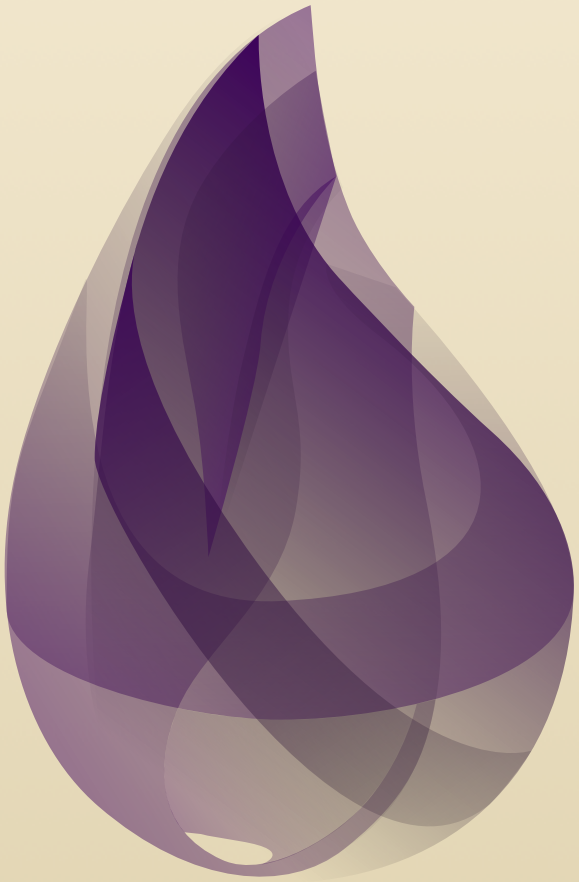
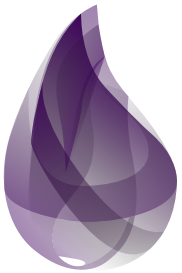
The Pragmatic Bookshelf



