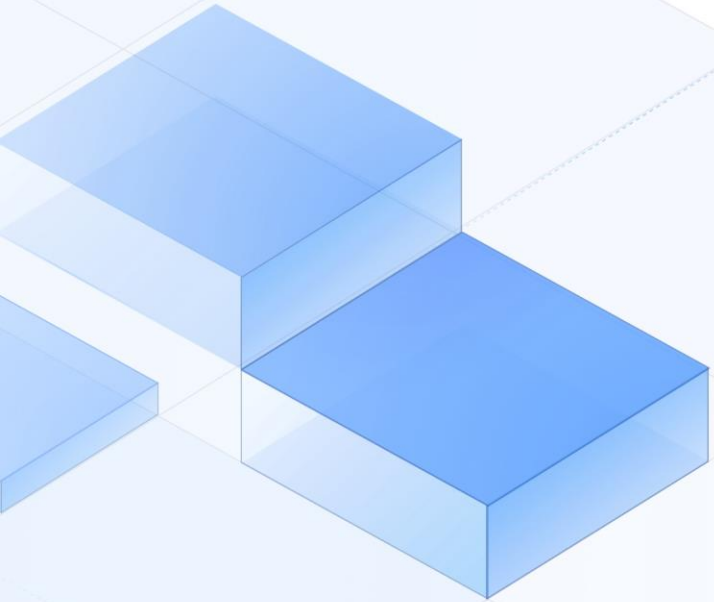
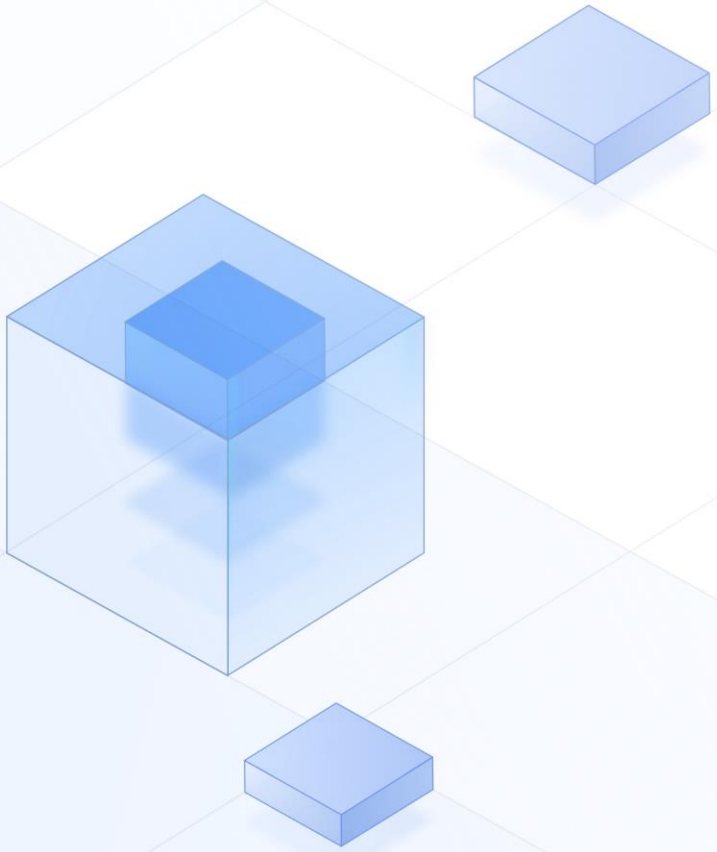
OCEANBASE

