Совершенствуя Clojure

perfecting clojure

Никита Прокопов
@nikitonsky

Практические советы

руководство по стилю ≈

неполное

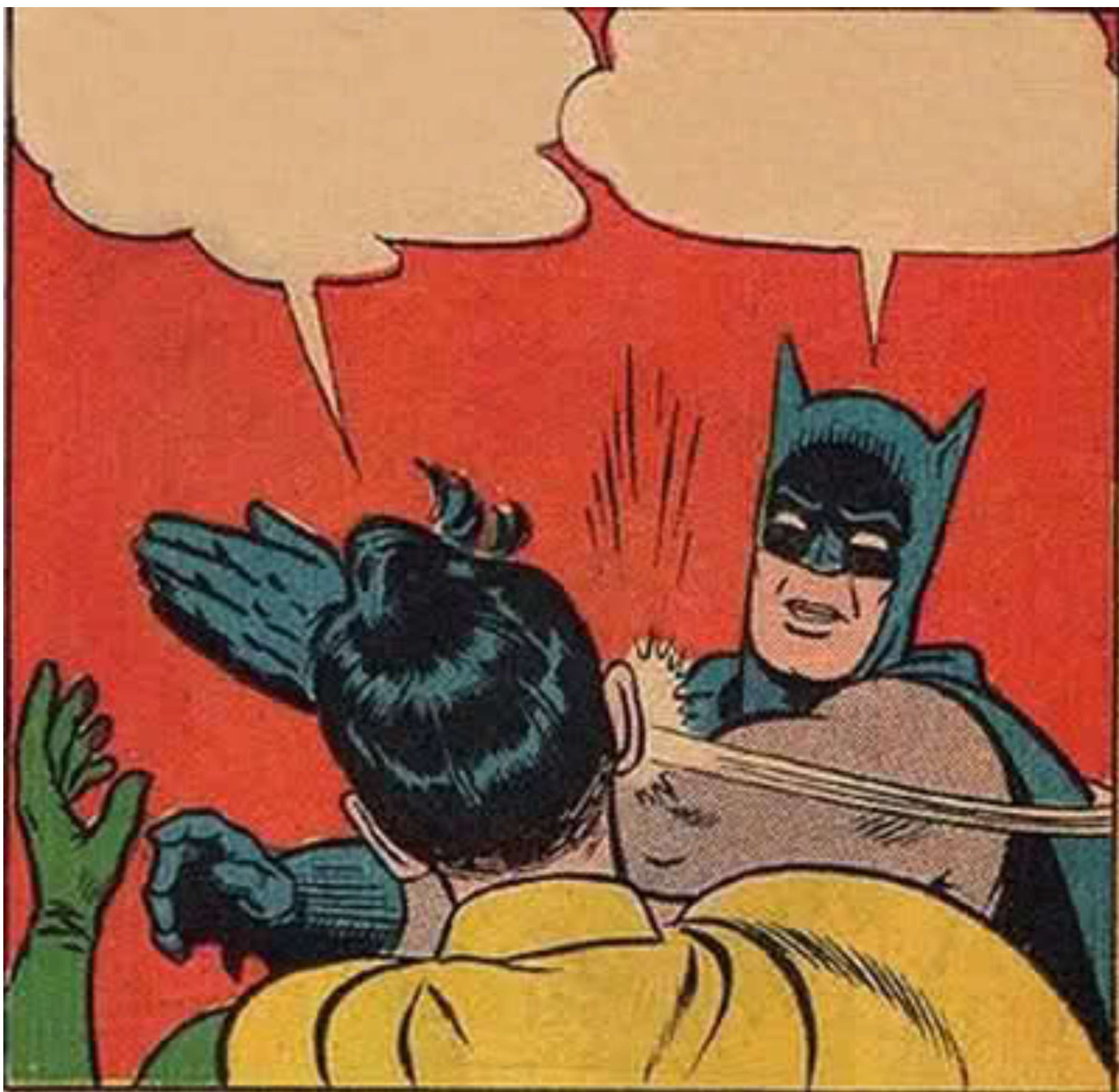
новое

оправданное

Наши идеалы

1. Наглядность
2. Читаемость
3. Компактность

Не используйте `use/refer`



```clojure
(ns ...
  (:use clojure.string))

...

(join ", " coll)
```

