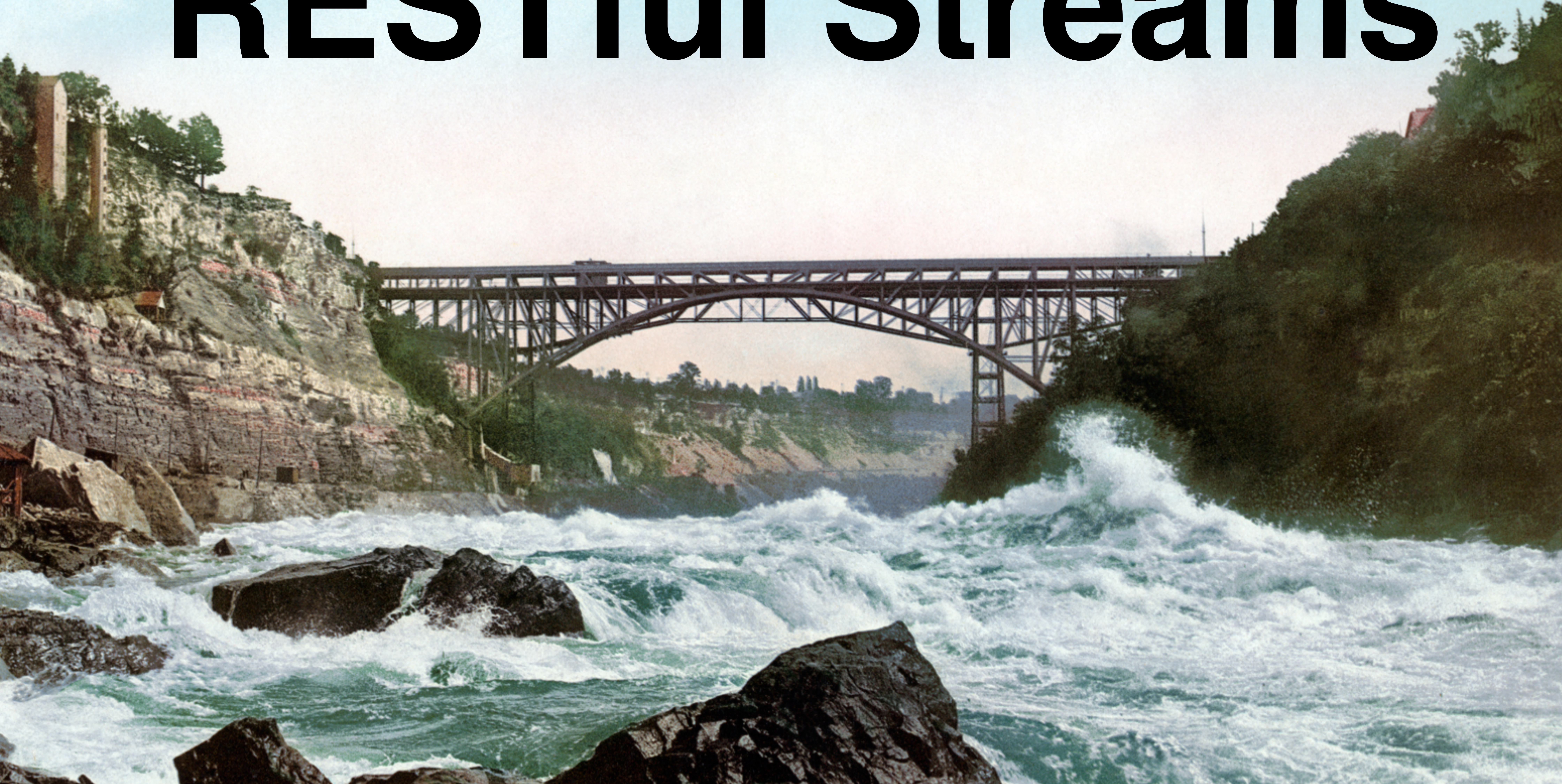
Building realtime applications with

RESTful Streams



An approach to building **realtime** web apps

2007: Rails 1.2

REST

*(mind **blown**)

RESTful Rails made for a **clean** design pattern that was **easier to test**, **secure**, and consume as an **API**

Sensible, lightweight Javascript libraries like **Backbone.js** and **Ember.js** hit the ground that **play nice with RESTful backends**

```
//Pretty simple stuff...
var user = new User();
user.fetch('/users/1.json');
```

HTTP Long Polling

```
// Poll every 10 seconds to keep
the channel model up-to-date.
setInterval(function() {
  user.fetch();
}, 10000);*
```

*As seen in the Backbone documentation

It *is* **simple**

Pile on the caching!