OceanBase分布式事务高级技术

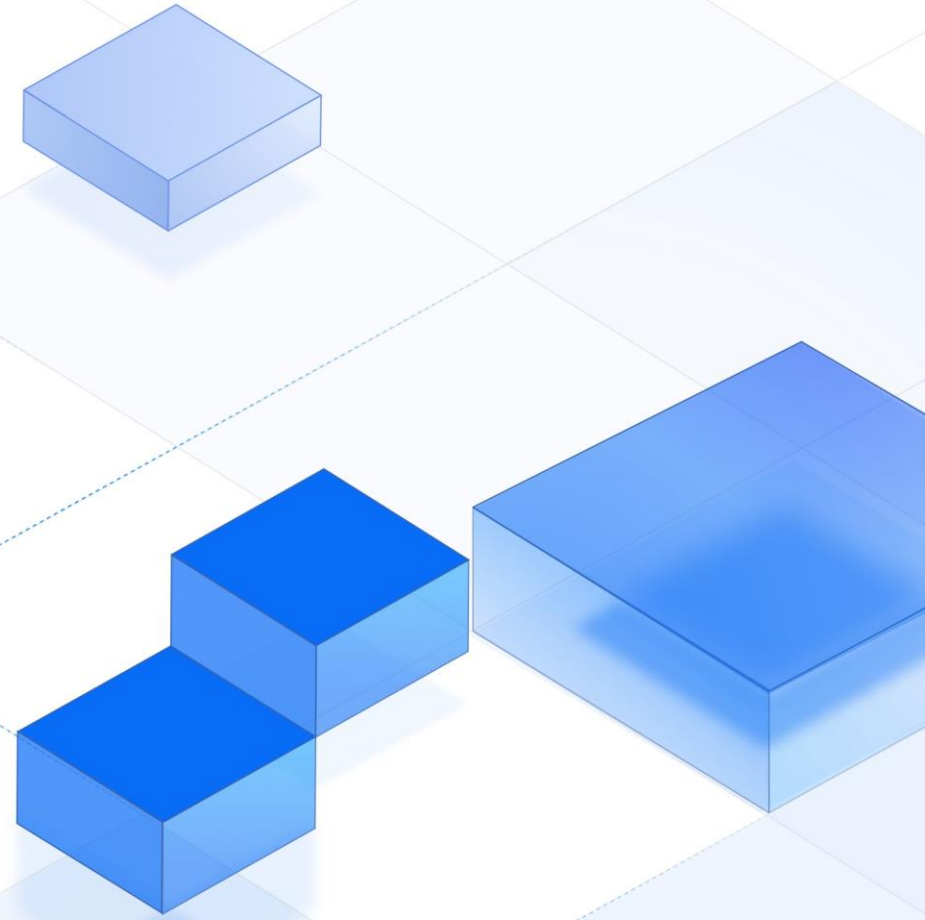
目录



目录

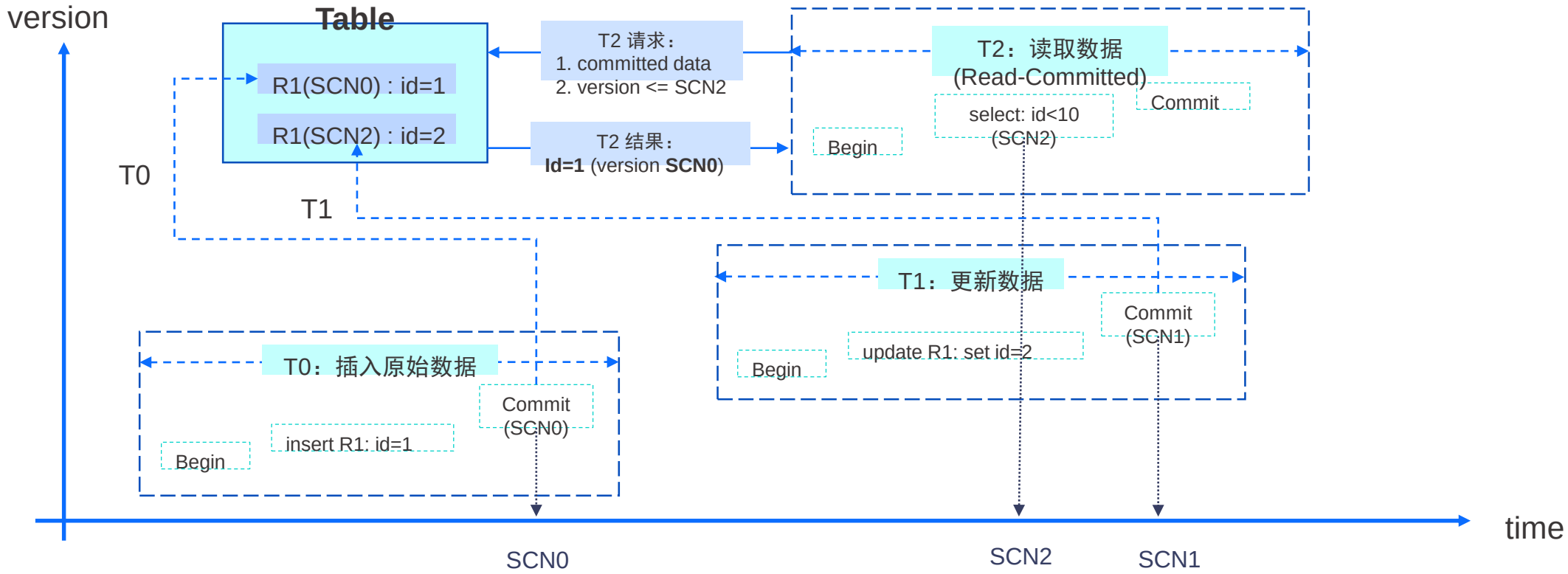


5.1 全局快照及分布式一致性读



5.1 传统数据库的实现原理

“快照隔离级别（Snapshot Isolation）”和“多版本并发控制（Multi-VersionConcurrency Control，简称MVCC）”： 两种技术的大致含义是：为数据库中的数据维护多个版本号（即多个快照），当数据被修改的时候，可以利用不同的版本号区分出正在被修改的内容和修改之前的内容，以此实现对同一份数据的多个版本做并发访问，避免了经典实现中“锁”机制引发的读写冲突问题



5.1 快照隔离级别（snapshot isolation）

√ 代表可能出现，× 代表不会出现。

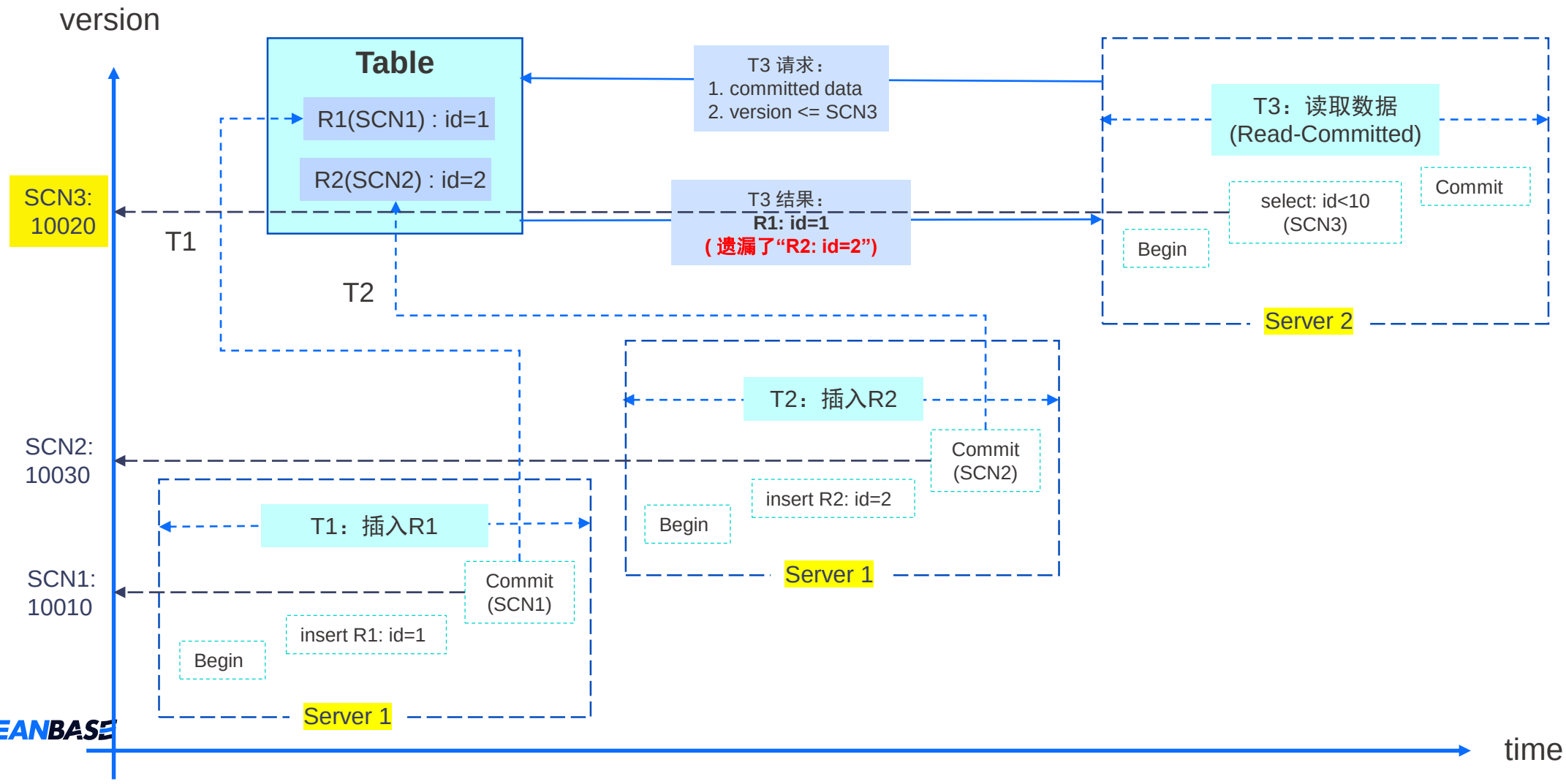
隔离级别	脏读	不可重复读	幻读
Read uncommitted	√	√	√
Read committed	×	√	√
Repeatable read	×	×	√
Serializable	×	×	×

时间顺序	事务A	事务B
1	开始事务	
2	第一次查询，小明的年龄为20岁	
3		开始事务
4	其他操作	
5		更改小明的年龄为30岁
6		提交事务
7	第二次查询，小明的年龄为30岁	
备注	按照正确逻辑，事务A前后两次读取到的数据应该一致	

时间顺序	事务A	事务B
1	开始事务	
2	第一次查询，数据总量为100条	
3		开始事务
4	其他操作	
5		新增100条数据
6		提交事务
7	第二次查询，数据总量为200条	
备注	按照正确逻辑，事务A前后两次读取到的数据总量应该一致	

