AI-T3-02

RAG における検索システムの権限分離と評価

Shuhei Fukami
Solutions Architect
アマゾン ウェブ サービス ジャパン合同会社

本セッションについて

想定聴講者

- Retrieval-Augmented Generation (RAG) の本番利用にむけた構築や改善を試みている
- AWS サービスの基本的な知識がある
- RAG の一般的な構成やその関連要素を理解している

ゴール

以下のような疑問に答えることができる

- RAG のテナント分離パターンや権限管理のベースアイディアはどんなものがある？
- RAG の精度改善をどこから、どのように始めればいいのか？

Agenda

RAG におけるテナント分離パターンと権限分離

- RAG におけるマルチテナントでの分離パターン
- 分離パターンにおけるデータストアごとの特性の考慮
- 認証情報を用いたテナントごとのルーティング

RAG における検索システムの評価と改善

- RAG における検索メカニズムの重要性
- 検索システムのオフライン評価指標
- 評価と改善におけるテナント分離パターンの影響

Agenda

RAG におけるテナント分離パターンと権限分離

- RAG におけるマルチテナントでの分離パターン
- 分離パターンにおけるデータストアごとの特性の考慮
- 認証情報を用いたテナントごとのルーティング

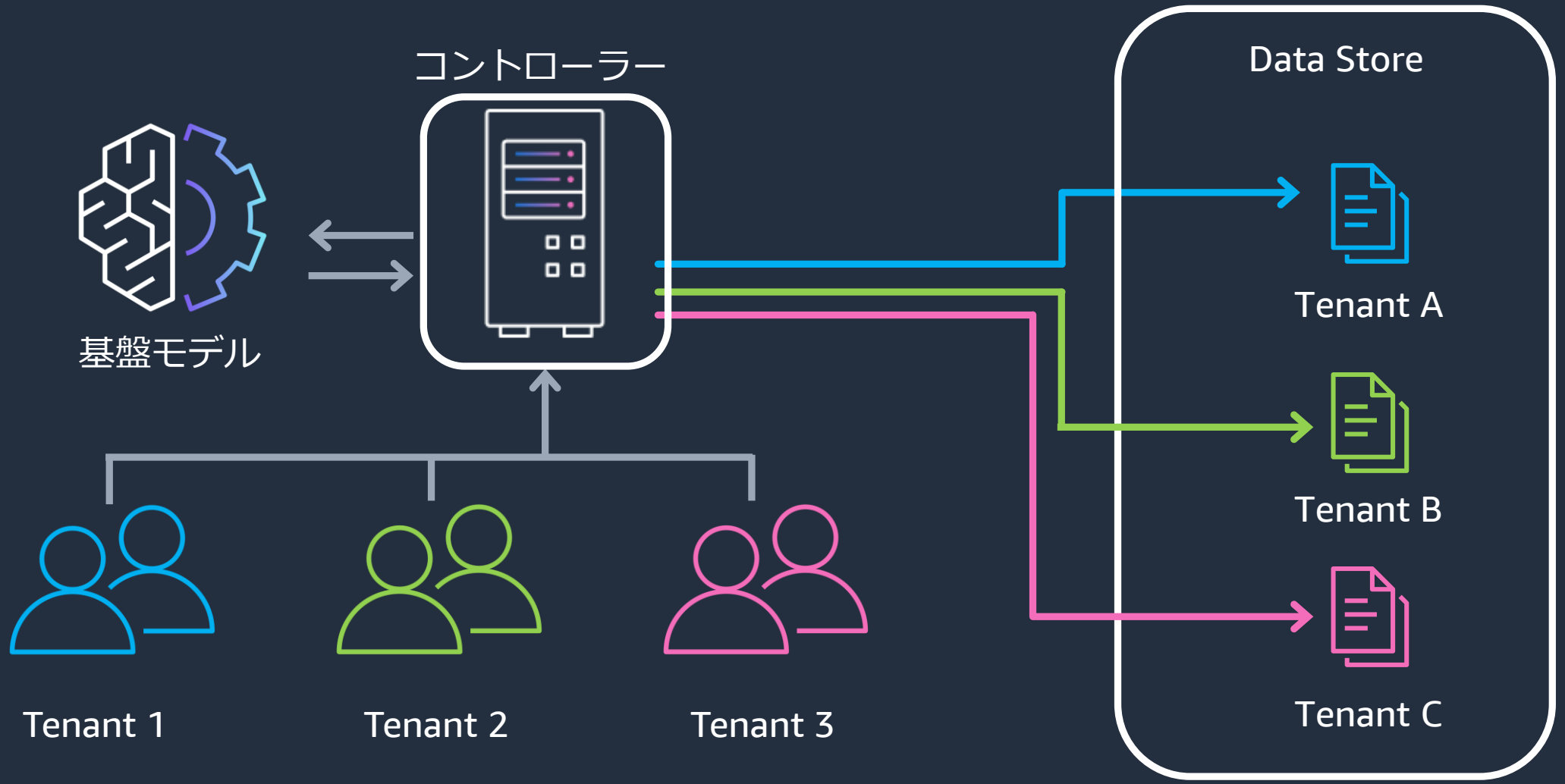
RAG における検索システムの評価と改善

- RAG における検索メカニズムの重要性
- 検索システムのオフライン評価指標
- 評価と改善におけるテナント分離パターンの影響

RAG におけるマルチテナントでの分離パターン

構成検討における考慮点

※ RAG : Retrieval-Augmented Generation

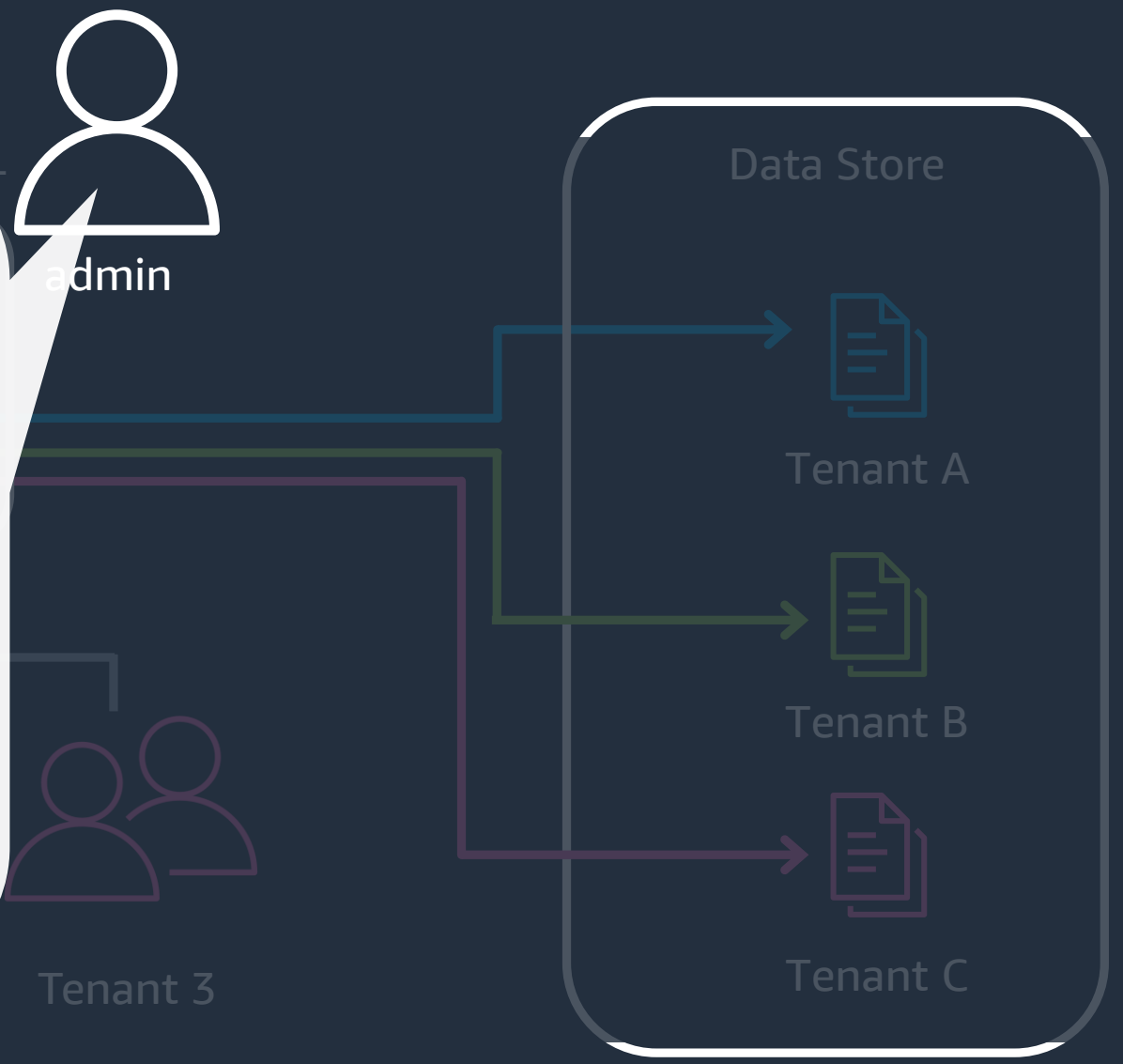


RAG におけるマルチテナントでの分離パターン

構成検討における考慮点

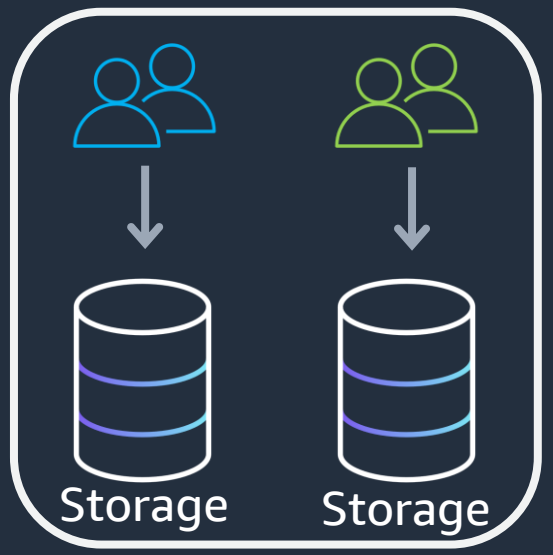
何を考慮して分離の
パターンを選択するべき？

どんな実装方法が
あるだろう？

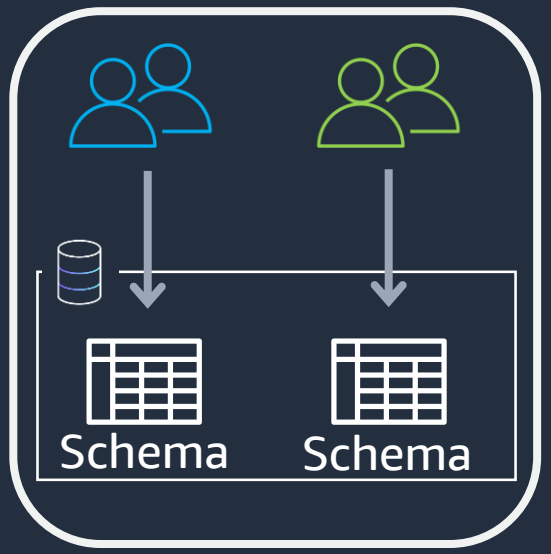


SaaS におけるデータ分離パターン

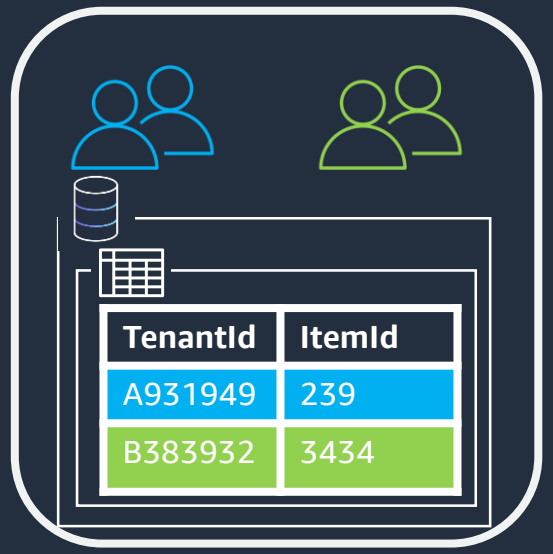
データストアにおける一般的な分離モデル



サイロ



ブリッジ



プール

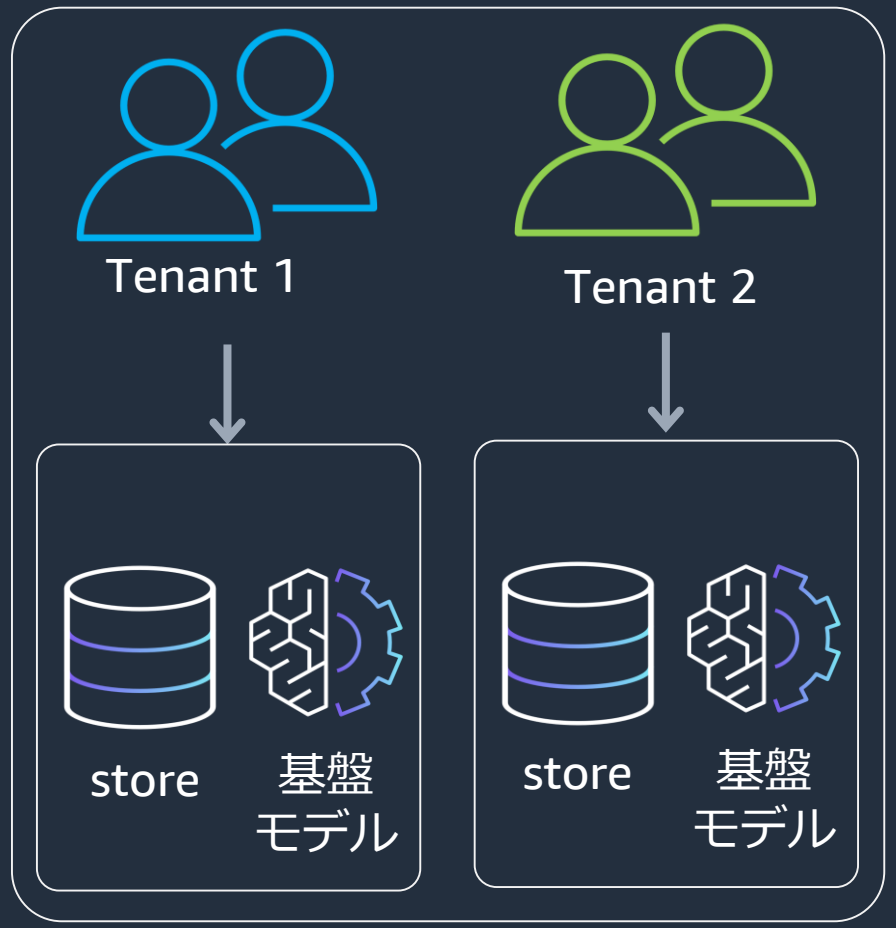
ユースケースごとに適切なパターンを選択することが重要

※「SaaS Storage Strategies by AWS」より

https://docs.aws.amazon.com/whitepapers/latest/multi-tenant-saas-storage-strategies/multi-tenant-saas-storage-strategies.html

