Automatic Type Annotations

Ambrose Bonnaire-Sergeant

Ph.D. Qualifying exam

A Story about Annotating…

Lucidchart

JavaScript (Google Closure annotated) → **?** → **TypeScript**

"How do we convert 600k lines of JavaScript to TypeScript, in an actively developed app?"

Option 1

- "Gradual" typing

Option 2

- Stop and sprint!

Chose: Option 2

- Annual 48 hour hackathon
- No devs working on core product for 48 hours!

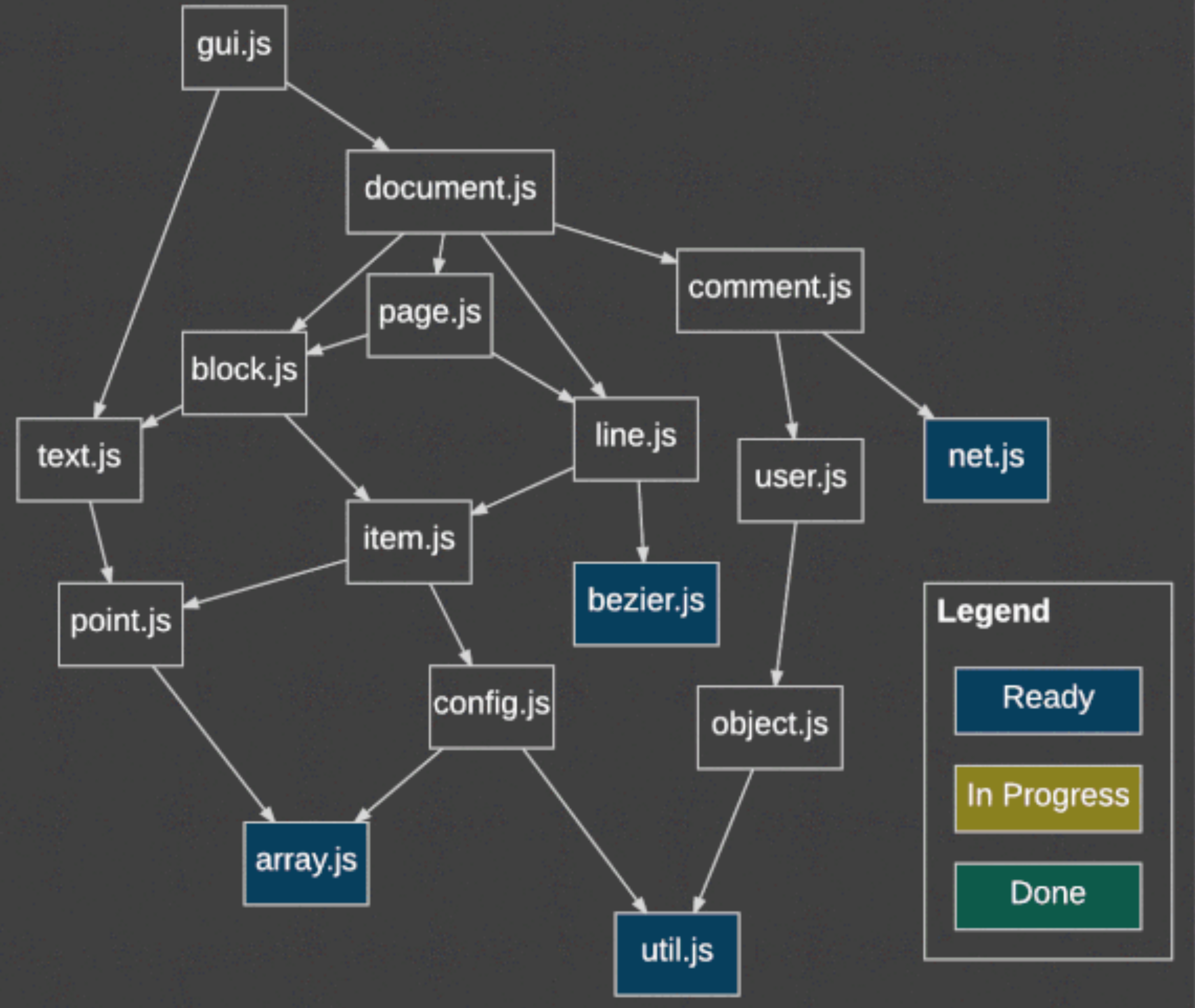
CTO's thoughts:

June 9th

Ben Dilts 8:53 AM

I think there's a zero percent chance you can get all our Closure code successfully building as Typescript in a matter of three days.

Six of us engineers decided to try anyway.



	A	
1	0/16	Assi
2	Can do	
3	Can do	
4	Can't do	
5	Can't do	
6	Can't do	
7	Can't do	
8	Can't do	
9	Can't do	
10	Can't do	
11	Can do	
12	Can't do	
13	Can't do	
14	Can't do	
15	Can't do	
16	Can't do	
17	Can do	

Clutz - Closure to TypeScript Declarations (`.d.ts`) generator.

build passing

This project uses the Closure Compiler to parse Closure-style JSDoc type annotations from ES5/ES2015 code, and generates a suitable TypeScript type definition file (`.d.ts`) for the exported API.

Gents - Closure to TypeScript converter

This repository also hosts `gents` - tool that generates TypeScript code out of Closure annotated `.js` . We host it in this repo together with `clutz` because they both wrap Closure Compiler to get the type information. As such `gents` shares `clutz` restriction that it only accepts code that is valid well-typed Closure JavaScript.


CONVERTING 600K LINES TO TYPESCRIPT IN 72 HOURS

November 16, 2017 Paul Draper & Ryan Stringham 15 Comments

Takeaways

Companies will <u>heavily invest</u> in transitioning to typed languages

Translation to typed languages can be <u>partially automated</u>

Background for My Research

Clojure
(untyped)

Manually annotate

Typed Clojure
(typed sister-language)

clojure.spec
(contract system)

My research objective

Create *effective tools* to ease the transition to annotated target languages.

