

ruby2c

Automatic translation of ruby code to C.

by Seattle.rb's

Ryan Davis <ryand-ruby@zenspider.com>

&

Eric Hodel <drbrain@segment7.net>

Overview

- Background information and Goals
- Introduction to metaruby
- Ruby2c Design
- Current Status
- ...some magic

Goals & Background

- The Problem
- A Proposed Solution
- Related Projects & Information

The Problem

- Simply put, writing ruby internals in C requires a mental context switch every time you go from ruby to C and back.
- C sucks.
 - This makes the internals harder to understand.
 - Which makes it harder to recruit otherwise good coders to work on ruby internals.
 - Which slows down ruby's development.

A Proposal

- Implement the whole thing in ruby, and translate to C.
- No more context switching.
- Able to test changes live in the system.
- More understandable internals.
- More accessible to others.
- Must be in a subset of ruby that is easily translatable to C.

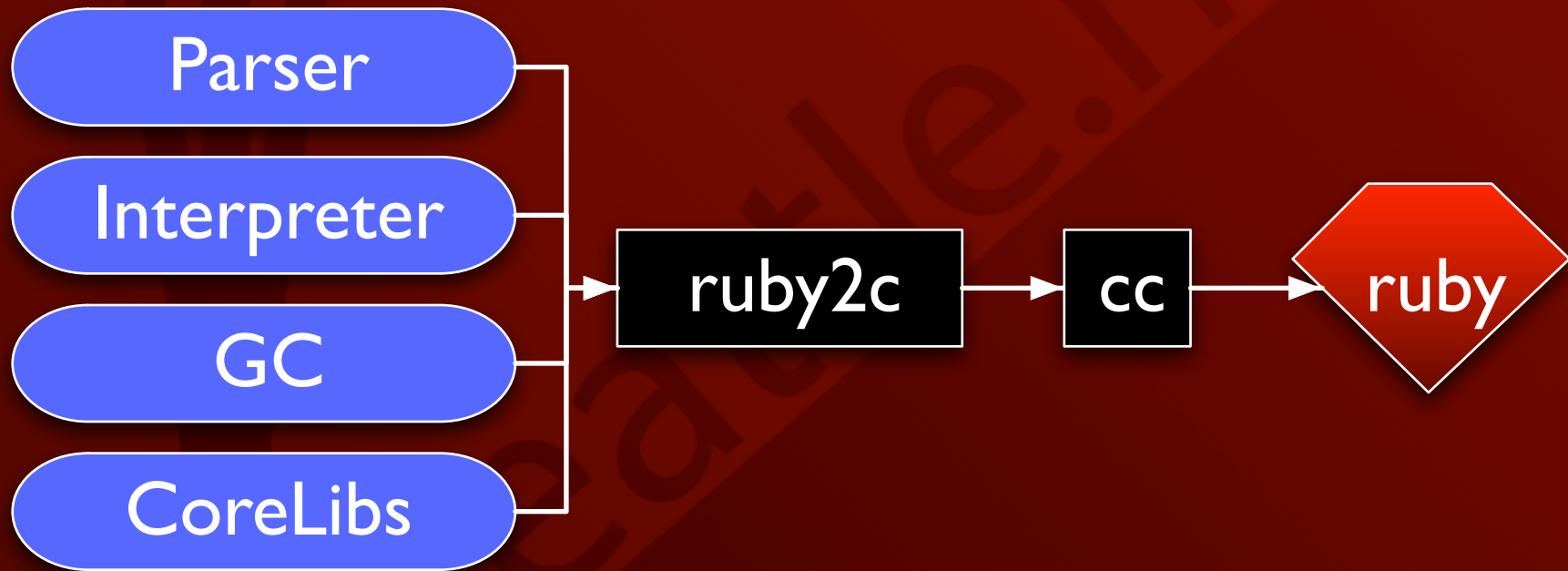
Related Projects & Info

- Projects outside of ruby-land:
 - Squeak Smalltalk is implemented in itself.
 - Newest version of Ungar's Self is as well.
 - Wirth's Pascal, Modula-2, and Oberon.
- Ruby-land projects:
 - YARV, jruby, lypanov's rubydium. *others?*
 - Matju's metaruby project is similar to our core library module, but otherwise unrelated.