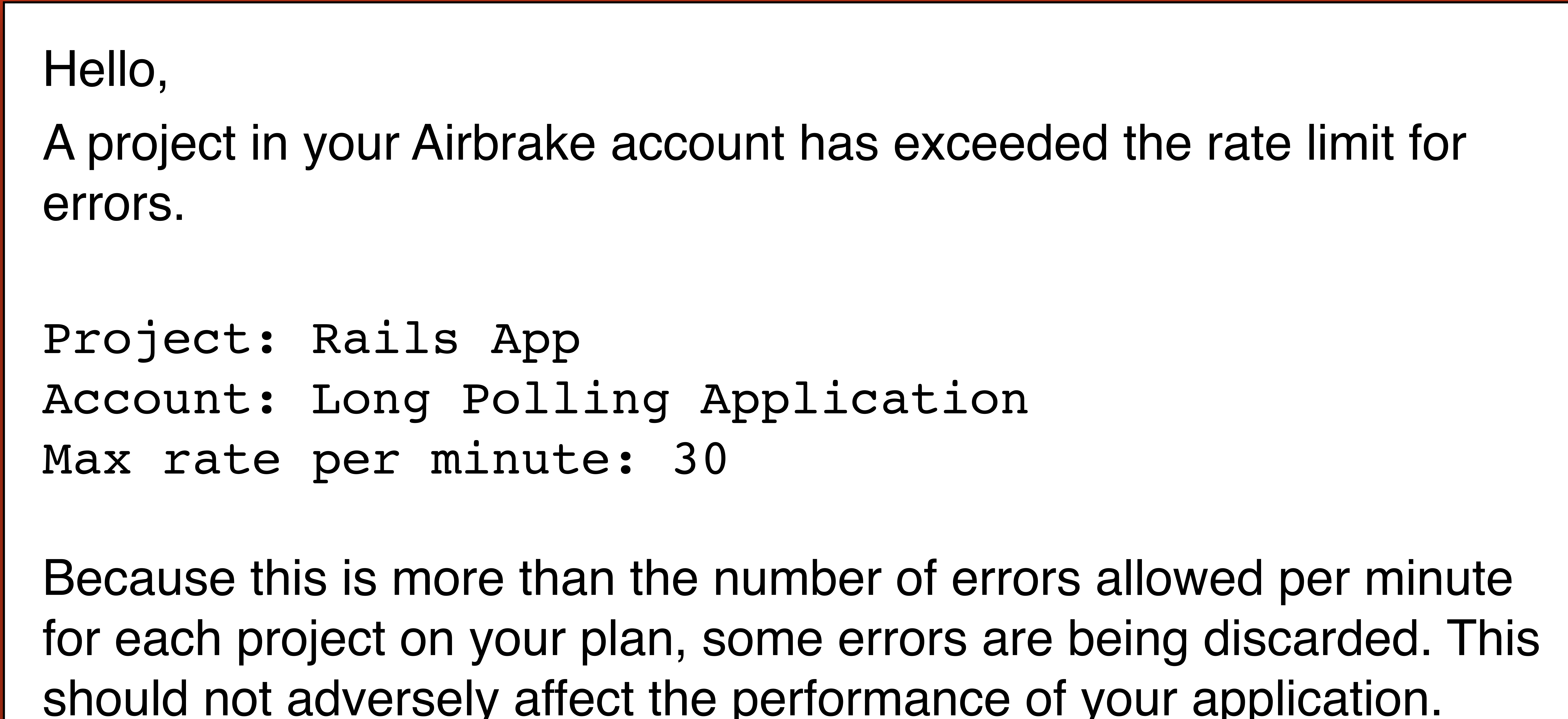
nginx cache

Highly optimized Rails metal

Redis counter caches

DB Caches

When **errors** happen, there are **lots of them**



Hello,

A project in your Airbrake account has exceeded the rate limit for errors.

```
Project: Rails App
Account: Long Polling Application
Max rate per minute: 30
```

