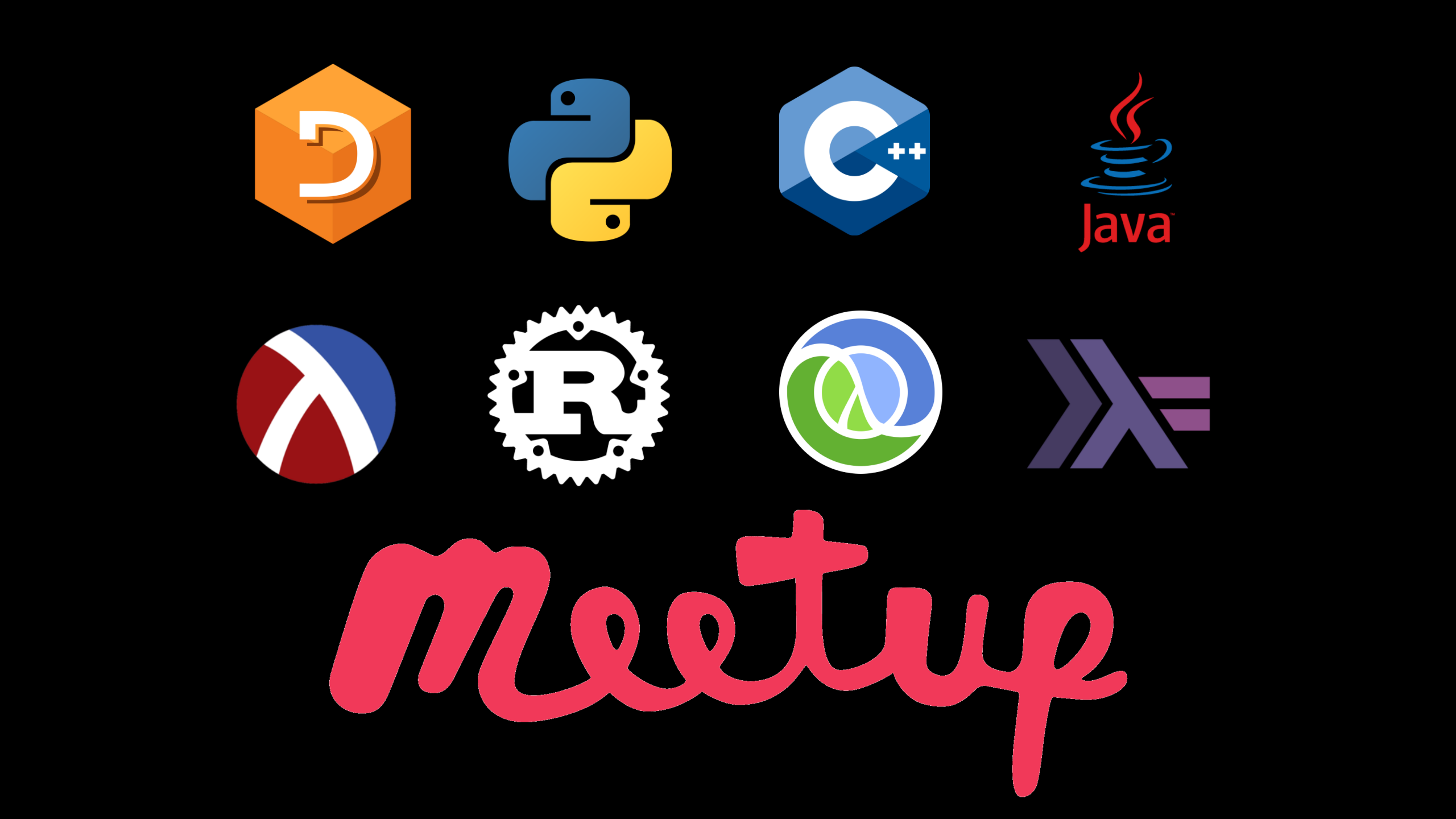
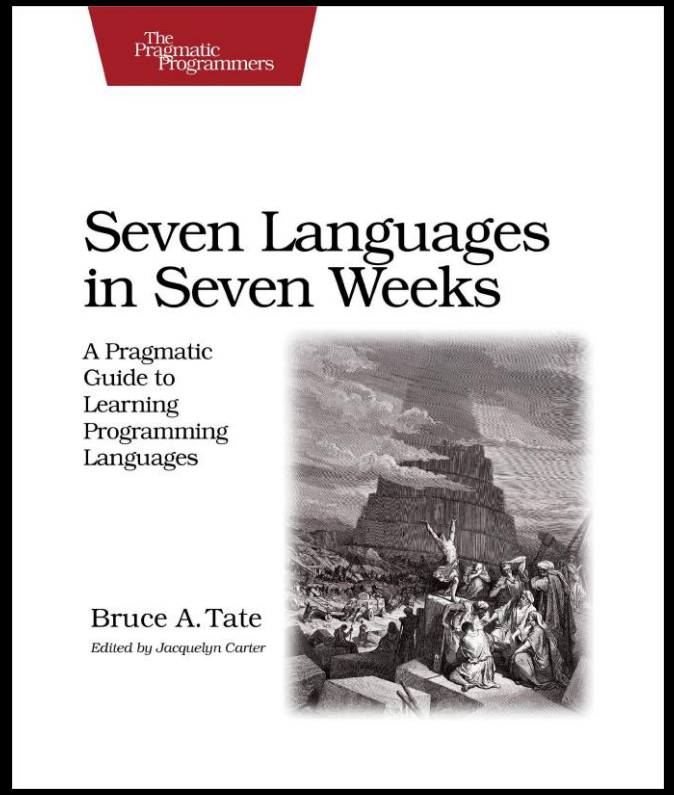

Java
meetup

# Berlin Code of Conduct


# 7 Languages in 7 Weeks

# Io





Thoughts:

- Website is beautiful
- Binary downloads from website are broken
- Building from source (github) gives you version from 2017
- Libraries listed in docs don’t match this version
- Should have chosen Self 😂
- @ / @@ were barely covered

## 3.1 Introducing Io

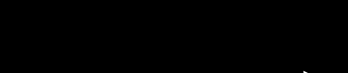
Steve Dekorte invented the Io language in 2002. It's always written with an uppercase *I* followed by a lowercase *o*. Io is a prototype language like Lua or JavaScript, meaning every object is a clone of another.

Written as an exercise to help Steve understand how interpreters work, Io started as a hobbyist language and remains pretty small today. You can learn the syntax in about fifteen minutes and the basic mechanics of the language in thirty. There are no surprises. But the libraries will take you a little longer. The complexity and the richness comes from the library design.

***Bruce Tate:*** *Why did you write Io?*

***Steve Dekorte:*** *In 2002, my friend Dru Nelson wrote a language called Cel (inspired by Self) and was asking for feedback on its implementation. I didn't feel I understood how programming languages work well enough to have anything useful to say, so I started writing a small language to understand them better. It grew into Io.*


SMALLTALK




Self


Io



Self

Io

APL [Iver79] was an interactive time-shared system that let its users write programs very quickly. Although not object-oriented, it exerted a strong influence on both Smalltalk and Self. Ingalls has reported its influence on Smalltalk [Inga81], and APL profoundly affected Ungar's experience of computing. In 1969, Ungar had entered the Albert Einstein Senior High School in Kensington, Maryland, one of only three in the country with an experimental IBM/1130 time-sharing system. Every Friday afternoon, students were allowed to program it in APL, and this was Ungar's first programming experience. Though Ungar didn't know it at the time, APL differed from most of its contemporaries: it was dynamically typed in that any variable could hold a scalar, vector, or matrix of numbers or characters. APL's built-in (and user-defined) functions were polymorphic over this range of types. It even had operators: higher-order functions that were parameterized by functions. The APL user experienced a live workspace of data and program and could try things out and get immediate feedback. Ungar sorely missed this combination of dynamic typing, polymorphism, and interpretive feel when he went on to learn such mainstream languages as FORTRAN and PL/I.



