Types for Ruby

Jeff Foster
University of Maryland, College Park

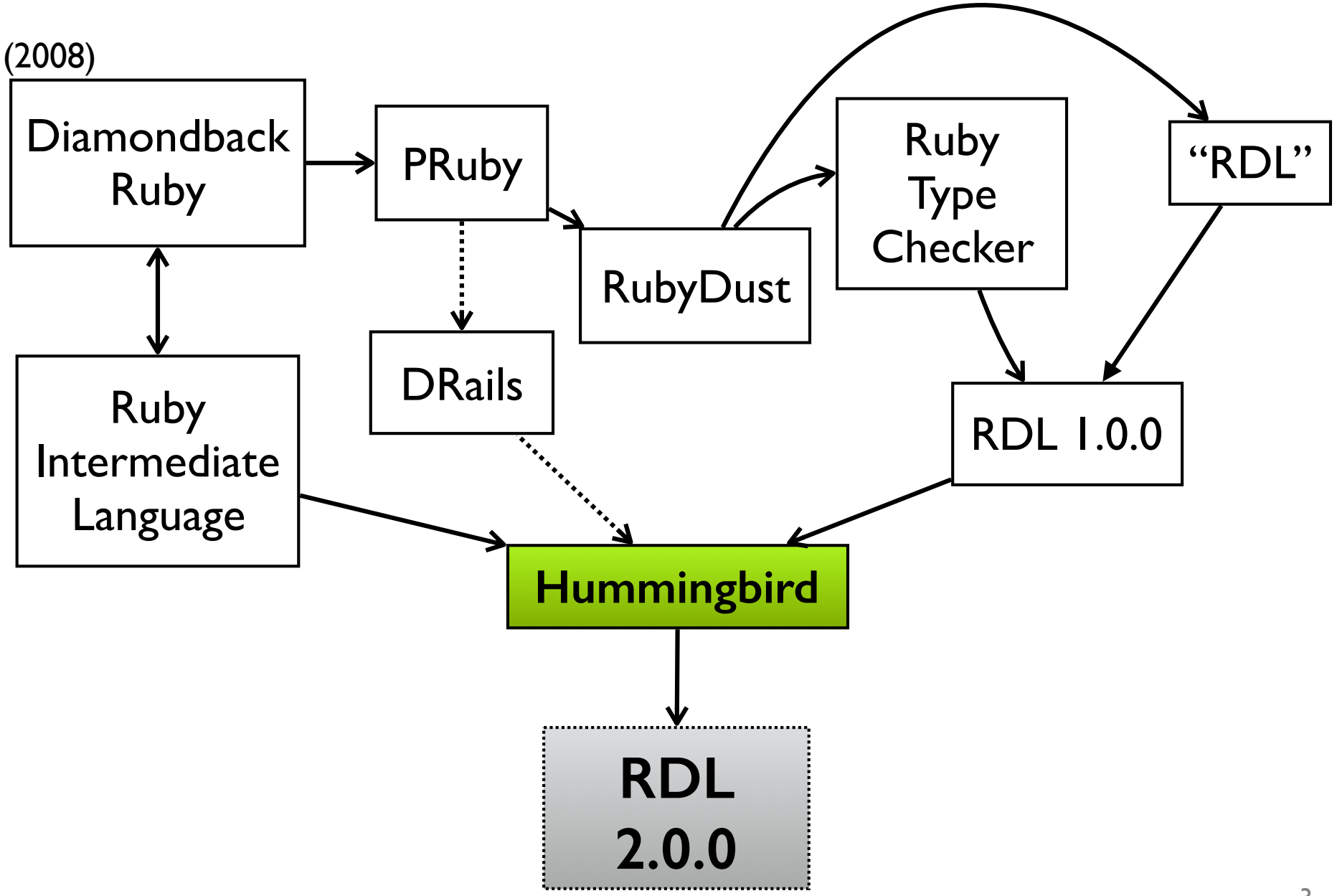
Joint work with Mike Furr, David An, Avik Chaudhuri, Stevie Strickland, Brianna Ren, John Toman, Mike Hicks

Types for Ruby

- Over last several years, have been working on bringing some benefits of static typing to Ruby
  - Ruby = Smalltalk + Perl
  - Popular for building web apps ("Ruby on Rails")
- Goal: Make types optional but useful
  - Develop a program without types (rapidly)
  - Include them (later) to provide static checking where desired
  - Find problems as *early* as possible (but not too early!)

We think we finally have it right!

