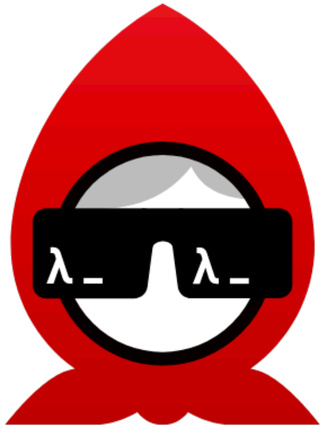


babashka

A native Clojure interpreter for scripting

The 2021 Graal Workshop

Michiel Borkent
@borkdude
2021-02-27

GraalVM + Clojure:

[clj-kondo)

a linter that sparks joy

Star 1.1k

(lsp)

Static analyzer and linter for Clojure

```
$ clj-kondo --lint - <<< '(inc 1 2)'
<stdin>:1:1: error: clojure.core/inc is called with 2 args but expects 1
linting took 12ms, errors: 1, warnings: 0
$
```

```
(let [x 1]
  (+ x y))
        Unresolved symbol: y
```

```
(inc (str :foo))
     Expected: number, received: string.
```

Star 1.8k

ba·bash·ka

```
$ time bb '(+ 1 2 3)'
6
0.00s  user 0.00s system 67% cpu 0.013 total
```

- Native Clojure scripting tool, single binary, no JVM (GraalVM compiled), **fast startup**
- Alternative to byte code compilation by Clojure compiler
- **Prevents context switch** to bash for Clojure devs writing build scripts
- **Batteries included** (arg parsing, JSON, http client/server, ...)
- Supports **multi-threading**
- **Source compatibility** with **JVM** Clojure + **GraalVM** = sane upgrade path = **low risk** adoption

The 5 second rule

- **Startup time vs performance**
  - Sweet spot: short running scripts (< 5 seconds): use babashka (Clojure **interpreter**)
  - Long running performance intensive processes: use **JVM** Clojure **compiler**
    - Compile with GraalVM native-image for fast startup

Namespaces
Built in libs

Shelling out

Java interop

```
#!/usr/bin/env bb

(ns example
  (:require [clojure.data.csv :as csv]
            [clojure.java.io :as io]
            [clojure.java.shell :refer [sh]]
            [clojure.pprint :as pp]
            [clojure.string :as str]))

(def db "/tmp/analysis.db")

(defn query [& args]
  (apply sh "sqlite3" "-quote" "-header" db args))

(defn create-db! []
  (when (not (.exists (io/file db)))
    (query "create table animals (
              name text, scientific_name text )")))

(defn text-val [s]
  (format "\"%s\"" s))

(defn int-val [i]
  (format "%s" i))

(defn make-row [& xs]
  (format "(%s)" (str/join ", " xs)))

(defn animal-row [{:keys [:name :scientific-name]}]
  (make-row (text-val name) (text-val scientific-name)))

(defn insert-rows! [animals]
  (let [rows (str/join "," (mapv animal-row animals))
        q (str/join " " ["insert into animals (name, scientific_name) values"
                         rows])
        {:keys [:exit :err]}
        (query q)]
    (when (not (zero? exit))
      (println "Error inserting var!" err)
      (System/exit exit))))

(.delete (io/file db))
(create-db!)
```

Parse sqlite CSV output

```clojure
(insert-rows! [{:name "Bison" :scientific-name "Bos gaurus"}
               {:name "Duck"  :scientific-name "Anas Platyrhynchos"}])

(defn csv-data->maps [csv-data]
  (map zipmap
       (->> (first csv-data) ;; First row is the header
            (map keyword) ;; Drop if you want string keys instead
            repeat)
       (rest csv-data)))

(defn all-animals []
  (let [csv-string (-> (query (str/join " " ["select * from animals"]))
                       :out)
        reader (java.io.StringReader. csv-string)]
    (with-open [rdr reader]
      (doall (csv-data->maps (csv/read-csv rdr :quote \')))))  )

(pp/print-table (all-animals))
```

