PostgreSQLでの
セマンティックサーチへの挑戦

堀本 泰弘
阿部 智晃

株式会社クリアコード

PostgreSQL Conference Japan 2025
2025-11-21

本日のゴール

PostgreSQLで簡単に
セマンティックサーチ
が使えることを
実感してもらう

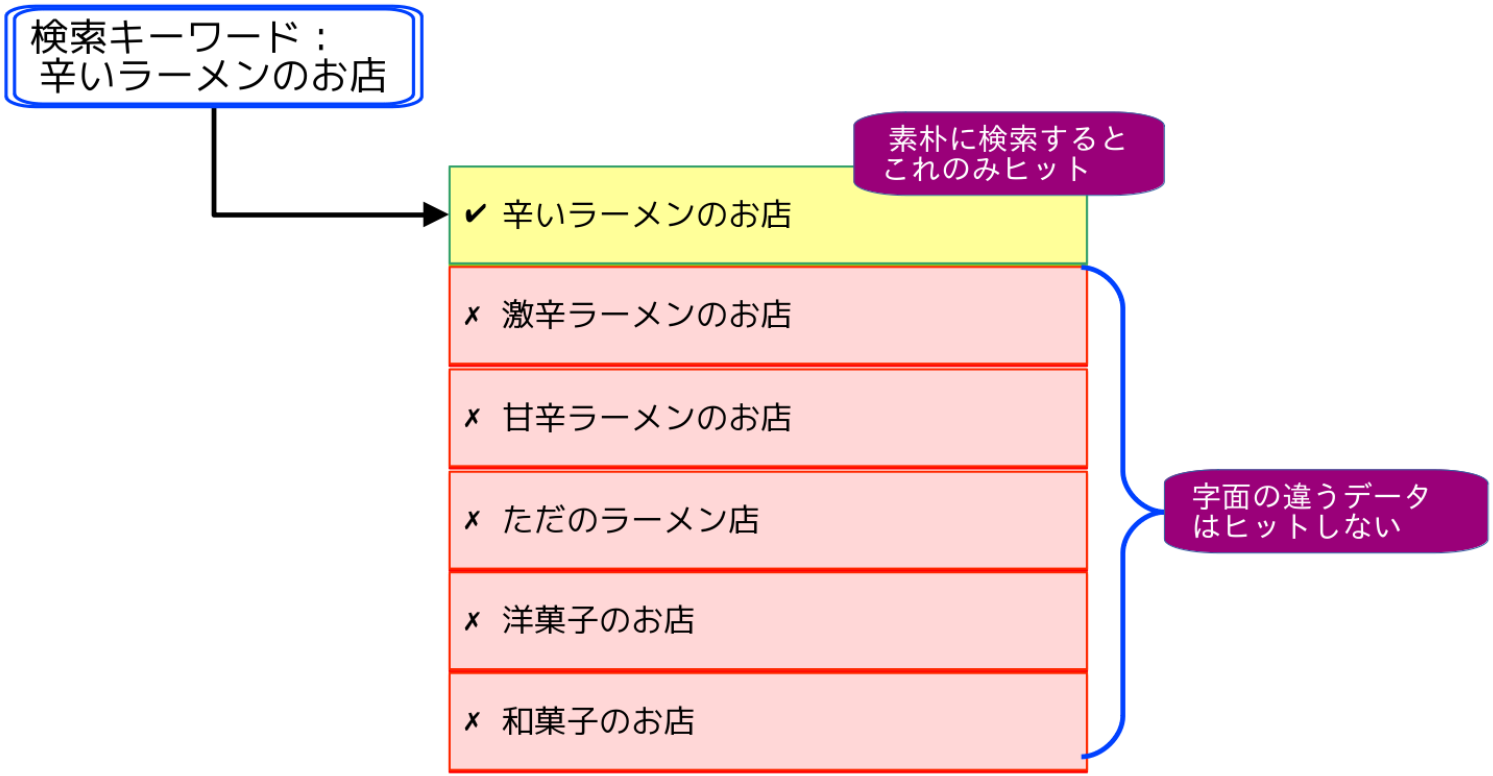
本日の内容

1. キーワード検索とセマンティックサーチの比較
2. セマンティックサーチを実現するために
3. PostgreSQLでセマンティクサーチ
4. 性能とユースケース

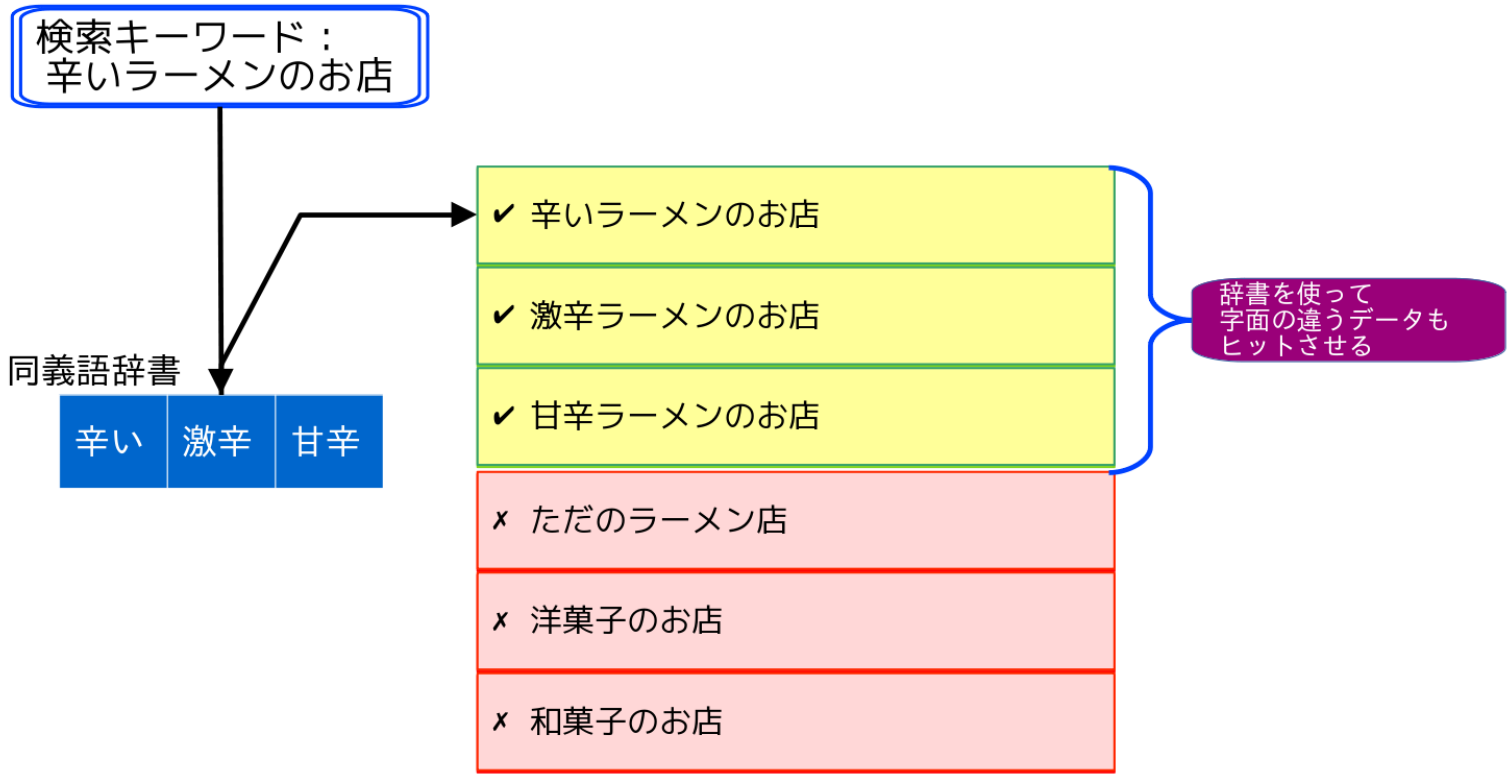
本日の内容

1. **キーワード検索とセマンティックサーチの比較**
2. セマンティックサーチを実現するために
3. PostgreSQLでセマンティクサーチ
4. 性能とユースケース

