Java Programcıları için Clojure'a Giriş*

JUG Ankara
Ekim 2014

Üstün Özgür

* Bu sunumda Rich Hickey'nin özgün Clojure for Java Programmers sunumu temel alınmıştır.

Clojure Felsefesi

- Java Virtual Machine üzerinde çalışan dinamik bir dil

- Derlenen (compiled) bir dil

- Bir Lisp dialekti. Lisp'in veri olarak kod (code-as-data) felsefesi ve makro desteği.

- Fonksiyonel programlama ağırlıklı. Immutable, persistent veri yapıları. Mutable state için transactional hafıza sistemi

Tarihçe

- 2007'de Rich Hickey tarafından açıklandı
- JVM ve CLR üzerinde başladı, 2011'de ClojureScript açıklandı
- Ekim 2012'de Thoughtworks Technology Radar'ında Adopt fazına alındı.
- Ana kullanıcı şirketler: Walmart Labs (Runa), Cognitect, Thoughtworks, Backtype (Twitter tarafından satın alındı, Storm ve Cascalog'un yaratıcısı)
- Hickey'nin "Simple Made Easy", "Are We There Yet?", "Hammock Driven Development" konuşmaları Clojure'dan bağımsız olarak programlama açısından büyük önem taşıyan konuşmalardır.

- Temeller
- Sözdizim (syntax)
- Diziler (Sequences)
- Java entegrasyonu

Clojure Temelleri

- Dinamik
- Fonksiyonel
  - Değişmeyen veri yapıları
- JVM üzerinde çalışan
- Açık kaynak

Dinamik dillerin avantajları

- Daha esnek
- Deneylere daha açık: deneme yanılma
- Daha kısa ve öz
- Problem odaklı

Dinamik diller

- Başka dillerin JVM portları
  - JRuby, Jython
- JVM için yazılmış diller
  - Groovy, Clojure

Neden Clojure?

- Kısa ve öz programlar, Lisp
- Java'ya yakın performans ve kolay Java entegrasyonu
- Fonksiyonel programlama ve concurrency

Dinamik Geliştirme

- REPL - Read Eval Print Loop
- Kodu anında tanımlama ve deneme
- İnteraktif ortam

Temel Veri Türleri

- Sayılar (tam, ondalıklı, kesirli) **123 1.23 3/4**
- String **"Ankara"** Karakter **\a \b**
- Semboller **ahmet mehmet**
- Keywordler **:ahmet :mehmet**
- Boolean **true false nil**
- Regex **#"\d{3}" #"ab?"**

Veri Yapıları

- List: linked list, öne eklenir
  - (3 4 5) (ahmet mehmet) (list 1 2 3)
- Vector: random access, sona eklenir
  - [3 4 5] [ahmet mehmet]
- Maps : key-value, dictionary
  - {:ad "Ustun" :soyad "Ozgur"}
  - {6 "Ankara" 34 "Istanbul"}
- Kümeler
  - #{"Besiktas" "Fenerbahce" "Galatasaray"}

Veri Yapıları

- Bütün bu yapılar iç içe geçebilir:
  - {:ad "Ustun" :soyad "Ozgur"
    :hobileri #{"Programlama" "Clojure"}
    :bildigi-diller ["Python" "JavaScript" "Clojure"]}

  - [{:id 1 :isim "Besiktas" :kadro [...]}
    {:id 2 :isim "Fenerbahce" :kadro [...]}

Syntax kuralları

- (operator ....)
- (+ 3 4 5)
- (* 3 4 5)
- (def x 3)

Operator türleri

- Fonksiyon
- Makro
- Özel form

Özel Formlar

- Argümanların evaluation'ı özel kurallara tabi
- **(def isim değer) ; global değişken**
- **(if condition first-rule second-rule)**
  - sadece birini evaluate eder.
- Çok az sayıda özel form
- **fn let loop recur do new . throw try set! quote var**

Makrolar

- Sözdizimsel şeker (Syntax sugar) eklemek için
- Kullanıcı tarafından eklenebilir
- Çoğu dilde syntax olarak tanımlanması gereken yapılar Lisplerde macrolar ile kullanıcılar tarafından eklenebilir.
- Makro fonksiyonuna argumanlar veri olarak girer, makro bunu işler ve yeni bir veri yapısı döndürür
- (if-not condition first-expr second-expr)
- (if (not condition) first-expr second-expr)
- Aslında çok basit. HTML template oluşturmak gibi.

