DTCC 2010-2022

第十三届中国数据库技术大会

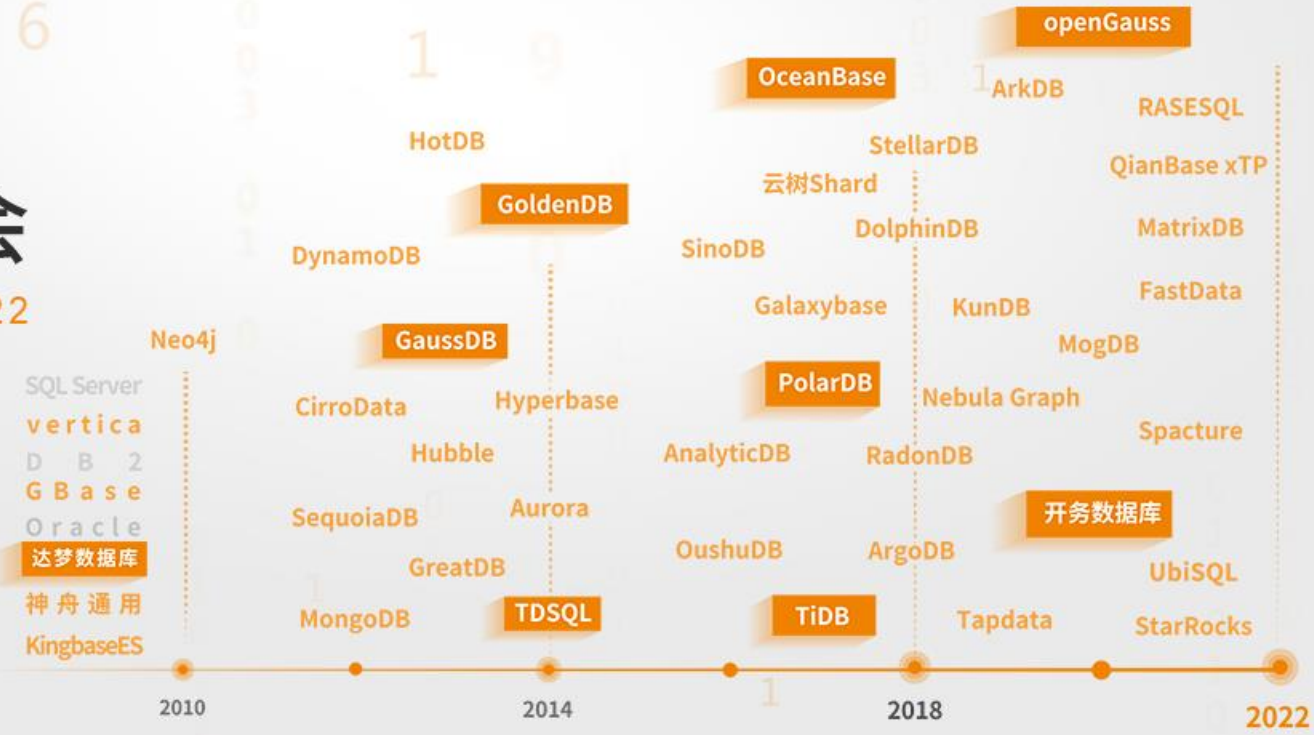
DATABASE TECHNOLOGY CONFERENCE CHINA 2022

数据智能 价值创新

IT168.com ChinaUnix.net ITPUB

线上直播 | 2022/12/14-16

数据来源：数据库产品上市商用时间



Amazon DynamoDB 助力现代化应用程序

李君
亚马逊云科技
数据库技术专家

议题

1. Amazon DynamoDB 简介
2. Amazon DynamoDB 极致弹性和底层设计
3. Amazon DynamoDB 设计最佳实践
4. Amazon DynamoDB 全球部署与服务集成

Amazon DynamoDB 简介

Amazon DynamoDB

快速且灵活的 NoSQL 数据库，规模没有限制

规模性能

- 提供更稳定的毫秒级延迟
- 每秒处理数百万个请求

无需管理服务器

- 免维护
- Auto-scaling
- 按需容量模式
- 高达 99.999% 的 SLA

适合大型企业

- ACID 事务
- 静态加密
- 连续备份，按需备份和还原
- 与其他亚马逊云科技服务集成

全局表

- 构建全球应用程序
- 实现对本地数据的快速访问
- 自动化全球复制

Amazon DynamoDB 非常适合：

无服务器的事件驱动型架构 • 遍及全球的弹性服务 • 高吞吐量工作负载

Amazon DynamoDB 用例

客户依赖DynamoDB来支持其任务关键型工作负载

金融服务

支付和欺诈检测
用户配置文件和活动
手机银行
(Capital One, Vanguard, Fannie Mae)

游戏

游戏状态
排行榜
玩家数据存储
(Riot Games, Electronic Arts, PennyPop)

软件和互联网

元数据存储
网络安全
数字信息
(Swiggy, Snap, Duolingo)

营销与广告

广告定位与属性
网络/社交媒体分析
深度链接
(AdRoll, GumGum, Branch, DataXu)

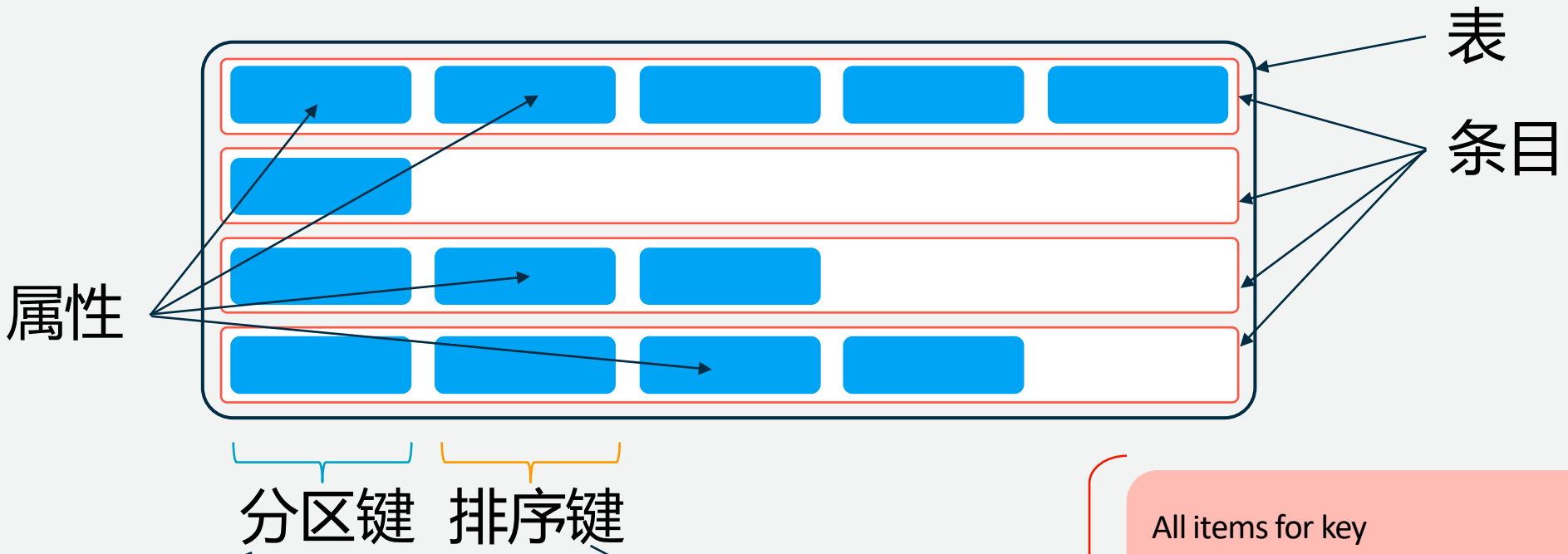
零售

购物车和订单
库存与履行
付款和优惠券赎回
(Nordstrom, Nike, Zalando, Mercado Libre)

媒体与娱乐

用户数据存储
媒体元数据存储
流媒体观看列表和书签
(Airtel Wynk, Amazon Prime, Netflix)

表



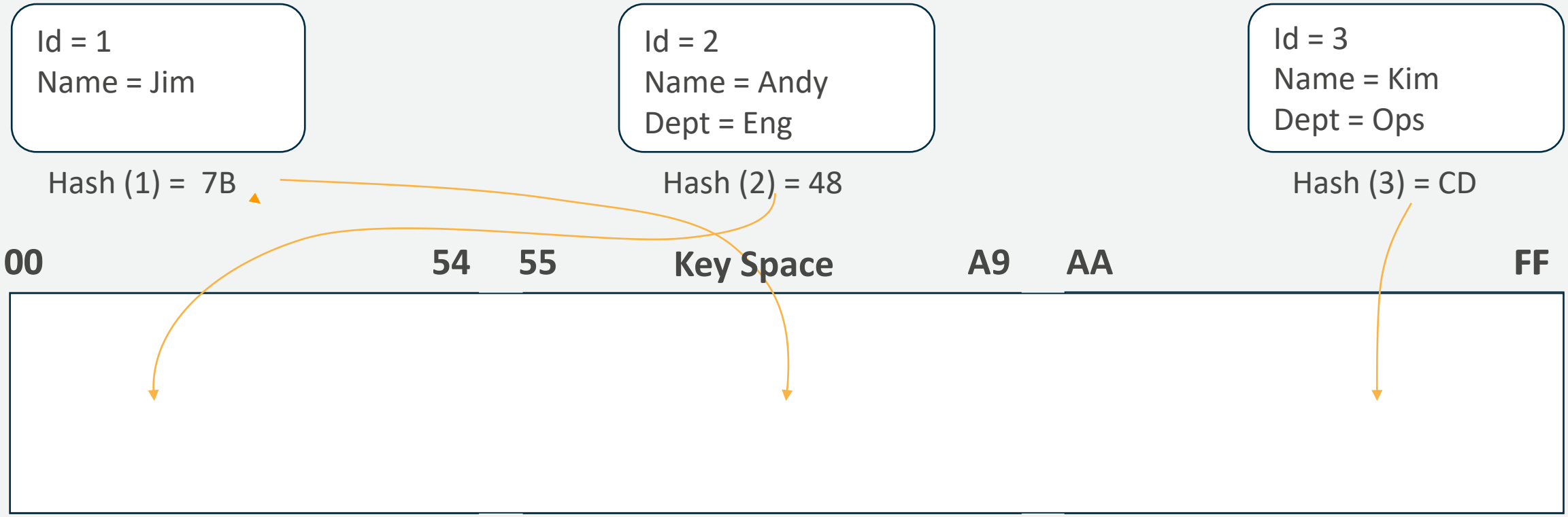
必须指定
键值访问方式
决定数据如何分布

可选
用于1:N的关系
提供很多的查询优化的能力

All items for key
==, <, >, >=, <=
“begins with”
“between”
“contains”
“in”
sorted results
counts
top/bottom N values

分区键

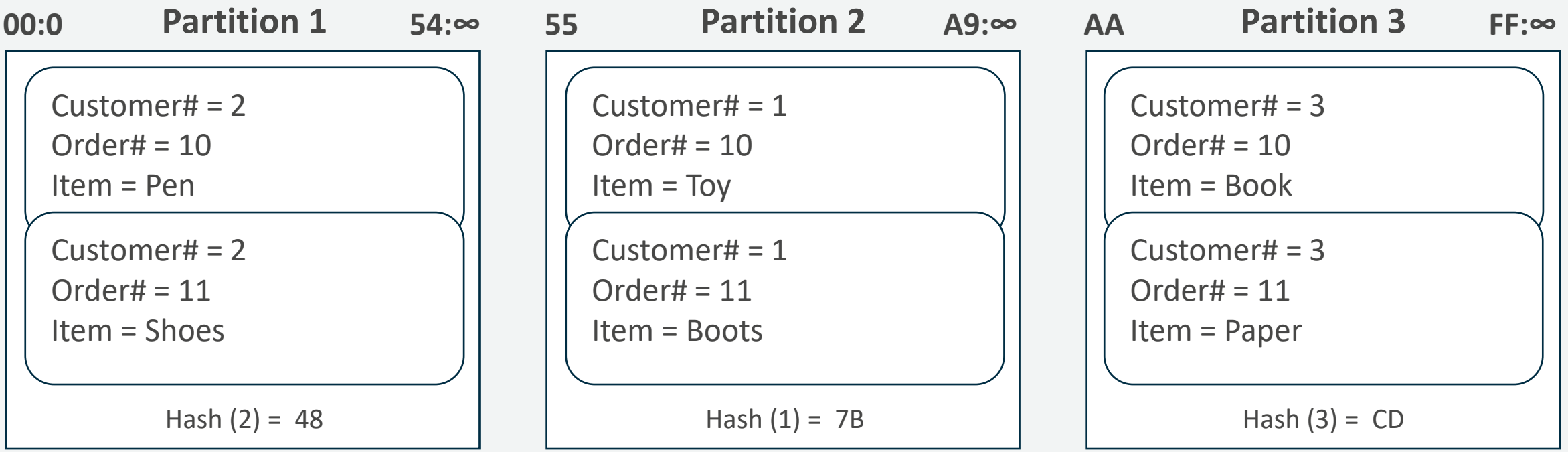
分区键唯一的标识了一条记录
分区键用来构建一个非排序的散列索引
使得表可以进行分区，从而满足扩展性的需求



