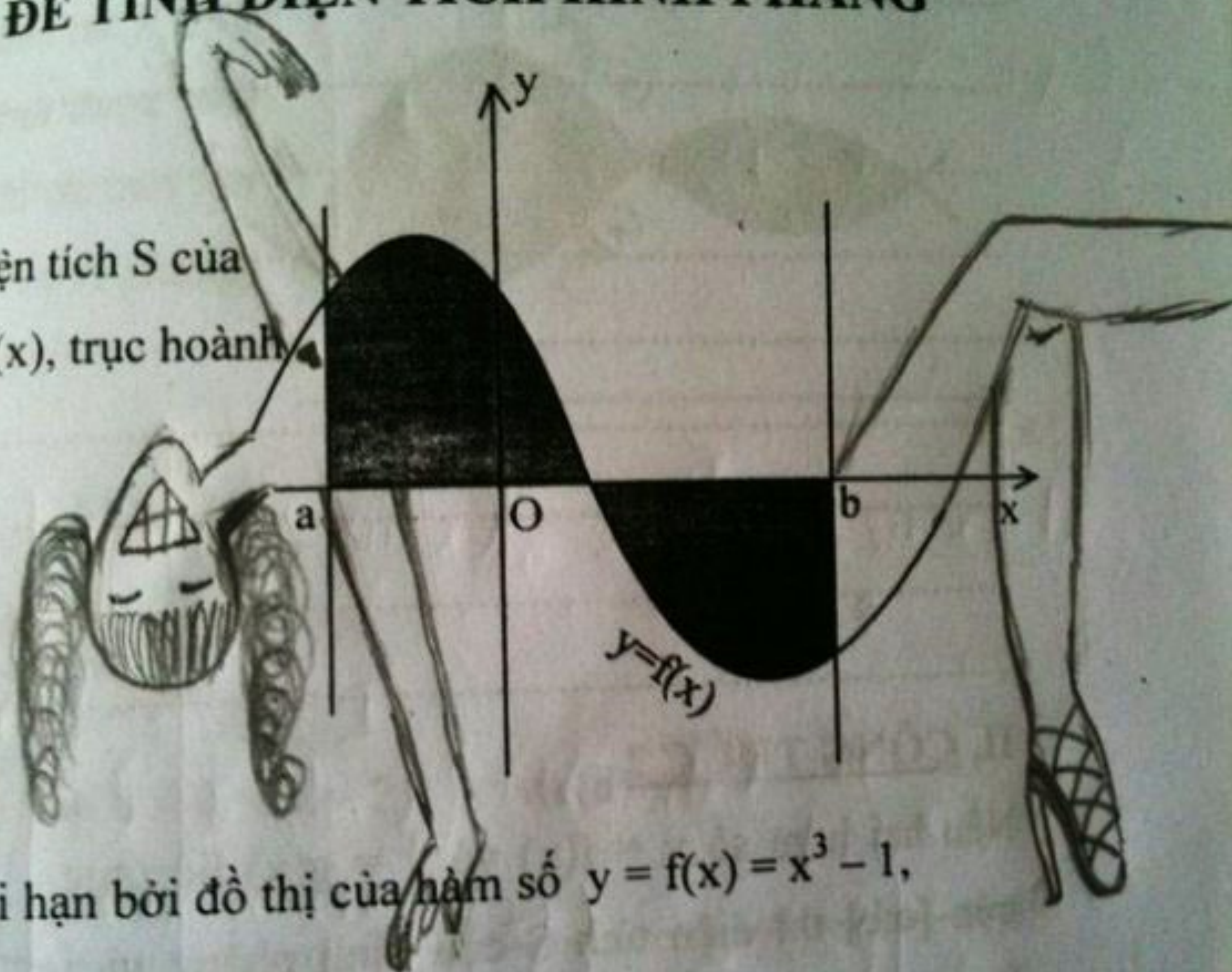


H PHÂN ĐỂ TÍNH DIỆN TÍCH HÌNH PHẪNG

[a;b] thì diện tích S của
n số $y = f(x)$, trục hoành



hằng giới hạn bởi đồ thị của hàm số $y = f(x) = x^3 - 1$,

hoành.

