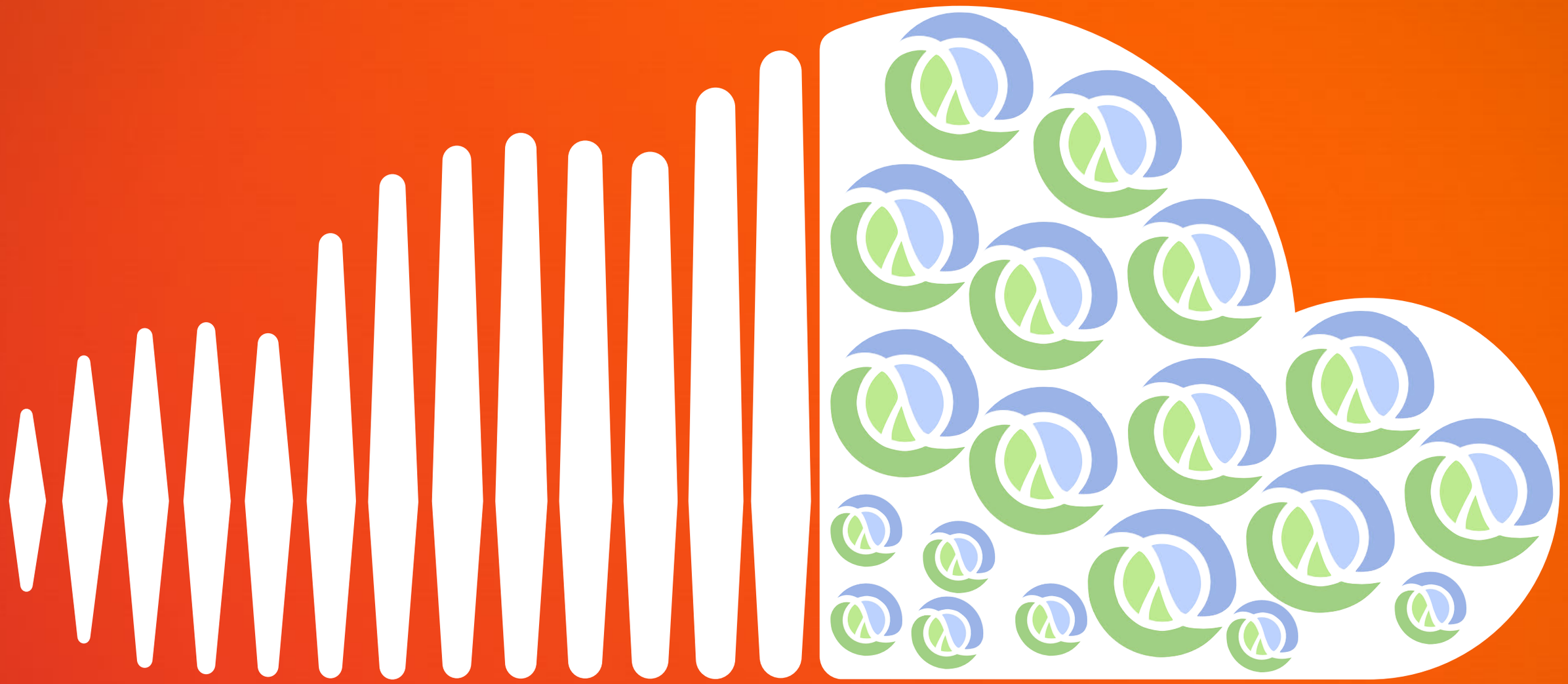
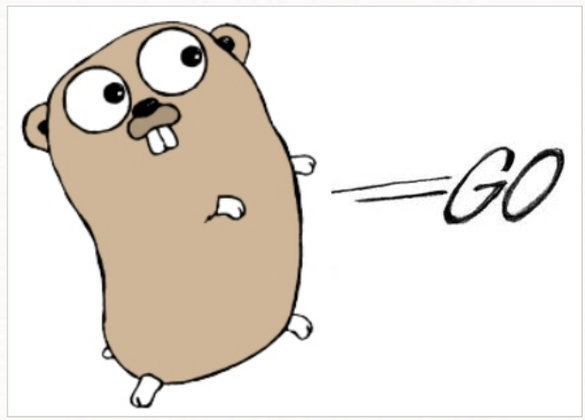


SOUNDCLOUD

Joseph Wilk

The Clojure Inside



Scala

Why Clojure?


It was there!
Functional
Immutable
Simple
LISP
Concurrency

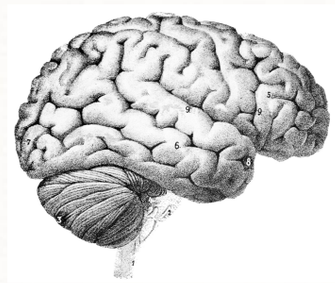

It was there!
Functional
Immutable
Simple
LISP
Concurrency

Programming Language

Computer

LISP Hurts

(Pick languages that challenge your mental model)

(())

```clojure
(defn filter
  "Returns a lazy sequence of the items in coll for which
  (pred item) returns true. pred must be free of side-effects."
  {:added "1.0"
   :static true}
  ([pred coll]
   (lazy-seq
    (when-let [s (seq coll)]
      (if (chunked-seq? s)
        (let [c (chunk-first s)
              size (count c)
              b (chunk-buffer size)]
          (dotimes [i size]
              (when (pred (.nth c i))
                (chunk-append b (.nth c i))))
          (chunk-cons (chunk b) (filter pred (chunk-rest s))))
        (let [f (first s) r (rest s)]
          (if (pred f)
            (cons f (filter pred r))
            (filter pred r))))))))
```

Code generation

```
(defmacro doseq
  "Repeatedly executes body (presumably for side-effects) with
  bindings and filtering as provided by \"for\".  Does not retain
  the head of the sequence. Returns nil."
  {:added "1.0"}
  [seq-exprs & body]
  (assert-args
     (vector? seq-exprs) "a vector for its binding"
     (even? (count seq-exprs)) "an even number of forms in binding vector")
  (let [step (fn step [recform exprs]
               (if-not exprs
                 [true `(do ~@body)]
                 (let [k (first exprs)
                       v (second exprs)]
                   (if (keyword? k)
                     (let [steppair (step recform (nnext exprs))
                           needrec (steppair 0)
                           subform (steppair 1)]
                       (cond
                         (= k :let) [needrec `(let ~v ~subform)]
                         (= k :while) [false `(when ~v
                                                ~subform
                                                ~@(when needrec [recform]))]
                         (= k :when) [false `(if ~v
                                               (do
                                                 ~subform
                                                 ~@(when needrec [recform]))
                                               ~recform)]))
                     (let [seq- (gensym "seq_")
                           chunk- (with-meta (gensym "chunk_")
                                             {:tag 'clojure.lang.IChunk})
                           count- (gensym "count_")
                           i- (gensym "i_")
                           recform `(recur (next ~seq-) nil 0 0)
                           steppair (step recform (nnext exprs))
                           needrec (steppair 0)
                           subform (steppair 1)
                           recform-chunk
                             `(recur ~seq- ~chunk- ~count- (unchecked-inc ~i-))
```

Functions what do functions

wee wee wee wee wee wee wee wee wee

Code is data
Data is code

Variable generation

Is that a class?

