

Eagle: Tcl Implementation in C#

Alt.NET 2011 (Seattle)

Joe Mistachkin



```
#  
# NOTE: Rapid access to data.  
#  
set conn [sql open \  
  *Data Source={local};database=master;user=root;password=root;  
#  
sql execute -execute reader some "select * from table1"  
#  
# NOTE: Full access to Tcl and extensions.  
#  
tcl load; set interp [tcl create]  
tcl eval $interp \  
  package require tk  
  package require expect  
  |
```

What is Tcl?

- Tcl (Tool Command Language) is an open-source scripting language created by John Ousterhout in 1988.
- Designed to be highly extensible and easily embeddable.
- Is it relatively “typeless”, has minimal syntax, no fixed grammar, and no keywords.

Who uses Tcl/Tk?

- America Online
- ActiveState
- BAE Systems
- BMW
- BitMover
- Boeing
- Broadcom
- Cisco
- DaimlerChrysler
- EDS
- Eolas Technologies
- F5 Networks
- HP
- IBM
- Intel
- Lucent
- Mentor Graphics
- Microsoft (?)
- Motorola
- NASA (JPL and others)
- NBC
- National Instruments
- Northrop Grumman
- Oracle
- Pixar
- Python
- QUALCOMM
- Raytheon
- Sun Microsystems
- Synopsys
- Texas Instruments
- TiVo
- US Department of Defense
- US National Institute of Standards and Technology
- US Postal Service
- ...

What is Eagle?

- Eagle or “Extensible Adaptable Generalized Logic Engine” is an open-source implementation of the Tcl scripting language written in C# for the CLR.
- Designed to be highly extensible and easily embeddable.
- Is it relatively “typeless”, has minimal syntax, no fixed grammar, and no keywords.
- Supports approximately 90% of the Tcl 8.4 core command set.

What is Eagle, really?

- Originally designed for the purpose of providing a first-class library for scripting applications written for the CLR.
- Eagle is the “glue” that allows various and diverse components to work together to accomplish a given task.
- It combines the best things about Tcl/Tk and PowerShell into one language.
- All other considerations were secondary, including performance and full Tcl compatibility.

What Eagle is not.

- Not based on the Microsoft Dynamic Language Runtime (DLR).
- Not intended for stand-alone application development (i.e. writing a large-scale application using the Eagle language by itself is not recommended).
- Unlikely to ever have a compiler.
- Not really a “drop-in” replacement for Tcl or Jacl.

Notable Features

- Dynamic language, loosely coupled with full introspection.
- Uses “duck typing” (like Tcl).
- Supports Unicode (obviously?).
- Integrates with CLR classes.
- Integrates with native libraries.
- Integrates with Tcl/Tk.
- Supports interactive debugging.
- Supports script cancellation.
- Supports read-only variables, commands, etc.
- Supports interpreter-wide variable tracing.
- Supports anonymous procedures and closures.
- Unified unit testing framework.
 - Capable of running tests in Tcl and Eagle.

Notable Features

(continued)

- Support for “custom commands” (more on this later).
 - All commands may be added, renamed, “subclassed”, or removed at any time and can be implemented in any managed language.
- Support for custom math functions.
 - For use with the expression parser (e.g. the [**expr**] command).
- Supports binary plugins (i.e. groups of commands).
 - The default plugin provides all the necessary boilerplate code to integrate with Eagle; however, plugins are free to override any default behaviors.
- Supports versioned “packages” which may refer to a binary plugin or a “pure script” package (as in Tcl).
- Supports “hidden” commands.
 - Preliminary support has been added for “safe” interpreter support and custom command execution policies written in managed code.

Syntax: Grouping

(evaluation, step #1)

```
set x 1
```

```
puts "grouping with quotes, x = $x"
```

```
puts {grouping with braces, x = $x}
```

```
proc argsProc { args } {  
  return [info level [info level]]  
}
```

```
puts [argsProc with brackets, x = $x]
```

Syntax: Substitution

(evaluation, step #2)

```
set name Joe; puts "Hello, $name"
```

```
puts [clock format [clock seconds]]
```

```
puts "\u2554\u2550\u2557\n\u2551\u2551\n\u2551\n\u255A\u2550\u255D"
```

Friendly Errors

```
% set
```

```
Error, line 1: wrong # args: should  
be "set varName ?newValue?"
```

```
% scope foo
```

```
Error, line 1: bad option "foo": must  
be close, create, current, destroy,  
eval, exists, list, open, set,  
unset, or vars
```

CLR Usage Example

(automation)

```
proc threadStart { args } {  
    set ::result $args; # do work here...  
}
```

```
object import System.Threading
```

```
set t [object create -alias -parameterTypes \  
    ParameterizedThreadStart Thread threadStart]
```

```
$t Start foo; $t Join; unset t
```

```
$::result ToString
```

Expressiveness == Power

```
object load -import System.Windows.Forms
```

```
proc threadstart {} {  
  [object create -alias Form] Show  
  after 0 nop; vwait ::forever  
}
```

```
[object create -alias \  
  System.Threading.Thread threadStart] \  
Start
```