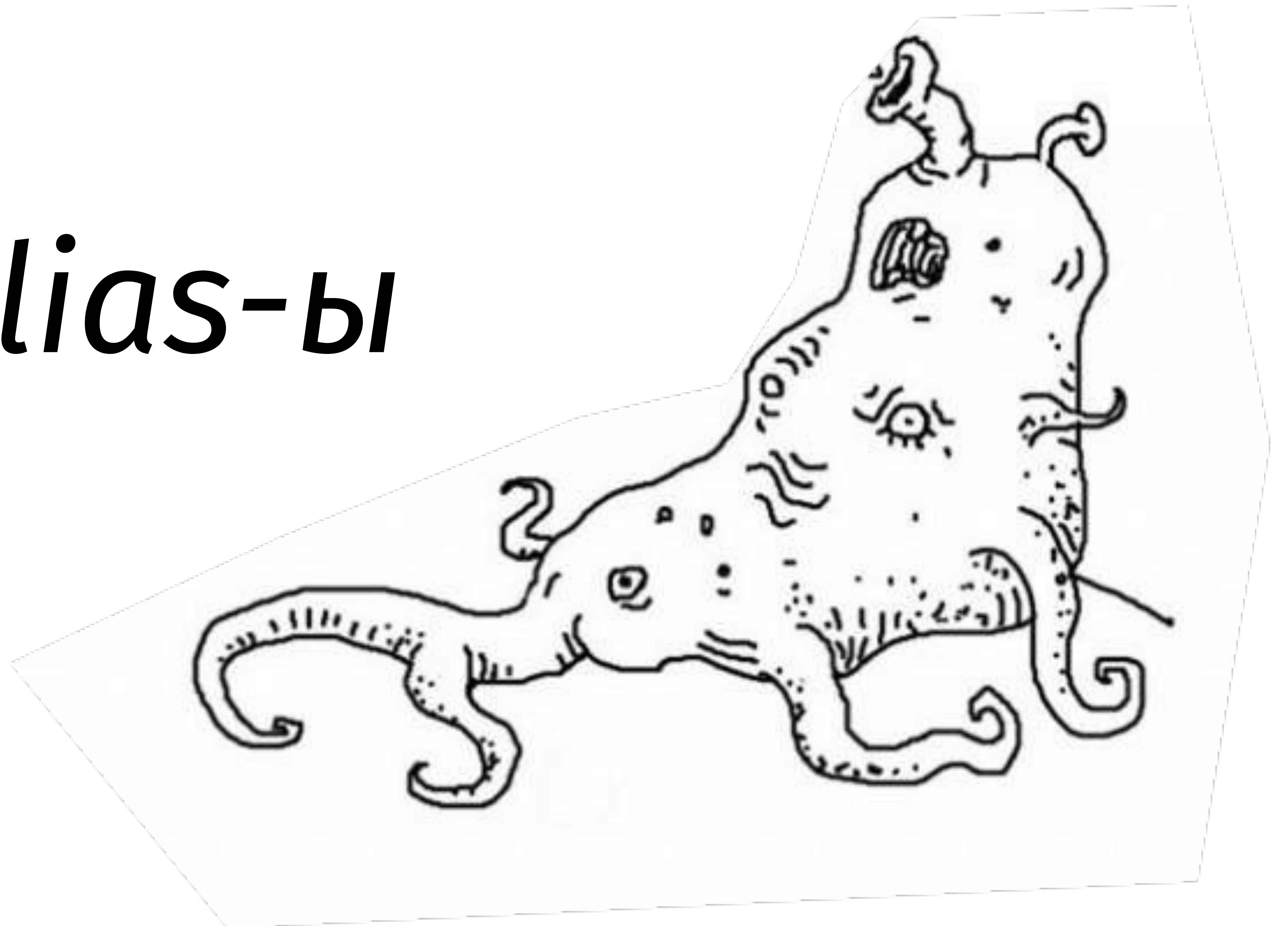
```clojure
(ns ...
  (:require [clojure.string :refer [join]))

...

(join ", " coll)
```

```
(ns ...
  (:require [clojure.string :as str]))

...

(str/join ", " coll)
```

Подчеркивает взаимосвязь между модулями

Бережет от коллизий имен

Глобально уникальные постоянные namespace alias-ы



```clojure
(ns ...
  (:require [clojure.string :as str))
```

```clojure
(ns ...
  (:require [clojure.string :as s]))
```

```clojure
(???/join ", " coll)
```

