# Scaling geodata with MapReduce

Nathan Vander Wilt

## Background

I’m Nate, a freelancer — I do web, native, embedded development using Cocoa, Django, node.js, C/C++, from SQLite to Couch. Of all, I love to talk about Couch the most :-)

Audience poll: who’s used Couchbase (or similar) “at scale” — tons of users?

Well, for better or for worse, the user in this user story is me. Here's how I turned almost ten years of personal geodata into something I can relive in less than two seconds. I hope you'll find some of these ideas helpful when dealing with tens of thousands of users.

## ON THE DOCKET:

- Views as “indexes”
- Basic location examples
- Geo Hacks

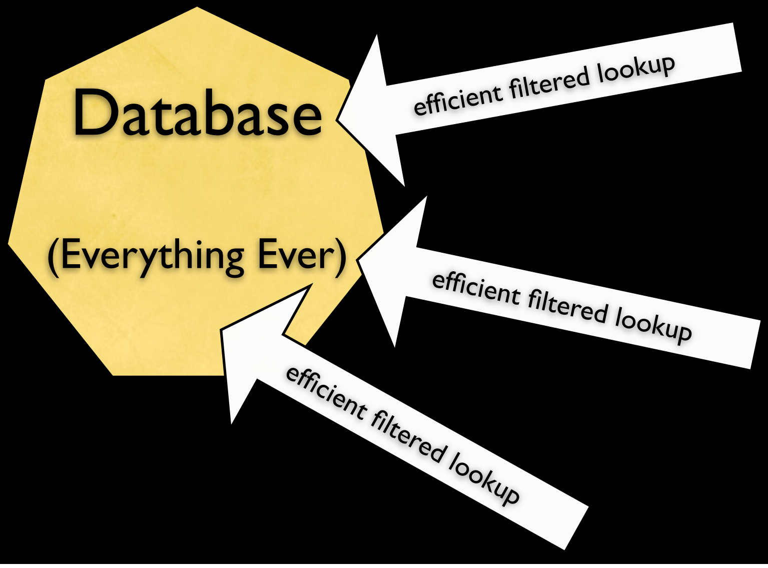
(pls to interrupt)

Gonna divide this up into three general chunks: make sure we’re all on the same page as far as concepts go, then dive into the main examples. Finally we’ll explore at some interesting twists of the available features.

Feel free to interrupt at any time. I’d like to have some discussion between each of these main sections, so you can also save questions for then.

## Views as “indexes”

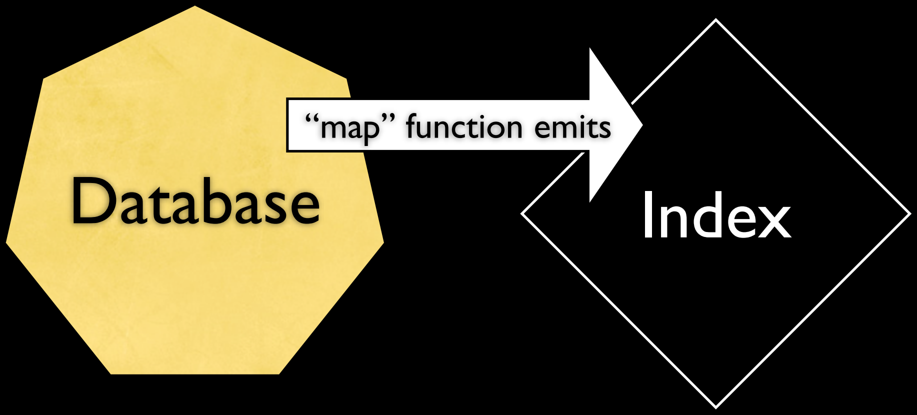
## Views as “indexes”


Database
(Everything Ever)
efficient filtered lookup
efficient filtered lookup
efficient filtered lookup

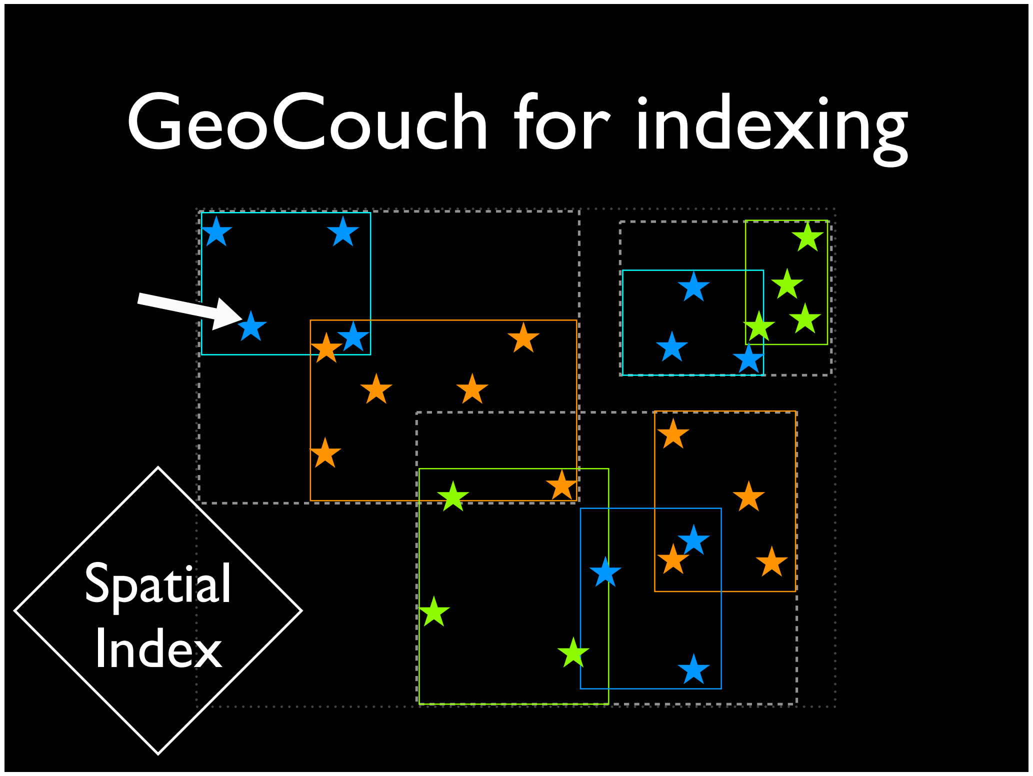
What do I mean by “indexes”?

Efficient lookup of data, extracted from documents. Think of a word index in a book, or the topical index of a “real” encyclopedia set.

