
# Parrot in a Nutshell

Dan Sugalski
dan@sidhe.org

# What is Parrot

- The interpreter for perl 6
- A multi-language virtual machine
- An April Fools joke gotten out of hand

# The official point of Parrot

- Build something to support perl 6
- Build a faster perl
- Build a cleaner perl

# The Unofficial point of Parrot

- Generic fast interpreter engine
- Annoy the Mono folks
- Really annoy the python people
- Try and take over the world

# Parrot's guiding goals

- Speed
- Maintainability
- Longevity
- Extensibility
- Flexibility

# Speed

- Faster is good
- A slow interpreter is of little use
- Things almost never get any faster
- Parrot sets the maximum speed

# Maintainability

- Try and keep things black-box
- Try and enforce a coding standard
- Open Source projects have limited engineering resources
- Things rarely get more maintainable
- Maintenance always consumes the most resources for projects that actually finish

# Longevity

- Software always lives on well past the point you think it should
- Designing for long-term expandability puts off the inevitable rewrites
- We're going to be looking at this code for the better part of a decade, so we'd best make sure we want to look at it

# Extensibility

- Everyone wants to add on with C
- Things are never quite fast enough
- There are lots of external libraries worth using
- A good extension mechanism is a great thing
- A bad one really hurts

# Flexibility

- General solutions have more long-term potential than specific ones
- People always do things you don't expect
- If you plan for it now, it doesn't cost more later
- Even if it costs a little more now

# Multi-language capabilities

- Support perl
- Support Dynamic Languages in general
- Nice target for language designers

# Support perl

- Perl 6 was the reason this all started
- Perl has the most complex semantics of all the dynamic languages in common use
- Perl is a major pain to implement, as languages go

# Support Dynamic Languages in general

- Includes Ruby and Python, for example
- Languages that have some compile-time uncertainty
- Actually trivial, just a matter of thinking about it

# Nice Target

- If we express all the semantics, it makes it easier for language designers
- Many now target C, or GCC
- Impedance mismatch lower for many languages

# Play Zork natively

- Yep, that's right
- `parrot -b:zmachine zork.dat`
- Doing this right is harder than doing Java, Python, or .NET bytecode
- If we can do this, we can do anyone's bytecode
- Plus it's really cool

# Interpreter basics

- Chunk of software that executes other software
- A CPU in software
- Generally flexible
- Generally slow
- Easy to write for
- Easy to write
- Portable

# CPU in software

- It's the "V" in VM
- The compiler writer can treat the interpreter as a form of CPU chip
- Allows much easier customization of 'core' behavior
- Sometimes software does become hardware
- But often it shouldn't

# Generally flexible

- Since the core functionality is in software, it's changeable
- Software generally has fewer limits to it than hardware
- It's easier to think about software than hardware, so more people can
- Incremental cost of changes much lower

# Generally slow

- It's a layer of indirection between the user's program and the hardware that executes it
- It's very easy to have impedance mismatch problems
- It's also very easy to let speed dribble away

# Easy to write for

- It's easy to gloss over the tough parts
- The interpreter should express itself in ways that are easy to use
- You built it as the target, so if it isn't, you've done something wrong

# Easy to write

- Interpreters are pure semantics, and semantics aren't that difficult to express
- Though defining the semantics can take much longer
- A simple interpreter can be put together in a week
- (Though parrot is far from simple)
- Just a SMOP, even for the fancy ones

# Portable

- Generally the expressed semantics are mostly platform-neutral
- Means the platform-specific stuff can be isolated
- Platform specific but unimplemented things can be emulated
- Normally just a small cluster of really nasty code to deal with

# Core Parrot Concepts

- Register based
- Language Neutral
- Very high level
- Introspective
- Mutable
- Threaded
- With continuations

# Register Based

- 32 each Integer, String, Float, and PMC
- Registers are a kind of named temporary slot for the interpreter
- Modeled on hardware CPUs
- Generally more efficient than a pure stack-based model

# Language Neutral

- In the *human* sense
- We don't force ASCII, Unicode, Latin-1, or EBCDIC on the user
- The engine has facilities to deal with data regardless of its encoding
- Important when dealing with native-language text, which is most text

# Very high level

- Threads, closures, continuations, aggregate operations, multimethod dispatch all core elements
- Don't relegate the hard stuff to libraries
- Easier on compiler writers
- Bit more of a pain for the interpreter implementers

# Introspective

- Code can inspect the current state of the interpreter
- Most, if not all, the core data structures are accessible
- Includes stack entries, symbol tables, lexical variables
- Also variable arenas, memory pools, and interpreter statistics

# Mutable

- Code can be created on the fly
- Libraries can be dynamically loaded
- Opcodes can be redefined on the fly
- Base variable methods can be overridden at runtime

# Threaded

- Threads are a core element of Parrot
- Threading interpreters is an interesting task
- For nasty values of “interesting”
- They must be designed in from the start

# With Continuations

- Really odd Lisp thing
- Sort of like a closure for control flow
- For reference, exceptions are a simplified kind of continuation
- They *will* rot your brain
- If your brain's already gone, though, they let you do some Terribly Clever Things

# Quick benchmark numbers

Simple MOPS benchmark

| | |
|---|---|
| Ruby 1.6.6: | 207 Sec |
| Perl 5.6.0: | 165 Sec |
| Python 2.2: | 136 Sec |
| Parrot: | 18.75 Sec |
| Parrot (faster): | 9 Sec |
| Parrot (fastest): | 4.9 Sec |

# Fetching Parrot

- http://dev.perl.org
- Anon CVS

  `:pserver:anonymous@cvs.perl.org:/cvs/public`
- http://www.parrotcode.org