Purpose-built Database へのいざない

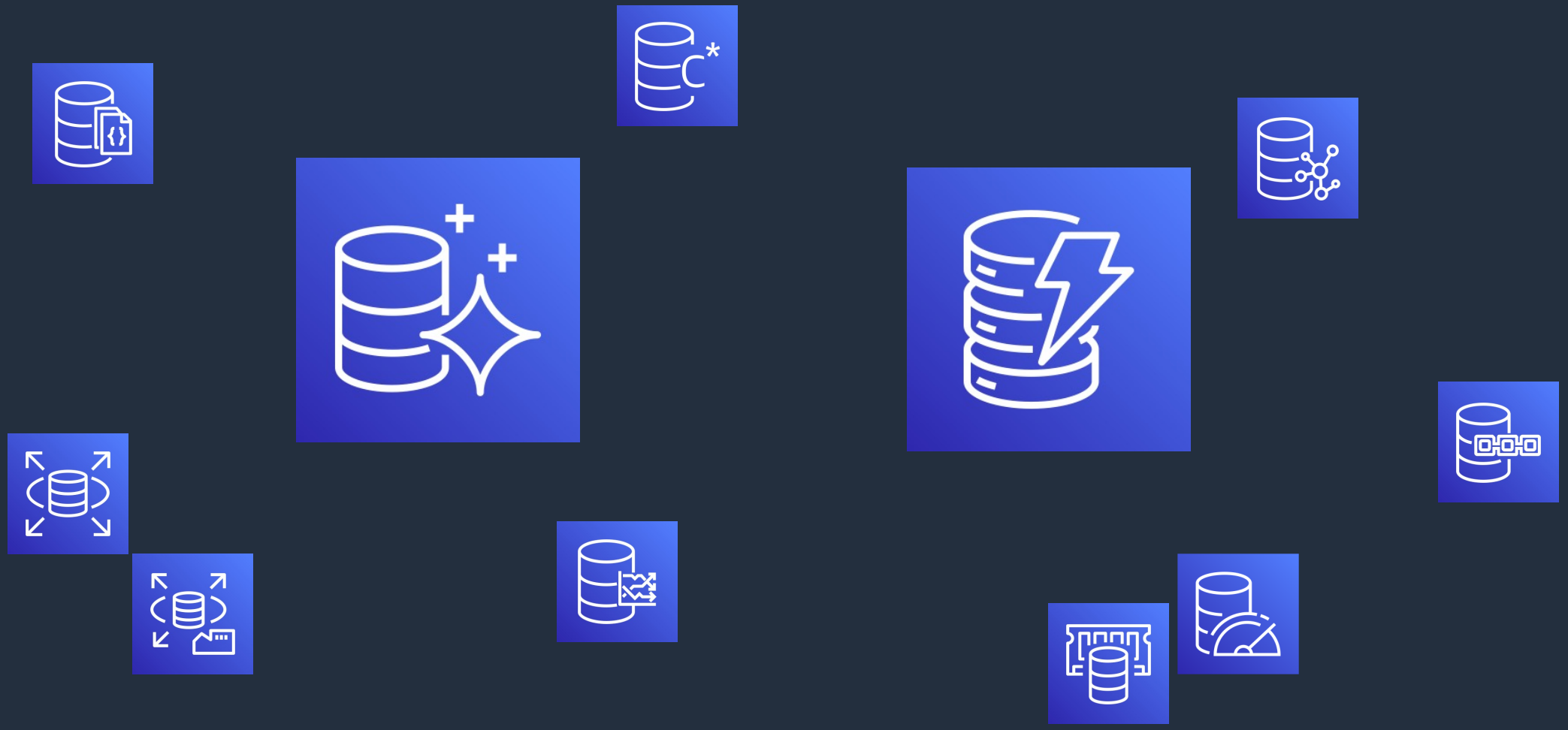
Akihiro Kuwano

自己紹介

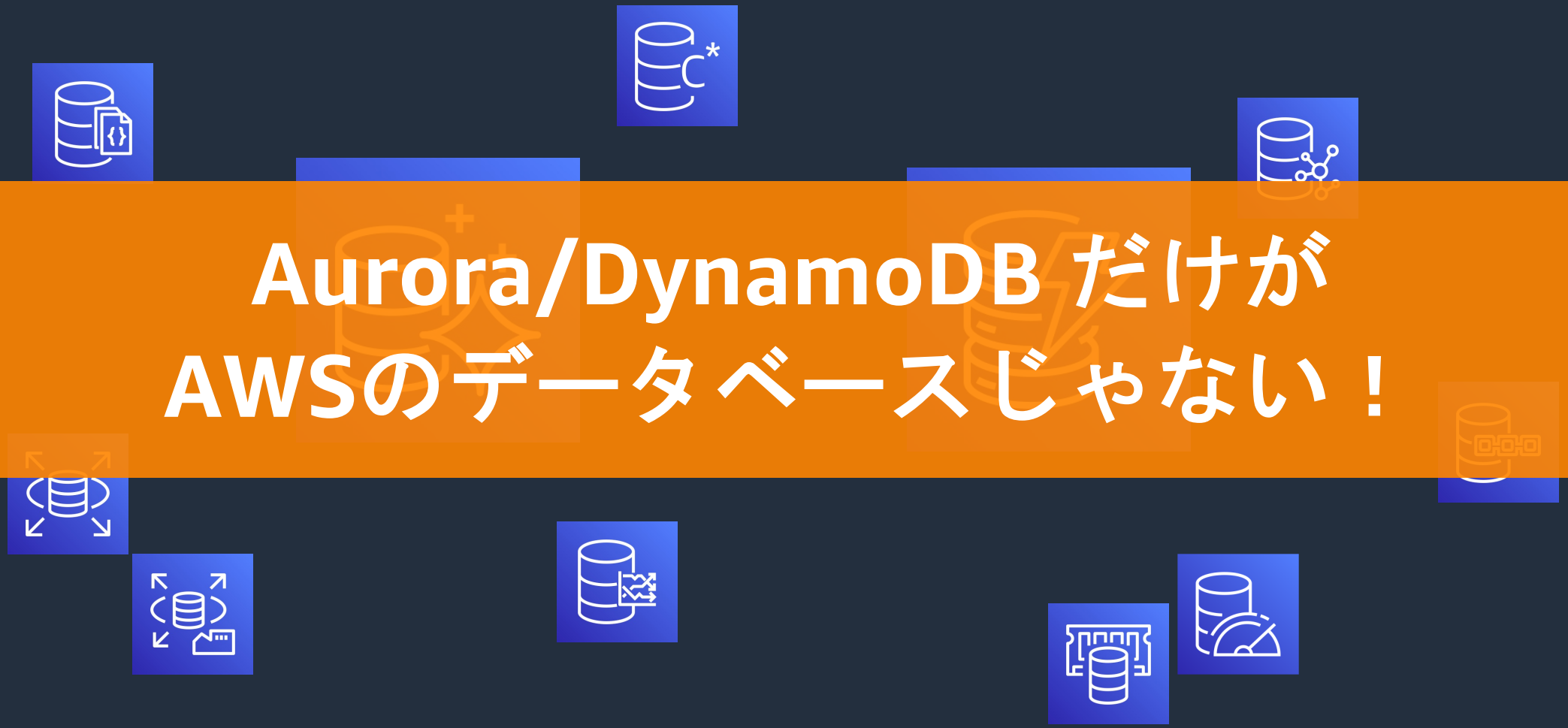
- 桑野 章弘(くわの あきひろ)
  - Amazon Web Services Japan k.k. 所属
  - 連絡先： kuwanoa@amazon.co.jp
  - Twitter : @kuwa_tw
- ソリューションアーキテクト
  - Amazon DocumentDBを始めとしたデータベース
  - 担当
- 前職
  - サイバーエージェント
- 趣味嗜好
  - 子育て（二人の子持ち）
  - ゲーム（Youtubeでマイクラの知識だけがついていくがプレイする時間が欲しい、、、）
  - インフラ全般（昔は自作サーバ等もやってました）

Agenda

- Aurora/DynamoDB だけがAWSのデータベースじゃない！
- Purpose-built Databaseとは
- じゃあどんなときに使えばいいの？
- どうやって選べばいいの？
- まとめ



めっちゃある



Aurora/DynamoDB だけが AWSのデータベースじゃない！

Purpose-built Databaseとは

最新のアプリケーション要件

より多くのパフォーマンス、拡張性、可用性が必要

電子商取引

メディア
ストリーミング

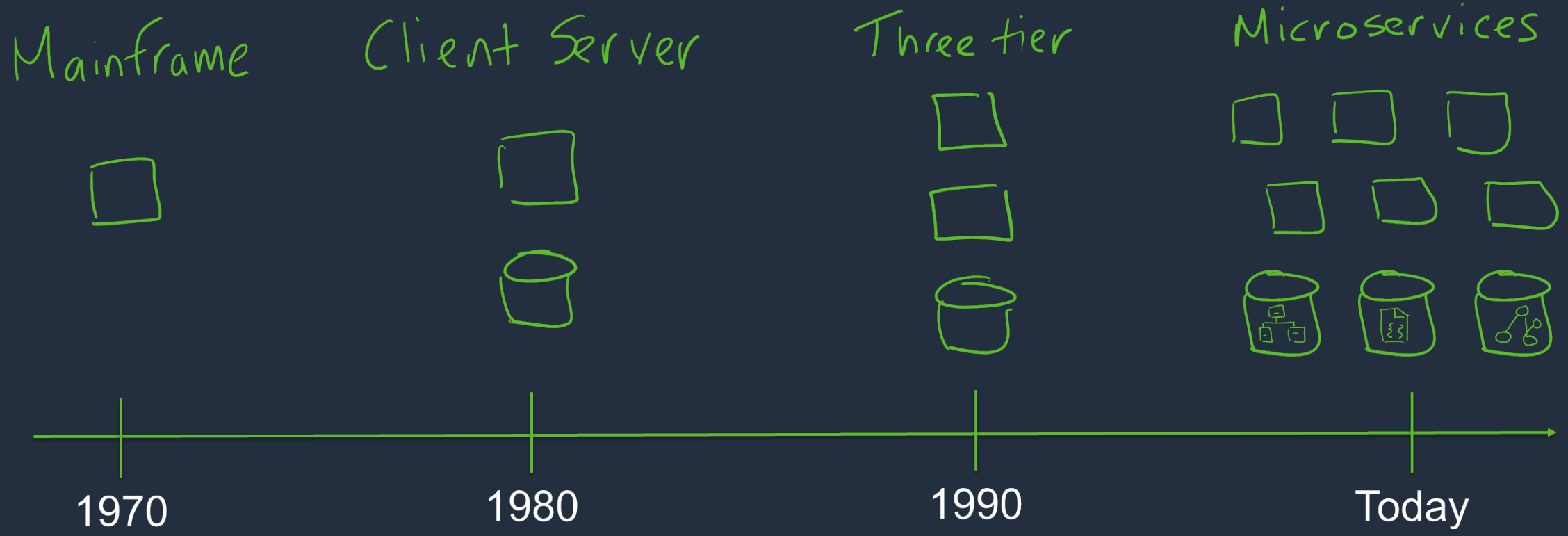
ソーシャル
メディア

オンライン
ゲーム

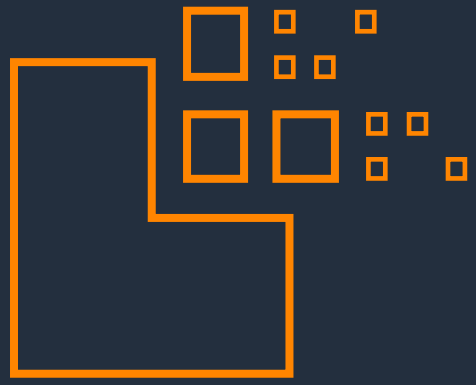
共有経済

アクセスユーザー	1M+
データボリューム	テラバイト、ペタバイト
アクセス元	世界中から
パフォーマンス	ミリ～マイクロ秒遅延
リクエストレート	毎秒百万
交通アクセス	ウェブ、モバイル、IoT、デバイス
スケール	スケールアップ/ダウン、スケールアウト/イン
事業マネタイズ	従量制課金制
開発者アクセス	API アクセス
アプリの変更頻度	毎週やそれ以下の場合も

アプリのアーキテクチャとパターンも進化



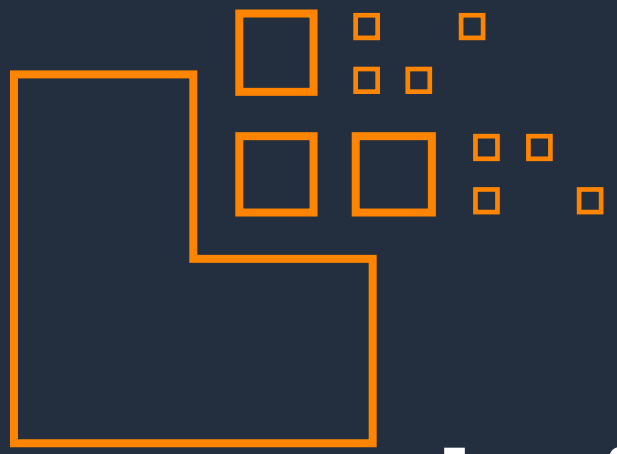
Purpose built



Purpose-built databases

When You’re a Hammer, Everything Looks Like a Nail

データベースの選択



Purpose built

The right tool for the right job

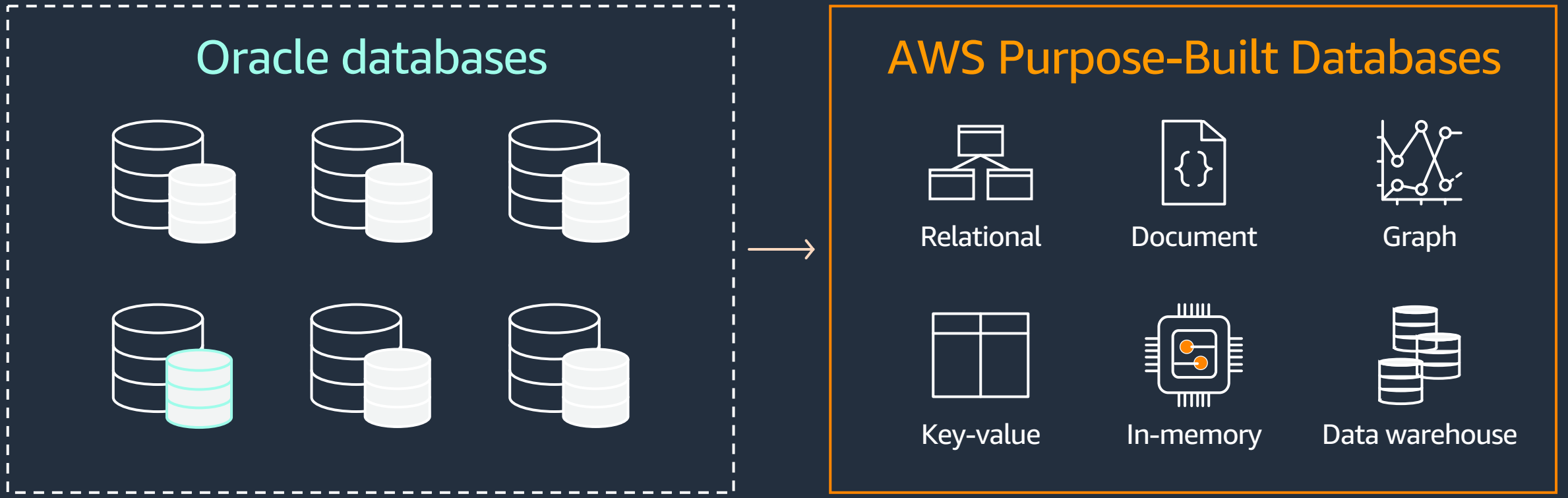
- AWS では多様なデータベースの選択肢
- ワークロードに応じて最適な選択が可能

適材適所の選択

https://www.allthingsdistributed.com/2018/06/purpose-built-databases-in-aws.html

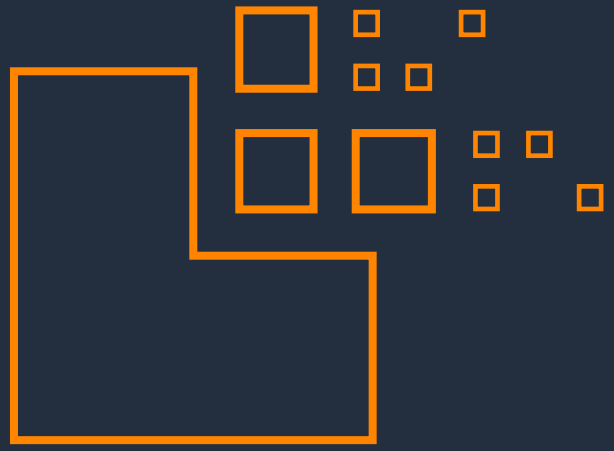
https://www.allthingsdistributed.com/2018/06/purpose-built-databases-in-aws.html

