ユーザー行動が成す力学系の実現とそれを用いた推薦システムを支えるAWS アーキテクチャ

株式会社Gunosy 開発本部 データ分析部
機械学習エンジニア 米田 武

AWS Summit Tokyo 2018/06/01

自己紹介

- 米田 武 / Takeshi Yoneda / マスタケ
    - Github/Twitter: @mathetake
- 2017/03/31:
    - MSc. in Mathematics at Osaka University
- 2017/04/01~
    - Machine learning engineer at Gunosy Inc.
    - Apply mathematics to ...
        - Recommendation System
        - Machine learning
        - Optimization problems
    - data engineering

-今回のテーマ-

ニュースパスの推薦システムを支える
AWSアーキテクチャ

背景»「ニュースパス」の開発

✔ 一人ひとりのユーザーに対して最適な記事を届けるニュースアプリ

✔ 新着記事は**10,000/day+**

✔ 大量の記事と大量の行動ログから厳選した記事を選ぶためのアルゴリズム開発

✔ 高速かつ安定にアルゴリズムを実行するためのAWSアーキテクチャ

背景» ニュース推薦システムの難しさ

✔ ニュースの価値は基本的に時間減衰

Examples:

- ◯◯さんが死去
- サッカー日本代表が勝利
- ◯◯県で震度５強

- これらのニュースは時事性が強いため時間が経てば経つほど読まれにくくなる

✔ 既存アルゴリズムの単純適用不可

- ログが溜まる頃には価値↓
- ユーザー興味の変化サイクルが早い
- 表面的な言葉の一致度のみを取ると質が担保されない
- etc...

お話すること

✔ ユーザーの行動の数理モデルとその実現

» 数理的な背景と共にそれを支えるアーキテクチャを紹介

✔ 高速推薦システムとそれを支えるアーキテクチャ

» ユーザー行動の数理モデルを駆使した高速な推薦システムと

» それを支えるアーキテクチャをご紹介

今回関係する主なサービス

✔ Amazon s3

- ストレージサービス

✔ Amazon DynamoDB

- NoSQLデータベースサービス

✔ DynamoDB Accelerator(DAX)

- DynamoDB用インメモリキャッシュ

✔ AWS Lambda

- サーバーレスな計算環境

✔ Amazon Kinesis

- ストリーミングデータ処理サービス

✔ Amazon EMR

- マネージドなHadoopサービス

1. ユーザーの行動の数理モデルと
その実現

ユーザーの行動心理

読みたい“ニュース”ってなんだろう

✔ **Local Popularity(局所話題性)**が大事

- 自身のコミュニティ/近所で流行している記事が読みたい

✔ **野次馬のダイナミクス**

- 近くで大きなニュースが起きる => 野次馬が出来る
- 野次馬を見つける => なにが起こっているのか知りたい

ユーザーの行動心理

読みたい"ニュース"ってなんだろう

これらを実現するユーザー行動の数理モデルを作りたい

✔ **Local Popularity(局所話題性)**が大事

- 自身のコミュニティ/近所で流行している記事が読みたい

✔ **野次馬のダイナミクス**

- 近くで大きなニュースが起きる => 野次馬が出来る
- 野次馬を見つける => なにが起こっているのか知りたい

ユーザーの行動の数理モデル

✔ ビジネス要件

» リアルタイム性*

- リアルタイムにユーザーの行動が反映される

» スケーラビリティ

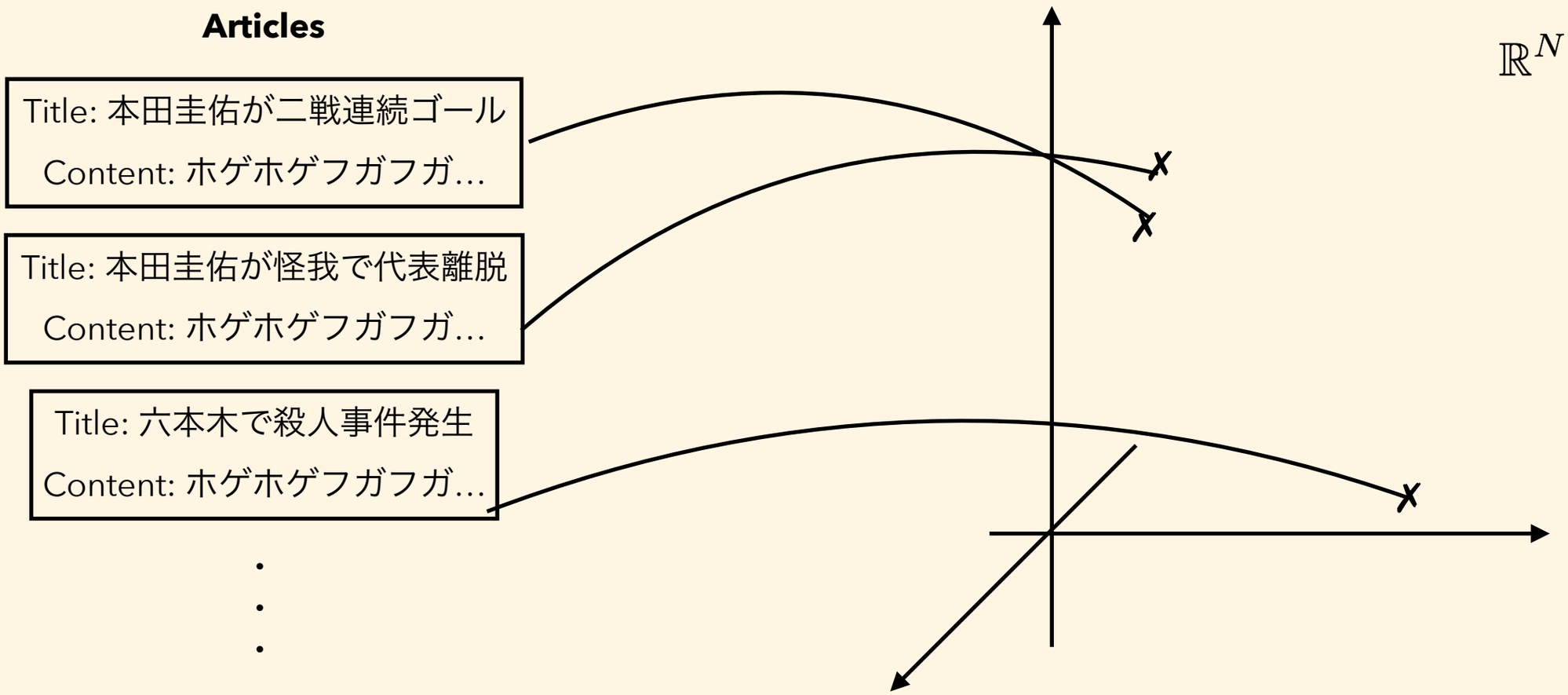
- ユーザーからの大量の行動ログをハンドリング
- サービス特有の高負荷に耐えうるアーキテクチャ

* 実際にはニアリアルタイムです

ユーザーの行動の数理モデル

By using common techniques of distributed word representation, we regard articles as embedded continuous vectors:

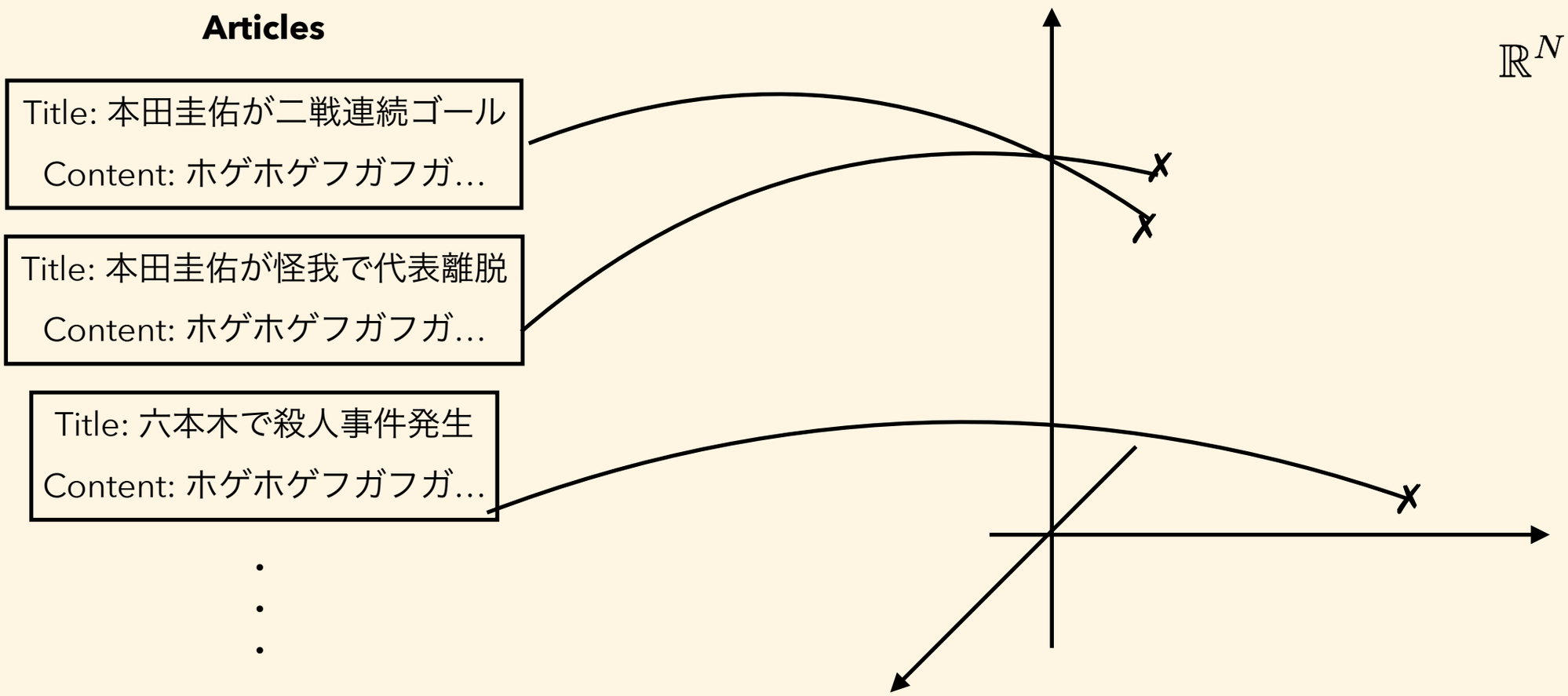
$$a \in \mathbb{R}^N : \text{articles}$$



ユーザーの行動の数理モデル

By using common techniques of distributed word representation, we regard articles as embedded continuous vectors: ニュース記事を連続なベクトルとして埋め込む

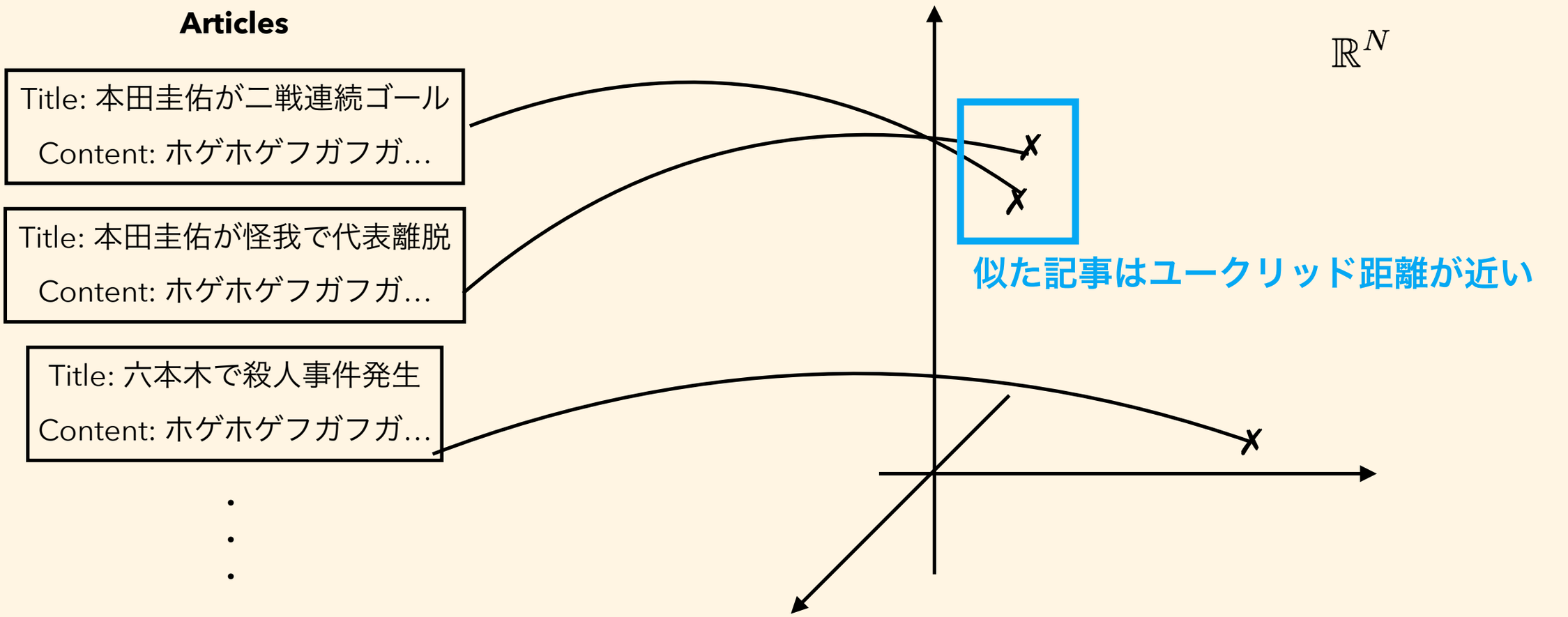
$$a \in \mathbb{R}^N : \text{articles}$$



ユーザーの行動の数理モデル

By using common techniques of distributed word representation, we regard articles as embedded continuous vectors:

$$a \in \mathbb{R}^N : \text{articles}$$



ユーザーの行動の数理モデル

Let u an user and $A^u = \{a_1^u, \ldots, a_M^u\}$ the set of recent M articles clicked by u. We define the *user vector* of user u as the weighted average of vectors in A^u:

$$u := \frac{1}{M} \sum_{a \in A^u}^{M} w_a a \in \mathbb{R}^N, w_a \in \mathbb{R}$$

where $\{w_a\}_{a \in A^u}$ are determined adaptively.

When u clicks an article a^*, his/her user vector is updated by

$$\begin{aligned} A^u_{\text{new}} &:= \{a^*, a_1^u, \ldots, a_{M-1}^u\} \\ u_{\text{new}} &:= \frac{1}{M} \sum_{a \in A^u_{\text{new}}} w_a a \end{aligned}$$

ユーザーの行動の数理モデル

Let u an user and $A^u = \{a_1^u, \ldots, a_M^u\}$ the set of recent M articles clicked by u. We define the *user vector* of user u as the weighted average of vectors in A^u:

クリックした直近M記事の重み付け平均

$$u := \frac{1}{M} \sum_{a \in A^u}^{M} w_a a \in \mathbb{R}^N, w_a \in \mathbb{R}$$

where $\{w_a\}_{a \in A^u}$ are determined adaptively.

When u clicks an article a^*, his/her user vector is updated by

$$\begin{aligned} A^u_{\text{new}} &:= \{a^*, a_1^u, \ldots, a_{M-1}^u\} \\ u_{\text{new}} &:= \frac{1}{M} \sum_{a \in A^u_{\text{new}}} w_a a \end{aligned}$$

ユーザーの行動の数理モデル

Let u an user and $A^u = \{a_1^u, \ldots, a_M^u\}$ the set of recent M articles clicked by u. We define the *user vector* of user u as the weighted average of vectors in A^u:

$$u := \frac{1}{M} \sum_{a \in A^u}^{M} w_a a \in \mathbb{R}^N, w_a \in \mathbb{R}$$

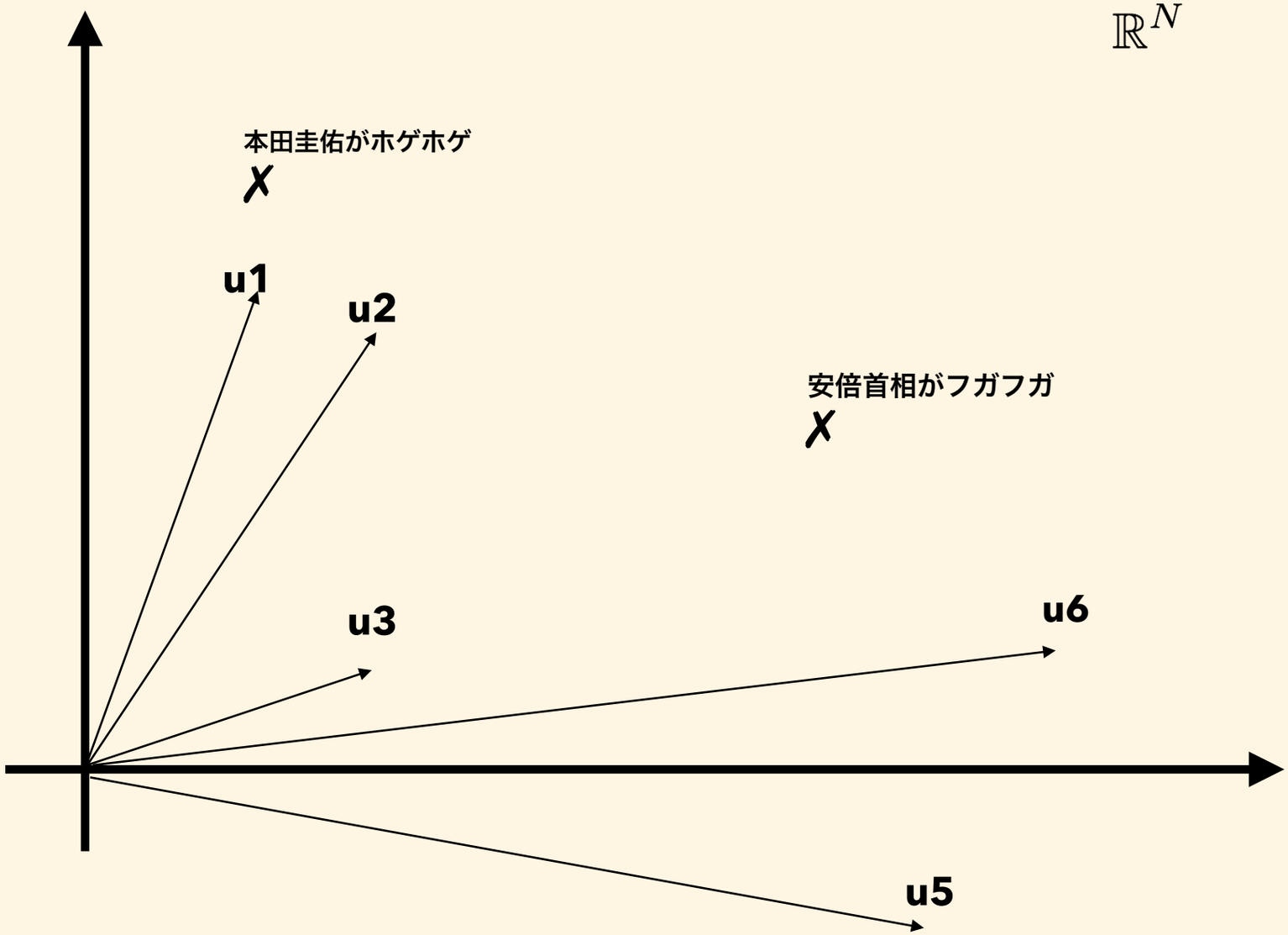
where $\{w_a\}_{a \in A^u}$ are determined adaptively.

When u clicks an article a^*, his/her user vector is updated by

$$\begin{aligned} A^u_{\text{new}} &:= \{a^*, a_1^u, \ldots, a_{M-1}^u\} \\ u_{\text{new}} &:= \frac{1}{M} \sum_{a \in A^u_{\text{new}}} w_a a \end{aligned}$$

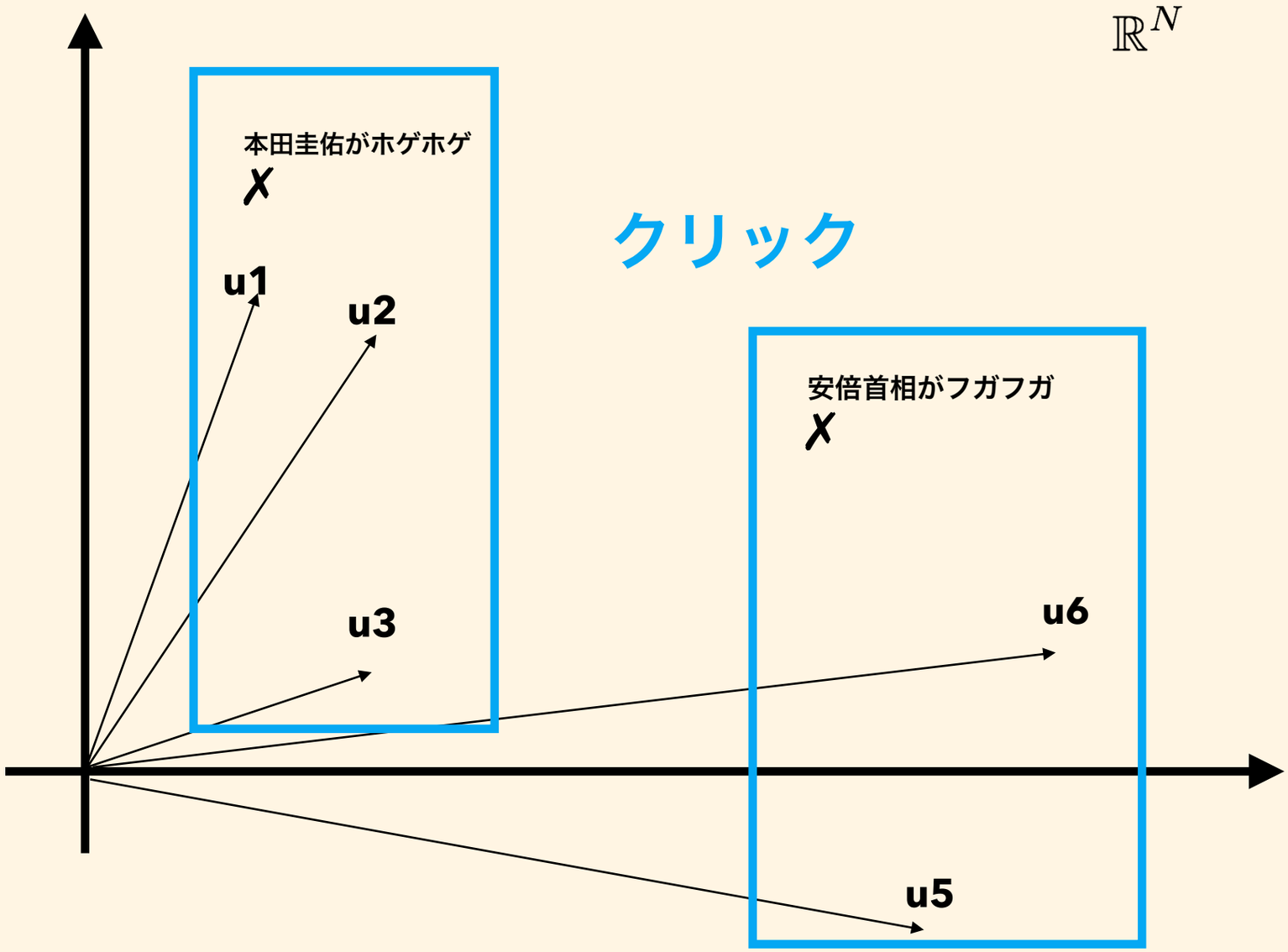
記事をクリックするとlpush/rtrimでログが更新され
それに基いてベクトルが再構築される