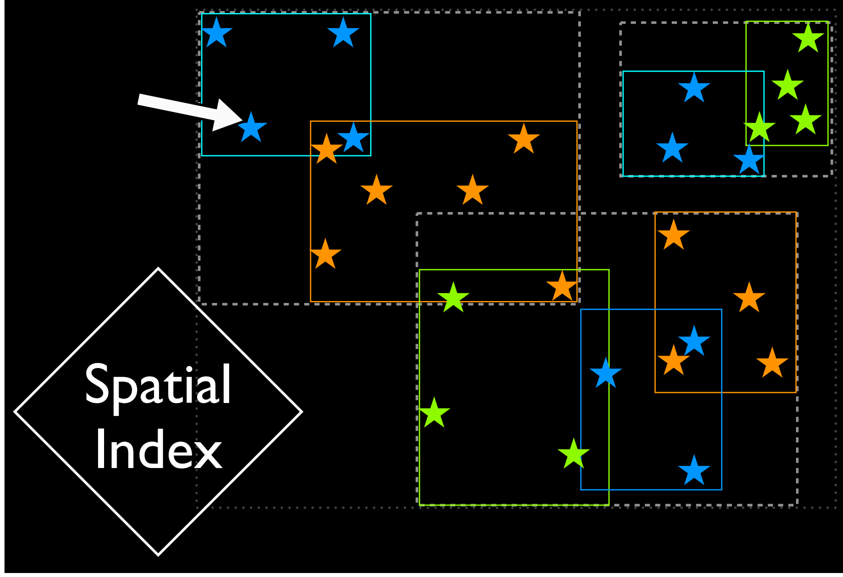
## Views as “indexes”



With Couch you have full control. Your code defines which terms — “keys” — go into this index.

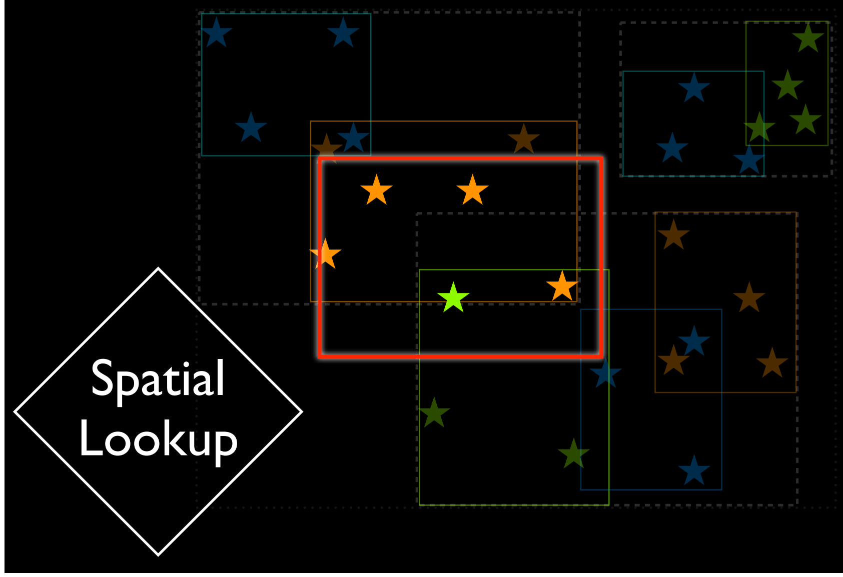


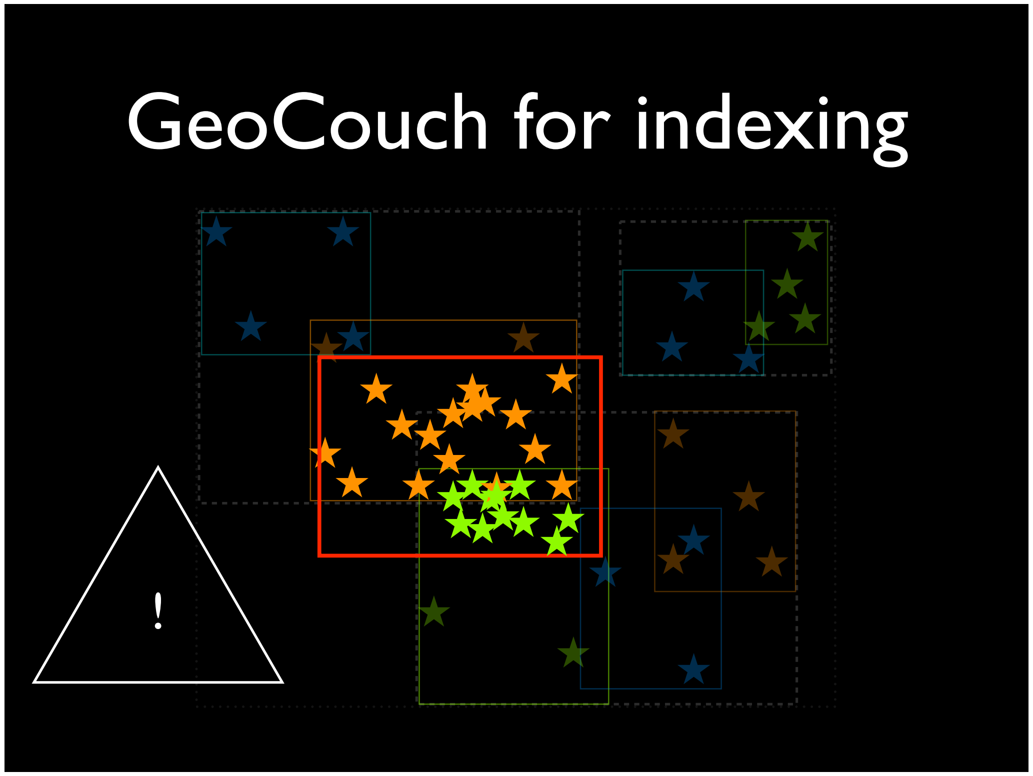
## GeoCouch for indexing



Using R-trees as a 2-dimensional index to speed bounding box queries.

