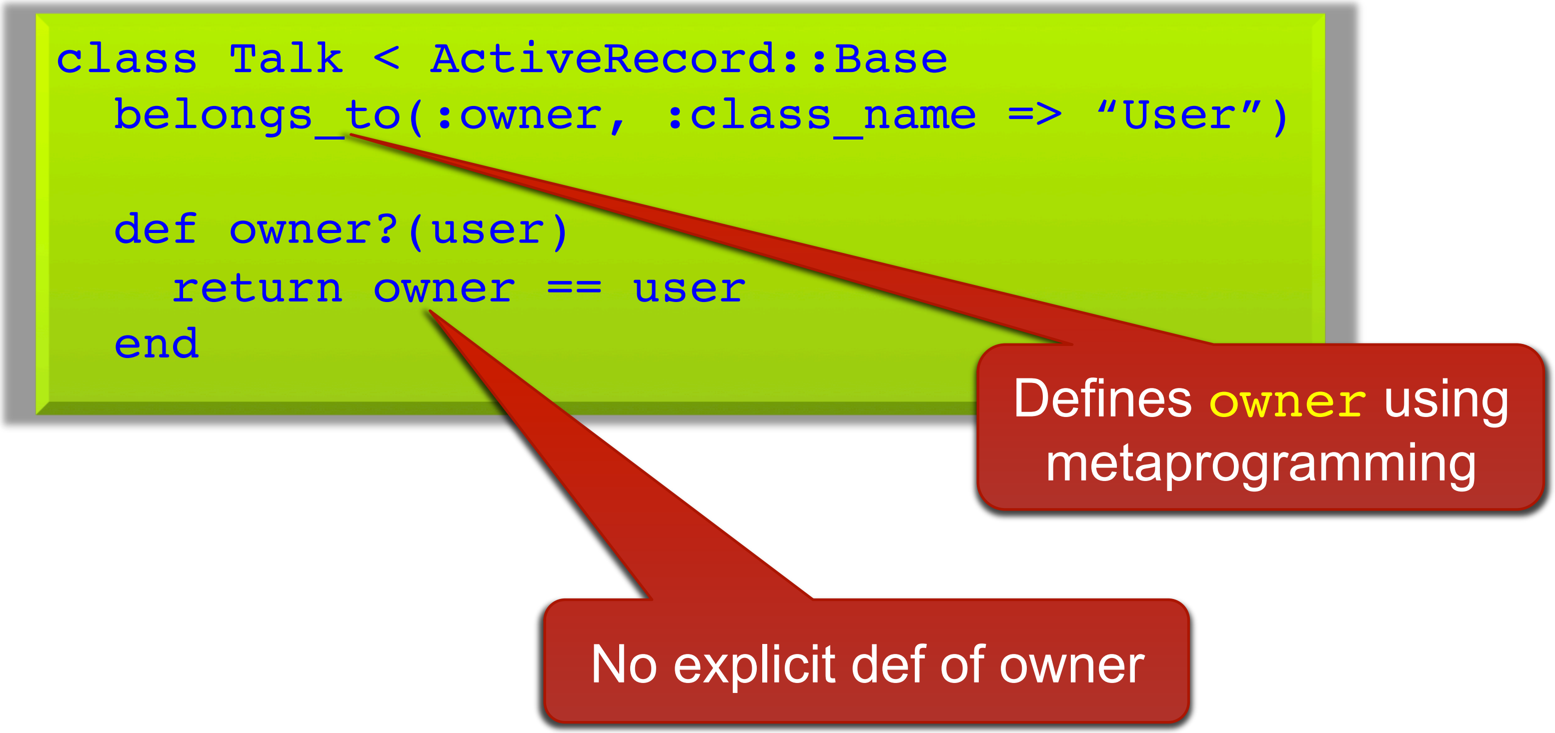
Type Checking and Metaprogramming

- Standard type systems cannot type check this simple Ruby on Rails example



```
class Talk < ActiveRecord::Base
  belongs_to(:owner, :class_name => “User”)

  def owner?(user)
    return owner == user
  end
```

Defines `owner` using metaprogramming

No explicit def of owner

Solution: Just-in-Time Type Checking

- Type annotations execute at run-time
  - Calling `type …` stores type info in global table
  - Extend **metaprogramming** to also generate types as meths created
    - Easy to do in practice
- Statically type check method body when called at run-time
  - Using current global type table
  - Memoize type checking for better performance