Fonksiyonlar

- Birinci sınıf, aynı sayılar ve stringler gibi
  - **(def dort 4)**
  - **(def square (fn [x] (* x x))**
  - **(square 4) ;; 16**
- Mapler ve keywordler de birer fonksiyon!
  - **(def ustun {:name "Ustun" :surname "Ozgur"}**
  - **(ustun :name) ;; "Ustun"**
  - **(:surname ustun) ;; "Ozgur"**
- Aslinda IFn interfaceini implemente eden her şey bir fonksiyon

Syntax Karşılaştırması

Java	Clojure
int i = 5;	(def i 5)

if (i == 0) {
  return y;
} else {
  return z;
}
	(if (zero? i)
	  y
	  z)

x * y * z	(* x y z)
foo(x, y, z)	(foo x y z)
foo.bar(x)	(. foo bar x)

Sequence'lar

- Normal Lisp'ler linked liste üzerine kuruludur.
- Sequence, Clojure'un buna getirdiği bir abstraction, aslında Java'daki ISeq interface'i
- (seq list)
- (first list)
- (rest list)

Zengin Sequence Fonksiyonları

```
(drop 2 [1 2 3 4 5]) -> (3 4 5)
(take 9 (cycle [1 2 3 4]))
-> (1 2 3 4 1 2 3 4 1)
(interleave [:a :b :c :d :e] [1 2 3 4 5])
-> (:a 1 :b 2 :c 3 :d 4 :e 5)
(partition 3 [1 2 3 4 5 6 7 8 9])
-> ((1 2 3) (4 5 6) (7 8 9))
(map vector [:a :b :c :d :e] [1 2 3 4 5])
-> ([:a 1] [:b 2] [:c 3] [:d 4] [:e 5])
(apply str (interpose \, "asdf"))
-> "a,s,d,f"

(reduce + (range 100)) -> 4950
```

Java Entegrasyonu

- (. Math PI)
- (.. System getProperties (get "java.version"))
- (new java.util.Date)

Swing örneği

Demo

Fonksiyonel Programlama

- 3 Temel Operasyon
- Map: Dizinin eleman sayısı değişmez. Dönüştürme
- Filter: Dizinin eleman sayısı azalır. Filtreleme
- Reduce: Dizi tek sayıya indirgenir. Özet çıkarma