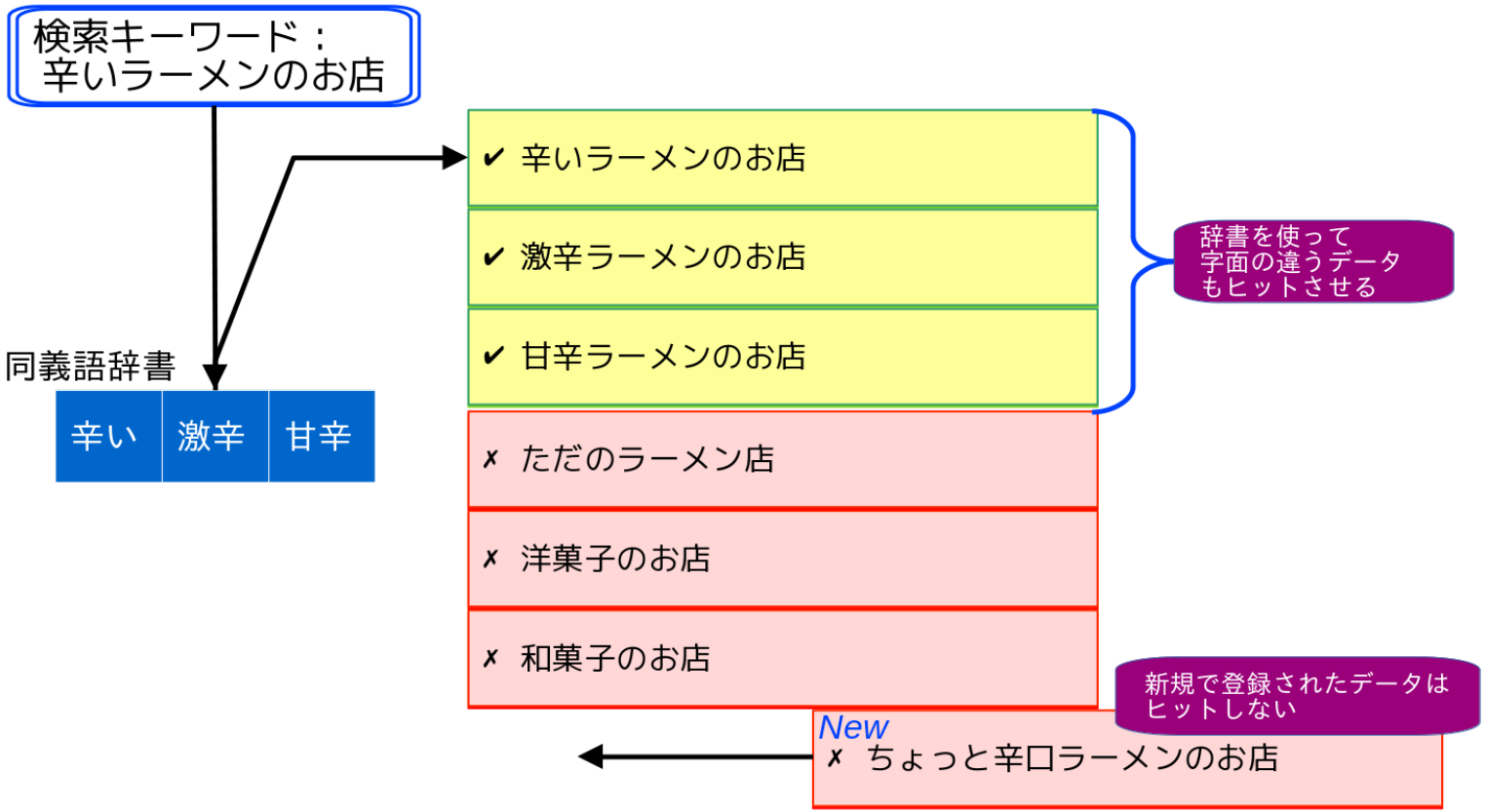
キーワード検索の利点と課題



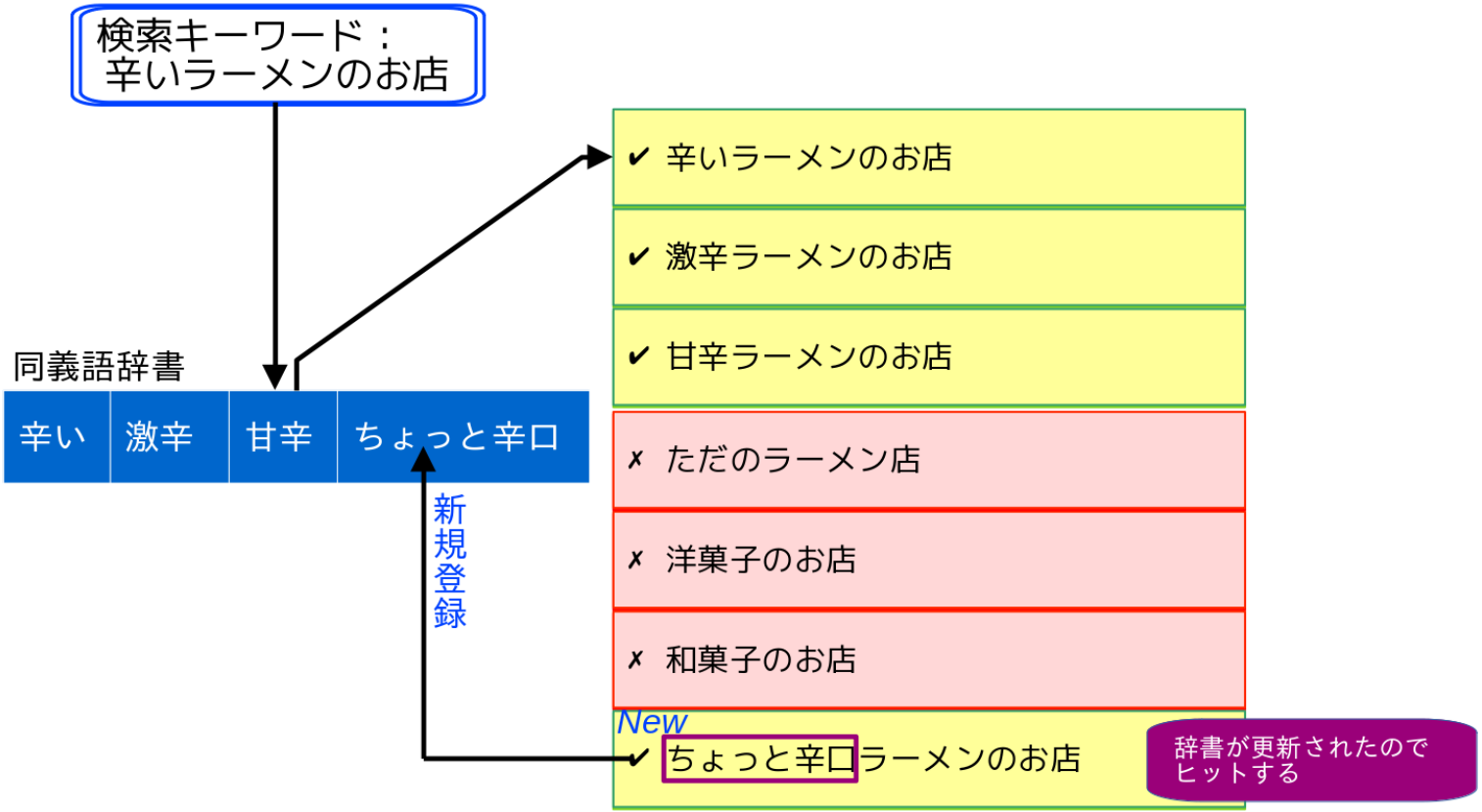
キーワード検索の利点と課題



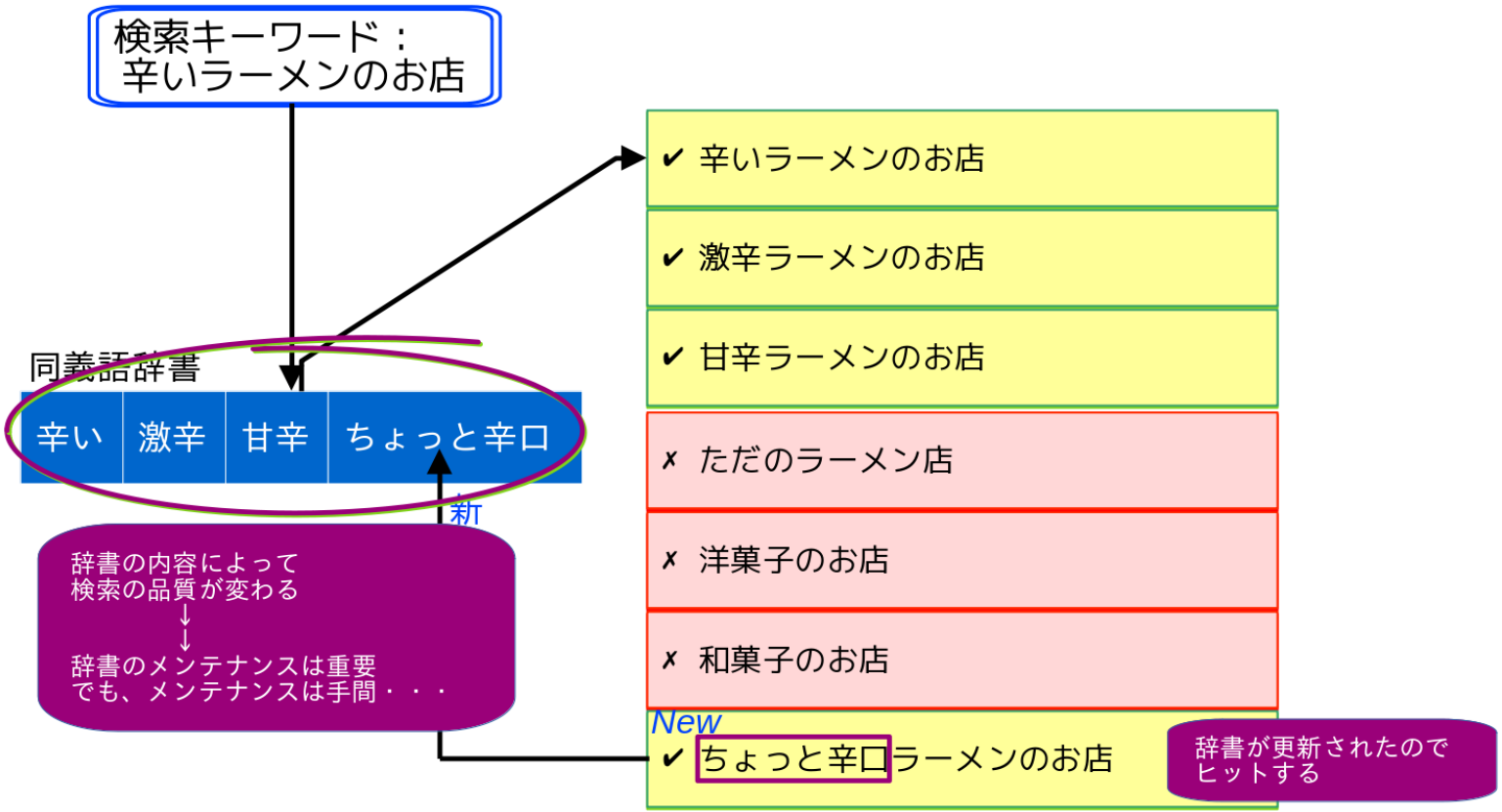
キーワード検索の利点と課題



