DTCC

第十四届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA

数智赋能 共筑未来

IT168.com ChinaUnix ITPUB

北京国际会议中心 | 2023/8/16-18

云原生模块化数据库

成章数据
CEO
陈亮

主题

- 云计算和硬件资源的灵活伸缩
- 模块化架构：OLTP 数据库功能模块的细粒度伸缩
- 模块化架构和多模态：持久化缓存数据库 (Redis)

云计算和可伸缩资源

- 多种资源
  - 计算资源： Intel, ARM, GPU
  - 内存资源
  - 存储资源： EBS, S3, DynamoDB, …
- 快速申请和释放
- 多种性价比
- 多种计费模型

可伸缩数据库

云计算

可伸缩计算资源
可伸缩内存资源
可伸缩存储资源

数据库功能模块

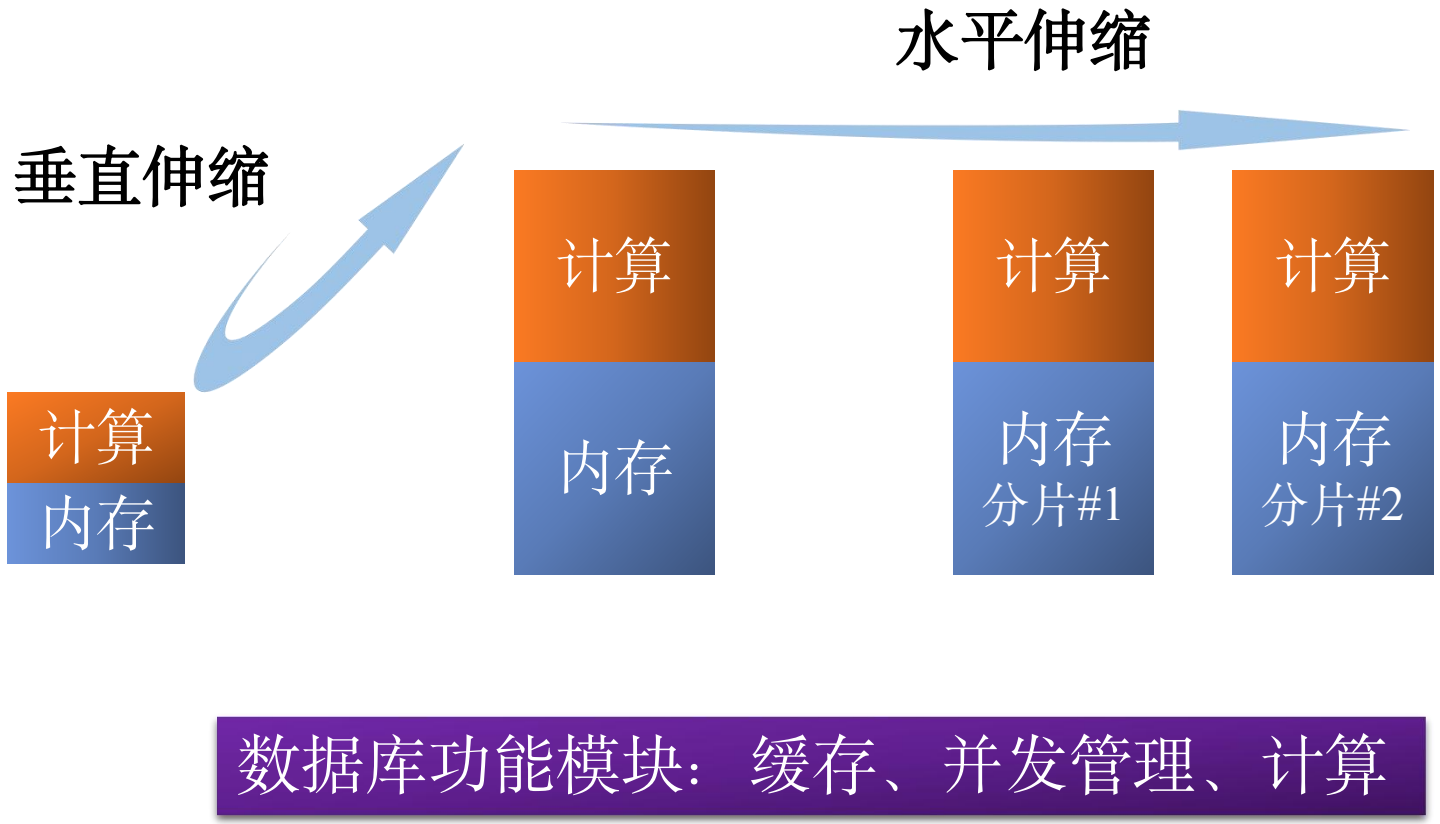
