PALO IT

Back To Front with ClojureScript

Interactive digital product development, made easy^{tm}.

Francois De Serres

Head of Digital Technology | fdeserres@palo-it.com | @fdserr

Agenda

- Getting started with ClojureScript
- Interactive development / Live coding
- Modern front-end blueprints, without ceremony
- Back to front: optimising the delivery process

> *"I did meet John McCarthy of LISP fame in 1977"*
>
> - Brendan Eich - of JavaScript fame.

Prerequisites

- A will to look beyond the parens
- JDK 8
- Leiningen
- Text editor + LISP structural editing (eg. Atom + Parinfer plugin)

  THE MANAGEMENT CANNOT BE HELD RESPONSIBLE FOR THE CONSEQUENCES OF EDITING LISP CODE WITHOUT PARINFER or PAREDIT.
- rlwrap (terminal line edit) if on the Mac

Getting started

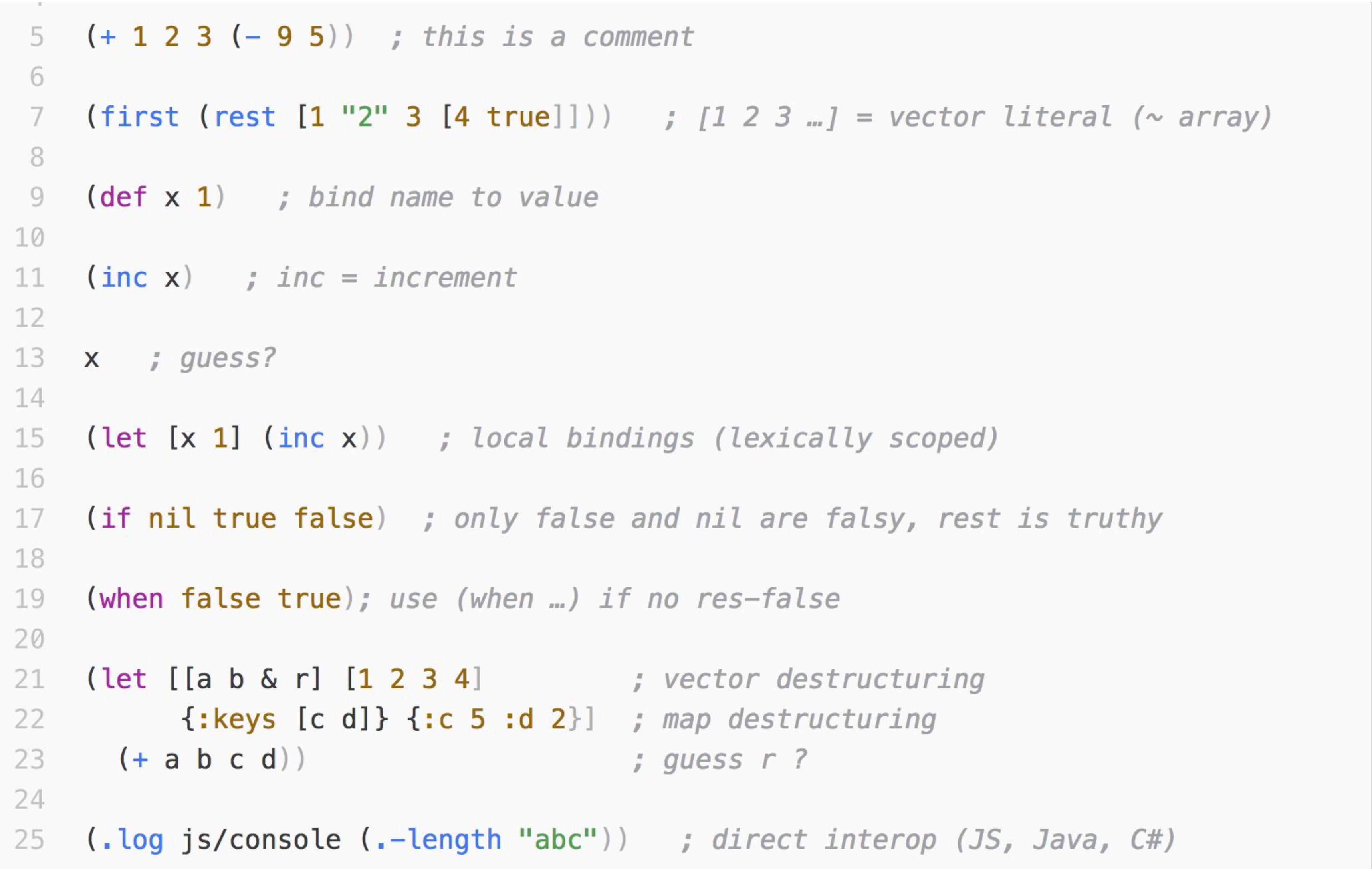
```
$ lein new devcards my-app
$ cd my-app
$ lein figwheel    # prepend with rlwrap on the Mac
```