Hummingbird Results

- Hummingbird: Implementation for Ruby
- Applied to range of Ruby apps
  - 3 Rails apps, 2 other metaprogramming apps, 1 plain app
- Hummingbird successfully type checks all 6 apps
  - 1.2x – 5.7x performance overhead
- Found type errors in earlier versions of 1 Rails app
- Also applied to 1 Rails app as it was updated in dev mode to test cache invalidation

Hummingbird Example

Pre-contract for `belongs_to`

args passed to `belongs_to`

Gets return type of getter

```
pre(:belongs_to) do |name, options|
  name_str = name.to_s
  cn = options[:class_name] if options
  cls_str = cn ? cn : name_str.singularize.camelize
  type name.singularize, “() -> #{cls_str}”
  type “#{name.singularize}=“, “(#{cls_str}) -> #{cls_str}”
end
```

pre contract adds types

```
:owner, “() -> User”
:owner=, “(User) -> User”
```

```
belongs_to(:owner, :class_name => “User”)
```

Hummingbird and Struc

Creates methods
`:type, :type=,`
`:account_name, :account_name=,`
`:amount, :amount=`
for `Transaction` class

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

Annotate the attribs to get getter/setter types

Can now type check ☺

```
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
```

zip names with annotated types

creates type annotations for getters/setters. i.e.,
`account_name, “() -> String”`