Self

Io


SMALLTALK


APL

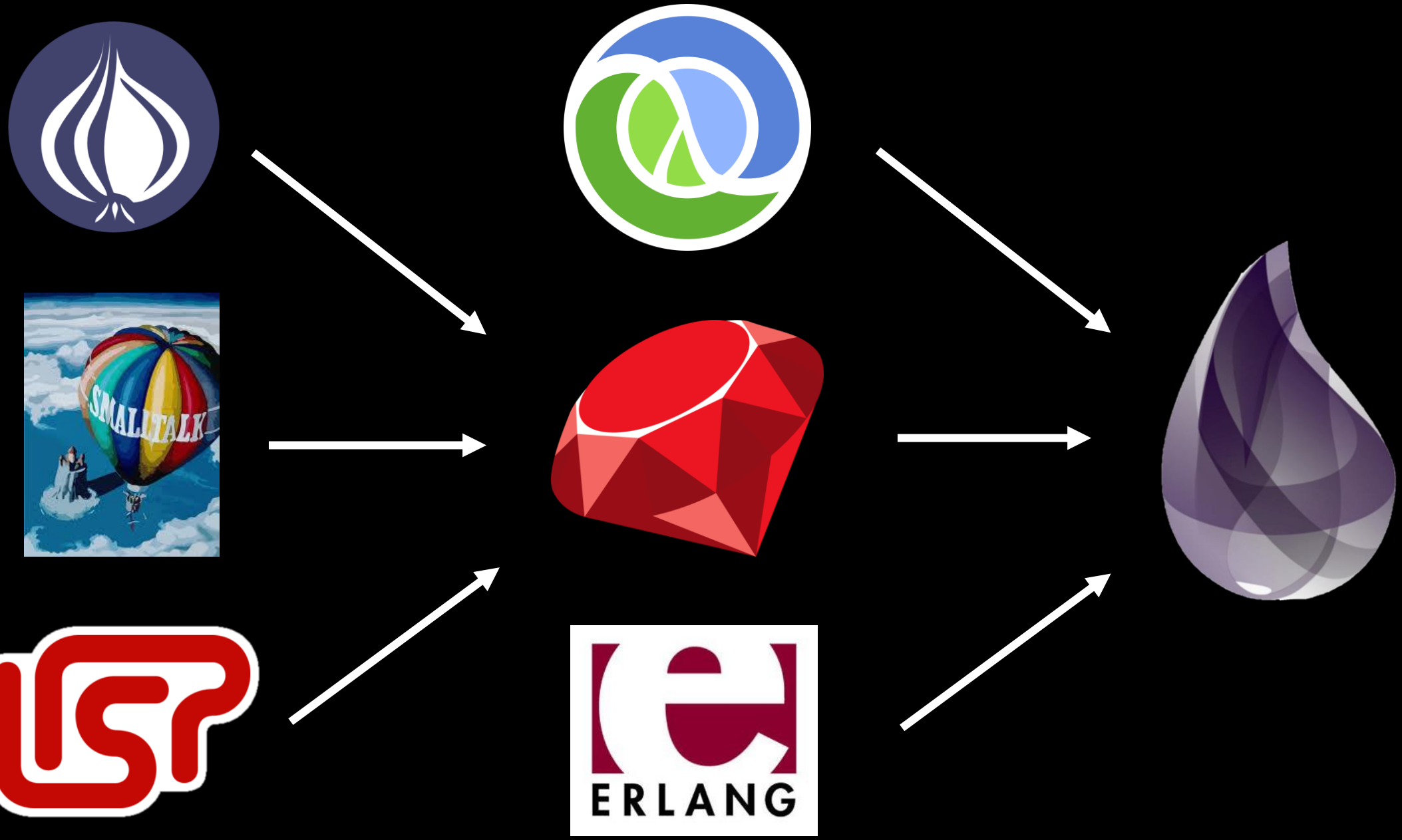

Self

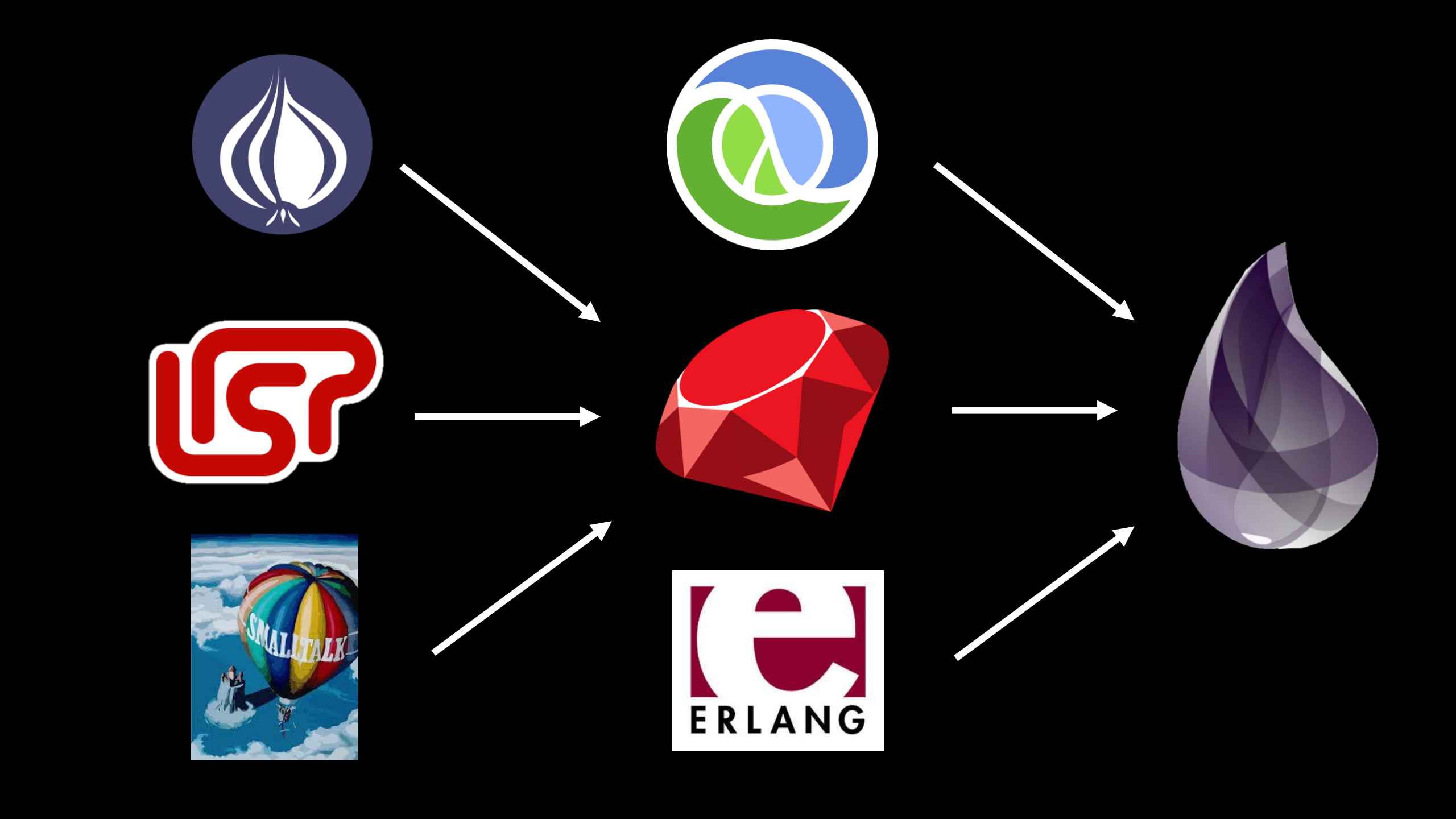

Io

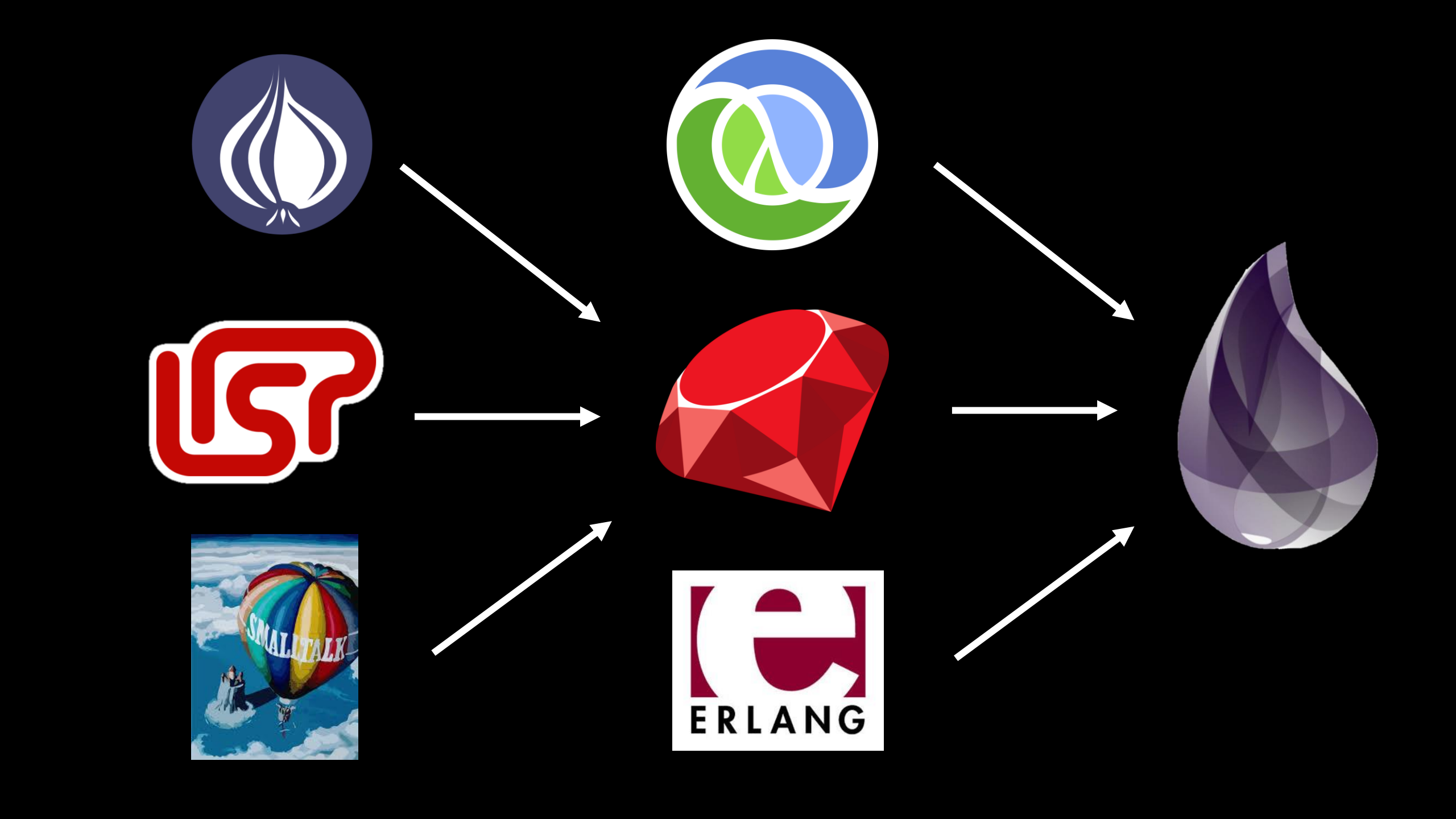

APL

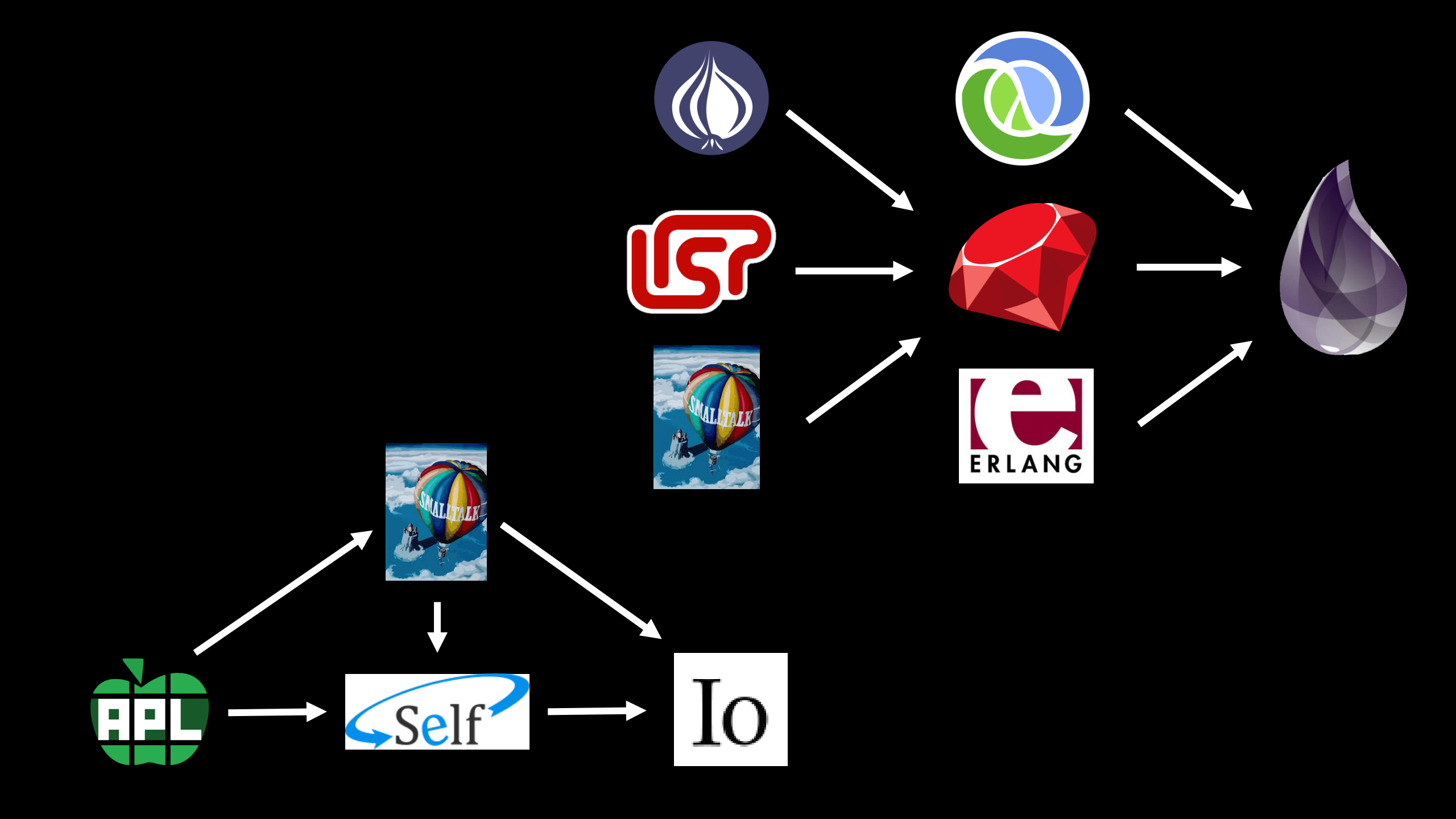
Self

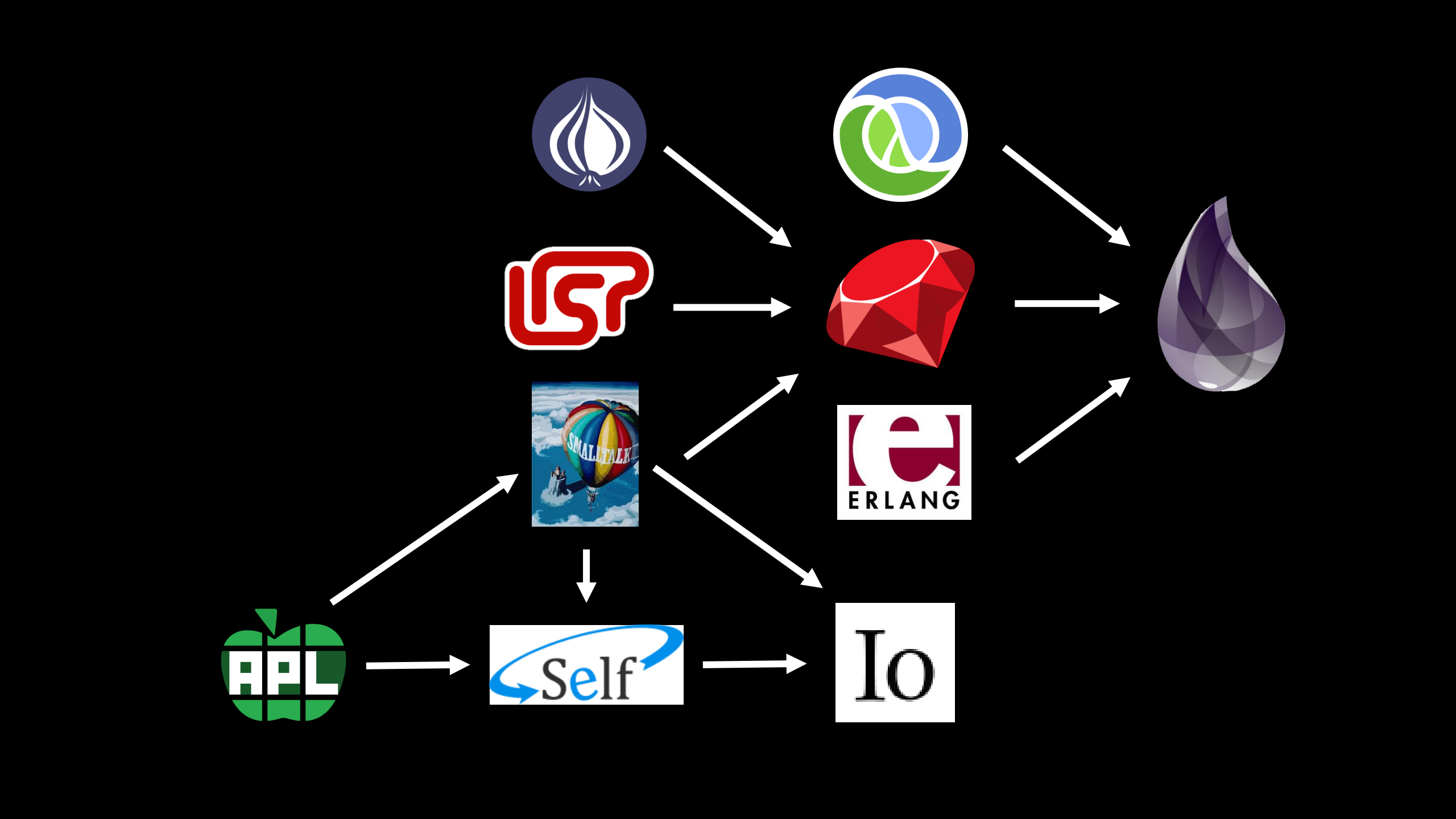
Io

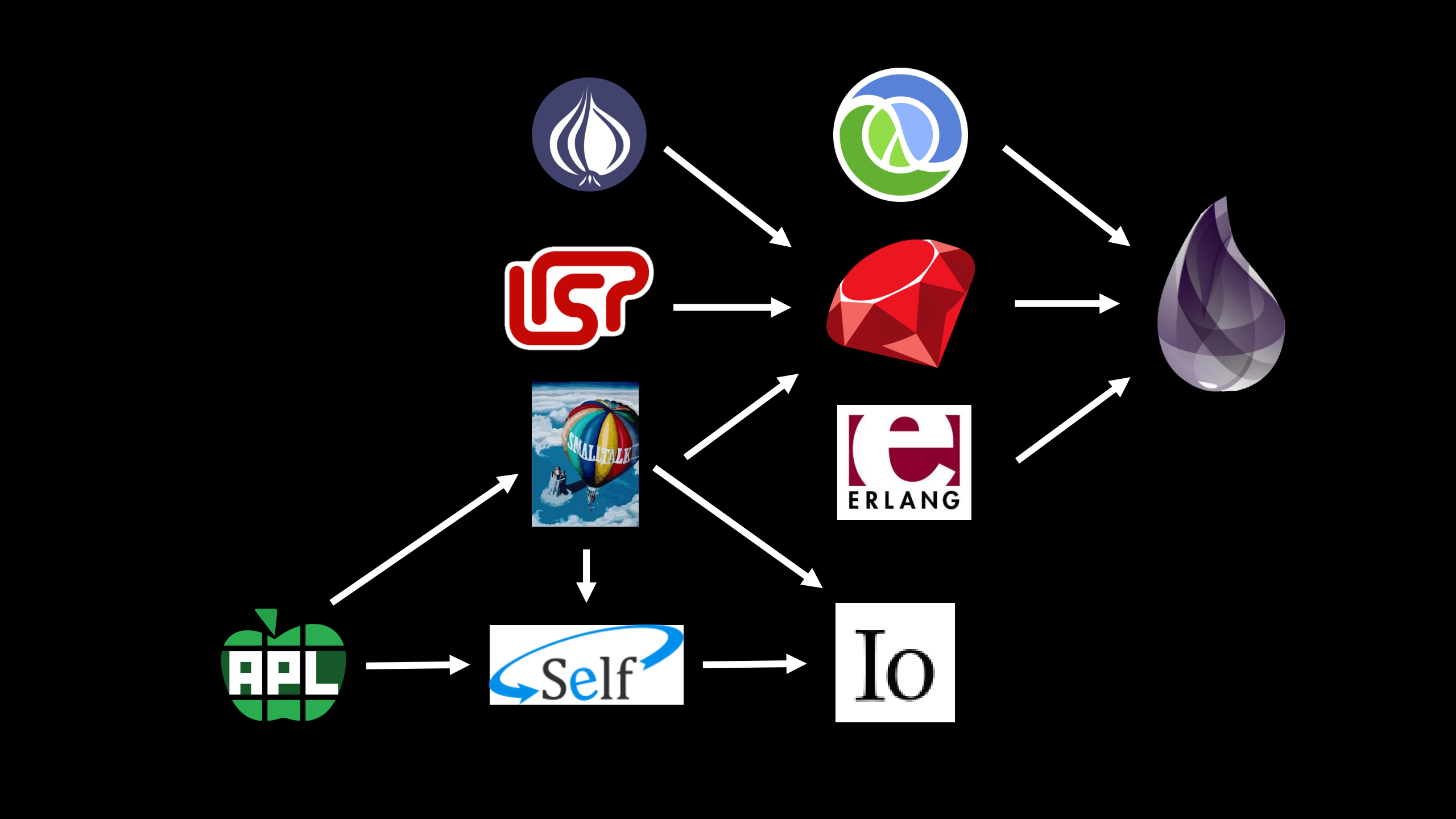

SMALLTALK
ERLANG

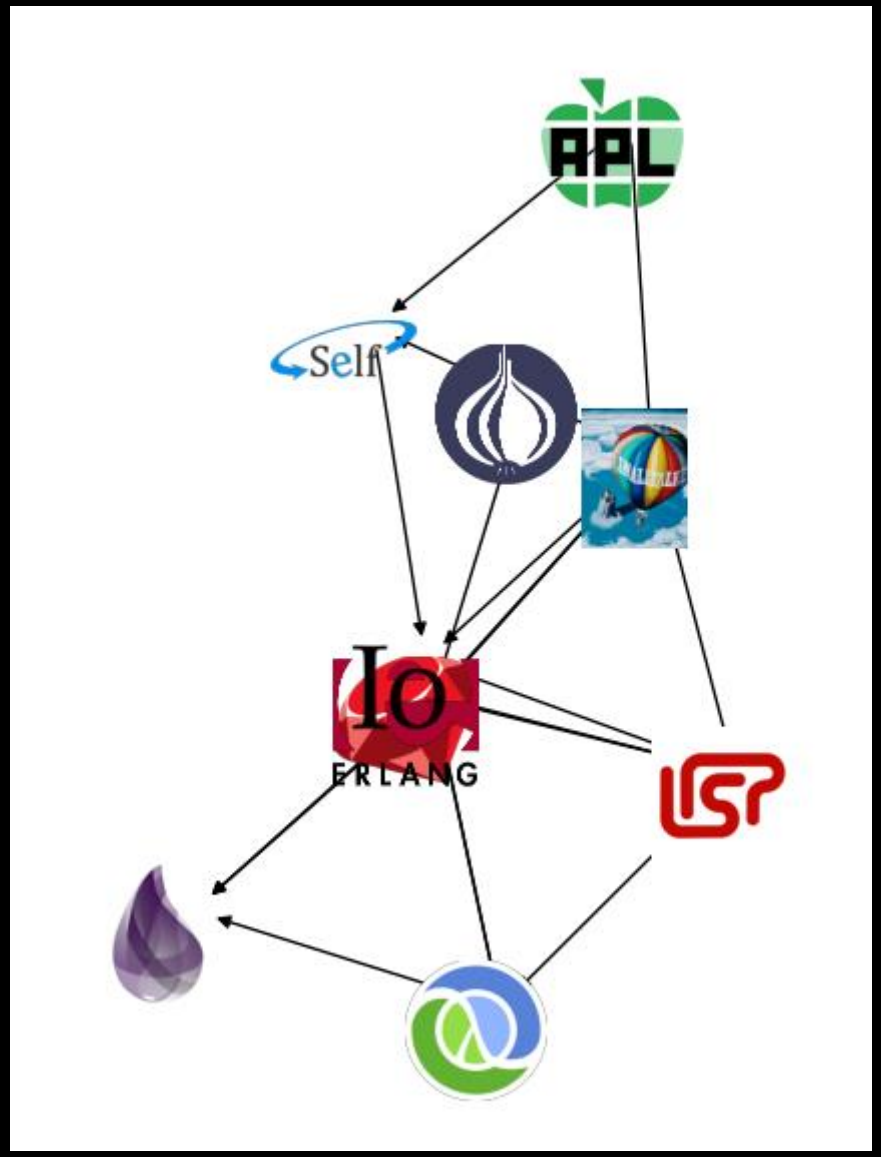

SMALLTALK
ERLANG


SMALLTALK
ERLANG

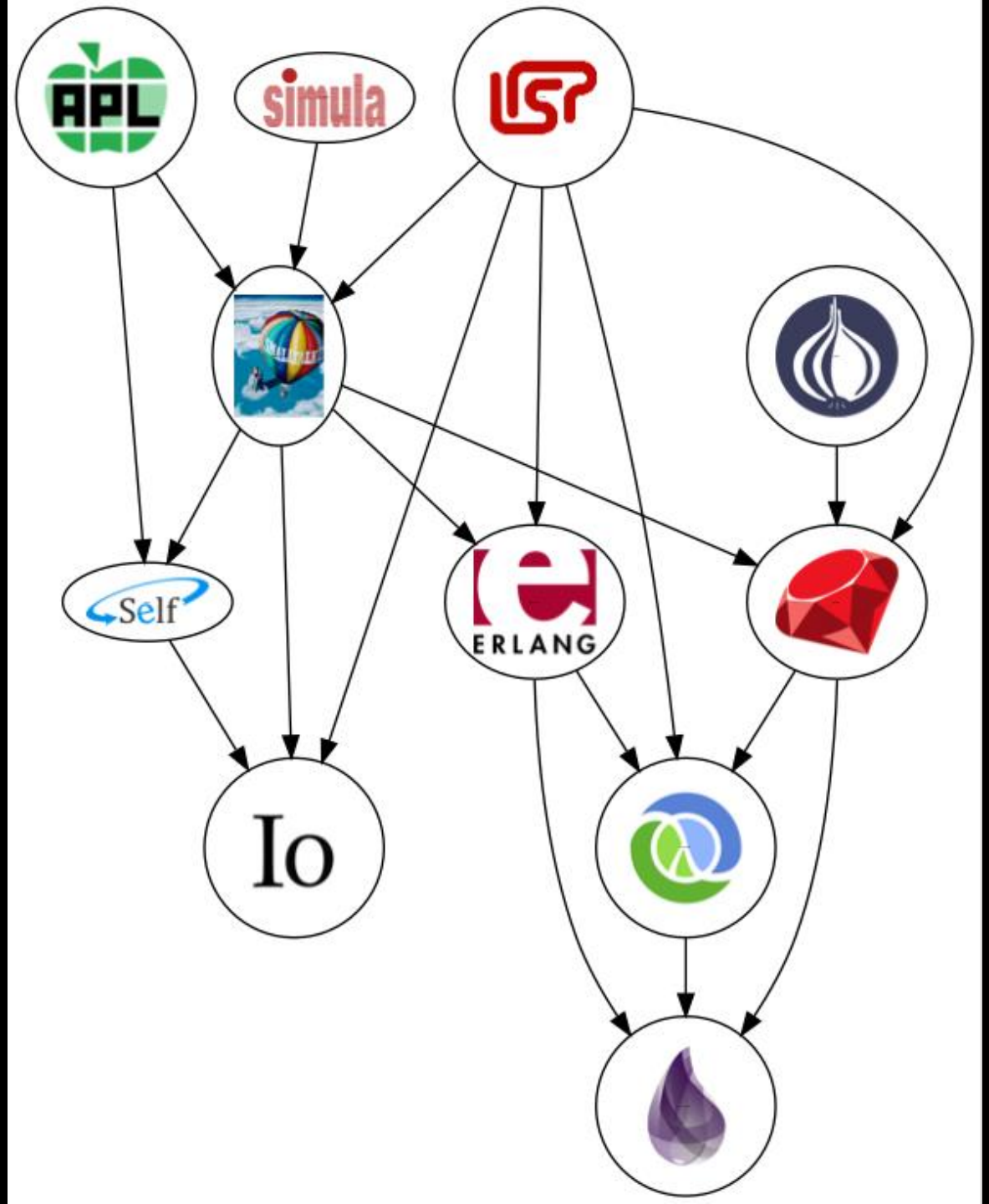

SMALLTALK
APL
Self
Io
SMALLTALK
ERLANG


SMALLTALK
ERLANG
APL
Self
Io

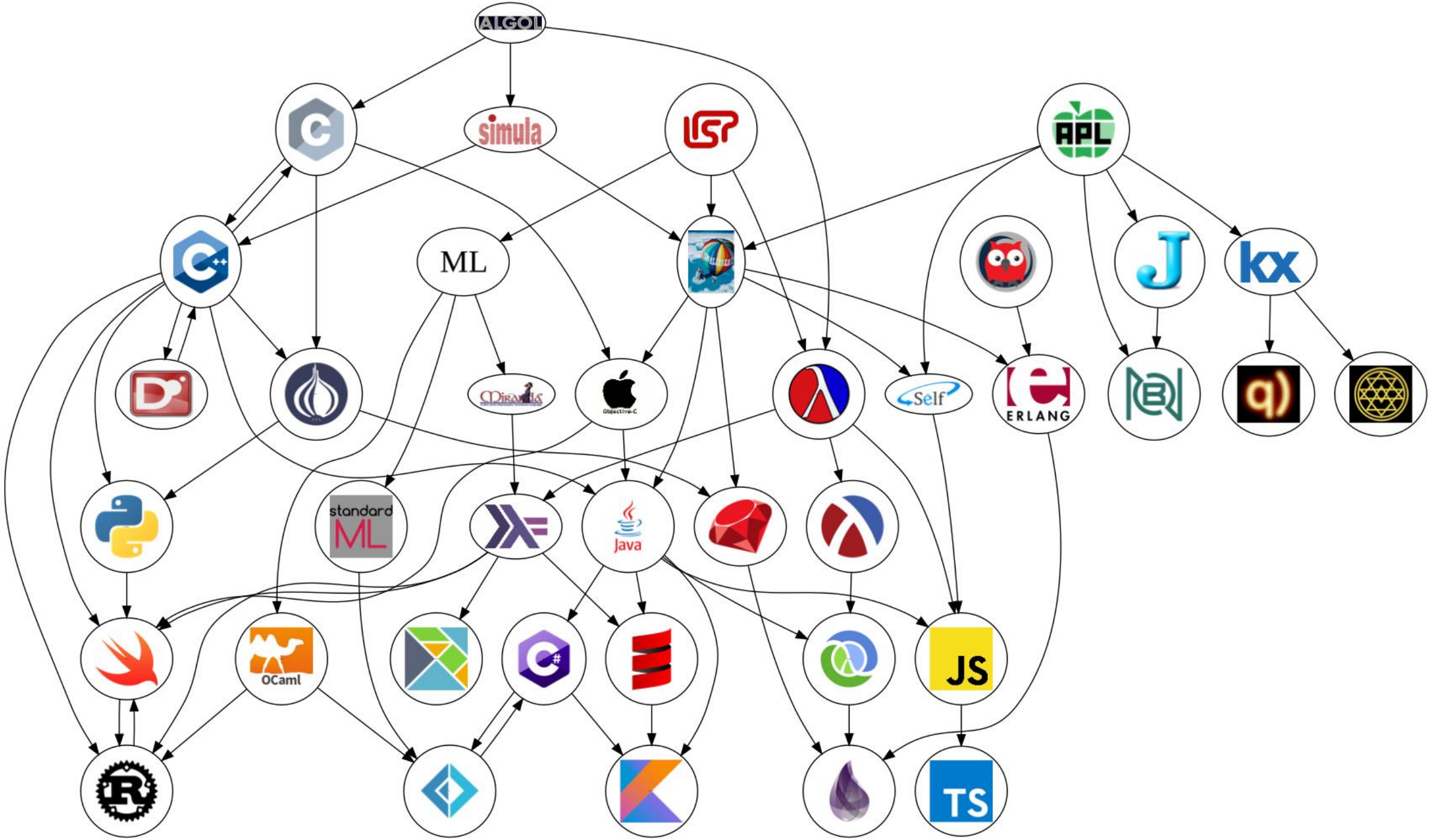

SMALLTALK
ERLANG
APL
Self
Io

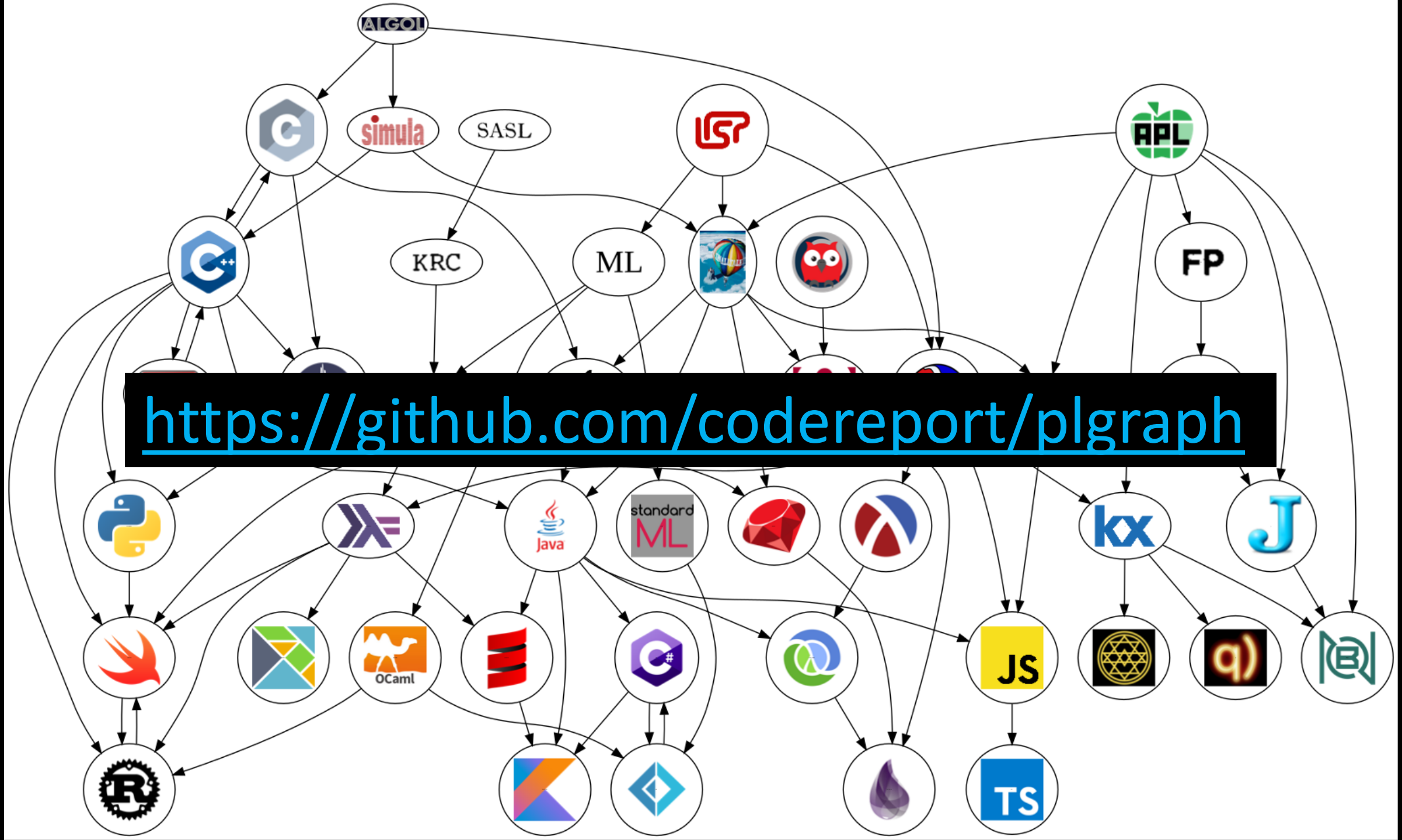


