Rakudo Perl 6 and Parrot

Jonathan Worthington

Linuxwochenende 2008

Rakudo Perl 6 and Parrot

Me

- From England

Me

- From England
  - And yes, I do like tea…

Rakudo Perl 6 and Parrot

Me

- From England
  - And yes, I do like tea…
  - …but I prefer it without milk in.

Me

- From England
  - And yes, I do like tea…
  - …but I prefer it without milk in.
- Currently living in Bratislava, Slovakia.

Me

- From England
  - And yes, I do like tea…
  - …but I prefer it without milk in.
- Currently living in Bratislava, Slovakia
  - Just an hour from here
  - Like Vienna, it's beautiful…

Me

- From England
    - And yes, I do like tea…
    - …but I prefer it without milk in.
- Currently living in Bratislava, Slovakia
    - Just an hour from here
    - Like Vienna, it's beautiful…
    - …but beer only costs 1 euro a pint ☺

Me

- From England
  - And yes, I do like tea…
  - …but I prefer it without milk in.
- Currently living in Bratislava, Slovakia
  - Just an hour from here
  - Like Vienna, it's beautiful…
  - …but beer only costs 1 euro a pint ☺
  - Er, half litre

My Talk

- An overview of three technologies
  - The Parrot VM – a virtual machine for dynamic languages
  - The Parrot Compiler Toolkit (= PCT) – a tool chain for rapidly developing compilers targeting Parrot
  - Rakudo, a Perl 6 implementation on Parrot built using PCT

Parrot

Parrot Is A Virtual Machine

- Virtual instruction set
  - Hides away the details of the underlying hardware
- Interface to IO, threading, etc.
  - Hides away the details of the underlying operating system
- "Write once, run anywhere"
  - Or as close as is realistically possible

Register Architecture

- .Net CLR and JVM are stack based
- Parrot is register based
  - Faster to interpret, since no stack pointer to keep (need to run on many odd platforms; shouldn't rely on JIT)
  - A little easier to JIT-compile too; "just" a register allocation problem
- Variable sized register frames per sub

HLL Feature Support

- Parrot provides support for a range of high-level language features
- By providing support for them at the VM level…
  - Compilers for different languages don't need to re-invent the wheel
  - Different languages can inter-operate

Examples Of HLL Features In The VM

- Common set of calling conventions
- Multiple dispatch
- Classes, attributes, methods, objects, inheritance, introspection (reflection)
- Namespaces
- Continuations, co-routines, closures
- Lexically scoped variables
- And more…

But Languages Are Different!

- We want to support a load of existing languages
  - Python
  - Ruby
  - PHP
  - JavaScript
- But they all have slightly different ideas about how certain things work…

PMCs

- PMC = Parrot Magic Class
- Implement some of a fixed set of methods that perform a range of common operations
- Range from simple things, like get an integer representation of this thing…
- …to more complex OO-related things, such as adding a parent class

Examples Of PMCs

- Integer PMC – implements methods relating to arithmetic
- Array PMC – implements methods relating to keyed access
- Class PMC – implements object orientation related methods
- Sub PMC – implements invoke method, and a few others (name, etc.)

Different Semantics, Common Interface

- The PMCs all provide the same interface
- Languages can implement this common interface to provide their own semantics
- For example, a Perl array can return "undefined" on access to an out-of-range element, whereas a Java array could throw an exception

Extensibility

- Don't need to have all the PMCs in the Parrot core; can build them into a dynamically linked library
- Can also dynamically load additional opcodes (instructions), to augment the VM's instruction set
- Language distribution = compiler + (optionally) PMC Library + (optionally) Opcode Library

PIR

- Parrot Intermediate Representation
- Essentially, the Parrot VM's "assembly"
- However, for some common things (like method calls), it turns some syntactic sugar into the several real instructions it takes to do it
- Also does register allocation for you, so compiler writers needn't worry about it

Some Simple PIR Examples

- "Hello, world!" – of course

```
.sub 'main' :main
    print "Hello, world!\n"
.end
```

- Compute The Answer

```
.sub 'main' :main
    $I0 = 25
    $I1 = 17
    $I2 = $I0 + $I1
    print $I2
    print "\n"
.end
```

Some Simple PIR Examples

- Factorial

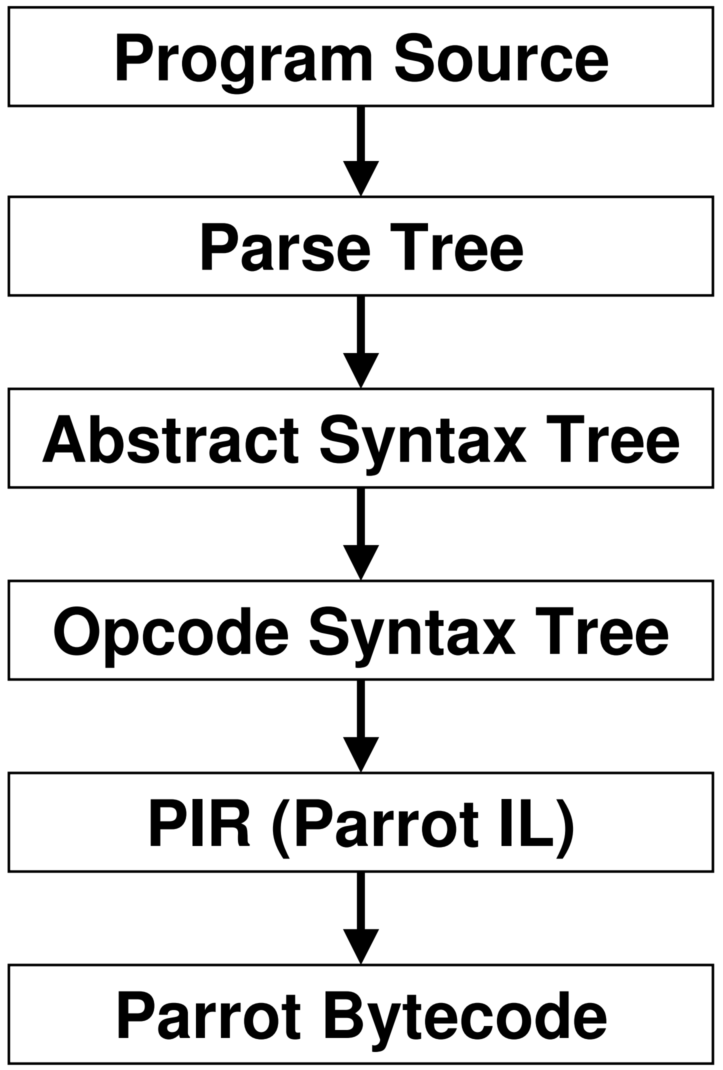
```
.sub 'fact'
    .param int n
    if n > 1 goto rec
    .return (1)
  rec:
    $I0 = n - 1
    $I1 = fact($I0)
    $I1 *= n
    .return ($I1)
.end
```

