Pragmaticomm

The Zen has happened before

Ruby is the proof that Smalltalk was right and why we'll use them with LISP's bastard children

Dr John Pagonis, Pragmaticomm Limited
Athens Ruby Meetup #6, CoLab Athens, 20 June 2011

Why am I doing this?

Because if we don't study the past we will screw up the future!

Because if we learn from the past we will invent the future faster!

Because there is a lot of snake oil out there as well as good ideas that we have forgotten or never knew about!

...and because I'm into software archaeology :-)

Before we start... a word from the wise

A reminder from Fred. P. Brooks' '' No Silver Bullet - essence and accidents of software engineering'',1986

There is NO silver bullet!

Language disclaimer...

- I am NOT any programming language bigot, expert or evangelist.
- I am a (legacy) C++ programmer (actually Symbian OS C++) that turned to Ruby.
- I enjoy using C++ for getting to the metal.
- I've taught 'C' to students and professionals because it is so basic (and blunt:-) and thus helps explain how things work.
- I am also very interested in Virtual Machines, Operating systems internals, languages and runtimes in general
- ..... even in the new ISO C++0x spec sometimes :-P

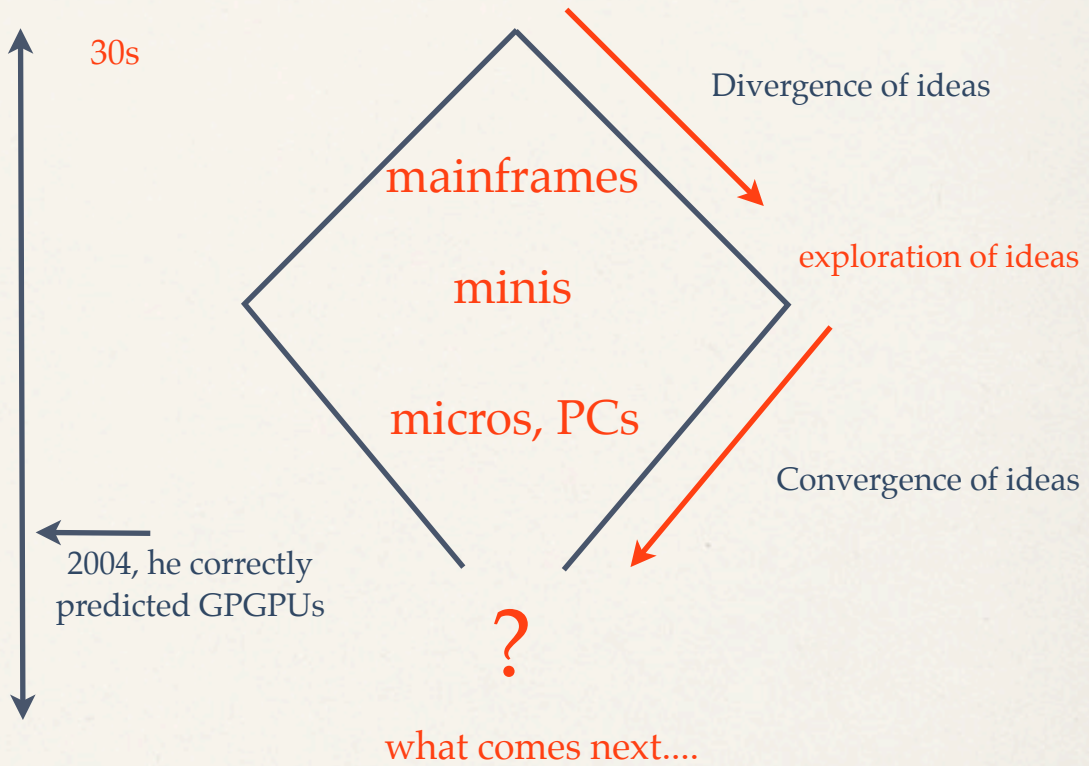
My experience with Ruby

- I was introduced to Ruby during a Python seminar at ACCU 2006 !!
- First got involved in 'skunkworks' while at Symbian circa 2007
- Ported in 2007/2008 with Pragmaticomm the Ruby 1.9.0.0 VM and Ruby 1.9.1p1 VM and some extensions (about 120 KLOC) to Symbian OS v9.1 (for Nokia's Symbian Research dept.) --R.I.P
- I've used Ruby for mobile programming, text filtering, classification, recommender systems, genetic algorithm and web app work (but not Rails:-).

...so I'm no expert, ok!

The diamond shape of computer architecture

I am intrigued by a 2004 F.P. Brooks speech where he discussed how computer architecture since the 30s has followed a diamond shape in terms of the ideas explored and finally established in the marketplace