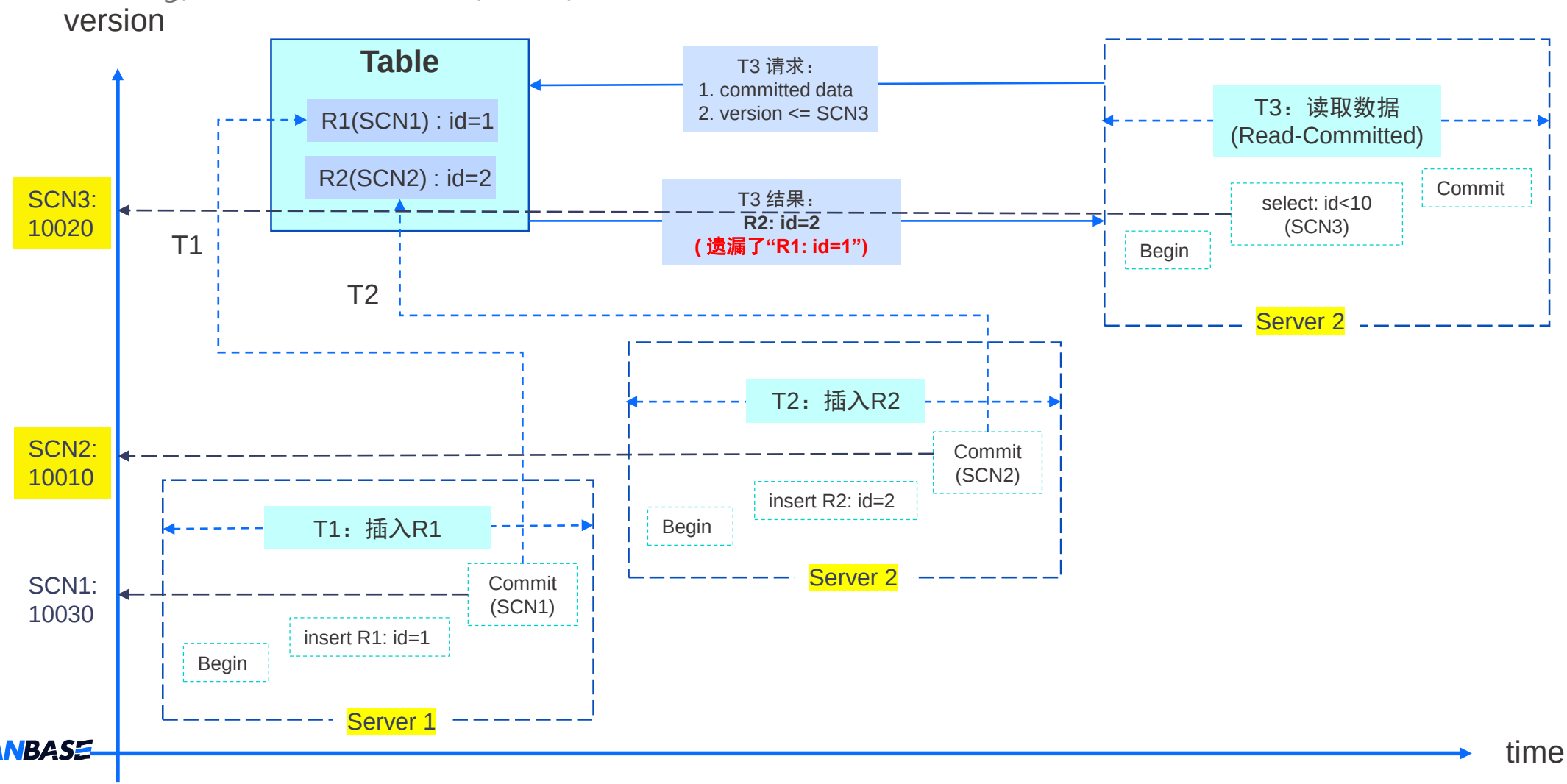
5.1 分布式数据库面临的挑战

和传统的数据库的单点全共享（即Shared-Everything）架构不同，OceanBase是一个原生的分布式架构，采用了多点无共享（即Shared-Nothing）的架构，在实现全局（跨机器）一致的快照隔离级别和多版本并发控制时会面临分布式架构所带来的技术挑战



5.1 分布式数据库面临的挑战

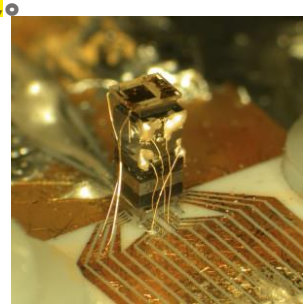
和传统的数据库的单点全共享（即Shared-Everything）架构不同，OceanBase是一个原生的分布式架构，采用了多点无共享（即Shared-Nothing）的架构，在实现全局（跨机器）一致的快照隔离级别和多版本并发控制时会面临分布式架构所带来的技术挑战



5.1 业界常用解决方案

分布式数据库应如何在全局（跨机器）范围内保证外部一致性，进而实现全局一致的快照隔离级别和多版本并发控制呢？大体来说，业界有两种实现方式：

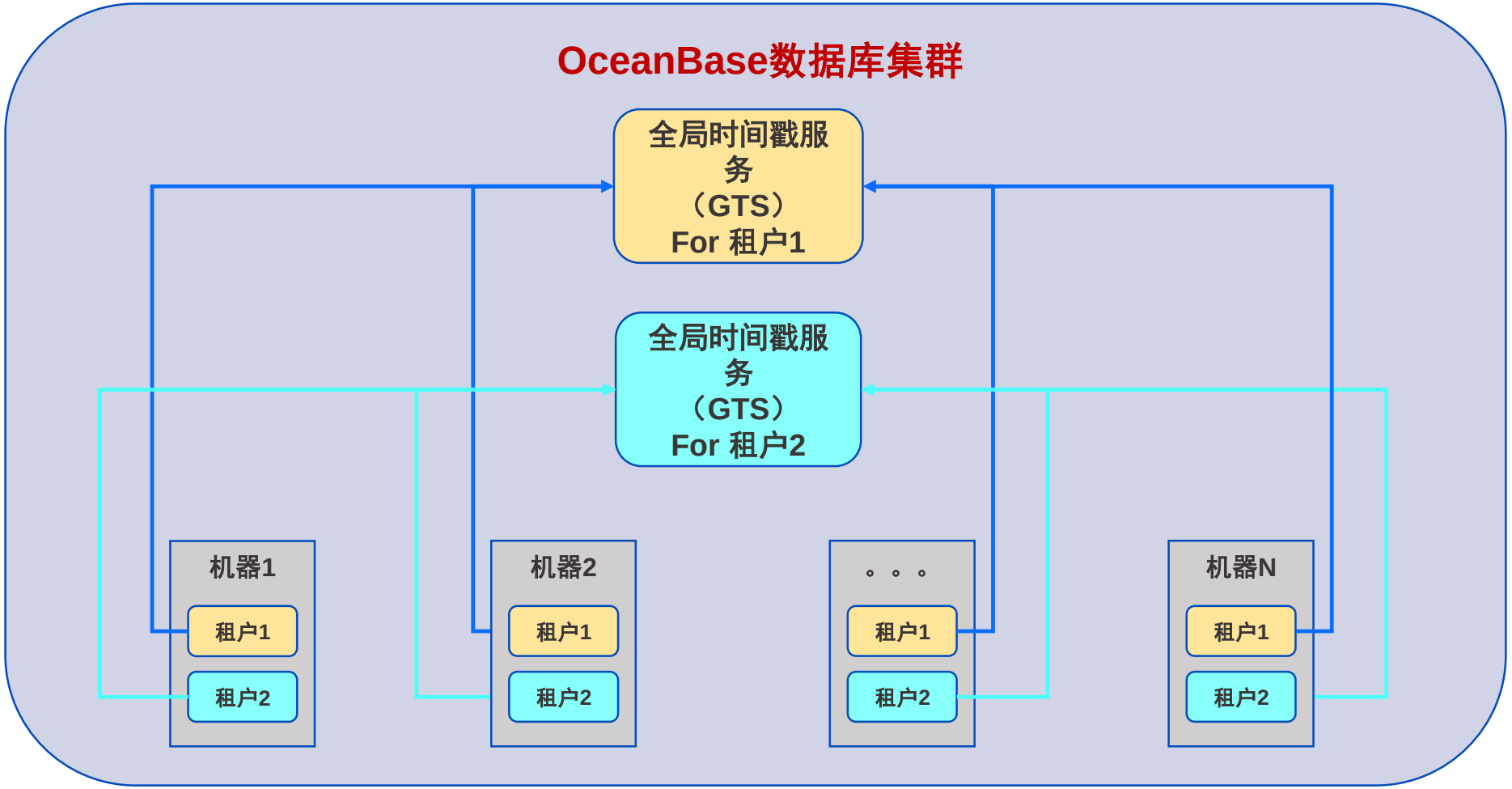
1. 利用特殊的硬件设备，如GPS和原子钟（Atomic Clock），使多台机器间的系统时钟保持高度一致，误差小到应用完全无法感知的程度。



