

# Serverless Applications with ClojureScript and Firebase

Jake McCrary  
@jakemcc

# Who am I?

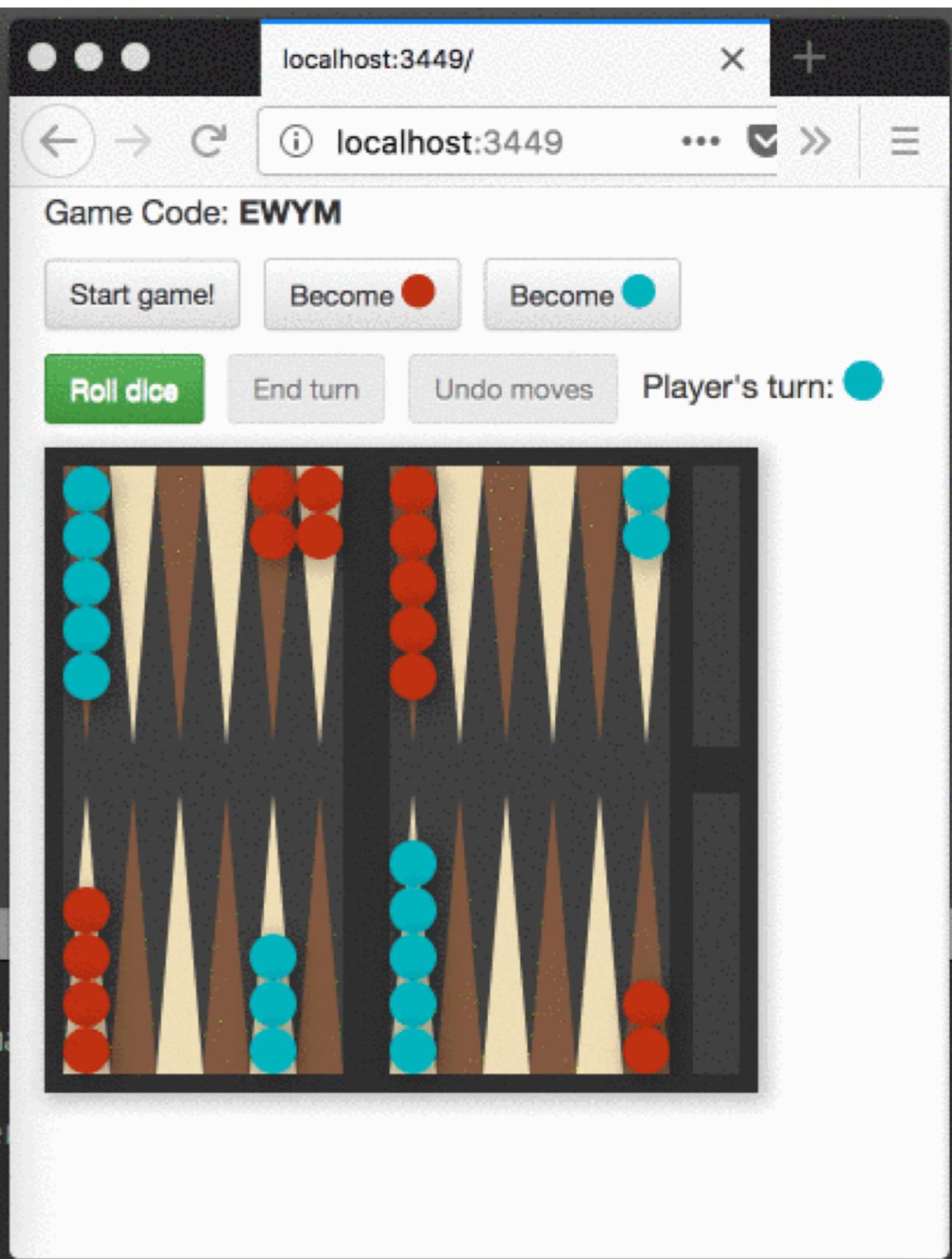
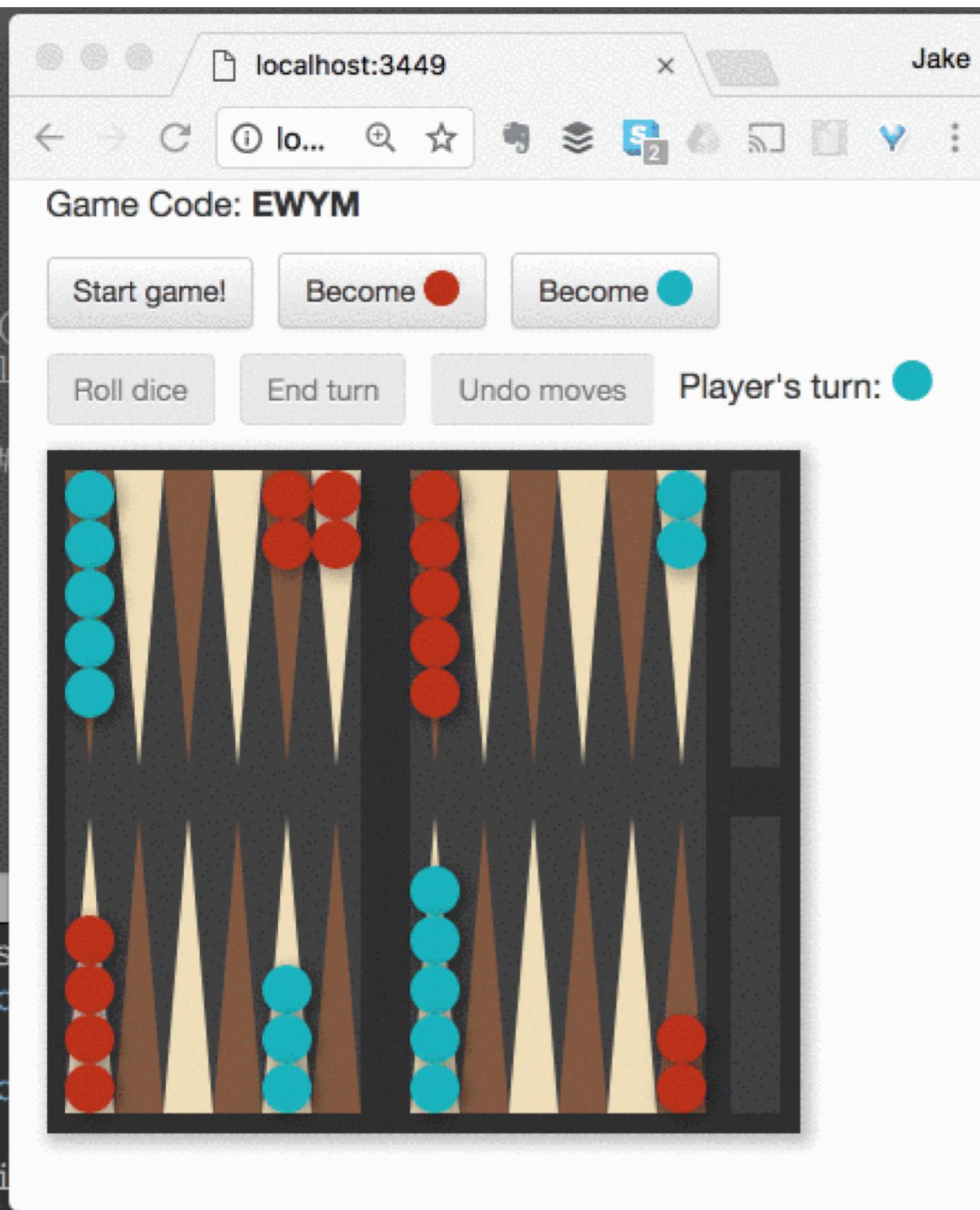
- >> Write at [jakemccrary.com](http://jakemccrary.com)
- >> Author of lein-test-refresh and lein-autoexpect
- >> Developer at Outpace Systems
- >> Adviser for Metonymy Labs

# Serverless Applications with ClojureScript and Firebase

# Serverless

## Applications Games

with ClojureScript and Firebase



Why?

