烽火星空

NJSD

中国（南京）软件开发者大会

China (Nanjing) Software Developers Conference

2016

从1到10

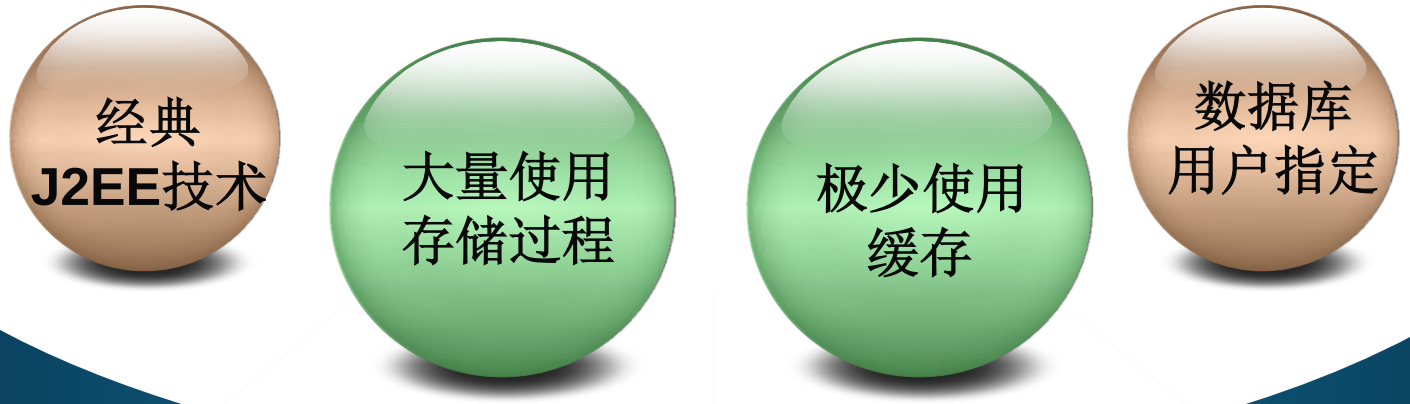
传统企业J2EE系统性能提升架构实践

www.exmobi.cn

内容

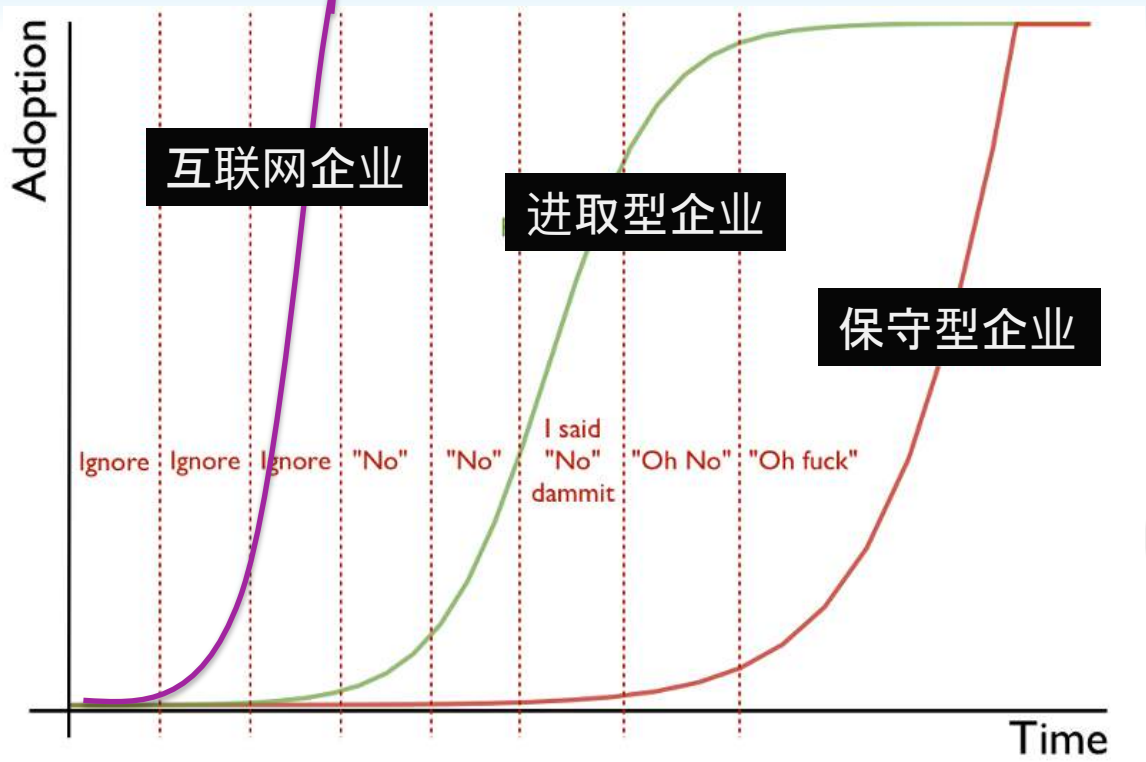
- 传统企业系统的特点
- 烽火星空的架构实践
- 分布式内存数据库架构
- 特点、优点、缺点
- 构架的技术实现细节

