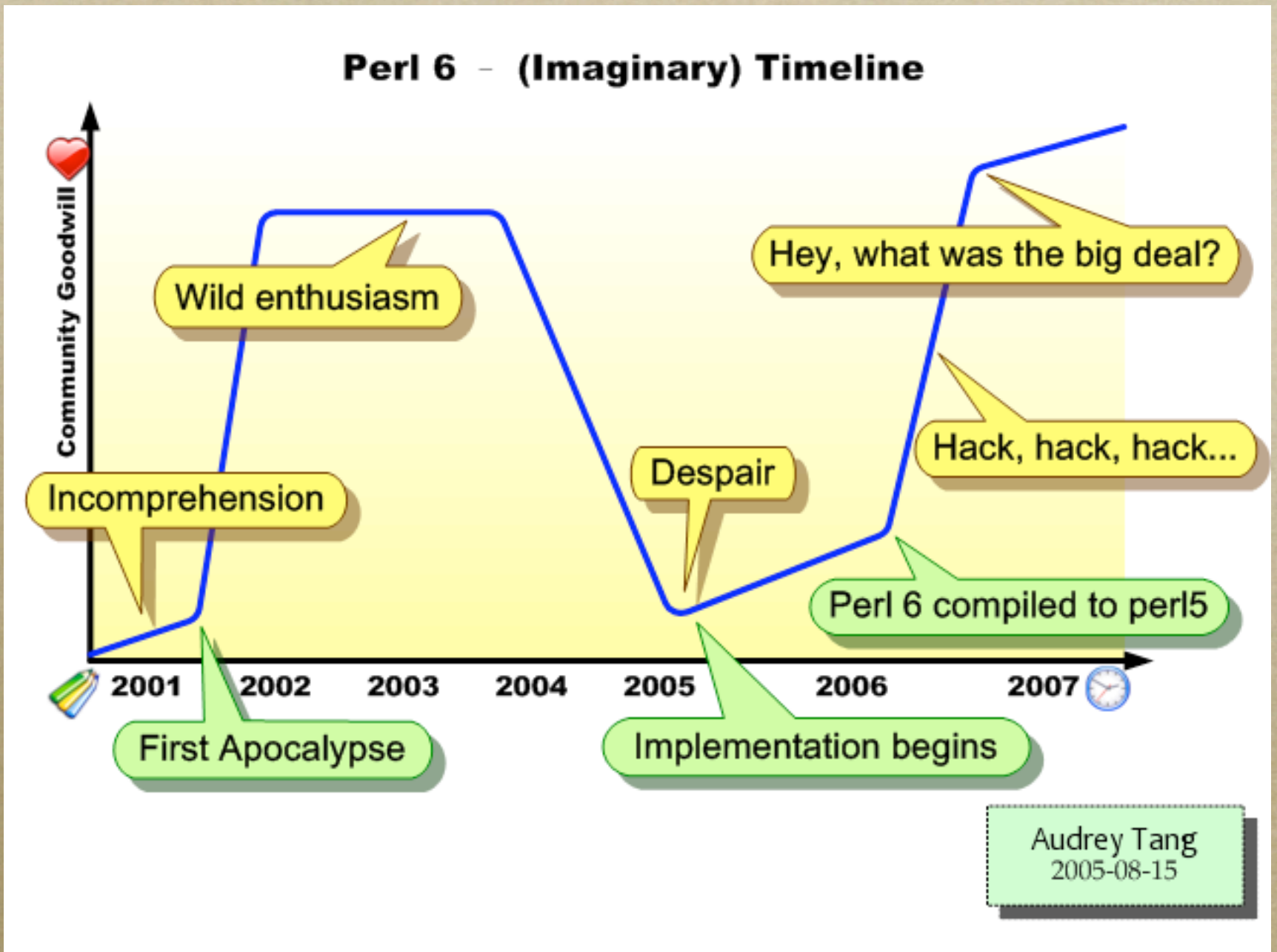
Perl 6

An overview

Damian on Perl 6


Perl 6 - (Imaginary) Timeline
Community Goodwill
Wild enthusiasm
Incomprehension
Despair
Hey, what was the big deal?
Hack, hack, hack...
Perl 6 compiled to perl5
2001
2002
2003
2004
2005
2006
2007
First Apocalypse
Implementation begins
Audrey Tang
2005-08-15

Pugs

- *Started by Audrey Tang in February 2005*
- *It's written is Haskell*

Pugs

- *Why Haskell*
  - *Shares many of the features Perl 6 will have.*
    - *Lazy list evaluation*
    - *Parsec is very similar to Perl 6 rules*
    - *Continuations*

Haskell Example

- `fibs :: [Int]`
- `fibs = 0 : 1 : [ a + b | (a, b) <- zip fibs (tail fibs)]`

- `take 10 fibs`
- For more information
  - Haskell Tutorial for C Programmers

Perl 6

- *Hello, world in Perl 6*
  - `say “Hello, world”;`
  - *or*
  - `print “Hello, world\n”;`
  - *or*
  - `“Hello, world”.say`

