Building Ruby in Smalltalk

Martin McClure

ESUG 2009


MAGLEV™

Why?

...and more

GemStone for Really Smart Dummies

GemStone/S

- Smalltalk Implementation
- Headless
- Shared transactional persistence
  - Shared “image”
  - Commit merges concurrent changes
  - Large
    - Thousands of concurrent VMs
    - Terabyte scale
  - Fast
    - 10K commits/sec

GemStone/S

It's Just Smalltalk

Transparent Persistence for Web Applications

GemStone•Linux•Apache•Seaside•Smalltalk

Version1.0 alpha 1

GemStone S 64™

The Big Choice

Challenge 1: The Compiler

Smalltalk Grammar

```
AExpression = Primary [ AMessage { ';' ACascadeMessage } ]
ABinaryMessage = ABinarySelector Primary [ UnaryMessages ]
ABinaryMessages = ABinaryMessage { ABinaryMessage }
ACascadeMessage = UnaryMessage | ABinaryMessage | AKeyWordMessage
AKeyWordMessage = AKeyWordPart { AKeyWordPart }
AKeyWordPart = KeyWord Primary UnaryMessages { ABinaryMessage }
AMessage = [UnaryMessages] [ABinaryMessages] [AKeywordMessage]
Array = '(' { ArrayItem } ')'
ArrayBuilder = '#[' [ AExpression { ',' AExpression } ] ']'
ArrayLiteral = '#' Array
CurlyArrayBuilder = '{' [ AExpression { '.' AExpression } ] '}'
ArrayItem = Number | Symbol | SymbolLiteral | StringLiteral |
        CharacterLiteral | Array | ArrayLiteral
Assignment = VariableName ':=' Statement
BinaryMessage = BinarySelector Primary [ UnaryMessages ]
BinaryMessages = BinaryMessage { BinaryMessage }
BinaryPattern = BinarySelector VariableName
Block = '[' [ BlockParameters ] [ Temporaries ] Statements ']'
BlockParameters = { Parameter } '|'
CascadeMessage = UnaryMessage | BinaryMessage | KeyWordMessage
Expression = Primary [ Message { ';' CascadeMessage } ]
KeyWordMessage = KeyWordPart { KeyWordPart }
KeyWordPart = KeyWord Primary UnaryMessages { BinaryMessage }
KeyWordPattern = KeyWord VariableName {KeyWord VariableName}
Literal = Number | NegNumber | StringLiteral | CharacterLiteral |
       SymbolLiteral | ArrayLiteral | SpecialLiteral
Message = [UnaryMessages] [BinaryMessages] [KeyWordMessage]
MessagePattern = UnaryPattern | BinaryPattern | KeyWordPattern
Method = MessagePattern [ Primitive ] MethodBody
MethodBody = [ Pragmas ] [ Temporaries ] [ Statements ]
NegNumber = '-' Number
Operand = Path | Literal | Identifier
Operator = '=' | '==' | '<' | '>' | '<=' | '>=' | '~=' | '~~'
ParenStatement = '(' Statement ')'
Predicate = ( AnyTerm | ParenTerm ) { '&' Term }
Primary = ArrayBuilder | CurlyArrayBuilder | Literal | Path | Block | SelectionBlock |
       ParenStatement | VariableName
Primitive = '<' [ 'protected' | 'unprotected' ] [ 'primitive:' Digits ] '>'
Pragmas =  Pragma [ Pragma ]
Pragma = '< PragmaBody '>'
PragmaBody =  UnaryPragma | KeywordPragma
UnaryPragma  = SpecialLiteral | UnaryPragmaIdentifier
KeywordPragma = PragmaPair [ PragmaPair ]
PragmaPair =  [ KeywordNotPrimitive | BinarySelector ] PragmaLiteral
KeywordNotPrimitive  is any  Keyword  other than  'primitive:'
UnaryPragmaIdentifier is any  Identifier  except 'protected' , 'unprotected' , 'requiresVc'
PragmaLiteral = Number | NegNumber | StringLiteral | CharacterLiteral |
               SymbolLiteral | SpecialLiteral
```

