Interpreting Languages for the Java Platform

http://www2.hursley.ibm.com/netrexx/

Mike Cowlishaw

IBM Fellow

mfc@uk.ibm.com

Overview

- A (very) brief introduction to NetRexx
- Demonstration -- compiling and interpreting NetRexx programs
- The compiler/interpreter implementation
- Questions?

What is NetRexx?

- A complete *alternative* to the Java language, for writing classes for the JVM
- Based on the simple syntax of Rexx, with Rexx decimal arithmetic
- Fully exploits the Java object model, exceptions, and binary arithmetic
- Automates type selection & declaration

NetRexx programs

hello.nrx

```
/* The classic greeting. */
say 'Hello World!'
```

Another simple program

```
/* cubit.nrx */

loop label prompt forever
  reply=ask
  select
    when reply.datatype('n') then say reply**3
    when reply='Quit' then leave prompt
    otherwise say 'eh?'
    end
  end prompt

say 'Done.'
```

Using other Java classes

```
method update(g=Graphics)
  shadow=createImage(getSize.width,-
      getSize.height)  -- make new image
  d=shadow.getGraphics -- graphics context
  maxx=getSize.width-1
  maxy=getSize.height-1
  loop y=0 to maxy
    col=Color.getHSBColor(y/maxy, 1, 1)
    d.setColor(col)
    d.drawLine(0, y, maxx, y)
  end y
  paint(g)             -- paint to screen
```

NetRexx Java implementation

- Current implementation first ***translates*** NetRexx to accessible Java source, or ***interprets*** it directly (or both)
- Runs on any Java platform
- Any class written in Java can be used
  - GUI, TCP/IP, I/O, DataBase, *etc.*
- Anything you could write in Java can be written in NetRexx

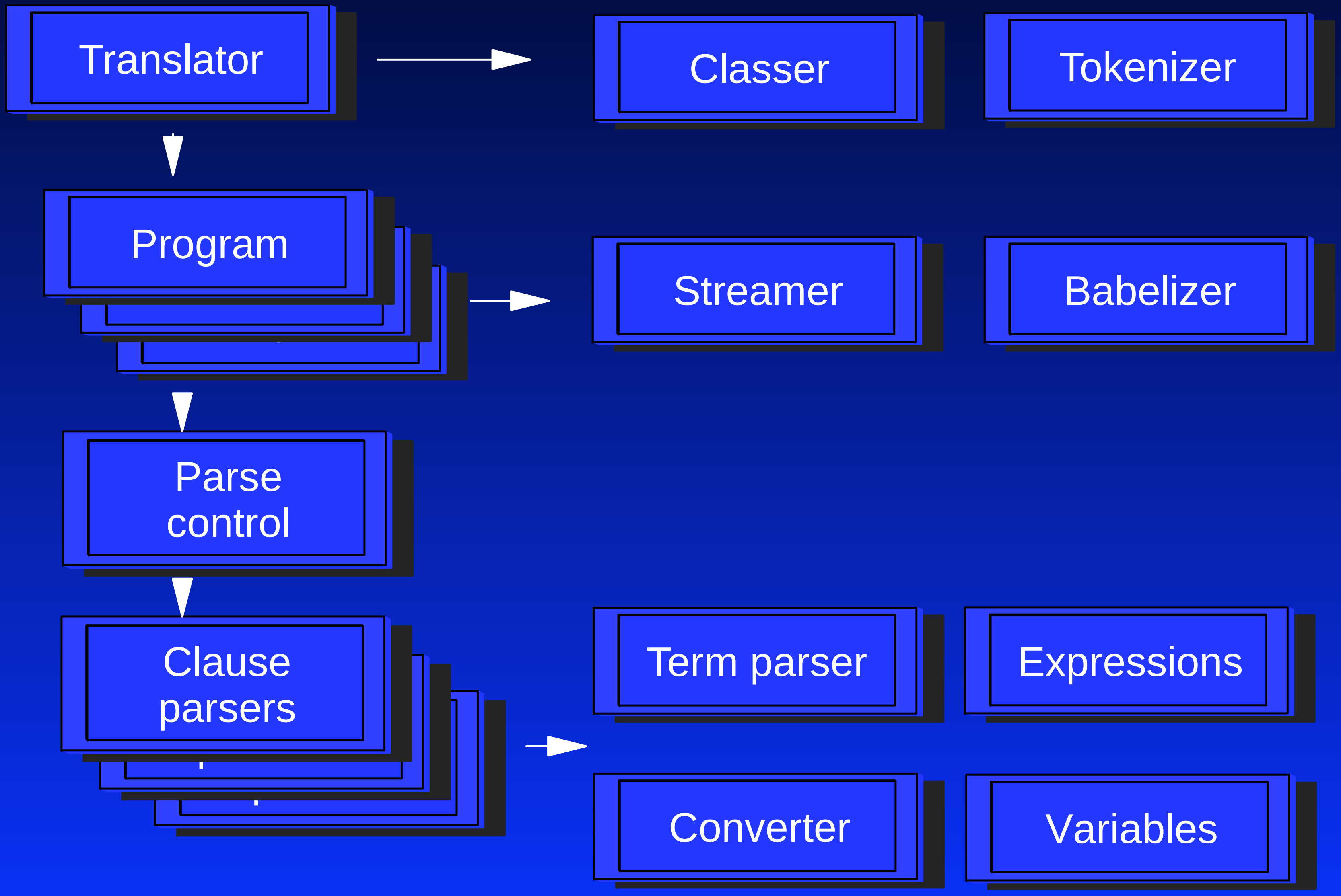
. . . and it's free.

Demonstration ...

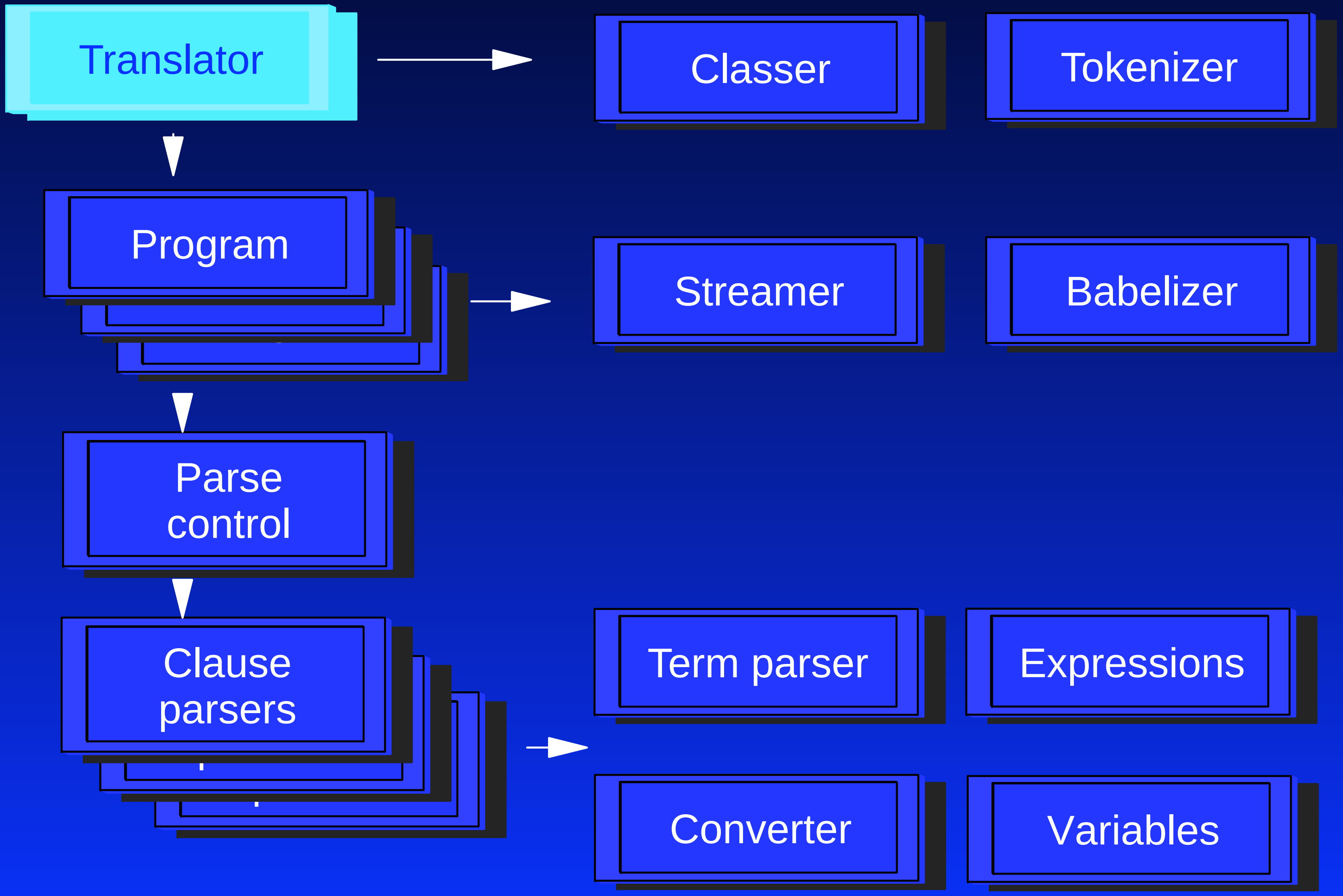
So how does it work?