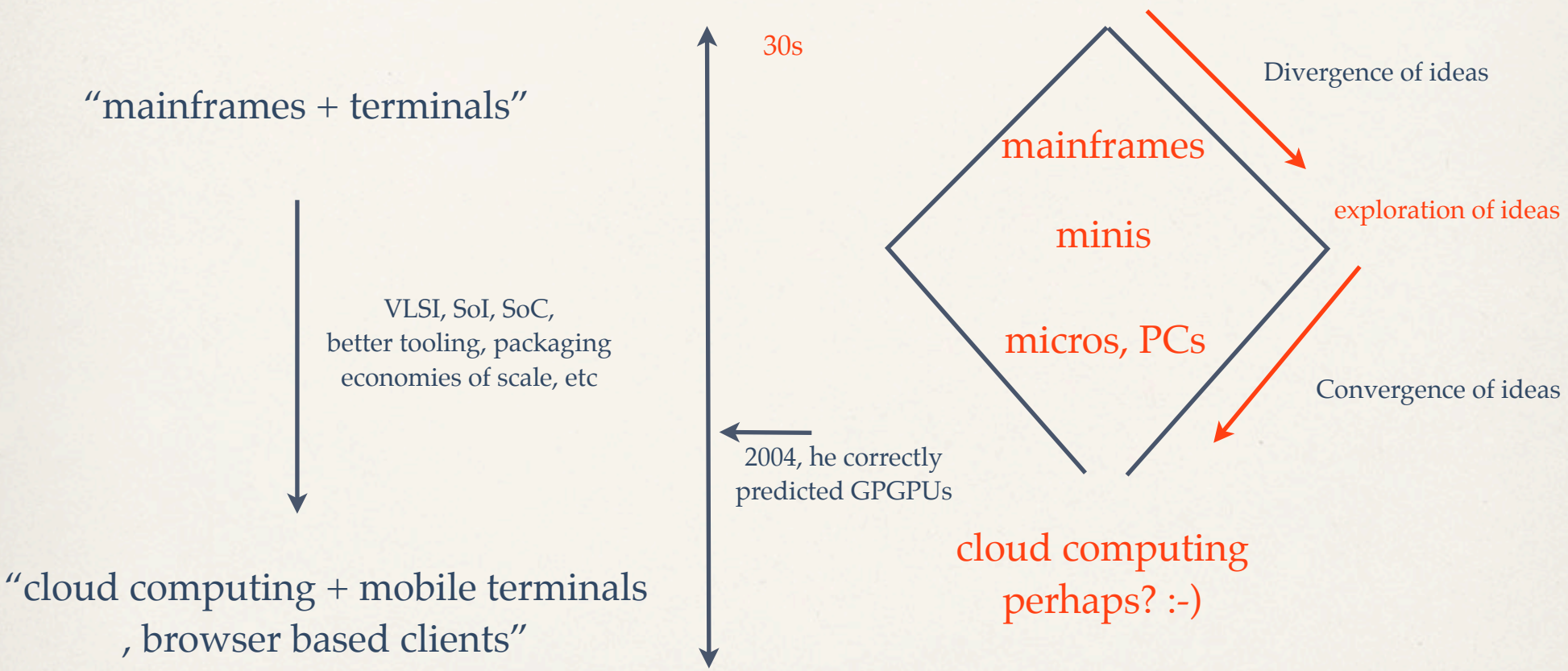


http://pages.cs.wisc.edu/~arch/www/brooks.html

rtsp://vstream.acm.org/FredBrooks/FredBrooks768k/FredBrooksFull768K.mp4

check out mins 13:00 -17:00

The diamond shape of computer architecture



So what about programming?

The striking similarity between the past and the future of computer architecture makes you think...

So what about programming?

The striking similarity between the past and the future of computer architecture makes you think...

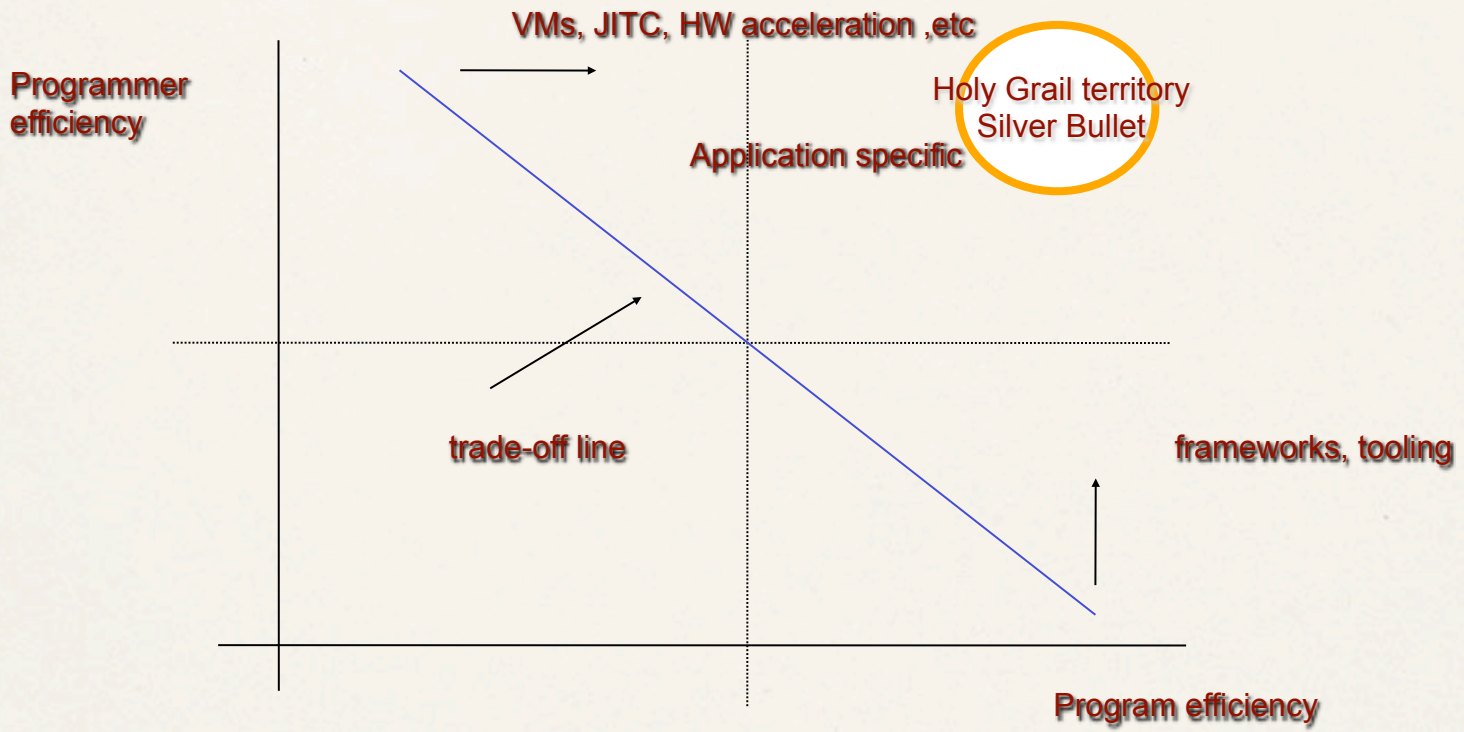
...are we going back to equivalents of punch-cards and programming for batch processing?!!! ; o)

Relax!

