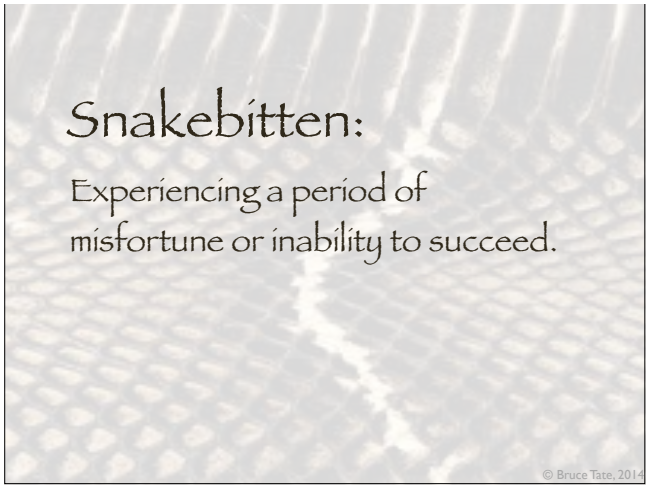
Snakebitten

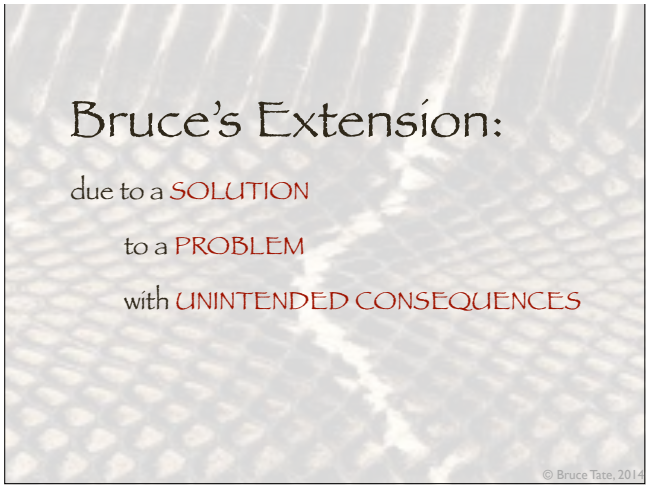
Danger and Misfortune in the Evolution of Languages

I believe our industry is in a bad place
I want to look at how we got here

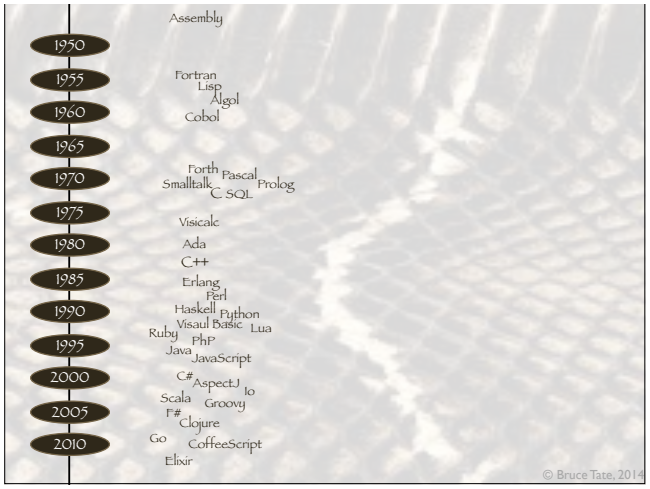
A biking trail near home
steep climb blind corner
there's a place where you can put your foot if you mess up
I didn't step in the snake... but it changed the way I ride



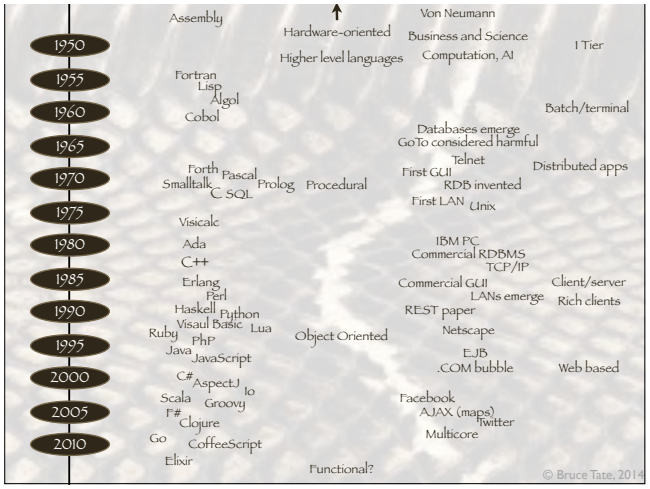
We have an expression in Texas Snakebitten
We say it a different way... SNAKEBIT
AINT NUTHIN GOOD EVER HAPPENED TO HIM... HE'S SNAKEBIT



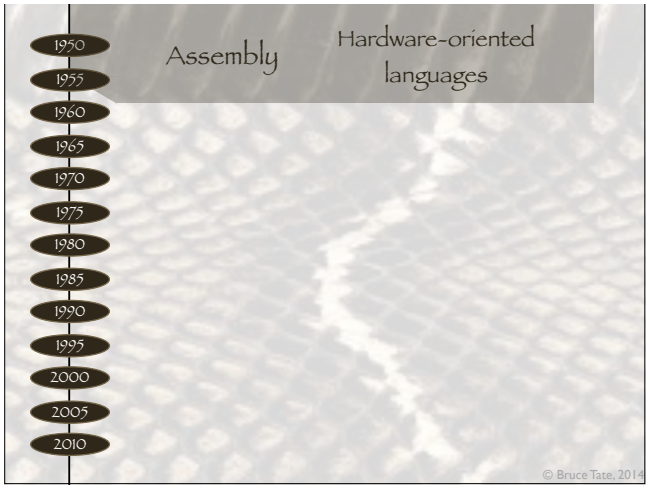
Bad things usually keep happening for a reason
We are snakebitten today... becoming increasingly clear.
How did we get there?



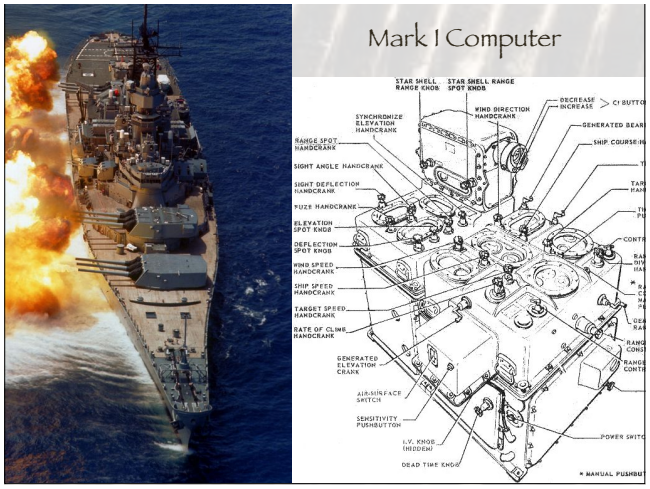
It's a long story.
Programming language timeline
inexact, incomplete



