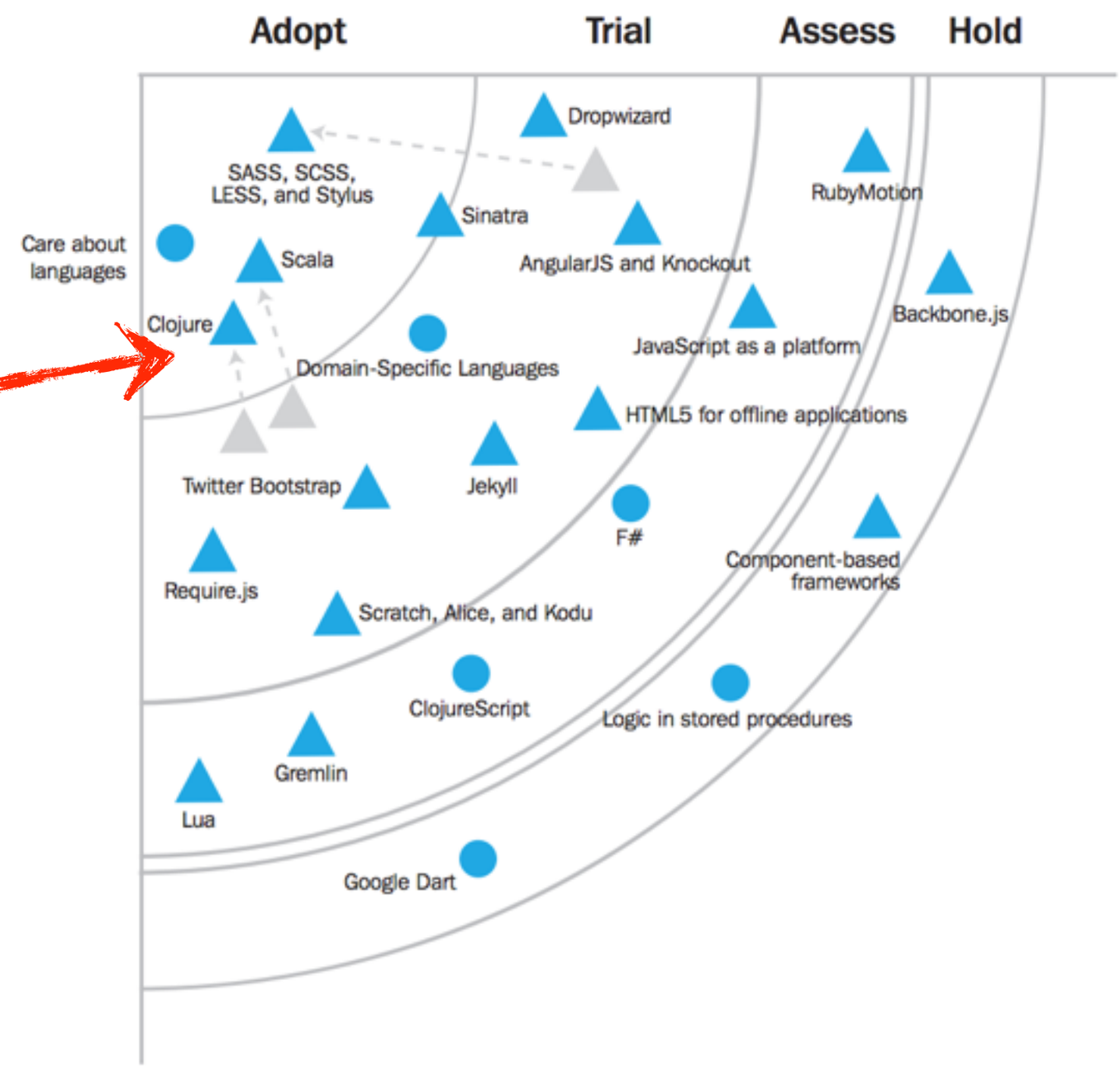
Simple by Design: Clojure

@stuarthalloway
stu@cognitect.com

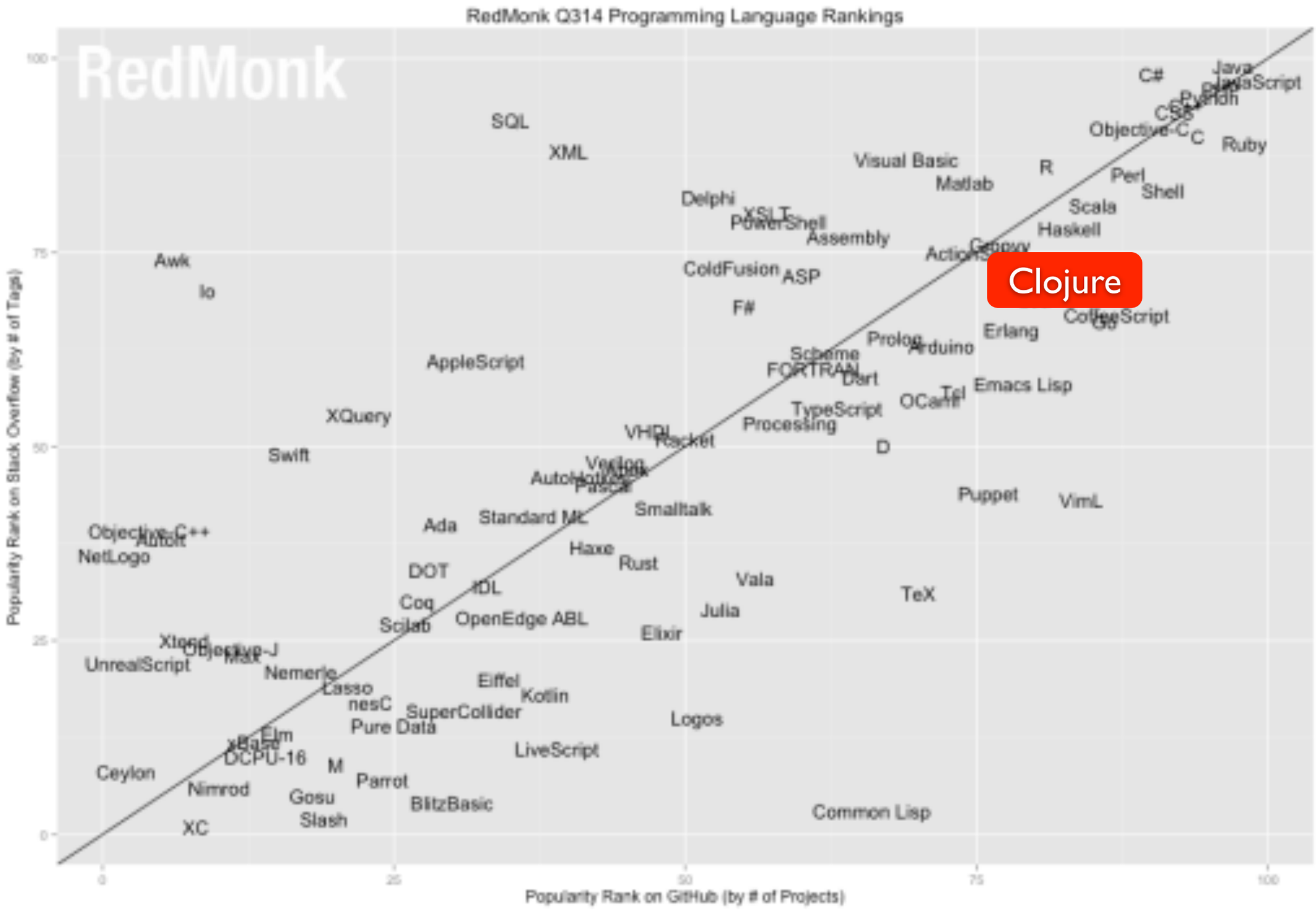
Thoughtworks Radar

Oct 2012:
Adopt Clojure



http://www.thoughtworks.com/radar

Redmonk Top 20



http://redmonk.com/sogrady/2014/06/13/language-rankings-6-14/

Design

specification of an artifact using components to meet goals subject to constraints

