Datomic

@stuarthalloway

the Datomic database is

indelible

chronological

flexible

powerful

simple

Datomic is *indelible*:

datoms cannot be modified or removed

why source control?

reify change

trigger workflows

compare points in time

prove a point (audit)

generic (vs. ad hoc) model of time

update-in-place (sad)

entity	attribute	value
jane	likes	broccoli

update-in-place (sad)

entity	attribute	value
jane	likes	pizza

indelible, not CRUD

- “create”
  - assertion
- “update”
  - retraction + assertion
- “delete”
  - retraction

indelible

entity	attribute	value	transaction	assert?
jane	likes	broccoli	1008	true
jane	likes	broccoli	1148	false
jane	likes	pizza	1148	true

Datomic is chronological

every datom is timestamped

- query as-of a particular moment in time

strong consistency

- ACID
- CAP

filtered retractions

entity	attribute	value	transaction	assert?
jane	likes	broccoli	1008	true
1008	txInstant	… 04:00	1008	true
jane	likes	broccoli	1148	false
jane	likes	pizza	1148	true
1148	txInstant	… 03:00	1148	true

as-of

entity	attribute	value	transaction	assert?
jane	likes	broccoli	1008	true
1008	txInstant	… 04:00	1008	true
jane	likes	broccoli	1148	false
jane	likes	pizza	1148	true
1148	txInstant	… 03:00	1148	true

history

entity	attribute	value	transaction	assert?
jane	likes	broccoli	1008	true
1008	txInstant	… 04:00	1008	true
jane	likes	broccoli	1148	false
jane	likes	pizza	1148	true
1148	txInstant	… 03:00	1148	true

“Code doesn’t exist
unless it’s checked into a
version control system”

“Data doesn’t exist unless it’s transacted into an indelible, chronological database”

Datomic is flexible

- Datomic stores granular *information*
  - datoms are the atoms of information
  - attribute-level schema
- model your domain
  - instead of torturing it to fit into tables

trading off flexibility (NoSQL)

"picking the right data model is the hardest part …"

"model your data to fit your queries"

- "don't model around relations"
- "don't model around objects"

flexibility

picking the right data model is the easiest part

model your data to fit your domain

- model relations
- model objects

no “impedance mismatch”

one db, many query styles

structure	**attribute**
k/v	AVET
row	EAVT
column	AEVT
document	EAVT, components
graph	VAET

Datomic is powerful

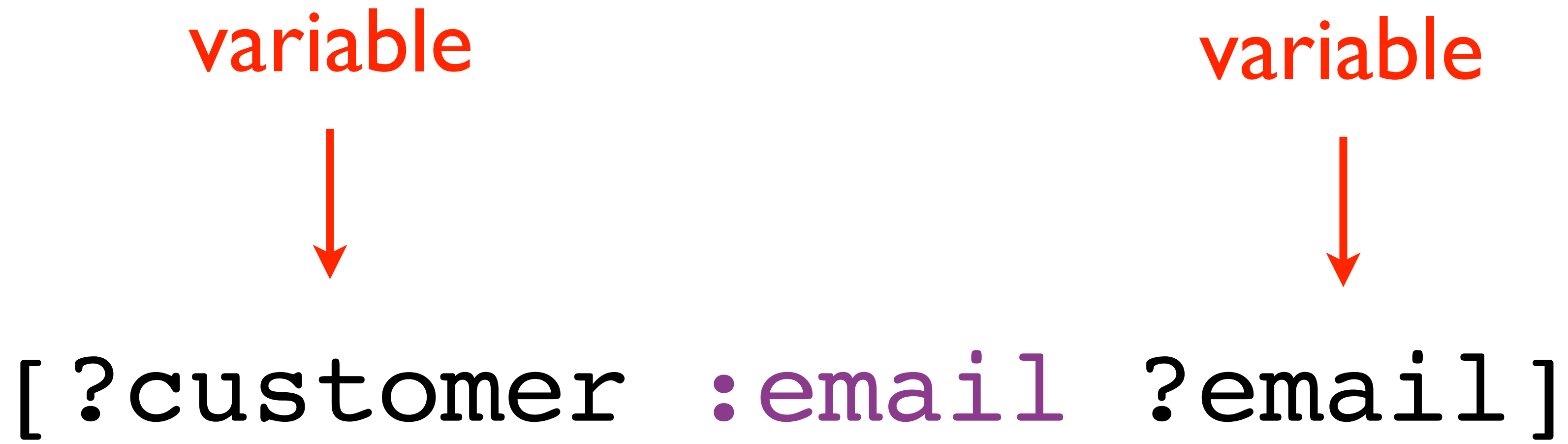
- datalog query
  - logic programming
  - pattern syntax
  - joins
  - rules

example database

entity	attribute	value
42	:email	jdoe@example.com
43	:email	jane@example.com
42	:orders	107
42	:orders	141

data pattern

Constrains the results returned,
binds variables



variable

variable

```
[?customer :email ?email]
```

find by email

entity	attribute	value
42	:email	jdoe@example.com
43	:email	jane@example.com
42	:orders	107
42	:orders	141

```
[?customer :email ?email]
```

the gourmet jerky problem

```
[:find ?customer ?product
 :where [?customer :shipAddress ?addr]
        [?addr :zip ?zip]
        [?product :product/weight ?weight]
        [?product :product/price ?price]
        [(Shipping/estimate ?zip ?weight) ?shipCost]
        [(<= ?price ?shipCost)]]
```