On the JVM about 1.3s

```
borkdude@MBP2019 ~/Dropbox/talks/graalworkshop2021 $ time bb sqlite_shell_raw.clj

| :name |   :scientific_name |
|-------+--------------------|
| Bison |         Bos gaurus |
|  Duck | Anas Platyrhynchos |
bb sqlite_shell_raw.clj  0.02s user 0.02s system 88% cpu 0.052 total
borkdude@MBP2019 ~/Dropbox/talks/graalworkshop2021 $
```

Enhancement: sql lib

tools.deps / maven integration

```clojure
;; Load HoneySQL from Clojars:
(deps/add-deps '{:deps {seancorfield/honeysql {:mvn/version "2.0.0-alpha2"}}})

(require '[honey.sql :as sql]
         '[honey.sql.helpers :as h])

(defn create-db! []
  (when (not (.exists (io/file db)))
    (query (first
            (sql/format
             (-> (h/create-table :animals)
                 (h/with-columns [[:name :text]
                                  [:scientific-name :text]])))))))

(defn insert-sql [animals]
  (sql/format (-> (h/insert-into :animals)
                  (h/values animals))
              {:inline true}))

(defn insert-rows! [animals]
  (let [sql (insert-sql animals)
        {:keys [:exit :err]}
        (query (first sql))]
    (when (not (zero? exit))
      (println "Error inserting var!" err)
      (System/exit exit))))
```

+50ms for loading 1400 lines of Clojure

100ms total

Enhancement: sqlite pod

```clojure
#!/usr/bin/env bb

(ns example
  (:require [babashka.deps :as deps]
            [babashka.pods :as pods]
            [clojure.java.io :as io]
            [clojure.pprint :as pp]))

(def db "/tmp/analysis.db")

;; sqlite pod, standalone, written in golang, ~9mb
(pods/load-pod 'org.babashka/go-sqlite3 "0.0.1")
(require '[pod.babashka.go-sqlite3 :as sqlite])

;; Load HoneySQL from Clojars:
(deps/add-deps '{:deps {seancorfield/honeysql {:mvn/version "2.0.0-alpha2"}}})

(require '[honey.sql :as sql]
         '[honey.sql.helpers :as h])

(def create-sql
  (sql/format
   (-> (h/create-table :animals)
       (h/with-columns [[:name :text]
                        [:scientific-name :text]]))))

(defn create-db! []
  (when (not (.exists (io/file db)))
    (let [sql (sql/format
               (-> (h/create-table :animals)
                   (h/with-columns [[:name :text]
                                    [:scientific-name :text]])))]
      (sqlite/execute! db sql))))
```