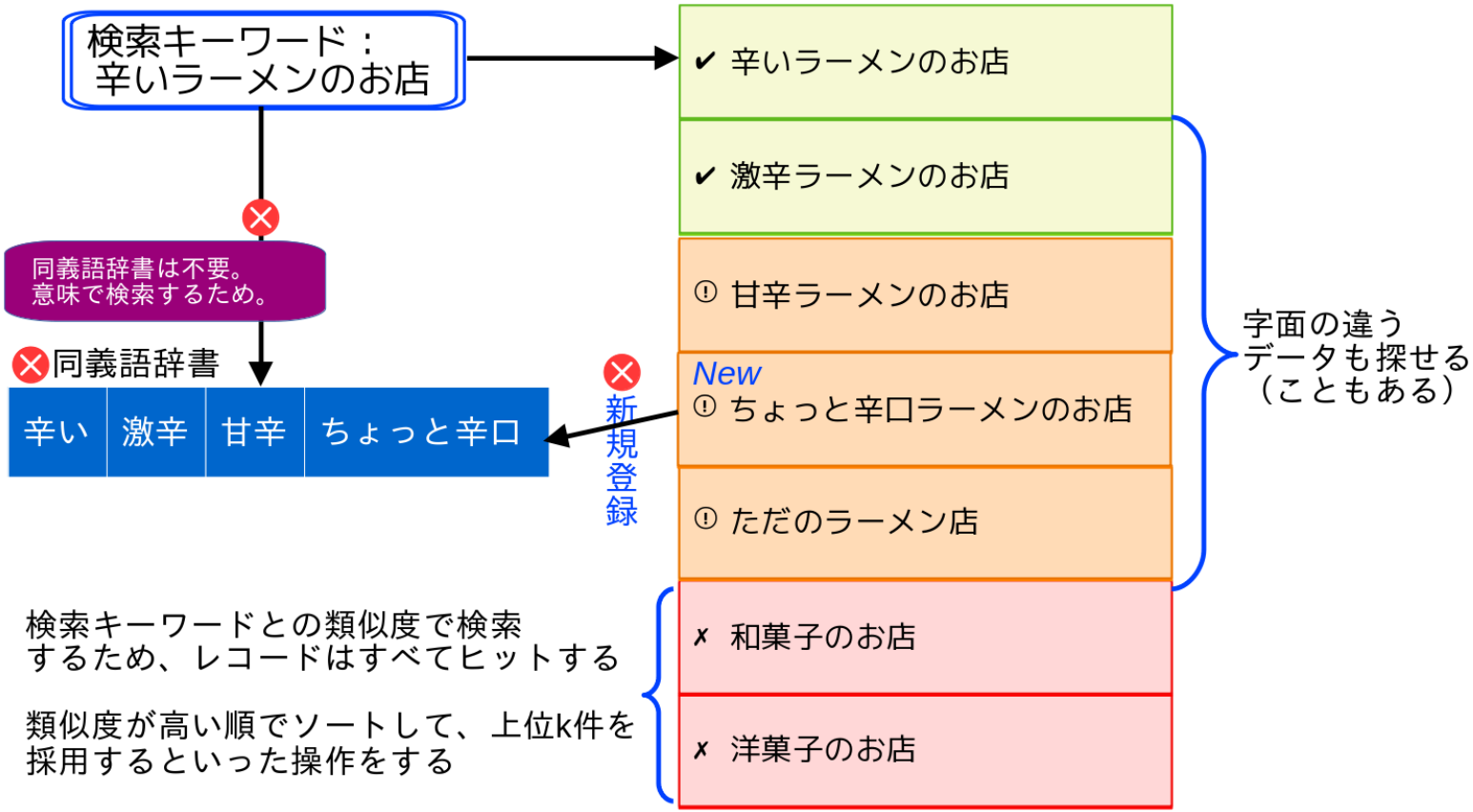
キーワード検索の利点と課題



キーワード検索の利点と課題



セマンティックサーチの利点と課題



本日の内容

1. キーワード検索とセマンティックサーチの比較
2. **セマンティックサーチを実現するために**
3. PostgreSQLでセマンティクサーチ
4. 性能とユースケース