OH MY ITS FAST

(Jetty/Netty but sssssh)

(Shhh confirmation Bias)

CPU usage	Memory
4 %	150 MB
4 %	150 MB
4 %	150 MB
4 %	150 MB
4 %	170 MB
4 %	150 MB
4 %	150 MB
4 %	160 MB
4 %	150 MB
5 %	140 MB

History

Rails

Me

Rails

Me

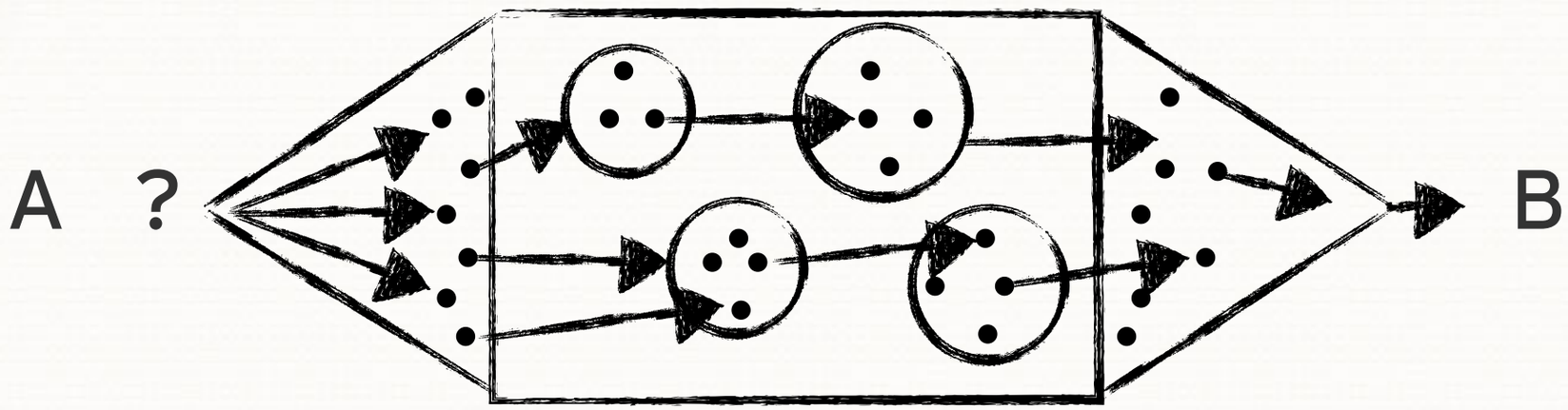

Ruby

Small

Replaceable

Pieces

Experiment



Opening
(Divergent)

Exploring
(Emergent)

Closing
(Convergent)

Clojure

Hystrix	https://github.com/Netflix/Hystrix
Turbine	https://github.com/Netflix/Turbine
RxJava	https://github.com/Netflix/RxJava
Netty	http://netty.io
Aleph	https://github.com/ztellman/aleph
Compojure	https://github.com/weavejester/compojure
Ring	https://github.com/ring-clojure/ring
Midje	https://github.com/marick/Midje

Maturity + Brevity & Denial (its not Java)

cons

quote cdr

atom eq

label car

apply cond

lambda

~~Java~~

Denial

```
(defmacro)
```

```
HttpServer server = HttpServer.create(address, 0)
server.createContext(path, handler);
server.setExecutor(null)
server.start();
```

```
(doto (HttpServer/create address 0)
  (.createContext path handler)
  (.setExecutor nil)
  (.start))
```

```
HttpServer server = HttpServer.create(address, 0)
server.createContext(path, handler);
server.setExecutor(null)
server.start();
```

```
(doto (HttpServer/create address 0)
  (.createContext path handler)
  (.setExecutor nil)
  (.start))
```

```
public class GetSoundCmd extends HystrixCommand<Sound> {
  private final HttpClient client;
  private final String urn;

  public FindSoundCmd(HttpClient client, String urn){
    this.client = client;
    this.urn = urn;
  }

  @Override
  protected Sound run(){
    return client.get("/sounds/" + urn, Sound.class)
  }

  @Override
  protected Sound getFallback(){
    return Sound.missing(id);
  }
}
new GetUserCmd(client, id).execute();
```

```
(:require [com.netflix.hystrix.core :as hystrix])

(hystrix/defcommand find-sound
  {:hystrix/fallback-fn #(constantly {})}
  [client urn]
  (http/get urn))

(get-sound client "soundcloud:sounds:123")
```

```
public class GetSoundCmd extends HystrixCommand<Sound> {
  private final HttpClient client;
  private final String urn;

  public FindSoundCmd(HttpClient client, String urn){
    this.client = client;
    this.urn = urn;
  }

  @Override
  protected Sound run(){
    return client.get("/sounds/" + urn, Sound.class)
  }

  @Override
  protected Sound getFallback(){
    return Sound.missing(id);
  }
}
new GetUserCmd(client, id).execute();
```

```
(:require [com.netflix.hystrix.core :as hystrix])

(hystrix/defcommand find-sound
  {:hystrix/fallback-fn #(constantly {})}
  [client urn]
  (http/get urn))

(get-sound client "soundcloud:sounds:123")
```

Interactivity

R read
E evaluate
P print
L loop

```
                :link             link
                :background_color background-color}]
    (into {} (remove value-nil? visual))))

(defn- timed-visuals [visuals {entry-time :entry_time :as visual}]
  (assoc visuals entry-time (representation-for-one visual)))

(defn- visuals-representation [visuals]
  {:visuals (reduce timed-visuals {} visuals)})

(defn- urn-for [coll]
  (:urn (first coll)))

(defn- urn-representation [visuals]
  {:urn (urn-for visuals)})

(defn tracking-representation [visuals]
  (if-let [tracking (:impression (or (first visuals) {}))]
    {:tracking {:impression tracking}}
    {}))

(defn representation [visuals]
  (if (seq visuals)
    (merge (visuals-representation visuals)
           (urn-representation visuals)
           (tracking-representation visuals))
    []))

(defn admin-representation [visuals]
```

