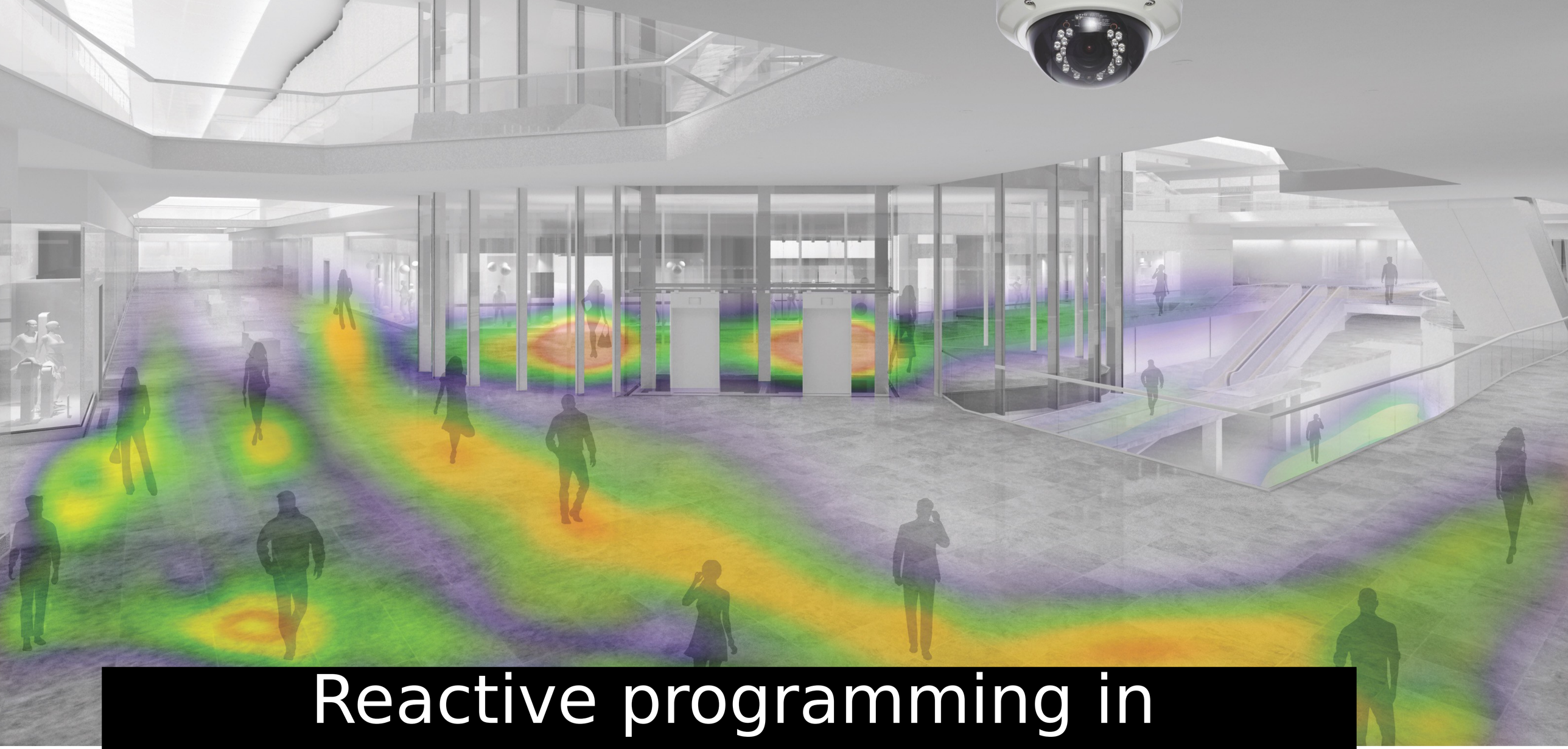


Reactive programming in ClojureScript using React.js wrappers

Konrad Szydlo @ Lambda Days, Krakow, 19 February 2016

retailic
FOLLOW THE CUSTOMER

AlephVision™

State of the web development

- More and more complex apps
  - Solving elaborate problems
  - Myriad of web browsers and devices
  - Performance and resources

- Plain vanilla JavaScript

- JQuery

- MVC

- React.js

What does it mean reactive?

