

# Amazon DynamoDB -NoSQL Service in the Cloud-

AWSマイスターシリーズ**RELOADED**

Shinpei OHTANI

ohtani@amazon.co.jp



# アジェンダ

- 📦 NoSQLとは何か
- 📦 Amazon DynamoDB
- 📦 DynamoDB機能
- 📦 DynamoDBでの開発
- 📦 コストモデル
- 📦 DynamoDBとエコシステム
- 📦 DynamoDB利用事例
- 📦 まとめ

# 進化・深化するAWSプラットフォーム

## お客様のアプリケーション



# AWSの提供するデータベースサービス

- データベースの管理負荷を減らす
- すぐに利用可能
- コストを最小化出来る



## NoSQL データベース

管理不要・スキーマレス・スケーラブル



## リレーショナルデータベース

管理レス・既存知識活用・移行容易性



## インメモリキャッシュ

読込高速化・DB負荷削減

# NoSQL

## (DynamoDB)

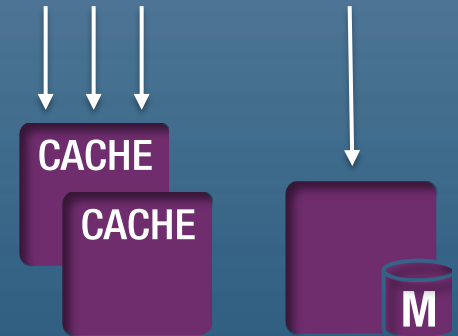
の前に . . .

# YesSQL(RDBMS)

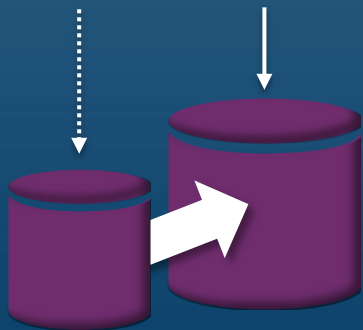
# YesSQL=Amazon RDS

Amazon RDS 設定	可用性の 向上	スループット の向上	レイテンシ の削減
ボタン1発スケール		✓	
マルチ AZ	✓		
リードレプリカ		✓	
ElastiCacheの利用		✓	✓

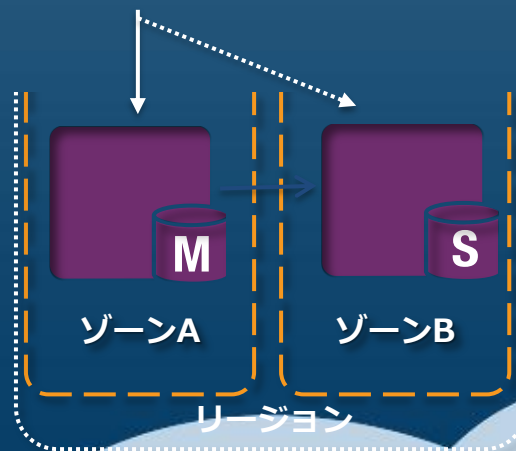
## ElastiCacheの利用



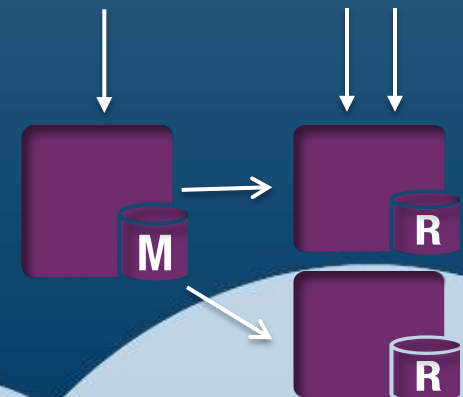
## ボタン1発スケール



## マルチAZ



## リードレプリカ





# NoSQLとはなにか

📦 Not Only SQLなデータベース群の総称

📦 特徴

- 目的特化型・ユースケース特化型が多い
- スケールアウト
- 伸縮自在
- 高い可用性

📦 特に書き込み負荷の軽減が目的な事が多い

# NoSQL (DynamoDB)

One Size  
Does Not  
Fit All

# データベース 4象限

by James Hamilton

スケール  
優先

シンプルな  
ストレージ  
機構

機能  
優先

目的  
特化型  
ストア



# Amazon DynamoDBとは？

- ・フルマネージドなNoSQLデータベース
- ・超高速・予測可能な一貫したパフォーマンス
- ・シームレスなスケラビリティ、そして低コスト

## Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Voshall 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 customers trust. The Amazon.com platform, which provides service for many websites 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.com is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon.com 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.

運用管理必要なし

低レイテンシ、SSD

プロビジョンスループット

無限に使えるストレージ



# Amazon 10年以上の経験の末に・・・

- Amazonの継続構築・改善・運用経験
  - 大規模なデータベースの構築経験
  - RDBMSからNoSQLデータベースまで
- AWSで培ってきた、拡張性が高く信頼性の高いクラウドサービスの構築経験
  - S3, SimpleDB, RDS...
- 上記の経験を活かしてDynamoDBという、データベースサービスを実現

# DynamoDBの特徴



管理不要である



プロビジョンスループット



データへの高速アクセス



信頼性が高い



# 管理不要なデータベース・サービス

- 管理不要で高い拡張性を提供
- サービス vs プロダクトの違い
  - DynamoDBはサービス
  - 他のデータベースはプロダクト
  - 運用コストに大きな差が出る



# プロビジョンスループット



# プロビジョンスループット




# プロビジョンスループットとは

- IOPSを開発者がいつでも指定できる
  - テーブル作成時、あとからでも可能
    - ダウンタイムなし
  - 秒間あたりの、読み/書きを指定
    - 1レコードが1KBとした場合の数値
  - 事実上IOPSの値は**無制限**

