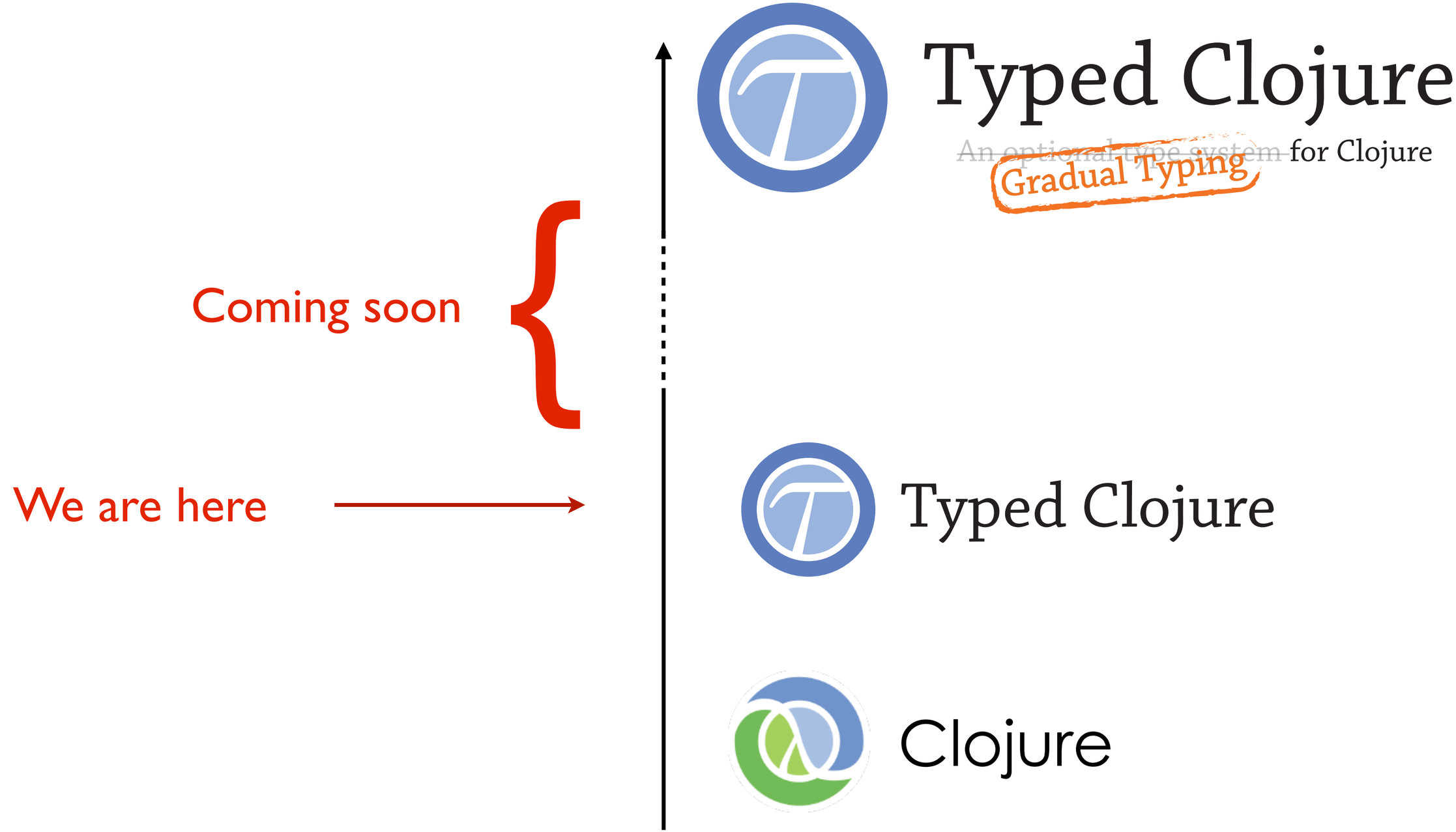
Typed Clojure

~~An optional type system~~ Gradual Typing for Clojure

From Optional to Gradual Typing

Ambrose Bonnaire-Sergeant

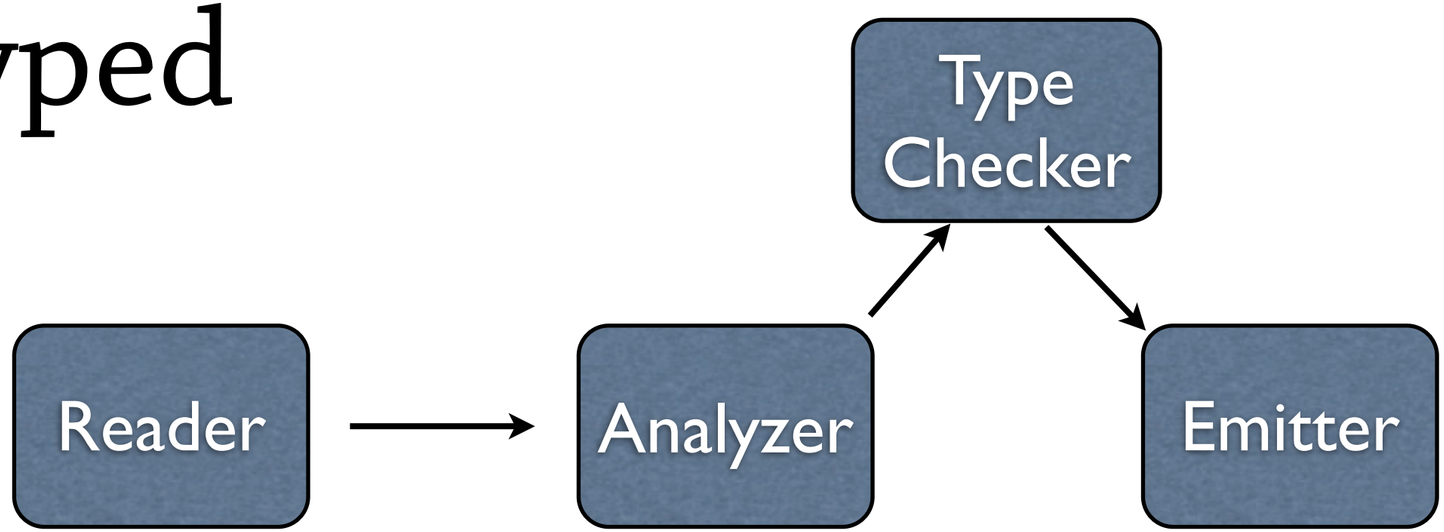

Typed Clojure
~~An optional type system~~ Gradual Typing for Clojure
Coming soon
We are here
Typed Clojure
Clojure

What is Optional Typing?

Untyped

Untyped

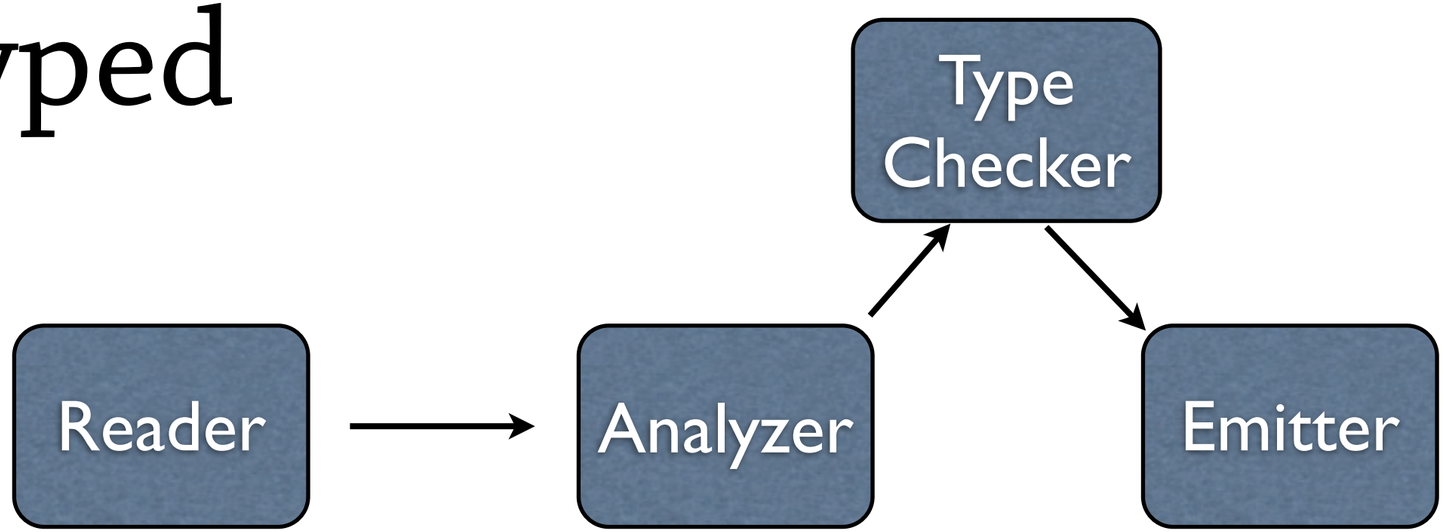
Typed



Optional Types

Untyped

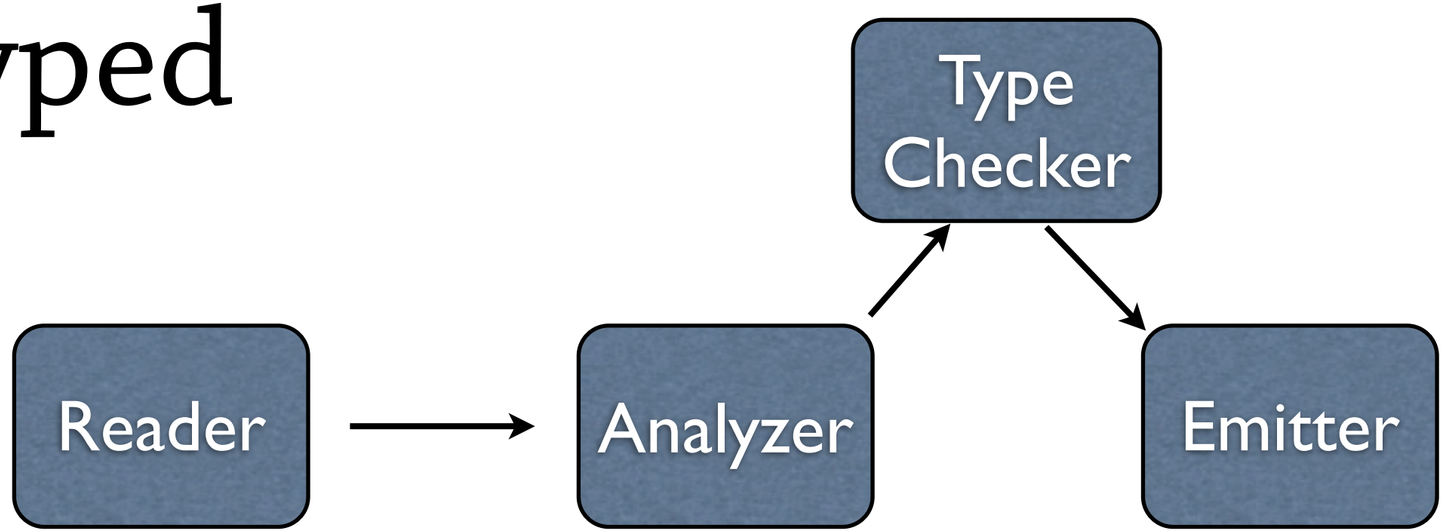
Typed



Swap at will

Untyped

Typed



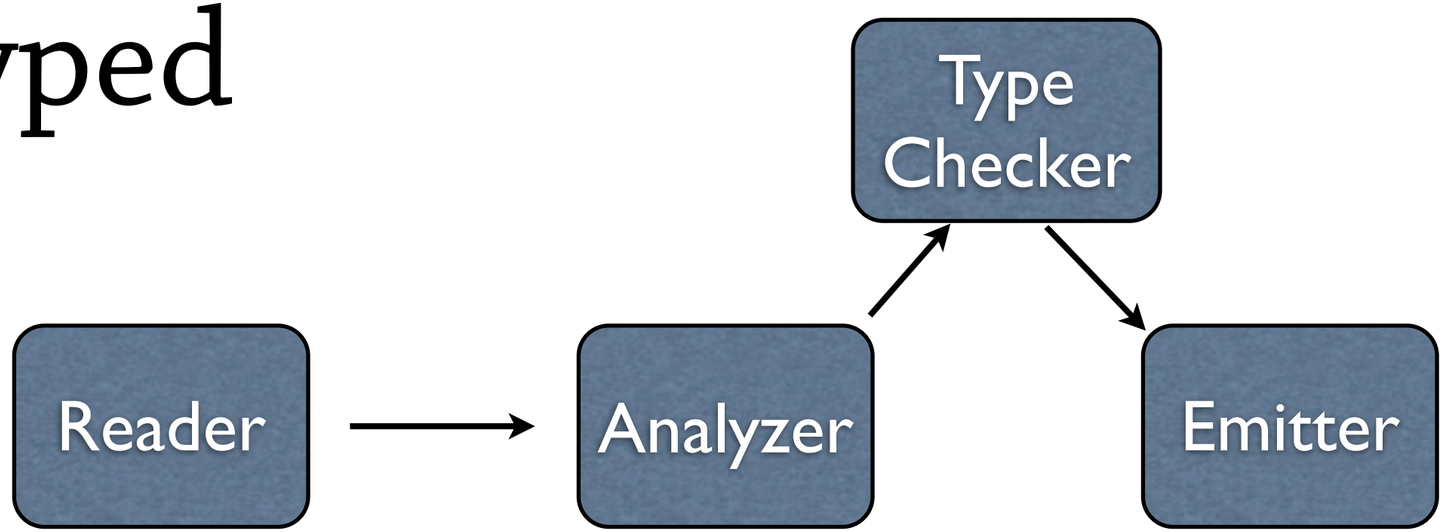
Optional Types

Swap at will

Runtime cannot depend on types

Untyped

Typed



Optional Types

Swap at will

Runtime cannot depend on types

Like a linter in practice

+ Optional Types = Typed Clojure

Var annotation

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

Machine-checked documentation

Normal Clojure code

Heterogeneous map type

```
(ann day [`{:day Int} -> Int])
(defn day [{d :day}] d)
```

Normal destructuring

Explicit nil/null

```
(ann get-parent [File -> (U nil Str)])
(defn get-parent [f]
  (.getParent f))
```