RAG におけるマルチテナントでの分離パターン

サイロモデル



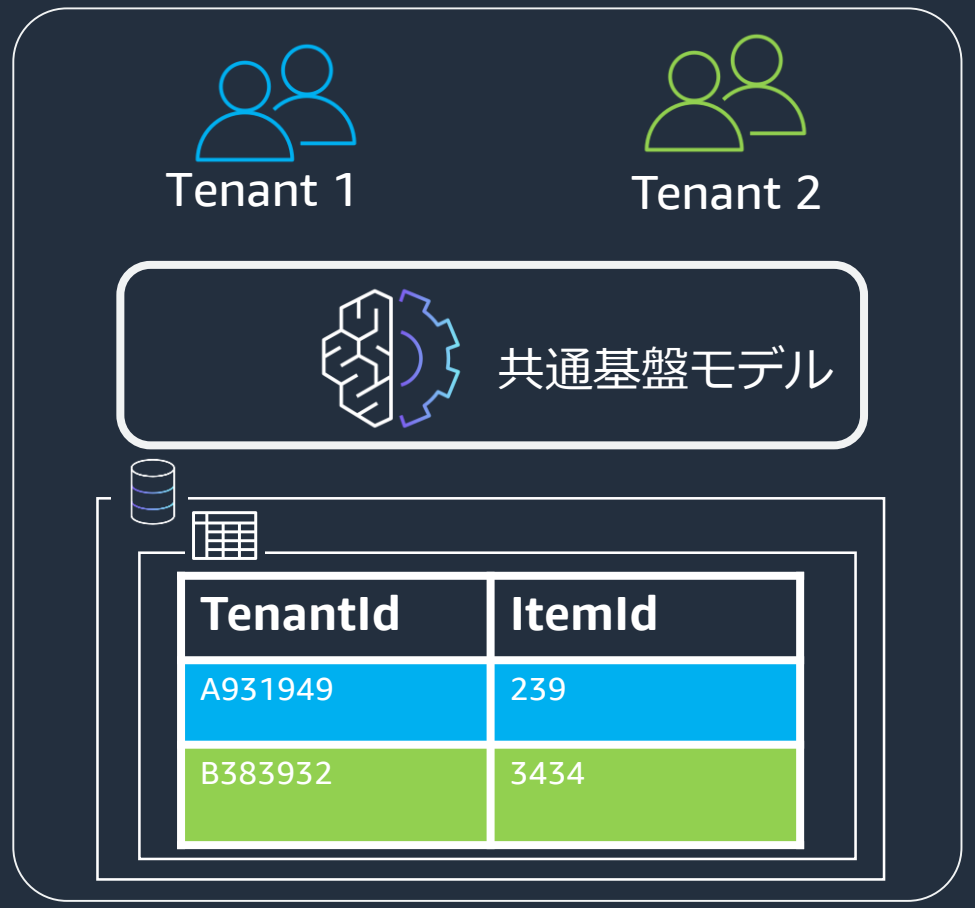
メリット

- **テナントごとに柔軟な対応が可能**
  - データストア側の設定の変更
    (ハイブリッド検索時の辞書の管理など)
  - embedding model や チャットのモデルのテナントごとの最適化
  - テナントごとのメンテナンスの調整
- **明確な分離境界**
- 他テナントの処理の影響を受けにくい
- 各コンポーネントのテナント別の利用状況を追跡しやすい

デメリット

- 個別の構成の検討の必要性
- 管理コストの増加

RAG におけるマルチテナントでの分離パターン
プールモデル

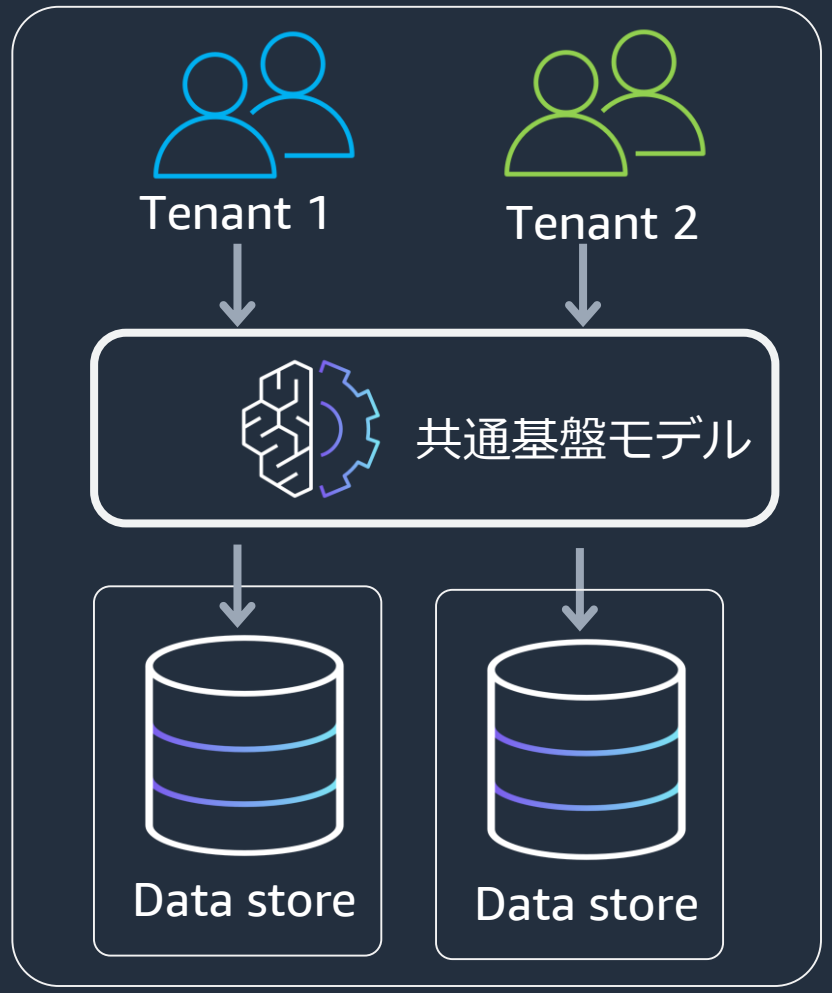


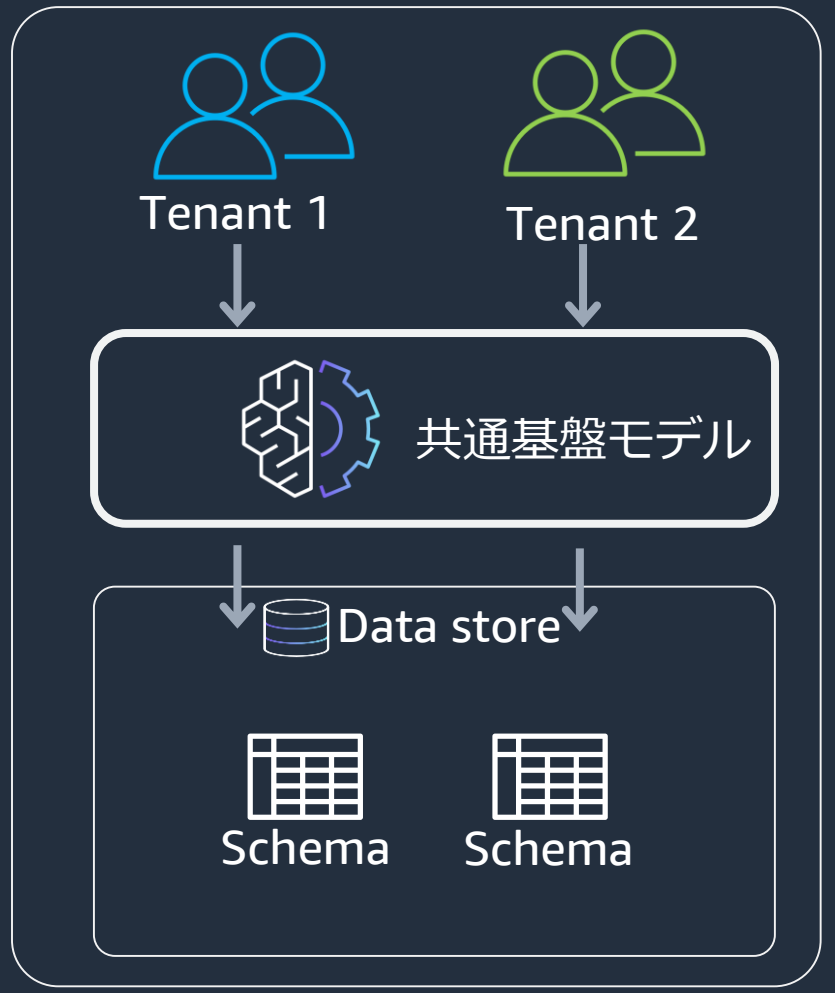
メリット
多数のテナントがある場合に
- コスト効率/リソース効率の向上
- 管理コストの削減

デメリット
- テナントごとの設定の難化
- 権限分離の複雑化
- 統計情報への他テナントの影響
- ノイジーネイバーの問題
- テナント別の利用状況の追跡の難化
- ドキュメント削除などの運用の複雑化

RAG におけるマルチテナントでの分離パターン

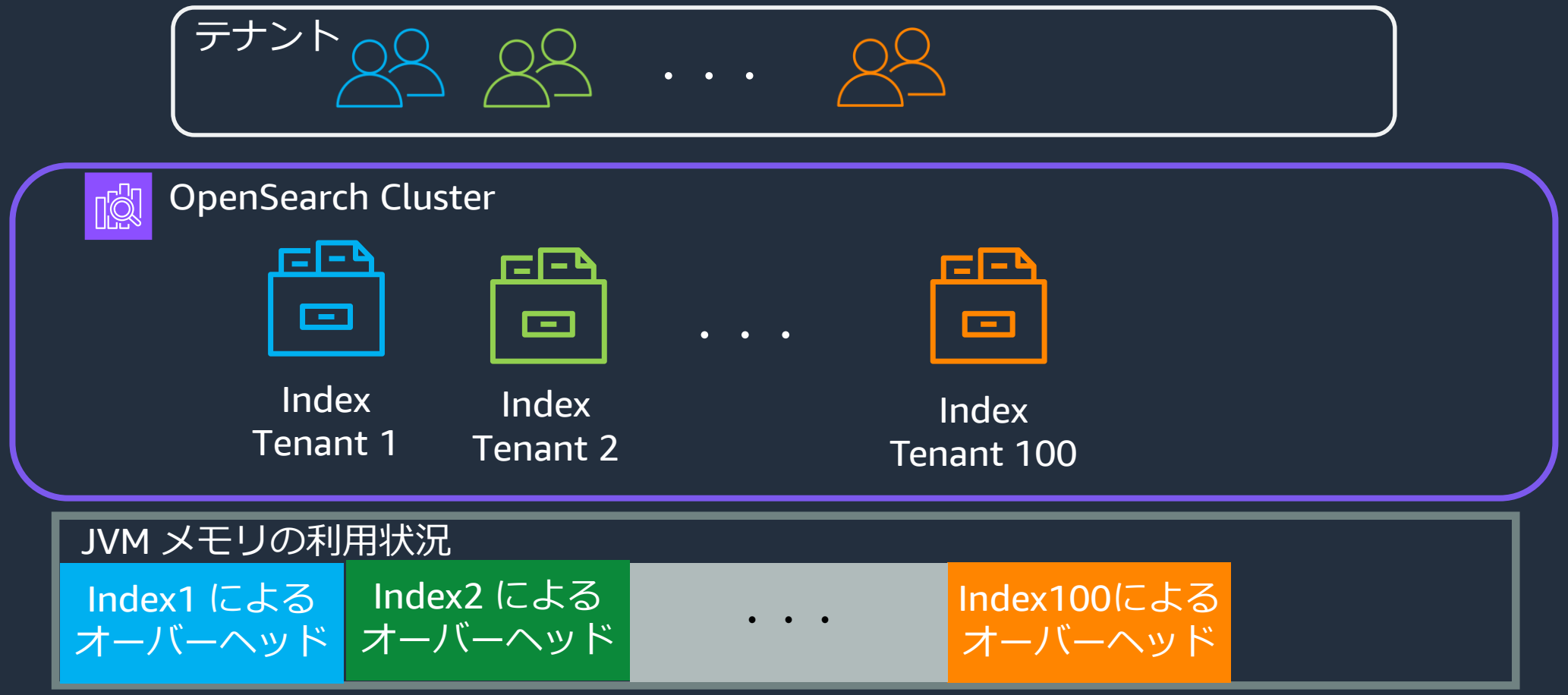
ブリッジモデル





分離パターンにおけるデータストアごとの特性の考慮

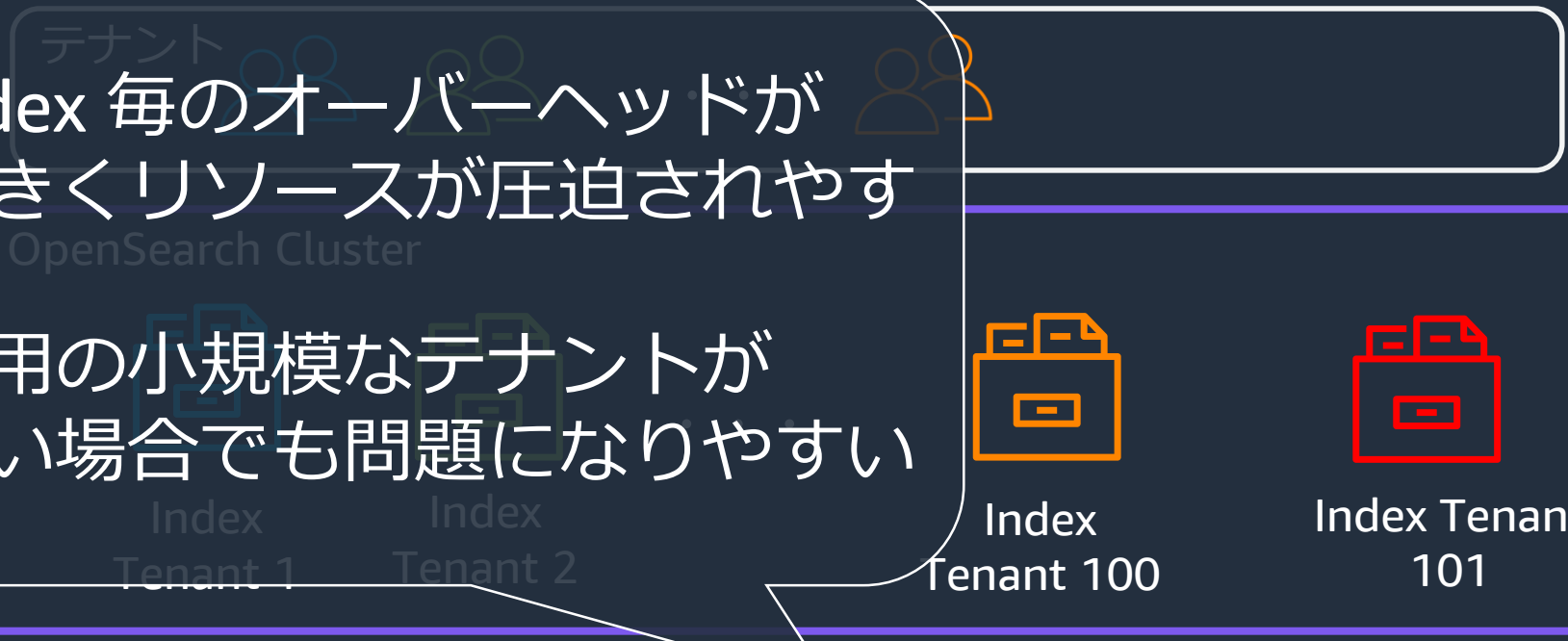
具体例1: OpenSearch で index 単位のブリッジモデルを利用する場合