缓存
并发管理
日志
数据存储
计算

云原生数据库

可伸缩缓存
可伸缩并发管理
可伸缩日志
可伸缩数据存储
可伸缩计算

“存算分离”抽象无法支持数据库功能模块的灵活伸缩

资源伸缩：计算+内存

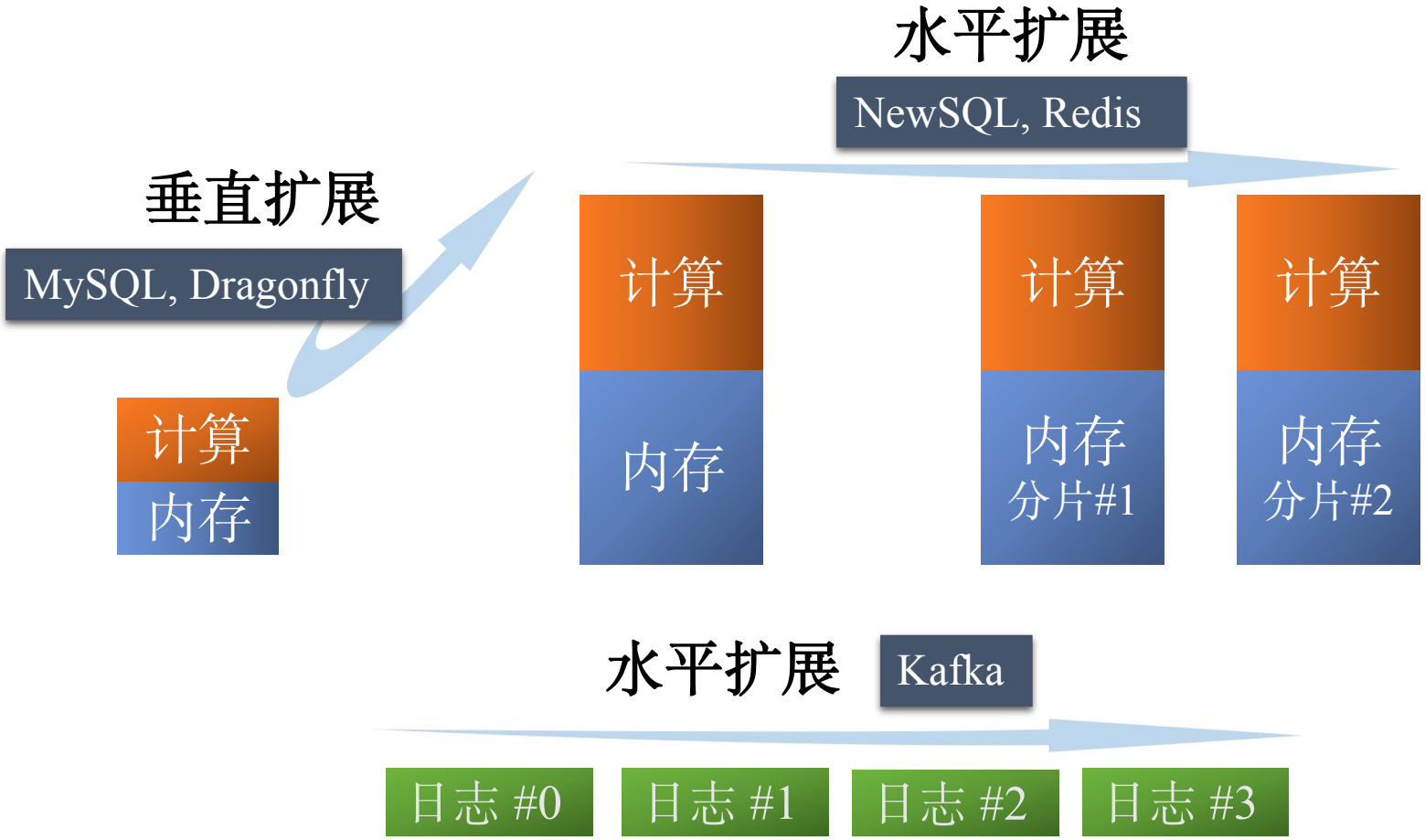


数据库功能模块：缓存、并发管理、计算

资源伸缩：存储

- 存储容量：从垂直扩展到水平扩展
- 同步写 (fsync) 只能水平扩展
  - 单位 fsync 延时: 100 us，每秒 10,000 fsync’s
  - 增加并发 → 提高延时

云原生数据库：混合扩展


水平扩展
NewSQL, Redis
垂直扩展
MySQL, Dragonfly
计算
内存
计算
内存
计算
内存
分片#1
计算
内存
分片#2
水平扩展
Kafka
日志 #0
日志 #1
日志 #2
日志 #3