ユーザーの行動の数理モデル



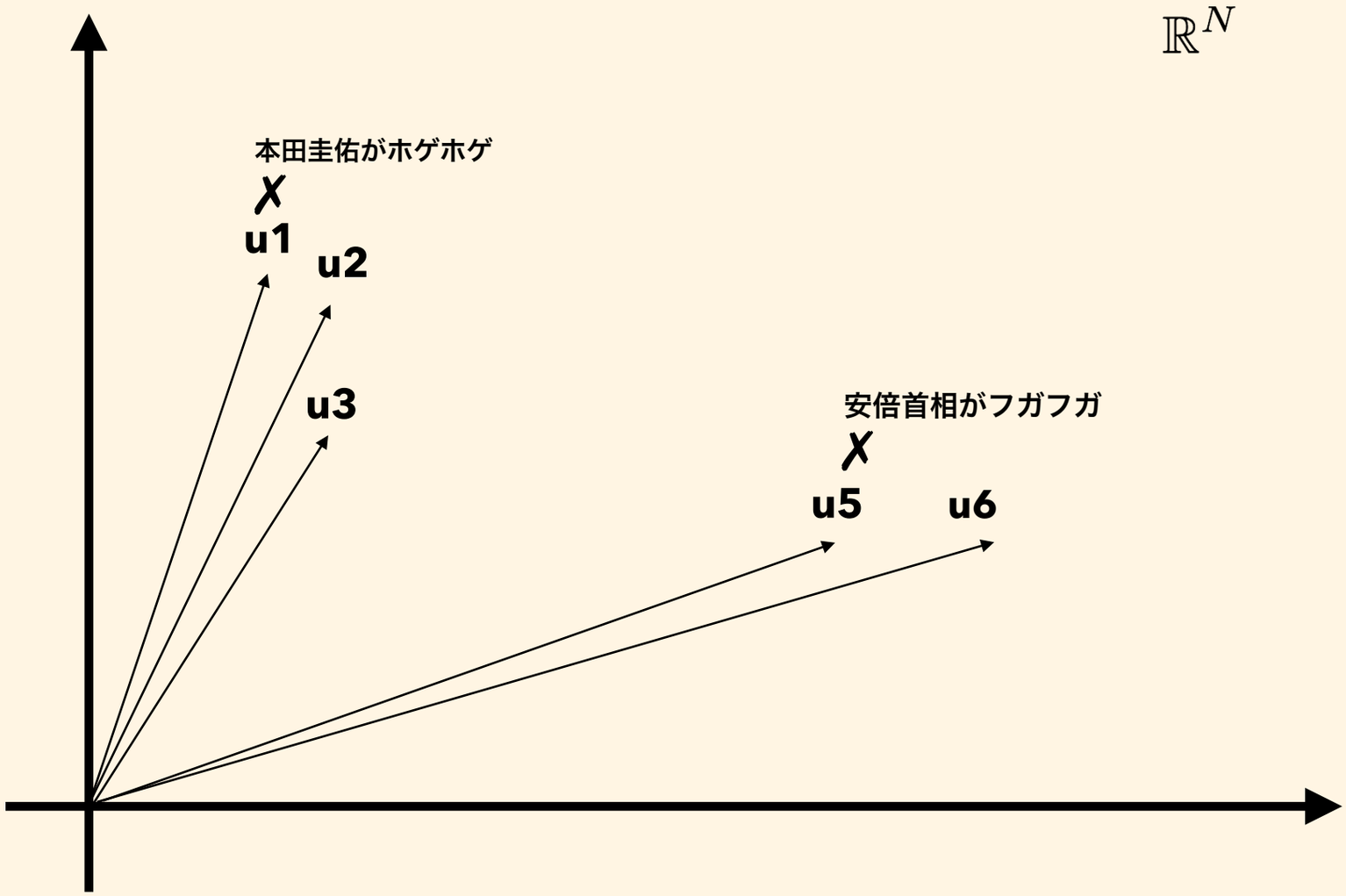
#1. ユーザー行動の数理モデルとその実現

ユーザーの行動の数理モデル



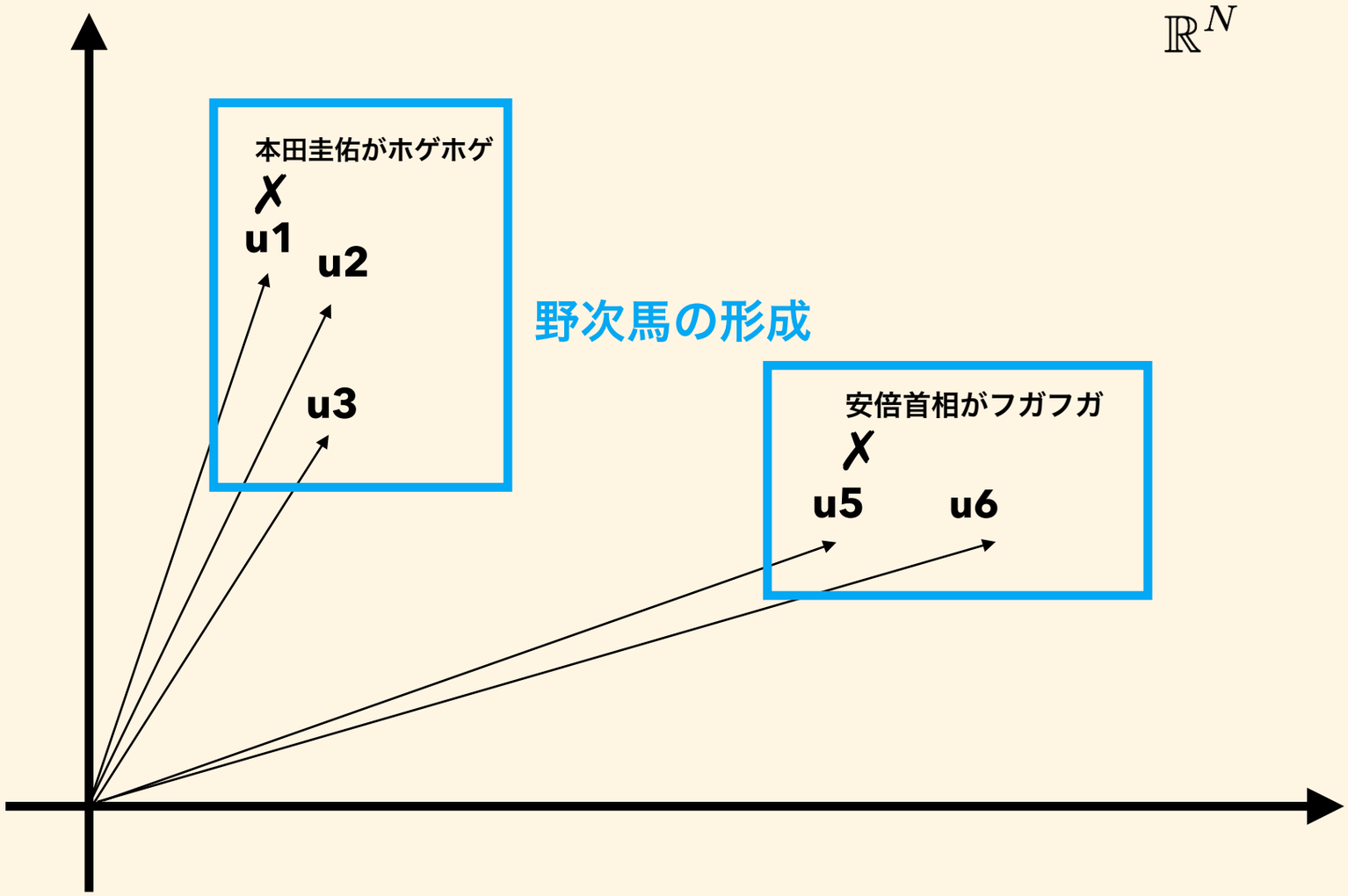
#1. ユーザー行動の数理モデルとその実現

ユーザーの行動の数理モデル



#1. ユーザー行動の数理モデルとその実現

ユーザーの行動の数理モデル



野次馬のダイナミクスを表現

#1. ユーザー行動の数理モデルとその実現

ユーザーの行動の数理モデル

Let u an user and $A^u = \{a_1^u, \ldots, a_M^u\}$ the set of recent M articles clicked by u. We define the *user vector* of user u as the weighted average of vectors in A^u:

$$u := \frac{1}{M} \sum_{a \in A^u}^{M} w_a a \in \mathbb{R}^N, w_a \in \mathbb{R}$$

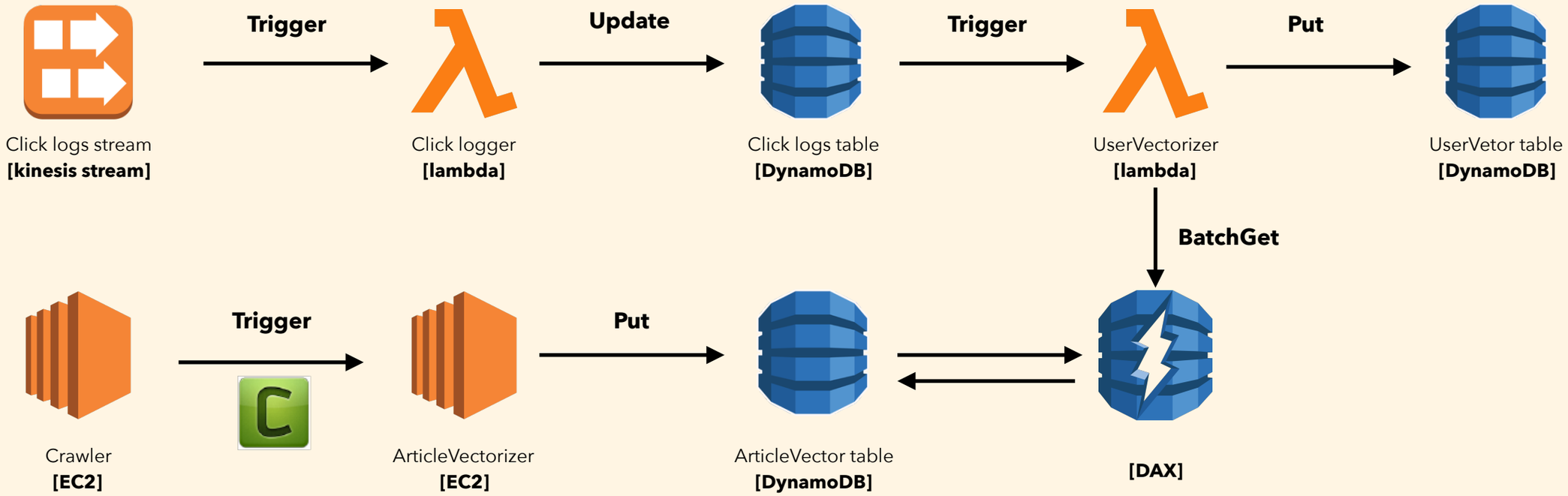
where $\{w_a\}_{a \in A^u}$ are determined adaptively.

When u clicks an article a^*, his/her user vector is updated by

$$\begin{aligned} A^u_{\text{new}} &:= \{a^*, a_1^u, \ldots, a_{M-1}^u\} \\ u_{\text{new}} &:= \frac{1}{M} \sum_{a \in A^u_{\text{new}}} w_a a \end{aligned}$$

即時性を担保しながら実行したい

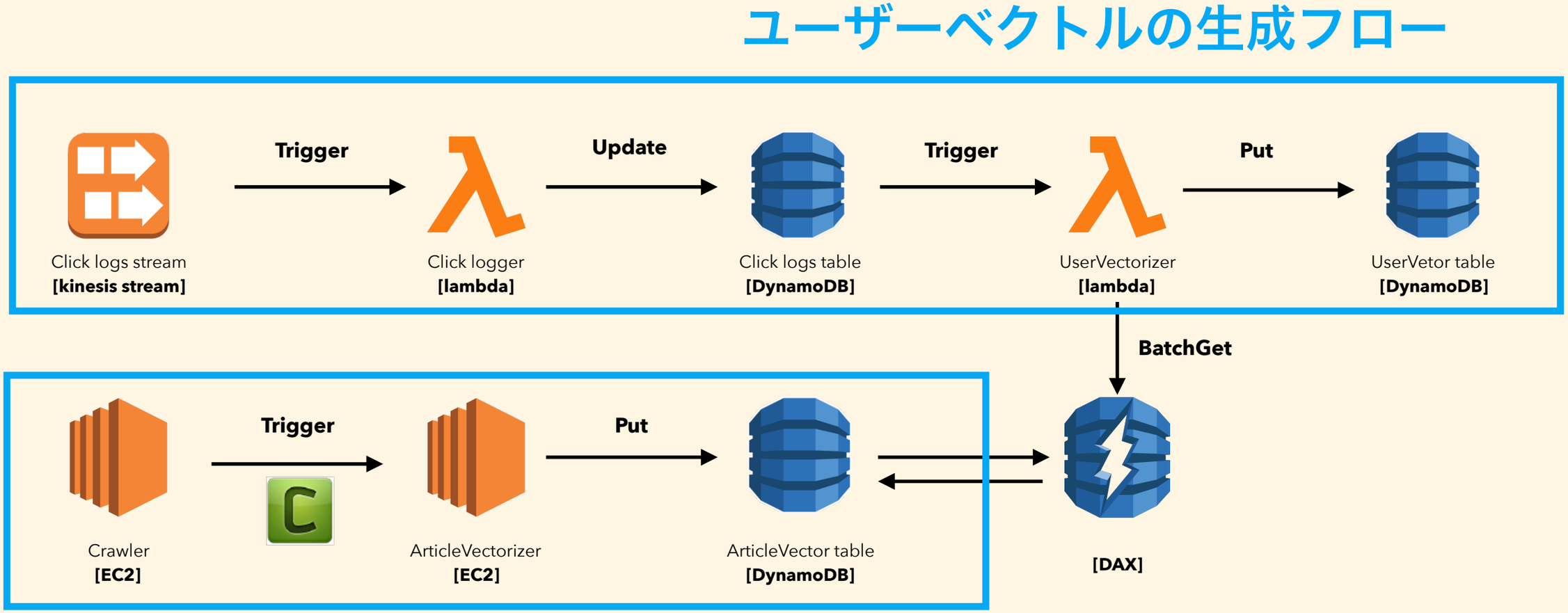
ダイナミクスの実現



詳細な議論: Gunosy テックブログ, “Gunosyのパーソナライズを支える技術 -1クリックで始まるパーソナライズ-” http://tech.gunosy.io/entry/realtime-vectorization-with-dynamodb

#1. ユーザー行動の数理モデルとその実現

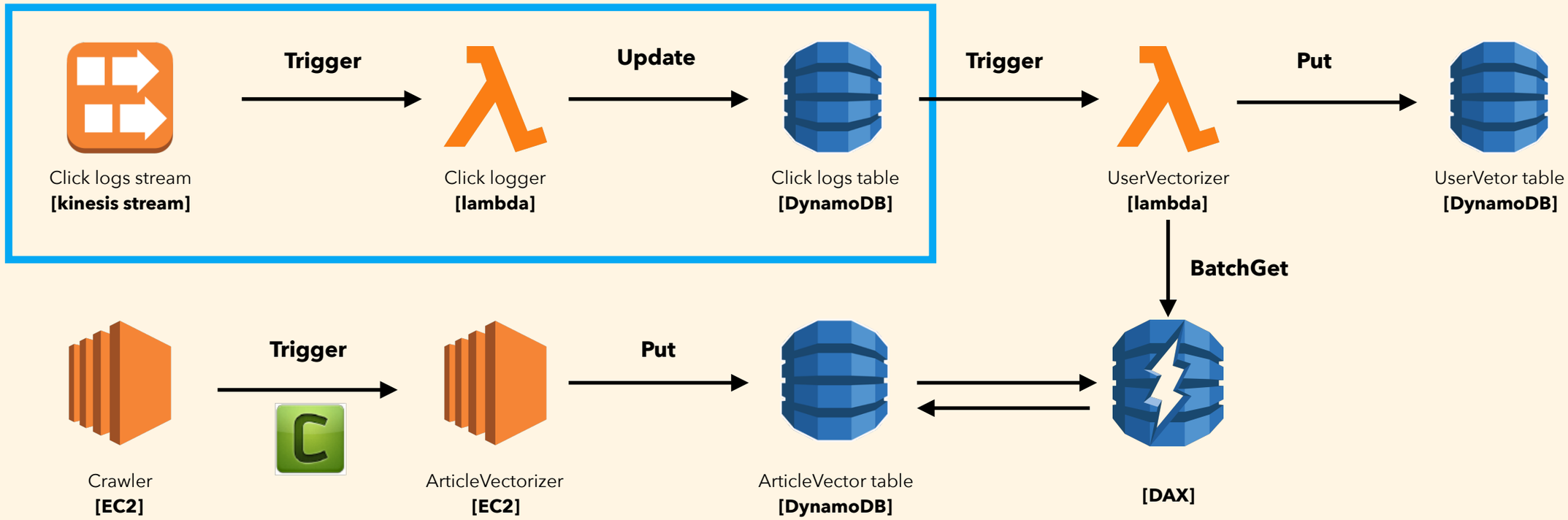
ダイナミクスの実現



詳細な議論: Gunosy テックブログ, “Gunosyのパーソナライズを支える技術 -1クリックで始まるパーソナライズ-” http://tech.gunosy.io/entry/realtime-vectorization-with-dynamodb

#1. ユーザー行動の数理モデルとその実現

ダイナミクスの実現



クリックログのグルーピング処理

詳細な議論: Gunosy テックブログ, “Gunosyのパーソナライズを支える技術 -1クリックで始まるパーソナライズ-” http://tech.gunosy.io/entry/realtime-vectorization-with-dynamodb

#1. ユーザー行動の数理モデルとその実現

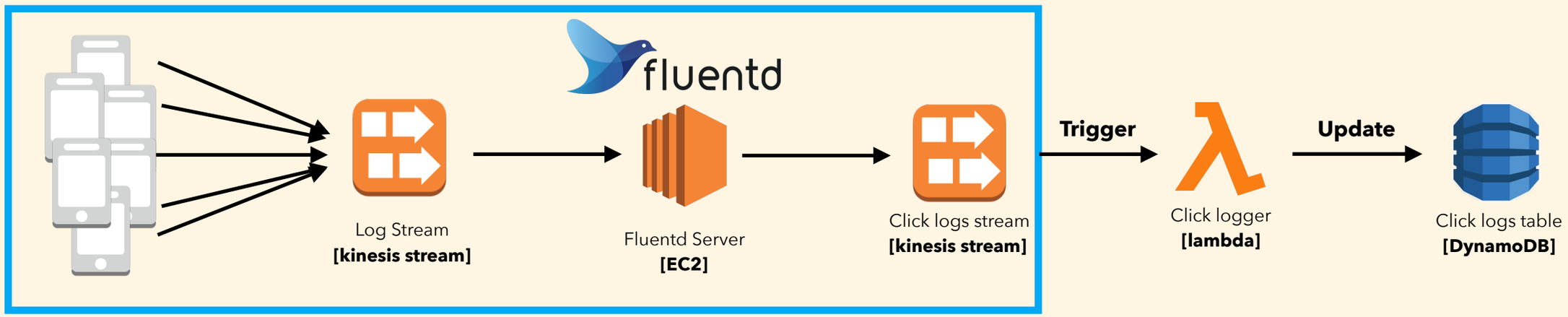
クリックログのグルーピング処理

✔ **Client app ⇒ Log stream[kinesis stream] ⇒ fluentd Server[EC2]**

- クリックに限らない各種ログが集約

✔ **fluentd Server ⇒ Click logs stream[kinesis stream]**

- クロックログがfilterされAmazon kinesis streamへ



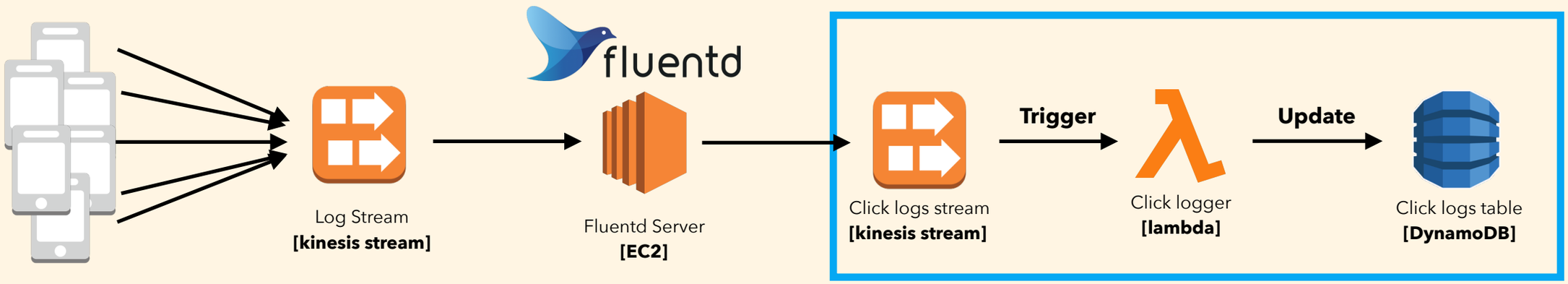
クリックログのグルーピング処理

✔ Click logs stream ⇒ Click logger[AWS lambda]

- Kinesis streamをトリガーとして設定

✔ Click logger ⇒ Click logs table[DynamoDB]

- updateリクエストにより直近M記事のクリックログのみ保持



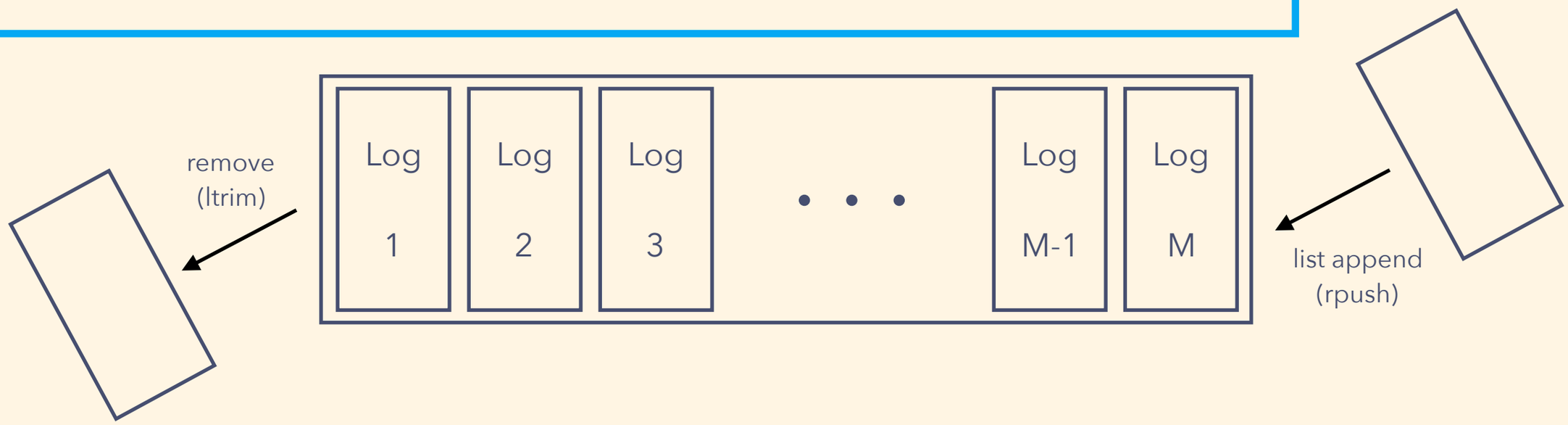
クリックログのグルーピング処理

✔ Click logs stream ⇒ Click logger[AWS lambda]