Simple

not compound

Valuable But Not Simple

convenience

beginner-friendliness

ease

familiarity

smaller size

lesser count

Simple

not compound

Agenda

examples of simplicity in Clojure

benefits of simplicity in Clojure

producing Clojure

Examples of Simplicity

syntax

protocols

values and references

Examples of Simplicity

syntax

protocols

values and references

"Hello World"

"Hello World"

that *is* the program

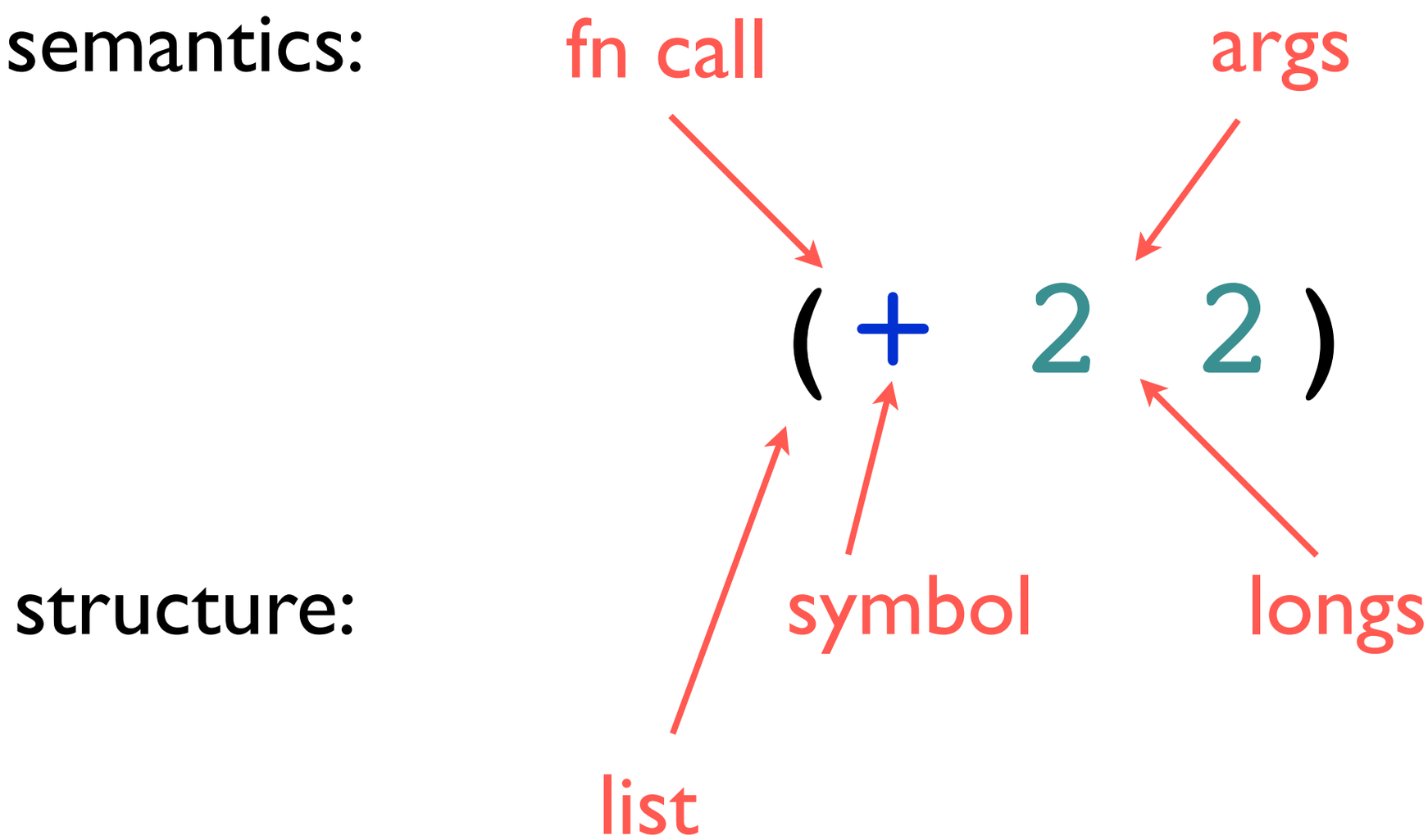
Everything is Data

```
{ :firstName "John"
  :lastName "Smith"
  :age 25
  :address {
    :streetAddress "21 2nd Street"
    :city "New York"
    :state "NY"
    :postalCode "10021" }
  :phoneNumber
    [ {:type "name" :number "212 555-1234"}
      {:type "fax" :number "646 555-4567" } ] }
```

type	examples
string	`"foo"`
character	`\f`
integer	`42, 42N`
floating point	`3.14, 3.14M`
boolean	`true`
nil	`nil`
symbol	`foo, +`
keyword	`:foo, ::foo`

type	properties	examples
list	sequential	`(1 2 3)`
vector	sequential and random access	`[1 2 3]`
map	associative	`{:a 100` `:b 90}`
set	membership	`#{:a :b}`

Function Call



Function Definition

define a fn

fn name

docstring

```
(defn greet
  "Returns a friendly greeting"
  [your-name]
  (str "Hello, " your-name))
```

arguments

fn body

...Still Just Data

symbol symbol string

```
(defn greet
  "Returns a friendly greeting"
  [your-name]
  (str "Hello, " your-name))
```

