Clojure

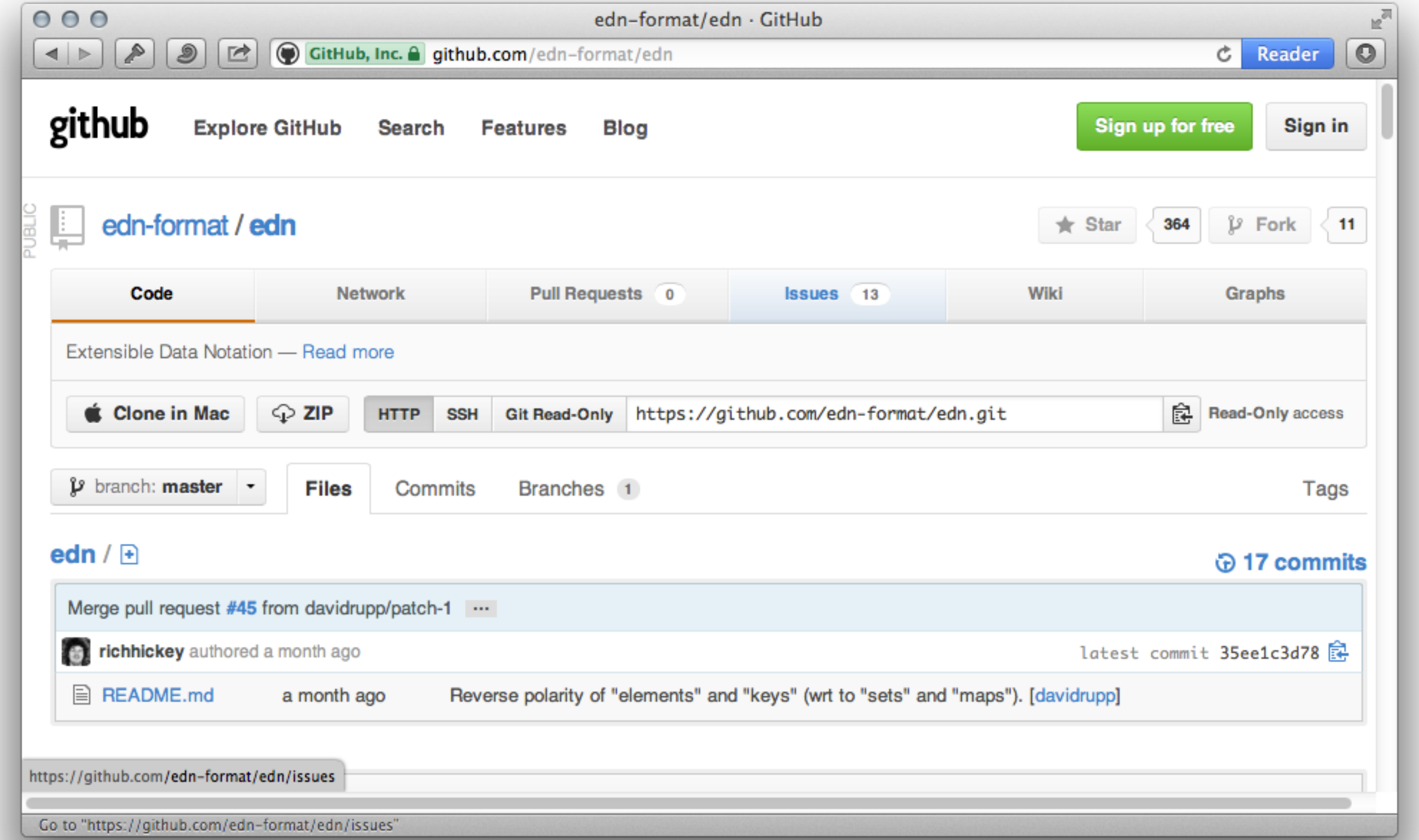
in 10 big ideas

@stuarthalloway
stu@cognitect.com

brought to you by

cognitect

1. edn



Extensible Data Notation — Read more

https://github.com/edn-format/edn.git

Merge pull request #45 from davidrupp/patch-1 ···

richhickey authored a month ago

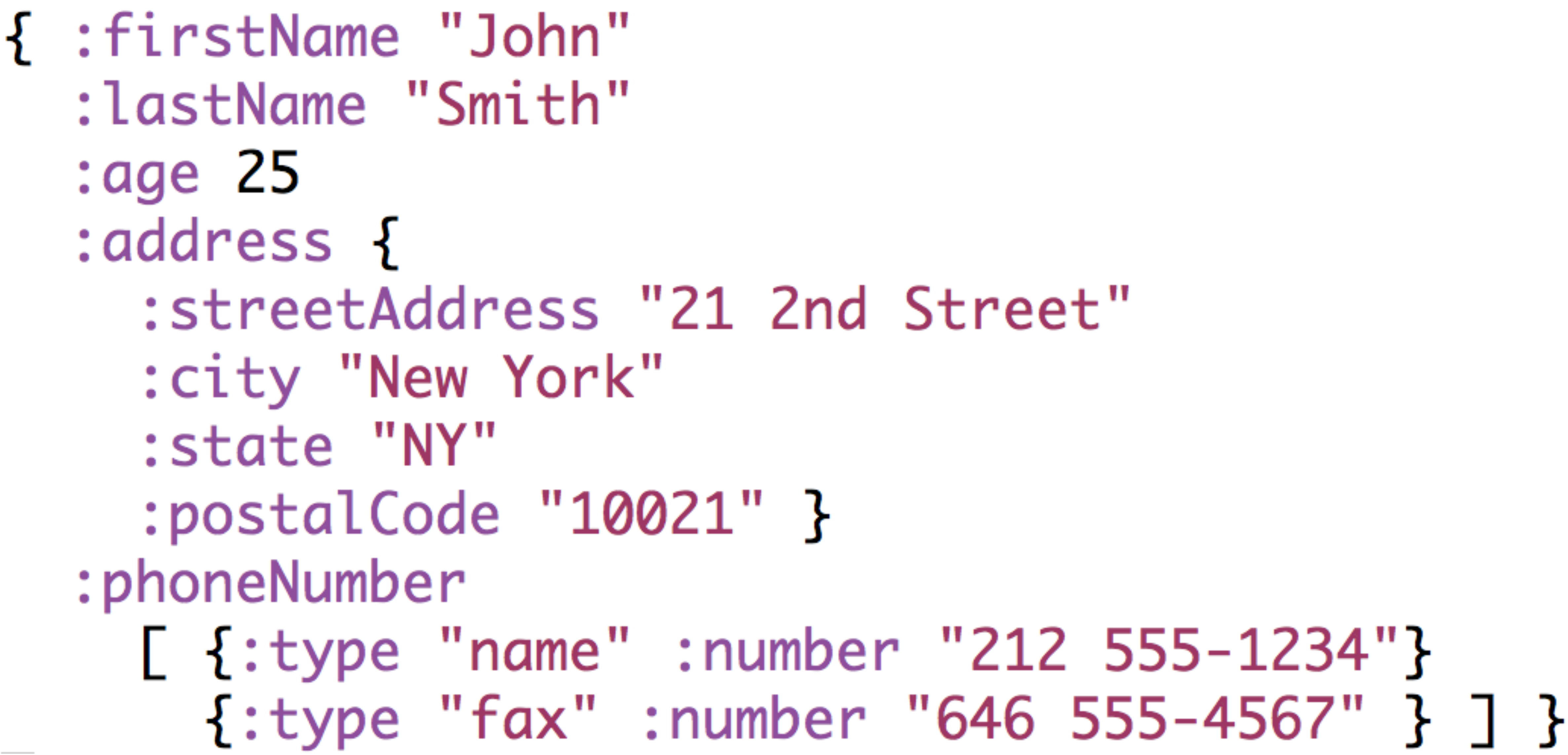
latest commit 35ee1c3d78

README.md	a month ago	Reverse polarity of "elements" and "keys" (wrt to "sets" and "maps"). [davidrupp]

https://github.com/edn-format/edn/issues

Go to "https://github.com/edn-format/edn/issues"

edn example



```
{ :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`

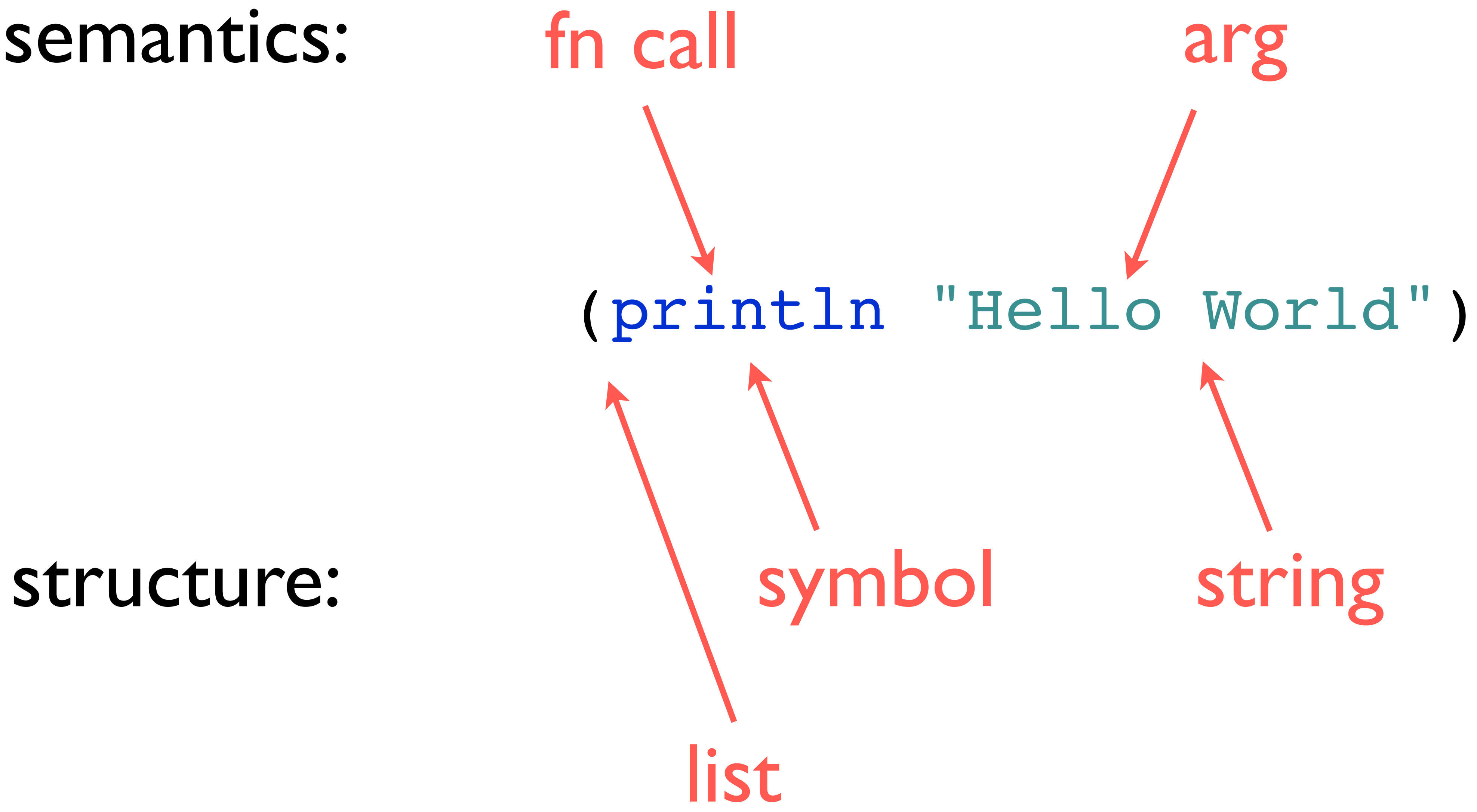
type	properties	examples
list	sequential	`(1 2 3)`
vector	sequential and random access	`[1 2 3]`

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

set	membership	`#{:a :b}`

program in data, not text

function call



semantics:

fn call

arg

