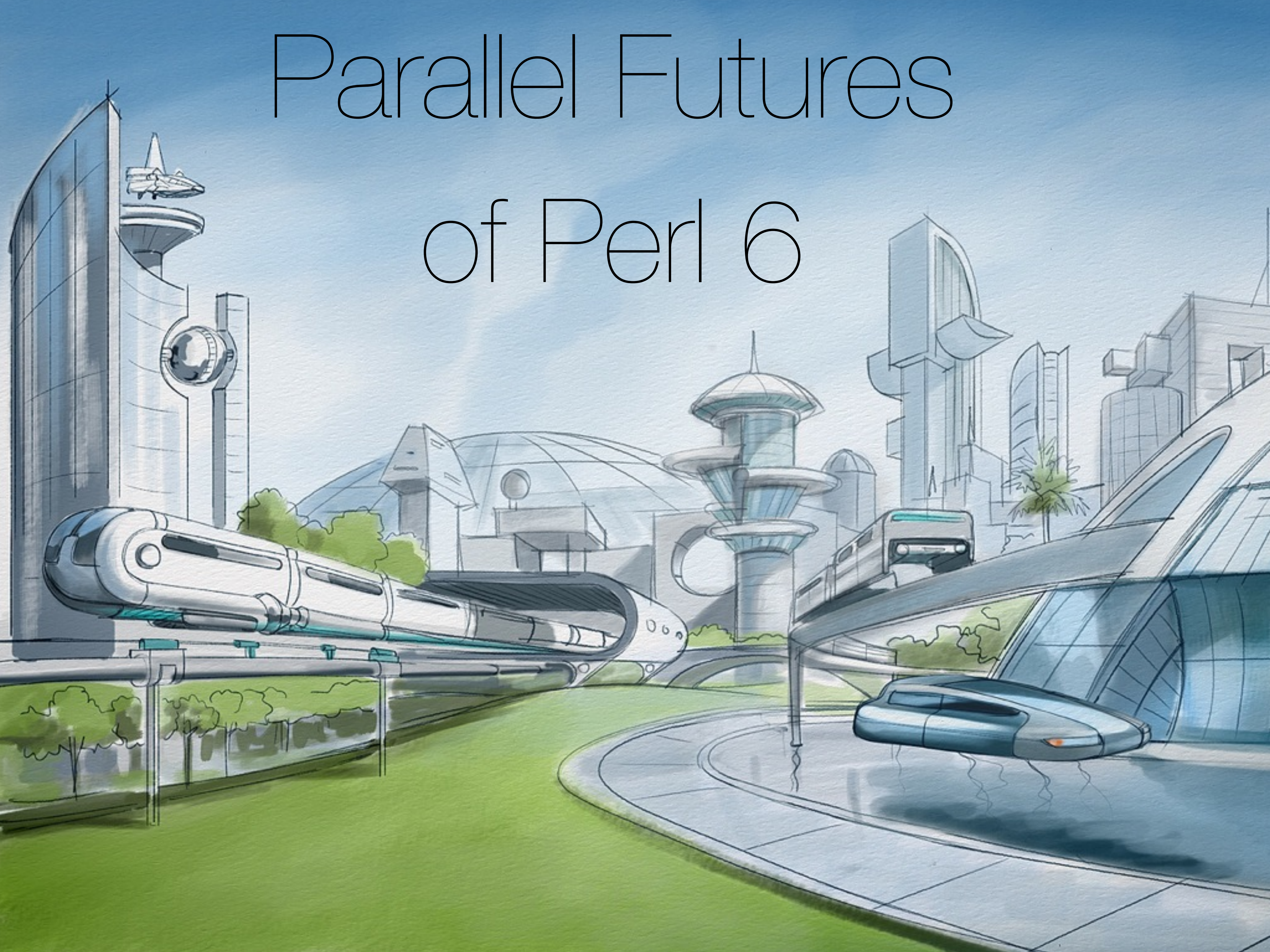
Perl 6 for Concurrency and Parallel Computing

or

Parallel Features of Perl 6

The Parallel Features of the Perl Six

Parallel Futures of Perl 6



Foreword

Interviews for Pragmatic Perl in 2013–2015

PRAGMATIC PERL

26

04/2015

pragmaticperl.com

Q: What is the most important feature of the programming languages in the future?

