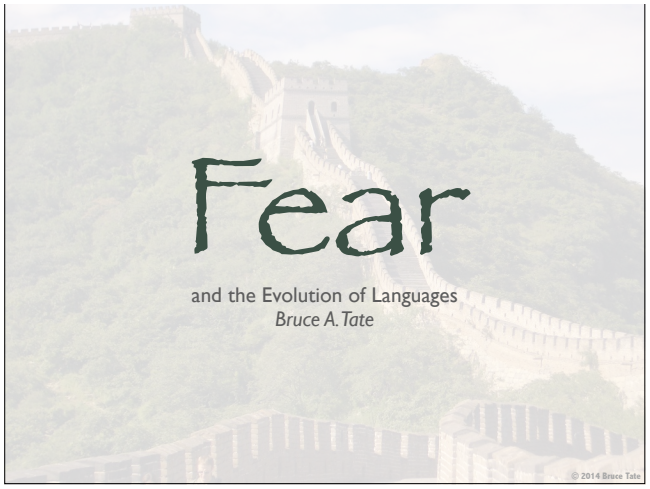
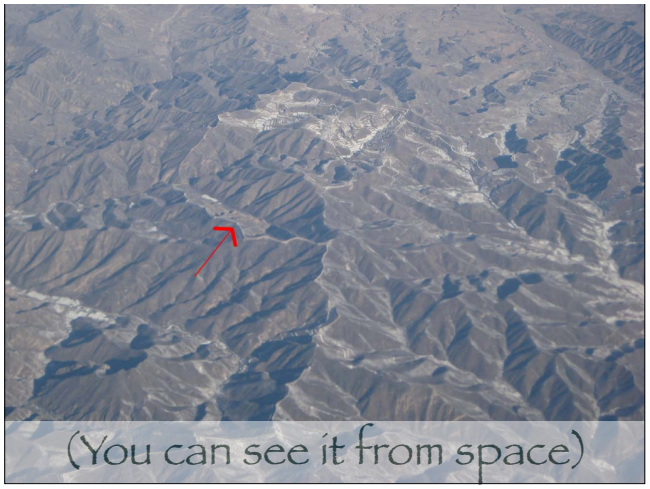


paddle along lake travis, 42 miles (about 70 Km), winds gusting to 25MPH (40KPH)

On a scale that's more grand, you can see the marks of fear all over the world. The most pronounced is the Great Wall of China.



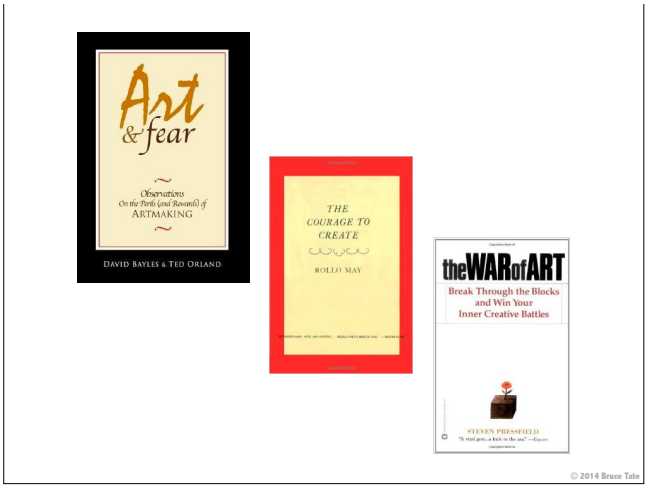
You can even see it from space. Fear visible from space. It's a great force on the planet. Shapes the way we behave by shaping the way that we think.

There's always been a great tension between fear and discovery.
Usually, for any great discovery, fear must be overcome.

Fear and language

creation

So I wanted to do a talk about fear's impact on language creation, but those things are in conflict.

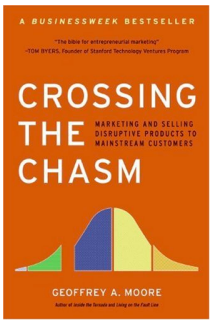


Many studies and many books.
Think “Writers block”

So that idea is out...

