clojure.spec

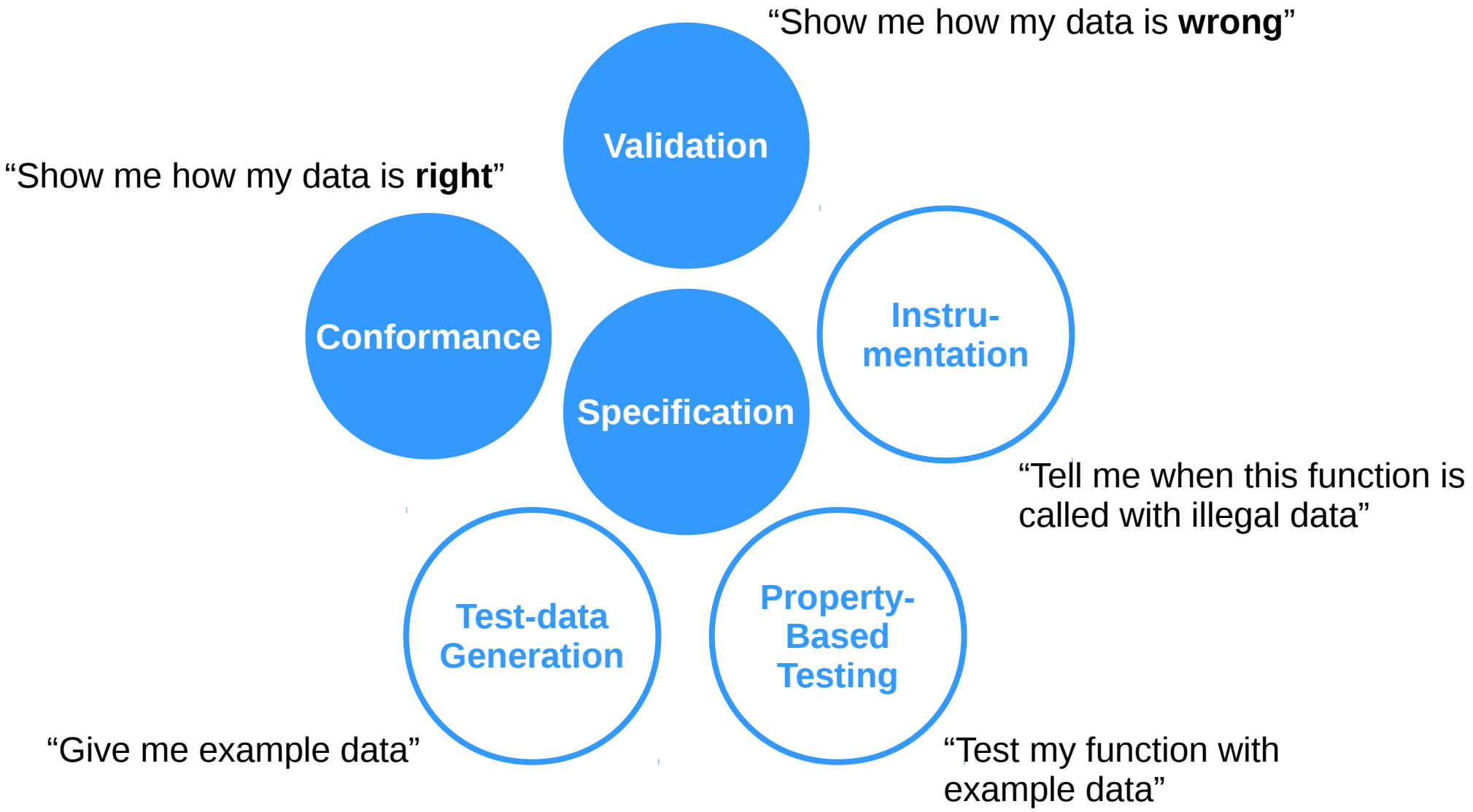
Agenda

- Features of spec
- Examples / Code
- Composing spec with multimethods

spec

=

Specifications for plain data


“Show me how my data is **wrong**”
Validation
“Show me how my data is **right**”
Conformance
Specification
Instru-
mentation
“Tell me when this function is called with illegal data”
Test-data
Generation
Property-
Based
Testing
“Give me example data”
“Test my function with example data”

