Taking Rakudo Forward: What I'm Hacking On

Jonathan Worthington

My previous talk:

Perl 6 from a user's perspective

This talk:

Perl 6 from an implementer's perspective

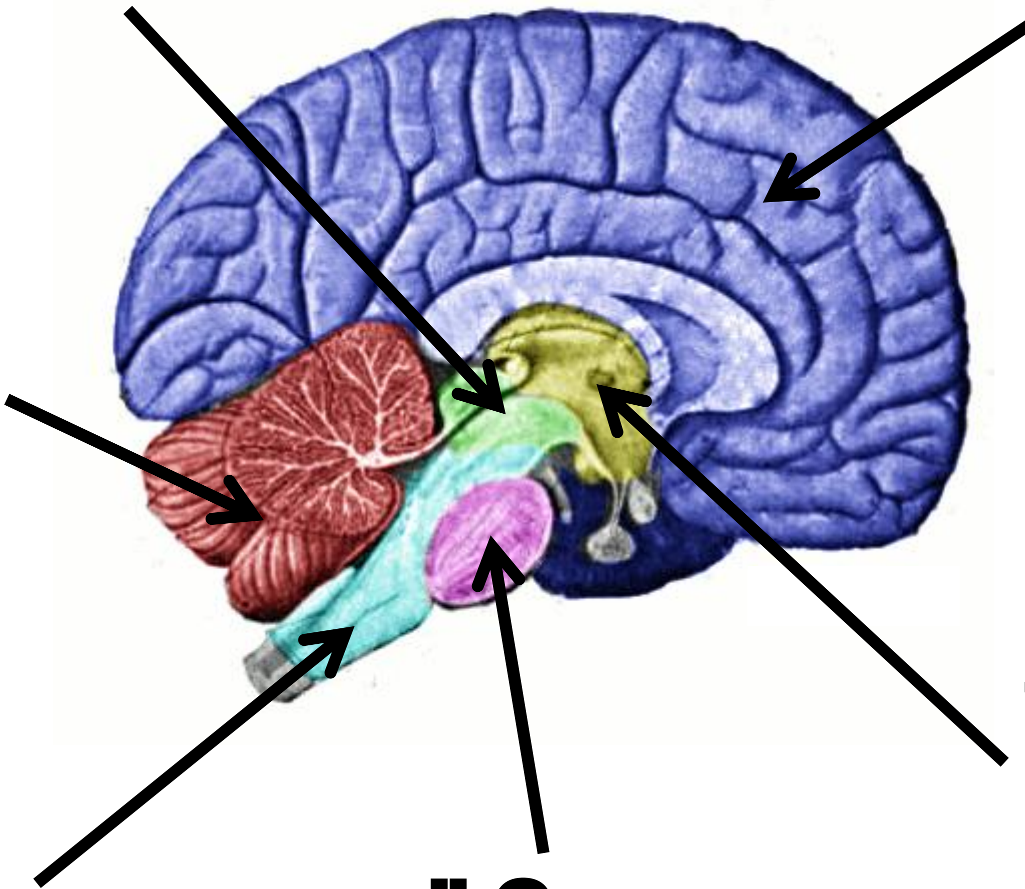
Taking Rakudo Forward: What I'm Hacking On

or

This talk:

A peek inside my brain

Taking Rakudo Forward: What I'm Hacking On



Bad puns

Perl 6 stuff

"I want a steak"

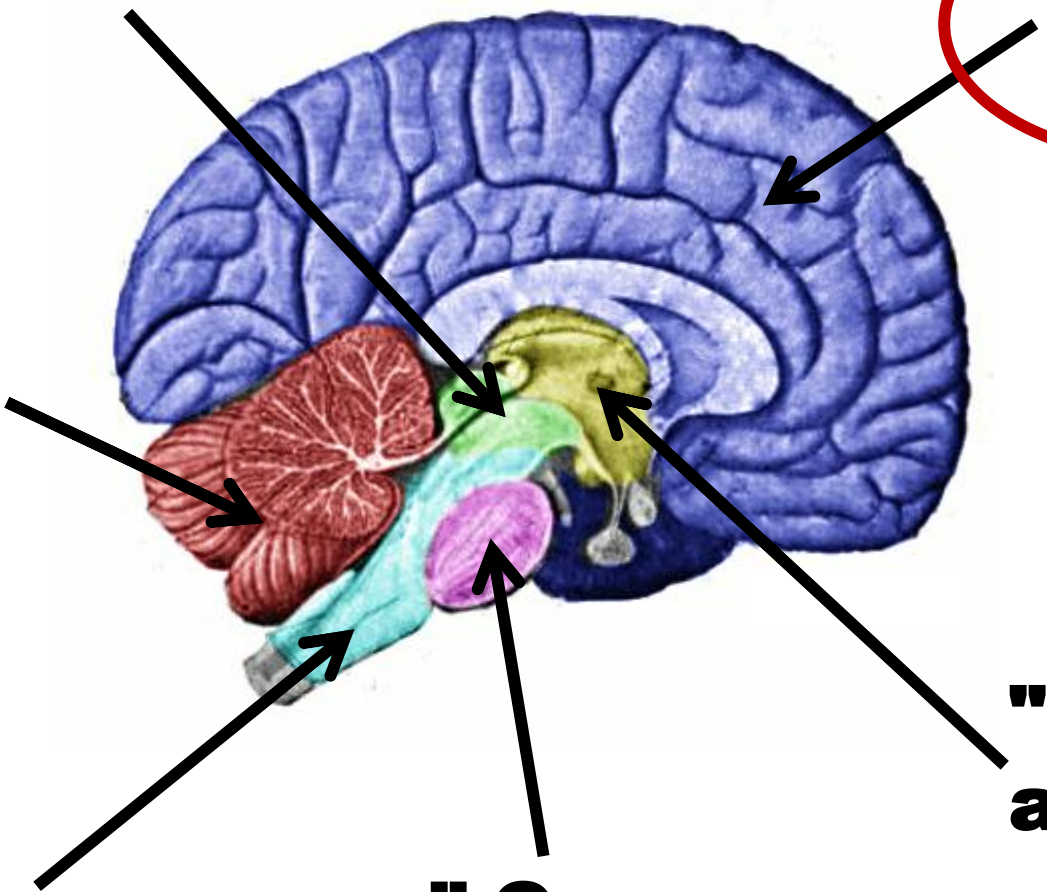
"I want a beer"

I/O

" Она красива! "

Image Attribution: WikiPedia

Taking Rakudo Forward: What I'm Hacking On



Bad puns

Perl 6 stuff

only this part

"I want a steak"

"I want a beer"

I/O

" Она красива! "

Image Attribution: WikiPedia

Rakudo Development Philosophy

Start off by achieving wide feature coverage but low feature "depth"

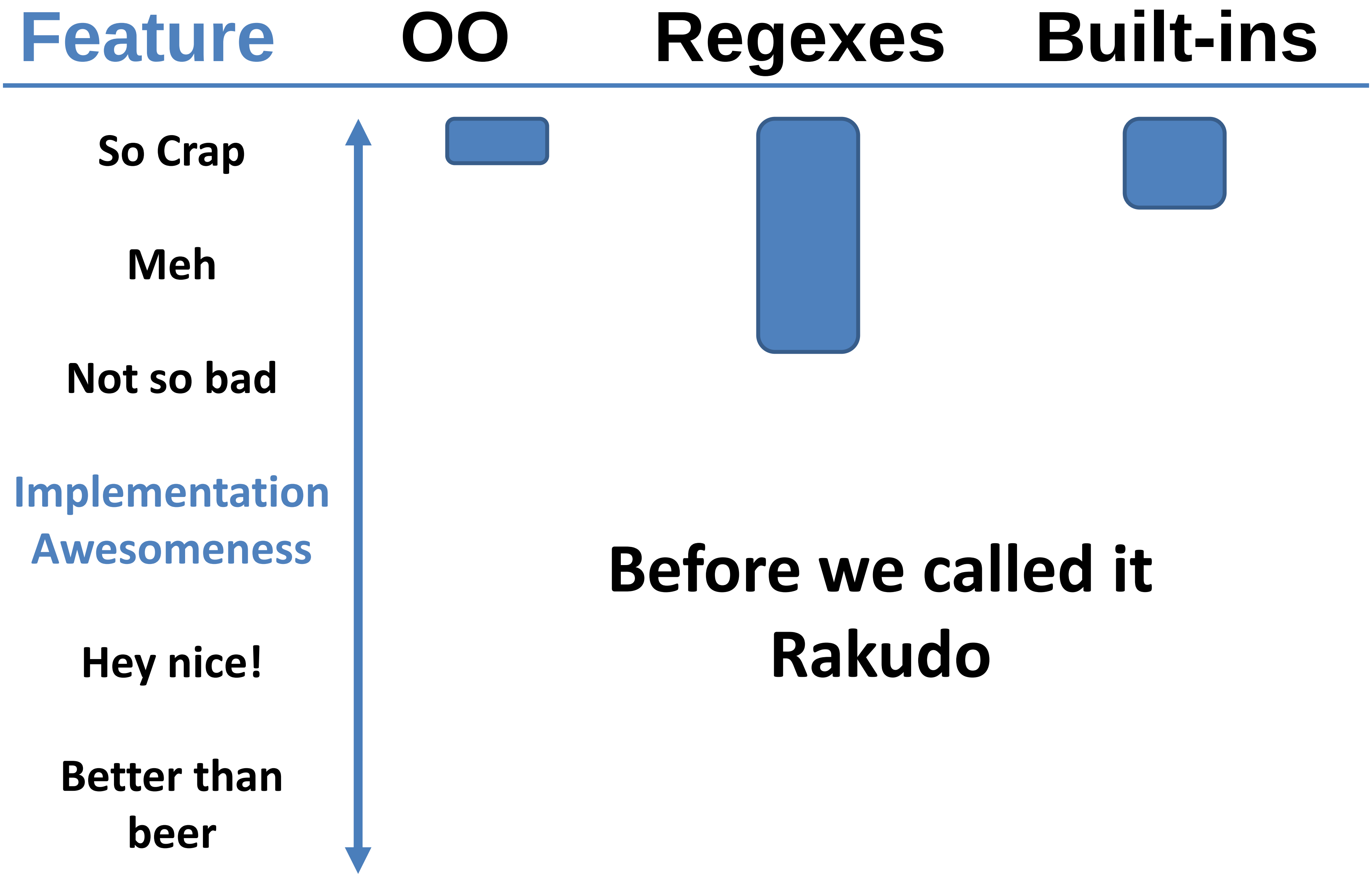
Taking Rakudo Forward: What I'm Hacking On

Feature OO Regexes Built-ins

Taking Rakudo Forward: What I'm Hacking On

Feature	OO	Regexes	Built-ins
So Crap			
Meh			
Not so bad			
Implementation Awesomeness			
Hey nice!			
Better than beer			

