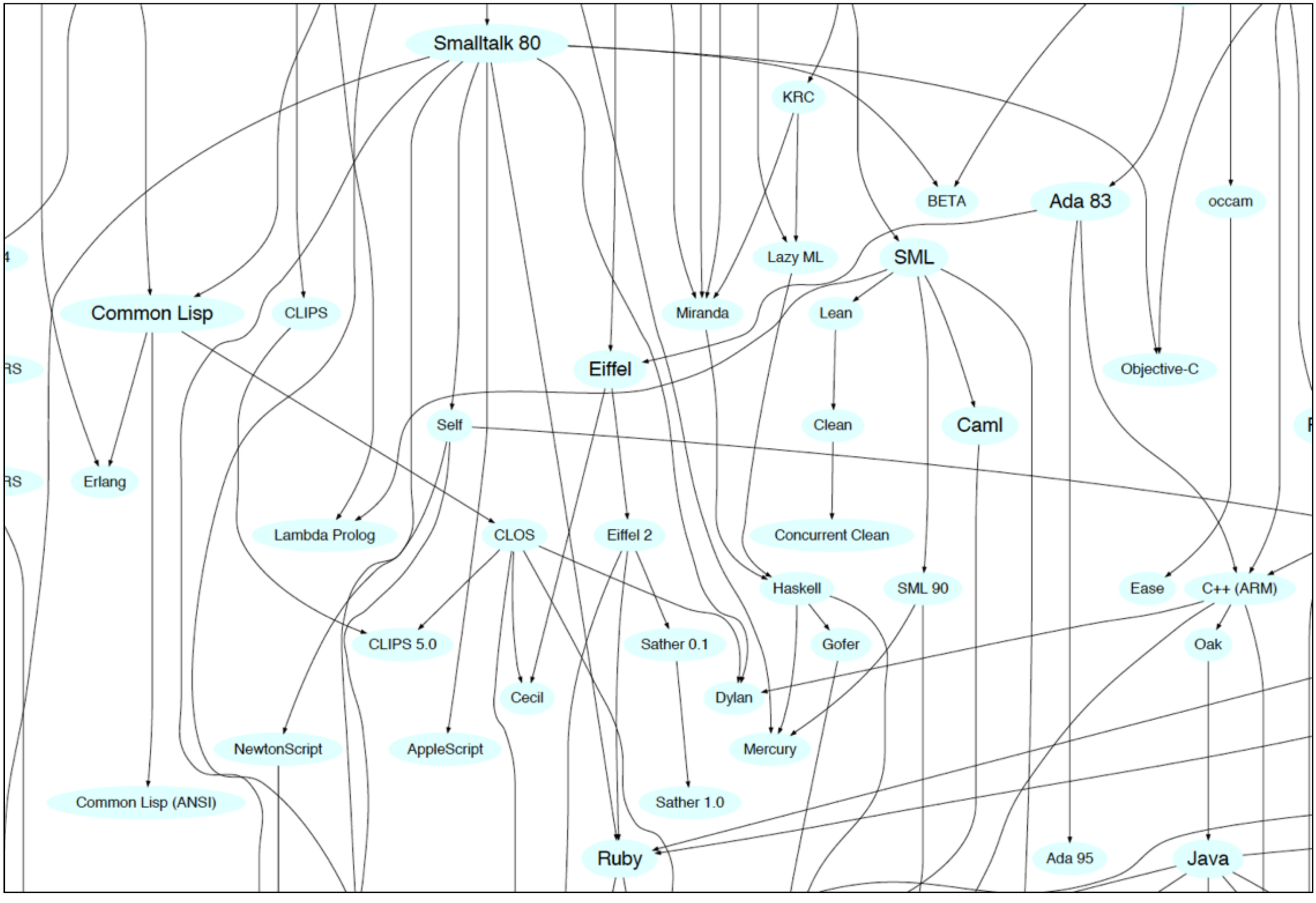
Family Tree


Fortran I
Algol 58
Lisp
APL
COBOL
Algol 60
PL/I
Smalltalk
Prolog
Pascal
Scheme
Fortran 77
ML
C (K&R)
Smalltalk 80
Ada 83
SML
Common Lisp
C++
Eiffel
Caml
Perl
Fortran 90
Python
Perl 5
Ruby
Java
JavaScript
OCaml
Haskell 98
C++ (ISO)
Scheme R5RS
C#
Python 2.0
Java 2 (v1.5 beta)
C# 2.0