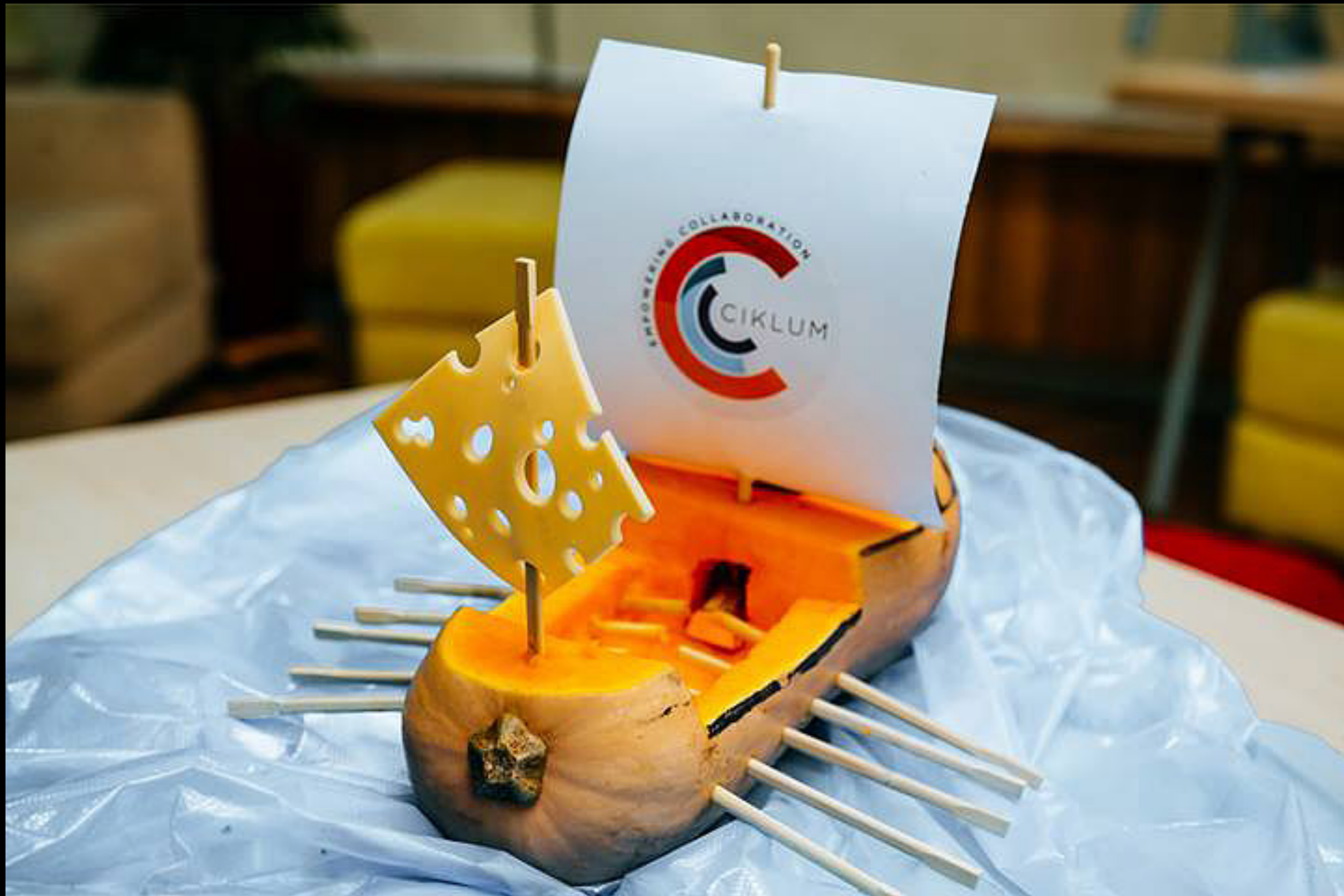


Clojure-Gopher
CSP with idiomatic Clojure



Александр Хохлов

@apoint



STICKY NOTES FOR YOUR CODE

with context and timeline

Unlimited repositories & collaborators
for open source and public projects



NOTIFY ME WHEN LAUNCHED



STICKY NOTES FOR YOUR CODE

with context and timeline

Unlimited repositories & collaborators
for open source and public projects



facebook.com/notes.io

notes.io

@notes_io

NOTIFY ME WHEN LAUNCHED

**Что ещё за
Clojure?**

Lisp для JVM

(и не только)

Handwritten musical notation on page 492, consisting of approximately 28 staves of music. The notation includes various rhythmic values and melodic lines. A notable feature is a section of the music on the lower half of the page where the notes are written as a series of left-facing parentheses '(((((' instead of the standard note heads.

Handwritten musical notation on page 493, consisting of approximately 28 staves of music. The notation includes various rhythmic values and melodic lines, continuing from the previous page.

Excerpt from a typical LISP program:

)))))))))))))))

Excerpt from a typical Node.js program:

}); }); }); }); }); });

Язык общего назначения

Функциональный

Со строгой динамической

типизацией

Прагматичный

Компактный синтаксис

Полиморфизм без ООП

Кодогенерация / макросы

**Иммутабельные персистентные
структуры данных**

Interop с Java/JavaScript

REPL-driven development



```
(defn blank? [s]
  (every? #(Character/isWhitespace %) s))
```



```
(defn blank? [s]
  (every? #(Character/isspace %) s))
```



```
(defn blank? [s]
  (every? #(Character/isWhitespace %) s))
```




```
(defn blank? [s]
  (every? #(Character/isspace %) s))
```



```
(defn blank? [s]
  (every? #(Character/isWhitespace %) s))
```



```
(defn blank? [s]
  (every? #(Character/isspace %) s))
```



(blank? “Non empty string”)

(blank? “ ”)

[1 2 3]

(1 2 3)

{:a 1 :b 2}

(def set #{:x :y :z})

'(1 2 3 4) "Look Ma!
no commas"

(- 8 2 2)

("one" "two" "three") (+ 1 1)

[:foo :bar]

{ :key1 "one" :key2 2 }



2016-01-17

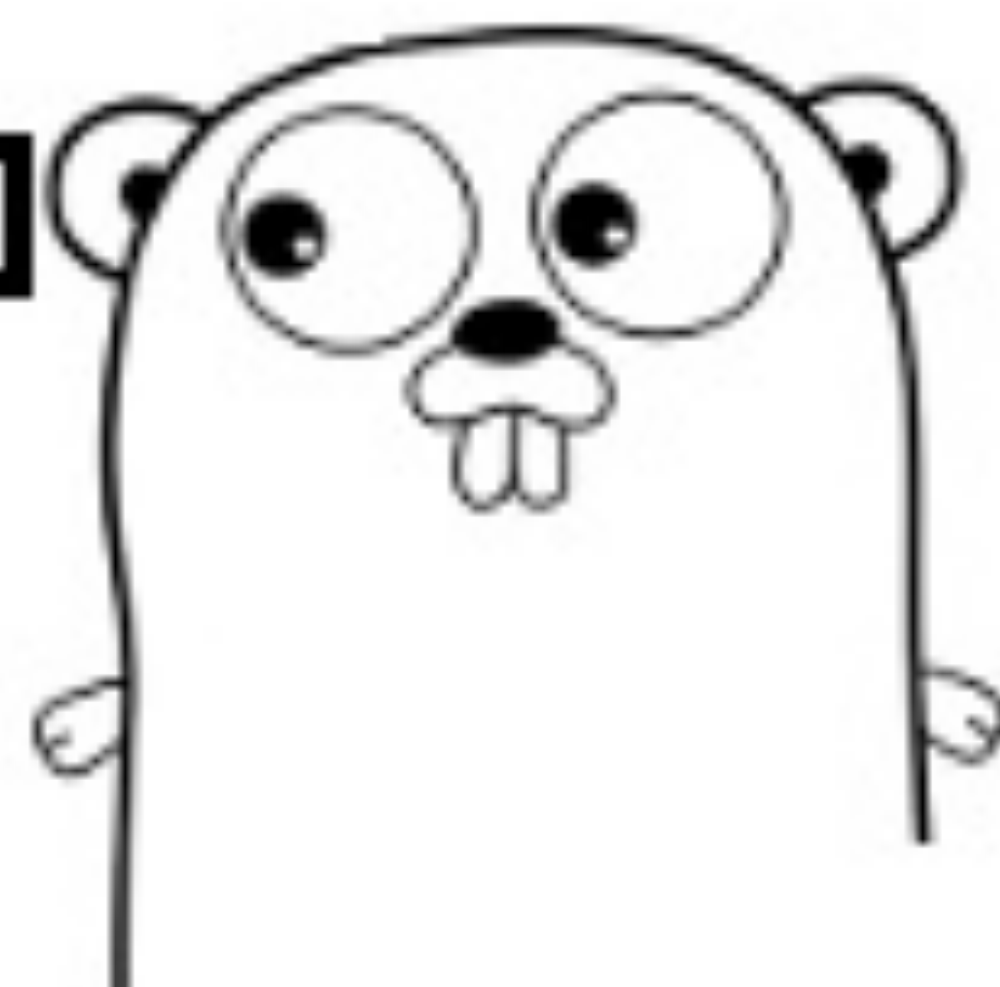


core.async

Channel (pipe)



Goroutine 1



Goroutine 2


```
(ns meetup.core
  (:require [clojure.core.async :refer :all :as async]))
```

```
(ns meetup.core  
  (:require [clojure.core.async :refer :all :as async]))
```

```
(def c (async/chan))
```

```
(ns meetup.core
  (:require [clojure.core.async :refer :all :as async]))
```

```
(def c (chan))
```

```
(def c2 (chan 10))
```

```
(ns meetup.core
  (:require [clojure.core.async :refer :all :as async]))
```

```
(def c (chan))
```

```
(def c2 (chan 10))
```

```
(def c3 (chan (dropping-buffer 1))) ;dropping newest
```

```
(ns meetup.core
  (:require [clojure.core.async :refer :all :as async]))

(def c (chan))
(def c2 (chan 10))
(def c3 (chan (dropping-buffer 1))) ;dropping newest
(def c3 (chan (sliding-buffer 1))) ;dropping oldest
```

```
(ns meetup.core
  (:require [clojure.core.async :refer :all :as async]))

(def c (chan))

(take! c #(println %))
(put! c 42)
```

```
(ns meetup.core
  (:require [clojure.core.async :refer :all :as async]))
```

```
(def c (chan 1))
```

```
(<!! c)
```

```
(>!! c 42)
```

```
;; used for thread coordination mainly
```

```
(ns meetup.core
  (:require [clojure.core.async :refer :all :as async]))
```

```
(def c (chan 1))
```

```
(thread (println (<!! c)))
```

```
(>!! c 42)
```

