Betriebssysteme

5a. Clojure

Winter 2024
Peter Sturm

Functional Programming (FP)

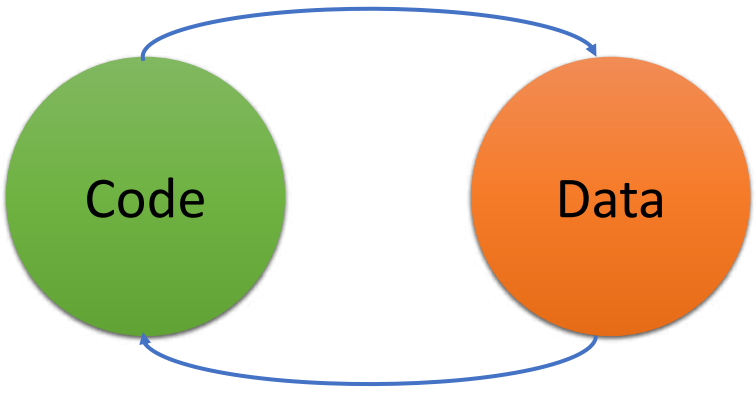
Minimalanforderung

Treat functions as something more than named subroutines for executing blocks of code

aus “Clojure in Action”

3

Functions are Values are Functions



4

Beispiel Clojure

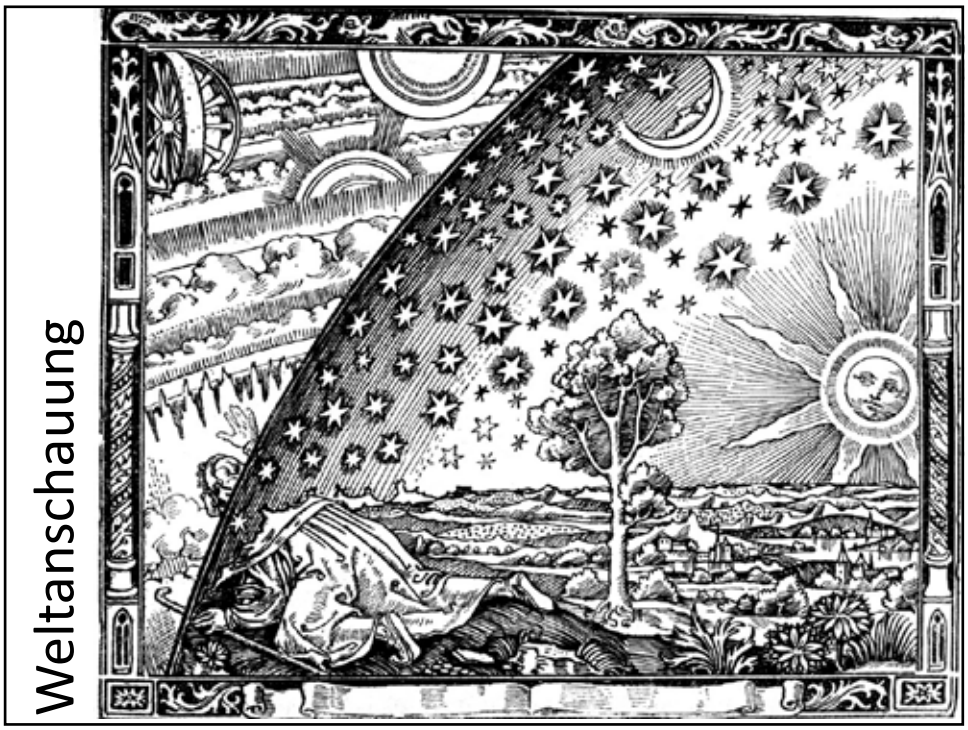
```
; ------------------------------------------------------------
; Data are code :-)
; ------------------------------------------------------------

(def mul_is_better_than_add
  (fn [code]
    (if (list? code)
      (map mul_is_better_than_add code)
      (if (= code '+) '* code)
      )
    )
  )

(def myprog '(+ (- 2 3) (+ 2 (* 4 5))))
(println (eval myprog))
(println (eval (mul_is_better_than_add myprog)))
```

5

Weitere Eigenschaften

- Pure Functions
  - Pure = Keine Seiteneffekte
- Referential Transparency
  - Gleiche Eingabe = Gleiche Ausgabe
- Immutable data structures as the default
- Controlled, explicit changes to state

