

**SIMPLE**

**REAL TIME ANALYTICS**

**WITH MONGODB**

# INSPIRED BY

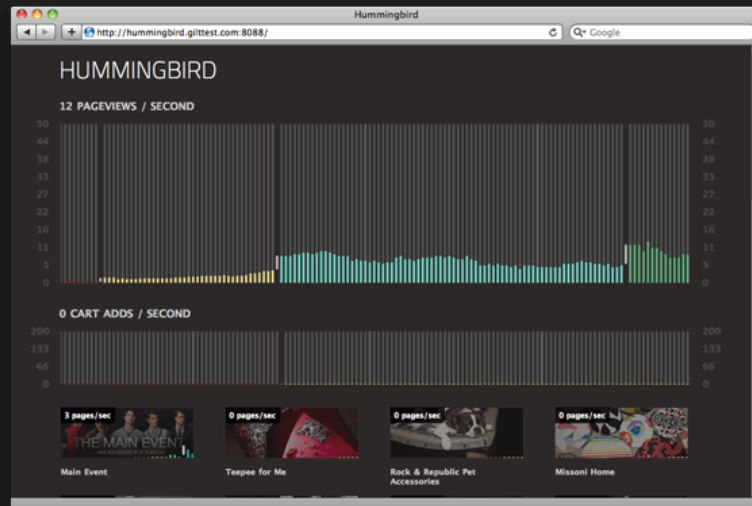
<http://hummingbirdstats.com/>

## HUMMINGBIRD

Real Time Web Traffic Visualization [Preview](#)



Note: Hummingbird is pre-alpha software. While it shouldn't screw up your production environment, it's definitely not yet production ready. In fact, we're still in the process of extracting it from the original app. So check it out, but don't expect it to fulfill your greatest desires yet.



Hummingbird lets you see how visitors are interacting with your website in real time.

And by "real time" we don't mean it refreshes every 5 minutes—WebSockets enable Hummingbird to update 20 times per second.

Hummingbird is built on top of [Node.js](#), a new javascript web toolkit that can handle large amounts of traffic and many concurrent users.

[Go to the GitHub page](#)

[View the Live Demo](#)

### How It works



# MORE AT

<https://devcenter.heroku.com/articles/realtime-polyglot-app-node-ruby-mongodb-socketio>



This article was contributed by **Ben Wen**

Ben works at **MongoLab**, a cloud hosting provider of the NoSQL MongoDB database. MongoLab monitors, backs up, and simplifies running MongoDB in production.



## Building a Real-time, Polyglot Application with Node.js, Ruby, MongoDB and Socket.IO

*Last Updated: 27 March 2012*

mongodb

nodejs

polyglot

ruby

socketio

### Table of Contents

- Overview
- MongoDB as a message queue
- Provision Ruby publisher app
- Configure MongoDB capped collection
- Deploy publisher app
- Using Ruby to queue messages
- Provision Node.js web app
- Sharing application resources
- Deploy web app
- Consuming messages in Node.js
- Pushing messages to the browser with Socket.IO
- Conclusion

Real-time apps, or evented apps that incorporate push-based interactivity, are the basis for a new generation of in-browser capabilities such as chat, large-scale games, collaborative editing and low-latency notifications. Though there



**PS - CODE ON GITHUB**

**[HTTPS://GITHUB.COM/JULIANBROWNE/RTSDEMO](https://github.com/julianbrowne/RTSDemo)**

## STEPS

- Make a *capped collection*
- Put stuff in
- Query it with a *tailable cursor*
- See stuff come out
- Send it somewhere useful

# MAKE A CAPPED COLLECTION

```
db.createCollection("name",  
    {capped:true, size:1000});
```

- Fixed size (bytes)
  - or docs (sort of)
- Loop-around writes
- A few constraints
  - can't delete
  - can't do collection.remove()
  - can't grow doc size
  - by default no \_id index
  - no \_id uniqueness



