

# The Erlang VM

or

How I Stopped Worrying  
And Started Loving Parallel Programming

Happi  
Erik Stenman



- Programming since 1980
- Erlang since 1994
- First native code compiler for Erlang
- HIPE
- Project Manager for Scala 1.0
- 2005-2010 CTO @ Klarna
- Chief Scientist @ Klarna
- Writing a book about the Erlang VM



# The Erlang VM

or

How I Stopped Worrying  
And Started Loving Parallel Programming



## DISCLAIMER

# Today's Most Boring Talk

*I don't like opinions!*

try to have only three:

Don't have opinions.

XSLT sucks ~~is~~ not that good.

Operator overloading is bad.

I will barely show any code.

It's how you think!



Now Imagine this!

# Entrepreneurs with an idea

Let's build  
A  
web service!





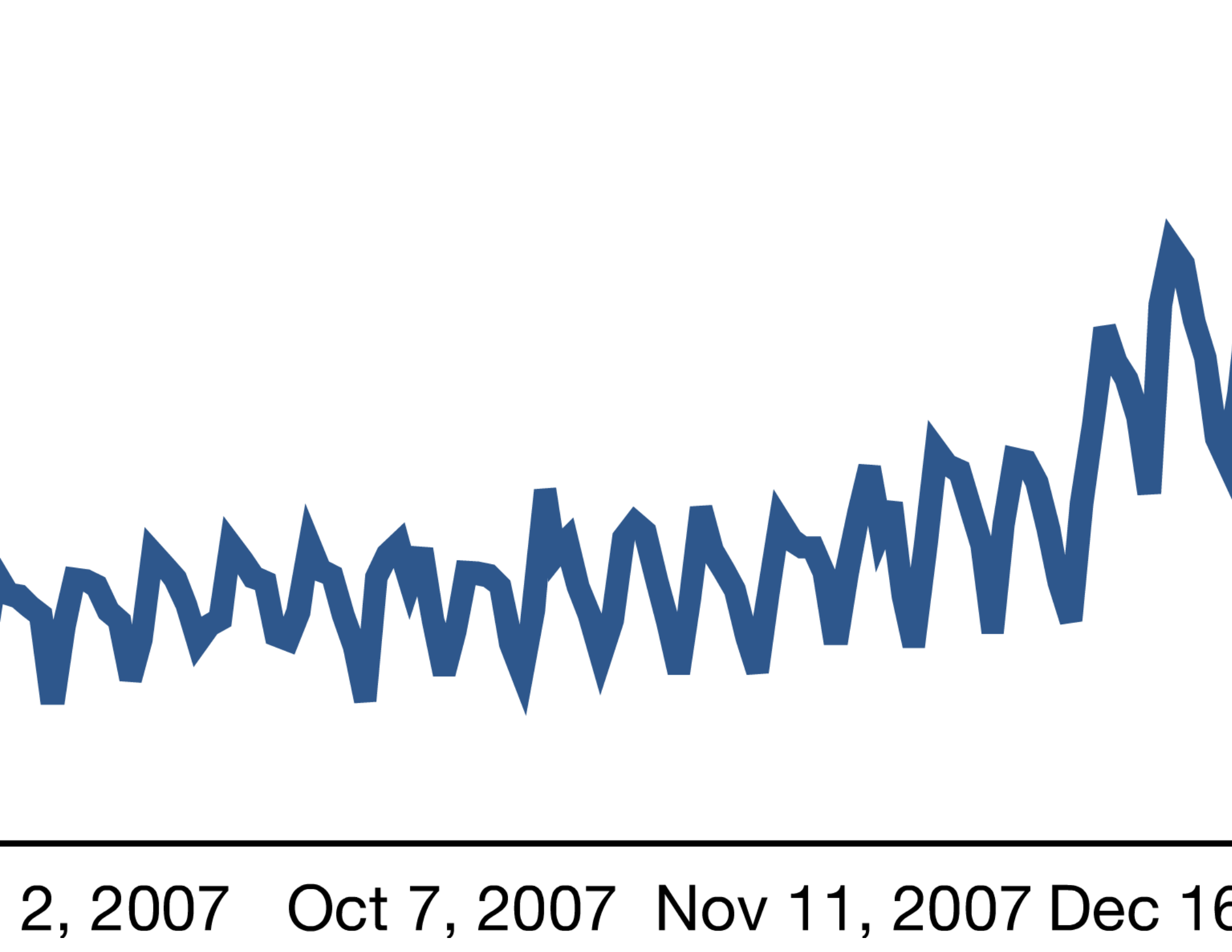
Apr 10, 2005 May 15, 2

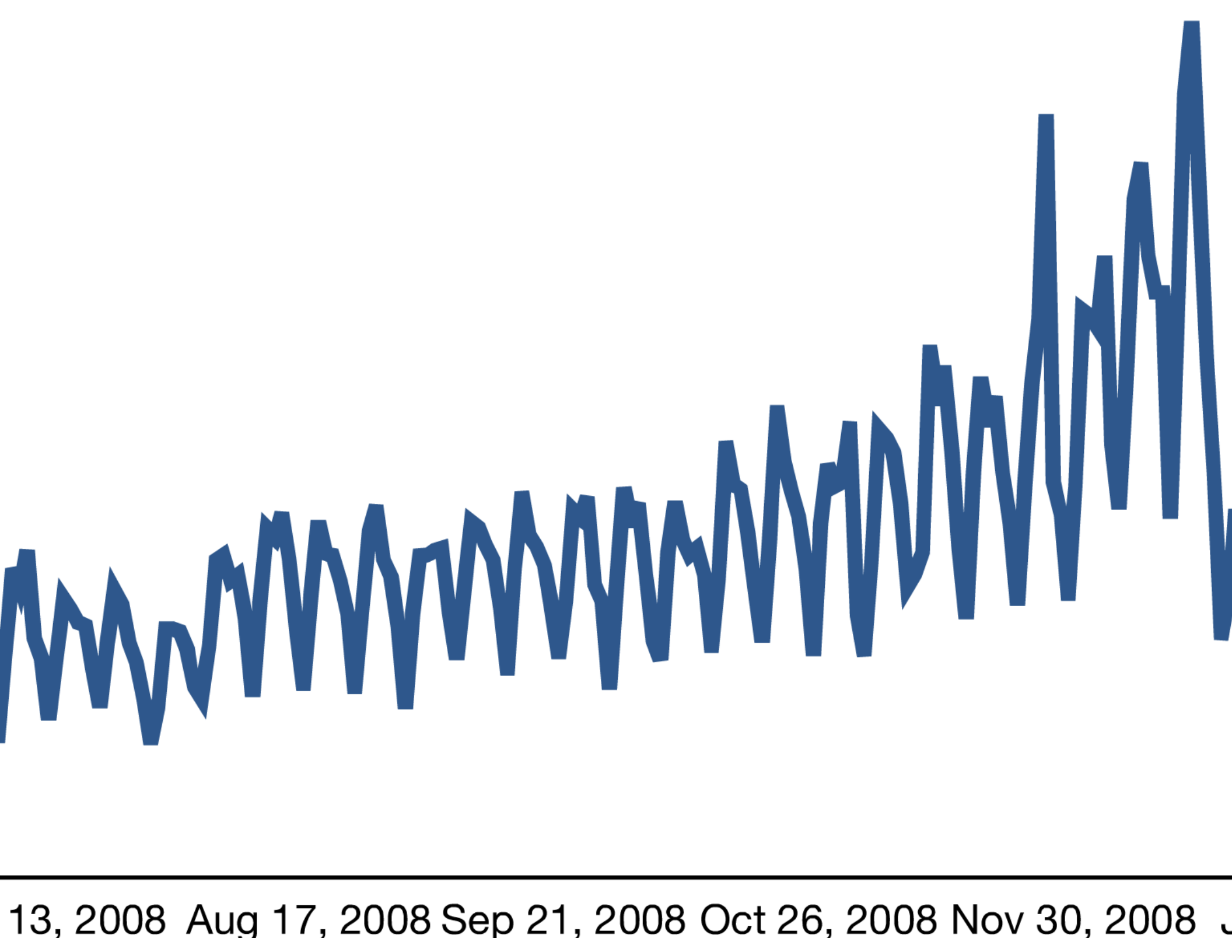


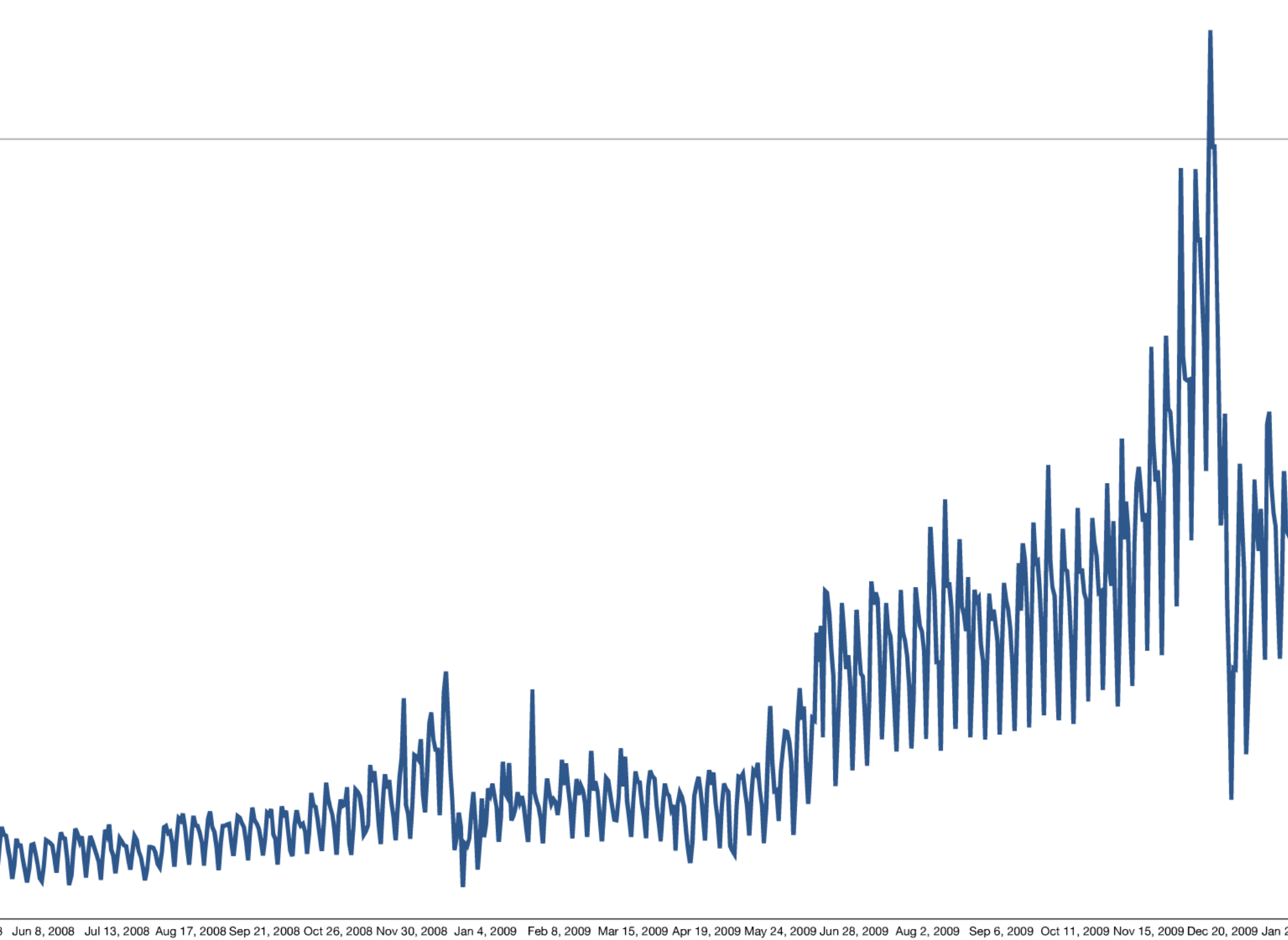
Dec 11, 2



6 Oct 22, 2006 Nov 26, 2006 De

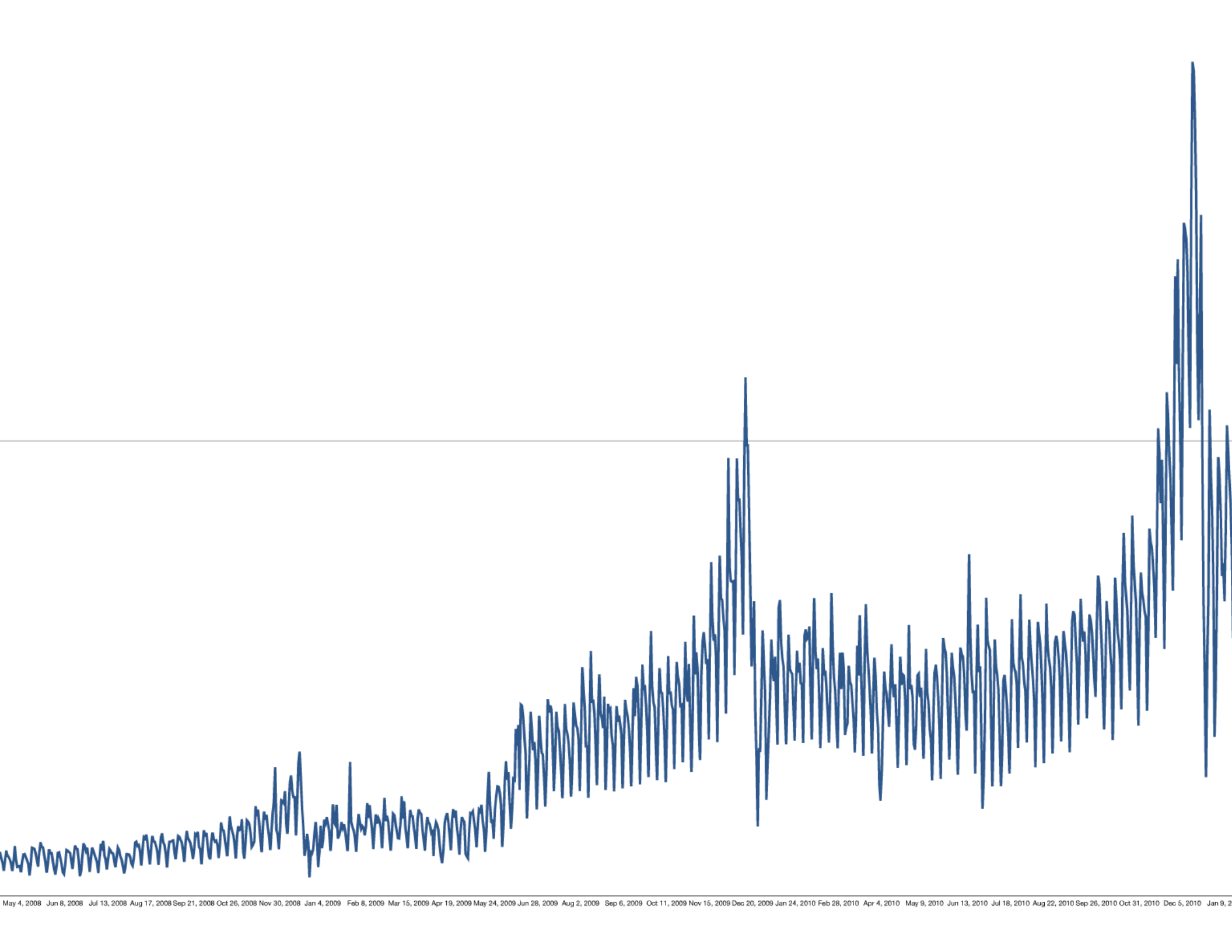


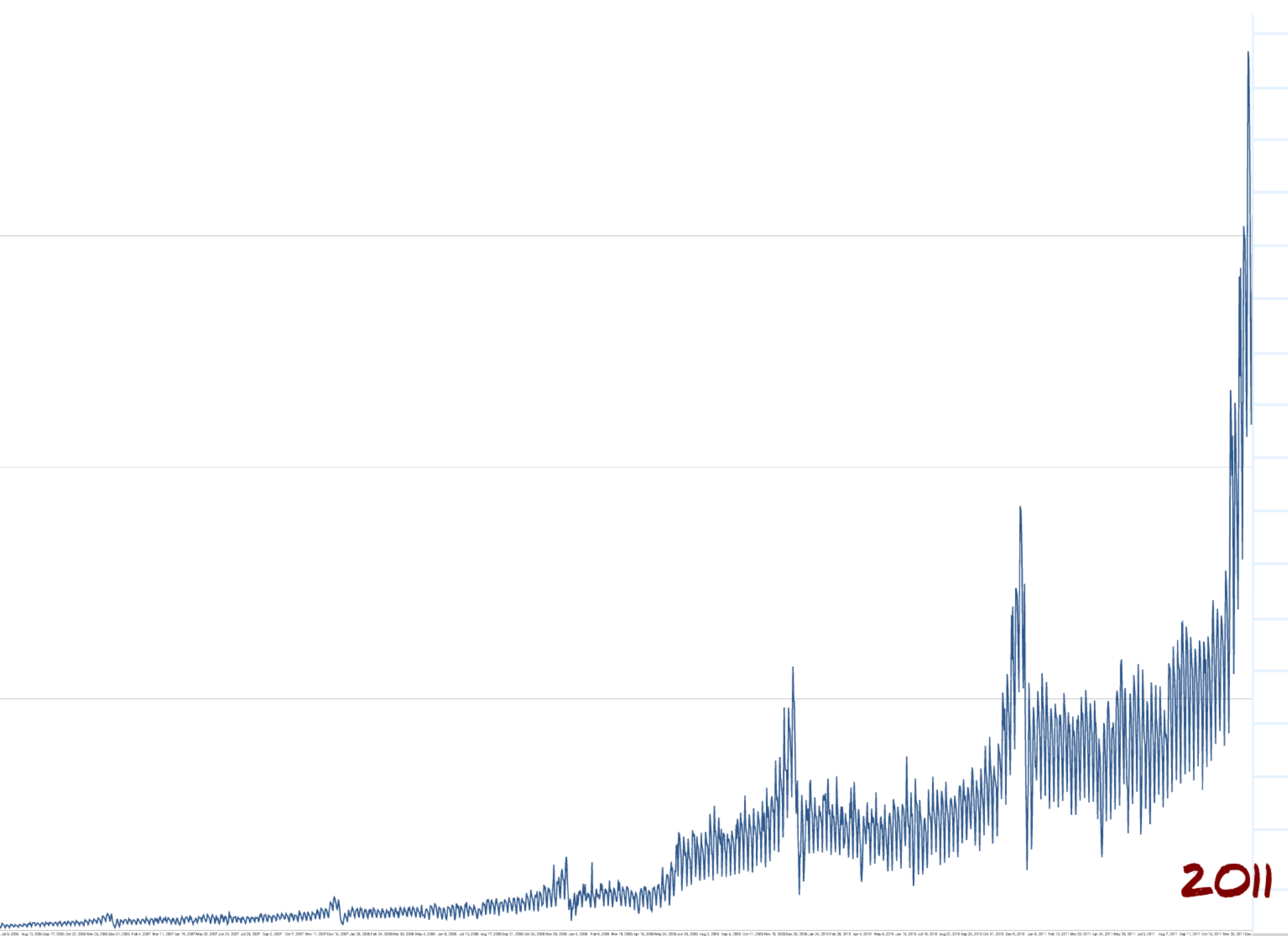




Jun 8, 2008 Jul 13, 2008 Aug 17, 2008 Sep 21, 2008 Oct 26, 2008 Nov 30, 2008 Jan 4, 2009 Feb 8, 2009 Mar 15, 2009 Apr 19, 2009 May 24, 2009 Jun 28, 2009 Aug 2, 2009 Sep 6, 2009 Oct 11, 2009 Nov 15, 2009 Dec 20, 2009 Jan 2, 2010