Spec

- New in Clojure 1.9
- Currently 1.9-alpha15

Validation

```
(ns example.core
    (:require [clojure.spec :as s]))

(s/def ::my-int-spec integer?)

(s/valid? ::my-int-spec 42)

=>  true

(s/valid? ::my-int-spec 42.1)

=>  false

(s/valid? ::my-int-spec “42”)

=>  false
```

Validation

```
(s/def ::my-coll-spec (s/coll-of string?))

(s/valid? ::my-coll-spec (list “A” “b” “ccc”))

=>  true

(s/valid? ::my-coll-spec [“x” “y” 42])

=>  false

(s/explain ::my-coll-spec [“x” “y” 42])

=>  In: [2] val: 42 fails spec: :example.core/my-coll-spec predicate: string?
```

Validation

```
(s/def ::my-map-spec
       (s/map-of keyword? ::my-coll-spec))

(s/valid? ::my-map-spec {::x ["a" "b" "c"]
                         ::y [ ]
                         ::z ["foo" "bar"]})

=>  true
```

resembles Map<Keyword, List<String>> in Java

Validation

```
(s/def ::my-map-spec
    (s/map-of keyword?
                  (s/coll-of string? :s/min-count 1)))

(s/valid? ::my-map-spec {::x ["a" "b" "c"]
                                ::y [ ]})

=>  false

(s/valid? ::my-map-spec {::x ["a" "b" "c"]
                                ::y ["foo"]})

=>  true
```

Take-Aways

- Specs can be
  - Composed
  - Recursive
- More powerful than classes
  - Able to check runtime facts
  - Use arbitrary predicates
    - → Provide your own predicates

Conformance

```
(s/def ::my-choice-spec
        (s/or ::my-map (s/map-of keyword?
                                 String?)
              ::my-coll (s/coll-of string?)))

(s/valid? ::my-choice-spec {:a “A” :b “B”})

=> true

(s/valid? ::my-choice-spec [“A” “B”])

=> true

(s/explain ::my-choice-spec 42)

=> val: 42 fails spec: :examples/my-choice-spec at: [::my-map]
predicate: map?
val: 42 fails spec: :examples/my-choice-spec at: [::my-coll]
predicate: coll?
```

Conformance

```
(s/def ::my-choice-spec
       (s/or ::my-map (s/map-of keyword
                                string?)
             ::my-coll (s/coll-of string?)))

(s/conform ::my-choice-spec {:a “A” :b “B”})

=> [::my-map {:a “A” :b “B”}]

(s/conform ::my-choice-spec [“A” “B”])

=> [::my-coll [“A” “B”]]

(s/conform ::my-choice-spec 42)

=> :clojure.spec/invalid
```

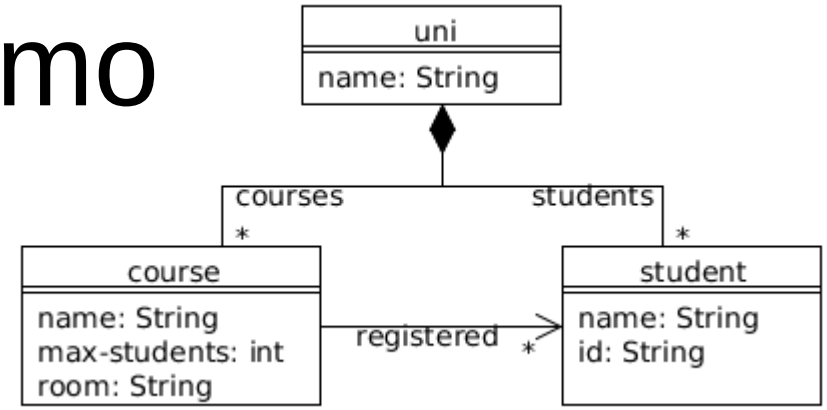
Take-Aways

- All choices require labeling (e.g. s/or)
- Conform
  - Returns [<label> <conformed value>]
  - Nests
- Conforming is
  - Extracting semantics
  - Parsing (context free grammars)
- Set logic with specs
  - s/or (union)
  - s/and (intersection)
  - Impossible with Java's classes