Sqlite pod: bb RPC-like extension

Pods are started only once: now 80ms total

No more shell output parsing, normal function calls

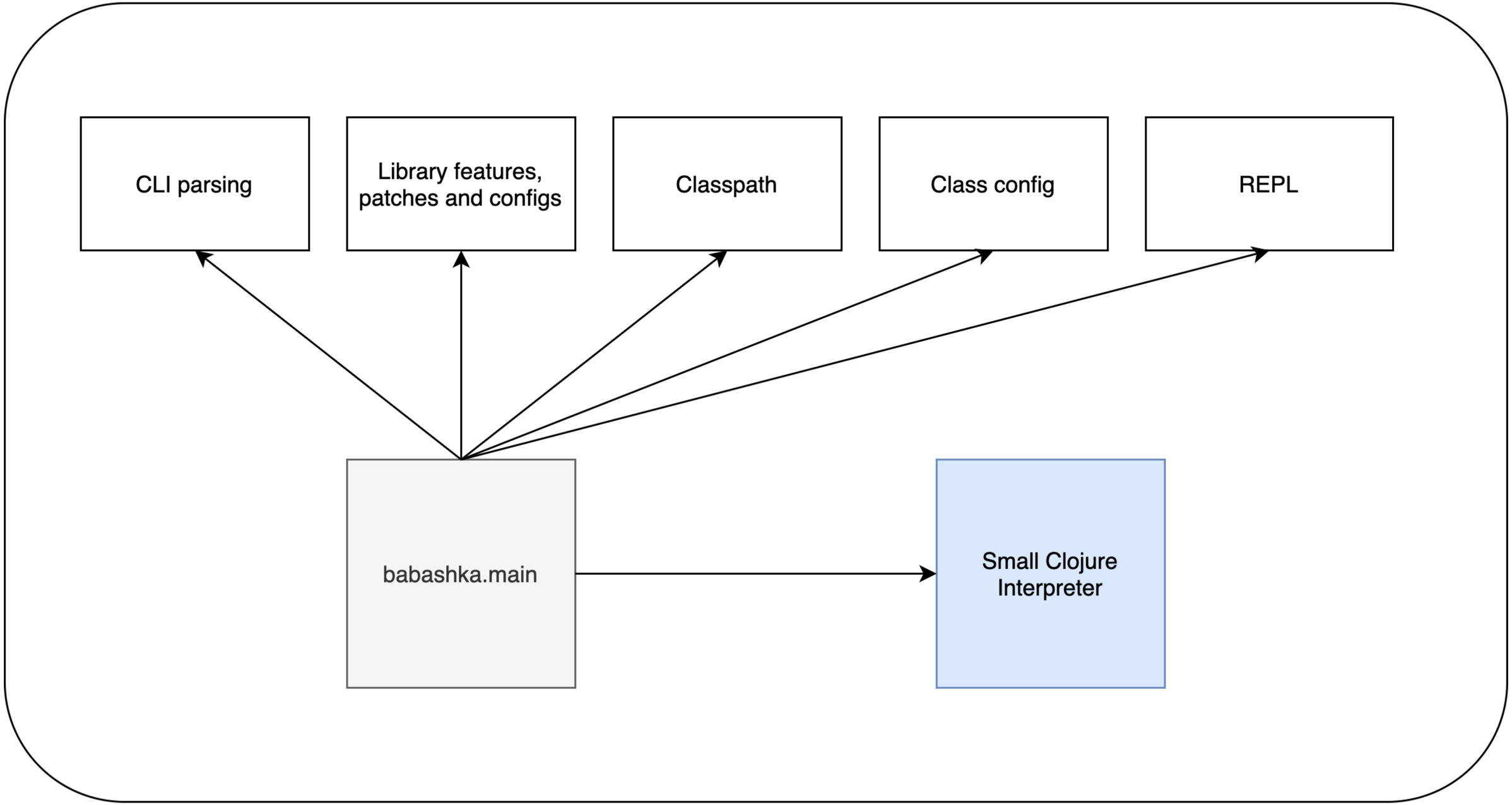
```clojure
(defn insert-sql [animals]
  (sql/format (-> (h/insert-into :animals)
                  (h/values animals))))

(defn insert-rows! [animals]
  (sqlite/execute! db (insert-sql animals)))

(.delete (io/file db))
(create-db!)
(insert-rows! [{:name "Bison" :scientific-name "Bos gaurus"}
               {:name "Duck"  :scientific-name "Anas Platyrhynchos"}])

(defn all-animals []
  (sqlite/query db ["select * from animals"]))

(pp/print-table (all-animals))
```


CLI parsing
Library features, patches and configs
Classpath
Class config
REPL
babashka.main
Small Clojure Interpreter
babashka

Supported classes for interop: reflection config

```
[ {
  "name" : "[B",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "[I",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "[Ljava.lang.Object;",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "[Ljava.time.temporal.TemporalField;",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "clojure.lang.ArityException",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "clojure.lang.BigInt",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "clojure.lang.ExceptionInfo",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.BufferedReader",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.BufferedWriter",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.ByteArrayInputStream",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.ByteArrayOutputStream",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.Console",
  "allPublicMethods" : true,
```