Java interop

Ad-hoc unions

```
(ann safe-inc [(U nil Int) -> Int])
(defn safe-inc [n]
  (if n
    (inc n)
    0))
```

Conditional flow

Never null

Type alias

```
(defalias Expr
  (U `{:op (Val :do), :exprs (Vec Expr)}
     `{:op (Val :val), :val Int}))

;; eg. {:op :do, :exprs [{:op :val, :val 1}]}
```

Arbitrary multimethod dispatch

```
(ann f [Expr -> Int])
(defmulti f :op)
(defmethod f :do [{exprs :exprs}]
  (apply + (map f exprs)))
(defmethod f :val [{val :val}]
  val)

(f {:op :do,
    :exprs [{:op :val, :val 30},
            {:op :val, :val 12}]})
;=> 42
```

Arbitrary multimethod dispatch

```
(ann f [Expr -> Int])
(defmulti f :op)
(defmethod f :do [{exprs :exprs}]
  (apply + (map f exprs)))
(defmethod f :val [{val :val}]
  val)

(f {:op :do,
    :exprs [{:op :val, :val 30},
            {:op :val, :val 12}]})
;=> 42
```

Arbitrary multimethod dispatch

```
(ann f [Expr -> Int])
(defmulti f :op)
(defmethod f :do [{exprs :exprs}]
  (apply + (map f exprs)))
(defmethod f :val [{val :val}]
  val)

(f {:op :do,
    :exprs [{:op :val, :val 30},
            {:op :val, :val 12}]})
;=> 42
```

Arbitrary multimethod dispatch

```
(ann f [Expr -> Int])
(defmulti f :op)
(defmethod f :do [{exprs :exprs}]
  (apply + (map f exprs)))
(defmethod f :val [{val :val}]
  val)

(f {:op :do,
    :exprs [{:op :val, :val 30},
            {:op :val, :val 12}]})
;=> 42
```

Arbitrary multimethod dispatch

```
(ann f [Expr -> Int])
(defmulti f :op)
(defmethod f :do [{exprs :exprs}]
  (apply + (map f exprs)))
(defmethod f :val [{val :val}]
  val)

(f {:op :do,
    :exprs [{:op :val, :val 30},
            {:op :val, :val 12}]})
;=> 42
```

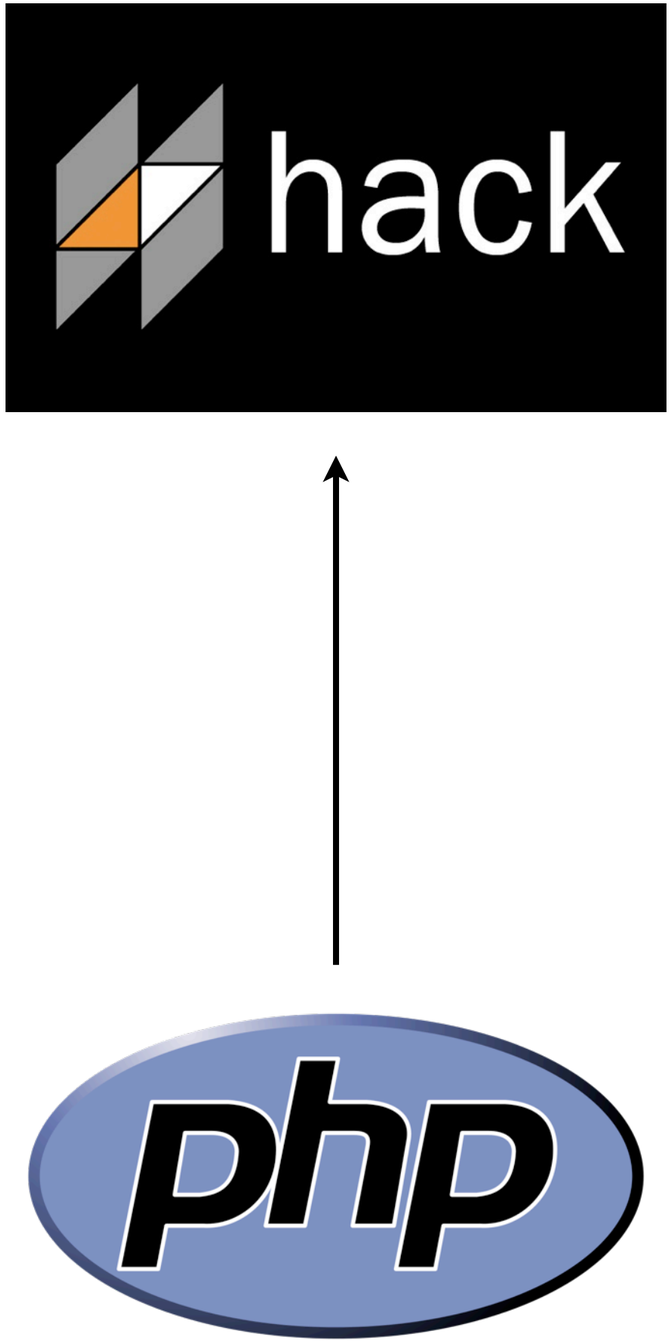
Arbitrary multimethod dispatch

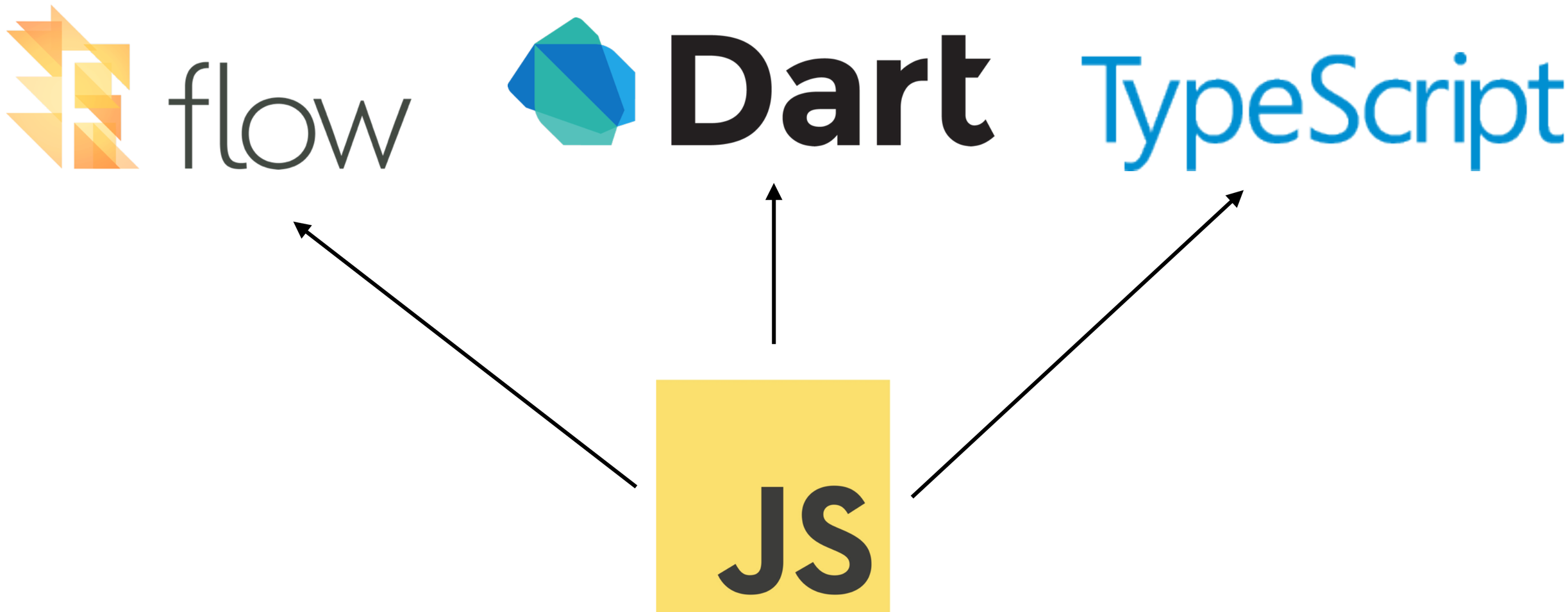
```
(ann f [Expr -> Int])
(defmulti f :op)
(defmethod f :do [{exprs :exprs}]
  (apply + (map f exprs)))
(defmethod f :val [{val :val}]
  val)

(f {:op :do,
    :exprs [{:op :val, :val 30},
            {:op :val, :val 12}]})
;=> 42
```

Arbitrary multimethod dispatch

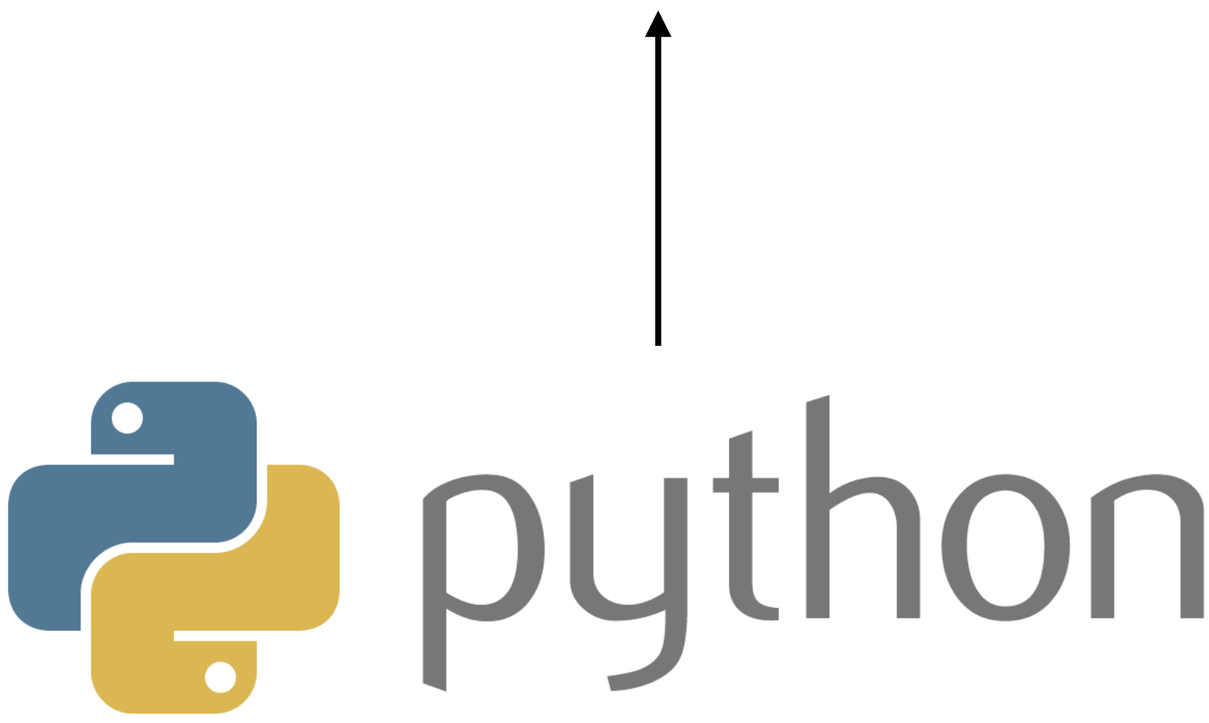
```
(ann f [Expr -> Int])
(defmulti f :op)
(defmethod f :do [{exprs :exprs}]
  (apply + (map f exprs)))
(defmethod f :val [{val :val}]
  val)

(f {:op :do,
    :exprs [{:op :val, :val 30},
            {:op :val, :val 12}]})
;=> 42
```

Optional Typing is all the rage


hack
php


flow
Dart
TypeScript
JS

mypy



Typed Clojure in Practice

circleci

Home About Blog Contact Login

Why we're supporting Typed Clojure, and you should too!

by circleci on September 27, 2013

tl;dr Typed Clojure is an important step for not just Clojure, but all dynamic languages. CircleCI is supporting it, and you should too.

Typed Clojure is one of the biggest advancements to dynamic programming languages in the last few decades. It shows that you can have the amazing flexibility of a dynamic language, while providing lightweight, optional typing. Most importantly, this can make your team more productive, and it's ready to use in production.

Even if you don't use Clojure, you should support the Typed Clojure campaign, because its success will help developers in your language realize how great optional typing can be in everyday code. Whether you write Ruby or Python or JavaScript or whatever, what we're learning from Typed Clojure can be applied to your language.

Why optional typing?

Dynamic languages have long been criticised for being hard to maintain at scale. When you grow to a large team or a large code base, it becomes more difficult to refactor a code base, to understand how it works, and to make sure it does what it should.

Sign Up For Updates From Our Blog

Follow Subscribe

Begin Free Trial

2013

CircleCI Data

- 2 year trial
- 87 typed namespaces
- 105 Java interactions
- 328 HMap operations
- 11 multimethods, 89 defmethods
- 407 (22%) checked def's, 1427 (78%) unchecked

circleci

Home About Blog Contact

Why we're no longer using Core.typed

by Marc O'Morain on September 23, 2015

In September 2013 we blogged about why we're supporting Typed Clojure, and you should too! Now, 2 years later, our engineering team has made a collective decision to stop using Typed Clojure (specifically the core.typed library). As part of this decision, we wanted to write a blog-post about our experience using core.typed.

The reason that we decided to stop using core.typed was because we found that the cost of using it was greater than the benefit we gained. This is a subjective view, of course, so we will detail our reasoning below.

The core.typed library is part of the Typed Clojure project. It is a library that adds optional typing to Clojure code. Core.typed allows the developer to add type-annotations to Clojure forms, and then a type-checking process can be run to verify the type-information of your program.

First of all, some background on our project. The repo for our main app contains 92,983

Sign Up For Updates From Our

Follow Subscr

Begin Free Trial

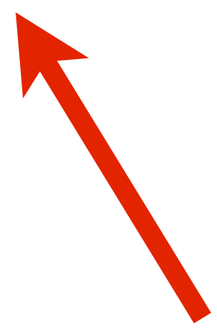
2015

Postmortem

- Slow type checking
- Incomplete support for Clojure idioms
- Missing third-party annotations

Postmortem

- Slow type checking
- Incomplete support for Clojure idioms
- Missing third-party annotations



407 (22%) checked def's, 1427 (78%) unchecked

Pitch: Gradual Typing

Check the 22% at compile-time

Check the 78% at runtime

Sanely handle interaction

407 (22%) checked def's, 1427 (78%) unchecked

What is Gradual Typing?

