高可用设计之道

JoeZou 邹辉

腾讯云

邹辉 湖北恩施 华科大

10年 华为+腾讯

不抽烟、不喝酒、五音不全、爱旅游

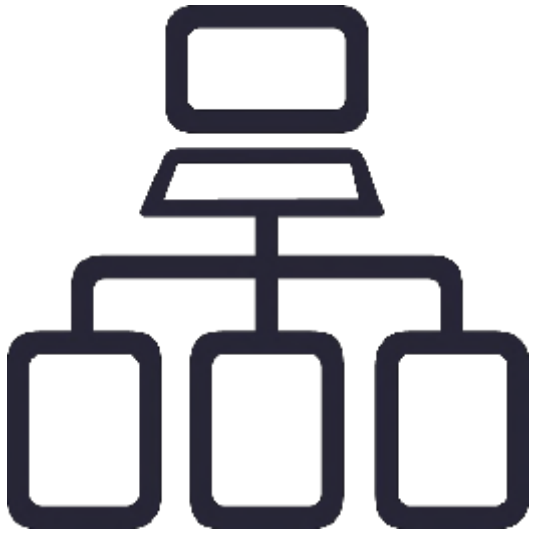
公共组件 存储服务 其它服务

云计算 产品化

DOCKER LB 监控 ……

腾讯云 基础产品中心 IAAS负责人

想与大家探讨的



- 如何衡量一个系统可用性？
- 构建一个高可用系统，可以从哪几个方面入手？
- 过载保护 、灰度发布、柔性可用 的概念及应用
- 常见的负载均衡技术实现。
- ……

1

可用性是什么？

- 影响可用性因素
- 可用性不是过度设计

2

架构设计 如何提升可用性

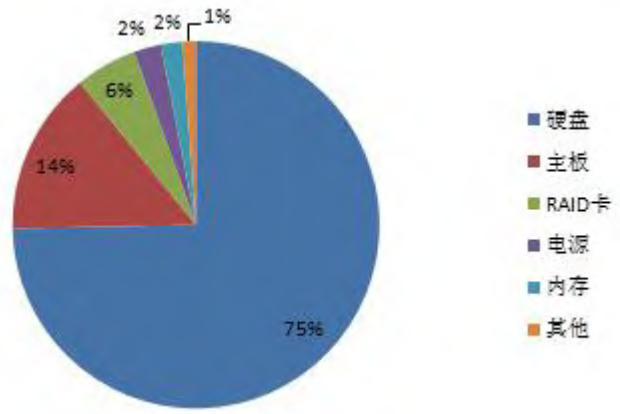
可用性的天敌：变更

4

天下武功 唯快不破

有人的地方就有故障

深圳 福永IDC-蛇口IDC间 线路被挖断

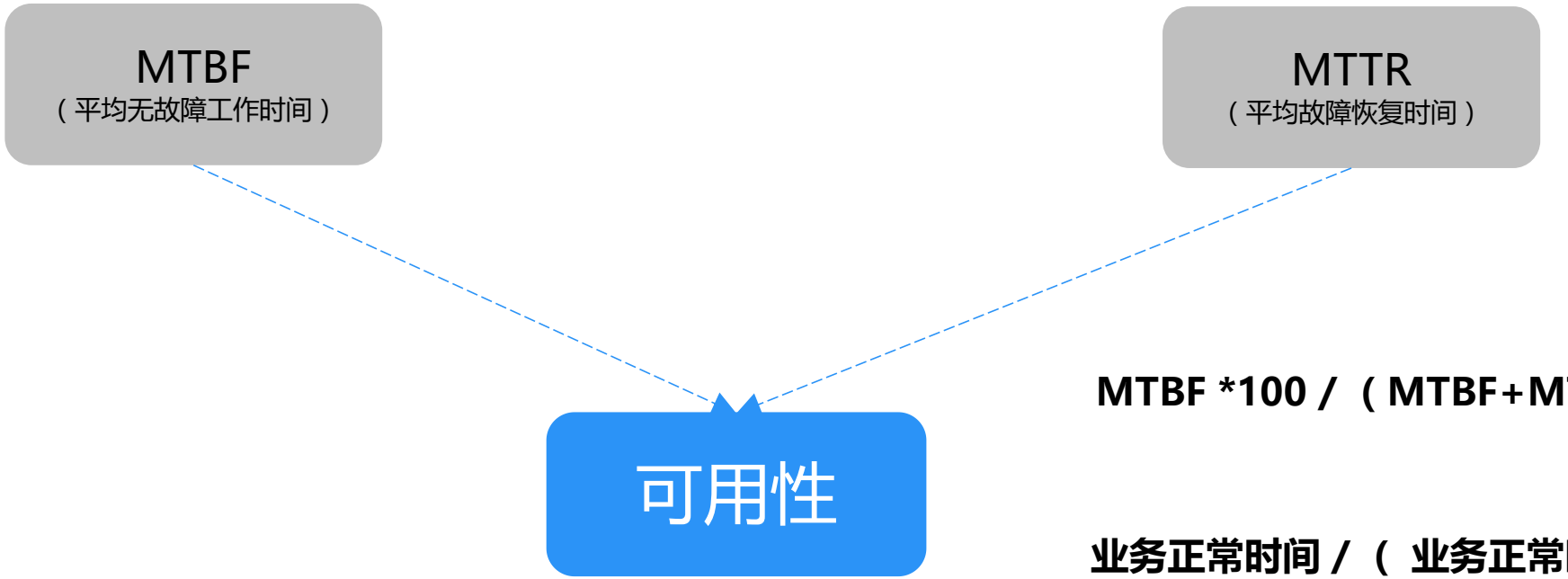


有人，就有意外（bug），有意外（bug）就有故障

故障，你碰上只是概率问题

软件	硬件	IDC	网络	人为
• OS挂死、重启 • 软件BUG • 变更发布	• 服务器宕机 • 磁盘只读 • 交换机故障	• 机架故障 • 机房故障 • 专线故障	• 光纤挖断 • 带宽满 • 流量陡增	• 误操作 • 遗漏 • 策略错误

可用性的概念



MTBF *100 /（MTBF+MTTR）%

业务正常时间 /（ 业务正常时+业务异常时间）

例如：XXX 业务一年中平均每3个月发生过一起故障，每次故障平均要20分钟才能恢复正常，那么业务 年可用性等于

*90*24*60 /（ 90 *24 *60 +20）=99.98%*

没有故障，就没有架构师这个职业

少出故障
自动恢复

快速发现故障

快速定位故障

快速解决故障

我们不能控制别人，但是我们能够控制自己

高可用 ！= 完美 ！= 过度设计

业务场景

成本

可用性

完美架构

矛盾 平衡

系统给谁用？ 核心价值在哪?