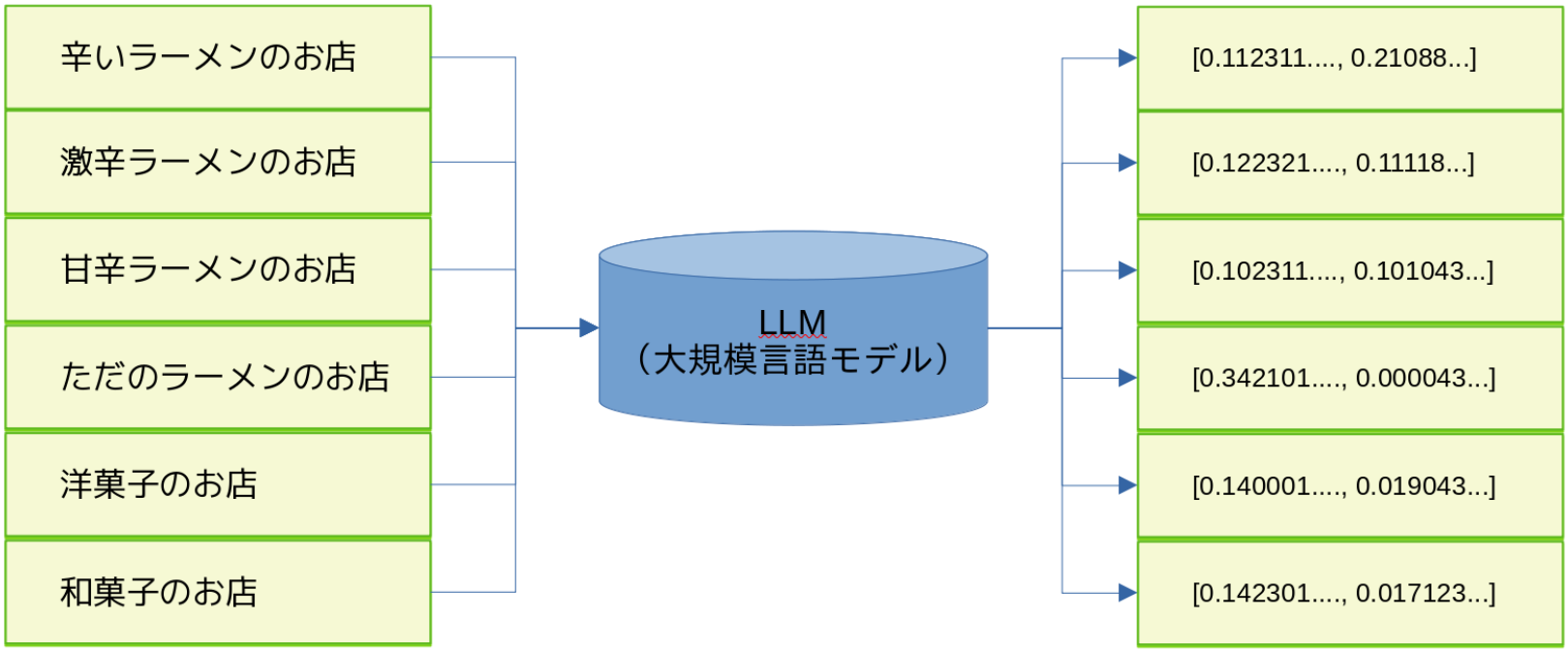
セマンティックサーチに必要な技術

1. テキスト -> ベクトルデータへの変換
2. ベクトルデータの類似度の計算
3. ベクトルデータの効率的な検索
4. ベクトルデータの圧縮

テキスト -> ベクトルデータへの変換

意味で検索するとは？

テキスト -> ベクトルデータへの変換



テキスト -> ベクトルデータへの変換

意味を**ベクトル**
として表現できる

テキスト -> ベクトルデータへの変換

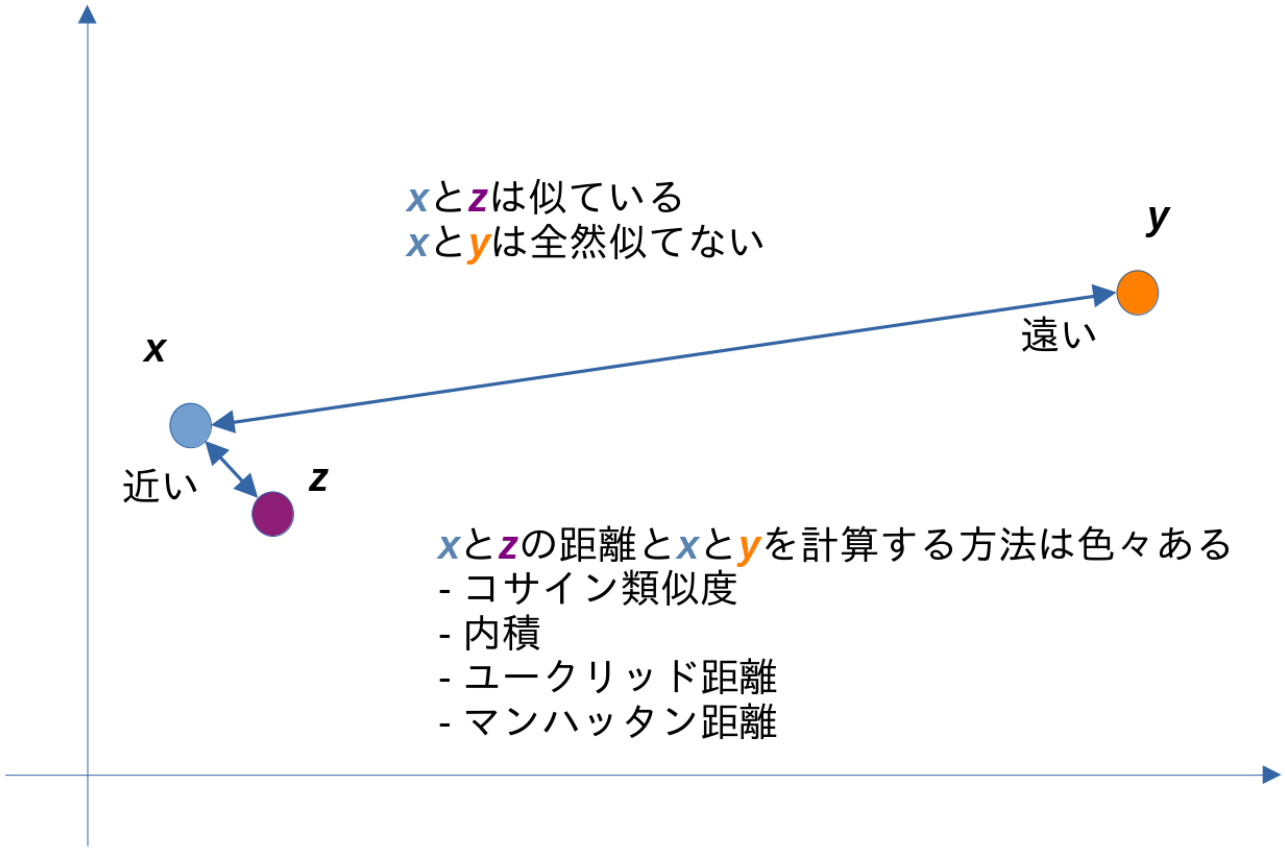
ベクトルは
距離を計算できる

ベクトルデータの類似度の計算

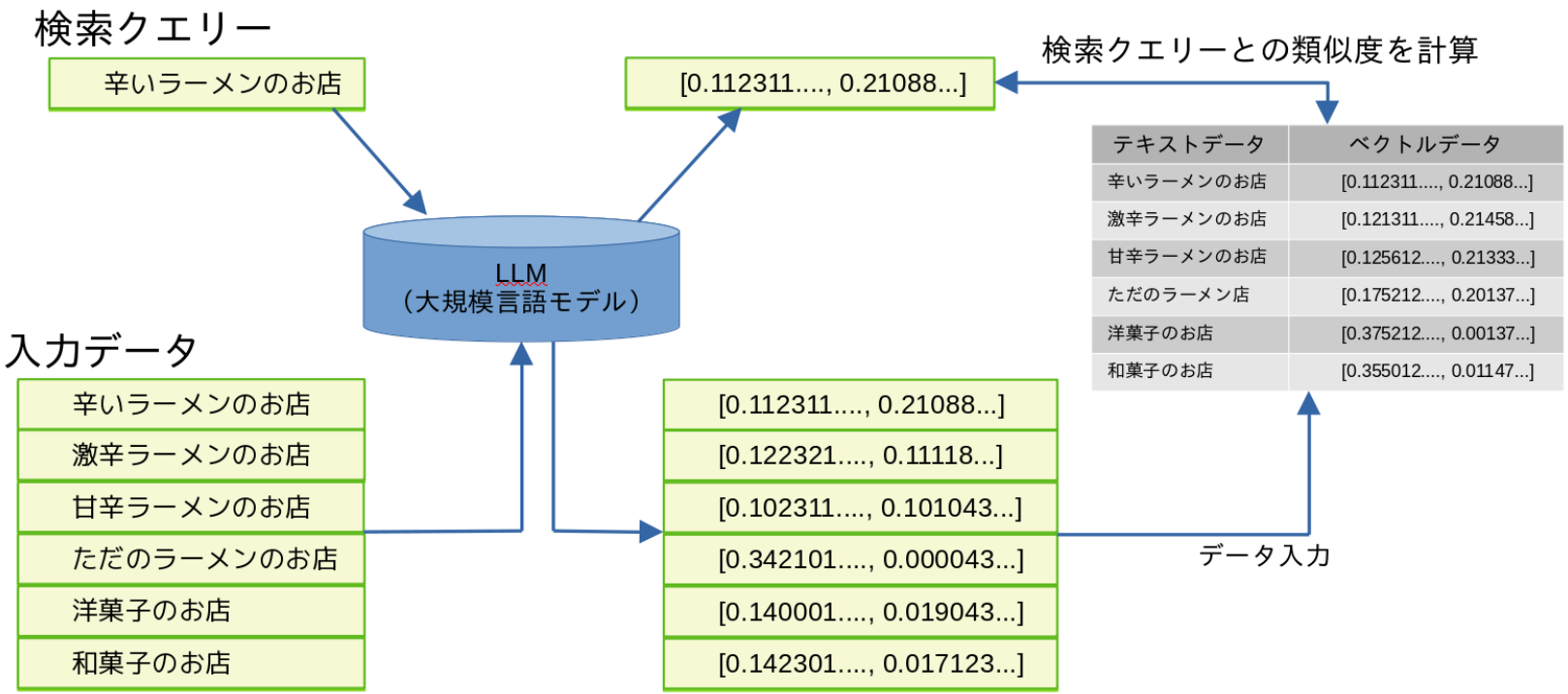
どれだけ似ているか

- ベクトル間の距離が近い ＝ 似ている
- ベクトル間の距離が遠い ＝ 似ていない

ベクトルデータの類似度の計算



ベクトルデータの類似度の計算



テキストデータ	ベクトルデータ
辛いラーメンのお店	[0.112311...., 0.21088...]
激辛ラーメンのお店	[0.121311...., 0.21458...]
甘辛ラーメンのお店	[0.125612...., 0.21333...]
ただのラーメン店	[0.175212...., 0.20137...]
洋菓子のお店	[0.375212...., 0.00137...]
和菓子のお店	[0.355012...., 0.01147...]

