

Microservices with Clojure

Microservices Meetup Rhein-Main

Michael Vitz & Joy Clark



Who are we?

Michael Vitz

michael.vitz@innoq.com
@michaelvitz

Joy Clark

joy.clark@innoq.com
@iamjoyclark





~80 people in Germany (Monheim, Berlin, Offenbach, Munich)

~20 people in Switzerland (Zurich, Cham)

Who are you?

Microservices?

Various aspects!

<http://martinfowler.com/articles/microservices.html>



“Do one thing and do it well!”

Easier to understand

```
$ man uniq
```

```
...
```

DESCRIPTION

```
The uniq utility reads the specified input_file comparing adjacent lines, and  
writes a copy of each unique input line to the output_file.
```

```
...
```



Understanding a monolith...

```
$ man my-huge-monolith
```

```
<endless text>
```

Composability

```
ps -ef |  
grep program-i-dont-like |  
grep -v grep |  
awk '{print $2}' |  
kill
```

“Do one thing and do it well!”?

Yes!

“One thing” - which?

“Organize around
business capabilities!”



Don't build things like “frontend”,
“api” and “database access”

—

instead, build “fulfillment”

and “payment”.

“Organize around
business capabilities!”?

Yes!

All of this could be done with
Libraries.

Would that be Microservices?



Code Abstractions

Function

(Class)

Library

Component

Service



Service

Communicates via network

Independently deployable

Technologically independent

Clear module boundaries



Microservices promise
very loose coupling
but
you have to handle
the complexity of
distributed systems!



This requires rules and
conventions across systems.



Good starting point:
<http://12factor.net/>

Rules & conventions

Project setup is done more than once
→ has to be easy

Deployment is done all the time
→ must be automated

Many apps are configured and write logs
→ should be simple for every app, common format,
Splunk or ELK

Rules & conventions

Many services are talking to each other

→ standardize on protocols and data formats

The network is not reliable

→ asynchronous programming and stability
patterns for decoupling and resilience

It's easy to lose track of what's actually happening

→ gather metrics for monitoring



Given such rules,
how to play nice using Clojure?



Clojure Crash Course

```
(println "Hello Frankfurt!")
```

Lisp + JVM

Functional programming language

Simple programming model

Clojure Crash Course

```
(println "Hello Frankfurt!")
```

Lisp + JVM

Functional programming language

Simple programming model

Clojure Crash Course

```
{:name "Clojure"  
 :features [:functional :jvm :parens]  
 :creator "Rich Hickey"  
 :stable-version {:number "1.8.0"  
                  :release "2016/01/19"}}
```

Clojure Crash Course

```
(defn main [args] (println "Hello World!"))
```

VS.

```
public static void main(String[] args) {  
    System.out.println("Hello World!");  
}
```

Clojure Crash Course

```
(+ 1 2 3)
```

```
> 6
```

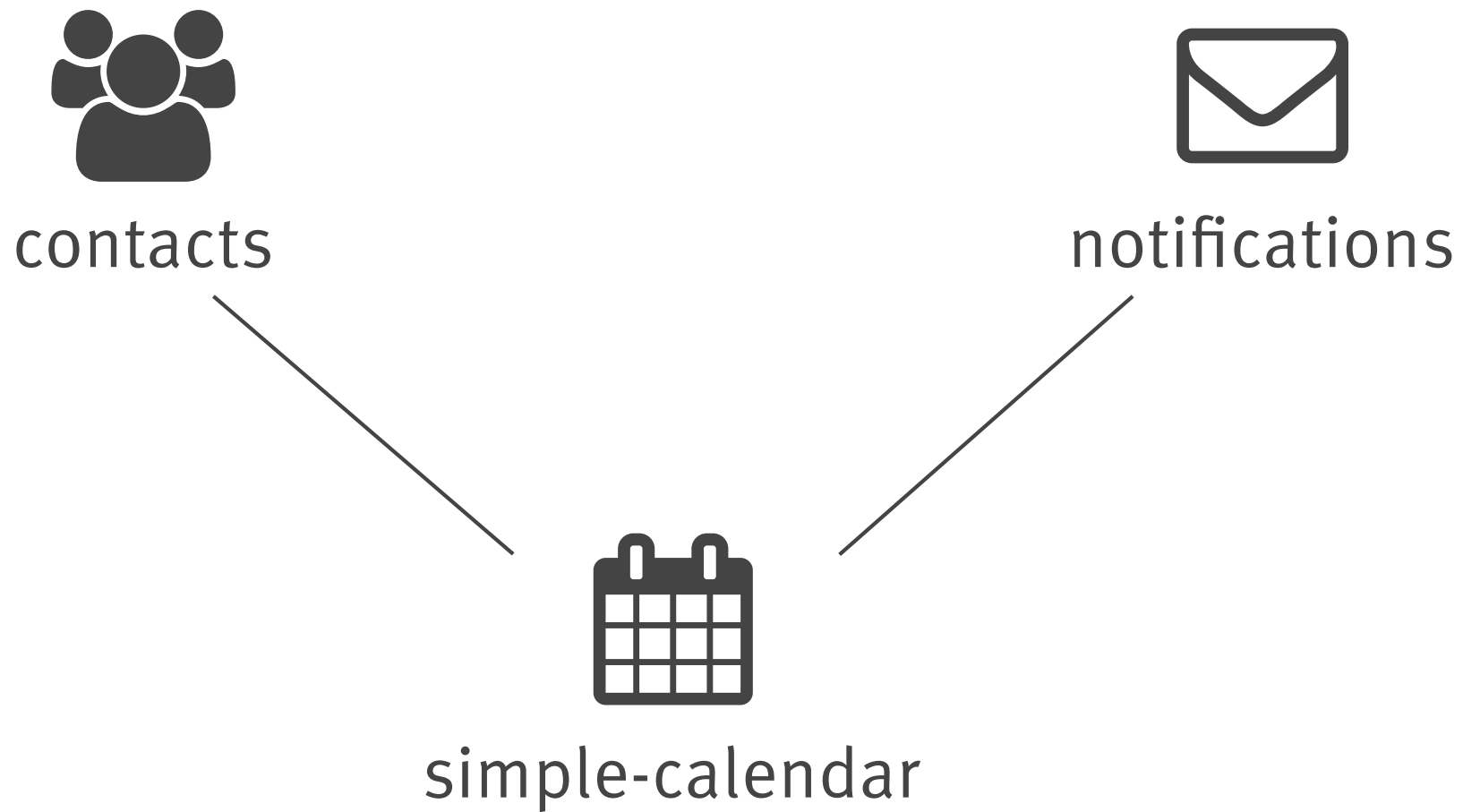
```
(:city {:name "innoQ"  
        :city "Offenbach"})
```

```
> "Offenbach"
```

```
(map inc [1 2 3])
```

```
> (2 3 4)
```


Example



Project setup



derivative of "Garuda Takeoff" by Simon_sees (CC BY 2.0)

Leiningen

Alternative to Maven

Describes Clojure project with generic data structures

Maven repository compatibility

Offers plugin system



Leiningen

```
(defproject simple-calendar "0.1.0-SNAPSHOT"
  :description "simple calendar app"
  :url "https://github.com/innoq/simple-calendar"
  :dependencies [[org.clojure/clojure "1.8.0"]
                 [compojure "1.4.0"]
                 [ring "1.4.0"]
                 [ring/ring-json "0.4.0"]
                 ...
                 [environ "1.0.2"]]
  :plugins [[lein-ring "0.9.7"]
            [lein-environ "1.0.2"]]
  :ring {:handler simple-calendar.core/webapp
         :init simple-calendar.core/init}
  :profiles {:uberjar {:aot :all}})
```



Automated deployment

derivative of “Launch Button -- SMASH Rocket Club 5-9-09 4” by Steven Depolo (CC BY 2.0)

Automated deployment

Not really a language thing ...