Parrot Compiler Toolkit

Writing Compilers Is Easy...

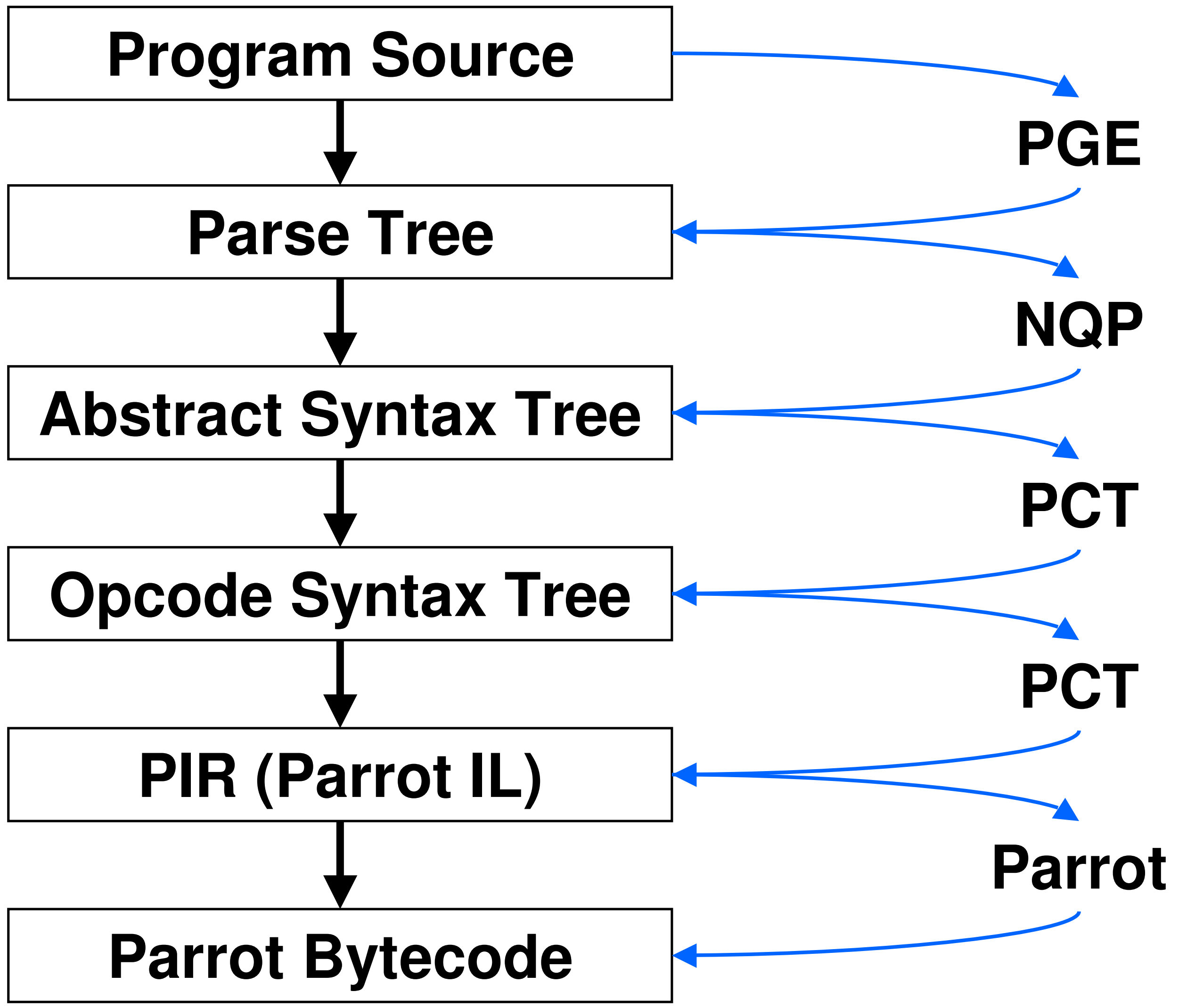
- ...if you have the right tools
- PCT aims to be a Right Tool
- You write the "front end":
  - Grammar, which specifies syntax
  - Actions, to produce an Abstract Syntax Tree from the Parse Tree
- The backend (from the AST down to Parrot bytecode) is done for you