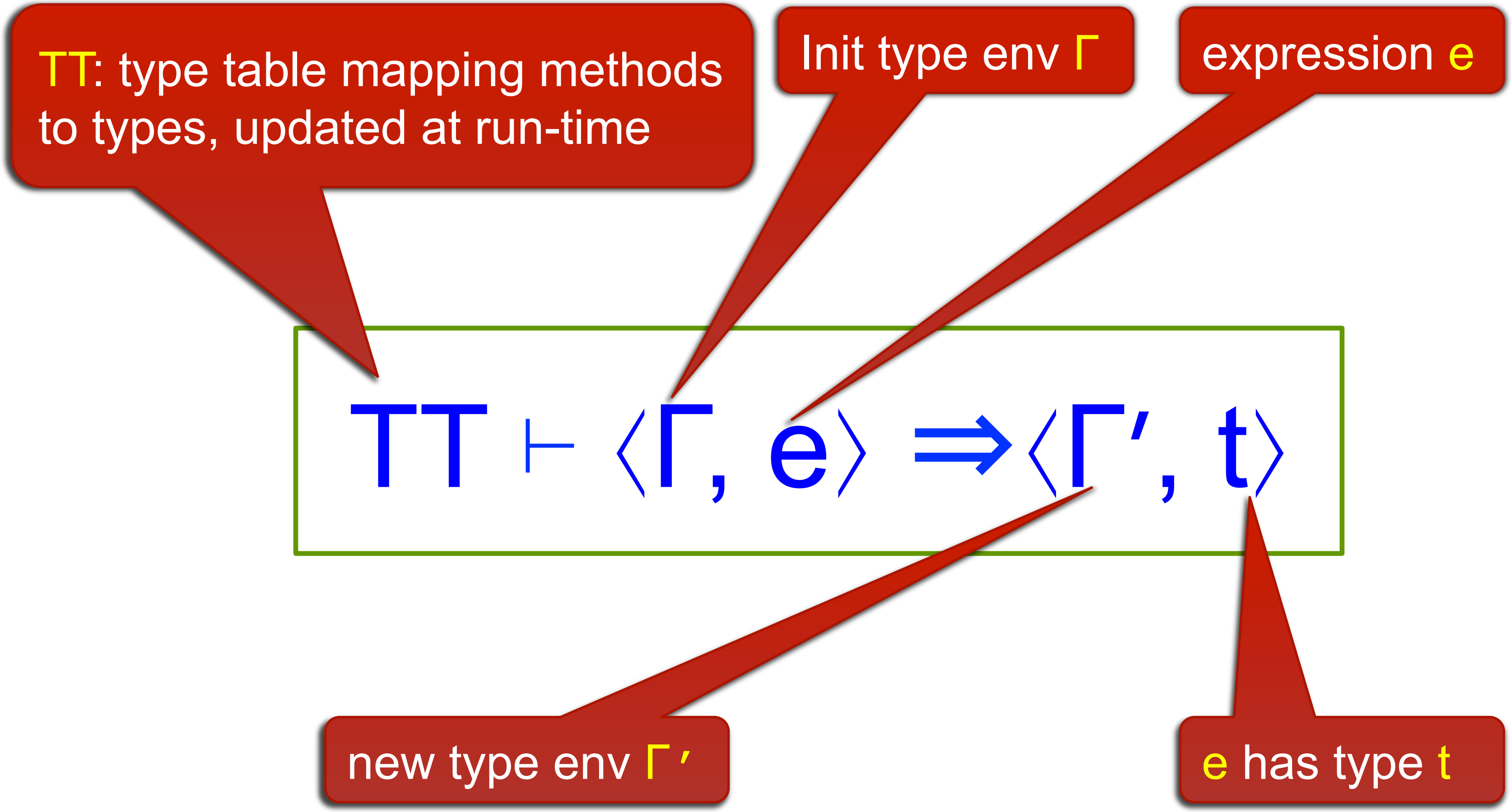
Core Calculus for Hummingbird

- Formalized Hummingbird in a core, Ruby-like language

  ```
  v ::= nil | [A]
  e ::= v | x | self | x = e | e; e | A.new | if e then e else e | e.m(e)
        | def A.m = λx.e | type A.m : t → t
  t ::= A | nil
  ```

  - Method definitions can occur at arbitrary points
  - Type “annotations” execute at run-time
- At e1.m(e2), statically type check body of m
  - Resulting typing proof cached for future reuse

