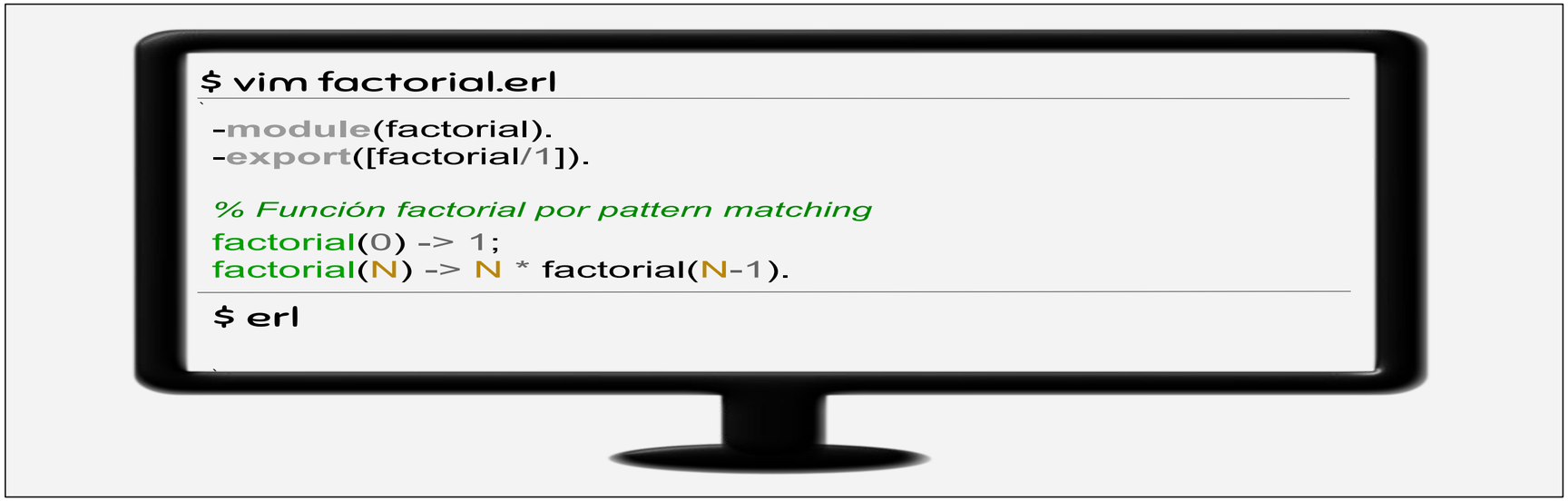
Introducción a Erlang. Fundamentos, y conceptos básicos.

