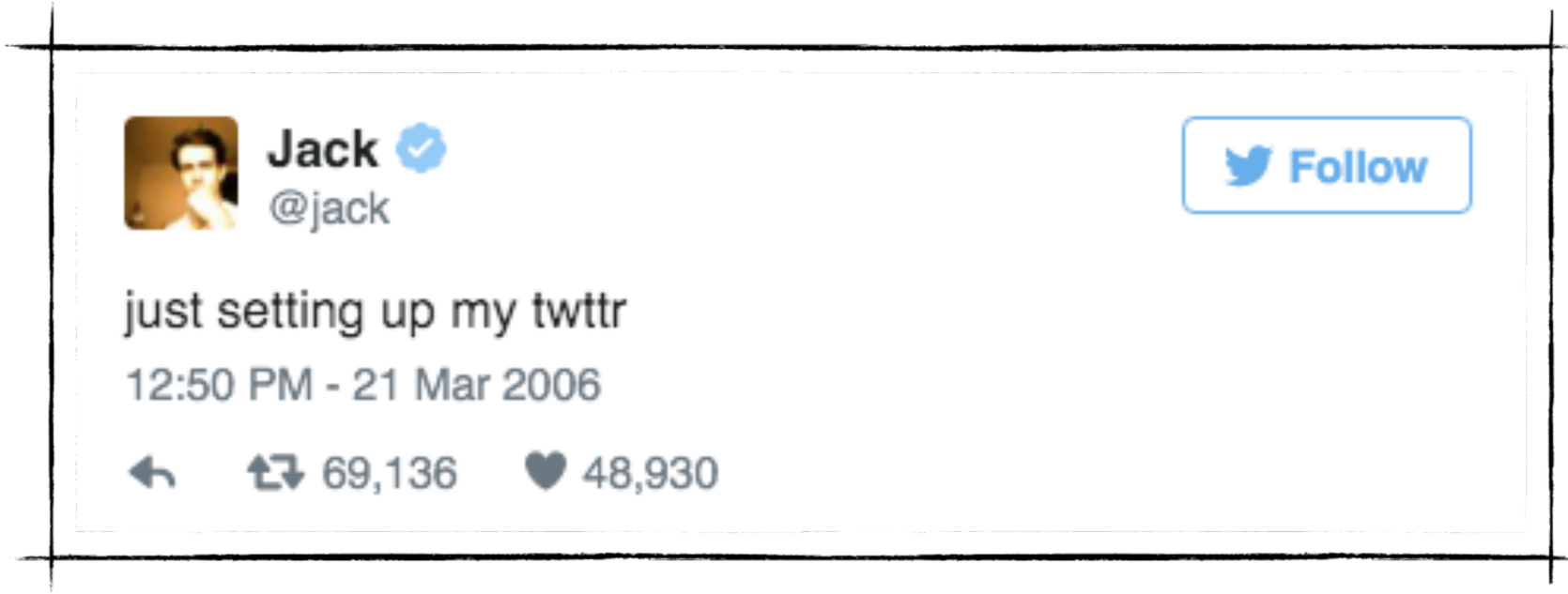
Reactive Programming with Elixir and RethinkDB