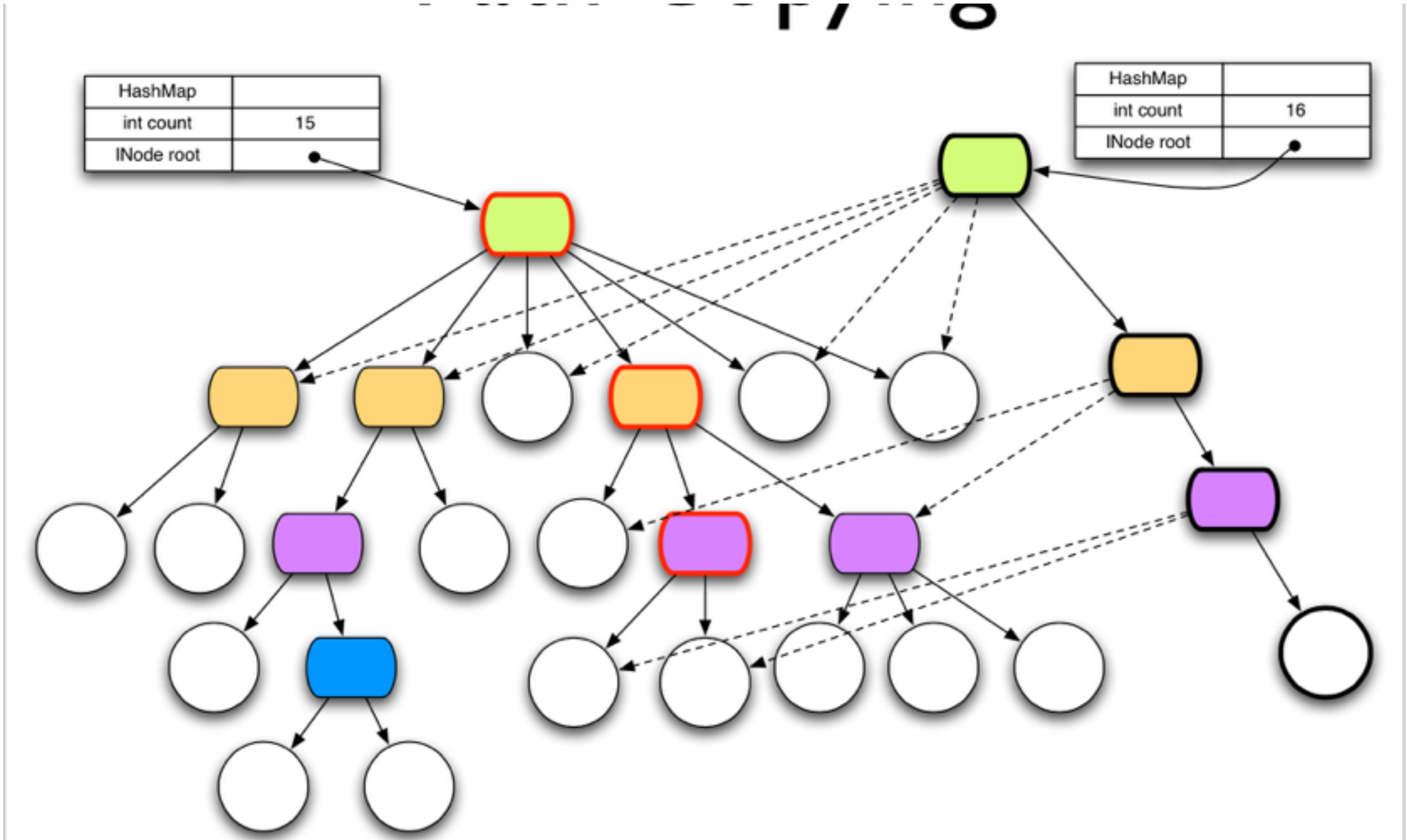
Neden fonksiyonel programlama?

- State ne kadar az olursa o kadar iyi
- Daha az hata
- Daha hızlı geliştirme
- Daha çok kod reuse

Neden popüler değildi?

- Bilgisayarların yavaşlığı
- von Neumann tarzı update in place geleneği
- Persistent data structureların bilinmemesi