Metaruby

- Ruby2c is a subset of the metaruby project.
- Metaruby intends to implement ruby's internals in ruby itself.
- The metaruby implementation will use ruby2c to convert itself to C and bootstrap a new ruby binary.
- Metaruby should be fully compatible w/ Matz's ruby.

Basic Architecture



Parser

- The parser needs to be rewritten in the ruby2c subset.
- LL vs LR, shouldn't matter
- (but I prefer LL so you might want to beat me to it)
- *We are recruiting for this module!*

Interpreter

- Needs to be rewritten in the ruby2c subset.
- Should be able to run any valid AST.
- Eric has an experimental interpreter written.

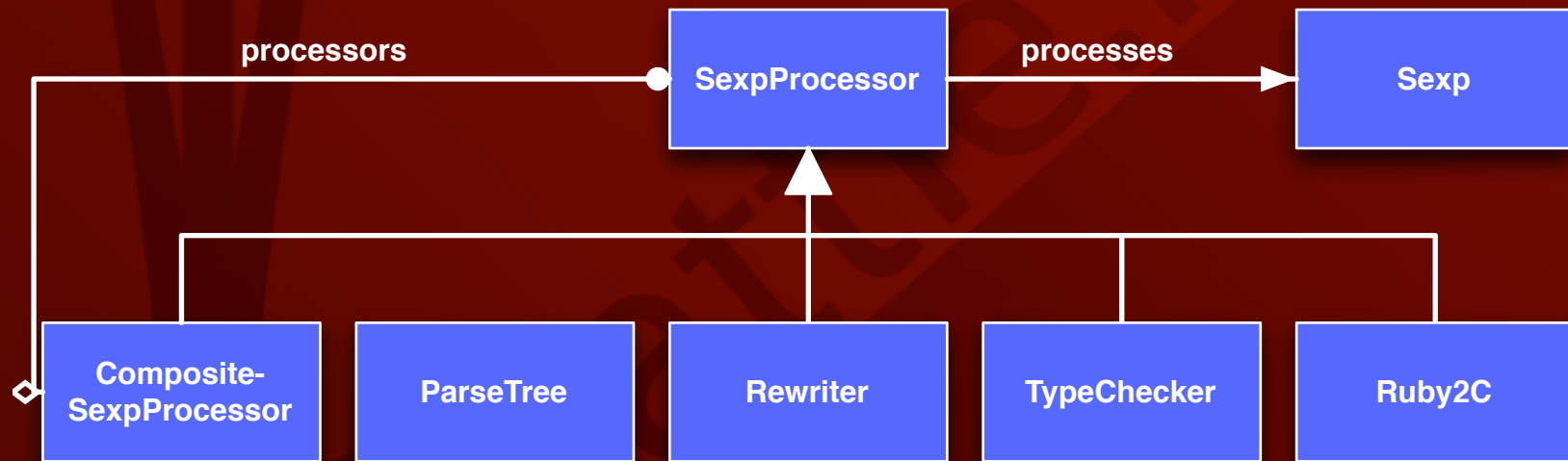
Garbage Collector

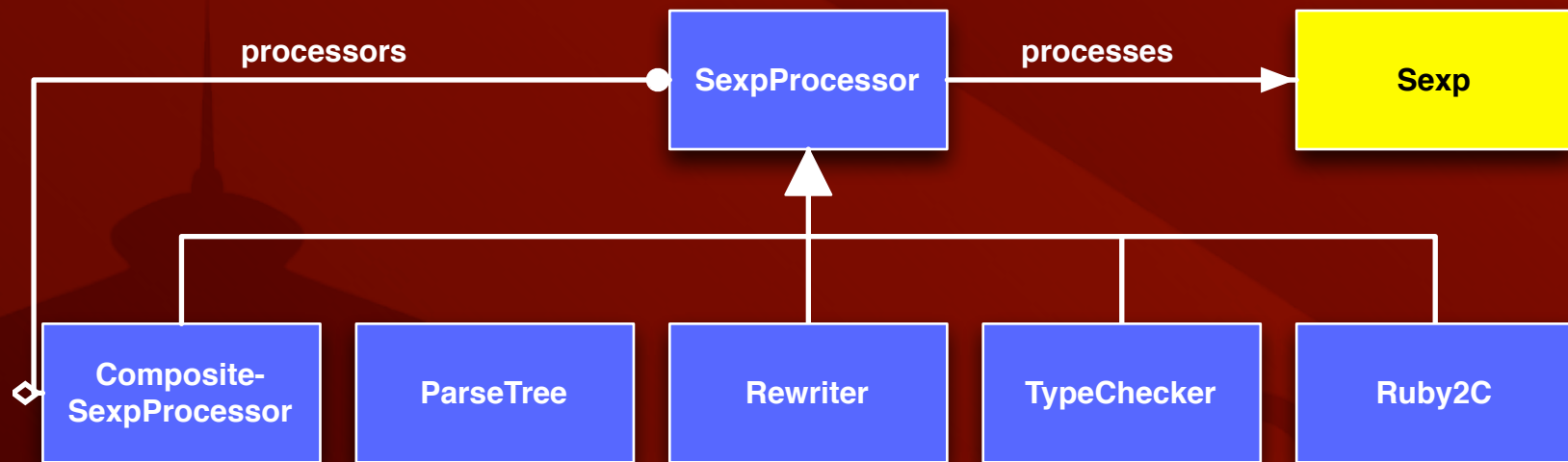
- Needs to be rewritten in the ruby2c subset.
- Probably the hardest part of our entire project.
- *We are recruiting for this module!*

