

基于地理位置的服务 Location Based Service

课程版本 v5.0 主讲 东邪



扫描二维码关注微信/微博
获取最新面试题及权威解答

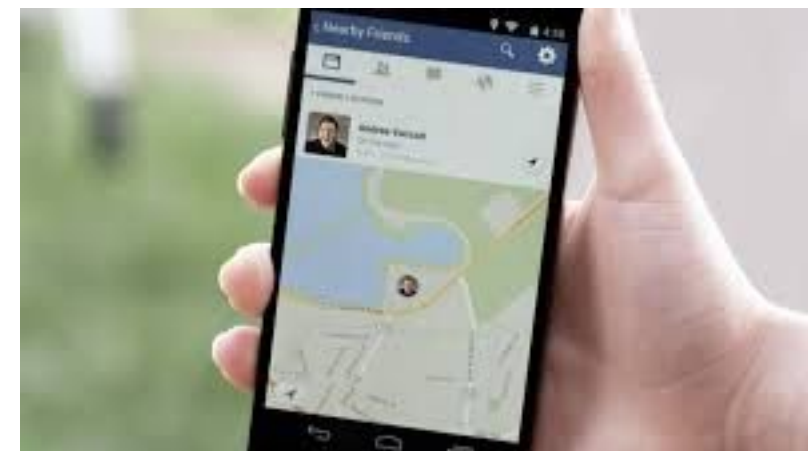
微信: [ninechapter](#)

微博: <http://www.weibo.com/ninechapter>

知乎: <http://zhuanlan.zhihu.com/jiuzhang>

官网: <http://www.jiuzhang.com>

- Design Uber
 - Design Facebook Nearby
 - Design Yelp
 - Design Pokemon Go
- 总结系统设计



Interviewer: Please design Uber

Similar questions:

How to design facebook nearby

How to design yelp

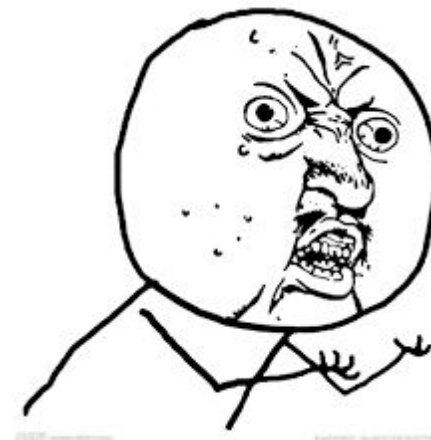


是否看过关于Uber Architecture 的视频, 讲座或是文章?



你大概会知道Uber用了如下一些技术

- RingPop
 - <https://github.com/uber/ringpop-node>
 - 一个分布式架构
 - 扩展阅读
 - <http://ubr.to/1S47b8g> [Hard]
 - <http://bit.ly/1Yg2dnd> [Hard]
- TChannel
 - <https://github.com/uber/tchannel>
 - 一个高效的RPC协议
 - RPC: Remote Procedure Call
- Google S2
 - <https://github.com/google/s2-geometry-library-java>
 - 一个地理位置信息存储与查询的算法
- Riak
 - Dynamo DB 的开源实现



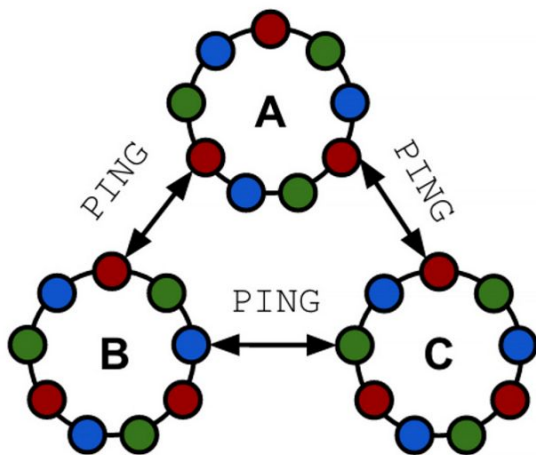
告诉我你看到这些词的感受是不是这样？

Read more on Uber Eng Blog

<http://eng.uber.com/>



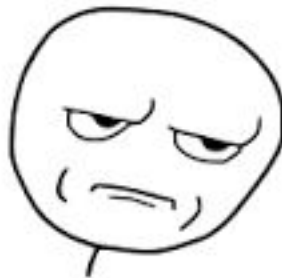
是不是答出Uber是怎么实现的, 就可以拿到Offer?



系统设计面试常见误区

以为答出该公司是怎么做的，就可以拿到Offer了

你他妈的在逗我？



Uber的架构非常小众

Uber用到的技术是自己设计出的一套东西

如果Uber面你这个题，你不可能比他们清楚，并且显得你是准备过的，不能代表你真实的能力

如果其他公司面你这个题，这也不会是期望答案

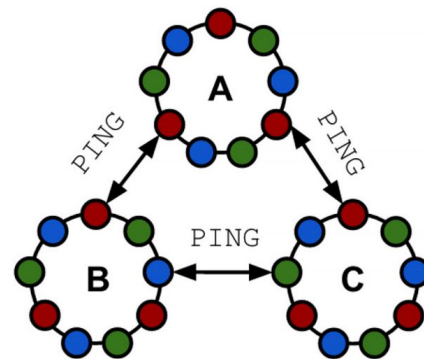
- Scenario 场景
 - Features
 - QPS / Storage
- Service 服务
 - Service Oriented Architecture
- Storage 数据
 - Schema

逻辑设计 Logic Design 50%
Make it work!



