AWS-34

マイクロサービスアーキテクチャは SaaS 開発者を救うか

櫻谷 広人
パートナーアライアンス統括本部 ISV パートナー本部 パートナーソリューションアーキテクト
アマゾン ウェブ サービス ジャパン合同会社

自己紹介

櫻谷広人 (さくらや ひろと)

SaaS Partner Solutions Architect

アマゾン ウェブ サービス ジャパン 合同会社

- 経歴

  主にバックエンドエンジニアとして Web サービスやネイティブアプリの開発を経験。

  SIer → フリーランス → スタートアップ → AWS

- 好きな領域

  サーバーレス、マイクロサービスアーキテクチャ

本セッションの対象者

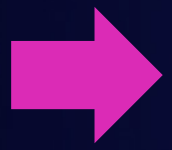
- これから新しく SaaS の開発に取り組まれる予定の方
- 現在 SaaS を本番運用されている方
  （特にスケーラビリティや信頼性について課題をお持ちの方）
- マイクロサービスアーキテクチャの導入に興味のある方

「SaaS 開発者」とは？

SaaS ビジネスに携わる全ての人
ex. アプリケーションエンジニア、インフラエンジニア、
事業開発、カスタマーサポート、etc.

本日のゴール

- マイクロサービスアーキテクチャと SaaS の親和性を理解する
- SaaS にマイクロサービスアーキテクチャを導入するにあたって設計の際に必要となる検討事項を知る



自社の要件に合わせて設計を始める！

~~全てのワークロードにマイクロサービスアーキテクチャを導入する~~

アジェンダ

- SaaS 開発におけるよくある課題
- モノリシックな SaaS の場合
- マイクロサービスな SaaS の場合
- SaaS の目線で見るサービス境界
- まとめ

SaaS 開発によくある課題

1. 多様なペルソナのサポート

ペルソナ 1：1000名規模のエンタープライズ企業

日中コンスタントにアクセスがある
多数のユーザーからの同時アクセスが多い
1回のデータ通信量は少なめ

SaaS

週末の夜にかけてアクセスが増えてくる
特定のユーザーからのみ利用される
バッチ機能の利用が多い

ペルソナ 2：十数名規模のスタートアップ

1. 多様なペルソナのサポート

ペルソナ 1：1000名規模のエンタープライズ企業

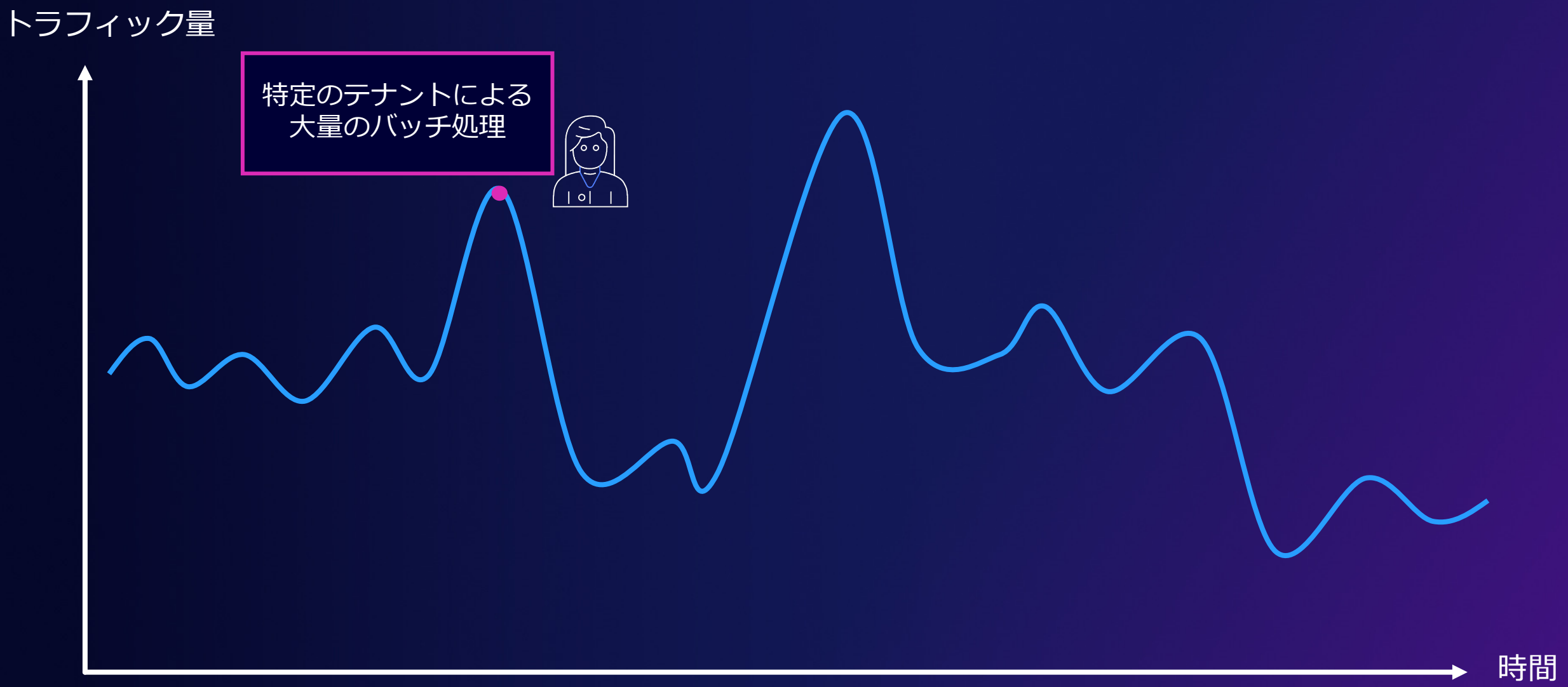
- データは他のテナントとは物理的に分離してほしい
- プライベートなネットワークでサービスを利用したい
- 可用性は 99.999 %以上が必要
- コストは惜しまない

