elixir

|> a crisp

|> new language

|> for the Erlang VM

presented by
Bryan Hunter
CTO Firefly Logic

Twitter
@bryan_hunter
@fireflylogic
@nashrb

#elixir

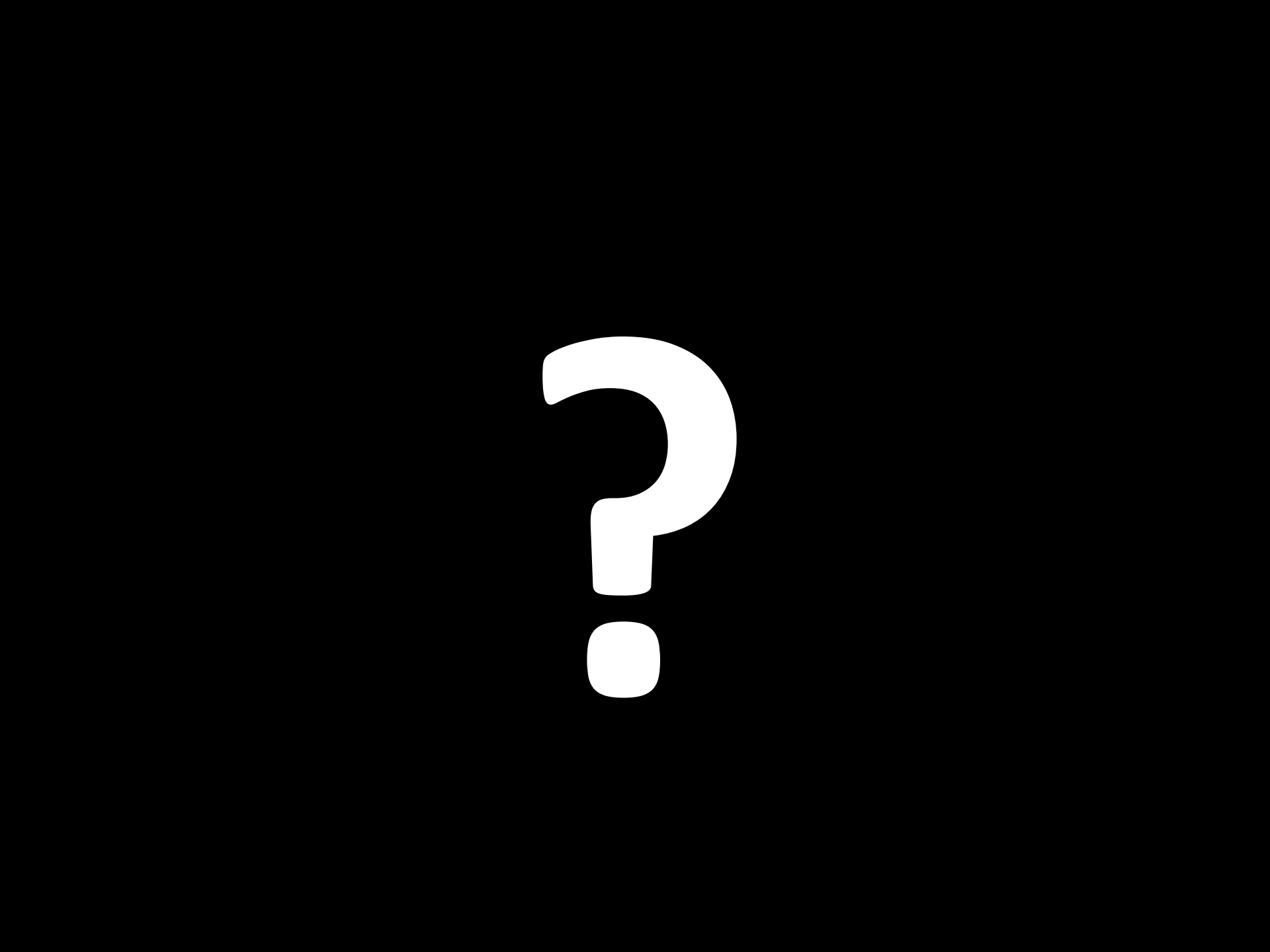
FIREFLY LOGIC

elixir

ERLANG

Abstract

Elixir is a modern and **expressive functional** programming language that delivers the **productivity** and "**developer joy**" of Ruby and all of the **raw power** of Erlang. Elixir was designed to take the **best language qualities** of Ruby, Clojure and Erlang. Elixir code runs on the **Erlang VM** and compiles to the same binary format (BEAM file) as Erlang, so it has zero-penalty access to Erlang libraries including **OTP**. Just like Erlang, it supports **hot code loading** and scales beautifully (**scales-up** to many cores, **scales-out** to many machines). What does it **look** like? What are the **pieces** and the **tools**? How do I **get started** with Elixir? Sound exciting? Good! Come join the **fun**!



Norway

2007

6 weeks

12 architects

20,000,000 LOC

1 big mess

5 minute chat

4 years

0 seconds downtime

Life changed

Google “NDC Erlang jumpstart”

NDC 2013

4

https://vimeo.com/68327403#t=22m35s

22:35

HD

Life changed

Hi.

Hey! You're….
that guy!

Um… yes, that Dave Thomas guy.

Ah! You’re one of the original signatories of the Agile Manifesto, right?

Yep

Dude! You’re the guy
who popularized Ruby

Boo-ya!

They call you the “king maker” of Ruby

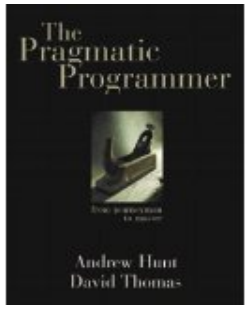
Agile and Ruby have had a huge influence on all devs.

Wait… what are you doing in Bryan’s talk?


elixir

“Programmers not on board with functional programming in 5 years will be maintenance programmers.”

- Dave Thomas @pragdave
GOTO Chicago 2014

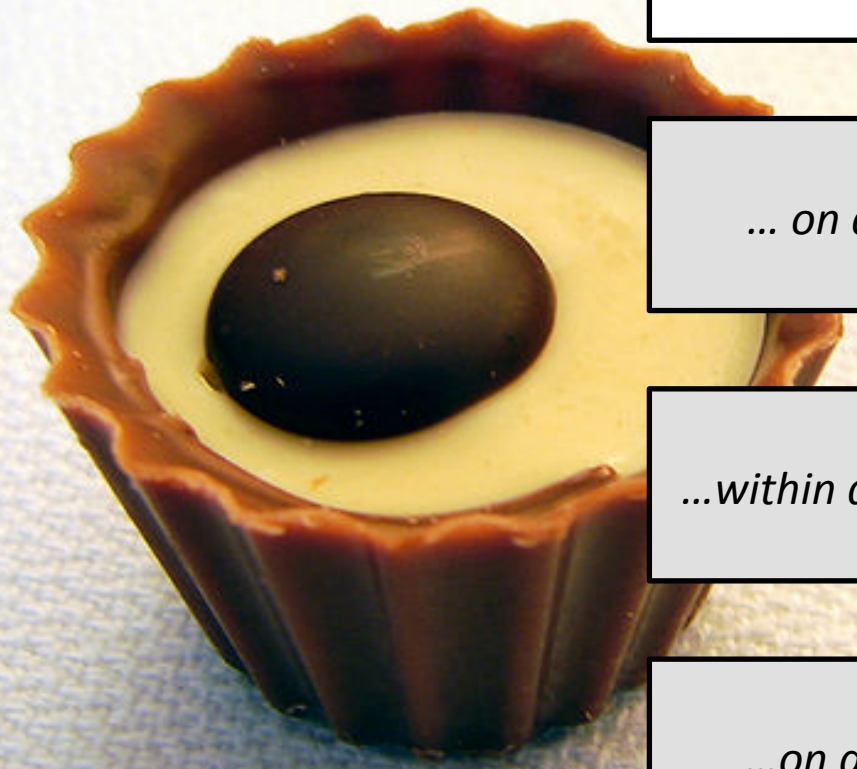


“The ‘*When*’ is pretty much now.”

-Robert Martin

“Functional Programming What? Why? When?”
NDC London 2013

What is your language's sweet spot?



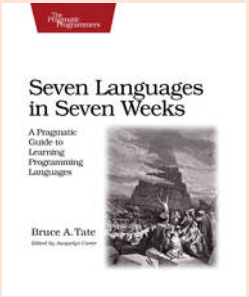
Doing Awesome-Thing-X

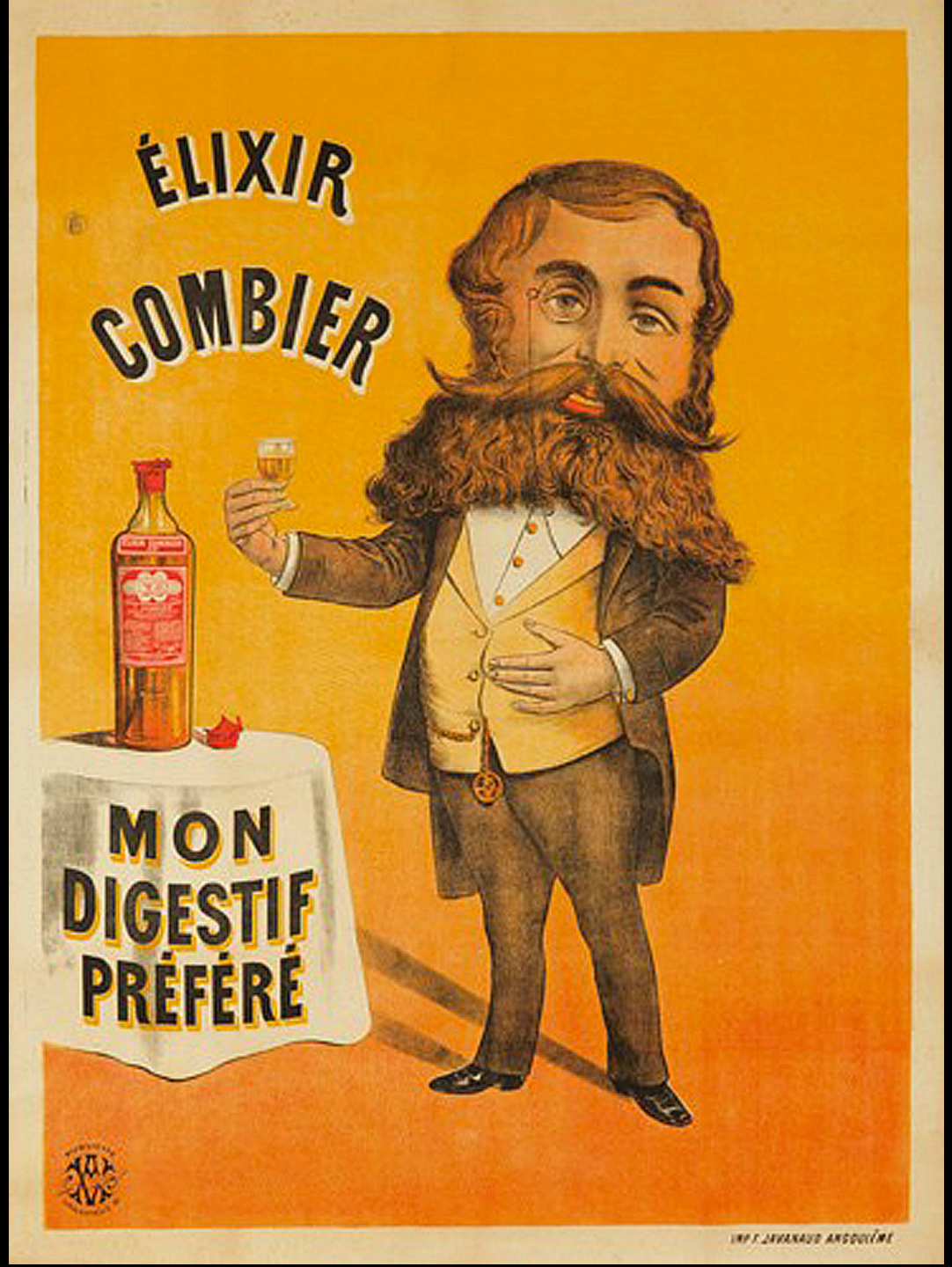
… on a single thread …

…within a single OS process…

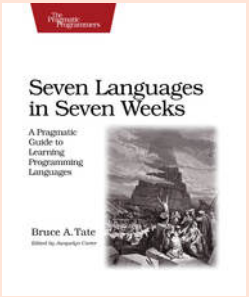
…on a single computer

History of Elixir





History of Elixir




ERLANG VM
TRUE
ELIXIR
TINCTURE OF
+ ERLANG + RUBY
+ CLOJURE + F#
LANGUAGE
INVIGORATING
O T
P

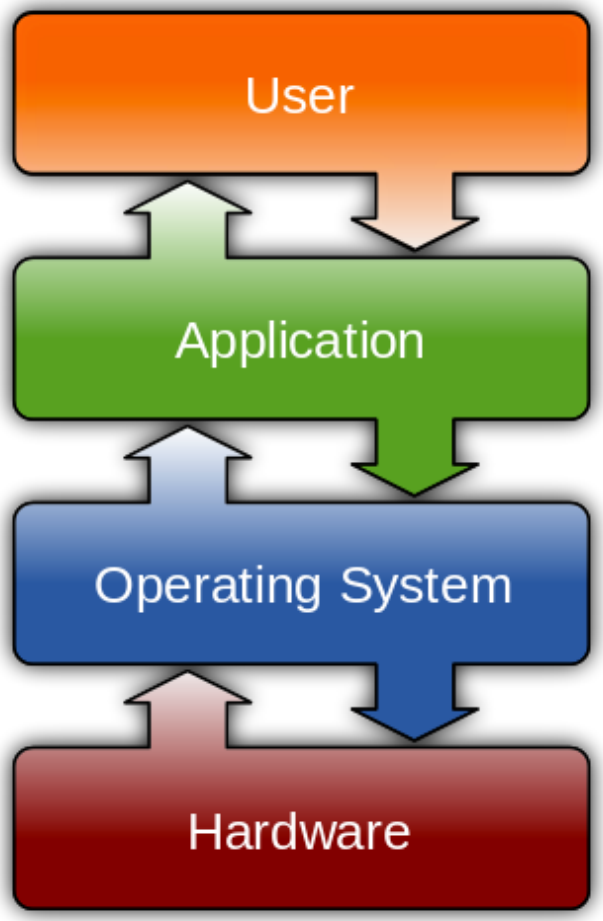

Fault-tolerant
Concurrent
Distributed
1
2
3
4
5
6
7
8
9
*
0
#
ERLANG

dev stack

250+ years

OS (Windows, Linux, Mac, None)