- Browse to http://localhost:3449/cards.html
- Devcards API: http://rigsomelight.com/devcards/#!/devdemos.defcard_api

Data Types

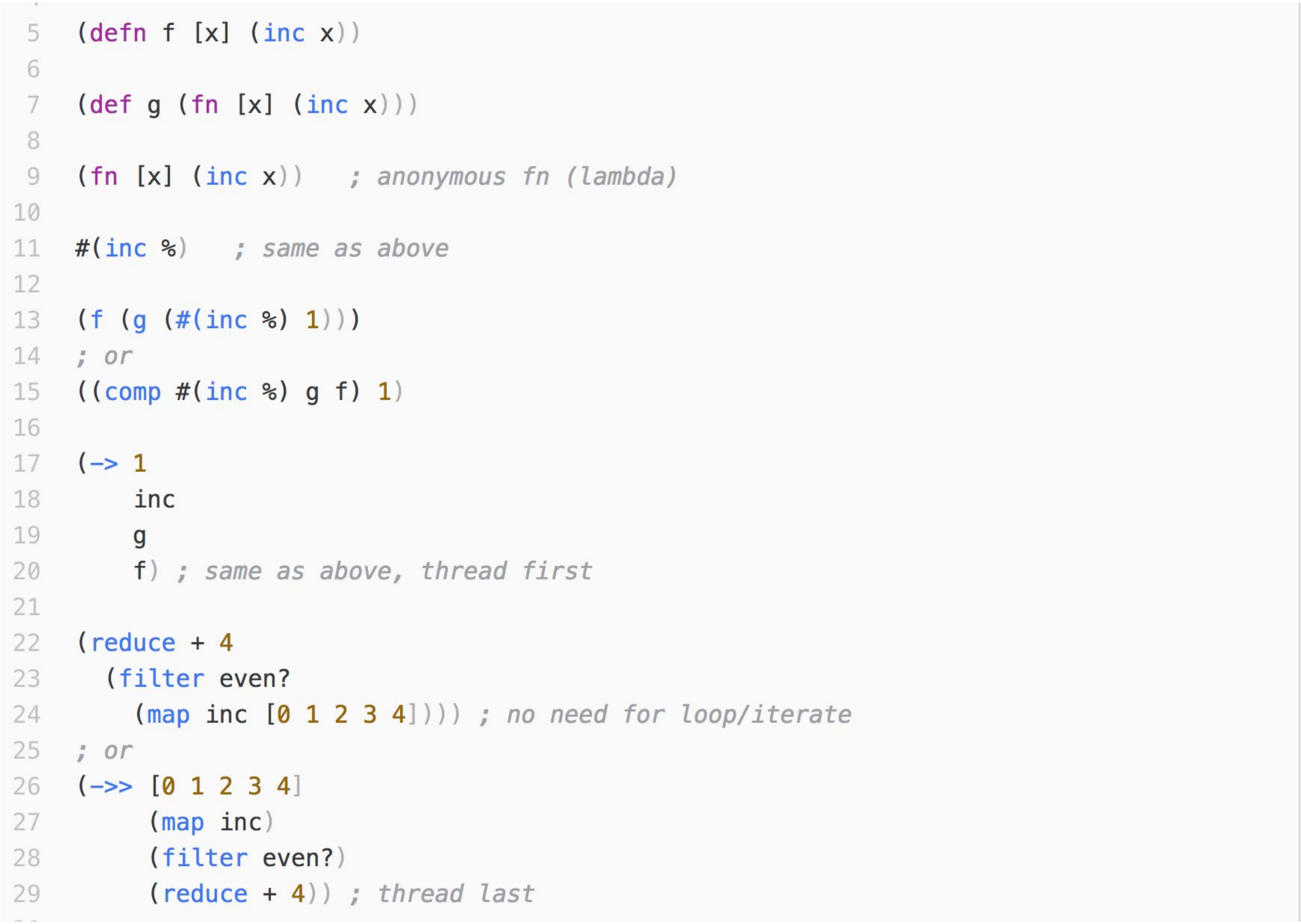
- String, Number, Boolean, Keyword (eg. **`:key`**, evaluates to itself)
- List, Vector, Map, Set: unified sequence abstraction (**`first`**, **`rest`**, **`conj`**, **`cons`**)
- Immutable values and data structures, one mutable reference type: Atom

