

Deploying Perl 6

Audrey Tang

**Perl 6
is here Today!**

Old News

Pugs & Parrot

**Great for
experimenting**

**But not
for production**

**...not until
Christmas**



**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 ~~);

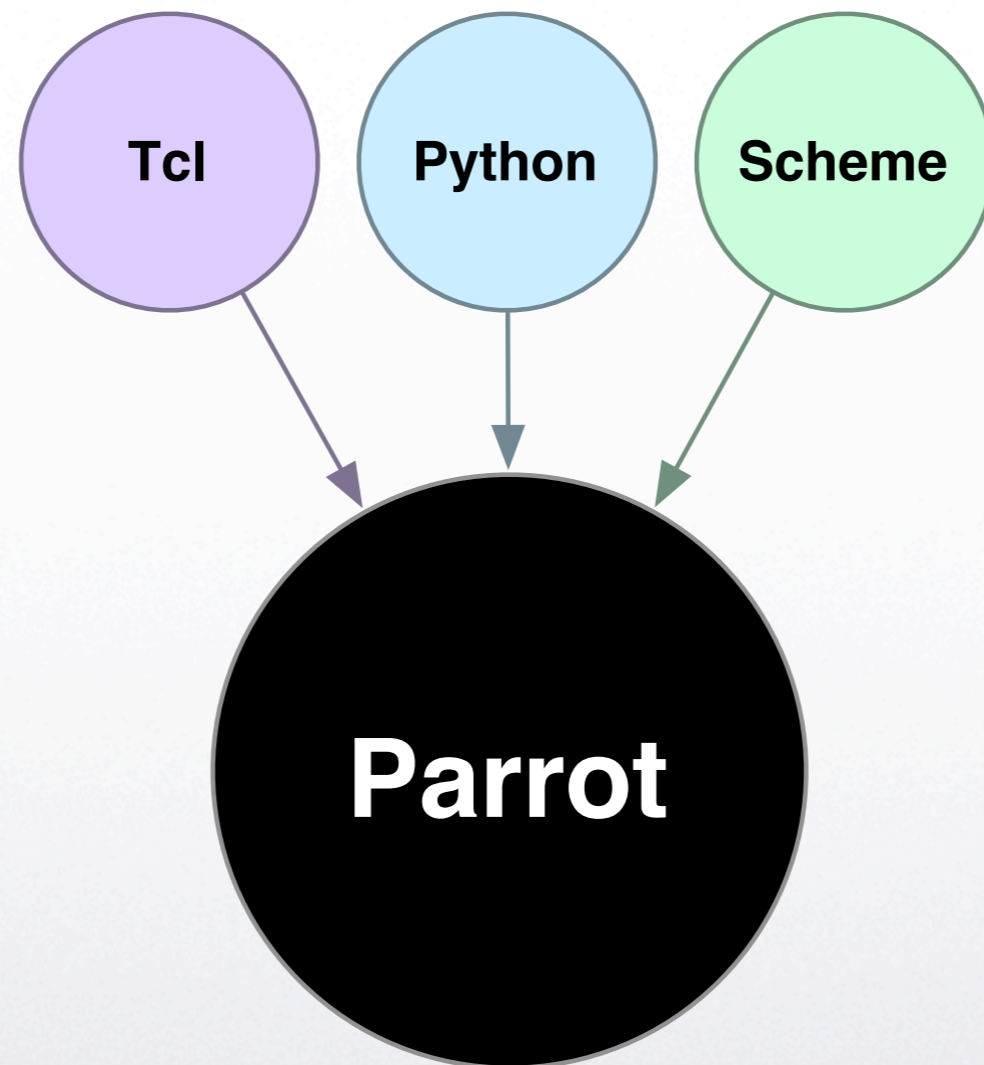
given (shift()) {