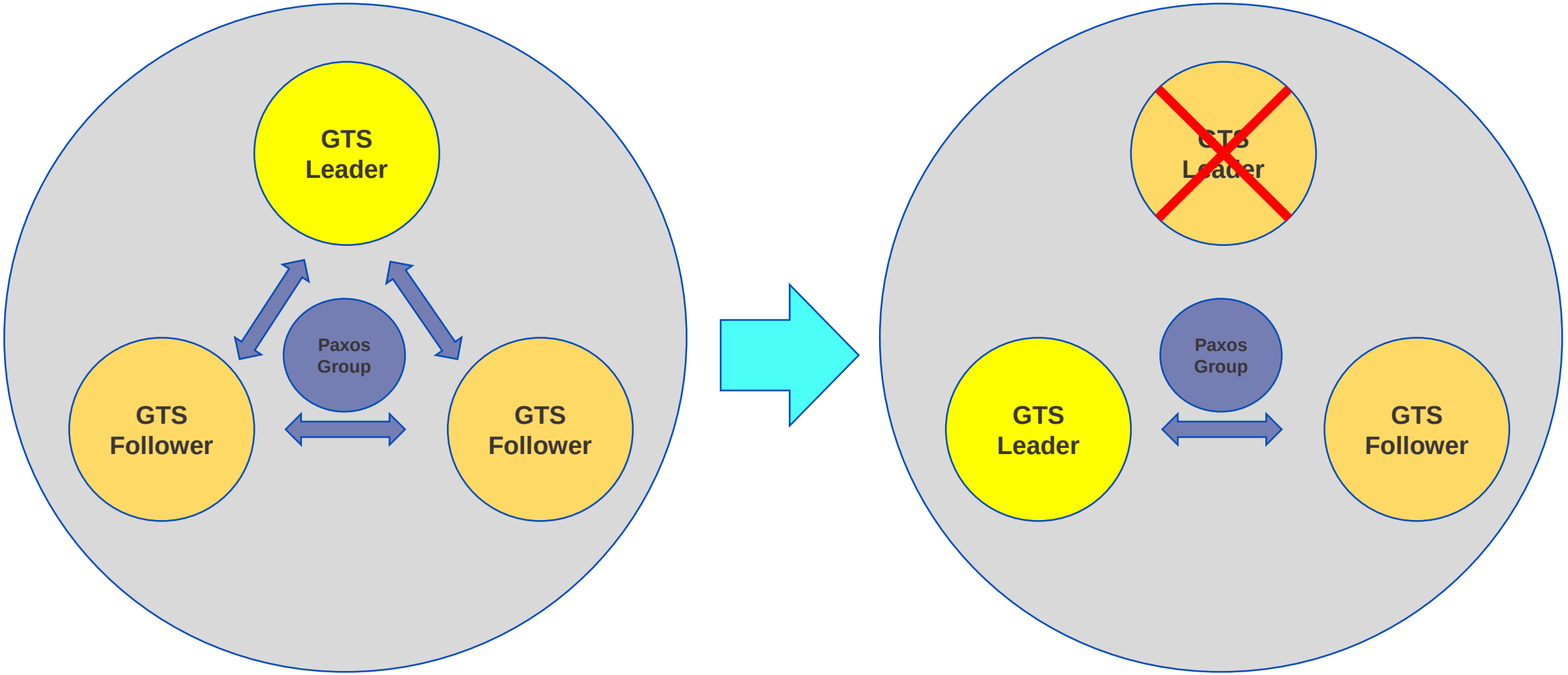
2. 版本号不再依赖各个机器自己的本地系统时钟，所有的数据库事务通过集中式的服务获取全局一致的版本号，由这个服务来保证版本号的单调向前。

5.1 OceanBase全局一致性快照技术

OceanBase数据库是利用一个集中式服务来提供全局一致的版本号。事务在修改数据或者查询数据的时候，无论请求源自哪台物理机器，都会从这个集中式的服务处获取版本号，OceanBase则保证所有的版本号单调向前并且和真实世界的时间顺序保持一致。



5.1 OceanBase全局一致性快照如何处理异常



5.1分布式事务跨机执行时，OceanBase通过多种机制保证ACID

A

原子性Atomicity

原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生

➢ 依赖两阶段提交协议保证分布式事务的原子性；

C

一致性Consistency

事务前后数据的完整性必须保持一致

➢ 保证主键唯一等一致性约束；

➢ 全局快照 - 单租户GTS服务，1秒钟内能够响应获取全局时间戳的调用次数超过200万次；

I

隔离性Isolation

多个用户并发访问数据库时，数据库为每个用户开启的事务，不能被其他事务的操作所干扰

➢ 采用MVCC进行并发控制，实现read-committed的隔离级别；

➢ 所有修改的行加互斥锁，实现写 - 写互斥；

➢ 读操作读取特定快照版本的数据，读写互不阻塞；

D

持久性Durability

一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响

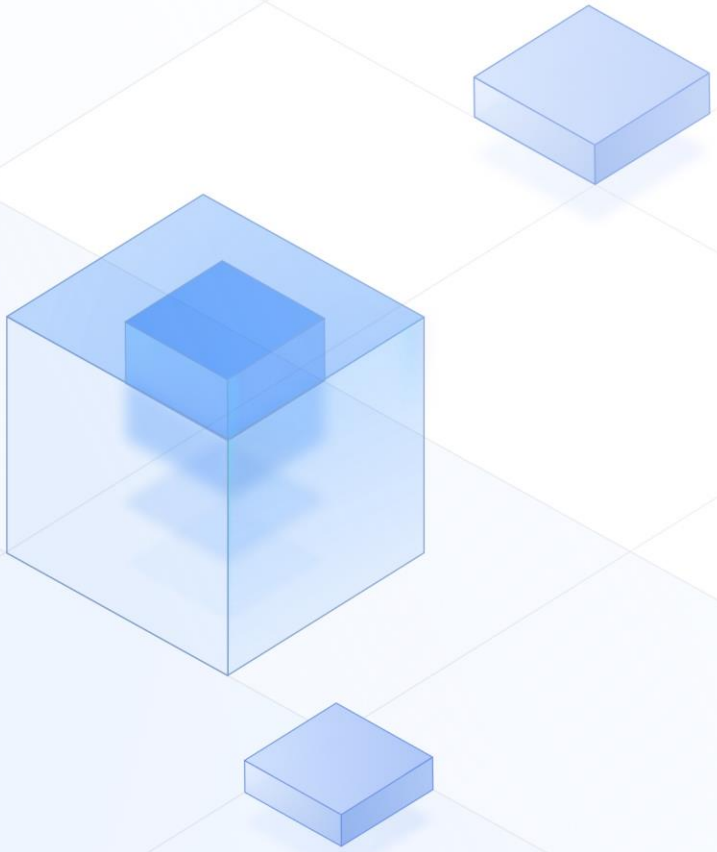
➢ Redo-Log使用Paxos协议做多副本同步

5.1 小结

使用“全局一致性快照”，OceanBase数据库便具备了在全局（跨机器）范围内实现“快照隔离级别”和“多版本并发控制”的能力，可以在全局范围内保证“外部一致性”，并在此基础之上实现众多涉及全局数据一致性的功能，比如全局一致性读、全局索引等。

和传统单点数据库相比，OceanBase在保留分布式架构优势的同时，在全局数据一致性上也没有降级，应用开发者就可以像使用单点数据库一样使用OceanBase，不必担心机器之间的底层数据一致性问题。