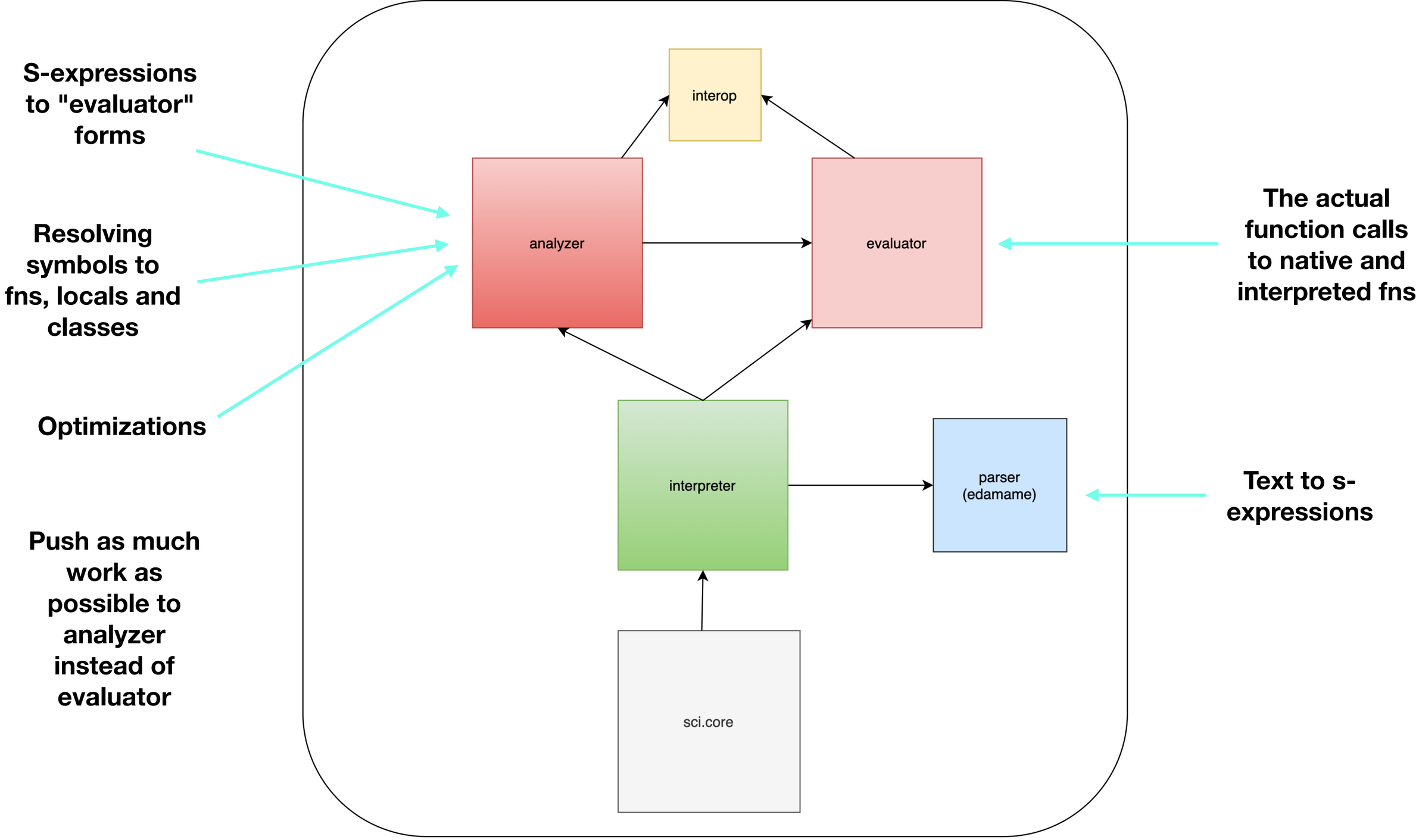
```
  "name" : "java.io.File",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.FileFilter",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.FileNotFoundException",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.FileReader",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.FilenameFilter",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.IOException",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.InputStream",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.InputStreamReader",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.OutputStream",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.PushbackInputStream",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.Reader",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.SequenceInputStream",
  "allPublicMethods" : true,
  "allPublicFields" : true,
```

```
  "name" : "java.io.SequenceInputStream",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.StringReader",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.StringWriter",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.io.Writer",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.ArithmeticException",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.AssertionError",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Boolean",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Byte",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Character",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Class",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.ClassNotFoundException",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Comparable",
  "allPublicMethods" : true,
  "allPublicFields" : true,
```

```
  "name" : "java.lang.Comparable",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Double",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Exception",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Float",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.IllegalArgumentException",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Integer",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Iterable",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Long",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Math",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Number",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.NumberFormatException",
  "allPublicMethods" : true,
  "allPublicFields" : true,
  "allPublicConstructors" : true
}, {
  "name" : "java.lang.Object",
  "allPublicMethods" : true,
  "allPublicFields" : true,
```