- Unconventional organization
- Structured like an interpreter, not like a compiler
- Parsing is not carried out 'up front', but on demand
- Parsing is identical for translation to Java or for direct interpretation, with full error checking at the point of parsing; allows multi-syntax

Overall translator organization



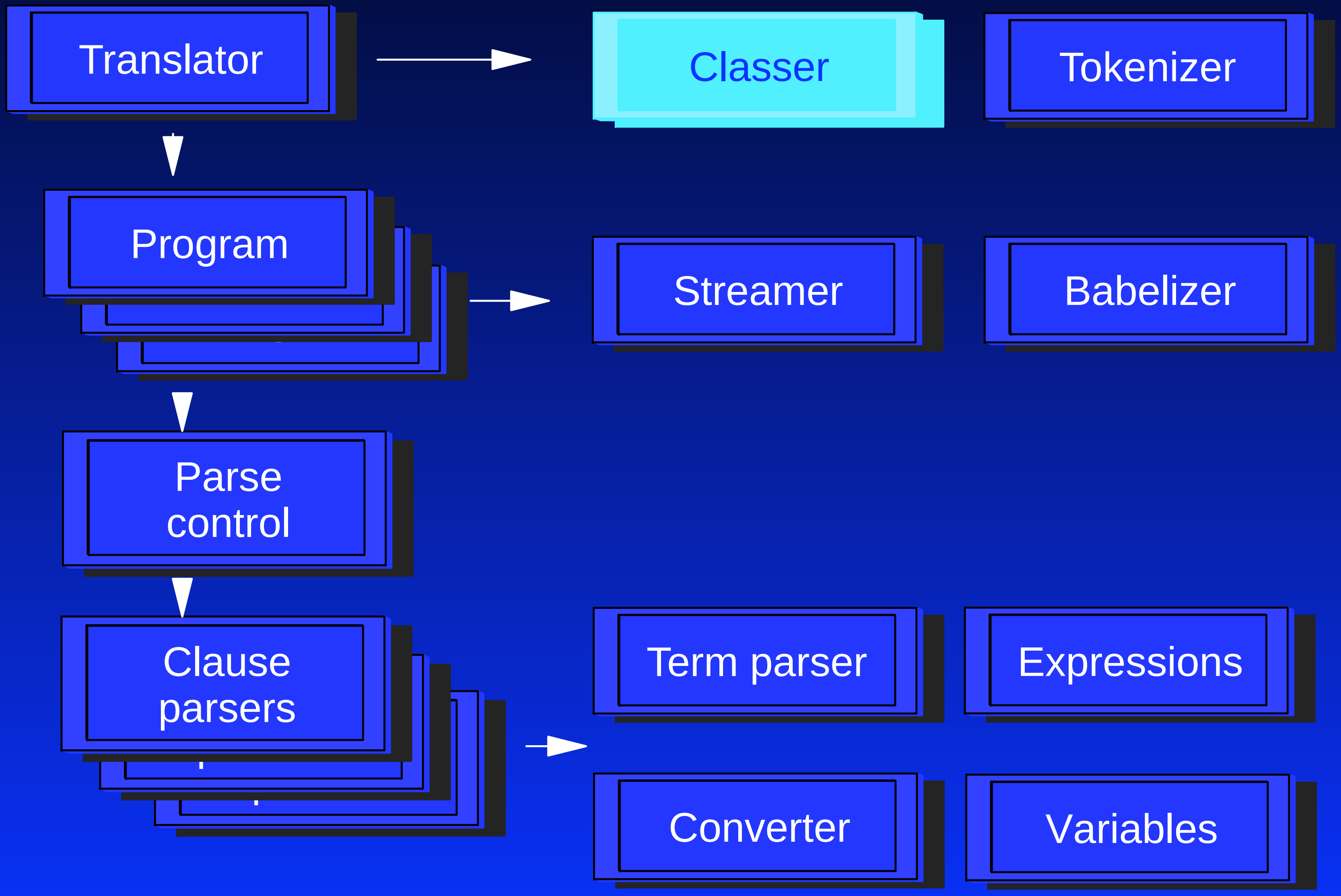
Overall translator organization



Translator

- Internal API for NetRexxC to use
- Factory, language, and programs setup
- Cross-program pass control (3 main passes)
- Manages compilation using javac
- Manages interpretation
- Top-level error handling