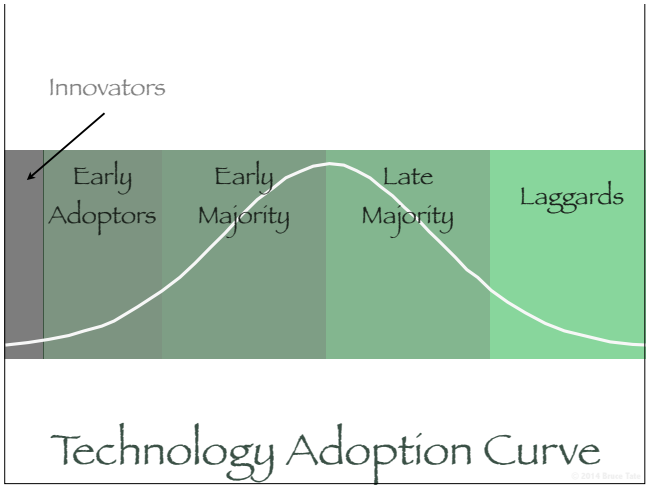
Fear and language adoption

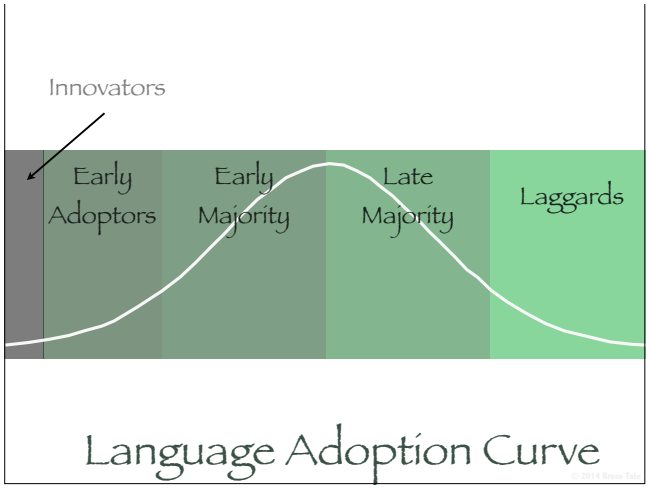
What I can talk about is fear and language adoption
long history... FUD ... appeal to fear...

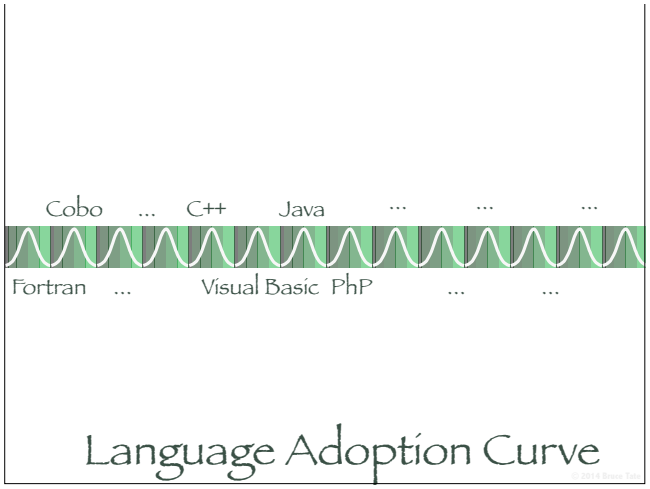


Geoffrey More 1991
groundbreaking technical marketing.

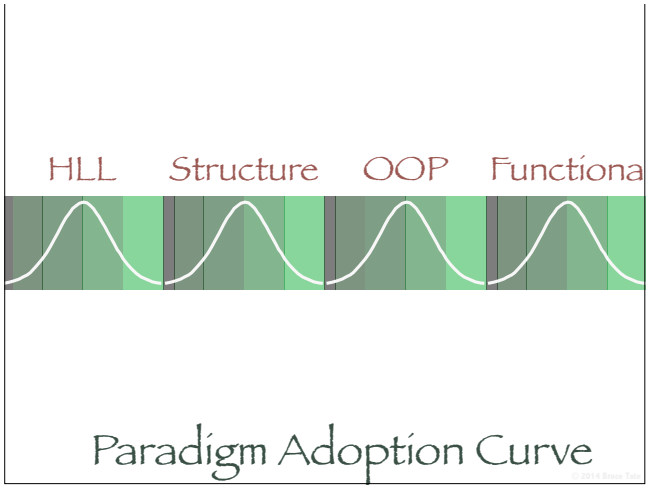

Technology Adoption Curve


Innovators
Early Adoptors
Early Majority
Late Majority
Laggards
Technology Adoption Curve

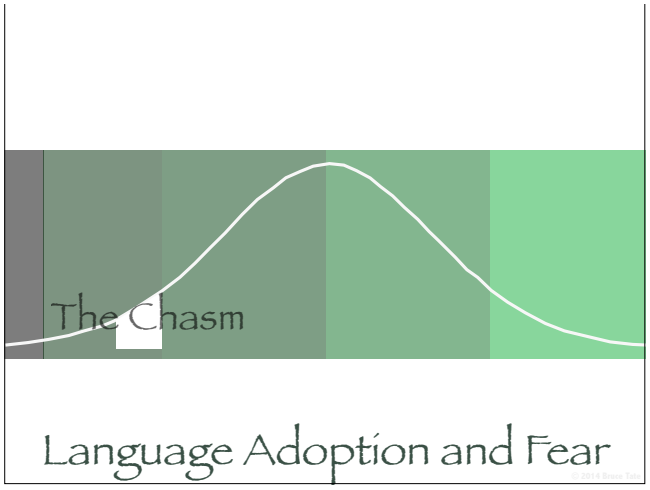




Curve comes in waves
The language curves are irregular... each language in each niche

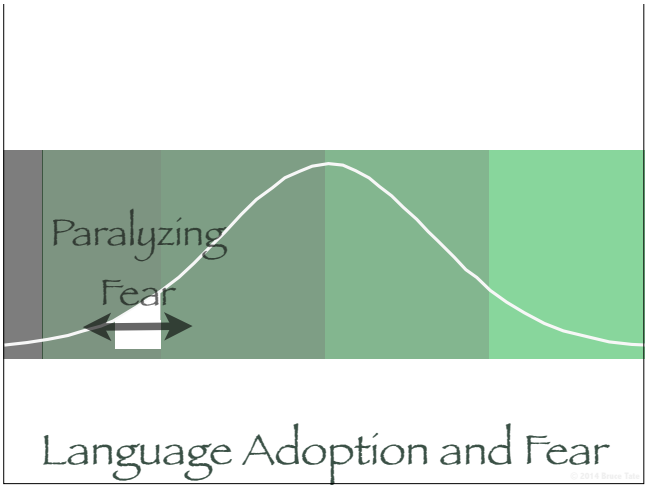


Paradigm adoption curves are more regular... every 20 years or so...
You can almost set your watch by it. Java and OOP was 1996...