- Scale 进化
 - Robust
 - Feature

架构设计 Infrastructure Design 50%
Make it robust!



System Design = Logic Design + Infrastructure Design

系统设计 = 逻辑设计 + 架构设计



Scenario 场景

需要设计哪些功能，设计得多牛



设计哪些功能？

- 第一阶段：
 - Driver report locations
 - Rider request Uber, match a driver with rider
- 第二阶段：
 - Driver deny / accept a request
 - Driver cancel a matched request
 - Rider cancel a request
 - Driver pick up a rider / start a trip
 - Driver drop off a rider / end a trip

- 第三阶段 *:

- Uber Pool
- Uber Eat

无所谓的加分项, 如果你都答到这里了, 说明你前面都秒杀了



Scenario - 设计得多牛？

问问面试官：

贵优步现在多少辆车了？

贵优步现在多少QPS呀？

贵优步遍布多少城市呀？



猜猜 Uber 2011 年的 QPS



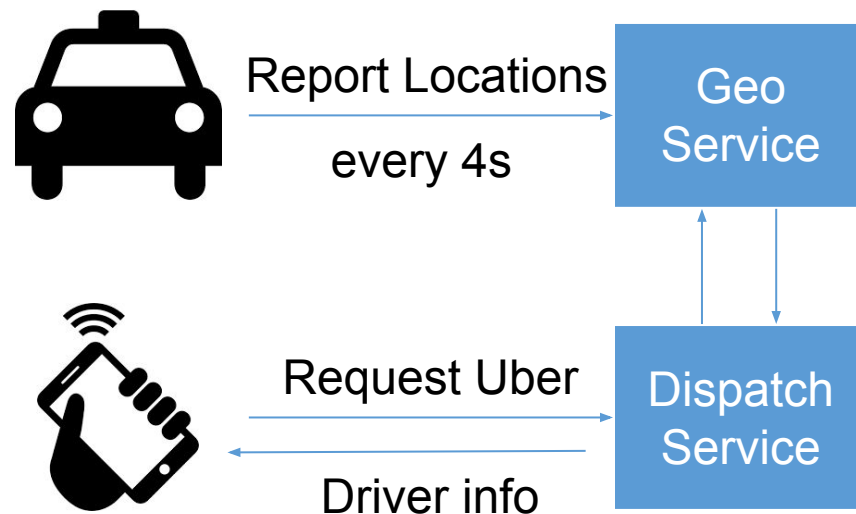
猜猜 Uber 2015 年的 QPS

Scenario - 设计得多牛

- 假设问到 20 万司机 **同时** 在线
 - Driver QPS = $200k / 4 = 50k$
 - Driver report locations by every 4 seconds
 - Peak Driver QPS = $50k * 3 = 150k$
 - Uber 官方自己的说法: 2015 新年夜的 Peak QPS 是 170K
 - Read More: <http://bit.ly/1FBSgMK>
 - Rider QPS 可以忽略
 - 无需随时汇报位置
 - 一定远小于 Driver QPS
- Uber 自己定的设计目标支持 1M QPS
 - 这些数字是多少并不重要, 你算给面试官看就好了
- 存储估算
 - 假如每条 Location 都记录: $200k * 86400 / 4 * 100\text{bytes}$ (每条位置记录) $\sim 0.5\text{ T} / \text{天}$
 - 假如只记录当前位置信息: $200k * 100\text{ bytes} = 20\text{ M}$