分区键：排序键

分区键和排序键共同唯一的标识一条记录
在一个分区键决定的散列索引里，数据按照排序键进行排列
每个排序键所对应的数据行数没有上限

- 除非你有本地二级索引（local secondary indexes）



主键：唯一地标识一个项目

简单主键：分区键

SensorId (分区键)	纬度	经度
SensorA	40.712784	-74.005941
SensorB	35.689488	139.691706

SensorLocation 表

复合主键：分区键+排序键

SensorId (分区键)	时间 (排序键)	值
SensorA	2018-01-03T10:15:30	30
SensorA	2018-01-04T10:19:30	35
SensorB	2018-03-04T11:21:20	28

SensorReadings 表

项目和属性 类型

键-值项目 →

ItemNumber	Title	LeadActor	Price	InCirculation
12	"My Favorite Video"	"James Smith"	5.00	true

字符串 ↑ "James Smith"

数字 ↑ 5.00

布尔值 ↑ true

文档项目 →

JSON

```
{
 "person_id" : 123,
 "last_name" : "Doe",
 "first_name" : "John",
 "next_anniversary" :
 {
  "year" : 2018,
  "month" : 5,
  "day" : 30
 },
 "children"  :
  ["Dick", "Harry", "Jane", "Mary" ]
}
```

映射 ←

列表 ←

读取和写入一致性

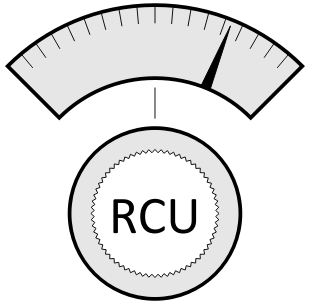
DynamoDB 会维护数据的多个副本以获得更高的持久性。

您可以在读取数据时指定所需的一致性级别。

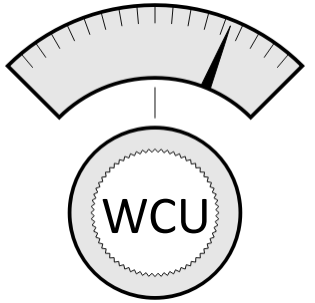
- 读取一致性级别
  - 最终一致性
  - 强一致性
  - 事务
- 写入一致性级别
  - 标准
  - 事务

读取和写入吞吐量

- 读取容量单位 (RCU)：
  强一致性读取：最大大小为 4KB 的项目的每秒读取次数
  最终一致性读取：最大大小为8KB的项目的每秒读取次数



- 写入容量单位 (WCU)：
  每秒写入大小为 1KB 的项目的次数



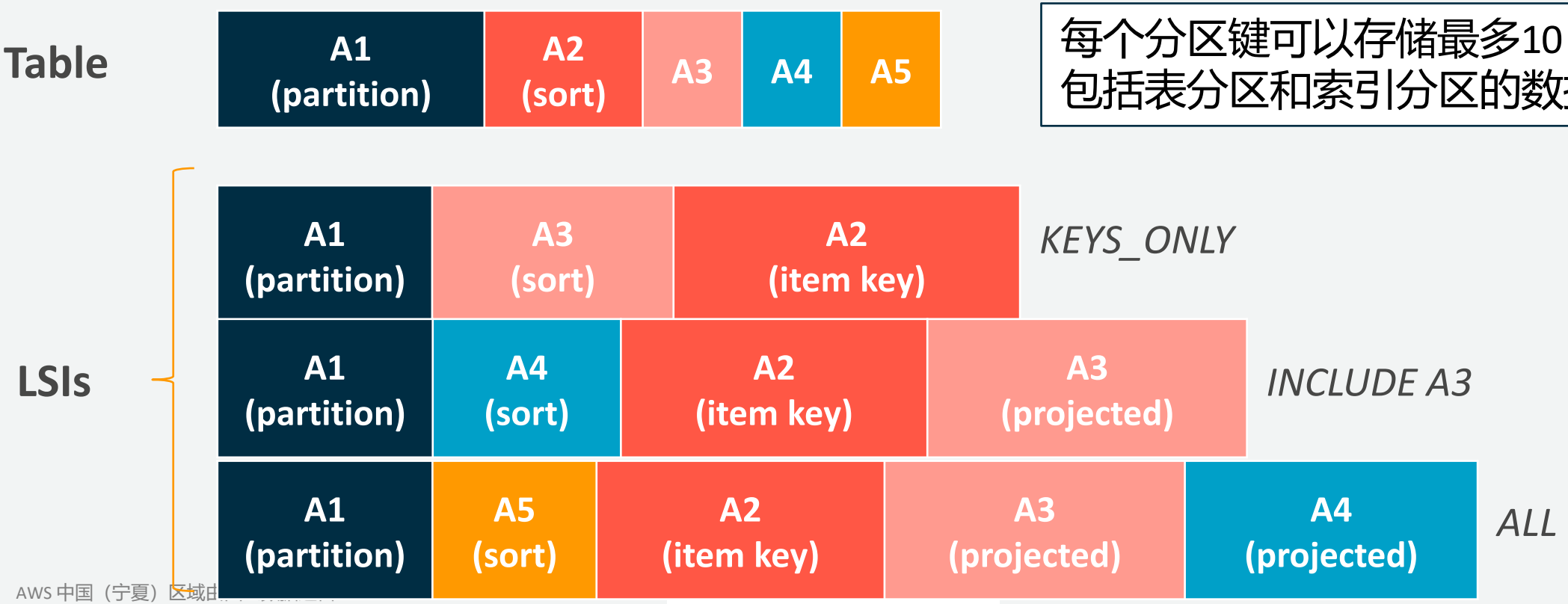
注意：吞吐量在各分区之间均衡分配。

二级索引

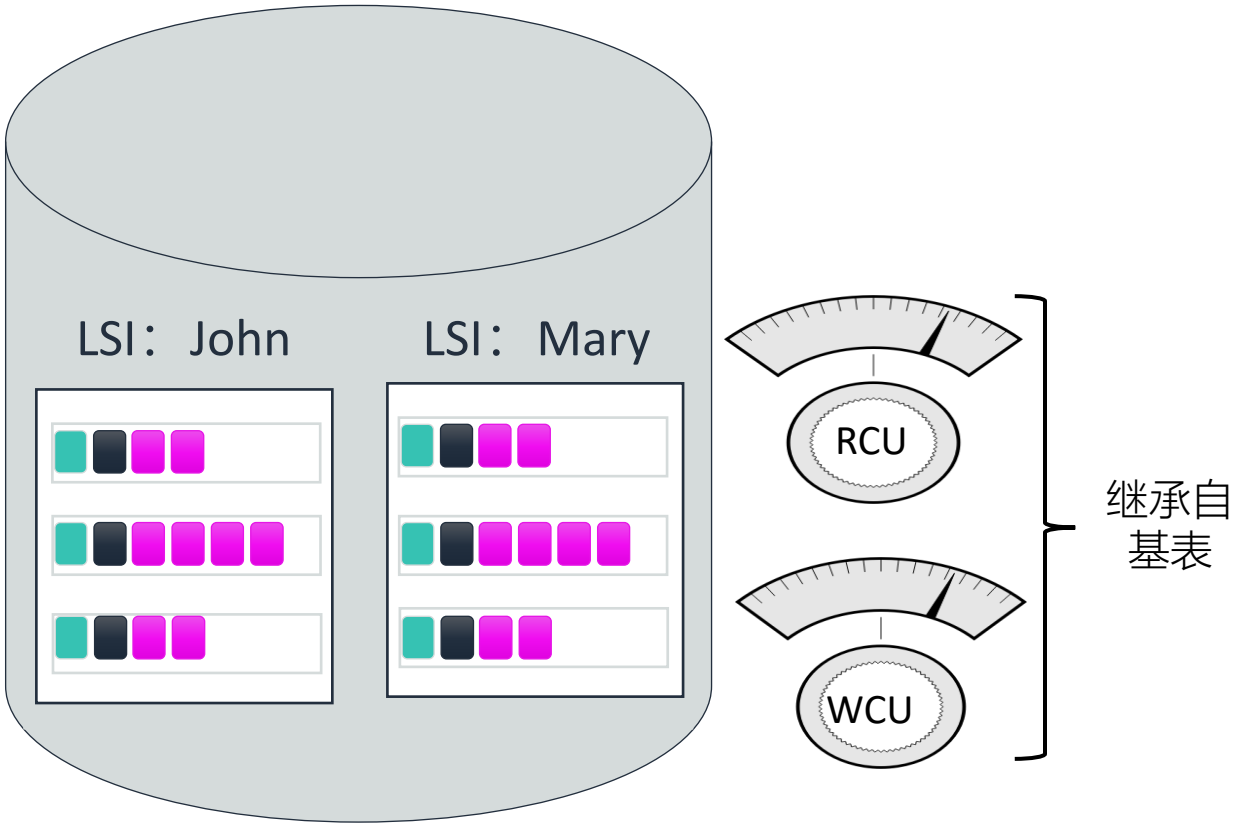
- 让您可以基于非主键属性查询数据
  - 二级索引会定义替代键
- 包含：
  - 替代键属性
  - 主键属性
  - 基表中其他属性的可选子集（投影属性）
    - 可选类型：KEYS_ONLY, INCLUDE A3, ALL
- 可以是以下其中一种类型：
  - 全局二级索引 (GSI)
  - 本地二级索引 (LSI)

本地二级索引 - Local Secondary Index (LSI)

可以选择与表不同的排序键
每个表分区对应一个索引分区



示例：本地二级索引



本地二级索引
分区键和排序键

TravelerSurvey 表

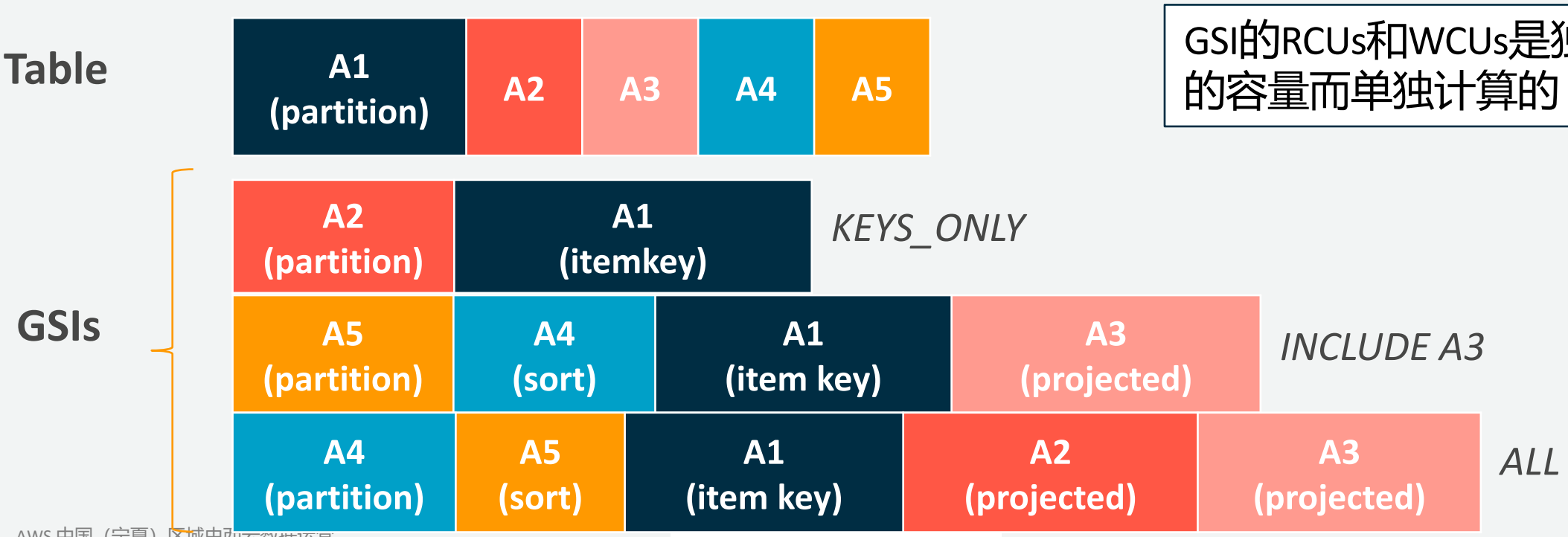
RepId（分区键）	TravelerId（排序键）	ContactDate
John Doe	1	2018/05/01
John Doe	3	2018/05/02
Mary Smith	2	2018/05/10

