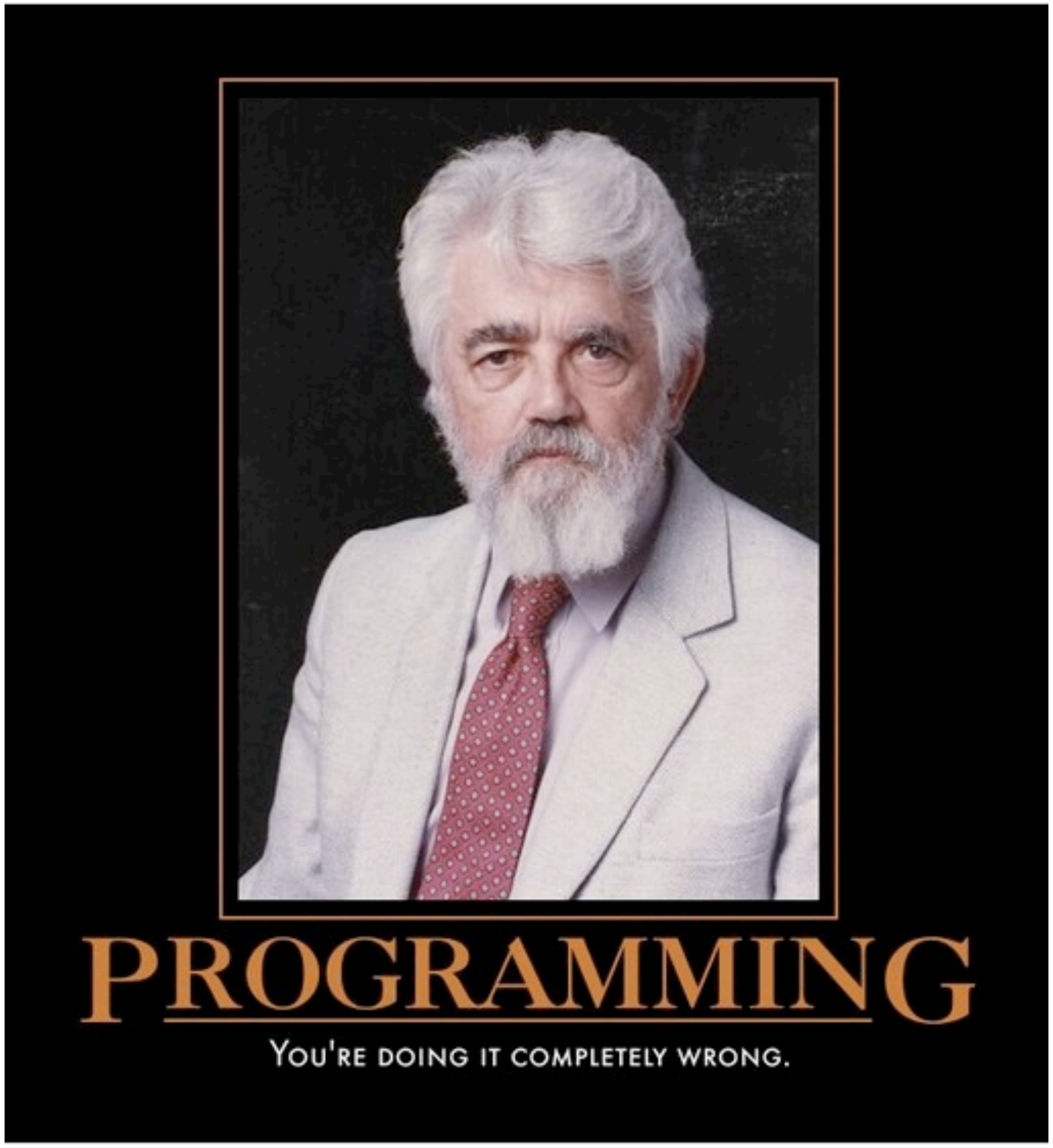
Zero to Clojure in 90 Minutes

Colin Jones
Software Craftsman at 8th Light
@trptcolin

What is Clojure?