DSUI

- Toy project
- Goals
  - Learn how to use spec for conformance
  - Create (read-only) UI for Clojure data
- Intended to be used with the REPL
- https://github.com/Azel4231/dsui
- Afterwards found: clojure.inspector/inspect-tree

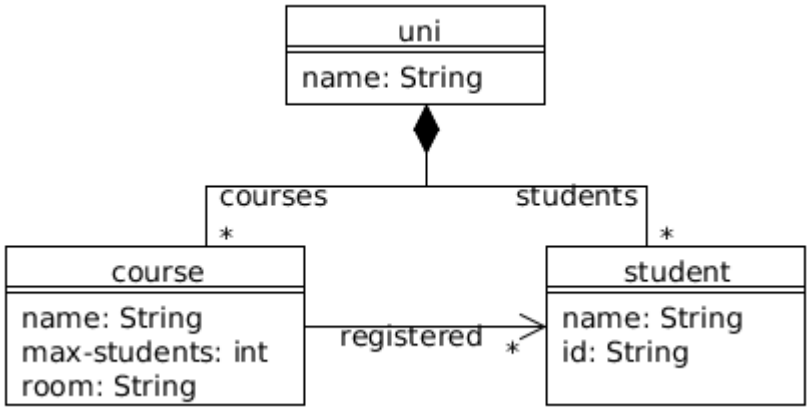
DSUI - Demo




```
{:name "Foo-University of Bar"
 :students [{:name "John Doe" :student-id "12345"}
            {:name "Jane Doe" :student-id "11111"}
            {:name "Dr. Who" :student-id "?"}]
 :courses [{:name "Linear Algebra"
            :max-students 15
            :room "Gauss"
            :registered ["11111" "?"]}
           {:name "Introduction to Algorithms"
            :max-students 25
            :room "Dijkstra"
            :registered ["12345" "?"]}]}
```

DSUI – Specs

```
(s/def ::dsui-spec (s/or ::table ::tbl-spec
                         ::list ::list-spec
                         ::named-tabsheet ::named-ts-spec
                         ::indexed-tabsheet ::indexed-ts-spec
                         ::form ::form-spec))

(s/def ::tbl-spec (s/and (s/coll-of map?) (s/every scalar-map?)
                         homogeneous?))

(s/def ::list-spec (s/and (complement map-entry?)
                          (s/coll-of scalar?))

(s/def ::named-ts-spec (s/map-of any? ::dsui-spec))

(s/def ::indexed-ts-spec (s/coll-of ::dsui-spec))

(s/def ::form-spec (s/map-of any? (s/or ::field scalar?
                                        ::nested-ui ::dsui-spec)))
```




```
{:name "Foo-University of Bar"
 :students [{:name "John Doe" :student-id "12345"}
            {:name "Jane Doe" :student-id "11111"}
            {:name "Dr. Who" :student-id "?"}]
 :courses [{:name "Linear Algebra"
            :max-students 15
            :room "Gauss"
            :registered ["11111" "?"]}
           {:name "Introduction to Algorithms"
            :max-students 25
            :room "Dijkstra"
            :registered ["12345" "?"]}]}
```

