架构迎接未来变化
IAS2017 • NANJING

Cloud Native架构一致性问题及解决方案

作者：王启军

王启军

华为公司架构部资深架构师，负责华为公司的Cloud Native、微服务架构推进落地，前后参与了华为手机祥云4.0、物联网IoT2.0的架构设计。曾任当当网架构师，主导电商平台架构设计，包括订单、支付、价格、库存、物流等。曾就职于搜狐负责手机微博的研发。目前运营微信公众号“奔跑中的蜗牛”，热爱开源，热爱分享。

个人公众号：奔跑中的蜗牛

2017/12/8

理论

强一致性

2PC/3PC

最终一致性

重试

可靠事件

Saga

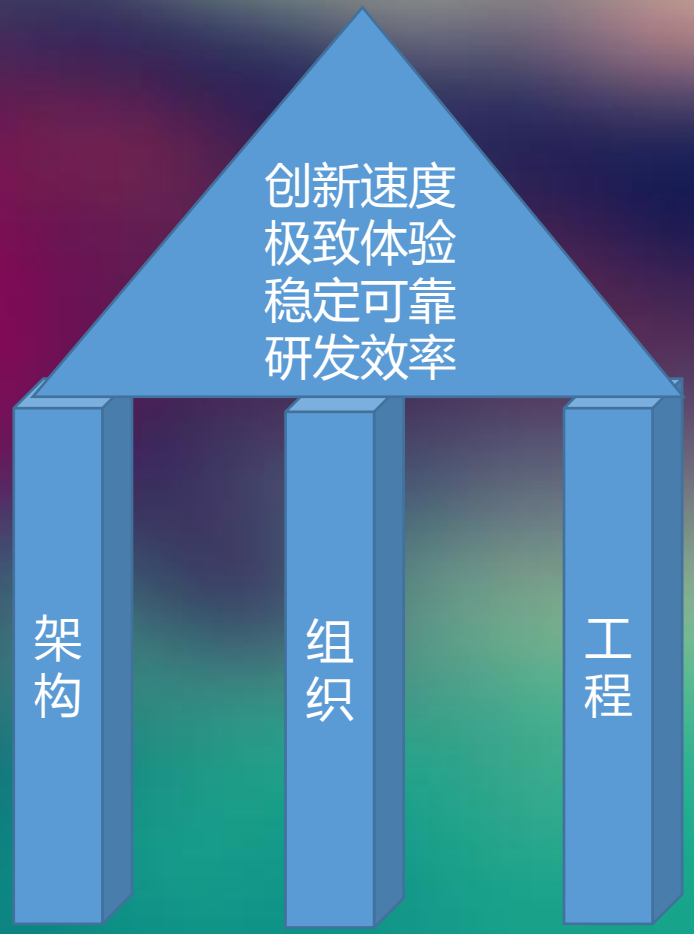
TCC

幂等

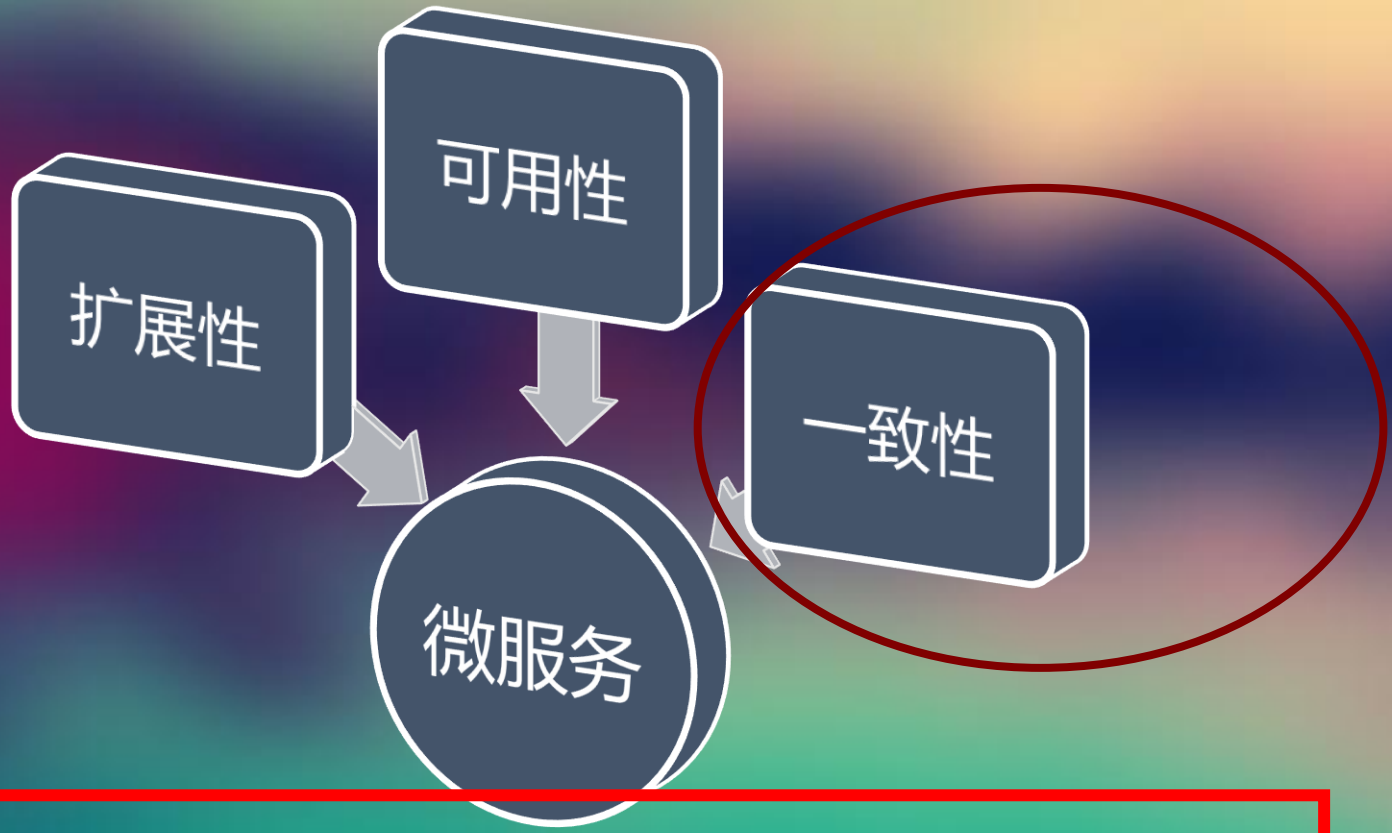
1

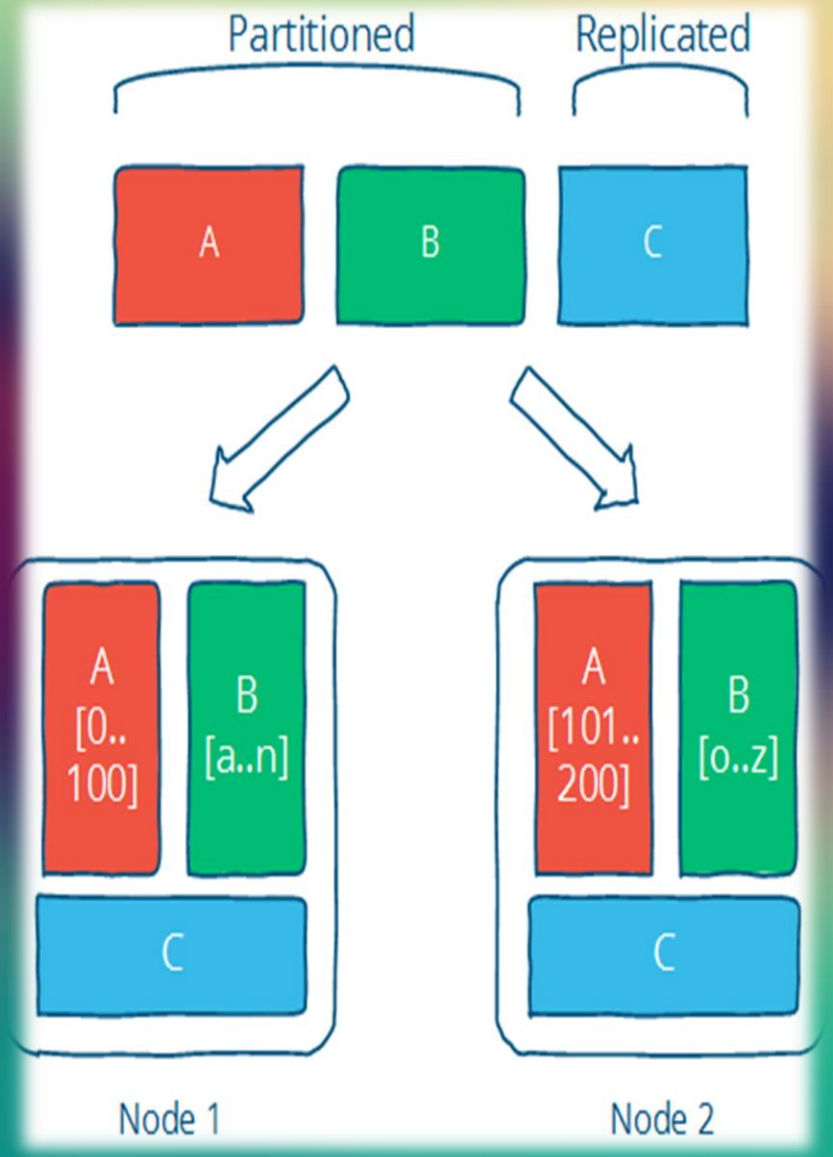
基础理论