SaaS

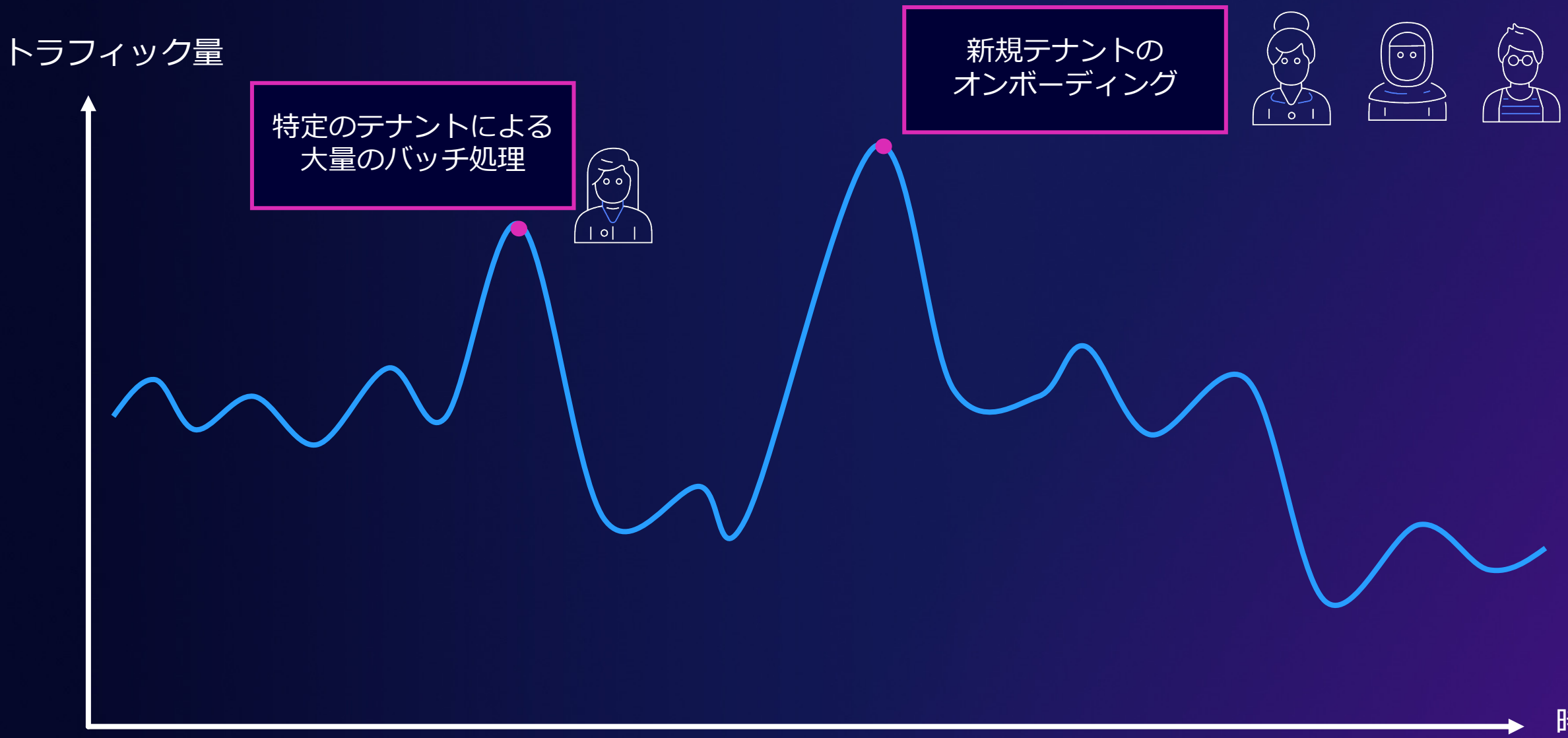
- セキュリティは最低限でOK
- 可用性は 99 %以上であればよい
- なるべくコストを抑えたい
- 一度に大量のデータをアップロードできるようにしてほしい