Actually many programming languages were designed with humans in mind.

But machines were not ready, so we had to revert to programming that was machine efficient rather than programmer efficient.

Consider this map



A brief timeline of (some) programming languages

Information Processing Language I/II 1954- 1958
Fortran 1954- 1956
ALGOL 1958
COBOL 1959
LISP 1956-1959
Simula 1962-1967
BASIC 1964
BCPL 1967
Logo 1968
'B' 1969
Pascal 1970
Smalltalk 1972 - 1980
SQL 1972
Prolog 1972
'C' 1972
Modula 1975
CwC, C++ 1980 -1983
Object Pascal 1986
Objective-C 1986
Self 1986 - 1995, 2006
Perl 1987
Erlang 1987 - 1993
Haskell 1990
Python 1989 - 1991
AppleScript 1989 - 1993
StrongTalk 1993 - 1996
Visual Basic 1991
Java 1995
Ruby 1995
PHP 1995
LiveScript 1995
'D' 1999
Scala 2002-2006
Clojure 2007
Groovy 2007
Go 2009
CoffeeScript 2009

A brief timeline of (some) programming languages

Information Processing Language I/II 1954- 1958
Fortran 1954- 1956
ALGOL 1958
COBOL 1959
LISP 1956-1959
Simula 1962-1967
BASIC 1964
BCPL 1967
Logo 1968
'B' 1969
Pascal 1970
Smalltalk 1972 - 1980
SQL 1972
Prolog 1972
'C' 1972
Modula 1975
CwC, C++ 1980 -1983
Object Pascal 1986
Objective-C 1986
Self 1986 - 1995, 2006
Perl 1987
Erlang 1987 - 1993
Haskell 1990
Python 1989-1991
AppleScript 1989 - 1993
StrongTalk 1993 - 1996
Visual Basic 1991
Java 1995
Ruby 1995
PHP 1995
LiveScript 1995
'D' 1999
Scala 2002-2006
Clojure 2007
Groovy 2007
Go 2009
CoffeeScript 2009

A brief timeline of (some) programming languages

Information Processing Language I/II 1954- 1958
Fortran 1954- 1956
ALGOL 1958
COBOL 1959
LISP 1956-1959
Simula 1962-1967
BASIC 1964
BCPL 1967
Logo 1968
'B' 1969
Pascal 1970
Smalltalk 1972 - 1980
SQL 1972
Prolog 1972
'C' 1972
Modula 1975
CwC, C++ 1980 -1983
Object Pascal 1986
Objective-C 1986
Self 1986 - 1995, 2006
Perl 1987
Erlang 1987 - 1993
Haskell 1990
Python 1989 - 1991
AppleScript 1989 - 1993
StrongTalk 1993 - 1996
Visual Basic 1991
Java 1995
Ruby 1995
PHP 1995
LiveScript 1995
'D' 1999
Scala 2002-2006
Clojure 2007
Groovy 2007
Go 2009
CoffeeScript 2009

Spot a pattern?

Information Processing Language I/II 1954- 1958
Fortran 1954- 1956
ALGOL 1958
COBOL 1959
LISP 1956-1959
Simula 1962-1967
BASIC 1964
BCPL 1967
Logo 1968
'B' 1969
Pascal 1970
Smalltalk 1972 - 1980
SQL 1972
Prolog 1972
'C' 1972
Modula 1975
CwC, C++ 1980 -1983
Object Pascal 1986

Objective-C 1986
Self 1986 - 1995, 2006
Perl 1987
Erlang 1987 - 1993
Haskell 1990
Python 1989 - 1991
AppleScript 1989 - 1993
StrongTalk 1993 - 1996
Visual Basic 1991
Java 1995
Ruby 1995
PHP 1995
LiveScript 1995
'D' 1999
Scala 2002-2006
Clojure 2007
Groovy 2007
Go 2009
CoffeeScript 2009

Spot a pattern? (let me help you:-)

Information Processing Language 1954- 1958
Fortran 1954- 1956
ALGOL 1958
COBOL 1959
LISP 1956-1959
Simula 1962-1967
BASIC 1964
BCPL 1967
Logo 1968
'B' 1969
Pascal 1970
Smalltalk 1972 - 1980
SQL 1972
Prolog 1972
'C' 1972
Modula 1975
CwC, C++ 1980 -1983
Object Pascal 1986
Objective-C 1986
Self 1986 - 1995, 2006
Perl 1987
Erlang 1987 - 1993
Haskell 1990
Python 1989 - 1991
AppleScript 1989 - 1993
StrongTalk 1993 - 1996
Visual Basic 1991
Java 1995
Ruby 1995
PHP 1995
LiveScript 1995
'D' 1999
Scala 2002-2006
Clojure 2007
Groovy 2007
Go 2009
CoffeeScript 2009

You can also observe this on O'Reilly's 'History of Programming Languages' Map
http://oreilly.com/news/graphics/prog_lang_poster.pdf

?

So are we going back to the ideas of LISP and Smalltalk?!

