

Erlang
SOLUTIONS

inaka

ERLANG & ELIXIR
FACTORY LITE
Buenos Aires 2017





ABSTRACT TYPES **& OTHER YERBAS**





OPAQUE TYPES & OTHER YERBAS



HELLO!

Brujo Benavides

elbrujohalcon@inaka.net

Inaka.net / www.erlang-solutions.com

@elbrujohalcon

inaka

Erlang
SOLUTIONS

ERLANG IS...

- ▶ **Functional**
- ▶ Concurrent
- ▶ Distributed
- ▶ Fault-Tolerant



THIS TALK IS ABOUT...

- ▶ **Types**
- ▶ Modules
- ▶ Functions
- ▶ Specs



BUT WHY TYPES?



WHAT IS **SOFTWARE?**



SOFTWARE AS **MODELING**



ENTITIES AS OBJECTS



ENTITIES AS INSTANCES



WHAT ABOUT ERLANG?



MANY OPTIONS

- ▶ **Basic** Types

- ▷ Lists, Tuples, Numbers

- ▷ Tagged Tuples

- ▷ Records

- ▷ Maps

- ▶ **Opaque** Types





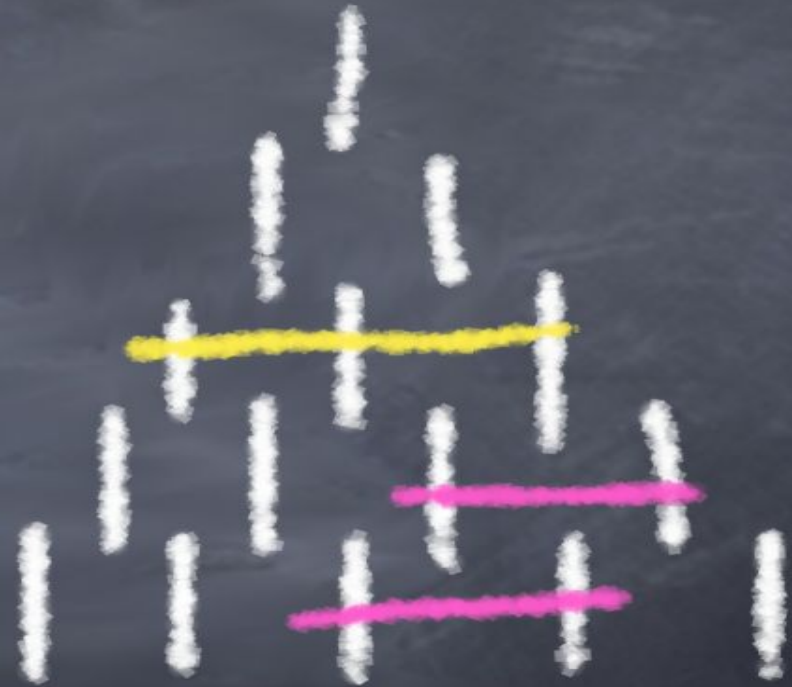