混合扩展

在任意规模下灵活扩展任意模块

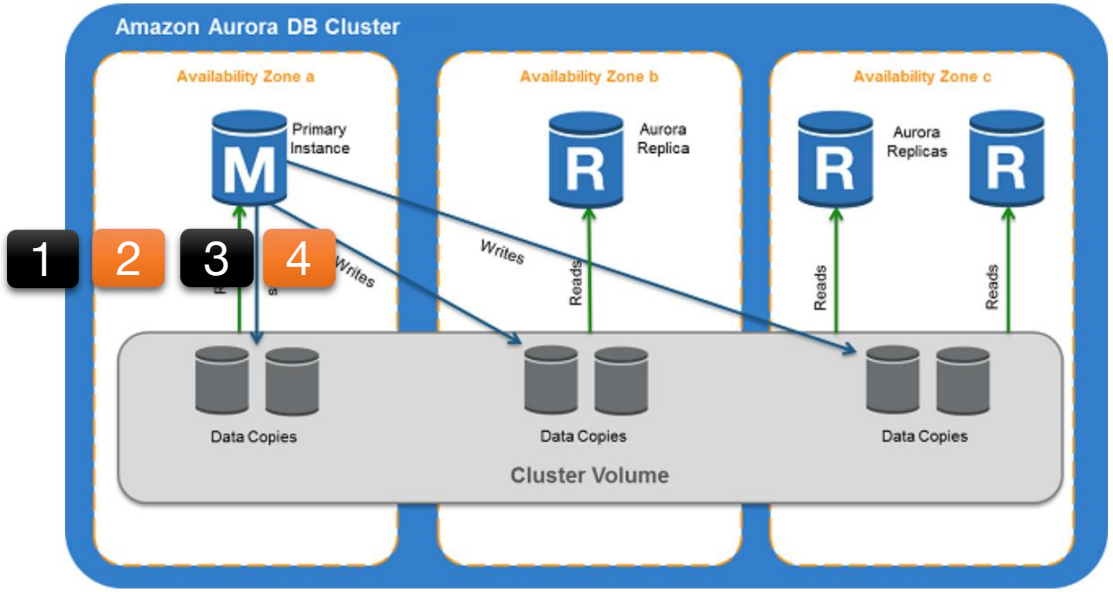
- 对小规模流量, 所有模块部署在一台机器，即单机数据库。
- 对大规模读流量，水平扩展缓存模块获得大规模分布式缓存。
- 对大规模写流量，水平扩展日志模块获得分布式并行日志。
- 对大数据，水平扩展存储模块来增加存储容量。

高性价比：扩展性能瓶颈模块

- 为了极致读性能, 扩展缓存模块将所有数据缓存，即纯内存数据库。
- 为了极致写性能，扩展日志模块获得低延时、高吞吐写入性能。

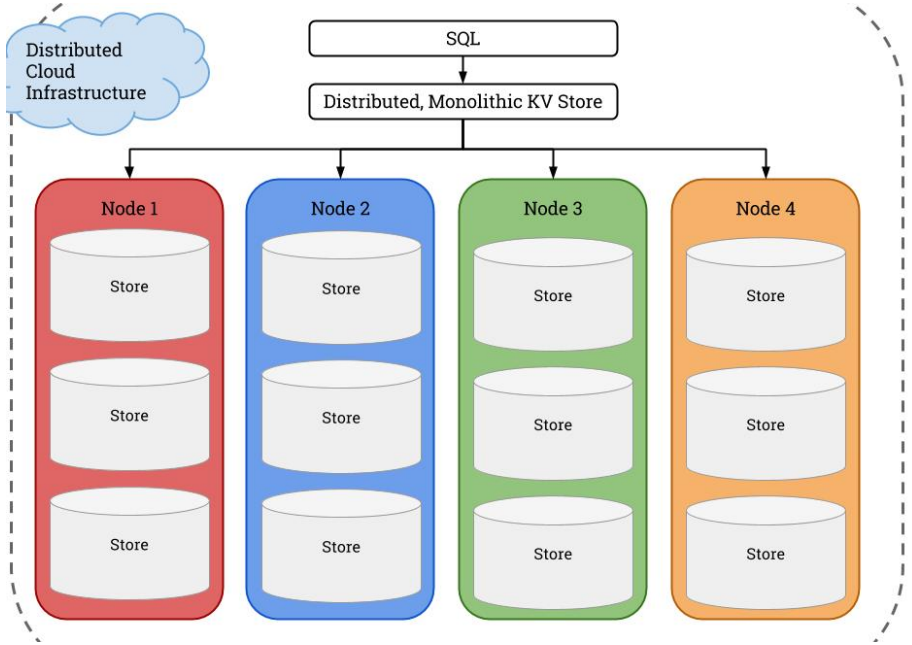
AWS Aurora

- 数据容量可扩展
- 缓存不可扩展
  - 每个节点缓存所有数据
  - 缓存率: 2% (200 GB mem, 10 TB data)
- 日志不可扩展
  - 单节点写入
  - 串行提交协议
    - LSN_k 提交成功前，$LSN_1 \cdots LSN_{k-1}$ 必须提交成功