# データへの高速なアクセス

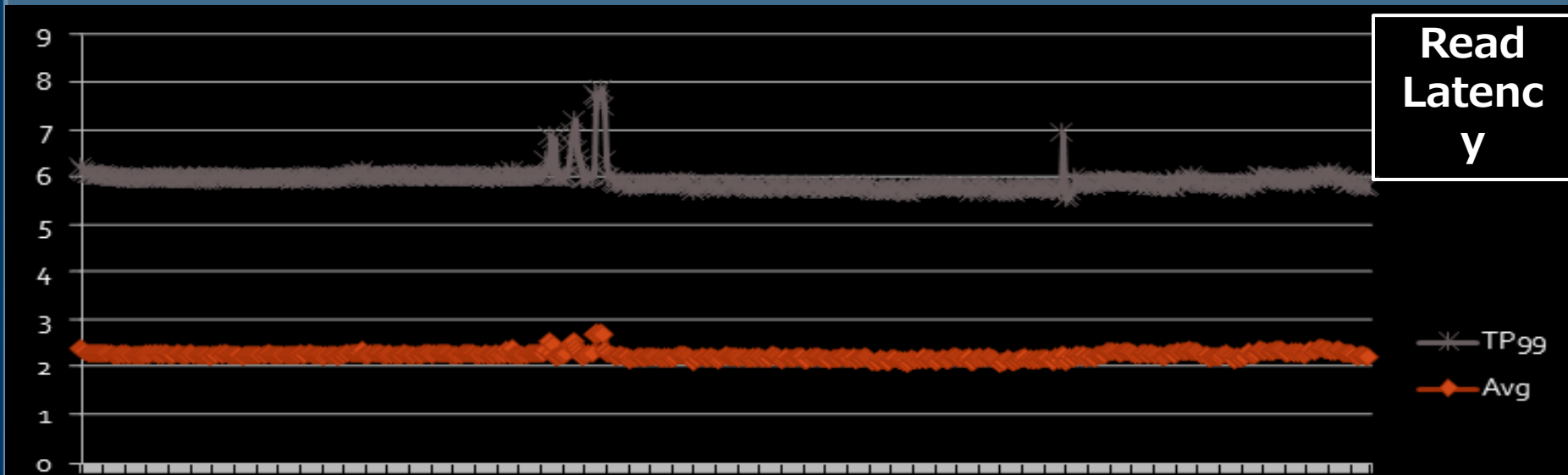
## AWSクラウドの特性を活用

- ストレージに**SSD**(Solid State Drive)利用
- SSD用に最適化
- データへの高速なアクセス

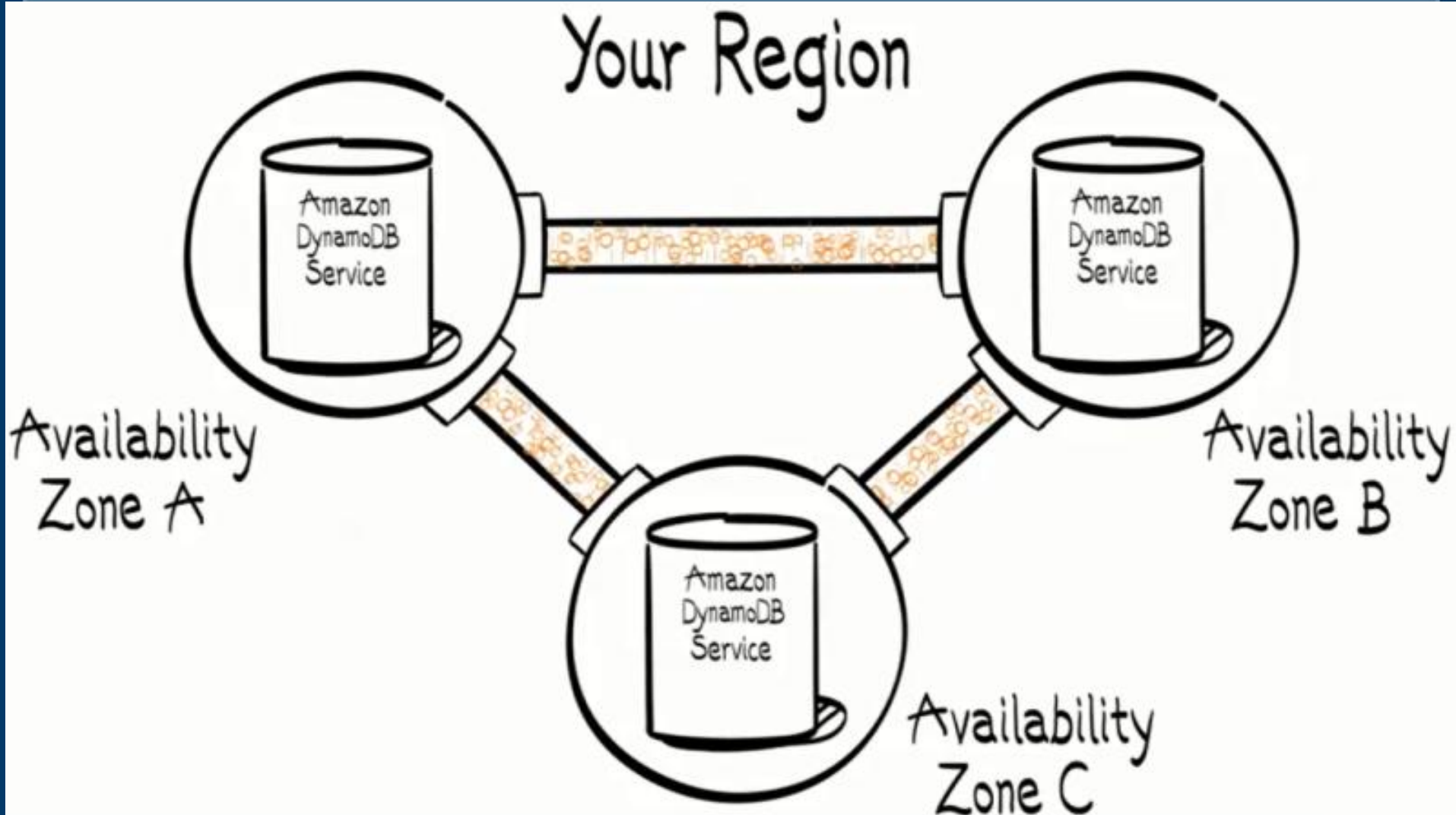
 データベースがいかに大規模になっていようと、数ミリ秒のアクセス速度が可能

# 非常に低いレイテンシ

数ミリ秒のレイテンシ



# DCレベルの障害にも耐える高い堅牢性



# DynamoDB基本機能





# DynamoDBのデータモデル

## テーブル

### アイテム

プライマリキー

アトリビュート

# プライマリーキー



## ハッシュキー

- シンプルなキーバリュー
- ハッシュ値なので、ソートなし



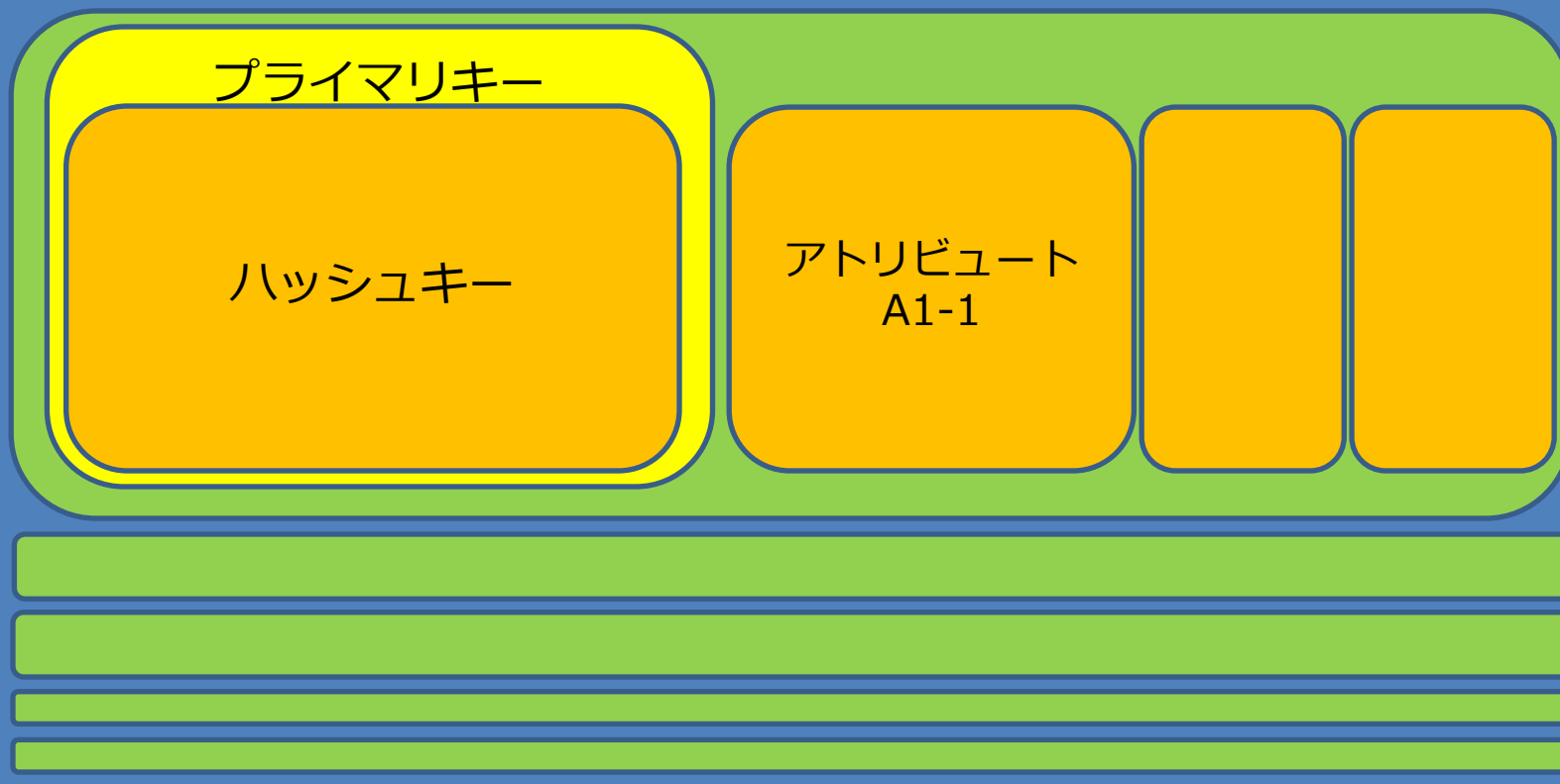
## ハッシュキー+レンジキー

- コンポジットプライマリーキー
- レンジキーはソートあり

# DynamoDBデータモデル

## ハッシュキーのみの場合

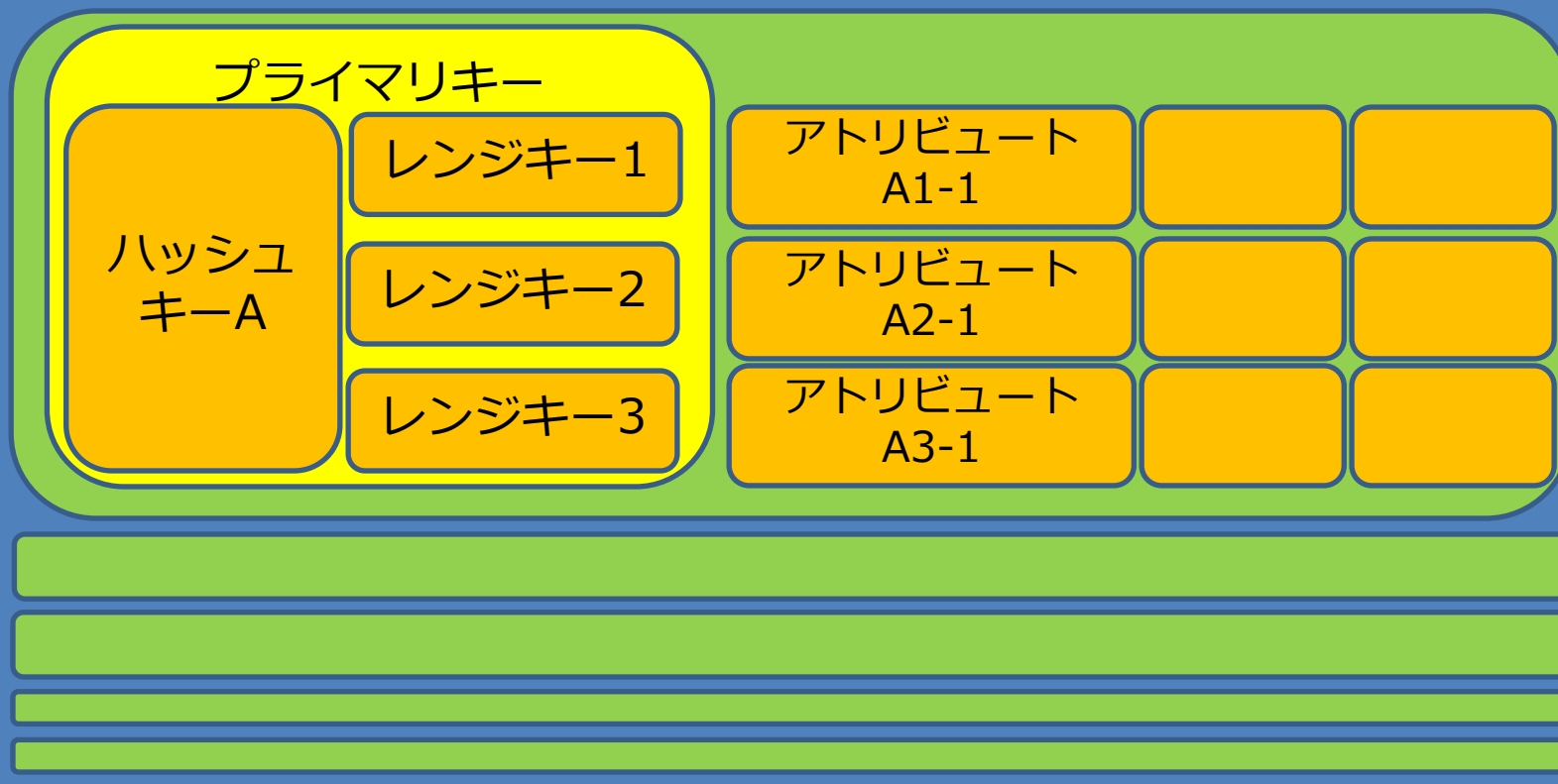
### アイテム



# DynamoDBデータモデル

## ハッシュキー+レンジキーの場合

### アイテム



# 一貫性モデル

## 強い一貫性

- 書き込み/読み込み

## 結果一貫性

- 読み込み

## 書き込みは即座にSSDで永続化

## 1行の書き込みはトランザクショナル

# (余談)一貫性の強度

## 強一貫性(WRITE/READ)

- 書き込み直後から値が反映される

## 因果一貫性

- イベント順序を維持し一貫性を維持

## 結果一貫性(READ)

- ある一定の伝播期間を持って、最終的に結果が維持できる

# DynamoDBが持てるデータ構造

## 4種類のデータ構造

- String
- Number
- String配列
- Number配列

# DynamoDBの基本クエリ

 Get/Put/Update/Delete/BatchGet

 Scan

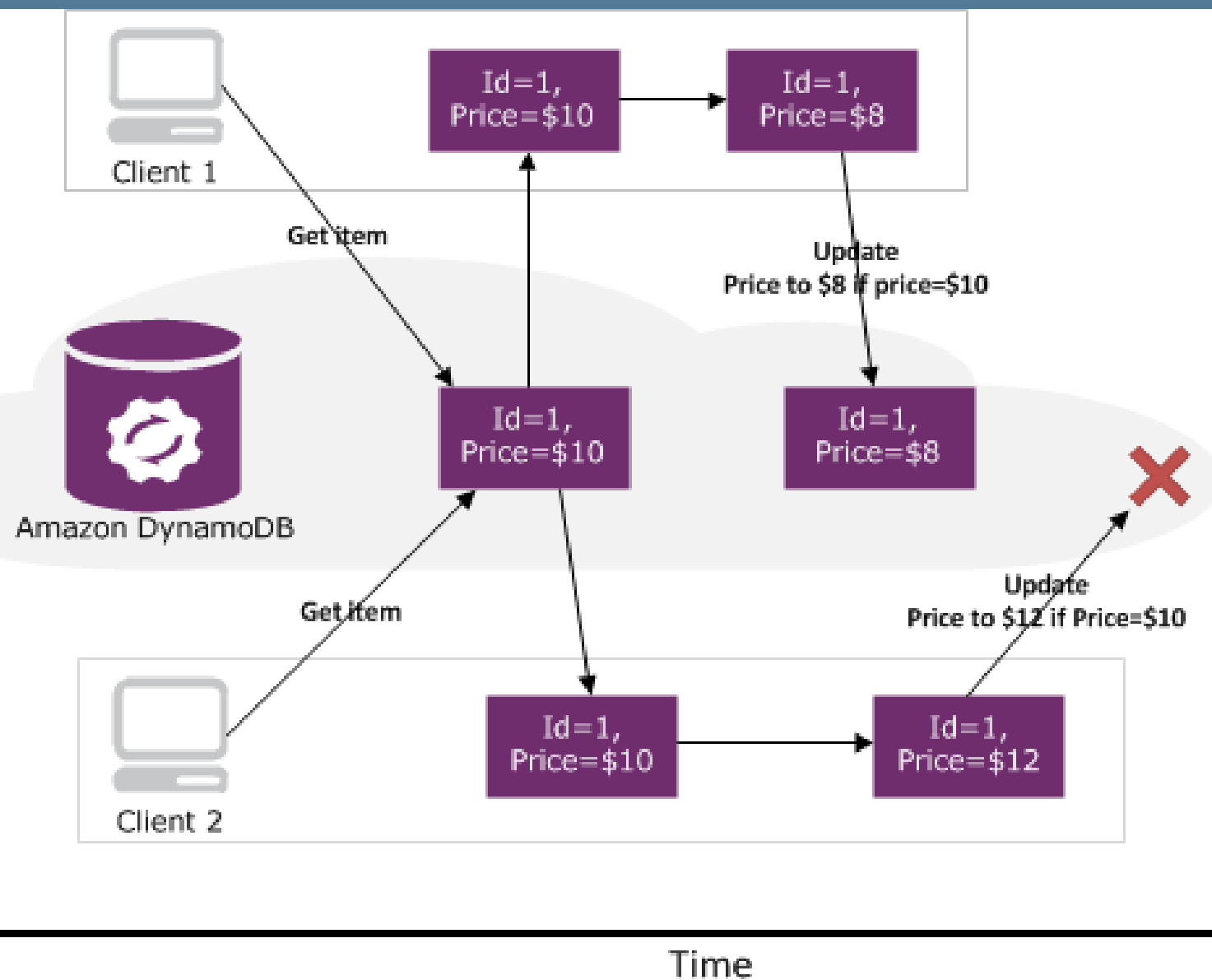
- テーブル総ナメする

 Query

- ハッシュ+レンジキーのみ



# 条件付き書込/アトミックカウンタ



# マネージメントコンソールで運用管理可能

AWS Management Console x

← → ↺ [https://console.aws.amazon.com/dynamodb/home#table:name=demo\\_stat\\_table;region=ap-northeast-1](https://console.aws.amazon.com/dynamodb/home#table:name=demo_stat_table;region=ap-northeast-1) ☆ BI 🗖

AWS\_Tools AWS Managemen... Pushing the Limit... Asana Google Reader (1... ATN Amazon S3, Micro... Simple Monthly C... その他のブックマーク