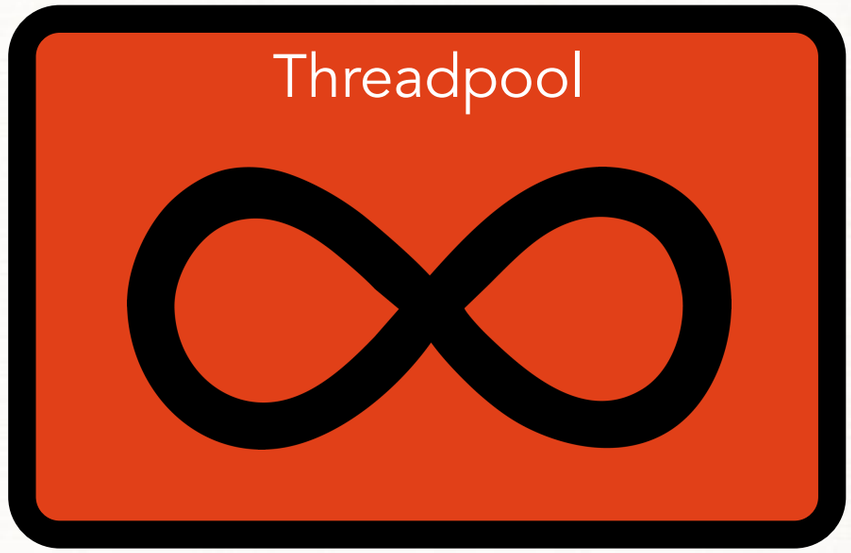
Services at scale

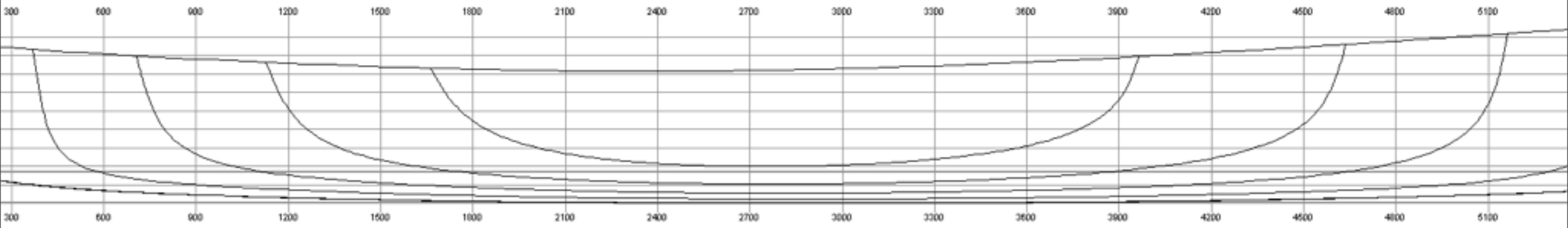

NETFLIX

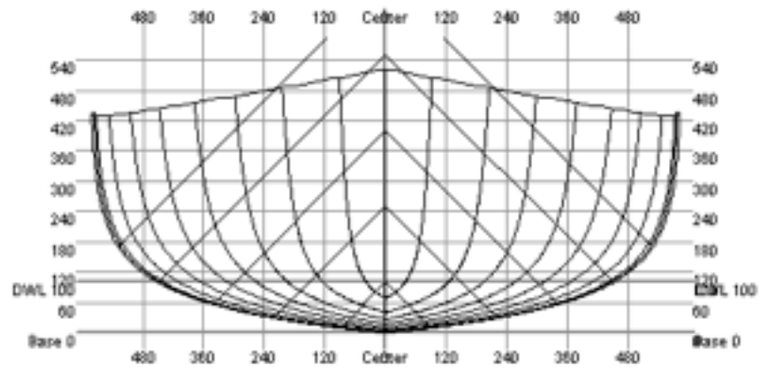
Grace in the face of failure

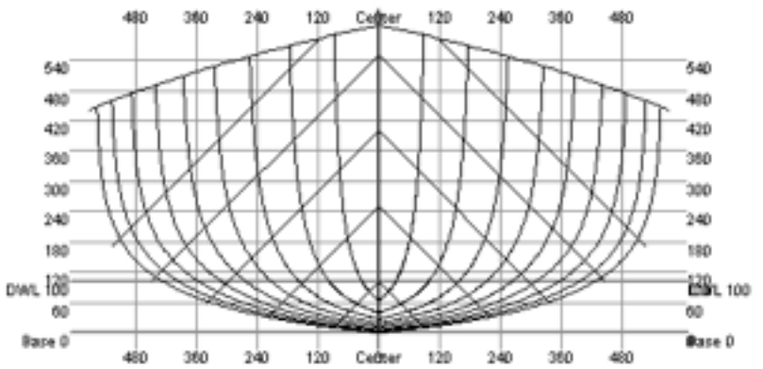
high **partition tolerance** in a **high latency cloud network**
(don't trust anything works)

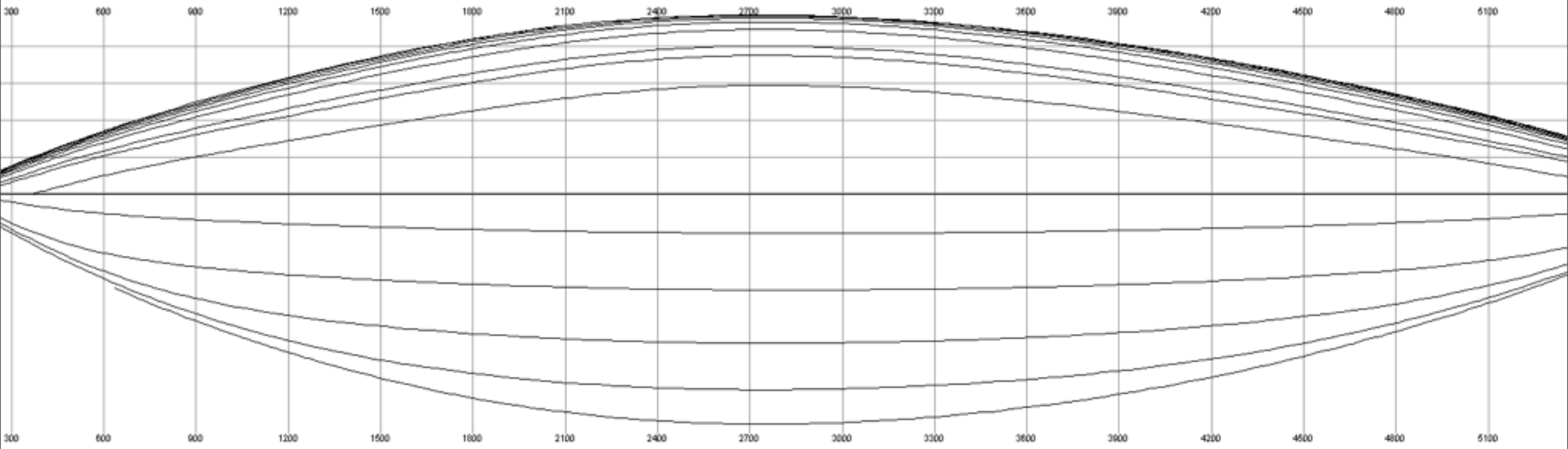
```clojure
(future (fn [] "from the future"))

(promise)

(agent)
```