Core Libraries

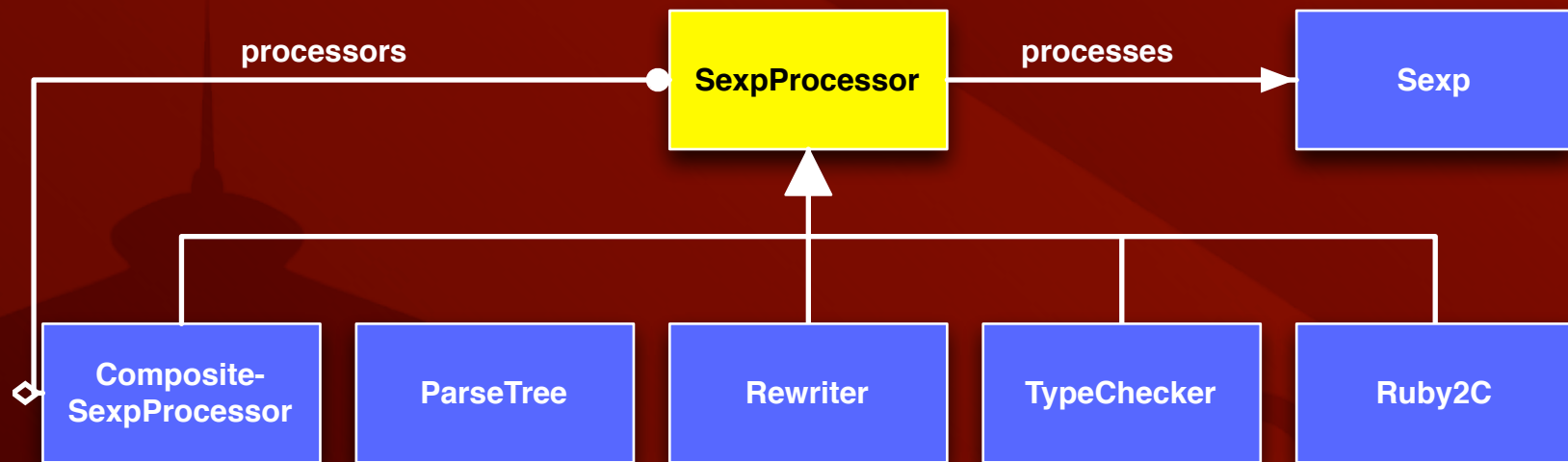
- Array, Hash, Time, etc... all need to be rewritten in the ruby2c subset.
- We've converted rubicon to help verify translation.
- Might be able to adopt other project's efforts on this one.
- *We are recruiting for this module!*