AWS Management Console > Amazon DynamoDB Shinpei Ohtani | Help ▾

AWS Elastic Beanstalk S3 EC2 VPC CloudWatch Elastic MapReduce CloudFront CloudFormation RDS ElastiCache SQS IAM SNS SES Route 53 **Amazon DynamoDB** Storage Gateway SWF

Tables

Region: Asia Pacific (Tokyo) Help

Filter:  Create Table Modify Throughput Delete Table Refresh 1 to 10 of 14 tables

Name	Status	Hash Key	Range Key	Read Throughput	Write Throughput
Orders-2012-01	ACTIVE	Order ID	-	100	100
ProductCatalog	ACTIVE	id	-	20	10
demo_stat_table	ACTIVE	ppid	pid	64	64
demo_table	ACTIVE	id	-	8	8
employee	ACTIVE	id	time	20	20
mogemoge	ACTIVE	id	time	20	20
my-favorite-movies-table	ACTIVE	name	-	10	10
mytable	ACTIVE	id	-	20	20
nectable	ACTIVE	id	-	100	50
perf_10	ACTIVE	id	-	10	10

Table Items

Details

**Monitoring**

Alarm Setup

Alarms and Usage Summary

Recent Alarms (in the past 24 hours)

-- No alarms triggered -- [Go to CloudWatch](#)

CloudWatch Metrics (for the past hour)

Click to view in CloudWatch. Links open in new windows or tabs

Consumed Read Capacity Units: 0

Consumed Write Capacity Units: 0.001

Get Item Latency (avg in milliseconds): 12.872

Put Item Latency (avg in milliseconds): 9.856

Query Latency (avg in milliseconds): 17.13

[View All Metrics...](#)

View Additional Metrics in CloudWatch

Select a metric to view in CloudWatch. This will open Amazon CloudWatch in a new browser window. Please visit [here](#) for more information.