A SAMPLE
SYSTEM: NIM



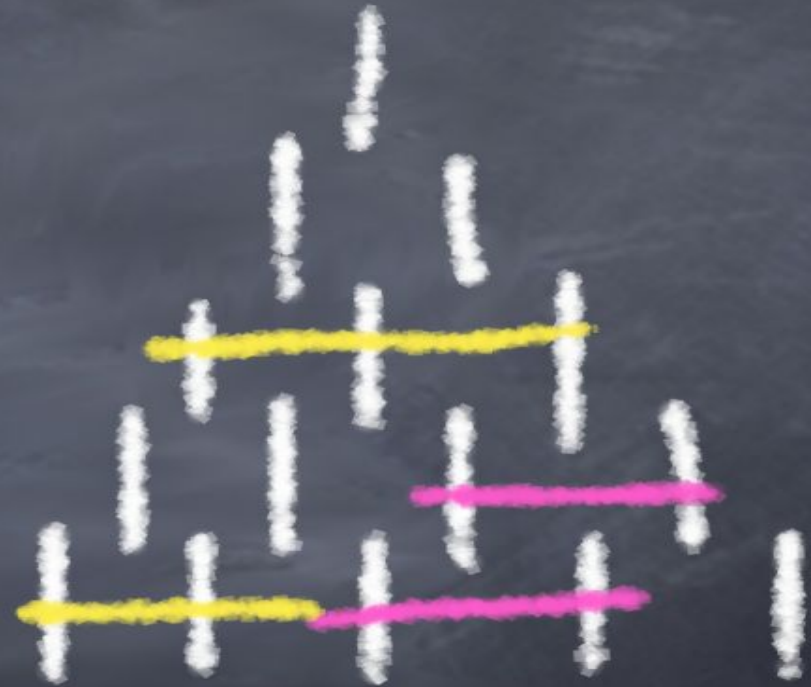
A SAMPLE
SYSTEM: NIM



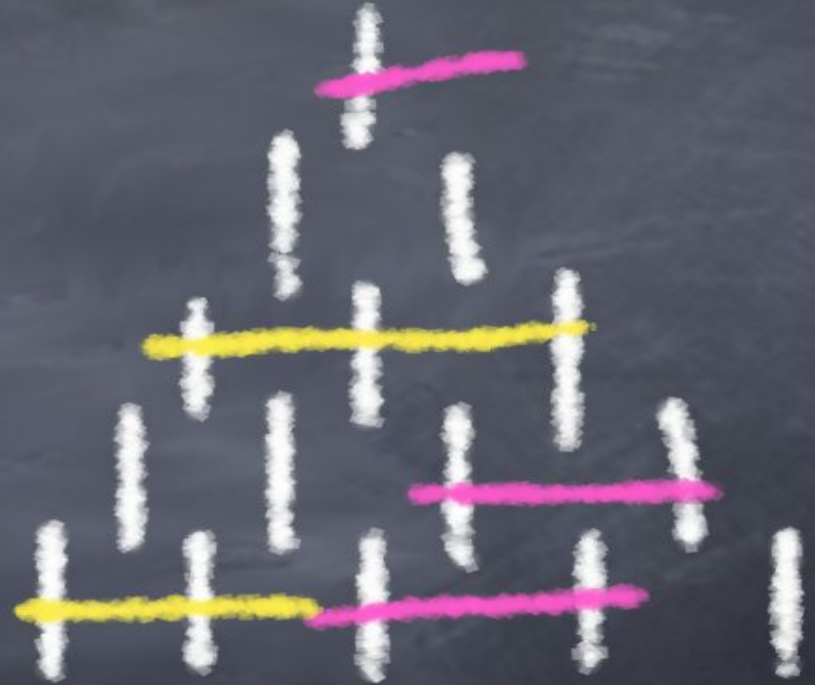
A SAMPLE
SYSTEM: NIM



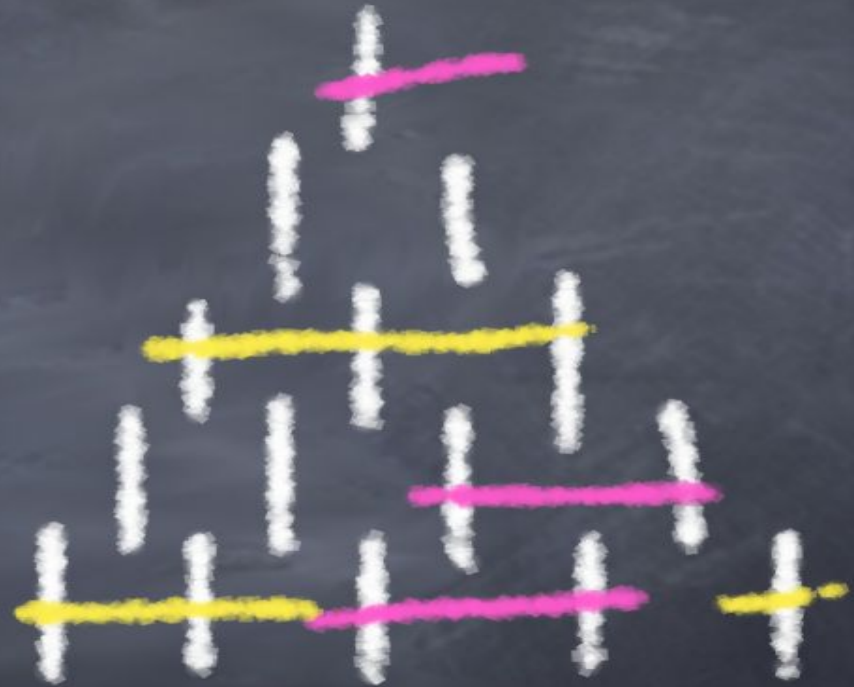
A SAMPLE
SYSTEM: NIM



A SAMPLE
SYSTEM: NIM



A SAMPLE
SYSTEM: NIM



A SAMPLE
SYSTEM: NIM



A SAMPLE
SYSTEM: NIM



A SAMPLE
SYSTEM: NIM

8

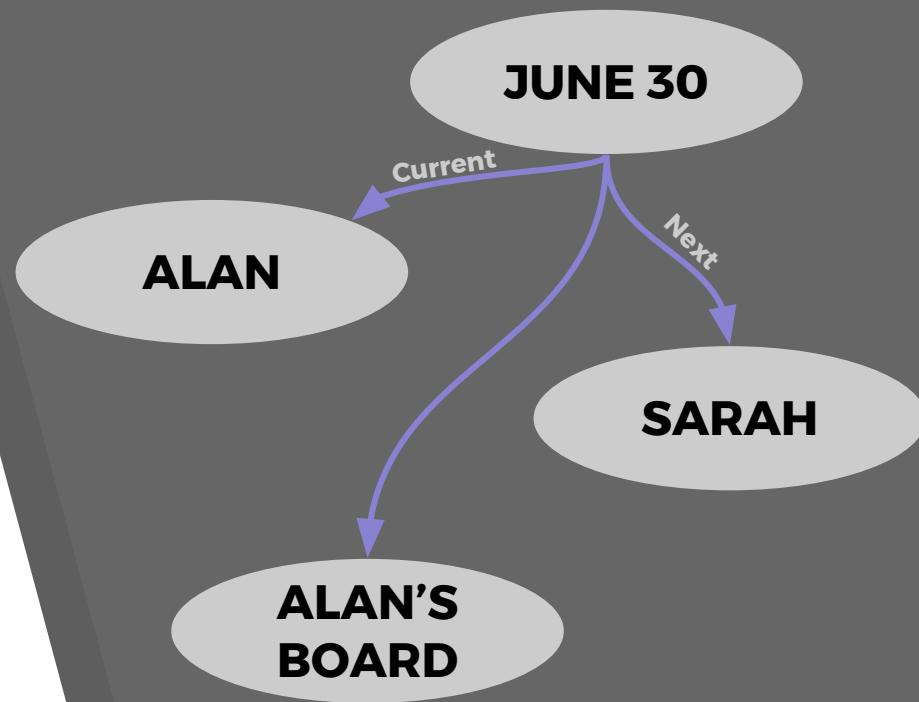