Basic ruby2c Design

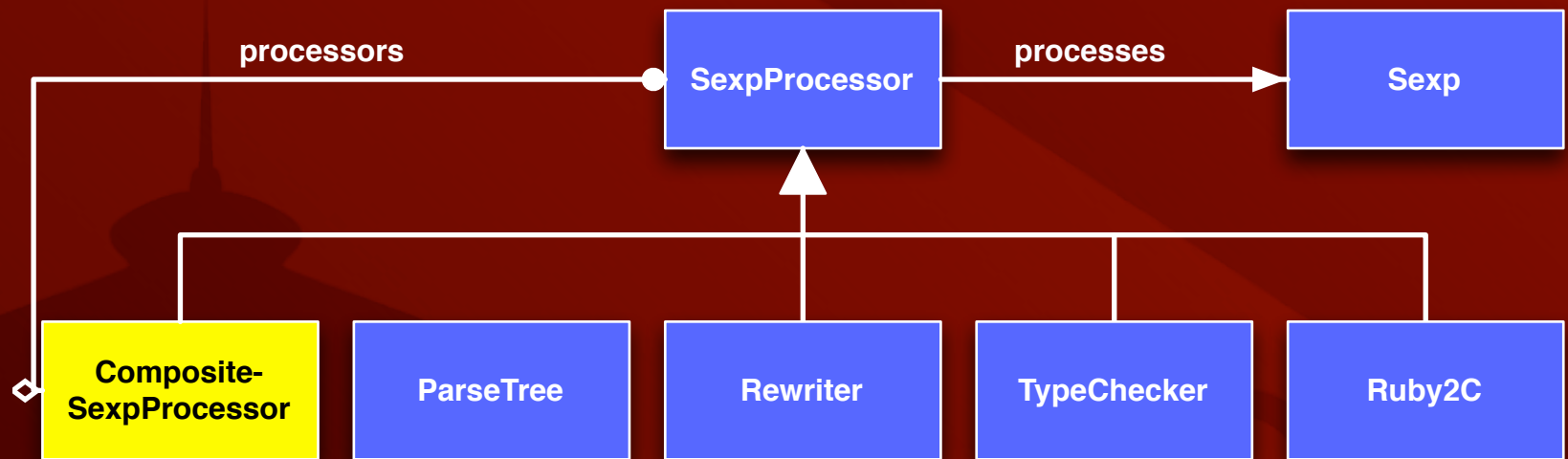




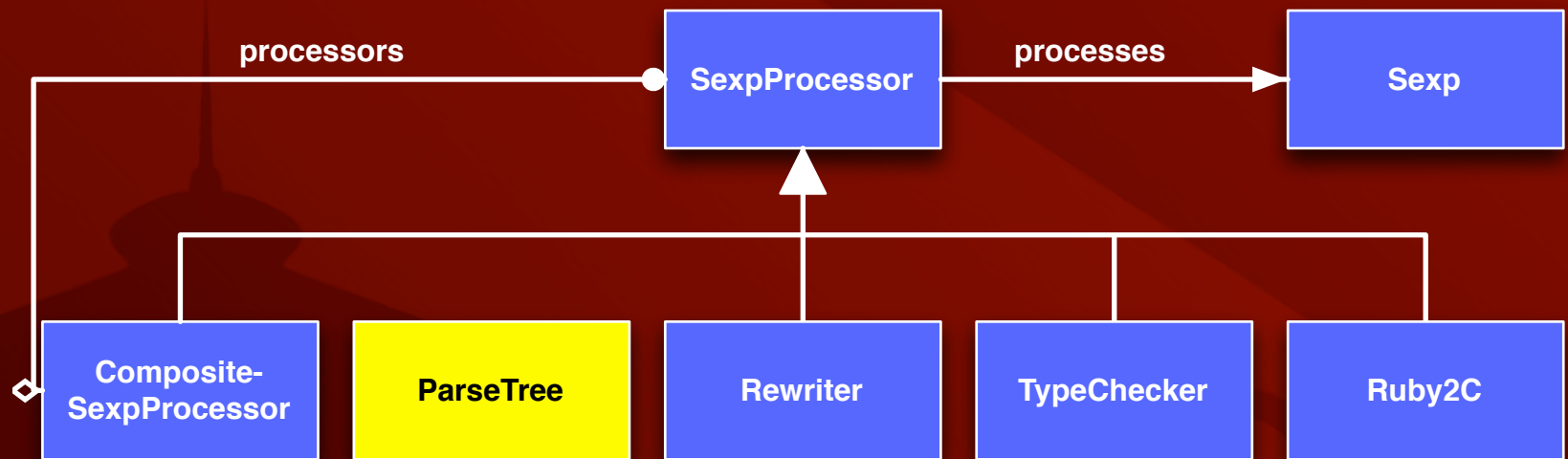
- Sexp subclasses Array.
- Contains an extra member: `sexp_type`.
- Has some extra (recursive) iterators like `each_of_type(type)`.
- Nothing too spectacular here.