(It's a Lisp)

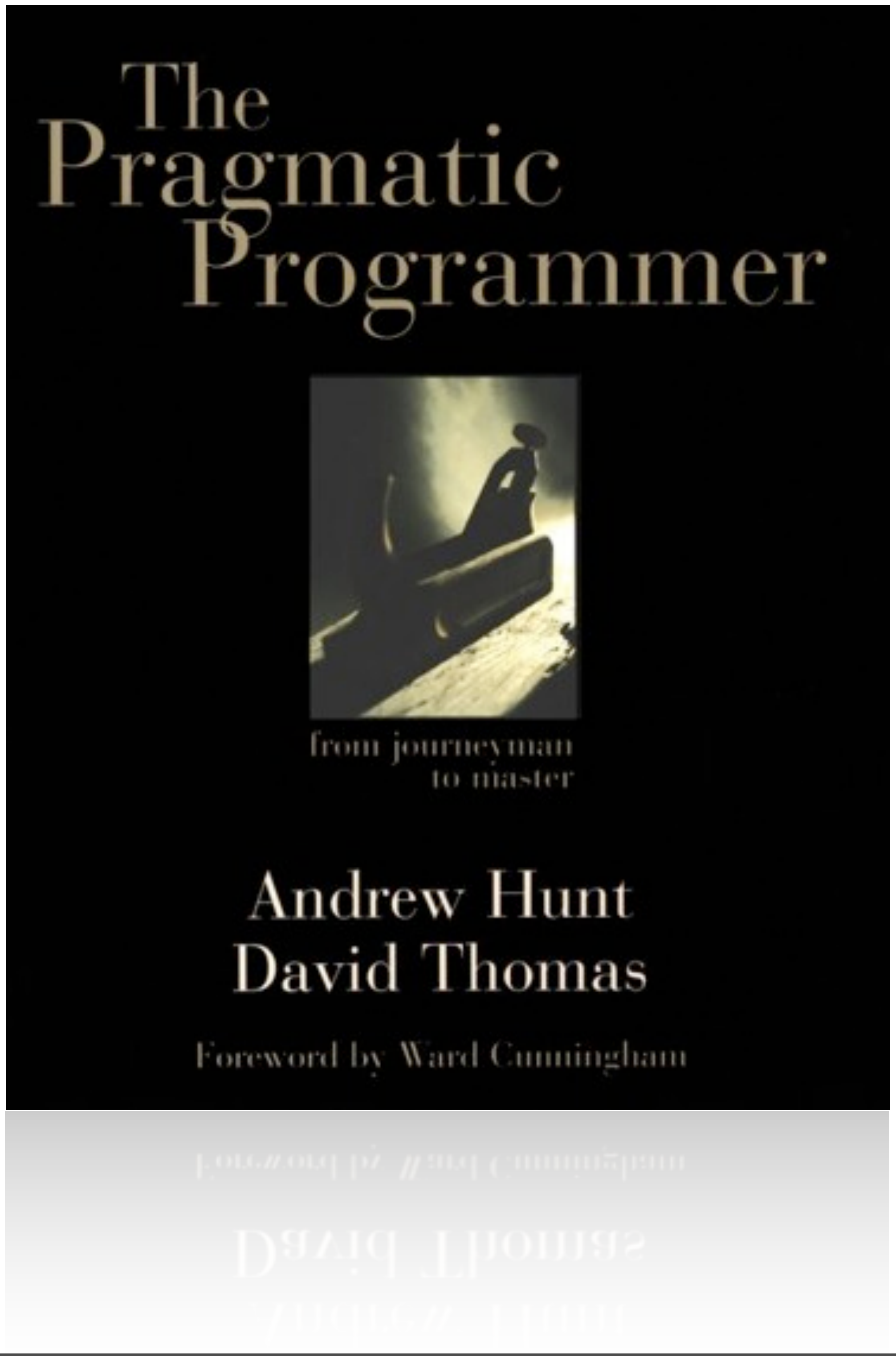


It runs on the JVM