TravelerSurveyByRepAndDate 本地二级索引

RepId（分区键）	ContactDate（排序键）	TravelerId
John Doe	2018/05/01	1
John Doe	2018/05/02	3
Mary Smith	2018/05/10	2

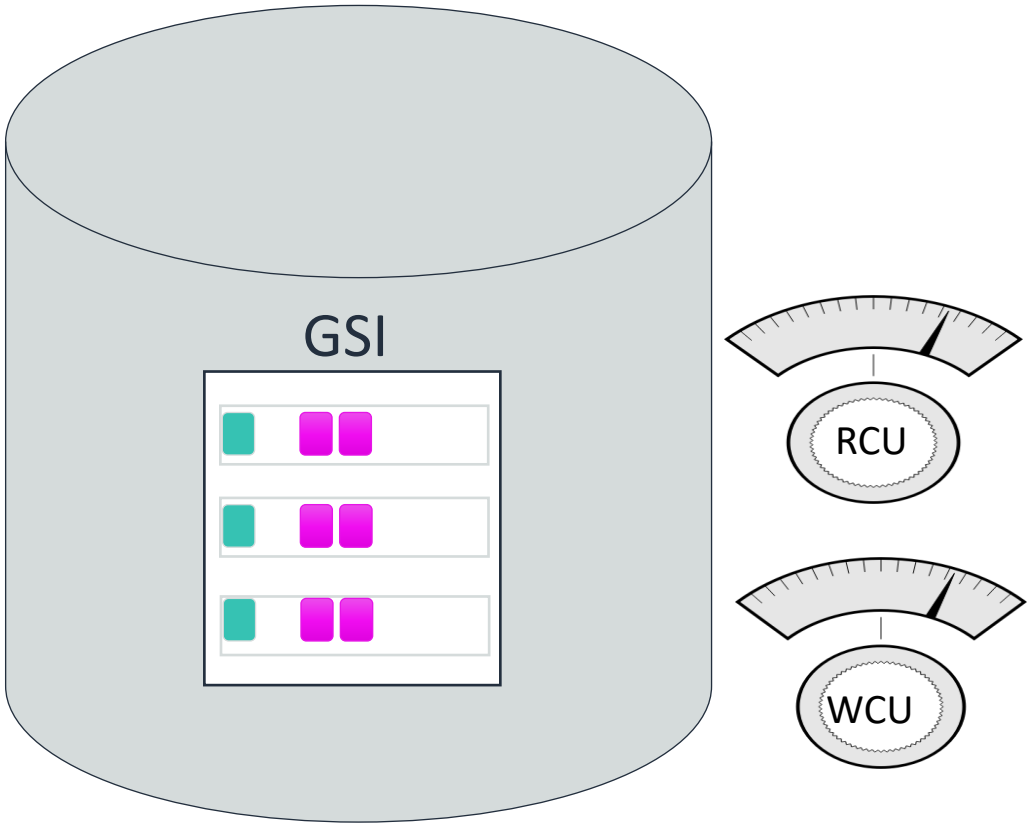
全局二级索引 - Global Secondary Index (GSI)

可以选择与表不同的分区键以及排序键
每个索引分区会对应所有的表分区



GSI的RCUs和WCUs是独立于表的容量而单独计算的

示例：全局二级索引



全局二级索引
分区键或分区和排序键

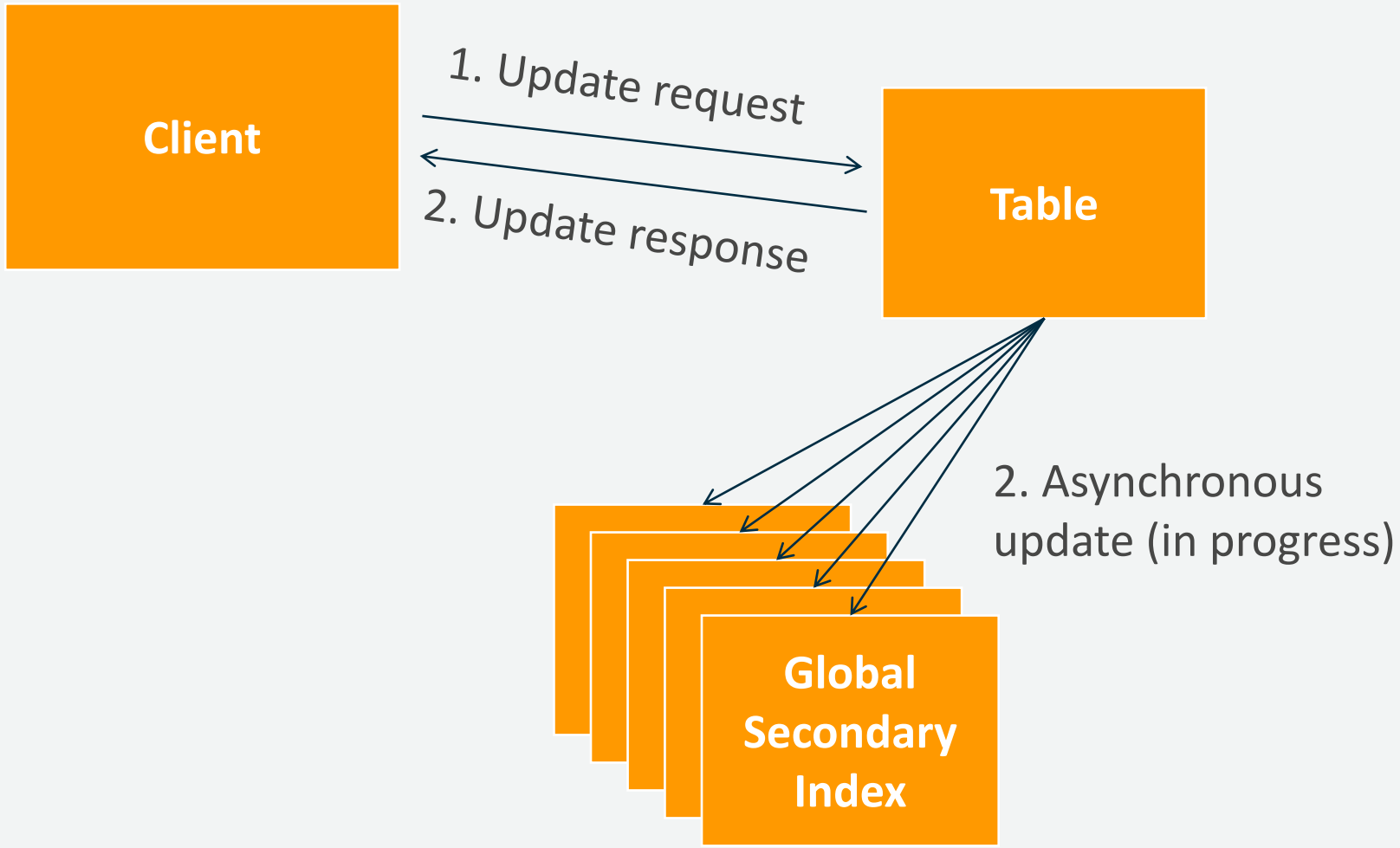
Destinations 表

TravelerId (分区键)	City	Date
1	Reno	2018/03/01
2	Boston	2018/04/15
3	Reno	2018/04/21

DestinationByCity 全局二级索引

City (分区键)	Date (排序键)	TravelerId
Reno	2018/03/01	1
Reno	2018/04/21	3
Boston	2018/04/15	2

GSI 是如何被更新的?