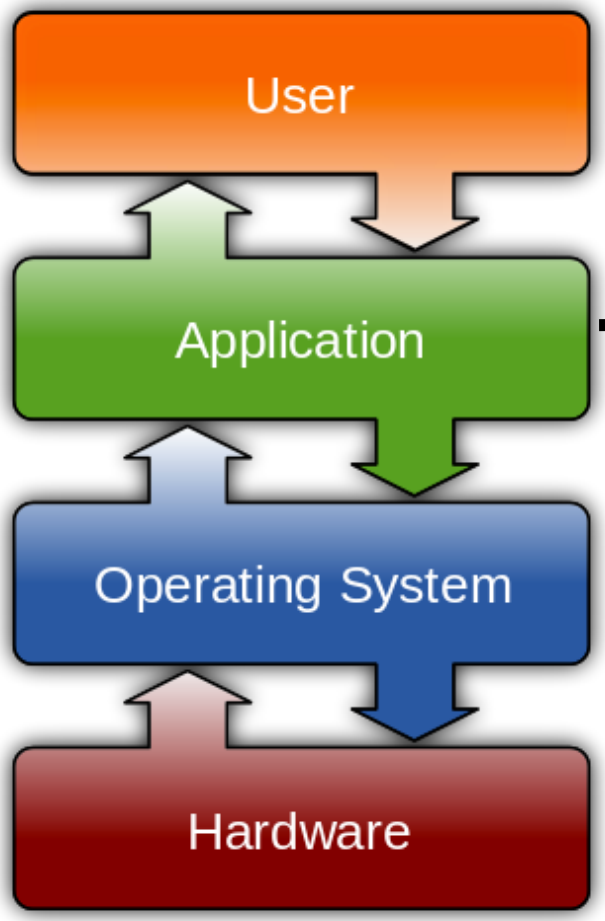
What does an OS do?



- Process management
- Interrupts
- Memory management
- File system
- Device drivers
- Networking
- Security
- I / O

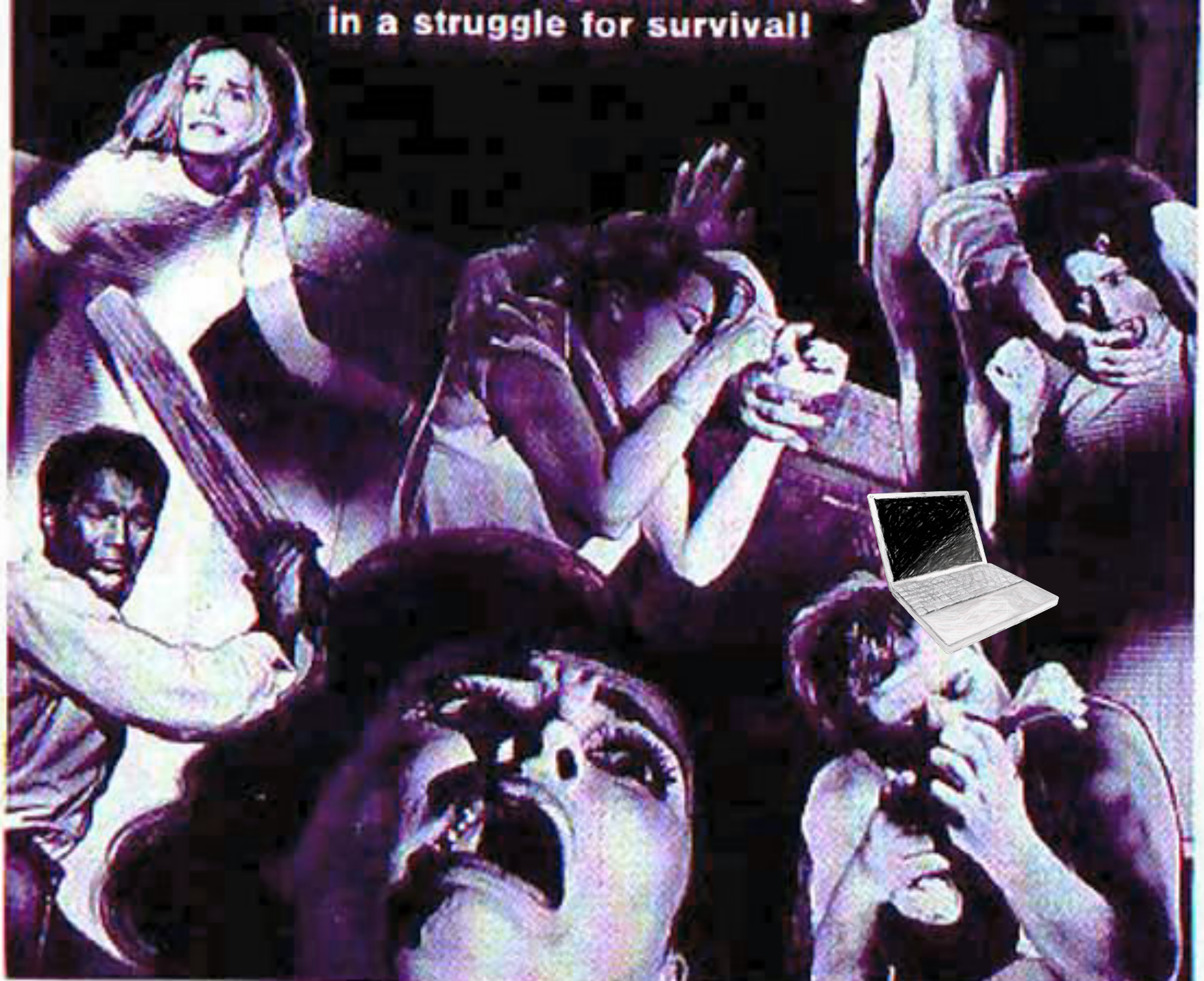
http://en.wikipedia.org/wiki/Operating_system

What does an App (C/C#/Ruby/…) do?



- Eat brains…
- *Brains!*
- ***Brains!!!***

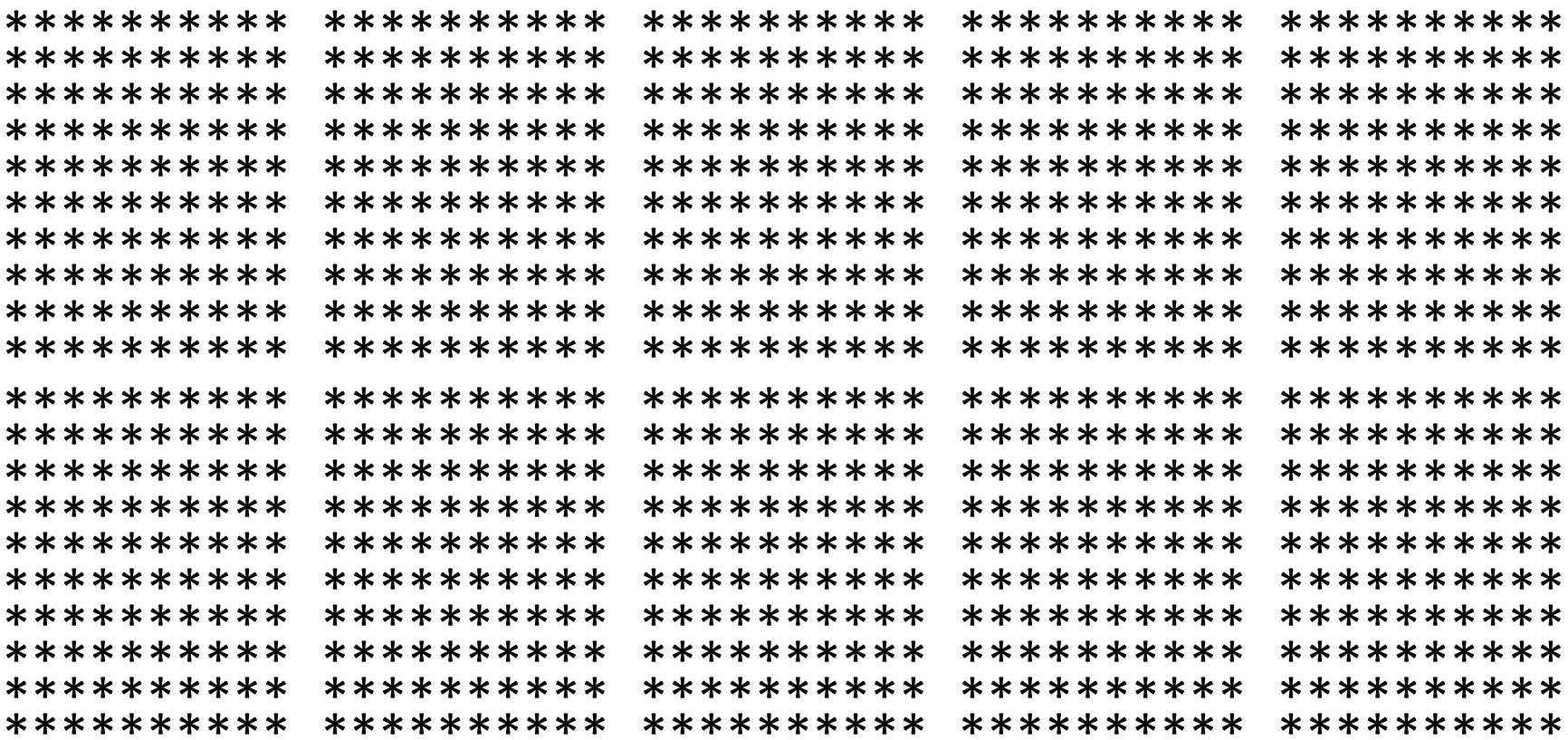
http://en.wikipedia.org/wiki/Operating_system


in a struggle for survival!

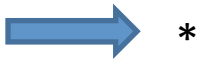

ErlangVM Processes

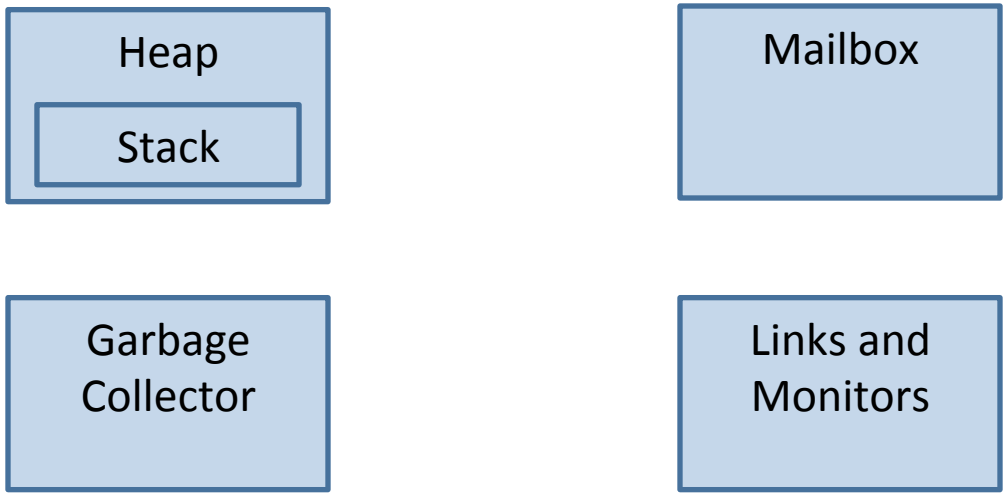
One or .NET 4.0 (or Java 6) Thread

(allocates one megabyte)

