Deploying Perl 6

Audrey Tang

Perl 6

is here Today!

Perl 6 is here Today!

(*YAPC::NA 2005*)

Pugs 6.2.12

- Released on June 26th
- 3x faster build time
- 10x faster compilation
- 2x faster runtime
- 2000+ commits since 6.2.11

Parrot 0.4.5

- Released last June 19th
- Unicode identifiers
- Hierarchical namespace
- New .NET CLR translator
- Much faster compiler tools

Great for

experimenting

But *not*

for production

...not *this*

Christmas


SPAIN

CPAN is

the language

Perl is

just its syntax

Perl

5.000b3h

(October 1994)

```
use 5.000;
use strict;
require 'fastcwd.pl';
require 'newgetopt.pl';
require 'exceptions.pl';
# ...
```

Continuity++

Pugs 6.2.2 (*June 2005*)

```
use v6-pugs;
use perl5:DBI;
use perl5:Encode;
use perl5:Template;
# ...
```

Still need to install Pugs

Perl

5.9.3

(Jan 2006)

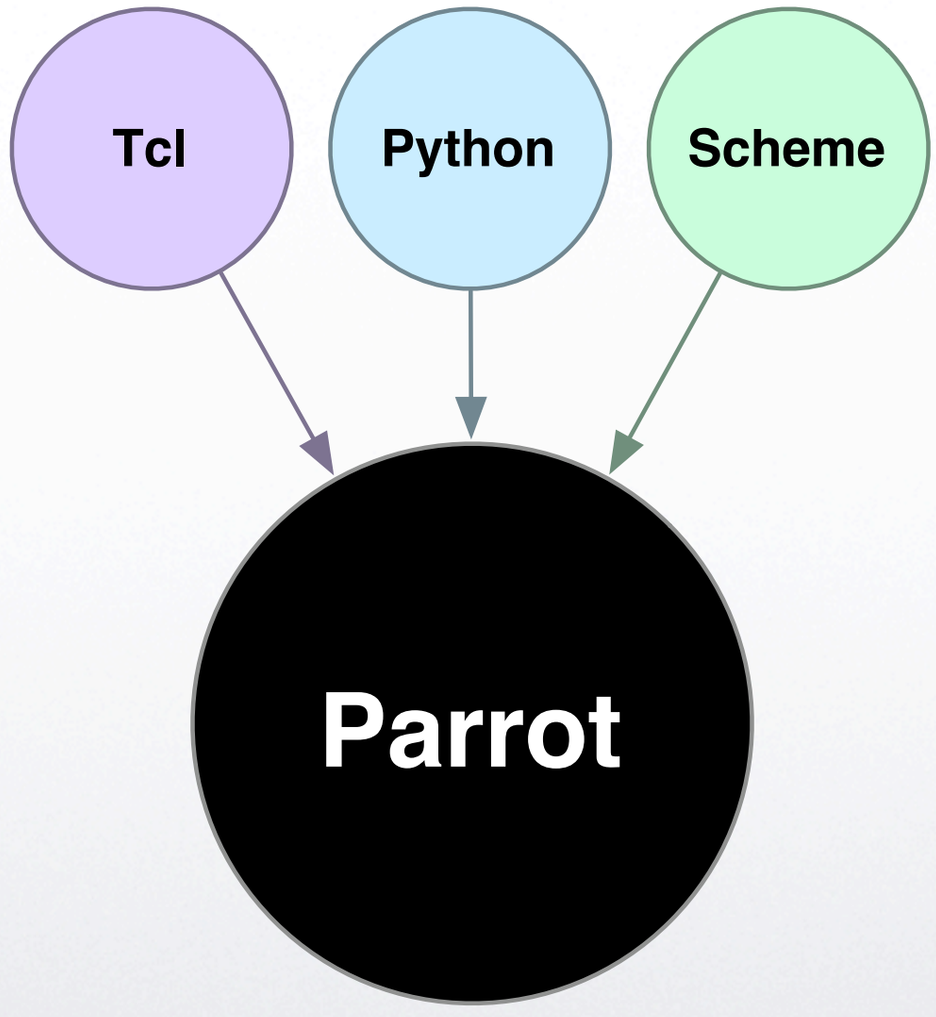
```
use v5.9.3;
use feature qw(switch say err ~~);

given (shift()) {
    when ['-h', '--help'] {
        say "Usage: $0";
    }
    default {
        $0 ~~ 'moose.exe' err die "Not Moose";
    }
}
```

How to get Perl 6 into Production?

