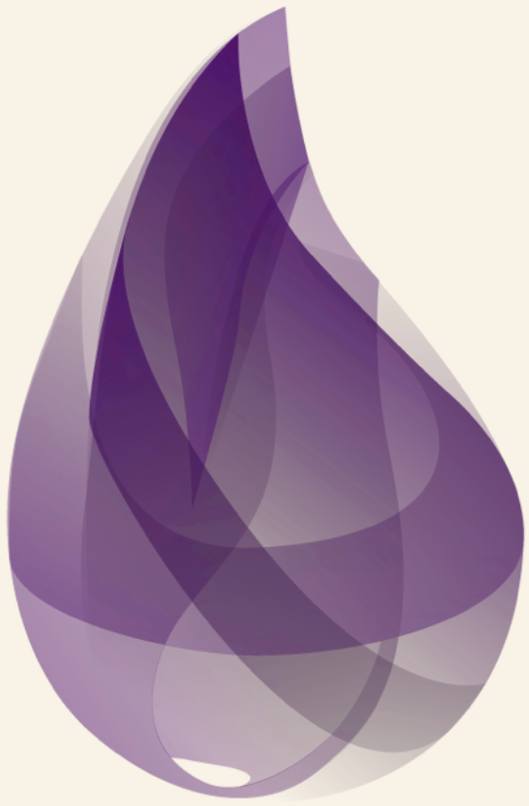


elixir

@elixirlang / elixir-lang.org

Wednesday, March 7, 2012

My talk today is about a new programming language on top of the Erlang VM called Elixir. I am not going to spend a lot of time talking about the syntax and how to X or Y, you are better off with those if you read the Getting Started guide. Instead, I am better off to make an impression if we discuss the reasons and goals behind Elixir.

Why?

plataformatec

tecnologia e engenharia de software

@plataformatec / plataformatec.com.br

Wednesday, March 7, 2012

I come from a web development background, I am co-founder of a consultancy company in Brazil, and we have huge clients, mainly from publishing and media companies. I am working with web for 8 years already.

The web is CHANGING

The web changed a lot in the last 10 years. And if we look at the next 10 years ahead, it will continue changing. And I believe those changes make a strong bet for Elixir and the Erlang ecosystem.

case #1

SPDY

Amazon.com: Online Shoppi

www.amazon.com

amazon.com

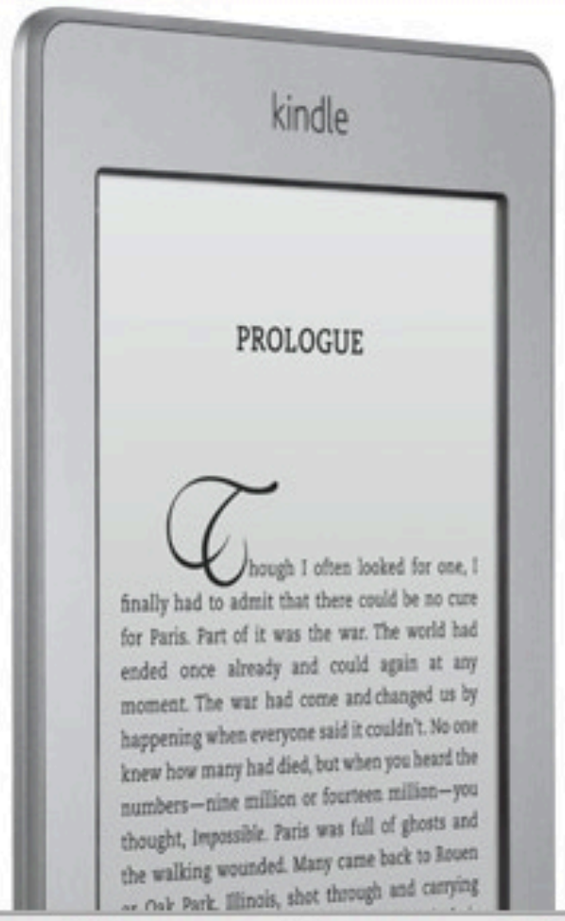
Hello. Sign in to get personalized recommendations. New customer? Start here.