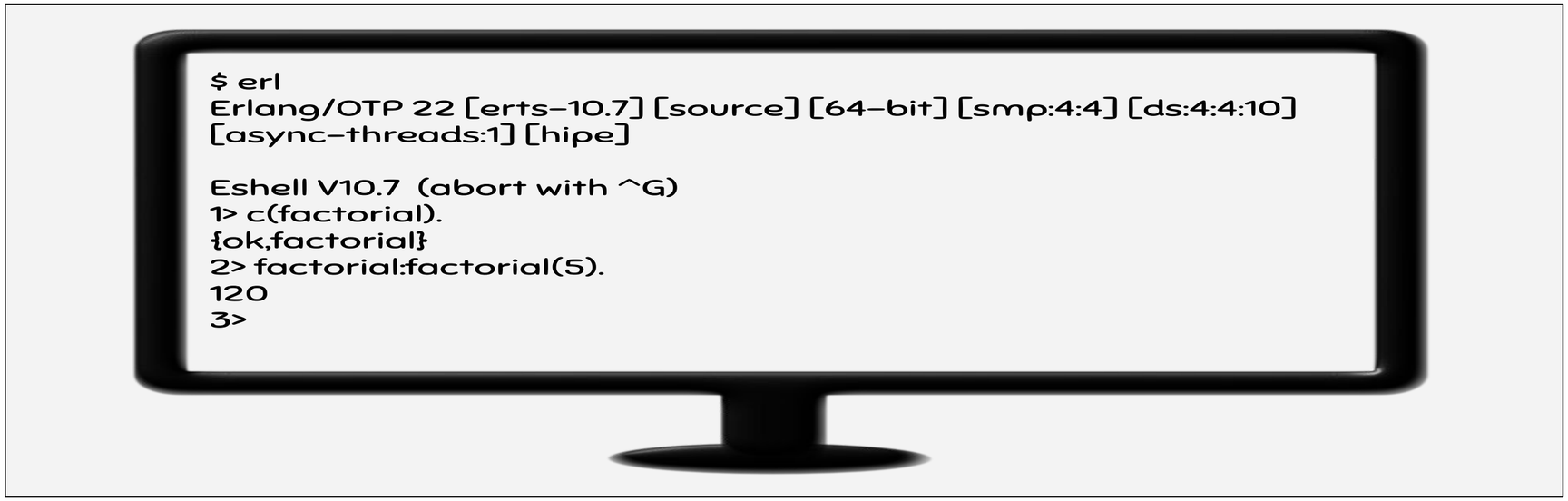

Visitar la
Página de
Erlang.org
Erlang
www

¿ Qué es Erlang ?

- Lenguaje Funcional **NO PURO.** Pero será lo más puro posible.
- Es de tipado dinámico
- Las funciones se definen usando recursión y pattern matching (como en Haskell)
- Es un lenguaje orientado a sistemas industriales a gran escala con requerimientos de tiempo real.
- Se basa en dos principios que ya hemos visto: la concurrencia y fiabilidad.
- Tiene además la posibilidad de ejecutar transparentemente de forma distribuido
- Utiliza una máquina virtual!



Primera aproximación a código Erlang. Ejemplo clásico de función recursiva por pattern matching: Factorial!
Primero se declara el modulo `math1`, y se declara una lista de funciones a exportar, que en nuestro caso es `factorial` además avisando que toma un argumento. Notar que todas las directivas terminan con un punto (.).
El pattern matching de factorial es en el primer argumento `0` lo asocia con el fragmento de la derecha de la flecha `->`, los casos se separan por puntos y coma `;`,
Y en el siguiente caso toma una variable `N` (notar que es en mayúscula), y ejecuta el código asociado a la derecha. El análisis de casos es secuencial, y se usa el primero que matchea, luego se reemplaza el valor del argumento en el cuerpo a ejecutar, y se ejecuta.
Utilizaremos la Shell de Erlang para ejecutar el código.



