mainstream

1. the principal or dominant course, tendency, or trend: the mainstream of American culture.

Random House Dictionary

How Do We Get Dynamic Languages Across the Chasm?
A little background about me

- Compilers, Smalltalk virtual machines, GCs, language design, development tools
- Tektronix Smalltalk/4404
- Helped launch OOPSLA and DLS
- Instantiations: OOD Team Dev. Smalltalk,
- Digitalk/Parcplace-Digitalk: Enterprise Scale Smalltalk
- ANSI Smalltalk
- (Re-) Instantiations: JOVE Java optimizing compiler, Eclipse tools
- Microsoft – JavaScript/ECMAScript 5
Dynamic Language Technology Waves Towards the Mainstream


“Lisp Machines”

High Perf Smalltalk & Lisp

Scripting Languages

Java Interlude

High Perf JavaScript
Being a Successful Innovator

- Have a vision
- Believe it is possible
- Do the right things
- Know your weaknesses
- Adjust to reality
- Don’t give up
1978 Tektronix
Getting a vision

- 2nd West Coast Computer Faire, March 1978
- Alan Kay “Don’t Settle for Less”
“Our curiosity about the Smalltalk-80 system had led us down a primrose path … there was little hope for performance high enough to lure users away from traditional programming systems…”

Blinded by Metaphors

- The Smalltalk Virtual Machine
  - “Just reimplement the VM using your own hardware and run the virtual image.”
- Tweaking wasn’t enough
- The key to Smalltalk performance was understanding that you weren’t building a computer, but implementing a language.
The true believers didn’t give up...  
A flowering of innovation

- Tektronix
- Deutsch/Schiffman
- Bosworth/Andersen
- Dave Thomas’ OTI crew
- Ungar and the self guys

Tektronix 4404
Tektronix Smalltalk
Built 1984
Demo Oct. 2010
Things you never want to see in coding guidelines for your language:

“Avoid allocating objects”

“Minimize how many function/method calls you make”
Holistic Design $\rightarrow$ High Performance

Retrofits seldom achieve satisfactory results

Data representations
Register usage
Code sequences
Activation Records
Closure representation
Memory allocation
GC approach

Procedure encodings
Interp/jit/native code
Caching strategies
Cache invalidation
Encodings
Algorithms
Fast paths and fallbacks
Start with the Fundamentals

- Basic data encodings: values/atoms/OOPs
  - Tagged/untagged, hit bit/low bits, arithmetic instruction sequences
  - Cycle counts on target processors (x86, x64, ARM)
- How fast can you allocate
- Tiny write-barriers
- Fast-path polymorphic resolution
- 0/1/2 argument call/returns
- Minimizing loads/stores

- Common usage statistics and traces are very valuable
- Don’t let the exceptional cases get too slow
  They are probably what makes your language unique
It Takes Three to Become an Expert

- One to learn the problem space
  - What are the key features and semantics of this language
  - What makes it slow
- One to explore the solution space
  - Study the literature
  - Experiment with design alternatives
  - Gain key insights and innovative solutions
- One to “put it all together”
  - Implement a clean, holistic design
Do Some Reading

Dude, I know you just want to write some code, but first you might want to take a look at:

• Representing Type Information in Dynamically Typed Languages, David Gudeman, University of Arizona, TR 93-27, October 1993
• OOPSLA Proceedings, particularly OOPSLA 86 → ≈ 2000
• The Implementation Techniques section of the Online Scheme Bibliography http://library.readscheme.org/page8.html
A Programming Language is a “means” not an “end”
What problems do your users really have?

- Client-side of client/server apps with rich UIs
  - From green screen to “modern” UIs
- Complex analytical business apps for rapidly evolving business sectors
  - Airline pricing
  - Insurance rating engines
  - Trading
  - Intelligence community

“Smalltalk - the Natural Successor to COBOL”

PC AI Mag circa 1994
http://www.pcai.com/web/ai_info/pcai_smalltalk.html
Basic Technology isn’t Enough

- We will never be able to use your languages if you can’t teach our guys how to go about designing object-oriented software

Hallway comment by an early adopter from a large enterprise at OOPSLA’88
Smalltalk became an “Enterprise Class” Development Tool
Cargill Lynx System
Lynx is a global grain trading system that supports over 1,500 users at 150 sites around the U.S. and has been in production for over 15 years.
1997 Smalltalk R.I.P.