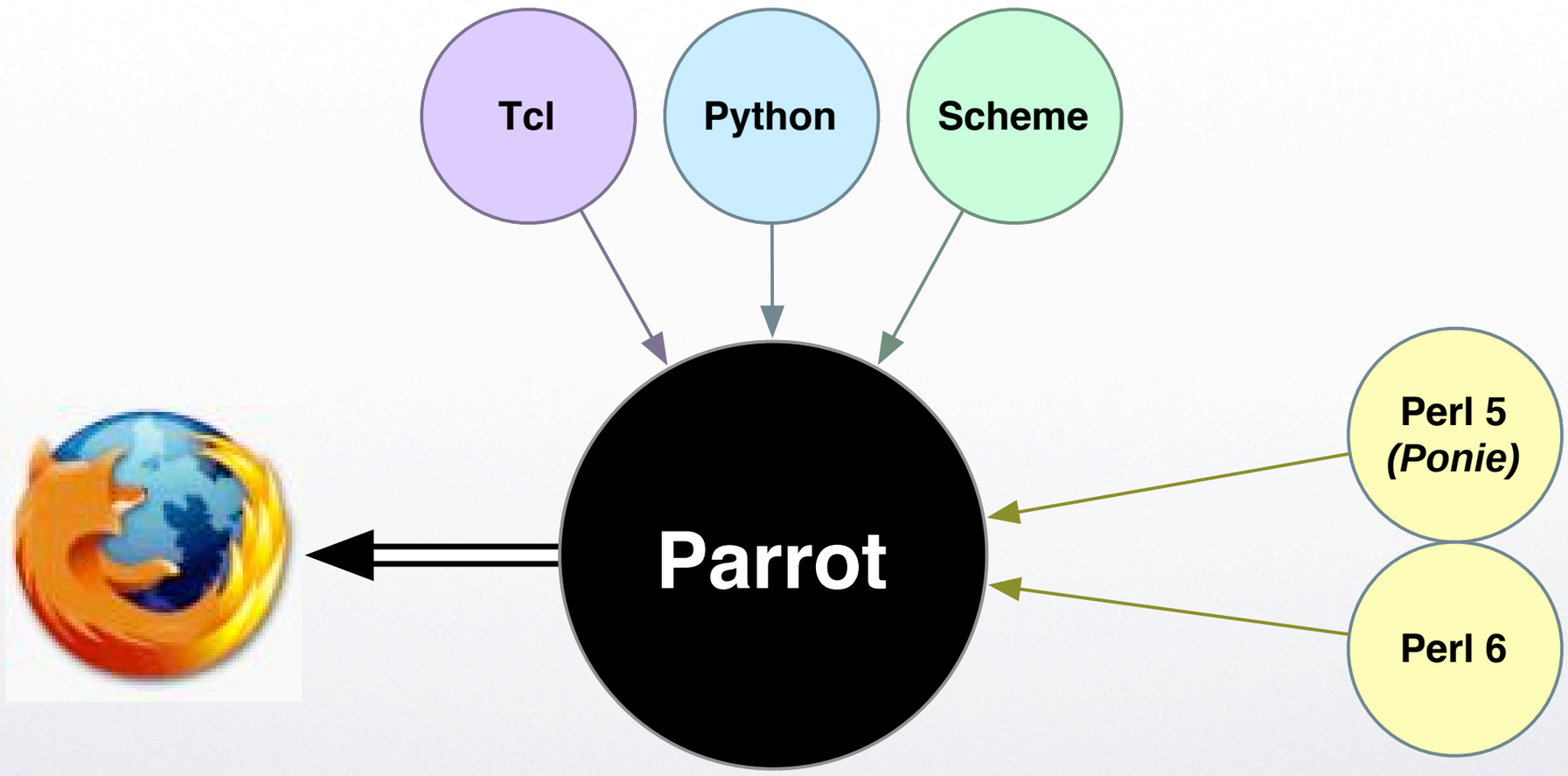
Production

- Work with existing code
- Must support Perl 5 and XS
- No from-scratch rewrites

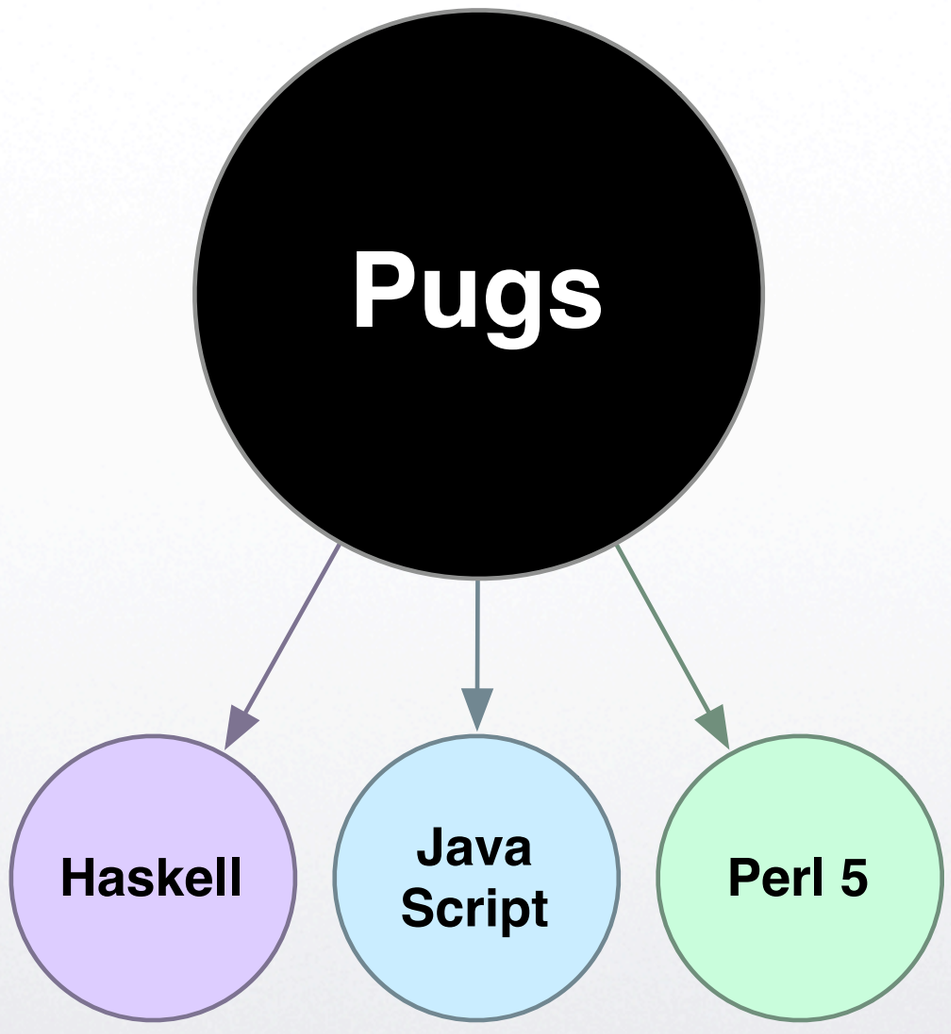
Frontends?