如果GSI没有足够的写容量，对表的写操作会被限流

选择 GSI 还是 LSI（*Local Secondary Index*）？

Global Secondary Index

- 索引的尺寸没有上限
- 读写容量和表是独立的
- 只支持最终一致性

Local Secondary Index

- 索引保存在表的分区中，每个分区键值的存储上限是10GB
- 使用的是表上定义的RCU和WCU
- 强一致性

DynamoDB 容量模式

亚马逊云科技

读取和写入吞吐量

读取容量单位 (RCU)：最大大小为 4KB 的项目的每秒强一致性读取次数

最终一致性读取使用了一半的预置读取容量。

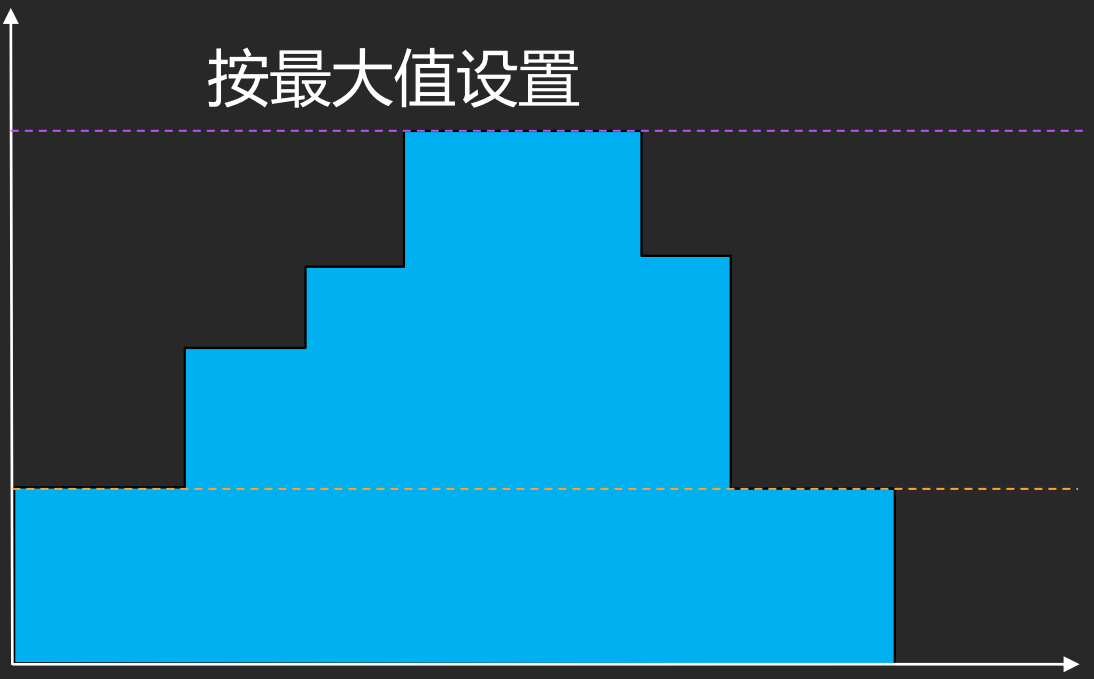
写入容量单位 (WCU)：每秒写入大小为 1KB 的项目的次数

注意：吞吐量在各分区之间平均分配。

选择合适的容量模式

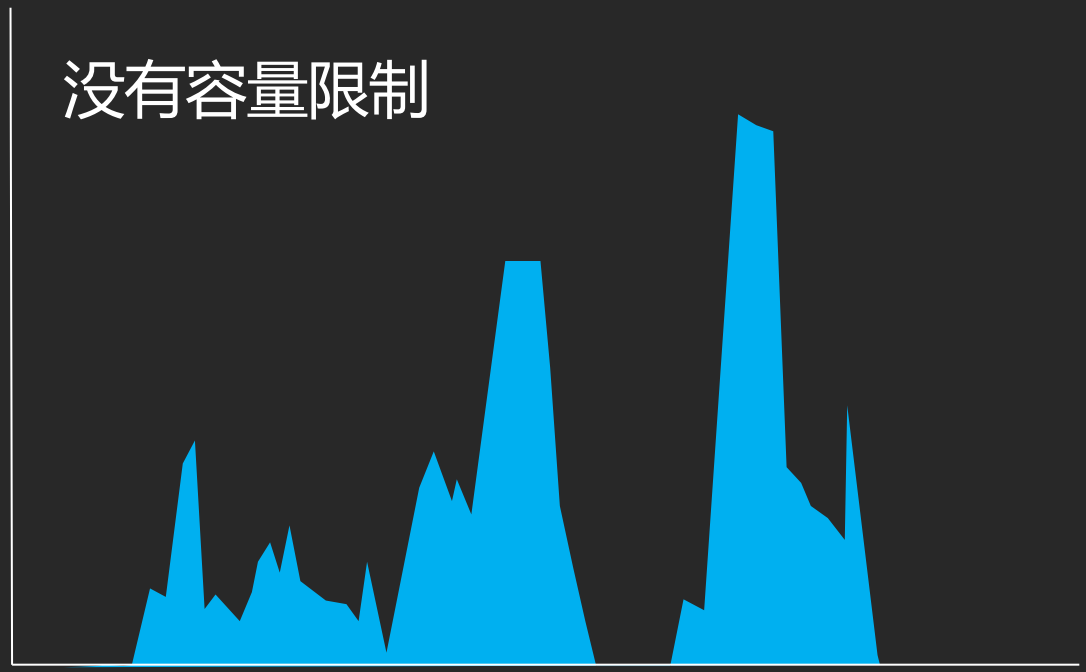
Provisioned Capacity

RCU/
WCU
level



按设置值收费，可调整

On-Demand



按每次请求收费

aws cost optimization

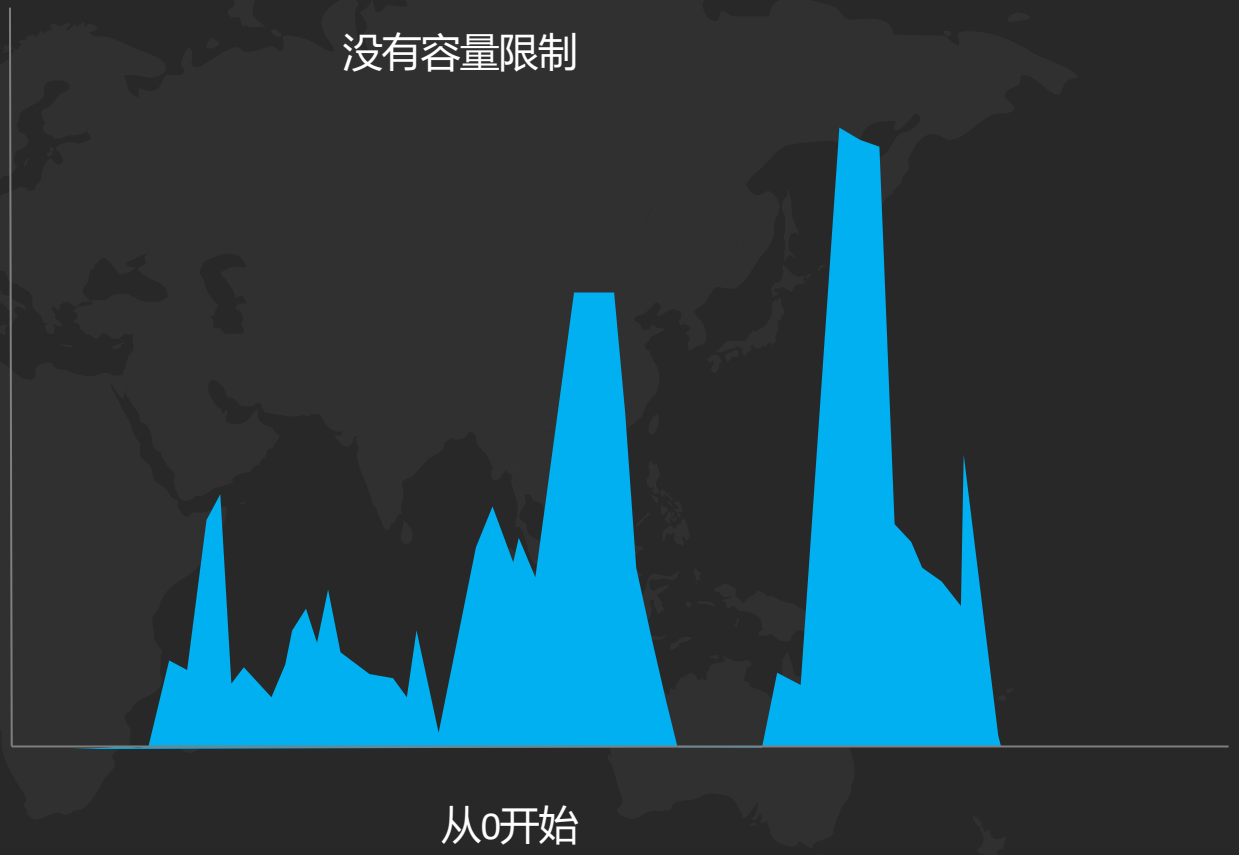
按需(On Demand)容量

特点:

- 不需要计划容量
- 只按照你发出的读和写付费

优点:

- 避免了过大的容量或者过小的容量
- 随着工作负载而自动调整



预置容量 Vs 按需容量

预置容量

- 稳定的工作负载
- 工作负载逐步变化
- 了解自己的流量模式
- 持续监控

按需容量

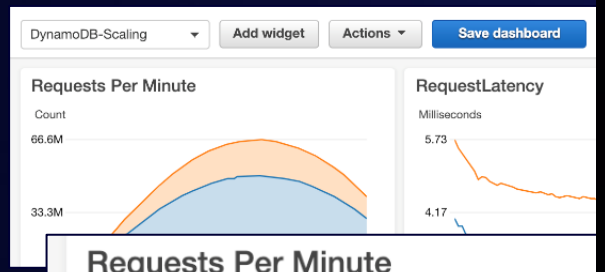
- 工作负载不可预知
- 工作负载经常会降到0
- 不了解自己的流量模式
- 没有运维