Taking Rakudo Forward: What I'm Hacking On



Feature | OO | Regexes | Built-ins

So Crap

Meh

Not so bad

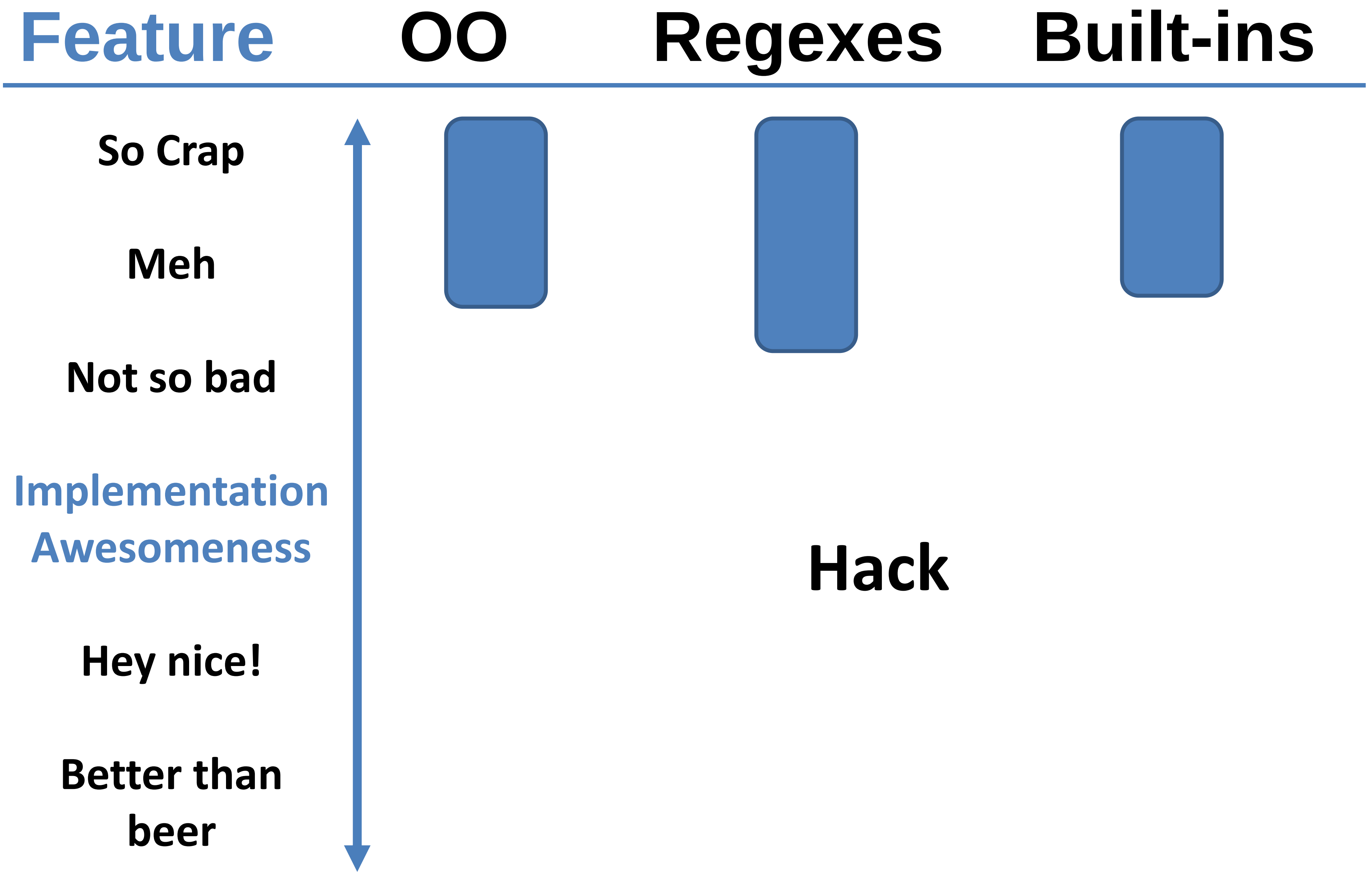
Implementation Awesomeness

Hey nice!

Better than beer

Before we called it Rakudo

Taking Rakudo Forward: What I'm Hacking On



Feature

OO

Regexes

Built-ins

So Crap

Meh

Not so bad

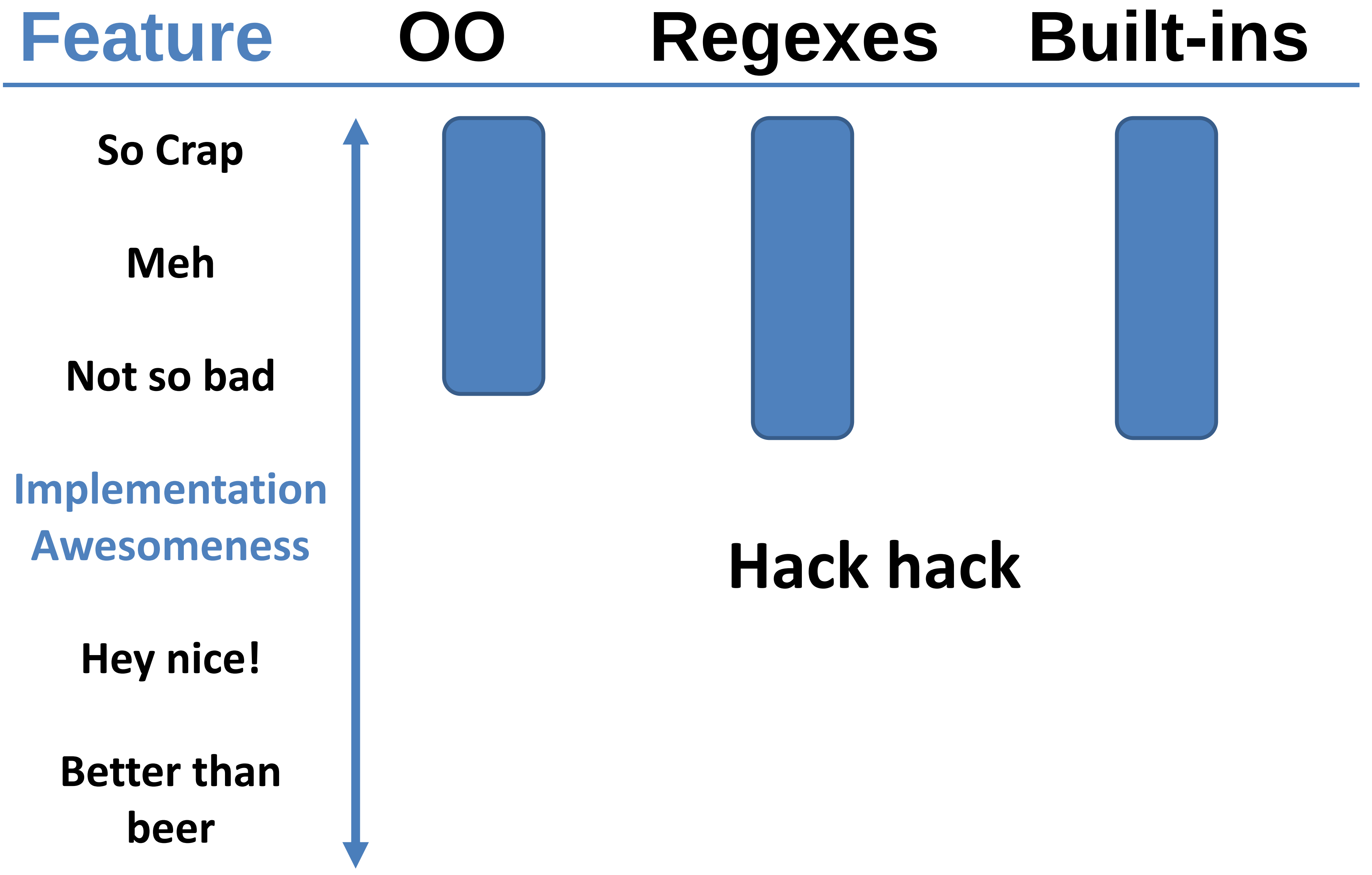
Implementation Awesomeness

Hey nice!