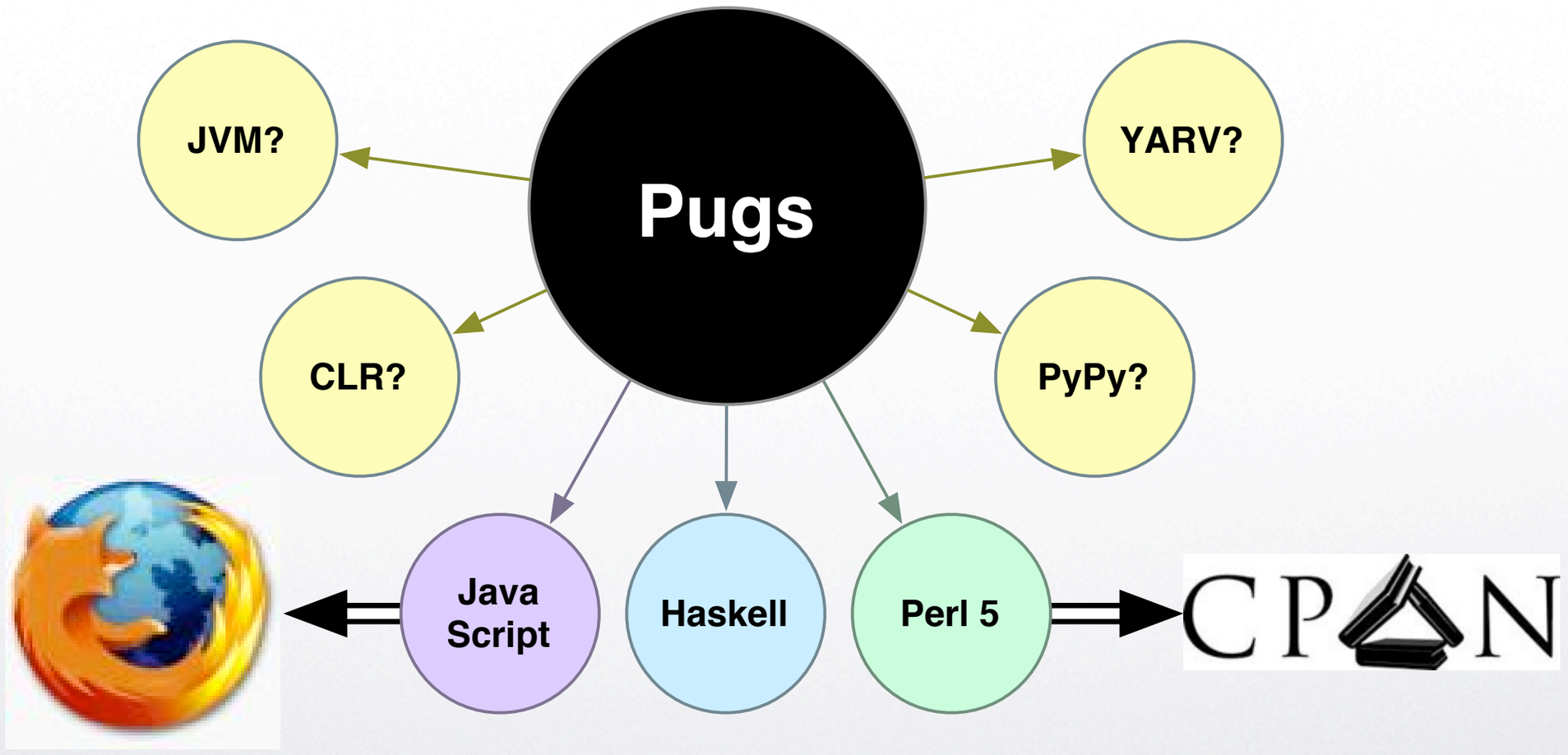
Frontends?



Backends!



Backends!



Pugs on Perl 5

Perl 6 Runtime Implemented as Perl 5 Modules

Sane Perl 5

(*not* source filters)

Available On CPAN Today

Moose.pm

What is Moose?

- *Complete* object model for Perl 5
- *Based on* the Perl 6 object model
- Production Ready

Moose, it's the new Camel

Møøse Fixation

Objects

With Class

```
use v6-pugs;
class Point;


has $.x is rw; # instance attributes
has $.y;       # default "is readonly"
```

```
method clear () {

    $.x = 0;          # accessor
    $!y = 0;          # direct slot access
}
```

```
use v5;
package Point;
use Moose;

has x => (is => 'rw');
has y => (is => 'ro');
```

```
sub clear {
    my $self = shift;
    $self->x(0);     # accessor
    $self->{y} = 0; # direct slot access
}
```

Subclassing

```
use v6-pugs;
class Point3D;


is Point;

has $.z;

method clear () {

    call;
    $!z = 0;
};
```

```
use v5;
package Point3D;
use Moose;

extends 'Point';

has z => (isa => 'Int');

override clear => sub {
    my $self = shift;
    super;
    $self->{z} = 0;
};
```

```
use v5;
package Point3D;
use Moose;

extends 'Point';

has z => (isa => 'Int');

after clear => sub {
    my $self = shift;

    $self->{z} = 0;
};
```