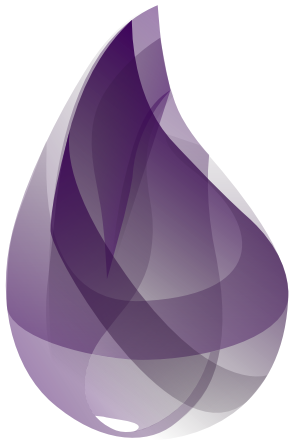
Introducing Elixir

Functional |> Concurrent |> Pragmatic |> Fun

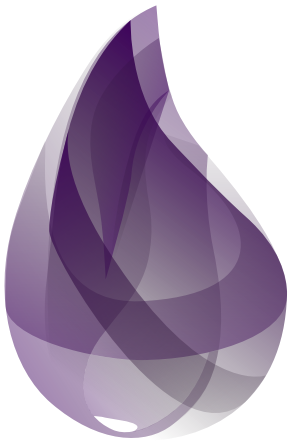
@/+ pragdave



Pragmatic Bookshelf



elixir

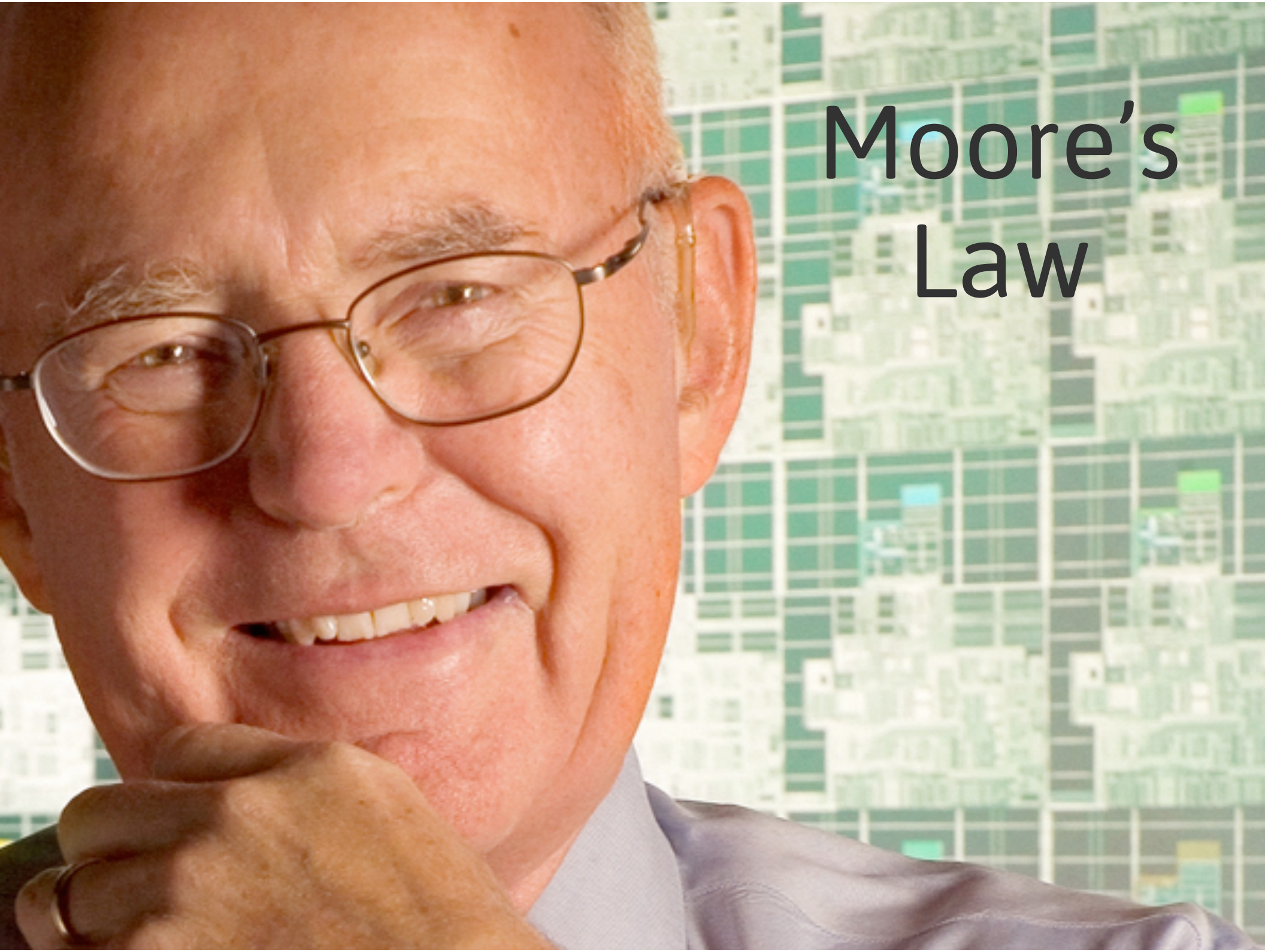


elixir

The future is **functional**

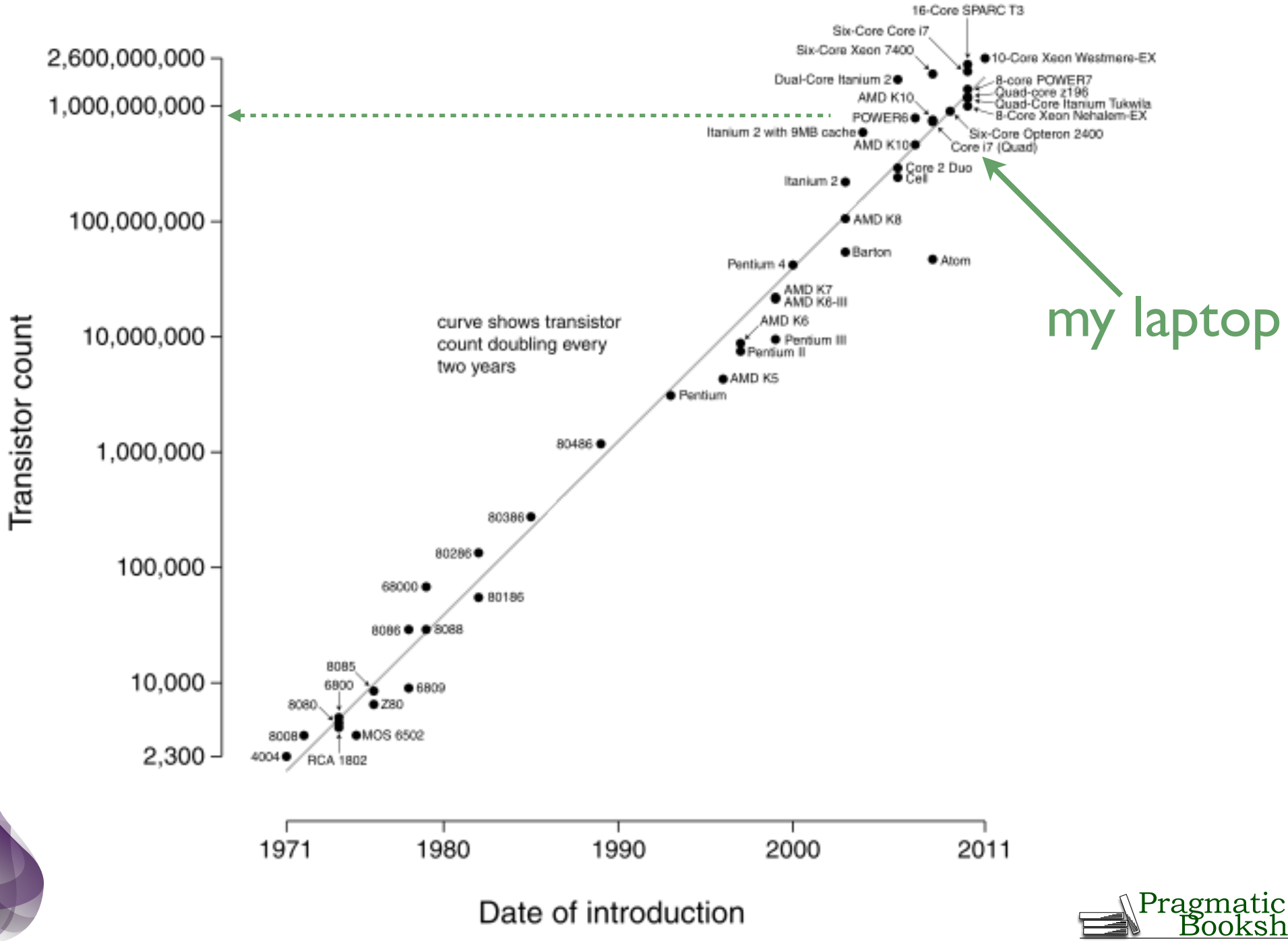
The future is **concurrent**