ベクトルデータの効率的な検索

全てのベクトルと類似度を計算するのか？

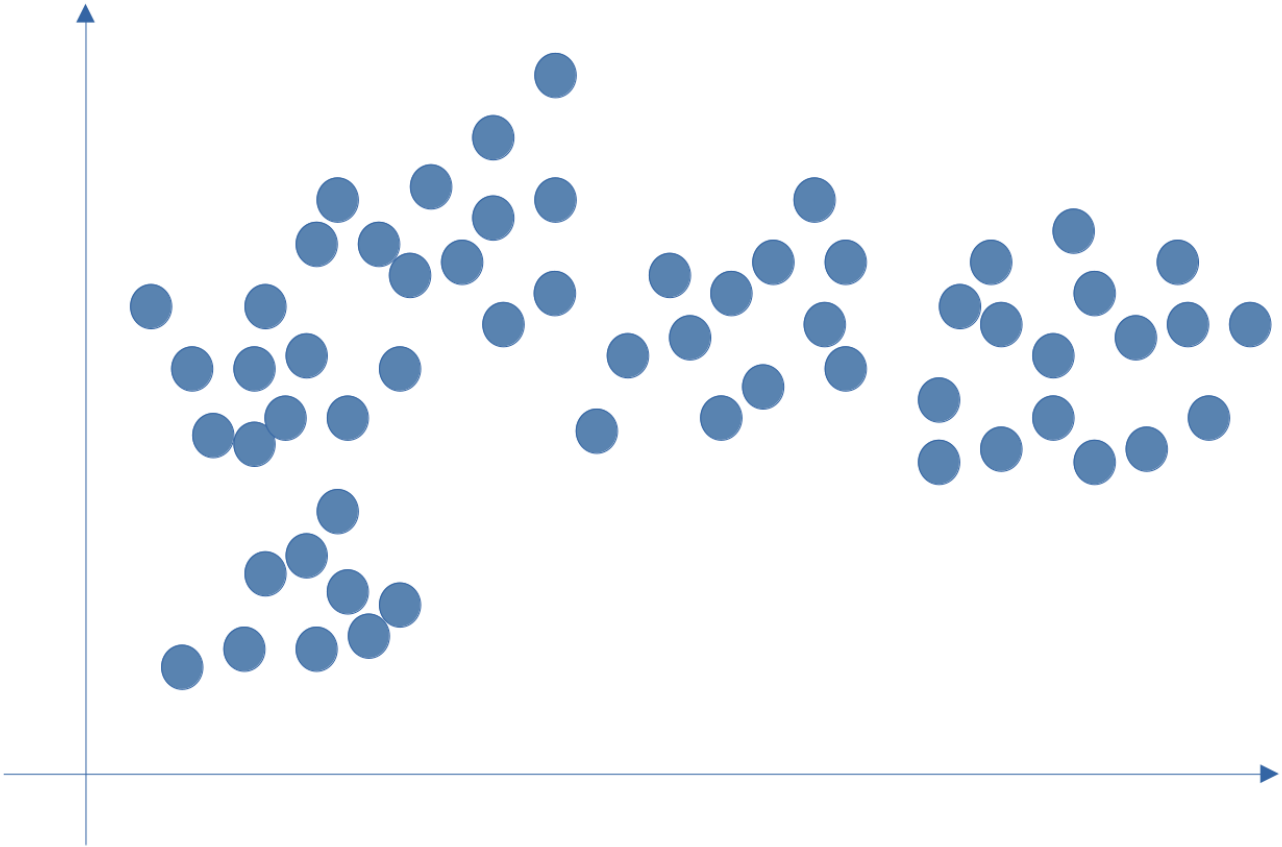
ベクトルデータの効率的な検索

線形探索は
現実的ではない

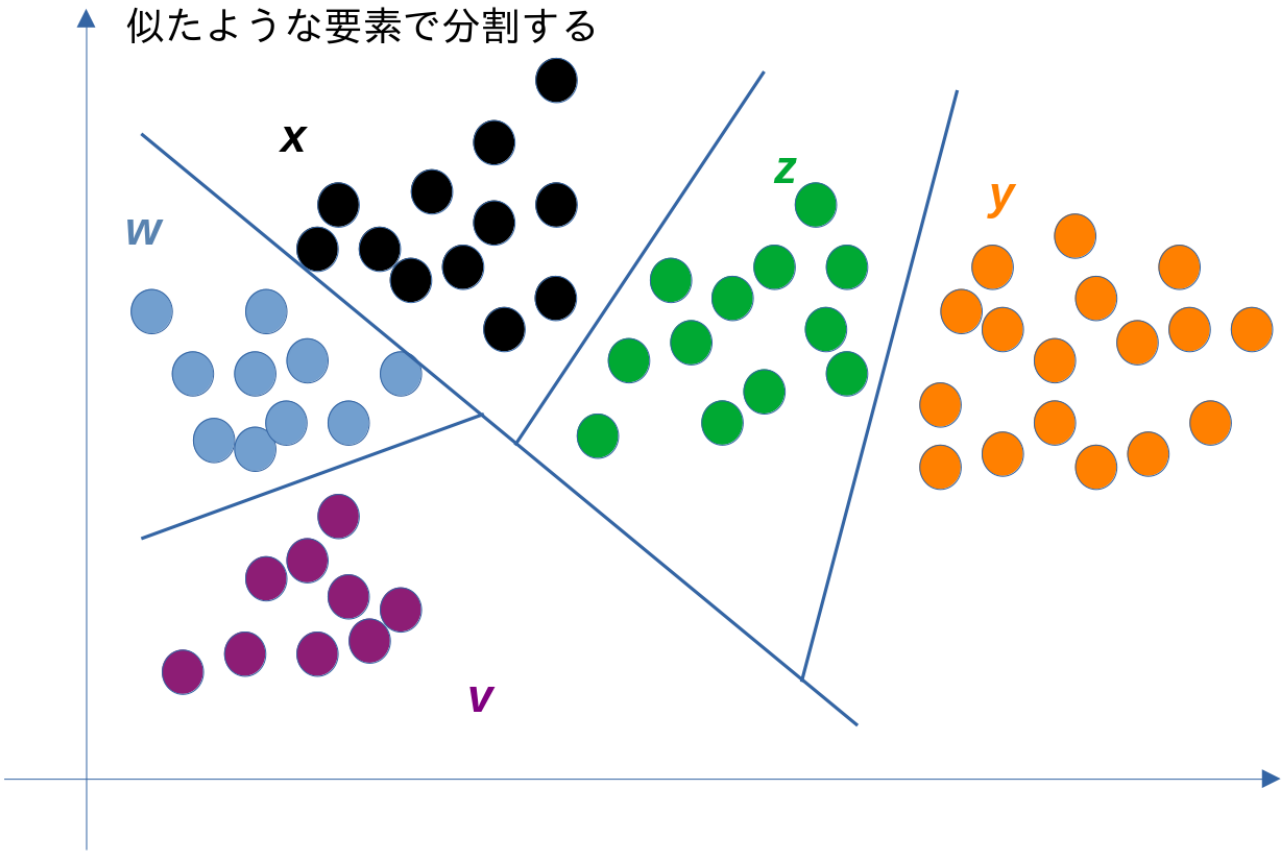
ベクトルデータの効率的な検索

一部のベクトルデータだけを検索

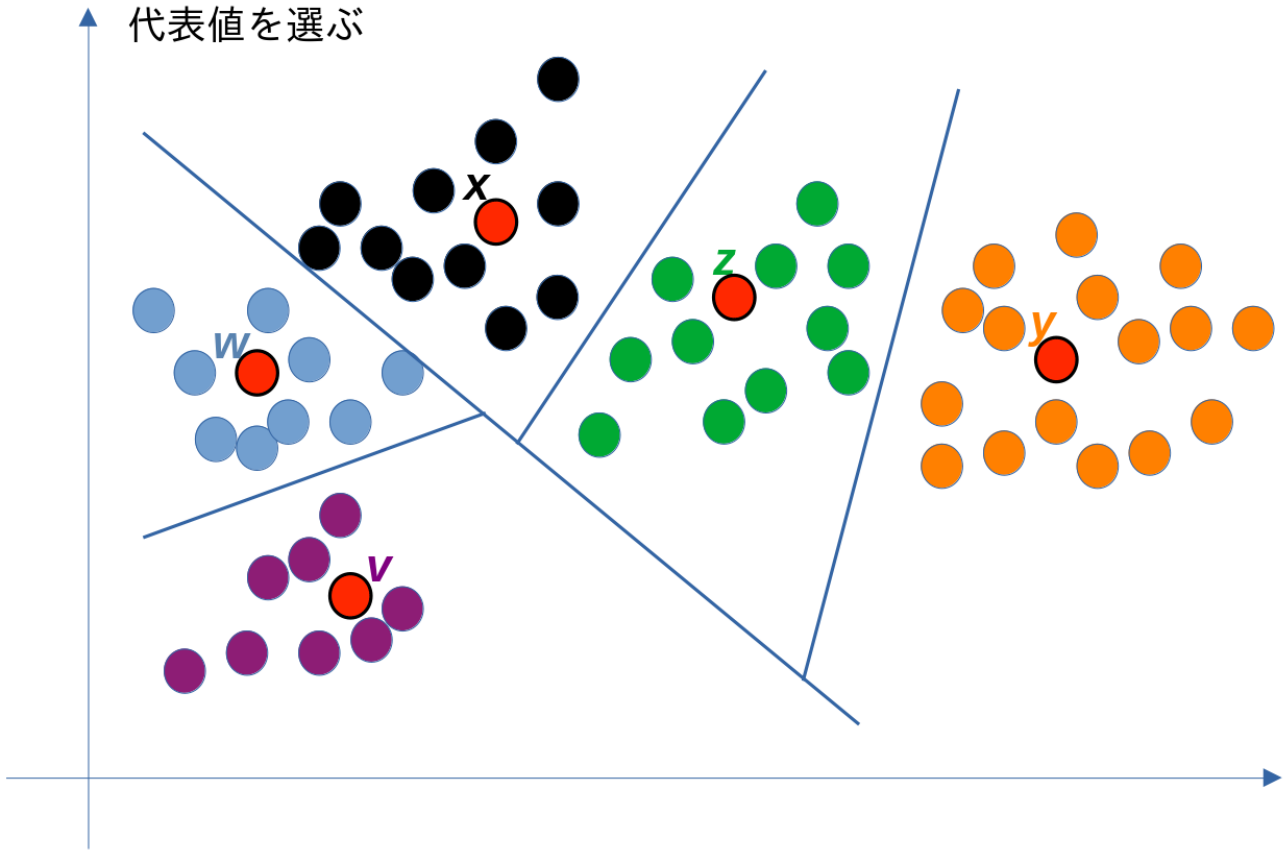
ベクトルデータの効率的な検索（イメージ）



