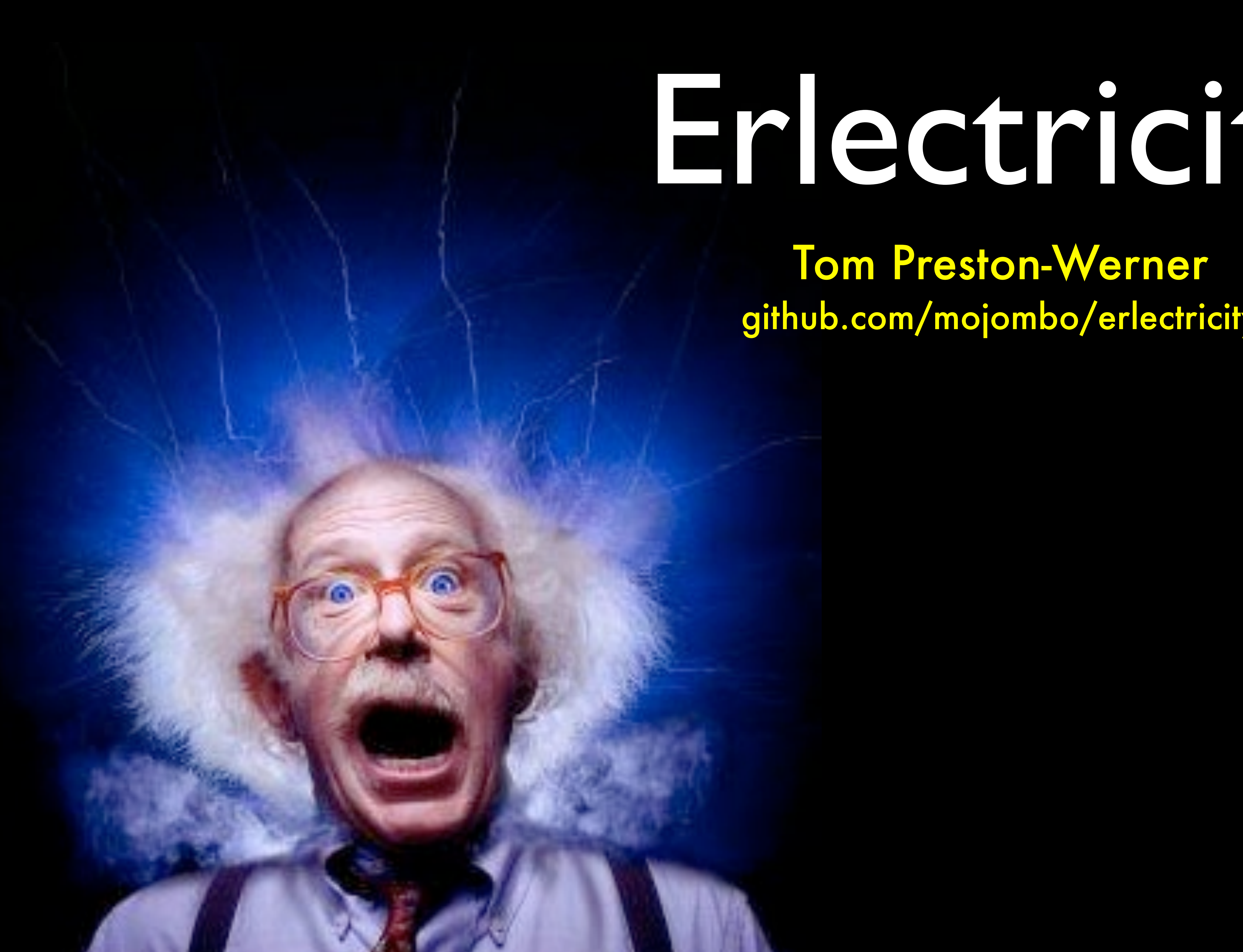


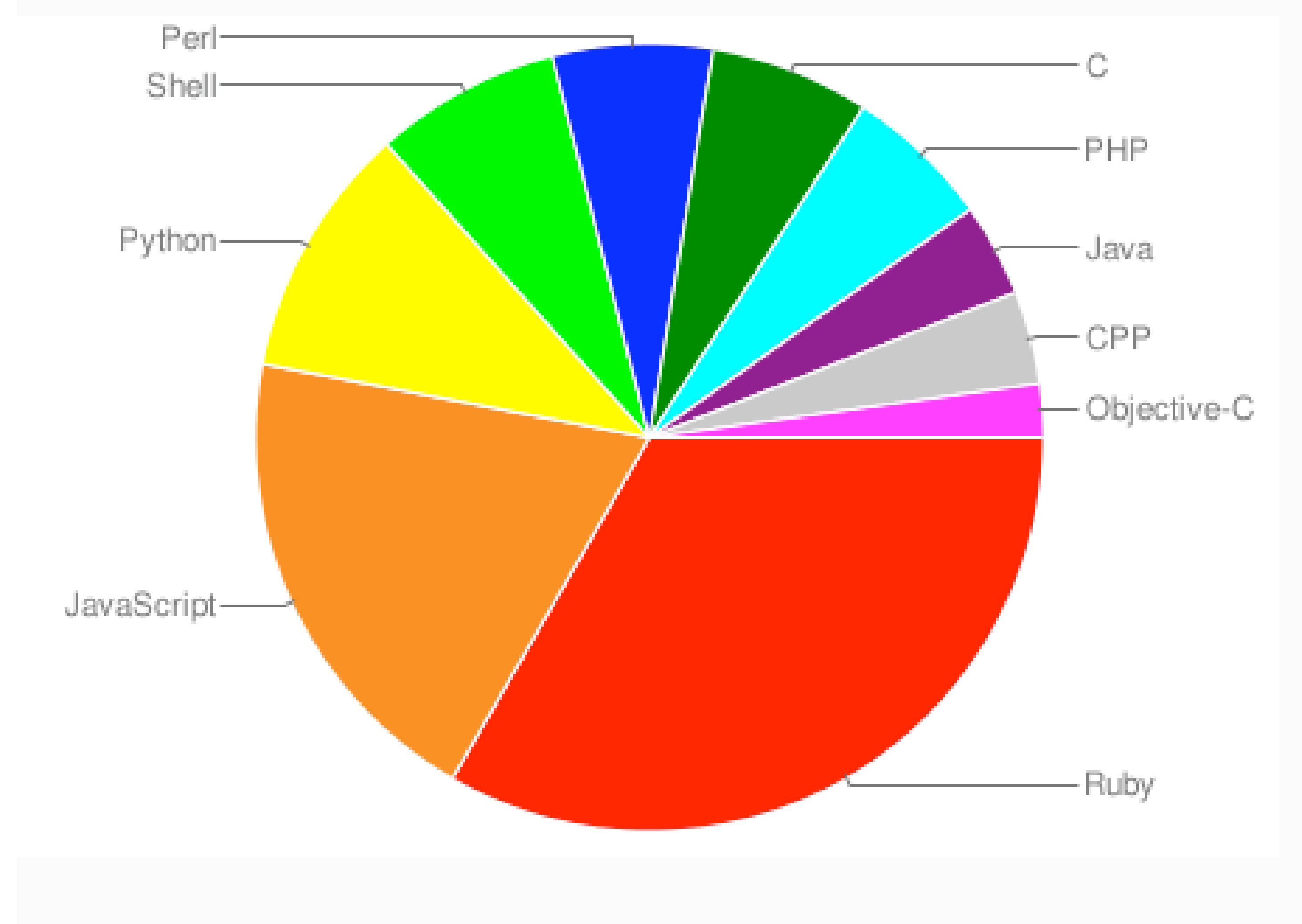
Erlectricity

Tom Preston-Werner

github.com/mojombo/erlectricity

github

Top Languages



Language Name	Percentage
Ruby	29%
JavaScript	17%
Python	9%
Shell	7%
Perl	6%
C	6%
PHP	5%
Java	4%
C++	4%
Objective-C	2%

Erlang is the #16 most popular language on GitHub.

Most Watched Overall

- KirinDave / **fuzed**
- tarcieri / **reia**
- cliffmoon / **dynomite**
- rklophaus / **nitrogen**
- tonyg / **reversehttp**

Most Forked Overall

- rklophaus / **nitrogen**
- halorgium / **couchdb**
- tarcieri / **reia**
- KirinDave / **fuzed**
- cliffmoon / **dynomite**