1 вещь, 1 имя

Не надо думать, как вызвать функцию

Поиск и замена работают

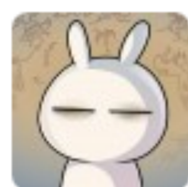
Общий словарь для общения

tors
nts

T MESSAGES
kbot
sky (you)
ers
mley
zo Cavallo

работала хвостовая оптимизация? (edited)

dragoncube 01:34
google



Yurii Makhotkin 01:52
Спс

dragoncube 03:07
gcd известный алгоритм, на нем часто объясняют про рекурсию и прочее

ищи про tail-call elimination gcd

`chat.server/connect` сломан

bold _italics_ ~strike~ `code` ```preformatted``` >quote

Длинные alias-ы

```
[clojure.spec :as spec]
[clojure.test :as test]
[clojure.java.io :as io]
[rum.core :as rum]
[clojure.string :as string]
[cognician.chat.server :as server]
[cognician.chat.server.schema :as server.schema]
[cognician.chat.ui.entries.core :as ui.entries.core
```

```
(d/user ...)

(server.db/user ...)
```

Самоочевидны

Меньше вероятность коллизий

Сложнее использовать —

— лучше разделение на модули

Явное лучше неявного

`set` как функция

`map` как функция

`seq` как проверка на пустоту

`nil` как `false`

```
(def possible-states #{:x :y :z})

(when (possible-states state)
  ...)
```

```clojure
(def possible-states #{:x :y :z})

(when (contains? possible-states state)
  ...)
```

Не нужно ментальной акробатики

Можно читать с любого места

Nil ≠ False

```
(when object
  ...)
```

Требуют расшифровки

Скрывают намерение

Чреваты ошибками

```
(when (some? object)
  ...)
```

```
(or (:param config)
    (:param default-config))

:param :: true | false | nil
```

```
(contains? some-set false)
(some-set false)

(some #(some? (:param %)) coll)
(some :param coll)

(filter some? coll)
(filter identity coll)
```

```
SEQ     empty?
STRING  clojure.string/blank?

NIL     when-some
        if-some
        nil?
        some?
```

First-order функции

```
(fn arg1 arg2 ...)
```

```
comp, partial, complement, every-pred, some-fn, ...
```

```
(map (comp str/capitalize str/trim) strings)
```

```
(map (comp str/capitalize str/trim) strings)

(map #(str/capitalize (str/trim %)) strings)
```

Выглядят непривычно

Требуют ментальной акробатики

Имена нужны

В Clojure:

— Можно сжимать код
— Она сделана под сжимание кода

Threading макросы -> ->> ...

Анонимные ~~алкоголики~~ функции

Деструктуринг

Функции высших порядков

~~Бесточечный стиль~~

```clojure
(defn created-at [db a v]
  (->> (d/datoms db :avet a v)
       first
       :tx
       (d/entity db)
       :db/txInstant))
```

```
(defn created-at [db a v]
  (let [datoms    (d/datoms db :avet a v)
        datom     (first datoms)
        tx        (:tx datom)
        tx-entity (d/entity db tx)]
    (:db/txInstant tx-entity)))
```

```
(reduce #(if (> %1 %2) %1 %2) nums)
```

```
(reduce #(if (> %1 %2) %1 %2) nums)

(reduce (fn [x res] (if (> x res) x res)) nums)
```

Имена помогают обрести почву под ногами

Threading для однотипных

```clojure
(->> users
     (remove #(nil? (:email %)))
     (filter #(> (:age %) 18)))
```

Одноразовые имена

```
(defn normalize-email [email]
  (let [email (str/trim email)
        email (str/lower-case email)]
    email))
```

```clojure
(defn normalize-email [email]
  (let [email (str/trim email)
        email (str/lower-case email)]
    email))
```

