百度搜索，
规模爆炸下的架构优化实践

吴永巍

2015.10

About me

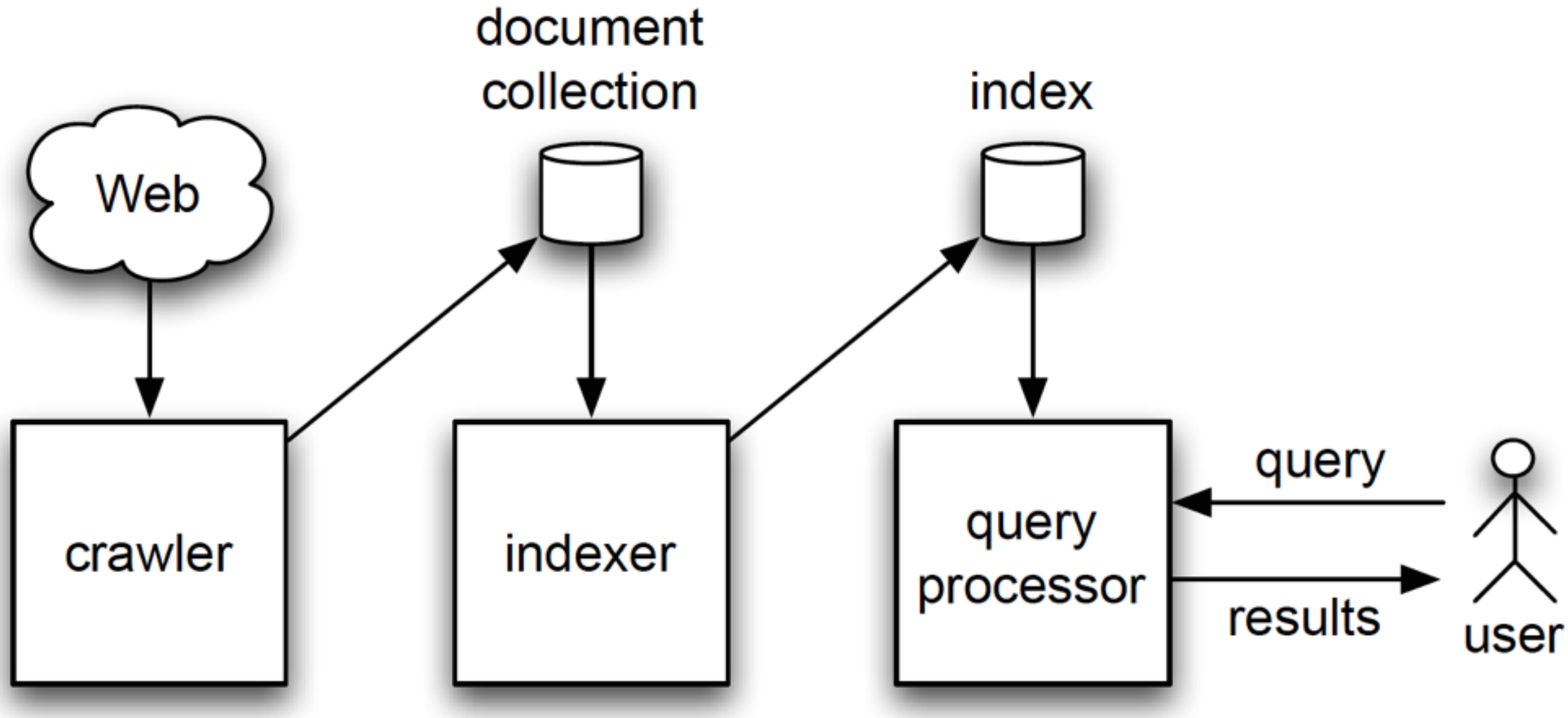
- 06年加入百度，主任架构师
- @网页搜索部，@上海
- 重点关注领域:
  - 搜索架构
  - 分布式存储与计算
  - 大规模系统设计与调优…

Web Search


Baidu 百度
QCon上海 2015
百度一下
百度首页 消息 设置
网页 新闻 贴吧 知道 音乐 图片 视频 地图 文库 更多»
百度为您找到相关结果约11,200个
搜索工具
QCon上海2015 | 全球软件开发大会
Register now for QCon London, a practitioner-driven conference designed for team leads, architects and project management, that tracks innovation in enterpris...
www.qconshanghai.com/ - 百度快照 - 评价
QCon上海
InfoQ是一个实践驱动的社区资讯站点,致力于促进软件开发领域知识与创新的传播。... 在10月15日~17日的QCon上海2015上,他将分享《Haskell中的函数与类型系统》。在大...
www.infoq.com/cn/qcons... V1 - 百度快照 - 89%好评
QCon北京2015 | 全球软件开发大会
Register now for QCon London, a practitioner-driven conference designed for team leads, architects and project management, that tracks innovation in enterpris...
www.qconbeijing.com/ V1 - 百度快照 - 评价
QCon上海2015大会启动筹备,15个专题征集演讲嘉宾
2015年6月18日 - QCon上海2015大会已经启动筹备,QCon组委会针对软件开发领域的热点与趋势,议定了15个最具实践价值的专题,涉及架构、容器、大数据、移动开发等热门领域,...
www.infoq.com/cn/news/... V1 - 百度快照 - 89%好评
QCon上海2015 | 全球软件开发大会
Register now for QCon London, a practitioner-driven conference designed for team leads, architects and project management, that tracks innovation in enterpris...
其他人还搜
展开
百度网盘
超大网络存储空间
冯大辉
任职数据库架构师
infoq
在线新闻/社区网站
csdn
开发者技术社区
51cto
互联网媒体机构
javaone
每年一次的Java盛会
世界知识产权日
尊重知识崇尚科学节日
数据库
组织存储和管理的仓库
相关大会

Web Search, more

Web Search Infrastructure



搜索引擎关注的架构指标

规模爆炸：股票

- 过去10年，+1090%（同期纳斯达克+114%）

规模爆炸：一些数字

几万亿

几千亿

千亿

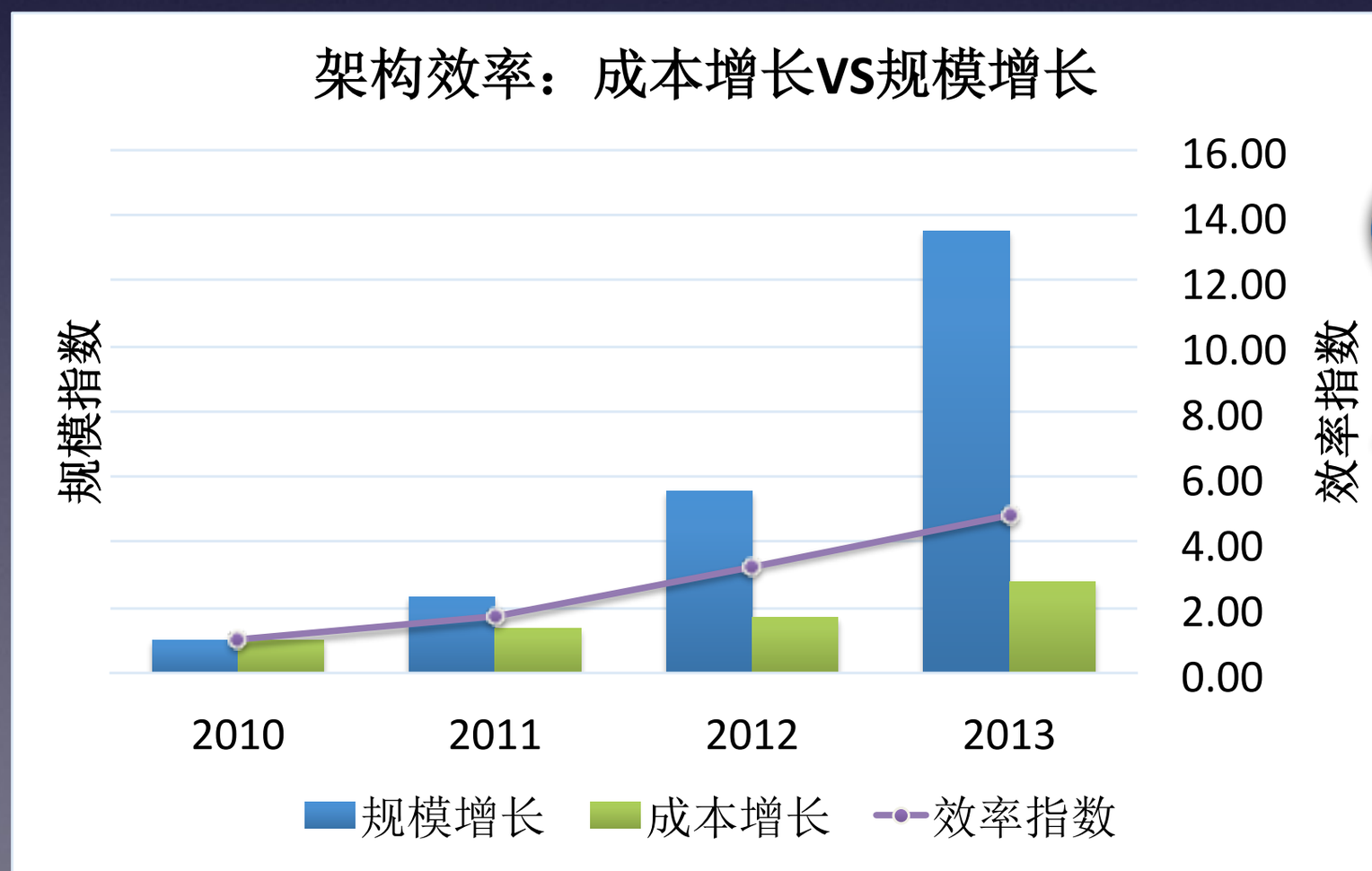
几十亿

比08年增长2个数量级

规模爆炸下的挑战

- 最大的挑战：成本！
- 规模 = 数据量 * 流量 * 算法复杂度

乘法关系!



架构工程师的价值!