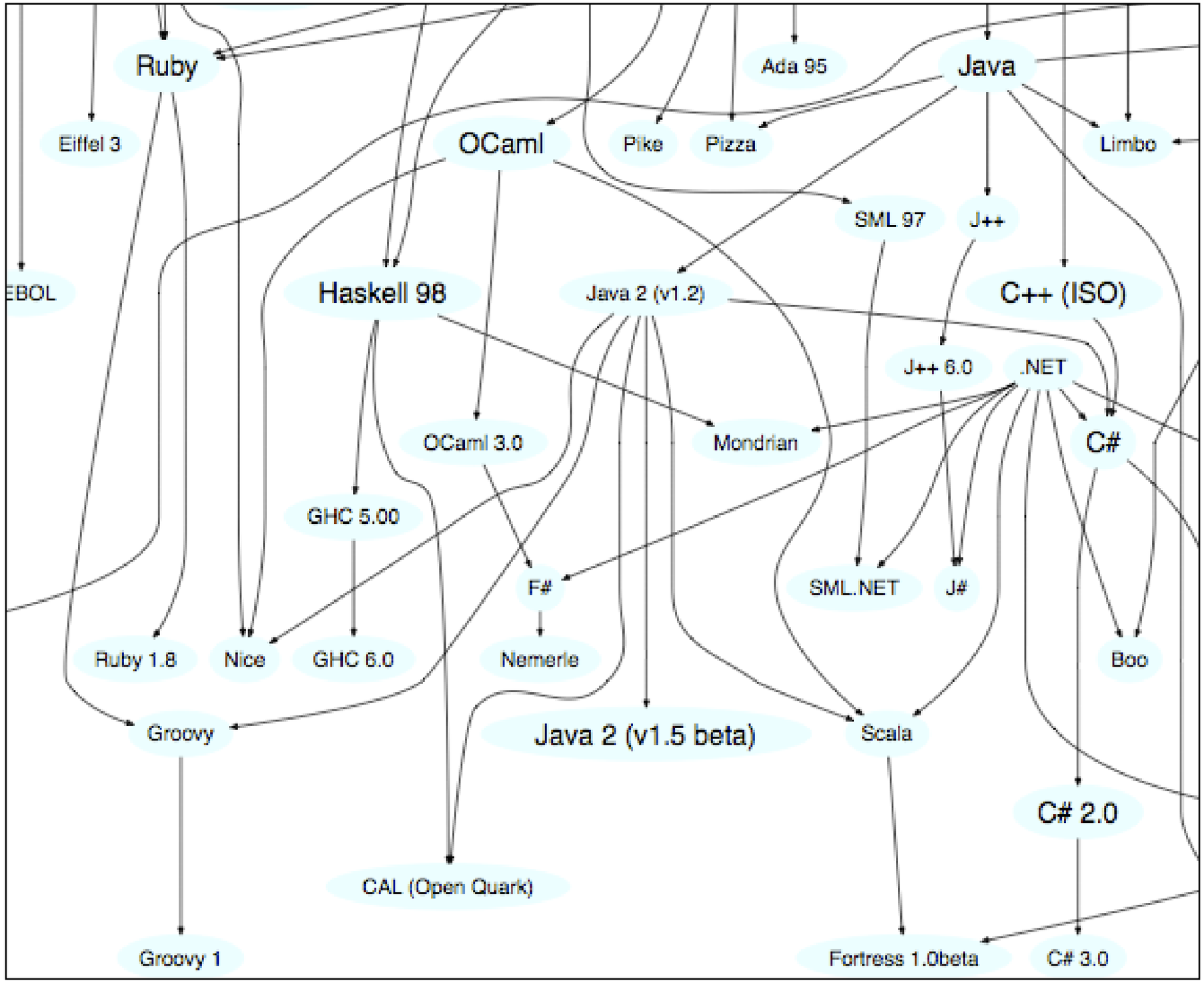
Lisp, Fortran Cluster

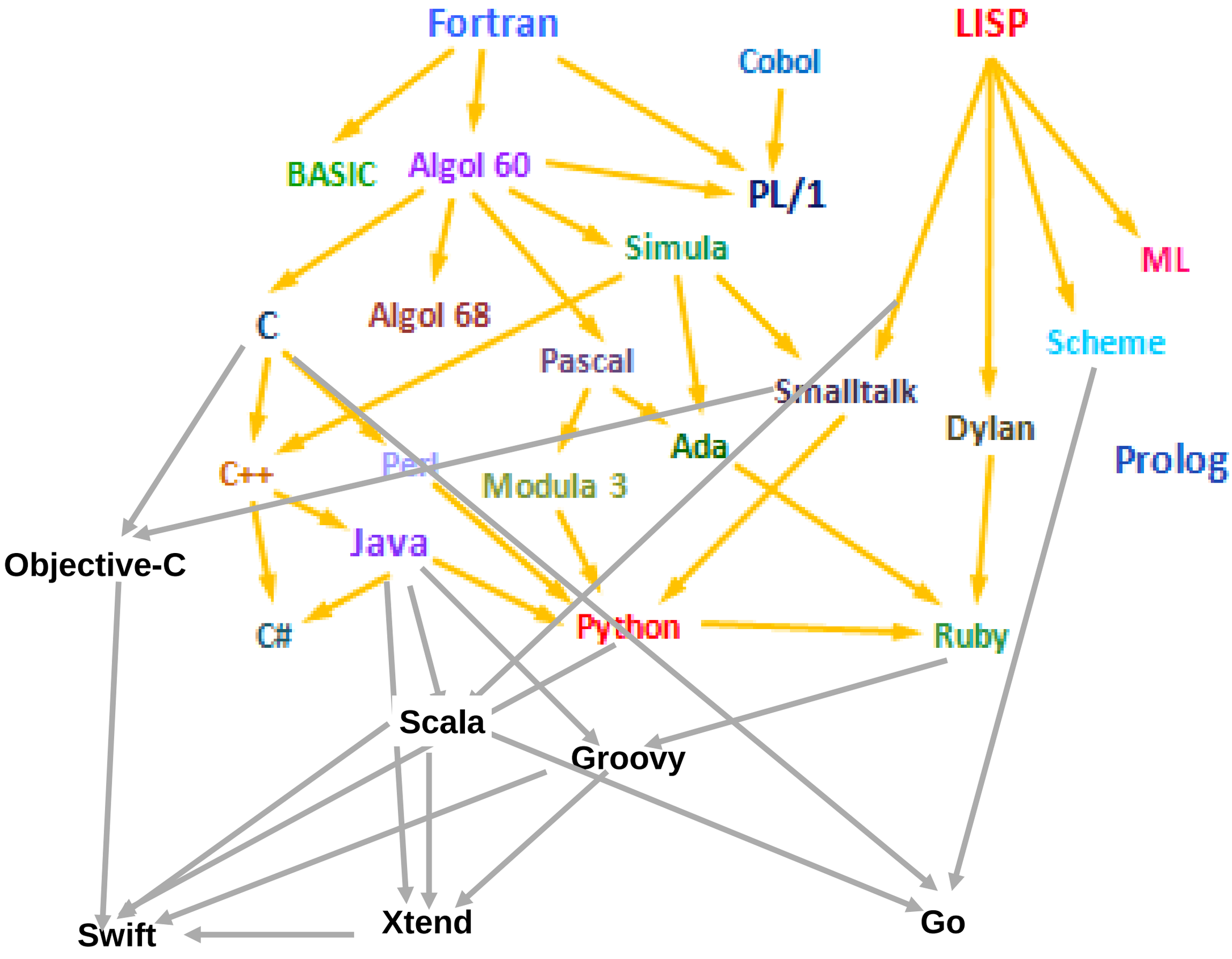


Smalltalk Cluster



Ruby, Groovy, Java, Scala Cluster




Fortran
LISP
Cobol
BASIC
Algol 60
PL/1
Simula
ML
C
Algol 68
Scheme
Pascal
Smalltalk
Dylan
Prolog
Ada
C++
Perl
Modula 3
Objective-C
Java
C#
Python
Ruby
Scala
Groovy
Swift
Xtend
Go