No idea (2 answers)

A: I don’t know

No idea (2 answers)

A: There’s no good answer

Syntax features (1/3)

A: Natural-like language

Syntax features (2/3)

A: Minimalism

Syntax features (3/3)

A: Extendability

Object system (1/3)

A: Flexible type casting

Object system (2/3)

A: Robustness

Object system (3/3)

A: Built-in introspection

Environment (1/4)

A: JVM support

Environment (2/4)

A: Execution in a browser

Environment (3/4)

A: Language inter-compatibility

Environment (4/4)

A: Embedding

Humanity (1/8)

A: Community

Humanity (2/8)

A: Humanism

Humanity (3/8)

A: Open source

Humanity (4/8)

A: Pragmatism

Humanity (5/8)

A: Mind control *(sic!)*

Humanity (6/8)

A: Expressing things easily

Humanity (7/8)

A: Domain oriented

Humanity (8/8)

A: Unobtrusiveness

Number 1 answer

Parallelism

Parallelism (1/12)

A: Parallelism

Parallelism (2/12)

A: Working with parallel resources

Parallelism (3/12)

A: Parallelism

Parallelism (4/12)

A: Good paralleling model

Parallelism (5/12)

A: Intuitive coroutines and multi-core support

Parallelism (6/12)

A: Parallelism

Parallelism (7/12)

A: Safe operation parallelism

Parallelism (8/12)

A: Built-in threading

Parallelism (9/12)

A: Qualitative abstract threading

Parallelism (10/12)

A: Parallelism

Parallelism (11/12)

A: Good parallelism

Parallelism (12/12)