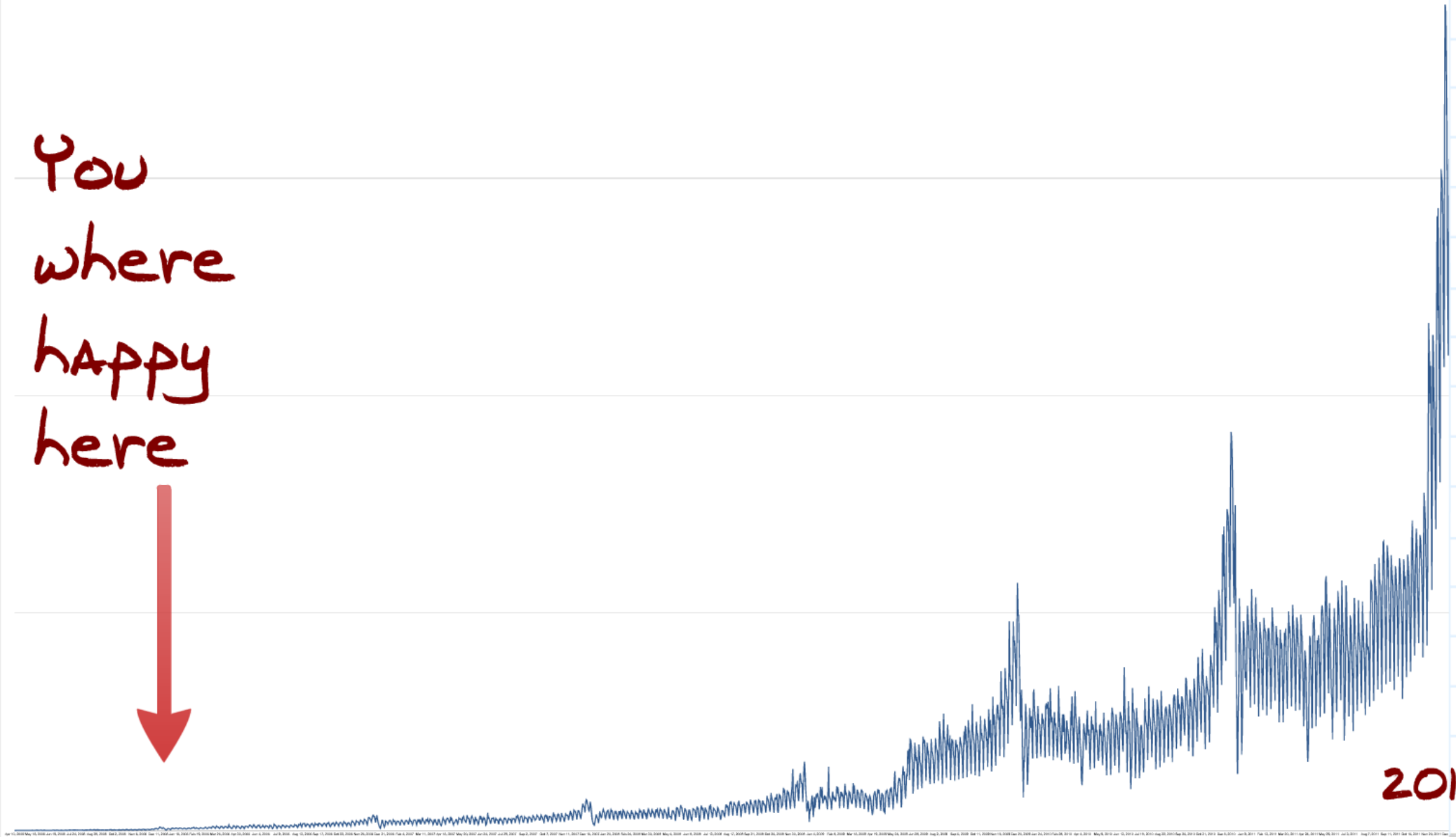
2011

This is what success looks like!

But this is pain

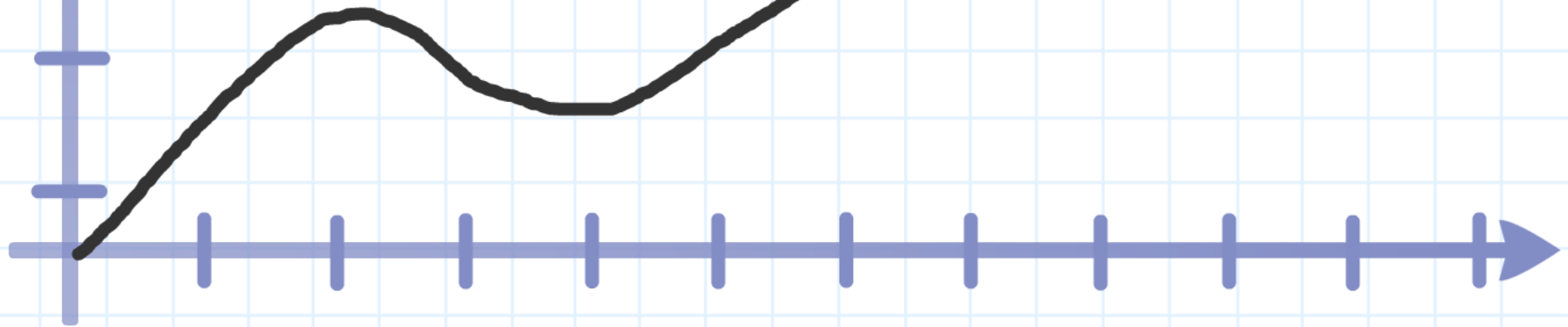


You  
where  
happy  
here



2011

If you want to  
build a scalable  
(web)service  
you want to do it in  
a language which  
helps you scale.



# Erlang

Fault Tolerant, Maintainable  
& Scalable

A scenic background image of a sunset over a body of water, framed by silhouettes of trees. The sun is a bright, glowing orb on the horizon, casting a warm orange and yellow light across the sky and reflecting on the water. A prominent red lens flare is visible on the right side of the image. The word 'Erlang' is written in a large, stylized, yellow font across the upper left portion of the image.

# Erlang

## **Fault Tolerant**

The right concurrency model.  
Error & exception handling done right  
Good libraries for the hard stuff





# Erlang

## Maintainable

Dynamically typed.

Symbolic and transparent data structures

An interactive shell



# Erlang

## Scalable

The right concurrency model.  
Good libraries for the hard stuff  
Weird but efficient strings for I/O





We're building the future with Erlang.  
Want to make history with us?

You Tube

# The right concurrency model

Lightweight processes  
Message passing  
Share nothing semantics  
Don't stop the world GC  
Monitors & Signals

• Processes as just memory  
• Promiscuous and extensible  
• No external threads  
• No external state



Error & exception handling done right



Dynamically & Strongly  
Typed  
Symbolic and transparent data structures

Enables:  
Hot code loading  
Moveable heaps & stacks  
Transparency of data  
Garbage Collection

Remember  
I don't have  
opinions  
this is all  
facts.

