

# **"Rakud'oh!"**

## **Making our compiler smarter**



**Jonathan  
Worthington**

**Rakudo \***

# Rakudo \*

**Released last summer**

# Rakudo \*

**Plenty to be happy about...**





**Gave people a feel for the power and beauty of Perl 6**

**Chained Comparisons Junctions**  
**Classes Signatures Grammars**  
**Perl 6 Regexes Multi-dispatch**  
**Lazy Lists Series Operator**  
**Roles Introspection Traits**  
**Meta-Operators Feeds MAIN**  
**Smart-matching Modules**  
**Native Library Calls Book**

**Covered A Wide Range of  
Perl 6 Features**





**Led to some weird areas of the spec getting worked out**





**Release attracted more people  
to the Perl 6 community**



# Rakudo \*

**...but plenty of weaknesses**



**Most things run slowly.  
Some run glacially slowly.**





**High memory usage - both base amount and when running**





**Various unhelpful errors and failure modes**





**Weak in areas of language  
extensibility**

**Make it work**

**THEN**

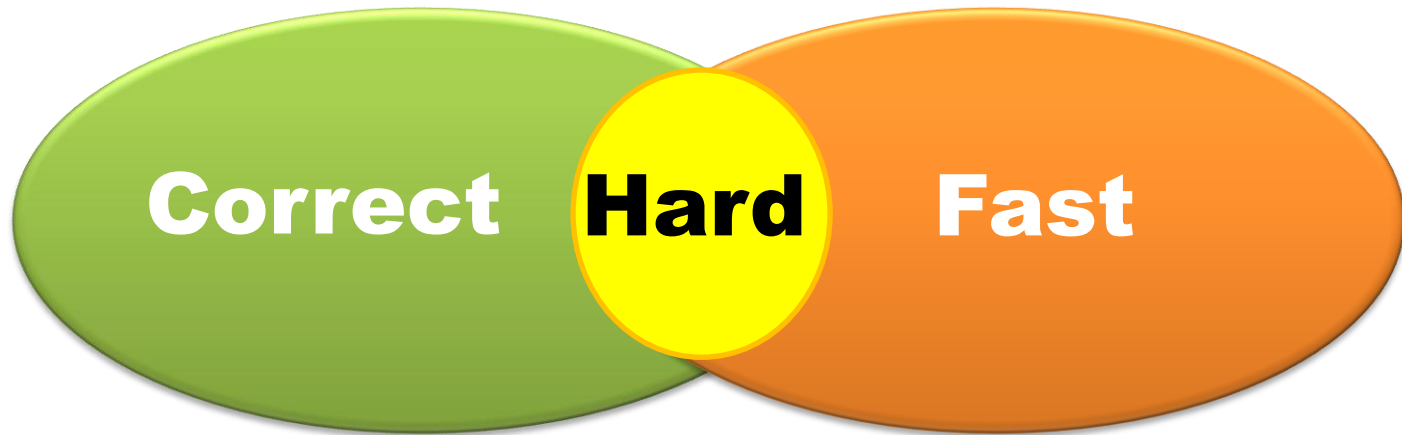
**Make it fast**



**Many things needed  
a few tries to get  
them correct.**

**(A few things needed many  
tries to get them correct. 😊)**

**The quick way to implement  
a feature with the correct  
semantics is very rarely  
the optimal one.**



**Didn't want to waste time  
making the wrong thing fast.**

**Now the development  
focus is changing.**

**Many implemented features  
now relatively stable.**

**Missing features aren't our  
main adoption blocker, but  
speed and memory usage are.**





**Some things in Rakudo today  
just make you say "D'oh!"**

# **Need to make Rakudo** **smarter**

**More analysis on the code  
that is being compiled**

**More awareness of the  
context of the compilation**

# The traditional view

**Compile  
Time**



**Run  
Time**

---

# The Perlsh view

**Compile  
Time**



**Run  
Time**



**Some of the**



**D'oh!**

**we're working on  
fixing now**

# Boxed constants created every usage

```
$x = 42;
```

**Should construct them once at  
compile time and stash them  
away in a constants table**

**Speed win: less time making  
objects, less GC churn**

# Overweight Types

**How many GC-able objects  
should a boxed integer be?**

# Overweight Types

**How many GC-able objects  
should a boxed integer be?**

**Today, **three**...**





# Overweight Types

**How many GC-able objects should a boxed integer be?**

**Today, **three**...**



**Should, of course, be **one**.**  
**It will be. Less memory, less to allocate, less GC churn.**

# Slow Type Checks

**Today, doing a type check tends to involve a method call and/or a bunch of named lookups**

**High cost for a relatively common operation**

**In new OO implementation, usually just a few pointer comparisons**

# Attribute Access

**Anyone spot the typo?**

```
class Golf {  
    has $!player;  
    has $!tee;  
    method play() {  
        $!player.goto($!pee) ;  
    }  
}
```



# Attribute Access

**Anyone spot the typo?**

```
class Golf {  
  has $!player;  
  has $!tee;  