目录



5.2 分布式系统中可能会出现的问题

场景 1

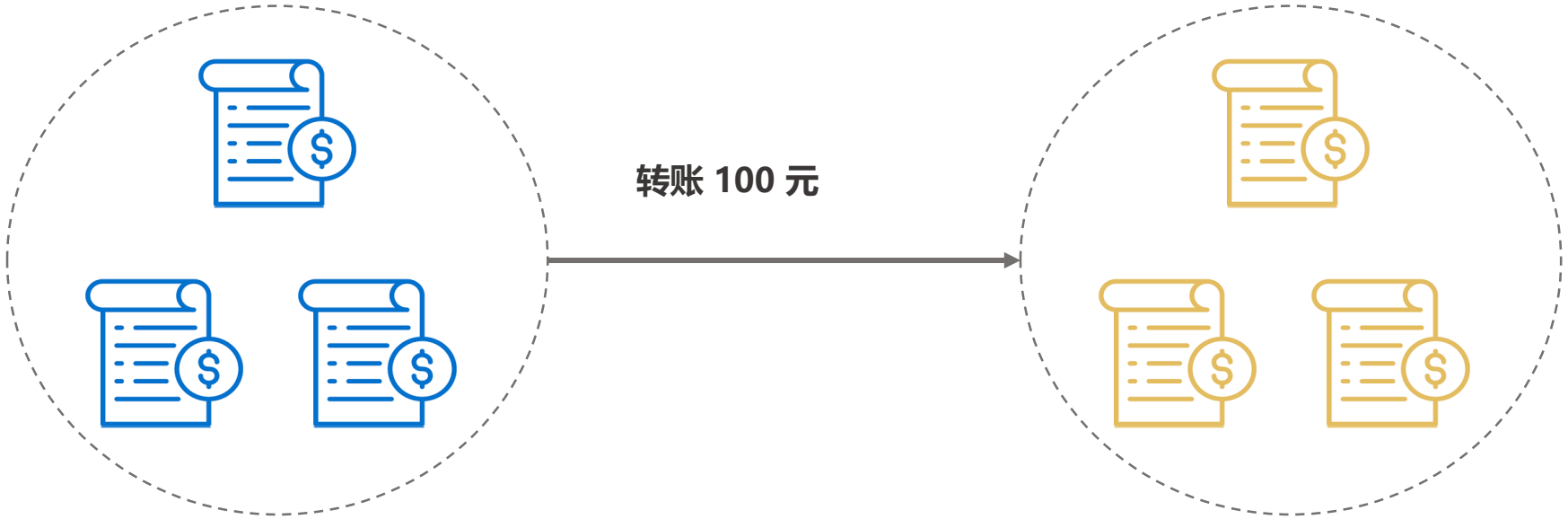
- 交易发生时，负责A账户扣100块钱的A机器死机，没有扣成功，而负责给账户B加100块钱的B机器工作正常，加上了100
- 负责给A账户扣100块钱的A机器正常，已经扣掉100，而负责给账户B加100块钱的B机器死机，100块没加上，那么用户就会损失100

场景 2

- A账户要扣掉100块，但是它的余额只有90块，或者已经达到了今天的转账限额，这种情况下，如果贸然给B账户加了100块，A账户却不够扣，麻烦了
- 如果B账户状态有异常（例如冻结），不能加100块，同样也麻烦了

5.2 分布式系统中场景1的解决方案

Paxos 多副本： 账户数据一定同时存在三台机器上。转账时，至少两台机器执行完毕才算转账完成，意味着三台机器里有一台坏掉，并不影响转账的执行



5.2 分布式系统中场景2的解决方案

引入裁判员：

1）裁判员问A账户：你的三台机器都没问题吧？A账户说：没问题。 你的账户允许扣100吗？A账户说：允许。

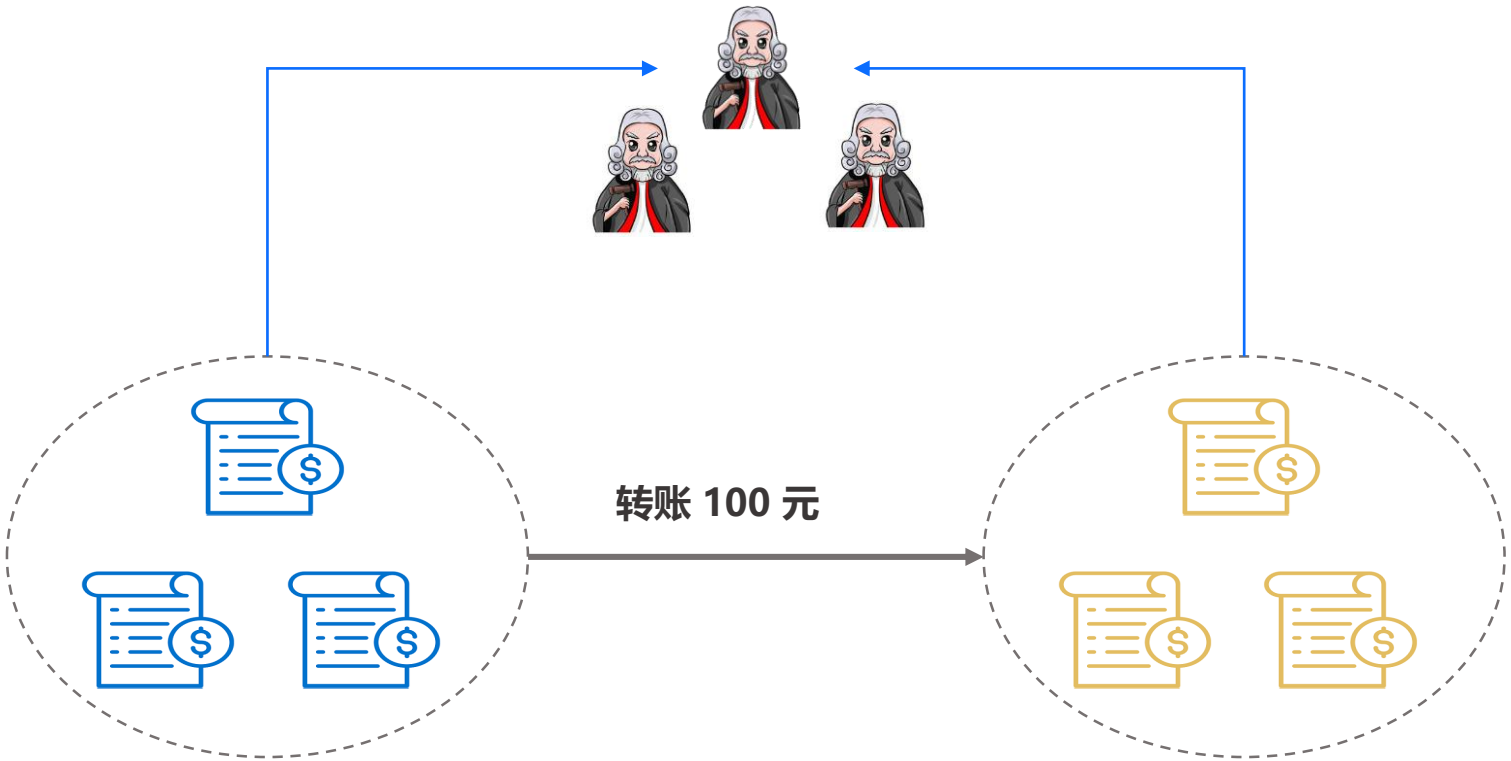
3）裁判员问B账户：你的三台机器都没问题吧？B账户说：没问题。你的账户状态能接受加100吗？B说：允许。