Meta

Objects

```
use v6-pugs;

Foo.meta.add_method(bar => method {
    say "Hello from Foo.bar";
});
Foo.meta.superclasses;
Foo.meta.compute_all_applicable_methods;
Foo.meta.find_all_methods_by_name($method_name);
```

```
use v5;
use Moose;
Foo->meta->add_method(bar => sub {
    print "Hello from Foo->bar\n";
});
Foo->meta->superclasses;
Foo->meta->compute_all_applicable_methods;
Foo->meta->find_all_methods_by_name($method_name);
```

Laziness

```
use v6-pugs;
class BinaryTree is rw;

has Any $.node;
has BinaryTree $.parent handles {
    parent_node => 'node'
};
has BinaryTree $.left = {
    lazy { BinaryTree.new( parent => self ) }
};
has BinaryTree $.right = {
    lazy { BinaryTree.new( parent => self ) }
};
```

```
use v5;
package BinaryTree;
use Moose;

has node => (is => 'rw', isa => 'Any');
has parent => (
    is => 'rw',
    isa => 'BinaryTree',
    handles => { parent_node => 'node' },
    weak_ref => 1,
);
has left => (
    is => 'rw',
    isa => 'BinaryTree',
    default => sub { BinaryTree->new(parent => $_[0]) },
    lazy => 1,
);
# ditto for “has right”
```

Type

Constraints

```
use v6-pugs;
class Address;
use perl5:Locale::US;
use perl5:Regexp::Common <zip $RE>;

my $STATES = Locale::US.new;
subset US_State of Str where {
    $STATES{any(<code2state state2code>)}{.uc};
};

has Str $.street is rw;
has Str $.city is rw;
has US_State $.state is rw;
has Str $.zip_code is rw where {
    $_ ~~ $RE<zip><US>{'-extended' => 'allow'}
};
```

```
use v5;
package Address;
use Moose;
use Moose::Util::TypeConstraints;
use Locale::US;
use Regexp::Common 'zip';

my $STATES = Locale::US->new;
subtype USState => as Str => where {
    $STATES->{code2state}{uc($_)}
    or $STATES->{state2code}{uc($_)};
}

has street => (is => 'rw', isa => 'Str');
has city => (is => 'rw', isa => 'Str');
has state => (is => 'rw', isa => 'USState');
has zip_code => (
    is => 'rw',
    isa => subtype Str => where {
        /$RE{zip}{US}{-extended => 'allow'}/
    },
);
```

Roles

```
use v6-pugs;
role Equality;

method equal_to ($other) { ... }

method not_equal_to ($other) {

    not $self->equal_to($other);
}
```

```
use v5;
package Equality;
use Moose::Role;
requires 'equal_to';

sub not_equal_to {
    my ($self, $other) = @_;
    not $self->equal_to($other);
}
```

Derived

Roles

```
use v6-pugs;
role Comparable;
does Equality;

method compare ($other) { ... }

method equal_to ($other) {

    $self->compare($other) == 0;
}

method greater_than ($other) {

    $self->compare($other) == 1;
}

method less_than ($other) {

    $self->compare($other) == -1;
}
```

```
use v5;
package Comparable;
with 'Equality';

requires 'compare';

sub equal_to {
    my ($self, $other) = @_;
    $self->compare($other) == 0;
}
```

```
sub greater_than {
    my ($self, $other) = @_;
    $self->compare($other) == 1;
}

sub less_than {
    my ($self, $other) = @_;
    $self->compare($other) == -1;
}
```

Abstract Interface

Roles

```
use v6-pugs;
role Printable;

method to_string { ... }
```

```
use v5;
package Printable;
use Moose::Role;

requires 'to_string';
```

Mixing In Roles

```
use v6-pugs;
class Person;
does Comparable;
does Printable;

has Str $.first_name;
has Str $.last_name;

method to_string () {

    "$.last_name, $.first_name";
}

method compare ($other) {

    $.to_string cmp $other.to_string;
}
```

```
use v5;
package Person;
use Moose;
with 'Comparable', 'Printable';

has first_name => (isa => 'Str');
has last_name => (isa => 'Str');

sub to_string {
    my $self = shift;
    $self->last_name . ', ' . $self->first_name;
}

sub compare {
    my ($self, $other) = @_;
    $self->to_string cmp $other->to_string;
}
```

More features

- Automatic Coercion
- Accessor Triggers
- Alternate layouts!

Pugs::Compiler::Rule

Regex

Objects

```
use v6-pugs;

my $txt = 'Car-ModelT,1909';
my $pat = rx{
     Car -
          [ ( Ferrari )
          | ( ModelT , (\d\d\d\d) )
          ]
};
$txt ~~ $pat err fail "Cannot match";
```

```
use v5;
use Pugs::Compiler::Regex;
my $txt = 'Car-ModelT,1909';
my $pat = Pugs::Compiler::Regex->compile(q(
    Car -
        [ ( Ferrari )
        | ( ModelT , (\d\d\d\d) )
        ]
));
$pat->match($txt) or die "Cannot match";
```

Match

Objects

```
use v6-pugs;

my $pat = rx{
    Car - [
        ( Ferrari ) | ( ModelT , (\d\d\d\d) )
    ]
};
```

```
my $match = ('Car-ModelT,1909' ~~ $pat);
say $match;                 # "Car-ModelT,1909"
say $match[0];              # undef
say $match[1];              # "ModelT,1909"
say $match[1][0];           # "1909"
say $match[1][0].from; # 11
say $match[1][0].to;   # 15
```

```
use v5;
use Pugs::Compiler::Regex;
my $pat = Pugs::Compiler::Regex->compile(q(
    Car - [
      ( Ferrari ) | ( ModelT , (\d\d\d\d) )
    ]
));

use feature qw( say );
my $match = $pat->match('Car-ModelT,1909');
say $match;                    # "Car-ModelT,1909"
say $match->[0];               # undef
say $match->[1];               # "ModelT,1909"
say $match->[1][0];            # "1909"
say $match->[1][0]->from; # 11
say $match->[1][0]->to;   # 15
```

