Elixir

Presented by @Dawei Ma


elixir

My programming language learning path

- C/C++
- Java/Scala (SpringBoot/Play2/Vert.x/RxJava)
- Swift/Objective-C (RxSwift)
- Python (Django/Flask/Tornado)
- Elixir/Erlang (Phoenix)
- Golang
- Javascript (Vue/Angular)
- PHP (Laravel)
- CommonLisp (Learning)
- Rust (Will Learn)

Rules for learning a language

- Must solve a problem
- Problem not addressed by current stack
- “Fast” is not a purpose
- Should be the best solution


C
If you're not allocating memory, you're not living!
C++
If you're not allocating memory, you're not living!
ASSEMBLY
GO
Google
RUBY
(& her companion Rails)
ERLANG
DESTROY ONE OF MY PROCESSES & I WILL ONLY GROW STRONGER
PYTHON
My formatting is my syntax!
PHP
Can I GET you guys anything?
R
FCI
JAVA
JAVASCRIPT
I'm technically functional.
HASKELL
(λx.x am pure)(I)

Perl

Rust

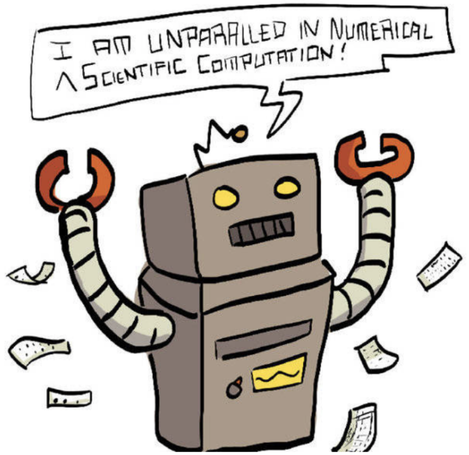
Lua

Lisp Dialects

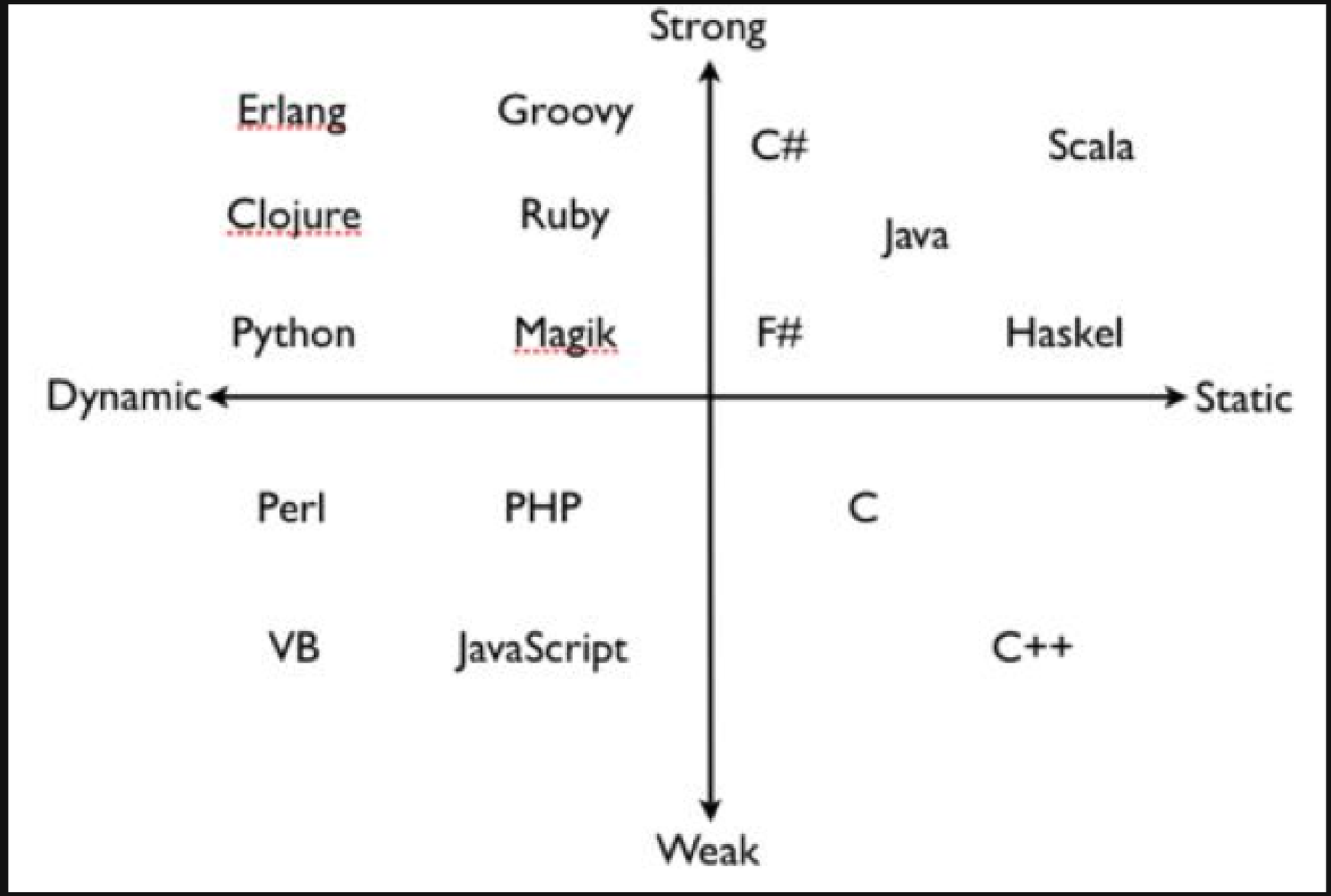
MATLAB

Elixir

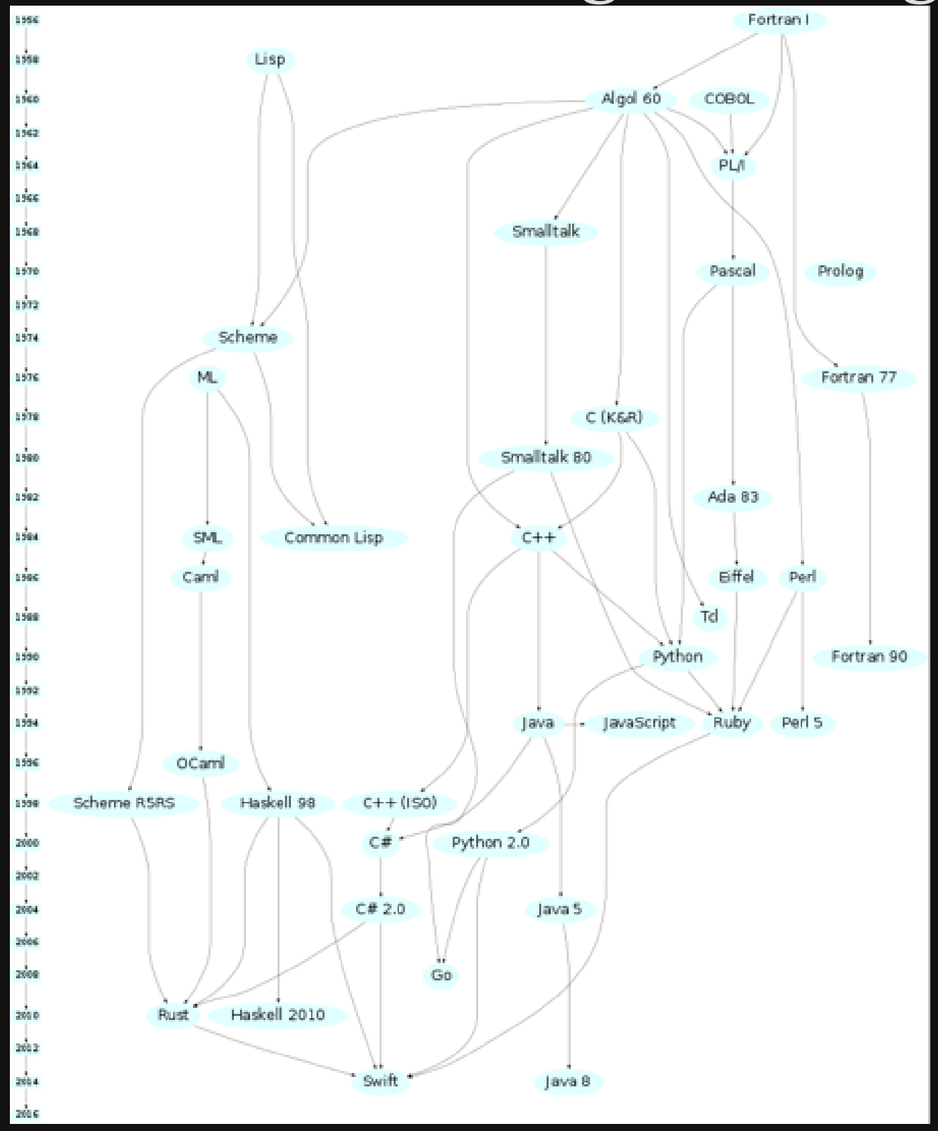
Fortran



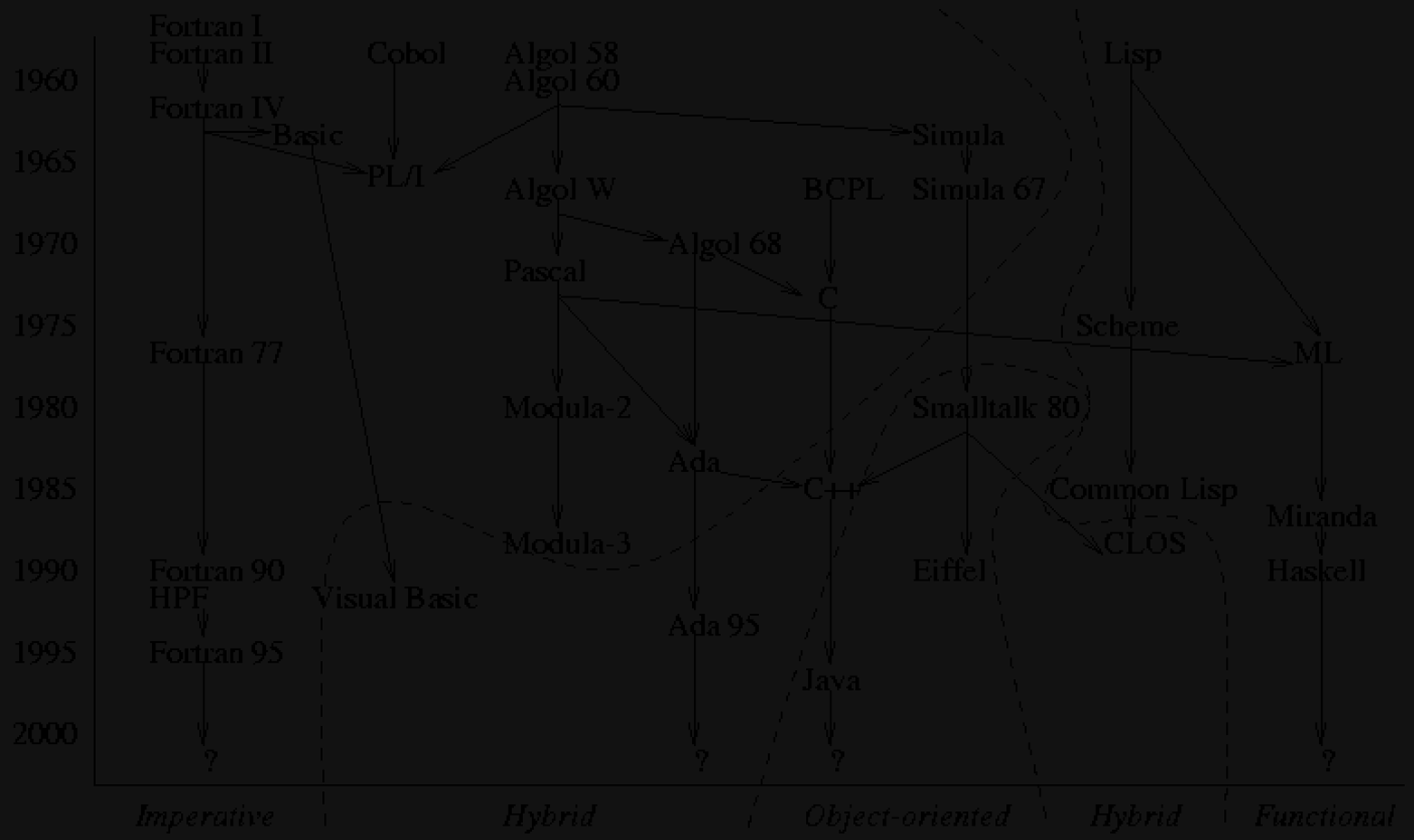
Type System


Strong
Erlang
Groovy
C#
Scala
Clojure
Ruby
Java
Python
Magik
F#
Haskel
Dynamic
Static
Perl
PHP
C
VB
JavaScript
C++
Weak