Native Library Example

(more automation)

```
set x [library declare -functionname \  
      GetConsoleWindow -returntype IntPtr \  
      -module [library load kernel32.dll]]
```

```
set hwnd [library call $x]
```

Tcl/Tk Example

(integration)

```
if {[isEagle]} then {  
  # NOTE: very dynamic link library calls  
  set hWnd [library declare -functionname GetWinProc  
  IntPtr -module [library load kernel32.dll  
  # NOTE: Automate .NET Framework classes  
  object load  
  set form [tcl load  
  $form Text "hello world"; form show  
  # NOTE: Rapid access to data  
  #  
  tcl eval $interp {eagle debug break}; # type "#go"  
  tcl unload
```

Custom Commands

- Together with minimal syntax and the ability to remove add, rename, or remove any command at any time, these are great for creating application-centric domain specific languages (DSL).
- Show example...

Interactive Debugging

- Single-step mode, breakpoints, and variable watches.
- Examine and modify interpreter state.
- Supports isolated evaluation.
- Scripting support via the **[debug]** command.

Interactive Debugging

(continued)

- Uses a customizable read-eval-print loop (REPL).
- Debugging “meta-commands” are prefixed with “#” (the Tcl comment character) having special meaning (i.e. they work) only when entered interactively.
- The **#help** meta-command may be used [by itself] to display the list of available meta-commands or with the syntax **#help <name>** to display usage information for a particular meta-command (e.g. **#help #vinfo**).
- Not yet integrated with Visual Studio.

Script Cancellation

(oddly similar to TIP #285, see <http://tip.tcl.tk/285>)

- Safely cancels a script being evaluated, synchronously or asynchronously.
- Example #1 (from C#):

```
Engine.CancelEvaluate(interpreter,  
true, null, ref result);
```

- Example #2 (from a script):

```
interp cancel -unwind
```

Variable Tracing

- Allows user-defined callback(s) to be executed when a variable is read, written, or deleted.
 - Currently, trace callbacks for a variable can only be specified upon variable creation (via the `SetVariableValue` or `AddVariable` APIs).
- Interpreter-wide variable traces are also supported.
 - They can monitor, modify, or cancel all requests to read, write, and delete any variable in the interpreter.

Anonymous Procedures

(compatible with Tcl 8.5)

```
set sum [list [list args] {  
    expr [list [join $args +]]  
}]
```

```
apply $sum 1 2 3; # returns 6
```

Closures

(using `[apply]` and `[scope]`)

```
set sum [list [list name args] {  
  scope create -open -args $name  
  if {[info exists sum]} then {  
    set sum 0  
  }  
  if {[length $args] > 0} then {  
    incr sum [expr [list [join $args +]]]  
  }  
}]
```

```
apply $sum foo 1 2 3; # returns 6
```

Design Philosophy

- Tcl heavily influenced the design of Eagle. In particular:
 - It obeys the “Endekalogue” (i.e. the “11 rules” that make up the Tcl 8.4 syntax).
 - Everything is a string (EIAS).
 - Every command is a string; the first word is the name of the command and the rest are arguments to the command.
 - Commands are free to interpret their arguments however they wish.
 - Every list is a string; however, not every string is a “well-formed” list.
 - The language supplies primitives “pieces” that can be combined and/or composed in useful ways.

Design Philosophy

(continued)

- However, there are some differences in the design that reflect the differences in the underlying platforms:
 - Minimal per-thread data; most state is stored in the interpreter.
 - Interpreters may be used from any thread and/or from multiple threads simultaneously (i.e. they have no thread affinity).
 - Each thread does have its own call stack and current call frame; however, the global call frame is shared between all threads.
 - No interpreter-wide result (i.e. the result of the last evaluation, etc).
 - This merits special attention because it significantly reduces the coupling between components.

What is missing?

(from the Tcl/Tk perspective)

- No Tk commands.
- No argument expansion syntax (introduced in Tcl 8.5).
- No namespace support (except the global namespace).
- No asynchronous input/output.
- No **[binary]**, **[fblocked]**, **[fileevent]**, **[format]**, **[glob]**, **[history]**, **[memory]**, **[scan]**, or **[trace]** commands.
- No **http** or **msgcat** packages.
- No **registry** or **dde** packages.
- Minimal support for the **tcltest** package (just enough to run the test suite).
- For the **[open]** command, command pipelines and serial ports are not supported.
- For the **[exec]** command, Unix-style input/output redirection and command pipelines are not supported.

Performance

(from the Tcl/Tk perspective)

- For some operations, Eagle can be up to up to two orders of magnitude slower than “real” Tcl.
- This is to be expected because Tcl is written in highly optimized C, has a mature byte-code compiler, and [most importantly] caches the computed internal representations of lists, integers, etc.

Where is the innovation?

- The “host application is always right” attitude.
 - The IHost interface, etc.
- One interpreter, multiple threads, safely.
- The “universal option parser”.
- The seamless integration with CLR objects.
 - Full support for overload resolution, properties, methods, fields, and events.
- The built-in debugging support.
 - Yes, it’s not as smooth as Visual Studio; however, it is complete, lightweight, and does not need to be installed to function.
- Simple development / deployment model (i.e. the “add a reference and go experience”).