Programming Languages as Reactions

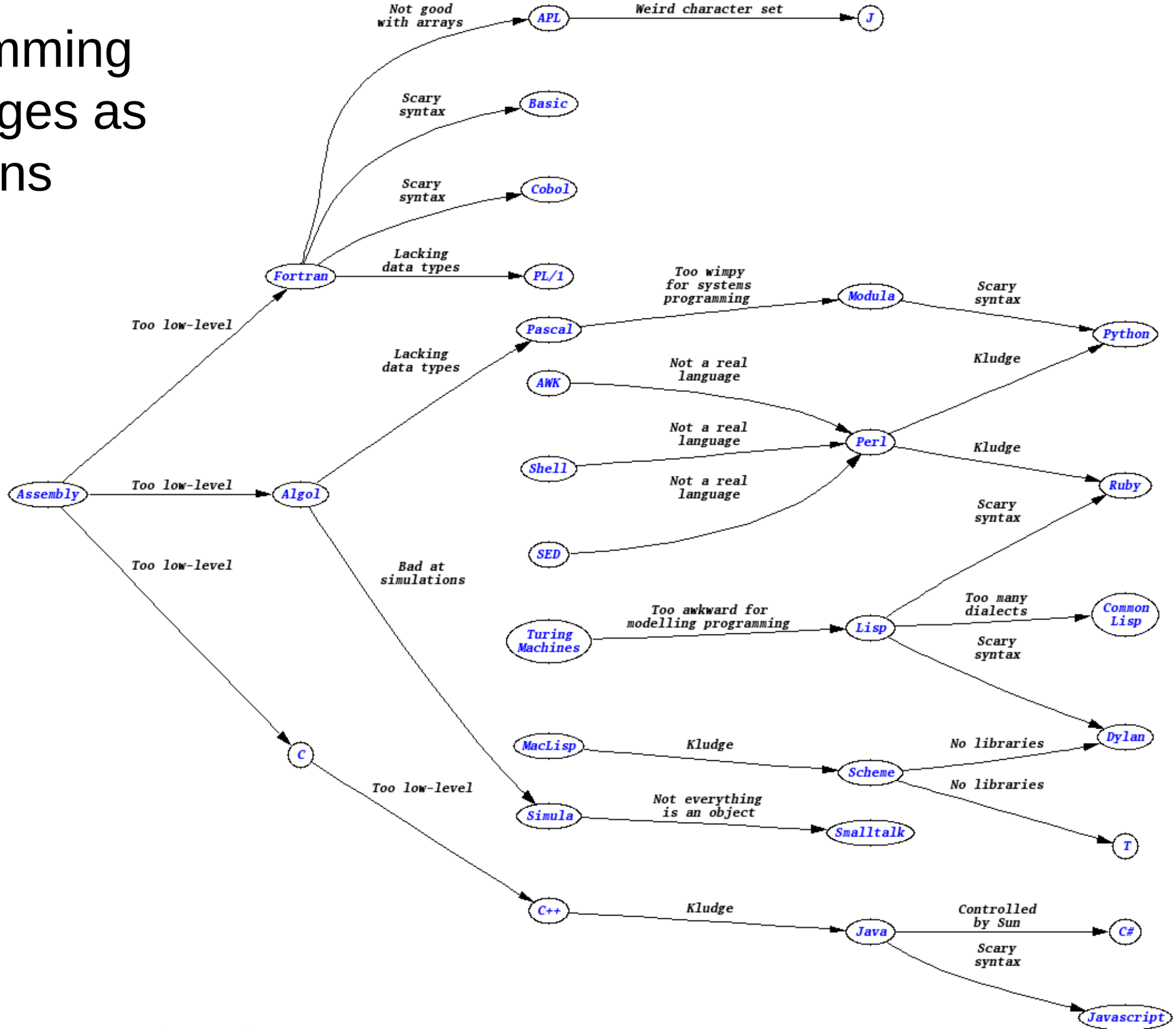
“Kevin Kelleher suggested an interesting way to compare programming languages:

to describe each in terms of the problem it fixes.

The surprising thing is how many, and how well, languages can be described this way.”

http://www.solipsys.co.uk/new/EveryLanguageFixesSomething.html

Programming Languages as Reactions



http://www.solipsys.co.uk/new/EveryLanguageFixesSomething.html

Mother Tongues

Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java, and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers – electronic lexicographers, if you will – aim to save, or at least document, the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua francas. Among the most endangered are Ada, APL, B (the predecessor of C), Lisp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *www.informatik.uni-freiburg.de/Java/misc/lang_list.html*. **– Michael Menduno**

Key

- 1954 **Year Introduced**
- **Active:** thousands of users
- **Protected:** taught at universities; compilers available
- **Endangered:** usage dropping off
- **Extinct:** no known active users or up-to-date compilers
- **Lineage continues**

Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)