A SAMPLE
SYSTEM: NIM

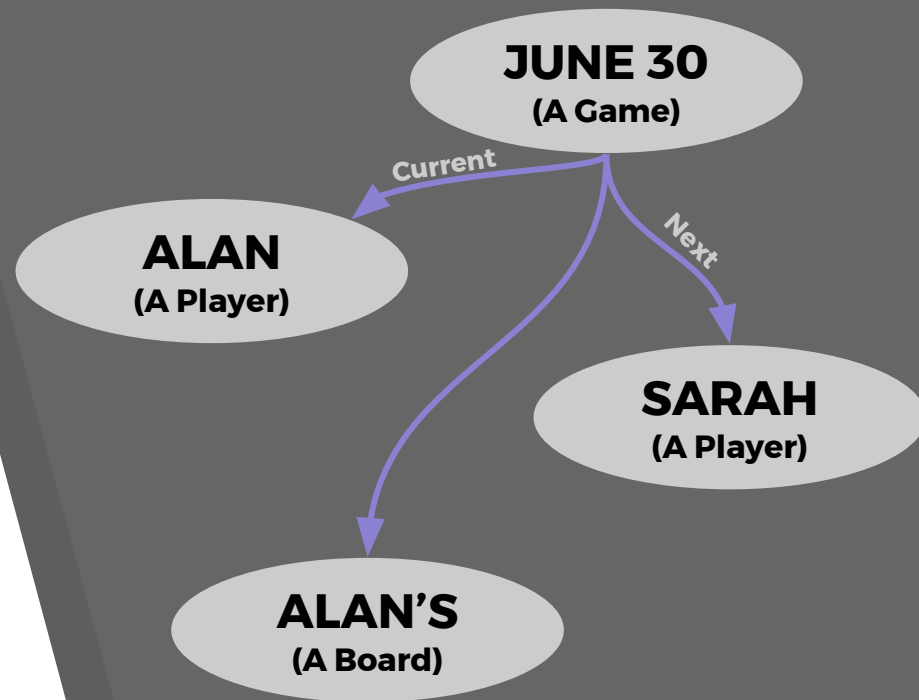
RECOGNIZING ENTITIES



RECOGNIZING ENTITIES

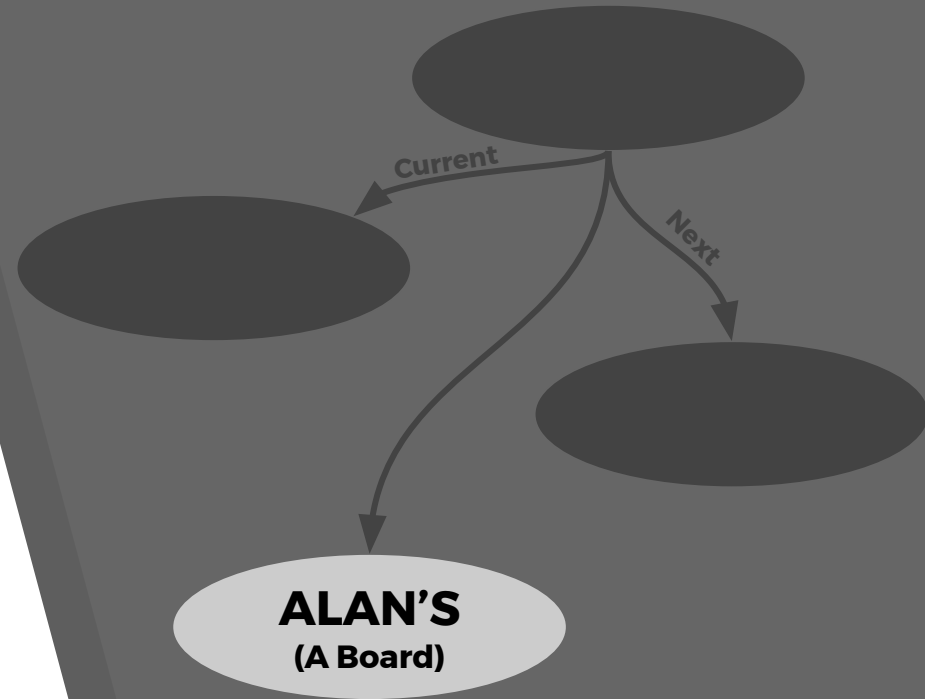
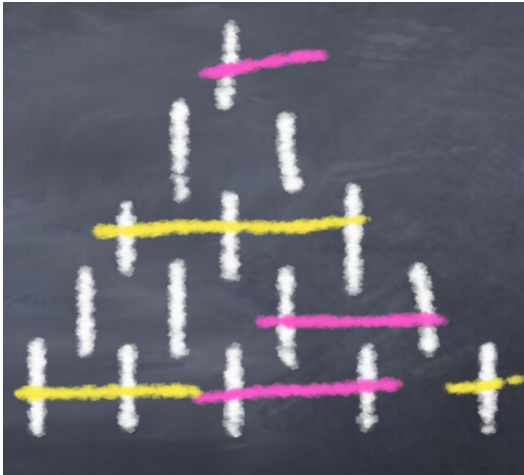


RECOGNIZING ENTITIES



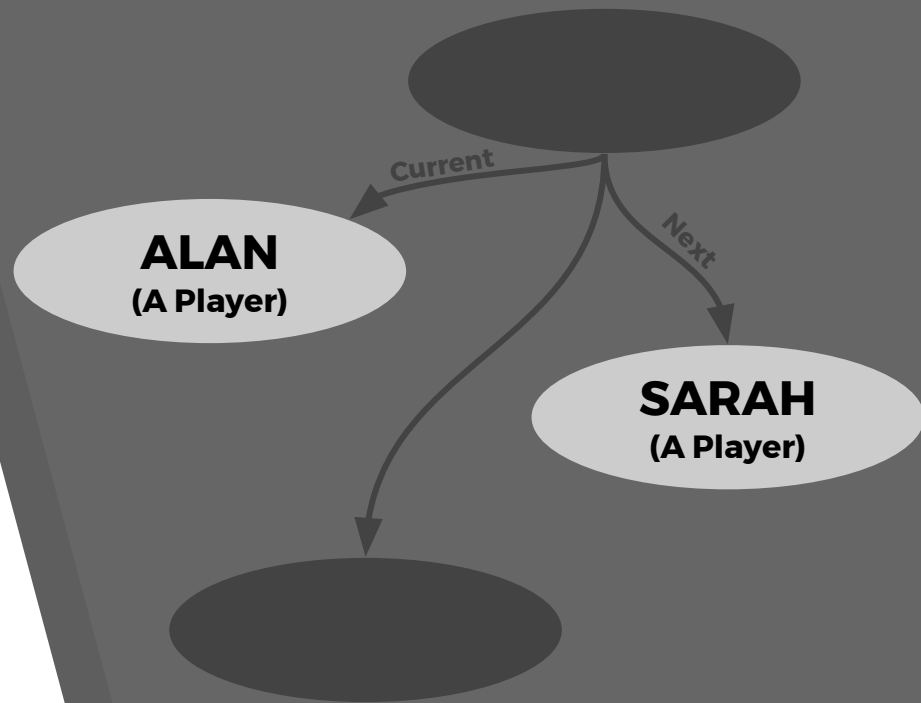
A BOARD

- ▶ Rows
 - ▶ With **Sticks**
 - ▶ Some **Crossed Out**
 - ▶ Some Still **Available**