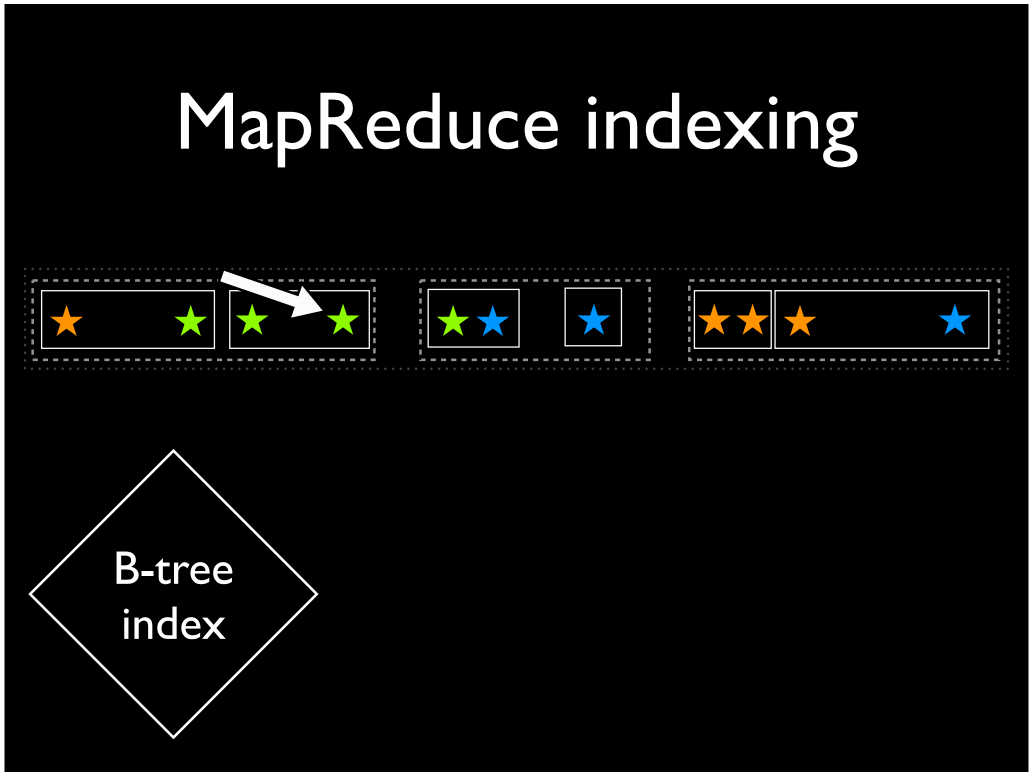
## GeoCouch for indexing





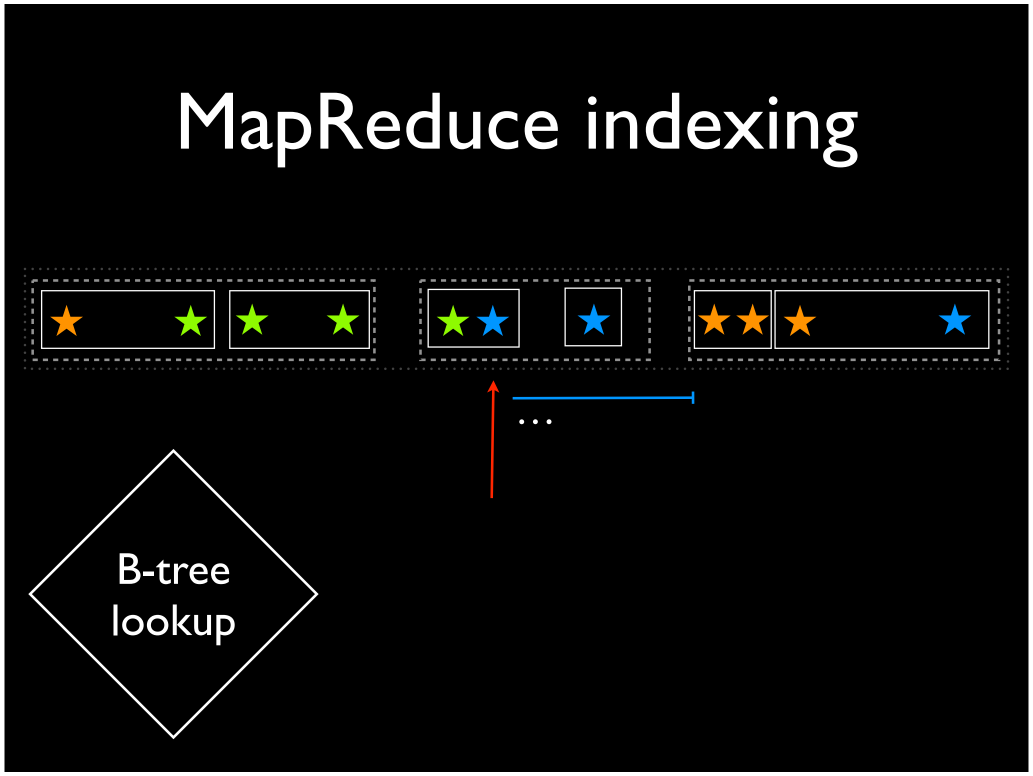
# GeoCouch for indexing

Drawbacks if a lot of points inside bounding box...