1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001

Fortran — Once one of the most widely used languages in science and engineering.
Fortran IV — Created for the IBM 7090/94.
Fortran 90 ISO/IEC
Flow-Matic — Originally B-0. One of the first English-like data-processing languages.
Cobol — The culprit behind many Y2K bugs. Also known as Common Business-Oriented Language.
Cobol 61
ANSI Cobol 74
ANSI Cobol 85
OO Cobol
IPL — Possibly the first list-processing language; used in AI guru Ed Feigenbaum's PhD thesis.
Lisp — Invented by John McCarthy at MIT, Lisp has an unusual syntax made up of lots of nested parentheses. Still popular with AI researchers.
George Radin's attempt to combine the best features of Fortran, Cobol, and Algol 60.
A Lisp-like language designed by MIT researcher Seymour Papert to make programming accessible to children through graphics and simple syntax.
A Lisp offshoot that's still used in introductory programming classes at universities.
Scheme
Scheme 84
Scheme IEEE
Scheme R5RS
Haskell 1.0
Haskell 1.1
Logo
Object Logo
Clos
Common Lisp Object System
Common Lisp
ANSI Common Lisp
PL/1
Algorithmic Language, designed as a portable language for scientific computations. Algol 68's complexity was either ahead of its time or rightfully doomed, depending on whom you ask.
ANSI PL/1
PL/M
Programming Language/Microcomputers
Sell
A simple object-oriented language.
Self 4.0
The version Xerox released to the public.
Smalltalk
The first language to show off the power of object-oriented code. Developed by Software Concepts Group, Xerox PARC, led by Alan Kay.
Smalltalk-80
NetRexx
Initially called Oak. Still one of the fastest-growing languages around, despite the standards feud between Sun and Microsoft. Somewhat like C++, Java allows for "write once, run anywhere" portability across the Net.
Algol 58 — The hoped-for Esperanto of the computing world. Designed by an international Zurich-based committee as a universal language, it was one of the first attempts at making software more portable. Initially called International Algebraic Language.
Algol 60
Rex 1.00
Rex 2.00
Restructured Extended Executor
An object-oriented language.
A simple language (A = argument, B = basic value, C = TI.
Interactive object-oriented language.
Object Rexx
Simula — Popular in Europe during the '70s, Simula introduced the now-standard concept of object-oriented, rather than procedural, programming. Designed by Kristen Nygaard and Ole-Johan Dahl.
Simula 67 — All-purpose version of Simula.
Java
Java 2 (v1.3)
Prolog — Programmation en Logique for natural-language processing. Popular for AI programs.
Cla — Called Cluster, it's an object-oriented academic language created to teach rigid engineering skills.
Macro and text-processing language used by Unix admins to manipulate large config files programmatically, rather than editing them by hand.
Objective-C
Eiffel
Eiffel 3
Microsoft's answer to Java, it is a key component of the Microsoft .Net platform for Web services.
ABC
Sather 0.1
Ruby
A hardware description language.
Now AWK, a pattern scanning and processing language.
AWK
NAWK
A popular language among Web site builders, it includes features missing from Perl.
Python
Python 1.5
ECMA Script
Possibly the most common language today. Adds object-oriented features to C.
Tool Command Language, "tickle." The duct tape of programming (a scripting language for patching together different languages).
Csm
JavaScript
Microsoft's version of JavaScript. Can't they share anything? :-)
JScript
C--, a scripting language.
C#
One of the most widely deployed languages today. Windows and Unix OSes are written mostly in C and its descendants.
C with Classes
C++
Internet C++
Concurrent C
Tcl
Tcl/TK
Tcl/Tk 8.1
BCPL — Basic Combined Programming Language
B — Used primarily for non-numeric programming.
C
ANSI C (C89)
ISO C (C95)
Updates C++ for the Net with a Java-like virtual machine, so code can run on any kind of computer.
ADA
ADA 83
ADA 95
The US Department of Defense's effort to craft a standard object-oriented language for its work. Named after Ada Lovelace, arguably the world's first computer programmer, and created by Jean Ichbiah's team at Honeywell.
Modula 3
Found in millions of Web pages. Originally dubbed LiveScript, it was renamed by Netscape marketers who licensed the name to ride Java's buzz. It has little in common with that language.
A teaching language named for French mathematician Blaise Pascal. Designed by legendary Swiss programmer Niklaus Wirth for simplicity, in reaction to the complexity of Algol 68.
Modula — Modular Language
Modula 2 ISO
Object-oriented Pascal, designed for simplicity.
Object Pascal
Oberon
Another creation of Niklaus Wirth. Later modified by Robert Griesemer for numerical apps on supercomputers.
Delphi
Pascal
Csh
The Swiss Army Knife of programming (aka Practical Extraction and Report Language), used for patching together different languages. Spawned a quasi-literary culture that writes Perl haiku.
Perl 1.000
Perl 4.000
sh
Ksh
String-processing language for text and formula manipulation, common in text processors. Invented by David Farber, Ralph Griswold, and Ivan Polonsky at Bell Labs.
Programmer's easy street – with this descendant of Snobol, there's no need to know the underlying OS.
C-Shell, a scripting language and command-shell interpreter. Written by C programmers to make Unix command lines more like C's.
Standard ML, a general-purpose language.
The kitchen sink of command-line programming features for Unix, based on the original Bourne shell (aka Bourne Again Shell).
Bash
Frame-based language.
King of the one-liners, also referred to as A Programming Language. An algebra-like language with a special set of characters for writing complex programs as short formulas. Designed by Kenneth Iverson at Harvard.
Snobol
Snobol4
A general-purpose language known for its beauty and grace. Designed by Ralph Griswold to be successor to Snobol4.
Icon
ML — Meta Language
SML
SML 90
SML 97
Word-based language first used to guide the National Radio Astronomy Observatory telescope at Kitt Peak in Arizona.
Forth
Page-description language for printers and graphics systems; 75 percent of all commercial documents are produced on PostScript printers.
Caml
Caml 2-6.1
Objective Caml
O Caml 2
O Caml 3.00
PostScript
PostScript Level 2
Categorical Abstract Machine Language
J
Although mocked by "real" programmers for its limited abilities, Basic has outlived many more advanced languages, as well as the RadioShack TRS-80 computers that made it a household word. Stands for Beginner's All-Purpose Symbolic Instruction Code.
APL
APL 96
Microsoft Basic
Basic
MS Basic 2.0
Visual Basic
The Rodney Dangerfield of programming languages. Popular for building Web sites with Microsoft's Visual Studio tools.

Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

Programming Languages

- Context
- Family Trees
- Characteristics
- Typing Spectrum
- Static vs Dynamic Typing Example
- Static to Inferred

Paul Graham’s Wish List for a Programming Language

- Lisp was designed in late 1950s.
- It was a radical departure from existing languages e.g. Fortran.
- It embodied nine new ideas, which can be looked upon as a wish list for a programming language.

http://www.paulgraham.com/icad.html

Paul Graham's Wish List for a Programming Language

Wish List	Description / Example
Conditionals	If-the-else constructs are taken for granted now, but Fortran didn’t have them.
A function type	Functions as data type just like integers or strings and can be stored in variables, passed as arguments, etc.
Recursion	The solution to a problem depends on solutions to smaller instances of the same problem i.e. by allowing a function to call itself within the program text.
Dynamic typing	Where values have types, not the variables.
Garbage collection	Automatic memory management by reclaiming memory occupied by objects that are no longer in use.
Programs composed of expressions	As opposed to a series of statements.
A symbol type	Symbols are effectively pointers to strings stored in a hash table. So you can test equality by comparing a pointer, instead of comparing each character.
A notation for code using trees of symbols and constants	e.g. expressing programs directly in parse trees that get built behind the scenes.
The whole language there all the time	i.e. no real distinction between read-time, compile-time and runtime.

Java

Wish List
Conditionals
A function type (from Java 8)
Recursion
Dynamic typing
Garbage collection
Programs composed of expressions
A symbol type
A notation for code using trees of symbols and constants
The whole language there all the time

http://www.paulgraham.com/icad.html

Groovy/Ruby/Python/Scala/Xtend
(from Neal Ford)

Wish List
Conditionals
A function type
Recursion
Dynamic typing
Garbage collection
Programs composed of expressions
A symbol type
A notation for code using trees of symbols and constants
The whole language there all the time

+ Metaprogramming

e.g. Xtend can generate Java code on the fly.

http://www.paulgraham.com/icad.html

Programming Languages

- Context
- Family Trees
- Characteristics
- Typing Spectrum
- Static vs Dynamic Typing Example
- Static to Inferred

What is Dynamic Typing?

“Variables’ type declarations are not mandatory and they will be generated/inferred on the fly, by their first use.”

http://blogs.agilefaqs.com/2011/07/11/dynamic-typing-is-not-weak-typing/

What is Static Typing?

“Variable declarations are mandatory before usage, else results in a compile-time error"

http://blogs.agilefaqs.com/2011/07/11/dynamic-typing-is-not-weak-typing/

http://blogs.agilefaqs.com/2011/07/11/dynamic-typing-is-not-weak-typing/

What is Strong Typing?

“Once a variable is declared as a specific data type, it will be bound to that particular data type.

You can explicitly cast the data type though.”

http://blogs.agilefaqs.com/2011/07/11/dynamic-typing-is-not-weak-typing/

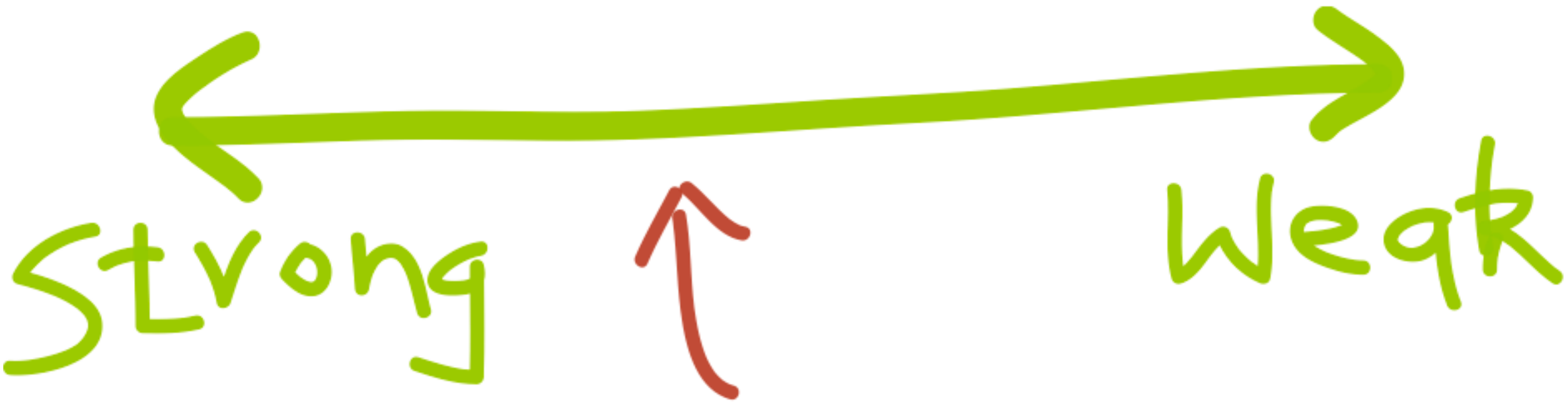
What is Weak Typing?

“Variables are not of a specific data type.

However it doesn’t mean that variables are not “bound” to a specific data type.

In weakly typed languages, once a block of memory is associated with an object it can be reinterpreted as a different type of object.”