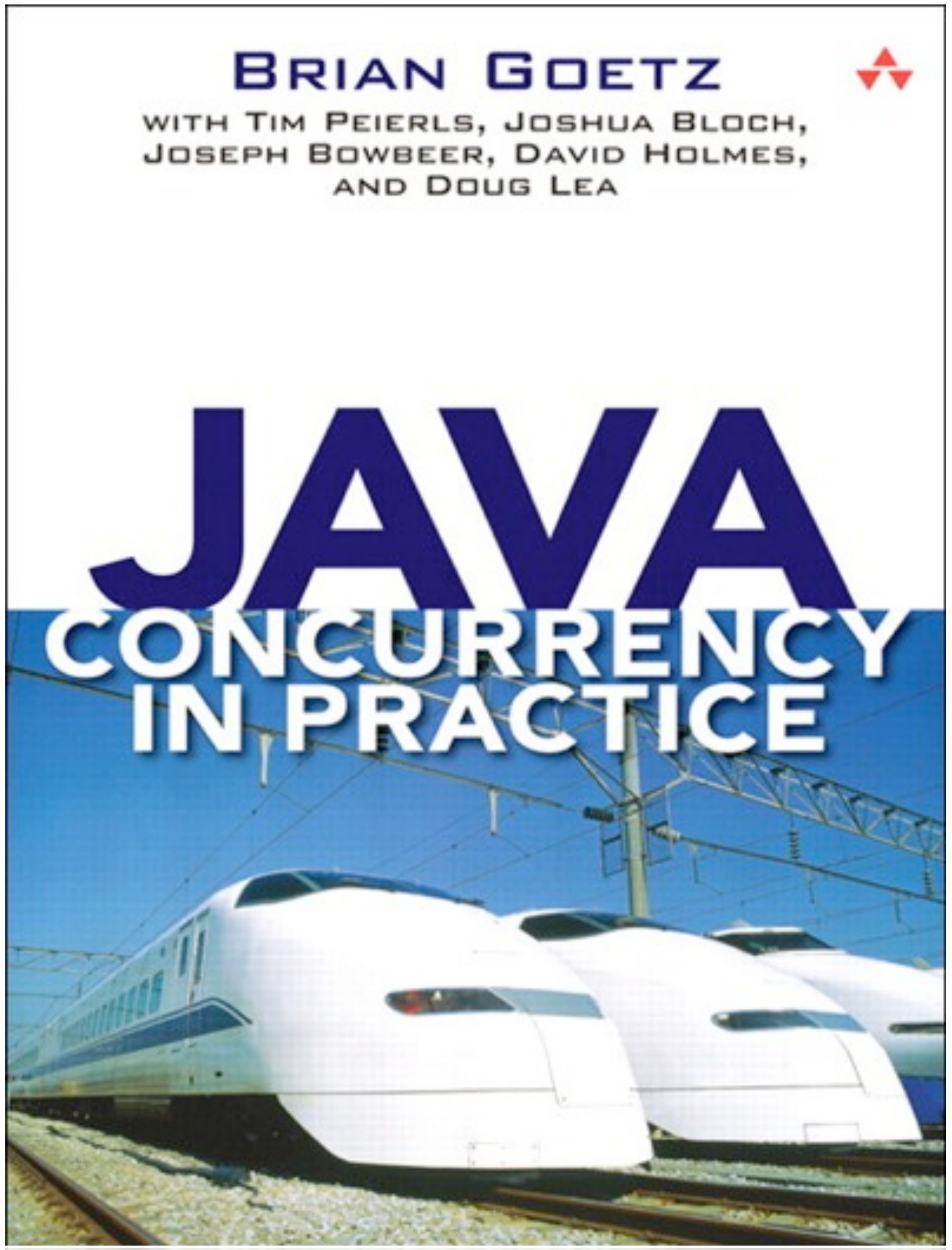
Created in 2007
by Rich Hickey

Why should we care?

Learning is good.