- 1995 IBM “bets” on Smalltalk
- 1997 Smalltalk is “dead” for new enterprise developments

How could things go so wrong so fast?
- A fad is not the mainstream
- Solving the wrong problems
  GUI designers and visual programming instead of deployment
- New problems require new solutions
  The Web

Java happened!
The solution that appears to be ready gets adopted (whether it really is or not)
Even so, Smalltalk Lives on in the Enterprise
Smalltalk’s Mark on the Mainstream

- “JIT” runtime techniques
- Generational garbage collection
- IDEs
- Frameworks
- Model-View-Controller
- Object-orient design methodologies and practices
- Software design patterns
- Refactoring and refactoring tools
- Agile development practices
- Test-driven development
The Java (and .NET) Era 1996-2005

- New problem requires new solutions
- Java starts as a fad and blows its browser client opportunity
- Even so, Java and .NET work their way into the server side mainstream
  - Familiar syntax
  - Conventional tools
  - More conventional deployment

- Smalltalk and Lisp language engineers and researchers “defect” to Java and .NET
Dynamic Languages 1995-2005
Retreating Into the Nooks and Crannies

  - “Scripting language … are intended not for writing applications from scratch but rather for combining components”
- Perl
- Python
- Ruby
- Lua
- Early JavaScript
Why no progress in DL performance 1995-2005?

- Starting from simple, unsophisticated interpreters
- Undemanding users and uses
- Coasting on Moore’s law
- Many implementers didn’t believe better performance was possible?
- Mostly unaware of past dynamic language achievements
- The experts and researchers were all working on Java
2005 – AJAX is “Discovered”

- People want to build highly interactive browser apps
- Highly interactive code needs to run close to the user
- The only language that is ubiquitous to all browsers is JavaScript
- Web developers start creating frameworks and doing “real programming” using JavaScript
Fast JavaScript “Engines” Became Important!!

- Some initial tweaking of existing engines but they quickly “hit the wall”
- Lars Bak and the V8 team show that “fast” is actually possible
- Today every major browser is devoting significant resources to a high-performance JavaScript engine
- JavaScript performance for major browsers has generally improved by an order of magnitude or more compared to 2005

A Dynamic Language is again making a run for the mainstream.
Some Observations on the New JavaScript Engines

• Most teams haven’t yet reach that 3rd implementation where it all comes together
  • Some teams are still on their 1st
• The performance bar is still too low
  • It isn’t clear that they yet match the 1995 level of Smalltalk performance
  • Everybody needs to stop chasing Sunspider
• JavaScript is harder to make fast than Smalltalk was
  • Some really new ideas would be helpful
• Why are the memory footprints so large?
• Memory management designs are generally weak
  • Build a great GC and then use it everywhere
• The JS engine needs to be part of a holistic browser design
JavaScript Seems Poised for the Software Development Mainstream

• JavaScript is the only “built-in” programming language for the ubiquitous browser/web application platform
• It isn’t clear how any other language or universal runtime can gain a similar position

JavaScript is the “VM” of the web-client platform

• But remember: We will never be able to use your languages if you can’t ...
• What are the “can’ts” for JavaScript?
  • Make it scalable for large programs
  • Improve it without breaking it
  • Continue improving on the performance, footprint, and power issues
  • Provide a great development experience
  • ...
Onward with JavaScript Innovation

- This is the beginning, not the end of opportunities for JavaScript innovation
- It’s not just about JavaScript, it’s the entire web-client technology stack
- Industry needs research contributions to support and feed the pragmatic engineering of the production implementations
- Researchers need clean, accessible, but realistic and usable research platforms to build within and upon
  - Not just a JavaScript engine but an entire browser technology stack
The Next Era of Computing Has Already Started

Dynamic Languages Are a Mainstream Technology for This Era