Datomic is simple

- simple
  - not complected
  - not woven together
  - orthogonal

a complected database

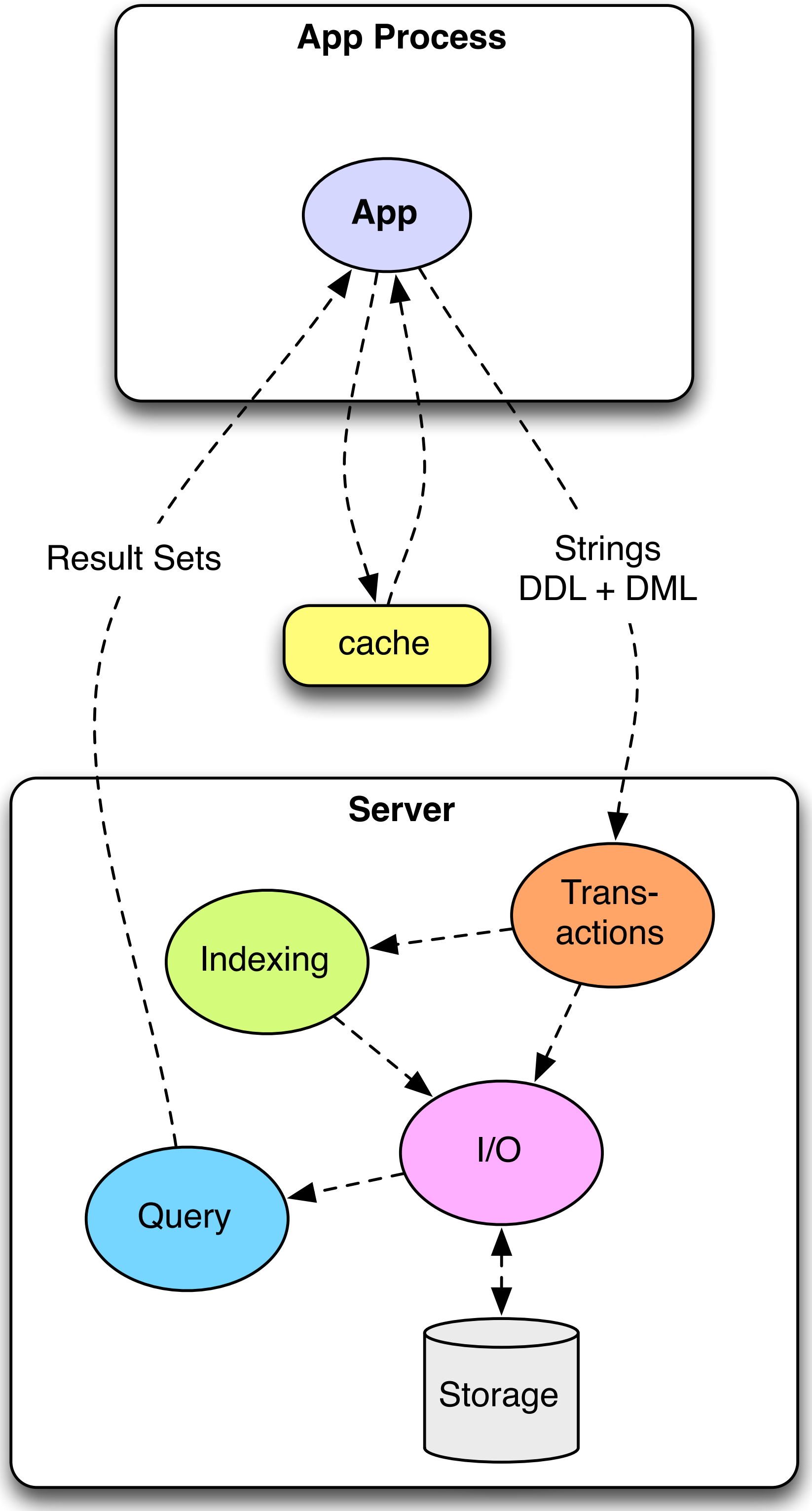
monolithic server

reads

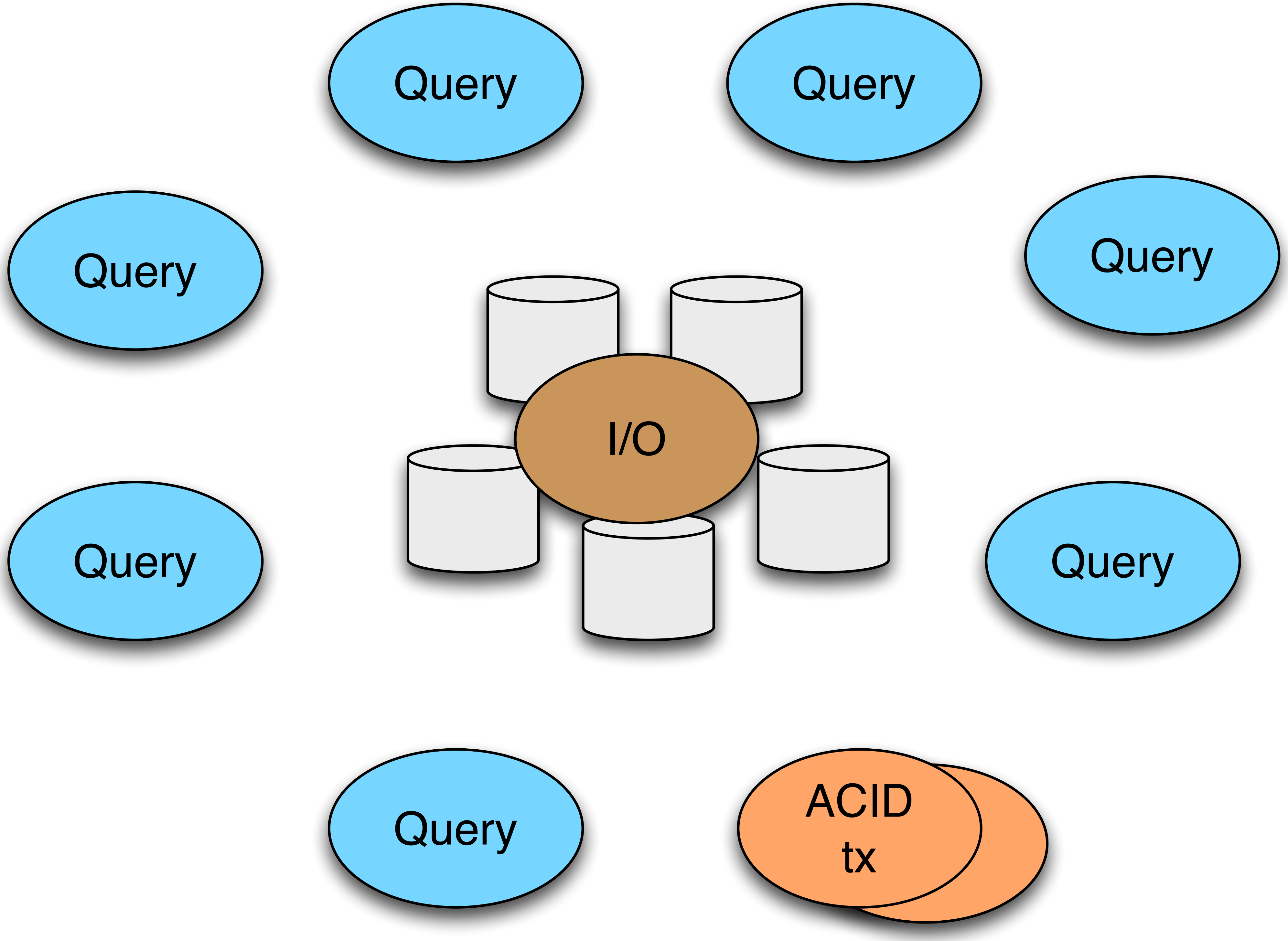
writes

indexing

storage



Datomic simplicity



architectural benefits

- run on your own storage
  - DynamoDB
  - SQL
  - Cassandra
- horizontal read scaling