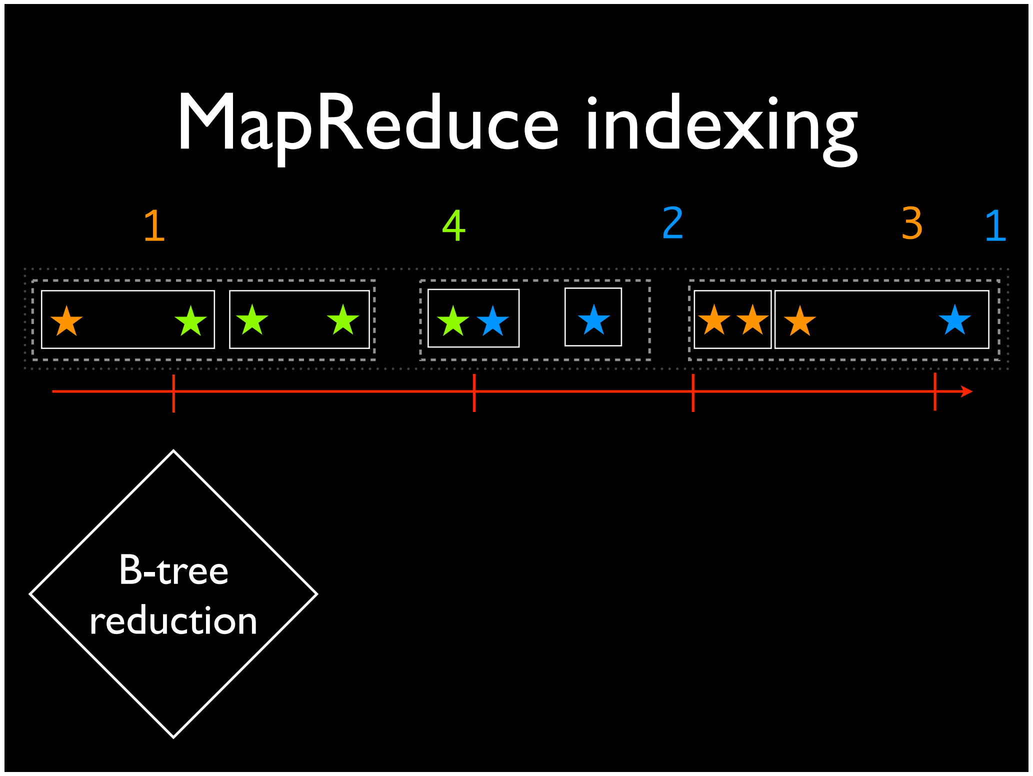
## MapReduce indexing

Use B-trees as a index to speed 1-dimensional lookups



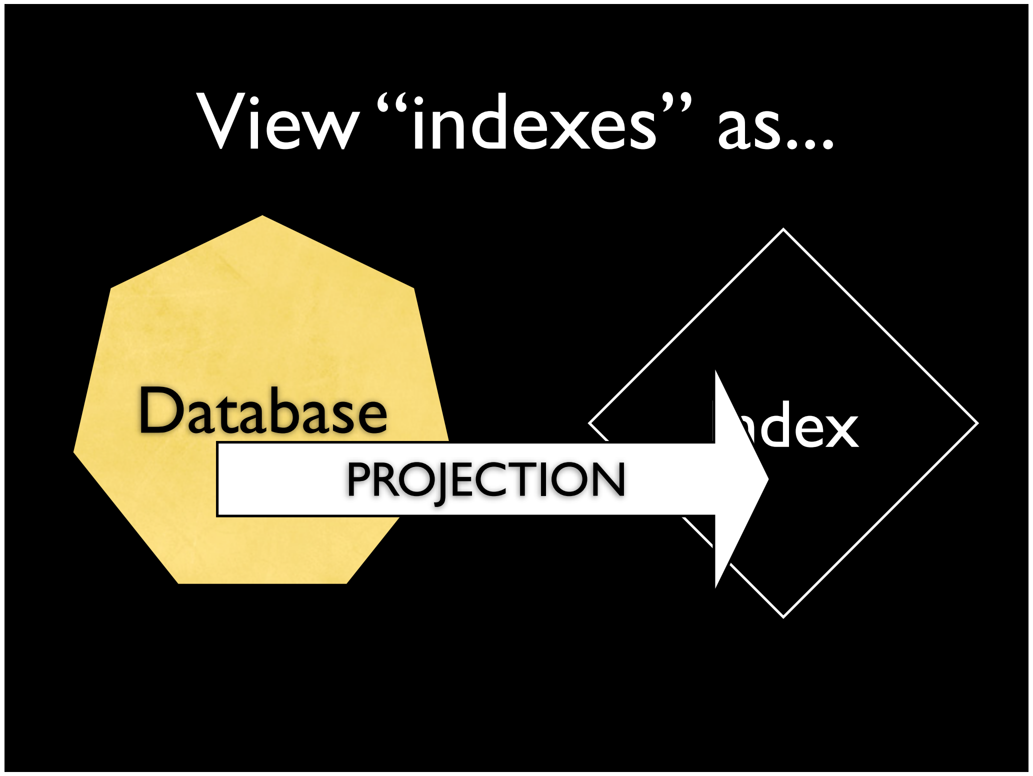
# MapReduce indexing

efficient “start/end” range queries



## MapReduce indexing

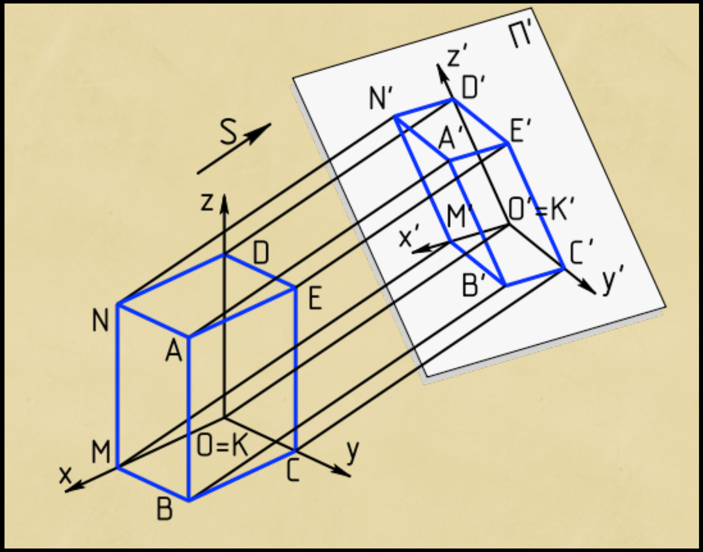
as well as grouped “reduce” queries!



## View "indexes" as...

Because Couch exposes its index keys directly, you can imagine the indexes as "projections" of data viewed from different perspectives.

## “Indexes” as projections



http://en.wikipedia.org/wiki/File:Axonometric_projection.svg

What do I mean by projection?

Example of 3D onto 2D: pictures.

## “Indexes” as projections

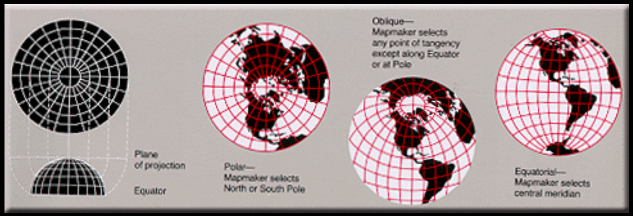
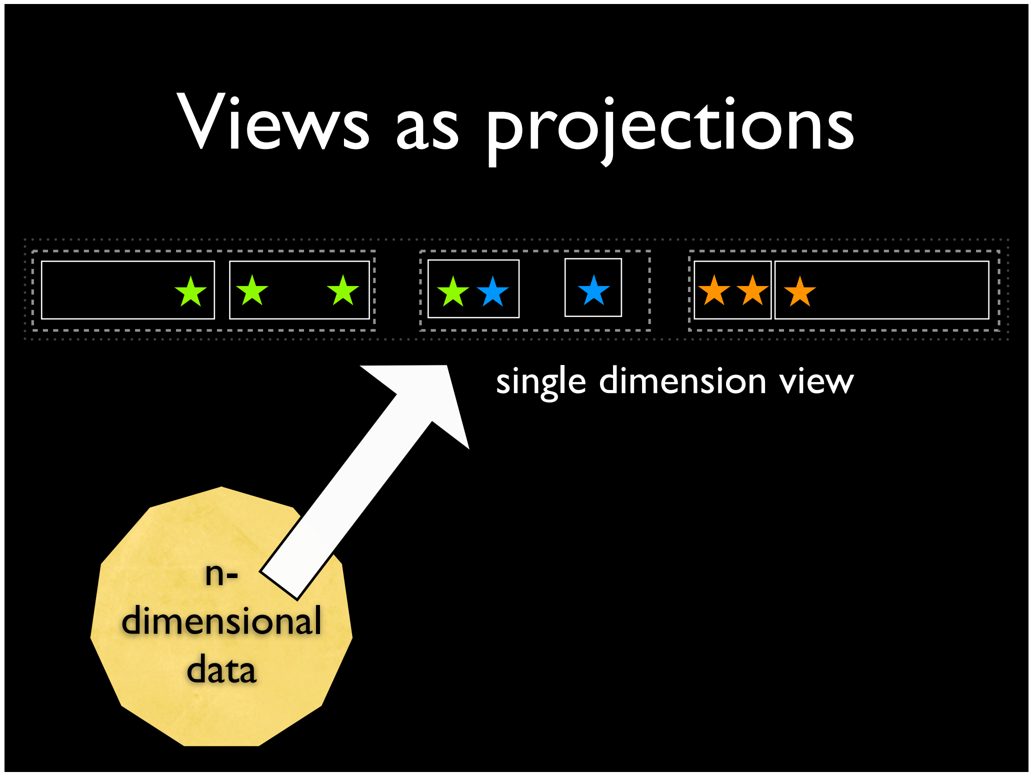
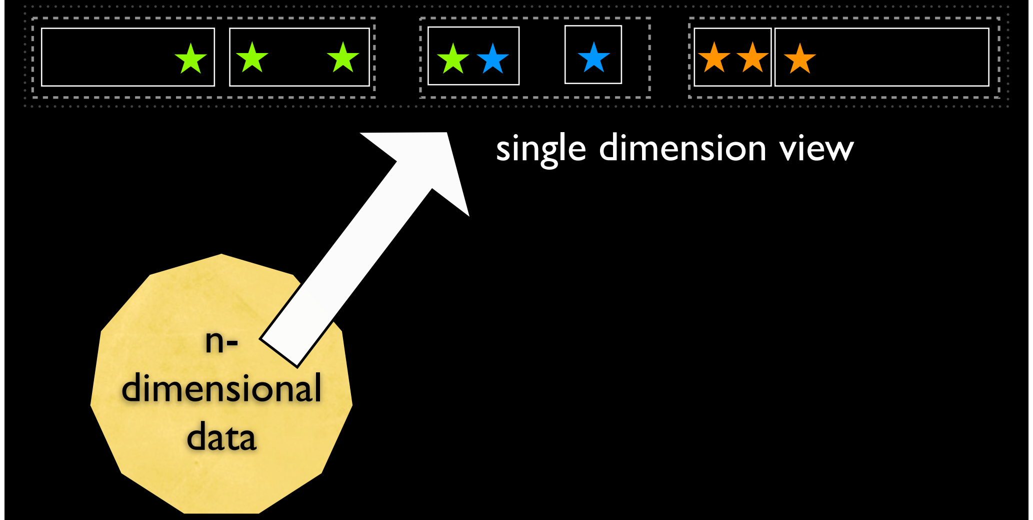