Syntax



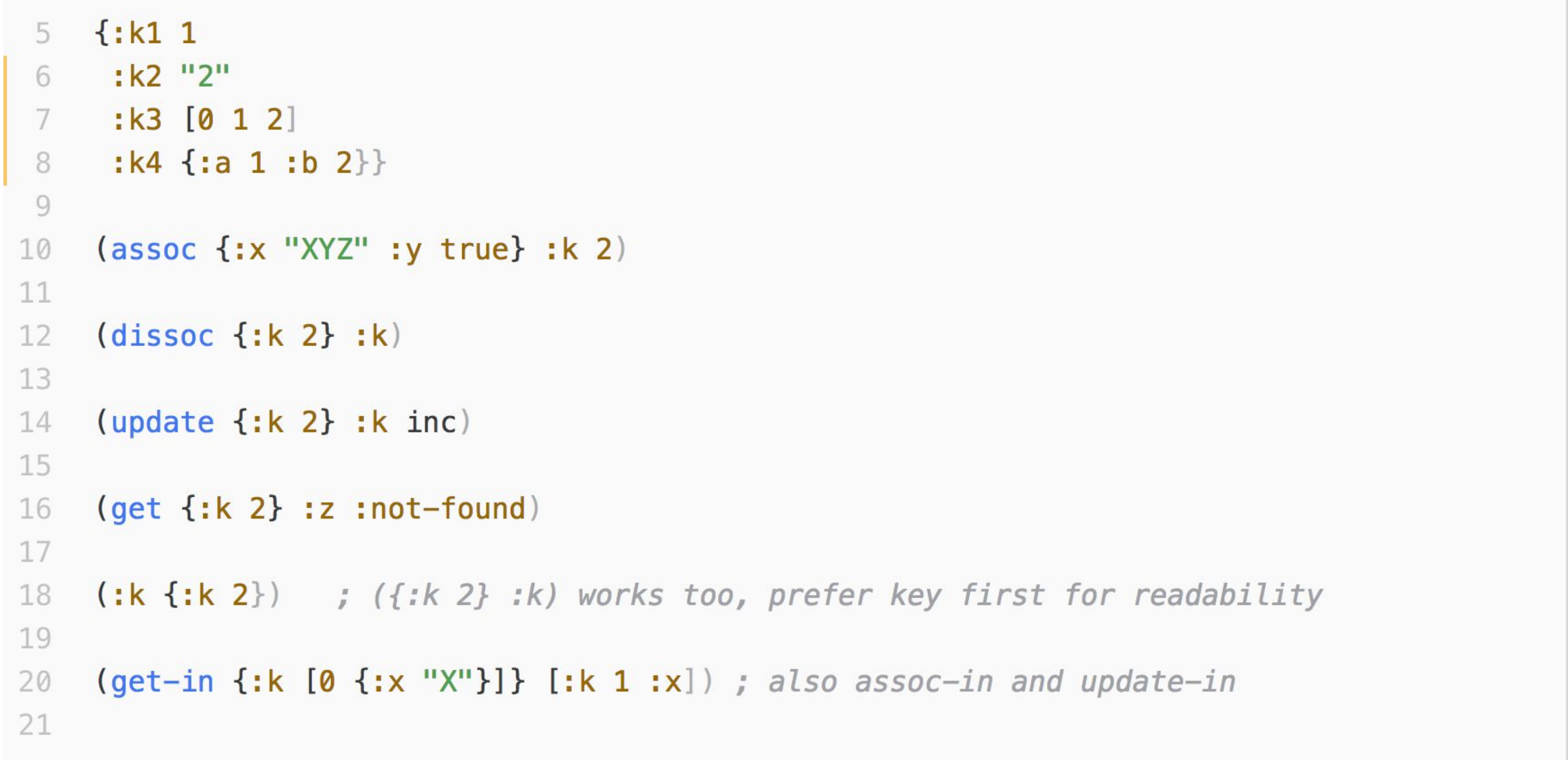
```
(+ 1 2 3 (- 9 5))  ; this is a comment

(first (rest [1 "2" 3 [4 true]]))   ; [1 2 3 …] = vector literal (~ array)

(def x 1)   ; bind name to value

(inc x)   ; inc = increment

x   ; guess?

(let [x 1] (inc x))   ; local bindings (lexically scoped)

(if nil true false)  ; only false and nil are falsy, rest is truthy

(when false true); use (when …) if no res-false

(let [[a b & r] [1 2 3 4]         ; vector destructuring
      {:keys [c d]} {:c 5 :d 2}]  ; map destructuring
  (+ a b c d))                    ; guess r ?

(.log js/console (.-length "abc"))   ; direct interop (JS, Java, C#)
```

