Amazon DynamoDB

Yuko Mori
Solutions Architect
Amazon Data Services Japan K.K.

amazon web services

自己紹介

- 森　祐孝(もり　ゆうこう)
  - アマゾン データサービスジャパン株式会社
  - ソリューションアーキテクト

- 経歴
  - Sierなどで、アプリケーション開発、PL、PMなどを担当
  - ゲーム会社（テクニカルディレクタ）
    - ブラウザソーシャルゲーム、スマートフォン向けソーシャルゲーム
- 担当
  - ソーシャルゲーム、コンソールゲーム系のお客様のAWS構築支援

amazon web services

Agenda

- DyanamoDBとは
- Table, API, Data Type
- Indexes
- Scaring
- ユースケース& ベストプラクティス
- DynamoDB Streams

amazon web services

DyanamoDBとは

amazon web services

Amazon DynamoDBの生い立ち

- Amazon.comではかつて全てのアクセスパターンをRDBMSで処理していた
- RDBMSのスケールの限界を超えるため開発されたDynamoが祖先

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

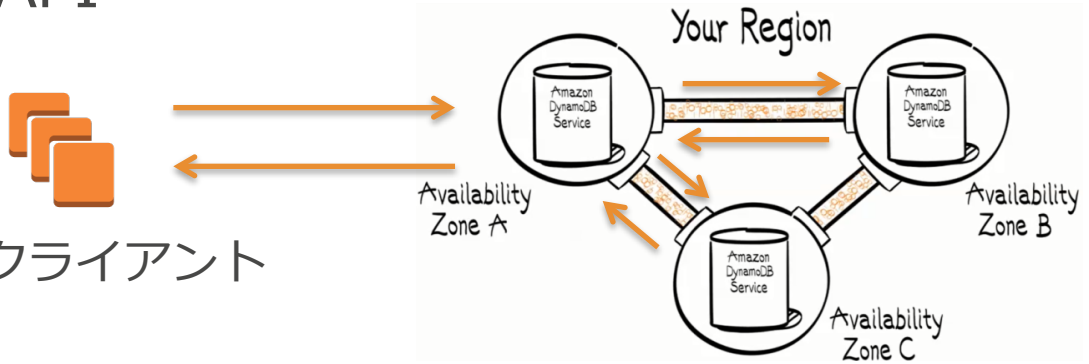
One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

- 結果整合性モデル採用による可用性向上
- HWを追加する毎に性能が向上するスケーラビリティ
- シンプルなクエリモデルによる予測可能な性能

Amazon DynamoDBの特徴

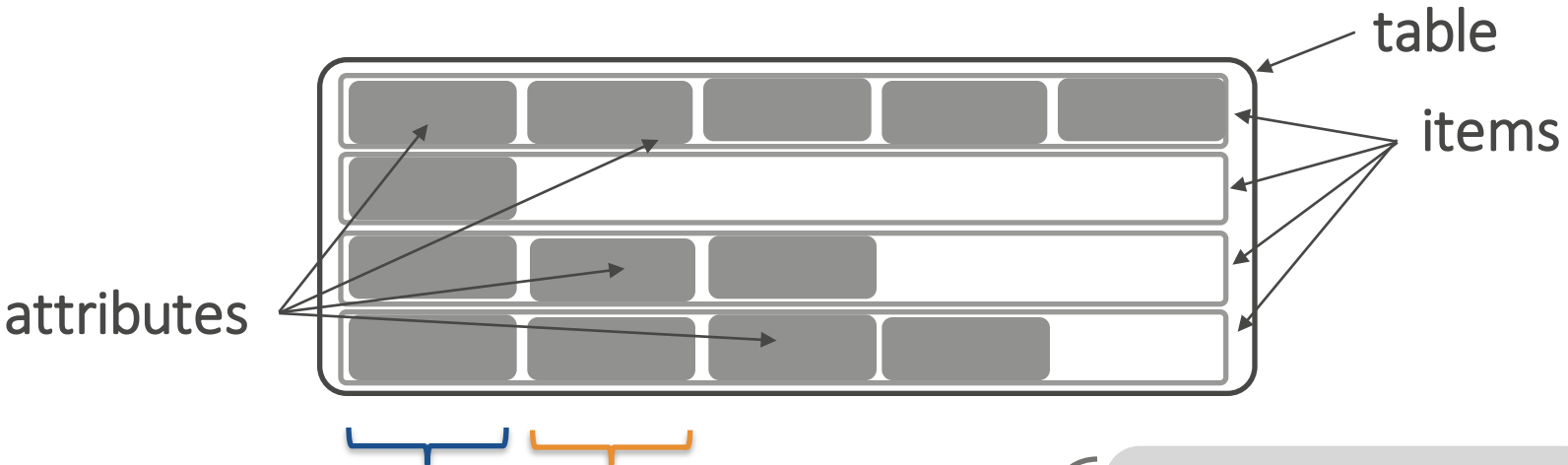
- 完全マネージド型の NoSQL データベースサービス
  - ストレージの容量制限がない
  - 運用管理必要なし
- ハイスケーラブル、低レイテンシー
- 高可用性– 3x レプリケーション
- シンプル且つパワフルAPI



Tables, API, Data Types

amazon web services

Table



必須
キーバリュー型のアクセスパターン
データ分散に利用される

オプション
1:Nモデルのリレーションシップ
豊富なQueryをサポート

ハッシュキー検索用
==, <, >, >=, <=
“begins with”
“between”
sorted results
counts
先頭/末尾 N件
ページ単位出力

Table API

- CreateTable
- UpdateTable
- DeleteTable
- DescribeTable
- ListTables

- Query
- Scan

- BatchGetItem
- BatchWriteItem

- GetItem
- PutItem
- UpdateItem
- DeleteItem

DynamoDB

In preview

Streams API

- Liststreams
- DescribeStream
- GetShardIterator
- GetRecords

amazon web services

AWS SDKs and CLI

- 各種言語むけのオフィシャルSDKやCLIを利用


Java


Python


PHP


.NET

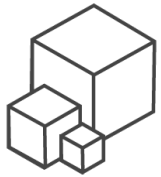

Ruby


nodeJS


Javascript
in the Browser


iOS


Android


AWS CLI

Data Types

- String (S)
- Number (N)
- Binary (B)

- String Set (SS)
- Number Set (NS)
- Binary Set (BS)

- Boolean (BOOL)
- Null (NULL)
- List (L)
- Map (M)

JSON用に定義

amazon web services

Documentデータ型 (JSON)

- データタイプ (M, L, BOOL, NULL) としてJSONをサポート
- Document SDKs
  - 単純なプログラミングモデル
  - JSONから、JSONへの変換
  - Java, JavaScript, Ruby, .NET

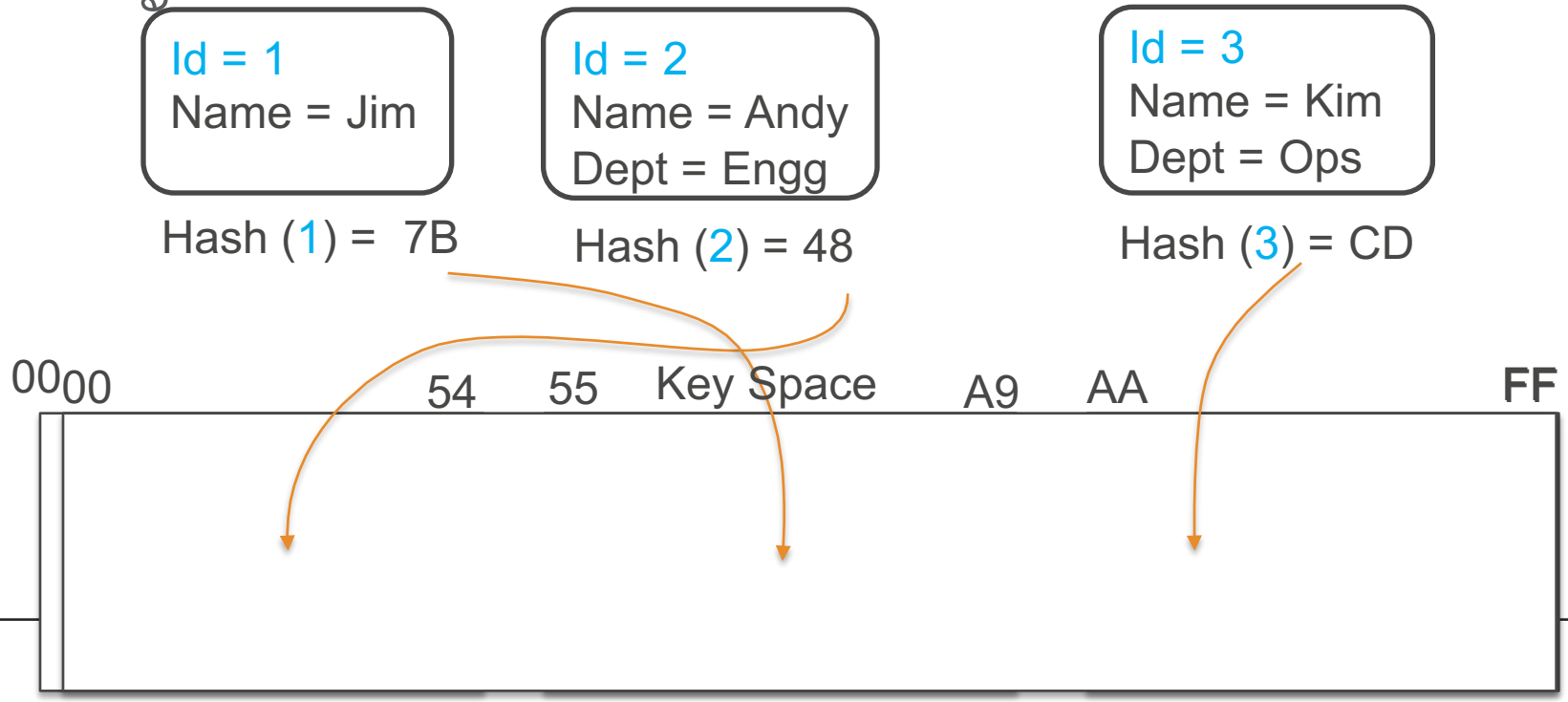
Javascript	DynamoDB
string	S
number	N
boolean	BOOL
null	NULL
array	L
object	M

```
var image = { // JSON Object for an image
    imageid: 12345,
    url: 'http://example.com/awesome_image.jpg'
};
var params = {
    TableName: 'images',
    Item: image, // JSON Object to store
};
dynamodb.putItem(params, function(err, data){
    // response handler
};
```