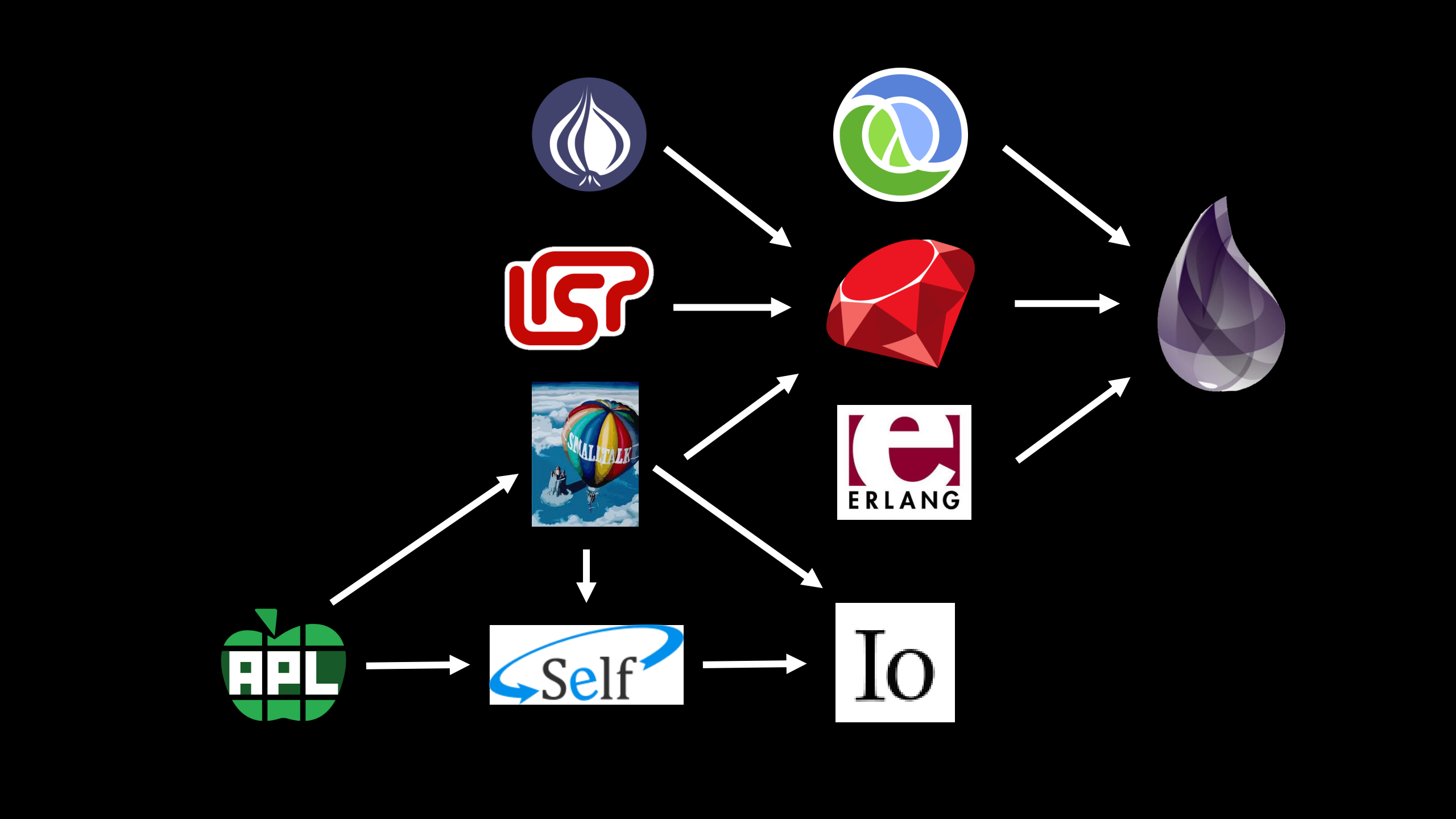

NetworkX
Network Analysis in Python




Graphviz


ALGOL
simula
APL
ML
Self
ERLANG
standard
ML
Java
OCaml
JS
TS


ALGOL
simula
SASL
KRC
ML
APL
FP
https://github.com/codereport/plgraph
Java
standard ML
kx
OCaml
JS
TS


SMALLTALK
ERLANG
APL
Self
Io

The code_report Blog

A blog about programming languages and more

Home Search About

# Galaxy Brain Programming Languages

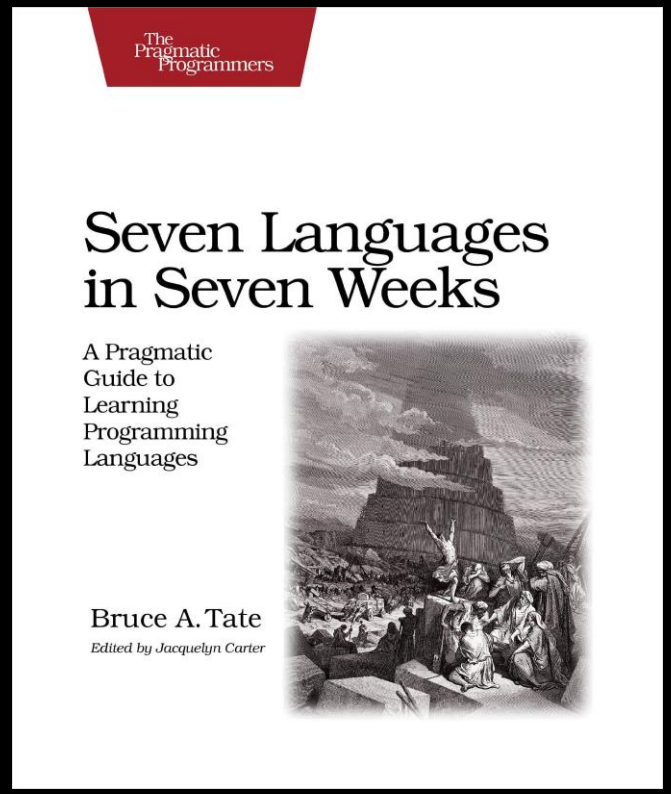
Conor Hoekstra · April 16, 2021

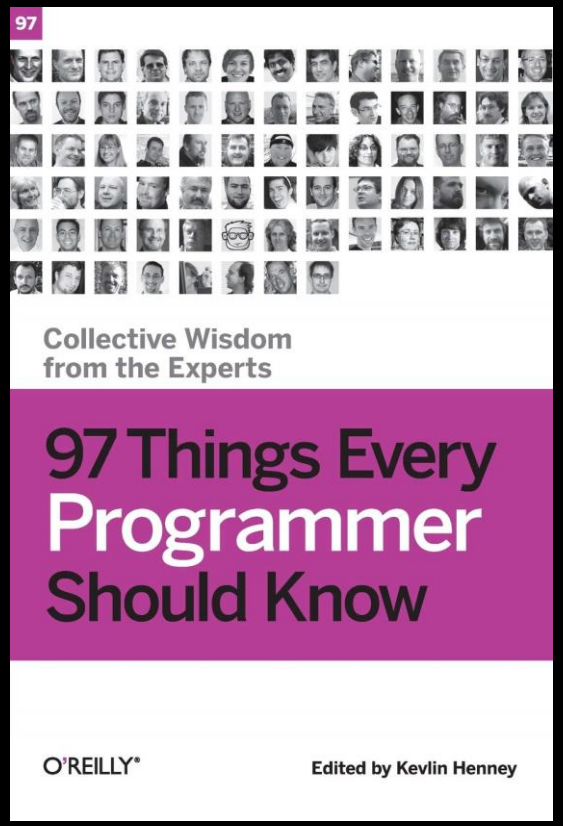
Programming Languages Programming Paradigms

One of my favorite tweets is by Ben Deane:

> **Conor Hoekstra** @code_report
>
> One of my favorite tweets.
>
> > **Ben Deane** @ben_deane
> > Replying to @pati_gallardo
> > Learn computational paradigms.
> >
> > C++, Lisp, Haskell, Smalltalk, Prolog, Forth, APL.
> >
> > If you know these, just about everything else is variations.
>
> 8:37 PM · Feb 17, 2021
>
> 21 Reply Share
>
> Read 3 replies

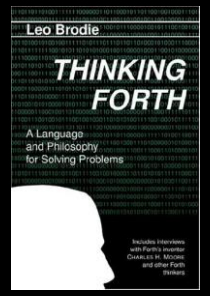
This tweet restates a point that Ben asserts in a blog post of his from 2007 called Six Languages Worth Knowing. The blog lists seven (6 + 1 that was added later) programming languages that he considers worth learning. Combining the tweet and the blog results in









The Pragmatic Programmers

# 7 Languages Every Programmer Should Know


Conor Hoekstra

```
Io> 1 + 2
==> 3
Io> "Hi ho, Io"
==> Hi ho, Io
Io> "Hi ho, Io" print
Hi ho, Io==> Hi ho, Io
Io> Vehicle := Object clone
==>  Vehicle_0x557ad04aea90:
  type                 = "Vehicle"
```

Types in Io are just conveniences. Idiomatically, an object that begins with an uppercase name is a type, so Io sets the type slot. Any clones of that type starting with lowercase letters will simply invoke their parents' type slot. Types are just tools that help Io programmers better organize code.

```
Io> Vehicle description := "Something to take you place"
==> Something to take you place
Io> Vehicle description = "Something to take you far away"
==> Something to take you far away
Io> Vehicle nonexistingSlot = "This won't work"

  Exception: Slot nonexistingSlot not found. Must define slot using := operator before updating.
  ---------
  message 'updateSlot' in 'Command Line' on line 1
```

```
Io> Vehicle description
==> Something to take you far away
Io> Vehicle slotNames
==> list(description, type)
Io> Vehicle type
==> Vehicle
Io> Object type
==> Object
```