Table Name: demo\_stat\_table

Metric Name: ConsumedReadCapacityUnits

Operation: ALL

[Show Metrics](#)

© 2008 - 2012, Amazon Web Services LLC or its affiliates. All rights reserved. | [Feedback](#) | [Support](#) | [Privacy Policy](#) | [Terms of Use](#) | An [amazon.com](#) company

# DynamoDBの監視・セキュリティ



## CloudWatchで運用監視可能

- アカウント全体またはテーブル毎
- メトリクスも豊富



## SNS経由で通知も可能



## セキュリティ

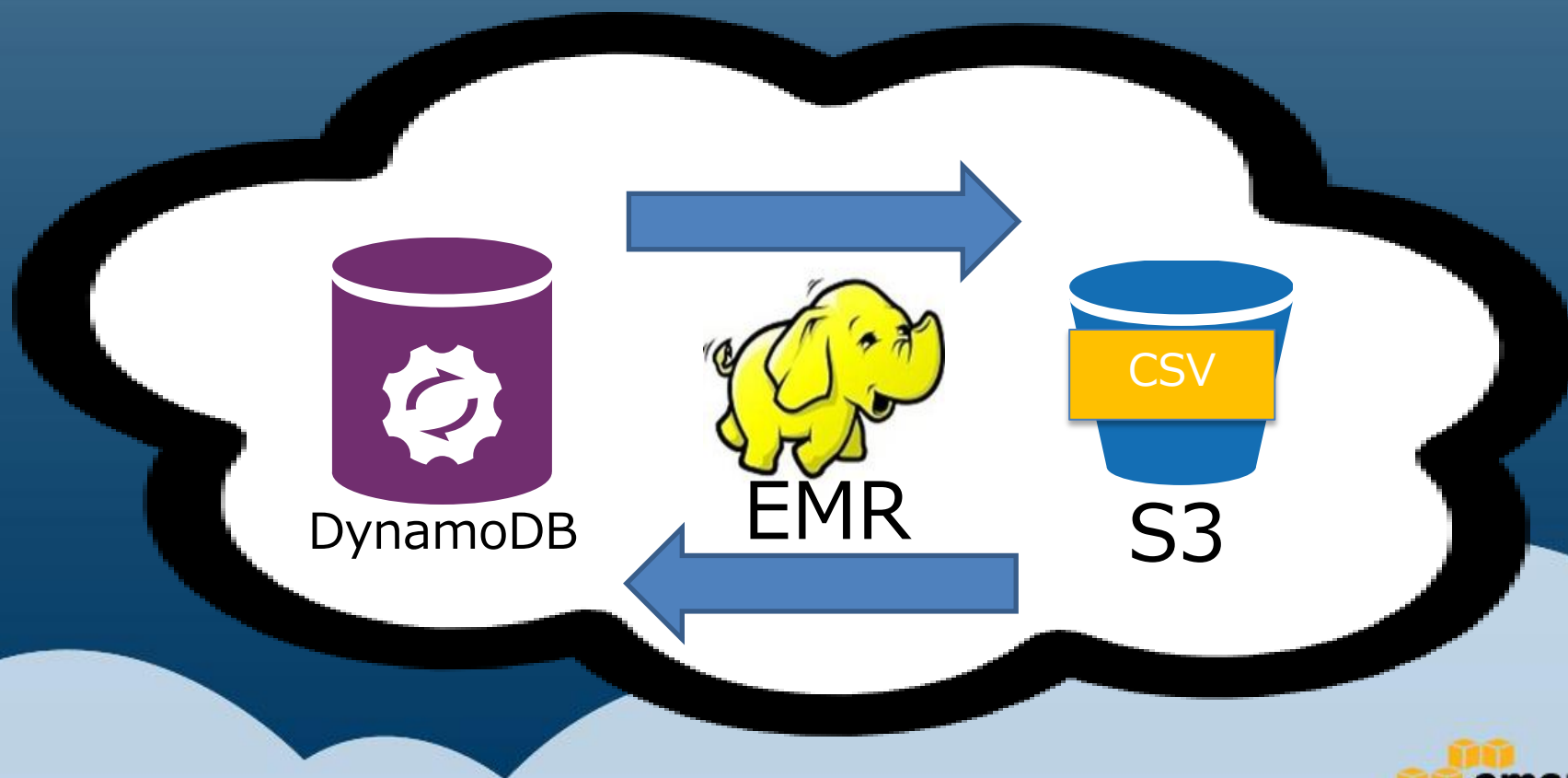
- IAM連携によりアクション・リソース共に可能
- アクション：UpdateできてDeleteできない等
- リソース：テーブル指定

# DynamoDB+EMR連携

- モダンデータベースはHadoop連携が必須
  - Hadoopに即連携できるのがベスト
- Elastic MapReduce=Hadoopサービス
- 小さく大量なデータはDynamoDBで蓄積
- DynamoDBでリアルタイム処理
- EMRでデータをオフロードしバッチ処理