```
(println "Hello World")
```

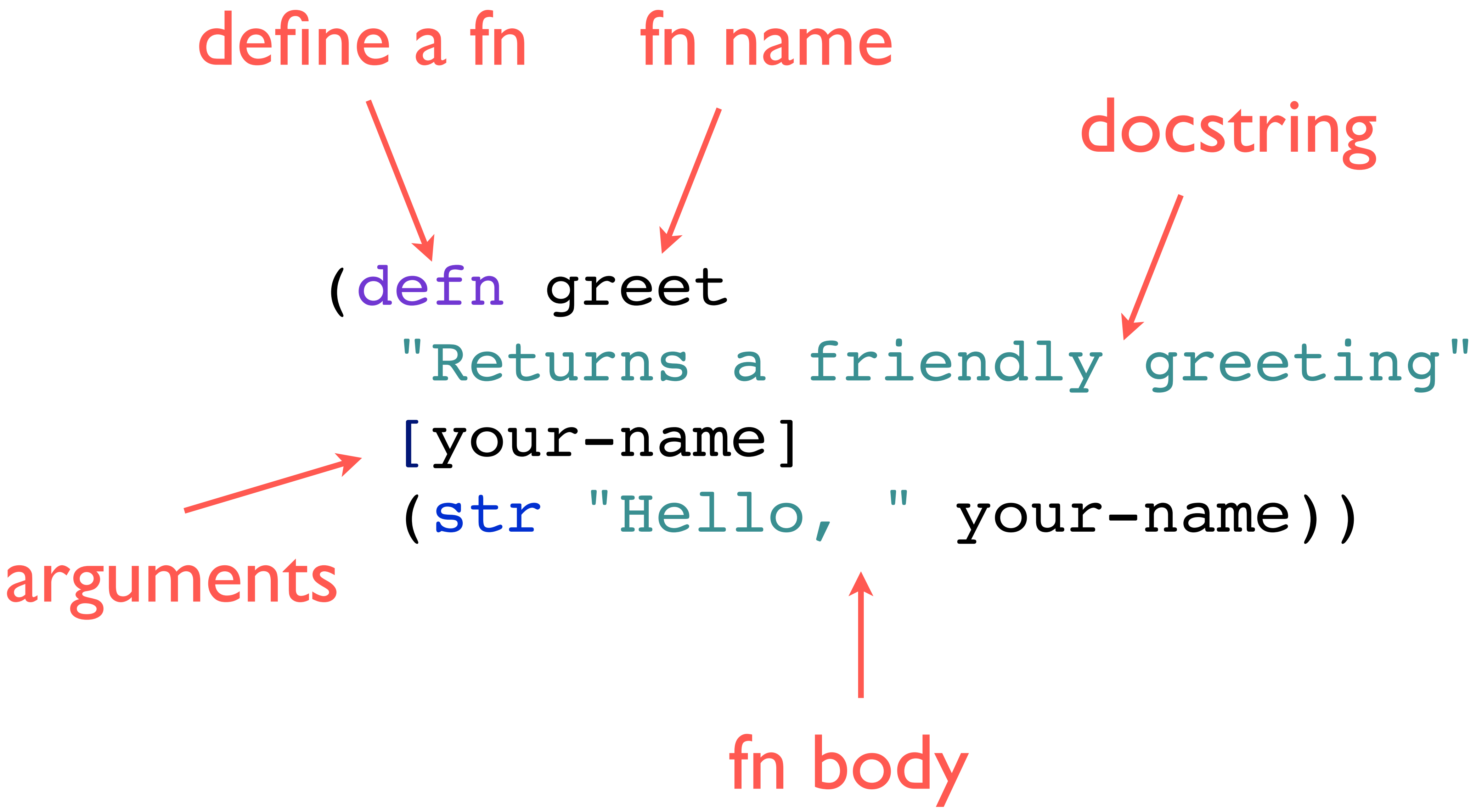
structure:

symbol

string

list

function def



define a fn

fn name

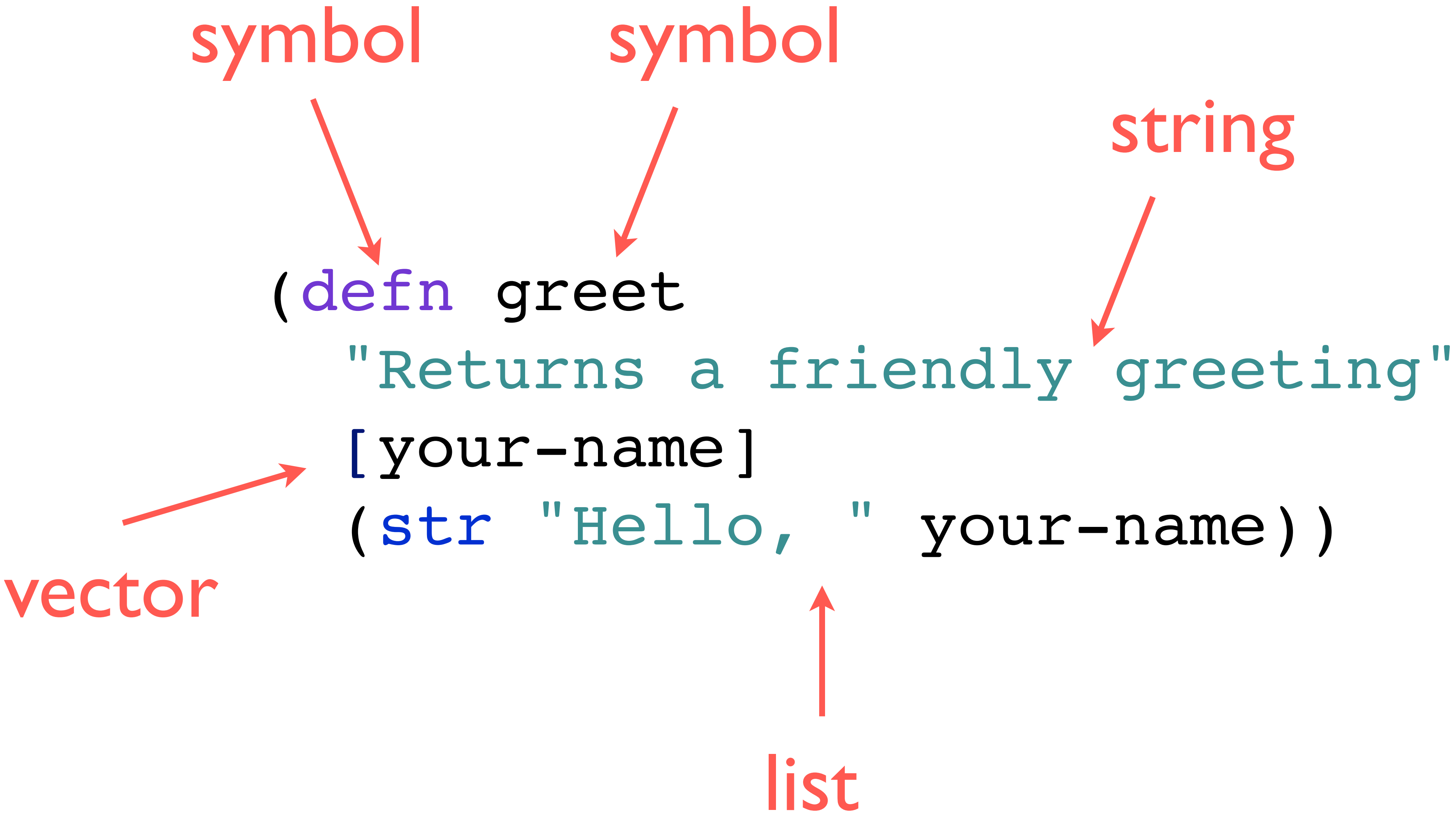
docstring

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

arguments

fn body

still just data



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

generic extensibility

#name edn-form

name describes interpretation of following element

recursively defined

all data can be literal

built-in tags

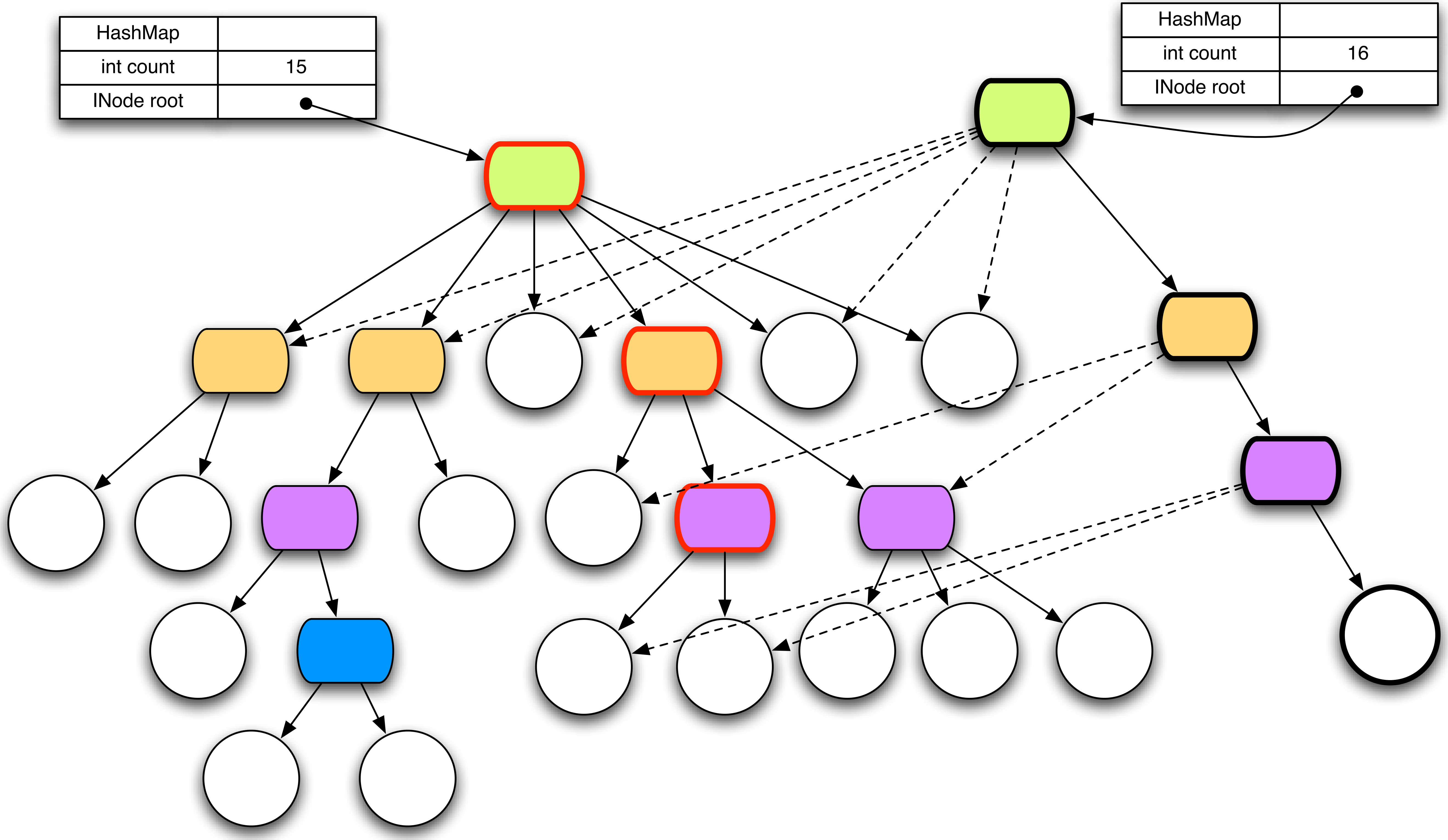
```
#inst "rfc-3339-format"
```

tagged element is a string in RFC-3339 format

```
#uuid "f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
```

tagged element is a canonical UUID string

2. persistent data structures



HashMap	
int count	15
INode root	●

HashMap	
int count	16
INode root	●

persistent data structures

immutable

“change” by function application

maintain performance guarantees

full-fidelity old versions

transience vs. persistence

characteristic	update-in-place	persistent
sharing	difficult	trivial
distribution	difficult	easy
concurrent access	difficult	trivial
access pattern	eager	eager or lazy
caching	difficult	easy
examples	Java, .NET collections relational databases NoSQL databases	Clojure, F# collections Datomic database

vectors

```
(def v [42 :rabbit [1 2 3]])

(v 1) -> :rabbit

(peek v) -> [1 2 3]

(pop v) -> [42 :rabbit]

(subvec v 1) -> [:rabbit [1 2 3]]
```

maps

```
(def m {:a 1 :b 2 :c 3})


(m :b) -> 2

(:b m) -> 2


(keys m) -> (:a :b :c)


(assoc m :d 4 :c 42) -> {:d 4, :a 1, :b 2, :c 42}


(dissoc m :d) -> {:a 1, :b 2, :c 3}


(merge-with + m {:a 2 :b 3}) -> {:a 3, :b 5, :c 3}
```

nested structure

```
(def jdoe {:name "John Doe",
           :address {:zip 27705, ...}})

(get-in jdoe [:address :zip])
-> 27705

(assoc-in jdoe [:address :zip] 27514)
-> {:name "John Doe", :address {:zip 27514}}

(update-in jdoe [:address :zip] inc)
-> {:name "John Doe", :address {:zip 27706}}
```

sets

```
(use clojure.set)
(def colors #{"red" "green" "blue"})
(def moods #{"happy" "blue"})

(disj colors "red")
-> #{"green" "blue"}

(difference colors moods)
-> #{"green" "red"}

(intersection colors moods)
-> #{"blue"}

(union colors moods)
-> #{"happy" "green" "red" "blue"}
```

3. sequences

first / rest /cons

```
(first [1 2 3])
-> 1

(rest [1 2 3])
-> (2 3)

(cons "hello" [1 2 3])
-> ("hello" 1 2 3)
```

take / drop

```
(take 2 [1 2 3 4 5])
-> (1 2)

(drop 2 [1 2 3 4 5])
-> (3 4 5)
```

predicates

```
(every? odd? [1 3 5])
-> true

(not-every? even? [2 3 4])
-> true

(not-any? zero? [1 2 3])
-> true

(some nil? [1 nil 2])
-> true
```

lazy and infinite

```
(set! *print-length* 5)
-> 5

(iterate inc 0)
-> (0 1 2 3 4 ...)

(cycle [1 2])
-> (1 2 1 2 1 ...)

(repeat :d)
-> (:d :d :d :d :d ...)
```

map / filter / reduce

```
(range 10)
-> (0 1 2 3 4 5 6 7 8 9)

(filter odd? (range 10))
-> (1 3 5 7 9)

(map odd? (range 10))
-> (false true false true false true
false true false true)

(reduce + (range 10))
-> 45
```

seqs work everywhere

collections

directories

files

XML

JSON

result sets

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

4. transducers

transducers

composable algorithmic transformations

independent of source/destination context

- element transformation only
- N->M

no intermediate aggregates

reducing fns

```
;; reducing function signature
whatever, input -> whatever

(reduce + [2 3 4])
=> 9
```

transformation needed

```
(def data [[1 -1 2] [3 4 5 -2]])
(reduce + data)
=> ClassCastException
```

collections of numbers
are not numbers

transducer

```
(def data [[1 -1 2] [3 4 5 -2]])

;; transducer signature
(whatever, input -> whatever) ->
(whatever, input -> whatever)

(transduce cat + data)
=> 12
```

transforms the reduction,
not the input!

naming an xf (xform)

```clojure
(def data [[1 -1 2] [3 4 5 -2]])

(def pos-values
  "Concat the positive values into the algorithm"
  (comp cat (filter pos?)))

(transduce pos-values + data)
=> 15
```

transducers cost/benefit

Cost

- higher level of abstraction

Benefits

- performance
- clarity
- reuse
- a la carte and in place upgrade of sequence code

transducer generality

```
(def ch (a/chan 10 pos-entries))
(>!! ch [1 -2 3])

(<!! ch)
=> ?

(<!! ch)
=> ?
```

like a j.u.c.Queue
but more general

transducer generality

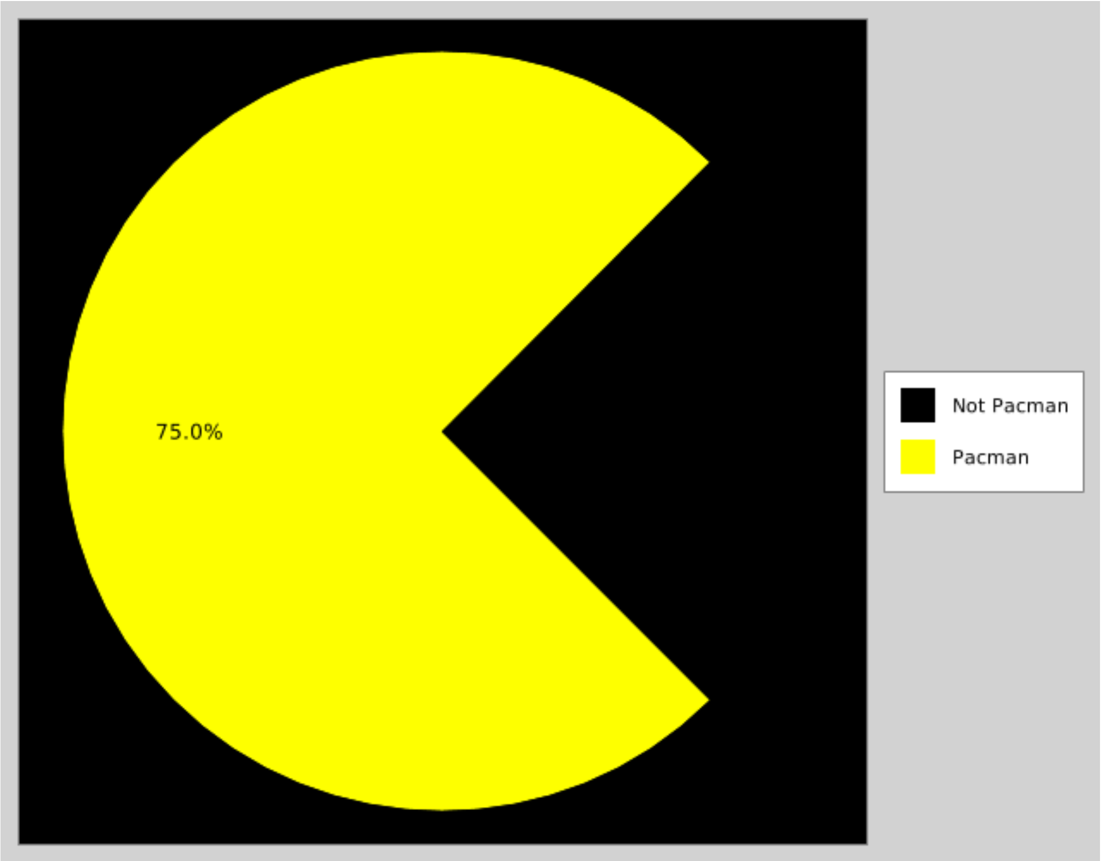
```
(def ch (a/chan 10 pos-entries))
(>!! ch [1 -2 3])

(<!! ch)
=> 1

(<!! ch)
=> 3
```

5. spec

expressivity

	Java types	spec
usage	mandatory	opt in
structure	classes	keyword maps etc.
predicates	🟥	arbitrary
composition	reference	reference
combination	🟥	boolean logic
syntax	🟥	regular expressions

clj-xchart and XChart



```clojure
(c/view
 (c/pie-chart
  [["Not Pacman" 1/4]
   ["Pacman" 3/4]]
  {:start-angle 225.0
   :plot {:background-color :black}
   :series [{:color :black} {:color :yellow}]}))
```

```clojure
(s/fdef c/xy-chart :args ::xy-chart-args)

(s/def ::xy-chart-args
  (s/and (s/cat :series ::xy/series
                :style (s/? ::sty/xy-styling))
         styling-matches-series?
         series-compatible-with-render-style?))

(s/def ::series (s/map-of ::series/series-name
                          ::series-elem))

(s/def ::series-elem
  (s/and (s/keys :req-un [::series/x ::series/y]
                 :opt-un [::series/error-bars ::style])
         series/axis-counts-match?
         series/data-compatible-with-render-style?))

(s/def ::y (s/every ::chartable-number :min-count 1))

(s/def ::chartable-number (s/and number? finite?))

(s/def ::xy-styling (s/merge ::styling-base
                             (s/keys :opt-in [::xy/render-style])))
```

```
(s/fdef c/xy-chart :args ::xy-chart-args)

(s/def ::xy-chart-args
  (s/and (s/cat :series ::xy/series
                :style (s/? ::sty/xy-styling))
         styling-matches-series?
         series-compatible-with-render-style?))

(s/def ::series (s/map-of ::series/series-name
                          ::series-elem))

(s/def ::series-elem
  (s/and (s/keys :req-un [::series/x ::series/y]
                 :opt-un [::series/error-bars ::style])
         series/axis-counts-match?
         series/data-compatible-with-render-style?))

(s/def ::y (s/every ::chartable-number :min-count 1))

(s/def ::chartable-number (s/and number? finite?))