amazon web services

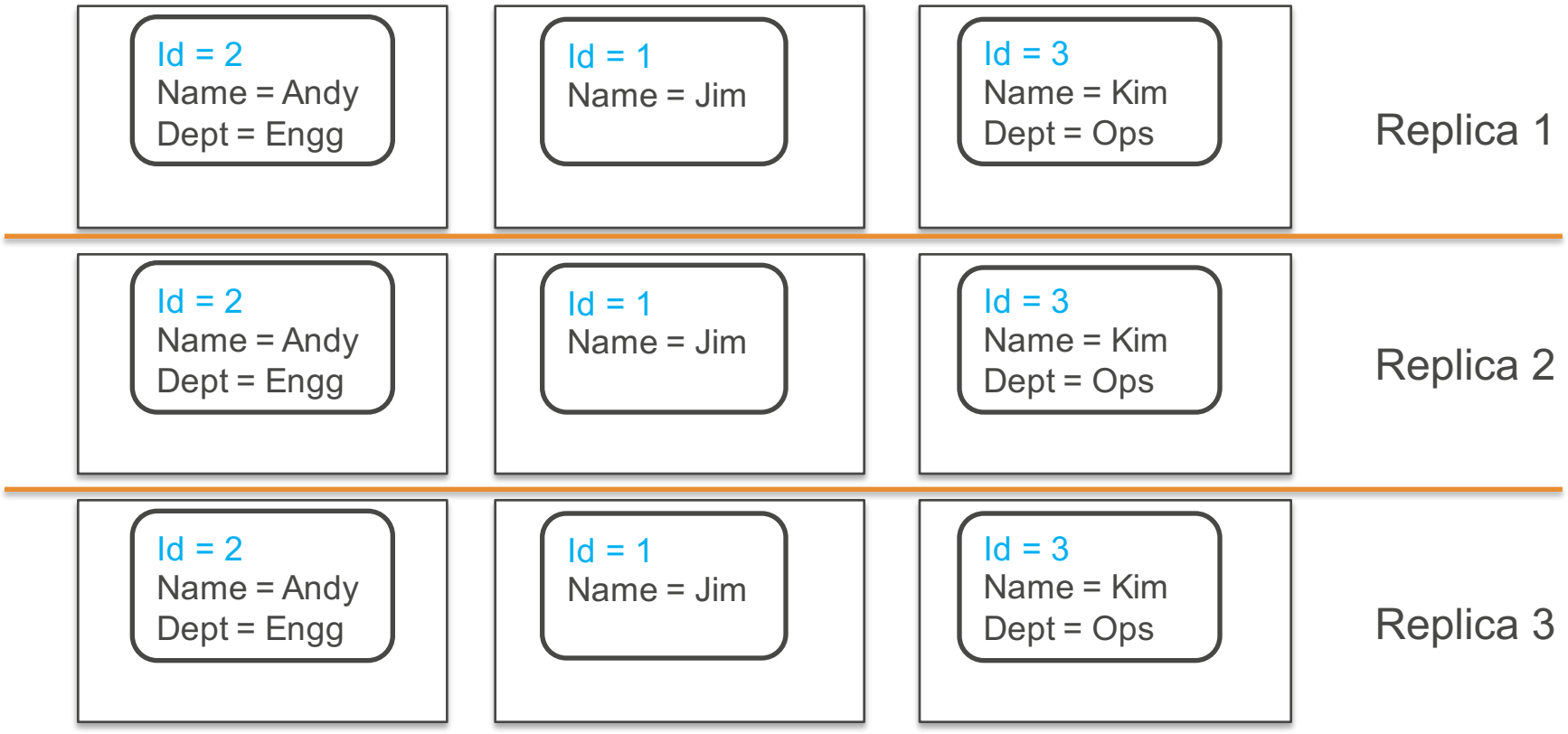
Hash Table

- Hash key は単体でプライマリキーとして利用
- 順序を指定しないハッシュインデックスを構築するためのキー
- テーブルは、性能を確保するために分割(パーティショニング)される場合がある



amazon web services

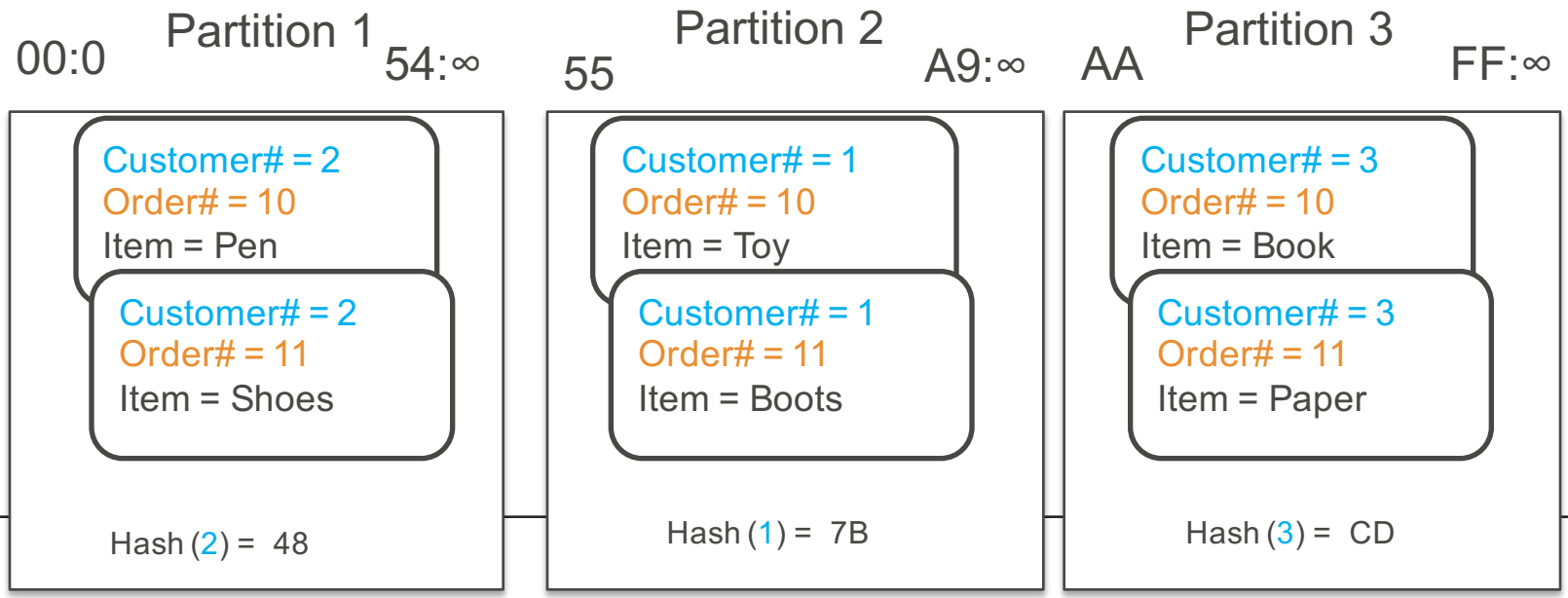
データは3箇所にレプリケーション



amazon web services

Hash-Range Table

- Hash + Rangeでプライマリキーとすることもできる
- Hash keyに該当する複数のデータの順序を保証するためにRange keyが使われる
- Hash Keyの数に上限はありません
  （Local Secondary Indexesを使用時は上限あり）



amazon web services

DynamoDBの整合性モデル

- Write
  - 少なくとも２つのレプリカでの書き込み完了が確認とれた時点でAck
- Read
  - デフォルト
    - 結果整合性のある読み込み
    - 最新の書き込み結果が反映されない可能性がある
  - Consistent Readオプションを付けたリクエスト
    - 強い整合性のある読み込み
    - Readリクエストを受け取る前までのWriteがすべて反映されたレスポンスを保証

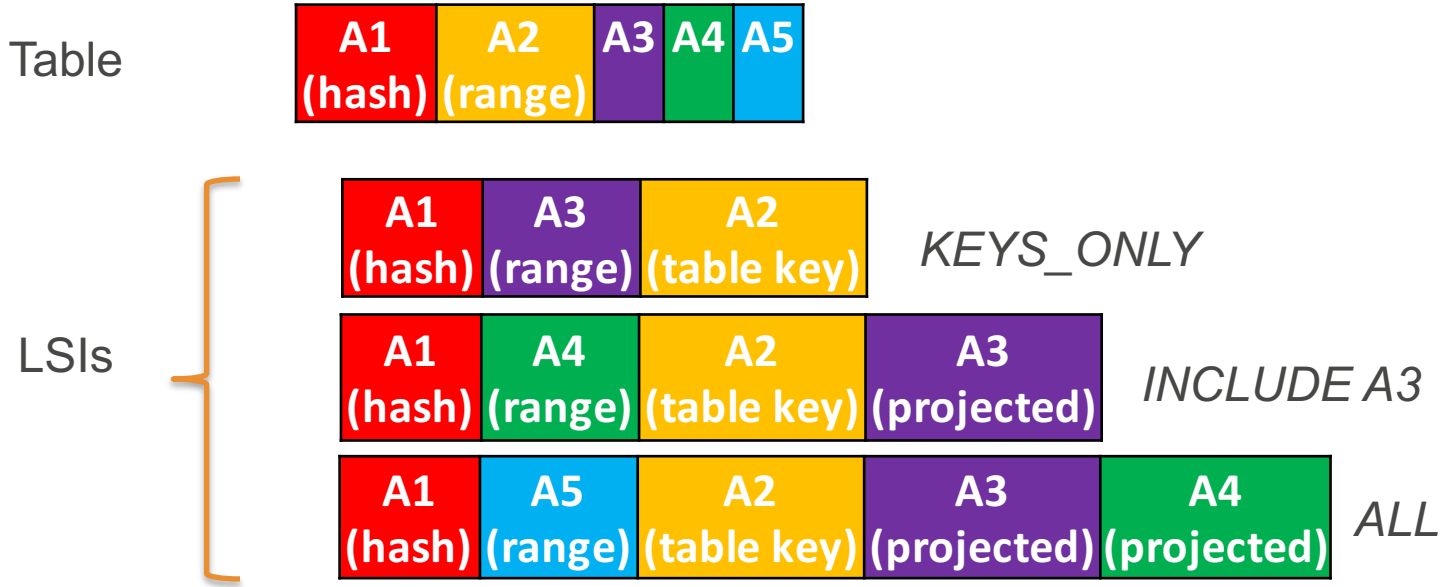
amazon web services

Indexes

amazon web services

Local Secondary Index (LSI)

