



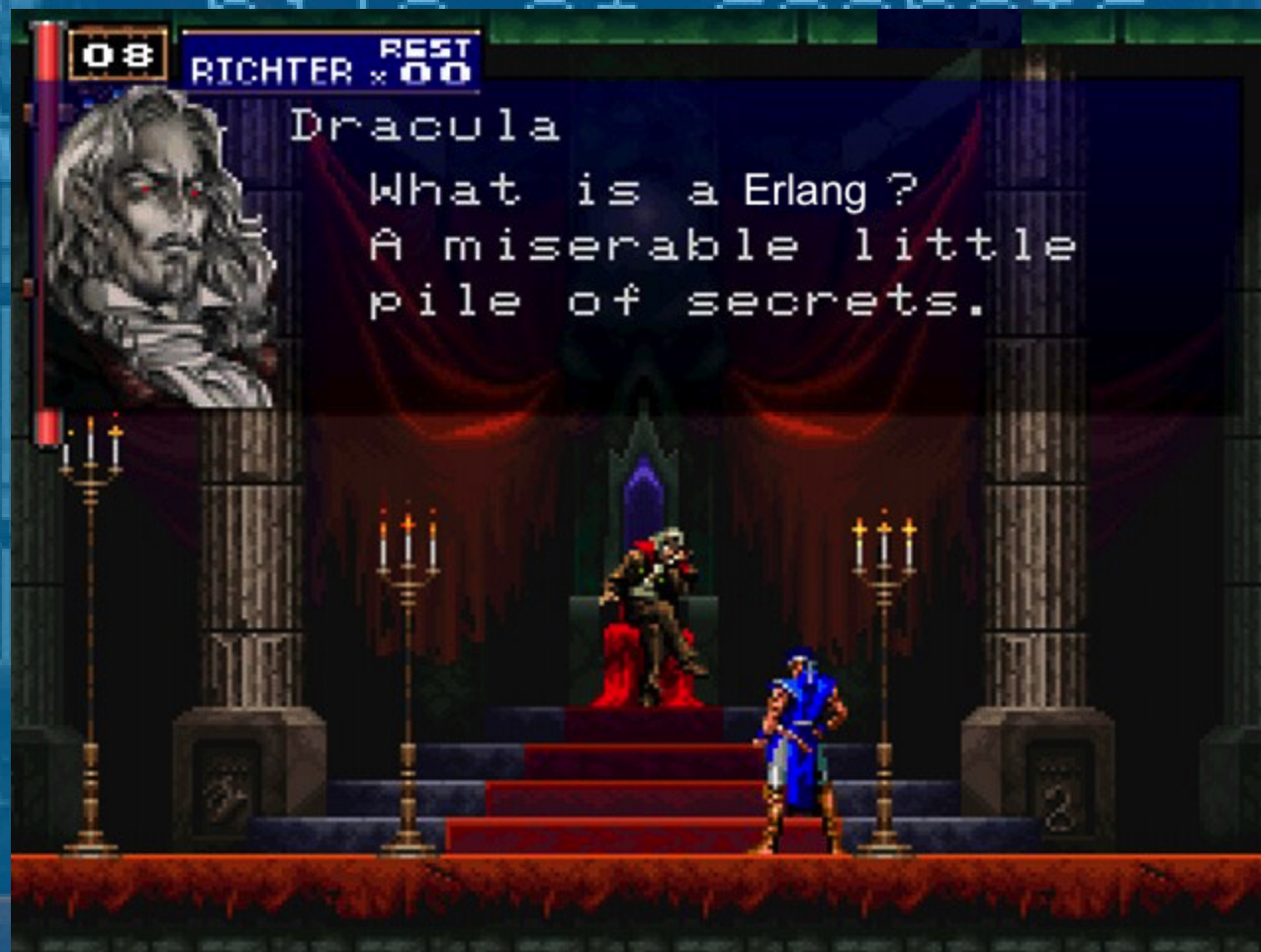
# Enter the Erlang 🦖🦎 with LFE

Todo lo que necesitas para ser un crack 🐅  
por David Cao




08

REST  
RICHTER x 00

# ¿Qué es Erlang?



# ¿Por qué Erlang?

- Es una tecnología probada con + **30** años en la ~~trinchera~~ industria  →  → 
- Manejo de concurrencia de forma sana !
- Soft-real time server side
- Capacidad de Tolerancia a fallos
- OTP: Patrones de diseño reales (~~Gang of Four~~)
- La VM es más un OS que un interprete del bytecode