(s/def ::xy-styling (s/merge ::styling-base
                             (s/keys :opt-in [::xy/render-style])))
```

regex for reusable syntax:
optional style follows series

```
(s/fdef c/xy-chart :args ::xy-chart-args)

(s/def ::xy-chart-args
  (s/and (s/cat :series ::xy/series
                :style (s/? ::sty/xy-styling))
         styling-matches-series?
         series-compatible-with-render-style?))

(s/def ::series (s/map-of ::series/series-name
                          ::series-elem))

(s/def ::series-elem
  (s/and (s/keys :req-un [::series/x ::series/y]
                 :opt-un [::series/error-bars ::style])
         series/axis-counts-match?
         series/data-compatible-with-render-style?))

(s/def ::y (s/every ::chartable-number :min-count 1))

(s/def ::chartable-number (s/and number? finite?))

(s/def ::xy-styling (s/merge ::styling-base
                             (s/keys :opt-in [::xy/render-style])))
```

required vs. optional fields

```
(s/fdef c/xy-chart :args ::xy-chart-args)

(s/def ::xy-chart-args
  (s/and (s/cat :series ::xy/series
                :style (s/? ::sty/xy-styling))
         styling-matches-series?
         series-compatible-with-render-style?))

(s/def ::series (s/map-of ::series/series-name
                          ::series-elem))

(s/def ::series-elem
  (s/and (s/keys :req-un [::series/x ::series/y]
                 :opt-un [::series/error-bars ::style])
         series/axis-counts-match?
         series/data-compatible-with-render-style?))

(s/def ::y (s/every ::chartable-number :min-count 1))

(s/def ::chartable-number (s/and number? finite?))

(s/def ::xy-styling (s/merge ::styling-base
                             (s/keys :opt-in [::xy/render-style]))
```

boolean combination
(& anonymous spec!)

```
(s/fdef c/xy-chart :args ::xy-chart-args)

(s/def ::xy-chart-args
  (s/and (s/cat :series ::xy/series
                :style (s/? ::sty/xy-styling))
         styling-matches-series?
         series-compatible-with-render-style?))

(s/def ::series (s/map-of ::series/series-name
                          ::series-elem))

(s/def ::series-elem
  (s/and (s/keys :req-un [::series/x ::series/y]
                 :opt-un [::series/error-bars ::style])
         series/axis-counts-match?
         series/data-compatible-with-render-style?))

(s/def ::y (s/every ::chartable-number :min-count 1))

(s/def ::chartable-number (s/and number? finite?))

(s/def ::xy-styling (s/merge ::styling-base
                             (s/keys :opt-in [::xy/render-style])))
```

collection size predicate

```
(s/fdef c/xy-chart :args ::xy-chart-args)

(s/def ::xy-chart-args
  (s/and (s/cat :series ::xy/series
                :style (s/? ::sty/xy-styling))
         styling-matches-series?
         series-compatible-with-render-style?))

(s/def ::series (s/map-of ::series/series-name
                          ::series-elem))

(s/def ::series-elem
  (s/and (s/keys :req-un [::series/x ::series/y]
                 :opt-un [::series/error-bars ::style])
         series/axis-counts-match?
         series/data-compatible-with-render-style?))

(s/def ::y (s/every ::chartable-number :min-count 1))

(s/def ::chartable-number (s/and number? finite?))

(s/def ::xy-styling (s/merge ::styling-base
                             (s/keys :opt-in [::xy/render-style]))
```

numeric range predicate

```
(s/fdef c/xy-chart :args ::xy-chart-args)

(s/def ::xy-chart-args
  (s/and (s/cat :series ::xy/series
                :style (s/? ::sty/xy-styling))
         styling-matches-series?
         series-compatible-with-render-style?))

(s/def ::series (s/map-of ::series/series-name
                          ::series-elem))

(s/def ::series-elem
  (s/and (s/keys :req-un [::series/x ::series/y]
                 :opt-un [::series/error-bars ::style])
         series/axis-counts-match?
         series/data-compatible-with-render-style?))

(s/def ::y (s/every ::chartable-number :min-count 1))

(s/def ::chartable-number (s/and number? finite?))

(s/def ::xy-styling (s/merge ::styling-base
                             (s/keys :opt-in [::xy/render-style])))
```

bring your own predicates

dev time power

	Java types	spec
instrumentation errors	no	yes
compilation errors	yes	no
DSL errors	no	yes
example generation	no	yes
automatic tests	no	yes
tool support	yes	yes

what is wrong with this?

```
(test/instrument [`xchart/xy-chart])

(xchart/xy-chart {"bad-doublings"
                      {:x [3 2 1] :y [4 5 7]
                       :style {:render-style :area}}})
```

instrumentation

```
(test/instrument [`xchart/xy-chart])

(xchart/xy-chart {"bad-doublings"
                  {:x [3 2 1] :y [4 5 7]
                   :style {:render-style :area}}})


ExceptionInfo Call to #'com.hypirion.clj-xchart/xy-chart
     did not conform to spec:
In: [0 "bad-doublings" 1]
fails spec: :com.hypirion.clj-xchart.specs.series.xy/series-elem
at: [:args :series 1] predicate: data-compatible-with-render-style?
:clojure.spec.test.alpha/caller {:file "example.clj",
                                 :line 25,
                                 :var-scope my.example/x}
```

instrumentation

```
(test/instrument [`xchart/xy-chart])

(-> (xchart/xy-chart {"bad-doublings"
                      {:x [3 2 1] :y [4 5 7]
                       :style {:render-style :area}}})
    (xchart/view))

ExceptionInfo Call to #'com.hypirion.clj-xchart/xy-chart
     did not conform to spec:
In: [0 "bad-doublings" 1]
fails spec: :com.hypirion.clj-xchart.specs.series.xy/series-elem
at: [:args :series 1] predicate: data-compatible-with-render-style?
:clojure.spec.test.alpha/caller {:file "example.clj",
                                 :line 25,
                                 :var-scope my.example/x}
```

where the data was bad

instrumentation

```
(test/instrument [`xchart/xy-chart])

(-> (xchart/xy-chart {"bad-doublings"
                      {:x [3 2 1] :y [4 5 7]
                       :style {:render-style :area}}})
    (xchart/view))

ExceptionInfo Call to #'com.hypirion.clj-xchart/xy-chart
     did not conform to spec:
In: [0 "bad-doublings" 1]
fails spec: :com.hypirion.clj-xchart.specs.series.xy/series-elem
at: [:args :series 1] predicate: data-compatible-with-render-style?
:clojure.spec.test.alpha/caller {:file "example.clj",
                                 :line 25,
                                 :var-scope my.example/x}
```

spec that failed

instrumentation

```
(test/instrument [`xchart/xy-chart])

(-> (xchart/xy-chart {"bad-doublings"
                      {:x [3 2 1] :y [4 5 7]
                       :style {:render-style :area}}})
    (xchart/view))

ExceptionInfo Call to #'com.hypirion.clj-xchart/xy-chart
     did not conform to spec:
In: [0 "bad-doublings" 1]
fails spec: :com.hypirion.clj-xchart.specs.series.xy/series-elem
at: [:args :series 1] predicate: data-compatible-with-render-style?
:clojure.spec.test.alpha/caller {:file "example.clj",
                                 :line 25,
                                 :var-scope my.example/x}
```

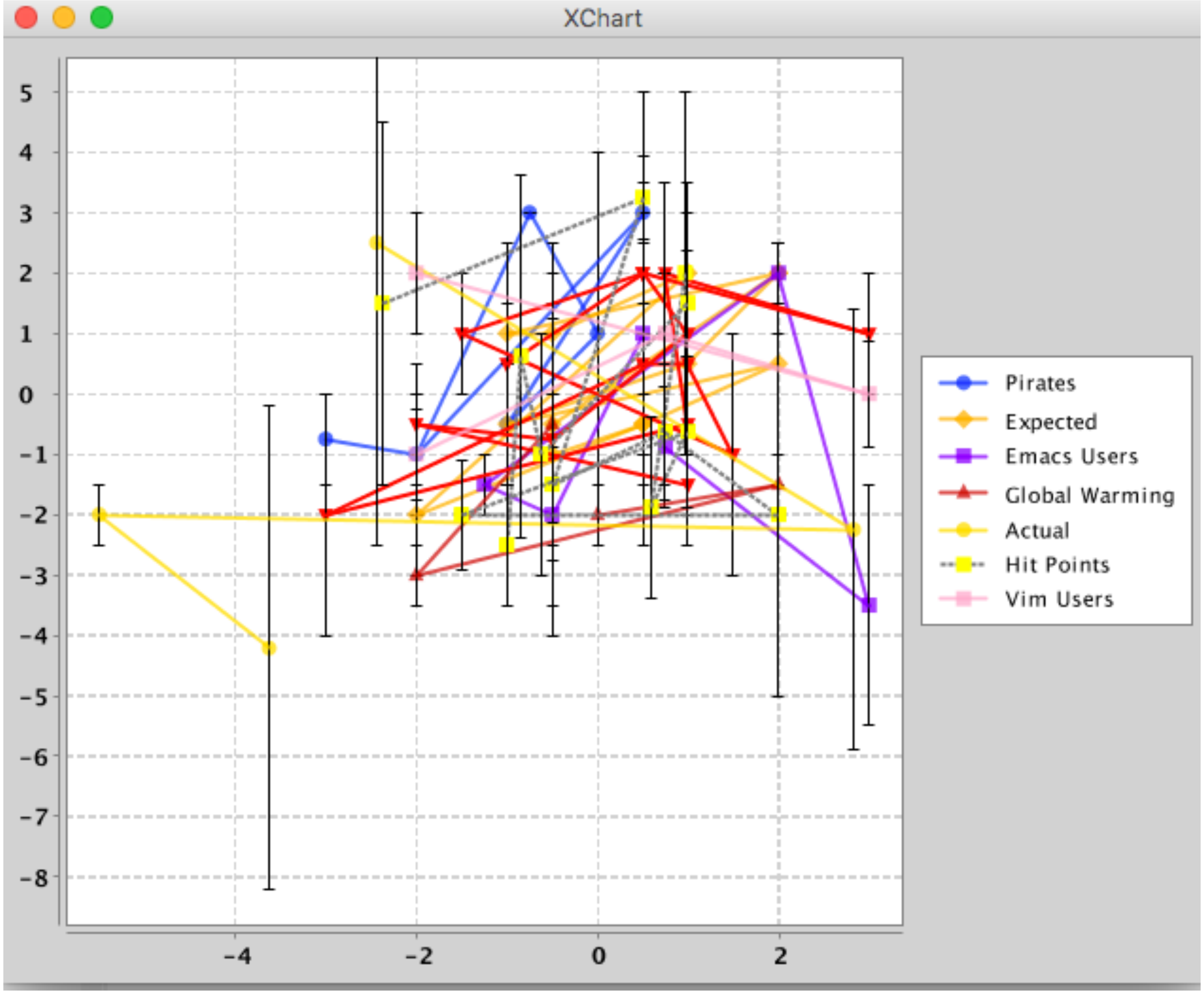
where the spec disagreed
with the data

instrumentation

```
(test/instrument [`xchart/xy-chart])

(-> (xchart/xy-chart {"bad-doublings"
                      {:x [3 2 1] :y [4 5 7]
                       :style {:render-style :area}}})
    (xchart/view))

ExceptionInfo Call to #'com.hypirion.clj-xchart/xy-chart
     did not conform to spec:
In: [0 "bad-doublings" 1]
fails spec: :com.hypirion.clj-xchart.specs.series.xy/series-elem
at: [:args :series 1] predicate: data-compatible-with-render-style?
:clojure.spec.test.alpha/caller {:file "example.clj",
                                 :line 25,
                                 :var-scope my.example/x}
```

where in the program
it happened

generating examples



generative testing

```
Thread 0 Crashed:: AppKit Thread  Dispatch queue: com.apple.main-thread
0   libsystem_kernel.dylib        0x00007fff8a94ff72 mach_msg_trap + 10
1   libsystem_kernel.dylib        0x00007fff8a94f3b3 mach_msg + 55
2   com.apple.CoreFoundation      0x00007fff966281c4 __CFRunLoopServiceMachPort + 212
3   com.apple.CoreFoundation      0x00007fff9662768c __CFRunLoopRun + 1356
4   com.apple.CoreFoundation      0x00007fff96626ed8 CFRunLoopRunSpecific + 296
5   com.apple.HIToolbox           0x00007fff8cb24935 RunCurrentEventLoopInMode + 235
6   com.apple.HIToolbox           0x00007fff8cb2476f ReceiveNextEventCommon + 432
7   com.apple.HIToolbox           0x00007fff8cb245af _BlockUntilNextEventMatchingListInModeWithFilter + 71
8   com.apple.AppKit              0x00007fff8aa50df6 _DPSNextEvent + 1067
9   com.apple.AppKit              0x00007fff8aa50226 -[NSApplication
_nextEventMatchingEventMask:untilDate:inMode:dequeue:] + 454
10  libosxapp.dylib               0x000000012687c3aa -[NSApplicationAWT
nextEventMatchingMask:untilDate:inMode:dequeue:] + 124
11  com.apple.AppKit              0x00007fff8aa44d80 -[NSApplication run] + 682
12  libosxapp.dylib               0x000000012687c14d +[NSApplicationAWT runAWTLoopWithApp:] + 156
13  libawt_lwawt.dylib            0x0000000126ebb55b -[AWTStarter starter:] + 905
14  com.apple.Foundation          0x00007fff8772afde __NSThreadPerformPerform + 279
15  com.apple.CoreFoundation      0x00007fff96648881 __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__ + 17
16  com.apple.CoreFoundation      0x00007fff96627fbc __CFRunLoopDoSources0 + 556
17  com.apple.CoreFoundation      0x00007fff966274df __CFRunLoopRun + 927
18  com.apple.CoreFoundation      0x00007fff96626ed8 CFRunLoopRunSpecific + 296
19  java                          0x0000000107893463 CreateExecutionEnvironment + 871
20  java                          0x000000010788f1ac JLI_Launch + 1952
21  java                          0x00000001078954c0 main + 101
22  java                          0x000000010788ea04 start + 52
```

https://github.com/stuarthalloway/clj-xchart/blob/master/examples/jvm_bug.repl

prod time power

	Java types	spec
validation	types only	all data
explanation	yes	no
conformance	no	yes
assertion	types only	all data

web service validator

```
(defn conform!
  [spec x]
  (when-not (s/valid? spec x)
    {:status 400
     :body {:cause (s/explain-str spec x)}}))
```

6. REPL

REPL

read: input stream -> data

eval: data -> data

print: data -> output stream

REPL Advantages

- Immediate interaction
  - “Faster than a speeding test”
- Interact with running programs
- No “pour concrete” phase
  - Copy code REPL dev session <-> program

Shell Limitations

	Shell	REPL
semantics	caveats	like programs
state	new abstractions	like programs
context	new wrappers	like programs
modifying code	new semantics	like programs
forward reference	new semantics	no (like programs!)
dependencies	new semantics	like programs
testing	new semantics	(should be) like programs
risks	classloaders confusing	<- yeah, that

openjdk.java.net/jeps/222

REPL Debugging

```
(defn foo
  [n]
  (cond (> n 40)① (+ n 20)②
        (> n 20)③ (- (first n) 20)④
        :else⑤ 0⑥))

(def n 24)

;; results evaluating with cursor at each position
① => false
② => 44
③ => true
④ => Broken! HaHa!
⑤ => :else
⑥ => 0
```

blog.cognitect.com/blog/2017/6/5/repl-debugging-no-stacktrace-required

7. core.async

http://swannodette.github.io/2013/08/02/100000-processes

problems

objects make terrible machines

function chains make poor machines

direct connections = tight coupling

callback hell in e.g. UI frameworks

so queues!

queue problems

consume real threads (e.g. JVM)

- don't play well with things that have thread-affinity

or you can't consume threads at all (e.g. JavaScript)

don't compose (e.g. JVM)

core.async

channels are better than queues

- don't: dictate process model
- do: composition with alt family

threads + go blocks are better than either alone

- and mix and match

Communicating Sequential Processes (CSP) model

search with 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)))))))))
```

