CLOJURE MEETUP - 17/05/2017

CLOJURE.SPEC THOUGHTS

GOALS FOR TODAY

- Intro to Clojure Spec
- On a real world example
- Return on experience

RETURN ON EXPERIENCE

- Contracts vs Types
- Entities vs Associations
- The Binary Tree challenge

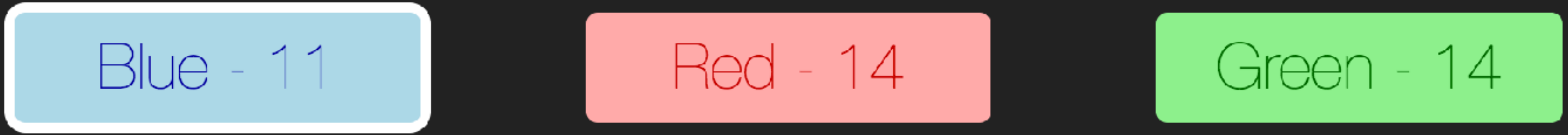
CLOJURE SPEC

INTRODUCTION & MOTIVATION

SPECIFY SHAPE OF DATA

GAME SCORE

- Three players: blue, red, green
- Each one has an positive integer score



GAME SCORE

```
{:player/blue   24
 :player/red    17
 :player/green  19}
```

SPEC FOR PLAYERS

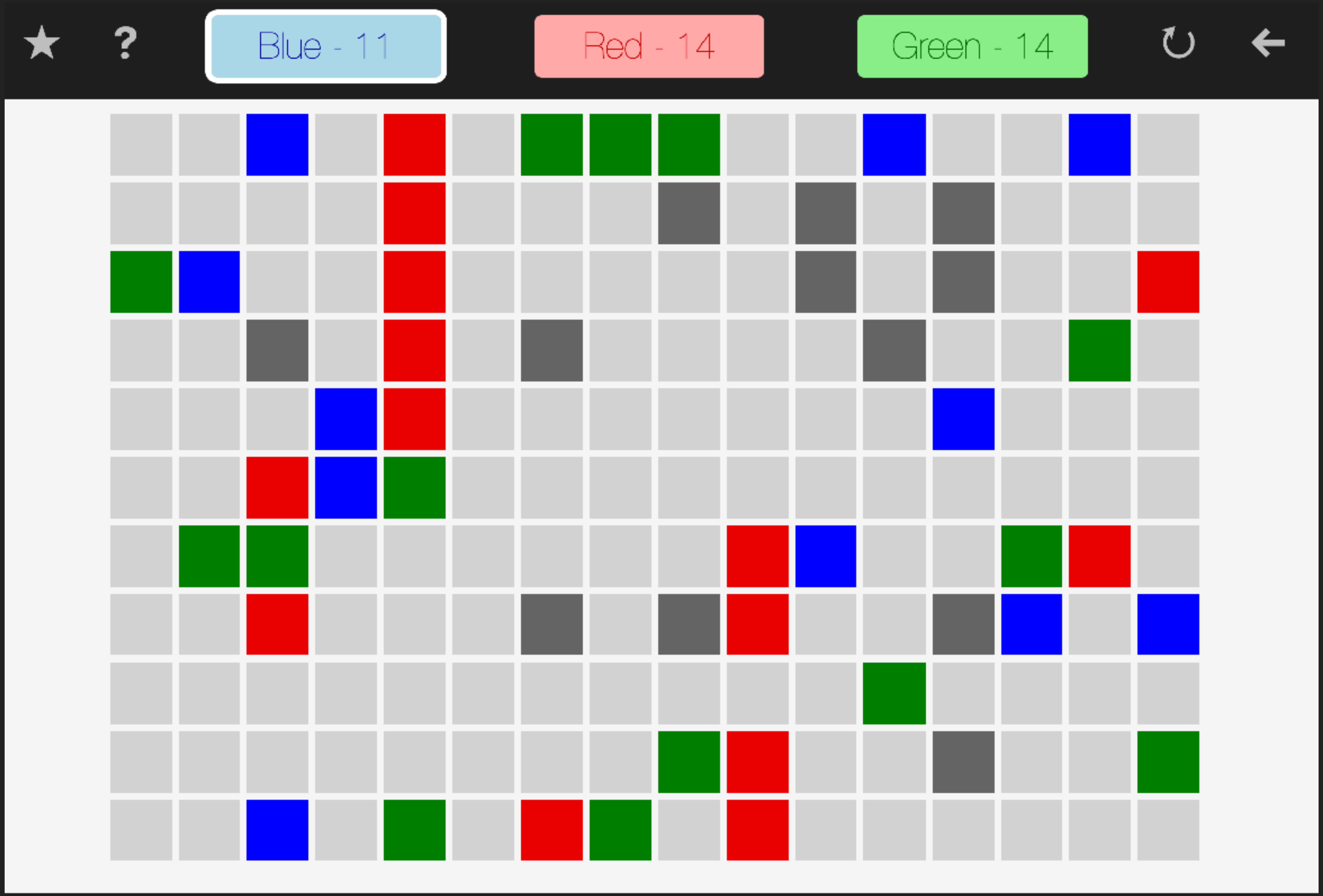
```
(s/def :player/player
  #{:player/blue
    :player/red
    :player/green})
```