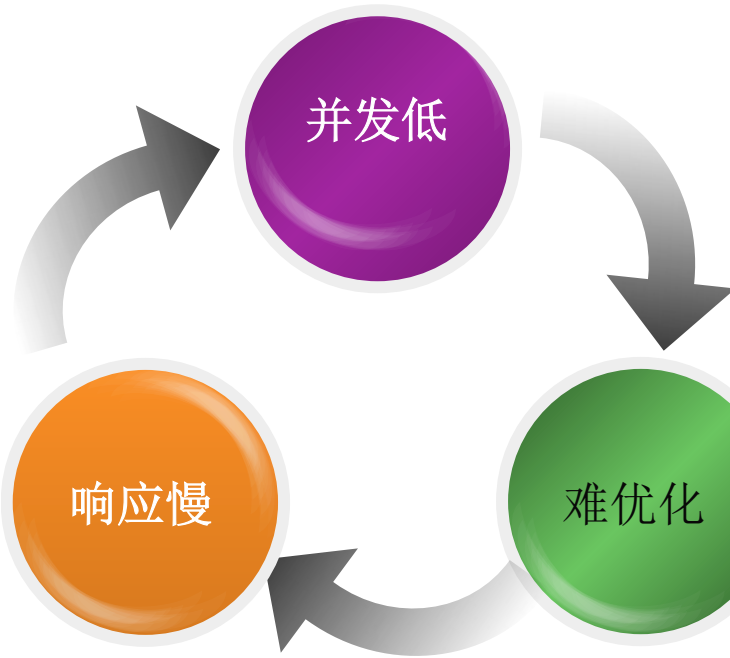
传统企业系统的技术特点



大部分基于J2EE技术构建
实施时间短，投入成本低
业务变化频繁，系统变更频繁
开发人员变更大，维护困难

传统企业系统的技术特点





传统企业系统性能慢，与此同时
大部分决策者对新技术的接受度比较低，接受缓慢
对于传统企业，新兴的技术框架不一定是最好的选择

传统企业系统的技术特点

- **数据库是性能瓶颈**

- 数据是企业的重要资产，目前大部分企业只信任Oracle、DB2等数据库
- 数据库服务器软硬件成本昂贵，难以利用数据库自身的分布式处理技术
- 很多开发人员钟情于存储过程的开发方式，数据库服务器同时承担了存储和业务逻辑计算
- 数据的水平和垂直切分会给企业业务的实现带来大量的复杂逻辑，极少被采用
- 数据库自身的缓存设置专业性强，精通人员少，难以实施

传统企业系统的技术特点

IO操作阻塞

数据库连接池一般限制在100以内，数据库操作的阻塞导致连接池容易耗尽

异步编程难堪大用

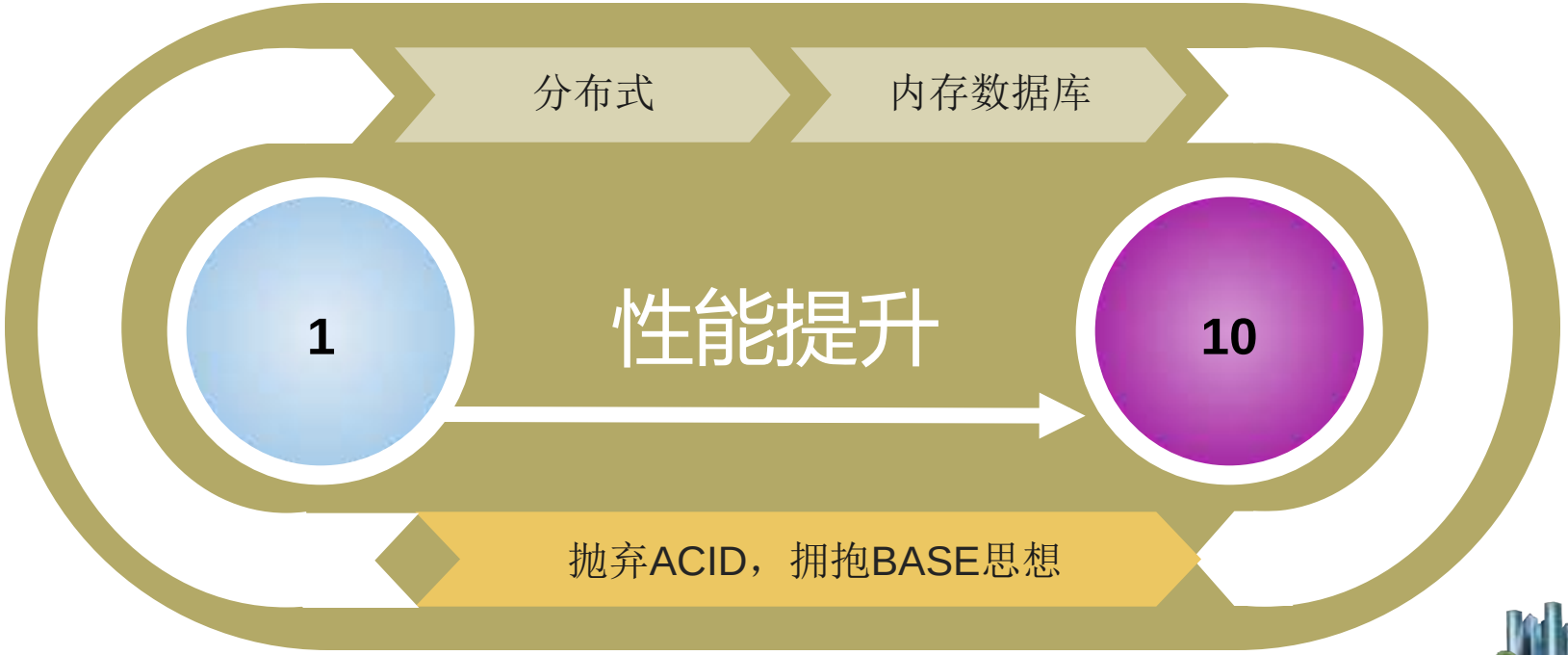
异步编程反人类习惯，组件多阻塞
协程（微线程）技术框架不够成熟
数据库终究还是瓶颈