    when ['-h', '--help'] {
        say "Usage: $0";
    }
    default {
        $0 ~~ 'moose.exe' err die "Not Moose";
    }
}
```

**How to get Perl 6
into Production?**

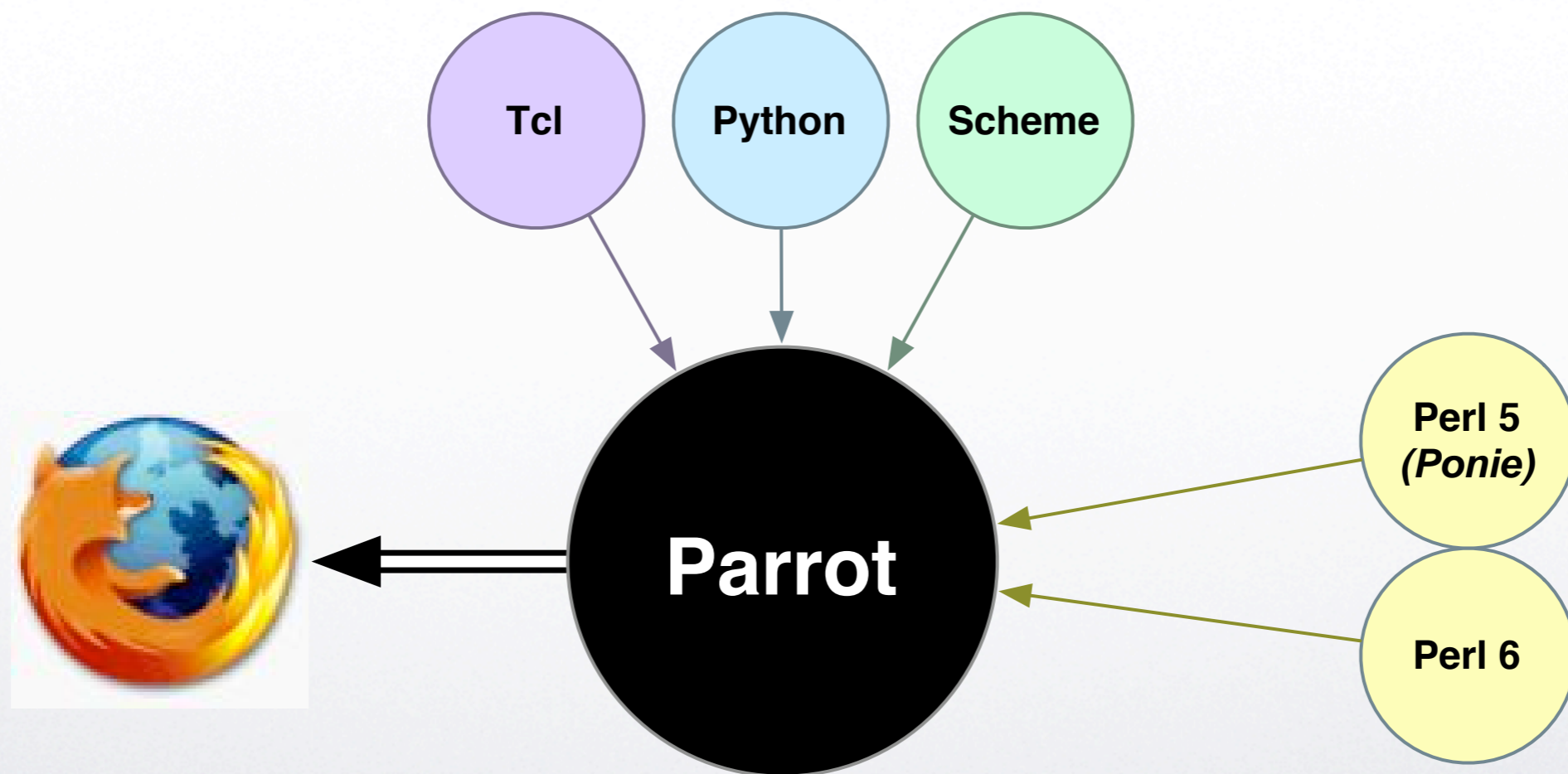
Production

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

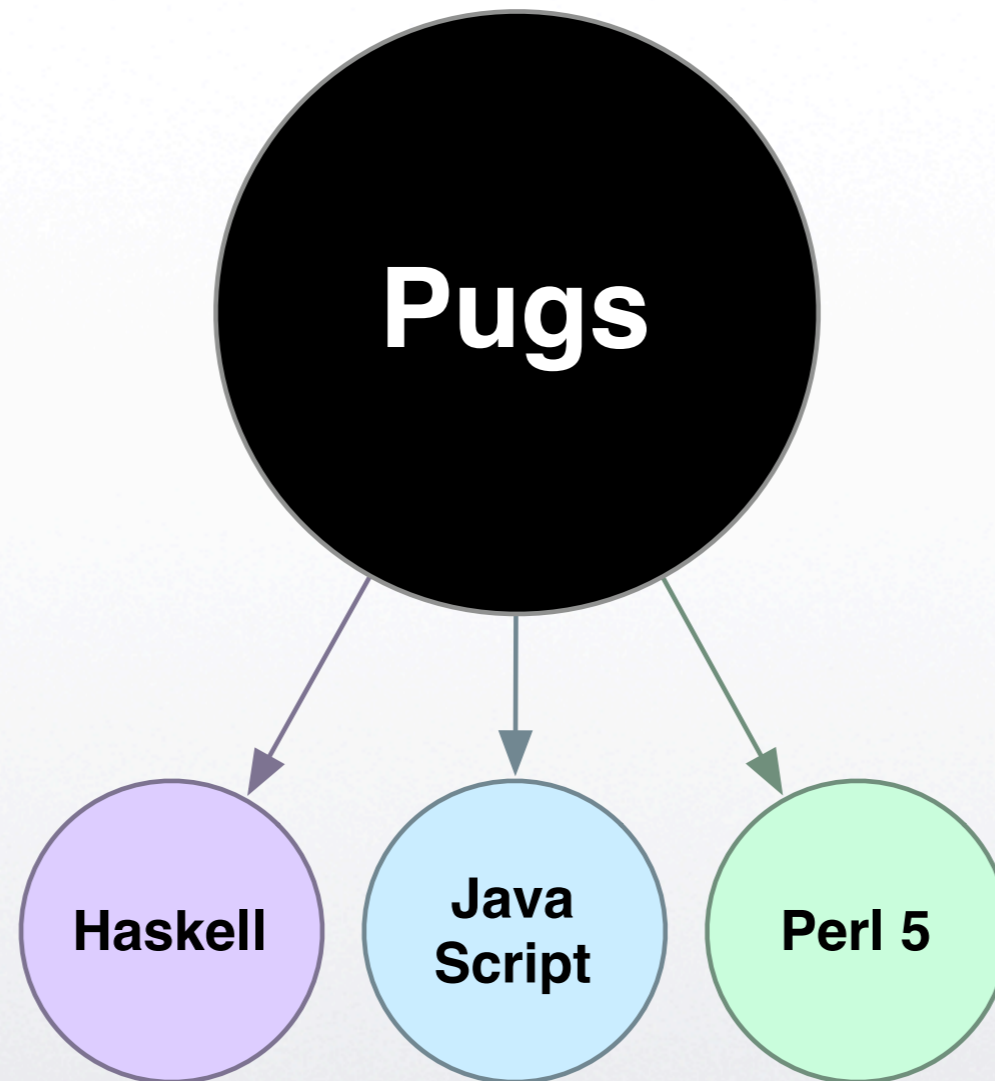