# THE TAILABLE CURSOR

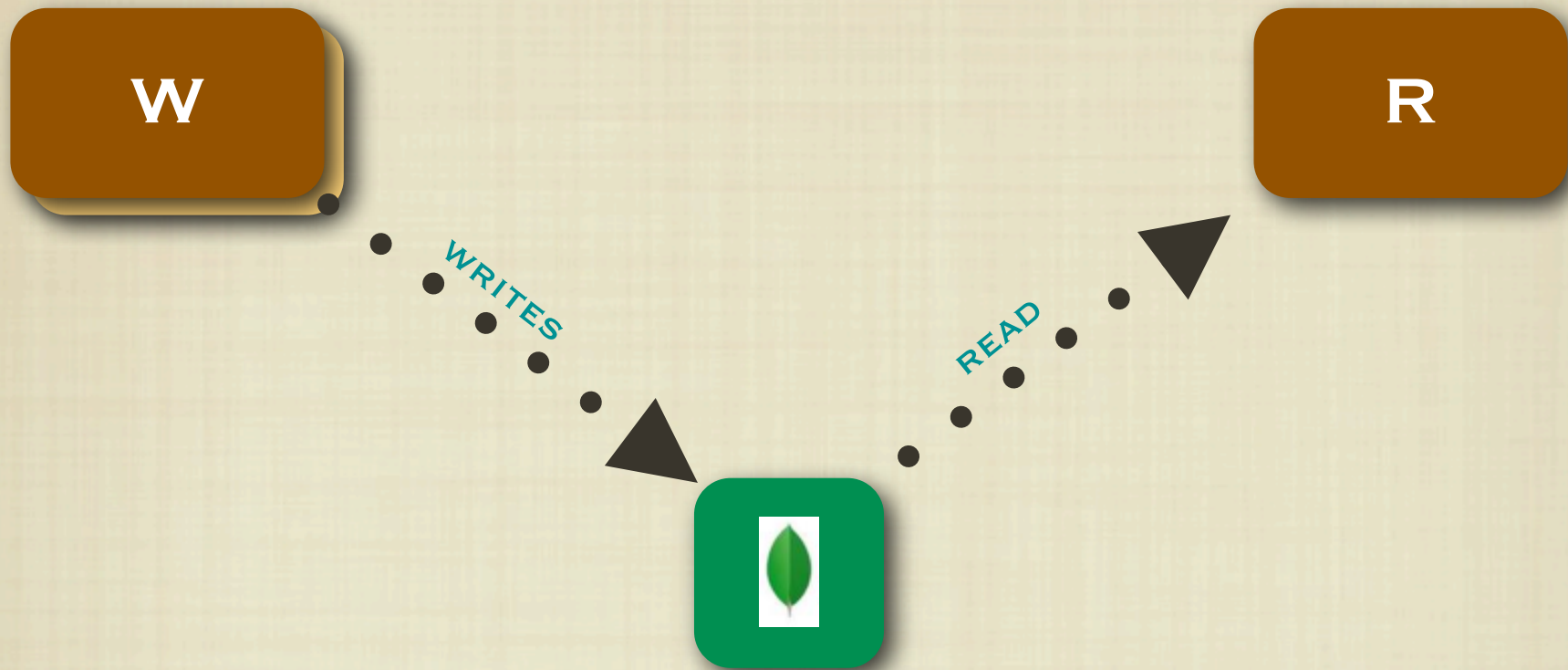
```
c = db.collection.find({})  
    .addOption(2)    // tailable  
    .addOption(32); // wait
```