Datomic is

indelible

chronological

flexible

powerful

simple

successful

“We needed the flexibility and agility of a startup, and the system of record / audit trail features one would find in legacy banking systems. I didn’t find a better option than **Datomic with this first class concept of time”**

http://www.datomic.com/nubanks-story.html

how did a small team
build this?

what made lisp different

feature	Java	Clojure
conditionals	✔	✔
variables	✔	✔
garbage collection	✔	✔
recursion	✔	✔
function type	*	✔
symbol type		✔
whole language available		✔
everything’s an expression		✔
homoiconicity		✔

http://www.paulgraham.com/diff.html

ask the internet

rare

cuts through any problem

intimidating

elegant

ask the internet

rare

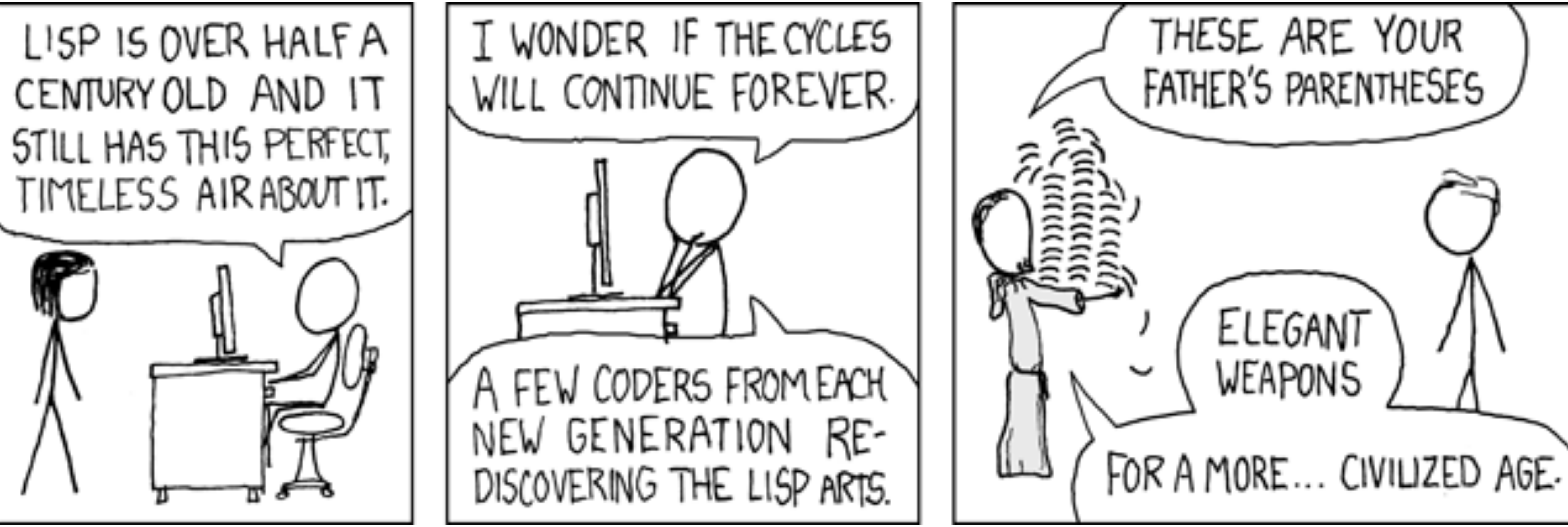
cuts through any problem

intimidating

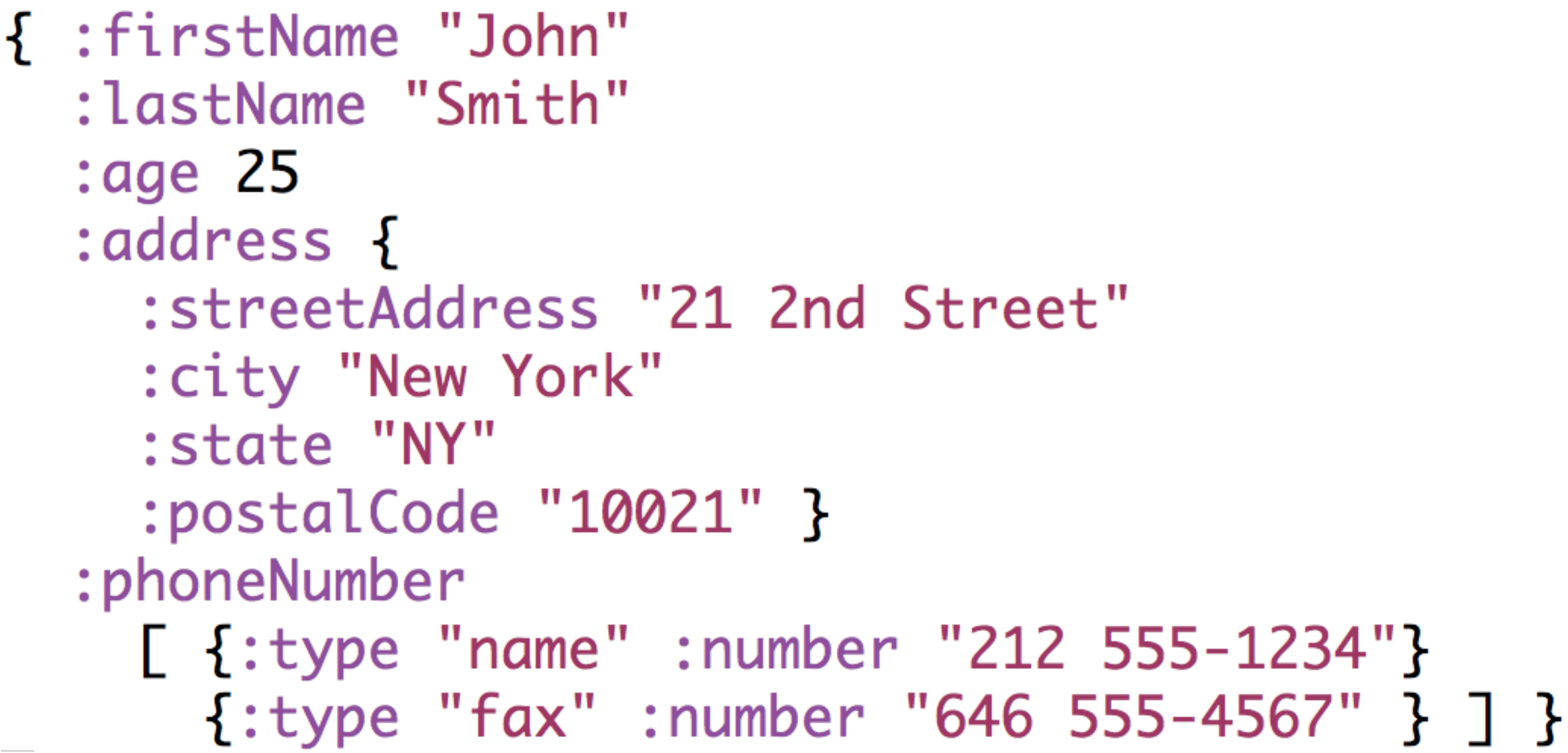
elegant

instantly recognizable emblem of people you don’t want to f* with**

https://xkcd.com/297/


LISP IS OVER HALF A CENTURY OLD AND IT STILL HAS THIS PERFECT, TIMELESS AIR ABOUT IT.
I WONDER IF THE CYCLES WILL CONTINUE FOREVER.
A FEW CODERS FROM EACH NEW GENERATION RE-DISCOVERING THE LISP ARTS.
THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.

extensible data notation (edn)



```
{ :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`

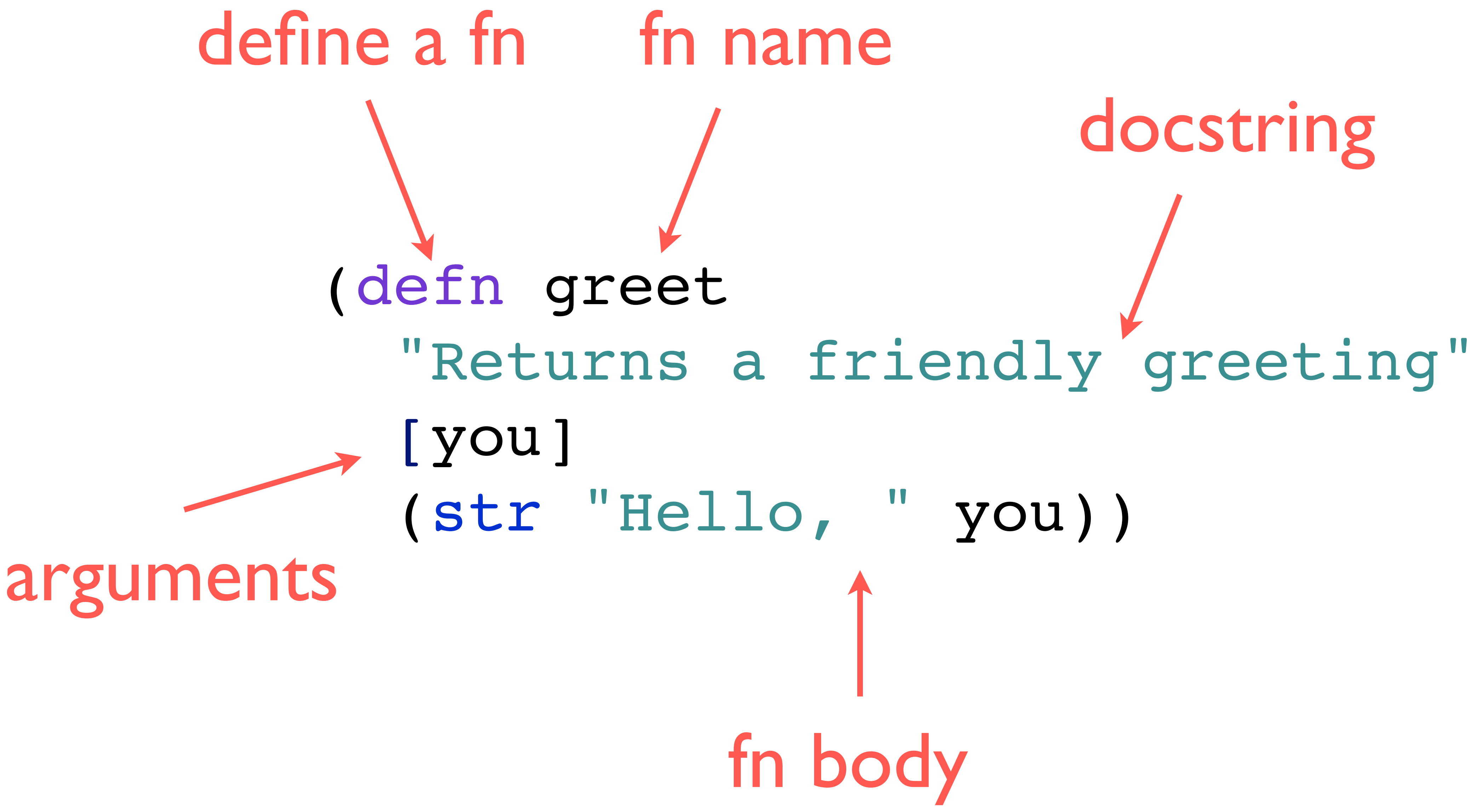
type	properties	examples
list	sequential	`(1 2 3)`
vector	sequential and random access	`[1 2 3]`

map	associative	`{:a 100`
		` :b 90}`

set	membership	`#{:a :b}`