时间、人力、资源 成本如何？

架构师

可用性要求 有多高？ 与核心价值是否冲突？

系统 是否可以拆分 核心和非核心？

可否利用一些 存在的开源软件或者云服务？

脱离业务场景 去做架构设计 就是耍流氓！！

1 可用性是什么？

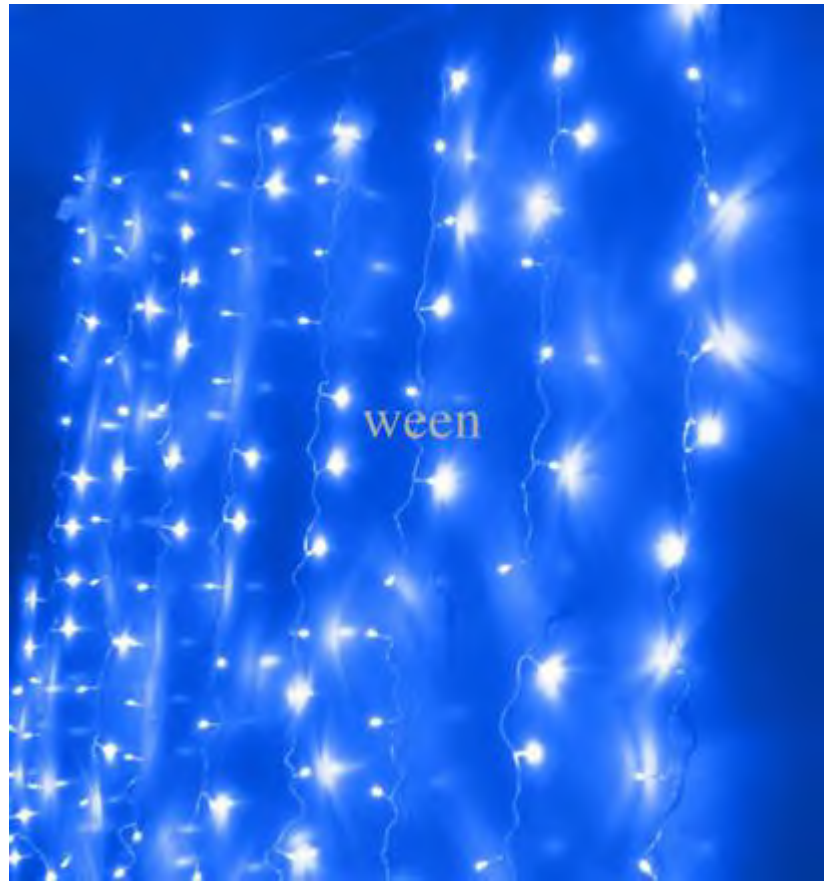
2 架构设计 如何 提升可用性

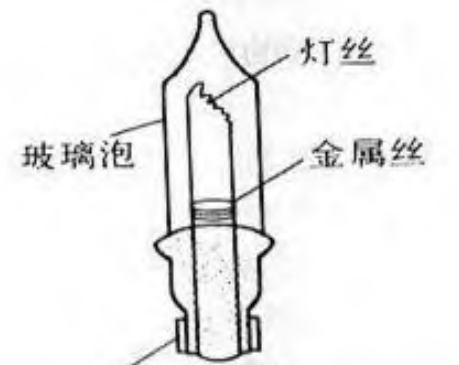
- 小彩灯带来的思考
- 腾讯典型设计思想

3 可用性的天敌：变更

