You are in a maze of deeply nested maps, all alike

Eric Normand
IN/Clojure 2019

PurelyFunctional.tv

Symptoms

Symptom:

I can’t remember what keys belong in this map

Symptom:

I don’t even know what kind of entity I have

Symptom:

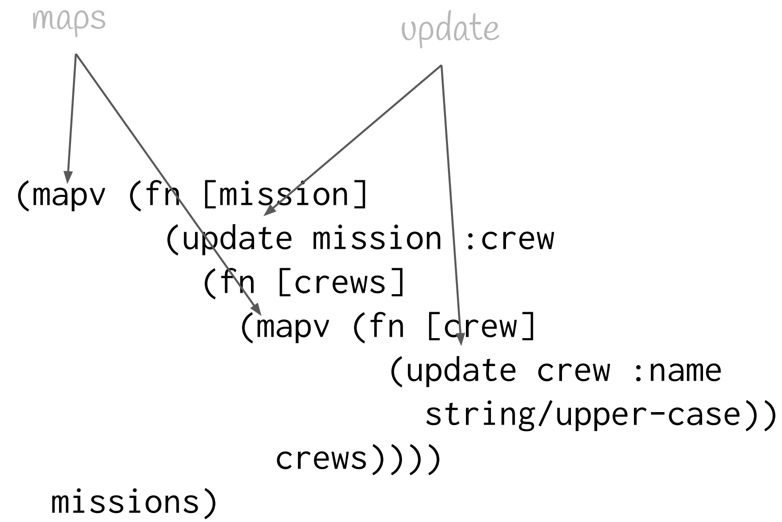
Working with deeply nested data is awkward

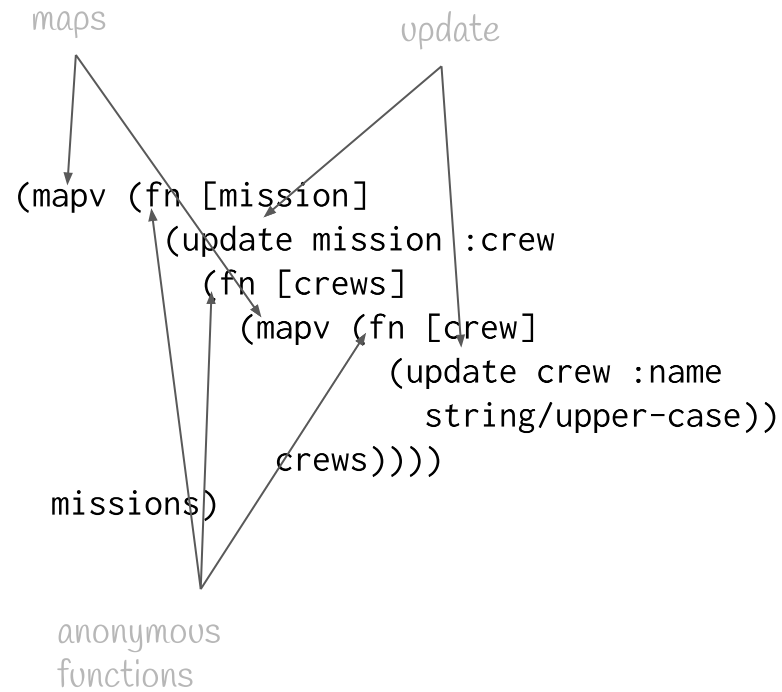
```
{:date #inst "1984-04-03T13:08:42.020"
 :name "Soyuz T-11"
 :spacecraft "Soyuz"
 :destination "Salyut 7"
 :crew [{:name "Yuri Malyshev"
          :position "Commander"}
         {:name "Gennadi Strekalov"
          :position "Flight Engineer"}
         {:name "Rakesh Sharma"
          :position "Research Cosmonaut"}]}
```

```
(mapv (fn [mission]
        (update mission :crew
          (fn [crews]
            (mapv (fn [crew]
                    (update crew :name
                      string/upper-case))
              crews))))
  missions)
```

maps

```
(mapv (fn [mission]
        (update mission :crew
          (fn [crews]
            (mapv (fn [crew]
                    (update crew :name
                      string/upper-case))
              crews))))
  missions)
```


maps
update
(mapv (fn [mission]
(update mission :crew
(fn [crews]
(mapv (fn [crew]
(update crew :name
string/upper-case))
crews))))
missions)


maps
update
(mapv (fn [mission]
(update mission :crew
(fn [crews]
(mapv (fn [crew]
(update crew :name
string/upper-case))
crews))))
missions)
anonymous
functions

```
(mapv (fn [mission]
        (update mission :crew
          (fn [crews]
            (mapv (fn [crew]
                    (update crew :name
                      string/upper-case))
              crews))))
  missions)
```

Symptom:

Long, convoluted functions

Technological Solutions

Spec

Specter

Symptoms	Spec	Specter
What keys?		
What entity?		
Deep nesting		
Long functions		

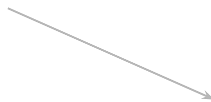
Symptoms	Spec	Specter
What keys?		
What entity?		
Deep nesting		
Long functions		

Underlying Problem:

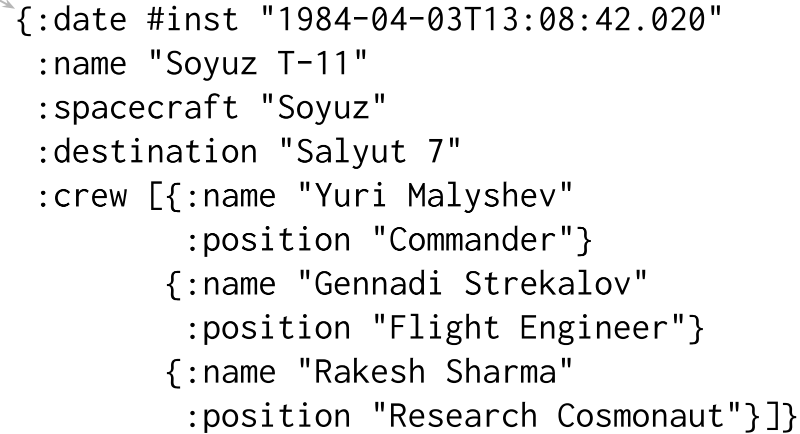
You're working at the wrong level of *meaning*

It’s just data

```
{:date #inst "1984-04-03T13:08:42.020"
 :name "Soyuz T-11"
 :spacecraft "Soyuz"
 :destination "Salyut 7"
 :crew [{:name "Yuri Malyshev"
         :position "Commander"}
        {:name "Gennadi Strekalov"
         :position "Flight Engineer"}
        {:name "Rakesh Sharma"
         :position "Research Cosmonaut"}]}
```




```
{:date #inst "1984-04-03T13:08:42.020"
 :name "Soyuz T-11"
 :spacecraft "Soyuz"
 :destination "Salyut 7"
 :crew [{:name "Yuri Malyshev"
         :position "Commander"}
        {:name "Gennadi Strekalov"
         :position "Flight Engineer"}
        {:name "Rakesh Sharma"
         :position "Research Cosmonaut"}]}
```


character
{:date #inst "1984-04-03T13:08:42.020"
:name "Soyuz T-11"
:spacecraft "Soyuz"
:destination "Salyut 7"
:crew [{:name "Yuri Malyshev"
:position "Commander"}
{:name "Gennadi Strekalov"
:position "Flight Engineer"}
{:name "Rakesh Sharma"
:position "Research Cosmonaut"}]}