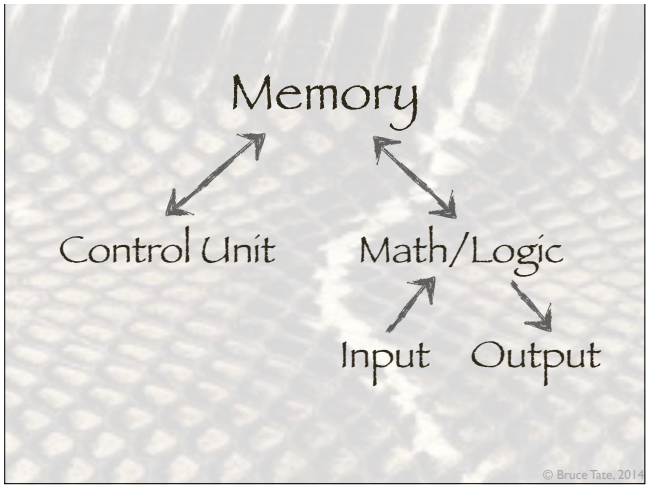
Every language on this page is a product of its history
Languages evolve for a reason CONTEXT MATTERS
The problems on the right gave rise to the languages and models on the left



Early languages were tied tightly to their machines... grandfather Von Neumann: Computer architecture (and subsequent family of languages)
NOY man

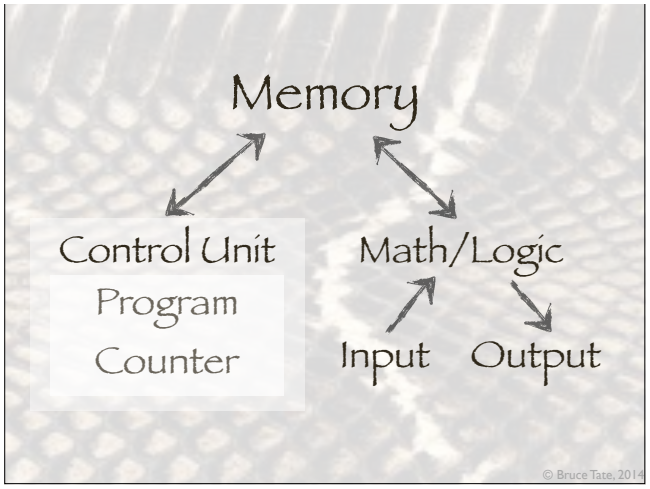


Not the only computer model, Mark I: analog, Electro-mechanical hand cranks: Wind direction, ship course, target direction
Note: output doesn't change with time!!!



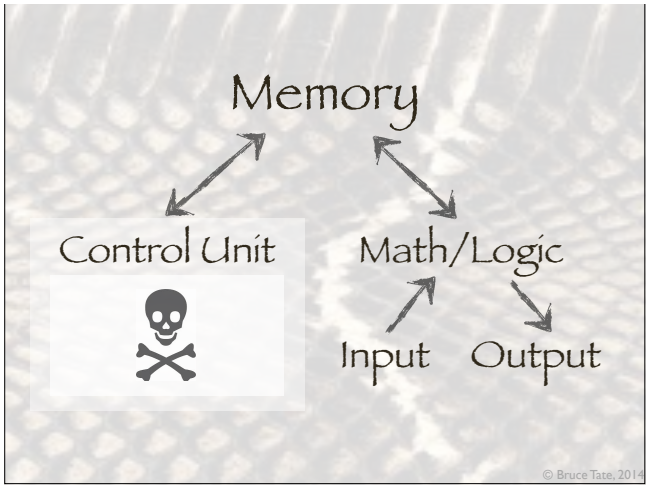
Von Neumann systems won

Burried in that control unit is a PROGRAM COUNTER

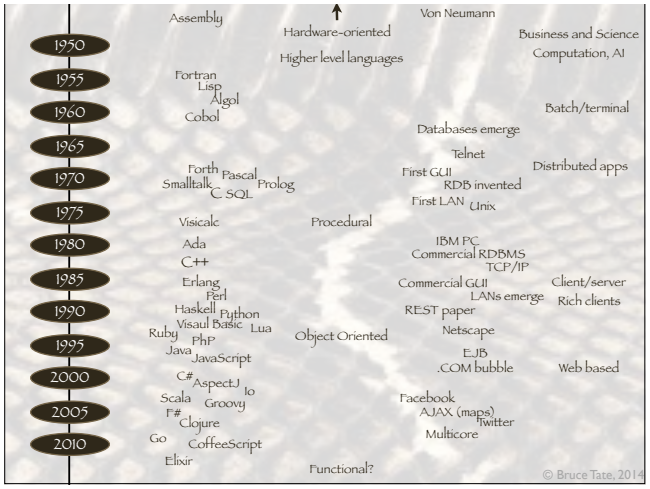


Von Neumann systems won

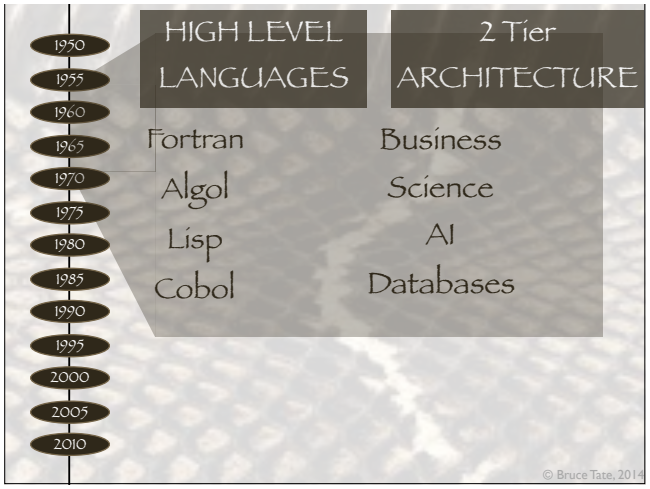
Burried in that control unit is a PROGRAM COUNTER



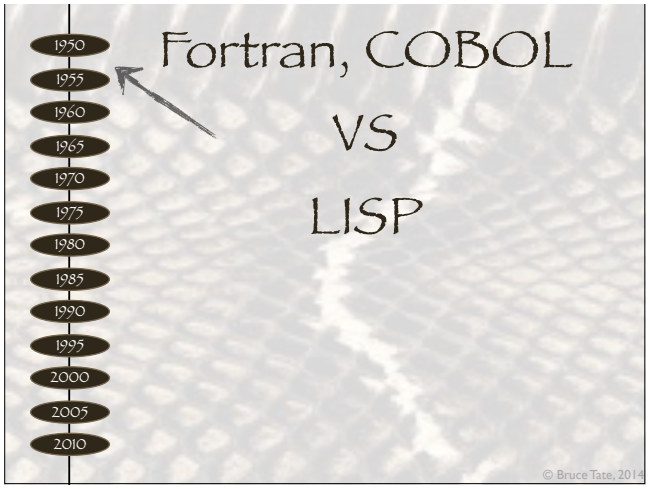
And we have mutable state.



So what does that have to do with programming languages?



New programming paradigm: high level language
Computers were faster
The problems were too complex for the programming model



When you look at a list of languages,
try to see the decisions that we made along the way that impact what we use

MTV had a show called Celebrity Deathmatch that might pit Madona vs Britney Spears

LISP

FORTRAN, COBOL

Lisp supports: high order functions, garbage collection, symbols, data as code...