```
"hello world"
```

hello you



```clojure
(defn greet
  "Returns a friendly greeting"
  [you]
  (str "Hello, " you))
```

the StrUtils problem

existing class

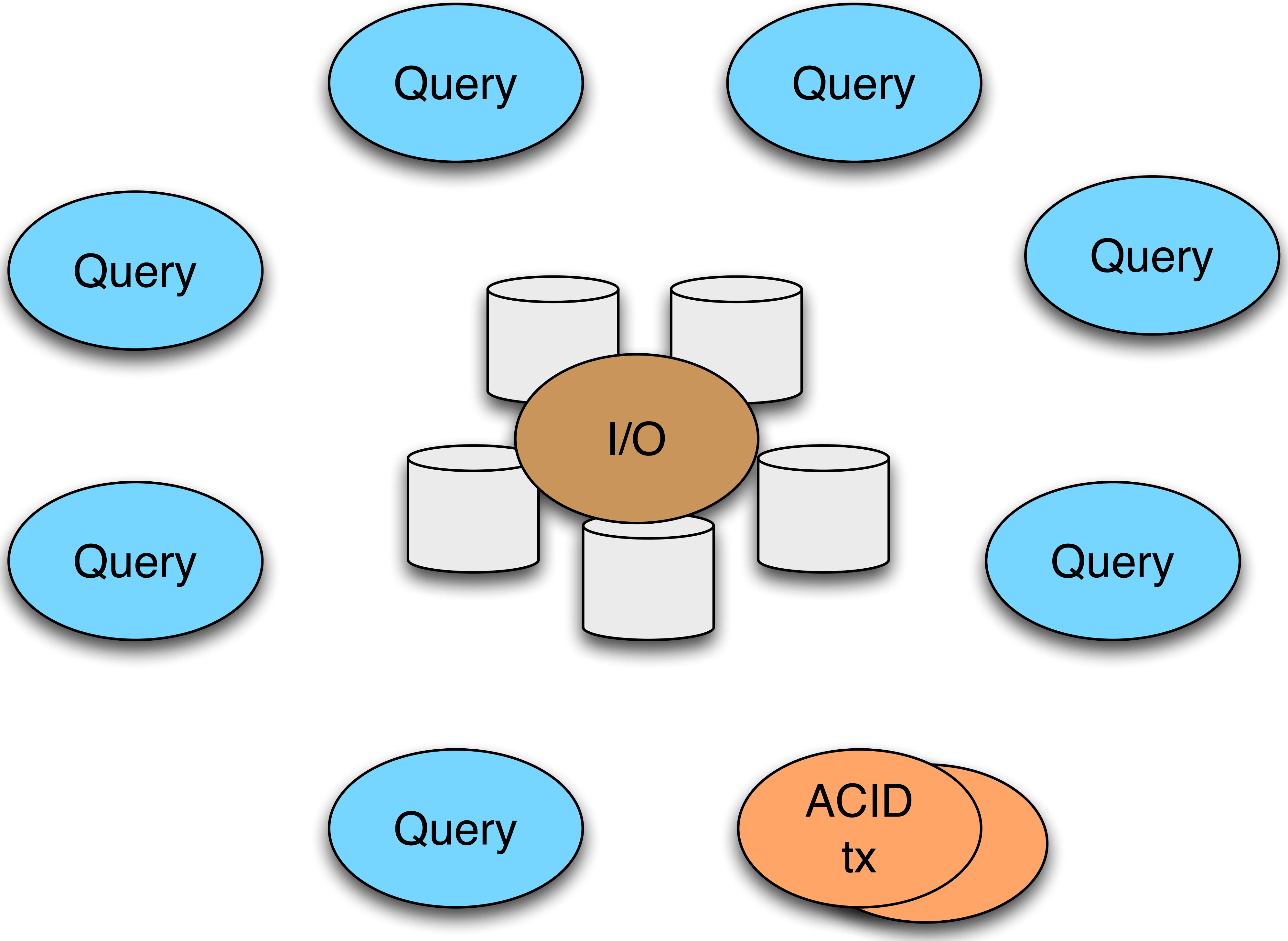
existing interface

never the twain shall meet

protocols

```
(defprotocol Blank
  (blank? [_]))

(blank? "foo")
=> IllegalArgumentException

(extend-protocol Blank
  String
  (blank?
   [s]
   (every? #(Character/isWhitespace %) s)))

(blank? "  ")

=> true
```

storage protocol



testing is hard

difficult to interpret tests as knowledge about system

difficult to achieve good coverage

costly to develop and maintain

trivial and serial tests vs. complex and parallel reality

generative testing

programmer models the domain

a *program* writes the individual tests

validate categoric properties, not specific outcomes

a program wrote this program

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Case 0

(def
 sql0
 "SELECT r1.e,r1.v\nFROM r1\nWHERE (r1.e=0 OR r1.e=1 OR r1.e=7) AND r1.a='b'")
(def e0 "Expected result for test query 0" #{[7 1] [0 8]})
(let [expected e0 actual
      (datomic.api/q
       (quote
        {:find [?r1-e ?r1-v],
         :where ([$r1 ?r1-e :b ?r1-v]
                 [(contains? #{0 7 1} ?r1-e)]),
         :in [$r1]}) db1)]
  (when-not (= expected actual)
    (throw (ex-info "Query results did not crosscheck"
                    {:expected expected, :actual actual}))))
```

reading test failures is hard

```
(tc/quick-check 100 some-complex-property)
```

```
{:fail [[10 1 28 40 11 -33 42 -42 39 -13
         13 -44 -36 11 27 -42 4 21 -39]]}
```

programmatic shrinking

```
(tc/quick-check 100 some-complex-property)
```

```
{:fail [[10 1 28 40 11 -33 42 -42 39 -13
         13 -44 -36 11 27 -42 4 21 -39]],
 :shrunk {:total-nodes-visited 38,
          :depth 18,
          :result false,
          :smallest [[42]]}}
```

communication is hard

objects make terrible machines

function chains make poor machines

direct-connect relationships

callback hell

j.u.c queues block real threads

threads are expensive and/or nonexistent

core.async (CSP)

first class processes

first class channels

concurrency primitive (coordination)