Amazon.com はどのような選択をしたのか



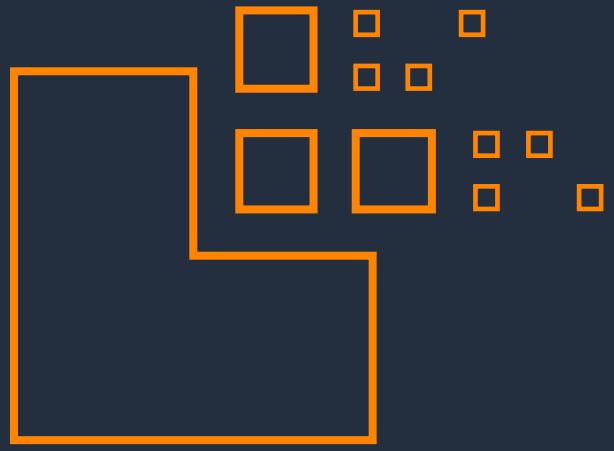
コスト削減、パフォーマンス向上、
より速い革新を実現するために、2016年から
すべてのOracleをAWSに移行開始

Purpose-Built Databases によって
ワークロードに最適なDBエンジンを提供でき、
コストとユーザー体験の最適化を実現

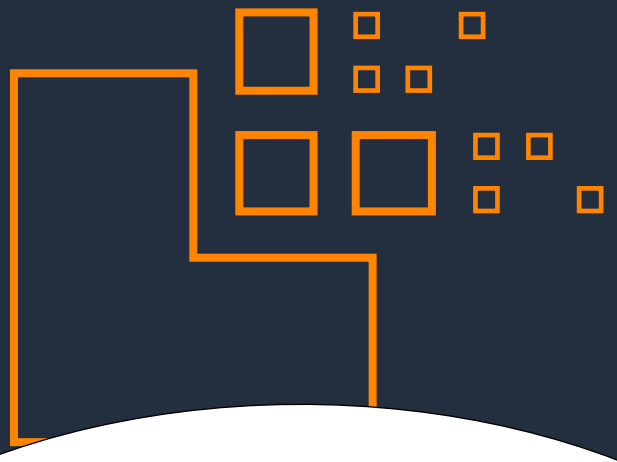


A one size fits all database doesn't fit anyone!
(一つのデータベースでは全てにフィットしない！)

じゃあどんなときに使えばいいの？



A one size fits all database doesn't fit anyone!
(一つのデータベースでは全てにフィットしない！)



A one size fits all database doesn't fit anyone!
(一つのデータベースでは全てにフィットしない！)