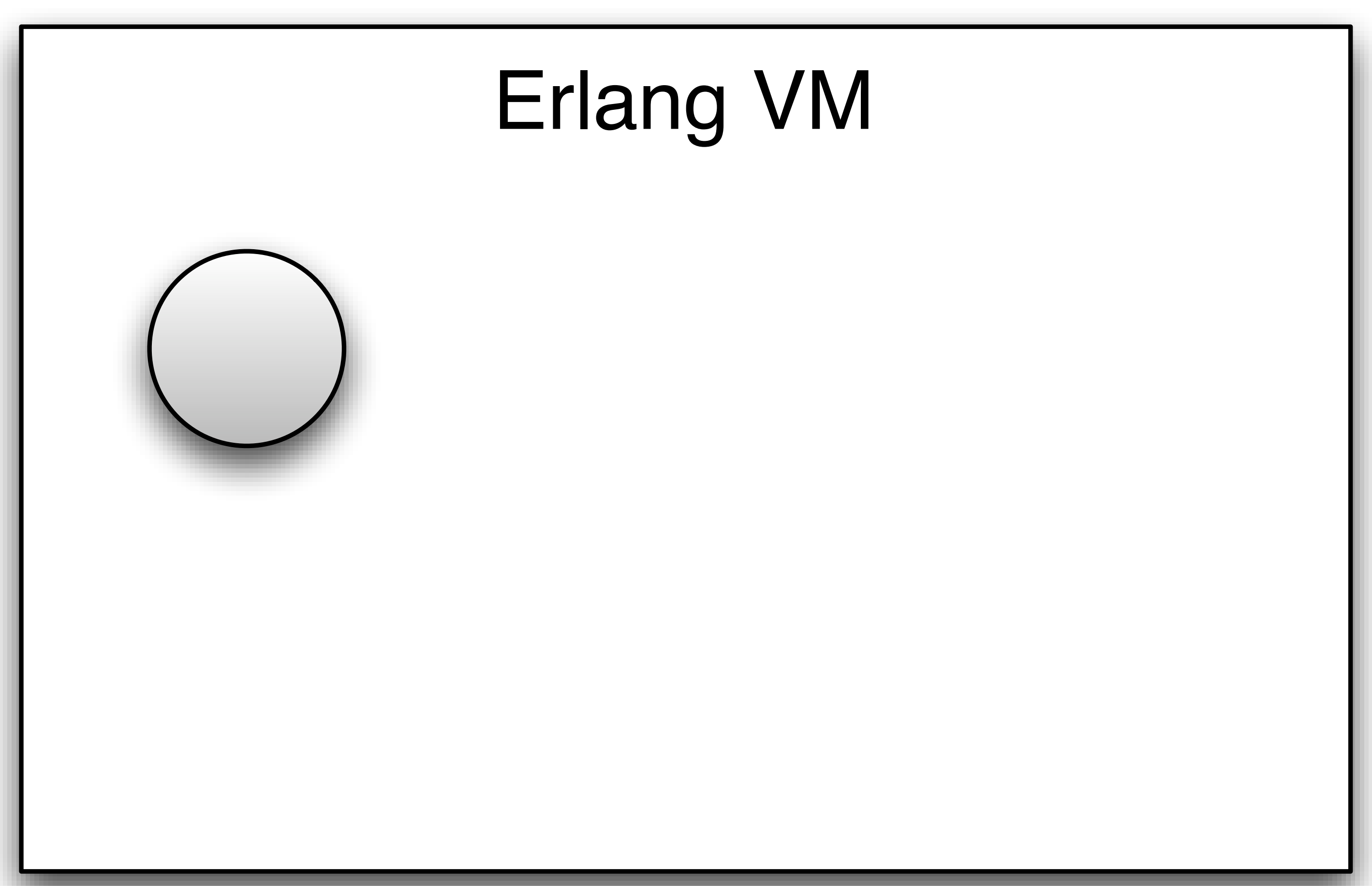
RUBY WAY
SECOND EDITION
HAL FULTON
DUMMIES
ALPHABETICAL BY PUBLISHER
Rails Recipes
Rails
Programming Ruby
Ruby Cookbook
Ruby Cookbook
Rails Cookbook
Ruby on Rails
O'REILLY
XML Databases