New languages emerge commercially because they solve a problem

It is a higher abstraction.
There is a cost... one we were not willing to pay.
Lisp is harder for beginners to read... harder to understand.

(LISP)

FORTRAN, COBOL

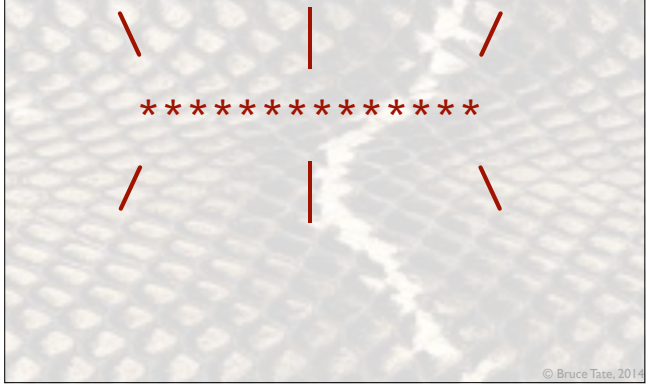
(lisp)

FORTRAN, COBOL

This marketplace decision has had a profound impact on our coding today

```
(reverse '(1 2))
```


Declarative. Beautiful.
The code describes the task



Gone

```
(sleep '50 'years)
```


Gone

```
      C     SWAP X,Y
      C
  100       TEMP = X
            X = Y
            Y = TEMP
```


Lots of assumptions.
Assumptions 1: There is only one user;
Assumption 2: these will be run in order;

Time is a program counter, values change with time

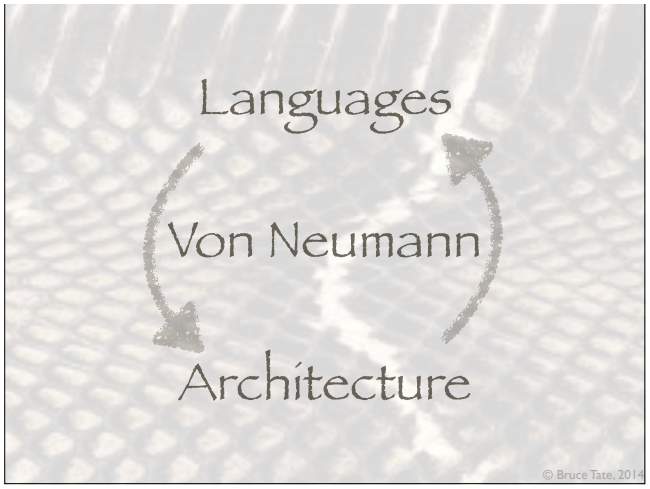
Most languages we use today accepted the compromise that started 50 years ago

Rich Hickey's language

People at this conference know exactly what this means

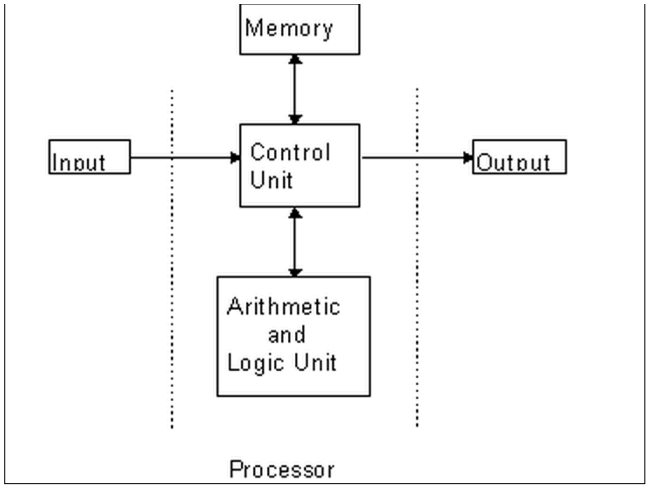
Are you getting the picture?


© Bruce Tate, 2014



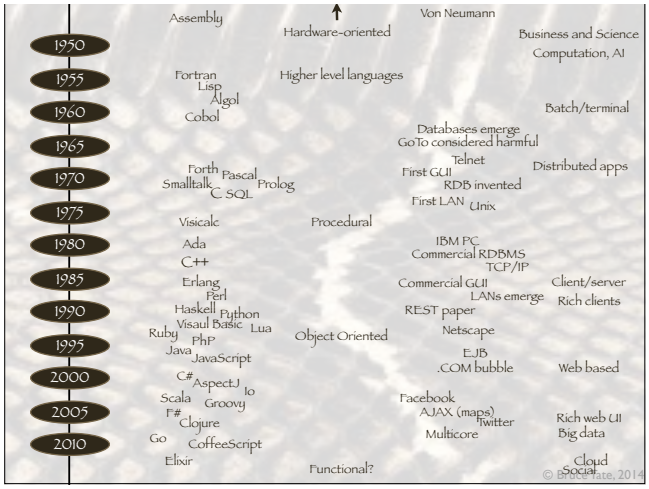
The languages reinforce the architecture
The architecture reinforces the languages.

Did language designers really care?

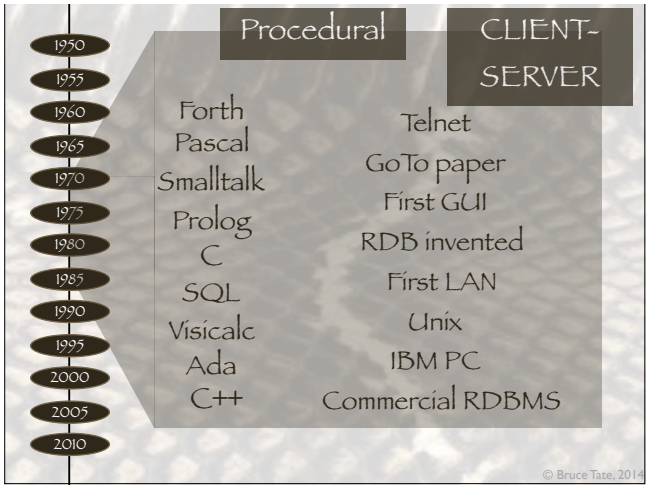


TWENTY FIVE YEARS LATER

first page of a Fortran training manual.