Named

Captures

```
use v6-pugs;

my $pat = rx{
    Car - [
        ( Ferrari )
      | ( ModelT , $<year>:=[\d\d\d\d] )
    ]
};
```

```
my $match = ('Car-ModelT,1909' ~~ $pat);
say $match;                    # "Car-ModelT,1909"
say $match[1];                 # "ModelT,1909"
say $match[1]<year>;           # "1909"
say $match[1]<year>.from; # 11
say $match[1]<year>.to;   # 15
```

```
use v5;
use Pugs::Compiler::Regex;
my $pat = Pugs::Compiler::Regex->compile(q(
    Car - [
        ( Ferrari )
      | ( ModelT , $<year>:=[\d\d\d\d] )
    ]
));
```

```
use feature qw( say );
my $match = $pat->match('Car-ModelT,1909');
say $match;                          # "Car-ModelT,1909"
say $match->[1];                     # "ModelT,1909"
say $match->[1]{year};               # "1909"
say $match->[1]{year}->from; # 11
say $match->[1]{year}->to;   # 15
```

Grammar

Modules

```
use v6-pugs;

grammar CarInfo;

regex car {
    Car - [ ( Ferrari ) | ( ModelT , <year> ) ]
}
regex year {
    \d\d\d\d
}

module Main;
my $match = ('Car-ModelT,1909' ~~ CarInfo.car);
```

```
use v5;
use Pugs::Compiler::Regex;
package CarInfo;
use base 'Pugs::Grammar::Base';
*Car - Pugs::Compiler::Regex->compile(q(
    Car - [ ( Ferrari ) | ( ModelT , <year> ) ]
))->code;
*year = Pugs::Compiler::Regex->compile(q(
    \d\d\d\d
))->code;

package main;
my $match = CarInfo->car('Car-ModelT,1909');
```

Result

Objects

```perl
# XXX - Typical Perl5 code
use v5;
my $txt = 'Car-ModelT,1909';
my $pat = qr{
    Car - (?: ( Ferrari ) | ( ModelT , (\d\d\d\d) ) )
}x;

my $obj;
if ($txt =~ $pat) {
    if ($1) {
        $obj = Car->new(color => "red");
    } elsif ($2) {
        $obj = Car->new(color => "black", year => $3);
    }
}
```

```
use v6-pugs;

my $txt = 'Car-ModelT,1909';
my $pat = rx{
    Car - [ Ferrari
              { return Car.new(:color<red>) }
          | ModelT , $<year>:=[\d\d\d\d]
              { return Car.new(:color<black> :$<year>) }
          ]
};

my $obj = $($txt ~~ $pat);
print $obj<year>; # 1909
```

```
use v5;
use Pugs::Compiler::Regex;
my $txt = 'Car-ModelT,1909';
my $pat = Pugs::Compiler::Regex->compile(q(
    Car - [ Ferrari
              { return Car->new(color => 'red') }
          | ModelT , $<year>:=[\d\d\d\d]
              { return Car->new(
                    color => 'black', year => $<year>) }
          ]
));
my $obj = $pat->match($txt)->();
print $obj->{year}; # 1909
```

Backtrack

Control

```
use v6-pugs;
"ModelT2005" ~~ regex {
    Car - ModelT \d* ;
};
```

```
use v5;
"ModelT2005" =~ qr{
    Car - ModelT \d* ;
}x;
```

```
use v6-pugs;
"ModelT2005" ~~ token {
    Car - ModelT \d* ;
};
```

```
use v5;
"ModelT2005" =~ qr{
    Car - ModelT (?> \d* ) ;
}x;
```

```
use v6-pugs;
"ModelT2005" ~~ rule {
    Car - ModelT \d* ;
};
```

```
use v5;
"ModelT2005" =~ qr{
    Car \s* = \s* ModelT \s+ (?> \d* ) \s* ;
}x;
```

More Components

- Pugs::Grammar::Precedence
- Pugs::Grammar::MiniPerl6
- Pugs::Compiler::RegexPerl5

Module::Compile

Everyone

hates Spiffy

```
use v5;
use Spiffy -Base;

my sub private {
    "It's a private method here";
}

sub public {
    $self->$private;
}

sub new() {
    my $self = super;
    $self->init;
    return $self;
}
```

Too much Magic

YAML

used Spiffy

Test::Base

uses Spiffy

IO::All

uses Spiffy

Kwiki

uses IO::All

Ergo...

Everyone

hates Ingy

What's hateful about Spiffy?

It's a

Source Filter!

```
use v5;
use Filter::Simple sub {
    s{(^ sub \s+ \w+ \s+ \{ )}
     {$1\nmy $self = shift;\n}mgx;
}
```

Filter::Simple Bad

- Adds dependency
- Slows down startup
- Breaks perl -d
- Wrecks other Source Filters

We can fix it!

```
use v5;
use Filter::Simple sub {
    s{(^ sub \s+ \w+ \s+ \{ )}
     {$1\nmy $self = shift;\n}mgx;
}
```

```
use v5;
use Filter::Simple::Compile sub {
    s{(^ sub \s+ \w+ \s+ \{ )}
     {$1\nmy $self = shift;\n}mgx;
}
```

How does that work?