Frontends?



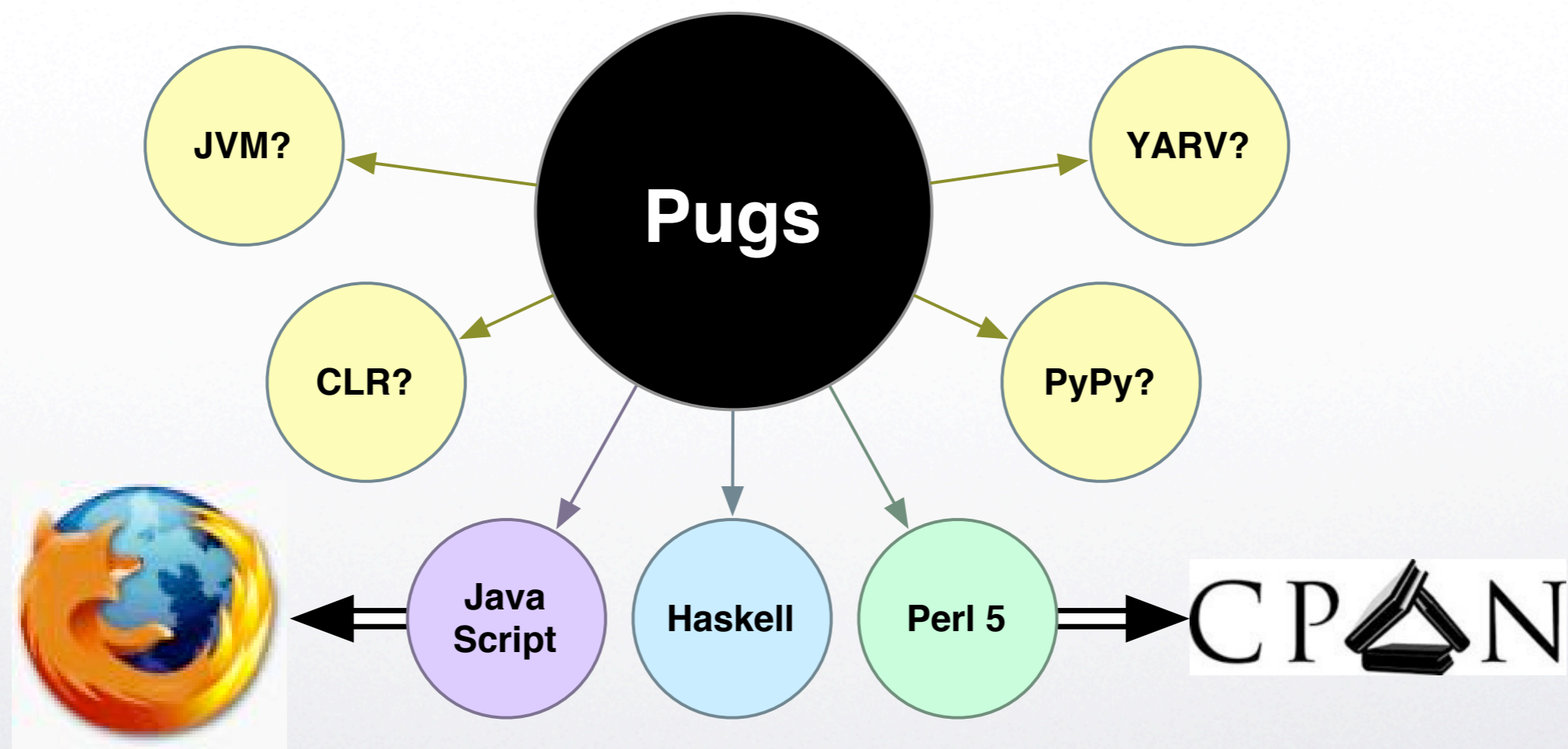
Frontends?



Backends!



Backends!



Pugs's Perl 5 Backend

Perl 6 Runtime Implemented in Perl 5

Sane Perl 5 (*not* source filters)

**Available
On CPAN
Today**

Moose.pm



Moose, it's the new Camel



Moose Fixation

Objects With Class