... but good tooling can help a lot

Leiningen makes building fat JARs easy

WAR files can be generated as well

Logging & configuration

Logging

Don't let every app handle log files

Just write everything to stdout

Let some external tool handle storage

Standardize on log format

org.clojure/tools.logging

Macros delegating to different implementations

Supports slf4j, Apache commons-logging, log4j
and java.util.logging

Logging levels: `:trace` `:debug` `:info` `:warn`
`:error` `:fatal`

org.clojure/tools.logging

```
(ns simple-calendar.core
  (:require [clojure.tools.logging :as log]))

(defn update-contact! [url]
  ...
  (log/info "Updated user" id new-email))
```

project.clj:

```
:dependencies [[org.clojure/tools.logging "0.3.1"]
               [log4j "1.2.17" :exclusions [javax.mail/mail
                                             javax.jms/jms
                                             com.sun.jdmk/jmxtools
                                             com.sun.jmx/jmxri]]
               [org.slf4j/slf4j-log4j12 "1.7.18"]
               ...]
```

org.clojure/tools.logging

log4j.properties:

```
log4j.rootLogger=INFO, standard
```

```
log4j.appender.standard=org.apache.log4j.ConsoleAppender
```

```
log4j.appender.standard.Target=System.out
```

```
log4j.appender.standard.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.standard.layout.ConversionPattern=%d{yyyy-mm-dd HH:mm:ss,SSS} [%p] %c - %m%n
```

stdout:

```
2015-11-18 13:11:54,468 [INFO] simple-calendar.core - Updated user 5f565040 eve@example.org
```

```
2015-11-18 13:11:54,476 [INFO] simple-calendar.core - Updated user 786494ef bob@example.org
```

Configuration

Store configuration in the environment, not in the codebase

Use mechanism that works for every application, e.g., environment variables

environ

Manages environment settings following 12factor

Reads values from

- Java system properties

- Environment variables

- `.lein-env` (created via Leiningen plugin from `profiles.clj`)

environ

```
(def contacts-feed (env :contacts-feed))
```

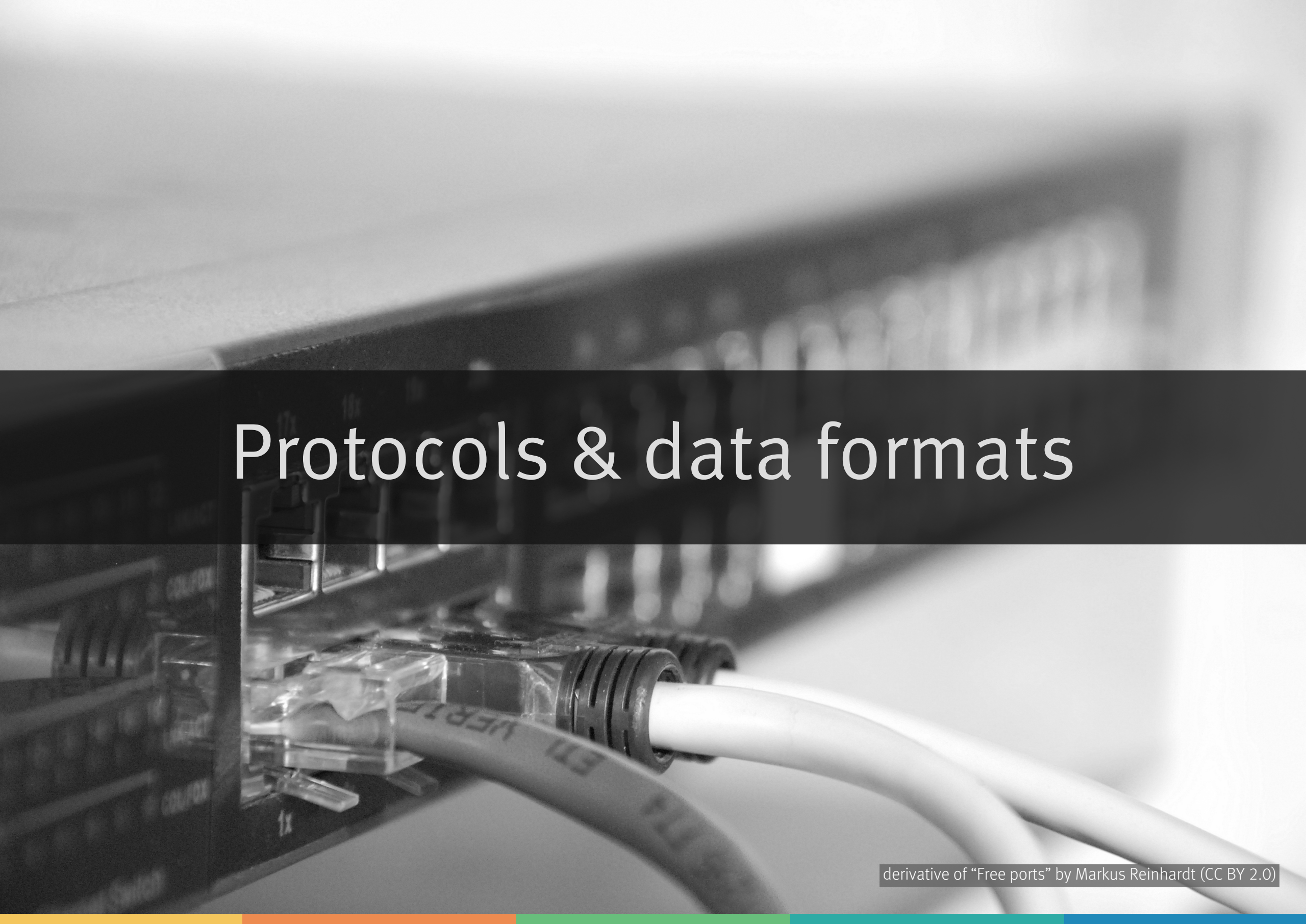
```
> java -Dcontacts.feed=http://contacts.example.org/feed  
      -jar standalone.jar
```

```
> CONTACTS_FEED=http://contacts.example.org/feed  
   lein ring server-headless
```

```
> lein ring server-headless
```

```
profiles.clj:
```

```
{:dev {:env {:contacts-feed "http://contacts.example.org/feed"}}}
```



Protocols & data formats

derivative of “Free ports” by Markus Reinhardt (CC BY 2.0)

Protocols & data formats

Avoid using different protocols everywhere

Standardize on the outside:

HTTP

JSON

XML

Atom



HTTP server

Ring for HTTP basics

Compojure for routing

Request & response are data

A web app is a function which takes a request and returns a response

Ring

```
(def example-request {:uri "/contacts"  
                      :request-method :get  
                      :headers {"Accept" "text/plain"}})
```

```
(defn example-app [req]  
  {:status 200  
   :body (str "Hello at " (:uri req))})
```

```
(example-app example-request)  
> {:status 200  
   :body "Hello at /contacts"}
```

Compojure

```
(defroutes contacts
  (GET "/contacts/:id" [id]
    (get-contact id))
  (PUT "/contacts/:id" [id :as request]
    (update-contact! id (:body request)))
  (POST "/contacts" request
    (add-contact! (:body request)))
  (GET "/feed" []
    {:status 200
     :headers {"Content-Type" "application/atom+xml"}
     :body (contacts-feed)})))
```

Frameworks

Quickly get a web app up and running.