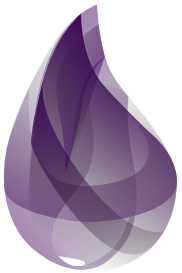
Moore's Law


Moore's
Law

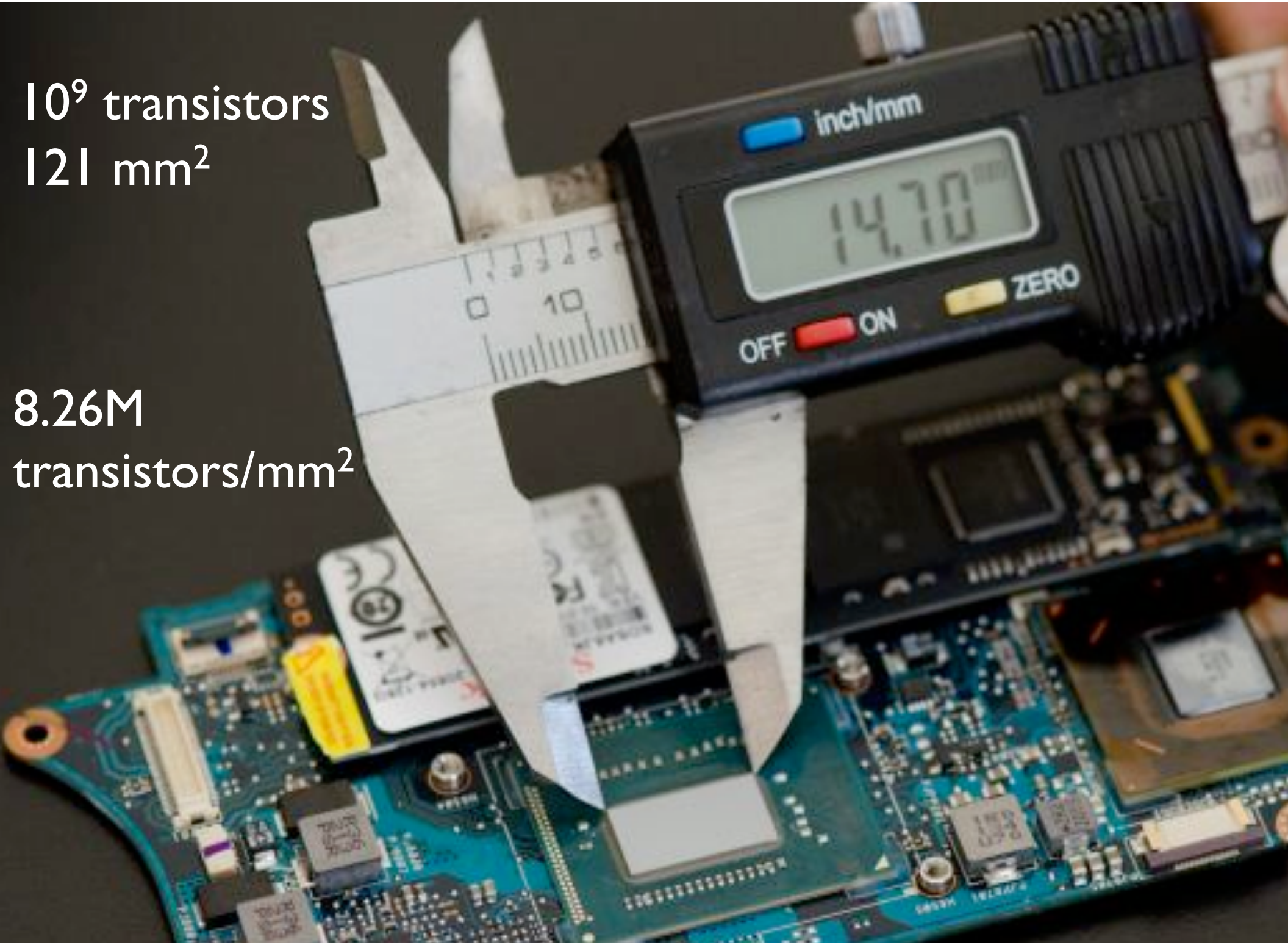
Microprocessor Transistor Counts 1971-2011 & Moore's Law

http://en.wikipedia.org/wiki/File:Transistor_Count_and_Moore%27s_Law_-_2011.svg





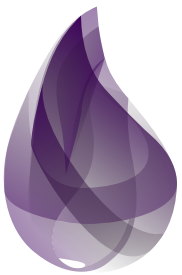
Pragmatic Bookshelf

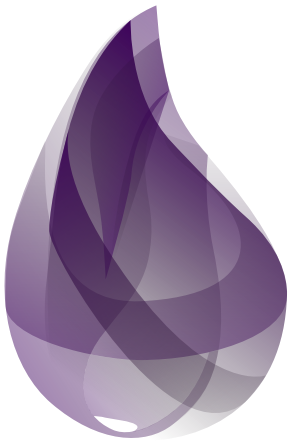