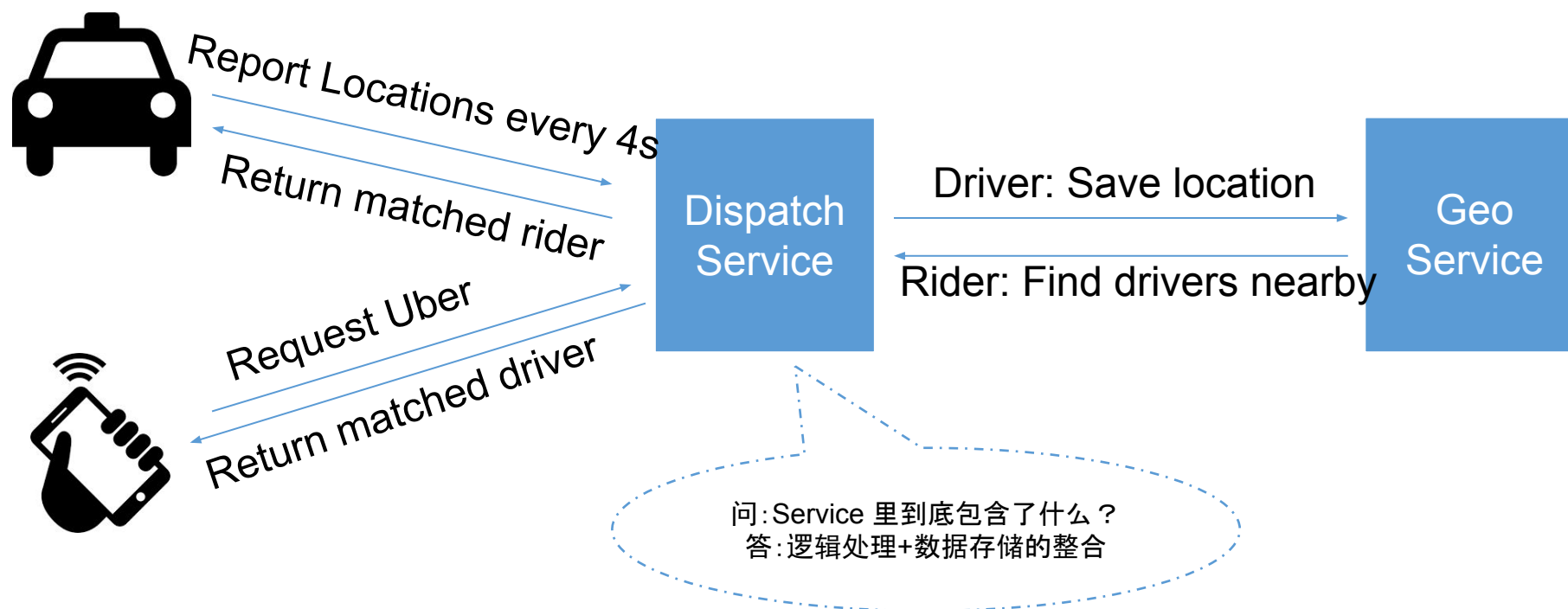
初步感觉: 150k 的写操作是不容小觑的
必须找一个写速度快的存储!

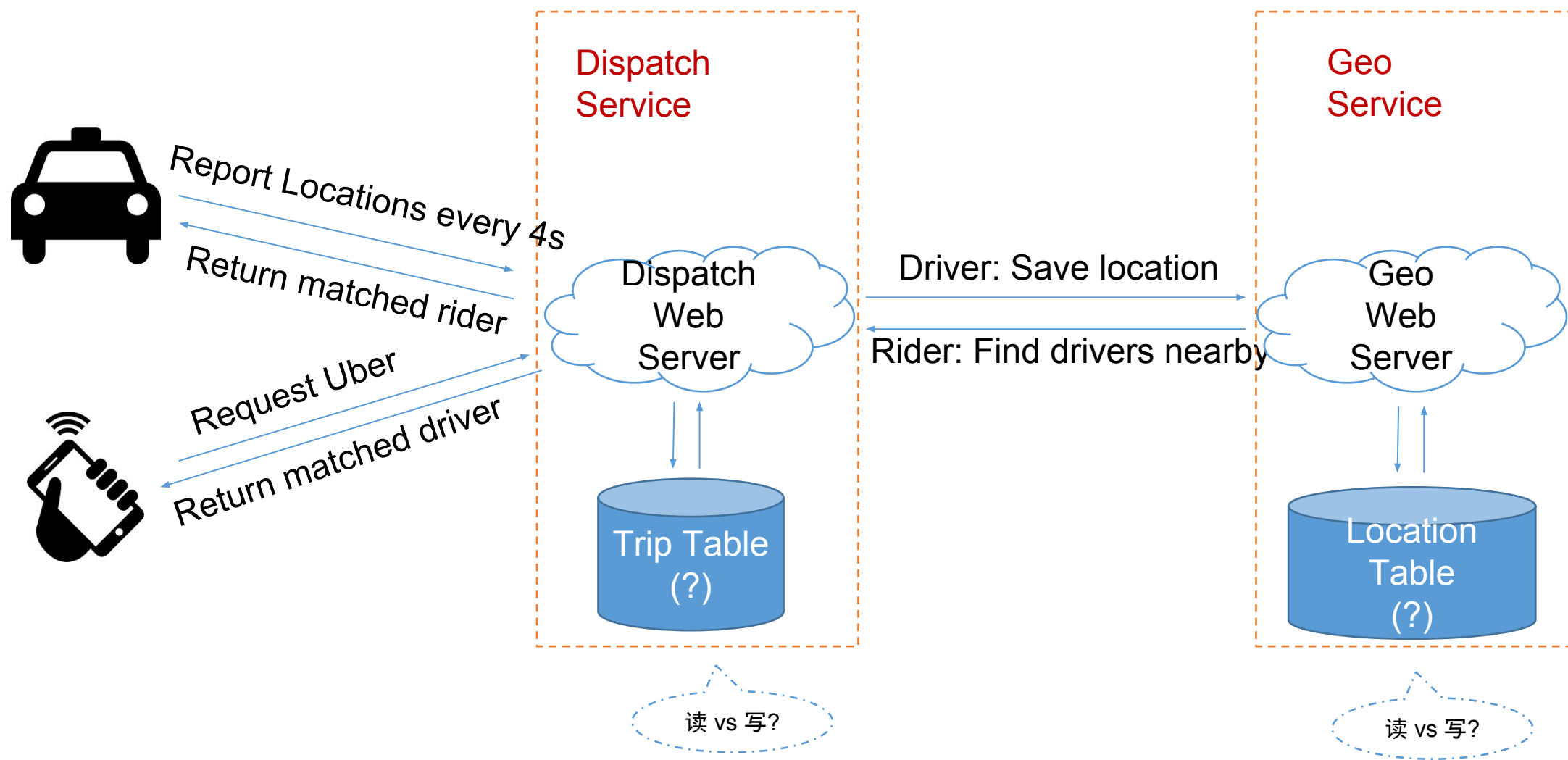
- Uber 主要干的事情就两件
 - 记录车的位置 GeoService
 - 匹配打车请求 DispatchService