Duct

Luminus

Pedestal

Modularity

tesla-microservices

org.clojure/data.json

Clojure data structures are JSON superset

```
(json/write-str {:name "Alice Miller"  
                 :teams ["Team A", "Team B"]})
```

```
> "{\"name\":\"Alice Miller\",\"teams\":[\"Team A\",\"Team B\"]}"
```

```
(json/read-str "{\"id\": 123, \"name\": \"Alice Miller\"}"  
               :key-fn keyword)
```

```
> {:id 123, :name "Alice Miller"}
```

HTML

HTML is represented with generic data structures

This allows processing HTML with standard Clojure functions

There are different formats, a popular one is called “hiccup”

HTML

```
<html>
  <body>
    <h1>hello!</h1>
    
    
  </body>
</html>
```

```
[ :html {}
  [ :body {}
    [ :h1 {} "hello!" ]
    [ :img { :src "some.jpg" } ]
    [ :img { :src "another.jpg" } ] ] ] ]
```

Atom

XML format for representing feeds of data

Useful for doing pub/sub without middleware

Creating Atom Feeds

```
(defn entry [event]
  [:entry
   [:title (-> event :type name)]
   [:updated (:timestamp event)]
   [:author [:name "contacts service"]]
   [:id (str "urn:contacts:feed:event:" (:id event))]
   [:content {:type "json"} (json/generate-string event)]]])
```

```
(defn atom-feed [events url]
  (clojure.data.xml/emit-str
   (clojure.data.xml/sexp-as-element
    [:feed {:xmlns "http://www.w3.org/2005/Atom"}
     [:id "urn:contacts:feed"]
     [:updated (-> events last :timestamp)]
     [:title {:type "text"} "contacts events"]
     [:link {:rel "self" :href url}]
     (map entry events)])))
```

Validation (clojure.spec)

```
[org.clojure/clojure "1.9.0-alpha5"]
```

```
(require '[clojure.spec :as s])
```

```
(defn id [] (str (java.util.UUID/randomUUID)))
```

```
(def event {:type :contact-created  
            :timestamp (.getTime (java.util.Date.))  
            :id (id)})
```

Validation (clojure.spec)

```
(s/def ::timestamp long?)
```

```
(s/def ::type keyword?)
```

```
(s/def ::id string?)
```

```
(s/def :unq/event (s/keys :req-un [::type ::timestamp ::id]))
```

```
(s/conform :unq/event event)
```

```
> {:type :contact-created  
   :timestamp 1465388506376  
   :id "c4f2381c-4152-4d77-b55f-fb7d7102998c"}
```

Validation (clojure.spec)

```
(s/conform :unq/event {:type :contact-created})
```

```
> :clojure.spec/invalid
```

```
(s/explain :unq/event {:type :contact-created})
```

```
> val: {:type :contact-created} fails spec: :unq/event predicate:  
  [(contains? % :timestamp) (contains? % :id)]
```

```
(s/explain :unq/event {:type "contact-created"})
```

```
> val: {:type :contact-created} fails spec: :unq/event predicate:  
  [(contains? % :timestamp) (contains? % :id)]  
In: [:type] val: "contact-created" fails spec: :user/type at:  
  [:type] predicate: keyword?
```

adamwynne/feedparser-clj

Retrieves and Parses RSS/Atom feeds

```
(def f
  (feedparser/parse-feed "https://www.innoq.com/de/podcast.rss"))

(:title f)
> "innoQ Podcast"

(count (:entries f))
> 18
```

Library for consuming feeds based on feedparser:
Feedworker

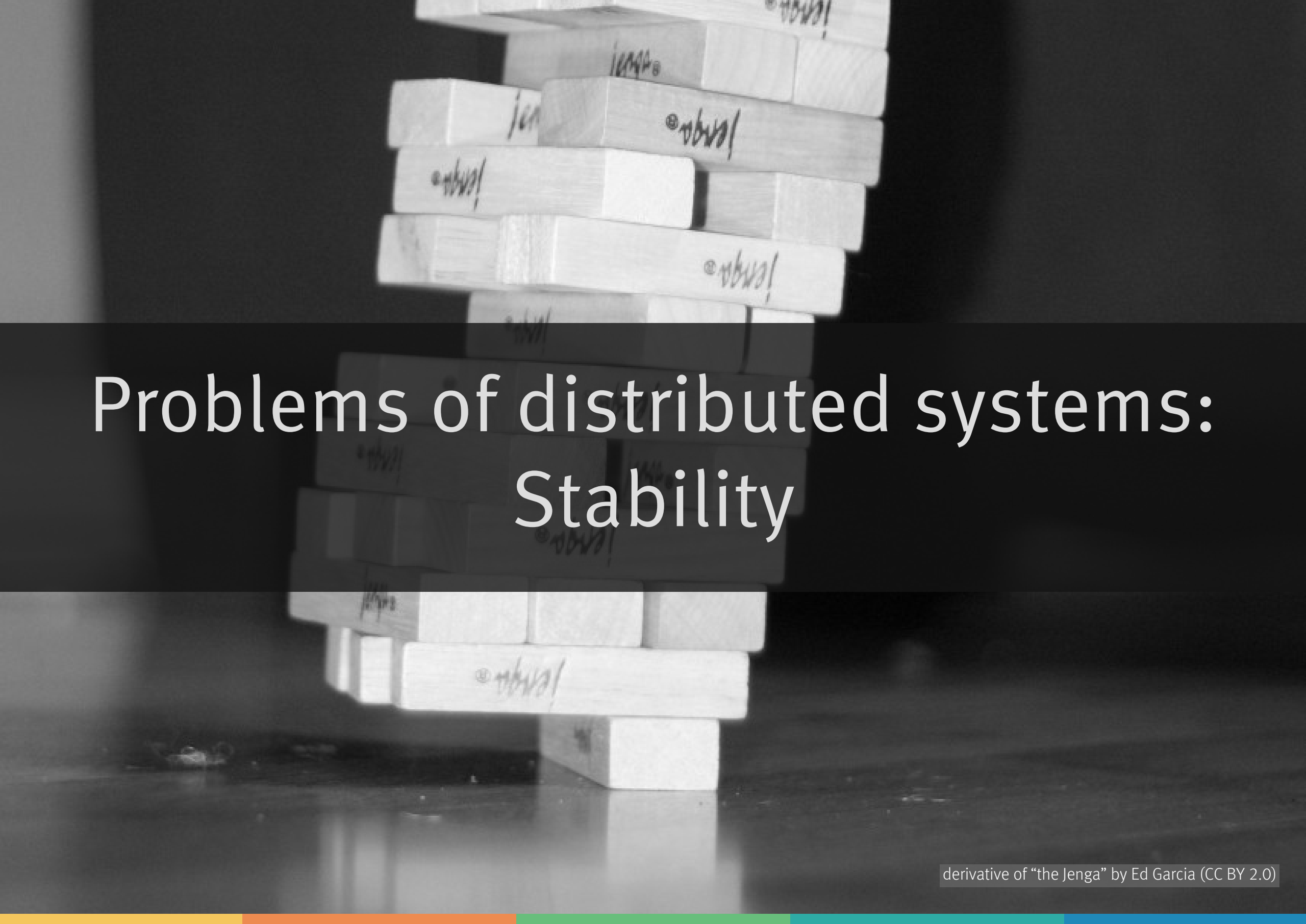
Lightweight messaging

Just a side note:

Kafka (clj-kafka)

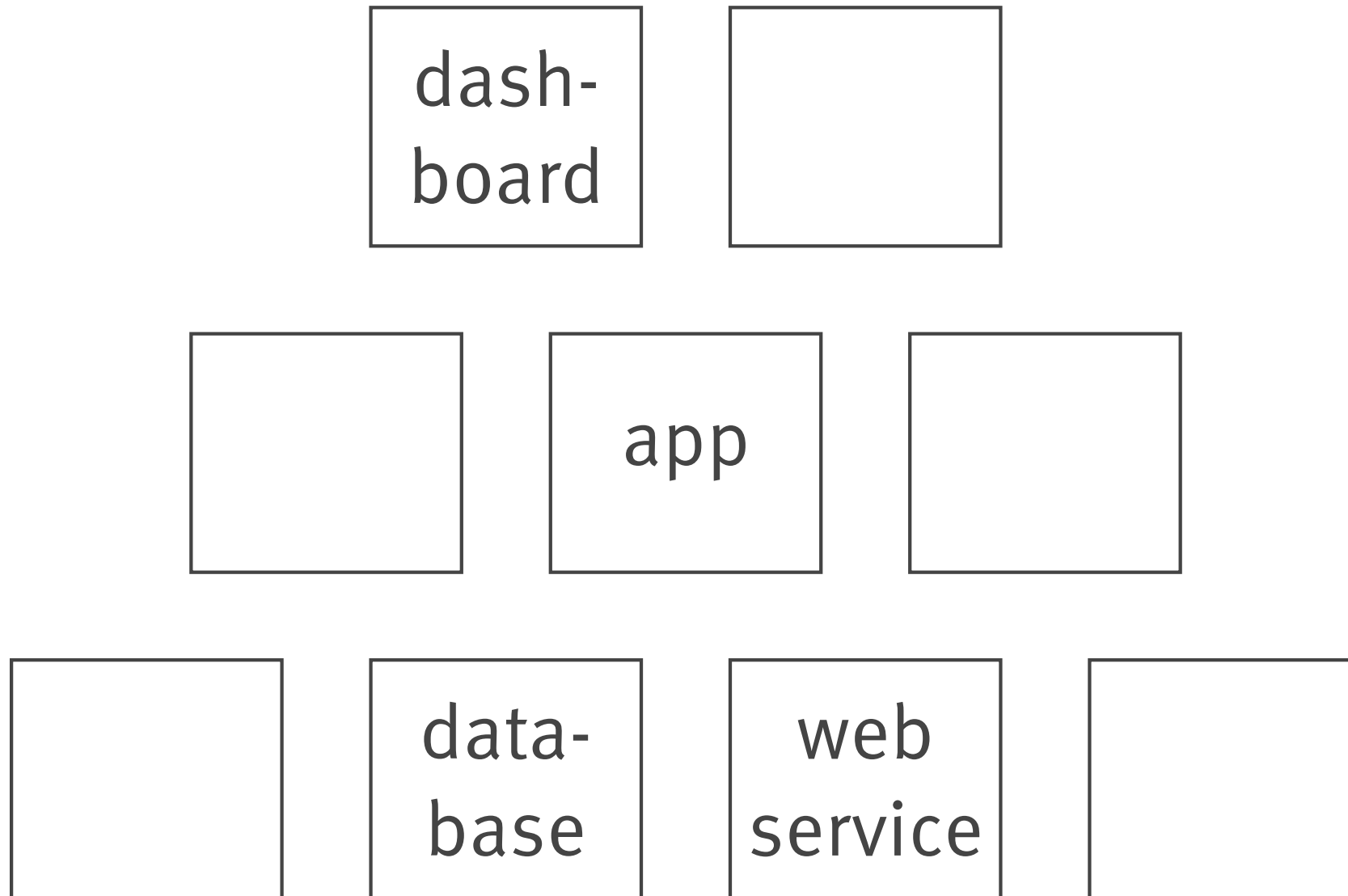
RabbitMQ (Langohr)

...