Interpreter: SCI

- Split out into its own project: Small Clojure Interpreter (sci)
- Written in Clojure itself: .cljc -> runs on JVM and ClojureScript.
- Leveraged by other native CLIs and ClojureScript projects
- Performance: not as good as compiled Clojure or a Truffle interpreter, but good enough for typical bash-like scripts
- Yields small images (~11mb) / JS bundles (~120kb gzipped)
- Can be used to glue together natively compiled functions using interpreted code



Small Clojure Interpreter

Sci: eval-string

```
(require '[sci.core :as sci])
(sci/eval-string "(+ 1 2 3)") ;;=> 6

(def ctx (sci/init {:namespaces {'foo {'x 1}}}))
(sci/eval-string* ctx "foo/x") ;;=> 1

(sci/eval-string* ctx "
  (require '[foo :refer [x]])
  (defn add-x [n] (+ n x))")

(sci/eval-string* ctx "(add-x 10)") ;;=> 11
```

Sci: mixing native and interpreted fns

```
(def ctx (sci/init {:namespaces
                    {'clojure.core {'assoc assoc}
                     'cheshire.core {'generate-string
                                     generate-string}}}))

(sci/eval-string* ctx
  "(cheshire.core/generate-string (assoc {:a 1} :b 2))")

;;=> {"a":1,"b":2}
```

Bootleg: sci-based static site CLI

A simple page:

```
$ cat example-simple.clj
[:html
 [:body
  [:h1 "A simple webpage"]
  [:p "Made with bootleg for maximum powers!"]]]
```

```
$ bootleg example-simple.clj
<html><body><h1>A simple webpage</h1><p>Made with bootleg for maximum powers!</p></body></html>
```

A dynamic example:

```
$ cat example-dynamic.clj
[:div.countdown
 (for [n (range 10 0 -1)]
   [:p n])
 [:p "blast off!"]]
```

```
$ bootleg example-dynamic.clj
<div class="countdown"><p>10</p><p>9</p><p>8</p><p>7</p><p>6</p><p>5</p><p>4</p><p>3</p><p>2</p><p>1</p><p
```

NextJournal (sci in browser)

Exercise 1.21: A dumbbell – Ne

nextjournal.com/samritchie/sicm-ch1-ex1-21

›Nextjournal

Explore Docs

Table of Contents

Here are the Lagrange equations, which, if you squint, are like Newton's equations from part a.

```
(let [L (L-free-constrained 'm_0 'm_1 'l)
      f ((Lagrange-equations L) q-rect)]
  (f 't))
```