Clojure

Tool-assisted

Typed Clojure

clojure.spec

Approach

1. Understand target language theory
2. Understand target language practice
3. Compare our tool with similar tools

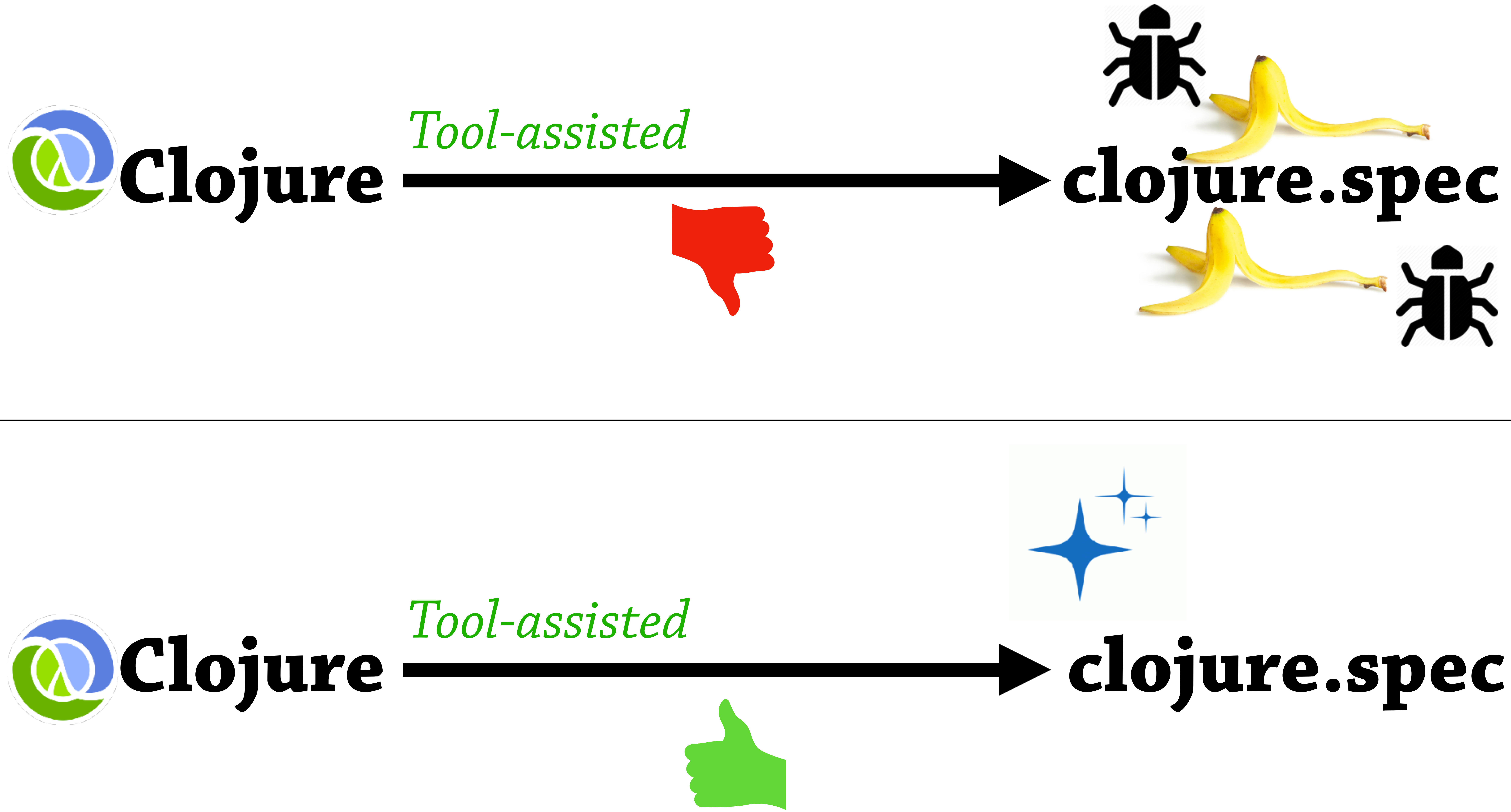
This Talk

1. Understand target language theory
   - **Quals question**: *spec theory*
   - Audience questions

2. Understand target language practice
   - **Quals question**: *spec practice*
   - Audience questions

3. Compare with similar tools
   - **Quals question**: *perf analysis*
   - Audience questions

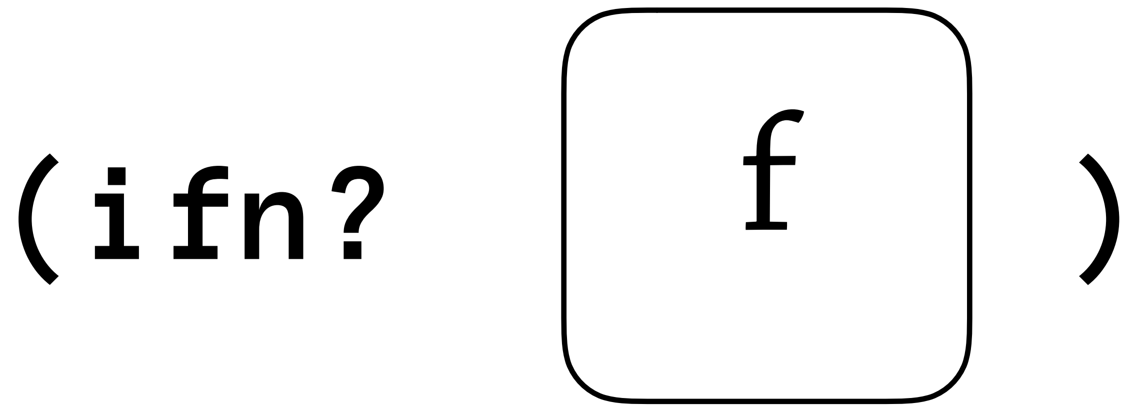
1. Understand target language theory


Clojure
Tool-assisted
clojure.spec
Clojure
Tool-assisted
clojure.spec

Quals Question

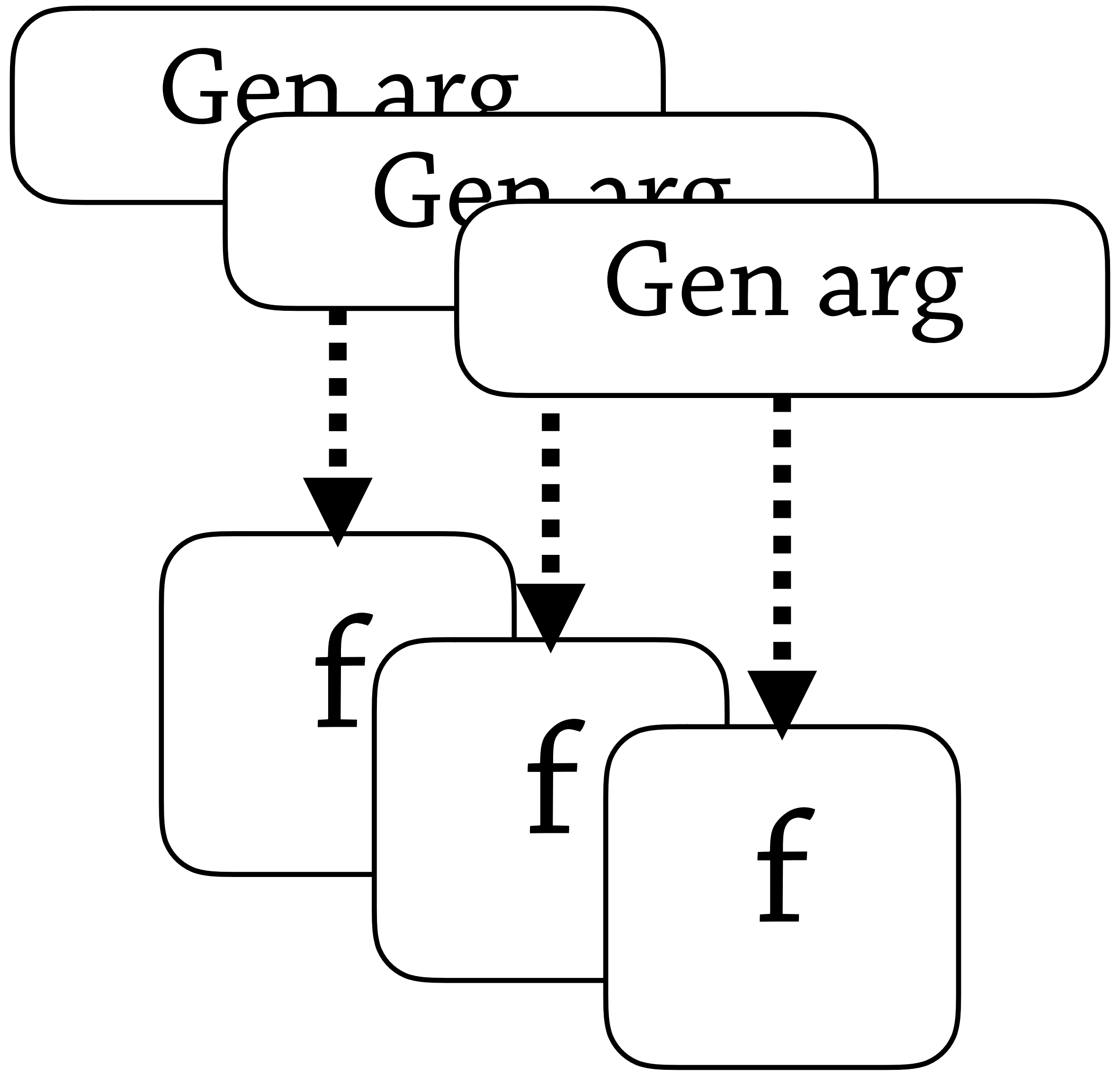
(spec theory)

1. Formulate a formal model for clojure.spec
2. Implement model in PLT Redex
3. Formulate consistency property between contracted and uncontracted execution
   (a) Test property in Redex

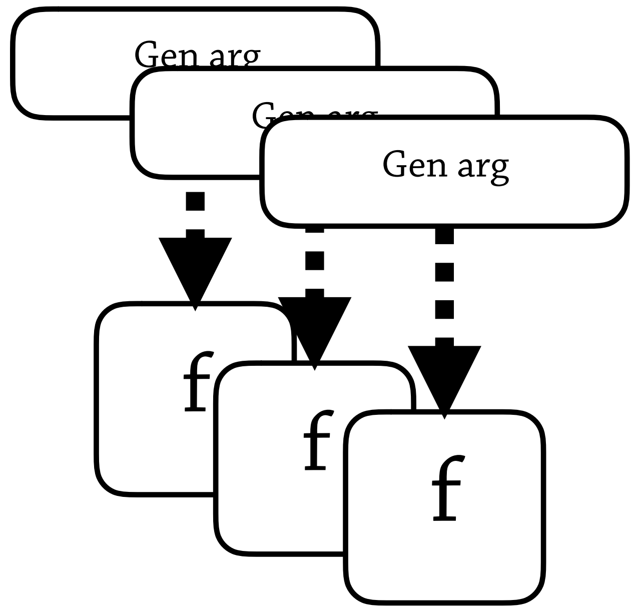


Tag-test

Proxy-based



Generative-testing based