Static Type Checking



TT: type table mapping methods to types, updated at run-time

Init type env Γ

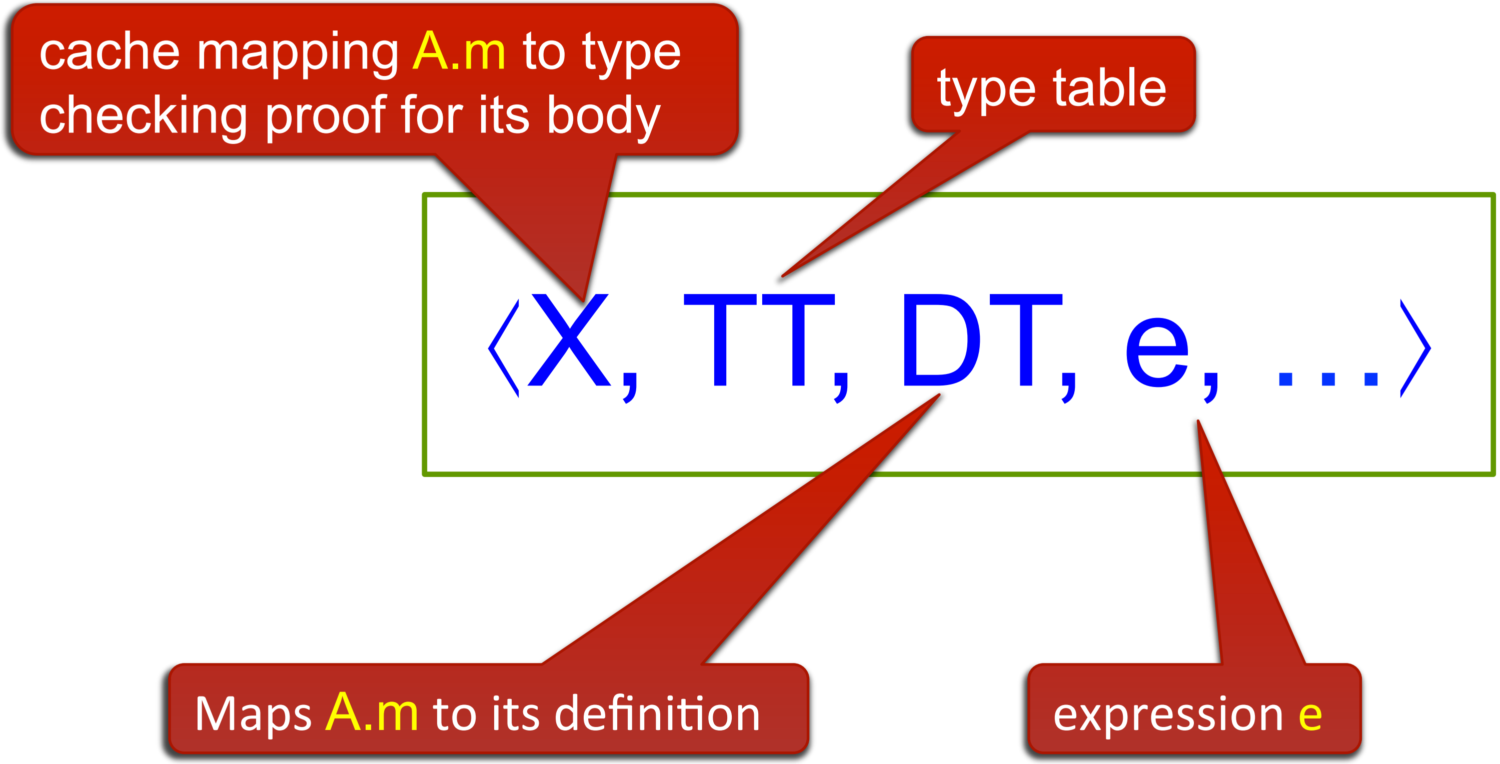
expression e

$$\boxed{TT \vdash \langle \Gamma, e \rangle \Rightarrow \langle \Gamma', t \rangle}$$

new type env Γ′

e has type t

Dynamic Semantics



cache mapping A.m to type checking proof for its body

type table

$\langle X, TT, DT, e, \ldots\rangle$

Maps A.m to its definition

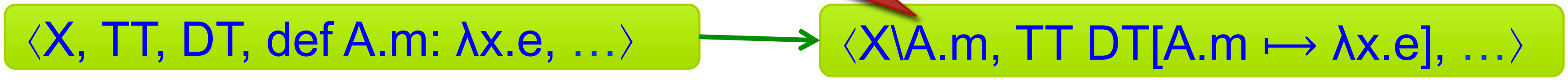
expression e

Type Semantics

$$\begin{tikzcd} \langle X, TT, DT, \text{type } A.m\text{: } (t1) \rightarrow t2, \ldots\rangle \arrow[r] & \langle X, TT[A.m : (t1) \rightarrow t2], DT, \text{nil}, \ldots\rangle \end{tikzcd}$$