Gradual typing forces all code to respect static invariants

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

Optional Typing

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

```
(square 2)
;=> 4
```

```
(square 2)
;=> 4
```

Optional Typing

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

```
(square 2)
;=> 4

(square nil)
;
; Expected Int,
;  found nil
```

```
(square 2)
;=> 4
```

Optional Typing

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

```
(square 2)
;=> 4

(square nil)
;
; Expected Int,
;  found nil
```

```
(square 2)
;=> 4
```

Optional Typing

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

```
(square 2)
;=> 4

(square nil)
;
; Expected Int,
;  found nil
```

```
(square 2)
;=> 4

(square nil)
; Exception:
;   NullPointerException
; ...
```

Optional Typing

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

```
(square 2)
;=> 4

(square nil)
;
; Expected Int,
;  found nil
```

```
(square 2)
;=> 4

(square nil)
; Exception:
;   NullPointerException
; ...
```

Optional Typing

Typed Clojure

~~An optional type system~~ Gradual Typing for Clojure

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

```
(square 2)
;=> 4
```

```
(square 2)
;=> 4
```

Gradual Typing

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

```
(square 2)
;=> 4

(square nil)
;
; Expected Int,
;  found nil
```

```
(square 2)
;=> 4
```

Gradual Typing

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

```
(square 2)
;=> 4

(square nil)
;
; Expected Int,
;  found nil
```

```
(square 2)
;=> 4

(square nil)
;
; Expected Int,
;  found nil
```

Gradual Typing

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

```
(square 2)
;=> 4

(square nil)
```

Gradual Typing

```
(ann square [Int -> Int])
(defn square [a] (* a a))
```

```
(square 2)
;=> 4
```