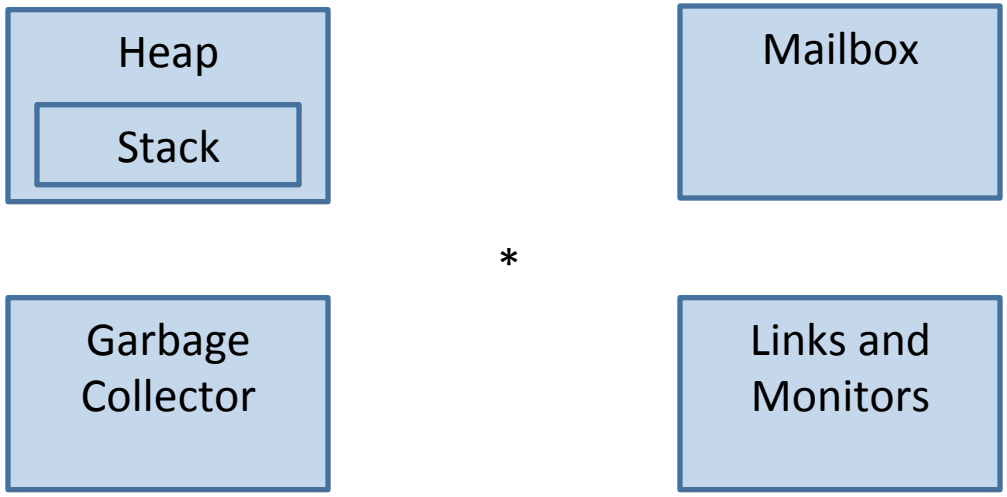


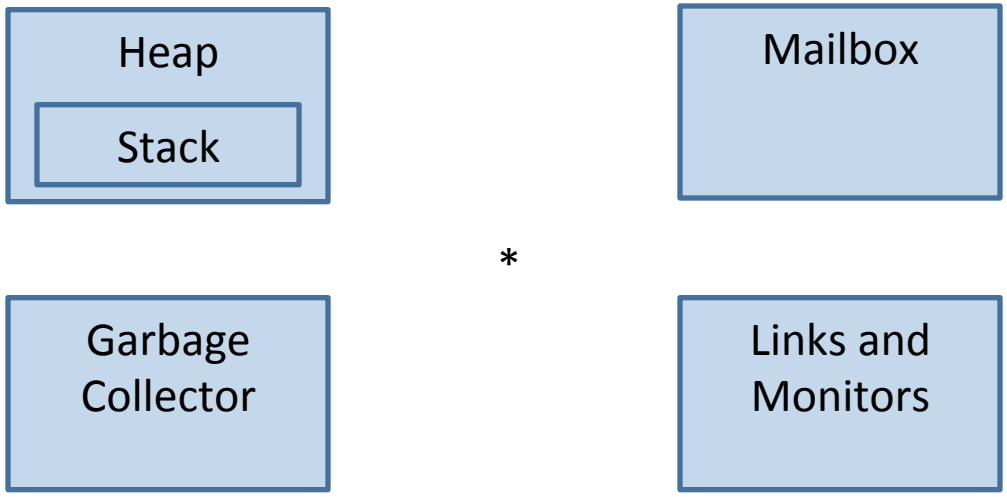
One ErlangVM Process (allocates one kilobyte)