规模爆炸下的挑战，More

- 服务质量，更全面的挑战：
- 可用性
- 低延迟
- …

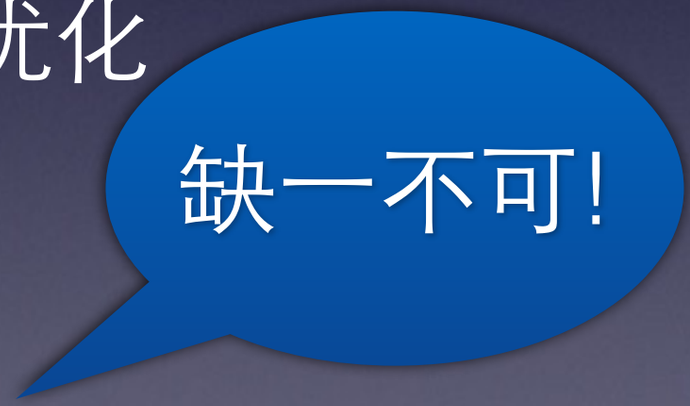
规模爆炸下，成本、扩展性、延迟、可用性的挑战与实践

规模爆炸，成本的挑战

成本，不得不谈的问题

- 百度2014年度财报：硬件成本48亿

- 架构工程师的义务：成本控制与优化
  - 宏观的优化：分布式，集群
  - 微观的优化：单机



痛点

- 用户吐槽“有些资料，百度搜不到”
- 技术上，要扩大数据规模，涵盖长尾数据
- 问题：要求加的机器数太多，会被challenge

挑战与机遇：差异化

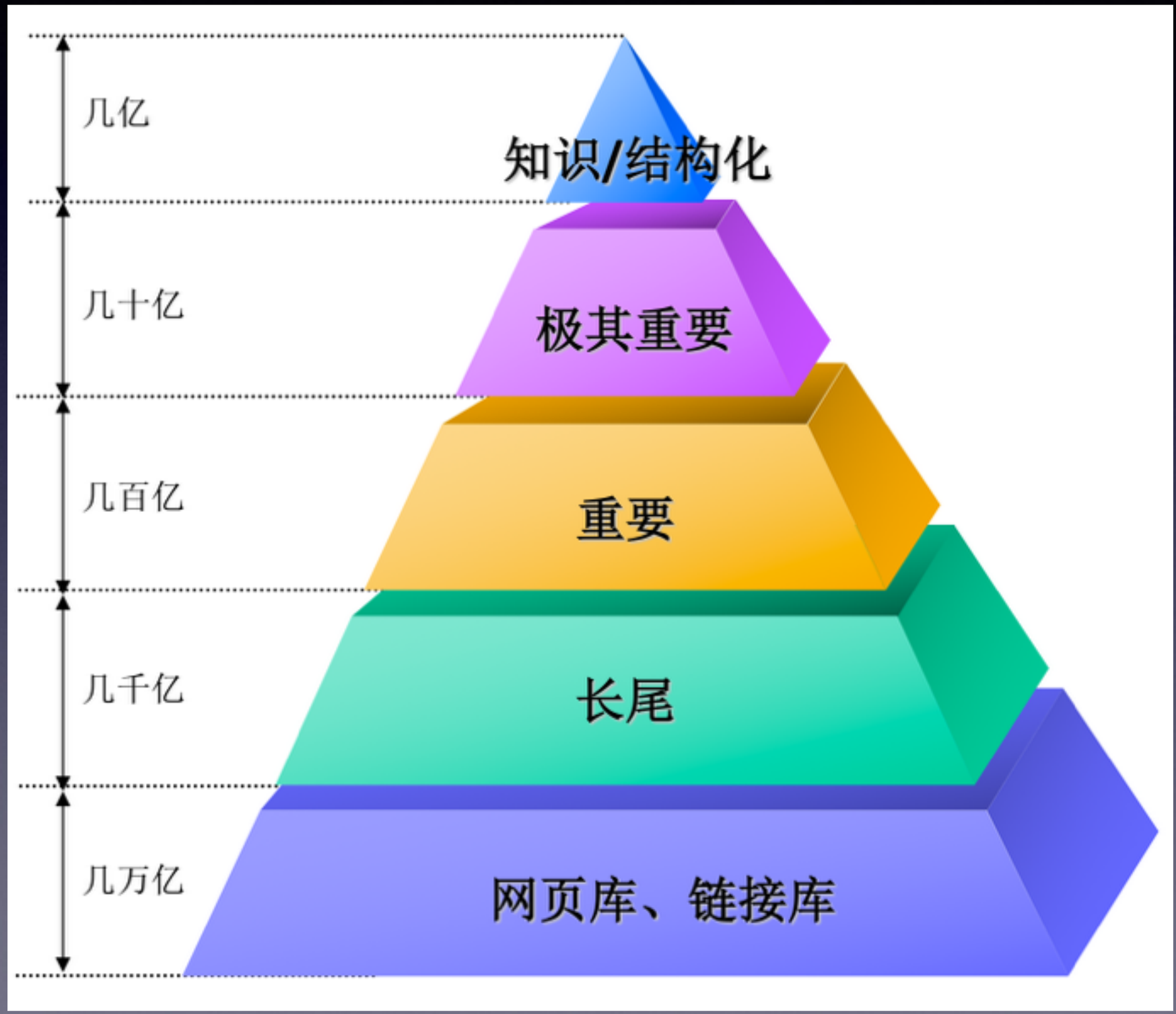
- 数据间的差异
- 流量间的差异
- 策略间的差异

分层架构

- 不要平均主义、大锅饭！

分层架构：数据分层

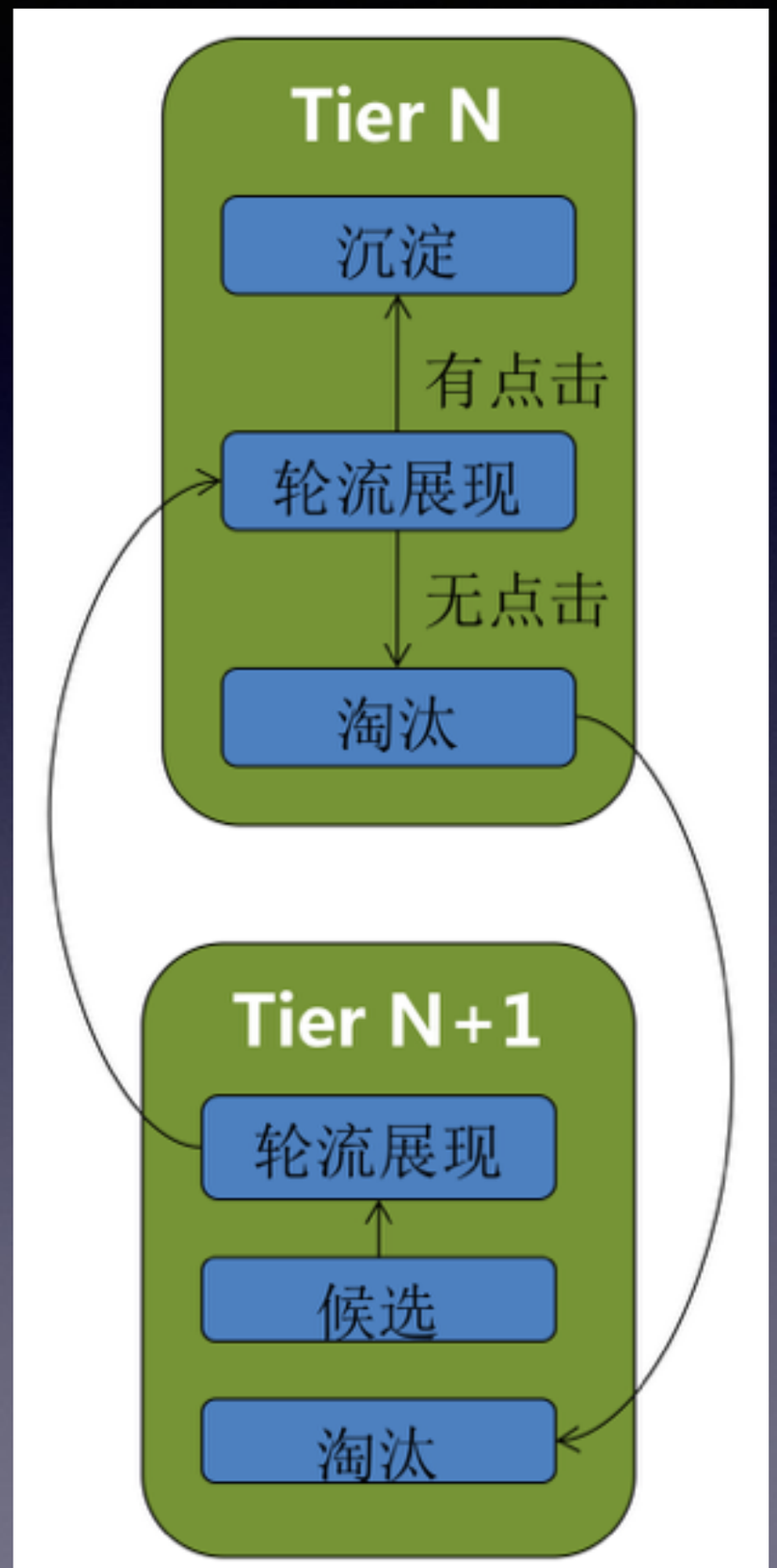
- 金字塔
- 区分对待



数据分层

- How?
  - 页面质量
  - 用户行为
  - 轮流展现&淘汰
- 不同的数据丰富度
- 不同的算法复杂度和查询策略

给机会，
不一棍子打死



流量分层

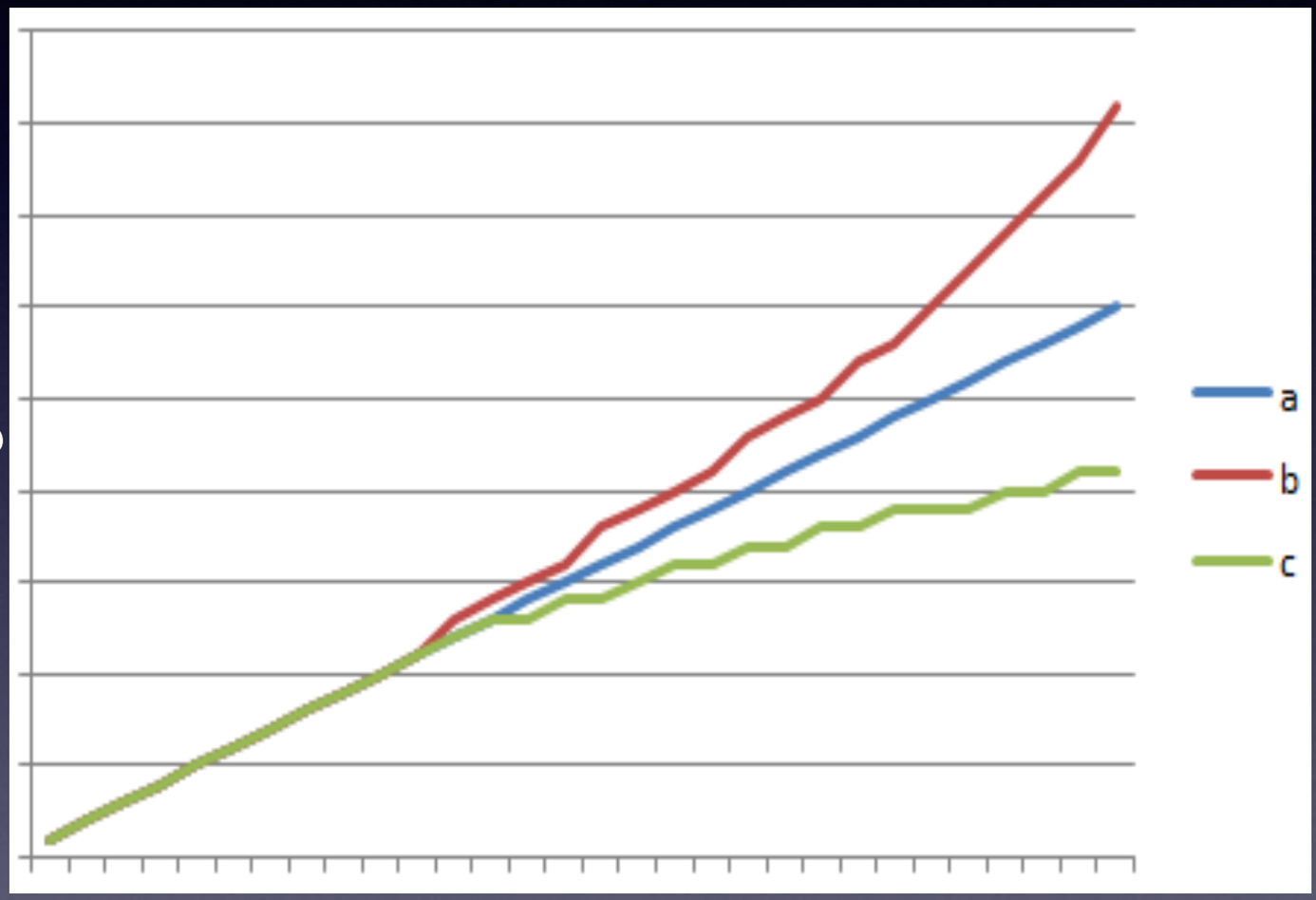
- 正常流量(白色)
- 疑似低质流量(灰色地带)
- 作弊流量(黑色)

- 区分对待，无需一视同仁
- 节省不少成本

分布式系统的成本，扩展性是关键!

扩展性，Scale out

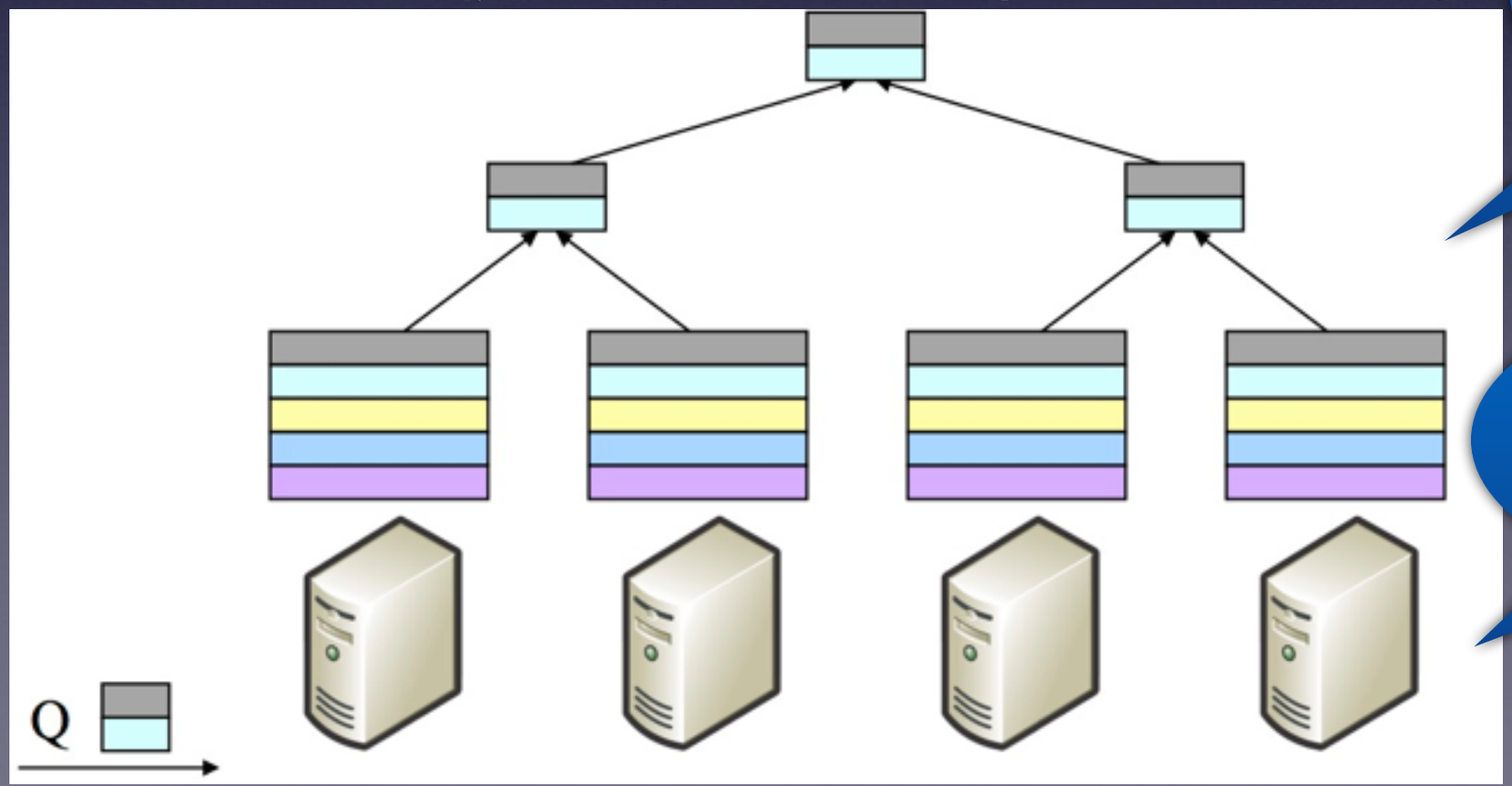
- 普通扩展：线性
- 2B扩展：差于线性
- 文艺扩展：优于线性?