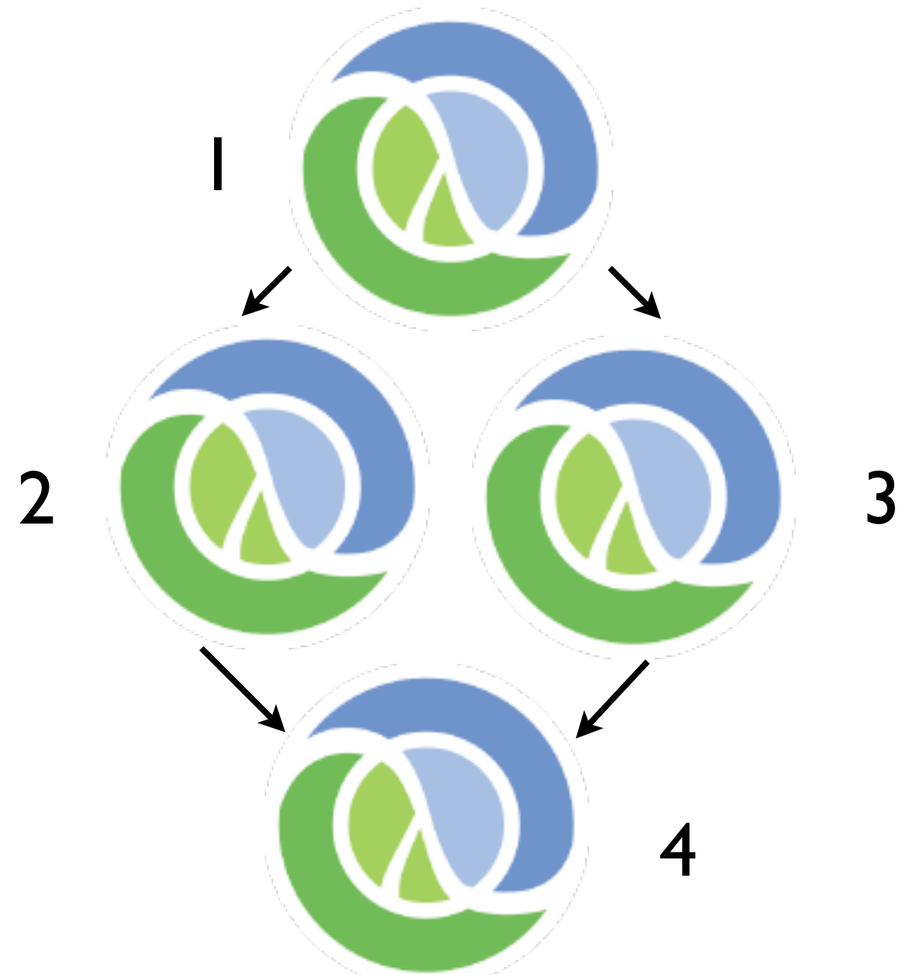
```
(square nil)
```

Compiles to:

```
(square (cast Int nil))
```

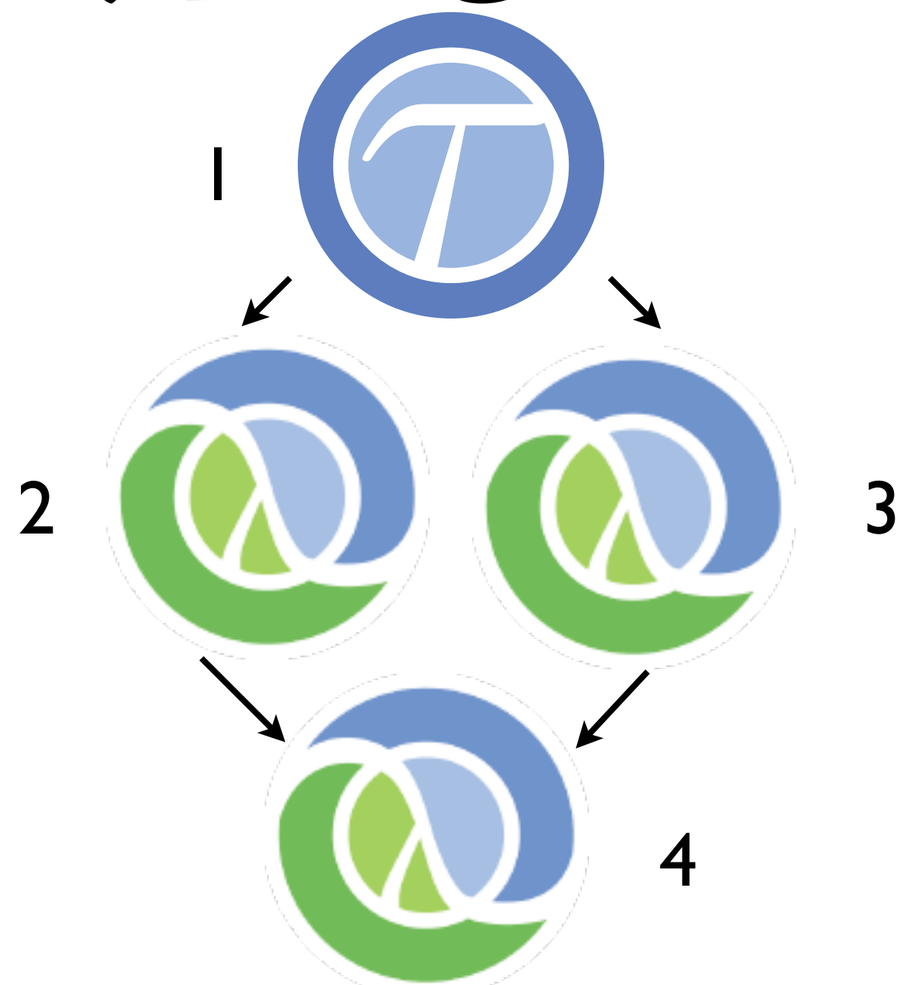
Gradual Typing

Optional Typing



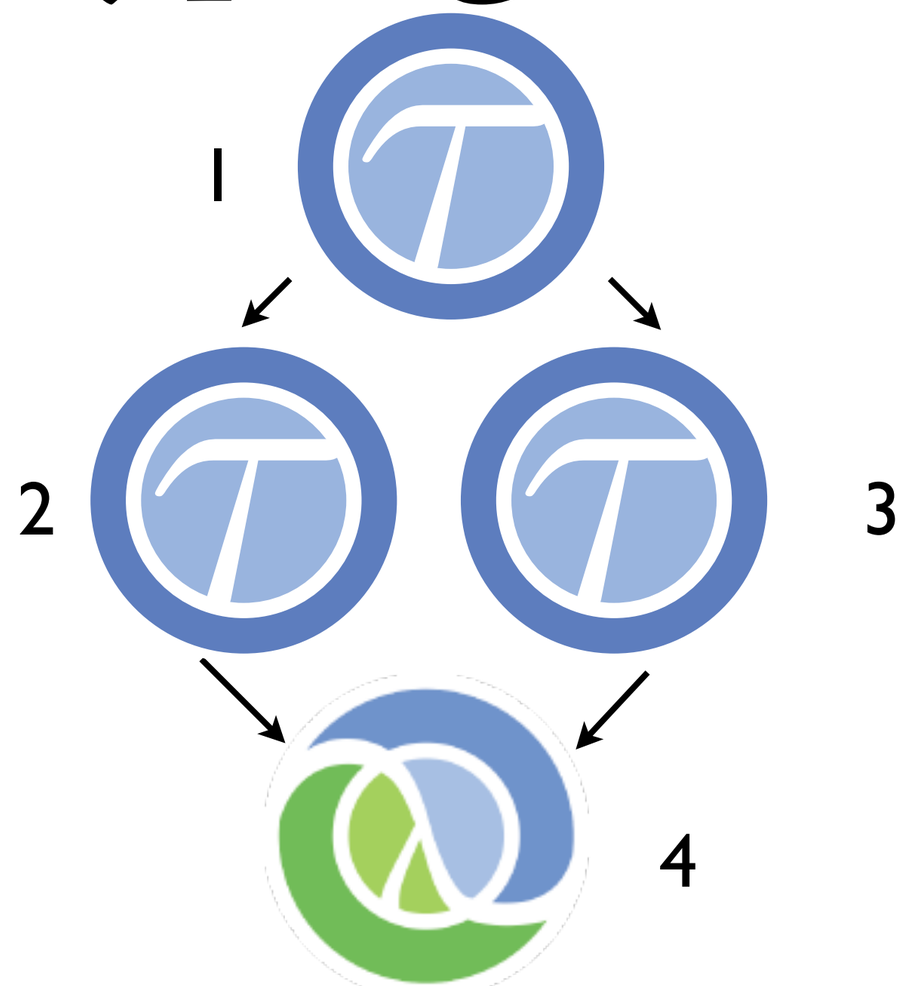
Depends on

Optional Typing



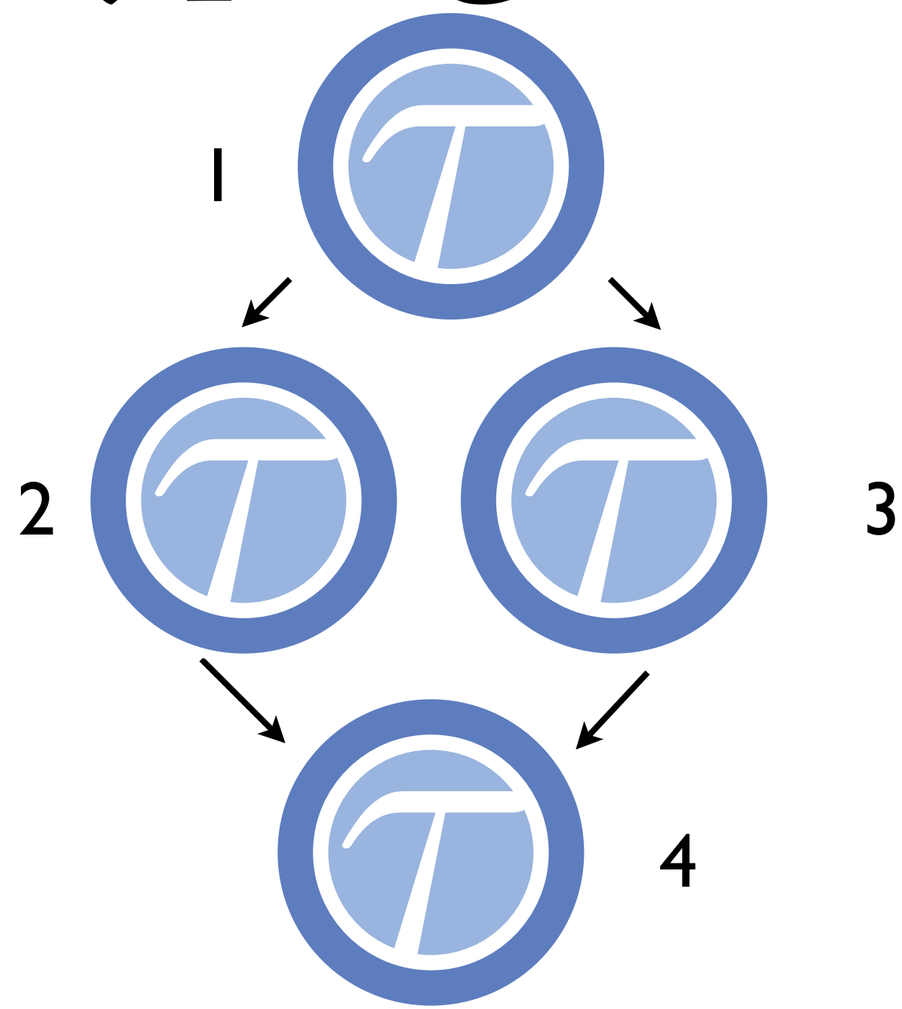
Depends on

Optional Typing



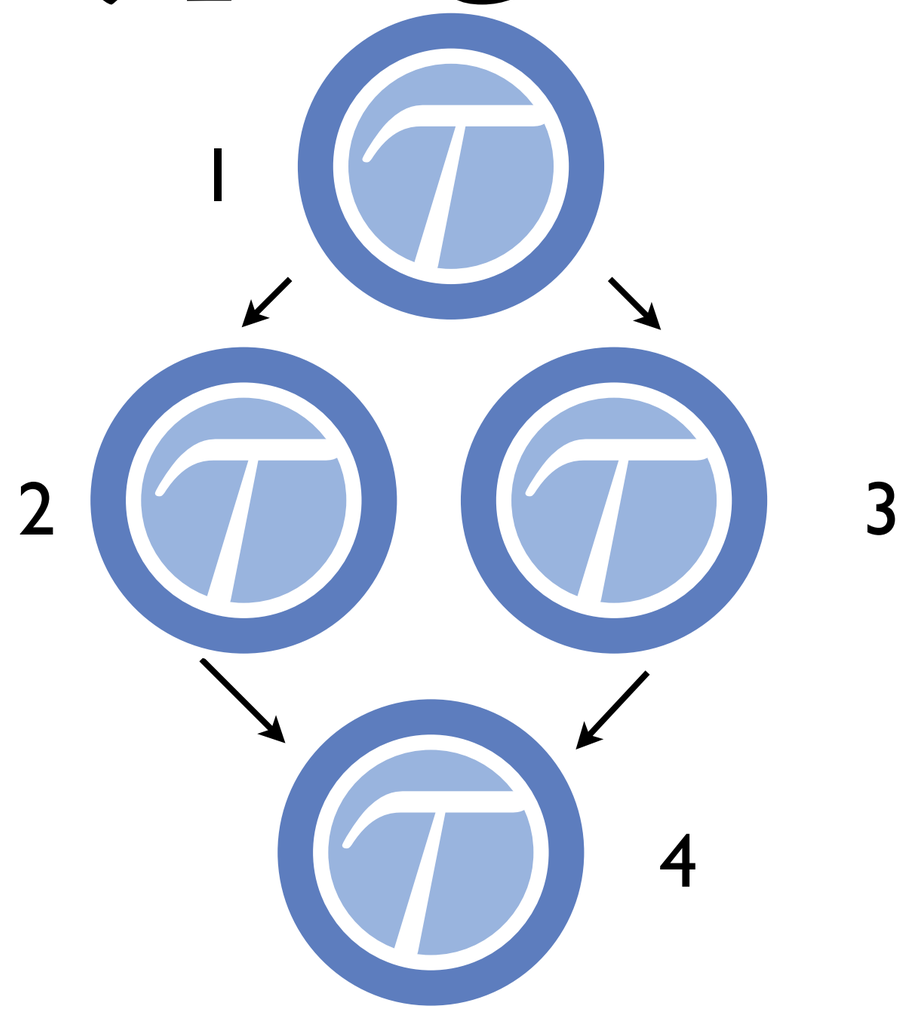
Depends on

Optional Typing



Depends on

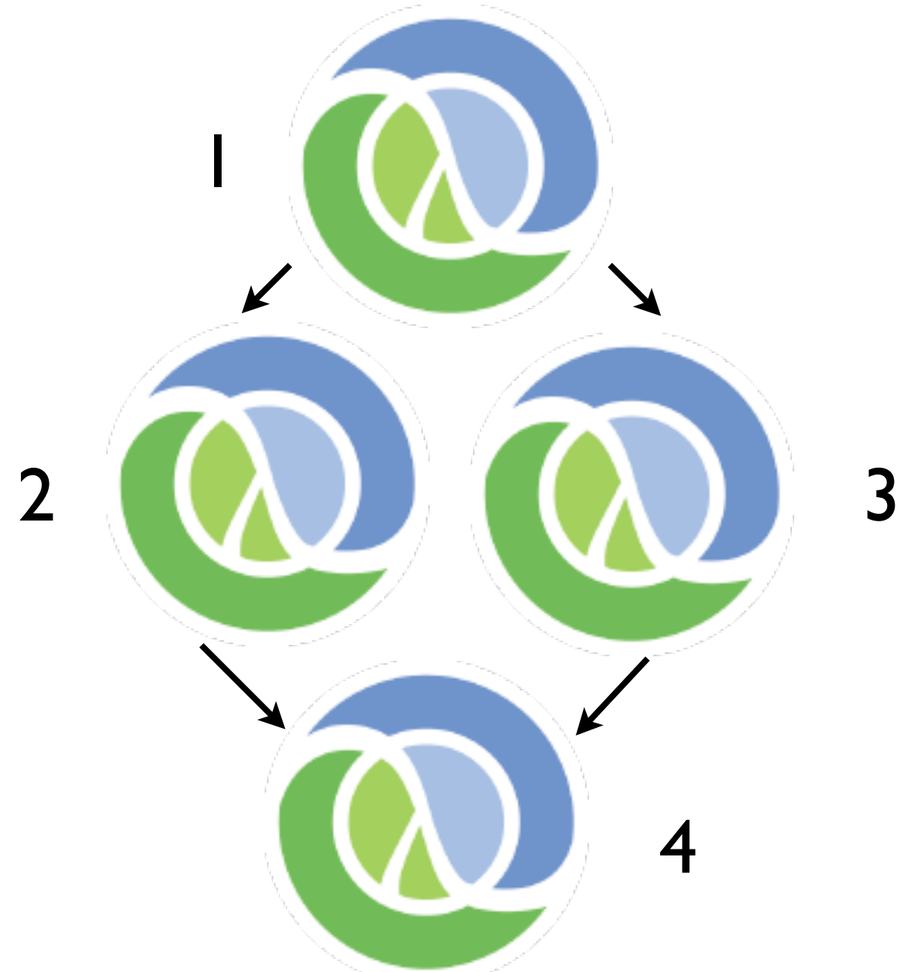
Optional Typing



Depends on

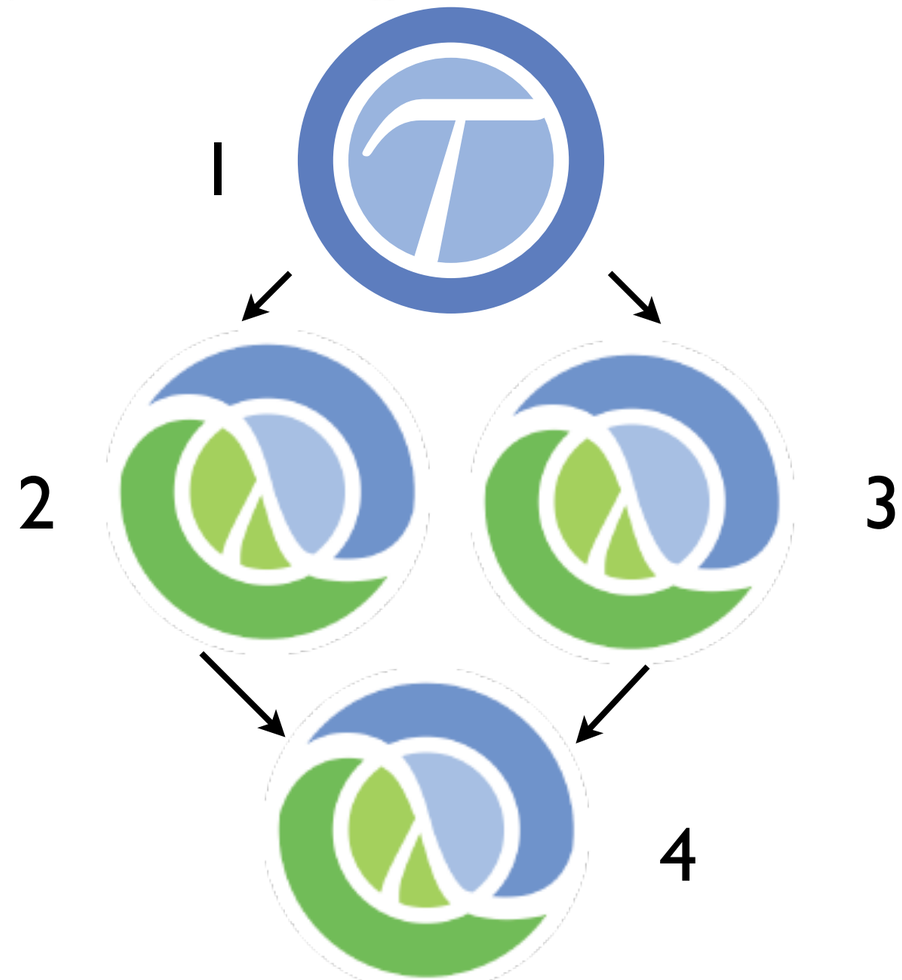
Typed invariants cannot be violated

Gradual Typing



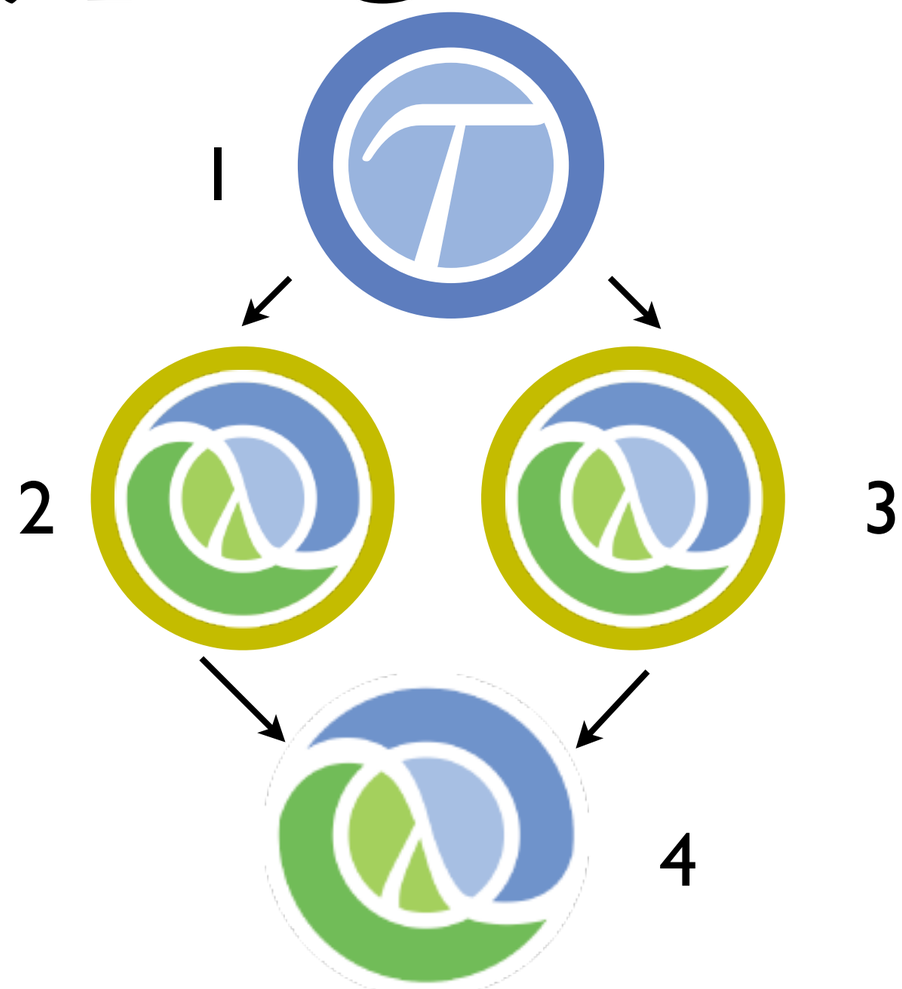
Depends on

Gradual Typing



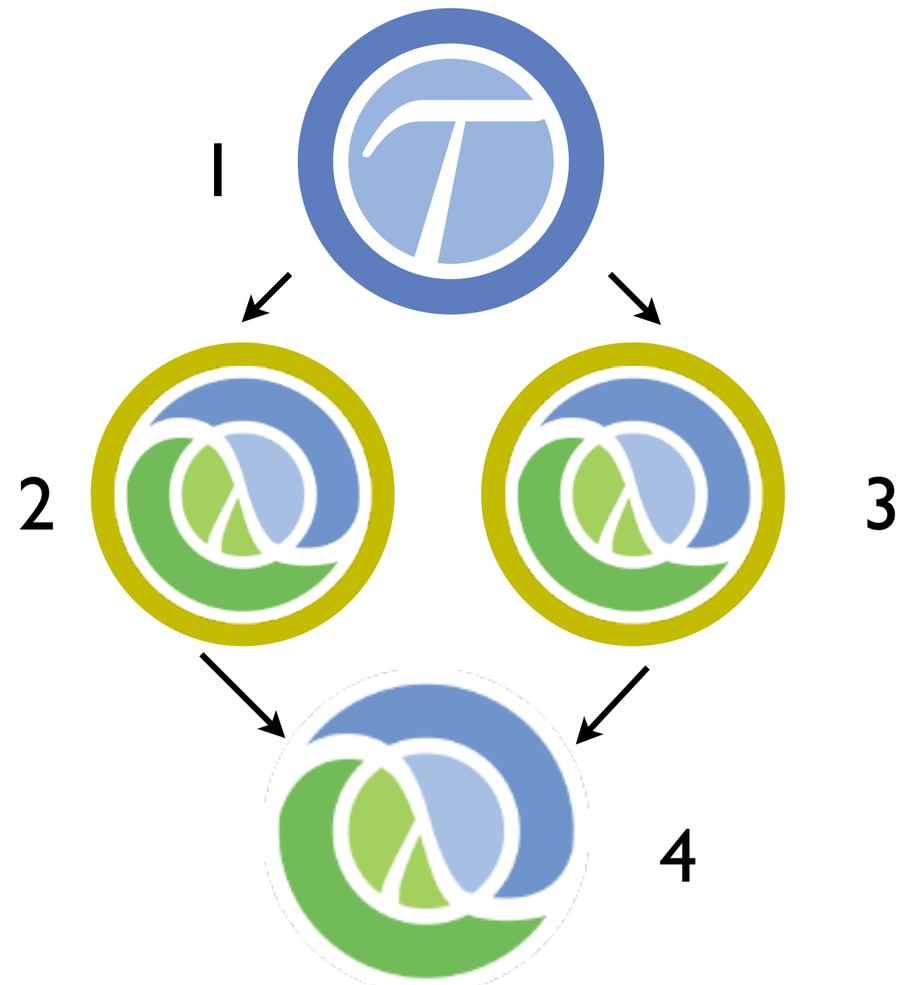
Depends on

Gradual Typing



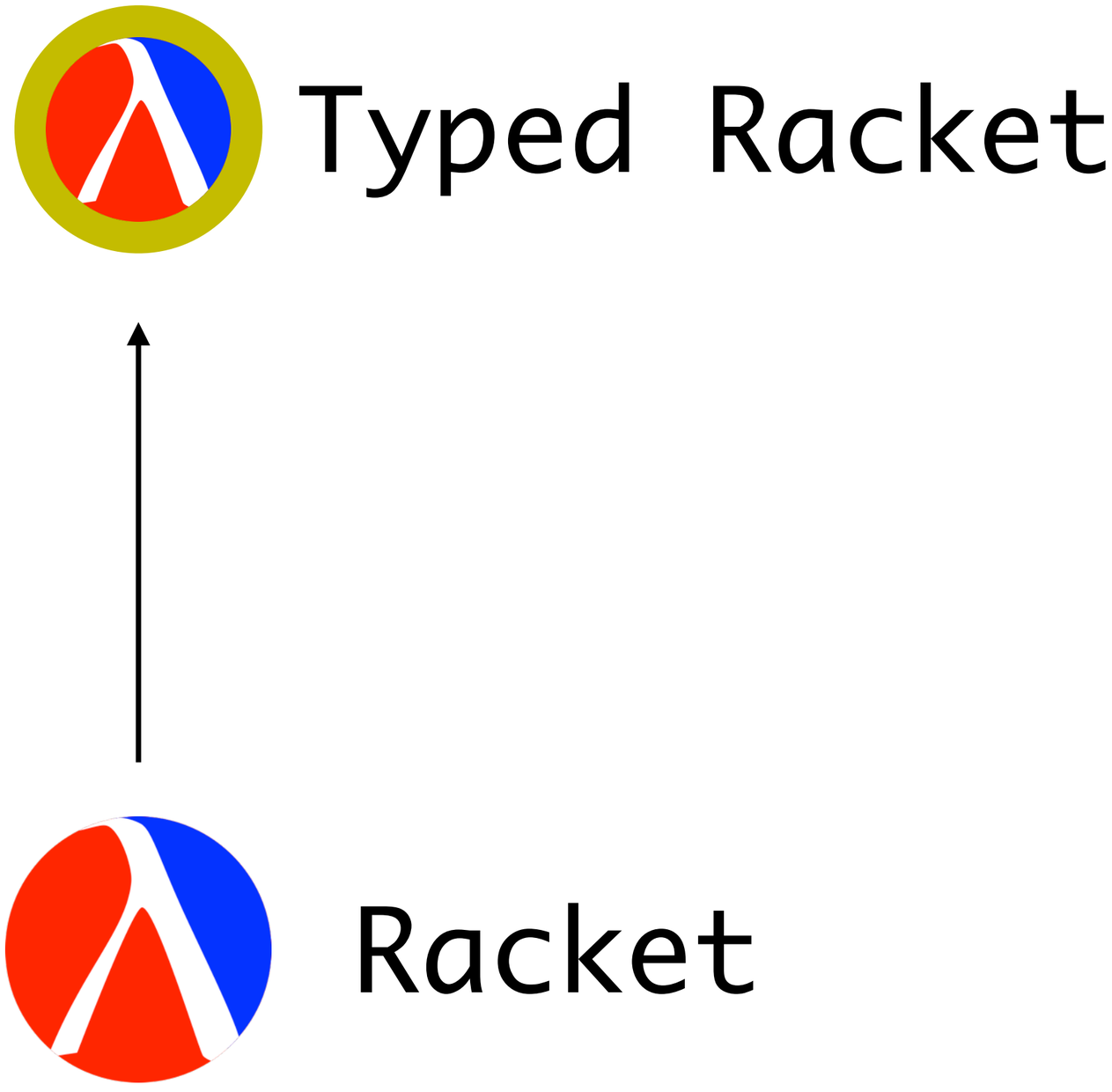