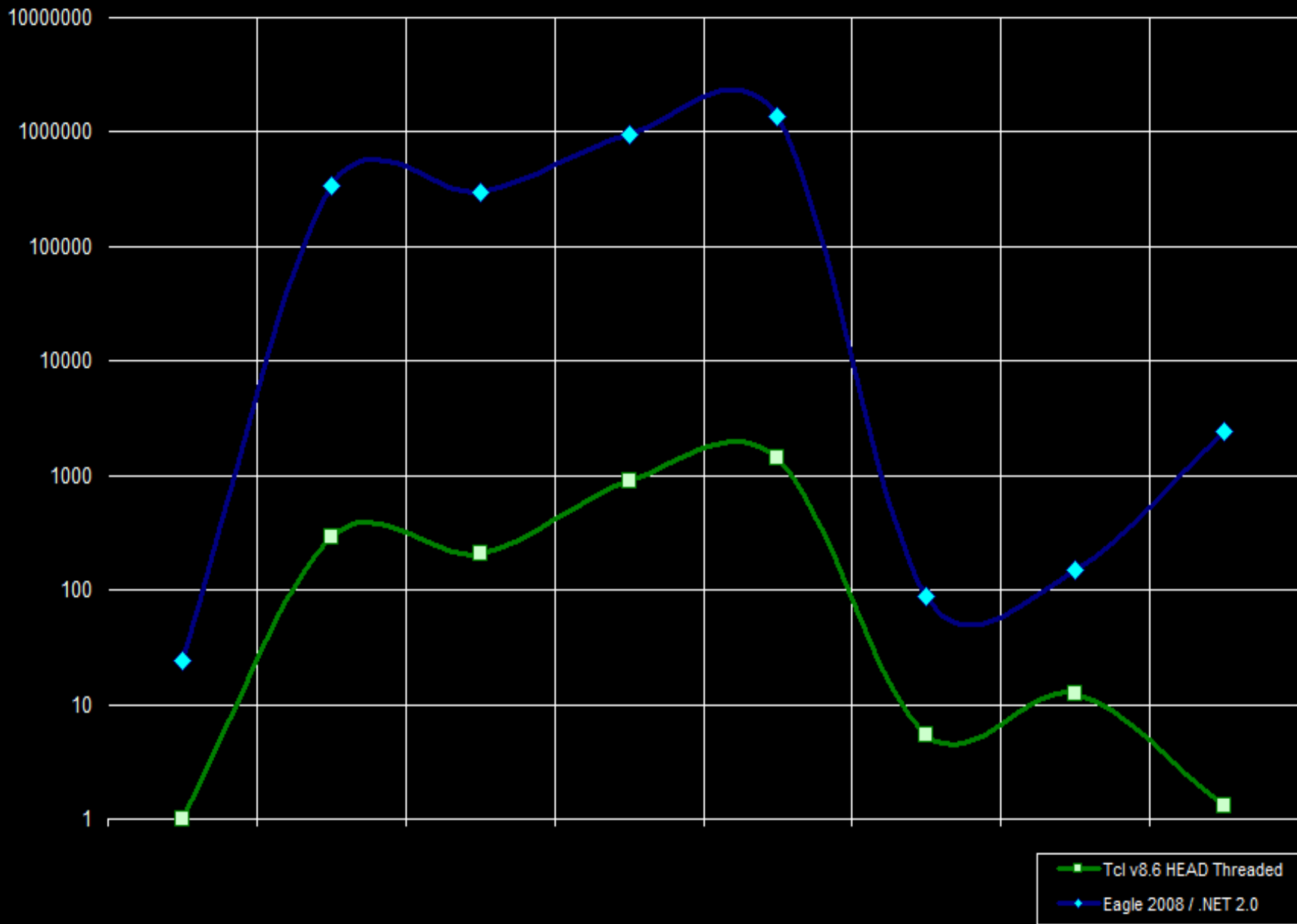
Why no compiler?

- Not enough time in the original schedule.
- Raw performance was not a primary consideration.
- Being correct, complete, and dynamic is more important than being fast.
- Long running scripts can be evaluated (and canceled as necessary) in secondary threads.
- The CLR just-in-time compiler is already pretty good.