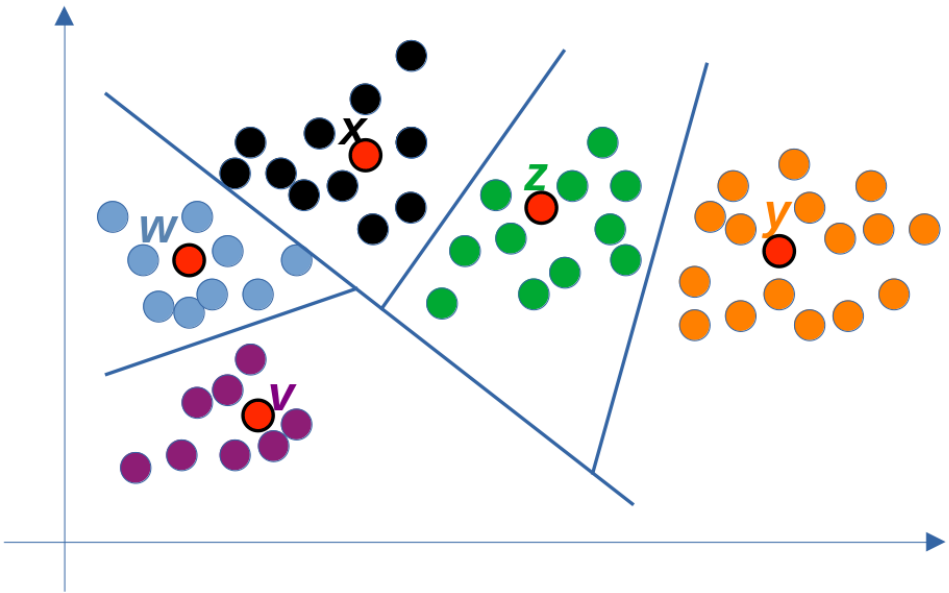
ベクトルデータの効率的な検索（イメージ）



ベクトルデータの効率的な検索（イメージ）



ベクトルデータの効率的な検索（イメージ）



代表値	その他のベクトル
v	
w	
x	
y	
z	

最初に代表値と比較
類似度の高いグループとだけ比較

検索クエリー

辛いラーメンのお店 → LLM（大規模言語モデル） → [0.112311...., 0.21088...]