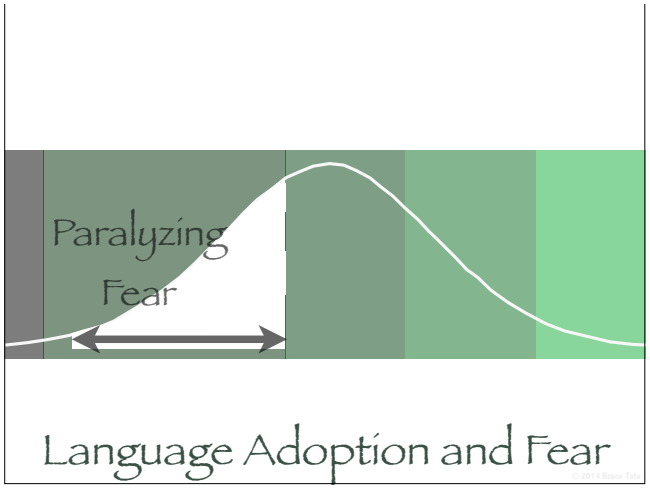


In Moore's book... early adopters are there
not enough momentum for early majority

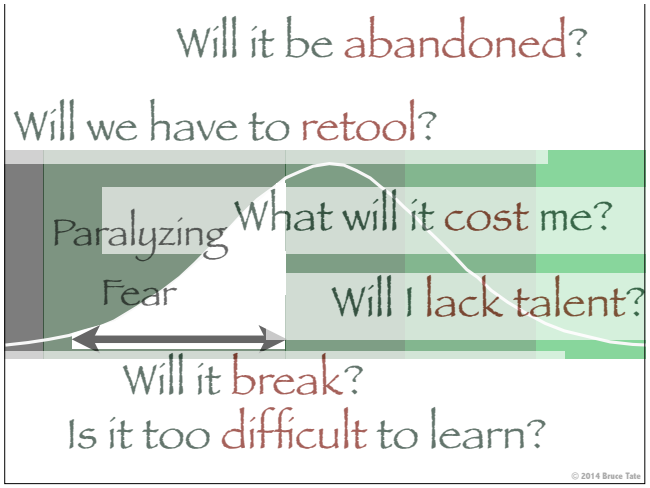
End Moore's theories... begin Bruce's theories
2 main fears associated with crossing the chasm



Paralyzing fear. This fear is why creating and selling new technology is so different.



Paralyzing fear makes the chasm wider and delays majority adoption

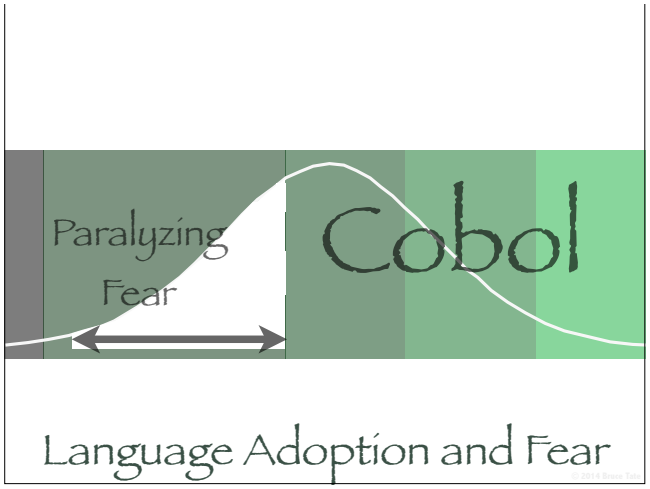


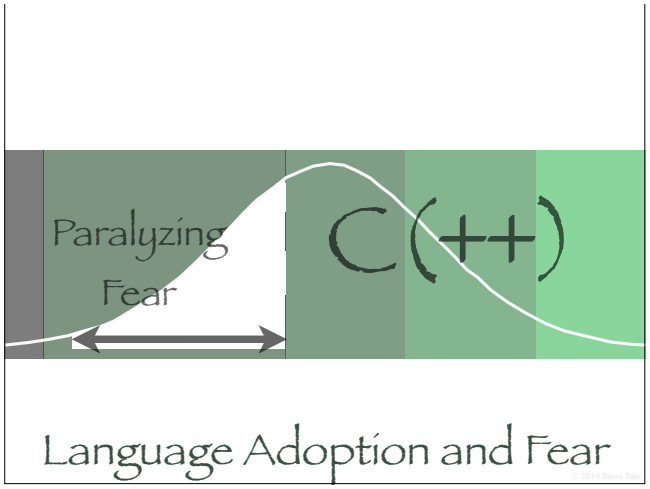
Some big words here... HIGH RISK meets HIGH COST
Stop the business for a couple of months and we'll see if this is going to work..