Peter Hamilton

Erlang Factory 2016

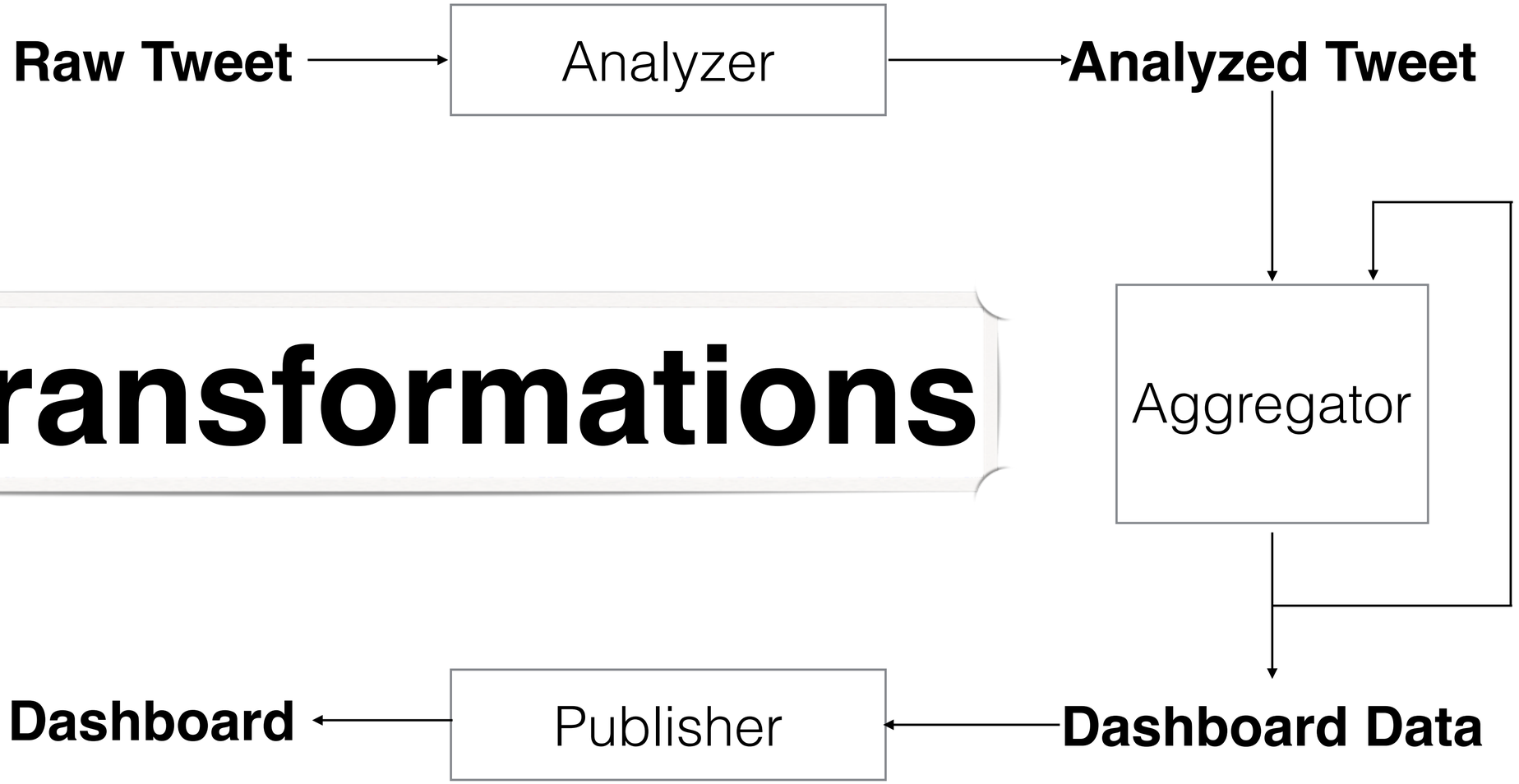
Let’s Build Something!

Which character do people tweet most?



The first tweet had 6 uses of ’t'