4 天下武功 唯快不破

节日小彩灯带来的思考



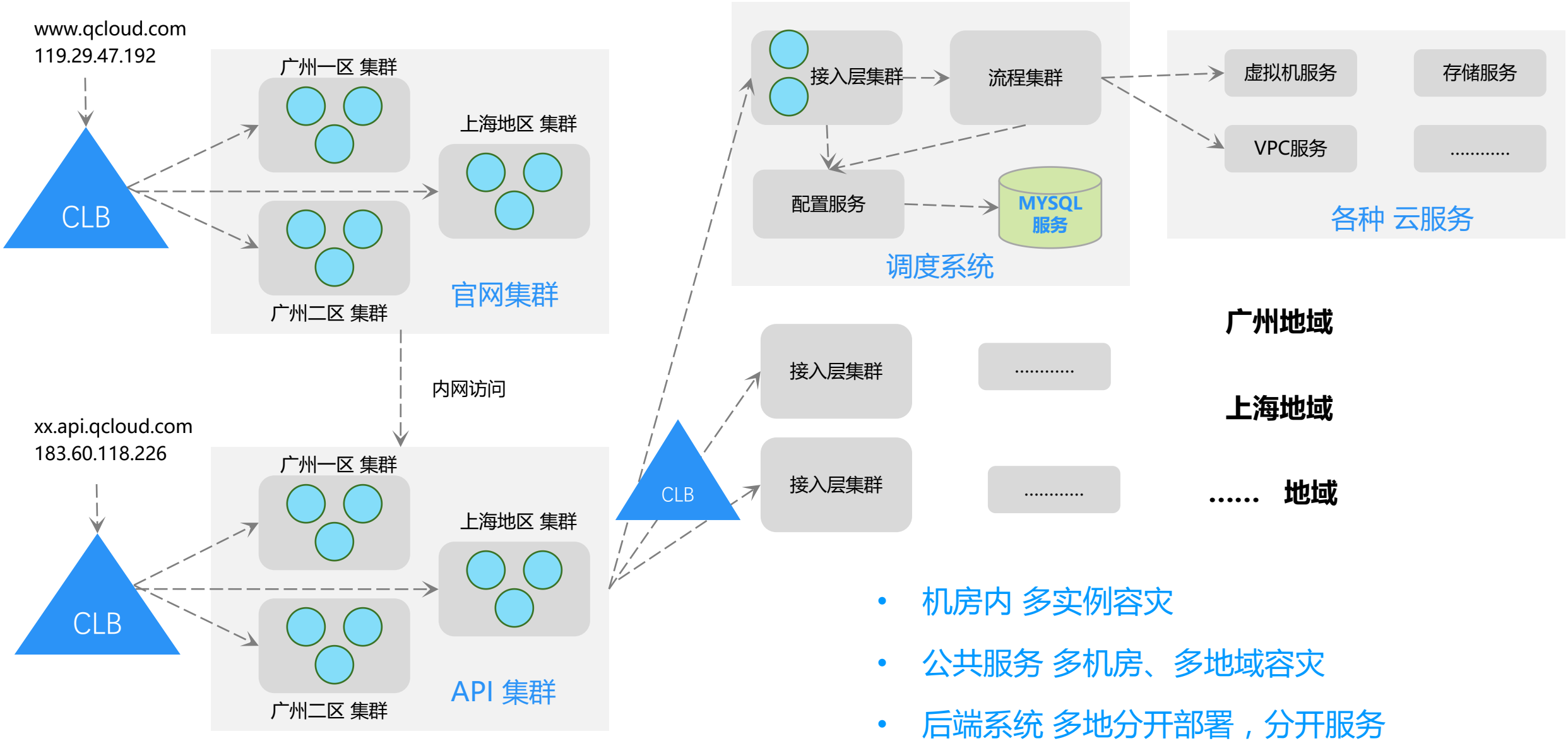


小彩泡的构造

金属丝 绝缘层，灯丝有问题时，绝缘层熔断开始导电

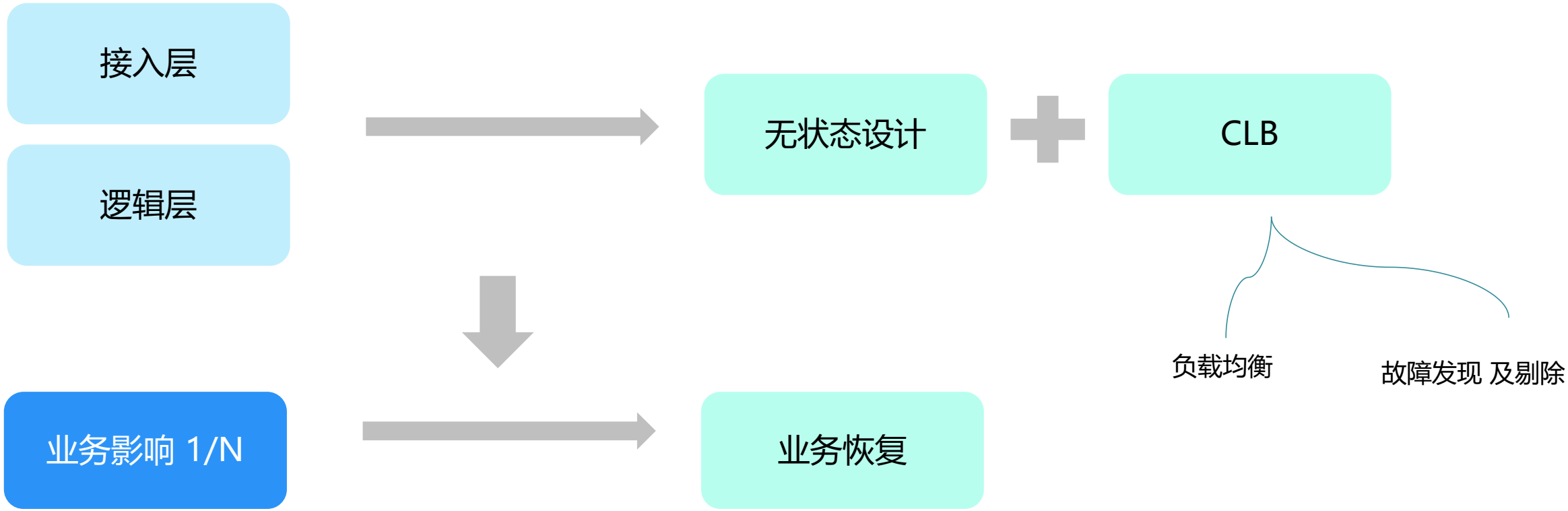
并联部署： 上线的基本要求

串联到并联的改变



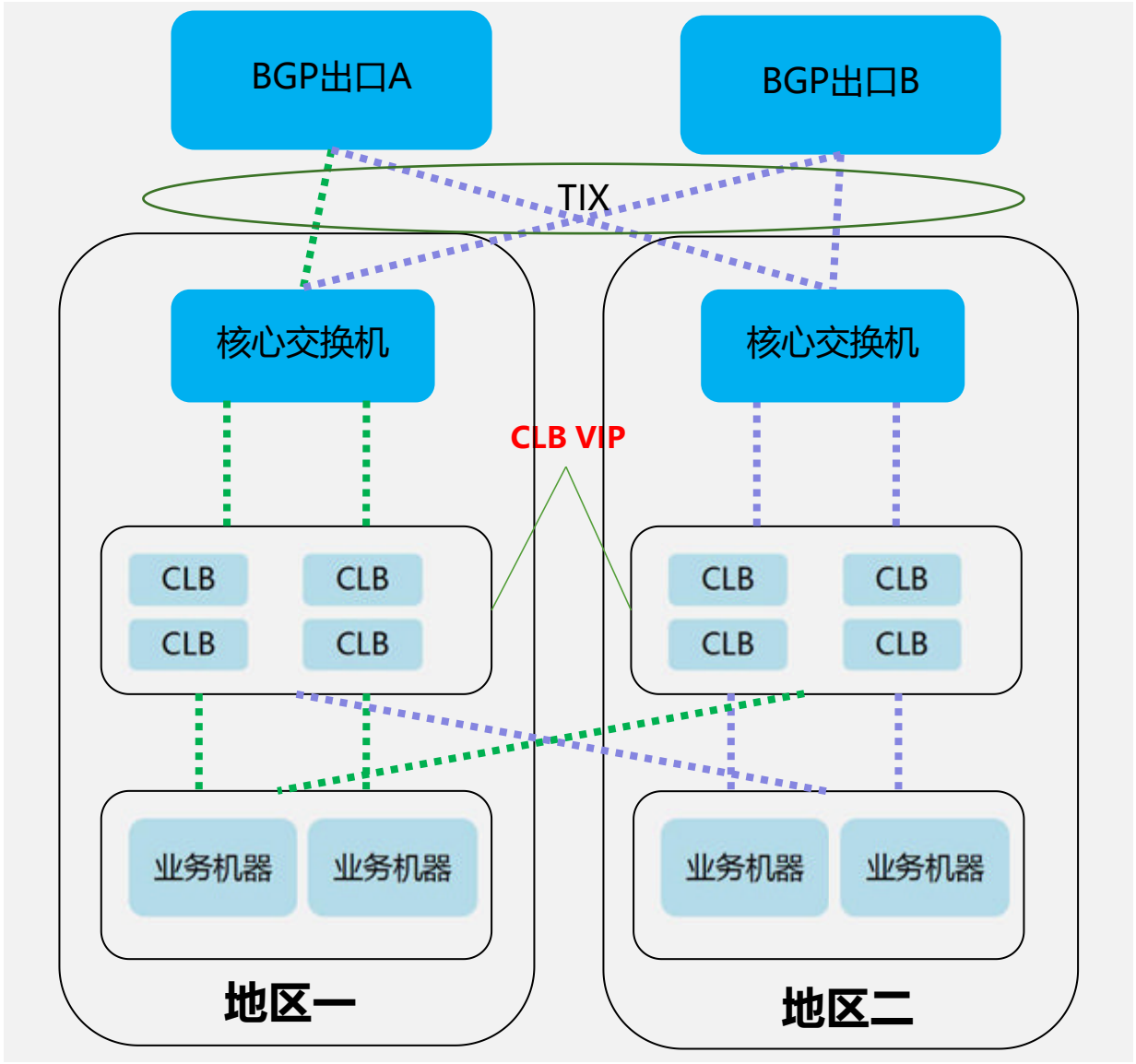
- 机房内 多实例容灾
- 公共服务 多机房、多地域容灾
- 后端系统 多地分开部署，分开服务