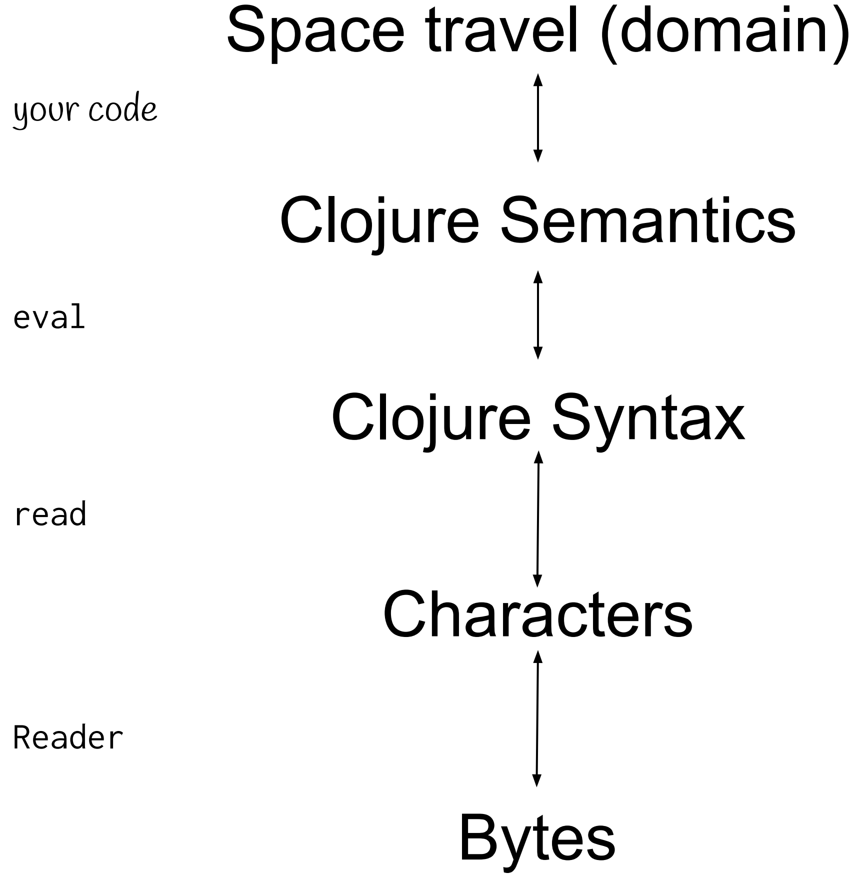
Space travel

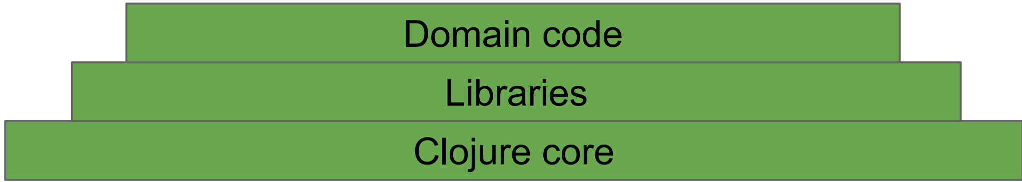
Clojure Semantics

Clojure Syntax

Characters

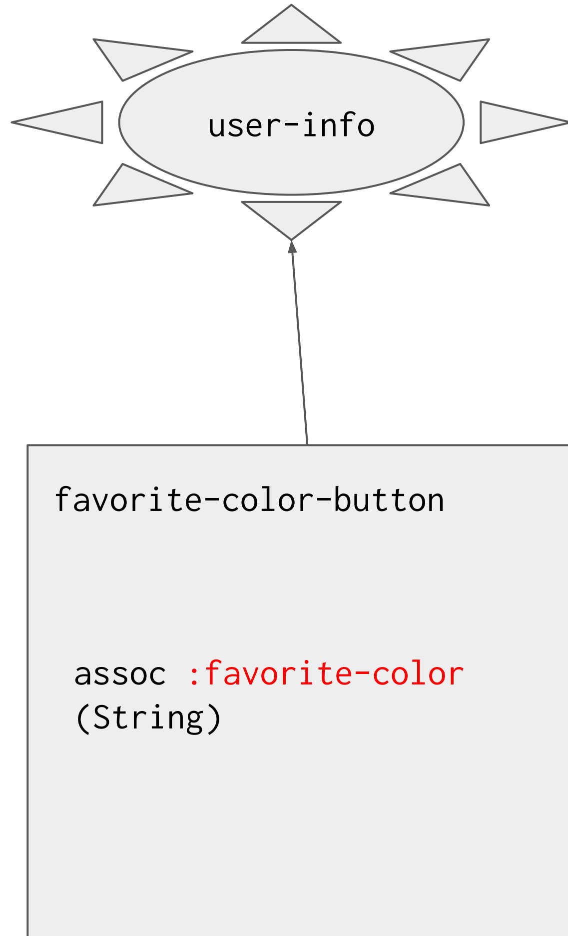
Bytes

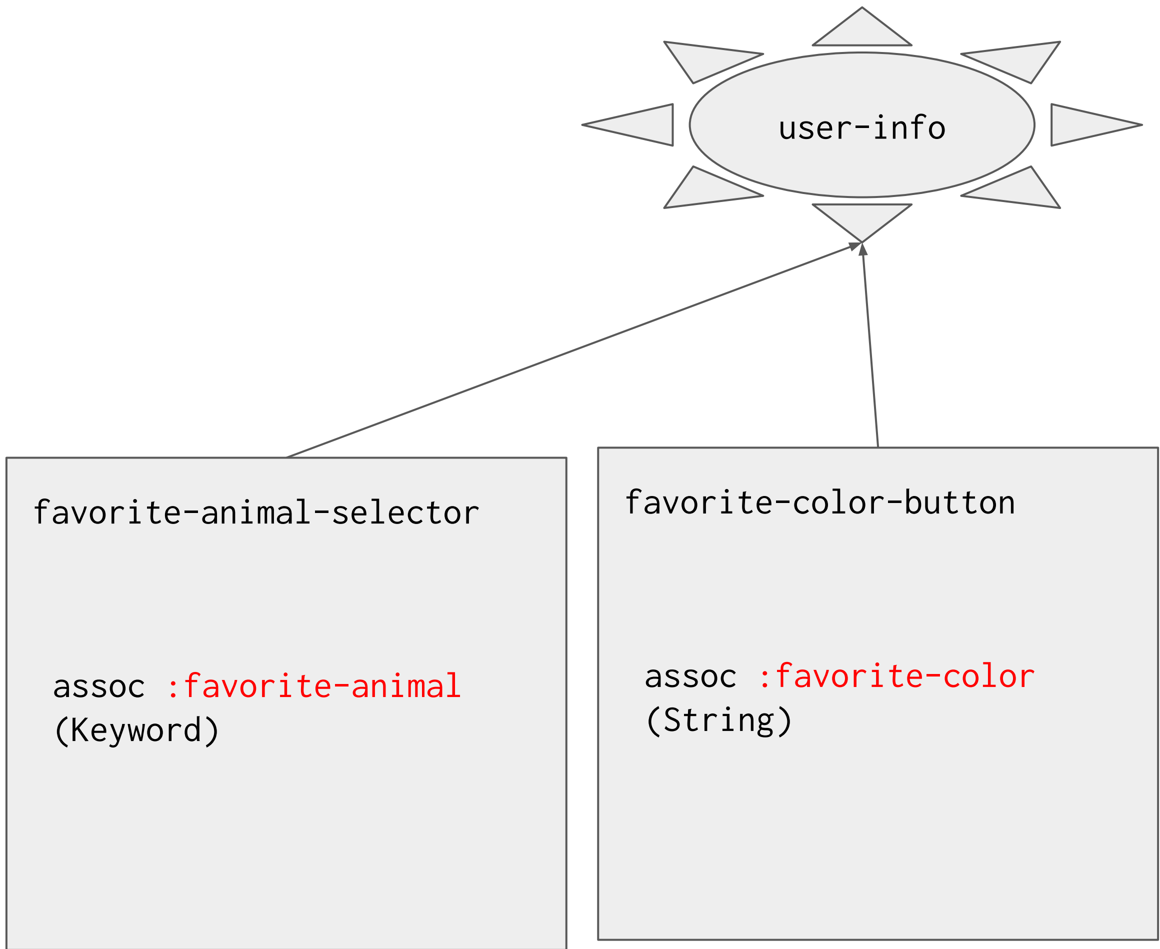

Space travel (domain)
your code
Clojure Semantics
eval
Clojure Syntax
read
Characters
Reader
Bytes