# Dynamically & Strongly Typed

Symbolic AND transparent data structures

Enables:

Hot code loading

Movable heaps & stacks

Transparency of data

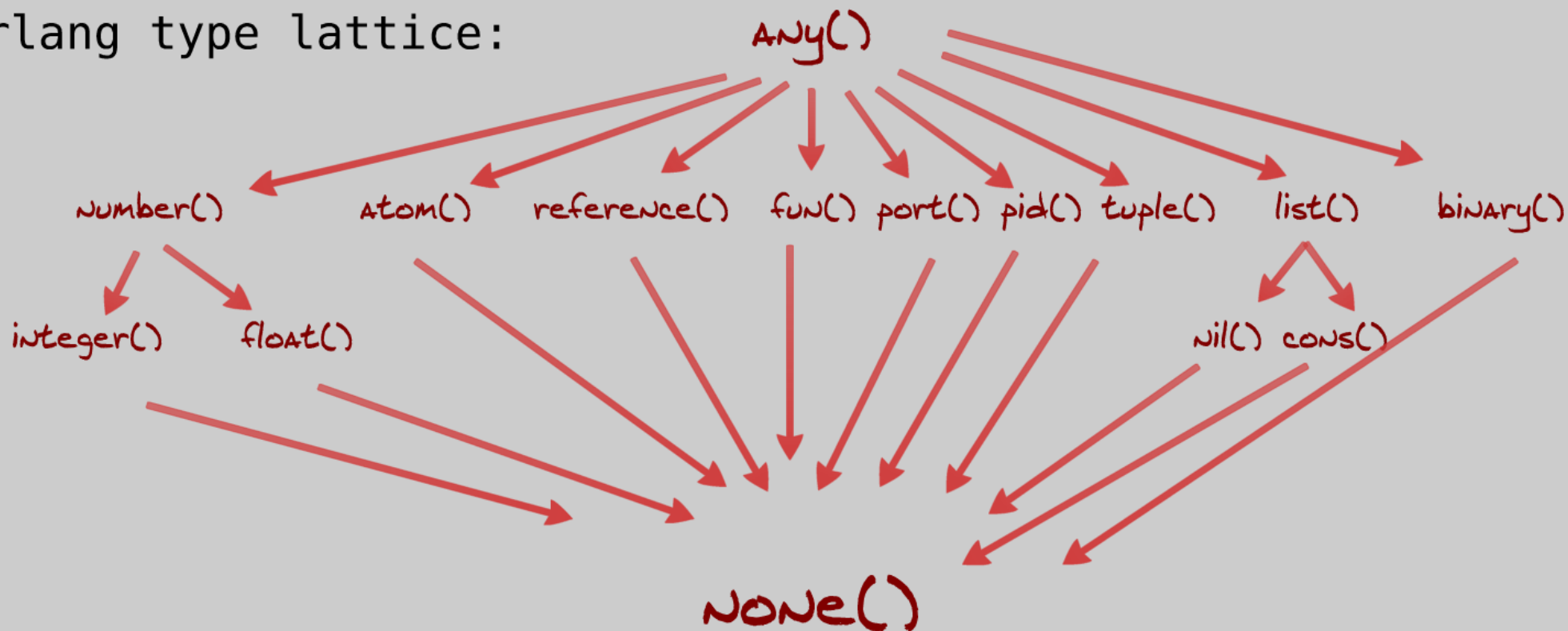
Garbage Collection



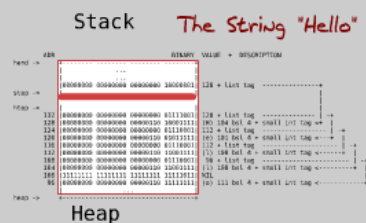
All values are tagged, some are boxed.



Erlang type lattice:



all values are tagged, some are boxed.



# Stack

The String "Hello"

|         | ADR |  | BINARY                              | VALUE                         | +       | DESCRIPTION |
|---------|-----|--|-------------------------------------|-------------------------------|---------|-------------|
| hend -> |     |  | +                                   |                               |         |             |
|         |     |  | ...                                 |                               |         |             |
|         |     |  | ...                                 |                               |         |             |
|         |     |  | 00000000 00000000 00000000 10000001 | 128 + list tag                | -----+  |             |
| stop -> |     |  |                                     |                               |         |             |
| htop -> |     |  |                                     |                               |         |             |
| 132     |     |  | 00000000 00000000 00000000 01111001 | 120 + list tag                | -----   | ++          |
| 128     |     |  | 00000000 00000000 00000110 10001111 | (H) 104 bsl 4 + small int tag | <+      |             |
| 124     |     |  | 00000000 00000000 00000000 01110001 | 112 + list tag                | -----   | ++          |
| 120     |     |  | 00000000 00000000 00000110 01011111 | (e) 101 bsl 4 + small int tag | <---+   |             |
| 116     |     |  | 00000000 00000000 00000000 01110001 | 112 + list tag                | -----   | ++          |
| 112     |     |  | 00000000 00000000 00000110 11001111 | (l) 108 bsl 4 + small int tag | <-----+ |             |
| 108     |     |  | 00000000 00000000 00000000 01110001 | 96 + list tag                 | -----   | ++          |
| 104     |     |  | 00000000 00000000 00000110 11001111 | (l) 108 bsl 4 + small int tag | <-----+ |             |
| 100     |     |  | 11111111 11111111 11111111 11111011 | NIL                           |         |             |
| 96      |     |  | 00000000 00000000 00000110 11111111 | (o) 111 bsl 4 + small int tag | <-----+ |             |
|         |     |  | ...                                 |                               |         |             |
| heap -> |     |  | +                                   |                               |         |             |

# Heap

# The right concurrency model

Lightweight processes  
Message passing  
Share nothing semantics  
Don't stop the world GC  
Monitors & Signals

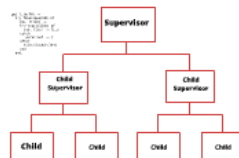
A Process is just memory



• Preemptive multitasking  
implemented through  
reduction counting



Error & exception handling done right



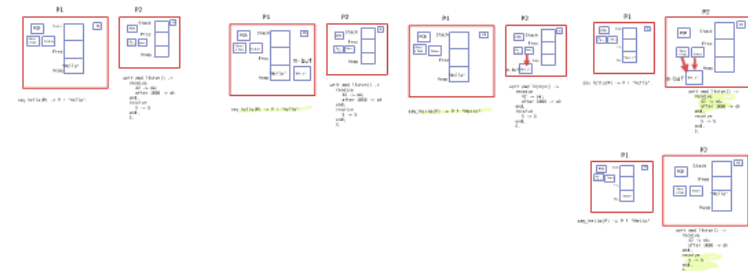
# The right concurrency model

Lightweight processes  
Message passing  
Share nothing semantics  
Don't stop the world GC  
Monitors & Signals

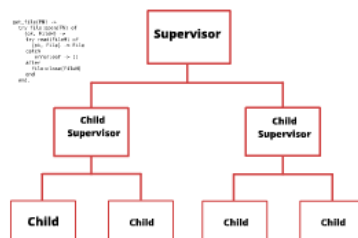
A Process is just memory



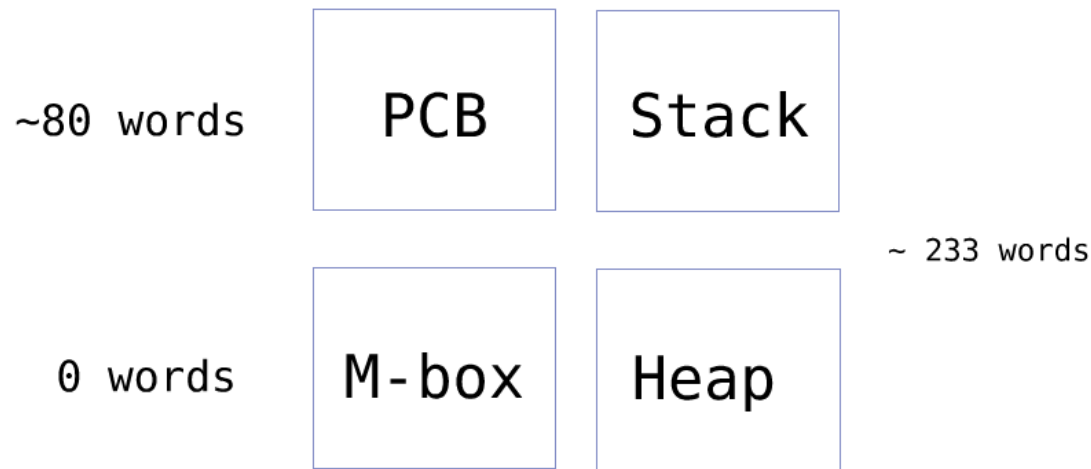
+ Preemptive multitasking  
implemented through  
reduction counting