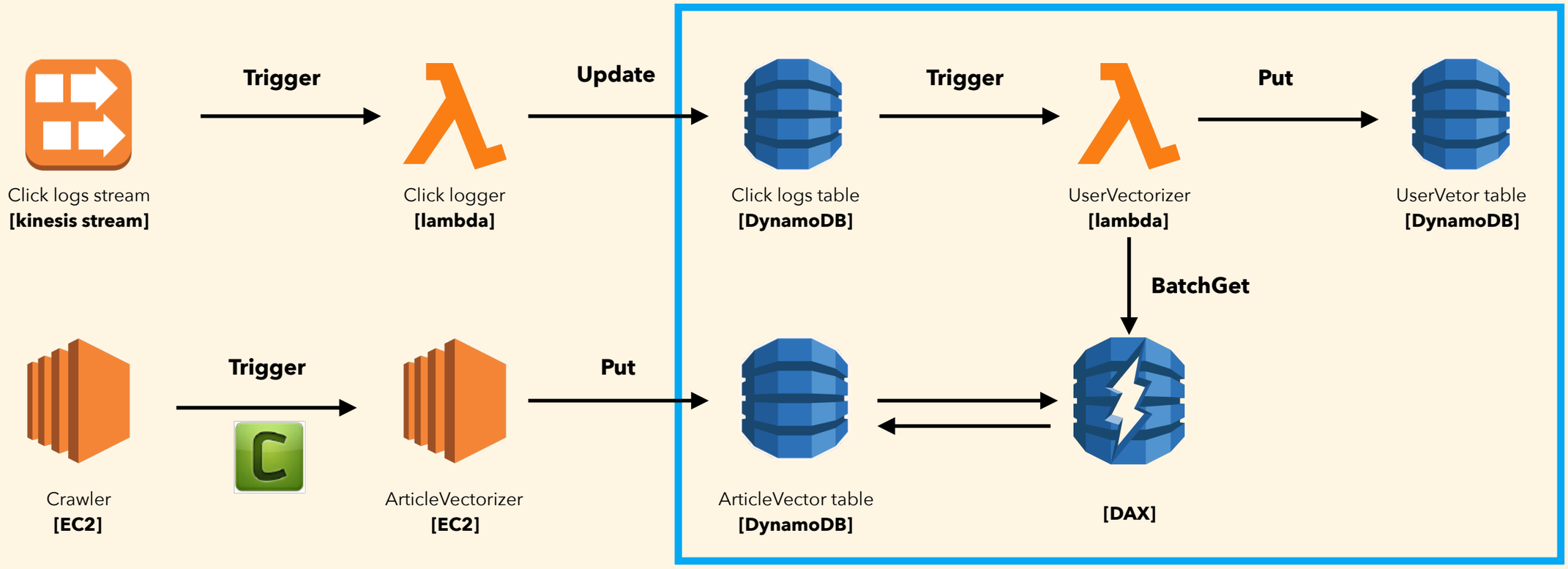
- Kinesis streamをトリガーとして設定

✔ Click logger ⇒ Click logs table[DynamoDB]

- updateリクエストにより直近M記事のクリックログのみ保持



ダイナミクスの実現



ユーザーのベクトル化処理

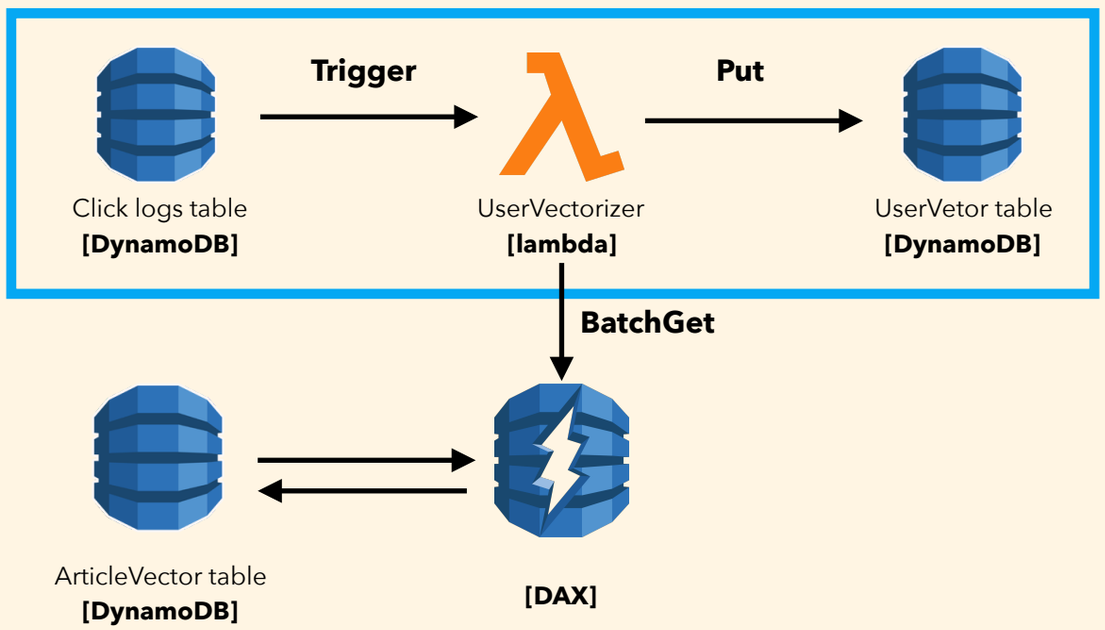
詳細な議論: Gunosy テックブログ, “Gunosyのパーソナライズを支える技術 -1クリックで始まるパーソナライズ-” http://tech.gunosy.io/entry/realtime-vectorization-with-dynamodb

#1. ユーザー行動の数理モデルとその実現

ユーザーのベクトル化処理

✔ Click logs table -> UserVectorizer[AWS lambda]

- DynamoDBStreamをトリガーとして起動
- 直近のクリックログと記事ベクトルを取得して計算⇒ PUT



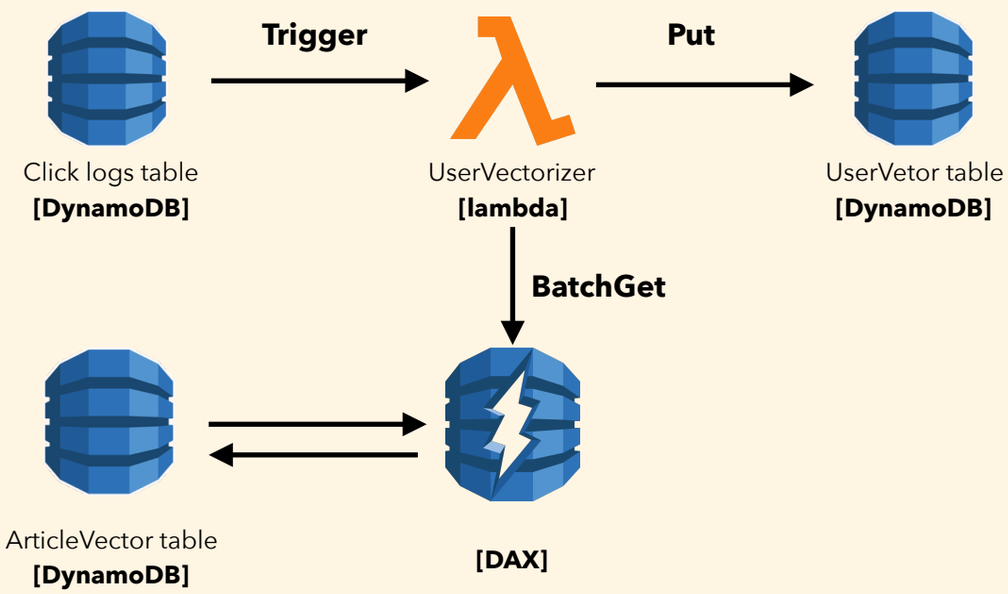
ユーザーのベクトル化処理

✔ Click logs table -> UserVectorizer[AWS lambda]

- DynamoDBStreamをトリガーとして起動
- 直近のクリックログと記事ベクトルを取得して計算⇒ PUT

複数Lambda間で共有出来る

レイテンシ・コストを考えキャッシュしたい



ユーザーのベクトル化処理

✔ **UserVectorizer –-(DAX) –-> ArticleVector table[DynamoDB]**

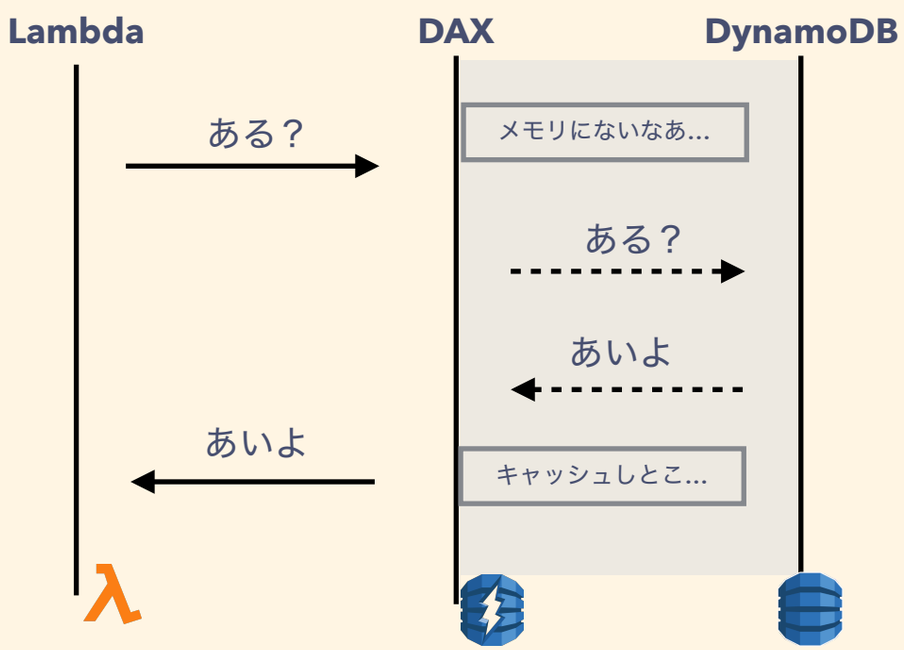
- DynamoDBへのGETリクエストのキャッシュといえば **DAX**

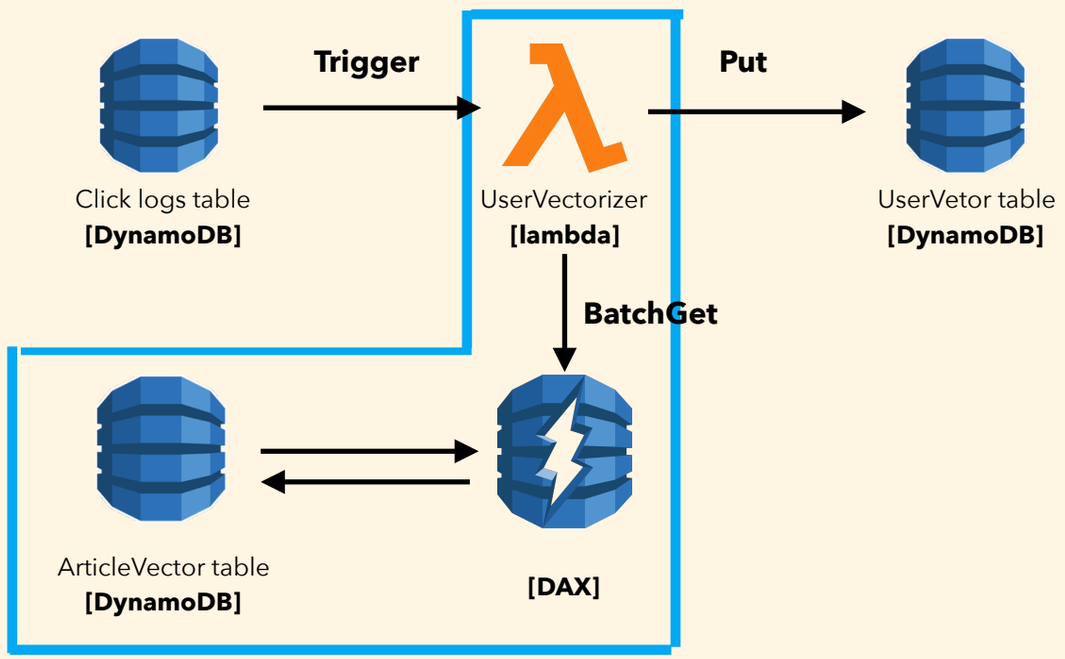
- UserVectorizerが使用する記事ベクトルの取得にキャッシュを挟む

ユーザーのベクトル化処理

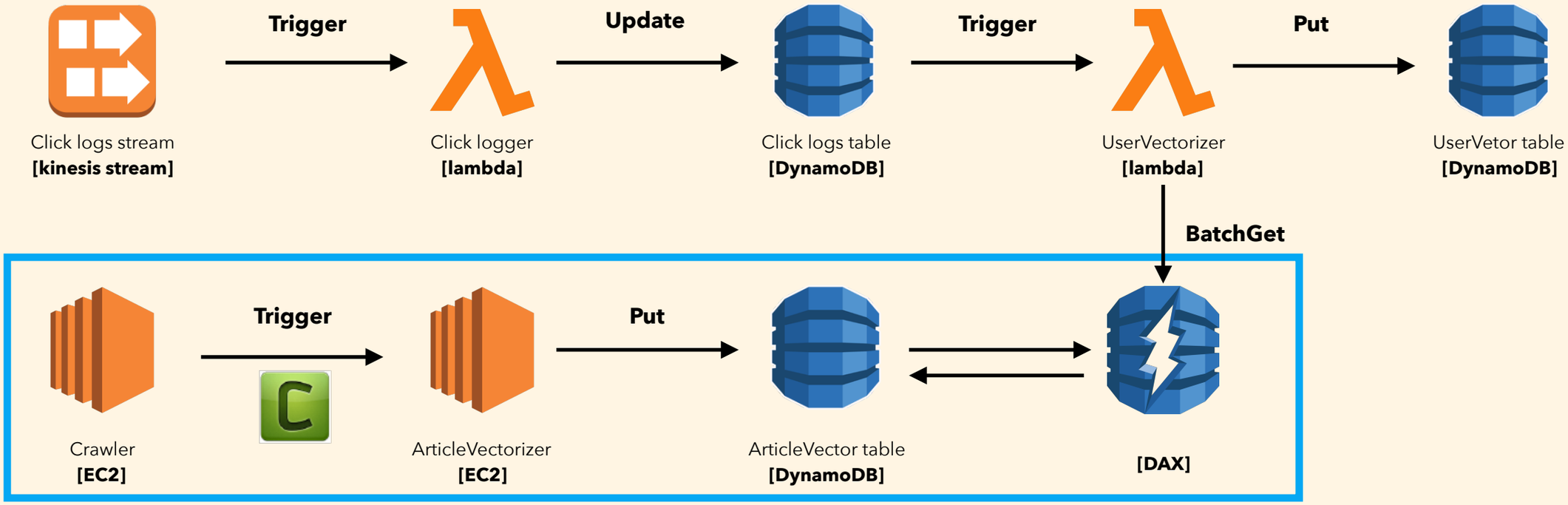
✔ DAX(Amazon DynamoDB Accelerator)

- DynamoDBの前段に(論理的に)置かれるインメモリキャッシュ
- ReadHeavyな処理に効果を発揮
- SDKはDAXと低レベルTCPで会話 ⇒ 低レイテンシー高スループット





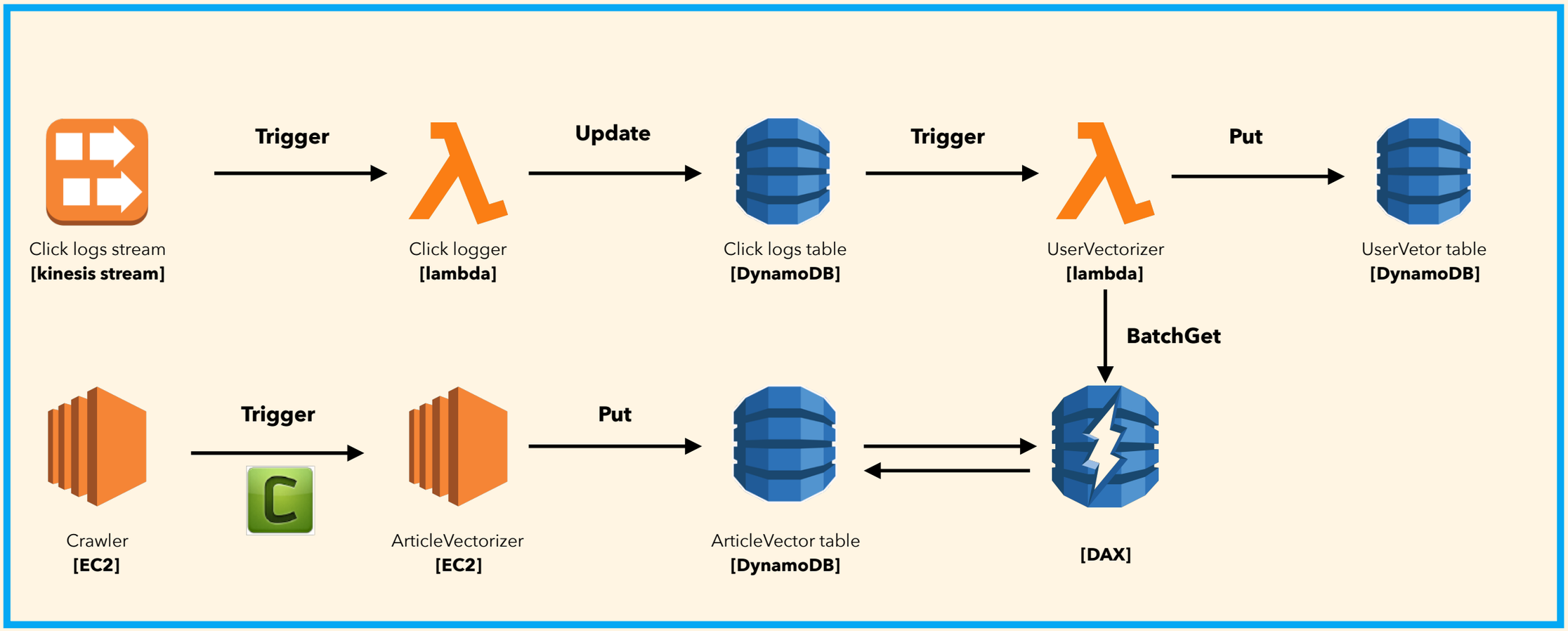
ダイナミクスの実現



✔ **記事のベクトル化処理**

- Crawlerが新着記事を取得し必要な情報と共にエンキュー
- ArticleVectorizer[EC2]がベクトル化処理をしてArticleVector tableにPUT