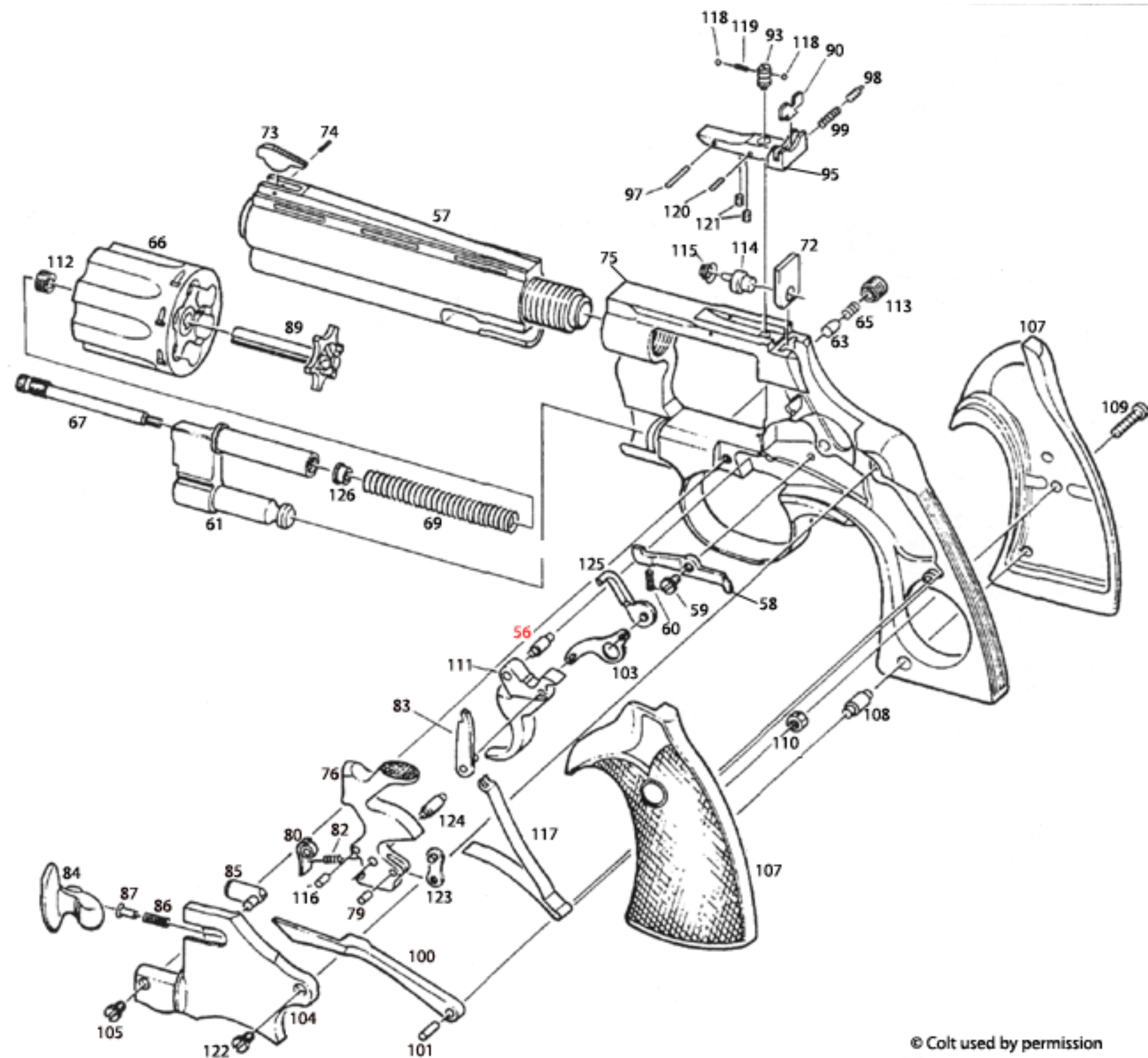
Опции — мапа

```
(defn f [& {:as opts}]
   … )

(f :opt1 … , :opt2 … )
```

```clojure
(defn f [& {:as opts}]
  … )

(f :opt1 … , :opt2 … )


(let [opts { :opt1 … , :opt2 … }]
  (f ???))
```

```clojure
(defn f [opts]
   … )

(f { :opt1 … , :opt2 … })


(let [opts { :opt1 … , :opt2 … }]
  (f opts))
```

Наглядный destructuring



```
(reduce
  #(assoc %2 (first %1) (second %1))
  {}
  id->user)
```

```clojure
(reduce
  (fn [[id user] acc]
    (assoc acc id user))
  {}
  id->user)
```

*Ссылки со звездочкой**

** не является публичной офертой*

Трудно придумать разные имена
для ссылки и ее содержимого

```
(def *state (atom nil))

(let [state @*state]
   … )
```

Не надо следить, когда что разыменовано

Отсылка к указателям в языке C

Совершенствуя Clojure

perfecting clojure

Никита Прокопов
github.com/tonsky
@nikitonsky