Error & exception handling done right



# A Process is just memory



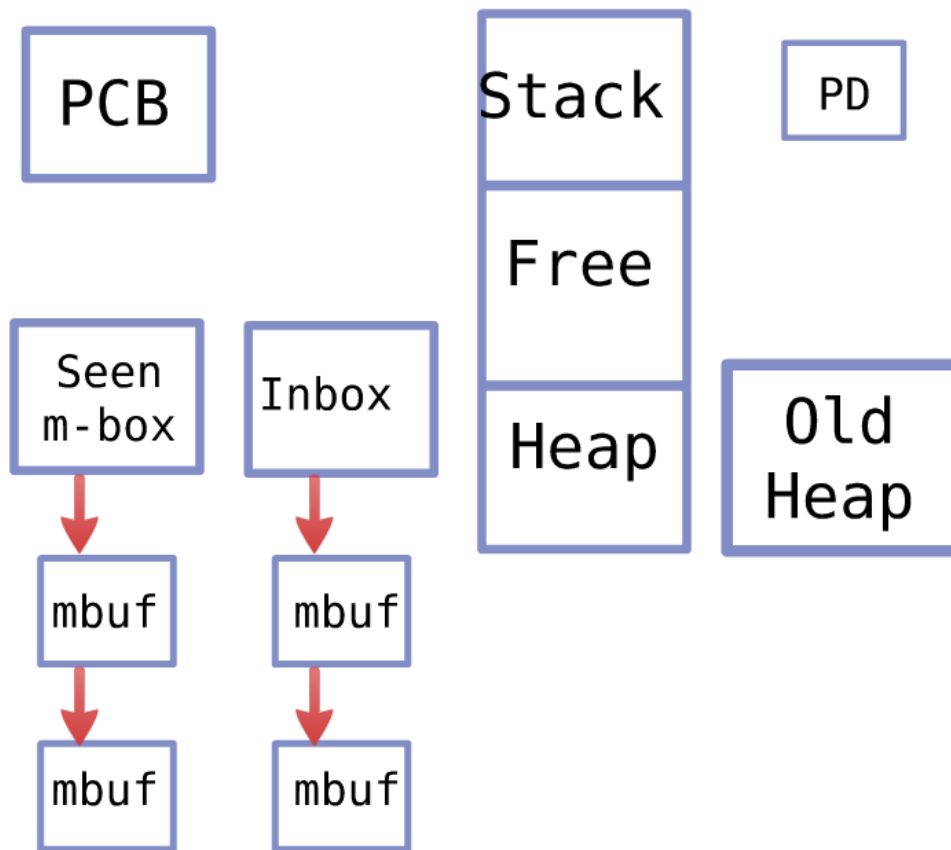
In reality it is a bit more complicated



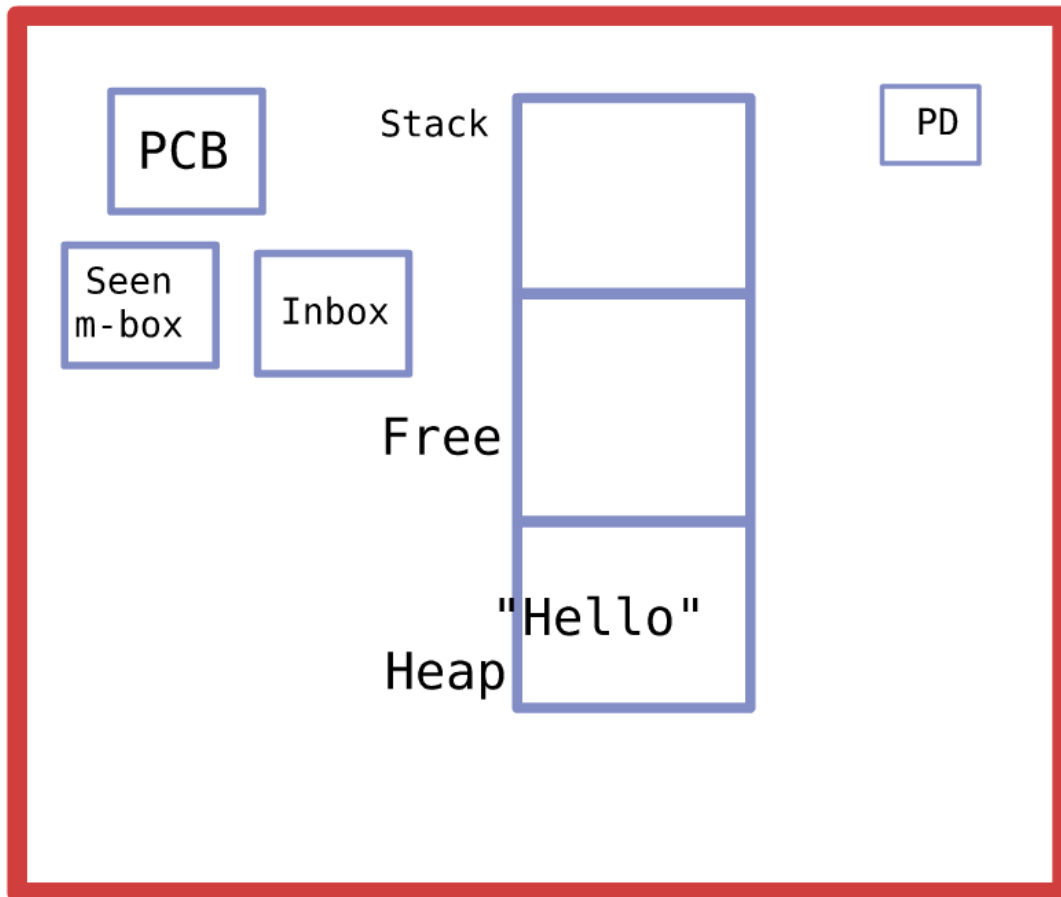
+ Preemptive multitasking  
implemented through  
reduction counting



In reality it is a bit more complicated

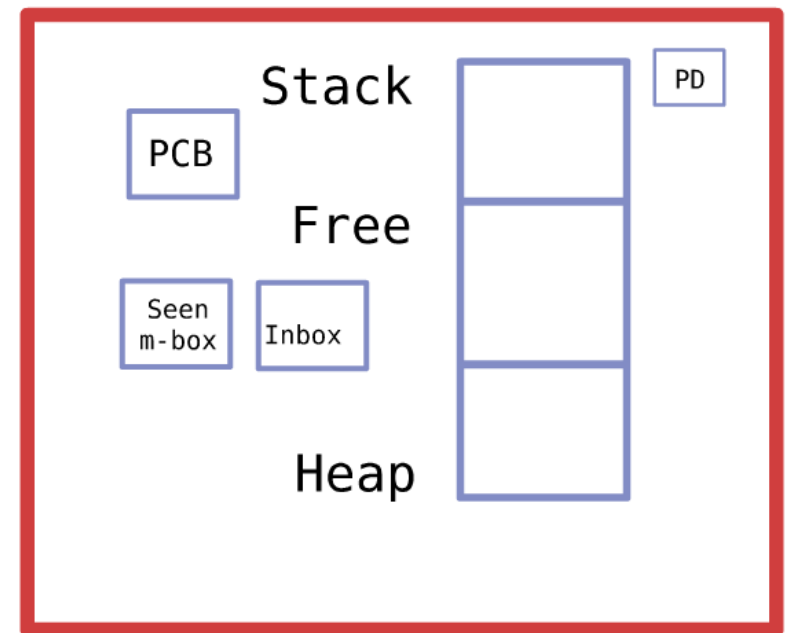


# P1



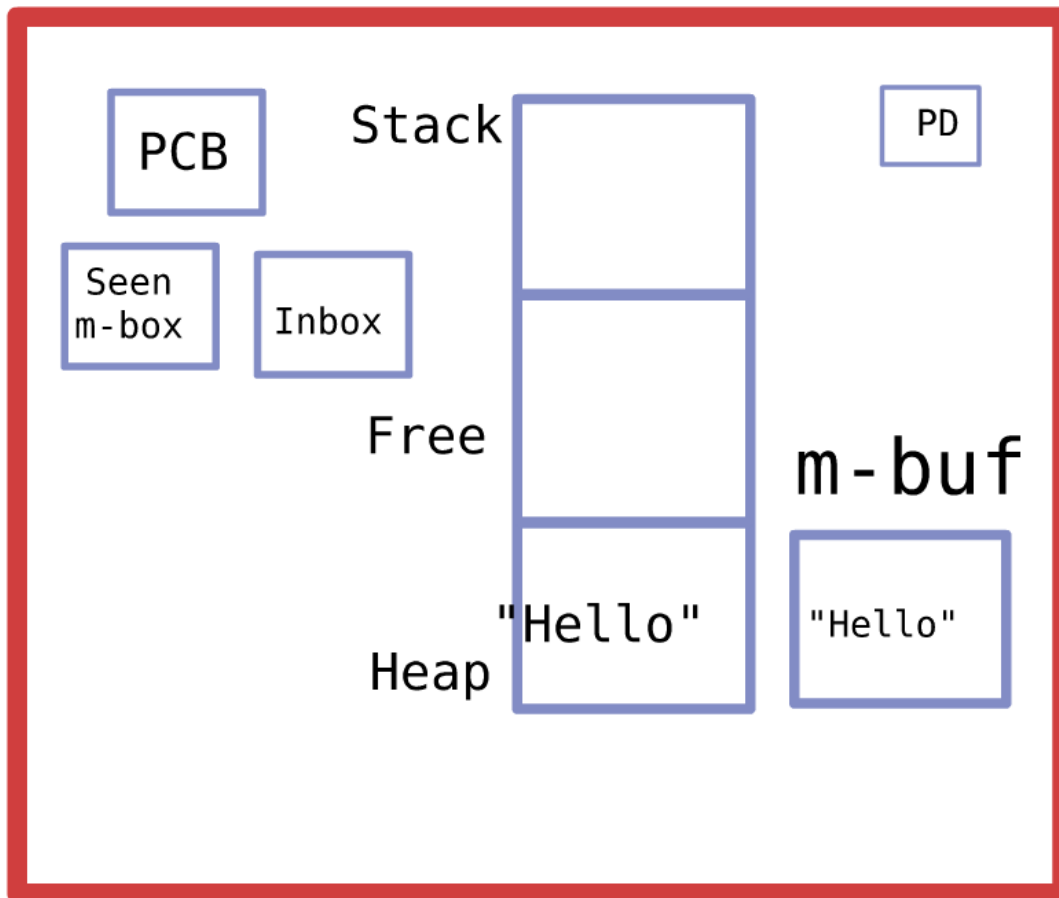
say\_hello(P) -> P ! "Hello".