4）这时，裁判员吹哨，A、B账户同时冻结。

5）A扣100，B加100，双方向裁判汇报“成功”。

6）裁判员再吹哨，A、B账户同时解冻。

当然，裁判员也要是多副本才好。

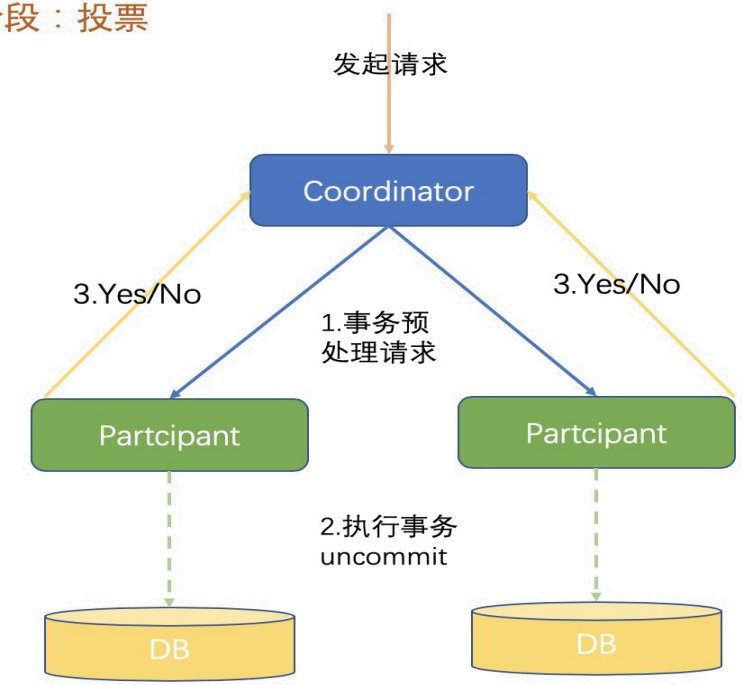


5.2 两阶段协议

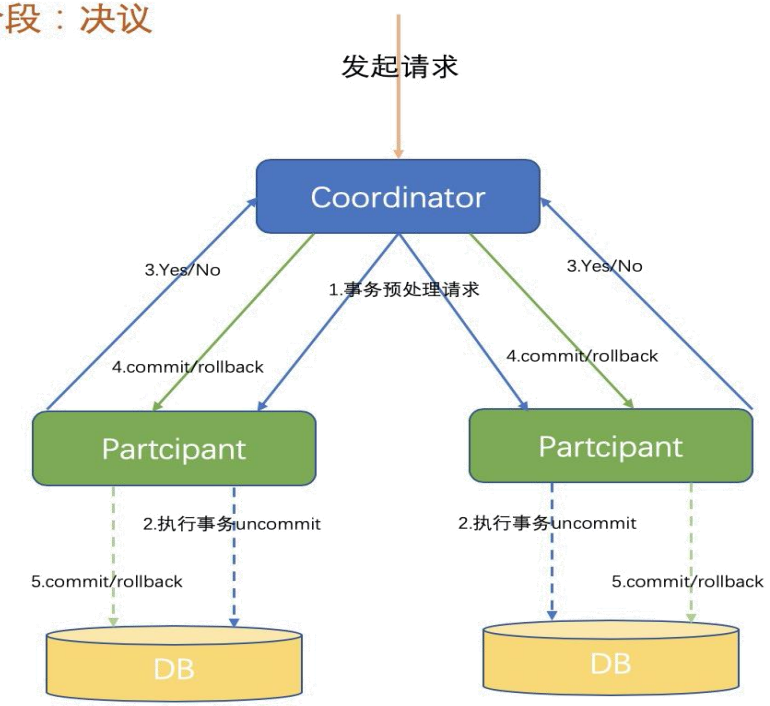
2PC是一个非常经典的强一致、中心化的原子提交协议。中心化指的是协调者（Coordinator），强一致性指的是需要所有参与者（partcipant）均要执行成功才算成功，否则回滚。

1. 第一阶段：协调者（coordinator)发起提议通知所有的参与者（partcipant），参与者收到提议后，本地尝试执行事务，但并不commit，之后给协调者反馈，反馈可以是yes或者no。
2. 第二阶段：协调者收到参与者的反馈后，决定commit或者rollback，参与者全部同意则commit，如果有一个参与者不同意则rollback。

第一阶段：投票



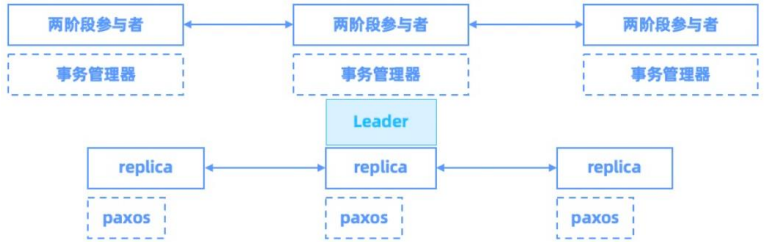
第二阶段：决议



5.2 OceanBase两阶段提交协议

标准两阶段提交协议

- 优点：状态简单，只依靠协调者状态即可确认和推进整个事务状态
- 缺点：协调者写日志，commit延时高

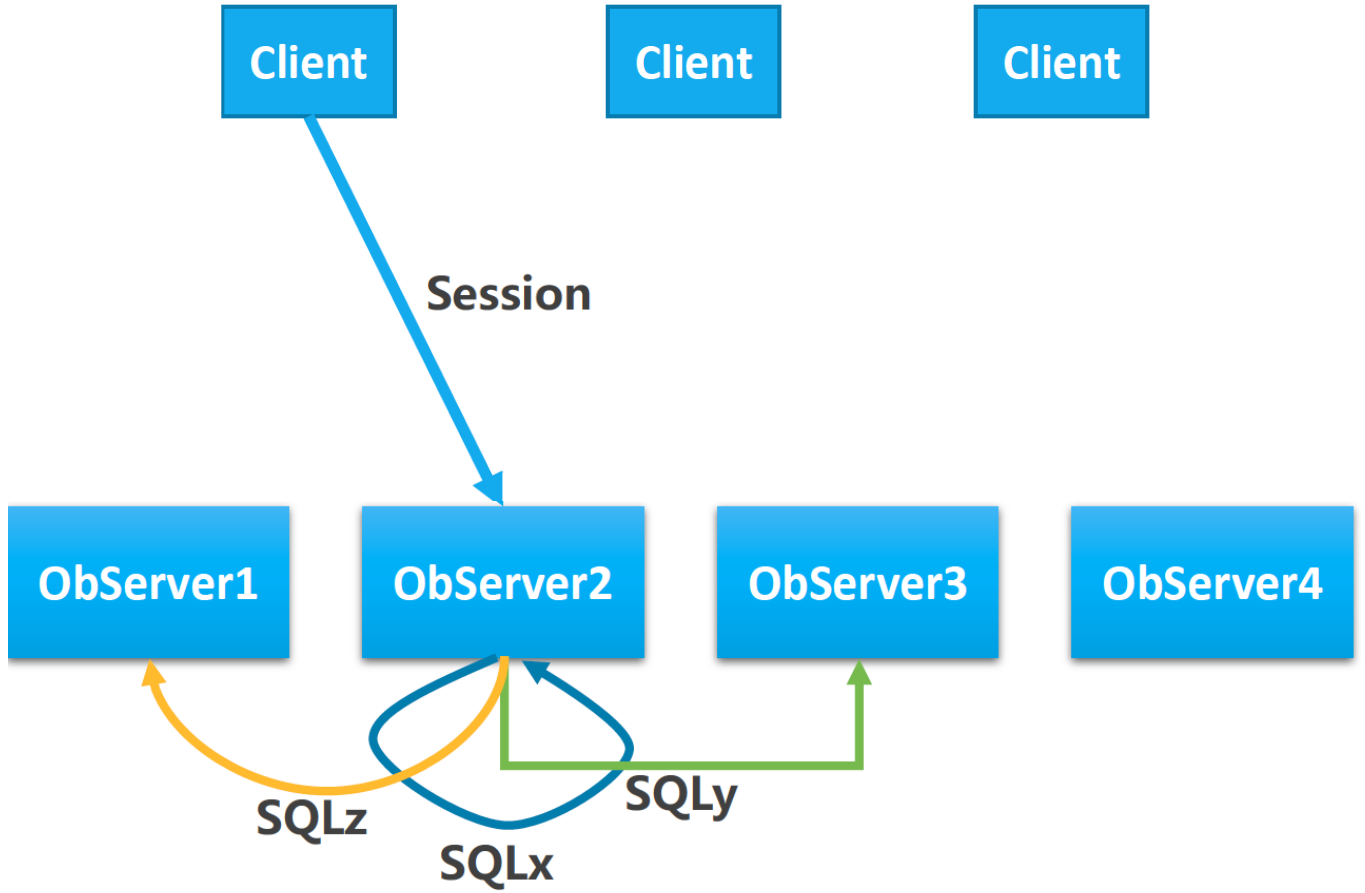


OceanBase两阶段提交协议

1. 协调者不写日志，变成了一个无持久化状态的状态机
2. 事务的状态由参与者的持久化状态决定
3. 所有参与者都prepare成功即认为事务进入提交状态，立即返回客户端commit
4. 每个参与者都需要持久化参与者列表，方便异常恢复时构建协调者状态机，推进事务状态
5. 参与者增加clear阶段，标记事务状态机是否终止