Model	***Features***
λc	
λc_s	f
λc_s^f	f 

E ::= C | L | X
| (E E ...)
| (if E E E)

C ::= N | O | B | nil | H | ERR

X ::= variable-not-otherwise-mentioned

ERR ::= (error any any ...)

L ::= (fn [X ...] E) | (fn X [X ...] E)

NONFNV ::= B | H | nil | N

V ::= O | L | NONFNV

V^e ::= V | ERR

H ::= (HashMap (V V) ...)

B ::= true | false

N ::= number

$\mathbb{Z}$ ::= natural

O ::= P
| inc | dec
| + | * | dissoc
| assoc | get

P ::= zero? | number? | boolean? | nil?

$\mathbb{C}$ ::= [] | (if $\mathbb{C}$ E E) | (V ... $\mathbb{C}$ E ...)

FS ::= (DefFSpec ($\mathbb{S}$ ...) $\mathbb{S}$)

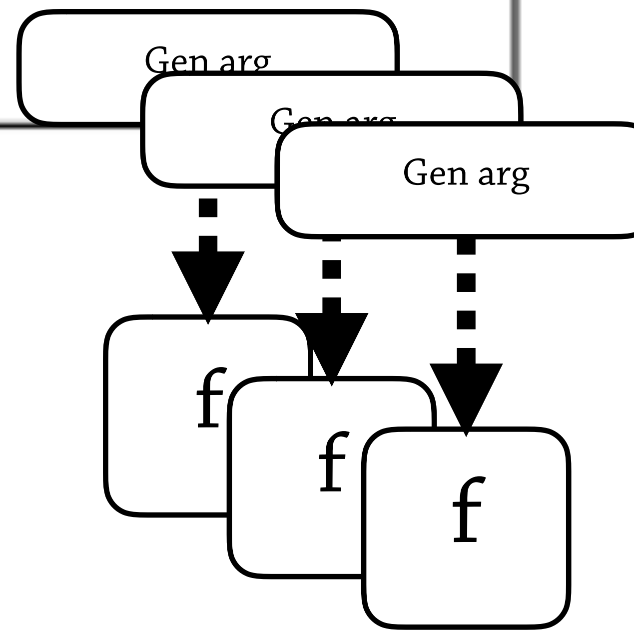
$\mathbb{S}$ ::= P

$\mathbb{C}$ ::= .... | (assert-spec $\mathbb{C}$ $\mathbb{S}$)

C ::= .... | (gen-spec $\mathbb{S}$)

E ::= .... | (assert-spec E $\mathbb{S}$)

$\mathbb{S}$ ::= .... | (FSpec ($\mathbb{S}$ ...) $\mathbb{S}$) | (FSpec ($\mathbb{S}$ ...) $\mathbb{S}$ $\mathbb{Z}$)



C[((fn (X ...) E) V ...)] ⟶ C[subst*[E, ([X ↦ V] ...)]] [β]
where unique[(X ...)], (same-length? (X ...) (V ...))