Better than beer

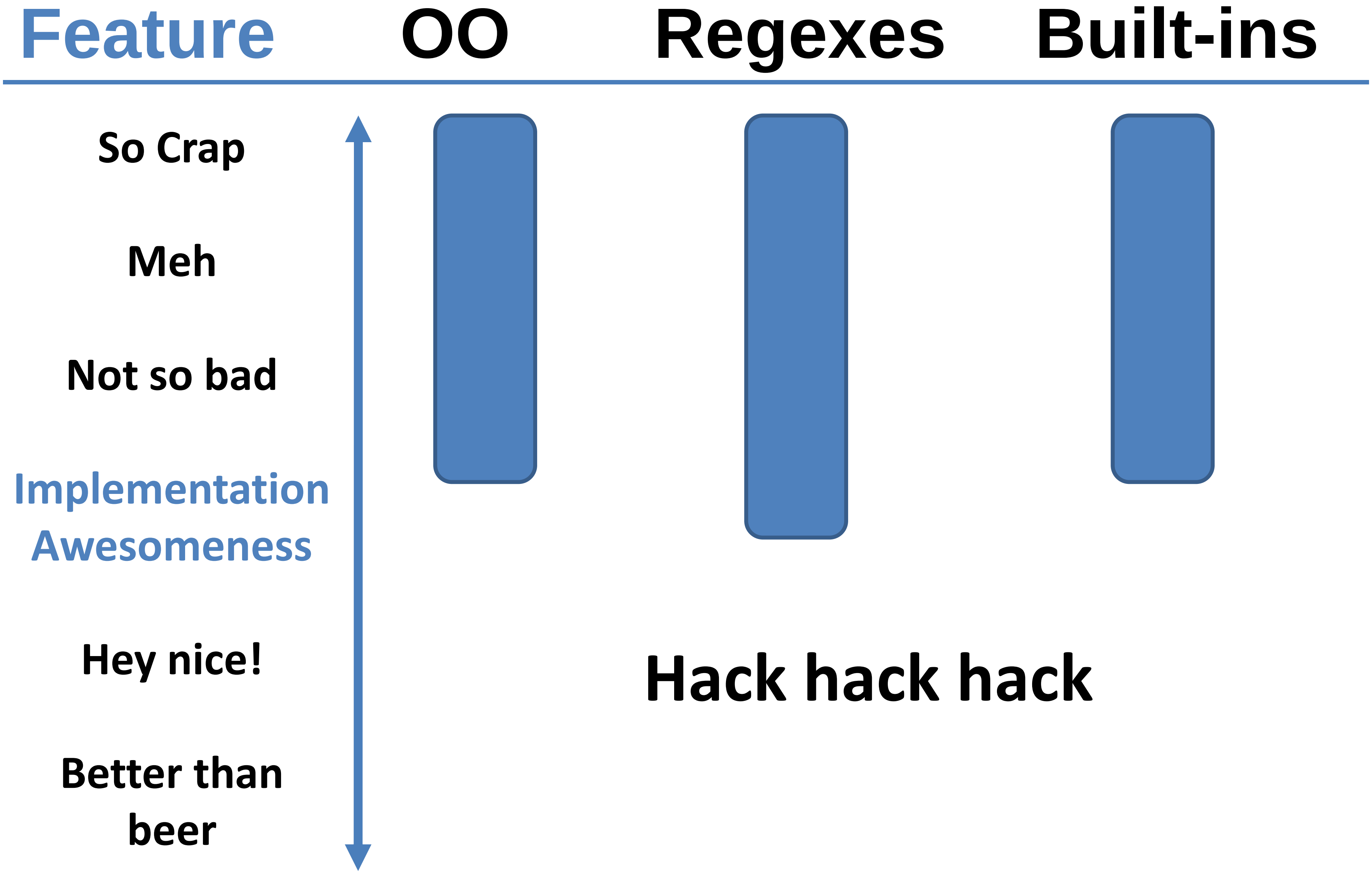
Hack

Taking Rakudo Forward: What I'm Hacking On



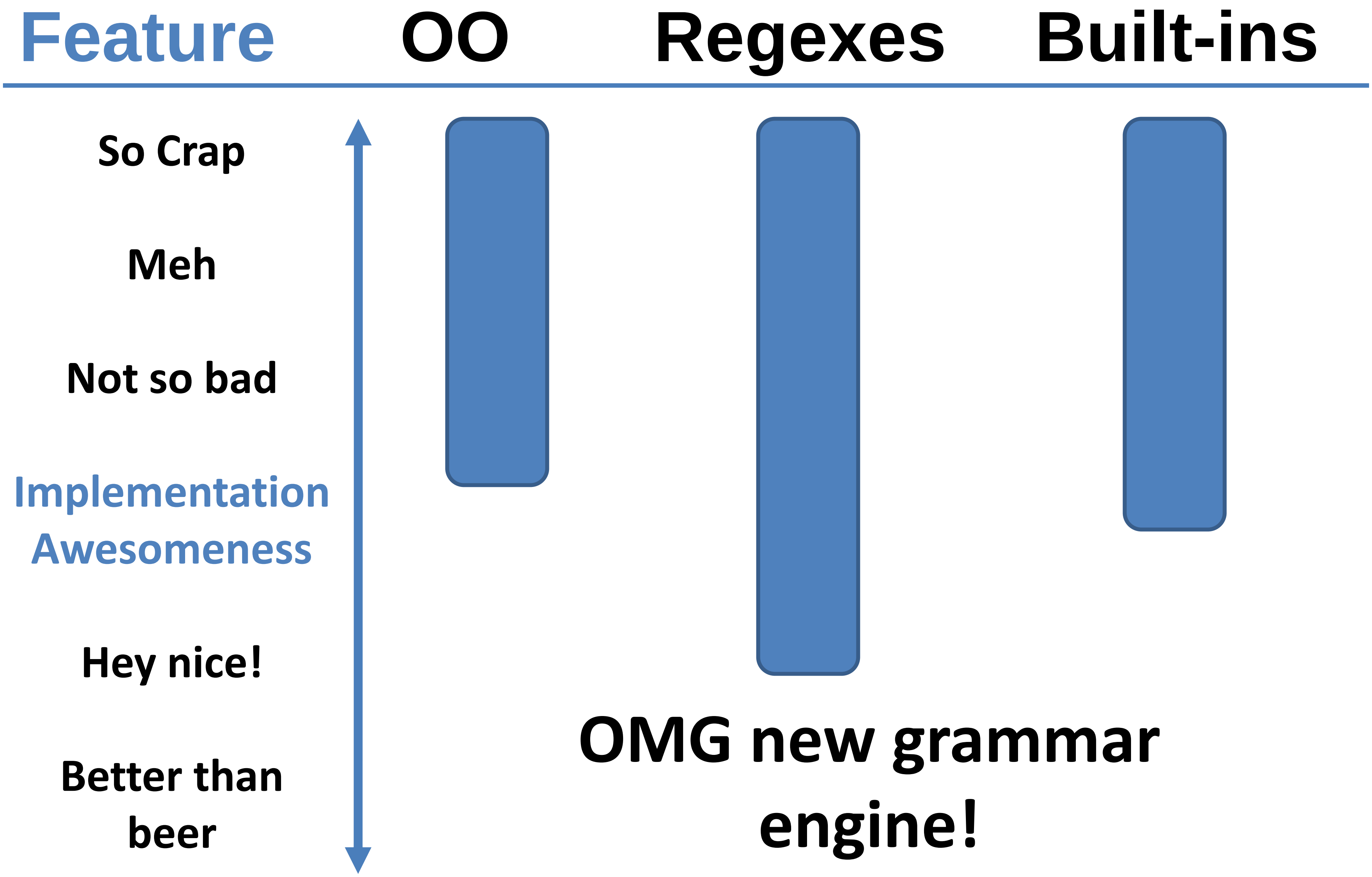
Hack hack

Taking Rakudo Forward: What I'm Hacking On



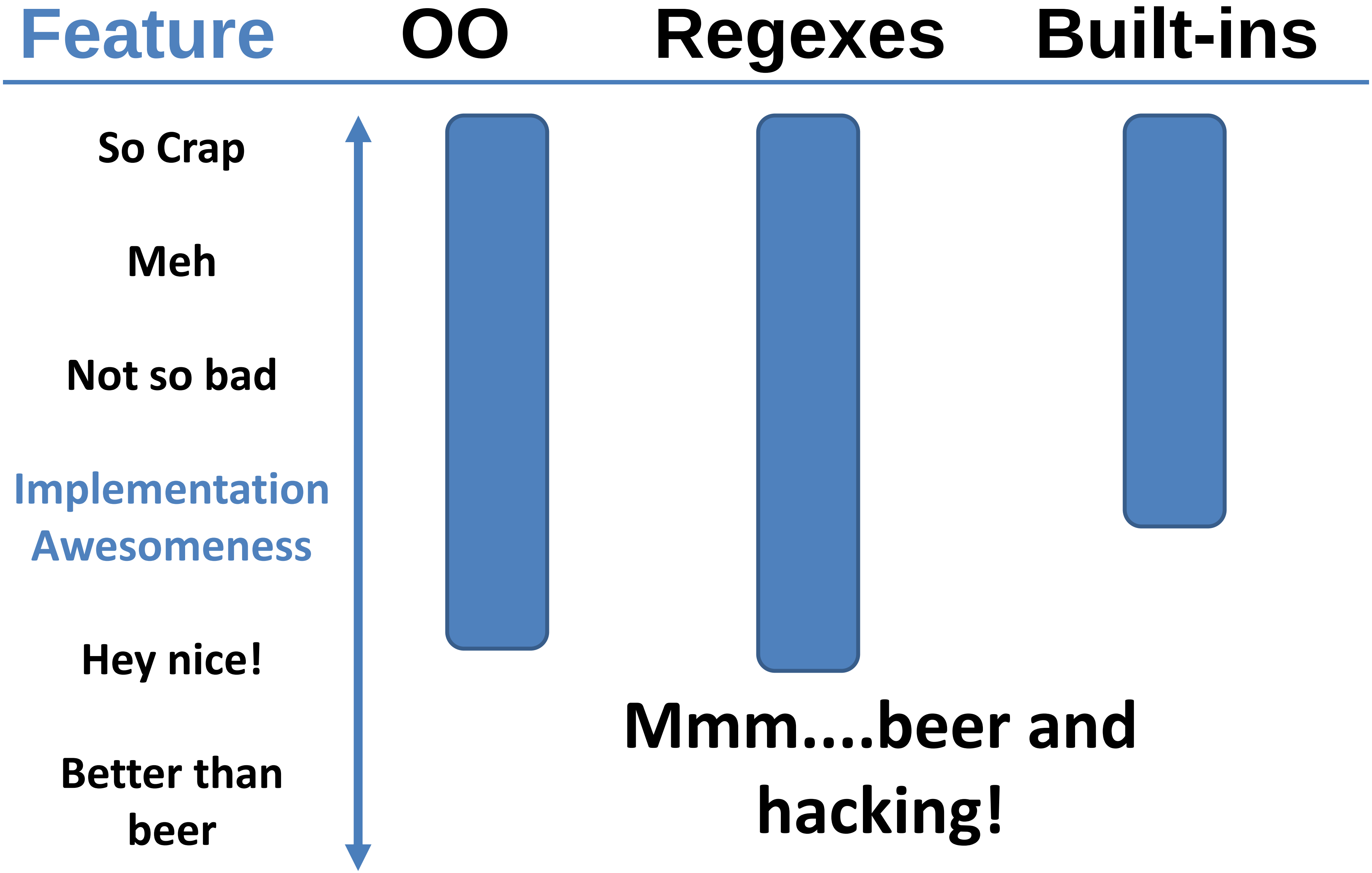
Hack hack hack

Taking Rakudo Forward: What I'm Hacking On



OMG new grammar engine!

Taking Rakudo Forward: What I'm Hacking On



Mmm....beer and hacking!