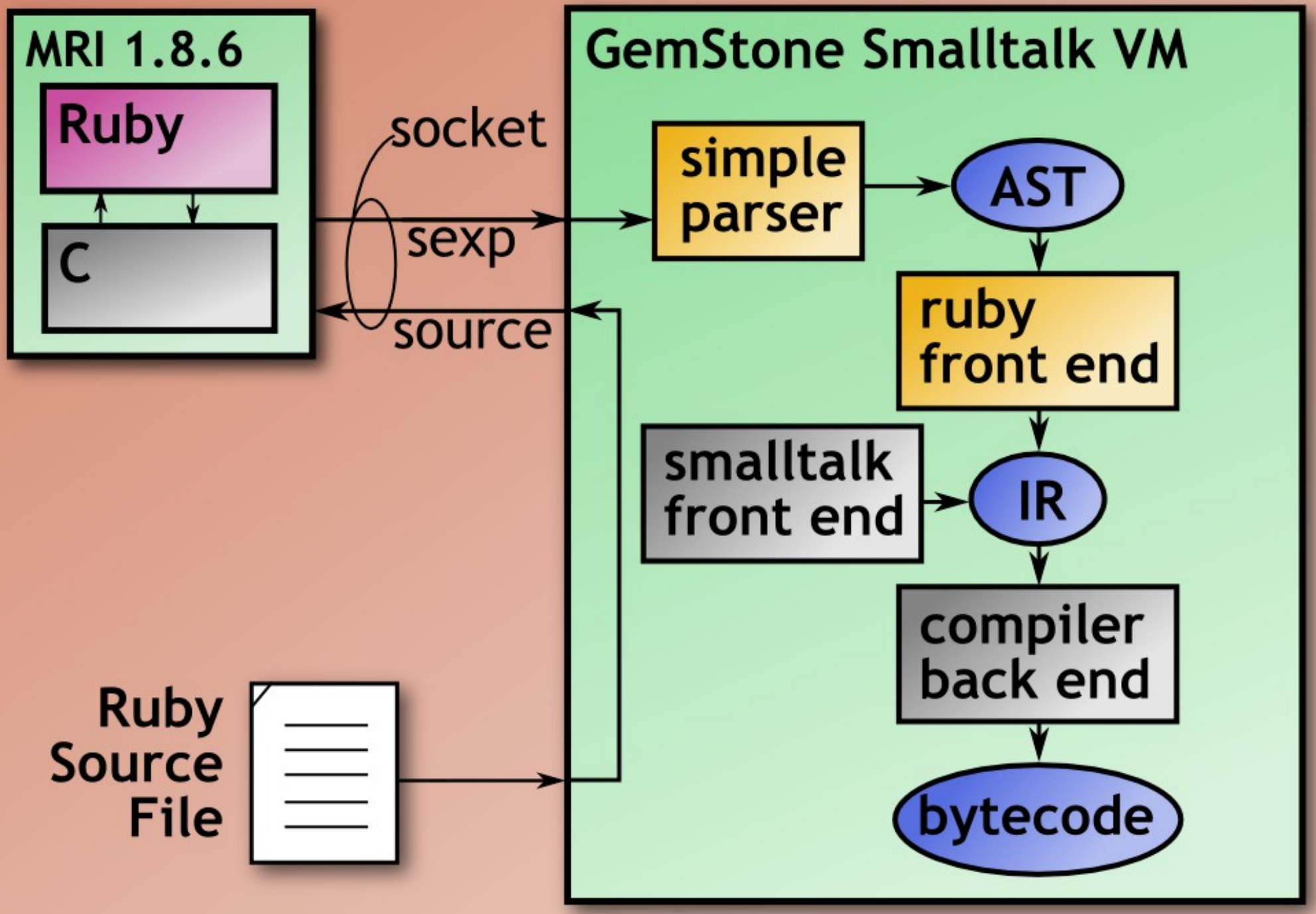
```
SelectionBlock = '{' Parameter } '|' Predicate '}'
Statement = Assignment | Expression
Statements = { [ Pragmas] { Statement '.' } } [ Pragmas] [ ['^'] Statement ['.'
[ Pragmas] ]]
Temporaries = '|' { VariableName } '|'
ParenTerm = '(' AnyTerm ')'
Term = ParenTerm | Operand
AnyTerm = Operand [ Operator Operand ]
UnaryMessage = Identifier
UnaryMessages = { UnaryMessage }
UnaryPattern = Identifier


ABinarySelector = any BinarySelector except comma
BinaryExponent = ( 'e' | 'E' | 'd' | 'D' ) ['-' | '+'] Digits
BinarySelector = ( SelectorCharacter [SelectorCharacter]) |
          ( '-' [ SelectorCharacter ] )
Character = Any Ascii character with ordinal value 0..255
CharacterLiteral = '$' Character
Comment = '"' { Character } '"'
DecimalExponent = ( 'f' | 'F' ) ['-' | '+'] Digits
Digit = '0' | '1' | '2' | ... | '9'
Digits = Digit {Digit}
Exponent = BinaryExponent | DecimalExponent | ScaledDecimalExponent
FractionalPart = '.' Digits [Exponent]
Identifier =  SingleLetterIdentifier | MultiLetterIdentifier
KeyWord = Identifier ':'