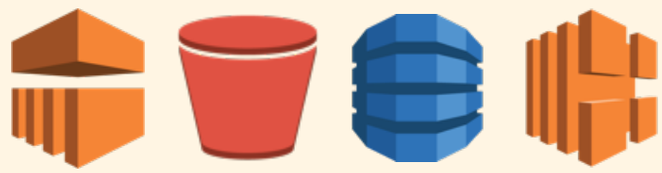
ダイナミクスの実現



野次馬のダイナミクスを(ほぼ)サーバーレスに実現

#1. ユーザー行動の数理モデルとその実現

2. 高速推薦システムと
それを支えるデータレイク



行動モデルから推薦へ

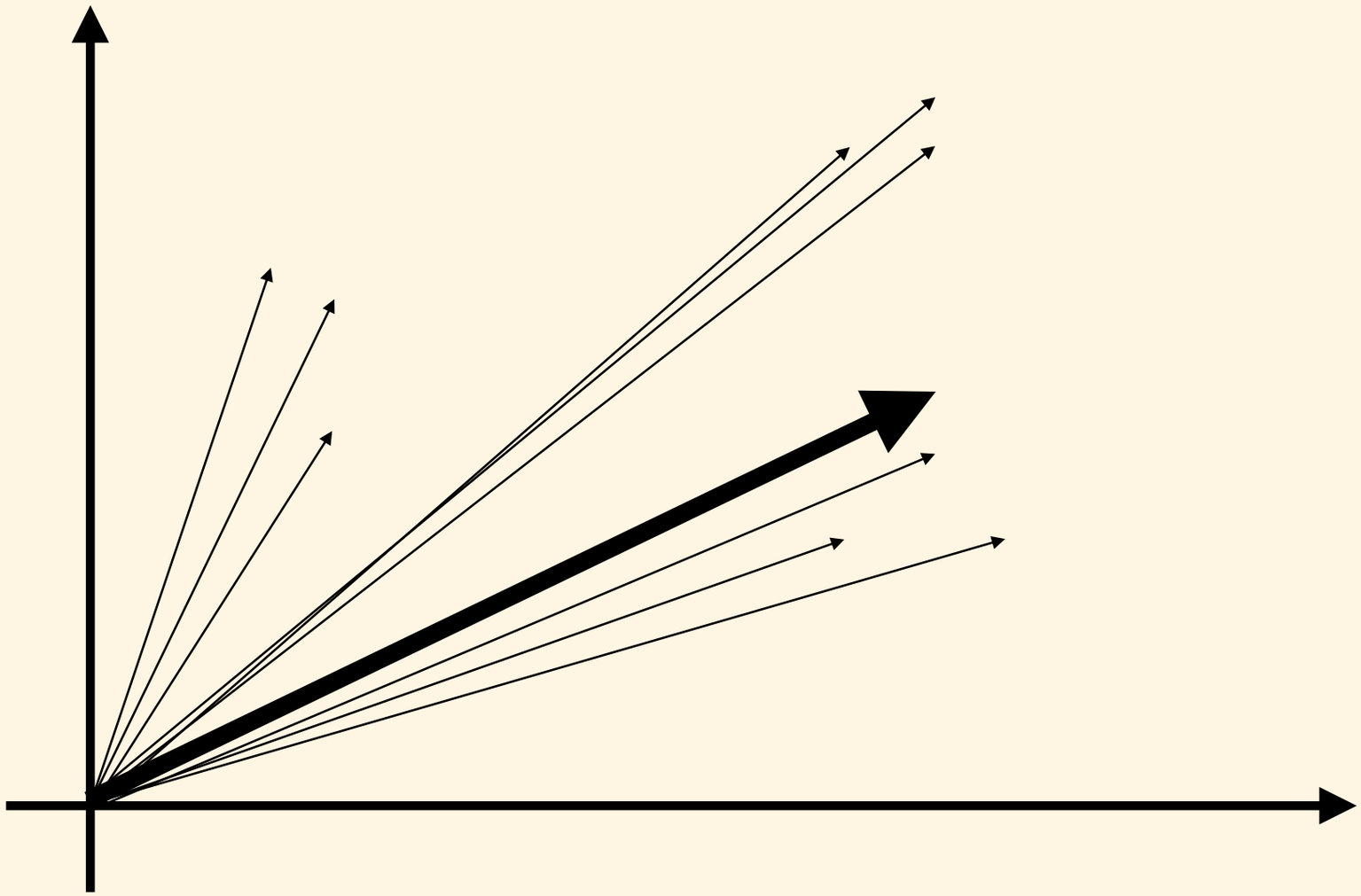
✔ ユーザー行動の数理モデルは実現出来た

- Kinesis / DynamoDB / DAX / lambda
- “ほぼ” サーバーレスに実現
- 十分に高いスケーラビリティ

✔ これを用いた高速/高精度な推薦システムを構築したい

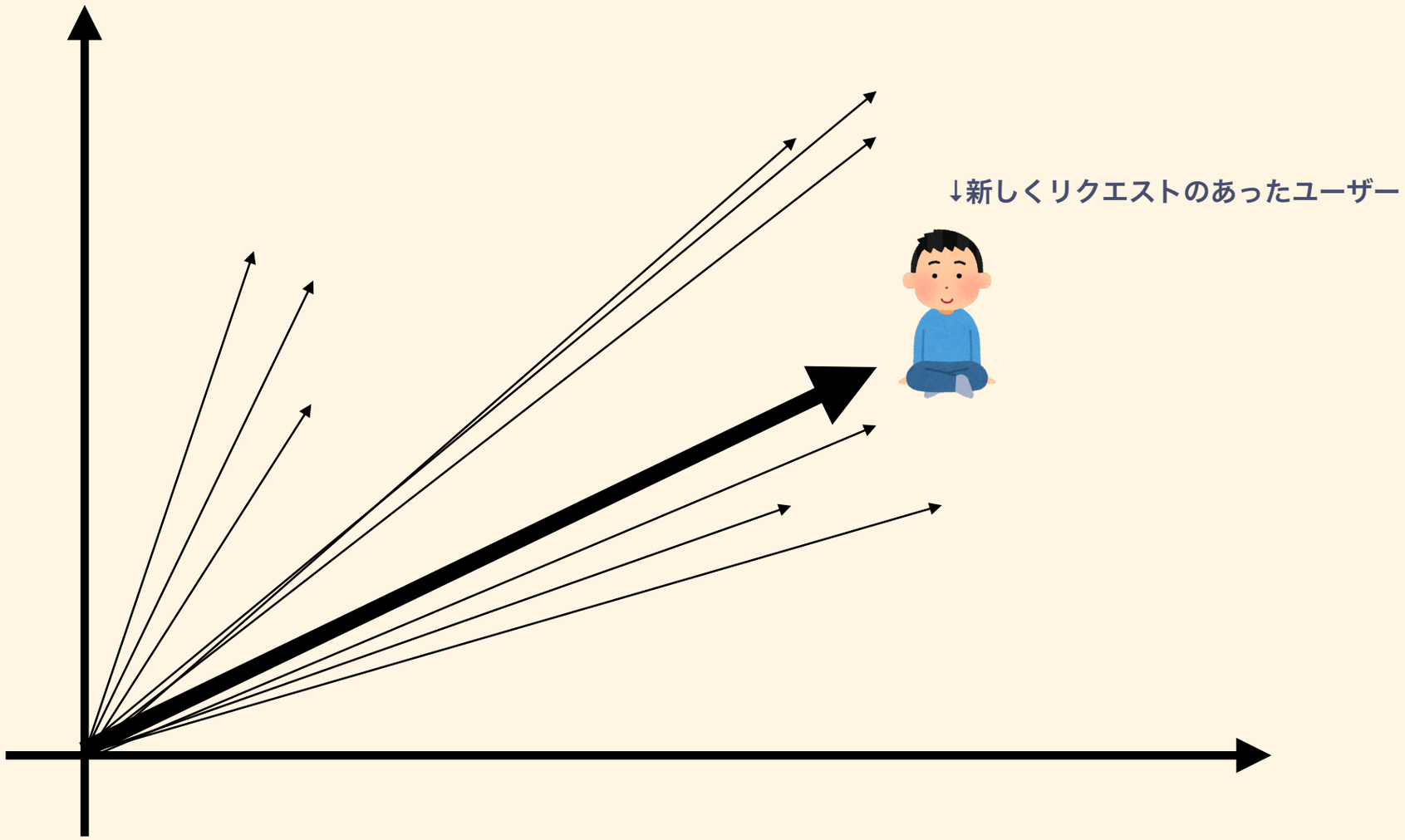
行動モデルから推薦へ

✔ 推薦リスト生成 = 流行探し



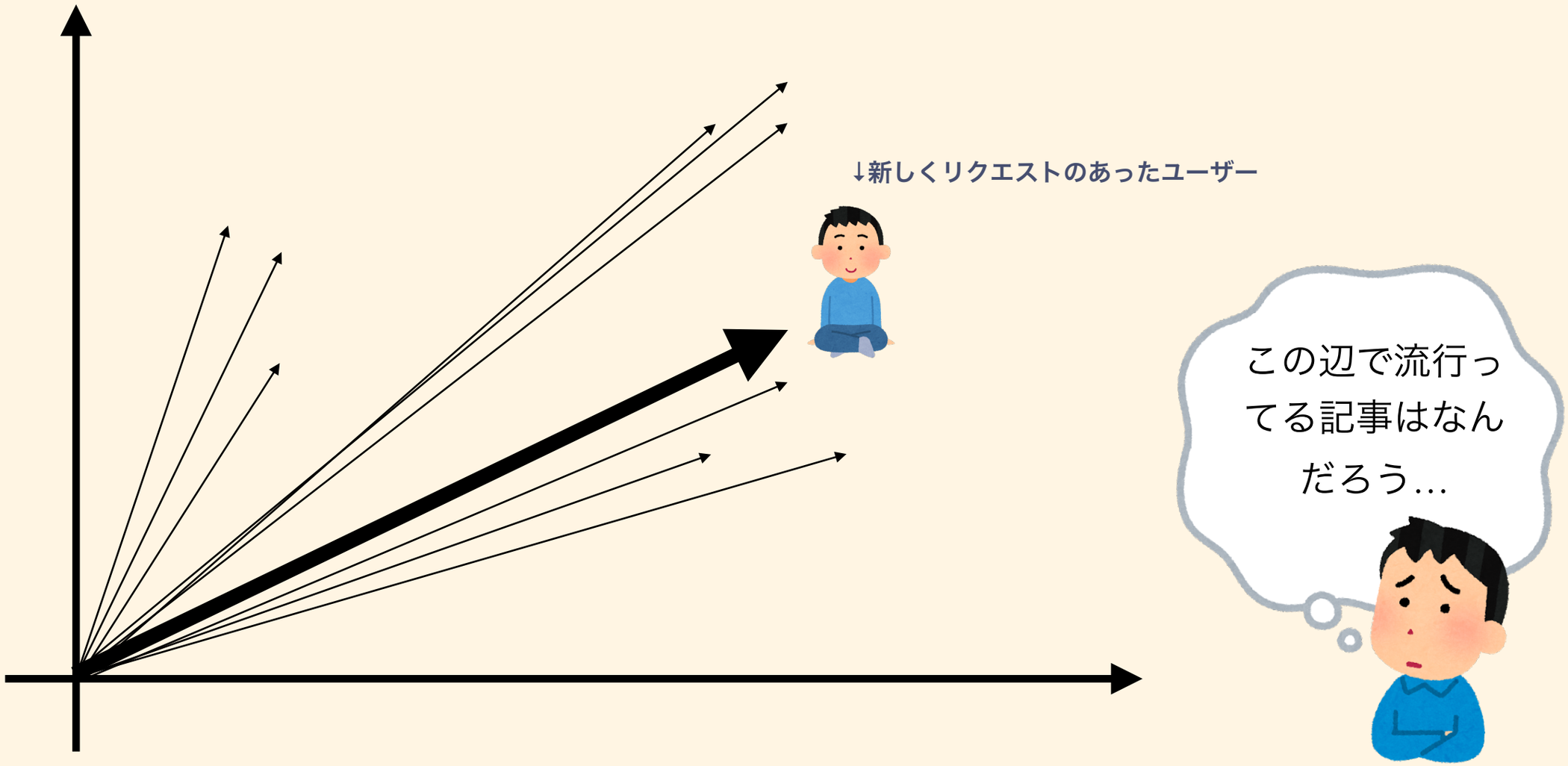
行動モデルから推薦へ

✔ 推薦リスト生成 = 流行探し



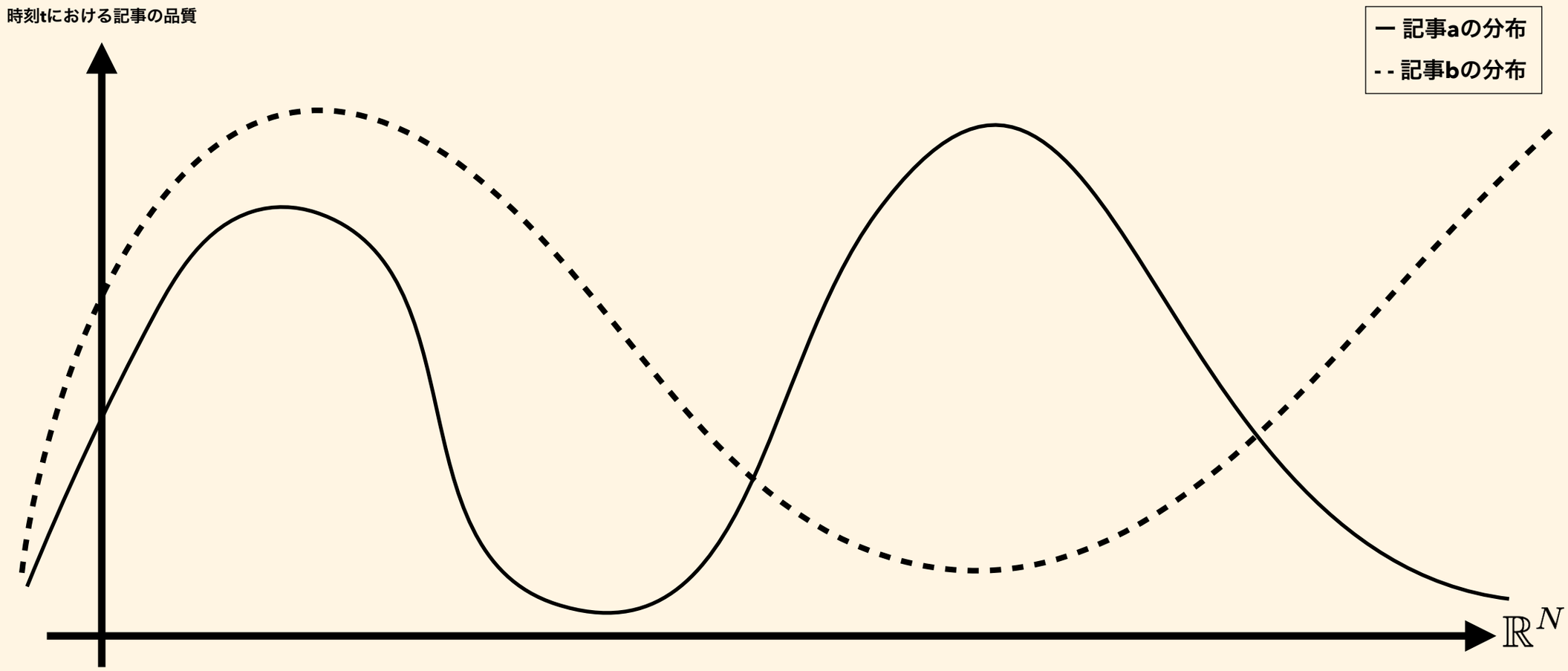
行動モデルから推薦へ

✔ 推薦リスト生成 = 流行探し



行動モデルから推薦へ

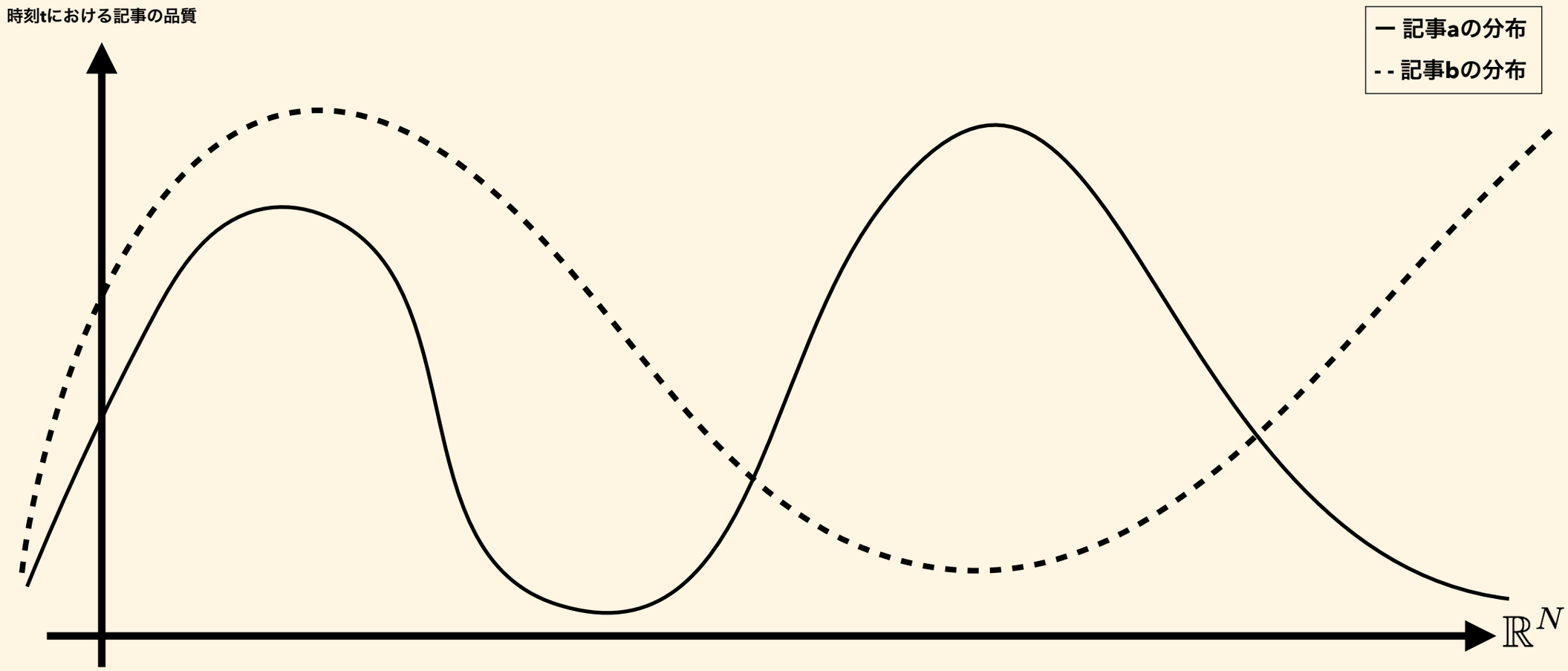
✔ 記事の”質”(流行り度, 鮮度 etc.) をベクトル空間上の分布として表現



行動モデルから推薦へ

✔ ユーザーからのリクエスト時、そのベクトルの周りでの密度の高さを計算

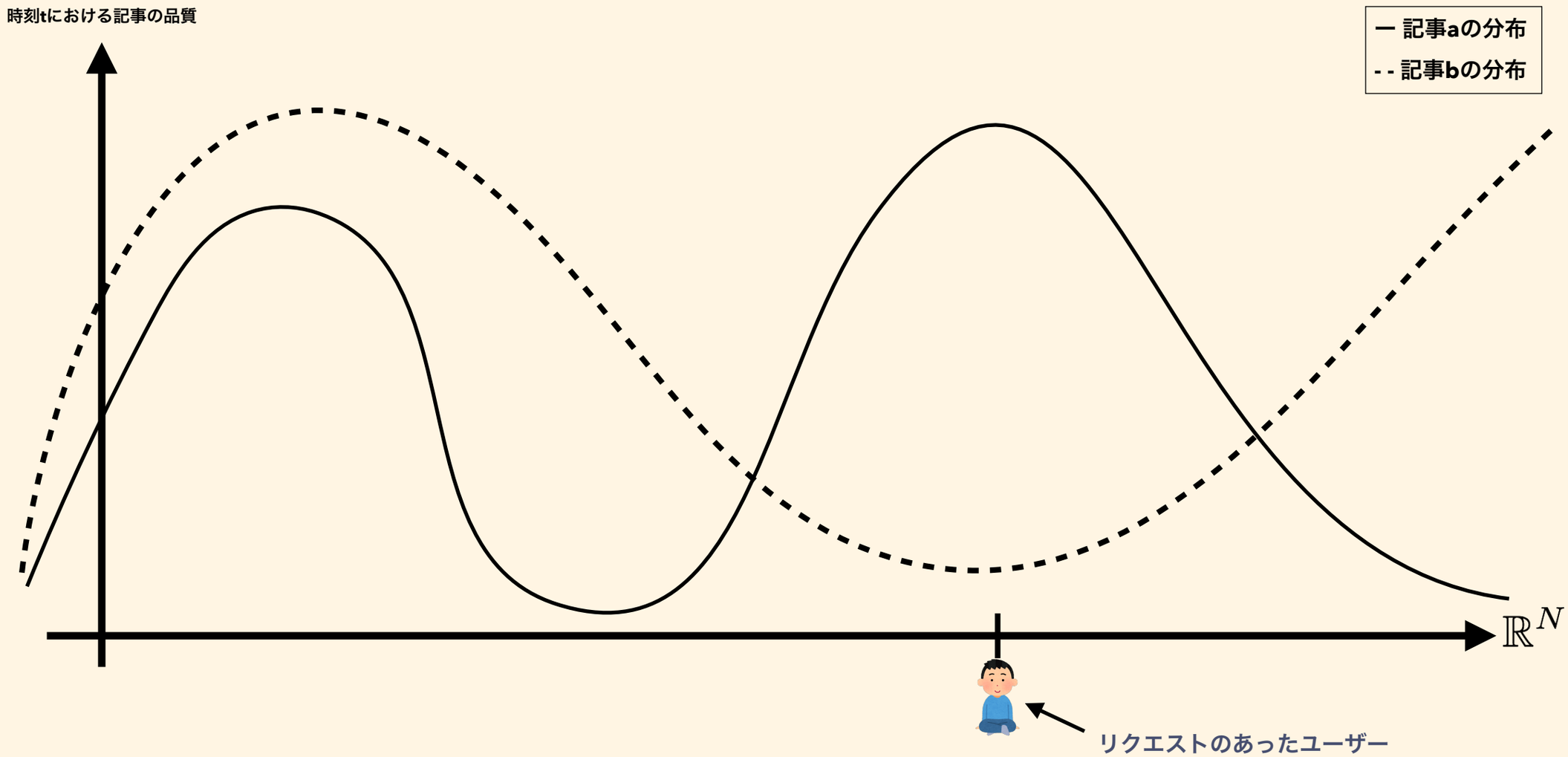
✔ その密度をスコアとして候補となる記事をソートしてユーザーに表示する



行動モデルから推薦へ

✔ ユーザーからのリクエスト時、そのベクトルの周りでの密度の高さを計算

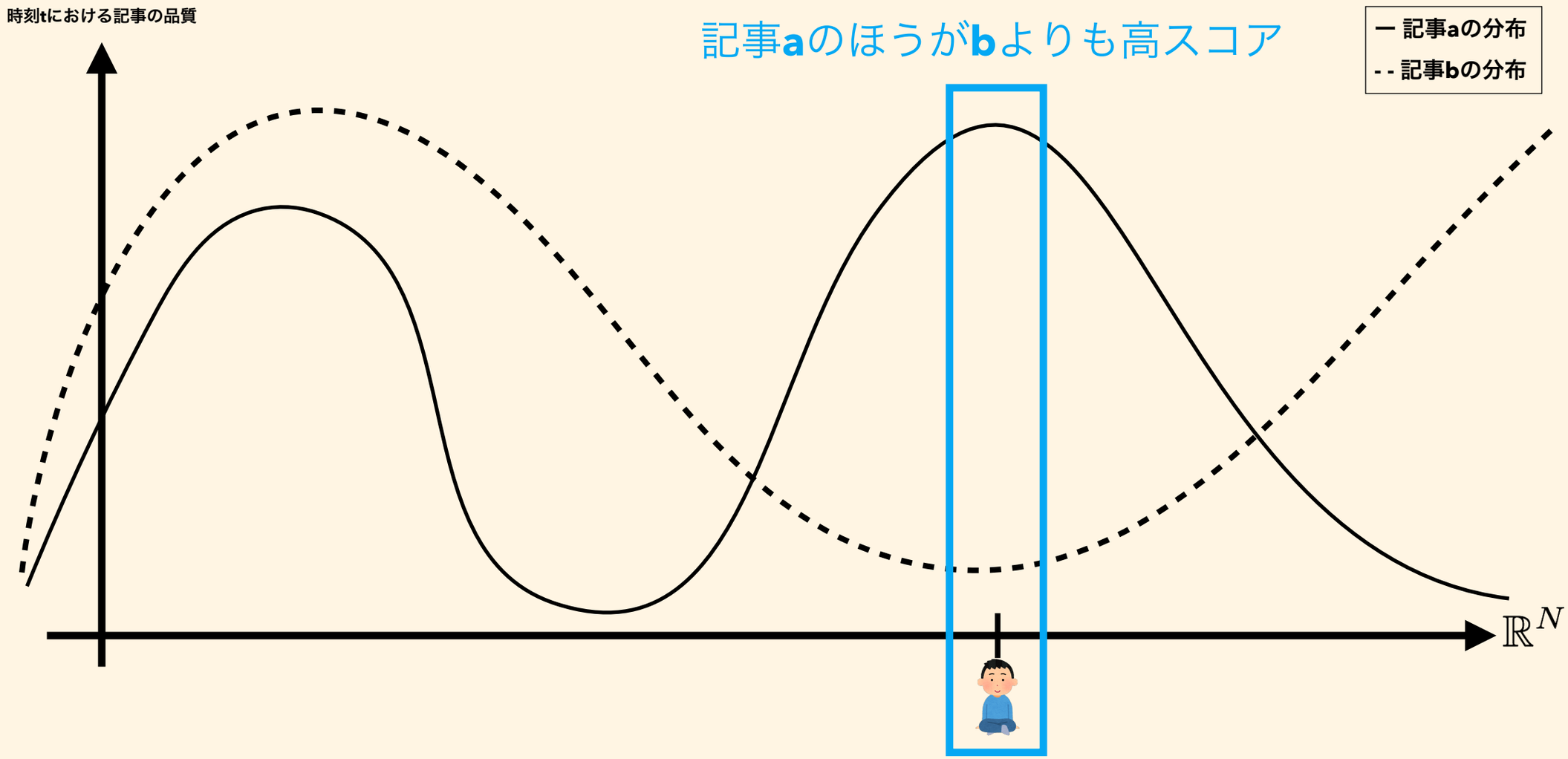
✔ その密度をスコアとして候補となる記事をソートしてユーザーに表示する



行動モデルから推薦へ

✔ ユーザーからのリクエスト時、そのベクトルの周りでの密度の高さを計算

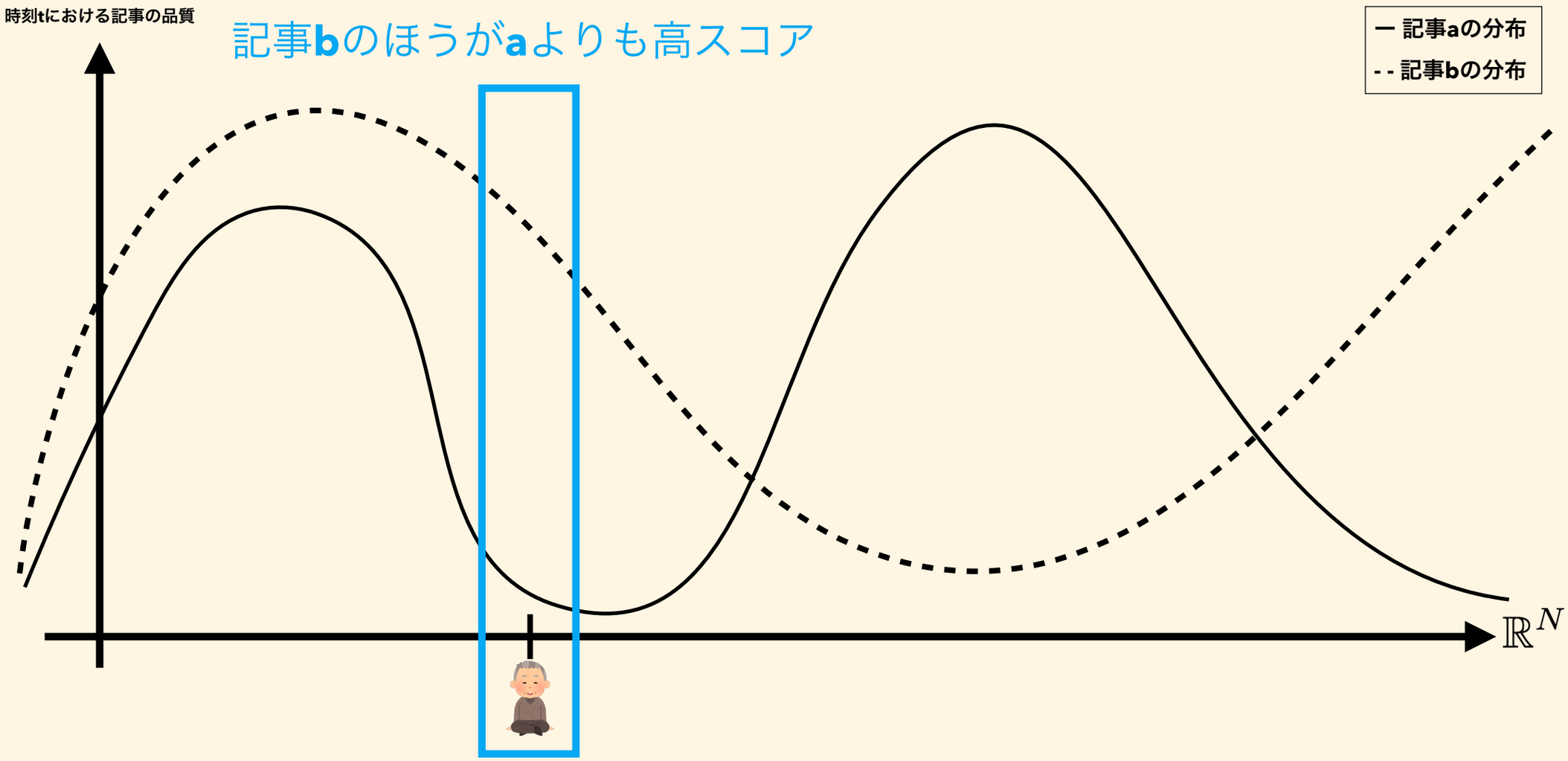
✔ その密度をスコアとして候補となる記事をソートしてユーザーに表示する



行動モデルから推薦へ

✔ ユーザーからのリクエスト時、そのベクトルの周りでの密度の高さを計算

✔ その密度をスコアとして候補となる記事をソートしてユーザーに表示する

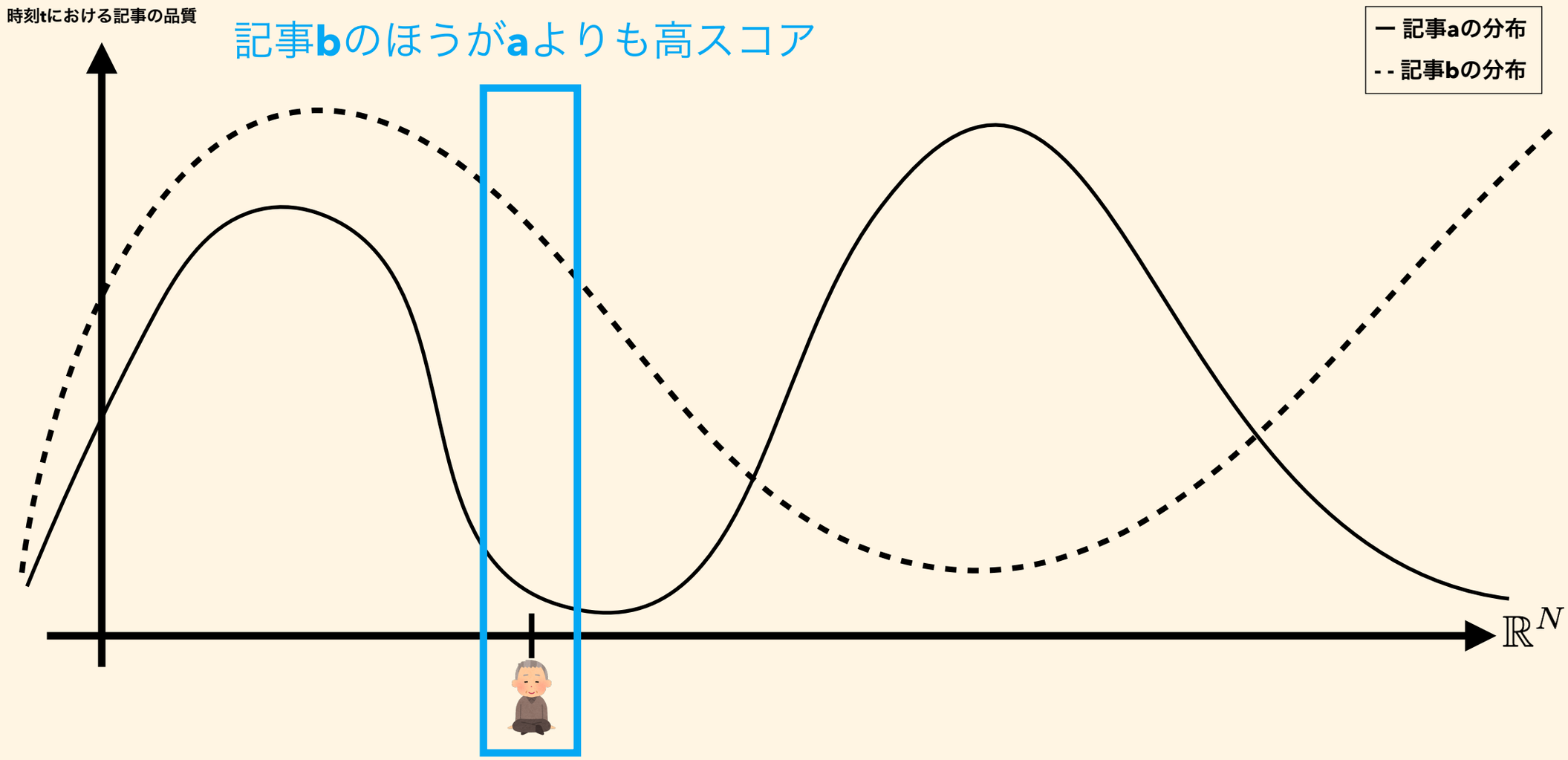


行動モデルから推薦へ

計算量大...