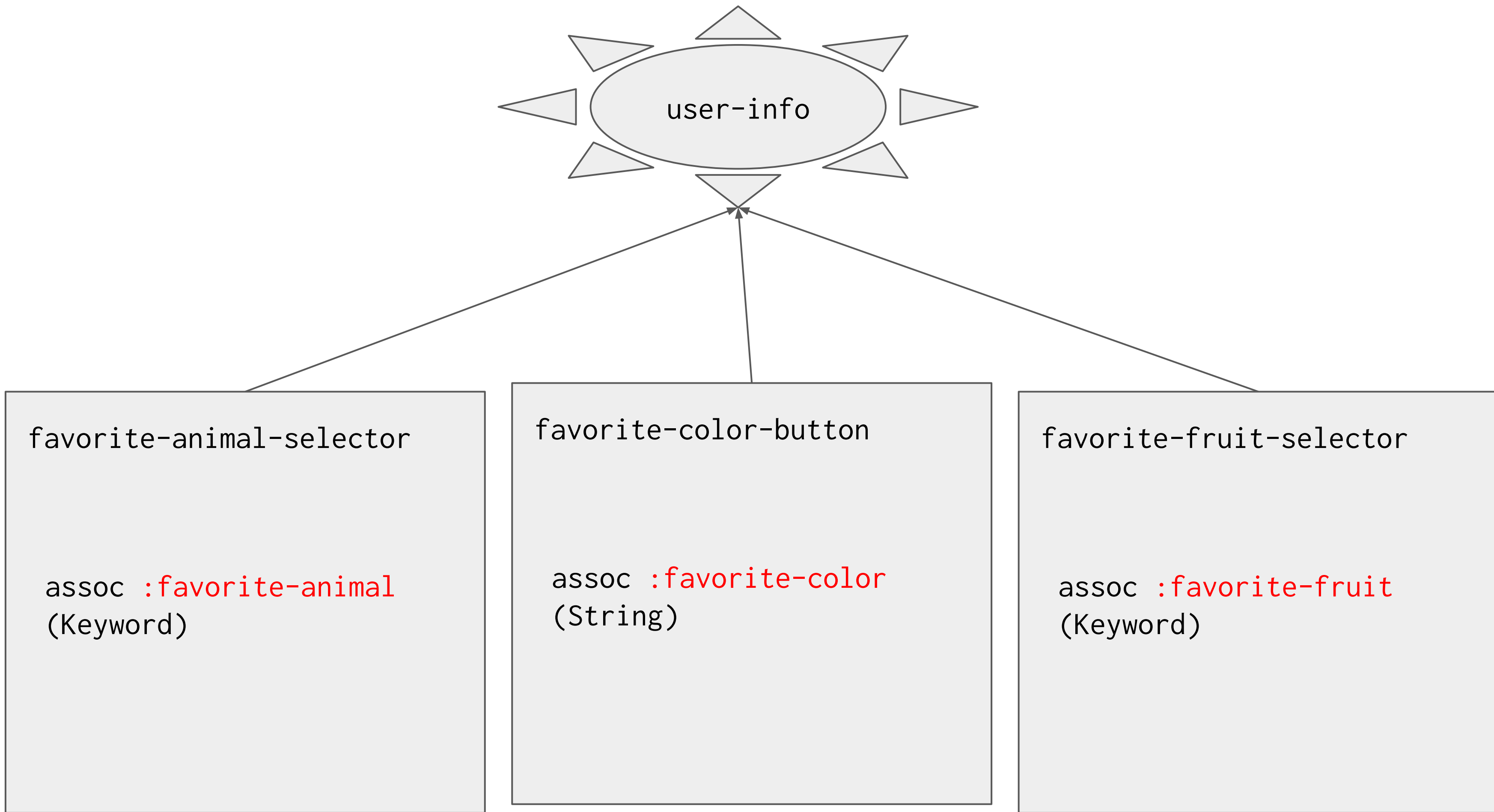

Domain code
Libraries
Clojure core

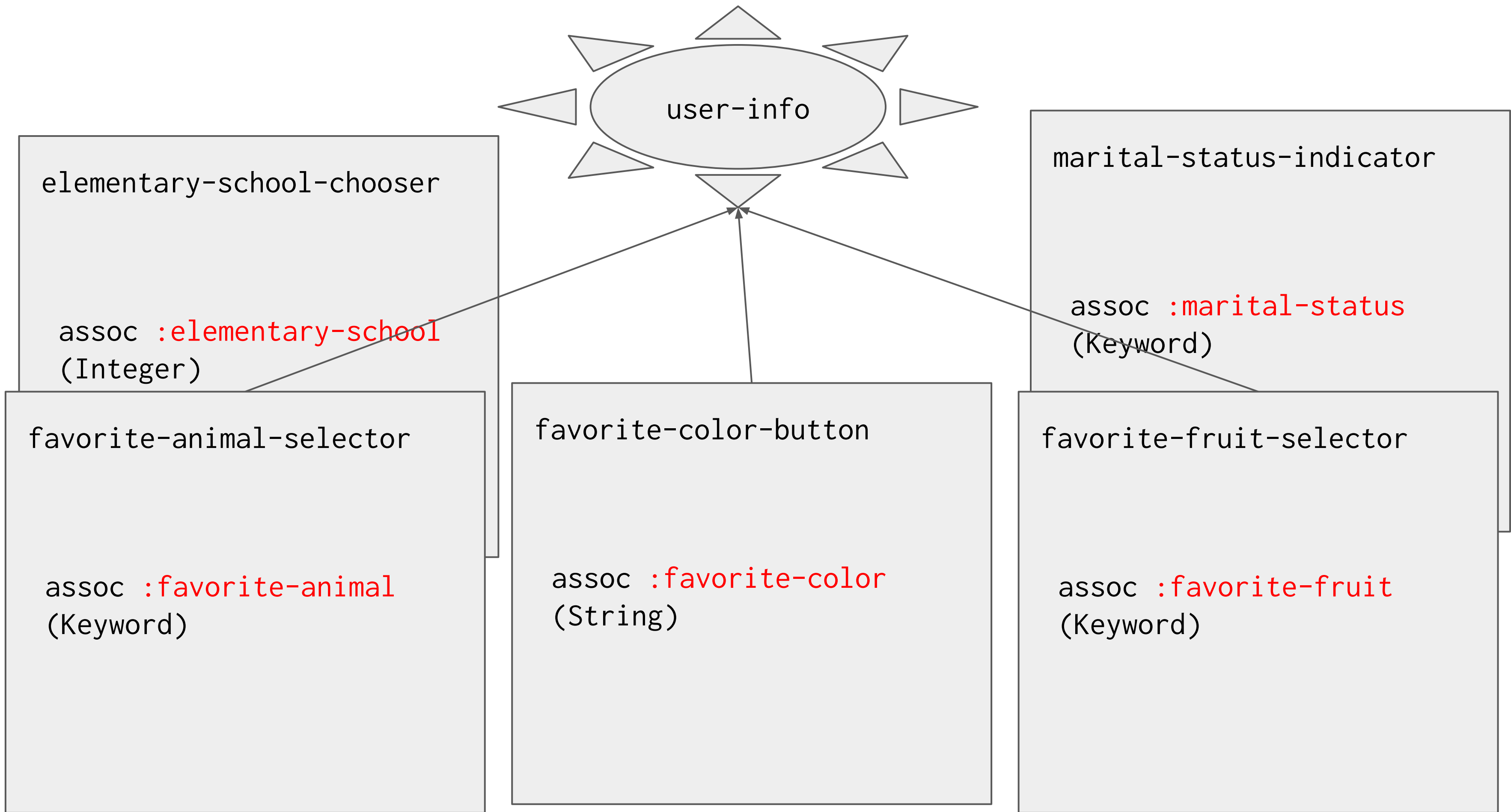
```clojure
(def user-info (reagent/atom {}))

(defn favorite-color-button [color] ;; reagent component
  [:button {:on-click (fn []
                        (swap! user-info assoc :favorite-color color))
            :style {:background-color color}}
    color])

(defn favorite-color-panel []
  [:div
    [:h2 "Favorite color"]
    [:h3 "Current: " (:favorite-color @user-info)]
    [:h3 "Change it by clicking below"]
    [favorite-color-button "green"]
    [favorite-color-button "blue"]
    [favorite-color-button "red"]])
```