coherent sequential logic

multi reader/writer

buffering

search with no threads, SLA

```
(defn search [query]
  (let [c (chan)
        t (timeout 80)]
    (go (>! c (<! (fastest query web1 web2))))
    (go (>! c (<! (fastest query image1 image2))))
    (go (>! c (<! (fastest query video1 video2))))
    (go (loop [i 0
               ret []]
          (if (= i 3)
            ret
            (recur (inc i)
                   (conj ret (alt! [c t] ([v] v)))))))))
```

http://talks.golang.org/2012/concurrency.slide#50

basic transduction

```
(a/transduce
  (comp
    (halt-when error?)
    (map ...)
    (filter ...))
  (completing ...)
  accumulator
  query)
```

clojure.spec

a standard, expressive, powerful, integrated system for specification and testing

a taste of spec

```
(s/def ::sku
  (s/and string?
         #(str/starts-with "SKU-" %)))

(s/def ::purchaser
  (s/or :account-id pos-int?
        :email string?))

(s/def ::quantity pos-int?)

(s/def ::import-record
  (s/tuple [::purchaser ::sku ::quantity]))
```

spec capabilities

what	how
what are the building blocks?	declarative structure
what invariants hold?	arbitrary predicates
how do I check?	validation
what went wrong?	explanation
what went right?	conformance
docs	autodoc
examples	sample generator
am I using this right?	instrumentation
is my code correct?	generative testing
a la carte self-check	assertion

spec vs. Java types and tests

	jUnit etc.	Java types	spec
expressive	very	no	very
powerful	stakeholder correctness	type correctness	stakeholder correctness
integrated	no	compile-time, must flow	dynamic
specification	no	static	yes
testing	manual	no	generative
agility	expensive	fragility	dynamic
reach	expensive	libs, apps	systemic

core values

simplicity

power

focus

stability

simplicity

just keep finding smaller abstractions?!

planning, agility, and patience

takes time and courage

power

design for maximum leverage from the platform (JVM)

- semantic fidelity
- performance

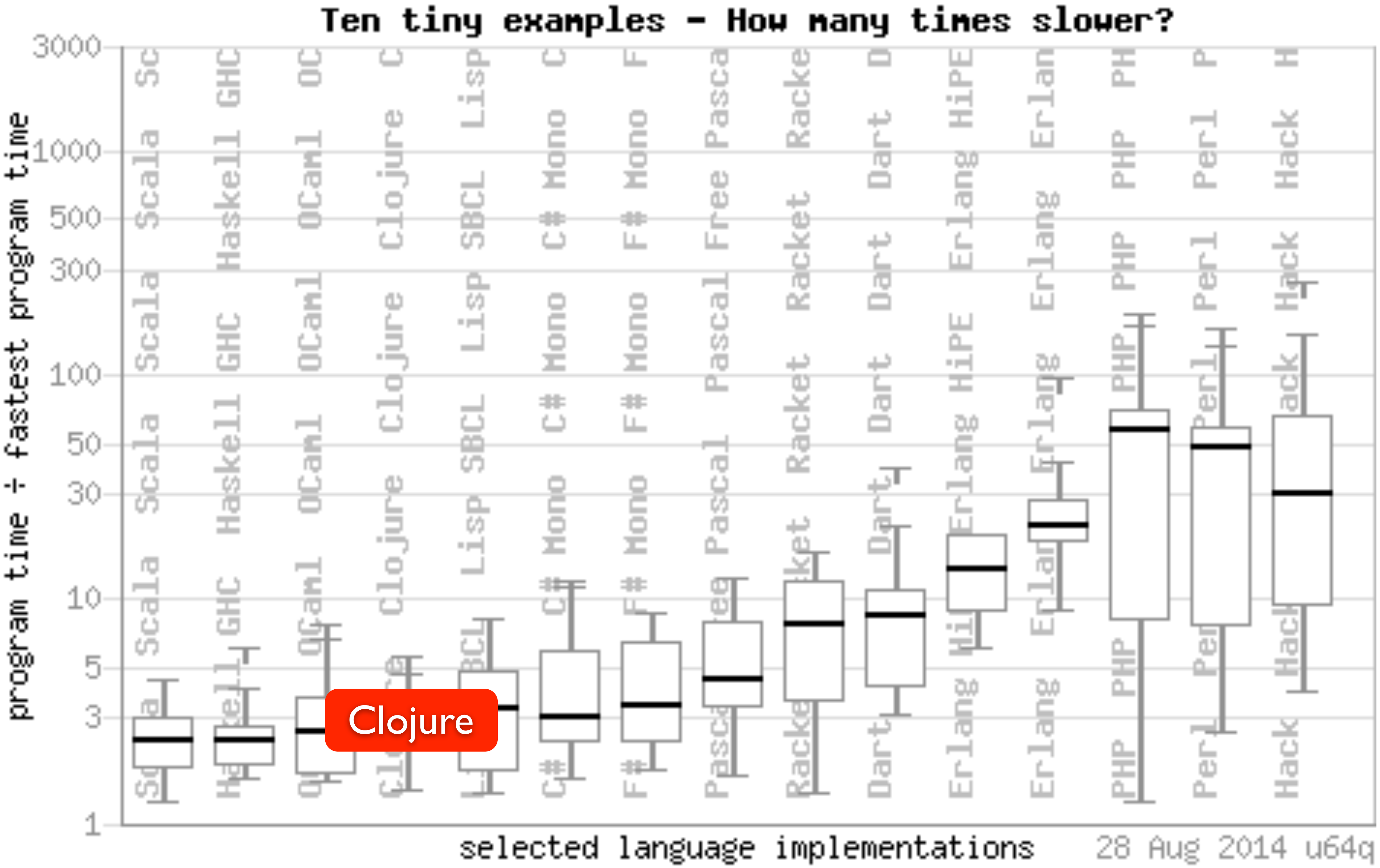
semantic fidelity

Clojure programs are Java programs

call Java from Clojure without wrappers

call Clojure from Java without wrappers

performance



note:
log
scale!