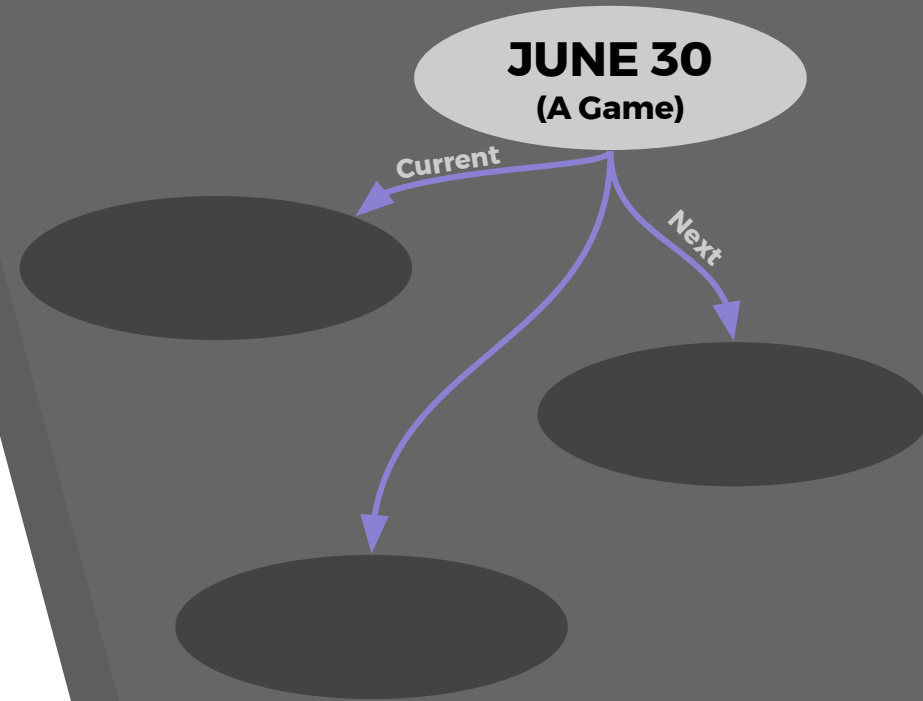
A PLAYER

- ▶ Name



A GAME

- ▶ Two **Players**
 - ▶ **Current**
 - ▶ **Next**
- ▶ A **Board**



BASIC TYPES

BASIC TYPES

- ▶ **Players** as **Strings**
- ▶ **Board** as a **5-Tuple**
 - ▶ **Rows** as **Lists of 0 | 1**
- ▶ **Game** as a **3-Tuple**

```
CurrentPlayer = "Alan",  
NextPlayer = "Sarah",  
Board =  
    { [1]  
      , [1, 1]  
      , [1, 1, 1]  
      , [1, 1, 1, 1]  
      , [1, 1, 1, 1, 1]  
    },  
Game =  
    { CurrentPlayer  
      , NextPlayer  
      , Board  
    }.
```

BASIC TYPES

- ▶ **Functions** use
 - ▶ **Pattern-Matching**
 - ▶ **BIFs**

```
cross(RowN, Pos, Len, Game) ->  
  {Current, Next, Board} =  
    Game,  
  Row = element(RowN, Board),  
  
  %% changes 1s by 0s in a row  
  NewRow =  
    cross(Pos, Len, Row),  
  
  NewBoard =  
    setelement(  
      RowN, Board, NewRow),  
  
  {Next, Current, NewBoard}.
```