Compilation Process



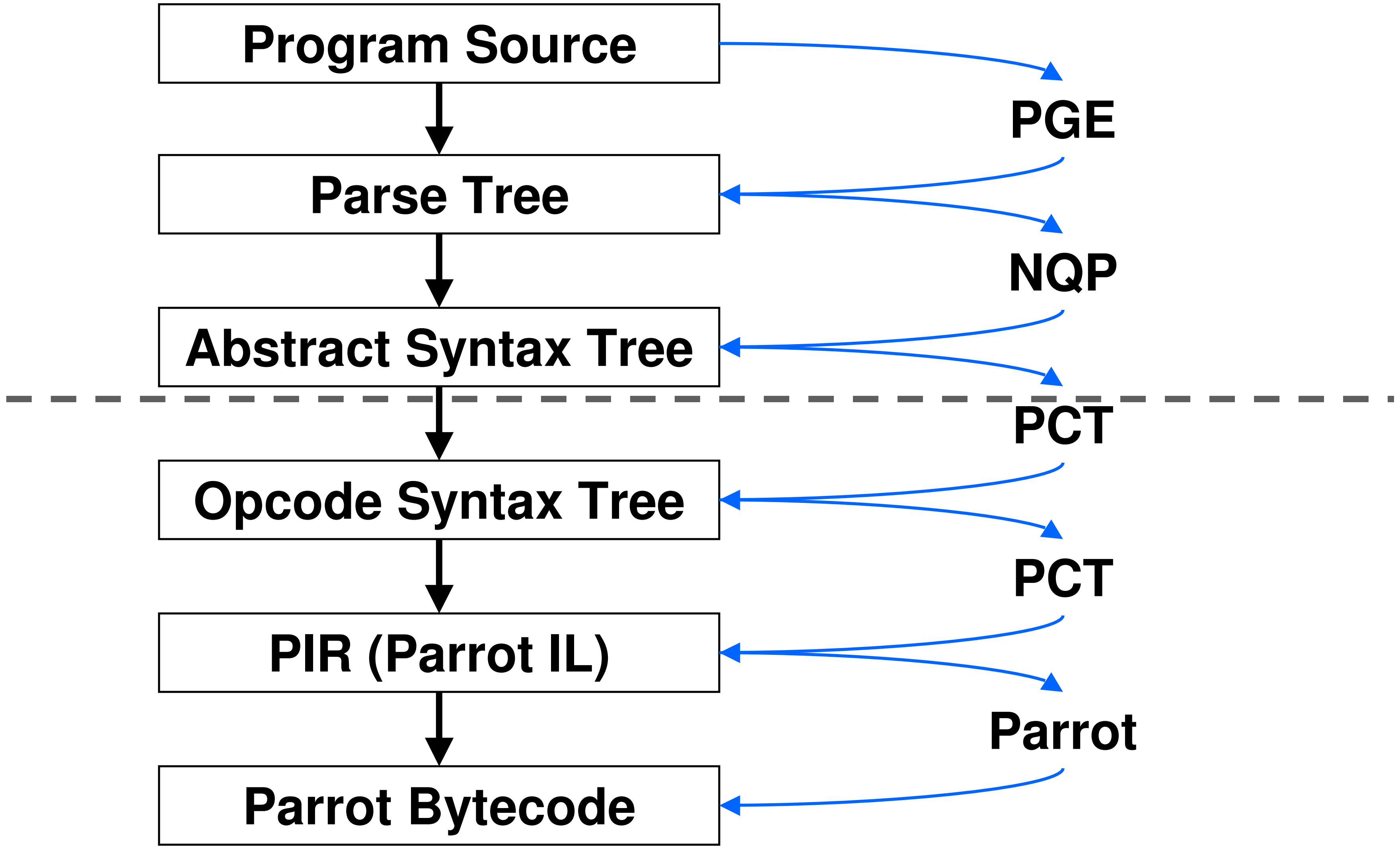
Rakudo Perl 6 and Parrot

Compilation Process

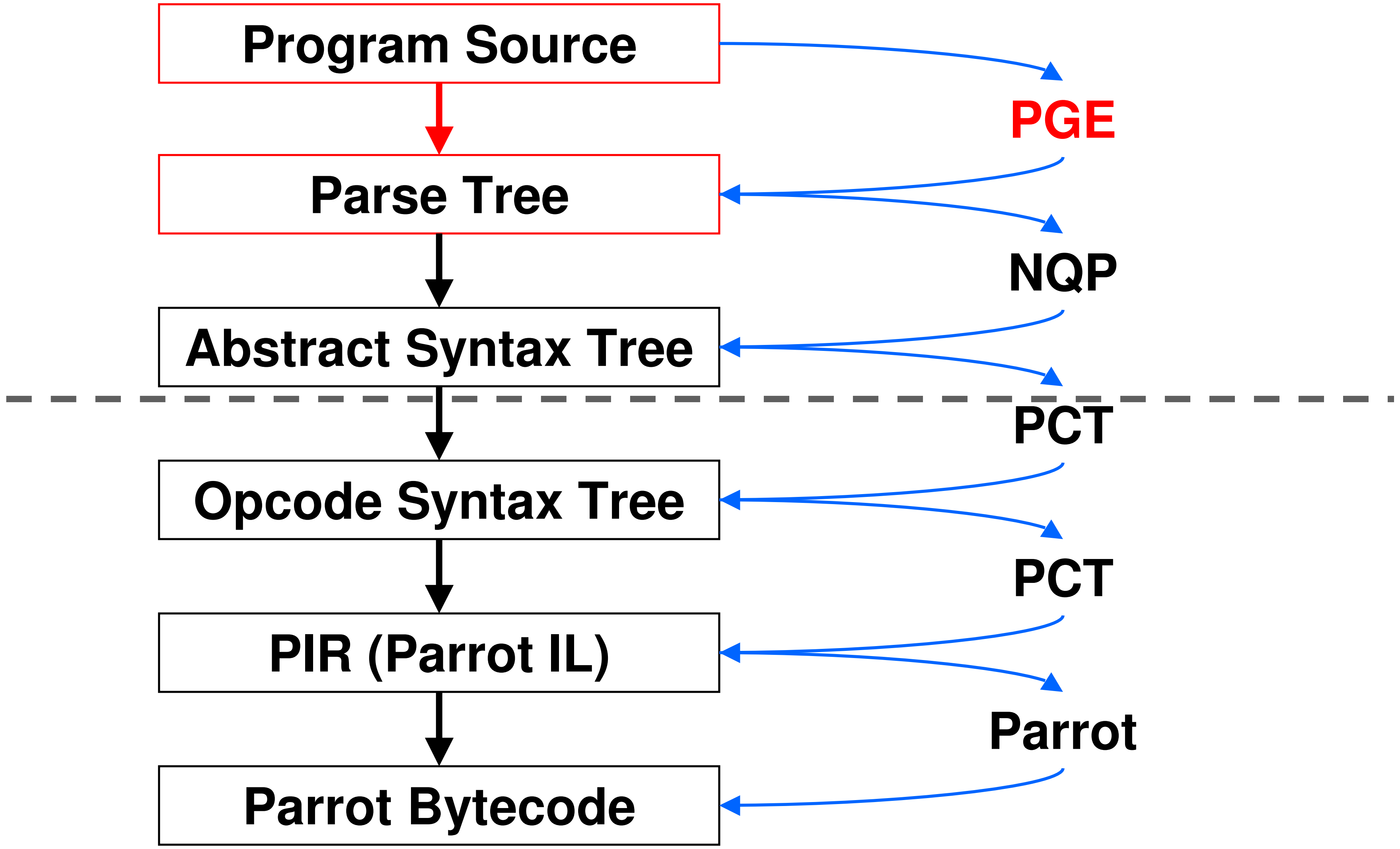


Rakudo Perl 6 and Parrot

Compilation Process



Compilation Process



PGE = Parrot Grammar Engine

- Implementation of Perl 6 rules
- A bit like regexes, but taken a step further so we can use them to write a full grammar