Cloud Native的组成

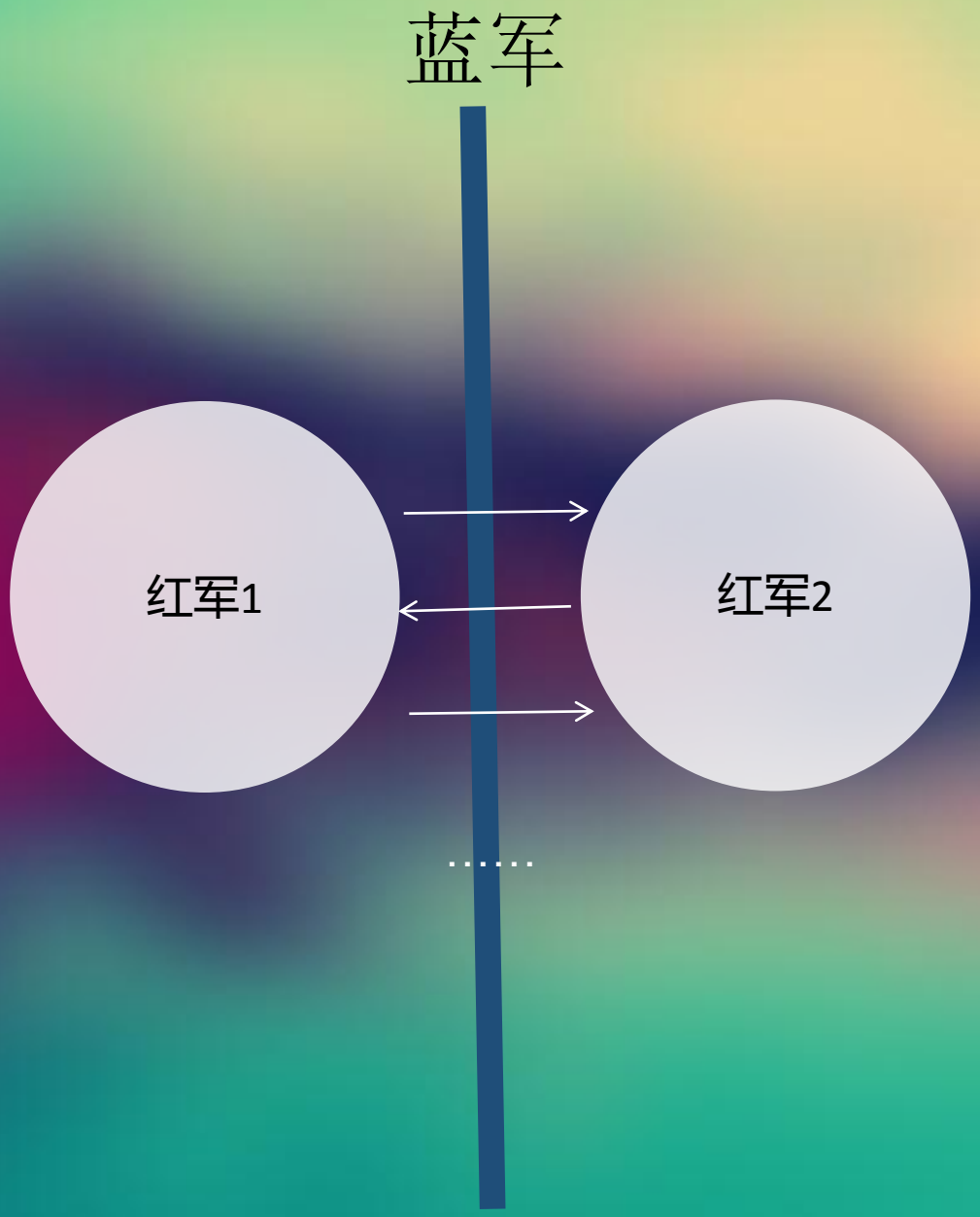


Cloud Native—架构

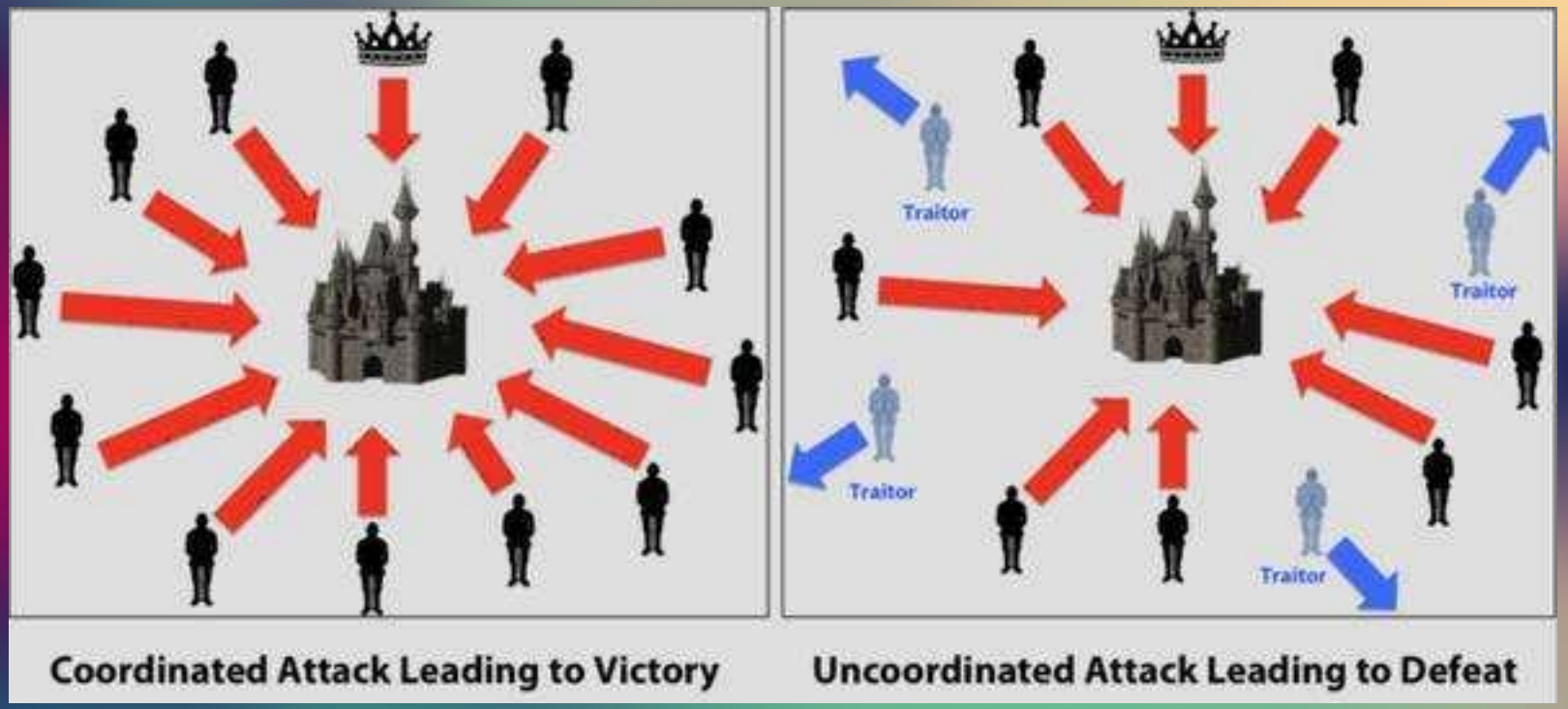



Partitioned
Replicated
A
B
C
A
[0..
100]
B
[a..n]
C
A
[101..
200]
B
[o..z]
C
Node 1
Node 2