✔ ユーザーからのリクエスト時、そのベクトルの周りでの密度の高さを計算

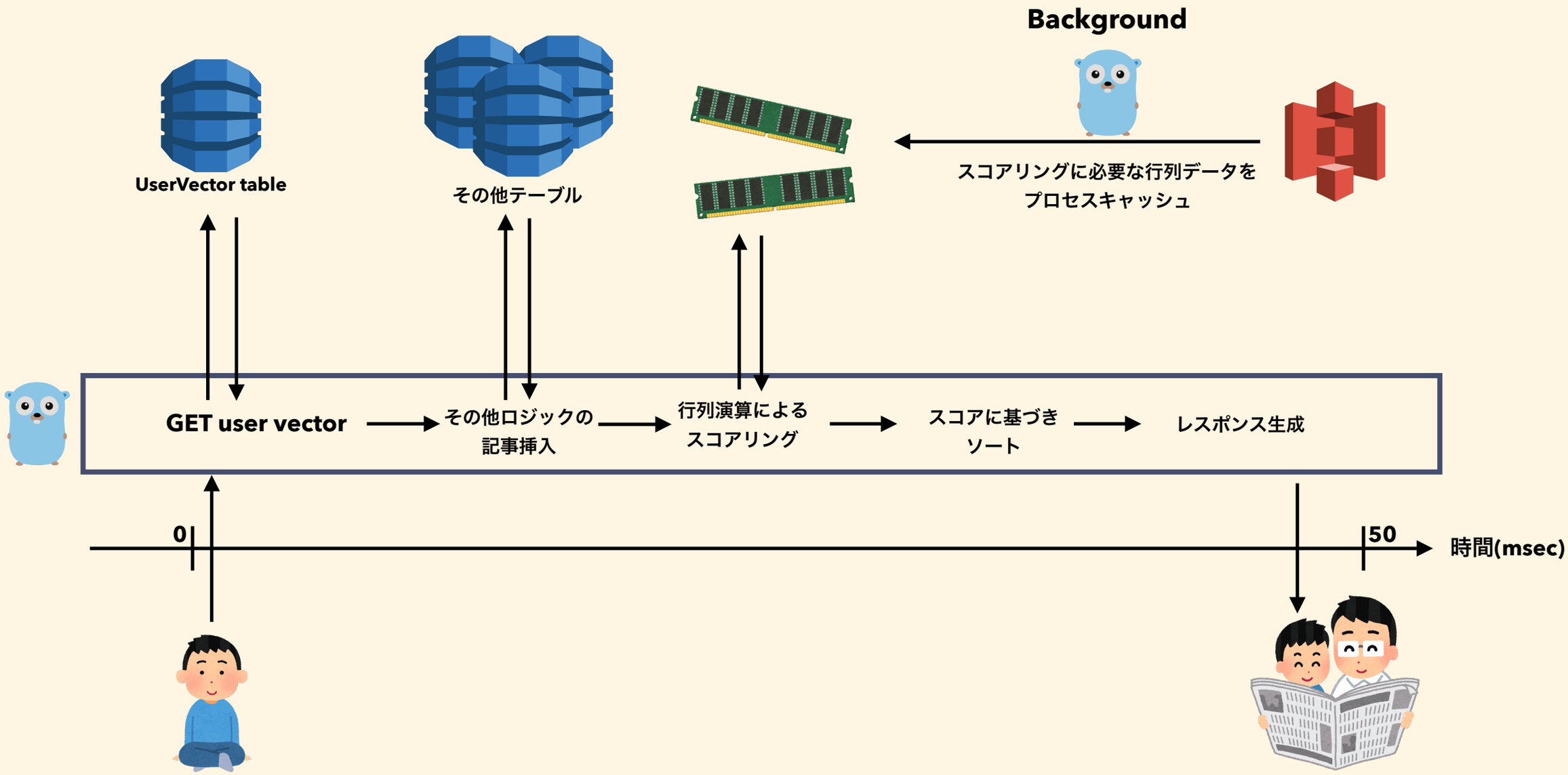
✔ その密度をスコアとして候補となる記事をソートしてユーザーに表示する



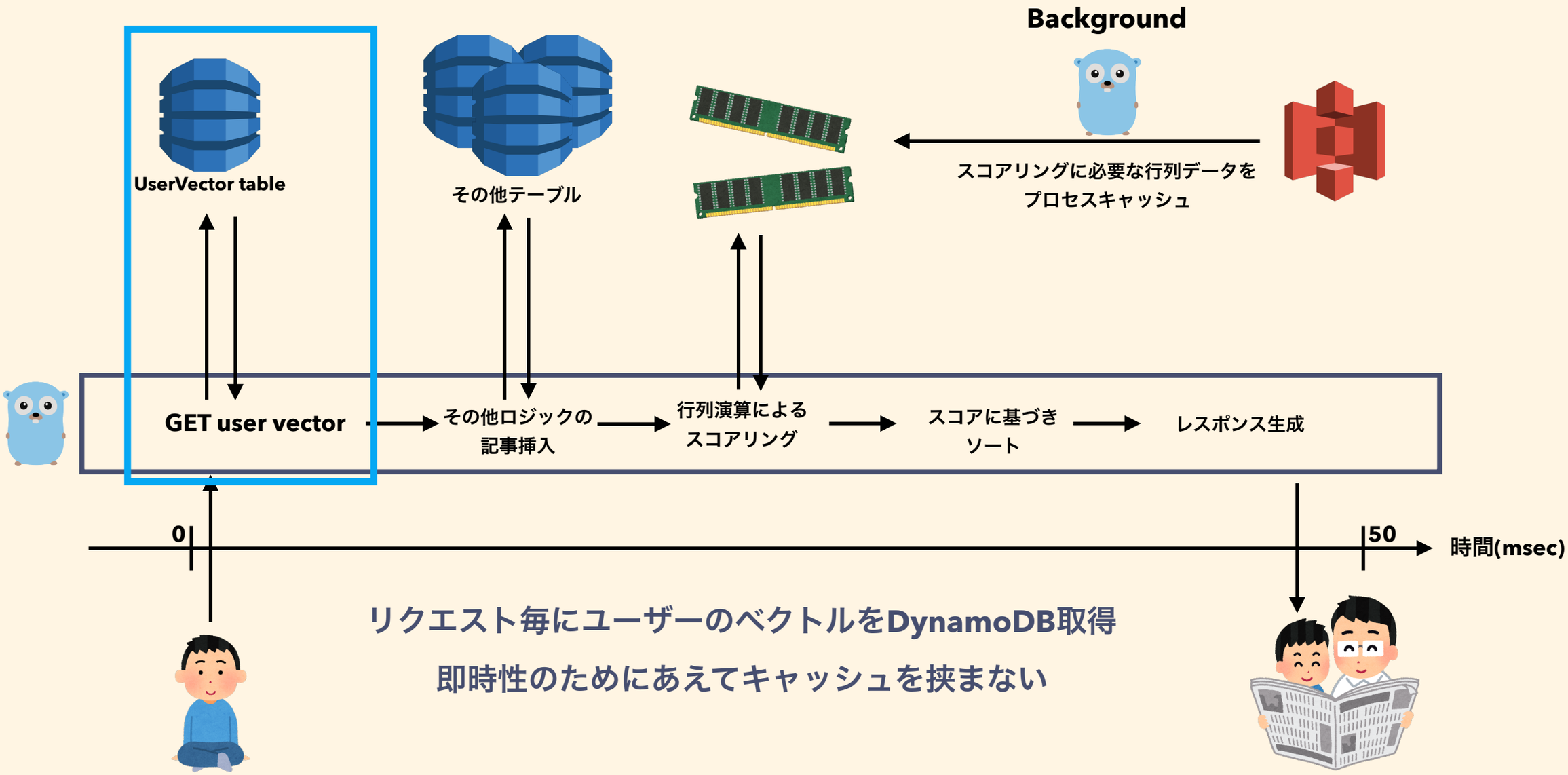
50msec or die.†

† https://tenshoku.mynavi.jp/it-engineer/knowhow/naoya_sushi/05

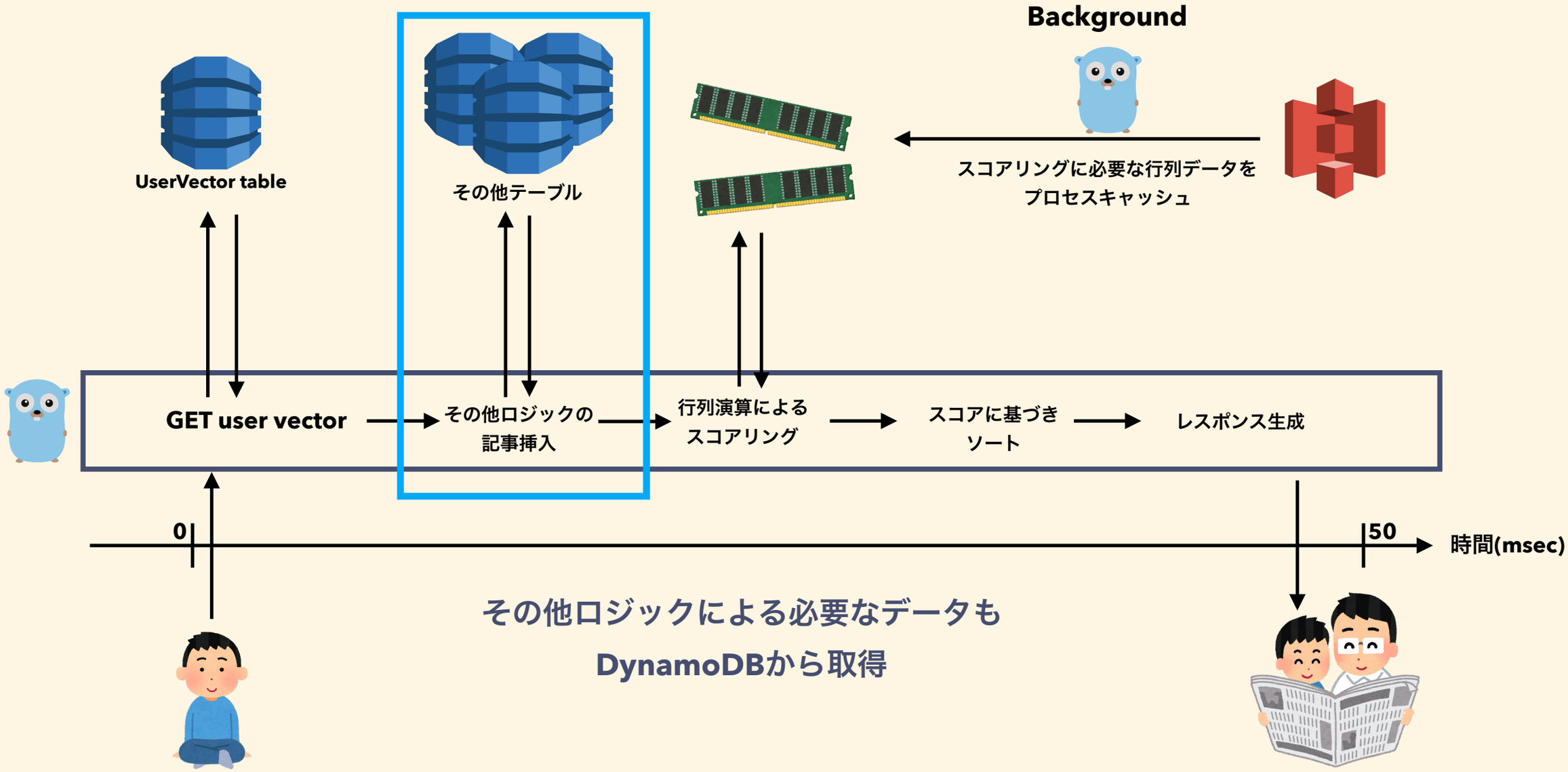
推薦リスト生成 in 50 msec



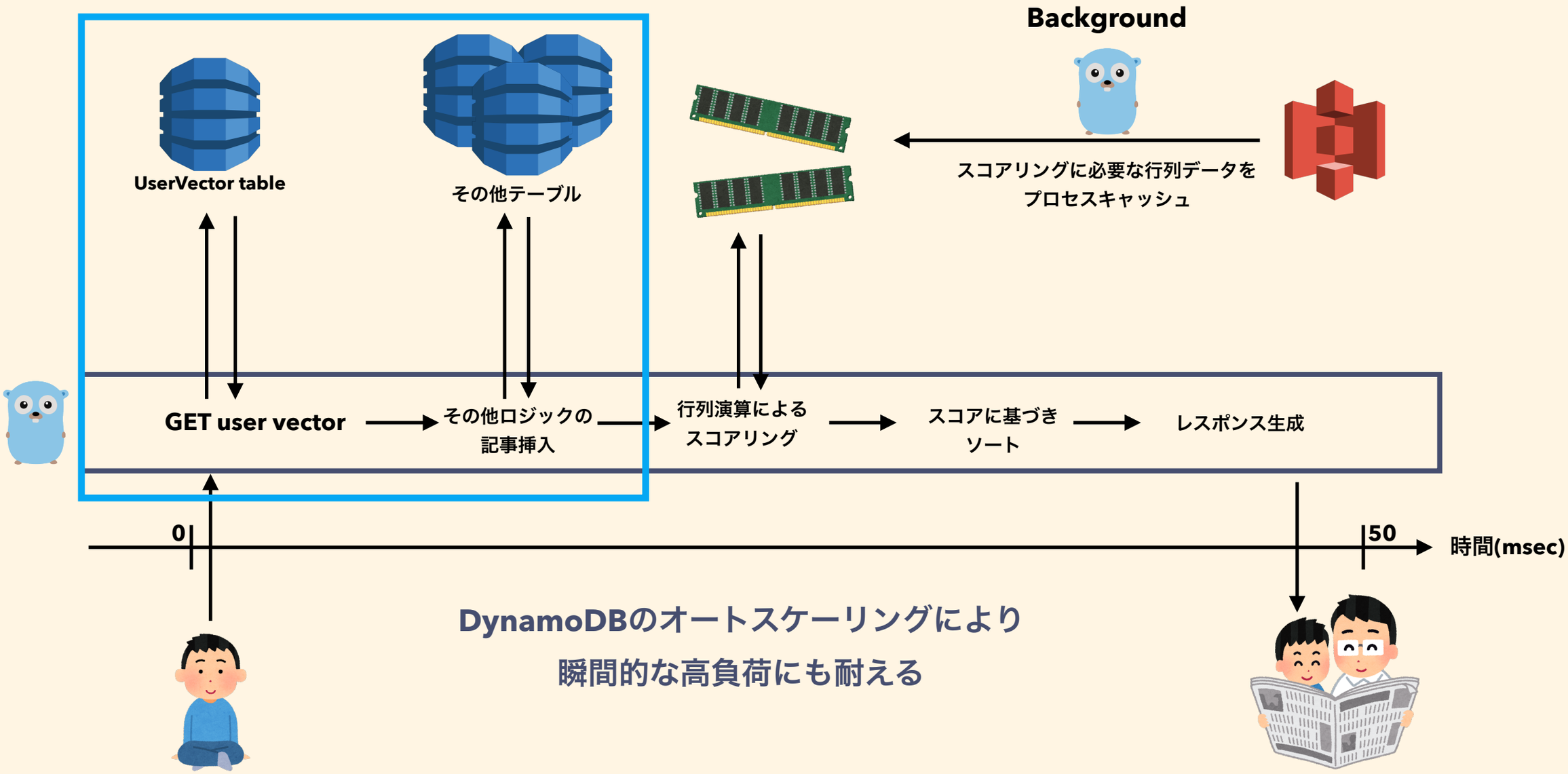
推薦リスト生成 in 50 msec



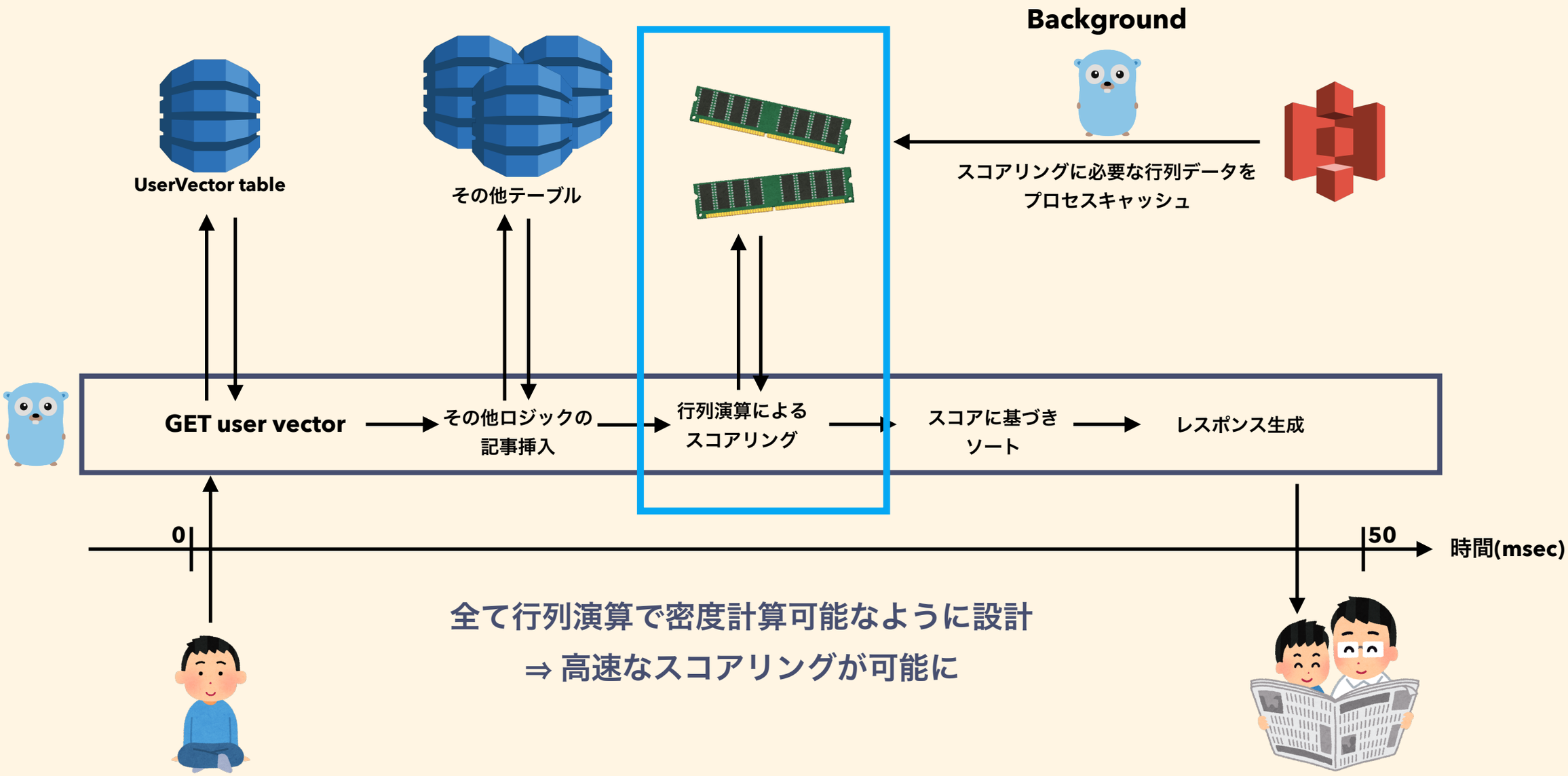
推薦リスト生成 in 50 msec



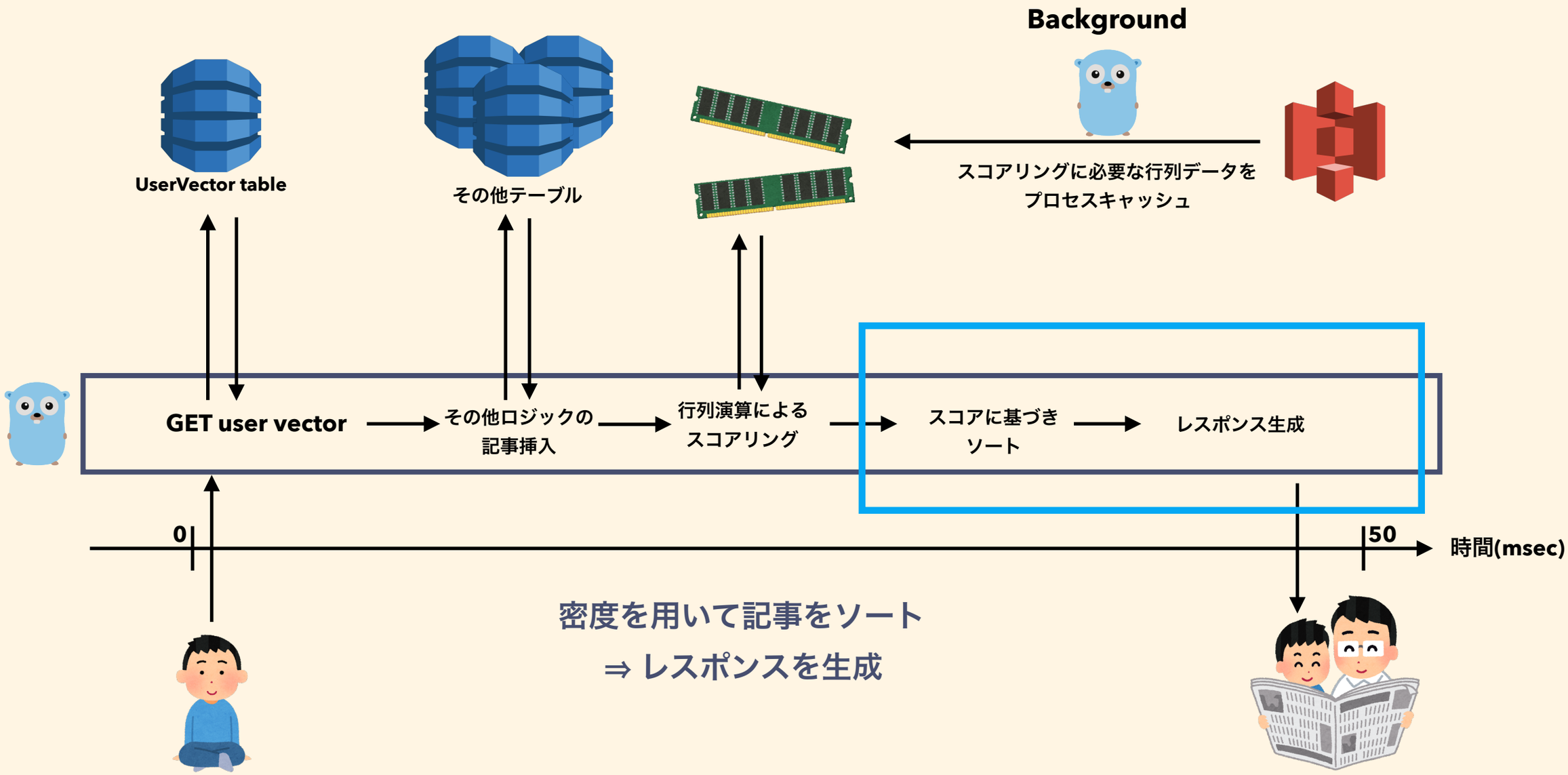
推薦リスト生成 in 50 msec



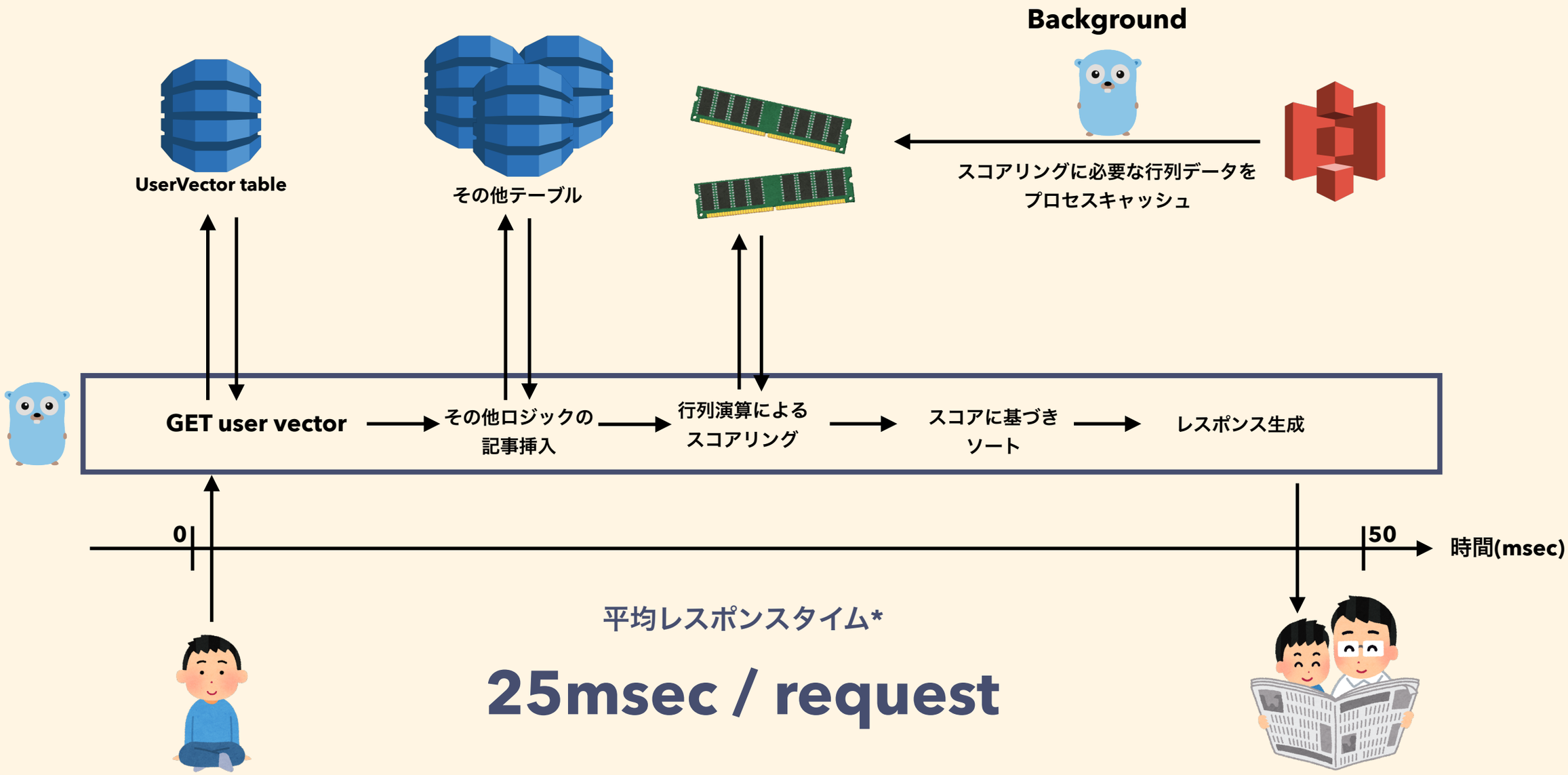
推薦リスト生成 in 50 msec



推薦リスト生成 in 50 msec

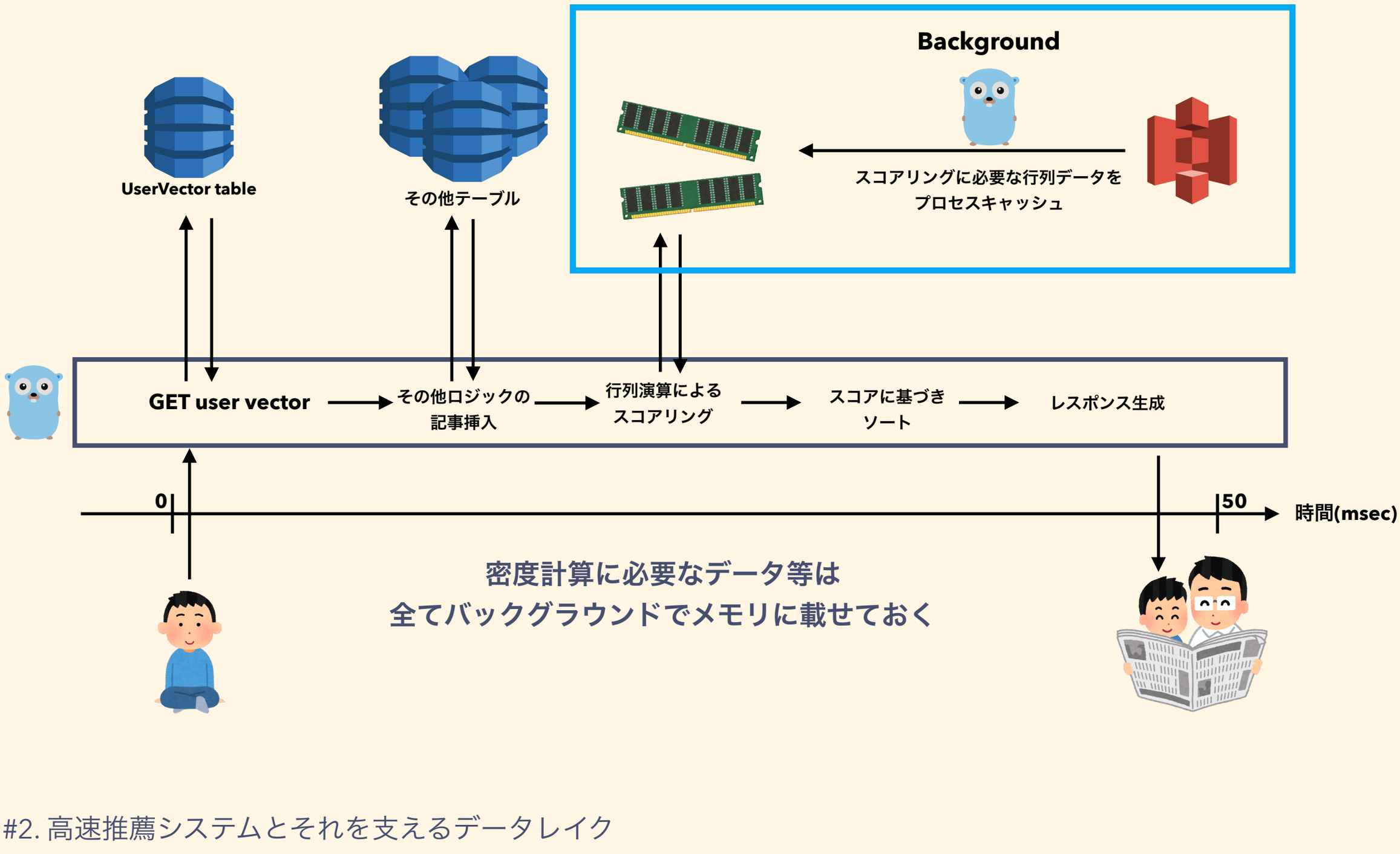


推薦リスト生成 in 50 msec

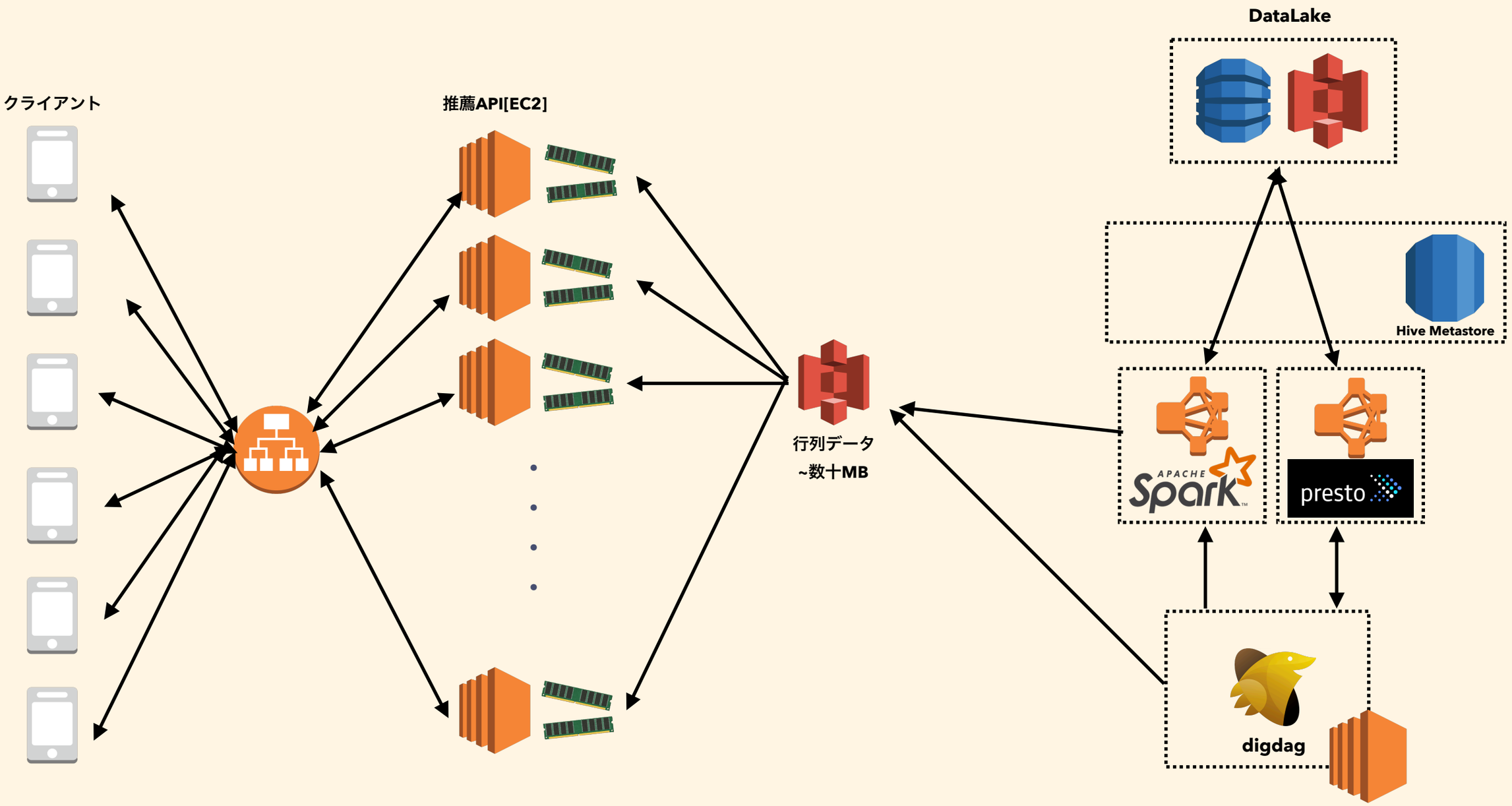


*Application Load Balancer のメトリクスの一つ `TargetResponseTime` の値