Are you depressed? Good.
Let's move on



Interesting. Stack-based language, functional programming model, a logic-based language
High level languages aren't enough: we have moved on to procedural programming
Database, networking, gui: performance! Many trained in Pascal; would work in C

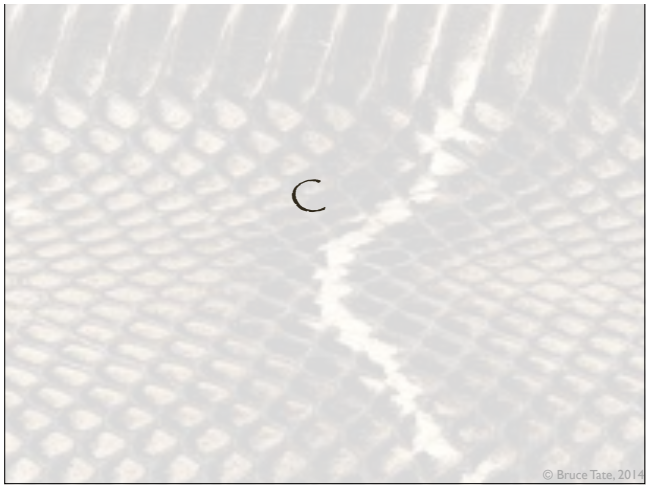
This time, applications language versus a systems language

```
Transcript show: 'hello world'.
```

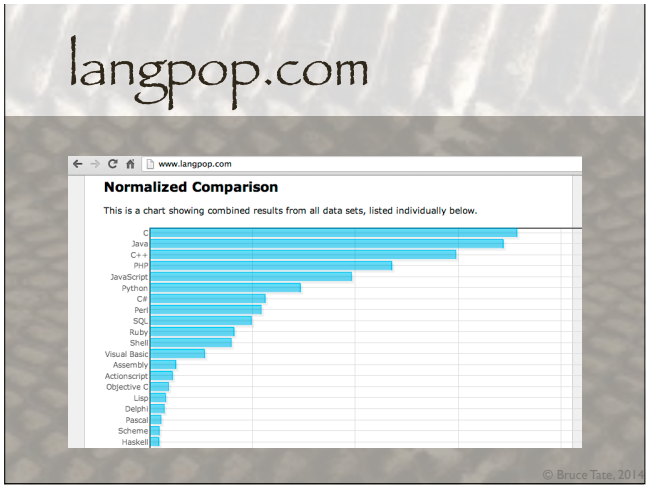
```
#include<stdio.h>
main()
{
    printf("Hello World");
}
```


We picked smalltalk... it could have been one of many
C: A Language designed to build an operating system
We've been using it and it's descendants as a general purpose language since 1980 or so

Really, applications language (Smalltalk, Pascal, Basic, Cobol, Fortran) vs C
New languages emerge commercially because they solve a problem

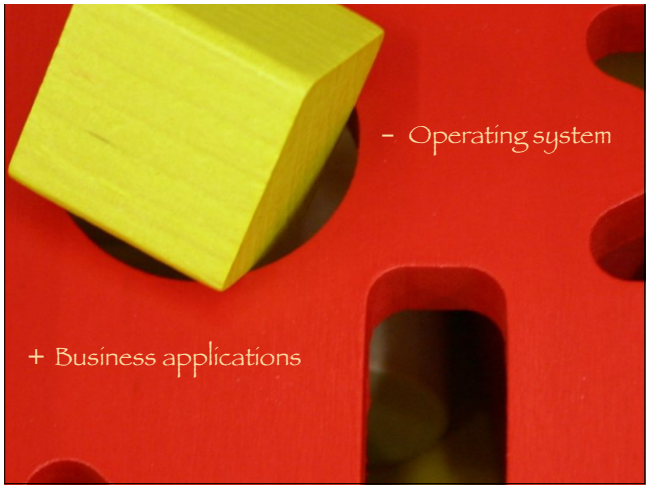


C had performance and portability
GUIs and networking demanded performance.



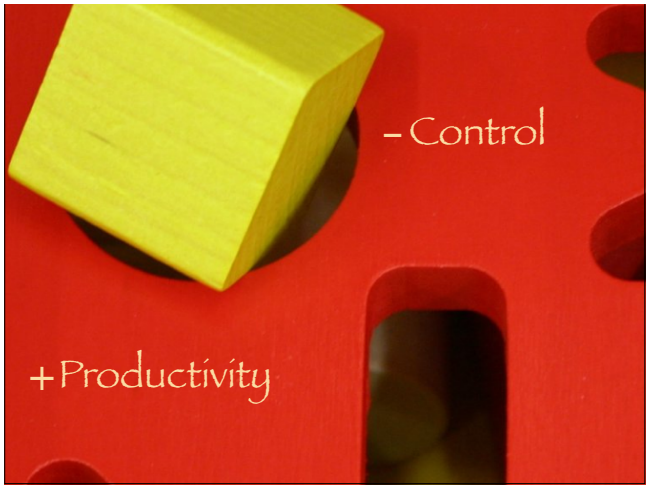
Maybe the biggest winner of all time.
Not an exact measurement of popularity

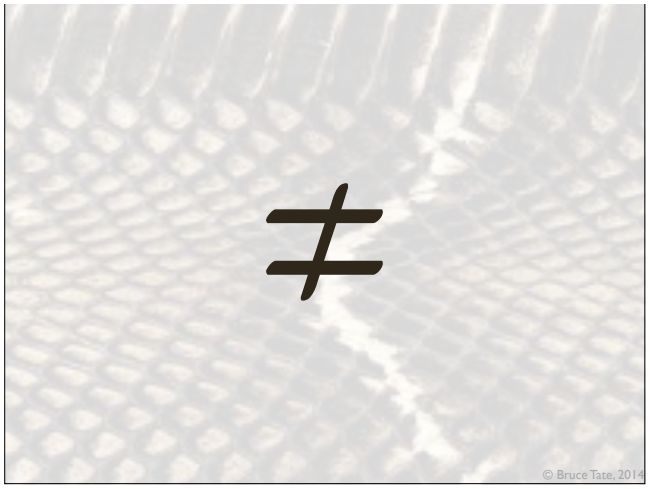
Top 3 languages, 4/5, 5/7



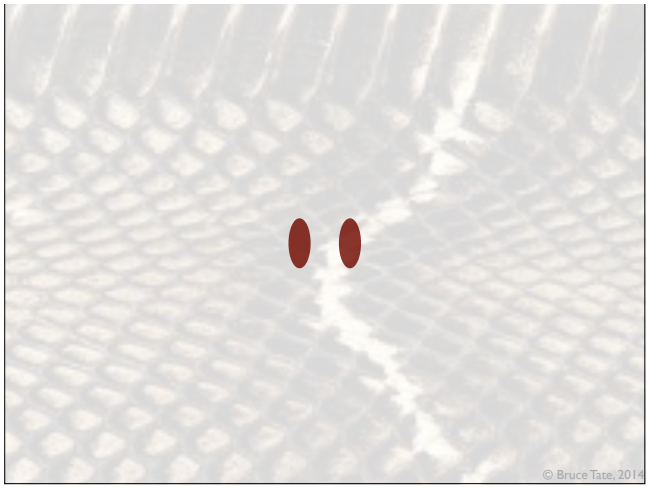
C was designed to build operating systems.
We used it to also build business applications.

C is a compromise between assembler and a high level language



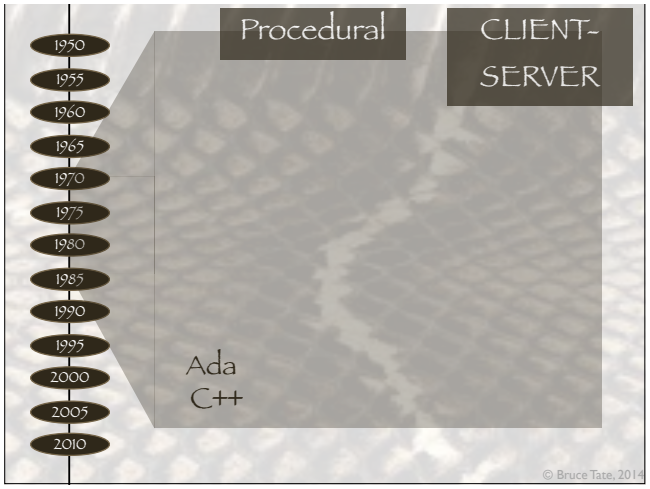


These problems are not the same.
Languages to solve them should not be either.