并联在 接入层+逻辑层 的应用

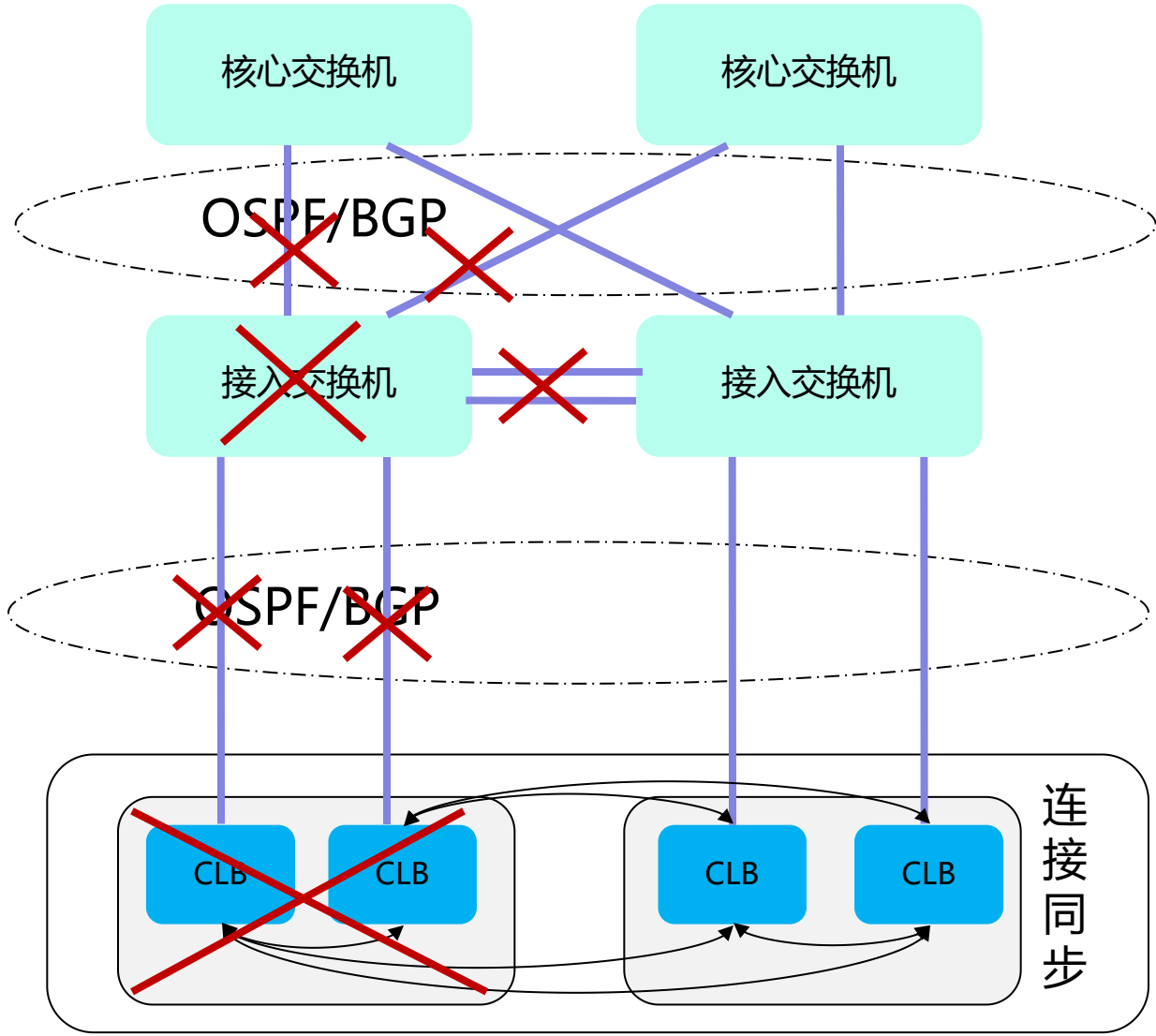


该无状态的就无状态，该CLB的就CLB

CLB 本身的并联设计

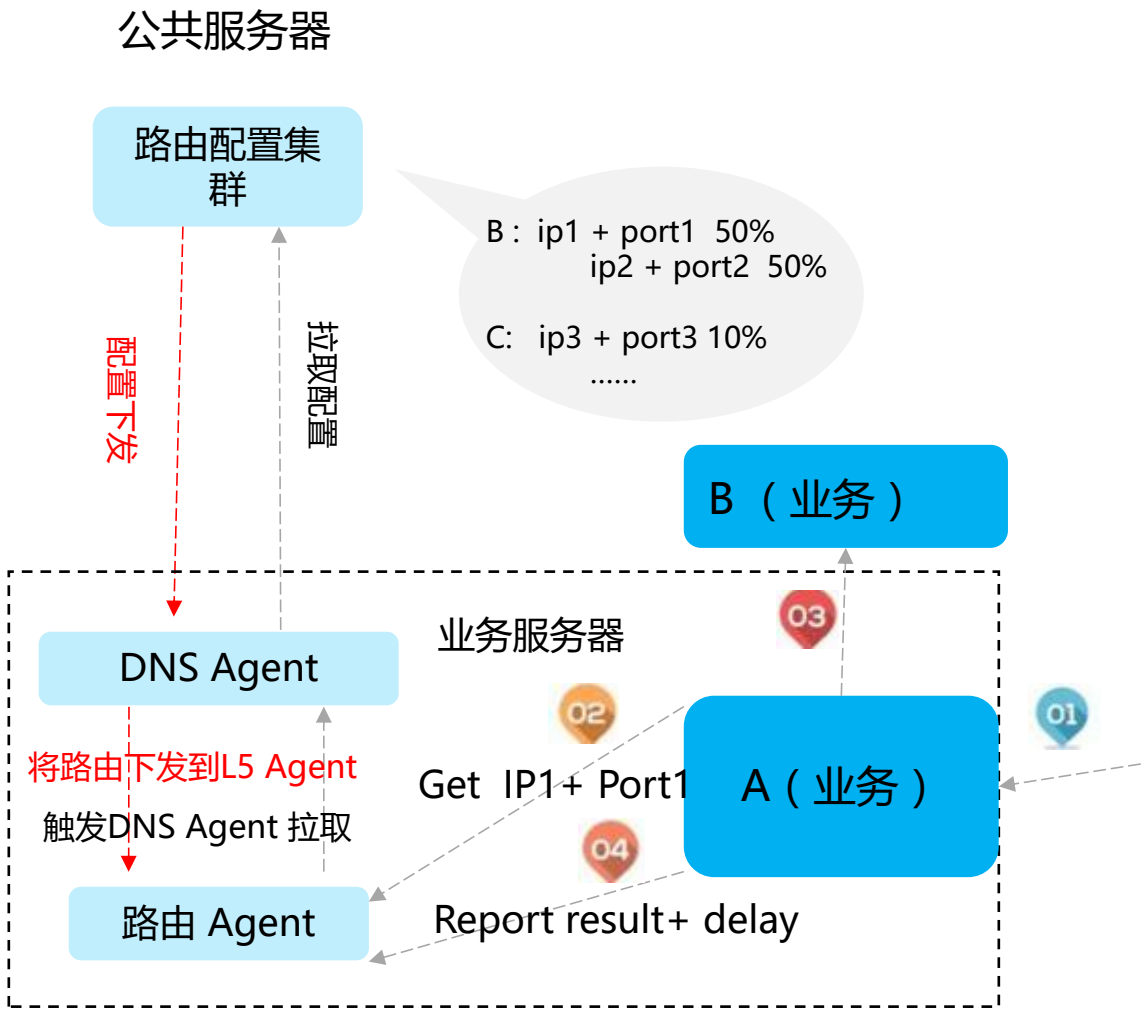


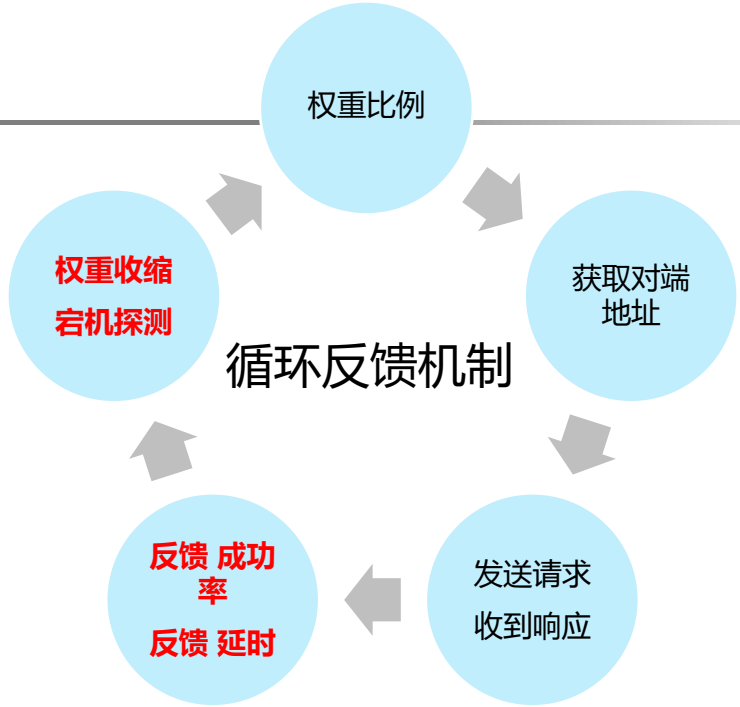
地区之间 容灾方案



地区内部 容灾方案

CLB的另一种思路

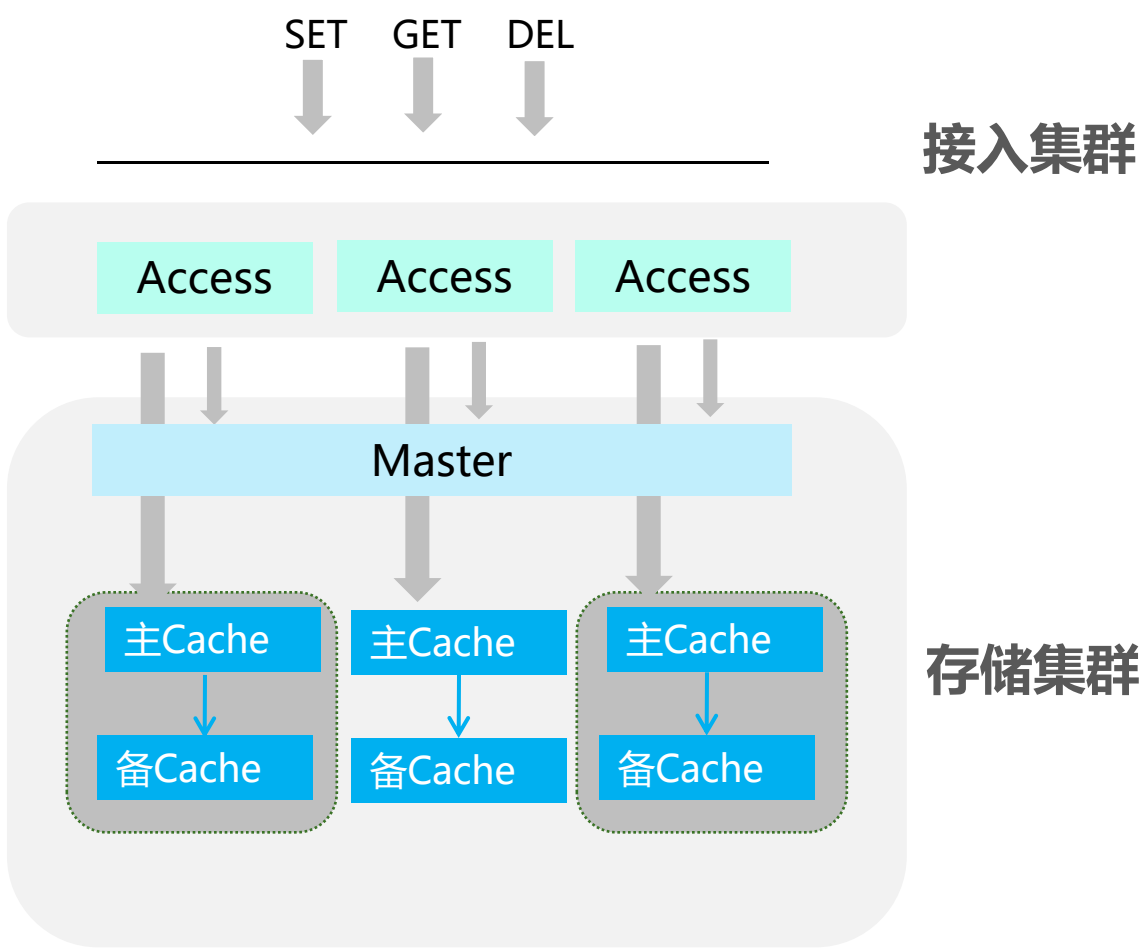




	CLB(LVS)	L5
功能与应用	外网、内网 负载均衡，容错，IP收敛、多运营商接入	内网负载均衡，容错
关键路径	构成业务请求的关键路径，增加一次网络通讯，和成功率有下降风险；延时	本质是路由决策系统，不构成业务请求的关键路径；故障对业务几乎无影响
成本	对机房和网络有一定要求，属于比较稀缺资源	内网任何地方可以使用，几乎零成本
运行方式	以VIP方式提供，对业务透明	需要业务修改代码调用API
容灾能力	外围探测，算法比较简单，敏感度低	业务应用层反馈，可靠性更高；算法基于更智能；可个性化定制