# No tenés que saber **OTP** !

Si sabés un poco de Lisp, podés aprender el ecosistema sobre la marcha 🧠📚



# Erlang es piola 100 ! para soft real time 🕒

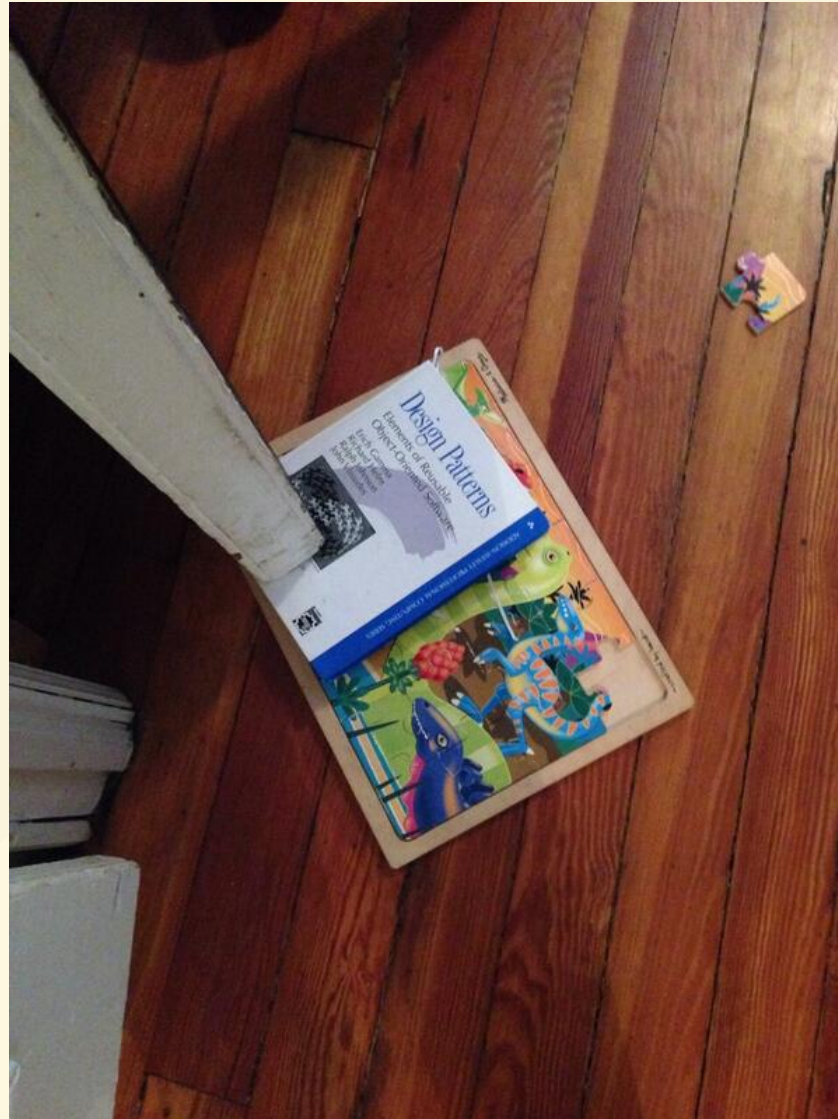
• Real 🕒 -> Perder un deadline es una falla total del sistema

• Soft 🍦 Real 🕒 -> La utilidad de un resultado se degrada después del deadline 📉, pero sigue siendo útil. En sistemas de streaming se valora fluidez del servicio.

La mayoría de los servicios son 24/7 soft-real time








# OTP: Patrones de diseño posta



# El Actor Model

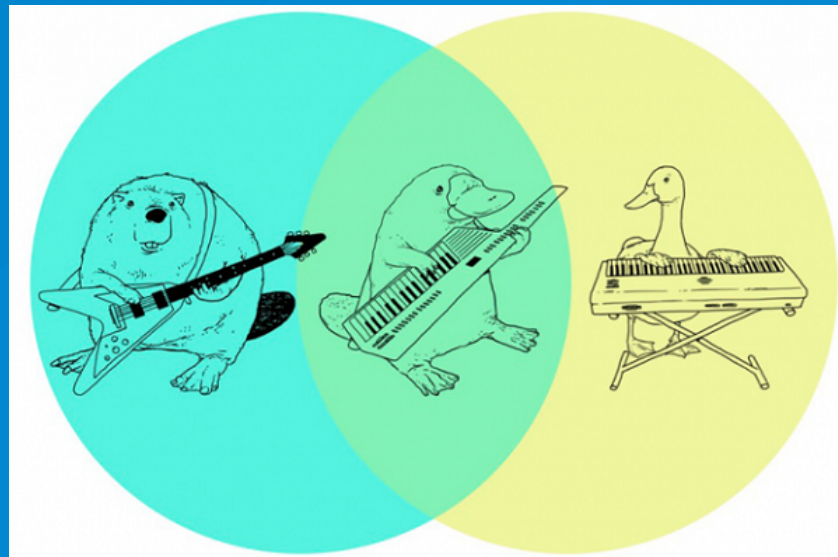
En Erlang los actores son procesos livianos aislados de la beam (NO son hilos del SO), no comparten memoria o no deberían (?)