Cache Invalidation on Def

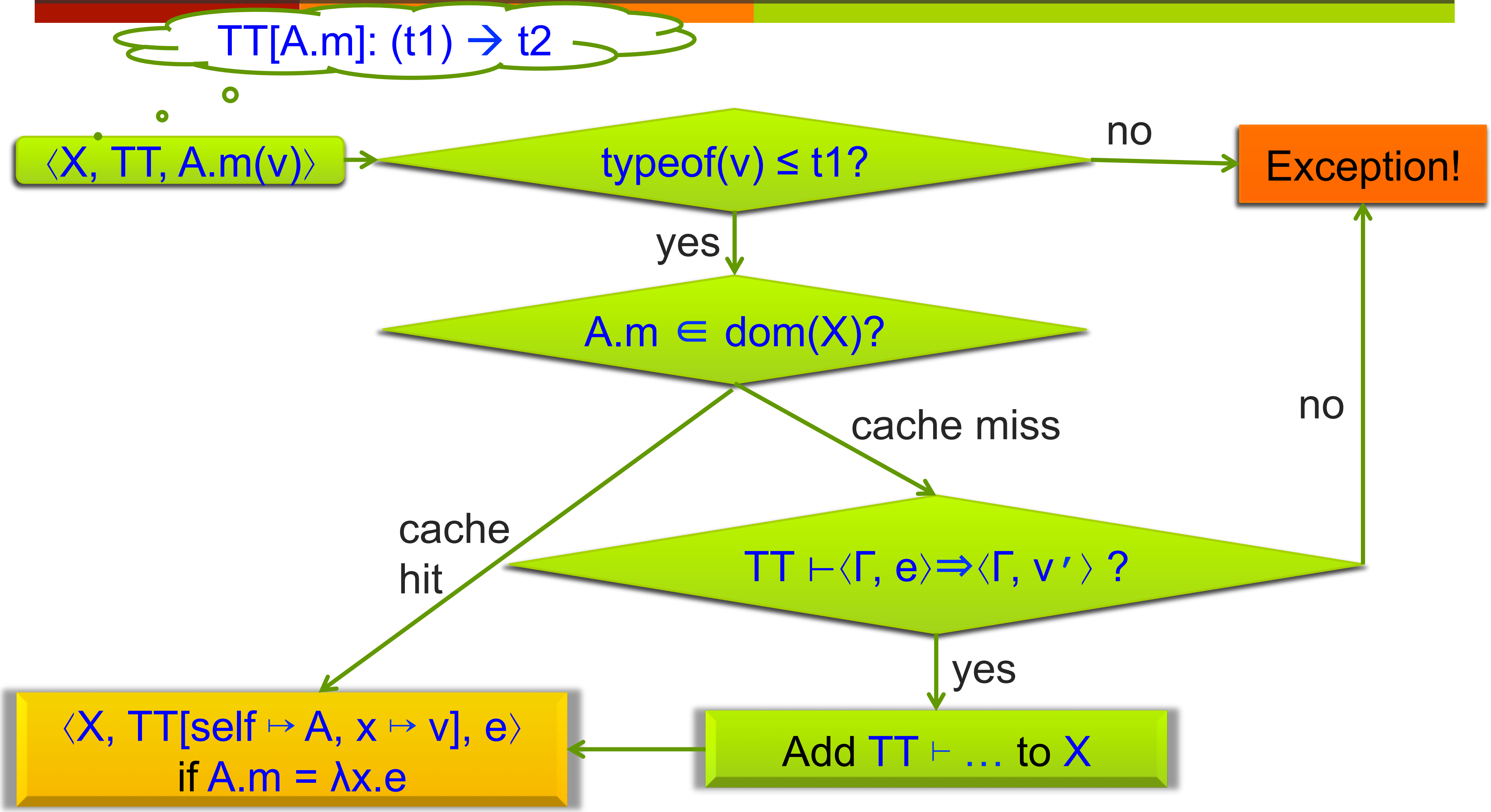
X with proofs for A.m and everything that depends on A.m. removed



⟨X, TT, DT, def A.m: λx.e, …⟩

⟨X\A.m, TT DT[A.m ⟼ λx.e], …⟩

Function Application



$TT[A.m]: (t1) \rightarrow t2$

$\langle X, TT, A.m(v) \rangle$

$typeof(v) \leq t1?$

no

Exception!

yes

$A.m \in dom(X)?$

cache miss

no

cache hit

$TT \vdash \langle \Gamma, e \rangle \Rightarrow \langle \Gamma, v' \rangle$?