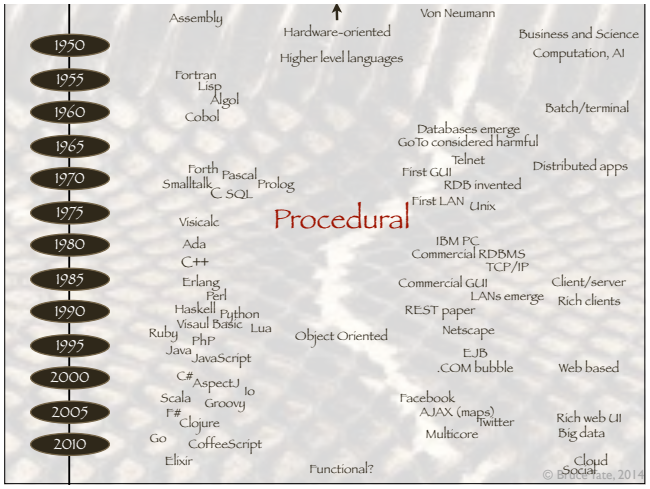
We chose wrong. The impact was catastrophic
I am not saying that C is a bad language
It's the wrong language for many of the jobs we threw at it

Before we leave this period...

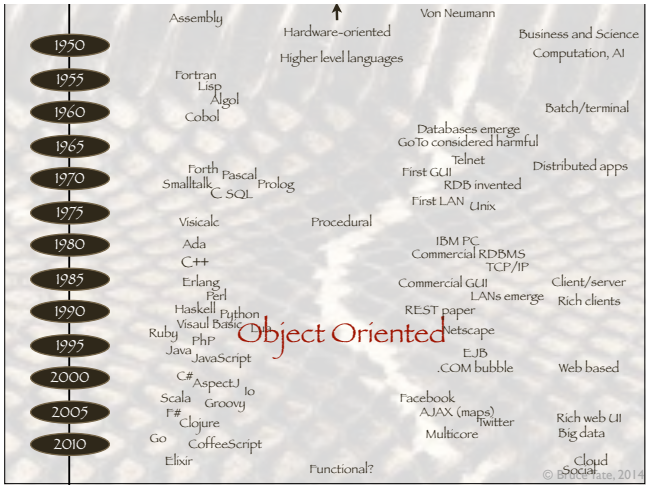


Look at the last two languages

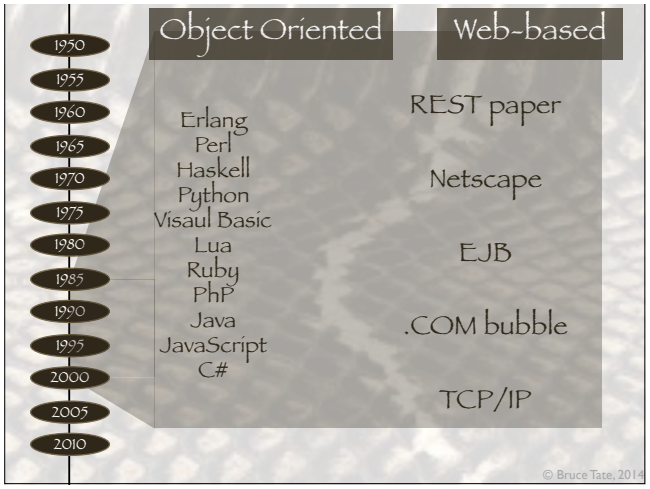
These are bridge languages. A bridge near my home.
Pascal was in effect a bridge language that taught us structured programming
New paradigms generally don't emerge in one shot
C++ and Ada were bridge languages. Movement toward OOP.



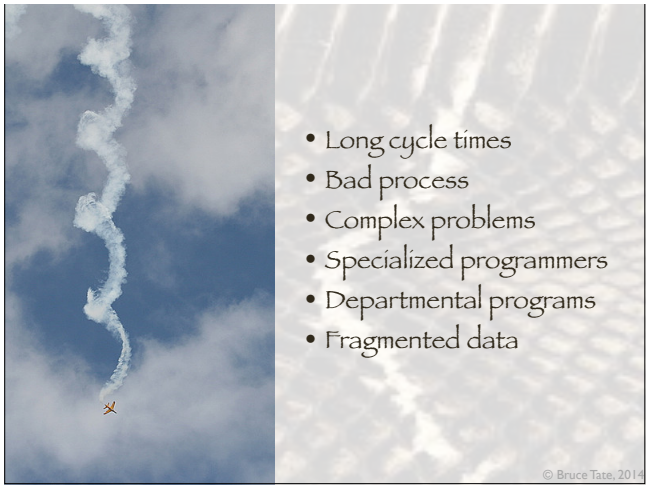
Procedural programming is out of gas.
- database, GUI, distributed transactions,



Procedural programming is out of gas.
- database, GUI, distributed transactions,



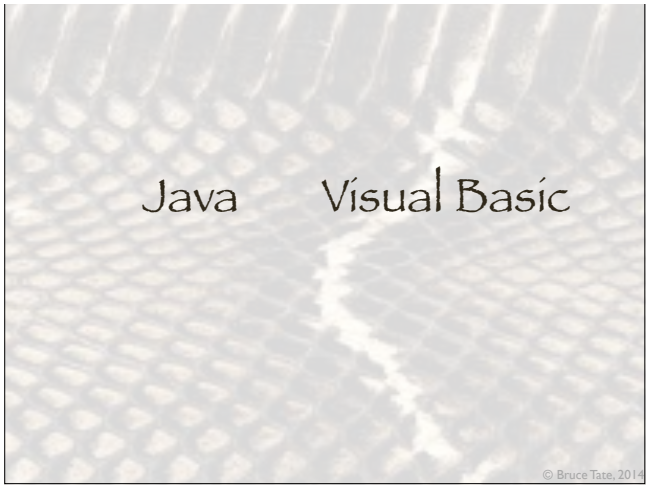
This one is going to hurt for a while



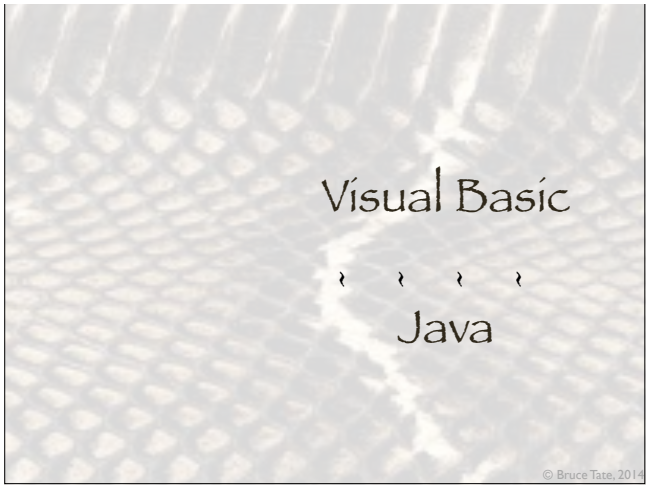
Centralized control for IT is failing.
IBM loses control to Microsoft
PL1 to C to Visual Basic