Functions



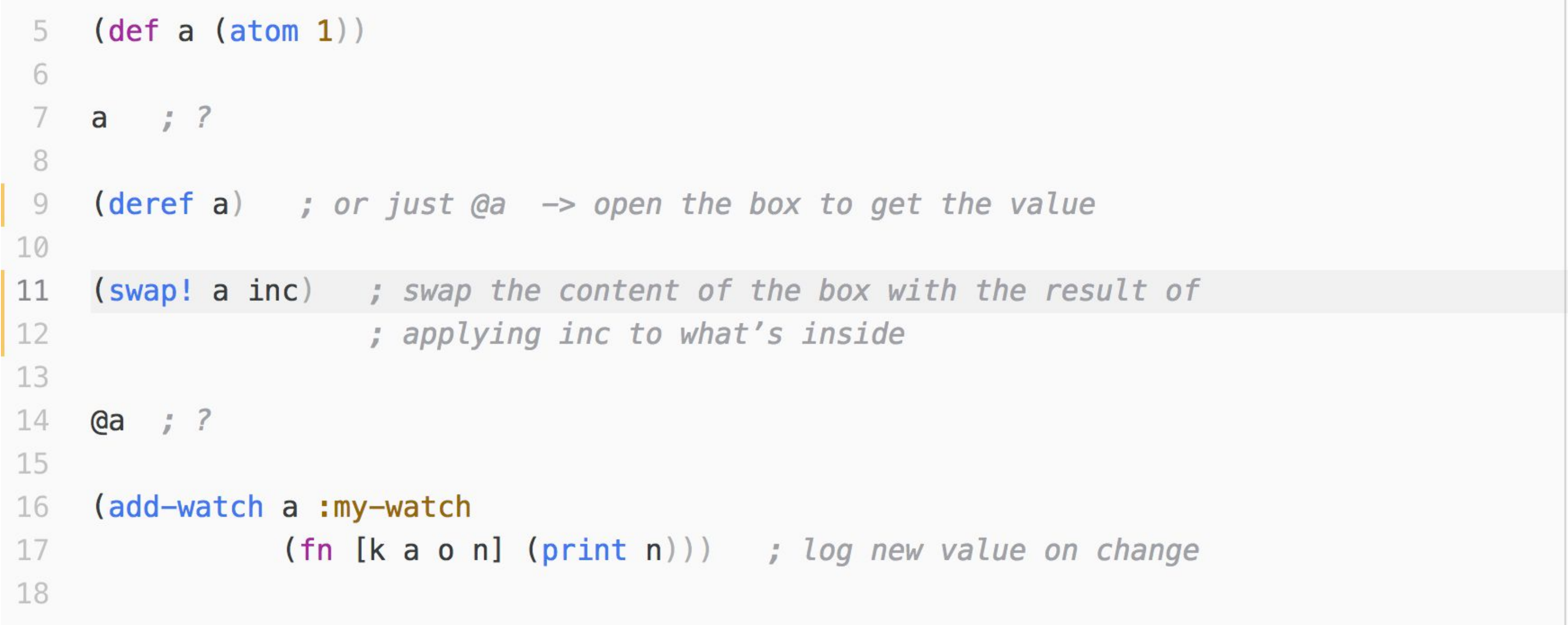
```clojure
(defn f [x] (inc x))

(def g (fn [x] (inc x)))

(fn [x] (inc x))   ; anonymous fn (lambda)

#(inc %)   ; same as above

(f (g (#(inc %) 1)))
; or
((comp #(inc %) g f) 1)

(-> 1
    inc
    g
    f) ; same as above, thread first

(reduce + 4
  (filter even?
    (map inc [0 1 2 3 4]))) ; no need for loop/iterate
; or
(->> [0 1 2 3 4]
     (map inc)
     (filter even?)
     (reduce + 4)) ; thread last
```