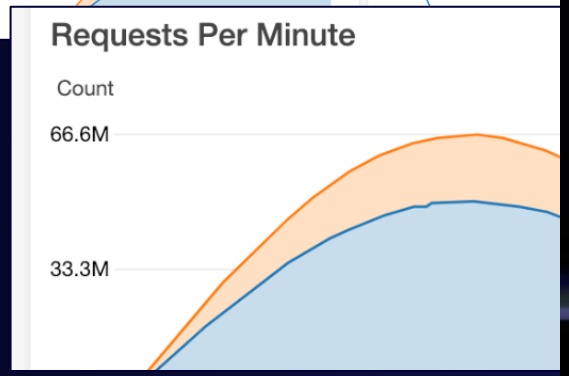
综合考虑成本，运维

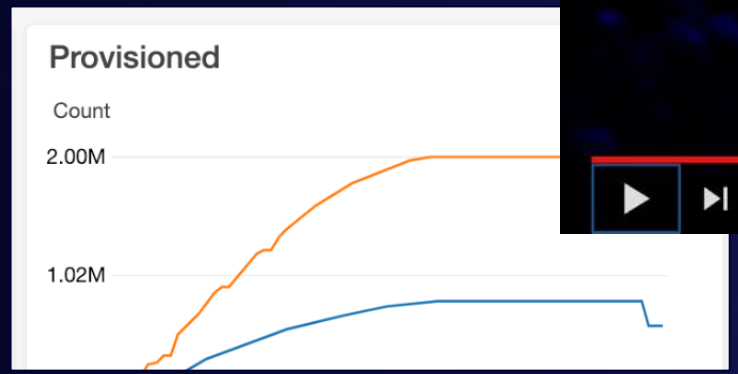
Amazon DynamoDB 极致弹性和底层设计

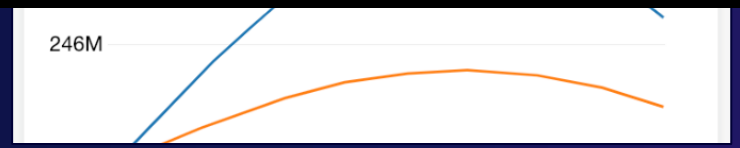
大规模高性能

PB级规模下的个位数毫秒

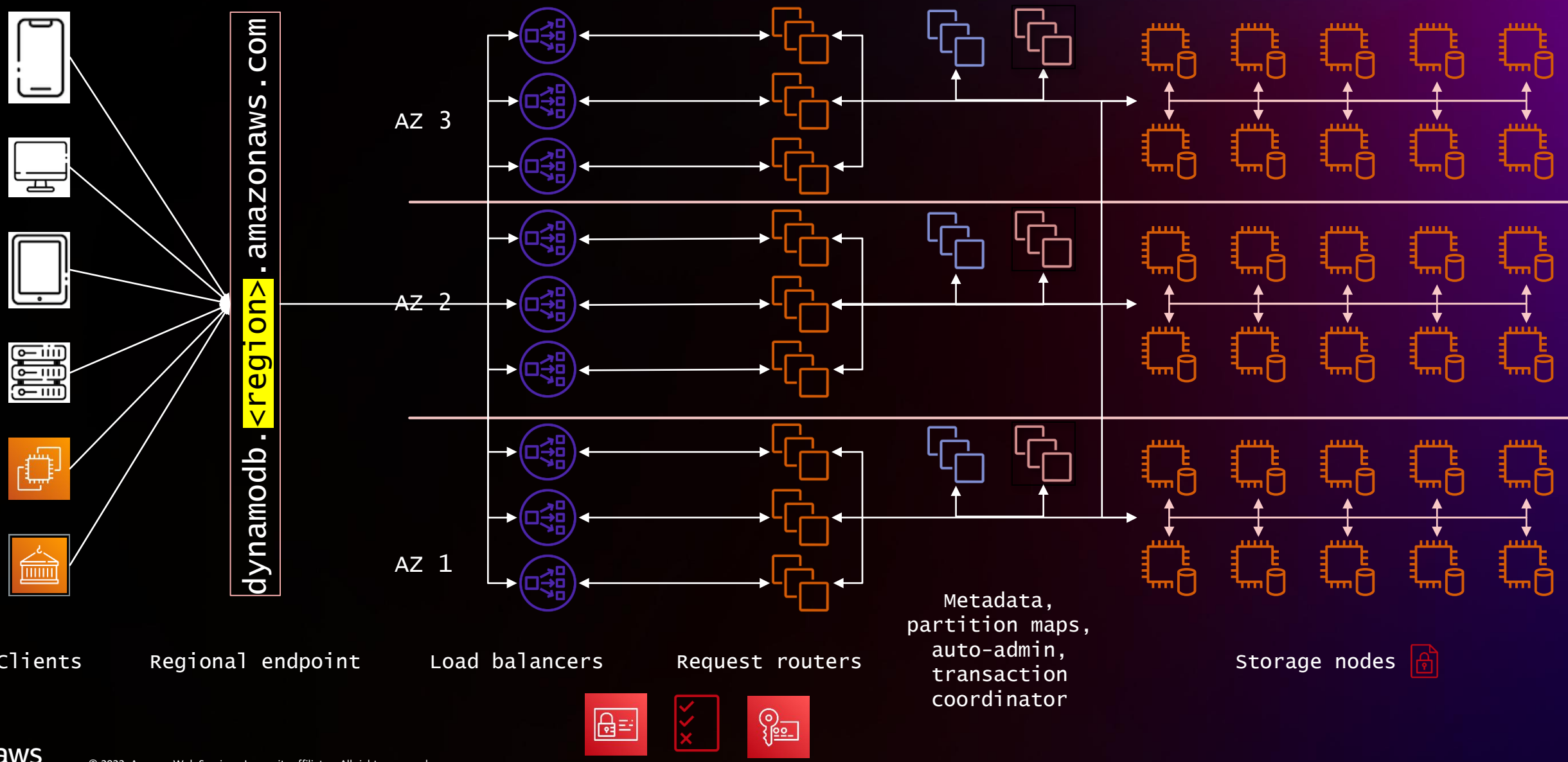




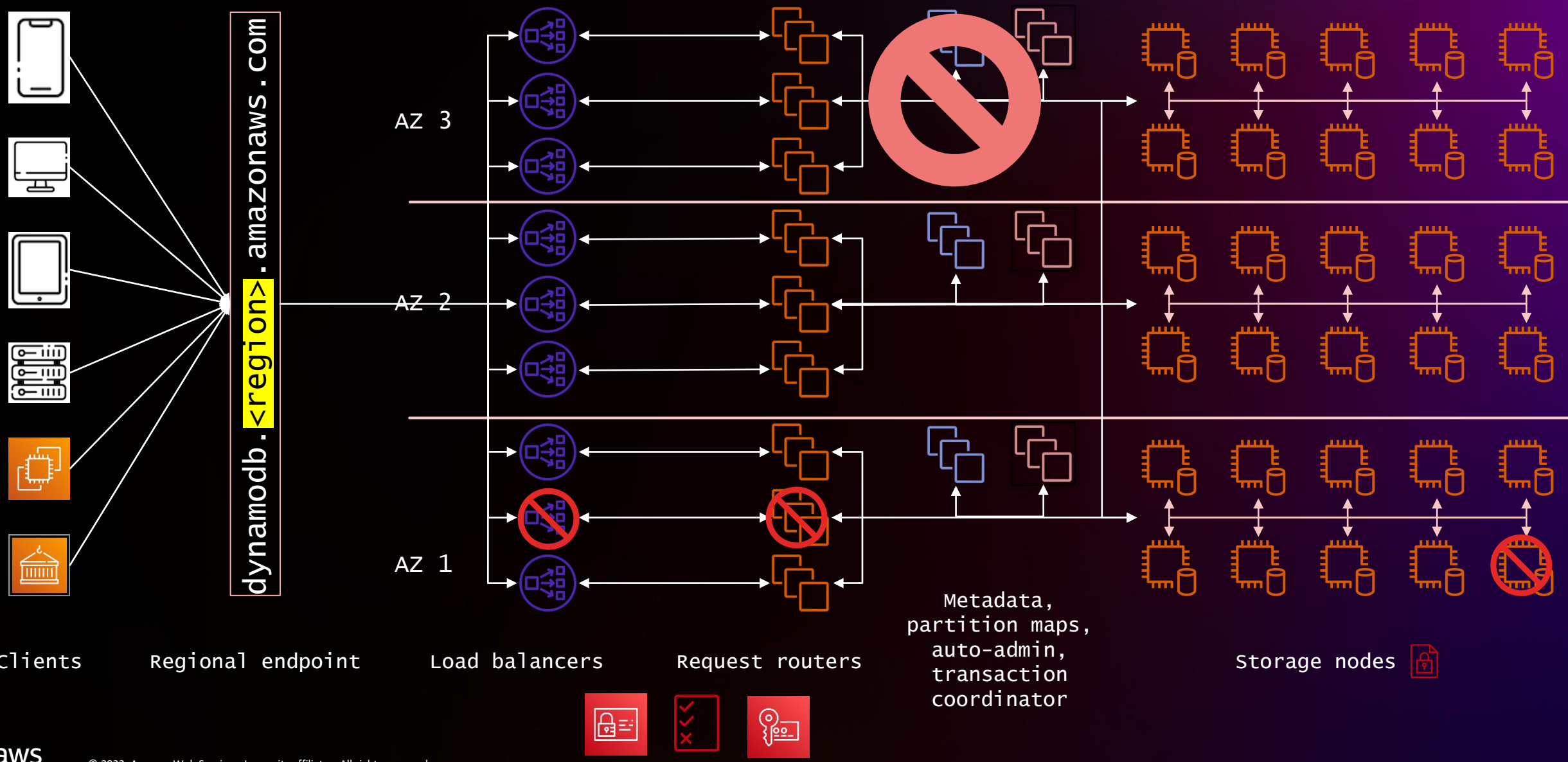




DynamoDB – High-level architecture



DynamoDB – High-level architecture

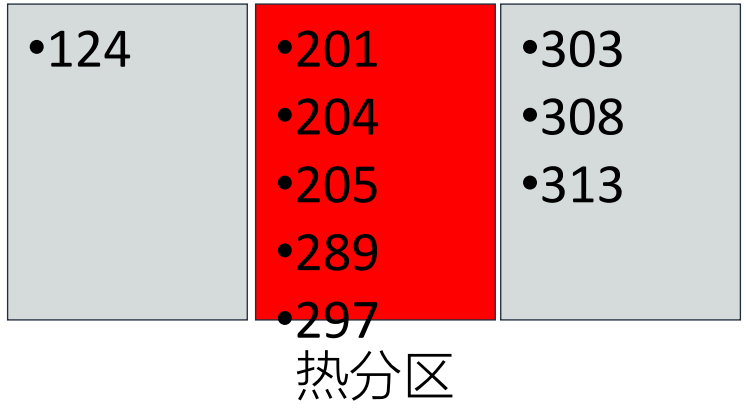


Amazon DynamoDB 设计最佳实践

最佳实践1: 慎重选择Hash Key以实现无限扩展

- 在分区键之间平均分配读写操作，以最大化吞吐量。

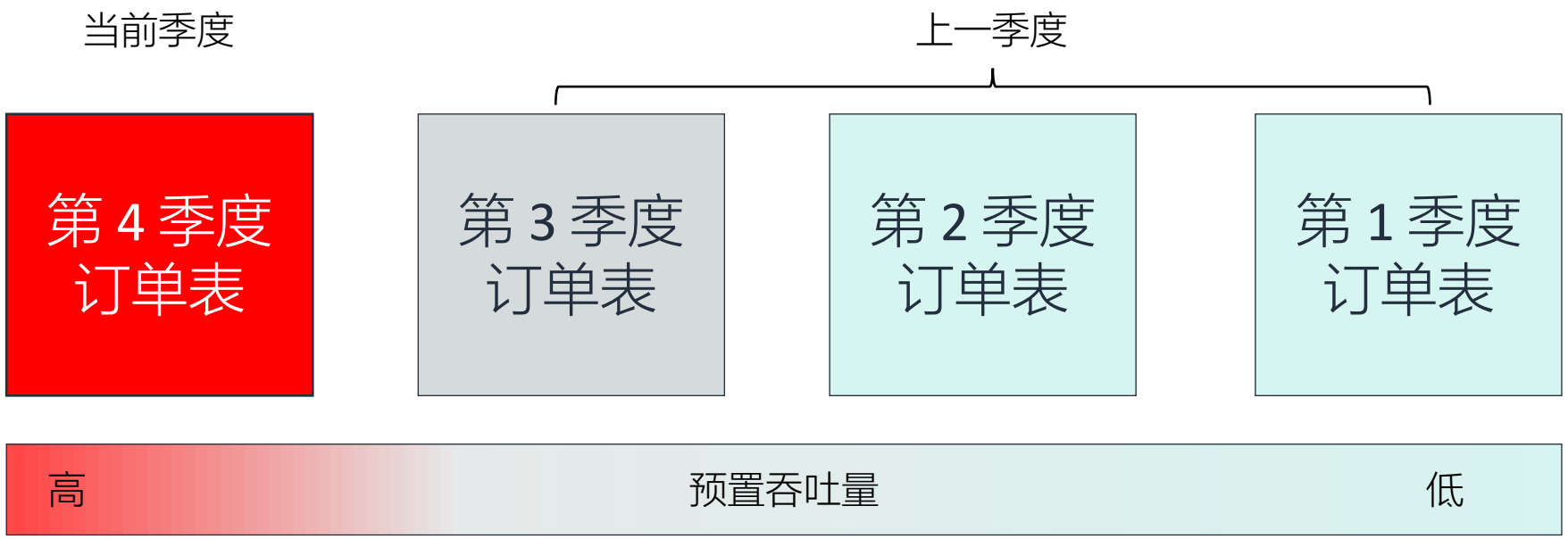
- 读/写操作仅限于少数分区。分区键的范围有限。



- 读/写操作分布在更多分区上。分区键包含计算的后缀。

最佳实践2: 区分表格存储时序型数据

- 将频繁访问的数据（热数据）与不频繁访问的数据（冷数据）分开



最佳实践3: 将大项目分开存储在另一张表

- 使用一对多表，而不是大量属性。

复杂论坛话题表

```
{
  "thread_id" : 123,
  "subject" : "How do I cook
potatoes?",
"replies"  :
    [ "Boil…", "Bake…", "Roast…",
"Fry…", "Mash…", "Chop…" ]
}
```

简单论坛话题表

```
{
  "thread_id" : 123,
  "subject" : "How do I cook
potatoes?"
}
```

回复表

```
{
  "thread_id" : 123,
  "reply_id" : "abc"
  "reply" : "Boil for 10
minutes"
}
{
  "thread_id" : 123,
  "reply_id" : "def"
  "reply" : "Bake for 45
minutes"
}
```

多种访问模式

- 将频繁访问的少量属性存储在单独的表中。

Company 表

```
{
  "companyName" : "Example",
  "stockPrice" : 285,
  "aboutCompany" : "Example company builds
   ships",
  "missionStatement": "Our mission is to
   build the best ships ….",
  "logoHighResolution": example.png"
 }
```

Company Stock 表

```
{
 "companyName" : "Example",
  "stockPrice": 285
}
```

Company 表

```
{
  "companyName" : "Example",
  "aboutCompany": "Example company
   builds ships",
  "missionStatement": Our mission is
   to build the best ships ….",
  "logoHighResolution": "example.png"
 }
```

本地二级索引

- 谨慎使用索引。
- 慎重选择投影。
  - 仅投影频繁请求的属性。
- 利用稀疏索引。

客户表

CustomerId	Customer Name	SportsNewsInterest
1	John	
2	Mary	Y
3	Tom	Y

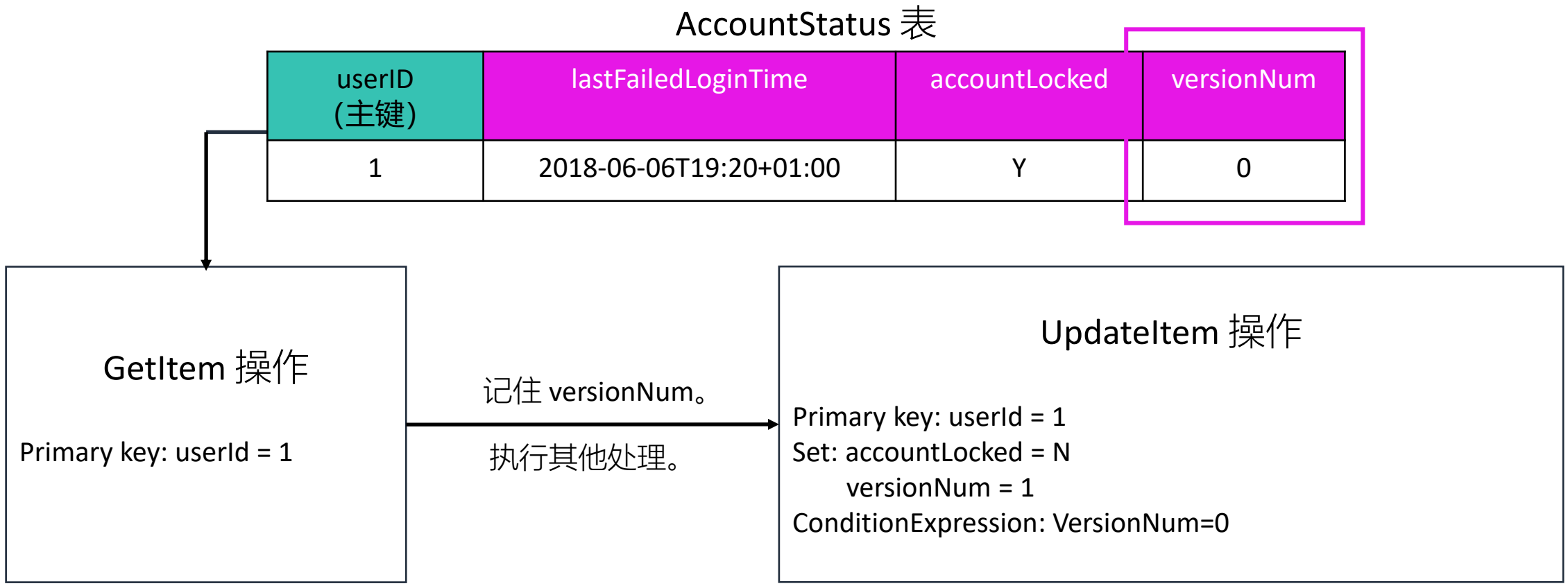
SportsInterest 索引

CustomerId	SportsNewsInterest
2	Y
3	Y

全局二级索引

- 选择可提供统一工作负载的键。
- 利用稀疏索引。
- 创建具有表属性子集的全局二级索引，以便快速查找。
- 用作最终一致性只读副本。

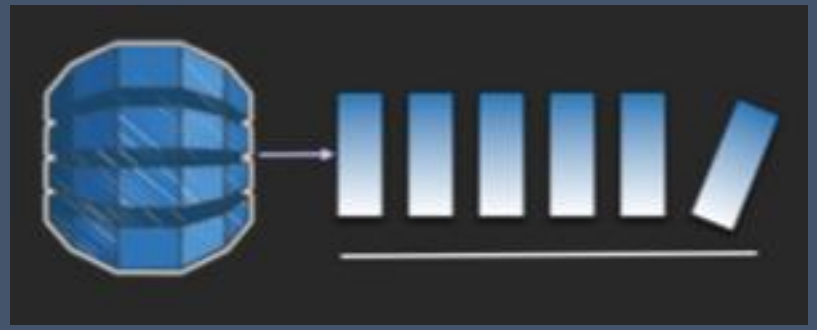
使用版本号的乐观锁


AccountStatus 表
userID
(主键)
lastFailedLoginTime
accountLocked
versionNum
1
2018-06-06T19:20+01:00
Y
0
GetItem 操作
Primary key: userId = 1
记住 versionNum。
执行其他处理。
UpdateItem 操作
Primary key: userId = 1
Set: accountLocked = N
versionNum = 1
ConditionExpression: VersionNum=0