The same people who abandoned IT
said save us

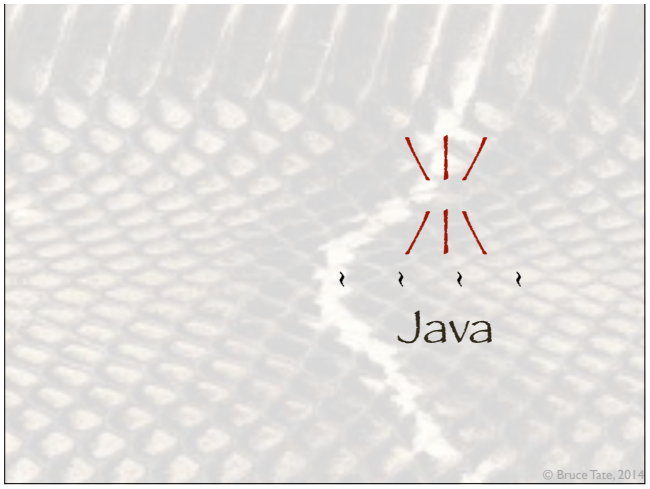
This time, applications language versus a systems language



VB put unprecedented control into the hands of departments
But Before Java, deployment was THE problem
IT shops could not build and deliver small apps fast enough; VB took hold
Business apps in a scripting language



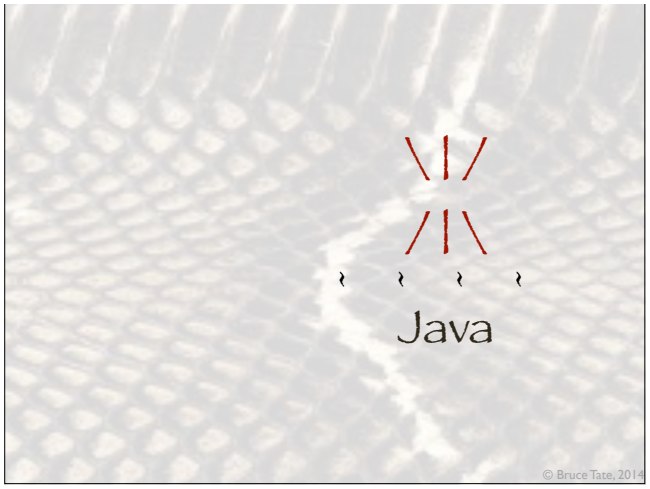
Java solved the deployment problem



Java took the market by bringing along C++ developers
It had the benefit of a bridge langauge

Poor Smalltalk guys never saw it coming... biding their time until OO emerges
Offer to solve the long development time problem

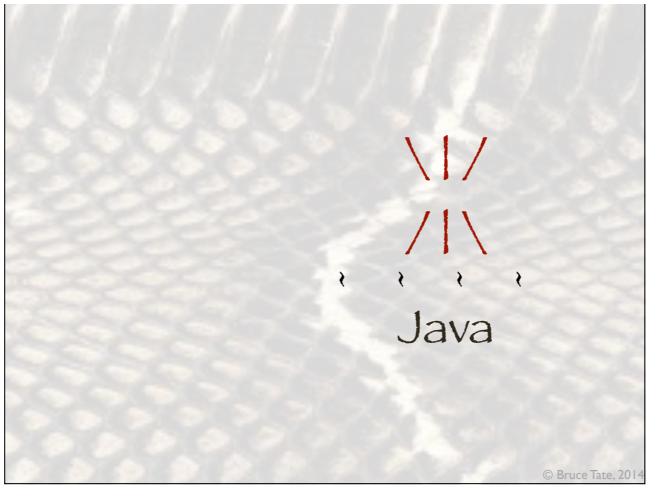
Static vs dynamic typing
Interactive versus compiled
Business applications versus systems code



Java took the market by bringing along C++ developers
Java had distribution
It had the benefit of applets, installing Java on all platforms that matter

If we have to switch paradigms, why OOP?
Isn't functional programming superior?

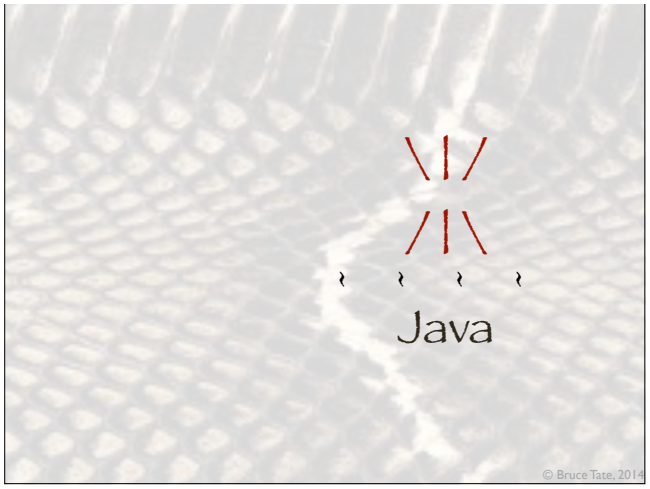
If we have to switch paradigms, why OOP?
Isn't functional programming superior?



It's not the set of language features that matters
it's the business problem you're trying to solve

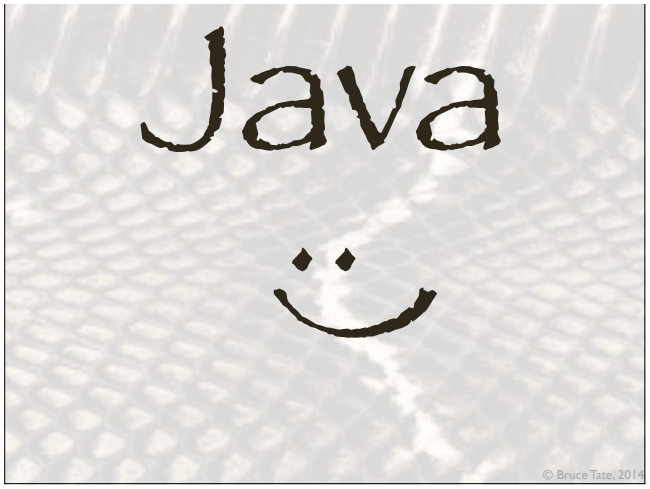
Why not pick the best language for the job?

Because integrating C is hard; C++ harder.
We have just had a taste of integrating and it was ugly ... CORBA, DCE



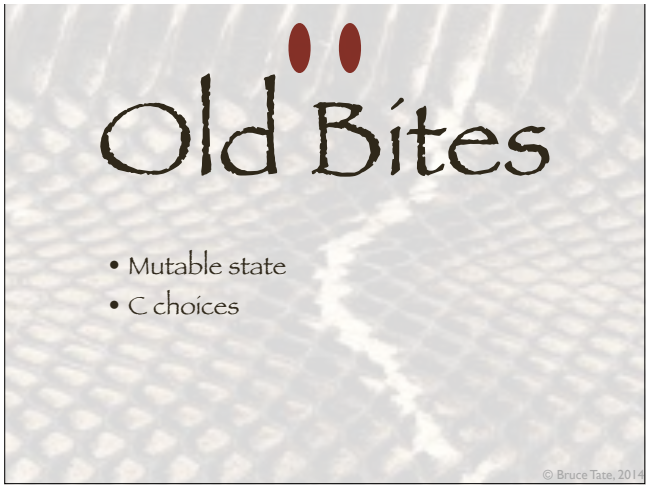
One language to rule them all, and in the darkness bind them.