两军问题

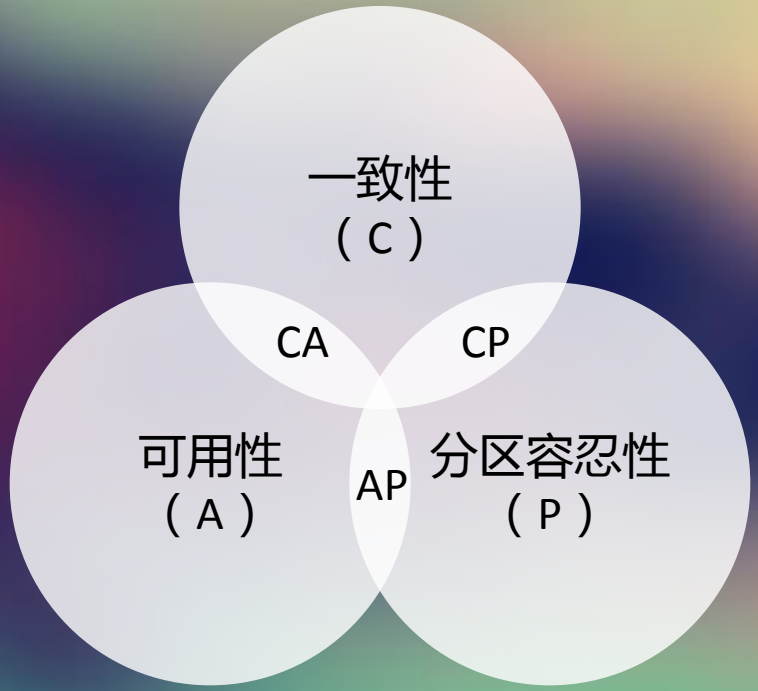


拜占庭将军问题



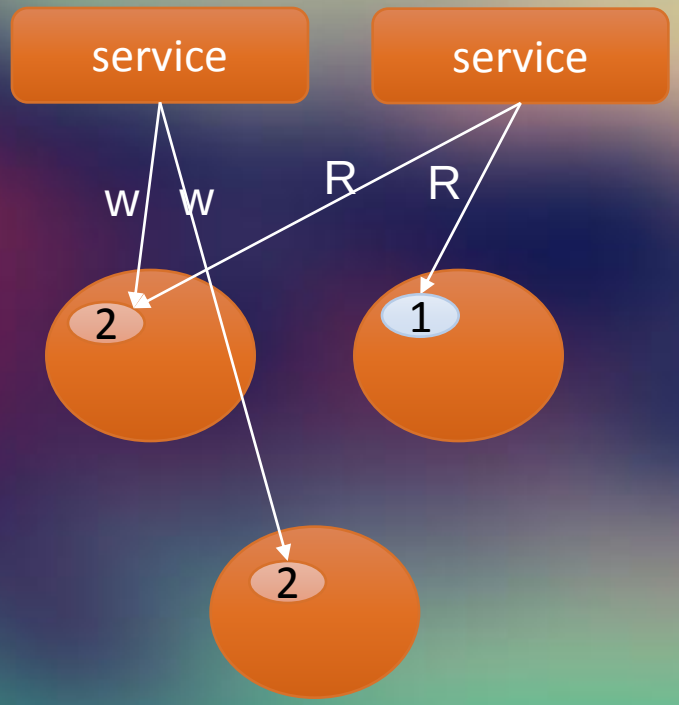
Source：www.cebnet.com.cn

CAP



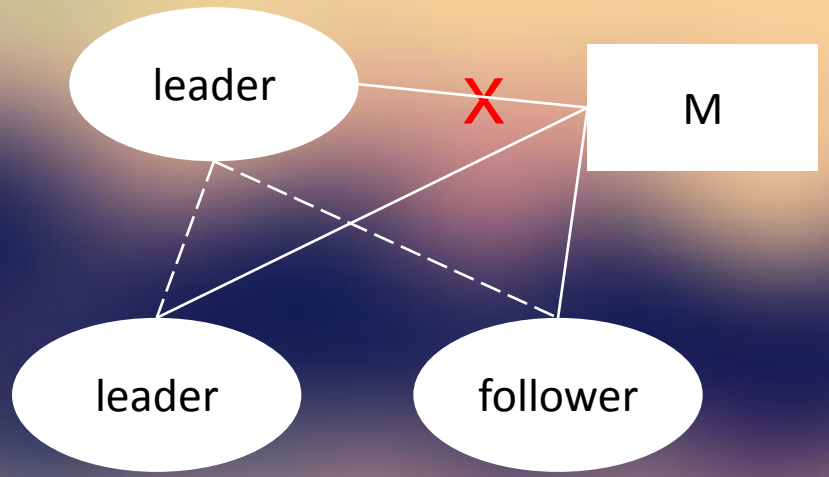
只能选择两者

NWR



W+R>N

Lease
Replicated state machine
Paxos



一致性分类

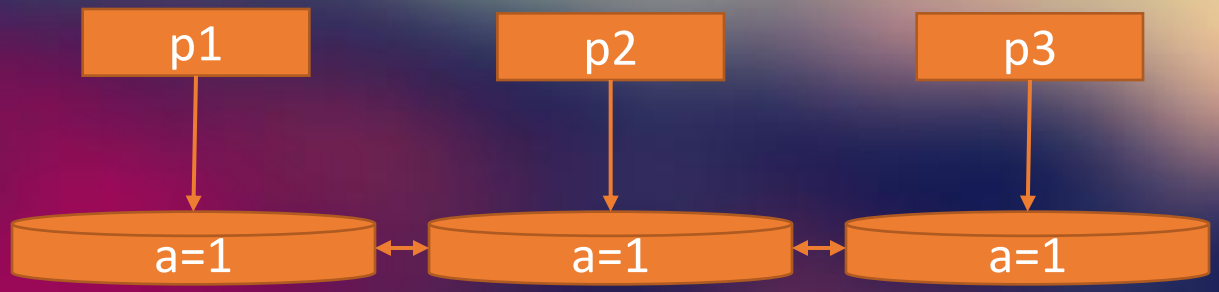
以数据为中心的一致性模型

- 严格一致性（Strict Consistency）
- 顺序一致性（Sequential Consistency）
- 因果一致性（Causal Consistency）
- FIFO一致性（FIFO Consistency）
- 弱一致性（Weak Consistency）
- 释放一致性（Release Consistency）
- 入口一致性（Entry Consistency）

以用户为中心的一致性模型

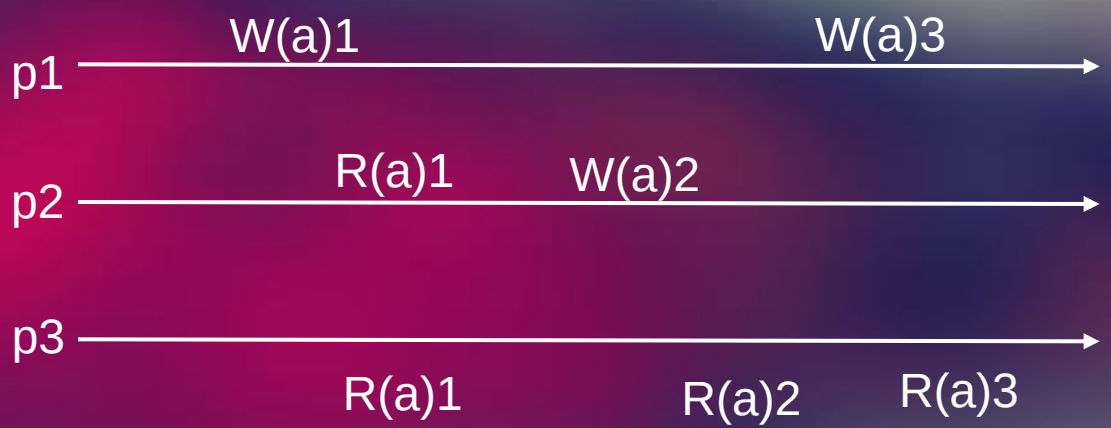
- 单调读一致性（Monotonic-read Consistency）
- 单调写一致性（Monotonic-write Consistency）
- 写后读一致性（Read-your-writes Consistency）
- 读后写一致性（Writes-follow-reads Consistency）

示例



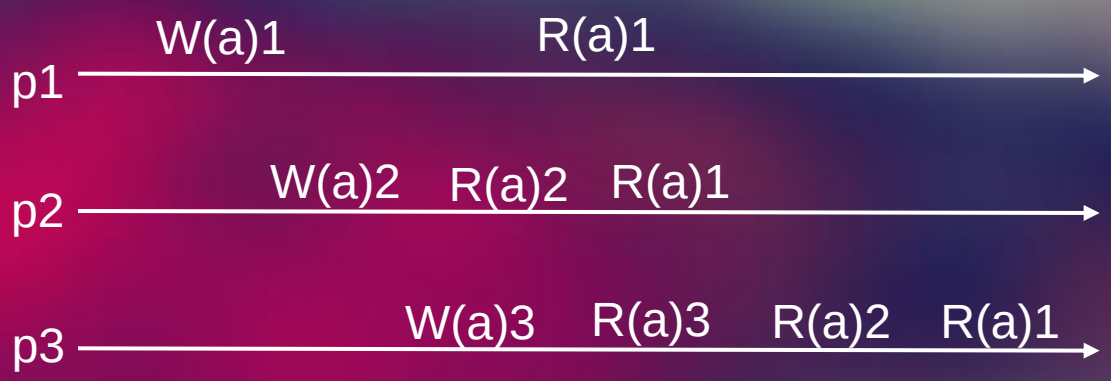
假设有多个进程，用p来表示，跟进程对应的有多份存储，a来表示变量，W(a)1来表示进程写入a=1，R(a)1表示读取a的值，结果为1。