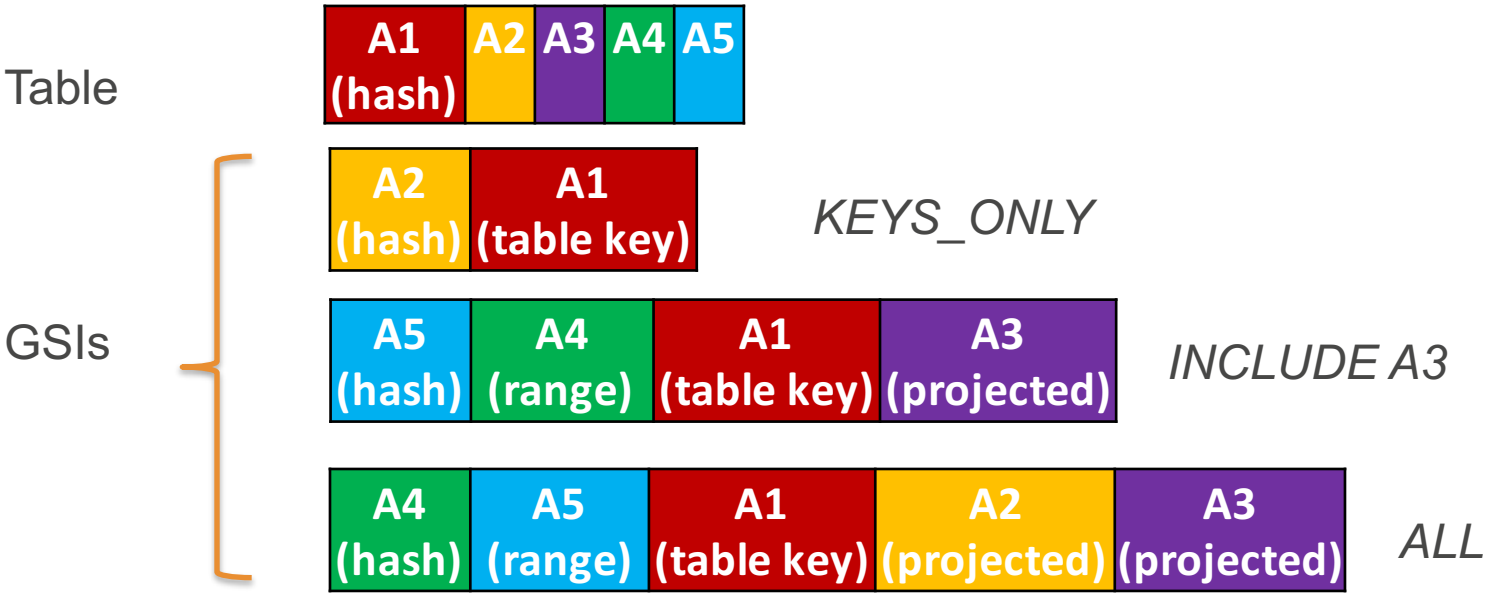
- Range key以外に絞り込み検索を行うkeyを持つことができる
- Hash keyが同一で、他のアイテムからの検索のために利用
- すべての要素（テーブルとインデックス）の合計サイズを、各ハッシュキーごとに 10 GB に制限



amazon web services

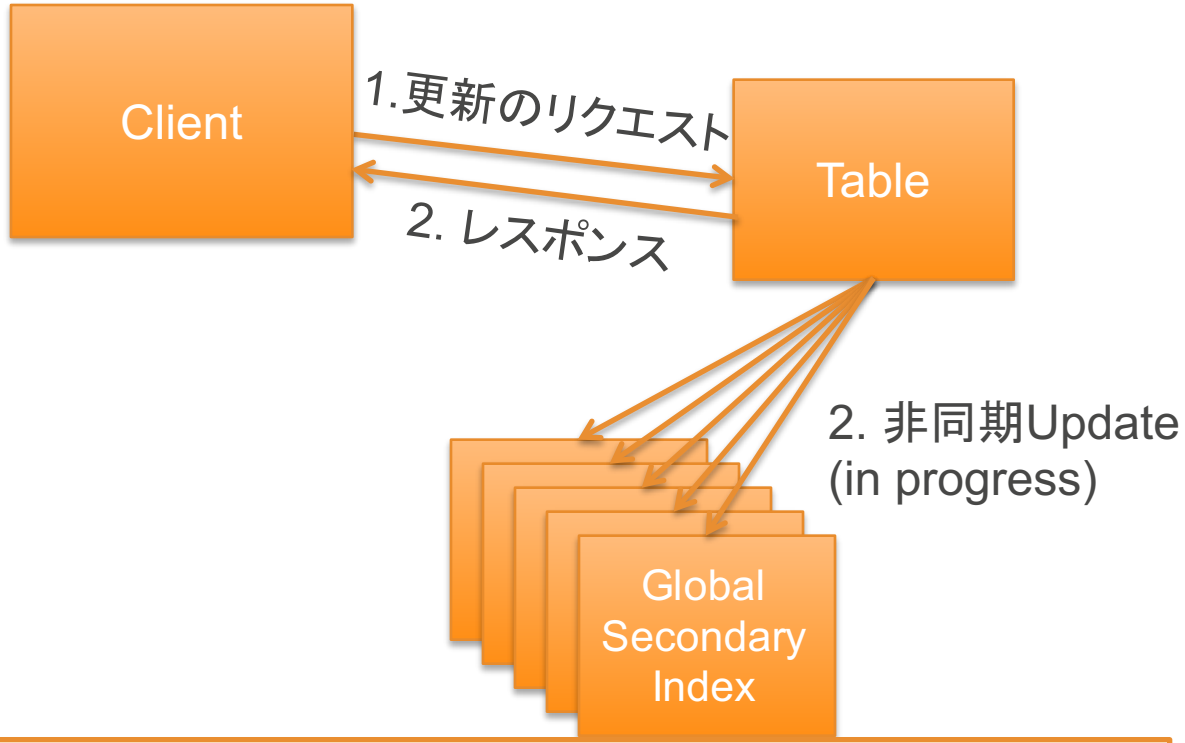
Global Secondary Index (GSI)

- Hash Key属性の代わりとなる
- Hash Keyをまたいで検索を行うためのインデックス



amazon web services

GSIの更新フロー



GSIにはテーブルとは独立したスループットをプロビジョンして利用するため十分なスループットが必要

amazon web services

Scaling

Scaling

- スループット
  - **DynamoDBはテーブル**単位で、読み書きのスループットを指定する必要がある

    （プロビジョニングするスループットキャパシティ）
- サイズ
  - テーブルには任意の数のアイテムが追加可能
    - 1 つのアイテムの合計サイズは 400 KB
    - local secondary index について、異なるハッシュキーの値ごとに最大 10GB のデータを格納

amazon web services

スループット

- テーブルレベルによってプロビジョニング
  - **Read Capacity Units (RCU)**
    - 1 秒あたりの読み込み項目数 x 項目のサイズ (4 KB ブロック)
    - 結果整合性のある読み込みをする場合はスループットが 2 倍

例1）アイテムサイズ：1.2KB(1.2/ 4 = 0.3 ⇒ 1　繰り上げ)
読み込み項目数1000回/秒
1000 × 1 = **1000** RCU

例2）アイテムサイズ:4.5KB (4.5 / 4≒1.1 ⇒ 2　繰り上げ)
読み込み項目数1000回/秒
1000 × 2 = **2000** RCU
※結果整合性のある読み込みの場合
1000 × 2 × ½ = **1000** RCU

amazon web services

スループット

- **Write Capacity Units (WCU)**
  - 1 秒あたりの書き込み項目数 x 項目のサイズ (1 KB ブロック)
  - 1KBを下回る場合は繰り上げられて計算

例1）アイテムサイズ：512B(0.512/ 1≒ 0.5 ⇒ 1 繰り上げ)
書き込み項目数 1000項目/秒
1000 × 1 = **1000** WCU

例2）アイテムサイズ:2.5KB (2.5 / 1 = 2.5 ⇒ 3 繰り上げ)
書き込み項目数1000項目/秒
1000 × 3 = **3000** WCU

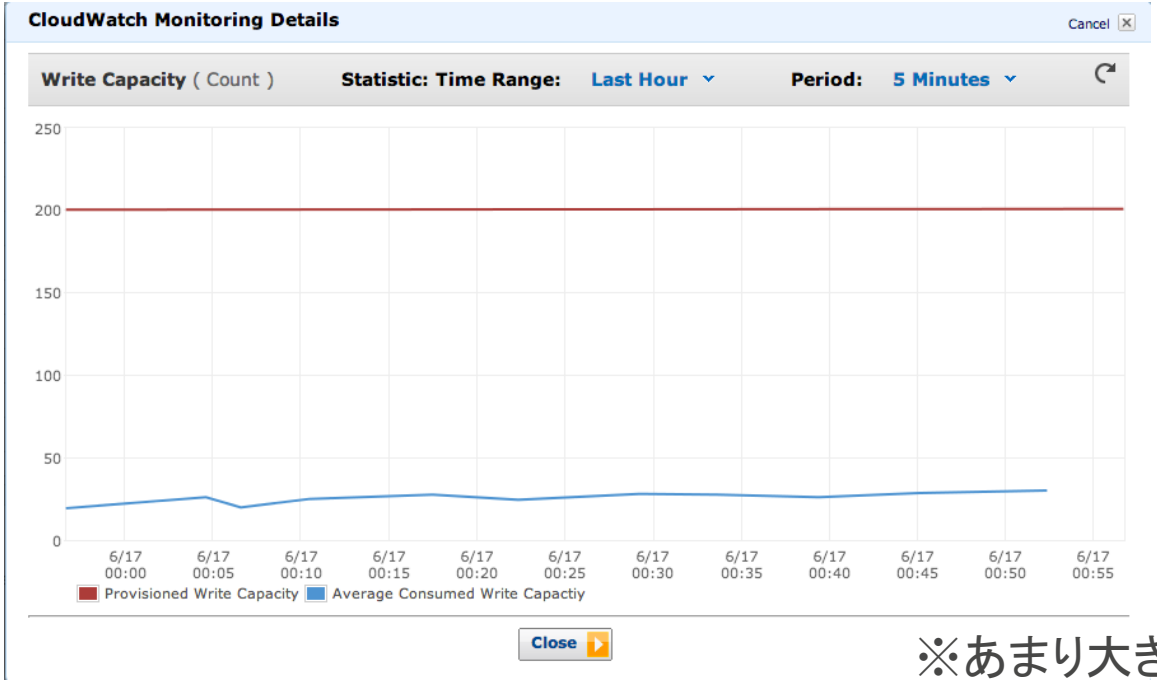
- 読み込みと書き込みのキャパシティユニットは独立して設定


RCU


WCU

amazon web services

スループットの設定



①概算の見積もりから、キャパシティユニットを大きめに設定

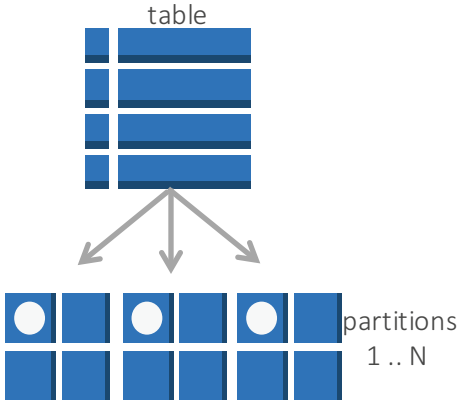
②CloudWatchにて負荷試験、実運用で様子を見て、キャパシティユニットを調整