基于客户端的 负载均衡 L5

并联 在缓存+数据库层 的应用

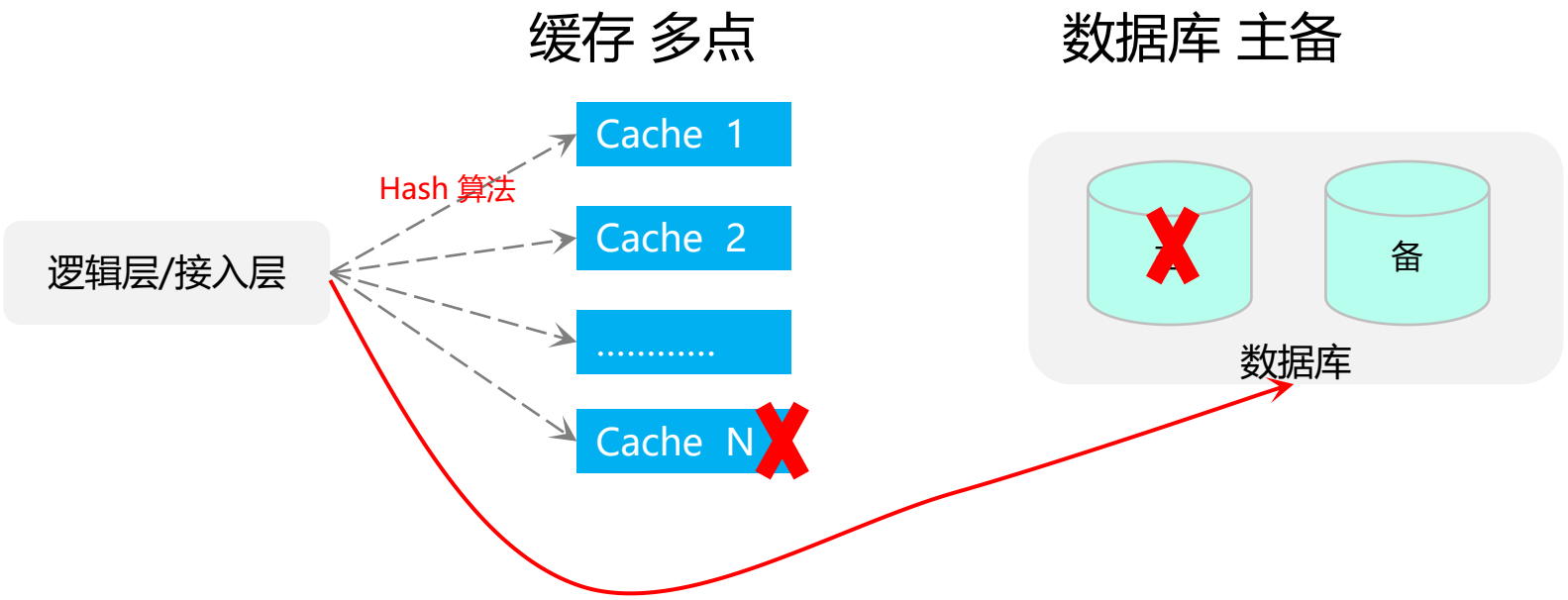


多套Cache，分布在多台机器

同Cache 主备部署，分布在两台机器

Cmem 缓存服务

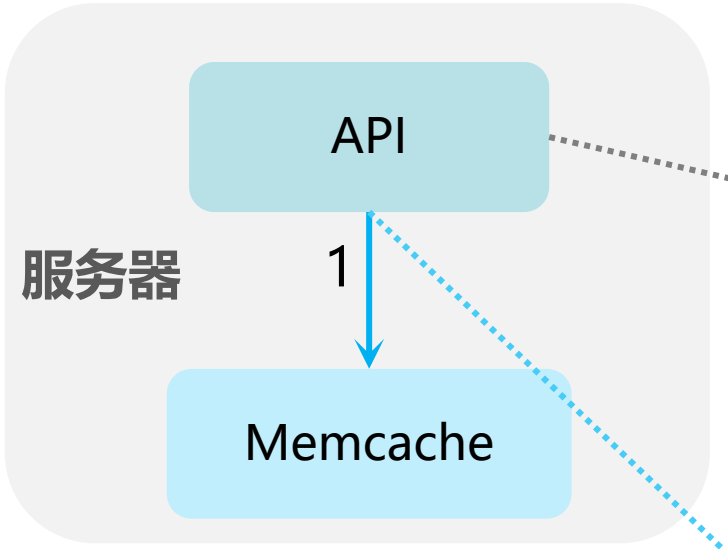
缓存要多，数据库要主备



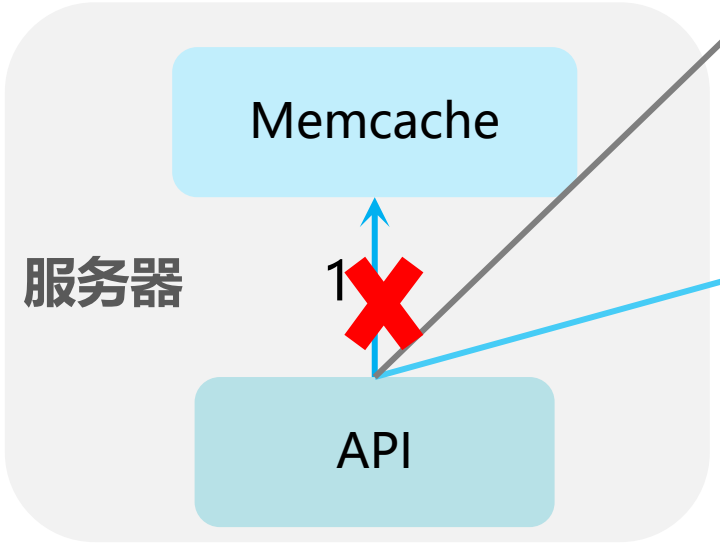
1/N 数据 受影响、请求转移到 数据库

自研的复杂性。建议用 云上的缓存+数据库服务

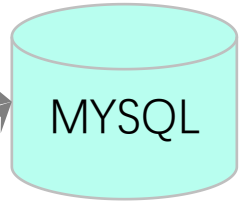
核心数据 要多级容灾



缓存过期时间 30分钟



API 核心数据： 接口 配置信息



3

缓存过期时间 30分钟

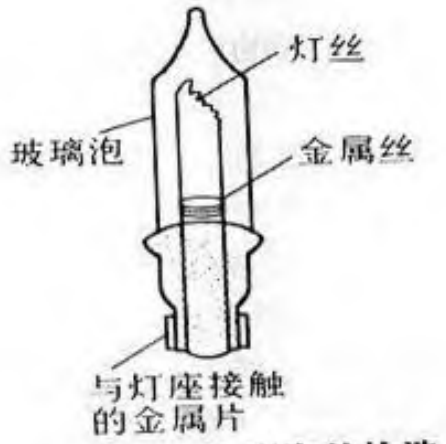
优先拉取本机Memcache ?

其次拉取CKV, 回写本机Memcache

最后拉取Mysql， 回写 Memcache 、CKV

延时 + 高频 + 高可用 需求

如何降低单个小彩灯故障？



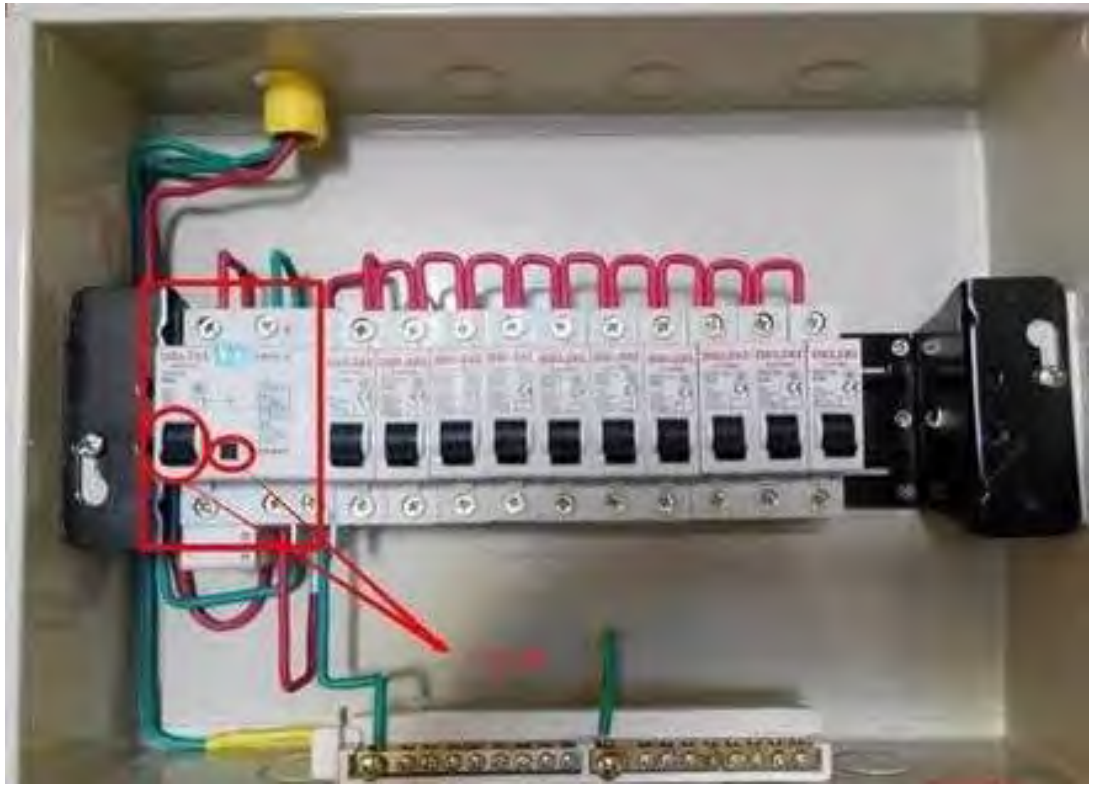
小彩泡的构造

除了架构上的并联，面对故障，我们还能做什么呢？

软件系统中的 雪崩效应 多米诺骨牌

我们需要把故障控制在一定范围！

过载保护，让故障得到控制



配电箱　跳闸

保险丝 熔断

电流过大时，自动跳闸 熔断，保护线路和电器

过载保护的方法

分离部署

- 快慢分离
- 轻重分离
- 用户分离

及早拒绝

- 后端反馈给前端繁忙
- 前端 保护后端
- 后端直接拒绝

量力而为

- 设置超时时间
- 设置队列长度

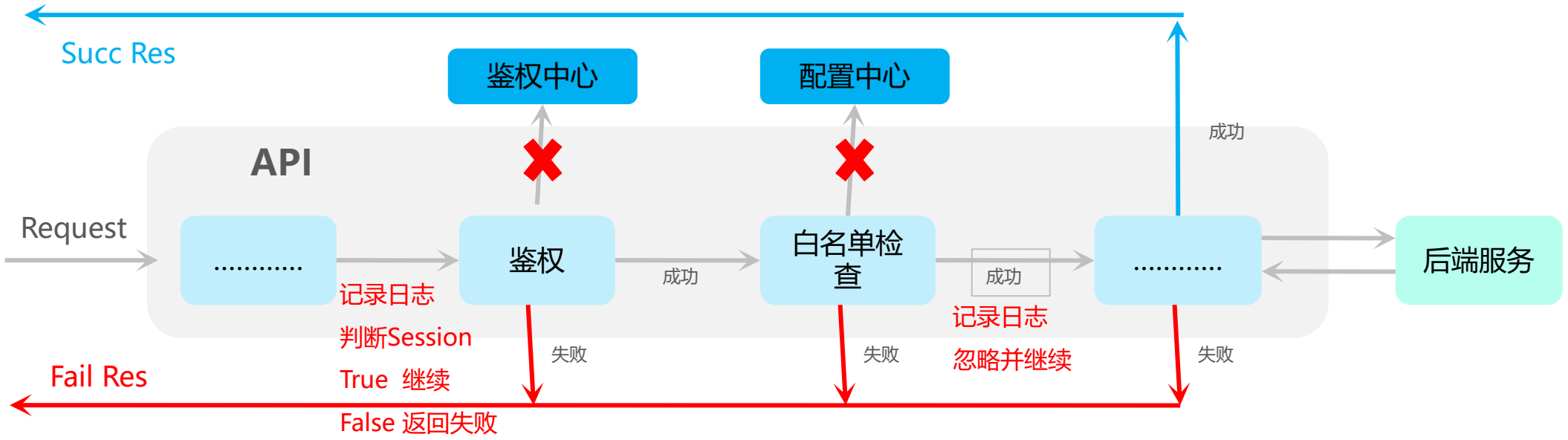
动态调节

- 监控告警
- 自动调节

如何降低单个小彩灯故障？

控制住故障后，如何让故障对业务的影响降到最低？

代码写的好不好，要看接口的柔性



接口级别的柔性可用：绕过故障的非关键接口

滴滴中的舍与得？

救火

重新检视产品逻辑

- **准确**显示我的当前位置
- 显示附近的出租车师傅
- **智能复杂**的分单策略
- **准确定位**出租车的位置
- 保持出租车位置**实时同步一致**