ペルソナ 2：十数名規模のスタートアップ

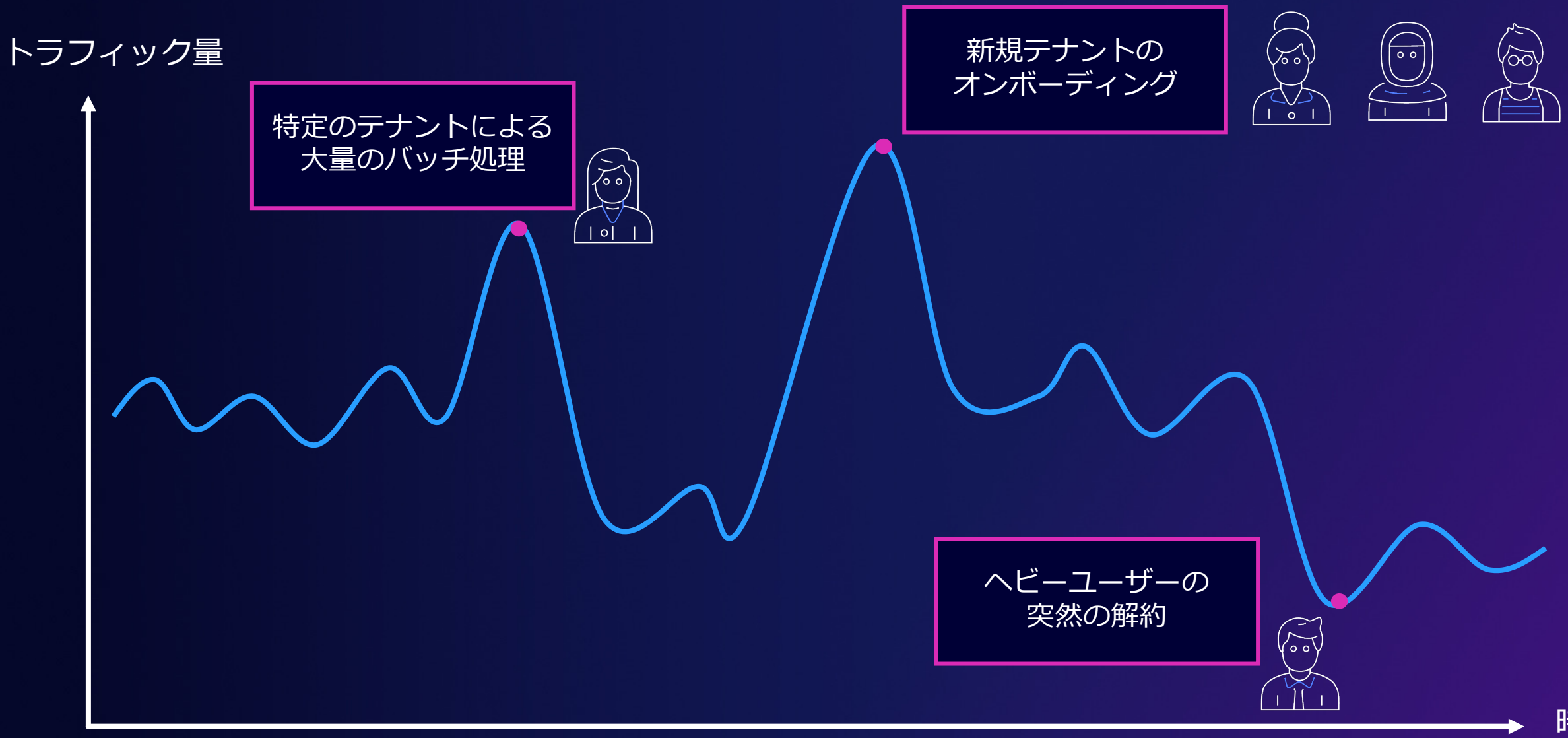
2. 予測できないトラフィック



2. 予測できないトラフィック

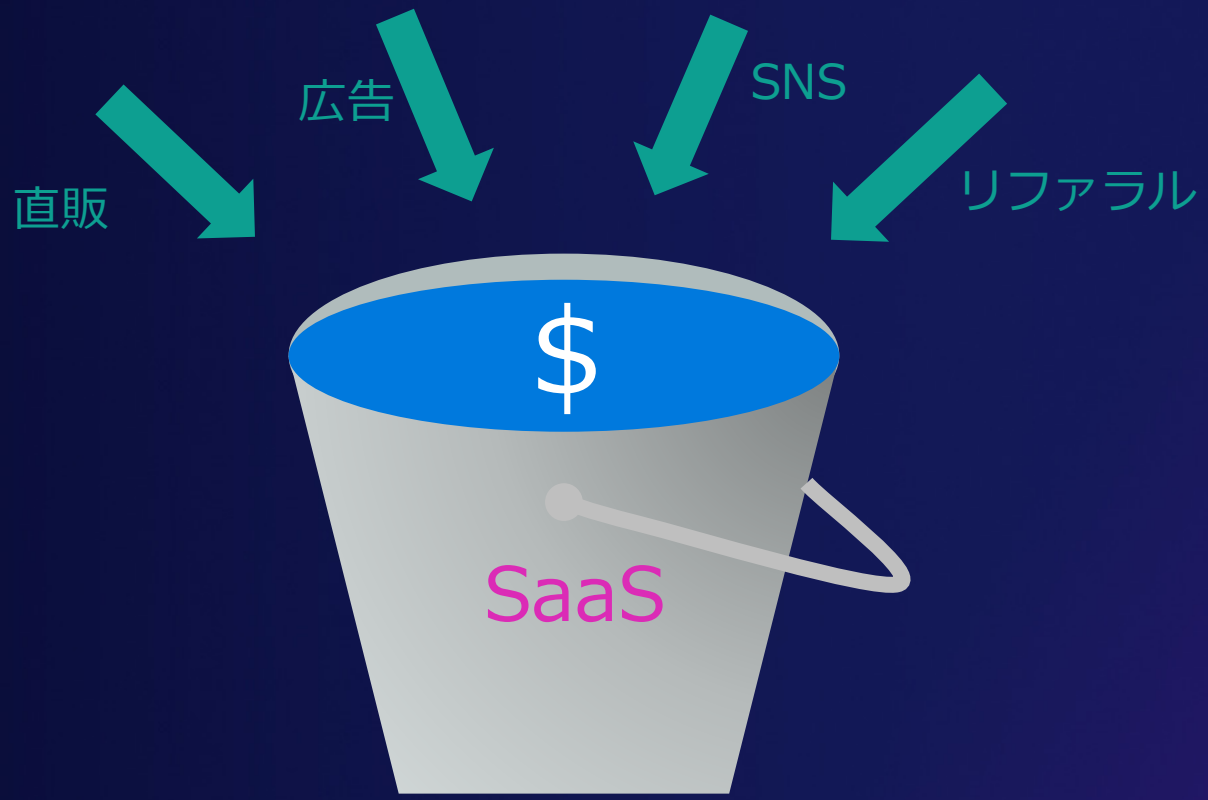


2. 予測できないトラフィック

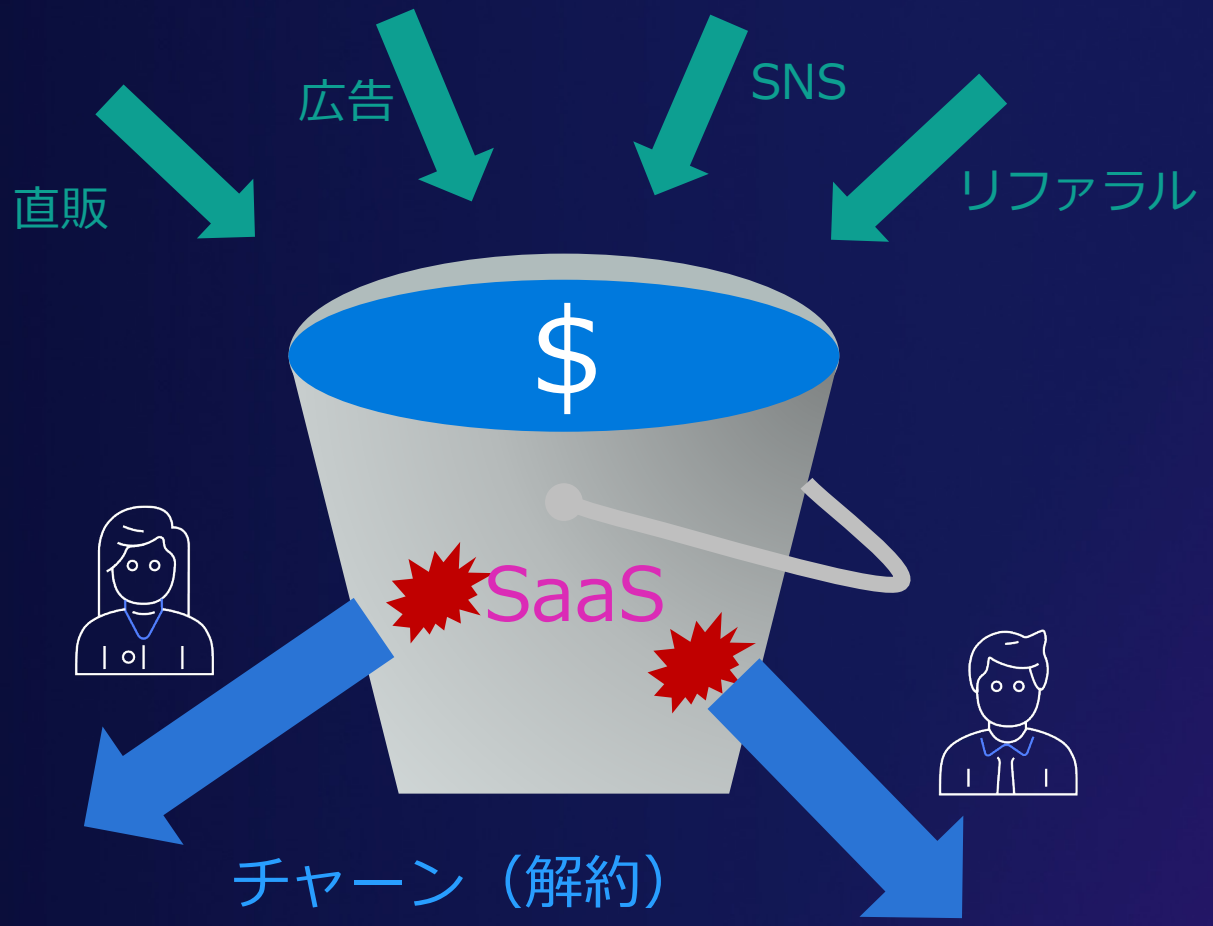


3. サービスエクスペリエンスの継続的改善

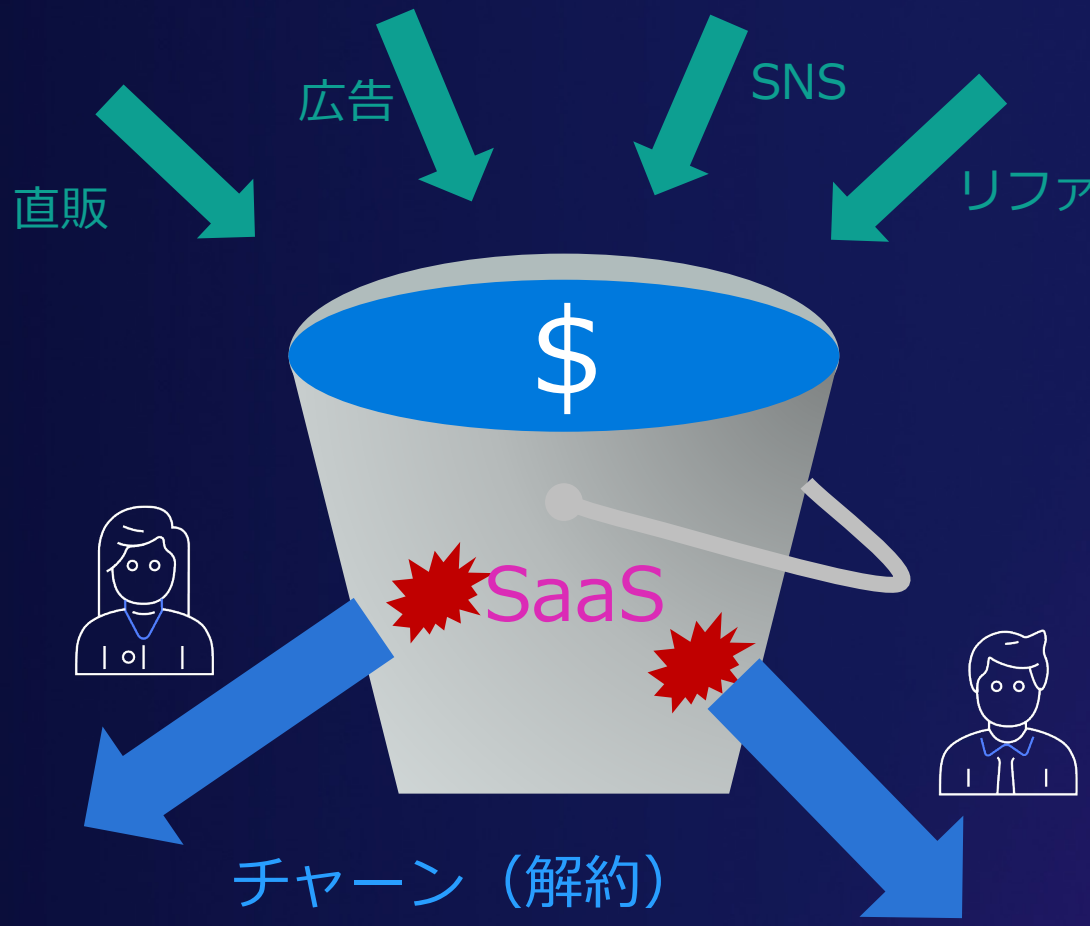
3. サービスエクスペリエンスの継続的改善



3. サービスエクスペリエンスの継続的改善



3. サービスエクスペリエンスの継続的改善



- 障害が多くて心配
- メンテナンスが長くて困る
- ページの表示が遅くてストレス
- 機能が追加されない
- サポートの返信が遅い
- 料金が高い

SaaS とは