DynamoDB Accelerator

Amazon DynamoDB 全球部署与服务集成

DynamoDB流

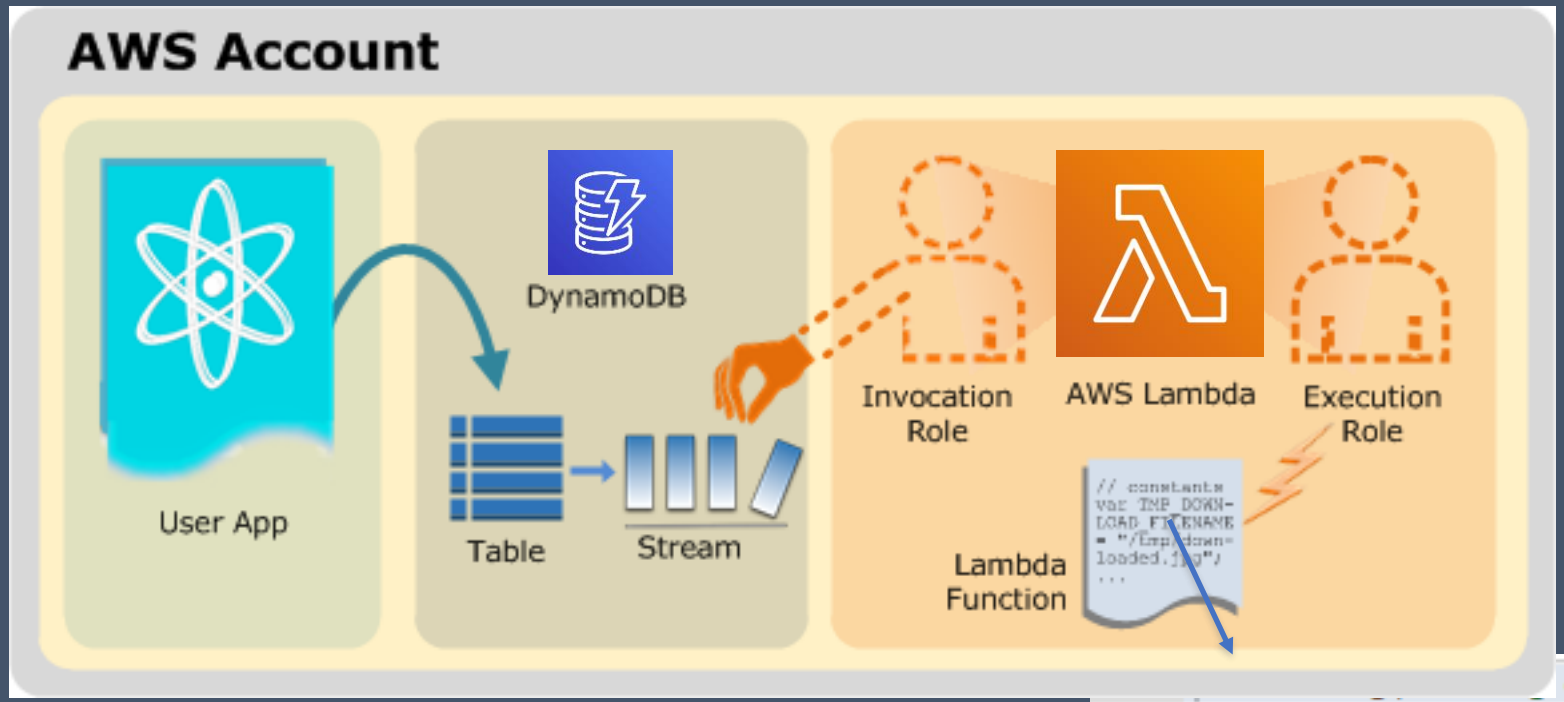


- DynamoDB流是针对表里的数据变化的顺序记录
- 在流里，针对表的变化的记录只会出现一次
- 在流里的记录的顺序与表上发生修改操作的顺序是一致的
- 流里的数据保存24小时，24小时以后自动删除
- 可以把表的数据被更新前的值以及更新后的值都写到流里
- 流里的数据通过API进行消费

DynamoDB流里记录的数据内容

- DynamoDB流可以配置为四种写入的数据内容：
- KEYS_ONLY – 只有分区键和排序键的数据被写入流
- NEW_IMAGE – 修改后的整条记录都被写入流
- OLD_IAMGE – 修改前的整条记录都被写入流
- NEW_AND_OLD_IMAGES – 修改前和修改后的整条记录都被写入流

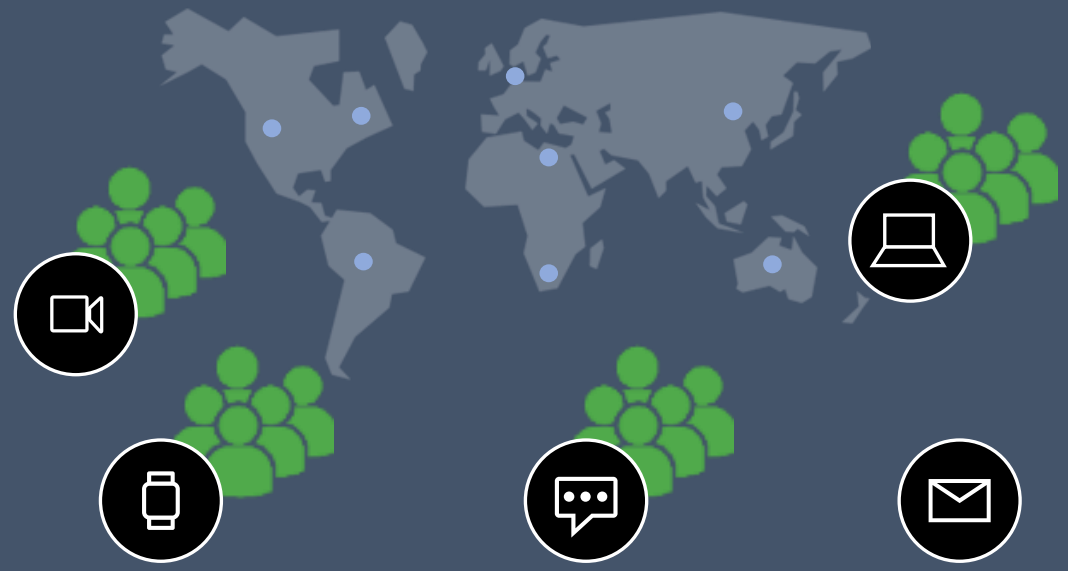
DynamoDB流和AWS Lambda触发器举例




```
event');
exports.handler = function(event, context) {
  console.log("Event: %j", event);
  for(i = 0; i < event.Records.length; ++i) {
    record = event.Records[i];
    console.log(record.EventID);
    console.log(record.EventName);
    console.log("DynamoDB Record: %j", record.Dynamodb);
  }
  context.done(null, "Hello World");  // SUCCESS with message
}
```

```
2015-03-21T07:44:58.883Z 2ca3769a-cf9e-11e4-b270-ad4d24b312ff INSERT
2015-03-21T07:44:58.883Z 2ca3769a-cf9e-11e4-b270-ad4d24b312ff DynamoDB Record:{ "NewImage": { "name": { "S": "sivar" }, "hk": { "S": "3" } }, "SizeBytes": 15, "StreamViewType": "NEW_AND_OLD_IMAG
2015-03-21T07:44:58.883Z 2ca3769a-cf9e-11e4-b270-ad4d24b312ff Message: "Hello World"
```

全球扩展的应用系统的特征

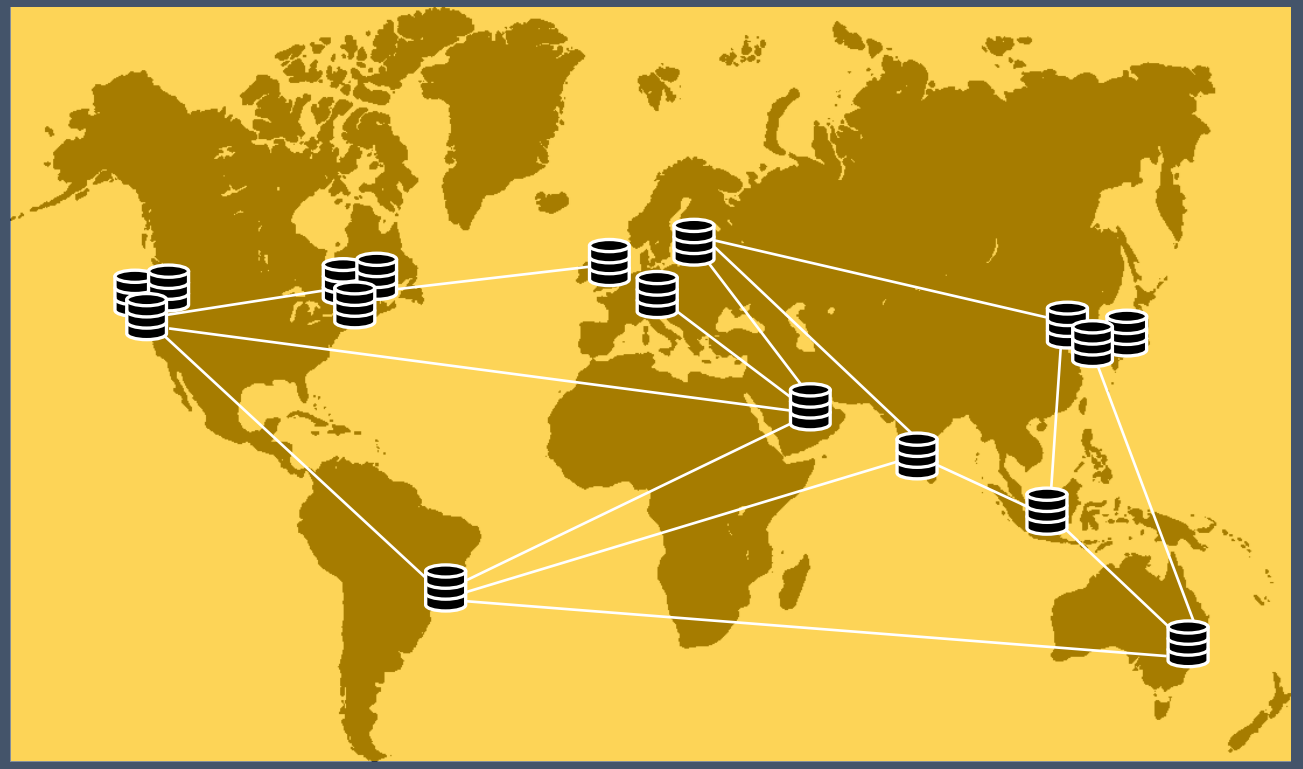


关系型 | 键值型 | 文档型 | 图表

用户规模	超过1百万
数据量	TB, PB, EB
访问用户	全球
性能	Milliseconds, microseconds
每秒请求数	百万
应用端	Mobile, IoT, devices
伸缩性	向上或者向下伸缩
成本	按照用量付费
开发方式	API访问

