

走进系统设计与新鲜事系统

Introducing System Design & News Feed System

课程版本: v5.2 本节主讲人: 东邪



扫描二维码关注微信小程序/公众号
获取第一手求职资料

版权声明

九章的所有课程均受法律保护，不允许录像与传播录像
一经发现，将被追究法律责任和赔偿经济损失

- 曾就职于亿级日活跃用户(DAU)的社交网络公司
- 多年基础架构(Infrastructure)经验
- 多年网站开发(Web Development)经验
- 北美顶尖IT企业Offer数10+





西毒老师
硅谷顶尖IT工程师
擅长搜索引擎系统



北丐老师
硅谷顶尖IT工程师
擅长分布式系统

面试

- 面试者
- 一个



成长

- 成为系统架构师
- 寻找兴趣点与发展方向
- 胜任更高难度的工作

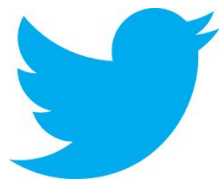
创业

- 了解业界的经验
- 了解挑战与机会

- 系统设计面试的两种形式
- 常见的系统设计面试问题
- 从 Design Twitter 介绍什么是系统设计
- 系统设计面试的常见错误
- 系统设计面试的评分标准
- 系统设计的九阴真经—— **4S** 分析法
- 后续课程安排

- 设计某某系统 Design XXX System

- 设计微博 Design Twitter
- 设计人人 Design Facebook
- 设计滴滴 Design Uber
- 设计微信 Design Whatsapp
- 设计点评 Design Yelp
- 设计短网址系统 Design Tiny URL
- 设计NoSQL数据库 Design NoSQL



- 找问题 Trouble Shouting

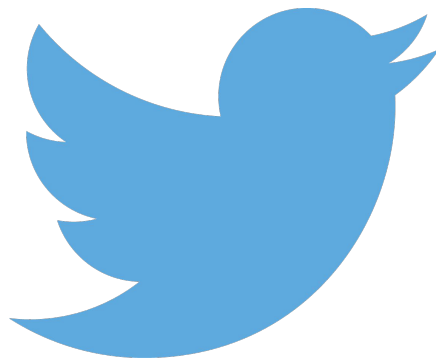
- 网站挂了怎么办 What happened if we can not access a website
- 网站太慢怎么办 What happened if a webserver is too slow
- 流量增长怎么办 What should we do for increasing traffic

面试官：请设计推特

Interviewer: Please design twitter



你第一句对面试官说的话是什么？



常见错误：关键词大师

Load Balancer, Memcache, NodeJS, MongoDB, MySQL,
Sharding, Consistent Hashing, Master Slave, HDFS, Hadoop ...

你想过没有：或许现在只有2个用户呢？

系统设计面试的评分标准

<http://www.jiuzhang.com/qa/1246/>

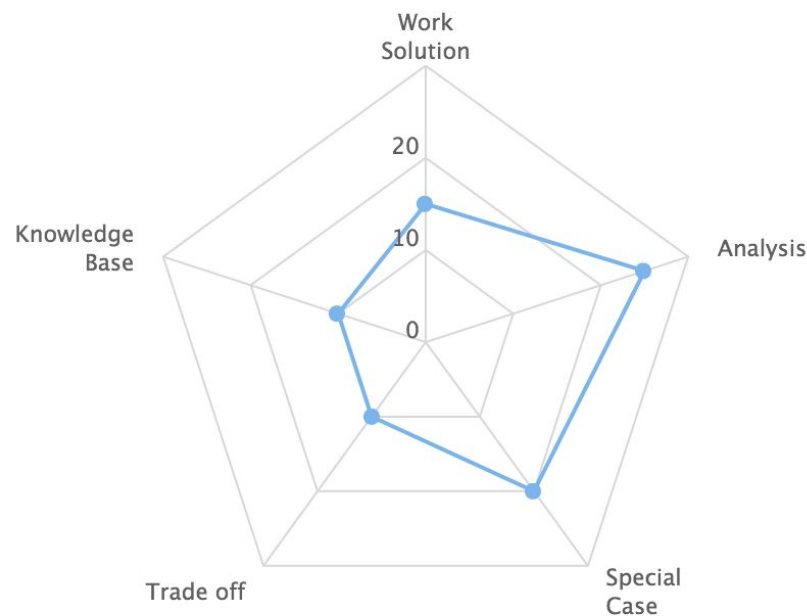
可行解 Work Solution 25%

特定问题 Special Case 20%

分析能力 Analysis 25%

权衡 Tradeoff 15%

知识储备 Knowledge Base 15%



系统设计问题的 **4S** 分析法

Senario, **S**ervice, **S**torage, **S**cale



- **S**enario 场景
 - 说人话: 需要设计哪些功能, 设计得多牛
 - Ask / Features / QPS / DAU / Interfaces
- **S**ervice 服务
 - 说人话: 将大系统拆分为小服务
 - Split / Application / Module
- **S**torage 存储
 - 说人话: 数据如何存储与访问
 - Schema / Data / SQL / NoSQL / File System
- **S**cale 升级
 - 说人话: 解决缺陷, 处理可能遇到的问题
 - Sharding / Optimize / Special Case