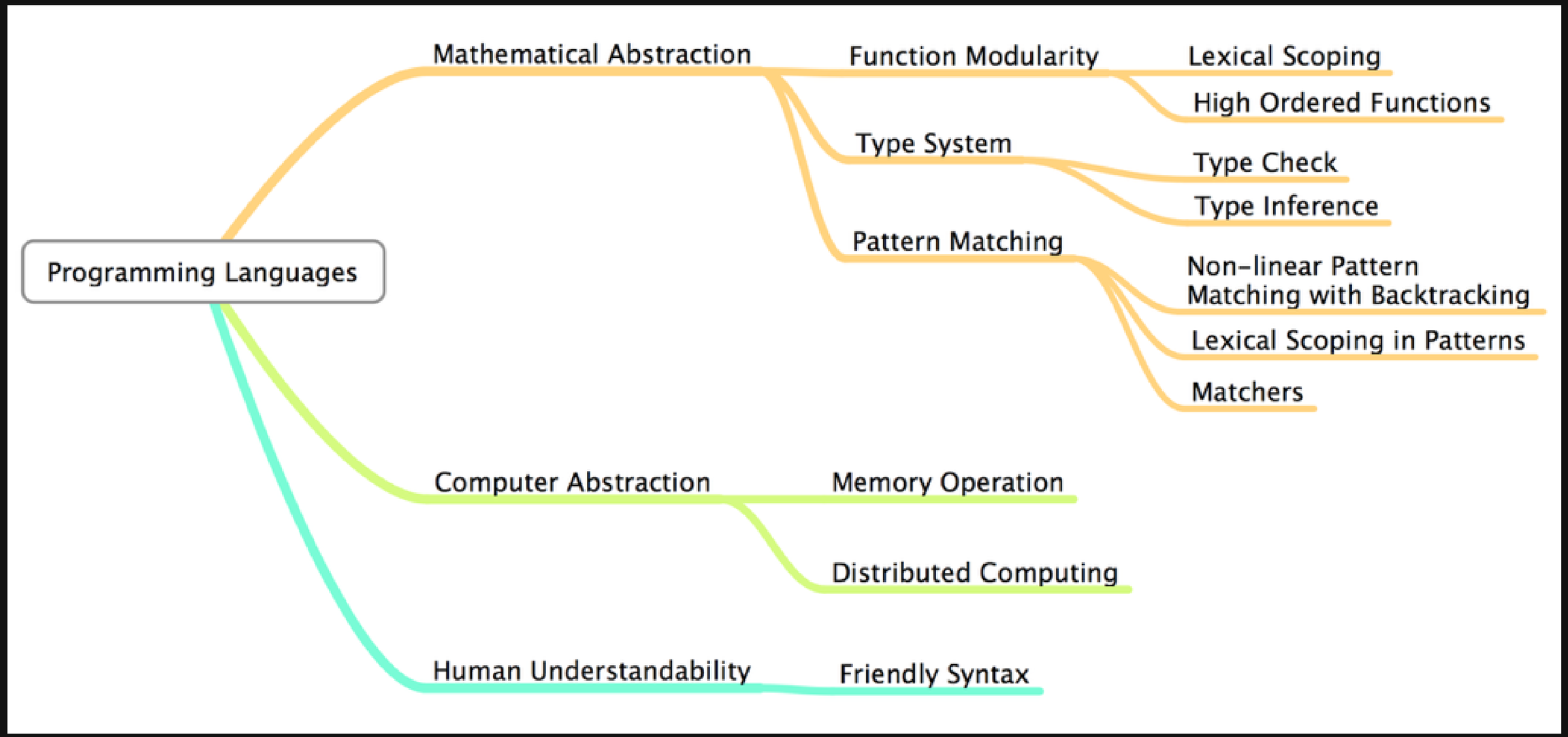
Evolution of Programming Languages

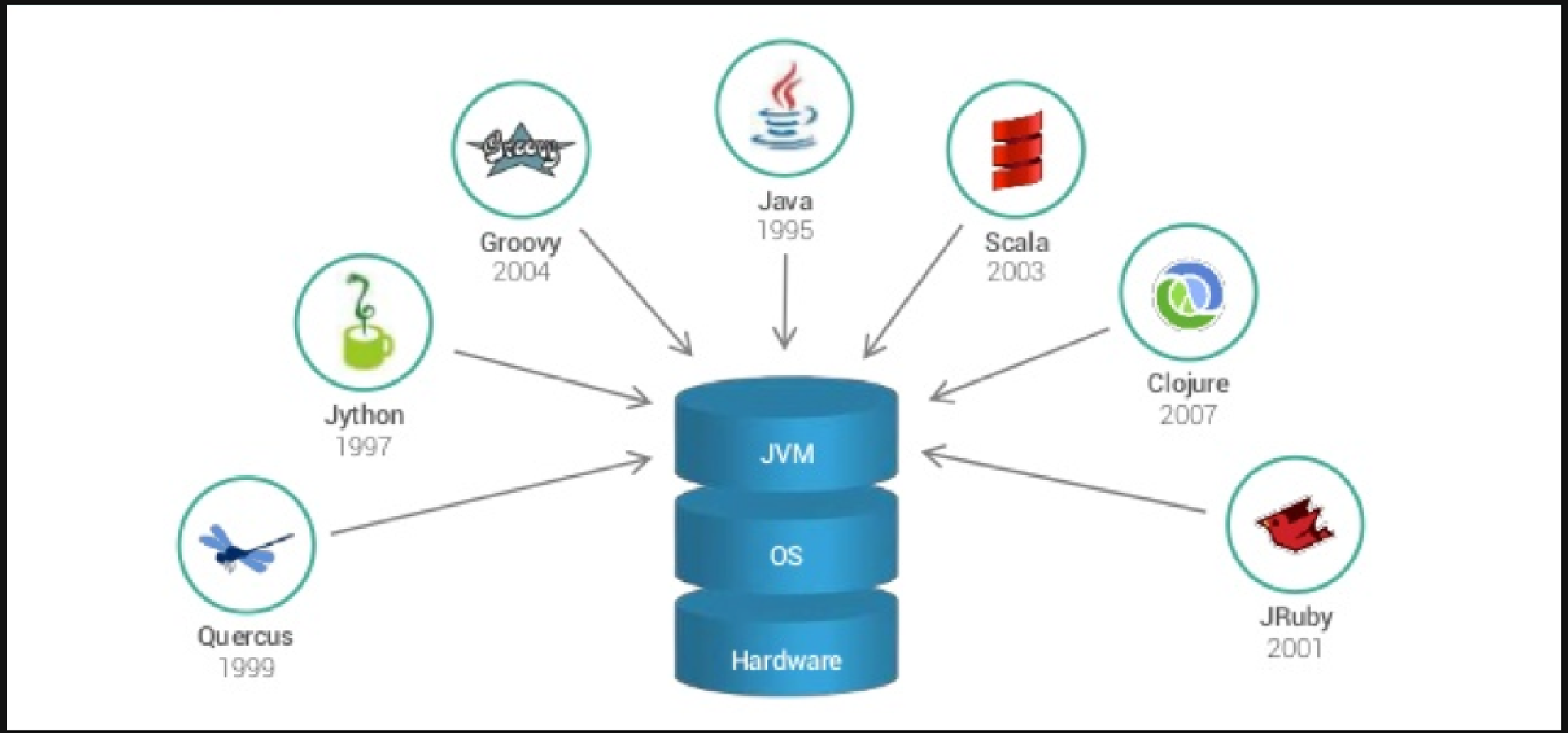


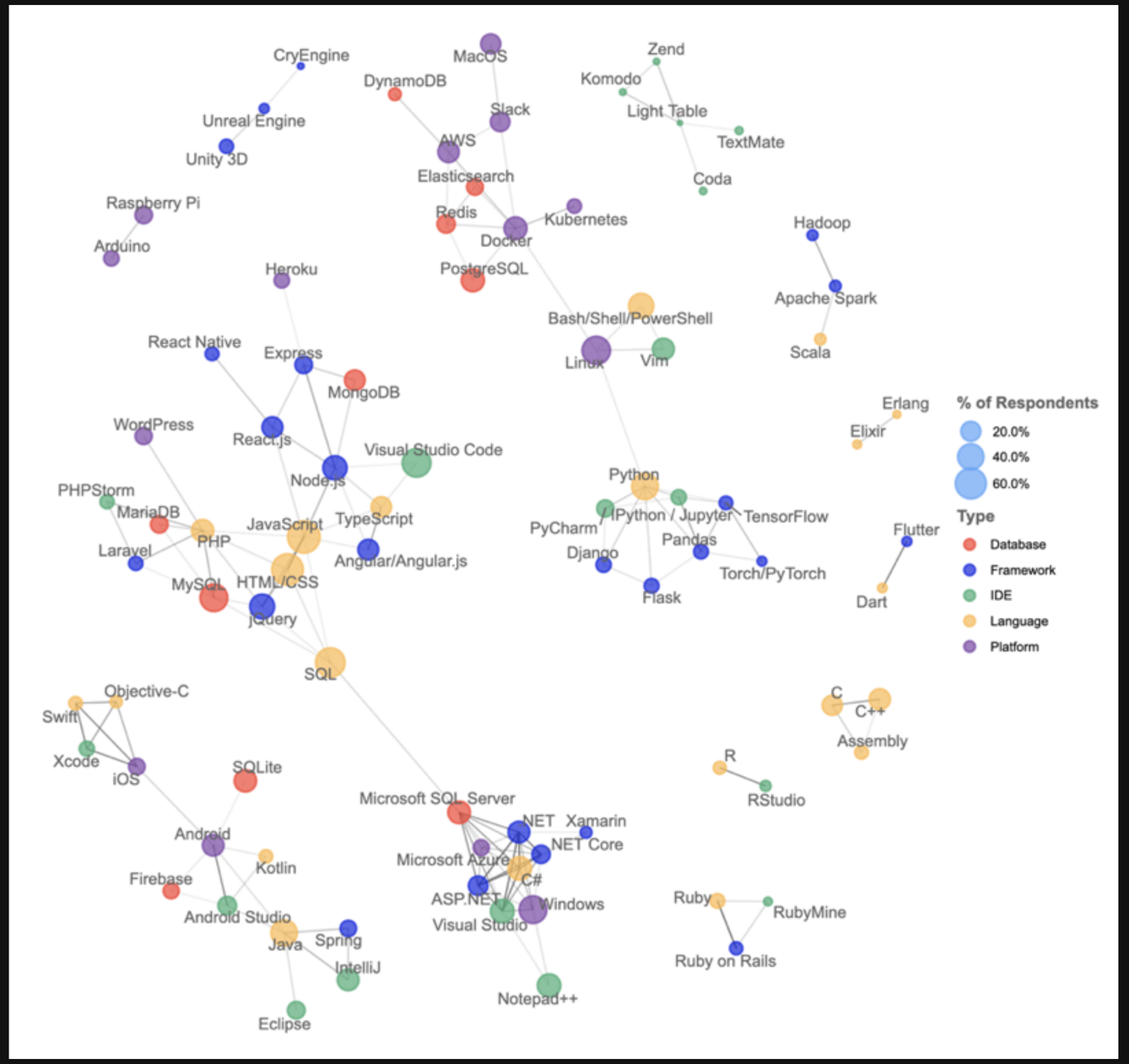
Classify


Fortran I
Fortran II
Cobol
Algol 58
Algol 60
Lisp
1960
Fortran IV
Basic
Simula
1965
PL/I
Algol W
BCPL
Simula 67
1970
Algol 68
Pascal
C
1975
Scheme
Fortran 77
ML
1980
Modula-2
Smalltalk 80
Ada
1985
C++
Common Lisp
Miranda
Modula-3
CLOS
1990
Fortran 90
Eiffel
Haskell
HPF
Visual Basic
Ada 95
1995
Fortran 95
Java
2000
?
?
?
?
Imperative
Hybrid
Object-oriented
Hybrid
Functional

Evaluate programming languages




Groovy
2004
Java
1995
Scala
2003
Jython
1997
Clojure
2007
JVM
OS
Hardware
Quercus
1999
JRuby
2001

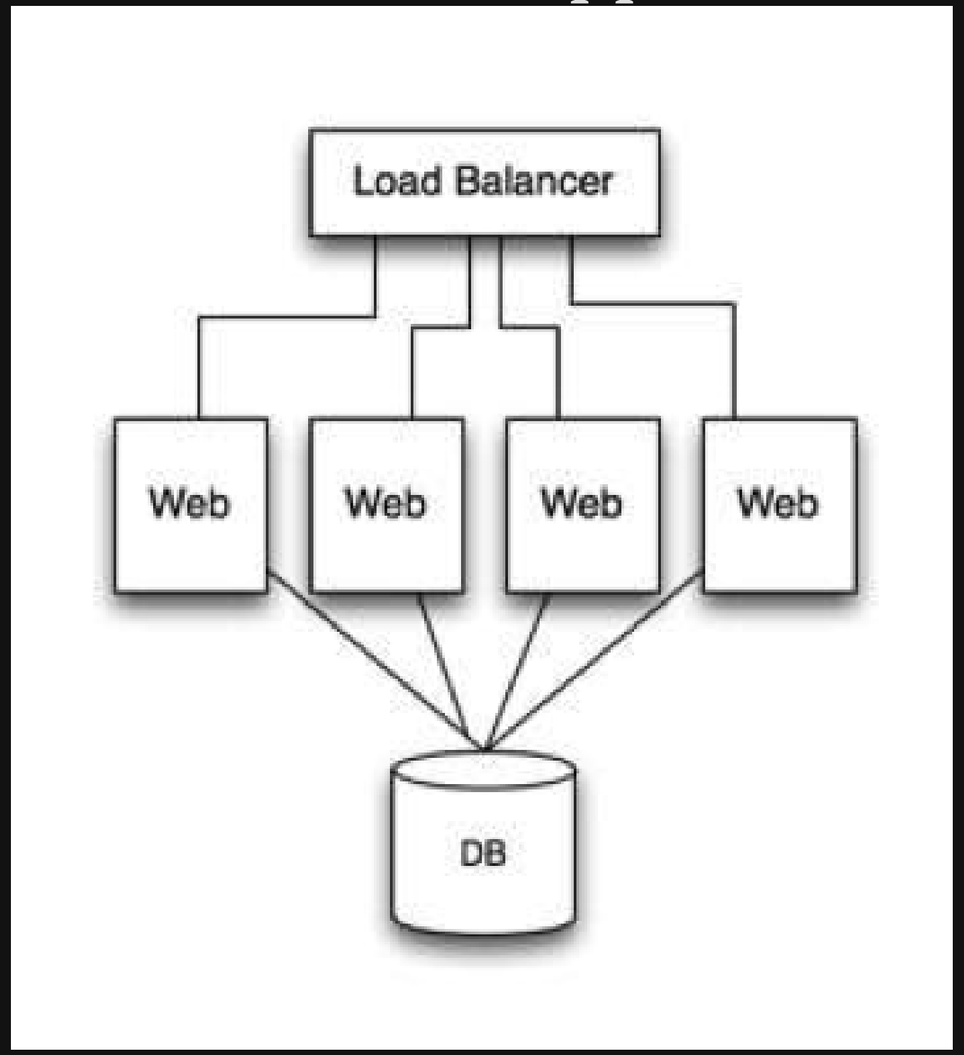

CryEngine
MacOS
Zend
DynamoDB
Komodo
Unreal Engine
Slack
Light Table
AWS
TextMate
Unity 3D
Elasticsearch
Coda
Raspberry Pi
Redis
Kubernetes
Hadoop
Docker
Arduino
PostgreSQL
Heroku
Apache Spark
Bash/Shell/PowerShell
React Native
Express
Linux
Vim
Scala
MongoDB
Erlang
WordPress
Elixir
React.js
Visual Studio Code
% of Respondents
Node.js
20.0%
Python
PHPStorm
40.0%
MariaDB
TypeScript
60.0%
PyCharm
IPython / Jupyter
TensorFlow
JavaScript
Type
Pandas
Flutter
PHP
Laravel
Django
Database
Angular/Angular.js
Torch/PyTorch
Framework
MySQL
HTML/CSS
Flask
Dart
IDE
jQuery
Language
SQL
Platform
Objective-C
C
Swift
C++
Assembly
Xcode
R
SQLite
iOS
RStudio
Microsoft SQL Server
NET
Xamarin
Android
NET Core
Microsoft Azure
Kotlin
C#
Firebase
ASP.NET
Windows
Ruby
RubyMine
Android Studio
Visual Studio
Java
Spring
Ruby on Rails
IntelliJ
Notepad++
Eclipse