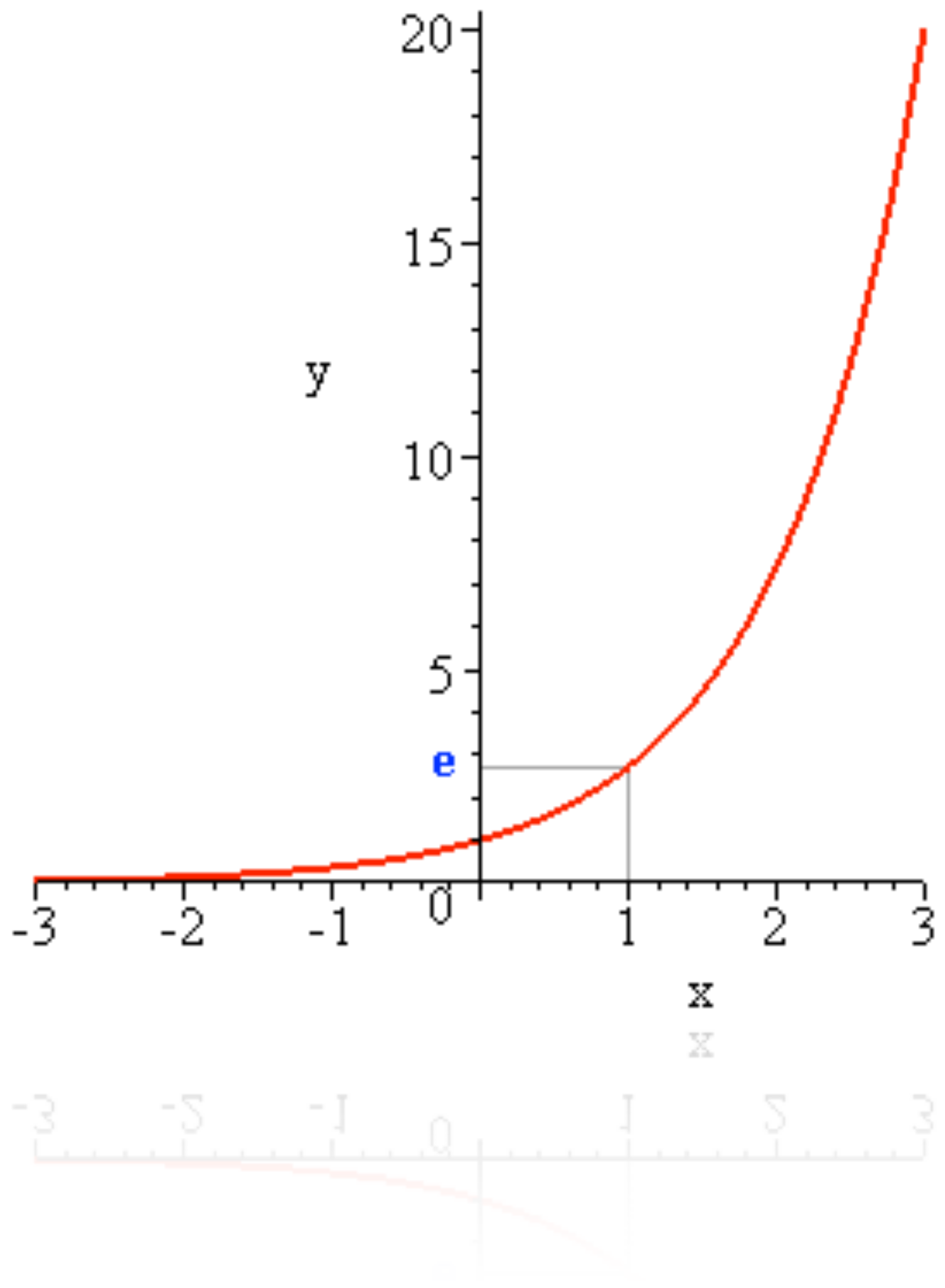


Concurrency can be scary.


BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH,
JOSEPH BOWBEER, DAVID HOLMES,
AND DOUG LEA
JAVA
CONCURRENCY
IN PRACTICE

Side effects can cause a mess.

Functional programming can help



Syntax & Data Structures

Expressions

```
(doc +)

(find-doc “regex”)
```

Numbers

Integer	`42`
Long	`9999999999999`
BigInteger	`9999999999999999999`
Double	`4.2`
BigDecimal	`4000.2M`

Ratio	`1/3`

More

String	`“go to the”`
Character	`\p \a \r \k`
Regex	`#”\d+”`

Nil	`nil`
Boolean	`true false`

Keyword	`:really/soon`
Symbol	`some-time`

Collections

List	`(1 2 3 4 5)`
Vector	`[1 2 3 4 5]`

Map	`{:first-name “colin”,`
	` :last-name “jones”}`

Set	`#{a b c d e}`

Expressions

```
(doc +)

(find-doc “regex”)
```

Diving in

Clojure Functional Koans

http://trptcolin.github.com

USB Drives

REPL

```
=> Read
=> Evaluate
=> Print
```

(+ JRE functional-koans) => WIN!

From Functional Koans directory:

Mac / Linux: `./repl.sh`

Windows: `repl`

From anywhere:

```
java -jar /path/to/clojure.jar
```

Functions

clojure.test

```
user=> (use ‘clojure.test)
nil
user=> (is (= 1 1))
true
user=> (is (= 1 2))
FAIL in clojure.lang.PersistentList$EmptyList@1
15)
expected: (= 1 2)
  actual: (not (= 1 2))
false
```

Equality

```
user=> (is (= “Colin” “Colin”))
true
user=> (is (= nil nil))
true
user=> (is (= ‘(1 2 3) [1 2 3]))
true
user=> (is (= 1.0 1 4/4))
true
```

Math fun(ctions)

```
user=> (is (= 15 (+ 1 2 3 4 5)))
true

user=> (is (< 1 2 3))
true
```

Hide and seq?

```
user=> (is (= :a (first [:a :b :c])))
true

user=> (is (= [:b :c] (rest [:a :b :c])))
true

user=> (is (= [:c :b :a] (reverse [:a :b :c])))
true
```

Defining our own functions

```
user=> (def square-1 (fn [x] (* x x)))
#‘user/square-1

user=> (is (= 9 (square-1 3)))
true

user=> (def square-2 #(* % %))
#‘user/square-2

user=> (is (= 9 (square-2 3)))
true

user=> (defn square-3 [x] (* x x))
#‘user/square-3

user=> (is (= 9 (square-3 3)))
true
```

map

```
user=> (defn square [x] (* x x))
#‘user/square

user=> (is (= ‘(1 4 9 16 25)
              (map square ‘(1 2 3 4 5))))
true
```

filter

```
user=> (is (= [odd?]
              (filter fn?
                      [“odd” :odd odd?])))
true
```

apply

```
user=> (is (= “dog” (str \d \o \g)))
true
user=> (def dog-letters [\d \o \g])
true

user=> (is (= “dog” (str dog-letters)))
FAIL in clojure.lang.PersistentList$EmptyList
expected: (= “dog” (str dog-letters))
  actual: (not (= “dog” “[\\d \\o \\g]”))
false

user=> (is (= “dog” (apply str dog-letters)))
true
```