Work Solution
NOT Perfect Solution

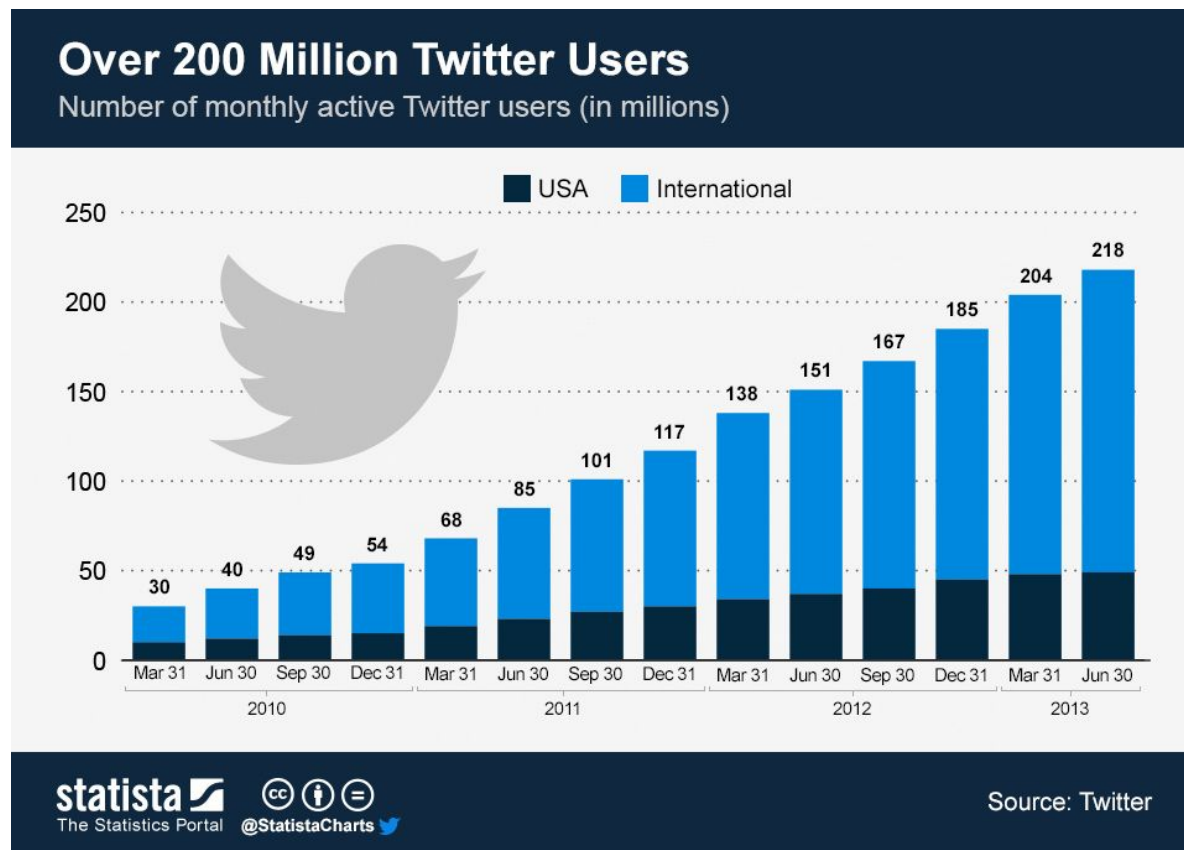
Scenario 场景

需要设计哪些功能，设计得多牛

1. Ask 问面试官
2. Analysis 分析



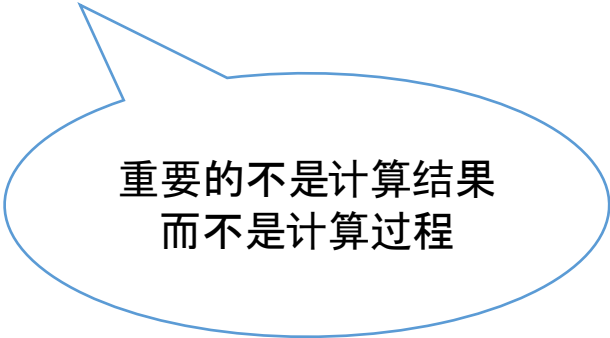
- 询问面试官：
 - 需要设计哪些功能(也可以自己想)
 - 需要承受多大的访问量？
 - 日活跃用户 Daily Active Users (DAU)
 - Twitter: MAU 313M, DAU ~150M+
 - Read more: <http://bit.ly/1Kml0M7>



- 第一步 Step 1: **Enumerate**
 - 说人话: 把Twitter的功能一个个罗列出来
 - Register / Login
 - User Profile Display / Edit
 - Upload Image / Video *
 - Search *
 - Post / Share a tweet
 - Timeline / News Feed
 - Follow / Unfollow a user
- 第二步 Step 2: **Sort**
 - 说人话: 选出核心功能, 因为你不可能这么短的时间什么都设计
 - Post a Tweet
 - Timeline
 - News Feed
 - Follow / Unfollow a user
 - Register / Login



- 并发用户 Concurrent User
 - 日活跃 * 每个用户平均请求次数 / 一天多少秒 = $150M * 60 / 86400 \sim 100k$
 - 峰值 Peak = Average Concurrent User * 3 ~ 300k
 - 快速增长的产品 Fast Growing
 - MAX peak users in 3 months = Peak users * 2
- 读频率 Read QPS (Queries Per Second)
 - 300k
- 写频率 Write QPS
 - 5k



重要的不是计算结果
而不是计算过程

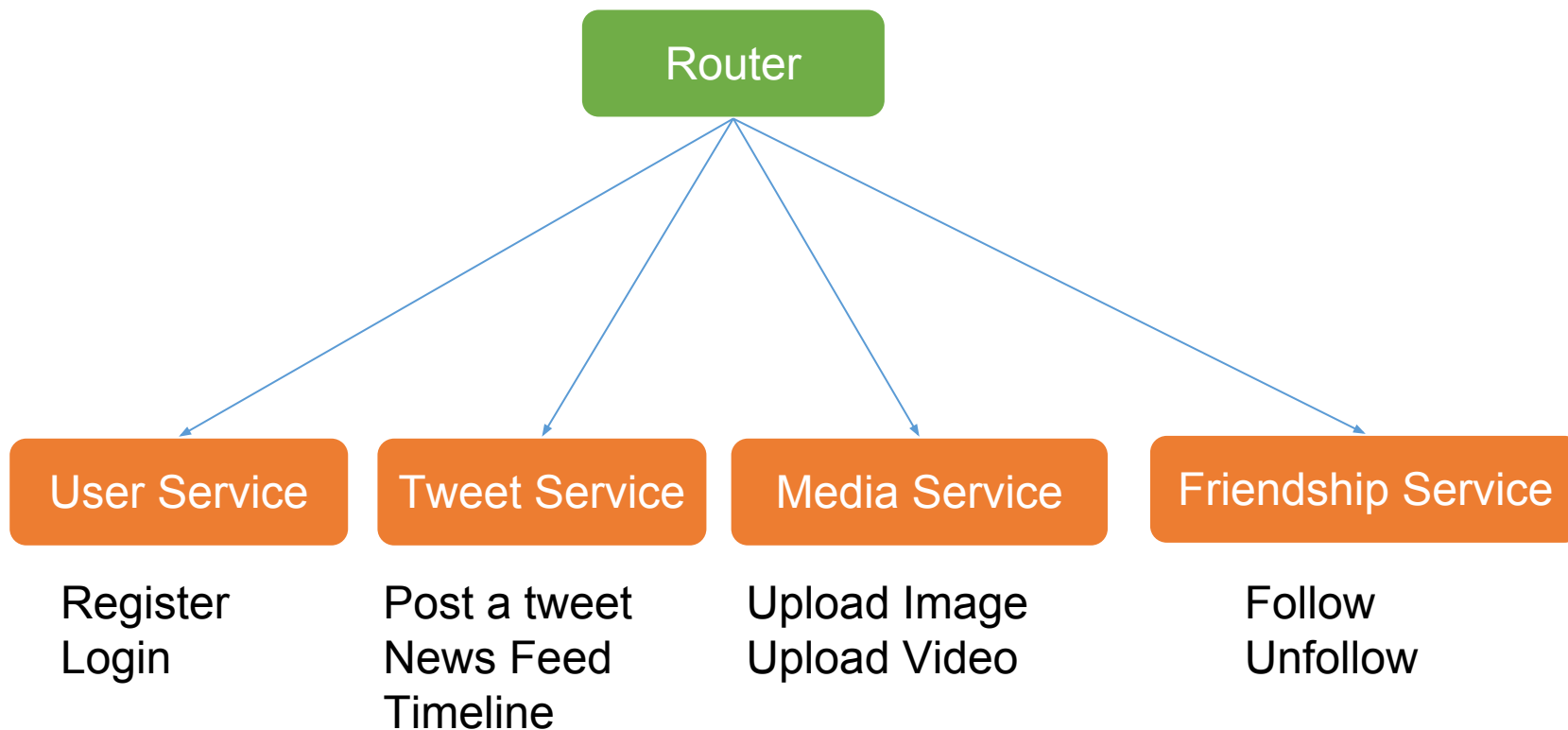
分析出 QPS 有什么用？

- QPS = 100
 - 用你的笔记本做 Web 服务器就好了
- QPS = 1k
 - 用一台好点的 Web 服务器就差不多了
 - 需要考虑 Single Point Failure
- QPS = 1m
 - 需要建设一个1000台 Web 服务器的集群
 - 需要考虑如何 Maintenance(某一台挂了怎么办)
- QPS和 Web Server (服务器) / Database (数据库) 之间的关系
 - 一台 Web Server 约承受量是 1k 的 QPS (考虑到逻辑处理时间以及数据库查询的瓶颈)
 - 一台 SQL Database 约承受量是 1k 的 QPS(如果 JOIN 和 INDEX query比较多的话, 这个值会更小)
 - 一台 NoSQL Database (Cassandra) 约承受量是 10k 的 QPS
 - 一台 NoSQL Database (Memcached) 约承受量是 1M 的 QPS