**ワーナーさん、
じゃあどんなかんじに
フィットしないの！？**

実際にフィットしない例

もっとスケールしたい！

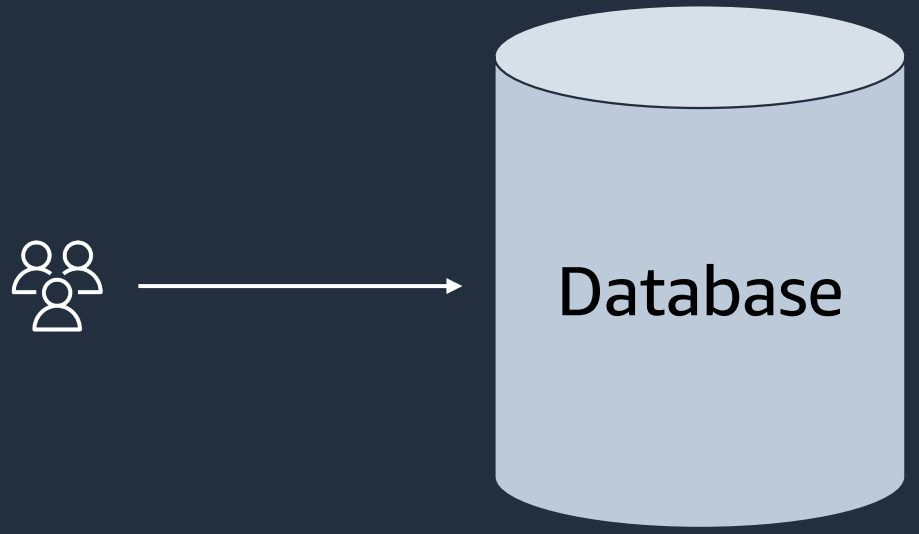
もっと低いレスポンスタイムがほしい！

多ノードを経由するルートの計算がしたい！（分かりづらい）

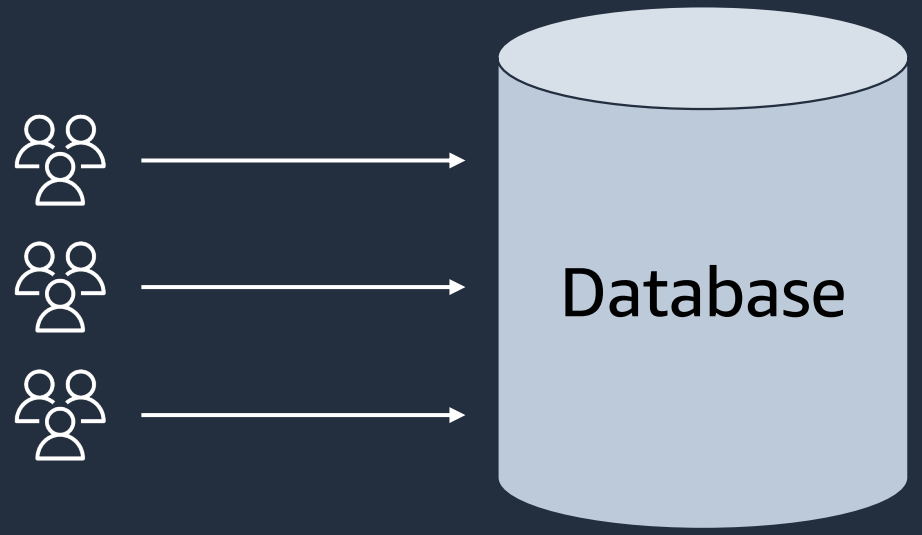
少しづつ項目の違うデータを扱いたい/柔軟にデータを扱いたい！

大量のノードからのCPU監視情報を扱いたい！

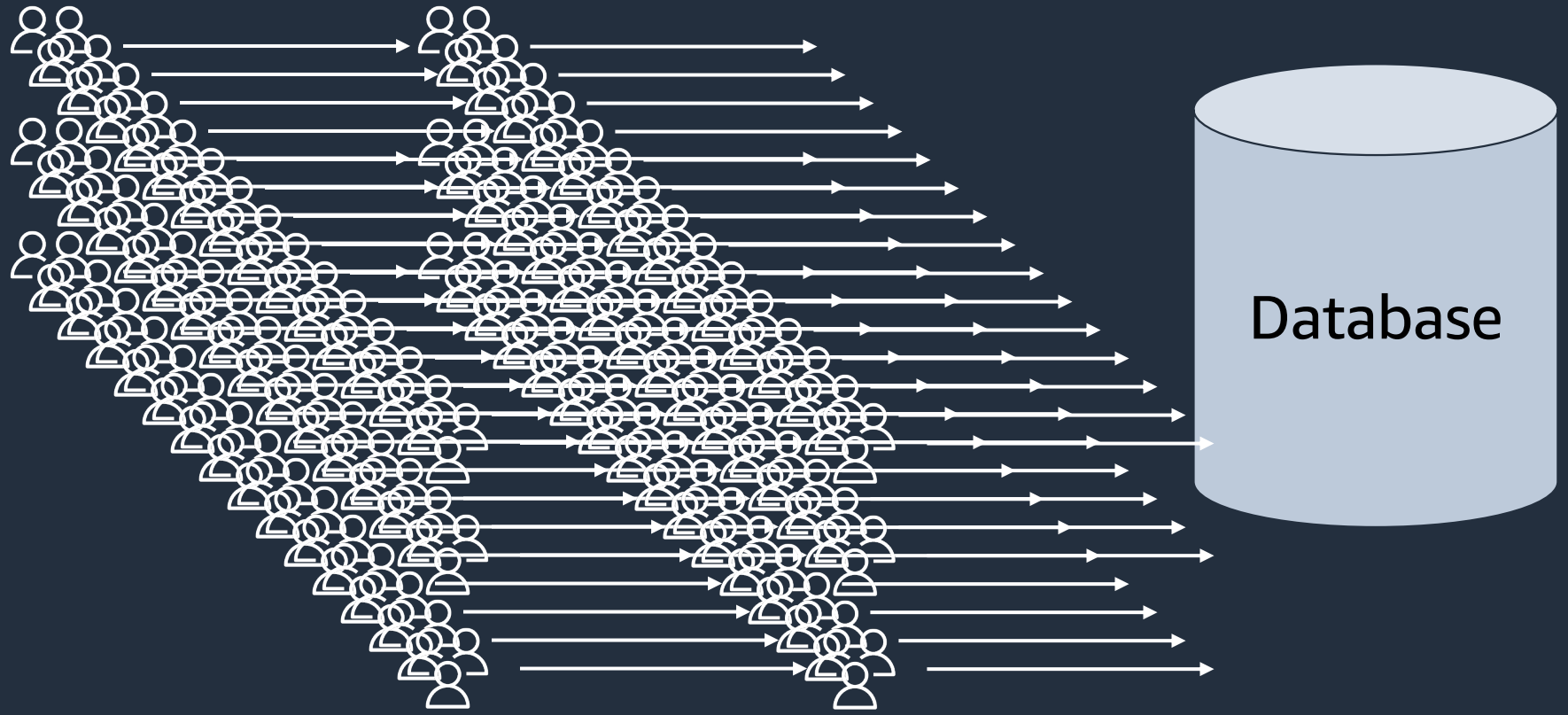
もっとスケールしたい！



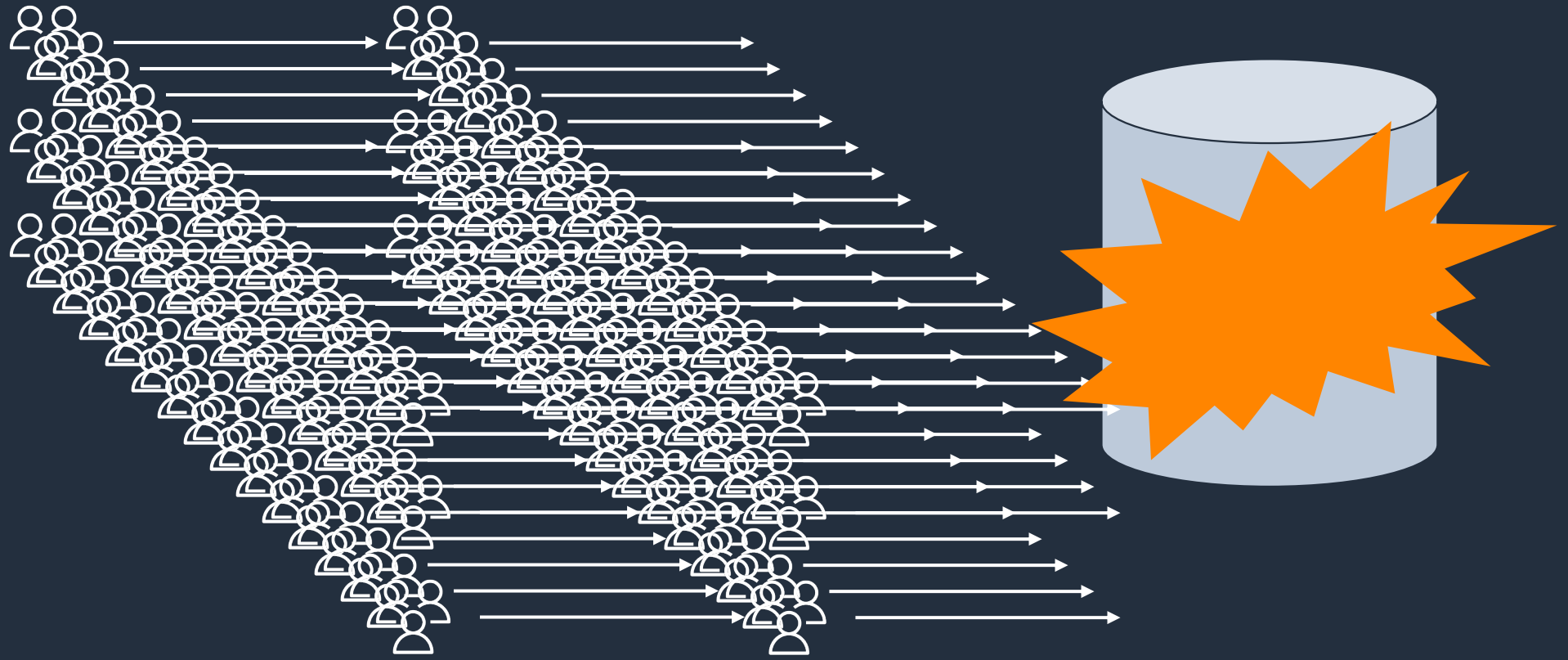
もっとスケールしたい！



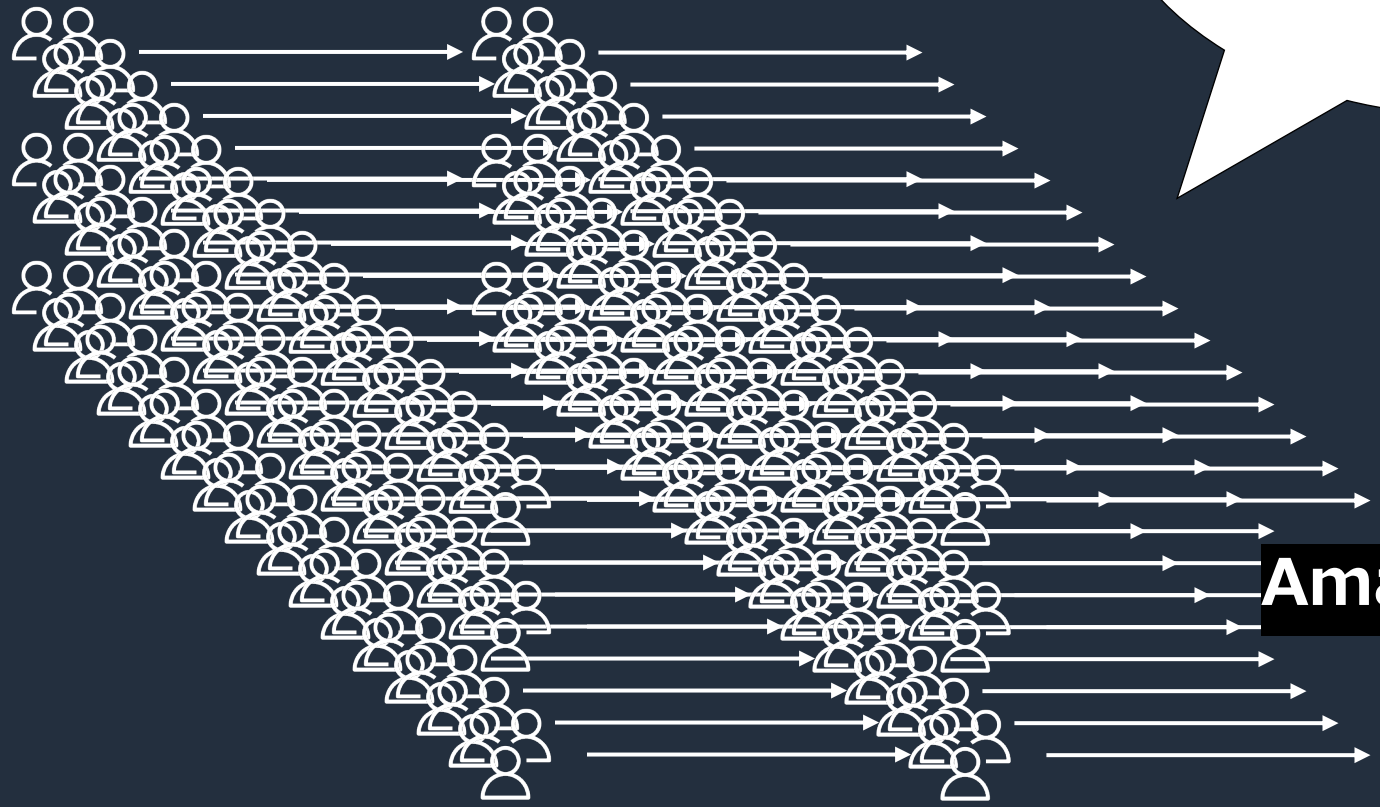
もっとスケールしたい！



もっとスケールしたい！



もっとスケールしたい！

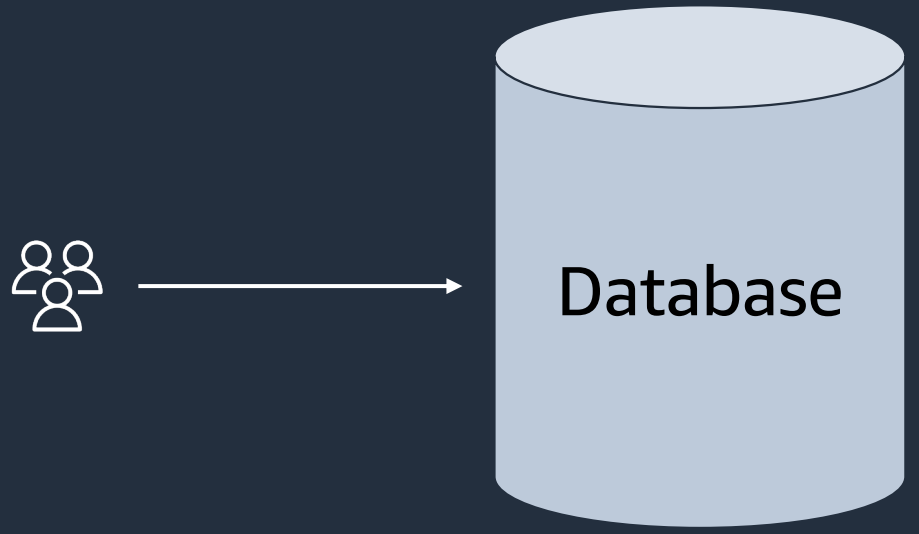


超大量のアクセス
にもスケール可能

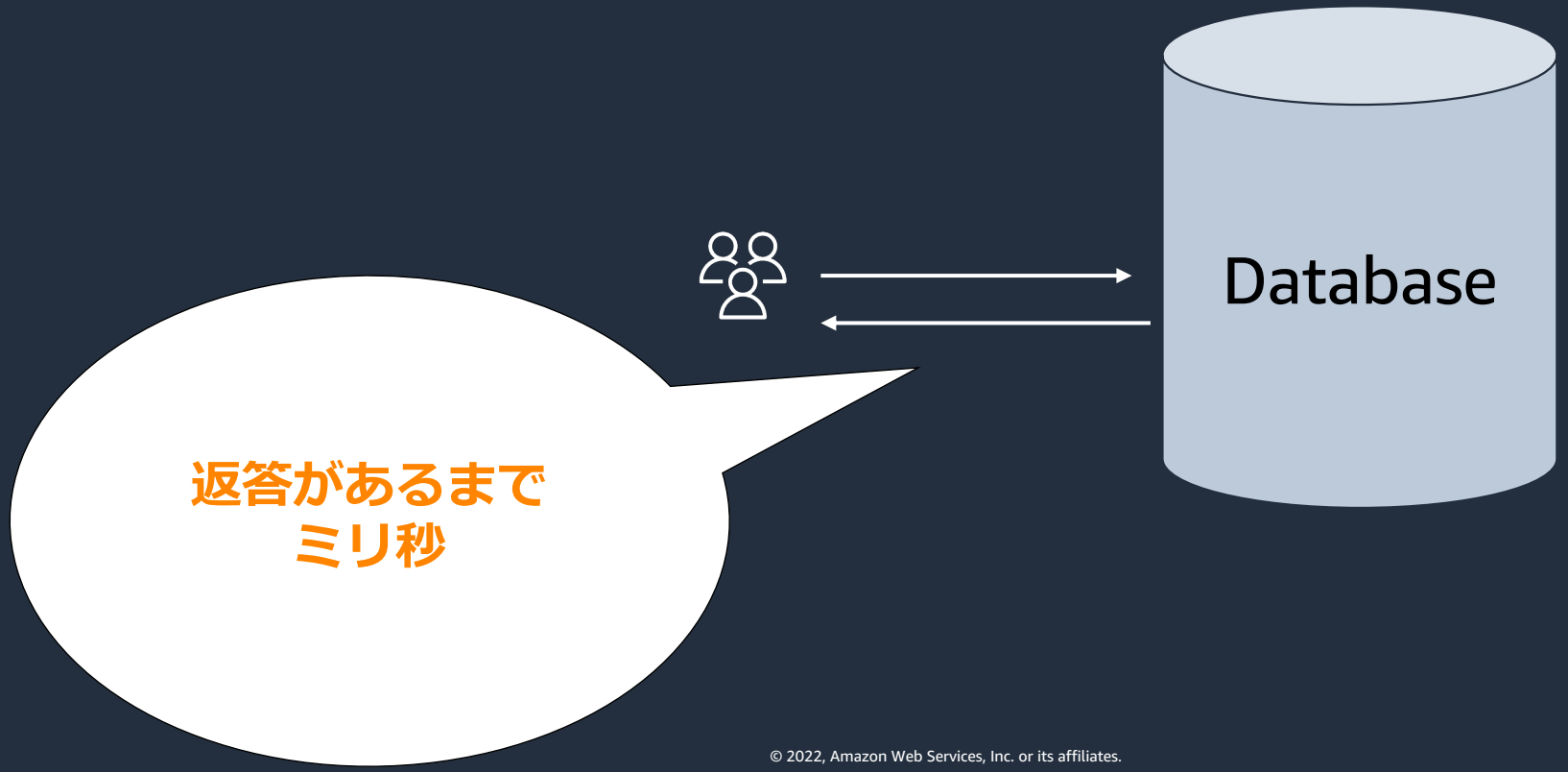
楽勝

Amazon DynamoDB

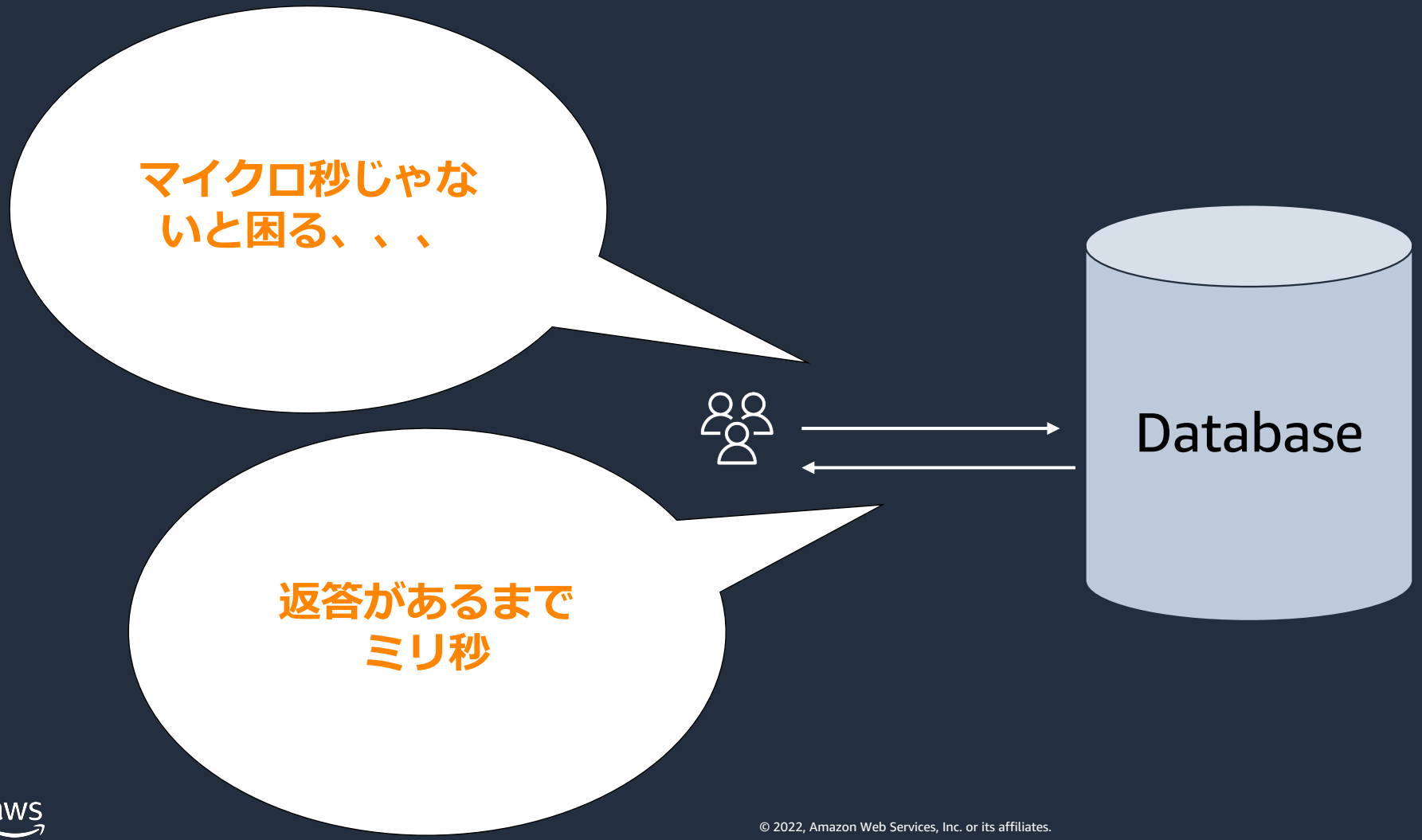
もっと低いレスポンスタイムがほしい！



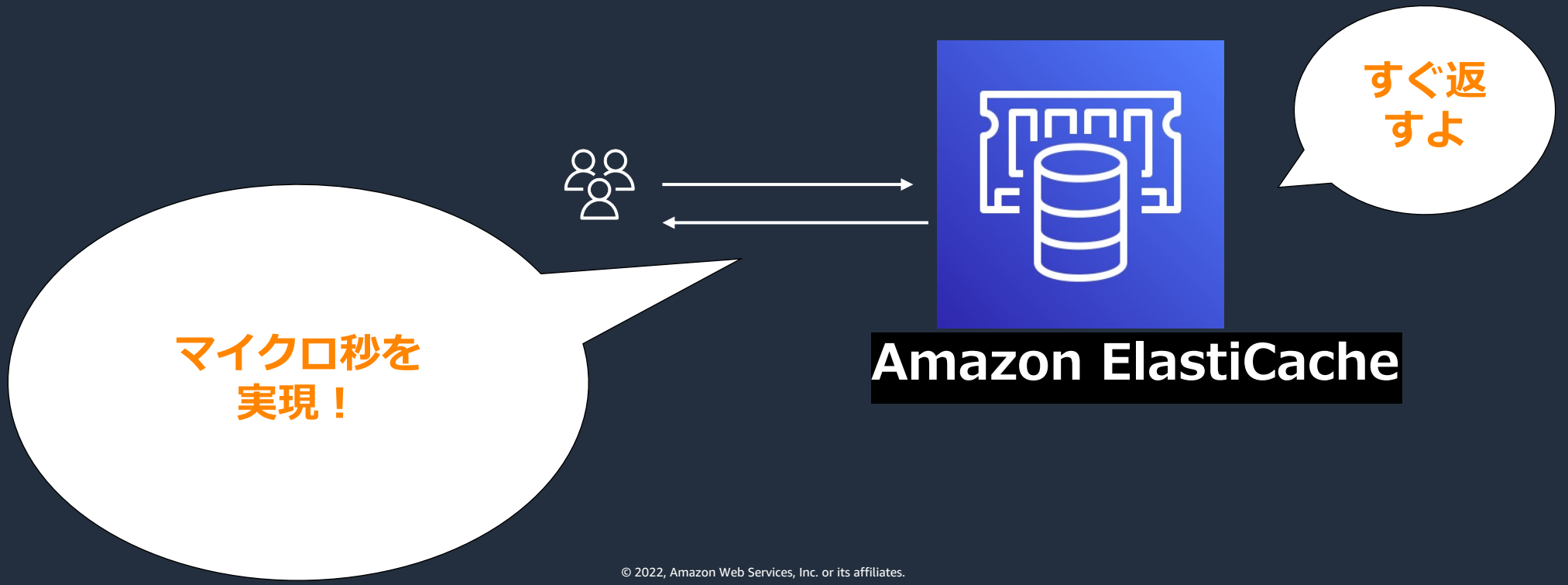
もっと低いレスポンスタイムがほしい！



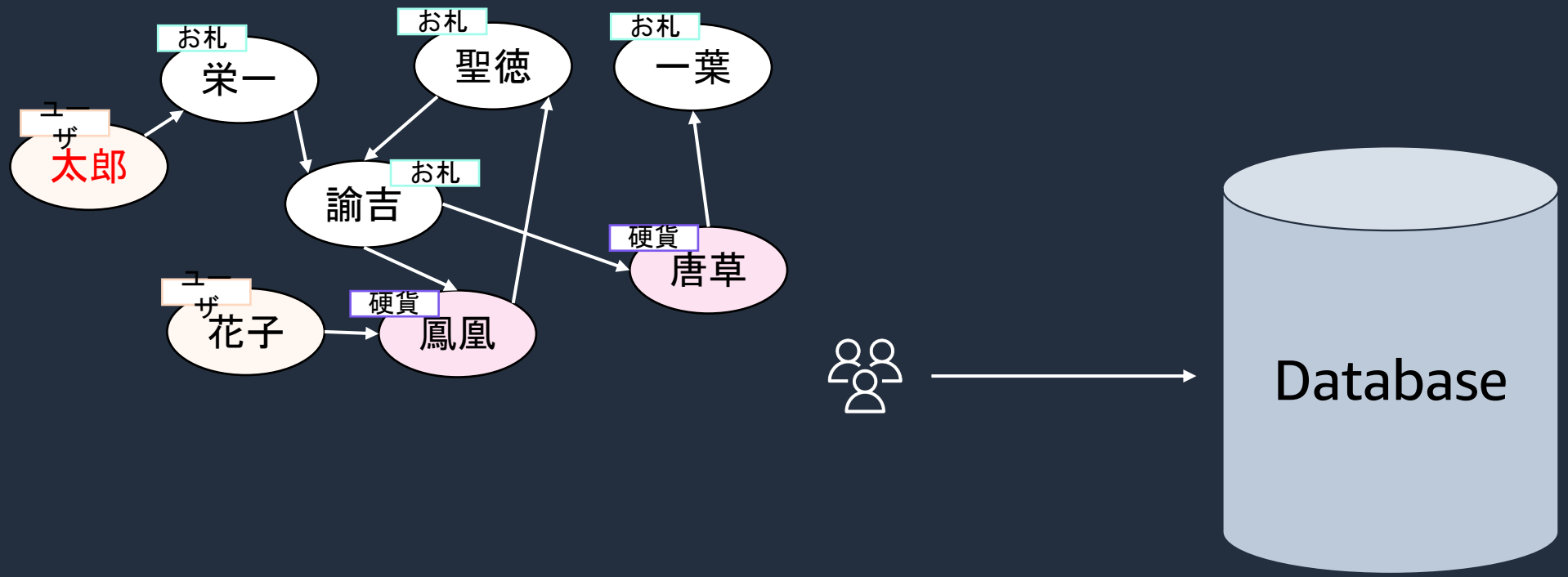
もっと低いレスポンスタイムがほしい！



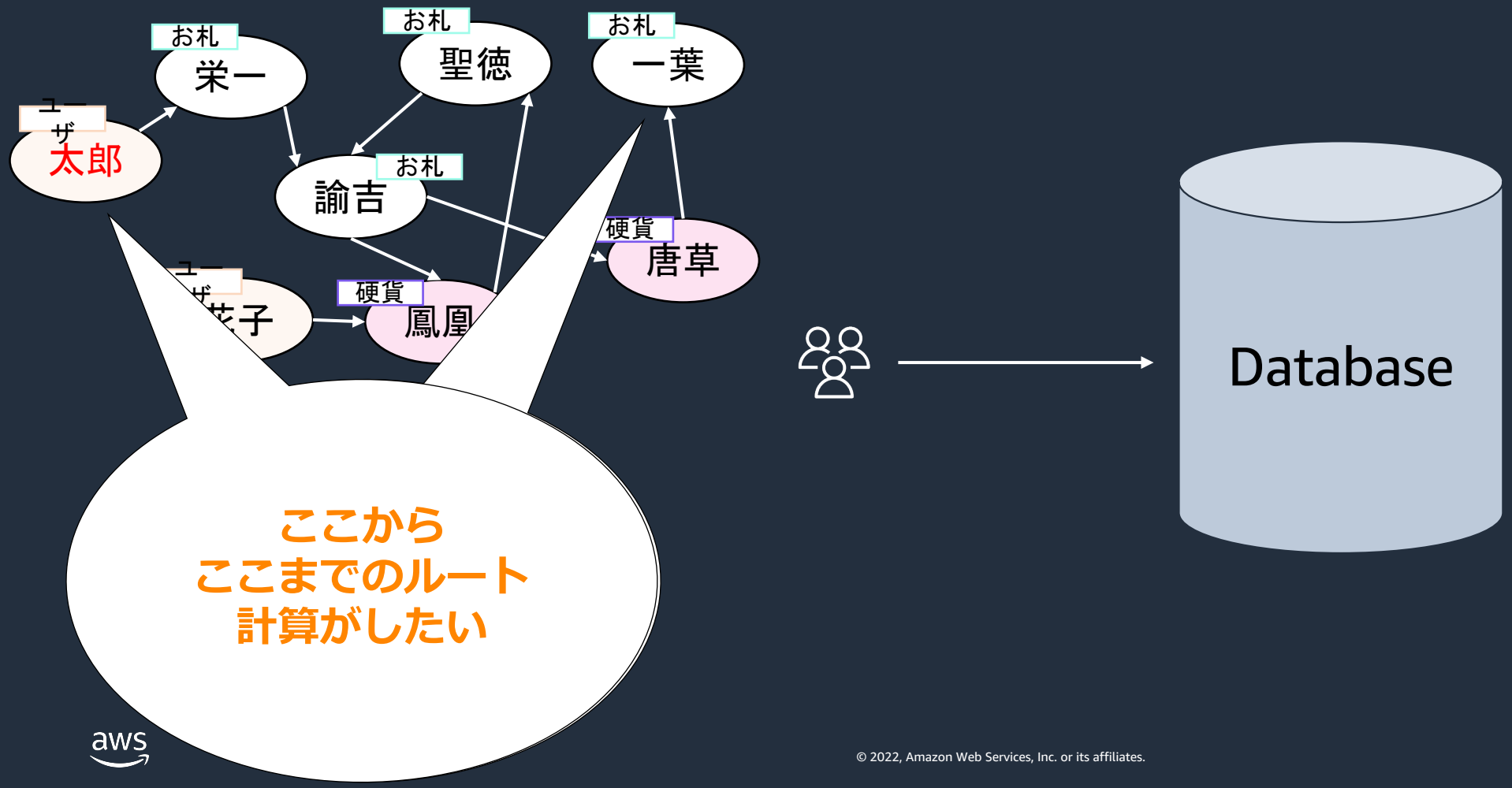
もっと低いレスポンスタイムがほしい！



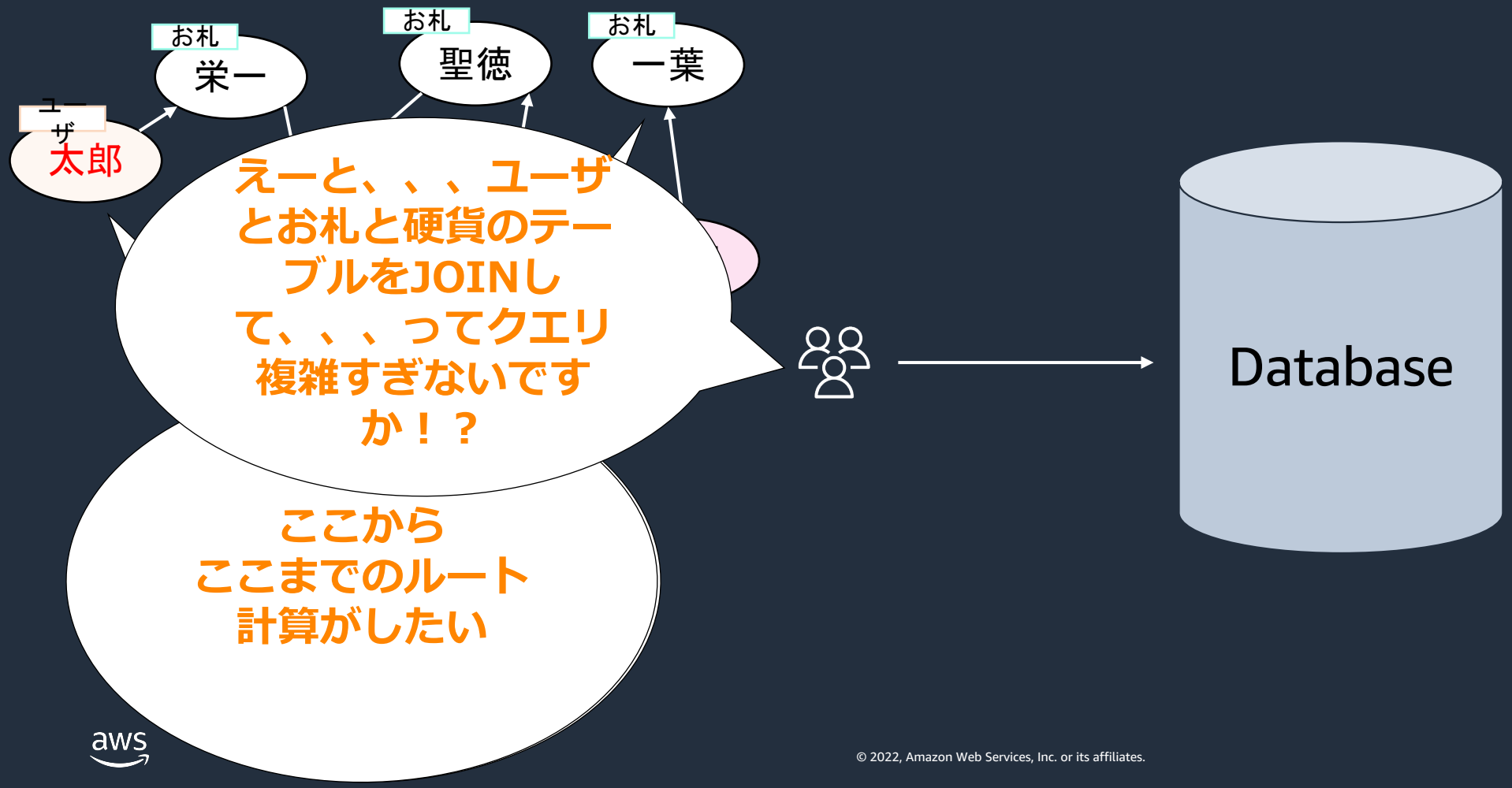
多ノードを経由するルートの計算がしたい！



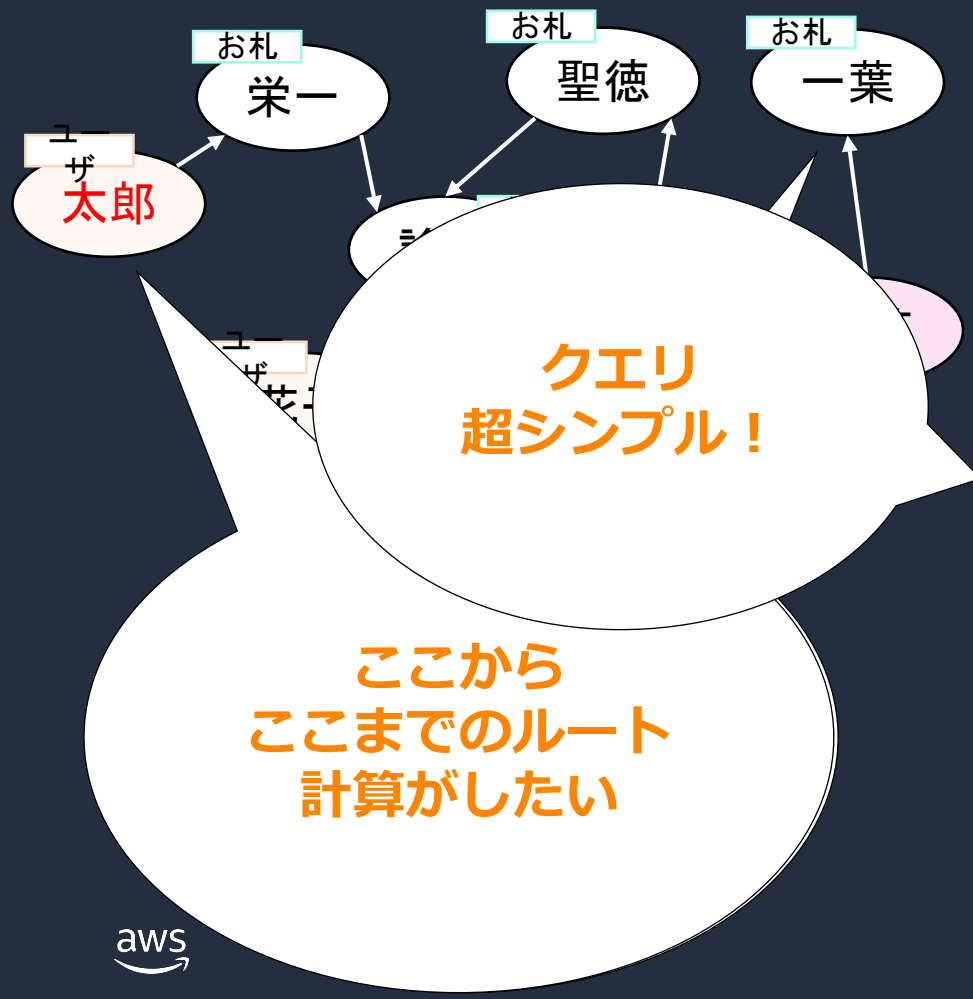
多ノードを経由するルートの計算がしたい！



多ノードを経由するルートの計算がしたい！



多ノードを経由するルートの計算がしたい！




```
Gremlin:
g.V().has("name","太郎")
.repeat(out().simplePath())