Timeline


(2008)
Diamondback Ruby
PRuby
RubyDust
Ruby Type Checker
"RDL"
Ruby Intermediate Language
DRails
RDL 1.0.0
Hummingbird
RDL 2.0.0

Diamondback Ruby

- How do we build a static type system that accepts "reasonable" Ruby programs?
  - What idioms do Ruby programmers use?
  - What typing features are needed?
  - Are Ruby programs even close to statically type safe?
- Uses type inference to reduce need for annotations

Basic Types

```
type String, :+, "(String) → String"

type String, :insert, "(Fixnum, String) → String"

type String, :upto,
      "(String) { (String) → Object } → String"
```

- Type annotations specify class, method, type
  - `:string` is a Ruby *symbol*
- Ruby methods can take optional block args

```
"a".upto("z") { |c| puts c }
```

Intersection and Union Types

```
type String, :[], "(Fixnum) → String or nil"
type String, :[], "(Fixnum, Fixnum) → String or nil"
type String, :[], "(Range or Regexp) → String or nil"
type String, :[], "(Regexp, Fixnum) → String or nil"
type String, :[], "(Regexp, String) → String or nil"
type String, :[], "(String) → String or nil"
```

- Intersections for overloading
  - Implemented via dynamic type tests in method body
- Unions in place of interfaces

Other Type Language Features

- Optional and vararg types

```
type String, :chomp, "(?String) → String"
type String, :delete, "(String, *String) → String"
```

- Structural types

```
type IO, :puts, "(*[to_s: () → String]) → nil"
```

- Generics

```
type Array, :[], "(Range) → Array<t>"
type Array, :[], "(Fixnum or Float) → t"
type Array, :[], "(Fixnum, Fixnum) → Array<t>"
```

- Self

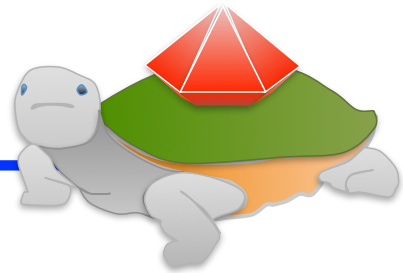
```
type Object, :clone, '() → self'
```

Tuples, Finite Hashes, and Singletons

```
def m() [ 1, 'two' ]; end
a, b = m
# a = 1, b = ‘two’
```

- `Array<Fixnum or String>` imprecise
- Instead, use *tuple type* `[Fixnum, String]`
- Even better, use *singleton type* `[1, String]`
  - (Note strings are mutable, can’t be singletons)
- Coerce tuples to arrays as needed
  - Note need to apply coercion retroactively!
- Similar approach for finite hashes like `{a: 1, b: 2}`

RIL and Diamondback Ruby



- Ruby Intermediate Language (RIL)
  - GLR parser for Ruby source code
  - Compact, simplified intermediate representation
  - Pretty-printer that outputs valid, executable Ruby code
- Diamondback Ruby
  - Static type inference for Ruby built on RIL
  - Ahead-of-time constraint generation and resolution
  - PRuby added profiling to handle some dyn features

Example Errors Found

- Typos in names
  - `Archive::Tar::ClosedStream` instead of `Archive::Tar::MiniTar::ClosedStream`
  - `Policy` instead of `Policies`
- Undefined variables

```
return rule_not_found if !@values.include?(value)
```

  - `rule_not_found` not in scope
  - Program did include a test suite, but this path not taken

Example Errors Found (cont'd)

- Bad method arguments

```
class Integer
  def to_bn
    OpenSSL::BN.new(self)
  end
end
```

  - `BN.new` expects `String`, not `Integer`
  - `3.to_bn` would cause a type error

Syntactic Confusion

```
assert_nothing_raised { @hash['a', 'b'] = 3, 4 }
...
assert_kind_of(Fixnum, @hash['a', 'b'] = 3, 4)
```

- First passes `[3,4]` to the `[]=` method of `@hash`
- Second passes `3` to the `[]=` method, passes `4` as last argument of `assert_kind_of`
  - Even worse, this error is suppressed at run time due to an undocumented coercion in `assert_kind_of`

Syntactic Confusion (cont'd)

```
flash[:notice] = "You do not have ..."
+ "..."
```

- Programmer intended to concatenate two strings
- But here the `+` is parsed as a unary operator whose result is discarded