WHAT'S THE BIG DEAL WITH ELIXIR?

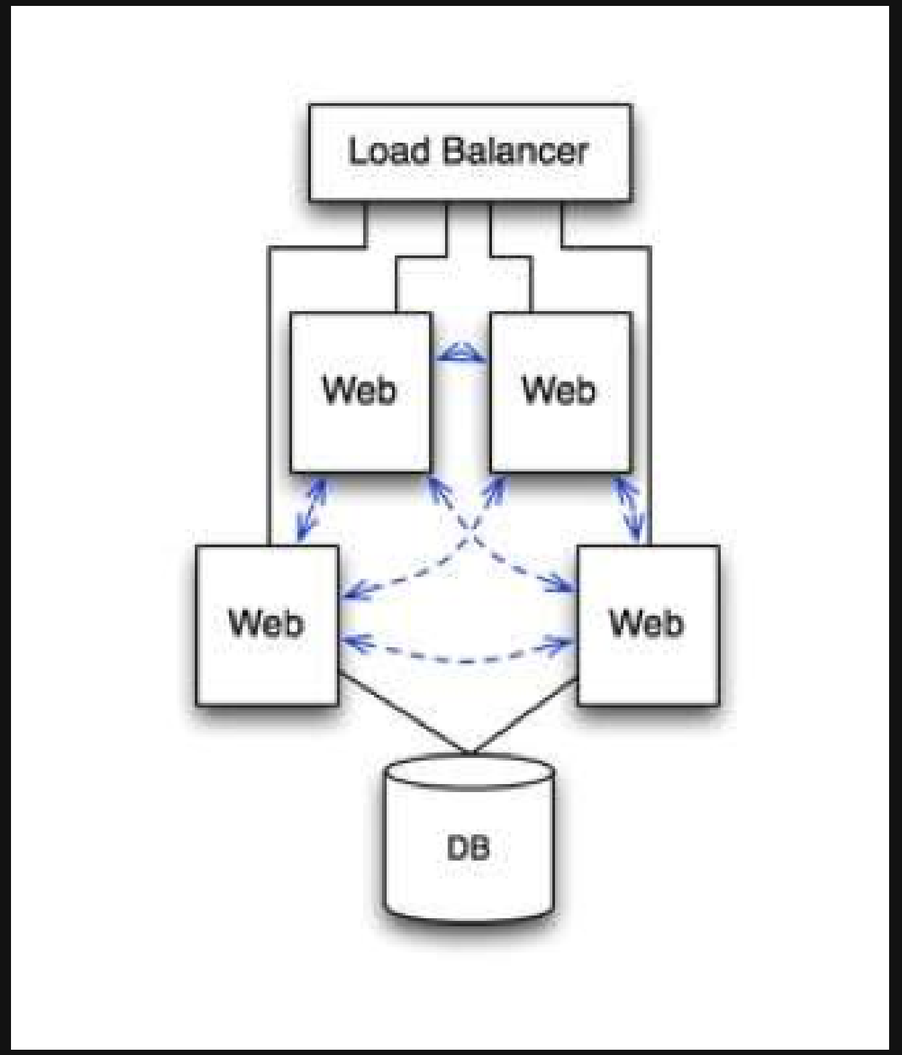
- Ruby-like focus on developer productivity
- Embedded database
- Compiles down to code to run on the BEAM Virtual Machine
- BEAM/OTP is what Erlang runs on
- Erlang/BEAM is the best existing language for concurrency, consistency and fault tolerance, hot code swapping
- Erlang does not focus on developer productivity
- Problem in Ruby is concurrency model

So what does that mean?

Standard Web App



OTP



What’s the big deal?

- Facebook paid $22 billion for WhatsApp
- WhatsApp had $10 million in revenue
- What was the big deal?
  - Erlang/OTP
  - 2 million users / server
  - No central relay point
  - Scales horizontally
  - Deploys w/o disconnect

Other languages

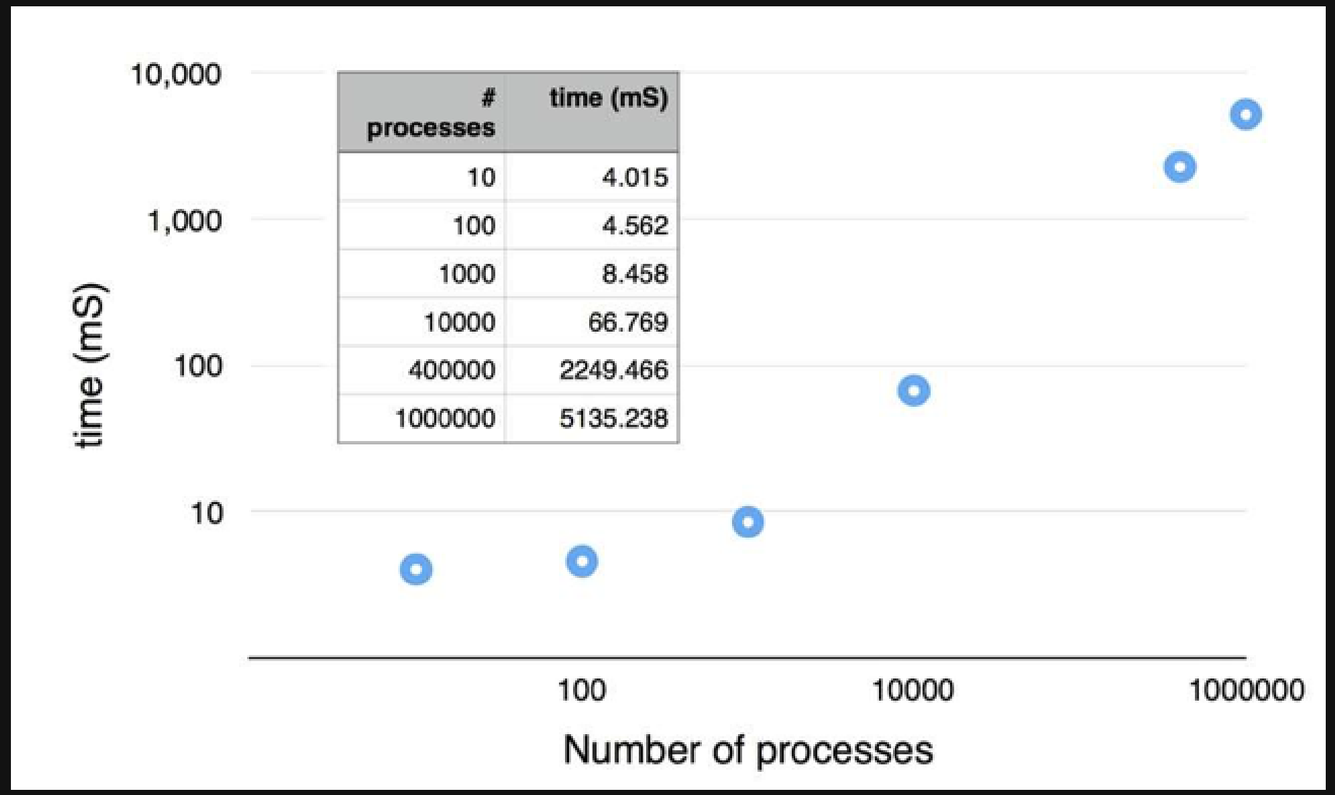
- Boot up
- Memory is shared
  - Where leaks come from
  - Changing shared memory requires a mutex lock
- Garbage collector periodically runs
  - Pause entire stack
- Requests run in threads in the same process
  - Threads are cooperatively scheduled
  - Deployment means shutting down current code, starting new code

Erlang/Elixir/OTP

- No memory is shared
- Data structures are immutable
- Each Erlang process (basically a light thread) has its own HEAP
  - Reclaimed on completion
- Code can be hot deployed
  - New code runs next time it’s accessed (existing code keeps running)
- Processes are prescheduled

Sound familiar?