Service 服务

将大系统拆分为小服务

1. Replay 重放需求
2. Merge 归并需求



Service 服务 – 将大系统拆分为小服务

- 第一步 Step 1: **Replay**
 - 重新过一遍每个需求, 为每个需求添加一个服务
- 第二步 Step 2: **Merge**
 - 归并相同的服务
- 什么是服务 Service?
 - 可以认为是逻辑处理的整合
 - 对于同一类问题的逻辑处理归并在一个 Service 中
 - 把整个 System 细分为若干个小的 Service

Storage 存储

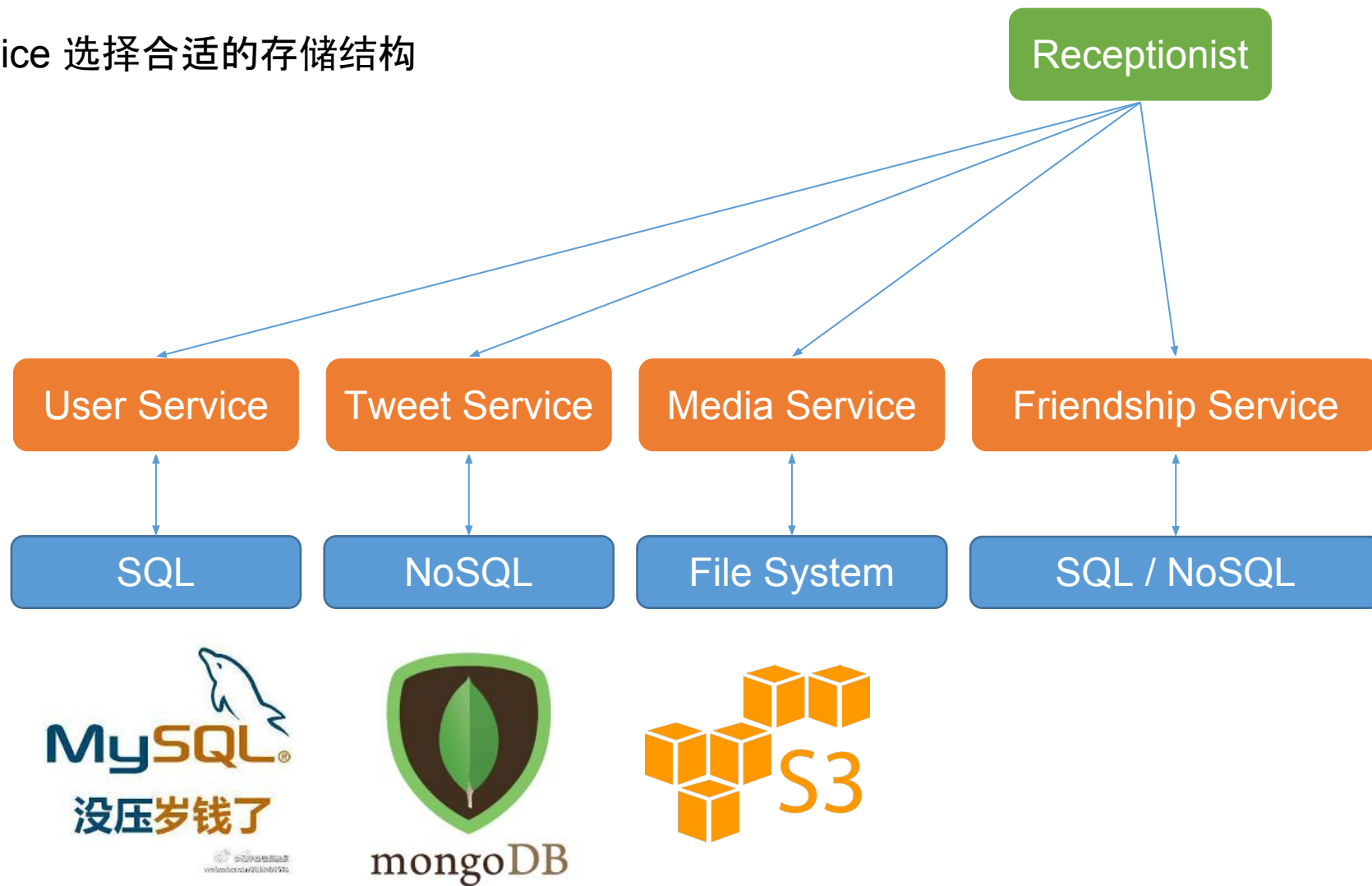
数据如何存储与访问

1. Select 为每个 Service 选择存储结构
2. Schema 细化表结构

- 关系型数据库 SQL Database
 - 小调查: Twitter的哪些信息适合放在关系型数据库中?
 - 用户信息 User Table
- 非关系型数据库 NoSQL Database
 - 小调查: Twitter的哪些信息适合放在非关系型数据库中?
 - 推文 Tweets
 - 社交图谱 Social Graph (followers)
- 文件系统 File System
 - 小调查: Twitter的哪些信息适合放在文件系统中?
 - 图片、视频 Media Files