- SexpProcessor provides a single method: `process (sexp)`
- Uses reflection to dynamically dispatch to `process_something (sexp)`
- *something* is determined by the type of the `sexp`.
- Enforces basic rules and also provides a generic processor.



- A simple composite pattern as applied to SexpProcessor
- Allows for chains of processors to be easily hooked together.



- ParseTree is a C extension via RubyInline.
- It returns a method's AST in sexp form.

```

def hello(n)
  1.upto(n) do
    puts "hello world"
  end
end

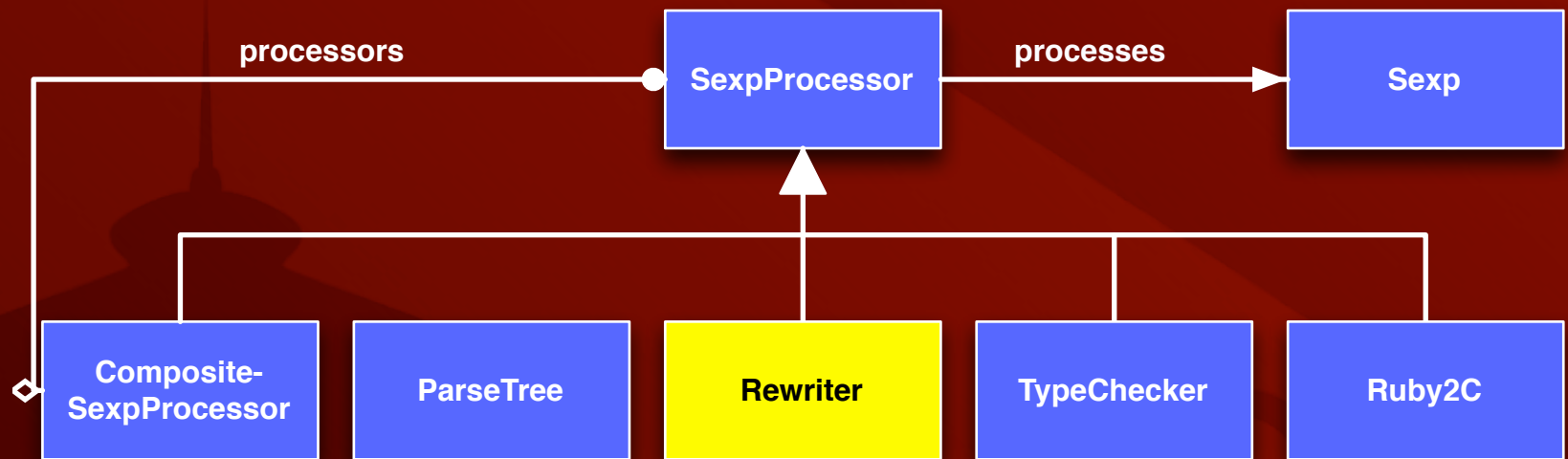
```

becomes:

```

[:defn,
 "hello",
 [:scope,
 [:block,
 [:args, "n"],
 [:iter,
 [:call,
 [:lit, 1],
 "upto",
 [:array, [:lvar, "n"]]]],
 nil,
 [:fcall,
 "puts",
 [:array,
 [:str, "hello world"]]]]]]]]

```



Rewriter cleans up sexps from ParseTree.
Makes it easier to deal with. More uniform.