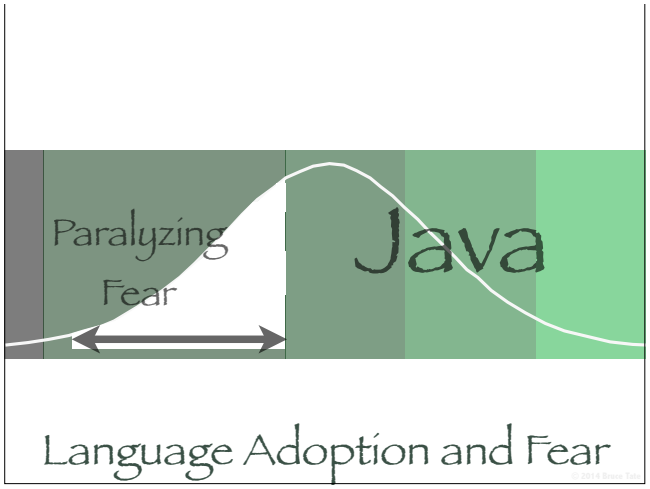
The chasm stops looking like this:

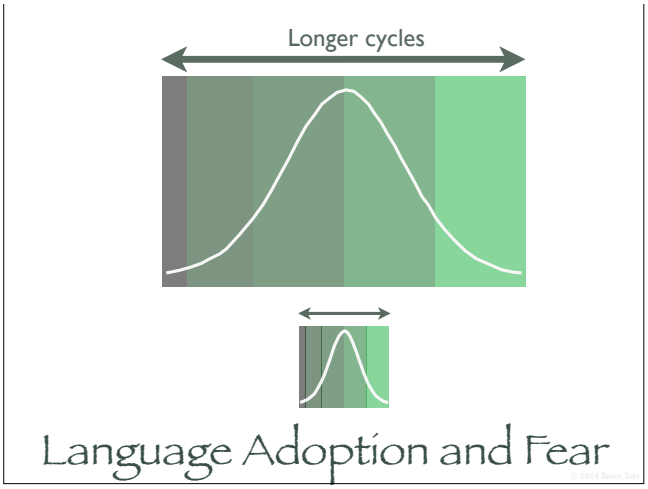
and starts looking more like this

This chasm is especially difficult for languages.
“Change the way you think” is scary, and risk.