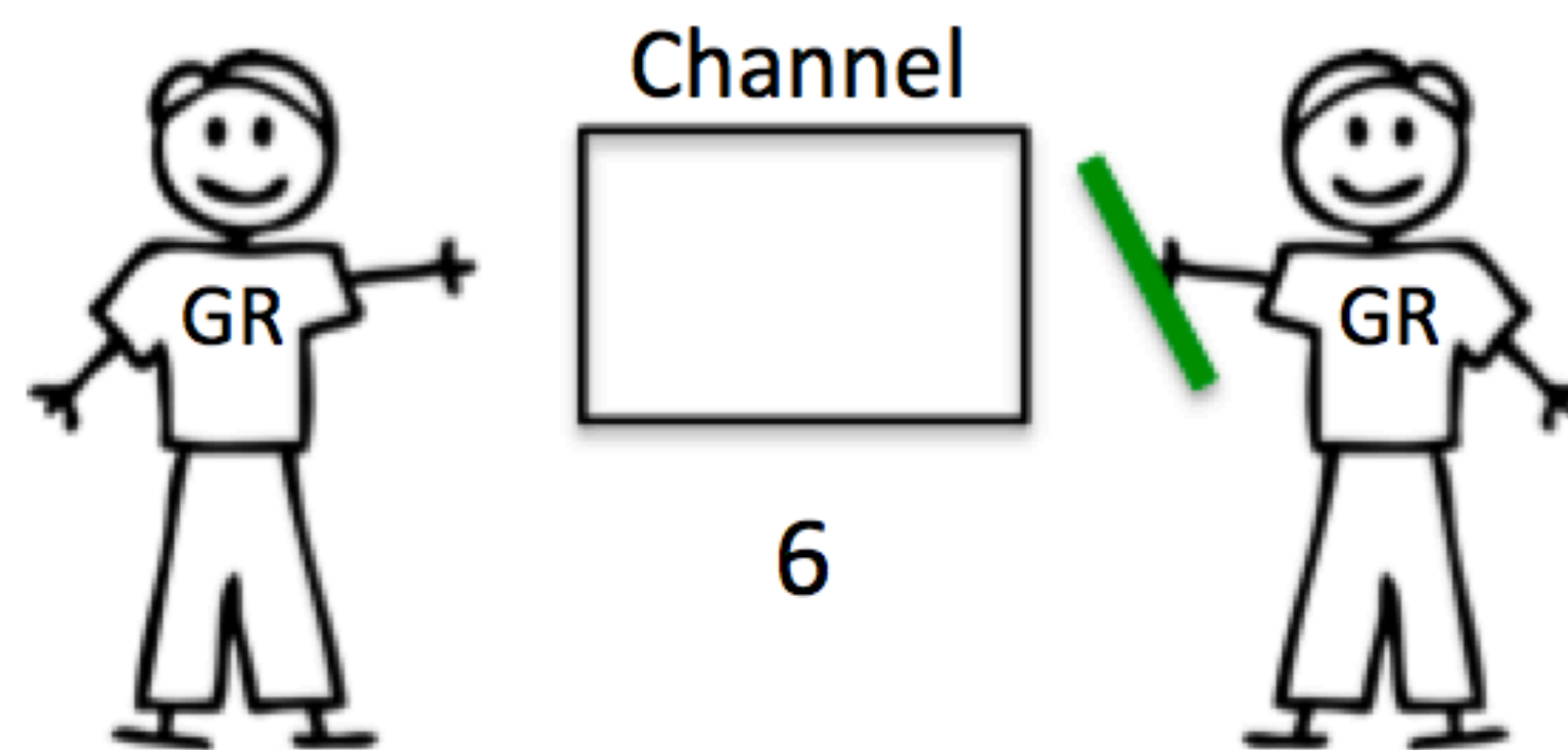
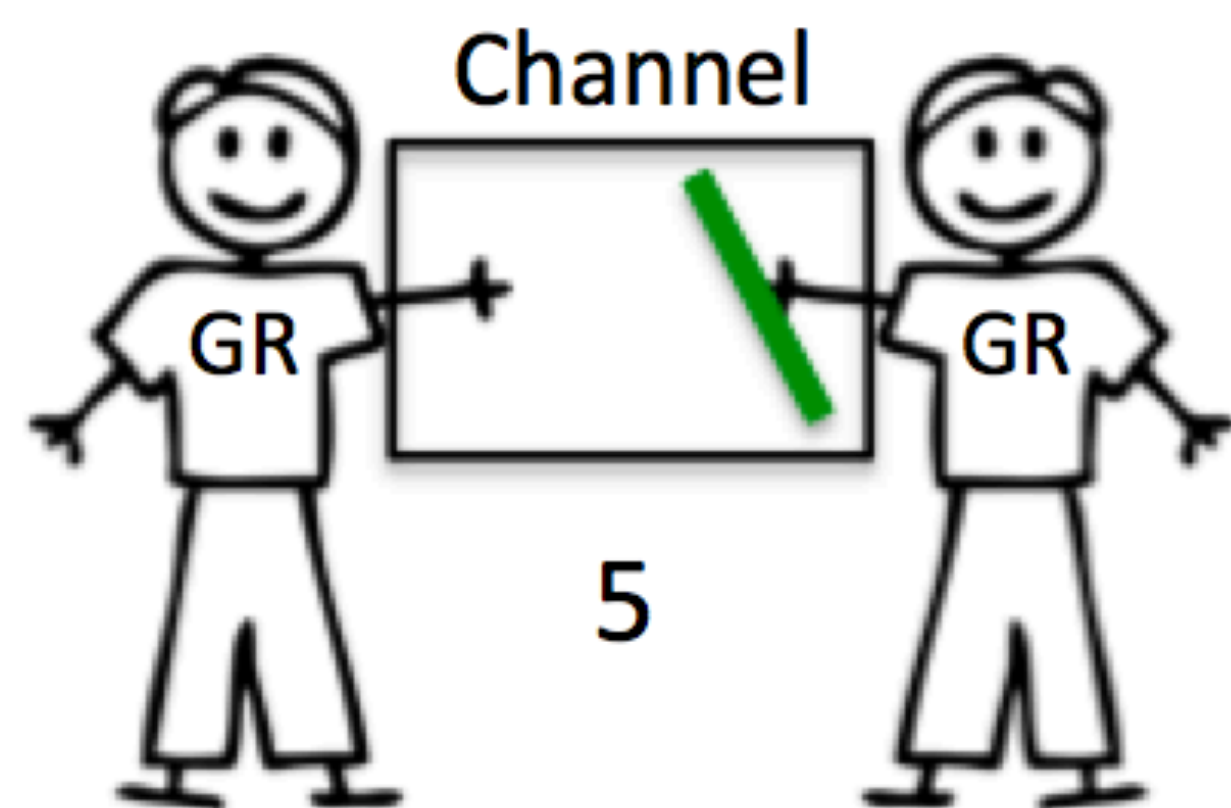
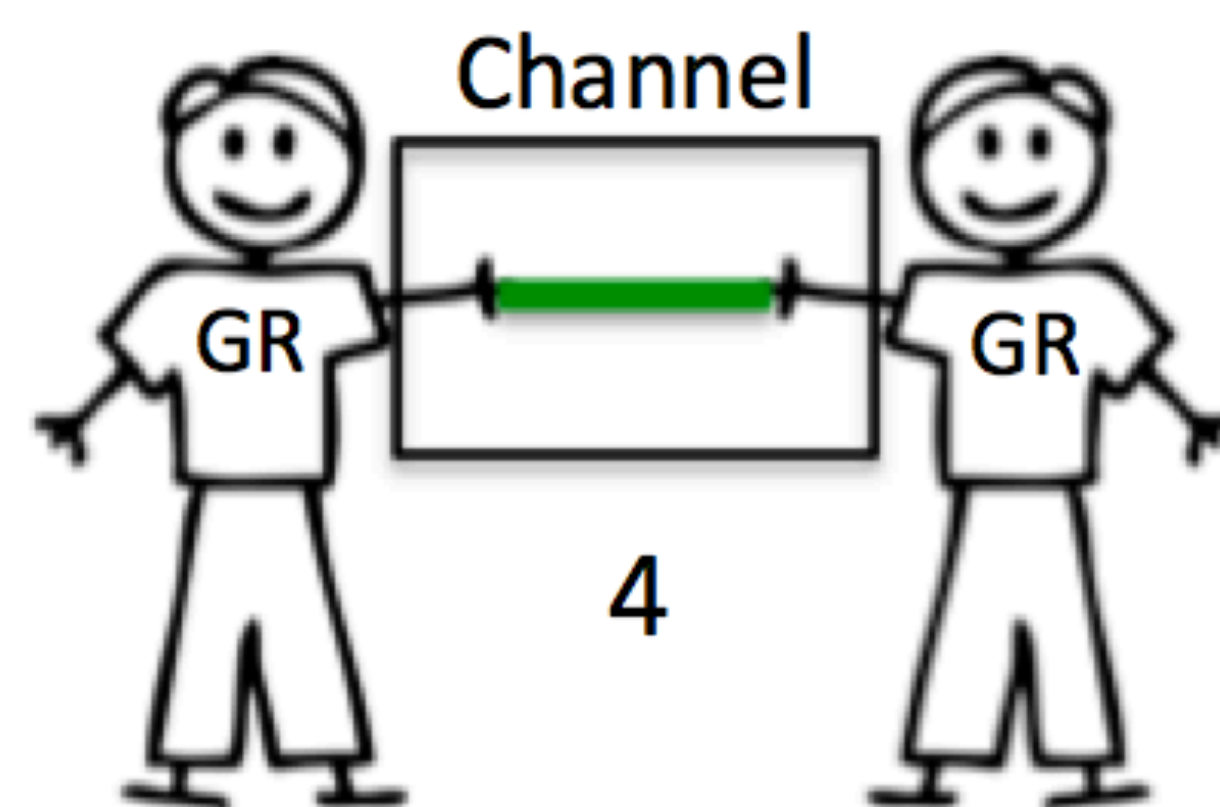
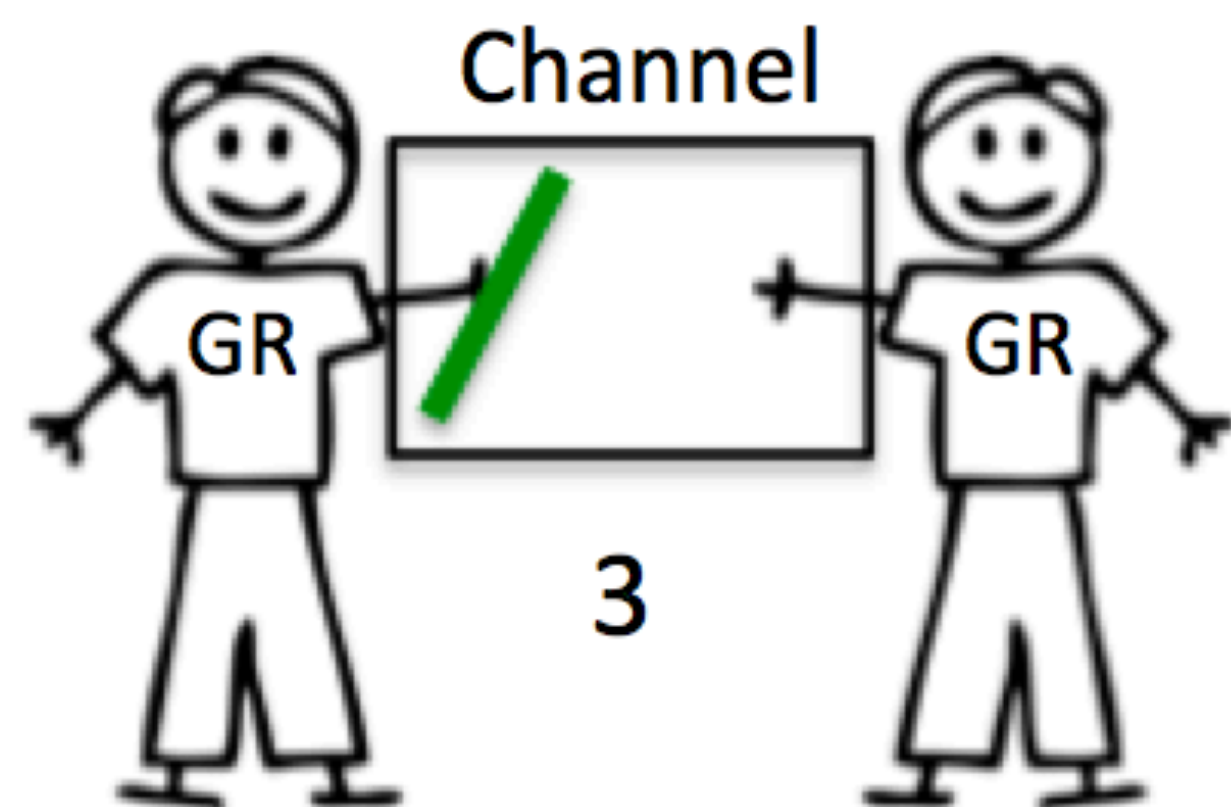
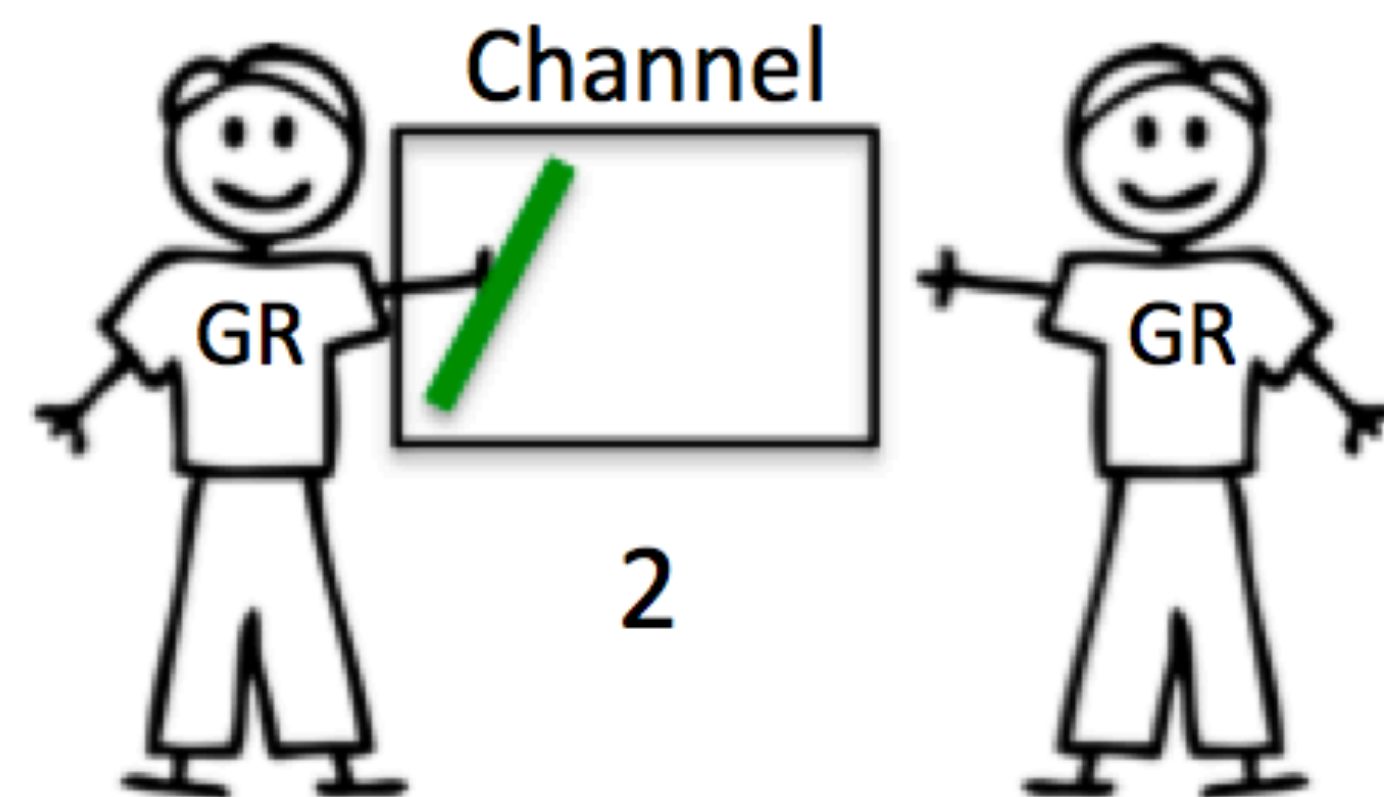
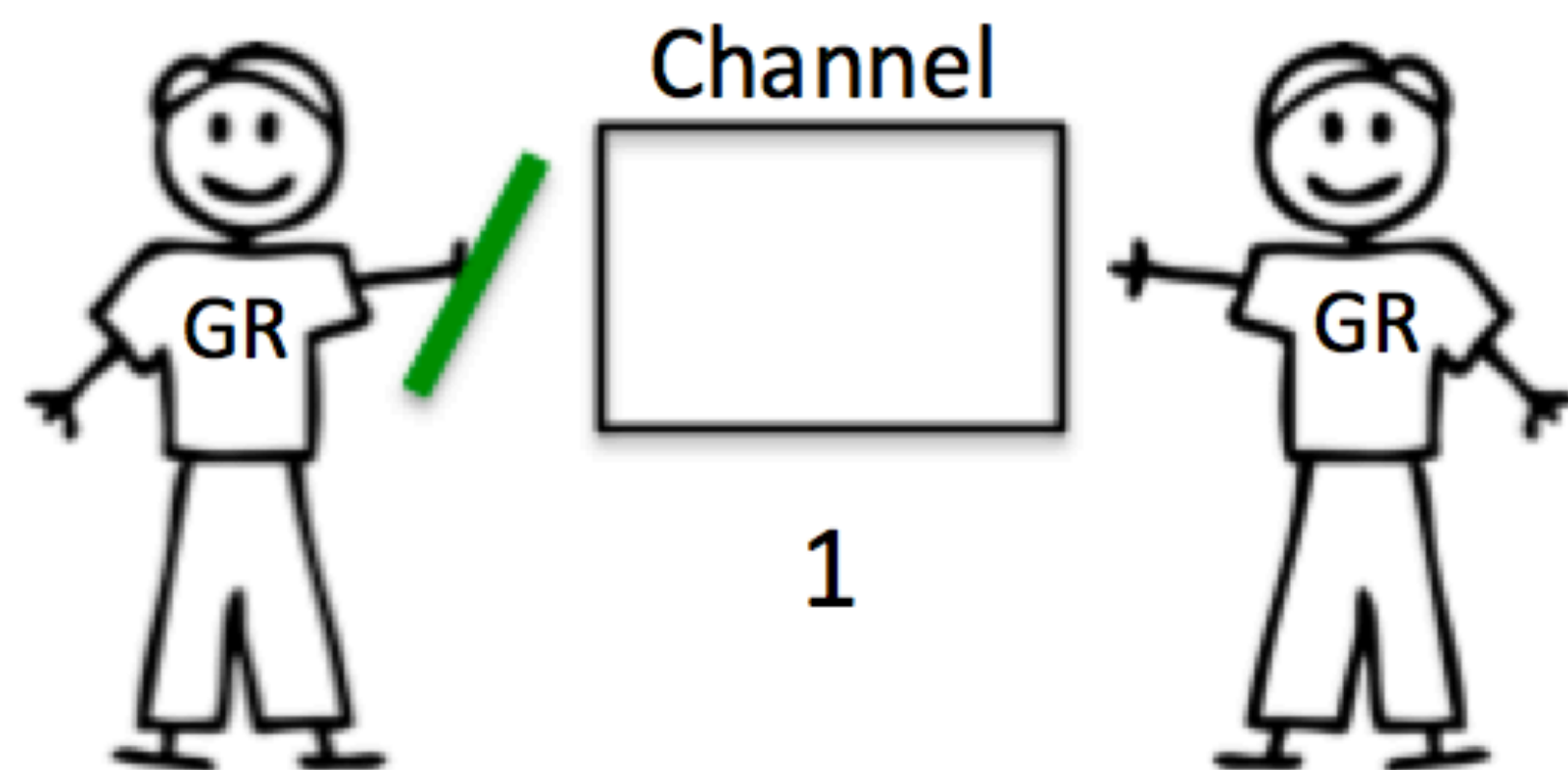
```
;; used for thread coordination mainly
```



```
(ns meetup.core
  (:require [clojure.core.async :refer :all :as async]))

(def c (chan))

(go (println (<! c)))
(go (>! c "Hello"))
```



```
(ns meetup.core
  (:require [clojure.core.async :refer :all :as async]))

(def a (chan))
(def b (chan))

(go (alts! [a b])) ;; returns [value chan]
```

```
(ns meetup.core
  (:require [clojure.core.async :refer :all :as async]))
```

```
(def a (chan))
```

```
(def b (chan))
```

```
(go (alts! [a b (timeout 1000)]))
```

```
;; returns [value chan] or closes after 1s
```