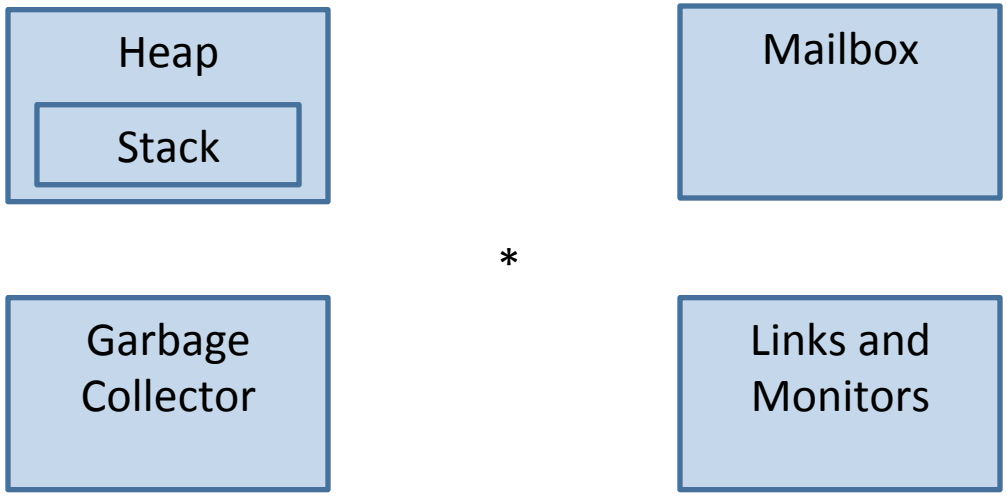


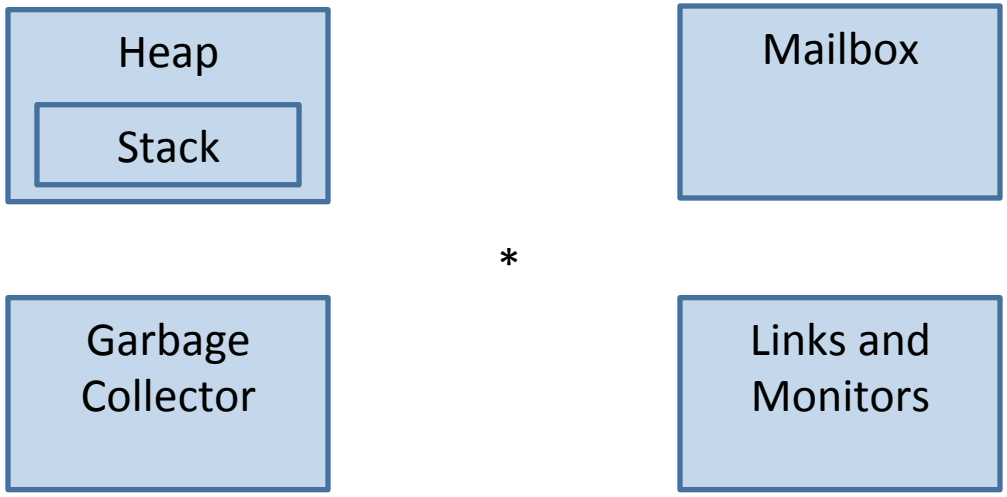

Heap
Stack
Mailbox
Garbage Collector
Links and Monitors

*

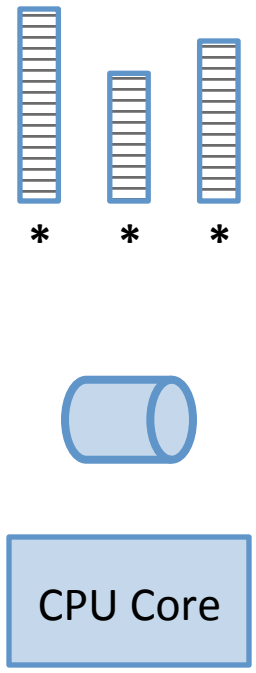

Heap
Stack
Mailbox
*
Garbage Collector
Links and Monitors


Heap
Stack
Mailbox
*
Garbage Collector
Links and Monitors

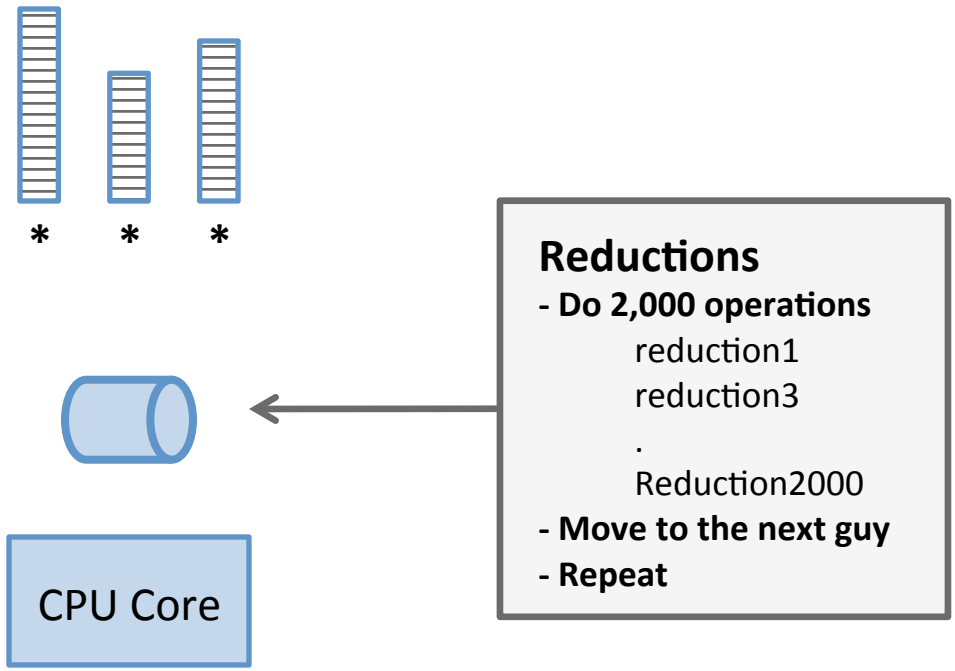

Heap
Stack
Mailbox
*
Garbage Collector
Links and Monitors


Heap
Stack
Mailbox
*
Garbage Collector
Links and Monitors

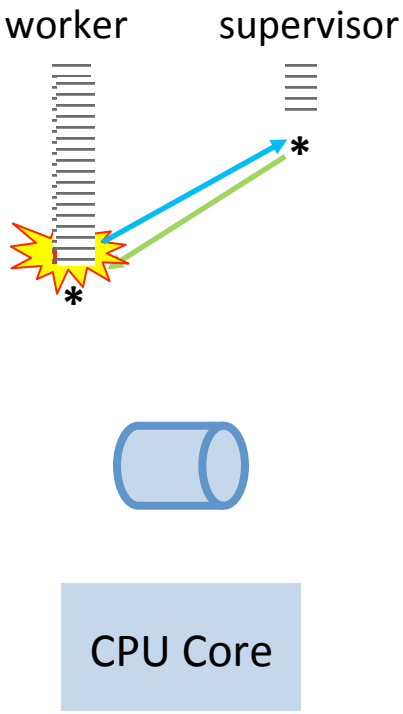
ErlangVM Scheduling

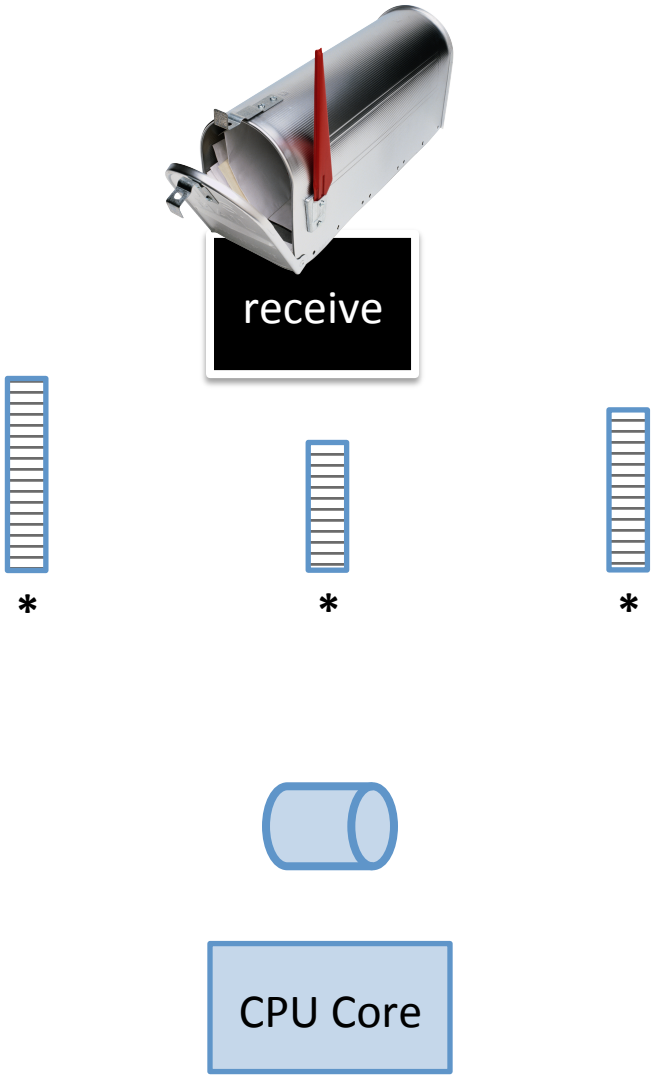

*
*
*
CPU Core

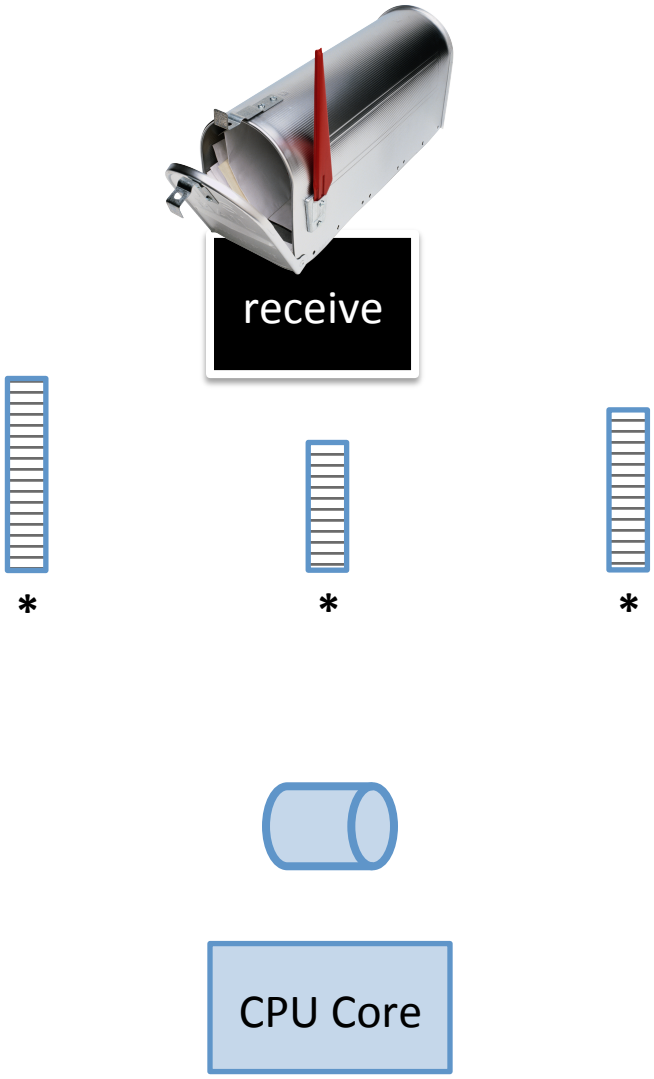
Near-zero context switching cost

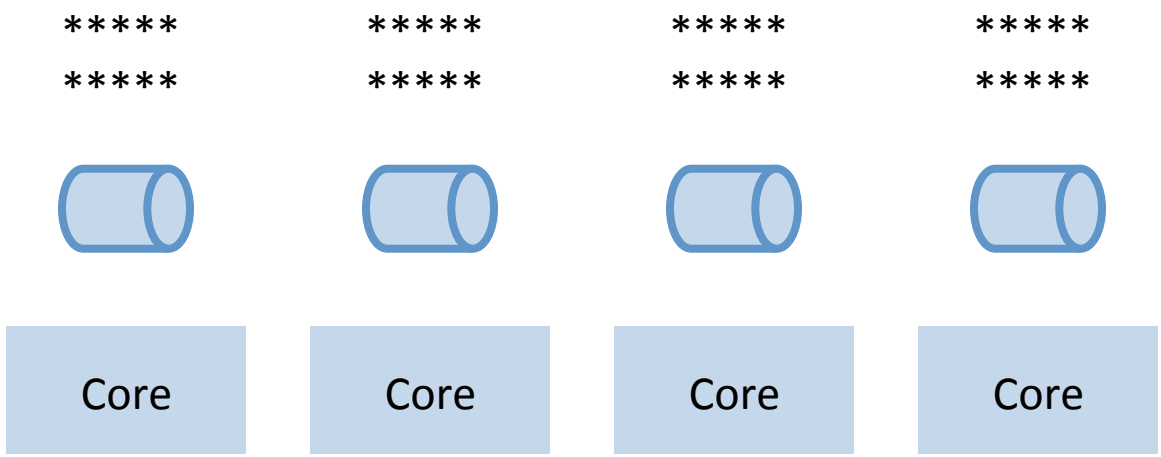


Supervision and “let-it-crash”

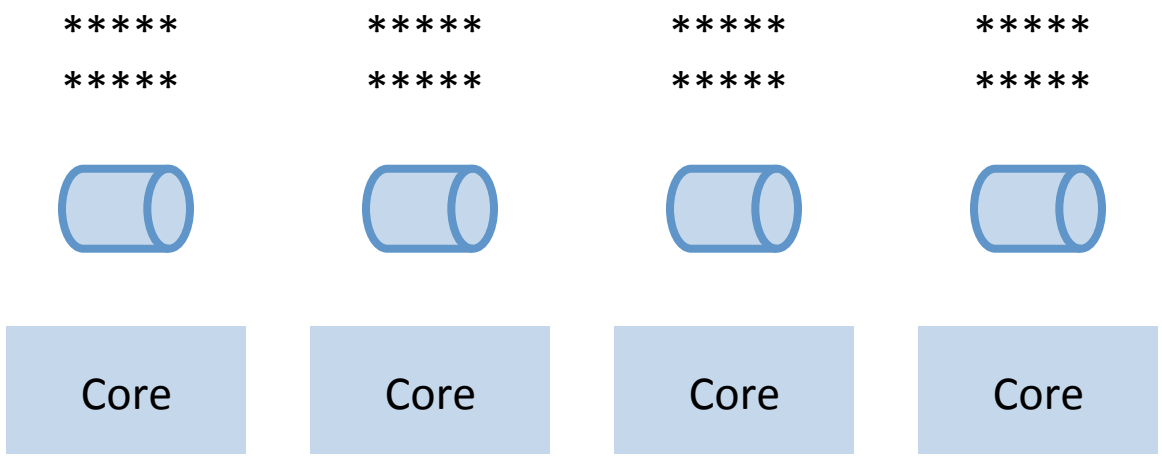



receive
*
*
*
CPU Core

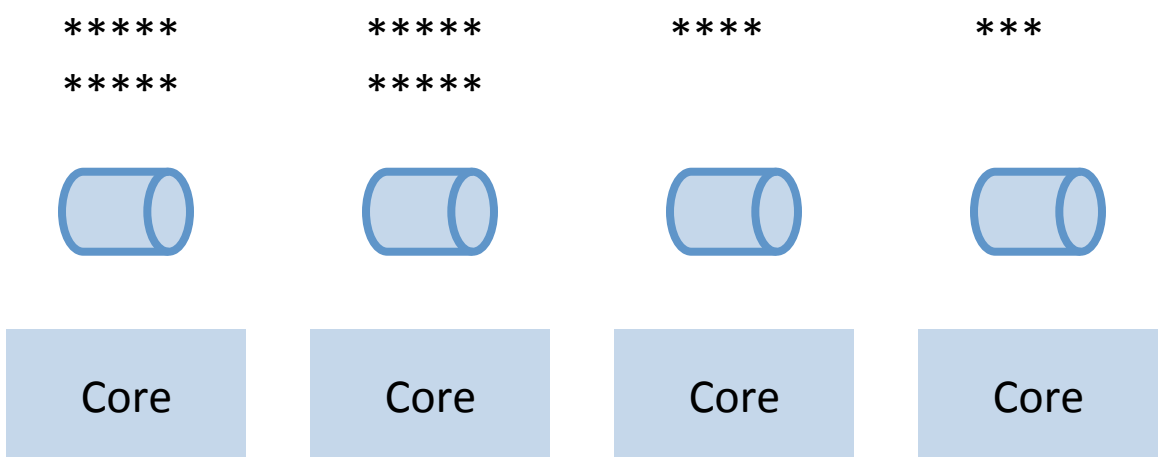

receive
*
*
*
CPU Core


*****
*****
*****
*****
*****
*****
*****
*****
Core
Core
Core
Core

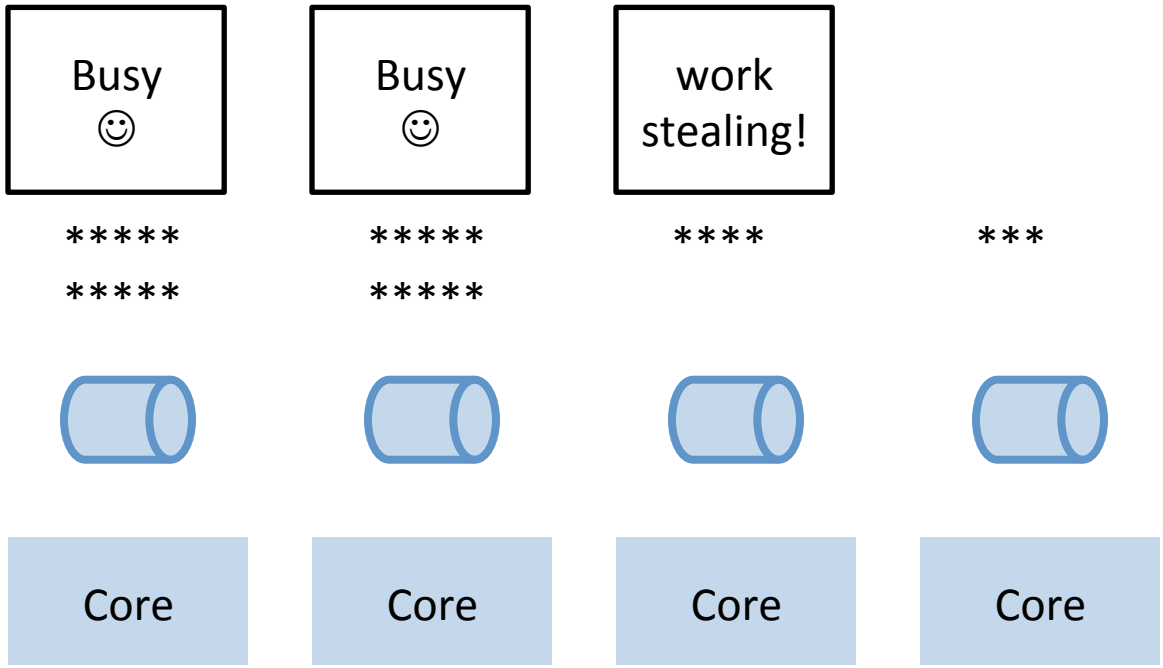
Scheduler migration : a game of balancing and compaction



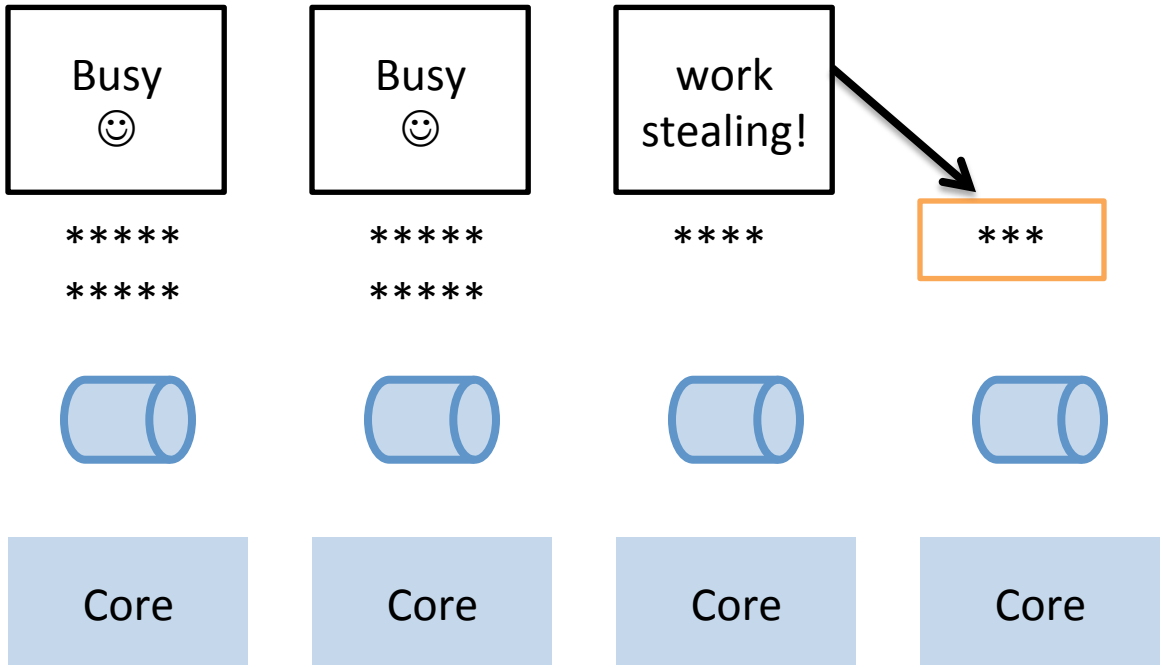
Compaction for energy saving & memory locality



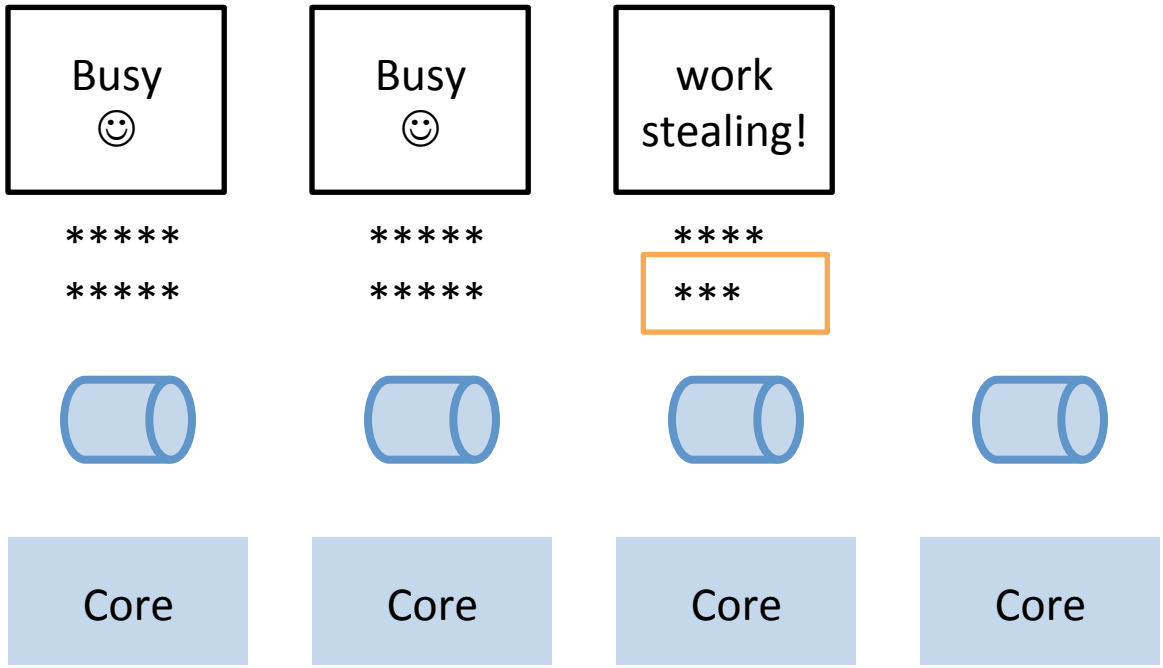
Compaction for energy saving & memory locality