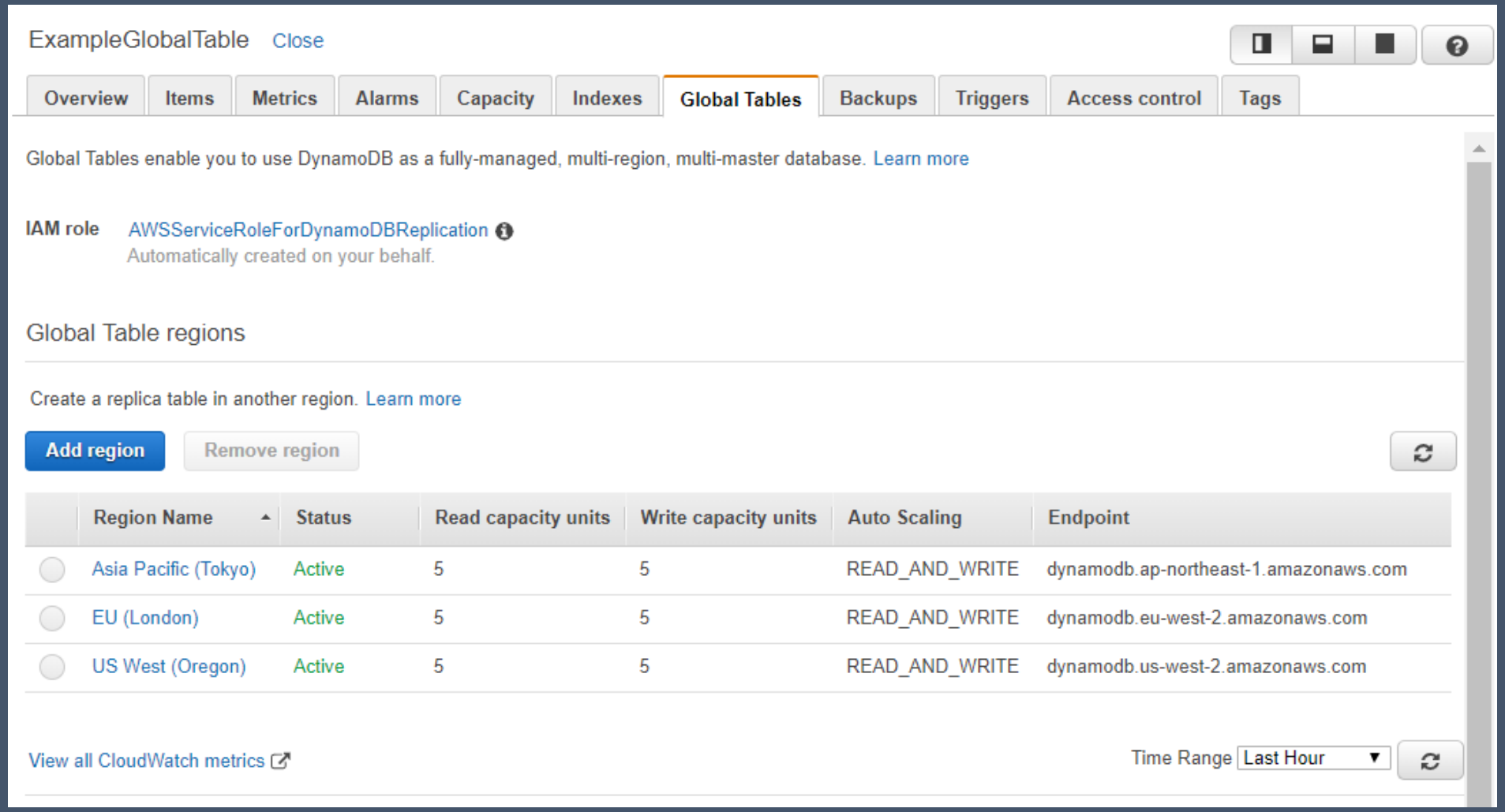
全局表(Global Tables)

第一个全托管的，多主，多区域的数据库

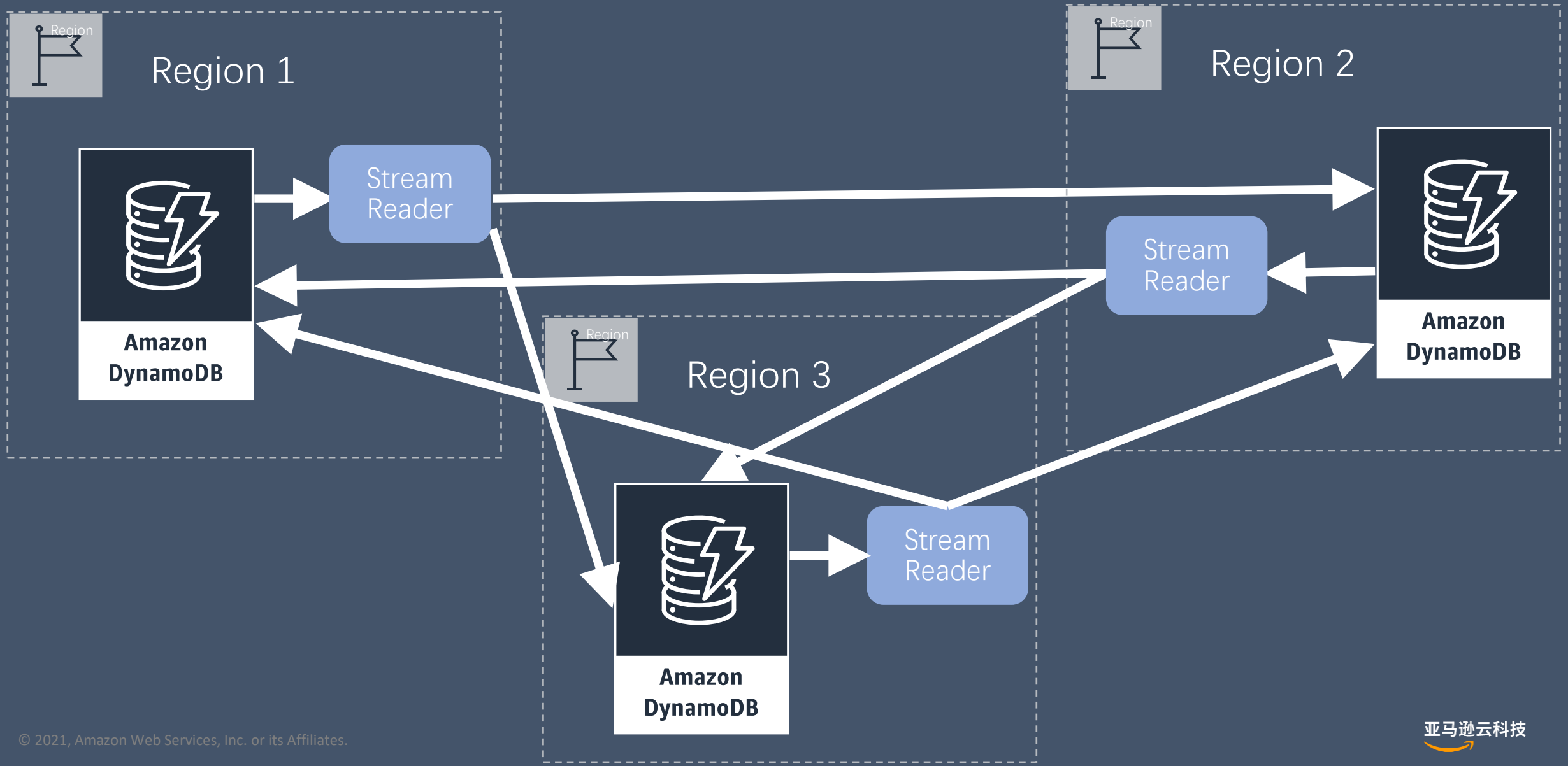


- 用于构建高性能的，全球分布的应用系统
- 对本区域的表的低延迟读取需求
- 多区域冗余
- 配置简单，无需应用程序多次写

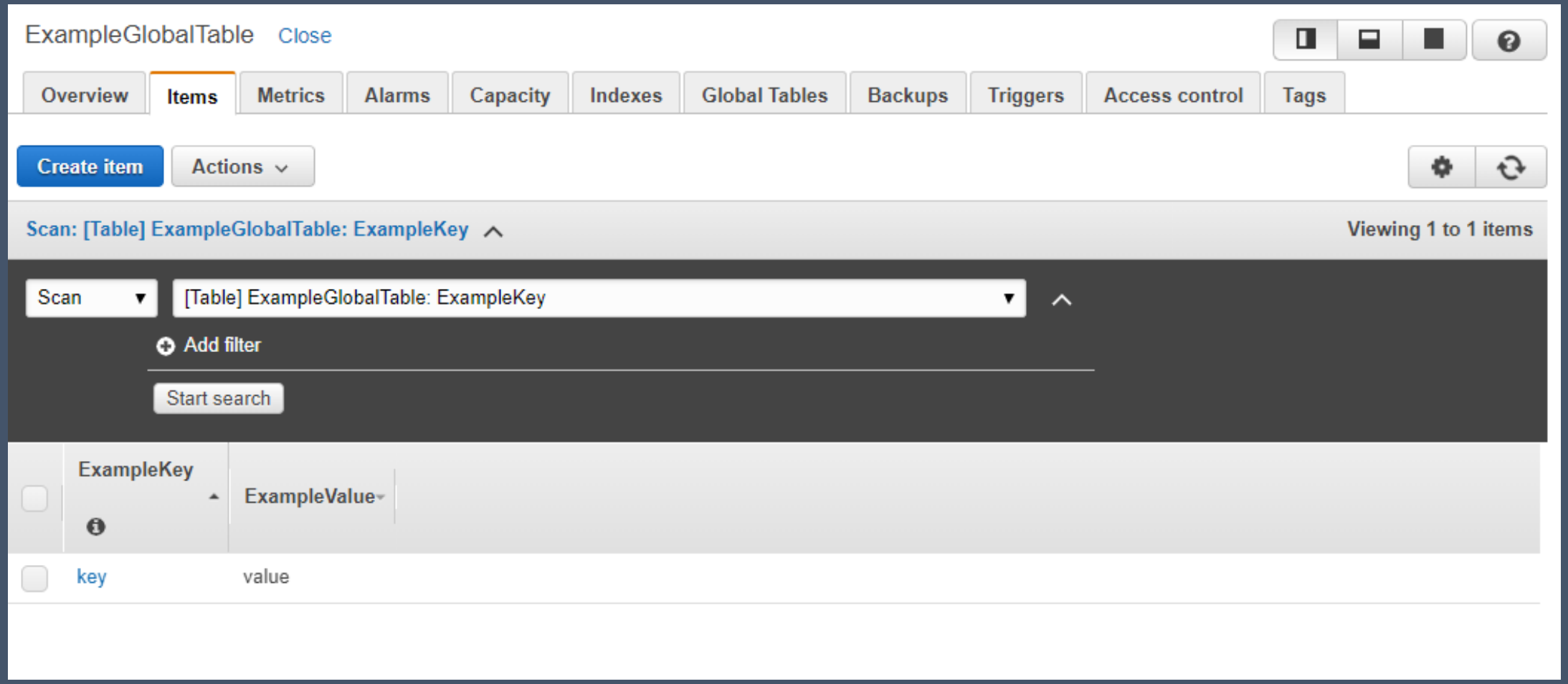
全局表(Global Tables)