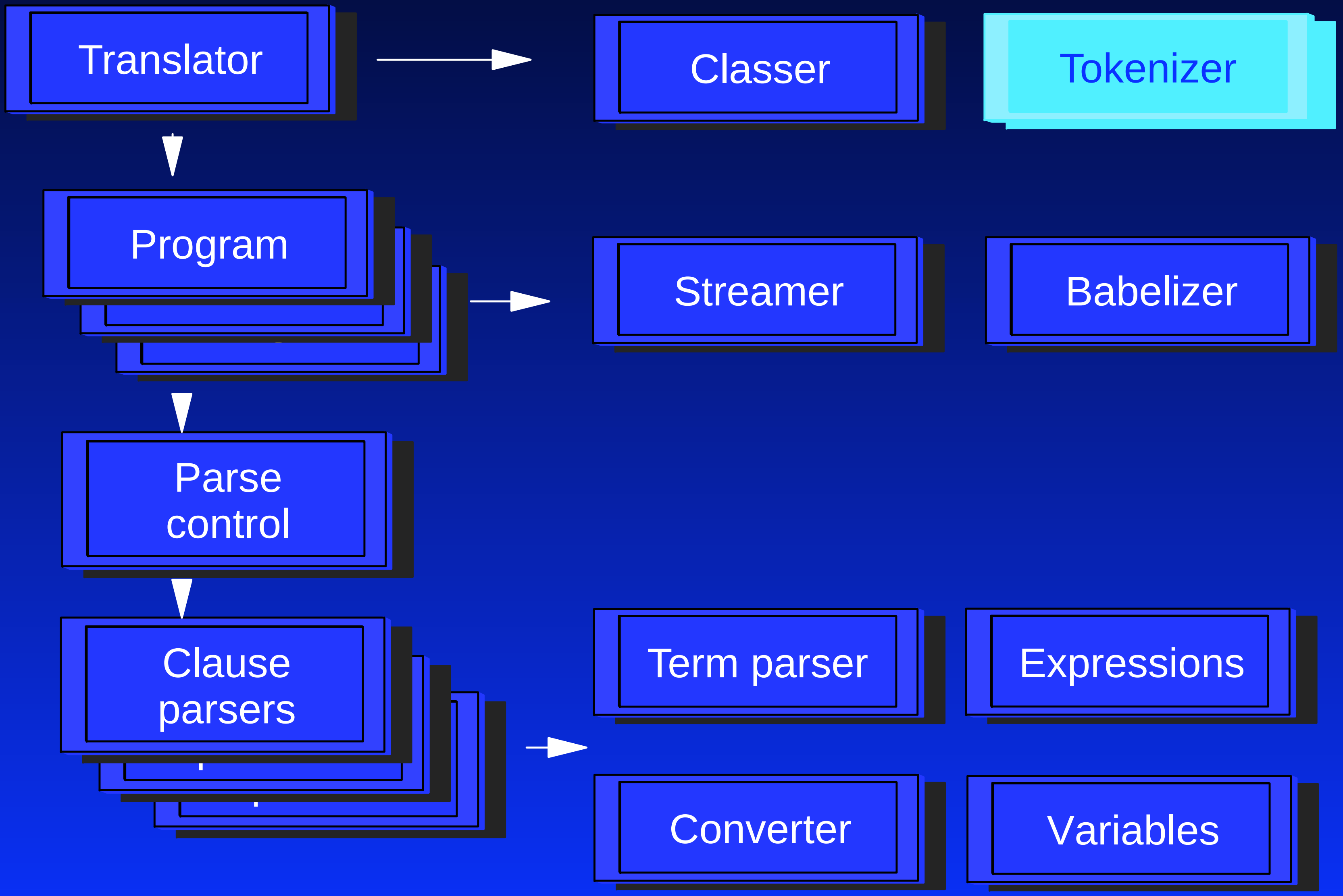
Overall translator organization



Classer

- Most difficult area of translation, due to changes in Java core over time
- In general 'owns' the external namespace
- Manages class path, ambiguous classes, *etc.*
- Locates, reads, and parses class images
- Locates methods and properties, based on costing algorithm

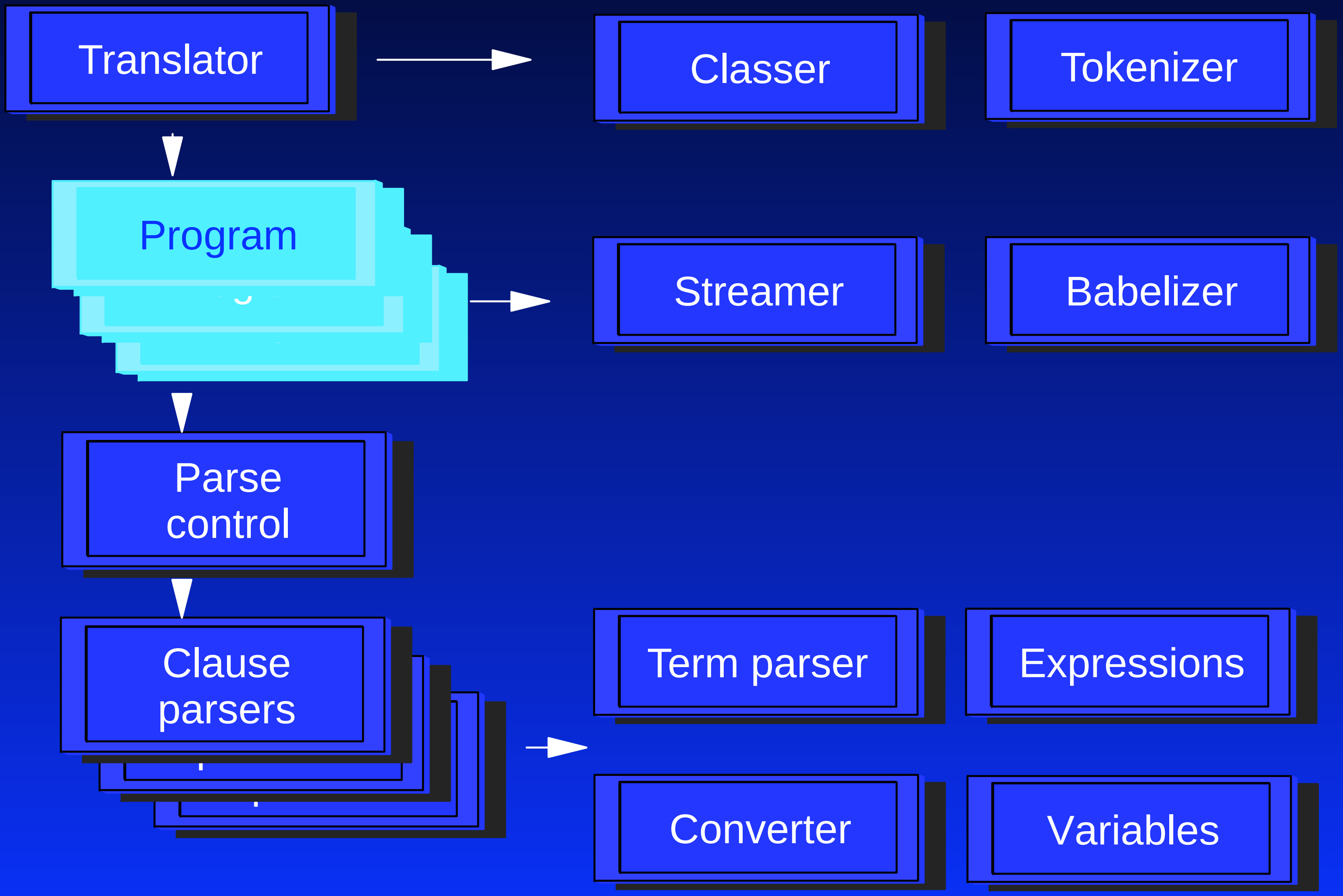
Overall translator organization



Tokenizer

- One of several shared resources
- Language-independent tokenizing of an input stream or array of character arrays
- Other shared resources include:
  - error message editor
  - base internal types (Tokens, Flags, Types, *etc.*)
  - trace code generator
  - interfaces (ClauseParser, ProgramSource, *etc.*)