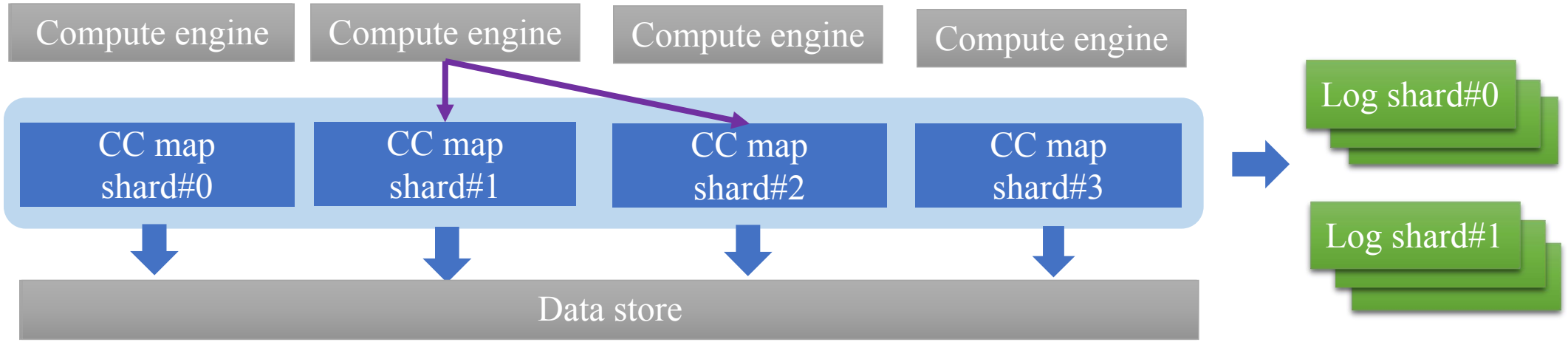


NewSQL

- 数据、缓存、日志可扩展，但必须同步扩展
  - 缓存规模 ∝ 数据节点数量
  - 日志规模 ∝ 数据节点数据
  - 水平扩展需要迁移大量数据，扩展缓慢。
  - 无法快速扩展来适应各类流量
    - 大规模读需要大规模缓存
    - 大规模写需要大规模日志