Maps



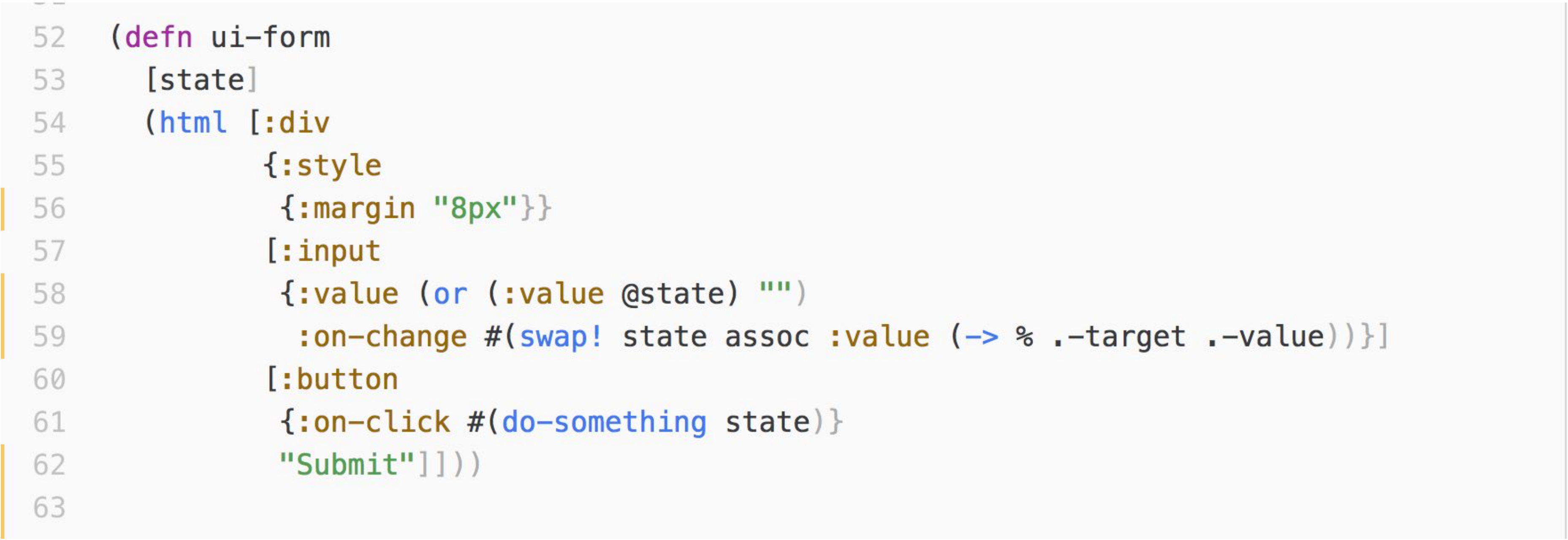
```
{:k1 1
 :k2 "2"
 :k3 [0 1 2]
 :k4 {:a 1 :b 2}}

(assoc {:x "XYZ" :y true} :k 2)

(dissoc {:k 2} :k)

(update {:k 2} :k inc)

(get {:k 2} :z :not-found)

(:k {:k 2})   ; ({:k 2} :k) works too, prefer key first for readability

(get-in {:k [0 {:x "X"}]} [:k 1 :x]) ; also assoc-in and update-in