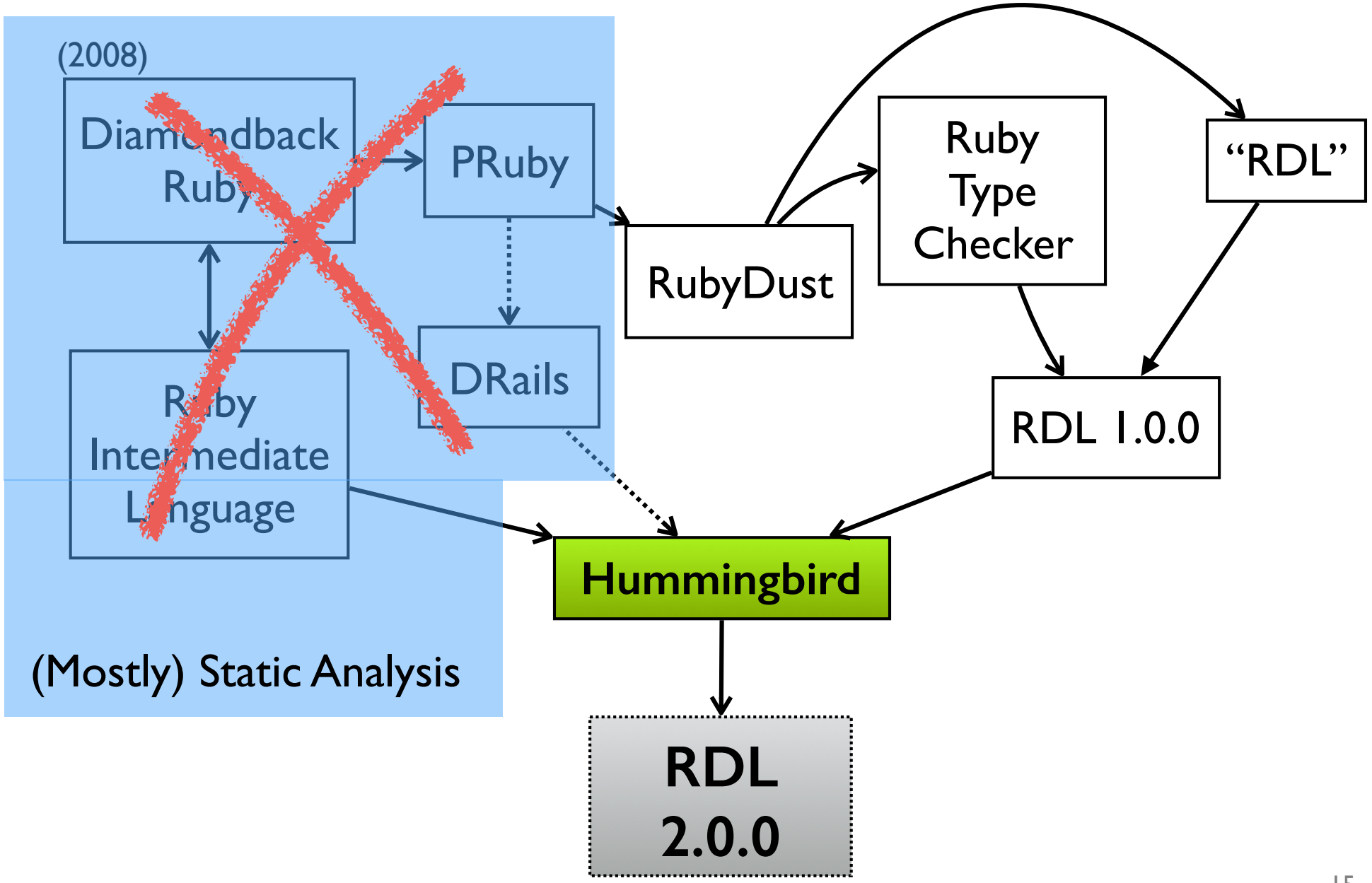
```
@count, @next, @last = 1
```

- Intention was to assign `1` to all three fields
- But this actually assigns `1` to `@count`, and `nil` to `@next` and `@last`

DRuby Was Promising, But...

- RIL somewhat fragile
  - Tied to Ruby 1.8.7; required fixing for Ruby 1.9, and we're now at Ruby 2.3!
- Profiling doesn't match well with Ruby
  - Ruby has no "compile" phase, so somewhat unnatural
- Overcoming limitations would require very complex static analysis
  - Complex type system = hard to predict = bad

Becoming Dynamic

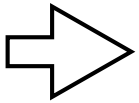


RubyDust and RTC

- **Ruby Dynamic Unraveling of Static Types**
  - Type inference
- The **Ruby Type Checker**
  - Type checking
- RDL
  - Contract system for Ruby
- Dynamic analysis (does not examine source code)
  - Does not require Ruby front-end
  - Language features that are tricky for static analysis become easy
    - Flow-sensitivity, highly dynamic features, etc.