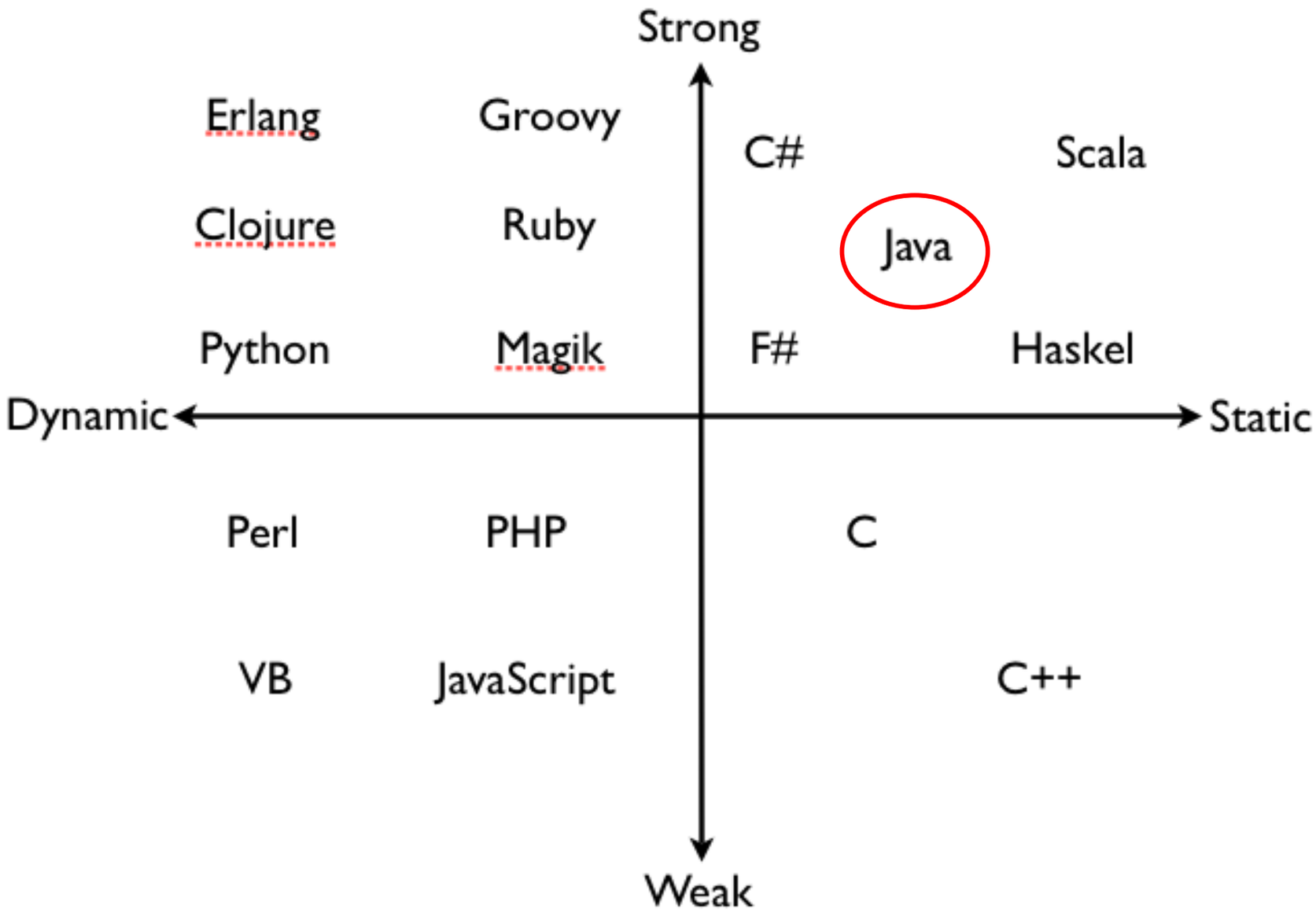
http://blogs.agilefaqs.com/2011/07/11/dynamic-typing-is-not-weak-typing/



How the runtime constraints you from treating objects of different types (in other words treating memory as blobs or specific data types)

http://blogs.agilefaqs.com/2011/07/11/dynamic-typing-is-not-weak-typing/

Typing Spectrum



http://blogs.agilefaqs.com/2011/07/11/dynamic-typing-is-not-weak-typing/

Programming Languages

- Context
- Family Trees
- Characteristics
- Typing Spectrum
- Static vs Dynamic Typing Example
- Static to Inferred

Java Example

- Java algorithm to filter a list of strings
- Only printing those with 3 or less characters (in this test case).

https://dzone.com/articles/java-groovy-part-2-closures-an

```
import java.util.ArrayList;
import java.util.List;

class Erase{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Groovy 1

- Also a valid Groovy program...

```
import java.util.ArrayList;
import java.util.List;

class Erase{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

https://dzone.com/articles/java-groovy-part-2-closures-an

Groovy 1

- Do we need generics?
- What about semicolons?
- Should standard libraries be imported?

```
import java.util.ArrayList;
import java.util.List;

class Erase{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

https://dzone.com/articles/java-groovy-part-2-closures-an

Groovy 2

- *ArrayList not given a generic type.*
- *No need for semicolons.*
- *No need to import libraries.*

```
class Erase
{
  public static void main(String[] args)
  {
    List names = new ArrayList()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    Erase e = new Erase()
    List short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (String s : short_names)
    {
      System.out.println(s)
    }
  }

  public List filterLongerThan(List strings, int length)
  {
    List result = new ArrayList();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```


https://dzone.com/articles/java-groovy-part-2-closures-an

Groovy 2

- Do we need the static types?
- Must we always have a main method and class definition?

```
class Erase
{
  public static void main(String[] args)
  {
    List names = new ArrayList()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    Erase e = new Erase()
    List short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (String s : short_names)
    {
      System.out.println(s)
    }
  }

  public List filterLongerThan(List strings, int length)
  {
    List result = new ArrayList();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```


https://dzone.com/articles/java-groovy-part-2-closures-an

Groovy 3

- *Types removed in method signature.*
- *main method and class definition removed.*

```
def filterLongerThan(strings, length)
{
  List result = new ArrayList();
  for (String s : strings)
  {
    if (s.length() < length + 1)
    {
      result.add(s)
    }
  }
  return result
}

List names = new ArrayList()
names.add("Ted")
names.add("Fred")
names.add("Jed")
names.add("Ned")
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
for (String s : short_names)
{
  System.out.println(s)
}
```

https://dzone.com/articles/java-groovy-part-2-closures-an

Groovy 3

- Should we have a special notation for lists?
- And special facilities for list processing?

