

## How not to Design a Scripting Language

Paul Biggar

Department of Computer Science and Statistics  
Trinity College Dublin

StackOverflow London, 28th October, 2009

# How not to Design a Scripting Language

Paul Biggar

Department of Computer Science and Statistics  
Trinity College Dublin

StackOverflow London, 28th October, 2009

1. means I just submitted
2. So that's what this talk is about

## About me

- PhD candidate, Trinity College Dublin
- **Topic:** Compilers, optimizations, scripting languages.

## About me

- PhD candidate, Trinity College Dublin
- **Topic:** Compilers, optimizations, scripting languages.

1. means I just submitted
2. So that's what this talk is about
3. Not much PHP

## About me

- PhD candidate, Trinity College Dublin
- **Topic:** Compilers, optimizations, scripting languages.

### PhD Dissertation

Design and Implementation of an Ahead-of-time PHP Compiler

**phc** (<http://phpcompiler.org>)

## About me

- PhD candidate, Trinity College Dublin
- **Topic:** Compilers, optimizations, scripting languages.

### PhD Dissertation

Design and Implementation of an Ahead-of-time PHP Compiler

**phc** (<http://phpcompiler.org>)

1. really, this is a talk about compilers and scripting languages

## How not to design a scripting language

- Compilers
- Scripting Languages

## How not to design a scripting language

- Compilers
- Scripting Languages

1. That's what's in the back of my head all the time

## How not to design a scripting language

- Compilers
- Scripting Languages
- *Speed*

## How not to design a scripting language

- Compilers
- Scripting Languages
- *Speed*

1. well, no-one actually has a good definition of scripting language!
2. I'm not talking about Bash or Powershell or VB, or some little language you wrote last week

### What is a scripting language?

- Javascript
- Lua
- Perl
- PHP
- Python
- Ruby

## What is a scripting language?

- Javascript
- Lua
- Perl
- PHP
- Python
- Ruby

1. well, no-one actually has a good definition of scripting language!
2. I'm not talking about Bash or Powershell or VB, or some little language you wrote last week
3. exposing their internals

### What is a scripting language?

- Javascript
- Lua
- Perl
- PHP
- Python
- Ruby

#### Common Features:

- Dynamic typing
- Duck typing
- Interpreted by default
- FFI via C API

## What is a scripting language?

- Javascript
- Lua
- Perl
- PHP
- Python
- Ruby

#### Common Features:

- Dynamic typing
- Duck typing
- Interpreted by default
- FFI via C API

## Language implementation

1. Reads one line at a time (kinda)
2. hence used in many scripting langs

- **Interpreters:** Easy, portable

## Language implementation

- **Interpreters:** Easy, portable



1. Converts source code into machine code programs (kinda)
2. Lots of time to optimize

## Language implementation

- **Interpreters:** Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*

## Language implementation

- **Interpreters:** Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*

1. 2 months, Joel, Dragon
2. For intermediate and advanced, there are also much better books

## Language implementation

- **Interpreters:** Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*

### NOT THE DRAGON BOOK

**Engineering a Compiler** by Cooper/Torczon

**Modern Compiler Implementation in X** by Appel

## Language implementation

- **Interpreters:** Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*

### NOT THE DRAGON BOOK

**Engineering a Compiler** by Cooper/Torczon

**Modern Compiler Implementation in X** by Appel

1. Amazing optimizations; Hotspot; dispatch, exceptions and arithmetic
2. Shockingly difficult to write

## Language implementation

- **Interpreters:** Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*
- **Just-in-time compilers:** Very difficult, unportable, *fast interpreter*.

## Language implementation

- **Interpreters:** Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*
- **Just-in-time compilers:** Very difficult, unportable, *fast interpreter*.

What's right with scripting languages?

1. dont dwell here -  
I'm not here to  
convince you
2. the bad things dont  
touch these

What's right with scripting languages?

1. dont dwell here -  
I'm not here to  
convince you
2. the bad things dont  
touch these
3. Python, Ruby; JS  
too in a certain way

What's right with scripting languages?

1. Elegant and well designed,

What's right with scripting languages?

1. Elegant and well designed,

1. dont dwell here -  
I'm not here to  
convince you
2. the bad things dont  
touch these
3. get things done, as  
Joel would say

## What's right with scripting languages?

- ① Elegant and well designed,
- ② High level of abstraction,

## What's right with scripting languages?

- ① Elegant and well designed,
- ② High level of abstraction,

1. dont dwell here -  
I'm not here to  
convince you
2. the bad things dont  
touch these
3. avoids many  
problems inherent  
in Java, C# and  
C++: verbosity,  
type systems

## What's right with scripting languages?

- ❶ Elegant and well designed,
- ❷ High level of abstraction,
- ❸ Dynamic typing (and duck typing).

## What's right with scripting languages?

- ❶ Elegant and well designed,
- ❷ High level of abstraction,
- ❸ Dynamic typing (and duck typing).