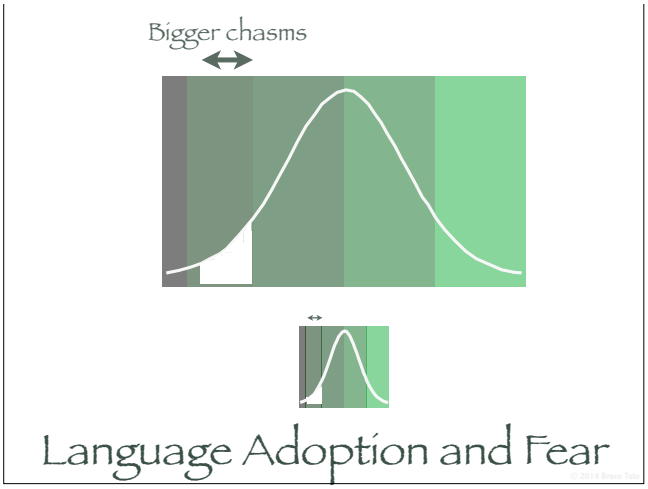


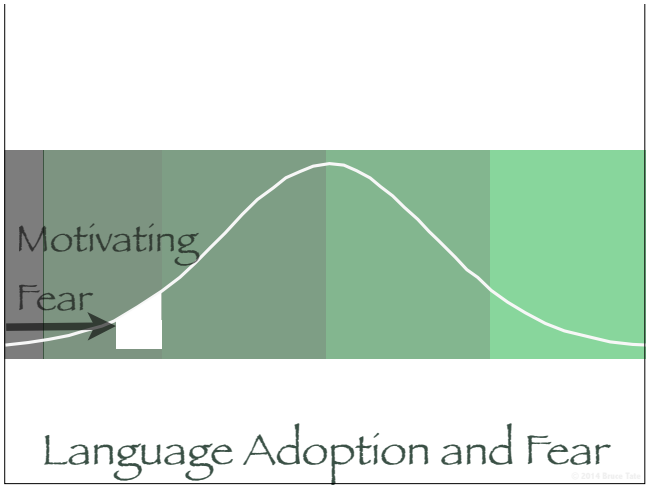

Paralyzing
Fear
C(++)
Language Adoption and Fear


Paralyzing
Fear
Java
Language Adoption and Fear

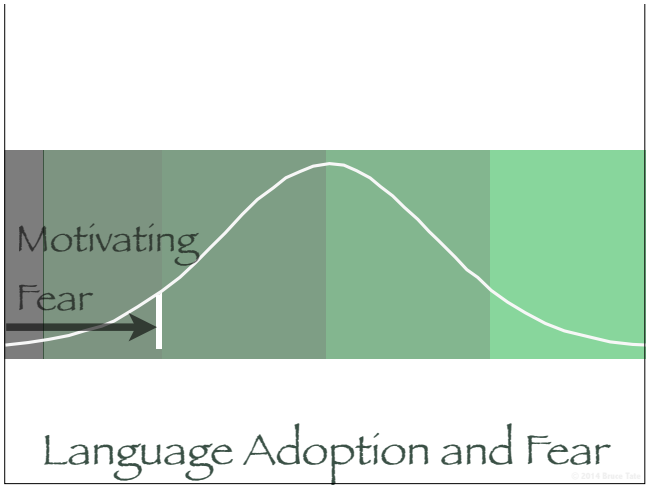


Paradigm adoption is even harder.

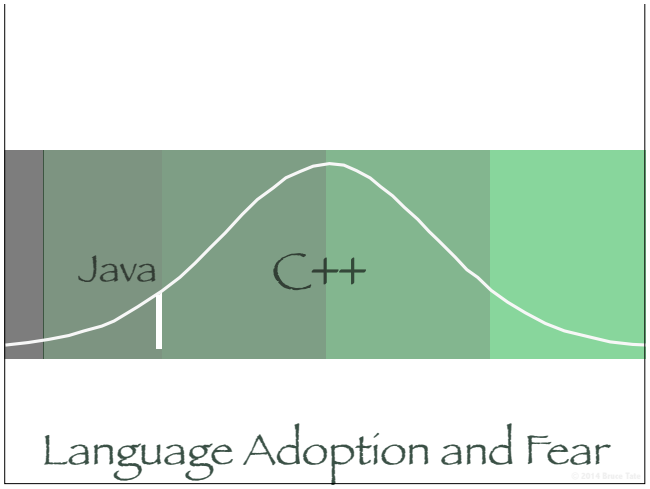




second major type of fear



Some external factor motivates the customers so strongly that it crosses the chasm



For example, C++ is entrenched firmly as the language of choice
(also, Visual Basic and a few others)

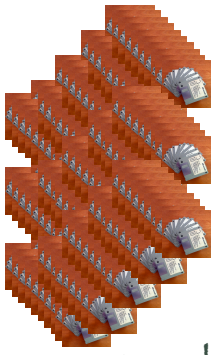
Usually, the paralyzing fear is too much to overcome
When the motivating fear gets big enough or the paralyzing fear shrinks,

We didn't know how to do client/server. Even when we could get the applications right
We couldn't get the management right. The big issue

(10 diskettes) X (9 registers per store)

Deployment Problem

(10 diskettes) X (9 registers per store) X (5 stores)



Deployment Problem

(10 diskettes) X
(9 registers per store) X
(5 stores) X
(3 services) X
(n fixpacks/year) ...

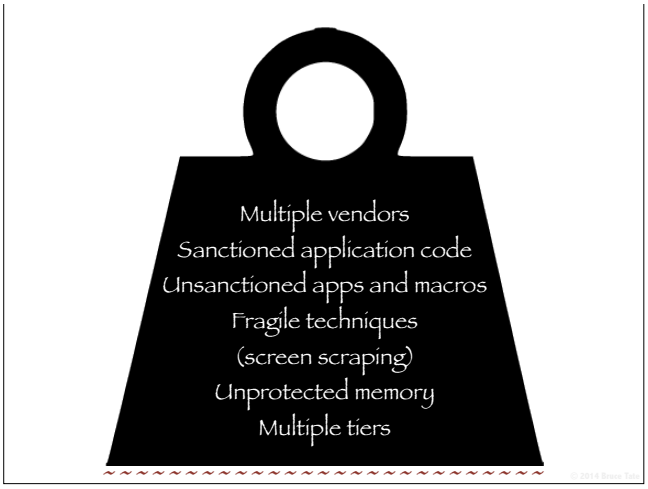
Deployment Problem

Now add...

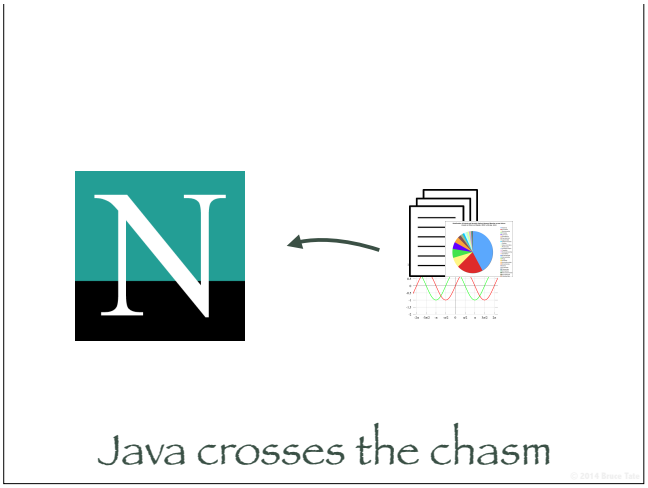
Deployment Problem

Just keep throwing variables at the hundreds or thousands of clients that users had to manage
With app development that could not keep up