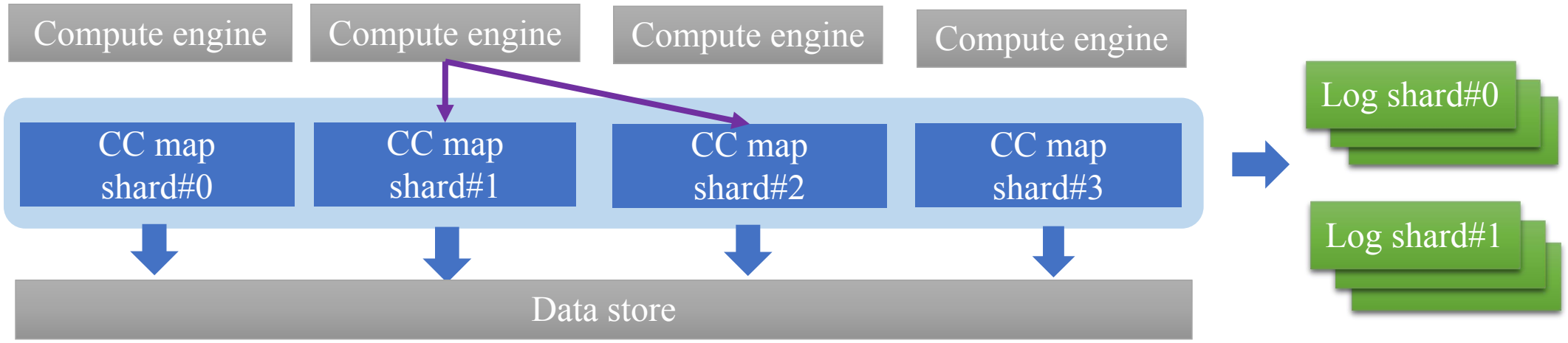


基于数据基层的模块化



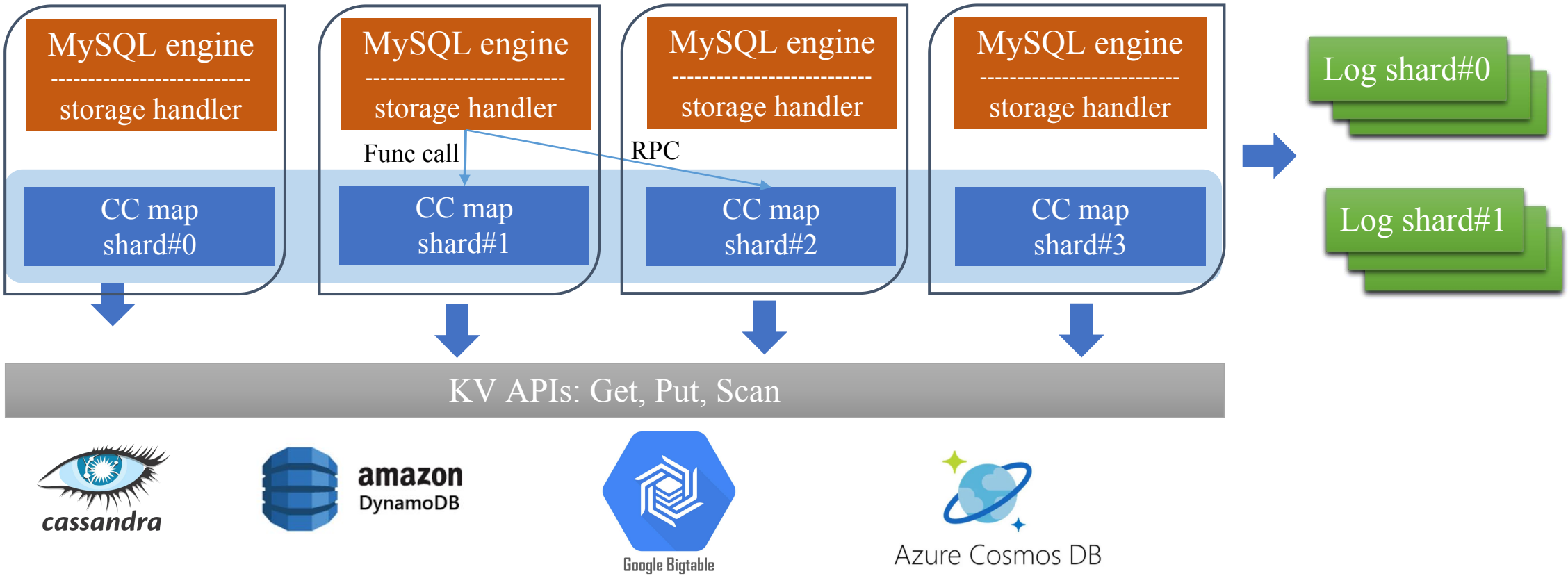
- 分布式内存表负责缓存和并发管理 (CC map)
  - Map: Key → <Payload, Lock>
- 分布式日志负责数据持久化
  - 数据修改关键路径: (1)内存表加锁, (2)同步写入日志, (3)内存中释放锁及更新数据
  - 基于逻辑时钟的并行提交协议