Image: USGS

Example of n-D onto 1-D: map functions



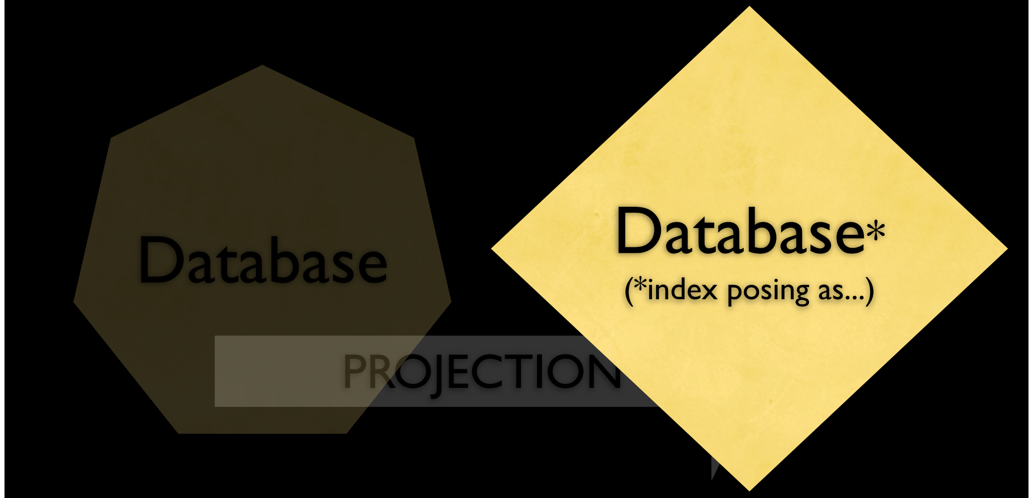
## Views as projections



Example of n-D onto 1-D: map functions!
Imagine each “aspect” of your data as its own dimension.

We’ll focus back on this in the last section on Geo Hacks, but it’s helpful to keep in mind

## View “indexes” as...



Because they’re sort of just a different “perspective” on your data, in all the examples ahead we’ll look at Couch indexes almost as databases themselves.

## Questions so far?

## Basic location indexes

## Location example #1

Where was I when...?

## Location example #1

Map function

```
for each pt in doc:
    emit(pt.timestamp, pt.coord)
```

## Location example #1

(Demo)

## Location example #2

Where have I been?

## Location example #2

Map function

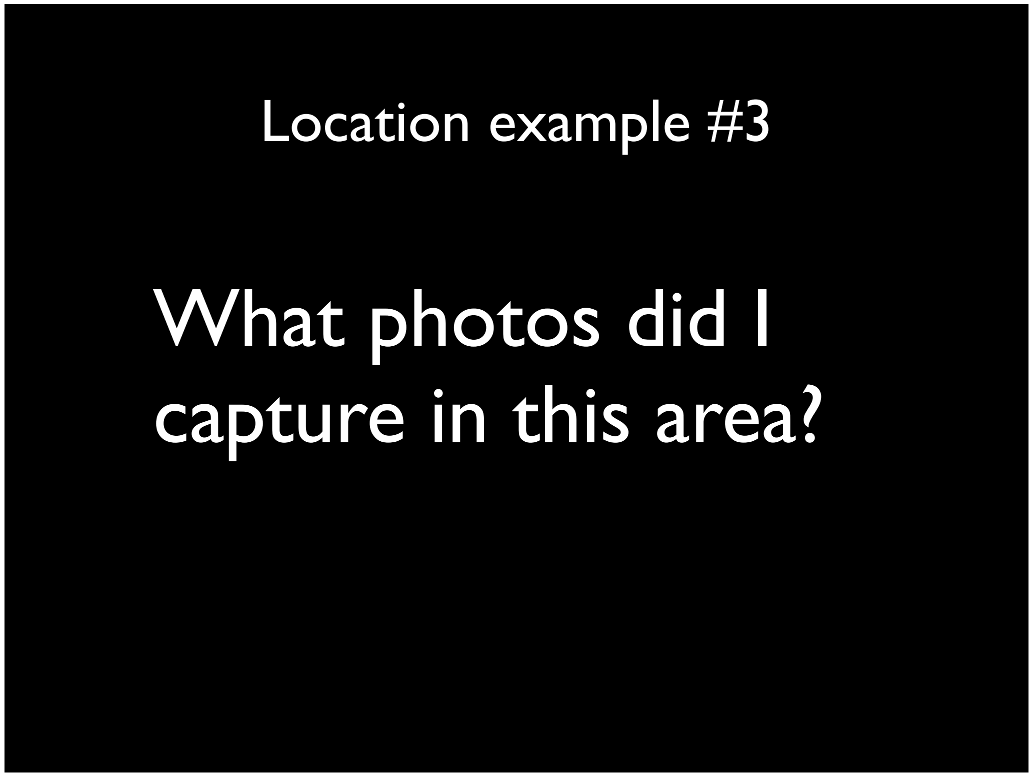
```
for each pt in doc:
    emit(pt.timestamp, pt.coord)
```

Reduce function

```
for each value in reduction:
    avg += value.coord
    n += value.n || 1
return {avg, n}
```

## Location example #2

(Demo)