vector

list

Complexities Avoided

lots of syntax to learn

ordering dependencies

operator precedence

code over data bias

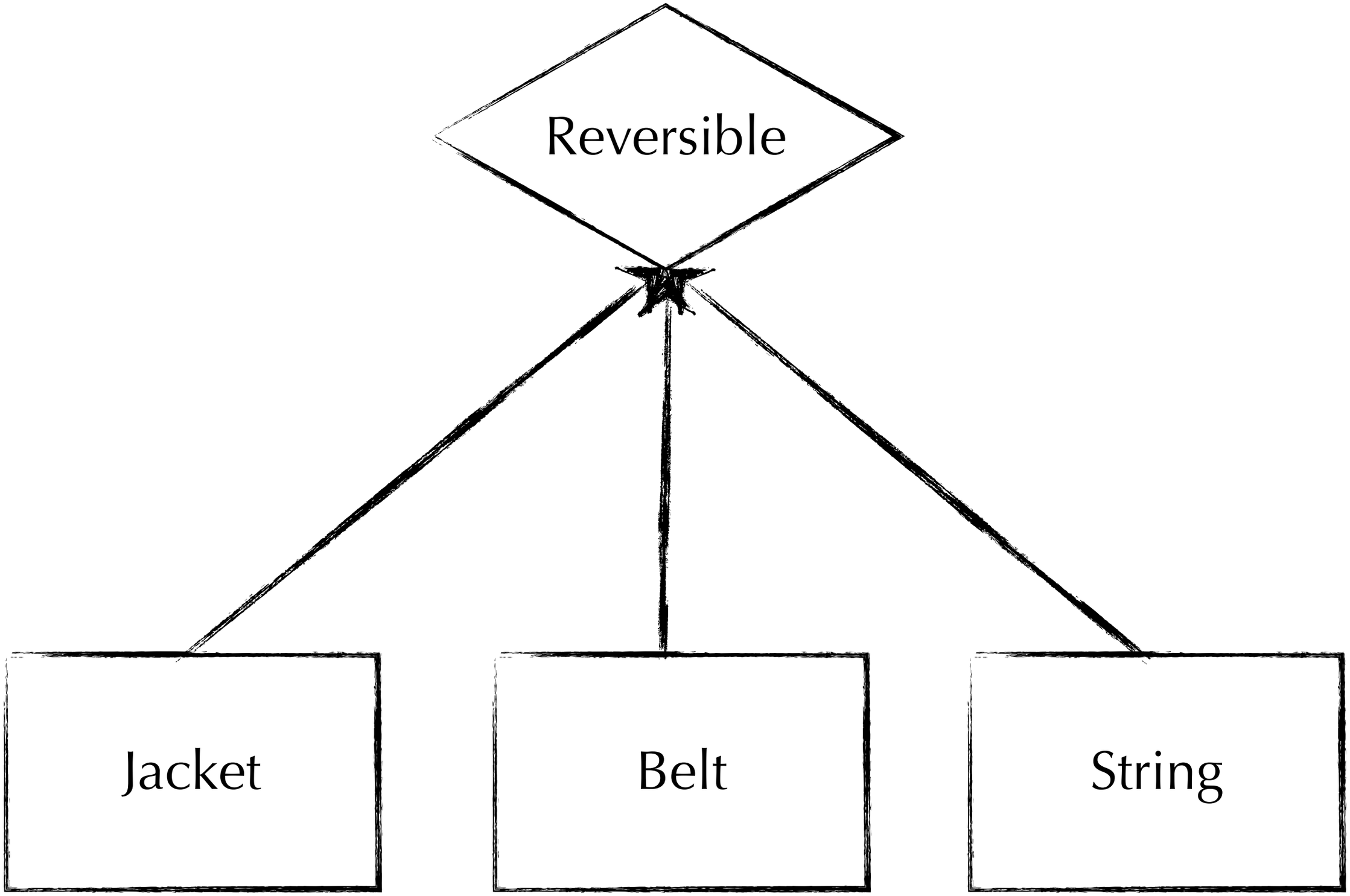
tedious metaprogramming

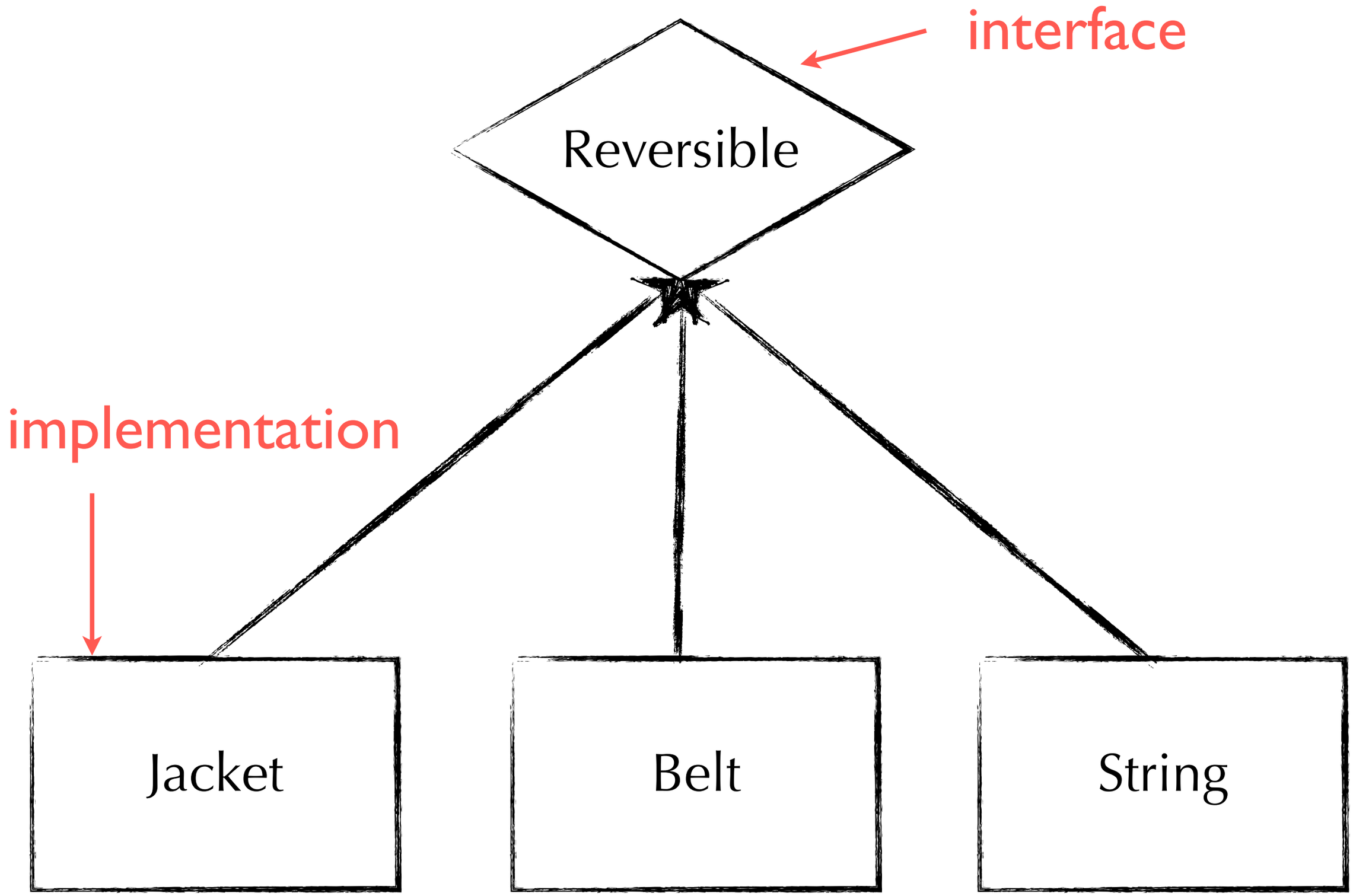
Examples of Simplicity

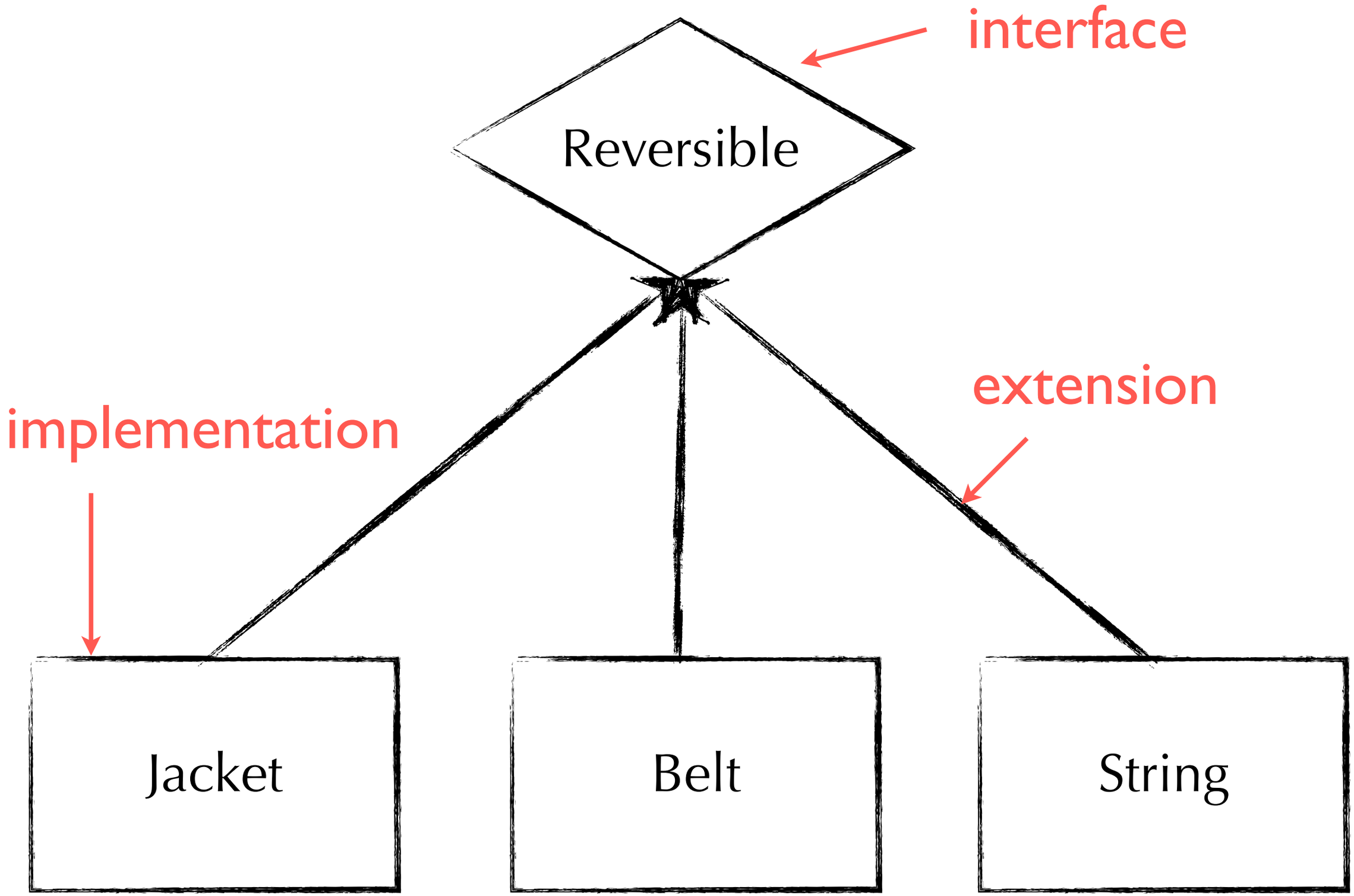
syntax

protocols

values and references


Reversible
Jacket
Belt
String


interface
Reversible
implementation
Jacket
Belt
String


interface
Reversible
extension
implementation
Jacket
Belt
String

interface

```
(defprotocol Reversible
  (reverse [_]))
```

implementation

```
(defrecord ReversibleTie [a b])
```

```
(extend-protocol Reversible
  ReversibleTie
  (reverse [tie] (->ReversibleTie (:b tie) (:a tie)))
  String
  (reverse [s] (-> s
                   StringBuilder.
                   .reverse
                   .toString)))
```