# P2



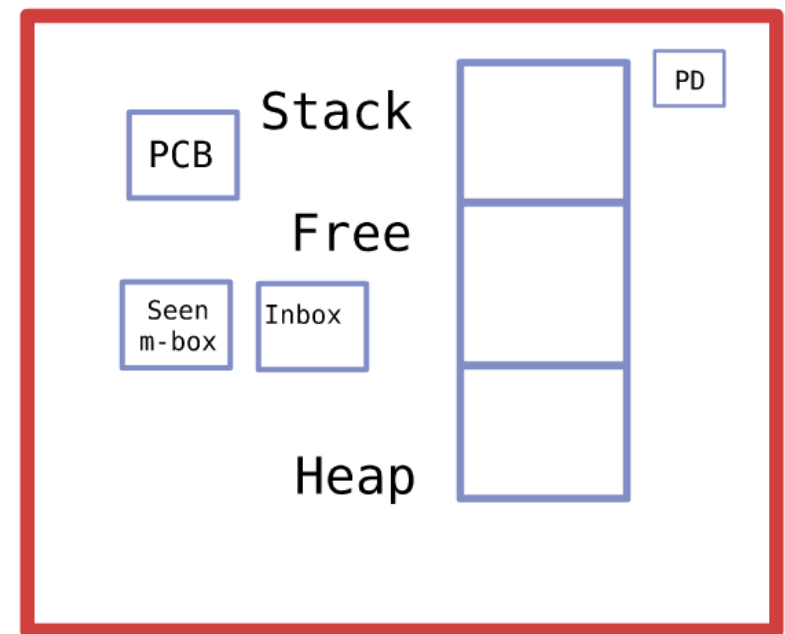
```
wait_and_listen() ->
  receive
    42 -> ok;
    after 1000 -> ok
end,
receive
  S -> S
end,
S.
```

# P1



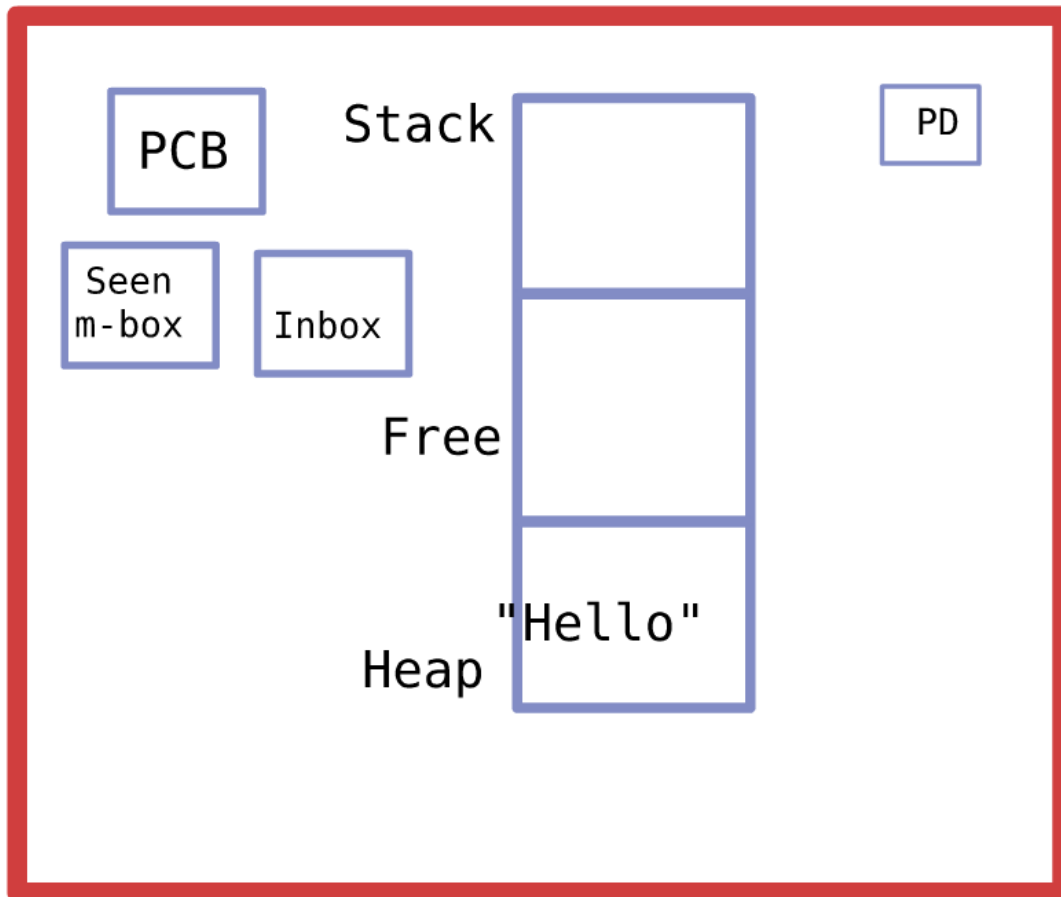
`say_hello(P) -> P ! "Hello".`

# P2

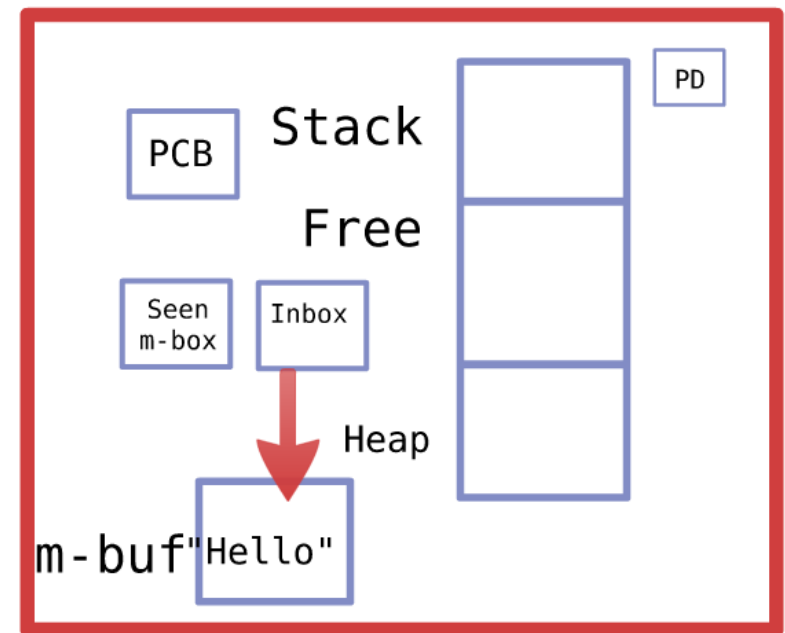


```
wait_and_listen() ->
  receive
    42 -> ok;
    after 1000 -> ok
end,
receive
  S -> S
end,
S.
```

# P1



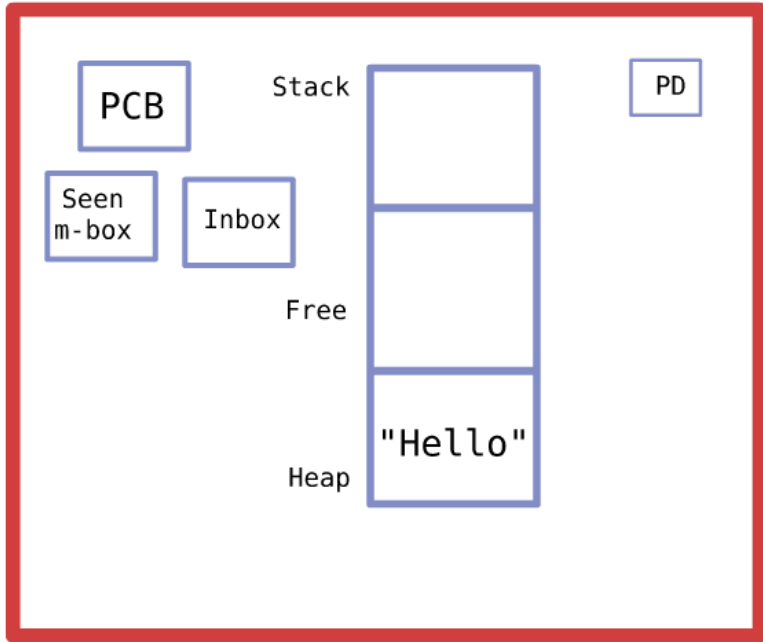
# P2



```
wait_and_listen() ->
  receive
    42 -> ok;
    after 1000 -> ok
end,
receive
  S -> S
end,
S.
```

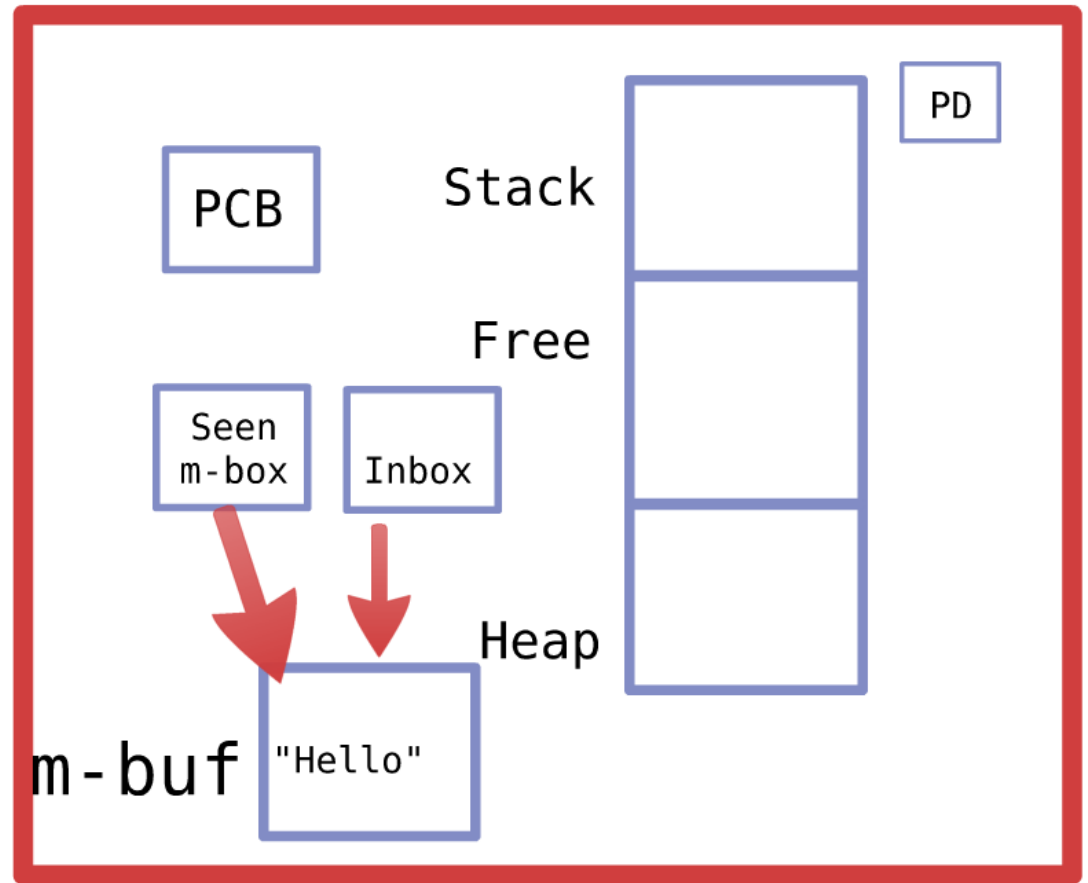
say\_hello(P) -> P ! "Hello".