We will never ever get rid of Fortan and C of course :-)

Smalltalk and Ruby (some characteristics)

Smalltalk

- Object oriented (all the way)
- Dynamically typed
- Strongly typed
- Interpreted and/or executed in VM
- Garbage collected
- Reflective
- Extensible
- Cross-platform
- Smalltalk is typically written in itself
- Has only 5 reserved keywords
- No control constructs
- Keyword syntax
- Code blocks are objects
- Makes extensive use of REPL
- Makes use of a persistent image
- Very advanced VMs

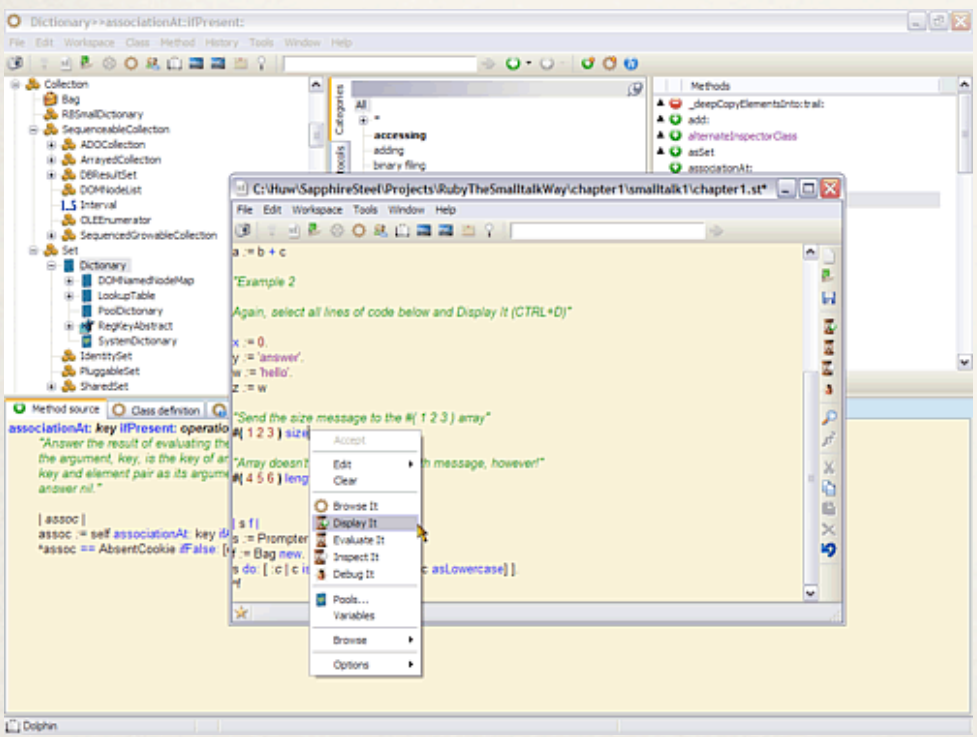
Ruby

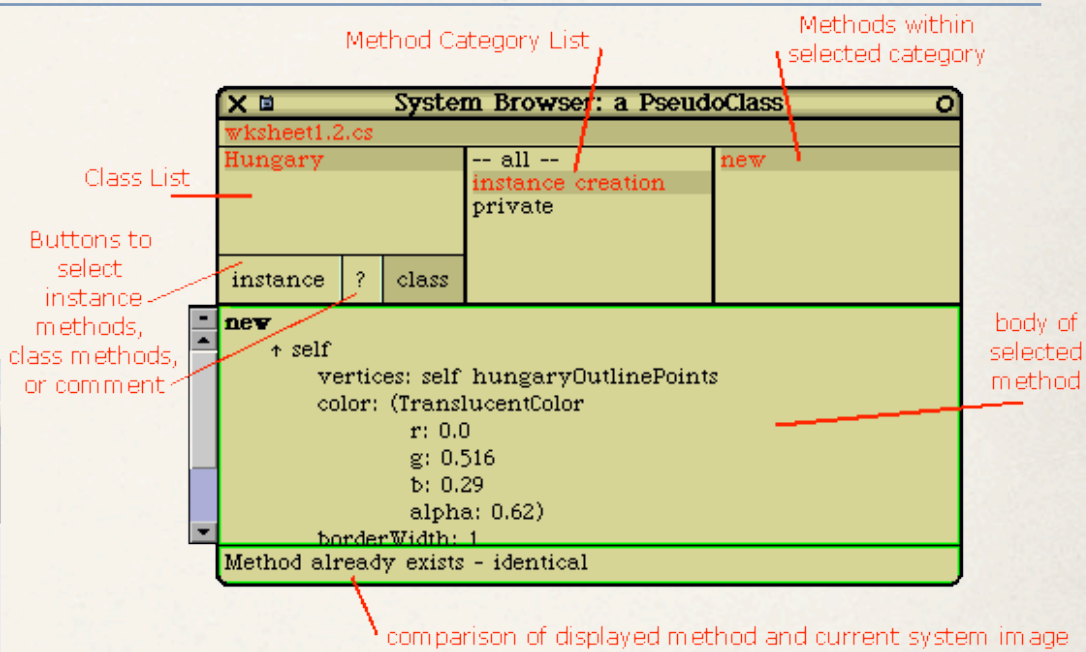
- Ruby is a genuine object-oriented language
- The result of every expression is an object
- Objects are garbage collected
- Like with Smalltalk and Objective-C, objects respond to messages
- Such messages contain a method's name together with the parameters that the method may need
- Ruby is a single inheritance language
- Classes can include the functionality of any number of 'mixins'
- Ruby is a dynamic (late-binding) language
- There is access control in Ruby
- You can use curly braces {} if you want to :-) (because it is important:-)
- Ruby syntax is easy, pleasant and familiar to most

I am not an expert but they look like they have too many things in common

Smalltalk IDE

In Smalltalk, the language and the IDE are traditionally coupled so that the IDE itself can be manipulated in Smalltalk by every programmer while working on a project.