extension

Complexities Avoided

adapter pattern

wrapper pattern

translator pattern

monkey patching

StringUtils

note that these are all combinatorial

Examples of Simplicity

syntax

protocols

values and references

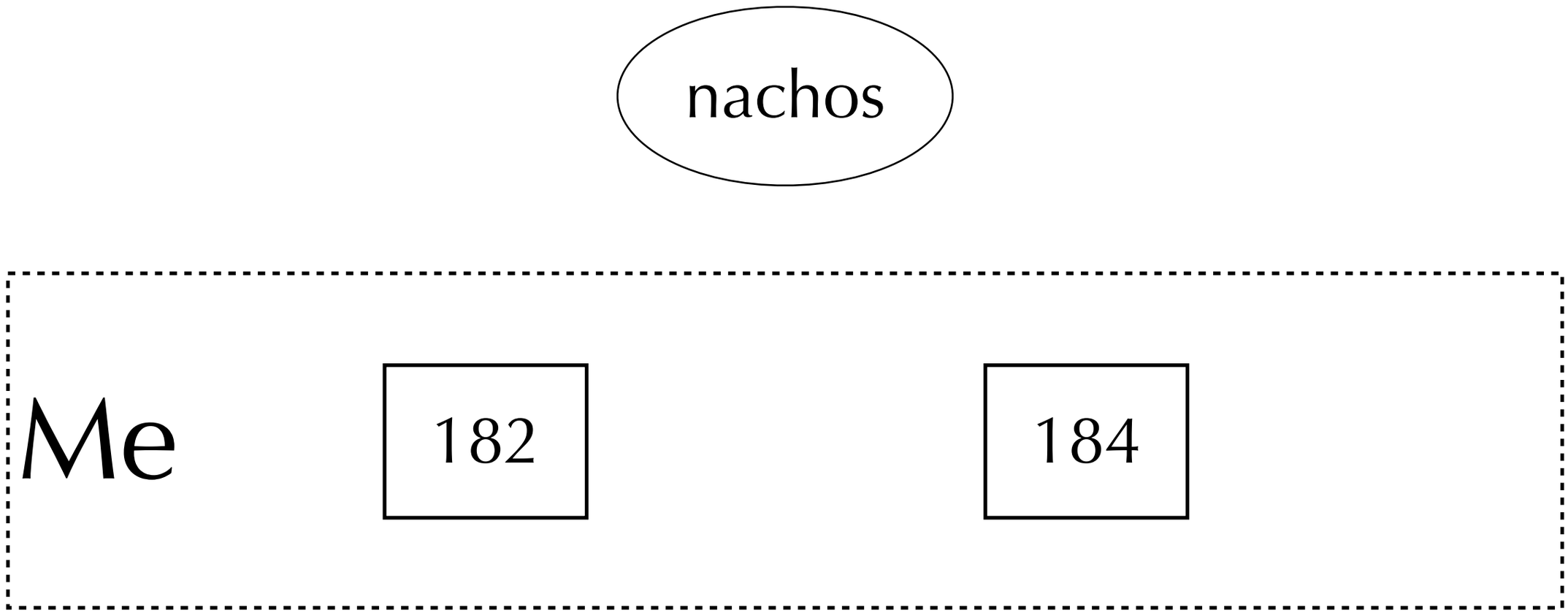
Me

182

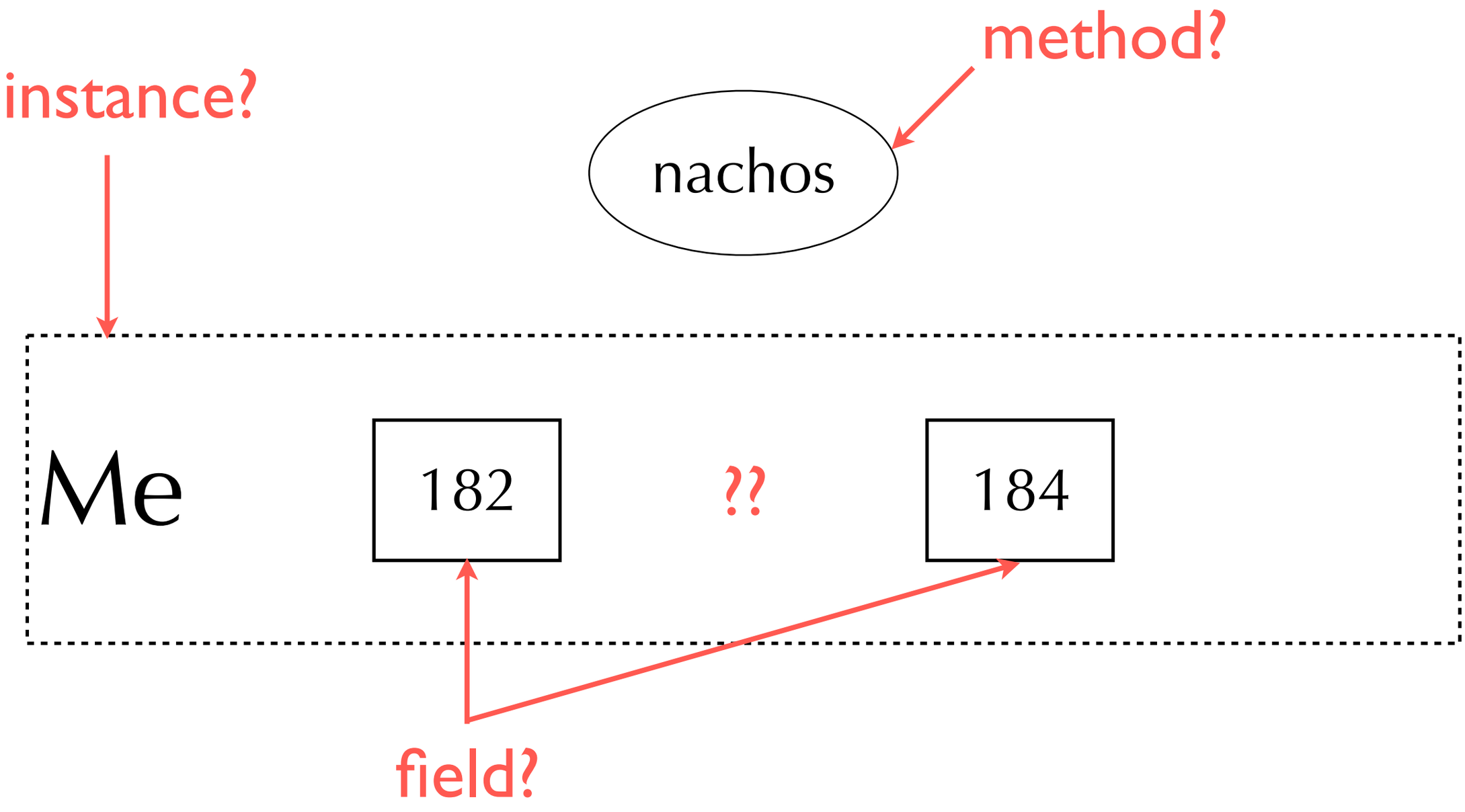
nachos

Me

182

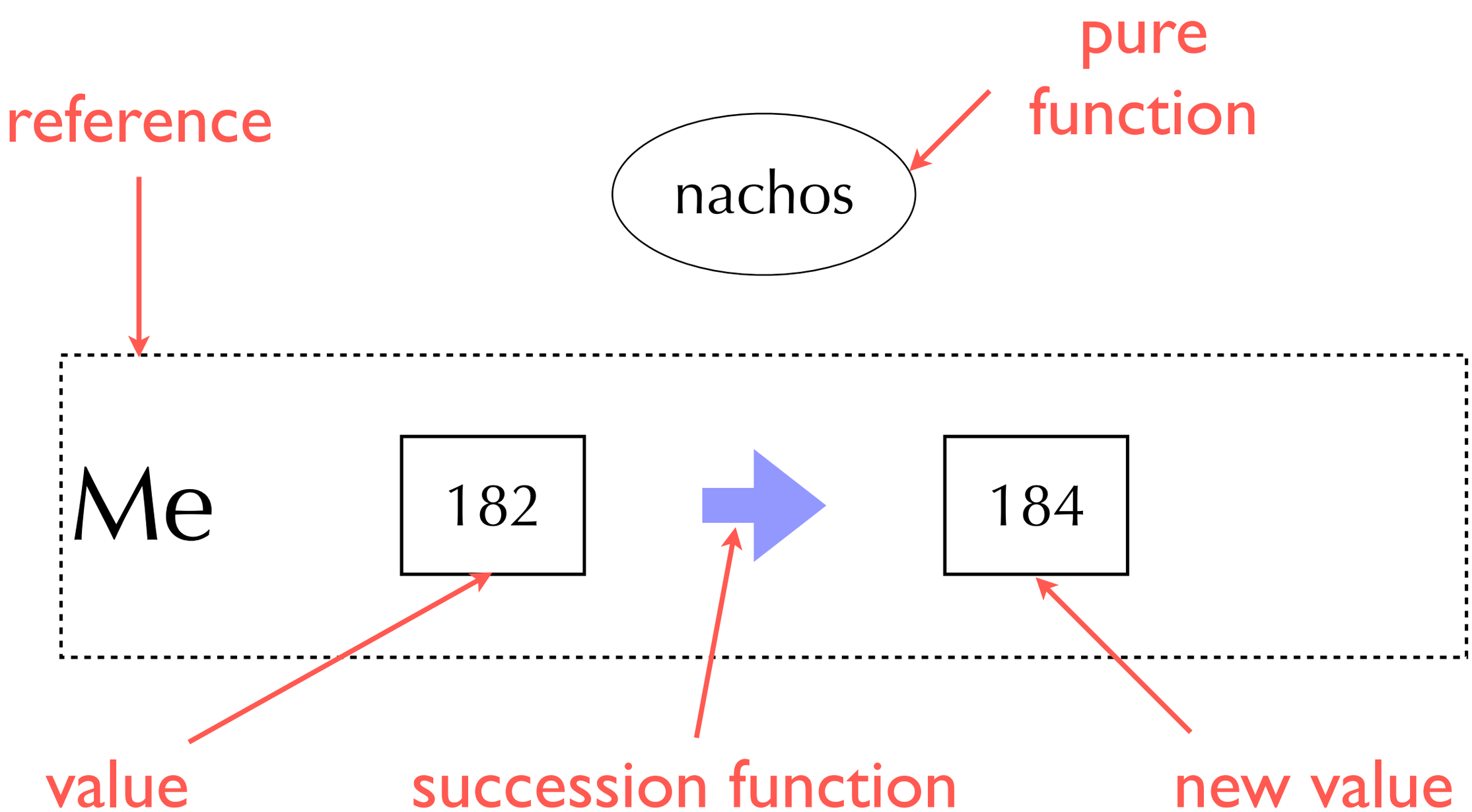

nachos
Me
182
184

Watch OO Flounder



If you have more things than names, your design is broken.