- Unlike more traditional tools like lex and yacc, where you write a tokenizer and a grammar, here you just write the parse rules and the tokenizer is generated for you

An Example Rule

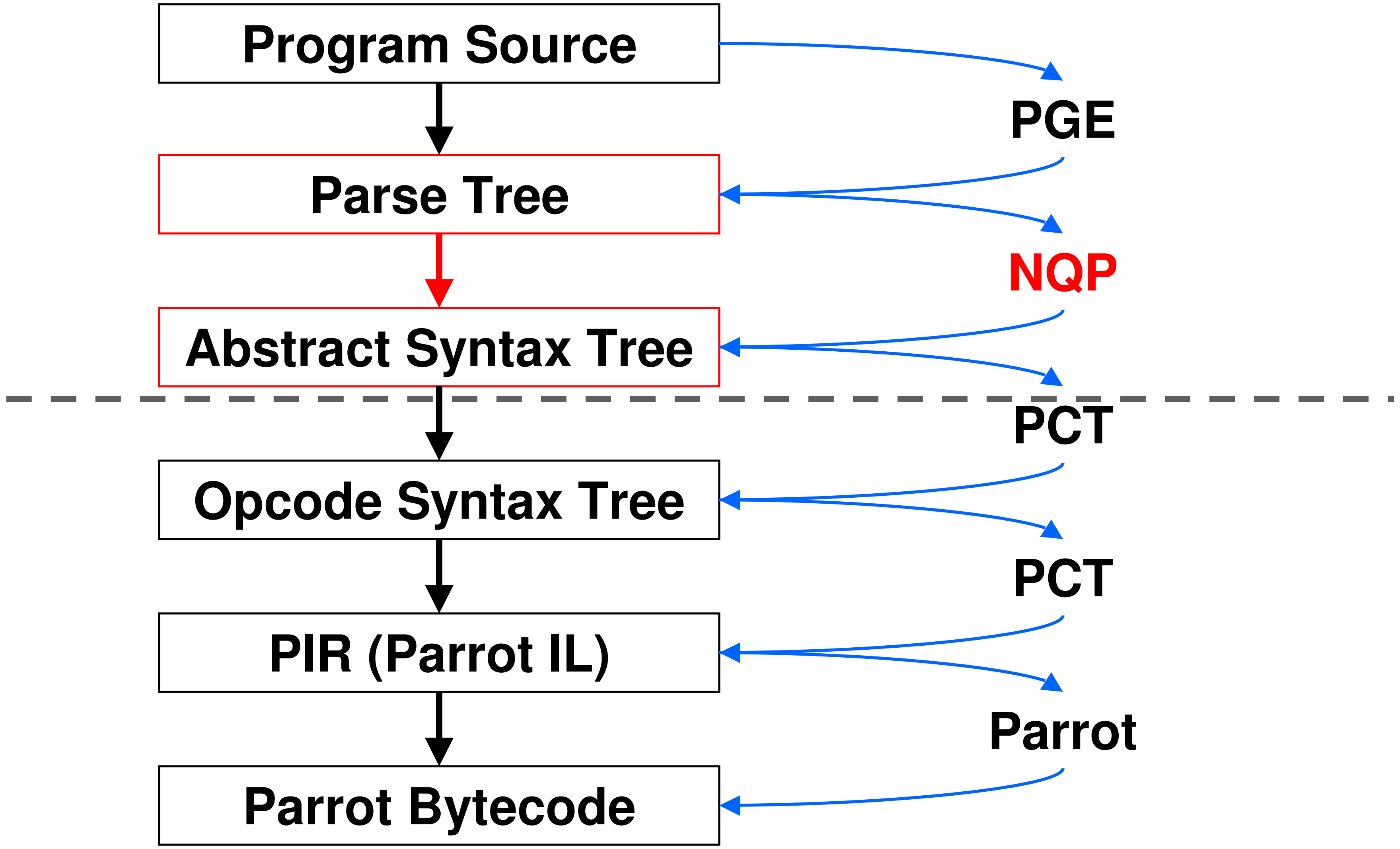
- You use PGE to write the grammar for your language
- For example, here's how we could parse an if statement

```
rule if_statement {
    'if' <expression> <block>
    {*}
}
```