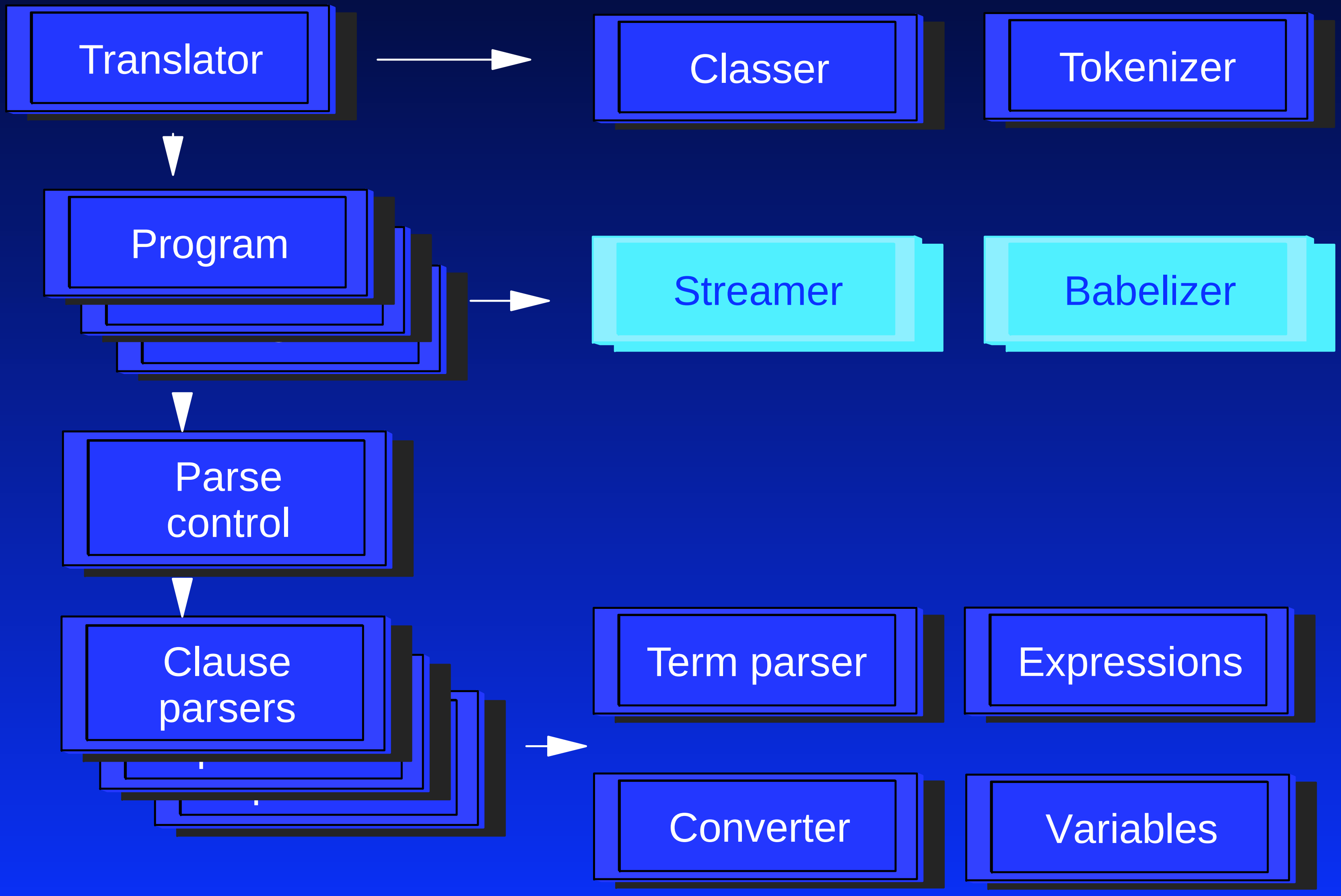
Overall translator organization



Program

- Represents exactly one of the programs being translated
- Each program may be in a different language, with different syntax (and different semantics at the statement level)
- Holds program-level objects (streamer, package information, imports, options, *etc.*)

Overall translator organization

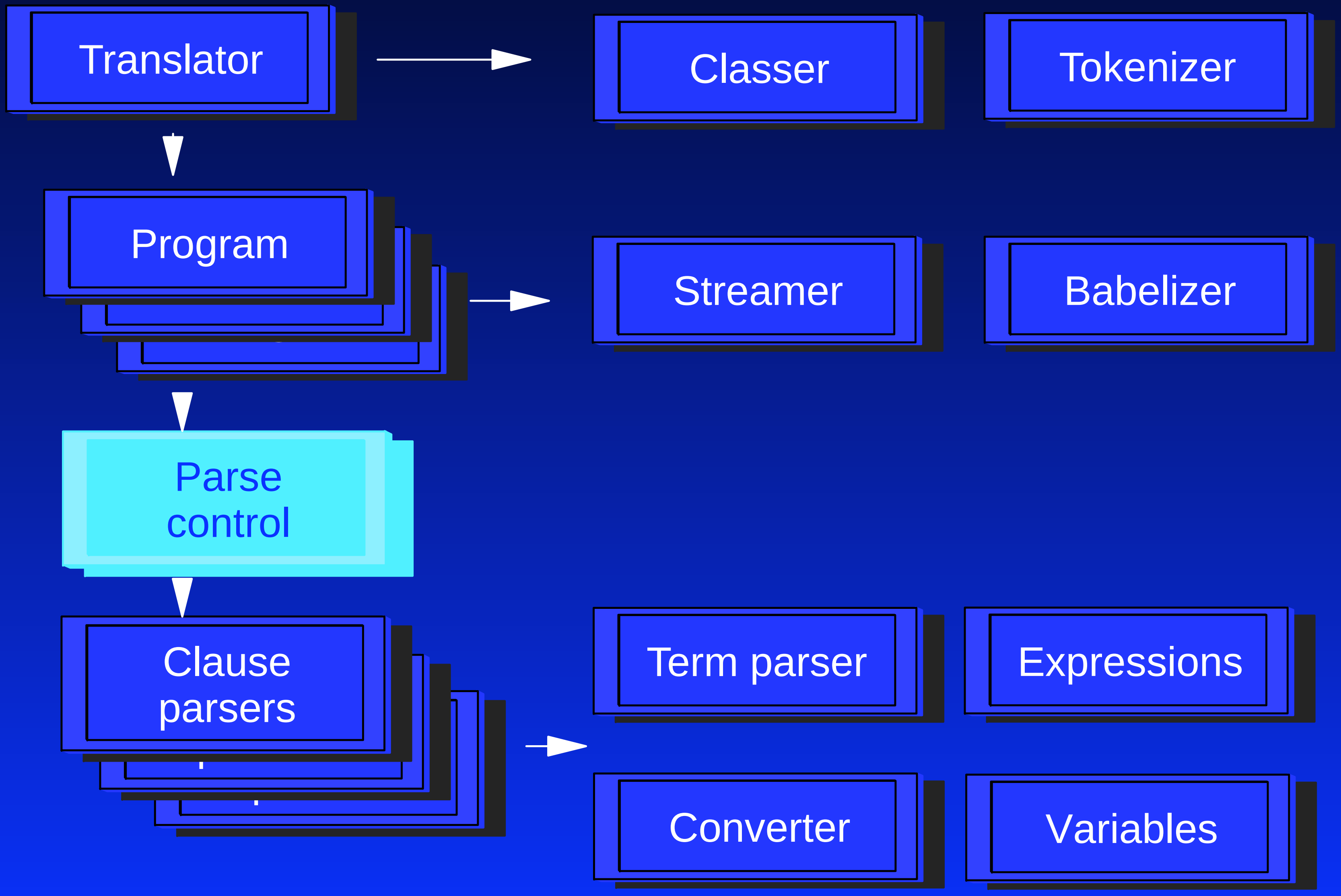


Streamer and Babelizer

- Streamer handles input and output streams
  - locates input files
  - names and creates output files
  - checks for conflicts
  - reads files on demand

- Babelizer converts internal representations to viewable strings, depending on the language
  - associates file extensions with languages
  - arrays shown as **[ ][ ]** or **[,]** or **(,)**
  - attributes spelled as appropriate for the language; *e.g.*, **shared** or **Friend**