6

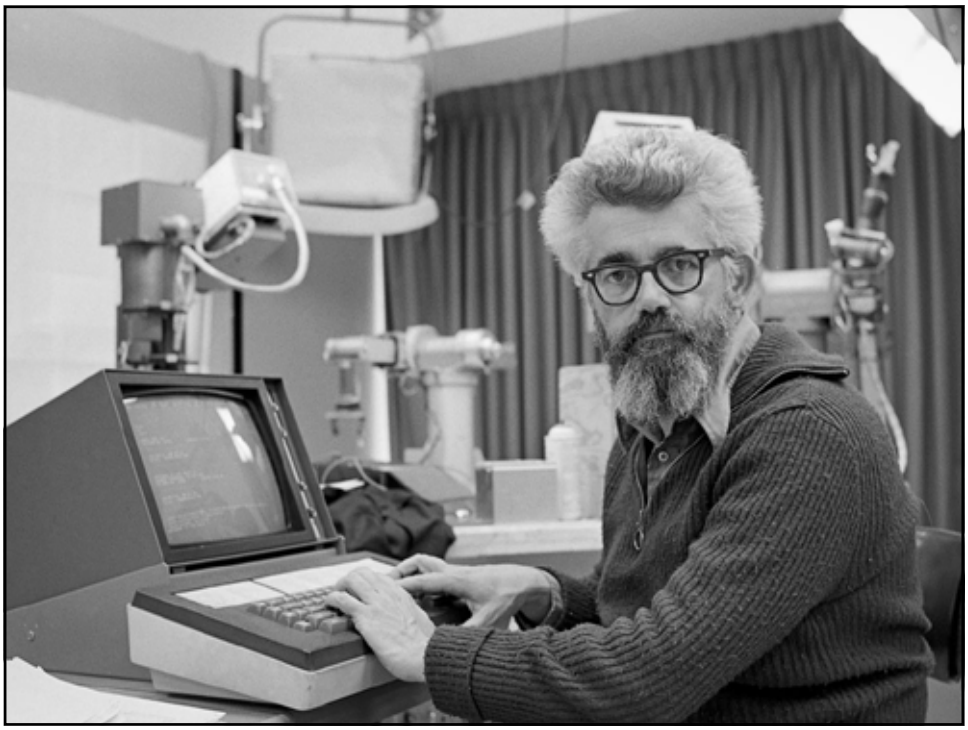

Weltanschauung

Lisp

10

Lisp

- 1958 bis heute
- John McCarthy
- Sprachfamilie
  - Common Lisp
  - Scheme
  - Emacs Lisps
- “Cutting Edge” Anwendungssysteme
  - NASA Pathfinder Mission-Planning
  - Hedge Fund Trading
  - Data Mining
  - Language Processing
  - ...



Any sufficiently complicated C or Fortran program contains an ad hoc, informally specified, bug-ridden, slow implementation of half of Common Lisp.

Philip Greenspun (http://philip.greenspun.com/research/)

13

cons, car, cdr (/ˈkʌdər/)

- S-Expressions (symbolic expressions)
- cons = Construct
  - (cons (cons a b) (cons c d)) = ((a b) (c d))
- car = “content of address register”
  - (car (cons x y)) == x
- cdr = “content of decrement register”
  - (cdr (cons x y)) == y

14

Clojure

15

Basics

- Dynamic Typing
- Functions as values
- Hosted on JVM

16

”Parallel Loop”

18

... und wieder Primzahlen

In [61]:
```
(def is_prime?
  (fn [v]
    (cond
      (= v 1) false
      (= v 2) true
      (= v 3) true
      (= (mod v 2) 0) false
      :else (loop [d 3]
              (cond (= (mod v d) 0) false
                    (> (* d d) v) true
                    :else (recur (+ d 2))
                    )
              )
      )
    )
  )
```

Out[61]: #'user/is_prime?

In [62]:
```
(is_prime? 11)
```

Out[62]: true

In [63]:
```
(is_prime? 42)
```

Out[63]: false

19

Viele Primzahlen

```
In [64]: (filter is_prime? (range 1 100))
Out[64]: (2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97)

In [65]: (count (filter is_prime? (range 1 100)))
Out[65]: 25

In [66]: (time (count (filter is_prime? (range 1 100))))
         "Elapsed time: 0.1812 msecs"
Out[66]: 25

In [67]: ; (time (count (filter is_prime? (range 1 20000000))))
```

The C version of this program needs approx. 7 seconds to count the number of primes between 1 and 20 million. Be careful when uncommenting the function above, because execution time can be 100 seconds and more.