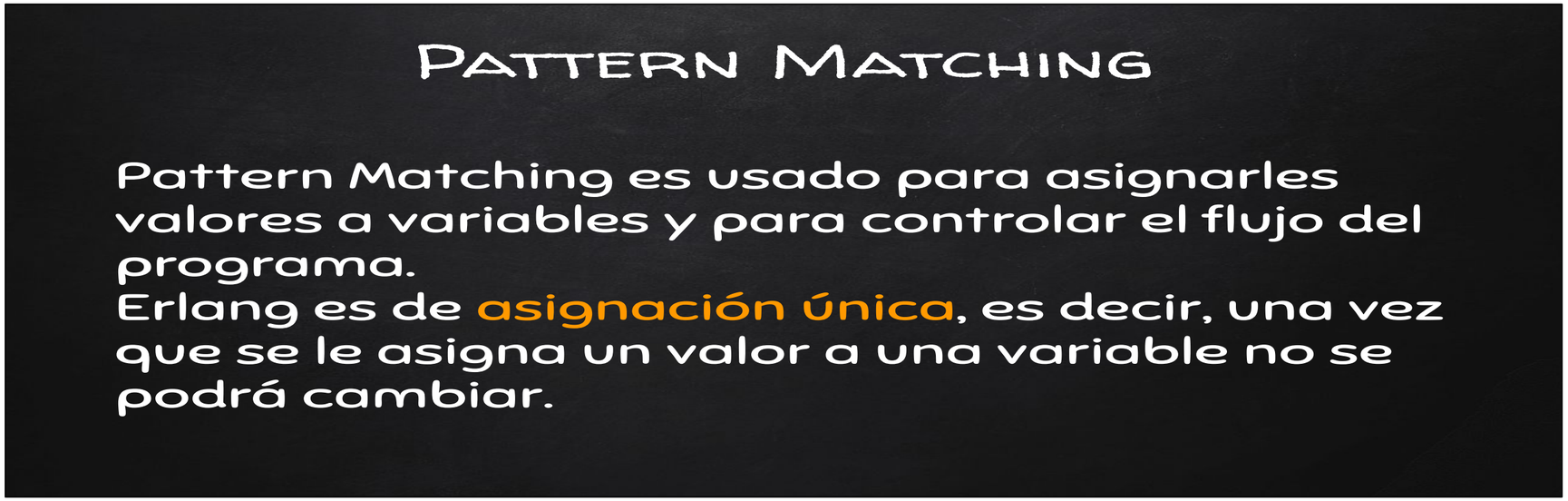
Slide que continúa de la llamada a la erlang shell.
Cargamos el modulo que declaramos en la slide anterior `math1`, usando `c(math1).` . El archivo que creamos tiene que estar en el mismo directorio donde ejecutamos la shell.
Nos avisa que fue cargado correctamente, y efecutamos la función `factorial` con 5.

Actividad Corta: 2 mins

Cargar y Ejecutar el código de la función Factorial.

Explicar que para salir de la EShell puede usar `q().` o ^G y q..

Tipos de datos en erlang.



Aclarar que Erlang es SSA (Single Static Assignment) y que se les puede asignar una única vez un valor a las variables.
Pattern Matching ya lo conocen y no debería hacer mucho más hincapié. Aclarar que la wildcard es `_`, y de ser significativo pueden darle nombre `_Nombre`

Actividad 3: Completar el fragmento de código.

min.erl :

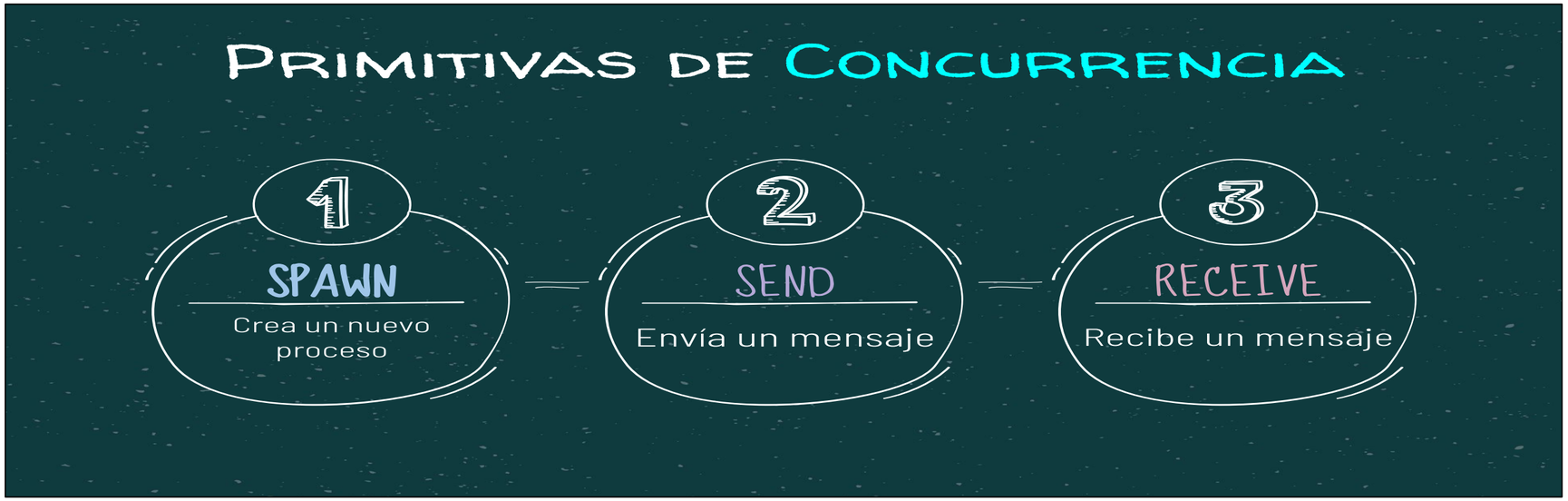
```
-module(min).
-export([min/1]).

min([Hd]) -> ??;
min([Hd|Tl]) ->
    Rest = min(Tl),
    if
        ??
    end.
```