Taking Rakudo Forward: What I'm Hacking On

Try to get something usable into user's hands earlier rather than later

Taking Rakudo Forward: What I'm Hacking On

Users

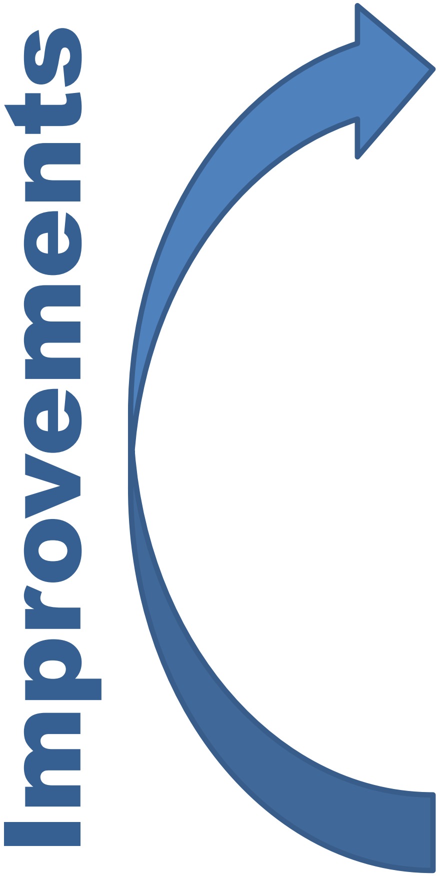
=

Community

=

Feedback

Taking Rakudo Forward: What I'm Hacking On



Users

=

Community

=

Feedback

Rakudo *

Useful, usable release aimed at early adopters

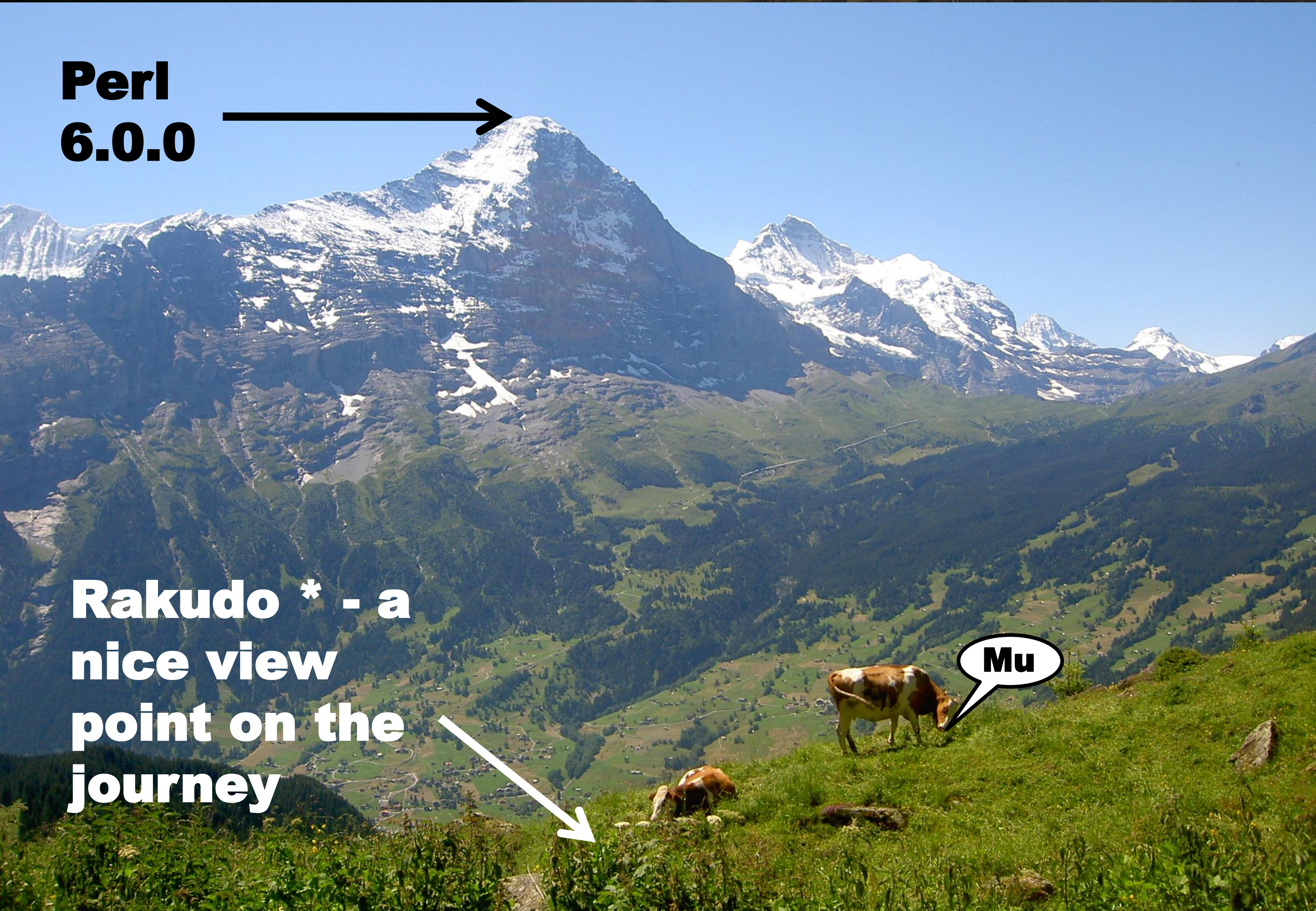
Lots of nice features ☺

Various issues ☹

Taking Rakudo Forward: What I'm Hacking On

Rakudo * - a nice view point on the journey

Taking Rakudo Forward: What I'm Hacking On



Perl 6.0.0

Rakudo * - a nice view point on the journey

Focus of today's talk:

The work I'm doing to help us complete the next big part of the climb

Introducing Meta-models: A Story

Once upon a time, I wrote a class.

```
class Lolcat is Cat {
    has $.caption;
    has $!lol-factor;
    method lol() {
        say($!lol-factor < 0   ?? 'wtf' !!
            $!lol-factor < 42  ?? 'lol' !!
                                  'rofl');
        }
    }
}
```

Taking Rakudo Forward: What I'm Hacking On

I thought my work was done, and I could go for a beer.

Taking Rakudo Forward: What I'm Hacking On

But then my class started asking me questions…

Taking Rakudo Forward: What I'm Hacking On

How was
I created?

```
class Lolcat is Cat {
    has $.caption;
    has $!lol-factor;
    method lol() {
        say($!lol-factor < 0   ?? 'wtf' !!
            $!lol-factor < 42  ?? 'lol' !!
                                  'rofl');
        }
    }
}
```

Taking Rakudo Forward: What I'm Hacking On

What does it mean to have methods?

```
class Lolcat is Cat {
    has $.caption;
    has $!lol-factor;
    method lol() {
        say($!lol-factor < 0    ?? 'wtf' !!
            $!lol-factor < 42   ?? 'lol' !!
                                  'rofl');
        }
    }
}
```

Taking Rakudo Forward: What I'm Hacking On

What does it mean to inherit?

```
class Lolcat is Cat {
    has $.caption;
    has $!lol-factor;
    method lol() {
        say($!lol-factor < 0   ?? 'wtf' !!
            $!lol-factor < 42  ?? 'lol' !!
                                  'rofl');
        }
    }
}
```

Taking Rakudo Forward: What I'm Hacking On

Do other classes all behave like me?

```
class Lolcat is Cat {
    has $.caption;
    has $!lol-factor;
    method lol() {
        say($!lol-factor < 0   ?? 'wtf' !!
            $!lol-factor < 42  ?? 'lol' !!
                                  'rofl');
        }
    }
}
```

Taking Rakudo Forward: What I'm Hacking On

What about prototype OO?

```
class Lolcat is Cat {
    has $.caption;
    has $!lol-factor;
    method lol() {
        say($!lol-factor < 0   ?? 'wtf' !!
            $!lol-factor < 42  ?? 'lol' !!
                                  'rofl');
        }
    }
}
```

Classes in Perl 6 are just one type of package.

We also have grammars and roles.

Taking Rakudo Forward: What I'm Hacking On

STD.pm

```
…
token package_declarator:class {
    :my $*PKGDECL := 'class';
    <sym> <package_def>
}
token package_declarator:grammar {
    :my $*PKGDECL := 'grammar';
    <sym> <package_def>
}
token package_declarator:role {
    :my $*PKGDECL := 'role';
    <sym> <package_def>
}
…
```

Taking Rakudo Forward: What I'm Hacking On

STD.pm

```
…
token package_declarator:class {
    :my $*PKGDECL := 'class';
    <sym> <package_def>
}
token package_declarator:grammar {
    :my $*PKGDECL := 'grammar';
    <sym> <package_def>
}
token package_declarator:role {
    :my $*PKGDECL := 'role';
    <sym> <package_def>
}
…
```

All have methods, attributes, semantics for inheritance and composition, etc.

Many more commonalities than differences.

Could bake the details deep in the implementation.

Not hackable, not extensible...and thus not Perl 6-like.

Taking Rakudo Forward: What I'm Hacking On

Idea!

Define an API and implement it for each type of package.

OO API

Make the API actually be a set of methods on an object

Different type of package = different type of object

Tweak an existing package type by subclassing

Taking Rakudo Forward: What I'm Hacking On

Implement the object model in terms of objects.

Extend the object model in terms of objects.

Meta-object

An object that specifies how some other object works

Meta-object Protocol

The set of methods that we implement in a meta-object

Taking Rakudo Forward: What I'm Hacking On

```
::LolCat := ClassHOW.new_type(name => 'LolCat');

LolCat.^add_parent(Cat);

LolCat.^add_attribute(Attribute.new(
    name => '$!caption', has_accessor => True
));
LolCat.^add_attribute(Attribute.new(
    name => '$!lol-factor'
));

LolCat.^add_method('lol', method () {
    ...
});

LolCat.^compose();
```

The 6model Project

Today's object implementation in Rakudo builds a layer on top of the Parrot built-in object model.

Taking Rakudo Forward: What I'm Hacking On

Allowed us to get to something that works well enough quickly

but

We've hit limits of this approach

Taking Rakudo Forward: What I'm Hacking On

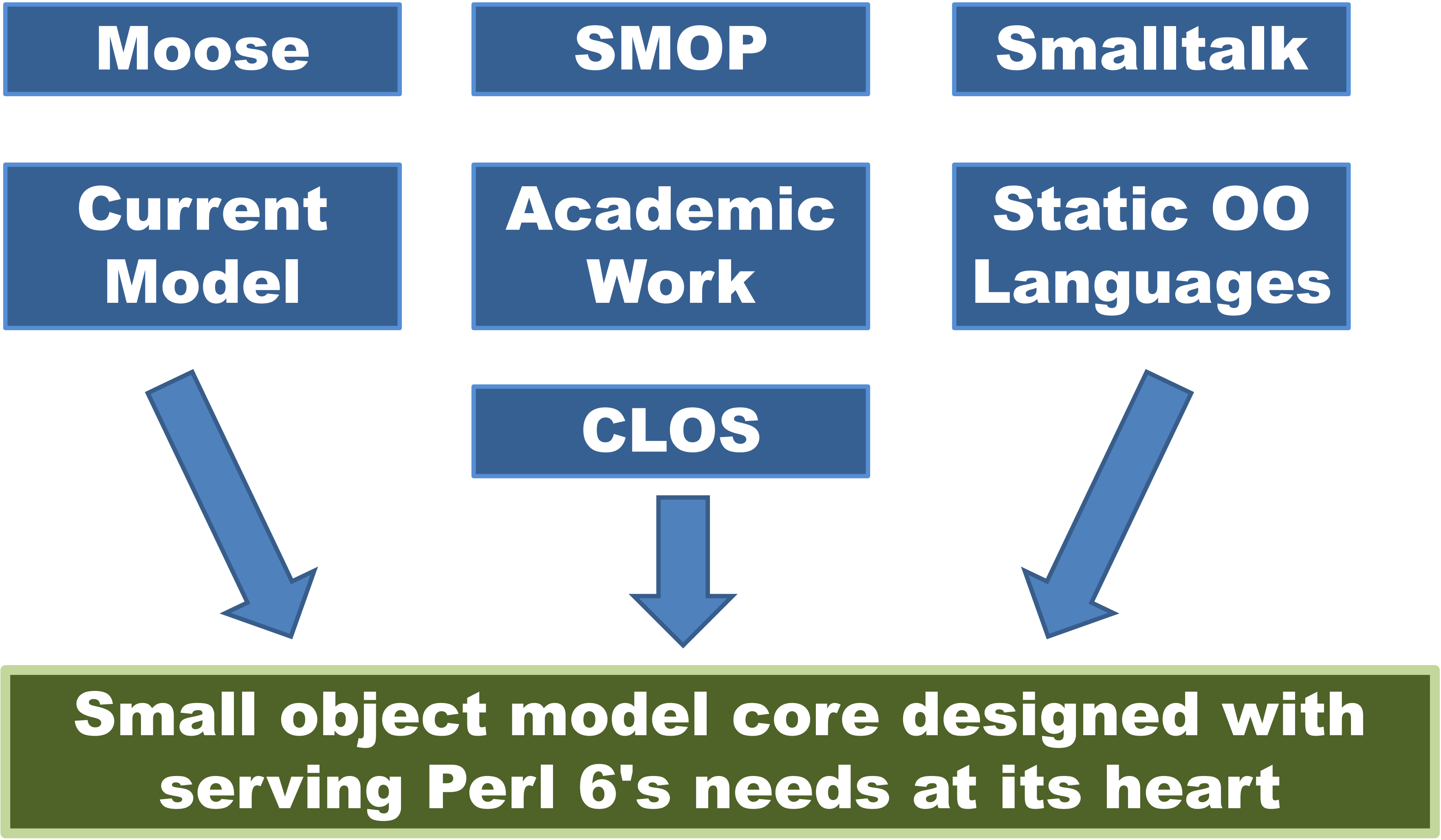
- ☹ Semantic gap hurts
- ☹ Hard to hack on or change
- ☹ Hard to reason about
- ☹ Tricky to port to other VMs
- ☹ Performance issues
- ☹ No easy path to implement type-driven optimizations
- ☹ No easy path to implement representation polymorphism

Let's look at this with fresh eyes.