## What's wrong with scripting languages?

**Symptoms:** Speed, Portability

1. portable to .Net, JVM, etc?
2. not fast means you cant use it as much

## What's wrong with scripting languages?

**Symptoms:** Speed, Portability



1. portable to .Net, JVM, etc?
2. not fast means you cant use it as much

## What's wrong with scripting languages?

**Symptoms:** Speed, Portability

**Problem:** Language designed for interpreters

- Run-time source code execution

## What's wrong with scripting languages?

**Symptoms:** Speed, Portability

**Problem:** Language designed for interpreters

- Run-time source code execution

1. portable to .Net, JVM, etc?
2. not fast means you cant use it as much

## What's wrong with scripting languages?

**Symptoms:** Speed, Portability

**Problem:** Language designed for **one specific** interpreter

- Run-time source code execution
- Foreign Function Interface

## What's wrong with scripting languages?

**Symptoms:** Speed, Portability

**Problem:** Language designed for **one specific** interpreter

- Run-time source code execution
- Foreign Function Interface

1. define FFI
2. glue code in C; wrap things in python data structures; expose interpreter internals
3. Works for all except JS - I'm using Python as example

## FFI

**Foreign Function Interface** based on CPython interpreter

- Access to C libraries
- Script C applications using Python scripts
- Rewrite hot code in C

## FFI

## FFI

**Foreign Function Interface** based on CPython interpreter

- Access to C libraries
- Script C applications using Python scripts
- Rewrite hot code in C

## FFI (good) implications

- Libraries not that slow
- Can break out of Python for slow code.

## FFI (good) implications

- Libraries not that slow
- Can break out of Python for slow code.

1. nice things about python are lost (high level, elegant

## FFI (bad) implications

- Language is allowed to be slow
- Must break out of Python for speed.

## FFI (bad) implications

- Language is allowed to be slow
- Must break out of Python for speed.

1. you cant change the interpreter much - it cant get faster

## FFI (worse) implications

- Legacy issues

## FFI (worse) implications

- Legacy issues

1. Jython, IronPython, PyPy, cant use the same code
2. cant even reimplement own language
3. by constrast, look at JS

## FFI (worse) implications

- Legacy issues
- Reimplementations

## FFI (worse) implications

- Legacy issues
- Reimplementations

1. rule for life, really
2. like pyrex, ctypes, etc
3. no implementation specific code at all
4. import functions directly, and access them from within Python without a line of C
5. Ruby libFFI

## FFI solution

### **Don't expose yourself!**

- Importing functions into Python with a Domain Specific Language is good

## FFI solution

### **Don't expose yourself!**

- Importing functions into Python with a Domain Specific Language is good



1. rule for life, really
2. like pyrex, ctypes, etc
3. for reimplementation
4. no implementation specific code at all
5. import functions directly, and access them from within Python without a line of C
6. Ruby libFFI

## FFI solution

### Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better

## FFI solution

### Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better

1. rule for life, really
2. like pyrex, ctypes, etc
3. no implementation specific code at all
4. import functions directly, and access them from within Python without a line of C
5. Ruby libFFI

## FFI solution

### Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better
- Declarative is best

## FFI solution

### Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better
- Declarative is best

1. rule for life, really
2. like pyrex, ctypes, etc
3. no implementation specific code at all
4. import functions directly, and access them from within Python without a line of C
5. Ruby libFFI

## FFI solution

### Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better
- Declarative is best
  
- Any reimplementaion can reuse the same libraries without any modifications
- CPython itself can change without hassle

## FFI solution

### Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better
- Declarative is best
  
- Any reimplementaion can reuse the same libraries without any modifications
- CPython itself can change without hassle

1. Should be used for source inclusion. But because they're interpreted, they can be used for:

## Dynamic source code generation

- `eval` and `dynamic include/import`

## Dynamic source code generation

- `eval` and `dynamic include/import`

## Dynamic source code generation

1. Should be used for source inclusion. But because they're interpreted, they can be used for:

### Dynamic source code generation

- `eval` and `dynamic include/import`
  - meta-programming

```
eval (mysql_read (...) [0]);
```

- `eval` and `dynamic include/import`
  - meta-programming

```
eval (mysql_read (...) [0]);
```

## Dynamic source code generation

1. Should be used for source inclusion. But because they're interpreted, they can be used for:

- `eval` and `dynamic include/import`
    - meta-programming
    - `.rc` files
- ```
username = "myname"  
password = "mypass"  
server = "srv.domain.com"
```

## Dynamic source code generation

- `eval` and `dynamic include/import`
  - meta-programming
  - `.rc` files

```
username = "myname"  
password = "mypass"  
server = "srv.domain.com"
```

## Dynamic source code generation

1. Should be used for source inclusion. But because they're interpreted, they can be used for:

- `eval` and dynamic `include/import`
    - meta-programming
    - `.rc` files
    - localization
- ```
$lang = ....;  
include ("localisation/locale.$lang.php");
```

## Dynamic source code generation

- `eval` and dynamic `include/import`
  - meta-programming
  - `.rc` files
  - localization

```
$lang = ....;  
include ("localisation/locale.$lang.php");
```

1. go to next slide straight away

Dynamic source code generation

We don't even know the full program source!!