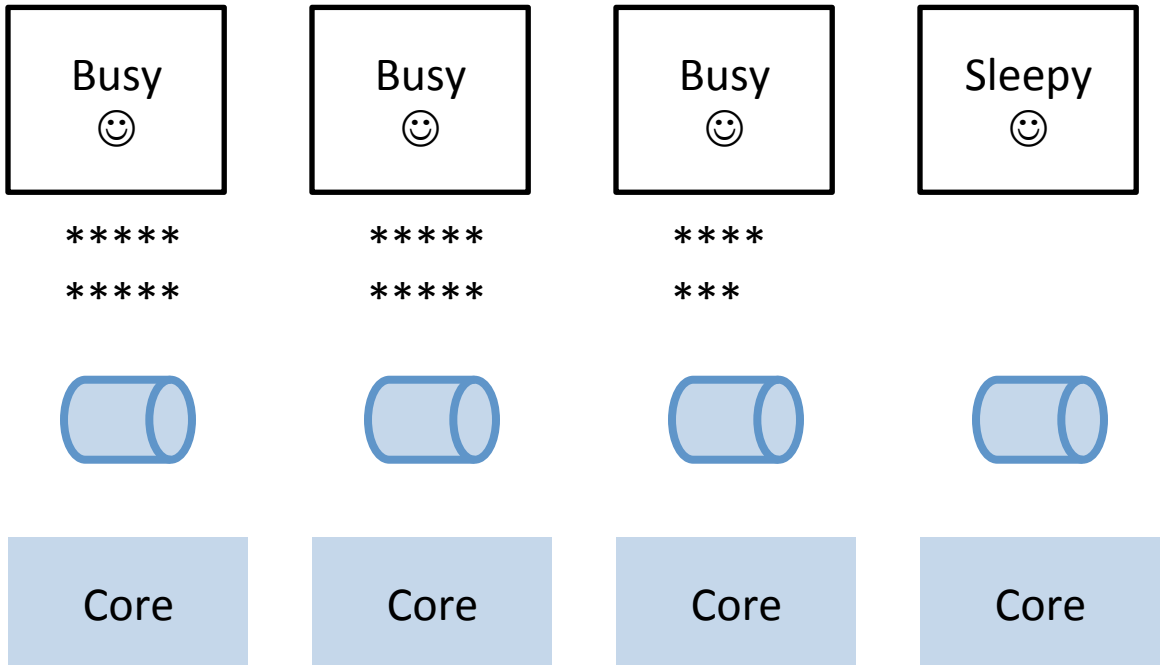
Compaction for energy saving & memory locality