C[((fn X_{rec} [X ...] E) V ...)] ⟶ C[subst*[E, ([X_{rec} ↦ (fn X_{rec} [X ...] E)] [X ↦ V] ...)]] [rec-β]
where unique[(X_{rec} X ...)], (same-length? (X ...) (V ...))

C[(if V E_1 E_2)] ⟶ C[E_1] [if-t]
where (truthy? V)

C[(if V E_1 E_2)] ⟶ C[E_2] [if-f]
where (not (truthy? V))

C[(O V ...)] ⟶ C[V_1] [δ]
where δ[(O V ...), V_1]

C[ERR] ⟶ ERR [error]
where (not (top-level-hole? C))

C[X] ⟶ (error unknown-variable X) [x-error]

(NONFNV V ...) ⟶ (error bad-application) [β-non-function]

C[((fn [X ...] E) V ...)] ⟶ (error argument-mismatch (arg-mismatch-msg (X ...) (V ...) (fn [X ...] E))) [β-mismatch]
where unique[(X ...)], (not (same-length? (X ...) (V ...)))

C[(f V ...)] ⟶ (error argument-mismatch (arg-mismatch-msg (X ...) (V ...) (fn [X ...] E))) [r...
where (not (same-length? (X ...) (V ...))), f = (fn X_f [X ...] E)

C[(gen-spec S)] ⟶ C[gen-spec*[S]] [gen-spec]

C[(assert-spec (fn [X ...] E) (DefFSpec (S_a ...) S_r))] ⟶ C[(fn [X ...] (assert-spec ((fn [X ...] E) (assert-spec X S_a) ...) S_r))] [assert-deffspec]

C[(assert-spec (fn X_n [X ...] E) (DefFSpec (S_a ...) S_r))] ⟶ C[(fn X_n [X ...] (assert-spec ((fn [X ...] E) (assert-spec X S_a) ...) S_r))] [assert-rec-deffspec]

C[(assert-spec V P)] ⟶ C[(if (P V) V (error spec-error (spec-violation-msg P V)))] [assert-spec-P?]

C[(gen-spec S)] ⟶ C[gen-spec*-hof[S]] [gen-spec]

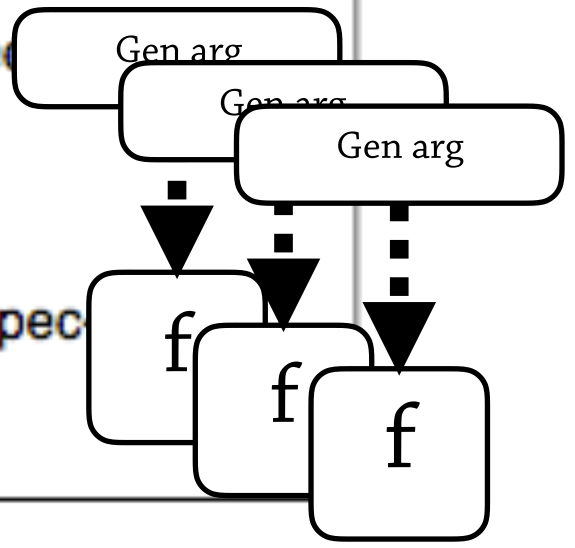
C[(assert-spec V (FSpec (S_a ...) S_r))] ⟶ C[(assert-spec V (FSpec (S_a ...) S_r ngenerations))] [assert-fspec-init]

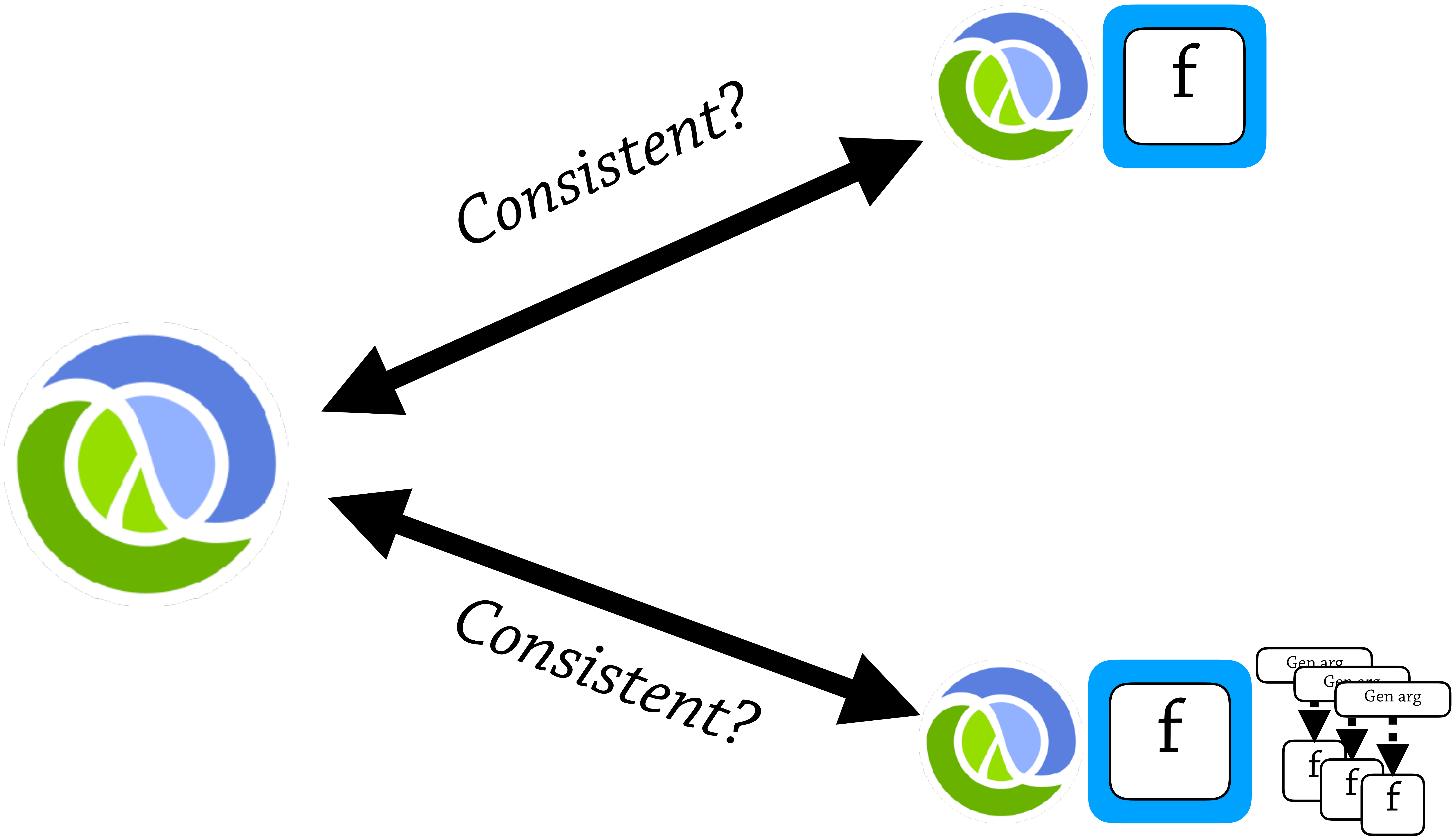
C[(assert-spec f (FSpec (S_a ...) S_r 0))] ⟶ C[f] [assert-fspec-stop]
where f = (fn [X ...] E)