Laziness

The whole numbers

```
user=> (def whole-numbers (iterate inc 0))
#‘user/whole-numbers

user=> (is (= (range 0 20) (take 20 whole-numbers)))
true

user=> (is (= (range 20 40)
              (take 20 (drop 20 whole-numbers))))
true
```

Let’s try it in the REPL

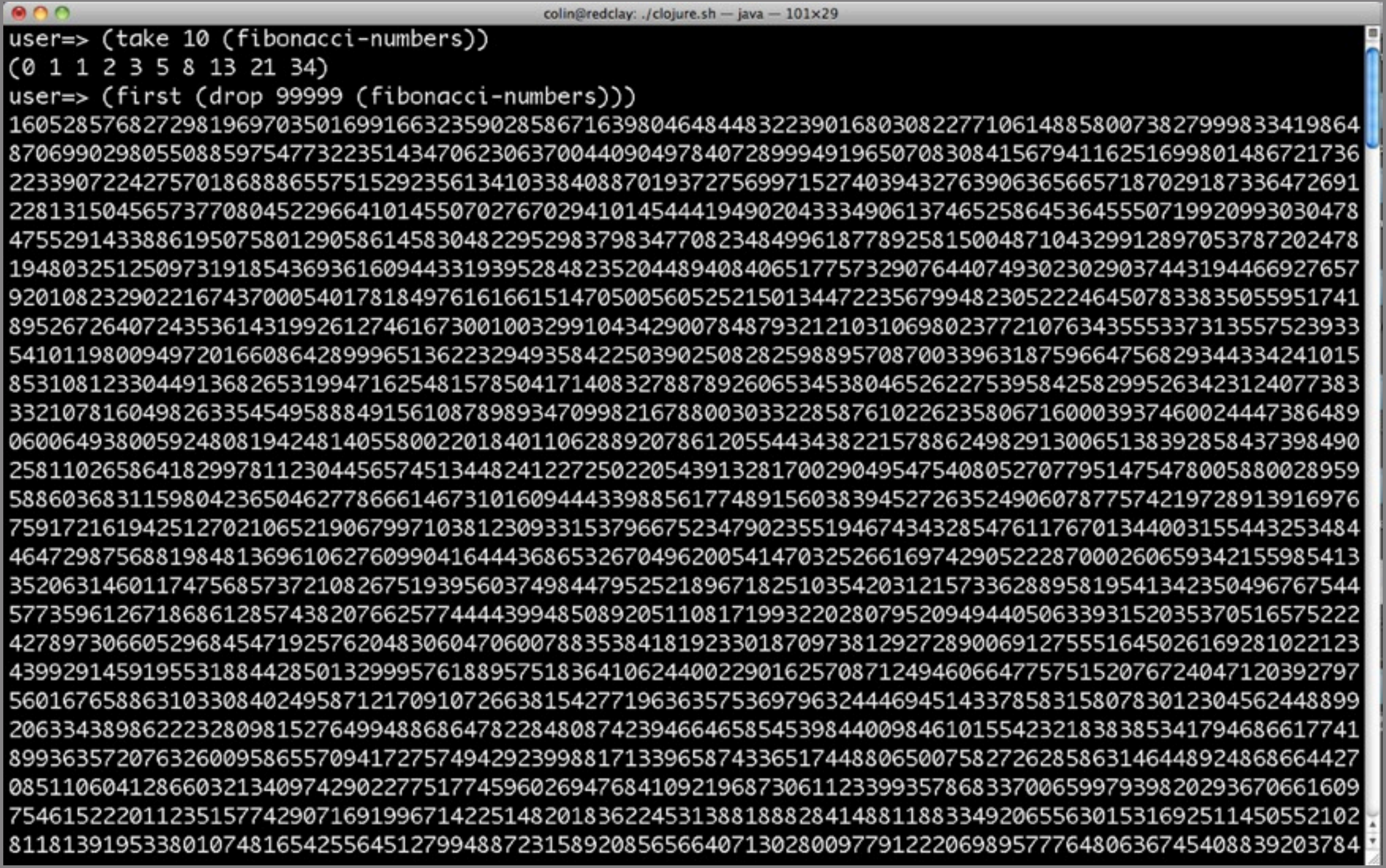
```
user=> (def whole-numbers (iterate inc 0))
#‘user/whole-numbers

user=> whole-numbers
(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 2                       3 34 35 36 37 38
39 40 41 42 4                       1 52 53 54 55 56
57 58 59 60 6                       9 70 71 72 73 74
75 76 77 78 7                       7 88 89 90 91 92
93 94 95 96 9                       104 105 106 107
108 109 110 1                       17 118 119 120
121 122 123 1                       30 131 132 133
134 135 136 1                       43 144 145 146
```