PROBLEMS

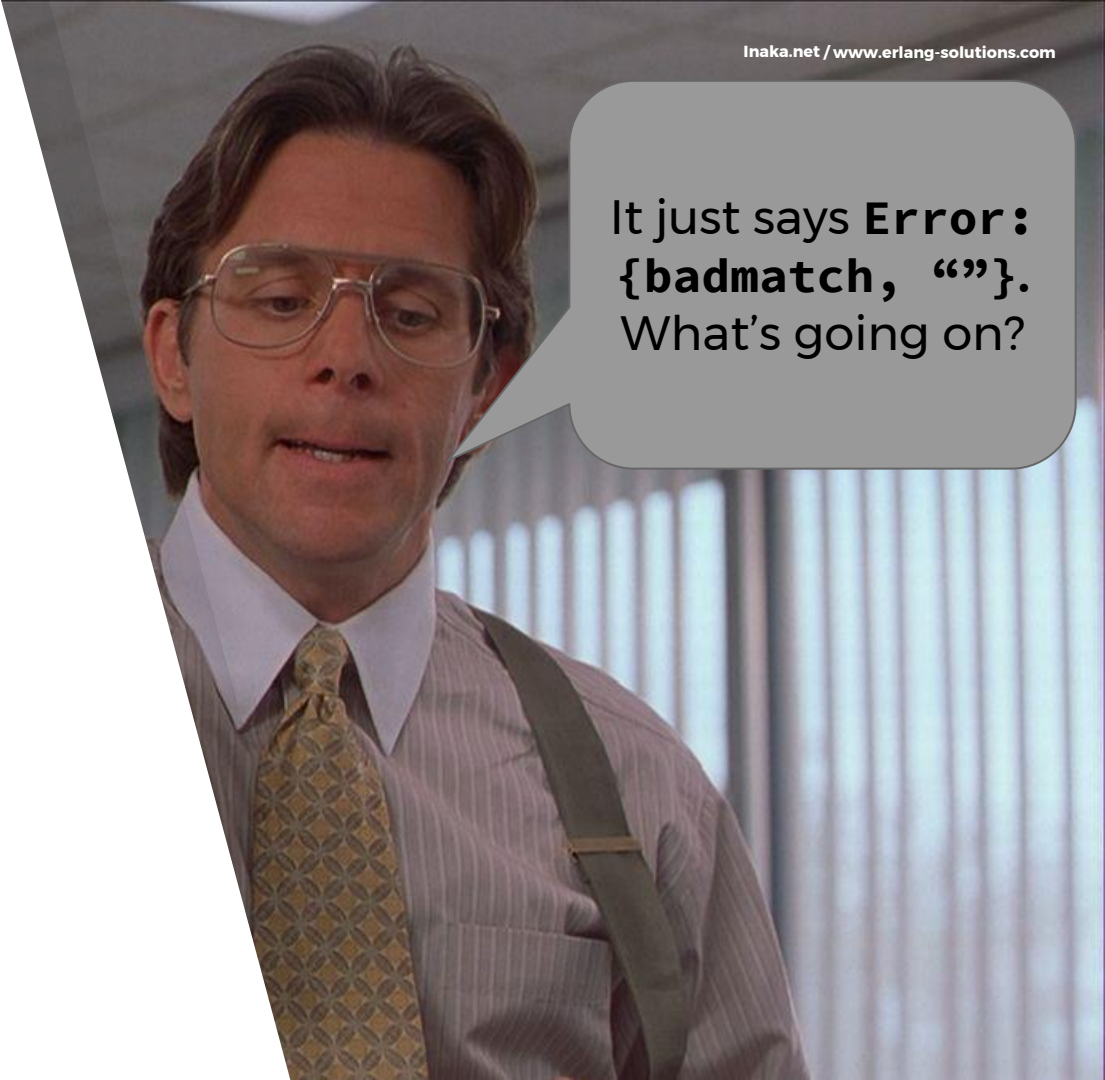


A photograph of Mark MacCallister, a character from the British sitcom 'The IT Crowd'. He is wearing a light-colored striped shirt, a patterned tie, and dark suspenders. He has dark hair and is wearing large, clear-rimmed glasses. He is looking slightly to the right with a neutral expression. The background is a blurred office setting with vertical blinds.

If you could go ahead and add a **name** to all **games**, that would be great!

PROBLEM #1:

Low Maintainability

A man with dark hair and glasses, wearing a light-colored striped shirt, a patterned tie, and suspenders, looks slightly confused or concerned. He is standing in front of a window with vertical blinds. A speech bubble originates from his mouth, containing text about an Erlang error.

It just says **Error:**
{badmatch, ""}.
What's going on?

PROBLEM #2: **Hard to Debug**

A photograph of Mark Zuckerberg, dressed in a light-colored striped shirt, a patterned tie, and suspenders. He is wearing large, clear safety glasses and has a slightly confused or questioning expression on his face. The background is a blurred office setting with vertical blinds.

I just added a
new row to my
board. Why
doesn't it work?

PROBLEM #3:

No Encapsulation

A photograph of Mark Zuckerberg wearing a light blue striped shirt, a patterned tie, and glasses. He is looking slightly to the right with a neutral expression. The background is a blurred office setting with vertical blinds.

OK, then... Tell me
what **functions** I
should use and
how.

