RE-FRAME

LET'S MAKE A CLOCK (WOW!)

TO UNDERSTAND RE-FRAME, YOU MUST BE ABLE TO LIVE WITHOUT IT.

BOOT

https://github.com/boot-clj/boot

BOOT TEMPLATES

```
boot -d seancorfield/boot-new new -t
  tenzing -n <name> [+a <option>]*
```

```
$ cd <projects dir>
$ boot -d seancorfield/boot-new new -t tenzing -n clock-reagent -a +reagent
$ cd clock-reagent
$ boot dev
# Open Chrome on localhost:3000
$ atom .
```

```
<body>
  <div id="container"></div>
  <script type="text/javascript" src="js/app.js"></script>
</body>
```

```
(ns clock-reagent.app
  (:require [reagent.core :as r]))

(defn clock-view []
  [:h1 "clock"])

(defn init []
  (r/render-component [clock-view]
                      (.getElementById js/document "container")))
```

```
(ns clock-reagent.app
  (:require [reagent.core :as r]))

(def db
  (r/atom (js/Date.)))

(defn clock-view []
  [:h1 (.toLocaleTimeString @db)])

(defn init []
  (r/render-component [clock-view]
                      (.getElementById js/document "container")))
```

```
(ns clock-reagent.app
  (:require [reagent.core :as r]))

(def db
  (r/atom (js/Date.)))

(defn tick! []
  (reset! db (js/Date.)))

(defn clock-view []
  [:h1 (.toLocaleTimeString @db)])

(defn init []
  (.setInterval js/window tick!)
  (r/render-component [clock-view]
                      (.getElementById js/document "container")))
```

RE-FRAME

```
$ cd ..
$ boot -d seancorfield/boot-new new -t tenzing -n clock-reframe -a +reagent
$ cd clock-reframe
$ boot dev
# Open Chrome on localhost:3000
$ atom .
# Edit build.boot, adding [re-frame "0.9.4"] dependency
```

```
(ns clock-reframe.app
  (:require [re-frame.core :as rf]
            [reagent.core :as re]))

(defn clock-view []
  [:h1 "clock"])

(defn init []
  (re/render-component [clock-view]
                       (.getElementById js/document "container")))
```

6 DOMINOES ...

```
event dispatch  ->
event handling  ->
effect handling ->
query handling  ->
view rendering  ->
DOM update      -> ...
```

... AND THEIR FUNCTIONS

```
event dispatch  -> rf/dispatch-sync, rf/dispatch
event handling  -> rf/reg-event-db, rf/reg-event-fx
effect handling -> re-frame, rf/reg-fx
query handling  -> rf/reg-sub
view rendering  -> rf/subscribe
DOM update      -> react
```

```
(defn init []
  (rf/dispatch-sync [:initialize])
  (re/render-component...

(rf/reg-event-db
  :initialize
  (fn []
    {:date (js/Date.)}))
```

```
(defn clock-view []
  (let [t @(rf/subscribe [:local-time])]
    [:h1 t]]))

(rf/reg-sub
  :local-time
  (fn [db _]
    (-> db
      :date
      (.toLocaleTimeString))))
```

```
(defn clock-view []
  (let [t @(rf/subscribe [:local-time])]
    [:h1 t]]))

(rf/reg-sub
  :local-time
  (fn [db _]
    (-> db
      :date
      (.toLocaleTimeString))))
```

```
(defn init []
  (rf/dispatch-sync...
  (.setInterval js/window #(rf/dispatch [:tick]) 1000)
  (r/render-component...

(rf/reg-event-db
  :tick
  (fn [db _]
    (assoc-in db [:date] (js/Date.))))
```

```
(ns clock-reframe.app
  (:require [re-frame.core :as rf]
            [reagent.core :as re]))


;; EVENT HANDLERS
(rf/reg-event-db
  :initialize
  (fn []
    {:date (js/Date.)}))


(rf/reg-event-db
  :tick
  (fn [db _]
    (assoc-in db [:date] (js/Date.))))


;; QUERY (aka SUBSCRIPTIONS)
(rf/reg-sub
  :local-time
  (fn [db _]
    (-> db
      :date
      (.toLocaleTimeString))))


;; VIEWS
(defn clock-view []
  (let [t @(rf/subscribe [:local-time])]
    [:h1 t]))


;; LAUNCH POINT
(defn init []
  (rf/dispatch-sync [:initialize])
  (.setInterval js/window #(rf/dispatch [:tick]) 1000)
  (r/render-component [clock-view] (.getElementById js/document "container")))
```

NOW, LET'S MAKE A GAME.