※あまり大きくし設定し過ぎると、パーテション分割時のキャパシティ分割に注意

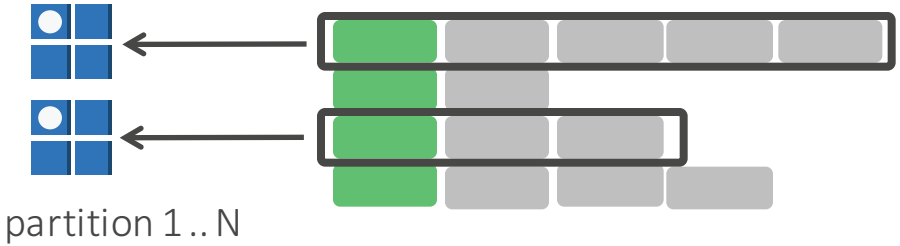
amazon web services

パーティショニング

- DynamoDBはプロビジョンされたスループットキャパシティを確保するためにテーブルを複数のパーティションに分散して格納



- ハッシュキーをパーティション間でのデータ分散に利用し、格納ストレージサイズやプロビジョンされたスループットによって自動的にパーティショニングが実施



amazon web services

パーティショニングの算出

①1 つのパーティションに対して、最大 3,000 個の読み込みキャパシティーユニットまたは 1,000 個の書き込みキャパシティーユニットを割り当てられる。

$$\#\ of\ Partitions_{(for\ throughput)} = \frac{RCU_{for\ reads}}{3000\ RCU} + \frac{WCU_{for\ writes}}{1000\ WCU}$$

②単一のパーティションには、約 10 GB のデータを保持される

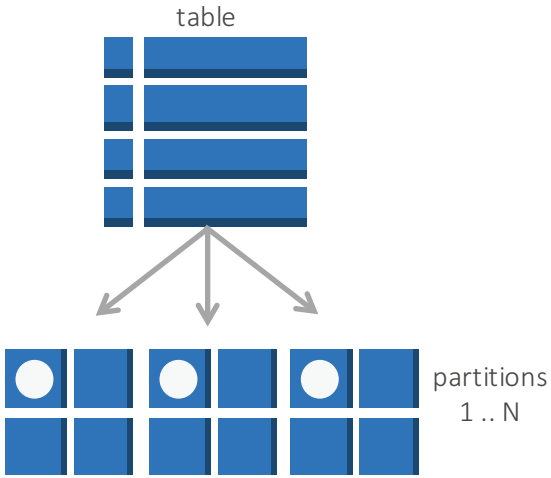
$$\#\ of\ Partitions_{(for\ size)} = \frac{Table\ Size\ in\ GB}{10\ GB}$$

amazon web services

パーティショニングの算出方法

- 大きいほうを採用

$$\#\ of\ Partitions = \mathrm{MAX}(\#of\ Partitions \mid \#of\ Partitions)$$
$$(total) \qquad (for\ size) \qquad (for\ throughput)$$



amazon web services

パーティショニング算出例

Table size = 8 GB, RCUs = 5000, WCUs = 500

- スループット

$$\#\ of\ Partitions_{(for\ throughput)} = \frac{5000_{RCU}}{3000\ RCU} + \frac{500_{WCU}}{1000\ WCU} = 2.17 = 3$$

- ストレージサイズ

$$\#\ of\ Partitions_{(for\ size)} = \frac{8\ GB}{10\ GB} = 0.8 = 1$$

- 大きいほうを採用

$$\#\ of\ Partitions_{(total)} = \mathrm{MAX}(1_{\ for\ size} | 3_{\ for\ throughput}) = 3$$

RCUとWCU の値は均一に各パーテションに割り当てられます

RCUs per partition = 5000/3 = 1666.67
WCUs per partition = 500/3 = 166.67
Data/partition = 8/3 = 2.66 GB

DynamoDB のスループットを最大限に活用

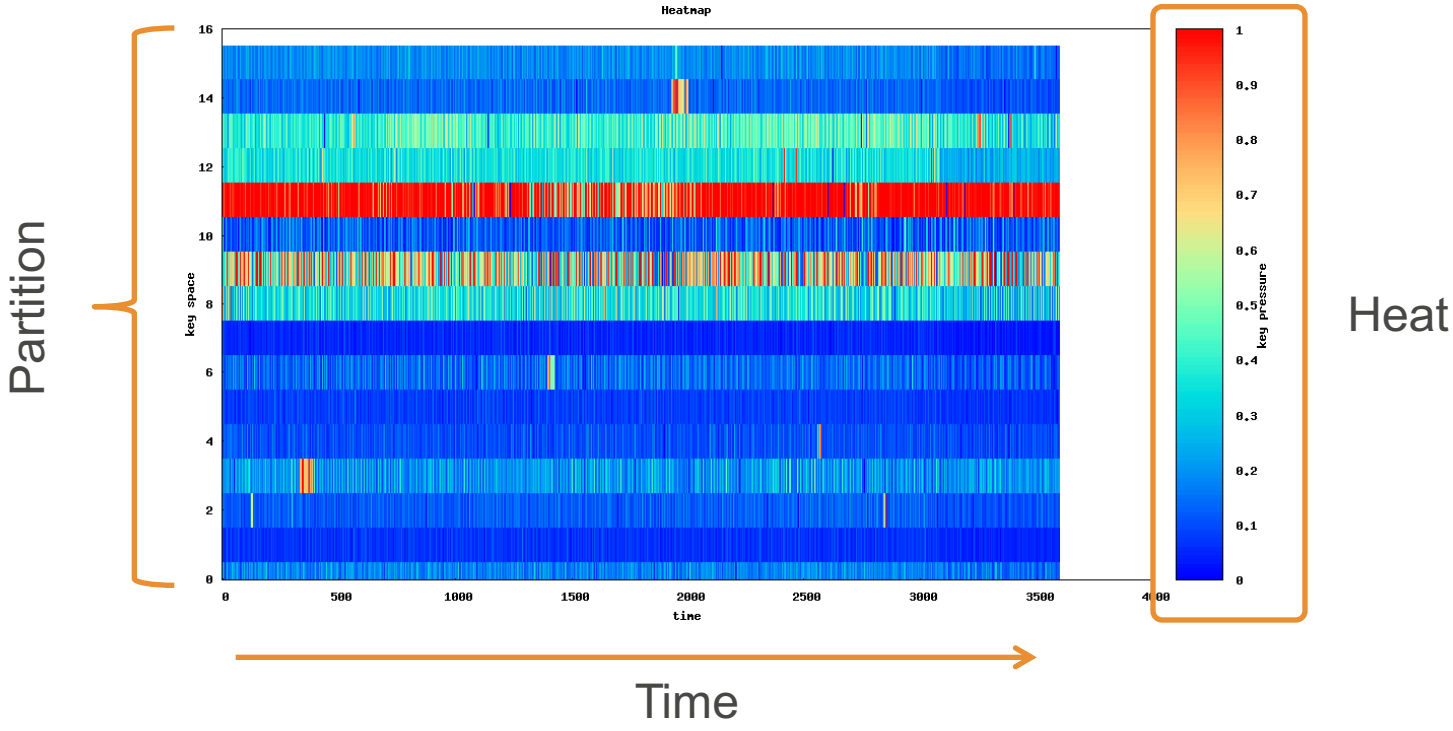
“DynamoDB のスループットを最大限に活用するには、テーブルを作成するときに、ハッシュキー要素に個別の値が多数含まれ、できるだけランダムかつ均一に値がリクエストされるようにします。”

– DynamoDB Developer Guide

- Space: キーアクセスはなるべく均等になるように
- Time: リクエストはなるべく均等な間隔で

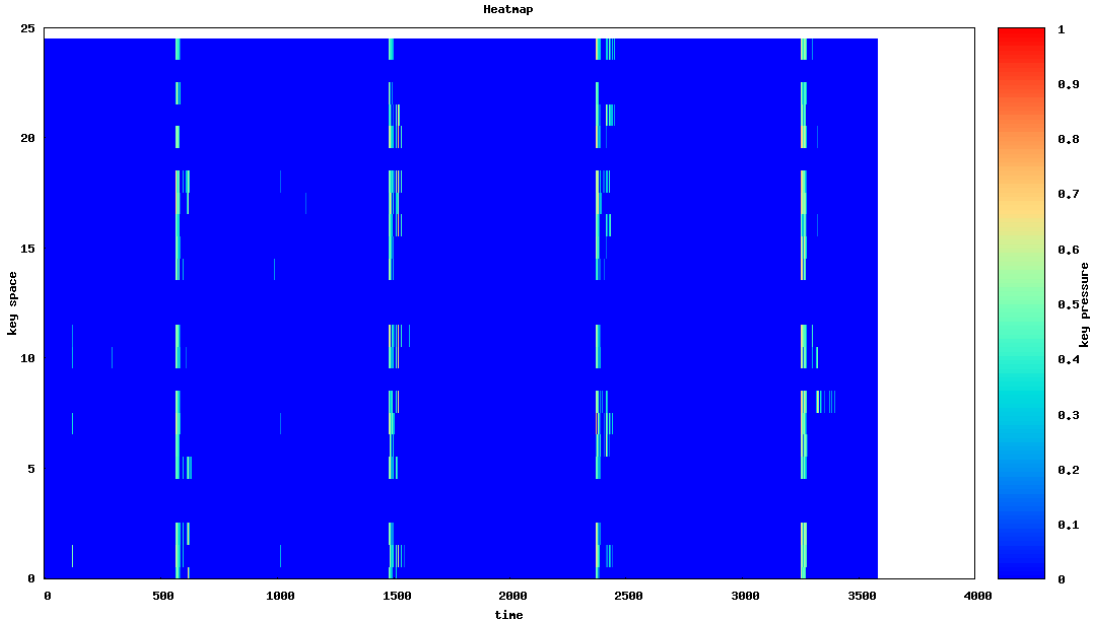
amazon web services

Example: Hot Keys



amazon web services

Example: Periodic spike



amazon web services

DynamoDBの料金体系

- プロビジョニングされたスループットで決まる時間料金
  - Read/Writeそれぞれプロビジョンしたスループットによって時間あたりの料金がきまる
  - 大規模に利用するのであればリザーブドキャパシティによる割引もあり
- ストレージ利用量
  - 保存したデータ容量によって決まる月額利用料金
  - 計算はGBあたりの単価が適用される