Why?  
quick

Why?  
Learn  
new tools

Why?

H u n



Photo credit: Arto Alanenpää, <https://commons.wikimedia.org/wiki/File:Legodubloartoalanenpaa5.JPG>, CC BY-SA 4.0



cijS

# Reagent

# Reagent

```
(defn simple-component []
  [:div
   [:p "I am a component!"]
   [:p.someclass
    "I have " [:strong "bold"]
    [:span {:style {:color "red"}}] " and red "] "text."]])
```

# Reagent: ratoms

```
(def click-count (reagent/atom 0))

(defn counting-component []
  [:div
    "The atom " [:code "click-count"] " has value: "
    @click-count
    [:input {:type "button" :value "Click me!"
             :on-click #(swap! click-count inc)}]])
```



**reframe**

SPAs are full of  
mutation

re-frame isolates  
the mutation

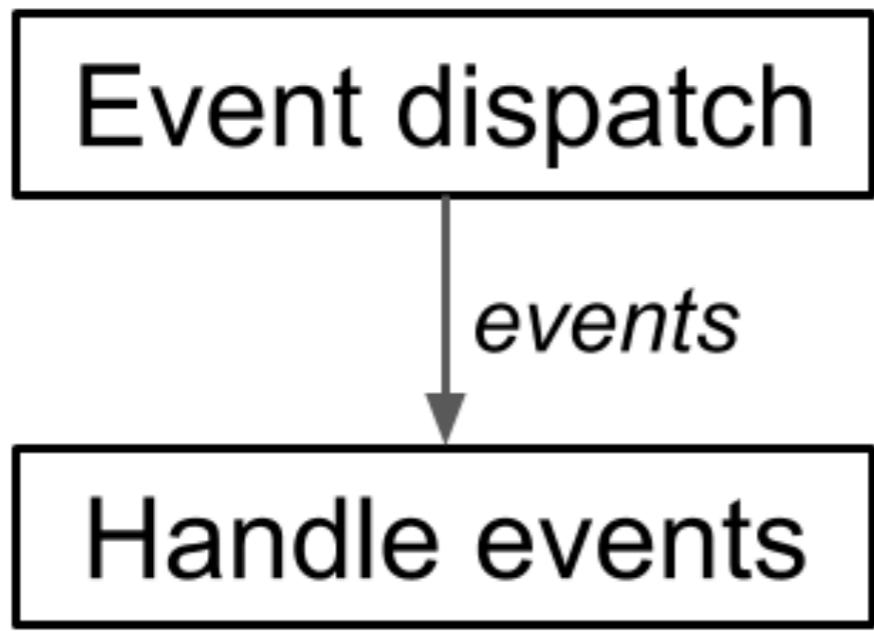
Event dispatch

# re-frame: event dispatch

```
(rf/dispatch [:join-game (random-code)])
```

Event dispatch





# re-frame: handle event

```
(rf/reg-event-fx
:join-game
(fn [coeffects [_ game-id]]
(let [db (:db coeffects)]
{:db (assoc db :game-id game-id)
:firebase/subscribe {:game-id game-id
:default db/initial-game}}))))
```

# re-frame: handle event

```
(rf/reg-event-fx
:join-game
(fn [coeffects [_ game-id]]
(let [db (:db coeffects)]
{:db (assoc db :game-id game-id)
:firebase/subscribe {:game-id game-id
:default db/initial-game}}))))
```

# re-frame: handle event

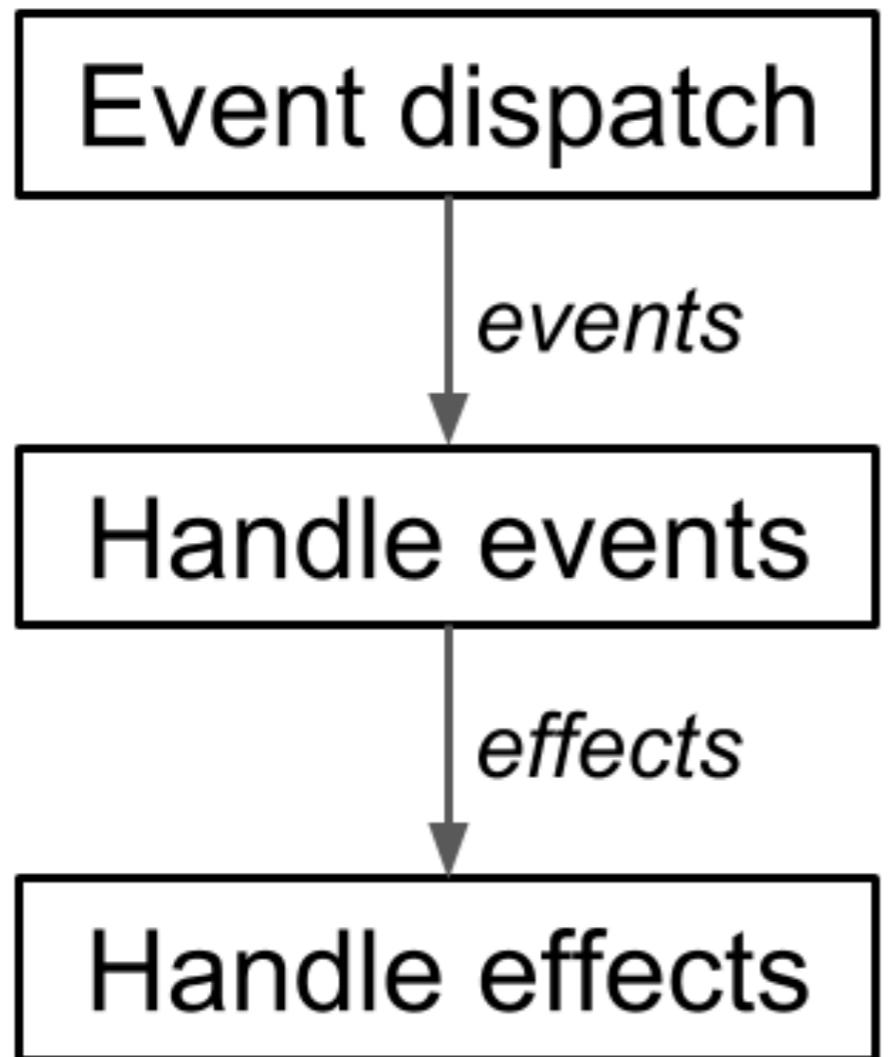
```
(rf/reg-event-db
 :sync
 (fn [db [_ data]]
  (assoc db :external data))))
```