A: Multi-tasking

Back to Perl 6

The idea is keeping things transparent

A Perl 6 user simply uses concurrency

A Perl 6 compiler makes it possible

A Perl 6 compiler makes it possible

The Perl 6 compiler makes it possible

Running examples with Rakudo Star

Running examples with Rakudo Star on MoarVM

Two kinds of parallel features

Roughly,

1) implicit
2) explicit

Operators

at a glance

1. Hyperops

A hyper operator

is a

meta operator

+

operator

+=

meta operator

>>+<<

hyper operator

»+«

hyper operator

>>+

hyper operator

»+

hyper operator

```
>>+>>
```

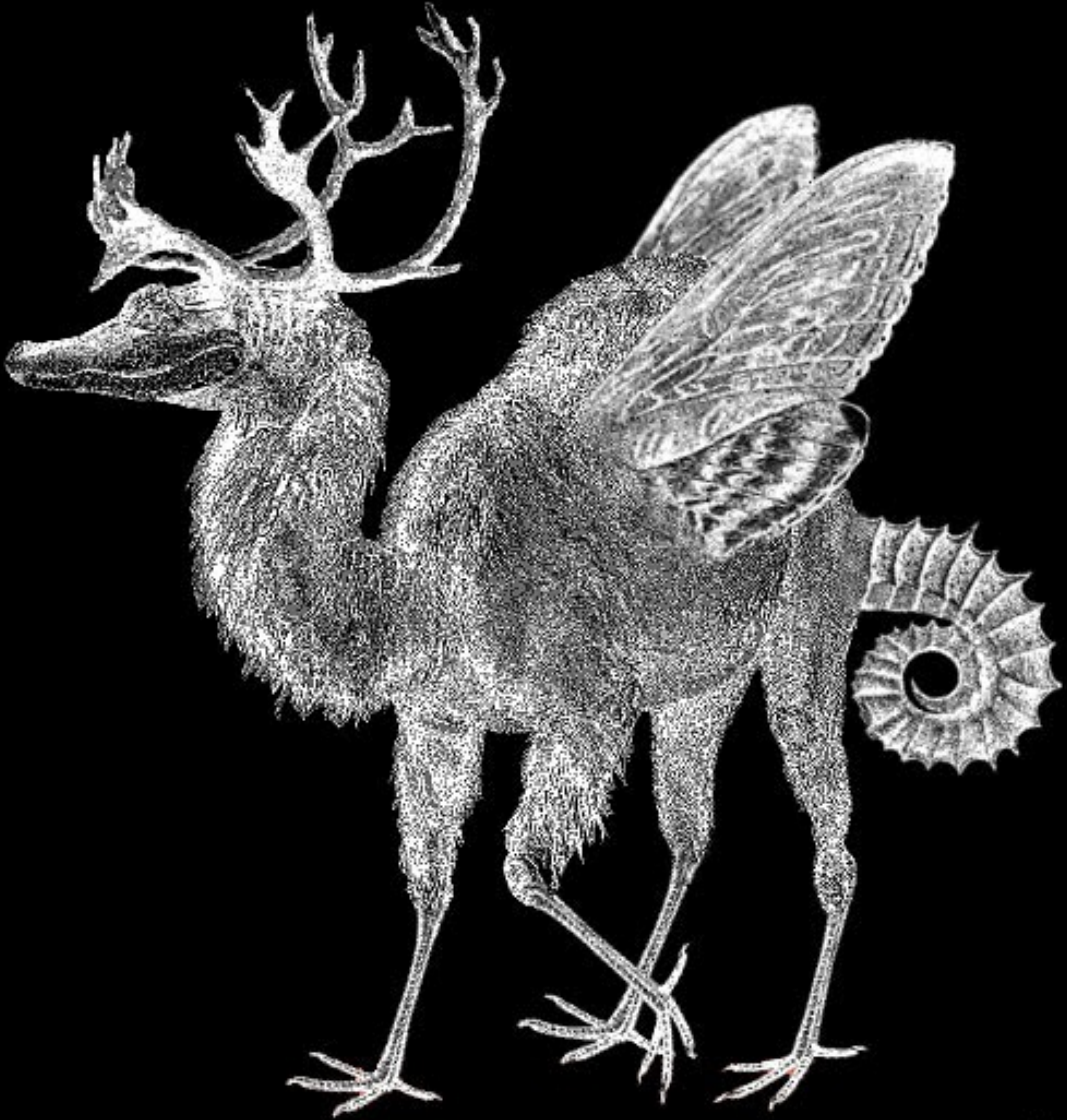
hyper operator

<<+<<

hyper operator

«+«

hyper operator



```
<<+>>
```

hyper operator

«+»

hyper operator

<<>>

hyper operator

«<»

hyper operator

```
@c = @a >>+<< @b
```

```
@c = @a >>+<< @b

@c[0] = @a[0] + @b[0];
```

```
@c = @a >>+<< @b

@c[0] = @a[0] + @b[0];
@c[1] = @a[1] + @b[1];
```

```
@c = @a >>+<< @b

@c[0] = @a[0] + @b[0];
@c[1] = @a[1] + @b[1];
@c[2] = @a[2] + @b[2];
```

```
@c = @a >>+>> 1
```

```
@c = @a >>+>> 1

@c[0] = @a[0] + 1;
```

```
@c = @a >>+>> 1

@c[0] = @a[0] + 1;
@c[1] = @a[1] + 1;
```

```
@c = @a >>+>> 1

@c[0] = @a[0] + 1;
@c[1] = @a[1] + 1;
@c[2] = @a[2] + 1;
```

2.
Junctions

Or

Quantum

Superpositions

Many values as one

```
my $j = 1 | 2 | 3 | 5;
my $j = 1 | 2 | 3 | 5;

say 1 if 3 == $j;
```

```
my $j = 1 | 2 | 3 | 5;

say 1 if 3 == $j;
```

```
my $j = 1 | 2 | 3 | 5;

say 1 if 3 == $j;
```

```
my $j = 1 | 2 | 3 | 5;

say 1 if 3 == $j;
```

```
my $j = 1 | 2 | 3 | 5;

say 1 if 3 == $j;
```

```
1
```

3.