多线程陷阱

线程过多会导致系统性能严重下降
大部分时间消耗在锁和线程切换
CPU缓存利用率低，易导致伪缓存

烽火星空架构实践

企业移动化业务使用率迅速提升，过去系统几十到数百的TPS，难以满足客户需求。烽火星空在MPlus等产品中应用分布式内存数据库技术，在同等硬件条件下，性能得到10倍以上的提升。



烽火星空架构实践

- 使用自主开发的分布式内存数据库技术框架，他好我也好

	使用前	使用后
系统性能	ART：100~4000ms TPS：200+ 分布式：性能佳	ART：10~1000ms TPS：2000+ 分布式：性能难改善
项目成本	开发3套存储过程 性能调优难度高	一举多得，成本低 性能优秀，实施顺畅

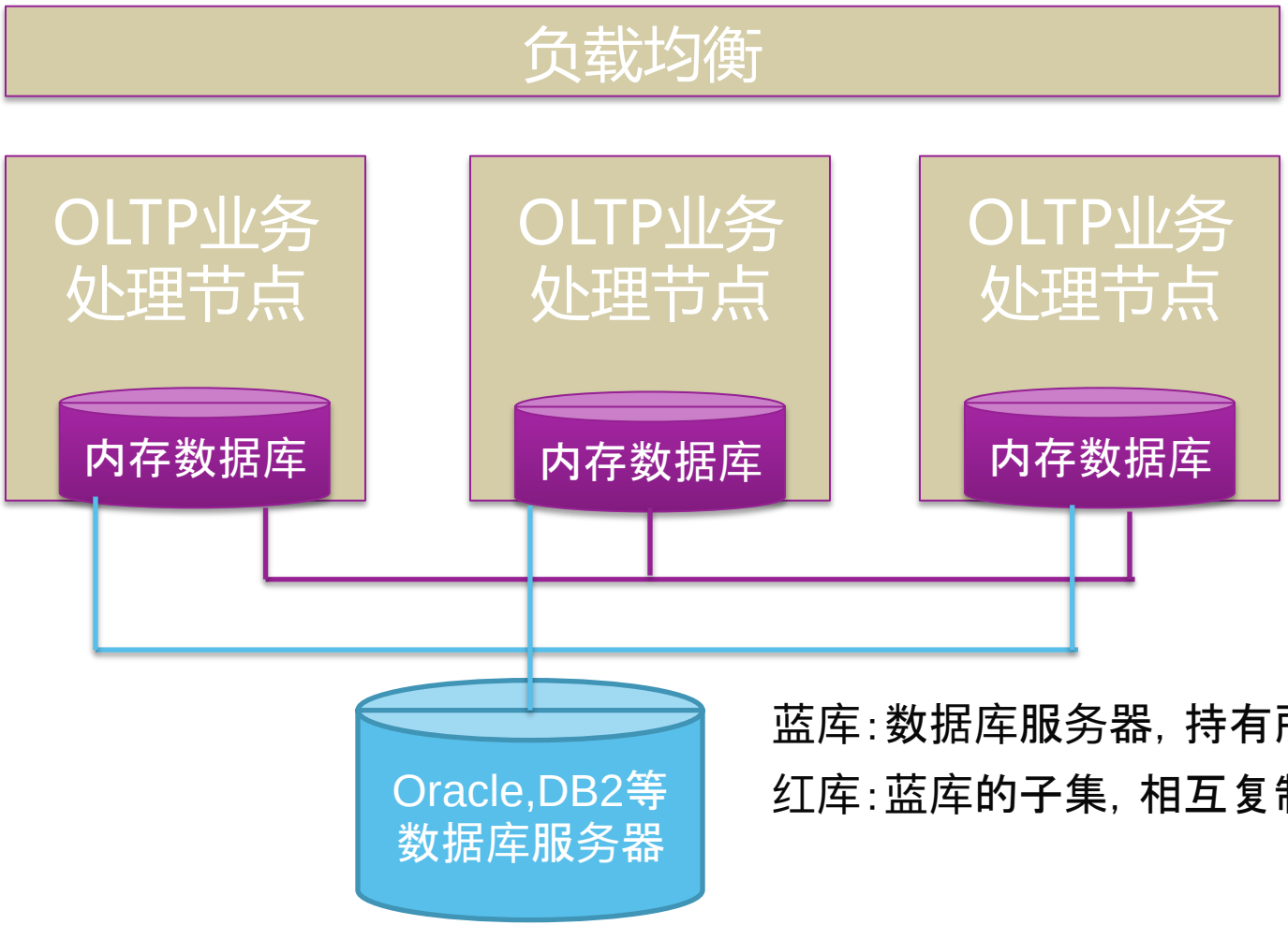
烽火星空架构实践

通过业务手段来处理一致性要求高的逻辑

一定是现有开发人员熟悉的开发模式
1小时内上手

部署简单，自治性高
框架无扰于开发人员

分布式内存数据库架构



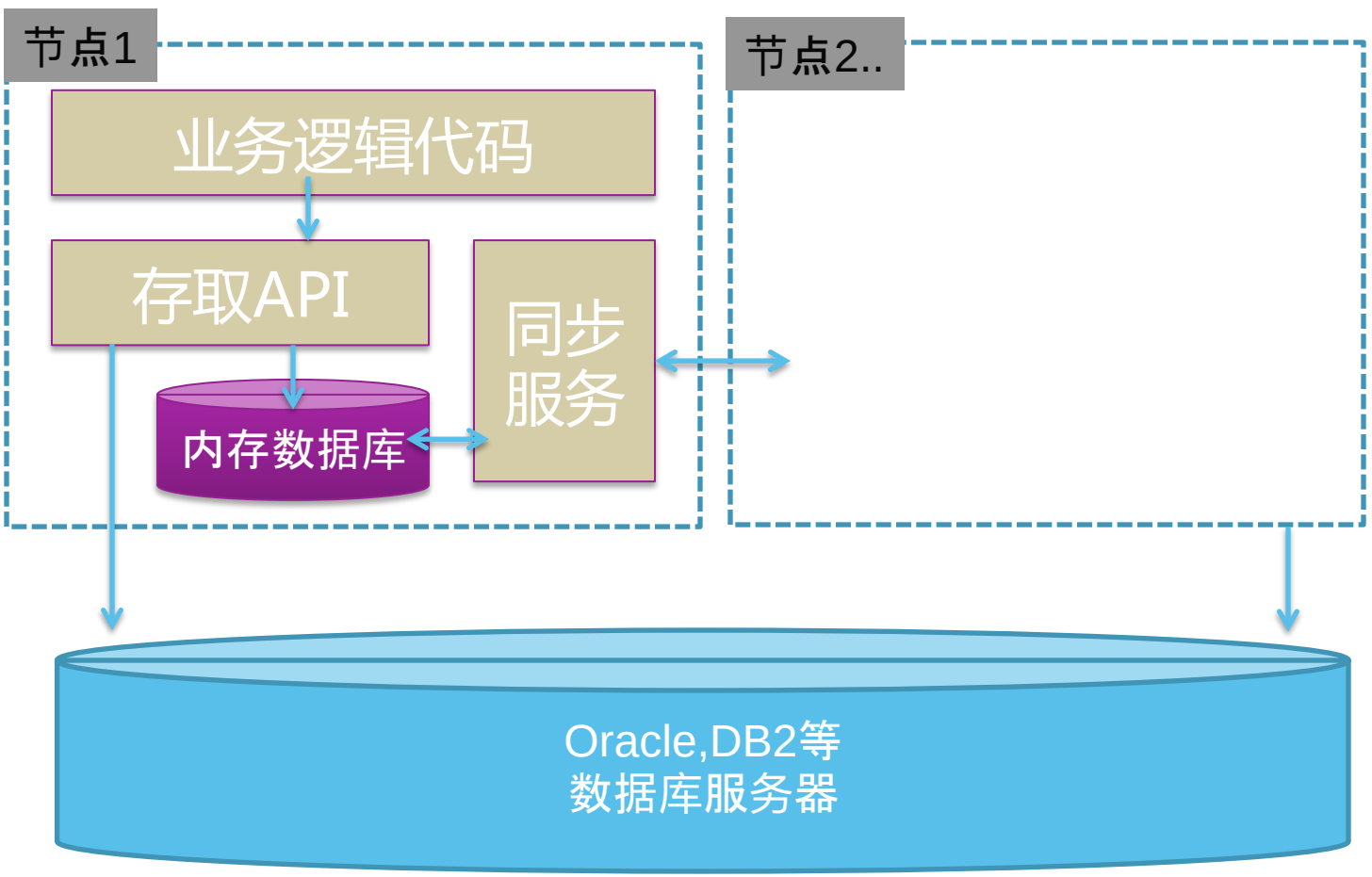
拓扑结构

蓝库与红库为君臣关系
君只有一个，臣有多个
有福(更新)同享
有难(查询)臣先挡

蓝库：数据库服务器，持有所有数据

红库：蓝库的子集，相互复制增量同步

分布式内存数据库架构

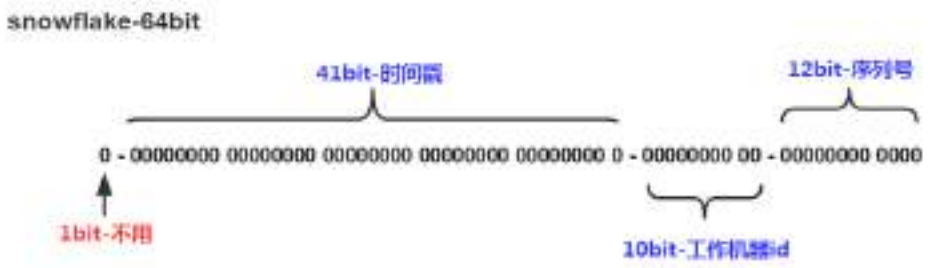