NET ARCHIVE
INTERNET ARCHIVE

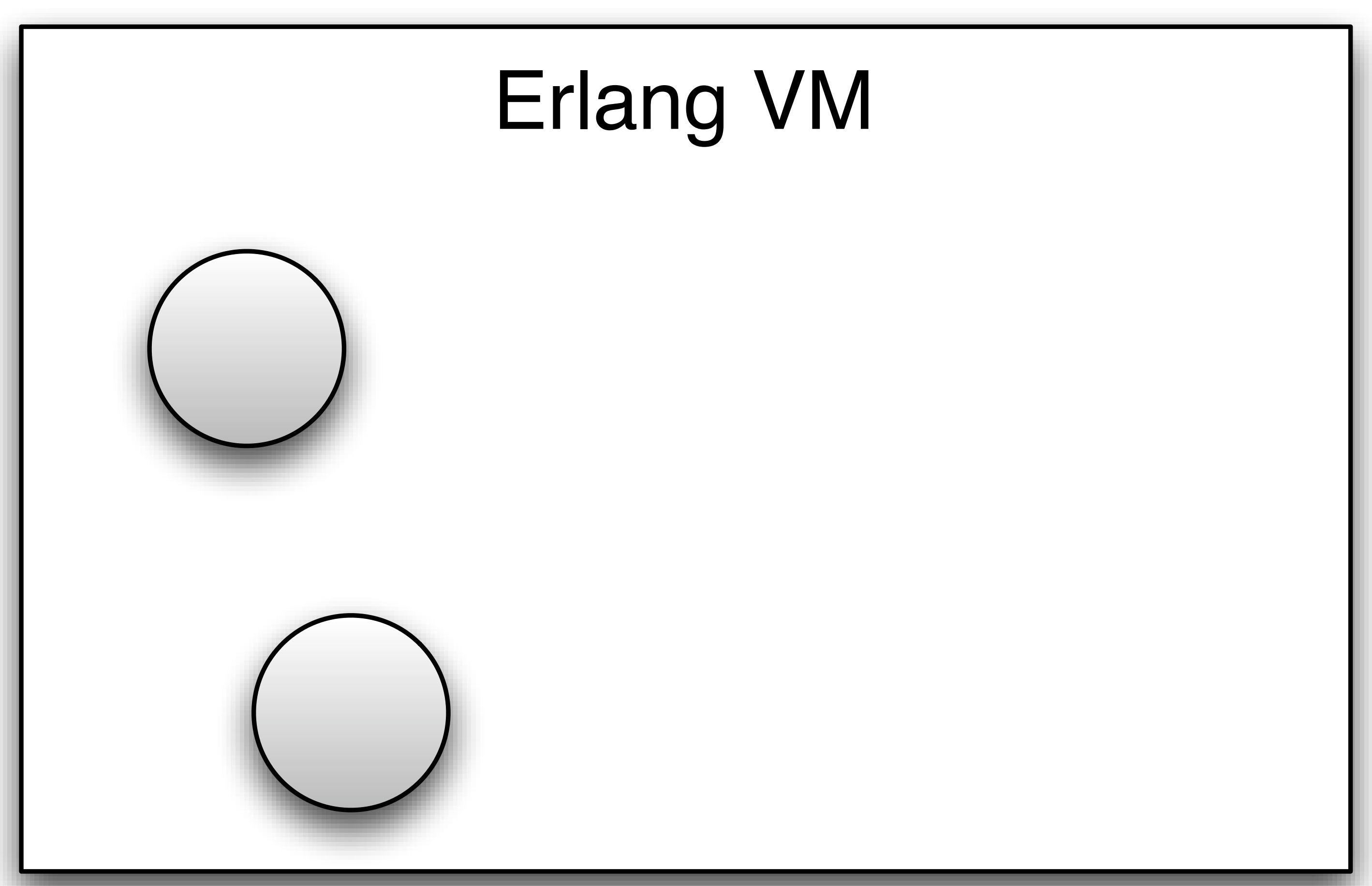
Like an Erlang Process that runs Ruby

Erlang VM

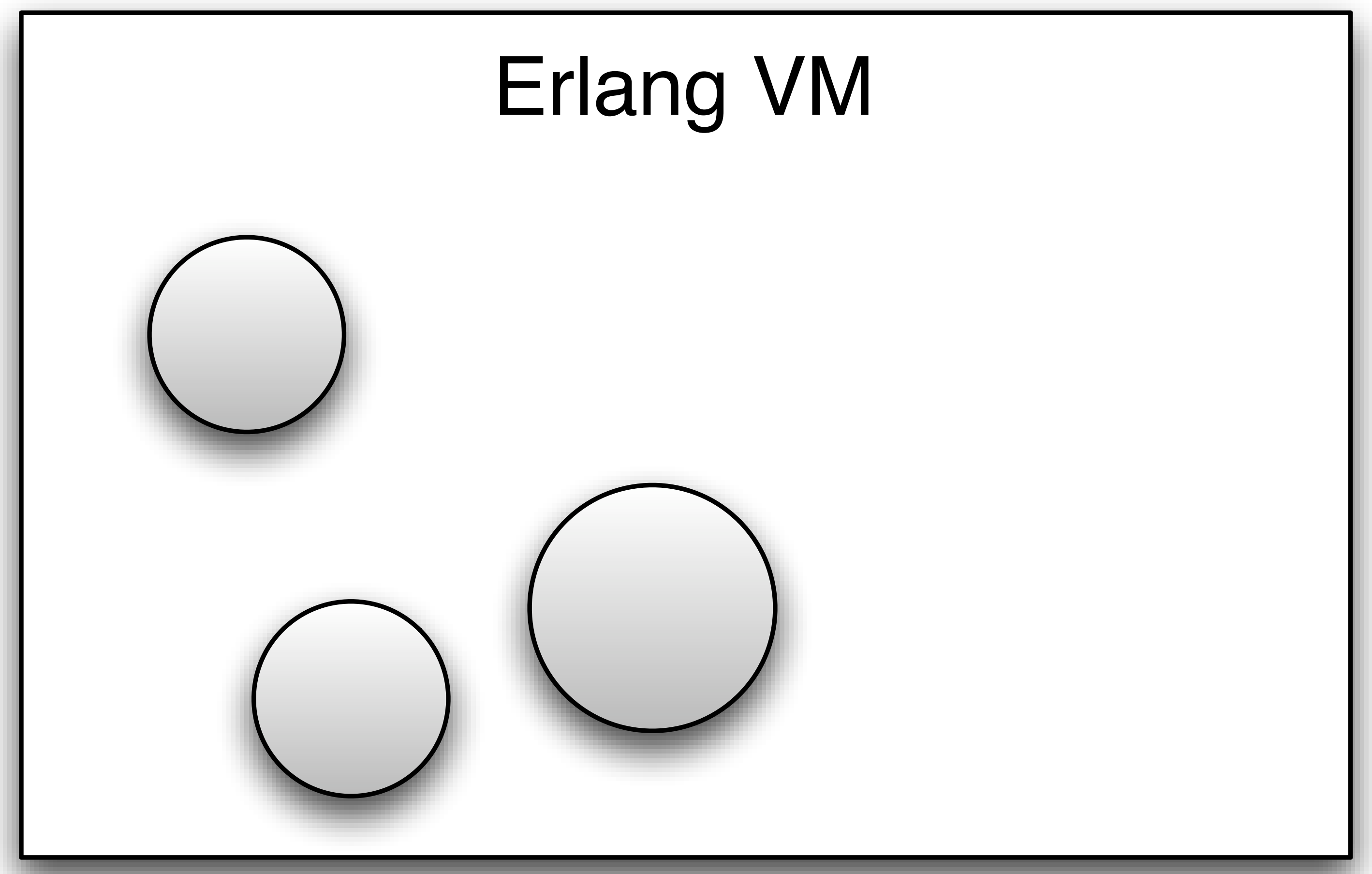
Like an Erlang Process that runs Ruby



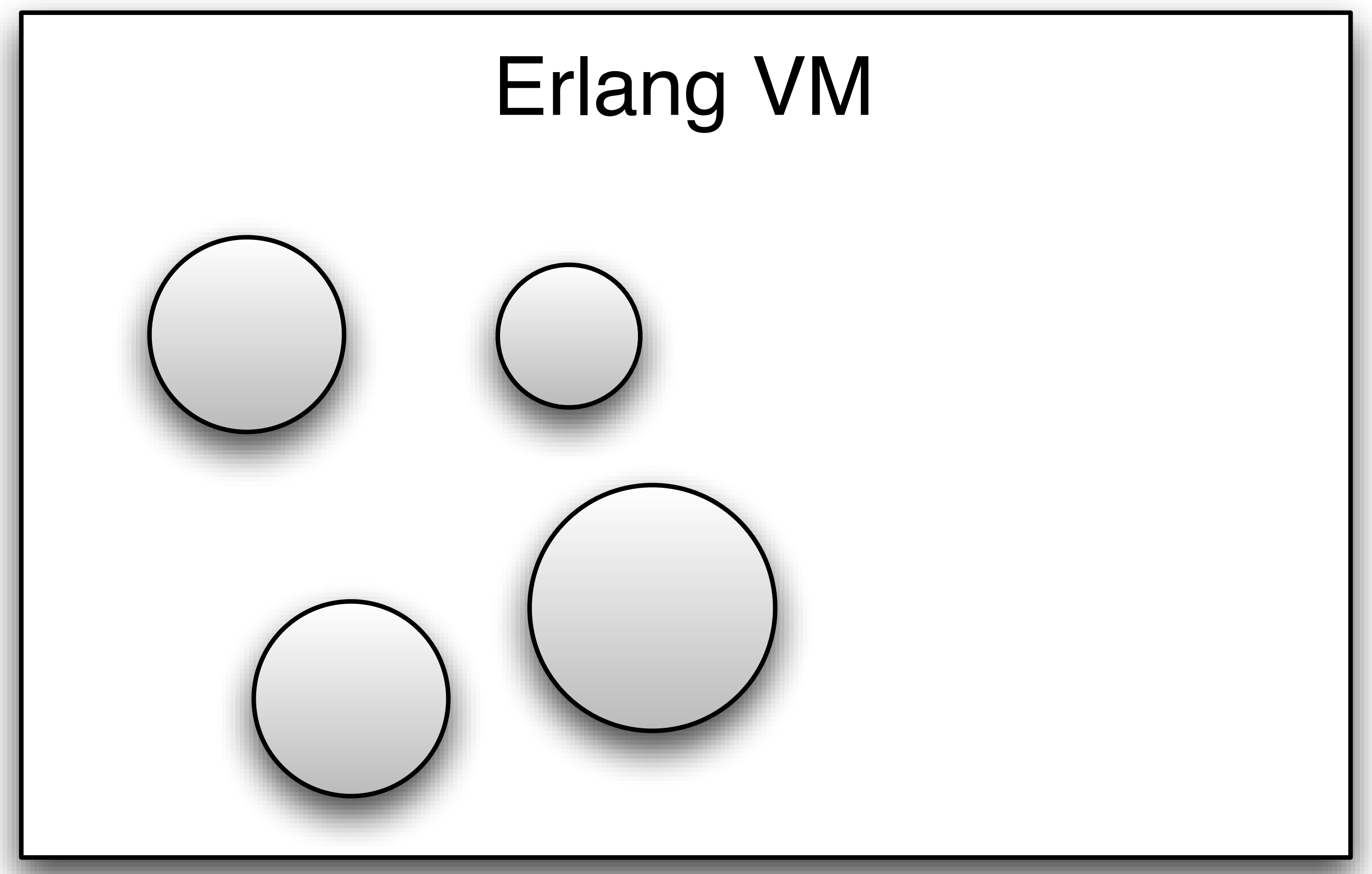
Like an Erlang Process that runs Ruby



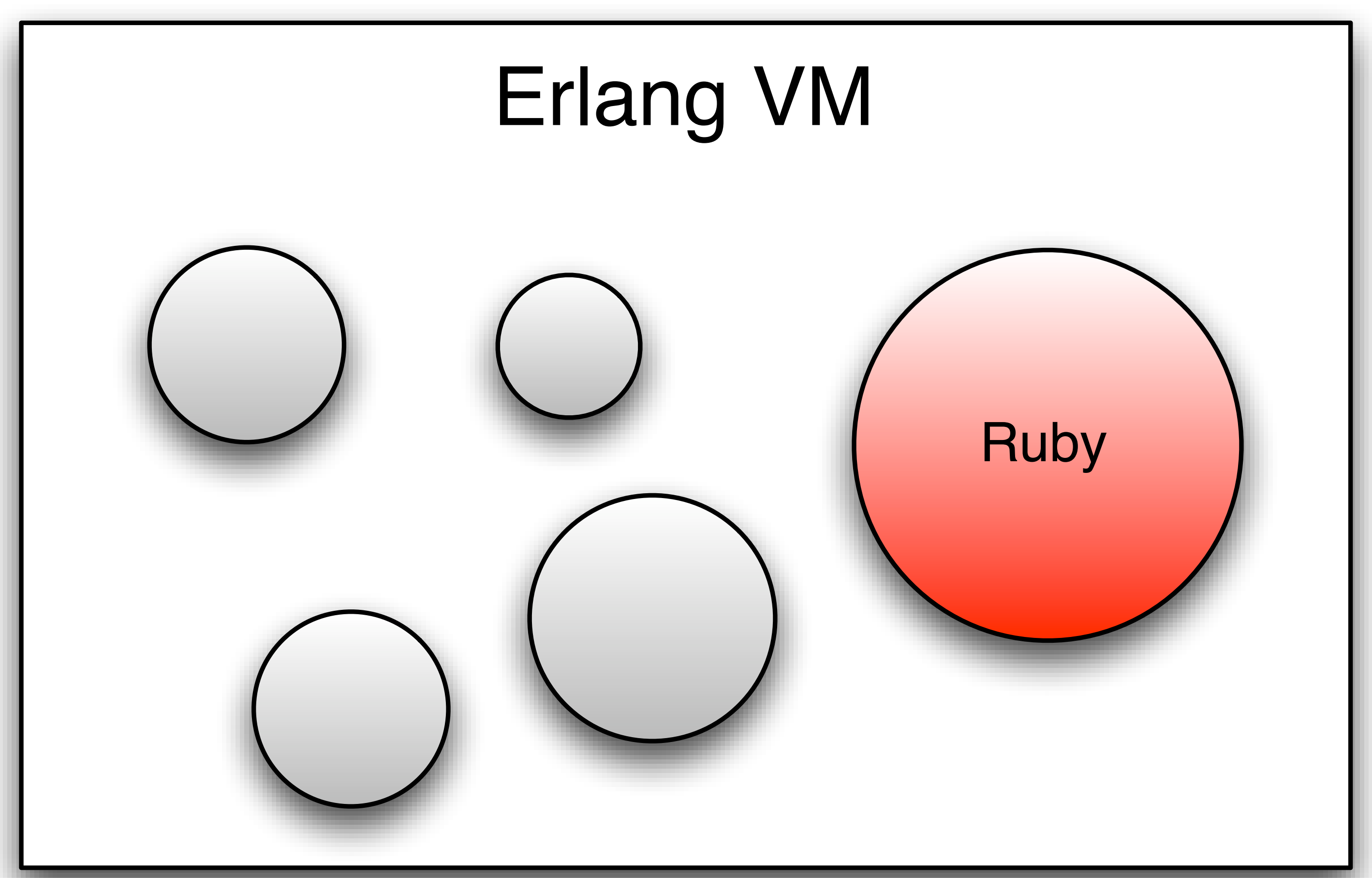
Like an Erlang Process that runs Ruby



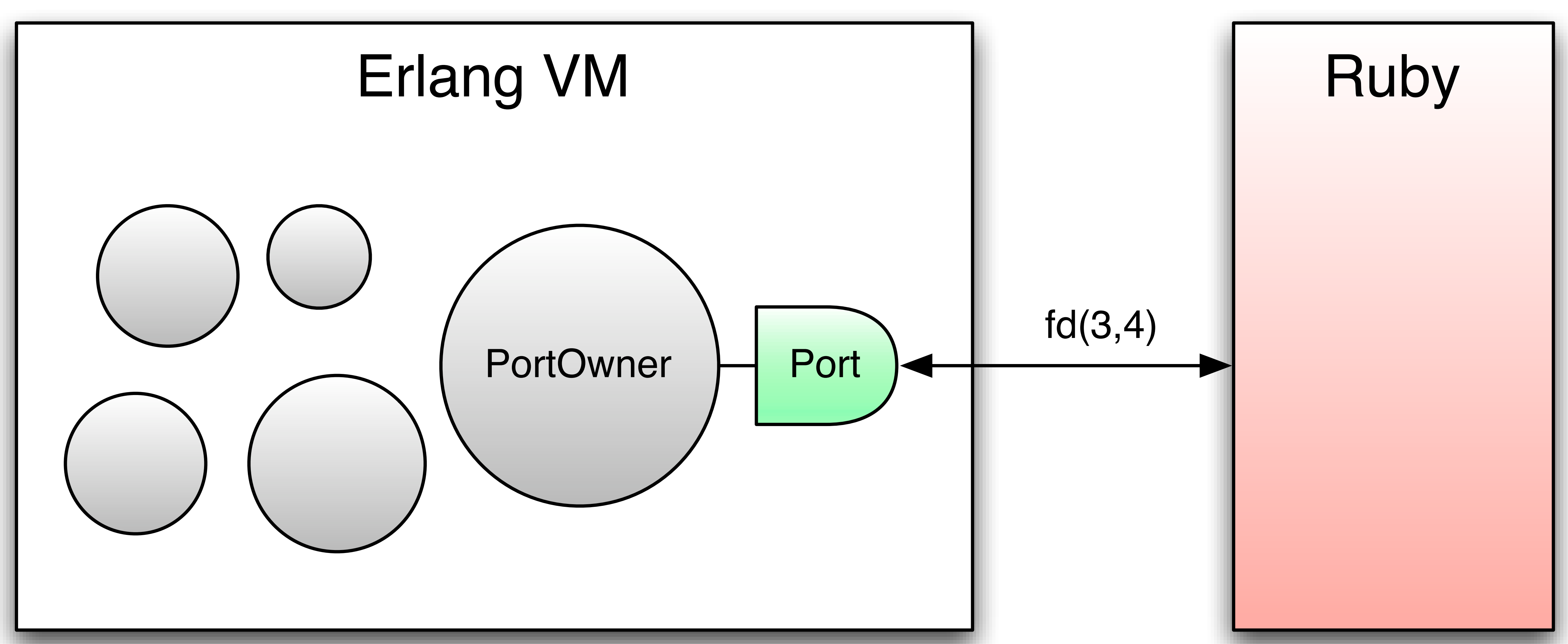
Like an Erlang Process that runs Ruby



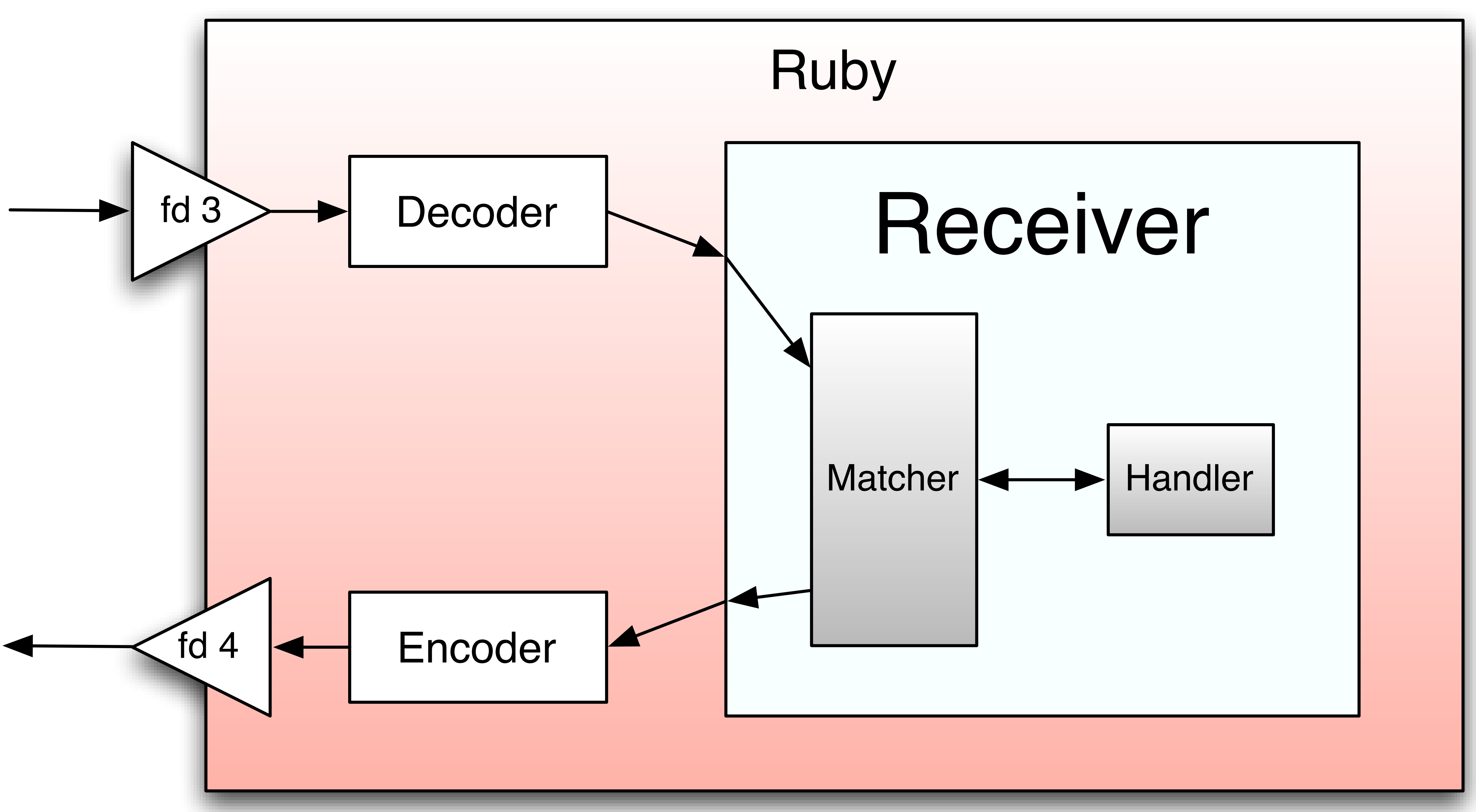
Like an Erlang Process that runs Ruby



What really happens



Inside Ruby



Time to Code

Time to Code

Announcing