Don't hold onto your head

```
user=> (def whole-numbers (iterate inc 0))
#'user/whole-numbers
user=> (first (drop 10000000 whole-numbers))
Exception in thread "main" java.lang.OutOfMemoryError: Java heap
space
```

```
user=> (defn whole-numbers [] (iterate inc 0))
#'user/whole-numbers
user=> (first (drop 10000000 (whole-numbers)))
10000000
```

BIG NUMBERS!!!



```
user=> (take 10 (fibonacci-numbers))
(0 1 1 2 3 5 8 13 21 34)
user=> (first (drop 99999 (fibonacci-numbers)))
16052857682729819697035016991663235902858671639804648448322390168030822771061488580073827999833419864
87069902980550885975477322351434706230637004409049784072899949196507083084156794116251699801486721736
22339072242757018688865575152923561341033840887019372756997152740394327639063656657187029187336472691
22813150456573770804522966410145507027670294101454441949020433349061374652586453645550719920993030478
47552914338861950758012905861458304822952983798347708234849961877892581500487104329912897053787202478
19480325125097319185436936160944331939528482352044894084065177573290764407493023029037443194466927657
92010823290221674370005401781849761616615147050056052521501344722356799482305222464507833835055951741
89526726407243536143199261274616730010032991043429007848793212103106980237721076343555337313557523933
54101198009497201660864289996513622329493584225039025082825988957087003396318759664756829344334241015
85310812330449136826531994716254815785041714083278878926065345380465262275395842582995263423124077383
33210781604982633545495888491561087898934709982167880030332285876102262358067160003937460024447386489
06006493800592480819424814055800220184011062889207861205544343822157886249829130065138392858437398490
25811026586418299781123044565745134482412272502205439132817002904954754080527077951475478005880028959
58860368311598042365046277866614673101609444339885617748915603839452726352490607877574219728913916976
75917216194251270210652190679971038123093315379667523479023551946743432854761176701344003155443253484
46472987568819848136961062760990416444368653267049620054147032526616974290522287000260659342155985413
35206314601174756857372108267519395603749844795252189671825103542031215733628895819541342350496767544
57735961267186861285743820766257744443994850892051108171993220280795209494405063393152035370516575222
42789730660529684547192576204830604706007883538418192330187097381292728900691275551645026169281022123
43992914591955318844285013299957618895751836410624400229016257087124946066477575152076724047120392797
56016765886310330840249587121709107266381542771963635753697963244469451433785831580783012304562448899
20633438986222328098152764994886864782284808742394664658545398440098461015542321838385341794686617741
89936357207632600958655709417275749429239988171339658743365174488065007582726285863146448924868664427
08511060412866032134097429022775177459602694768410921968730611233993578683370065997939820293670661609
75461522201123515774290716919967142251482018362245313881888284148811883349206556301531692511450552102
81181391953380107481654255645127994887231589208565664071302800977912220698957776480636745408839203784
```