- Erik Meijer - What does it mean to be Reactive?
  - Lots of definitions
  - Difficult to agree

- Components reacting to events and changing state

- Apples
- Oranges
- Grapes

- Add pineapple
  - But how exactly?

Solution #1

- Insert a new list item
  - Find the right place

Solution #2

- Throw out the entire list and rebuild it
  - Rendering large scale amounts of DOM is slow

- Most JS frameworks help to mitigate problem #1
  - Make DOM easier to navigate
  - Tie element to a controller keeping it up to date

- React solves problem #2

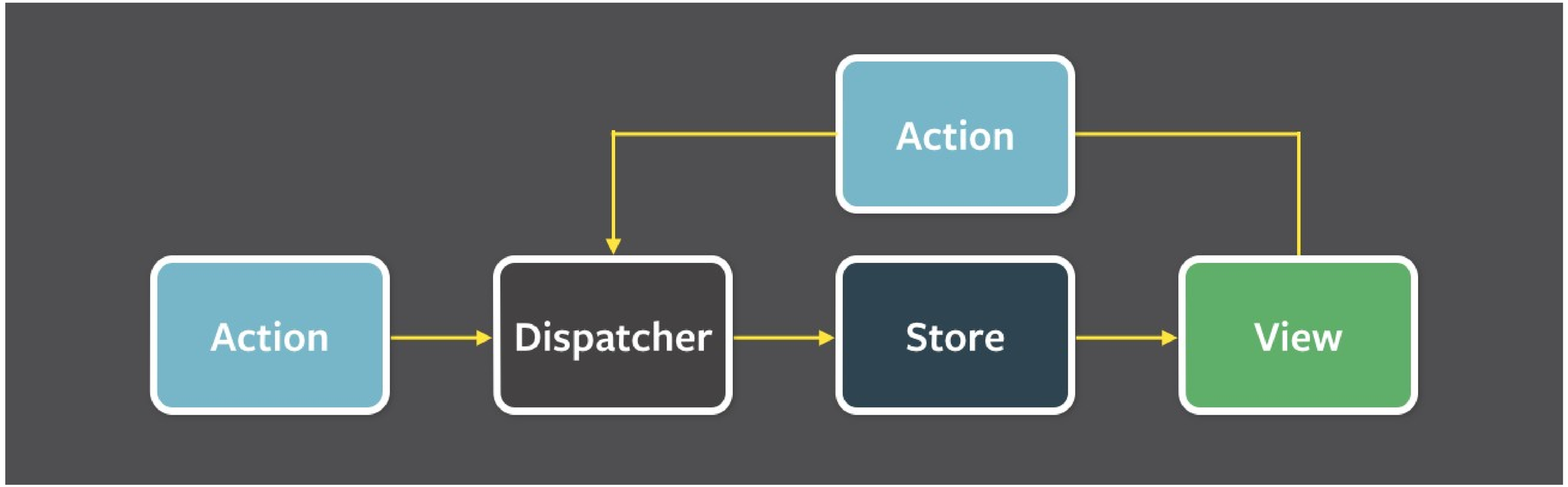
- Virtual DOM (two copies)
  - Original
  - Updated

- Compare new virtual DOM with previous
- Render only bits that have changed

- Pretending to re-render entire page is fast
- Easier to reason about

- V in MVC
  - Interfaces with changing data
- Can combine with frameworks e.g. AngularJS

- Flux architecture
  - Once-directional data flow
  - Action → Dispatcher → Store → View



JavaScript sucks

- this
- Verbose syntax
- Confusing, inconsistent automatic operator conversions
- NaN
- Global variables by default
- etc.