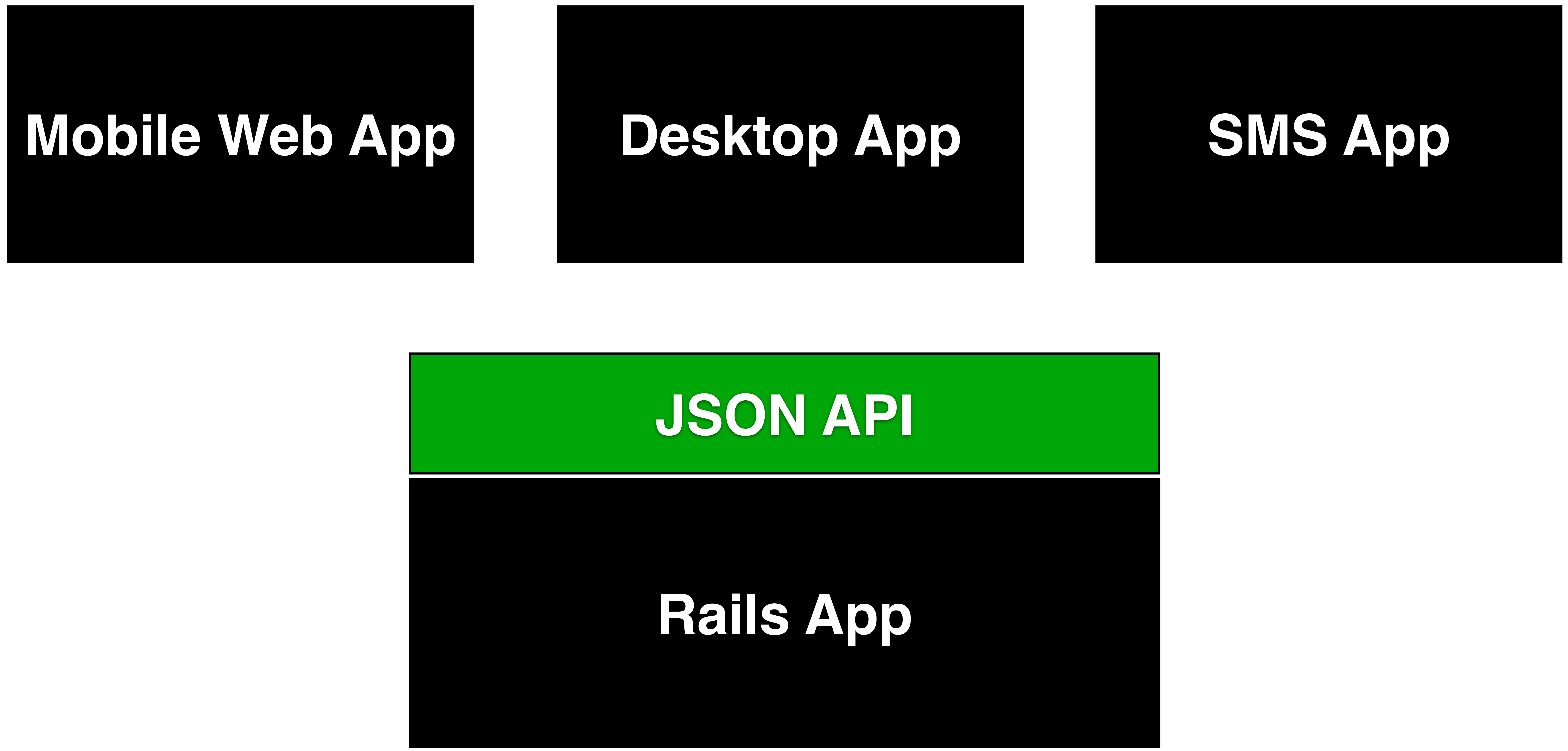
Because this is more than the number of errors allowed per minute for each project on your plan, some errors are being discarded. This should not adversely affect the performance of your application.