Depends on

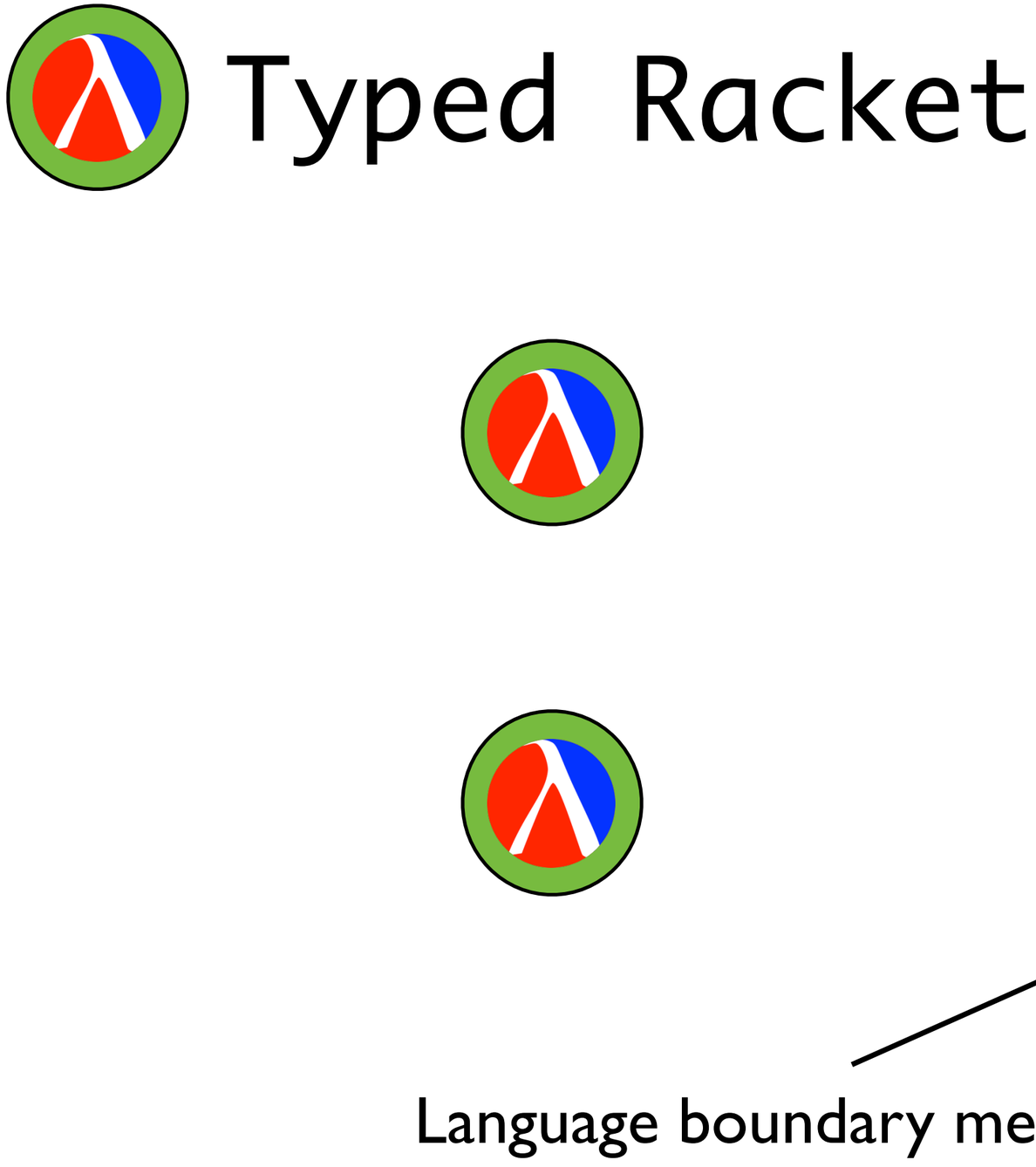
Gradual Typing

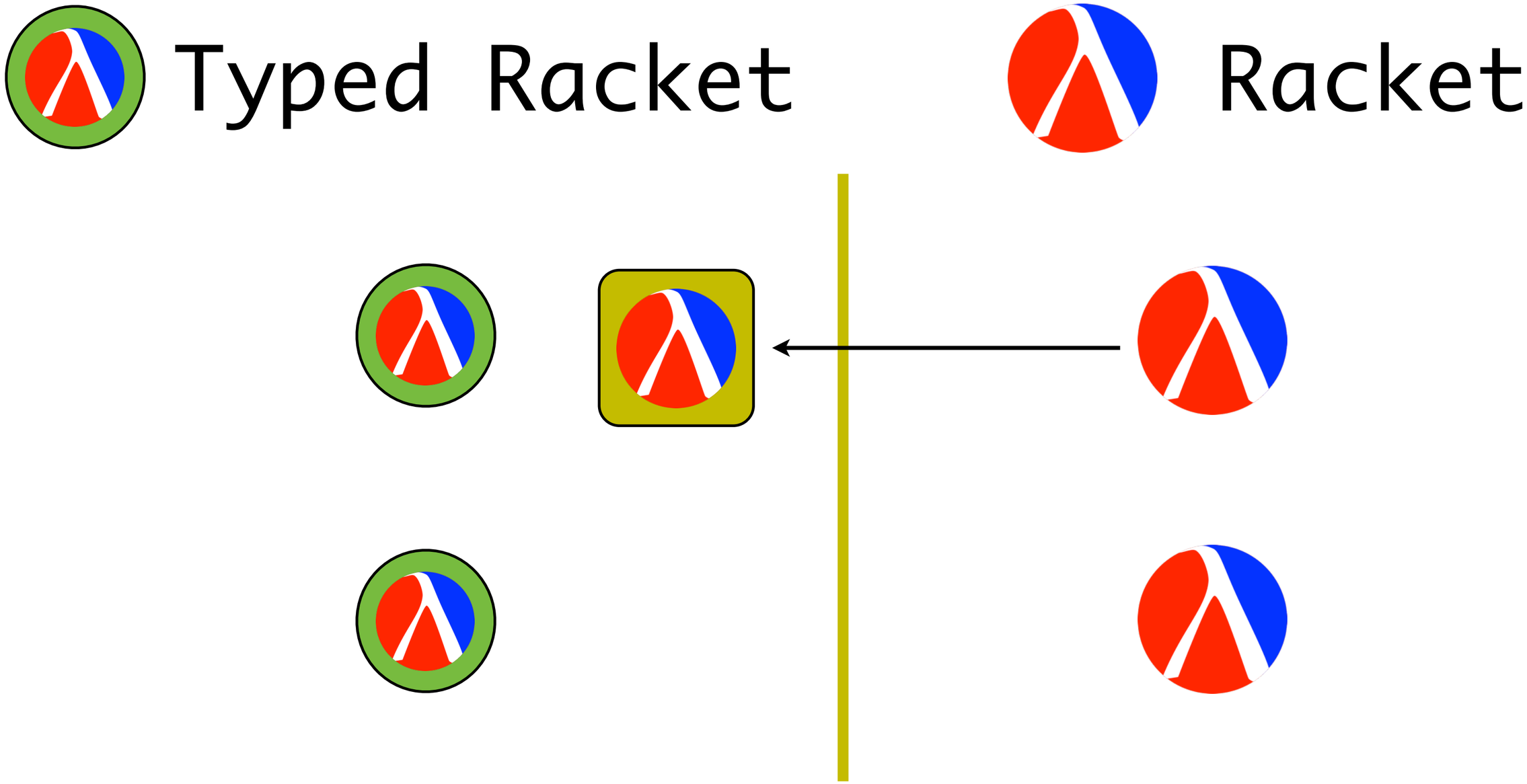


Depends on

Typed invariants cannot be violated

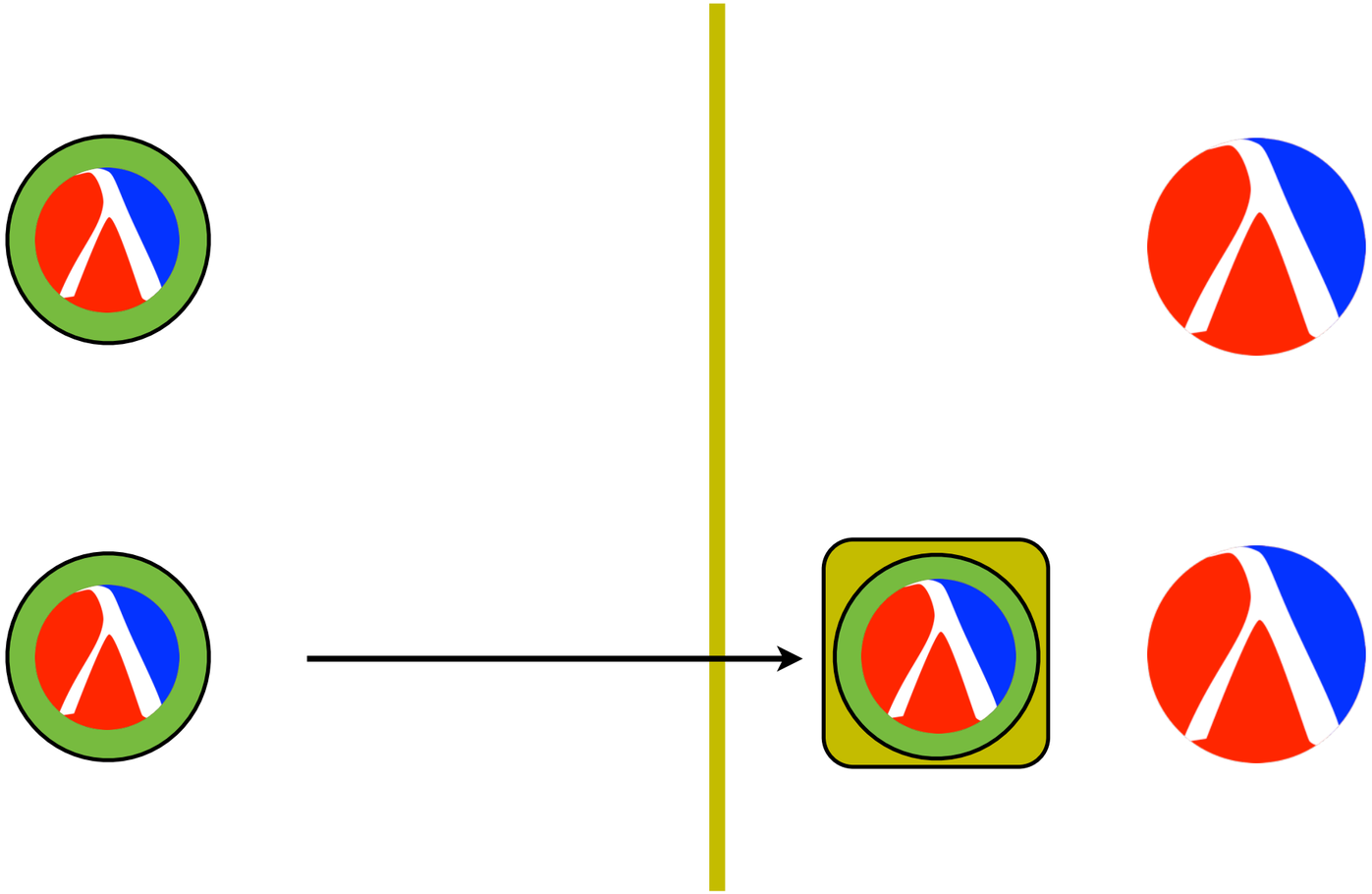

Typed Racket
Racket


Typed Racket
Racket
Language boundary mediator


Typed Racket
Racket

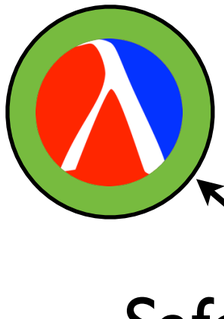

Typed Racket


Racket

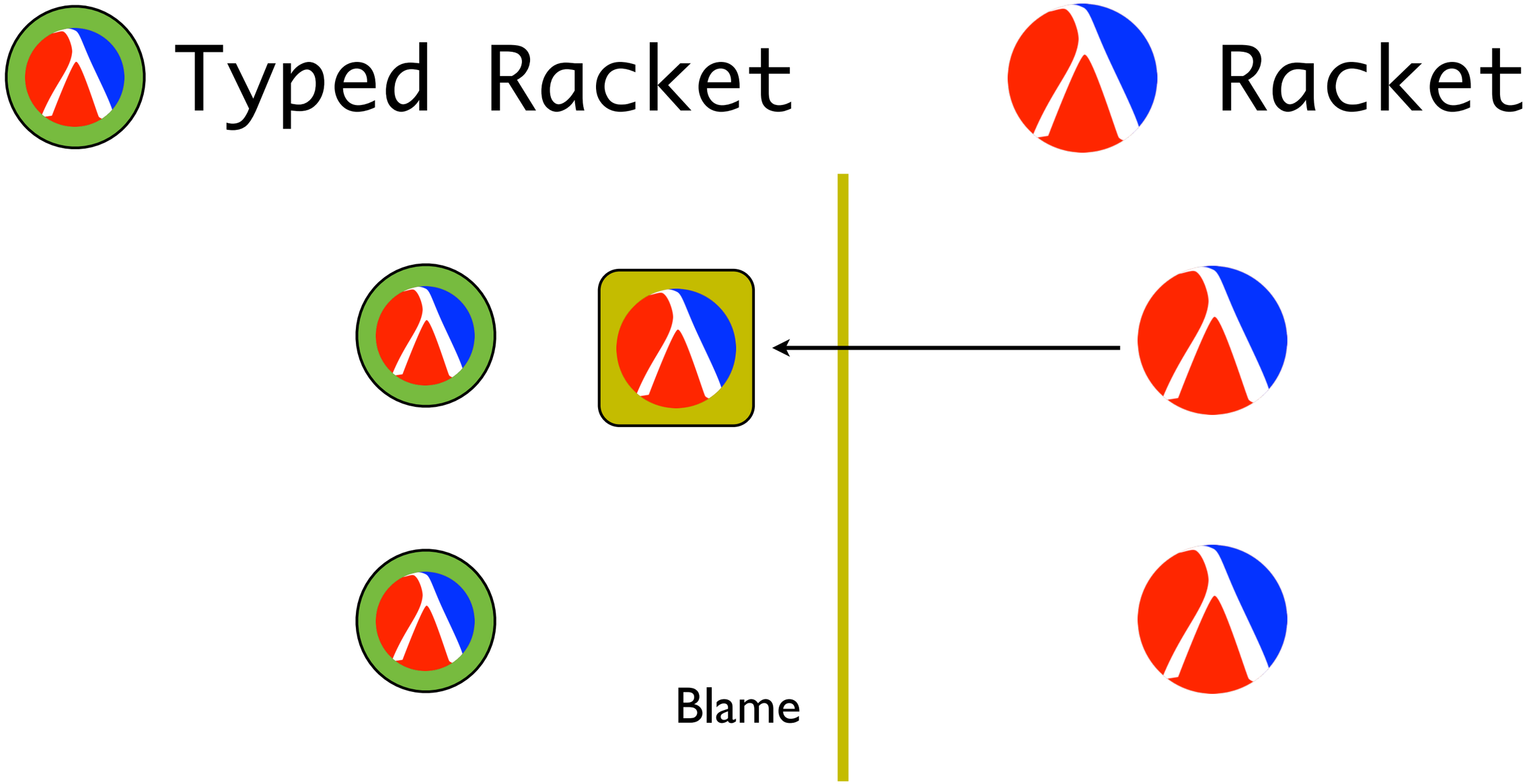


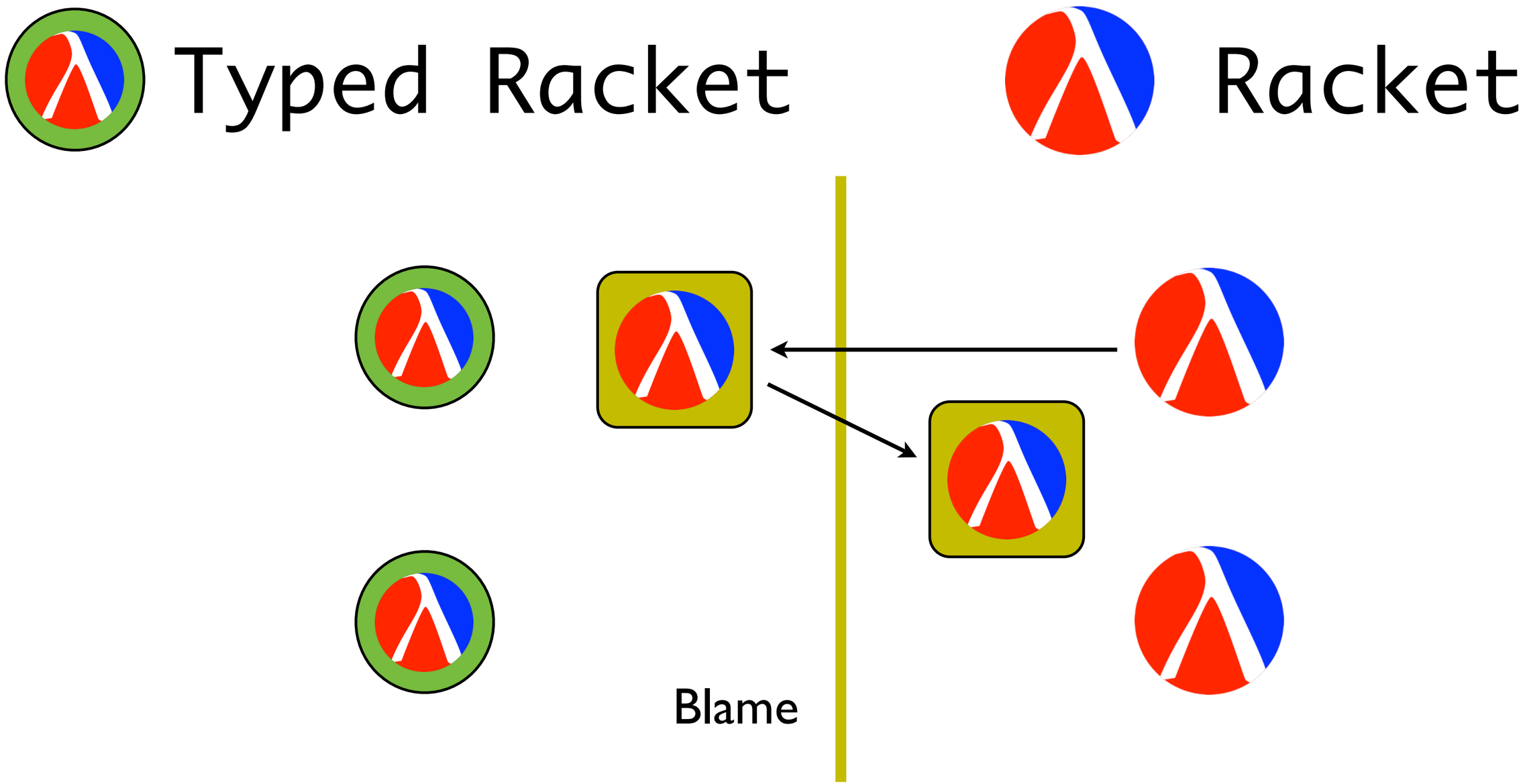
Typed Racket

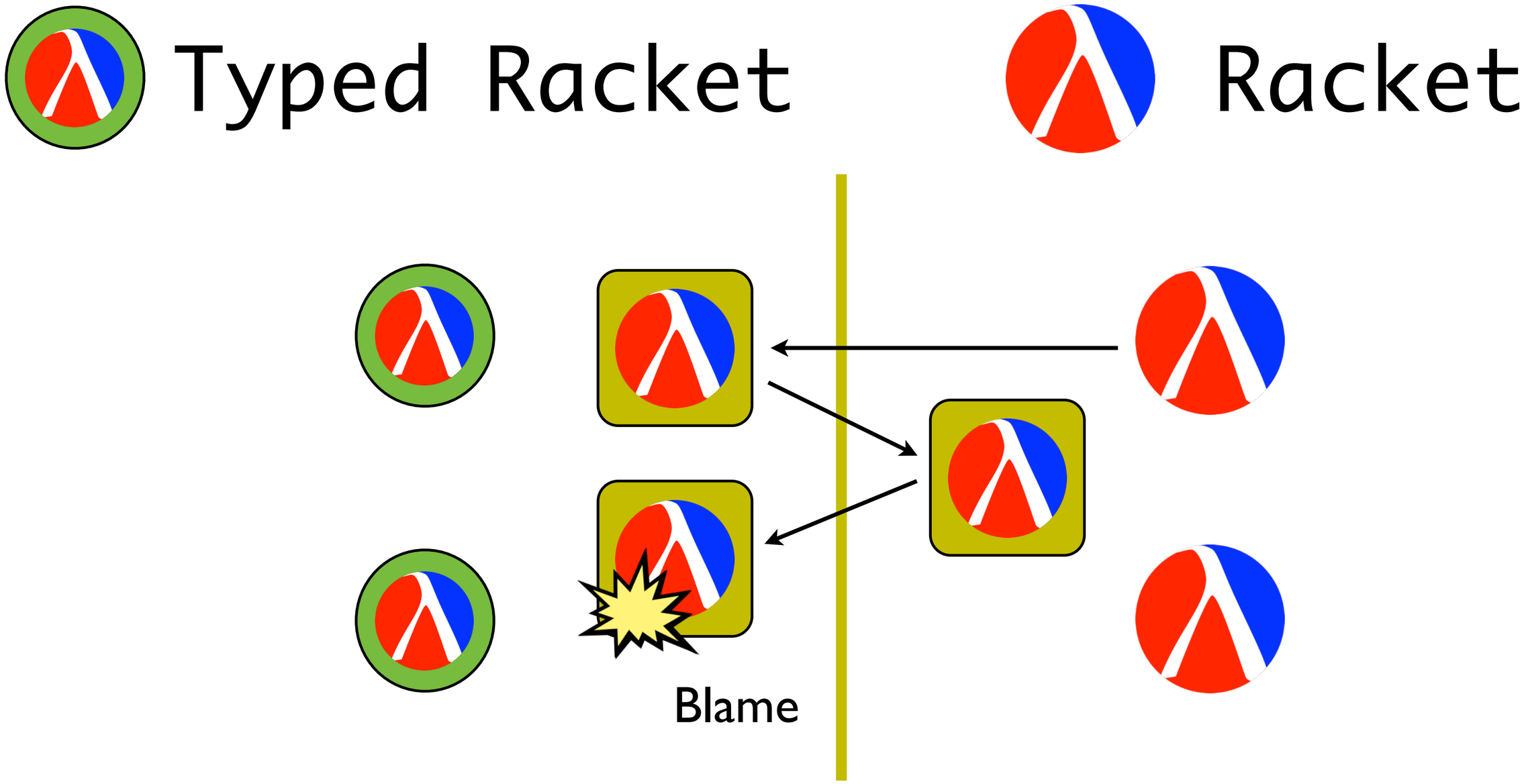
Racket

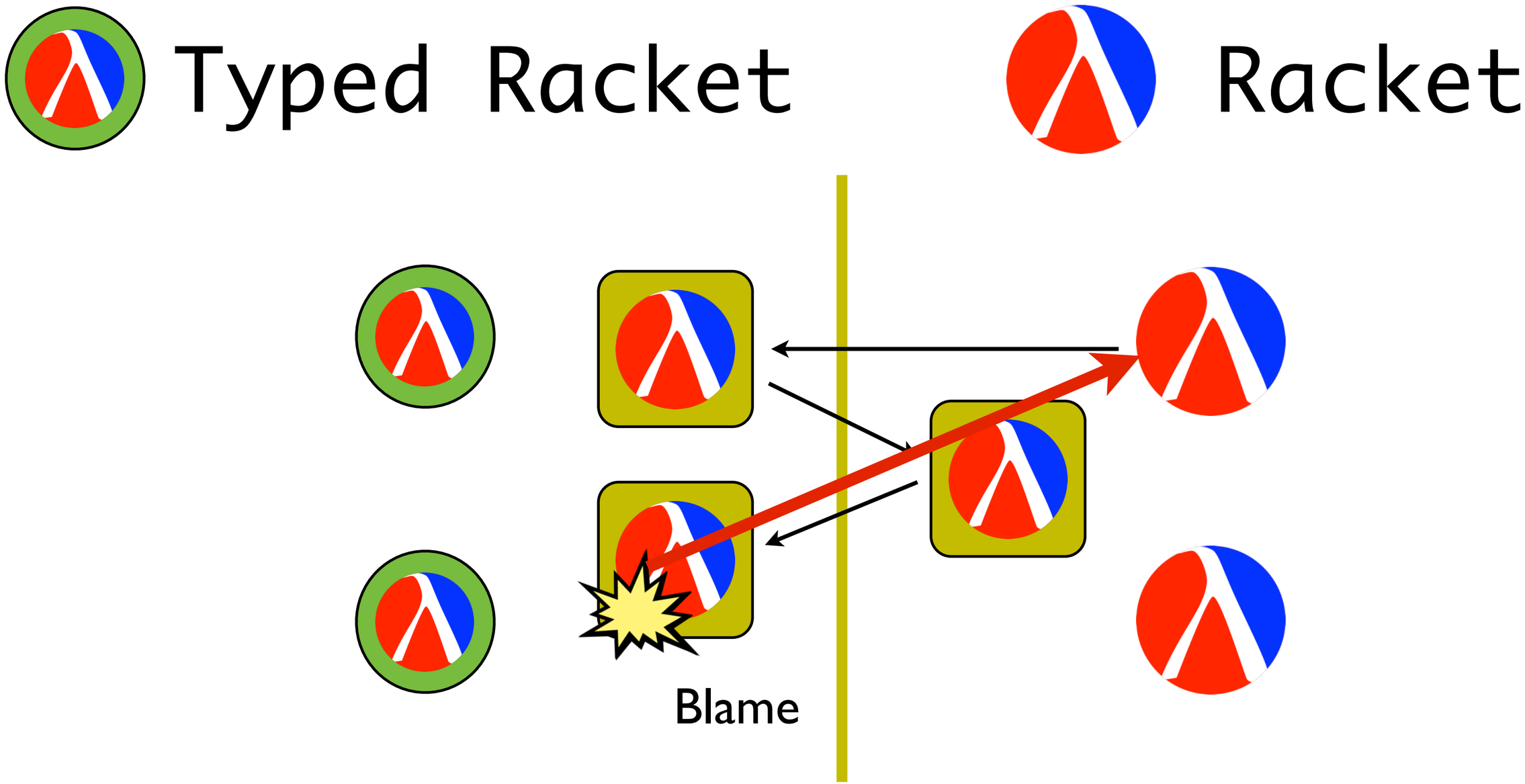


Safe optimisations


Typed Racket
Racket
Blame

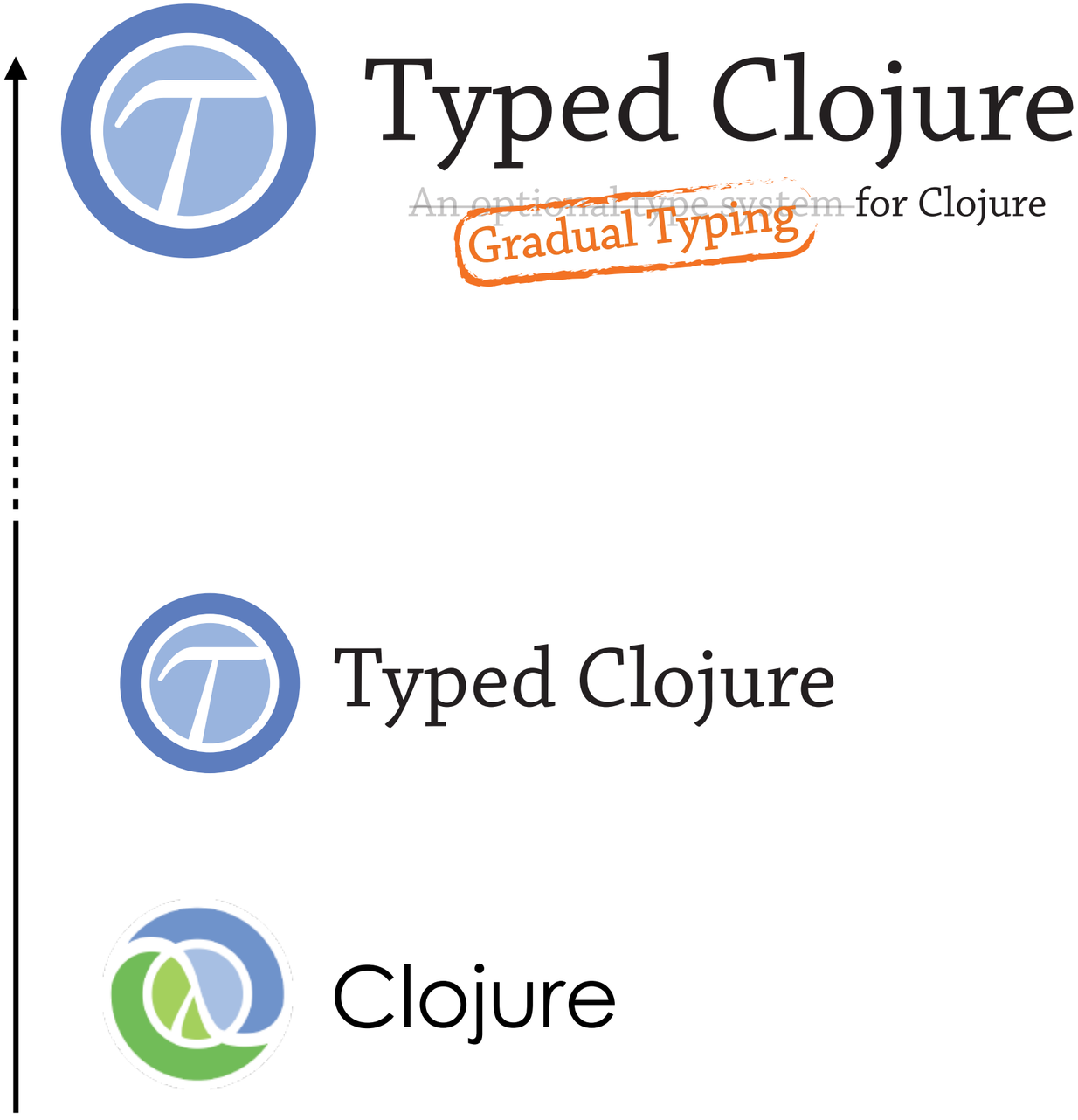

Typed Racket
Racket
Blame


Typed Racket
Racket
Blame


Typed Racket
Racket
Blame