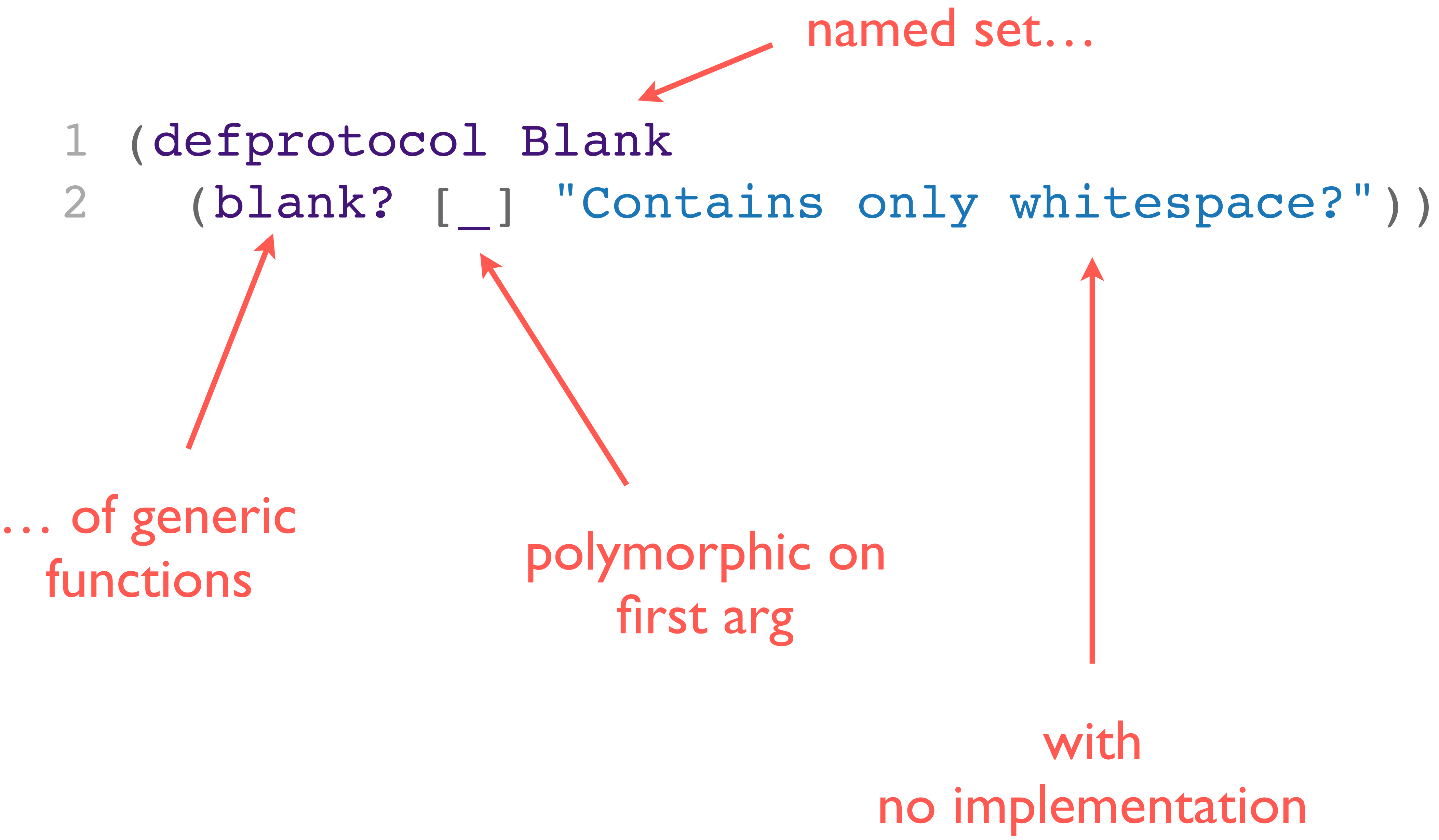
http://talks.golang.org/2012/concurrency.slide#50

```
4577564229292150809747676620914927402676658543329331224646181671857414658491709665999465566236034518
2474114890775540910374809790651568960932973763657908366442554603714211402350945593838464169205906664
0593763804077759260664515836237982120496867366347552888876410083519248290728143125529440753958783384
1681552013214282529921136058896240754916055788292919508334244074682311407545673715100189751445450024
5356942571782209220547590978629659840129267264897696745395548715727643425067187015062522256671401722 4
9117537695500791157876011699126282547698965682163620949096637419020461494587768878117377007178501924
2669333599710473603516505640653285066100280459838999696659002542671799652610246493447496372193022921
3249738840934524571557177835579618759452539388116559788174086383891493268713859864268950438580459752
8000707138877182570227318153725604842798622002811692909899973790373408402961809410864243130467958215
3536874357464456797583929080267525363841743493702250337820552383868653056581299496241833035767665508
0801026467570644424651306296512510755510793369261712169918676087013221656279804949716345455475280235
3278758954024312743113375530066341338311645803267525368689151248105483466144795990379811637997772648
3227620735785270658017230593823453895667682041809589668740160631544559298820937971024755242989933726
7093275627418904524043171091961306884757511040362153115542737305314254181719886334824141230272262927
2561880674455585270781057925333927240956106155644232723776391903741300584897022365197615062135132863
6790624756802436666202325256555852413768224151052712817925121216280737661431516212124355510578818445
5692154633517664889010333543645169597922779827850169933529455004066404739098937892757054023379201543
1244596660194613702236649608198018308837269369233238683156898827648836181533375301281837293137028933
0833071708978803853666692338311623192225611049834037370752171408632073824605135660999713919686784465
367082056787319521294323212910323183096498952617305555772273698548775241663928824381147345699630128
6476092364751030812516014708885115276123094108100767368892403037746466762568401925285533587390984221
8641724714272803492901662364367770746066255050594323351984714501612308735422102905778713717083685835
6341741522762856220232815572793117238514432222209929294218008485148503439521600894687226215805537060
6584295597125264900574760844965111278776777842151899713894714419217606145711367154491874638583225042
5880614465337458208880957385133815831215908897959662833684277915830571203815088296530161993920897 03
4976252514382234109413006474494314844420853793612730400926301876213749603986075129289763754017418584
8894813003699367096156199303862151632071557426072223248235812092976144472377431651852432246424107870
4764426600706552046038666815014572546547083922718397136526924759132921328939237631628293047719569028
9872375576625966271083053392338267274810139189251116483211279507821471341336976309631054217430491055
7470845811461951582938596955248090560072464878725101334702102368602648076117632463014572677255118018
8403694614893000703745866268855627198320139951661175370507321284847077704286592085813325445036917314
5697892803154766057455128889116174735684578848769209218604568233707209926809363766238133916757002440
8034481207027870657300335116042733355529870488381412007231770418847447615947713857202864234301654208
1478772586081299446435346823879610068218213663557481184202339765482379485049512395898526609664425919
7244391909947972201807124500495663868489998492683061263763037746461231986141096196422573686966365885
4422980196123290838577963751248700719328890261451679706842427174580795019934027309487917047626307746
6779785595849100976875147796308907799483209698135083942587417512479096759239151217495266472970030648
2178173763751391184243983577900933370781701982759586577417119590436169156373317742358716824718149020
0684543834458877960921287083076406491980467471896542517002083031530534593026362152261137104735063685
5117278906790754903581750697201834732569428599838564749229241297810977797699549599896237849980528348
5923316738186303587370429477942465725183113882361227430907967644675539423932605073412189976383180202
1132386474208990866812652870734416683755952448730811681009878946889425540981024087760773212097863814
7840728154350126473555213400293390764968251849462440120234539543731548266728391688069931499201665036
0947209353210786512040087438835461795641779932750208481957806617283658038031572104464084866610463115
9068874144795803490669053727611828875462595582627484177226879648828018479190895791218483504057652150
6125895149507540730469051953456023408466294044546852149368555157838964797288762886598529169228064396
5920230442238065554195137197248193772356734514715192450326961748517624623924502951618177568978102654
5117336903146541302627843152115962265602451680338689381594102446188468984930737476595488966965535260
2302585312522240476515240722856075879495217672249073297443480961095106357222181483447066651178109047
```

8. protocols

https://clojure.org/reference/protocols

protocols



```
(defprotocol Blank
  (blank? [_] "Contains only whitespace?"))
```

named set…

… of generic functions

polymorphic on first arg

with
no implementation

extending a protocol

```
(blank? "  ")
=> IllegalArgumentException ...

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

  nil
  (blank? [_] true))

(blank? "  ")
=> true

(blank? "hello")
=> false

(blank? nil)
=> true
```

extension options

`extend`: base functional implementation

`extend-protocol` to N types, nil

`extend-type` to N protocols

extend inline in **`deftype`** & **`defrecord`** definitions

extend inline anonymously with **`reify`**

for default: **`extend-protocol`** to **`Object`**

defrecord

```
(defrecord Foo [a b c])
-> user.Foo
```

named type
with slots

```
(def f (Foo. 1 2 3))
-> #'user/f
```

positional
constructor

```
(:b f)
-> 2
```

keyword access

casydht*

```
(class f)
-> user.Foo
```

plain ol' class

```
(supers (class f))
-> #{clojure.lang.IObj clojure.lang.IKeywordLookup java.util.Map
 clojure.lang.IPersistentMap clojure.lang.IMeta java.lang.Object
 java.lang.Iterable clojure.lang.ILookup clojure.lang.Seqable
 clojure.lang.Counted clojure.lang.IPersistentCollection
 clojure.lang.Associative}
```

*Clojure abstracts so you don't have to

from maps...

```
(def stu {:fname "Stu"
          :lname "Halloway"
          :address {:street "200 N Mangum"
                    :city "Durham"
                    :state "NC"
                    :zip 27701}})
```

data-oriented

```
(:lname stu)
=> "Halloway"
```

keyword access

```
(-> stu :address :city)
=> "Durham"
```

nested access

```
(assoc stu :fname "Stuart")
=> {:fname "Stuart", :lname "Halloway",
    :address ...}
```

update

nested update

```
(update-in stu [:address :zip] inc)
=> {:address {:street "200 N Mangum",
              :zip 27702 ...} ...}
```

...to records!

```
(defrecord Person [fname lname address])
(defrecord Address [street city state zip])
(def stu (Person. "Stu" "Halloway"
                  (Address. "200 N Mangum"
                            "Durham"
                            "NC"
                            27701)))
```

object-oriented

```
(:lname stu)
=> "Halloway"

(-> stu :address :city)
=> "Durham"
```

still data-oriented: everything works as before

type is there when you care

```
(assoc stu :fname "Stuart")
=> :user.Person{:fname "Stuart", :lname"Halloway",
                :address ...}

(update-in stu [:address :zip] inc)
=> :user.Person{:address {:street "200 N Mangum",
                          :zip 27702 ...} ...}
```

9. ClojureScript

https://clojurescript.org/

why ClojureScript

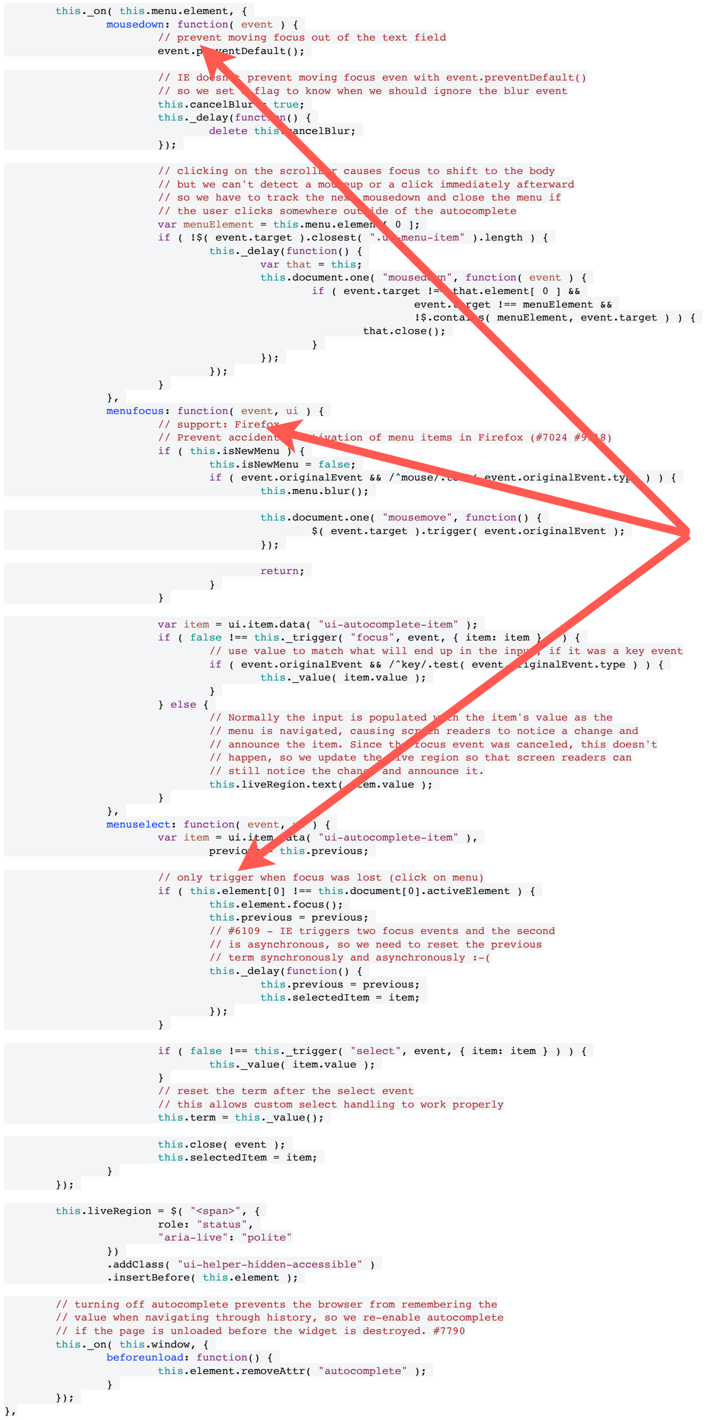
power of Clojure

share code across JS and JVM

Google Closure whole program optimization

core.async

no more callback hell



```
    this._on( this.menu.element, {
        mousedown: function( event ) {
            // prevent moving focus out of the text field
            event.preventDefault();

            // IE doesn't prevent moving focus even with event.preventDefault()
            // so we set a flag to know when we should ignore the blur event
            this.cancelBlur = true;
            this._delay(function() {
                delete this.cancelBlur;
            });

            // clicking on the scrollbar causes focus to shift to the body
            // but we can't detect a mouseup or a click immediately afterward
            // so we have to track the next mousedown and close the menu if
            // the user clicks somewhere outside of the autocomplete
            var menuElement = this.menu.element[ 0 ];
            if ( !$( event.target ).closest( ".ui-menu-item" ).length ) {
                this._delay(function() {
                    var that = this;
                    this.document.one( "mousedown", function( event ) {
                        if ( event.target !== that.element[ 0 ] &&
                                event.target !== menuElement &&
                                !$.contains( menuElement, event.target ) ) {
                            that.close();
                        }
                    });
                });
            }
        },
        menufocus: function( event, ui ) {
            // support: Firefox
            // Prevent accidental activation of menu items in Firefox (#7024 #9118)
            if ( this.isNewMenu ) {
                this.isNewMenu = false;
                if ( event.originalEvent && /^mouse/.test( event.originalEvent.type ) ) {
                    this.menu.blur();

                    this.document.one( "mousemove", function() {
                        $( event.target ).trigger( event.originalEvent );
                    });

                    return;
                }
            }

            var item = ui.item.data( "ui-autocomplete-item" );
            if ( false !== this._trigger( "focus", event, { item: item } ) ) {
                // use value to match what will end up in the input, if it was a key event
                if ( event.originalEvent && /^key/.test( event.originalEvent.type ) ) {
                    this._value( item.value );
                }
            } else {
                // Normally the input is populated with the item's value as the
                // menu is navigated, causing screen readers to notice a change and
                // announce the item. Since the focus event was canceled, this doesn't
                // happen, so we update the live region so that screen readers can
                // still notice the change and announce it.
                this.liveRegion.text( item.value );
            }
        },
        menuselect: function( event, ui ) {
            var item = ui.item.data( "ui-autocomplete-item" ),
                previous = this.previous;

            // only trigger when focus was lost (click on menu)
            if ( this.element[0] !== this.document[0].activeElement ) {
                this.element.focus();
                this.previous = previous;
                // #6109 - IE triggers two focus events and the second
                // is asynchronous, so we need to reset the previous
                // term synchronously and asynchronously :-(
                this._delay(function() {
                    this.previous = previous;
                    this.selectedItem = item;
                });
            }

            if ( false !== this._trigger( "select", event, { item: item } ) ) {
                this._value( item.value );
            }
            // reset the term after the select event
            // this allows custom select handling to work properly
            this.term = this._value();

            this.close( event );
            this.selectedItem = item;
        }
    });

    this.liveRegion = $( "<span>", {
                role: "status",
                "aria-live": "polite"
            })
            .addClass( "ui-helper-hidden-accessible" )
            .insertBefore( this.element );

    // turning off autocomplete prevents the browser from remembering the
    // value when navigating through history, so we re-enable autocomplete
    // if the page is unloaded before the widget is destroyed. #7790
    this._on( this.window, {
        beforeunload: function() {
            this.element.removeAttr( "autocomplete" );
        }
    });
},
```

jQuery Autocompleter:

reaction directly tied to events,

state smeared everywhere

https://github.com/jquery/jquery-ui/blob/9e00e00f3b54770faa0291d6ee6fc1dcbad028cb/ui/jquery.ui.autocomplete.js#L202-L307

```
(defn listen
  ([el type] (listen el type nil))
  ([el type f] (listen el type f (chan)))
  ([el type f out]
    (events/listen el (keyword->event-type type)
      (fn [e] (when f (f e)) (put! out e)))
    out))
```