10^9 transistors
121 mm^2

8.26M
transistors/mm^2

8.26M
transistors/mm^2

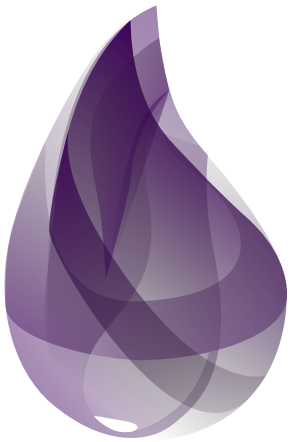




elixir

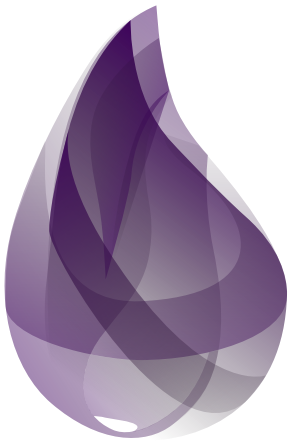
The future is **functional**

The future is **concurrent**



elixir

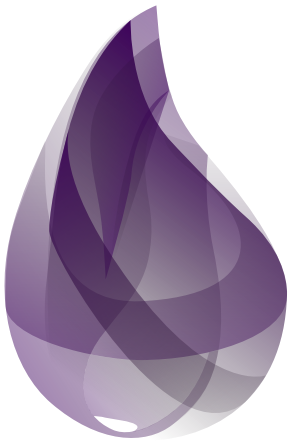
- Immutable data
- Pattern Matching
- Hygenic Macros
- blah blah...