- 痛点：
  - 加机器也没用 or 加太多机器了

大规模检索系统，如何sharding?

- 搜索成本，与shard数量O(N)相关
- shard数量 = 数据总规模 / 单位节点数据量
- 数据总规模越来越大，节点越来越小

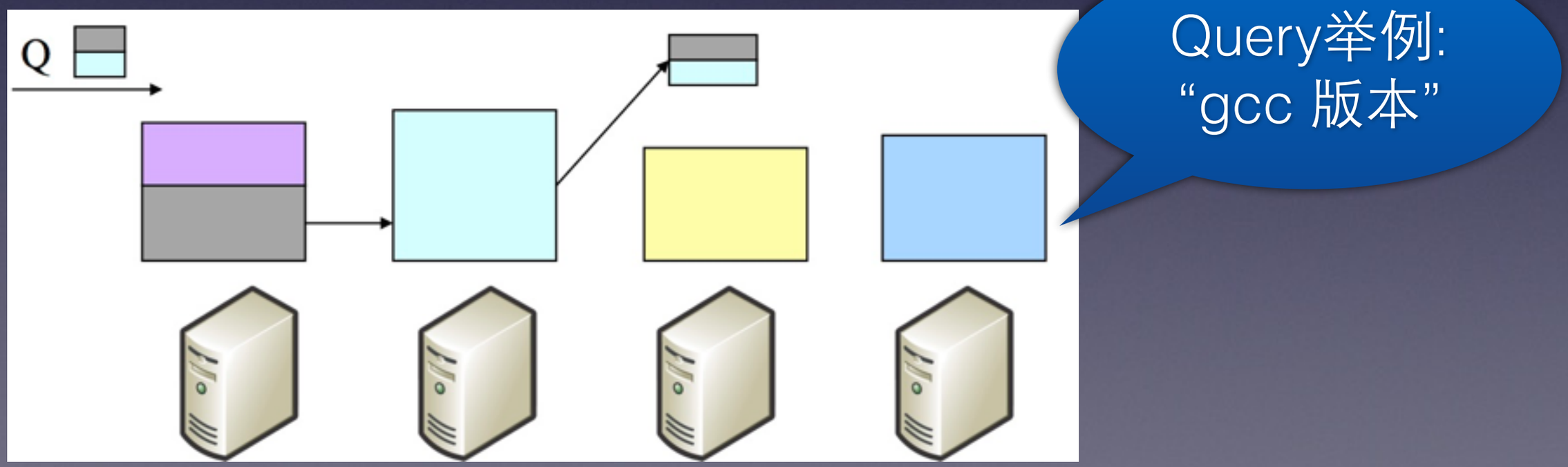


直观做法：
把网页集合分N份
N越来越大

Query举例：
“gcc 版本”