Rewritten nodes colored green

focus for
next slide

```

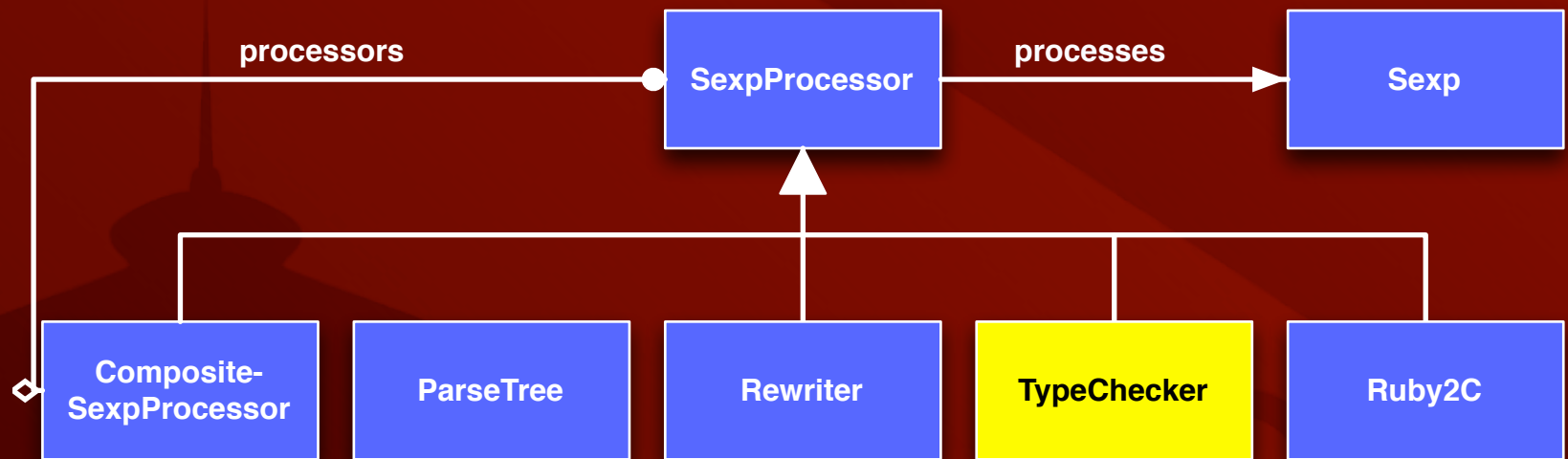
[:defn,
 "hello",
 [:scope,
 [:block,
 [:args, "n"],
 [:iter,
 [:call,
 [:lit, 1],
 "upto",
 [:array, [:lvar, "n"]]],
 nil,
 [:fcall,
 "puts",
 [:array,
 [:str, "hello world"]]]]]]]]]
  
```

becomes:

```

[:defn,
 "hello",
 [:args, "n"],
 [:scope,
 [:block,
 [:lasgn, "temp_var1", [:lit, 1]],
 [:while,
 [:call, [:lvar, "temp_var1"],
 "<=", [:array, [:lvar, "n"]]],
 [:block,
 [:call, nil, "puts",
 [:array, [:str, "hello world"]]],
 [:lasgn,
 "temp_var1",
 [:call, [:lvar, "temp_var1"], "+",
 [:array, [:lit, 1]]]]]]]]]]
  