We need JavaScript

- JS is (almost) everywhere
- Runtime is improving
  - Web Sockets
  - WebRTC
  - Canvas
  - Push notifications

- Build “better” language on top of JS

- `1 + 2` // simple, clear
- `(+ 1 2)` ; OMG! So confusing
- Clearly programmers aren't that smart as some think

ClojureScript

- Modern LISP
- Immutable data structures
- State vs Identity
- REPL
- Transducers
- core.async
- And much much more

- ClojureScript + React.js

- Pure React.js is not idiomatic in ClojureScript
- Leverage immutable data structures

- ClojureScript React.js wrappers on steroids

Rum

- Not a framework but a library

- Om, Reagent, Quiescent have built-in component behaviour model

Rum

- Two-level API

Rum

- Define your component and render it

- (rum/defc name doc-string? [params*] render-body+)

- (rum/defc h1 [text]
  [:h1 text])

- (rum/mount (h1 "important") js/document.body)

- Simple component not reactive
- Agnostic of data storage and behaviour model

- Use mixins to control component's behaviour

- Pre-built component mixins
- Mimic behaviour of other wrappers

- (rum/defc name doc-string? [< mixins+]? [params*] render-body+)

- Static mixin
  - Checks if args have changed
  - Avoids re-rendering if they are the same

```
(rum/defc h1 < rum/static [text]
  [:h1 text])
```

- (rum/mount (h1 "important text") body)

- (rum/mount (h1 "important text") body) ;; render won’t be called
- (rum/mount (h1 "important") body) ;; this will cause re-rendering

Local Mixin

- Rum local
  - Per-component local state

Our Services

Web Development	**£300**
Design	**£400**
Integration	**£250**
Training	**£220**
Total: £0	

Our Services

Web Development	£300
Design	£400
Integration	£250
Training	£220
Total: £300	

Our Services

Web Development	£300
Design	£400
Integration	£250
Training	£220
Total: £520	

```
(rum/defcs make-service < (rum/local "") [state service]
  (let [local (:rum/local state)]
    [:p {:class @local
         :on-click (fn [_] (swap! local toggle-class service))}
     (:name service)
     [:b (str " £" (:price service))]]))
```

```
(defn toggle-class [current-class service]
 (let [price (:price service)]
  (if (= current-class "active")
    (do (swap! order-form-total - price)
        "")
    (do (swap! order-form-total + price)
        "active"))))
```

Reactive mixin

- Creates “reactive” component
- Tracks references to data
- Auto-updates when value stored changes
- Reactive mixin is like Reagent wrapper

Our Services

Web Development	£300
Design	£400
Integration	£250
Training	£220
Total: £520	

- (def order-form-total (atom 0))

- (rum/defc total-sum < rum/reactive []
  [:p {:id "total"} (str "Total: £" (rum/react order-form-total))])

```
(defn toggle-class [current-class service]
 (let [price (:price service)]
   (if (= current-class "active")
     (do (swap! order-form-total - price)
         "")
     (do (swap! order-form-total + price)
         "active"))))
```

Cursored mixin

- Interface to sub-trees inside an atom
- Like an Om wrapper

```
(def state (atom {:app {
                   :settings {:header {:colour "#cc3333" }}}}))
```

```
(rum/defc label < rum/cursored [colour text]
  [:label {:style {:colour @colour}} text])
```

```
(rum/defc body < rum/cursored  [state]
  [:div
    (label (rum/cursor state [:app :settings :headers :colour])
"First label" )])
```

- (swap! state-cursor assoc-in [:app :settings :header :colour] "#cc56bb")

- One global state
  - Atomic swaps
  - Always consistent
  - Easy to implement undo – reset to previous state
  - Serialize app state and send it over the wire

Custom mixin

- Use React.js life-cycle functions
  - Will mount
  - Did mount
  - Should update
  - Will unmount

- Scrolling mixin

```
(defn scroll-view [height]
  {:did-mount      (fn [state]
                     (if-not (::scrolled state)
                       (do
                         (set-scroll-top! height)
                         (assoc state ::scrolled true))
                       state))
   :transfer-state (fn [old new]
                     (assoc new ::scrolled (::scrolled old)
                                ::with-scroll-view true))})
```

- Build new mixins easily

- Mix and match different mixins

- The curious case of Om

- The good, the bad and the future

- Root UI is given root cursor
- Passes sub-trees of cursors to its sub-components

```
{:list {:title "My list:" :colour :brown
   :items
    {:item_1 {:text "text1" :colour :blue}
     :item_2 {:text "text2" :colour :green}
     :item_3 {:text "text3" :colour :red}}}}
```

My list:

- text1
- text2
- text3

Case #1

- UI cross-cutting concerns?