```
use v6-pugs;  
class Point;
```

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

```
method clear () {
```

```
    $.x = 0; # accessible within the class  
    $.y = 0;
```

```
}
```

```
use v5;  
package Point;  
use Moose;  
  
has x => (is => 'rw');  
has y => (is => 'ro');  
  
sub clear {  
    my $self = shift;  
  
    $self->{x} = 0;  
    $self->y(0);  
}
```

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;  
};
```

Constraints

```
use v6-pugs;  
class BankAccount;  
  
has Int $.balance is rw = 0;  
  
method deposit ($amount) {  
    $.balance += $amount;  
}  
  
method withdraw ($amount) {  
    my $current_balance = $.balance;  
    ($current_balance >= $amount)  
        err fail "Account overdrawn";  
    $.balance = $current_balance - $amount;  
}
```

```
use v5;
package BankAccount;
use Moose;

has balance => (
    isa => 'Int', is => 'rw', default => 0
);
sub deposit {
    my ($self, $amount) = @_;
    $self->balance($self->balance + $amount);
}

sub withdraw {
    my ($self, $amount) = @_;
    my $current_balance = $self->balance;
    ($current_balance >= $amount)
        or die "Account overdrawn";
    $self->balance($current_balance - $amount);
}
```

```
use v6-pugs;  
class CheckingAccount;  
  
is BankAccount;  
  
has BankAccount $.overdraft_account is rw;  
  
method withdraw ($amount) {  
  
    my $overdraft = $amount - $.balance;  
    if ($.overdraft and $overdraft > 0) {  
        $.overdraft_account.withdraw($overdraft);  
        $.deposit($overdraft);  
    }  
    call;  
};
```

```
use v5;
package CheckingAccount;
use Moose;
extends 'BankAccount';

has overdraft_account => (
    isa => 'BankAccount', is => 'rw'
);
before withdraw => sub {
    my ($self, $amount) = @_;
    my $overdraft = $amount - $self->balance;
    if ($self->overdraft_account and $overdraft > 0) {
        $self->overdraft_account->withdraw($overdraft);
        $self->deposit($overdraft);
    }
};
```

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"
```

Subtypes

```
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'}/
    },
);

```

More features

- **Roles**
- **Coercion**
- **Metaclasses**

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

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;
```

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 it work?

Little-known fact:

“use Foo”

looks for Foo.pmc

before Foo.pm

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

**Save filtered 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!
{
    use Acme::Y2K;
    pacyage Foo;
    mydir "tmp";
}
my $normal_code_here;
```

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 );
```

```
# In your code  
package MyApp;  
use MyHandyModules;  
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-pugs;

v6.pm

**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 file - do not edit!
#####((( 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"}
#####
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...