不同的sharding方式

- 按Term / 词 / 短语来做sharding
- 搜索成本，与网页规模O(1)关系，与Query有关

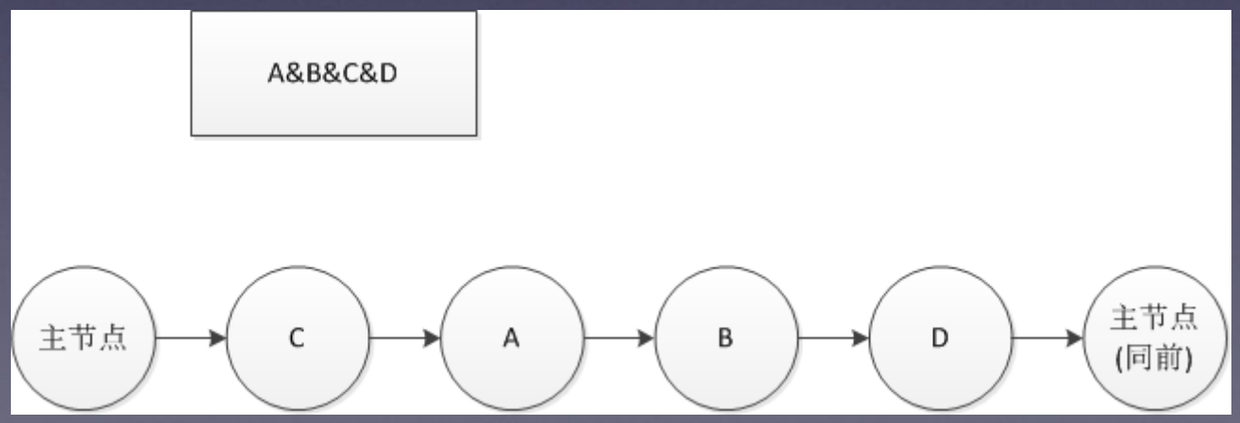


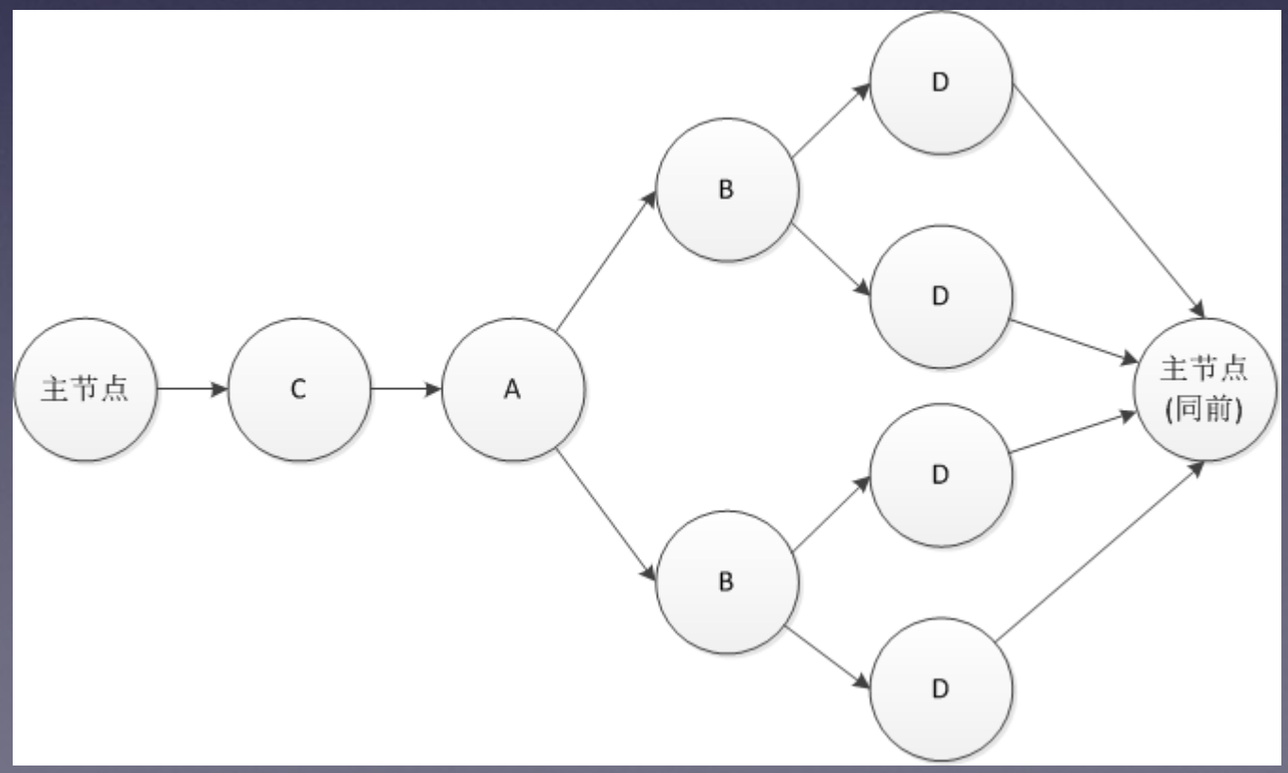
合适的场景，合适的sharding

- 低成本支持千亿级别索引!
  - 88%的查询访问节点数<10%

单位成本:
几十分之一

- 只有6.2%的查询需要访问所有机器



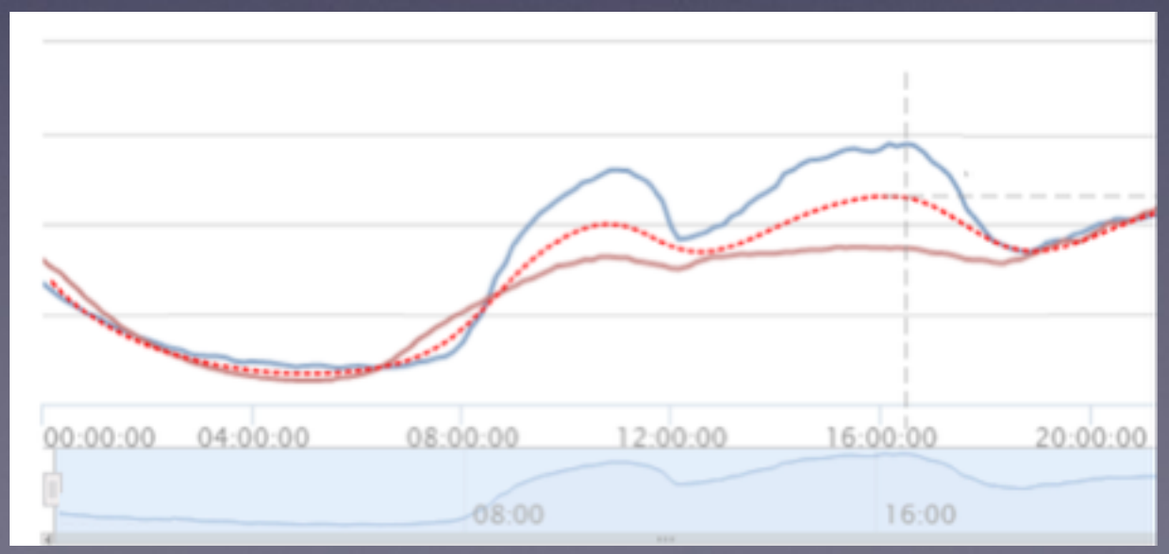


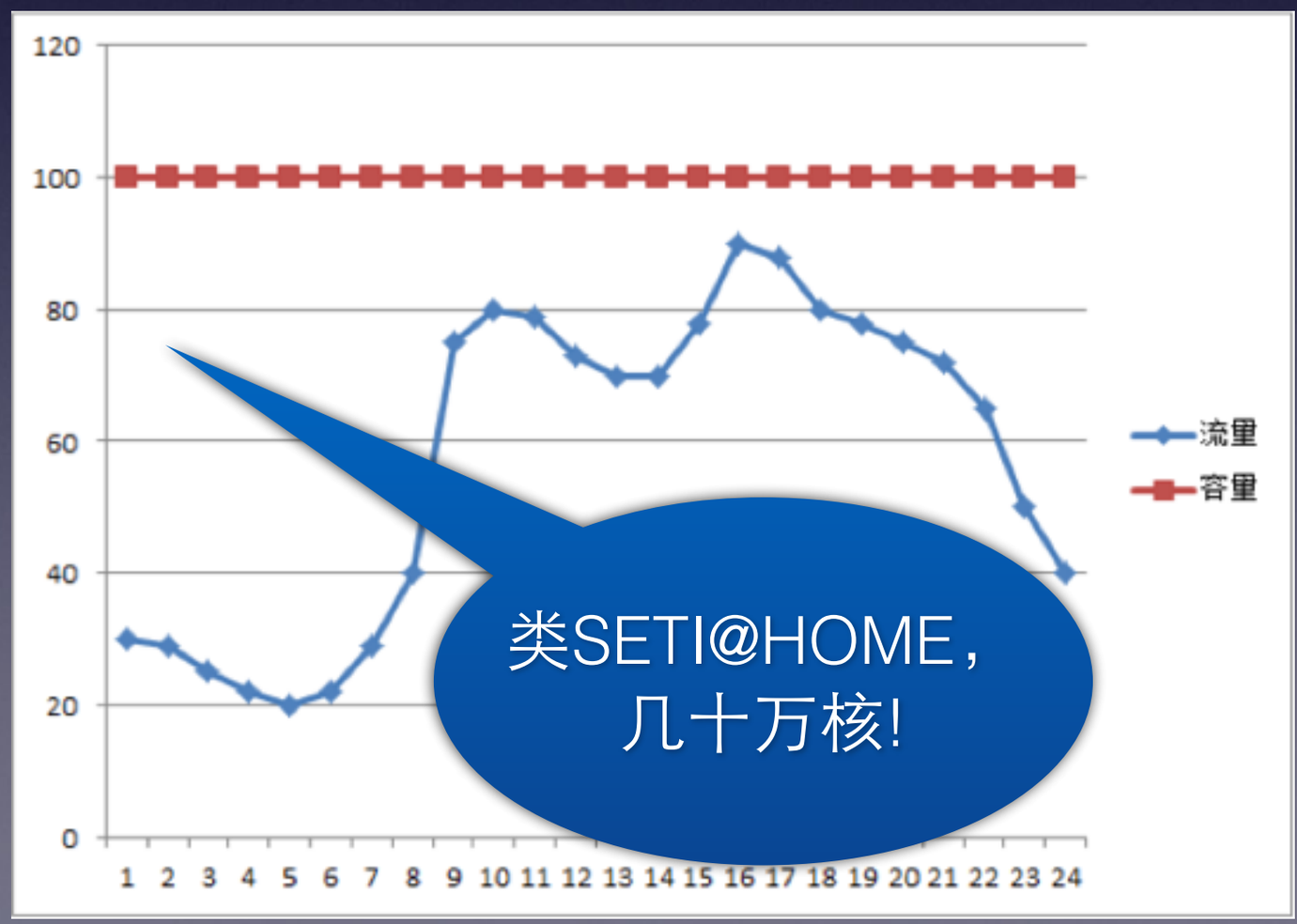
痛点

- 矛盾:
  - 需要扩容加大量机器
  - 已有资源，负载低，利用率并不高
- 被老板challenge && 程序员的自尊

大规模系统，利用率是成本的关键

- 混合部署：错峰 & 互补
- 在线与离线的统一调度（利用闲散资源）：Baidu Volunteer Computing
- 架构前提：容忍波动



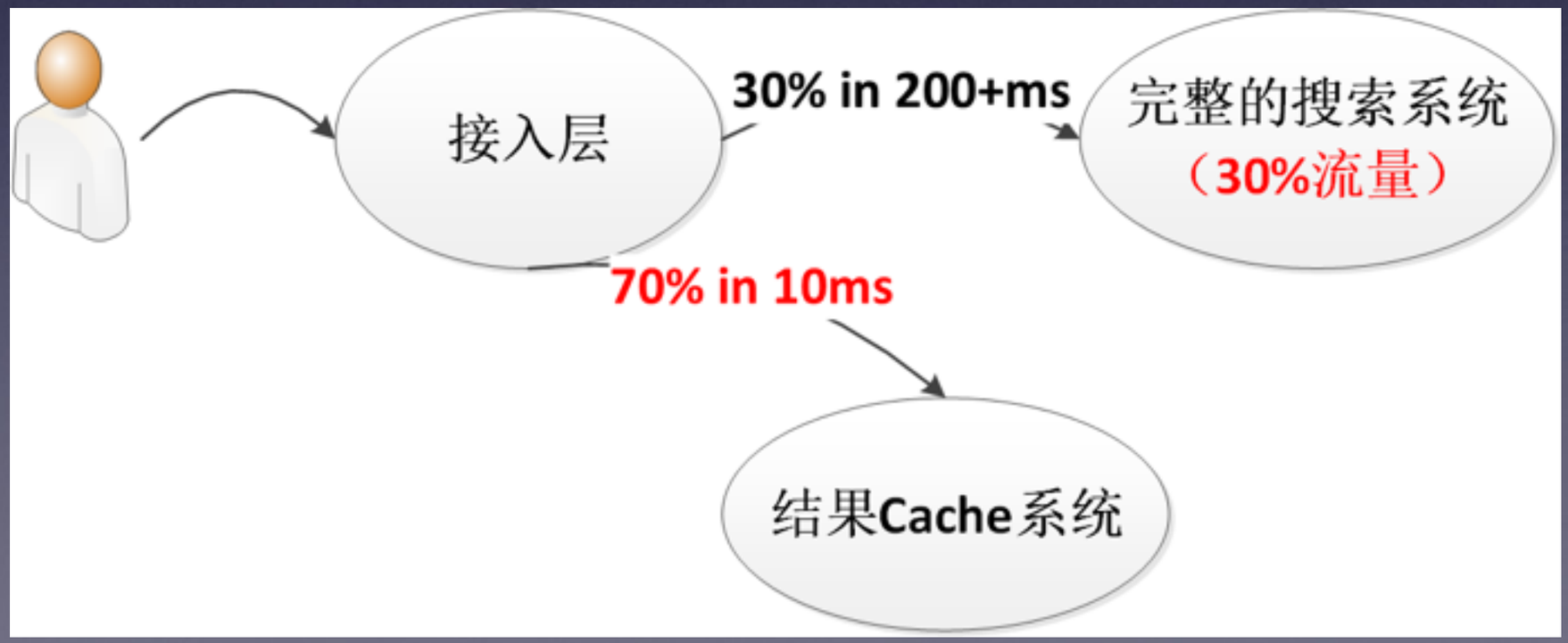


都说Cache是万金油

哪里不舒服抹哪里

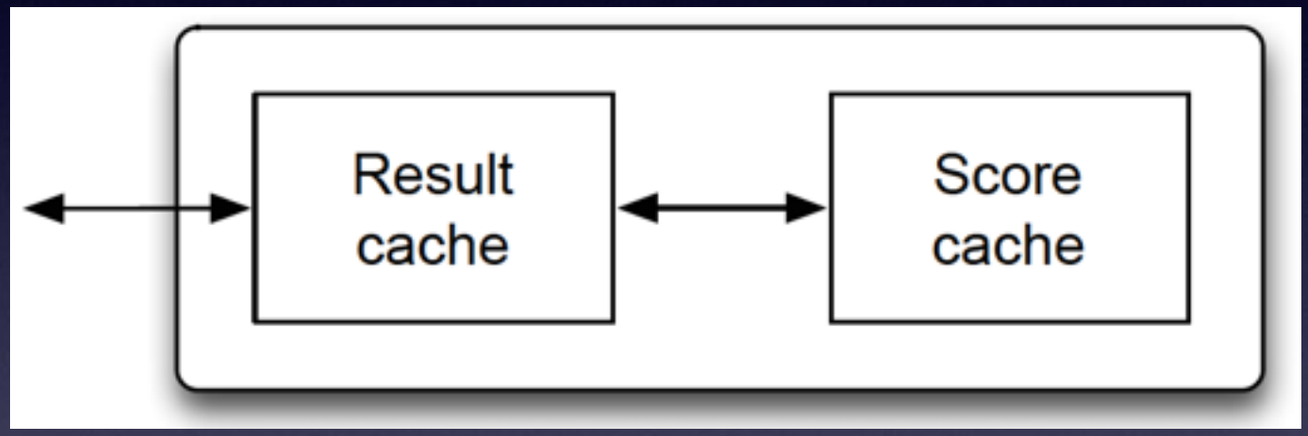
规模爆炸下，Cache的价值

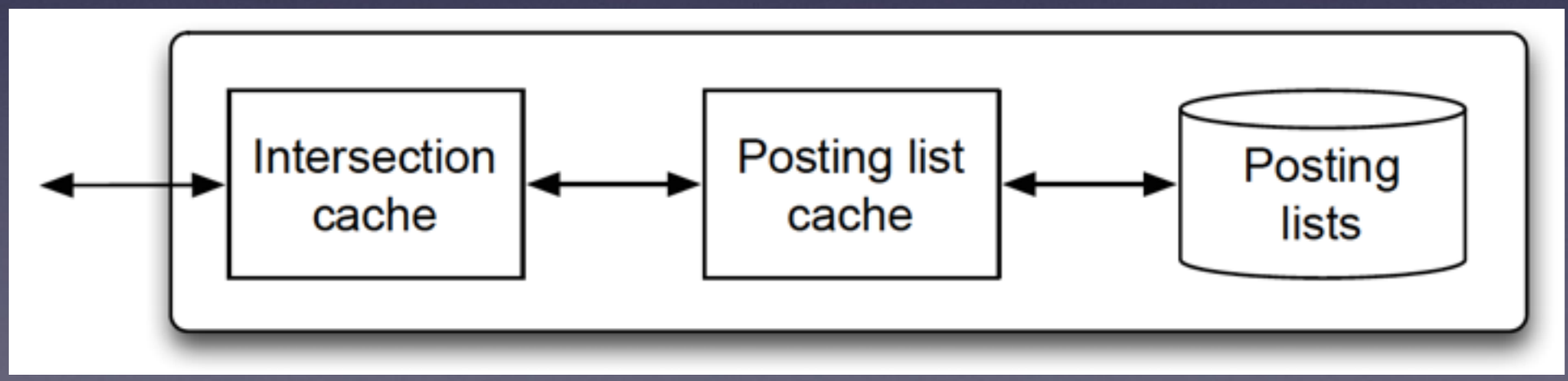
- 成本：大规模系统，完整计算的成本很高
- 延迟：简单取数据 VS 大规模算
- 可用性：多一套备选方案
- 唯一的代价：
  - 时效性



Cache for Search

- 各种类型，各种level，能Cache的都Cache：
- 搜索结果
- 倒排索引
- 属性数据
- 中间片段
- ……





No!
Never淘汰!

说到Cache，就会说到淘汰算法

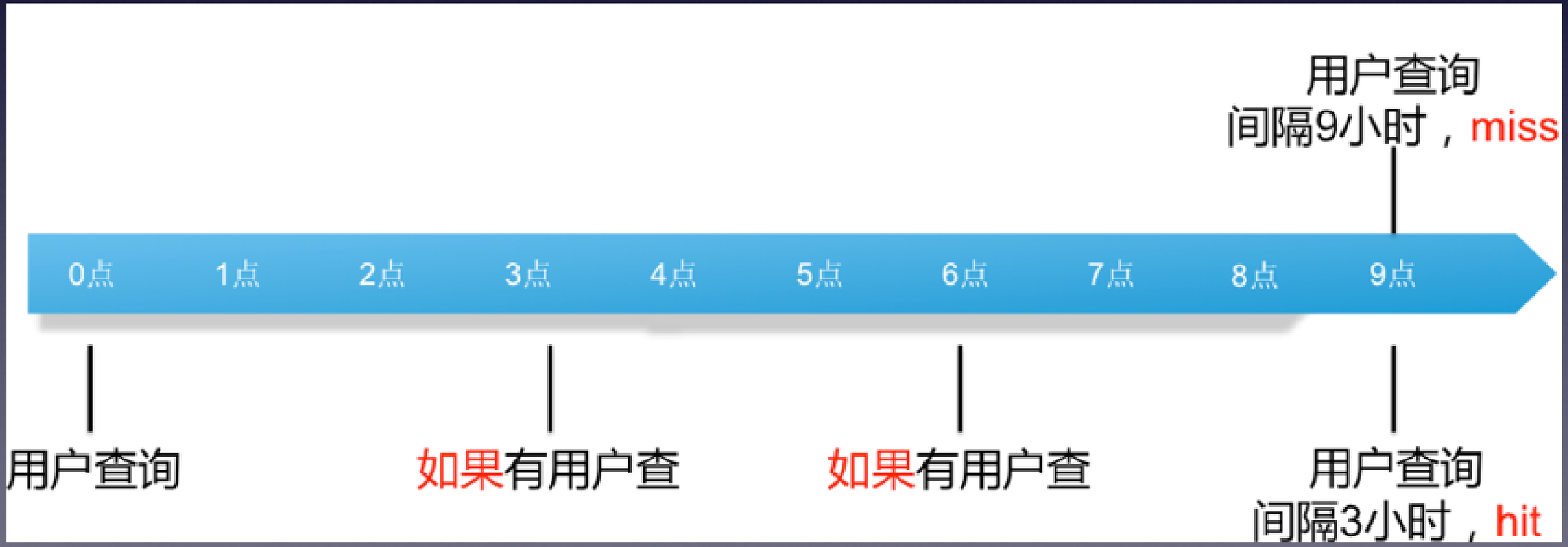
LRU? LFU? FIFO?