分離パターンにおけるデータストアごとの特性の考慮

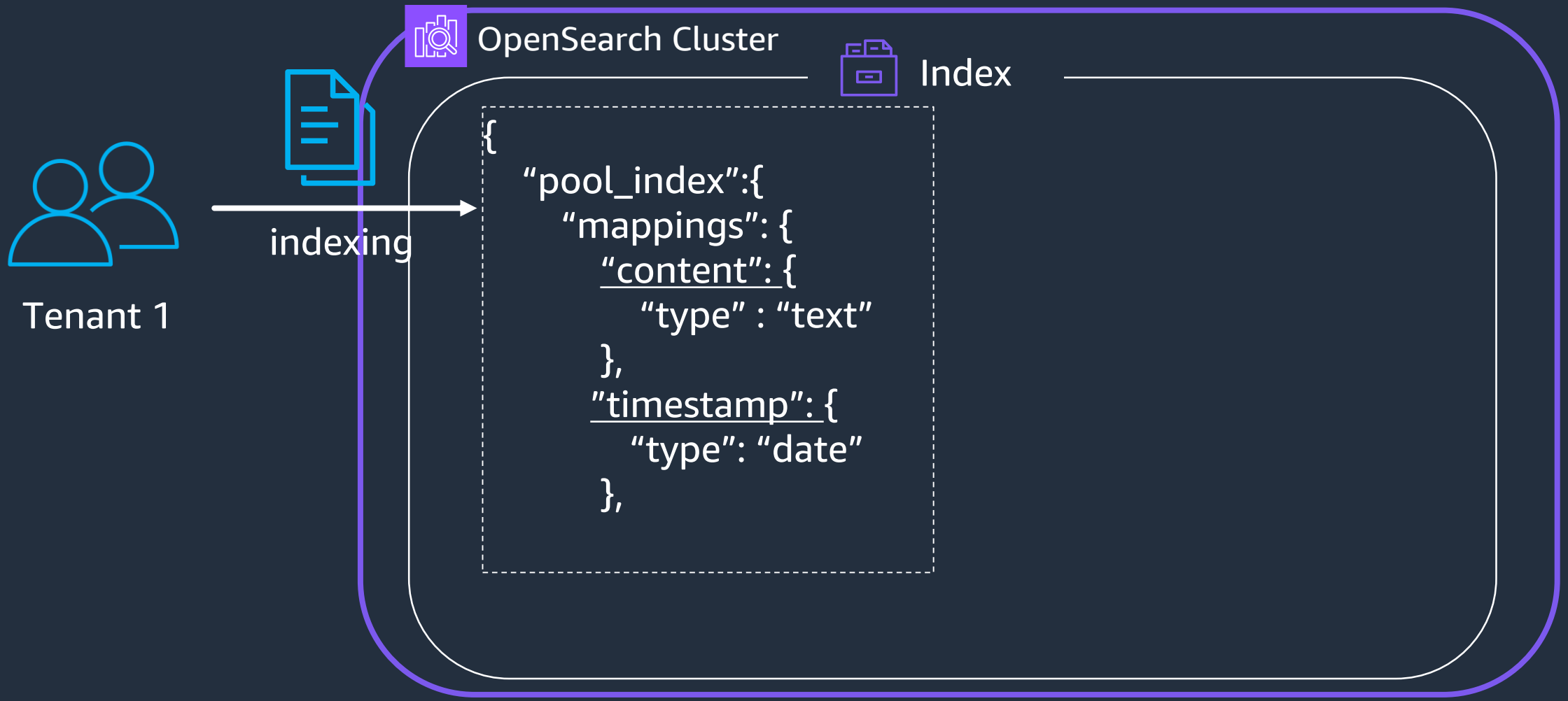
具体例1: OpenSearch で index 単位のブリッジモデルを利用する場合