Overall translator organization



Translator

Classer

Tokenizer

Program

Streamer

Babelizer

Parse control

Clause parsers

Term parser

Expressions

Converter

Variables

Parse control

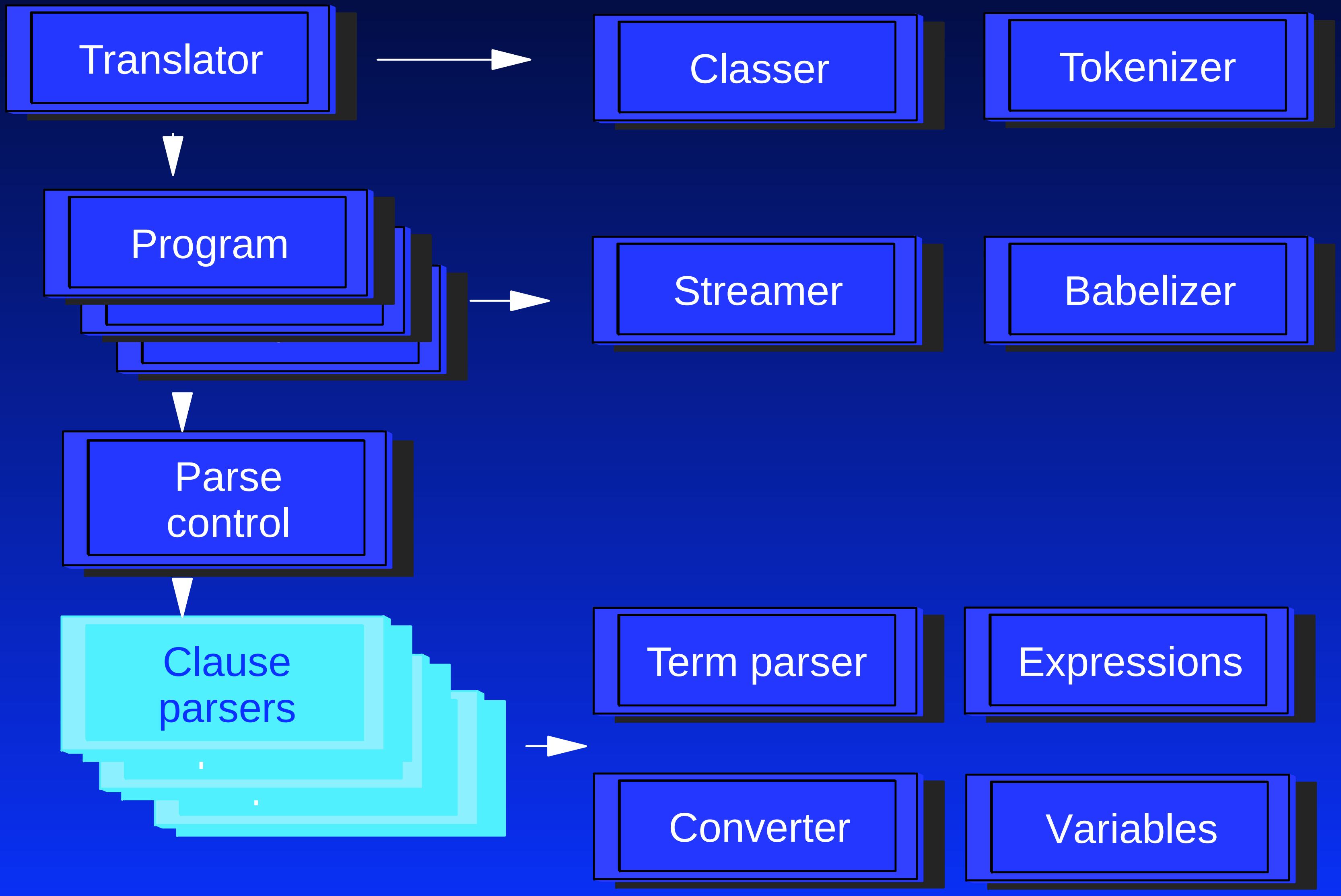
- State machine for static parsing

- Language-dependent (hence one instance per program)

- Three levels of parsing, deferred where possible:
  - parseProgram
  - parseClassBody
  - parseMethodBody

- Parsing-related utilities (pushLevel, popLevel, *etc*.)

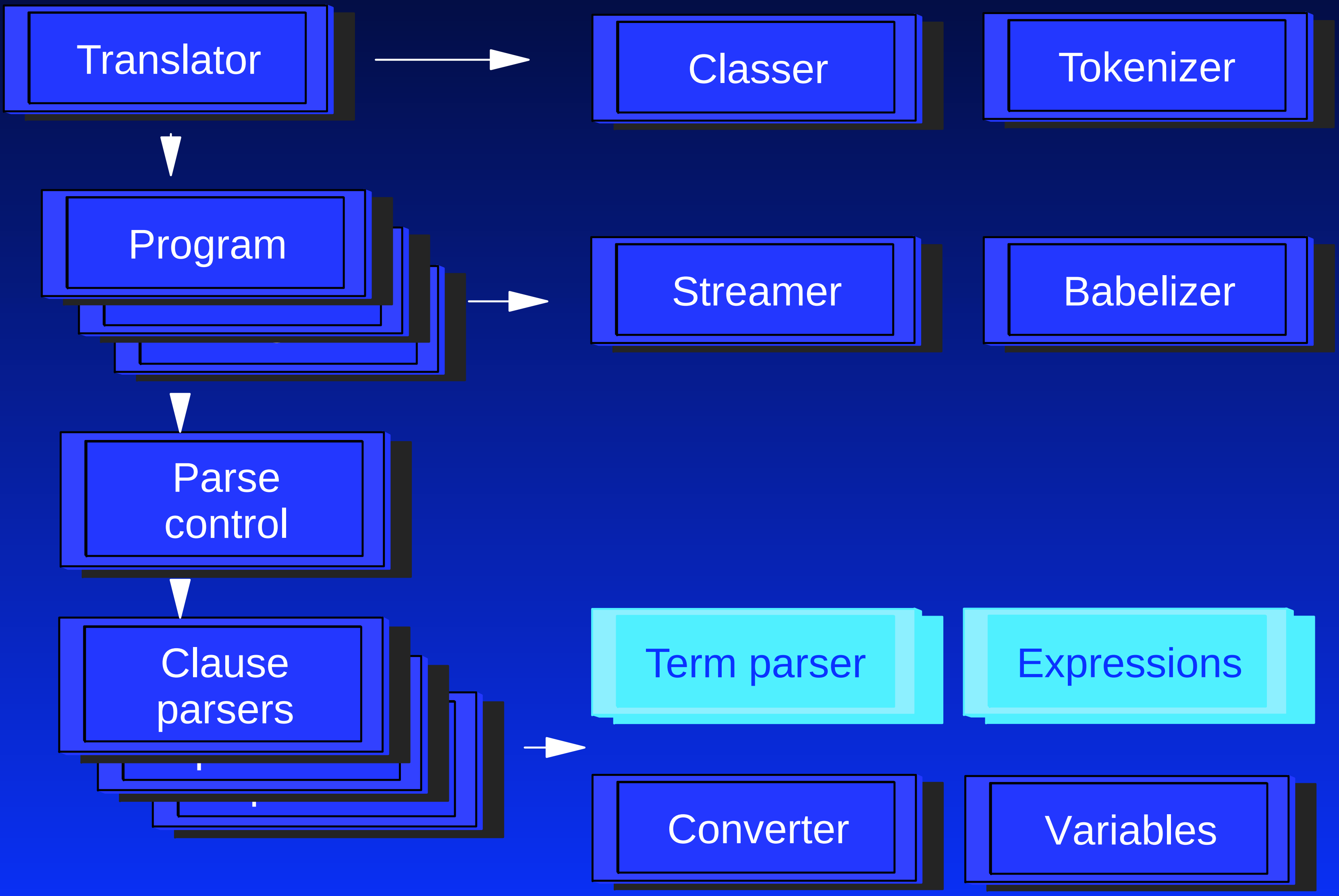
Overall translator organization



Clause parsers

- Each knows about a single clause in one language (Do, Catch, End, Nop, Say, *etc.*)
- Each has a scan method (lexical parse)
- Each has a generate method, for Java code
- Each has an interpret method
- generate and interpret share information gleaned during scan (which may have been multi-pass)