```

Mutable State



```clojure
(def a (atom 1))

a   ; ?

(deref a)   ; or just @a  -> open the box to get the value

(swap! a inc)   ; swap the content of the box with the result of
                ; applying inc to what’s inside

@a  ; ?

(add-watch a :my-watch
           (fn [k a o n] (print n)))   ; log new value on change
```

ReactJS

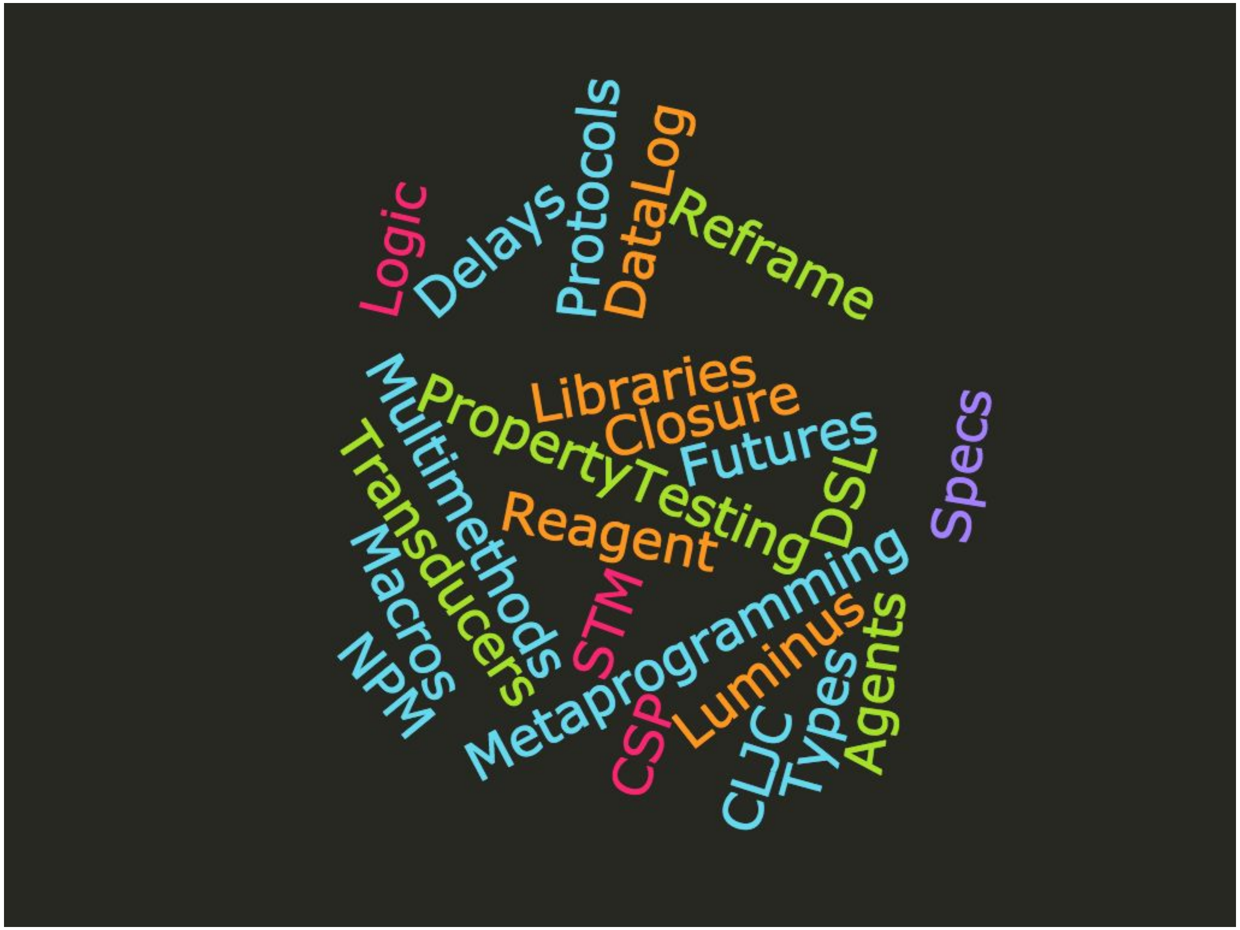


```
(defn ui-form
  [state]
  (html [:div
         {:style
          {:margin "8px"}}
         [:input
          {:value (or (:value @state) "")
           :on-change #(swap! state assoc :value (-> % .-target .-value))}]
         [:button
          {:on-click #(do-something state)}
          "Submit"]]))
```