詳細はこちらを参照
http://aws.amazon.com/jp/dynamodb/pricing/

ユースケース及び、ベストプラクティス

amazon web services

リアルタイム投票システム

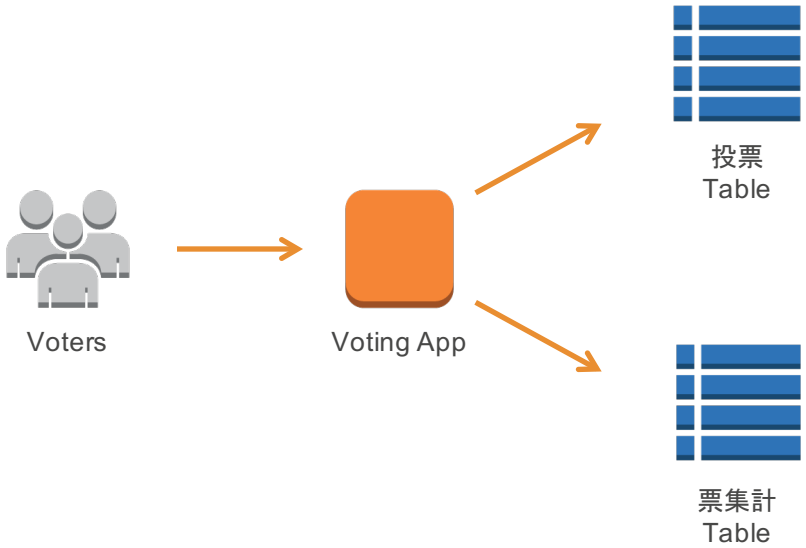
Write-heavy items

amazon web services

投票システムの要件

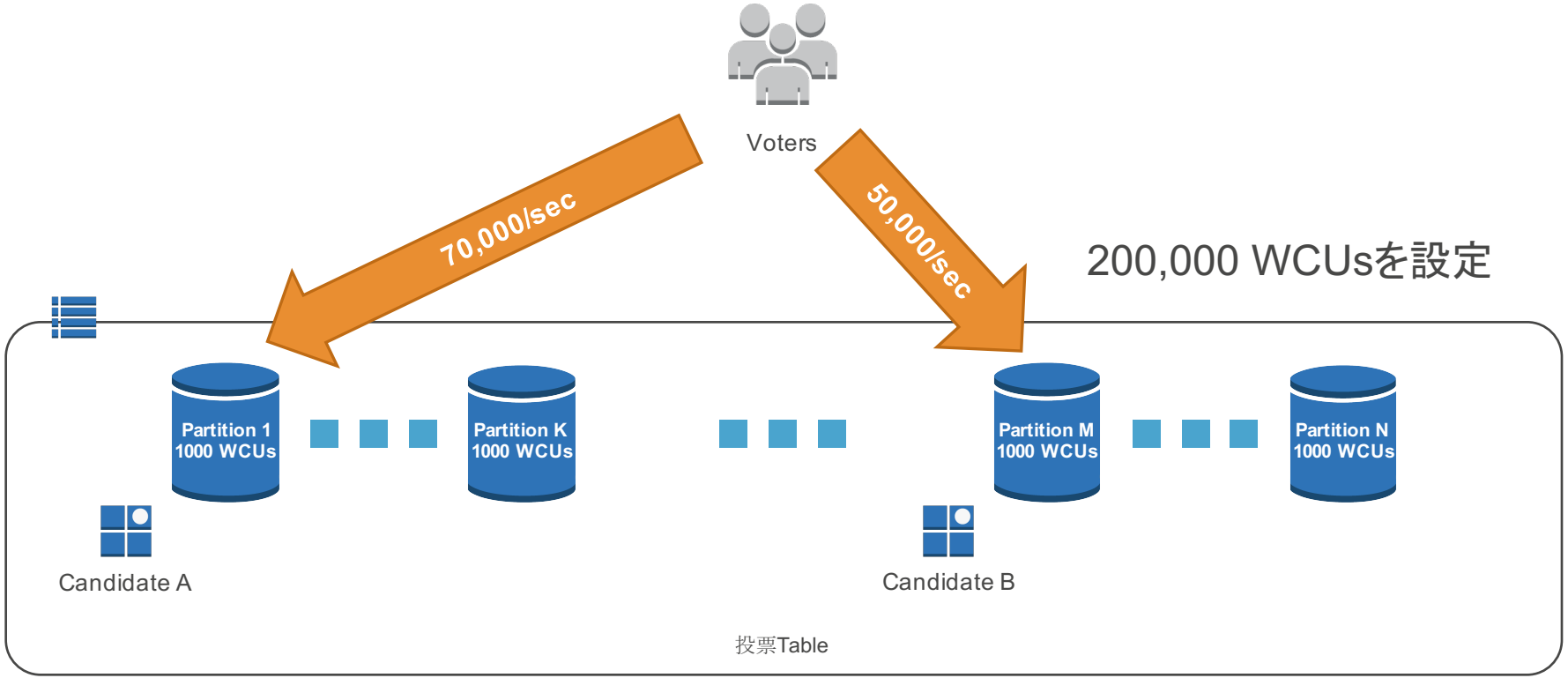
- 投票は一回のみ
- 一度投票したものは変えられない
- リアルタイムに集計
- 投票者の統計、分析を行いたい