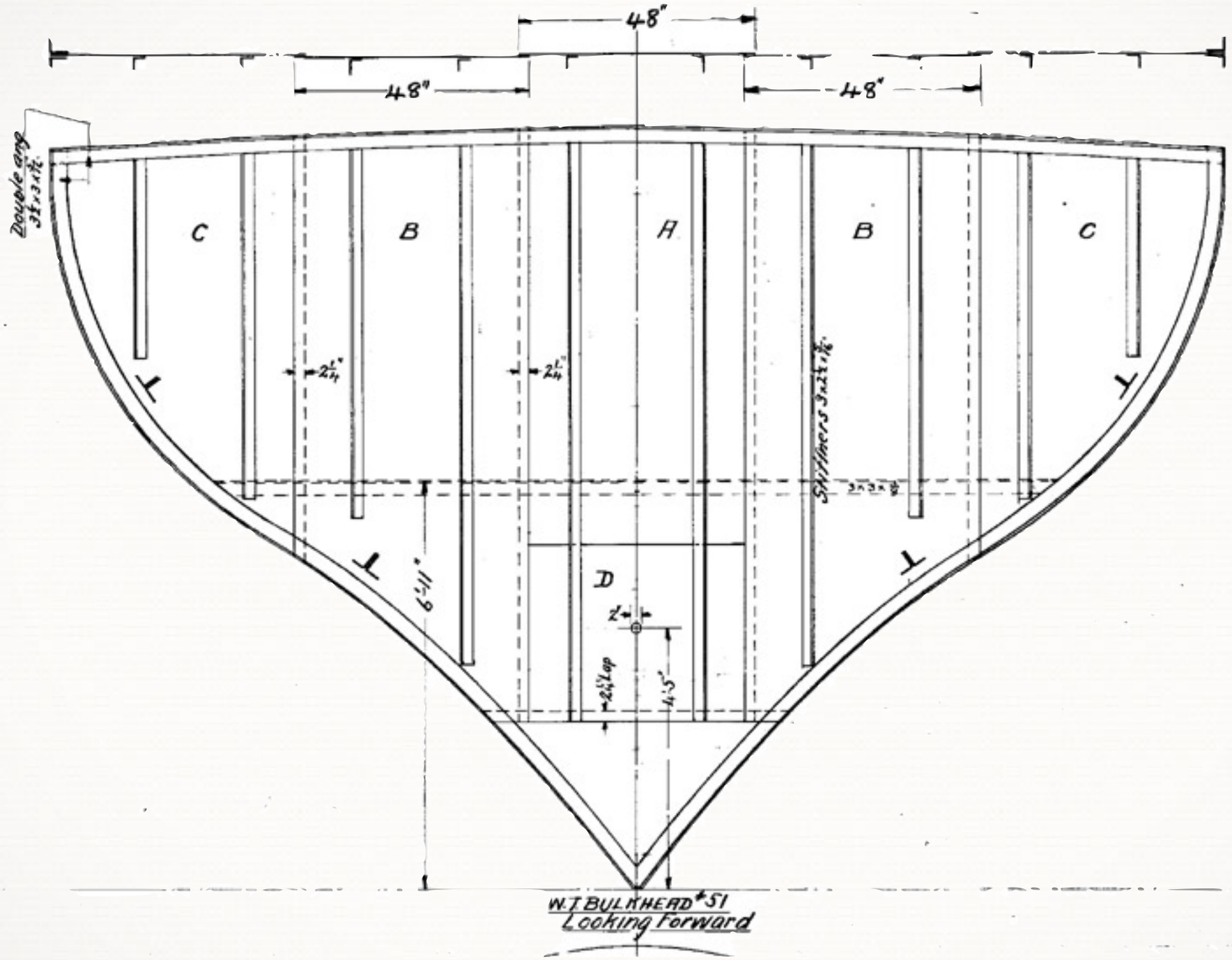


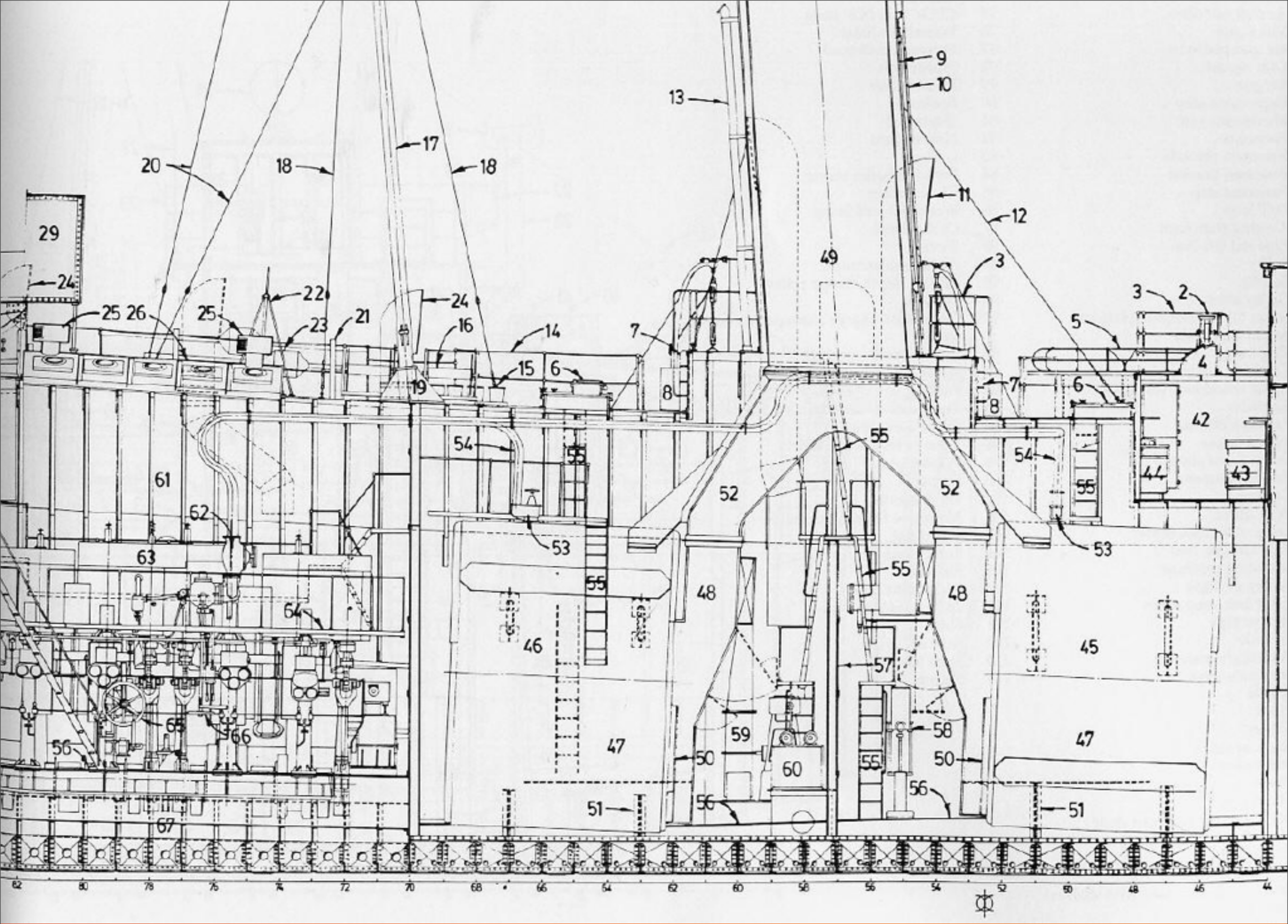

300
600
900
1200
1500
1800
2100
2400
2700
3000
3300
3600
3900
4200
4500
4800
5100


480
360
240
120
Center
540
480
420
360
300
240
180
120
DWL 100
60
Base 0


480
360
240
120
Center
540
480
420
360
300
240
180
120
DWL 100
60
Base 0


300
600
900
1200
1500
1800
2100
2400
2700
3000
3300
3600
3900
4200
4500
4800
5100


48"
48"
48"
Double angs 3½x3½"
C
B
A
B
C
Y
Y
Y
2½"
2½"
Shifters 3x2½x⅜"
3x3x⅜
6'-11"
D
Z
2½ Leg
4'-3"
W.T. BULKHEAD #51
Looking Forward


9
10
13
17
20
18
18
11
12
49
3
29
24
22
24
25 26
25
23
21
16
14
7
15
6
19
8
3
2
5
4
7
6
8
42
54
55
61
62
63
64
65
66
56
67
53
55
52
52
55
53
48
48
46
45
57
59
58
47
47
50
60
55
50
56
56
51
51
44
43
55
62
60
58
56
54
52
50
48
46
44