Let's make it a function:

```
In [8]: (defn count_primes [a b]
             (count (filter is_prime? (range a b))))
Out[8]: #'user/count_primes

In [9]: (count_primes 1 100)
Out[9]: 25
```

Partitionierung

```
In [68]: (defn partition_range [a b p]
             (map
                 (fn [i] [(+ a (* i p)) (+ a (* (+ i 1) p))]) ; Create interval [a+i*p a+(i+1)*p]
                 (range 0 (+ (quot (- b a) p) 1))
             )
         )
Out[68]: #'user/partition_range

In [70]: (partition_range 1 100 10)
Out[70]: ([1 11] [11 21] [21 31] [31 41] [41 51] [51 61] [61 71] [71 81] [81 91] [91 101])

In [71]: (defn count_primes_in_interval [i] (apply count_primes i))
Out[71]: #'user/count_primes_in_interval

In [72]: (count_primes_in_interval [1 100])
Out[72]: 25
```

21

pmap

```
In [73]: (def n 10000000)
         (def p 1000)

Out[73]: #'user/p

In [74]: (time (reduce + (map count_primes_in_interval (partition_range 1 n p))))

         "Elapsed time: 34709.74235 msecs"

Out[74]: 664579

In [75]: (time (count (filter is_prime? (range 1 n))))

         "Elapsed time: 34737.840394 msecs"

Out[75]: 664579

In [76]: (time (reduce + (pmap count_primes_in_interval (partition_range 1 n p))))

         "Elapsed time: 7730.629128 msecs"

Out[76]: 664579
```

22

Future

23

Future

```
(ns future.core
  (:gen-class))

(defn -main
  "First steps using futures"
  [& args]

  (defn long_calculation [x]
    (Thread/sleep (* x 1000))
    (+ x 1)
    )

  (def f1
    (future (long_calculation 1)))
  (println "Future f1 defined")

  (def f2
    (future (long_calculation 5)))
  (println "Future f2 defined")

  (println @f2) ; blocks for approx. 5 seconds
  (println @f1) ; returns immediately because the result is available

  (shutdown-agents)
  )
```

24

Primzahlen zählen mittels Futures

```
In [78]: (def n_primes (future (count_primes 1 100)))
Out[78]: #'user/n_primes

In [81]: @n_primes ; or (deref n_primes)
Out[81]: 25

In [82]: (time (reduce
                (fn [s f] (+ s @f))
                0
                (map
                 (fn [i] (future (count_primes_in_interval i)))
                 (partition_range 1 n p))))
         "Elapsed time: 7683.091572 msecs"
Out[82]: 664579
```

25

Promise

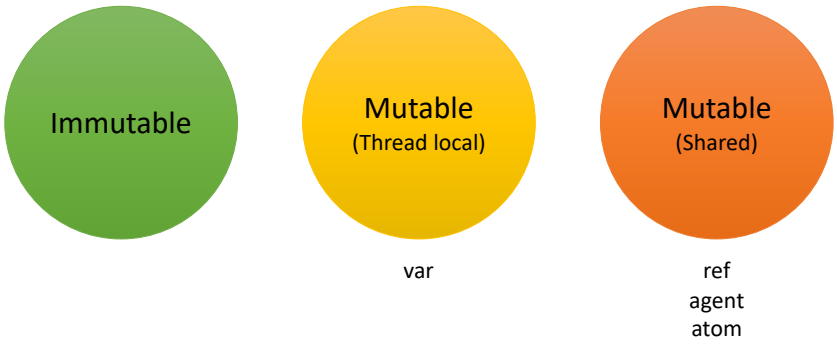
26

Promise

```
1  (ns promise.core
2    (:gen-class))
3
4  (defn -main
5    "A simple example using promises"
6    [& args]
7
8    (def p (promise))
9
10   ; another thread will deliver the promise ...
11
12   (future
13     (Thread/sleep 2000)
14     (deliver p 42) ; ... excatly after 2 seconds
15   )
16
17   (println "Additional thread created: ")
18   (println "Waiting for the promise to be delivered")
19   (println @p)
20
21   (shutdown-agents)
22 )
23
```

27

State or no State?



var

ref
agent
atom

28

Atom

29

Ref

30

Helper

```
3
4  (defn join_threads [ts]
5    (doall (map (fn [t] (deref t)) ts))
6    )
7
8  (defn do_the_work [n_threads n_loops f]
9    (defn main [] (dotimes [l n_loops] (f)))
10   (def threads (map (fn [i] (future (main))) (range 1 (+ n_threads 1))))
11   (join_threads threads)
12   )
```

31

Ref

```
40  (println "Using refs for up and down")
41  (def ref_up (ref 0))
42  (def ref_down (ref 0))
43  (def ref_inconsistencies (atom 0))
44
45  (defn ref_up_and_down []
46    (dosync
47     (if (not= (+ @ref_up @ref_down) 0) (swap! ref_inconsistencies + 1))
48     (alter ref_up + 1)
49     (alter ref_down + -1)))
50
51  (time (do_the_work n_threads n_loops ref_up_and_down))
52
53  (println "Finished with refs:")
54  (println @ref_up)
55  (println @ref_down)
56  (print "Number of inconsistencies: ")
57  (println @ref_inconsistencies)
58
```