  method play() {  
    $!player.goto($!pee) ;  
  }  
}
```

**Should detect and report at  
compile time, not runtime.**

(Even for custom meta-objects.)

# Attribute Access

**Often, we can map attributes to slot indexes at compile time.**

```
...  
    has $!player;    0  
    has $!tee;       1  
...                  ...
```

**Then attribute access will be mostly pointer follows, rather than needing to do hash lookups**

# Too Many Allocations

**Various common operations currently end up allocating an object as they do their work**

**Multi-dispatch cache lookups  
Method lookups**

**They will stop doing so. Making them faster, and less GC churn.**



# Type info not used

**Will this code ever work?**

```
my Int $x = 4.2;
```

# Type info not used

**Will this code ever work?**

```
my Int $x = 4.2;
```

**No, so complain about it at  
compile time!**

**Today that's harder to implement than it  
should be. Soon we'll have the  
infrastructure to do so.**

# Type info not used

**Which multi-dispatch candidate  
will be called here?**

```
multi foo(Int $x) { 1 }  
multi foo(Num $x) { 2 }  
my Int $x = something();  
foo($x);
```

**Can decide at compile time!**  
**Important since all built-in  
operators are multi-dispatch**  
(So it's a pre-requisite for inlining.)


# Type info not used

**The new method is defined in the top type; all objects have one**

```
my $dinner = борщ.new();
```


**Construct a v-table and dispatch such method calls by index**

callmeth "new"



ACCEPTS	CODE
Bool	CODE
new	CODE
...	...

callmeth 2



CODE
CODE
CODE
...



# Type info not used

**The more type information we have in a program, the more calls we will be able to optimize**


**Which is how gradual typing is supposed to work 😊**

**(BTW, the same approach can be used to implement a pragma that warns about calling unknown methods.)**

# Dumb lexical access

**We statically know where to find a lexical variable - but today we walk scopes looking for names**

```
my $kitten-mass = 0;  
for @cats -> $cat {  
    if $cat.is_kitten {  
        $kitten-mass += $cat.mass;  
    }  
}
```



**"2 scopes down, in slot 0"**

# Leaky Extensions

**Currently, language tweaks  
always end up global**

```
{  
  sub postfix:<!>($n) { [*] 1..$n }  
  say 10!;  
}  
say 5!;
```

**Should actually be lexically  
scoped (so the 5! fails to parse)**

# Leaky Extensions

**While this was a contrived example, knowing exactly which language we're parsing is an important part of keeping Perl 6 sanely extensible.**

**The same set of changes should open the door to implementing macros too.**



**And the list goes on...**

**Lots we can do to make  
common things faster.**

**Optimize the building  
blocks that all programs  
are made from.**

**Later, still plenty of clever  
optimizations to explore. 😊**

**Спасибо!**

# Questions?

**Blog: [6guts.wordpress.com](http://6guts.wordpress.com)**

**Twitter: [jnthnwrthngtn](https://twitter.com/jnthnwrthngtn)**

**Slides: [jnthn.net/articles.shtml](http://jnthn.net/articles.shtml)**