Taking Rakudo Forward: What I'm Hacking On

Small object model core designed with serving Perl 6's needs at its heart

Learn from...



Moose

SMOP

Smalltalk

Current Model

Academic Work

Static OO Languages

CLOS

Small object model core designed with serving Perl 6's needs at its heart

So what do I want out of this process?

Small Low-Level Core

Write the rest in Perl 6

(or a subset of it)

Taking Rakudo Forward: What I'm Hacking On

Tension between

"low-level and fast at runtime"

and

"high level, hackable, extensible and maintainable"

"What are the core primitives to try and get really fast?"

Method dispatch in the common, optimizable cases

Attribute access

Type checks

Object instantiation

Don't need to worry quite so much over…

Taking Rakudo Forward: What I'm Hacking On

Type construction
(happens at compile time)

Role composition

Introspection

The uncommon cases

Conclusions

Primitives will be:

Method dispatch

Attribute storage and lookup

Object allocation

Build everything else (classes, inheritance, roles, introspection) out of them

Representation Polymorphism

How do we represent an object in memory?

How do we store attributes?

How do we box/unbox native types?

These are all issues related to representation.

Perl 6 offers representation polymorphism, to allow classes to choose (or let the class user choose) a representation strategy.

Taking Rakudo Forward: What I'm Hacking On

Possible to leave a class open to being instantiated with different representations

```
class Color::RGB is repr(*) {
    has uint8 $.red;
    has uint8 $.green;
    has uint8 $.blue;
}
```

“I want to store lots of these in an array” => bit-packed representation

“Just one, fast access” => typical word-aligned representation

Conclusions

We shall have two core APIs.

HOW API = control over dispatch, declarations, introspection

REPR API = control over object allocation, attribute storage (and if applicable, GC interaction)

Gradual Typing

How much type information is there in this code?

```
sub get_cat_from_rescue_center($type, $owner) {
    my Cat $rescued = cat_search($type);
    $rescued.owner = $owner;
    return $rescued;
}

my $kitteh = get_cat_from_rescue_center(
    'tabby', 'Anna');
```

How much type information is there in this code?

```
sub get_cat_from_rescue_center($type, $owner) {
    my Cat $rescued = cat_search($type);
    $rescued.owner = $owner;
    return $rescued;
}

my $kitteh = get_cat_from_rescue_center(
    'tabby', 'Anna');
```

`Cat`

How much type information is there in this code?

```
sub get_cat_from_rescue_center($type, $owner) {
    my Cat $rescued = cat_search($type);
    $rescued.owner = $owner;
    return $rescued;
}

my $kitteh = get_cat_from_rescue_center(
    'tabby', 'Anna');
```

Cat   Any

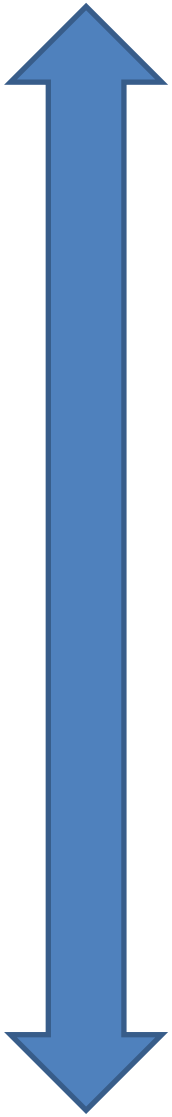
How much type information is there in this code?

```
sub get_cat_from_rescue_center($type, $owner) {
    my Cat $rescued = cat_search($type);
    $rescued.owner = $owner;
    return $rescued;
}

my $kitteh = get_cat_from_rescue_center(
    'tabby', 'Anna');
```

Cat Any Mu

Taking Rakudo Forward: What I'm Hacking On



No extra type information provided

Fully Statically typed program

The compiler lets you choose how much type information to provide

and

tries to give you more benefits as you give it more information to work with

Taking Rakudo Forward: What I'm Hacking On

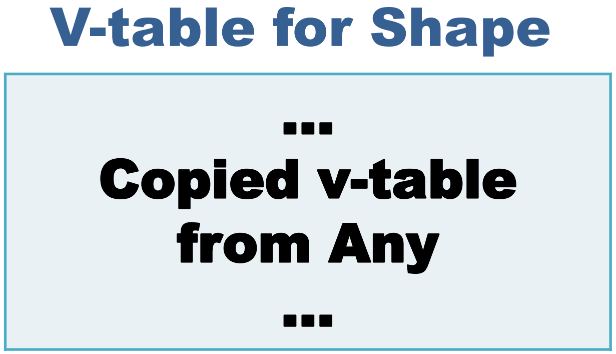
A key place we can take advantage of type information is to optimize method dispatches

Normally, we look up methods in a hash table

Faster is to index into a v-table

Taking Rakudo Forward: What I'm Hacking On

```
class Shape {
    has $.name;
    method area() { ... }
}
class Square is Shape {
    method area($side) { $side ** 2 }
}
```

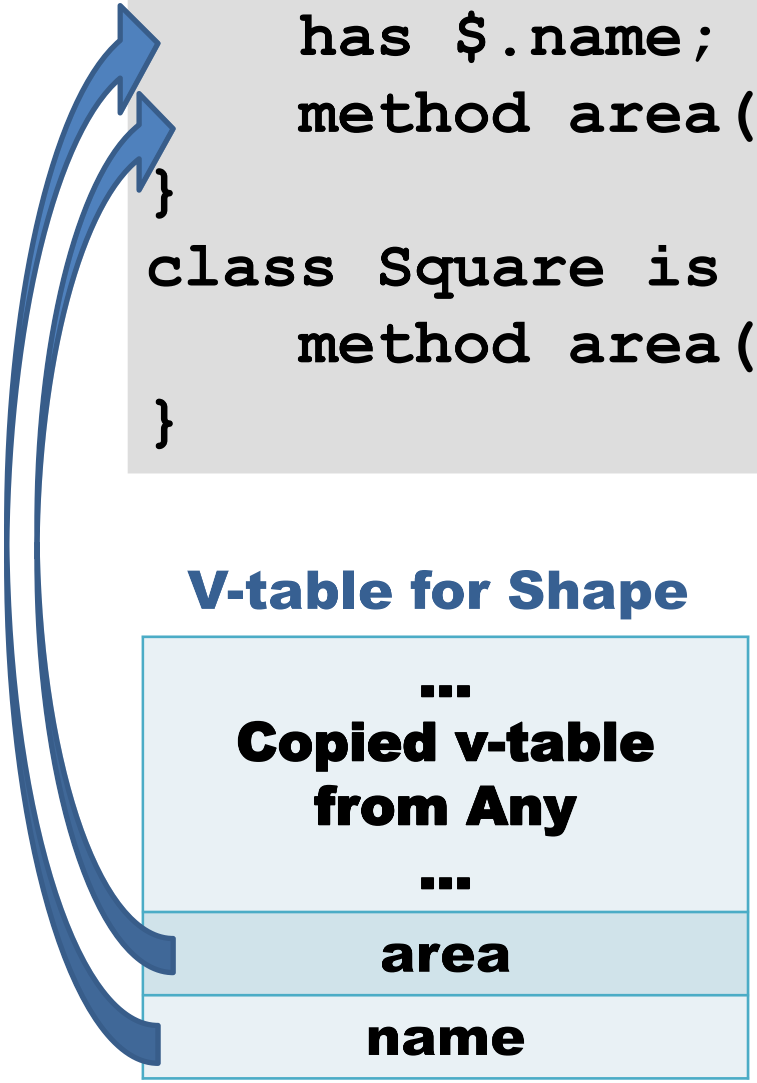


V-table for Shape

...
Copied v-table from Any
...

Taking Rakudo Forward: What I'm Hacking On

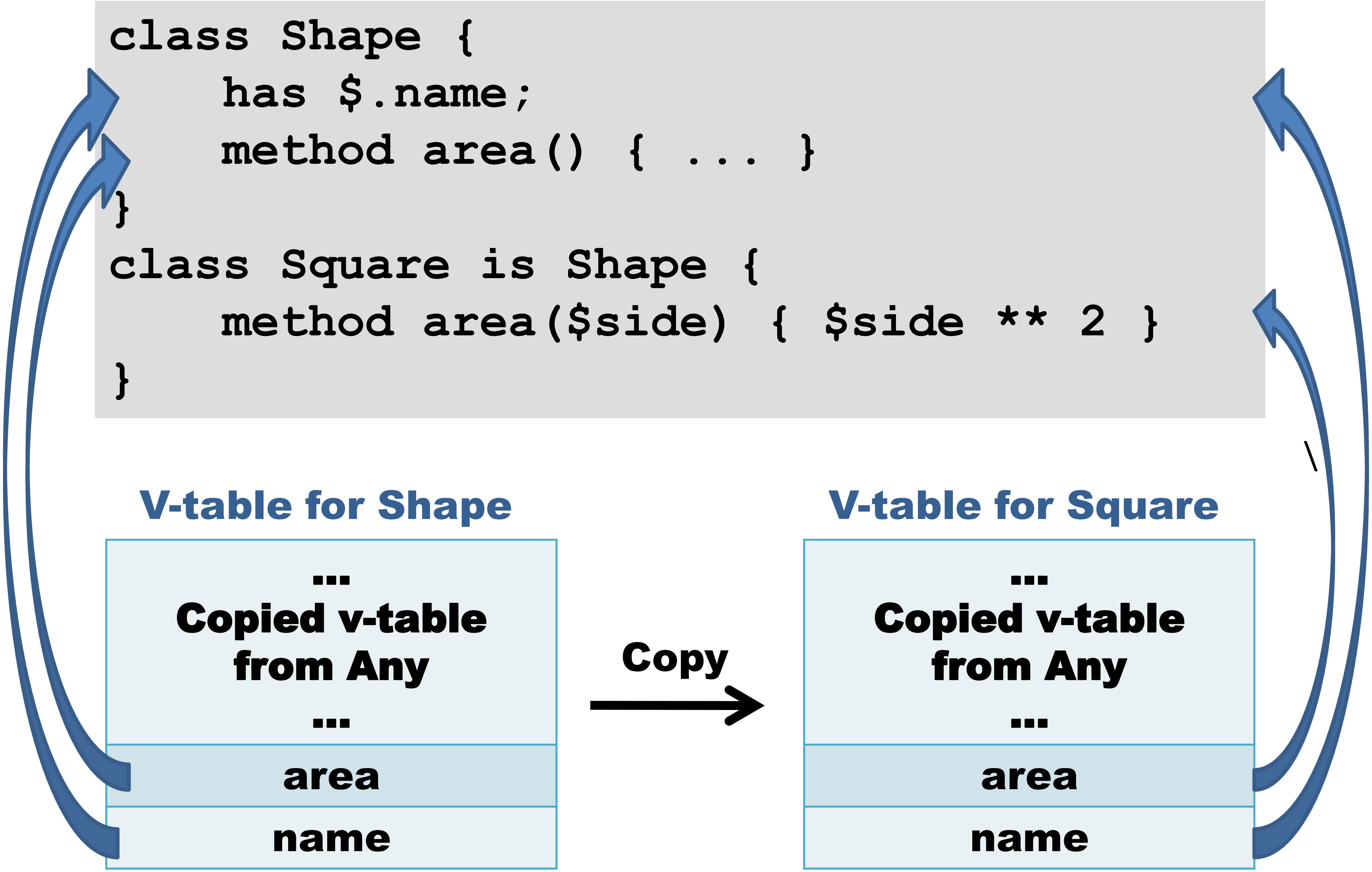
```
class Shape {
    has $.name;
    method area() { ... }
}
class Square is Shape {
    method area($side) { $side ** 2 }
}
```