推薦リスト生成 in 50 msec

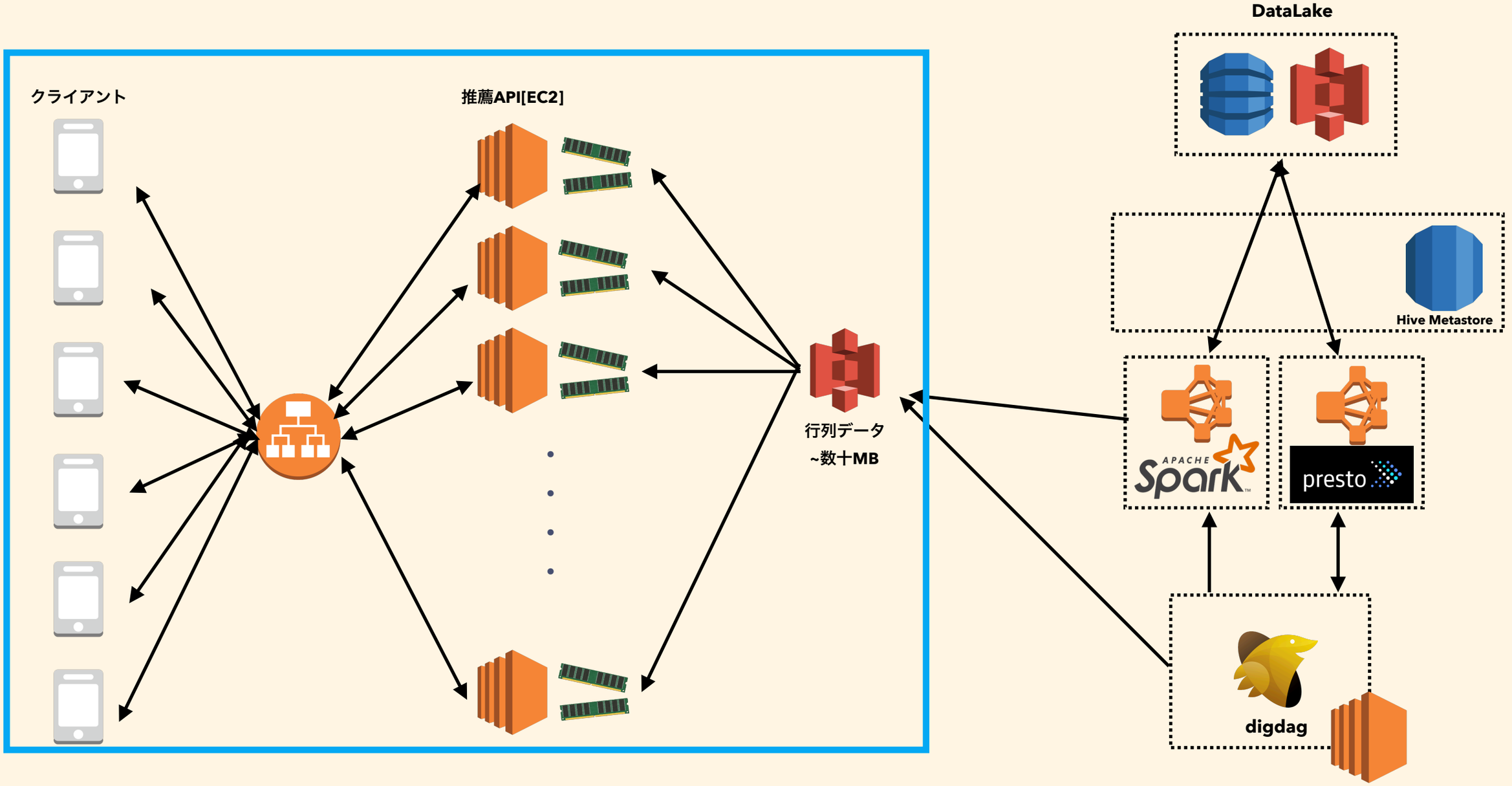


推薦APIの裏側で


クライアント
推薦API[EC2]
DataLake
Hive Metastore
行列データ
~数十MB
APACHE Spark
presto
digdag

推薦APIの裏側で

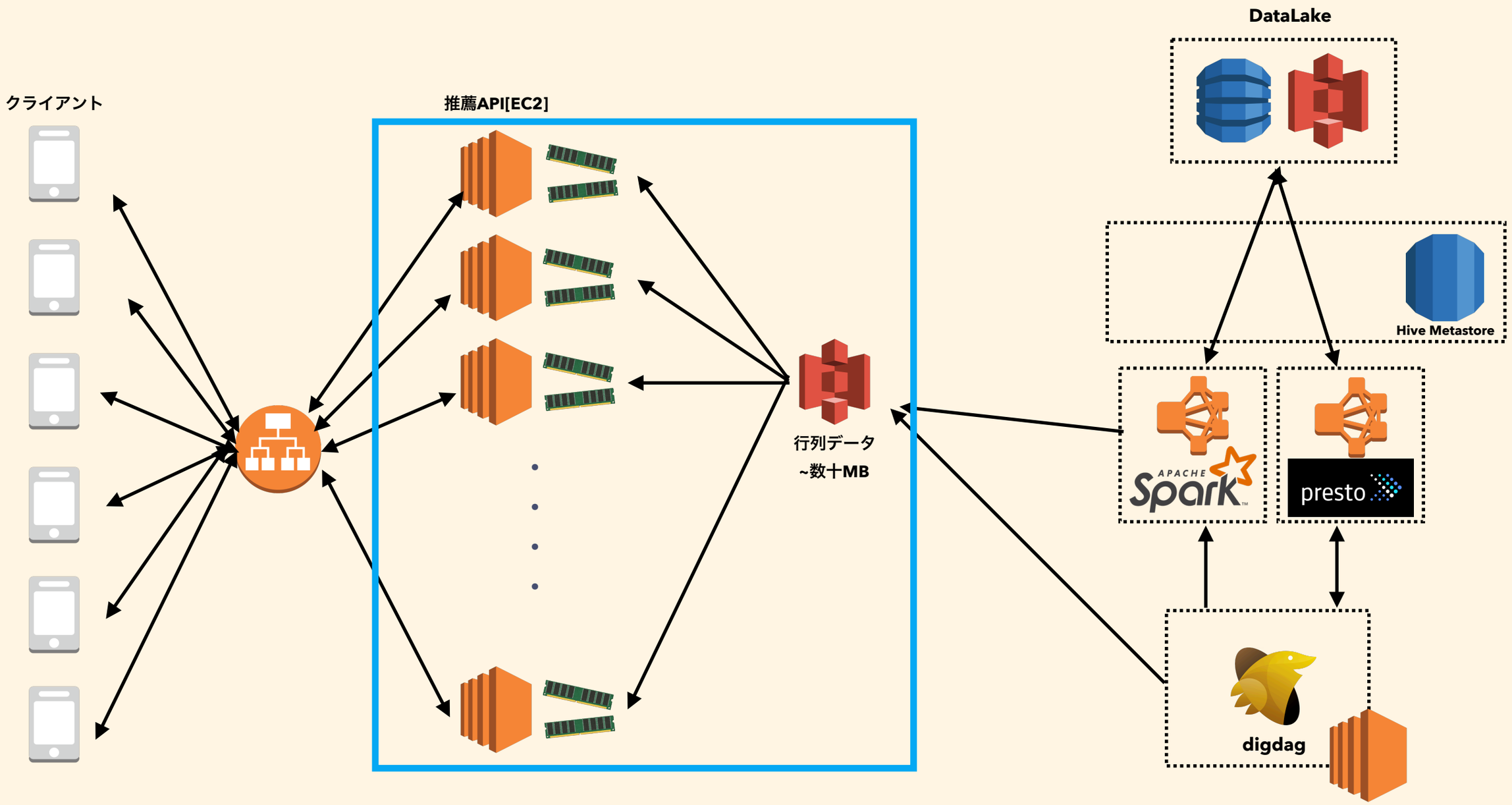
✔ 先程の世界線



#2. 高速推薦システムとそれを支えるデータレイク

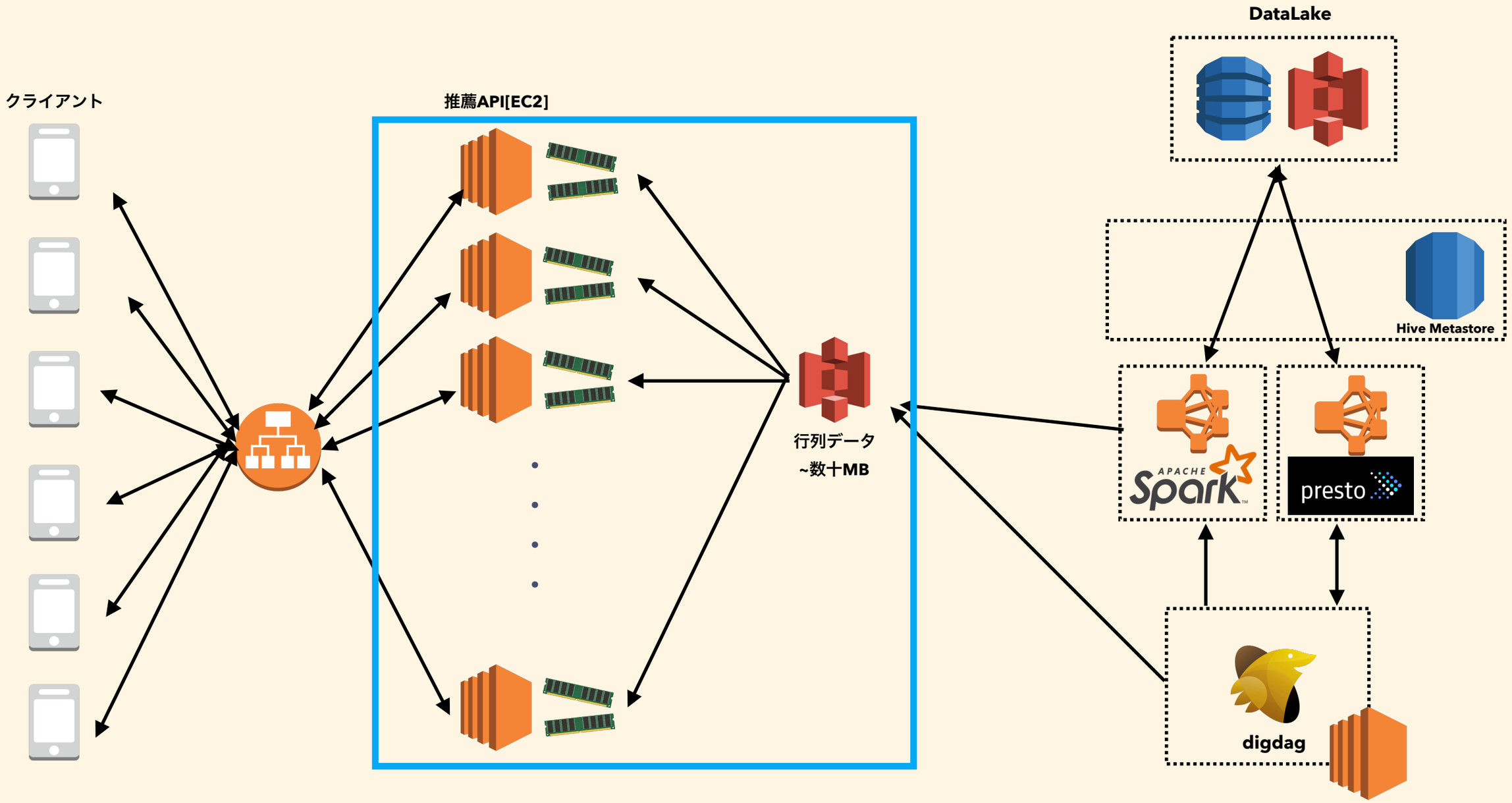
推薦APIの裏側で

✔ s3から各インスタンスに行列データ(数十MB)がばらまかれる



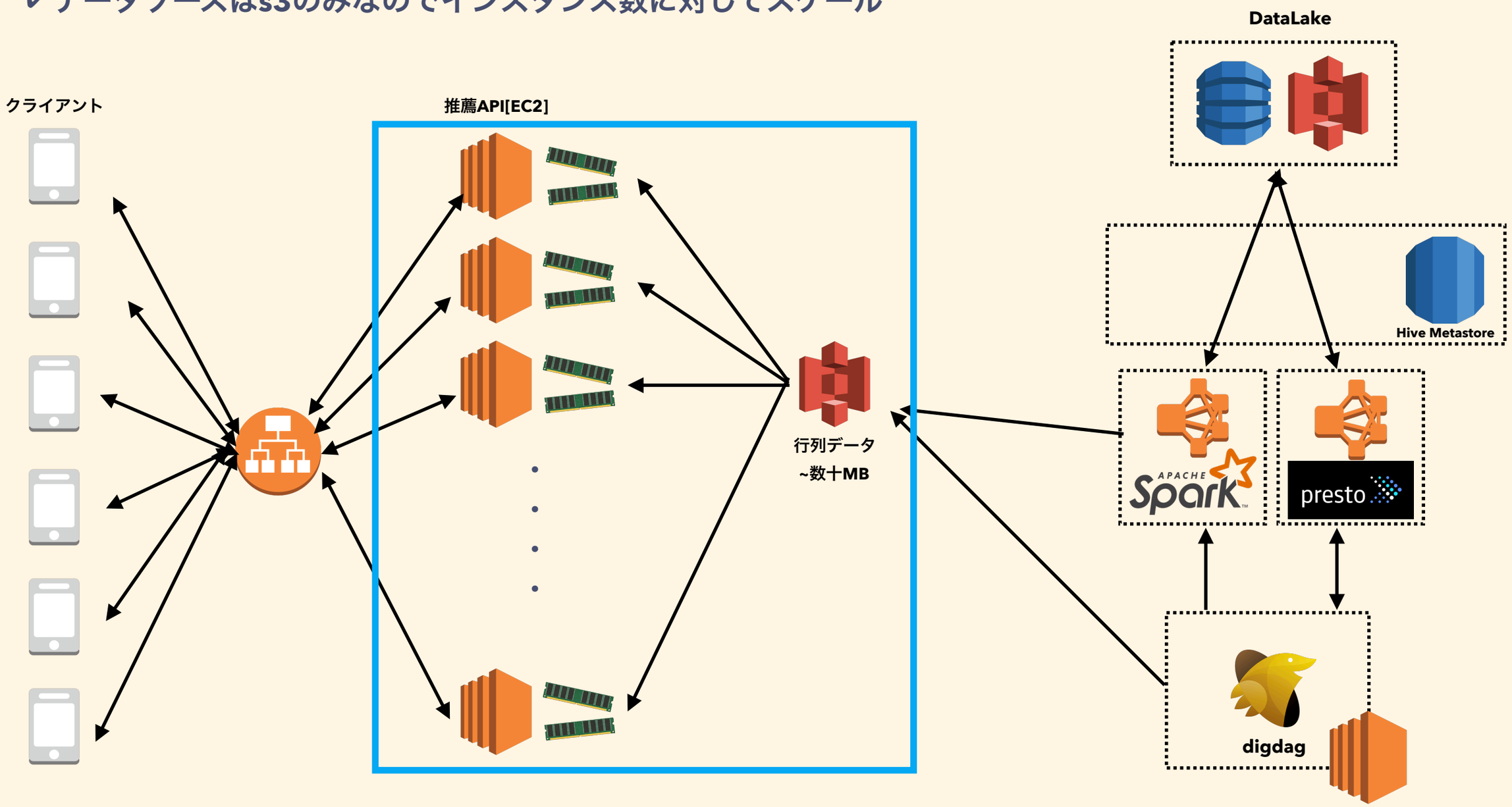
推薦APIの裏側で

✔ ユーザーからのリクエストとは非同期でメモリーにロード



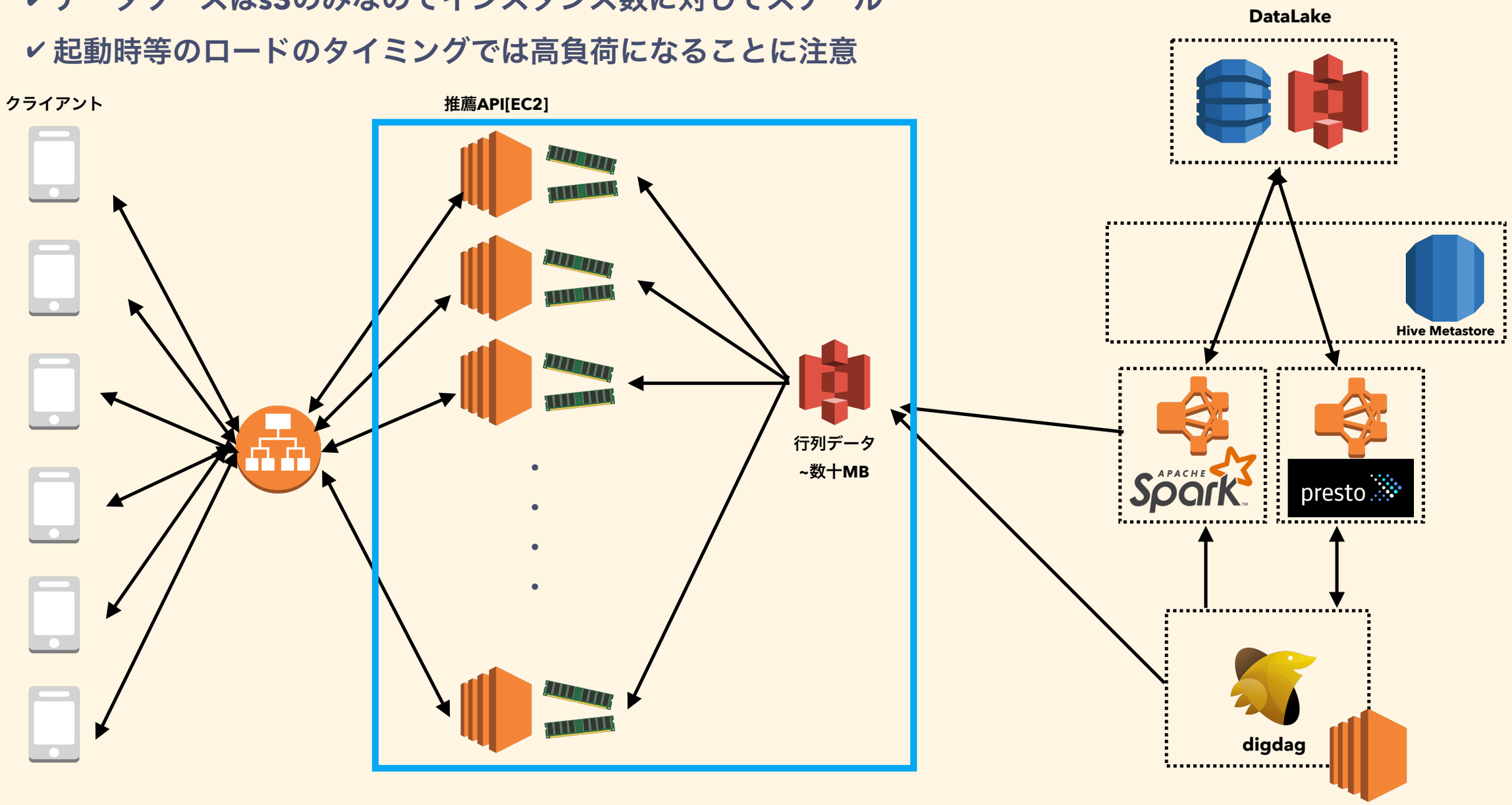
推薦APIの裏側で

- ✔ ユーザーからのリクエストとは非同期でメモリーにロード
- ✔ データソースはs3のみなのでインスタンス数に対してスケール



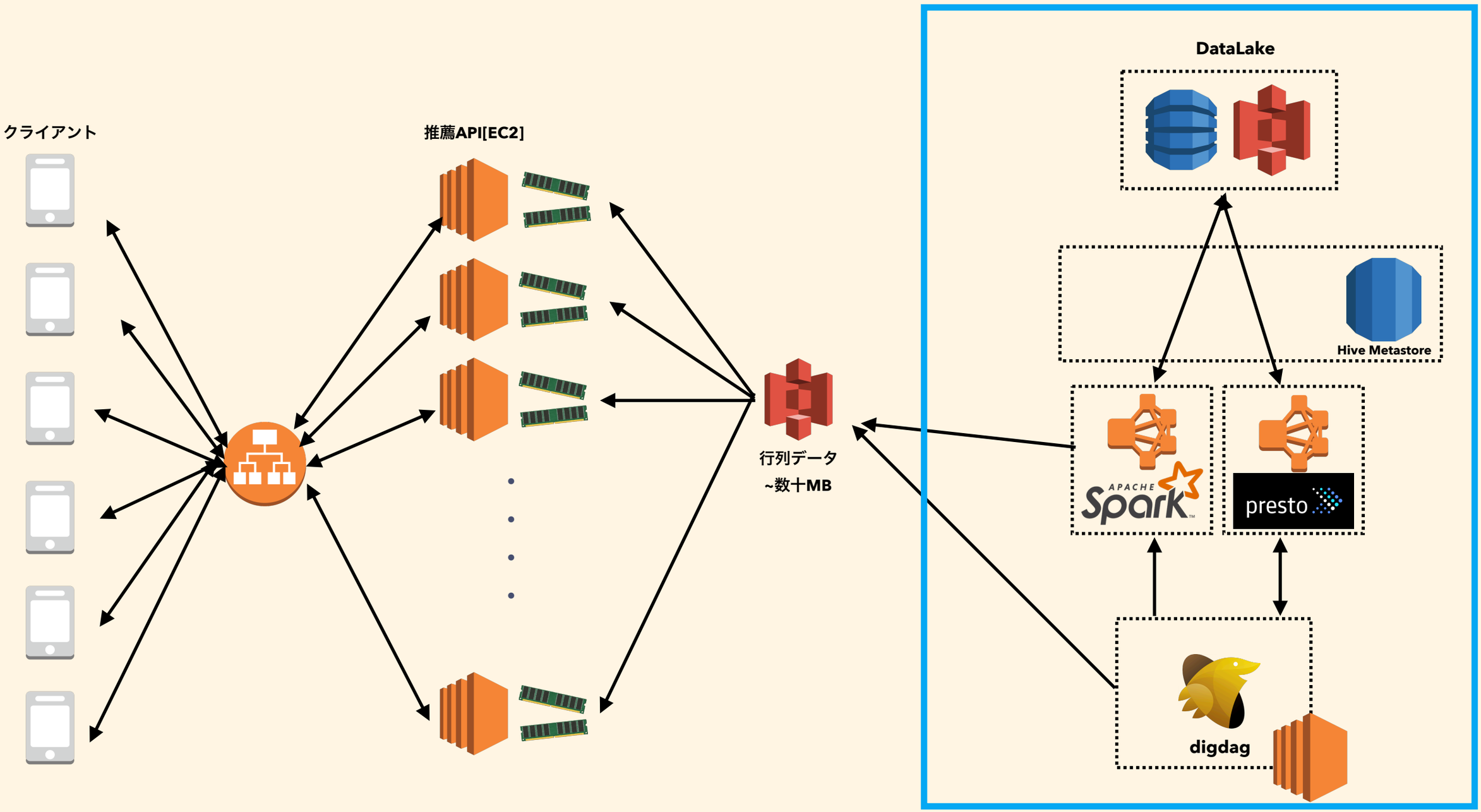
推薦APIの裏側で

- ✔ ユーザーからのリクエストとは非同期でメモリーにロード
- ✔ データソースはs3のみなのでインスタンス数に対してスケール
- ✔ 起動時等のロードのタイミングでは高負荷になることに注意