这张图里漏了什么？

- Driver 如何获得打车请求？
 - Report location 的同时, 服务器顺便返回匹配上的打车请求






```
class Trip {  
    public Integer tripId;  
    public Integer driverId, riderId;  
    public Double Latitude, Longitude;  
    public Integer status;  
    public Datetime createdAt;  
}  
  
class Location {  
    public Integer driverId;  
    public Double Latitude, Longitude;  
    public Datetime updatedAt;  
}
```

Trip Table	type	comments
id	pk	primary key
rider_id	fk	User id
driver_id	fk	User id
lat	float	Latitude 纬度
lng	float	Longitude 经度
status	int	New request / waiting for driver / on the way to pick up / in trip / cancelled / ended
created_at	timestamp	

Location Table	type	comments
driver_id	fk	Primary key
lat	fk	纬度
lng	fk	经度
updated_at	timestamp	存最后更新的时间, 这样知道司机是不是掉线了

LBS 类系统的难点： 如何存储和查询地理位置信息？

如，查询某个乘客周围5公里内的司机



```
SELECT * FROM Location
WHERE lat < myLat + delta
      AND lat > myLat - delta
      AND lng < myLng + delta
      AND lng > myLng - delta;
```

——这个基本等同于扫描了一遍所有的数据

Storage 存储 —— 地理位置信息的存储与查询

- Google S2

- Read more: <http://bit.ly/1WgMpSJ>
- Hilbert Curve: <http://bit.ly/1V16HRa>
- 将地址空间映射到 2^{64} 的整数
- 特性: 如果空间上比较接近的两个点, 对应的整数也比较接近
- Example: $(-30.043800, -51.140220) \rightarrow 10743750136202470315$