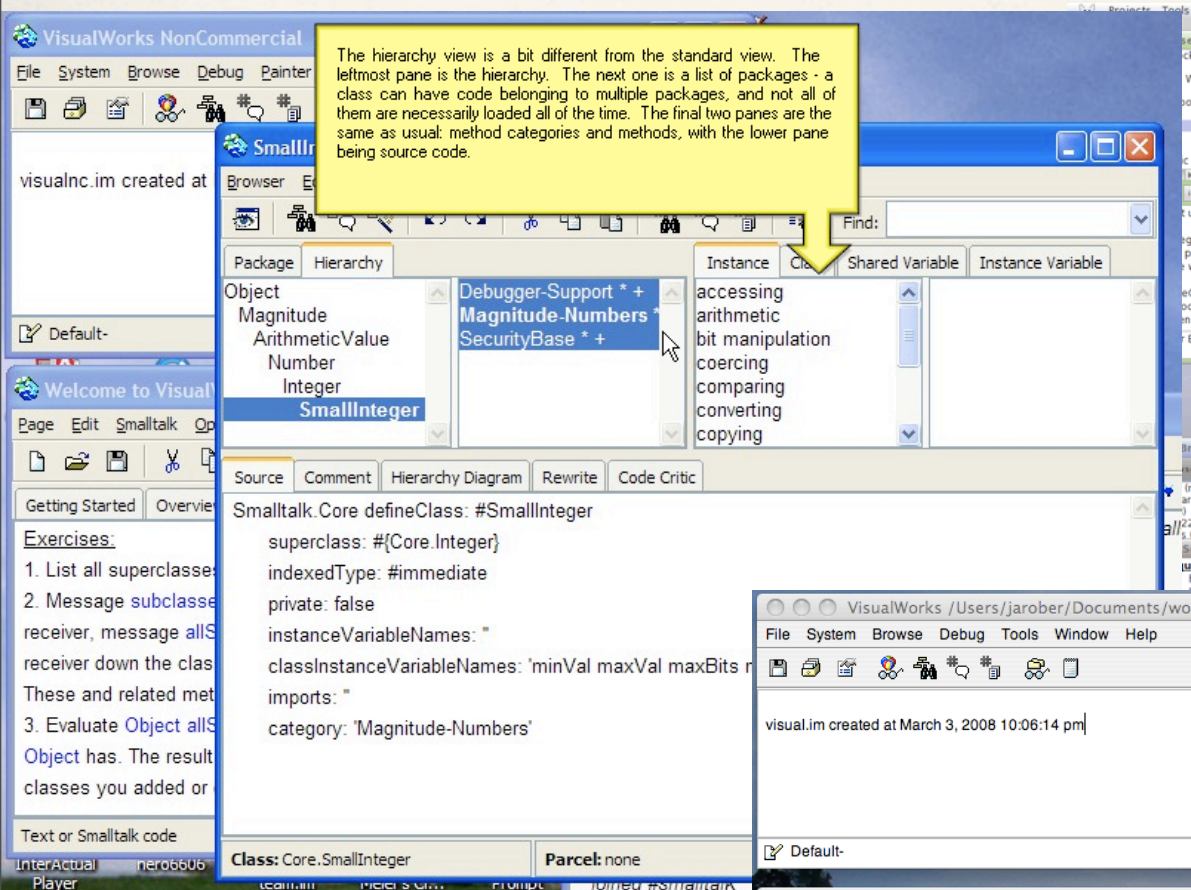




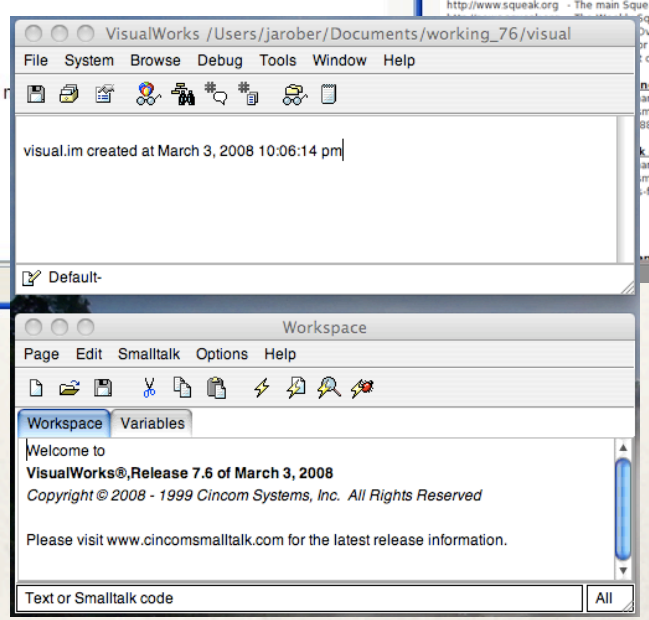
In Smalltalk, typically the state of the running program and the IDE are saved (and executed) as one image file (of bytecode, source, docs and metadata).