Method Interception

```ruby
def foo(x)
  # foo body
end
```



```ruby
alias __rdl_foo_old, foo

def foo(x)
  # special stuff for x
  r = __rdl_foo_old(x)
  # special stuff for r
  return r
end
```

	x (of type t)	r (of type t)
RDL	check precond	check postcond
RTC/ RubyDust	`err unless x.type ≤ t` `x = x.wrap(t)`	`err unless r.type ≤ t` `r = r.wrap(t)`

RTC and RubyDust Wrappers

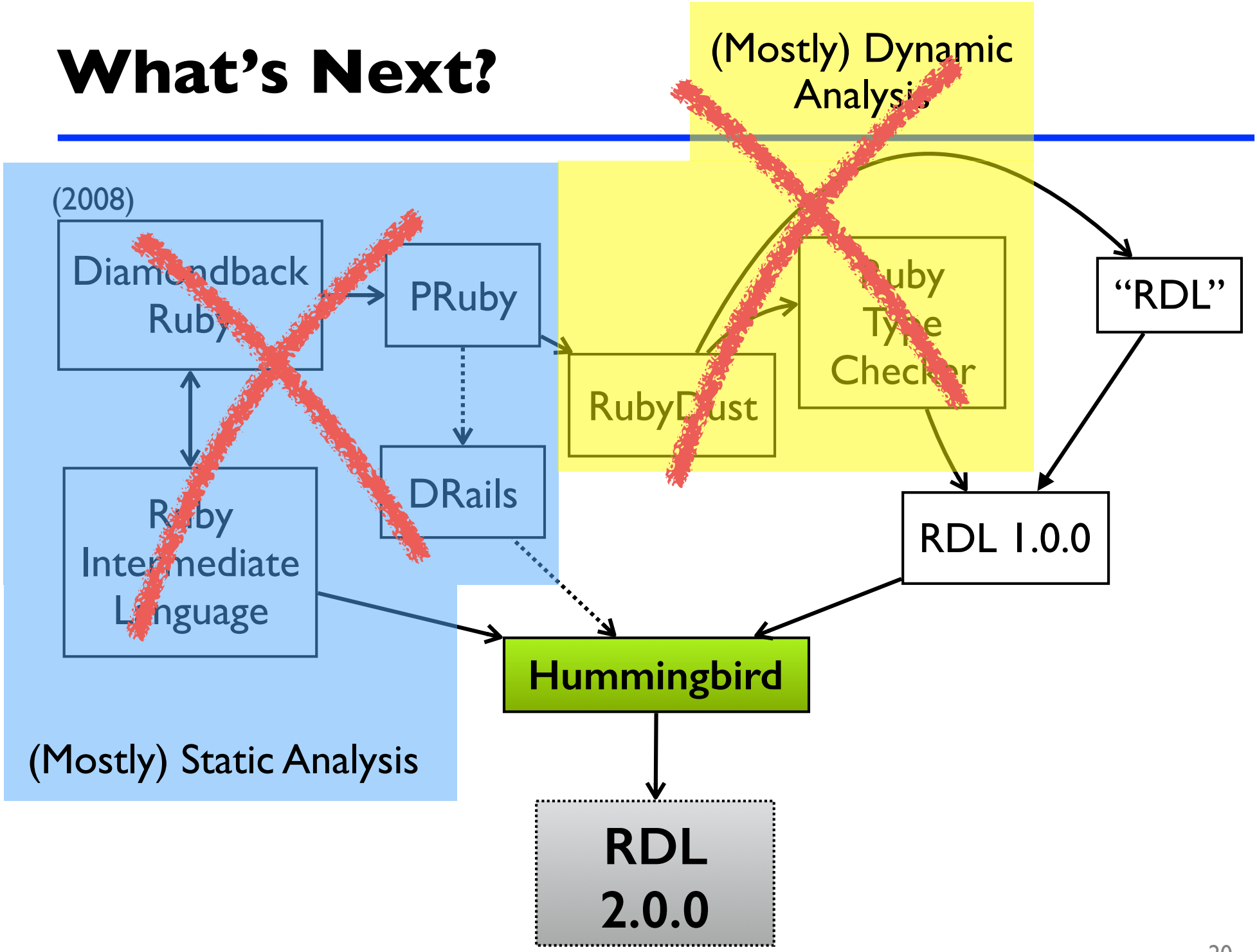
- `x = o.wrap(t)` returns new obj that
  - Delegates (almost) all methods to `o`
  - Associates type `t` with object `o`
- When wrapped object `x` used, check/constrain `x`'s type
  - `y = "hello" + x` requires `x.type ≤ String`
  - `y = x + "hello"` requires `x.type ≤ [+:(String)→t']`

Results

- Soundness theorem for RubyDust
  - If every path through each method explored, then inferred types are sound
- Some performance improvements in RTC

- But still too heavyweight!
  - `wrap` really slows things down
  - Sometimes two orders of magnitude!