Clojure's Simplicity



Values and References

```
(defprotocol Nachos
  (yum [_] "eat some nachos"))

(defrecord Person [name lbs]
  Nachos
  (yum [person]
       (update-in person [:lbs] + 2)))

(def me (atom (->Person "Stu" 182)))

(def me-before @me)

(swap! me yum)

(def me-after @me)
```

Values and References

functional

```
(defprotocol Nachos
  (yum [_] "eat some nachos"))

(defrecord Person [name lbs]
  Nachos
  (yum [person]
       (update-in person [:lbs] + 2)))
```

```
(def me (atom (->Person "Stu" 182)))

(def me-before @me)

(swap! me yum)

(def me-after @me)
```

Values and References

```
(defprotocol Nachos
  (yum [_] "eat some nachos"))

(defrecord Person [name lbs]
  Nachos
  (yum [person]
       (update-in person [:lbs] + 2)))

(def me (atom (->Person "Stu" 182)))

(def me-before @me)

(swap! me yum)

(def me-after @me)
```

update semantics

Values and References

```
(defprotocol Nachos
  (yum [_] "eat some nachos"))

(defrecord Person [name lbs]
  Nachos
  (yum [person]
       (update-in person [:lbs] + 2)))

(def me (atom (->Person "Stu" 182)))

(def me-before @me)

(swap! me yum)

(def me-after @me)
```

multiple point-in-time values

Complexities Avoided

incidental ~~complexity~~ temporal reasoning

single-threading

locking

defensive copying

setter methods

String vs. StringBuilder vs. StringBuffer

*note **(again!)** that these are all combinatorial*

Benefits of Clojure

concision

generality

robustness

agility

Benefits of Clojure

concision

generality

robustness

agility

StringUtils
indexOfAny

https://commons.apache.org/proper/commons-lang/javadocs/api-2.6/org/apache/commons/lang/StringUtils.html

indexOfAny Spec

```
StringUtils.indexOfAny(null, *)                    = -1
StringUtils.indexOfAny("", *)                      = -1
StringUtils.indexOfAny(*, null)                    = -1
StringUtils.indexOfAny(*, [])                      = -1
StringUtils.indexOfAny("zzabyycdxx",['z','a']) = 0
StringUtils.indexOfAny("zzabyycdxx",['b','y']) = 3
StringUtils.indexOfAny("aba", ['z'])               = -1
```

indexOfAny Impl

```
// From Apache Commons Lang, http://commons.apache.org/lang/
public static int indexOfAny(String str, char[] searchChars) {
  if (isEmpty(str) || ArrayUtils.isEmpty(searchChars)) {
    return -1;
  }
  for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    for (int j = 0; j < searchChars.length; j++) {
      if (searchChars[j] == ch) {
        return i;
      }
    }
  }
  return -1;
}
```

- Corner Cases

```
public static int indexOfAny(String str, char[] searchChars) {
  when (searchChars)
    for (int i = 0; i < str.length(); i++) {
      char ch = str.charAt(i);
      for (int j = 0; j < searchChars.length; j++) {
        if (searchChars[j] == ch) {
          return i;
        }
      }
    }
  }
}
```

- Type Decls

```
indexOfAny(str, searchChars) {
  when (searchChars)
    for (i = 0; i < str.length(); i++) {
      ch = str.charAt(i);
      for (j = 0; j < searchChars.length; j++) {
        if (searchChars[j] == ch) {
          return i;
        }
      }
    }
  }
}
```

+ When Clause

```
indexOfAny(str, searchChars) {
  when (searchChars)
    for (i = 0; i < str.length(); i++) {
      ch = str.charAt(i);
      when searchChars(ch) i;
    }
  }
}
```

+ Comprehension

```
indexOfAny(str, searchChars) {
  when (searchChars)
    for ([i, ch] in indexed(str)) {
      when searchChars(ch) i;
    }
  }
}
```

Lispify

```
(defn index-filter [pred coll]
  (when pred
    (for [[idx elt] (indexed coll) :when (pred elt)] idx)))
```

Benefits of Clojure

concision

generality

robustness

agility

imperative	functional
searches strings	searches any sequence
matches characters	matches any predicate
returns first match	returns lazy seq of all matches

+ Generality

```
; idxs of heads in stream of coin flips
(index-filter #{:h}
[:t :t :h :t :h :t :t :t :h :h])
-> (2 4 8 9)

; Fibonaccis pass 1000 at n=17
(first
  (index-filter #(> % 1000) (fibo)))
-> 17
```

Clojure
programs can have fewer
lines of code than OO
programs have files

Benefits of Clojure

concision

generality

robustness

agility

	imperative	**functional**
functions	1	1
classes	1	0
internal exit points	2	0
variables	3	0
branches	4	0
boolean ops	1	0
function calls*	6	3
total	***18***	***4***

Benefits of Clojure

concision

generality

robustness

agility

Plain Immutable Collection Objects (PICOs)

PICOS Everywhere

collections

directories

files

XML

JSON

result sets

web requests

web responses

sessions

configuration

metrics

logs

Consuming JSON

What actors are in more than one movie currently topping the box office charts?

http://developer.rottentomatoes.com/docs/read/json/v10/Box_Office_Movies

Consuming JSON

```
find the JSON input
download it
parse json
walk the movies
accumulating cast
extract actor name
get frequencies
sort by highest frequency
```

http://developer.rottentomatoes.com/docs/read/json/v10/Box_Office_Movies

Consuming JSON

```
(->> box-office-uri
     slurp
     json/read-json
     :movies
     (mapcat :abridged_cast)
     (map :name)
     frequencies
     (sort-by (comp - second)))
```

http://developer.rottentomatoes.com/docs/read/json/v10/Box_Office_Movies

Consuming JSON

```
["Shiloh Fernandez" 2]
["Ray Liotta" 2]
["Isla Fisher" 2]
["Bradley Cooper" 2]
["Dwayne \"The Rock\" Johnson" 2]
["Morgan Freeman" 2]
["Michael Shannon" 2]
["Joel Edgerton" 2]
["Susan Sarandon" 2]
["Leonardo DiCaprio" 2]
```