V-table for Shape

... Copied v-table from Any ...
area
name

Taking Rakudo Forward: What I'm Hacking On



```
class Shape {
    has $.name;
    method area() { ... }
}
class Square is Shape {
    method area($side) { $side ** 2 }
}
```

V-table for Shape

...
Copied v-table
from Any
...

area

name

Copy

V-table for Square

...
Copied v-table
from Any
...

area

name

Conclusions

Compiling method dispatches to v-table lookups means we need the meta-objects built and available at compile time

Single unified compile time and runtime MOP...

...and a place to hang a v-table

The Model So Far

Taking Rakudo Forward: What I'm Hacking On

Object

...

Taking Rakudo Forward: What I'm Hacking On

Object

...

Meta-object

new_type
add_method
add_parent
add_attribute
compose
methods
parents
attributes
...

Taking Rakudo Forward: What I'm Hacking On

Object

...

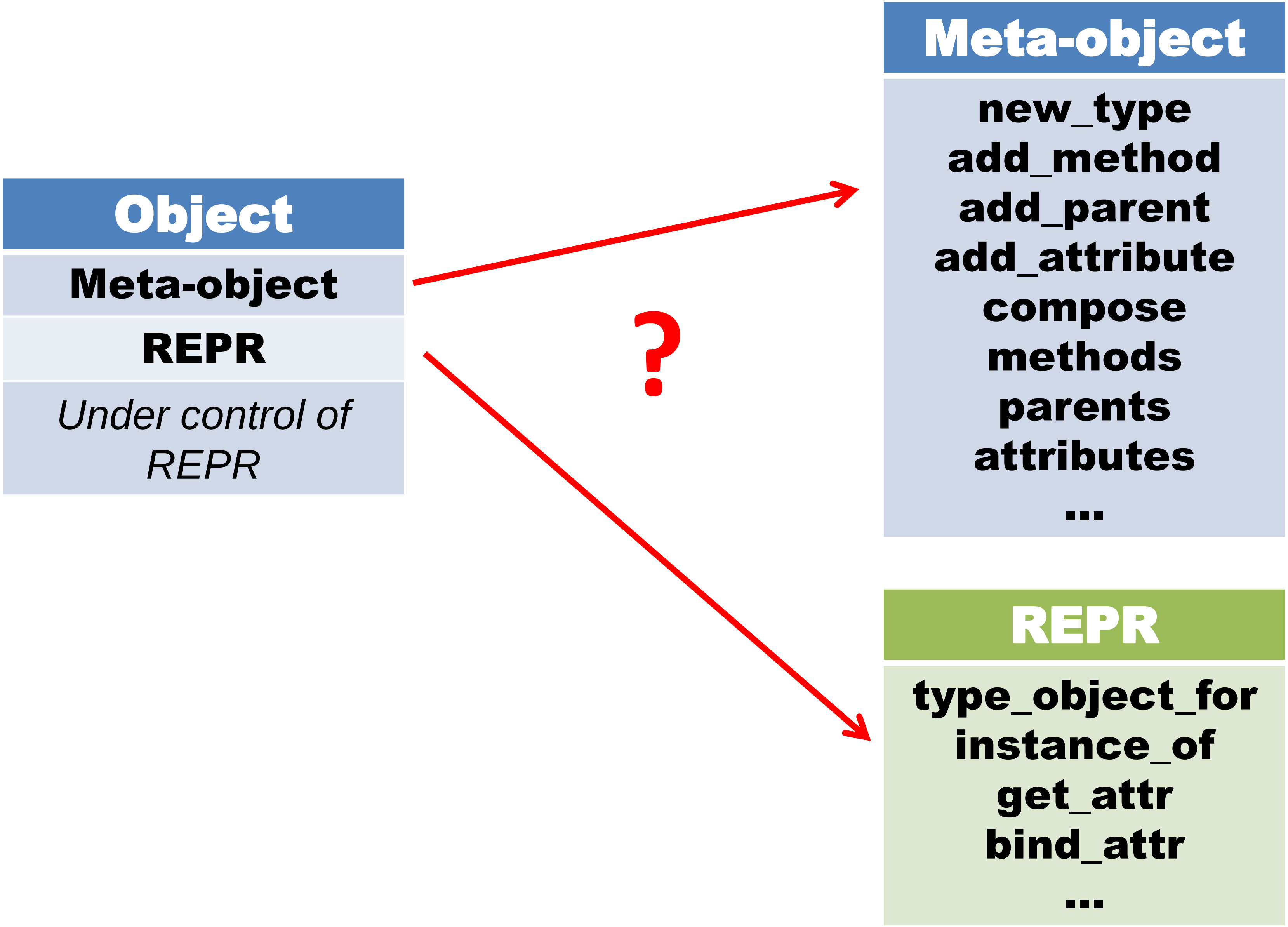
Meta-object

new_type
add_method
add_parent
add_attribute
compose
methods
parents
attributes
...

REPR

type_object_for
instance_of
get_attr
bind_attr
...

Taking Rakudo Forward: What I'm Hacking On



Object

Meta-object

REPR

Under control of REPR

?

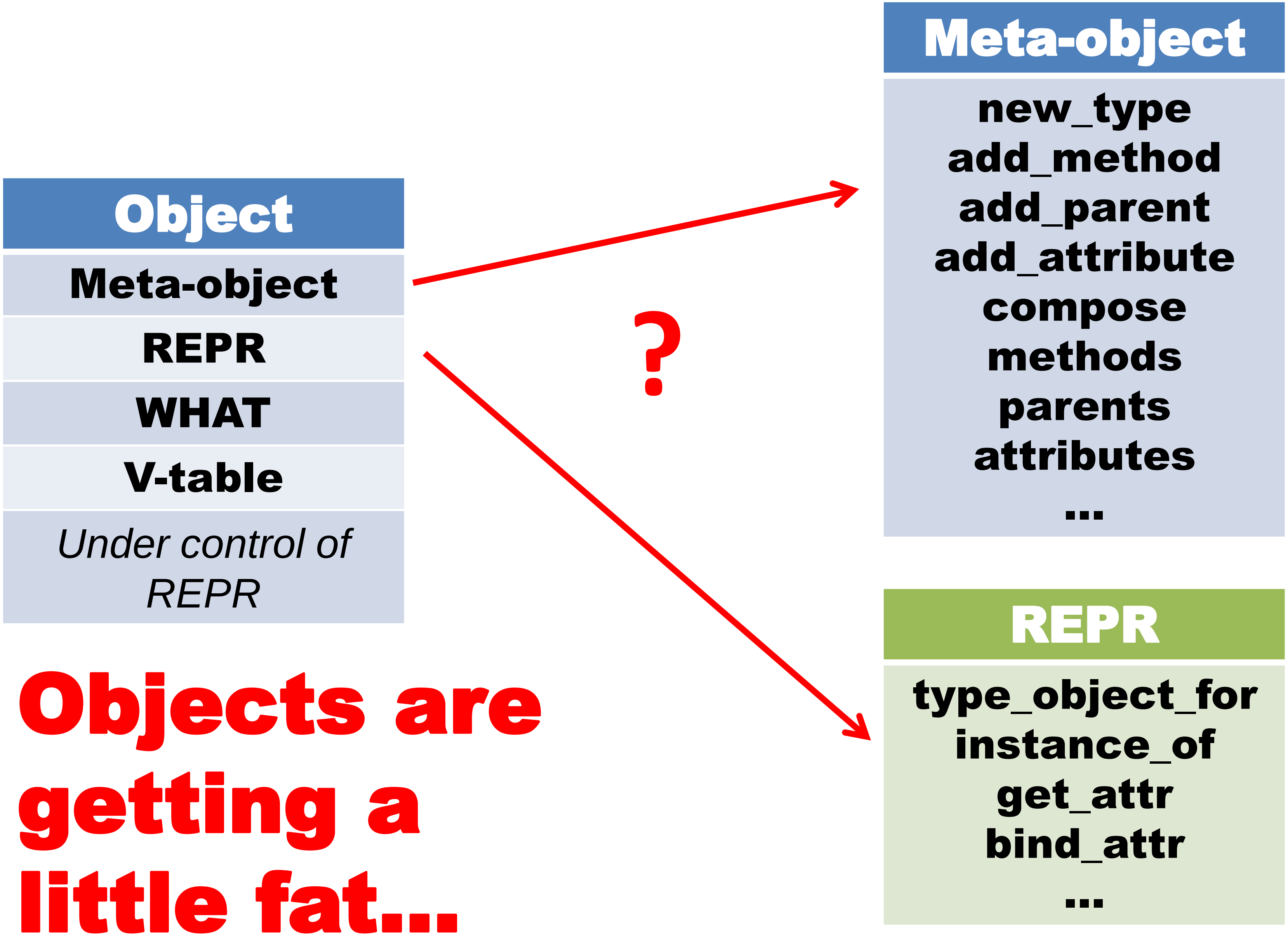
Meta-object

new_type
add_method
add_parent
add_attribute
compose
methods
parents
attributes
…

REPR

type_object_for
instance_of
get_attr
bind_attr
…

Taking Rakudo Forward: What I'm Hacking On



Object

Meta-object

REPR

WHAT

V-table

Under control of REPR

?