Your Amazon.com | Today's Deals | Gifts & Wish Lists | Gift Cards

Shop All Departments

Search All Departments

- Unlimited Instant Videos
- MP3s & Cloud Player — 18 million songs, play anywhere
- Amazon Cloud Drive — 5 GB of free storage
- Kindle
- Appstore for Android — Get Tracing ABC for free today
- Digital Games & Software
- Audible Audiobooks
- Books
- Movies, Music & Games
- Electronics & Computers
- Home, Garden & Tools
- Grocery, Health & Beauty



kindle touch

Now shipping internationally

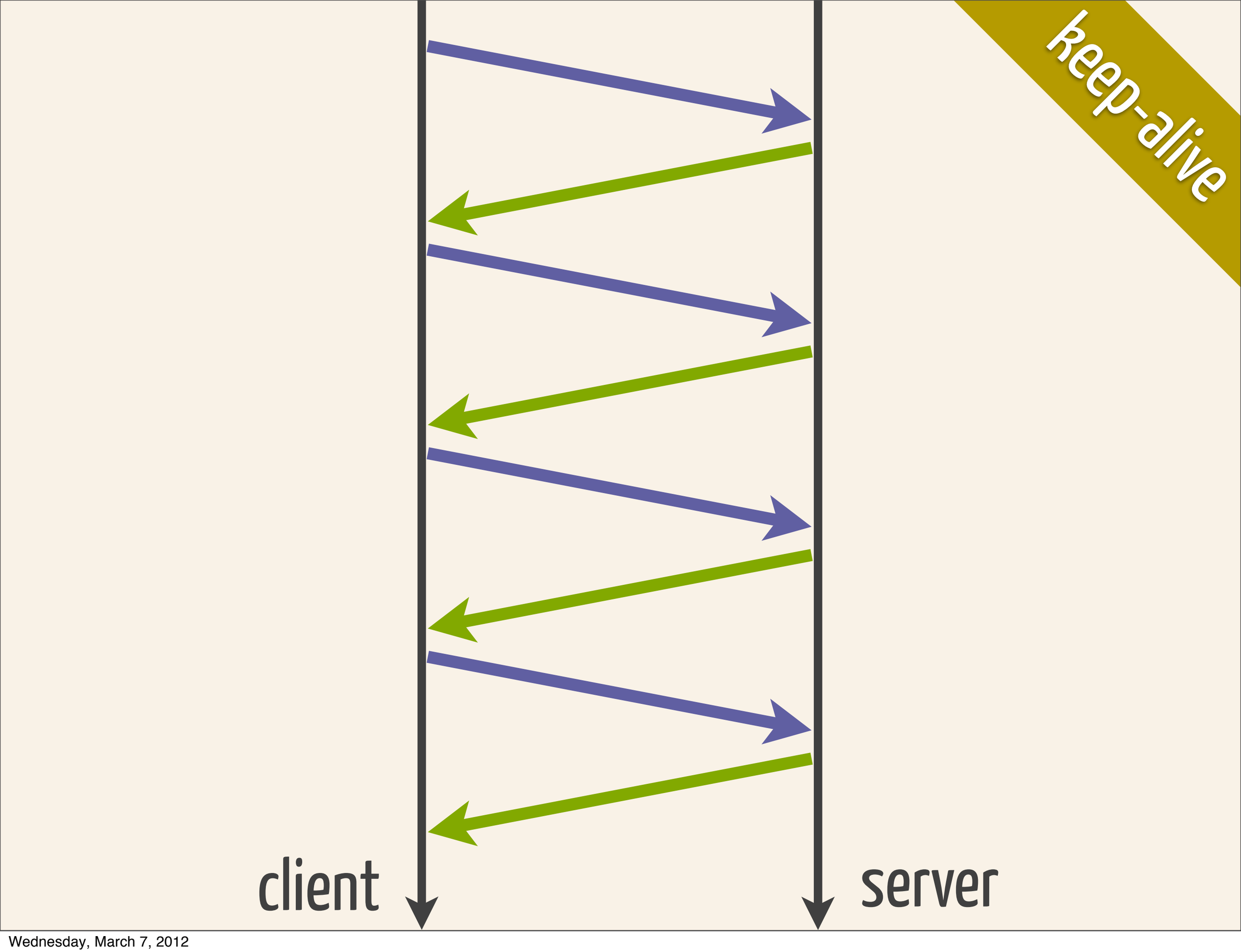
$139 Order now

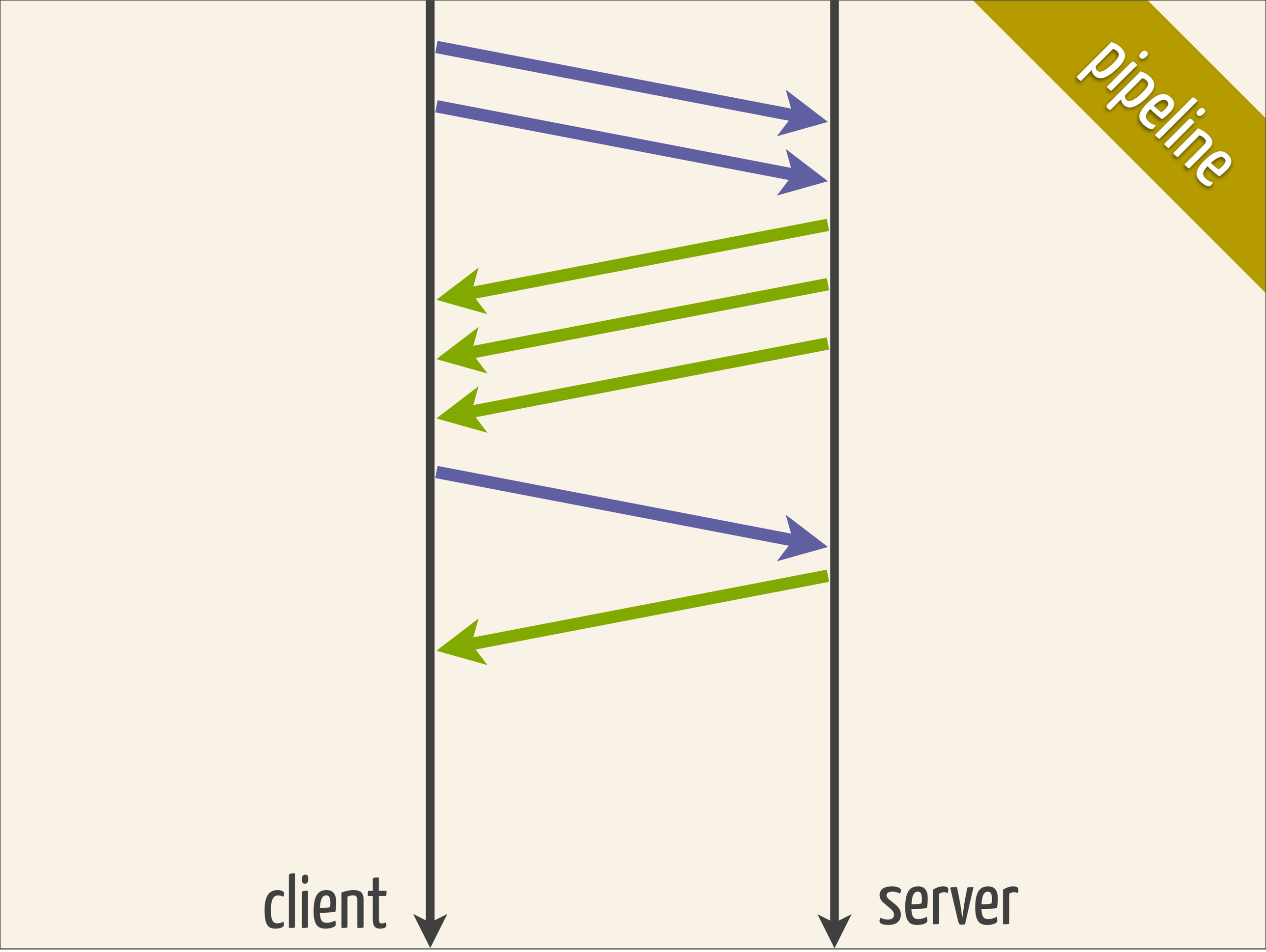
Elements | Resources | Network | Scripts | Timeline | Profiles | Audits | Console