Persistent data structure ve veri paylaşımı

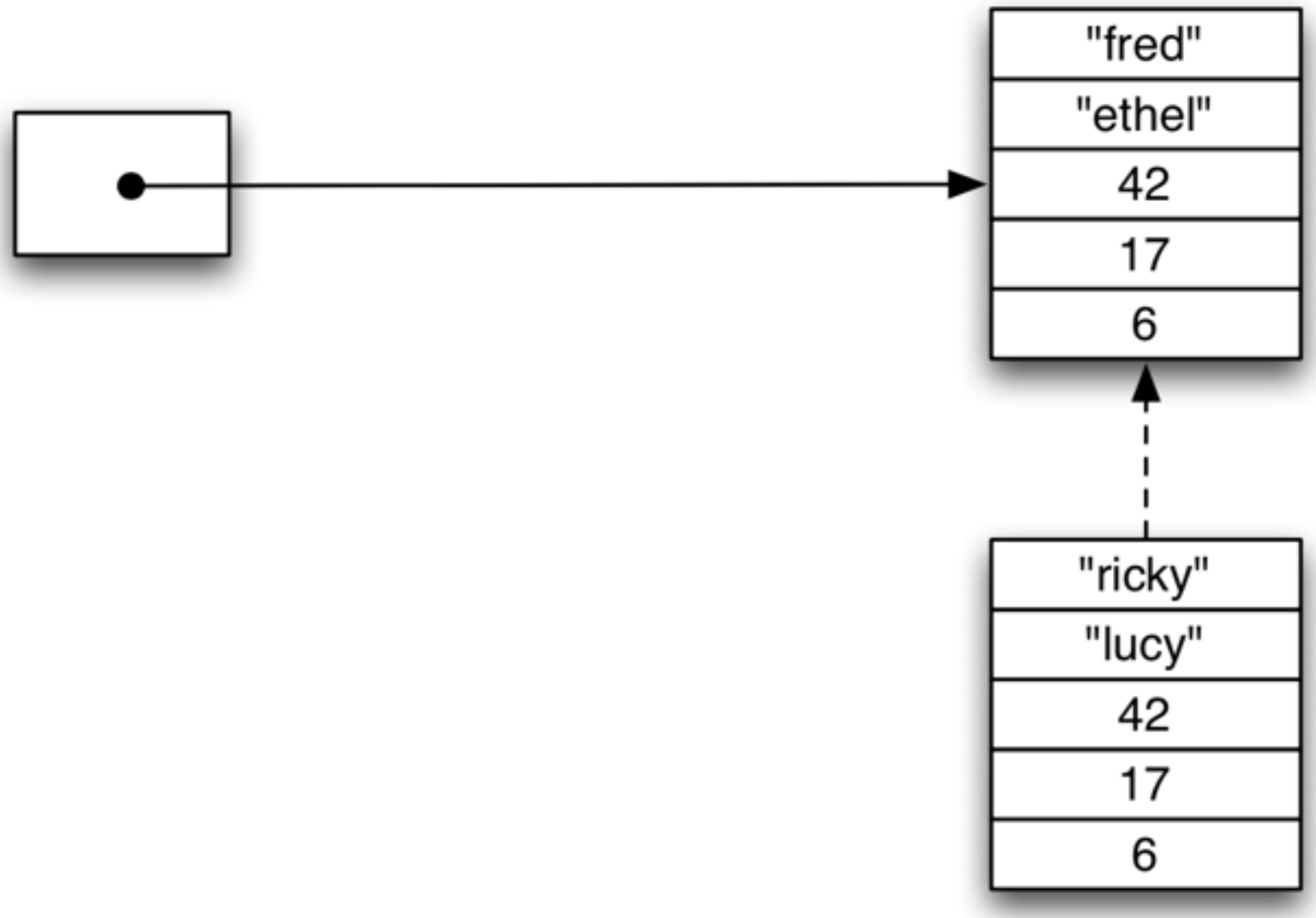


Varsayılan olarak Immutability

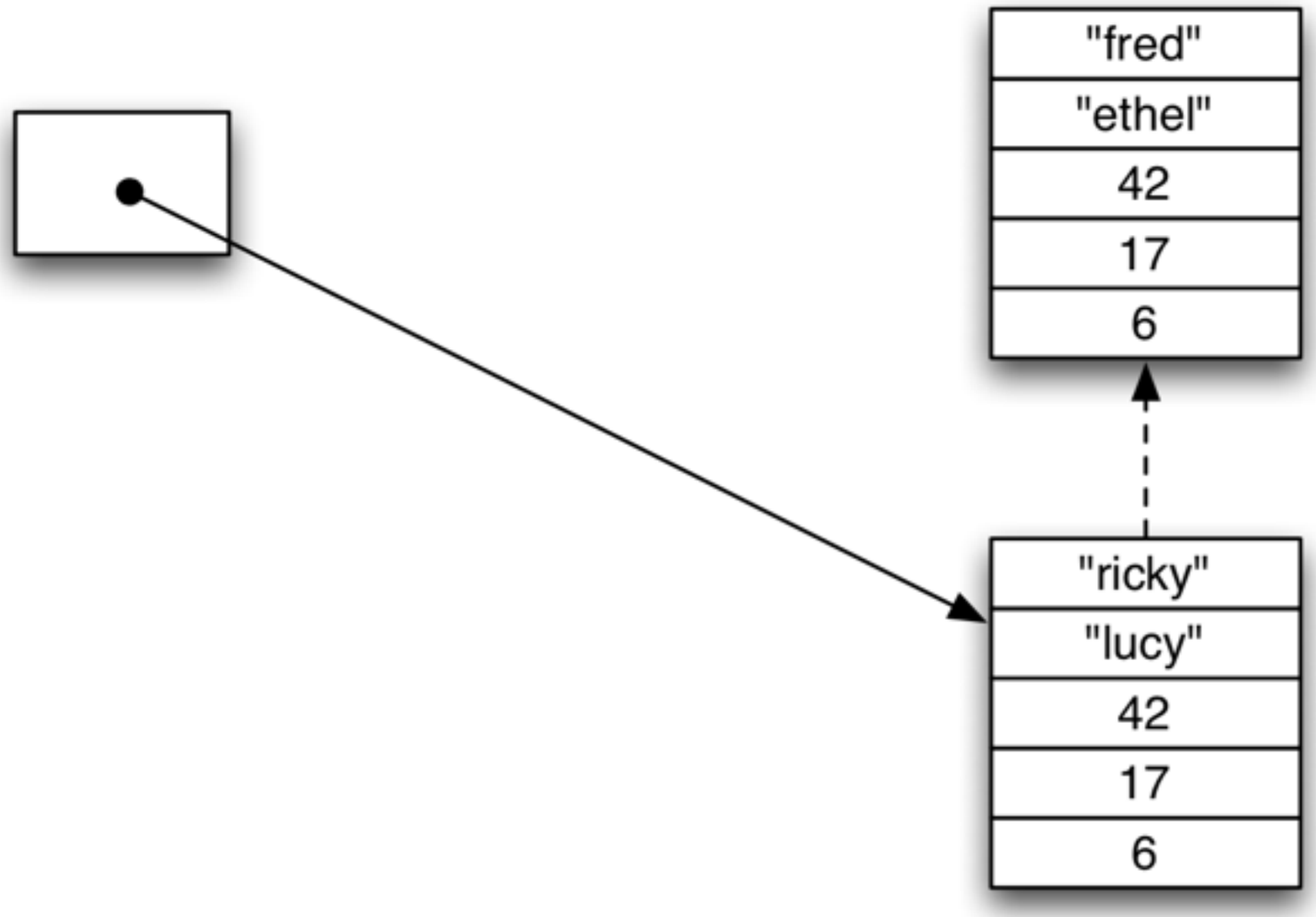
- Değer değişimleri kontrollü olmalı
- Bunun için 3 temel yapı
  - Atom
  - Ref
  - Agent

Atom

- (def x (atom 0))
- (swap! x inc)
- (reset! x 10)
- (def x (atom {:ad "Ustun" :yas 29})
- (swap! x update-in :yas 30)


"fred"
"ethel"
42
17
6
"ricky"
"lucy"
42
17
6

Atomic Güncelleme



Java Interfaceleri ve Protokoller

Demo

Daha birçok özellik

Metadata

Recursive functional looping

Destructuring binding in *let/fn/loop*

Python Haskell tarzı List comprehensions (*for*)

Relational set algebra

Multimethods

Parallel computation

Namespaces, zippers, XML

Clojure

- Fonksiyonel
- Immutable veri yapıları
- Veri odaklı dinamik programlama
- Kolay JVM entegrasyonu

Teşekkürler

- Sorular?

- Üstün Özgür
- ustun@ustunozgur.com
- @ustunozgur