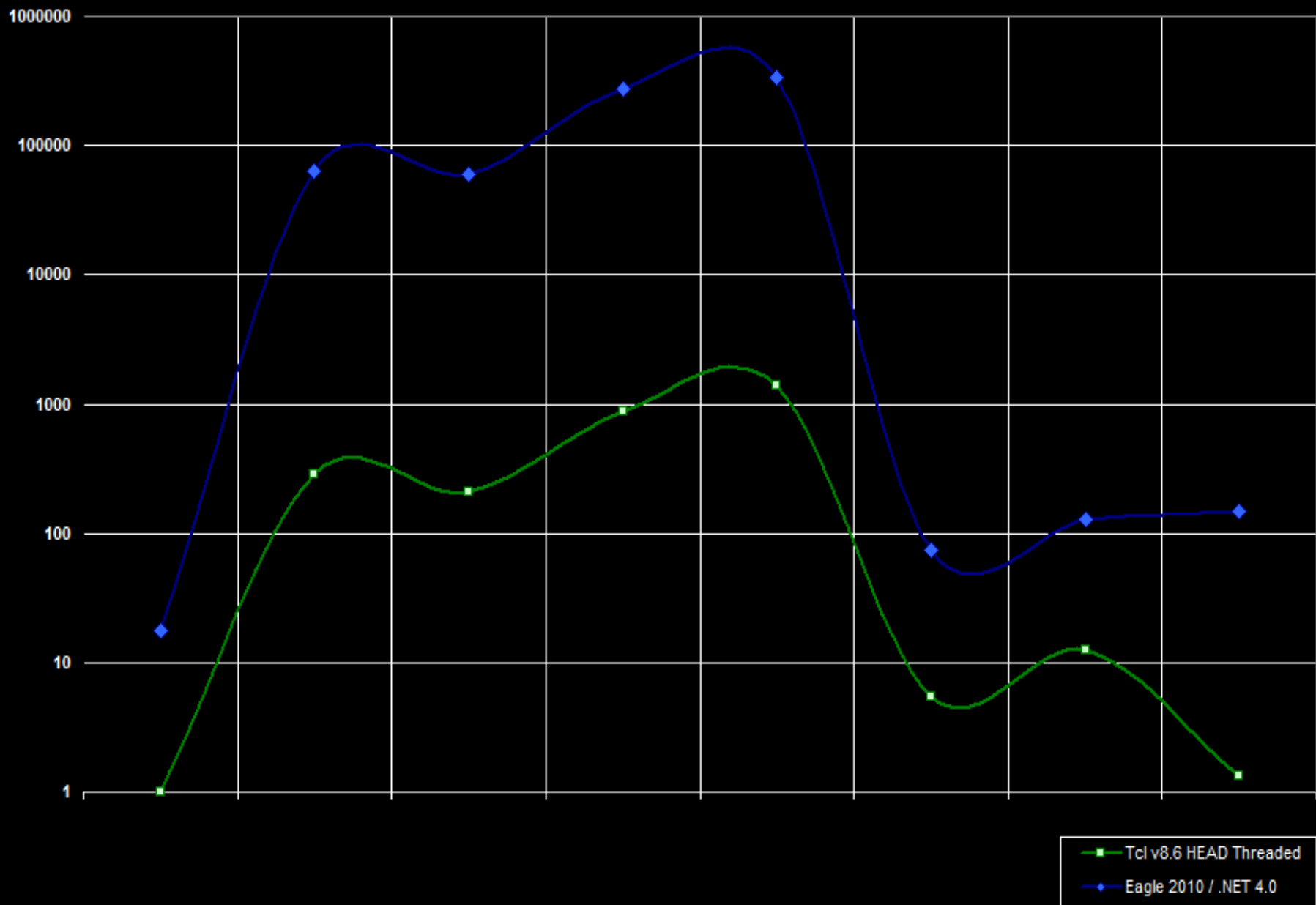
Performance Problems

- Can be much slower than native Tcl, even for the simplest operations.
- Over time, targeted optimizations have been added for all critical code paths.