ClojureScript Autocompleter: put events on channels

↓

state all in one place, handle by simple loop

```
(defn menu-proc [select cancel menu data]
  (let [ctrl (chan)
        sel  (->> (resp/selector
                    (resp/highlighter select menu ctrl)
                    menu data)
               (r/filter vector?)
               (r/map second))]
    (go (let [[v sc] (alts! [cancel sel])]
          (do (>! ctrl :exit)
            (if (or (= sc cancel)
                    (= v ::resp/none))
              ::cancel
              v))))))
```

“blocking” operations

http://swannodette.github.io/2013/08/17/comparative/

calling JavaScript

method call

```
(.write js/document "Hello, world!")
```

read field

```
(def page-title (.-title js/document))
```

null this

```
(def green (.color js/Raphael "#00ff00"))
(def green (Raphael/color "#00ff00"))
```

write field

```
(set! (.-title js/document) "New Page Title")
```

constructor

```
(def date (js/Date. 2013 3 17))
```

try/catch

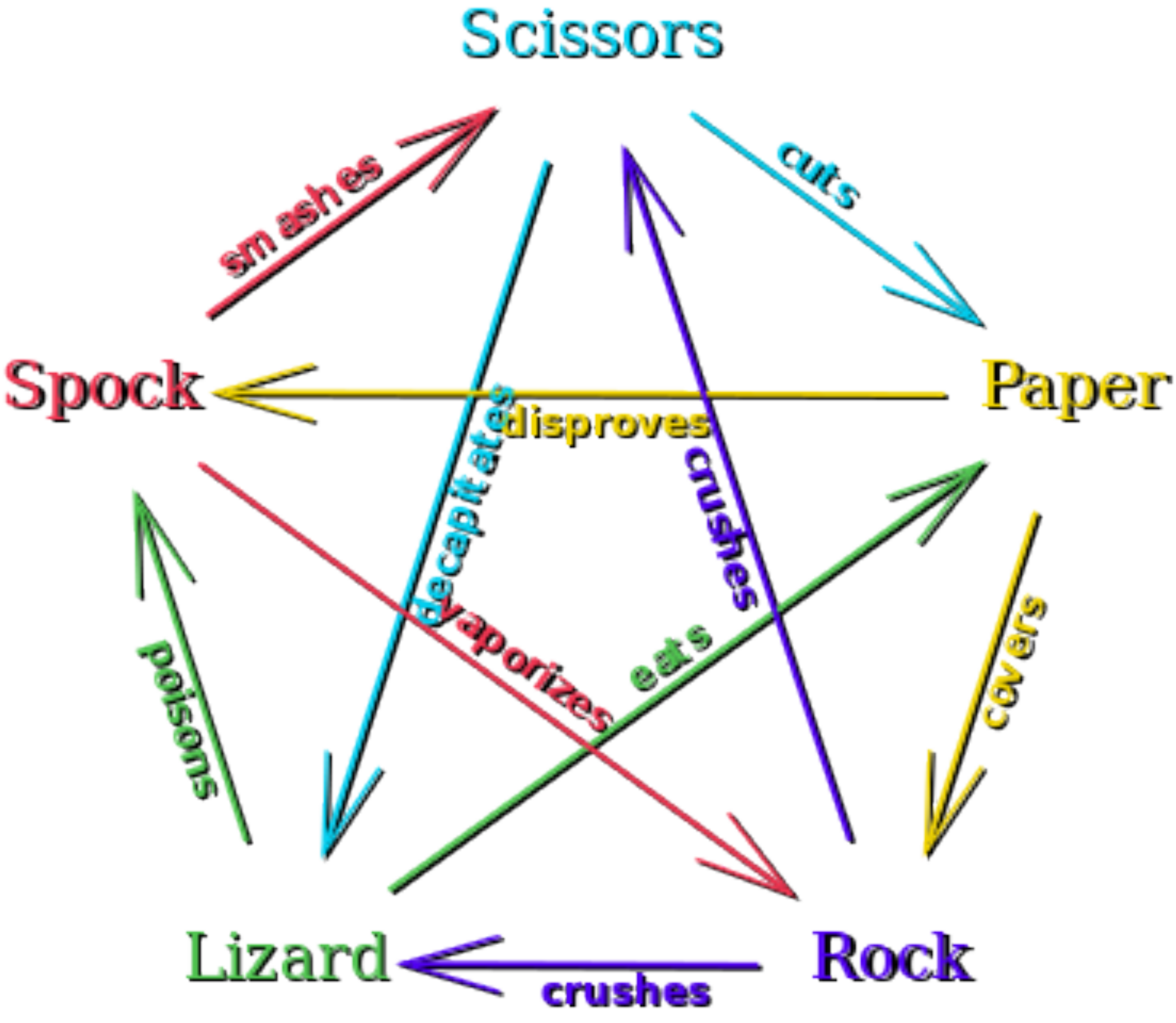
```
(try
  #_code
  (catch js/Error e
    (.log js/console (.-message e)))
    (finally
      #_cleanup))
```

Calling ClojureScript

```clojure
;; ClojureScript
(ns com.example.your-project)

(defn ^:export hello [name]
  (str "Hello, " name))
```

```javascript
// JavaScript
com.example.your_project.hello("Computer");
//=> "Hello, Computer"
```

10. logic


Scissors
cuts
Paper
covers
Rock
crushes
Lizard
poisons
Spock
smashes
disproves
vaporizes
eats
decapitates
crushes

relations, facts, and query

```
(defrel rps winner defeats loser)

(fact rps :scissors :cut :paper)
(fact rps :paper :covers :rock)
...
(fact rps :rock :breaks :scissors)

(run* [verb]
      (fresh [winner]
             (rps winner verb :paper)))
```

generic search

relation slots can be inputs
or outputs

relations, facts, and query

```
(defrel rps winner defeats loser)

(fact rps :scissors :cut :paper)
(fact rps :paper :covers :rock)
...
(fact rps :rock :breaks :scissors)

(run* [winner]
      (fresh [verb loser]
             (rps winner verb loser)))
```

generic search

different bindings,
different query!

example database

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

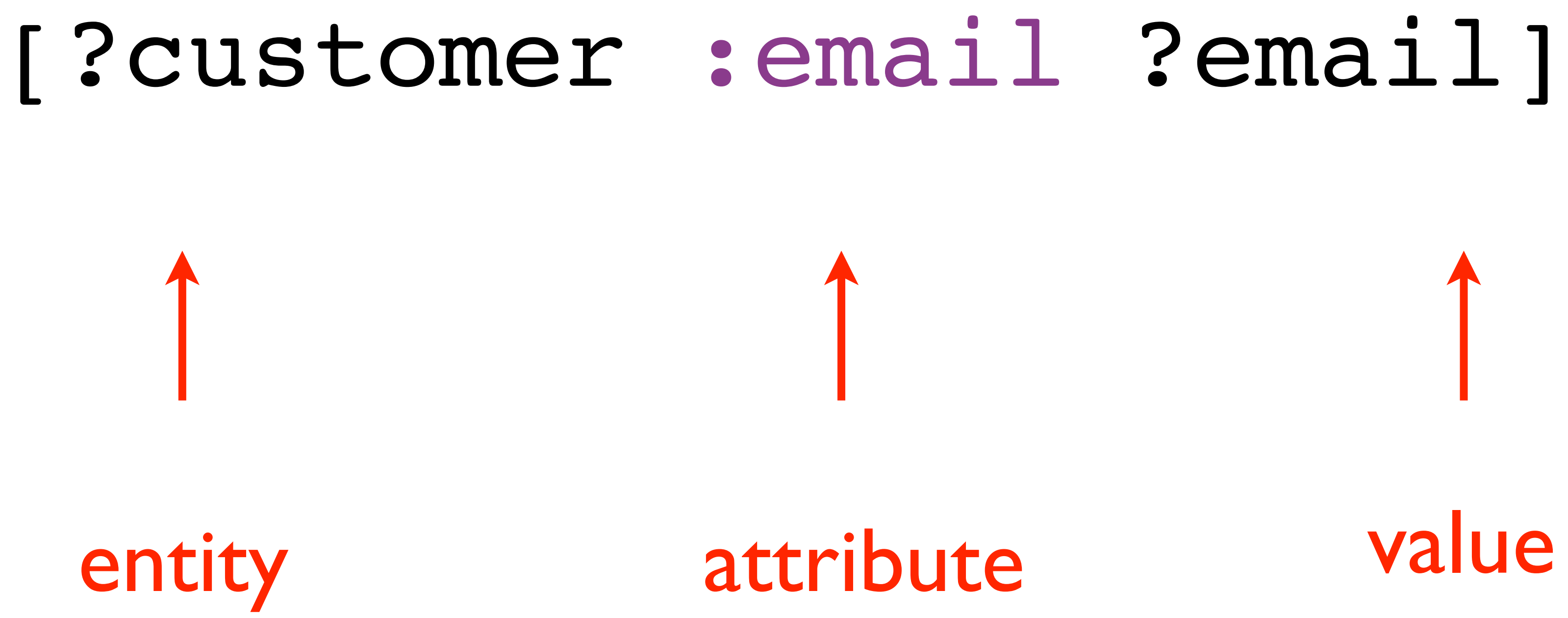
data pattern

Constrains the results returned,
binds variables

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

data pattern

Constrains the results returned,
binds variables



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

data pattern

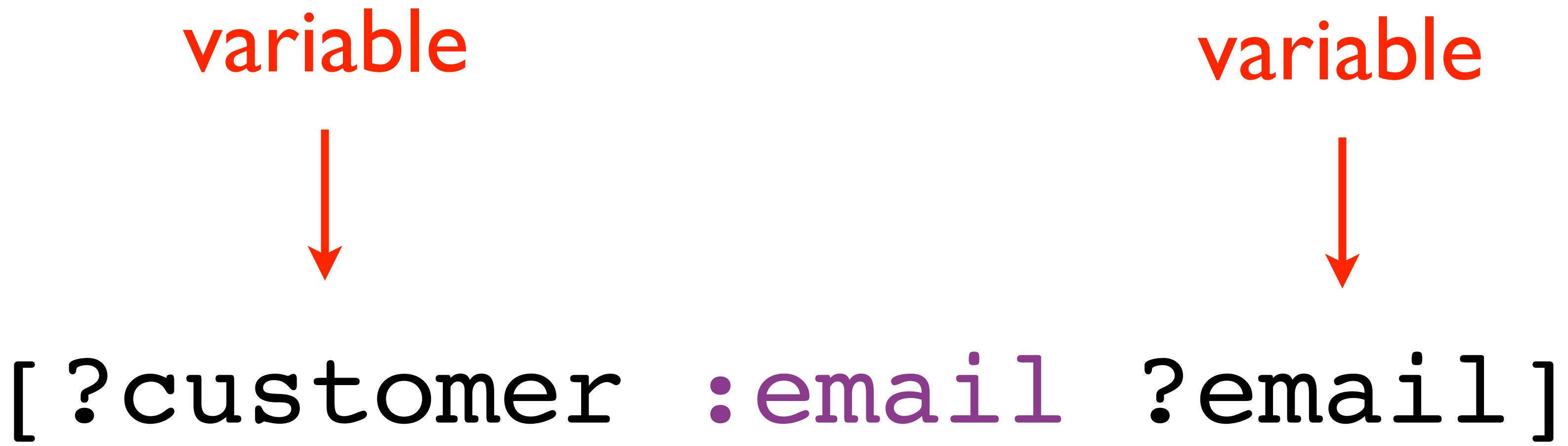
Constrains the results returned,
binds variables

constant

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

data pattern

Constrains the results returned,
binds variables



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

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

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

constants anywhere

“Find a particular customer’s email”

```
[42 :email ?email]
```

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

```
[42 :email ?email]
```

variables anywhere

“What attributes does customer 42 have?

```
[42 ?attribute]
```

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

```
[42 ?attribute]
```

variables anywhere

“What attributes and values does customer 42 have?

```
[42 ?attribute ?value]
```

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

```
[42 ?attribute ?value]
```

where clause

data pattern

```
[:find ?customer
 :where [?customer :email]]
```

find clause

variable to
return

```
[:find ?customer
 :where [?customer :email]]
```

implicit join

"Find all the customers who have placed orders."

```
[:find ?customer
 :where [?customer :email]
        [?customer :orders]]
```

predicates

Functional constraints that can appear in a :where clause

```
[(< 50 ?price)]
```

adding a predicate

“Find the expensive items”

```
[:find ?item
 :where [?item :item/price ?price]
        [(< 50 ?price)]]
```

functions

*Take bound variables as inputs
and bind variables with output*

```
[(shipping ?zip ?weight) ?cost]
```

function args

```
[(shipping ?zip ?weight) ?cost]
```

bound inputs

function returns

```
[(shipping ?zip ?weight) ?cost]
```

bind return
values

calling a function

“Find me the customer/product combinations where the shipping cost dominates the product cost.”

```
[: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)]]
```

calling a function

“Find me the customer/product combinations where the shipping cost dominates the product cost.”

navigate from customer to zip

```
[: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)]]
```

calling a function

“Find me the customer/product combinations where the shipping cost dominates the product cost.”

get product facts needed *during query*

```
[: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)]]
```

calling a function

“Find me the customer/product combinations where the shipping cost dominates the product cost.”

call web service
to bind shipCost

```
[: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)]]
```

calling a function

"Find me the customer/product combinations where the shipping cost dominates the product cost."

```
[: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)]]
```

constrain price

calling a function

“Find me the customer/product combinations where the shipping cost dominates the product cost.”

return customer, product pairs

```
[: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)]]
```

protocols

targeting platforms

REPL

immutability

spec

seqs

reducers

core.async

refs

datalog

edn

core.logic

11. unified succession model

https://stackoverflow.com/questions/39309911/what-is-the-unified-update-model/39310567#39310567