Does not work for large datasets or streams

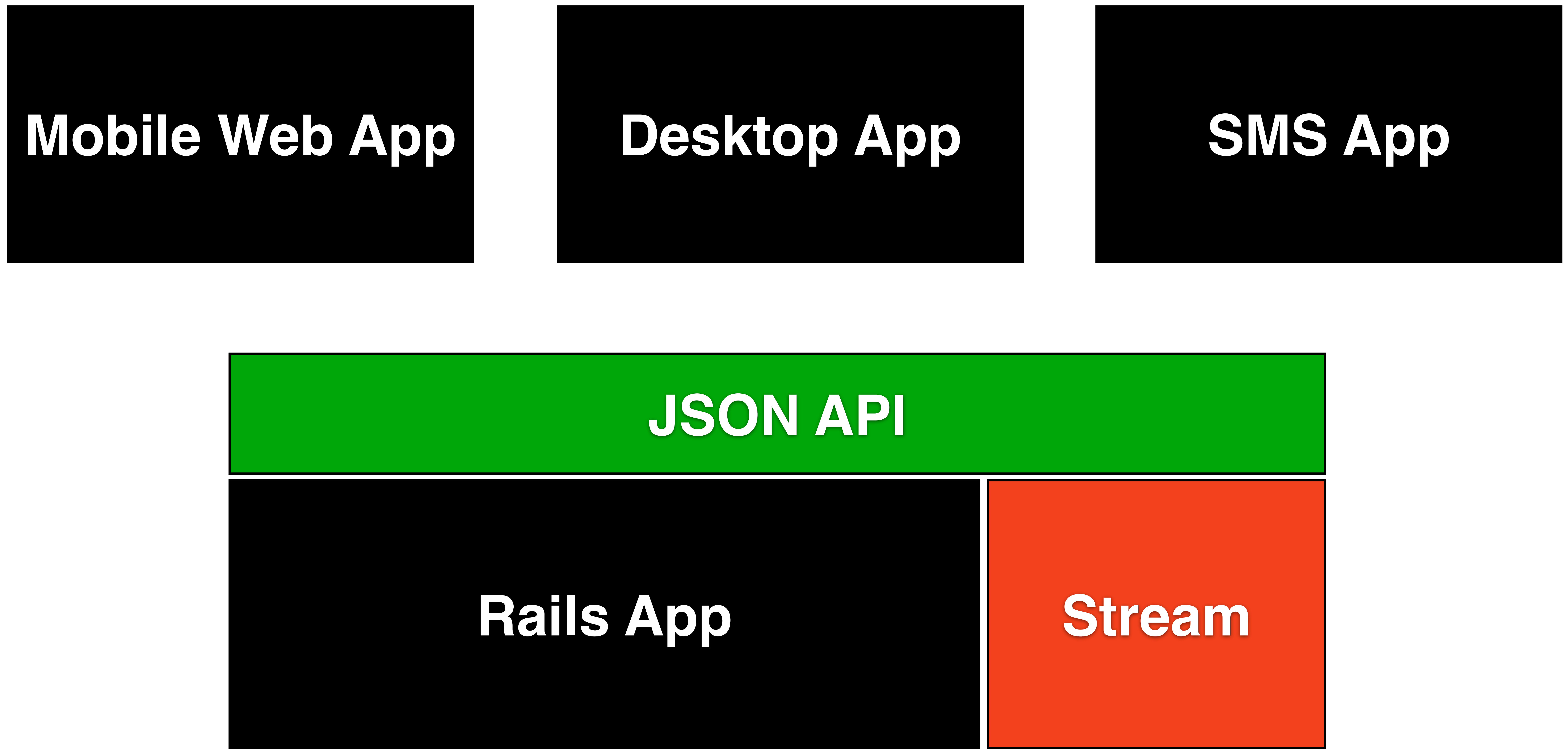
For larger development teams, **monolithic apps** can **slow things down**

Rails App Maximus

Decompose app and team **into smaller pieces**



...and sprinkle in some streaming



Stream

?

Socket.IO didn’t feel quite right

- Problems simulating a full-duplex low-latency socket when using transports other than WS
- Routing on Channels, not URIs (no “/users/:id”)
- It felt like “too much” in the wrong areas and “too little” in the right areas