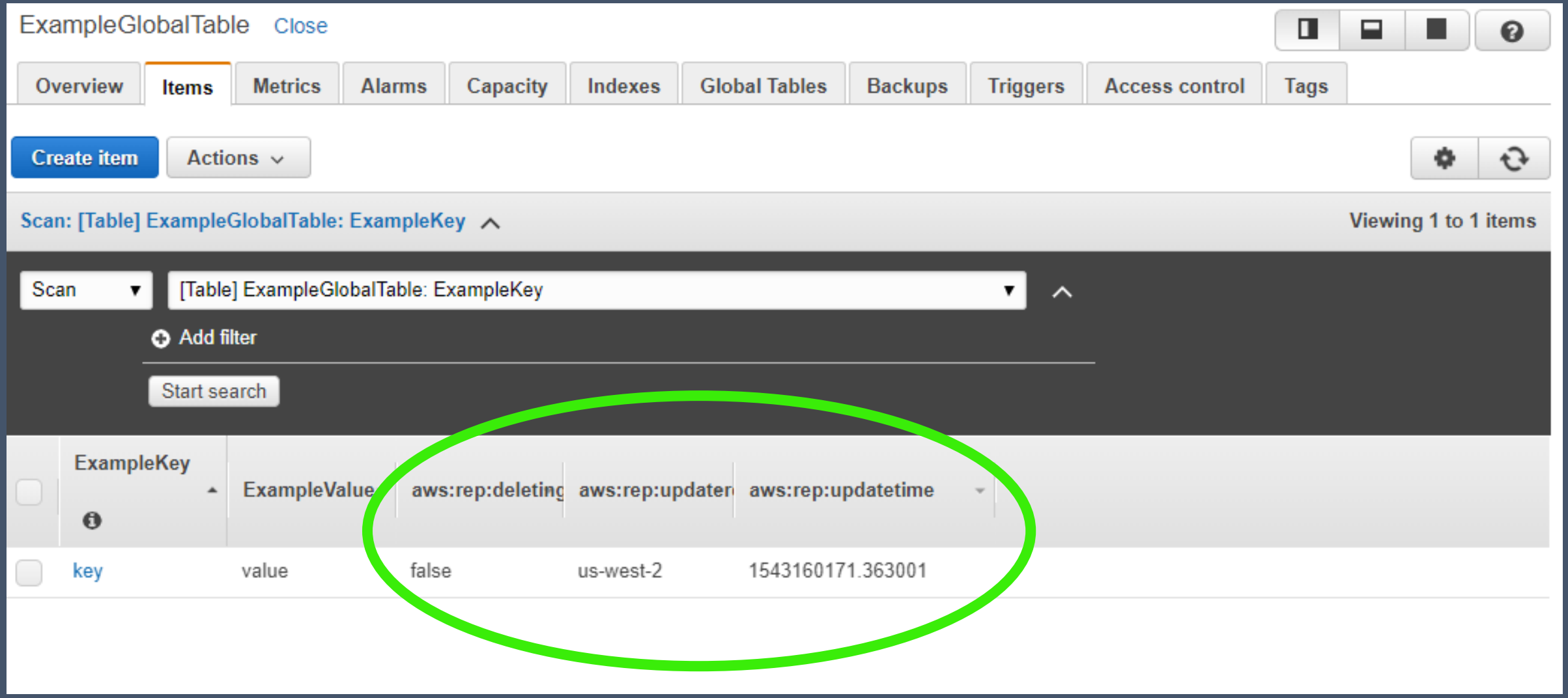
多区域复制



插入数据(PutItem)



数据同步以后

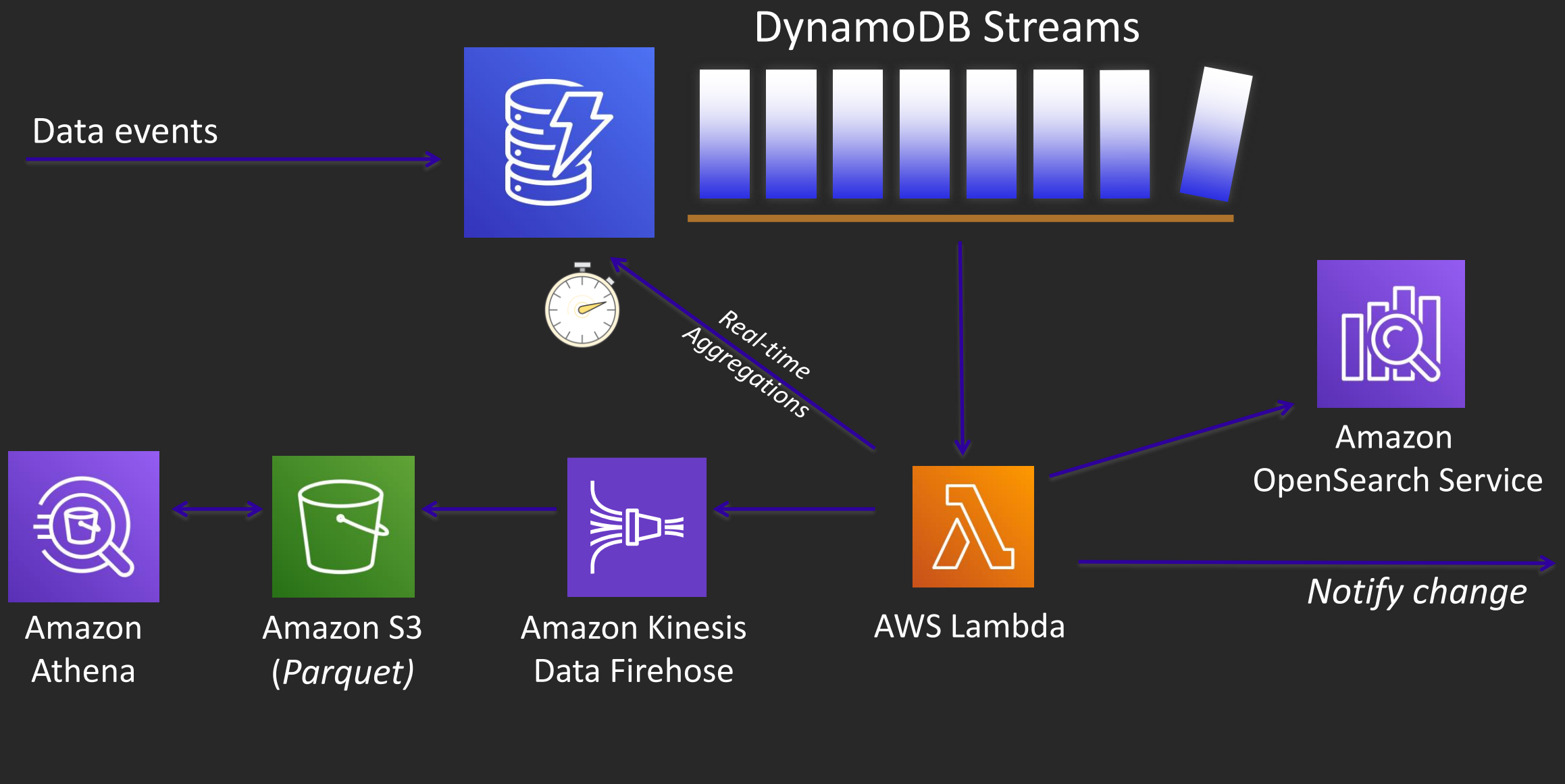


冲突处理机制

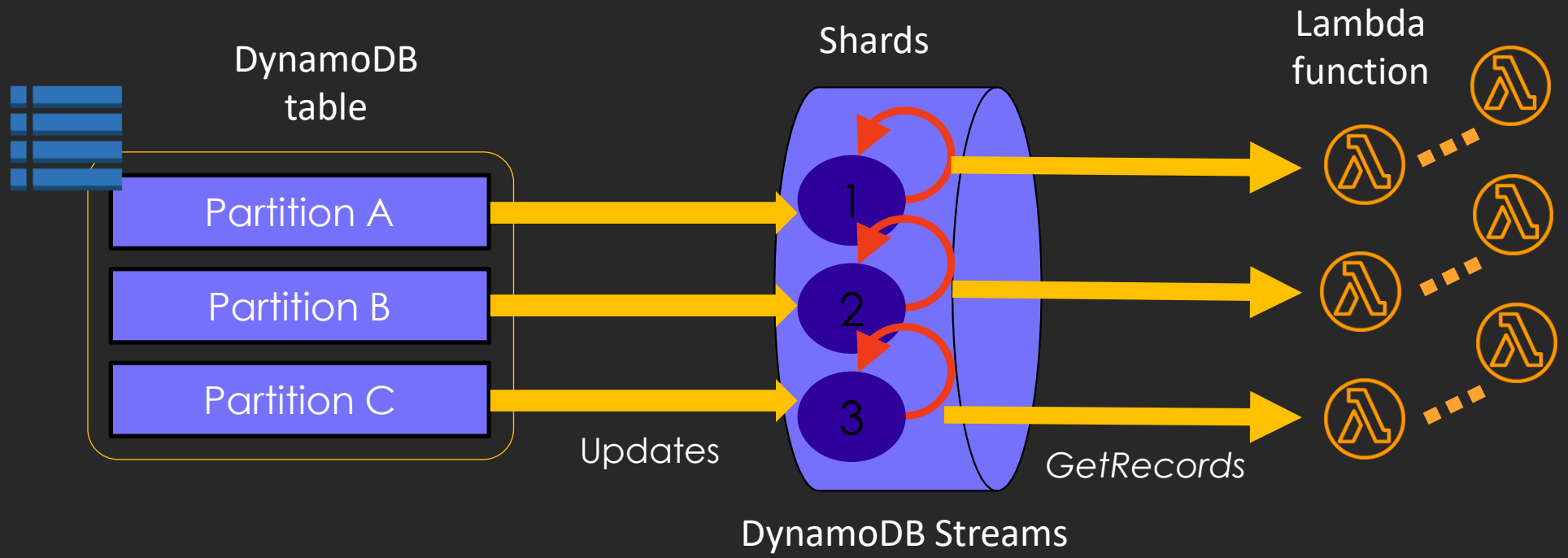
- 大部分情况下数据在区域之间的同步时间为秒级别
- 如果多个区域对同一条记录进行修改，则以最后一个修改请求为主
- 需要为表

复杂查询的处理

Serverless 和事件驱动架构

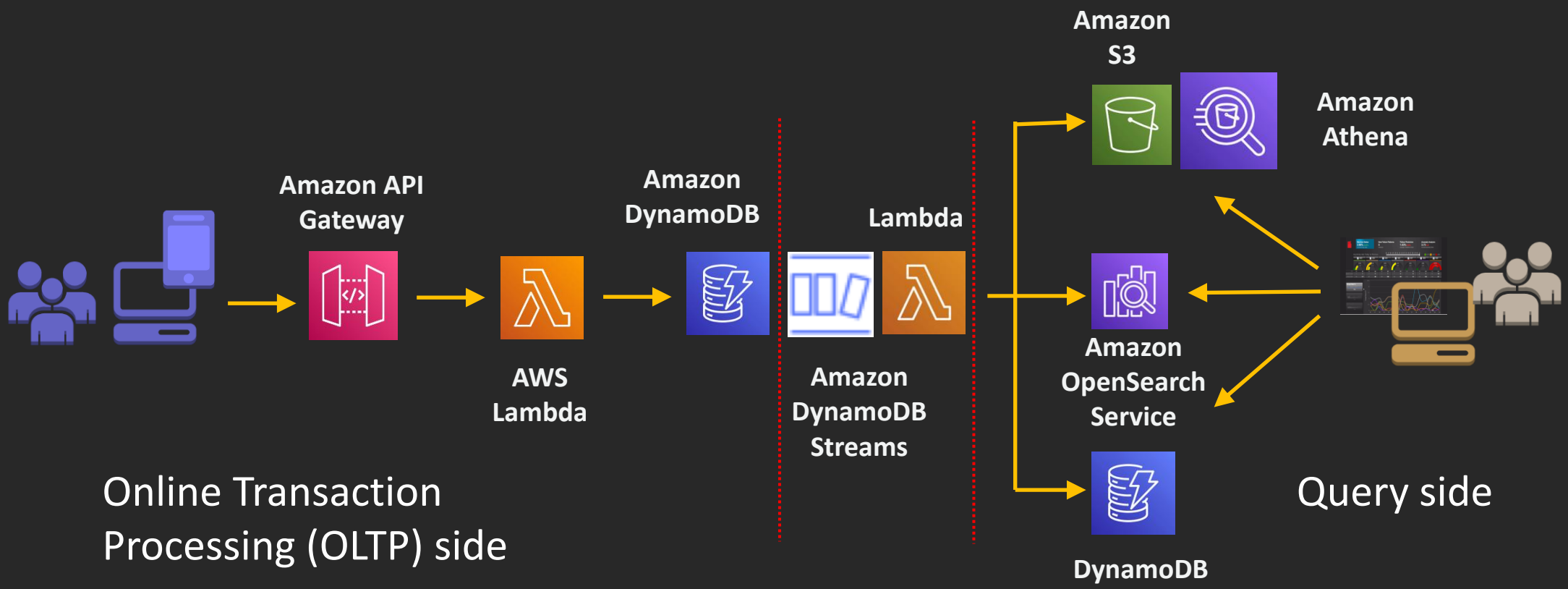


DynamoDB Streams and Lambda



DynamoDB Streams + AWS Lambda = 可靠的“至少一次”事件传递

Serverless 应用示例



关键要点

NoSQL 非常适合 OLTP

NoSQL 并不意味着没有关系（ERD 仍然很重要）

数据建模（为查询而组织设计表结构）

Serverless 和事件驱动架构实现更多云原生应用

选择正确的DynamoDB 容量模式 – 经济又高效


openGauss
OceanBase
ArkDB
RASESQL
HotDB
StellarDB
QianBase xTP
云树Shard
GoldenDB
DolphinDB
MatrixDB
DynamoDB
SinoDB
FastData
Galaxybase
KunDB
SQL Server
vertica
D B 2
GBase
Oracle
达梦数据库
Spacture
开务数据库
UbiSQL
神舟通用
TDSQL
TiDB
Tapdata
StarRocks
KingbaseES
2010
2014
2018

THANKS