严格一致性（Strict Consistency）



严格一致性要求任何写操作都能立刻同步到其他所有进程，任何读操作都能读取到最新的修改。

顺序一致性（Sequential Consistency）



满足顺序一致性

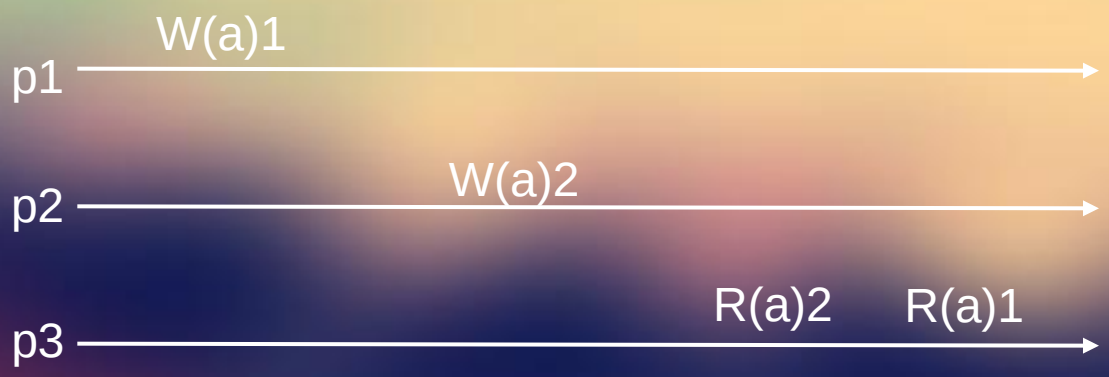
不满足顺序一致性

既然全局时钟导致严格一致性很难实现，顺序一致性放弃了全局时钟的约束，改为分布式逻辑时钟实现。

顺序一致性是指所有的进程以相同的顺序看到所有的修改。读操作未必能及时得到此前其他进程对同一数据的写更新。但是每个进程读到的该数据的不同值的顺序是一致的。

因果一致性（Causal Consistency）

不满足因果一致性



满足因果一致性

因果一致性是一种弱化的顺序一致性。

所有进程必须以相同的顺序看到具有潜在因果关系的写操作。不同进程可以以不同的顺序看到并发的写操作。

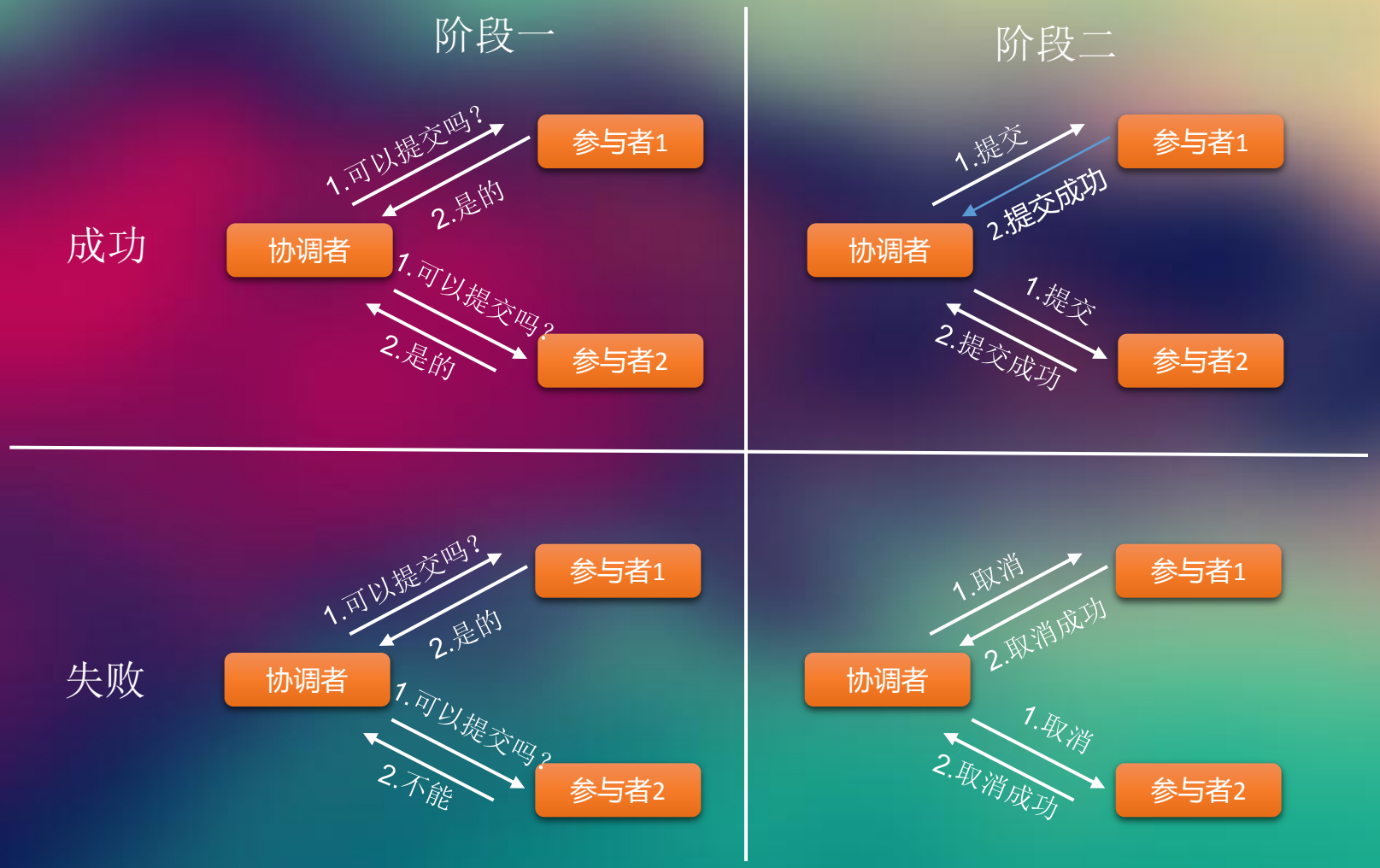
业界常用一致性分类

弱一致性 Weak