- You put a {*} in place to indicate that we should run an action

Rakudo Perl 6 and Parrot

Compilation Process



NQP = Not Quite Perl 6

- A subset of Perl 6
- Contains just enough to allow you to produce an Abstract Syntax Tree from the parse tree
  - Variables and literals
  - Binding (but not assignment)
  - Conditionals and loops
  - Object instantiation and method calls

NQP = Not Quite Perl 6

- This method is called when the parser encounters the {*} in the grammar

```
method if_statement($/) {
     my $then := $( $<block> );
     $then.blocktype('immediate');
     my $past := PAST::Op.new(
          $( $<EXPR> ), $then,
          :pasttype('if'),
          :node( $/ )
     );
     make $past;
}
```

NQP = Not Quite Perl 6

- We are passed $/, the match object, which describes what was parsed

```
method if_statement($/) {
    my $then := $( $<block> );
    $then.blocktype('immediate');
    my $past := PAST::Op.new(
        $( $<EXPR> ), $then,
        :pasttype('if'),
        :node( $/ )
    );
    make $past;
}
```

NQP = Not Quite Perl 6

- Named captures ($<....>) give you the match object for the sub rules

```
method if_statement($/) {
    my $then := $( $<block> );
    $then.blocktype('immediate');
    my $past := PAST::Op.new(
        $( $<EXPR> ), $then,
        :pasttype('if'),
        :node( $/ )
    );
    make $past;
}
```

NQP = Not Quite Perl 6

- Writing $($<...>) gets you the AST for that match object

```
method if_statement($/) {
     my $then := $( $<block> );
     $then.blocktype('immediate');
     my $past := PAST::Op.new(
          $( $<EXPR> ), $then,
          :pasttype('if'),
          :node( $/ )
     );
     make $past;
}
```

NQP = Not Quite Perl 6

- We instantiate a new AST node of type Op

```
method if_statement($/) {
     my $then := $( $<block> );
     $then.blocktype('immediate');
     my $past := PAST::Op.new(
          $( $<EXPR> ), $then,
          :pasttype('if'),
          :node( $/ )
     );
     make $past;
}
```

NQP = Not Quite Perl 6

- This node has two children: the condition and the block to run

```
method if_statement($/) {
    my $then := $( $<block> );
    $then.blocktype('immediate');
    my $past := PAST::Op.new(
        $( $<EXPR> ), $then,
        :pasttype('if'),
        :node( $/ )
    );
    make $past;
}
```

NQP = Not Quite Perl 6

- Also specify the type of operation; PCT will then generate the appropriate code

```
method if_statement($/) {
    my $then := $( $<block> );
    $then.blocktype('immediate');
    my $past := PAST::Op.new(
        $( $<EXPR> ), $then,
        :pasttype('if'),
        :node( $/ )
    );
    make $past;
}
```

NQP = Not Quite Perl 6

- Also specify the match object that we made this from, for line numbers etc.

```
method if_statement($/) {
    my $then := $( $<block> );
    $then.blocktype('immediate');
    my $past := PAST::Op.new(
        $( $<EXPR> ), $then,
        :pasttype('if'),
        :node( $/ )
    );
    make $past;
}
```