Bulkhead

Threadpool

100 threads

Threadpool

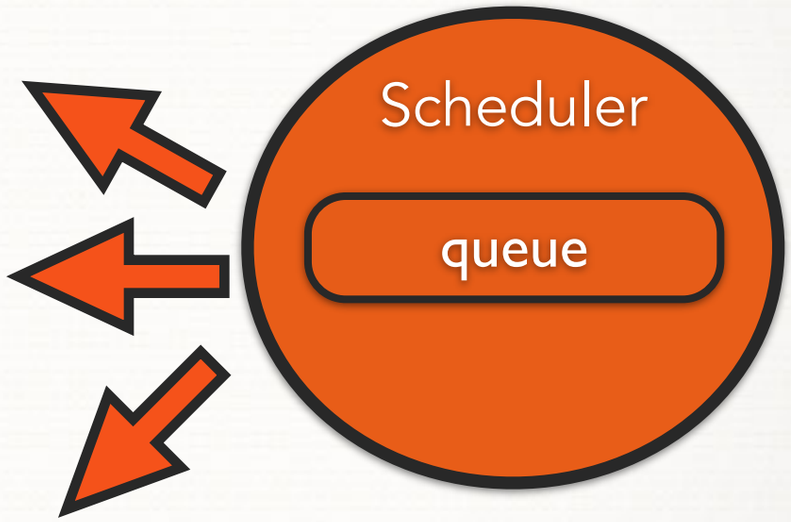
10 threads

Threadpool

10 threads

Threadpool

30 threads



```clojure
(defn find-user [user-urn]
  (let [response (get-user user-urn)]
    (if (= 200 (:status response))
      (:body response)
      {})))
```

```clojure
(hystrix/defcommand find-user
  {:hystrix/fallback-fn (constantly {})}
  [user-urn]
  (let [response (get-user user-urn)]
    (if (= 200 (:status response))
      (:body response)
      {})))
```

```clojure
(find-user 123)

(hystrix/queue #‘find-user 123)
```

```clojure
(defn find-user [user-urn]
  (let [response (get-user user-urn)]
    (if (= 200 (:status response))
      (:body response)
      {})))
```

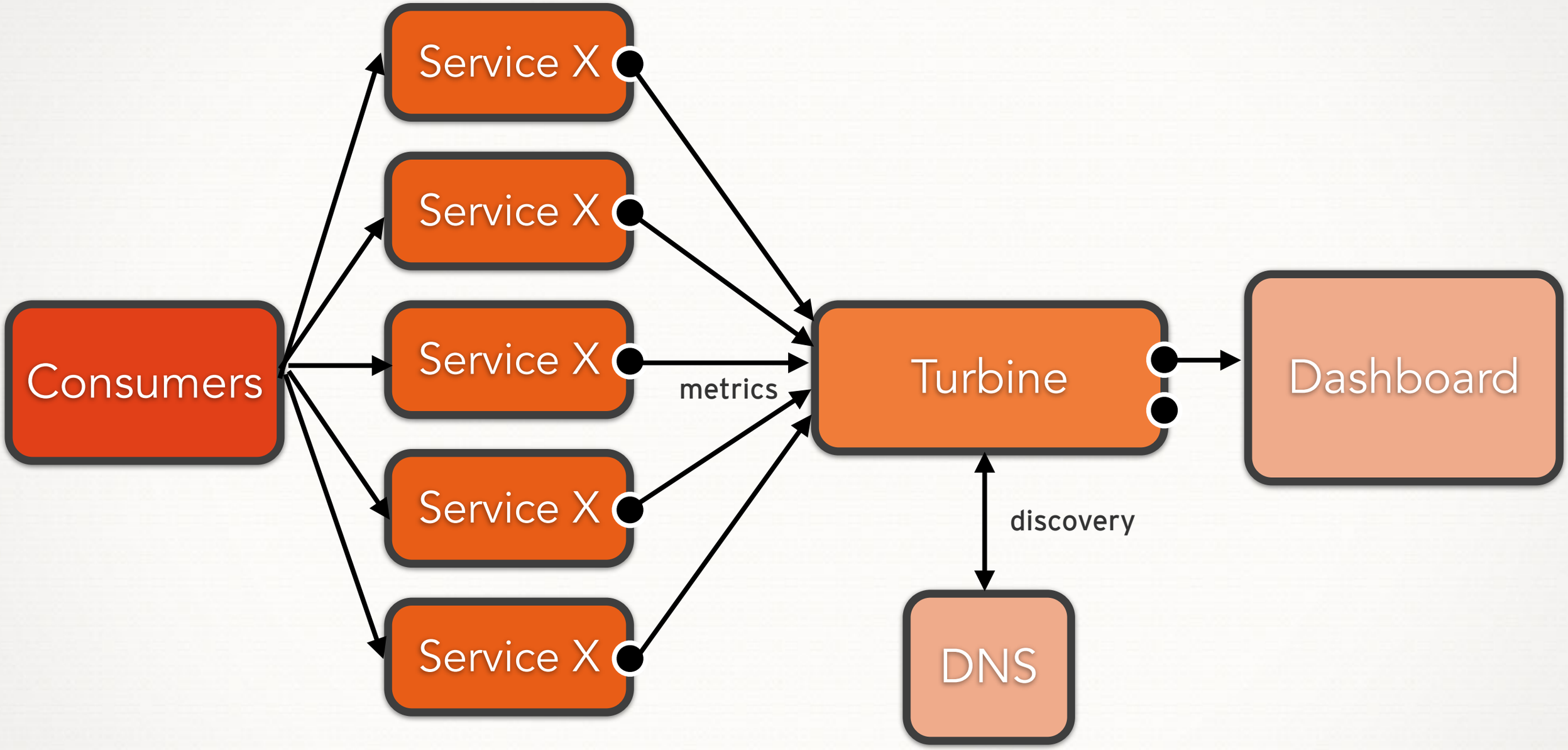
```clojure
(hystrix/defcommand find-user
  {:hystrix/fallback-fn (constantly {})}
  [user-urn]
  (let [response (get-user user-urn)]
    (if (= 200 (:status response))
      (:body response)
      {})))
```

```clojure
(find-user 123)

(hystrix/queue #'find-user 123)
```

```clojure
(defn find-user [user-urn]
  (let [response (get-user user-urn)]
    (if (= 200 (:status response))
      (:body response)
      {})))
```

```clojure
(hystrix/defcommand find-user
  {:hystrix/fallback-fn (constantly {})}
  [user-urn]
  (let [response (get-user user-urn)]
    (if (= 200 (:status response))
      (:body response)
      {})))
```

```clojure
(find-user 123)

(hystrix/queue #'find-user 123)
```


Service X
Service X
Service X
Service X
Service X
Consumers
metrics
Turbine
Dashboard
discovery
DNS

https://github.com/Netflix/Hystrix

The Sound of SoundCloud inside SoundCloud by Joseph Wilk on SoundCloud - Hear the world's sounds

https soundcloud.com/josephwilk/the-sound-of-soundcloud-inside

Reader

GitHub SK Live Cuke Gist Ruby QuickRef Ruby-Doc Rails Rdoc Rspec Cucumber lighthouse rspec / cucumber Events jQuery

Stream Explore Go Pro Upload The Sound of SoundCl... More

Joseph Wilk

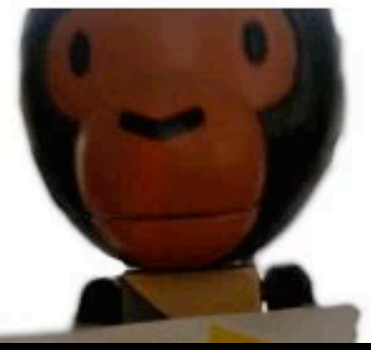
1 month

The Sound of SoundCloud inside SoundCloud

Write a comment ...

Add to playlist Add to group Share

824 3 2 More stats



• AI

Takes live Hystrix metrics across a pool of internal services and normalises them to piano pitches. Played as the data comes in.

