Erlang

yet another language

Sanity check

```
> X = 3.
```

Sanity check

```
> X = 3.
> X.
3
```

Sanity check

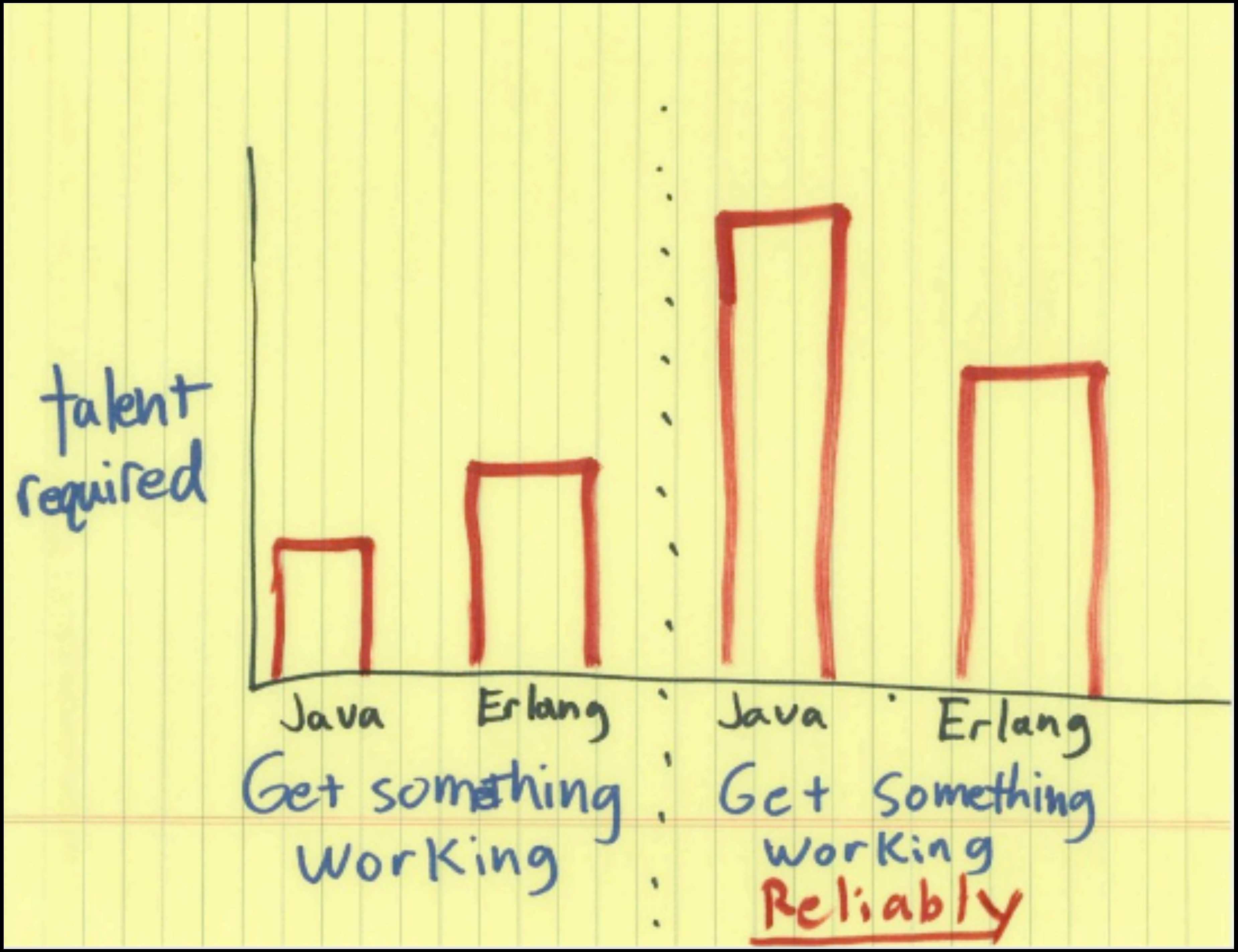
```
> X = 3.
> X.
3
> X = X+1.
```

Sanity check

```
> X = 3.
> X.
3
> X = X+1.
** exception error: no match of
right hand side value 2
```

Why Erlang?