Overall translator organization



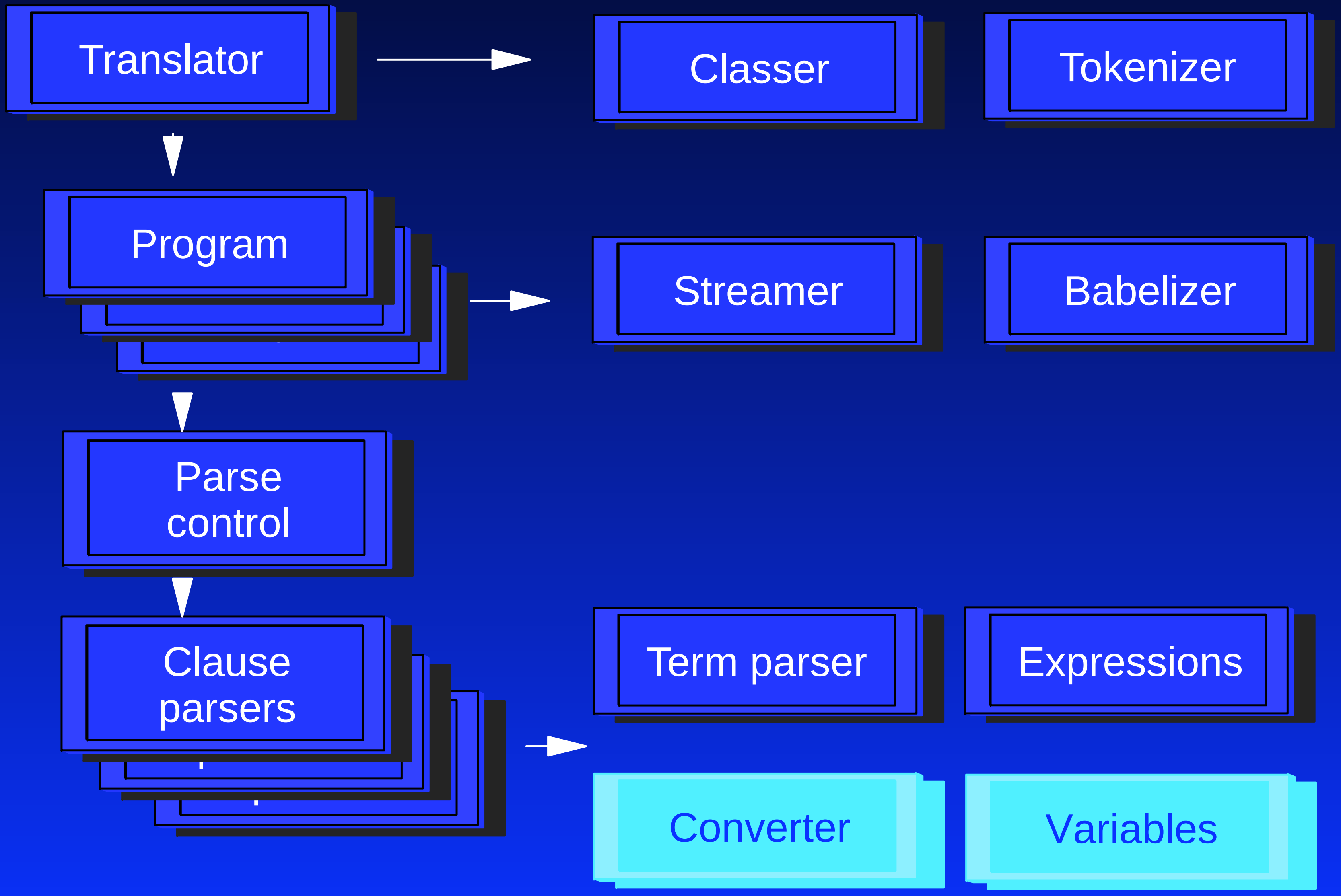
Term and Expression parsers

- Recursively call each other to parse terms and expressions. For example:

```
(Rexx vector.get('key')).substr(i+1, j)
```

- Term parser is more complicated than Expression parser, and is easily the largest class in the translator (100K characters, including comments)
- Like clause parsers, both can emit Java code or execute (interpret) the term or expression

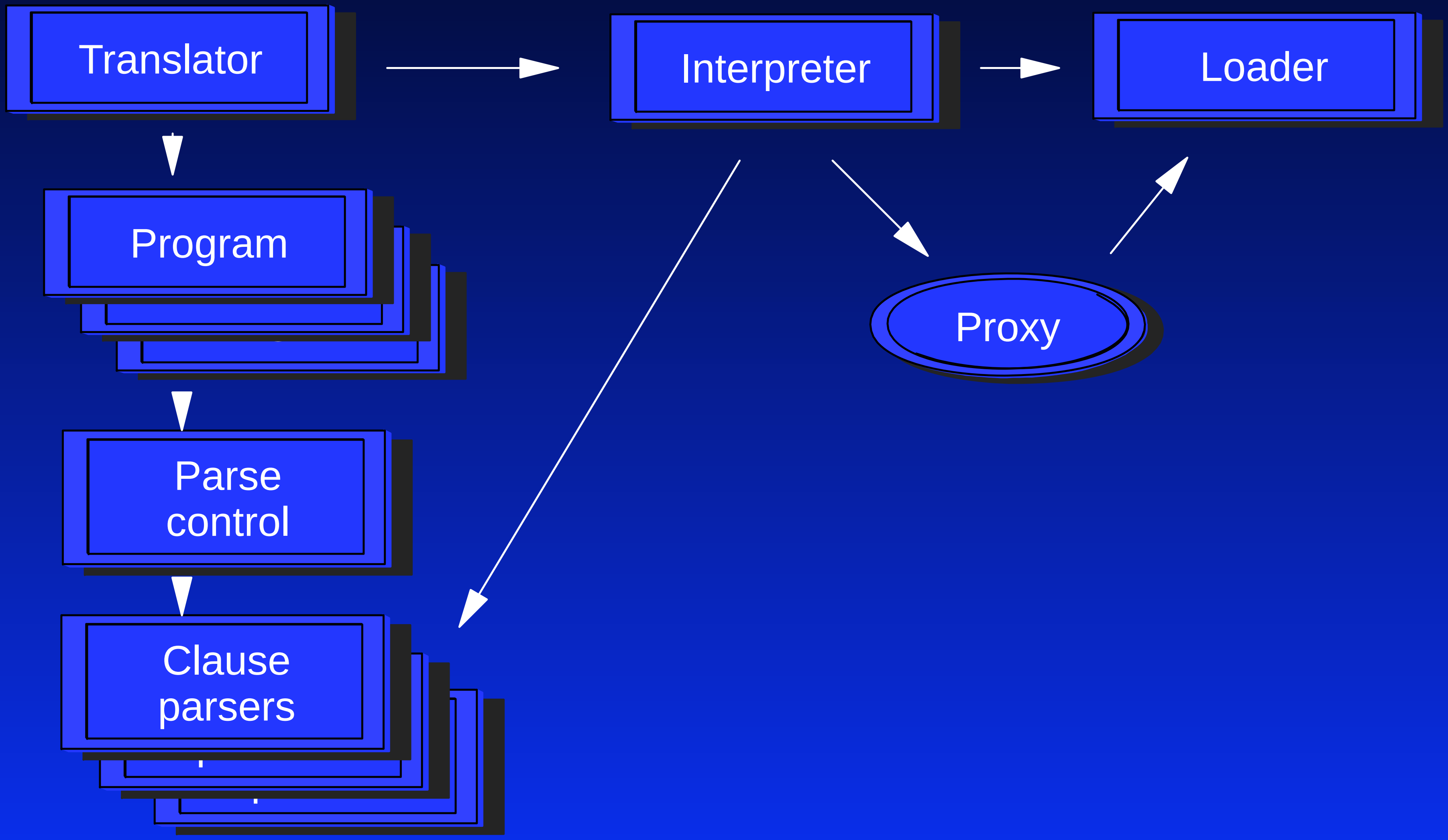
Overall translator organization



Converter and Variable manager

- Converter understands type inferences
  - costs conversions (used for method finding and error checking)
  - effects conversions (emits Java code or interprets)
- Variable manager handles both class (static and instance) and method variables
  - all properties and local variables during scan passes
  - only static (Class) properties and local variables are handled during interpretation - instance properties are held in a real object