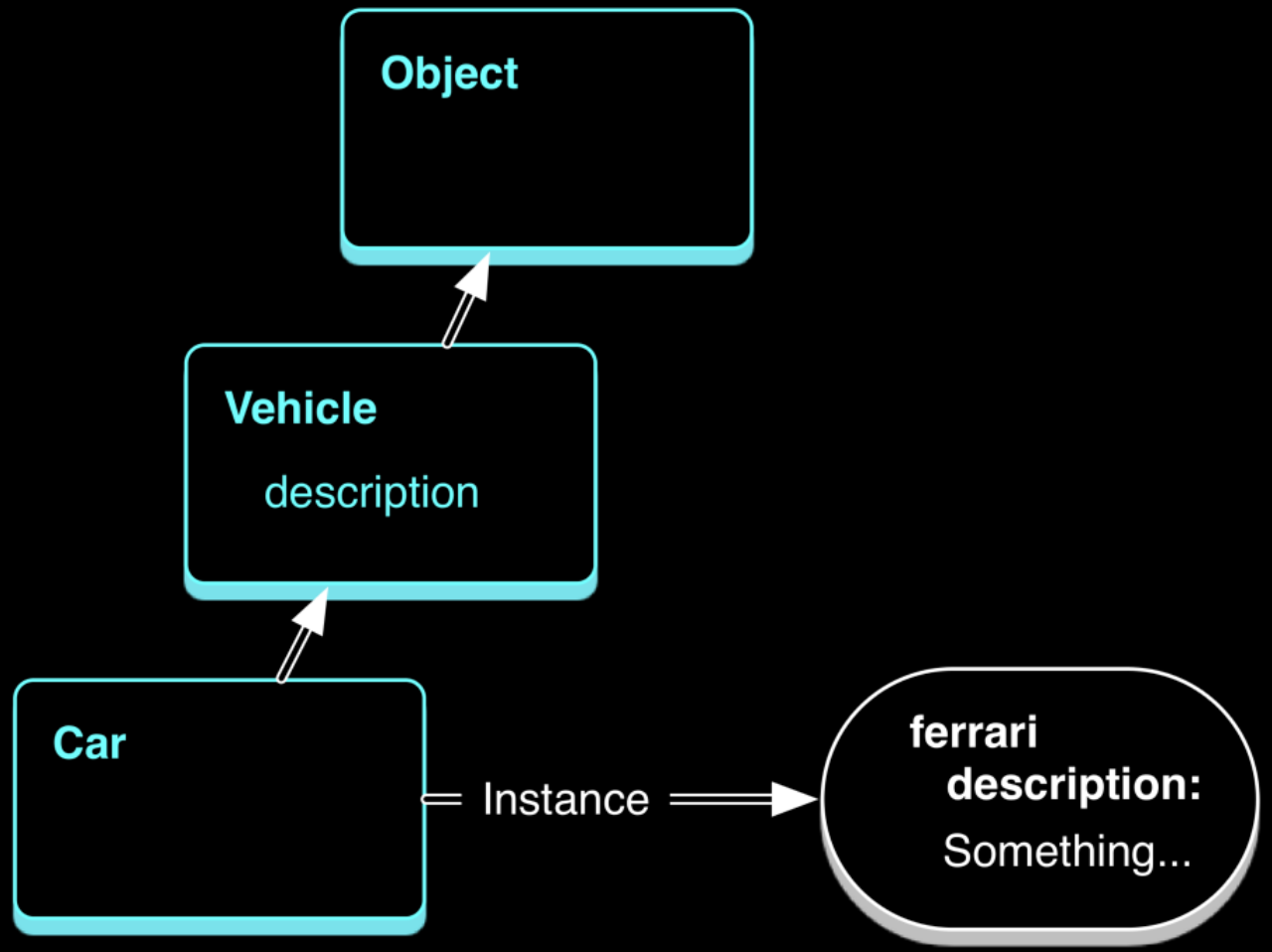



Figure 3.1: An object-oriented design

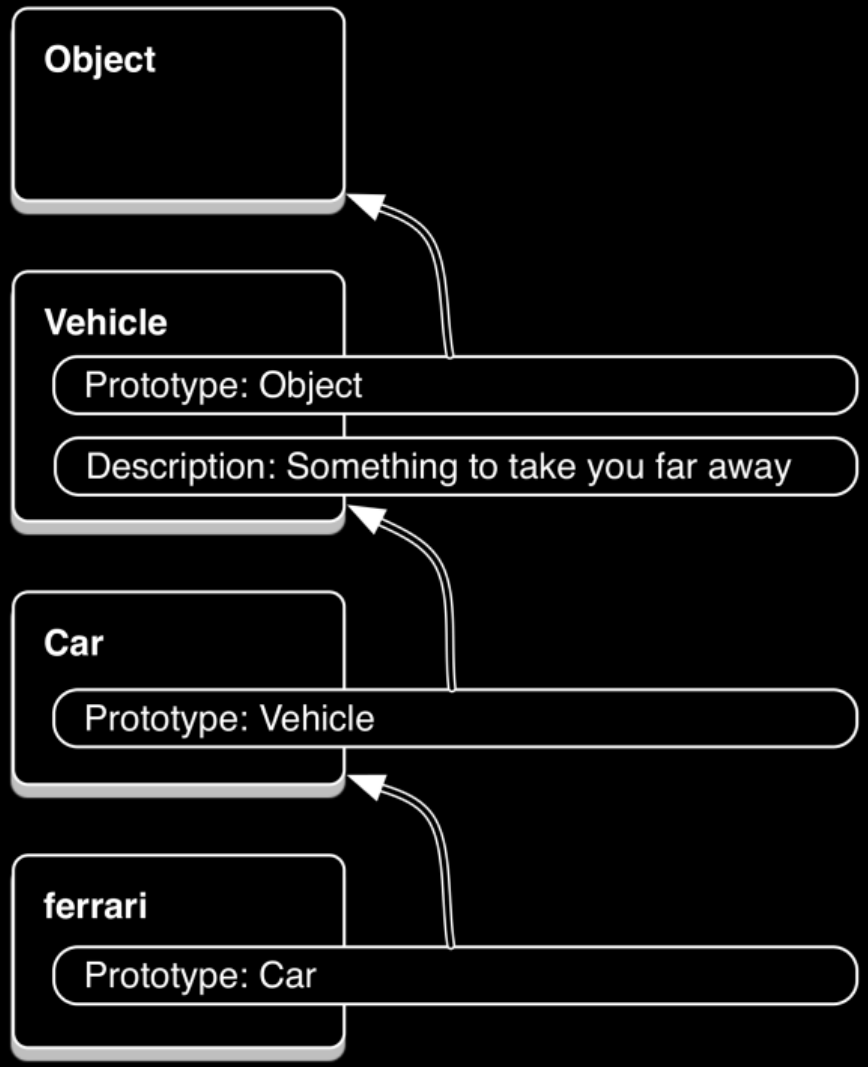



Figure 3.2: Inheritance in Io

In Ruby and Java, classes are templates used to create objects. bruce = Person.new creates a new person object from the Person class. They are different entities entirely, a class and an object. Not so in Io. bruce := Person clone creates a clone called bruce from the prototype called Person. Both bruce and Person are objects. Person is a type because it has a type slot. In most other respects, Person is identical to bruce. Let's move on to behavior.

```
Io> ferrari proto
==>  Car_0x557ad04f0860:
  drive               = method(...)
  type                = "Car"

Io> Car proto
==>  Vehicle_0x557ad04aea90:
  description         = "Something to take you far away"
  type                = "Vehicle"
```

The prototype programming paradigm seems clear enough. These are the basic ground rules:

- Every *thing* is an object.
- Every *interaction* with an object is a message.
- You don't instantiate classes; you clone other objects called *prototypes*.
- Objects remember their prototypes.
- Objects have slots.
- Slots contain objects, including method objects.
- A message returns the value in a slot or invokes the method in a slot.
- If an object can't respond to a message, it sends that message to its prototype.

Do:

- Run an Io program from a file.
- Execute the code in a slot given its name.

```
// 1. Run an Io program from a file.

// io conor_hoekstra_solutions.io

// Conor says
// BQN is my favorite language!

// 2. Execute the code in a slot given its name

Person := Object clone
conor := Person clone
conor speak := method("BQN is my favorite language!" println)
"Conor says" println
conor speak
```

```
Io> OperatorTable
==> OperatorTable_0x55d6fd684c40:
Operators
  0   ? @ @@
  1   **
  2   % * /
  3   + -
  4   << >>
  5   < <= > >=
  6   != ==
  7   &
  8   ^
  9   |
  10  && and
  11  or ||
  12  ..
  13  %= &= *= += -= /= <<= >>= ^= |=
  14  return
```

```
Assign Operators
  ::= newSlot
  :=  setSlot
  =   updateSlot
```

Do:

1. A Fibonacci sequence starts with two 1s. Each subsequent number is the sum of the two numbers that came before: 1, 1, 2, 3, 5, 8, 13, 21, and so on. Write a program to find the nth Fibonacci number. fib(1) is 1, and fib(4) is 3. As a bonus, solve the problem with recursion and with loops.
2. How would you change / to return 0 if the denominator is zero?
3. Write a program to add up all of the numbers in a two-dimensional array.
4. Add a slot called myAverage to a list that computes the average of all the numbers in a list. What happens if there are no numbers in a list? (Bonus: Raise an Io exception if any item in the list is not a number.)
5. Write a prototype for a two-dimensional list. The dim(x, y) method should allocate a list of y lists that are x elements long. set(x, y, value) should set a value, and get(x, y) should return that value.
6. Bonus: Write a transpose method so that (new_matrix get(y, x)) == matrix get(x, y) on the original list.
7. Write the matrix to a file, and read a matrix from a file.
8. Write a program that gives you ten tries to guess a random number from 1–100. If you would like, give a hint of “hotter” or “colder” after the first guess.

```
// 1. A Fibonacci sequence starts with two 1s. Each subsequent number is the

Number fib ::= method(if (self <= 2, 1, (self - 1) fib + (self - 2) fib))

// Test
"First 10 fibonacci numbers:" println
for (i, 1, 10, i fib println)
```

```
// 3. Write a program to add up all of the numbers in a two-dimensionalarray.

List sum2DArray := method(self flatten sum)
```