**Event dispatch**

*events*

**Handle events**

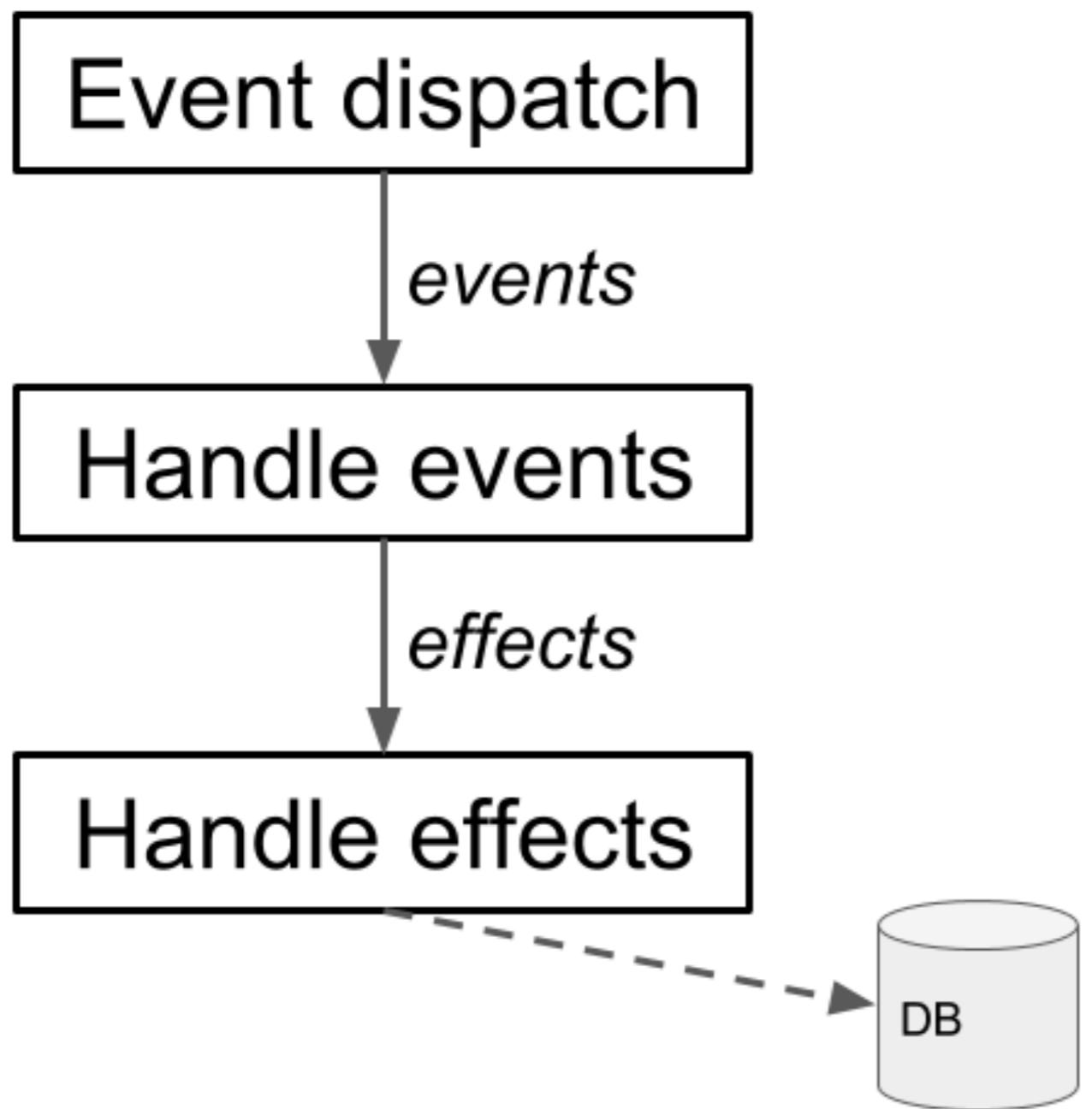
*effects*

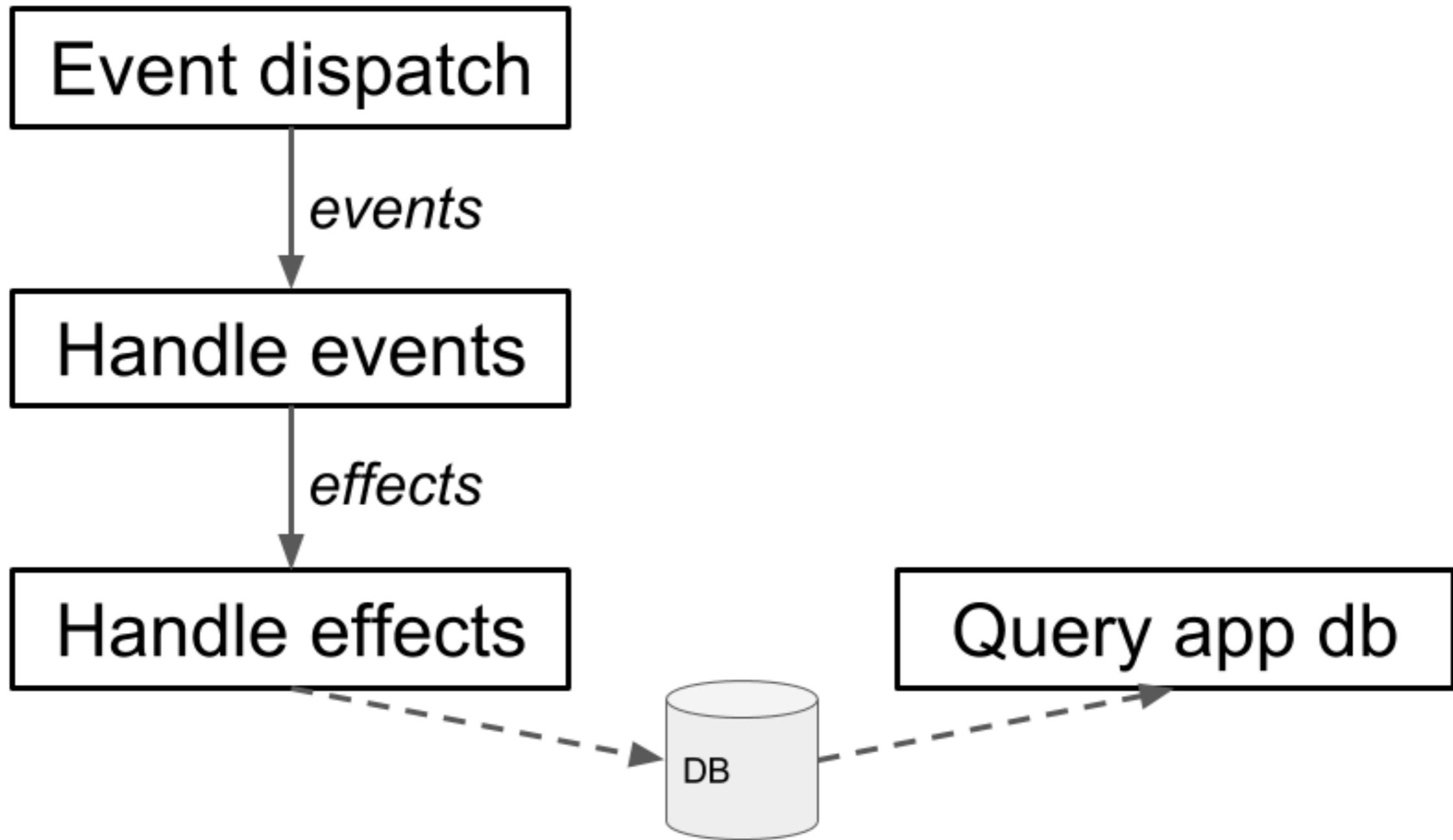


# re-frame: Handle effects

```
;; from re-frame.fx namespace
```

```
(reg-fx
  :db
  (fn [value]
    (if-not (identical? @app-db value)
      (reset! app-db value)))))
```





# re-frame: query app db

```
(rf/reg-sub  
  :game-id  
  (fn [db]  
    (:game-id db))))
```

# re-frame: query app db

```
(rf/reg-sub  
  :my-turn?  
  (fn [_ _]  
    [(rf/subscribe [:me])  
     (rf/subscribe [:current-player])])  
  (fn [[me current-player]]  
    (same-player? me current-player))))
```