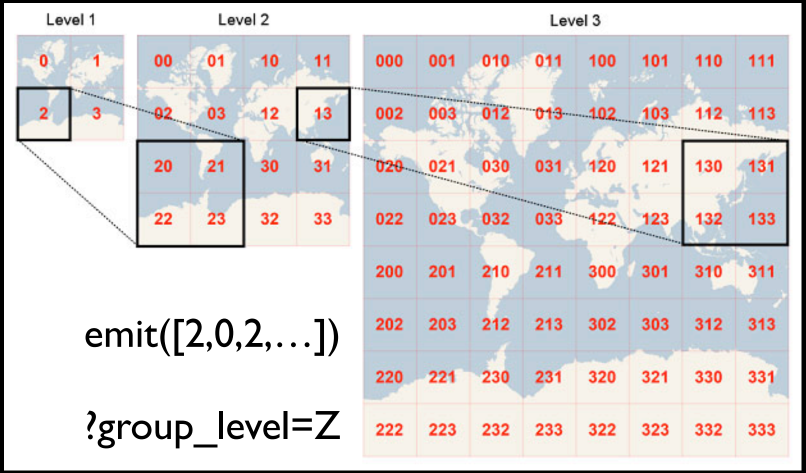


## Location example #3

What photos did I capture in this area?

The reason I’ve been recording my location for so many years.

# Location example #3



emit([2,0,2,…])

?group_level=Z

Image: http://msdn.microsoft.com/en-us/library/bb259689.aspx

Tiles, quadkeys

## Location example #3

Map function

```
for each pt in doc:
    key = quadkey(pt.coord)
    emit(key, pt)
```

Reduce function (basic)

```
_count
```

## Location example #3

(Demo)

## Questions now?

Have I managed to confuse anyone by now?

## Scalable geo hacks

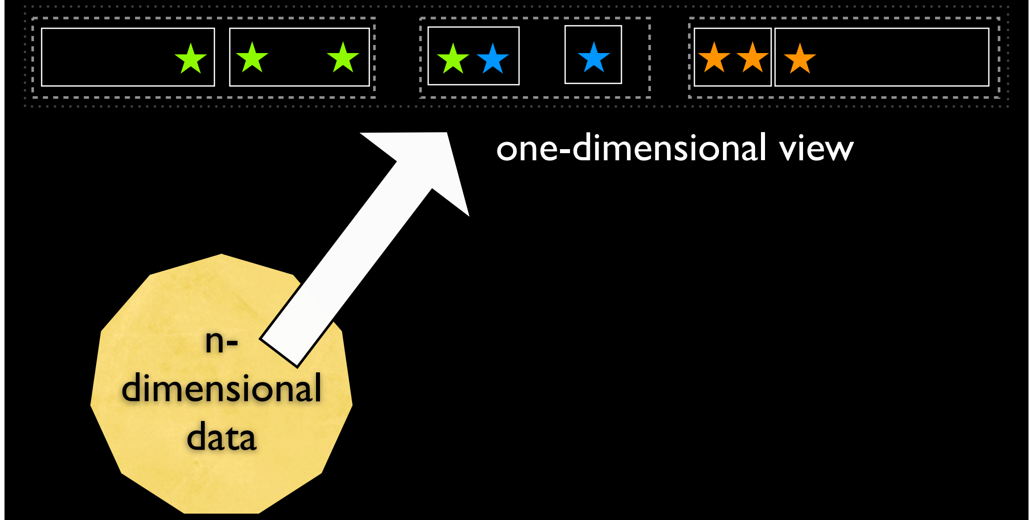
Geo hacks -> scalable geo hacks.
Keep in mind that everything we’ve talked about so far will scale well. I have “only” a few million location breadcrumbs — that’s a lot, but you might have many many more. The same index trees that got me this far are designed to keep going farther.

Now we’re going to talk about some hacks: some I haven’t actually used yet, the next makes your code uglier, and the last one adds <u>cheating</u> on top of that. But all of these “hacks” still have some good scaling properties.

Hack #1

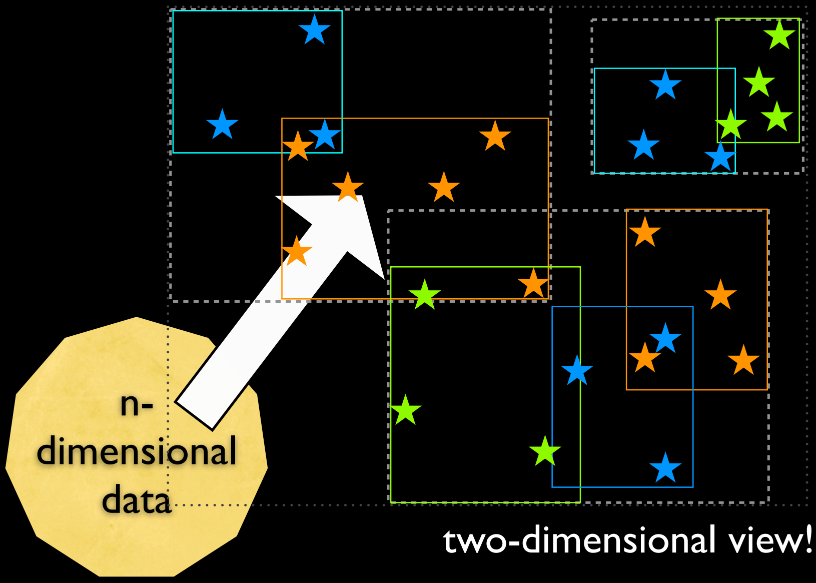
## Geo: not just for geo

## Hack #1



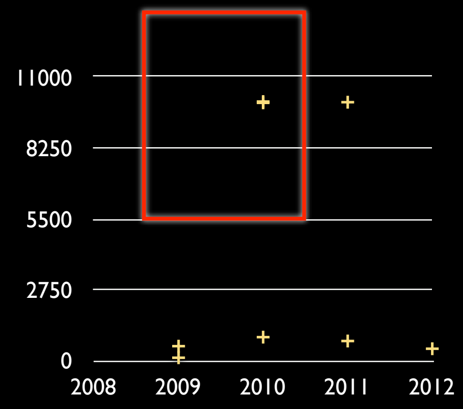
We talked about this: “projecting” an aspect of multi-dimensional data onto a 1-d sorted index. But why not project onto a...