Clojure Functional Koans

Background

based on EdgeCase's Ruby Koans

Aaron Bedra of Relevance

Towards Clojure Enlightement

Mac / Linux: `./run.sh`

Windows: `run`

Managing State

vars

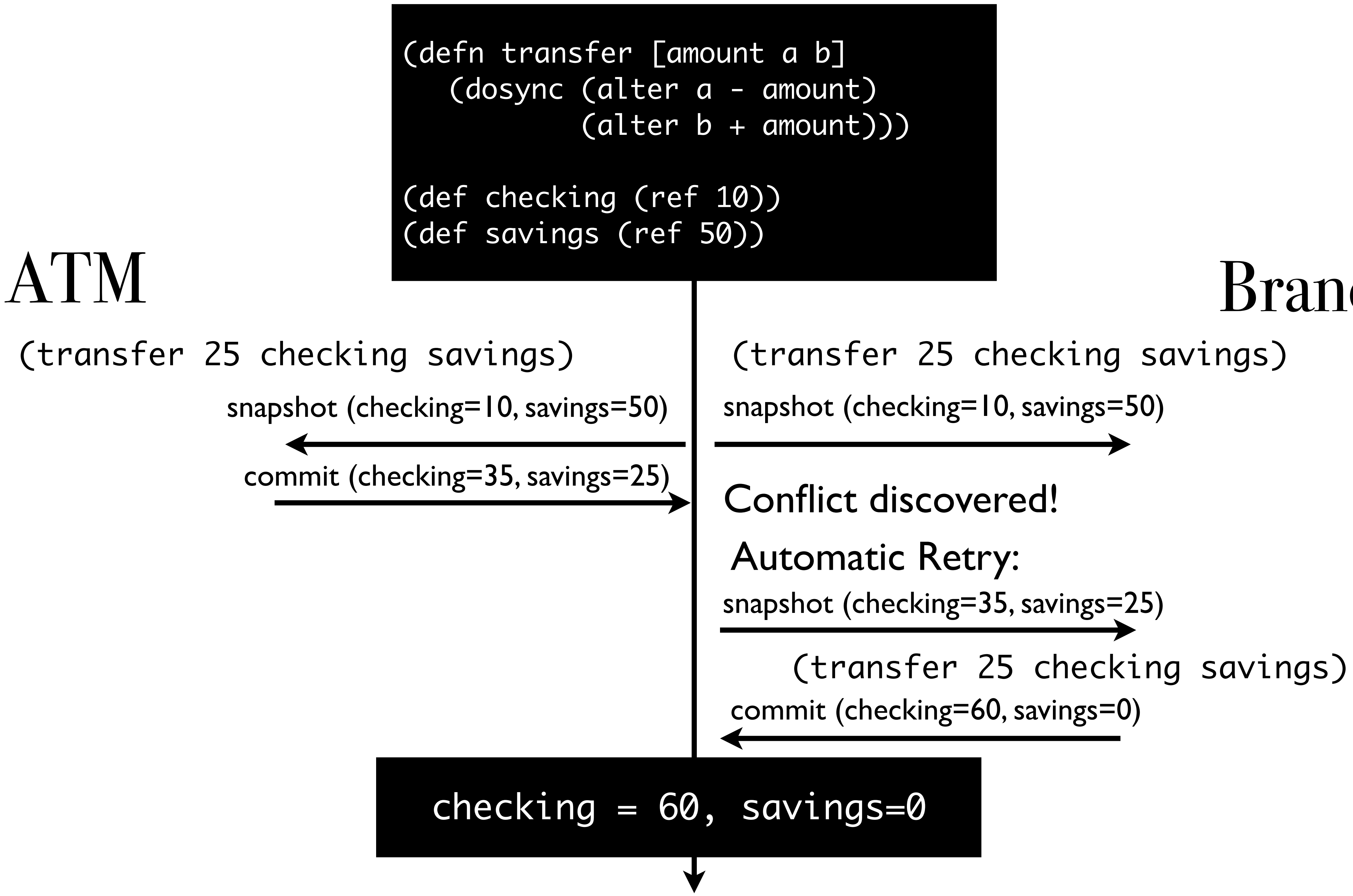
```
user=> (def x 42)
#‘user/x

user=> x
42

user=> (let [x :foo]
         x)
:foo

user=> x
42
```

refs

```
user=> (def attendees (ref 30))
#‘user/attendees
user=> attendees
#<Ref@343aff84: 30>
user=> @attendees
30
user=> (alter attendees dec)
java.lang.IllegalStateException: No transaction running
user=> (dosync (alter attendees dec))
29
user=> @attendees
29
```

refs

Bank Accounts



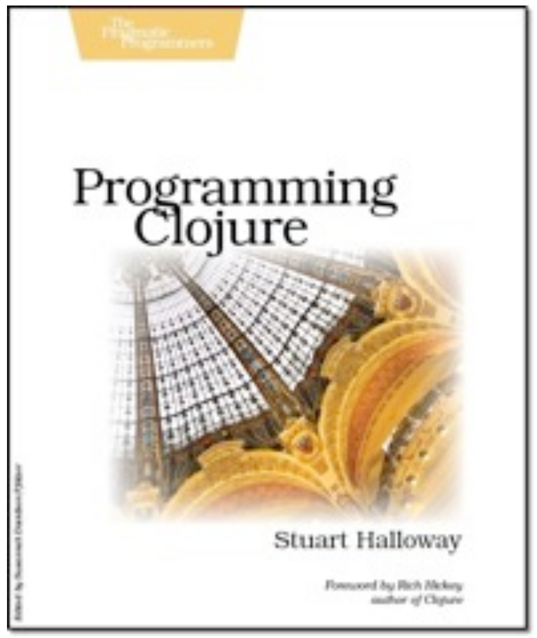
```
(defn transfer [amount a b]
   (dosync (alter a - amount)
           (alter b + amount)))

(def checking (ref 10))
(def savings (ref 50))
```