5.2 OB两阶段提交

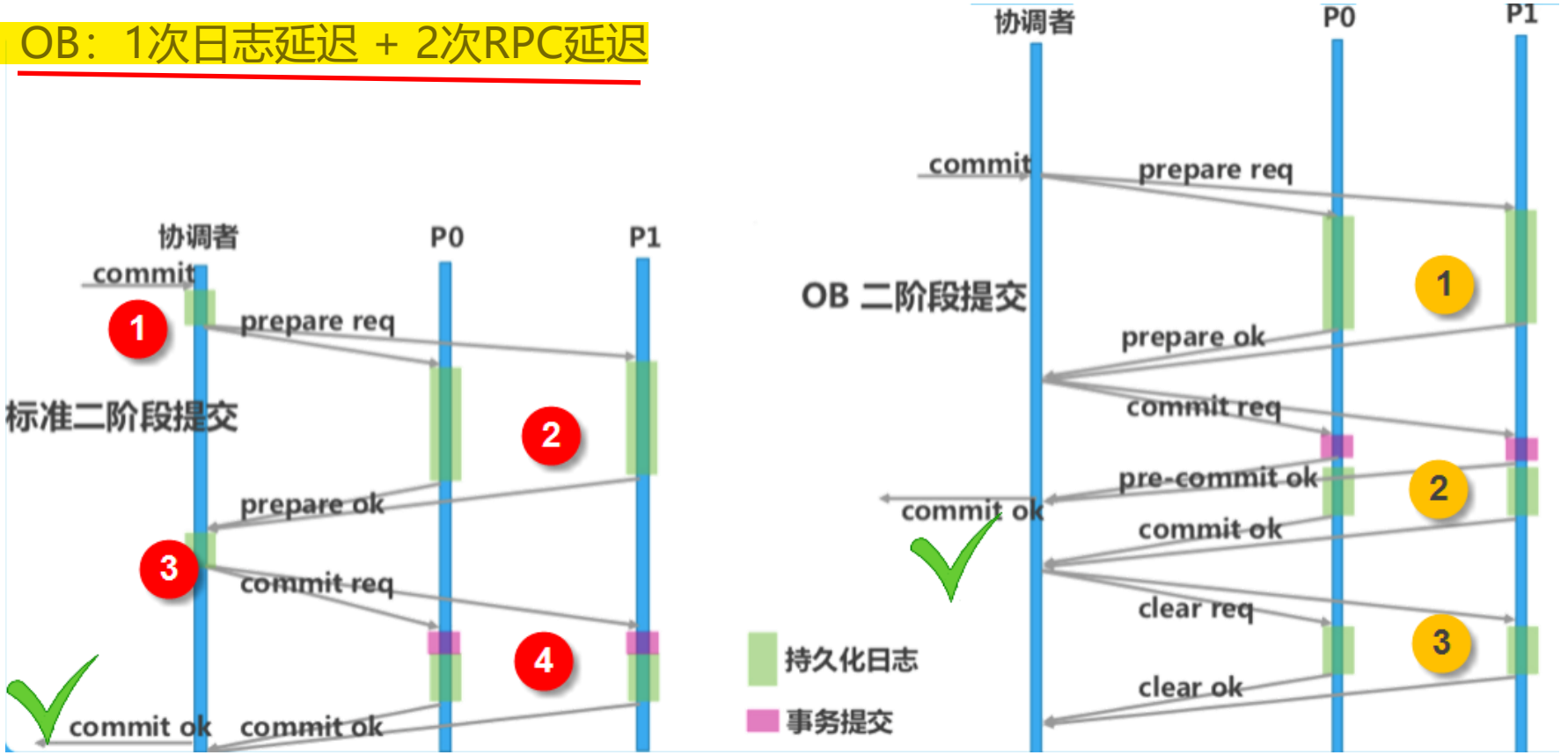
1. 业务不感知是否分布式事务，如果只有一个参与者使用一阶段提交；否则自动使用两阶段提交
2. 参与者的实体是 Partition（Px，Py，Pz）
3. 指定第一个参与者 Px 作为协调者，发送 end_trans 消息给 Px 并告知参与者列表：Px，Py，Pz



5.2 OB两阶段提交延迟分析

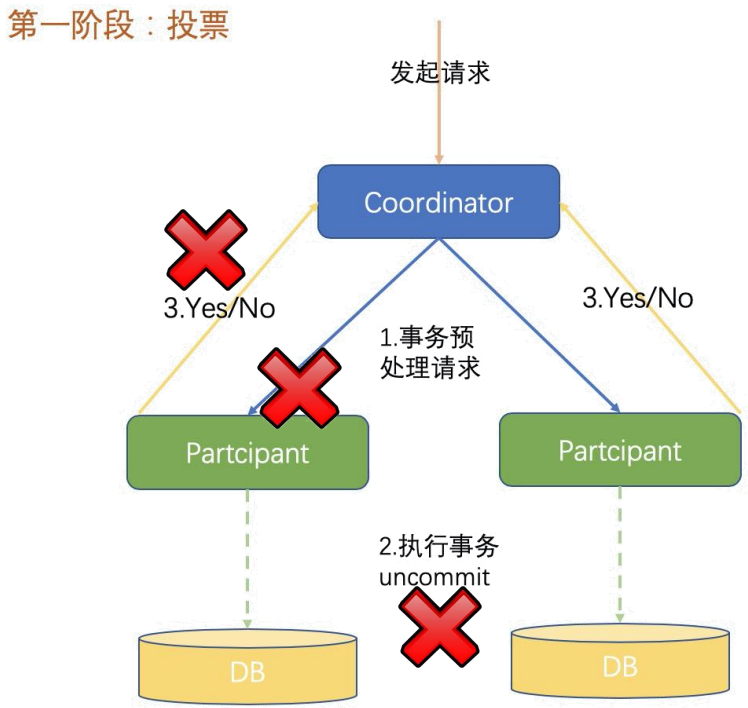
用户感知的commit延迟

- 标准：4次日志延迟 + 2次RPC延迟
- OB：1次日志延迟 + 2次RPC延迟

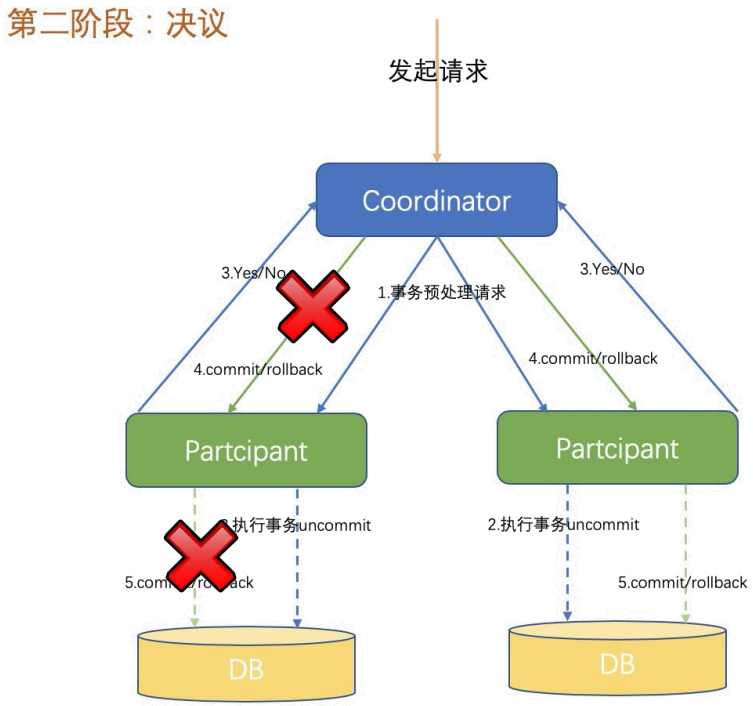


5.2 分布式事务的高可用

- 如果事务在prepare状态落盘之前发生宕机，机器恢复后事务会回滚



- 如果事务处于commit阶段，由于clog已经落盘，即使发生宕机场景，事务都会执行完成，只是业务端可能会收到事务unknown的回复，需要业务端confirm事务的状态