Joseph Wilk

41 | 10

Related

Compositional Complexity

a

b

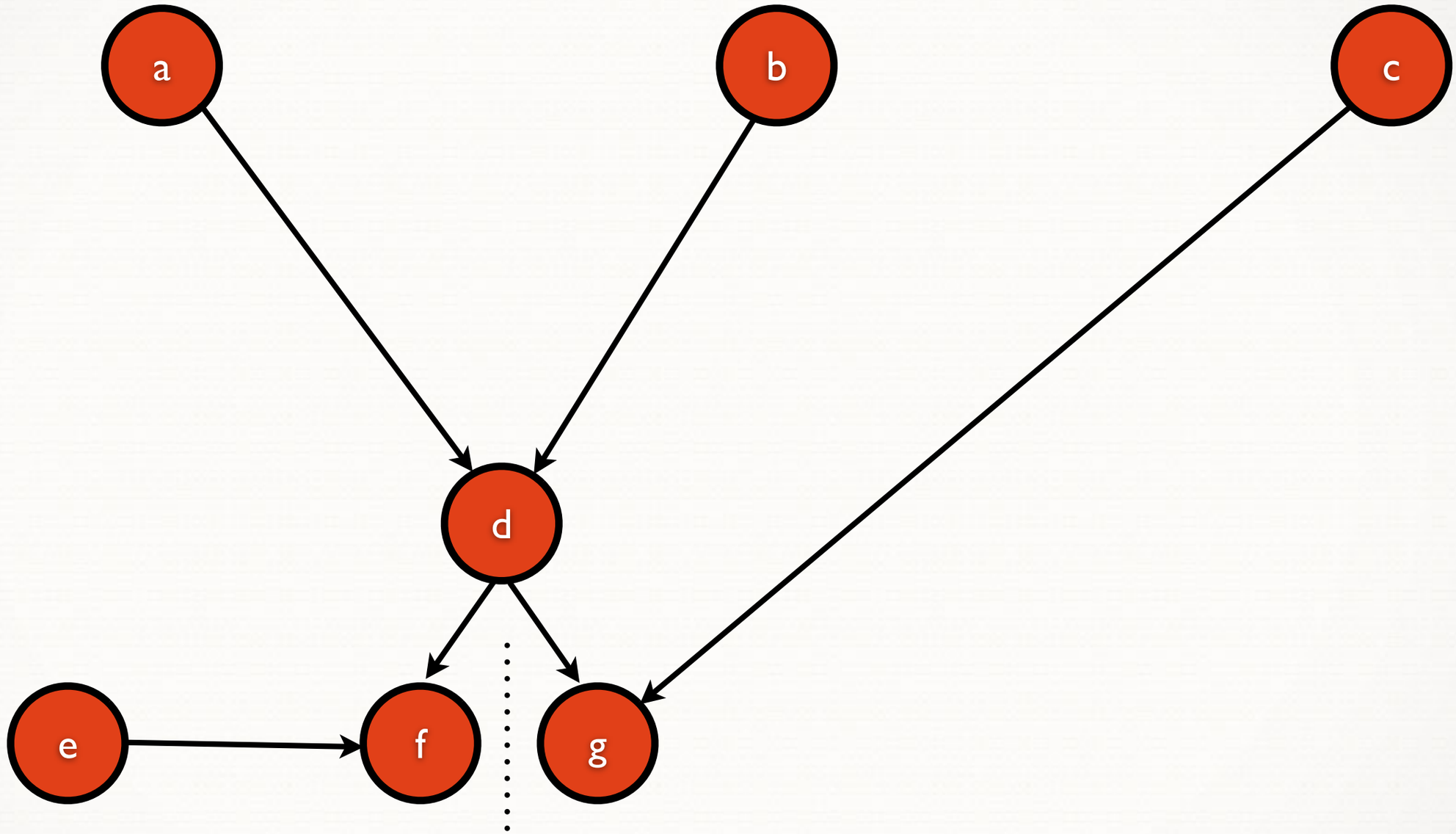
c

d

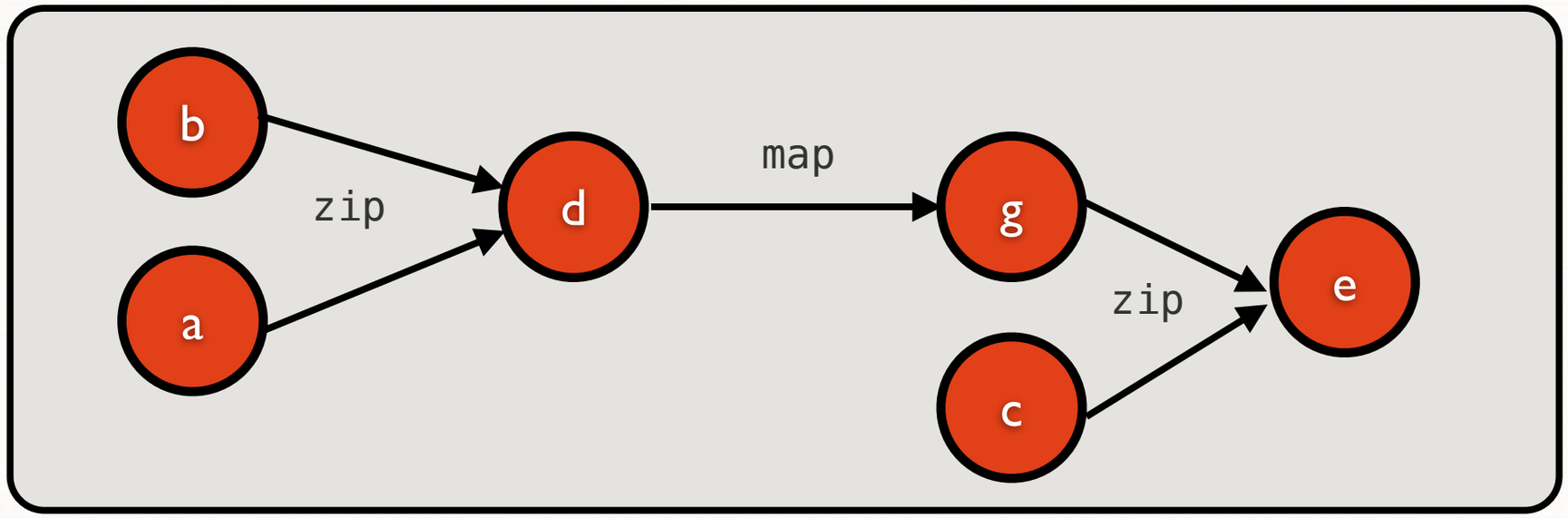
e

f

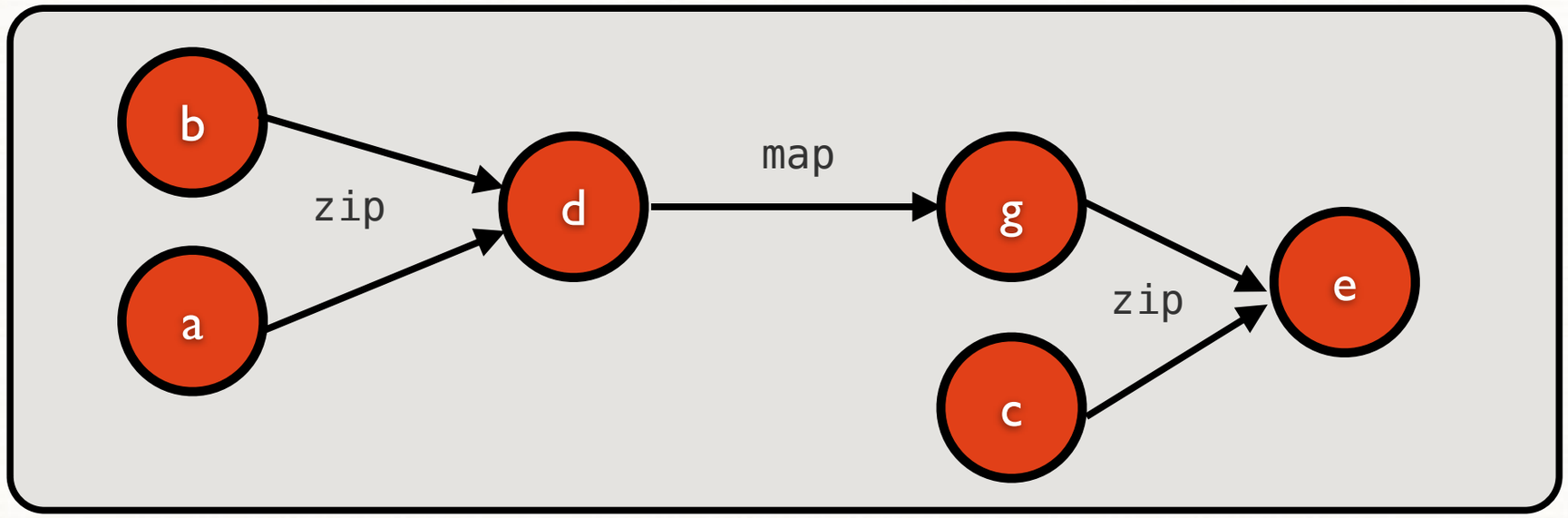
g


a
b
c
d
e
f
g

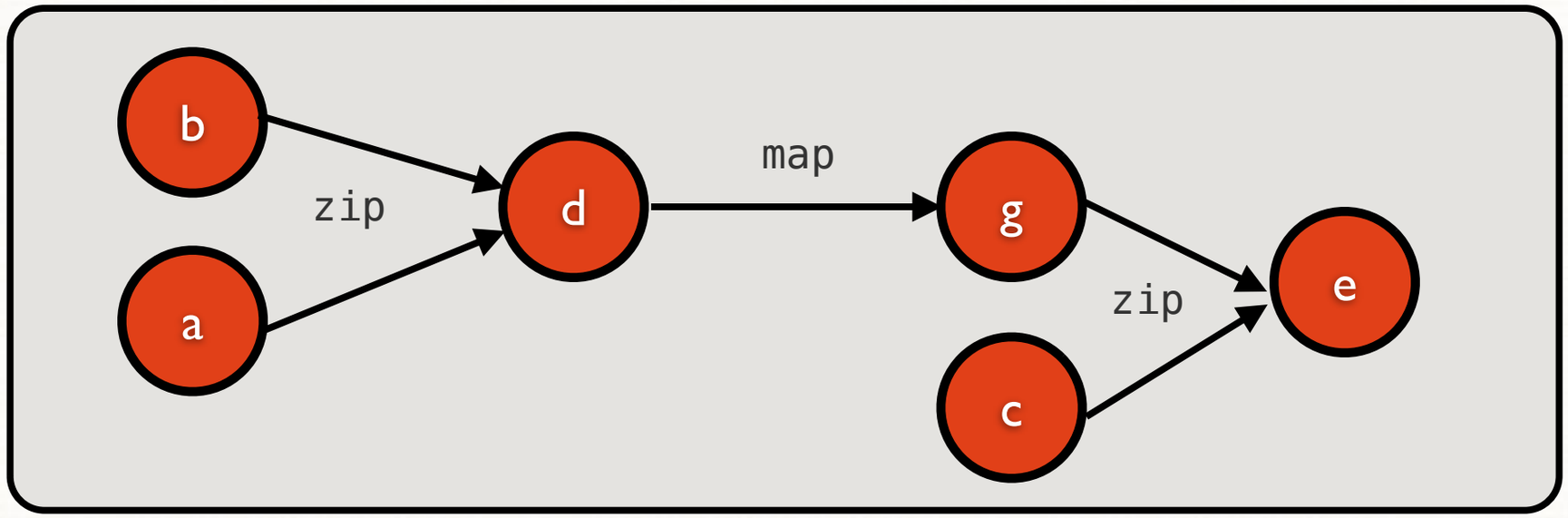
List Processing



List Processing

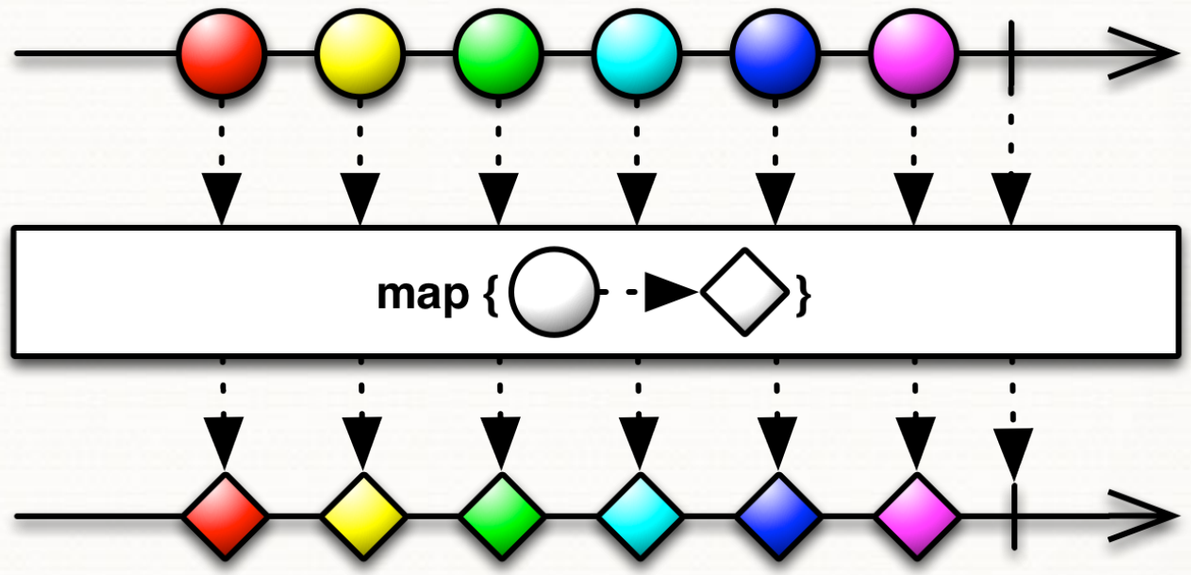


LisP



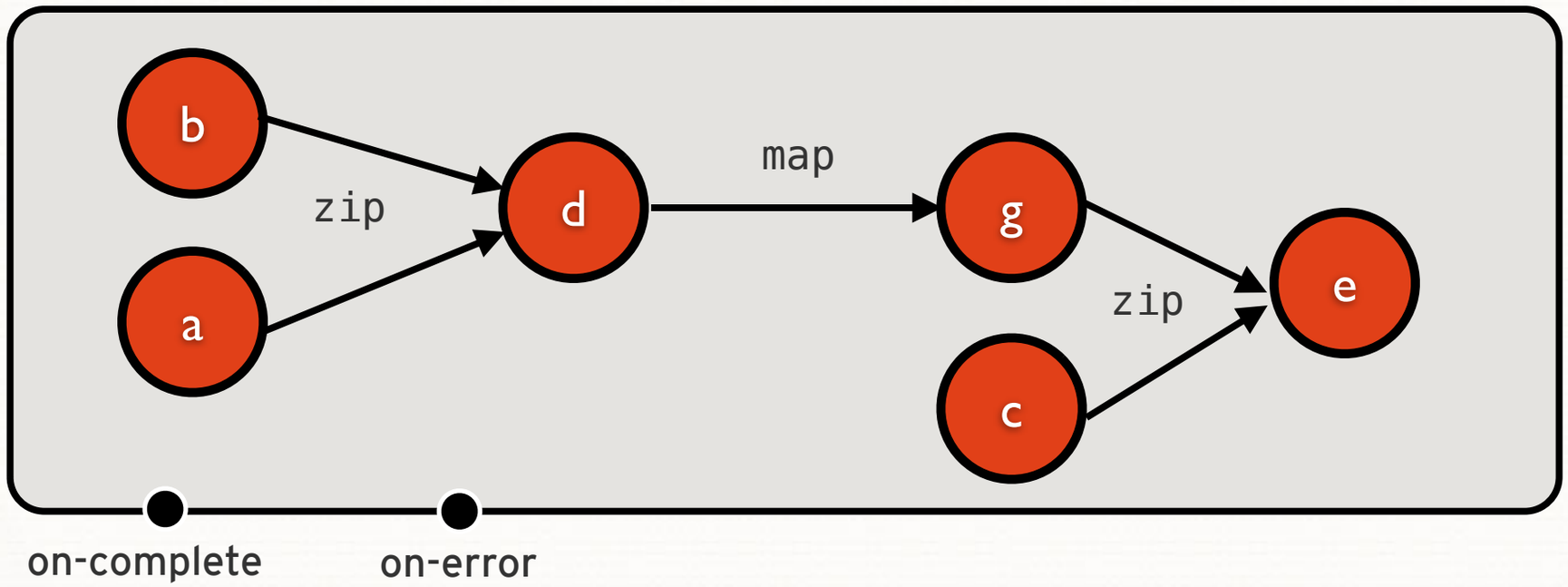
RxJava

(map)
(zip)
(filter)
(partition)
(reduce)



https://github.com/Netflix/RxJava

Ending Callback Hell




```
(.subscribe
  (fn [result] (enqueue channel {:status 200 :body (json/encode result})
  (fn [error]  (log/exception error request))))))
```

Request & Response

```
(.subscribe
  (fn [result] (enqueue channel {:status 200 :body (json/encode result}})
  (fn [error]  (log/exception error request))))))
```

```