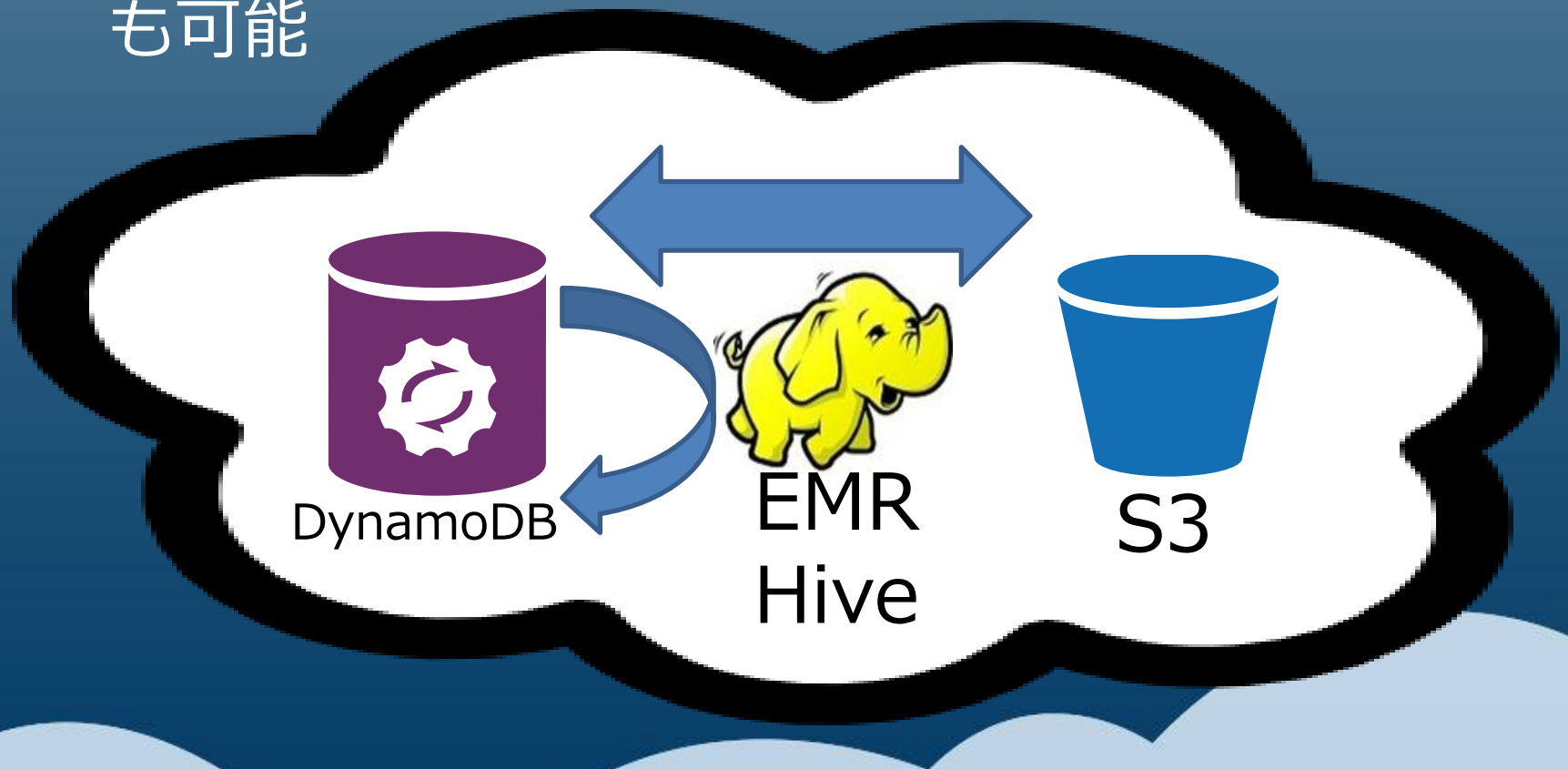
# EMR連携1：データのインポート/エクスポート

- ❏ Elastic MapReduce=Hadoopサービス
- ❏ オープンデータフォーマットであるCSVへ展開



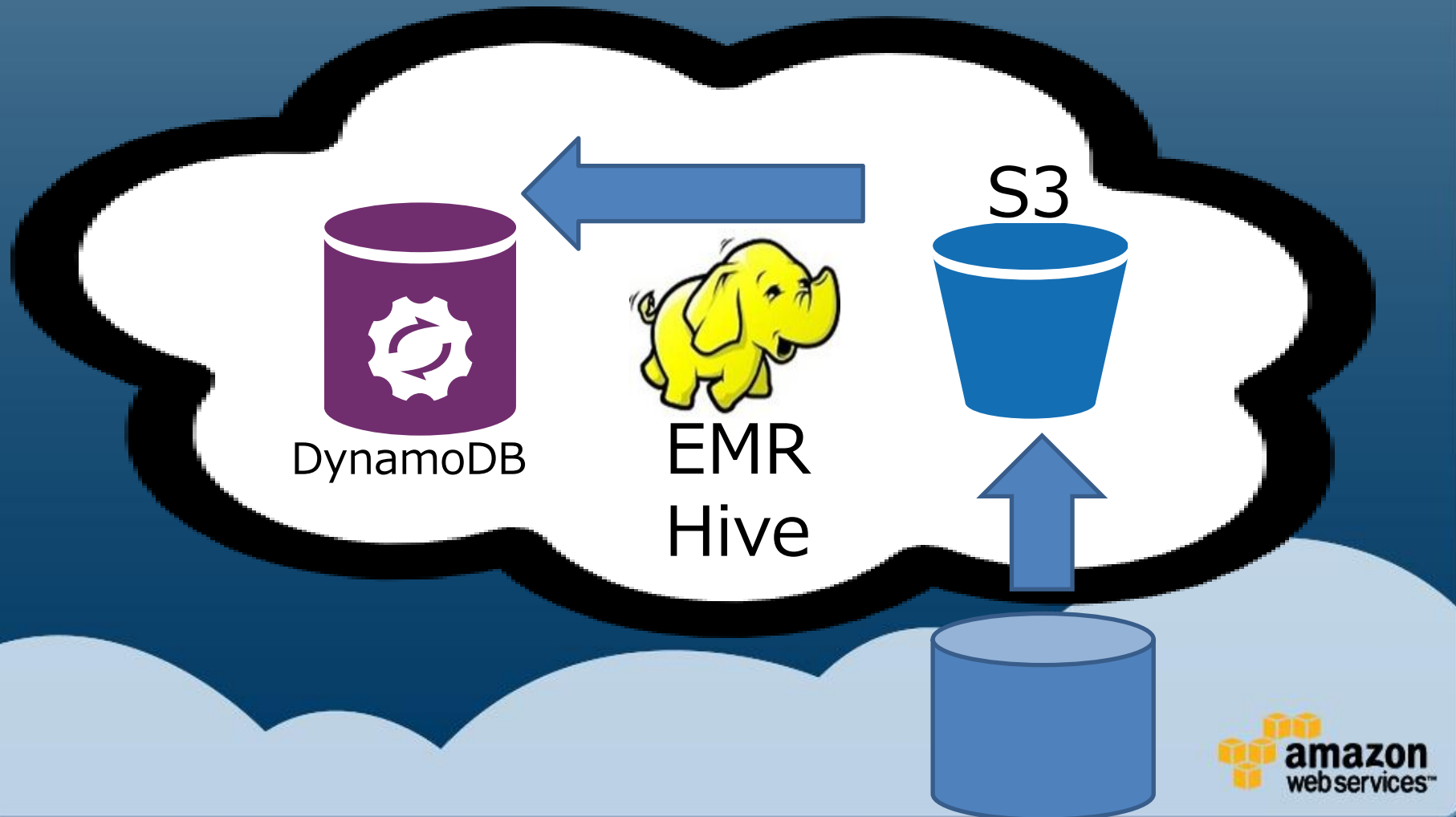
# EMR連携2：データのジョイン

- EMR上のSQLライクな言語(Hive)でのジョインも可能



# EMR連携3：別データベースからの移行

📦 S3経由でDynamoDBへデータをロード



# DynamoDBの制約



## テーブルサイズ

- 制限なし



## プロビジョンスループット

- テーブル単位
  - 10,000 read、10,000 write
- アカウント単位
  - 20,000 read、20,000 write
- 上限は申請により解除可能



# DynamoDBの制約(2)



## プロビジョンスルーブットの更新

- スルーブットを上げる
  - 最低10%、最大200%まで
- 上限を下げる
  - 1テーブル1日1回まで



## データ容量

- 1アイテムは64KBまで(名前と値含む)
- ハッシュキーは2048KBまで
- レンジキーは、1024KBまで

# DynamoDBの コストモデル



# DynamoDBのコストモデル

- 実は非常にわかりやすい、しかも計算しやすい
- コストファクタは下記のみ
  - データストレージ容量
  - プロビジョンスループット指定値
  - データ転送量

# 料金 (東京リージョン利用)

- データ保存容量に応じた料金
  - \$1.20 GBあたり/月
- 読込/書込スループットに応じた料金
  - \$0.012 /時間 50読込みスループットあたり
  - \$0.012 /時間 10書込みスループットあたり
- データ転送量
  - INは無料
  - OUTは\$0.201/GBから

# 料金 (東京リージョン利用)

- 無料使用枠
  - DynamoDB の保存容量 100 MB までは無料
  - さらに書込み最大 5 回/秒、  
読込み最大 10 回/秒のスループットを無料

# 料金試算例(1)

- 10 書込スループット/秒
- 100 読込スループット/秒
- 10GB のデータ
- 1 KB のアイテムサイズ

月額コストは、約**2900**円  
(\$38 - 77円/\$換算)

## 料金試算例(2)

- 100 書込スループット/秒
- 100 読込スループット/秒
- 50GB のデータ
- 1 KB のアイテムサイズ

月額コストは、約**10000**円  
(\$140 - 77円/\$換算)

# DynamoDBでの 開発とTips





# DynamoDBの利用を推奨するケース

- 高速アクセスが必要なアプリケーション
- 新しく構築するプロジェクト
- モバイル、ソーシャルなアプリケーション
- 管理の手間を省き、TCOの削減したい
- 複雑なクエリーが必要でないとき
- ビッグデータにリアルタイムアクセスさせたい場合

# YesSQL(RDS) vs NoSQL(DynamoDB)

下記項目を検討して、使い分け・併用する技術が重要。

要素	YesSQL=RDBMS(RDS)	NoSQL (DynamoDB)
アプリケーションのタイプ	<ul style="list-style-type: none"><li>既存アプリケーション</li><li>ビジネスプロセス中心</li></ul> 例: 金融業務、ERP、複数プロセスを行う業務ワークフロー	<ul style="list-style-type: none"><li>Webスケールが必要なアプリ</li><li>大量の小さな書き込みと読み込み</li></ul> 例: ソーシャルメディア, モバイル、ショッピングカート, ユーザプリファレンス、ビッグデータ
アプリケーションの特徴	<ul style="list-style-type: none"><li>リレーショナルデータモデル</li><li>複雑なクエリ, ジョインと更新</li></ul>	<ul style="list-style-type: none"><li>シンプルなデータモデル</li><li>レンジクエリ, シンプルな更新</li></ul>
スケール	エンジニアがDB構築時に選択 クラスタリング, パーティショニング, シャーディング	シームレス、かつオンデマンドで自動的にスケールする
QoS	<ul style="list-style-type: none"><li>パフォーマンス – RDB要素依存</li><li>信頼性 – RDSが管理</li><li>可用性 – RDSが管理</li><li>堅牢性 – RDSが管理</li></ul>	<ul style="list-style-type: none"><li>パフォーマンス -ユーザが常に調整可能</li><li>信頼性 – DynamoDBが管理</li><li>可用性 – DynamoDBが管理</li><li>堅牢性 – DynamoDBが管理</li></ul>
スキルセット	既存のプログラミングスキル SQL + プログラミング言語	Webスタイルのプログラミング AWS SDKでプログラミング

**AWSではYesSQLからNoSQLまで提供している**

# 余談：NoSQLサービスとプロダクト

- コアビジネスは何か？
- サービス
  - 運用は楽になる
  - ロックイン？
- プロダクト
  - 運用は非常に大変
  - 自分で自動化する必要あり

- サービス
  - DynamoDB、RDS、ElastiCache
- プロダクト
  - EC2上のDB

# お客様がDynamoDBでやること

- ❏ 必要なテーブルを作成する
- ❏ 各テーブル毎に読み書きのIOPSを指定する
- ❏ SDKを使って、アプリケーションを書く

# お客様がDynamoDBでやらなくてよい事

- ❏ テーブルの容量拡張
- ❏ ハードウェア拡張やスケーラビリティの確保
- ❏ データベースの障害管理・フェイルオーバー機能の実装・管理
- ❏ データベースの堅牢性の維持