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

focus

- related coding principles
  - single-responsibility principle
  - don’t repeat yourself (dry)
- Clojure superpowers
  - simplicity & homoiconicity
  - systemic generality

systemic generality

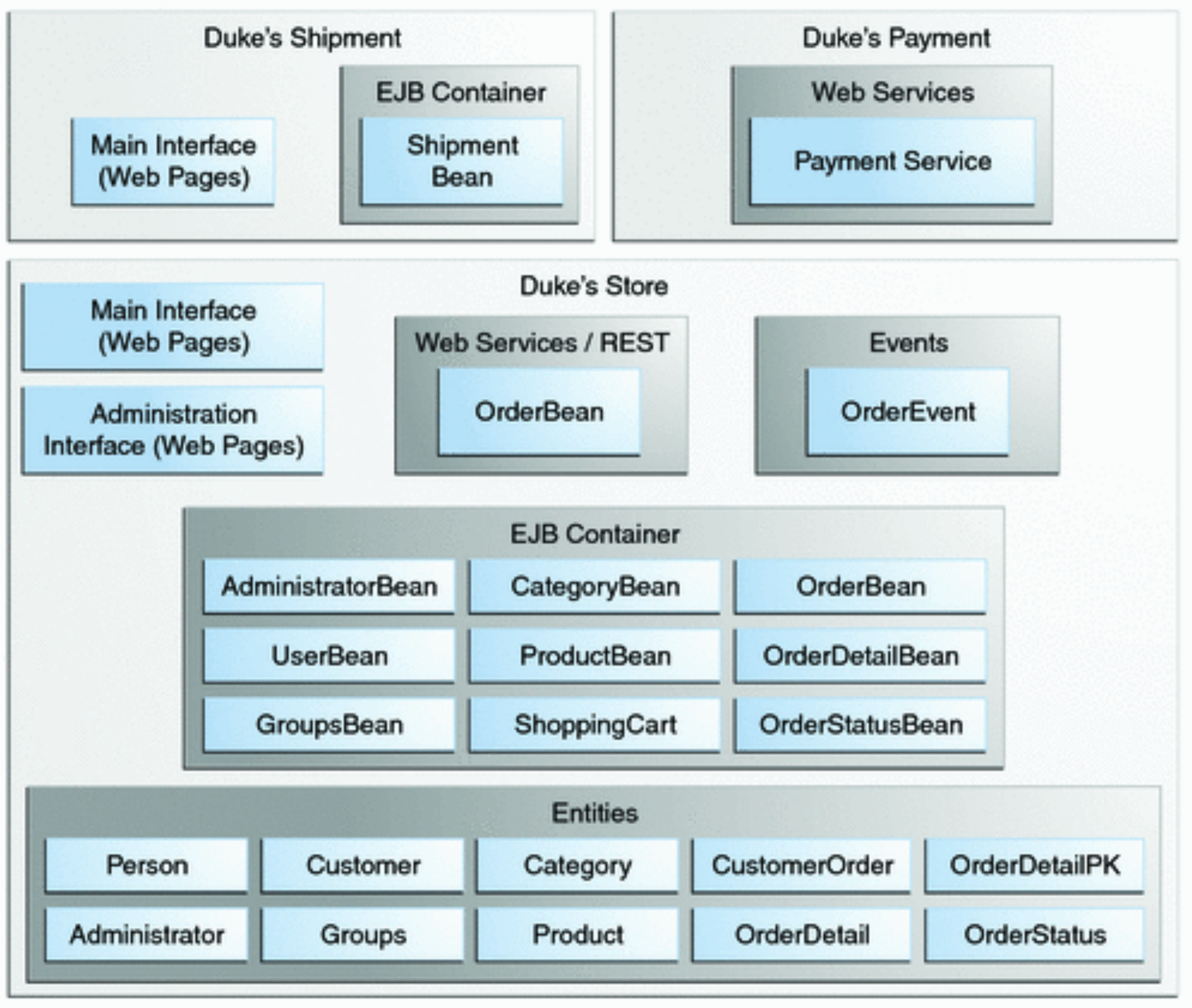
generality

all domains use the same general-purpose data structures and functions

systemic

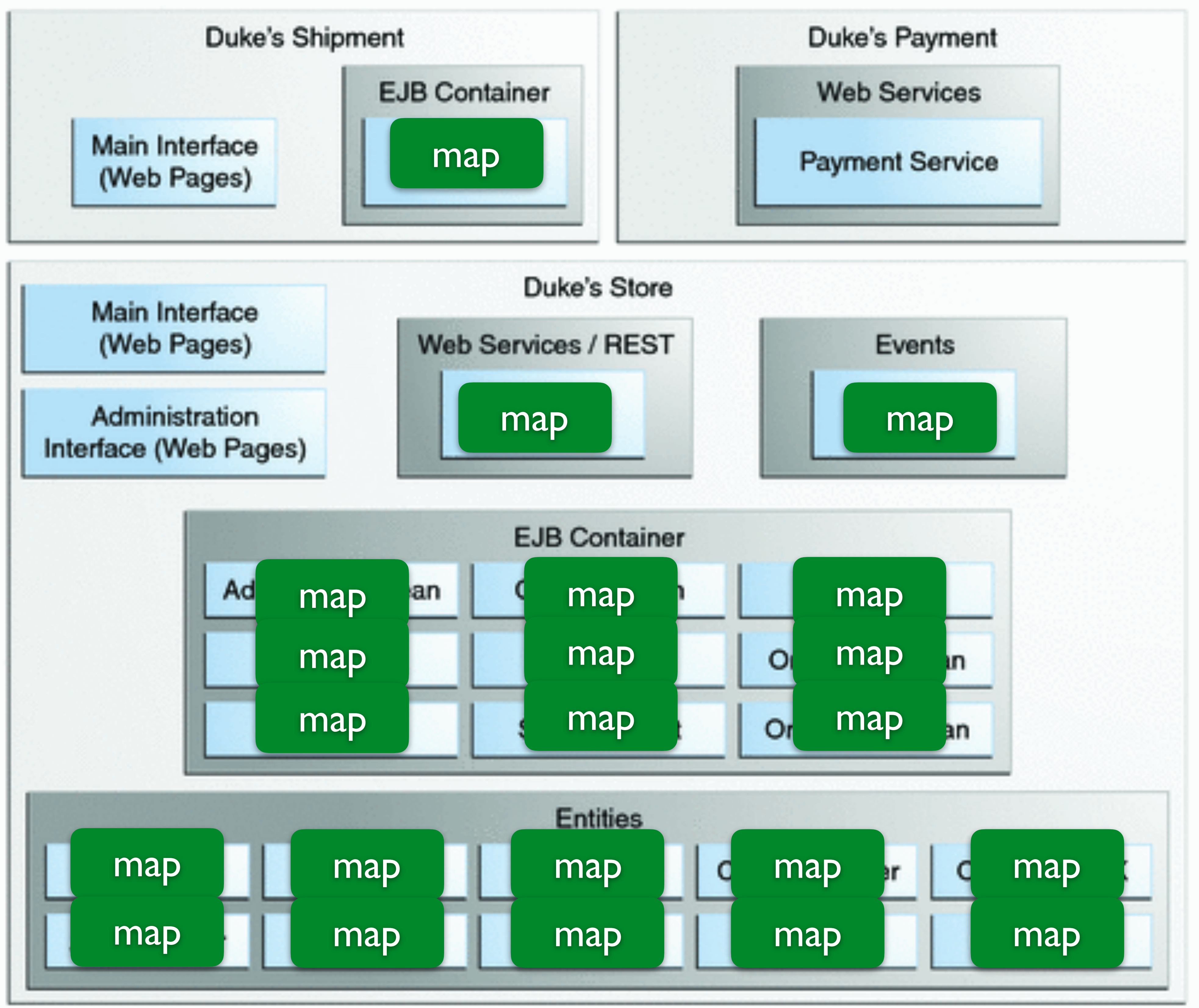
in libs, in apps, in config, on wires, at rest