- Difference is size of the allocations
  - An Erlang process is 0.5 kb
  - A Go goroutine is 2 kb (version 1.4)
  - A Java thread is 1024 kb on 64 bit VM
  - PHP request varies by how much is loaded
  - Laravel averages 7-12mb / request



# processes	time (mS)
10	4.015
100	4.562
1000	8.458
10000	66.769
400000	2249.466
1000000	5135.238

Programming Elixir, Chapter 15 Laptop w/ 4 cores and 4gb of RAM counting concurrently 1,000,000 processes =

- 0.48 gb in Elixir
- 1.91 gb in Golang (go routines)
- 977 gb in Java (threads)

Immutable Data

- There’s no passing pointers
- Add something to a list, get a new list
- Everything is “message passing”
  - Avoids mutex locks
  - Enables per-process garbage collection
  - Makes calling a function locally, in another process or on another machine transparent

3 Databases Built In

- ETS – Erlang Term Storage
  - In memory table storage for a node
- DETS – Disk-based Erlang Term Storage
  - Disk table storage for a node

3 Databases Built In

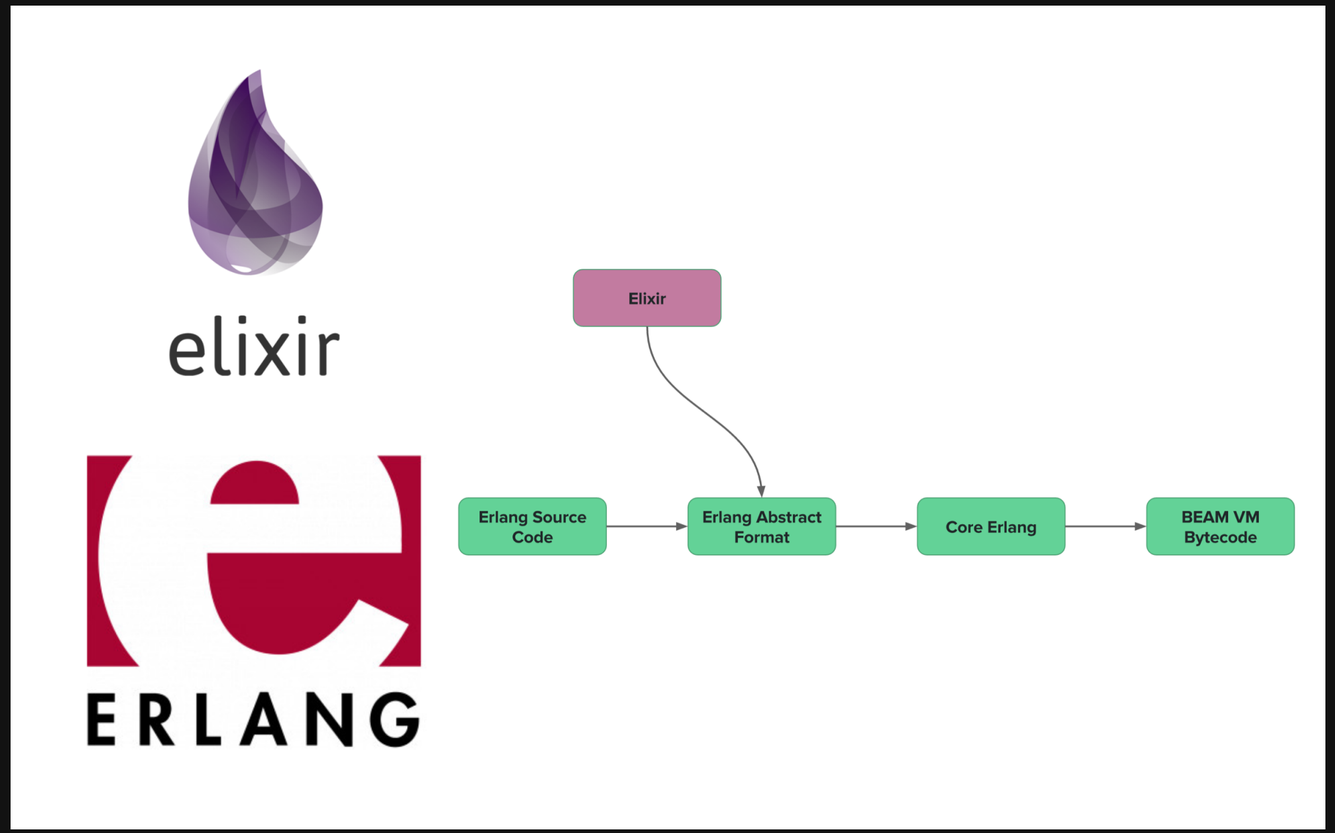
- Mnesia - #awesome
  - A relational/object hybrid data model that is suitable for telecommunications applications.
  - A DBMS query language, Query List Comprehension (QLC) as an add-on library.
  - Persistence. Tables can be coherently kept on disc and in the main memory.
  - Replication. Tables can be replicated at several nodes.
  - Atomic transactions. A series of table manipulation operations can be grouped into a single atomic transaction.
  - Location transparency. Programs can be written without knowledge of the actual data location.
  - Extremely fast real-time data searches.
  - Schema manipulation routines. The DBMS can be reconfigured at runtime without stopping the system.

Loops?

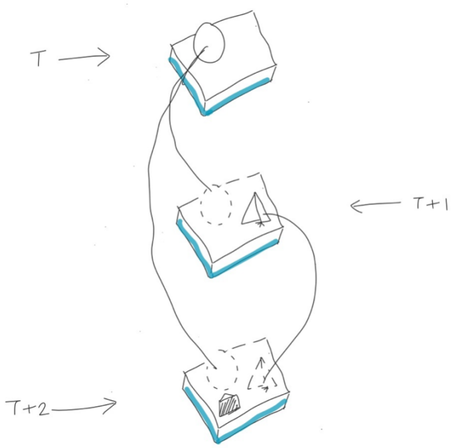
How do you have a for loop with an immutable increment?

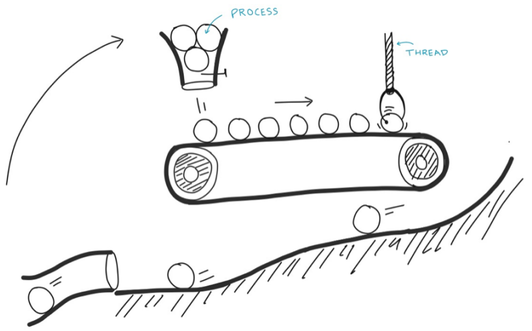
Recursion. Lots of recursion.

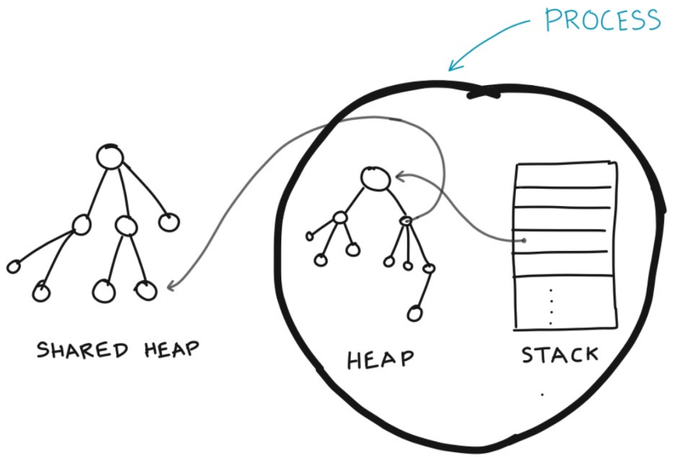
> “If Java is the right one to run anywhere, then Erlang is the right one to run forever.”
>
> - Joe Armstrong


elixir
ERLANG
Elixir
Erlang Source Code
Erlang Abstract Format
Core Erlang
BEAM VM Bytecode

elixir on beam


T
T+1
T+2


PROCESS
THREAD

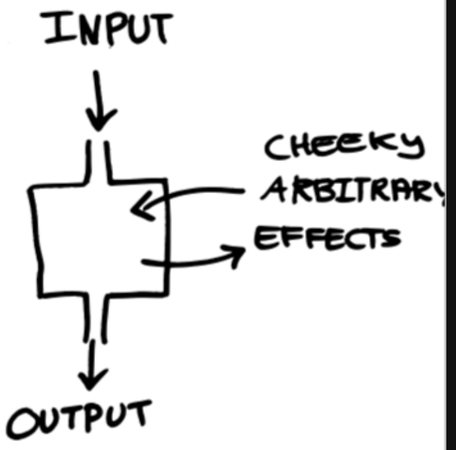

PROCESS
SHARED HEAP
HEAP
STACK

why functional?