yes

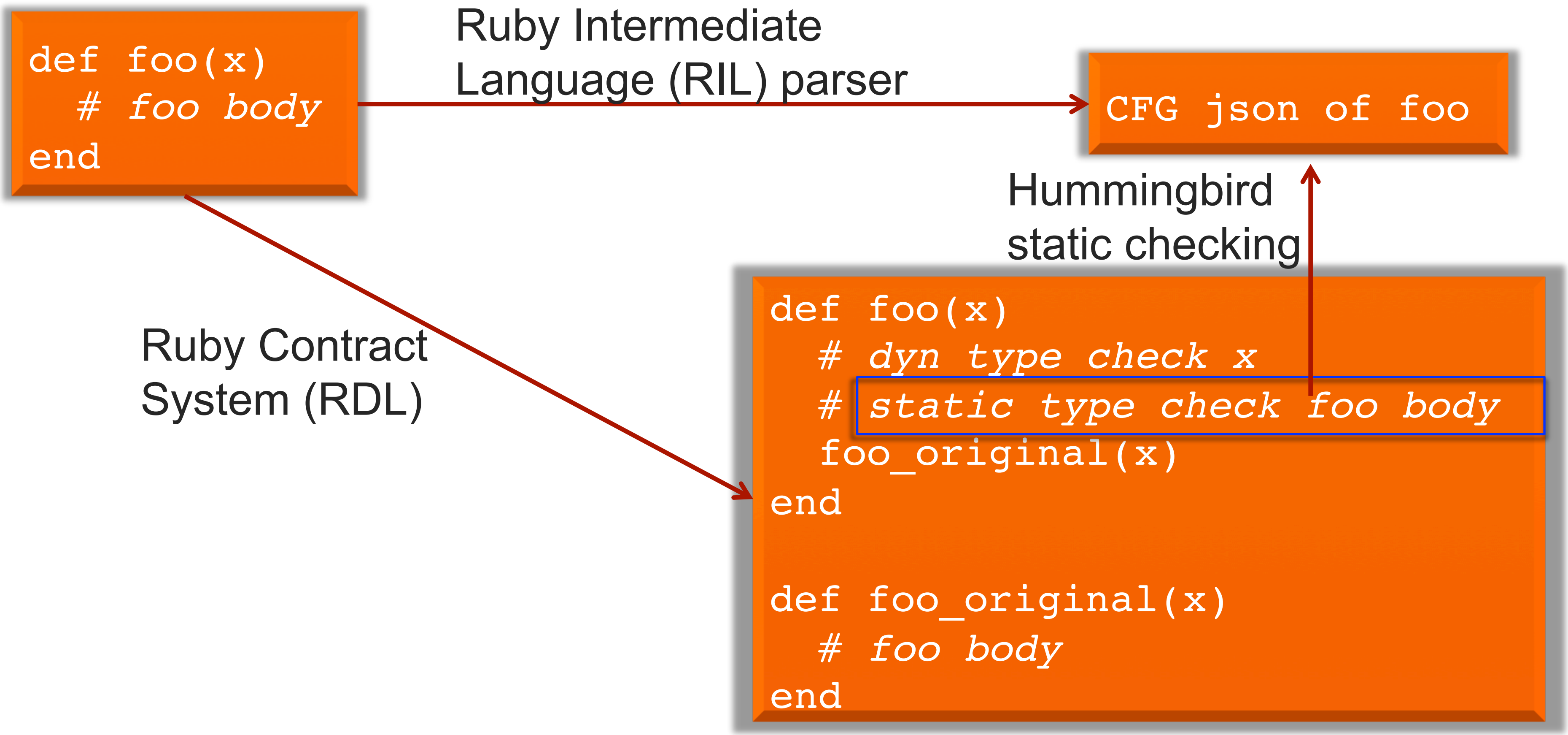
$\langle X, TT[self \mapsto A, x \mapsto v], e \rangle$
if $A.m = \lambda x.e$

Add $TT \vdash \ldots$ to X

Soundness Theorem

- Theorem: Well-typed methods do not go wrong at runtime
  - E.g, they don’t call methods with bad argument types
- Some errors may still occur at runtime
  - Invoking method on nil
  - Calling method whose body does not type check at run-time
  - Calling method that has type annotation but is itself undefined

Implementation



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

Ruby Intermediate Language (RIL) parser

```
CFG json of foo
```

Hummingbird static checking

Ruby Contract System (RDL)

```
def foo(x)
  # dyn type check x
  # static type check foo body
  foo_original(x)
end

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

Cache Invalidation in Rails Dev Mode

- Rails automatically reloads modified files in dev mode
  - Implementation does not support arbitrary invalidation
- Hummingbird invalidates parts of cache on reload
  - Record method dependencies at run-time
  - If method changed/deleted, remove method and its dependencies from cache

Type Casts

```
type(Marshal, ‘self.load’, ‘(…) → Object’)
field_type(:@@cache, “Hash<A, B>”)
r = Marshal.load(…)
@@cache = r     # type error!
```

Type Casts