Meta-object

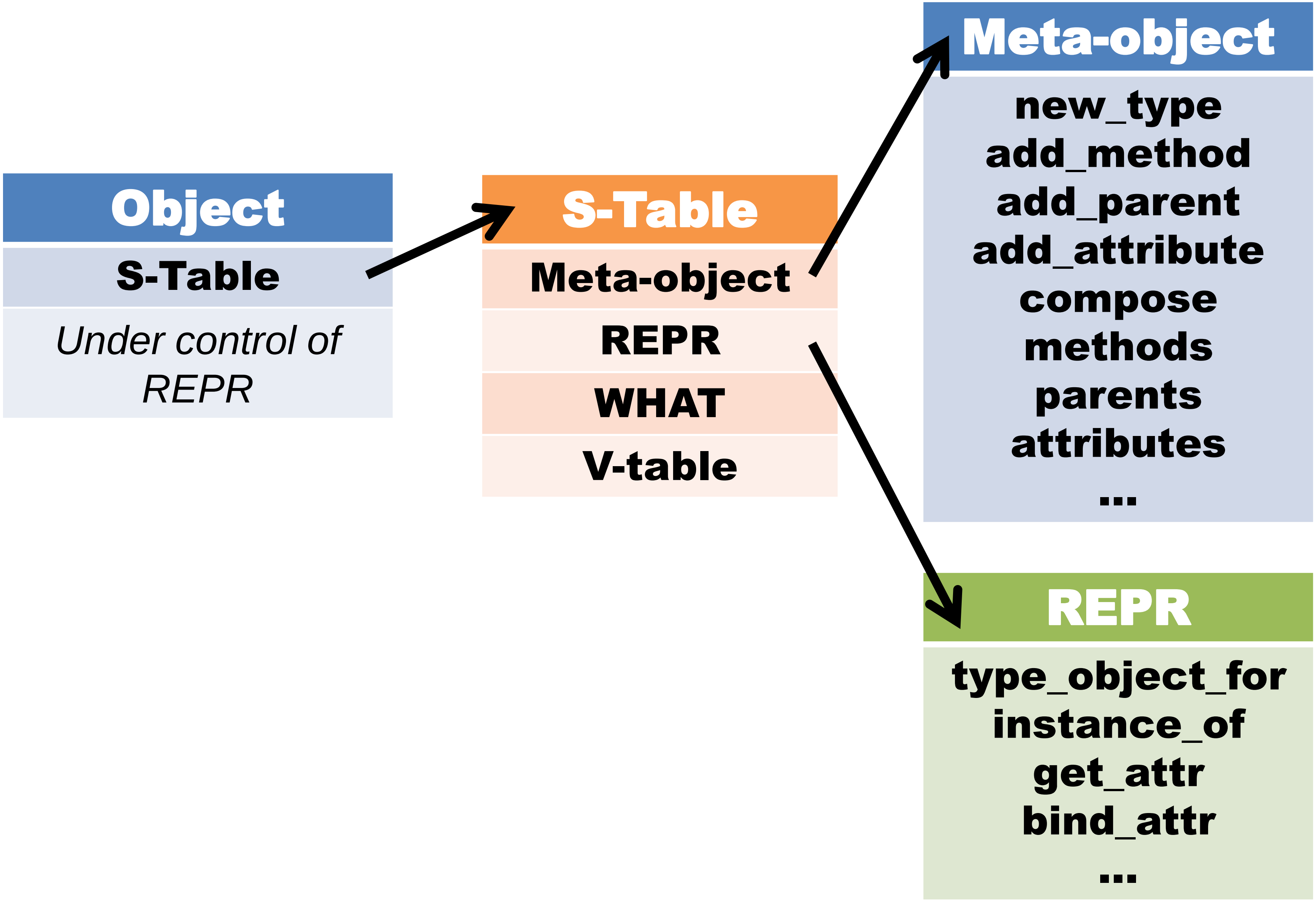
new_type
add_method
add_parent
add_attribute
compose
methods
parents
attributes
...

REPR

type_object_for
instance_of
get_attr
bind_attr
...

Objects are getting a little fat...

Taking Rakudo Forward: What I'm Hacking On



Object

S-Table

Under control of REPR

S-Table

Meta-object

REPR

WHAT

V-table

Meta-object

new_type
add_method
add_parent
add_attribute
compose
methods
parents
attributes
...

REPR

type_object_for
instance_of
get_attr
bind_attr
...

Bounded Serialization

Taking Rakudo Forward: What I'm Hacking On

We build the meta-objects and S-tables at compile time

but

We need them at runtime

Serialize (freeze) them at the end of the compile, and deserialize (thaw) them at program startup

Taking Rakudo Forward: What I'm Hacking On

One of the main reasons that Rakudo’s startup time is so bad today is that we have to construct all of the built-in types at startup.

Want to just serialize them all once and be able to quickly deserialize them each startup.

Taking Rakudo Forward: What I'm Hacking On

TRICKY PROBLEM IS TRICKY.

😐

Taking Rakudo Forward: What I'm Hacking On

Pizza.pm

```
use Food;
class Pizza {
    has $.diameter;
    has @.toppings;
}
```

Food.pm

```
class Food {
    has $.hot;
    has $.vegetarian;
}
```

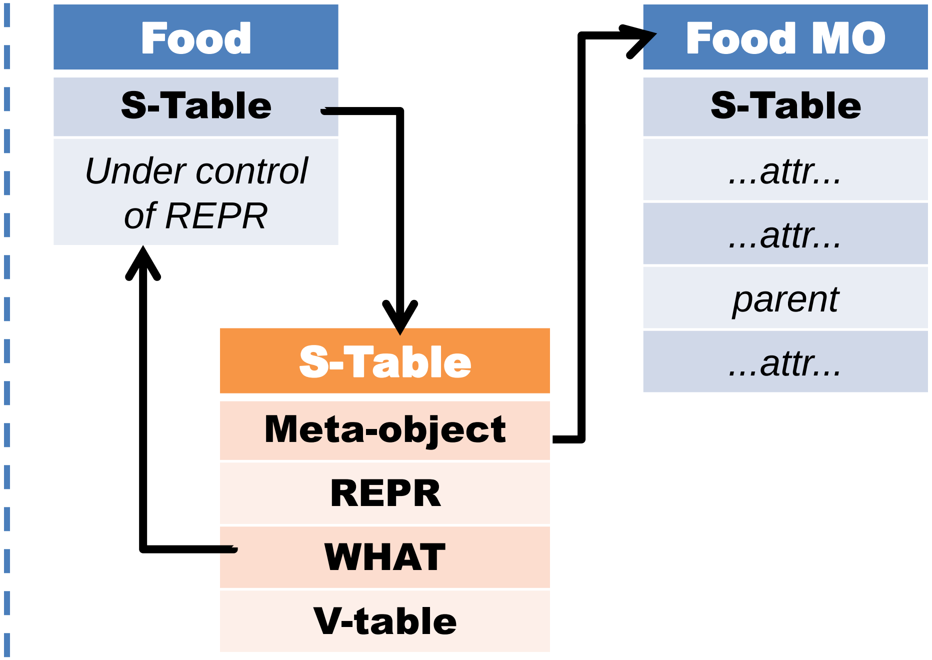
Taking Rakudo Forward: What I'm Hacking On

Pizza.pm

```
use Food;
class Pizza {
    has $.diameter;
    has @.toppings;
}
```

Food.pm

```
class Food {
    has $.hot;
    has $.vegetarian;
}
```



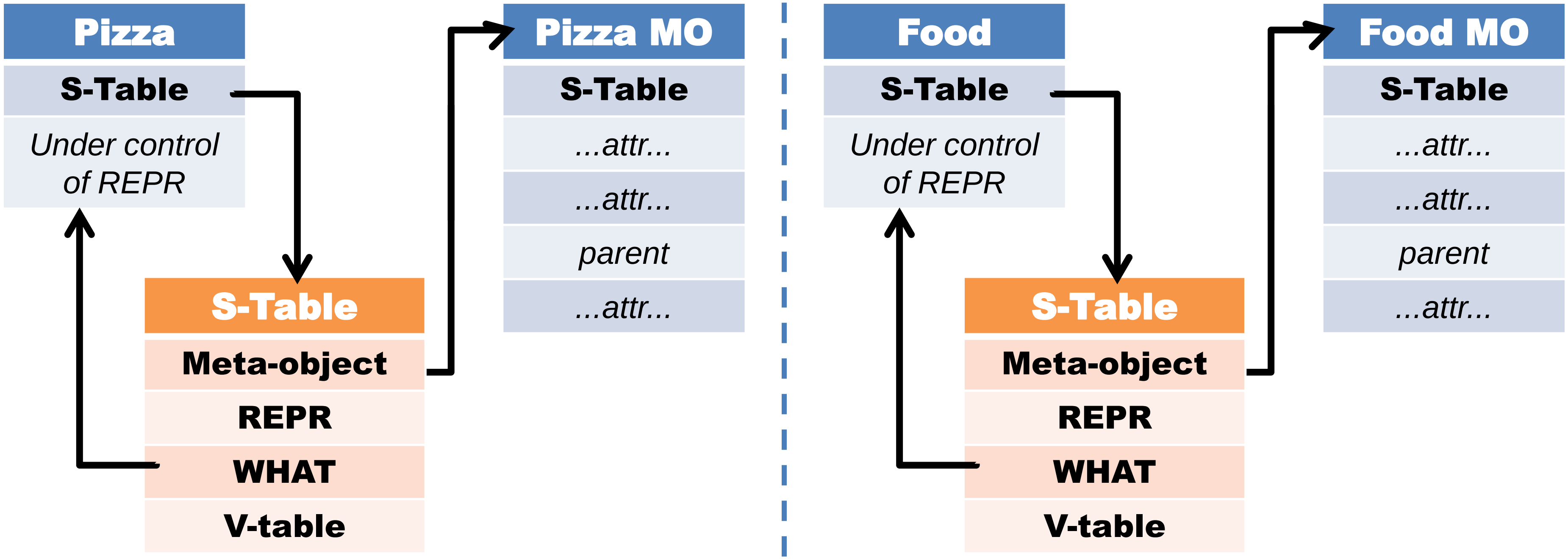
Taking Rakudo Forward: What I'm Hacking On

Pizza.pm

```
use Food;
class Pizza {
    has $.diameter;
    has @.toppings;
}
```

Food.pm

```
class Food {
    has $.hot;
    has $.vegetarian;
}
```



Pizza
S-Table
Under control of REPR

S-Table
Meta-object
REPR
WHAT
V-table

Pizza MO
S-Table
...attr...
...attr...
parent
...attr...

Food
S-Table
Under control of REPR

S-Table
Meta-object
REPR
WHAT
V-table

Food MO
S-Table
...attr...
...attr...
parent
...attr...