Done!

Clojure/Script has much more under the hood,

that **you don't need to know about** to get started.



Show some code already!

Reloaded

- Figwheel: hot code reload that just works (immutability helps a lot)
- Devcards: focus, experiment (with state history), test, document
- Not just for UI, great for applications with complex state and transitions
- REPL into the live app (debugger++)

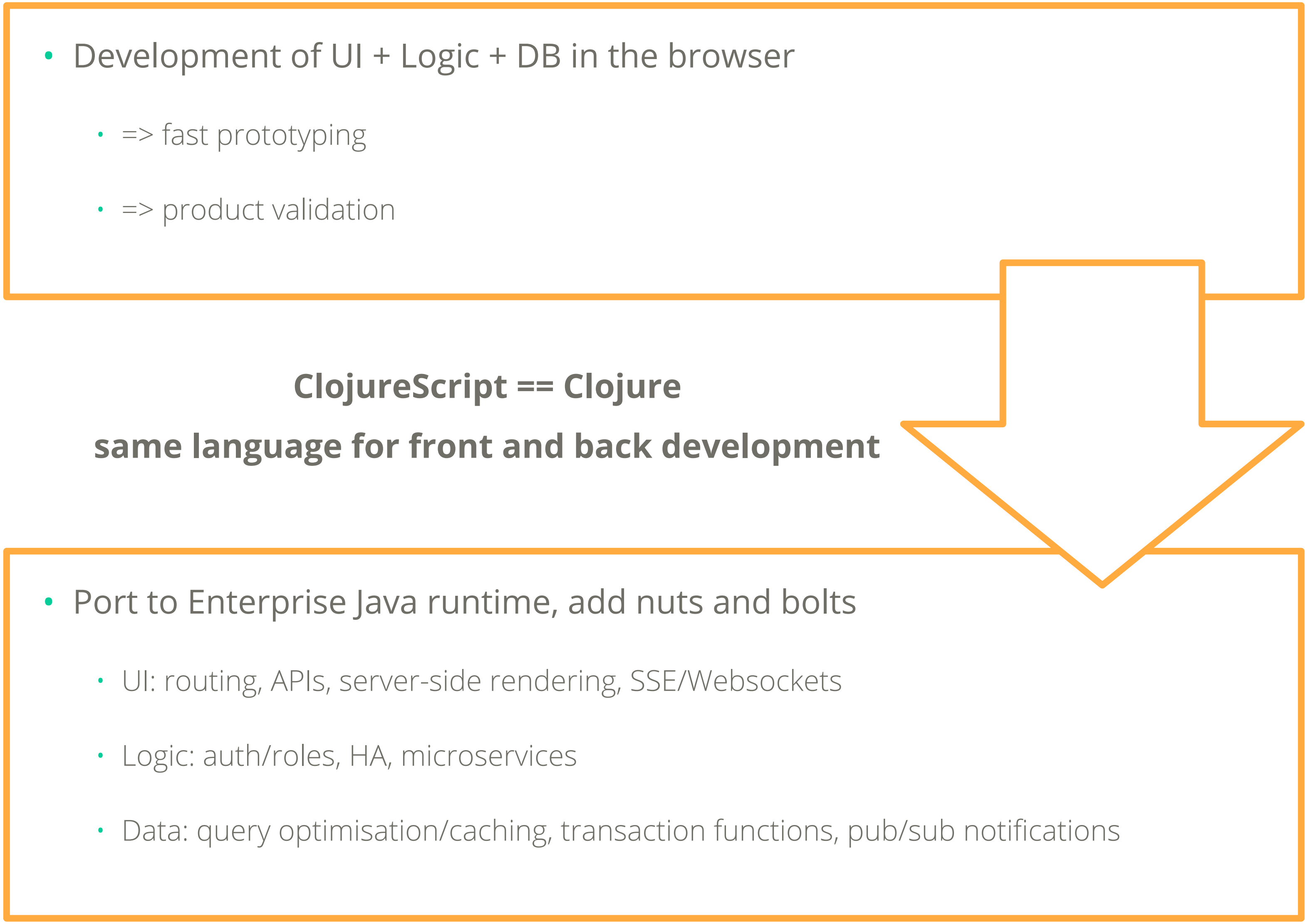
Blueprints

$$S \xrightarrow{r} UI$$

$$S \xrightarrow{t} S'$$

- Reactive UI
  - ClojureScript <3<3<3 ReactJS
- Normalised store
  - Immutable, persistent data structures
  - Functional transforms / queries, atomic transactions by default
  - Optional: DataScript in-browser database
- Flux
  - UI -> dispatch -> store -> UI -> ...

Back to Front



- Development of UI + Logic + DB in the browser
  - => fast prototyping
  - => product validation

ClojureScript == Clojure

same language for front and back development

- Port to Enterprise Java runtime, add nuts and bolts
  - UI: routing, APIs, server-side rendering, SSE/Websockets
  - Logic: auth/roles, HA, microservices
  - Data: query optimisation/caching, transaction functions, pub/sub notifications

Next…

- The Onyx platform
  - Event streaming, lambda architecture , CQRS, real-time data processing, ETL, …
  - Spark, Storm, Flink, Map/reduce, …
  - Provides a compatible, single process, ClojureScript runtime
