

Clojure in 10 Minutes

Clojure Meet-Up Copenhagen

September 23, 2010

Martin Jul, mj@ative.dk / @mjul

What is Clojure?

- Lisp
- Functional Programming
- Immutable, persistent data structures
- Software Transactional Memory
- Object-oriented (sort of)
- Macros
- Managed code: JVM and CLR

Clojure is the future!
-- Uncle Bob

*Getting off the
Lisp Island*



Syntax and Data Types

`; This is a comment, semicolon is similar to // in C# / Java`

`:: List is the most common data structure
(1 2 3)`

List

`:: Vector is a list that is indexable by position
[1 2 3]`

Vector

`:: Maps are associative data structures
{:key "value", :id 42}`

Map

`:: Sets are mathematical sets
#{1 2 3}`

Set

`:: Expressions are lists of form (function arg-1 arg-2 ...)
(println "Hello, World")
(+ 1 2 3)`

Expression

file: core.clj

Functional Programming

```

(def answer 42)

(defn double [x]
  (* 2 x))

(defn max [a b]
  (if (< a b) b a))

(defn factorial [x]
  (reduce * 1 (range 1 (inc x))))

;; Higher-order functions
(map double my-vec)
(reduce + 0 my-list)
(filter even? my-list)
(remove even? my-list)
(sort-by :name [{:id 1, :name "Anders"} {:id 2, :name "Bjarne"}])

(def my-list (list 1 2 3))
(def my-vec [10 20 30])
(def my-map {:clojure "Rich", :perl "Larry", :python "Guido", :ruby "Matz"})
(def my-set #{:a :b :c})

;; List comprehension
(for [x (range 3)]
  (str "Element " x))

;; Map operations
(keys my-map)
(vals my-map)
(assoc my-map :c++ "Bjarne")

;; Seq operations
(first my-list)
(rest my-vec)
(conj my-list "a" "b" "c")
(cons "CAR" my-list)

```

file: functional.clj

Simple and Concise

```
public class StringUtils {  
    public static boolean isBlank(String str) {  
        int strLen;  
        if (str == null || (strLen = str.length()) == 0) {  
            return true;  
        }  
        for (int i = 0; i < strLen; i++) {  
            if ((Character.isWhitespace(str.charAt(i)) == false)) {  
                return false;  
            }  
        }  
        return true;  
    }  
}
```

*This makes my
eyes hurt!*

```
(defn blank? [s]  
  (every? #(Character/isWhitespace %) s))
```

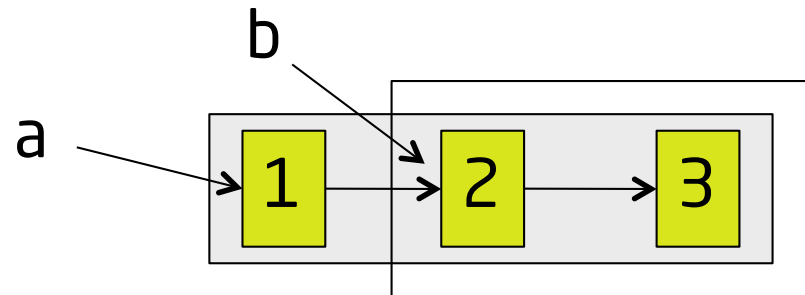
Source: Programming Clojure by Stuart Halloway (Pragmatic Programmers, 2009)



"State - you're doing it wrong"

- Immutable
- Persistent (strukturnal sharing, not copying)

```
(def a (list 1 2 3))  
(def b (rest a))
```



- Simpler code
- Less concurrency issues

*e.g. stable
enumerations*

STM - Software Transactional Memory

```
(defn post [account amount msg]
  (conj account {:amount amount, :msg msg}))
```

```
(defn transfer [from to amount msg]
  (dosync
    (alter from post (- amount) msg)
    (alter to post amount msg)))
```

```
(defn balance [account]
  (reduce + 0 (map :amount account)))
```

```
(deftest transfer-test
  (testing "Transfer between accounts."
    (let [a (ref [])
          b (ref [])]
      (transfer a b 10 "test")
      (is (= [{:amount -10, :msg "test"}] @a) "Money should be deducted from a")
      (is (= [{:amount 10, :msg "test"}] @b) "Money should be added to b"))))
```

*concurrency-safe
in-memory
transactions!*

file: stm.clj

Object Oriented with Protocols

```
(defprotocol Price
  (price [x]))
(defprotocol Vat
  (vat [x]))
```

```
(defn price-with-vat [x]
  (+ (price x) (vat x)))
```

```
(defn standard-vat [x]
  (* (/ 25 100) (price x)))
```

```
(extend-type menu-item
  Price
  (price [x] (:price x))
  Vat
  (vat [x] (standard-vat x)))
```

```
(price-with-vat espresso)
(vat iver)
```

```
(defrecord menu-item [name price])
(def espresso (menu-item. "Espresso" 12))
(def cortado (menu-item. "Cortado" 16))
```


```
(defrecord stamp [series name value])
(def iver (stamp.
  "Royal Danish Navy 500th Anniversary"
  "Iver Huitfeldt" 5.50))
```

```
(extend-type stamp
  Price
  (price [s] (:value s))
  Vat
  (vat [s] 0))
```