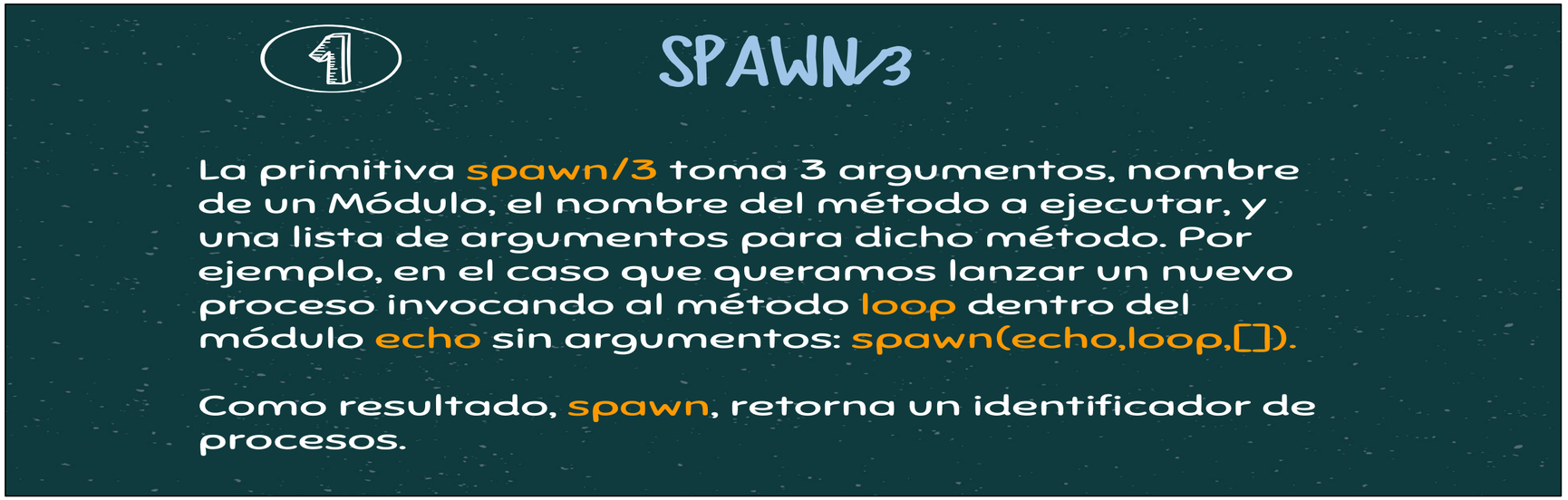
Acá introducimos otro concepto fundamental de Erlang: Ejercicio de Pattern matching!
Al terminar de codear, cargar el archivo en la EShell.
Links útiles:
https://erlang.org/doc/reference_manual/expressions.html#if
https://erlang.org/doc/reference_manual/expressions.html#term-comparisons

Spoiler alert!