- プロダクトではなくサービス

SaaS とは

- プロダクトではなくサービス
- どんな体験を提供できるか（期待されているか）

SaaS とは

- プロダクトではなくサービス
- どんな体験を提供できるか（期待されているか）
- リリースしたら終わりではない。運用し続けなければならない

SaaS とは

- プロダクトではなくサービス
- どんな体験を提供できるか（期待されているか）
- リリースしたら終わりではない。運用し続けなければならない
- 契約が取れたら終わりではない。サポートし続けなければならない

SaaS とは

- プロダクトではなくサービス
- どんな体験を提供できるか（期待されているか）
- リリースしたら終わりではない。運用し続けなければならない
- 契約が取れたら終わりではない。サポートし続けなければならない
- 市場、顧客のビジネスの変化に適応し続けなければならない

SaaS とは

- プロダクトではなくサービス
- どんな体験を提供できるか（期待されているか）
- リリースしたら終わりではない。運用し続けなければならない
- 契約が取れたら終わりではない。サポートし続けなければならない
- 市場、顧客のビジネスの変化に適応し続けなければならない

現状維持は SaaS にとって衰退を意味する

とはいえ開発運用は大変なことだらけ。。。

どうすれば継続的に体験を
改善し続けられるだろうか 🤔

とはいえ開発運用は大変なことだらけ。。。

どうすれば継続的に安全に体験を
改善し続けられるだろうか 🤔

とはいえ開発運用は大変なことだらけ。。。

どうすれば継続的に安全に
スケーラブルに体験を
改善し続けられるだろうか 🤔

とはいえ開発運用は大変なことだらけ。。。

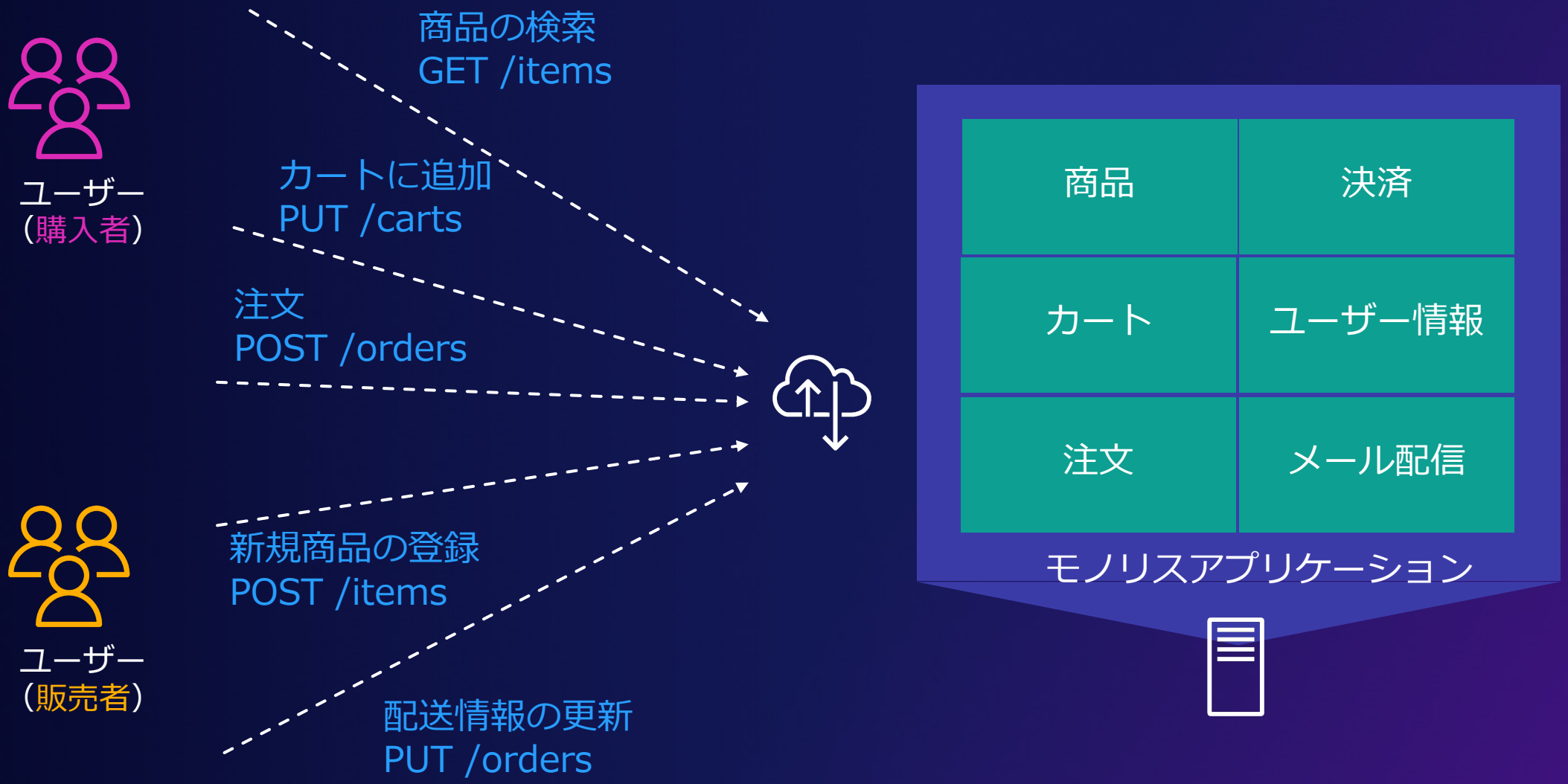
どうすれば継続的に安全に
スケーラブルに運用負荷をかけず
体験を改善し続けられるだろうか 🤔