pub/sub

Transducers

mix/merge

Pipelines



The page at localhost:3000 says:

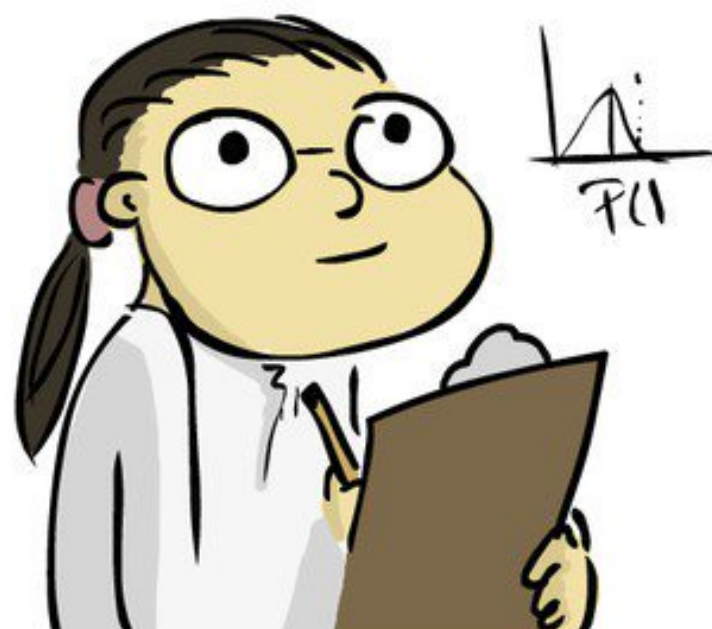
Helloooooooooo ClojureScript

OK

PYTHON



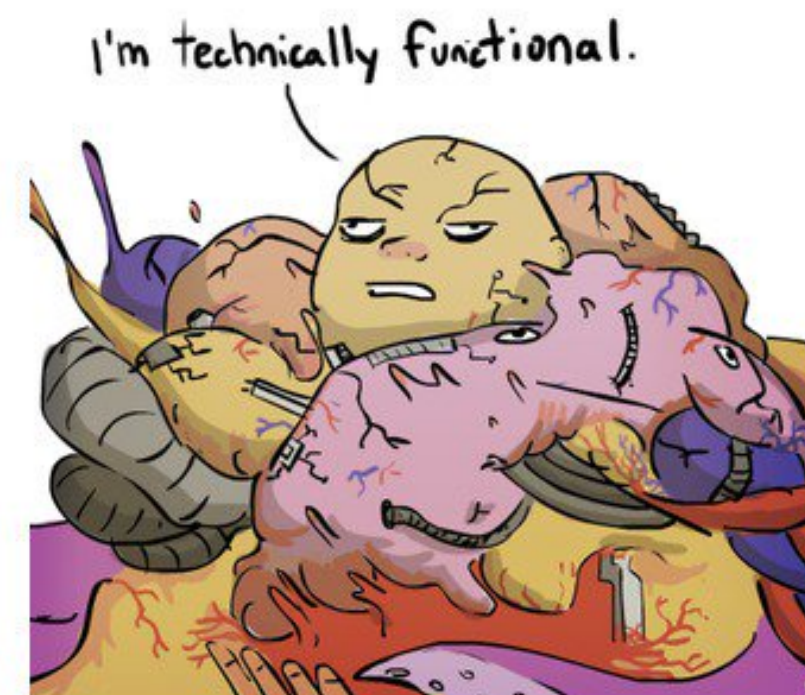
R



JAVA



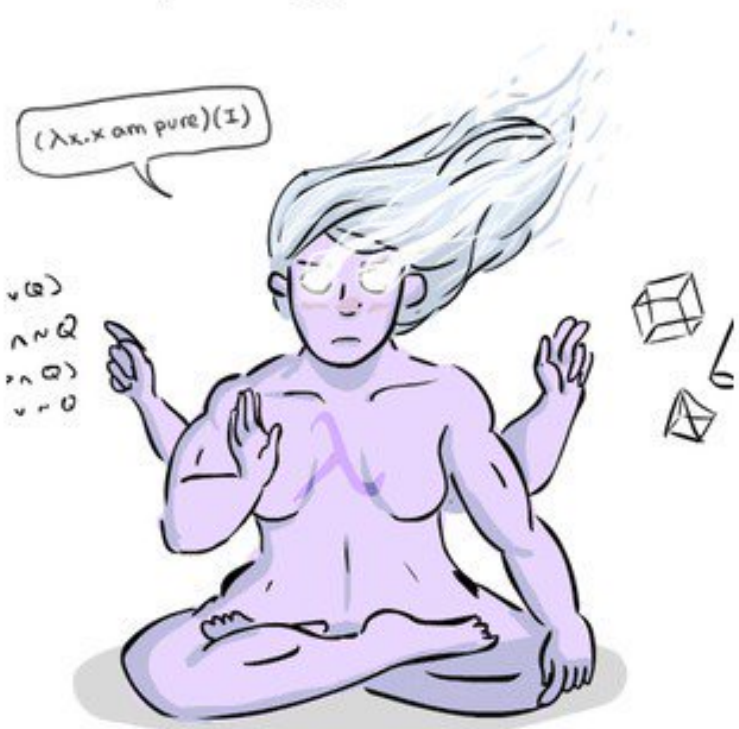
JAVASCRIPT



PHP



HASKELL



PERL



C



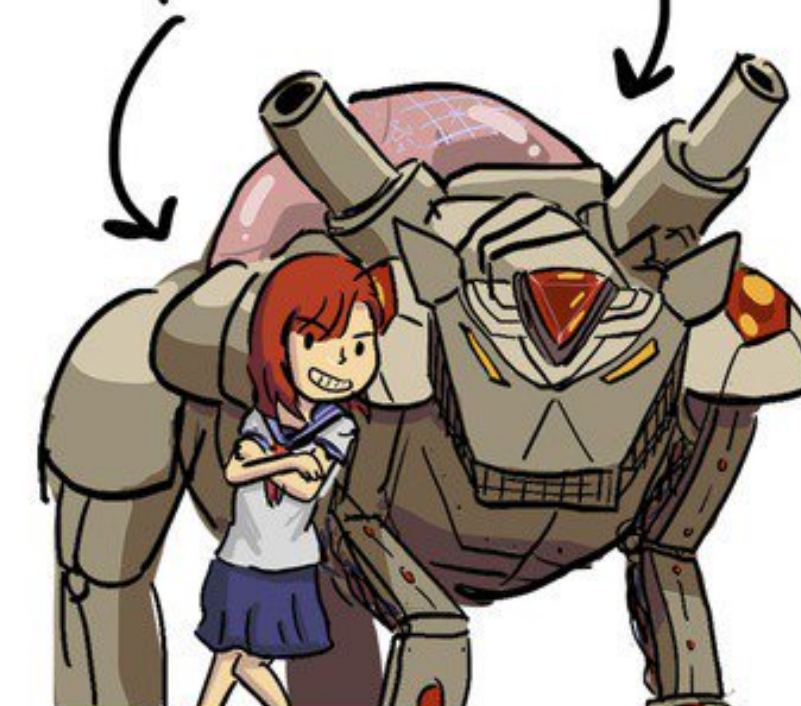
C#



C++



RUBY ON RAILS



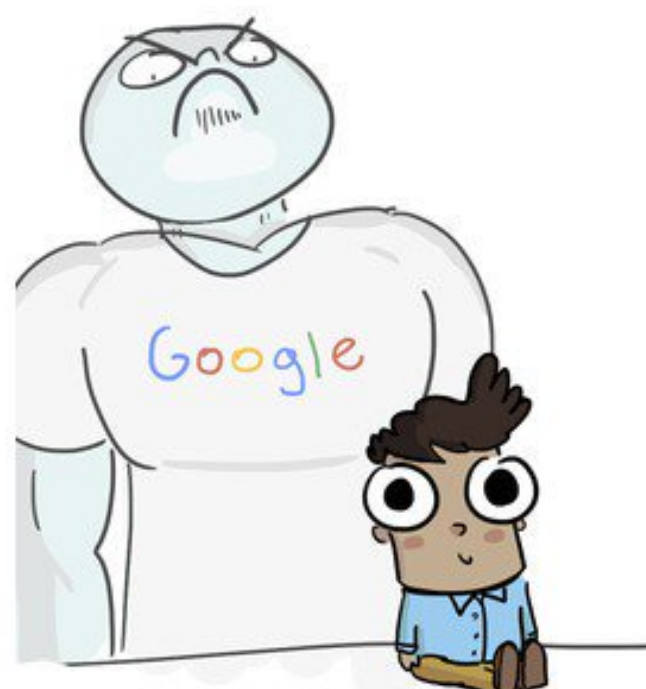
ASSEMBLY



ERLANG



GO



BRAINFUCK

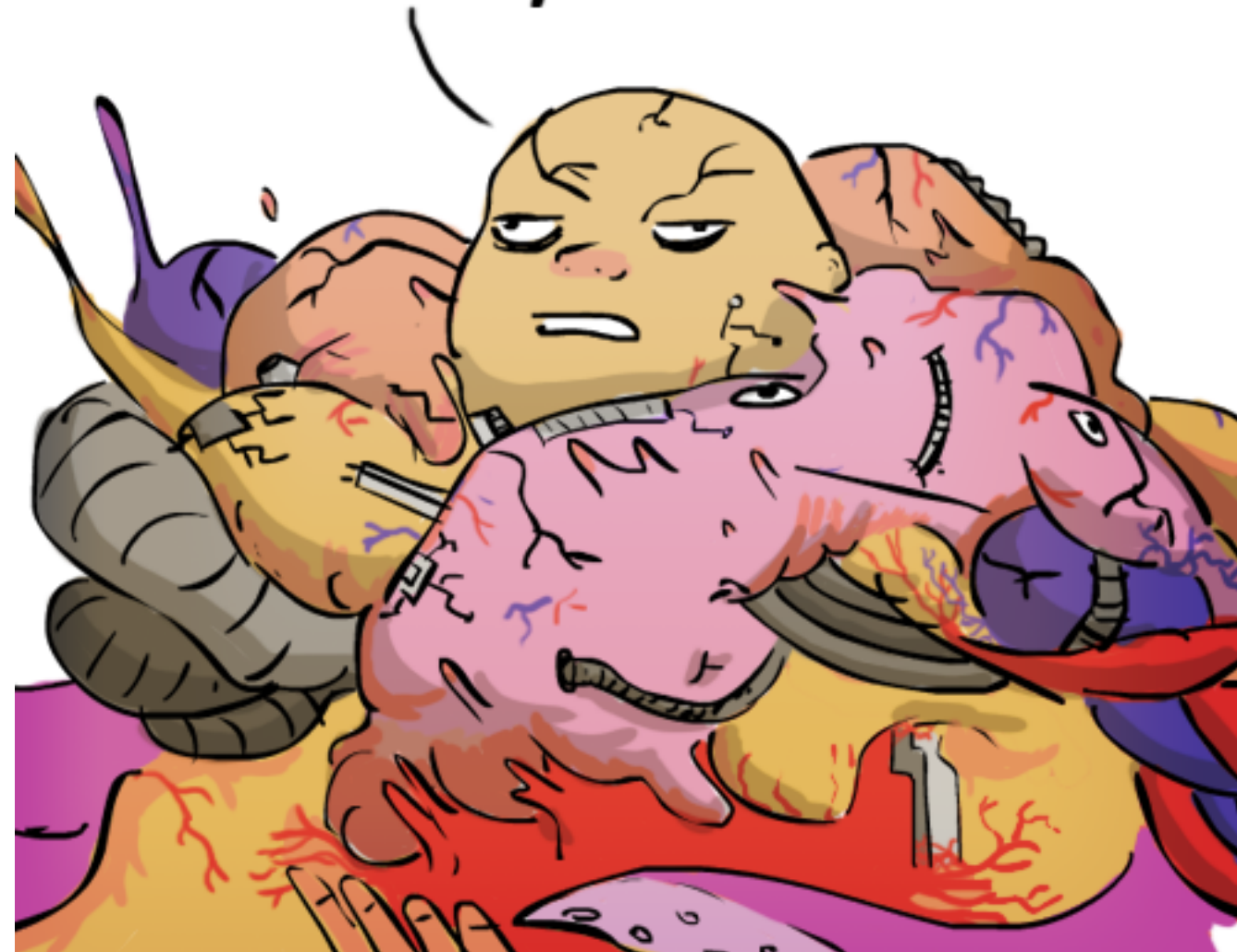


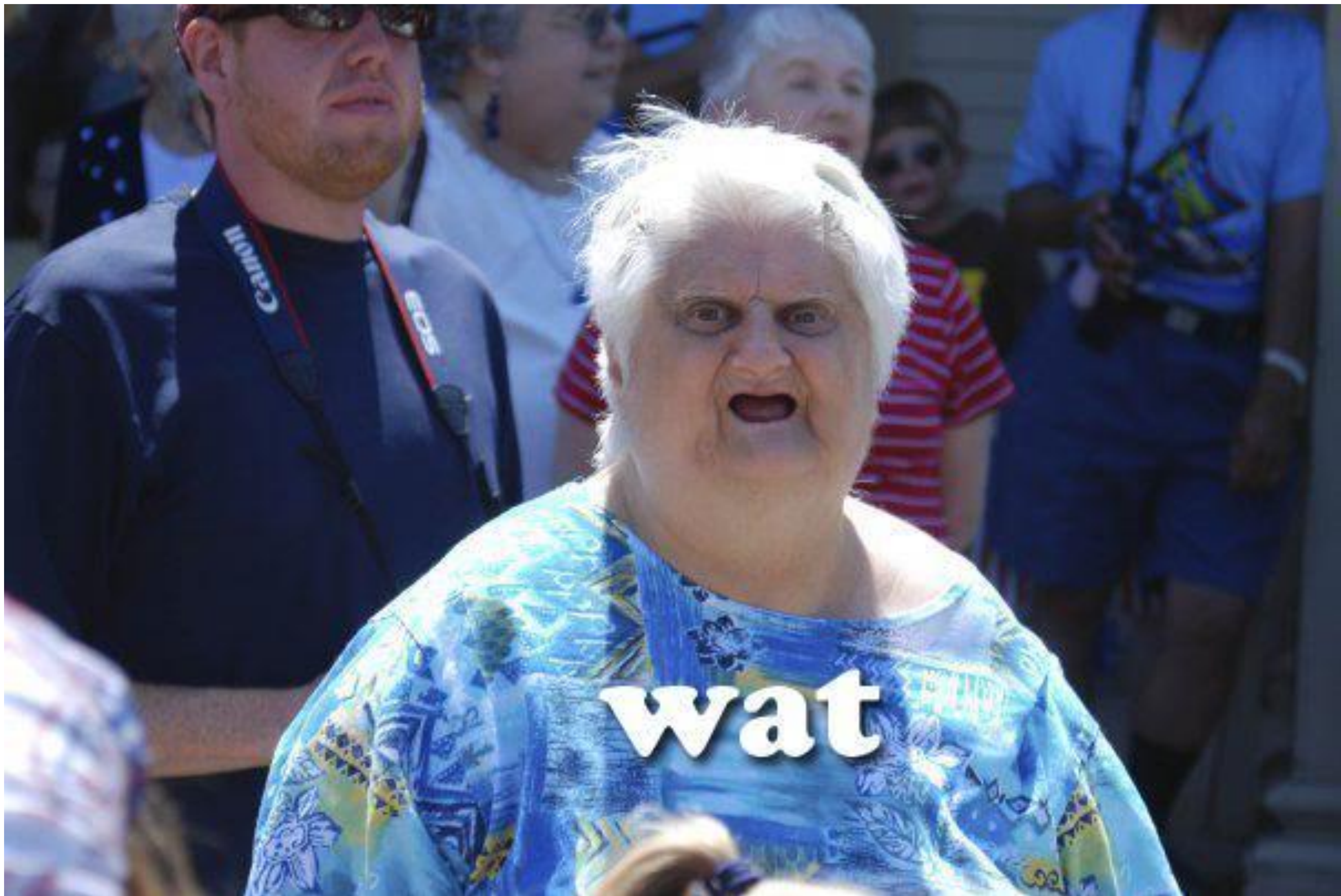
Rust



JAVA SCRIPT

I'm technically functional.





wat

A photograph of a woman with short white hair, wearing a blue patterned top, looking shocked with her mouth wide open. She is in a crowd of people, some of whom are wearing blue shirts with 'Canon' and 'USA' logos. The image is dimmed and has a dark blue overlay.