Meteor

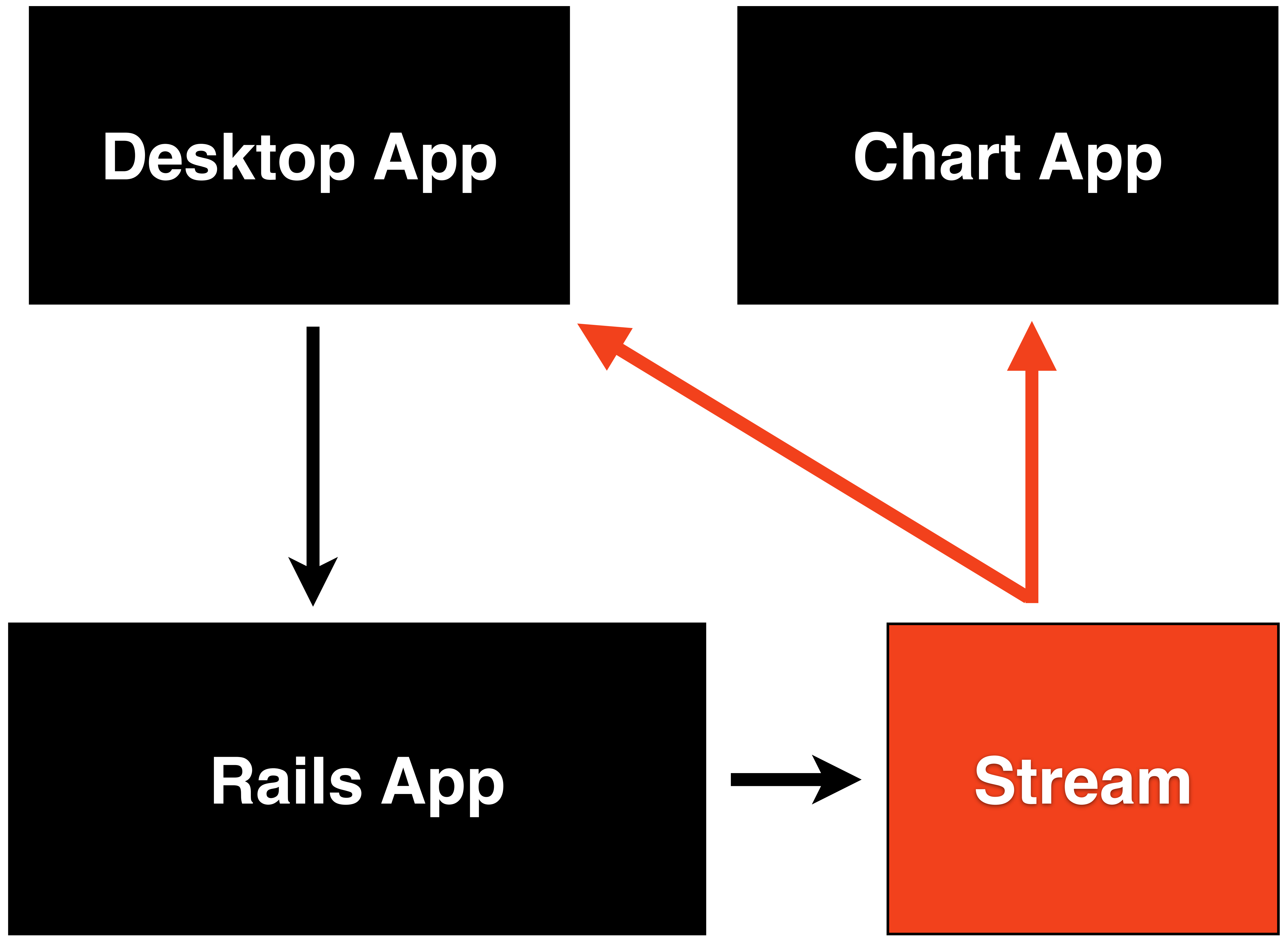
- New to the game, looks very promising in some areas
- For our team composition, its too tightly coupled and would end up becoming monolithic

“What problem am I ***really*** trying to solve?”

Web apps are really **great at persisting data** from clients and **serving it up fast**, but...

Web apps are **lousy at pushing data** from the server to the client **when something changes**

“All I want to do is **push resources**”



Firehose.io

Build realtime web applications

How does **Firehose**.io **work**?

```
$ gem install firehose

# Install rabbitmq

$ firehose server
```

URLs are the **exchange**, **Resources** are the **messages**

Publish

```
$ curl -X PUT -d "{name: ‘Fred’}" "http://
127.0.0.1:7474/users/1.json"
```

Subscribe

```
$ curl "http://127.0.0.1:7474/users/1.json"
```

Publishing from **ActiveRecord**

```
require ‘net/http’

class User < ActiveRecord::Base
  after_commit do
    req = Net::HTTP::Put.new("/users/#{id}/firehose.json")
    req.body = to_json
    Net::HTTP.start('127.0.0.1', 7474).request(req)
  end
end
```

```
// Backbone.js and Firehose.io

var user = new User({
  name: "Freddy Jones
});

new Firehose.Client()
  .uri('//users/1.json')
  .message(function(msg){
    return user.set(JSON.parse(msg));
  }).connect();
```