32

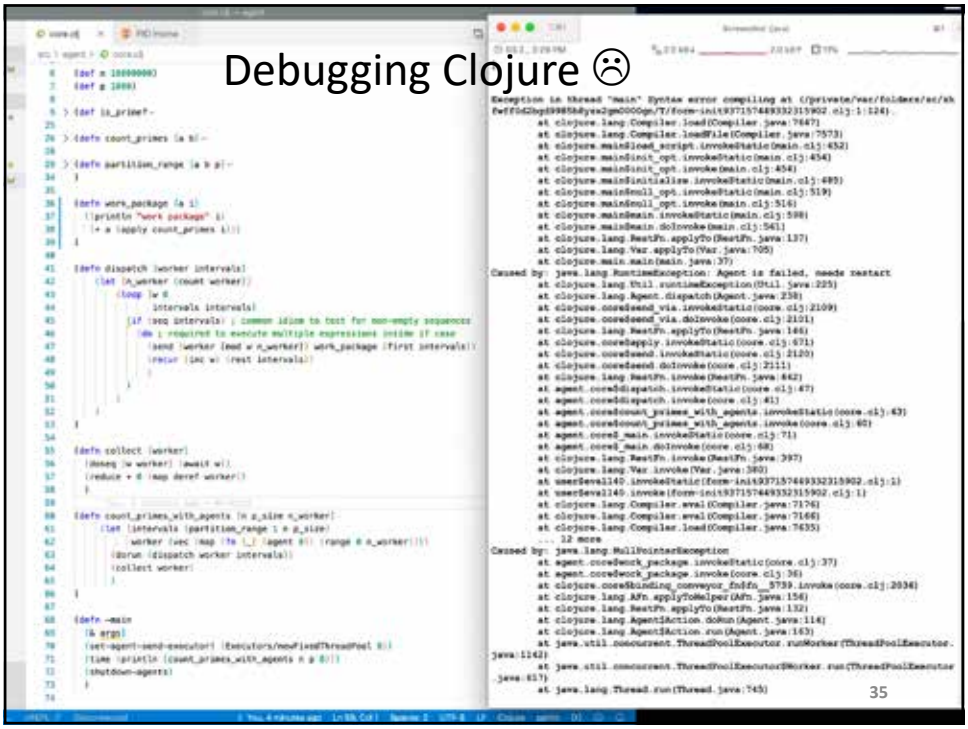
Atom

```
21  (println "Using atoms for up and down")
22  (def atom_up (atom 0))
23  (def atom_down (atom 0))
24  (def atom_inconsistencies (atom 0))
25
26  (defn atom_up_and_down []
27    ; (dosync ... ) does not help
28    (if (not= (+ @atom_up @atom_down) 0) (swap! atom_inconsistencies + 1))
29    (swap! atom_up + 1)
30    (swap! atom_down + -1))
31
32  (time (do_the_work n_threads n_loops atom_up_and_down))
33
34  (println "Finished with atoms:")
35  (println @atom_up)
36  (println @atom_down)
37  (print "Number of inconsistencies: ")
38  (println @atom_inconsistencies)
39
```

33

Agent

34



35

In [47]:
```
(defn work_package [a i] (+ a (apply count_primes i)))
```
Out[47]:
```
#'user/work_package
```

In [48]:
```
(defn dispatch [worker intervals]
    (let [n_worker (count worker)]
        (loop [w 0
               intervals intervals]
          (if (seq intervals) ; common idiom to test for non-empty sequences
            (do ; required to execute mltiple expressions inside if case
              (send (worker (mod w n_worker)) work_package (first intervals))
              (recur (inc w) (rest intervals))
              )
            )
          )
        )
    )
)
```
Out[48]:
```
#'user/dispatch
```

In [49]:
```
(defn collect [worker]
  (doseq [w worker] (await w))
  (reduce + 0 (map deref worker))
  )
```
Out[49]:
```
#'user/collect
```

In [50]:
```
(defn count_primes_with_agents [n p_size n_worker]
     (let [intervals (partition_range 1 n p_size)
           worker (vec (map (fn [_] (agent 0)) (range 0 n_worker)))]
       (dorun (dispatch worker intervals))
       (collect worker)
       )
)
```
Out[50]:
```
#'user/count_primes_with_agents
```

In [53]:
```
(time (count_primes_with_agents n p 8))
```
```
"Elapsed time: 7509.989129 msecs"
```
Out[53]:
```
664579
```

36

Links

- Clojure Rationale
  https://clojure.org/about/rationale
- Identity and State in Clojure
  https://clojure.org/about/state

38

Begleitliteratur