✓ ClojureScript

$$\begin{bmatrix} \frac{l\,m_0\,D^2x_0(t) - F(t)\,x_1(t) + F(t)\,x_0(t)}{l} \\ \frac{l\,m_0\,D^2y_0(t) - F(t)\,y_1(t) + F(t)\,y_0(t)}{l} \\ \frac{l\,m_1\,D^2x_1(t) + F(t)\,x_1(t) - F(t)\,x_0(t)}{l} \\ \frac{l\,m_1\,D^2y_1(t) + F(t)\,y_1(t) - F(t)\,y_0(t)}{l} \\ \frac{-l^2 + (x_1(t))^2 - 2\,x_1(t)\,x_0(t) + (x_0(t))^2 + (y_1(t))^2 - 2\,y_1(t)\,y_0(t) + (y_0(t))^2}{2\,l} \end{bmatrix}$$

clj-kondo hooks

```clojure
(ns math.expression
  (:require [slingshot.slingshot :refer [throw+ try+]]))

(defn read-file [file]
  (try+
   (prn file)
   (catch [:type :tensor.parse/bad-tree] {:keys [treex hint]}
                                                  Unresolved symbol: treex
     (prn hint)
     (throw+))
   (catch Object _
     (throw+))))
```

```clojure
(ns math.expression
  (:require [slingshot.slingshot :refer [throw+ try+]]))

(defn read-file [file]
  (try+
   (prn file)
   (catch [:type :tensor.parse/bad-tree] {:keys [treex hint]}
                                                  unused binding treex
     (prn hint)
     (throw+))
   (catch Object _
     (throw+))))
```

```clojure
(defn try+ [{:keys [node]}]
  (let [children (rest (:children node))
        [body catches]
        (loop [body children
               body-exprs []
               catches []]
          (if (seq body)
            (let [f (first body)
                  f-sexpr (api/sexpr f)]
              (if (and (seq? f-sexpr) (= 'catch (first f-sexpr)))
                (recur (rest body)
                       body-exprs
                       (conj catches (expand-catch f)))
                (recur (rest body)
                       (conj body-exprs f)
                       catches)))
            [body-exprs catches]))
        new-node (api/list-node
                  [(api/token-node 'let)
                   (api/vector-node
                    [(api/token-node '&throw-context) (api/token-node nil)])
                   (api/token-node '&throw-context) ;; use throw-context to avoid warning
                   (with-meta (api/list-node (list* (api/token-node 'try)
                                                    (concat body catches)))
                     (meta node))])]
    ;; (prn (api/sexpr new-node))
    {:node new-node}))
```

Clojure + GraalVM

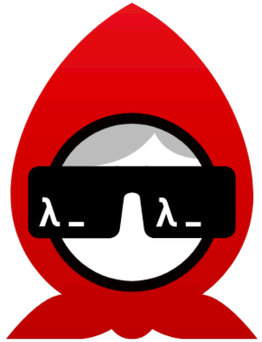
- CLJ-1472: issue with GraalVM and locking macro
  - Solved in **1.10.2**
  - MethodHandle issue: solved in **GraalVM 21.0.0**
- https://github.com/lread/clj-graal-docs
- https://github.com/BrunoBonacci/graalvm-clojure/

```
{
  "name": "java.lang.reflect.AccessibleObject",
  "methods" : [{"name":"canAccess"}]
}
```

```
static {
        MethodHandle pred = null;
        try {
                if (! isJava8())
                        pred = MethodHandles.lookup().findVirtual(Method.class, "canAccess", MethodType.methodType(boolean.class, Object.class));
        } catch (Throwable t) {
                Util.sneakyThrow(t);
        }
        CAN_ACCESS_PRED = pred;
}
```

Truffle

- Espresso compilation of Clojure compiler?
- Clojure on Truffle? (Thesis from 2015)
- AOT of guest language?
- Defining new classes at runtime?
- Mixing host language AOT-ed fns called from guest language?



ba·bash·ka

Selected talks:

- Babashka and GraalVM; taking Clojure to new places
- Writing Clojure on the command line
- Babashka and sci internals

On Github:

- https://github.com/babashka/babashka
- https://github.com/borkdude/sci

Michiel Borkent
@borkdude