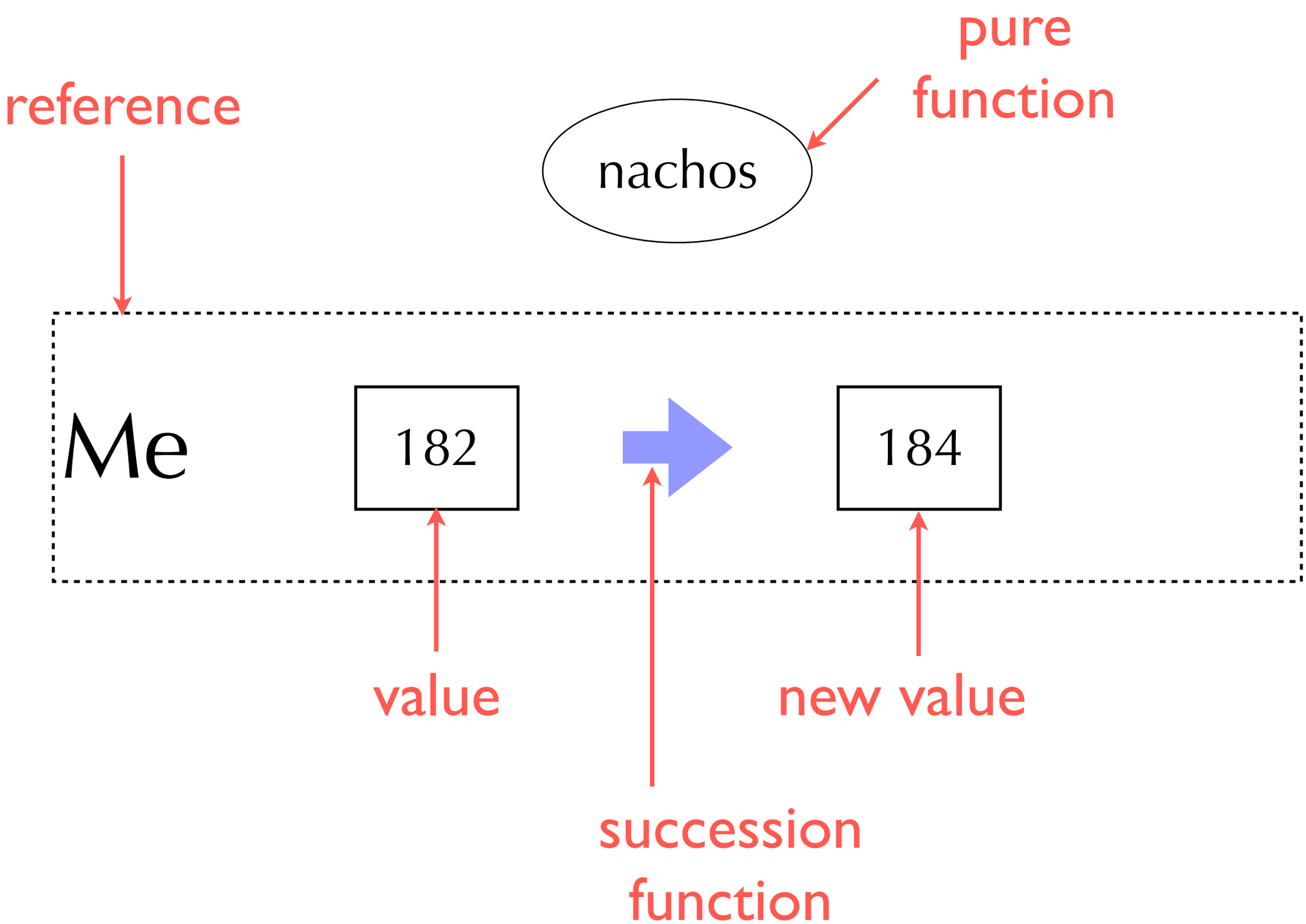
http://developer.rottentomatoes.com/docs/read/json/v10/Box_Office_Movies

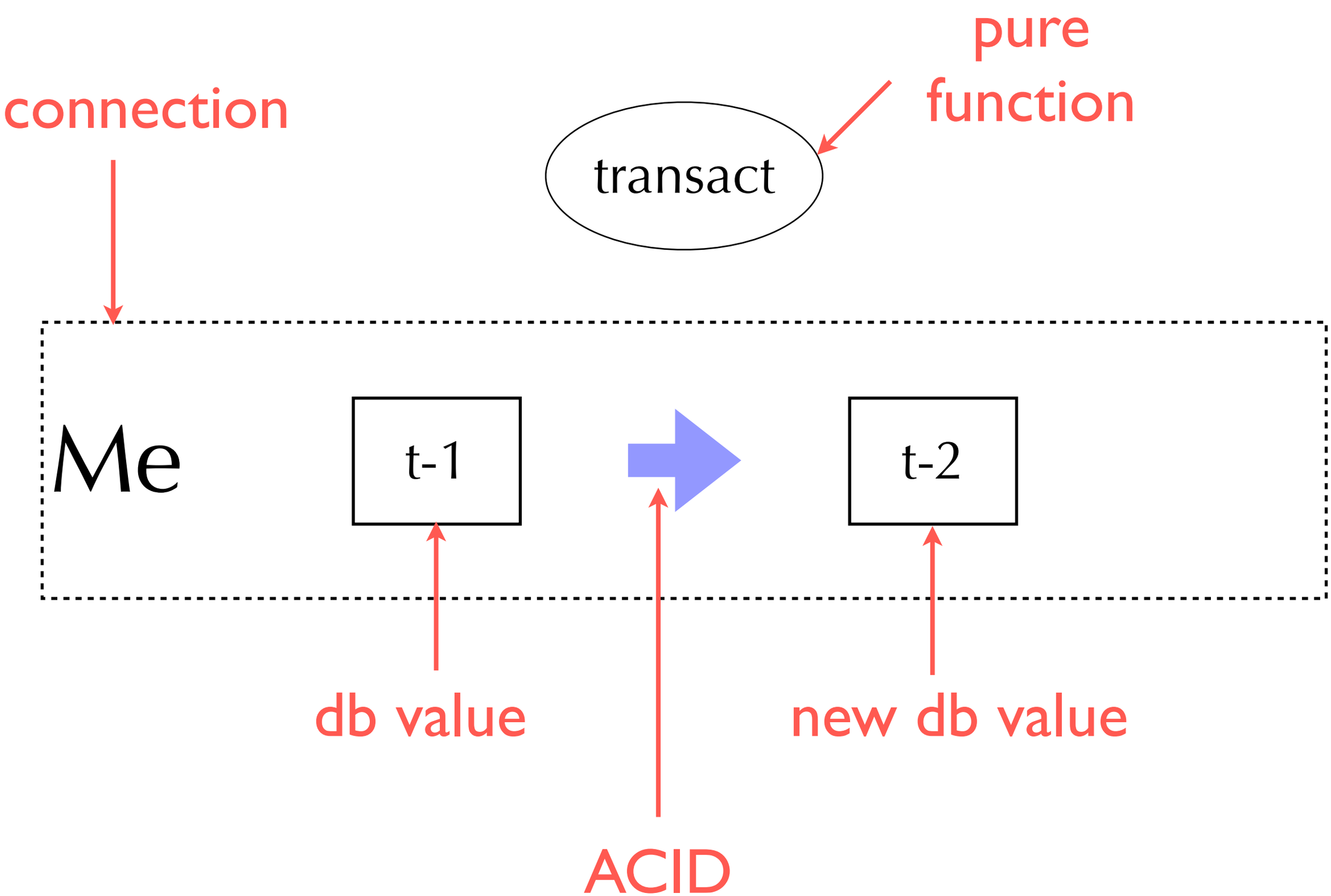
PICOs for Big Data

```
(defn my-data-2 []
  (->>
    (pig/load-tsv "input.tsv")
    (pig/map (fn [[a b c]]
               {:sum (+ (Integer/valueOf a) (Integer/valueOf b))
                :name c}))
    (pig/filter (fn [{:keys [sum]}]
                  (< sum 5)))))

=> (pig/dump (my-data-2))
[{:sum 3, :name "foo"}]
```

https://github.com/Netflix/PigPen


pure
function
reference
nachos
Me
182
184
value
new value
succession
function


pure function
connection
transact
Me
t-1
t-2
db value
new db value
ACID

Datomic

ACID data of record

persistent data structures: “scm for business data”

distributed, componentized, read scalable & elastic

information and logic as PICOs in any *peer process*

Connect and Query

```
Connection conn =
connect("datomic:ddb://us-east-1/mb/mbrainz");

Database db = conn.db();

Set results = q(..., db);

Set crossDbResults = q(..., db1, db2);

Entity e = db.entity(42);
```