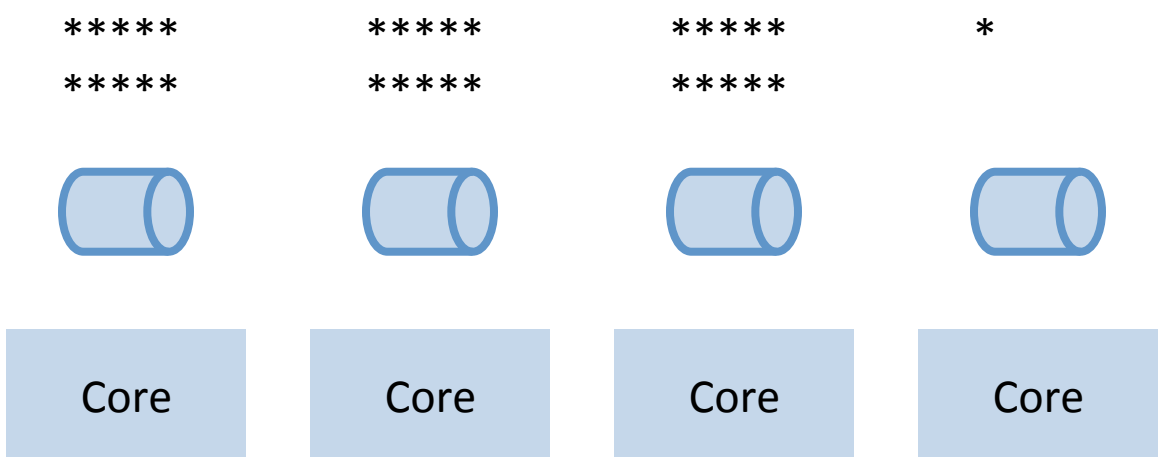


Compaction for energy saving & memory locality

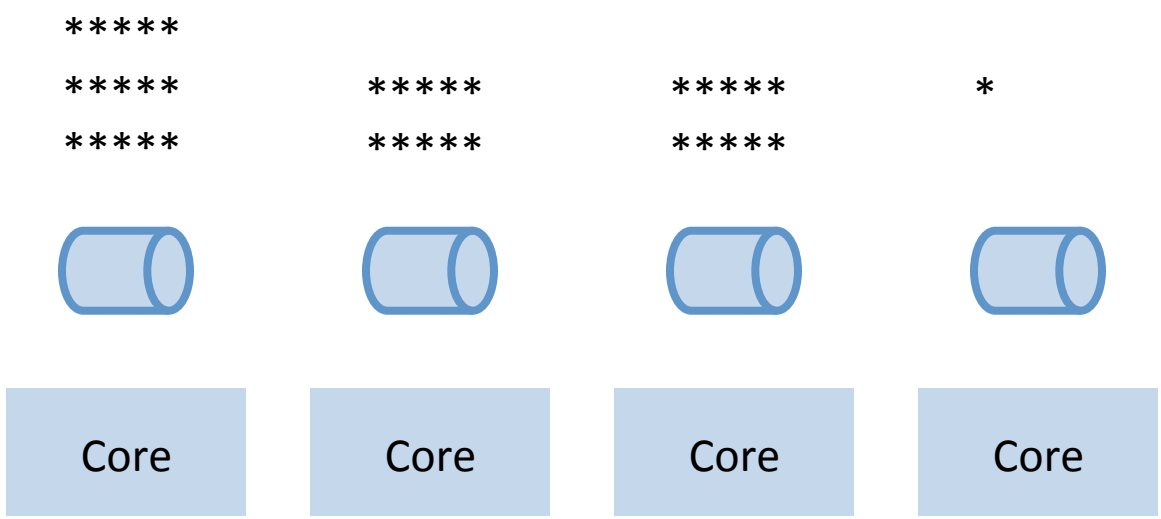


Compaction for energy saving & memory locality




*****
*****
*****
*****
*****
*****
*
Core
Core
Core
Core

Scheduler migration : balance to prevent overburden



Scheduler migration : balance to prevent overburden

*****
*****

*****
*****

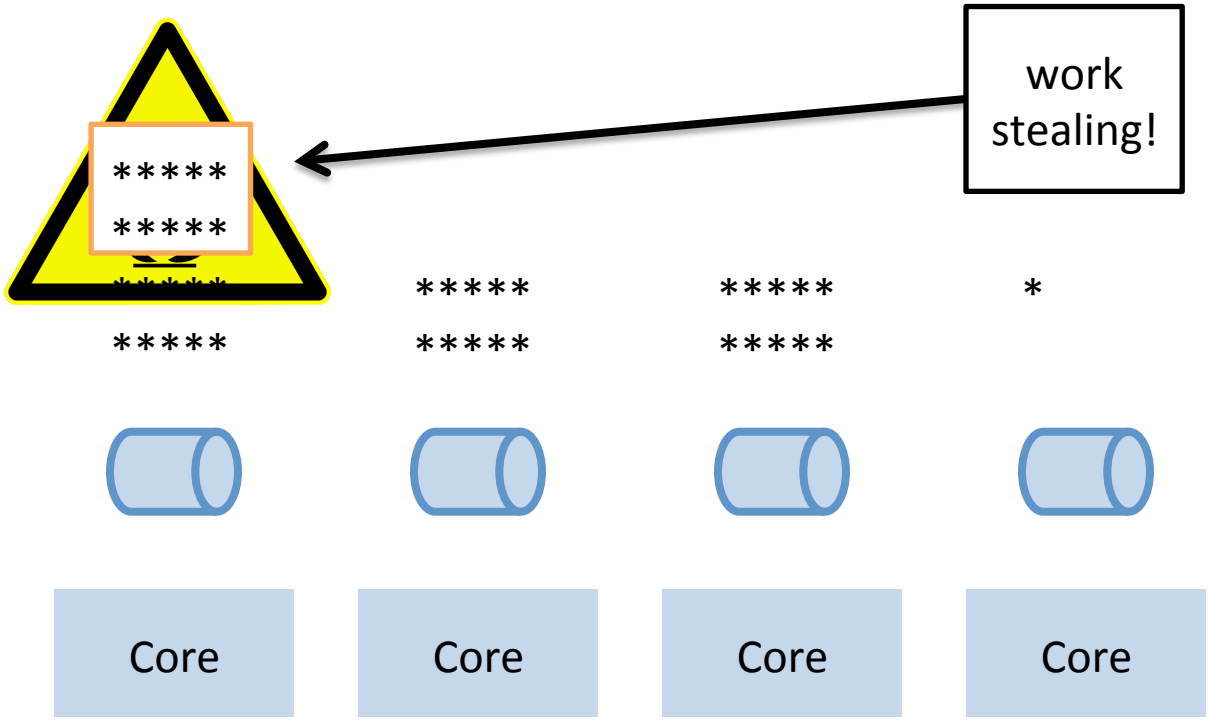
*****
*****

*

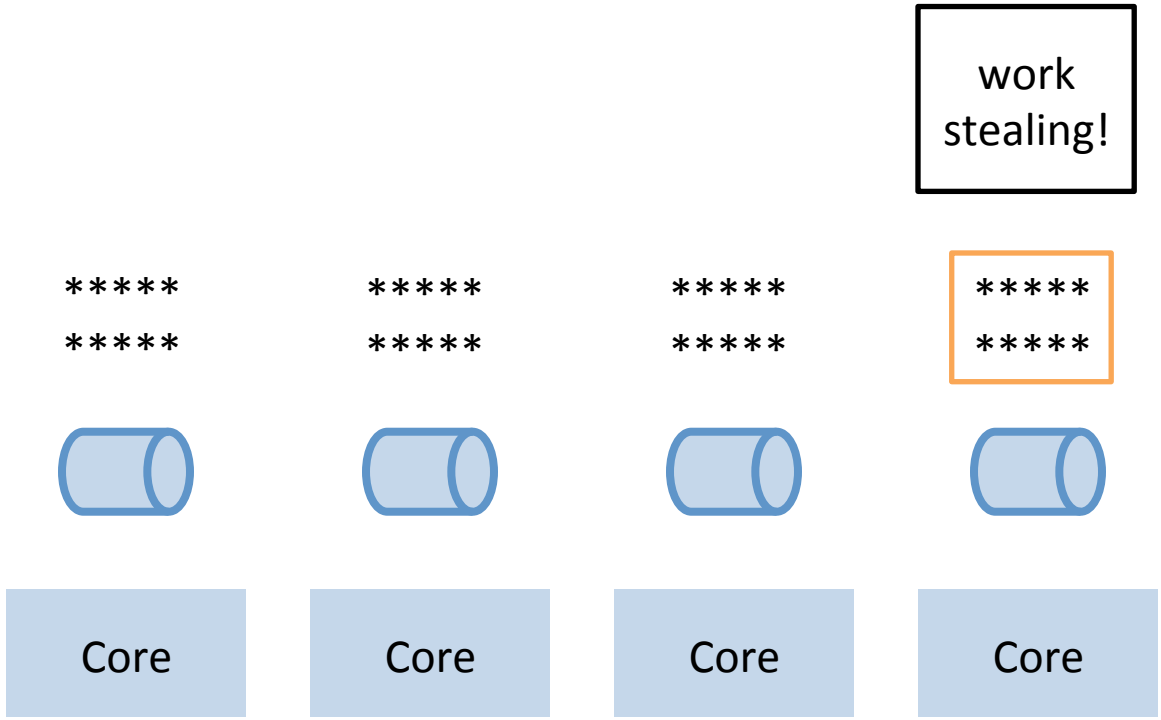



Core | Core | Core | Core

Scheduler migration : balance to prevent overburden

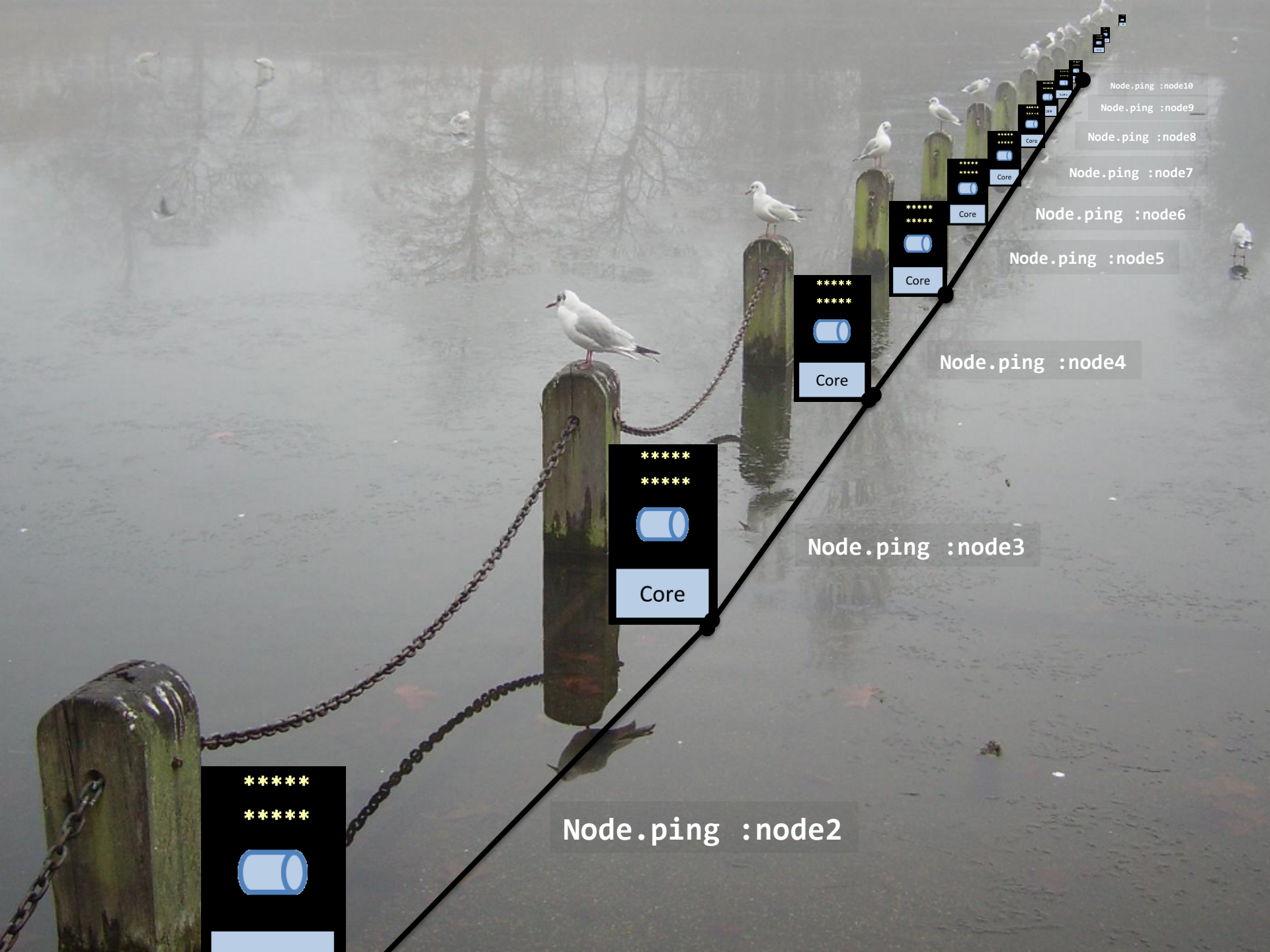


Scheduler migration : balance to prevent overburden



massive concurrency
preemptive multitasking
soft real-time
low latency over raw throughput


*****
*****
Core

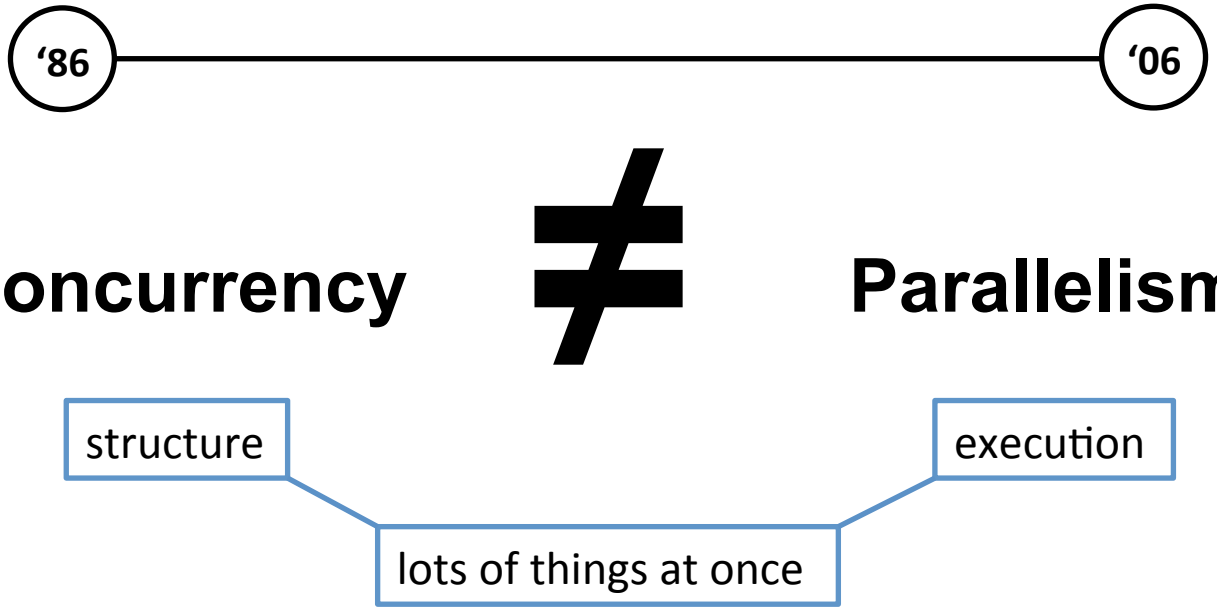

*****
*****
Core
Node.ping :node2
Node.ping :node3
Node.ping :node4
Node.ping :node5
Node.ping :node6
Node.ping :node7
Node.ping :node8
Node.ping :node9
Node.ping :node10

```elixir
defmodule Manyproc do

  def start_and_time(how_many) do
    {microseconds, :done} = :timer.tc(Manyproc, :start, [how_many])
    IO.puts "Elapsed: #{microseconds/1_000_000}"
  end

  def start(how_many), do: start_proc(how_many, self())

  def start_proc(0, pid) do
    # Last one. Send an 'ok' to the first Pid in the chain.
    send p
  end

  def star
    # Crea
    new_pi

    # Send an 'ok' message to the child process
    send new_pid, :ok

    # Wait at the mailbox until we receive an 'ok' message (from our parent).
    receive do
      :ok -> :done
    end
  end
end
```

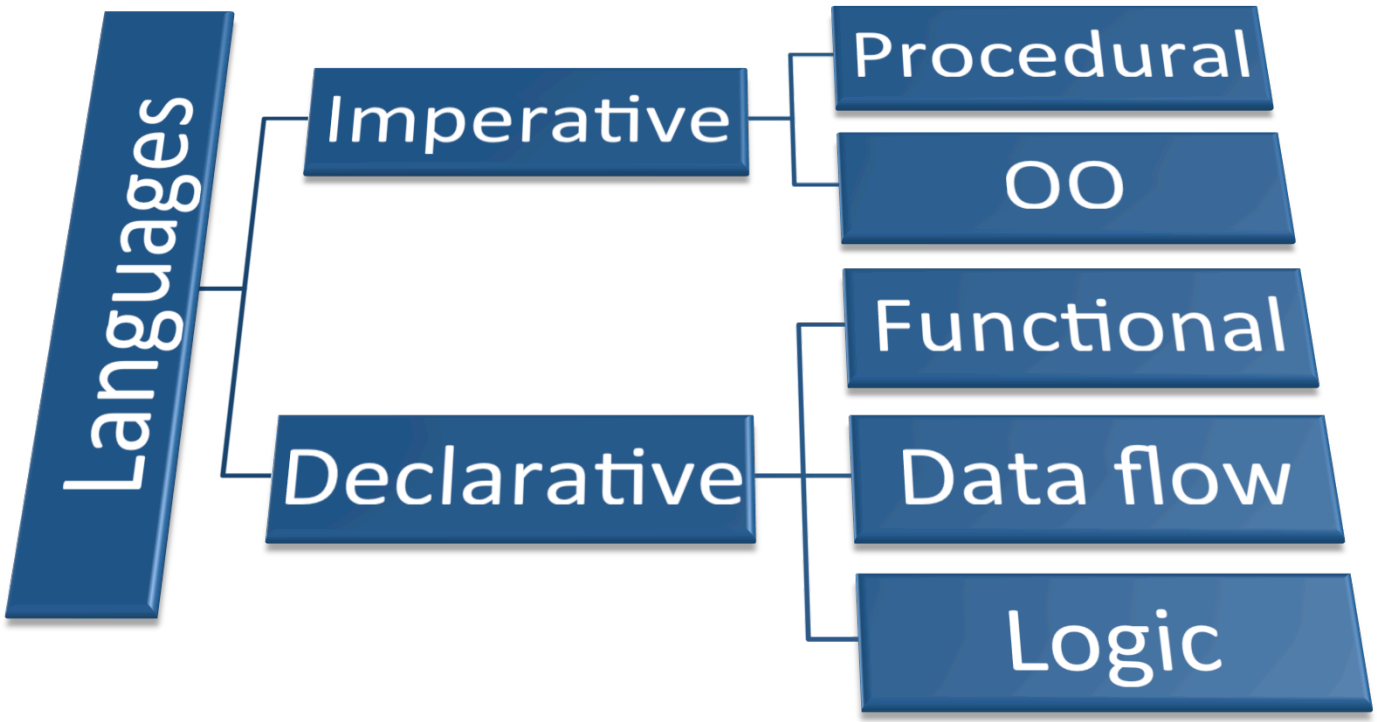
Time for a Demo?

“The performance
of a concurrent language
is predicated by three things:
the **context switching time,**
The **message passing time**,
and the **time to create a process**.”

–Mike Williams


‘86
‘06
Concurrency
≠
Parallelism
structure
execution
lots of things at once

Princeton University 1930s



ErlangVM as the mobile backend

19 billion reasons to learn about the Erlang VM

64 billion messages per day (~741,000 per second)

600 million active users

8,000 cores

Hundreds of nodes

10 Erlang engineers

60 million users per Erlang engineer


ERLANG VM
TRUE
TINCTURE OF
+ ERLANG + RUBY
+ CLOJURE + F#
LANGUAGE
INVIGORATING
ELIXIR
OTP

Interactive Elixir (IEX)

$

Interactive Elixir (IEX)

```
$ iex
```

Interactive Elixir (IEX)

```
$ iex
Erlang/OTP 17 [erts-6.1]
Interactive Elixir (0.15.1)
iex(1)>
```

Mix

```
mix          # Runs project
mix compile # Compile source
mix deps     # Dependencies
mix new      # Creates new project
mix test     # Runs project tests
(a ton more)
```

Types

```
iex> 1                          # integer
iex> 0x1F                       # integer
iex> 1.0                        # float
iex> :atom                      # atom
iex> "elixir"                   # string
iex> [1, 2, 3]                  # list
iex> {1, 2, 3}                  # tuple
iex> %{name: “Bryan”}           # map
```

Pattern Matching

```
iex(1)>
```

Pattern Matching

```
iex(1)> list = [1,2,3]
```

Pattern Matching

```
iex(1)> list = [1,2,3]
[1, 2, 3]
iex(2)>
```

Pattern Matching

```
iex(1)> list = [1,2,3]
[1, 2, 3]
iex(2)> case list do
```

Pattern Matching

```
iex(1)> list = [1,2,3]
[1, 2, 3]
iex(2)> case list do
...(2)>
```

Pattern Matching

```
iex(1)> list = [1,2,3]
[1, 2, 3]
iex(2)> case list do
...(2)> [11,12,13] -> "won't match"
```

Pattern Matching

```
iex(1)> list = [1,2,3]
[1, 2, 3]
iex(2)> case list do
...(2)> [11,12,13] -> "won't match"
...(2)>
```

Pattern Matching

```
iex(1)> list = [1,2,3]
[1, 2, 3]
iex(2)> case list do
...(2)> [11,12,13] -> "won't match"
...(2)> [1,x,3] -> "matches & binds #{x} to x”
```

Pattern Matching

```
iex(1)> list = [1,2,3]
[1, 2, 3]
iex(2)> case list do
...(2)> [11,12,13] -> "won't match"
...(2)> [1,x,3] -> "matches & binds #{x} to x"
...(2)>
```

Pattern Matching

```
iex(1)> list = [1,2,3]
[1, 2, 3]
iex(2)> case list do
...(2)> [11,12,13] -> "won't match"
...(2)> [1,x,3] -> "matches & binds #{x} to x"
...(2)> _ -> "default"
```

Pattern Matching

```
iex(1)> list = [1,2,3]
[1, 2, 3]
iex(2)> case list do
...(2)> [11,12,13] -> "won't match"
...(2)> [1,x,3] -> "matches & binds #{x} to x"
...(2)> _ -> "default"
...(2)>
```

Pattern Matching

```
iex(1)> list = [1,2,3]
[1, 2, 3]
iex(2)> case list do
...(2)> [11,12,13] -> "won't match"
...(2)> [1,x,3] -> "matches & binds #{x} to x"
...(2)> _ -> "default"
...(2)> end
```

Pattern Matching

```
iex(1)> list = [1,2,3]
[1, 2, 3]
iex(2)> case list do
...(2)> [11,12,13] -> "won't match"
...(2)> [1,x,3] -> "matches & binds #{x} to x"
...(2)> _ -> "default"
...(2)> end
"matches & binds 2 to x"
iex(3)>
```

if *(and UTF-8)*

```
if lang == :japanese do
  IO.puts “ハローワールド”
else
  IO.puts “Hello World”
end
```

if *(without an else)*

```
if 2+2 == 3 do
   IO.puts “Uh oh! 1984”
end
```

Digit Separators

```
iex(1)>
```

Digit Separators

```
iex(1)> 100000
```

Digit Separators

```
iex(1)> 100000
100000
iex(2)>
```

Digit Separators

```
iex(1)> 100000
100000
iex(2)> 100_000
```

Digit Separators

```
iex(1)> 100000
100000
iex(2)> 100_000
100000
iex(3)>
```

Digit Separators

```
iex(1)> 100000
100000
iex(2)> 100_000
100000
iex(3)> 100_000 + 5_000
```