C[(assert-spec f (FSpec (S_a ...) S_r Z))] ⟶ C[(do (assert-spec (f (gen-spec S_a) ...) S_r) (assert-spec f (FSpec (S_a ...) S_r (sub1 Z))))] [assert-fspec-gen]
where (< 0 Z), f = (fn [X ...] E)

C[(assert-spec f (FSpec (S_a ...) S_r Z))] ⟶ C[(do (assert-spec (f (gen-spec S_a) ...) S_r) (assert-spec f (FSpec (S_a ...) S_r (sub1 Z))))] [assert-re...
where (< 0 Z), f = (fn nme [X ...] E)

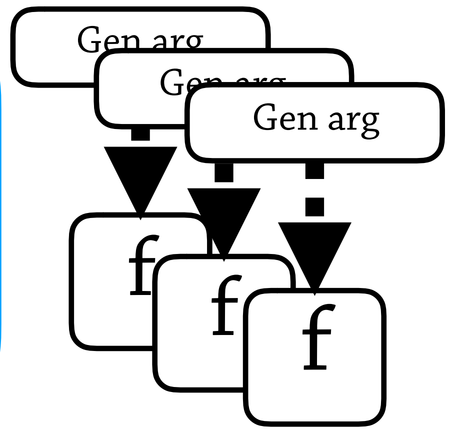
C[(assert-spec NONFNV (FSpec (S_a ...) S_r Z))] ⟶ (error spec-error (nonf-spec-error-msg NONFNV)) [assert-fspec...