Letter = 'A' | 'B' | ... | 'Z' | 'a' | 'b' | ... | 'z' | '_'
MultiLetterIdentifier = Letter { Letter | Digit }
Number = RadixedLiteral | NumericLiteral
Numeric = Digit | 'A' | 'B' | ... | 'Z'
NumericLiteral = Digits ( [FractionalPart] | [Exponent] )
Numerics = Numeric { Numeric }
Parameter = ':' VariableName              [NOTE: white space allowed
                                    between  :  and variableName ]
Path = Identifier '.' PathIdentifier { '.' PathIdentifier }
PathIdentifier  =  Identifier  |  '*'
RadixedLiteral = Digits ( '#' | 'r' ) ['-'] Numerics
ScaledDecimalExponent =  's' ['-' | '+'] Digits
SelectorCharacter = '+' | '\' | '*' | '~' | '<' | '>' | '='
          | '|' | '/' | '@' | '%' | ',' | '?' | '!'
SingleLetter 'A' | 'B' | ... | 'Z' | 'a' | 'b' | ... | 'z'
SingleLetterIdentifier =  SingleLetter
SpecialLiteral = 'true' | 'false' | 'nil' | '_remoteNil'
StringLiteral = '"' { Character | '""' } '"'
Symbol = Identifier | BinarySelector | ( Keyword { Keyword } )
SymbolLiteral = '#' ( Symbol | StringLiteral )
VariableName = Identifier
```