Data Transformations



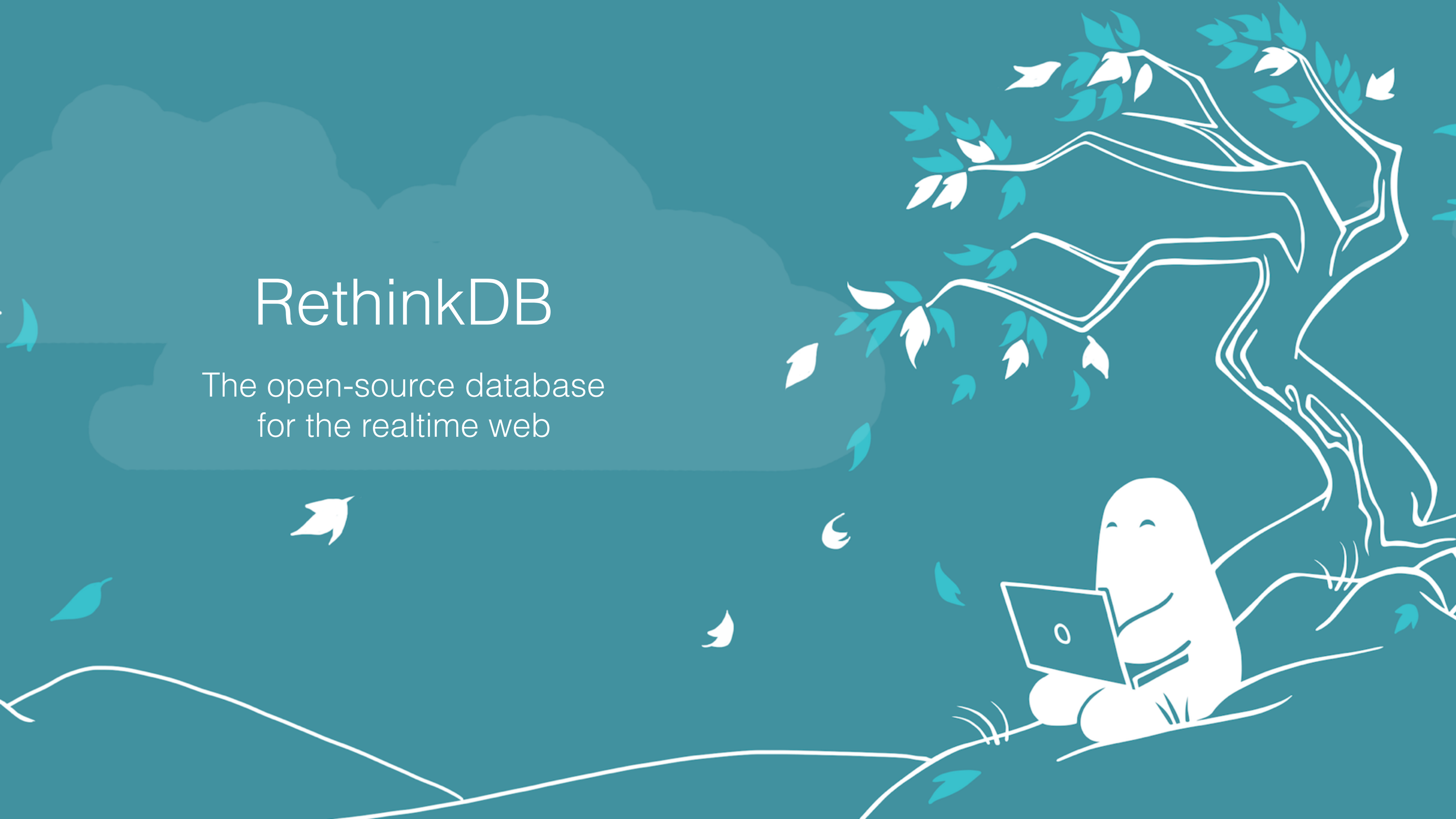
Elixir Stream

```
tweet_stream
  |> Stream.map(&analyze_tweet/1)
  |> Stream.scan(initial_dashboard, &update_dashboard/2)
  |> Stream.each(&broadcast/1)
  |> Stream.run
```

Problems?

RethinkDB

The open-source database
for the realtime web



ReQL

```
table(“people”)
  |> filter(%{name: “John”})

table(“people”)
  |> filter(lambda fn (person) ->
    person[:age] > count(person[:name])
  end)

table(“people”)
  |> merge(lambda fn (person) ->
    if (person[:age] < 13) do
      %{name: “PRIVATE”}
    else
      %{}
    end
  end)
```

More ReQL

```
# Get all posts, include author

table(“posts”)
  |> eq_join(“author_id”, table(“people”))
  |> zip

# Get people and any posts they might have written

table(“people”)
  |> merge(lambda fn (person) ->
    posts = table(“posts”)
      |> filter(%{author_id: person[:id]})
      |> coerce_to(:array)
    %{posts: posts}
  end)
```

Even More ReQL

```
# Count posts per author

table(“posts”)
  |> group(“author_id”)
  |> count

# Word count per author

table(“posts”)
  |> group(“author_id”)
  |> map(lambda &(count(&1[:post]))
  |> sum
```