最终一致性 Eventually
强一致性 Strong

2

如何实现强一致

两阶段提交（2PC）



① 第一阶段锁定数据之后，如果协调者挂掉了，将没有角色去指挥参与者是否应该提交，这个数据会一直被锁定。也就是说如何保证协调者的可用性？

② 如果第二阶段参与者1提交成功，参与者2提交失败了（挂掉了，没返回ACK），此时协调者不知道该如何处理，因为参与者1已经提交成功，外部可以访问了。

③ 可扩展性极差。

三阶段提交（3PC）

实现的难度更大。

性能也更低。

可扩展性极差。

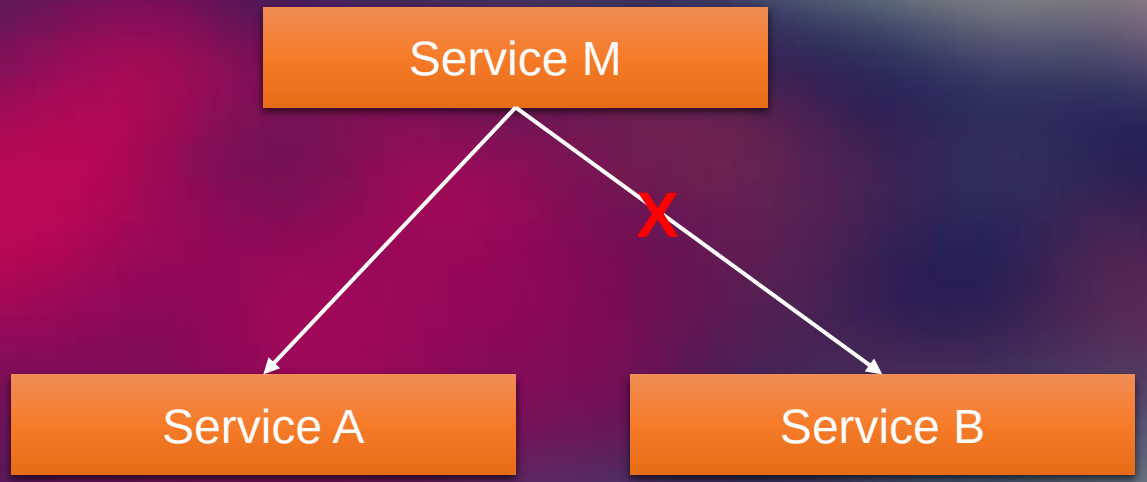
银行转账、12306是强一致吗？

一人生病，全家吃药。

3

如何实现最终一致

调用失败怎么办？



重试！！！

超时时间。

重试的次数。

重试的间隔时间。

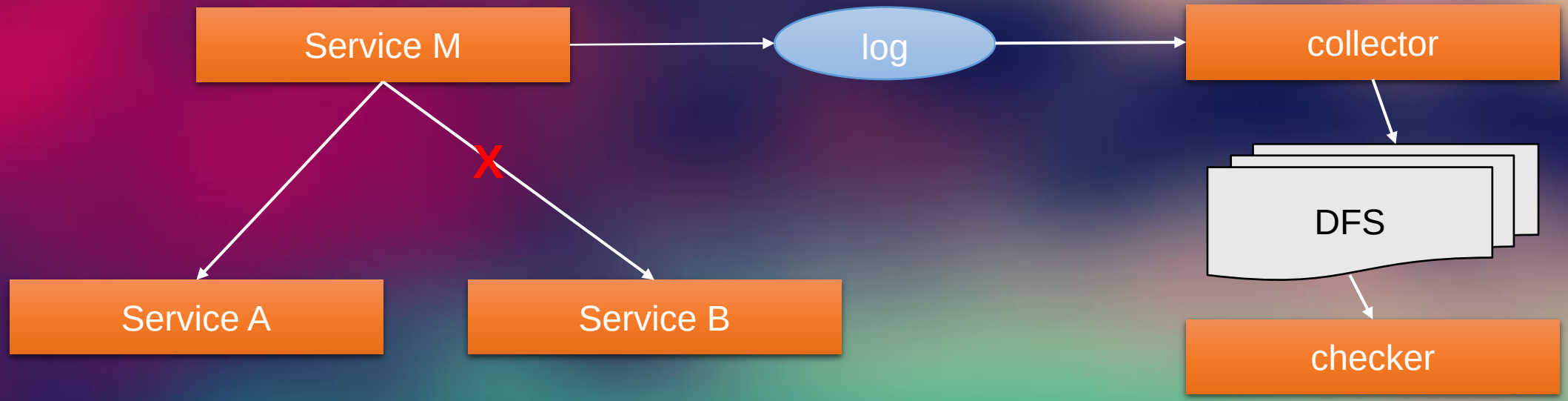
重试间隔时间的衰减度。

重试前失败怎么办？？？

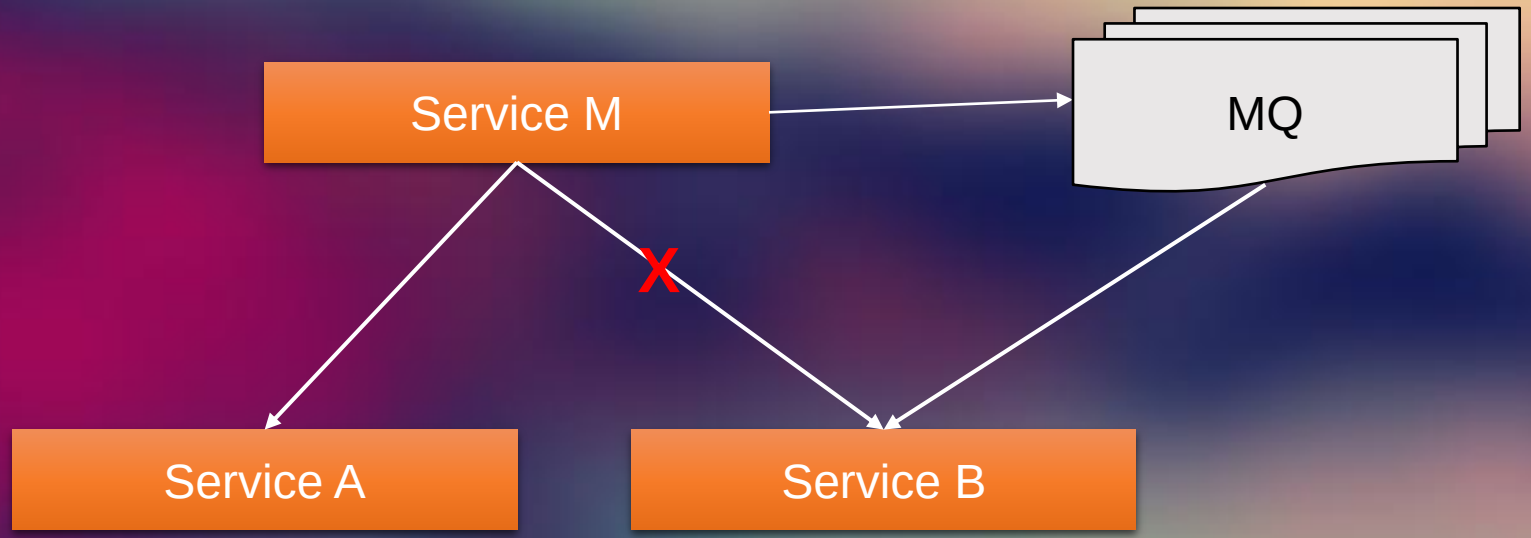