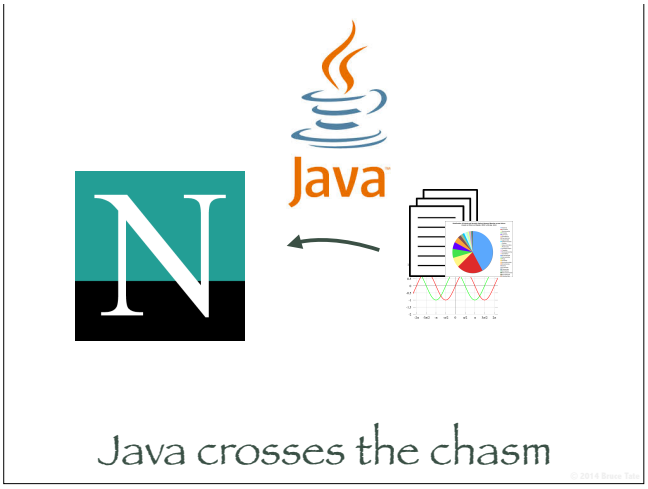
Even to the point of writing business applications in basic
Or scraping and advancing the screen when the Cobol apps could not be refit



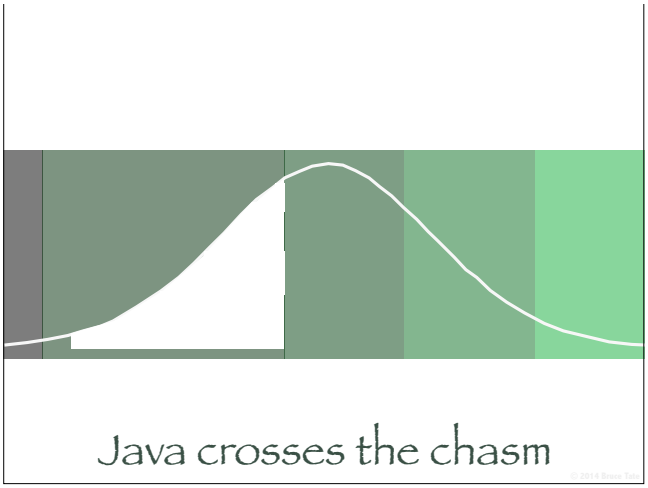
And often with large chunks in C++ (with no memory protection across applications)
The weight was crushing



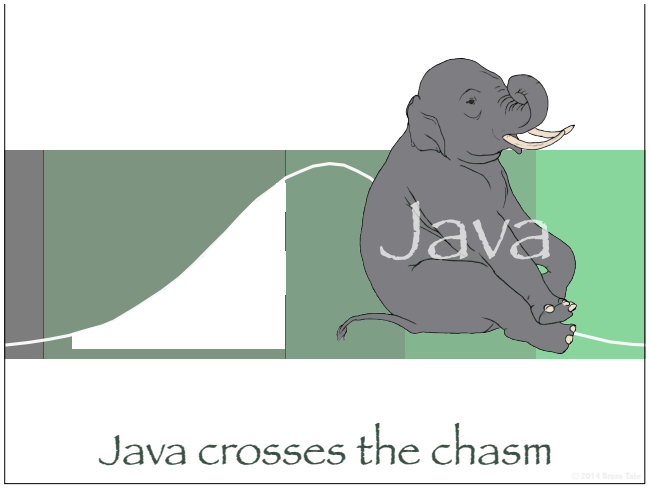
It's a very different thing to deploy a browser. Just a browser. And everything else can live in the browser.



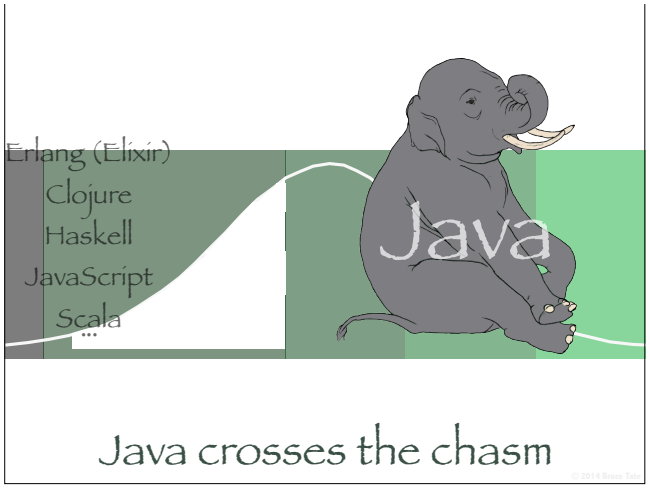
The promise was applets but servlets worked better.



So here we stand



With a two ton elephant sitting just on the other side of the chasm



and the language of your dreams sitting just on the other side.
What pressing fear can make that elephant get up and move?