.until(has("name","一葉")).path()
```

Amazon Neptune

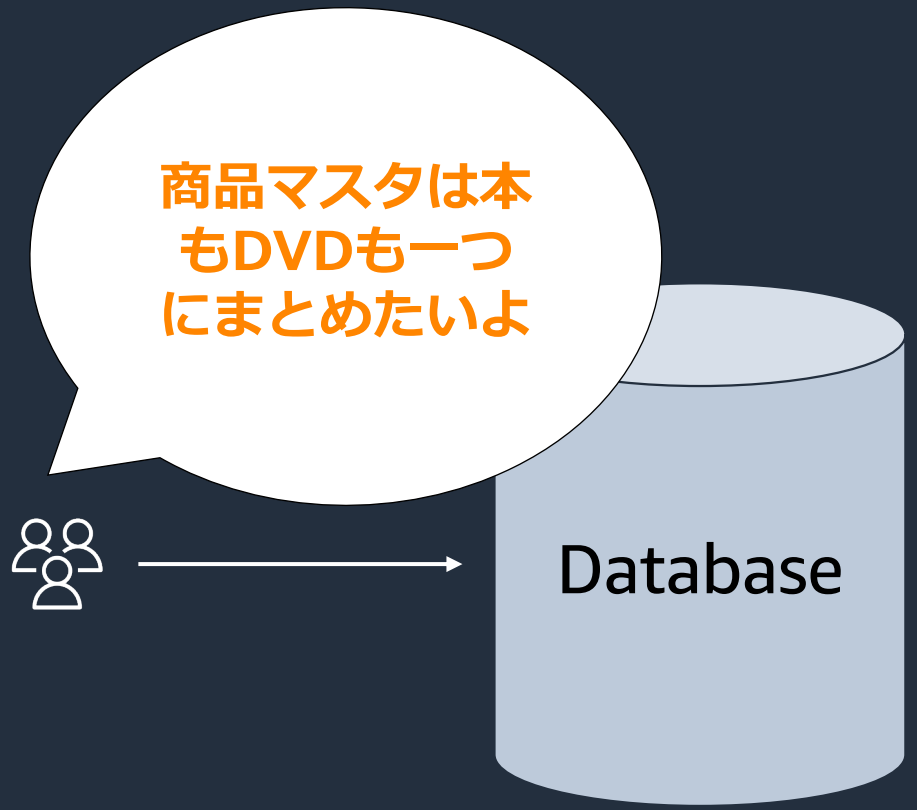
経路なら
任せて

少しづつ項目の違うデータを扱いたい/柔軟にデータを扱いたい！


```
{
   "_id" : ObjectId("5f3e3ba61a60d45b9a288581"),
   ” productType" : ”DVD”,
   "name" : ”The kuwano book the movie",
   ”releaseDate" : "2015-9-27",
   ”Director" : ”Akihiro Kuwano”,
   ”Actor" : [
      ”Akihiro Kuwano",
      “Narita Takashi"
   ]
}
```


共存したい

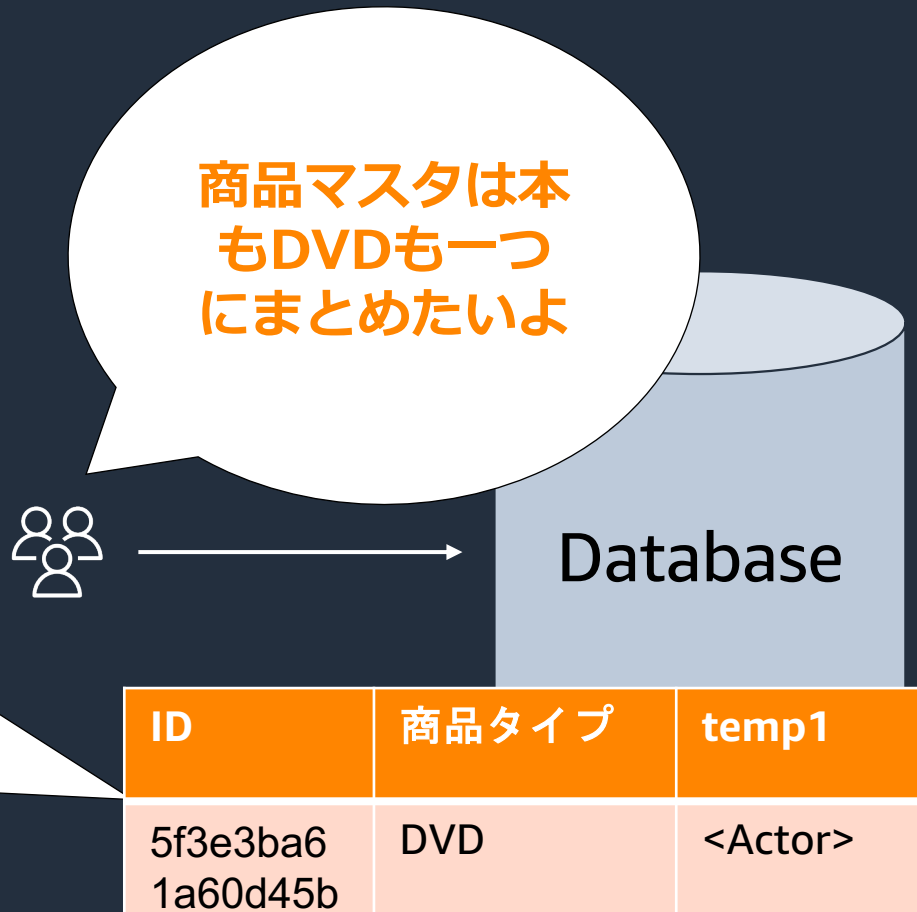

```
{
   "_id" : ObjectId("5f3e3ba61a60d45b9a288582"),
   “productType" : ”Book” ,
   "name" : ”The kuwano book",
   ”releaseDate" : "2015-9-27",
   ”Author" : ”Akihiro Kuwano”
}
```

少しづつ項目の違うデータを扱いたい/柔軟にデータを扱いたい！

```
{
   "_id" : ObjectId("5f3e3ba61a60d45b9a288581"),
   ” productType" : ”DVD”,
   "name" : ”The kuwano book the movie",
   ”releaseDate" : "2015-9-27",
   ”Director" : ”Akihiro Kuwano”,
   ”Actor" : [
      ”Akihiro Kuwano”
      “Narita Takash
   ]
}

{
   "_id" : ObjectId("5
   “productType" : ”Boo
   "name" : ”The kuwano book ”,
   ”releaseDate" : "2015-9-27",
   ”Author" : ”Akihiro Kuwano”
}
```