Never(因为容量)淘汰的Cache

- 只会因为不满足Freshness要求而淘汰
- 技术方案：
  - 超大容量的Cache集群
  - 用上SSD、Disk
  - 千台规模

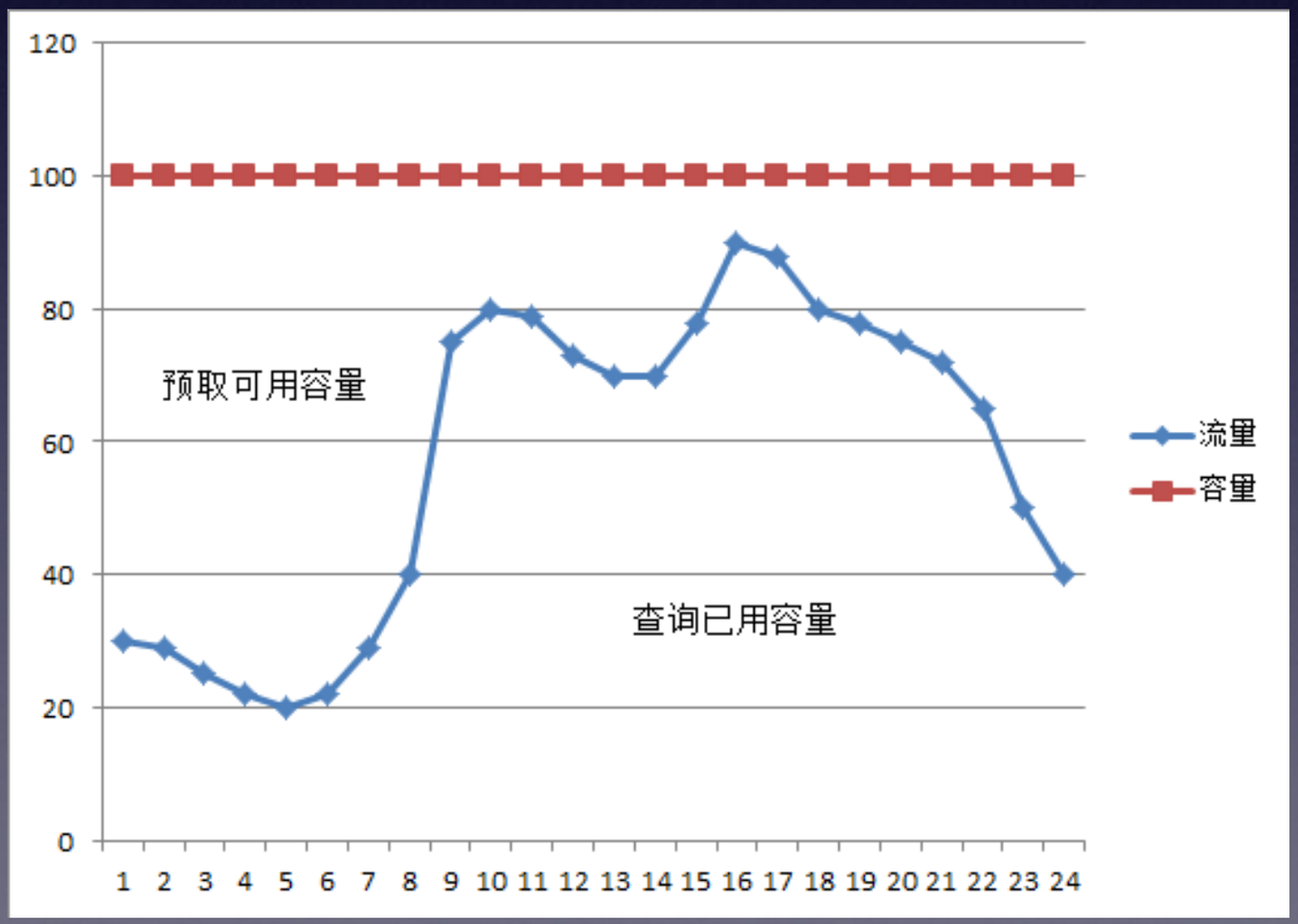
智能预取，always keep cache fresh

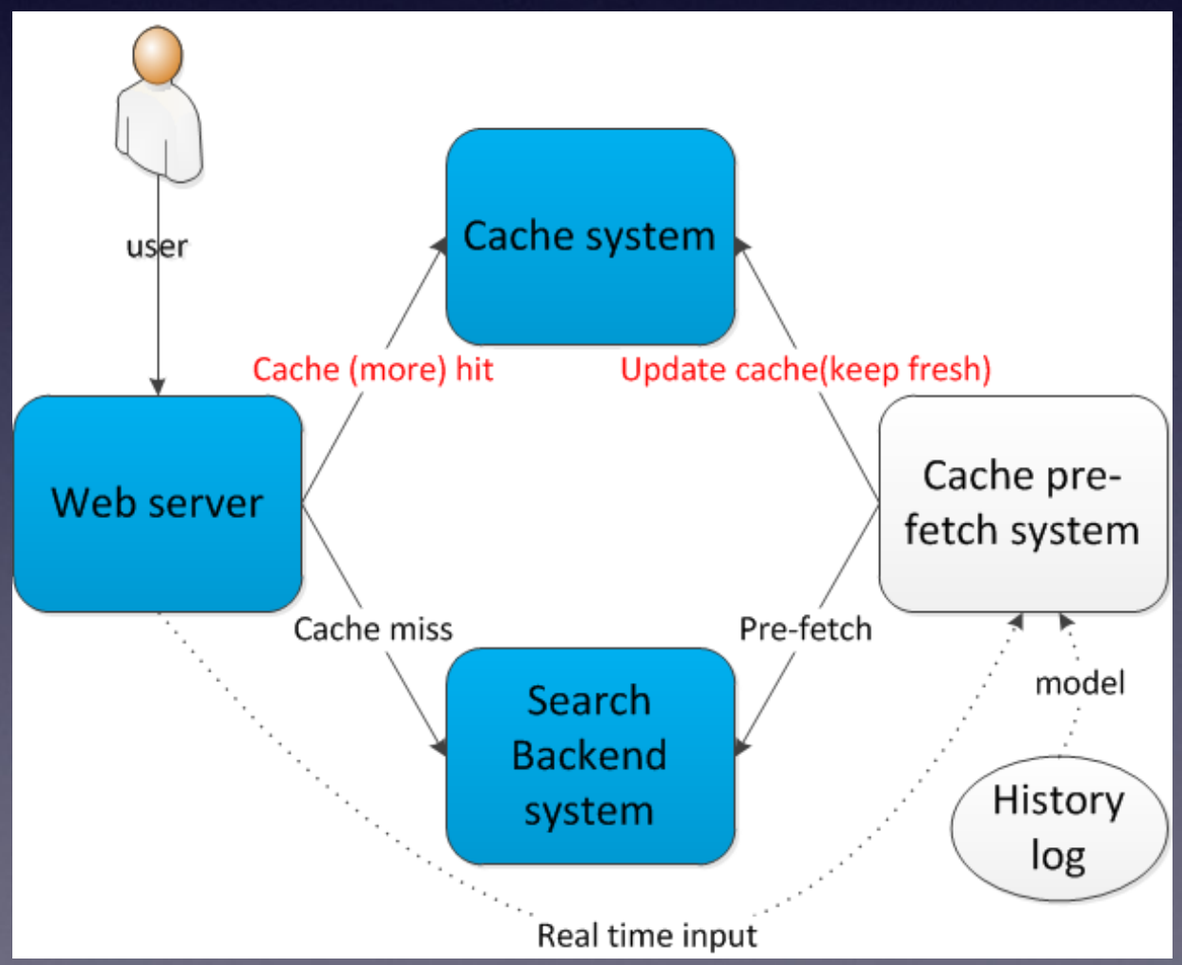
- 优化 “只会因为不满足Freshness要求而淘汰”



智能预取，always keep cache fresh

- 收益：早高峰10+%





更多Cache的优化实践

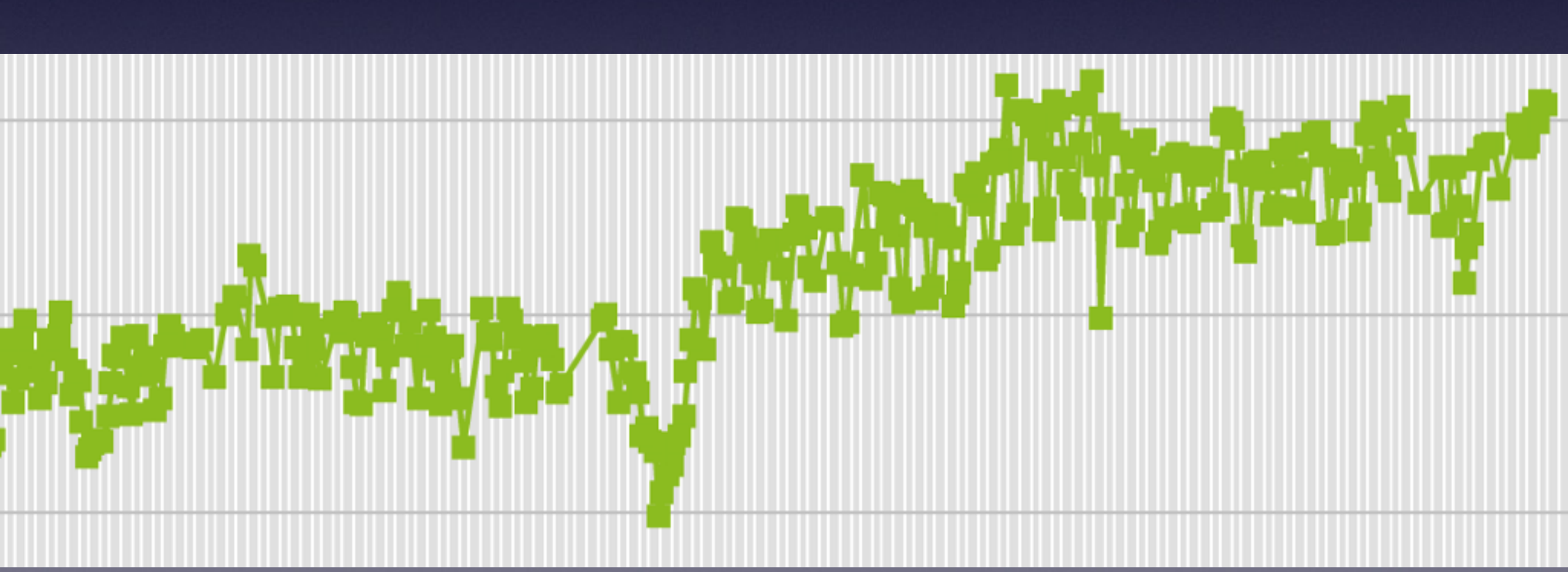
不要谈Cache必Memcached,Redis

- 分布式远程Cache不是唯一
- 各种：系统Cache, 本地Cache, Bloomfilter, Flashcache…
- 动态&静态，结合
- 系统的重要组成部分，避免波动