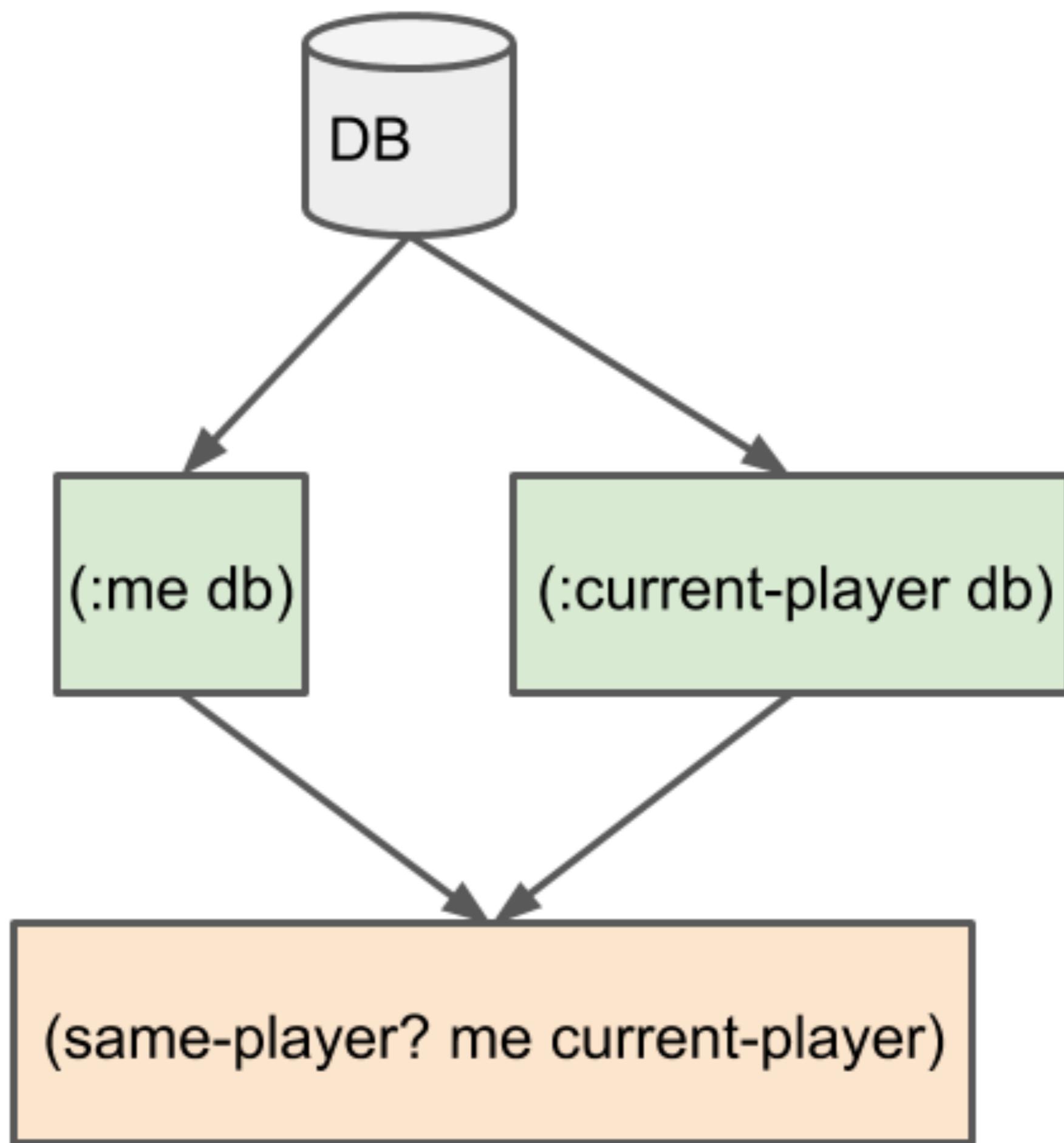
# re-frame: query app db

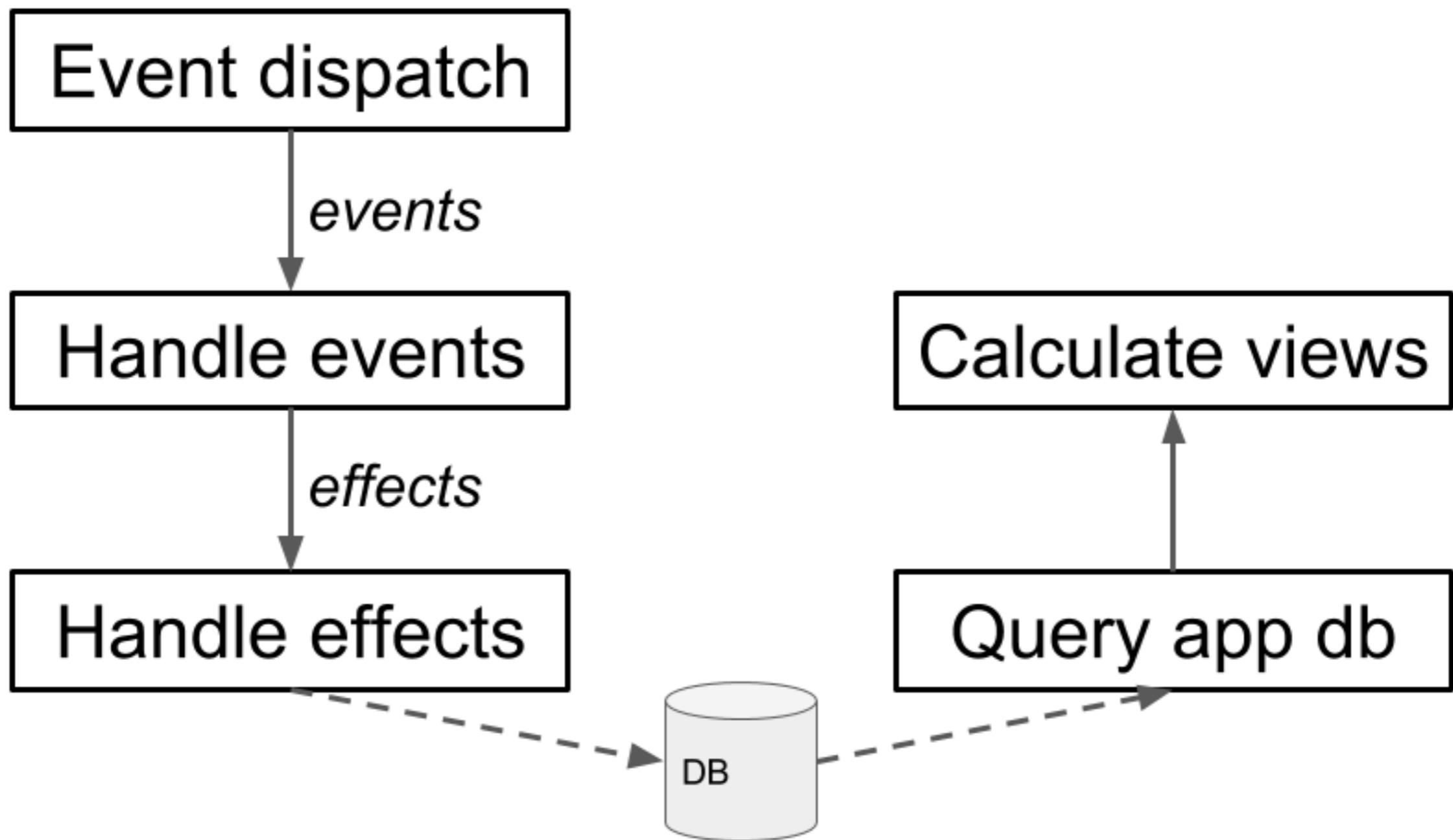
```
(rf/reg-sub  
  :my-turn?  
  (fn [_ _]  
    [(rf/subscribe [:me])  
     (rf/subscribe [:current-player])])  
  (fn [[me current-player]]  
    (same-player? me current-player))))
```