SPEC FOR SCORES

```
(s/def :game/scores
   (s/map-of
      :player/player
      pos-int?))
```

DYNAMIC: RUN-TIME

DYNAMIC



https://github.com/QuentinDuval/triboard

DYNAMIC

```
(s/def :game/board
  (s/every
    (s/every
      :player/player
      :count height)
      :count width))
```

USEFUL: OFFER NICE GOODIES

USEFUL: OFFER NICE GOODIES

```
(s/valid? :game/turn turn)

(s/conform :game/turn turn)

(gen/sample (s/gen :game/turn) 1)
```

USEFUL: OFFER NICE GOODIES

- Validation, Instrumentation
- Parsing, Destructuring
- Generate samples / tests

FLEXIBLE: INFORM, NOT CONSTRAIN

EXAMPLE: A GAME TURN

```
{:game/board         board
 :game/scores        scores
 :game/transitions transitions}
```

NO NEED TO SPEC EVERYTHING

```
(s/def :game/turn
  (s/keys :req
    [:game/board
     :game/scores]))
```

NO NEED TO SPEC EVERYTHING

```
(s/fdef new-first-turn
  :args   (s/cat ...)
  :ret    :game/turn)
```

NO NEED TO SPEC EVERYTHING

```
{:game/board      board
 :game/scores     scores}
```

FLEXIBLE: PRESENCE VS CONFORMANCE

PRESENCE VS CONFORMANCE

```
(s/def :game/transitions
  (s/map-of
    :game/coordinate
    :game/transition))
```

PRESENCE VS CONFORMANCE

```
{:game/board        board
 :game/scores       scores
 :game/transitions []} ;;BOOM!
```

PRESENCE VS CONFORMANCE

```
{:game/board             board
 :game/scores            scores
 :game/transitions {}} ;;PASS
```

EXPRESS COMMITMENT WITH CLIENT

EXPRESSING COMMITMENT

- Define a contract with the client
  - Preconditions (never ask for more)
  - Postconditions (never provide less)

EXPRESSING COMMITMENT

- Avoid strong coupling of types
  - Exact same fields
  - Ordering of fields

- Type-Safety vs Easy Evolution

RETURN ON EXPERIENCE

PROBLEMS WITH SYMMETRY

BACK TO GAME/TURN SPEC

```
(s/def :game/turn
  (s/keys :req
    [:game/board
     :game/scores]))
```

FINE FOR OUTPUT

```
(s/fdef new-first-turn
  :args  (s/cat ...)
  :ret   :game/turn)
```

WHAT ABOUT SYMMETRY?

```clojure
(s/fdef next-turn
  :args
  (s/cat :turn  :game/turn
         :move  :game/transition)
  :ret   :game/turn)
```

WHAT ABOUT SYMMETRY?

```clojure
(s/fdef next-turn
  :args
  (s/cat :turn :game/turn
         :move :game/transition)
  :ret   :game/turn)
```

NEXT-TURN SYMMETRY

- Need transitions as input
    - Input spec is incomplete
    - Post conditions might not be fulfilled

- No wish to commit keyword presence

SOLUTION 1: COMMIT EVERYTHING

```
(s/def :game/turn
  (s/keys :req
    [:game/board
     :game/scores
     :game/transitions]))
```