とはいえ開発運用は大変なことだらけ。。。

どうすれば継続的に安全に
スケーラブルに運用負荷をかけず
ユーザーが望む体験に
改善し続けられるだろうか 🤔

モノリシックな SaaS の場合

モノリシックな EC サイト



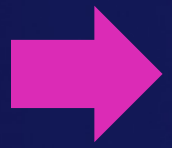
モノリスにおける課題

- 非効率なスケーリング
- 関係者間調整のオーバーヘッド
- 変更による影響範囲の広さ
- テスト/ビルドに要する時間の長さ
- 利用可能な技術の制限

etc.

モノリスにおける課題

- 非効率なスケーリング
- 関係者間調整のオーバーヘッド
- 変更による影響範囲の広さ
- テスト/ビルドに要する時間の長さ
- 利用可能な技術の制限



どうすれば継続的に安全に
スケーラブルに運用負荷をかけず
ユーザーが望む体験に
改善し続けられるだろうか 🤔

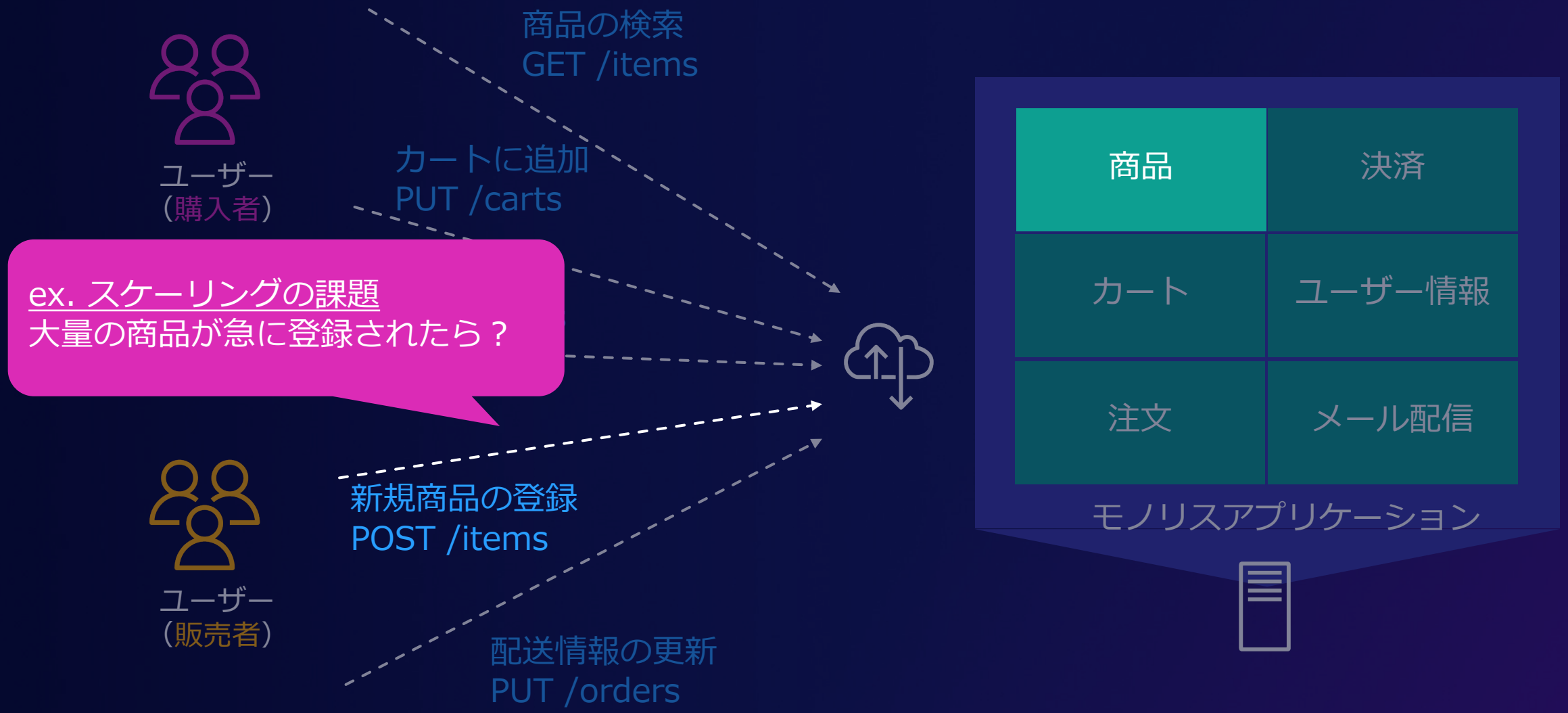
etc.

