U0E0XL064 : oh wait... it's not the same.
U0E0XL064 : concat gives that stackoverflowerror...
U051SA920 : <@U0E0XL064> `(into [] xf m)`
U0E0XL064 : right, thx.
U0539NJF7 : also: <https://stuartsierra.com/2015/04/26/clojure-donts-concat>
U0E0XL064 : funny... I read that last week. stupid me :stuck_out_tongue:
U0539NJF7 : :slightly_smiling_face:
U0E0XL064 : well, actually it's `(flatten (into [] xf m)` I was after.
U050487DQ : <@U0E0XL064> or second `map` should be `mapcat` :slightly_smiling_face:
U0E0XL064 : nice.
U051HUZLD : is there something builtin or better for this?```
```
(defn least-common-ancestor [path1 path2]
  (let [i (min (count path1) (count path2))]
    (reduce
      (fn [p idx]
        (let [v1 (nth path1 idx)
              v2 (nth path2 idx)]
          (if (= v1 v2)
            (conj p v1)
            (reduced p))))
      [], (range i))))

(least-common-ancestor
  [:a :b :c]
  [:a :b :d])
=&gt; [:a :b]
```

U051SA920 : <@U051HUZLD> `(take-while some? (map (fn [a b] (when (= a b) a)) [0 1 2] [0 1 3]))`
U3HKE2SLW : This one has much better runtime performance, mostly stemming from the omission of `nth`
U051HUZLD : :bellissimo:
U051SA920 : Well you could avoid one `nth` call here if you use `reduce-kv` and you can guarantee it's a vec
U051HUZLD : not worth it. on the other hand, count being &gt; 10 is highly unlikely (in my case).
U051SA920 : Yeah then the `map` version will be quicker. Reduce is often slower for smallish input
U051SA920 : And if you want to avoid the intermediate sequence: `(sequence (comp (map #(when (= %1 %2) %1) ) (take-while some?))   [0 1 2] [0 1 3])`
U051HUZLD : <@U051SA920> on such small inputs it is not worth it:```
```
(time  (dotimes [n 1000]    (take-while some? (map (fn [a b] (when (= a b) a)) [0 1 2] [0 1 3]))))
"Elapsed time: 1.166446 msecs"
(time  (dotimes [n 1000]    (sequence (comp (map #(when (= %1 %2) %1) ) (take-while some?))   [0 1 2] [0 1 3])))
"Elapsed time: 7.258858 msecs"
```

U051SA920 : <@U051HUZLD> Don't forget a `doall` :slightly_smiling_face:
U051HUZLD : ```(time (dotimes [n 1000]   (doall (take-while"Elapsed time: 3.356031 msecs"```

U051HUZLD : :opieop:
U051HUZLD : omg what am I doing right now?
U051SS2EU : if you want to measure calculation time (or run for side effects) and don't need the result, use dorun instead of doall
U051SS2EU : also, sequence is not typically more efficient than regular lazy ops in most cases iirc
U051SS2EU : I'd start with the general concept of event sourcing, which is more general and more usable than CQRS
U5ZAJ15P0 : <@U051SS2EU> right, sorry, I think my question was misphrased. Right now I am more concerned about how to structure my code at the application level (how to organise functions and manage side effects)
U051SS2EU : event sourcing is a strategy for this