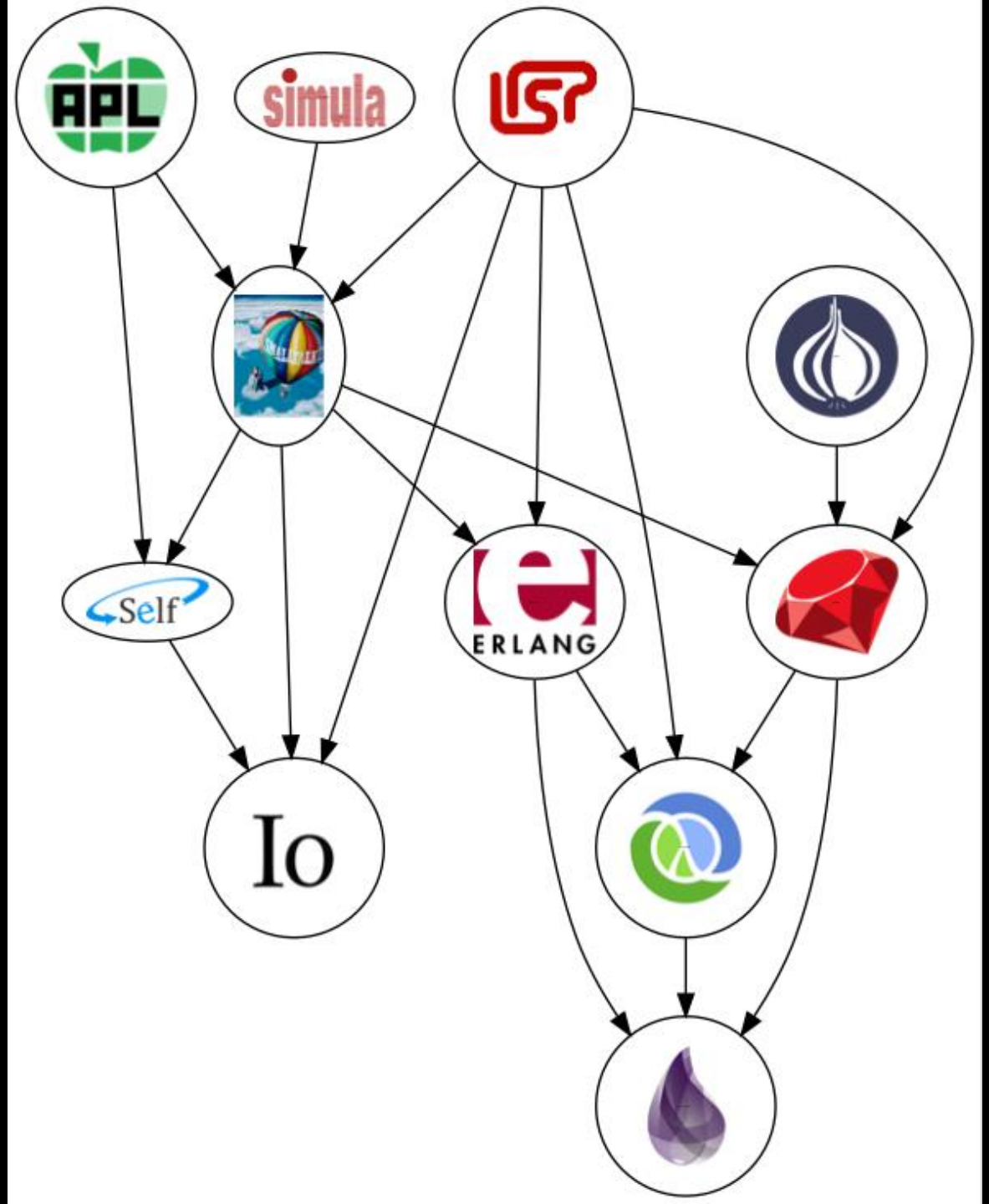
```
Game := Object clone
Game play := method(
    targetValue := 100 fakeRandom
    guessNo := 1
    lastGuess := 0
    found := false
    while (guessNo <= 10 and found not,
        "Guess a number between 1 and 100: " print
        guess := File standardInput readLine asNumber
        if (guess == targetValue,
            found = true; "You found it!" println,
            "Nope, not it! " println
            if (lastGuess == 0,
                lastGuess = guess,
                lastDiff := (targetValue - lastGuess) abs
                currDiff := (targetValue - guess) abs
                if (currDiff < lastDiff, "But you are warmer.", "And you are colder.") println
                lastGuess = guess
            )
        )
        guessNo = guessNo + 1
    )
)
```

## Coroutines

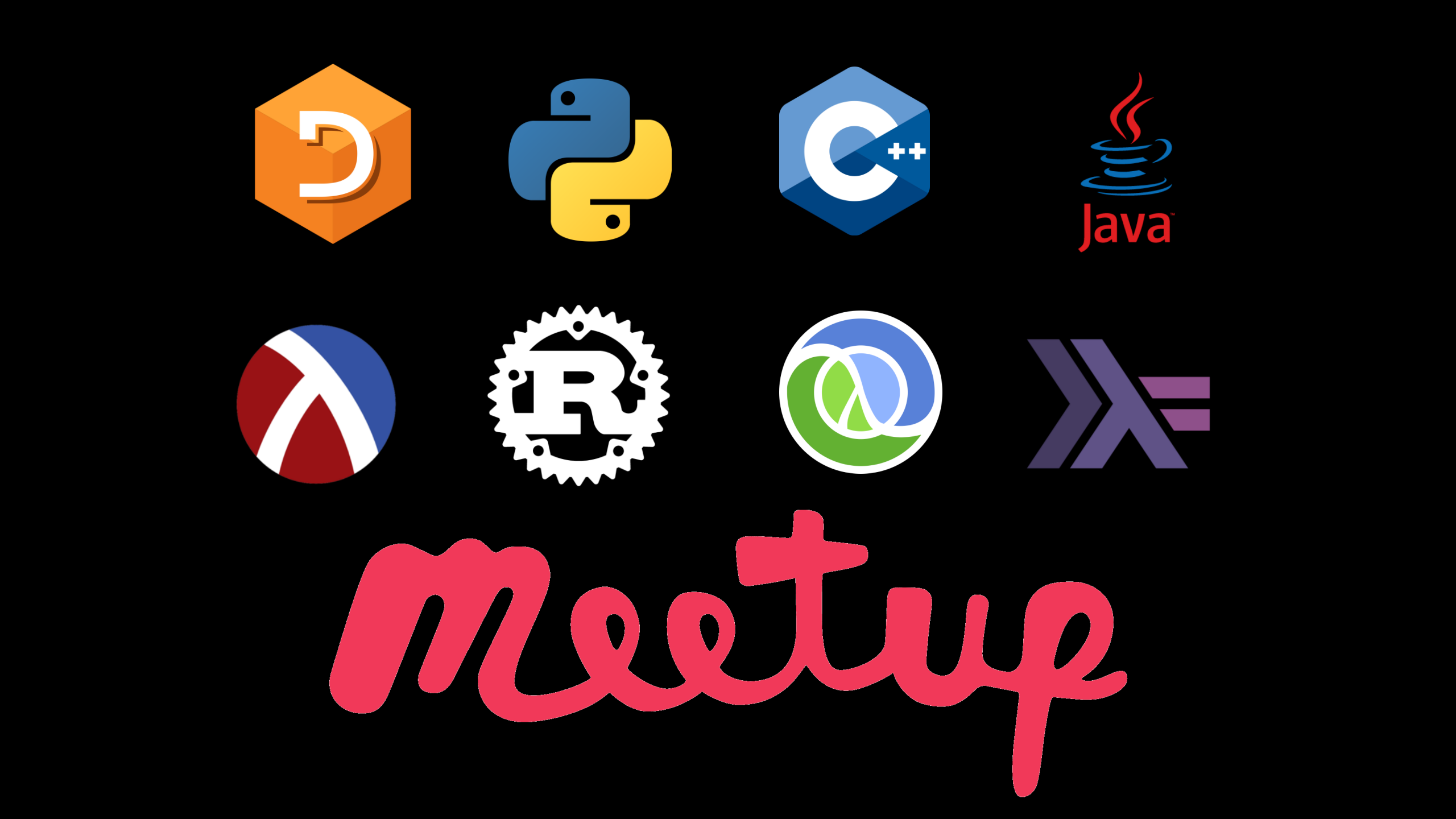
The foundation for concurrency is the coroutine. A coroutine provides a way to voluntarily suspend and resume execution of a process. Think of a coroutine as a function with multiple entry and exit points. Each yield will voluntarily suspend the process and transfer to another process. You can fire a message asynchronously by using @ or @@ before a message. The former returns a future (more later), and the second returns nil and starts the message in its own thread. For example, consider this program:

# @ or @@

```
psum := method(
    tempValues := List clone setSize(self rows)
    for (i, 0, self rows - 1,
        tempValues atPut(i, self at(i) @sum)
    )
    tempValues sum
)
```




Graphviz


Java
meetup