```
def filterLongerThan(strings, length)
{
  List result = new ArrayList();
  for (String s : strings)
  {
    if (s.length() < length + 1)
    {
      result.add(s)
    }
  }
  return result
}

List names = new ArrayList()
names.add("Ted")
names.add("Fred")
names.add("Jed")
names.add("Ned")
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
for (String s : short_names)
{
  System.out.println(s)
}
```

https://dzone.com/articles/java-groovy-part-2-closures-an

Groovy 4

- *special notation for lists used*
- *list processing closures used.*

```
def filterLongerThan(strings, length)
{
  return strings.findAll {it.size() <= length}
}

names = ["Ted", "Fred", "Jed", "Ned"]
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
short_names.each {System.out.println(it)}
```

https://dzone.com/articles/java-groovy-part-2-closures-an

Groovy 4

- Method needed any longer?
- Is there an easier way to use common methods (e.g. println)?
- Are brackets always needed?

```
def filterLongerThan(strings, length)
{
  return strings.findAll {it.size() <= length}
}

names = ["Ted", "Fred", "Jed", "Ned"]
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
short_names.each {System.out.println(it)}
```

https://dzone.com/articles/java-groovy-part-2-closures-an

Groovy 5

- *Method removed*
- *Used common method notation*
- *Removed non necessary brackets.*

```
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
println short_names.size()
short_names.each {println it}
```

https://dzone.com/articles/java-groovy-part-2-closures-an

```java
import java.util.ArrayList;
import java.util.List;

class Erase{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

```groovy
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
println short_names.size()
short_names.each {println it}
```

Java vs Groovy?

https://dzone.com/articles/java-groovy-part-2-closures-an

Programming Languages

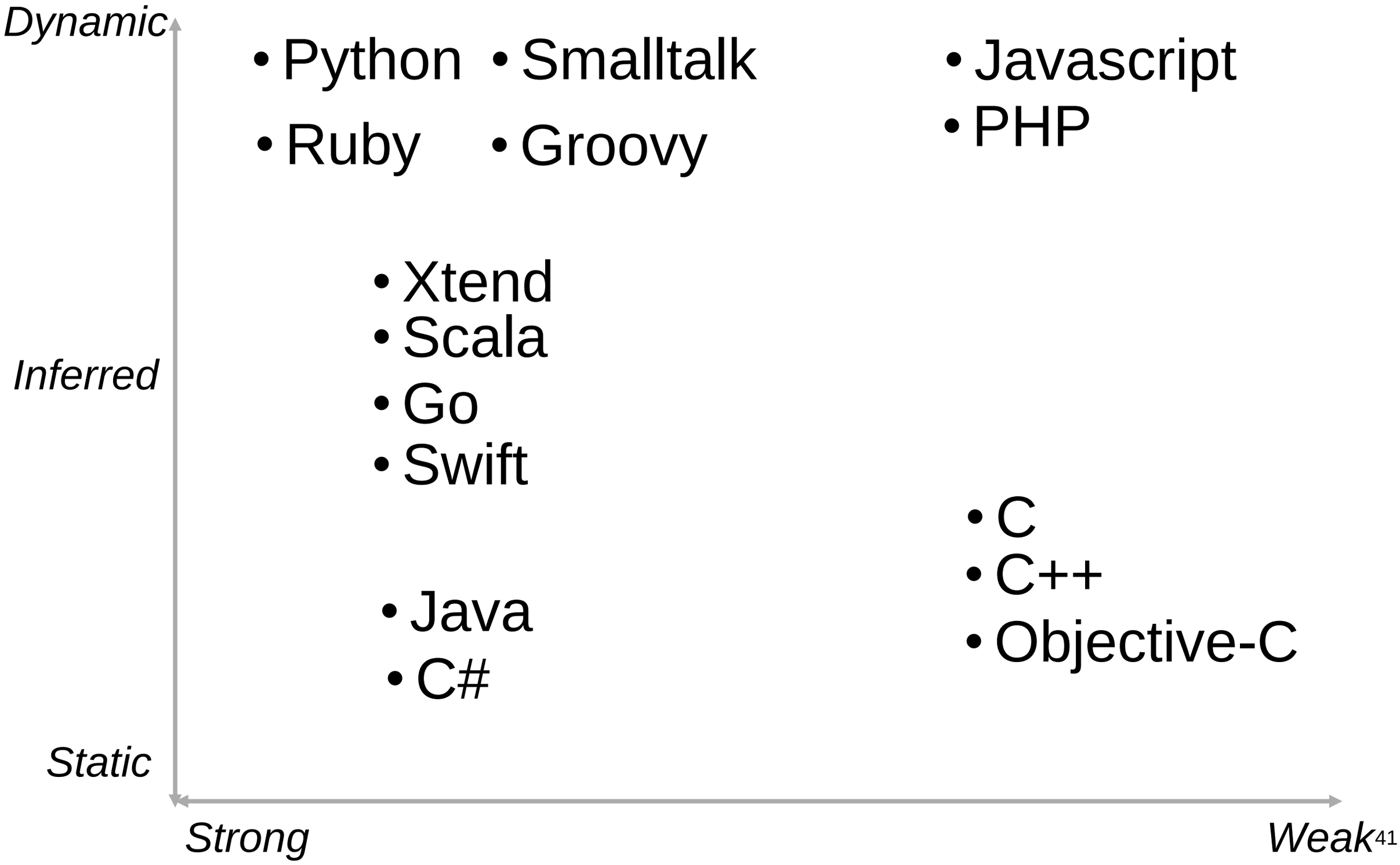
- Context
- Family Trees
- Characteristics
- Typing Spectrum
- Static vs Dynamic Typing Example
- Static to Inferred

Another Approach to Types?