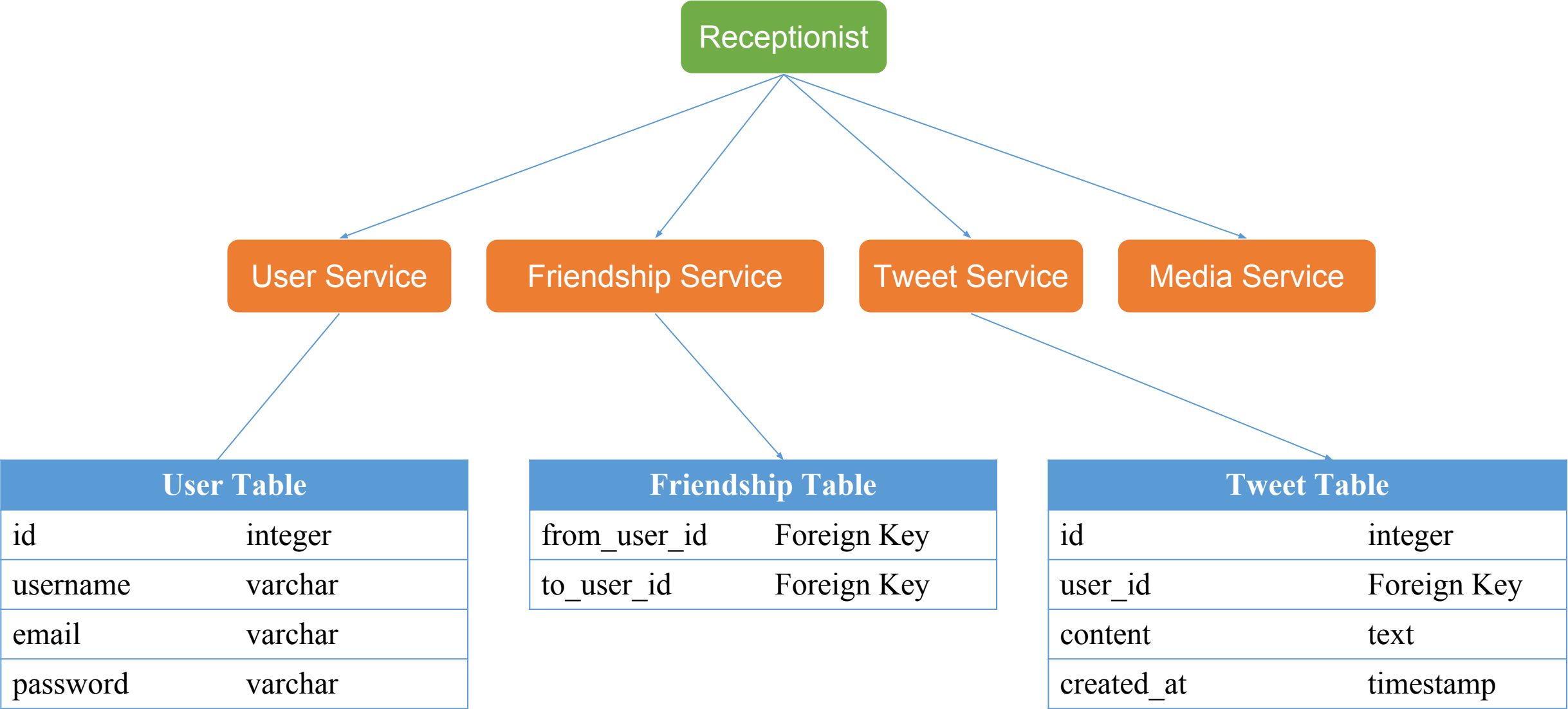
- 第一步 Step 1: **Select**
 - 为每个 Application / Service 选择合适的存储结构
- 第二步 Step 2: **Schema**
 - 细化数据表结构
- 程序 = 算法 + 数据结构
- 系统 = 服务 + 数据存储



Interviewer: Please design schema



请设计数据库的表结构



休息 5 分钟 Take a break



News Feed如何存取？



Lady Gaga发一个推文之后会发生什么？

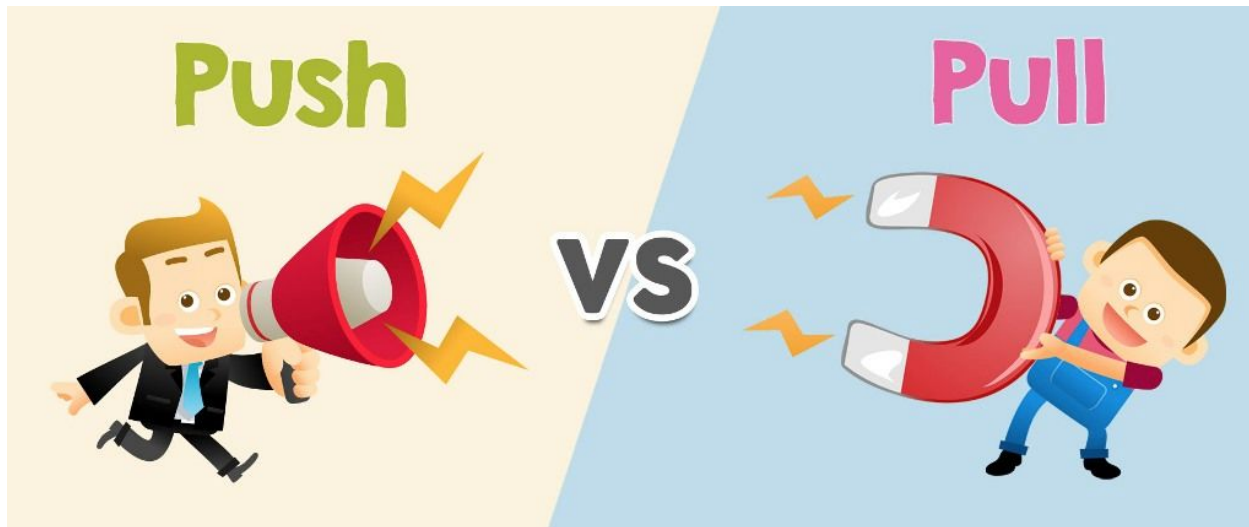


如何存取 Likes？



如何实现 Follow && Unfollow？

Interviewer: News Feed 如何存取？



- 什么是新鲜事 News Feed？
 - 你登陆 Facebook / Twitter / 朋友圈 之后看到的信息流
 - 你的所有朋友发的信息的集合
- 有哪些典型的新鲜事系统？
 - Facebook
 - Twitter
 - 朋友圈
 - RSS Reader
- 新鲜事系统的核心因素？
 - 关注与被关注
 - 每个人看到的新鲜事都是不同的



- 算法

- 在用户查看News Feed时, 获取每个好友的前100条Tweets, 合并出前100条News Feed
 - K路归并算法 Merge K Sorted Arrays