The Most ReQL

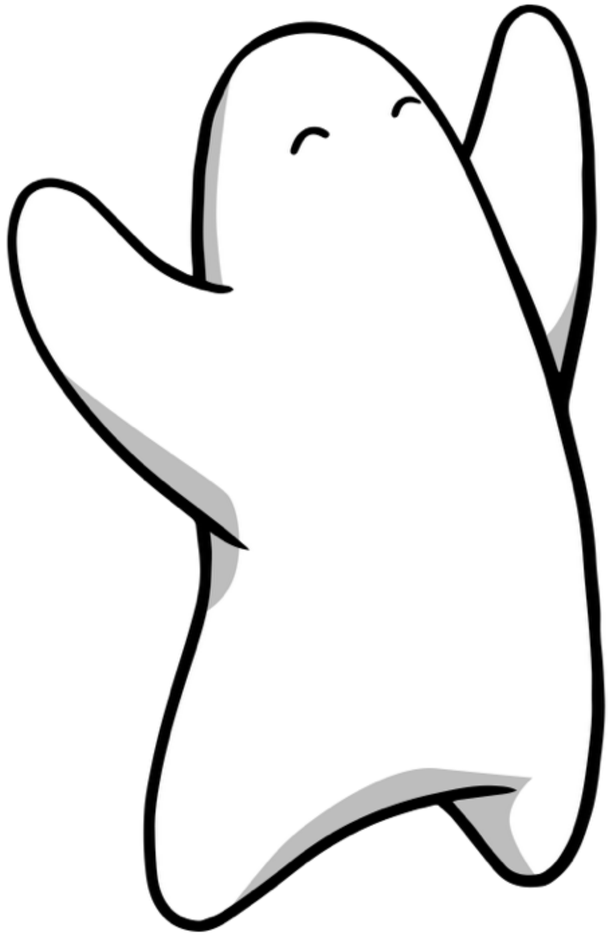
```
# Get counts of frequent/occasional authors (including staff)

table(“people”)
  |> group(lambda fn (person) ->
    post_count = table(“posts”)
      |> filter(%{author_id: person[:id]})
      |> count
    (post_count < 20)
      |> coerce_to(:string)
  end)
  |> map(lambda fn (author) ->
    1 + count(author[:staff])
  end)
  |> sum
  |> ungroup
  |> map(&values(&1))
  |> coerce_to(:object)
  |> map(lambda fn (obj) ->
    %{frequent: obj[“true”], occasional: obj[“false”]}
  end)
```

Limitations

ACID?

CAP?

Test results: aphyr.com and Jepson tests



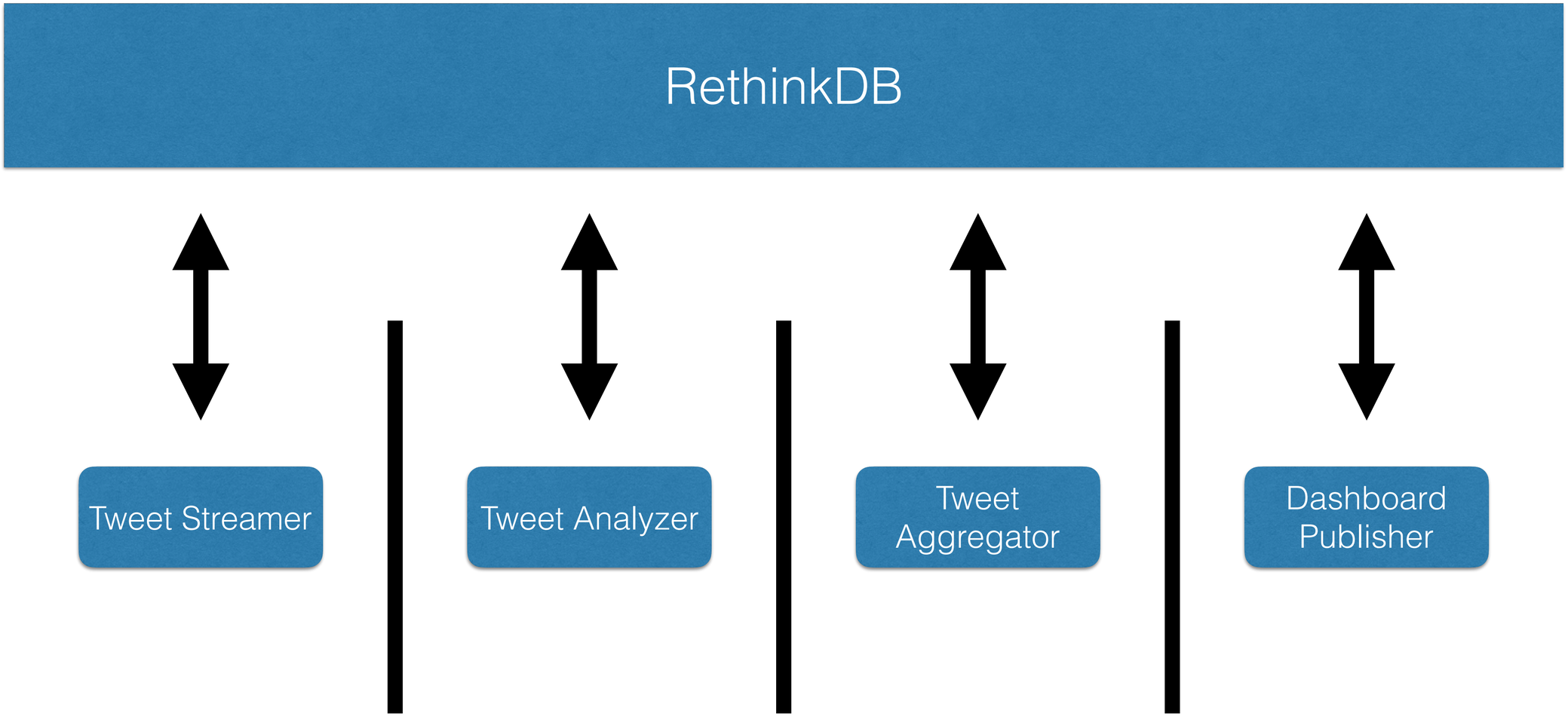
Let’s Use RethinkDB!