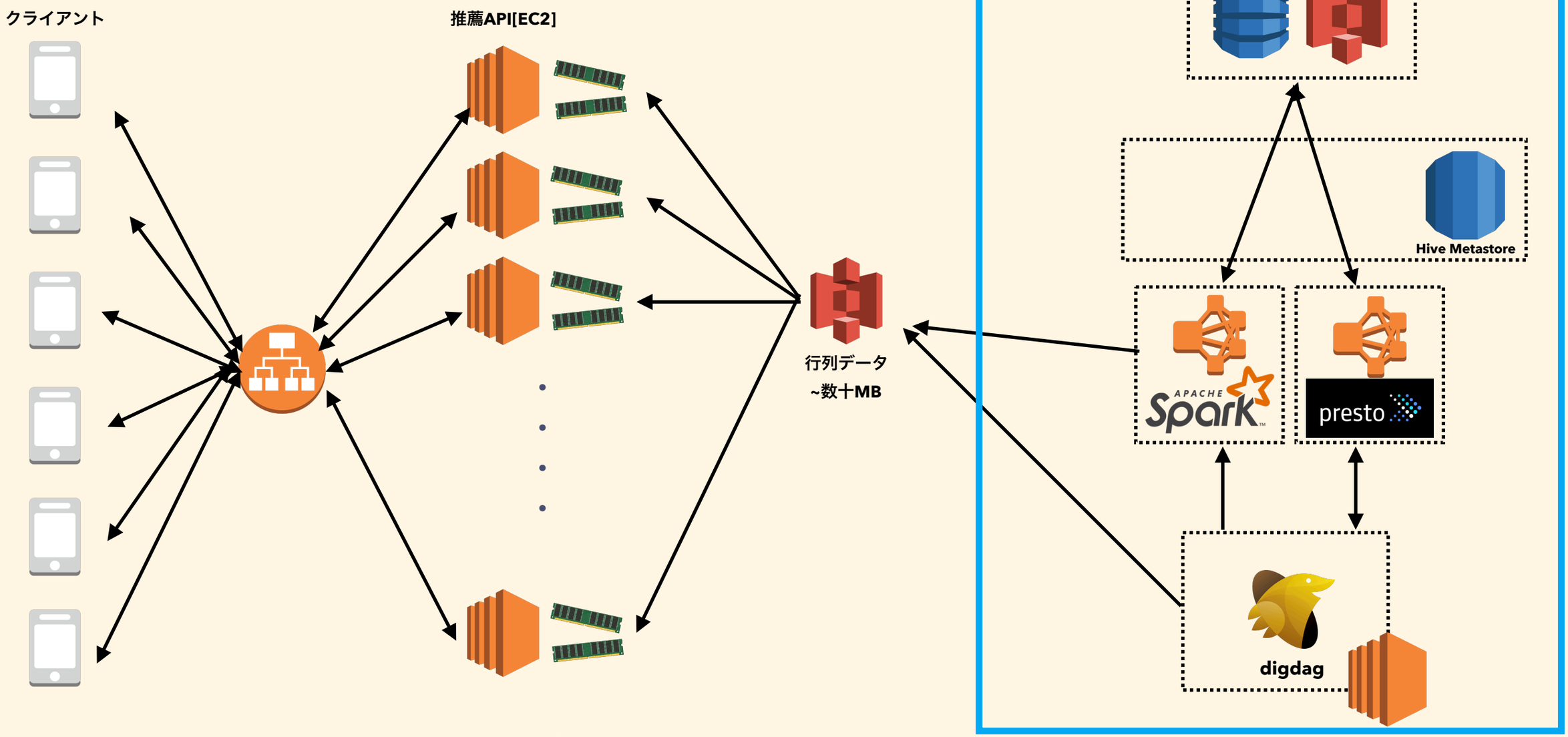
推薦APIの裏側で

✔ 行列データ生成の舞台裏

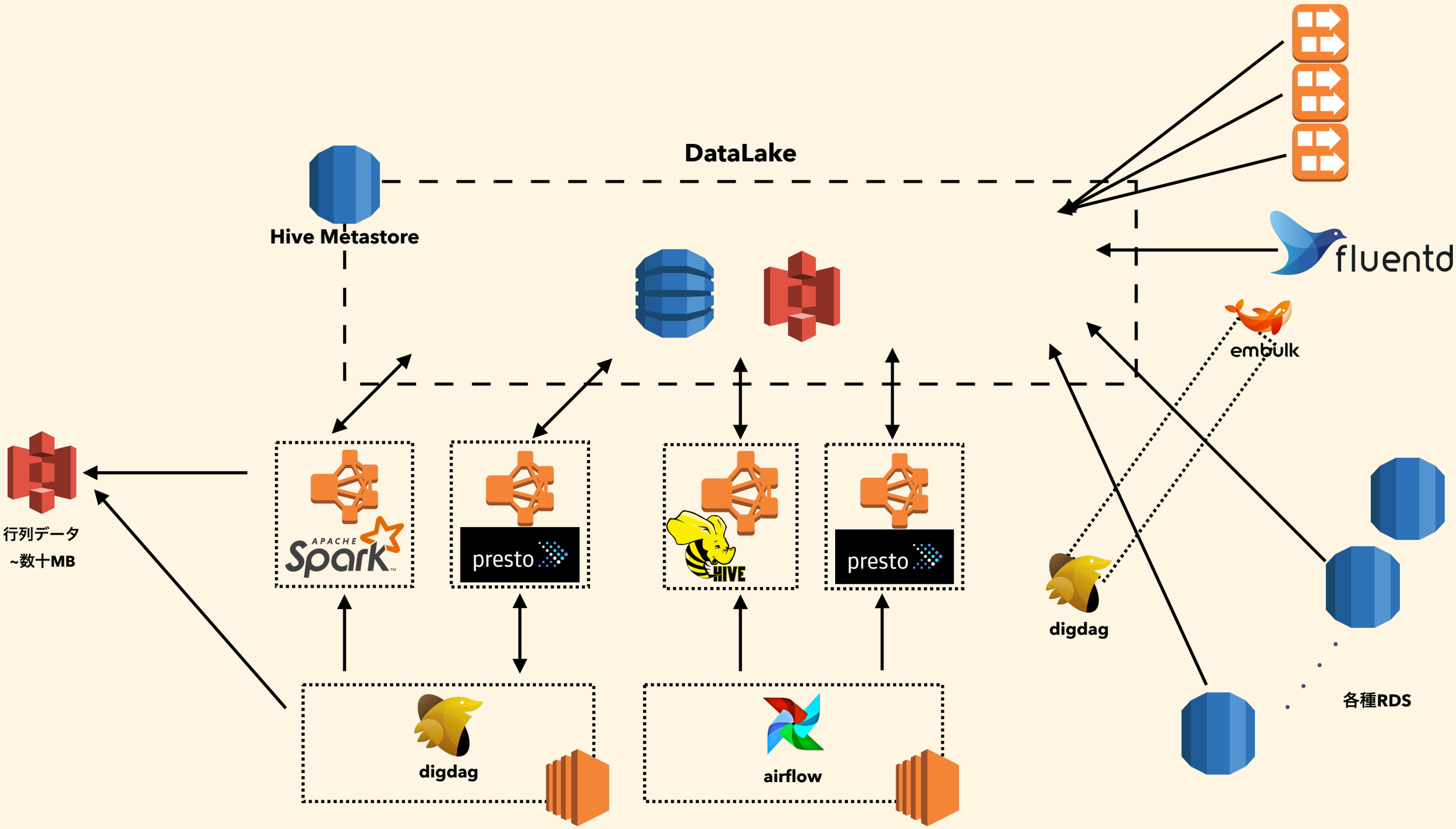


推薦APIの裏側で

- ✔ 行列データ生成の舞台裏
- ✔ EMR(Elastic MapReduce)をフル活用



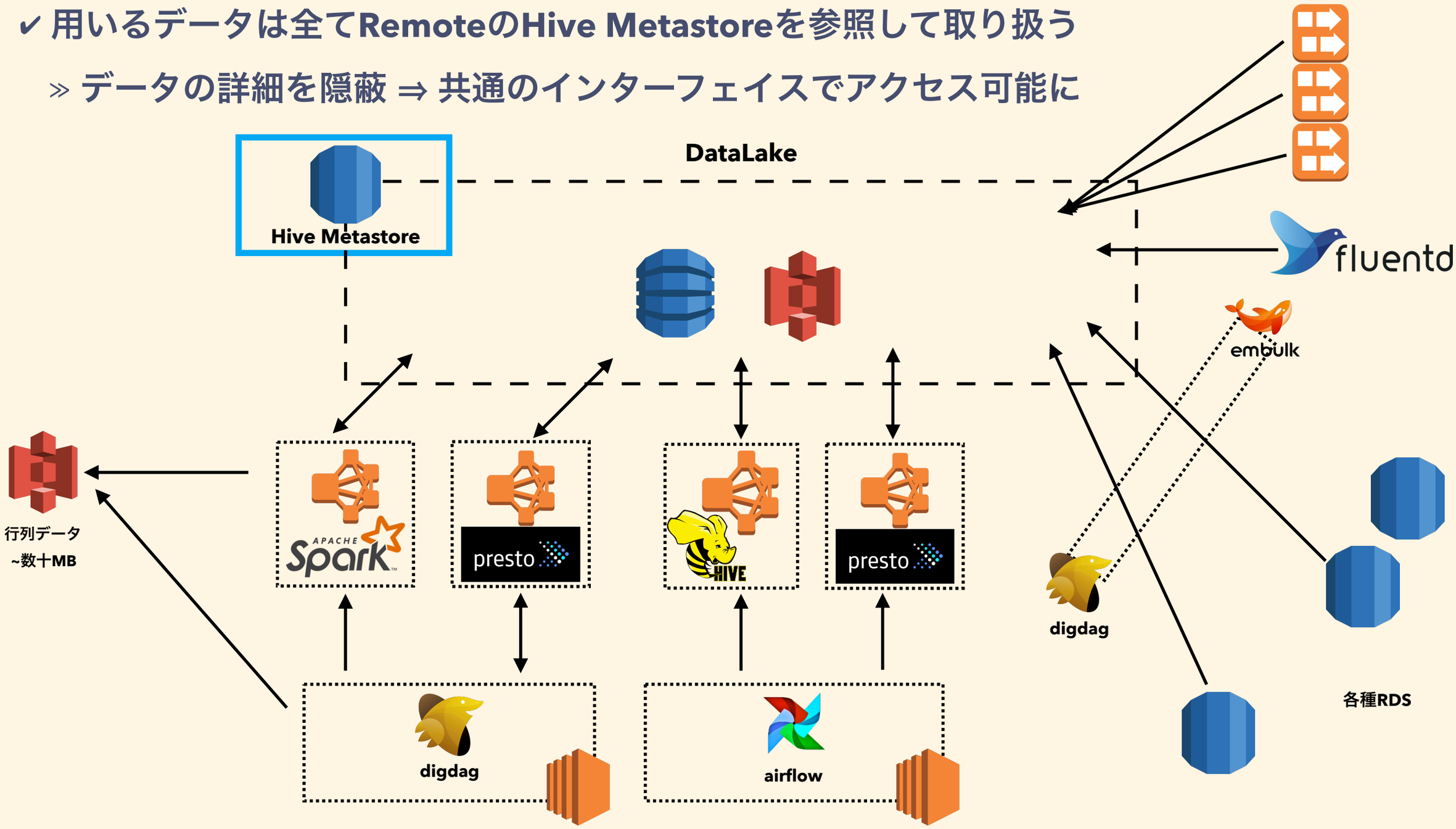
行列データ生成の舞台裏



行列データ生成の舞台裏

✔ 用いるデータは全てRemoteのHive Metastoreを参照して取り扱う

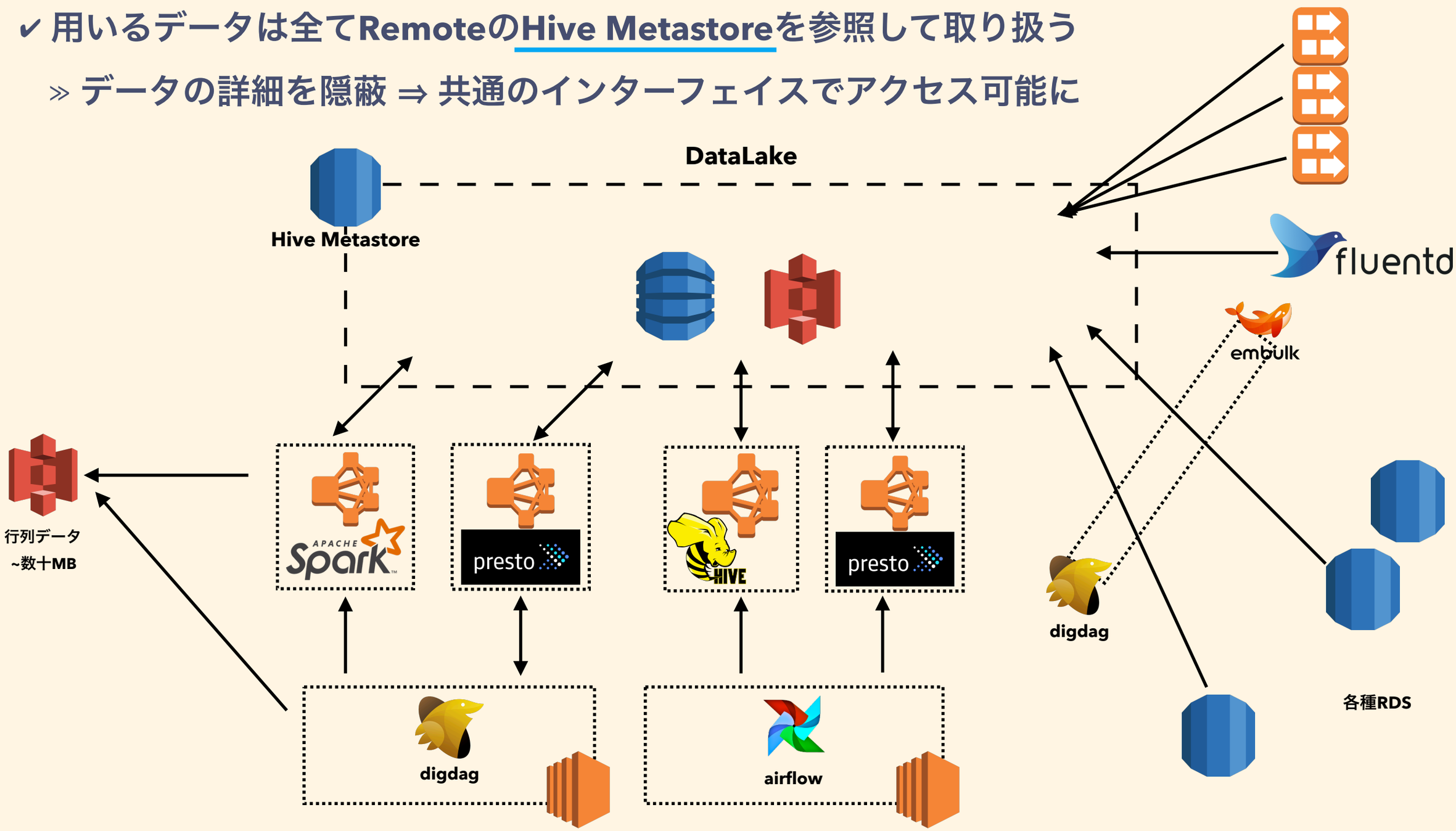
» データの詳細を隠蔽 ⇒ 共通のインターフェイスでアクセス可能に



行列データ生成の舞台裏

✔ 用いるデータは全てRemoteのHive Metastoreを参照して取り扱う

» データの詳細を隠蔽 ⇒ 共通のインターフェイスでアクセス可能に



行列データ生成の舞台裏

✔ (Remote) Hive Metastore

» 「テーブル定義とHDFS*上のデータの対応に関するメタデータ」を管理するもの

» クライアント(Spark/Hive/Presto)は実体にアクセスする前にMetastoreから情報を取得

» メタデータそのものはAmazon RDS for MySQL内に保持

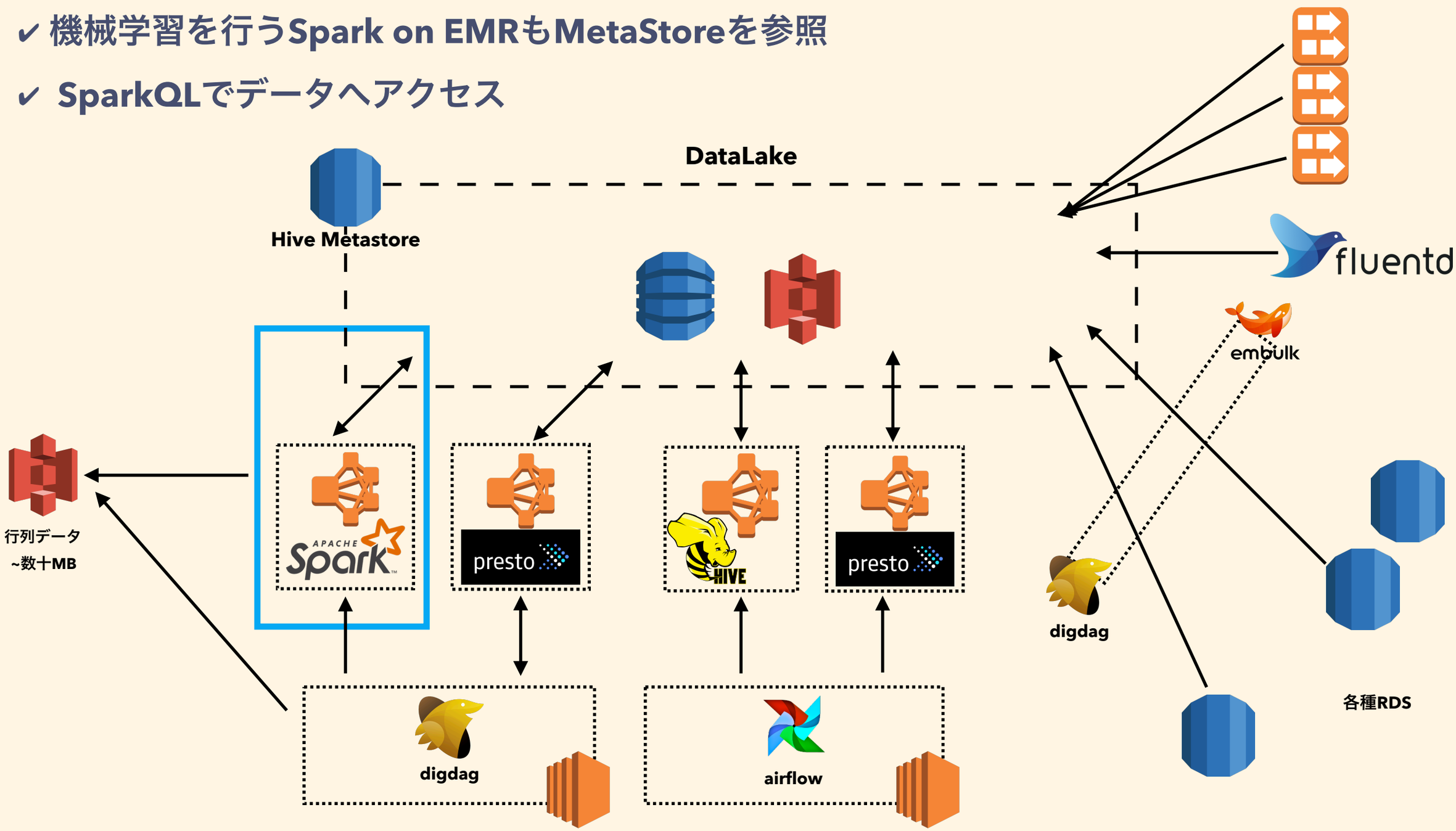
» テーブル定義例 👉

```
CREATE EXTERNAL TABLE `users`(
  `id` bigint,
  `enabled` boolean,
  `admin_enabled` boolean,
  `created_at` string,
  `updated_at` string
) STORED AS PARQUET
LOCATION 's3://hogefuga-log/hive/user.db/users'
TBLPROPERTIES (‘parquet.compress'='SNAPPY');
```

*HDFS = Hadoop Distributed File System. s3は実際にはFile Systemではないが、Hadoop S3 filesystem によりあたかもs3がHDFSであるようにやり取りが出来る

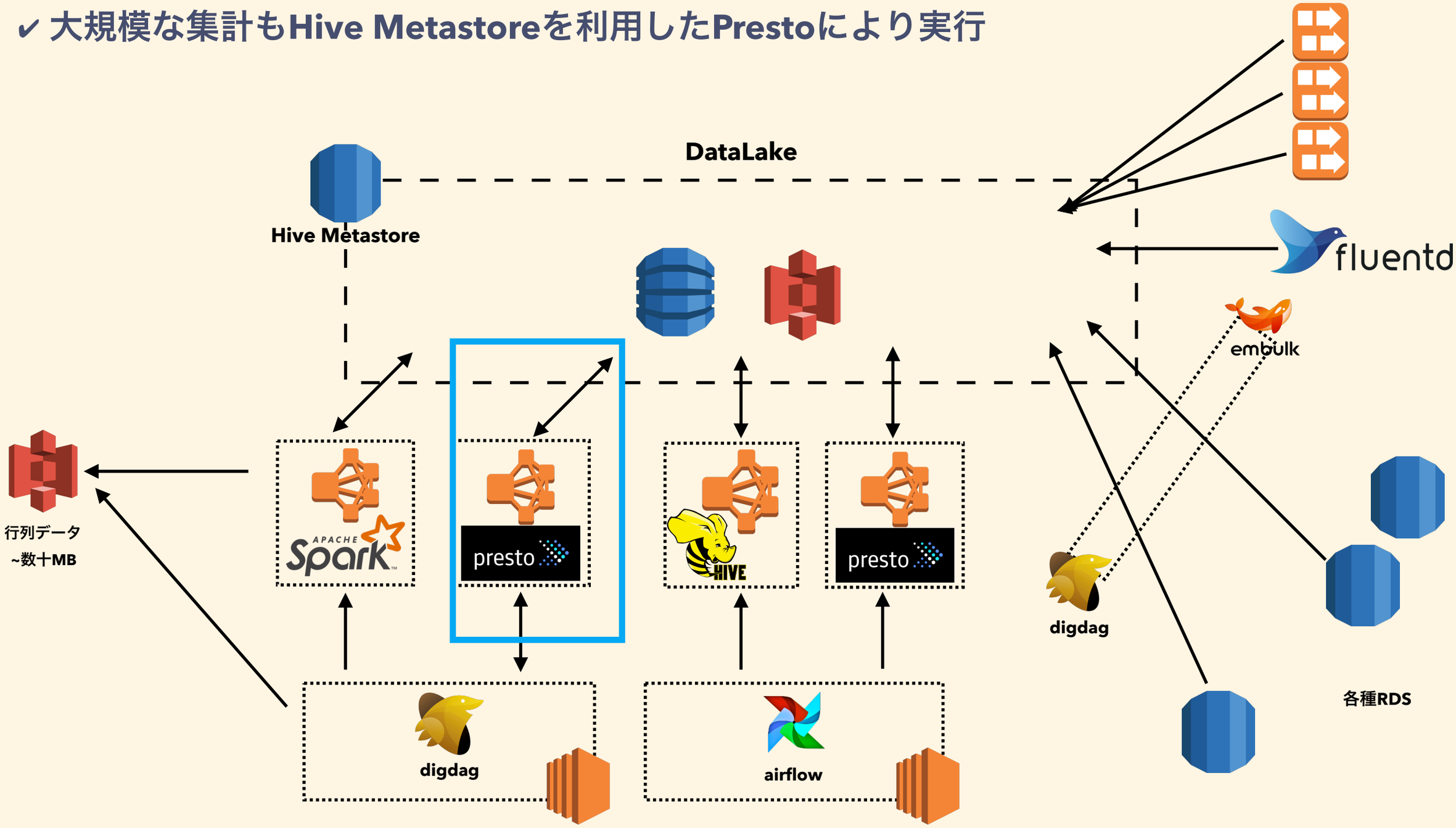
行列データ生成の舞台裏

- ✔ 機械学習を行うSpark on EMRもMetaStoreを参照
- ✔ SparkQLでデータへアクセス



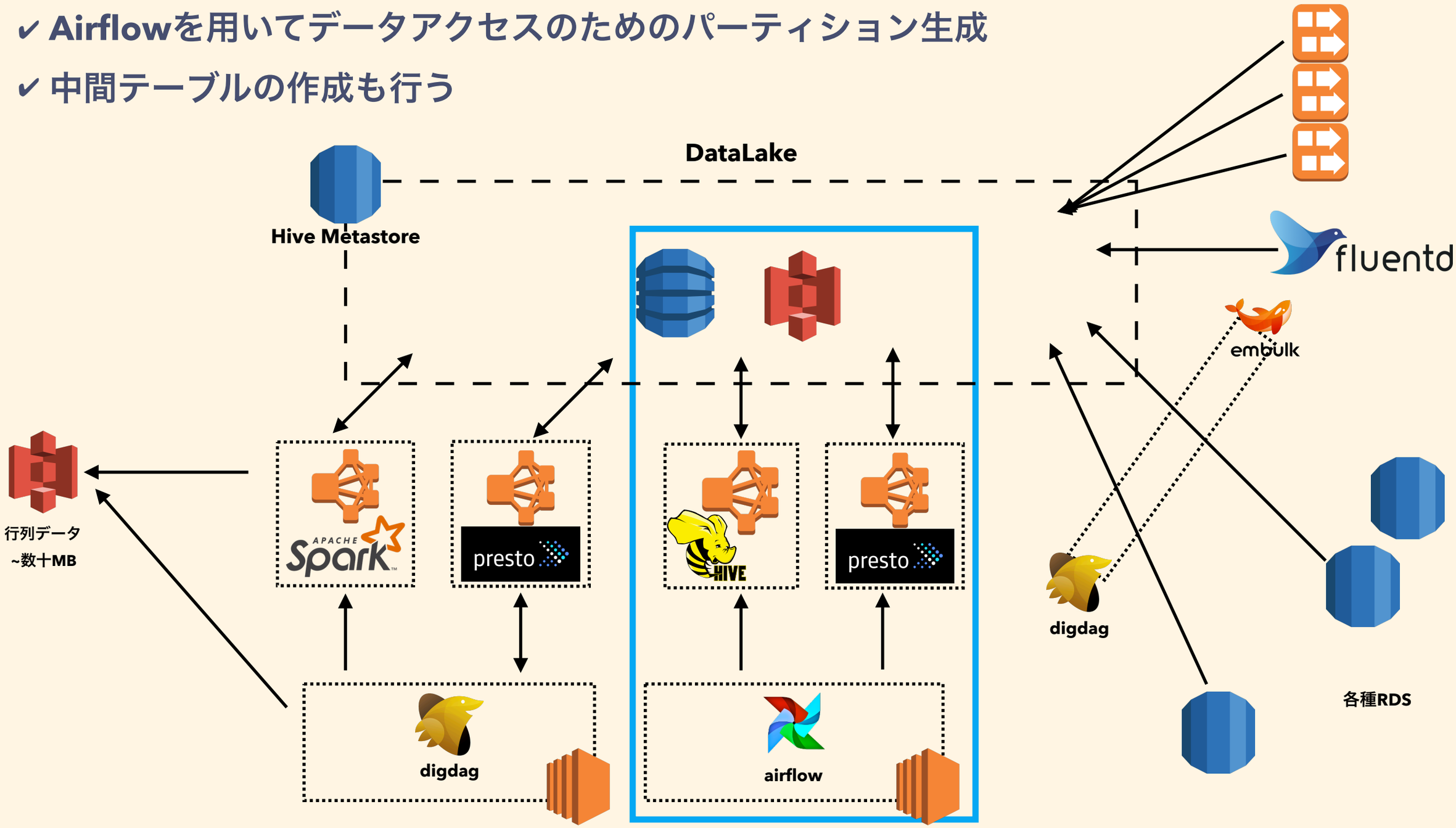
行列データ生成の舞台裏

✔ 大規模な集計もHive Metastoreを利用したPrestoにより実行