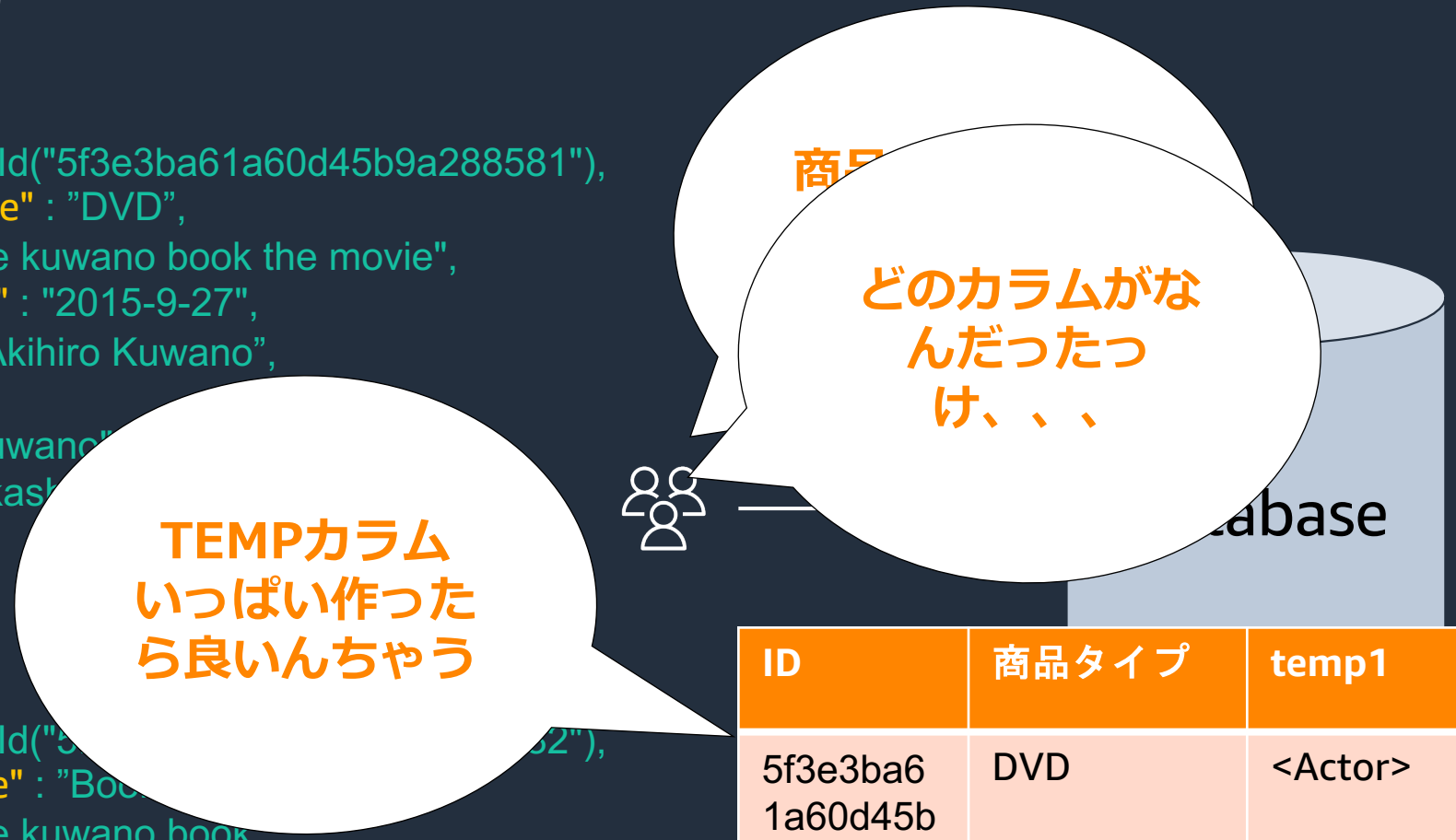


ID	商品タイプ	temp1	temp2	temp3
5f3e3ba6 1a60d45b 9a288581	DVD	<Actor>	<Author>	<未使用>

少しづつ項目の違うデータを扱いたい/柔軟にデータを扱いたい！

```
{
   "_id" : ObjectId("5f3e3ba61a60d45b9a288581"),
   ” productType" : ”DVD”,
   "name" : ”The kuwano book the movie",
   ”releaseDate" : "2015-9-27",
   ”Director" : ”Akihiro Kuwano”,
   ”Actor" : [
      ”Akihiro Kuwano”
      “Narita Takash
   ]
}

{
   "_id" : ObjectId("5
   “productType" : ”Boo
   "name" : ”The kuwano book ”,
   ”releaseDate" : "2015-9-27",
   ”Author" : ”Akihiro Kuwano”
}
```



ID	商品タイプ	temp1	temp2	temp3
5f3e3ba6 1a60d45b 9a288581	DVD	<Actor>	<Author>	<未使用>

少しづつ項目の違うデータを扱いたい/柔軟にデータを扱いたい！

```
{
   "_id" : ObjectId("5f3e3ba61a60d45b9a288581"),
   ” productType" : ”DVD”,
   "name" : ”The kuwano book the movie",
   ”releaseDate" : "2015-9-27",
   ”Director" : ”Akihiro Kuwano”,
   ”Actor" : [
      ”Akihiro Kuwano",
      “Narita Takashi"
   ]
}
```

共存したい

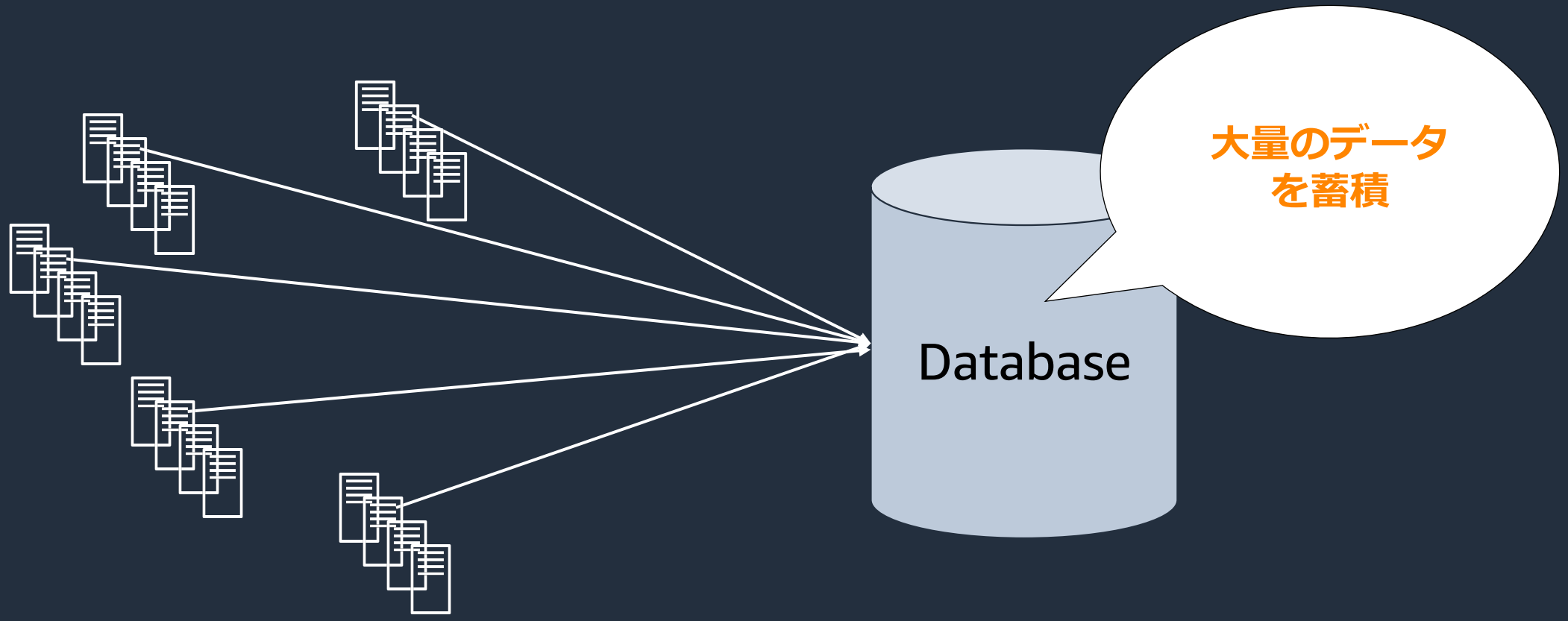
```
{
   "_id" : ObjectId("5f3e3ba61a60d45b9a288582"),
   “productType" : ”Book” ,
   "name" : ”The kuwano book",
   ”releaseDate" : "2015-9-27",
   ”Author" : ”Akihiro Kuwano”
}
```

商品マスタに本もDVDも一つにまとめられた！！！
シンプル！！！

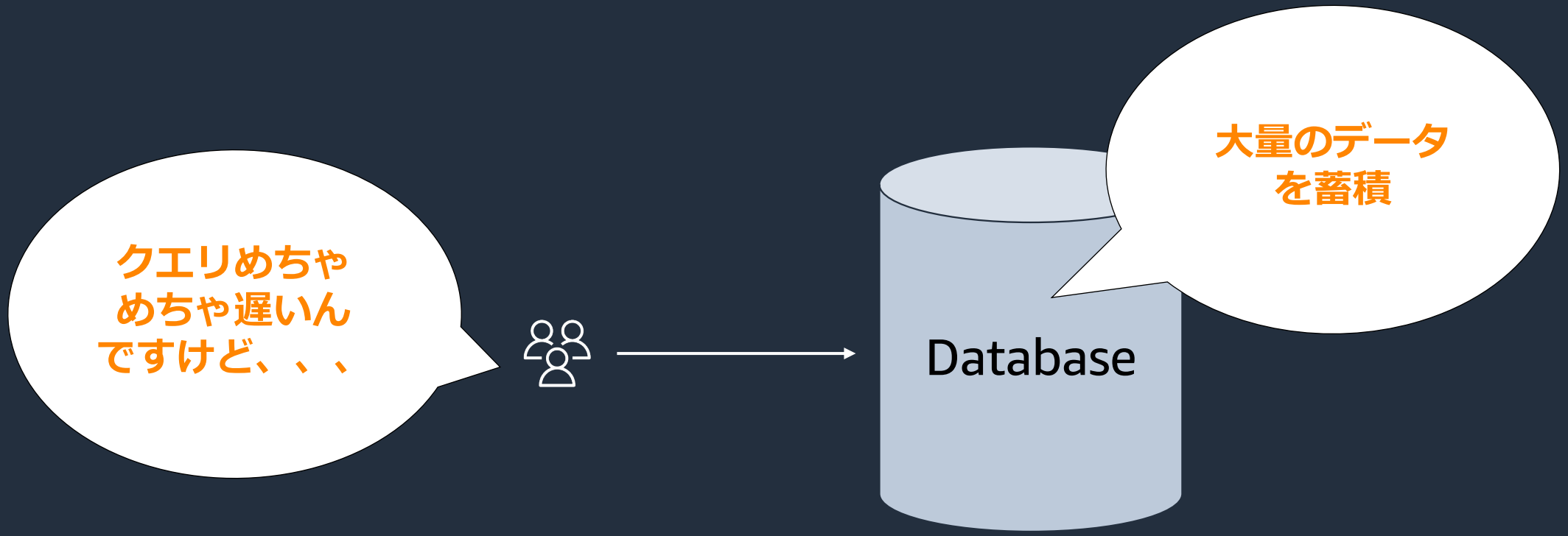
なんでも入るよ

Amazon DocumentDB

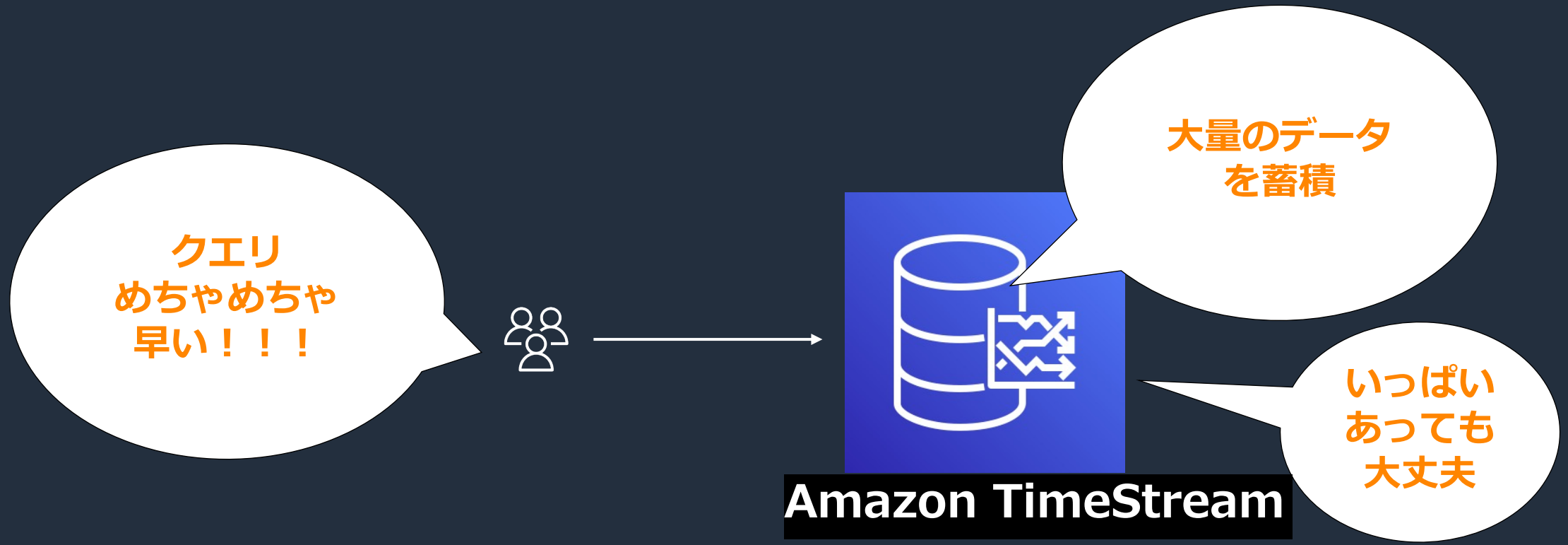
大量のノードからのCPU監視情報を扱いたい！



大量のノードからのCPU監視情報を扱いたい！

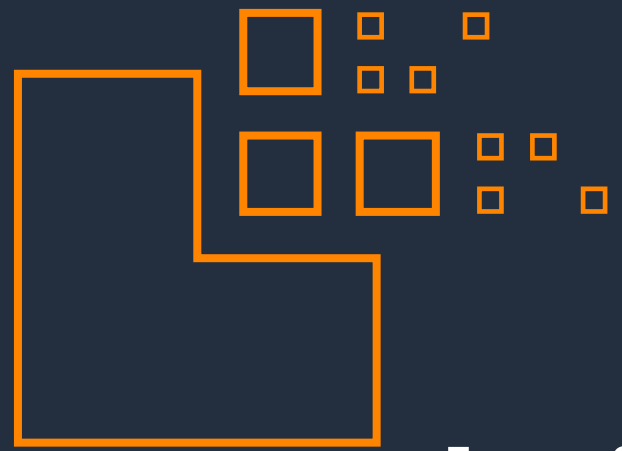


大量のノードからのCPU監視情報を扱いたい！



このように、、、

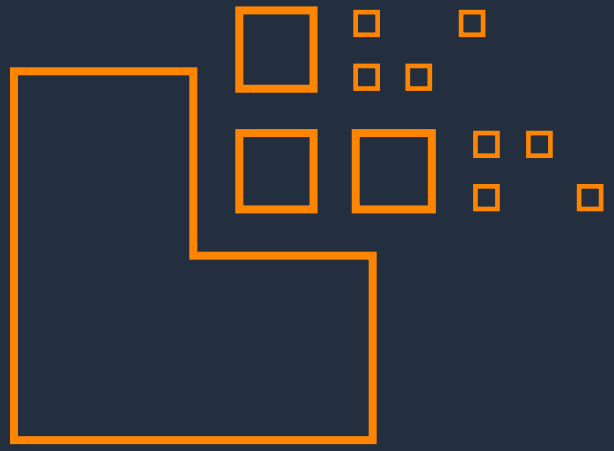
- 各データベースの特徴を活かして選択することでより高いパフォーマンス、より低いコスト、より低いTCOを実現することが可能になる
- これがAWSがPurpose-built Databaseといっている内容になります



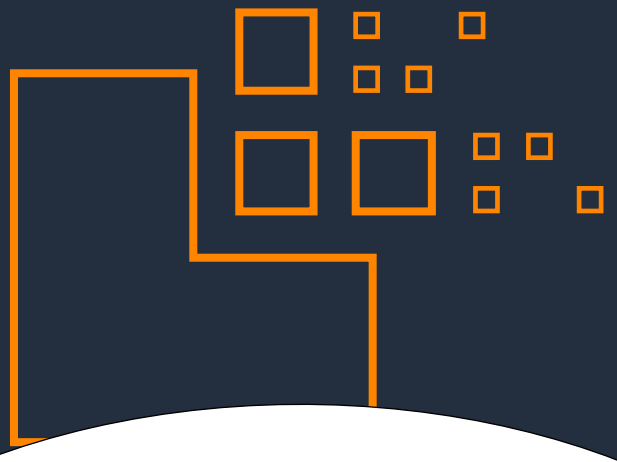
Purpose built

The right tool for the right job

どうやって選べばいいの？



A one size fits all database doesn't fit anyone!
(一つのデータベースでは全てにフィットしない！)



A one size fits all database doesn't fit anyone!
(一つのデータベースでは全てにフィットしない！)

**ワーナーさん、
実際にはどんな感じに選んだら
良いの？**

適切なデータベースを選択する方法

- まず最初に作るアプリケーションにどのような要件が存在するのかをブレイクダウンする
- 項目の例
  - アクセスパターン
  - データ量
  - スケールパターン
  - ユースケース
    - グラフDBや、台帳DBの様なユースケースが明確なものは決めやすい
  - 機能要件
  - エンジニアのスキルセット
  - などなど

あるアプリケーションの要件その1

Social media

ユーザー: 1,000 以上
データ量: GB
場所: リージョナル
パフォーマンス: ミリ秒
リクエスト: 数千/sec
アクセス: ウェブ
拡張: スケールアップ/ダウン
開発者アクセス: ドライバアクセス