リアルタイム投票システムのアーキテクチャ



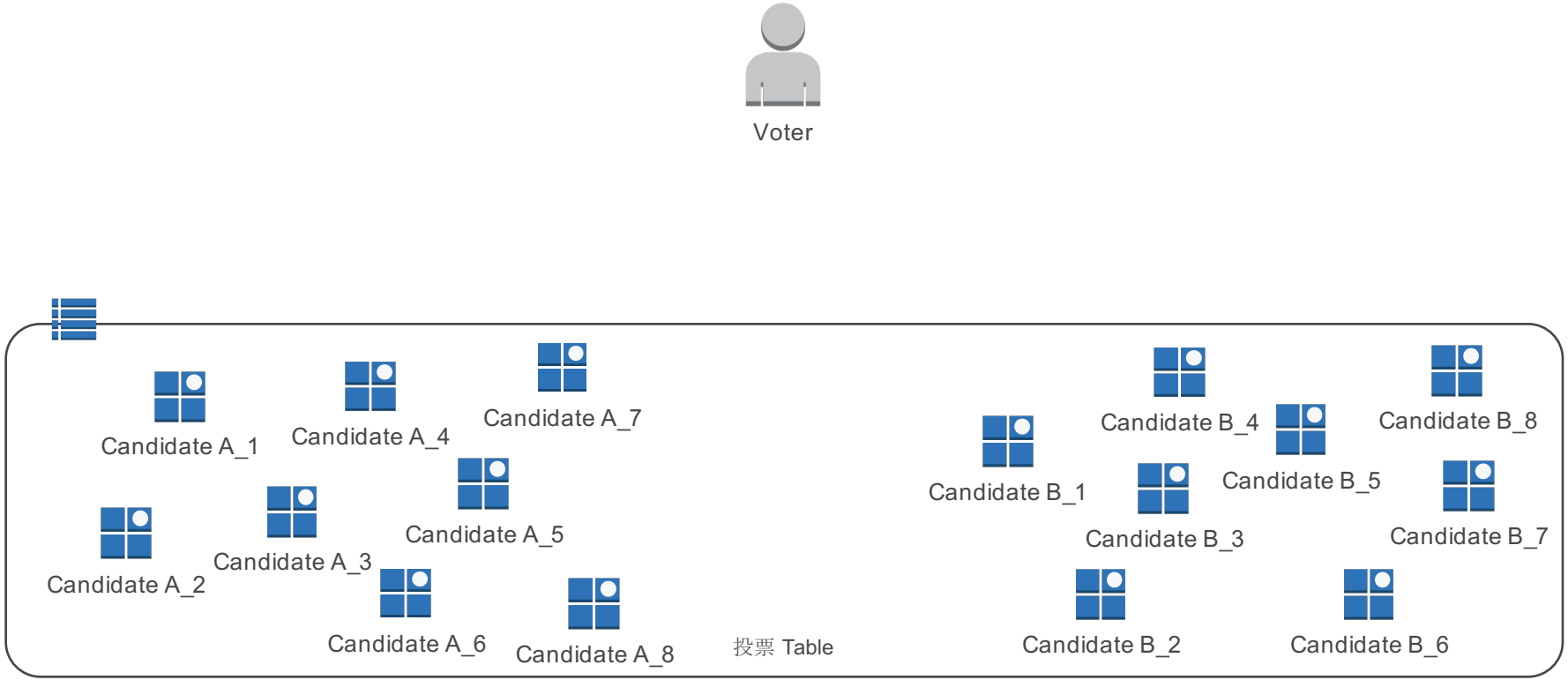
amazon
web services

スケールすることによるボトルネック

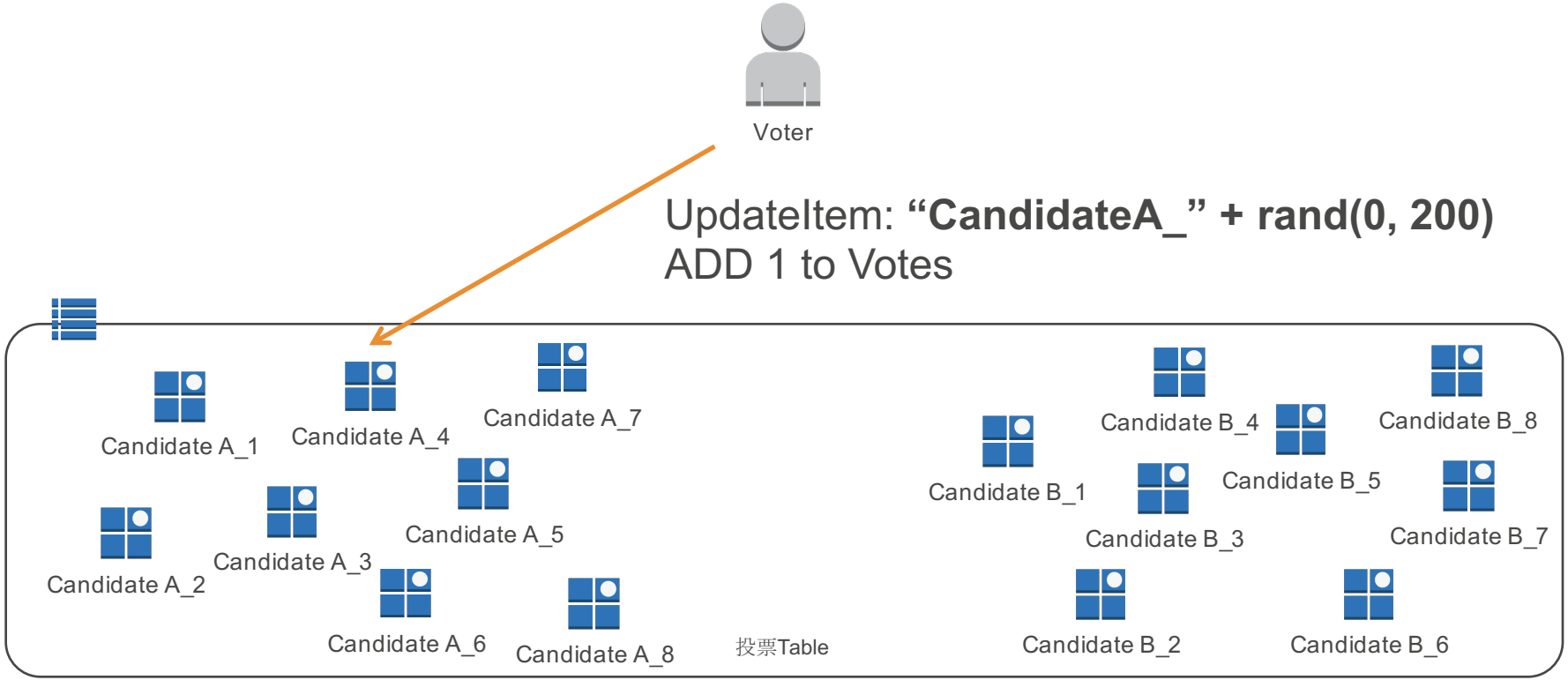


amazon web services

シャーディングでの書き込み



シャーディングでの書き込み



正確な投票

Voter

1. 投票データの登録、重複排除

2. 投票データの集計

投票Table

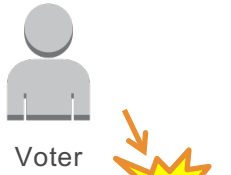
UserId	Candidate	Date
1	A	2013-10-02
2	B	2013-10-02
3	B	2013-10-02
4	A	2013-10-02

投票集計Table

Segment	Votes
A_1	23
B_2	12
B_1	14
A_2	25

amazon web services

正確な集計は?



投票Table

UserId	Candidate	Date
1	A	2013-10-02
2	B	2013-10-02
3	B	2013-10-02
4	A	2013-10-02

投票集計Table

Segment	Votes
A_1	23
B_2	12
B_1	14
A_2	25

amazon web services

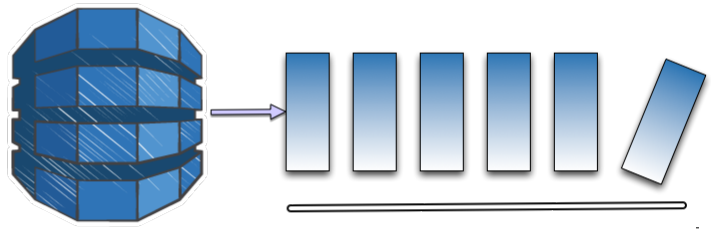
DynamoDB Streams

New

In preview

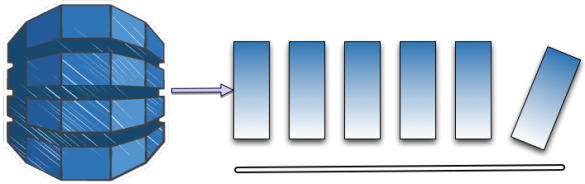
What is DynamoDB Streams?

It is a Streams of updates
Scales with your table



amazon web services

DynamoDB Streams

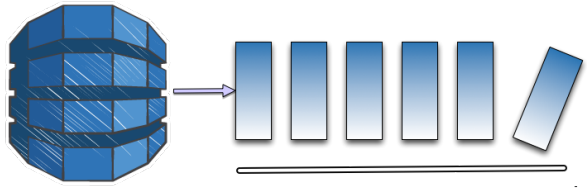


New

- テーブルの更新の情報を保持
- 非同期に更新
- シリアライズされたデータ
- アイテム毎の厳密な管理

- 高耐久性
- テーブルよるスケール
- 有効期限は24時間
- 1秒未満の遅延書き込み

View types

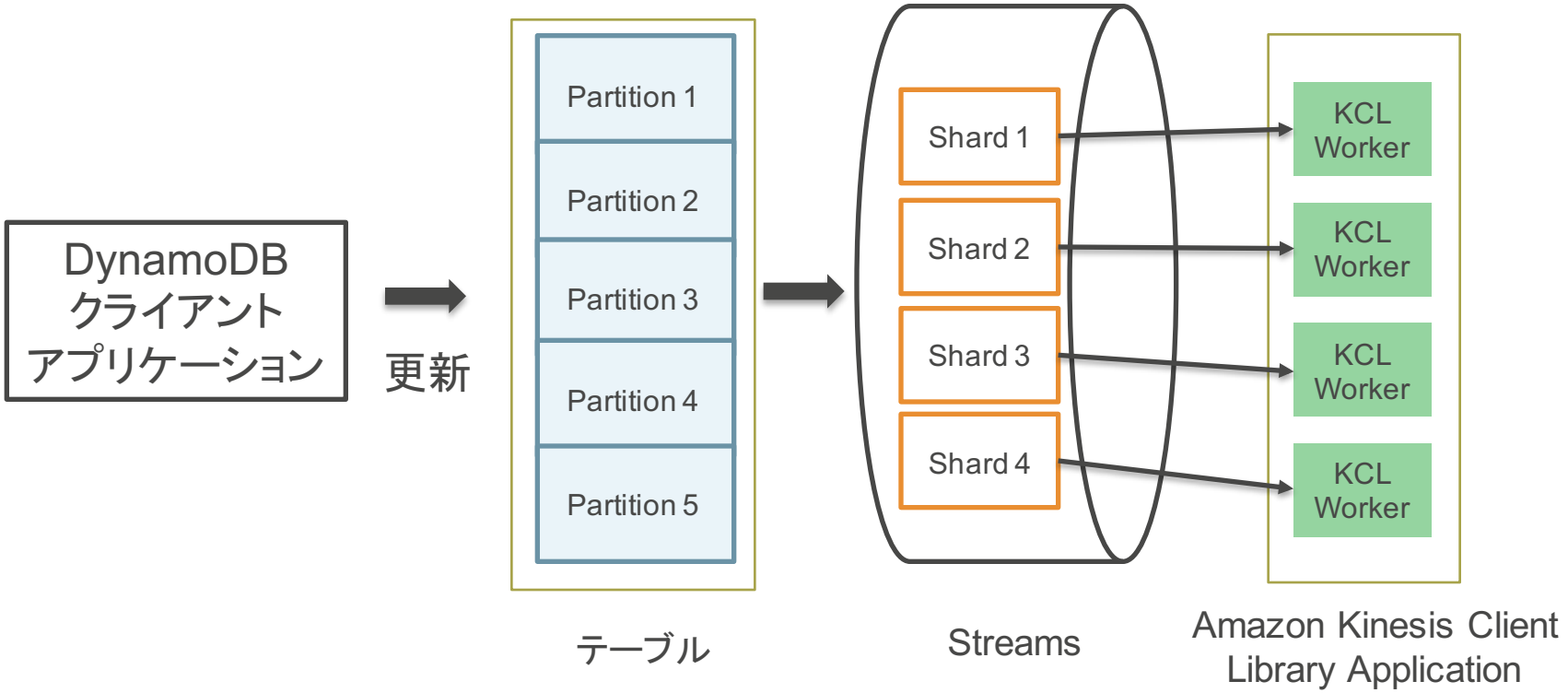


更新情報（Name = John, Destination = Mars）
⇒(Name = John, Destination = Pluto)

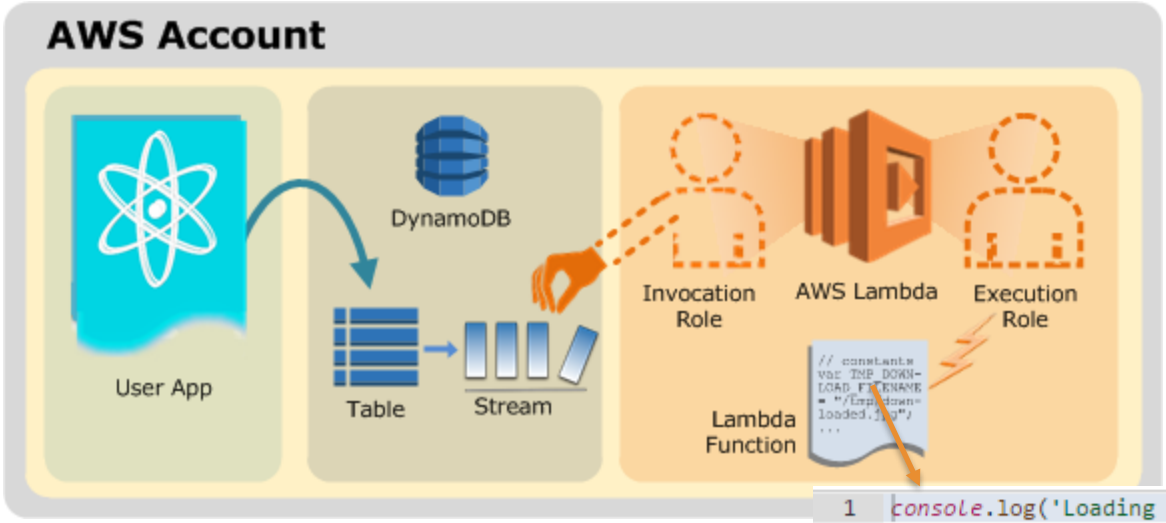
View Type	Destination
Old Image – 更新前の情報	Name = John, Destination = Mars
New Image – 更新後の情報	Name = John, Destination = Pluto
Old and New Images	Name = John, Destination = Mars Name = John, Destination = Pluto
Keys Only	Name = John

amazon web services

DynamoDB Streams and Amazon Kinesis Client Library



DynamoDB Streams and AWS Lambda



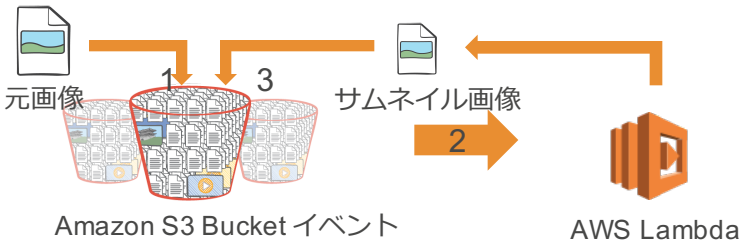

```
1  console.log('Loading event');
2  exports.handler = function(event, context) {
3    console.log("Event: %j", event);
4    for(i = 0; i < event.Records.length; ++i) {
5      record = event.Records[i];
6      console.log(record.EventID);
7      console.log(record.EventName);
8      console.log("DynamoDB Record: %j", record.Dynamodb);
9    }
10   context.done(null, "Hello World");  // SUCCESS with message
11 }
```

```
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"
```