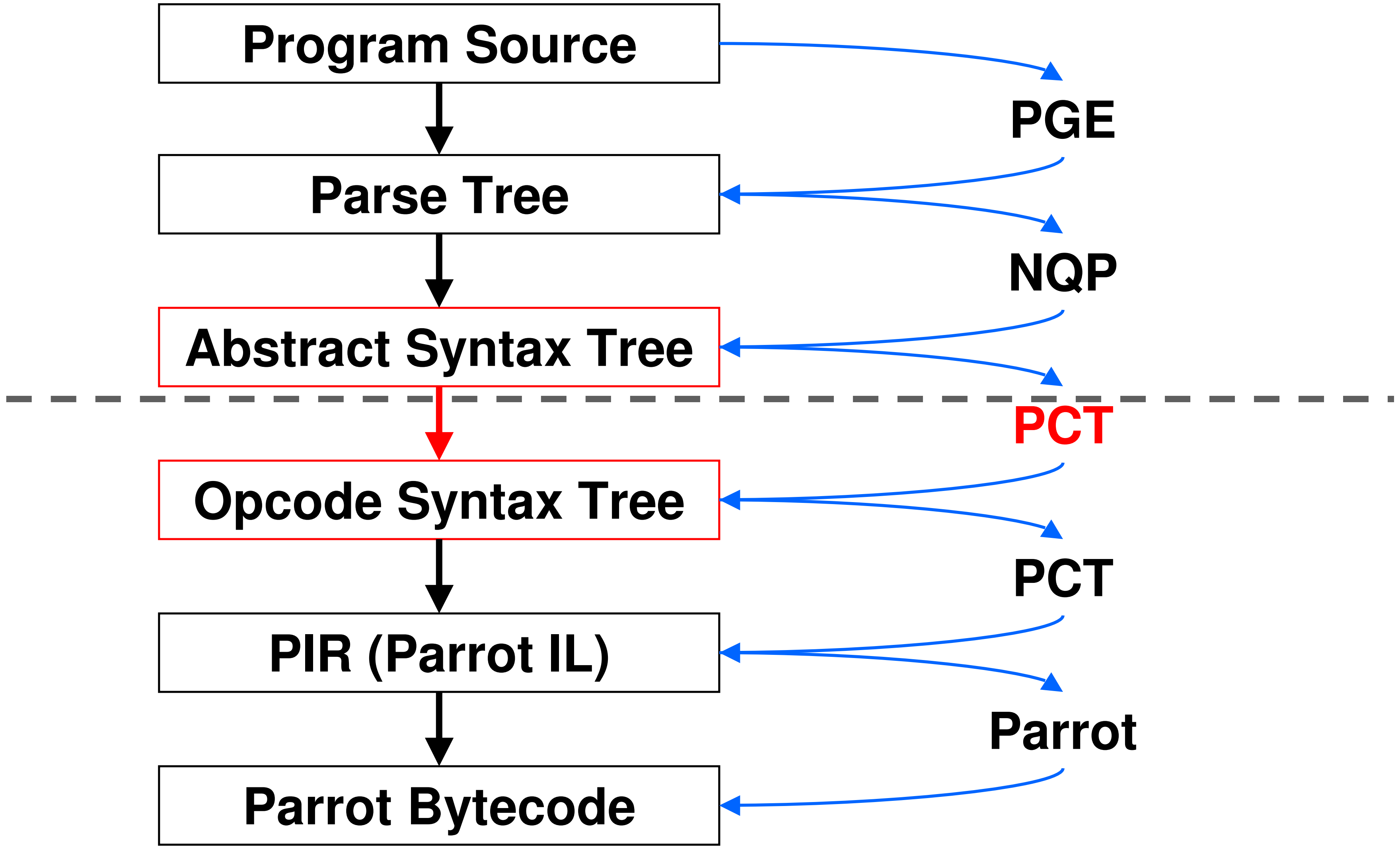
NQP = Not Quite Perl 6

- The "make" statement specifies the tree node we have made

```
method if_statement($/) {
    my $then := $( $<block> );
    $then.blocktype('immediate');
    my $past := PAST::Op.new(
        $( $<EXPR> ), $then,
        :pasttype('if'),
        :node( $/ )
    );
    make $past;
}
```

Rakudo Perl 6 and Parrot

Compilation Process

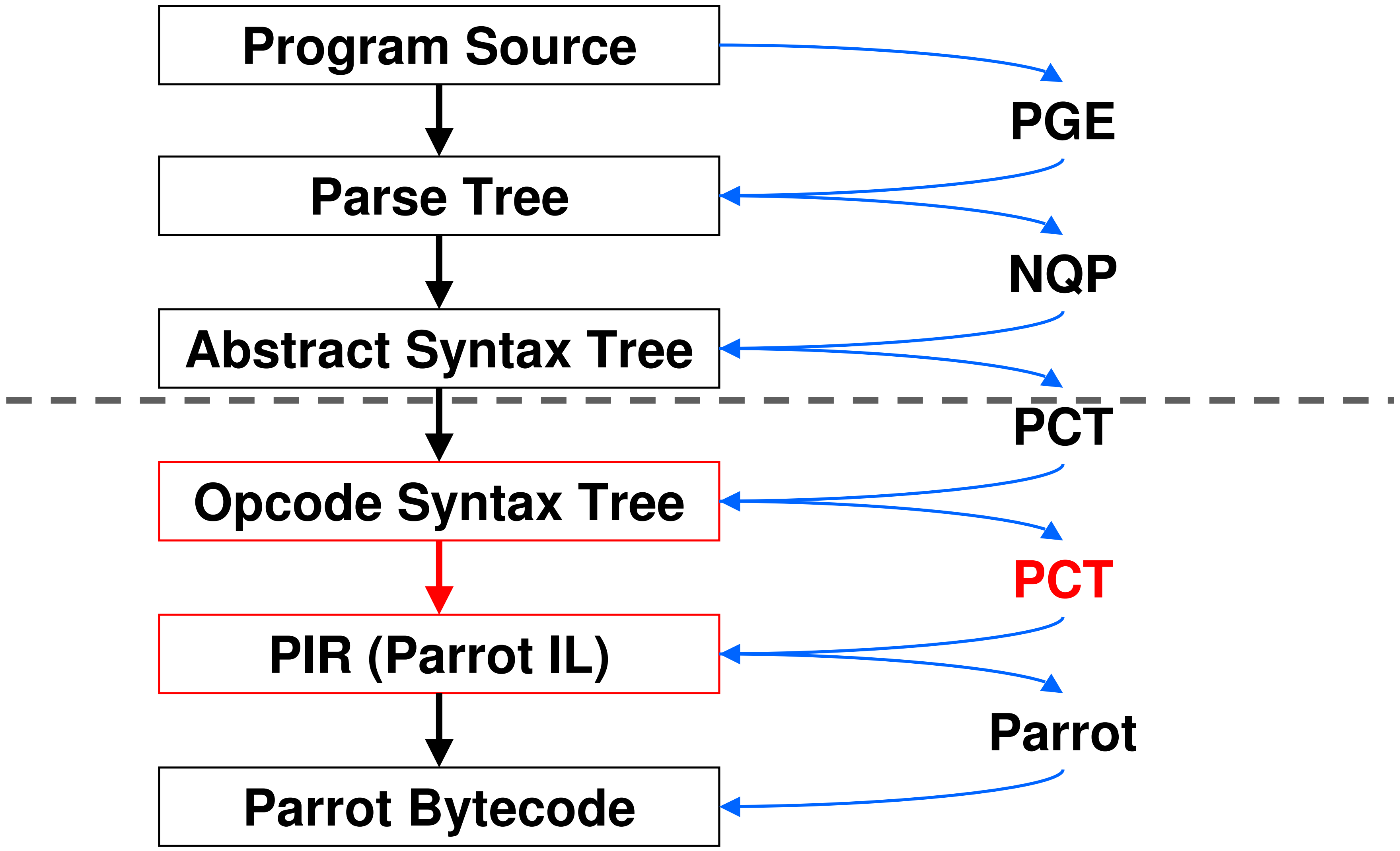


PAST to POST

- POST is the Parrot Opcode Syntax Tree
  - Tree representation of Parrot assembly program
  - Often one node = one instruction
- The PAST compiler, part of PCT, transforms a PAST node into (usually many) POST nodes