高峰期瘫痪

- 极致体验？
- 无损设计

- 理想很美好，现实很残酷

有损架构师的紧急手术

2天完成的有损手术

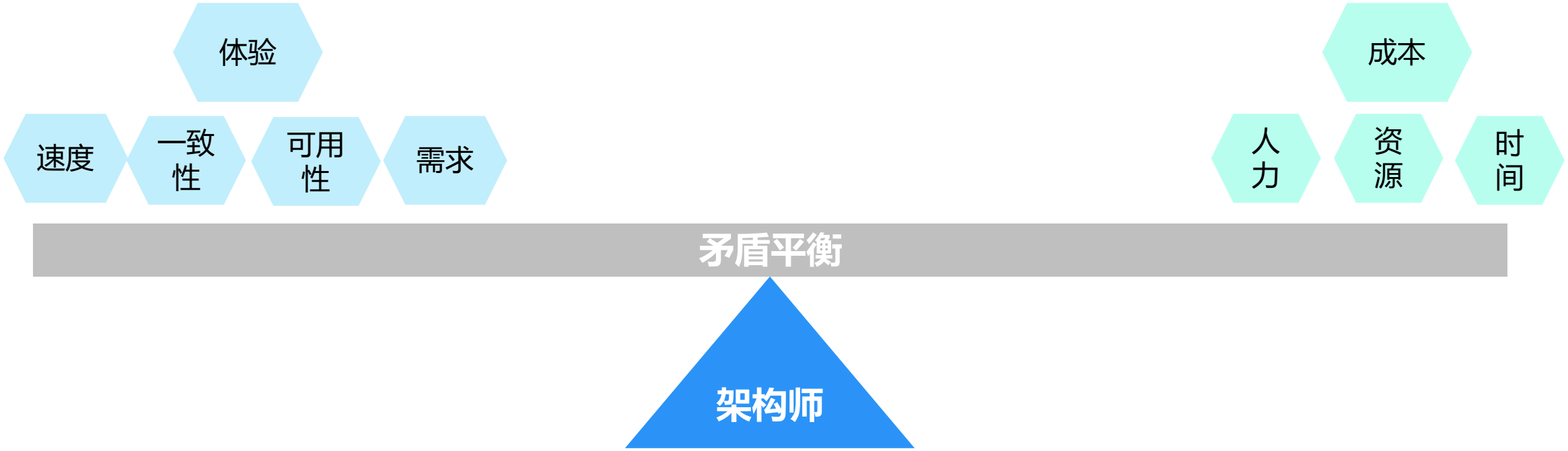
- 柔性处理我的当前位置
- 显时最近时间部分附近出租车
- 高峰期算法可降级的分单策略
- 取消实时定位出租车当前位置