What's Next?


(Mostly) Dynamic Analysis
(2008)
Diamondback Ruby
PRuby
RubyDust
Ruby Type Checker
"RDL"
DRails
Ruby Intermediate Language
RDL 1.0.0
Hummingbird
(Mostly) Static Analysis
RDL 2.0.0

Ruby on Rails

- Very popular web app framework
  - Model - class whose instances are DB rows
  - Controller - receives HTTP requests
  - View - HTTP responses; HTML with embedded Ruby
- Large community, lots of users
  - Basecamp, GitHub, Shopify, Airbnb, Twitch, SoundCloud, Hulu, Zendesk, Square, Highrise, …
- Favors "convention over configuration"
  - Watch out for metaprogramming!

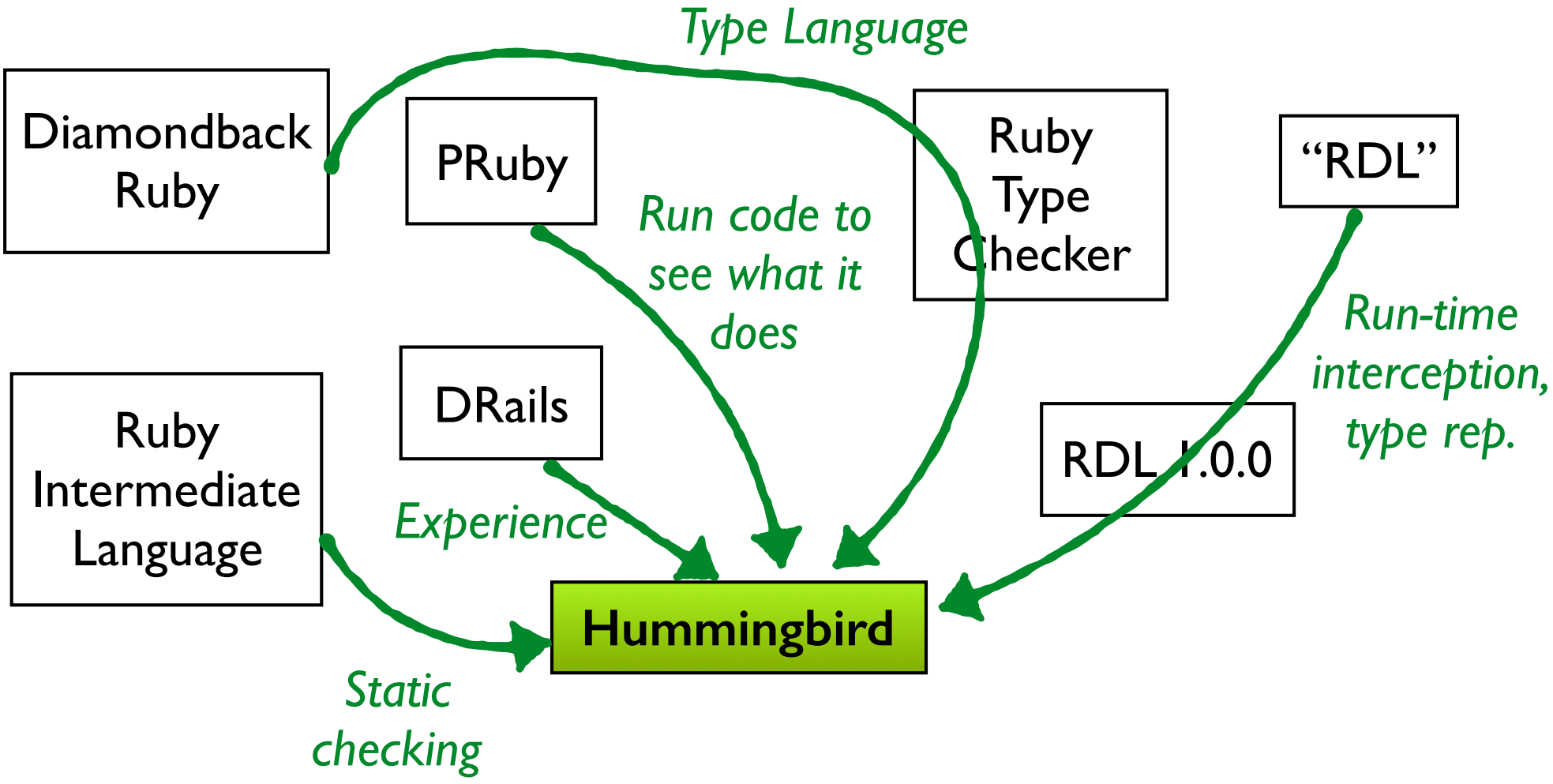
Metaprogramming in Rails

```
class Talk < ActiveRecord::Base
  belongs_to :owner, :class_name ⇒ "User"

  def owner?(user)
    return owner == user      # uh oh…
  end
```