アプリケーションの開発に集中してほしい

# DynamoDBで開発するには

- 📦 SDKを使う
- 📦 APIをたたく
- 📦 サードパーティライブラリを使う

# DynamoDB SDK

## Java

- <http://aws.amazon.com/sdkforjava/>

## PHP

- <http://aws.amazon.com/sdkforphp/>

## Ruby

- <http://aws.amazon.com/sdkforruby/>

## .NET

- <http://aws.amazon.com/sdkfornet/>

# 例：DynamoDBのJava SDK

## 低レベル

- AmazonDynamoDBClient
- AmazonDynamoDBAsyncClient

## 高レベル

- DynamoDBMapperフレームワーク



# AmazonDynamoDBClientの作成

```
AmazonDynamoDBClient client =  
    new AmazonDynamoDBClient(credentials);
```

```
// エンドポイントをセット
```

```
client.setEndpoint("https://dynamodb.ap-  
northeast-1.amazonaws.com");
```

[http://docs.amazonwebservices.com/general/latest/gr/rande.html#ddb\\_region](http://docs.amazonwebservices.com/general/latest/gr/rande.html#ddb_region)

# テーブルの作成

**Create Table**Cancel X

**PRIMARY KEY**

PROVISIONED  
THROUGHPUT CAPACITY

THROUGHPUT ALARMS  
(optional)

**Table Name:**   
Table will be created in ap-northeast-1 region

**Primary Key:**

DynamoDB is a schema-less database. You only need to tell us your primary key attribute(s).

Primary Key Type: ☐ Hash ☒ Hash and Range

☐ String ☒ Number

Hash Attribute Name:

☐ String ☒ Number

Range Attribute Name:

 Choose a hash attribute that ensures that your workload is evenly distributed across hash keys.  
For example, "Customer ID" is a good hash key, while "Game ID" would be a bad choice if most of your traffic relates to a few popular games.  
[Learn more about choosing your primary key](#)

CancelContinue Help 

# テーブルの作成

```
AmazonDynamoDB client = createClient();  
//ハッシュキー、レンジキーの作成とセット  
KeySchema keySchema = new KeySchema();  
    keySchema.setHashKeyElement(hashKey);  
    keySchema.setRangeKeyElement(rangeKey);  
ProvisionedThroughput throughput = //プロビジョンスループット作成  
    throughput.setReadCapacityUnits(20L); //読み込みで20  
    throughput.setWriteCapacityUnits(20L); //書き込みで20  
CreateTableRequest request = ... //テーブル生成のリクエスト作成  
request.setProvisionedThroughput(throughput);  
CreateTableResult result = client.createTable(request);
```

# テーブルの作成

 テーブル名とプライマリーキーを作成

アイテム

プライマリーキー

ハッシュ  
キー

レンジキー

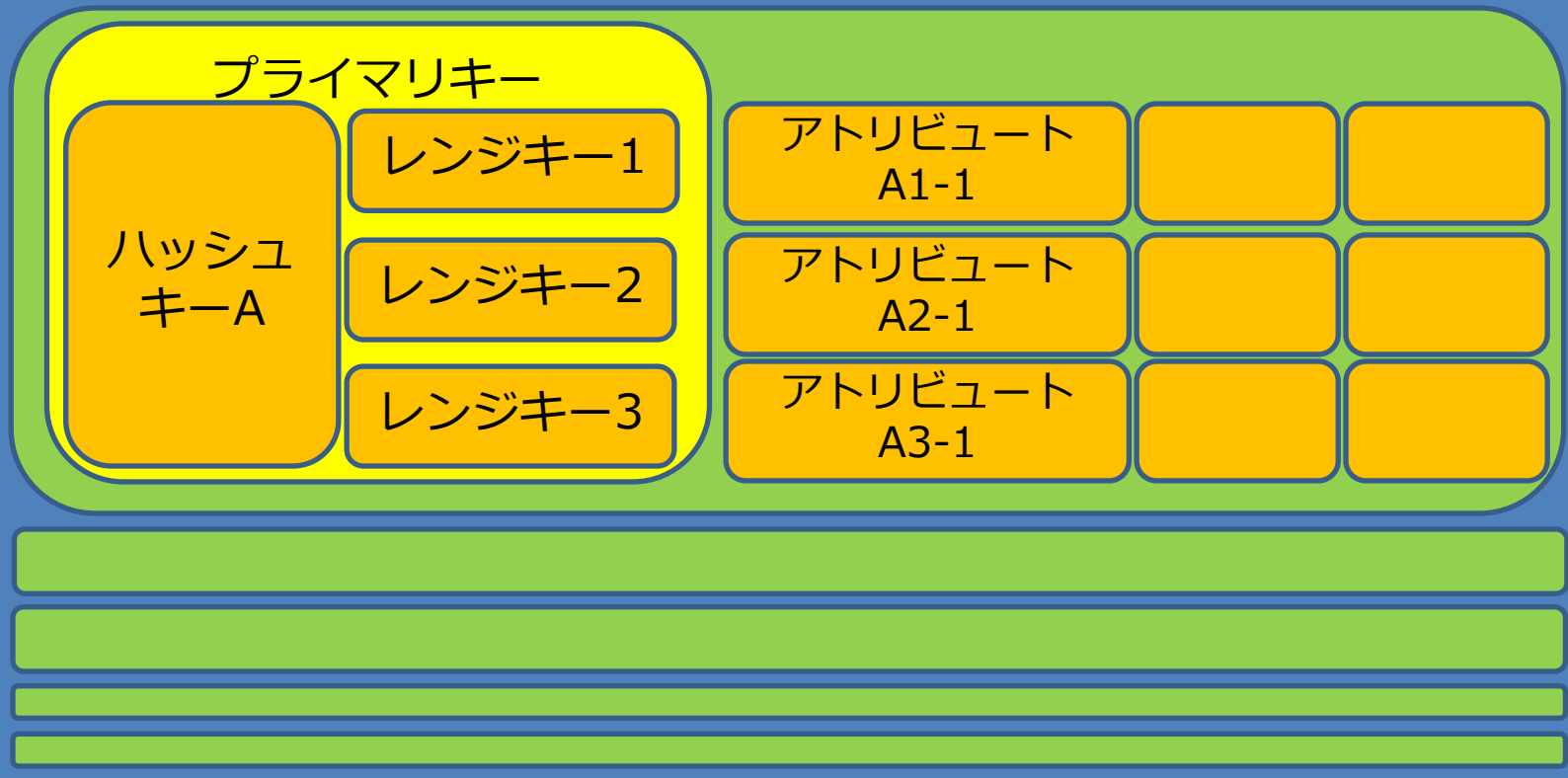
# アイテムの挿入

```
Map<String, AttributeValue> item =  
    new HashMap<String, AttributeValue>();  
{  
    item.put("id", new AttributeValue().withN(String.valueOf(i)));  
    item.put("time", new AttributeValue().withN(String.valueOf(t)));  
    item.put("firstname", new AttributeValue("firstname" + i));  
    item.put("lastname", new AttributeValue("lastname" + i));  
    item.put("role", new AttributeValue().withSS("Sales"));  
}  
PutItemRequest putItemRequest =  
    new PutItemRequest(tableName, item);  
client.putItem(putItemRequest);
```