状态？？？状态！！！

宠物 vs 牲畜

改进方案



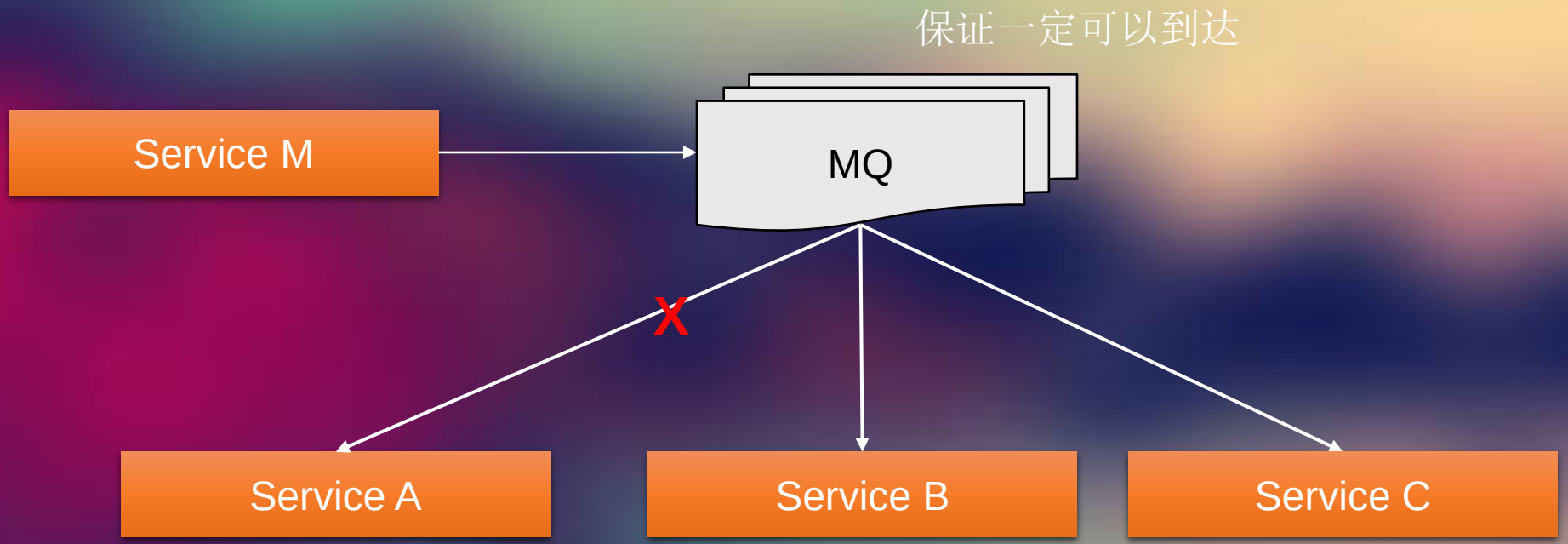
可靠事件



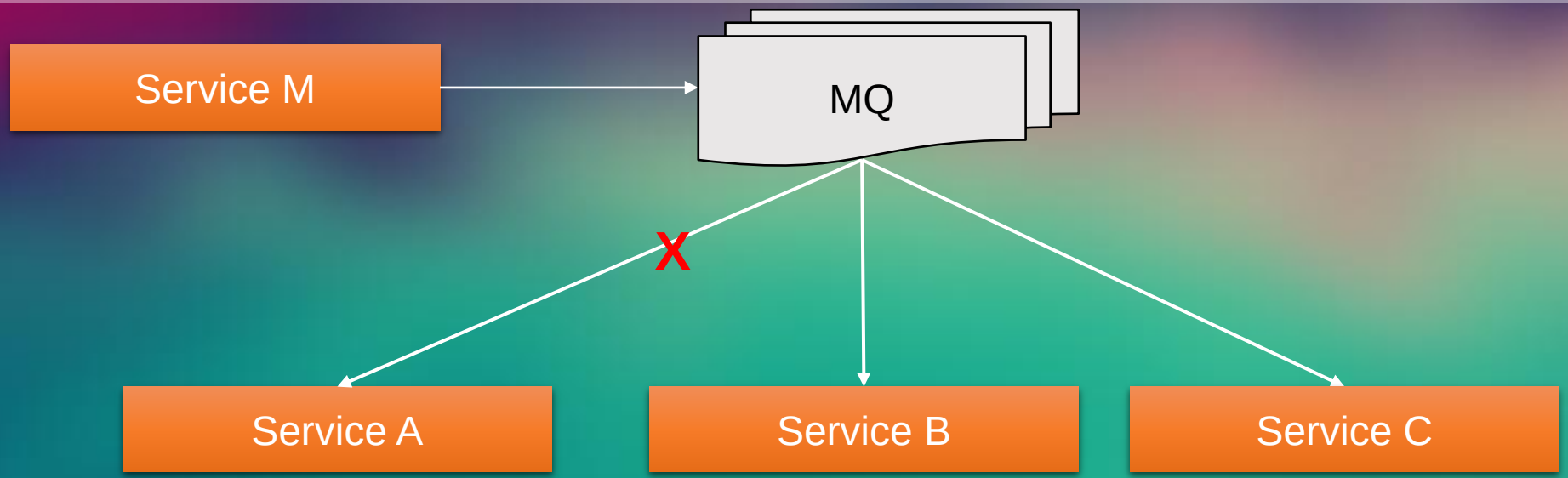
状态？？？状态！！！

要么全部成功，要么全部失败。

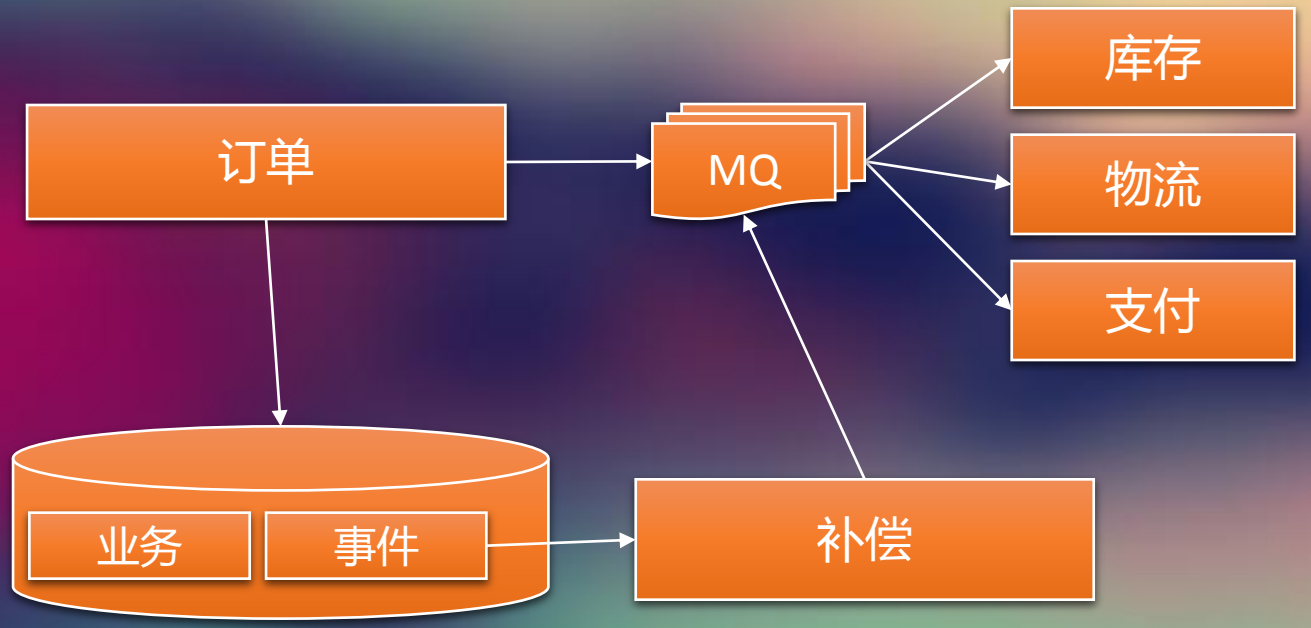
可靠事件



怎么保证写后读一致性？



案例

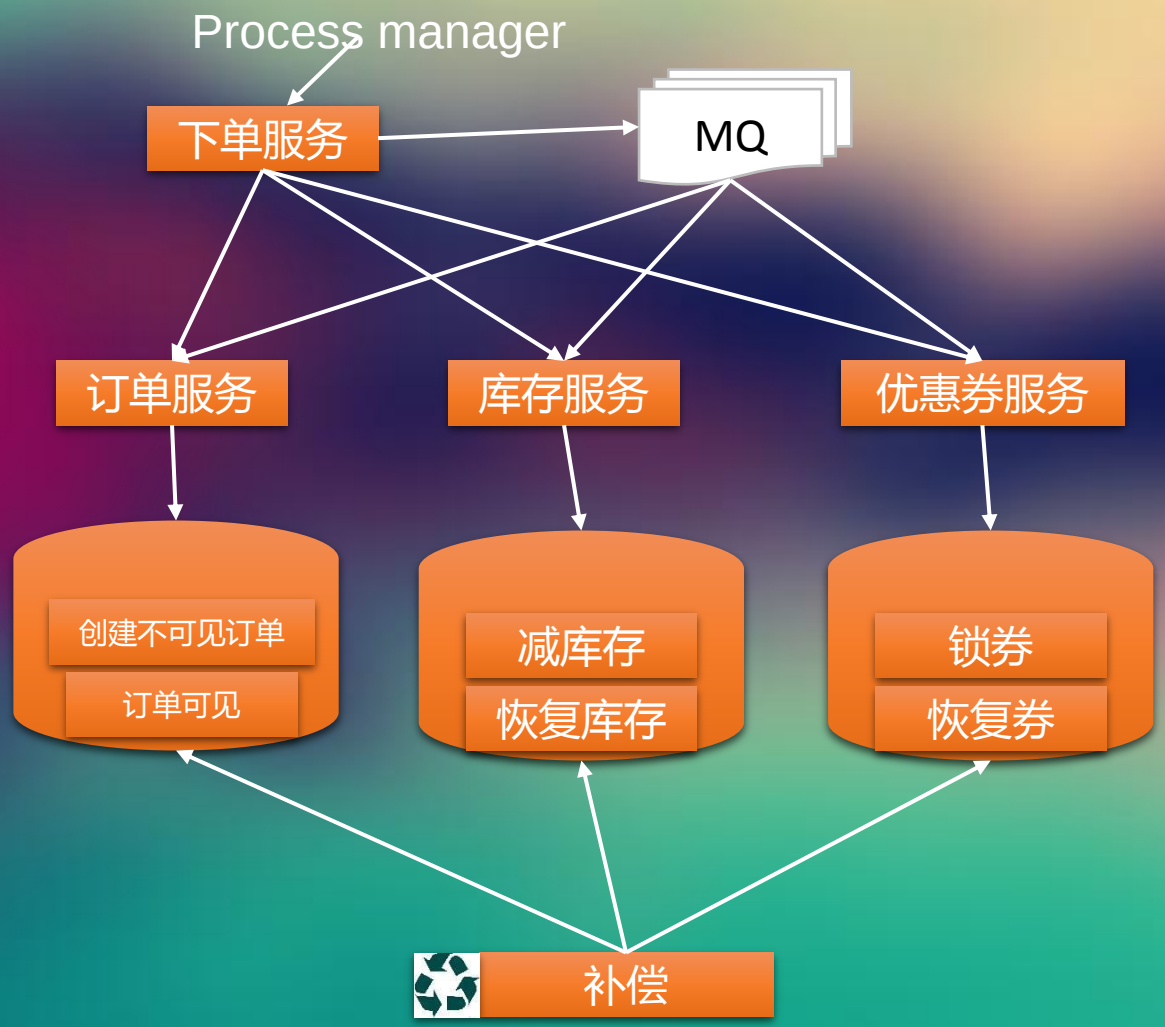


长事务的一致性

Saga事务模型

核心思想：拆分为多个本地事务

Saga事务模型


Process manager
下单服务
MQ
订单服务
库存服务
优惠券服务
创建不可见订单
订单可见
减库存
恢复库存
锁券
恢复券
补偿

如果失败怎么办？

① 定时任务检查

② 失败发消息

③ 正向调用时写回退日志

往往数据库是瓶颈，如何平衡压力？

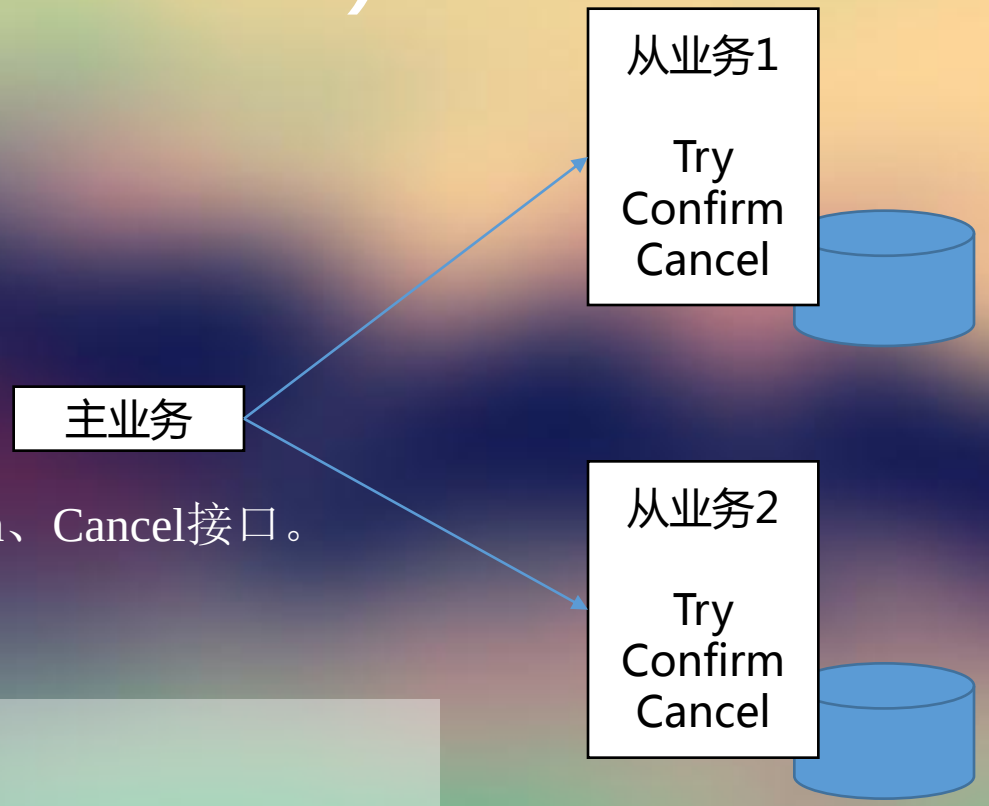