- Se comunican por msg  (mutabilidad)
- Tiene su propio **mail box** 
- No  lock/mutex para la concurrencia  100 
- Cada actor tiene su propio heap y garbage collector
- Cualquier parecido con la POO es pura coincidencia(?)

# Behaviour

Es un patrón de diseño **AH RE** desarrollado en un módulo, es parecido a la herencia en POO o una interfaz en Java.

Una cierta cantidad de callbacks (firma) tienen que ser definidos para que el **comportamiento** funcione.

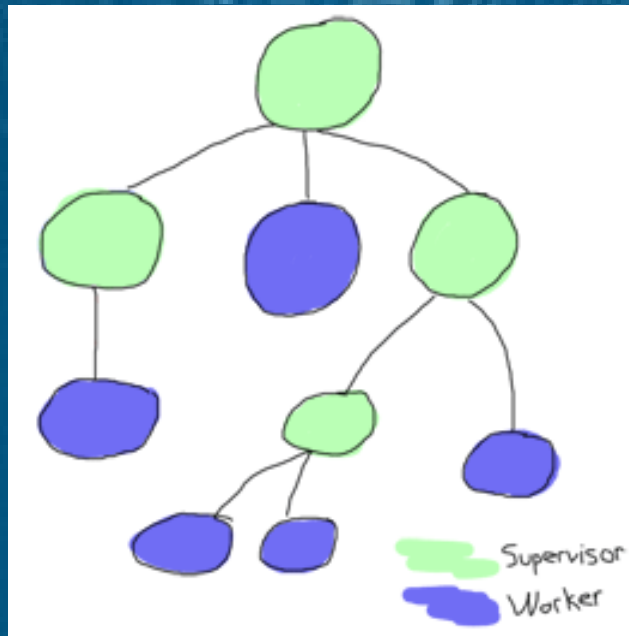




# Supervisor

El supervisor es un **behaviour** responsable de arrancar, parar y monitorear sus procesos hijos.

Siempre es un nodo del arbol de procesos, en cambio los procesos worker son hoja o terminales.



# El Gen Server aka Microservicios

Es el patrón para escribir servidores genericos en erlang.

La idea es separa la funcionalidad del manejo de la concurrencia del servidor a travez de callbacks.

**handle\_call:** llamadas síncronas (esperan respuesta)

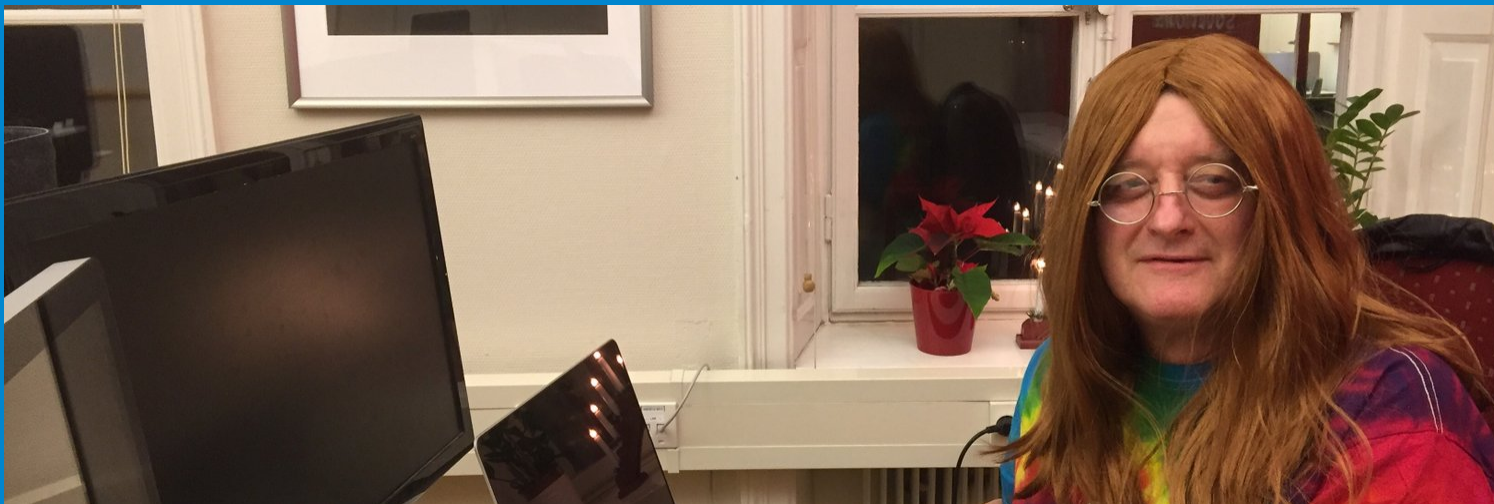
**handle\_cast:** llamadas asíncronas (Sin esperar respuesta)

Lo qur uno programe dentro del handle va a ser servido por un proceso de la beam.