- No explicit def of `owner`
  - Defined via metaprogramming in call to `belongs_to`
- Can't type check in {DRuby, PRuby, RubyDust, RTC}!
  - DRails was an attempt to handle Rails, but not robust

Putting the Right Ideas Together



Just-in-Time Type Checking

- Type annotations execute at run-time
  - Calling `type` stores type [DRuby] info in global table [RTC/RDL]
  - Metaprogramming [DRails] can generate types as meths created!
    - *(Also inspired by PRuby)*
- Statically check method body [RIL] at call
  - Uses current global type table
  - Memoize type checking for better performance

Type "Annotations" = Method Calls

```
type String, :[], "(Fixnum) → String or nil"
type String, :[], "(Fixnum, Fixnum) → String or nil"
…
```

- Map from `[String, :[]]` to types in `$__rdl_info`
  - Side effect: can query type info!

```
$ rdl_query String#[]
String#[]: (Fixnum, ?Fixnum) -> String
String#[]: (Regexp or Range<Fixnum>) -> String
String#[]: (Regexp, Fixnum) -> String
String#[]: (Regexp, String) -> String
String#[]: (String) -> String
```

Types and Metaprogramming

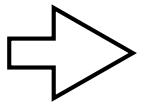
Add precondition

…which is a lambda with two args

Zany string manipulation!

```
pre(:belongs_to) do |name,options|
  # slightly simplified
  if options
    cls_str = options[:class_name]
  else
    cls_str = name.singularize.camelize
  end
  type name.singularize, "() → #{cls_str}"
  type "#{name.singularize}=", "(#{cls_str}) → #{cls_str}"
end
```

```
belongs_to(:owner,
  :class_name ⇒ "User")
```



```
type :owner, "() → User"
type :owner=, "(User) → User"
```

Another Example

Transaction has meths type, type=, …

```
Transaction = Struct.new(:type, :account_name, :amount)
Transaction.add_types(String, String, String)
t = # some Transaction
name = t.account_name
```

Now can be type checked!

For each (member, type) pair…

```
class Struct
  def self.add_types(*types)
    members.zip(types).each do |name, t|
      self.class_eval do
        type name, "() → #{t}"
        type "#{name}=", "(#{t}) → #{t}"
end end end end
```

Create type sigs

Implementation

- Method source code from RIL (→ JSON)
- Types from RDL
- Sometimes need type casts

```
type Marshal, "self.load", "(…) → Object"
var_type :@@cache, "Hash<A, B>"

r = Marshal.load(…)
@@cache = r.rdl_cast("Hash<A, B>")
```