elixir

Because all programming is transformation

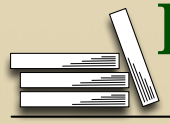
- **data**
- **code**



elixir

transformation

Let's Code



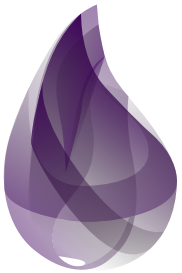
Pragmatic Bookshelf

```elixir
defmodule Dictionary do

  def signature_of(word) do
    word |> to_char_list |> Enum.sort |> to_string
  end

end
```

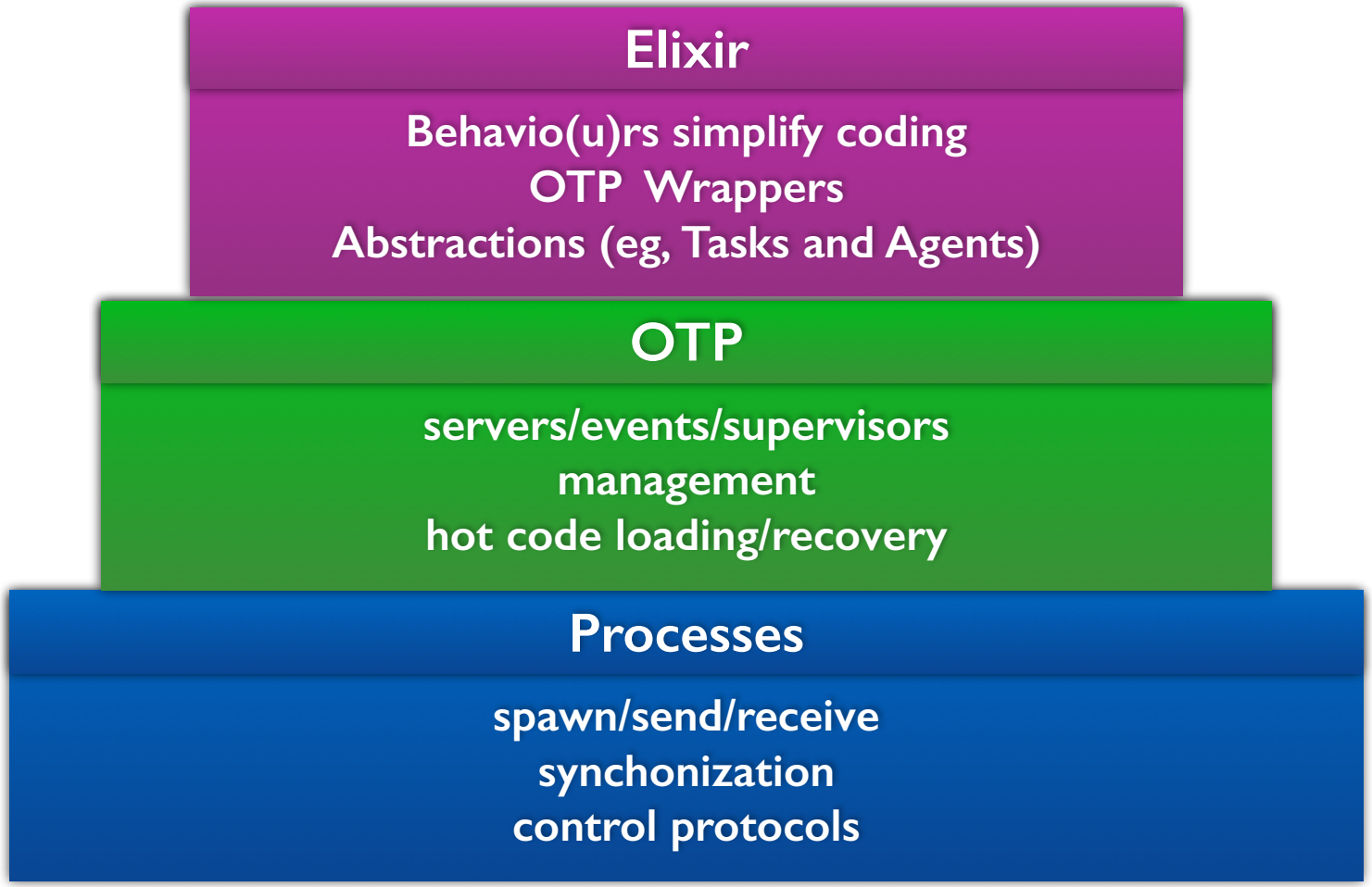
```
defmodule Dictionary do

  def signature_of(word) do
    word |> to_char_list |> Enum.sort |> to_string
  end

end

iex> Dictionary.signature_of “seat"
"aest"

iex> Dictionary.signature_of “teas"
"aest"
```