- index 毎のオーバーヘッドが大きくリソースが圧迫されやすい
- 利用の小規模なテナントが多い場合でも問題になりやすい



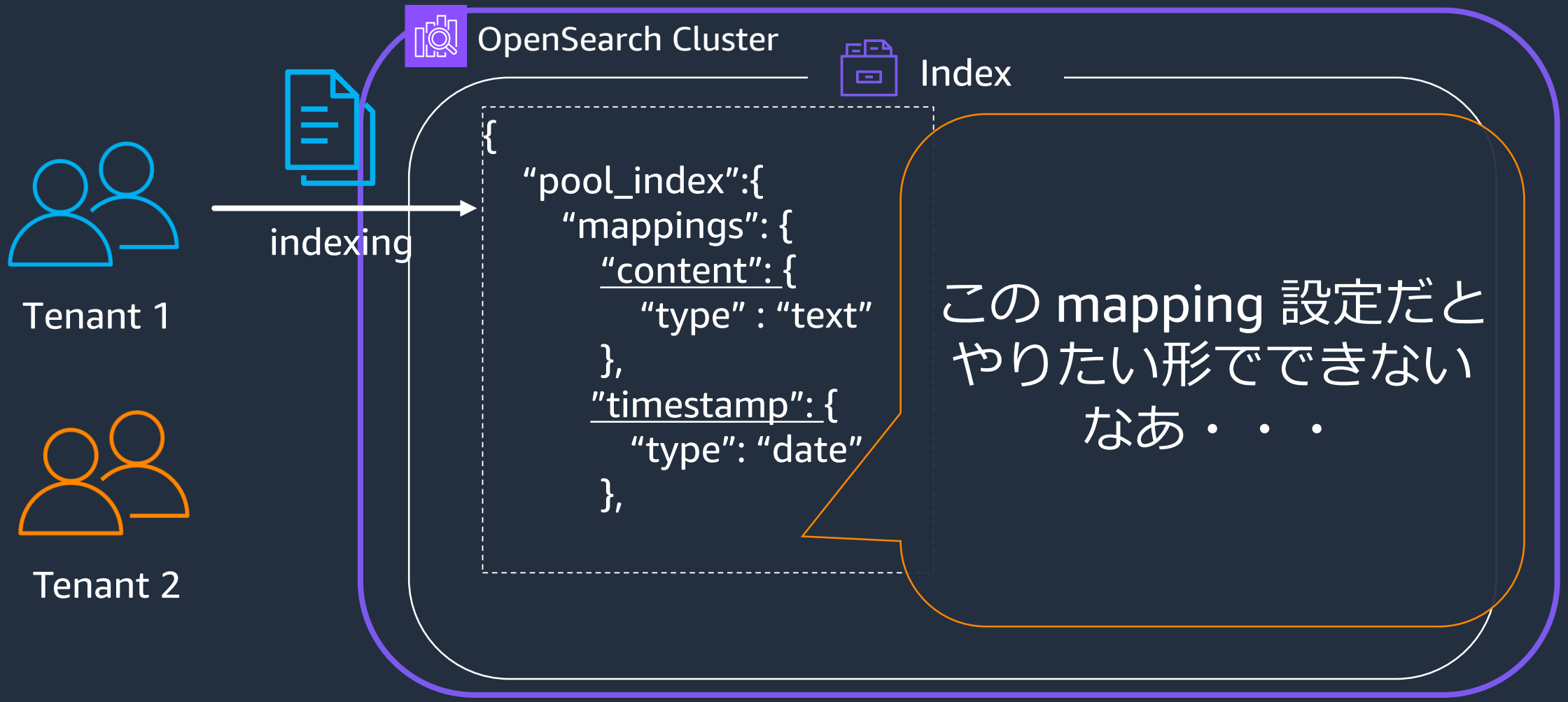
分離パターンにおけるデータストアごとの特性の考慮

具体例2：OpenSearch で プールモデルを利用する場合



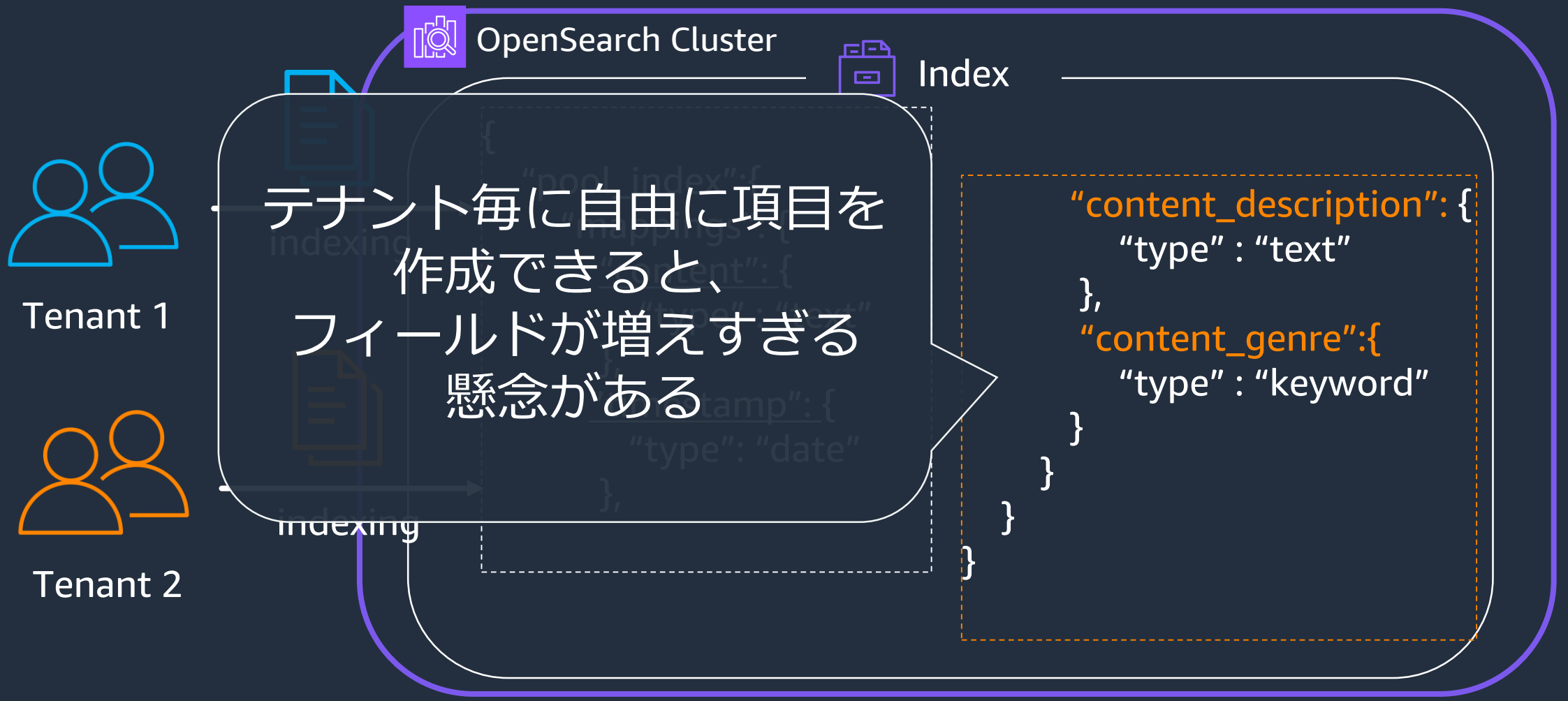
分離パターンにおけるデータストアごとの特性の考慮

具体例2：OpenSearch で プールモデルを利用する場合



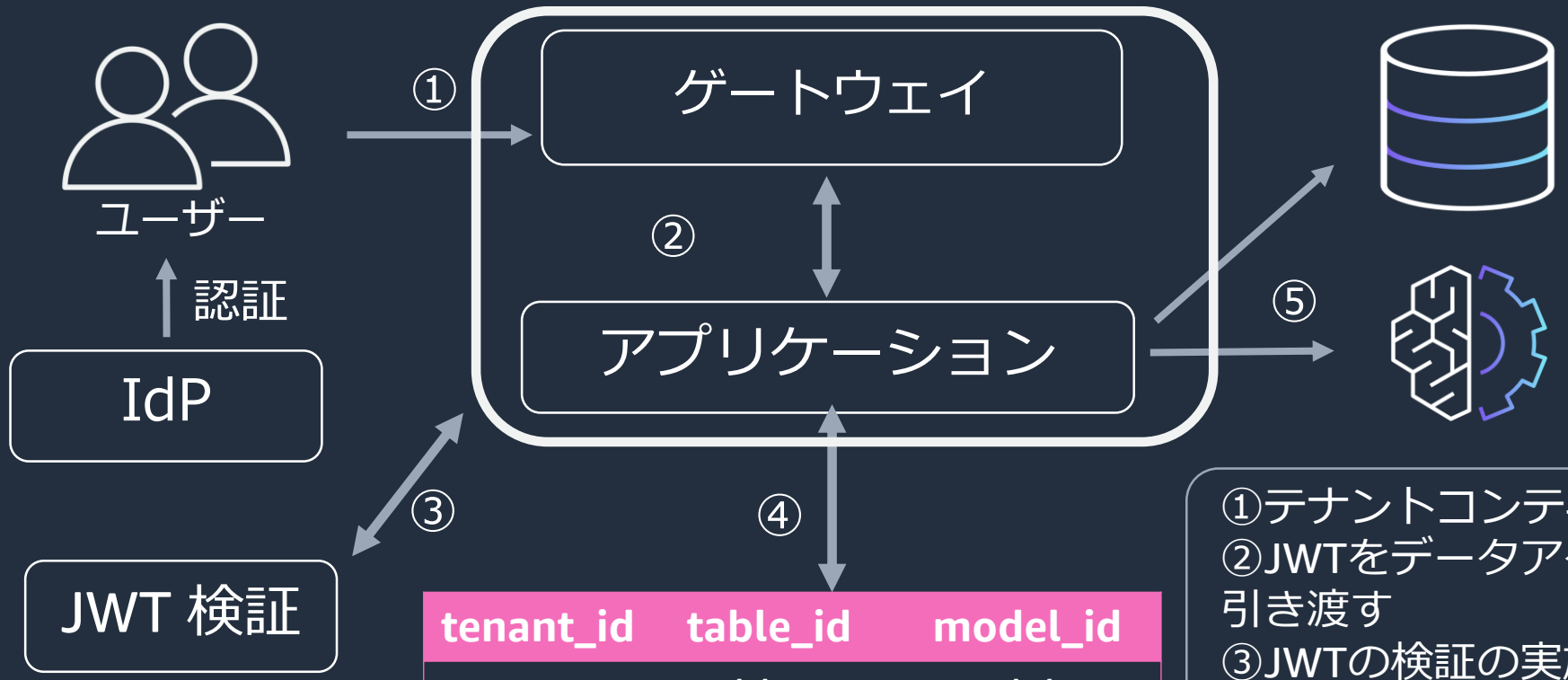
分離パターンにおけるデータストアごとの特性の考慮

具体例2：OpenSearch で プールモデルを利用する場合



認証情報を用いたテナントごとのルーティング

一般的な認証情報に基づいたルーティングの流れ

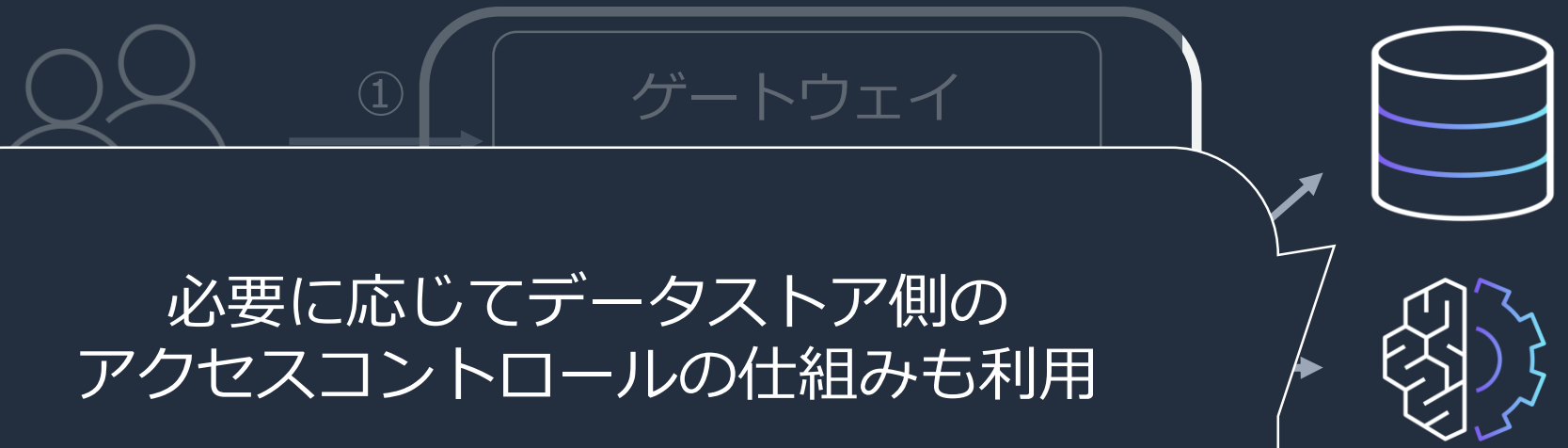


tenant_id	table_id	model_id
Tenant1	Table1	model1
Tenant2	Table2	model2
Tenant3	Table2	moedl3

①テナントコンテキストを含んだJWTを渡す
②JWTをデータアクセス管理コンポーネントに引き渡す
③JWTの検証の実施
④JWTのテナント情報と紐づいた適切なアクセス先を取得
⑤適切なデータストアやモデルにアクセス

認証情報を用いたテナントごとのルーティング

一般的な認証情報に基づいたルーティングの流れ



必要に応じてデータストア側の
アクセスコントロールの仕組みも利用

ex) PostgreSQL : Row Level Security
OpenSearch : FGAC (Fine-grained access control) の
index permissions / Document Level Security

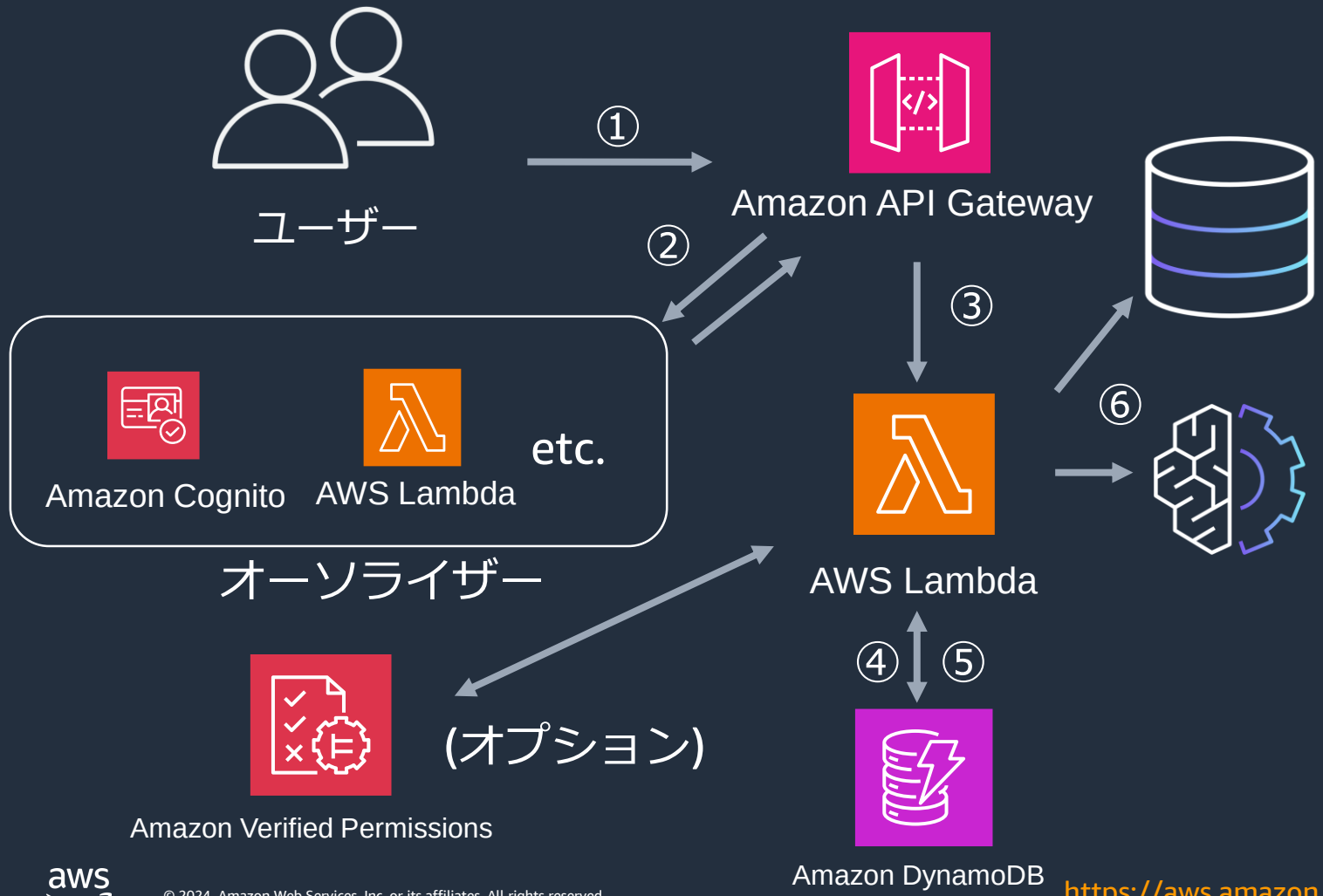
テナントコンテキストを含んだJWTを渡す
JWTをデータアクセス管理コンポーネントに
を渡す
JWTの検証の実施
JWTのテナント情報と紐づいた
適切なアクセス先を取得

Tenant2	Table2	model2
Tenant3	Table3	model3

https://aws.amazon.com/jp/blogs/news/multi-tenant-data-isolation-with-postgresql-row-level-security/
https://docs.aws.amazon.com/ja_jp/opensearch-service/latest/developerguide/fgac.html
https://opensearch.org/docs/latest/security/access-control/document-level-security/

補足：AWS での実装オプション

マネージドな仕組みを利用したテナントごとのルーティング



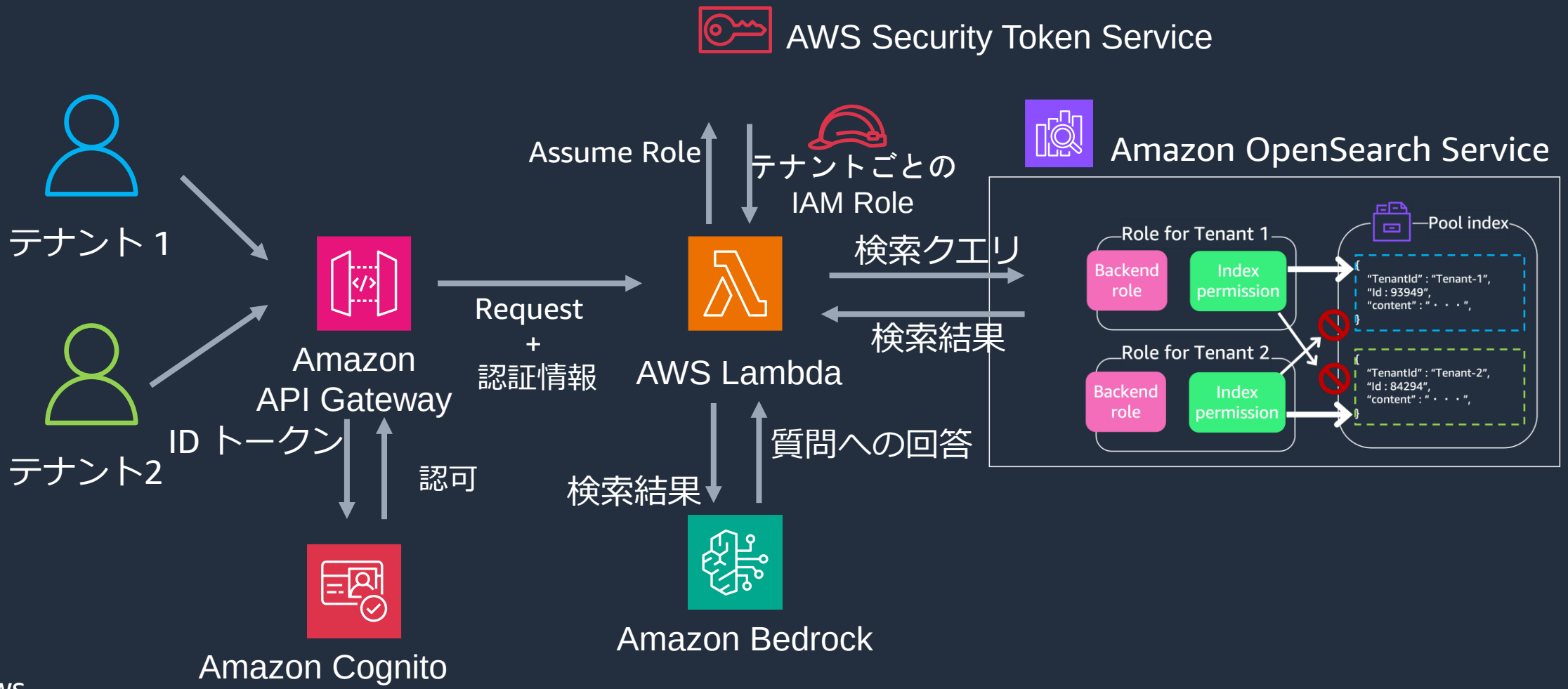
1. テナントコンテキストを含んだトークン
2. Authorizer による検証
3. トークンとリクエスト
4. テナント情報とマッピングされたアクセス先を返却
5. アクセス先のデータベース情報
6. 紐づいたデータストアやモデルにアクセス

(オプション) ユーザーのコンテキスト情報から追加の認可の仕組みを実施

https://aws.amazon.com/jp/builders-flash/202410/multi-tenant-saas-pattern/
https://www.youtube.com/watch?v=XZh56nvUodQ

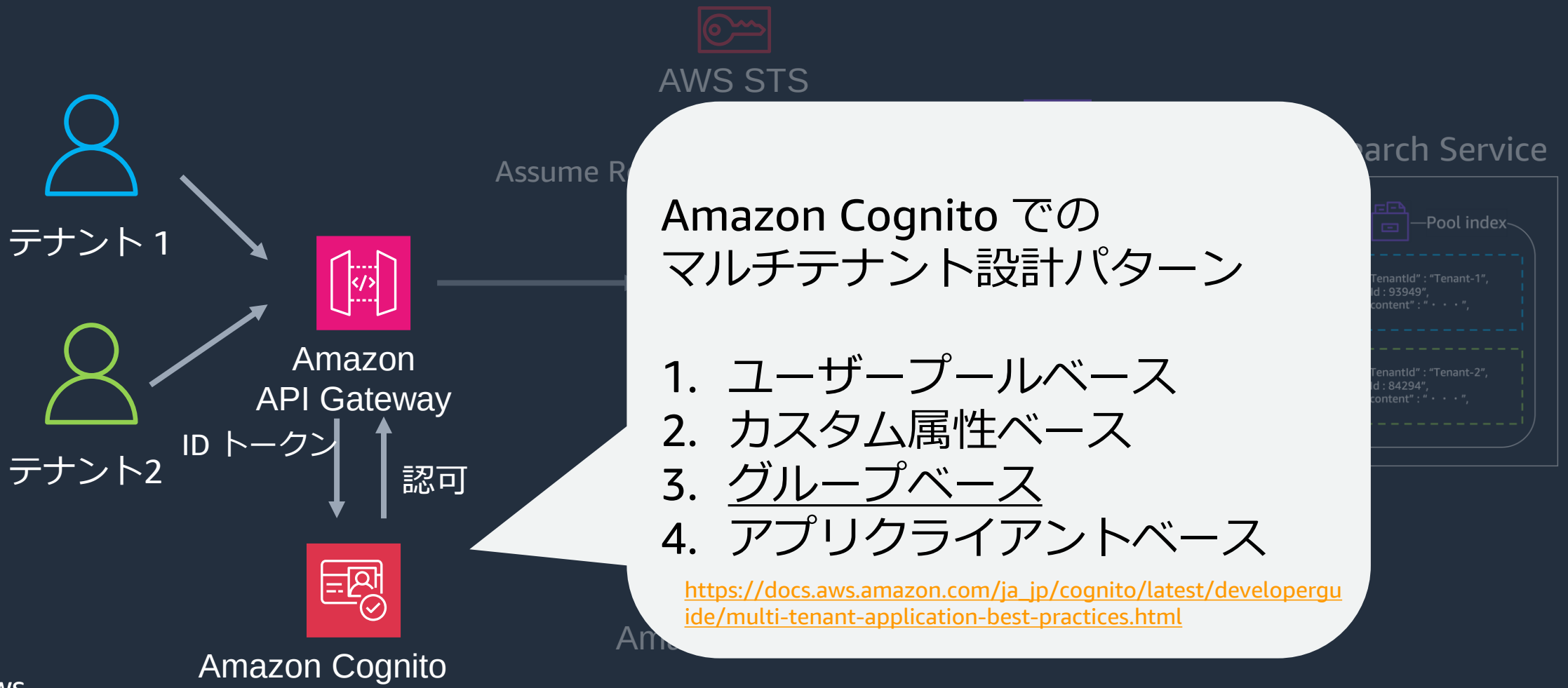
RAG におけるテナント分離の実装例

マネージドな機能を利用してアクセスコントロールの実装を容易に



RAG におけるデータのテナント分離の実装例

① Cognito を用いたマルチテナント



Amazon Cognito での
マルチテナント設計パターン

1. ユーザープールベース
2. カスタム属性ベース
3. グループベース
4. アプリクライアントベース

https://docs.aws.amazon.com/ja_jp/cognito/latest/developerguide/multi-tenant-application-best-practices.html