あるアプリケーションの要件その2

Ride hailing　Media streaming　Social media　Dating

ユーザー: 1,000,000 以上
データ量: PB
場所: グローバル
パフォーマンス: ミリ秒
リクエスト: 数百万/sec
アクセス: ウェブ/モバイル
拡張: スケールアウト/イン
開発者アクセス: APIアクセス

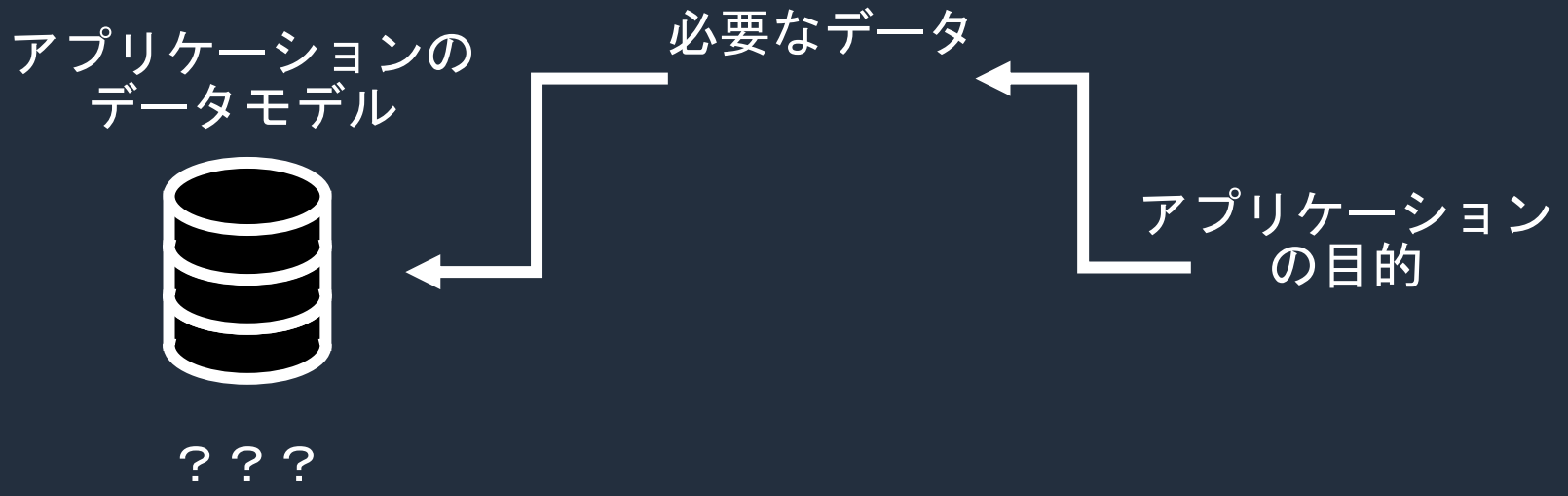
そしてもう一つ大事なことは、、、

Working backwards – クエリを想定したデザイン

Working backwards – クエリを想定したデザイン

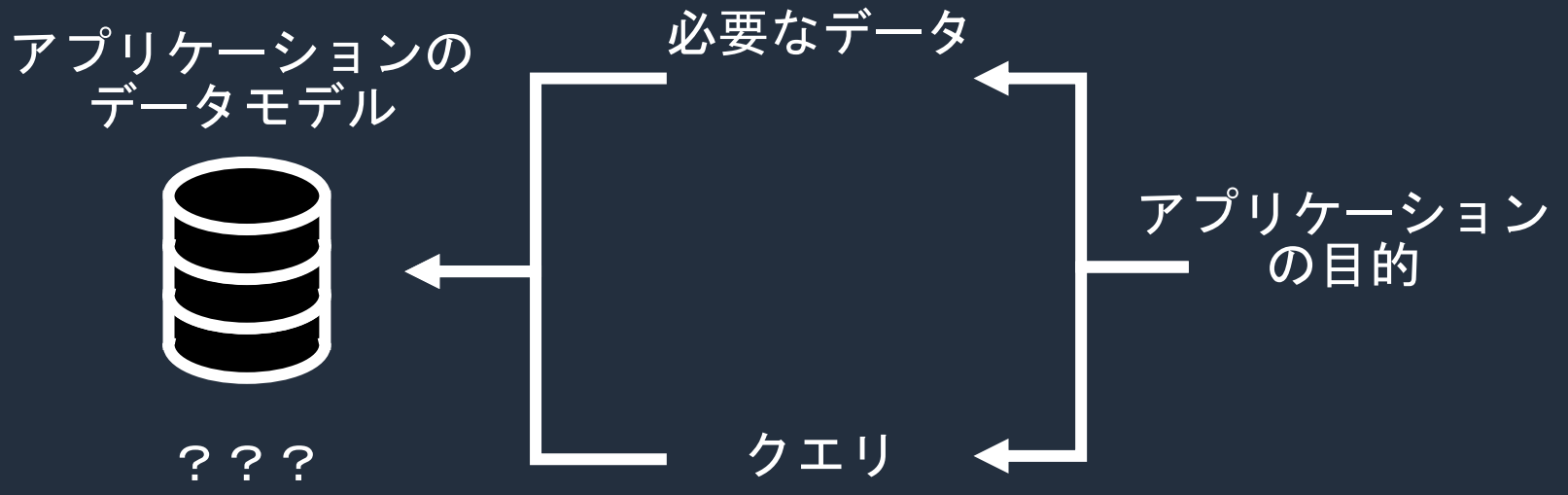
アプリケーション
の目的

Working backwards – クエリを想定したデザイン



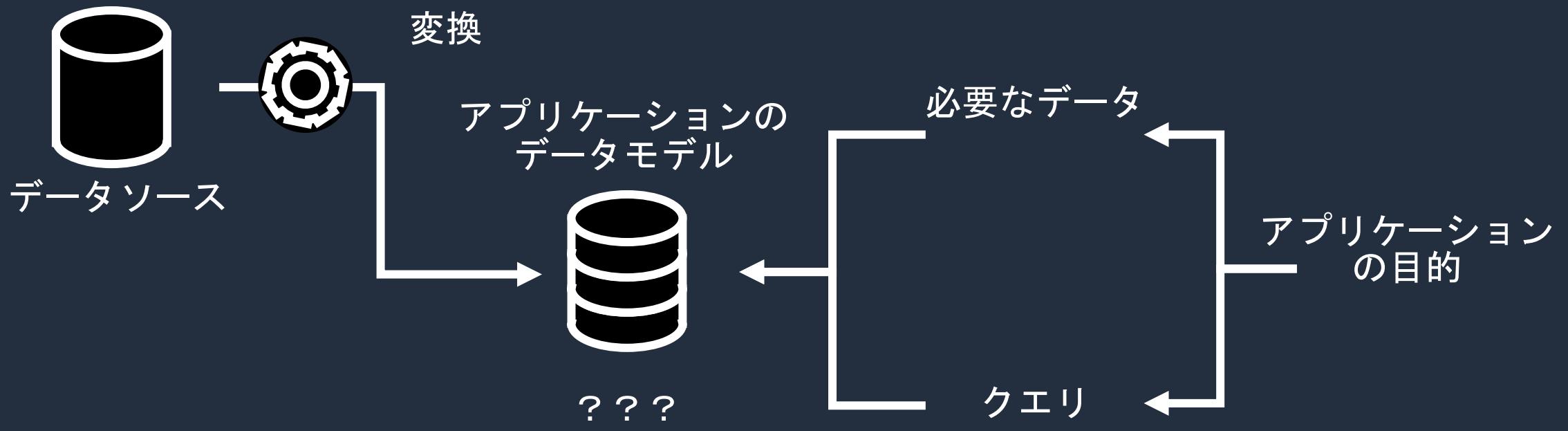
目的を達成するためにはどのようなデータが必要なのか？

Working backwards – クエリを想定したデザイン



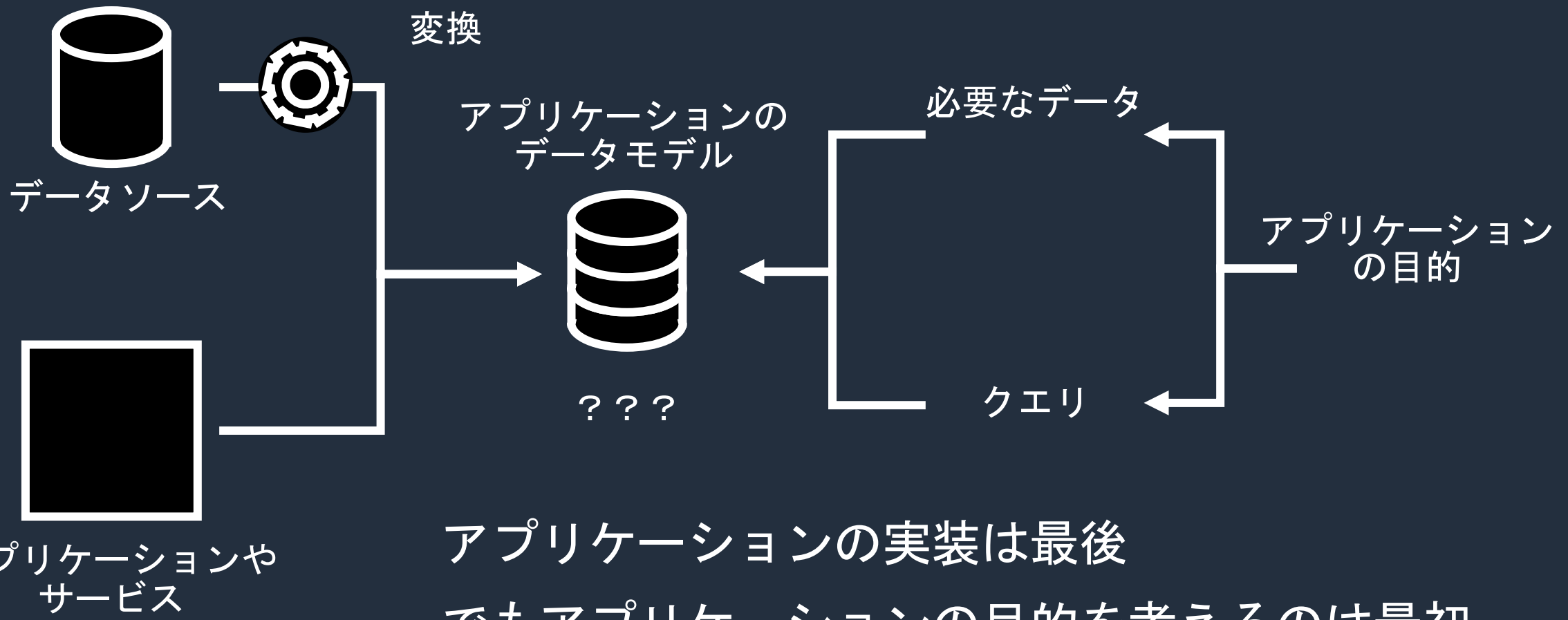
そしてどのようなクエリが必要となるのか？

Working backwards – クエリを想定したデザイン



最後にデータソースからの変換を考える

Working backwards – クエリを想定したデザイン



アプリケーションの実装は最後

でもアプリケーションの目的を考えるのは最初

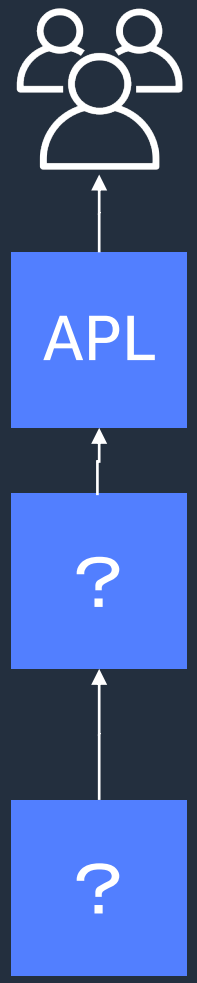
アプリケーションによるデータベースの選定例

アプリケーションとデータベース

- では実際にアプリケーションの要件を決めつつ選定を行ってみましょう

コンテンツ管理システム

- ニュースサイトなど、文書などのコンテンツ管理を行うシステム
- 不定形なデータ構造
- コメントなどをつけることができる
- タグで分類
- 著者名も管理



Amazon Web Services ブログ

新機能 – Redis 6 互換の Amazon ElastiCache

by Channy Yun | on 13 OCT 2020 | in Open Source | Permalink | Share

最新の Redis 5.0 互換の Amazon ElastiCache 以降、 5.0.6 でのアップストリームサポートを含め、Am
ElastiCache for Redis では多くの改善が行われました。

今年の初め、1 つのリージョンのクラスターを最大 2 つの他のリージョンのクラスターにレプリケートでき
Datastore for Redis 機能を発表しました。最近では、18 個のエンジンおよびノードレベルの CloudWa
を追加することで、Redis フリートをモニタリングする機能が向上しました。リソースレベルのアクセス許
めのサポートも追加され、特定の ElastiCache リソースに AWS Identity and Access Management (IA
ルアクセス許可を割り当てることができるようになりました。

そして本日、**Redis 6** 互換の Amazon ElastiCache for Redis を発表することになりました。今回のリリ
Amazon ElastiCache for Redis にいくつかの新機能や重要な機能が追加されています。

- **マネージドロールベースのアクセス制御** – Amazon ElastiCache for Redis 6 では、Redis コマン
ースのアクセス制御 (RBAC) の設定に使用できるユーザーとユーザーグループを作成および管理でき
した。複数のアプリケーションで、互いのデータにアクセスすることなく、同じ Redis クラスター

アプリケーション要件の整理

- 条件を列挙し、まずは右のようなアクセス要件を考える

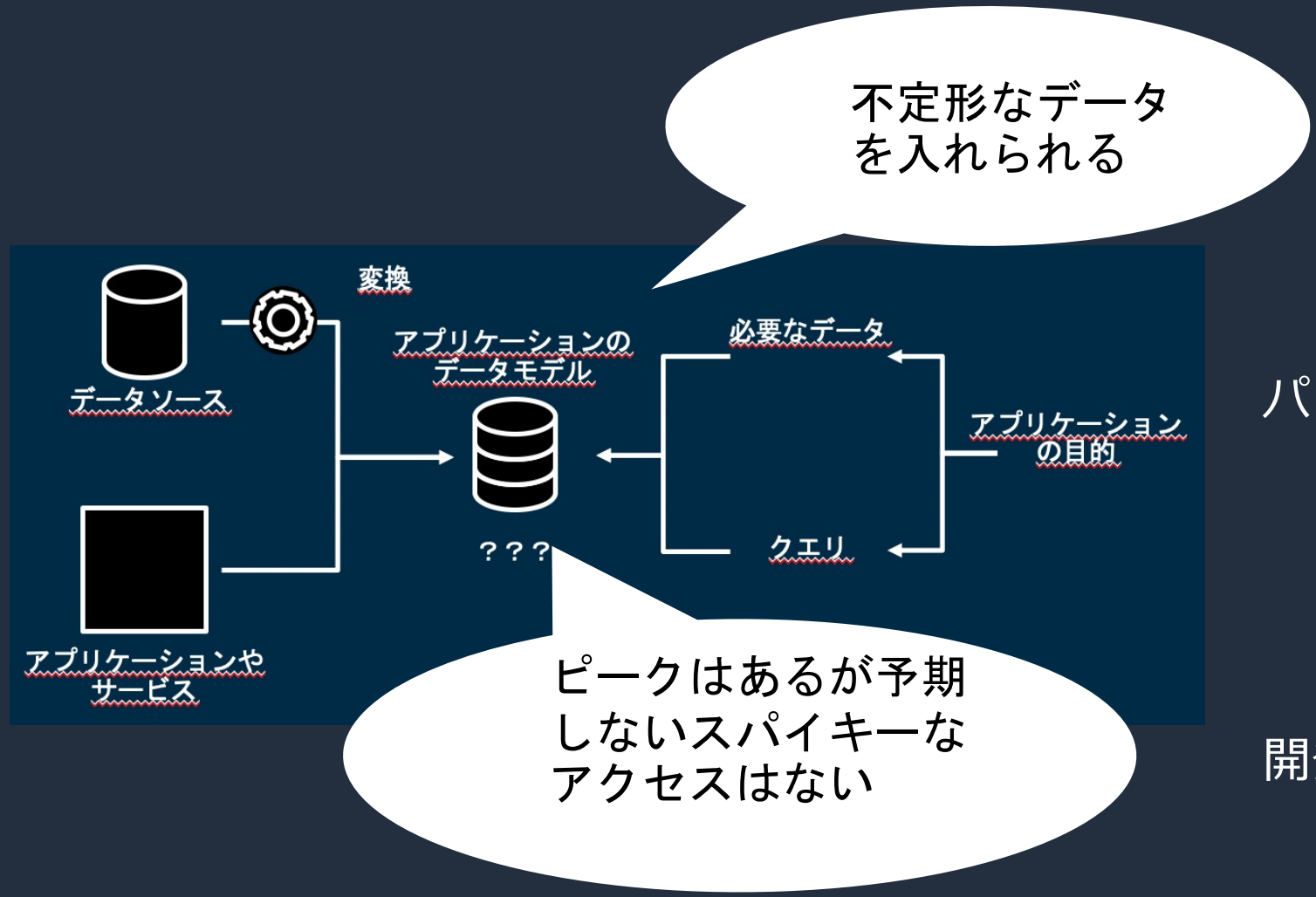
ユーザー: 1,000,000 以上
データ量: TB
場所: リージョナル
パフォーマンス: ミリ秒
リクエスト: 数万/sec
アクセス: ウェブ/モバイル
拡張: スケールアップ/ダウン、スケールアウト/イン
開発者アクセス: APIアクセス