- => Fast prototyping of complex distributed computations in the browser and/or on NodeJS (live coding applies)
- => Deploy and run the same code (workflows and jobs) on the cluster (Docker, Kubernetes, Mesos/Marathon, …)

Conclusion

- ClojureScript is simple and easy to learn, we get newbies up and running in a couple days.
- Modern blueprints are included, no libraries or framework needed, no boilerplate.
- Robust hot code reloading with immutable state really is a game changer.
- Fast, iterative prototyping empowers a leaner, faster product design and delivery process.

Learn, explore, build, test and validate in the browser with immediate, interactive feedback,
then develop/port back-end db model, queries, transaction, business logic, APIs, ...

Thank you!

- Repo: https://github.com/fdserr/clarc
- Leiningen: https://leiningen.org
- Parinfer: https://shaunlebron.github.io/parinfer/
- ClojureScript cheat sheet: http://cljs.info/cheatsheet/
- Devcards: https://github.com/bhauman/devcards
- DataScript (links to Datomic): https://github.com/tonsky/datascript
- The Onyx Platform: http://www.onyxplatform.org
- Applicative State Transition systems - John Backus 1977 Turing Award Lecture: http://wwwusers.di.uniroma1.it/~lpara/LETTURE/backus.pdf

Let’s keep in touch!

Francois De Serres

Head of Digital Technology,
PALO IT

@fdserr

Francois De Serres

fdserr

@fdserr