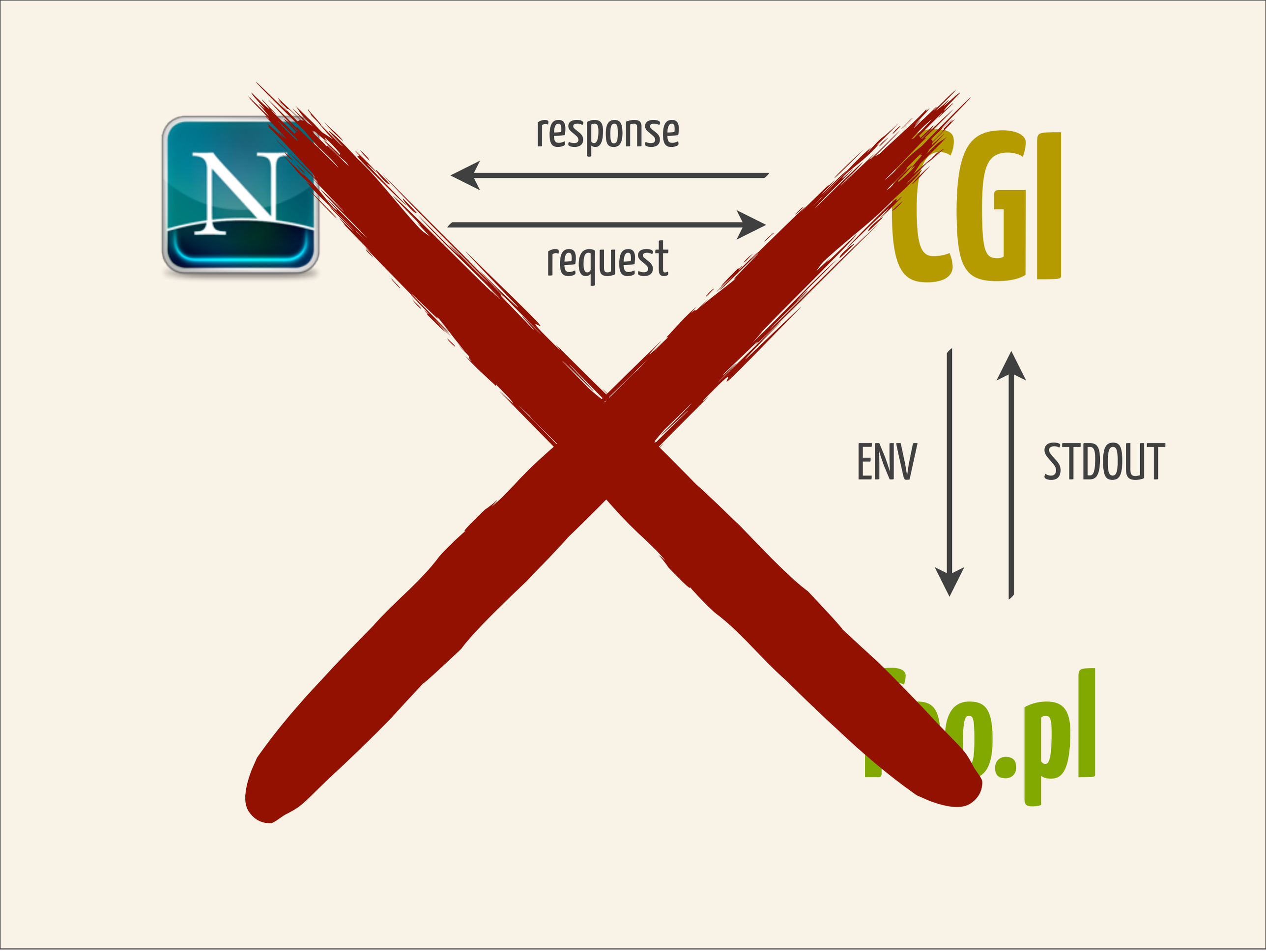
Name / Path	Method	Status / Text	Type	Initiator	Size / Content	Time / Latency	Timeline
fls-na.amazon.com/1[illegible]-jserr/1/OP	GET	Bad Request	text/plain	Script	0B	560ms	
Gateway /uedata/18[illegible]055-0836402	GET	200 OK	image/gif	http://www.amazon.c Script	704B 43B	574ms 573ms	
DFS-BR-0[illegible]azon-Free-Shipping s0.2mdn.net/2191096	GET	200 OK	applicatio...	Other	26.88KB 26.56KB	243ms 151ms	