Elixir Stream

```
tweet_stream
  |> Stream.map(&analyze_tweet/1)
  |> Stream.scan(initial_dashboard, &update_dashboard/2)
  |> Stream.each(&broadcast/1)
  |> Stream.run
```

Our Original

Restructure with RethinkDB



RethinkDB.Changefeed

OTP Behaviour

```
use RethinkDB.Changefeed
```

Define the query, database and initial state

```
init(opts) :: {:subscribe, query, db, state} | {:stop, reason}
```

Process updates

```
handle_update(update, state) :: {:next, state} | {:stop, reason, state}
```

Plus GenServer callbacks

Tweet Streamer

```
def start_link(topic) do

    ExTwitter.stream_filter(track: topic)
      |> Stream.each(fn el ->

        table("tweets")
          |> insert(%{text: el.text, state: “RAW"})
          |> run

    end) |> StreamRunner.start_link
end
```

Tweet Analyzer (Part 1)

```
def init(_opts) do

    query = table("tweets")
      |> filter(%{state: "RAW"})
      |> changes(include_initial: true)

    {:subscribe, query, DB, nil}

end
```

Tweet Analyzer (Part 2)

```
def handle_update(data, _state) do
    Enum.each(fn
        %{“new_val” => el, “old_val” => nil} ->
            most_used_char = get_most_used_char(el)
            table(“tweets”)
              |> get(el[“id”])
              |> update(lambda fn (tweet) ->
                if (tweet[“state”] == “RAW”) do
                  %{state: “PROCESSED”, most_used_char: most_used_char}
                else
                  %{}
                end
              end) |> run
    end)
    {:next, nil}
end
```

Tweet Aggregator (Part 1)

```
def init(_opts) do

    query = table("tweets")
      |> filter(%{state: "PROCESSED"})
      |> changes(include_initial: true)

    {:subscribe, query, DB, nil}

end
```

Tweet Aggregator (Part 2)

```
def handle_update(data, _state) do
    Enum.each(data, fn
        %{“new_val” => el, “old_val” => nil} ->
            update_tweet_state(“PROCESSED”, “RECORDED”)
            char = el[“most_used_char”]
            table(“dashboard”)
              |> get(“character_dashboard”)
              |> update(lambda fn (dashboard) ->
                %{char => default(dashboard[char], 0) +1}
            end) |> run
    end)
    {:next, nil}
end
```

Dashboard Publisher (Part 1)

```
def init(_opts) do

    query = table("dashboards")
      |> get(“character_dashboard”)
      |> changes(include_initial: true, squash: true)

    {:subscribe, query, DB, nil}
end
```