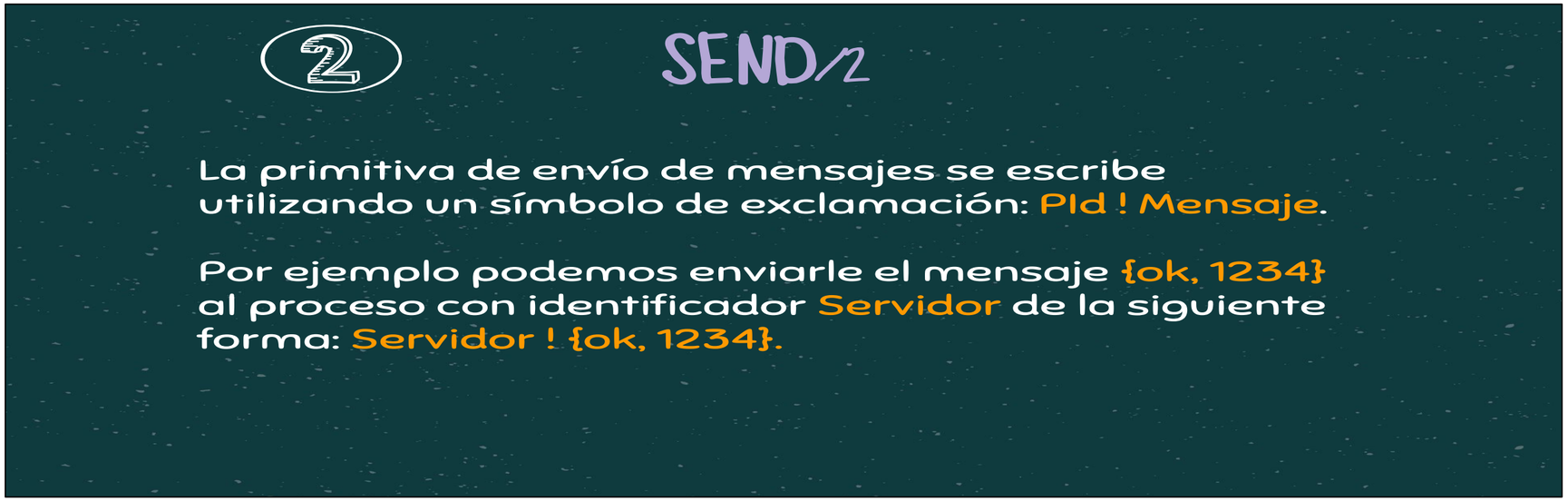
Introducción al modelo de actores. Procesos de Erlang = Actores!