# データの挿入

 アトリビュートに何を入れるかはアイテム毎で自由

## アイテム



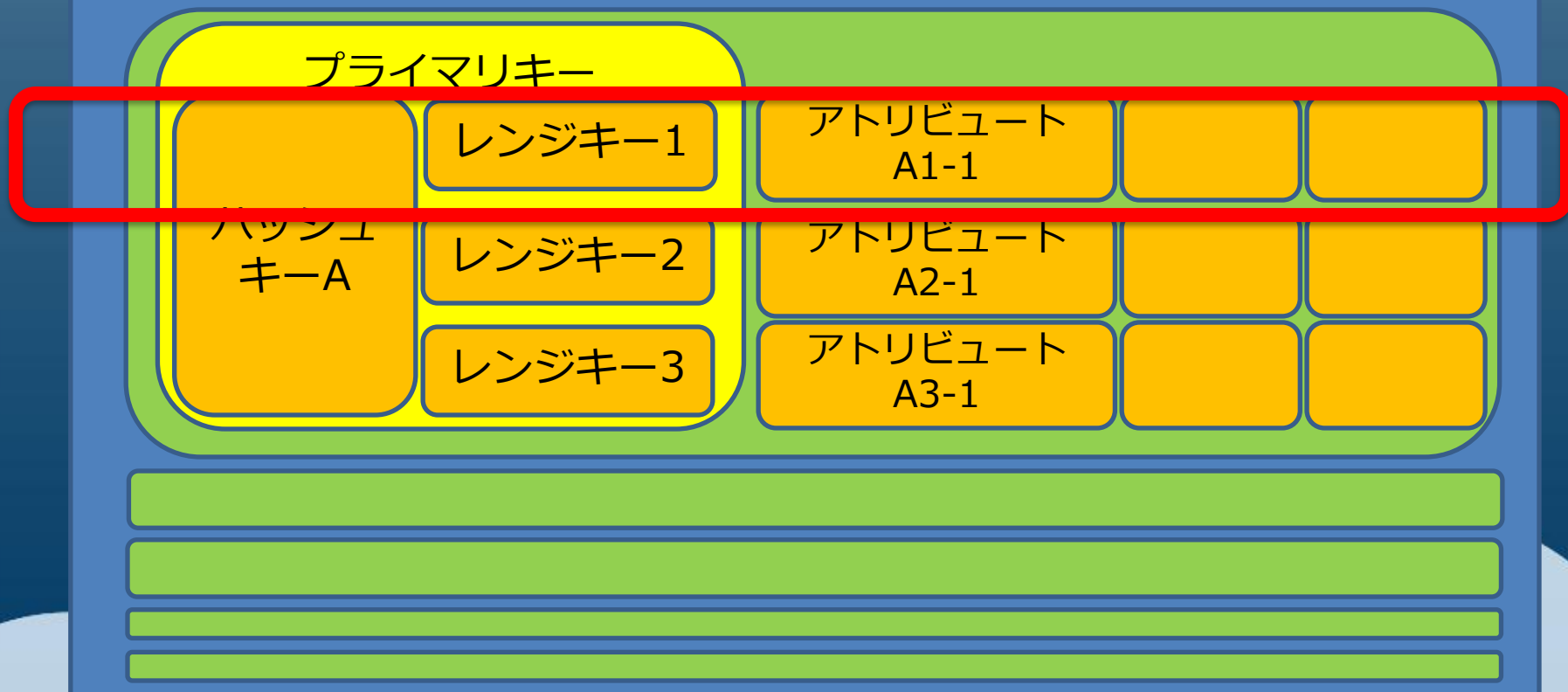
# アイテムの取得

```
Key key = new Key();
key.setHashKeyElement(new AttributeValue().withN("1"));
key.setRangeKeyElement(
    new AttributeValue().withN(String.valueOf(time)));
GetItemRequest getItemRequest =
    new GetItemRequest(tableName, key);
GetItemResult item = client.getItem(getItemRequest);
Map<String, AttributeValue> result = item.getItem();
for (Map.Entry<String, AttributeValue> e : result.entrySet()) {
    System.out.println(e.getKey() + " : " + e.getValue());
}
```

# DynamoDBからのGET

 ハッシュキー+レンジキー指定

アイテム





# アイテムの更新

```
Key k = new Key().withHashKeyElement(  
    new AttributeValue().withN("4")).withRangeKeyElement(  
    new AttributeValue().withN(dateString));  
Map<String, AttributeValueUpdate> updates =  
    new HashMap<String, AttributeValueUpdate>();  
updates.put("firstname", new AttributeValueUpdate()  
    .withValue(new AttributeValue().withS("edited_firstname4")));  
UpdateItemRequest updateItemRequest =  
    new UpdateItemRequest().withTableName(tableName)  
    .withReturnValues(ReturnValue.ALL_NEW)  
    .withKey(k).withAttributeUpdates(attributeUpdates);  
UpdateItemResult result = client.updateItem(updateItemRequest);
```

# アイテムのスキャン

//フルスキャンはテーブル全部検索する

```
ScanRequest scanRequest = new ScanRequest()
    .withTableName(tableName);
ScanResult fullScanResult = client.scan(scanRequest);
List<Map<String, AttributeValue>> items =
    fullScanResult.getItems();
for (Map<String, AttributeValue> map : items) {
    for (Map.Entry<String, AttributeValue> e : map.entrySet()) {
        System.out.println(String.format("%s, %s", e.getKey(),
            e.getValue()));
    }
    System.out.println("");
}
```

# DynamoDBのSCAN

 テーブルから総取得

アイテム



# アイテムのクエリ

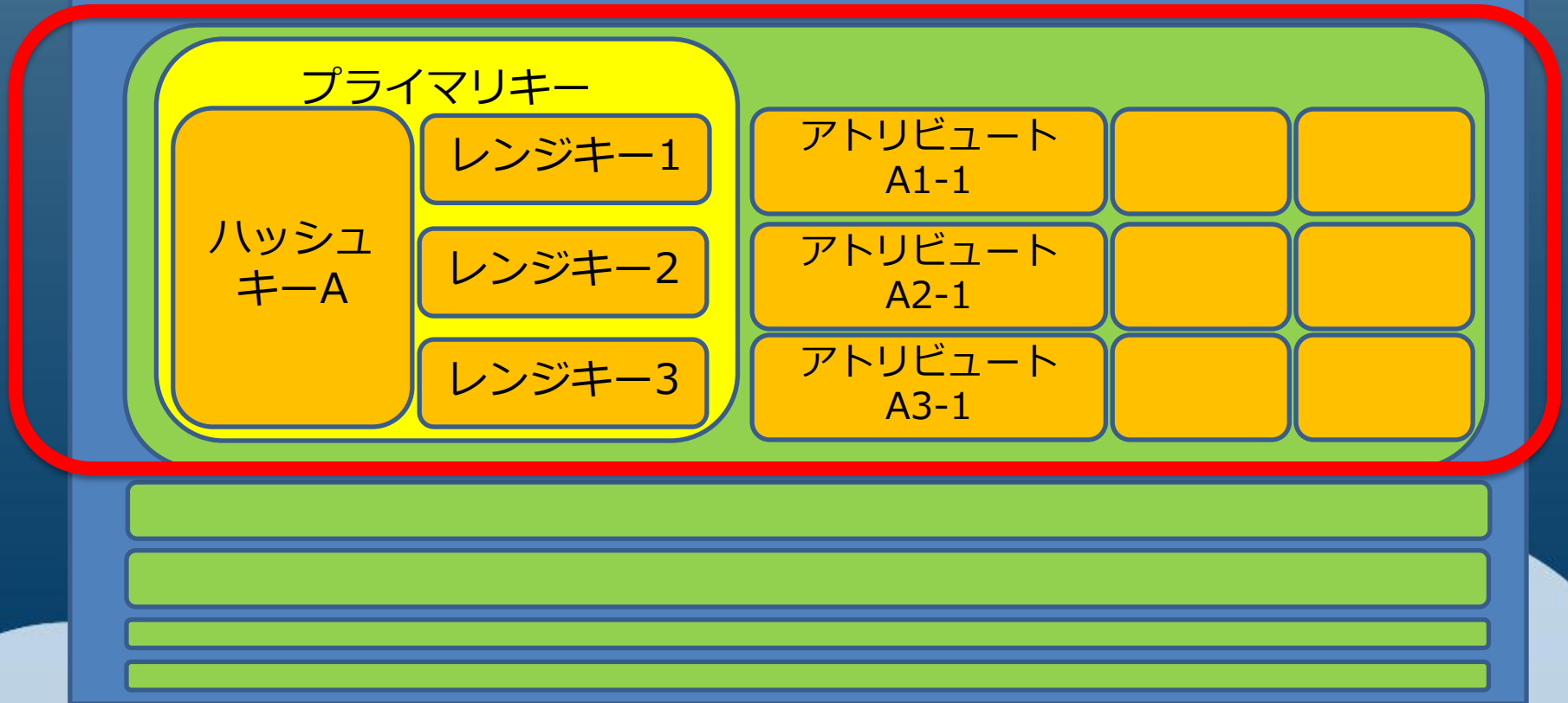
//ハッシュキー(ID)が4のアイテムを抜き出す。レンジキーは指定しない

```
QueryRequest q = new QueryRequest(tableName,  
                                new AttributeValue().withN("4"));  
QueryResult qr = client.query(q);  
for (Map<String, AttributeValue> items : qr.getItems()) {  
    for (Map.Entry<String, AttributeValue> item : items.entrySet())  
    {  
        AttributeValue value = item.getValue();  
        System.out.println(item.getKey() + " : " + value.getS());  
    }  
    System.out.println("");  
}
```

# DynamoDBのQUERY

## ハッシュキーのみ指定

### アイテム



# DynamoDBMapper

📦 シンプルなクラスをデータモデルとみなして処理可能

📦 アノテーションベースでデータを操作可能

- `@DynamoDBTable(tableName = "employee")`
- `@DynamoDBHashKey(attributeName = "id")`
- `@DynamoDBRangeKey(attributeName = "time")`
- `@DynamoDBAttribute`

📦 楽観的ロック機能

- `@DynamoDBVersionAttribute`

# DynamoDBMapper Model

```
@DynamoDBTable(tableName = "employee")
```

```
public class Employee {
```

```
    @DynamoDBHashKey(attributeName = "id")
```

```
    public long getId() { return id; }
```

```
    @DynamoDBRangeKey(attributeName = "time")
```

```
    public long getTime() { return time; }
```

```
    @DynamoDBAttribute(attributeName = "firstname")
```

```
    public String getFirstname() { return firstname; }
```

```
    @DynamoDBVersionAttribute
```

```
    public Long getVersion() { return version; }
```

```
}
```

# DynamoDBMapperの操作

```
DynamoDBMapper mapper = new DynamoDBMapper(client);  
//従業員を取得  
Employee e = mapper.load(Employee.class, hashkey, rkey);
```

```
//従業員を挿入  
Employee emp = new Employee();  
emp.setId(1000);  
emp.setTime(t);  
emp.setFirstname("HLFirstname");  
emp.setLastname("HLLastname");  
emp.setRole(asSet("TechSales"));  
mapper.save(emp);
```