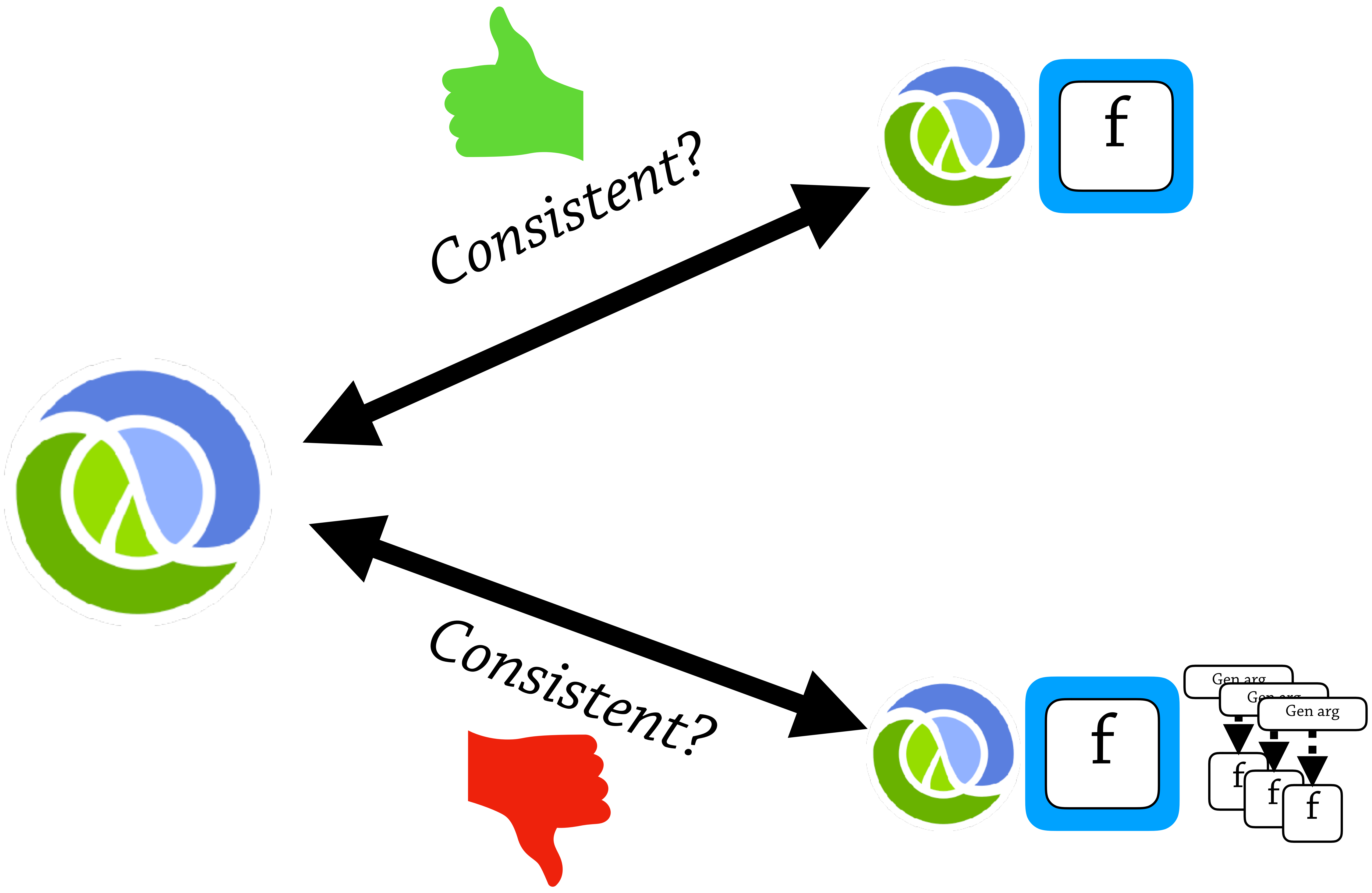




Consistent?
f
Consistent?
f
Gen arg
f
f
f

Uncontracted	*Contracted*	
`1`	`(assert-spec 1 int?)`	`(assert-spec 1 nil?)`
`1`	`1`	`Spec ERROR: expected nil, found 1`
	Consistent?	*Consistent?*

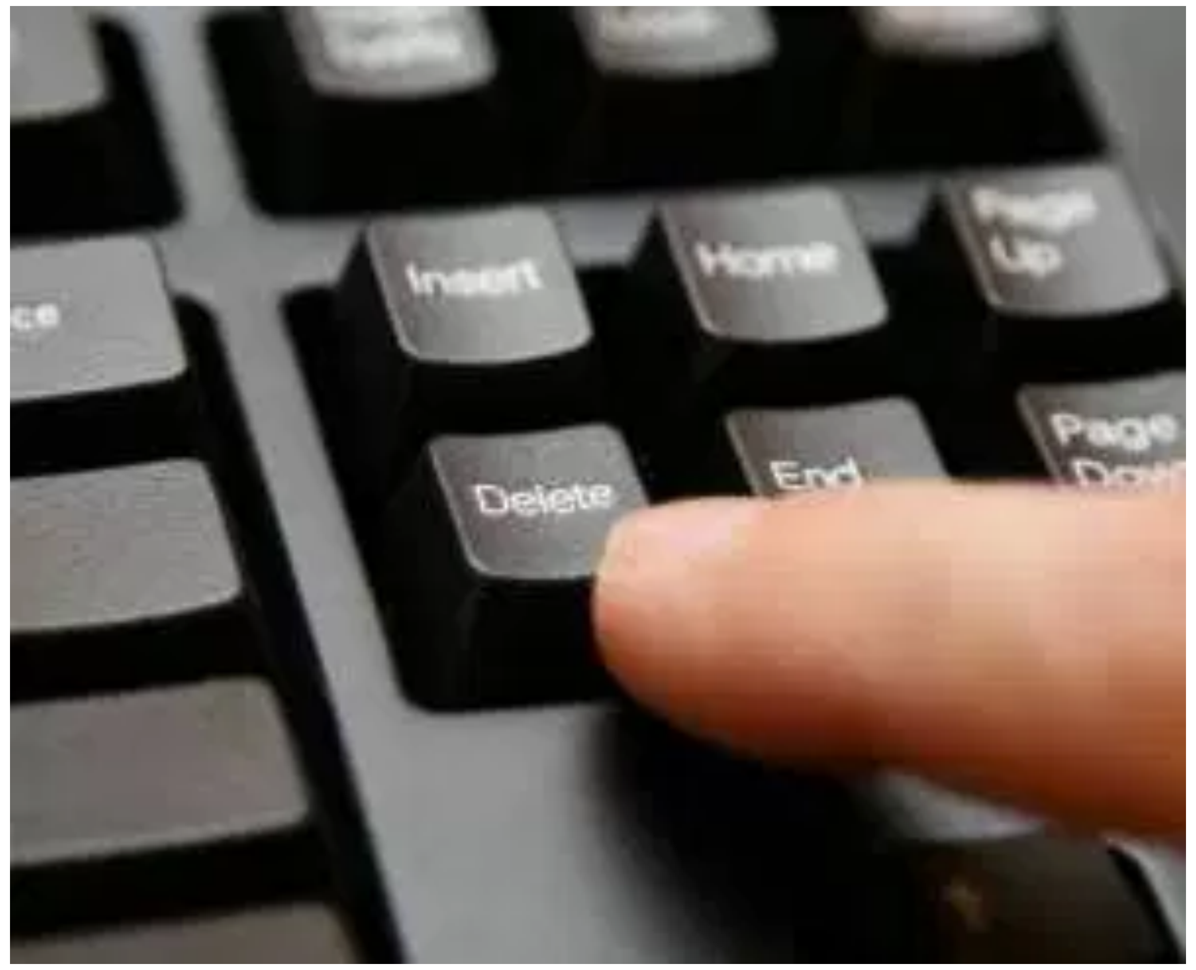
Uncontracted	*Contracted*
	
`(fn a [] (a))`	`(assert-spec (fn a [] (a)) (fspec :args (cat)))`
↓	↓
`(fn a [] (a))`	$\bot$
	✗ *Consistent?*