ベクトルデータの圧縮

自然言語を変換した
ベクトルデータは
大きい

ベクトルデータの圧縮

次元数：大
▶データサイズ：大
▶メモリーに乗らない
▶ 性能劣化

ベクトルデータの圧縮

データ量の削減が必要

圧縮方法

主な圧縮方法

1. ベクトルの次元を削減
2. ベクトルの量子化

圧縮方法

どちらの方法も
圧縮前のベクトルの
特徴をなるべく維持
して圧縮

本日の内容

1. キーワード検索とセマンティックサーチの比較
2. セマンティックサーチを実現するために
3. **PostgreSQLでセマンティクサーチ**
4. 性能とユースケース

PostgreSQLのセマンティクサーチ

セマンティックサーチを使うには？

- pgvectorを使う
- PGroongaを使う

pgvectorとは？

- 類似したベクトルデータを検索する機能を提供するPostgreSQLの拡張機能
- 類似度の計算方法が多数提供されている
- インデックスを使用して検索を高速化できる

PGroonga（ぴーじーるんが）とは？

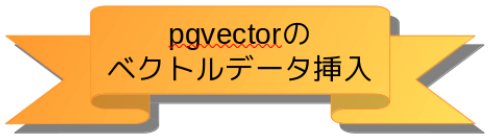
- 全言語対応の超高速全文検索機能を提供する拡張
- PostgreSQLのインデックスとして使える

PGroonga（ぴーじーるんが）とは？

- PostgreSQLのデータを使って全文検索する
  = ゼロETLで利用できる
- PostgreSQLの構文をほぼそのまま使える
  = 学習コストが低い
- **4.0.5からセマンティックサーチが使える**

pgvectorとPGroongaの違い

pgvectorのベクトルデータ挿入



データ投入

辛いラーメンのお店
激辛ラーメンのお店
甘辛ラーメンのお店
ただのラーメン店
洋菓子のお店
和菓子のお店

ベクトル化

[0.112311...., 0.21088...]
[0.121311...., 0.21458...]
[0.125612...., 0.21333...]
[0.175212...., 0.20137...]
[0.375212...., 0.00137...]
[0.355012...., 0.01147...]

テキストデータとベクトルデータを
一緒にINSERT

テキストデータ	ベクトルデータ
辛いラーメンのお店	[0.112311...., 0.21088...]
激辛ラーメンのお店	[0.121311...., 0.21458...]
甘辛ラーメンのお店	[0.125612...., 0.21333...]
ただのラーメン店	[0.175212...., 0.20137...]
洋菓子のお店	[0.375212...., 0.00137...]
和菓子のお店	[0.355012...., 0.01147...]

ユーザーが自分で実施

pgvectorとPGroongaの違い

PGroongaのベクトルデータ挿入

PGroongaの
ベクトルデータ挿入

データ投入

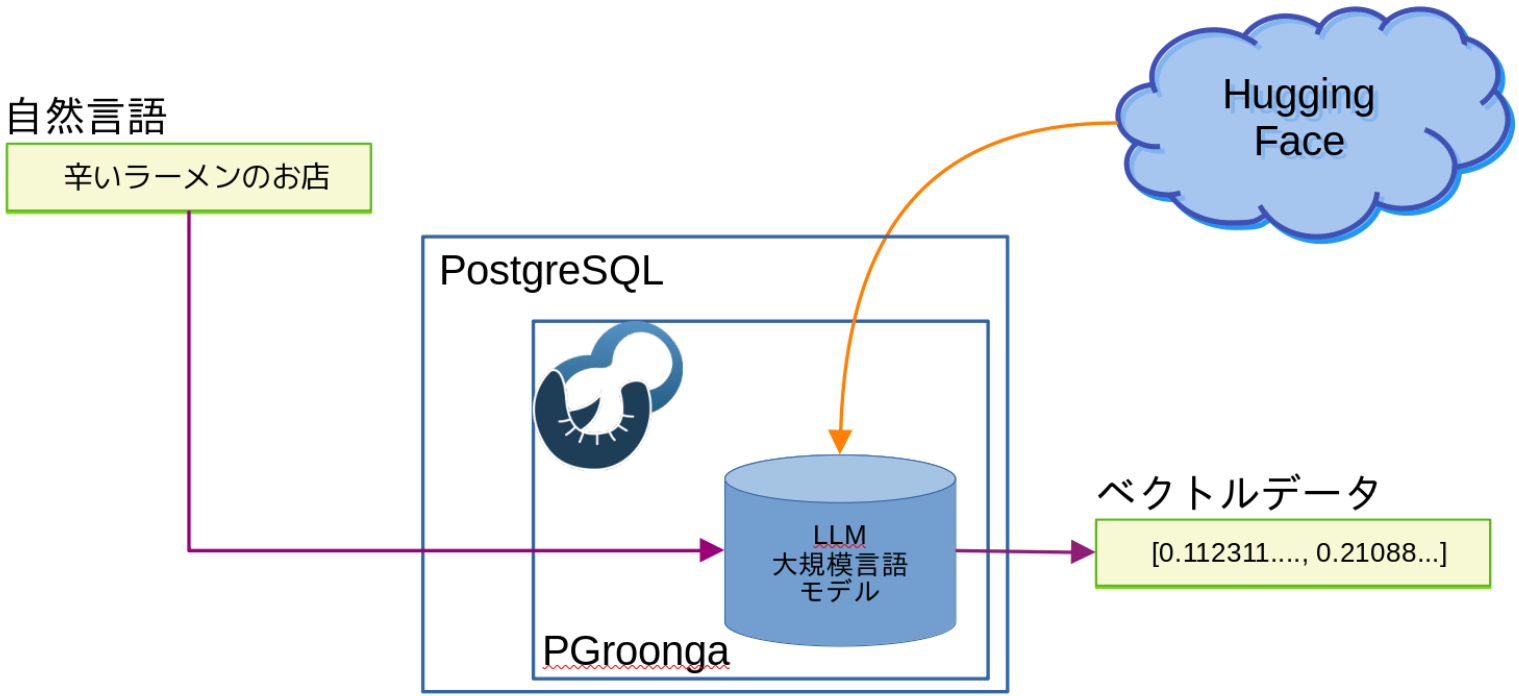
辛いラーメンのお店
激辛ラーメンのお店
甘辛ラーメンのお店
ただのラーメン店
洋菓子のお店
和菓子のお店

テキストデータのみINSERT
= 普通にINSERTするだけ

テキストデータ	ベクトルデータ
辛いラーメンのお店	[0.112311...., 0.21088...]
激辛ラーメンのお店	[0.121311...., 0.21458...]
甘辛ラーメンのお店	[0.125612...., 0.21333...]
ただのラーメン店	[0.175212...., 0.20137...]
洋菓子のお店	[0.375212...., 0.00137...]
和菓子のお店	[0.355012...., 0.01147...]