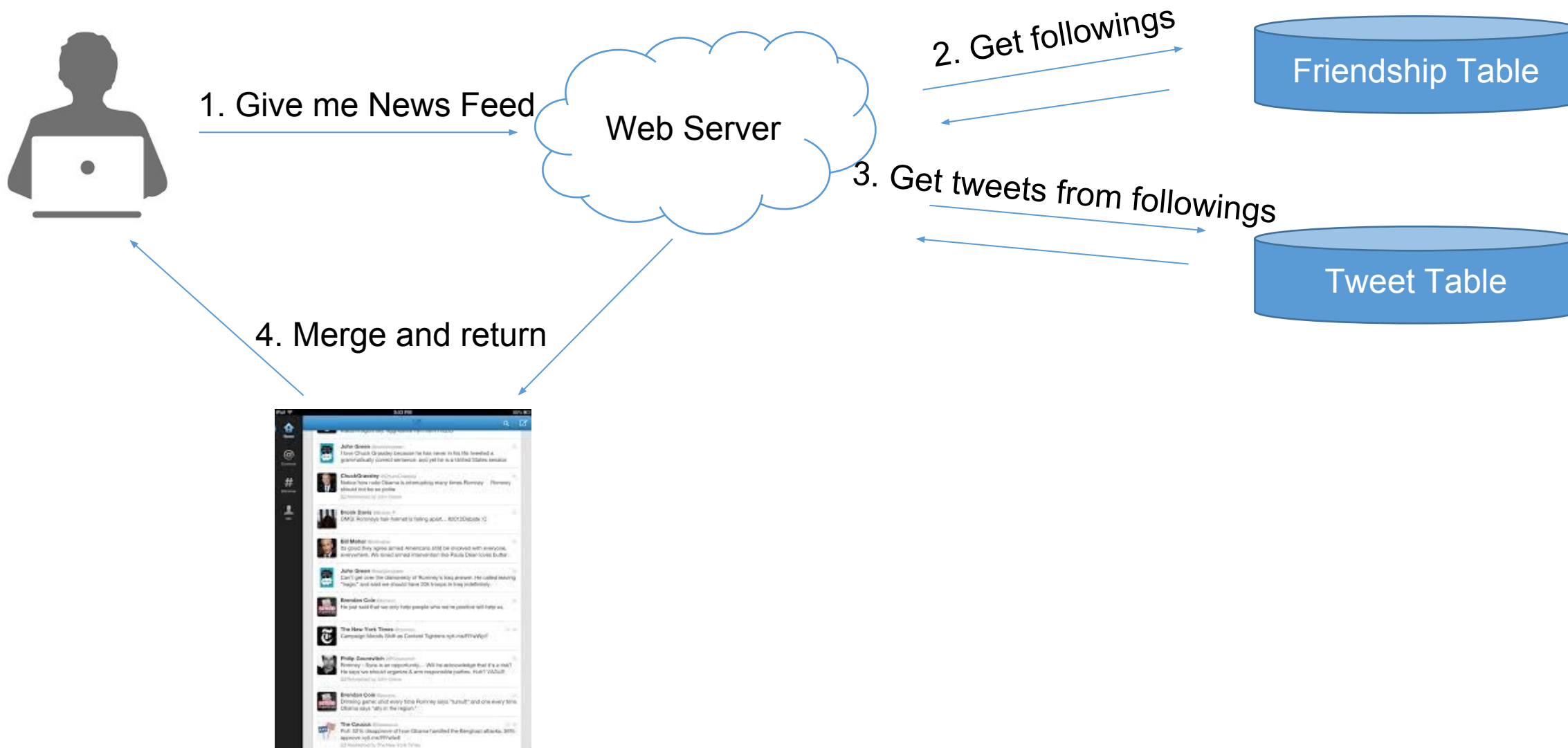
- 复杂度分析

- News Feed => 假如有N个关注对象, 则为N次DB Reads的时间 + K路归并时间(可忽略)
 - 为什么K路归并的时间可以忽略?
- Post a tweet => 1次DB Write的时间

Pull = 主动撩妹



Storage 存储 – Pull 原理图



Interviewer: Pull模型有什么缺陷么？



Pull 的缺陷是什么？

- getNewsFeed(request)
 - followings = DB.getFollowings(user=request.user)
 - news_feed = empty
 - for follow in followings:
 - tweets = DB.getTweets(follow.to_user, 100)
 - news_feed.merge(tweets)
 - sort(news_feed)
 - return news_feed
- postTweet(request, tweet)
 - DB.insertTweet(request.user, tweet)
 - return success

N次DB Reads非常慢
且发生在用户获得News
Feed的请求过程中

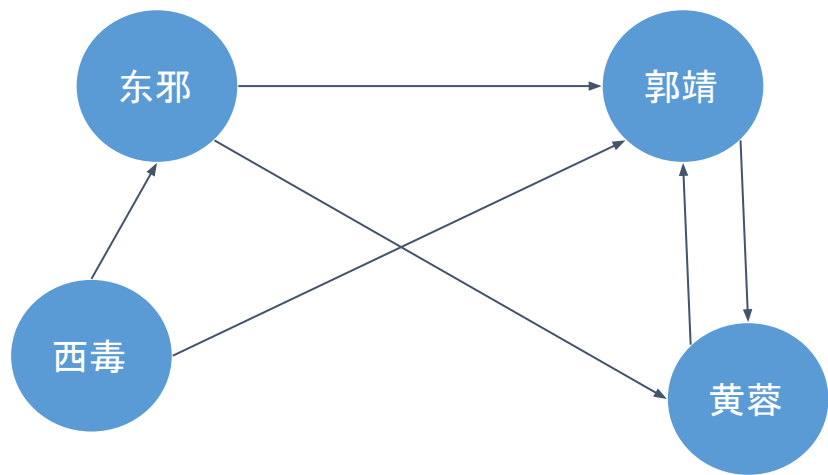
- 算法
 - 为每个用户建一个List存储他的News Feed信息
 - 用户发一个Tweet之后, 将该推文逐个推送到每个用户的News Feed List中
 - 关键词: **Fanout**
 - 用户需要查看News Feed时, 只需要从该News Feed List中读取最新的100条即可
- 复杂度分析
 - News Feed => 1次DB Read
 - Post a tweet => N个粉丝, 需要N次DB Writes
 - 好处是可以用异步任务在后台执行, 无需用户等待

Push = 坐等被撩

News Feed Table	
id	integer
owner_id	Foreign Key
tweet_id	Foreign Key
created_at	timestamp



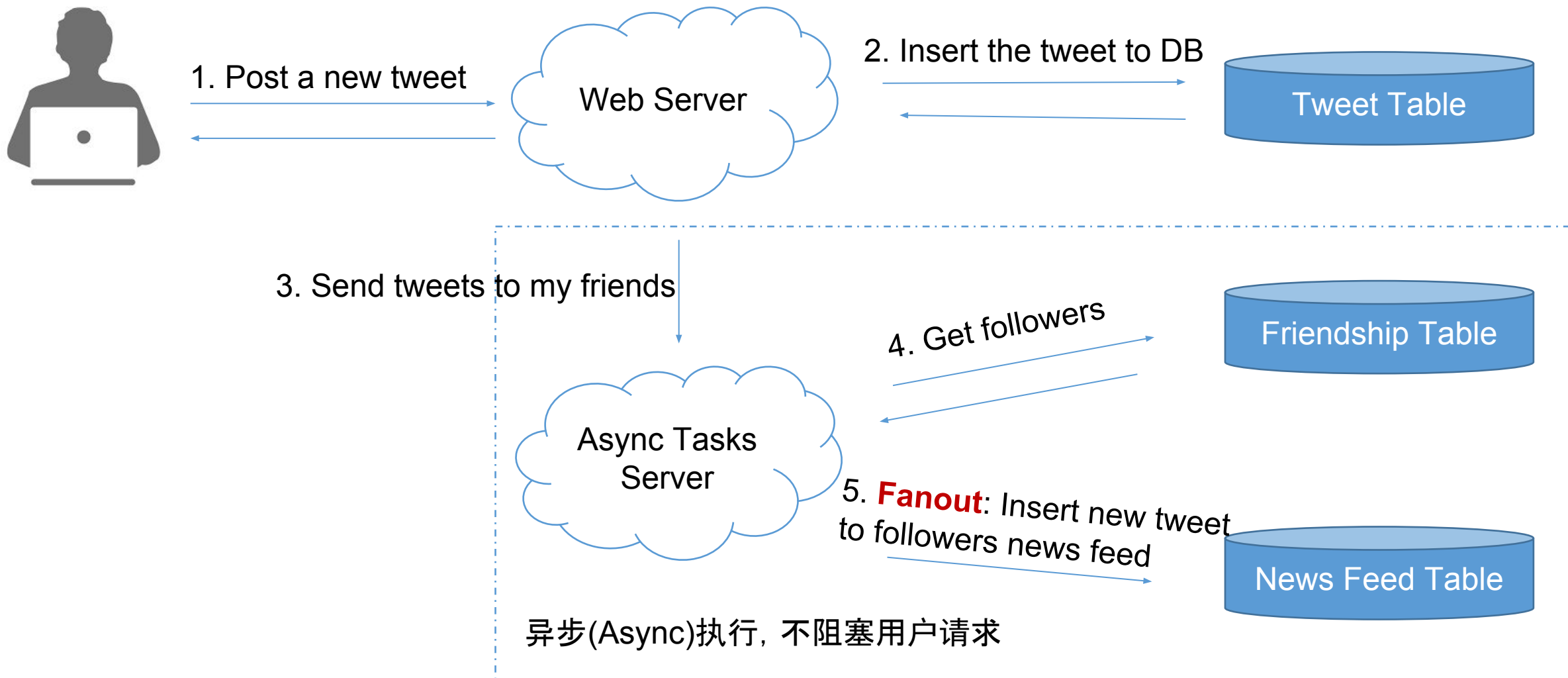
东邪西毒，郭靖黄蓉的好友关系如下：



东邪发了一条帖子：“好想念超风”
西毒发了一条帖子：“好想念嫂子”
郭靖发了一条帖子：“不知道华筝怎么样了”
黄蓉发了一条帖子：“男人都不是好东西”