规模爆炸下
低延迟的挑战

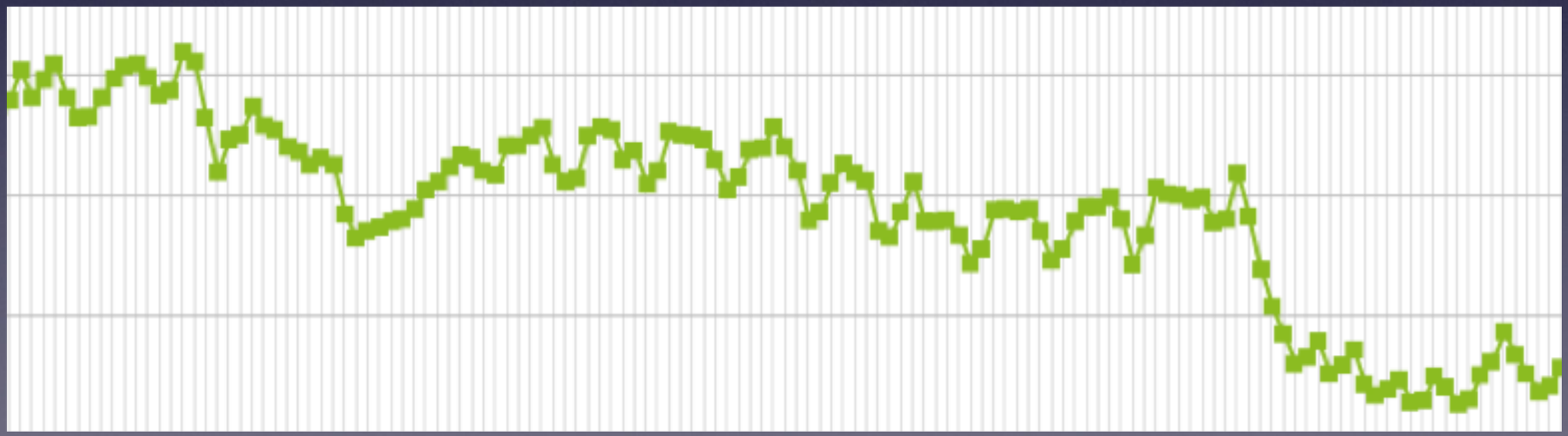
痛点

- 没有好的架构机制保证，延迟很快就被“劣化”了
- 同时，不能阻碍规模爆炸、策略迭代与复杂化



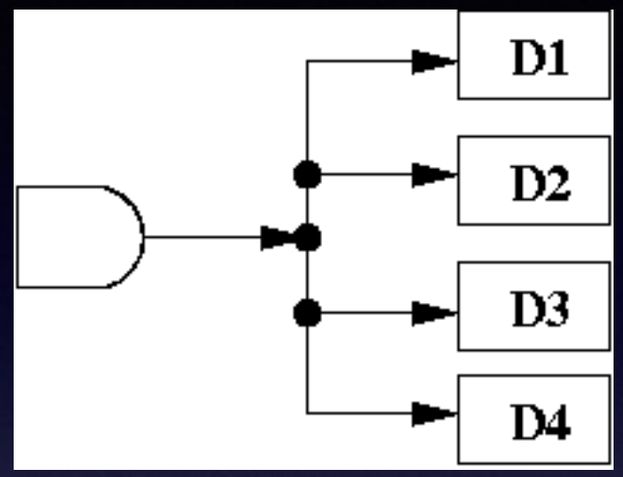
优化 & 不劣化

- 打江山
- 守江山



大扇出，木桶效应

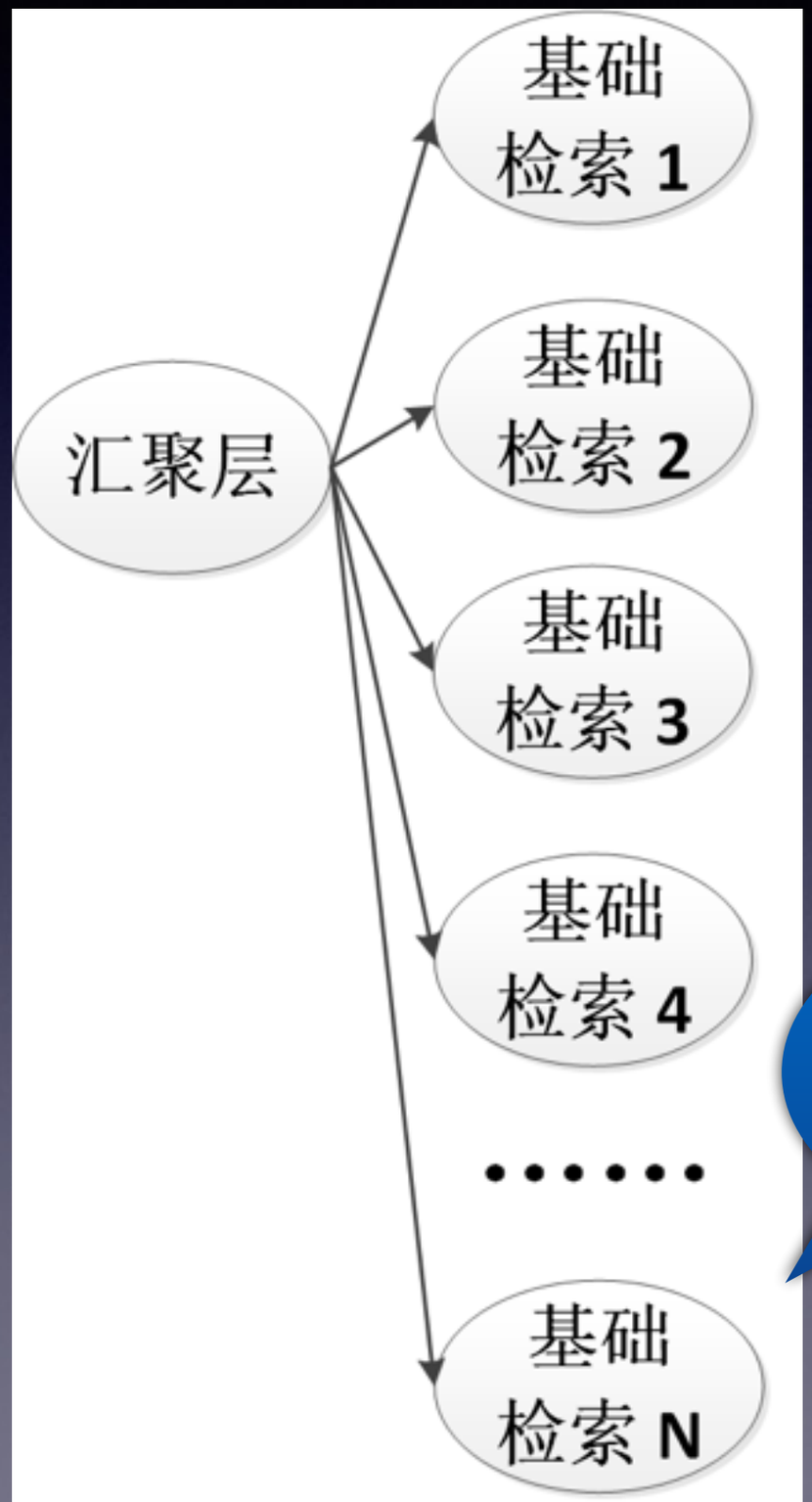
- 大扇出（large fan-out）
- 短板决定整体



- 云环境
  - 容器，虚拟化，多租户
  - 高利用率 VS 隔离的不完备
  - 波动是不可避免的

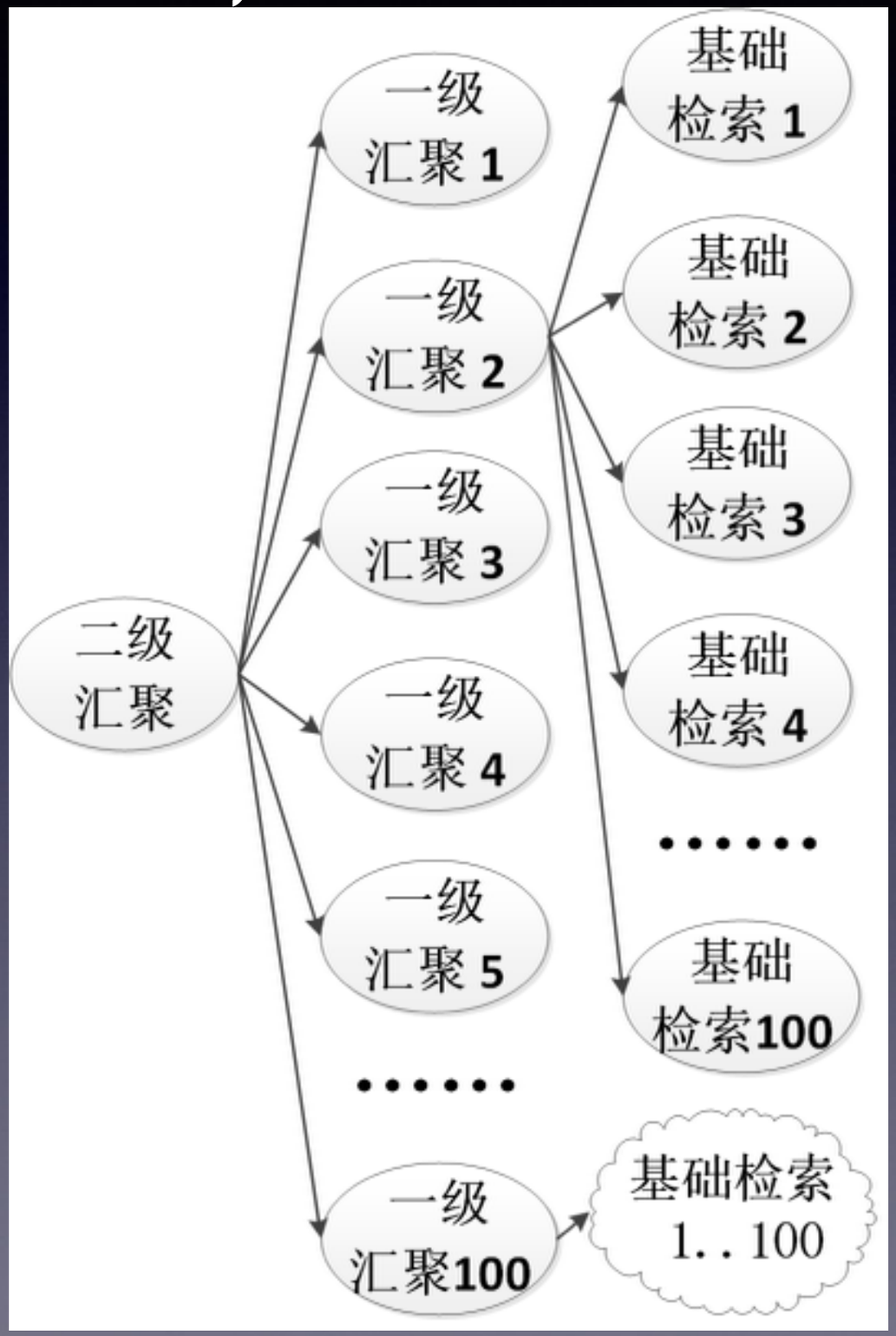
大扇出，波动下的延迟

- 举例，一个服务：
  - 均值10ms
  - 0.1%>1s
- 汇聚100个节点：
  - 1-0.999^100=9.5% > 1s
- 汇聚1000个节点：
  - 1-0.999^1000=63% > 1s