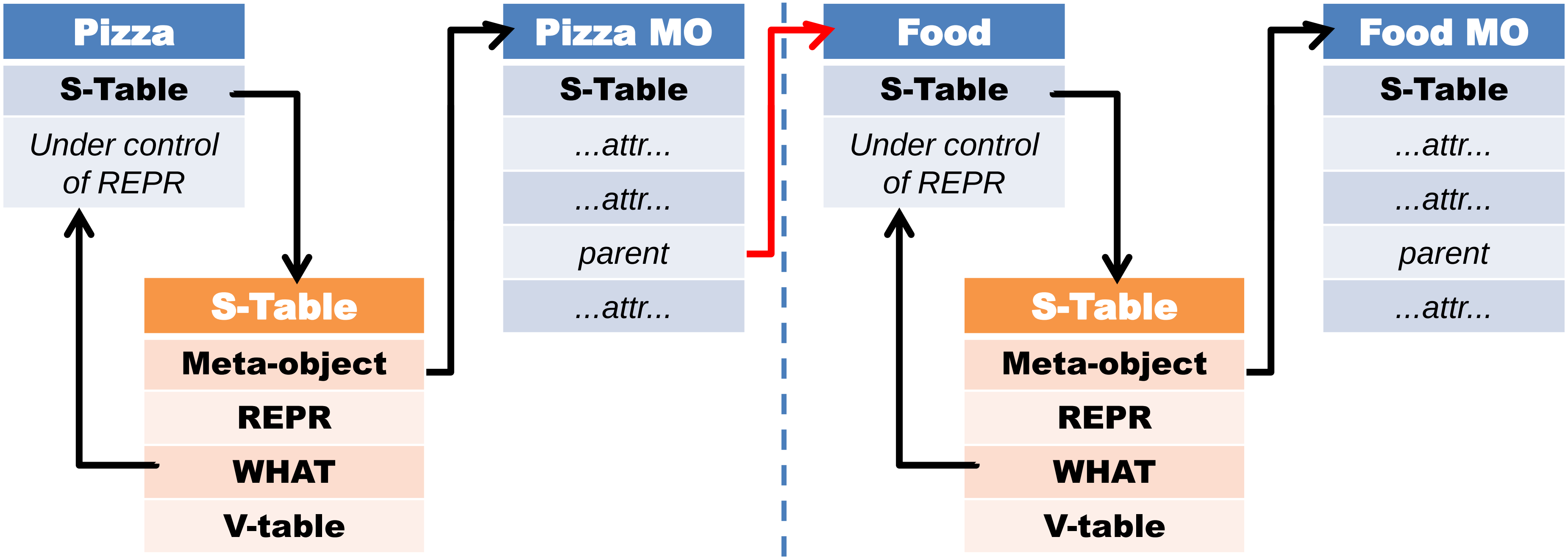
Taking Rakudo Forward: What I'm Hacking On

Pizza.pm

```
use Food;
class Pizza {
    has $.diameter;
    has @.toppings;
}
```

Food.pm

```
class Food {
    has $.hot;
    has $.vegetarian;
}
```



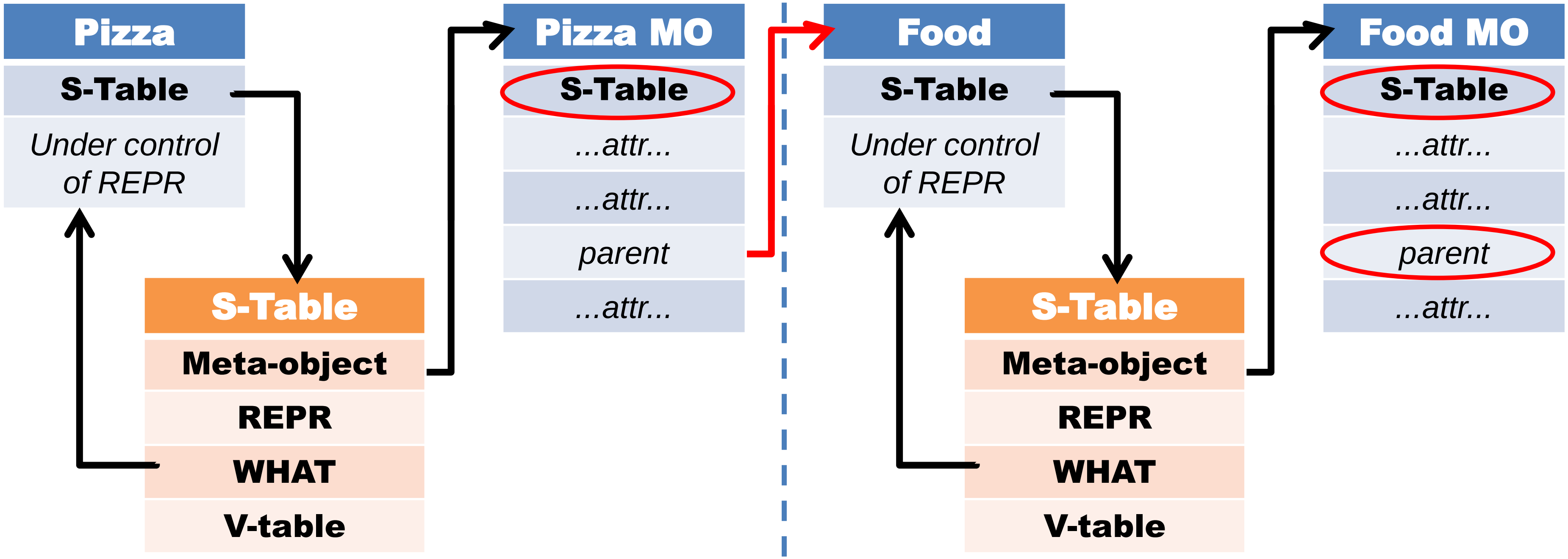
Taking Rakudo Forward: What I'm Hacking On

Pizza.pm

```
use Food;
class Pizza {
    has $.diameter;
    has @.toppings;
}
```

Food.pm

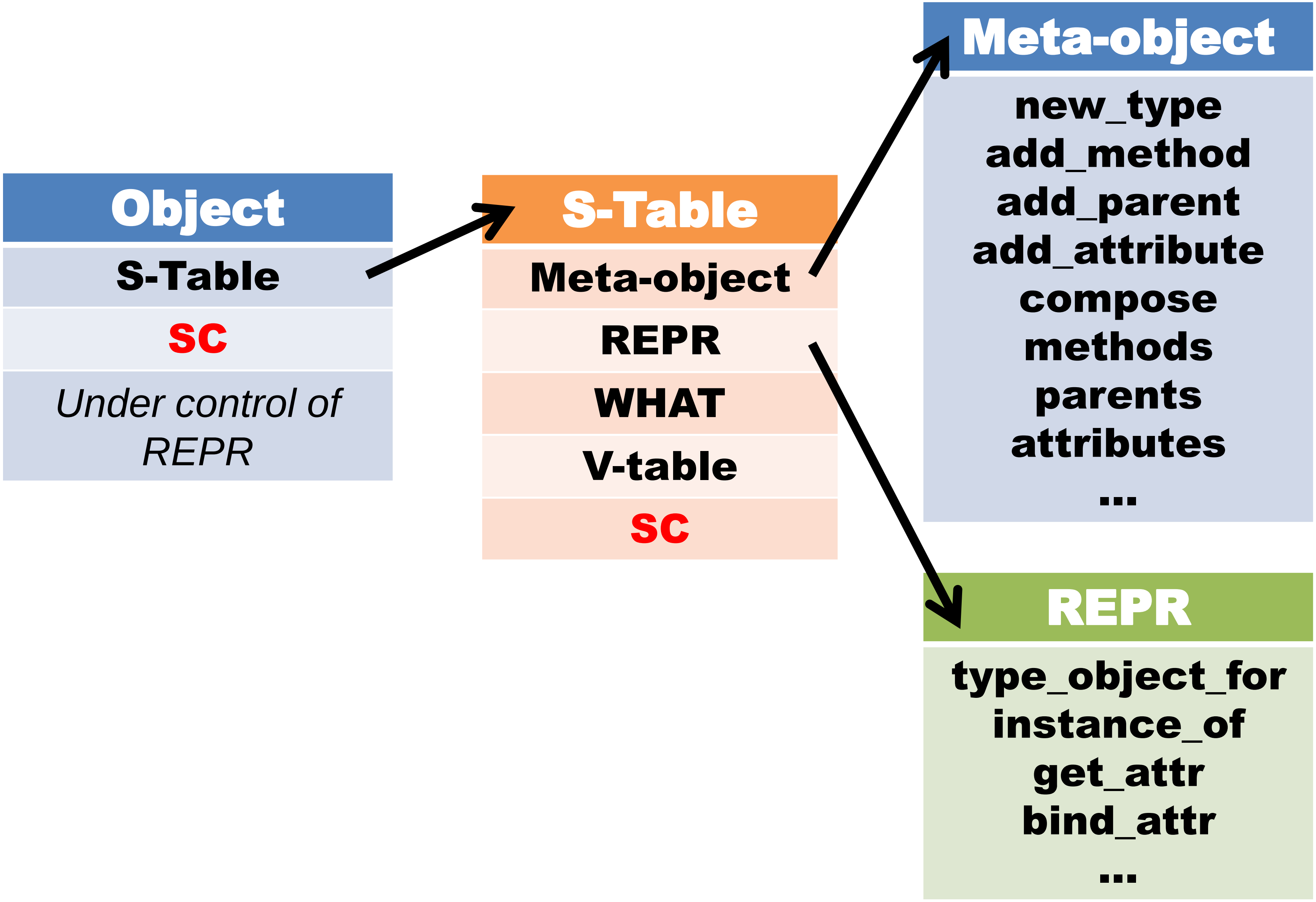
```
class Food {
    has $.hot;
    has $.vegetarian;
}
```



Taking Rakudo Forward: What I'm Hacking On

Give every object and every S-table a pointer to a Serialization Context.

Taking Rakudo Forward: What I'm Hacking On



Object

S-Table

SC

Under control of REPR

S-Table

Meta-object

REPR

WHAT

V-table

SC

Meta-object

new_type

add_method

add_parent

add_attribute

compose

methods

parents

attributes

...

REPR

type_object_for

instance_of

get_attr

bind_attr

...

When serializing, we visit objects added to our SC.

If it’s not in an SC, serialize it and visit its children.

If it already has an SC, serialize a fixup (reference) so we can link it.

Taking Rakudo Forward: What I'm Hacking On

VM
Portability

Taking Rakudo Forward: What I'm Hacking On

Today Rakudo only runs on and targets the Parrot VM.

Just as Perl 5 supports many platforms, in Perl 6 we want to support many runtimes.

“Perl 6 should be available everywhere.”

Taking Rakudo Forward: What I'm Hacking On

Small meta-model core

=

Small amount to port

Design is quite naturally portable. \o/

Current Status

Taking Rakudo Forward: What I'm Hacking On

The core of the model has been implemented.

Working representation polymorphism.

First, working cut of an implementation of classes.

Taking Rakudo Forward: What I'm Hacking On

Today, the core so far is implemented on:

Parrot
.Net CLR
JVM

In the future there will likely be more

but

don’t want to spread limited development resources too thin.

What now?

Taking Rakudo Forward: What I'm Hacking On

NQP

Finish filling out ClassHOW

Push it into the bootstrapped NQP on Parrot

Implement serialization contexts and serialization

Update NQP to use them

Taking Rakudo Forward: What I'm Hacking On

Rakudo

Get the grammar and actions to run on the updated NQP

Re-write the meta-objects to work with the new object model

Use serialization contexts

Debug until it works ☺

Taking Rakudo Forward: What I'm Hacking On

.Net/JVM

Get ClassHOW to run

Get NQP tests to pass

Bootstrapped, self-hosting NQP

Get Rakudo to run there

Merci beaucoup!

Questions?