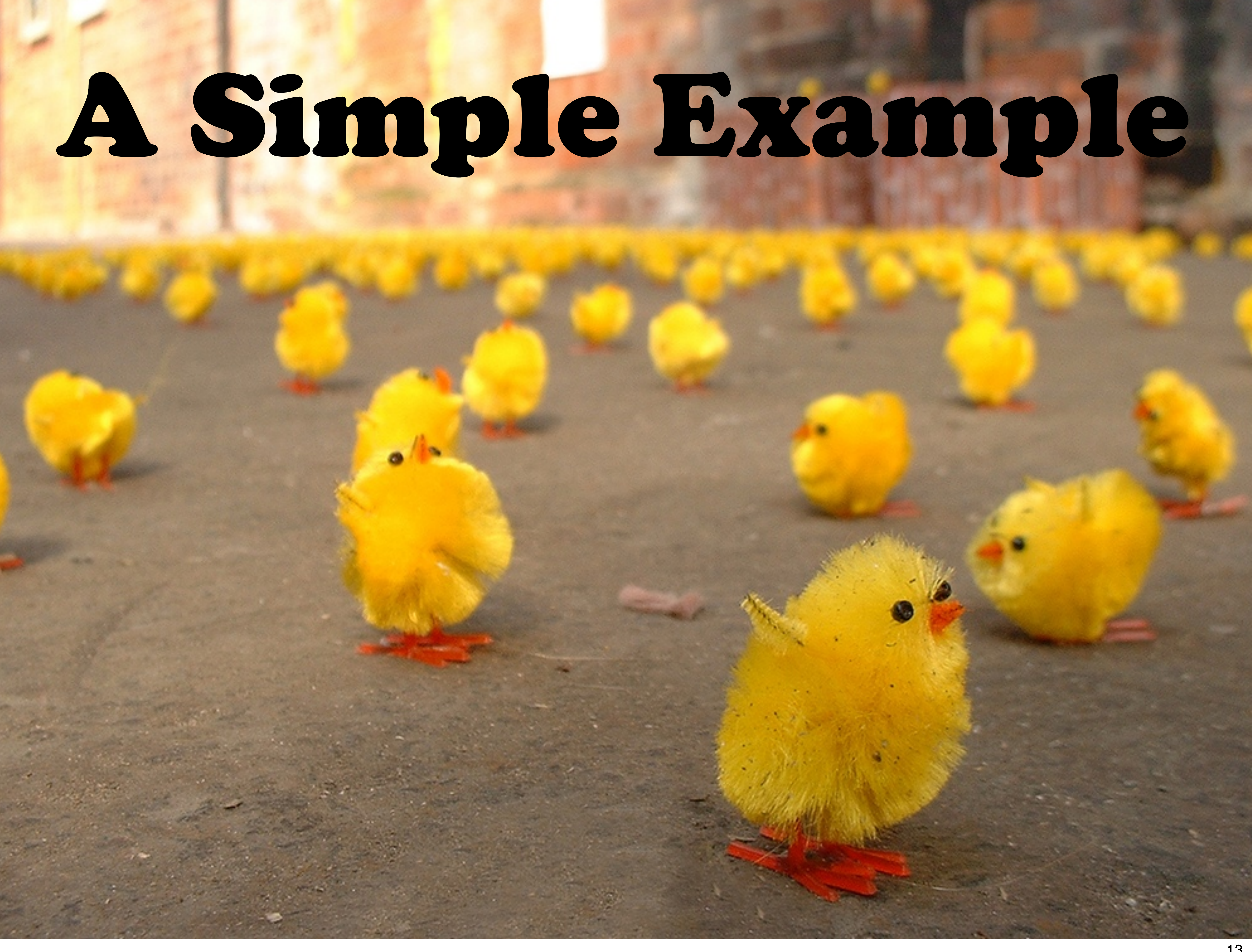
Erlectricity 1.0.0 Release

Time to Code

Announcing

Erlectricity 1.0.0 Release

```
gem install erlectricity
```


A Simple Example

Erlang Code

```
-module(echo).
-export([test/0]).

test() ->
  Cmd = "ruby echo.rb",
  Port = open_port({spawn, Cmd},
                   [{packet, 4},
                    nouse_stdio, exit_status, binary]),
  Payload = term_to_binary({echo, <<"hello world!">>}),
  port_command(Port, Payload),
  receive
    {Port, {data, Data}} ->
      {result, Text} = binary_to_term(Data),
      io:format("~p~n", [Text])
  end.
```