Rakudo Perl 6 and Parrot

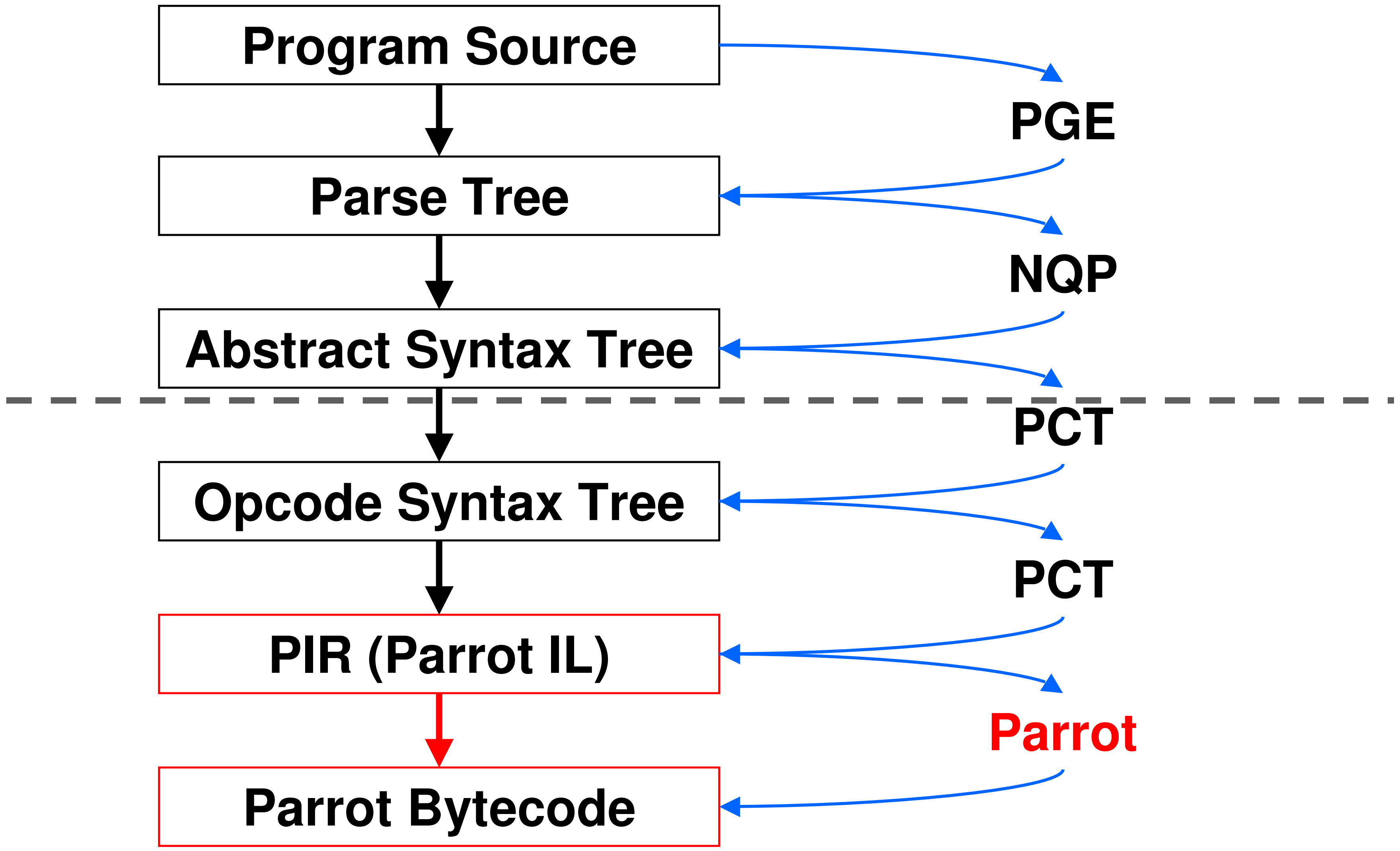
Compilation Process



POST to PIR

- PIR = Parrot Intermediate Representation
- Text based rather than tree based
- The Parrot VM itself understands PIR, so for now we have to turn the POST tree into PIR
- One day, we may be able to go direct from the tree to the bytecode

Compilation Process



PIR to Parrot Bytecode

- The Parrot VM actually executes bytecode – a binary representation of the program
- It contains a compiler that turns PIR into Parrot Bytecode
- We can write the bytecode to disk so we can load it again in the future => don't need to compile our program every time => performance!

Languages

- PCT is being used to build compilers for a range of languages…
  - Perl 6 (Rakudo)
  - PHP (Pipp)
  - Python (Pynie)
  - Ruby (Cardinal)
  - SmallTalk (ChitChat)
  - LOLCODE

Yes, LOLCODE

```
HAI
CAN HAS STDIO?
I HAS A VAR
IM IN YR LOOP
   UP VAR!!1
   VISIBLE VAR
   IZ VAR BIGGER THAN 10? KTHXBYE
IM OUTTA YR LOOP
KTHXBYE
```

Perl 6

What is Perl 6?

- Perl 5 has been and continues to be very widely used
  - Perl community is still very active
  - Perl is less fashionable than it once was, but no less useful
- Perl 6 is a ground-up re-design and re-implementation of the language
- Not backward compatible

Why start from scratch?

- Perl was first released in 1987
- It's now more than 20 years later; it'd be nice to think we learned a few things about languages in that time ☺
- Perl 5 internals are difficult to get into and extend
- Breaking backwards compatibility gives much more design freedom

Migration

- As usual with Perl, There's More Than One Way To Do It
- Implementation of a Perl 5 to Perl 6 translator is underway
  - Will retrain comments and all that a parser usually throws away
- Will be able to use Perl 5 modules from Perl 6

Isn't it taking a while?