- Dashboard with a list that can be expanded, shrank

- Save the list presentation options to general settings?

- Cursor-oriented architecture requires you to structure app state as a tree (matching UI)

- Quite often your data are not a tree

- Changing item's text is easy

Case #2

- Deleting an item should be easy too, right?

- Does delete button/option belong to an item or a list?

- Deleting an item requires list manipulation

- Rum is flexible use it if you can

Enter Om.next

- New version of Om

- Actually a reboot

- Experience of the last few years
- Inspired by:
  - Facebook's Relay
  - Netflix's Falcor
  - Cognitect's Datomic

- Display date of past purchases of certain item

```
{:current-user
 {:purchases
  [{:id 146
    :product-name "organic rice"
    :date #inst "2015-12-17T17:13:59.167-00:00"
    :previous {:id 140
               :product-name "organic rice"
               :date #inst "2015-12-14T17:05:58.144-00:00"
               :previous {:id 136
                          :product-name "organic rice"
                          :date #inst "2015-12-10T17:05:13.512-00:00"
                          :previous ; and so on... }}}
   {:id 145
    :product-name "soy milk"
    :date #inst "2015-12-17T17:09:25.961-00:00"
    :previous {:id 141
               :product-name "soy milk"
               :date #inst "2015-12-15T17:05:36.797-00:00"
               :previous {:id 128
                          :product-name "soy milk"
                          :date #inst "2015-12-07T17:04:51.124-00:00"
                          :previous ; and so on...  }}}]}}
```

```
{:current-user {:recent-purchases [[:purchase-id 146]
                                   [:purchase-id 145]]}
 :purchased-items {146 {:id 146
                        :product-name "organic rice"
                        :date #inst "2015-12-17T17:13:59.167-00:00"
                        :previous [:purchase-id 144]}
                   145 {:id 145
                        :product-name "soy milk"
                        :date #inst "2015-12-17T17:09:25.961-00:00"
                        :previous [:purchase-id 143]}
                   144 {:id 144
                        :product-name "organic rice"
                        :date #inst "2015-12-17T17:05:58.144-00:00"
                        :previous [:purchase-id 141]}
                   143 {:id 143
                        :product-name "soy milk"
                        :date #inst "2015-12-17T17:05:36.797-00:00"
                        :previous [:purchase-id 138]}
                   ;; and so on...  }}
```

- In Om.next components define a query to data they need

- Cursor navigates a tree, query can navigate a graph

- [{:current-user [{:recent-purchases [{:previous [:date]}]}]}]

```
{:current-user
 {:recent-purchases
  [{:previous {:date #inst "2015-12-17T17:05:58.144-00:00"}}
   {:previous {:date #inst "2015-12-17T17:05:36.797-00:00"}}]}}
```

```
{:current-user {:recent-purchases [[:purchase-id 146]
                                   [:purchase-id 145]]}
 :purchased-items {146 {:id 146
                        :product-name "organic rice"
                        :date #inst "2015-12-17T17:13:59.167-00:00"
                        :previous [:purchase-id 144]}
                   145 {:id 145
                        :product-name "soy milk"
                        :date #inst "2015-12-17T17:09:25.961-00:00"
                        :previous [:purchase-id 143]}
                   144 {:id 144
                        :product-name "organic rice"
                        :date #inst "2015-12-17T17:05:58.144-00:00"
                        :previous [:purchase-id 141]}
                   143 {:id 143
                        :product-name "soy milk"
                        :date #inst "2015-12-17T17:05:36.797-00:00"
                        :previous [:purchase-id 138]}
                   ;; and so on...  }}
```

```
{:current-user
 {:recent-purchases
  [{:previous {:date #inst "2015-12-17T17:05:58.144-00:00"}}
   {:previous {:date #inst "2015-12-17T17:05:36.797-00:00"}}]}}
```