Perl 6

- *Some things are the same.*
  - *$scalar*
  - *@array*
  - *%hash*

Perl 6

- *Some things are different*
  - *@array[0]*
  - *%hash{'key'}*
  - *$ref = @array*
    - *$ref[0]*

Arrays

- *my @array = (5,6,7,9);*
- *my $count = @array.elems;*
- *my $last = @array.last;*

Hashes

- *Hashes are a list of pairs*
  - *$pair = 'key' => 'value';*
    - *=> is the pair constructor*
    - *Can be written*
      - *:key<value>;*

Hashes

- *my %hash = (‘first’ => ‘one’, ‘second’ => ‘two’);*
  - *my @pairs = %hash;*
  - *say @pairs[0][1]; # one*
  - *%hash.keys*
  - *%hash.values*
  - *%hash.kv # flattens to list*

Types

- *Perl 6 has an optional type system*
  - *my Int $integer;*
  - *my Str @strings;*

Control Structures

- *IF statements*
  - *if $balance < 5.00 {*
    say 'Balance is low';
    }
- *Parentheses are optional for conditions*

Control Structures

- *Perl 6 switch statement*
- *Given/When*
- *given $action {*
  when $_ ~~ Web::Page::Object { $blue.render };
  when $_ ~~ 'back' { $server.back; }
  default { say 'No match'; }
  };
- *~~ is the Smart Match operator*

Loops

- *loop 'loop'*
  - *loop {*
    say $a++;
    brake if $a >= 10;
    }
    loop (my $x = 0; $x < 10; $x++) {
    say $x;
    }
- *while 'loop'*
  - *while $count < 10 {*
    say $count;
    $count++;
    }

Loops

- *For 'loop'*
  - *for @stuff {*
    say $_;
    }
  - *for @stuff -> $item {*
    say $item;
    }
  - *for (@list1, @list2) -> $item1, $item2 {*
    say $item1 ~ ' ' ~ $item2;
    }
  - *for @even -> $one, $two { say $one ~ ' ' ~ $two }*

Loops

- *Property blocks*
  - *for 1..4 {*
    NEXT { print " potato, "; }
    LAST { print '.'; }
    print;
    }
    - *# 1 potato, 2 potato, 3 potato, 4.*
  - *for 5..7 {*
    my $potato = "$count potato, ";
    NEXT { print $potato; }
    LAST { print $potato, 'more'; }
    }
    - *# 5 potato, 6 potato, 7 potato, more.*

Subroutines

- *They can be used just like Perl 5*
  - *@_ based*
- *sub name($first_name, $last_name) {... }*
- *If you use the signature the argument won't flatten*
  - *sub complex(@array, %hash) { ... }*
- *Placeholder Variables*
  - *my $sub = { $^a.say; $^b.say; } # unicode sorted*
  - *$sub('test', 'test2');*
  - *# test*
    # test2

Objects

- *Quick object example*
  - *class Name {*

```
    has $.first_name is rw;
    has $.last_name is rw;
    method full_name($self:) {
        say $.first_name ~ ' ' ~ $.last_name;
    }
}

my $name = Name.new;
$name.first_name = 'Robert';
$name.last_name = 'Boone';
$name.full_name; # Robert Boone
```

Code

- *Perl 5 to Perl 6 comparison*

Quick Sort

- *Perl 5*
- ```
  sub qsort {
      @_ or return ();
      my $p = shift;
      (qsort(grep $_ < $p, @_), $p, qsort(grep $_ >= $p, @_));
  }
  ```
  ```

Quick Sort

- *Perl 6*
  - *multi quicksort () { () }*
  - *multi quicksort (*$x, *@xs) {*

```
    my @pre  = @xs.grep:{ $_ <  $x };
    my @post = @xs.grep:{ $_ >= $x };
    (@pre.quicksort, $x, @post.quicksort);
}
```

Quick Sort

- *Haskell*
- *sort :: (Ord a) => [a] -> [a]*
- *sort [] = []*
- *sort (pivot:rest) = sort [y | y <- rest, y < pivot]*
  ++ [pivot] ++
  sort [y | y <- rest, y >=pivot]

Observer Pattern

- The essence of this pattern is that one or more objects (called observers or listeners) are registered (or register themselves) to observe an event which may be raised by the observed object (the subject). (The object which may raise an event generally maintains a collection of the observers.) - from Wikipedia
- Perl 5
- Perl 6

File access

- *Perl 6*

More features

- *Lisp like macros*
- *Grammars and rules to replace regular expressions - OO regexes*
- *Junctions - any, all, none*

Software

- *Pugs - http://www.pugscode.org/*
- *Perl 6 - http://dev.perl.org/perl6/*
- *Perl6::Bible*
- *Wikipedia*

Perl 6

- *There is a lot I didn't cover.*
- *Thing change everyday.*