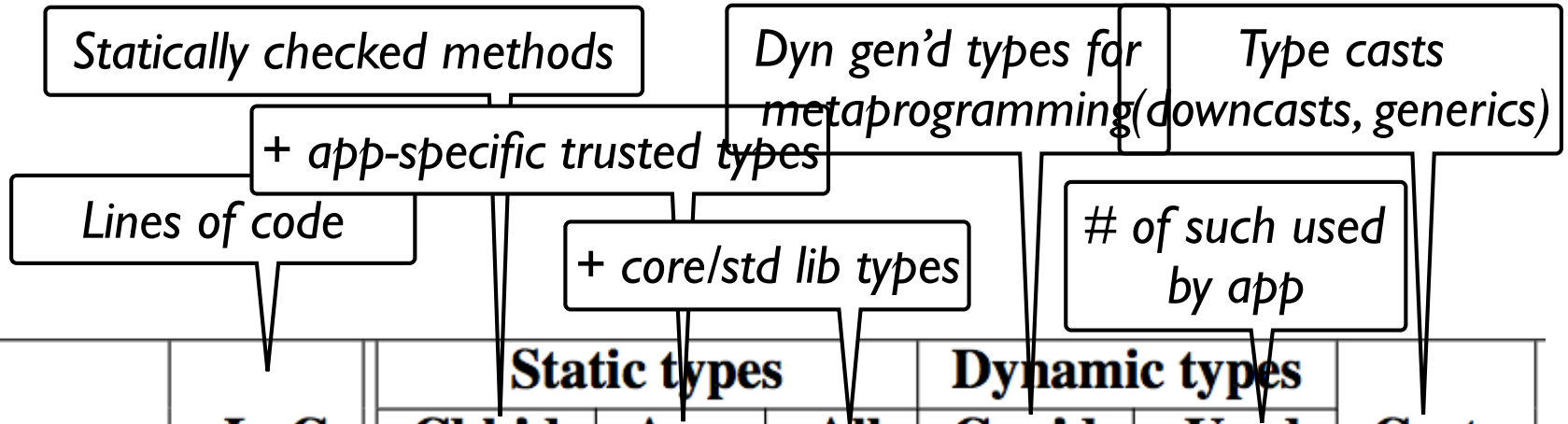
- Care needed for mixins

```
module M def foo(x) bar(x) end end
class C; include M; def bar(x) x + 1 end end
class D; include M; def bar(x) x.to_s end end
```

Experiments

- Ran on 2.3 GHz Intel Core i7 with 8GB memory
- Tests came with app or we wrote them with the goal of covering all app methods
- 6 Applications
  - 3 Rails: Talks, Pubs, Boxroom
  - 2 other metaprogramming: Rolify, Credit Card Transactions (CCT)
  - 1 plain: Countries

Type Checking Results: Annotations



App	LoC	Static types			Dynamic types		Casts
		Chk'd	App	All	Gen'd	Used	
Talks-1/4/2013	1,055	111	201	363	990	45	31
Boxroom-1.7.1	854	127	221	306	534	93	17
Pubs-1/12/2015	620	47	86	171	445	33	13
Rolify-4.0.0	84	14	24	71	26	2	15
CCT-3/23/2014	172	23	27	75	6	3	6
Countries-1.1.0	227	33	40	111	0	0	22