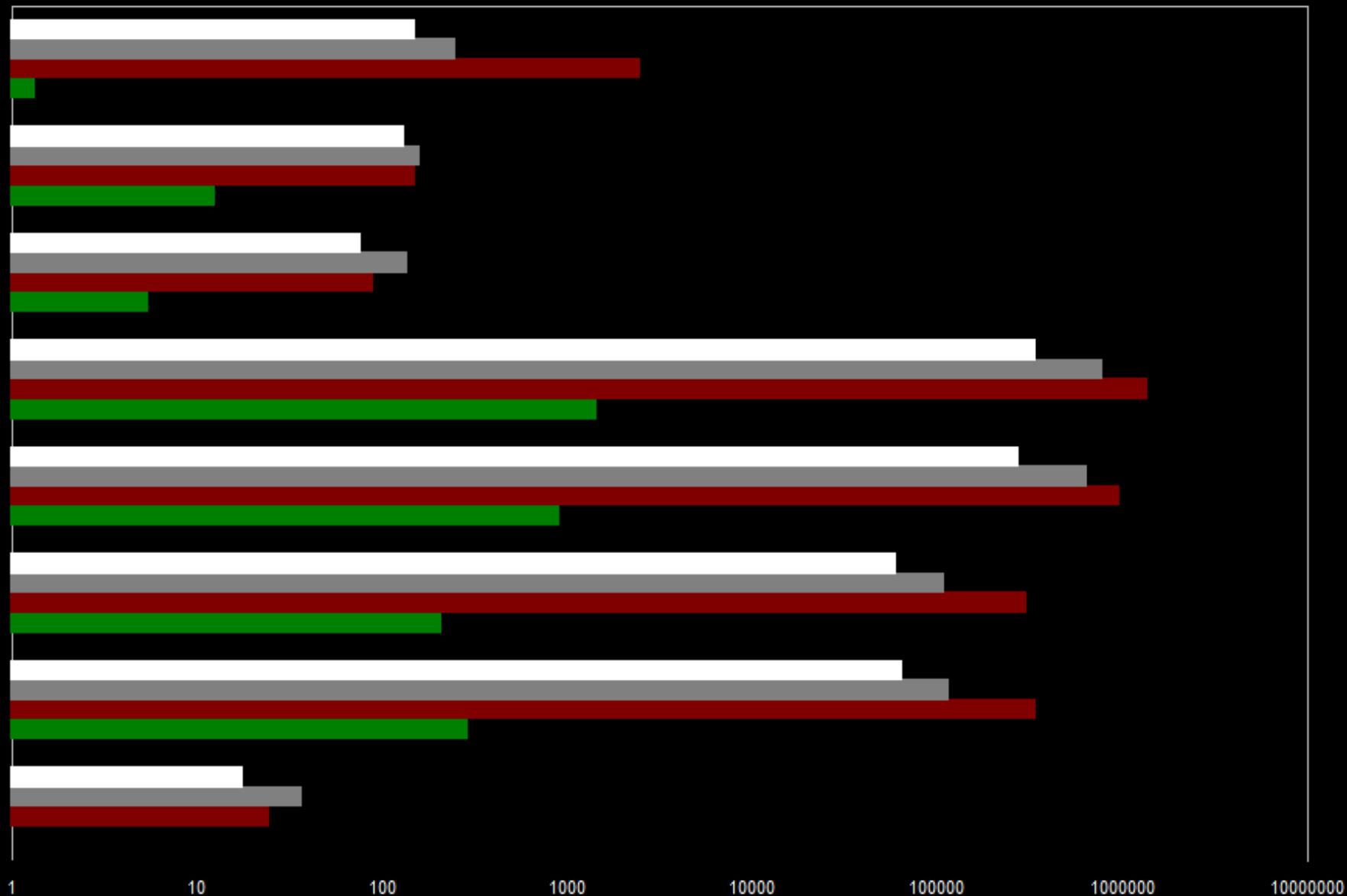
Tcl (HEAD) vs. Eagle 2008



Tcl (HEAD) vs. Eagle 2010



Tcl (HEAD) vs. Eagle



Tcl v8.6 HEAD Threaded

Eagle 2008 / .NET 2.0

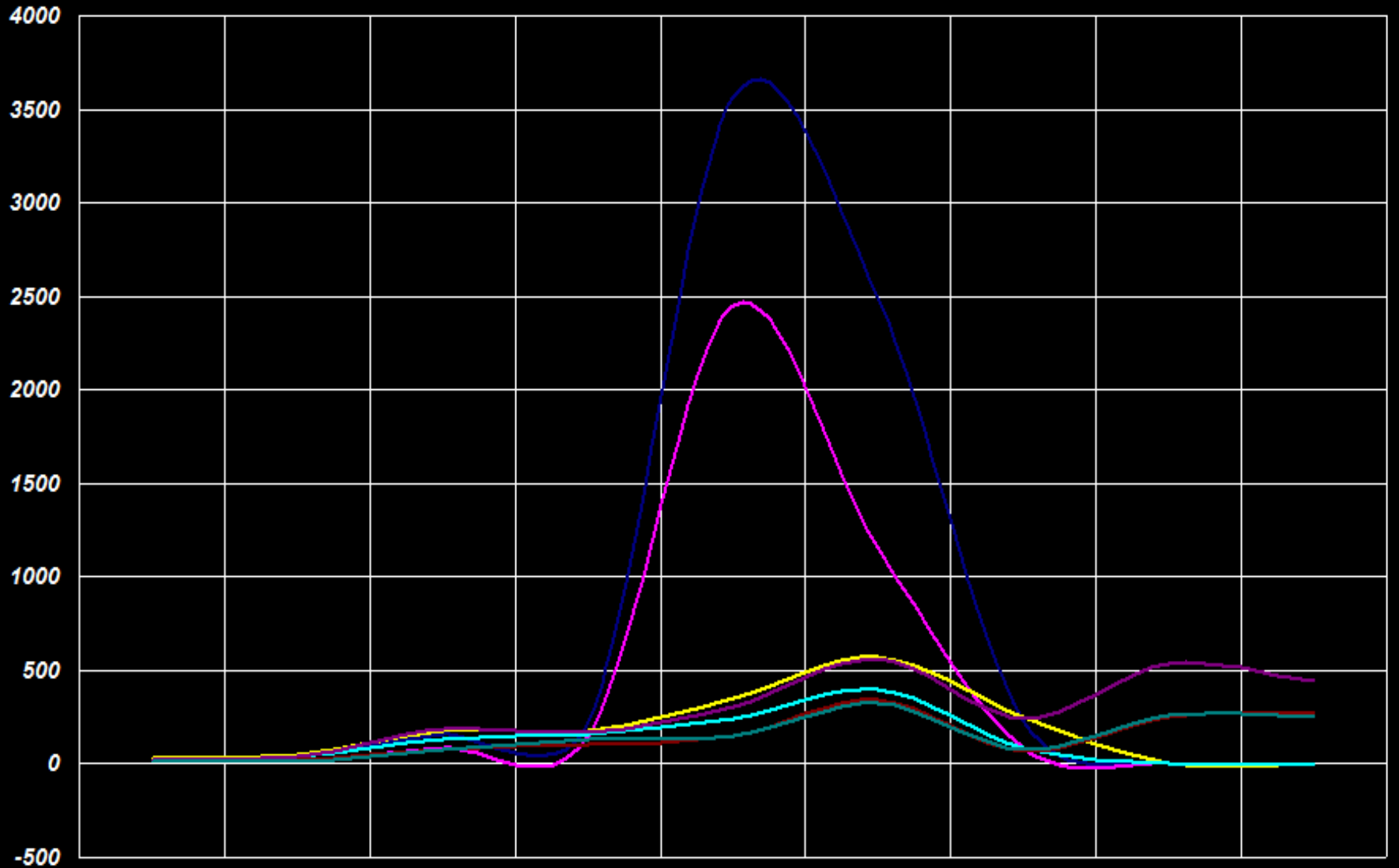
Eagle 2009 / .NET 2.0

Eagle 2010 / .NET 4.0