Subscribing from **Backbone.js**

Current implementation runs on

Thin + RabbitMQ

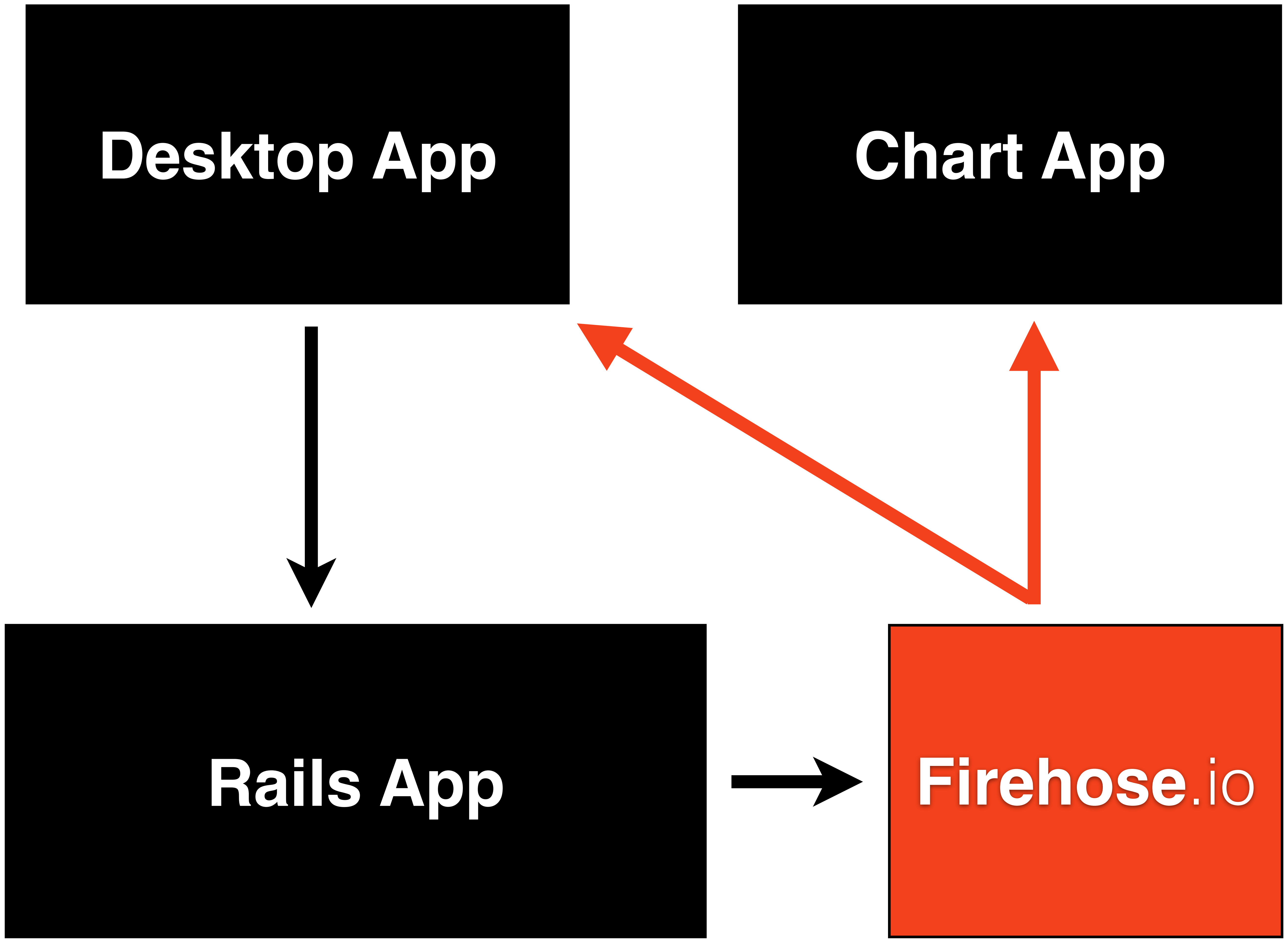
```
when 'GET'
  EM.next_tick do
    subscription = Firehose::Subscription.new(cid)
    subscription.subscribe path do |payload|
      subscription.unsubscribe
      env['async.callback'].call([200, {}, [payload]])
    end
  end
  Firehose::Rack::AsyncResponse

when 'PUT'
  body = env['rack.input'].read
  Firehose::Publisher.new.publish(path, body)
  [202, {}, []]

else
  [501, {}, ["#{method} not supported."]]
end
```