更精准, 库函数API丰富

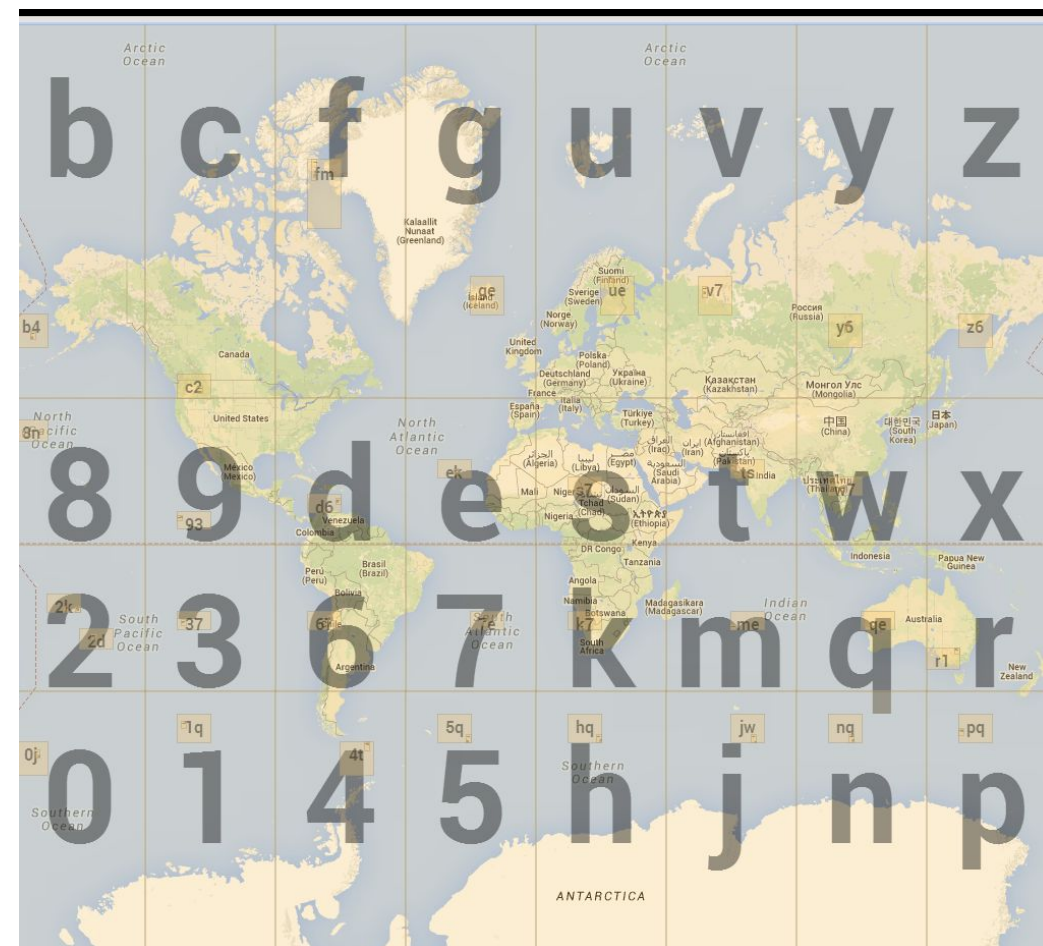
- Geohash

- Read more: <http://bit.ly/1S0Qzeo>
- Peano Curve
- Base32: 0-9, a-z 去掉 (a,i,l,o)
 - 为什么用 base32 ? 因为刚好 2^5 可以用 5 位二进制表示
- 核心思路二分法
- 特性: 公共前缀越长, 两个点越接近
- Example: $(-30.043800, -51.140220) \rightarrow 6feth68y4tb0$

比较简单, 准确度差一些

- Examples:
- LinkedIn HQ: 9q9hu3hhsjxx
- Google HQ: 9q9hvu7wbq2s
- Facebook HQ: 9q9j45zvr0se

geohash length	lat bits	lng bits	lat error	lng error	km error
1	2	3	± 23	± 23	± 2500
2	5	5	± 2.8	± 5.6	± 630
3	7	8	± 0.70	± 0.7	± 78
4	10	10	± 0.087	± 0.18	± 20
5	12	13	± 0.022	± 0.022	± 2.4
6	15	15	± 0.0027	± 0.0055	± 0.61
7	17	18	± 0.00068	± 0.00068	± 0.076
8	20	20	± 0.000085	± 0.00017	± 0.019



- 北海公园: lat=39.928167, lng=116.389550
- 二分(-180,180) 逼近经度┆左半部记0┆右半部记1
 - (-180, 180)里116.389550在右半部 → 1
 - (0, 180)里116.389550在右半部 → 1
 - (90, 180)里116.389550在左半部 → 0
 - (90, 135)里116.389550在右半部 → 1
 - (112.5, 135)里116.389550在左半部 → 0
- 二分(-90,90) 逼近纬度, 下半部记0, 上半部记1
 - (-90,90) 里 39.928167 在上半部 → 1
 - (0,90) 里 39.928167 在下半部 → 0
 - (0,45) 里 39.928167 在上半部 → 1
 - (22.5,45) 里 39.928167 在上半部 → 1
 - (33.75,45) 里 39.928167 在上半部 → 1
 - ... (还可以继续二分求获得更多的精度³)

课后作业:

<http://www.lintcode.com/problem/geohash/>

先经后纬, 经纬交替

1110011101

W X

查询Google半径2公里内的车辆

- 找到精度误差 > 2公里的最长长度

geohash length	lat bits	lng bits	lat error	lng error	km error
1	2	3	± 23	± 23	± 2500
2	5	5	± 2.8	± 5.6	± 630
3	7	8	± 0.70	± 0.7	± 78
4	10	10	± 0.087	± 0.18	± 20
5	12	13	± 0.022	± 0.022	± 2.4
6	15	15	± 0.0027	± 0.0055	± 0.61
7	17	18	± 0.00068	± 0.00068	± 0.076
8	20	20	± 0.000085	± 0.00017	± 0.019



- Google HQ: 9q9hvu7wbq2s
- 找到位置以9q9hv以开头的所有车辆



怎样在数据库中实现该功能？

- SQL 数据库
 - 首先需要对 geohash 建索引
 - CREATE INDEX on geohash;
 - 使用 Like Query
 - SELECT * FROM location WHERE geohash LIKE `9q9hv%`;
- NoSQL - Cassandra
 - 将 geohash 设为 column key
 - 使用 range query (9q9hv0, 9q9hvz)
- NoSQL - Redis / Memcached
 - Driver 的位置分级存储
 - 如 Driver 的位置如果是 9q9hvt, 则存储在 9q9hvt, 9q9hv, 9q9h 这 3 个 key 中
 - 6位 geohash 的精度已经在一公里以内, 对于 Uber 这类应用足够了
 - 4位 geohash 的精度在20公里以上了, 再大就没意义了, 你不会打20公里以外的车
 - key = 9q9hvt, value = **set** of drivers in this location



从数据存储与查询的角度
哪种结构更合？

- 能够熟悉每种数据存储结构的特性，对于面试十分加分！
- SQL 可以，但相对较慢
 - 原因1: Like query 很慢，应该尽量避免
 - 即便有index，也很慢
 - 原因2: Uber 的应用中，Driver 需要实时 Update 自己的地理位置
 - 被index的column并不适合经常被修改
 - B+树不停变动，效率低
- NoSQL – Cassandra 可以，但相对较慢
 - 原因：Driver 的地理位置信息更新频次很高
 - Column Key 是有 index 的，被 index 的 column 不适合经常被“修改”
- NoSQL – Memcached 并不合适
 - 原因1: Memcached 没有持久化存储，一旦挂了，数据就丢失
 - 原因2: Memcached 并不原生支持 set 结构
 - 需要读出整个 set，添加一个新元素，然后再把整个set 赋回去