ベクトルデータは
INSERT時に自動生成

テキスト -> ベクトルデータへの変換



pgvectorのメリット

- ベクトルデータへ変換するサービスは複数あるので、用途に合わせた組み合わせができる
- ベクトルデータ変換と検索でリソースを分割できる

pgvectorのデメリット

- 変換したベクトルデータをシステム間でやりとりする必要がある

PGroongaのメリット

- ベクトルデータをシステム間でやりとりする必要がない
- RaBitQというアルゴリズムを使って、データサイズを少なくできる

PGroongaのデメリット

- ベクトルデータへの変換は固定的
- ベクトルデータ変換と検索でリソースが同一

RaBitQとは？

ベクトルデータの
量子化手法の一つ

RaBitQを採用した理由

データ量を大幅に
少なく出来る

データ量を大幅に少なく出来る

各32bit浮動少数点数を
1ビットで表現

データ量を大幅に少なく出来る

単純にデータサイズは
1/32になる

pgvectorとPGroongaでデータ登録と検索

pgvectorのデータ登録と検索

```
CREATE EXTENSION IF NOT EXISTS vector;



CREATE TABLE contents_for_pgvector (
  content text,
  content_embedding vector(384)
);
```

pgvectorのデータ登録と検索

```
INSERT INTO contents_for_pgvector (content, content_embedding)
       SELECT 'I am a boy', (pgroonga_language_model_vectorize(
                                 'hf:///groonga/all-MiniLM-L6-v2-Q4_K_M-GGUF',
                                 'I am a boy'));

INSERT INTO contents_for_pgvector (content, content_embedding)
       SELECT 'I am a dog', (pgroonga_language_model_vectorize(
                                 'hf:///groonga/all-MiniLM-L6-v2-Q4_K_M-GGUF',
                                 'I am a dog'));

INSERT INTO contents_for_pgvector (content, content_embedding)
       SELECT 'I am a king', (pgroonga_language_model_vectorize(
                                 'hf:///groonga/all-MiniLM-L6-v2-Q4_K_M-GGUF',
                                 'I am a king'));
```

pgvectorのデータ登録と検索

```
SELECT * FROM contents_for_pgvector;
--  I am a boy  | [-0.05031514,0.10813845,...,-0.12843993]
--  I am a dog  | [-0.03515085,-0.0059523215,...,-0.024966048]
--  I am a king | [-0.030026972,0.057919234,...,-0.09476562]
-- (3 行)
```

pgvectorのデータ登録と検索

```
SELECT contents_for_pgvector.content,
       (contents_for_pgvector.content_embedding <#> query_embedding.query) * -1 AS inner_product
  FROM contents_for_pgvector,
       (
          SELECT CAST(pgroonga_language_model_vectorize(
                         'hf:///groonga/all-MiniLM-L6-v2-Q4_K_M-GGUF',
                         'boy') AS vector) AS query
       ) query_embedding
ORDER BY inner_product DESC;
--    content   |   inner_product
-- -------------+-------------------
--  I am a boy  | 0.5909400582313538
--  I am a dog  | 0.2753480076789856
--  I am a king | 0.1733587086200714
-- (3 行)
```

PGroongaのインデックス登録

```
DROP INDEX IF EXISTS pgroonga_content_semantic_search_index;
CREATE INDEX pgroonga_content_semantic_search_index ON contents_for_pgroonga
 USING pgroonga (content pgroonga_text_semantic_search_ops_v2)
 WITH (plugins = 'language_model/knn',
       model = 'hf:///groonga/multilingual-e5-large-Q4_K_M-GGUF');
```

PGroongaのデータ登録と検索

```
CREATE EXTENSION IF NOT EXISTS pgroonga;



CREATE TABLE contents_for_pgroonga (
  content text
);
```

PGroongaのデータ登録と検索

```
INSERT INTO contents_for_pgroonga VALUES ('I am a boy');
INSERT INTO contents_for_pgroonga VALUES ('I am a dog');
INSERT INTO contents_for_pgroonga VALUES ('I am a king');
SELECT * FROM contents_for_pgroonga;
--     content
-- ------------
--   I am a boy
--   I am a dog
--   I am a king
-- (3 行)
```

PGroongaのデータ登録と検索

```
SELECT content, pgroonga_score(tableoid, ctid)
  FROM contents_for_pgroonga
 WHERE content &@* pgroonga_condition('boy');
--     content    |    pgroonga_score
-- ---------------+---------------------
--   I am a boy   | 0.8671597838401794
--   I am a king  | 0.5496522784233093
--   I am a dog   | 0.5450347065925598
-- (3 行)
```

PGroongaのデータ登録と検索

PGroongaが内部で保持している
ベクトルデータ

```
-- [1,1,"I am a boy","mjctvB1JDj5pur67C/ ... ICPMFQi7wSL8q9"],
-- [2,2,"I am a dog","BwGXu4YNUj2Yi5k9j+ ... nDPPL4ijySlqy7"],
-- [3,3,"I am a king","1lklOxiXvj2rSB69VG ... tbvVHmDr1wSFK9"]
```

本日の内容

1. キーワード検索とセマンティックサーチの比較
2. PostgreSQLでセマンティクサーチ
3. セマンティックサーチを実現するために
4. **性能とユースケース**

測定環境

- GPUなし
- PostgreSQL 18
- PGroonga mainブランチ
- pgvector v0.8.1
  - IVFFlatインデックス

測定に使ったデータ

- Wikipediaのタイトル
  - 1,477,194 件

測定項目

- 検索速度
- ベクトルデータのサイズ
- インデックス作成時間

検索速度：PGroonga

検証用のクエリ：

```
SET enable_seqscan = off;



SELECT COUNT(title) FROM wikipedia
ORDER BY title <&@*> pgroonga_condition('異世界転生して無双したよ')
LIMIT 5;
```

検索速度：pgvector

検証用のクエリ：

```
SET enable_seqscan = off;



SELECT COUNT(title) FROM wikipedia
ORDER BY embedding <-> '['異世界転生して無双したよ'のベクトル]'
LIMIT 5;
```

検索速度：結果：PGroonga

5回実行した中央値：108.868 ms
ベクトルの生成も含む

検索速度：結果：pgvector

5回実行した中央値：12.166 ms
ベクトルの生成は含まず

ベクトルデータサイズ

- float4の384次元のベクトルデータ:
  4バイト * 384 = 1,536バイト

PGroongaのベクトルデータサイズ

- PGroongaではfloat4の384次元のベクトルデータを量子化
- 「iJDX5WFlJ7wwvR ... +sHiH0KxDv1PoHT8=」という64バイトのバイナリーで保持
- 64文字 = 64バイト

PGroongaのインデックス作成時間

- PGroongaの場合インデックスの作成時間は「ベクトル化 + インデックスの作成時間」
- ベクトル化がすごく遅い
- ただ、GPUの有無や、使う言語モデルによっても大きく速度が変わる

ユースケース:
PostgreSQLのドキュメントを検索

テーブルを消す
クエリを検索

キーワード検索：結果

```
SELECT title, SUBSTRING(content FROM 0 FOR 10)
FROM jpug_doc_contents
WHERE content LIKE '%テーブル%消す%'
LIMIT 5;
--         title         |    substring
-- ------------------+-----------------
--  contrib             | 付録F 追加で提供
--  ddl-priv            | 5.8. 権限
--  ddl-schemas         | 5.10. スキー
--  explicit-locking    | 13.3. 明示的
--  extend-extensions | 36.17. 関連
-- (5 rows)
```

セマンティックサーチ：結果

PGroongaだと直接テキストで検索できる

```
SELECT title, SUBSTRING(content FROM 0 FOR 10)
FROM jpug_doc_contents
ORDER BY content <&@*> pgroonga_condition('テーブルを消す')
LIMIT 5;
--         title         |   substring
-- ------------------+--------------
--  tutorial-table     | 2.3. 新しいテ
--  ddl-basics         | 5.1. テーブル
--  ddl-alter          | 5.7. テーブル
--  sql-delete         | DELETEDEL
--  sql-droptablespace | DROP TABL
-- (5 rows)
```

まとめ

PostgreSQLから簡単にセマンティックサーチを使えるようになりました！