Erlang Code

```
-module(echo).
-export([test/0]).

test() ->
  Cmd = "ruby echo.rb",
  Port = open_port({spawn, Cmd},
                   [{packet, 4},
                    nouse_stdio, exit_status, binary]),
  Payload = term_to_binary({echo, <<"hello world!">>}),
  port_command(Port, Payload),
  receive
    {Port, {data, Data}} ->
      {result, Text} = binary_to_term(Data),
      io:format("~p~n", [Text])
  end.
```

Erlang Code

```
-module(echo).
-export([test/0]).

test() ->
  Cmd = "ruby echo.rb",
  Port = open_port({spawn, Cmd},
                   [{packet, 4},
                    nouse_stdio, exit_status, binary]),
  Payload = term_to_binary({echo, <<"hello world!">>}),
  port_command(Port, Payload),
  receive
    {Port, {data, Data}} ->
      {result, Text} = binary_to_term(Data),
      io:format("~p~n", [Text])
  end.
```

Erlang Code

```
-module(echo).
-export([test/0]).

test() ->
  Cmd = "ruby echo.rb",
  Port = open_port({spawn, Cmd},
                   [{packet, 4},
                    nouse_stdio, exit_status, binary]),
  Payload = term_to_binary({echo, <<"hello world!">>}),
  port_command(Port, Payload),
  receive
    {Port, {data, Data}} ->
      {result, Text} = binary_to_term(Data),
      io:format("~p~n", [Text])
  end.
```

Erlang Code

```
-module(echo).
-export([test/0]).

test() ->
  Cmd = "ruby echo.rb",
  Port = open_port({spawn, Cmd},
                   [{packet, 4},
                    nouse_stdio, exit_status, binary]),
  Payload = term_to_binary({echo, <<"hello world!">>}),
  port_command(Port, Payload),
  receive
    {Port, {data, Data}} ->
      {result, Text} = binary_to_term(Data),
      io:format("~p~n", [Text])
  end.
```

Ruby Code

```
require 'rubygems'
require 'erlectricity'

receive do |f|
  f.when([:echo, String]) do |text|
    f.send!([:result, "You said: #{text}"])
    f.receive_loop
  end
end
```

Ruby Code

```
require 'rubygems'
require 'erlectricity'

receive do |f|
  f.when([:echo, String]) do |text|
    f.send!([:result, "You said: #{text}"])
    f.receive_loop
  end
end
```

Ruby Code

```
require 'rubygems'
require 'erlectricity'

receive do |f|
  f.when([:echo, String]) do |text|
    f.send!([:result, "You said: #{text}"])
    f.receive_loop
  end
end
```

Ruby Code

```
require 'rubygems'
require 'erlectricity'

receive do |f|
  f.when([:echo, String]) do |text|
    f.send!([:result, "You said: #{text}"])
    f.receive_loop
  end
end
```

Ruby Code

```
require 'rubygems'
require 'erlectricity'

receive do |f|
  f.when([:echo, String]) do |text|
    f.send!([:result, "You said: #{text}"])
    f.receive_loop
  end
end
```

Ruby Code

```
require 'rubygems'
require 'erlectricity'

receive do |f|
  f.when([:echo, String]) do |text|
    f.send!([:result, "You said: #{text}"])
    f.receive_loop
  end
end
```

Data Conversion and Matching

```
{echo, <<"hello world!">>}
```

↓

```
[:echo, "hello world!"]
```

~

```
[:echo, String]
```

```
send(Port, test).

f.when(:test)
  { p :ok }

# :ok
```

Erlang ⟶

```
send(Port, test).

f.when(:test)
  { p :ok }

# :ok
```

```
send(Port, test).

f.when(:test)
  { p :ok }

# :ok
```

Ruby →

```
send(Port, test).

f.when(:test)
  { p :ok }
```

Ruby Output → `# :ok`

```
send(Port, {atom, symbol}).

f.when([:atom, Symbol])
  { |sym| p sym }

# :symbol
```

```
send(Port, {number, 1}).

f.when([:number, Fixnum])
  { |num| p num }

# 1
```

```
send(Port, {string, <<"foo">>}).

f.when([:string, String])
  { |str| p str }

# "foo"
```

```
send(Port, {array, [1,2,3]}).

f.when([:array, Array])
  { |arr| p arr }

# [1, 2, 3]
```

```
send(Port, {array,
  [<<"abc">>, <<"def">>]}).

f.when([:array, Array])
  { |arr| p arr }

# ["abc", "def"]
```

```
send(Port, {bool, true}).

f.when([:bool, Erl.boolean])
  { |bool| p bool }

# true
```

```
send(Port, {hash, [{key,val}]}).

f.when([:hash, Erl.hash])
  { |hash| p hash }

# {:key=>:val}
```

```
send(Port, {object,
  {1,{2},3,<<"four">>}}).

f.when([:object, Any])
  { |any| p any }

# [1, [2], 3, "four"]
```

Round Trip Conversions

Erlang	Ruby	Erlang
foo	:foo	foo
true	true	true
false	false	false
1	1	1
1.0	1.0	1.0
<<"baz">>	"baz"	<<"baz">>
"bar"	[98, 97, 114]	{98, 97, 114}
[1, 2, 3]	[1, 2, 3]	{1, 2, 3}

Erlang	Ruby	Erlang
foo	:foo	foo
true	true	true
false	false	false
1	1	1
1.0	1.0	1.0
<<"baz">>	"baz"	<<"baz">>
"bar"	[98, 97, 114]	{98, 97, 114}
[1, 2, 3]	[1, 2, 3]	{1, 2, 3}

Erlang	Ruby	Erlang
foo	:foo	foo
true	true	true
false	false	false
1	1	1
1.0	1.0	1.0
<<"baz">>	"baz"	<<"baz">>
"bar"	[98, 97, 114]	{98, 97, 114}
[1, 2, 3]	[1, 2, 3]	{1, 2, 3}

Erlang	Ruby	Erlang
foo	:foo	foo
true	true	true
false	false	false
1	1	1
1.0	1.0	1.0
<<"baz">>	"baz"	<<"baz">>
"bar"	[98, 97, 114]	{98, 97, 114}
[1, 2, 3]	[1, 2, 3]	{1, 2, 3}

Erlang	Ruby	Erlang
foo	:foo	foo
true	true	true
false	false	false
1	1	1
1.0	1.0	1.0
<<"baz">>	"baz"	<<"baz">>
"bar"	[98, 97, 114]	{98, 97, 114}
[1, 2, 3]	[1, 2, 3]	{1, 2, 3}

Erlang	Ruby	Erlang
foo	:foo	foo
true	true	true
false	false	false
1	1	1
1.0	1.0	1.0
<<"baz">>	"baz"	<<"baz">>
"bar"	[98, 97, 114]	{98, 97, 114}
[1, 2, 3]	[1, 2, 3]	{1, 2, 3}

Erlang	Ruby	Erlang
foo	:foo	foo
true	true	true
false	false	false
1	1	1
1.0	1.0	1.0
<<"baz">>	"baz"	<<"baz">>
"bar"	[98, 97, 114]	{98, 97, 114}
[1, 2, 3]	[1, 2, 3]	{1, 2, 3}

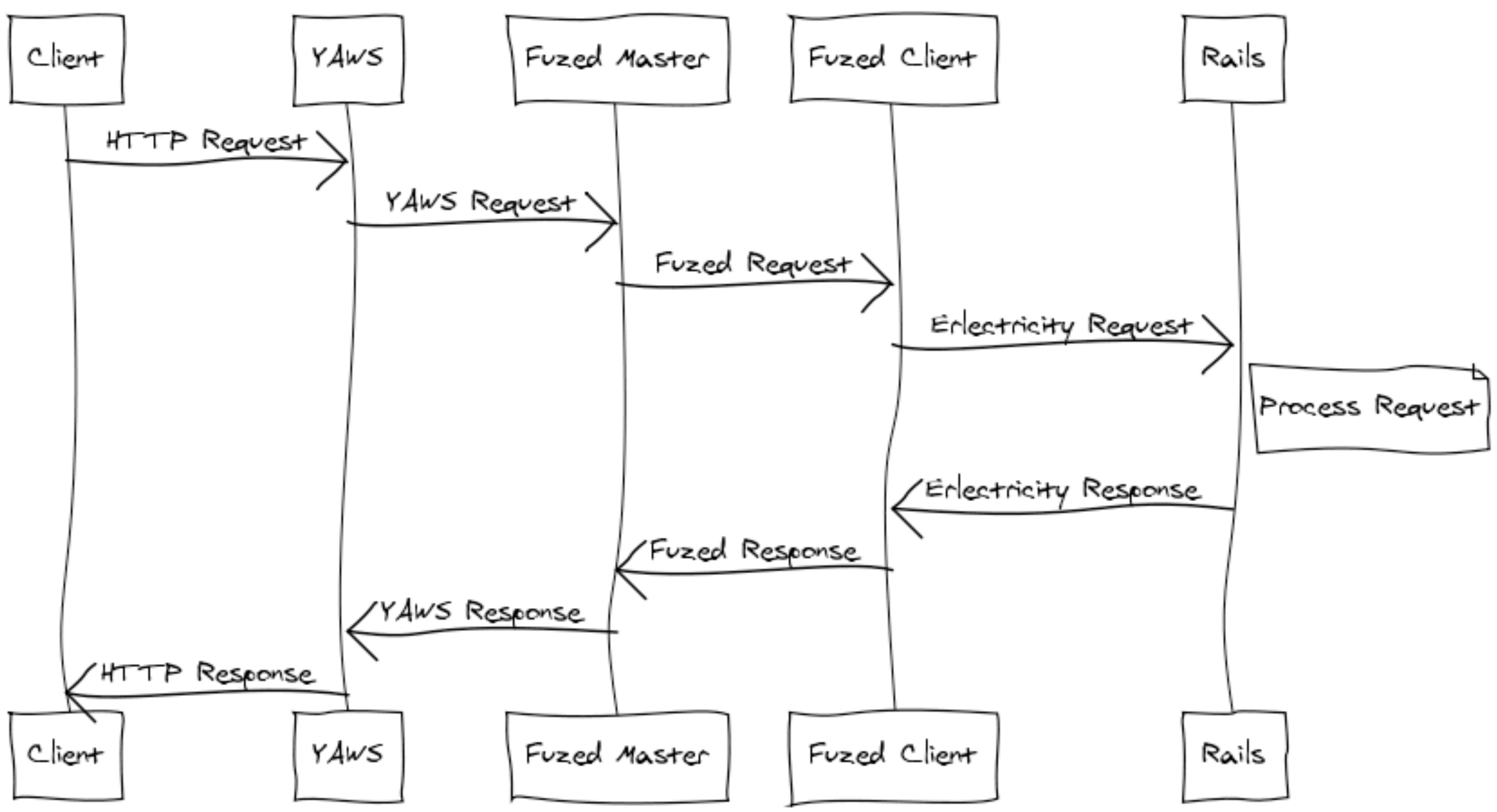
Erlang	Ruby	Erlang
foo	:foo	foo
true	true	true
false	false	false
1	1	1
1.0	1.0	1.0
<<"baz">>	"baz"	<<"baz">>
"bar"	[98, 97, 114]	{98, 97, 114}
[1, 2, 3]	[1, 2, 3]	{1, 2, 3}