Little-known fact:

“use Foo”

looks for Foo.pmc

before Foo.pm

```
% echo 'print "Hello\n"' > Foo.pmc
% perl -MFoo -e1
Hello
```

Save precompiled result to *.pmc*...

...no filtering needed next time!

Module::Compile Good

- Free of user-side dependencies
- Fast startup time
- Debuggable source is all in .pmc
- Allows composable precompilers

```
package Foo;
use Module::Compile-base;

sub pmc_compile {
    my ($class, $source, $context) = @_;
    # Convert $source into $compiled_output...
    return $compiled_output;
}
```

Filter::Simple::Compile

```
# Drop-in replacement to Filter::Simple
package Acme::Y2K;
use Filter::Simple::Compile sub {
    tr/y/k/;
}
```

```
# It's lexical!
my $normal_code_here;
{
    use Acme::Y2K;
    pacyage Foo;
    mydir "tmp";
}
my $normal_code_there;
```

Filter::Macro

```
package MyHandyModules;
use Filter::Macro;

# lines below will be expanded into caller's code
use strict;
use warnings;
use Fatal qw( open close );
use FindBin qw( $Bin );
```

```perl
# In your code
package MyApp;
use MyHandyModules;
print "I'm invoked from $Bin";
```

```perl
# In your code
package MyApp;

use strict;
use warnings;
use Fatal qw( open close );
use FindBin qw( $Bin );

#line 3
print "I'm invoked from $Bin";
```

```
# Makefile.PL
use inc::Module::Install;

name 'MyApp';
all_from 'lib/MyApp.pm';

pmc_support;

WriteAll;
```

No dependency on

MyHandyModules.pm

Inline::Module

```
# Aww...
package MyApp;
use File::Slurp qw( slurp );
use HTTP::MessageParser;
```

```
# Yay!
package MyApp;
use Inline::Module 'File::Slurp' => qw( slurp );
use Inline::Module 'HTTP::MessageParser';
```

Zero

Dependencies

What about

Deploying Perl 6?

use v6-alpha;

v6.pm

(on CPAN now!)

Write Perl 6

compile to Perl 5

Source:

Rule.pm

```
use v6-pugs;

grammar Pugs::Grammar::Rule;
rule ws :P5 {
    ^((?:\s|\#(?-s:.)*)+)
}
# ...more rules...
```

Target:

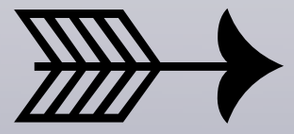
Rule.pmc

```
# Generated by v6 0 (Module::Compile 0.15) - do not edit!
#line 1 ##########((( 32-bit Checksum Validator )))###################
BEGIN { use 5.006; local (*F, $/); ($F = __FILE__) =~ s!c$!!; open(F)
or die "Cannot open $F: $!"; binmode(F, ':crlf'); unpack('%32N*',<F>)
== 0x1D6399E1 or die "Checksum failed for outdated .pmc file: ${F}c"}
#line 1 ##############################################################
package Pugs::Grammar::Rule;
use base 'Pugs::Grammar::Base';
*{'Pugs::Grammar::Rule::ws'} = sub {
    my $grammar = shift;
    #warn "rule argument is undefined" unless defined $_[0];
    $_[0] = "" unless defined $_[0];
    my $bool = $_[0] =~ /^((?:\s|\#(?-s:.)*)+)(.*)$/sx;
    return {
        bool => $bool,
        match => $1,
        tail => $2,
        #capture => $1,
    }
};
# ...more rules...
```

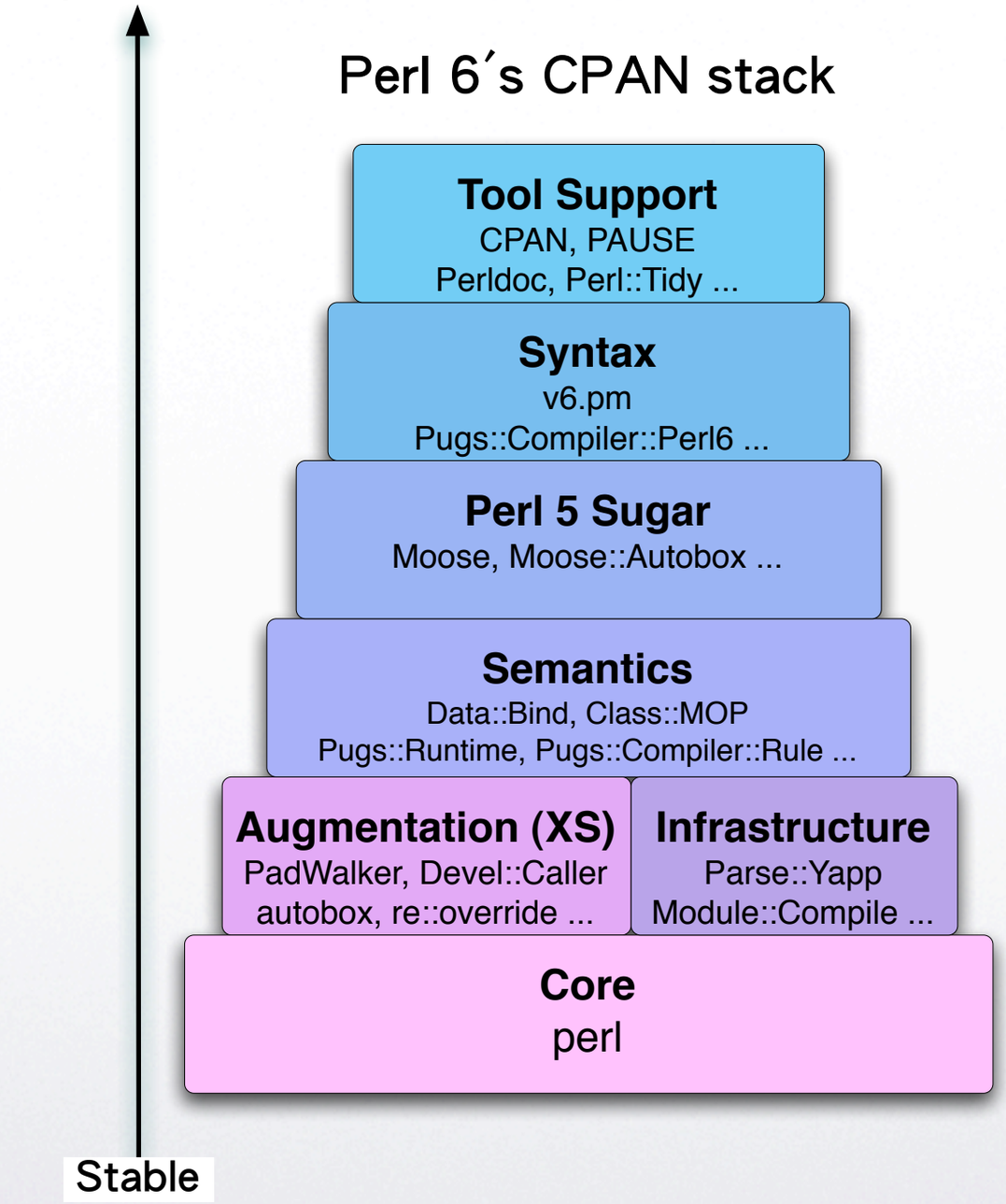
"CPAN is the language, Perl 6 is just a better syntax sugar!"