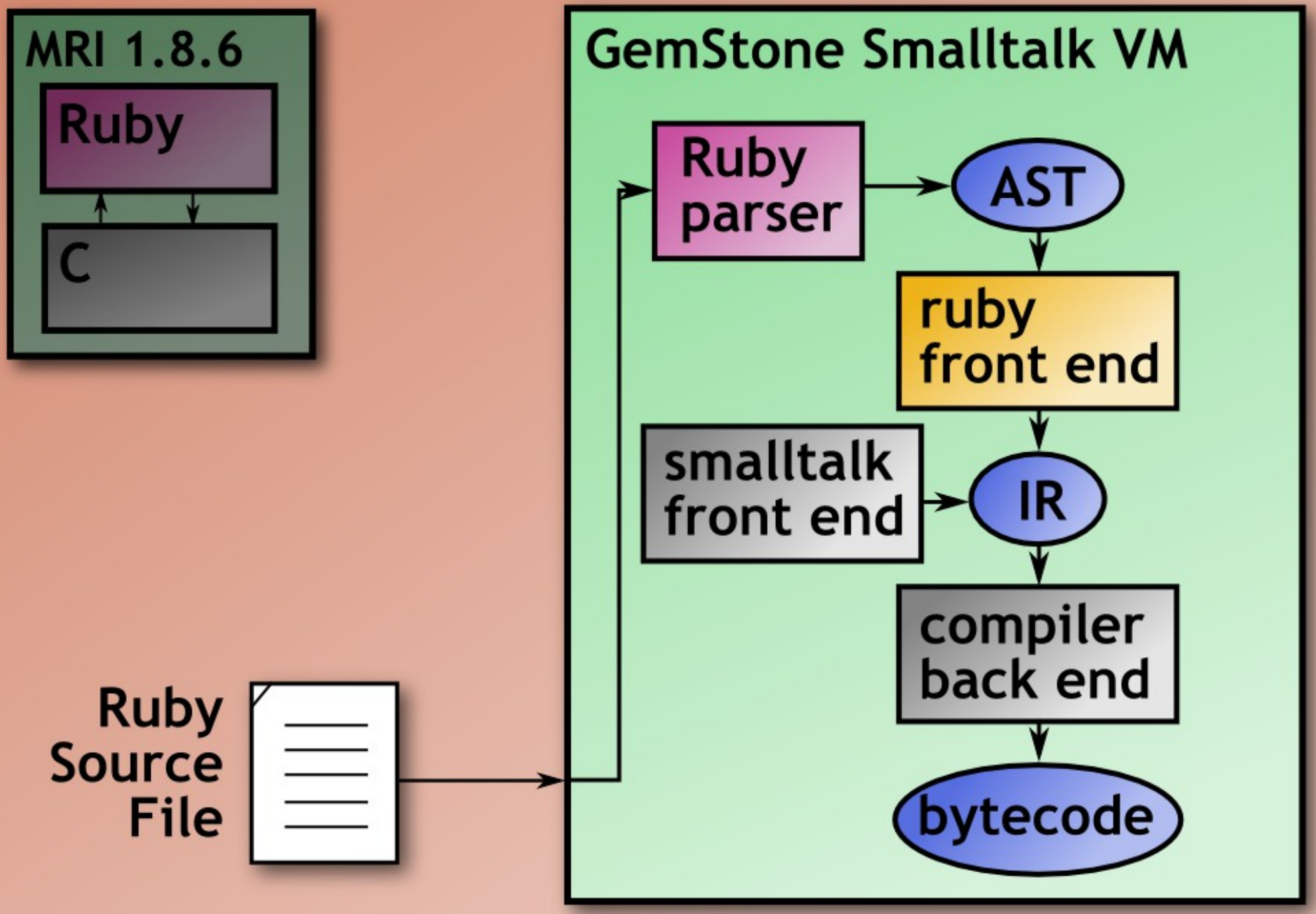
Ruby Grammar

Ruby 1.8.6 parse.y

Ruby 1.8.6 parse.y

Cheat


MRI 1.8.6
Ruby
C
socket
sexp
source
GemStone Smalltalk VM
simple parser
AST
ruby front end
smalltalk front end
IR
compiler back end
bytecode
Ruby Source File


MRI 1.8.6
Ruby
C
GemStone Smalltalk VM
Ruby parser
AST
ruby front end
smalltalk front end
IR
compiler back end
bytecode
Ruby Source File

Challenge 2: Arity

Arity Challenges

- Number of message send arguments and method definition parameters may be different.
- '*' Argument and parameter – Array expansion for variable arity
- '&' block passing
- Smalltalk VM expects sender and receiver arity to always be fixed and to match

Cheat

Synthetic 'Bridge' Methods

- def foo(a,*b)
  …
  end
- Compiles:
  foo, foo*, foo&, foo*&,
  foo:, foo:*, foo:&, foo:*&
  foo::, foo::*, foo::&, foo::*&
  foo:::, foo:::*, foo:::&, foo:::*&

Synthetic 'Bridge' Methods

- some_object.foo(a,b,*c)
  compiles a send to selector foo::*
- Bridge method foo::* adapts number of arguments and resends to actual foo variant

Challenge 3: Coexistence of Ruby and Smalltalk

Immediate Classes

Not Cheat

Environment-specific behavior

- Method dictionary per class per environment
- Each send site specifies an environment
- Complicates full method lookup
- Does not affect send-site caches

