Transducers and core.async

Transducers in Practice Workshop - CUFP 2017

core.async

- Core.async is a library (not shipped with Clojure)
- It nicely complement already existent concurrency models
- It is based on Hoare's Communicating Sequential Processes (CSP) model
- Go-loops avoid blocking by parking threads in the background.
- One of the main driver for transducers was to re-use the same sequential processing on top of CSP “channels”

Why core.async and transducers?

- Items flowing through channels are similar to items in a collection
- On entering/exiting the channel they can be transformed, aggregated, etc.
- No need to a special “async/map” implementation
- Just reuse the same sequential processing on channels

core.async essentials for the lab

- `(def in (chan 1))` creates a channel with buffer size = 1
- **(def out (chan 1 xform))** also takes optional xform
- `(close! in)` claims back resources and stops running loops
- >!   Writes into a channel   <!   Reads from a channel

Resources

- Communicating Sequential Processes (CSP) paper
- Core.async walk-through
- Extended guide

Lab 03
Transducers and core.async

Goal of Lab3

- Task 1: create channels and orchestrate them so incoming products end up in a local cache
- Task 2: use the “prepare-data” transducer from lab01 to transform incoming products.

Open `transducers-workshop.lab03` to start.