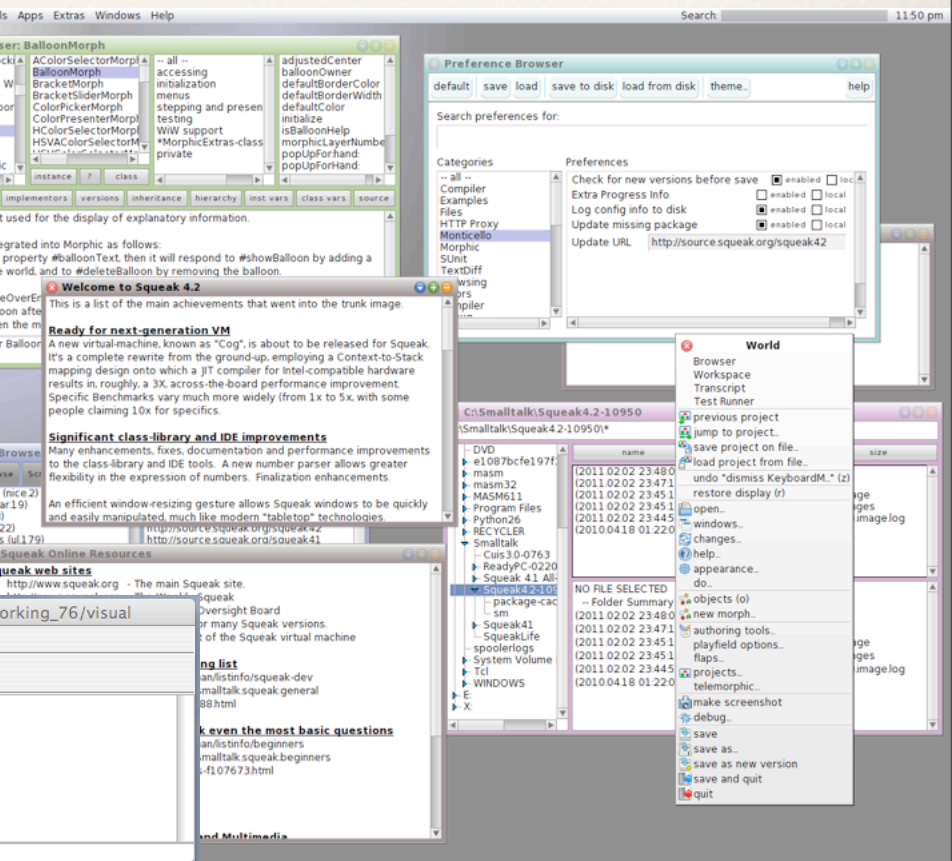
Pragmaticomm

Smalltalk environments - don't you want some for Ruby?



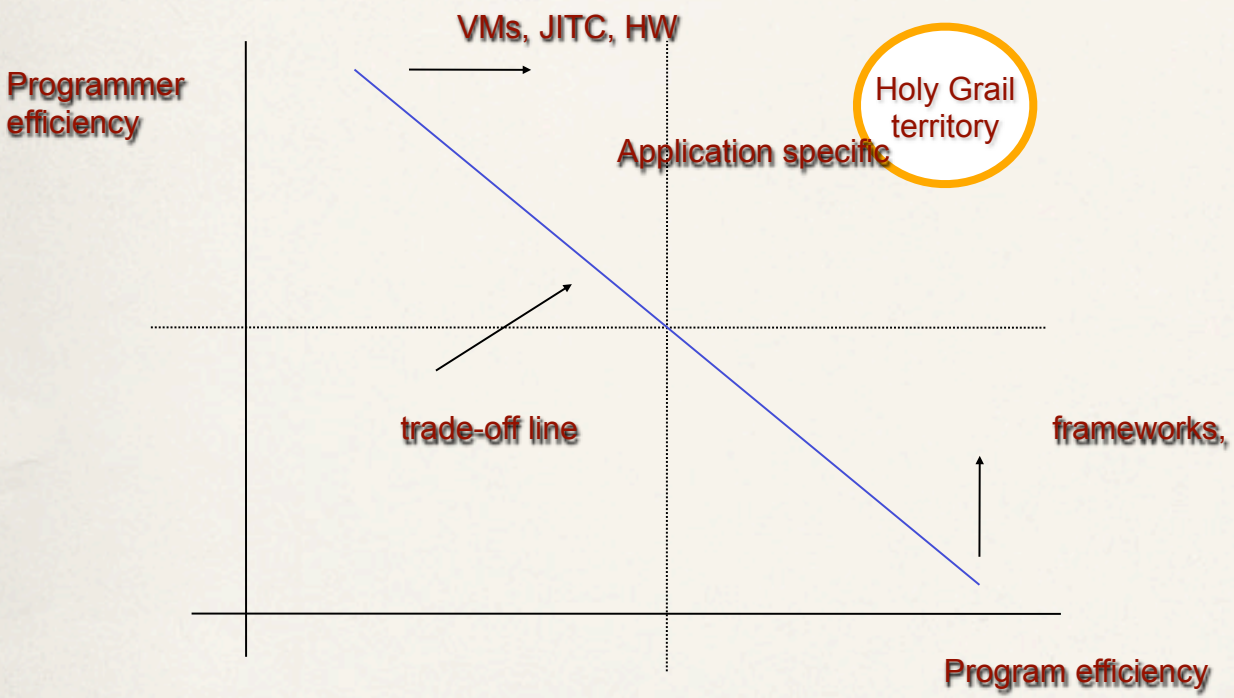
Cincom VisualWorks





Squeak

Software is all about people and economics



Economics have changed

- Programmer time is more expensive
- CPU time is cheaper
- More functionality yields higher system complexity that leads to more entropy which leads to higher cost
- Shorter time to market is vital

Programmers have changed

- Programmers care less
- They have more things to do and info to assimilate

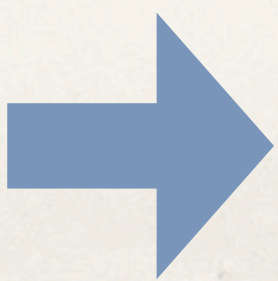
More than ever before, it is that the fewer the people, the better the quality of the code and cheaper the cost of the product. So they need to be hyper productive. One way to achieve this, is with better suited programming languages.

Machine vs Programmer efficiency (both needed)

Traditionally program efficient languages have adopted static binding and the edit-compile-link-execute cycle into programer lives. They execute as native instructions on a CPU.

Programmer efficient (general purpose) languages have usually opted for late binding and read-evaluate-print-loop interactivity which Smalltalk took to the extreme. They execute under an interpreter or virtual machine which typically slows them down.