RAG におけるデータのテナント分離の実装例

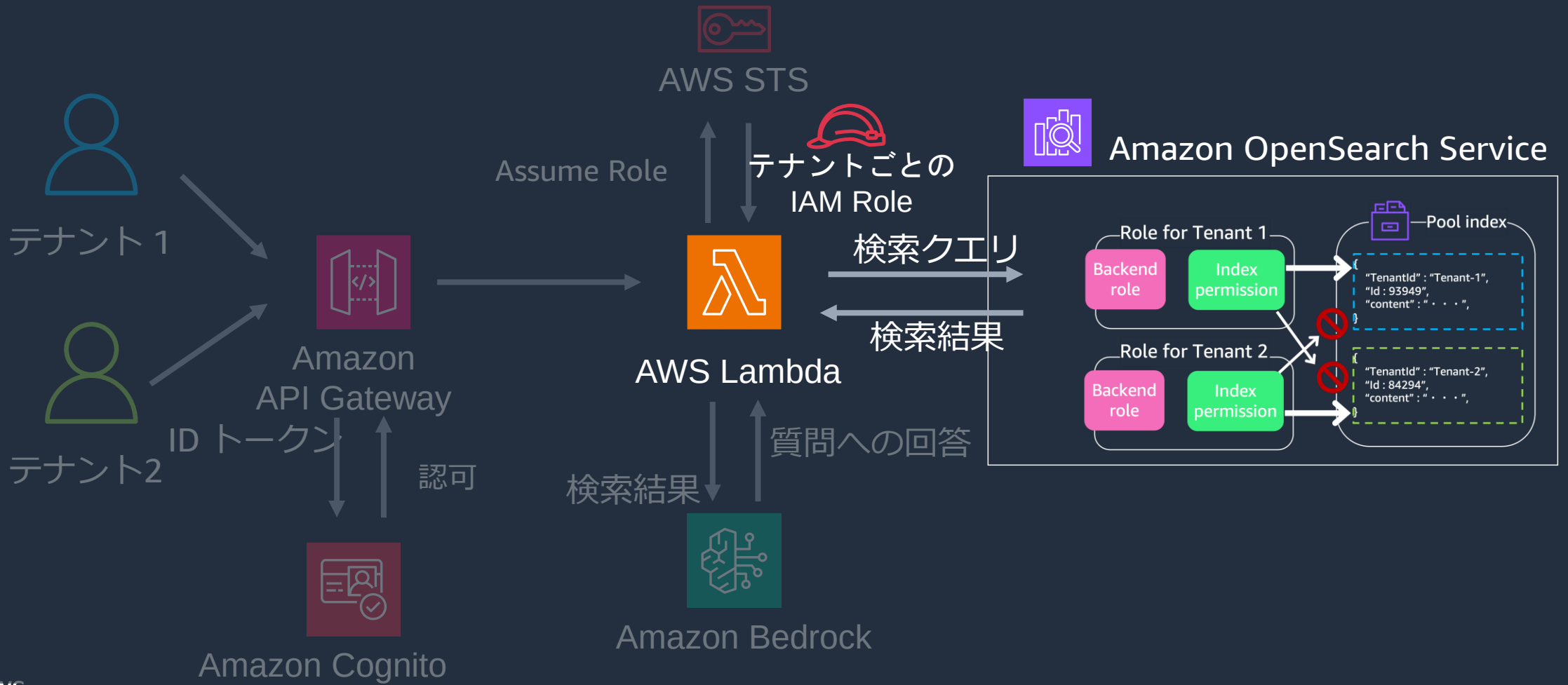
② Cognito の認証情報をもとにテナント別の Role を assume する

Authorization の中身


```
{
  "sub": "aaaaaaaa-bbbb-cccc-
dddd-eeeeeeeeeeee",
  "cognito:groups": ["tenant_1"],
  "aud": "xxxxxxxxxxxxexample",
  "email_verified": true,
  "token_use": "id",
  "auth_time": 1500009400,
  "iss": "https://cognito-idp.us-
east-1.amazonaws.com/us-east-
1_example",
  "cognito:username": "janedoe",
  "exp": 1500013000,
…
}
```


RAG におけるデータのテナント分離の実装例

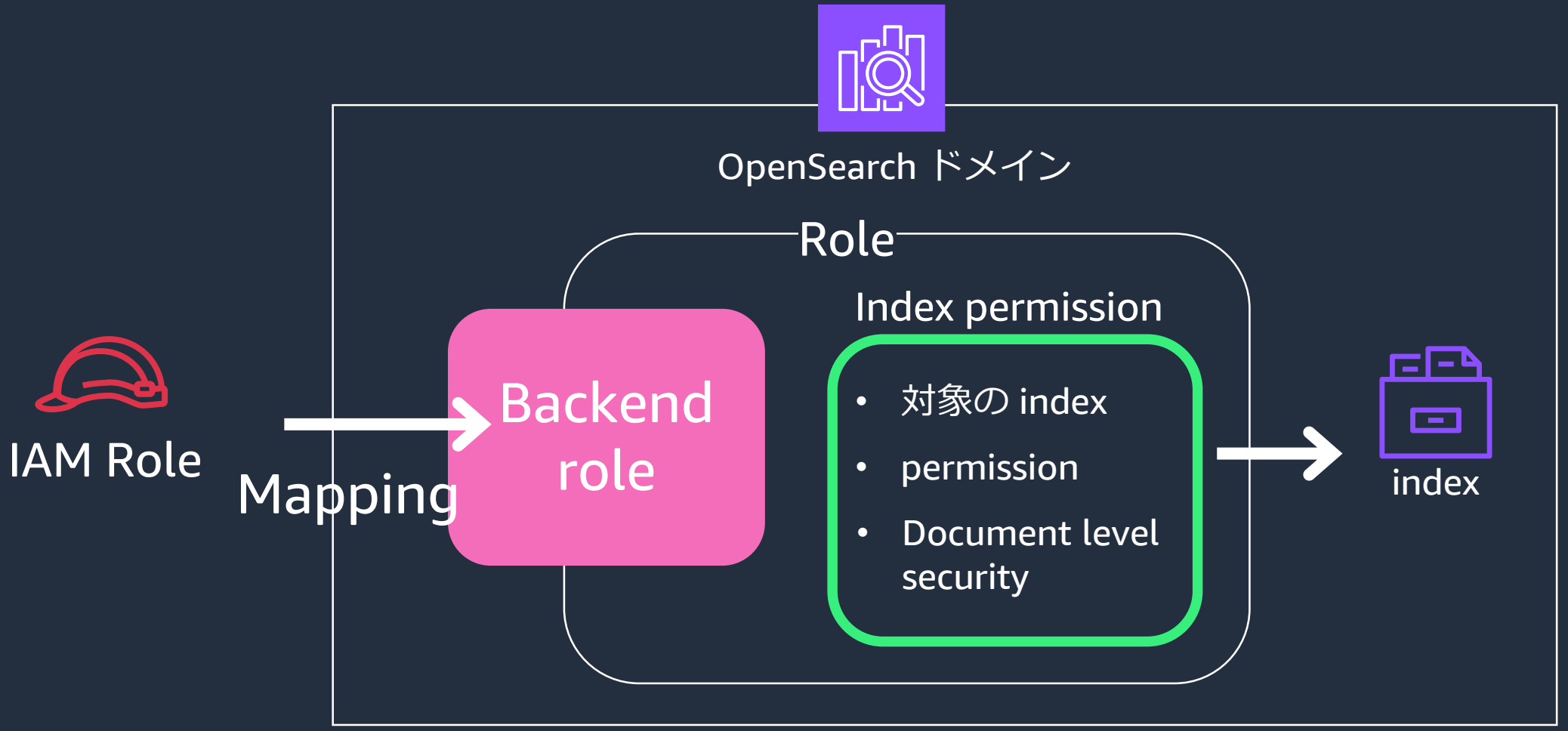
③ assume した IAM Role と 紐付けた OpenSearch のRole 権限を利用したアクセス管理



https://docs.aws.amazon.com/ja_jp/opensearch-service/latest/developerguide/fgac.html
https://opensearch.org/docs/latest/security/access-control/document-level-security/

補足：Amazon OpenSearch Service でのマルチテナンシーの実現

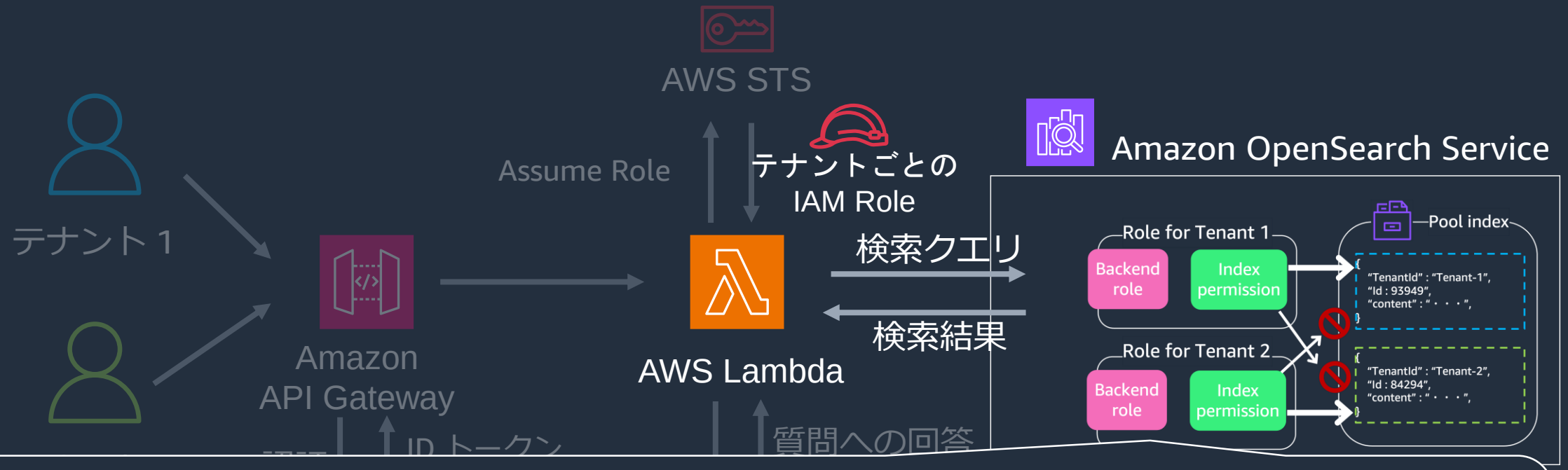
FGAC(Fine-grained access control) によるユーザー単位の詳細な権限管理



https://docs.aws.amazon.com/ja_jp/opensearch-service/latest/developerguide/fgac.html
https://opensearch.org/docs/latest/security/access-control/document-level-security/

RAG におけるデータのテナント分離の実装例

③ assume した IAM Role と 紐付けた OpenSearch のRole 権限を利用したアクセス管理

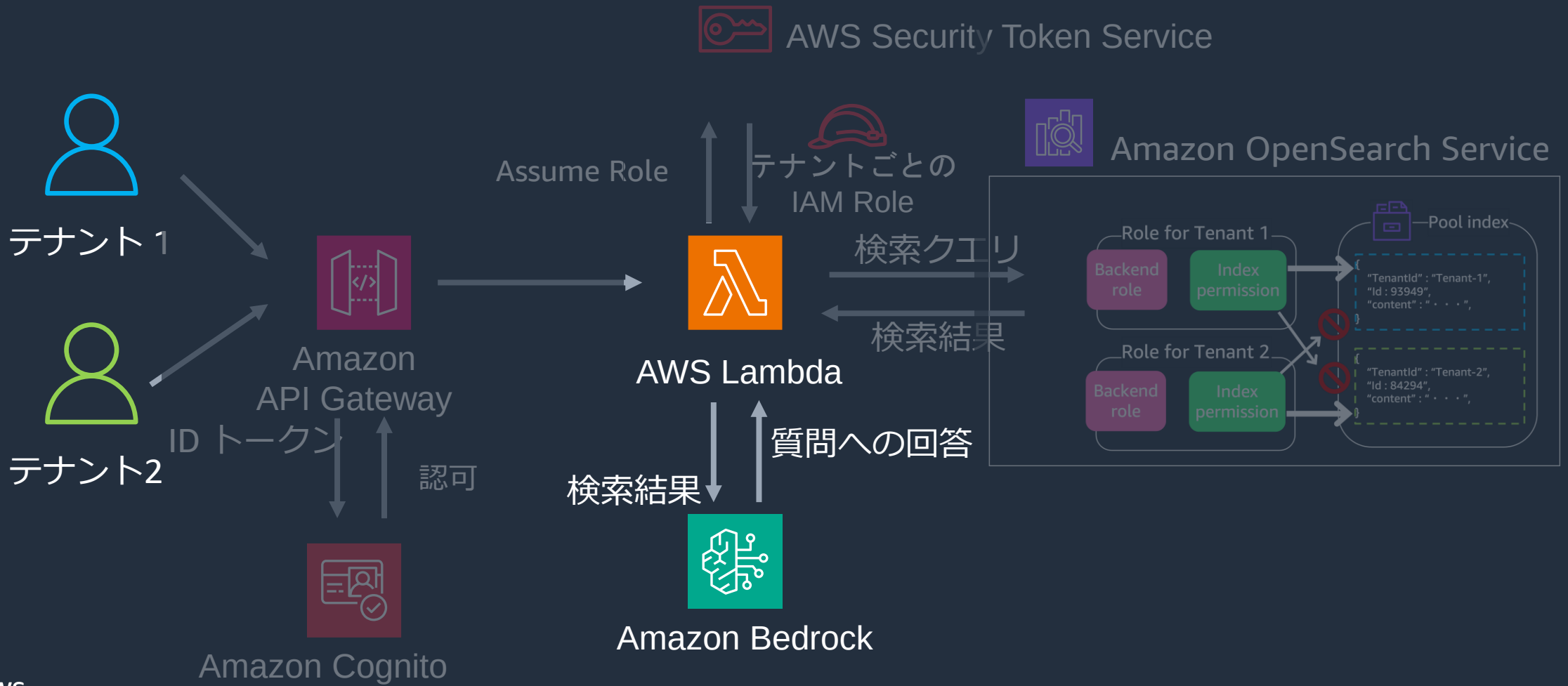


- OpenSearch Service の FGAC (Fine-grained access control) を利用して テナントごとの IAM Role と OpenSearch のロールを紐付け
- ロールの設定で該当の tenant id フィールドの値を持つドキュメントのみにアクセス制御

https://docs.aws.amazon.com/ja_jp/opensearch-service/latest/developerguide/fgac.html
https://opensearch.org/docs/latest/security/access-control/document-level-security/