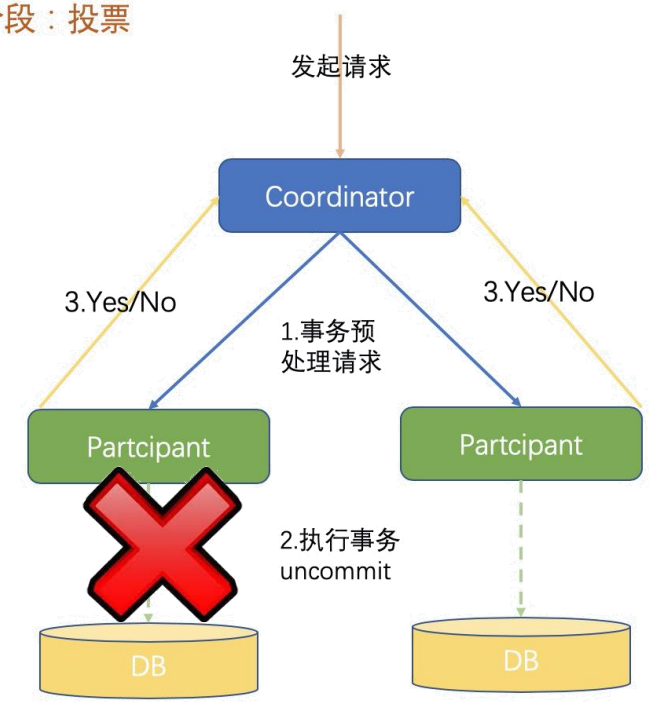


5.2 两阶段提交过程中参与者宕机

还未进入Prepared状态

- 参与者所有事务状态丢失
- 参与者会应答协调者prepare unknown消息
- 事务最终会abort

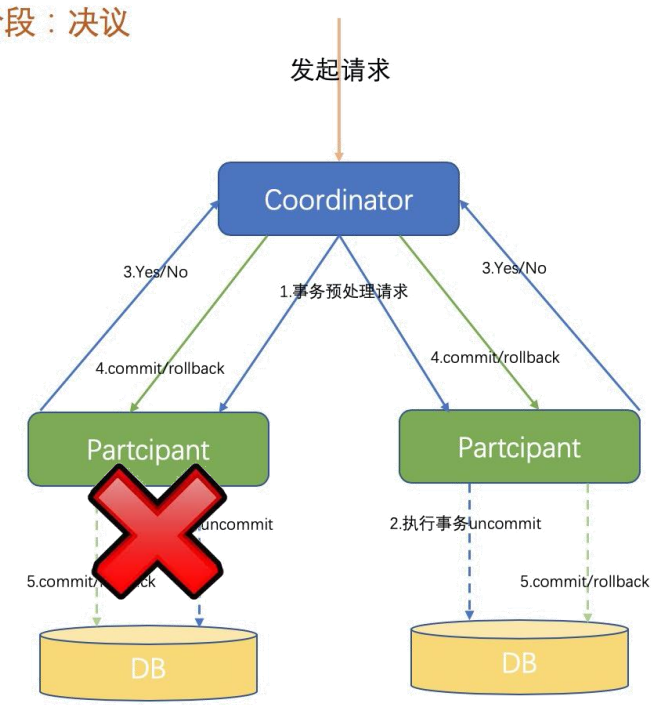
第一阶段：投票



已进入Prepared状态

- 状态已经Paxos同步
- 系统会自动选择一个副本，作为新的leader并恢复出prepare状态，协调者继续推进

第二阶段：决议



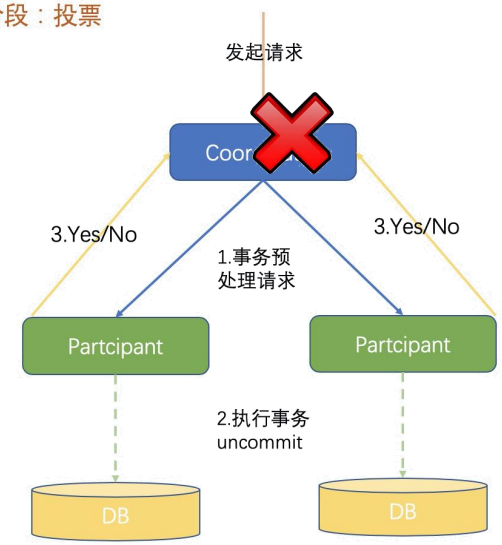
5.2 两阶段提交过程中协调者宕机

1. 协调者与第一个参与者是同一个partition
   - 参与者状态机恢复遵从参与者自己的逻辑
   - 协调者状态机恢复由参与者回复协调者的消息触发

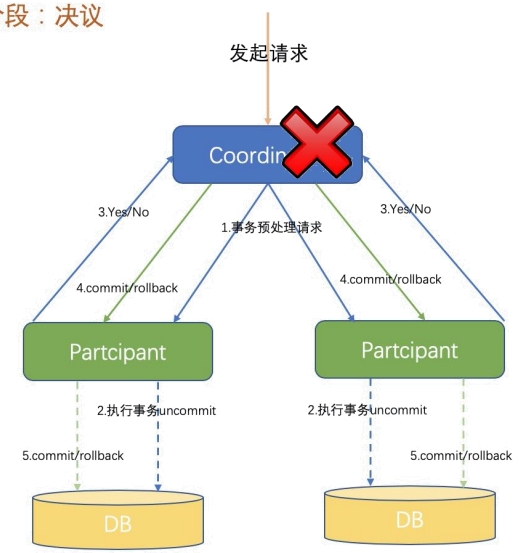
2. 参与者发送prepare ok后未收到协调者进一步消息（commit/abort）时，认为上一条回复消息丢失，会定时重新发送上一条消息

3. 所有参与者都记录全部参与者列表

第一阶段：投票



第二阶段：决议



5.2 分布式事务底层优化

1. 单分区事务：不走2PC，直接写一条日志即可完成事务提交
2. 单机多分区事务→ 优化的两阶段提交
3. 多机多分区事务→完整的两阶段提交 →prepare, commit/abort

5.2 分布式事务调优方法

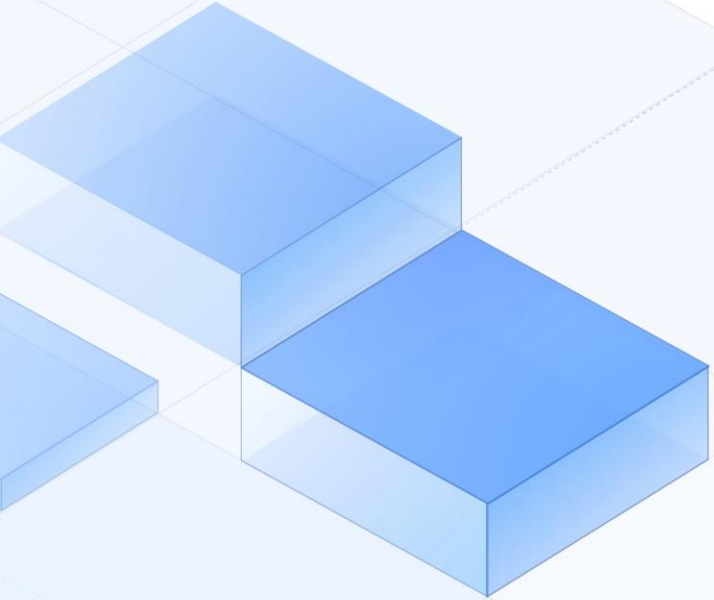
1. 业务数据模型设计原则：尽量避免跨机分布式事务

2. 单sql语句不建议跨机器

   通过table group、primary_zone把相关的表的leader放在同一个机器上

3. 慎重选择事务中的第一条语句，因为Obproxy的路由规则

感谢学习



OCEANBASE