针对高峰有损设计

- 不加设备下，扛住70%高峰请求

服务级别 的 柔性可用： 根据资源确定不同服务级别

抓住最核心的，学会讨价还价



产品级别的柔性可用：学会砍需求~

好不好，看代码

优秀的代码 是 高可用架构的基石 ！

1 可用性是什么？

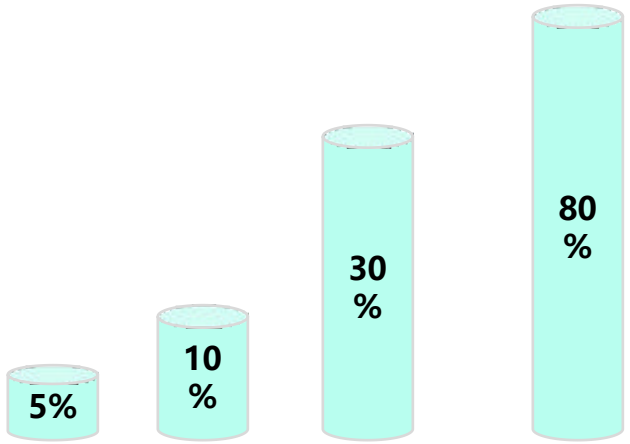
2 架构设计 如何提升可用性

3 可用性的天敌：变更

- 环境问题
- 灰度发布

4 天下武功 唯快不破

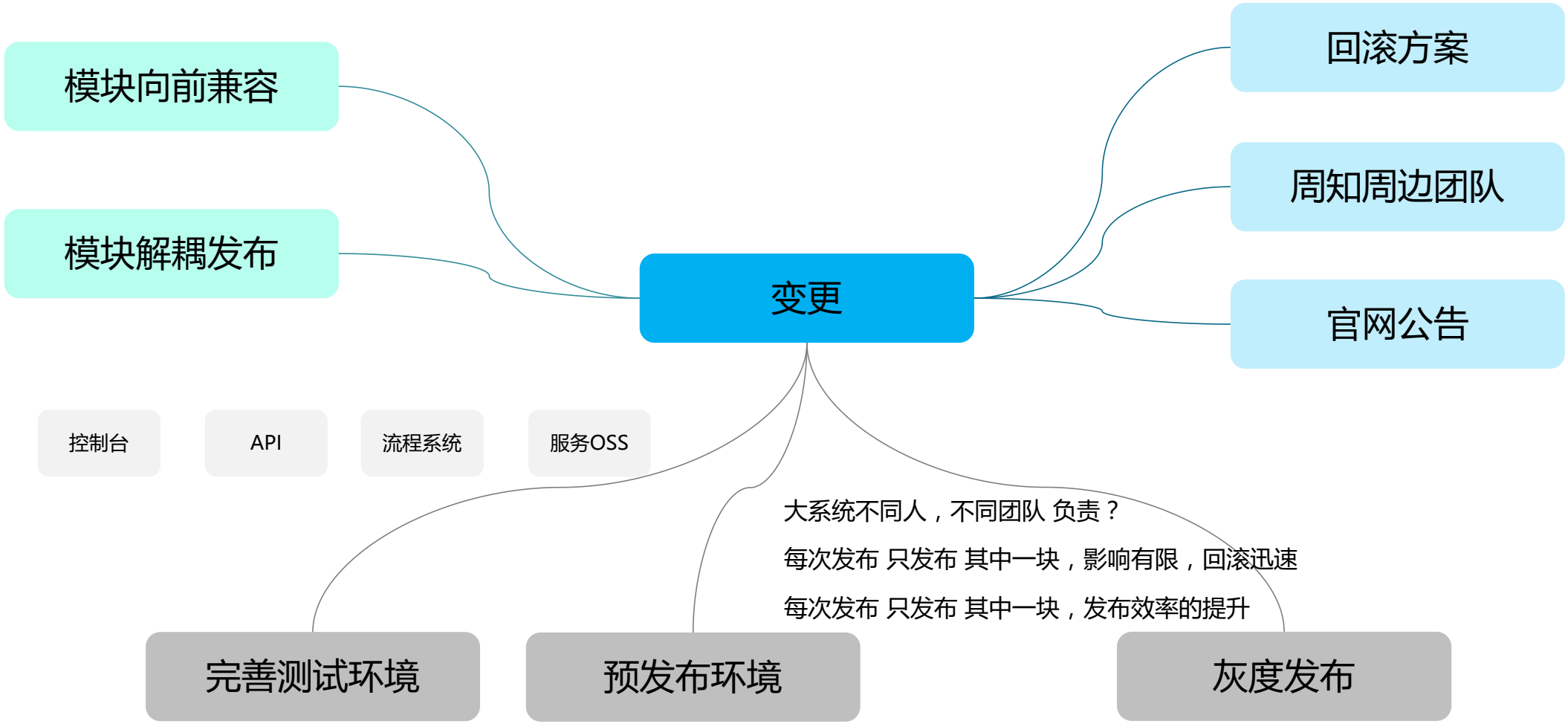
变更、高可用的最大敌人



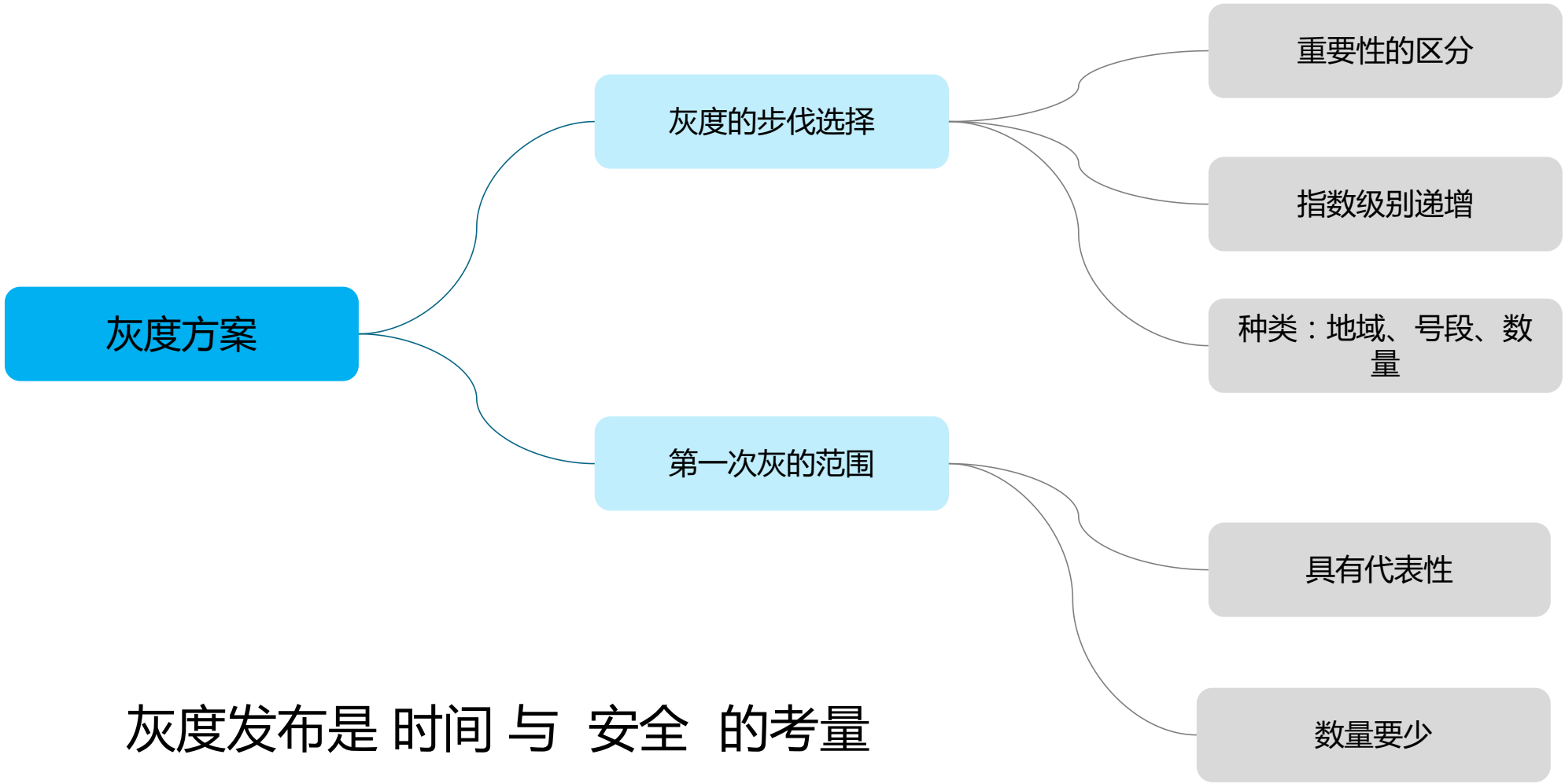
变更带来的故障 率？

变更流程是一个持续优化改进的过程

如何减少变更带来的故障



灰度发布 高可用的 最后一道关口



灰度发布是 时间 与 安全 的考量

1 可用性是什么？

2 架构设计 如何提升可用性

3 可用性的天敌：变更

4 天下武功 唯快不破

在客户发现问题之前发现问题

天下武功、无坚不摧、唯快不破

问题的发现 最快可以做到 秒 级~

网络拨测

硬件资源监控

软件日志监控

CGI、接口拨测

网络延时、吞吐

模调/运行数据

模拟服务拨测

进程监控

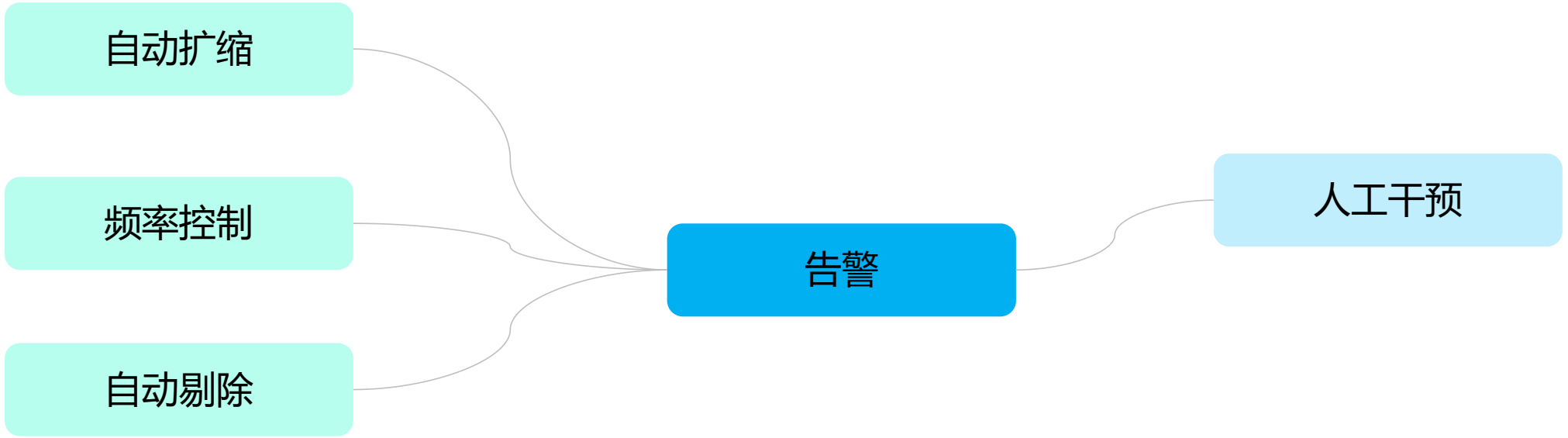
统一告警平台

电话

短信、微信

邮件

能不处理的不处理、能自动处理的自动处理、最后再人工干预



回顾

架构师的职责：取得产品与架构的平衡

MTBF

MTTR

并联系统

无状态+CLB

多点 + 主备

多级容灾

过载保护

柔性可用

代码基石

环境齐备

灰度发布

监控完善

Q&A