Interpretation



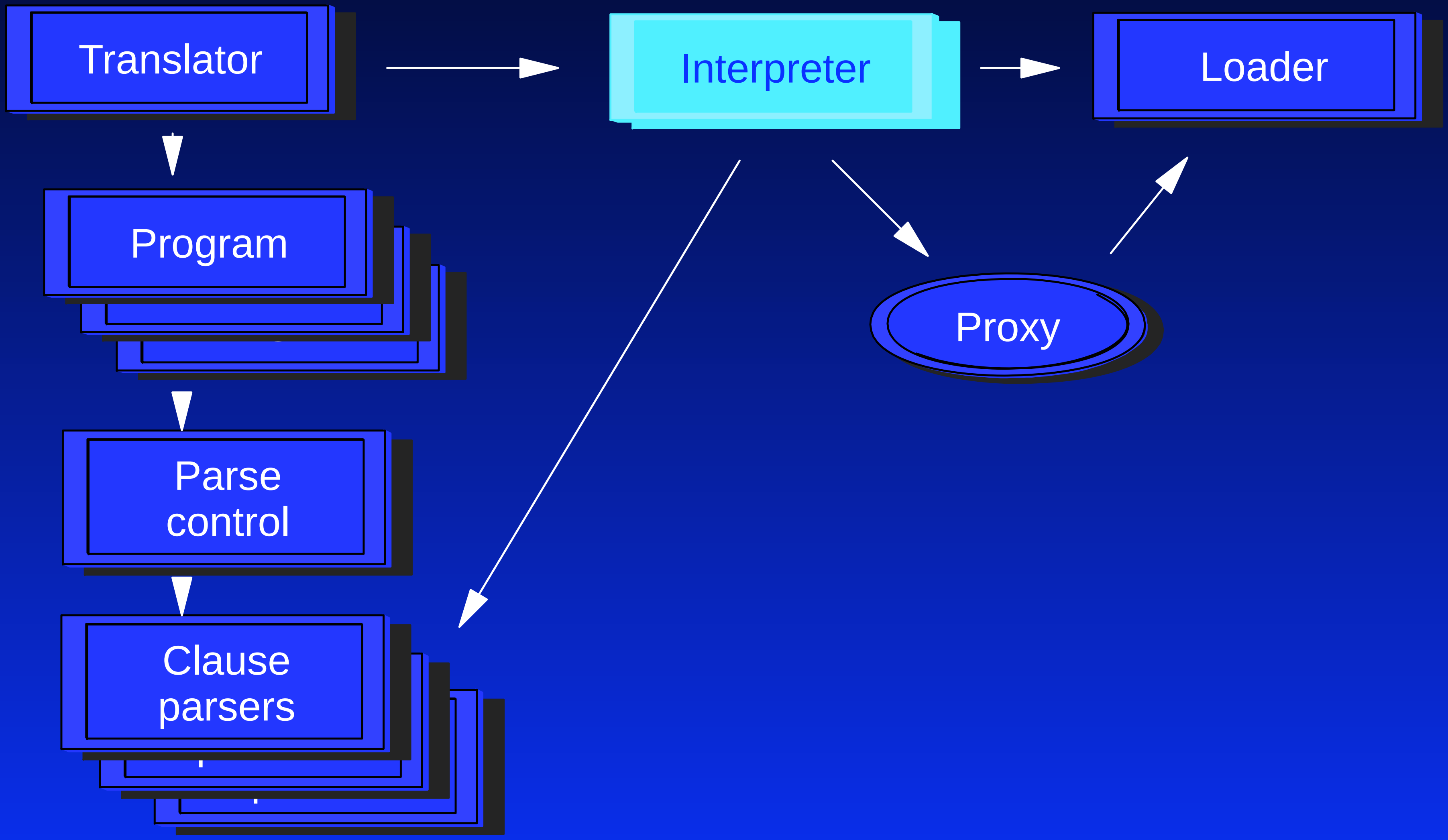
General principle

- First, programs are parsed (to determine classes, properties, and methods with their signatures)

- For each class, a *proxy* (stub) class is created
  - this has all the properties just as in a 'real' class
  - for each method, it has *only* the definition and return
  - when a method is invoked through Java reflection, it immediately calls the interpreter, which interprets the code in the method body

- Real instances are created, so interpreted classes are visible to the JVM for callbacks, *etc*.

Interpretation



Interpreter

- Primary task is interpreting method bodies, by finding each clause in turn and invoking its interpret method

- When a class is first used or an instance is constructed, interprets initialization code (properties, numeric context, *etc.*)

- Handles Java reflection (access to real properties, instances of objects, arrays, *etc.*)

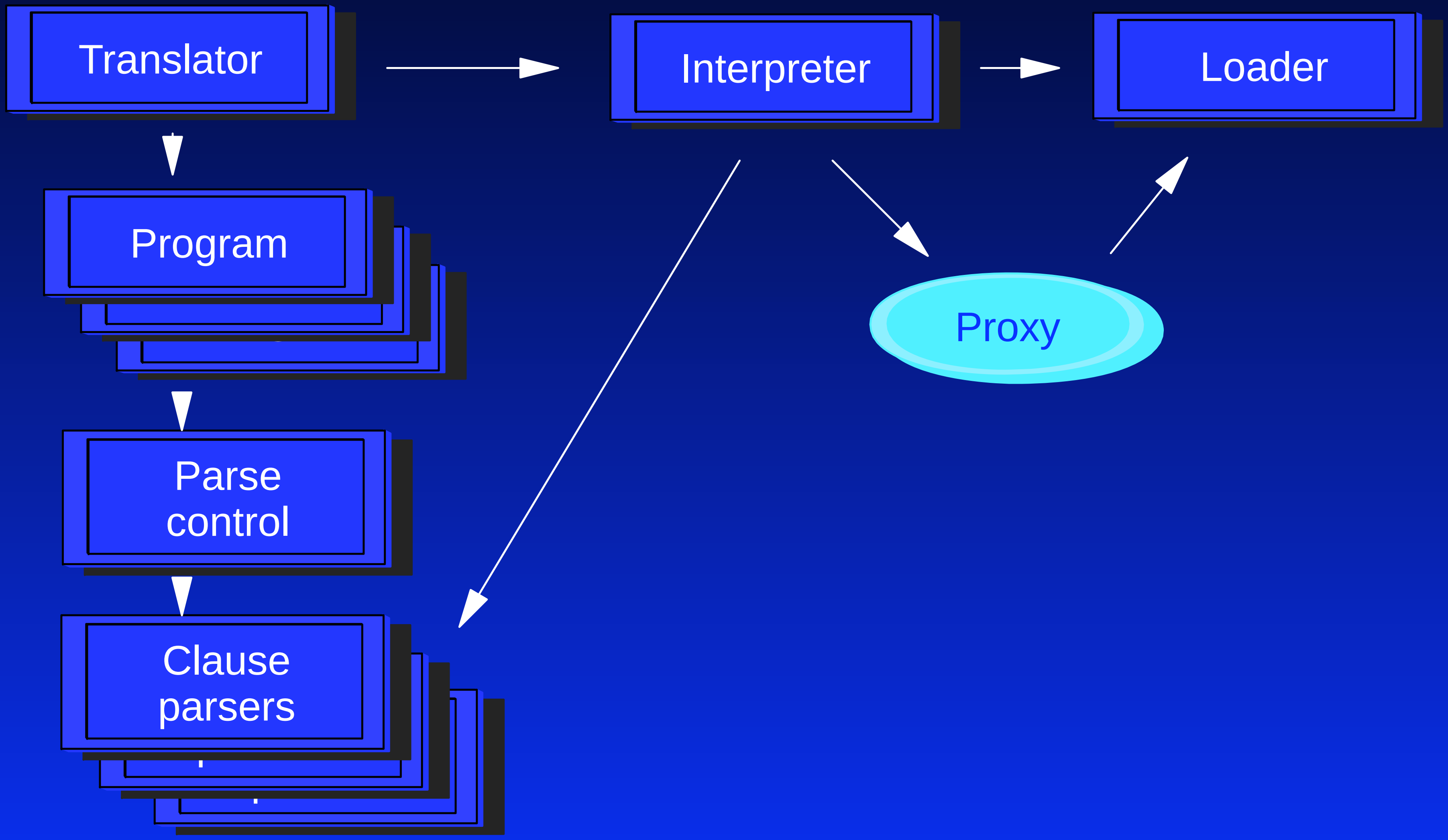
Interpreter complications

- Signals -- have to be wrapped, and cannot be passed through a reflection call

- Constructors -- arguments to super(x, y) call must be interpreted, then the super(x, y) call must be made by the proxy class, and only then can the constructor method body be interpreted