TCC（Try-Confirm-Cancel）

TCC的优势：

- 在业务层处理，平衡数据库的压力。

TCC的代价：

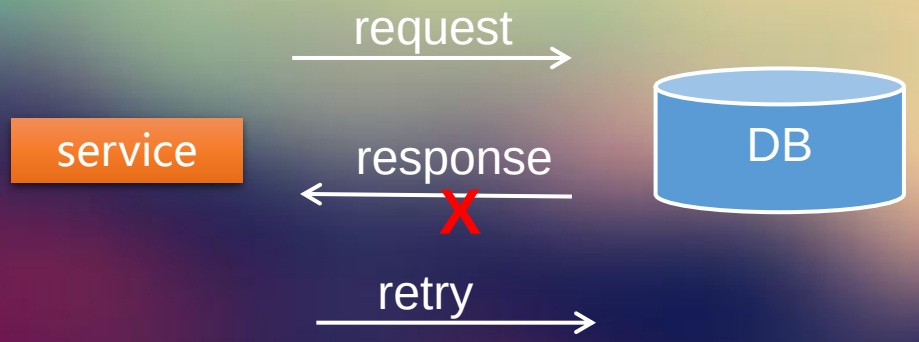
- 增加业务复杂度，需要提供相应的Try、Confirm、Cancel接口。
- 需要提供幂等性接口。



敲黑板！！！

① Try成功，协调者挂掉怎么办？

② 业务1提交成功，业务2失败怎么办？



如何保障幂等？

如何保障幂等？

把编号为5的记录的A字段设置为0，这种操作不管执行多少次都是幂等的。

把编号为5的记录的A字段增加1。这种操作显然就不是幂等的。

方案一、数据库加锁

方案二、分布式锁

方案三、唯一约束

方案四、增加流水表

商品ID	库存	状态	……
1101	99	1	……

商品ID	库存	状态	……
1101	99	1	……

理论

强一致性

2PC/3PC

最终一致性

重试

可靠事件

Saga

TCC

幂等

Thank you

赶上浪潮，如此，便不枉此生