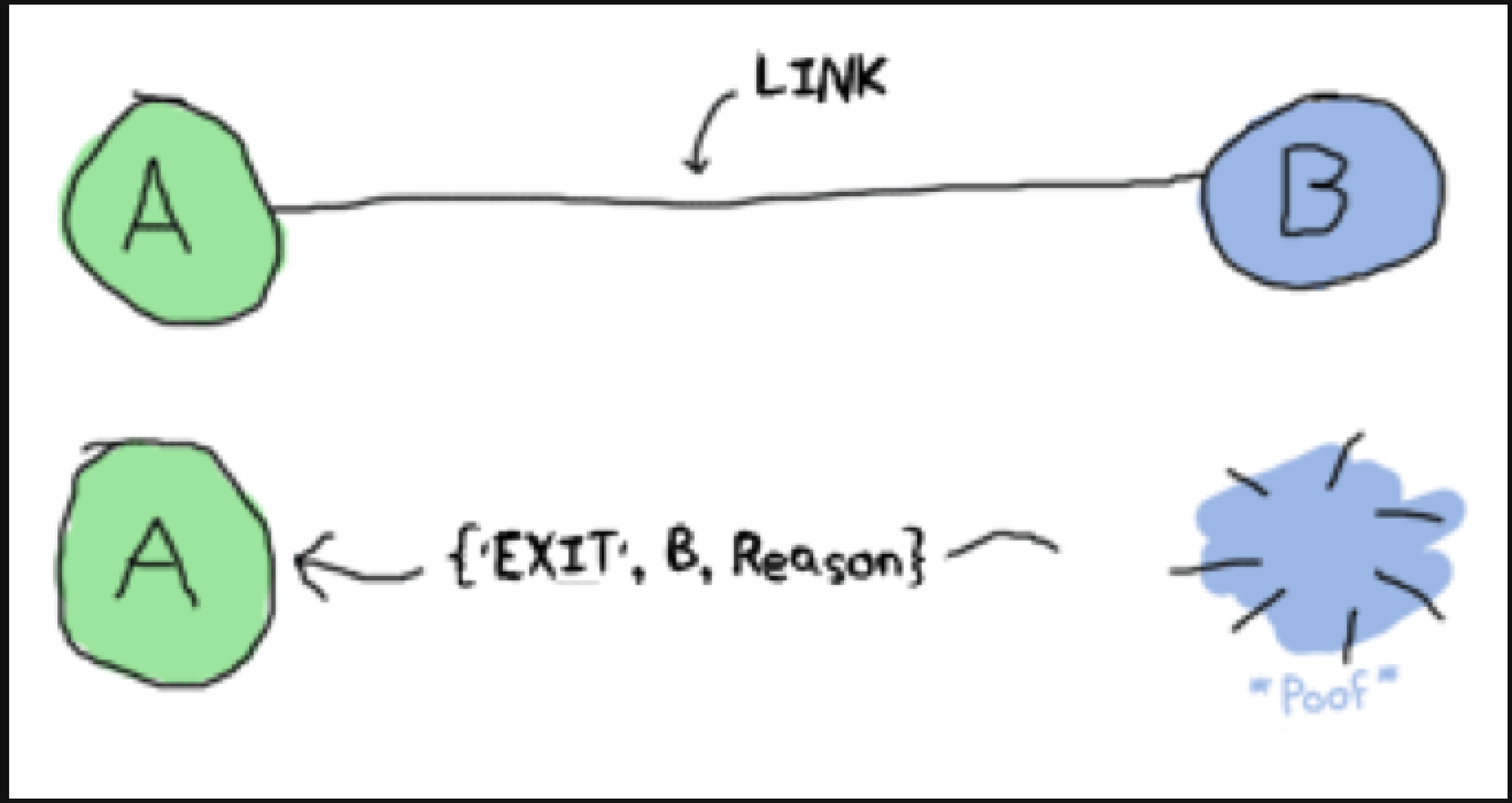
- Predictable
- Easier to test
- Immutable data
- Easier concurrency
- Simpler (yes, really)

Functions

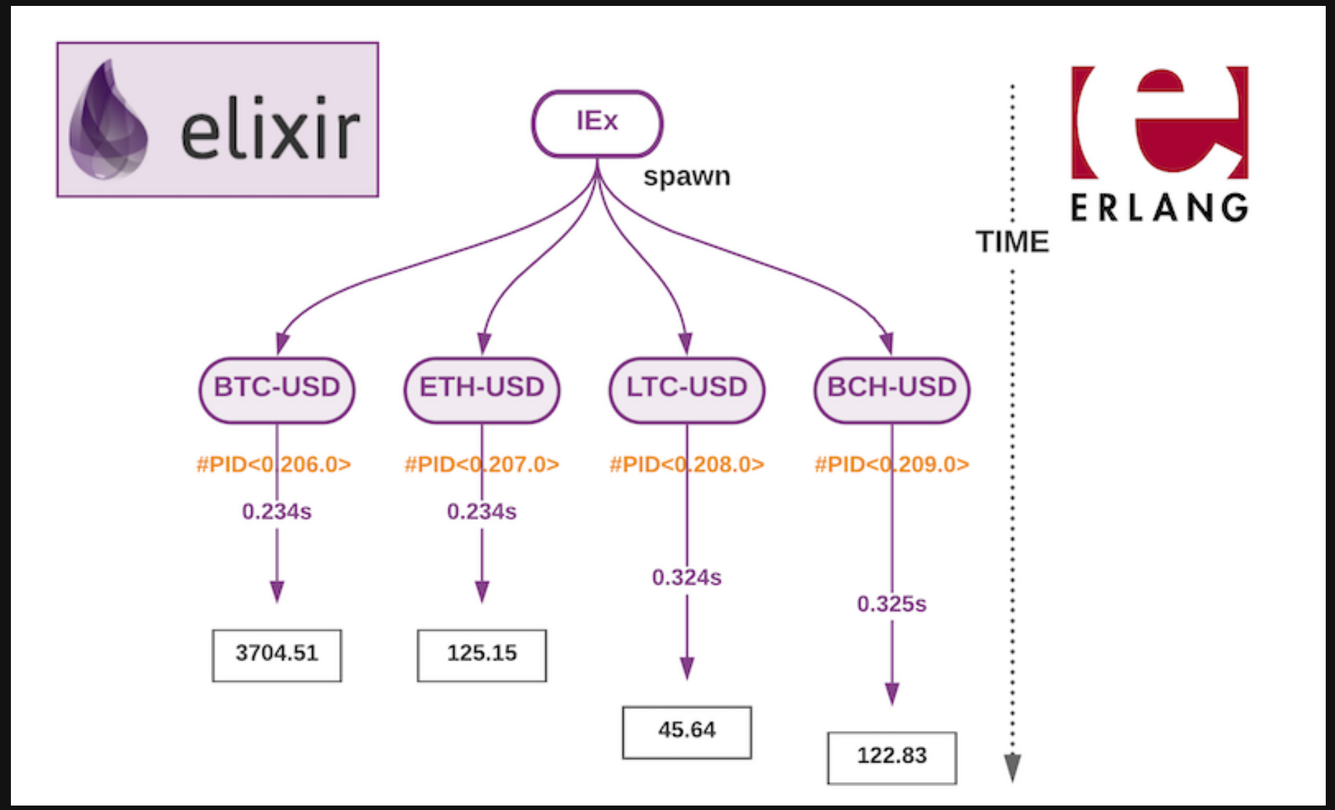
Procedures

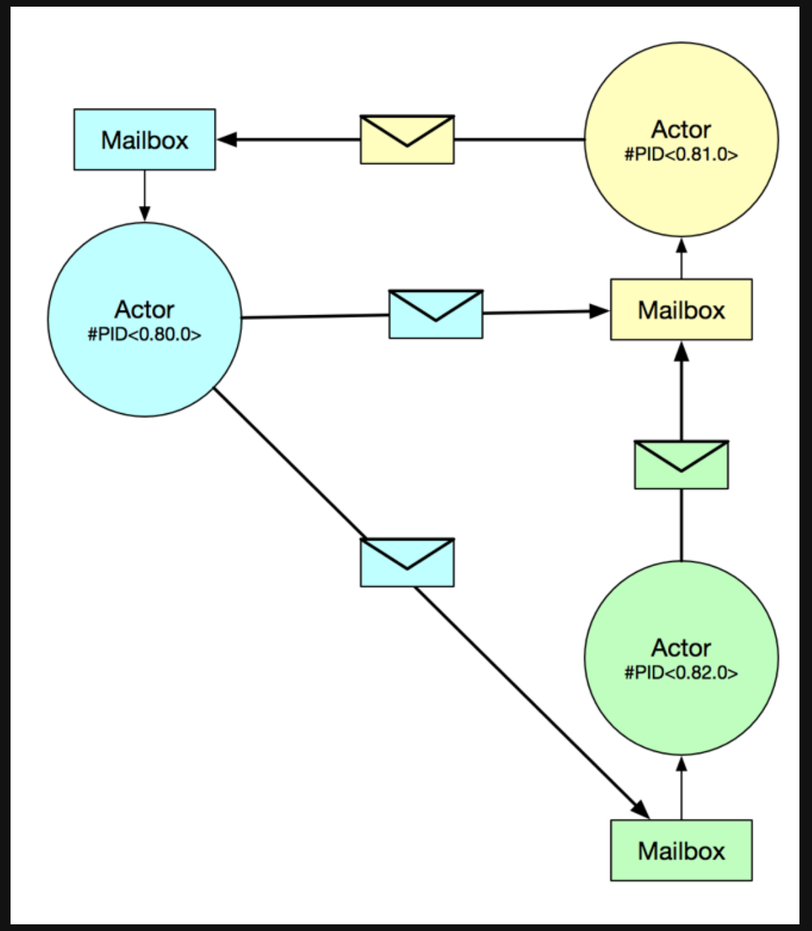


Let it crash



Let it crash

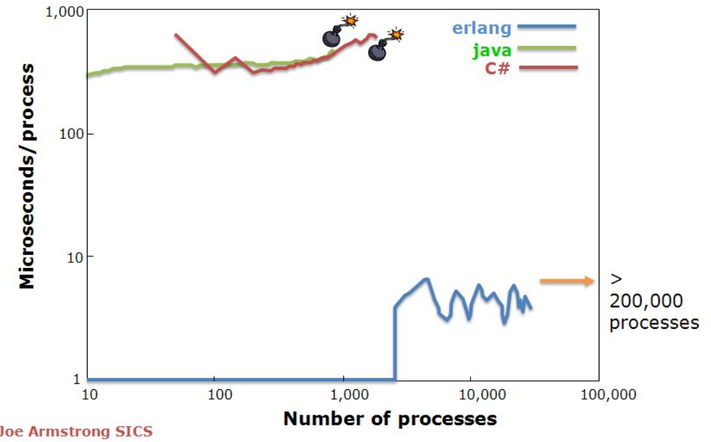

elixir
IEx
spawn
TIME
ERLANG
BTC-USD
ETH-USD
LTC-USD
BCH-USD
#PID<0.206.0>
#PID<0.207.0>
#PID<0.208.0>
#PID<0.209.0>
0.234s
0.234s
0.324s
0.325s
3704.51
125.15
45.64
122.83


Mailbox
Actor
#PID<0.81.0>
Actor
#PID<0.80.0>
Mailbox
Actor
#PID<0.82.0>
Mailbox

OVERVIEW OF ERLANG

- Everything in Erlang is process.
- Processes are strongly isolated.
- Process creation and destruction is light weight process.
- Message Passing is the only way processes can interact.
- Processes are identified by Pids.
- Processes share no resources.