DSUI – Conformed Data

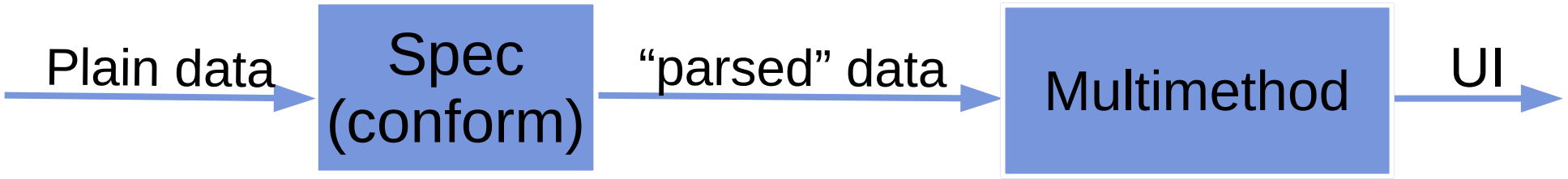
```
[:dsui.core/form
   {:name [:dsui.core/field "Foo-University of Bar"]
    :students [:dsui.core/nested-ui
                    [:dsui.core/table
                       [{:name "John Doe", :student-id "12345"}
                        {:name "Jane Doe", :student-id "11111"}
                        {:name "Dr. Who", :student-id "?"}]]]
    :courses  [:dsui.core/nested-ui
                  [:dsui.core/indexed-tabsheet …….
```

Developing DSUI

- Modeling the data
  - Take usual approach
  - Very much like modeling classes
- Get parser for free
- Extremely easy to test (data in → data out)

Developing DSUI

- I have parsed data, now what?
- Generate UI based on label
  - Need for polymporphism
  - Multimethods to the rescue

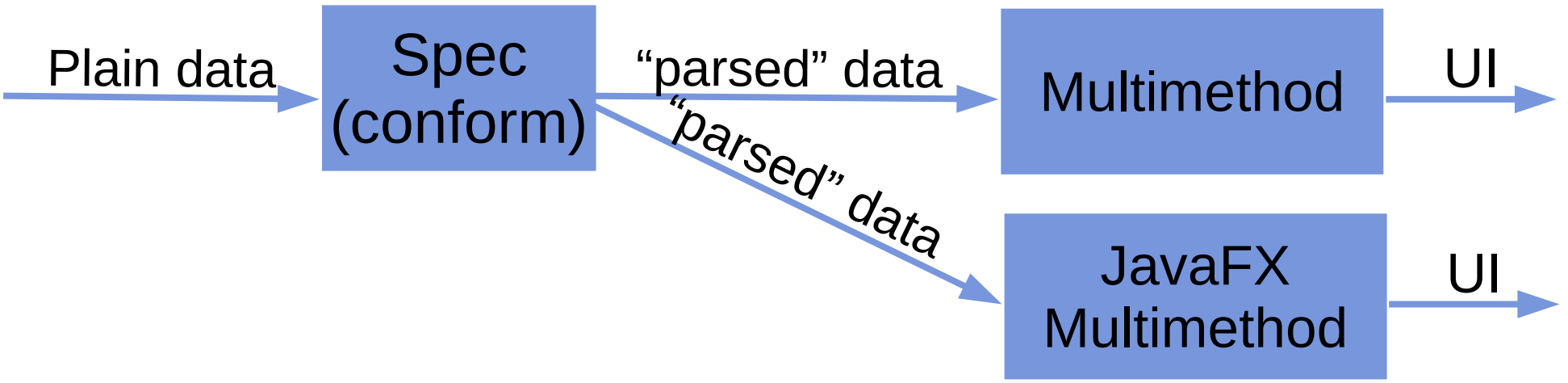



```
(defn dsui-panel [ds]
      (create (s/conform ::dsui-spec ds)))
```

- Spec does most of the decision-making

Developing DSUI

- Working with plain data is surprisingly easy
- Composing spec+multimethods is like LEGOs
- Simplicity
  - Enables reuse (modular by default)
  - Enables incremental development



Take-Aways

- Write specs, get parsers for free
- Multimethods
  - Work nicely with specs
  - Pattern matching on stereoids

Spec drawbacks (vs. Classes)

- Maps+Specs
  - Require more memory
  - Slower
- Checking only at runtime (mostly)
- Specs can conflict

High-Level Take-Aways

- Spec made me rethink the relationship between
  - Data modeling
  - Types
  - Grammars
- Spec is a way of extracting semantics
- Spec is Clojure’s missing piece

Links

- Cognicast about clojure.spec – Rich Hickey
- Introduction to clojure.spec – Arne Brasseur
- Agility & Robustness: clojure.spec – Stuart Halloway
- Spec Guide