SOLUTION 1: COMMIT EVERYTHING

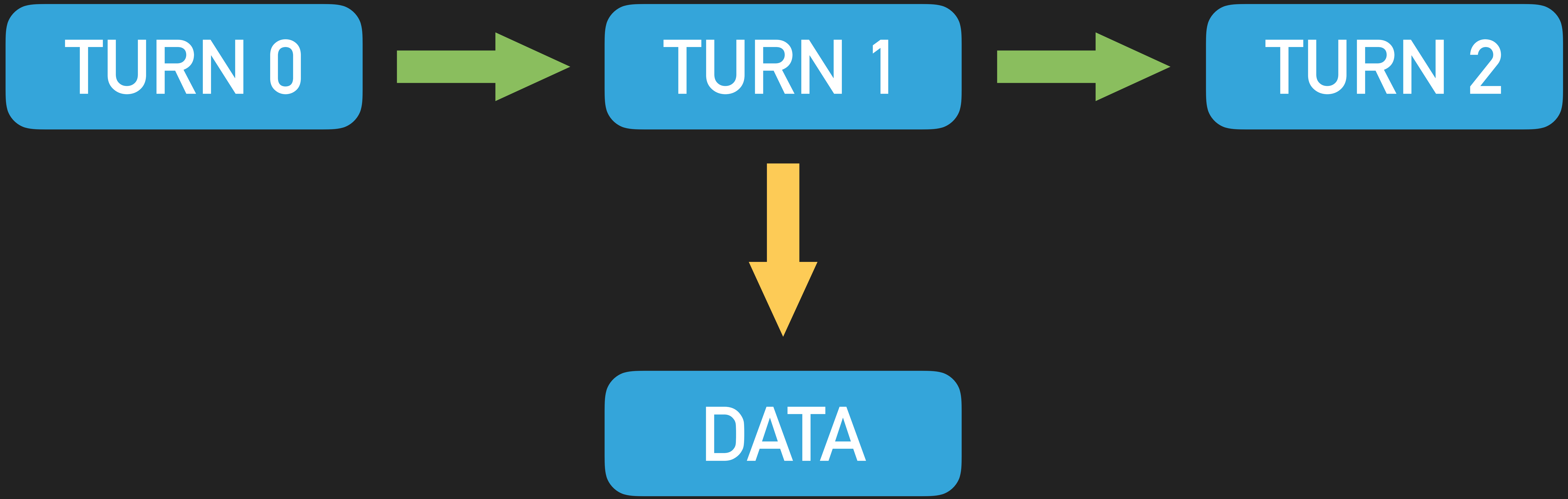
- Still hidden dependencies
  - Between board, transition, scores…
  - Test samples might not pass
- Precision leads to duplication

SOLUTION 2: MODIFY DEFINITION OF TURN

- A valid turn is:
  - An initial new-first-turn
  - Followed by N next-turn

- Generate turn by successive next-turn

SOLUTION 2: MODIFY DEFINITION OF TURN



Spec turn information

IMPLEMENTATION VS INFORMATION

- Spec is meant for information
  - Data orientation of Clojure
  - Code is data, macros
- Spec is not meant for implementation

RETURN ON EXPERIENCE

ENTITIES VS ASSOCIATIONS

A DIFFERENT SPEC FOR SCORES

```
{:player/blue   24
 :player/red    17
 :player/green 19}
```

A DIFFERENT SPEC FOR SCORES

```
(s/def :player/scores
  (s/map-of
    :player/player
    pos-int?))
```

A DIFFERENT SPEC FOR SCORES

```
(s/def :player/red   pos-int?)
(s/def :player/green pos-int?)
(s/def :player/blue  pos-int?)

(s/def :game/scores
  (s/keys :req
    [:player/red
     :player/green
     :player/blue]))
```

ENTITY VS ASSOCIATIONS

- Assign specs to entities (intrinsic meaning)
- Use key sets for entities membership
- Use map-of for associations

THE CHALLENGE

MODELLING A BINARY TREE

THE BINARY TREE CHALLENGE

```
class BinaryTree<A>
{

    A value;
    BinaryTree<A> lhs;
    BinaryTree<A> rhs;

};
```

INTEGER BINARY TREE (REPRESENTATION)

```
[1
  {:lhs [5 {}]
   :rhs [2
              {:lhs [3 {}]
               :rhs [4 {}]
  }]
```

ENTITIES, NOT A GOOD FIT

```
(s/def :tree/lhs :tree/b-tree)
(s/def :tree/rhs :tree/b-tree)

(s/def :tree/b-tree
  (s/cat :value any?
         :children
         (s/keys :opt
          [:tree/lhs :tree/rhs])))
```

ENTITIES, NOT A GOOD FIT

```
(s/def :tree/lhs :tree/b-tree)
(s/def :tree/rhs :tree/b-tree)

(s/def :tree/b-tree
  (s/cat :value any?
         :children
         (s/keys :opt
          [:tree/lhs :tree/rhs]))
```

ASSOCIATIONS

```
(s/def :int-tree
  (s/cat
    :value int?
    :children
    (s/map-of #{:lhs :rhs}
                 :int-tree)))
```

ASSOCIATIONS

```
(s/def :int-tree
  (s/cat
    :value int?
    :children
    (s/map-of #{:lhs :rhs}
              :int-tree)))
```

USING MACROS FOR GENERICS

```
(def-btree-of
  :int-tree int?)

(def-btree-of
  :string-tree string?)
```

CLOJURE SPEC

CONCLUSION & LINKS

CONCLUSION

- Trade-off between safety and coupling
- Oriented toward information
- Requires different thinking than types

RESOURCES

- https://github.com/QuentinDuval/ClojureMeetup-2017-05-17
- https://clojure.org/about/spec
- https://clojure.org/guides/spec
- https://www.youtube.com/watch?v=oyLBGkS5ICk

BONUS NOTES

- Easier to evolve VS easier to refactor
  - Forbidding all else forbids change
- You stay with dynamic typing for flexibility
  - It is not about being lazy or unsafe