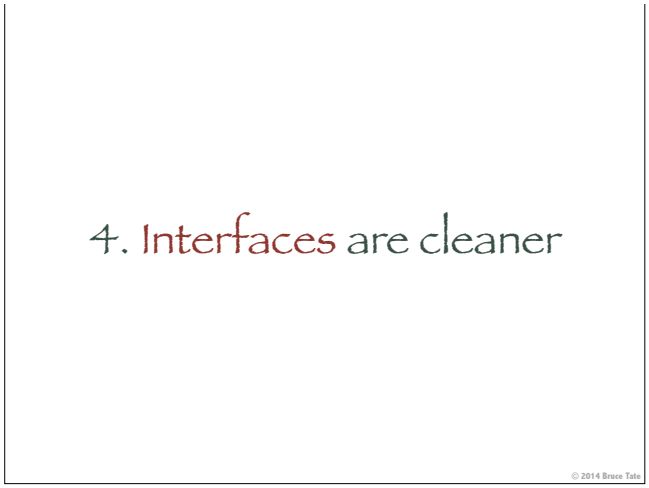
What's

What's happening to make the paralyzing fears less oppressive?

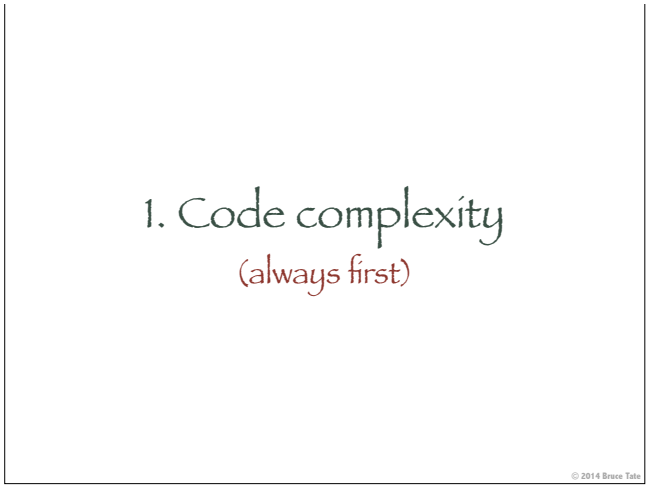
Example: Rails. Language developed in Japan, promoted and discovered by British expatriate in Dallas, gave rise to a framework invented in Denmark with a core team that spans most of the continents (not Africa or Antarctica)
Internet makes it easy to find answers and fix problems... unprecedented access high on the food chain

You have heard me say that new paradigms need bridge languages. C++ served that purpose for OOP.
Even Java has closures now. C++ was a bridge language to OOP, just as languages like Scala and even Ruby help us bridge to fp

You can already deploy Elixir using Heroku, and we're not even to version 1.0 yet. You don't have to invest in the software and infrastructure.



The Internet makes it easy for elements of an application to communicate. All kinds of good options abound to integrate the old to the new so that it's easier to take the journey with smaller steps. This makes all of the difference in the world. My company will start writing Elixir this year. We'll start with the back-end.



Complexity is always motivational --- things getting harder --- universal driver across time
today's app is distributed, secure, concurrent, integrated, fast, interactive, global, stable... list goes on
Java has crosscutting concerns and is running out of meaningful ways to manage them.

```
defmodule VidStore do
  use StateMachine

  state :available, [
    rent: [ to: :rented,
            calls: [ &VidStore.renting/1 ]]]

  state :rented, [
    return: [ to: :available,
              calls: [ &VidStore.returning/1 ]],
    lose: [ to: :lost,
            calls: [ &VidStore.losing/1 ]]]
  state :lost, []

  ...
end
```


In Elixir, I can use macros to provide code organization at compile time that is not available to me at run time.
I can effectively rewrite the AST to change the language.

```
defmodule VidStore do
  use StateMachine

  state :available, [
    rent: [ to: :rented,
            calls: [ &VidStore.renting/1 ]]]

  state :rented, [
    return: [ to: :available,
              calls: [ &VidStore.returning/1 ]],
    lose: [ to: :lost,
            calls: [ &VidStore.losing/1 ]]]
  state :lost, []

  ...
end
```

VidStore.rent video

VidStore.return video

video |> rent |> lose

You've seen the benefits at the API side, but the client of that API also gets benefits.
It's trivial to take a video through its states.

2. Multicore and Distribution

(the real Y2K)

This is the greatest challenge our generation of programmers will solve. Success or failure will define us.

```
defmodule Chatroom do
  use OtpDsl.GenServer, initial_state: HashDict.new()

  defcall enter(name), users do
    send_all(users, "#{name} has entered the room")
    reply(:ok, Dict.put(users, name, _from))
  end

  defcall leave(name), users do
    d = Dict.delete(users, name)
    send_all(users, "#{name} has left the room")
    reply(:ok, d)
  end

  defcall message(name, message) do
    send_all(users, message)
    reply(:ok, d)
  end

  defp send_all(users, message) do
    Enum.each(Dict.values(users), User.send_line(&1, message))
  end
end
```

replaces...


```
defmodule Chatroom2 do
  use GenServer.Behaviour
  def enter(name) do
    :gen_server.call(:chatroom, { :enter, name })
  end

  def leave(name) do
    :gen_server.call(:chatroom, { :leave, name })
  end

  def message(name, message) do
    :gen_server.call(:chatroom, { :message, name, message })
  end

  def init(_args) do
    { :ok, HashDict.new() }
  end

  def handle_call({ :enter, name }, from, users) do
    send_all(users, "#{name} has entered the room")
    { :reply, :ok, Dict.put(users, name, from) }
  end

  def handle_call({ :leave, name }, _from, users) do
    d = Dict.delete(users, name)
    send_all(users, "#{name} has left the room")
    { :reply, :ok, d }
  end

  def handle_call({ :message, name, message }, _from, users) do
    send_all(users, message)
    { :reply, :ok, users }
  end

  defp send_all(users, message) do
    Enum.each(Dict.to_list(users), fn { user, pid } ->
      User.send_line(user, message)
    end)
  end
end
```

replaces...