基于数据基层的模块化

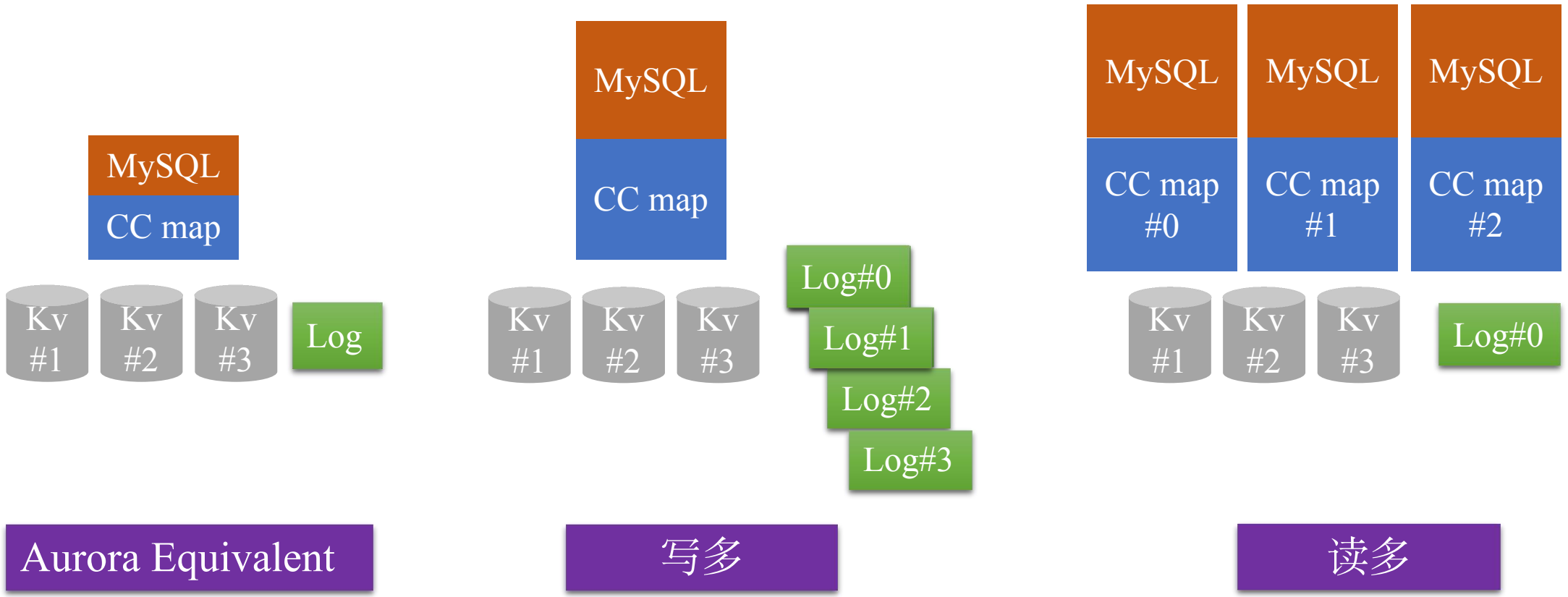


- 缓存替换算法将冷热数据在缓存和存储间置换
  - 更新的缓存数据在踢出前写入数据存储
- 分布式容错协议利用分布式日志实现内存节点的 failover
  - 未写入存储的缓存数据从日志恢复到缓存

基于数据基层的云原生 MySQL



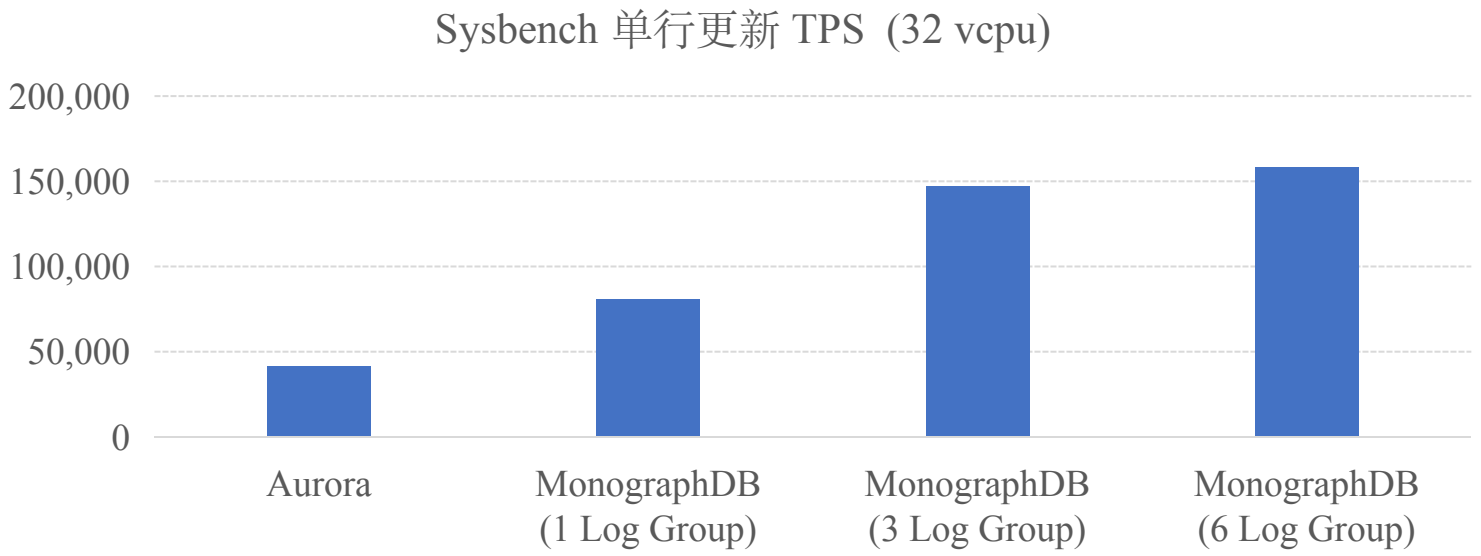
混合扩展的 OLTP 数据库



案例分析：写性能

- Bechmark: Sysbench, oltp_update_non_index
- 每日志分区跨 3 AZ 复制
  - 1, 3, 6 日志组
- 日志网盘
  - GCP Persistent Disk (SSD)
- 单计算/内存节点: 32 vcpu (Intel Ice Lake), 256 GB mem
- 延时限制
  - avg: 5-7 ms
  - 95%: <= 10 ms

结果1: 并行提交协议和多日志分区



- 并行提交协议有助于写入性能
- 多个日志分区进一步提升性能

结果2:垂直扩展计算和水平扩展日志

Log Group #	计算内存节点 32 vcpu (TPS)	计算内存节点 64 vcpu (TPS)
1	80,947	84,827
3	147,033	155,733
6	158,400	165,864