Still needs work!

In Progress



Experimental



Stable

Native Type Bridge

Moose::Autobox

Builtin Objects

*Pugs::Runtime::**

Calling Convention

Data::Bind

Multi Dispatch

Sub::Multi

Even More Sugar

re::override

Translators

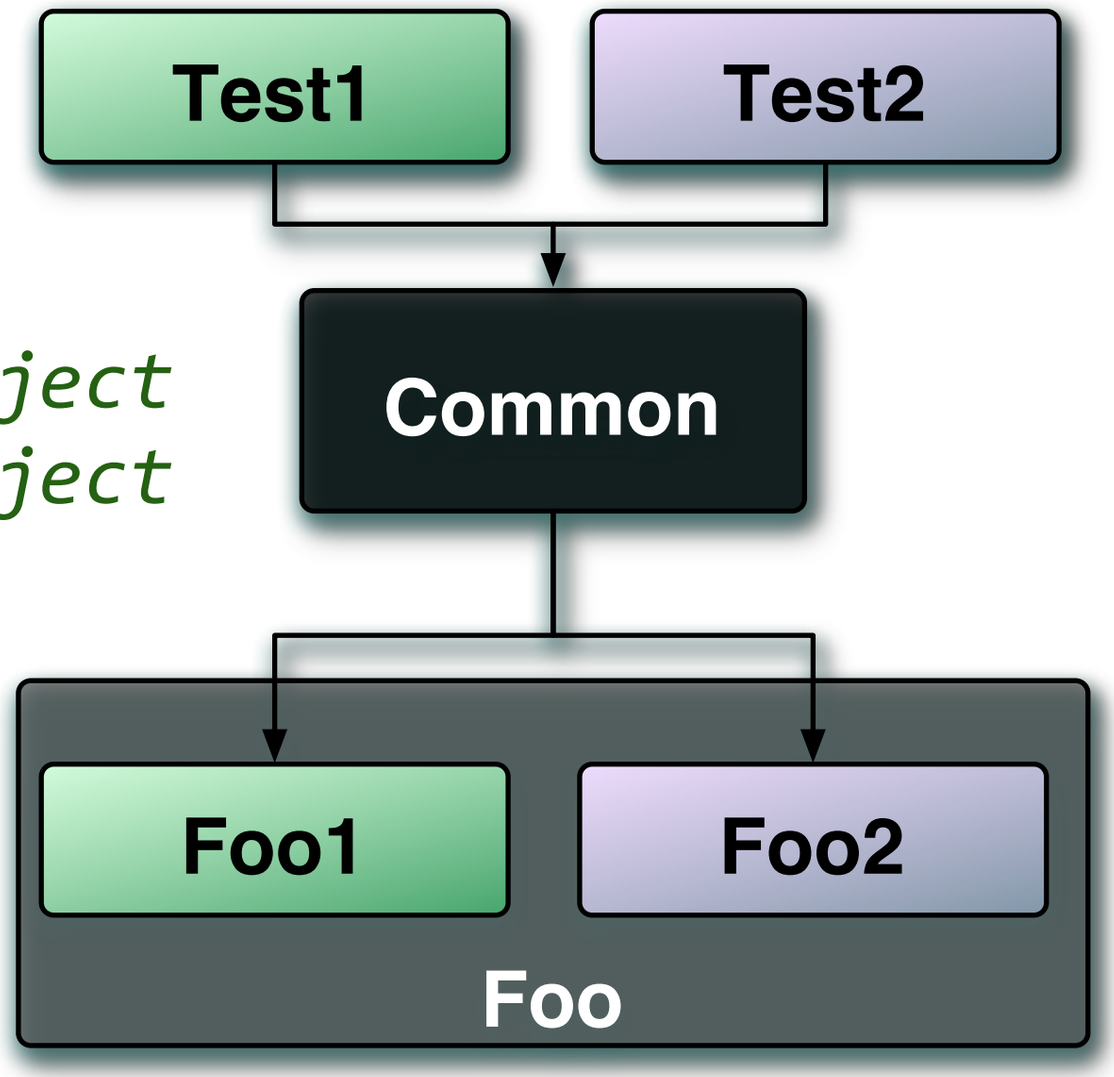
MAD ➠ Perl6

Multiversioning

only.pm

```perl
# The use case
use v5;
package Foo1; sub new { bless {}, 'Foo1' }
package Foo2; sub new { bless {}, 'Foo2' }
package Test1; sub t1 { Common->foo }
package Test2; sub t2 { Common->foo }
package Common; sub foo { Foo->new }
```

```perl
package main;
warn Test1::t1(); # Foo1 object
warn Test2::t2(); # Foo2 object
```



```
# The implementation
use v5;
BEGIN { $^P |= 0x01 }