in-place effects

subprograms are machines

programming: sticking together a bunch of moving parts

reasonable if memory is *very* (1970s) expensive

a better way: refs

new memories use new places

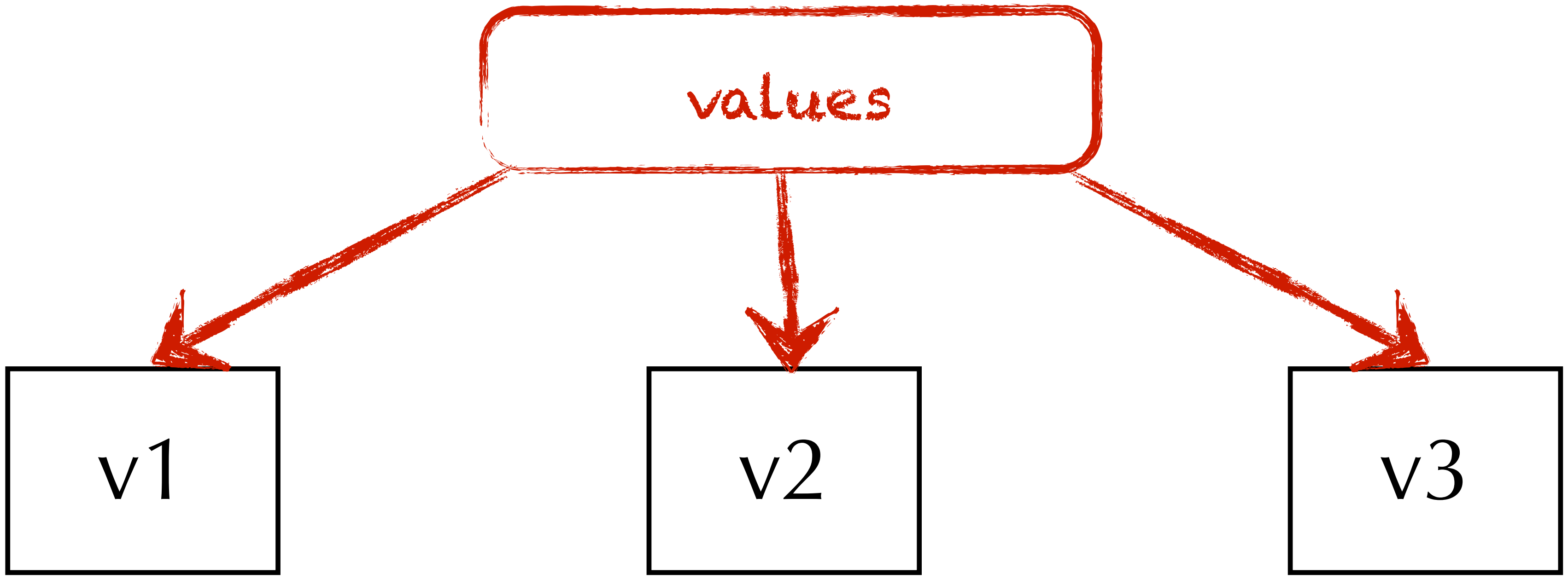
change encapsulated by constructors

references refer to point-in-time value

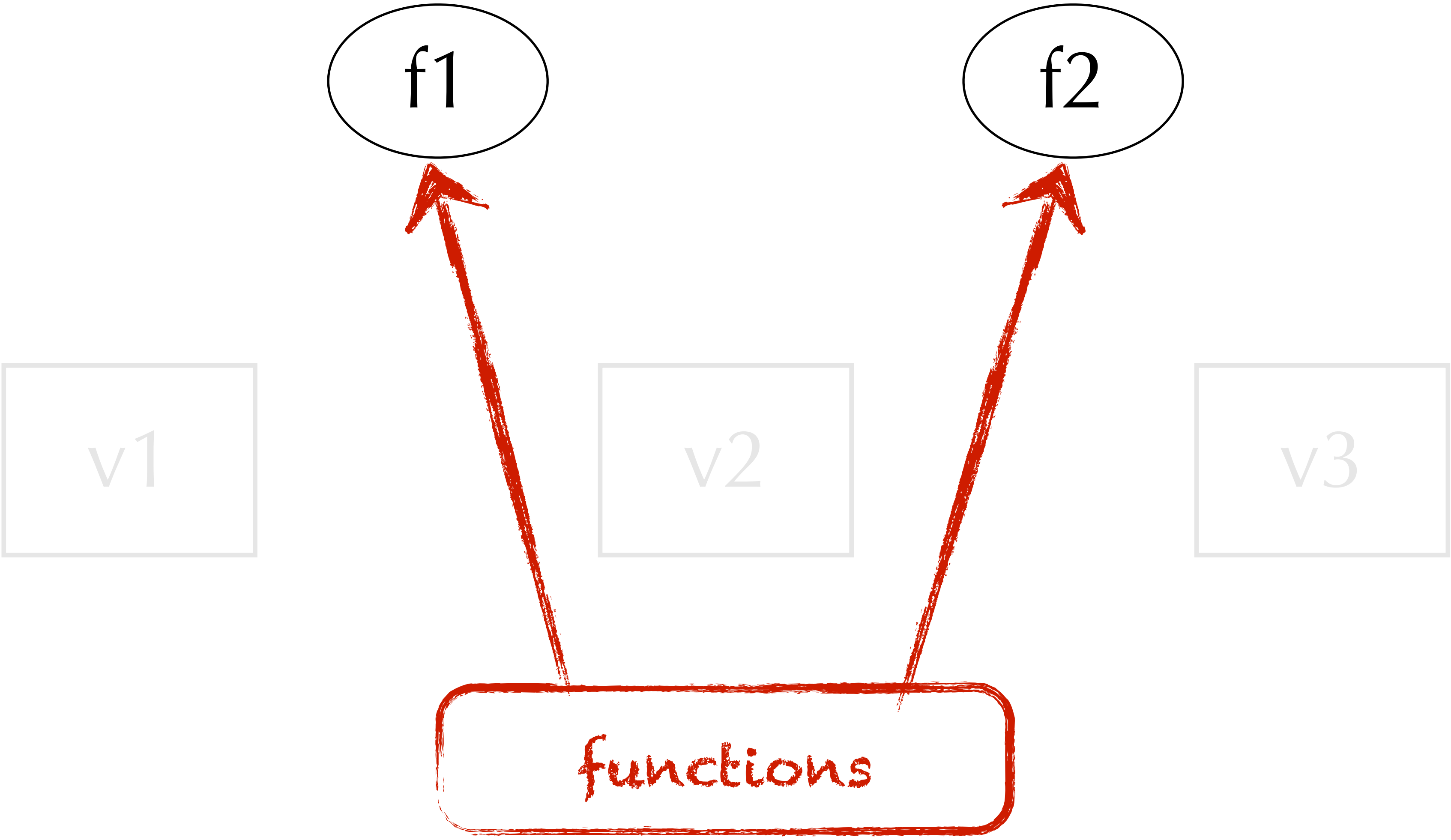
references see a *succession of values*

compatible with many update semantics

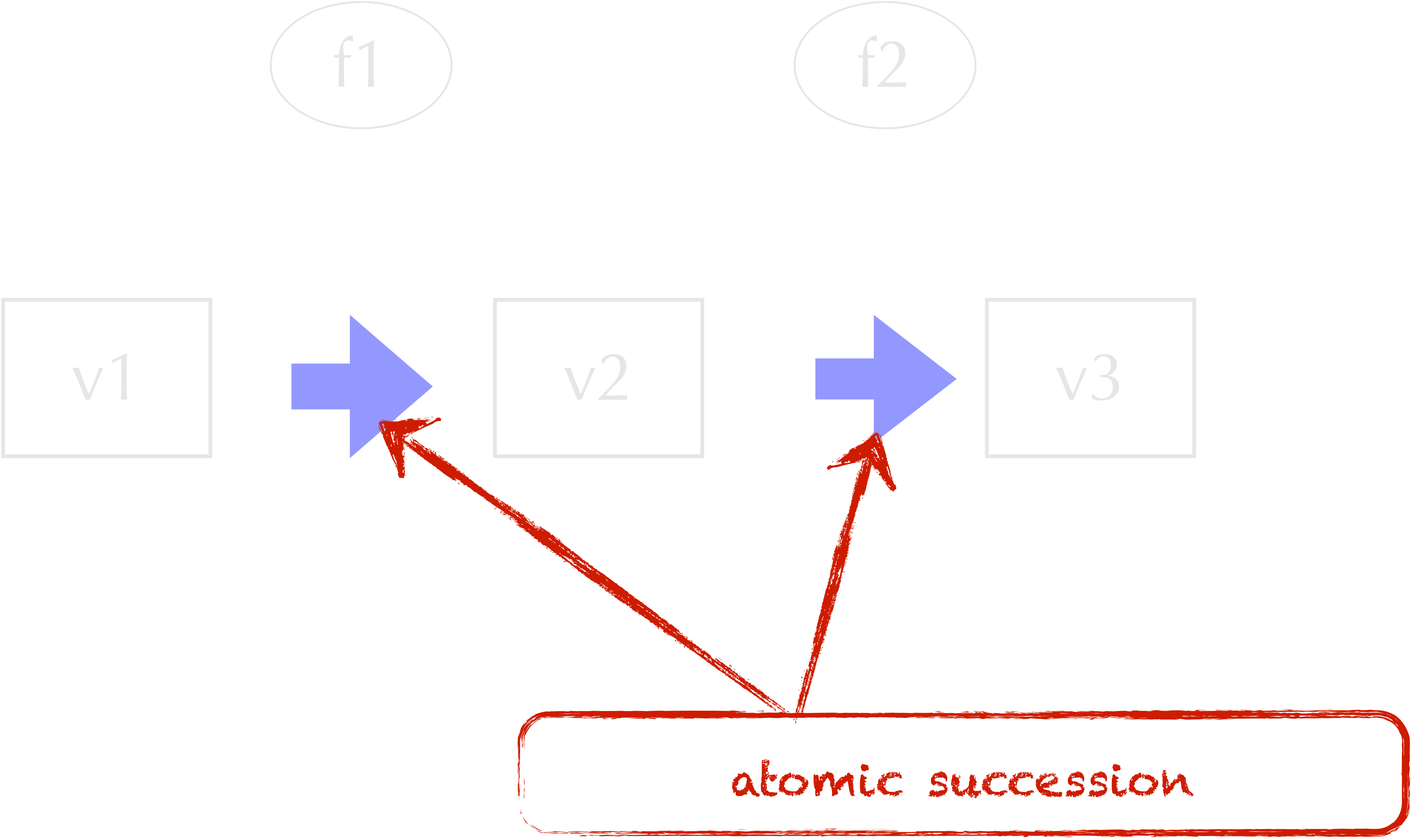
value succession



value succession

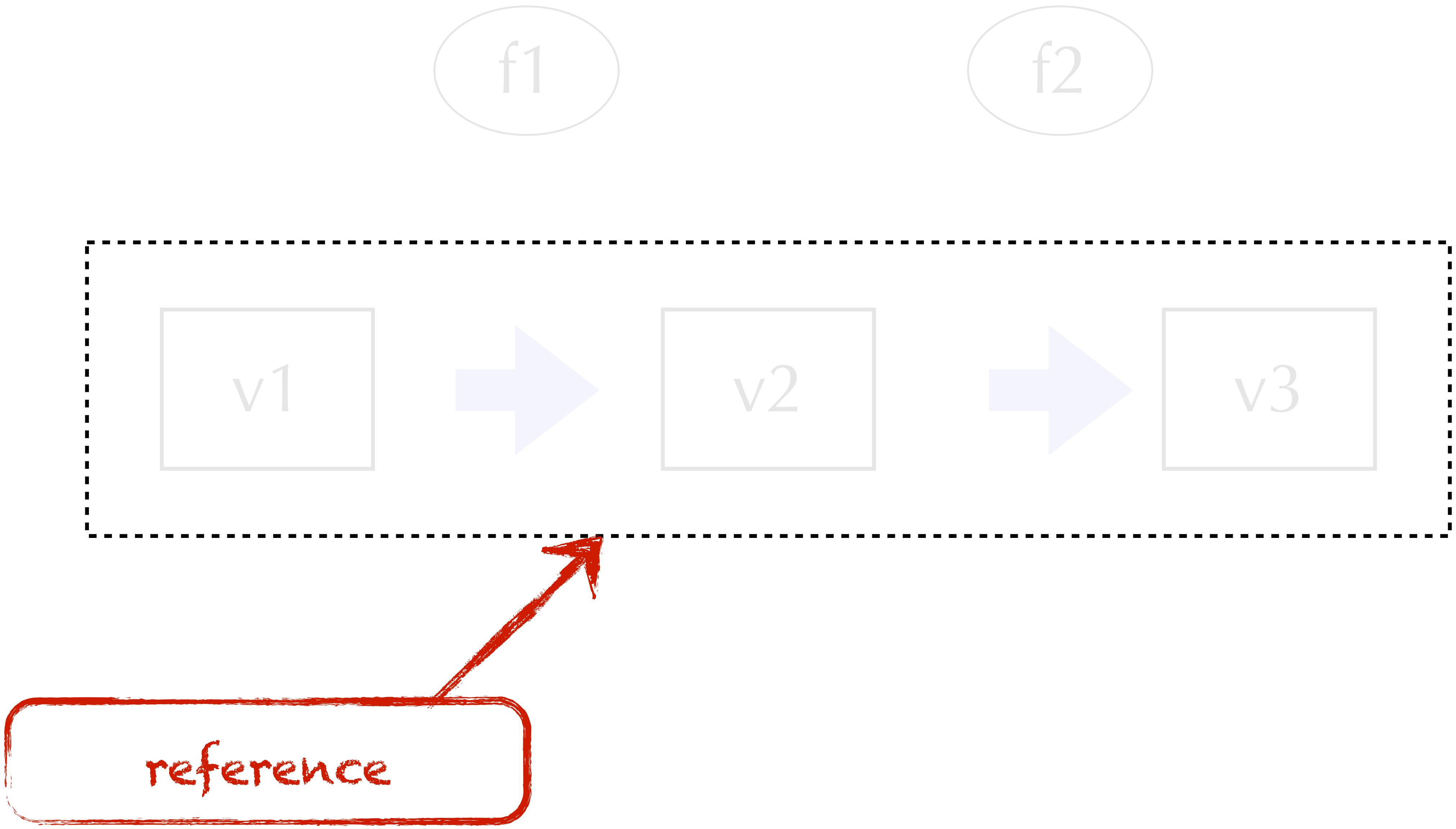


value succession

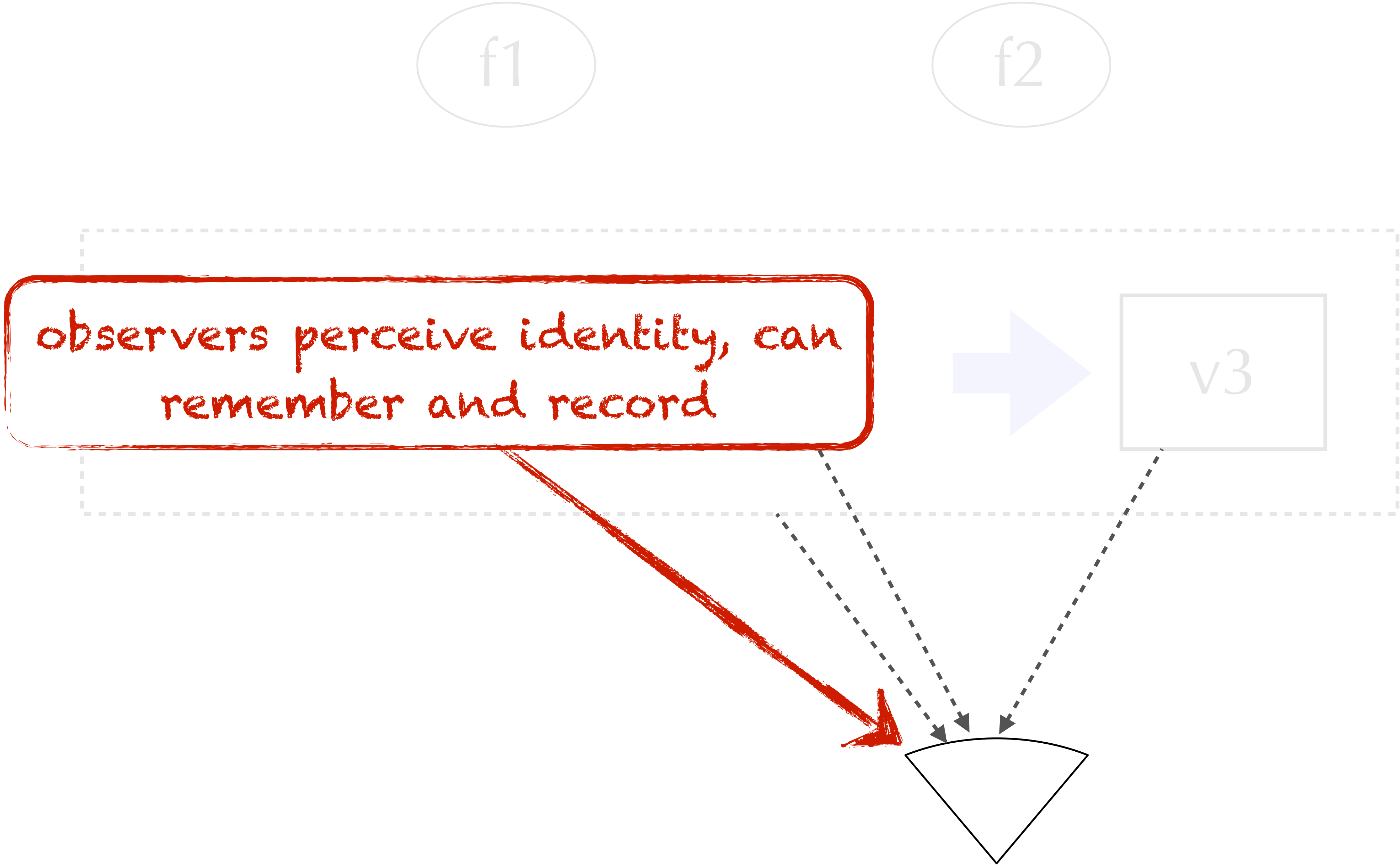


atomic succession

reference