Program efficiency



Programmer efficiency

... of course Fortran is a different story :o)

Evolution

As computer hardware evolved and improved
so have Virtual Machines since the 50s

Evolution

As computer hardware evolved and improved
so have Virtual Machines since the 50s

There has been a lot of top notch work to improve
and optimise VMs, due to LISP, Smalltalk and Self

Evolution

As computer hardware evolved and improved
so have Virtual Machines since the 50s

There has been a lot of top notch work to improve
and optimise VMs, due to LISP, Smalltalk and Self

A great amount of VM state of the art has ended up
in Smalltalk and especially Java VMs these days

Evolution

As computer hardware evolved and improved
so have Virtual Machines since the 50s

There has been a lot of top notch work to improve
and optimise VMs, due to LISP, Smalltalk and Self

A great amount of VM state of the art has ended up
in Smalltalk and especially Java VMs these days

So is it that to invent the future for Ruby we need
to look back into the success and failure of the
Smalltalk family?

Then why do we use Ruby ?

Smalltalk which is still alive and kicking in various advanced, enterprise, embedded and mission critical systems, could be characterised as “disruptive technology” that didn’t “cross the chasm”. This was partly due to politics, business decisions and the available technology at the time of its introduction.

With Ruby we’ve been luckier because we avoided the politics and both technology and people were ready for its acceptance.

Ironically, Java is one of the reasons why the best Smalltalk implementations and research from IBM, OTI and Sun were (mostly) abandoned to later fuel JVMs !!!

Spot another pattern?

Information Processing Language I/II 1954- 1958
Fortran 1954- 1956
ALGOL 1958
COBOL 1959
LISP 1956-1959
Simula 1962-1967
BASIC 1964
BCPL 1967
Logo 1968
'B' 1969
Pascal 1970
Smalltalk 1972 - 1980
SQL 1972
Prolog 1972
'C' 1972
Modula 1975
CwC, C++ 1980 -1983
Object Pascal 1986
Objective-C 1986
Self 1986 - 1995, 2006
Perl 1987
Erlang 1987 - 1993
Haskell 1990
Python 1989 - 1991
AppleScript 1989 - 1993
StrongTalk 1993 - 1996
Visual Basic 1991
Java 1995
Ruby 1995
PHP 1995
LiveScript 1995
'D' 1999
Scala 2002-2006
Clojure 2007
Groovy 2007
Go 2009
CoffeeScript 2009

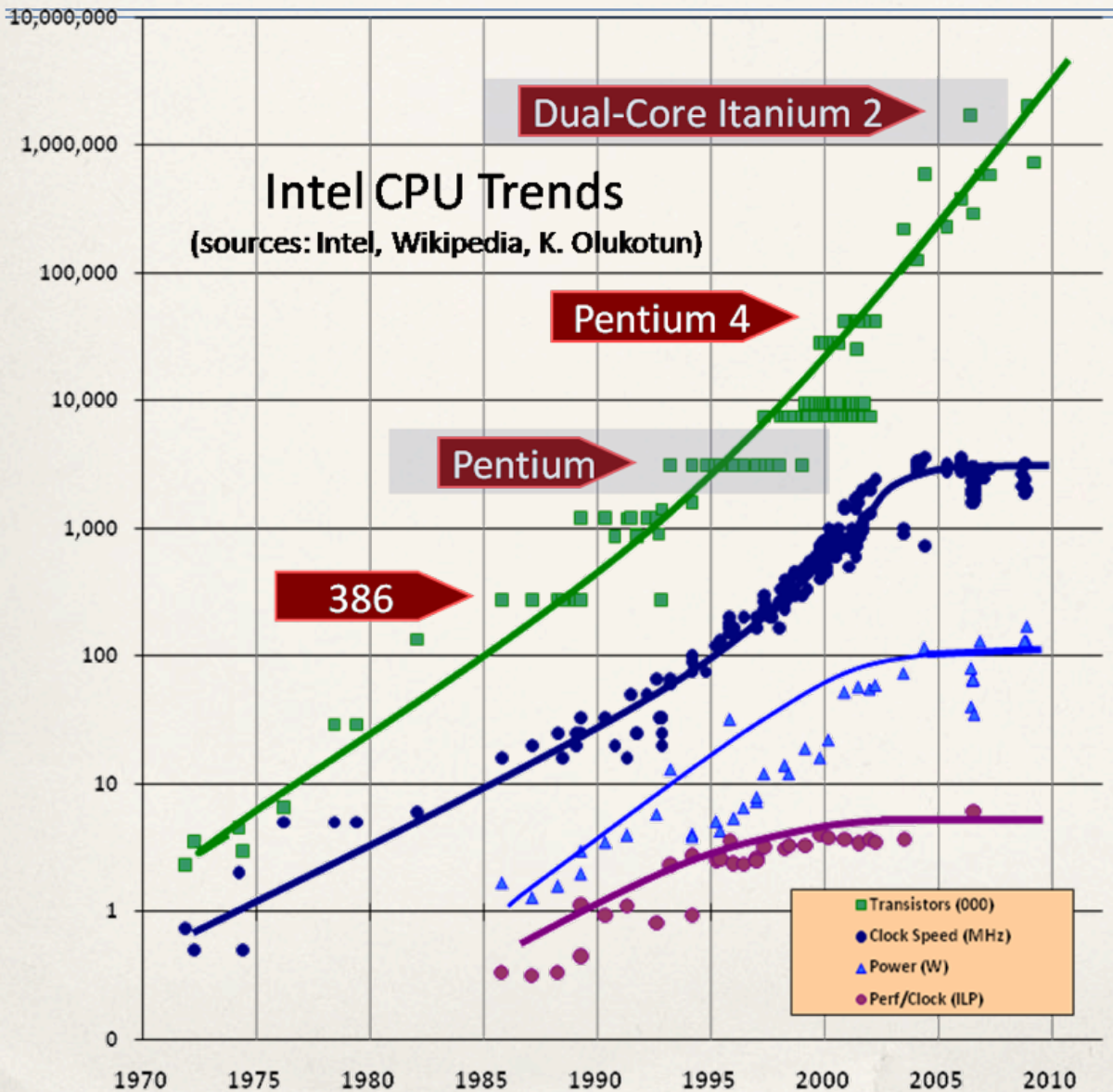
Spot another pattern?