如果是设计Yelp呢？

NoSQL - Redis

数据可持久化

原生支持list, set等结构

读写速度接近内存访问速度 >100k QPS



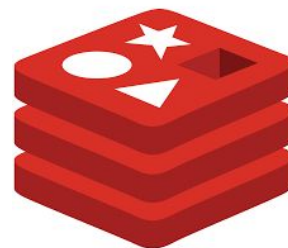
打车用户的角度

- 用户发出打车请求, 查询给定位置周围的司机
 - $(lat, lng) \rightarrow geohash \rightarrow [driver1, driver2, \dots]$
 - 先查6位的 geohash 找0.6公里以内的
 - 如果没有再查5位的 geohash 找2.4公里以内的
 - 如果没有再查4位的 geohash 找20公里以内的

Location Table	
key	geohash
value	{driver1_id, driver2_id, driver3_id ...}

- 匹配司机成功, 用户查询司机所在位置
 - $driver1 \rightarrow (lat, lng)$

Driver Table	
key	driver_id
value	(lat, lng, status, updated_at, trip_id)



指向UserTable, UserTable存在其他数据库中, 可以是SQL数据库

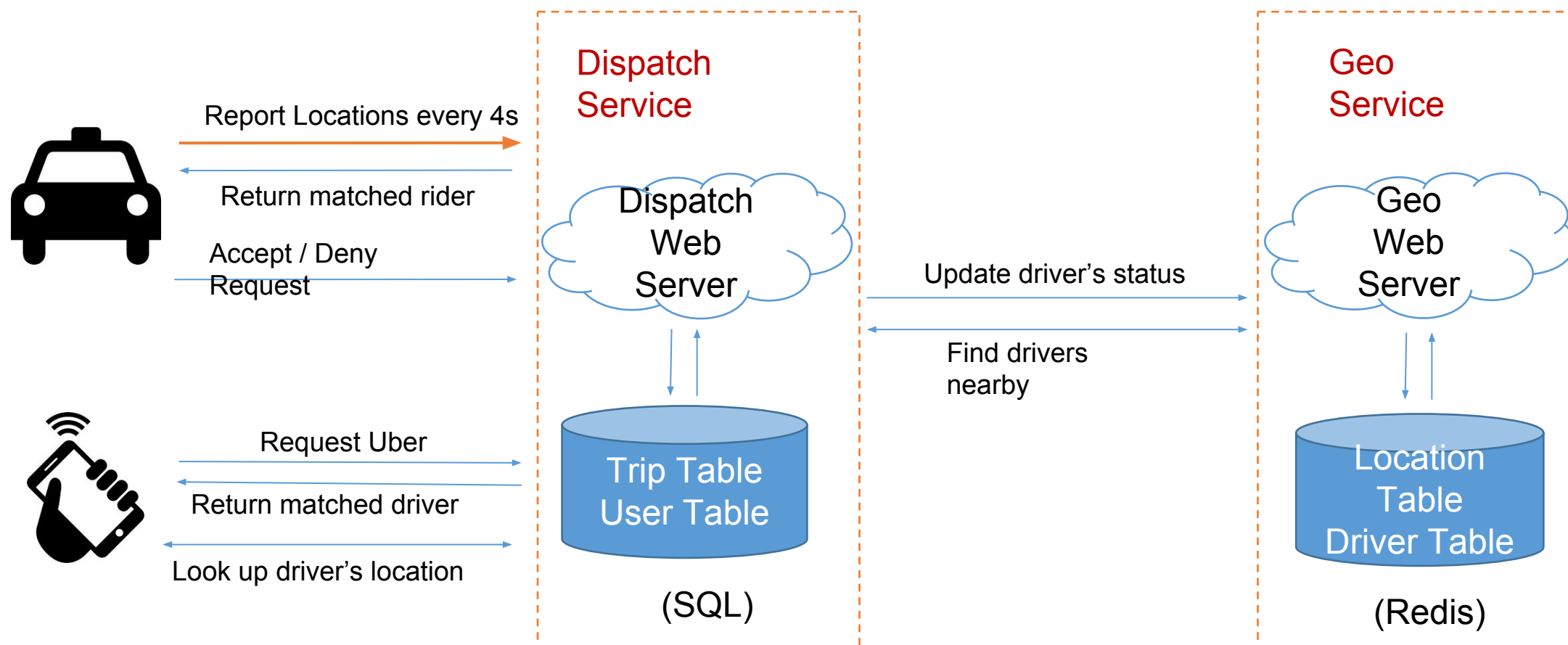
司机的角度

- 司机汇报自己的位置
 - 计算当前位置 lat, lng 的 geohash
 - geohash4, geohash5, geohash6
 - 查询自己原来所在的位置
 - geohash4', geohash5', geohash6'