Connect and Query

```
Connection conn =
connect("datomic:ddb://us-east-1/mb/mbrainz");

Database db = conn.db();

Set results = q(..., db);

Set crossDbResults = q(..., db1, db2);

Entity e = db.entity(42);
```

database is a lazily realized value, available to all peers equally

Producing Clojure

Design

specification of an artifact using components to meet goals subject to constraints

Goal

give skilled devs superpowers to build business software systems

Goal

give skilled devs superpowers to build business software systems

Goal

give skilled devs superpowers to build business software systems

Constraints

for wide adoption

- open source
- target established platforms

for viability

- performance
- stability

Constraints

for wide adoption

- open source
- target established platforms

for viability

- performance
- stability

Open Source

licensed under EPL

contributor agreement

artifacts in Maven Central

not just language: bunch of libs too

Might Surprise You

we take patches, not pull requests

we prefer designs over patches

we prefer problem statements over designs

Constraints

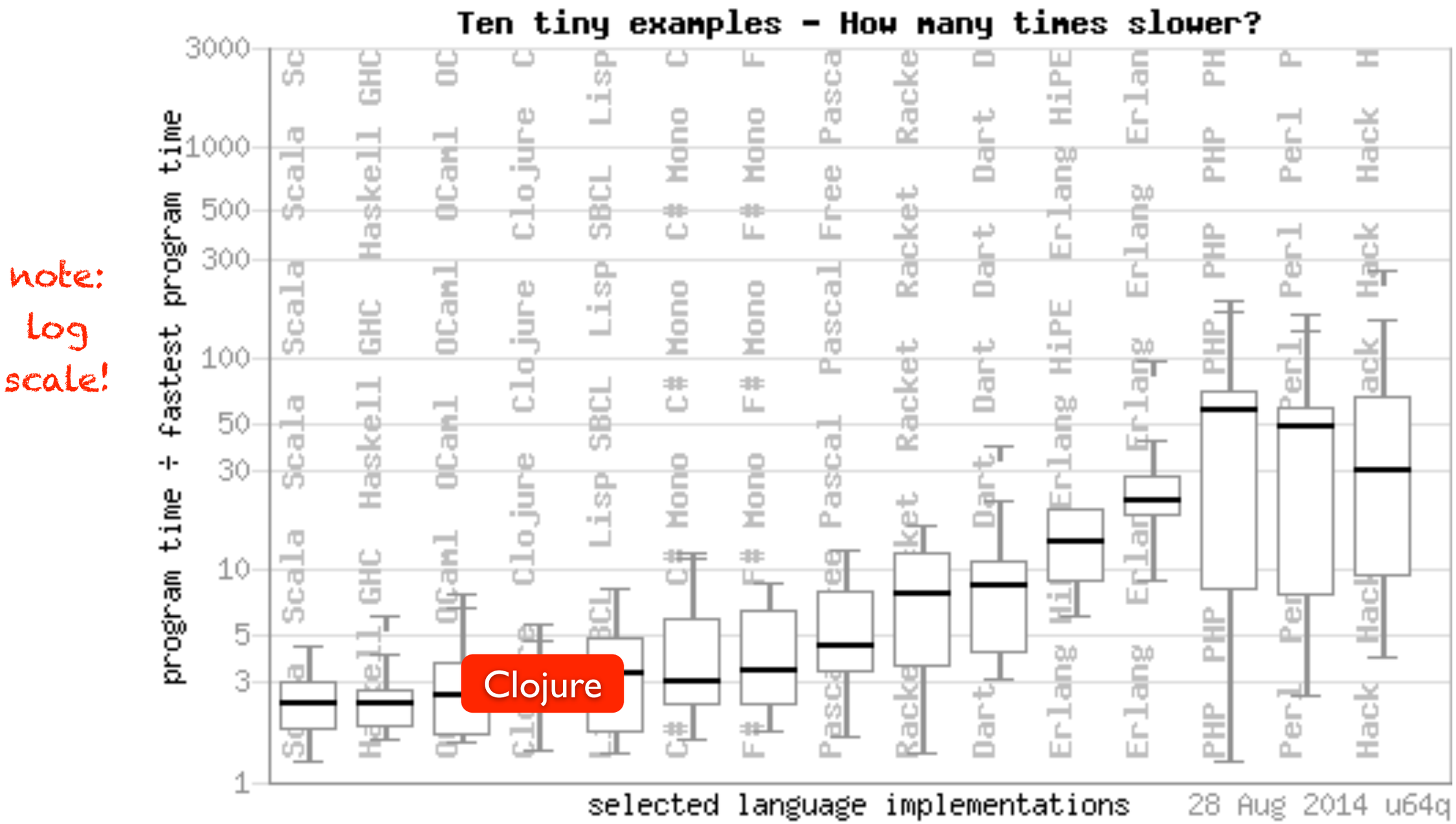
for wide adoption

- open source
- target established platforms

for viability

- performance
- stability

Server Performance



note: log scale!

http://benchmarksgame.alioth.debian.org/u64q/which-programs-are-fastest.php

Constraints

for wide adoption

- open source
- target established platforms

for viability

- performance
- stability

The Pragmatic Programmers

Programming Clojure

2009

Stuart Halloway

Edited by Susannah Davidson Pfalzer

Maintaining *Programming Clojure*

release	date	breakage*
1.0	05/2009	-
1.1	12/2009	None
1.2	08/2010	None
1.3	09/2011	Small
1.4	04/2012	None
1.5	03/2013	None
1.6	03/2014	None
1.7	TBD	None

One size does not fit all

Examples of Simplicity

syntax

protocols

values and references

PICOs!

Benefits of Clojure

concision

generality

robustness

agility

cognitect

@stuarthalloway

Clojure
http://clojure.com. The Clojure language.
http://cognitect.com/. The company behind Clojure, ClojureScript, & Datomic.
http://blog.cognitect.com/cognicast/. The Cognicast.
http://bit.ly/clojure-bookshelf. 40 recommendations from Rich.
http://clojure.in/. Planet Clojure.

@stuarthalloway
https://github.com/stuarthalloway/presentations/wiki. Presentations.
https://github.com/stuarthalloway/exploring-clojure. Sample Code.
http://pragprog.com/book/shcloj2/programming-clojure. *Programming Clojure*.
mailto:stu@cognitect.com

cognitect