# THE TAILABLE CURSOR

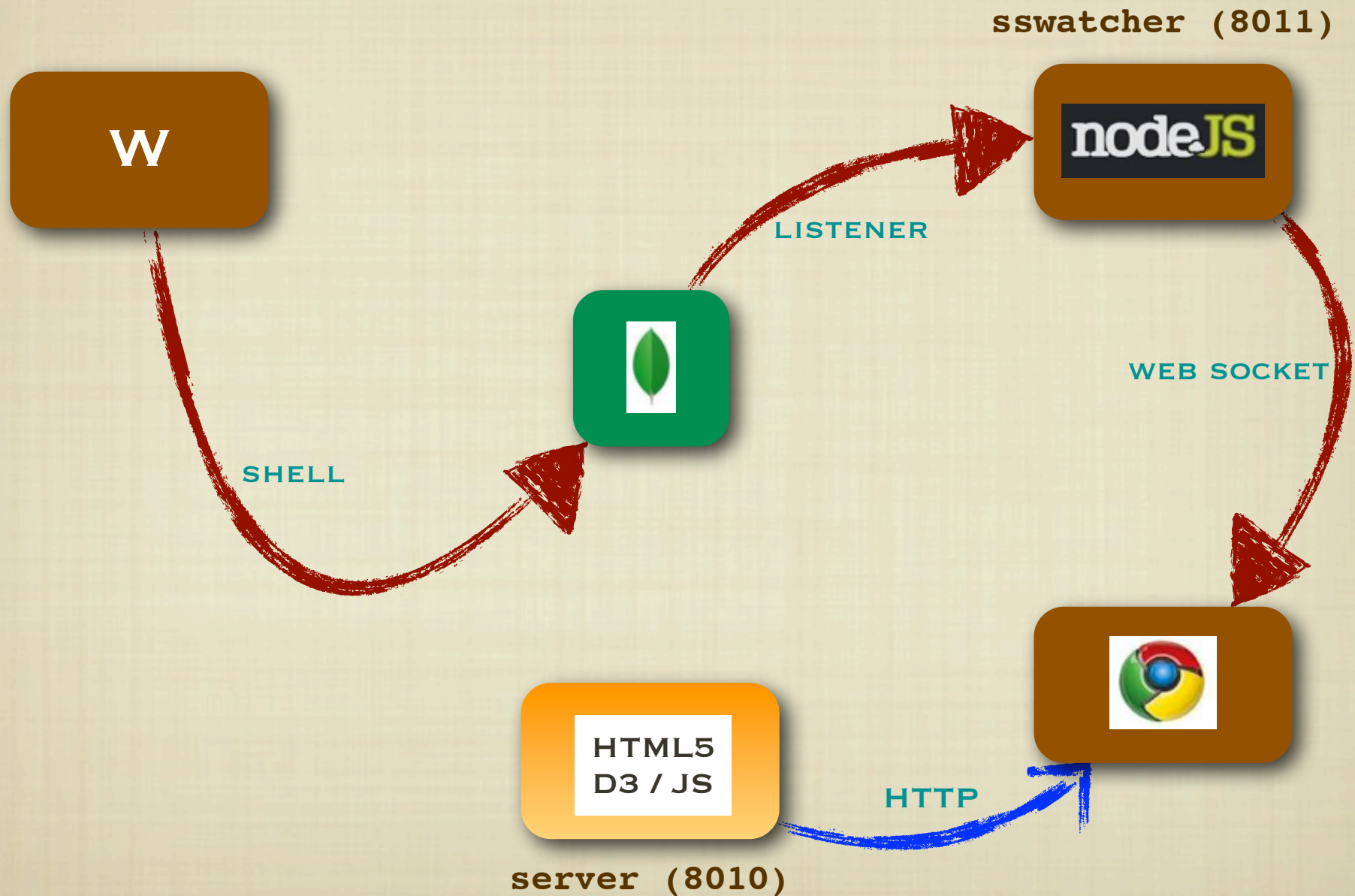
```
c = db.collection.find({})  
    .addOption(2)    // tailable  
    .addOption(32); // wait  
  
for(i=0;i<5;i++) {  
    while(c.hasNext()) {  
        var doc = c.next();  
        printjson( doc );  
    }  
}
```






# PUT STUFF IN / GET STUFF OUT



# SEND IT SOMEWHERE USEFUL



# CAPPED COLLECTION EXEMPLAR

-  Replica Set
-  Powered by the Oplog
-  Knows All Sees All



# SEE ALL