The big winner. Maybe the biggest winner of all time... in theory

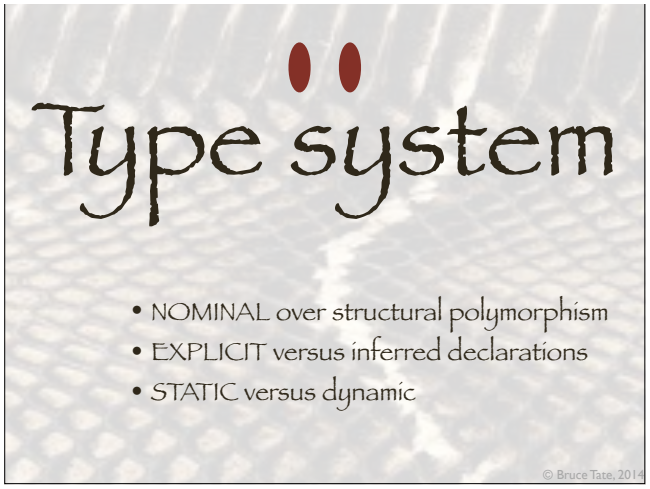


We solved the deployment problem (browser, JVM big wins)
Internet platform, changing paradigms

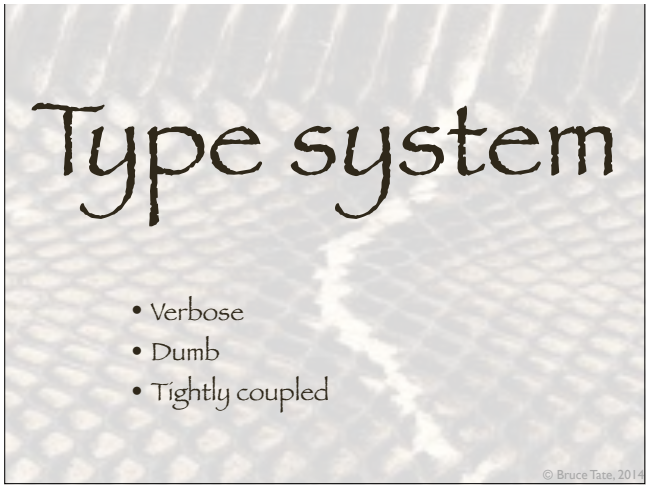
Downside: snakebites



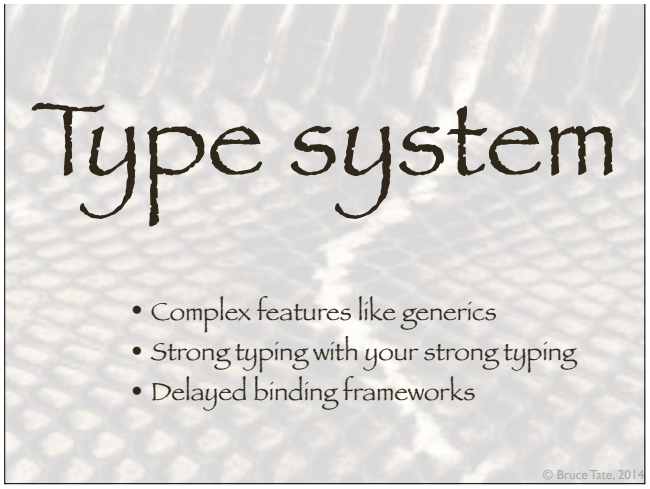
OOP is all about wrapping mutable state with behavior
C decisions: Syntax, static typing, base control structures
Java is good for programming threads and sockets
Must be good for building business applications too.



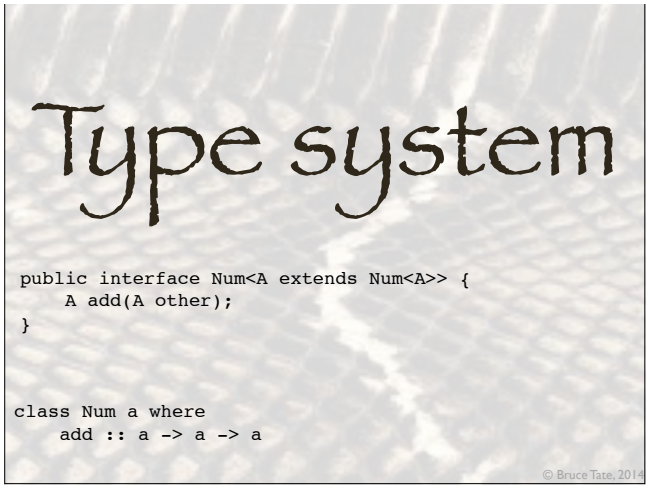
Just a bunch of academic words...Typing based on name rather than structure
Explicit versus inferred typing
Static typing with the above leads to earlier binding



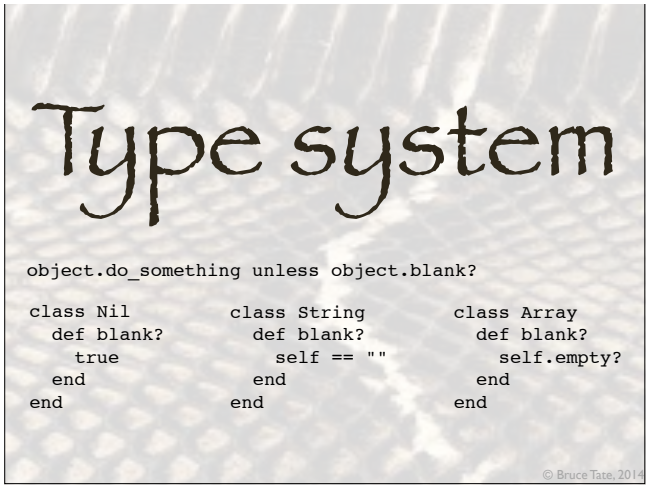
Type erasure. Java thinks this code is trying to implement the same interface twice.