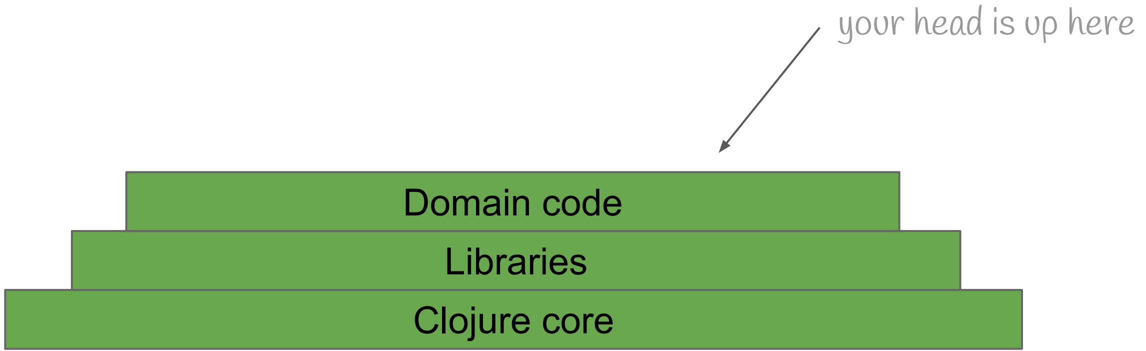
```clojure
(def user-info (reagent/atom {}))

(defn favorite-color-button [color] ;; reagent component
  [:button {:on-click (fn []
                        (swap! user-info assoc :favorite-color color))
            :style {:background-color color}}
    color])

(defn favorite-color-panel []
  [:div
    [:h2 "Favorite color"]
    [:h3 "Current: " (:favorite-color @user-info)]
    [:h3 "Change it by clicking below"]
    [favorite-color-button "green"]
    [favorite-color-button "blue"]
    [favorite-color-button "red"]])
```

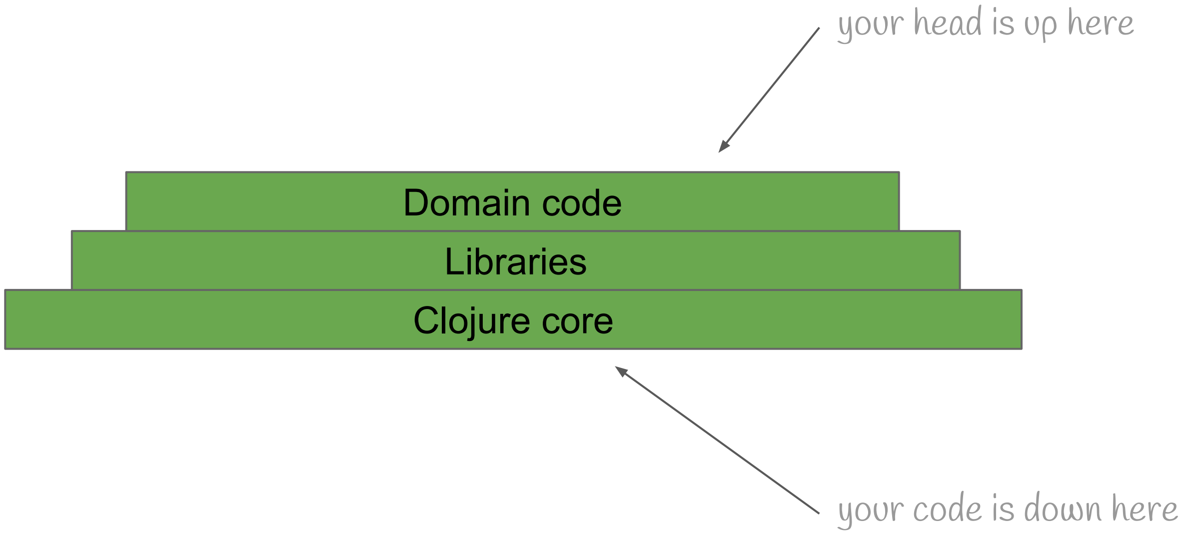

user-info
favorite-color-button
assoc :favorite-color
(String)