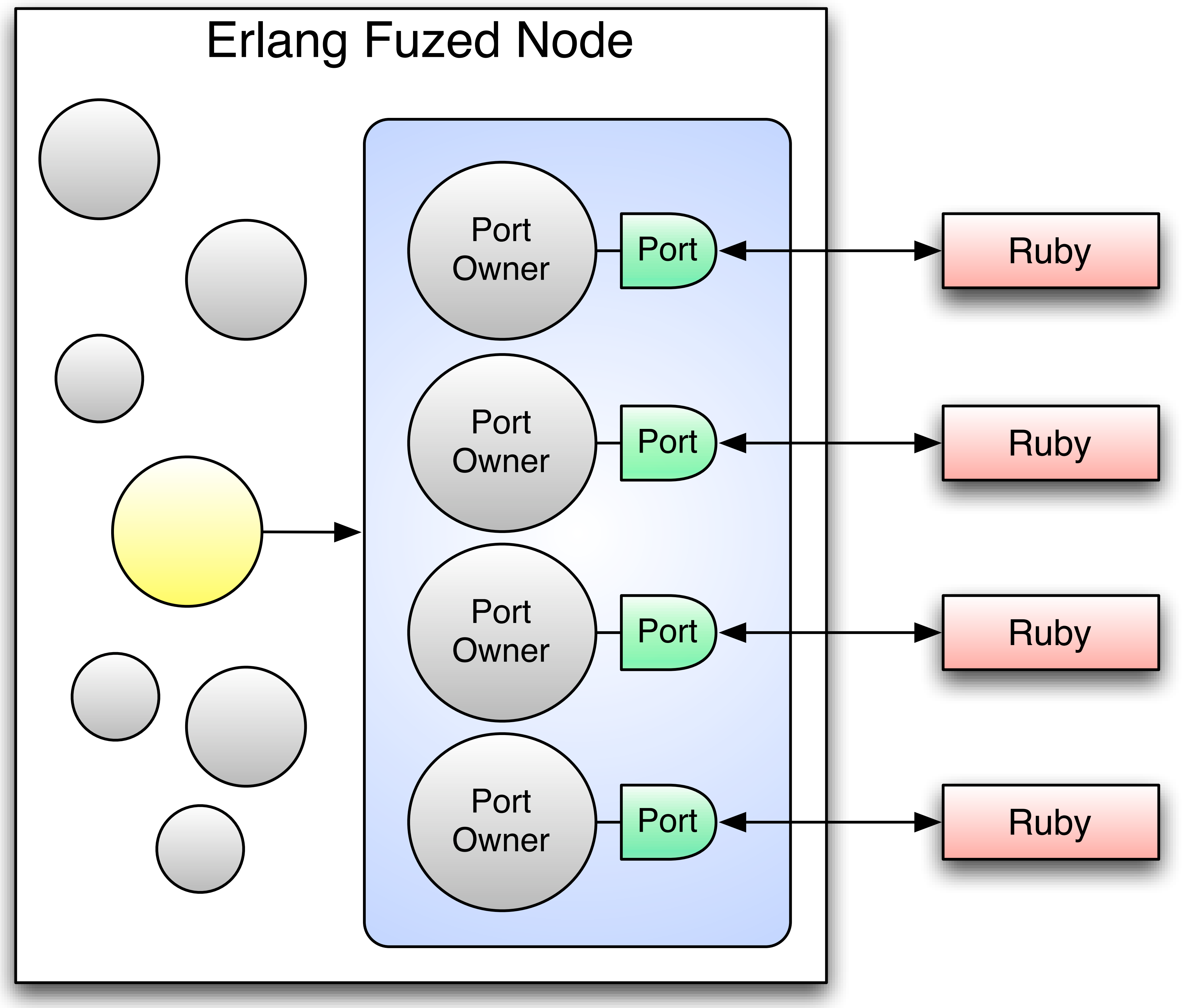
Erlang	Ruby	Erlang
foo	:foo	foo
true	true	true
false	false	false
1	1	1
1.0	1.0	1.0
<<"baz">>	"baz"	<<"baz">>
"bar"	[98, 97, 114]	{98, 97, 114}
[1, 2, 3]	[1, 2, 3]	{1, 2, 3}

EITHER BE A GOOD
EXAMPLE OR A
HORRIBLE WARNING
MOUNT
TIRE

CHADDAVIS

Fuzed




Erlang Fuzed Node
Port Owner
Port
Ruby
Port Owner
Port
Ruby
Port Owner
Port
Ruby
Port Owner
Port
Ruby

HAVE
NOTHING
EXCEPT MY
GENIUS

The most beautiful binary format of all time

Port Invocation

```
open_port({spawn, Cmd},
          [{packet, 4},
           nouse_stdio,
           exit_status,
           binary]).
```

Term Transfer Protocol

```
[1, 2, 3]
```

Four Byte Packet Length

```
<<0,0,0,7,131,107,0,3,1,2,3>>
```

Max Packet Size

256^4 bytes

=

4,294,967,296 bytes

=

4 GiB

One Byte “Magic Number”

```
<<0,0,0,7,131,107,0,3,1,2,3>>
```

One Byte Type

```
<<0,0,0,7,131,107,0,3,1,2,3>>
```

Type Lookup

```
SMALL_INT = 97            PID = 103
INT = 98                  SMALL_TUPLE = 104
SMALL_BIGNUM = 110        LARGE_TUPLE = 105
LARGE_BIGNUM = 111        NIL = 106
FLOAT = 99                STRING = 107
ATOM = 100                LIST = 108
REF = 101                 BIN = 109
NEW_REF = 114             FUN = 117
PORT = 102                NEW_FUN = 112
```

Type Lookup

```
SMALL_INT = 97          PID = 103
INT = 98                SMALL_TUPLE = 104
SMALL_BIGNUM = 110      LARGE_TUPLE = 105
LARGE_BIGNUM = 111      NIL = 106
FLOAT = 99              STRING = 107
ATOM = 100              LIST = 108
REF = 101               BIN = 109
NEW_REF = 114           FUN = 117
PORT = 102              NEW_FUN = 112
```

Two Byte Data Length

```
<<0,0,0,7,131,107,0,3,1,2,3>>
```

Data Content

```
<<0,0,0,7,131,107,0,3,1,2,3>>
```

Try it at home!

```
Eshell V5.6.4  (abort with ^G)
1> term_to_binary([1, 2, 3]).
<<131,107,0,3,1,2,3>>
```

A bit more complex

```
{ok, [99, <<”beers”>>]}
```

A bit more complex

```
{ok, [99, <<"beers">>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<"beers">>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<”beers”>>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<”beers”>>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<"beers">>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<"beers">>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<”beers”>>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<”beers”>>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<”beers”>>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<"beers">>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<"beers">>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<”beers”>>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<"beers">>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

A bit more complex

```
{ok, [99, <<"beers">>]}
```

```
<<131,104,2,100,0,2,111,
  107,108,0,0,0,2,97,99,
  109,0,0,0,5,98,101,
  101,114,115,106>>
```

```
1> A = fun() -> 42 end.
#Fun<erl_eval.20.67289768>

2> B = term_to_binary(A).
<<131,112,0,0,2,139,0,213,76,172,89,236,217,232,254,
184,178,23,144,76,140,192,206,0,0,0,1,0,0,...>>

3> size(B).
653

4> C = binary_to_term(B).
#Fun<erl_eval.20.67289768>

5> apply(C, []).
42
```

Erlectricity

Erlectricity

=

Erlectricity

=

The scalability of Erlang

Erlectricity

=

The scalability of Erlang

+

Erlectricity

=

The scalability of Erlang

+

The ease of Ruby

Towards Erlectricity

See Also

Erlix

github.com/KDr2/erlix

Rebar

github.com/mojombo/rebar

Binary Erlang Term

Binary Erlang Term

BERT