模块
关系

分布式内存数据库架构

开发约束

1 不使用存储过程
2 不使用触发器
3 不设计外键
4 主键业务无关
5 不使用序列发生器等
6 不使用自增字段
7 尽可能使用单表查询



红库中的表，
主键使用64bit整数
分布式自增序列
采用snowflake算法

分布式内存数据库架构

数据存储引擎

性能优化	优化前	优化后
查询性能	约18万/秒	约40万/秒
修改性能	约15万/秒	约30万/秒

（SQLITE引擎在不做修改的情况，在单线程操作下具有最高性能）

优化的内存数据库存储引擎
基于ｓｑｌｉｔｅ进行改造

1 去掉数据库加锁机制
2 在存取库保证单线程执行
更高效的利用ＣＰＵ缓存
3 增加HASH索引

优化后，带来如下优势：
１）数据存取更快
２）事务不会产生死锁

分布式内存数据库架构

查询示意：

```
Row  row = db.query("SELECT catalog,type,name FROM t_asset_types WHERE assetid=?",
                    new Object[]{id});
System.println(" catalog=" + row.col("catalog", "")
            + ", type=" + row.col("type", 0));

Rows rows = db.querys("SELECT catalog, name FROM t_asset WHERE type=? ORDER BY ? ",
                   new Object[]{type, "name"});
System.println(" assets=" + rows.size());
Row row = rows.row(i);
...
```

更新（INSERT、UPDATE、DELETE）示意：

```
int deleted = db.update("DELETE FROM t_asset WHERE assetid=?", new Object[]{id});
```

异步更新（提交请求，不等待结果）示意：

```
db.asyncUpdate("INSERT INTO t_operates(..) VALUES(?...);", new Object[]{...});
```

事务：

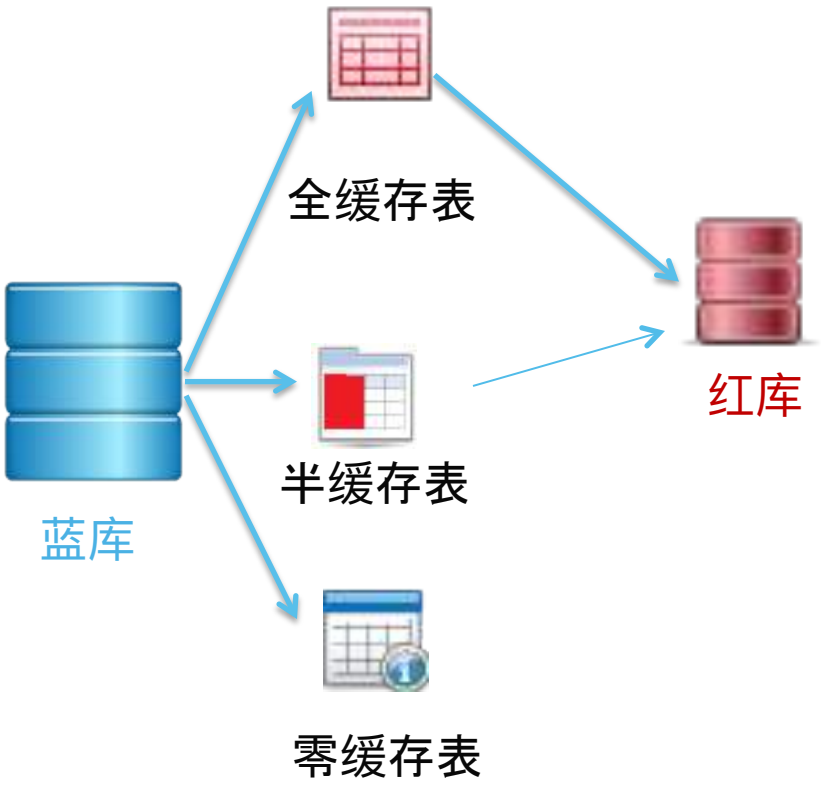
```
Transaction ts = new Transaction();
ts.beginTransaction();
ts.update(sql, parameters);
ts.update(....)
ts.commit(); / ts.rollback();
```