Consistent?
f
Consistent?
f
Gen arg
f

Break for Questions

2. Understand target language practice

Our tool's output

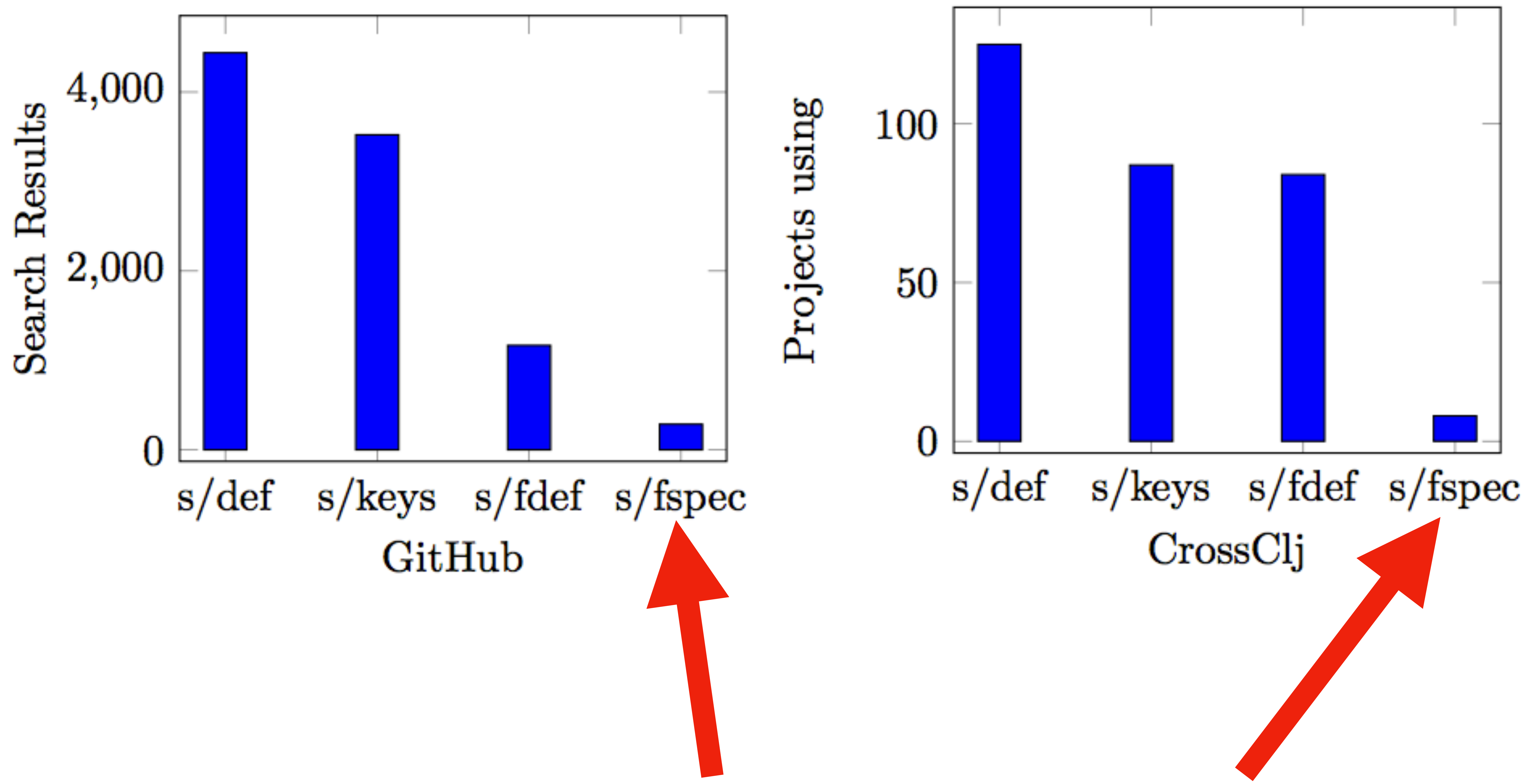


Updated code

Quals Question
(spec practice)

1. Examine clojure.spec usage in real-world code bases
2. Analyze frequency and precision of higher-order function annotations

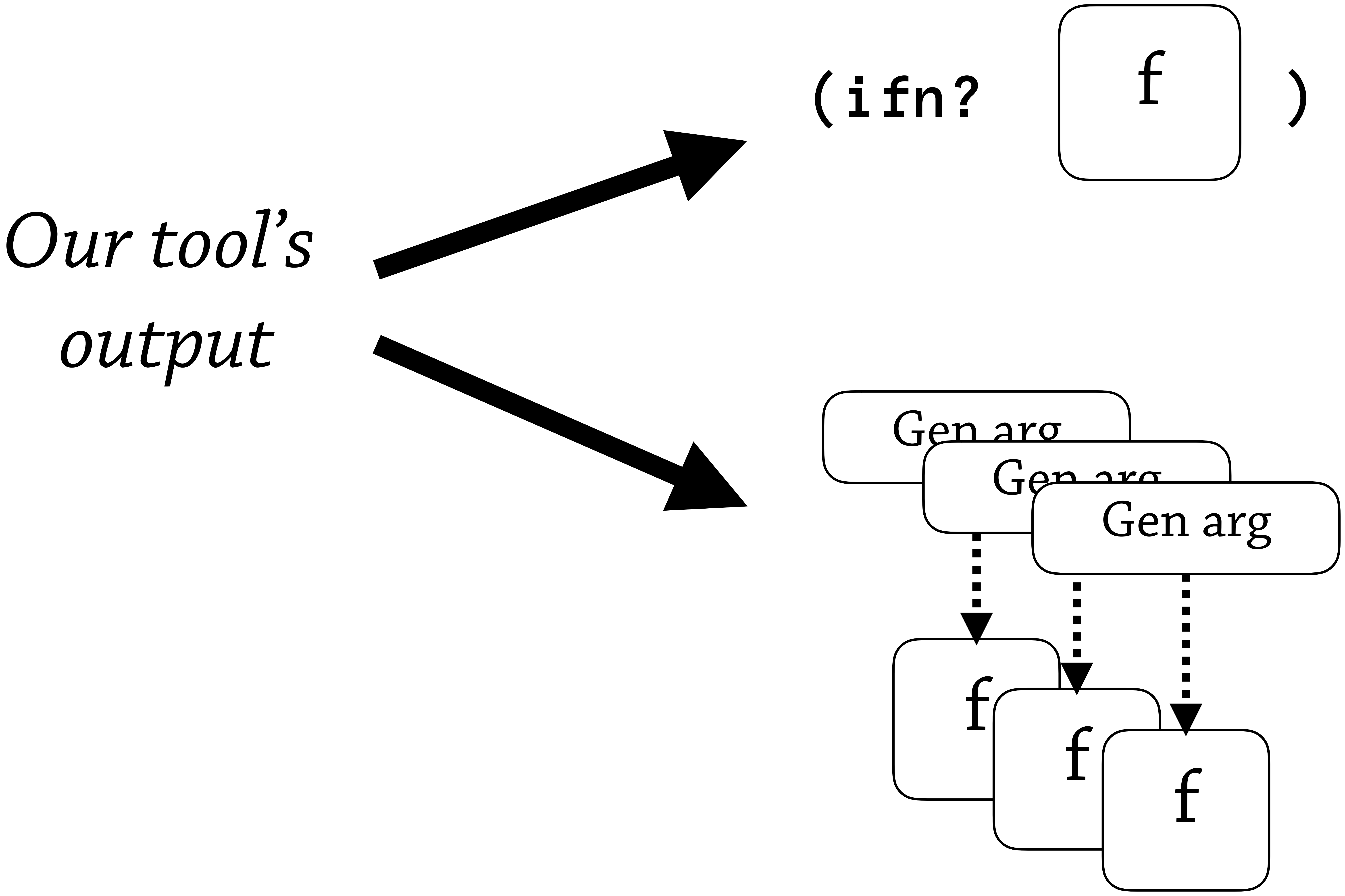
Searches say generative testing is not that popular



s/fspec = Gen testing-based function contract

Different function contracts rarely occur in the same project

	Search terms	# Projects	Ratio of <Tag-test>:<Gen-testing> function specs
Search 1	`clojure.spec && fspec`	18	3:79
Search 2	`clojure.spec && ifn?`	17	188:0