# re-frame: query app db

```
(rf/reg-sub  
  :my-turn?  
  (fn [<-->]  
    [(rf/subscribe [:me])  
     (rf/subscribe [:current-player])])  
  (fn [[me current-player]]  
    (same-player? me current-player))))
```

```
(rf/reg-sub
 :my-turn?
 :<- [:me]
 :<- [:current-player]
 (fn [[me current-player]]
  (same-player? me current-player))))
```



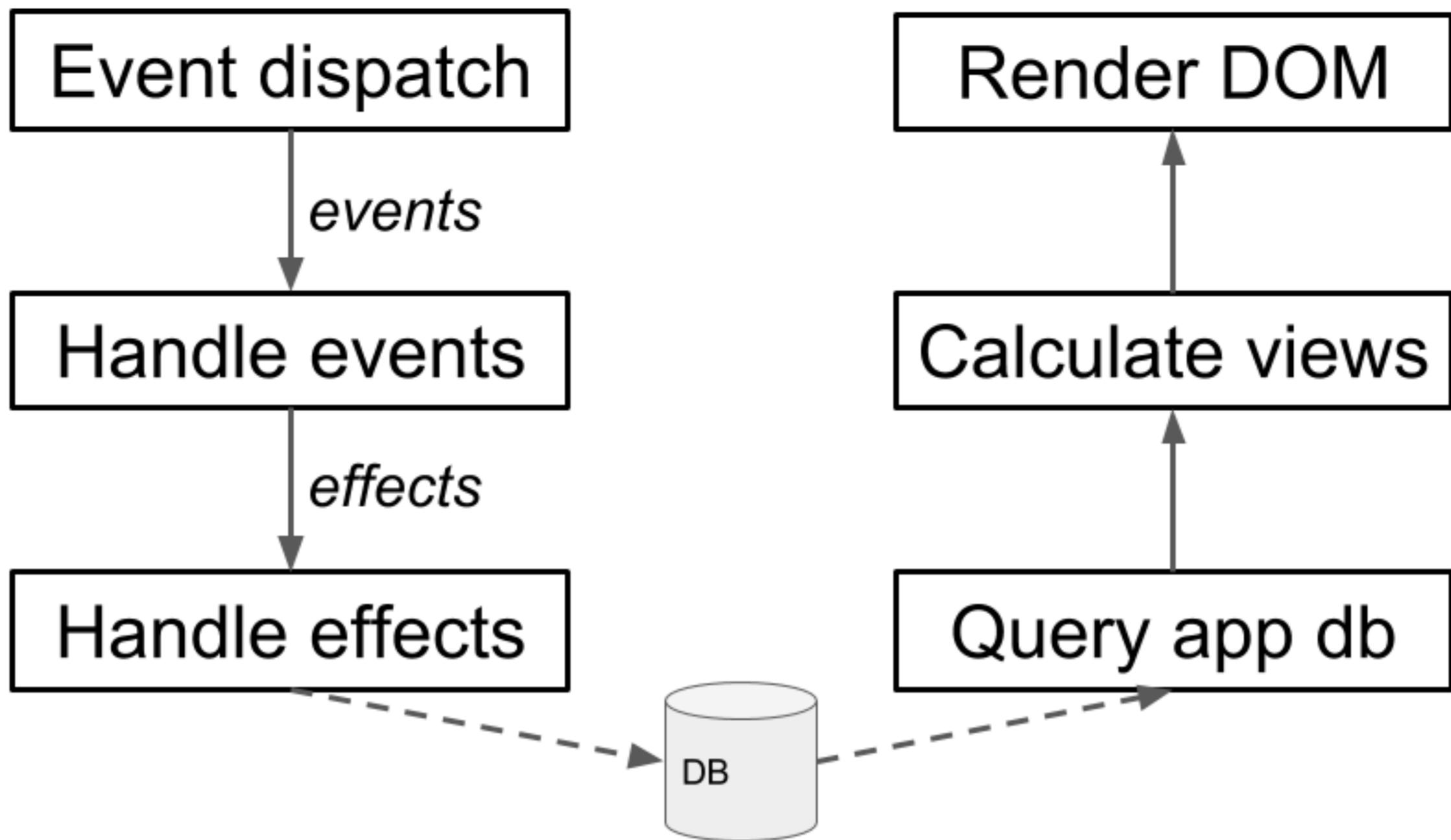


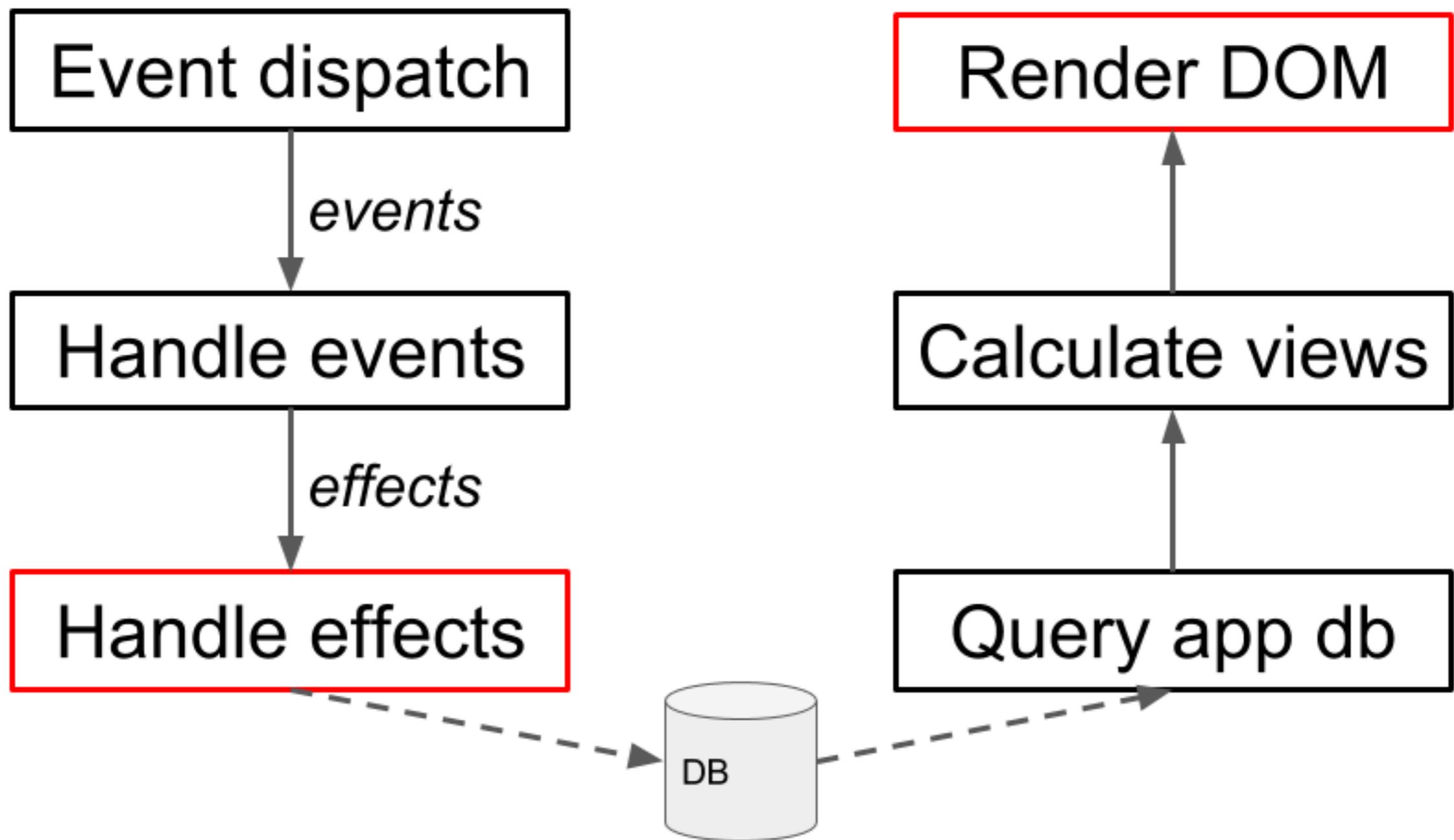
# re-frame: calculate views

```
(defn main-panel []
  [:div
    [:div {:class "container"}
      (if @(rf/subscribe [:game-id])
          [on-going-game-view]
          [start-game-view])]]))
```