```
[1, 2, 3]
```

Term

BERT

```
<<131,107,0,3,1,2,3>>
```

Simple Data Types Only

```
SMALL_INT = 97
INT = 98
FLOAT = 99
ATOM = 100
SMALL_BIGNUM = 110
LARGE_BIGNUM = 111
SMALL_TUPLE = 104
LARGE_TUPLE = 105
NIL = 106
STRING = 107
LIST = 108
BIN = 109
```

highly compact binary

easy to parse

not human readable

generic data types

Binary Erlang Term - Remote Procedure Call

Binary Erlang Term - Remote Procedure Call

BERT-RPC

BERT-RPC Basics

BERT-RPC Call

```
{call, myapp, add, [1, 2]}
```

↓

```
{reply, 3}
```

BERT-RPC Call

```
{call, myapp, add, [1, 2]}
```

↓

```
{reply, 3}
```

BERT-RPC Call

```
{call, myapp, add, [1, 2]}
```

↓

```
{reply, 3}
```

BERT-RPC Call

```
{call, myapp, add, [1, 2]}
```

↓

```
{reply, 3}
```

BERT-RPC Call

```
{call, myapp, add, [1, 2]}
```

↓

```
{reply, 3}
```

BERT-RPC Call

```
{call, myapp, add, [1, 2]}
```

↓

```
{reply, 3}
```

BERT-RPC Call

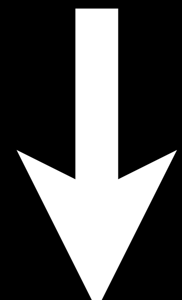
```
{call, myapp, add, [1, 2]}
```

↓

```
{reply, 3}
```

BERT-RPC Call

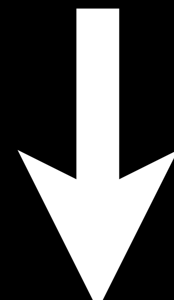
```
{call, myapp, add, [1, 2]}
```



```
-or-
```

BERT-RPC Call

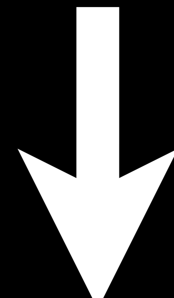
```
{call, myapp, add, [1, 2]}
```



```
{error, [server, 2, “unknown module”]}
```

BERT-RPC Call

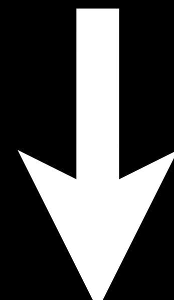
```
{call, myapp, add, [1, 2]}
```



```
{error, [server, 2, “unknown module”]}
```

BERT-RPC Call

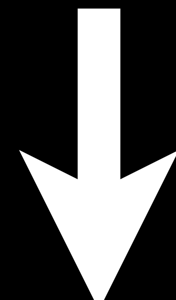
```
{call, myapp, add, [1, 2]}
```



```
{error, [server, 2, “unknown module”]}
```

BERT-RPC Call

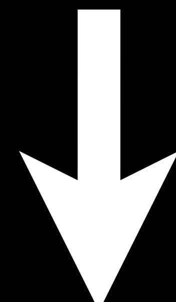
```
{call, myapp, add, [1, 2]}
```



```
{error, [server, 2, “unknown module”]}
```

BERT-RPC Call

```
{call, myapp, add, [1, 2]}
```



```
{error, [server, 2, “unknown module”]}
```

BERT Packet (BERTP)

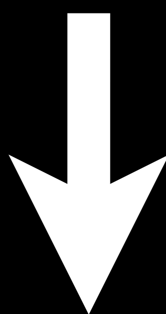
BERTP = 4 byte length header + BERT

```
{call, myapp, add, [1, 2]}
```

```
<<0,0,0,29,131,104,4,100,0,4,99,97,108,108,100,0,5,109,121,
97,112,112,100,0,3,97,100,100,107,0,2,1,2>>
```

Transfer via BERTP

```
{call, myapp, add, [1, 2]}
<<0,0,0,29,131,104,4,100,0,4,99,97,108,108,100,0,5,109,121,
97,112,112,100,0,3,97,100,100,107,0,2,1,2>>
```



```
<<0,0,0,13,131,104,2,100,0,5,114,101,112,108,121,97,3>>
{reply, 3}
```

BERT-RPC Cast

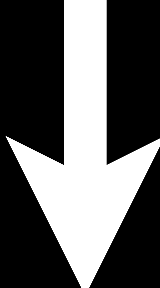
```
{cast, myapp, incr, [5]}
```

↓

```
{noreply}
```

BERT-RPC Cast

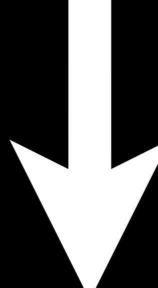
```
{cast, myapp, incr, [5]}
```



```
{noreply}
```

BERT-RPC Cast

```
{cast, myapp, incr, [5]}
```



```
{noreply}
```

BERT-RPC Cast

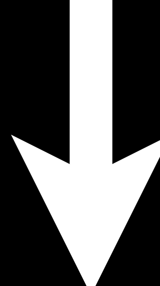
```
{cast, myapp, incr, [5]}
```

↓

```
{noreply}
```

BERT-RPC Cast

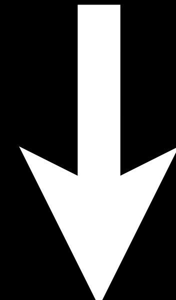
```
{cast, myapp, incr, [5]}
```



```
{noreply}
```

Optional Opaque ID

```
{call, myapp, add, [1, 2],
    [{id, “fc1e542”}]}
```

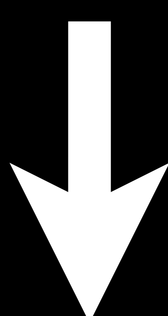


```
{reply, 3, [{id, “fc1e542”}]}
```

BERT-RPC
Streaming

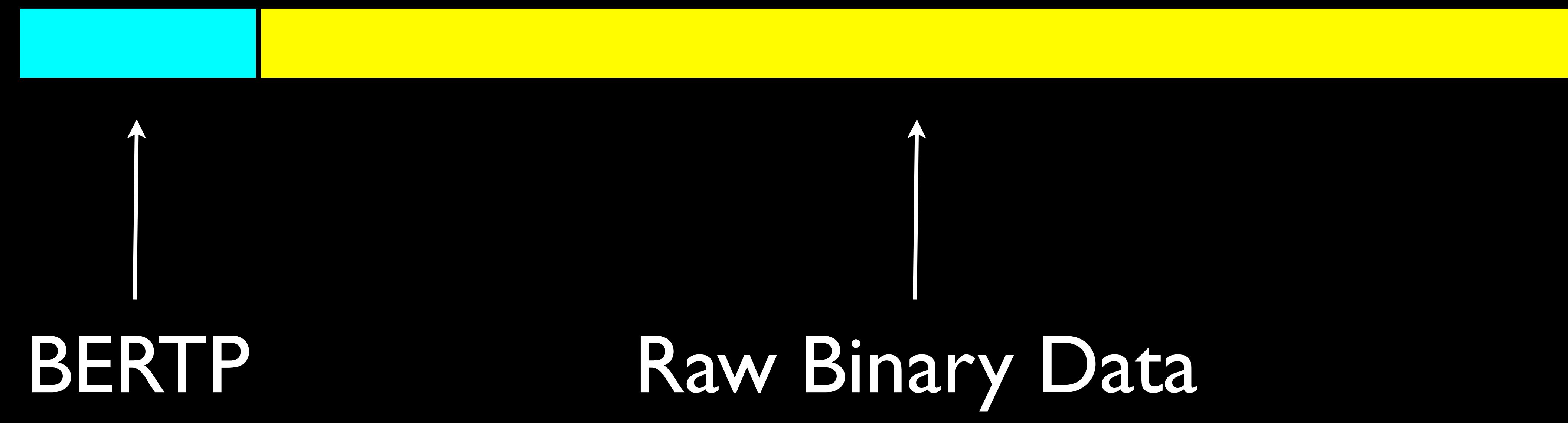
Known Length Streaming Reply

```
{call, myapp, log, [“a.log”]}
```



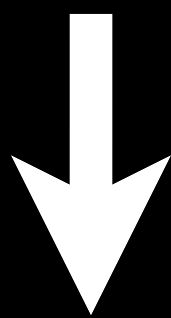
```
<<0,0,0,59,131,104,3,100,0,5,114,101,112,108,121,106,108,0,0,0,1,104,3,100,0,6,115,116,
114,101,97,109,100,0,4,98,121,116,101,108,0,0,0,1,104,2,100,0,6,108,101,110,
103,116,104,98,0,4,1,134,106,106,73,44,32,91,50,48,48,57,45,48,52,45,50,51,...>>
{reply, [], [{stream, byte,
[{length, 262534}]}]}
<byte-0>...<byte-262533>
```

Known Length Streaming Reply



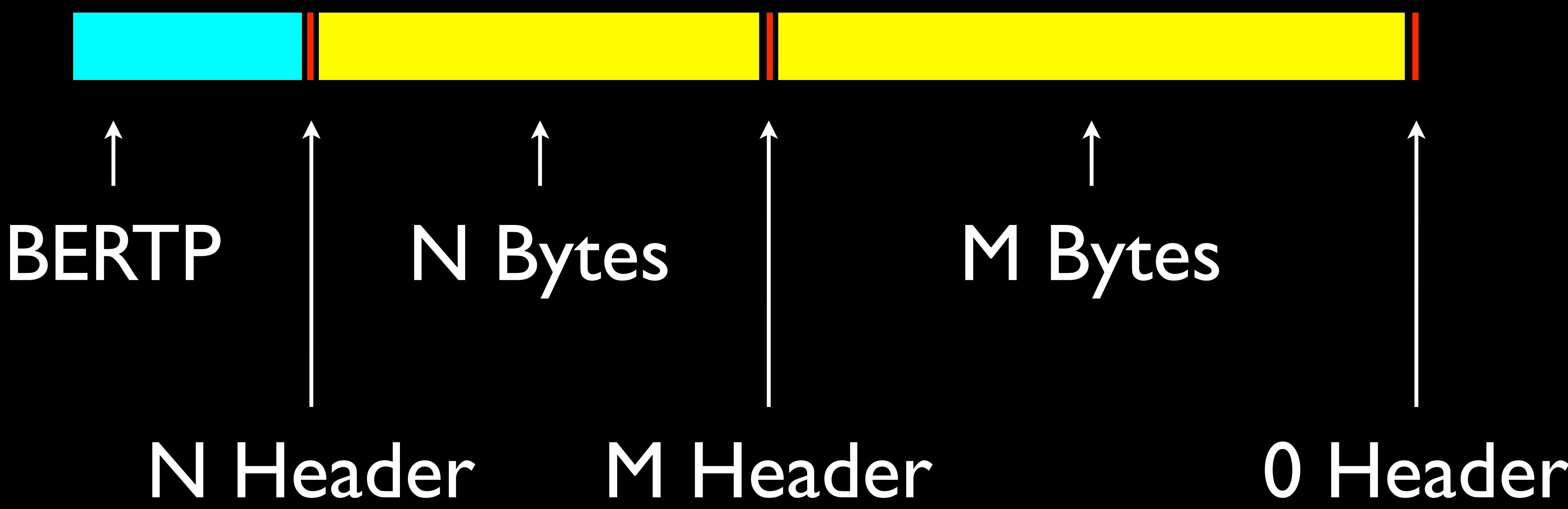
Chunked Streaming Reply