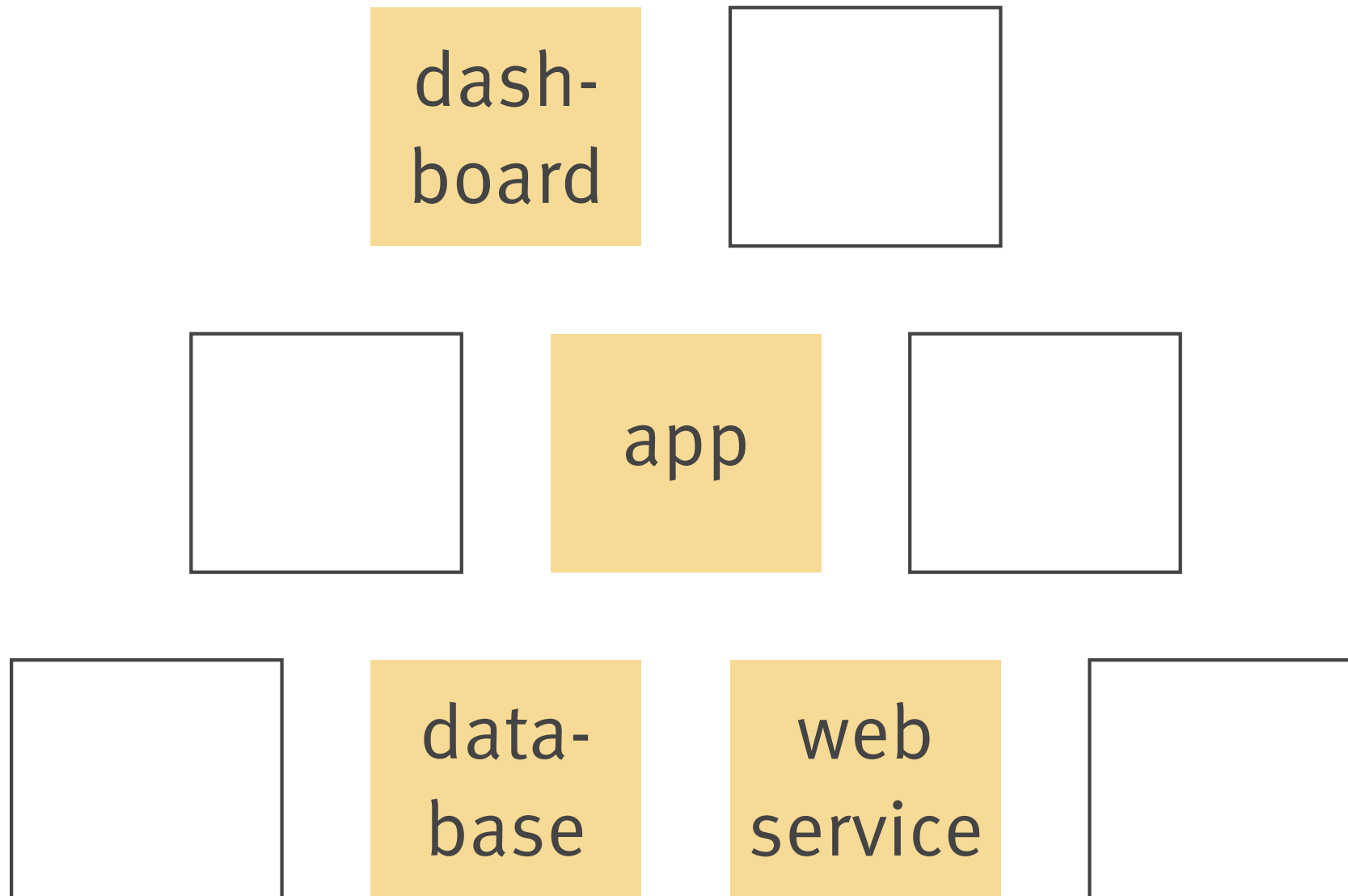


Problems of distributed systems: Stability

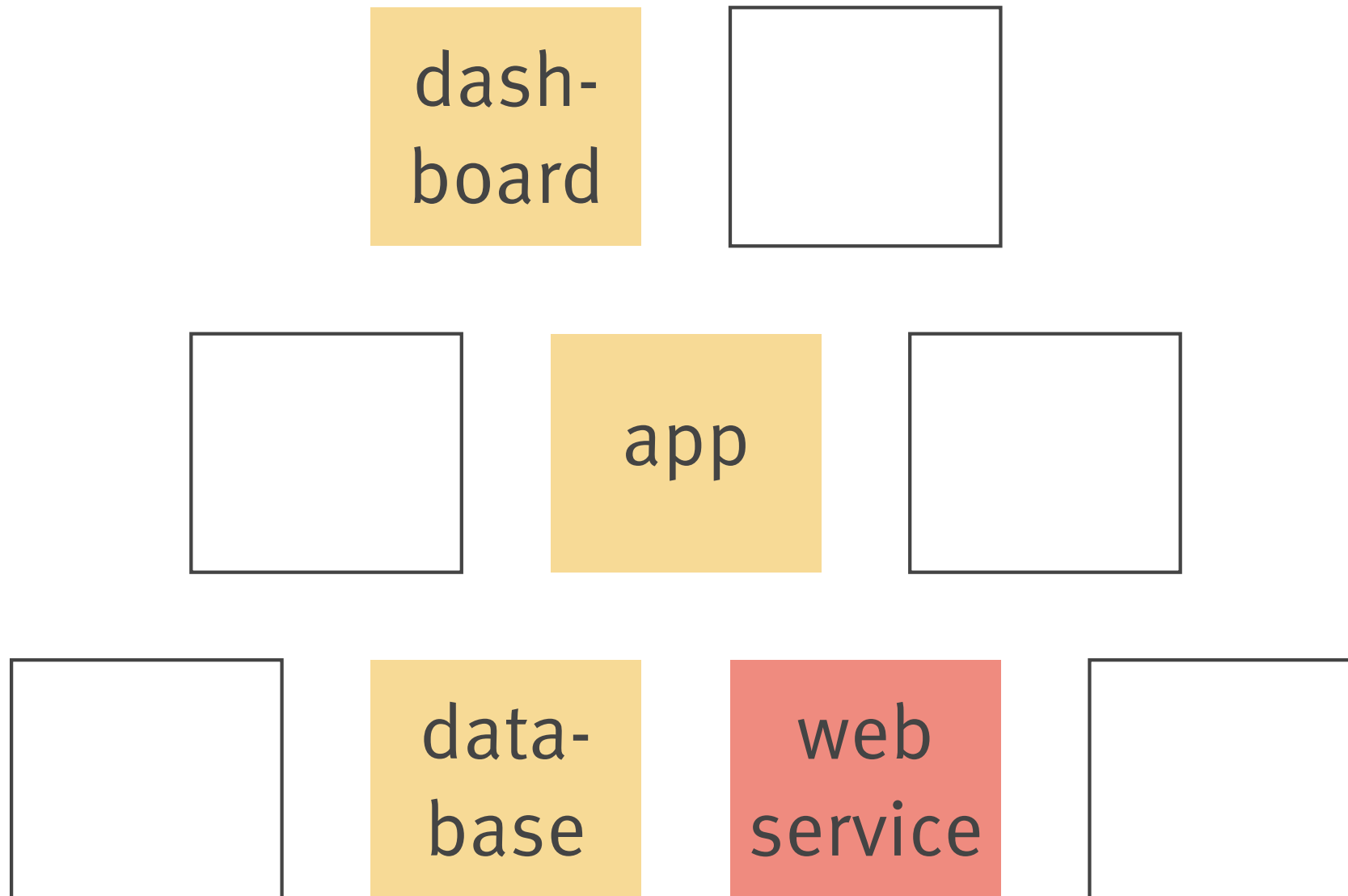
Cascading failures



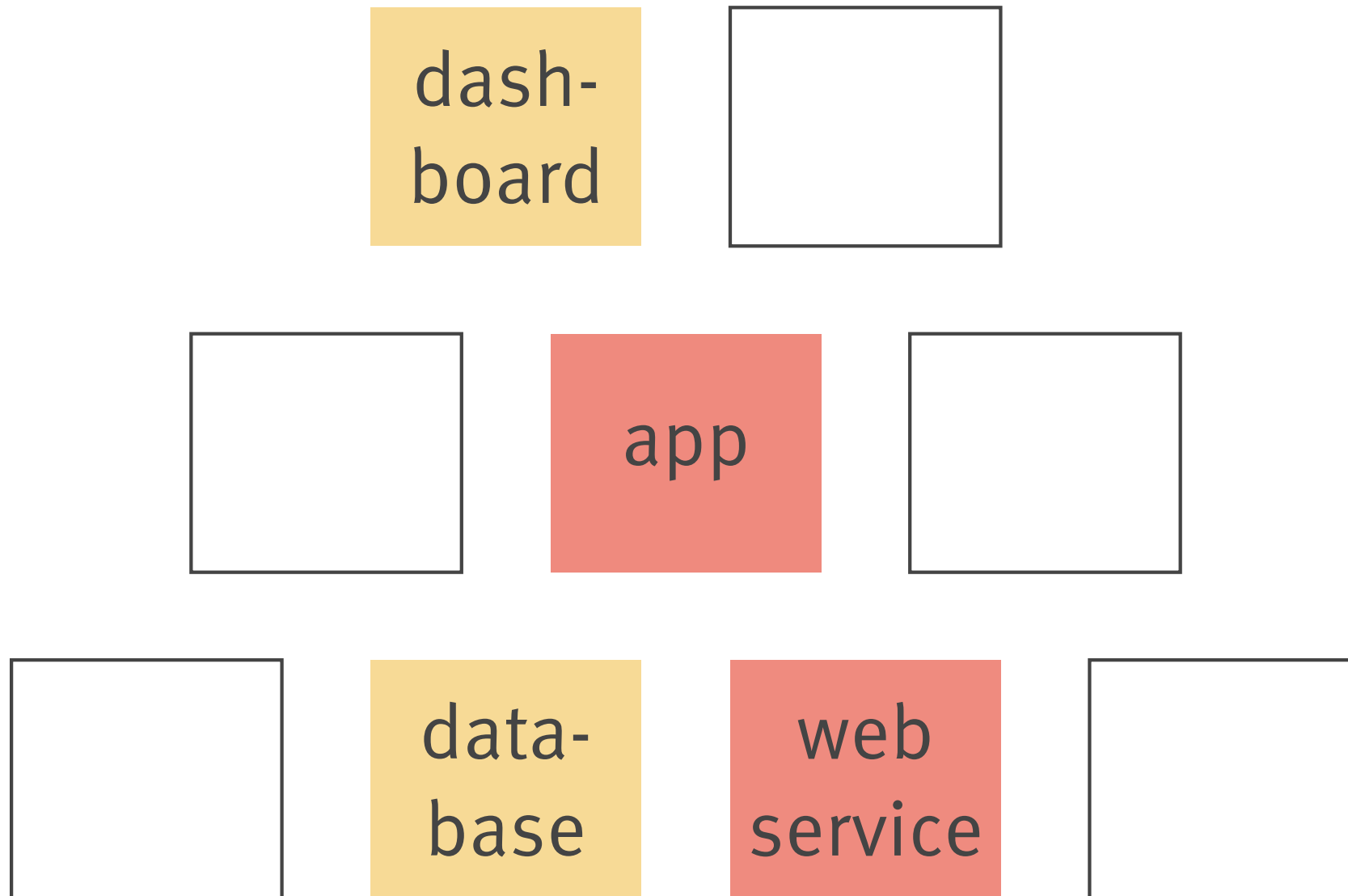
Cascading failures



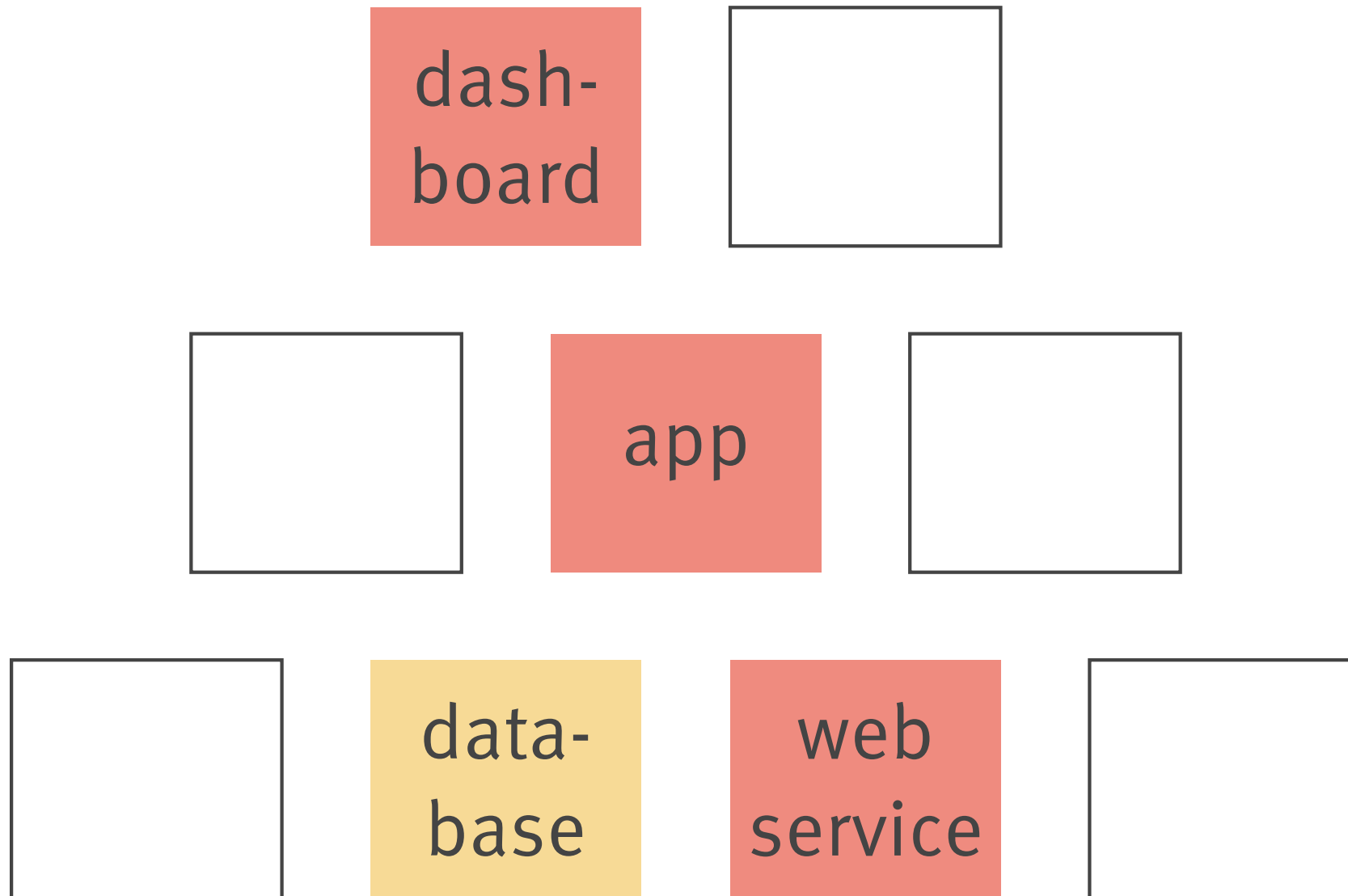
Cascading failures



Cascading failures



Cascading failures



How to achieve stability?

A black and white illustration of the Titanic sinking. The top half shows the ship's bow and upper decks as it plunges into the water, with smoke billowing from the funnels. The bottom half shows a chaotic scene of people in lifeboats and the water, with many people waving or reaching out. The ship's name 'TITANIC' is visible on the bow.

Bulkheads

derivative of "Titanic sinking" by Willy Stöwer (Public Domain)

Asynchronous communication

Helps to decouple and separate different application parts

E-mail notifications do not need to be sent synchronously

org.clojure/core.async

Supports asynchronous programming and communications

Messages can be sent to and read from channels

Channels can be buffered or unbuffered

Blocking and non-blocking operations possible

org.clojure/core.async

```
(def notifications (chan 1000))
```

org.clojure/core.async

```
(def notifications (chan 1000))
```

org.clojure/core.async

```
(def notifications (chan 1000))
```

```
(defn send-notification [email event-link]
  (go (>! notifications {:email email
                          :event-link event-link})))
```

org.clojure/core.async

```
(def notifications (chan 1000))
```

```
(defn send-notification [email event-link]
```

```
  (go (>! notifications {:email email  
                        :event-link event-link})))
```

org.clojure/core.async

```
(def notifications (chan 1000))
```

```
(defn send-notification [email event-link]
```

```
  (go (>! notifications {:email email  
                        :event-link event-link})))
```

org.clojure/core.async

```
(def notifications (chan 1000))
```

```
(defn send-notification [email event-link]  
  (go (>! notifications {:email email  
                        :event-link event-link})))
```

```
(defn notify-user [email event-link]  
  ...)
```

```
(defn start-notifier []  
  (go-loop [message (<! notifications)]  
    (notify-user (:email message) (:event-link message))  
    (recur (<! notifications))))
```

org.clojure/core.async

```
(def notifications (chan 1000))
```

```
(defn send-notification [email event-link]  
  (go (>! notifications {:email email  
                        :event-link event-link})))
```

```
(defn notify-user [email event-link]  
  ...)
```

```
(defn start-notifier []  
  (go-loop [message (<! notifications)]  
    (notify-user (:email message) (:event-link message))  
    (recur (<! notifications))))
```