What's done in Typed Clojure?

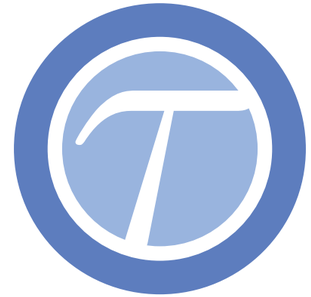
STL 2014


Typed Clojure
~~An optional type system~~ Gradual Typing for Clojure
Typed Clojure
Clojure

Automatic type hints →

Typed REPL →

STL 2014 →



Typed Clojure

~~An optional type system~~ Gradual Typing for Clojure

Typed Clojure

Clojure

Typed REPL

```
(ns ^:core.typed my-ns)
...
my-ns=> (inc 1)
:- Long
2
```

Typed REPL

```
(ns ^:core.typed my-ns)
...
my-ns=> (inc 1)
:- Long
2
my-ns=> (inc nil)
Expected Number, found nil
in: (clojure.lang.Numbers/inc nil)

Found 1 error
my-ns=>
```

Enabling Typed REPL

```
project.clj:

...
:repl-options
  {:nrepl-middleware
   [clojure.core.typed.repl/wrap-clj-repl]}
```

require+check w/ typed REPL

```
(ns
  my-inc-fail)

(inc nil)

; (require ‘my-inc-fail)
; NullPointerException
```

require+check w/ typed REPL

```
(ns
  my-inc-fail)

(inc nil)

; (require ‘my-inc-fail)
; NullPointerException
```

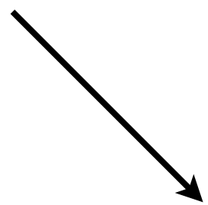
```
(ns ^:core.typed
  my-inc-fail)

(inc nil)

; (require ‘my-inc-fail)
; Type Error:
;  Expected Num, given nil
```

Cache for free

```
;; Cached
(require ‘my-ns)
```

Cache for free

```
;; Cached
(require ‘my-ns)
```



```
(load ‘‘my_ns’’)
```

Cache for free

```
;; Cached
(require ‘my-ns)
```

```
(load ‘‘my_ns’’)
```

```
Check via Typed REPL
;; Transitive deps cached
```

Automatic type hints

```
(ns ^:core.typed my-ns)

(defn get-parent [a :- Any]
  {:pre [(instance? java.io.File a)]}
  (.getParent a))
```