Feeds

```
my @a = 1..10;
my @a = 1..10;
@a ==> grep {$_ mod 2};
```

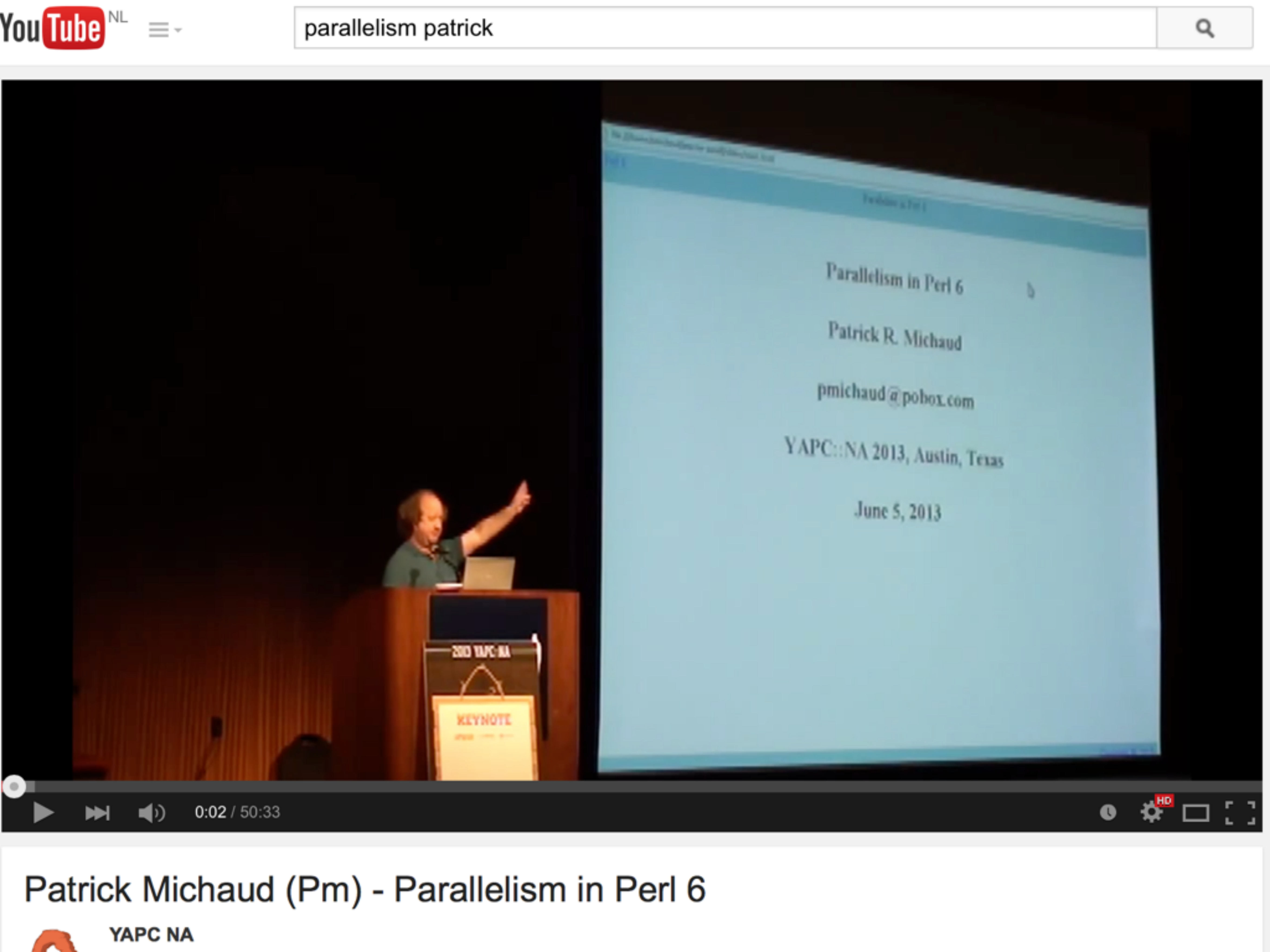
```
my @a = 1..10;
@a ==> grep {$_ mod 2};
```

```
1 3 5 7 9
```

```
my @a = 1..10;
@a
   ==> grep {$_ mod 2}
   ==> map {$_ ** 2};
```

```
my @a = 1..10;
@a
   ==> grep {$_ mod 2}
   ==> map {$_ ** 2};
```

```
1 9 25 49 81
```



Parallelism in Perl 6

Patrick R. Michaud

pmichaud@pobox.com

YAPC::NA 2013, Austin, Texas

June 5, 2013

Patrick Michaud (Pm) - Parallelism in Perl 6

YAPC NA

4.