Digit Separators

```
iex(1)> 100000
100000
iex(2)> 100_000
100000
iex(3)> 100_000 + 5_000
105000
iex(4)>
```

Maps

```
iex(1)>
```

Maps

```
iex(1)> person = %{name: "Bryan", beardy: true}
```

Maps

```
iex(1)> person = %{name: "Bryan", beardy: true}
%{beardy: true, name: "Bryan"}
iex(2)>
```

Maps

```
iex(1)> person = %{name: "Bryan", beardy: true}
%{beardy: true, name: "Bryan"}
iex(2)> person[:name]
```

Maps

```
iex(1)> person = %{name: "Bryan", beardy: true}
%{beardy: true, name: "Bryan"}
iex(2)> person[:name]
"Bryan"
iex(3)>
```

Maps

```
iex(1)> person = %{name: "Bryan", beardy: true}
%{beardy: true, name: "Bryan"}
iex(2)> person[:name]
"Bryan"
iex(3)> person.name
```

Maps

```
iex(1)> person = %{name: "Bryan", beardy: true}
%{beardy: true, name: "Bryan"}
iex(2)> person[:name]
"Bryan"
iex(3)> person.name
"Bryan"
iex(4)>
```

Maps

```
iex(1)> person = %{name: "Bryan", beardy: true}
%{beardy: true, name: "Bryan"}
iex(2)> person[:name]
"Bryan"
iex(3)> person.name
"Bryan"
iex(4)> %{person | name: "Bryan Hunter"}
```

Maps

```
iex(1)> person = %{name: "Bryan", beardy: true}
%{beardy: true, name: "Bryan"}
iex(2)> person[:name]
"Bryan"
iex(3)> person.name
"Bryan"
iex(4)> %{person | name: "Bryan Hunter"}
%{beardy: true, name: "Bryan Hunter"}
iex(5)>
```

Structs

```
iex(1)>
```

Structs

```
iex(1)> defmodule Person do
```

Structs

```
iex(1)> defmodule Person do
...(1)>
```

Structs

```
iex(1)> defmodule Person do
...(1)> defstruct name: "", sleepy: false
```

Structs

```
iex(1)> defmodule Person do
...(1)> defstruct name: "", sleepy: false
...(1)>
```

Structs

```
iex(1)> defmodule Person do
...(1)> defstruct name: "", sleepy: false
...(1)> end
```

Structs

```
iex(1)> defmodule Person do
...(1)> defstruct name: "", sleepy: false
...(1)> end
{:module, Person, [name: "", sleepy: false]}
iex(2)>
```

Structs

```
iex(1)> defmodule Person do
...(1)> defstruct name: "", sleepy: false
...(1)> end
{:module, Person, [name: "", sleepy: false]}
iex(2)> %Person{}
```

Structs

```
iex(1)> defmodule Person do
...(1)> defstruct name: "", sleepy: false
...(1)> end
{:module, Person, [name: "", sleepy: false]}
iex(2)> %Person{}
%Person{name: "", sleepy: false}
iex(3)>
```

Structs

```
iex(1)> defmodule Person do
...(1)> defstruct name: "", sleepy: false
...(1)> end
{:module, Person, [name: "", sleepy: false]}
iex(2)> %Person{}
%Person{name: "", sleepy: false}
iex(3)> bryan = %Person{name: "Bryan",sleepy: true}
```

Structs

```
iex(1)> defmodule Person do
...(1)> defstruct name: "", sleepy: false
...(1)> end
{:module, Person, [name: "", sleepy: false]}
iex(2)> %Person{}
%Person{name: "", sleepy: false}
iex(3)> bryan = %Person{name: "Bryan",sleepy: true}
%Person{name: "Bryan", sleepy: true}
iex(4)>
```

Structs

```
iex(1)> defmodule Person do
...(1)> defstruct name: "", sleepy: false
...(1)> end
{:module, Person, [name: "", sleepy: false]}
iex(2)> %Person{}
%Person{name: "", sleepy: false}
iex(3)> bryan = %Person{name: "Bryan",sleepy: true}
%Person{name: "Bryan", sleepy: true}
iex(4)> %Person{name: "Bryan", city: "Nashville”}
```

Structs

```
iex(1)> defmodule Person do
...(1)> defstruct name: "", sleepy: false
...(1)> end
{:module, Person, [name: "", sleepy: false]}
iex(2)> %Person{}
%Person{name: "", sleepy: false}
iex(3)> bryan = %Person{name: "Bryan",sleepy: true}
%Person{name: "Bryan", sleepy: true}
iex(4)> %Person{name: "Bryan", city: "Nashville"}
** (CompileError) iex:4: unknown key :city for struct Person
iex(5)>
```

Enumerables & Streams

```
iex(1)>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>   Enum.sum
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>   Enum.sum
7500000000
iex(3)>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>   Enum.sum
7500000000
iex(3)> 1..100_000 |>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>   Enum.sum
7500000000
iex(3)> 1..100_000 |>
...(3)>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>   Enum.sum
7500000000
iex(3)> 1..100_000 |>
...(3)>   Stream.map(&(&1 * 3)) |>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>   Enum.sum
7500000000
iex(3)> 1..100_000 |>
...(3)>   Stream.map(&(&1 * 3)) |>
...(3)>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>   Enum.sum
7500000000
iex(3)> 1..100_000 |>
...(3)>   Stream.map(&(&1 * 3)) |>
...(3)>   Stream.filter(odd?) |>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>   Enum.sum
7500000000
iex(3)> 1..100_000 |>
...(3)>   Stream.map(&(&1 * 3)) |>
...(3)>   Stream.filter(odd?) |>
...(3)>
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>   Enum.sum
7500000000
iex(3)> 1..100_000 |>
...(3)>   Stream.map(&(&1 * 3)) |>
...(3)>   Stream.filter(odd?) |>
...(3)>   Enum.sum
```

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>   Enum.sum
7500000000
iex(3)> 1..100_000 |>
...(3)>   Stream.map(&(&1 * 3)) |>
...(3)>   Stream.filter(odd?) |>
...(3)>   Enum.sum
7500000000
iex(4)>
```

Both implement the `Enumerable` protocol

Enumerables & Streams

```
iex(1)> odd? = &(rem(&1, 2) != 0)
#Function<6.90072148/1 in :erl_eval.expr/5>
iex(2)> 1..100_000 |>
...(2)>   Enum.map(&(&1 * 3)) |>
...(2)>   Enum.filter(odd?) |>
...(2)>   Enum.sum
7500000000
iex(3)> 1..100_000 |>
...(3)>   Stream.map(&(&1 * 3)) |>
...(3)>   Stream.filter(odd?) |>
...(3)>   Enum.sum
7500000000
iex(4)>
```


ERLANG VM
TRUE
TINCTURE OF
+ ERLANG + RUBY
+ CLOJURE + F#
LANGUAGE
INVIGORATING
ELIXIR
O T
P

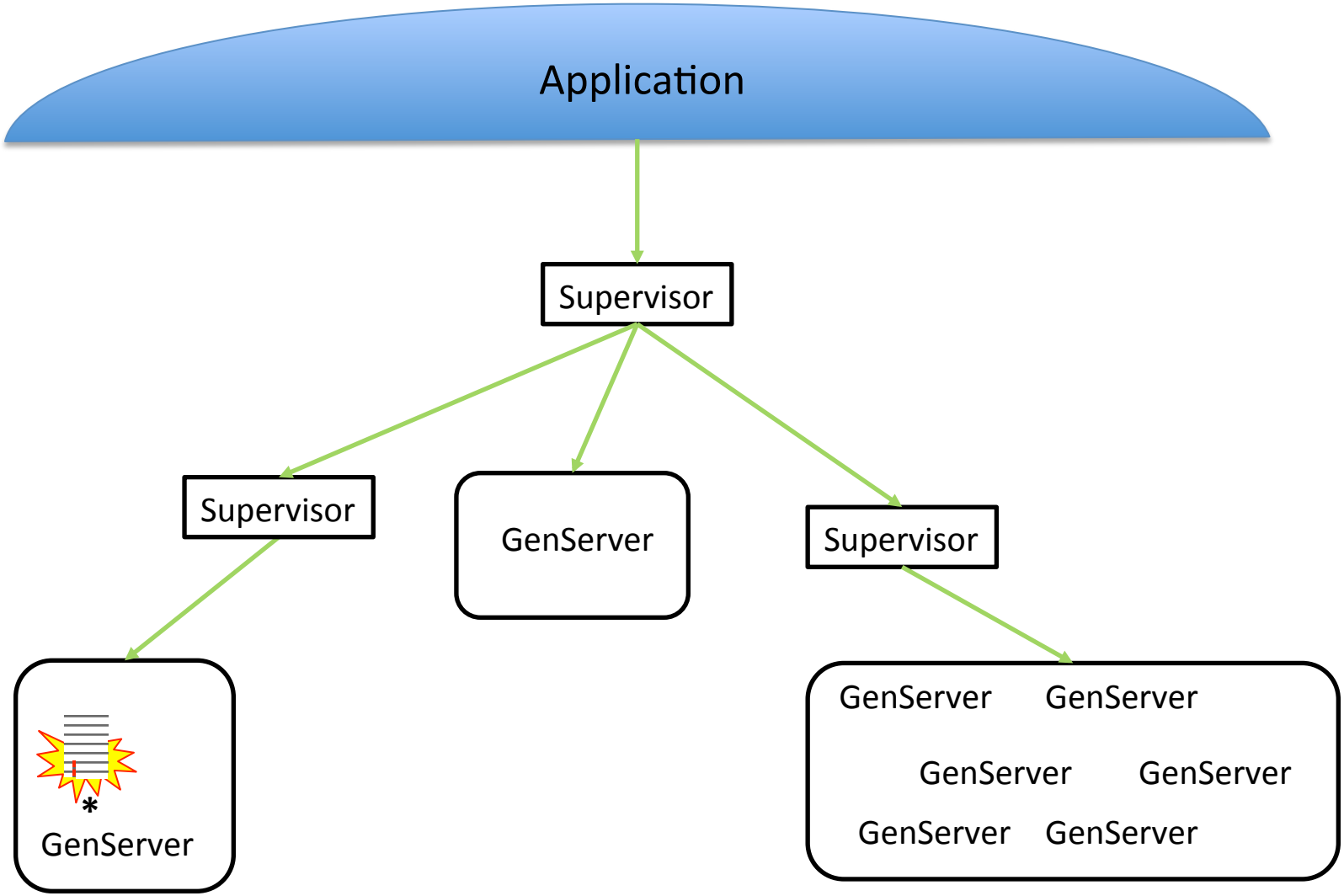
Behaviours

GenServer

Supervisor

Application

GenEvent


Application
Supervisor
Supervisor
GenServer
Supervisor
GenServer
GenServer
GenServer
GenServer
GenServer
GenServer
GenServer

Agents

```
iex>
```

Agents

```
iex> {:ok, agent} = Agent.start_link fn -> [] end
```

Agents

```
iex> {:ok, agent} = Agent.start_link fn -> [] end
{:ok, #PID<0.57.0>}
iex>
```

Agents

```
iex> {:ok, agent} = Agent.start_link fn -> [] end
{:ok, #PID<0.57.0>}
iex> Agent.update(agent, fn list->["eggs"|list] end)
```

Agents

```
iex> {:ok, agent} = Agent.start_link fn -> [] end
{:ok, #PID<0.57.0>}
iex> Agent.update(agent, fn list->["eggs"|list] end)
:ok
iex>
```

Agents

```
iex> {:ok, agent} = Agent.start_link fn -> [] end
{:ok, #PID<0.57.0>}
iex> Agent.update(agent, fn list->["eggs"|list] end)
:ok
iex> Agent.update(agent, fn list->["milk"|list] end)
```

Agents

```
iex> {:ok, agent} = Agent.start_link fn -> [] end
{:ok, #PID<0.57.0>}
iex> Agent.update(agent, fn list->["eggs"|list] end)
:ok
iex> Agent.update(agent, fn list->["milk"|list] end)
:ok
iex>
```

Agents

```
iex> {:ok, agent} = Agent.start_link fn -> [] end
{:ok, #PID<0.57.0>}
iex> Agent.update(agent, fn list->["eggs"|list] end)
:ok
iex> Agent.update(agent, fn list->["milk"|list] end)
:ok
iex> Agent.get(agent, fn list->list end)
```

Agents

```
iex> {:ok, agent} = Agent.start_link fn -> [] end
{:ok, #PID<0.57.0>}
iex> Agent.update(agent, fn list->["eggs"|list] end)
:ok
iex> Agent.update(agent, fn list->["milk"|list] end)
:ok
iex> Agent.get(agent, fn list->list end)
["milk", "eggs"]
iex>
```

Agents

```
iex> {:ok, agent} = Agent.start_link fn -> [] end
{:ok, #PID<0.57.0>}
iex> Agent.update(agent, fn list->["eggs"|list] end)
:ok
iex> Agent.update(agent, fn list->["milk"|list] end)
:ok
iex> Agent.get(agent, fn list->list end)
["milk", "eggs"]
iex> Agent.stop(agent)
```

Agents