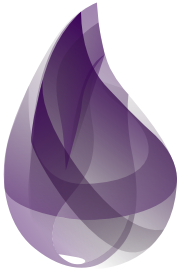
Transformation

- The |> operator makes transformation explicit
- OO focus: information hiding
- FP focus: information transformation



Concurrency





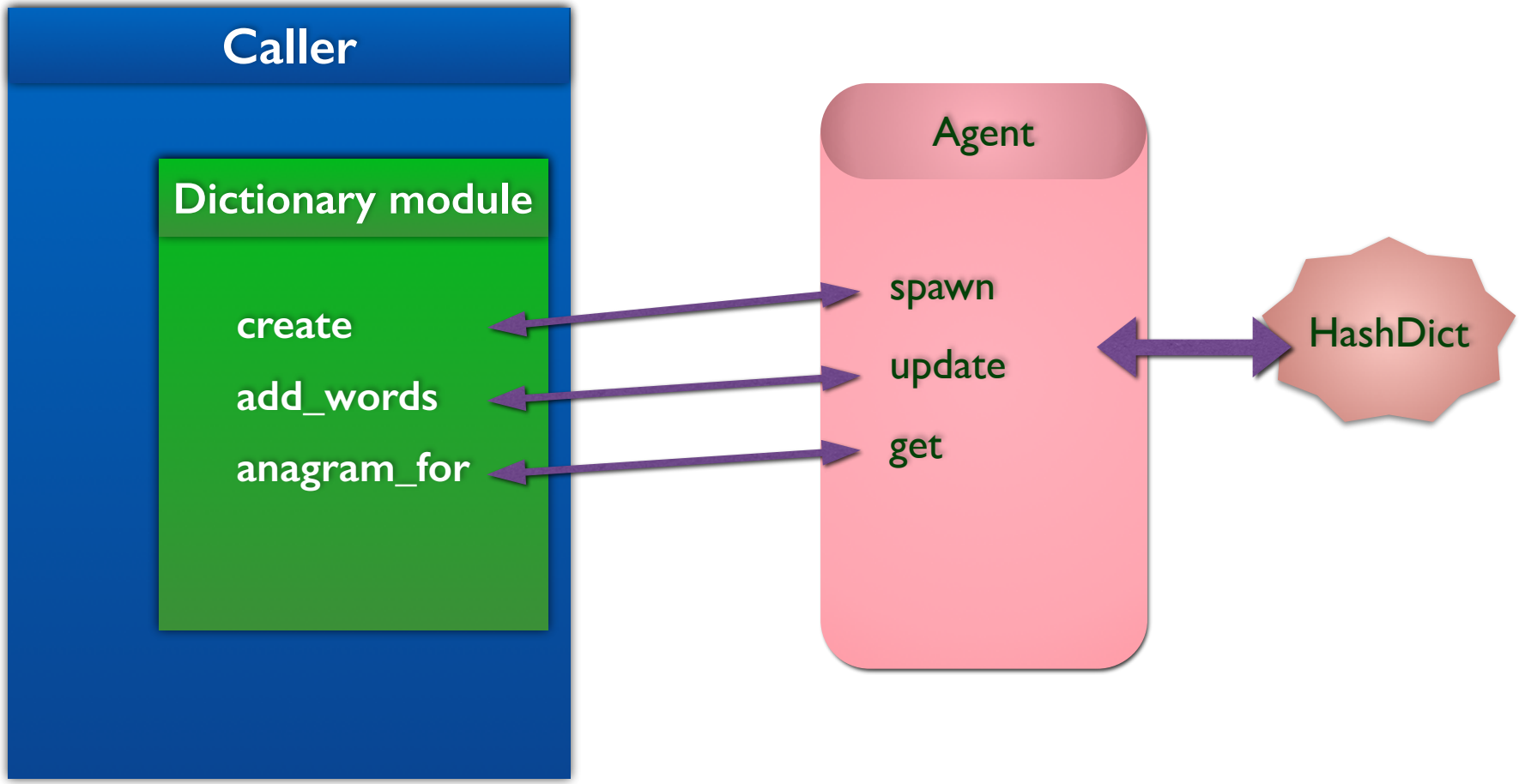
Pragmatic Bookshelf

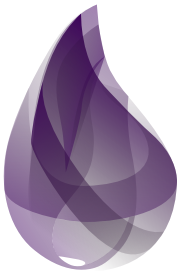
Agents

```
Agent.start_link(fn -> initial_state end, name: SomeName)

    Agent.update(SomeName, fn state -> new_state end)

    Agent.get(SomeName, fn state -> return_value end)
```

All functions run in the agent's process

Concurrency





```elixir
defmodule Dictionary do

  def signature_of(word) do
    word |> to_char_list |> Enum.sort |> to_string
  end

  def start_link do
    Agent.start_link(fn -> HashDict.new end, name: __MODULE__)
  end

  def add_some_words(words) do
    Agent.update(__MODULE__, &add_words(&1, words))
  end

  def anagrams_of(word) do
    Agent.get(__MODULE__, fn dict -> Dict.get(dict, signature_of(word)) end)
  end

  defp add_words(dict, words) do
    Enum.reduce words, dict,
      fn {signature, word}, dict ->
        Dict.update(dict, signature, [word], fn words -> [word|words] end)
      end
  end

end
```

Tasks

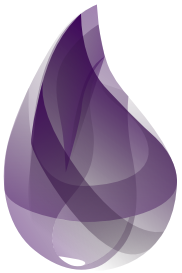
```elixir