```
{call, myapp, log, ["a.log"]}
```



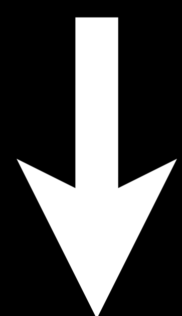
```
<<0,0,0,59,131,104,3,100,0,5,114,101,112,108,121,106,108,0,0,0,1,104,3,100,0,6,115,116,
114,101,97,109,100,0,4,98,121,116,101,108,0,0,0,1,104,2,100,0,6,108,101,110,
103,116,104,98,0,4,1,134,106,106,0,0,7,38,73,44,32,91,50,52,45,50,51,...,0,0,0,0>>
{reply, [], [{stream, byte, chunked}]}
<n-header><n-bytes><m-header><m-bytes>
<0-header>
```

Chunked Streaming Reply



BERT Streaming Reply

{call, myapp, friends, ["mojombo"]}

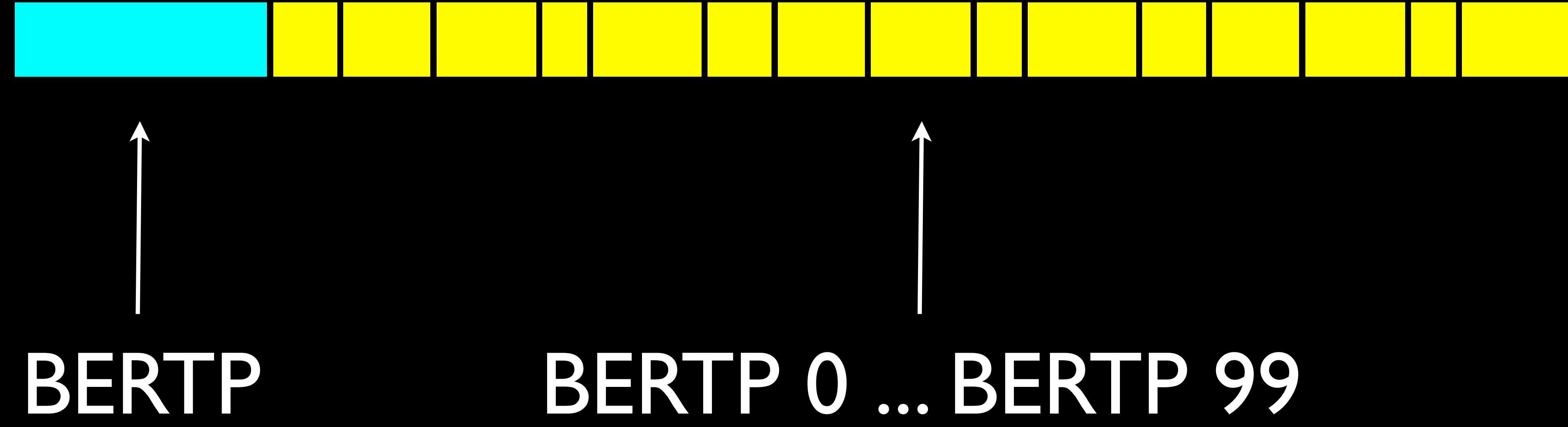


<<0,0,0,59,131,104,3,100,0,5,114,101,112,108,121,106,108,0,0,0,1,104,3,100,0,6,115,116,
114,101,97,109,100,0,4,98,121,116,101,108,0,0,0,1,104,2,100,0,6,108,101,110,103,116,
104,98,0,4,1,134,106,106,0,0,0,29,131,104,2,100,0,4,117,115,101,114,104,2,100,0,4,110,97,
109,101,107,0,7,109,111,106,111,109,98,111,0,0,0,35,131,104,2,100,...>>

{reply, [], [{stream, bert,
[{length, 100}]}]}

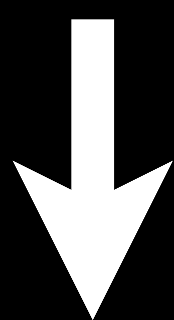
<BERTP-0>...<BERTP-99>

BERT Streaming Reply



Terminated BERT Streaming Reply

{call, myapp, friends, [“mojombo”]}

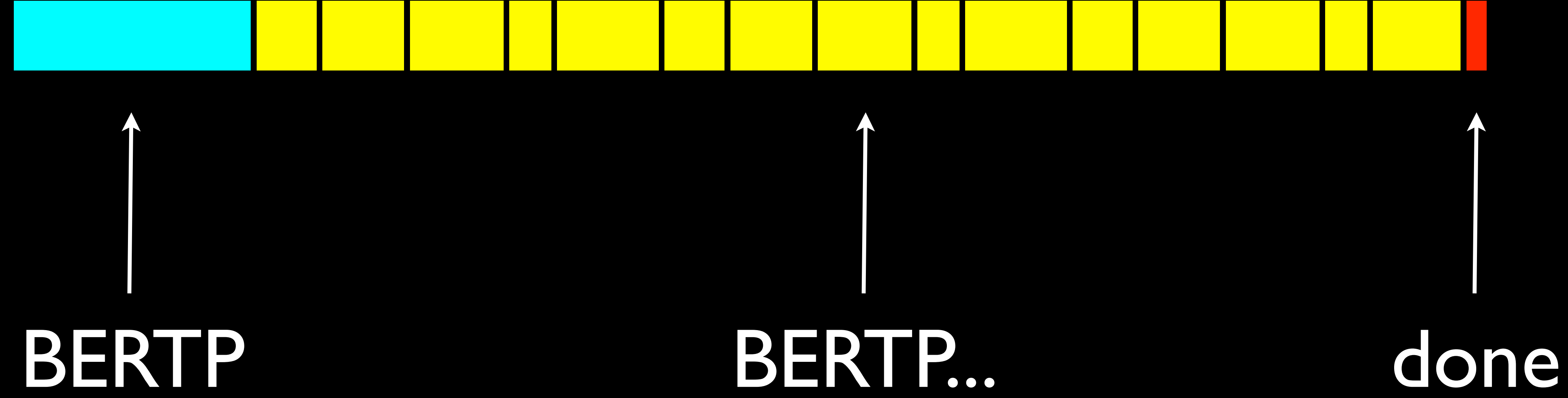


<<0,0,0,59,131,104,3,100,0,5,114,101,112,108,121,106,108,0,0,0,1,104,3,100,0,6,115,116,
114,101,97,109,100,0,4,98,121,116,101,108,0,0,0,1,104,2,100,0,6,108,101,110,103,116,
104,98,0,4,1,134,106,106,0,0,0,29,131,104,2,100,0,4,117,115,101,114,104,2,100,0,4,110,97,
109,101,107,0,7,109,111,106,111,109,98,111,0,0,0,35,131,104,2,100,...>>

{reply, [], [{stream, bert,
[{terminator, done}]}]}

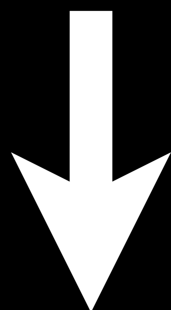
<BERTP>...done

BERT Streaming Reply



BERT Streaming Request

```
{call, myapp, samples, [],
[{stream, bert, [{length, 100}]}]}
<BERTP 0>...<BERTP 99>
```

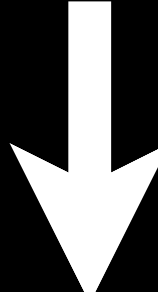


```
{reply, {ok, 100}}
```

BERT-RPC Exotica

Persistent Connections

```
{persistence, enable}
```



```
{result, enabled}
```

Asynchronous Call

```
{call, myapp, factor, [161],
[{async, “bert.example.com:9377”},
{id, “7af639b”}]}
```

↓ ↓

```
{reply, async,                 {reply, [7, 23],
[{id, “7af639b”}]}             [{id, “7af639b”}]}
```

Tom Preston-Werner

http://tom.preston-werner.com/

@mojombo on Twitter

github.com/mojombo

Credits

http://www.flickr.com/photos/cayusa/981372736/

http://www.flickr.com/photos/chadwho1ders/2722656894/

http://www.flickr.com/photos/strph/2626298199/

http://www.flickr.com/photos/dominicspics/1149242842/

http://www.flickr.com/photos/mysterybee/1659336160/

http://www.flickr.com/photos/mattpelletier/437061827/

http://www.flickr.com/photos/coljay72/2399545998/

http://www.flickr.com/photos/major_clanger/4850772/