Dynamic source code generation

We don't even know the full program source!!



1. self-perpetuating cycle
2. shame, scripting languages could really use static analysis
3. high-level means high level optimizations? no
4. redundancy elimination - some cant be done by hand

## So they can't be compiled (ahead-of-time)

### Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

## So they can't be compiled (ahead-of-time)

### Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

1. self-perpetuating cycle
2. shame, scripting languages could really use static analysis
3. high-level means high level optimizations? no
4. redundancy elimination - some cant be done by hand

## So they can't be compiled (ahead-of-time)

Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
t = ...;
for (i = 0; i < strlen(t); i++)
{
    s[i] = t[i];
}
```

## So they can't be compiled (ahead-of-time)

Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
t = ...;
for (i = 0; i < strlen(t); i++)
{
    s[i] = t[i];
}
```

1. self-perpetuating cycle
2. shame, scripting languages could really use static analysis
3. high-level means high level optimizations? no
4. redundancy elimination - some cant be done by hand

## So they can't be compiled (ahead-of-time)

Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
t = ...;
_temp = strlen(t);
for (i = 0; i < _temp; i++)
{
    s[i] = t[i];
}
```

## So they can't be compiled (ahead-of-time)

Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
t = ...;
_temp = strlen(t);
for (i = 0; i < _temp; i++)
{
    s[i] = t[i];
}
```

1. self-perpetuating cycle
2. shame, scripting languages could really use static analysis
3. high-level means high level optimizations? no
4. redundancy elimination - some cant be done by hand

## So they can't be compiled (ahead-of-time)

### Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
alert ($('#li').get(0).nodeName);
```

## So they can't be compiled (ahead-of-time)

### Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
alert ($('#li').get(0).nodeName);
```

1. self-perpetuating cycle
2. shame, scripting languages could really use static analysis
3. high-level means high level optimizations? no
4. redundancy elimination - some cant be done by hand

## So they can't be compiled (ahead-of-time)

### Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
alert ($('#li')[0].nodeName);
```

## So they can't be compiled (ahead-of-time)

### Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
alert ($('#li')[0].nodeName);
```

1. instead, they must be JIT compiled. We'll still never get cool optimizations
2. worse, the research just gets us, at run-time, expensively, information we could get at compile-time.

## JIT compiled

Tracemonkey

<http://hacks.mozilla.org/2009/07/tracemonkey-overview/>

## JIT compiled

Tracemonkey

<http://hacks.mozilla.org/2009/07/tracemonkey-overview/>

1. instead, they must be JIT compiled. We'll still never get cool optimizations
2. worse, the research just gets us, at run-time, expensively, information we could get at compile-time.

## JIT compiled

### Tracemonkey

<http://hacks.mozilla.org/2009/07/tracemonkey-overview/>

### Type Analysis for Javascript

Simon Holm Jensen, Anders Møller and Peter Thiemann  
SAS '09  
<http://www.brics.dk/TAJS/>

## JIT compiled

### Tracemonkey

<http://hacks.mozilla.org/2009/07/tracemonkey-overview/>

### Type Analysis for Javascript

Simon Holm Jensen, Anders Møller and Peter Thiemann  
SAS '09  
<http://www.brics.dk/TAJS/>

1. do it like Perl, or C++

Fix at **language** design time

- No dynamic `include`; no `eval`.
  - Compile-time meta-programming

Fix at **language** design time

- No dynamic `include`; no `eval`.
  - Compile-time meta-programming



## 1. sandbox them

### Fix at **language** design time

- No dynamic `include`; no `eval`.
  - Compile-time meta-programming
  - `.rc` files

### Fix at **language** design time

- No dynamic `include`; no `eval`.
  - Compile-time meta-programming
  - `.rc` files

1. With a compiled model, we know all the files

## Fix at **language** design time

- No dynamic `include`; no `eval`.
  - Compile-time meta-programming
  - `.rc` files
  - localization

## Fix at **language** design time

- No dynamic `include`; no `eval`.
  - Compile-time meta-programming
  - `.rc` files
  - localization

1. unfortunately  
chose a paradigm  
that nobody knows:  
stack-based;  
whoops!

## Doing it right

- Factor
  - compiled model
  - compile-time meta-programming
  - declarative FFI

## Doing it right

- Factor
  - compiled model
  - compile-time meta-programming
  - declarative FFI

1. this is the shit we should be researching, instead of finding ways around intracable problems
2. javascript 10px
3. table based
4. should be trivial

## Open research problems

- Optimizing *boxing*
- High-level optimizations
- Combining ahead-of-time and JIT compilation

## Open research problems

- Optimizing *boxing*
- High-level optimizations
- Combining ahead-of-time and JIT compilation

Conclusion

Design the next scripting  
language right

Compiled and interpreted models

Conclusion

Design the next scripting  
language right