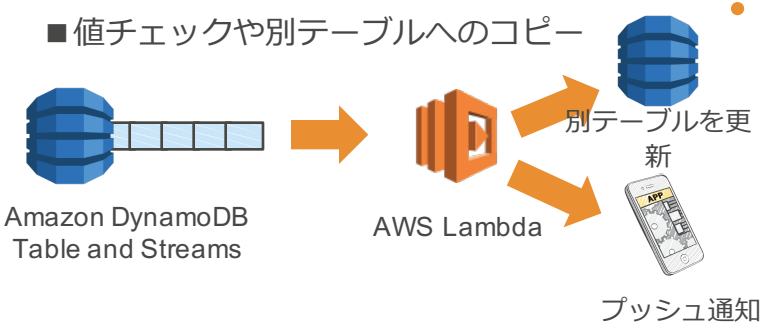
AWS Lambda

イベントをトリガーにコードを実行するコンピュートサービス

■イメージのリサイズやサムネイルの作成



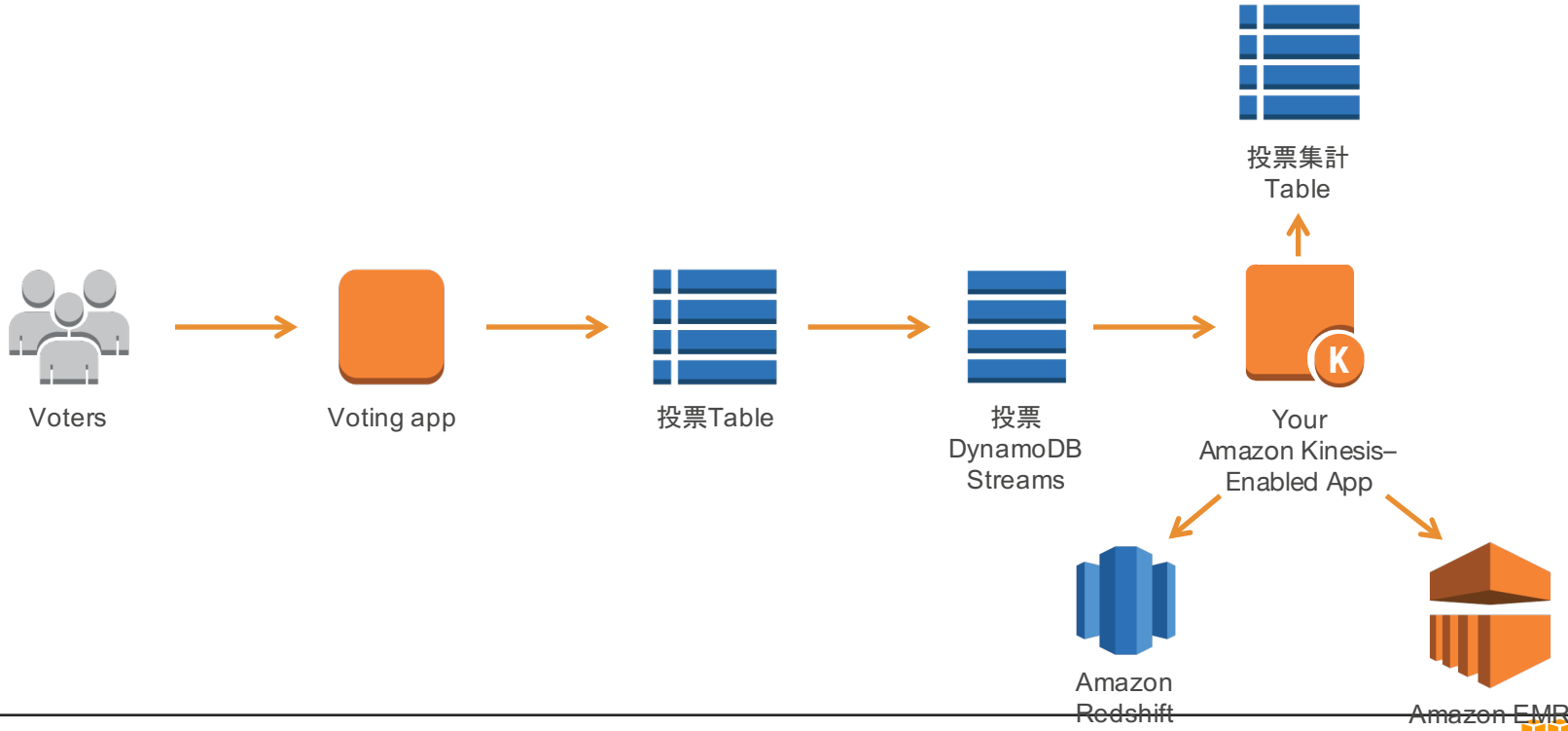
■値チェックや別テーブルへのコピー



- **特徴** (http://aws.amazon.com/jp/lambda/)
  - OS、キャパシティ等インフラの管理不要
  - S3、Kinesis、SNS等でのイベント発生を元にユーザが用意したコード(Node.js,java 8)を実行
  - ユーザアプリからの同期/非同期呼び出し
- **価格体系** (http://aws.amazon.com/jp/lambda/pricing/)
  - コード実行時間(100ms単位)
  - Lambdaファンクションへのリクエスト回数
  - 1月あたり100万リクエスト、400,000GB/秒が無料で利用可能