user-info
favorite-animal-selector
assoc :favorite-animal
(Keyword)
favorite-color-button
assoc :favorite-color
(String)


user-info
favorite-animal-selector
assoc :favorite-animal
(Keyword)
favorite-color-button
assoc :favorite-color
(String)
favorite-fruit-selector
assoc :favorite-fruit
(Keyword)


user-info
elementary-school-chooser
assoc :elementary-school
(Integer)
marital-status-indicator
assoc :marital-status
(Keyword)
favorite-animal-selector
assoc :favorite-animal
(Keyword)
favorite-color-button
assoc :favorite-color
(String)
favorite-fruit-selector
assoc :favorite-fruit
(Keyword)


your head is up here
Domain code
Libraries
Clojure core


your head is up here
Domain code
Libraries
Clojure core
your code is down here

```
(assoc user-info :favorite-color color)
```

domain

```
(defn set-favorite-color [user-info color]
  (assoc user-info :favorite-color color))
```

Clojure core and libraries

```clojure
(def user-info (reagent/atom {}))

(defn favorite-color-button [color] ;; reagent component
  [:button {:on-click (fn []
                        (swap! user-info assoc :favorite-color color))
            :style {:background-color color}}
    color])

(defn favorite-color-panel []
  [:div
    [:h2 "Favorite color"]
    [:h3 "Current: " (:favorite-color @user-info)]
    [:h3 "Change it by clicking below"]
    [favorite-color-button "green"]
    [favorite-color-button "blue"]
    [favorite-color-button "red"]])
```

```
(def user-info (reagent/atom {}))

(defn favorite-color-button [color] ;; reagent component
  [:button {:on-click (fn []
                        (swap! user-info set-favorite-color color))
            :style {:background-color color}}
    color])

(defn favorite-color-panel []
  [:div
    [:h2 "Favorite color"]
    [:h3 "Current: " (:favorite-color @user-info)]
    [:h3 "Change it by clicking below"]
    [favorite-color-button "green"]
    [favorite-color-button "blue"]
    [favorite-color-button "red"]])
```

```clojure
(ns my-app.user-info)

(defn set-favorite-color [user-info color]
  (assoc user-info :favorite-color color))
```

```clojure
(ns my-app.user-info)

(defn set-favorite-color [user-info color]
  (assoc user-info :favorite-color color))

(defn set-favorite-animal [user-info animal]
  (assoc user-info :favorite-animal animal))

(defn set-favorite-fruit [user-info fruit]
  (assoc user-info :favorite-fruit fruit))
```

Objection:

But Eric, isn't that a lot of code?

```clojure
(def user-info (reagent/atom {}))

(defn favorite-color-button [color] ;; reagent component
  [:button {:on-click (fn []
                        (swap! user-info assoc :favorite-color color))
            :style {:background-color color}}
    color])

(defn favorite-color-panel []
  [:div
    [:h2 "Favorite color"]
    [:h3 "Current: " (:favorite-color @user-info)]
    [:h3 "Change it by clicking below"]
    [favorite-color-button "green"]
    [favorite-color-button "blue"]
    [favorite-color-button "red"]])
```

```clojure
(def user-info (reagent/atom {}))

(defn set-favorite-color [user-info color]
  (assoc user-info :favorite-color color))

(defn favorite-color-button [color] ;; reagent component
  [:button {:on-click (fn []
                        (swap! user-info set-favorite-color color))
            :style {:background-color color}}
    color])

(defn favorite-color-panel []
  [:div
    [:h2 "Favorite color"]
    [:h3 "Current: " (:favorite-color @user-info)]
    [:h3 "Change it by clicking below"]
    [favorite-color-button "green"]
    [favorite-color-button "blue"]
    [favorite-color-button "red"]])
```

Con:

3x the code

Con:

3x the code

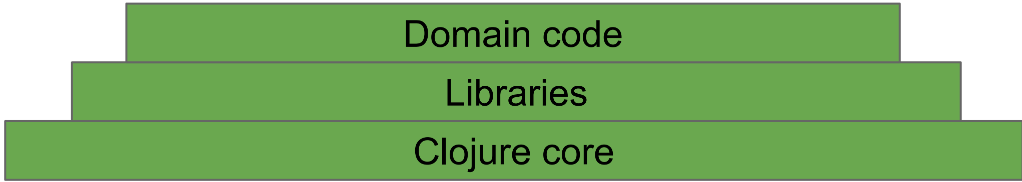
Pro:

Find things in ⅓ the time

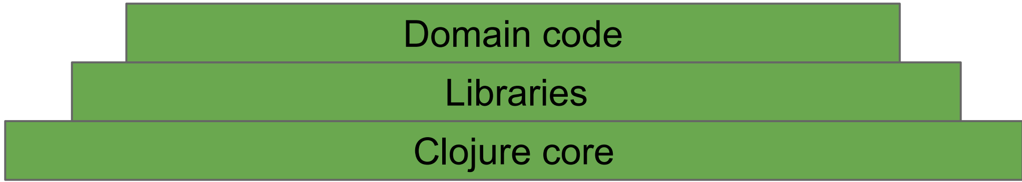
Probably duplicated operations

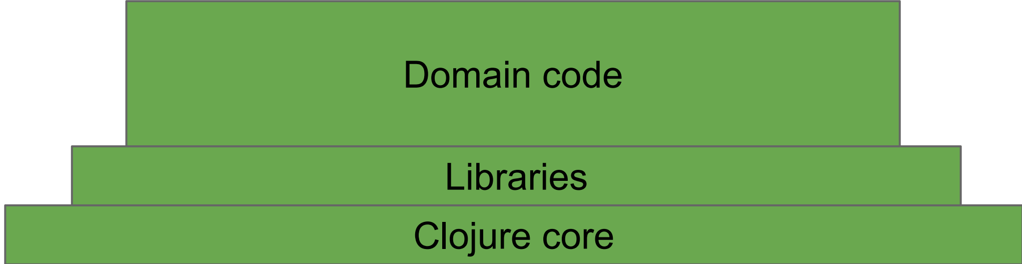
Parts can evolve separately

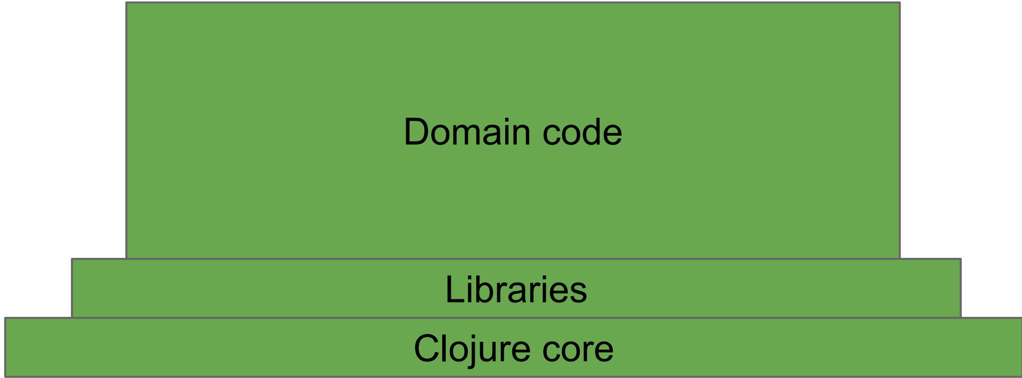
Less to keep in your head

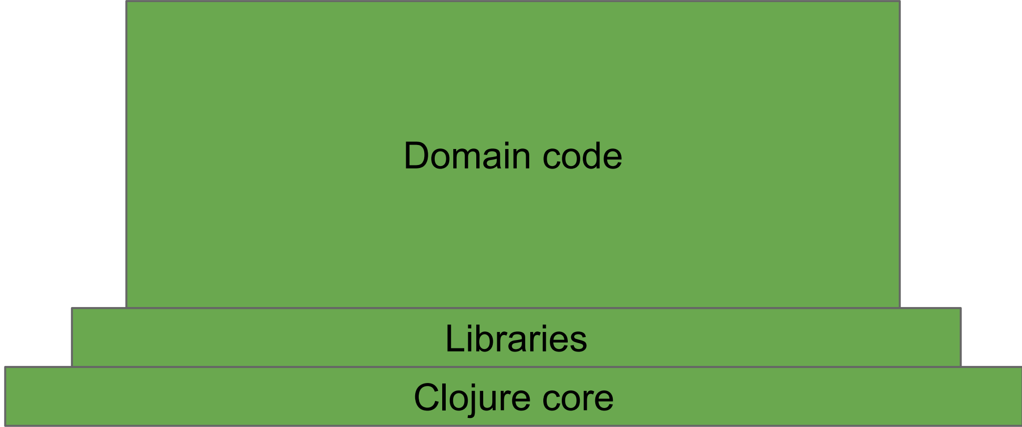

Domain code
Libraries
Clojure core

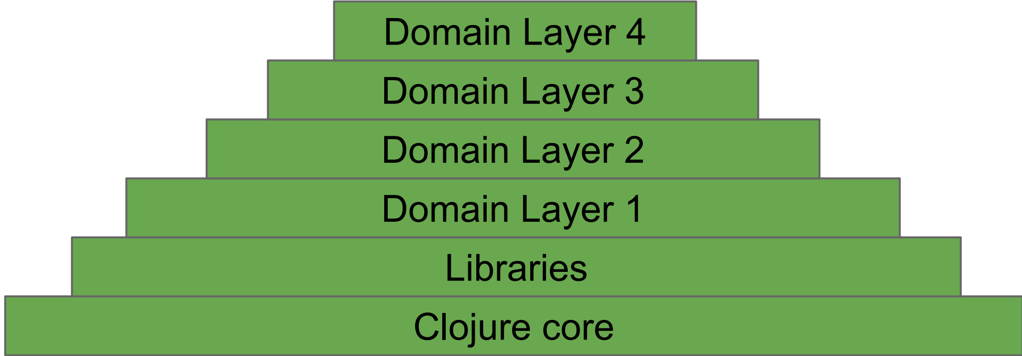
The biggest regret I hear frequently is doing data operations everywhere.


Domain code
Libraries
Clojure core


Domain code
Libraries
Clojure core


Domain code
Libraries
Clojure core


Domain code
Libraries
Clojure core


Domain Layer 4
Domain Layer 3
Domain Layer 2
Domain Layer 1
Libraries
Clojure core

```
{:date #inst "1984-04-03T13:08:42.020"
 :name "Soyuz T-11"
 :spacecraft "Soyuz"
 :destination "Salyut 7"
 :crew {:commander          {:name "Yuri Malyshev"
                             :favorite-color "blue"}
        :flight-engineer    {:name "Gennadi Strekalov"
                             :favorite-fruit :banana}
        :research-cosmonaut {:name "Rakesh Sharma"
                             :favorite-animal :sheep}}}
```

R. SHARMA
Р ШАРМА
СССР

```
(assoc-in mission [:crew :research-cosmonaut :training :centrifuge :status] :pass)
```

```
(assoc-in mission [:crew :research-cosmonaut :training :centrifuge :status] :pass)
```

```
(update-in mission [:crew :research-cosmonaut :training :centrifuge :status] :pass)
```

mission

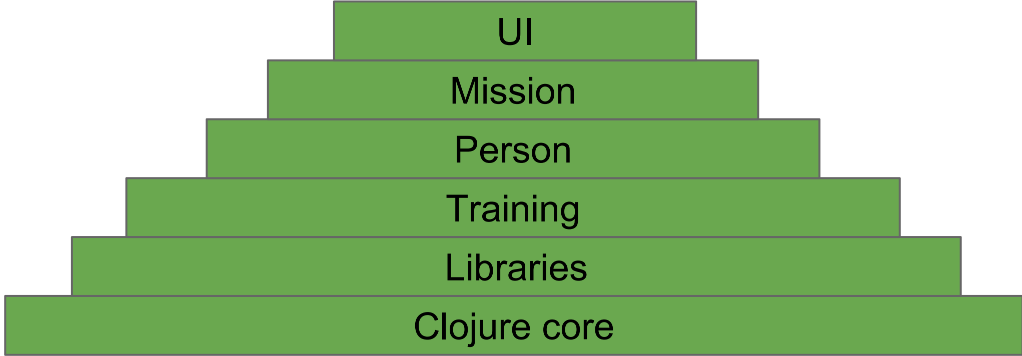
person

training

```clojure
(ns my-app.training)

(defn set-status [record training status]
  (assoc record training status))
```

```clojure
(ns my-app.person
  (:require [my-app.training :as training])

(defn set-training-status [person training status]
  (update person :training training/set-status training status))
```

```clojure
(ns my-app.mission
  (:require [my-app.person :as person]))

(defn set-training-status [mission position training status]
  (update-in mission [:crew position] person/set-training-status training status))
```


UI
Mission
Person
Training
Libraries
Clojure core

Law of Demeter

a given object should assume as little as possible about the structure or properties of anything else (including its subcomponents)

Objection:

But Eric, aren't you encapsulating?
Isn't the point of Clojure that it's all data?

Summary:

We are encapsulating and it's still data.