- 给定日志规模，扩展计算内存资源无法大幅提升写性能
- 写性能由计算(内存)、日志联合决定

结果3:水平扩展日志和存储成本

GCP	Volume (per GB)	IOPS (per IOPS)
SSD	$0.17	included
Extreme	$0.125	$0.065

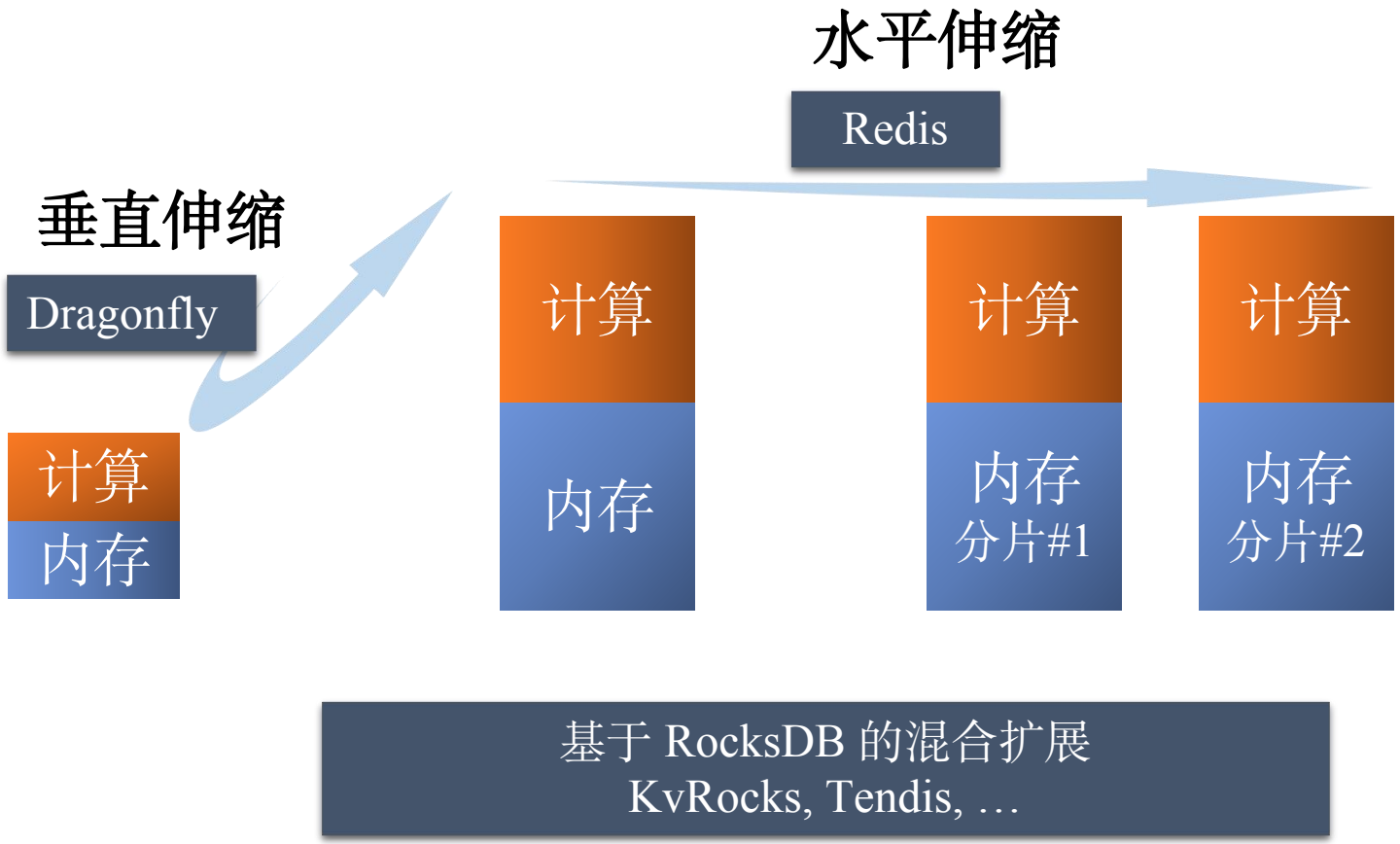
64 vcpu, SSD

Log group #	TPS	Monthly cost ($)
3	155,733	0.17*30 (GB) *3 (replicas) *3 = 46
6	165,865	0.17*30 (GB) *3 (replicas) * 6 = 92