Dashboard Publisher (Part 2)

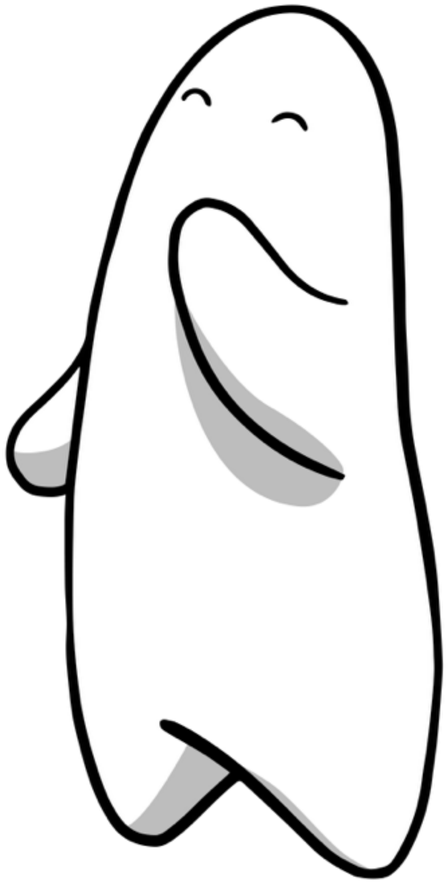
```
def handle_update(%{"new_val" => val}, _state) do

    process_dashboard(val)
    |> publish_dashboard

    {:next, nil}
end
```

Is it better?

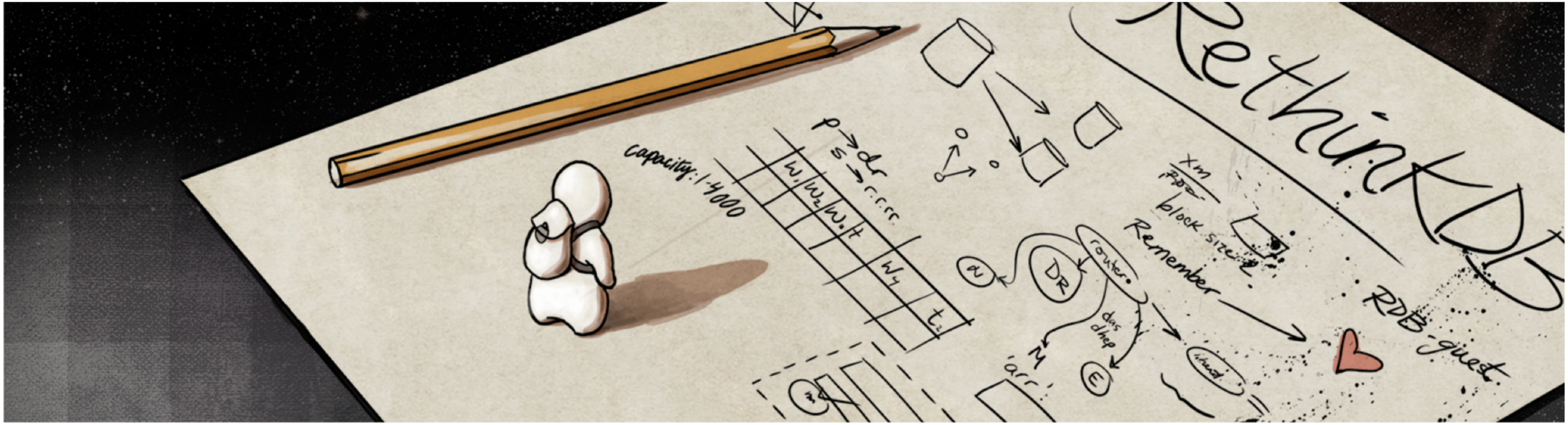
Pros

Is it better?