Channels

```
my $c = Channel.new;
my $c = Channel.new;
$c.send(42);
```

```
my $c = Channel.new;
$c.send(42);
say $c.receive;
```

```
42
```

```
my $ch = Channel.new;
my $ch = Channel.new;
for <1 3 5 7 9> {
    $ch.send($_);
}
```

```
my $ch = Channel.new;
for <1 3 5 7 9> {
    $ch.send($_);
}
while $ch.poll -> $x {
    say $x;
}
```

5.
Promises

```
my $p = Promise.new;
my $p = Promise.new;
say $p.status;
```

```
Planned
```

```
my $p = Promise.new;
$p.keep;
```

```
my $p = Promise.new;
$p.keep;
say $p.status;
```

```
Kept
```

```
my $p = Promise.new;
$p.break;
```

```
my $p = Promise.new;
$p.break;
say $p.status;

Broken
```

Factory methods

start

```
my $p = start {42};
my $p = start {42};
say $p.WHAT;
```

```
(Promise)
```

```
my $p1 = start {sleep 2};
my $p1 = start {sleep 2};
my $p2 = start {sleep 2};
my $p1 = start {sleep 2};
say $p1.status;
my $p2 = start {sleep 2};
say $p2.status;
```

```
my $p1 = start {sleep 2};
say $p1.status;
my $p2 = start {sleep 2};
say $p2.status;
```

```
Planned
Planned
```

```
my $p1 = start {sleep 2};
my $p2 = start {sleep 2};
sleep 3;
```

```
my $p1 = start {sleep 2};
my $p2 = start {sleep 2};
sleep 3;
say $p1.status;
say $p2.status;
```

```
my $p1 = start {sleep 2};
my $p2 = start {sleep 2};
sleep 3;
say $p1.status
say $p2.status
Kept
Kept
```

`start`

in a thread

in

```
my $p = Promise.in(3);
my $p = Promise.in(3);

for 1..5 {
    say "$_ {$p.status}";
    sleep 1;
}
```

```
my $p = Promise.in(3);

for 1..5 {
    say "$_ {$p.status}";
    sleep 1;
}
```

1 Planned

```
1 Planned
2 Planned
```

```
1 Planned
2 Planned
3 Planned
```

```
1 Planned
2 Planned
3 Planned
4 Kept
```

```
1 Planned
2 Planned
3 Planned
4 Kept
5 Kept
```

Example: Sleep Sort

```
@*ARGS
```

```
for @*ARGS -> $a {




}
```

```
for @*ARGS -> $a {

        Promise.in($a)



}
```

```
for @*ARGS -> $a {

        Promise.in($a).then({
            say $a;
        })

}
```

```
my @promises;
for @*ARGS -> $a {
    @promises.push(
        Promise.in($a).then({
            say $a;
        })
    );
}
```

```
my @promises;
for @*ARGS -> $a {
    @promises.push(
        Promise.in($a).then({
            say $a;
        })
    );
}

await(|@promises);
```

```
my @promises;
for @*ARGS -> $a {
    @promises.push(
        Promise.in($a).then({
            say $a;
        })
    );
}

await(|@promises);
```

```
$ ./sleep-sort.pl
$ ./sleep-sort.pl 3 1 2
$ ./sleep-sort.pl 3 1 2
1
```

```
$ ./sleep-sort.pl 3 1 2
1
2
```

```
$ ./sleep-sort.pl 3 1 2
1
2
3
```

Home work: Channels inside Promises

More: Schedulers

More:
Suppliers

More:
I/O and Suppliers

More:
Signals

More:
Threads

More:
Atomic

More:
Locks

More:
Semaphores

__END__

Andrew Shitov
andy@shitov.ru

April 2015