# P1



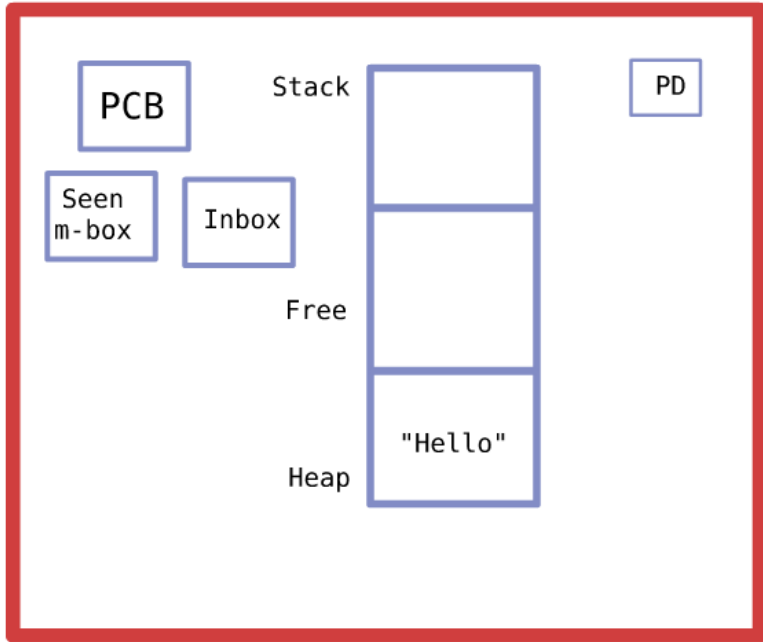
y\_hello(P) -> P ! "Hello".

# P2



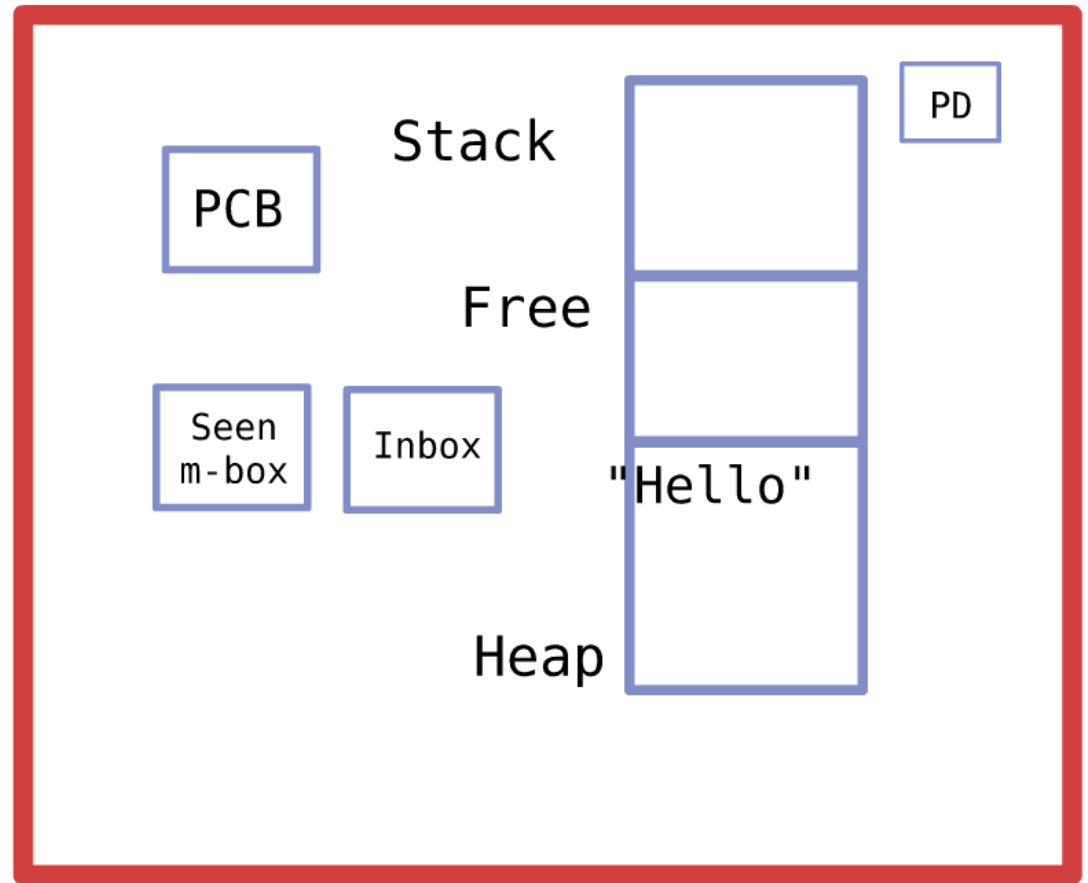
```
wait_and_listen() ->
  receive
    42 -> ok;
    after 1000 -> ok
  end,
  receive
    S -> S
  end,
  S.
```

# P1



\_hello(P) -> P ! "Hello".

# P2



```
wait_and_listen() ->
  receive
    42 -> ok;
    after 1000 -> ok
  end,
  receive
    S -> S
  end,
  S.
```

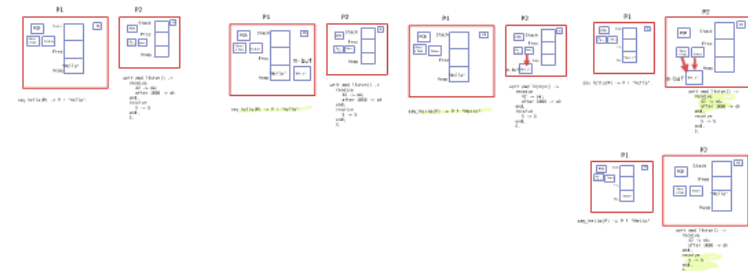
# The right concurrency model

Lightweight processes  
Message passing  
Share nothing semantics  
Don't stop the world GC  
Monitors & Signals