News Feed Table			
id	owner_id	tweet_id	created_at
1	东邪	东邪:好想念超风	2016/10/15 16:30:00
2	西毒	东邪:好想念超风	2016/10/15 16:30:00
3	西毒	西毒:好想念嫂子	2016/10/15 16:35:00
4	郭靖	郭靖:不知道华筝怎么样了	2016/10/15 17:00:00
5	东邪	郭靖:不知道华筝怎么样了	2016/10/15 17:00:00
6	西毒	郭靖:不知道华筝怎么样了	2016/10/15 17:00:00
7	黄蓉	郭靖:不知道华筝怎么样了	2016/10/15 17:00:00
8	黄蓉	黄蓉:男人都不是好东西	2016/10/15 18:00:00
9	郭靖	郭靖:男人都不是好东西	2016/10/15 18:00:00
10	东邪	郭靖:男人都不是好东西	2016/10/15 18:00:00

黄蓉登陆“射雕APP”之后可以看到的所有帖子通过一句 SQL 查询可以拿到：
select * from news_feed_table where owner_id=黄蓉 order_by created_at desc limit 20;



Interviewer: Push模型有缺陷么？



Push 的缺陷是什么？

- getNewsFeed(request)
 - return DB.getNewsFeed(request.user)
- postTweet(request, tweet_info)
 - tweet = DB.insertTweet(request.user, tweet_info)
 - AsyncService.fanoutTweet(request.user, tweet)
 - return success
- AsyncService::fanoutTweet(user, tweet)
 - followers = DB.getFollowers(user)
 - for follower in followers:
 - DB.insertNewsFeed(tweet, follower)

异步执行

followers的数目可能很大



Pull vs Push

哪一种方式更好？
你喜欢撩妹还是被撩？



- 热门Social App的模型
 - Facebook – Pull
 - Instagram – Push + Pull
 - Twitter – Pull
- 误区
 - 不坚定想法, 摇摆不定
 - 不能表现出Tradeoff的能力
 - 无法解决特定的问题



- 用过前3个步骤的分析, 我们已经得到了一个可行方案
- Scenario 场景
 - 和面试官讨论
 - 搞清楚需要设计哪些功能
 - 并分析出所设计的系统大概所需要的 Concurrent Users / QPS / Memory / Storage 等
- Service 服务
 - 合并需要设计功能, 相似的功能整合为一个Service
- Storage 存储
 - 对每个 Service 选择合适的存储结构
 - 细化数据表单
 - 画图展示数据存储和读取的流程
- 得到一个 Work Solution 而不是 Perfect Solution
- 这个Work Solution 可以存在很多待解决的缺陷

Scale 扩展

How to Scale? 系统如何优化与维护

1. Optimize 优化
2. Maintenance 维护

Scale 扩展 - 如何优化系统

- 第一步 Step 1: **Optimize**
 - 解决设计缺陷 Solve Problems
 - Pull vs Push, Normalize vs De-normalize
 - 更多功能设计 More Features
 - Edit, Delete, Media, Ads
 - 一些特殊用例 Special Cases
 - Lady Gaga, Inactive Users
- 第二步 Step 2: **Maintenance**
 - 鲁棒性 **Robust**
 - 如果有一台服务器/数据库挂了怎么办
 - 扩展性 **Scalability**
 - 如果有流量暴增, 如何扩展

Scale 扩展 – 解决Pull的缺陷

- 最慢的部分发生在用户读请求时(需要耗费用户等待时间)
 - 在 DB 访问之前加入Cache
 - Cache 每个用户的 Timeline
 - N次DB请求 → N次Cache请求 (N是你关注的好友个数)
 - Trade off: Cache所有的? Cache最近的1000条?
 - Cache 每个用户的 News Feed
 - 没有Cache News Feed的用户: 归并N个用户最近的100条Tweets, 然后取出结果的前100条
 - 有Cache News Feed的用户: 归并N个用户的在某个时间戳之后的所有 Tweets
- 课后作业: 对比MySQL 和 Memcached 的 QPS
 - Memcached QPS / MySQL QPS ~ 100

Scale 扩展 – 解决Push的缺陷

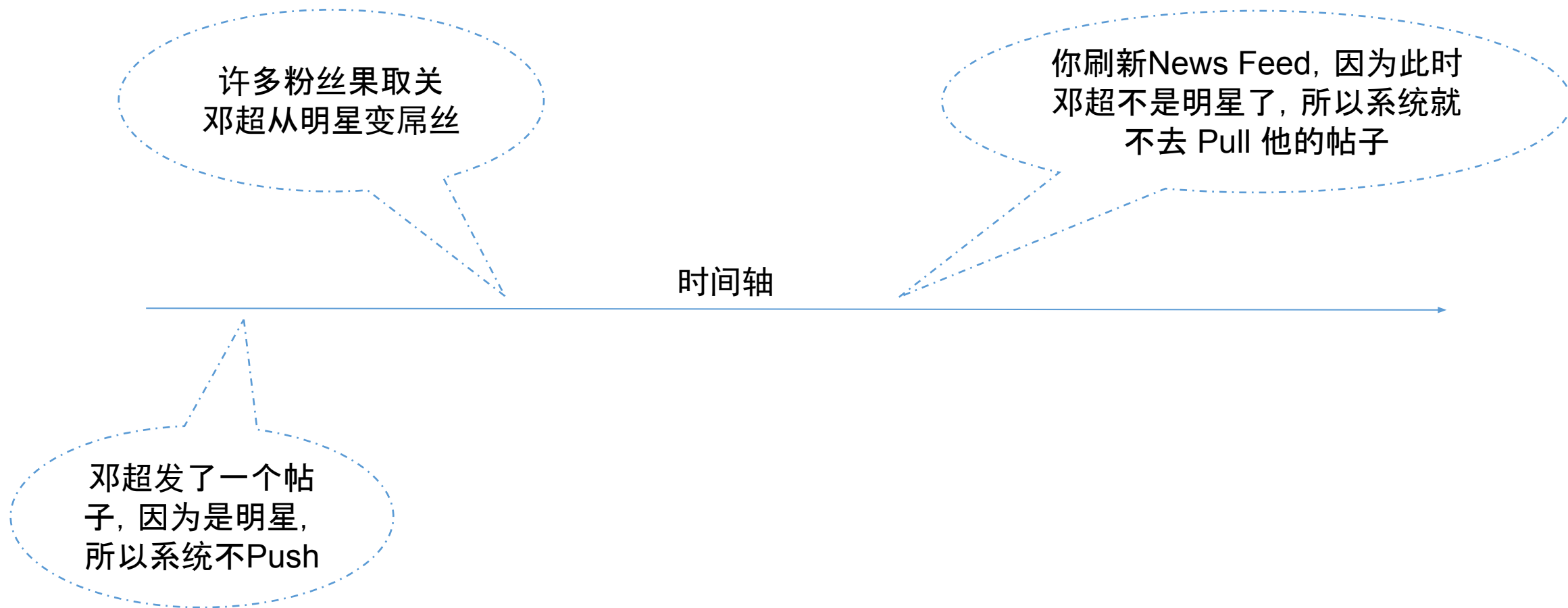
- 浪费更多的存储空间 Disk
 - 与Pull模型将News Feed存在内存(Memory)中相比
 - Push模型将News Feed存在硬盘(Disk)里完全不是个事儿
 - Disk is cheap
- 不活跃用户 Inactive Users
 - 粉丝排序 Rank followers by weight (for example, last login time)
- 粉丝数目 followers >> 关注数目 following
 - Lady Gaga问题
 - 无解？完全切换回Pull？
 - Trade off: Pull + Push vs Pull