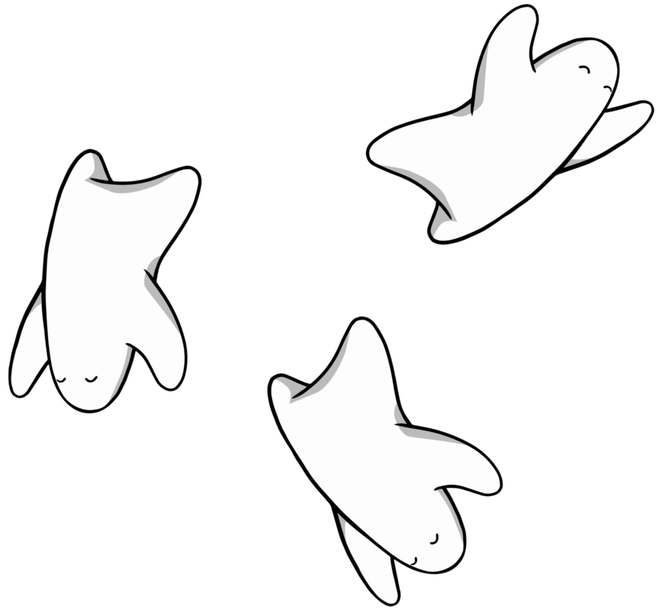
Cons



Rethinking Your Designs



Thank you!



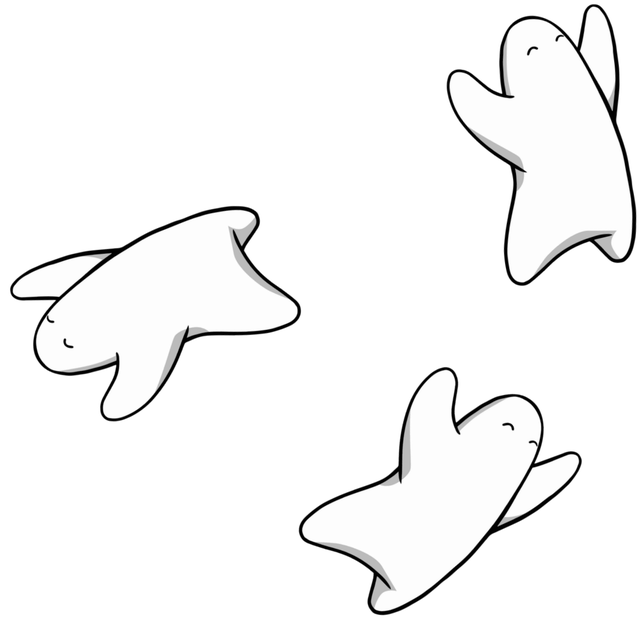
Please reach out with any questions!

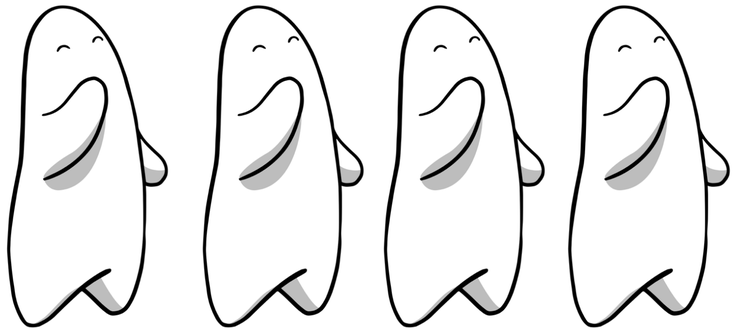
github: **hamiltop/rethinkdb-elixir**

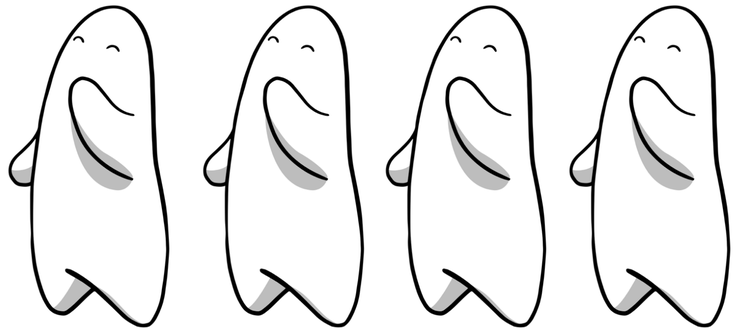
slack: **#rethinkdb** on elixir-lang

email: **peterghamilton@gmail.com**

twitter: **hamiltop**







Special thanks to **Annie Ruygt** at **RethinkDB** for the artwork!