Gradually Typed Clojure

Ambrose Bonnaire-Sergeant
@ambrosebs

Language choice

Language choice

- Static typing vs. dynamic typing

Language choice

- Static typing vs. dynamic typing
- Gradual typing: mix typed and untyped code in the same program

What is Gradual Typing?

- untyped + annotations = typed
- interaction between typed and untyped code

Examples

- Typescript (Javascript)
- Typed Racket (Racket)

Clojure

- Dynamically typed
- Lisp

core.typed

- Gradual typing for Clojure
- Static checking only

core.typed Design Goals

Better Errors

Better Errors

- Earlier

Better Errors

- Earlier
- More descriptive

Type System

- Designed to be statically sound
- Typescript and Dart are statically unsound

Experimentation

Experimentation

- Interesting problems

Experimentation

- Interesting problems
- "What if null was a separate type?"

Experimentation

- Interesting problems
- “What if null was a separate type?”
- Multimethods and Protocols

Experimentation

- Interesting problems
- “What if null was a separate type?”
- Multimethods and Protocols
- Array covariance

Annotations

Annotations

- All checked vars must be annotated

Annotations

- All checked vars must be annotated
- Also function parameters

Annotations

- All checked vars must be annotated
- Also function parameters
- Some special forms have “typed” equivalents

```
(ann my-fn (Fn [Number -> Number]
               [Symbol -> Symbol]
               [Boolean -> Keyword]))
(defn my-fn [a]
  ...)
```

Changing existing code

- Goal: understand Clojure idioms
- It's ok to require annotations, but not to force code restructuring

Optional Checking

- Type checking is provided separately to compilation
- Code that fails type checking will still compile and run as usual
- Up to the programmer to use when needed

Occurrence Typing

- Dynamically typed idioms
- Better types as the program progresses
- Recognise conditionals and assertions

Occurrence Typing

```
(cond
  (symbol? a) ...
  (number? a) ...
  :else ...)
```

Occurrence Typing

```
(let [a ...
      _ (assert (symbol? a))]
  ...)
```

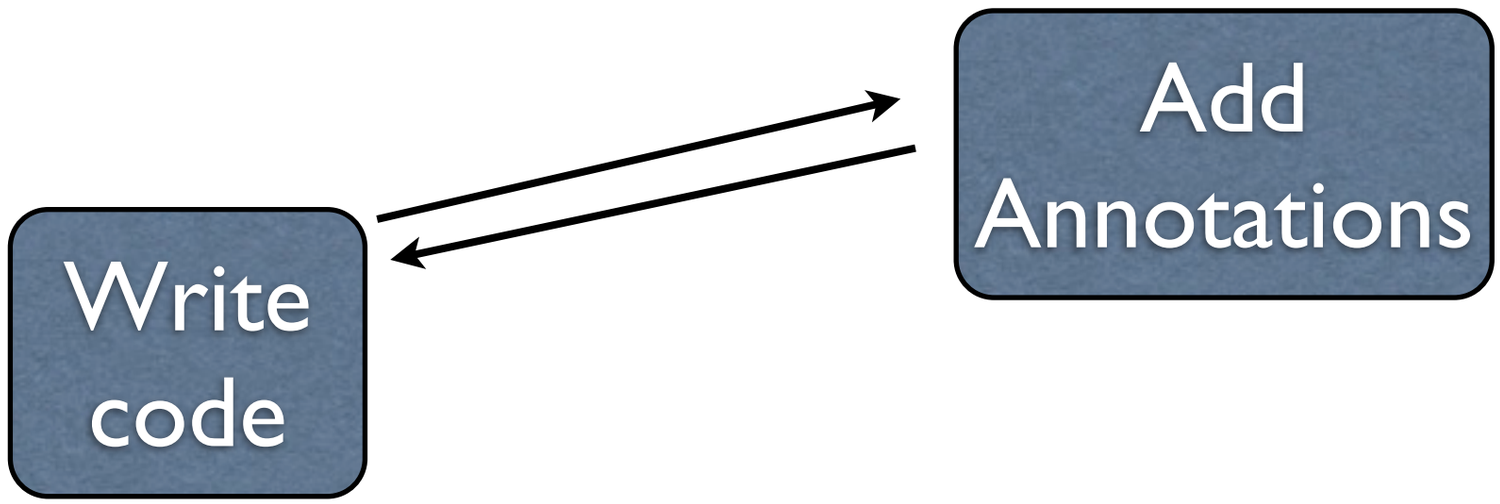
Variable-Arity Polymorphism

```
(map + a)

(map + a b c d e f)
```