Process creation times

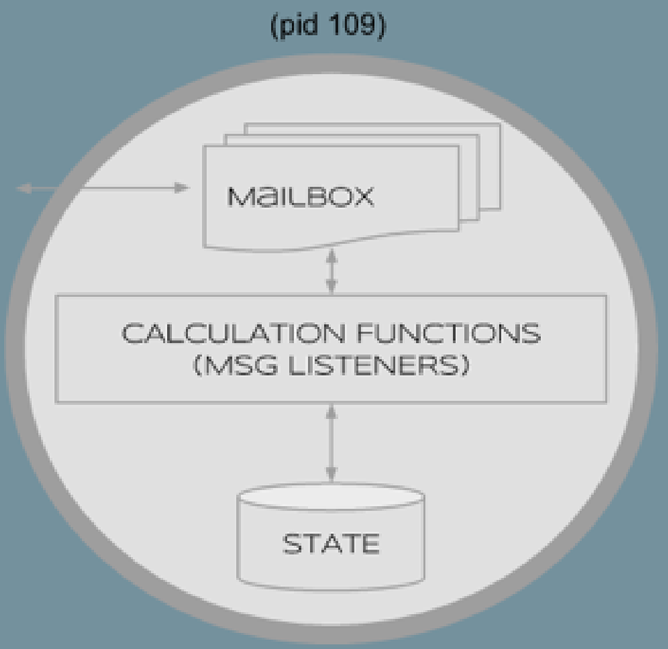


Source: Joe Armstrong SICS

OS

- Process (1)
- Process (2)
- Process (3)
- ....
- Process (n)
- Erlang VM
  -- pid 109
  -- pid 206
  -- pid 3114
  -- ...

Erlang Virtual Machine



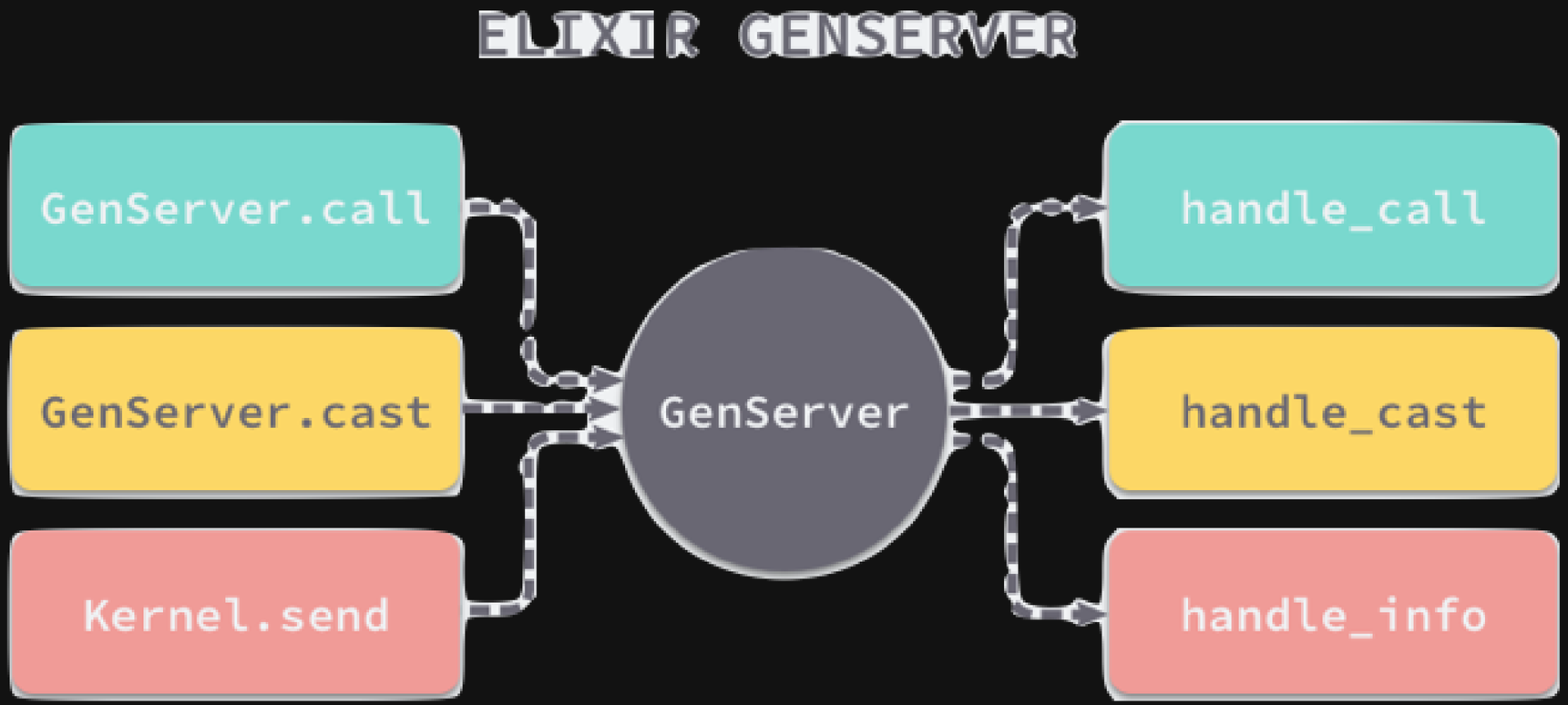
Pattern Matching Functions

```
defmodule Factorial do
    def of(0), do: 1
    def of(n), do: n * of(n-1)
end
defmodule PrintStuff do
  def print({:error, stuff}) do
    IO.puts “ERROR! #{stuff}”
  end
  def print({:ok, stuff}), do: IO.puts stuff
end
PrintStuff.print({:ok, stuff})
```

Parallel Map

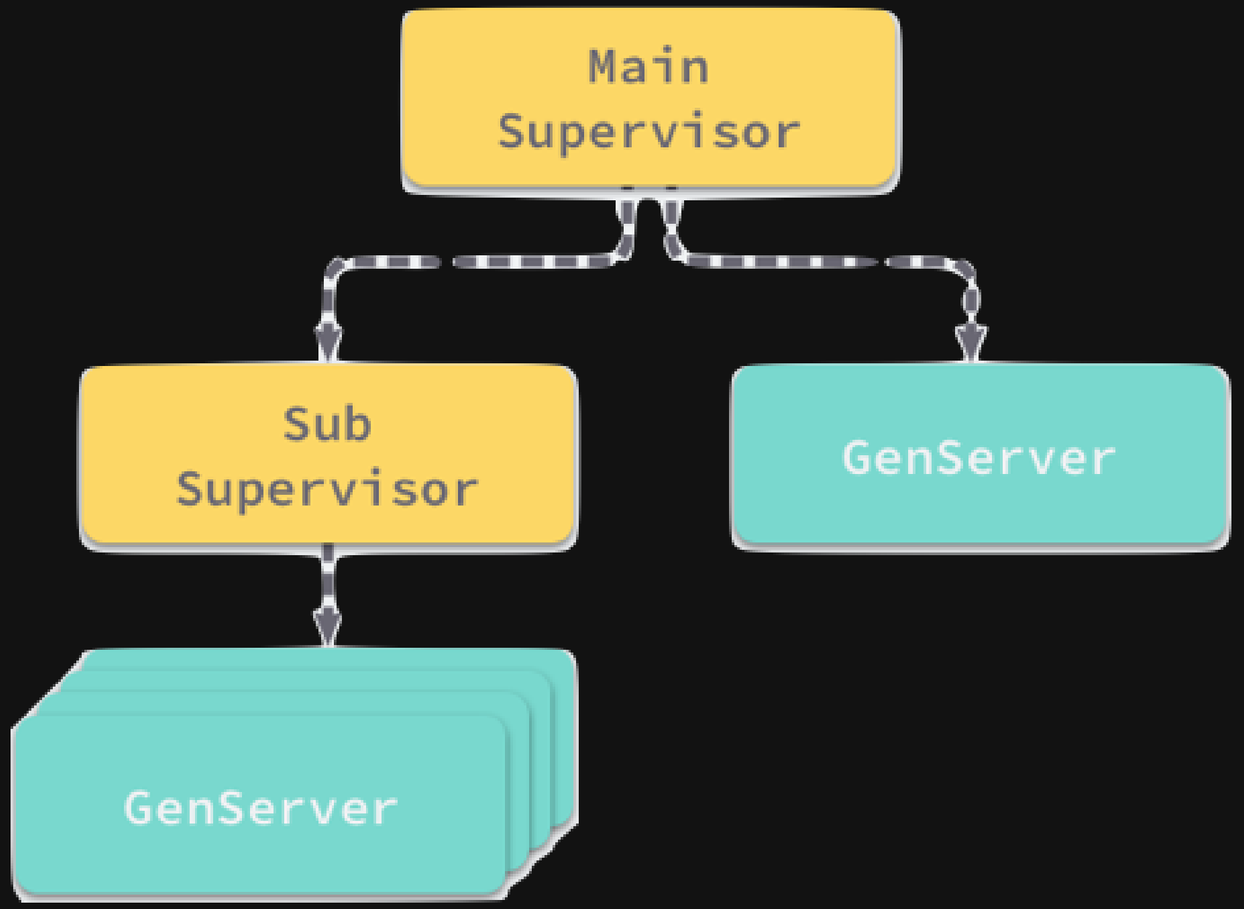
```
defmodule Paraller do
  def pmap(collection, fun) do
    me = self
    collection
    |> Enum.map(fn (elem) ->
        spawn_link fn -> (send me, { self, fun.(elem) }) end
       end)
    |> Enum.map(fn (pid) ->
         receive do { ^pid, result } -> result end
       end)
  end
end
```