destroyallsoftware.com/talks/wat

A large, semi-transparent watermark of the ClojureScript logo is centered in the background. It features the word "cljjs" in a stylized font, with "clj" in a dark green color and "js" in a dark blue color. The logo is enclosed within a circular border that is also split into green and blue segments.

ClojureScript

Состояние
Иммутабельность

Макросы

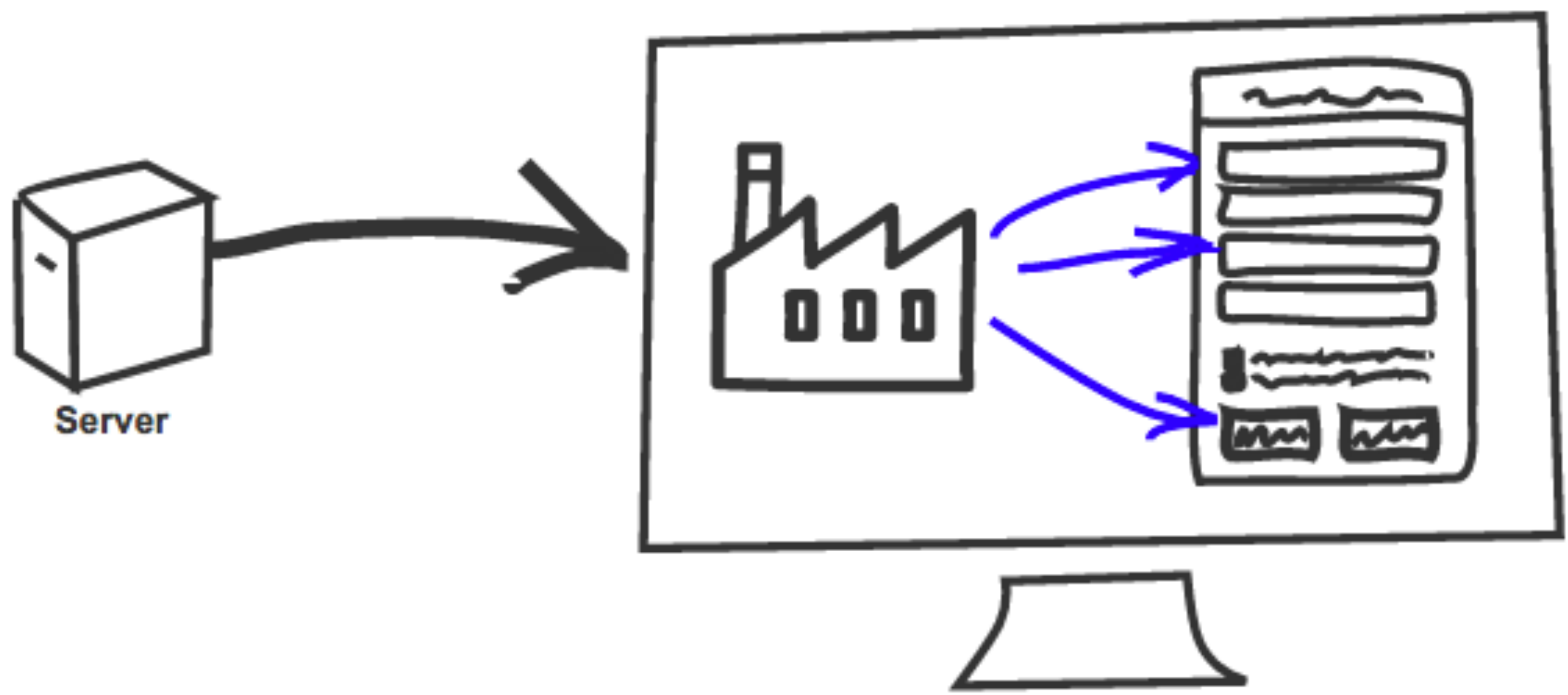
core.async

```
(ns meetup-cljs.core
  (:require-macros [cljs.core.async.macros :refer [go]])
  (:require [cljs.core.async :as async]))

(def state (atom {}))
(def state-watch []
  (let [c (async/chan 1)]
    (add-watch state :w (fn [_ _ _ n] (go (!> c n))))
    c))
```

```
(def state (atom {}))
(def watch-state []
  (let [c (async/chan 1)]
    (add-watch state :w (fn [_ _ _ n] (go (!> c n))))
    c))

(go-loop [] (js/console.log (async/<! (watch-state)))
  (recur))
;; do other stuff
(reset! state 42)
```



```
(defn get [payload]
  (let [push (.push ws-fetch-channel "fetch" payload)
        ok-chan (async/chan)
        error-chan (async/chan) ]
    (.receive push "ok" #(put! ok-chan %))
    (.receive push "error" #(put! error-chan %))
    [chan error-chan]))
```



```
(defn get-kitty []  
  (go  
    (let [[ok-c error-c] (get :kitty)  
          [value c] (alts! [ok-c error-c (async/timeout 1000)])]  
          (condp = c  
            ok-c :>> #(do-smth value)  
            error-c :>> #(show-error value)  
            (show-error "Timeout")))))
```

- core.async
- Source maps
- Tooling
- Documentation
- Figwheel
- cljx/cljc
- Om

- re-frame
- CLJSJS
- Transit
- Bootstrap
- Planck
- Devcards
- ...



*Спасибо за
внимание!*