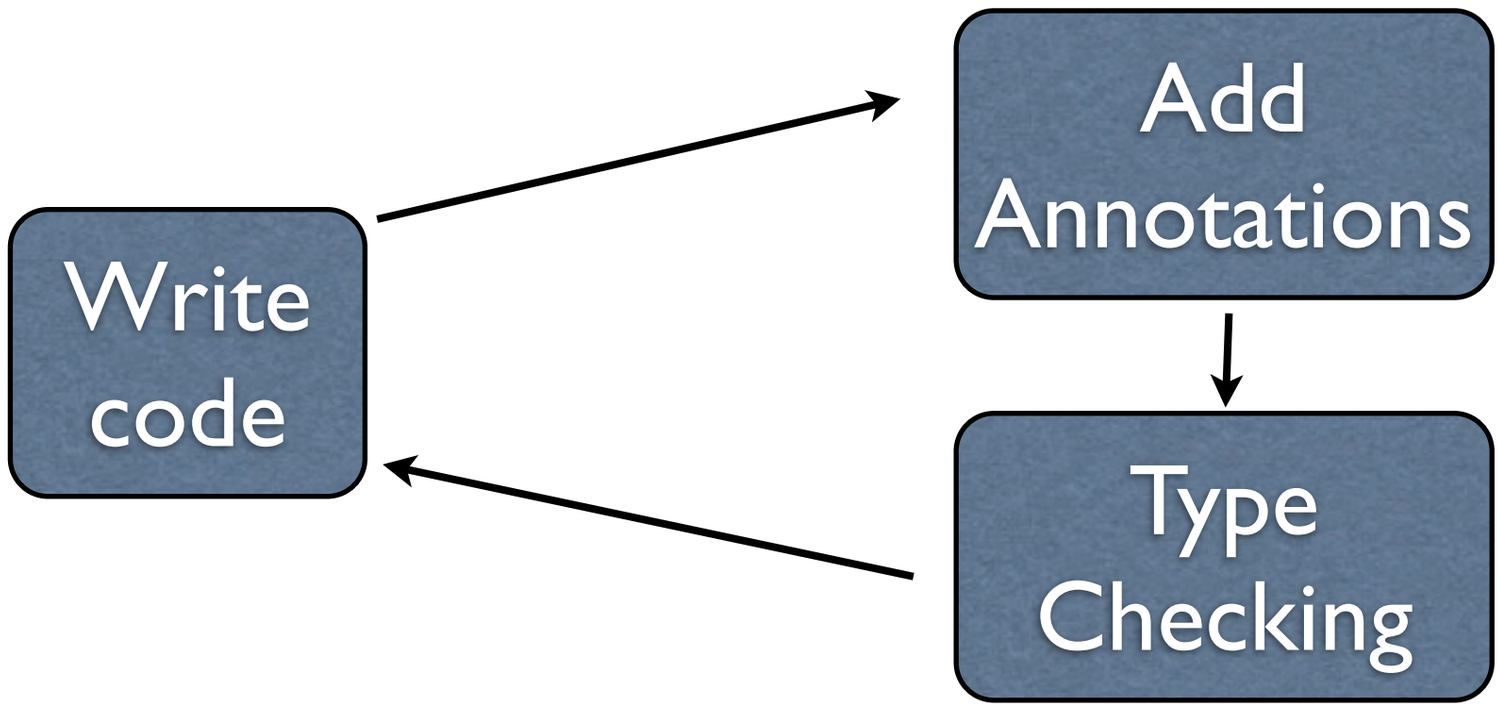
Java Interop

```
(ann f File)
(def f (File. "a"))

(ann prt (U nil String))
(def prt (.getParent ^File f))

(non-nil-return java.io.File/getName :all)
(ann nme String)
(def nme (.getName ^File f))
```

Workflows



Workflows



Macros

- Type checking operates on fully macroexpanded expressions
- Macro definitions are not checked

Future Work

Performance

- Can core.typed improve performance?
- Needs further investigation
- Wait for CinC

Clojurescript

- CLJS has some nasty runtime type errors

Is core.typed Production Ready?

- Almost
- Lazy-loading
- Documentation

Questions?