Task.async(fn -> computation end, name: SomeName)
```

```elixir
Task.await(SomeName, timeout)
```

```elixir
defmodule WordlistLoader do
  def load_from(file_names) do
    file_names
    |> Stream.map(fn name -> Task.async(fn -> load_task(name) end) end)
    |> Enum.map(&Task.await/1)
  end

  defp load_task(file_name) do
    file_name
    |> File.open!
    |> IO.stream(:line)
    |> Stream.map(&String.strip/1)
    |> Enum.map(fn word -> { Dictionary.signature_of(word), word } end)
    |> Dictionary.add_some_words
  end
end

Dictionary.start_link

1..4
|> Enum.map(&"words/list#{&1}")
|> WordlistLoader.load_from_files

IO.inspect Dictionary.anagrams_of "crate"
```

Why Elixir?

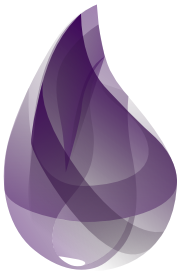
It doesn't have to be Elixir



Why Elixir?

But you do need to learn

functional programming.

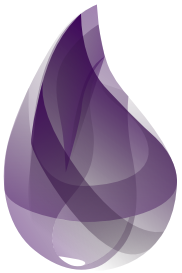


Why Elixir?

But you do need to learn

functional programming,

concurrent programming.



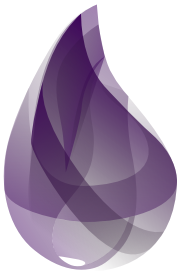
Why Elixir?

But you do need to learn

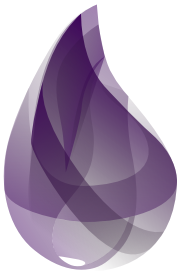
functional programming.

concurrent programming, and

distributed programming.



So why not have fun at the same time?