```
iex> {:ok, agent} = Agent.start_link fn -> [] end
{:ok, #PID<0.57.0>}
iex> Agent.update(agent, fn list->["eggs"|list] end)
:ok
iex> Agent.update(agent, fn list->["milk"|list] end)
:ok
iex> Agent.get(agent, fn list->list end)
["milk", "eggs"]
iex> Agent.stop(agent)
:ok
```

Tasks

```
task = Task.async(fn -> do_some_work() end)

data = do_some_other_work()

# Next, await the response from do_some_work
other_data = Task.await(task)
```

Two Neat Extra Bits

Phoenix is to Elixir

as

Rails is to Ruby

Ecto is to Elixir

as

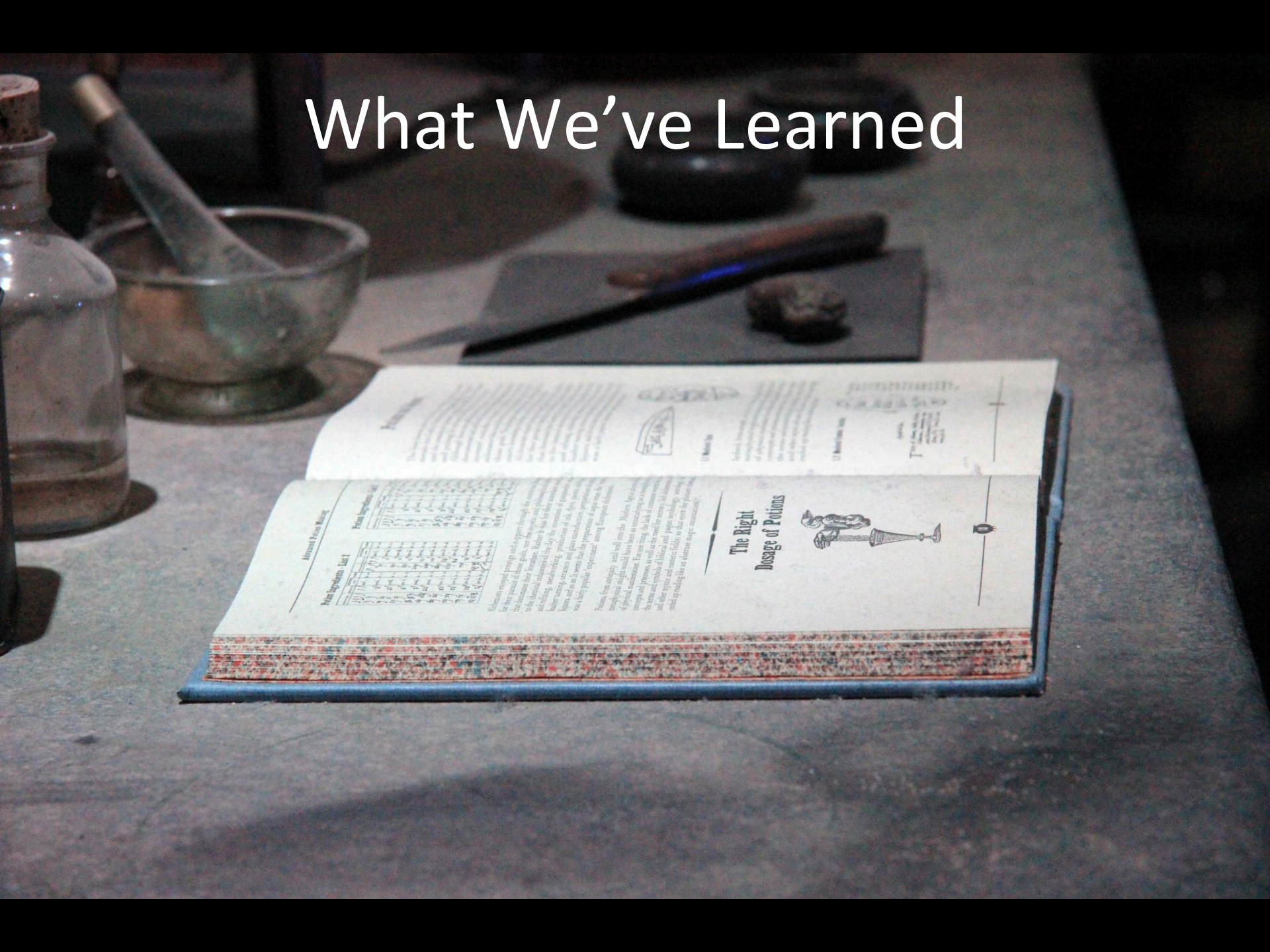
Active Record is to Ruby

as

LINQ is to .NET

Time for a demo?

What We've Learned



Everything
available
to Erlang
is available
to Elixir

Zero
inter-op
penalties

Erlang

makes the hard
things easy
and the easy
things hard

Elixir

makes the hard
things easy
and the easy
things easy too

Mostly Erlang

Erlang and sometimes other functional programming

Home About

Panel

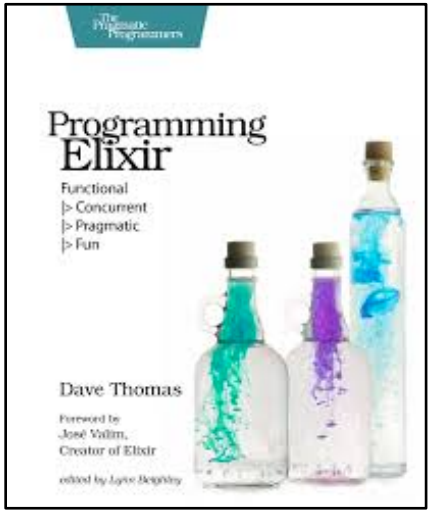
- Bryan Hunter (@bryan_hunter)
- Jose Valim (@josevalim)
- Simon St. Laurent (@simonstl)
- Robert Virding (@rvirding)
- Joe Armstrong (@joeerl)
- Fred Hebert (@mononcqc)
- Eric Merit (@ericbmerritt)

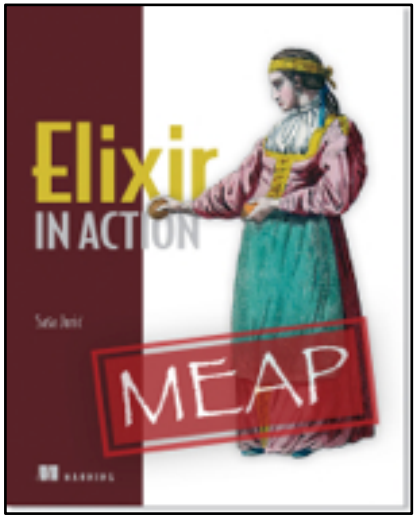
019 Elixir With José Valim

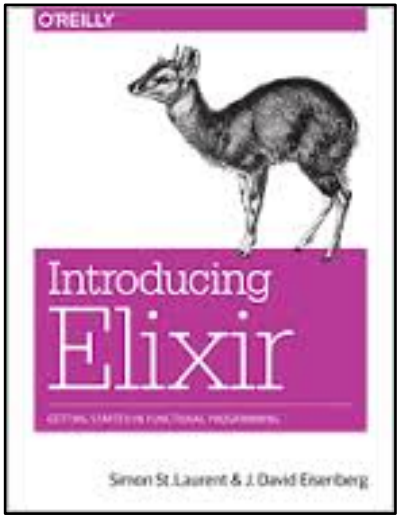
Posted on October 7, 2013 by mostlyerlang

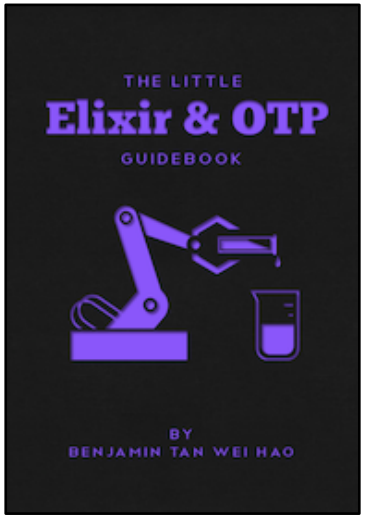
Download Link: http://mostlyerlang.files.wordpress.com/2013/10/mostly-erlang-19-elixir.mp3

Books


Programming
Elixir
Functional
|> Concurrent
|> Pragmatic
|> Fun
Dave Thomas
Foreword by
José Valim,
Creator of Elixir


Elixir
IN ACTION
MEAP


O'REILLY
Introducing
Elixir
Simon St.Laurent & J. David Eisenberg


THE LITTLE
Elixir & OTP
GUIDEBOOK
BY
BENJAMIN TAN WEI HAO

Elixir Sips


elixir sips
with Josh Adams
Learn elixir with
two short videos
each week.

Join your local polyglot FP group

NashFP

NASHVILLE FUNCTIONAL PROGRAMMERS

http://nashfp.org

elixir-nash

Nashville, TN

Founded Aug 17, 2014

About us...

beamers 47

Upcoming Meetups 6

Welcome!

✚ SUGGEST A NEW MEETUP

Upcoming 6 | Past | Calendar

Let's talk elixir

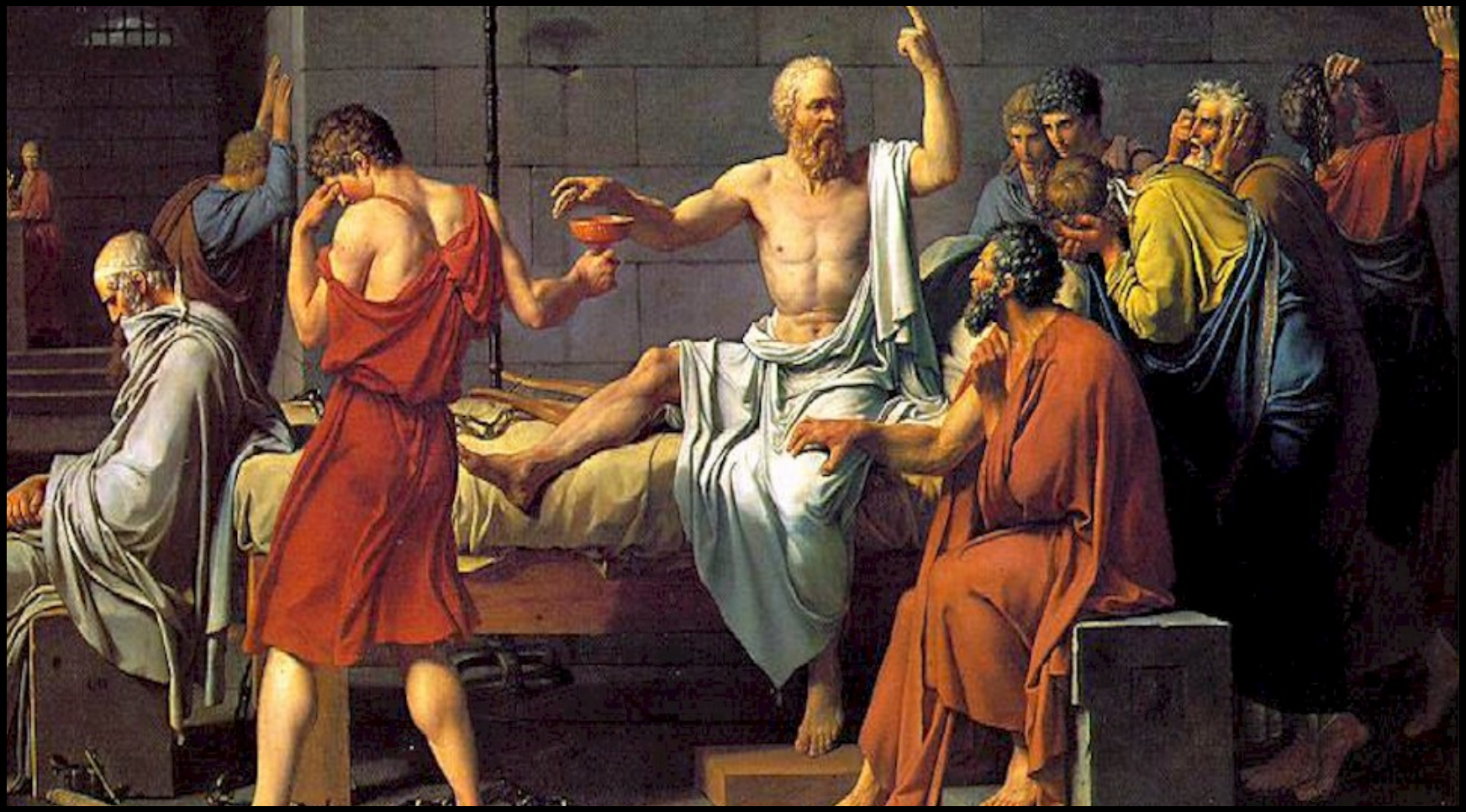
Firefly Logic
1000 Main St Ste 201, Nashville, TN (map)

This is the monthly elixir meetup. We will meet on the 3rd Monday of every month. Let's get together and talk about all things elixir. LEARN MORE

Hosted by: Malcolm Roberts (Organizer)

Mon Jan 19
6:00 PM
✓ I'M GOING
8 going
2 comments

FP and Ethics



Questions? Feedback?

Bryan Hunter

Twitter: **@bryan_hunter**

Email: bryan.hunter@fireflylogic.com

Firefly Logic, Inc.

1000 Main Street #201

Nashville, TN 37206

http://fireflylogic.com

contact@fireflylogic.com