# re-frame: calculate views

```
(defn main-panel []
  [:div
    [:div {:class "container"}
      (if @(rf/subscribe [:game-id])
          [on-going-game-view]
          [start-game-view])]]))
```





# Additional re-frame

>> spec the database

# Additional re-frame

- >> spec the database
- >> interceptors

# Additional re-frame

- >> spec the database
- >> interceptors
- >> use namespaced keywords :panel1/edit  
and :panel2/edit

# re-frame: local reagent/atom

```
(defn todo-input [title]
  (let [val (reagent/atom title)
        save #(rf/dispatch [:save (-> @val str string/trim)])]
    (fn [props]
      [:input (merge props
                      {:value @val
                       :on-blur save
                       :on-change #(reset! val (-> % .-target .-value)))
                     :on-key-down #(case (.-which %)
                                     13 (save)
                                     27 (fn [] (reset! val ""))
                                     nil)}))))
```



# Firebase

messaging

authentication

storage

cloud functions

hosting

realtime database

**Firebase**  
**Hosting**

Firebase

Realtime

Database

# Realtime Database

```
(defn init []
  (js/firebase.initializeApp
    #js {:apiKey "AIzaSyAb1IwbJHuNylWap1MFraBKVYRUFk4ShsY",
         :authDomain "bg-example-81c6b.firebaseio.com",
         :databaseURL "https://bg-example-81c6b.firebaseio.com",
         :projectId "bg-example-81c6b",
         :storageBucket "bg-example-81c6b.appspot.com",
         :messagingSenderId "391548202910"}))
```

# Realtime Database

```
(defn db-ref [path]
  (.ref (js/firebase.database)
        (string/join "/" path)))

(defn save! [ref data]
  (.set ref (pr-str data)))
```

# Realtime Database: Subscribe

```
(defn subscribe [ref]
  (.on ref "value"
    (fn [snapshot]
      (when-let [d (.val snapshot)]
        (rf/dispatch [:sync (reader/read-string d)))))))
```



## How to draw an owl

1.



2.



1. Draw some circles

2. Draw the rest of the owl

Step 1

Step 1:  
Create a new project

lein new re-frame bg +re-frisk

# Step 2: Modify template

1. Update dependencies

# Step 2: Modify template

1. Update dependencies
2. `rm -rf src/clj`

# Step 2: Modify template

1. Update dependencies
2. rm -rf src/clj
3. mv src/cljs/bg/core.cljs src/cljs/bg/  
main.cljs

# Step 2: Modify template

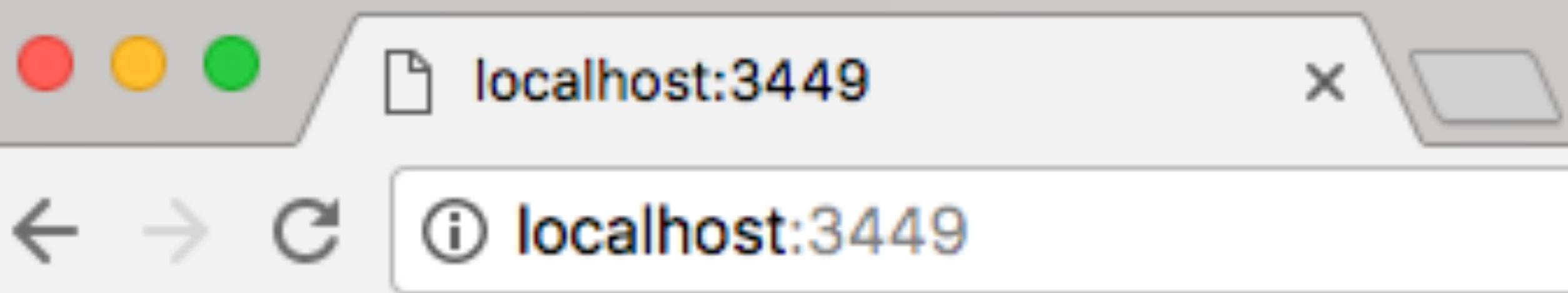
1. Update dependencies
2. rm -rf src/clj
3. mv src/cljs/bg/core.cljs src/cljs/bg/  
main.cljs
4. Update references from bg.core to bg.main.

# Step 2: Modify template

1. Update dependencies
2. rm -rf src/clj
3. mv src/cljs/bg/core.cljs src/cljs/bg/  
main.cljs
4. Update references from bg.core to bg.main.
5. Add [cljsjs/firebase "4.8.1-0"] dependency in  
project.clj

# Step 3: Build

lein figwheel dev



Hello from re-frame

# Hello from re-frame

A screenshot of a development environment showing a browser window and a code editor side-by-side.

The browser window on the left displays the text "Hello from re-frame" at the URL "localhost:3449".

The code editor on the right shows a file named "views.cljs" with the following content:

```
1 (ns backgammon.views
2   (:require [re-frame.core :as rf]
3             [backgammon.subs :as subs]))
4
5 (defn main-panel []
6   (let [name (rf/subscribe [:subs/name])]
7     [:div "Hello" @name]))
```