Information Processing Language I/II 1954- 1958
Fortran 1954- 1956
ALGOL 1958
COBOL 1959
LISP 1956-1959
Simula 1962-1967
BASIC 1964
BCPL 1967
Logo 1968
'B' 1969
Pascal 1970
Smalltalk 1972 - 1980
SQL 1972
Prolog 1972
'C' 1972
Modula 1975
CwC, C++ 1980 -1983
Object Pascal 1986

Objective-C 1986
Self 1986 - 1995, 2006
Perl 1987
Erlang 1987 - 1993
Haskell 1990
Python 1989 - 1991
AppleScript 1989 - 1993
StrongTalk 1993 - 1996
Visual Basic 1991
Java 1995
Ruby 1995
PHP 1995
LiveScript 1995
'D' 1999
Scala 2002-2006
Clojure 2007
Groovy 2007
Go 2009
CoffeeScript 2009

Support for high Concurrency

Of Moore's law and GHz speeds



"The Free lunch is Over" ,

by Herb Sutter, 2005

Multi-core CPUs, GPGPUs, SMP, AMP
Cloud computing and high parallelism
are now the norm from desktop, to server,
to embedded and mobile computing.
There is NO escape!

We can't scale-up, so we must scale-out

=> not easy

graph lifted from
http://www.gotw.ca/publications/concurrency-ddj.htm

So what is the enemy of high concurrency?

So what is the enemy of high concurrency?

State !

Functional programming languages such as LISP present functions which are pure computations that never keep state.

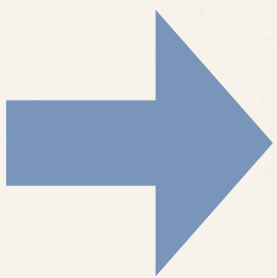
Functions in this case are side-effect free, they take values and return values and every function is itself a value.

Spot another pattern?

Web,
Facebook,
Twitter,
Google,
mobiles,
cloud
computing,
NoSQL,
Graph DBs,
MapReduce,
Hadoop,
Mahoot,
R,
Flickr,
YouTube,
etc.

Spot another pattern?

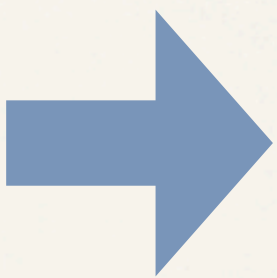
Web,
Facebook,
Twitter,
Google,
mobiles,
cloud
computing,
NoSQL,
Graph DBs,
MapReduce,
Hadoop,
Mahoot,
R,
Flickr,
YouTube,
etc.



Big Data

Spot another pattern?

Web,
Facebook,
Twitter,
Google,
mobiles,
cloud
computing,
NoSQL,
Graph DBs,
MapReduce,
Hadoop,
Mahoot,
R,
Flickr,
Youtube,
etc.



Big Data

So big that many times it is not easy or possible to fit them into RAM in order to do the required processing !!

Hence many times we opt to distribute data for parallel processing.

So what's coming?

- OOP helps us battle complexity and build large systems
- FP helps us battle parallelism and to process "big data"

Smalltalk's and LISP's children combined are excellent candidates for dealing with the future (as well as for being the basis for DSLs).

Multi-paradigm programming of course

For a moment it looked like disciplines represented by Smalltalk and LISP were going to be united on top the ideas of the JVM and .NET/DLR.

Unfortunately at the moment politics again seem to be destroying this...

This is why I am happy that projects like MacRuby, MagLev and Rubinius exist!

http://www.macruby.org/ uses the LLVM infrastructure for JIT compilation and AOT compilation, no GIL, multithreaded GC

http://ruby.gemstone.com/ is based on a highly optimised and proven Smalltalk VM

http://rubini.us/ is following the Smalltalk VM philosophy of building the Ruby VM in Ruby taht allows many optimisations

and of course I'm really happy that Ruby 1.9.x
YARV and JRuby keep us going every day :-)

Wouldn't it be nice

- To use one (Ruby) bytecode to execute Ruby, Clojure, JavaScript etc natively on the same VM side-by-side?
- To have an agreed spec for this bytecode so that we could have many compatible implementations?
- To have these VMs handle concurrency efficiently?
- To not have to wait more and hope for JSR-292 and pray that politics won't derail JVMs?
- Have a Smalltalk-like IDE built entirely in Ruby (like a Redcar ++ http://redcareditor.com)?
- To be able to reuse with the above all existing native libraries (C,C++ etc)?

To probe further

- “The Mythical Man-month” by F.P. Brooks, 1975
- “Object-Oriented Programming: An evolutionary approach” by Brad. J. Cox and A. J. Novobilski, 1986
- “Smalltalk-80: The interactive programming environment” by Adele Goldberg, 1984 (The red book)
- “Smalltalk-80: The language and its implementation” by Adele Goldberg and David Robson, 1983 (The blue book)
- “Smalltalk-80: Bits of history words of advise” by Glenn Krasner, 1983 (The green book)
- “Common LISP: The Language” by Guy L. Steele jr, 1984
- “Crossing the Chasm” by Geoffrey Moore, 1991/1999

Thank you :-)

www.pagonis.org/Publications

www.pragmaticomm.com

twitter.com/JohnPagonis