`(lisp (flavoured (erlang)))`

LFE es un dialecto de LISP creado por [Roberto Virding](#) sobre la Erlang VM.

Es un Lisp2+, LFE tiene diferentes namespaces. Podes tener una fun `help` y una var `help`



# Números

Los enteros pueden ser tan grandes como quieras o te quedás sin memoria, lo que suceda primero 😊💧

Hay de punto flotante, pero a nadie le importa (?)  
Tampoco hay **'nan** ni **'infinity**, los tenes que fabricar.

```
lfe> ( / 1000000444 991)
1009082.1836528758
lfe> (/ 1.0 0.0)
exception error: error in arithmetic expression
    in (erlang : / 1.0 0.0)
```

# Cadenas

las cadenas en Erlang son **listas** 🍏 🎩 ... de enteros 😊

```
lfe> (== "Ceci n'est pas une " (99 104 97 238 110 101))  
"Ceci n'est pas une chaîne"
```

```
lfe> (lfe_io:format "Maximale ascii est: ~c. "  
    (list (lists:max "Ceci n'est pas une chaîne")))  
Maximale ascii est: î. ok
```

```
erlang> io:format("Maximale ascii est: ~c. ",  
    [lists:max("Ceci n'est pas une chaîne")]).  
Maximale ascii est: î. ok
```




# Átomos

son **enums** que se representan así mismos, los átomos empiezan con comilla simple '.

```
lfe> (erlang:is_atom 'desinflamante')  
true
```

```
erlang> erlang:is_atom(desinflamante).  
true
```

- **'true** tiene un valor **truthy** y el **'false** **falsey**
- No hay **null** , pero podés definir el átomo **'null**, **'undefined**, **'none**, **'nothing**, **'lol**, **'ahre**

# Tuplas

Podes construir tuplas, tripletas, cuartetos ...

```
lfe> (tuple 'ok "I am a pickle!")  
#("I am a pickle!")  
lfe> (tuple 1 2 3 4 5)  
#(1 2 3 4 5)
```

```
lfe> #(1 9)  
#(1 9)  
lfe> (:= #(1 9) (tuple 1 9))  
true
```

# Mapas

Son estructuras clave valor

```
lfe> (map 'key 'value)
#M(key value)
lfe>
lfe> (map 'lfe "Erlang" 'creator "Robert Virding")
#M(lfe "Erlang" creator "Robert Virding")
```

# Módulos, Funciones y Pattern Matching

```
lfe> (set (tuple 'error msg) (tuple 'error "Error :("))  
#(error "Internal error")  
lfe> msg  
"Error :("
```

El nombre de un modulo es **atom()**!

Además el Pattern Matching es conceptualmente similar al dispatcher dinámico en la POO.

```
(defmodule conversion  
  (export (convert-length 1)))  
  
(defun convert-length  
  (((tuple 'centimeter x)) (tuple 'inch (/ x 2.54)))  
  (((tuple 'inch y)) (tuple 'centimeter (* y 2.54))))
```

# Módulos, aplicaciones y librerías

En Erlang tener un **main(args ...)** no tiene mucho sentido (como punto de entrada), es mas para nostalgicos de otros lenguajes.

Dado que que podes tener varios procesos escuchando.



# Ejemplo de Exercism

```
(defmodule leap
  (export all))

(defun leap-year
  ((year) (when (== 0 (rem year 400)))
    'true)
  ((year) (when (== 0 (rem year 100)))
    'false)
  ((year) (when (== 0 (rem year 4)))
    'true)
  ((_year)
    'false))
```

# Ejemplo de Exercism

```
(defmodule rna-transcription
  (export (to-rna 1)))

(defun to-rna-char
  ([#\G] #\C)
  ([#\C] #\G)
  ([#\T] #\A)
  ([#\A] #\U))

(defun to-rna (dnaList)
  (lists:map #'to-rna-char/1 dnaList))
```

# Erlang viene con 3 DBs build-in

**ETS** (Erlang Term Storage) es una BD en memoria para guardar todo tipo de termino Erlang

**DETS** (Disk ETS) es similar a la ETS pero con persistencia en disco con un límite de 2G.

**Mnesia** es una capa comstruida sobre la ETS y la DETS que permite transacciones.

Las mas usadas son ETS y Mnesia







# Dieselpunk