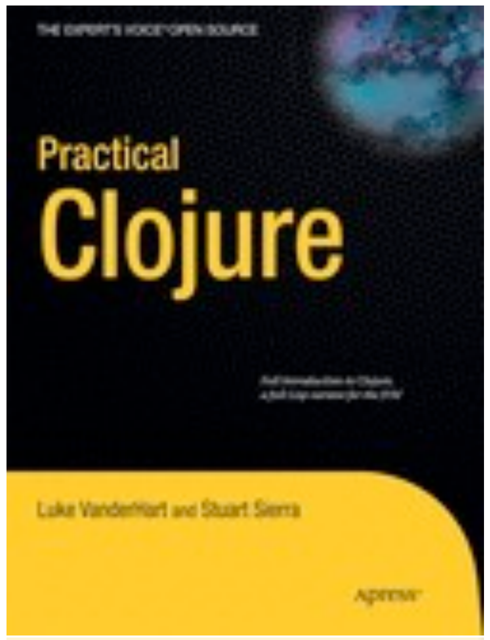
ATM
Branch
(transfer 25 checking savings)
snapshot (checking=10, savings=50)
commit (checking=35, savings=25)
(transfer 25 checking savings)
snapshot (checking=10, savings=50)
Conflict discovered!
Automatic Retry:
snapshot (checking=35, savings=25)
(transfer 25 checking savings)
commit (checking=60, savings=0)

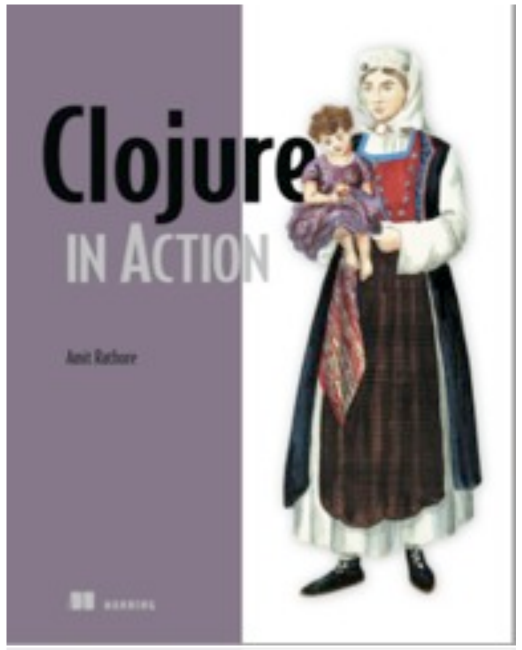
```
checking = 60, savings=0
```

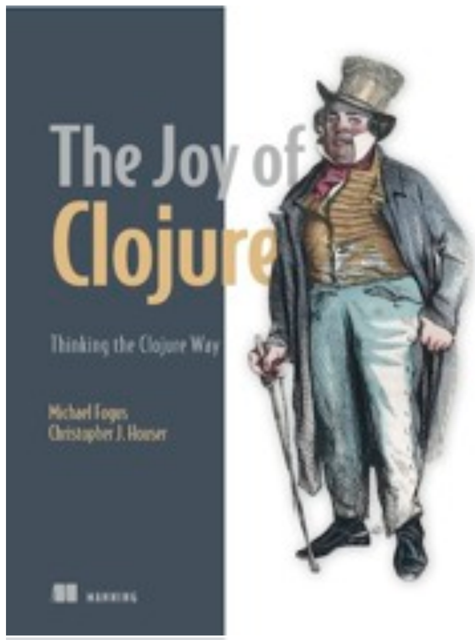
Further Study

Books


Programming Clojure
Stuart Halloway


Practical Clojure
Luke VanderHart and Stuart Sierra
Apress


Clojure IN ACTION
Amit Rathore


The Joy of Clojure
Thinking the Clojure Way
Michael Fogus
Christopher J. Houser

The Internets

Functional Koans

http://github.com/relevance/functional-koans/tree/clojure

Web docs

http://clojure.org/

Google Group

http://groups.google.com/group/clojure

IRC

#clojure on freenode.net

Questions?

Software CRAFTSMANSHIP north america

http://scna.softwarecraftsmanship.org

Early Bird Registration ends Aug 15th

$395

Uncle Bob Martin, Corey Haines, Dave Astels, Ken Auer, Chad Fowler, Keavy McMinn, Michael Feathers, Doug Bradbury, Enrique Comba Riepenhausen, and more...

CHICAGO, IL

October 15-16, 2010

Hosted by
8th Light, Inc
and
Obtiva