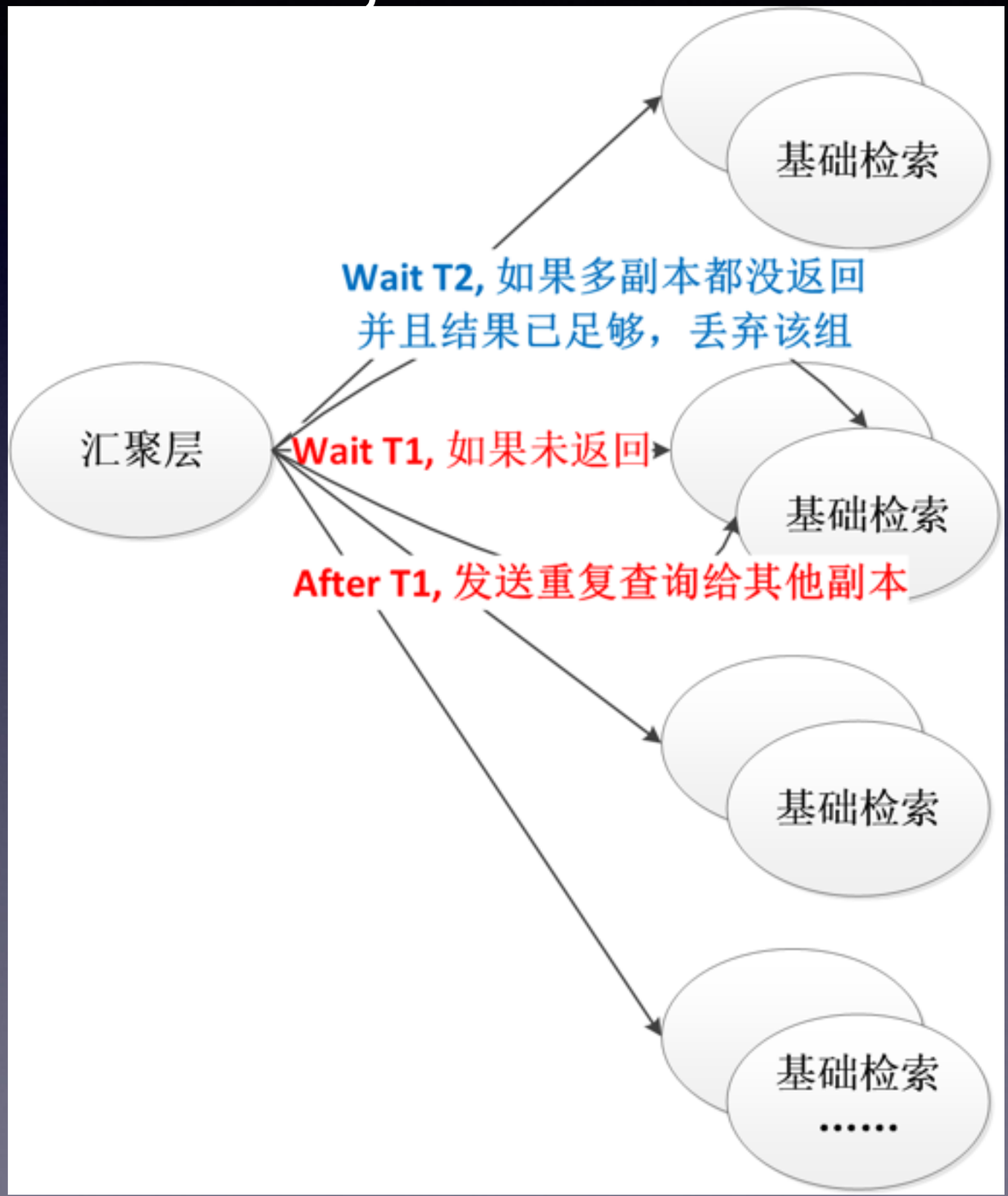
Our solution,1

- 多级汇聚
- 类似:
  - 团队大了，引入二线经理



Our solution, 2

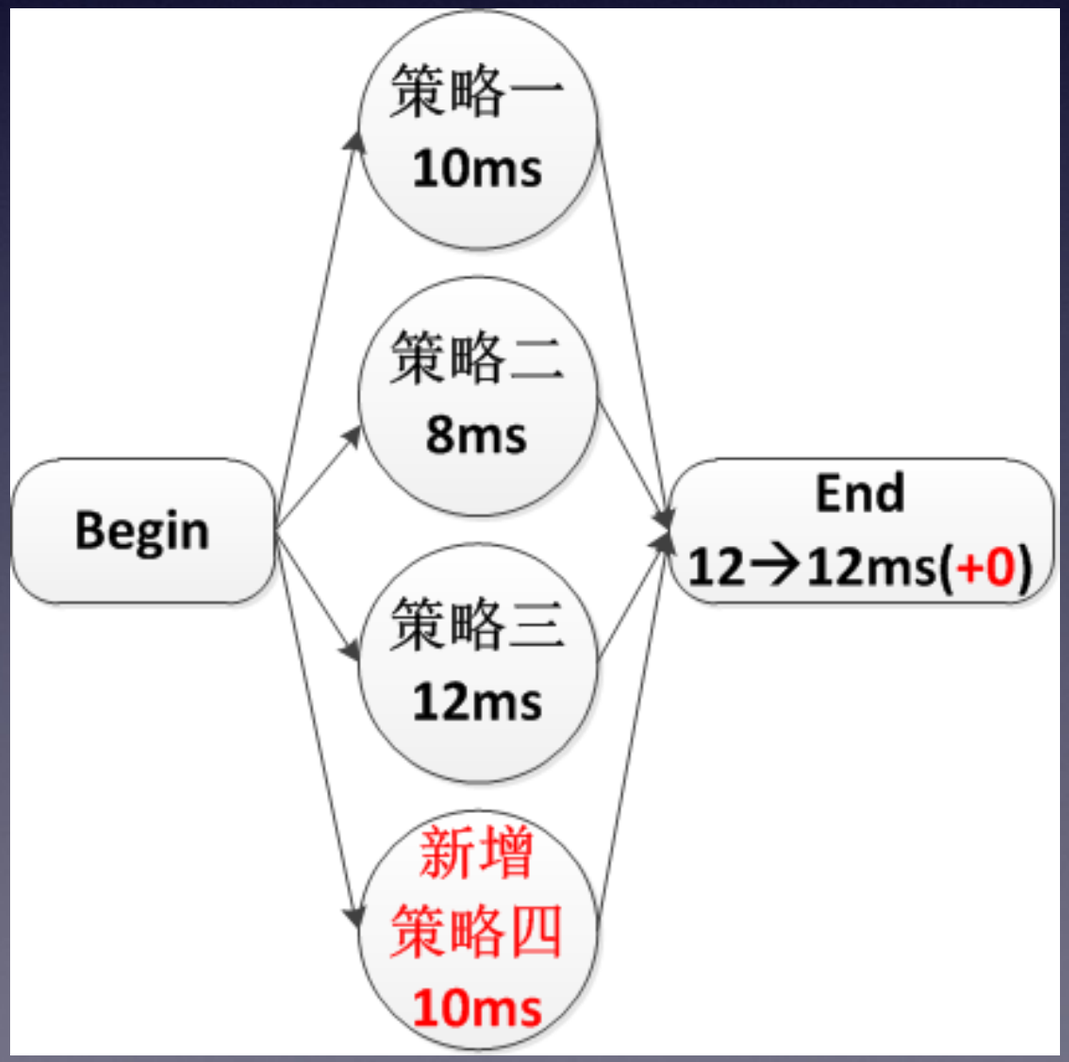
- 少量冗余换延迟
- 弹性架构
- 类似：
  - 投入更多帮助后进的
  - 放弃无药可救的
- 长尾，均值降低了75%



大规模系统的延迟，
串行是最大的阻碍

痛点：策略迭代与复杂化


Begin
策略一
10ms
策略二
8ms
策略三
12ms
新增
策略四
10ms
End
30→40ms(+10)


Begin
策略一
10ms
策略二
8ms
策略三
12ms
新增
策略四
10ms
End
12→12ms(+0)

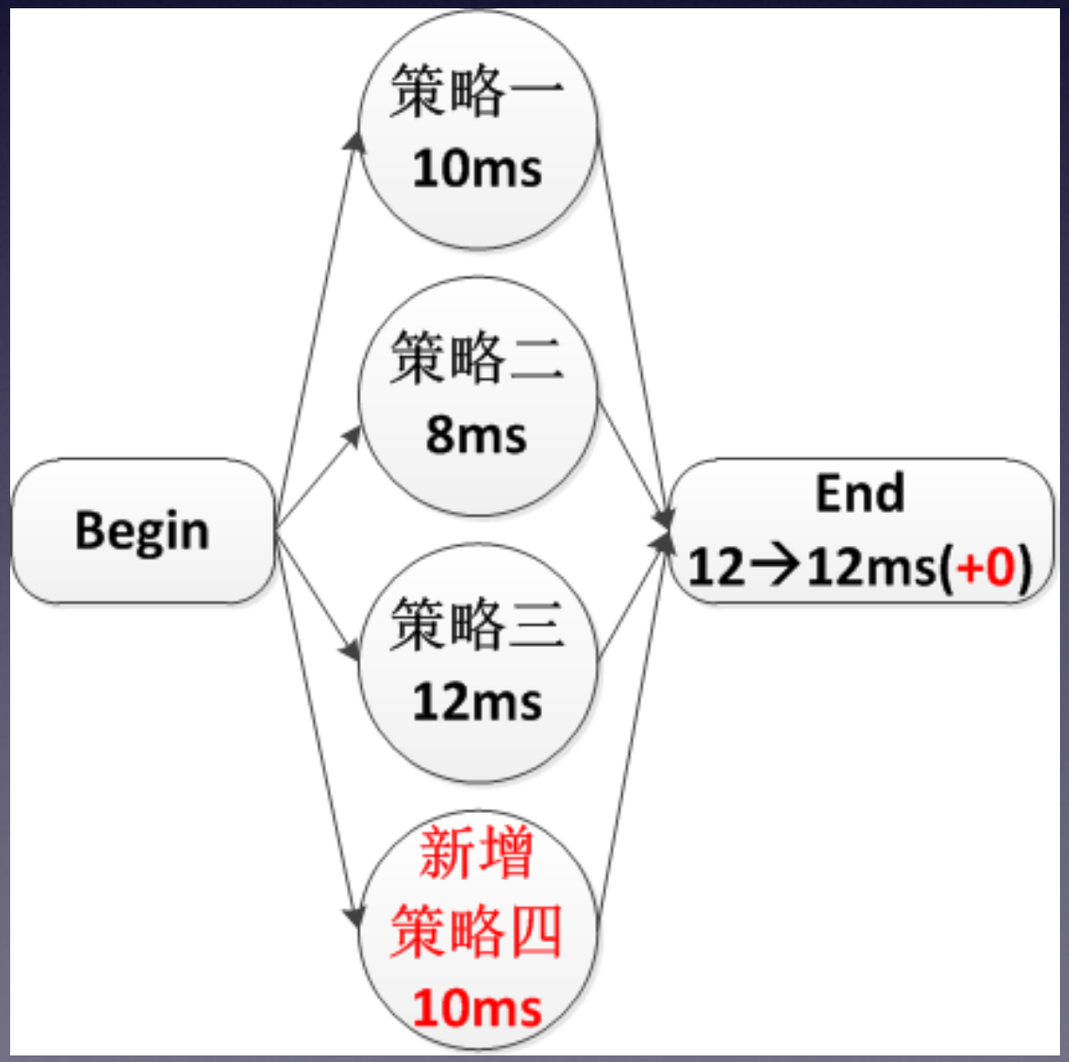
延迟：串行 VS 并行

并行的物理基础

- 多集群
- 多机
- 单机多CPU
- 单CPU多core
- (超线程技术)

简单例子，怎么改并行?


Begin
策略一
10ms
策略二
8ms
策略三
12ms
新增
策略四
10ms
End
30→40ms(+10)


Begin
策略一
10ms
策略二
8ms
策略三
12ms
新增
策略四
10ms
End
12→12ms(+0)

策略并行：聪明的写代码

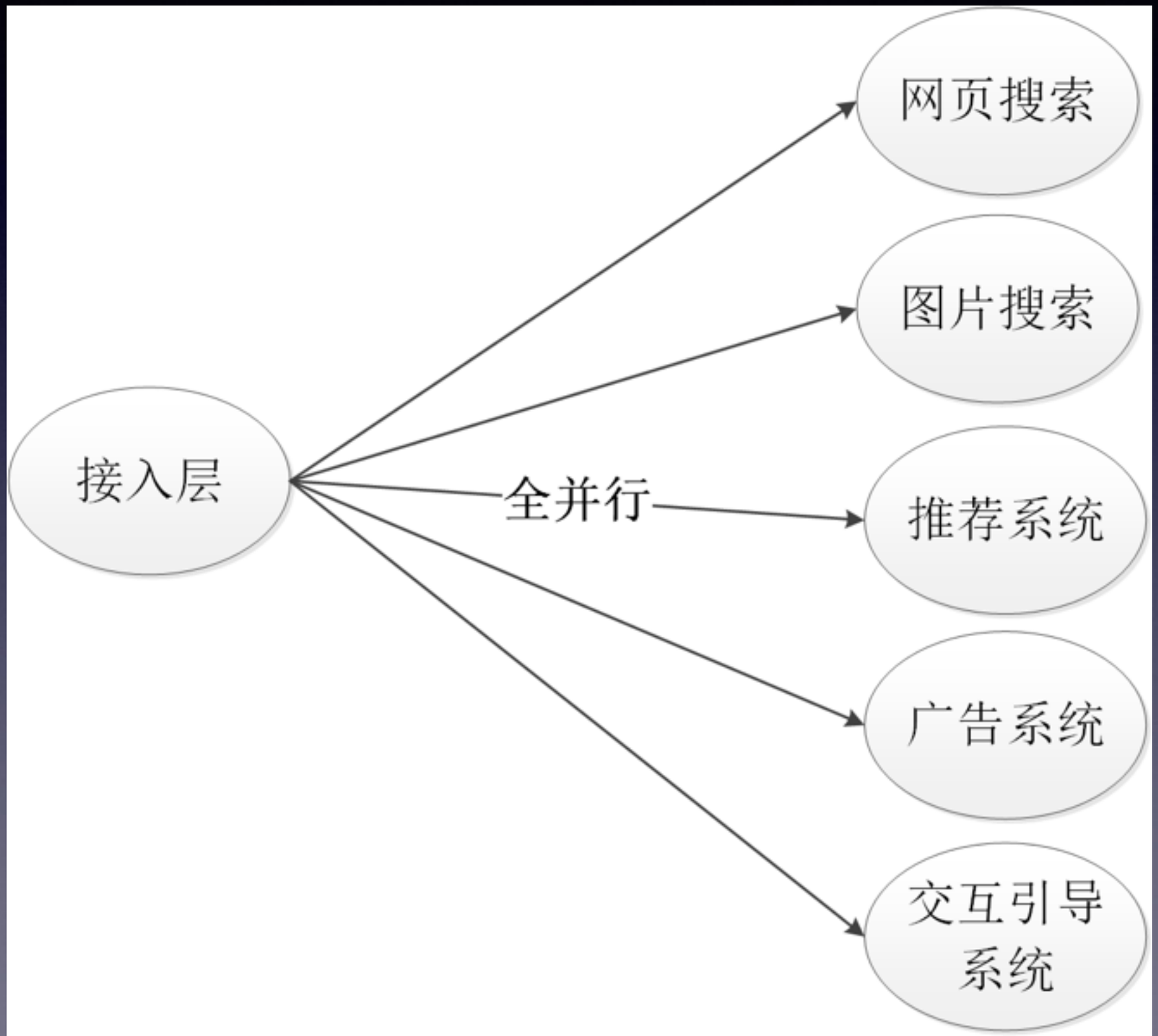
- 你可以：
  - 手写多线程+队列+互斥
- 也可以：
  - openmp
  - 一些通用库天生支持并行，如eigen

```
//一行代码，串行变并行
#pragma omp parallel for num_threads(6)
//多个独立策略
for (int i=0; i<N; i++)
{
    strategy(i, result+i)
}
//继续处理result数组
```