```
type(Marshal, ‘self.load’, ‘(…) → Object’)
field_type(:@@cache, “Hash<A, B>”)
r = Marshal.load(…)
@@cache = r.rdl_cast(“Hash<A, B>”)
```

- Static type checking: Assume r has casted type
- Run-time: check if r is a member of casted type

Experiments

- Ran on Mac with 2.3 GHz Intel Core i7 and 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 Annotations

- Non-type checked annotations
  - Common Ruby core and standard libraries
    - String, Fixnum, etc
  - Type signatures for common Rails metaprogramming
    - E.g., belongs_to, …
  - App-specific Rails helper methods
- Type checked annotations
  - All application methods (manually annotated)

Type Checking R

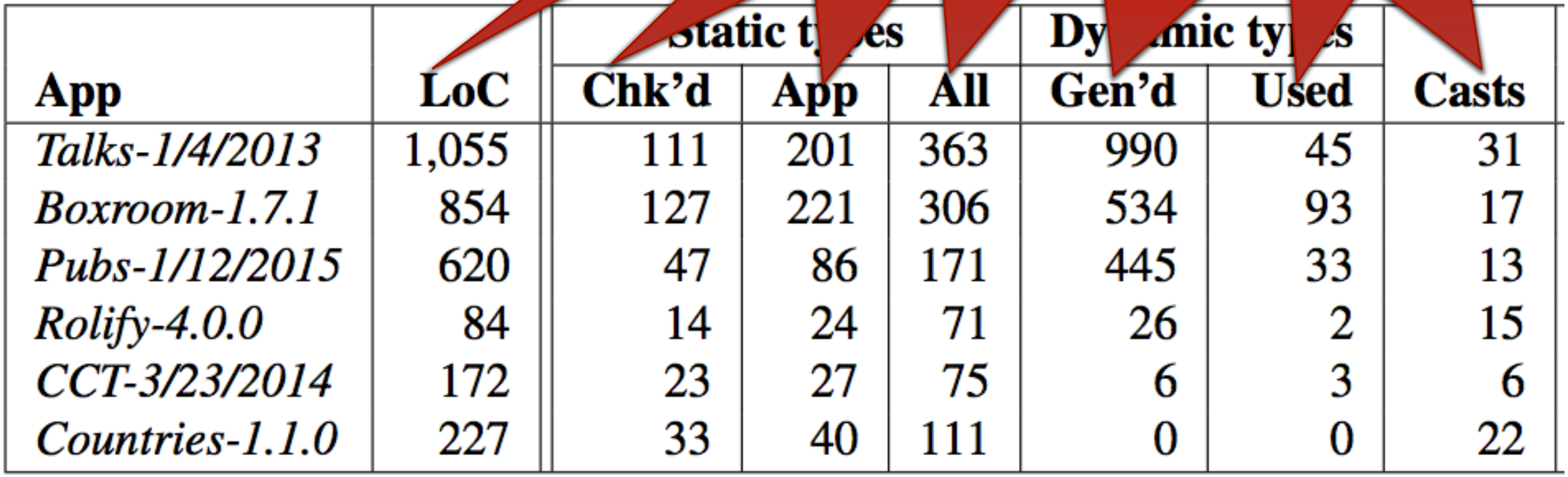
Chk
trust

Ch
Core/standard lib

types that

Safe down casts +
Adding parameters to raw generic types like Array

Dyn generated types



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

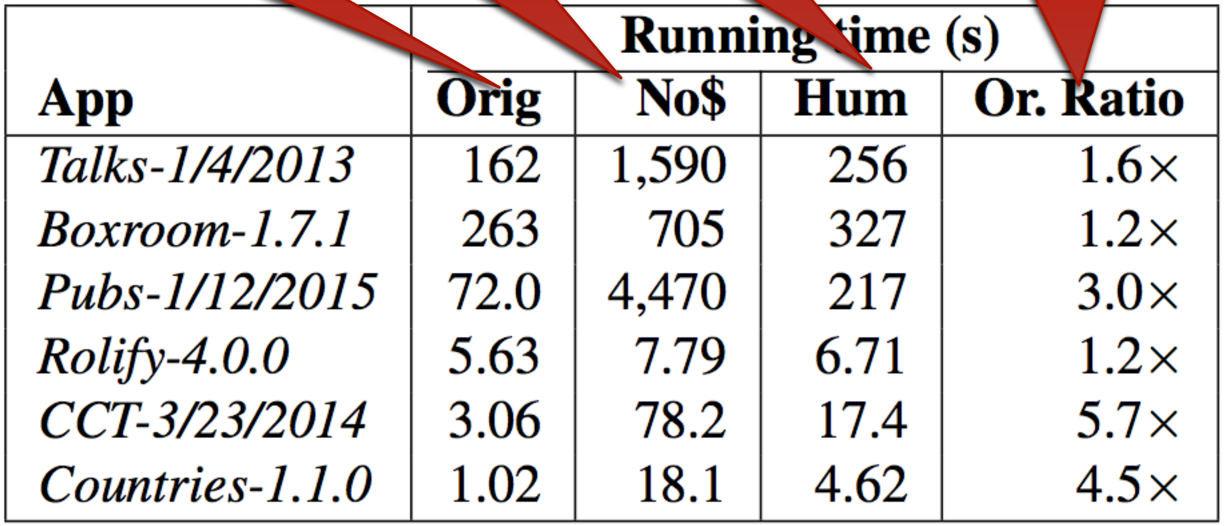
Type Checking Results: Performance

w/o Humm

with Hum
but no ca

with Hummingbird