my %dep_map = (
    Test1 => { Foo => 'Foo1' },
    Test2 => { Foo => 'Foo2' },
);
```

```
sub DB::sub {
    my $cls = (split(/::/, $DB::sub, 2))[0];
    while (my ($k, $v) = each %{ $dep_map{$cls} }) {
        %{"$k\::"} = %{"$v\::"};
    }
    goto &$DB::sub;
}
```

Commits

Welcome!

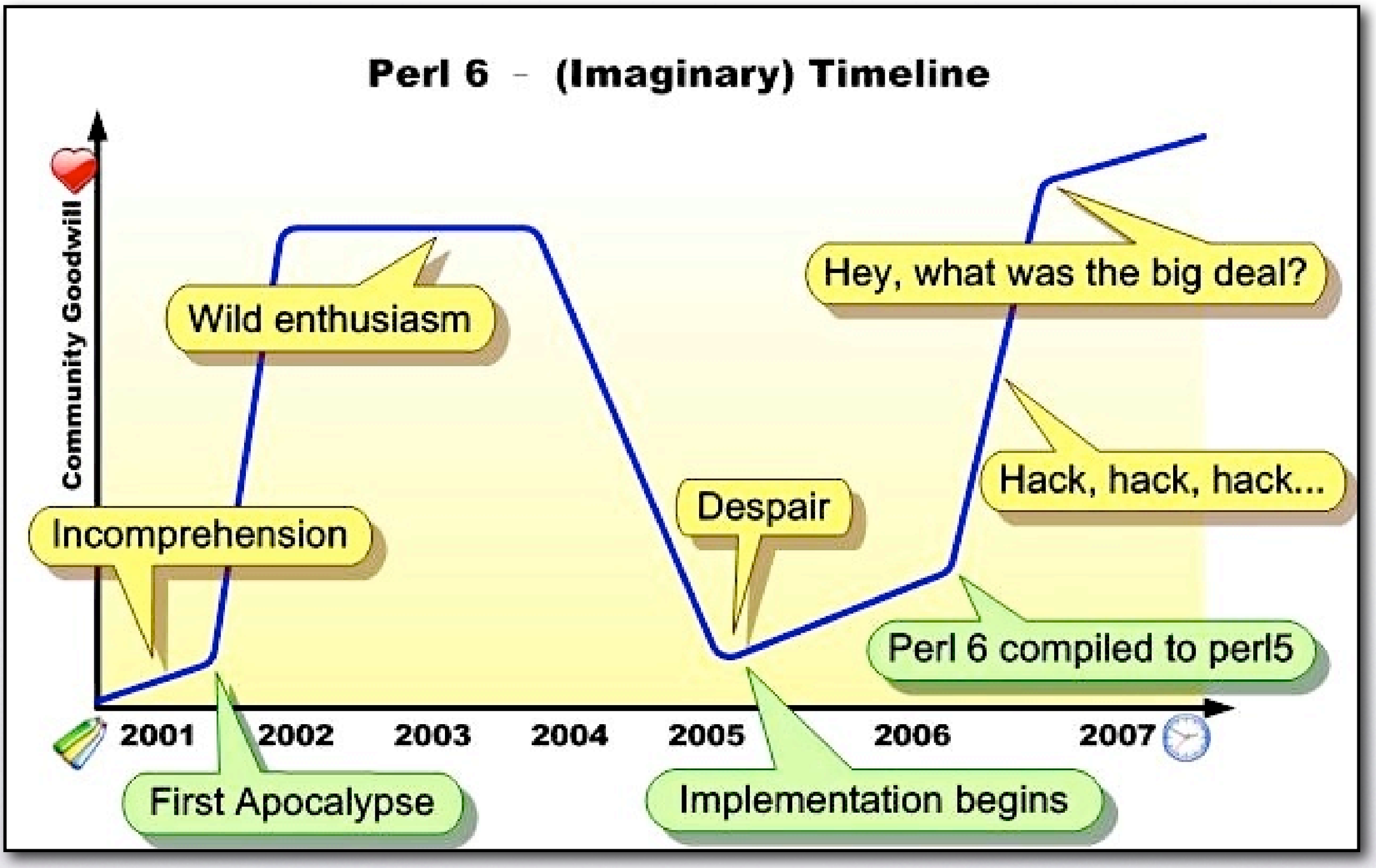
When will Perl 6 be released?

(

By Christmas!

When Perl 6 is out, every day will be like Christmas!

The Advent starts Now


Perl 6 - (Imaginary) Timeline
Community Goodwill
Wild enthusiasm
Incomprehension
First Apocalypse
Despair
Implementation begins
Hey, what was the big deal?
Hack, hack, hack...
Perl 6 compiled to perl5
2001
2002
2003
2004
2005
2006
2007

Thank you!