Scale 扩展 – Lady Gaga

- 粉丝 Followers **65.5M**
 - Justin Bieber **80M** on Instagram
 - 谢娜 **90M** on Weibo
 - 以上数据来自2017年3月
- Push 的挑战
 - Fanout 的过程可能需要几个小时！
- 面试时错误的回答方案
 - 既然 Push 不行，那我就切换到 Pull 吧！
 - 说起来好容易啊！
- 正确的思路
 - 尝试在现有的模型下做最小的改动来优化
 - 比如多加几台用于做 Push 任务的机器, Problem Solved!
 - 对长期的增长进行估计，并评估是否值得转换整个模型



- Push 结合 Pull 的优化方案
 - 普通的用户仍然 Push
 - 将 Lady Gaga 这类的用户, 标记为明星用户
 - 对于明星用户, 不 Push 到用户的 News Feed 中
 - 当用户需要的时候, 来明星用户的 Timeline 里取, 并合并到 News Feed 里
- 摇摆问题
 - 明星定义
 - followers > 1m
 - 邓超掉粉
 - 邓超某天不停的发帖刷屏, 于是大家果取关, 一天掉了几十万粉
 - 解决方法
 - 明星用户发 Tweet 之后, 依然继续 Push 他们的 Tweet 到所有用户的 News Feed 里
 - 原来的代码完全不用改了
 - 将关注对象中的明星用户的 Timeline 与自己的 News Feed 进行合并后展示
 - 但并不存储进自己的 News Feed 列表, 因为 Push 会来负责这个事情。



- 为什么既然大家都用Pull, 我们仍然要学习Push?
 - 系统设计不是选择一个最好的方案
 - 而是选择一个最合适的方案
 - 如果你没有很大的流量, Push是最经济最省力的做法
- 系统设计面试也并不是期望你答出最优的解决方法, 而是从你的分析当中判断你对系统的理解和知识储备。
- 什么时候用 Push?
 - 资源少
 - 想偷懒, 少写代码
 - 实时性要求不高
 - 用户发帖比较少
 - 双向好友关系, 没有明星问题(比如朋友圈)
- 什么时候用 Pull?
 - 资源充足
 - 实时性要求高
 - 用户发帖很多
 - 单向好友关系, 有明星问题

Interviewer: 如何实现 follow & unfollow?

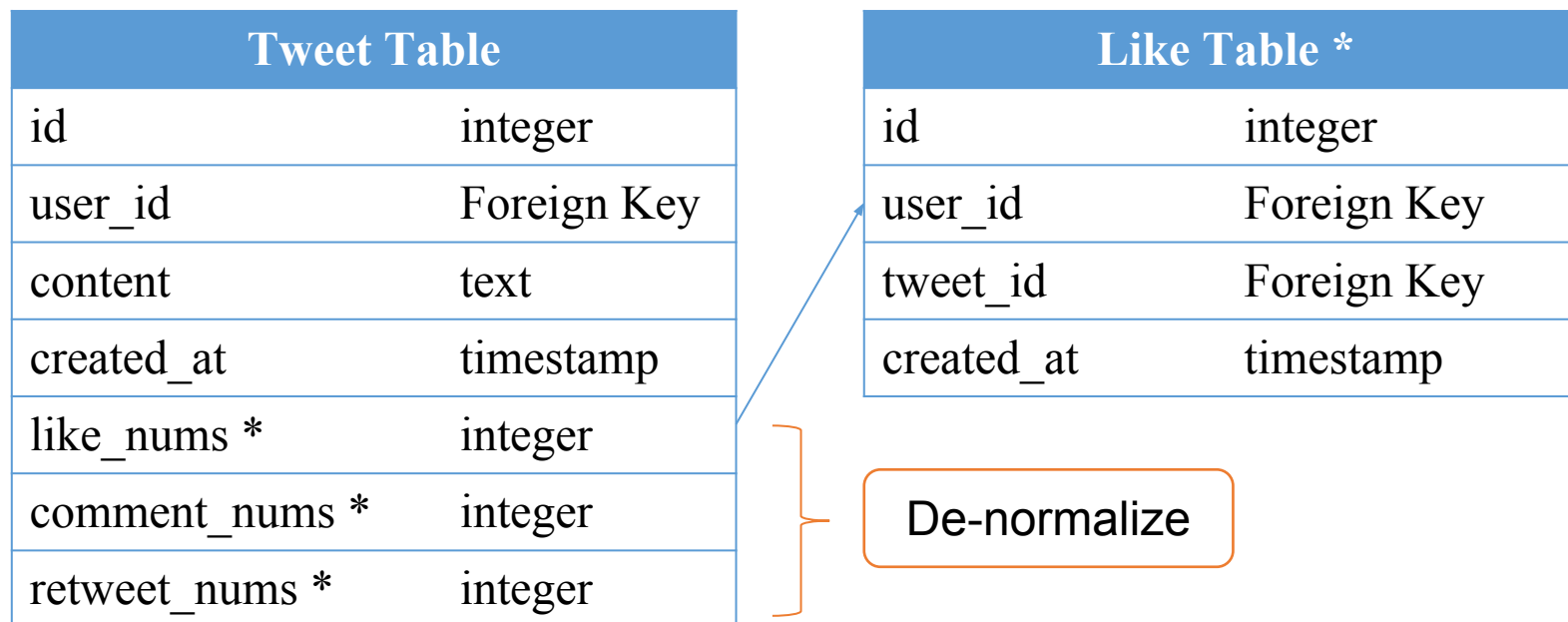


Scale 扩展 – 果取关问题

- 如何实现 follow 与 unfollow?
 - Follow 一个用户之后, 异步地将他的 Timeline 合并到你的 News Feed 中
 - Merge timeline into news feed asynchronously.
 - Unfollow 一个用户之后, 异步地将他发的 Tweets 从你的 News Feed 中移除
 - Pick out tweets from news feed asynchronously.
- 为什么需要异步 Async?
 - 因为这个过程一点都不快呀
- 异步的好处?
 - 用户迅速得到反馈, 似乎马上就 follow / unfollow 成功了
- 异步的坏处?
 - Unfollow 之后刷新 News Feed, 发现好像他的信息还在
 - 不过最终还是会被删掉的

* Interviewer: 如何存储 likes?