‘cause it makes hard things easier



http://www.infoq.com/presentations/katz-couchdb-and-me

Syntax

Commas (,) separate expressions
Semicolons (;) end clauses
Periods (.) end everything

I like that; it's somewhat like english, I believe.

Basics

Variables

Var

JavaStyle

Not_So_Much_Java_Style

Yeah_its_weird_at_first

Atoms

ok

false

nil

else

this_is_an_atom

`back-tick for spaced atoms`

nop

Tuples

```
{tuple}
{index, 2}
{nested, {data, 1}, {structures}}
```

Lists

[]

[1,2,3]

[1, 2, 3, four, 5.0]

[1, 2] ++ [3, 4]

List Comprehensions

```
[X || X <- [1,2,3]]

[X || X <- [1,2,3], X < 2]
```

Functions

```
my_function(X) -> ok.

Anonymous = fun(X) -> X * X end.
```

Pattern Matching

```
3 = 3

Value = {4, 2}

{_, Value} = {4, 2}

[Head|Tail] = [1, 2, 3, 4]

[First, Second|Remaining] = [1, 2, 3, 4]
```

Pattern Matching

```
{Width, Height} = {5.2, 7}

{ok, Value} = {ok, [1, 2]}

{ok, Content} = file:open("filename", [read])
```

Guards

```
max(X, Y) when X > Y -> X;

max(X, Y) -> Y.
```

Modules, Arity

```
-module(my_module).
-export(start/0, loop/2).

-import(other_module).

lists:search(5, List).
```

Built in Functions

(aka BIF's)

atom_to_binary/2

atom_to_list/1

is_list/1

list_to_existing_atom/1

spawn/1

!/2

+/1

...

Non-features

Booleans

true

false

ok

else

null

nil

Hash

```
[{key, 5}, {another_key, 9}]

[{key, {sub, 5}, {another_key, {nested}}]
```

Strings

“string”
“another string”
“etc..”

```
do_something(This, That) ->
    Var = calculate(That),
    This + Var.
```

example.erl

```erlang
-module(example).
-export(do_something/2, start/0).

do_something(This, That) ->
    Var = calculate(That),
    This + Var.

start() ->
    ok.
```

Control Flow

```
length(List) ->
    if
        List == [] ->
            0;
        true ->
            1+ length(tail(List))
    end.
```

```
length(List) ->
    case List of
        [] ->
            0;
        _ ->
            1+ length(tail(List))
    end.
```

```
length(List) ->
    case List of
        [] ->
            0;
        [H|Tail] ->
            1+ length(Tail)
    end.
```

```
length([]) ->
    0;
length([_|Tail]) ->
    1 + length(Tail).
```

Concurrency

Shared memory

Message Passing

- Asynchronous send
- Synchronous receive
- Each process has its “mailbox”

```erlang
-module(example).
-compile(export_all).

start() ->
    spawn(fun() -> loop([]) end).

loop(X) ->
    receive
        exit ->
            ok;
        Any ->
            io:format("Received:~p~n", [Any]),
            loop(X)
    end.
```

```
> Pid = a:start().
<0.37.0>

> Pid ! "hello world!".
Received:"hello world!"
"hello world!"

> Pid ! {pi, 3.1417}.
Received:{pi,3.1417}
{pi,3.1417}

> Pid ! exit.
exit

> Pid ! "are you there?".
"are you there?"

> erlang:processes().
% not there
```

prime time!

Also...

- Mnesia
  - real-time, distributed database
- Supervisors
  - error “trapping”, fault-tolerance
- -behavior()
  - code reuse; “functional abstract classes”

Also...

- Hot code swap
  - fault tolerance; high availability
- Libraries
- Reliability

"Erlang has been running massive systems for 20 years. Erlang-powered phone switches have been running with nine nines availability — only 31ms downtime per year"