(defn handler [request] {:status 200 :body "Hello World"})
```

```
(use 'lamina.core)

(defn handler [response-channel request]
  (enqueue response-channel
    {:status 200 :body "Hello World"}))
```

https://github.com/ztellman/aleph

Request & Response

```
(.subscribe
  (fn [result] (enqueue channel {:status 200 :body (json/encode result})
  (fn [error]  (log/exception error request)))))
```

```
(defn handler [request] {:status 200 :body "Hello World"})
```

```
(use 'lamina.core)

(defn handler [response-channel request]
  (enqueue response-channel
    {:status 200 :body "Hello World"}))
```

https://github.com/ztellman/aleph

Request & Response

```
(.subscribe
  (fn [result] (enqueue channel {:status 200 :body (json/encode result}))
  (fn [error]  (log/exception error request))))))
```

```
(defn handler [request] {:status 200 :body "Hello World"})
```

```
(use 'lamina.core)

(defn handler [response-channel request]
  (enqueue response-channel
    {:status 200 :body "Hello World"}))
```

https://github.com/ztellman/aleph

Request & Response

```
(.subscribe
  (fn [result] (enqueue channel {:status 200 :body (json/encode result})
  (fn [error]  (log/exception error request))))))
```

```
(defn handler [request] {:status 200 :body "Hello World"})
```

```
(use 'lamina.core)

(defn handler [response-channel request]
  (enqueue response-channel
    {:status 200 :body "Hello World"}))
```

https://github.com/ztellman/aleph

Its upside down

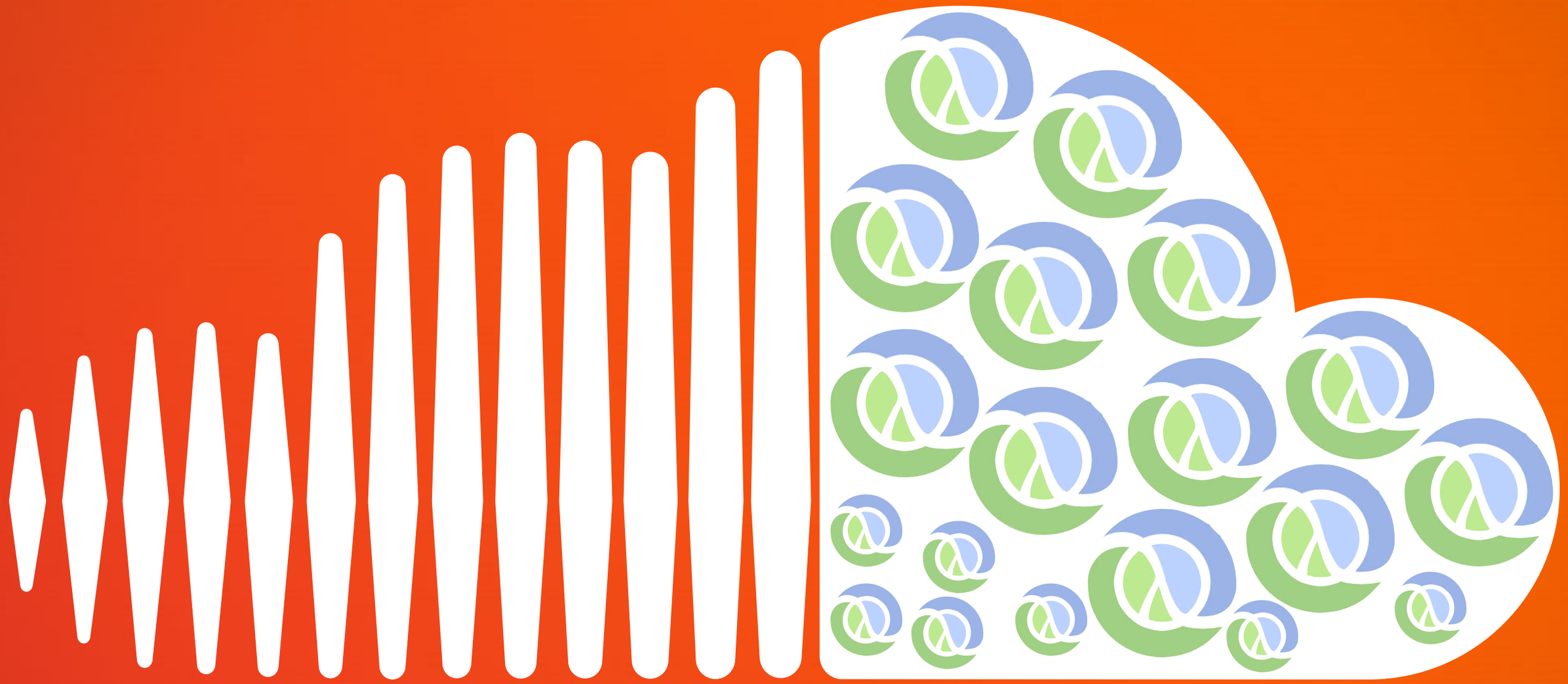
attern matching

Clojure Itches

parameter redeye

Corrupts your brain

One of the many paths



SOUNDCLOUD

Joseph Wilk