PROBLEM #4: No Documentation

DOCUMENTATION **WITHOUT TYPES**

```
-spec cross(RowN, Pos, Len, Game) -> Game when  
  RowN      :: 1..5,  
  Pos       :: 1..5,  
  Len       :: 1..5,  
  Game      :: {Player, Player, Board},  
  Player    :: string(),  
  Board     :: {Row, Row, Row, Row, Row},  
  Row       :: [Stick, ...],  
  Stick     :: 0 | 1.
```

DOCUMENTATION **WITH** TYPES

```
-type row_num() :: 1..5.
-type pos()      :: 1..5.
-type len()      :: 1..5.
-type game()     :: {player(), player(), board()}.
-type player()   :: string().
-type board()    :: {row(), row(), row(), row(), row()}.
-type row()      :: [stick(), ...].
-type stick()    :: 0 | 1.

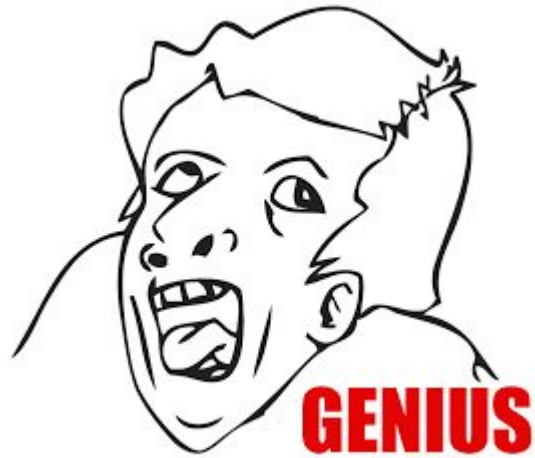
-spec cross(row_num(), pos(), len(), game()) -> game().
```


SHARING TYPES



SHARING TYPES

- Copy & Paste



SHARING TYPES

- ▶ Copy & Paste
- ▶ Header Files



SHARING TYPES

- ▶ Copy & Paste
- ▶ Header Files
- ▶ Export Types



SOURCE MODULE

```
-type row_num() :: 1..5.
-type pos()      :: 1..5.
-type len()      :: 1..5.
-type game()     :: {player(), player(), board()}.
-type player()   :: string().
-type board()    :: {row(), row(), row(), row(), row()}.
-type row()      :: [stick(), ...].
-type stick()    :: 0 | 1.

-export_type([row_num/0, pos/0, len/0, game/0, player/0,
              board/0, row/0, stick/0]).
```

OTHER MODULES

```
-spec cross( source_mod:row_num()  
            , source_mod:pos()  
            , source_mod:len()  
            , source_mod:game()  
            ) -> source_mod:game().
```

Who is source_mod?

“

The module where you define your shared types should only include the **Opaque Data Type** implementation.

- **Brujo**

NIM MODULE

```
-spec start(  
    nim_player:name(), nim_player:name()) -> nim_game:t().  
start(P1, P2) ->  
    nim_game:new(nim_player:new(P1), nim_player:new(P2),  
        nim_board:new()).  
  
-spec cross(  
    nim_board:row_num(), nim_board:pos(),  
    nim_board:len(), nim_game:t()) -> nim_game:t().  
cross(RowN, Pos, Len, Game) ->  
    nim_game:cross(RowN, Pos, Len, Game).
```

NIM_GAME MODULE

```
-opaque t() ::  
    {nim_player:t(), nim_player:t(), nim_board:t()}.  
-export_type([t/0]).  
  
-export([new/3, cross/4]).  
  
-spec new(  
    nim_player:t(), nim_player:t(), nim_board:t()) -> t().  
new(Current, Next, Board) ->  
    {Current, Next, Board}.
```

NIM_GAME MODULE

```
-spec cross( nim_board:row_num()  
            , nim_board:pos()  
            , nim_board:len()  
            , t()  
            ) -> t().
```

```
cross(RowN, Pos, Len, Game) ->  
    {Current, Next, Board} = Game,  
    NewBoard = nim_board:cross(RowN, Pos, Len, Board),  
    {Next, Current, NewBoard}.
```

NIM_BOARD MODULE

```
-opaque t() :: { nim_row:t()  
                  , nim_row:t()  
                  , nim_row:t()  
                  , nim_row:t()  
                  , nim_row:t()  
                  }.  
-export_type([t/0]).
```

NIM_BOARD MODULE

```
-spec new() -> t().
```

```
new() ->
```

```
{ nim_row:new(1)  
  , nim_row:new(2)  
  , nim_row:new(3)  
  , nim_row:new(4)  
  , nim_row:new(5)  
}.
```


NIM_BOARD MODULE

```
-spec cross(row_num(), pos(), len(), t()) -> t().  
cross(RowN, Pos, Len, Board) ->  
  Row = element(RowN, Board),  
  NewRow = nim_row:cross(Pos, Len, Row),  
  setelement(RowN, Board, NewRow).
```