specificity



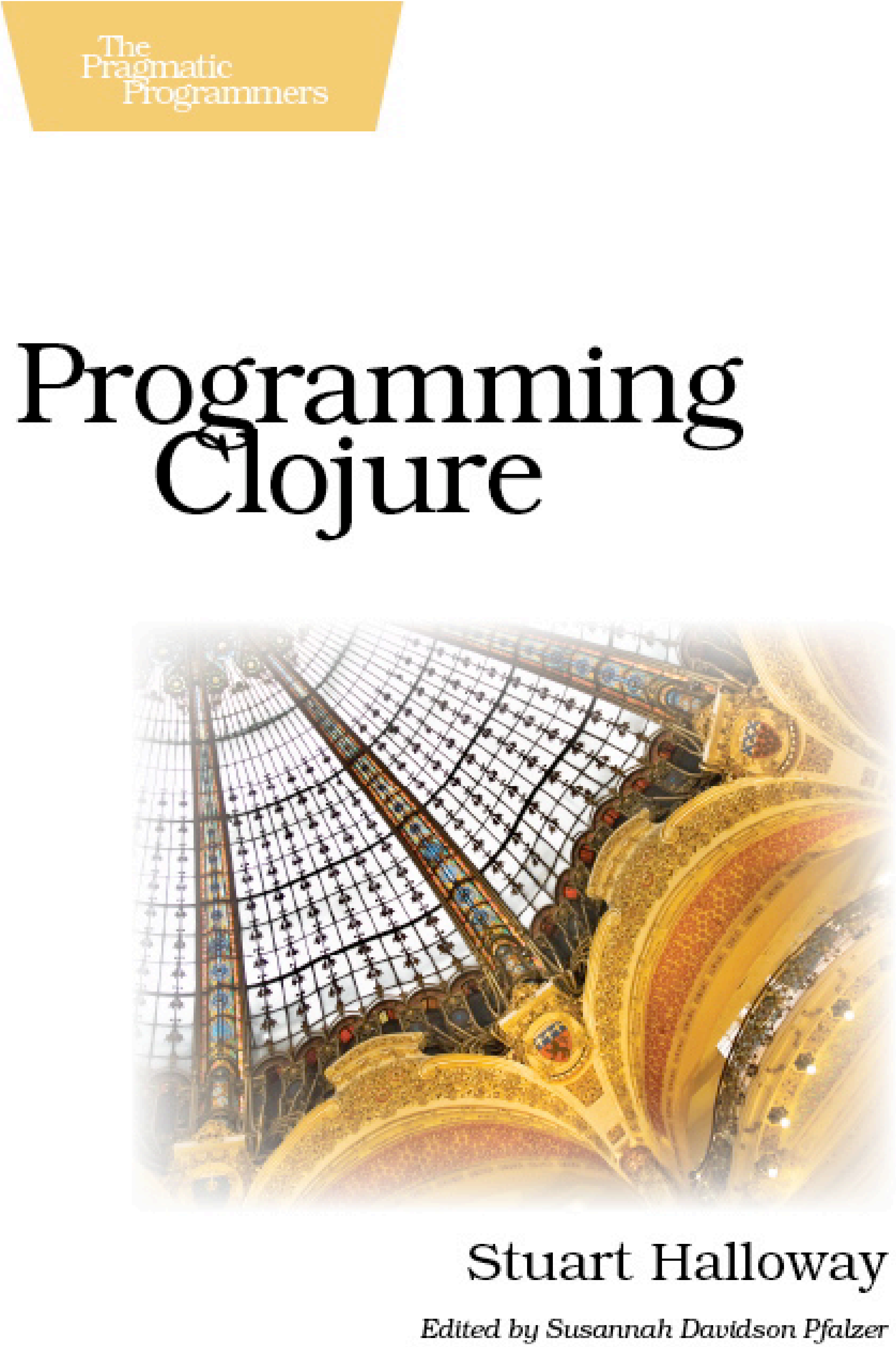
docs.oracle.com/javaee/6/tutorial/doc/figures/dukesforest-architecture.gif

generality


Duke's Shipment
Main Interface (Web Pages)
EJB Container
map
Duke's Payment
Web Services
Payment Service
Duke's Store
Main Interface (Web Pages)
Administration Interface (Web Pages)
Web Services / REST
map
Events
map
EJB Container
map
map
map
map
map
map
map
map
map
Entities
map
map
map
map
map
map
map
map
map
map

stability

- semantic versioning is completely broken
- we aspire to a different model
  - growth: accretion, relaxation, fixation
  - no breakage



The Pragmatic Programmers

Programming Clojure

Stuart Halloway

Edited by Susannah Davidson Pfalzer

2009

evidence for stability

Clojure release	date	breakage
1.0	05/2009	-
1.1	12/2009	-
1.2	08/2010	-
1.3	09/2011	1 LOC in test suite
1.4	04/2012	-
1.5	03/2013	-
1.6	03/2014	-
1.7	06/2015	-
1.8	01/2016	-
1.9	TBD	-

takeaways

build for yourself

value individuals *and* tools

value planning *and* agility

listen

trust yourself

will these ideas make me a better programmer?

Datomic

@stuarthalloway