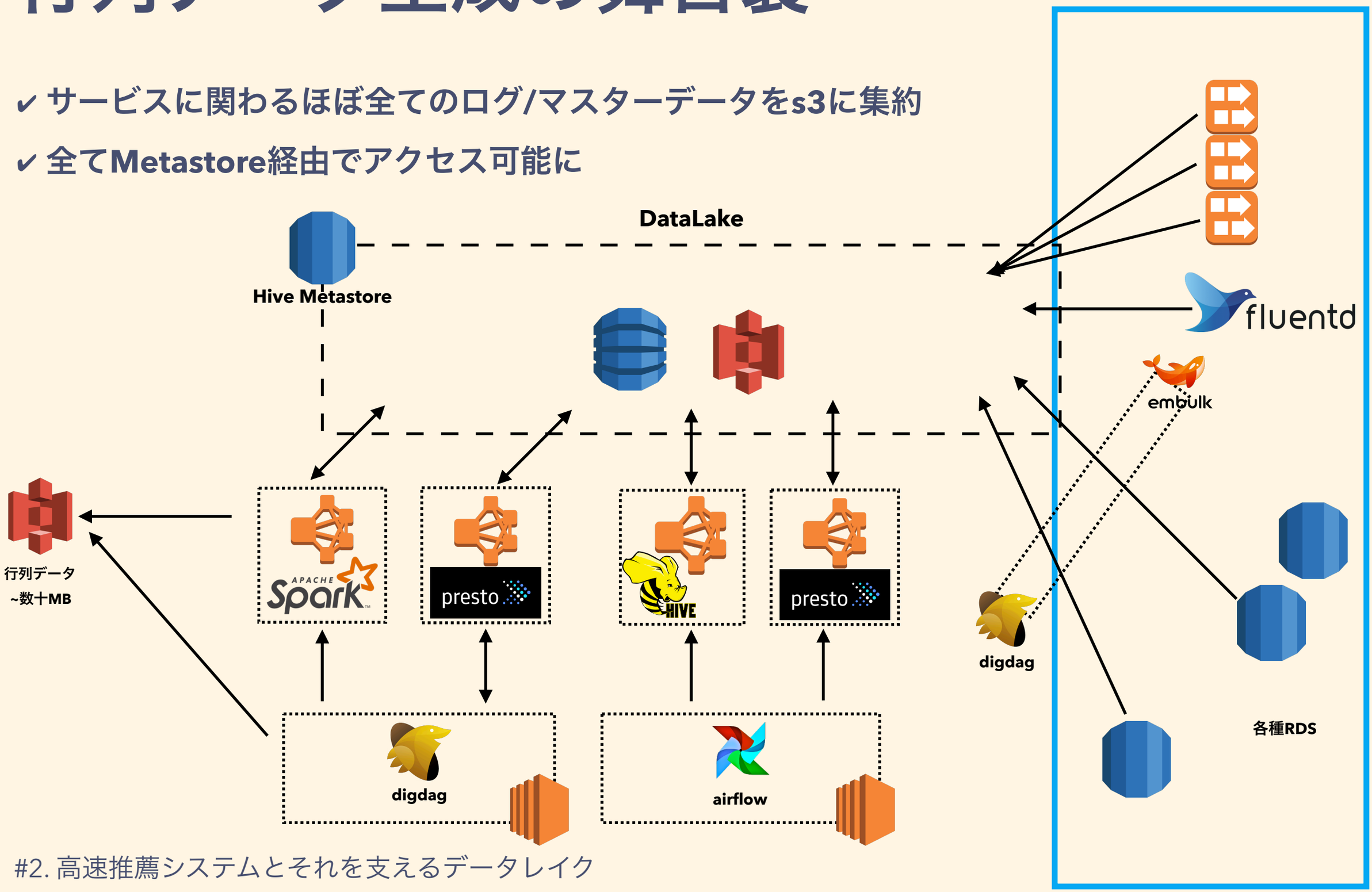
行列データ生成の舞台裏

- ✔ Airflowを用いてデータアクセスのためのパーティション生成
- ✔ 中間テーブルの作成も行う



行列データ生成の舞台裏

- ✔ サービスに関わるほぼ全てのログ/マスターデータをs3に集約
- ✔ 全てMetastore経由でアクセス可能に



S3 + Hive Metastore = Awesome!

✔ データの詳細を隠蔽してアクセスが可能

- スキーマを予め定めておく ⇒ HQLを管理
- 変更がある場合もデータの実体に影響を与えずスキーマ変更可能 –> ~~Drop Tableやり放題~~

✔ 各アプリケーションからほぼ共通のインターフェイスでアクセス可能

- SELECT column1, column2 FROM hive.db.hoge WHERE dt = '20180601' ...

✔ 非常に高いスケーラビリティ

- バックエンドがS3なので**負荷の心配をする必要がほとんどない** <– write(=s3へのput) し放題
- 実際に処理を実行するEMRクラスタには注意しておく
  - Hive Metastoreを共通利用することで、EMRクラスタを用途別に分けて管理という戦略も取れる
  - バッチ処理用クラスタ, アドホック分析用クラスタ, 推薦システム用クラスタ, ...

#2. 高速推薦システムとそれを支えるデータレイク

まとめ

まとめ

✔ **ユーザー行動成す力学系とそれを支えるアーキテクチャ**

- Lambda, Kinesis stream, DynamoDB, DAX をフル活用
- DAXにより高速かつ安定的にユーザーの興味の変化を反映

✔ **高速な推薦システムとそれを支えるデータレイク**

- 耐久性の高い推薦API
  - DynamoDBのオートスケーリング & s3から非同期に行列データをロード
- S3 + Hive Metastoreによるデータレイク
  - データの詳細を隠蔽 & 利便性の向上
  - S3による耐久性&可用性

We are hiring!

✔ 募集職種例

» データ分析エンジニア

» サービスのKPI等の統計情報の設計 / 収集 / 分析

» 機械学習・自然言語処理エンジニア

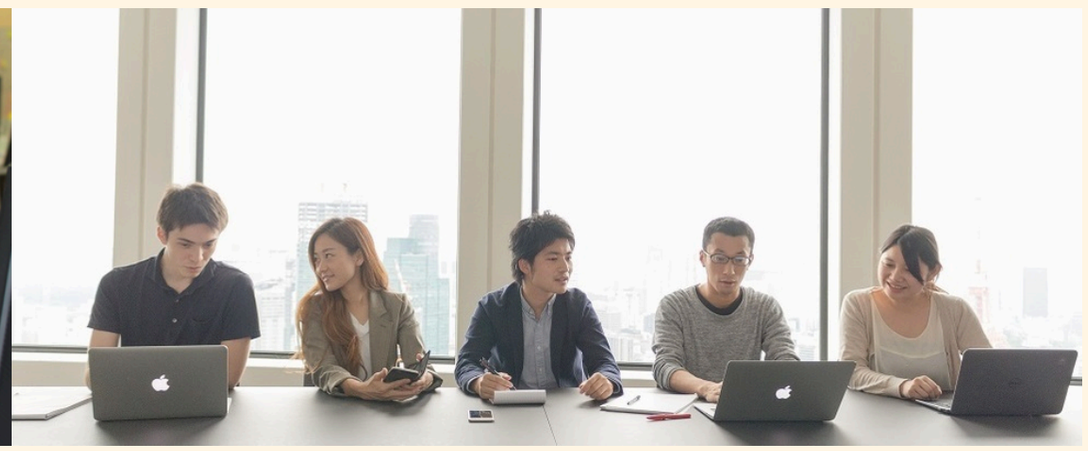
» 上記技術を含め数理モデルを駆使したアルゴリズムの開発

» アプリ開発エンジニア

» iOS / Android アプリの開発

» サーバーサイドエンジニア

» 各プロダクトのサーバーサイド開発



https://gunosy.co.jp/recruit/requirements/engineer/

Gunosy 採用