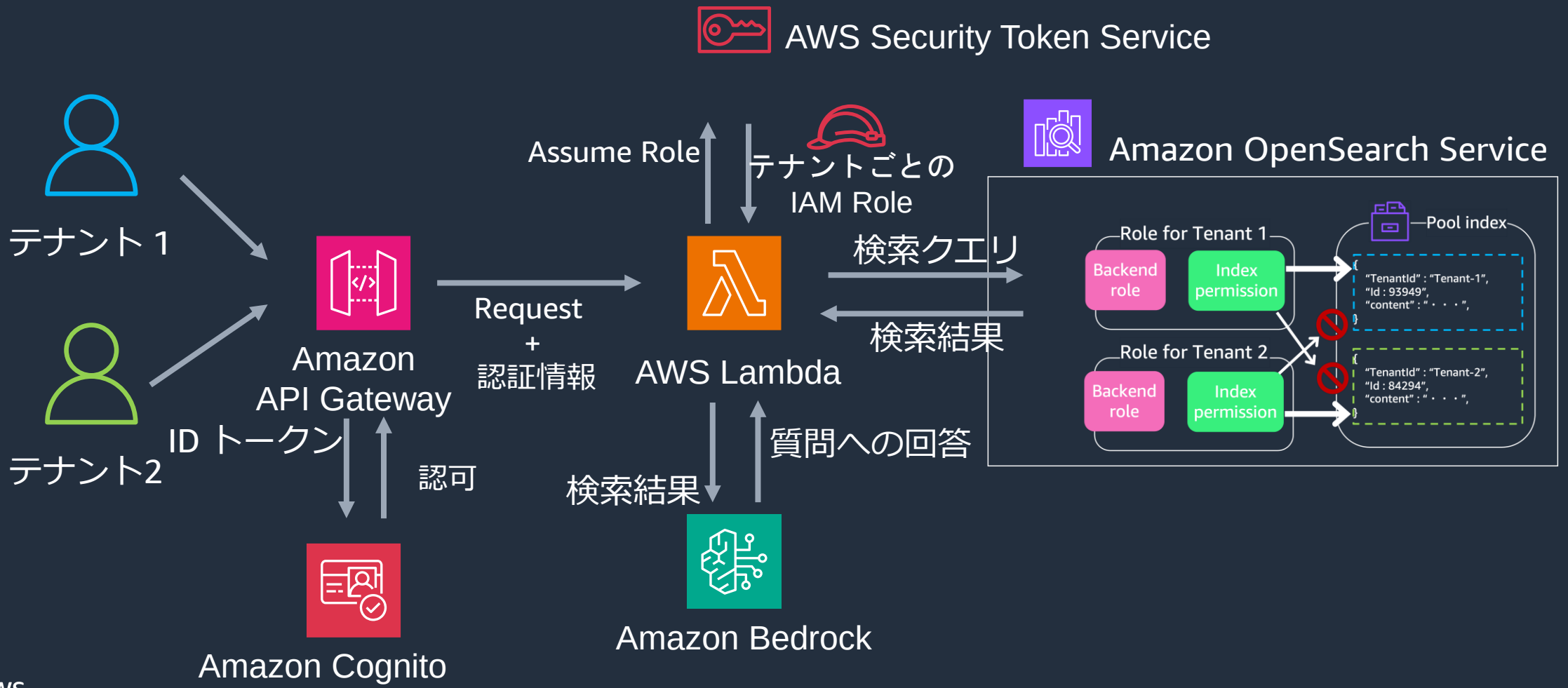
RAG におけるデータのテナント分離の実装例

④ 検索結果をもとに LLM で回答を生成する



RAG におけるデータのテナント分離の実装例

マネージドな機能を利用してアクセスコントロールの実装を容易に



Summary

RAG における検索システムの権限分離と評価

- 分離パターンはパターン毎の特性と
  データストアの特性の両方を考慮した選択が重要
- トークン内のコンテキスト情報を用いることで
  効率的なテナントごとのアクセス制御が可能
- マネージドな仕組みを用いることで
  標準仕様に従った制御の仕組みを簡単に実現

Agenda

RAG におけるテナント分離パターンと権限分離

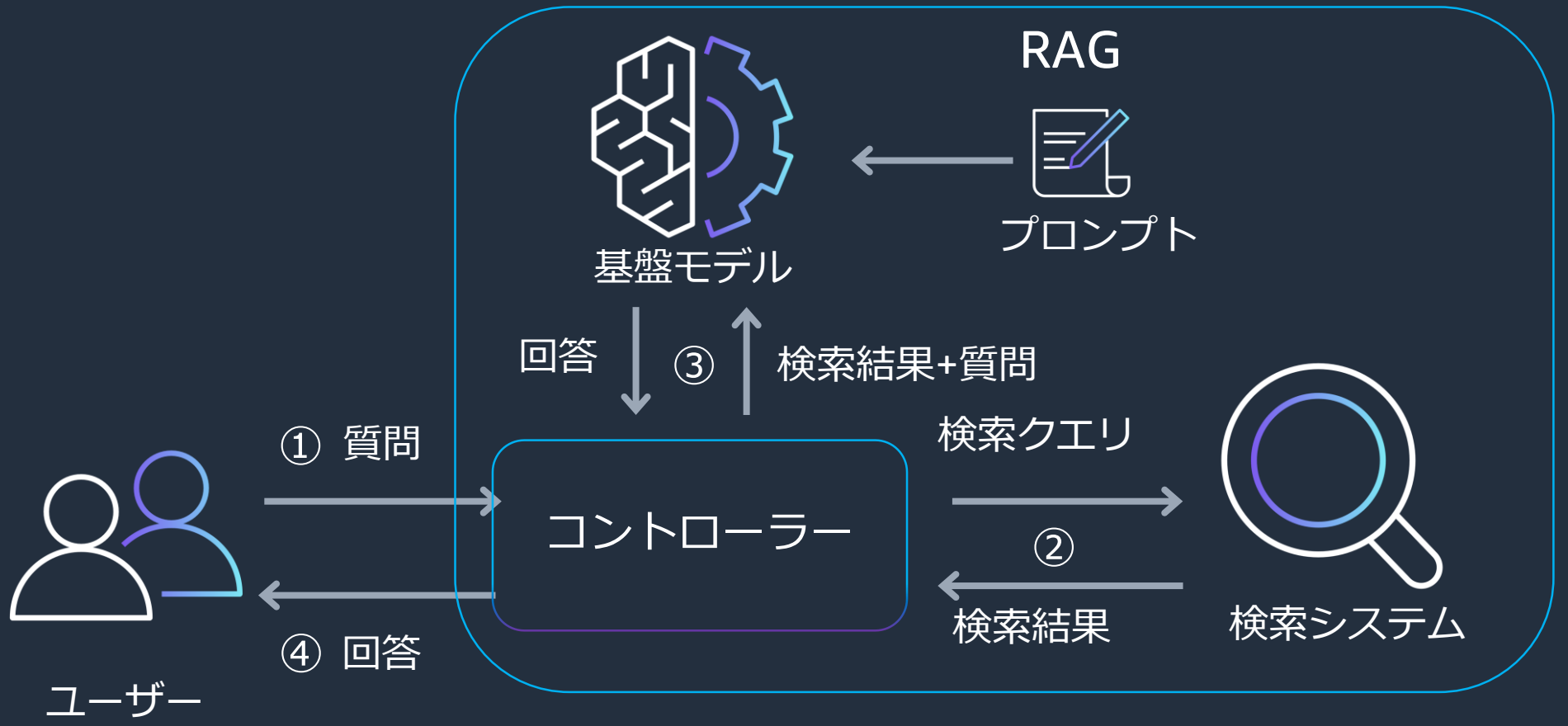
- RAG におけるマルチテナントでの分離パターン
- 分離パターンにおけるデータストアごとの特性の考慮
- 認証情報を用いたテナントごとのルーティング

RAG における検索システムの評価と改善

- RAG における検索メカニズムの重要性
- 検索システムのオフライン評価指標
- 評価と改善におけるテナント分離パターンの影響

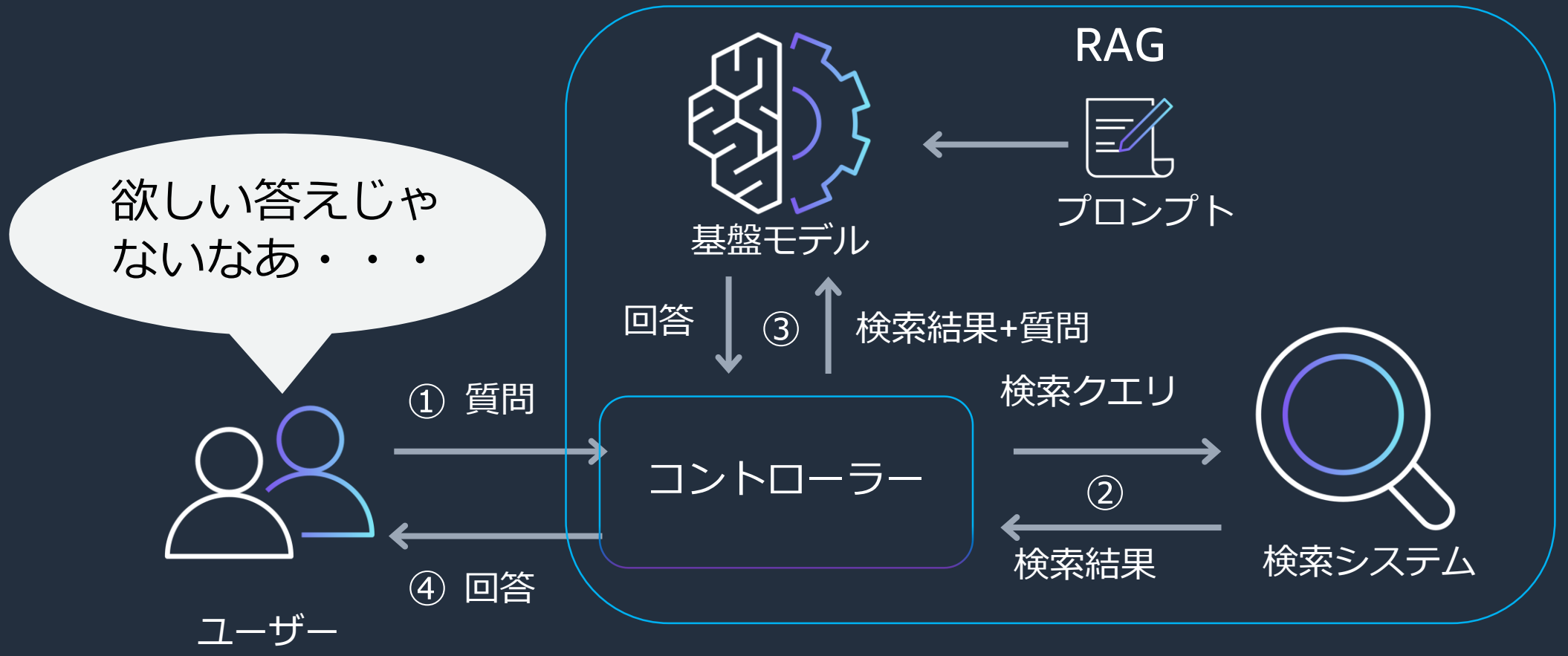
RAG のコンポーネントとその評価と改善

RAG(Retrieval-Augmented Generation) の評価の難しさ



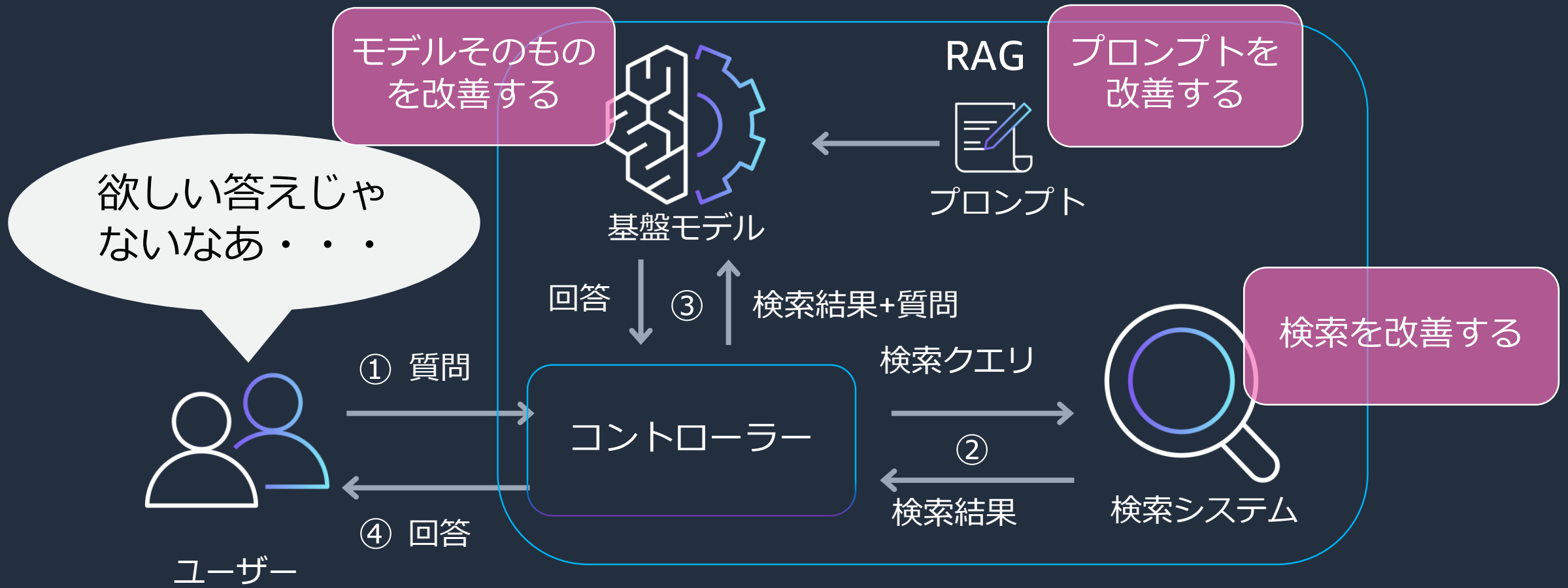
RAG のコンポーネントとその評価と改善

RAG(Retrieval-Augmented Generation) の評価の難しさ



RAG のコンポーネントとその評価と改善

RAG(Retrieval-Augmented Generation) の評価の難しさ



どの要素が結果に寄与しやすい？

RAG における検索メカニズムの重要性

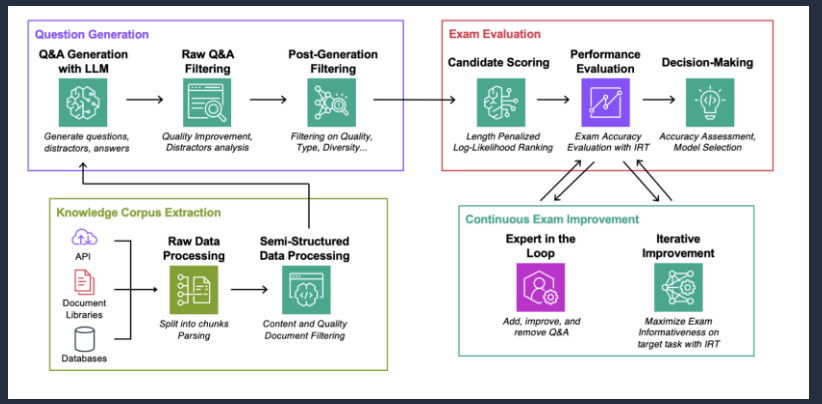
特筆すべきは、単にモデルサイズを拡大するよりも
検索メカニズムの最適化がより大きな利益をもたらす
可能性があることを発見した