64 requests | 247.20KB transferred

All | Documents | Stylesheets | Images | Scripts | XHR | Fonts | WebSockets | Other


keep-alive
client
server


pipeline
client
server



This abstraction no longer works. Well, it could be made to work, but it would be unnecessarily complex.

case #2

Smarter clients


HTML

Paradigm shift

Long-running connections

Binary serialization protocols

case #3

Multi-core

Wednesday, March 7, 2012

I want to be able to use all core on my machines without a need to start many, many processes. Most web applications today are single-process, few languages get concurrency properly.

Which technology is well-known for handling many long-running, concurrent connections?

Erlang has proven time after time that it is a good solution for these problems.

What?

Now that we know “why?” I want to explain what we want to achieve with Elixir.

I am part of the Rails Core Team, one of the biggest web frameworks out there. Rails main focus is not on performance, but enhanced developer productivity. There are many aspects of Ruby, an extremely dynamic language, that make Rails possible, how can we achieve that for the Erlang VM?

goal #1

Productivity

goal #2

Extensibility

This is a very thin line because sometimes you may need to do trade-offs between extensibility and robustness.

Wednesday, March 7, 2012

The Erlang Environment has accumulated many tools and practices throughout time. Robust, fault-tolerant applications with hot code swap are characteristics we plain to maintain.

DISTRIBUTED FAULT-TOLERANT APPLICATIONS WITH HOT-CODE SWAPPING

goal #3

Compatibility

How?

How Elixir helps us achieve those goals?

feat #1

Homoiconicity

atom

```
is_atom(:foo)
```

```
{ :is_atom, 1, [:foo] }
```

function

line

args

```
1 + 2
```

```
{ :+, 1, [1,2] }
```

function

line

args

elixir

```
defmacro unless(expr, opts) do
  quote do
    if(!unquote(expr), unquote(opts))
  end
end
```

```
unless(true, do: exit())
```

DOMAIN SPECIFIC LANGUAGES

It allows us to create constructs specific to the domain we are tackling, allowing libraries/frameworks to create higher abstractions for us so we can enjoy higher productivity.

elixir

```
respond_to(request) do
html:
  response.render("template")
json:
  response.ok(to_json(record))
end
```

1. Inspect Accept header

```
Accept: text/html,application/json;q=0.9,*/*;
```

2. Detect formats available

```
respond to html or json
```

3. Invoke the negotiated format

```
response.render("template")
```

feat #2

Executable definition

Makes a good mix with macros in order to generate specific modules.

erlang

```
-module(foo).

sum(A, B) -> A + B.
```

```erlang
-module(foo).
io:format("hello").
sum(A, B) -> A + B.
```

elixir

```
defmodule Foo do
  IO.puts "hello"

  def sum(a, b) do
    a + b
  end
end
```

Wednesday, March 7, 2012

And has many will generate code that will make associating posts and comments easier. There are many Erlang DB mappers that implements similar constructs using callbacks or attributes but it usually require playing with the Erlang code loader or parse transforms. Since in Elixir it is supported by the language, the implementation is much simpler. Remember less code, less bugs!