- Protected (synchronized) blocks of code must truly be protected to be thread-safe

Interpretation

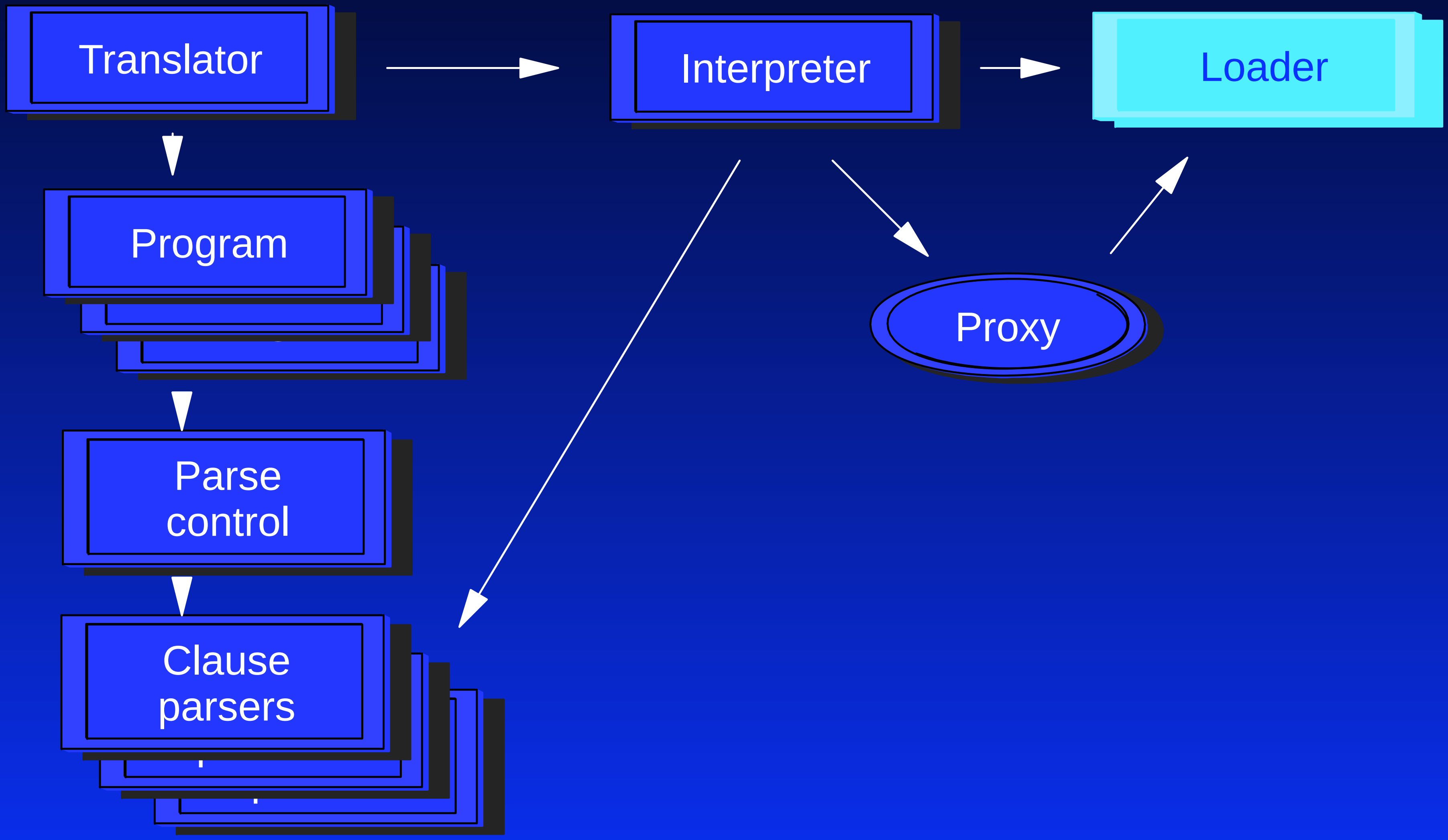


Proxy class

- Builds a binary class image (in a byte array) for a class that is to be interpreted

- Tedious but relatively straightforward - the code for every method is essentially the same
  - collect arguments (wrapped if necessary) into an Object array
  - invoke the interpreter to interpret the method body
  - get the returned Object; unwrap or cast it as required, and return it to caller

Interpretation



Proxy class Loader

- A Java classloader is needed to actually load a class into the JVM
- If the built-in one were used then a class could never be redefined; classes are only unloaded when the object that loaded them is unloaded
- Complication: we also have to load any external (compiled) private classes, as otherwise they appear to be in a different package and hence would not be accessible when they should be

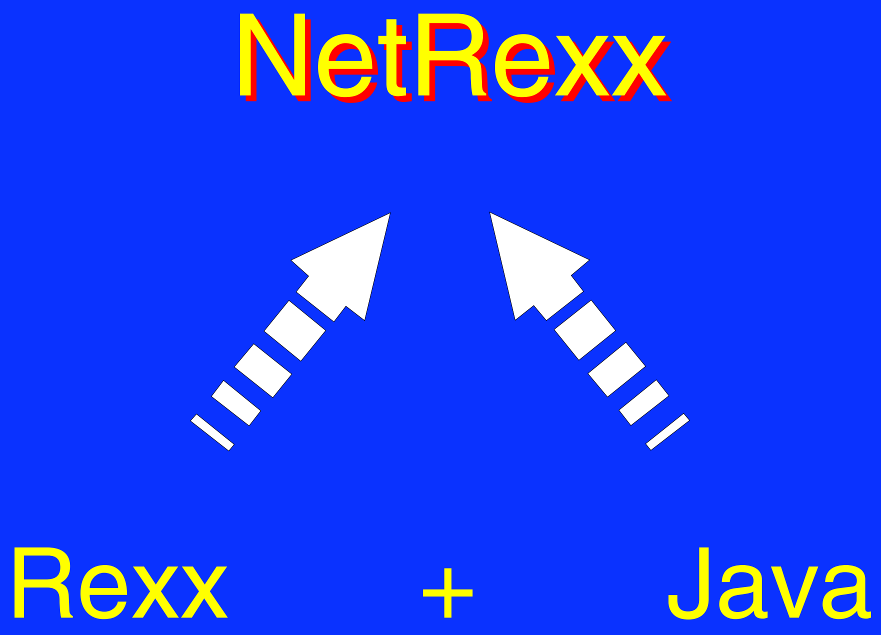
Summary

- True interpretation of JVM-based languages *can* be done
- The primary benefit is development productivity
- Using a single language for scripting and application development is a reality

Questions?

... Please fill in your evaluation form!

http://www2.hursley.ibm.com/netrexx/



NetRexx

Rexx + Java

Strong typing doesn’t need extra typing