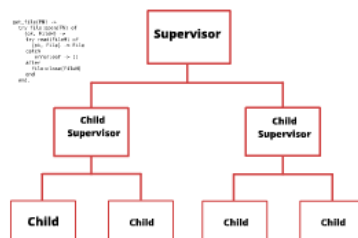
A Process is just memory



+ Preemptive multitasking  
implemented through  
reduction counting

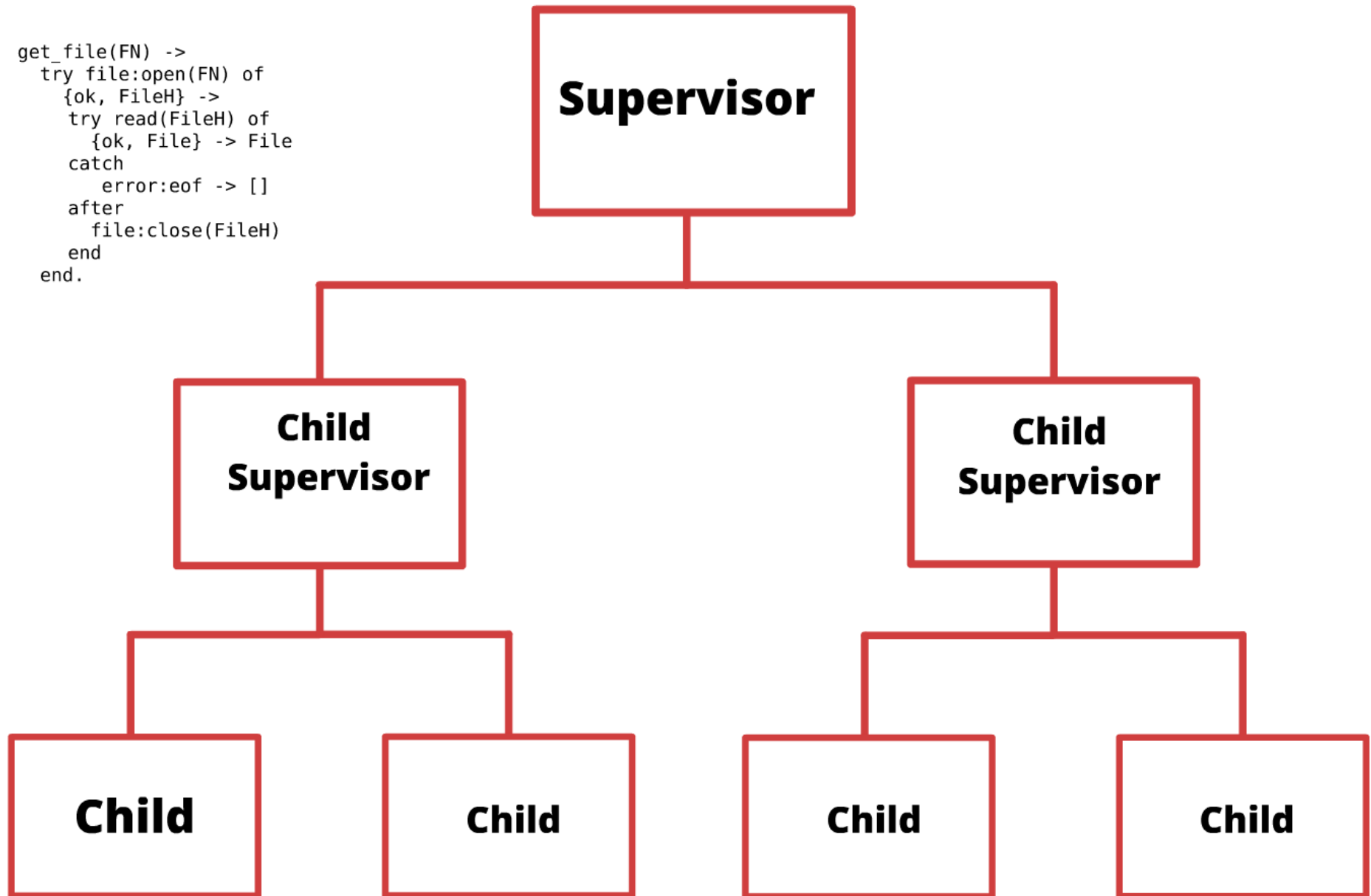


Error & exception handling done right



# or & exception handling done right

```
get_file(FN) ->  
  try file:open(FN) of  
    {ok, FileH} ->  
      try read(FileH) of  
        {ok, File} -> File  
      catch  
        error:eof -> []  
      after  
        file:close(FileH)  
      end  
  end  
end.
```





```
get_file(FN) ->
  try file:open(FN) of
    {ok, FileH} ->
      try read(FileH) of
        {ok, File} -> File
      catch
        error:eof -> []
      after
        file:close(FileH)
      end
    end
  end.
```



# An interactive shell

Fantastic for operation and maintenance:

```
1> lists:foldl(  
    fun ({_,V}, Acc) -> V + Acc end,  
    0,  
    [process_info(P, heap_size)  
      || P <- processes()]).  
46813  
2>
```

Fantastic for prototyping and testing during development

Drawback: Erlang developers don't save their test driven development tests.

Good libraries for the hard stuff

OTP

gen\_server

supervisor

gen\_fsm

# Weird but efficient strings for I/O

Strings are just list of integers.

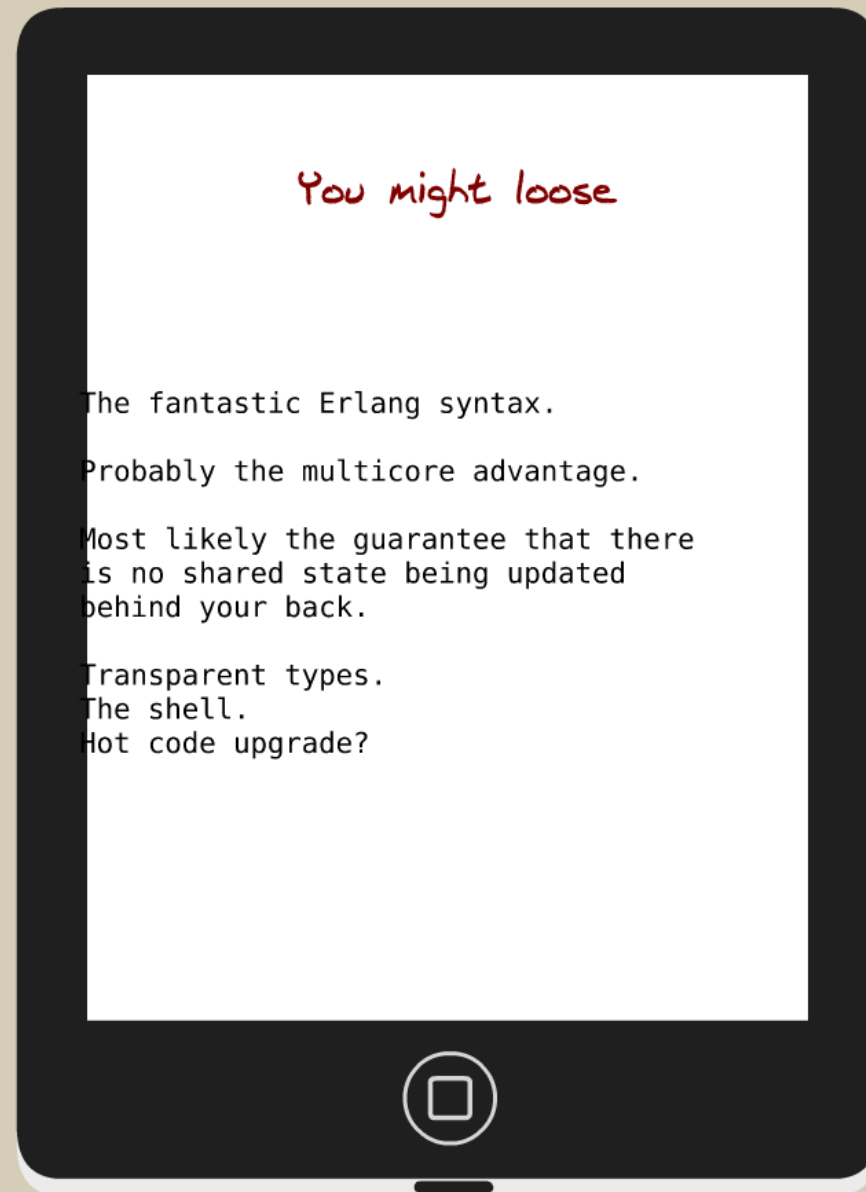
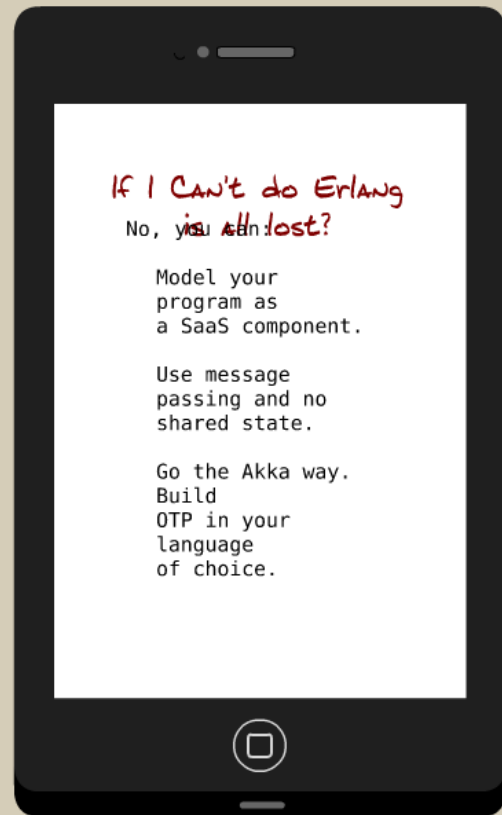
Can be iso8859 characters or unicode codepoints.

I/O lists: A nested list of  
binaries and strings.

String concatenation takes  
constant time:

`concat(S1,S2) -> [S1,S2].`

# I Can't do Erlang



If I Can't do Erlang  
is all lost?

No, you can:

Model your  
program as  
a SaaS component.

Use message  
passing and no  
shared state.

Go the Akka way.  
Build  
OTP in your  
language  
of choice.

The fanta

Probably

Most like  
is no sha  
behind yo

Transpare  
The shell  
Hot code

# do ng

What do Erlang  
can lose?

your  
m as  
component.

ssage  
g and no  
state.

Akka way.

your  
ge  
ice.

You might loose

The fantastic Erlang syntax.

Probably the multicore advantage.

Most likely the guarantee that there  
is no shared state being updated  
behind your back.

Transparent types.

The shell.

Hot code upgrade?

# A language makes you think

The most important aspect of Erlang has nothing to do with all the clever solutions in the implementation.

The most important aspect is that makes you think of a problem in a new way.


In a concurrent way.





YouTube





Conclusion:  
The future  
looks bright!

Questions?

**PS.**  
**I might have lied in the disclaimer  
in the beginning.**





# t great programmers.



adox: Phil Lennart SPJ John  
mmers makes it easier  
at programmers.

# God way to get great programmers.



Lennart SPJ

Nice paradox:

Phil

John

The lack of Erlang programmers makes it easier  
for us to find great programmers.

There are many great C and Java programmers,  
I'm sure, but they are hidden by hordes of  
mediocre programmers.

Programmers who know a functional programming  
language are often passionate about  
programming.

TM

Passionate programmers makes Great Programmers