Primitivas de Concurrencia

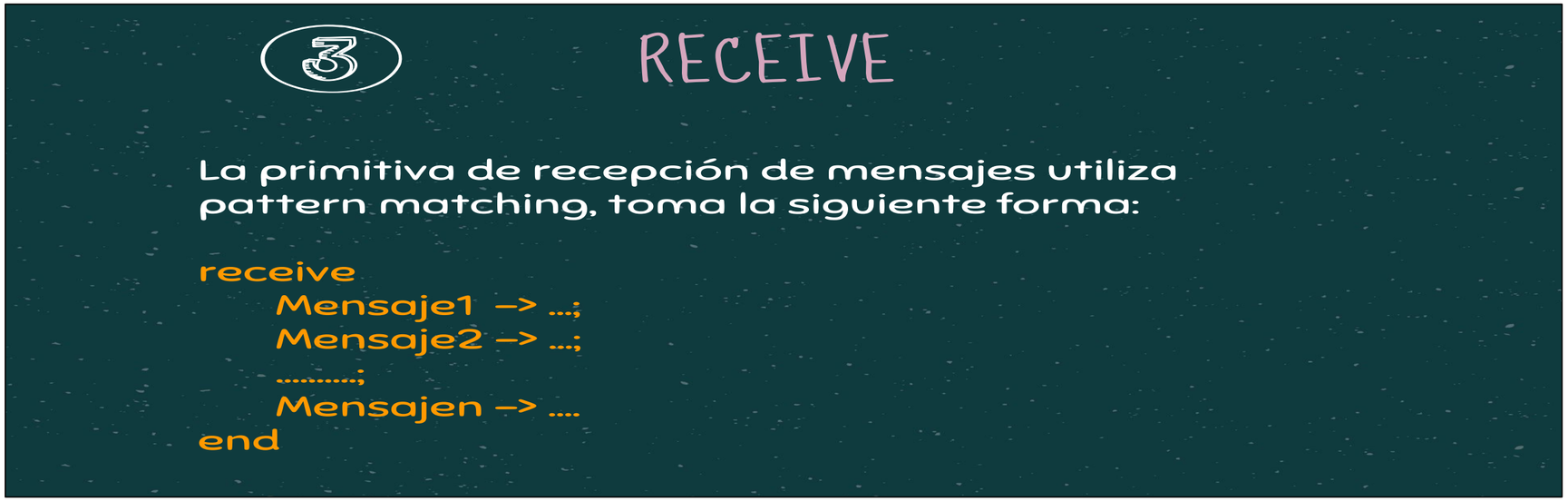




Primitiva de creación de procesos.Esto debería hacerles acordar al `fork()` de C. Aunque recordemos que estamos en realidad creando un actor! (En Erlang los procesos = actores).



Primitiva de envío de mensajes. Resaltar la diferencia con send de C. La magia está en que los identificadores de procesos dicen mucho más información que un simple número de ejecución en una computadora. Esto lo veremos más adelante
Detalles, los mensajes son evaluados antes de realizar el envío, es decir, si se quiere enviar el resultado de una función, se evalúa la función y se envía el resultado. Lo mismo puede suceder con el PID.



Primitiva de envío de mensajes.
La primitiva de recepción de mensajes se queda entonces esperando a que llegue un mensaje, y cuando llegue buscá el patrón que coincida con el mensaje y ejecuta el código correspondiente.
El resultado del fragmento de código será el resultado del fragmento ejecutado dependiendo del patrón del mensaje.

¡Live coding!

Escribamos un programa en Erlang que cree dos procesos, y se envíen mensajes entre ellos.

Un objeto contador es un objeto que soporta una operación que adiciona uno a su valor, y una operación de extracción del valor.

Envío y Recepción de Mensajes

Los envíos de mensajes no al proceso sino a un buzón de mensajes de un proceso.
Mientras que receive lo que hace es mirar en el buzón del proceso actual.
Esto quiere decir que receive es selectivo, y va a buscar el primer mensaje que se corresponda con una cláusula del Pattern Matching.

Rompemos con la idea de que los procesos se mandan mensajes entre ellos, y se introduce la noción de buzón.
Cuando se envía un mensaje, el mensaje se almacena en el buzón de un proceso.
Cuando se espera recibir un mensaje, se busca en el buzón.
Si ningún mensaje se puede asociar a algún patrón del matching, el proceso se va a dormir hasta que lo despierte la llegada de un mensaje en el buzón.
Los mensajes que no se matchean se guardan para eventual procesamiento.

Entonces Erlang soluciona todos los problemas que teníamos en C! Erlang es lo más, Martín te odio que nos hiciste programar en C, presentaste algunas funciones, y penosamente explicaste cómo funciona la interne! Erlang está pensado y diseñado para crear procesos y enviar mensajes entre ellos, lo que permite que los sistemas informáticos evolucionan fácilmente. Pero no es la solución a todos nuestros problemas, es una herramienta más que podrán usar llegado al caso, mientras que C soluciona otros problemas donde Erlang no funciona muy bien, como la interacción con el sistemas operativo.

Cómo seguimos? Todavía faltan un montón de conceptos de Erlang pero considero que son mejores para incorporar mediante la práctica y el uso de Erlang. Cuestiones como PIDs, BIF (Built-In Functions), etc, que vamos a trabajar en la práctica que verán publicada en la proximidad temporal.