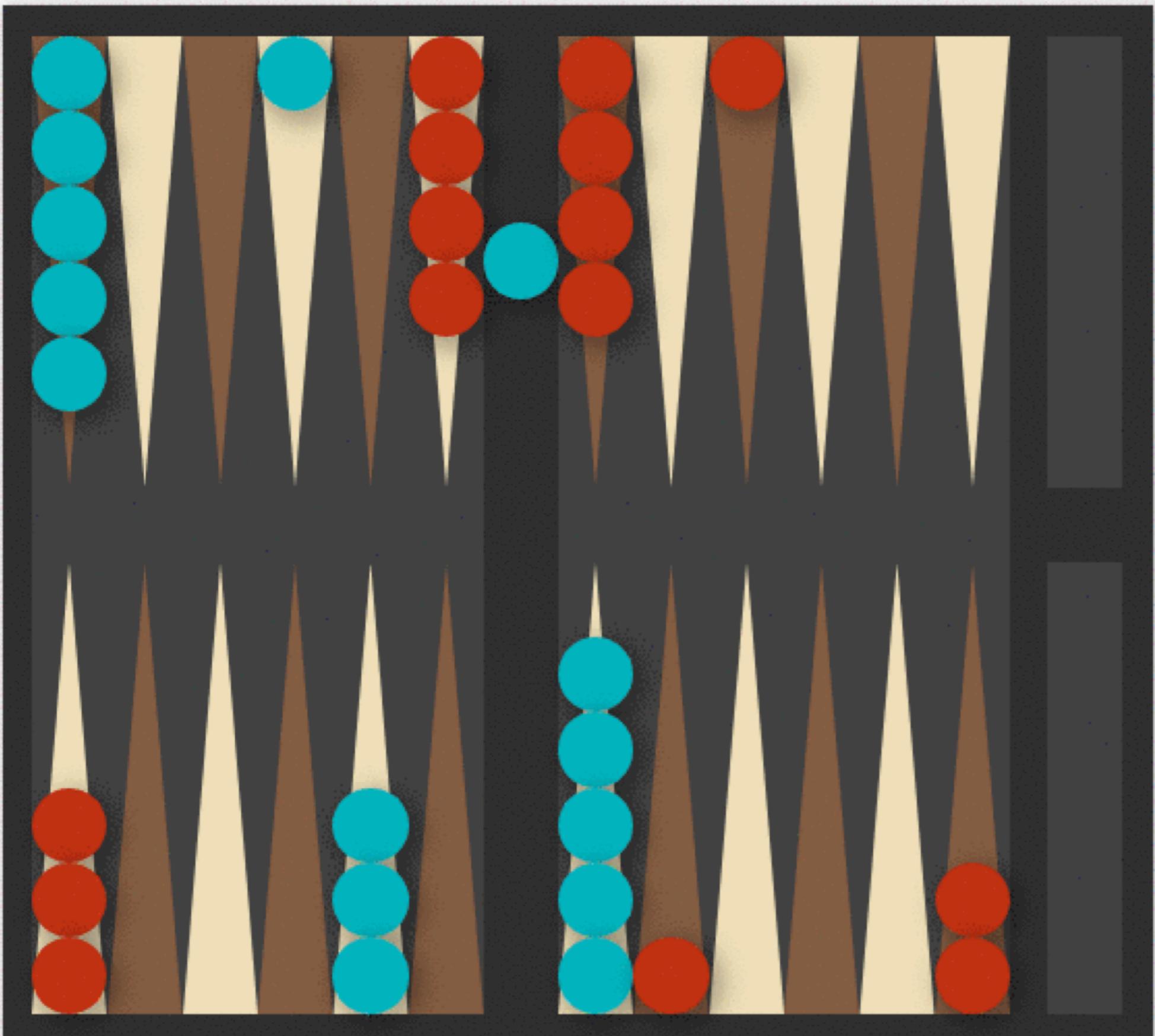
The code editor interface includes tabs for "views.cljs", "All L7", and "Git:master".

Roll dice

End turn

Undo moves

Player's turn: ●



► re-frisk



acquired her next business

# Step 4: Deploy

# Step 4: Deploy

1. Sign-in at <https://firebase.com>

# Step 4: Deploy

1. Sign-in at <https://firebase.com>
2. Go to console and add a new Firebase project

# Step 4: Deploy

1. Sign-in at <https://firebase.com>
2. Go to console and add a new Firebase project
3. Copy your project's configuration from the Firebase console



Welcome to Firebase!

# Get started here.



Add Firebase to  
your iOS app



Add Firebase to  
your Android app



Add Firebase to  
your web app



Receive email updates about new Firebase features, research,  
and events

SIGN UP



Discover Firebase

# Step 4: Deploy

1. Sign-in at <https://firebase.com>
2. Go to console and add a new Firebase project
3. Copy your project's configuration from the Firebase console
4. `npm install -g firebase-tools`

# Step 4: Deploy

1. Sign-in at <https://firebase.com>
2. Go to console and add a new Firebase project
3. Copy your project's configuration from the Firebase console
4. `npm install -g firebase-tools`
5. `firebase init`

# Step 4.5: Initialize project

```
$ firebase init
```

? Which Firebase CLI features do you want to setup for this folder?

- Database:** Deploy Firebase Realtime Database Rules
- Firestore:** Deploy rules and create indexes for Firestore
- Functions:** Configure and deploy Cloud Functions
- Hosting:** Configure and deploy Firebase Hosting sites
- Storage:** Deploy Cloud Storage security rules

# Step 4: Deploy

1. Sign-in at <https://firebase.com>
2. Go to console and add a new Firebase project
3. Copy your project's configuration from the Firebase console
4. `npm install -g firebase-tools`
5. `firebase init`
6. `lein do clean, cljsbuild once min`

# Step 4: Deploy

1. Sign-in at <https://firebase.com>
2. Go to console and add a new Firebase project
3. Copy your project's configuration from the Firebase console
4. `npm install -g firebase-tools`
5. `firebase init`
6. `lein do clean, cljsbuild once min`
7. `firebase deploy`



https://bg-example-81c6b.firebaseio.com



Secure

https://bg-example-81c6b.firebaseioapp.com

# Hello from re-frame

Step 5:  
Implement game  
logic

```
x      java      %1  x      bash      ● %2
```

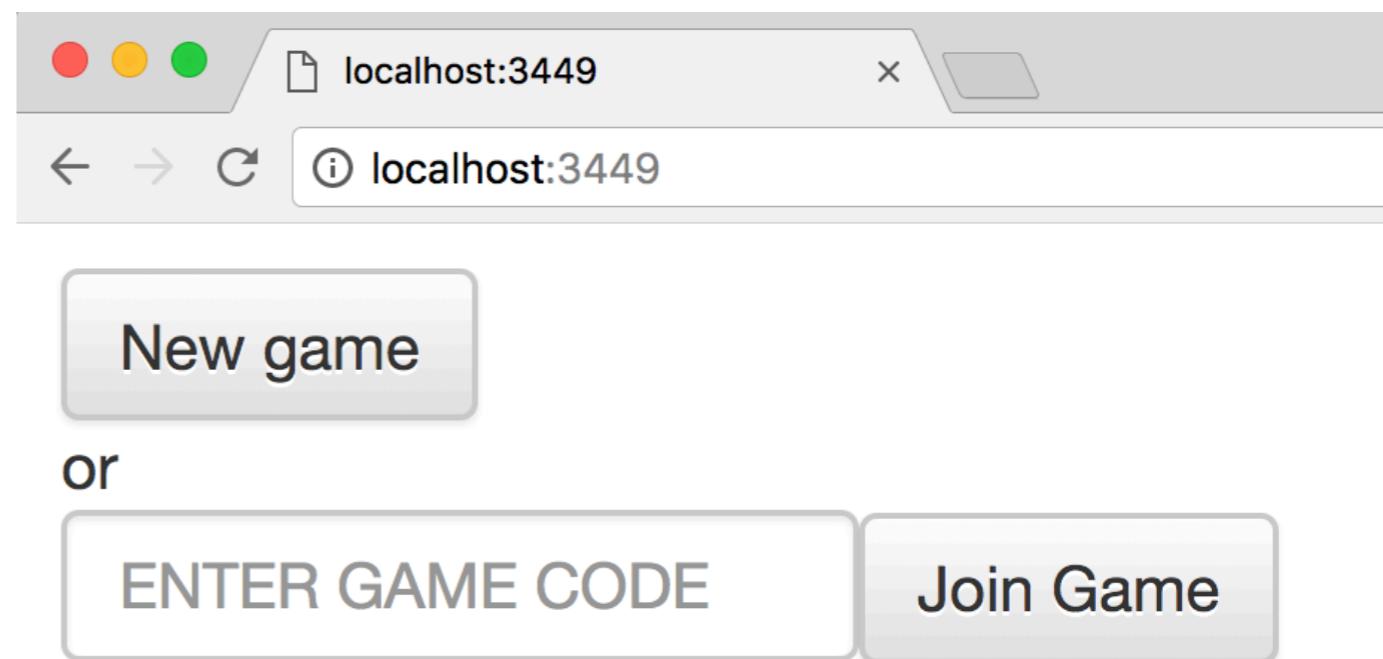
```
*****
***** Running tests *****
:reloading (re-frame.db re-frame.loggers re-frame
.registrar re-frame.interceptor re-frame.cofx re-
frame.utils re-frame.trace re-frame.subs re-frame
.std-interceptors re-frame.events re-frame.router
re-frame.fx re-frame.core bg.game bg.game-test)

Ran 7 tests containing 72 assertions.
0 failures, 0 errors.

Passed all tests
Finished at 07:07:55.858 (run time: 0.689s)
```

```
6
7 (deftest test-moving-piece
8   (is (= {:points [[:black] []]
9           :bar []})
10      (game/move-piece {:points [] [:b
11                                :bar []}
12                                :black
13                                1
14                                0})))
15
16  (is (= {:points [[:black] []]
17            :bar [:red]}
-:--- game_test.cljc 1% L8 (ClojureC
```