ADT IMPLEMENTATION

- ▶ **Module Name** is **Type Name**
- ▶ **Opaque Type** **t/O**
 - ▶ Exported
- ▶ Constructor: **new/X**
- ▶ Accessors with **t()** in the spec

```
-module(my_adt).
```

```
-opaque t() :: ..type def...
```

```
-export_type([t/O]).
```

```
-export([new/X]).
```

```
-export([other_funs/Y, ...]).
```

```
-spec new(...) -> t().
```

```
new(Arguments) -> ..instance...
```

```
-spec other_fun(... t() ...) ->  
    other_thing() | t().
```

WHY?



PROBLEMS



PROBLEMS

~~1. Low Maintainability~~

~~2. Hard to Debug~~

~~3. No Encapsulation~~

~~4. No Documentation~~



EXTRA BENEFITS

- ▶ Makes **dialyzer** happy
- ▶ Makes **edoc** happy
- ▶ More **organized** code bases
- ▶ You can...
 - ▷ Switch **implementations**
 - ▷ Apply **common interfaces**
 - ▷ Sumo **behaviors**
 - ▷ Elixir **protocols**
 - ▷ Implement OOP **Heuristics**
 - ▷ Ask **Hernán** about them!



THE **KEY** CONCEPT

- ▶ `nim_player:t() != string()`
- ▶ `nim_row:t() != [1 | 0]`
- ▶ `nim_board:t() != {[1|0], ...}`
- ▶ `nim_game:t() != {string(), ...}`

You have **BETTER MODELS**

You've built

A BETTER SOFTWARE!



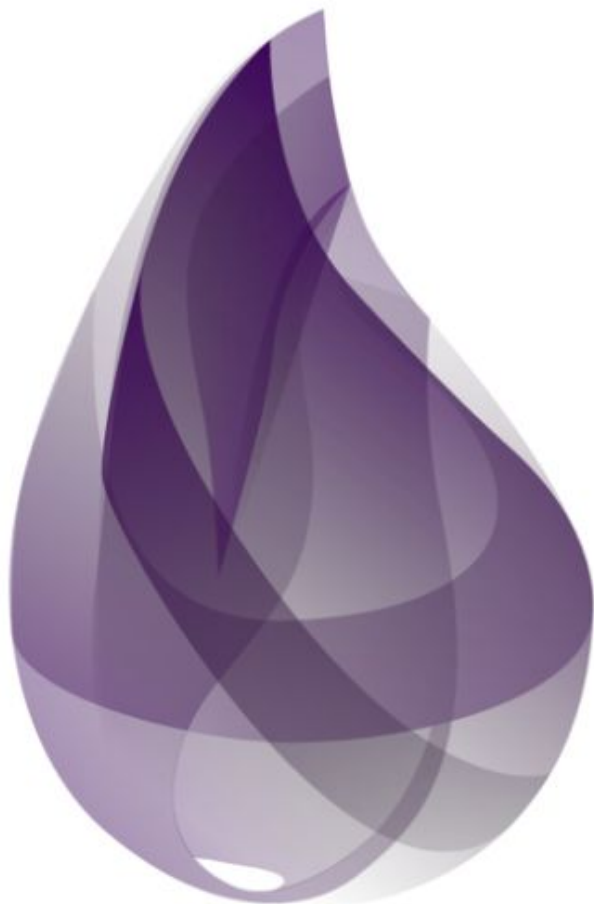
DOWNSIDES

- ▶ More **code**
 - ▷ More **modules**
 - ▷ More **functions**
- ▶ **Performance** concerns?
 - ▷ **Benchmark** your code
 - ▷ `-compile({inline,...`
- ▶ Less **“smart”** code
 - ▷ **Pattern Matching**
 - ▷ **List Comprehensions**



IN ELIXIR

- ▶ You can define types with:
 - ▷ **defstruct**
 - ▷ `@type t ::`
`%__MODULE__{...}`
- ▶ Then you can use your types like **MyADT.t**
- ▶ And you can implement **protocols**



FURTHER READING

- ▶ [Heuristics for OOP](#)
(by Hernán Wilkinson)
- ▶ [Heuristics for Erlang](#)
(by Brujo Benavides)
- ▶ [Erlang and Code Style](#)
(by Jesper L. Andersen)
- ▶ [Everything](#)
(by erlang-questions /
Slack / etc.)



THANK YOU

Any questions?

elbrujohalcon@inaka.net

Inaka.net / www.erlang-solutions.com

@elbrujohalcon

inaka

Erlang
SOLUTIONS