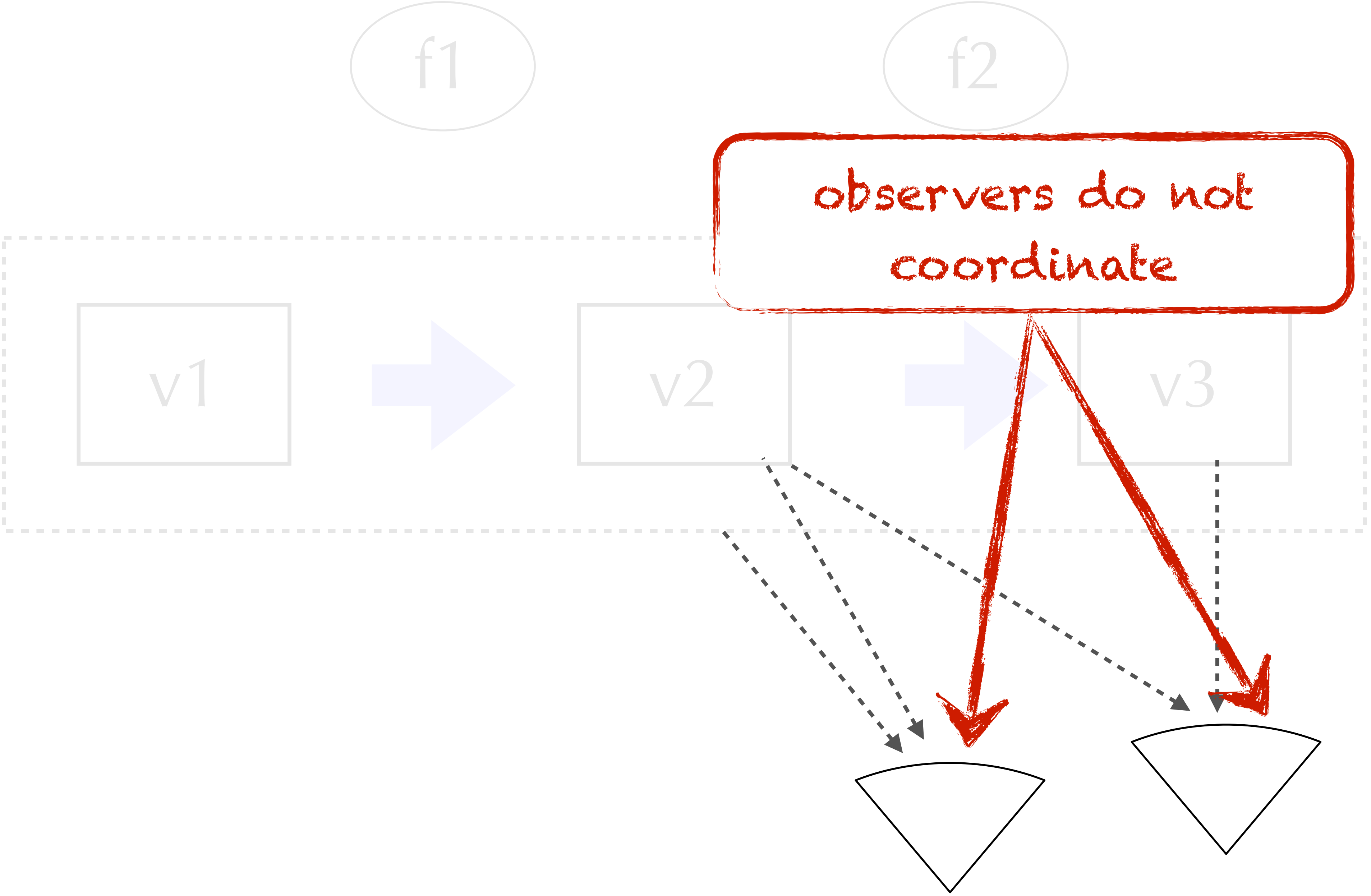


observers



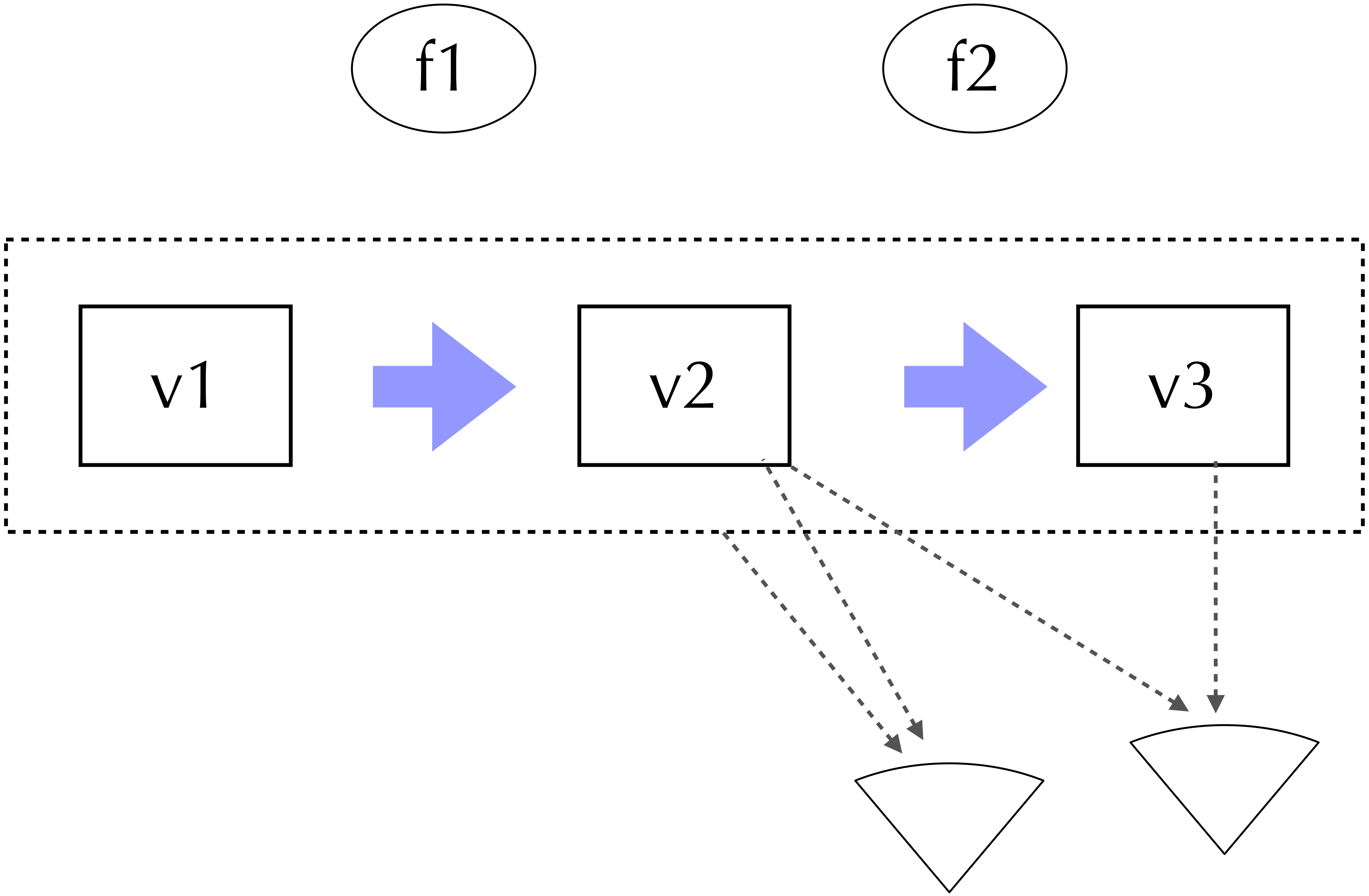
observers perceive identity, can remember and record

no coordination



observers do not coordinate

unified succession model



atoms

```
(def counter  (atom 0))
(swap! counter + 10)
```

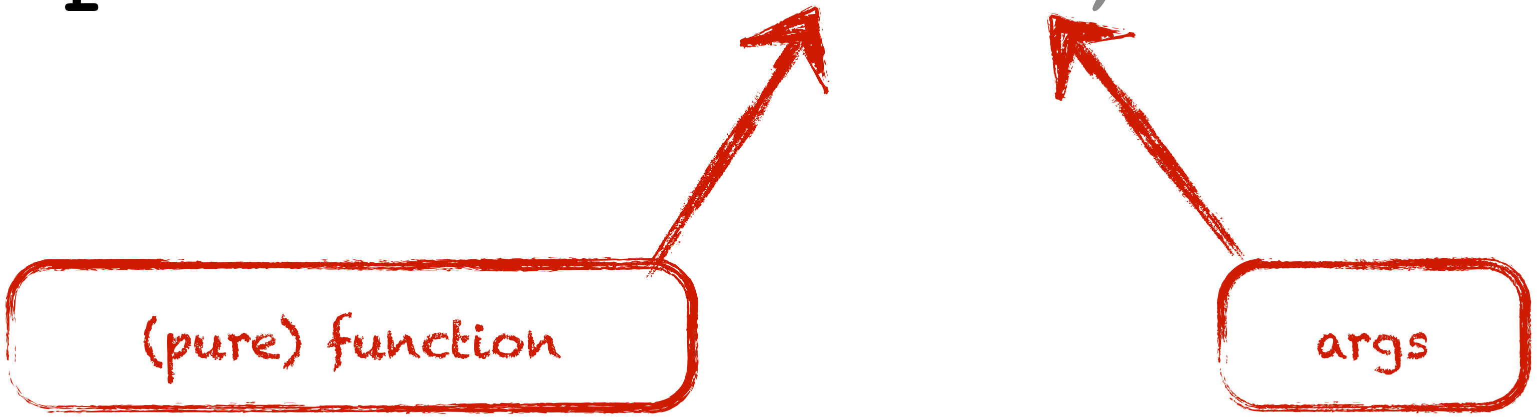
atoms

reference constructor

```
(def counter  (atom 0))
(swap! counter + 10)
```

atoms

```
(def counter  (atom 0))
(swap! counter + 10)
```



(pure) function

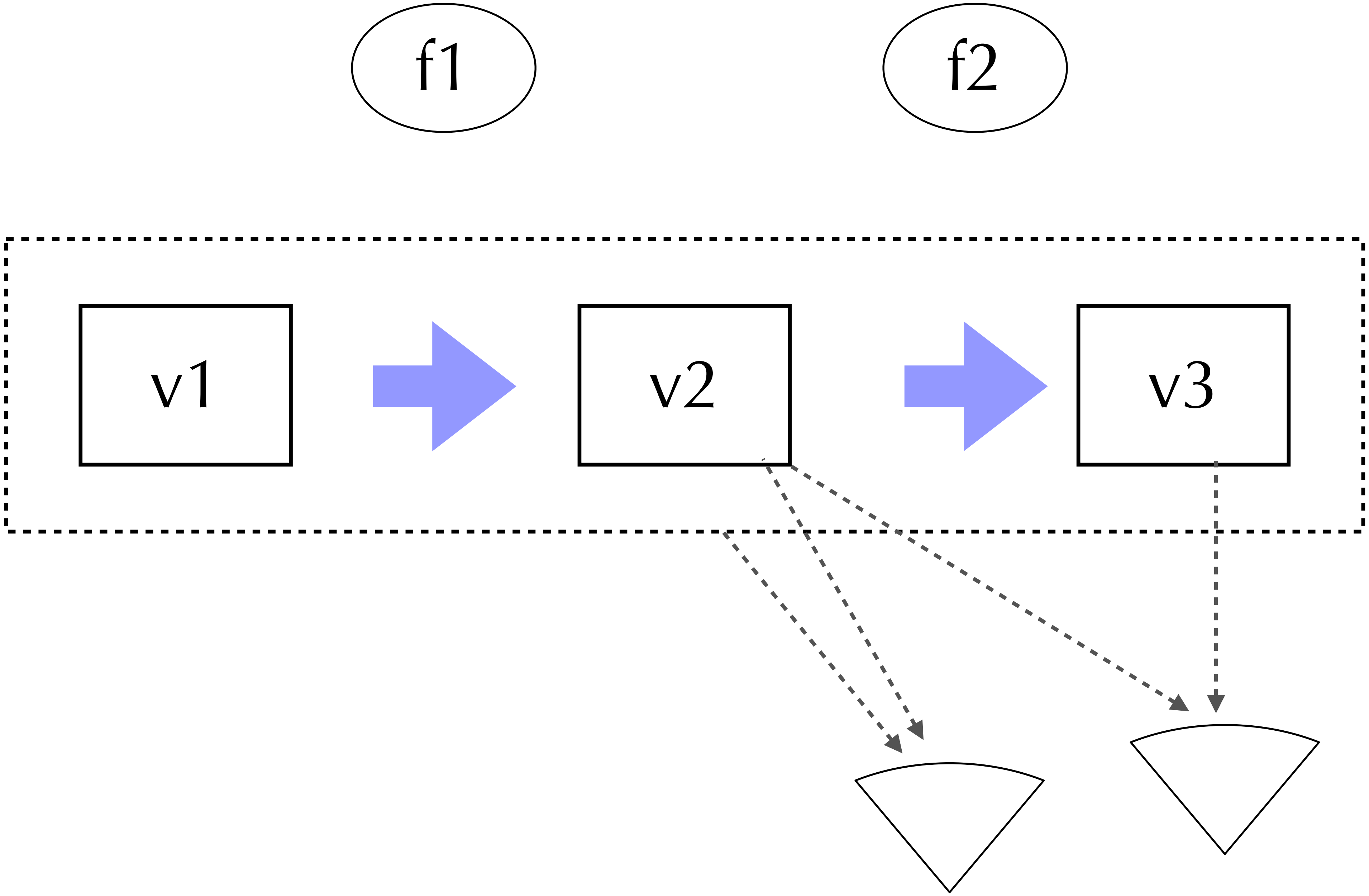
args

atoms

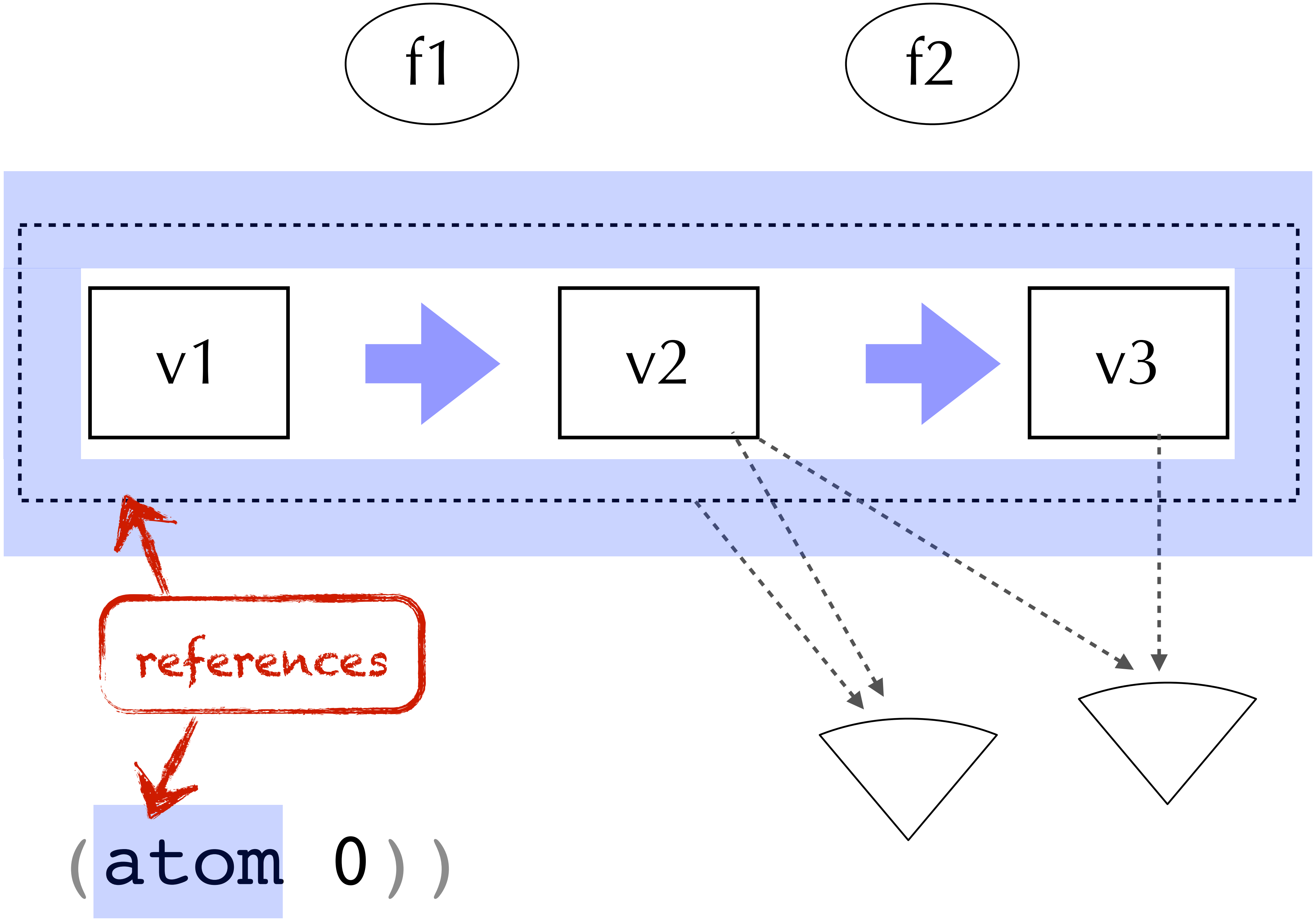
```
(def counter  (atom 0))
(swap! counter + 10)
```