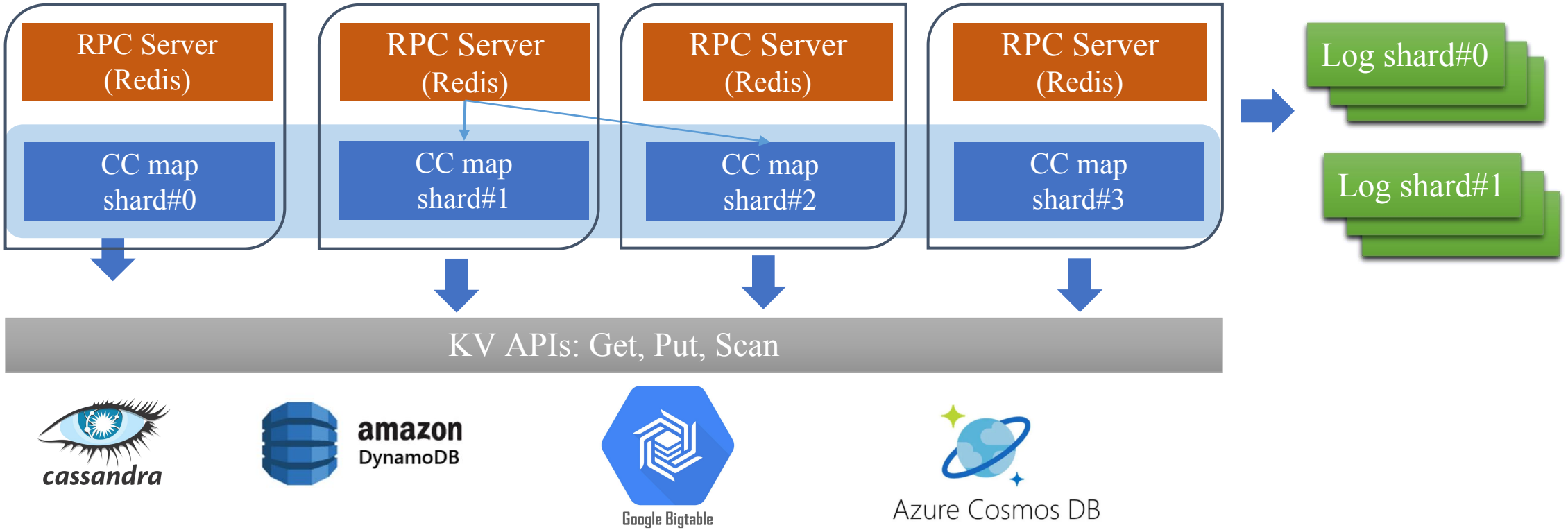
64 vcpu, Extreme

Log group #	TPS	Monthly cost ($)
3	168,450	0.125*30 (GB) *3 (replicas) *3 = 33.75 **0.065*9*3000 = 1,755**
6	254,574	0.125*30 (GB) *3 (replicas) * 6 = 67.5 **0.065*18*3000 = 3,510**

混合扩展的缓存数据库


水平伸缩
Redis
垂直伸缩
Dragonfly
计算
内存
计算
内存
计算
内存
分片#1
计算
内存
分片#2
基于 RocksDB 的混合扩展
KvRocks, Tendis, …

基于数据基层的缓存数据库



基于数据基层的缓存数据库

- 原生 Redis 内存对象
  - RocksDB 需要把 Redis 对象映射成 blob

Redis 数据定义：

```
struct RedisObj : public TxRecord {}
struct ListObj : public RedisObj {
  List<string> data_;
};
```

MySQL 数据定义：

```
struct RowObj : public TxRecord {
  string row_;
};
```

数据基层实例化：
CcMap<string, RedisObj>

数据基层实例化：
CcMap<RowKey, RowObj>

基于数据基层的缓存数据库

- TxCommand 修改 RedisObj 状态，并返回修改后的结果
  - RocksDB: 将 Redis 命令映射成若干个 Put(), Get() 的组合

Redis 数据访问：

```
struct Lpush : public TxCommand {
    void Apply(TxRecord &) override {}
};
```

SQL 数据访问：

```
struct UpdateCmd : public TxCommand {
    void Apply(TxRecord &) override {}
};
```

- 写入路径
  - 加锁
  - (optional) 写日志: 持久化 <key, TxCommand, commit_ts>
  - 放锁，作用 TxCommand 到 Redis 对象，并返回

测试

- 64 vcpu, 1 log group
- 500 connections

	Get	Set (No WAL)	Rpush (No WAL)
RocksDB-based	548,000	410,000	124,000
MonographDB for Redis	1,530,000	1,550,000	564,000

总结

- 云原生数据库需要功能模块的细粒度伸缩
- 基于数据基层的模块化架构实现了数据库的混合扩展
- 模块化架构带来全新的产品力
  - 覆盖多种流量场景
  - 更经济的扩展
  - 为在线业务提供丰富的产品矩阵 (OLTP、缓存数据库)

敬请期待
MySQL 和 **Redis** 版本 @ AWS, GCP, Azure

THANKS