適切に調整されていない検索コンポーネントは、
検索を全く行わない場合よりも精度が悪化する
可能性がある

引用元 : ICML *Automated Evaluation of Retrieval-AugmentedLanguage Models with Task-Specific Exam Generation*

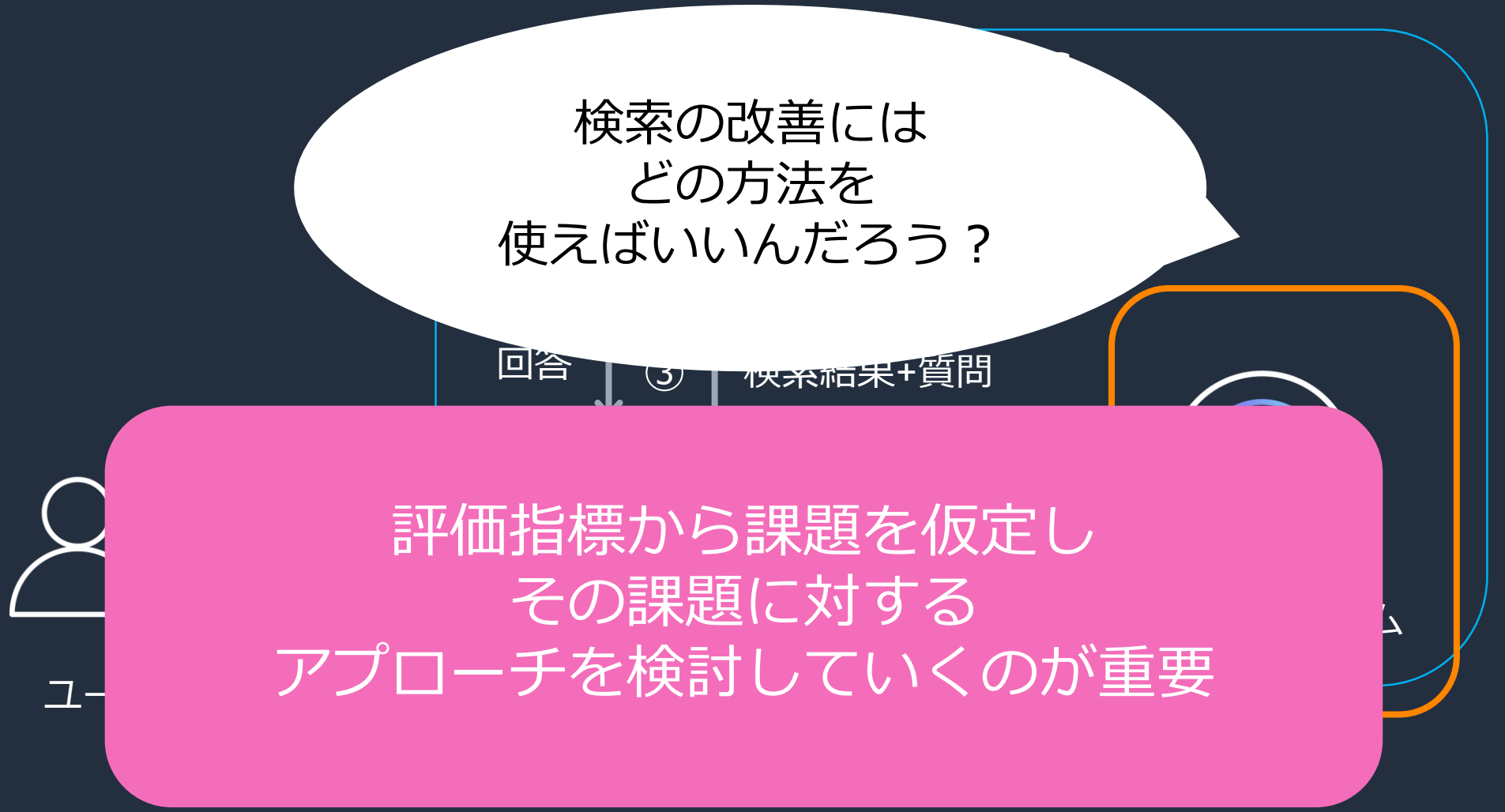
https://arxiv.org/pdf/2405.13622

- RAG の の自動評価手法を提案
- それを用いたモデル・プロンプト・
  検索メカニズムなどの影響の検証を実施



RAG のコンポーネントとその評価と改善

RAG(Retrieval-Augmented Generation) の評価の難しさ



https://aws.amazon.com/jp/blogs/news/a-practical-guide-to-improve-rag-systems-with-advanced-rag-on-aws/

検索システムのオフライン評価指標

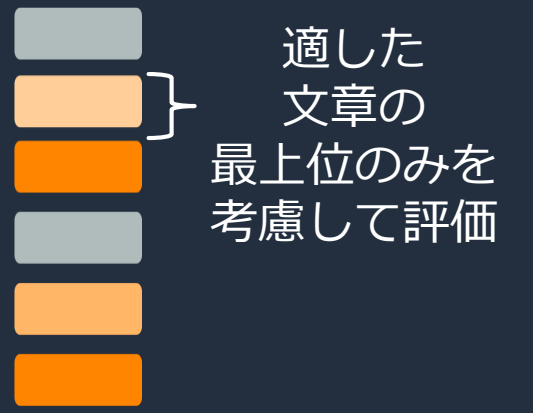
様々な評価指標

nDCG(normalized Discounted Cumulative Gain)

検索結果ランキング（並び）を考慮して、
情報要求との関連度（Gain）をもとに計算される
-> 検索結果全体を考慮した評価を行う

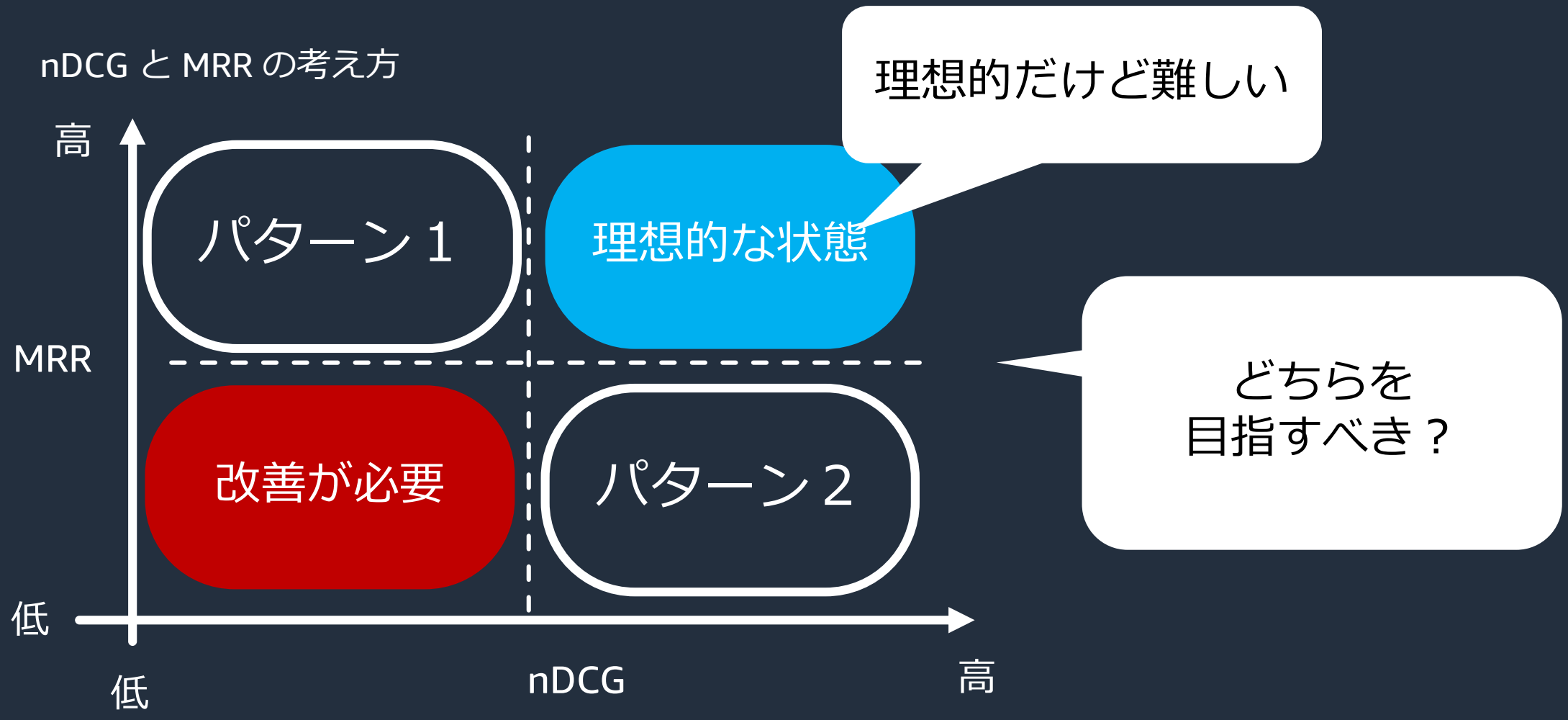
MRR(Mean Reciprocal Rank)

検索結果の並びのうち情報要求に適した
ドキュメントがはじめて現れた順位をもとに計算する
-> 検索結果の一部のみを考慮した評価を行う



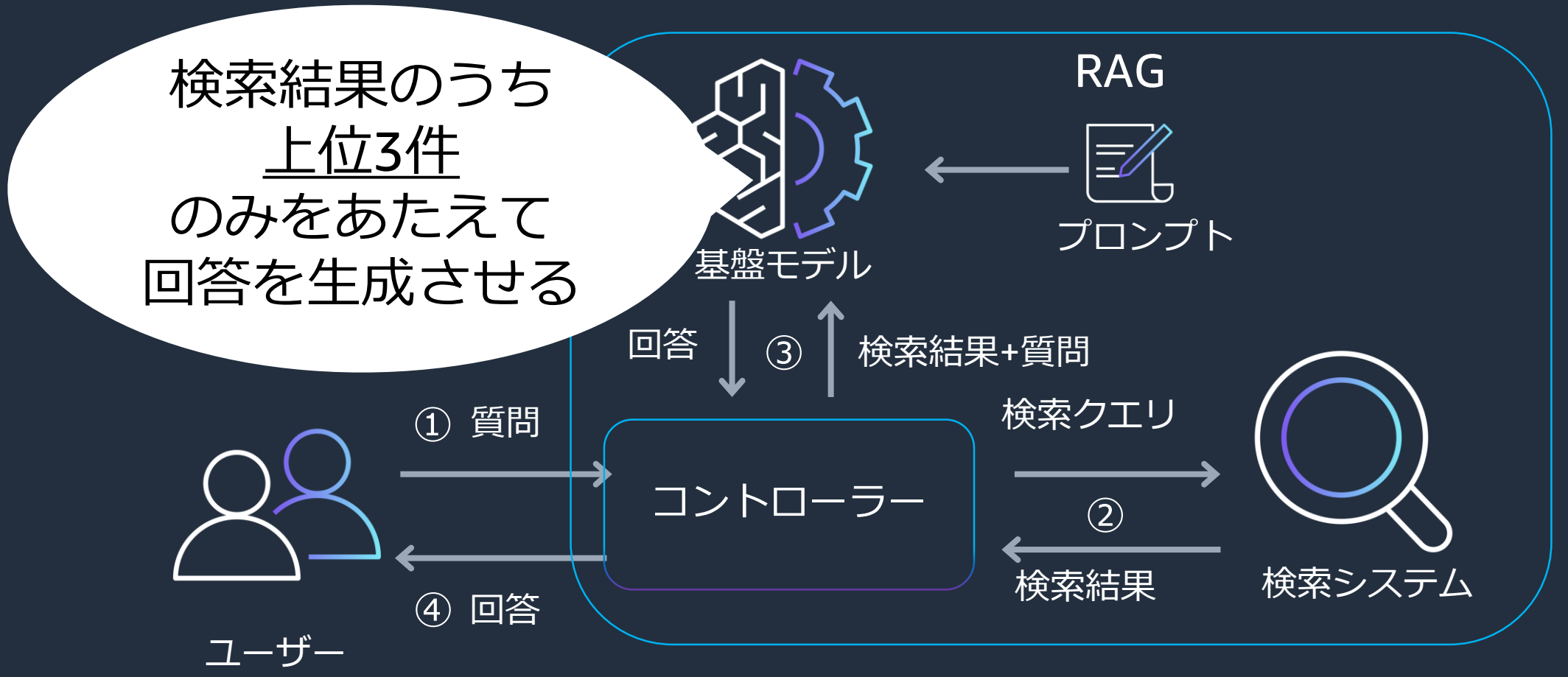
検索のオフライン評価指標

目指すべき改善の方向を決める



検索のオフライン評価指標

想定ユースケース例



検索のオフライン評価指標のパターン例

パターン1

検索結果

順位	検索１	検索２	検索３	検索４	・・・	検索５
1	○	✕	○	◎		▲
2	✕	▲	✕	○		◎
3	✕	▲	✕	▲		✕
4	✕	✕	✕	✕		✕
5	✕	✕	✕	✕		✕
6	✕	✕	✕	✕		✕
7	✕	✕	✕	✕		✕
⋮	⋮	⋮	⋮	⋮		⋮