极简存取API

根据缓存配置规则，ＳＱＬ被自动解析和路由，
在红库或蓝库或２者中执行

缓存规则

分布式内存数据库架构



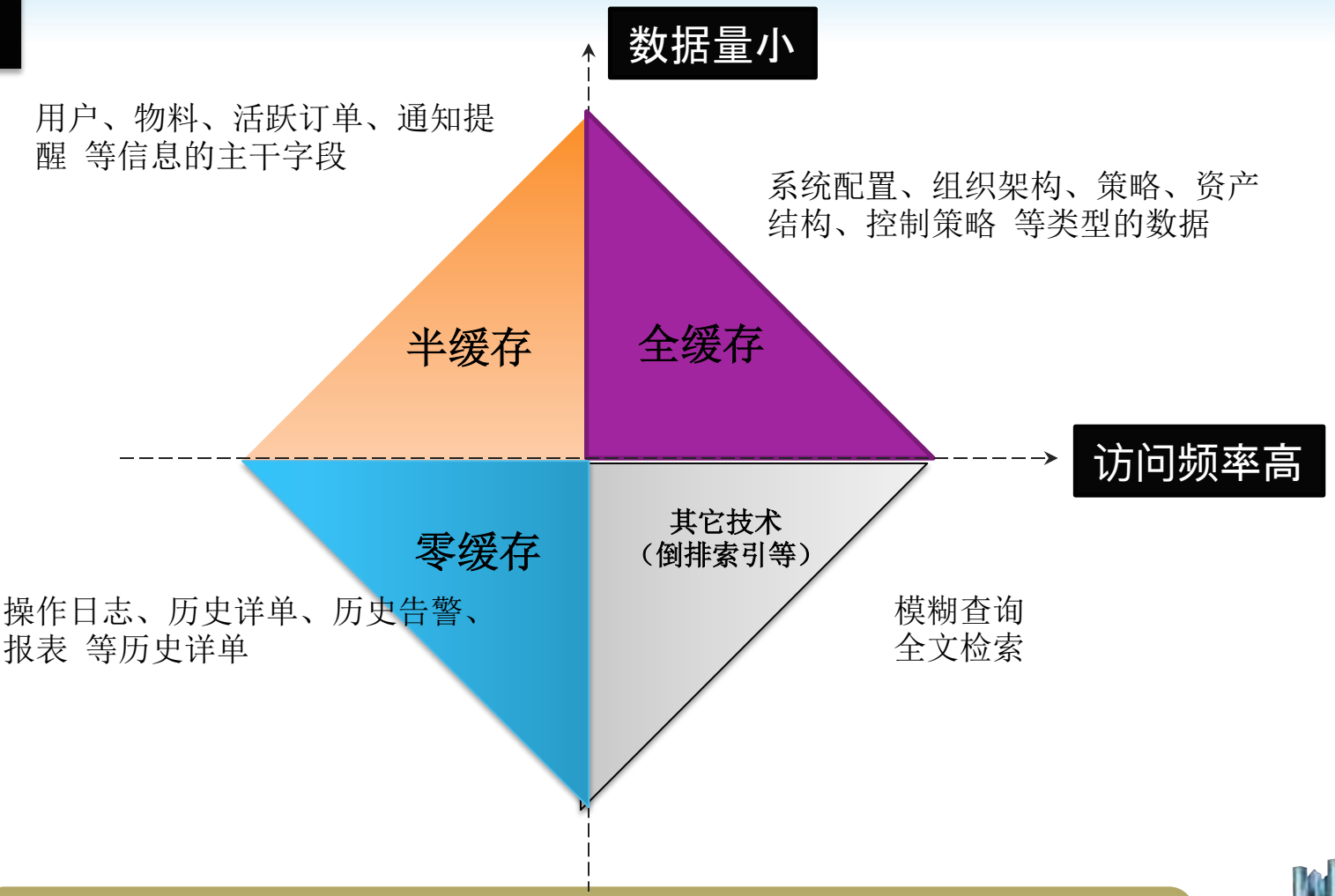
节点在启动时，根据ＳＱＬ文件和配置文件中的缓存规则，建立内存表，同步数据

以表为单位配置缓存规则：

1 全缓存表：所有数据和内存数据库同步
2 索引缓存表：内存数据库只缓存部分字段
3 无缓存表：内存数据库不存此表

配置策略

分布式内存数据库架构



根据业务类型，判断表的读写类型，配置缓存策略

特点、优点、缺点

特点：配置型缓存

基于SQL的缓存，技术门槛低
分布式，横向扩展，可启停
配置型缓存，无需代码改动

优点：实施简单

无需进行复杂的缓存逻辑编码
代码简单

缺点：约束严格

数据库设计约束多，抛弃编写存储过程的开发习惯
对数据库操作具有独占性

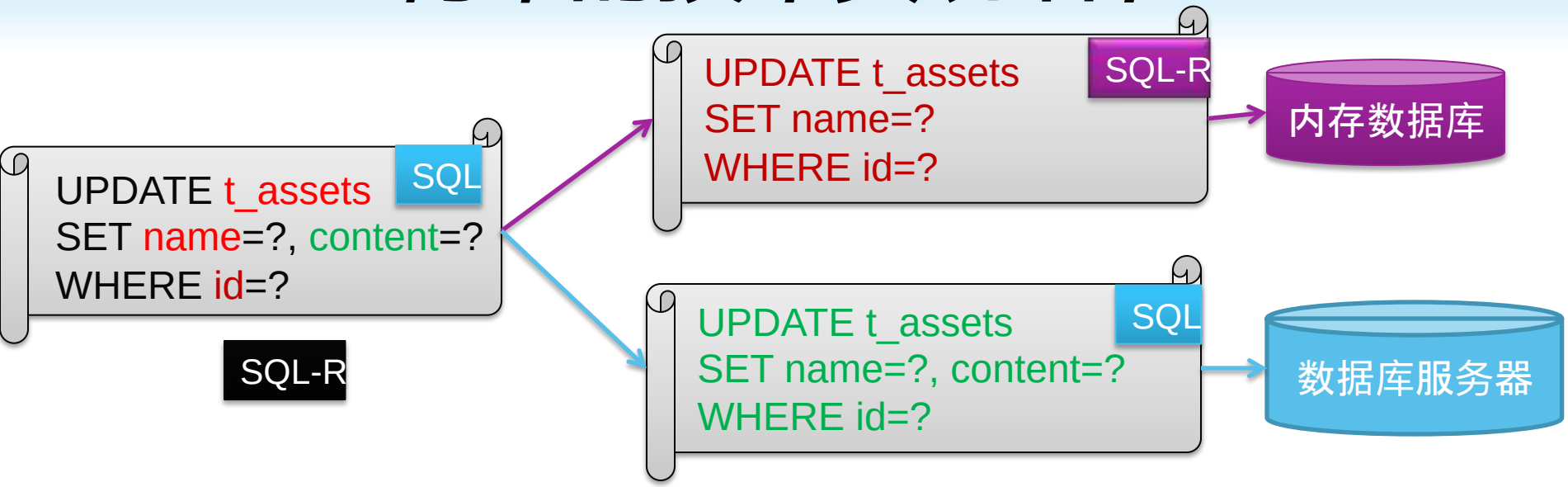
查询语句分析执行

构架的技术实现细节

```
SELECT catalog, name, content, description
FROM t_assets
WHERE  id=?
ORDER BY display_order
```