What is slow?

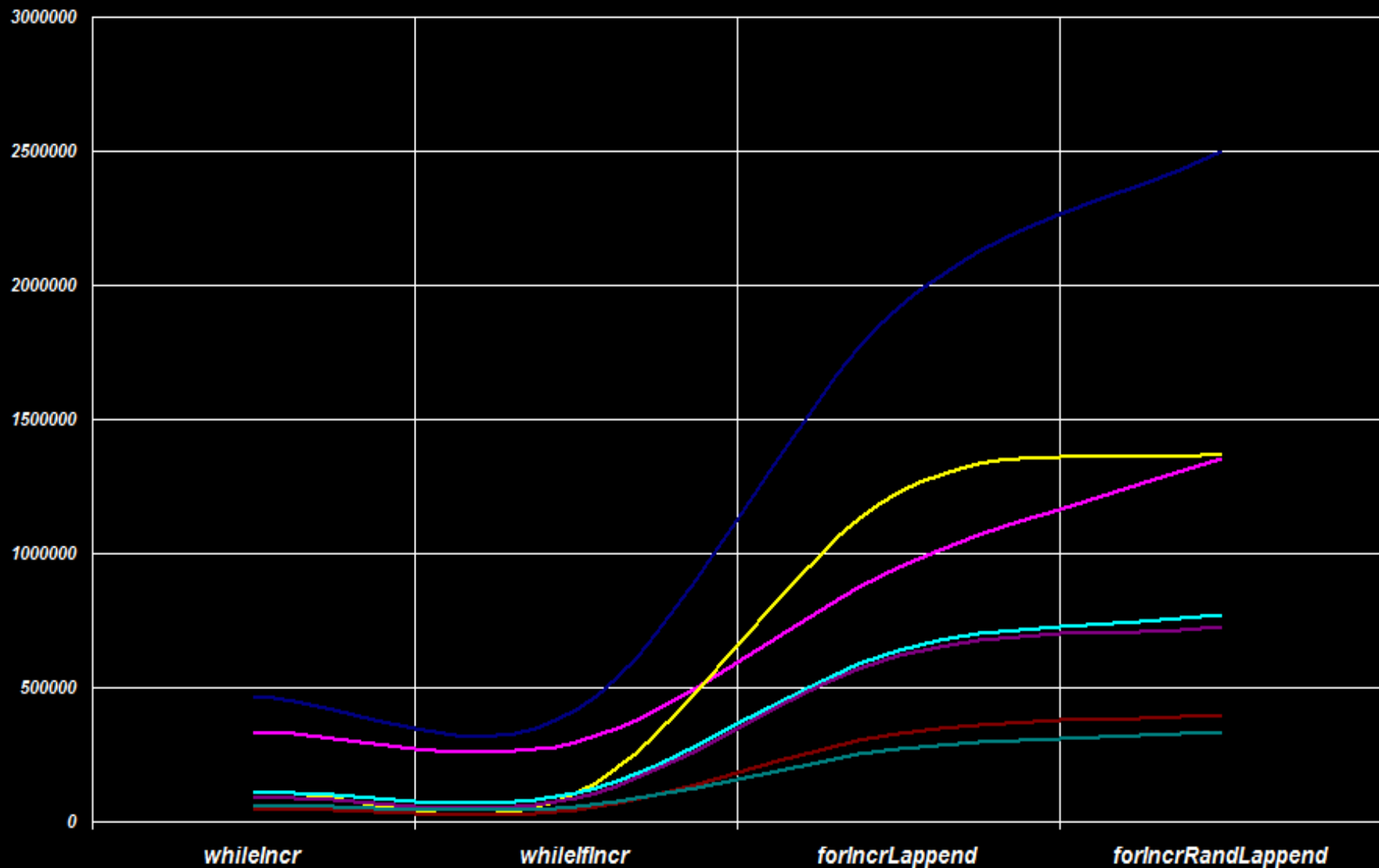
- Parsing strings into lists.
- Building lists from strings.
- Expression evaluation, primarily string-to-type conversions.
- All other performance issues are insignificant compared to these three.