- Yes, because…
  - It's a large and complex project
  - Doing some things that haven't been done in other languages before
  - Much is done by volunteers, though there has been and is some funding
- Perl 5 is still very powerful and being actively developed too

So What's New?

- Covering all of the new features of Perl 6, along with the changes from Perl 5, would take quite a while
- Going to give you a very quick look at some of the highlights

Chained Comparisons

- How often do you write stuff like:

```
if ($x >= 0 && $x <= 100) {
    ...
}
```

- In Perl 6 that is just:

```
if 0 <= $x <= 100 {
    ...
}
```

- Note that we can drop the parentheses on the condition too

Junctions

- How often do you write stuff like:

```
if ($drink eq 'Beer' || $drink eq 'Wine') {
    ...
}
```

- In Perl 6 that is just:

```
if $drink eq 'Beer' | 'Wine' {
    ...
}
```

- Junction = something that you can use where you'd use a single value, but acts as all of them simultaneously

Powerful New Object Model

- Now syntax for declaring classes, attributes and methods
- The method call operator is now .

```
class Beer is Drink {
    has $.units;
    has $.type;

    method consume($consumer) {
        $consumer.alcohol += $.units;
        self.spill() if $consumer.drunk;
    }
}
```

New Signature Syntax

- Rather than just getting an array of parameters, you can write a signature

```
sub add($x, $y) {
    return $x + $y;
}
say add(37, 5); # 42
```

```
sub greet($name, :$greeting = 'Guten Tag') {
    say "$greeting, $name";
}
greet('Hans'); # Guten Tag, Hans
greet('Lena', greeting => 'Privet');
                 # Privet, Lena
```

New Regex Syntax

- Many languages today use Perl Compatible Regular Expressions
- Regex are known for being rather, well, scary and line-noise-ish
- Perl 6 gives them a thorough re-working
- Easy to build reusable bits of regex and compose them

New Regex Syntax

- Whitespace not matched as part of the regex now – so space stuff out!
- [...] is a non-capturing group
- Put a literal in quotes, like elsewhere

```
regex Decimal { \d+ ['.' \d+]? };
```

- Build other regex out of it

```
regex Temperature { <Decimal> \s* [C|F] };
regex HighTemp { 'High' \W+ <Temperature> }
regex LowTemp  { 'Low' \W+ <Temperature> }
```

Grammars

- Can collect a bunch of regex together into grammars (like classes but with regex instead of methods)
- Good for building parsers
    - Unlike traditional regex, has control of backtracking
    - Auto-generates a lexer for you
- Perl 6 will parse itself

Multiple Dispatch

- One name, different signatures

```
sub win(Paper $x, Stone $y)     { True }
sub win(Scissors $x, Paper $y) { True }
sub win(Stone $x, Scissors $y) { True }
sub win(Any $x, Any $y)         { False }
```

- When you do a call, it dispatches to the best candidate
- All operators are really just multiple dispatch subs => overloading is just writing a multi-sub

Rakudo

Rakudo vs. Perl 6

- Perl 6 is the name of the language
- Rakudo is an implementation of the Perl 6 language
- However, it's not the only one that is in progress, or that exists
- Pugs = Perl 6 in Haskell, currently not maintained
- kp6 and SMOP are two others

Why "Rakudo"?

- Suggested by Damian Conway
- Some years ago, Con Wei Sensei introduced a new martial art in Japan named "The Way Of The Camel"
- In Japanese, this is "Rakuda-do"
- The name quickly became abbreviated to "Rakudo", which also happens to mean "paradise" in Japanese

Status

- Rakudo is not a complete implementation of Perl 6 yet
- However, all of the code that I showed in the previous section of the talk will run on Rakudo today…
- …and much more.
- Uses PCT, meaning much of it is in Perl 6 rules and a subset of Perl 6

What's (Mostly) Implemented?

- Variables: scalars, arrays, and hashes
- Wide range of operators
- Conditionals and loops
- Subroutines with signatures
- Classes, attributes, methods, inheritance, delegation, roles and composition, runtime mix-ins
- Enumerations

What's (Mostly) Implemented?

- Regexes and grammars
- Type constraints on variables, parameters and attributes
- Multiple dispatch (very much a work in progress, but getting there)
- Basic junction support (but much, much more to do here)
- Basic I/O

Compilation To Bytecode

- You can compile your programs or modules down to Parrot bytecode
- Means you don't have to run the compiler every time you want to run the program
- Additionally, use mmap when it's available, so Parrot instances can share the bytecode file

mod_perl6

- There is also basic support for writing Apache handlers in Perl 6 now
- mod_parrot provides most of what is needed, and mod_perl6 is a thin layer (written mostly in Perl 6 itself) on top of that
- Easy to make mod_your_language
- Yes, we do have mod_lolcode ☺

Learning More

Where To Learn More

- The Parrot Website
  http://www.parrot.org/
- The Parrot Blog recently had an 8-part PCT tutorial posted
  http://www.parrotblog.org/
- The Perl 6 implementation on Parrot (named Rakudo) has a site here
  http://www.rakudo.org/

Get Involved!

- Join the Parrot and/or Perl 6 compiler mailing list
- Pop onto the IRC channel
- Get the source and start hacking
  - Partial implementations of many languages – come and help us get your favorite one running on Parrot
  - Or if you like C, lots of VM guts work

Rakudo Perl 6 and Parrot

Come!

- There will be a Perl Workshop, one day in Vienna, one day in Bratislava

- 7th and 8th of November

http://conferences.yapceurope.org/tcpw2008/

Danke

Questions?