## Hack #1

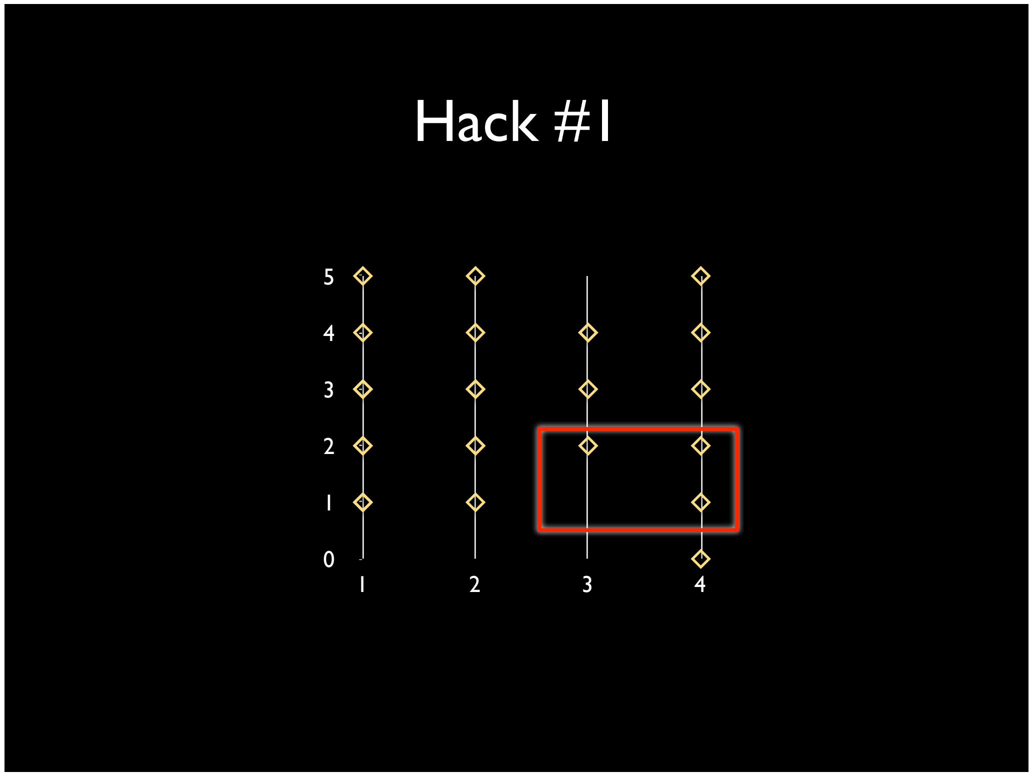


2-d index!

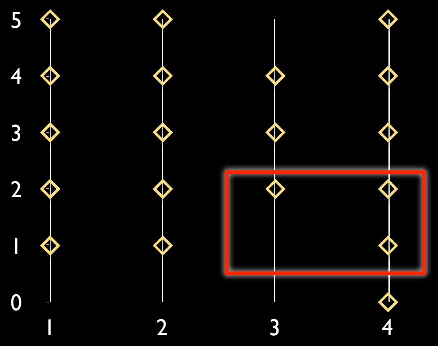
## Hack #1



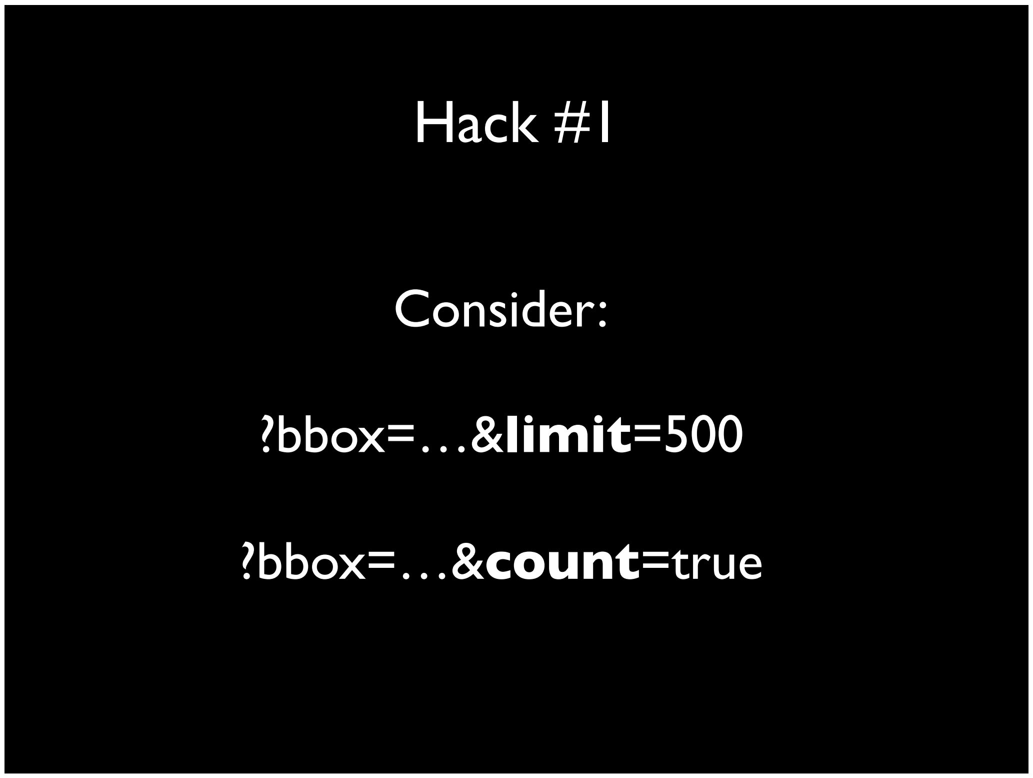
Time and altitude, e.g. when did I fly in 2009–2010



## Hack #1



Camera and rating, e.g. clean up bad photos from recent cameras



## Hack #1

Consider:

?bbox=…&**limit**=500

?bbox=…&**count**=true

Unfortunately, this does not provide a *sorted* index. GeoCouch is scalable in the sense that the determining which objects are within the bounding box will stay fast, but the result set might be manageable only when counting or limiting.

## Hack #1

(No demo, sorry)

## Hack #2

All log(n) you can eat

## Hack #2

Map function (usual timestamp/location emit)

...

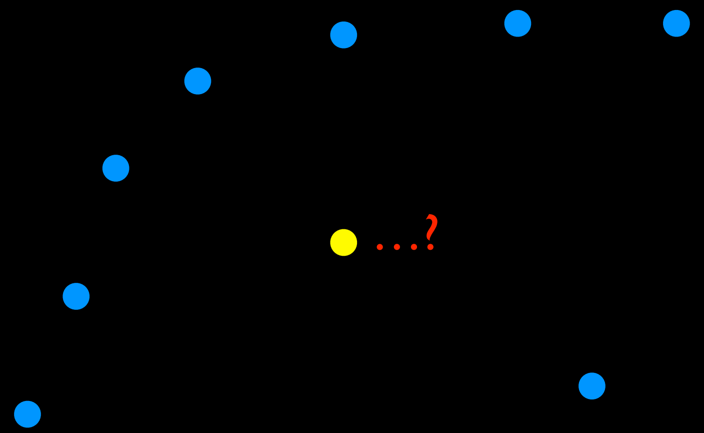
Reduce function

```
n = Math.log(values.length)
while averages.length < n:
    averages.push(another)
return averages
```