```

Still needs work!

In Progress

Intrinsic Objects

Moose::Autobox

Builtin Objects
Pugs::Runtime::*

Calling Convention

Data::Bind

Even More Sugar
re::override

Translators

MAD Perl

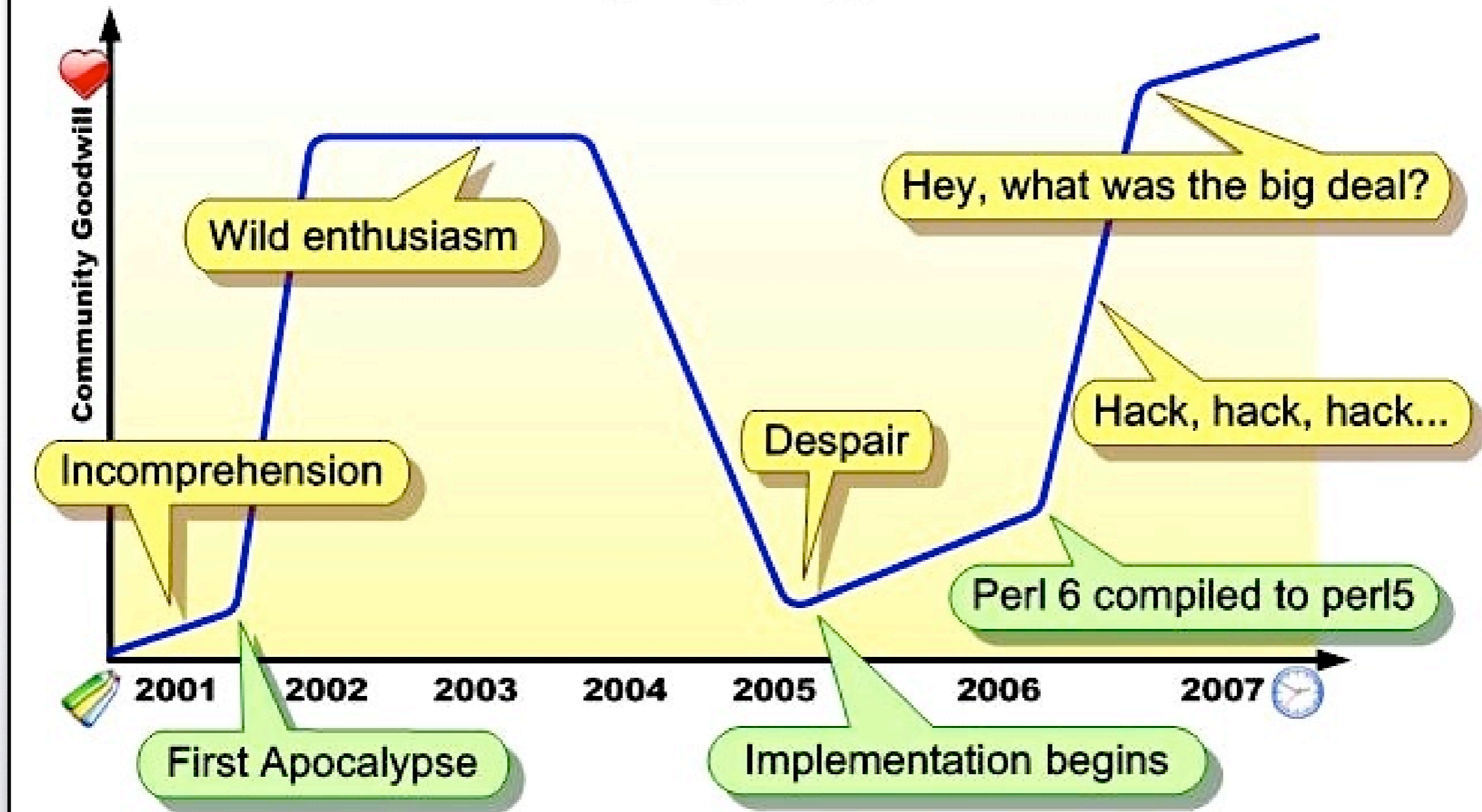
**Multiversioning
only.pm**

CPAN Toolchain

JIB.pm

Commits welcome!

Perl 6 - (Imaginary) Timeline



**When will
Perl 6 be released?**



By Christmas!

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



Thank you!