*Open for extension,
closed for modification*

file: oo.clj


Multi-methods



```
(defmulti type-info class)
(defmethod type-info String [x] (str "It is a String: " x))
(defmethod type-info menu-item [x] (str "It is a menu item!"))
(defmethod type-info :default [x] (str "It is a " (class x)))
```

*Open-Closed Principle
goodness*

```
(defrecord menu-item [name type price])
(def espresso (menu-item. "Espresso" :drink 12))
(def cortado (menu-item. "Cortado" :drink 16))
(def burger (menu-item. "Burger Royale" :food 100))
```

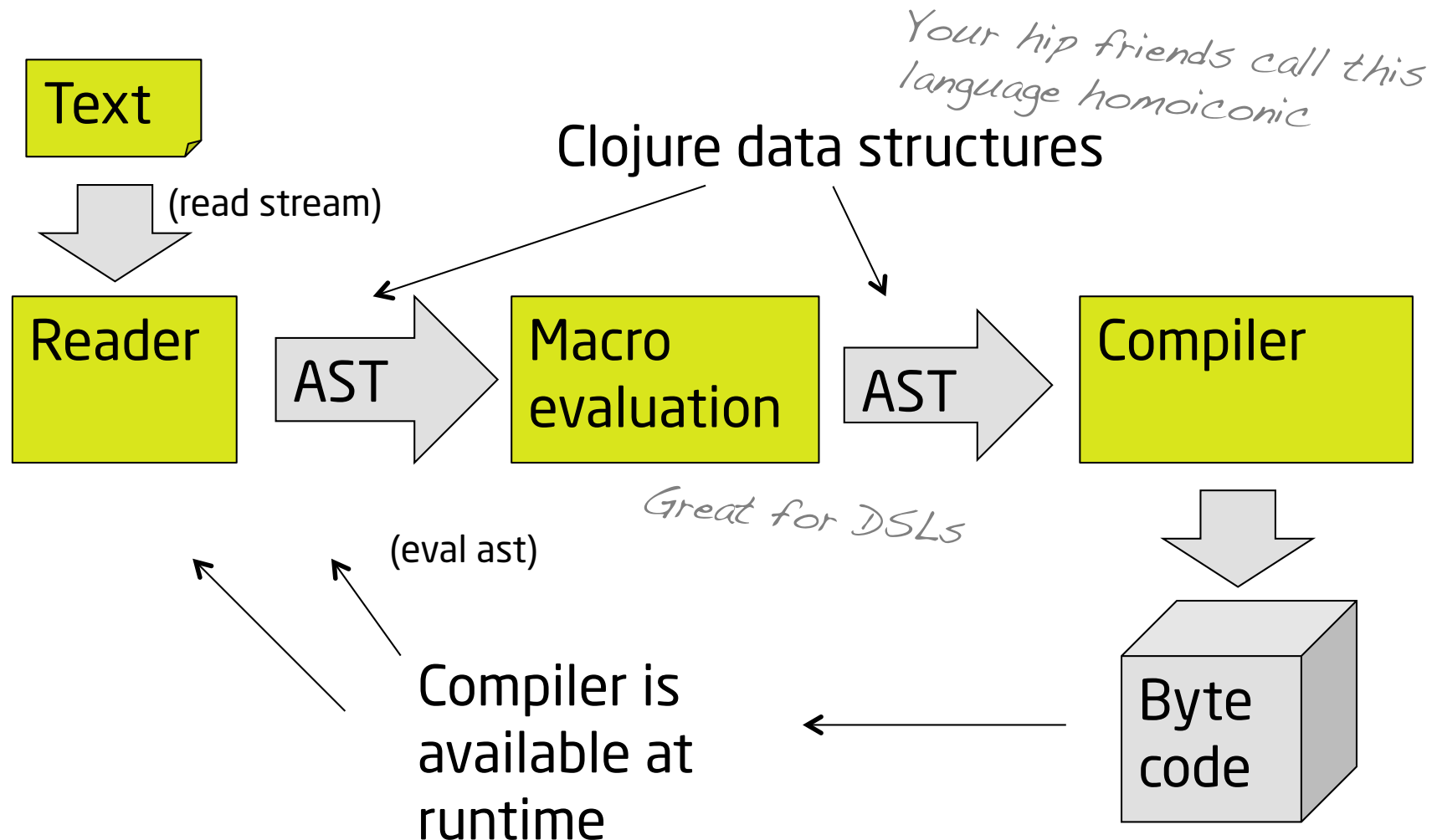


```
(defmulti description :type)
(defmethod description :drink [x] (str "Drink a wonderful " (:name x)))
(defmethod description :food [x] (str "Savour a tasty " (:name x)))
```

*Not your mother's
dispatch*

file: multi.clj

The Clojure Compiler



Example: Compojure web app

*Shorter than your
web.config!*

```
(ns hello-world
  (:use compojure.core, ring.adapter.jetty)
  (:require [compojure.route :as route]))

(defroutes main-routes
  (GET "/" [] "<h1>Hello World</h1>")
  (route/not-found "<h1>Page not found</h1>"))

(run-jetty main-routes {:port 8080})
```

Source: <http://github.com/weavejester/compojure>