Hack #3

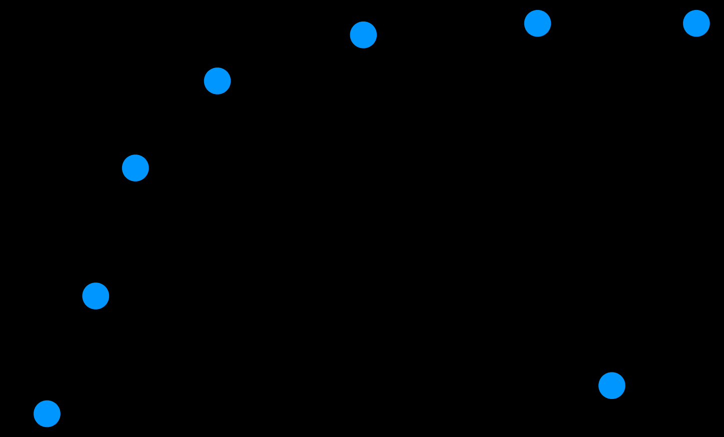
## Out-of-the ordinary location summary

## Hack #3

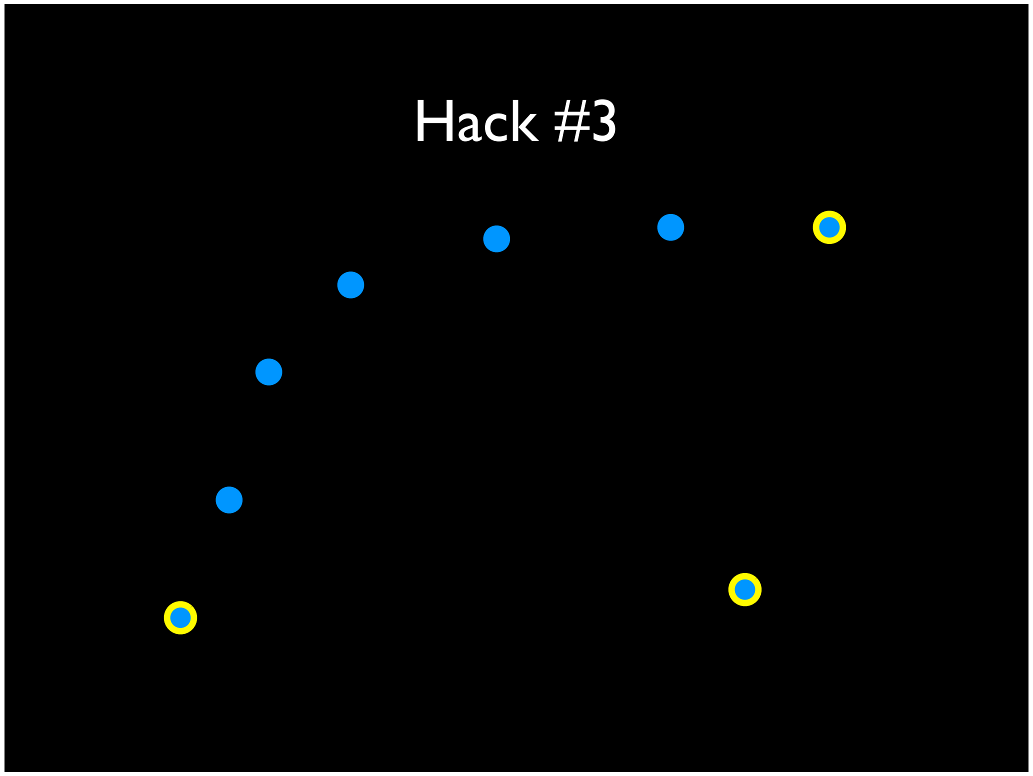


Averaging locations puts dots on places I've never actually been.

## Hack #3



What might be more interesting is to pick good samples of actual locations I've been too. "Representative" locations so to speak. What's a good "representative" location? The most common, ones with a lot of other points nearby, ones closest to that average we calculate?



## Hack #3

Finally realized that it was actually the “outliers” that provided the most interesting answer to “where have I all been”!

## Hack #3

Map function

…

Reduce function

```
n = Math.log(values.length)
while outliers.length < n:
    outliers.push(another)
return outliers
```

using log(n) trick from before

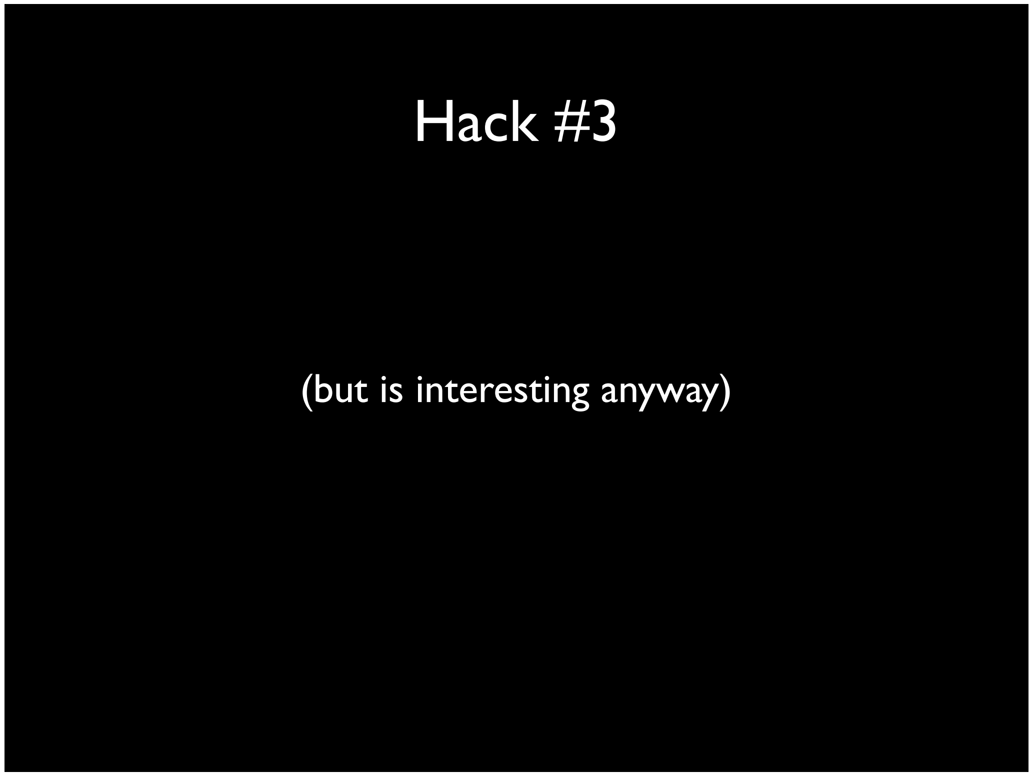
## Hack #3

**Caution:** RESULT IS "UNDEFINED"

This cheats.

Reduce function should be "commutative and associative for the array value input, to be able reduce on its own output and get the same answer"

This is not really associative: the result depends on the internal tree structure. The points picked are random/unstable.



## Hack #3

(but is interesting anyway)

In this case, I’d rather have randomly picked but practically useful output data, than stable “averages”.

## Hack #3

(Demo)

Thanks!
(Any more questions?)

@natevw
http://exts.ch