amazon web services

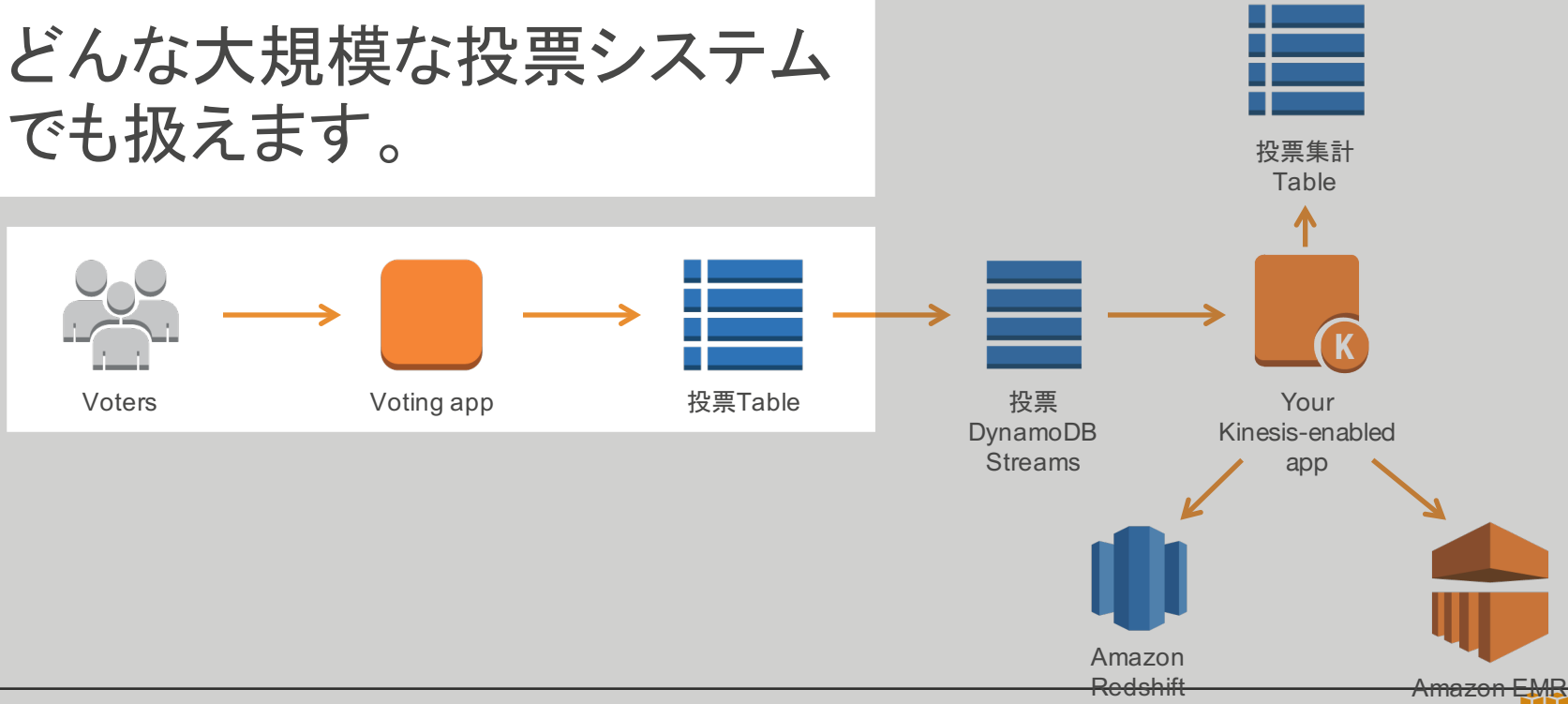
Real-time voting architecture



amazon
web services

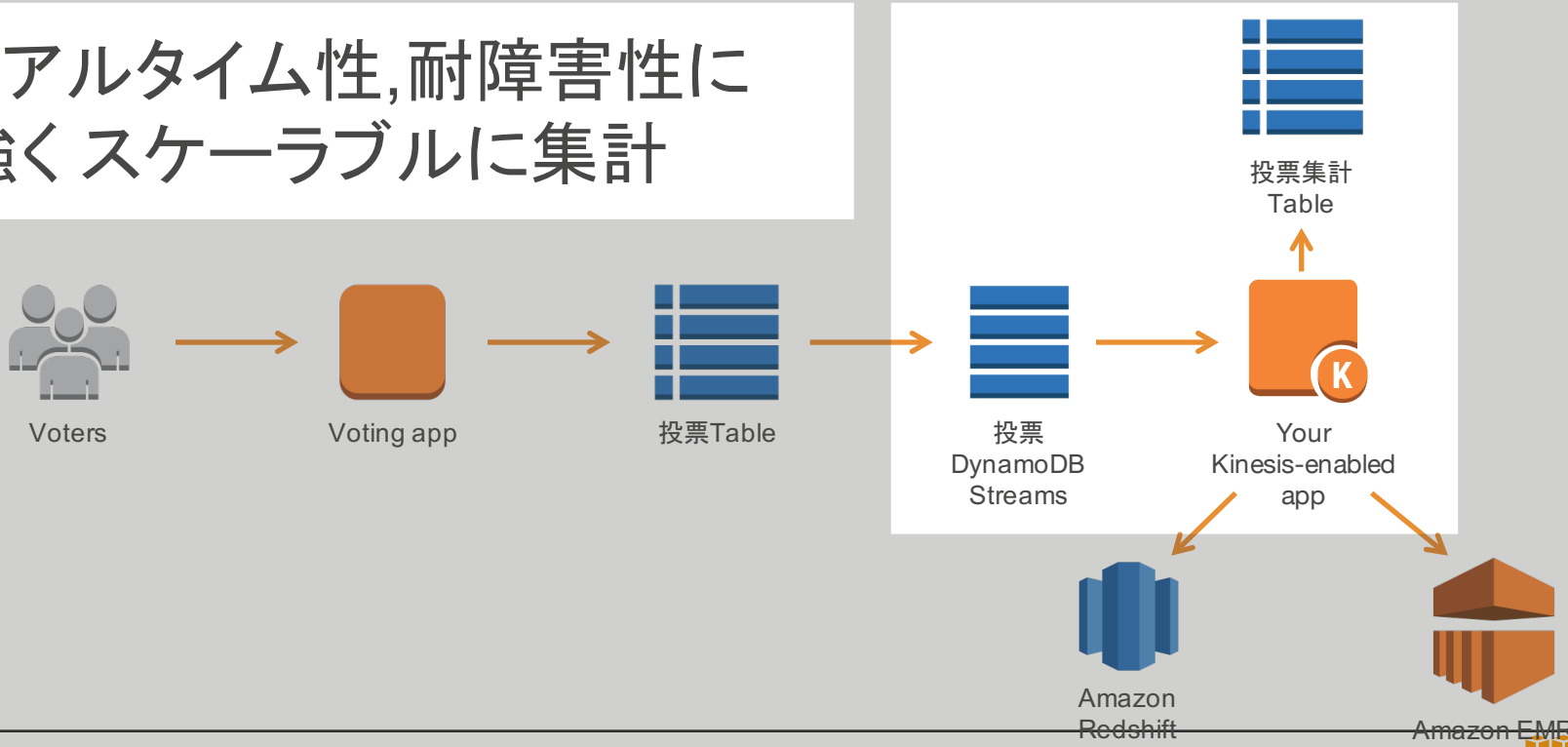
Real-time voting architecture

どんな大規模な投票システムでも扱えます。



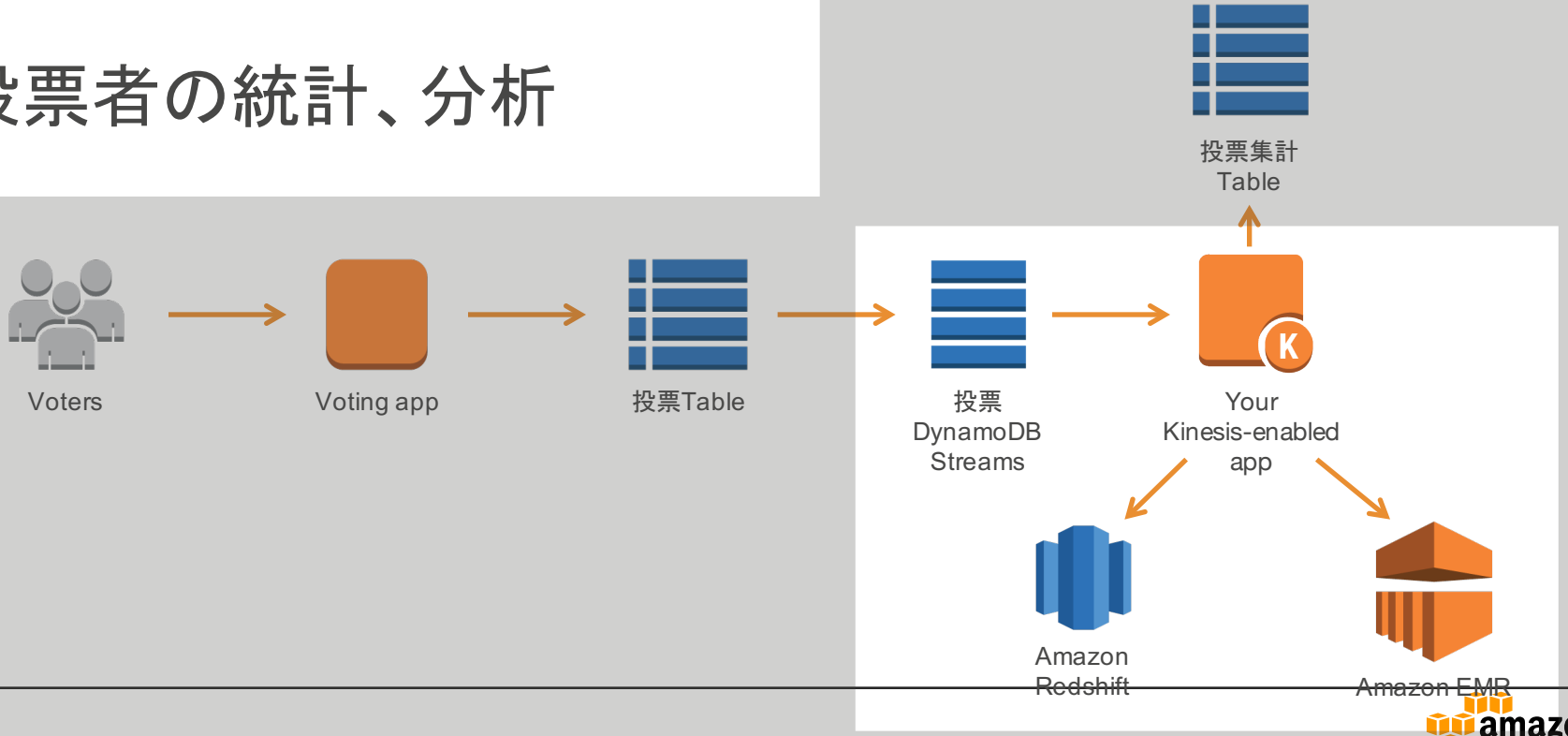
Real-time voting architecture

リアルタイム性,耐障害性に
強くスケーラブルに集計



Real-time voting architecture

投票者の統計、分析



DynamoDBが使われているユースケース

- KVSとして
  - Webアプリケーションのセッションデータベース
  - ユーザー情報の格納するデータベース
- 広告やゲームなどのユーザー行動履歴DBとして
  - ユーザーIDごとに複数の行動履歴を管理するためのデータベース
- ソーシャルアプリのバックエンドとして
  - モバイルアプリから直接参照できるデータベースとして
- 他にも
  - バッチ処理のロック管理
  - フラッシュマーケティング
  - ストレージのインデックス

amazon web services

NoSQL vs RDB

NoSQL

得意/メリット
➔スケーラビリティ

不得意/デメリット
➔複雑なクエリ
➔トランザクション

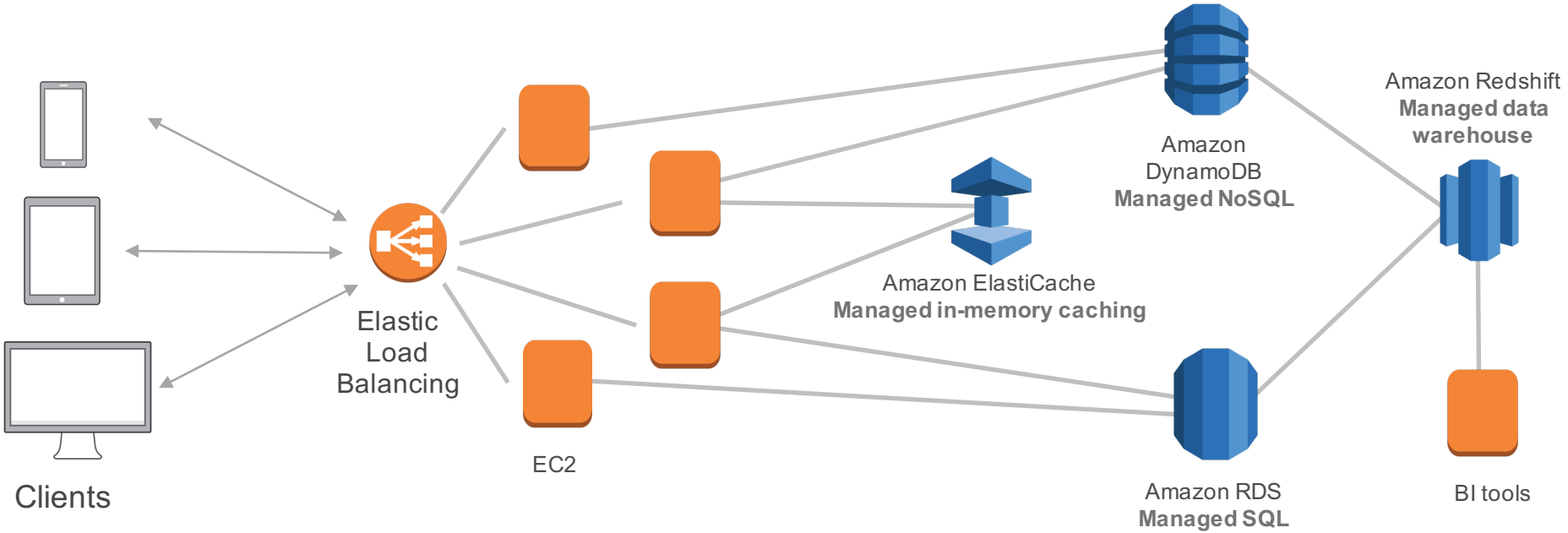
RDB

得意/メリット
➔柔軟なクエリ
➔トランザクション

不得意/デメリット
➔スケーラビリティ

amazon web services

ユースケースに合わせてDB製品を選択



amazon web services

ありがとうございました！