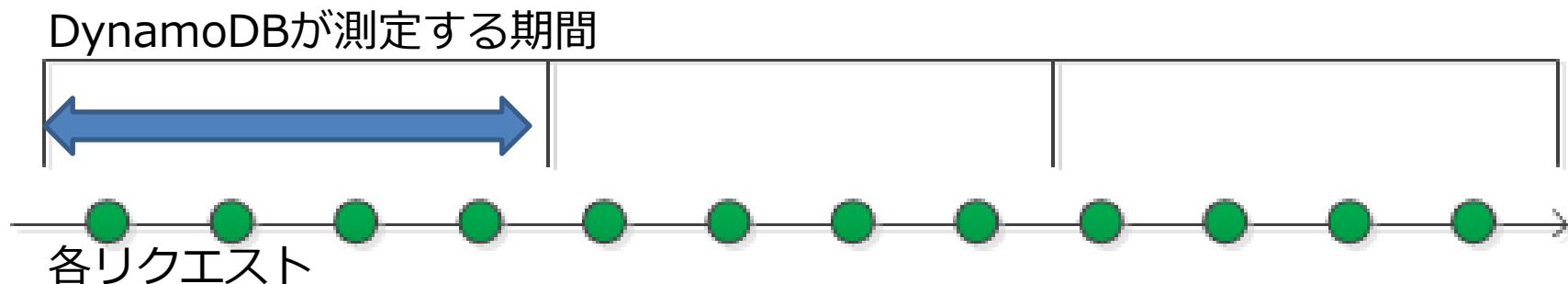


# DynamoDBでスケールさせるには

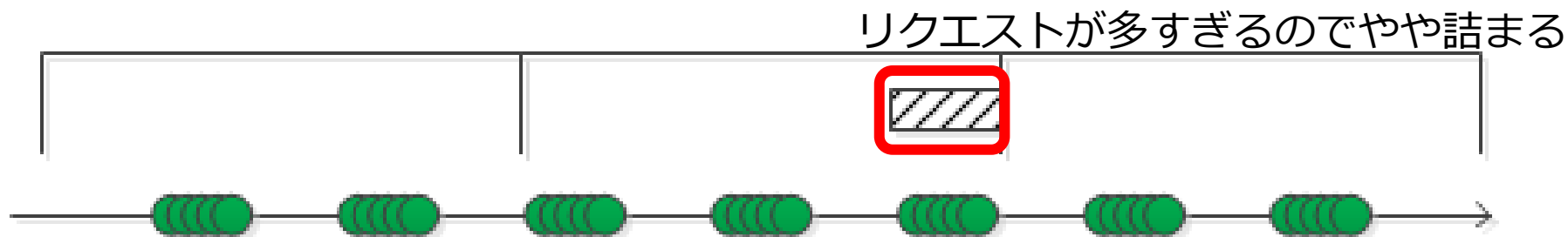
- プライマリキーできちんと分散させる
  - ポイントはハッシュキーで適度に分散するように設計する事
- アイテムの大きさを大きくしすぎない
  - 大きなデータはS3へ置く
- Scanはなるべく避ける

# DynamoDBでスケールさせるには(良い例)

- 理想ケース：リクエストとそのサイズが均等

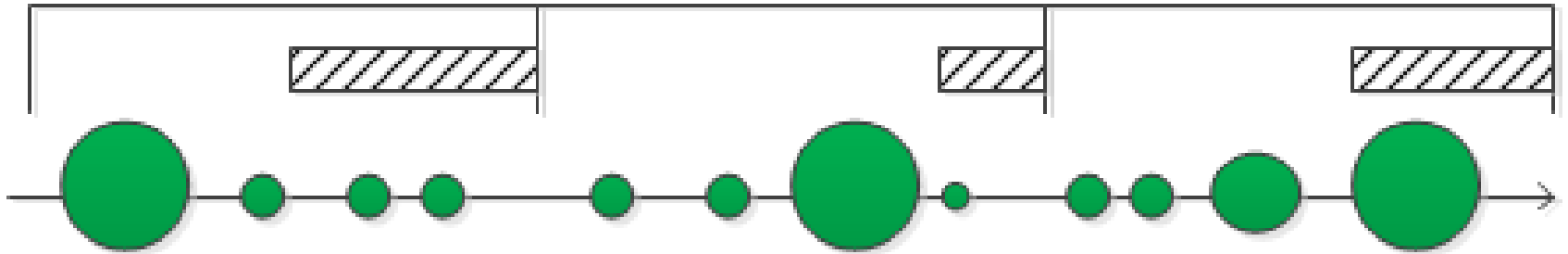


- 良好ケース：サイズは均等ただしバーストあり

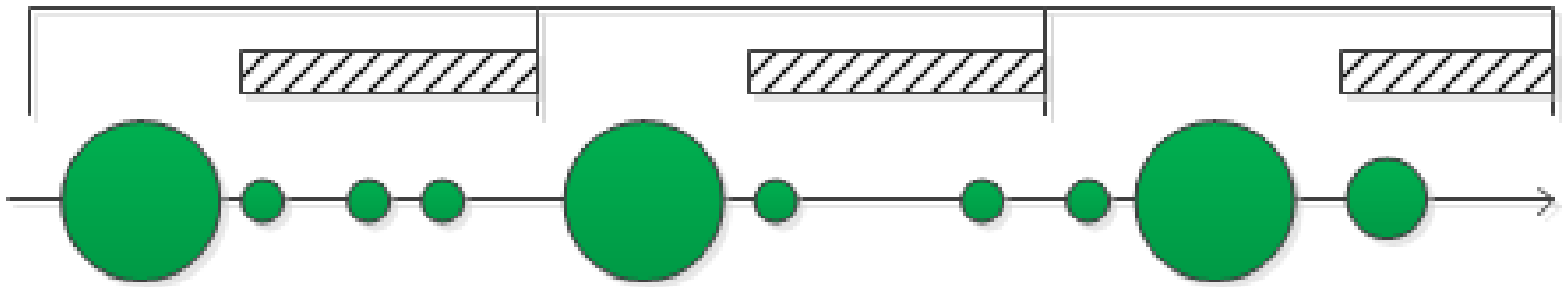


# DynamoDBでスケールさせるには(悪い例)

- ・ 悪いケース1：リクエストサイズのばらつき



- ・ 悪いケース2：頻繁な大量のテーブル全体検索



# DynamoDBを使った設計方法

- キーをいかに分散するか
- データ構造が柔軟なのを活かす
  - データの保存方法よりクエリ重視
    - クエリ毎にデータを持つ方法
    - データは検索されてこそ
- コンポジットキー
- 一貫性とスケールのトレードオフ
- とはいえDynamoDBは力技も効く

# DynamoDBと エコシステム



# 広がるDynamoDBのエコシステム

## 1. Datomic

1. <http://datomic.com/>

## 2. 各種オープンソースライブラリ

1. Scala

2. Node.js

3. Python

## 3. もちろんAWS SDKも！

# DynamoDBを使うためのライブラリまとめ

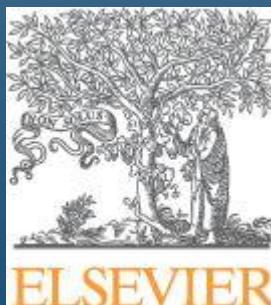
言語	ライブラリ	マッパー	モック
ColdFusion	<a href="#">CFDynamo</a>		
Django	<a href="#">django-dynamodb-sessions</a>		
Erlang	<a href="#">DinerI</a>		
Java	<a href="#">AWS SDK for Java</a>	<a href="#">jsoda</a> <a href="#">phoebe-dynamodb</a>	
.NET	<a href="#">AWS SDK for .NET</a>		
Node.js	<a href="#">Dynode</a> <a href="#">awssum</a> <a href="#">dynamo</a>		
Perl	<a href="#">Net::Amazon::DynamoDB</a>		
PHP	<a href="#">AWS SDK for PHP</a>		
Python	<a href="#">Boto</a>	<a href="#">DynamoDB-Mapper</a>	
Ruby	<a href="#">AWS SDK for Ruby</a>	<a href="#">mince_dynamo_db</a> <a href="#">Dynamoid</a>	<a href="#">fake_dynamo</a> <a href="#">clientside_dynamo_db</a>

# DynamoDB利用事例





# DynamoDBの利用事例



# Amazon DynamoDBとは？

- ・フルマネージドなNoSQLデータベース
- ・超高速・予測可能な一貫したパフォーマンス
- ・シームレスなスケラビリティ、そして低コスト

## Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Voshall 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 customers trust. The Amazon.com platform, which provides service for many websites 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.com is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon.com 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.

運用管理必要なし

低レイテンシ、SSD

プロビジョンスループット

無限に使えるストレージ



# DynamoDBを始めるには？

- <http://aws.amazon.com/dynamodb/>
- 開発者ガイド
  - <http://bit.ly/ddbguide>