Transports *only* include

WebSockets + HTTP long polling

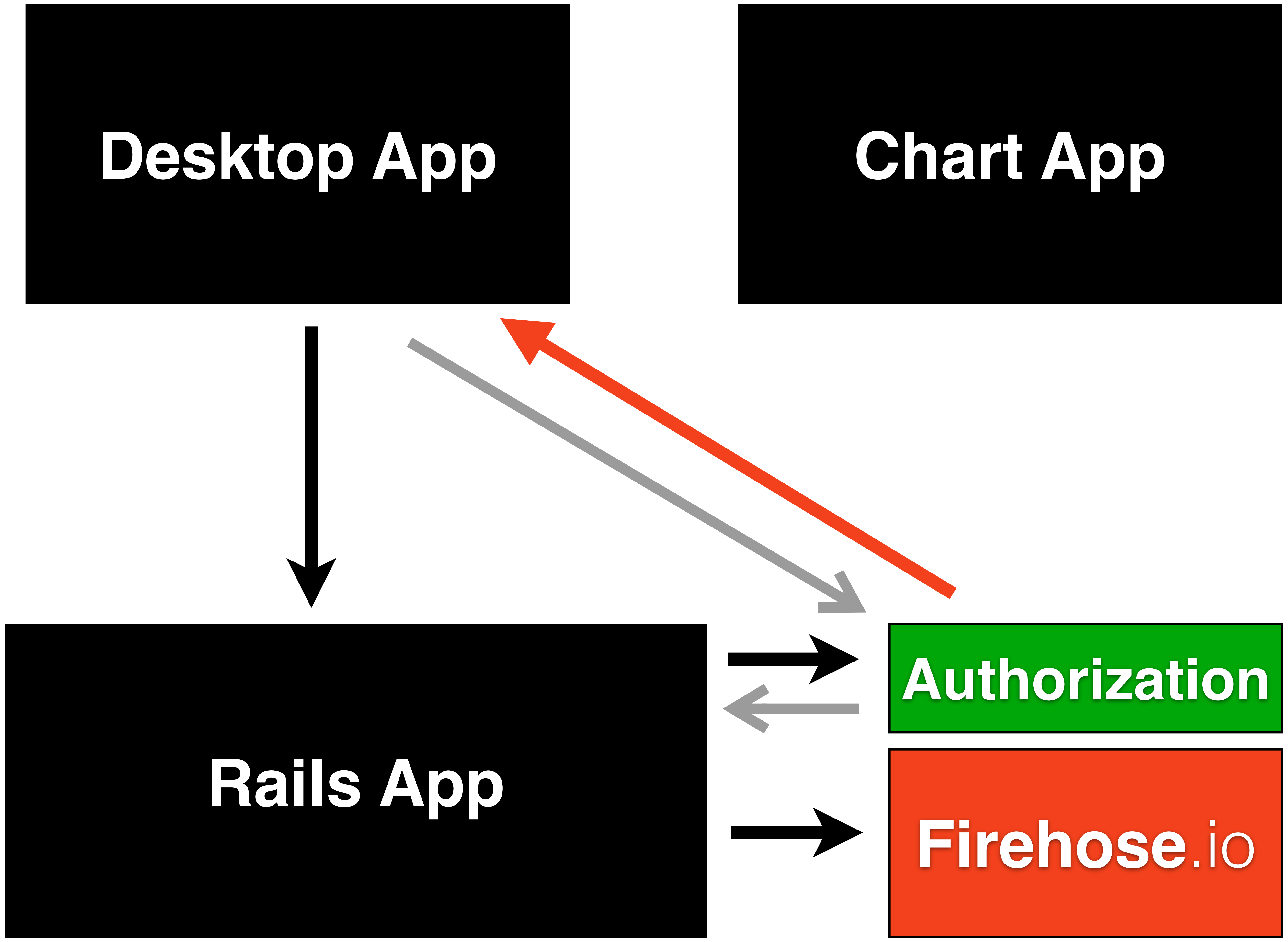
It hangs off the side so its

Minimally Invasive



Firehose.io
Experiments

Authorization Proxy with **Goliath**



Different backends

ZMQ, Redis, Erlang, node.js

You can help!

Get it now at

Firehose.io

Join the team at

PollEv.com/jobs

@bradgessler