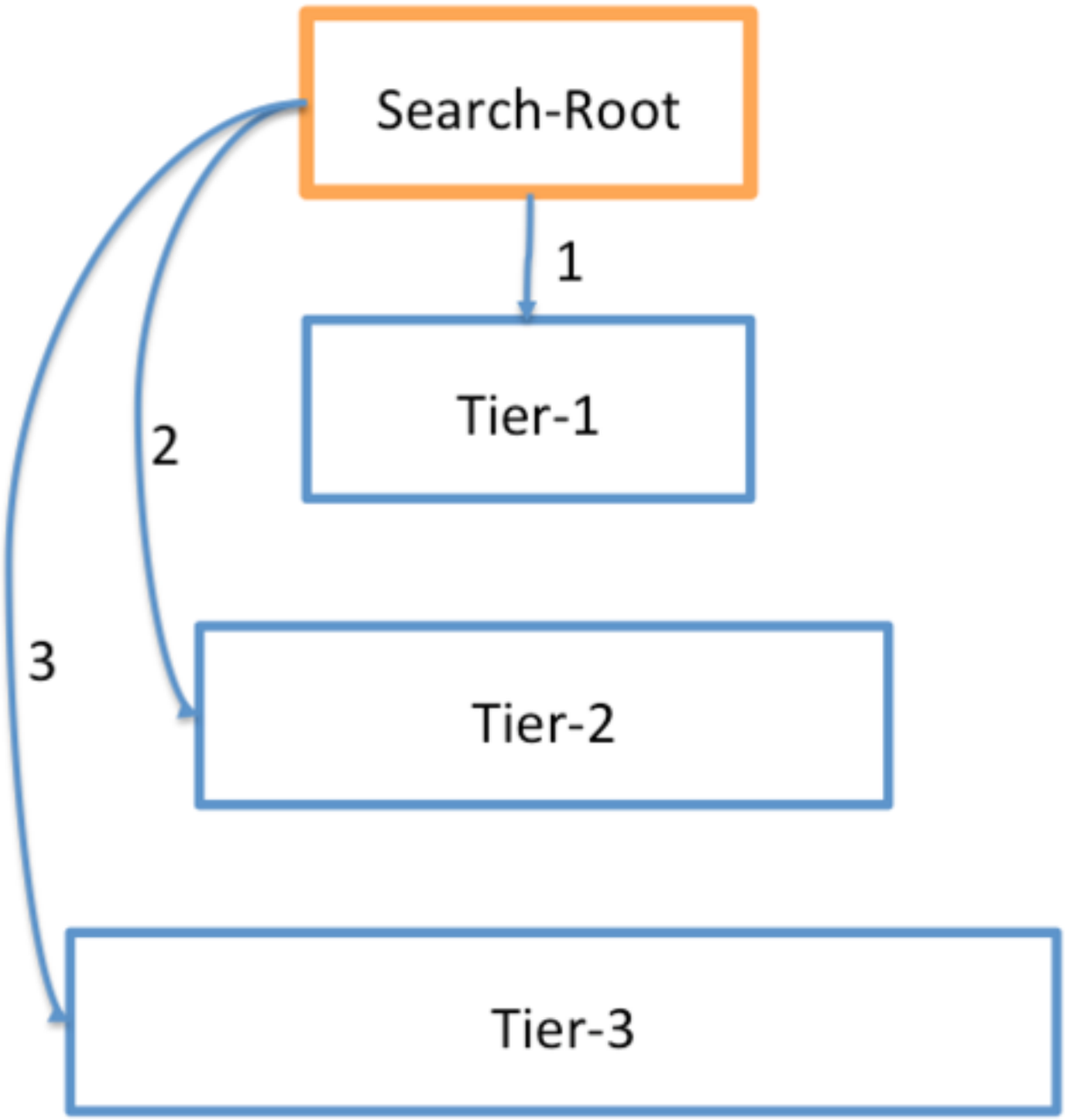
```
Matrix2d mat;
mat = mat*mat;
```

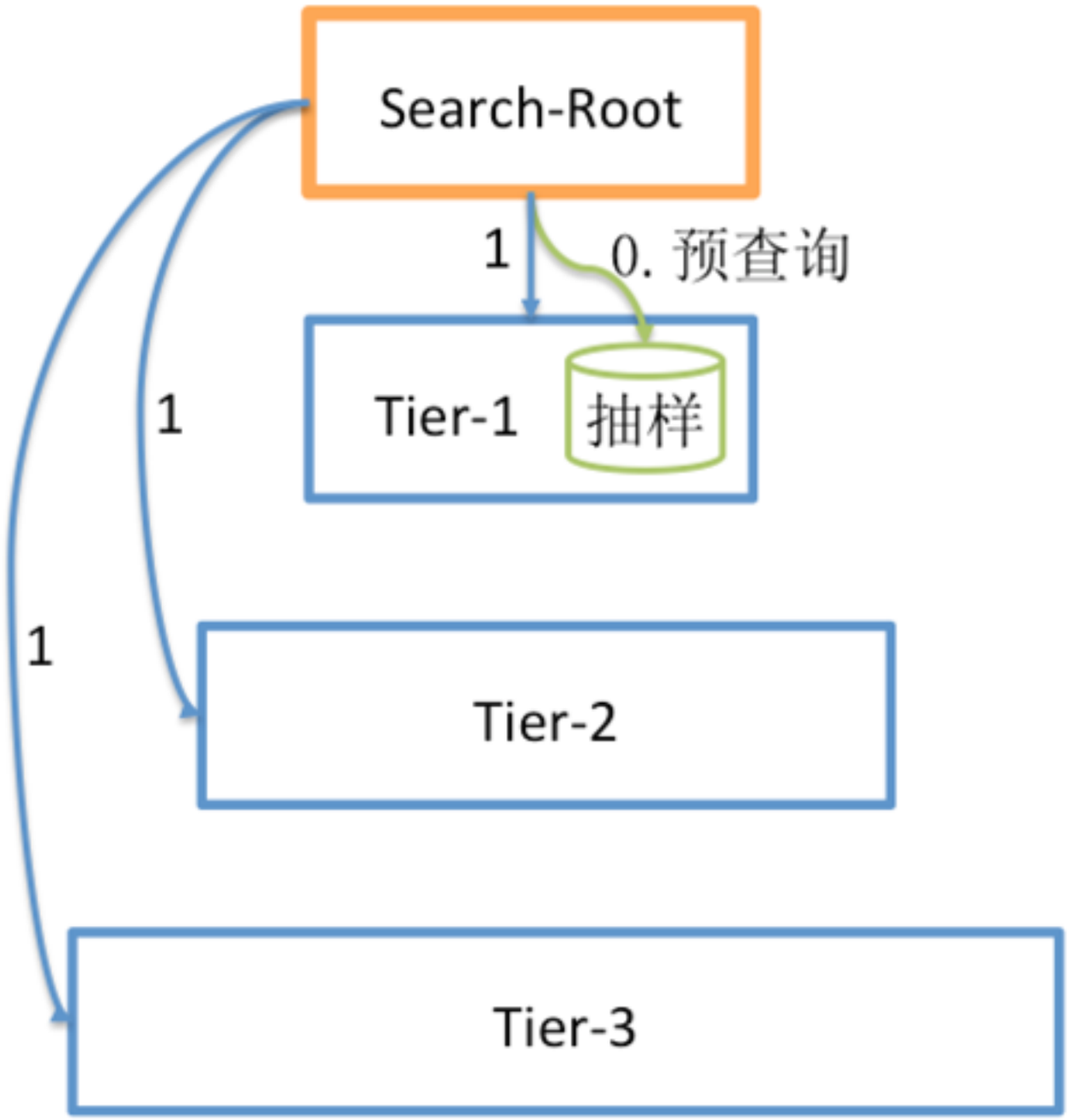
```
OMP_NUM_THREADS=8 ./test_eigen
```

服务间并行



预测 for 并行


Search-Root
1
Tier-1
2
3
Tier-2
Tier-3


Search-Root
1
0. 预查询
Tier-1
抽样
1
1
Tier-2
Tier-3

规模爆炸，更多架构挑战

可用性的挑战和优化实践

- 复杂系统的挑战：模块多，波动大，迭代多
- 弹性架构
- 抓大放小，保核心服务
- 动态策略，有多少能力做多少事
- 降级机制，Plan B, Plan C D…
- 容忍局部的不稳定，进程隔离

一些大趋势：大数据，实时

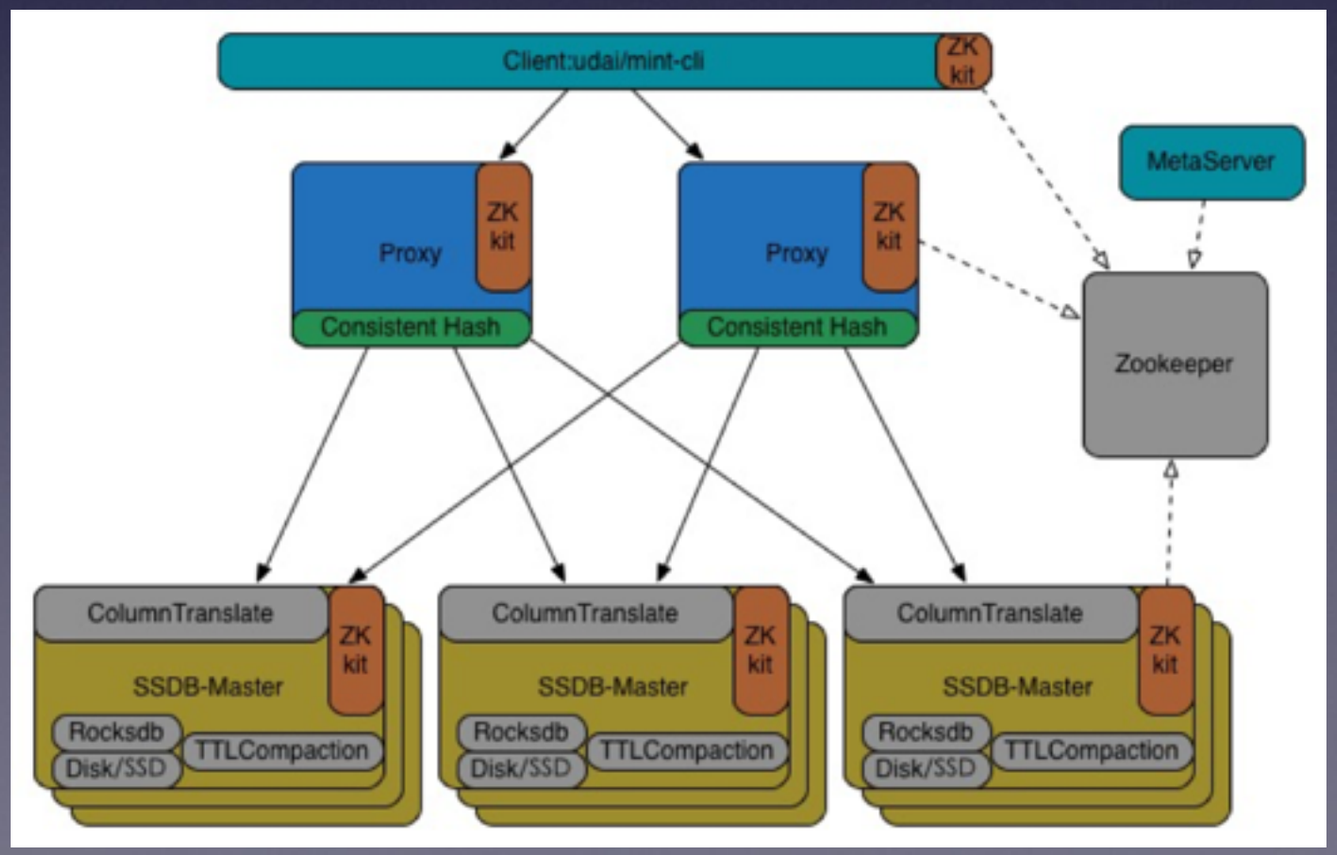
- 超大规模的存储系统，可扩展，低成本
- 实时读写，高QPS
- 在线，低延迟&高可用
- 灵活Schema，加减数据、版本

超大规模业务，定制存储

- “业务定制存储”
- 在线&离线，不同的解决方案

在线：数千台规模
Redis ->SSDB/Rocksdb

- 复用，搭积木，复用开源，回报开源（Tera等）



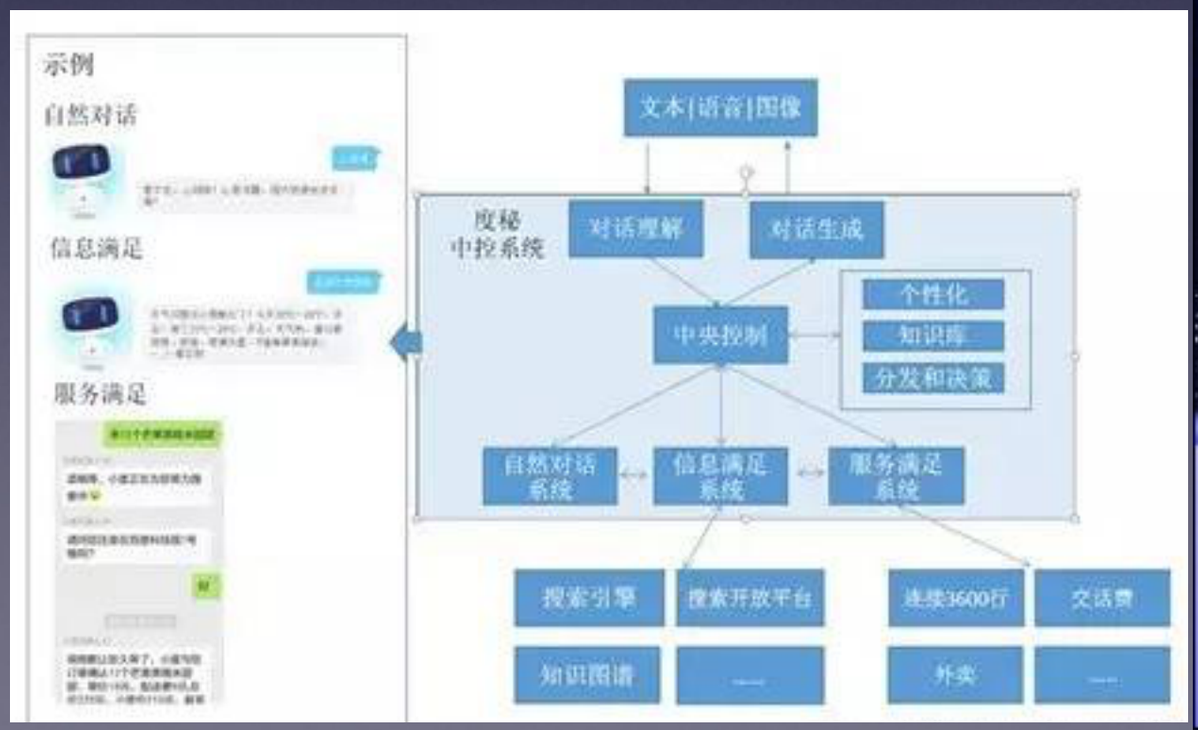
https://github.com/baidu

规模爆炸，更多挑战(略)

- 多IDC的挑战
- 离线计算，复杂度和实时性要求的提高
- 大规模机器学习的离线训练和在线使用
- 更多复杂子系统的引入
- 监控，统计与问题定位
- …

未来的一些趋势与挑战

- 奇点临近，规模继续爆炸
- 个性化
- 智能化



经验与教训

1. 宏观与微观优化并重
2. 结合差异化，分层架构很有效
3. 超大规模业务，适合定制系统
4. 建立好的机制，控制不劣化，比优化更重要
5. 好的架构师：从当前最大痛点出发 & 适当的预判
6. 大规模系统的复杂性是难以避免的，正视

Thanks!

欢迎加入我们，共同挑战规模爆炸!

上海 / 北京 / 深圳

wuyongwei03@baidu.com

Baidu 百度