- *Type Inference* : the compiler draws conclusions about the types of variables based on how programmers use those variables.
  - Yields programs that have some of the conciseness of Dynamically Typed Languages
  - But - decision made at *compile time*, not at *run time*
  - More information for static analysis - refactoring tools, complexity analysis, bug checking etc...
- Haskell, Scala, **Xtend,** Java (from 7 onwards)

```
object InferenceTest1 extends Application
{
  val x = 1 + 2 * 3              // the type of x is int
  val y = x.toString()           // the type of y is String
  def succ(x: int) = x + 1       // method succ returns int values
}
```

Typing Spectrum



Java Example -> **Xtend**

- Unlike Groovy – this is NOT an Xtend Program.

```
import java.util.ArrayList;
import java.util.List;

class Erase{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

def & var

Are semicolons necessary?

```
import java.util.ArrayList;
import java.util.List;

class Erase{
  def static void main(String[] args)
  {
    var List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    var Erase e = new Erase();
    var List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  def List<String> filterLongerThan(List<String> strings, int length)
  {
    var List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

No semicolons

```
import java.util.ArrayList;
import java.util.List;

class Erase{
  def static void main(String[] args)
  {
    var List<String> names = new ArrayList<String>()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    var Erase e = new Erase()
    var List<String> short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (String s : short_names)
    {
      System.out.println(s)
    }
  }

  def List<String> filterLongerThan(List<String> strings, int length)
  {
    var List<String> result = new ArrayList<String>()
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

Can some types be inferred?

Type inference

```
import java.util.ArrayList;
import java.util.List;

class Erase{
  def static void main(String[] args)
  {
    var names = new ArrayList<String>()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (s : short_names)
    {
      System.out.println(s)
    }
  }

  def filterLongerThan(List<String> strings, int length)
  {
    var result = new ArrayList<String>()
    for (s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

What about Collection Literals?

Collection Literals

```
import java.util.ArrayList;
import java.util.List;

class Erase{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (s : short_names)
    {
      System.out.println(s)
    }
  }

  def filterLongerThan(List<String> strings, int length)
  {
    var result = new ArrayList<String>()
    for (s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

Can Lambas simplify code?

Lambdas

```
import java.util.ArrayList;
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    short_names.forEach[System.out.println(it)]
  }

  def filterLongerThan(List<String> strings, int length)
  {
    val result = new ArrayList<String>()
    strings.forEach[ if (it.length() < length + 1)
      {
        result.add(it)
      }]
    result
  }
}
```

What are List Comprehensions?

Filters/List Comprehensions

```
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    short_names.forEach[System.out.println(it)]
  }

  def filterLongerThan(List<String> strings, int length)
  {
      val list = strings.filter[it.length() <= 3]
      list
  }
}
```

Do we need the class Erase at all?

Final Version

```
class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    println(names)
    var  short_names =  names.filter[it.length() <= 3]
    println(short_names.size())
    short_names.forEach[println(it)]
  }
}
```

```java
import java.util.ArrayList;
import java.util.List;

class Erase{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

java

```xtend
class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    println(names)
    var  short_names =  names.filter[it.length() <= 3]
    println(short_names.size())
    short_names.forEach[println(it)]
  }
}
```

xtend

```groovy
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
println short_names.size()
short_names.each {println it}
```

groovy

Back to our Java Example

- Java algorithm to filter a list of strings
- Only printing those with 3 or less characters (in this test case).

```
import java.util.ArrayList;
import java.util.List;

class Erase{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Swift

```
import Foundation

class Erase
{
  func main()
  {
    var names:String[] = String[]()
    names.append ("ted")
    names.append ("fred")
    names.append ("jed")
    names.append ("ned")
    println(names)
    var short_names:String[] = filterLongerThan(names, length:3)
    for name:String in short_names
    {
      println (name)
    }
  }

  func filterLongerThan (strings : String[], length : Int) -> String[]
  {
    var result:String[] = String[]()
    for s:String in strings
    {
      if countElements(s) < length + 1
      {
        result.append(s)
      }
    }
    return result
  }
}

var erase:Erase = Erase()
erase.main()
```

Swift

- Type Inference

```
import Foundation

class Erase
{
  func main()
  {
    var names = String[]()
    names.append ("ted")
    names.append ("fred")
    names.append ("jed")
    names.append ("ned")
    println(names)
    var short_names = filterLongerThan(names, length:3)
    for name in short_names
    {
      println (name)
    }
  }

  func filterLongerThan (strings : String[], length : Int) -> String[]
  {
    var result = String[]()
    for s in strings
    {
      if countElements(s) < length + 1
      {
        result.append(s)
      }
    }
    return result
  }
}

var erase = Erase()
erase.main()
```

Swift

- Literals

```
import Foundation

class Erase
{
  func main()
  {
    var names = ["ted", "fred", "jed", "ned"]
    var short_names = filterLongerThan(names, length:3)
    for name in short_names
    {
      println (name)
    }
  }

  func filterLongerThan (strings : String[], length : Int) -> String[]
  {
    var result = String[]()
    for s in strings
    {
      if countElements(s) < length + 1
      {
        result.append(s)
      }
    }
    return result
  }
}

var erase = Erase()
erase.main()
```

Swift

- Closures

```
import Foundation

class Erase
{
  func main()
  {
    var names = ["ted", "fred", "jed", "ned"]
    var short_names = names.filter { countElements($0) < 4 }
    for name in short_names
    {
      println (name)
    }
  }
}

var erase = Erase()
erase.main()
```

Swift

- Final version

```
import Foundation

var names = ["ted", "fred", "jed", "ned"]
println(names)
var short_names = names.filter { countElements($0) < 4 }
println(short_names)
```

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Java

```groovy
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
short_names.each {println it}
```

Groovy

```
var names = #["Ted", "Fred", "Jed", "Ned"]
println(names)
var short_names =  names.filter[it.length() <= 3]
short_names.forEach[println(it)]
```

Xtend

```
var names = ["ted", "fred", "jed", "ned"]
println(names)
var short_names = names.filter { countElements($0) < 4 }
println(short_names)
```

Swift