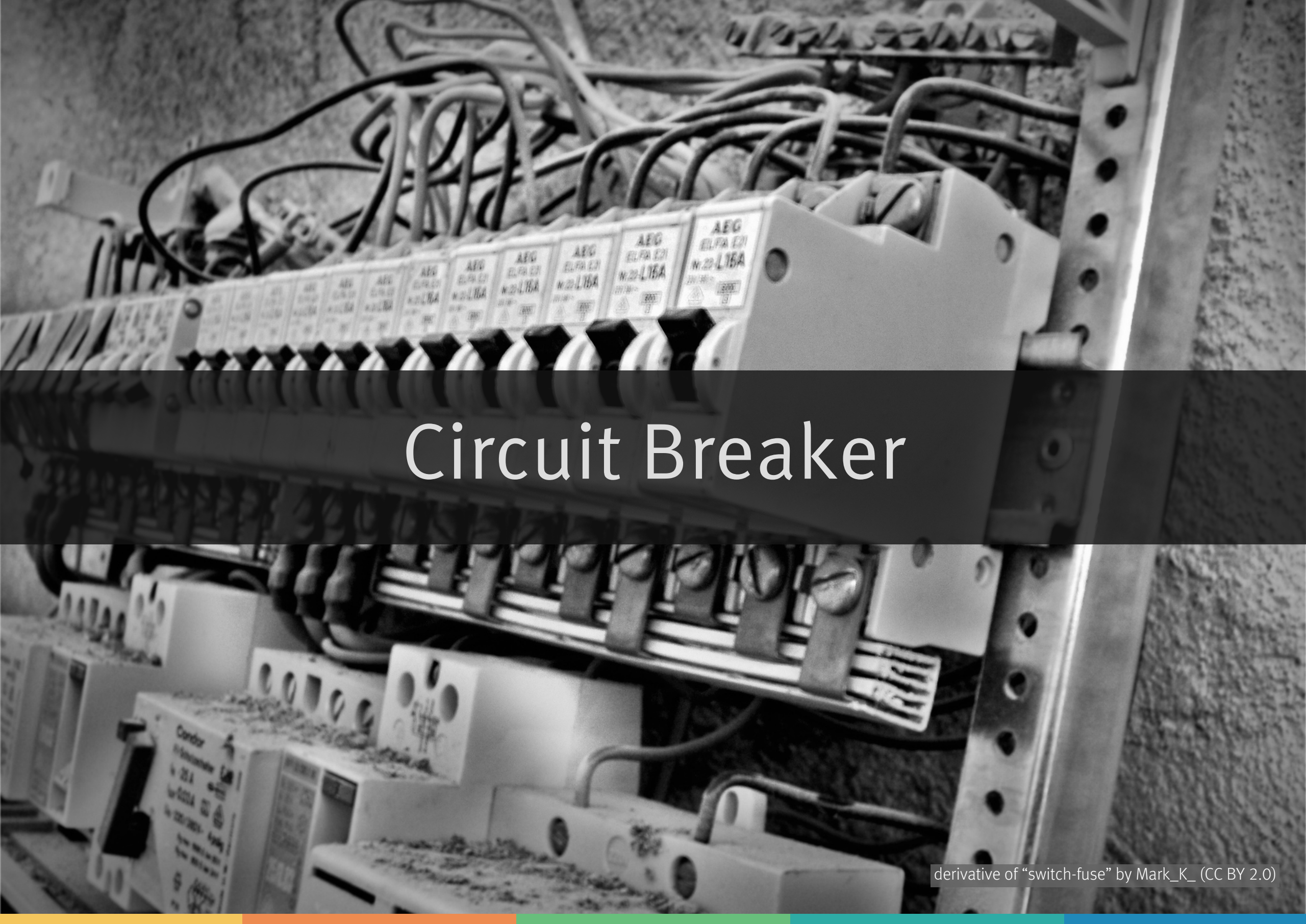
org.clojure/core.async

```
(def notifications (chan 1000))
```

```
(defn send-notification [email event-link]  
  (go (>! notifications {:email email  
                        :event-link event-link})))
```

```
(defn notify-user [email event-link]  
  ...)
```

```
(defn start-notifier []  
  (go-loop [message (<! notifications)]  
    (notify-user (:email message) (:event-link message))  
    (recur (<! notifications))))
```

Circuit Breaker

derivative of "switch-fuse" by Mark_K_ (CC BY 2.0)

com.netflix.hystrix/hystrix-clj

Idiomatic Clojure wrapper for Hystrix

```
(hystrix/defcommand notify-user [email event-link]  
  (client/post notification-service  
    {:content-type :json  
     :body ... })))
```

Will throw exception in case of timeout or other failure

com.netflix.hystrix/hystrix-clj

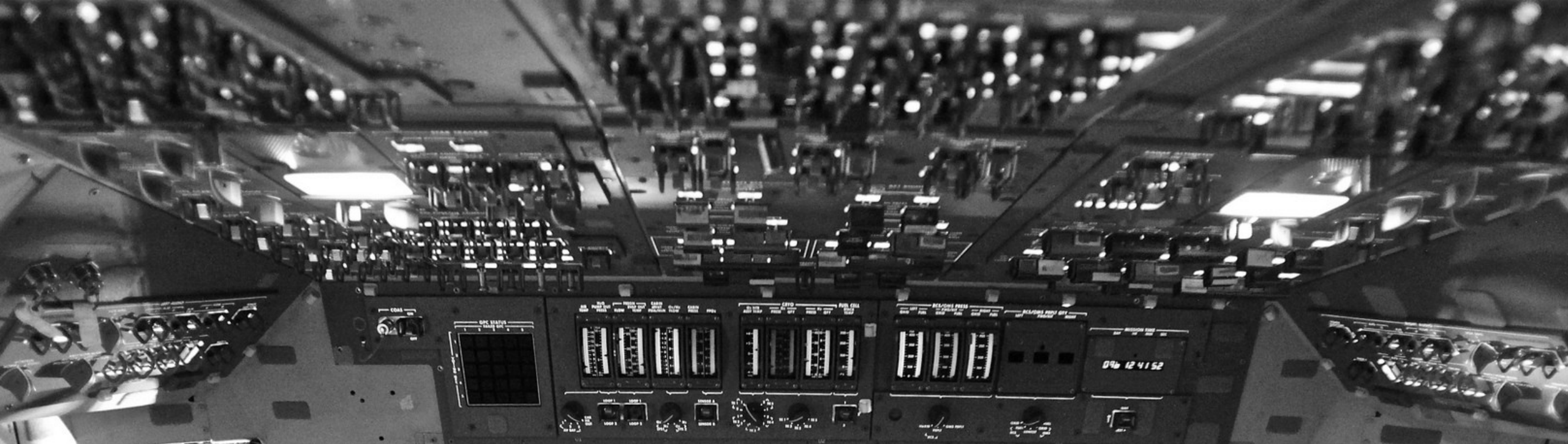
```
; returns true if sucessful, false if circuit-breaker open
(hystrix/defcommand notify-user
  {:hystrix/fallback-fn notify-fallback}
  [email event-link]
  (client/post notification-service
    {:content-type :json
     :body ... })
  true)

; returns false if circuit-breaker open
(defn notify-fallback [email event-link]
  (let [isOpen (.isCircuitBreakerOpen hystrix/*command*)]
    ; add message to queue again
    (send-notification email event-link)
    (not isOpen)))
```

com.netflix.hystrix/hystrix-clj

```
(defn start-notifier []  
  (go-loop [message (<! notifications)]  
    (if-not (notify-user (:email message)  
                (:event-link message))  
      (do  
        (log/error "Cannot reach notification service"  
                    "- will wait until next try")  
        (<! (timeout 5000))))  
    (recur (<! notifications))))
```

hystrix-event-stream-clj available as well



Monitoring



derivative of "Space Shuttle Endeavour's Control Panels" by Steve Jurvetson (CC BY 2.0)

Questions

How many HTTP errors are occurring?

Are database queries failing?

Is that backend service slow again?

How many jobs are in the queue?