Map

Entity

Map

Entity

- enumerate keys
- serialize to json
- print
- compare with =
- hash code
- try out data operations at REPL

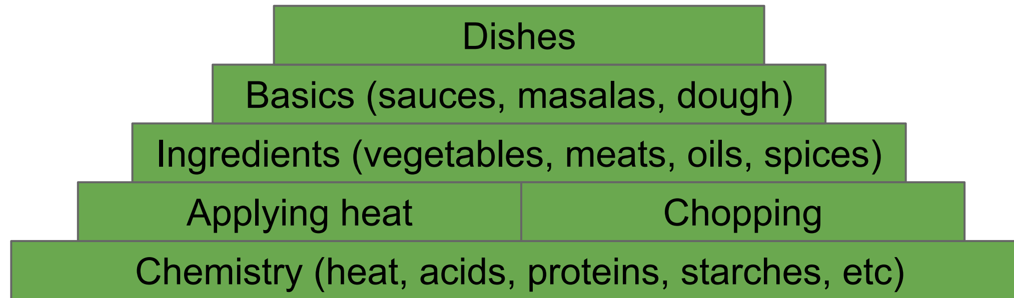
Map Entity

tension

- enumerate keys
- serialize to json
- print
- compare with =
- hash code
- try out data operations at REPL

- domain operations
- maintain invariants
- program at a high level
- easier to understand

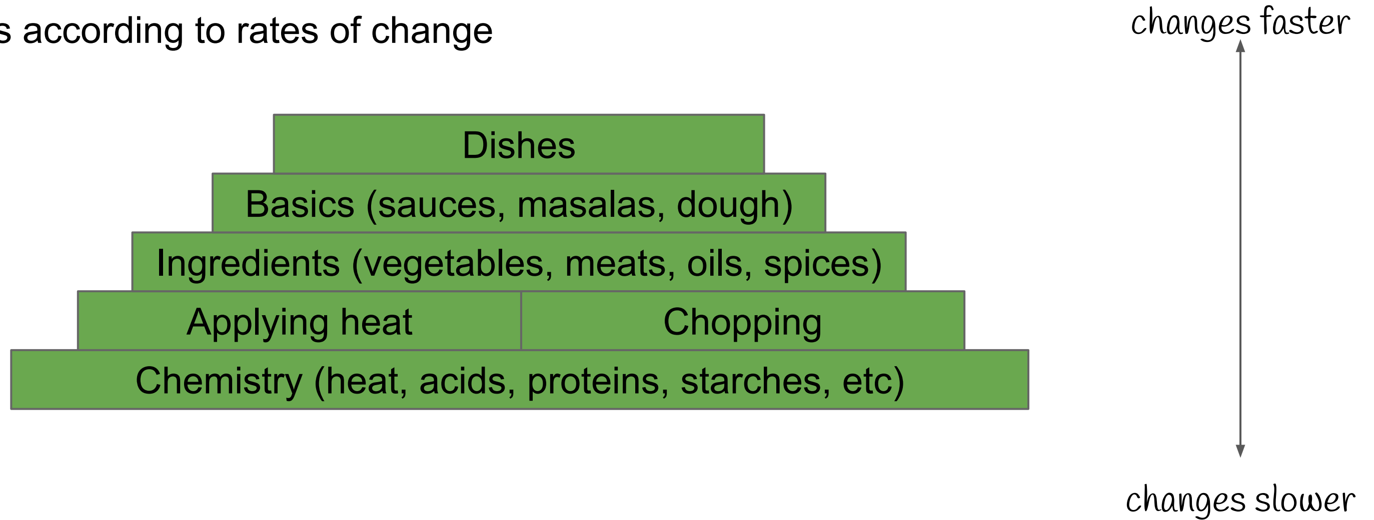
Stratified Design



Stratified Design

Principle:

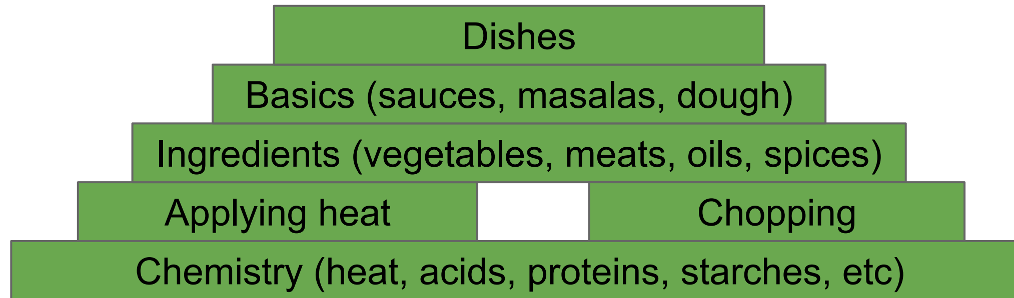
separate things according to rates of change



Stratified Design

Principle:

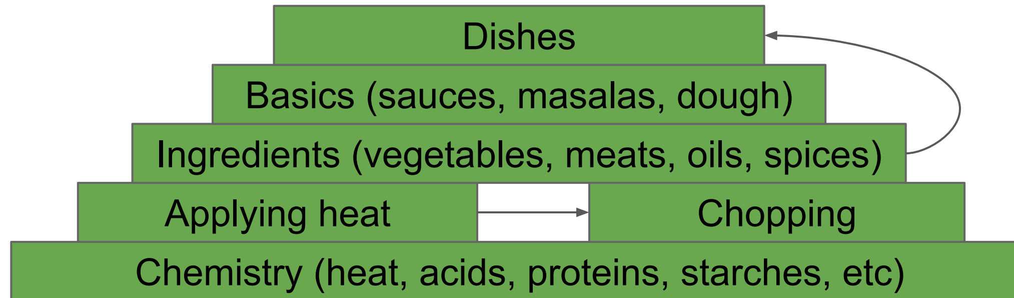
each layer can only require downward



Stratified Design

Principle:

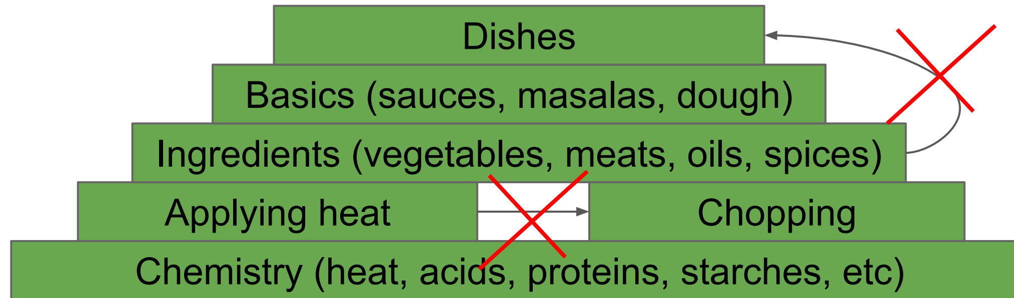
each layer can only require downward



Stratified Design

Principle:

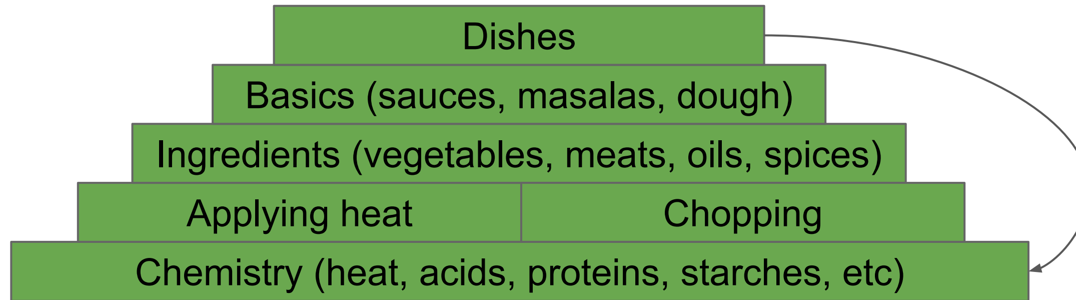
each layer can only require downward



Stratified Design

Principle:

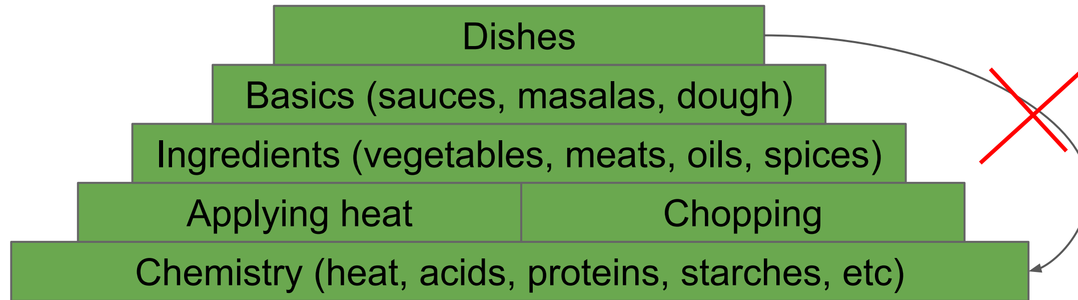
be careful if you skip layers



Stratified Design

Principle:

be careful if you skip layers



Constructors

Functions for generating entities

Benefits

- Define the entity's keys
- Check required keys
- Check types
- Check invariants
- Default values
- Calculated values

Constructors

```
(defn ->person [& {:keys [name
                          trainings]
                   :or {trainings []}}]
  (assert (string? name))
  (let [name (string/trim name)]
    (assert (not (empty? name)))
    {:name name
     :trainings trainings}))
```

default

type + required

calculation

invariant

key names

Combining Operations

Functions that combine 2 or more entities of the same type

Benefits

- Hardest type of operation first
- Constrain the design (good thing)
- Biggest potential for algebraic reasoning

Combining operations

```
(ns my-app.training)

(defn combine-training-status [a b]
  (cond
    (= :pass a) :pass
    (= :pass b) :pass
    :else       :fail))

(defn combine-trainings [a b]
  (merge-with combine-training-status a b))
```

two statuses

two training records

Meaning

Implementation

Meaning

Sounds

Software

Characters

Meaning

you want to be working up here

Data

Meaning
you want to be working up here
but you're stuck down here
Data

Code Smells

Smell:

Deep paths with `get-in`, `update-in`, `assoc-in`

Smell:

Large hashmaps with lots of keys

```
{:id 1232
 :email "eric@lispcast.com"
 :phone "504-982-9167"
 :street1 "332 Main St"
 :street2 "Apartment 54"
 :city "New Orleans"
 :state "LA"
 :zip "70117"
 :favorite-color "blue"
 :favorite-animal :orangutan
 :favorite-fruit :apple}
```

```
{:id 1232
 :email "eric@lispcast.com"
 :phone "504-982-9167"
 :street1 "332 Main St"
 :street2 "Apartment 54"
 :city "New Orleans"
 :state "LA"
 :zip "70117"
 :favorite-color "blue"
 :favorite-animal :orangutan
 :favorite-fruit :apple}
```

```
{:id 1232
 :contact-info {:email "eric@lispcast.com"
                :phone "504-982-9167"
                :address {:street1 "332 Main St"
                          :street2 "Apartment 54"
                          :city "New Orleans"
                          :state "LA"
                          :zip "70117"}}
 :favorites {:color "blue"
             :animal :orangutan
             :fruit :apple}}
```

Smell:

Using 3rd party API results directly

```
{:tags [21 134 353 386 388],
 :format "standard",
 :date "2017-04-24T11:59:33",
 :slug "kjetil-valle-and-bendik-solheim-lambdaconf-2017-interview",
 :meta {:_edd_button_behavior []},
 :ping_status "closed",
 :yst_prominent_words [3680 979 1318 497 505 849 3783 495 3938],
 :featured_media 8087,
 :content
 {:rendered
  "<p>......",
  :protected false},
 :modified_gmt "2018-04-11T11:31:44",
 :excerpt
 {:rendered
  "<p>......",
  :protected false},
 :type "post",
 :custom
 {:seen_in_lesson_rss "0",
  :markdown_content
  "......",
  :s3_filename "",
  :availability "Paid",
  :wistia_download_link "",
  :tweet_ids "",
  :tweet
  "....",
  :video_length "",
  :footer_script "",
  :language "a:1:{i:0;s:7:\"Clojure\";}",
  :inline_featured_image "0",
  :last_tweet_time "",
  :git_tag "",
  :github_repo "",
  :header_script "",
  :wistia_id "",
  :autopause_times ""},
 :template "",
 :modified "2018-04-11T11:31:44",
 :title
 {:rendered
  "# ...."},
 :author 652,
 :date_gmt "2017-04-24T11:59:33",
 :comment_status "closed",
 :categories [123],
 :sticky false,
 :status "draft",
 :link "https://purelyfunctional.tv/?p=8086",
 :id 8086,
 :_links
 {:version-history
```

```clojure
{:id 143
 :video-id "44fvv3"
 :sequence-number "003"
 :name "Concurrent Clojure"
 :duration 34223
 :programming-language 4
 :cover nil
 :course-id 74
 :s3-links {:original {:filename "ConcurrentClojure.mp4"
                       :filesize 321554}
            :hd {:filename "ConcurrentClojure.mp4"
                 :filesize 321443}
            :md {:filename "ConcurrentClojure.mp4"
                 :filesize 54443}
            :phone {:filename "ConcurrentClojure.mp4"
                    :filesize 3223}}}
```

Know what level of meaning you are at

- Look for semantic layers
- Avoid crossing semantic layers
- Organize your entities and operations
- Think about combining operations
- Add a layer of indirection between your service and other systems
- Use constructors to make getting it right easier
- Name operations to give things meaning

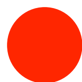
Eric Normand

LispCast

Follow Eric on:

Eric Normand

@EricNormand

 lispcast.com

eric@lispcast.com