


or: Just how General Purpose is it?



## The Prehistory of Java

- What shaped my thinking
- Something of a geek autobiography



## First Learned

- FOCAL
  - build stuff fast!
- PDP-8 assembler
  - build fast stuff!

for the day :-)

## Eventually

- Fortran, Cobol, Basic, CDC assembler, ...
  - jobs
    - Multics Pascal compiler
    - VAX Cobol (never shipped as product)
- Pascal, Simula, Lisp, Algol, IBM assembler, ...
  - when I finally went to school



## Designing my first scripting language: YORKMT

- Started life as a magtape copy program
- Feature creep added data correction, imaging, device handling, ...
- Too many changes!
- Experience with TECO suggested adding scripting. Success!
- Users did unexpected things: instrument calibration, diagnostics, ...



## Lessons along the way

- Threading is cool
  - Hydra, Cm*, SMP BSD, Hibbards Algol68
- ANDF is easy (!)
  - PERQ->VAX code migration (the PhD thesis I should have written)
- The power of trustworthy pointers
  - Multics, Hydra, Cm*, Pascal, Mesa
- Optimizers can do surprising things
  - BLISS, MUMBLE (the other thesis I should have written)
- Code-as-algebra is awesome!
  - PhD thesis, ACE



## Unix Emacs

- Reinforced YORKMT lessons
  - Power of scripting
  - People do the damndest things (everything gets stretched to its limits)
- Added
  - The power of the community

## NeWS

- Networked Extensible Window System
  (predated X11)
- A window system based on scripting....
  PostScript (!!)
- More insanely unusual users
  - (never kid yourself into believing you know what will be done with what you've built)
- "Technical" success, but two nagging issues:
  - Scripting performance
    - (cope with user surprise scale)
  - Security



## Trivia Question:

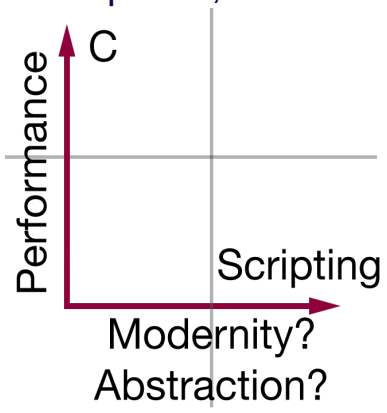
What was Java called,
before it was called "oak"?



## GreenTalk



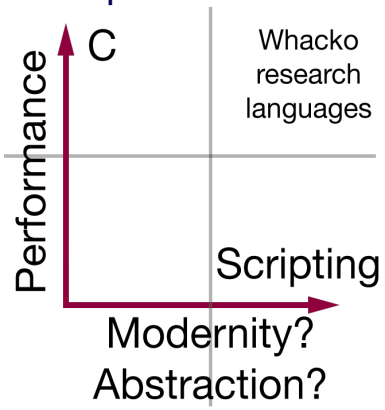
## Perceptions, circa 1990



This dichotomy always bugged me



Perception of Academia





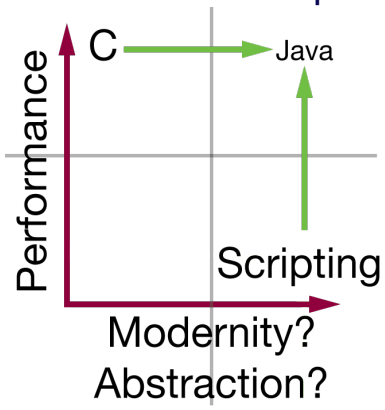

This perception always bugged me



A Big (but quiet) Goal:

How close could I get to a "scripting" feel, without giving up too much?

## A subversive compromise






## A Wolf in Sheeps Clothing

- C syntax to make developers comfortable
- Lisp/Smalltalk/Mesa/Simula/ ... to get the job done



## The original concept

- Was all about building networks of things, orchestrated by a scripting language
- (Unix shells, AppleScript, ...)

but I was real nervous



## Commercial software priority list, 1990

- Compatibility (V2V)
- Performance
- Portability
- Reliability
- Networking
- Multithreading
- Security

## Consumer Electronics priority list

- Security (safety)
- Networking (connectivity)
- Portability (cost: CPU du jour)
- Reliability
- Performance
- Multithreading
- Compatibility (rewrite for every product)



## Shake and Stir

- Security
- Reliability
- Networking
- Portability
- Compatibility
- Multithreading
- Performance



## BUT

- Performance that sucks
- Sucks



## Design space is gravitationally lumpy

"Lisp is a black hole. If you try to design something like Lisp, it gets sucked in and becomes Lisp"

Guy Steele, hallway conversation, 1980ish

But orbital dynamics are chaotic

- Byte coding is another black hole

## ANDF

- Architecturally Neutral Distribution Format
- One of the Holy Grails of the early 1990s
- Got bogged down by politics and complexity
- Tended to be persistent forms of ASTs
- Which said too much about how compilers were build
- Food Fight



## Virtual Machines to the Rescue

- PERQ experience said performance could be good
- verifier invented to enforce security issues
  - imposes restrictions on control&data flow
  - that are also useful for compilers
- Says "just enough"
- Abstracts-away AST issues



## The Big Restriction:

- pointer/object integrity
  - ie. no fraud
- Some languages depend on fraud:
  - C/C++/Objective C



## The Other Big Restriction

- Reactionary method calls
  - Simple simula-esque object model
  - The more complex object models didn't seem to be worth their weight
- A building block
- Tricks with interfaces

## Why a stack machine?

- (why not?)
- No preconceived register architecture
- Compact
- "free" live/dead analysis
- reverse polish encoding of AST



## Control Flow

- Goto statement
  - Caused flow analysis problems
  - Poof!  Dead!
- JSR/RET
  - oops!



## Primitive Types

- They're all about performance
- Design Goal: a=b+c
- in one instruction



## Java performance

- The myth:
  - it's interpreted, therefore slow
- The truth:
  - it's highly optimized
  - dynamic metrics
  - generally beats C/C++
    -2% LINPACK, +4% SciMark
  - often beats Fortran (matricies are the issue)
  - GC is a lot faster than malloc/free
- Dynamic compilation beats static
- No need to be afraid of methods!
  - who needs opcodes?

# Thanks!