Metrics

Logging provides a stream of events

Metrics provide aggregated state

Popular library: Dropwizard Metrics

Two steps: collect metrics, then publish them

TEMPERATURE LIQUIDE DE
REFROIDISSEMENT APRES

CT 001
6 POU 31

Gauges

REFROIDISSEUR AIR DE BARRAGE

Gauges

Current state of one single value

Number of jobs in the queue

Some configured value

Ratio of cache hits to misses

Gauges

```
(def metrics-registry (new-registry))

(gauge-fn metrics-registry "jobs-ready"
  #(query database
    "select count(*)
    from jobs
    where status = 'ready'"))
```

OUR NATIONAL DEBT:
\$8,348,833,366,044.
YOUR Family share
\$91,130.
THE NATIONAL DEBT CLOCK

Counters

Counters

Single numeric value

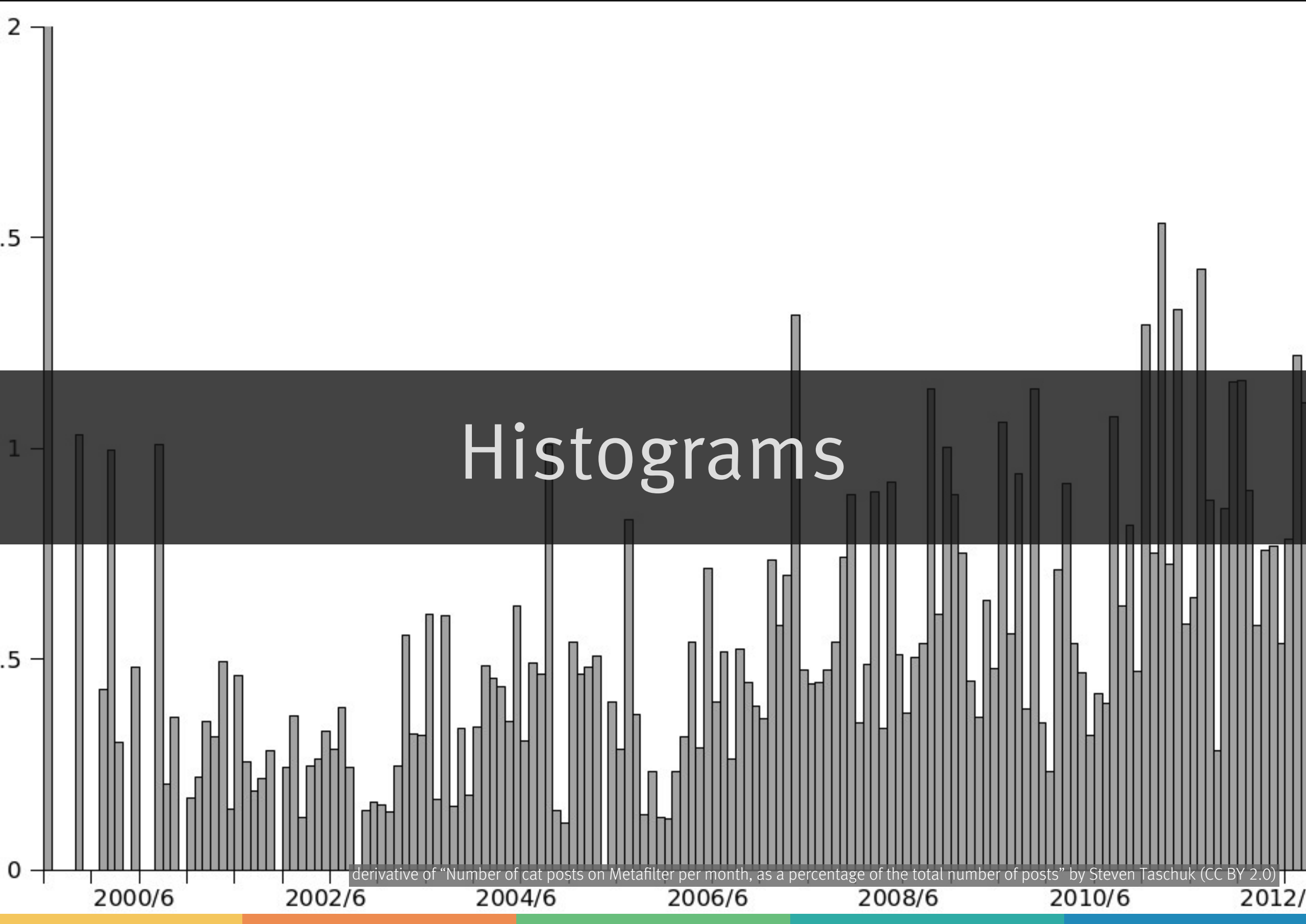
For example, number of logged in users

Counters

```
(def logged-in-users (counter metrics-registry "logged-in-users"))

(POST "/login" [name pwd]
  ...
  (inc! logged-in-users)
  ...)

(POST "/logout" [name]
  ...
  (dec! logged-in-users)
  ...)
```



Histograms

Distribution of numerical data (min, max, mean, standard deviation, quantiles)

For example, number of search results

Different value “reservoirs”, e.g.,

- Entire application lifetime

- Last N searches

- Last N minutes

Histograms

```
(def number-of-results
  (histogram metrics-registry "number-of-search-results"))

(defn search [query]
  (let [results (execute query)]
    (update! number-of-results (count results))
    results))
```


CS100i

Intelligent Counterpulsation

ALARM
MESSAGES

ADVISORIES

ECG

• LEAD

• GAIN

PRESSURE
SOURCE

IAB FILL MODE

SLOW GAS
ALARM

OPERATION
MODE

IAB STATUS

ECG

TRIGGER

HEART RATE
bpm

SYSTOLIC
mmHg

DIASTOLIC
mmHg

MEAN
mmHg

AUG.
mmHg

AUG. ALARM
mmHg

R-Wave Deflate

II
Normal

Direct

Auto

On

Auto

0.5 mV

140

80

20

Infl.

Auto

4.0

Defl.

97

64

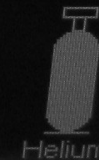
51

72

104

86

Meters



Helium

Datascope®

CE 0044

Meters

Rate of an event (per second)

Total count of events

Average rate over application lifetime

Rate in the last 1, 5 and 15 minutes

For example, incoming requests

Meters

```
(def inc-requests-meter  
  (meters/meter metrics-registry "incoming-requests-meter"))  
  
(defn requests-meter [handler]  
  (fn [req]  
    (meters/mark! inc-requests-meter)  
    (handler req)))
```




Timers

Timers

Histogram & meter

Histogram of duration of an activity, meter of occurrence of the activity

For example, specific type of database query

Timers

```
(def inc-requests-timer
  (timers/timer! metrics-registry "incoming-requests-timer"))

(defn requests-timer [handler]
  (fn [req]
    (timers/time! inc-requests-timer
      (handler req))))
```

Reporting Metrics

Reporting

Metrics stores current state in its registry

Should be reported to external tool

- Visualization

- Historical data

Reporters for console, JMX, Ganglia, Graphite, CSV etc. included

Conclusion

Organize around business capabilities

Rules & conventions needed

Standardize interfaces, logging and configuration

Different solutions to improve stability

Monitor your systems

Good support in Clojure



Example Microservices

Simple Calendar: <https://github.com/innoq/simple-calendar>

Simple Contacts: <https://github.com/innoq/simple-contacts>

Thank you! Questions?

<https://www.innoq.com/en/talks/2016/06/microservices-clojure/>