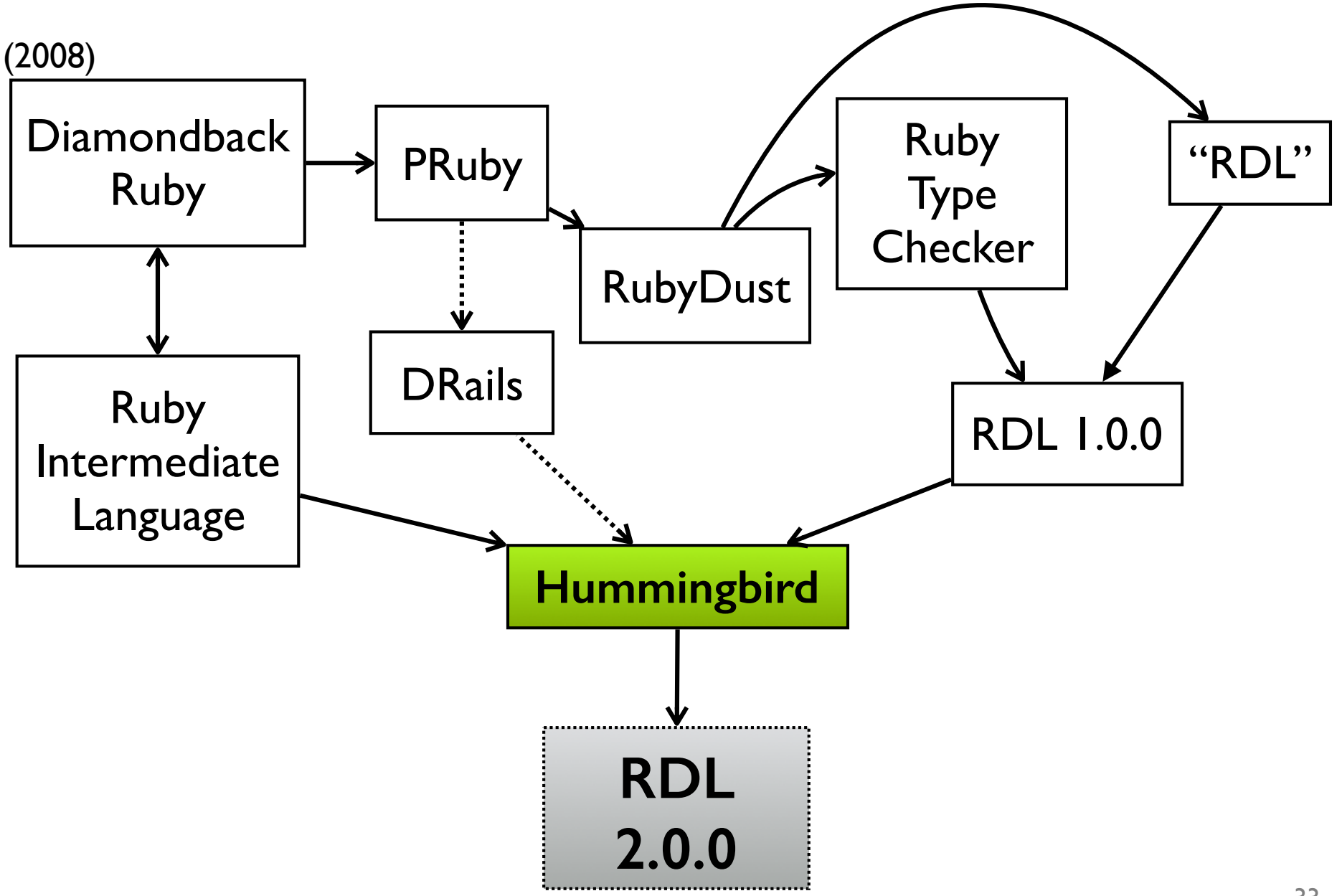
Type Checking Results: Performance

App	Running time (s)			
	Orig	No$	Hum	Or. Ratio
Talks-1/4/2013	162	1,590	256	1.6×
Boxroom-1.7.1	263	705	327	1.2×
Pubs-1/12/2015	72.0	4,470	217	3.0×
Rolify-4.0.0	5.63	7.79	6.71	1.2×
CCT-3/23/2014	3.06	78.2	17.4	5.7×
Countries-1.1.0	1.02	18.1	4.62	4.5×

Type Errors in Talks

version	code	bug
1/8/12-4	`copute_edit_fields`	misspelled method
1/7/12-5	`@list.talks.upcoming {|a, b| ...}`	extra block
1/26/12-3	`subscribed_talks(true)`	wrong arg type
1/28/12	`handler.object`	undefined method

RDL 2.0.0


(2008)
Diamondback Ruby
PRuby
RubyDust
Ruby Type Checker
"RDL"
Ruby Intermediate Language
DRails
RDL 1.0.0
Hummingbird
RDL 2.0.0

Switching Front-Ends

- RIL is written in OCaml
  - Might reduce user base if require ocaml installation!
- RDL Hummingbird impl. uses different parser
  - Someone *else* keeps it up to date with latest Ruby!
  - Side effect: great error reporting!

```
typecheck.rb:149:in `error':  (RDL::Typecheck::StaticTypeError)
a.rb:7:3: error: got type `String' where return type `Fixnum' expected
a.rb:7:   return 'forty-two'
a.rb:7:   ^~~~~~~~~~~~~~~~~
```

Occurrence Typing

- Needed to type check methods with ∩ types

```
type "(Fixnum) → Fixnum"
type "(String) → String"
def m(x)
  if x.is_a? Fixnum
    ...
  elsif x.is_a? String
    ...
  else
    raise "Error!"
  end
end
```

  - Many ways to introspect on types
  - Singleton types very useful here

Performance

- Still too many dynamic checks

```
type "(Fixnum) -> Fixnum"
def m(x)
  return p(x) # no need to dyn check p's arg type!
end

type "(Fixnum) -> Fixnum"
def p(y)
  return y
end
```

  - Will require code rewriting

Usability

- Will anyone want to use this?
  - Some interest from Ruby community
  - But still remains to be seen

- Advertising plans
  - ruby-talk mailing list
  - Some existing contacts in industry, core Ruby devs
  - Rubyconf 2016
  - …?

Conclusions

- The answer: Just-in-time static type checking
  - A long time to figure out, but now seems obvious!
- RDL 2.0.1 is available!

  https://github.com/plum-umd/rdl
  https://rubygems.org/gems/rdl
  - v1.0.0 released Dec 18, 2015
  - 4,957 downloads as of Dec 5, 2016
- Lots of future research work
  - Back to type inference
  - Dependent/refinement types
  - …?