# Step 6: Implement views



# The New button

```
[::div.row
  [:button {:class ["btn" "btn-default"]
            :on-click #(rf/dispatch [:join-game
                                      (random-four-characters)])}
    "New game"]]
```

# The New button

```
[::div.row
  [:button {:class ["btn" "btn-default"]
            :on-click #(rf/dispatch [:join-game
                                      (random-four-characters)])}
    "New game"]]
```

# :join-game event handler

```
(rf/reg-event-fx
:join-game
(fn [coeffects [_ game-id]]
(let [db (:db coeffects)]
{:db (assoc db :game-id game-id)
:firebase/subscribe {:game-id game-id
:default db/initial-game}}))))
```

# :join-game event handler

```
(rf/reg-event-fx
:join-game
(fn [coeffects [_ game-id]]
(let [db (:db coeffects)]
{:db (assoc db :game-id game-id)
:firebase/subscribe {:game-id game-id
:default db/initial-game}}))))
```

# :firebase/subscribe effect handler

```
(rf/reg-fx
  :firebase/subscribe
  (fn [{:keys [game-id default]}]
    (let [ref (db-ref [game-id])]
      (.once ref "value"
        (fn received [snapshot]
          (subscribe ref)
          (if-let [data (.val snapshot)]
            (rf/dispatch [:sync (reader/read-string data)])
            (do (save! ref default)
                (rf/dispatch [:sync default])))))))))
```

# :firebase/subscribe effect handler

```
(rf/reg-fx
  :firebase/subscribe
  (fn [{:keys [game-id default]}]
    (let [ref (db-ref game-id)]
      (.once ref "value"
        (fn received [snapshot]
          (subscribe ref)
          (if-let [data (.val snapshot)]
            (rf/dispatch [:sync (reader/read-string data)])
            (do (save! ref default)
                (rf/dispatch [:sync default])))))))))
```

# :firebase/subscribe effect handler

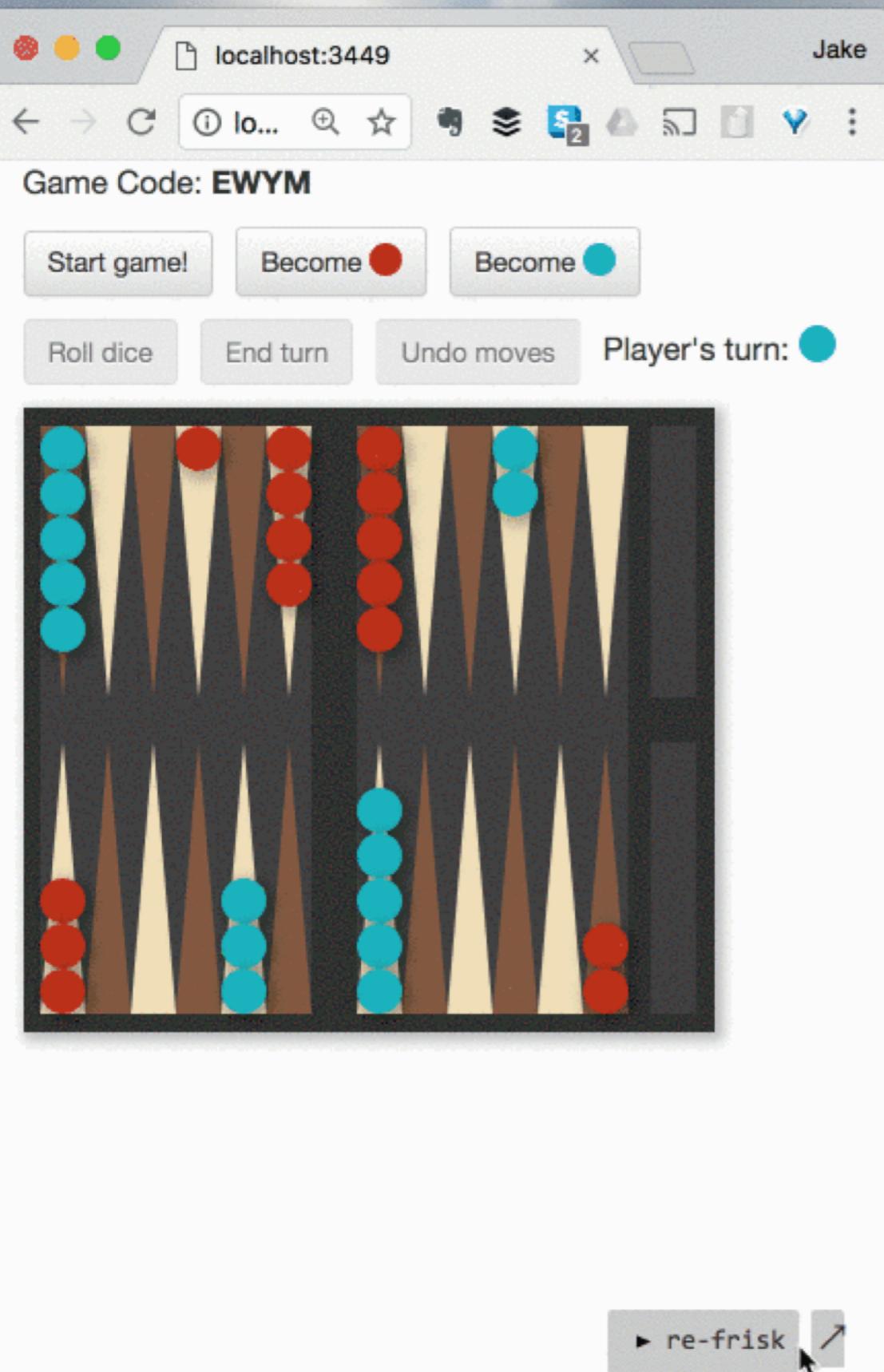
```
(rf/reg-fx
  :firebase/subscribe
  (fn [{:keys [game-id default]}]
    (let [ref (db-ref game-id)]
      (.once ref "value"
        (fn received [snapshot]
          (subscribe ref)
          (if-let [data (.val snapshot)]
            (rf/dispatch [:sync (reader/read-string data)])
            (do (save! ref default)
                (rf/dispatch [:sync default])))))))))
```

# :firebase/subscribe effect handler

```
(rf/reg-fx
  :firebase/subscribe
  (fn [{:keys [game-id default]}]
    (let [ref (db-ref game-id)]
      (.once ref "value"
        (fn received [snapshot]
          (subscribe ref)
          (if-let [data (.val snapshot)]
            (rf/dispatch [:sync (reader/read-string data)])
            (do (save! ref default)
                (rf/dispatch [:sync default])))))))))
```

Step 7: Play  
game!

# Tips



# Review

CLJS + reagent +  
re-frame is a  
compelling stack  
for building SPAs

CLJS tooling  
provides quick  
feedback cycles

Firebase provides  
good tools for  
serverless  
programming

# More resources

- >> #re-frame and #reagent in Clojurians slack
- >> re-frame repo has a lot of documentation
- >> Eric Normand has some great articles on  
PurleyFunctional.tv
- >> <https://github.com/jakemcc/backgammon>

# Thank you

Jake McCrary  
<https://jakemccrary.com>  
jake@jakemccrary.com  
@jakemcc