- Om.next normalizes data* you have into one you expect

* with a little help from you

```
{:list/one [{:product-name "soy milk" :price 10}
            {:product-name "eggs" :price 22}
            {:product-name "juice"  :price 26}]
 :list/two [{:product-name "eggs" :price 22 :discount 7}
            {:product-name "apples" :price 8}
            {:product-name "cereals" :price 37 :discount 15}]}
```

```
(defui Product
      static om/Ident
      (ident [this {:keys [product-name]}]
            [:product/by-name product-name])
      static om/IQuery
      (query [this]
            '[:name :price :discount])
      Object
      (render [this]
            ;; ... elided ...))
```

```
(d{:list/one
 [[:product/by-name "soy milk"]
  [:product/by-name "eggs"]
  [:product/by-name "juice"]],
 :list/two
 [[:product/by-name "eggs"]
  [:product/by-name "apples"]
  [:product/by-name "cereals"]],
 :product/by-name
 {"soy milk" {:name "soy milk", :price 10},
  "eggs" {:name "eggs", :price 22, :discount 7},
  "juice" {:name "juice", :price 26},
  "apples" {:name "apples", :price 8},
  "cereals" {:name "cereals", :price 37, :discount 15}}
```

- Store data the way you want and let Om.next convert data to match your UI

Om.next

- Currently in alpha stage
- Production ready in a couple of months

Datascript

- Because good things come in pairs
  - Nikita Prokopov creator of Rum and Datascript

- Immutable in-memory database
  - Datalog as a query engine

- Central, uniform approach to manage all application state

- Schemaless DB < Datascript < Relational schema
  - Data already normalized

```
 {:db/id #db/id[:db.part/db]
  :db/ident :product/name
  :db/valueType :db.type/string
  :db/cardinality :db.cardinality/one
  :db/unique :db.unique/identity
  :db/doc "Product's name"}
{:db/id #db/id[:db.part/db]
  :db/ident :product/price
  :db/valueType :db.type/number
  :db/cardinality :db.cardinality/one
  :db/doc "Product's price"}
```

Datalog query

```
[:find ?name ?price ?discount
  :where
  [?p :product/name ?name]
  [?p :product/price ?price]
  [?p :product/discount ?discount]]
```

```
(defui Product
      static om/Ident
      (ident [this {:keys [product-name]}]
            [:product/by-name product-name])
      static om/IQuery
      (query [this]
            '[:name :price :discount])
      Object
      (render [this]
            ;; ... elided ...))
```

Find specific item

```
[:find ?name ?price ?discount
  :in $ ?name
  :where
  [?p :product/name ?name]
  [?p :product/price ?price]
  [?p :product/discount ?discount]]
```

```
(rum/defc product-item [db name]

  (let [item (d/q '[:find ?name ?price ?discount
                    :in $ ?name
                    :where
                    [?p :product/name ?name]
                    [?p :product/price ?price]
                    [?p :product/discount ?discount]]
                  db name)]

    (when item

      [:p name "$" (:product/price item) "discount $" (:product/discount item)])))
```

- Write components that depend on return values of queries

- Database as a value

Summary

- React.js is interesting – give it a go
- ClojureScript is interesting too
- React.js + ClojureScript = awesome
- Interesting wrappers for React.js in ClojureScript with some cool technologies

- Questions???

Konrad Szydlo
github
retailic.com

Tweet me: @ryujinkony

Retailic

- Retailic helps retail to draw business conclusions from consumer behaviour and develop corresponding software solutions.

Resources

- https://www.youtube.com/watch?v=sTSQlYX5DU0
  Eric Meijer
- https://medium.com/@carloM/reactive-programming-with-kefir-js-and-react-a0e8bb3af636#.td9mdlfzr
  Reactive programming with react.js
- http://blog.ractivejs.org/posts/whats-the-difference-between-react-and-ractive/
  React.js vs Ractive
- http://www.funnyant.com/reactjs-what-is-it/

Resources

- https://www.destroyallsoftware.com/talks/wat
- https://github.com/active-group/reacl
- https://github.com/reagent-project/reagent
- https://github.com/tonsky/rum
- https://github.com/tonsky/datascript
- https://github.com/omcljs/om/wiki