atomic succession

atoms

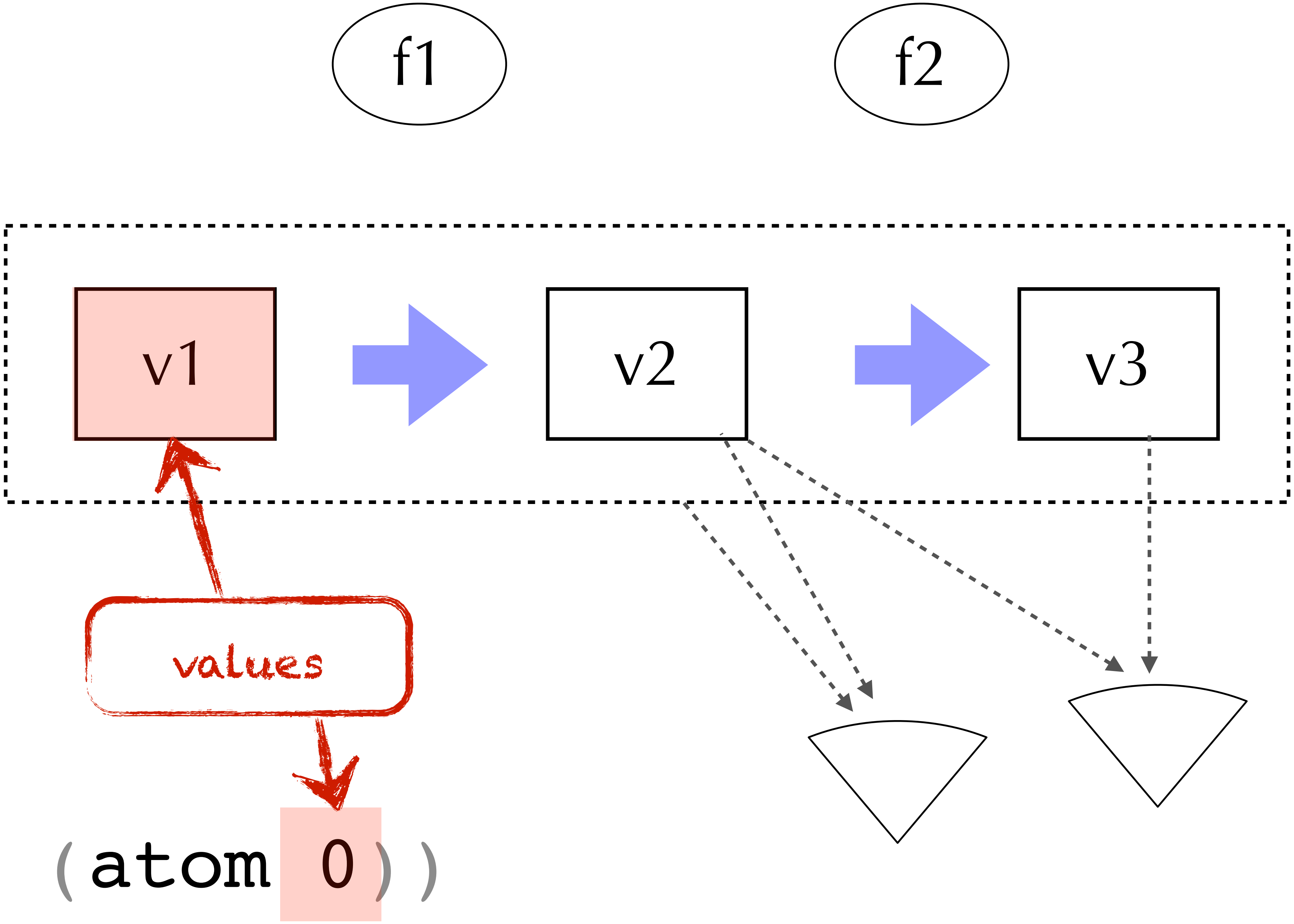


atoms



```
(atom 0))
```

atoms



f1

f2

v1

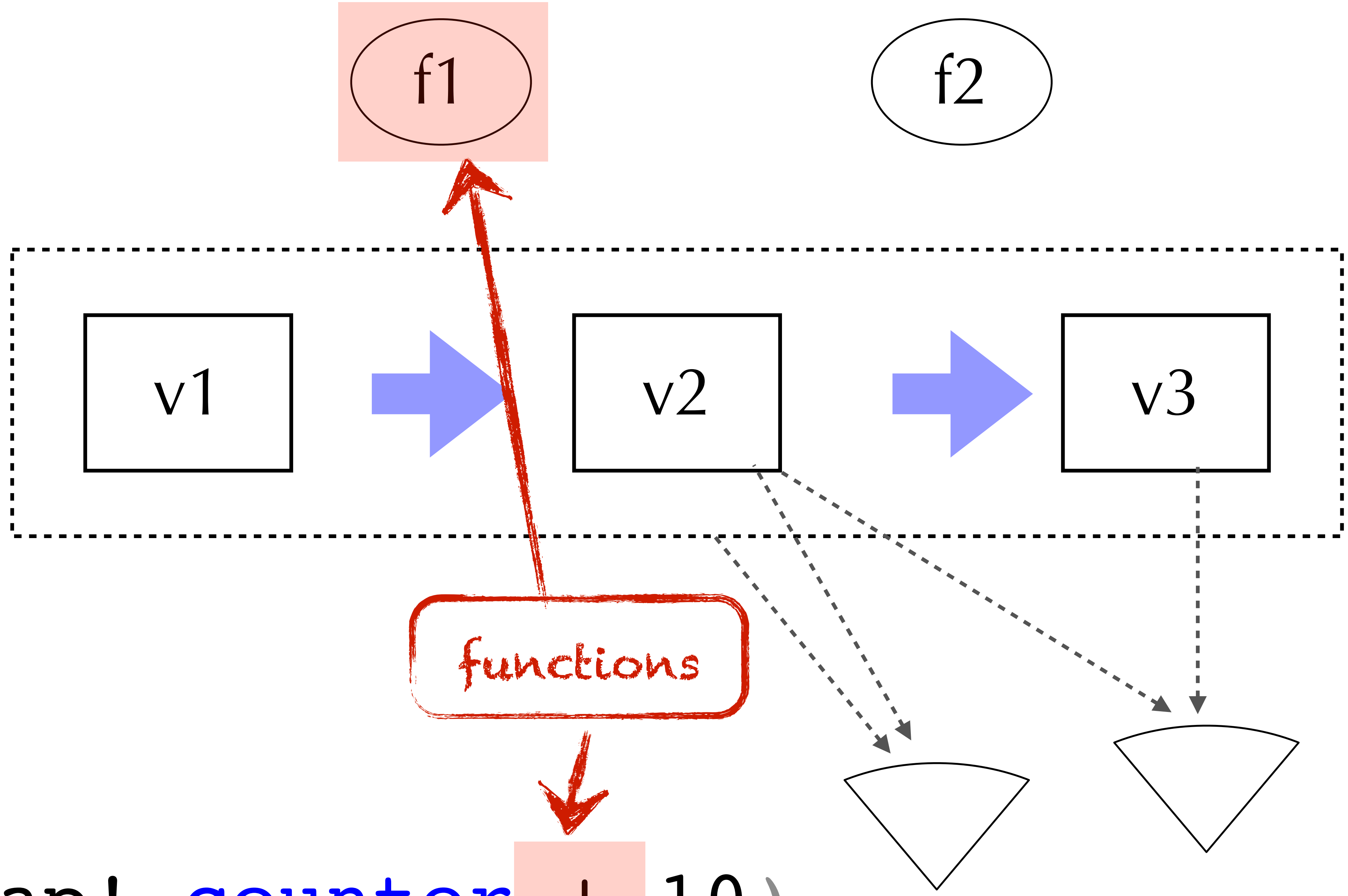
v2

v3

values

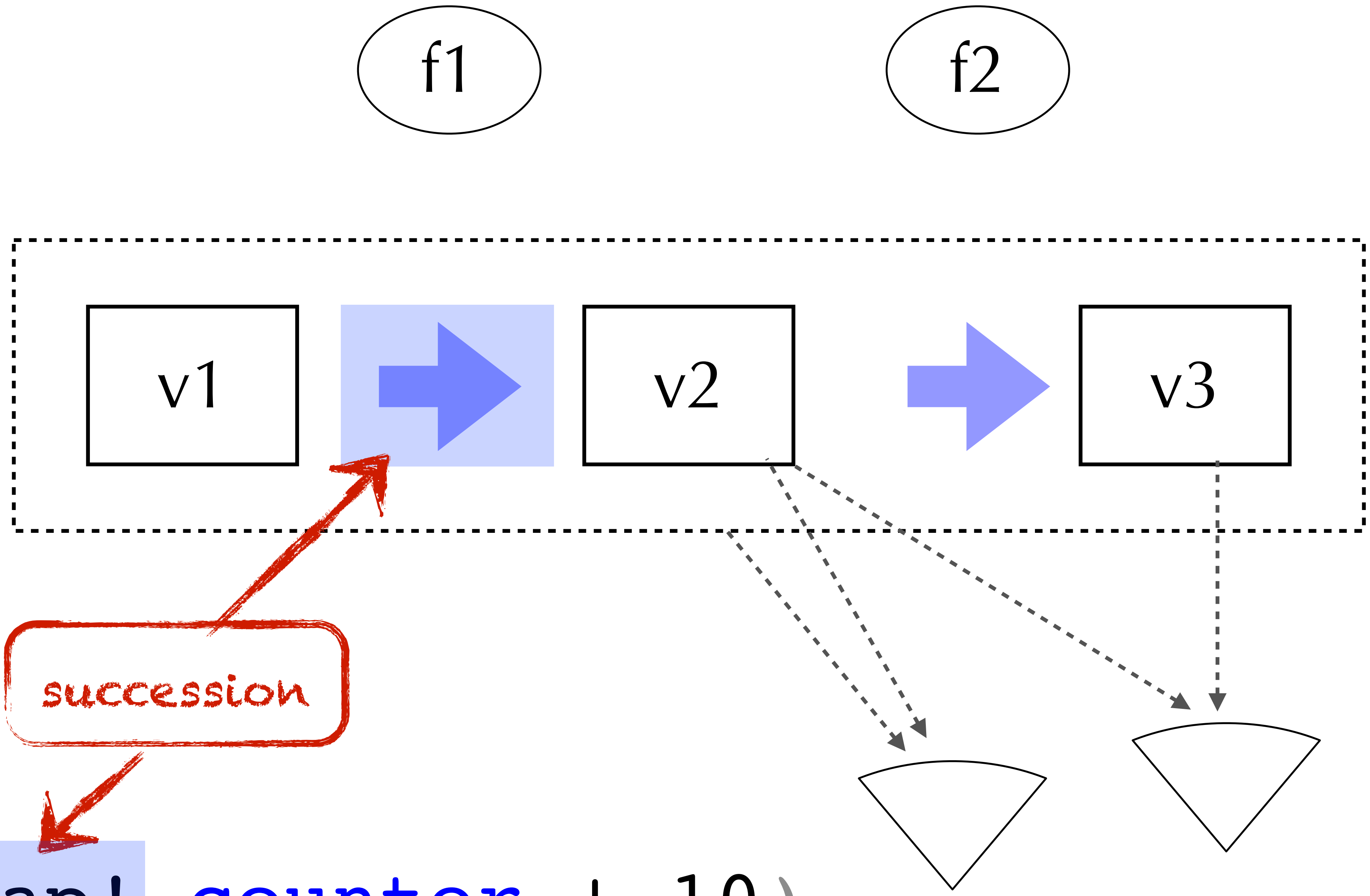
```
(atom 0)
```

atoms



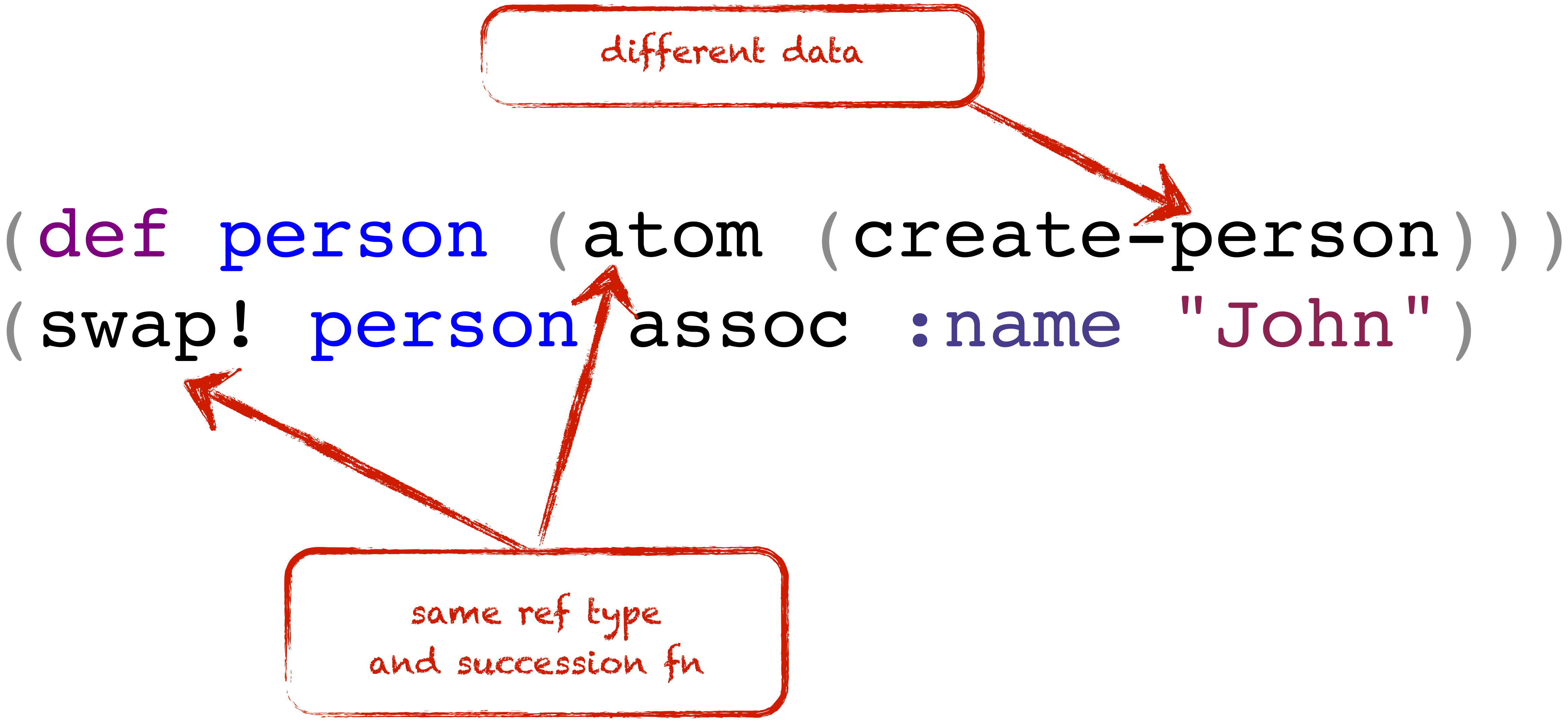
```
(swap! counter + 10)
```

atoms



```
(swap! counter + 10)
```

bigger structure



different data

```
(def person (atom (create-person)))
(swap! person assoc :name "John")
```

same ref type
and succession fn

varying semantics

different kind of ref

```
(def number-later (promise))
(deliver number-later 42)
```

different succession

entire database

```
(def conn (d/connect uri)
(transact conn data)
```

entire database

```
(def conn (d/connect uri))
(transact conn data)
```

succession fn

agent →

send	processor-derived pool
send-off	IO-derived pool
send-via	user-specified pool

atom ⇄

compare-and-set!	conditional
reset!	boring
swap!	functional transformation

connection

transact ⇄	ACID
transact-async →	ACID

ref ⇄

alter	functional transformation
commute	commutative

var ⇄

alter-var-root	application config

var binding ⇄

binding, set!	dynamic, binding-local

brought to you by

cognitect