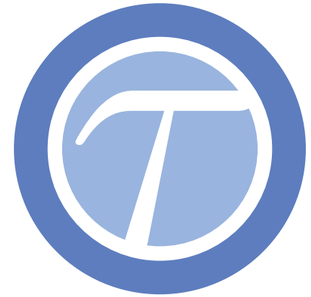
Non-reflective via static types

Next steps

Automatic type hints →

Typed REPL →

STL 2014 →



Typed Clojure

~~An optional type system~~ Gradual Typing for Clojure

Typed Clojure

Clojure

Check typed exports →

Check untyped imports →

Better proxy story →

Automatic type hints →

Typed REPL →

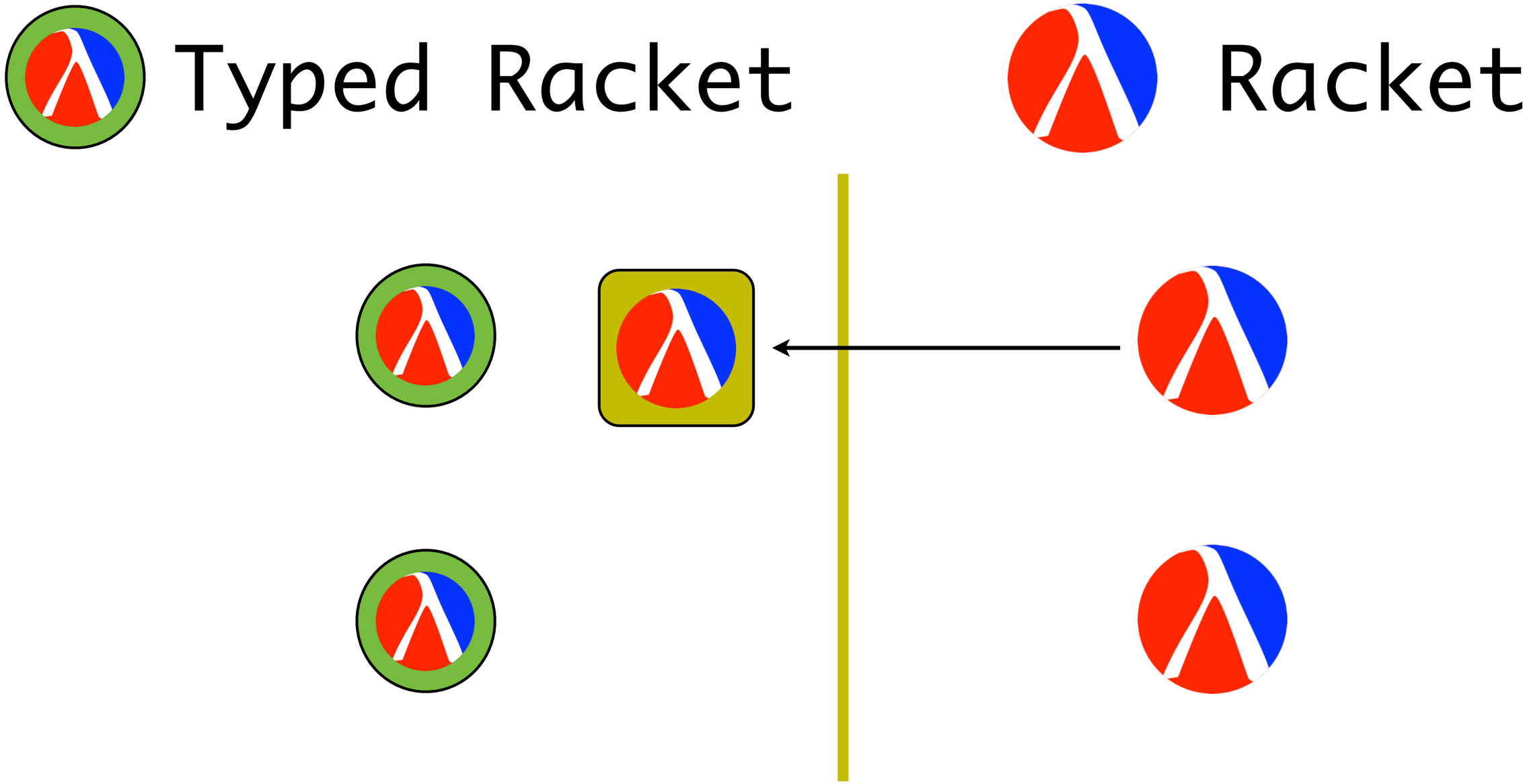
STL 2014 →

Typed Clojure

~~An optional type system~~ Gradual Typing for Clojure

Typed Clojure

Clojure


Typed Racket
Racket

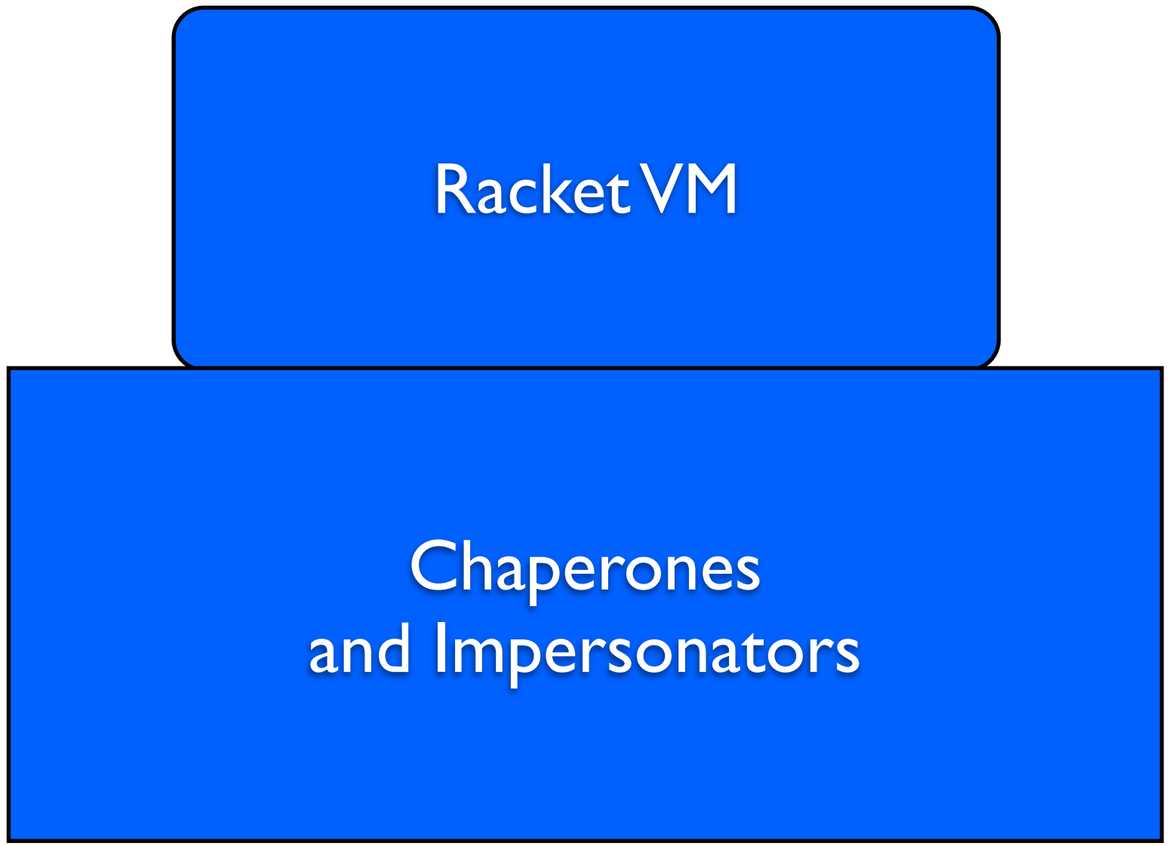
Proxy problem

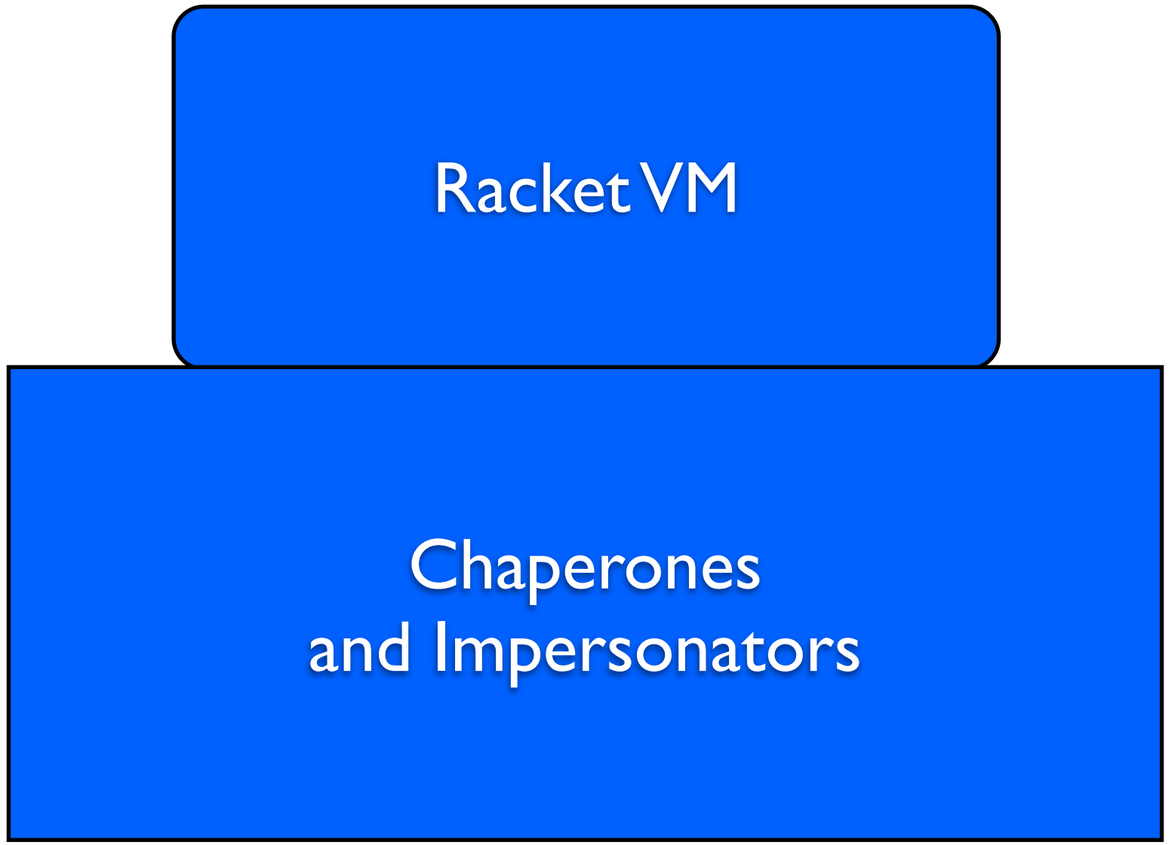
```
(deftype A [])

(proxy [A] [])
; CompilerException java.lang.VerifyError:
;  Cannot inherit from final class
```

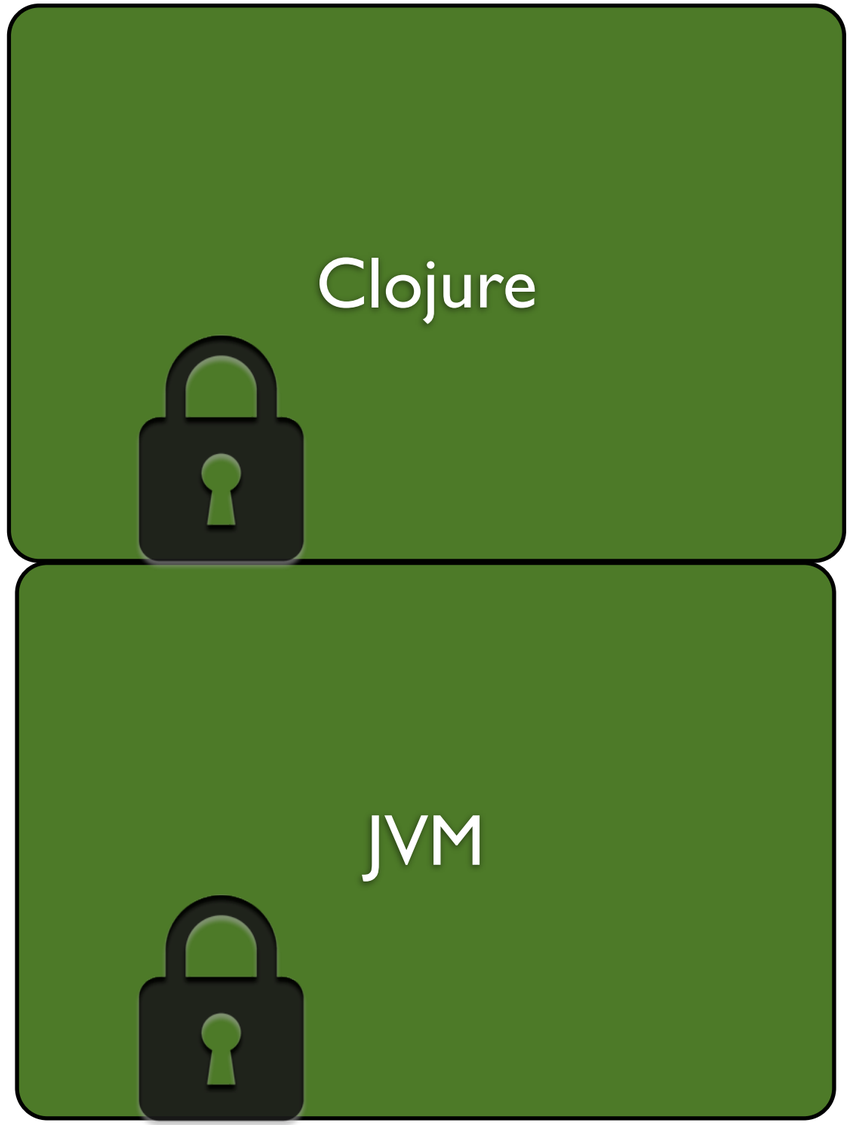
How to intercept methods?

```
(defprotocol IPoint
  (get-x [this])
  (get-y [this]))

(deftype Point [x y]
  IPoint
  (get-x [this] x)
  (get-y [this] y))
```

```
(proxy [Point] []
  (get-x [this]
    {:post [(integer? %)]}
    (get-x this))
  (get-y [this]
    {:post [(integer? %)]}
    (get-y this)))

; CompilerException java.lang.VerifyError:
;  Cannot inherit from final class
```


Racket VM
Chaperones
and Impersonators


Racket VM
Chaperones
and Impersonators


Clojure
JVM

Check untyped imports

```
(ns ^:core.typed my-ns
  (:require [my-untyped :as u]
            [clojure.core.typed :as t]))

(t/import-untyped u/uinc [Int -> Int])

(u/uinc 41)
```

```
(ns my-untyped)

(defn uinc [n]
  ‘‘hello’’)
```

Check untyped imports

```
(ns ^:core.typed my-ns
  (:require [my-untyped :as u]
            [clojure.core.typed :as t]))

(t/import-untyped u/uinc [Int -> Int])

(u/uinc 41)
```

```
(ns my-untyped)

(defn uinc [n]
  ‘‘hello’’)
```

Check untyped imports

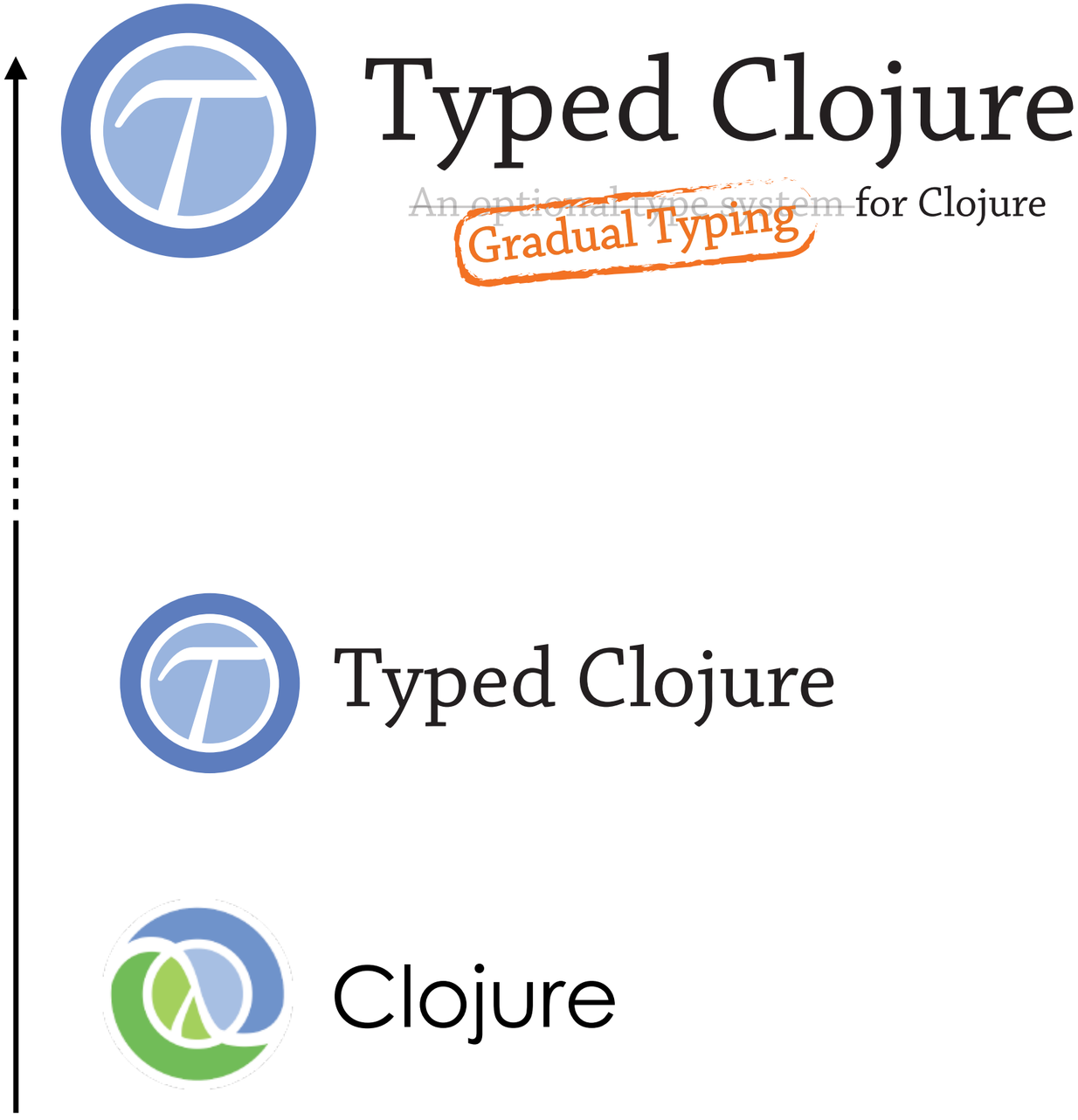
```
(ns ^:core.typed my-ns
  (:require [my-untyped :as u]
            [clojure.core.typed :as t]))

(t/import-untyped u/uinc [Int -> Int])
```

Blame

```
(u/uinc 41)
```

```
(ns my-untyped)

(defn uinc [n]
  ‘‘hello’’)
```

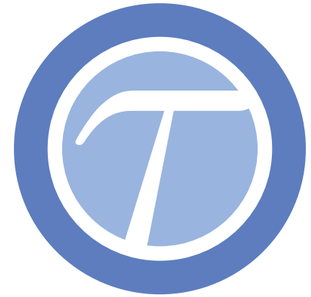
STL 2014


Typed Clojure
~~An optional type system~~ Gradual Typing for Clojure
Typed Clojure
Clojure

Automatic type hints →

Typed REPL →

STL 2014 →



Typed Clojure

~~An optional type system~~ Gradual Typing for Clojure

Typed Clojure

Clojure

Check typed exports →

Check untyped imports →

Better proxy story →

Automatic type hints →

Typed REPL →

STL 2014 →

Typed Clojure

~~An optional type system~~ Gradual Typing for Clojure

Typed Clojure

Clojure

Gradual Typing for Clojure

Check typed exports

Check untyped imports

Better proxy story

Automatic type hints

Typed REPL

STL 2014

Typed Clojure

~~An optional type system~~ Gradual Typing for Clojure

Typed Clojure

Clojure

Gradual Typing for Clojure

Check typed exports

Check untyped imports

Better proxy story

Automatic type hints

Typed REPL

STL 2014

Thanks!

Typed Clojure

~~An optional type system~~ Gradual Typing for Clojure

Typed Clojure

Clojure