アプリケーション要件の整理

- 条件を列挙し、まずは右のようなアクセス要件を考える
- 更に以下の様な要件も考えてみる
  - 不定形なデータを入れられる
  - ピークはあるが予期しないスパイキーなアクセスはない

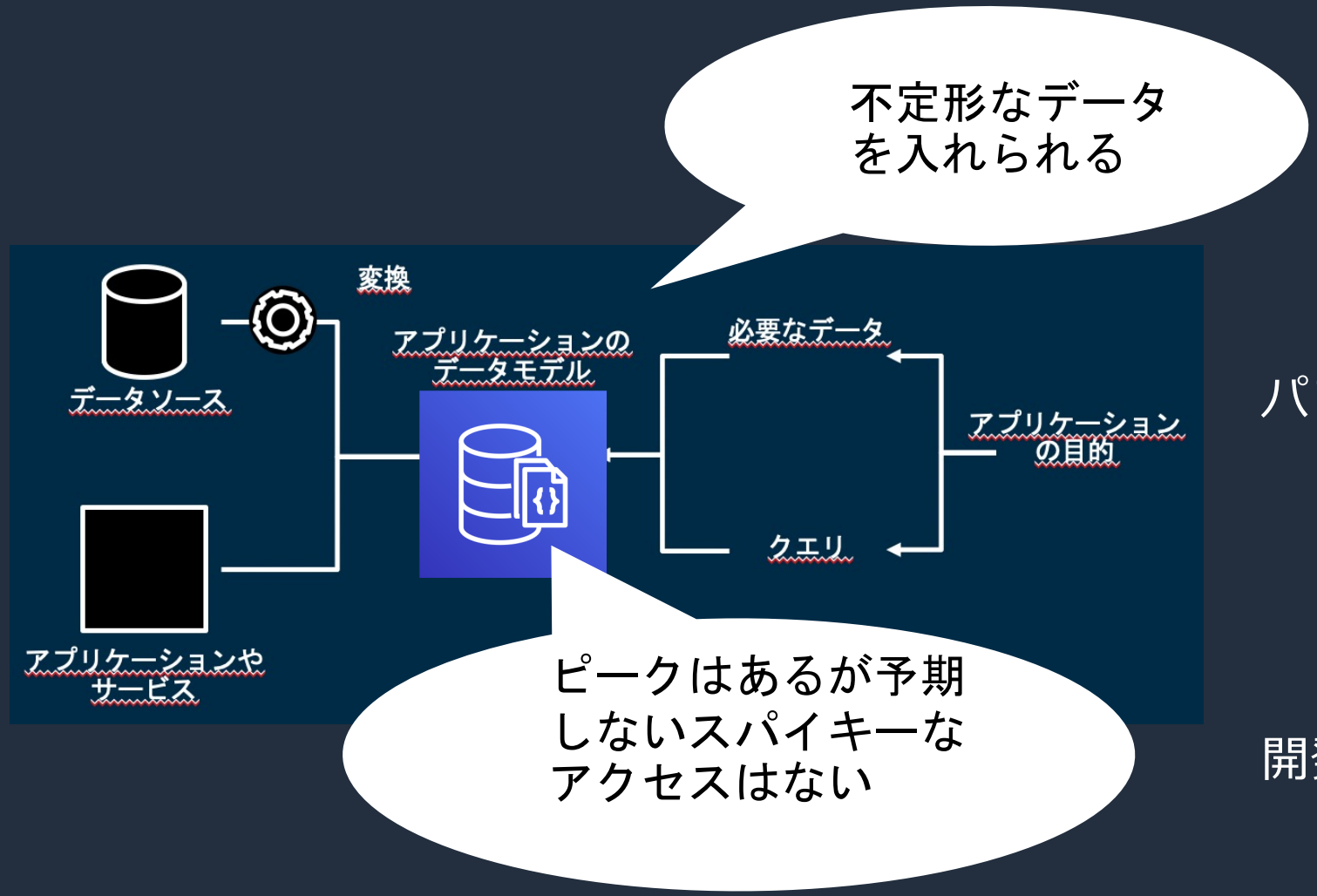
ユーザー:	1,000,000 以上
データ量:	TB
場所:	リージョナル
パフォーマンス:	ミリ秒
リクエスト:	数万/sec
アクセス:	ウェブ/モバイル
拡張:	スケールアップ/ダウン、スケールアウト/イン
開発者アクセス:	APIアクセス

Working backwards してみる



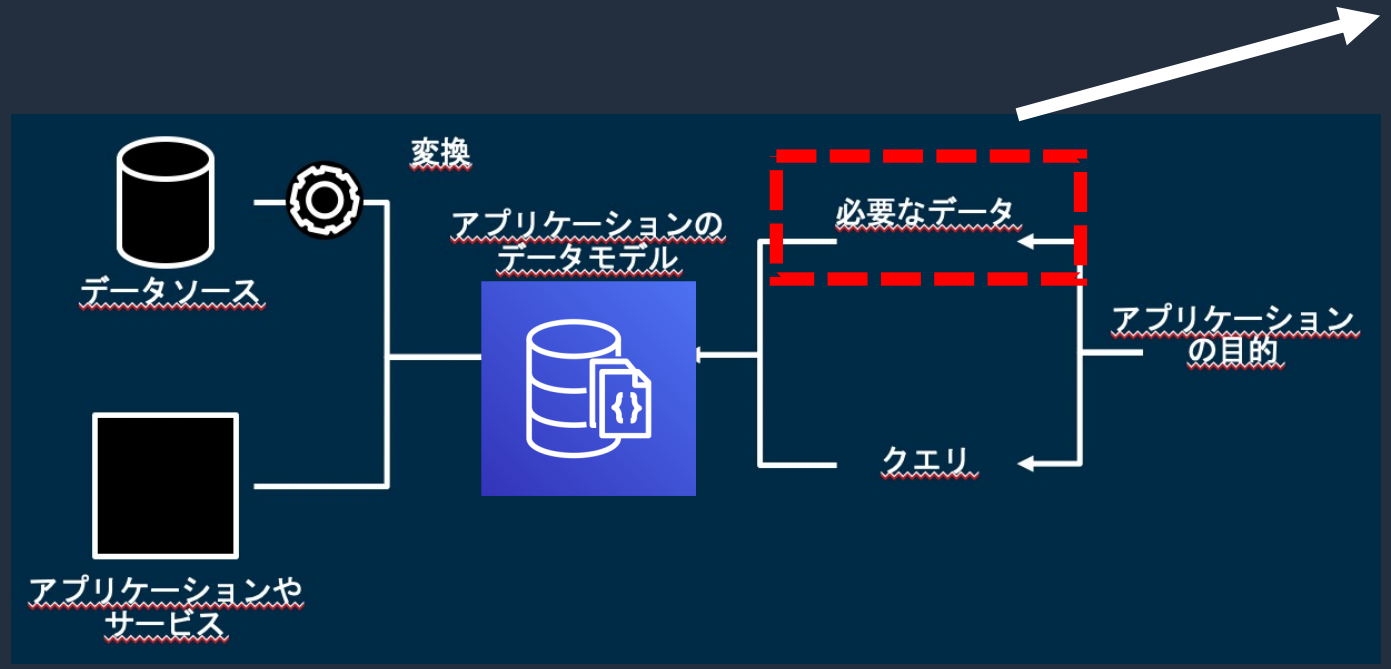
ユーザー: 1,000,000 以上
データ量: TB
場所: リージョナル
パフォーマンス: ミリ秒
リクエスト: 数万/sec
アクセス: ウェブ/モバイル
拡張: スケールアップ/ダウン、スケールアウト/イン
開発者アクセス: APIアクセス

Working backwards してみる



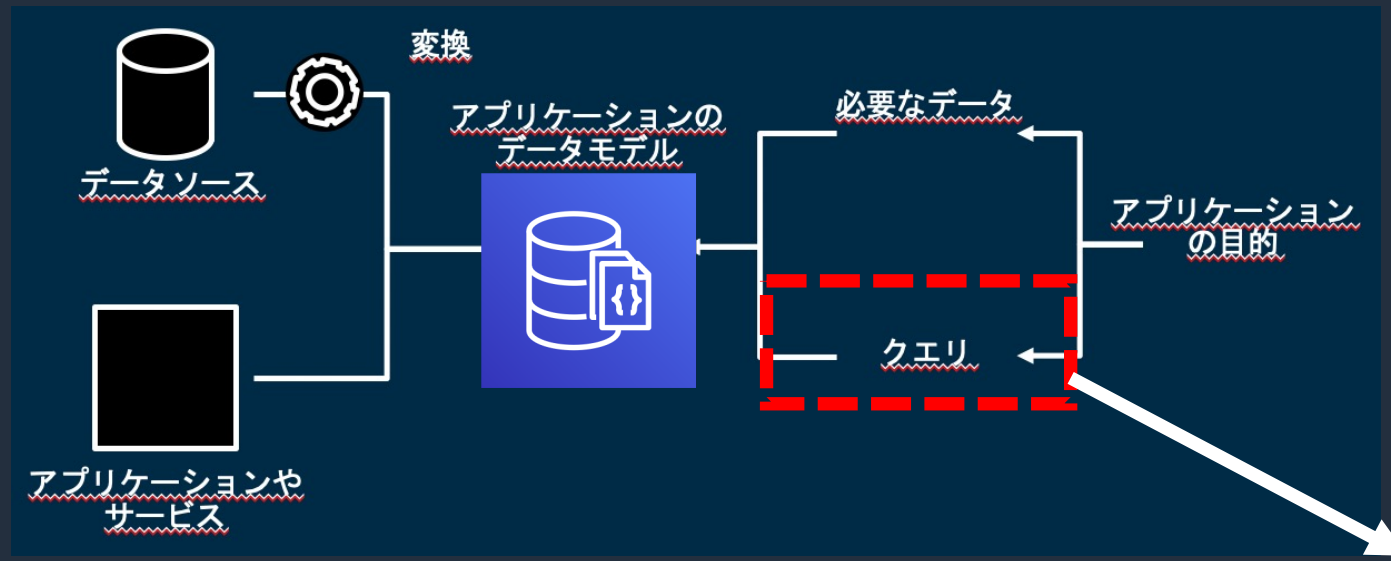
ユーザー: 1,000,000 以上
データ量: TB
場所: リージョナル
パフォーマンス: ミリ秒
リクエスト: 数万/sec
アクセス: ウェブ/モバイル
拡張: スケールアップ/ダウン、スケールアウト/イン
開発者アクセス: APIアクセス

Working backwards してみる



項目	内容
記事ID	abcde
作成日時	2020/10/21 00:00:00
執筆者	Akihiro Kuwano
タグ	blog
本文	＜テキスト＞
（コメント）	＜テキスト＞

Working backwards してみる

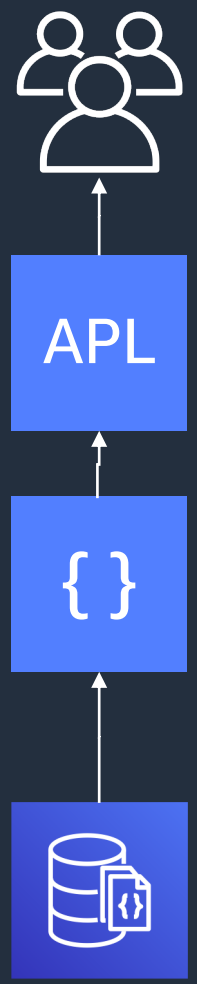


項目	内容
記事ID	abcde
作成日時	2020/10/21 00:00:00
執筆者	Akihiro Kuwano
タグ	blog
本文	<テキスト>
（コメント）	<テキスト>

クエリ
記事IDにマッチした記事を取得する
作成日時でソートしてXX件表時
執筆者で検索
タグで検索
記事IDについてるコメント一覧の取得

結果：コンテンツ管理システム

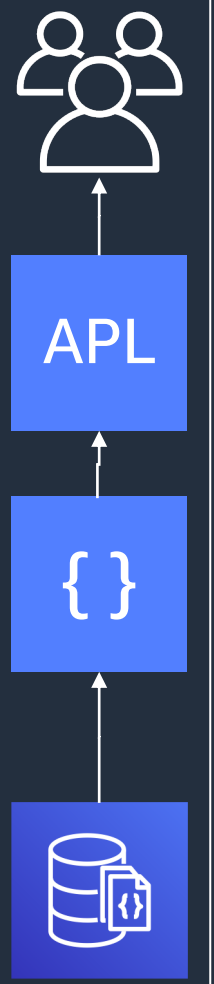
- ニュースサイトなど、文書などのコンテンツ管理を行うシステム
- 不定形なデータ構造でも対応可能
- 文書ごとの関係性や、タグ付け、コメントなども表現しやすい
- 文書の簡易検索は $regex なども利用可能
- スケールも可能
- DocumentDBの開発効率の良さが活きる




```
{
  _id: abcde,
  title: "My Blog",
  created: ISODate("2020-08-26…"),
  author: { _id: “kuwanoa”, name: ‘Akihiro Kuwano' },
  tags: [ “blog”, … ],
  detail: { text: ”テキスト…”}
}
```

結果：コンテンツ管理システム

- ニュースサイトなど、文書などのコンテンツ管理を行うシステム
- 不定形なデータ構造でも対応可能
- 文書ごとの関係性や、タグ付け、コメントなども表現しやすい
- 文書の簡易検索は $regex なども利用可能
- スケールも可能
- DocumentDBの開発効率の良さが活きる




```
{
  _id: abcde,
  title: "My Blog",
  created: ISODate("2020-08-26…"),
  author: { _id: “kuwanoa”, name: ‘Akihiro Kuwano' },
  tags: [ “blog”, … ],
  detail: { text: ”テキスト…" },
  comment: [
   {
     author: {name: ”Takashi” },
     created: ISODate("2020-08-26…"),
     detail: { text: ”コメント…" }
   },
   {
     author: {name: “Zabbio” },
     created: ISODate("2020-08-26…"),
     detail: { text: ”コメント…" }
   },
  ]
}
```

ちょっとまった！

ここで条件がもし変わったらどうなるでしょうか

- 同じアプリケーションだけどちょっとだけ条件を変えてみましょう

アプリケーション要件の整理：再び

- アクセス要件は一箇所を除いて同じです

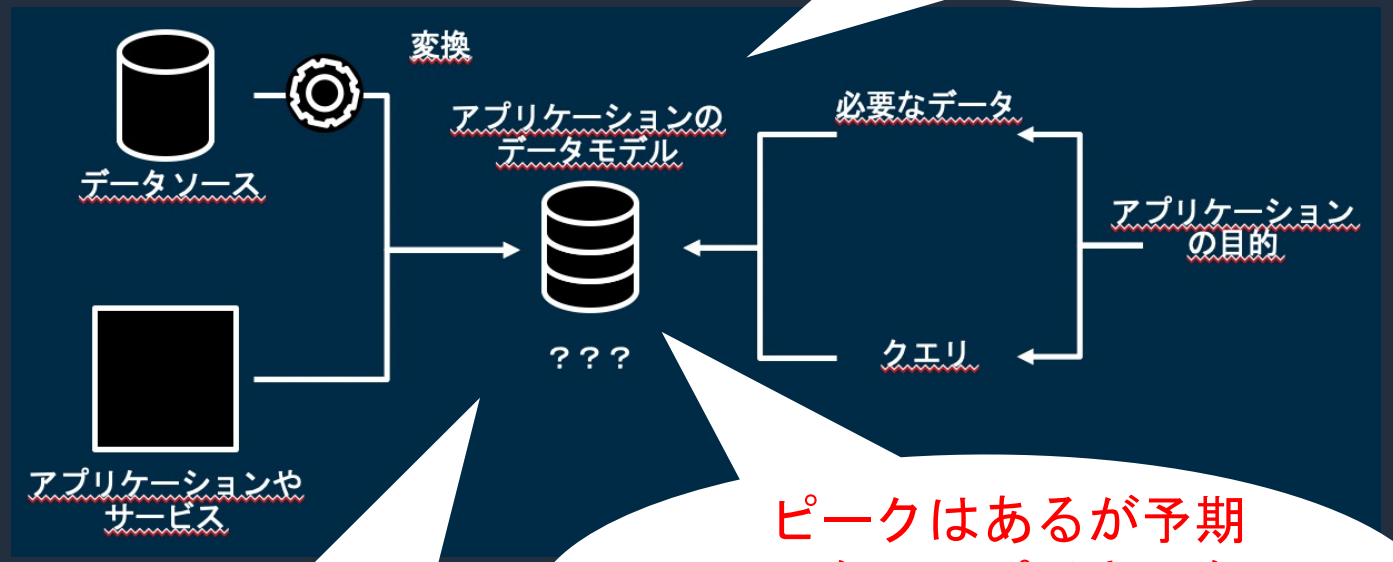
ユーザー:	1,000,000 以上
データ量:	TB
場所:	リージョナル
パフォーマンス:	ミリ秒
リクエスト:	**数百万/sec**
アクセス:	ウェブ/モバイル
拡張:	スケールアップ/ダウン、スケールアウト/イン
開発者アクセス:	APIアクセス