1）表属于“全缓存表”时：查询语句在 内存数据库中执行
2）表属于“零缓存表”时：查询语句在 数据库服务器执行
3）表属于“半缓存表”时，执行“蓝色污点原则”：
SQL 语句中只要有字段为蓝色（未缓存），则在数据库服务器执行
SQL语句中字段全部为红色时，则在内存数据库中执行

构架的技术实现细节

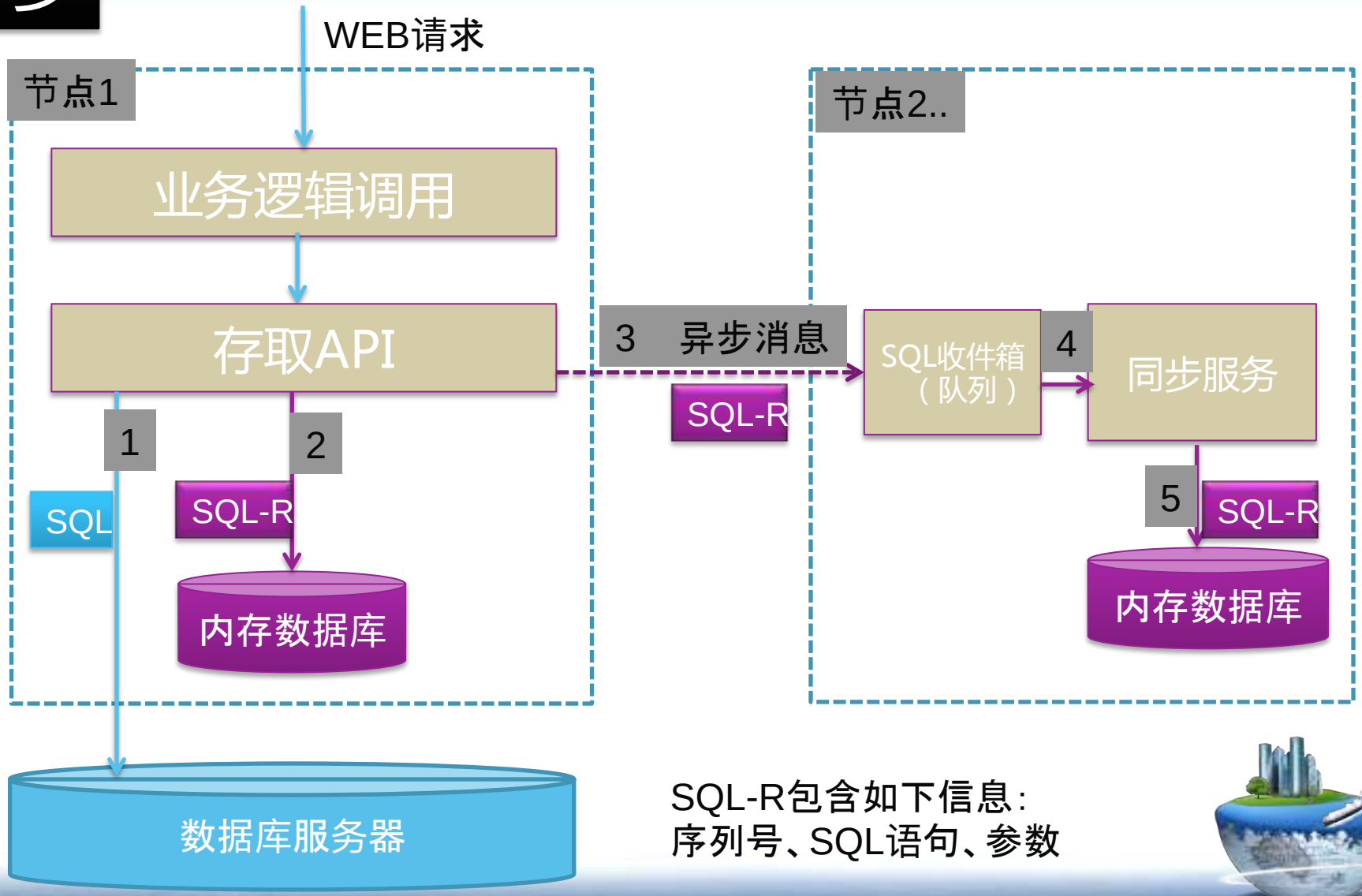


更新语句分析执行

1）表属于“全缓存表”时：SQL语句在DiskDB中执行成功后，在MemDB执行
2）表属于“零缓存表”时：SQL语句在DiskDB中执行
3）表属于“半缓存表”时，
SQL要求：条件语句中的字段必须为索引字段，且为红色
SQL语句在DiskDB中执行，
抽取红色字段，组成SQL语句，在MemDB执行