feat #3

Protocols

Wednesday, March 7, 2012

Protocols are a mechanism to implement polymorphism.

erlang

```
-module(json).
to_json(Item) when is_list(Item) ->
to_json(Item) when is_binary(Item) ->
to_json(Item) when is_number(Item) ->
```

elixir

```
defprotocol JSON, [
  to_json(item)
]
```

```elixir
defimpl JSON, for: List do
  # ...
end

defimpl JSON, for: Binary do
  # ...
end

defimpl JSON, for: Number do
  # ...
end
```

elixir

```
defimpl JSON, for: Array do
  # ...
end
```

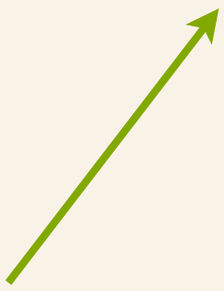
feat #4

Compatibility

There is no conversion cost for calling Erlang from Elixir and vice-versa

erlang

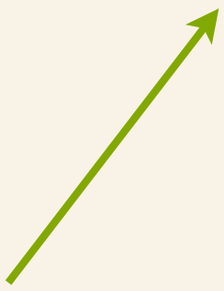
atom



```
lists:flatten([1,2,3])
```

elixir

atom



```
:lists.flatten([1,2,3])
```

elixir

```
Erlang.lists.flatten([1,2,3])
```

elixir

```
require Erlang.lists, as: List
List.flatten([1,2,3])
```

You can write Elixir code that when compiled has no dependency at all on Elixir

feat #5

References

feat #6

Dynamic Records

When?

Today: Release of the website

March: Release v0.9 of Elixir

May: Beta release of Dynamo

A modern approach to programming for the Erlang VM.

```
defprotocol String::Insp
  only: [BitString, List,

defimpl String::Inspect,
  def inspect(false), do:
  def inspect(true),  do:
  def inspect(nil),   do:
  def inspect(:""),   do:

  def inspect(atom) do
```

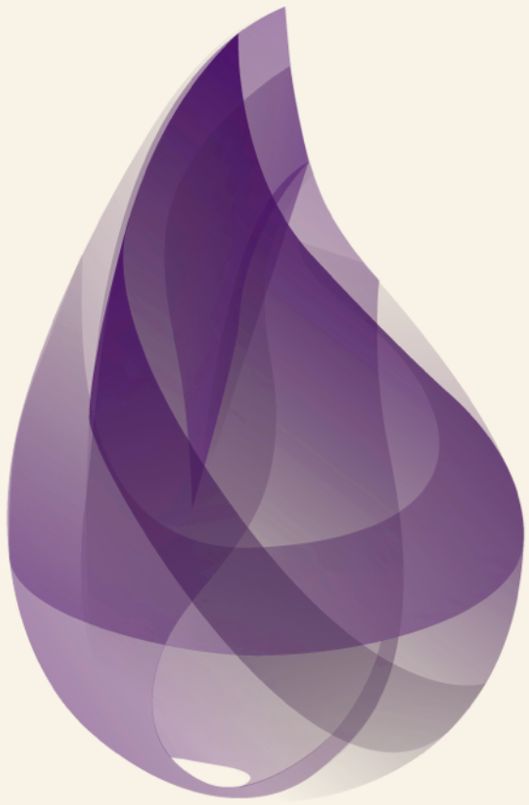
WHAT IS ELIXIR?

Elixir is a programming language built on top of the Erlang VM. As Erlang, it is a functional language built to support distributed, fault-tolerant, non-stop applications with hot code swapping.

Elixir is also dynamic typed but, differently from Erlang, it is also homoiconic, allowing meta-programming via macros. Elixir also supports polymorphism via protocols (similar to Clojure's), dynamic records and provides a reference mechanism.

Finally, Elixir and Erlang share the same bytecode and data types. This means you can invoke Erlang code from Elixir (and vice-versa) without any conversion or performance hit. This allows a developer to mix the expressiveness of Elixir with the robustness and performance of Erlang.

If you want to learn more, you can either check the getting started guide or the repository on Github.



elixir

@elixirlang / elixir-lang.org