Too big to show

DSL from Dave Thomas:
https://github.com/pragdave/otp_dsl

Example from Peter Minten

Thinking in Elixir: Hiding Your Messages
http://pminten.github.io/blog/2013/09/14/thinking-in-elixir-hide-your-messages/

code example: Copyright © 2014 - Peter Minten

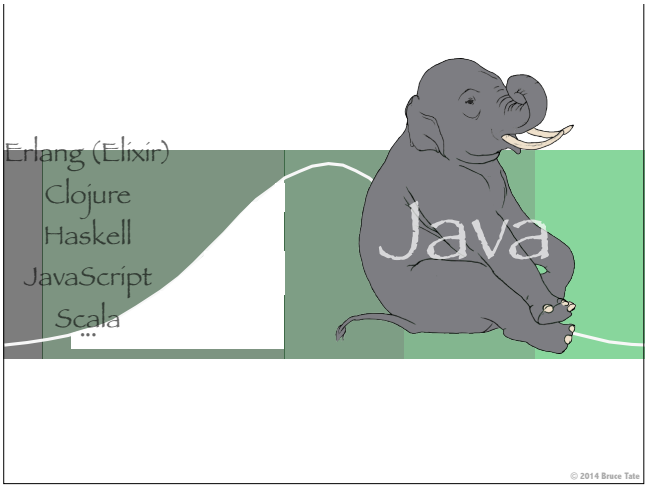
References

3. Browser complexity

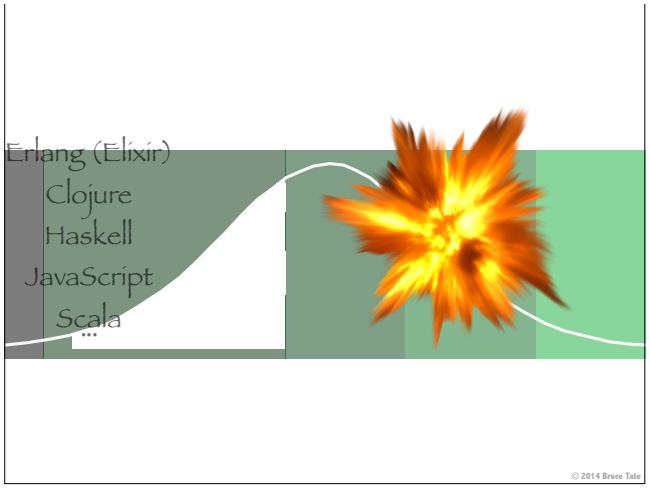
Is JavaScript the best we can do?

```
import Mouse
import Window

drawPaddle w h x =
  filled black (rect 80 10)          |>
  moveX (toFloat x - toFloat w / 2) |>
  moveY (-(toFloat h * 0.45))

display (w, h) x = collage w h
  [ drawPaddle w h x ]

main = lift2 display Window.dimensions Mouse.x
```

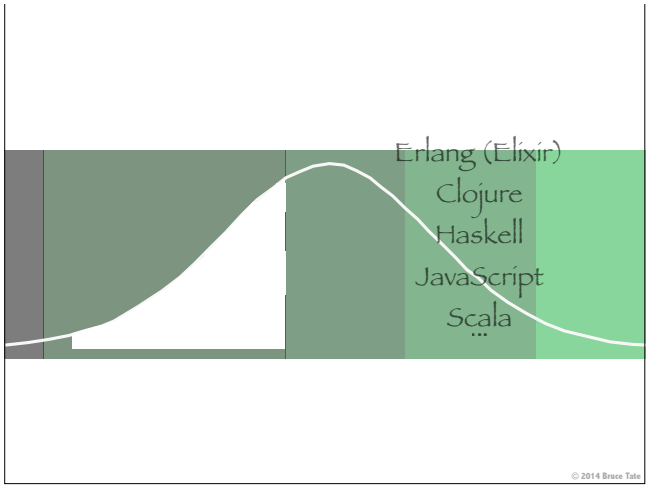

Benefits: Strong typing to catch those bugs our JS developers miss
End to Callback Hell.
Compiles to JavaScript



This is what it's going to take



to remove Java and OOP



and cross the chasm

References

DSL from Dave Thomas:
https://github.com/pragdave/otp_dsl

Example from Peter Minten

Thinking in Elixir: Hiding Your Messages
http://pminten.github.io/blog/2013/09/14/thinking-in-elixir-hide-your-messages/

code example: Copyright © 2014 - Peter Minten

Talks in PDF form at