and caching

Orig vs Hum ratio



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

Rails Live Version U

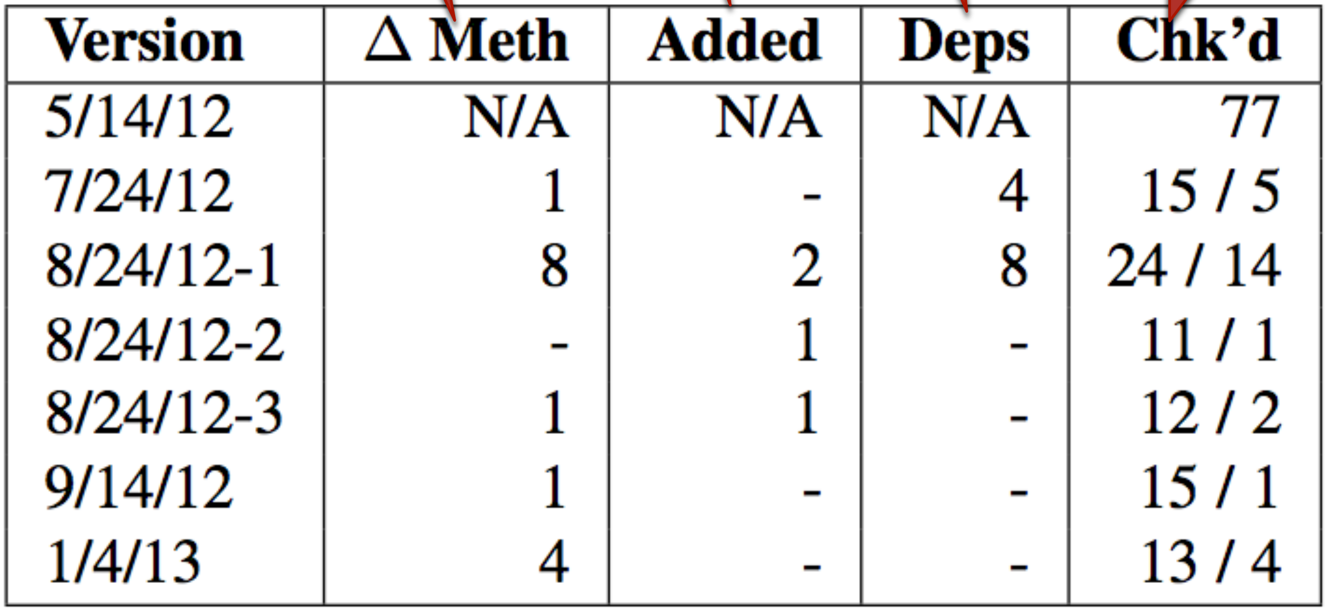
Modified metho

Ne

Depe

Method re-checked (A/B)
A: actual methods rechecked due to limitation in Hummingbird
B: methods rechecked if no limitation
Initial version checked 77 methods

- Updated a Talks sion to 6
  - Modified files a omatically ad
- Cache invalidation u eful since mall nu ber of m ds changed by each upd te



Version	Δ Meth	Added	Deps	Chk'd
5/14/12	N/A	N/A	N/A	77
7/24/12	1	-	4	15 / 5
8/24/12-1	8	2	8	24 / 14
8/24/12-2	-	1	-	11 / 1
8/24/12-3	1	1	-	12 / 2
9/14/12	1	-	-	15 / 1
1/4/13	4	-	-	13 / 4

Conclusions

- Hummingbird: just-in-time static type checking for Ruby
  - Type information tracked dynamically
  - Methods checked statically at run-time when called
- Easy to check code that use metaprogramming-generated methods
- Caching used to eliminate redundant type checking

https://www.cs.umd.edu/~bren/pldi16_artifact/
https://github.com/plum-umd/rdl