アプリケーション要件の整理：再び

- アクセス要件は一箇所を除いて同じです
- 更に以下の様な要件も考えてみる
  - 不定形なデータを入れられる
  - ピークはあるが予期しないスパイキーなアクセスがある
  - 結果リクエスト数も多い

ユーザー: 1,000,000 以上
データ量: TB
場所: リージョナル
パフォーマンス: ミリ秒
リクエスト: 数百万/sec
アクセス: ウェブ/モバイル
拡張: スケールアップ/ダウン、スケールアウト/イン
開発者アクセス: APIアクセス

Working backwards してみる：再び



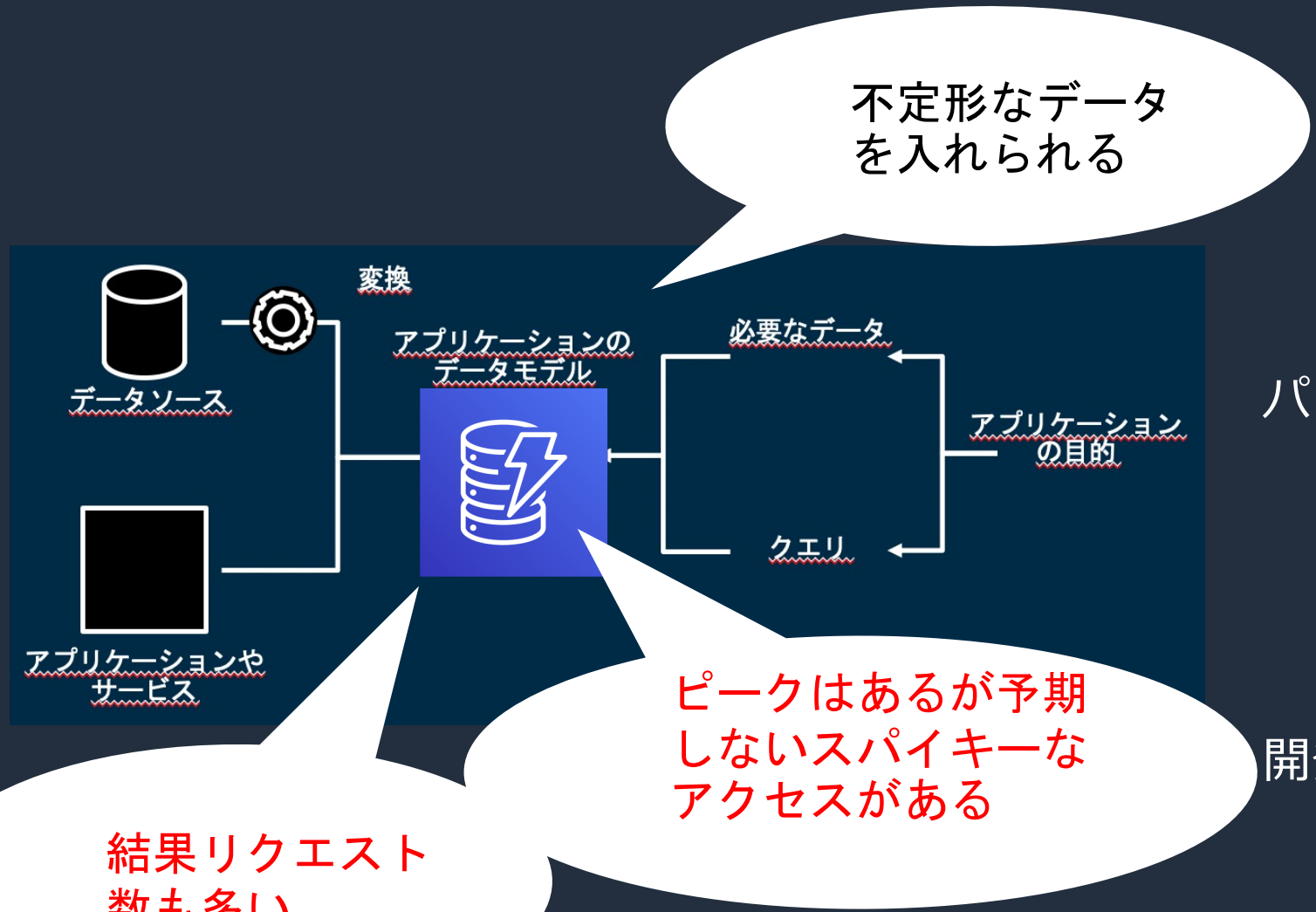
不定形なデータを入れられる

ピークはあるが予期しないスパイキーなアクセスがある

結果リクエスト数も多い

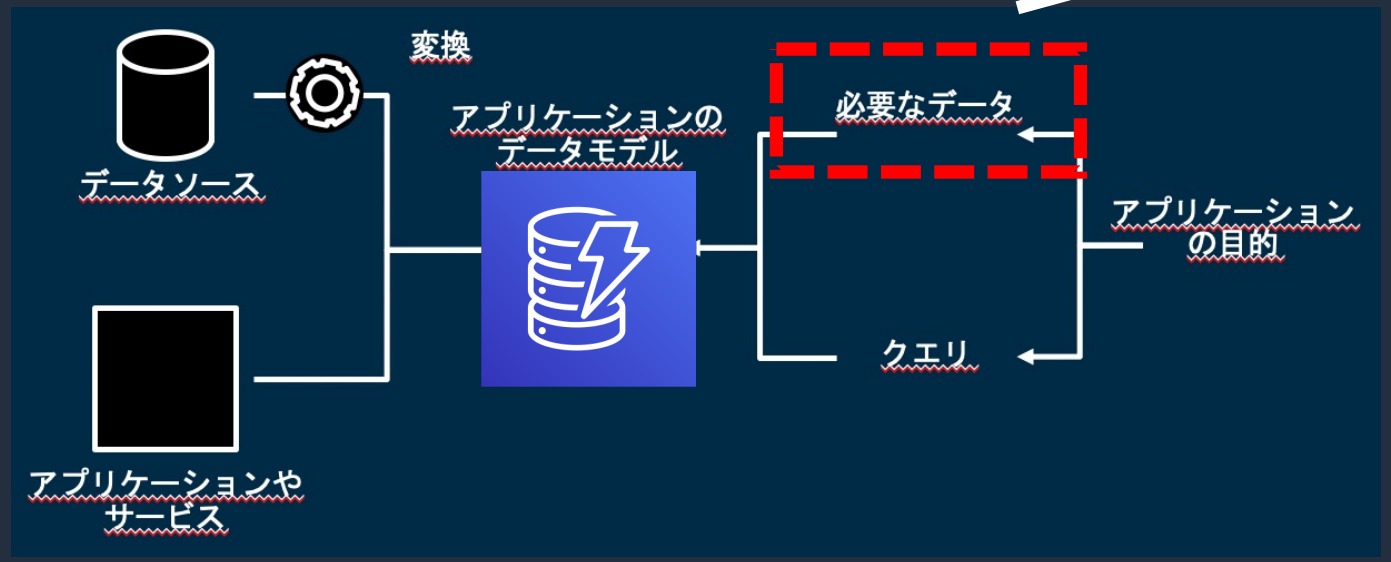
ユーザー: 1,000,000 以上
データ量: TB
場所: リージョナル
パフォーマンス: ミリ秒
リクエスト: 数万/sec
アクセス: ウェブ/モバイル
拡張: スケールアップ/ダウン、スケールアウト/イン
開発者アクセス: APIアクセス

Working backwards してみる：再び



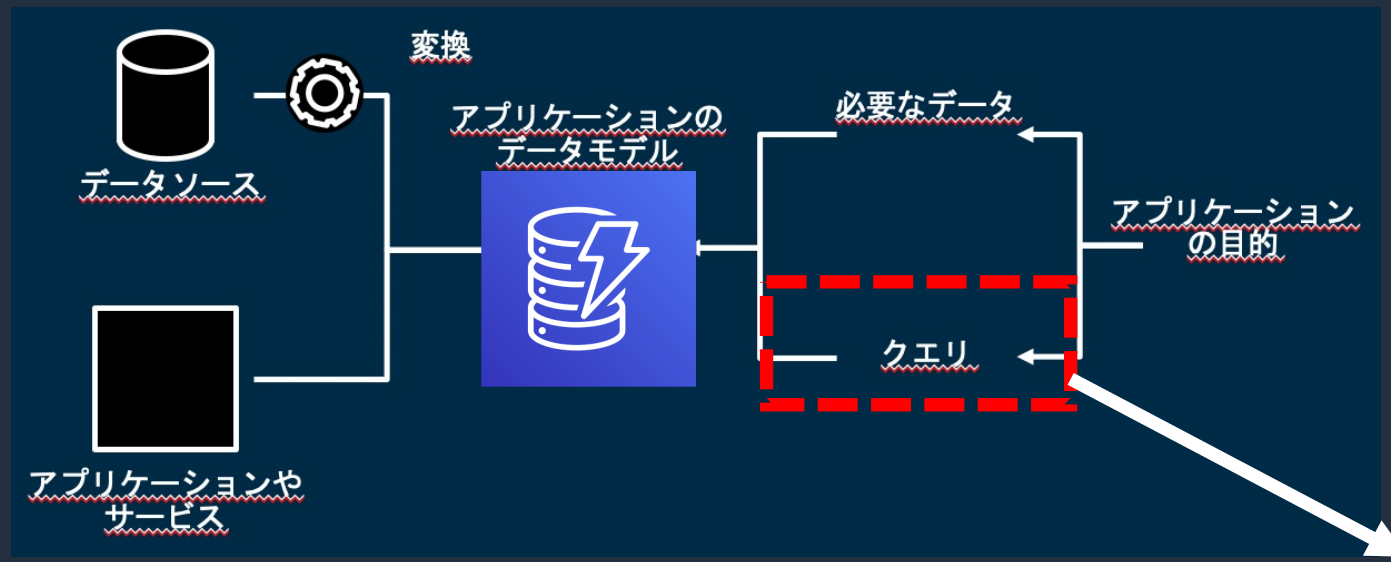
ユーザー: 1,000,000 以上
データ量: TB
場所: リージョナル
パフォーマンス: ミリ秒
リクエスト: 数万/sec
アクセス: ウェブ/モバイル
拡張: スケールアップ/ダウン、スケールアウト/イン
開発者アクセス: APIアクセス

Working backwards してみる ：再び



項目	内容
記事ID	abcde
作成日時	2020/10/21 00:00:00
執筆者	Akihiro Kuwano
タグ	blog
本文	＜テキスト＞
（コメント）	＜テキスト＞

Working backwards してみる ：再び

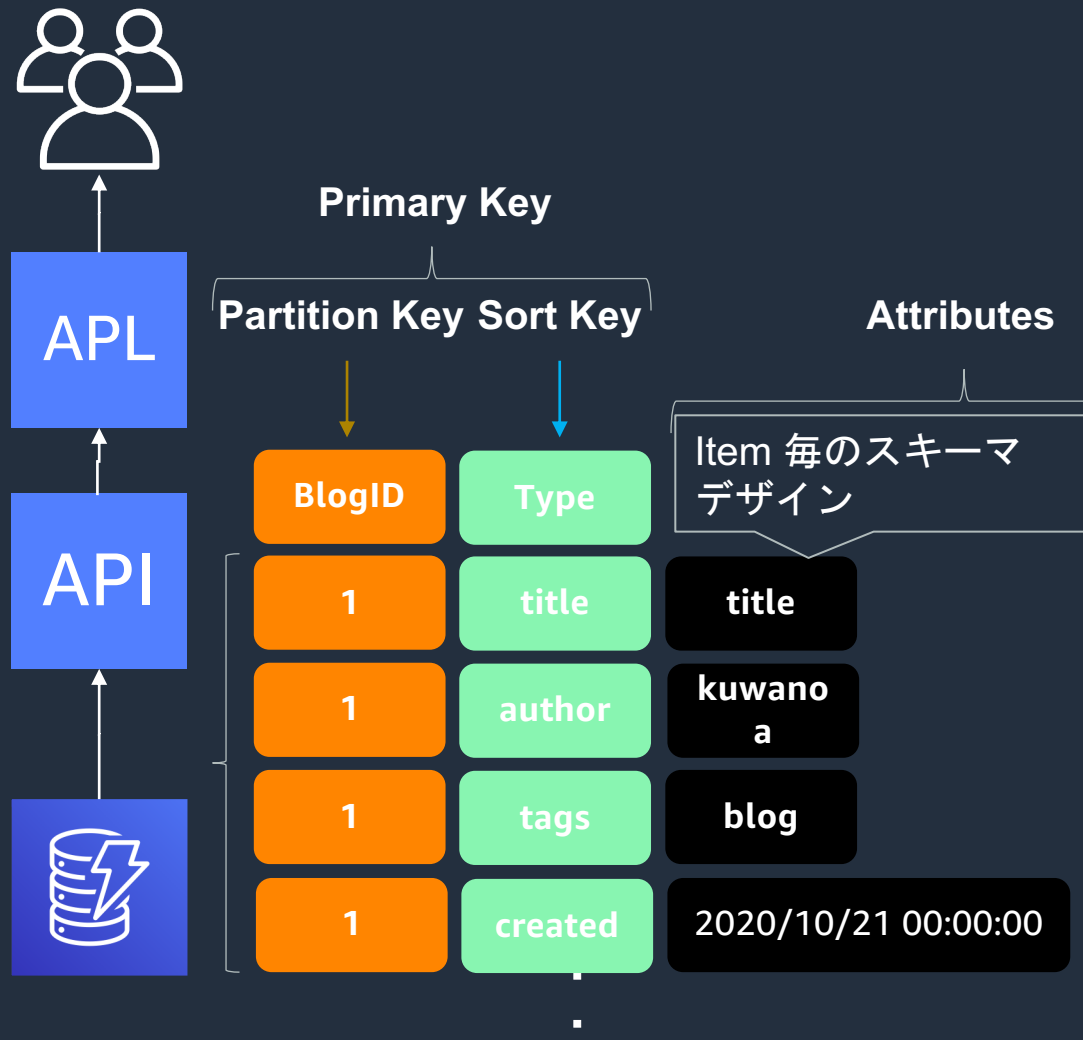


項目	内容
記事ID	abcde
作成日時	2020/10/21 00:00:00
執筆者	Akihiro Kuwano
タグ	blog
本文	<テキスト>
（コメント）	<テキスト>

クエリ
記事IDにマッチした記事を取得する
作成日時でソートしてXX件表時
執筆者で検索
タグで検索
記事IDについてるコメント一覧の取得

コンテンツ管理システム

- ニュースサイトなど、文書などのコンテンツ管理を行うシステム
- 不定形なデータ構造でも対応可能
  - モデリングの手法は変わる
- スケールなどについての考慮事項が減る
- 検索などはDynamoDB StreamsでOpenSearchなどに振ってもいい



データベース選定のまとめ

- 要件をブレイクダウンしていき最終的に一番メリットを享受できるデータベースを選択する
  - 高い開発効率、低い運用負荷など

- 同じアプリケーションほぼ同じ要件でも何を重視するかで選択肢は変わる

まとめ

まとめ

- AWSのデータベースサービスは様々な種類のものが存在している
- アプリケーション要件で使い分けることにより最適化することが可能＝Purpose-built Database
- 選び方はアプリケーションの要件から逆算して選ぶのがセオリー＝Working Backwards
- 新しいプロジェクトや、今のプロジェクトの改善の際にどんなデータベースを使えばいいか考えてみてはいかがでしょう！？

まとめ

- AWSのデータベースサービスは様々な種類のものが存在している
- アプリケーション要件で使い分けることにより最適化することが可能＝Purpose-built Database
- 選び方はアプリケーションの要件から逆算して選ぶのがセオリー＝Working Backwards
- 新しいプロジェクトや、今のプロジェクトの改善の際にどんなデータベースを使えばいいか考えてみてはいかがでしょう！？

- 😍＜迷ったらSAに相談してください！）

Thank you!