Eagle Performance



— Eagle 2008 / Mono 2.6 — Eagle 2008 / .NET 2.0 — Eagle 2009 / Mono 2.6 — Eagle 2009 / .NET 2.0
— Eagle 2010 / Mono 2.6 — Eagle 2010 / .NET 2.0 — Eagle 2010 / .NET 4.0

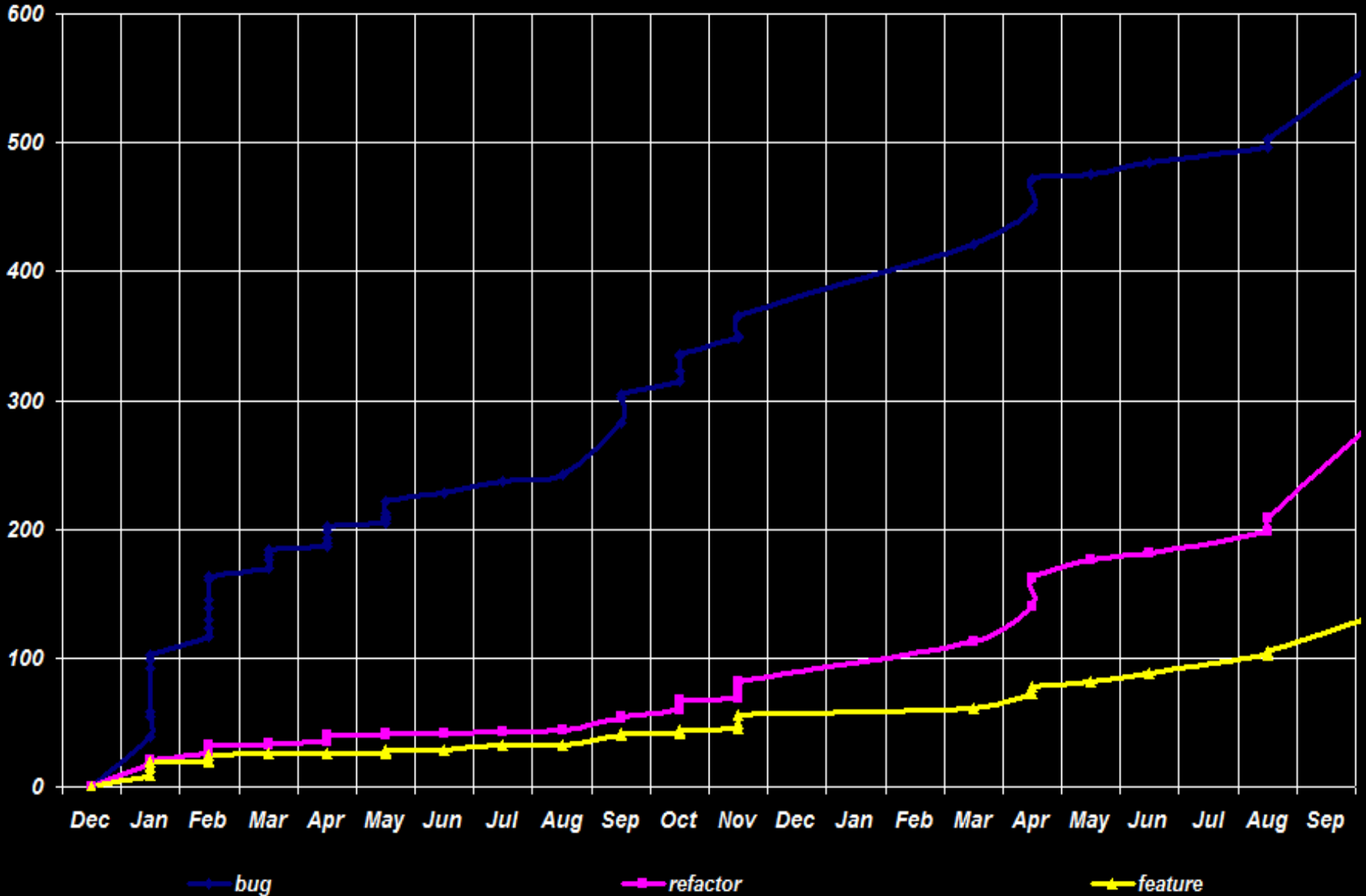
Eagle Loop Performance



Legend:

- Eagle 2008 / Mono 2.6
- Eagle 2008 / .NET 2.0
- Eagle 2009 / Mono 2.6
- Eagle 2009 / .NET 2.0
- Eagle 2010 / Mono 2.6
- Eagle 2010 / .NET 2.0
- Eagle 2010 / .NET 4.0