What does that mean to you?
Delayed typing frameworks like
EJB/EJB2/EJB3/Spring/ASPECTJ



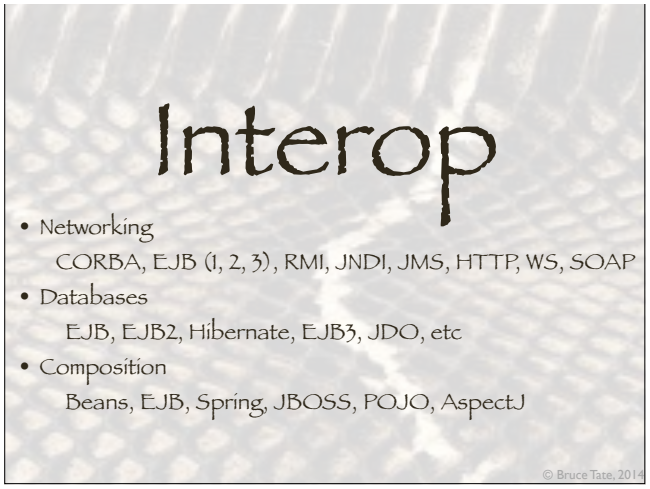
This is an example of the power of Java's generics versus Haskell's type classes



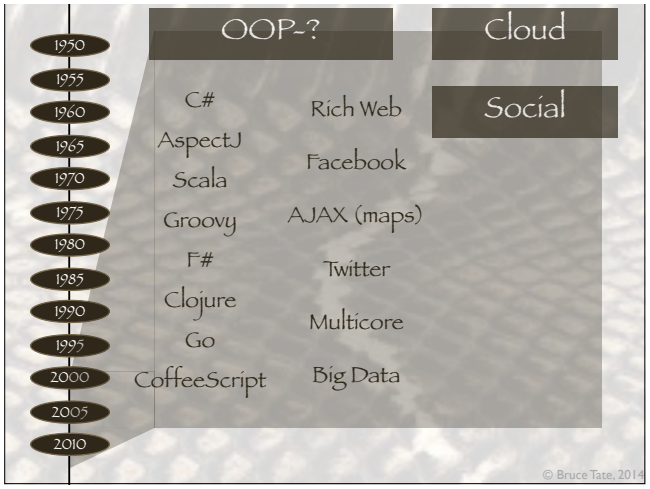
Dynamic typing is not perfect, but it helps.

There's a reason that Smalltalk used dynamic typing. For OO, it helps you build cleaner, simpler code

The bad kind of state. Not enough “final”
The wrong kind of polymorphism. Nominal versus structural.
Inheritance has turned out to be problematic.

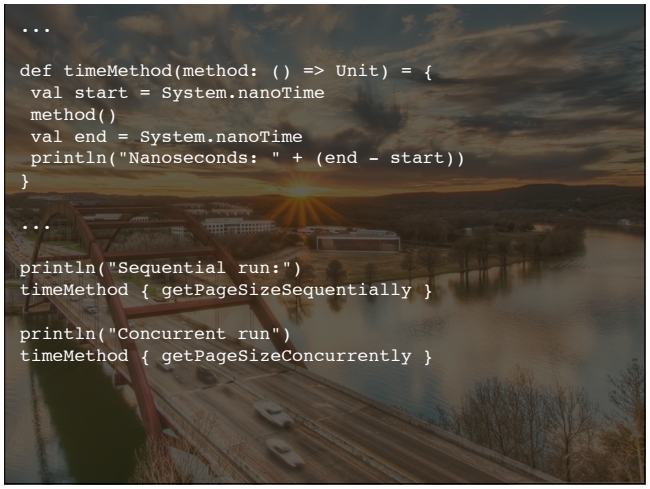


Around the edges, OOP has been a disaster.
Inheritance doesn’t work across system boundaries. Contrast with Erlang...



Look at the languages that are trying to fix things in Java or JavaScript that stink
Big drivers: multicore; the cloud; the user interfaces; complexities of social media and big data

- Python/Ruby
- Groovy is bridging Java toward something that's more dynamic
- JavaScript has OO concepts but is being used in increasingly functional ways



A bridge near my home
New paradigms generally don't emerge in one shot
C++ and Ada were bridge languages. Movement toward OOP.

Scala has many of the things we look for in a bridge language.
It doesn't have to be wildly successful to help.
All code doesn't have to be pure functional for Scala to help.

Who's going to take on Java?

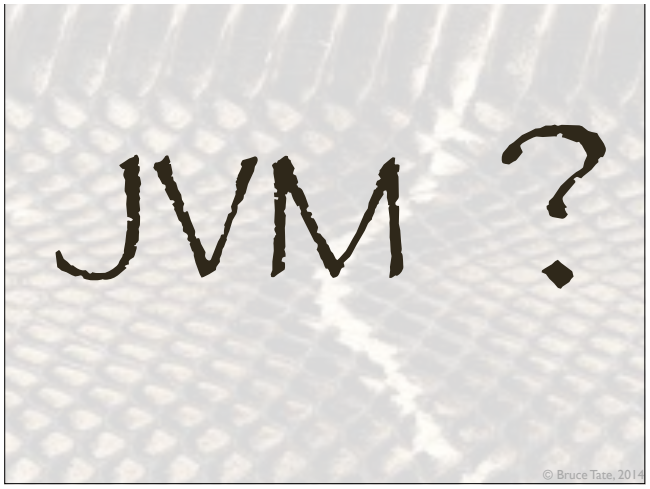
The current landscape:

Can Java handle the problems we're poised to throw at it?

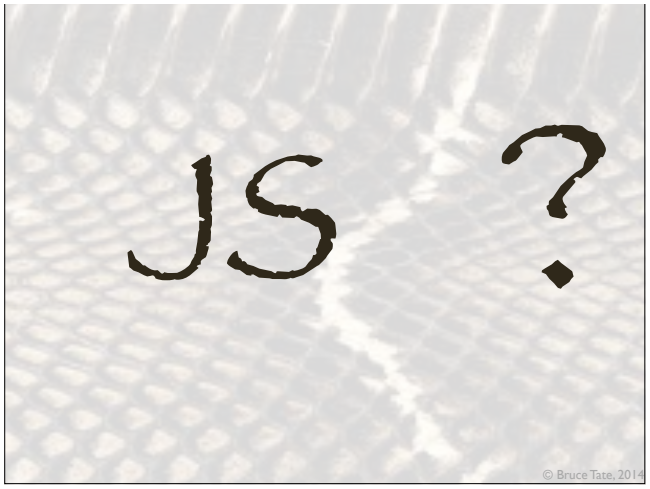
The current landscape:

Can Java handle the problems we're poised to throw at it?

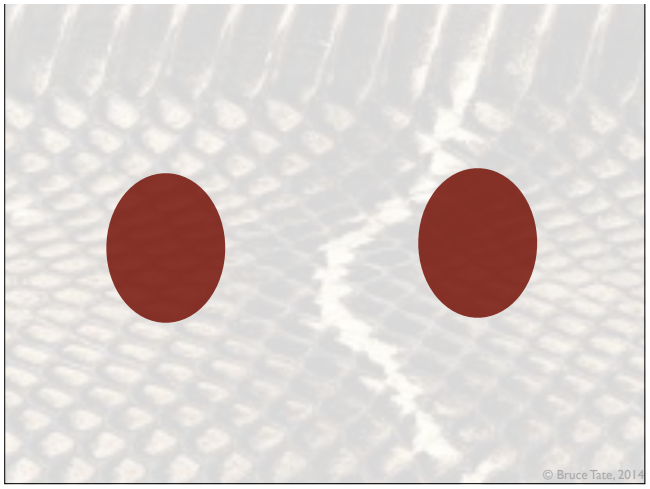
With better APIS across the cloud,
with more diverse problems
do we have to trust one true language any more?



Is the JVM or LAMP the right way to manage the cloud?



Is JavaScript on the client the best that we can do on the client?
Will it escape to the server?



We are going to be making some big decisions soon. My advice: look for the snakebites.
Whatever happens, look for the snakebites. You get to vote.

Use the languages that provide the anti-venom.
Support them.


?
© Bruce Tate, 2014