Our tool's output
(ifn? f)
Gen arg
Gen arg
Gen arg
f
f
f

Break for Questions

3. Compare with similar tools

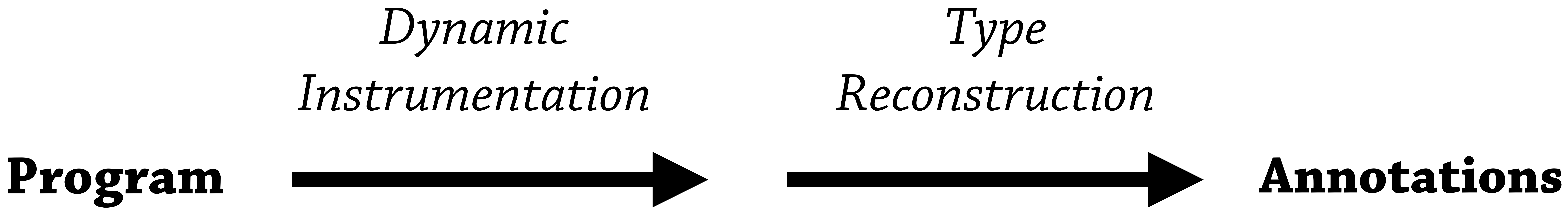
3. Compare with similar tools

Why is this useful?

- Ensure performance of our tool is reasonable compared to existing tooling
- Better understand tradeoffs we made by comparing with other approaches

Quals Question *(perf analysis)*

1. Compare time+space complexity vs. Daikon
2. Can we reuse Daikon's optimizations?
3. How expressive are Daikon annotations?



Program
Dynamic
Instrumentation
Type
Reconstruction
Annotations

Our Tool's Type Reconstruction

t=0	Initial		
t=1	Observed x is an int	x : Int ✓	
t=2	Observed y is a bool	x : Int	y : Bool ✓
t=3	Observed x is a bool	x : Int U Bool ✓	y : Bool
t=4	Observed x is a symbol	x : Any ✓	y : Bool

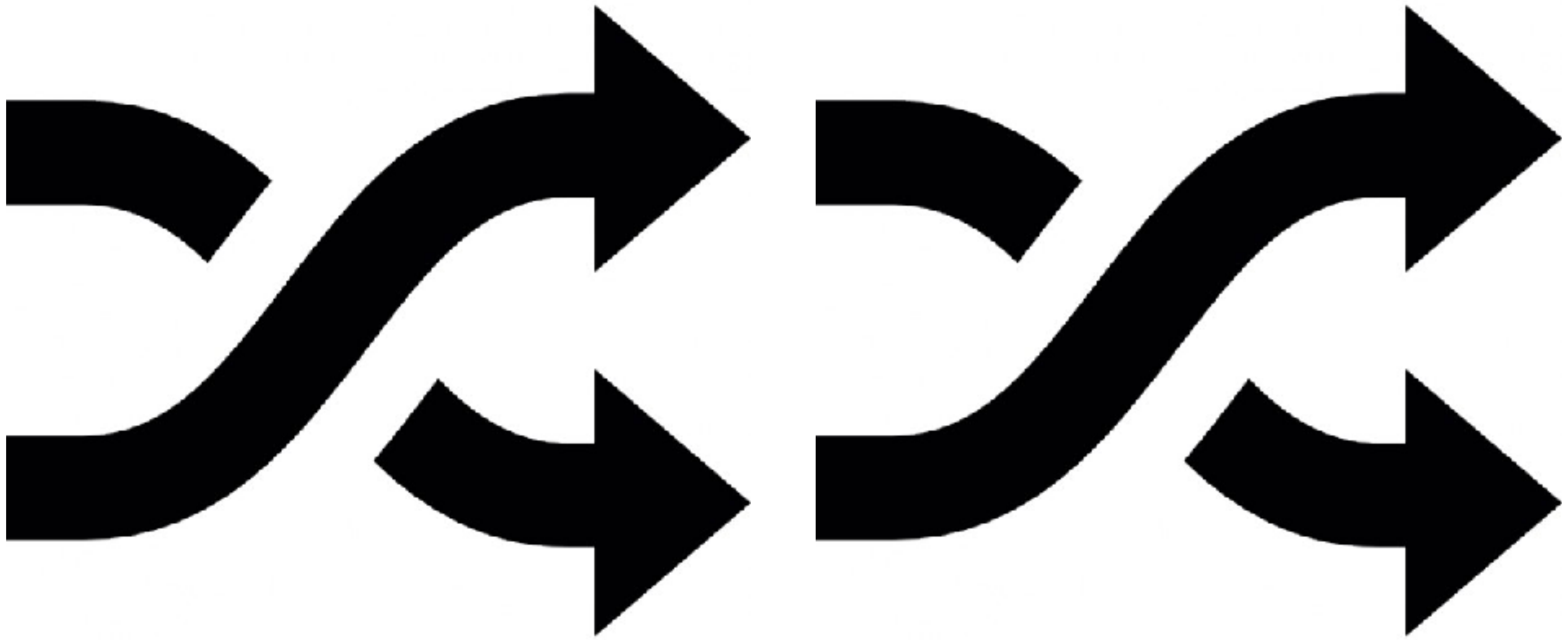
Daikon's Type Reconstruction

t=0	**Initial**	even(x) odd(x)	even(y) odd(y)	even(z) odd(z)
t=1	**Observed** $x = 3$	~~even(x)~~ odd(x)	even(y) odd(y)	even(z) odd(z)
t=2	**Observed** $y = 4$	~~even(x)~~ odd(x)	even(y) ~~odd(y)~~	even(z) odd(z)

Processing traces on-line

Dynamic Instrumentation

Type Reconstruction



Break for Questions

Recap

I want to create *effective tools* to ease the transition to annotated target languages.

Approach:

1. Understand target language theory
2. Understand target language practice
3. Compare our tool with similar tools

Thanks

“Intertwined arrows” designed by Freepik