* Interview: Lady Gaga 发个自拍会怎样？

无数人点赞，评论，转发，跪舔

猜一猜可能会发生什么？

惊群效应 Thundering Herd

Read more: <http://bit.ly/1ZsD5tW>

<http://www.jiuzhang.com/article?tags=redis&tags=system-design>

- 数据库承受不住压力
 - 对于同一条数据短时间出现大量的请求
 - 因为是同一条数据, 所以什么load balancer, sharding, consistent hashing都不管用
- 解决方案: 那我加上 Cache 呗?
 - 真这么容易?
- Follow Up:
 - 点赞, 转发, 评论, 都会修改这条 Tweet 的基本信息, 如何更新?
 - Keywords: write through, write back, look aside
- Follow Up:
 - Cache 失效如何破?
 - 因为内存不够或者 Cache 决策错误, 热点信息被踢出了 Cache, 会发生什么?
 - 一大波僵尸袭来——DB会瞬间收到一大波对该数据请求, 然后就可以安心的挂了
 - 解决方法:
 - Facebook Lease Get (from Facebook Paper)
 - Read more: <http://bit.ly/1jDzKZK>

Scale 扩展 – 通用问题 Common Questions

- 数据库服务器挂了怎么办？How to maintenance?
- 用户逐渐怎么怎么办？How to scale?
 - 服务器顶不住压力怎么办？
 - 数据库顶不住压力怎么办？
- 以上两个问题，将在第二节课 Database 的专题中涉及！

- 需求分析 Scenario
- 服务图谱 Service Graph
- 表单设计 Schema Design
- 推与拉 Push vs Pull
 - 明星问题 Lady Gaga
 - 僵尸粉问题 Inactive Users
- 果取关问题 follow & unfollow
- 标准化与非标准化 Normalize vs De-normalize (加分项)
- 热点问题 Hot Spot / Thundering Herd (加分项)

Twitter 的相关问题

- 同类型的类似问题
 - Design Facebook
 - Design Instagram
 - Design Friend Circle (朋友圈)
 - Design Google Reader (RSS Reader)
- 系统中的小部件设计
 - 如何查询共同好友? Mutual Friends
 - <http://www.jiuzhang.com/qa/954/>

系统设计面试总结

Conclusion

4S

Scenario, Service, Storage, Scale



Ask before design

问清楚再动手设计

不要一上来就冲着个巨牛的方案去设计

切忌不要做关键词大师

No more no less

不要总想着设计最牛的系统
要设计够用的系统

Work solution first

先设计一个基本能工作的系统，然后再逐步优化

Done is better than perfect! —— Mark Zuckerberg



Analysis is important than solution

系统设计没有标准答案

记住答案是没用的

通过分析过程展示知识储备

权衡各种设计方式的利弊

系统设计 v5.1 有什么变化？

- v1.0:4节课
- v2.0:7节课, 更贴近真实面试
- v3.0:7节课, 多位老师讲述自己擅长的领域, 增加 LintCode 阶梯训练 (20题)
- v4.0:8节课, 根据往届同学的反馈优化课程, 增加新的 LintCode 练习题 (共30题)
- **v5.0~5.1:9节课, 根据往届同学的反馈优化课程, 增加新的 LintCode 练习题 (共40题)**
-
- LintCode 阶梯训练:
 - <http://www.lintcode.com/ladder/8/>
 - 唯一可以刷系统设计面试题的题库！
 - 40多道独家系统设计练习题帮助你学透系统设计的知识点
 - 掌握代码实现才能“忽悠”得更有底气！

后续课程安排

九章邀你华山论剑

<http://www.jiuzhang.com/course/2/>



课程大纲及上课时间



章节	内容	北京时间	美东时间	美西时间
1	走进系统设计与新鲜事系统 Introduction to System Design & News Feed System 【免费试听】	2017/08/13 04:30:00	2017/08/12 16:30:00	2017/08/12 13:30:00
2	从用户系统中理解数据库与缓存 Design User System - Database & Memcache	2017/08/20 04:30:00	2017/08/19 16:30:00	2017/08/19 13:30:00
3	一致性哈希算法 & 设计短网址系统 Consistent Hashing & Design Tiny Url	2017/08/21 04:30:00	2017/08/20 16:30:00	2017/08/20 13:30:00
4	以GFS为例探索分布式文件系统 Distributed File System	2017/08/27 04:30:00	2017/08/26 16:30:00	2017/08/26 13:30:00
5	爬虫系统与搜索建议系统 Web Crawler & Google Suggestion	2017/08/28 04:30:00	2017/08/27 16:30:00	2017/08/27 13:30:00
6	分布式计算系统 - Map Reduce 的原理与应用	2017/09/10 04:30:00	2017/09/09 16:30:00	2017/09/09 13:30:00
7	基于地理位置信息的系统设计 Location Based Service	2017/09/11 04:30:00	2017/09/10 16:30:00	2017/09/10 13:30:00
8	以Big Table为例探索分布式数据库 Distributed Database	2017/09/17 04:30:00	2017/09/16 16:30:00	2017/09/16 13:30:00
9	聊天系统与访问限制系统 Message System & Rate Limiter	2017/09/18 04:30:00	2017/09/17 16:30:00	2017/09/17 13:30:00

学员福利

九章QA板块访问权限(助教老师答疑, 最新面经, 内推资源)

学员专属QQ群(助教老师答疑, 同学一起打鸡血)

LintCode 系统班阶梯训练 1 年使用权限

QA

谢谢大家



扫描二维码关注微信小程序/公众号
获取第一手求职资料

- Cache是什么？
 - 你可以认为相当于算法中的HashMap
 - 是Key-value的结构
- Cache存在哪儿？
 - Cache存在内存里
- 常用的Cache工具/服务器有什么？
 - Memcached
 - Redis
- 为什么需要用Cache？
 - Cache因为是内存里，所以存取效率比DB要高
- 为什么不全放Cache里？
 - 内存中的数据断电就会丢失
 - Cache 比硬盘贵

- News Feed 和 Timeline 的定义和区别？
 - News Feed: 新鲜事, 我朋友+我发的所有帖子按照某种顺序排列的整合(比如按照时间排序)
 - 用户打开Twitter之后首先看到的界面就是News Feed界面, 这些 tweets 来自你关注的用户
 - Timeline: 某个用户发的所有帖子
 - 用户点开某个人的页面之后, 看到这个人发的所有帖子
 - 在有的系统中, 这两个概念的定义会完全反过来, 这里我们统一按照上面的定义。
- 什么是消息队列
 - 简单的说就是一个先进先出的任务队列列表
 - 做任务的worker进程共享同一个列表
 - Workers从列表中获得任务去做, 做完之后反馈给队列服务器
 - 队列服务器是做异步任务必须有的组成部分
- 哪些工具可以做消息队列
 - 最常用的有 RabbitMQ, ZeroMQ, Redis, Kafka

NewsFeed Table 中 Pull Model 和 Push Model 的区别？

<http://www.jiuzhang.com/qa/2074/>

<http://www.jiuzhang.com/qa/2031/>

<http://www.jiuzhang.com/qa/1741/>

NewsFeed 如何实现 Pagination？

<http://www.jiuzhang.com/qa/1839/>

Twitter Pull 模型里用cache来存timeline时, 关于保持实时性的问题

<http://www.jiuzhang.com/qa/962/>