Challenge 4: Per-instance Behavior

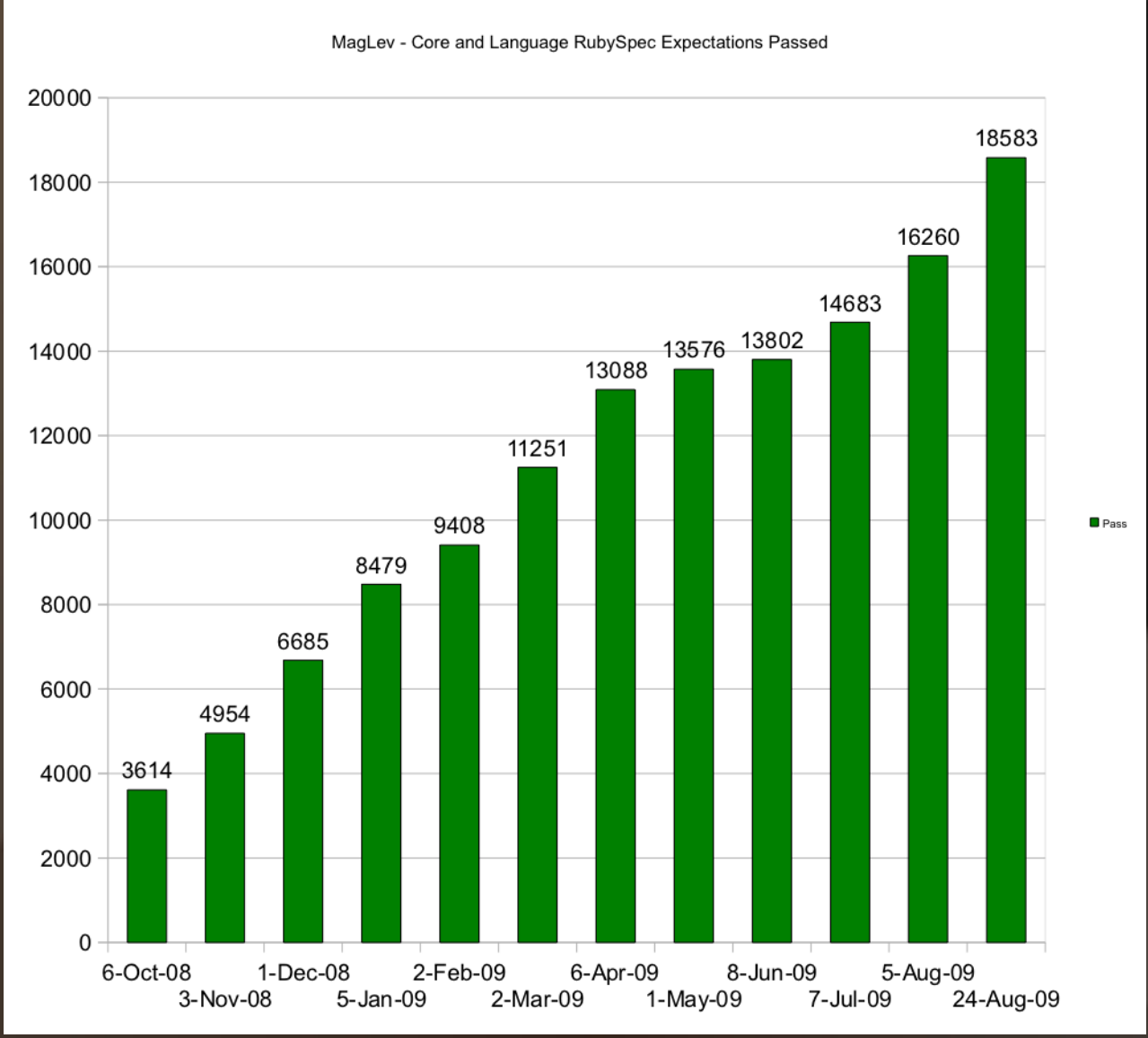
Challenge 5: Per-instance Variables

In Ruby, an instance variable is created by the act of assigning to it.

Possible instvars are hard to find

```
class MyExample
    def initialize(someValue)
        @my_instvar = someValue
    end
end
…
class MyExample
    def setAlpha(newAlpha)
        @alpha = newAlpha
    end
end
```

Core Tests Passed Over Time



Did we make the right choice?

Yes We Did

Cheat

Not cheating at all, really

Thank You

Questions?