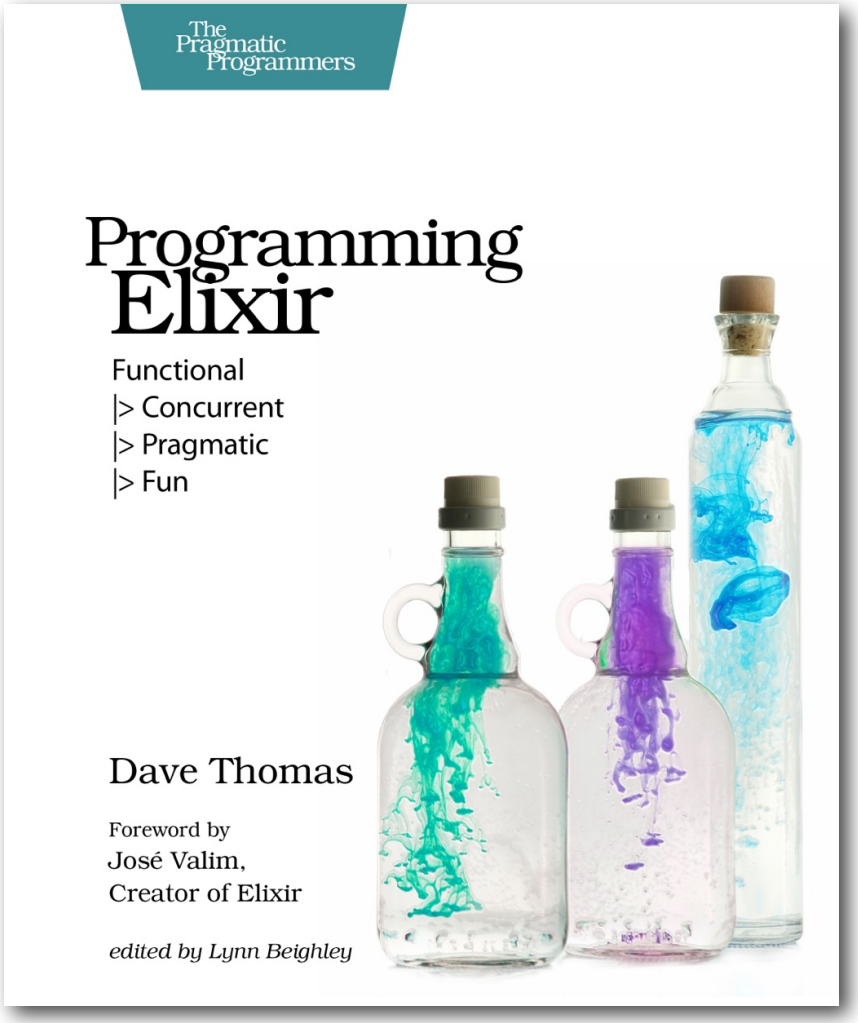
Because STICKERS!

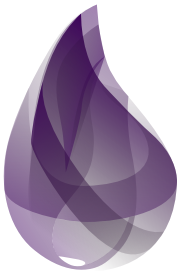
Pragmatic Bookshelf

elixir-lang.org

pragprog.com/titles/elixir

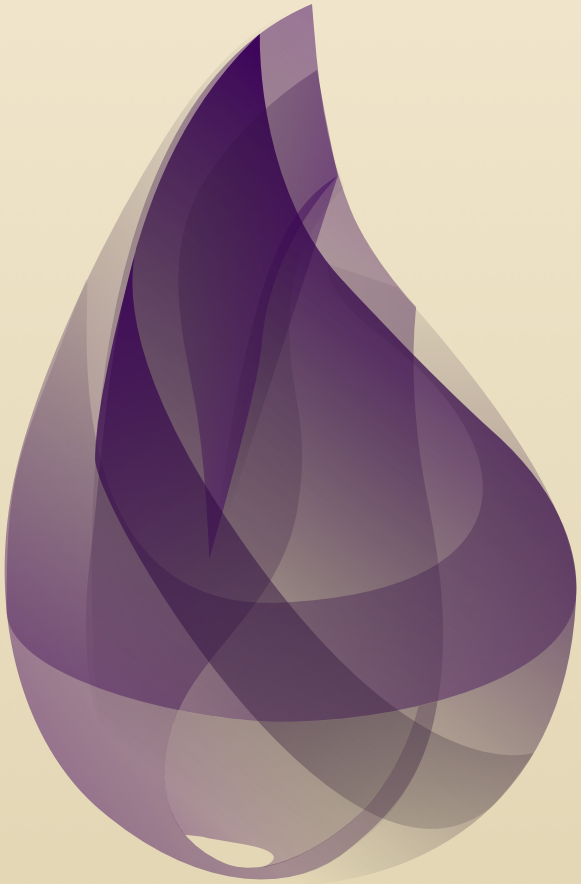
@/#pragdave





Pragmatic Bookshelf

The Pragmatic Bookshelf



Introducing Elixir

Functional |> Concurrent |> Pragmatic |> Fun

@/+ pragdave