更新同步

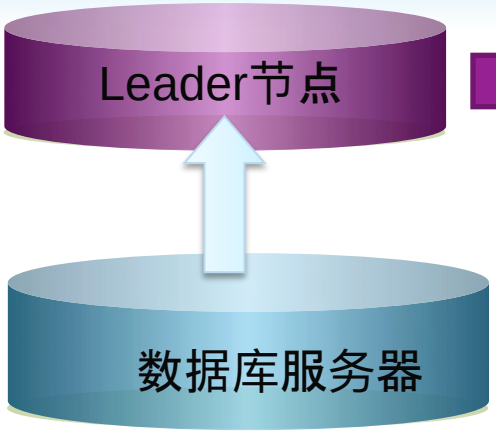
构架的技术实现细节



SQL-R包含如下信息：
序列号、SQL语句、参数

构架的技术实现细节

启动阶段同步



冷复制：复制数据期间不处理业务请求，Leader节点采用

热复制：复制数据期间，Leader节点可以处理业务请求，Slave节点采用

启动阶段

1. 确定身份

确定自己是否是Leader节点

1.1. 主库复制

从数据库服务器复制数据

1.2. 辅库复制

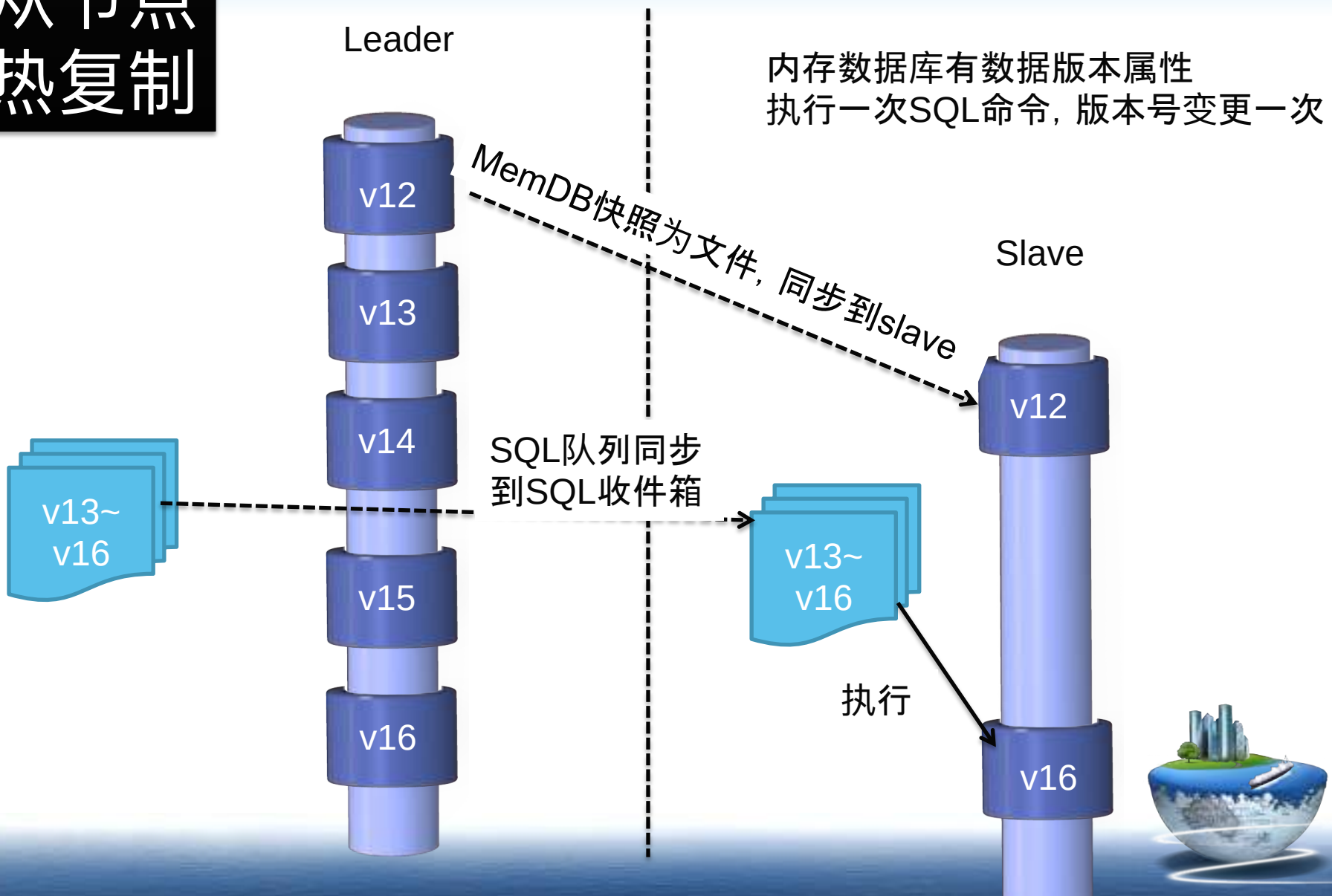
从Leader节点复制数据

2. 完成

进入运行状态

构架的技术实现细节

从节点热复制



构架的技术实现细节

事务一致性分析

不同的业务请求，有不同的一致性要求
以医院开通的在线挂号系统为例

A 低

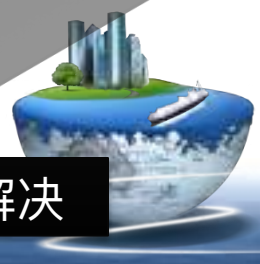
1）对数据更新的延时不敏感
2）写少读多
例如：
医院介绍
专家介绍
新闻动态

B 中

1）能忍受少量延时
2） 会话内的修改即使生效（自己看到自己的修改）
例如：
询问、回复，点赞

C 高

1）多人修改所有人同时可见
例如：
挂专家号
（资源有限）
预约手术



对于A、B，分布式内存数据库完全可以Cover

C通过消息队列处理方式解决