OTP


ELIXIR GENSERVER
GenServer.call
GenServer.cast
Kernel.send
GenServer
handle_call
handle_cast
handle_info

```elixir
defmodule Stack do
  use GenServer

  def init(stack) do
    {:ok, stack}
  end

  def handle_call(:pop, _from, [head | tail]) do
    {:reply, head, tail}
  end

  def handle_cast({:push, item}, state) do
    {:noreply, [item | state]}
  end
end

{:ok, pid} = GenServer.start_link(Stack, [:hello])

GenServer.call(pid, :pop)            #=> :hello
GenServer.cast(pid, {:push, :world}) #=> :ok
GenServer.call(pid, :pop)            #=> :world
```

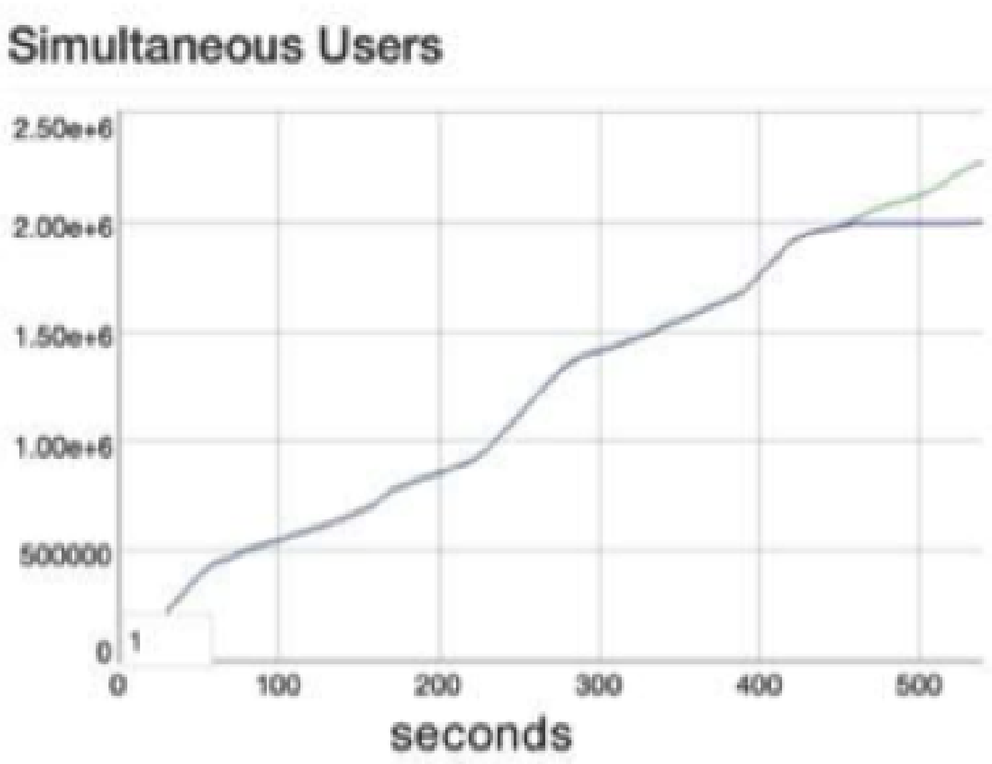
ELIXIR PROCESS TREE

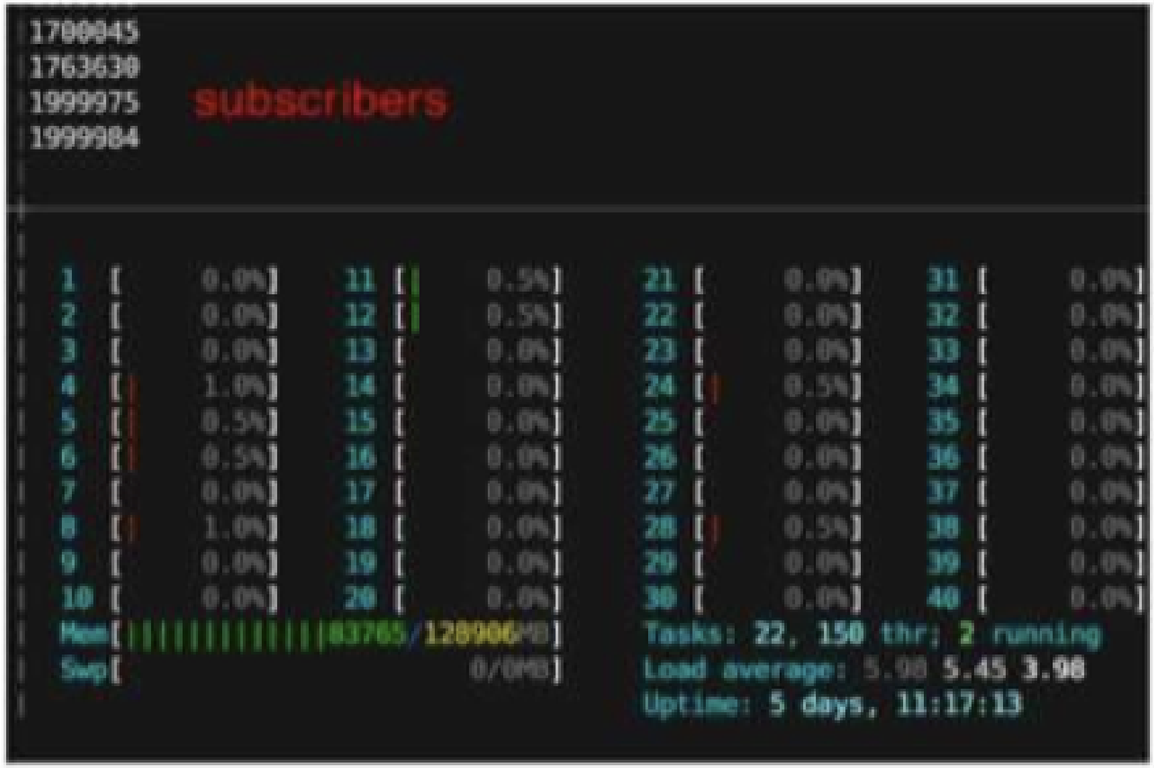


Phoenix Framework

The Road to 2 Million Websocket Connections in Phoenix

By Gary Rennie · 10 months ago · v1.0.0





Phoenix LiveView

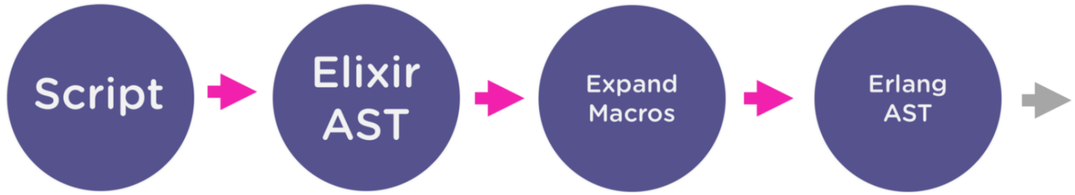
real-time user experiences with server-rendered HTML

No JavaScript!

Macros - Metaprogramming

```
defmodule MyMacros do
  defmacro nice_print({:+, _meta, [lhs, rhs]}) do
    quote do
      IO.puts """
        #{unquote(lhs)}
      + #{unquote(rhs)}
        --
        #{unquote(lhs+rhs)}
      """
    end
  end
end
```

```
iex(1)> require MyMacros
[MyMacros]
iex(2)> MyMacros.nice_print 2 + 3
  2
+ 3
  --
  5
```




```
iex> quote do: 1 + 2
{:+, [context: Elixir, import: Kernel], [1, 2]}
```

Erlang is Full Stack

Table 1.1 Comparison of technologies used in two real-life web servers

Technical requirement	Server A	Server B
HTTP server	Nginx and Phusion Passenger	Erlang
Request processing	Ruby on Rails	Erlang
Long-running requests	Go	Erlang
Server-wide state	Redis	Erlang
Persistable data	Redis and MongoDB	Erlang
Background jobs	Cron, Bash scripts, and Ruby	Erlang
Service crash recovery	Upstart	Erlang

Q&A

THANKS!