◎ ○ ▲：関連性あり
✕：関連性なし

検索のオフライン評価指標のパターン例

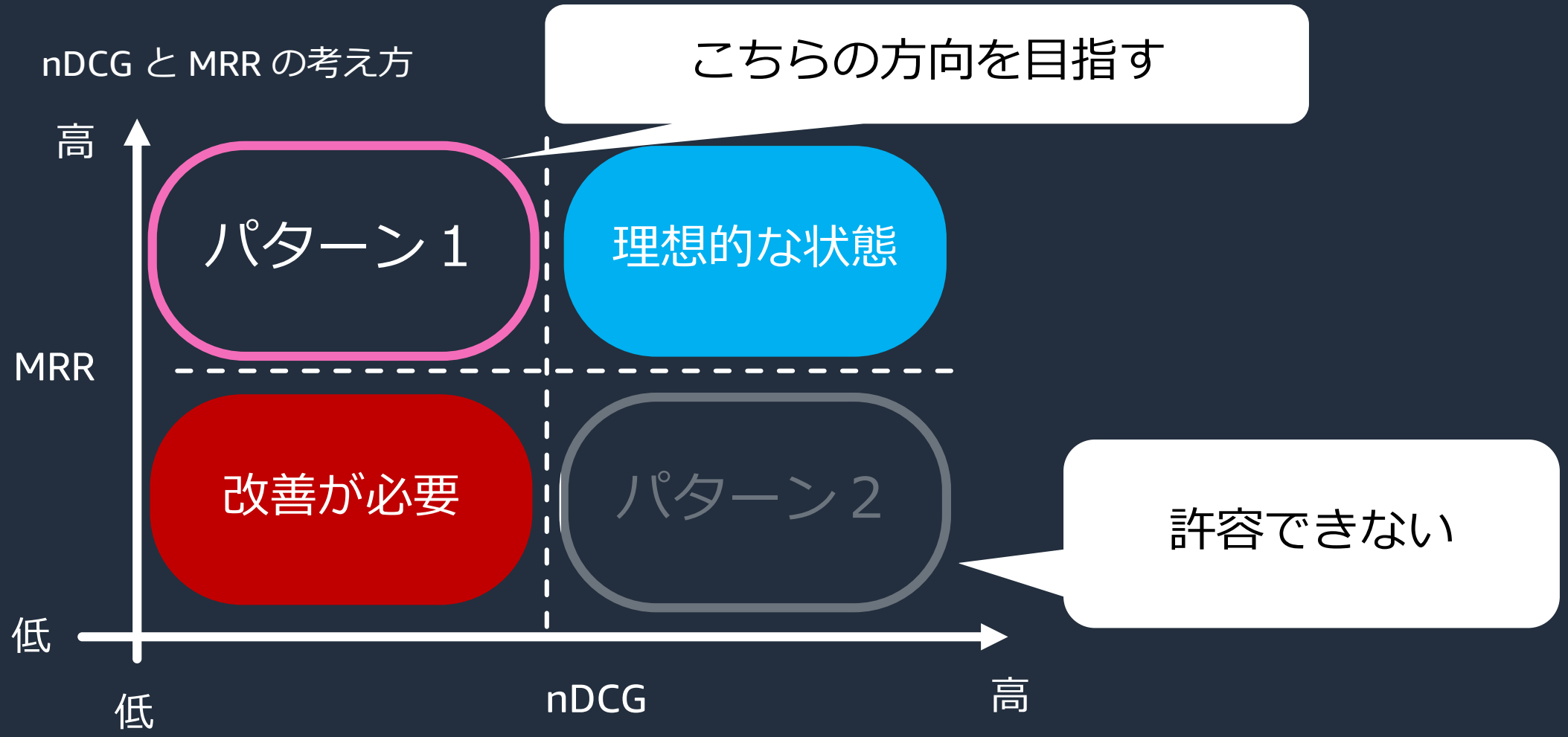
パターン２

検索結果

順位	１回目	2回目	3回目	４回目	・・・	N回目
1	✕	✕	✕	✕		✕
2	✕	✕	✕	✕		✕
3	✕	✕	✕	✕		✕
4	◎	✕	◎	◎		◎
5	◎	◎	◎	◎		◎
6	◎	◎	◎	○		◎
7	○	◎	○	○		○
⋮	⋮	⋮	⋮	⋮		⋮

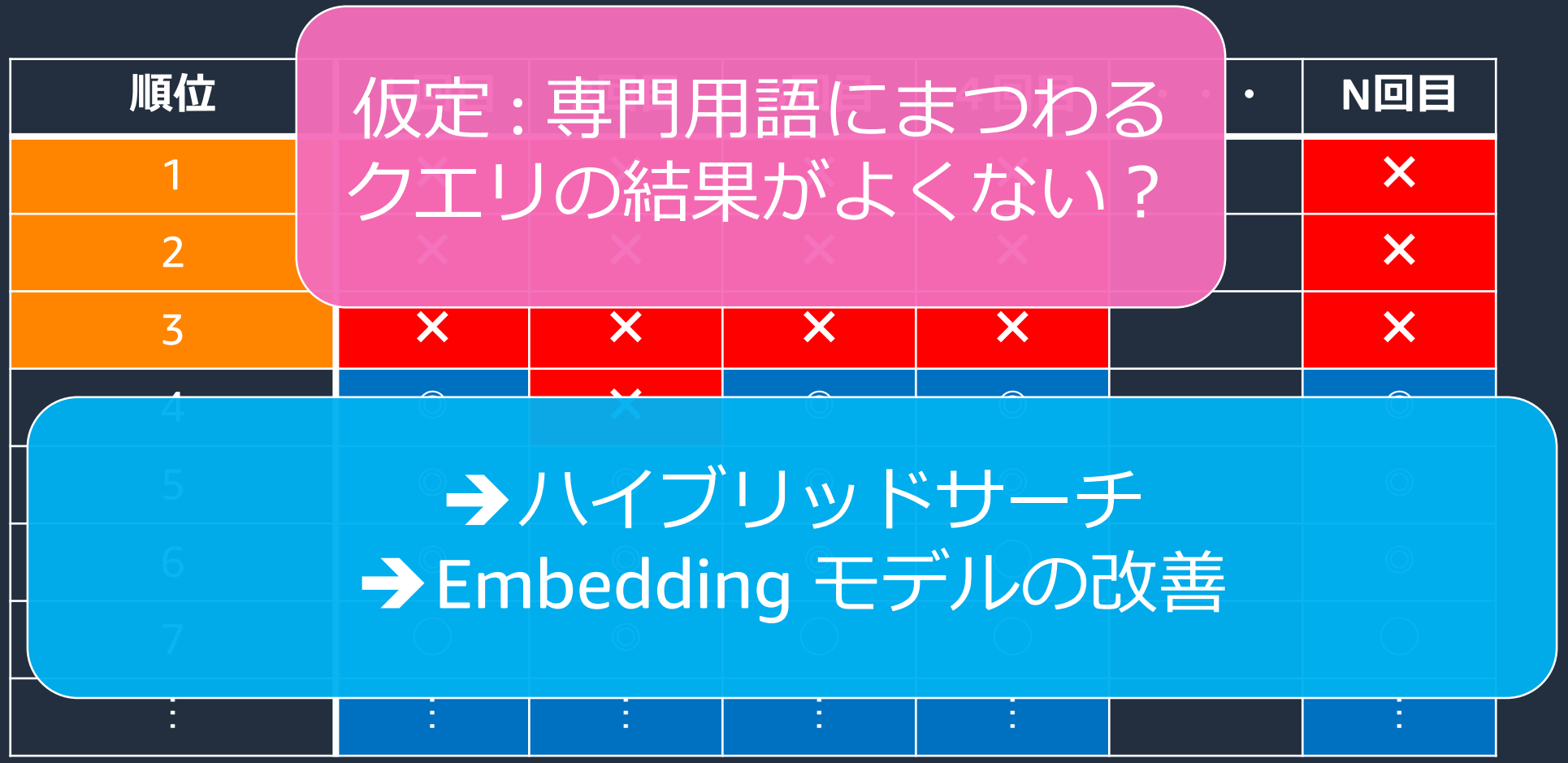
◎ ○ ▲：関連性あり
✕：関連性なし

検索のオフライン評価指標
今回のユースケース例の場合の考え方



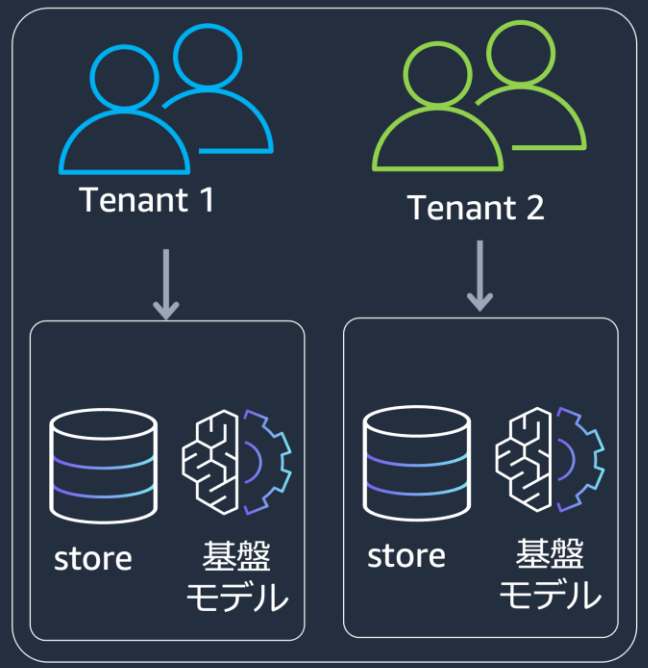
検索のオフライン評価指標

複数の指標から現在の課題を仮定して改善に取り組む

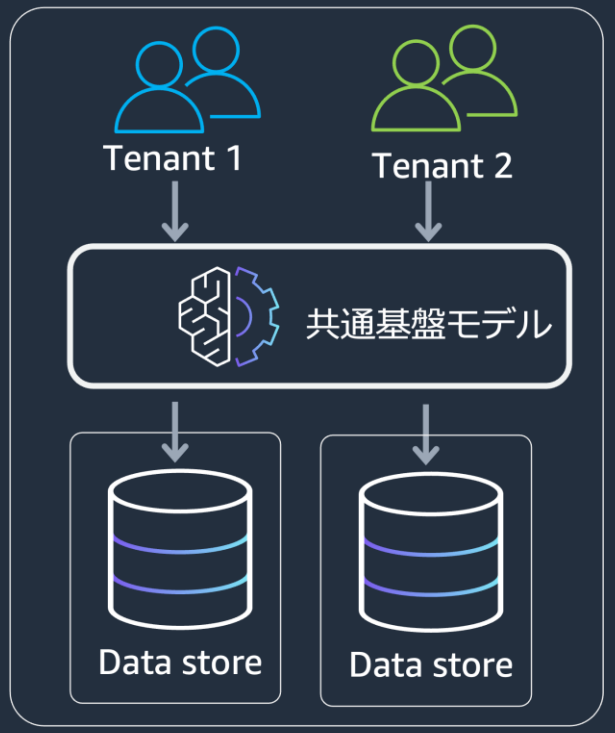


評価と改善における分離パターンの影響

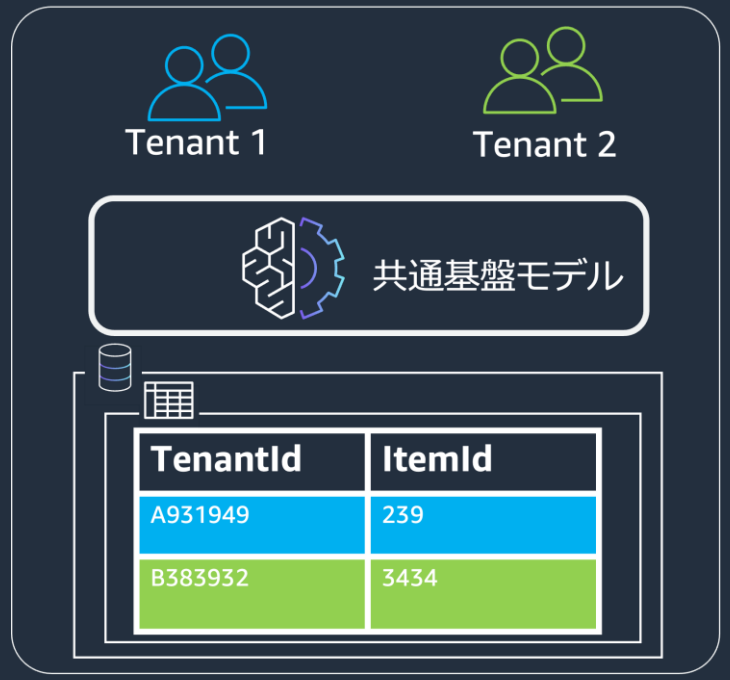
パターンによってどのような形で
評価と改善を実施できるかも変わる



サイロ
テナントごとの
評価と改善が容易



ブリッジ
テナントごとの
評価と改善が部分的



プール
テナントごとの
評価と改善が難しい

Summary

- データの分離パターンはデータストアの特性も考慮した選択が重要
- トークン内のコンテキスト情報を利用して効率的なテナントごとのアクセス制御が可能
- マネージドな仕組みを用いることで標準仕様に従った制御の仕組みを簡単に実現
- RAG のコンポーネントにおける検索メカニズムの精度は全体の精度に大きな影響を及ぼす
- ユースケースごとにどのメトリクスが重要になるかを考慮して、複数のメトリクスをもとに改善していくことが重要
- 実現したい評価と改善のサイクルも考慮に入れた分離パターンの選択

Summary

- データの分離パターンはデータストアの特性も考慮した選択が重要
- トークン内のコンテキスト情報を利用して効率的なテナントごとのアクセス制御が可能
- マネージドな仕組みを用いることで標準仕様に従った制御の仕組みを簡単に実現
- RAG のコンポーネントにおける検索メカニズムの精度は全体の精度に大きな影響を及ぼす
- ユースケースごとにどのメトリクスが重要になるかを考慮して、複数のメトリクスをもとに改善していくことが重要
- 実現したい評価と改善の仕組みも考慮に入れた分離パターンの選択

Appendix

より詳しい SaaS の認証認可のパターン: https://www.youtube.com/watch?v=XZh56nvUodQ

より複雑なアクセス制御の方法について：
https://aws.amazon.com/jp/blogs/machine-learning/design-secure-generative-ai-application-workflows-with-amazon-verified-permissions-and-amazon-bedrock-agents/

精度評価のためのツール

RAGAS：https://docs.ragas.io/en/stable/
RAGChecker：https://github.com/amazon-science/RAGChecker
Rageval：https://github.com/gomate-community/rageval

OpenSearch Rank eval API：https://opensearch.org/docs/latest/api-reference/rank-eval/
LlamaIndex Retrieval Evaluation：https://docs.llamaindex.ai/en/stable/examples/evaluation/retrieval/retrievereval/
ranx：https://github.com/amenra/ranx

Thank you!

Shuhei Fukami
Solutions Architect
AWS Japan