} 对比是否发生变化
并将变化的部分在 Redis 中进行修改

 - 在 Driver Table 中更新自己的最后活跃时间
- 司机接受打车请求
 - 修改 Trip 状态
 - 用户发出请求时就已经在 Trip Table 中创建一次旅程 T 并 Match 上最近的司机
 - 在 Driver Table 中标记自己的状态进入不可用状态
- 司机完成接送 T 结束一次 Trip
 - 在 Trip Table 中修改旅程状态
 - 在 Driver Table 中标记自己的状态进入可用状态

1. 乘客发出打车请求, 服务器创建一次Trip
 - 将 trip_id 返回给用户
 - 乘客每隔几秒询问一次服务器是否匹配成功
2. 服务器找到匹配的司机, 写入Trip, 状态为等待司机回应
 - 同时修改 Driver Table 中的司机状态为不可用, 并存入对应的 trip_id
3. 司机汇报自己的位置
 - 顺便在 Driver Table 中发现有分配给自己的 trip_id
 - 去 Trip Table 查询对应的 Trip, 返回给司机
4. 司机接受打车请求
 - 修改 Driver Table, Trip 中的状态信息
 - 乘客发现自己匹配成功, 获得司机信息
5. 司机拒绝打车请求
 - 修改 Driver Table, Trip 中的状态信息, 标记该司机已经拒绝了该trip
 - 重新匹配一个司机, 重复第2步



Take a break

5 分钟后回来

Scale 拓展

看看有哪些问题没有解决, 需要优化
出现故障怎么办

有什么隐患？

需求是150k QPS

Redis 的读写效率 > 100 QPS

是不是1-2台就可以了？

Interviewer: Redis server is down?

随便挂一台，分分钟损失几百万\$



DB Sharding

目的1: 分摊流量

目的2: Avoid Single Point Failure



按照什么来Sharding?

按照城市Sharding

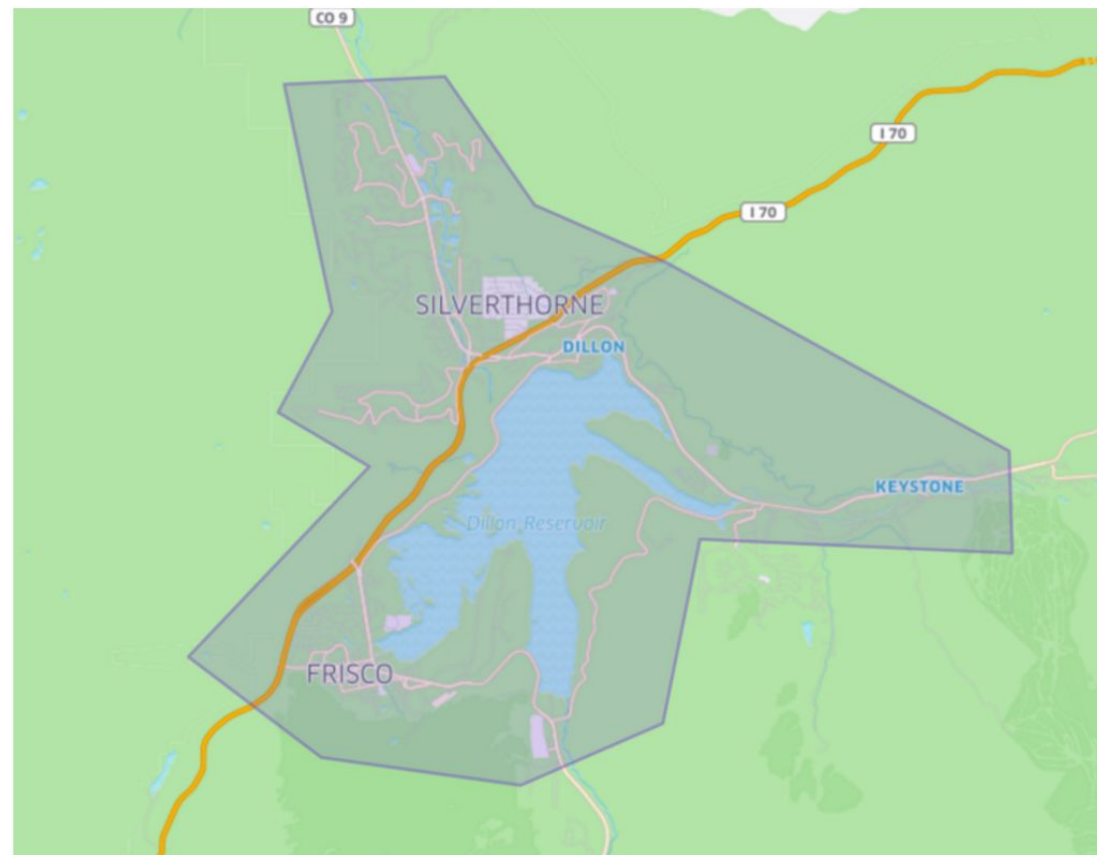
难点1: 如何定义城市?

难点2: 如何根据位置信息知道用户在哪个城市?



为什么不能按照其他的比如 user_id 来 sharding?