```



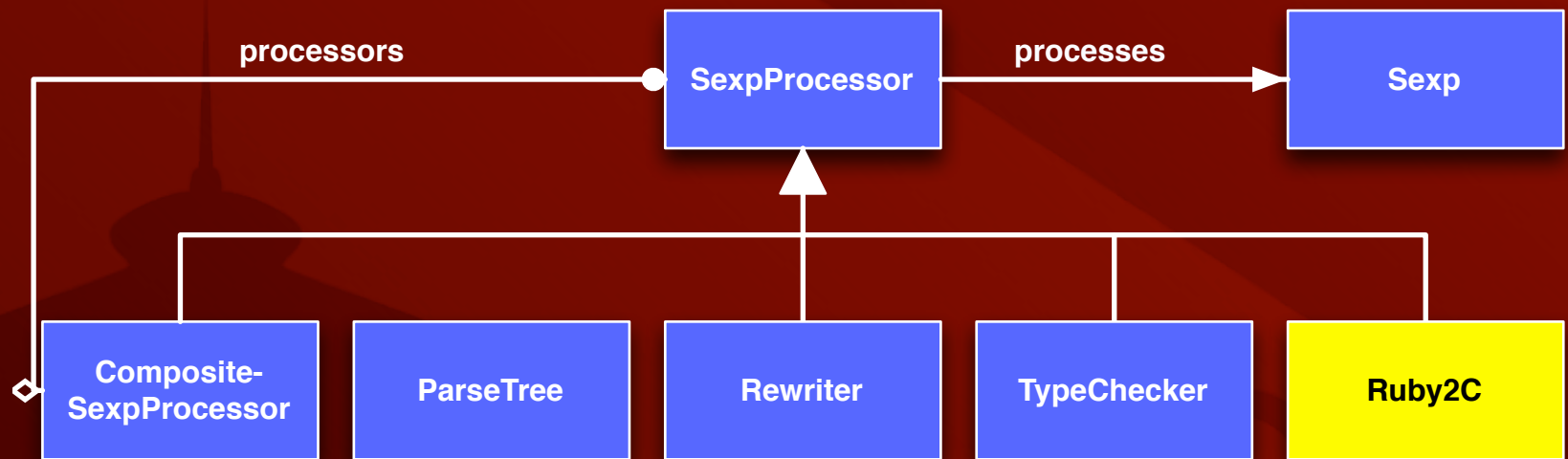


- TypeChecker infers and unifies types, adding them to the sexp.
- Starts to get very unreadable at this stage.
- Hence, the subset of last slide.

```
[:call,
  [:lvar, "temp_var1"],
  "<=",
  [:array, [:lvar, "n"]]],
```

becomes:

```
[:call,
  [:lvar, "temp_var1", Type.long],
  "<=",
  [:array, [:lvar, "n", Type.long]],
  Type.bool],
```



```
[:call,
  [:lvar, "temp_var1", Type.long],
  "<=",
  [:array,
    [:lvar, "n", Type.long]],
  Type.bool],
```

becomes:

```
temp_var1 <= n
```

finally:

```
def hello(n)
  1.upto(n) do
    puts "hello world"
  end
end
```

becomes:

```
void
hello1(long n) {
  long temp_var1;
  temp_var1 = 1;
  while (temp_var1 <= n) {
    puts("hello world");
    temp_var1 = temp_var1 + 1;
  }
}
```

A Smaller Example:

- Ruby parses code
- ParseTree extracts AST
- Rewriter doesn't touch it
(SexpProcessor.process converts it to a Sexp)
- TypeChecker unifies it
- Ruby2C translates to C

Ruby

```
false and true
```

ParseTree

```
case NODE_AND:  
  add_to_parse_tree(current, node->nd_1st);  
  add_to_parse_tree(current, node->nd_2nd);  
  break;
```

```
[:and, [:false], [:true]]
```

Rewriter

```
:and does not get rewritten
```

```
s(:and, s(:false), s(:true))
```

TypeChecker

```
def process_and(exp)  
  rhs = process exp.shift  
  lhs = process exp.shift  
  rhs.sexp_type.unify lhs.sexp_type  
  rhs.sexp_type.unify Type.bool  
  return t(:and, rhs, lhs, Type.bool)  
end
```

```
[:and, [:false, Type.bool], [:true,  
  Type.bool], Type.bool]
```

Ruby2C

```
def process_and(exp)  
  lhs = process exp.shift  
  rhs = process exp.shift  
  return "#{lhs} && #{rhs}"  
end
```

```
Qfalse && Qtrue;
```

Current Status

- *Everything* shown on these slides came from running real code.
- The design is *fully implemented*, we are expanding our supported subset of ruby.
- Simple ruby sexp interpreter for longs only was written in *one day*.
- We think this helps validate our design.

Extra Magic

- Integrated into RubyInline
- *13 lines of ruby!*
- Automatic optimization of ruby code!

13 Lines of Ruby

```
module Inline
  class Ruby < Inline::C
    def initialize(mod)
      super
    end

    def optimize(meth)
      src = RubyToC.translate(@mod, meth)
      @mod.class_eval "alias :#{meth}_slow :#{meth}"
      @mod.class_eval "remove_method :#{meth}"
      c src
    end
  end
end
```


Automatic Optimization:

```
class MyTest
```

```
  def factorial(n)
```

```
    f = 1
```

```
    n.downto(2) { |x| f *= x } becomes:
```

```
    return f
```

```
  end
```

```
  inline(:Ruby) do |builder|
```

```
    builder.optimize :factorial
```

```
  end
```

```
end
```

```
static VALUE factorial(VALUE self,  
                        VALUE _n) {
```

```
  long n = NUM2INT(_n);
```

```
  long f;
```

```
  long x;
```

```
  f = 1;
```

```
  x = n;
```

```
  while (x >= 2) {
```

```
    f = f * x;
```

```
    x = x - 1;
```

```
  };
```

```
  return INT2NUM(f);
```

```
}
```

and **dynamically** replaces the ruby version!
in this case, an 8.8x speed-up!

Want to Help?

- Contact either person on the title page.
- A ruby2c subset spec is coming soon.
- Lots to write, and much of it should be fun!