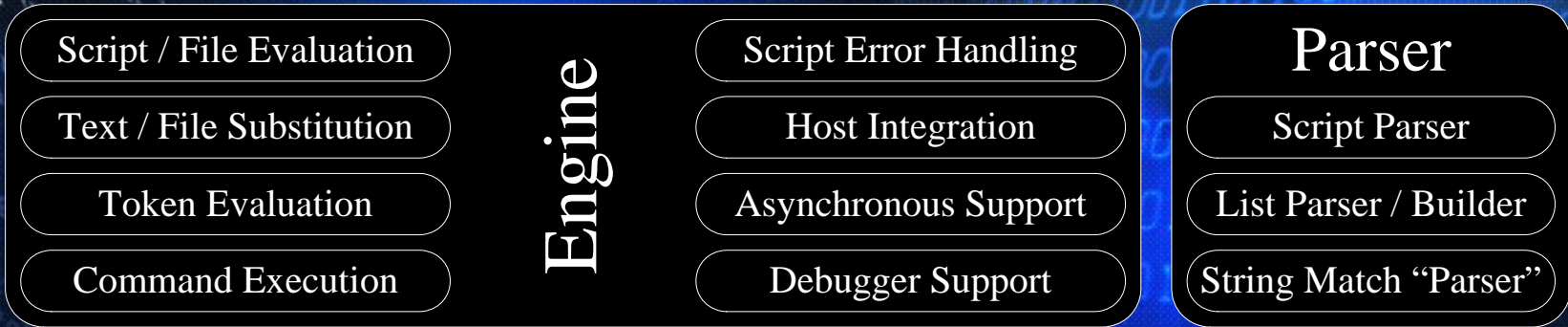
Eagle Changes Over Time



Architectural Problems, Part 1

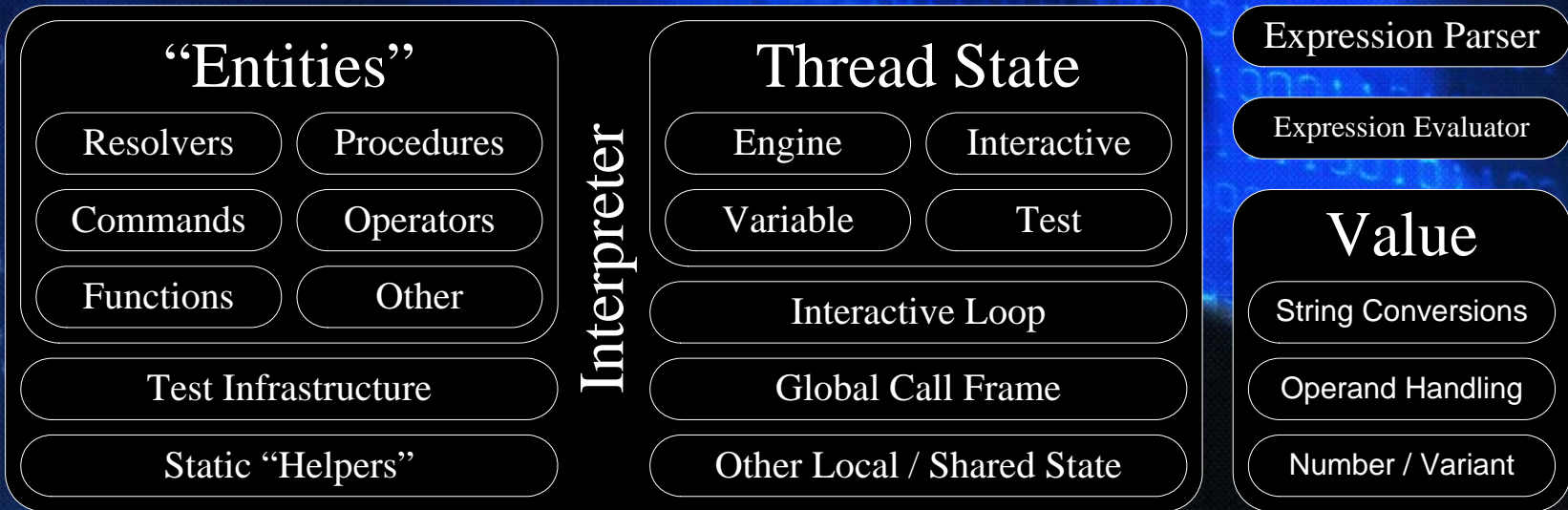
- The interpreter class is far too large.
 - Break into multiple files or classes.
 - Move all entity management to a new component.
 - The interactive loop code is too large.

Major Components



Option Parser

Object Marshaller



Host Integration

Tcl/Tk Integration

Architectural Problems, Part 2

- The engine is too recursive.
 - There would potentially be a lot of benefits from something like NRE.
- The components are too tightly coupled and have some circular dependencies.

The Mono Saga

- Mono support was added in 2009.
- It has never been perfect because of serious bugs in the Mono platform.
- Eagle should build and run correctly on recent versions of Mono (e.g. 2.6.7 and 2.10) for Windows and Unix.

How can I help?

- Test in your environments and report any issues you find.
- Contribute to the documentation and/or the test suite.
- Provide feedback, suggest features, or flames.

Where is it?

<https://eagle.to/>

Questions and Answers

```
if {[lsEagle]} then {
# NOTE: Very dynamic link library call.
set hWnd [library declare -functionname GetDlgItem
IntPtr -module [library load kernel32.dll]
# NOTE: Automate .NET Framework classes,
object load -import System.Windows.Forms
set form [object create alias Form]
$form Text "hello world"
# NOTE: Rapid access to data.
set conn [sql open \
"Data Source=(local);Database=master;User=sa;Password=;"]
sql execute -execute reader some "select * from sysobjects"
# NOTE: Full access to Tcl and extensions.
tcl load; set interp [tcl create]
tcl eval $interp {
package require Tk
package require Expect
}
```