- 用多边形代表城市的范围
- 问题本质: 求一个点是否在多边形内
 - 计算几何问题
- 城市的数目: 400个
- 乘客站在两个城市的边界上怎么办?
 - 找到乘客周围的2-3个城市
 - 这些城市不能隔太远以至于车太远
 - 汇总多个城市的查询结果
 - 这种情况下司机的记录在存哪个城市关系不大



Interviewer: How to check rider is in Airport?

同样可以用Geo Fence

类似机场这样的区域有上万个，直接 $O(N)$ 查询太慢

分为两级Fence查询，先找到城市，再在城市中查询Airport Fence

Read More: <http://ubr.to/20qK4F4>

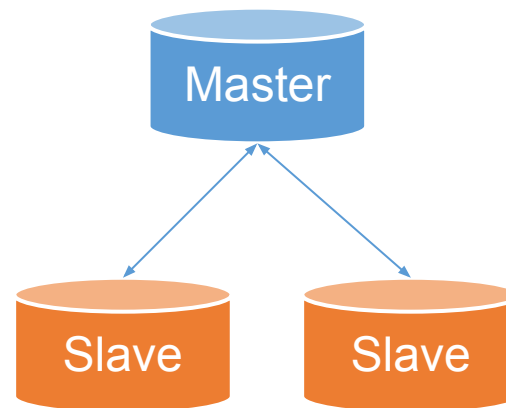
Interviewer: How to reduce impact on db crash?

多台 Redis 虽然能减少损失
但是再小的机器挂了，都还是会影响



如何减小损失？

- 方法1: Replica by Redis
 - Master - Slave
- 方法2: Replica by yourself
 - 底层存储的接口将每份数据写3份
 - sharding key 从 123 (city_id) 扩展为
 - 123-0
 - 123-1
 - 123-2
 - 读取的时候, 从任意一份 replica 读取数据
 - 读不到的时候, 就从其他的 replica 读
 - 三份 replica 极有可能存在3个不同的机器上, 同时挂掉的概率很小很小
 - 当然也有可能不巧存在一个机器上, 这个问题如何解决请参考Dynamo DB的论文
- 方法3: 让更强大的NoSQL数据库帮你处理 —— Riak / Cassandra
 - 既然一定需要用多台机器了, 那么每台的流量也就没有150k QPS这么高了
 - 用 Riak / Cassandra 等NoSQL数据库, 能够帮助你更好的处理 Replica 以及机器挂掉之后恢复的问题



- 分析出 Uber 是一个写密集的应用
 - 与大部分应用都是读密集不一样
- 解决 LBS 类应用的核心问题 – Geohash / Google S2
 - 如何存储司机的地理位置
 - 如何查询乘客周围的车
- 分析出一个 Work Solution, 说明整个打车的流程
- 分析出按照城市进行 Sharding
- 知道如何做 Geo Fence 的查询
- 通过分析选择合适的数据库
- 考虑到单点失效(多机)与数据故障(备份)如何解决
- 深入理解 NoSQL DB 的实现以及了解 Ring Pop的实现 *
 - 只能靠大家自己读论文了 <http://bit.ly/1mDs0Yh>
- 设计 Uber Pool / Uber Eat *

WEAK

HIRED



HIRED

小调查



发现问题和解决问题哪个更重要？

系统设计面试常见误区

以为必须想到完美的解决方案

- Rate Limiter
 - http://www.mitbbs.com/article_t/JobHunting/33166641.html
- 算法面试
 - 注重程序的实现
 - 注重解决问题的能力
- 系统面试
 - 注重思考的过程
 - 注重发现问题的能力

- 一上来就抛出各种高大上但是你根本不熟悉的关键词
 - 给出一个可行方案比关键词更可靠
- 以为系统设计面试不考编程
 - 很多系统设计问题, 都包含了算法的设计与实现
 - Twitter (Merge k Sorted Arrays) [九章算法班]
 - Tiny Url (Base62)
 - Google Suggestion (Trie) [算法强化班]
 - Web Crawler (Multi Threading)
 - Amazon Top 10 Products (Hash Heap) [算法强化班]
- 以为答出贵公司的做法就可以通过面试
 - 公司自己的技术自己最熟悉, 也就知道自己所用技术的缺陷
 - 答一个面试官不太熟悉的内容, 比较容易“忽悠”
- 过分注重解决问题, 而不是发现问题
 - 系统设计面试的得分点是: 你有没有想到会发生这个情况, 而不是你知不知道这个情况怎么解决

- <http://www.lintcode.com/problem/mini-yelp/>
- <http://www.lintcode.com/problem/geohash/>
- <http://www.lintcode.com/problem/geohash-ii/>
- <http://www.lintcode.com/problem/mini-cassandra/>

附录: 扩展阅读

- Uber's Early Architecture
 - <http://bit.ly/1Q1IzGL> [Easy] [Video]
- Scaling Uber's Real-time Market Platform
 - <http://bit.ly/1FBSgMK> [Medium] [Video]
- RingPop
 - <http://ubr.to/1S47b8g> [Hard] [Blog]
 - <http://bit.ly/1Yg2dnd> [Hard] [Video]
- Point in polygon
 - <http://bit.ly/1N1Zjlu> wiki
- Dynamo DB
 - <http://bit.ly/1mDs0Yh> [Hard] [Paper]