モノリスにおける課題

- 非効率なスケーリング
- 関係者間調整のオーバーヘッド
- 変更による影響範囲の広さ
- テスト/ビルドに要する時間の長さ
- 利用可能な技術の制限

etc.

モノリシックな EC サイト



モノリシックな EC サイト

ex. 開発効率の課題
コード変更の影響範囲は？
1回のビルド/テストにどれくらいかかる？
デプロイが競合してしまった場合は？

ユーザー
（購入者）

カートに追加
PUT /carts

注文
POST /orders

ユーザー
（販売者）

新規商品の登録
POST /items

配送情報の更新
PUT /orders

モノリシックな EC サイト

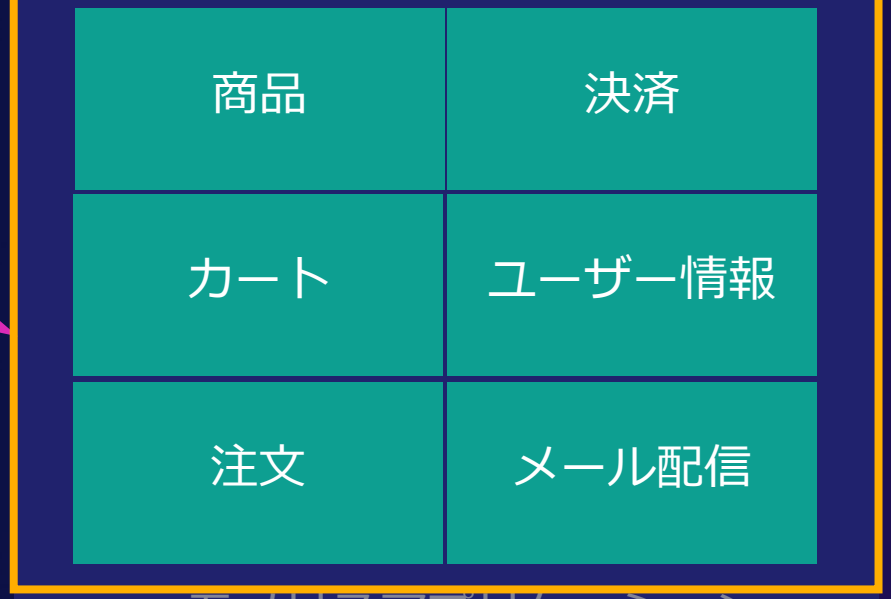
商品の検索
GET /items

ex. 技術選定の課題
同じ言語/フレームワークで開発できる？
新しい技術 (ex. コンテナ) を試したいときは？

ユーザー
（販売者）

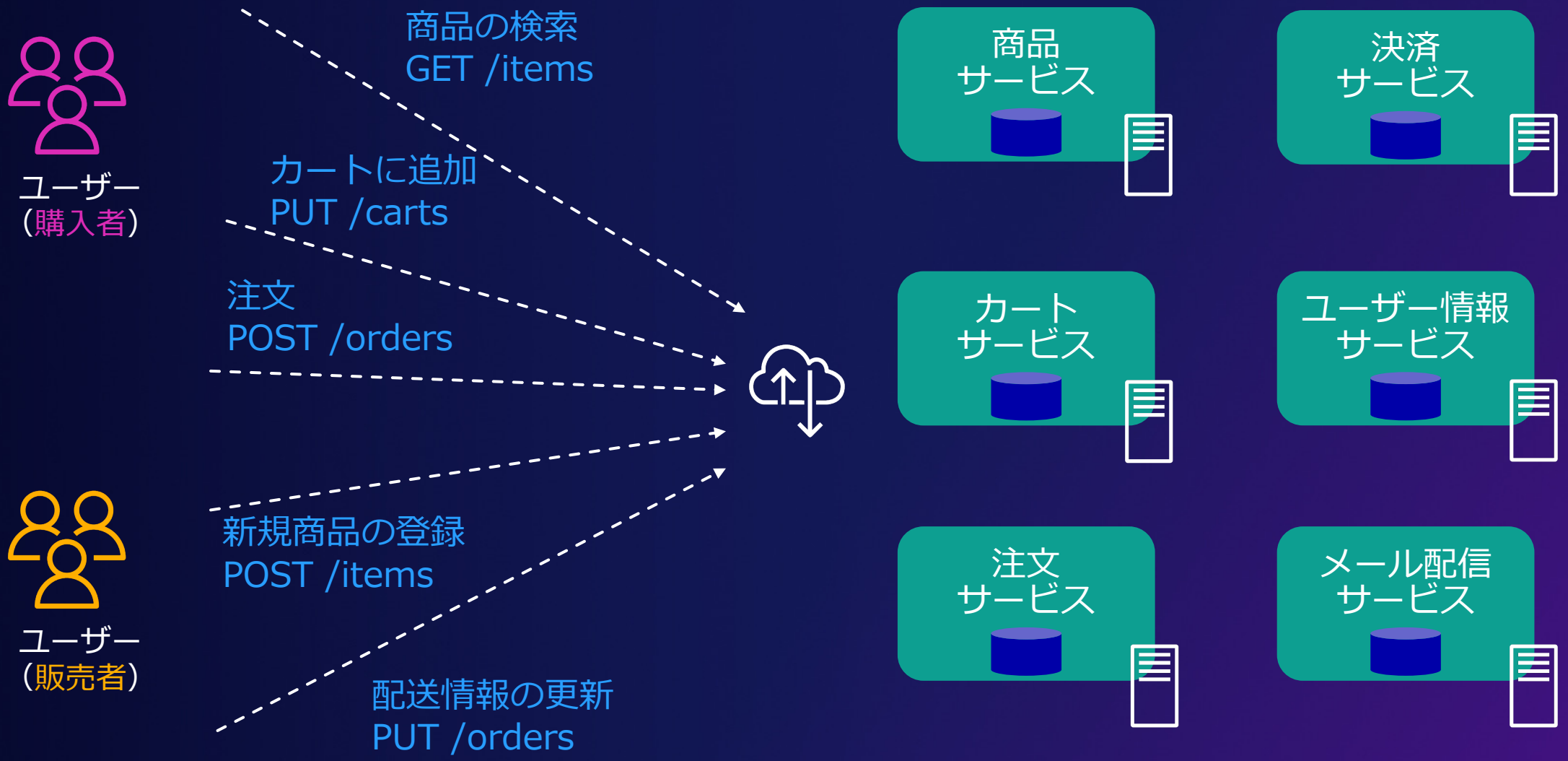
新規商品の登録
POST /items

配送情報の更新
PUT /orders

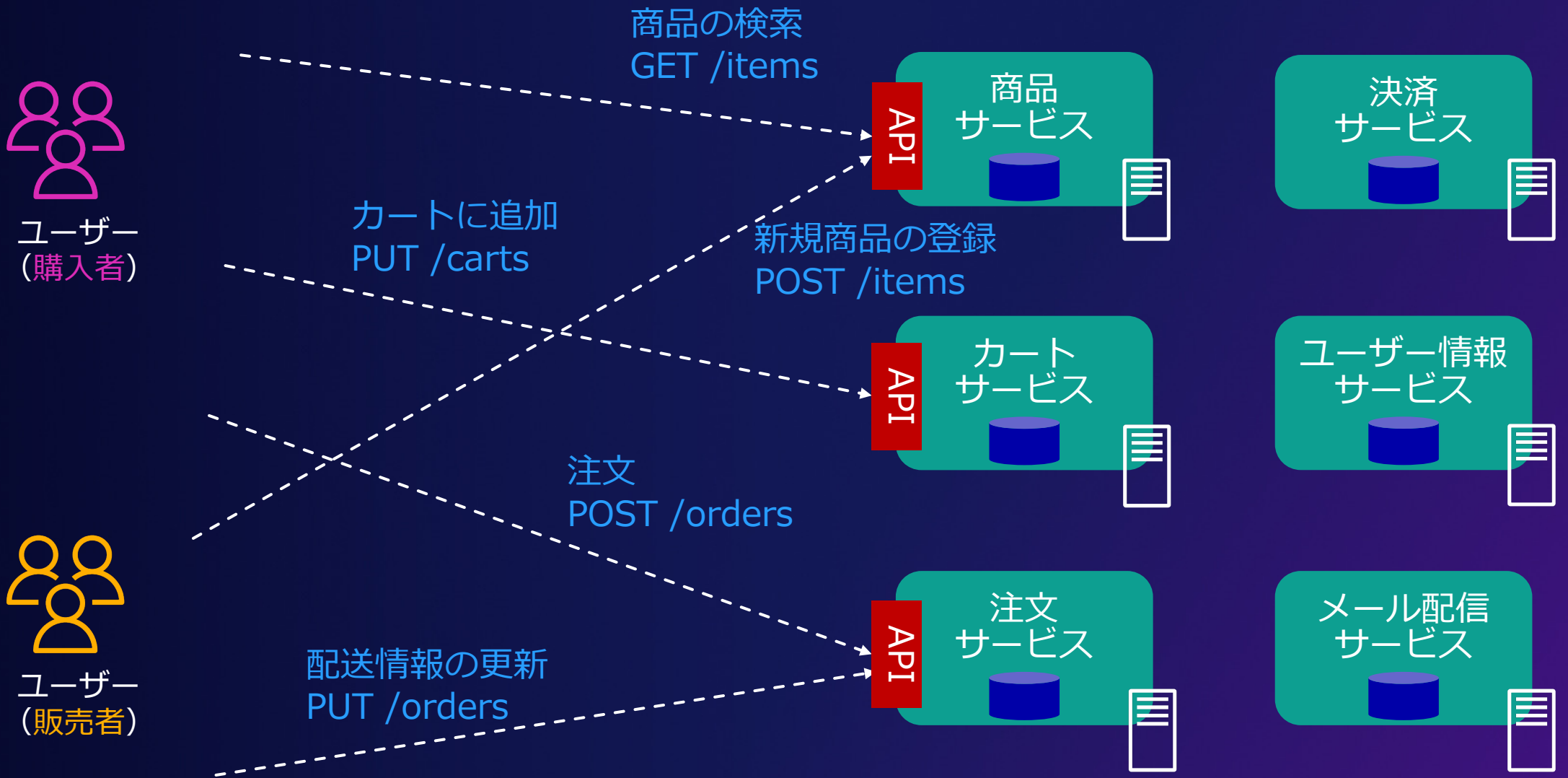


マイクロサービスな SaaS の場合

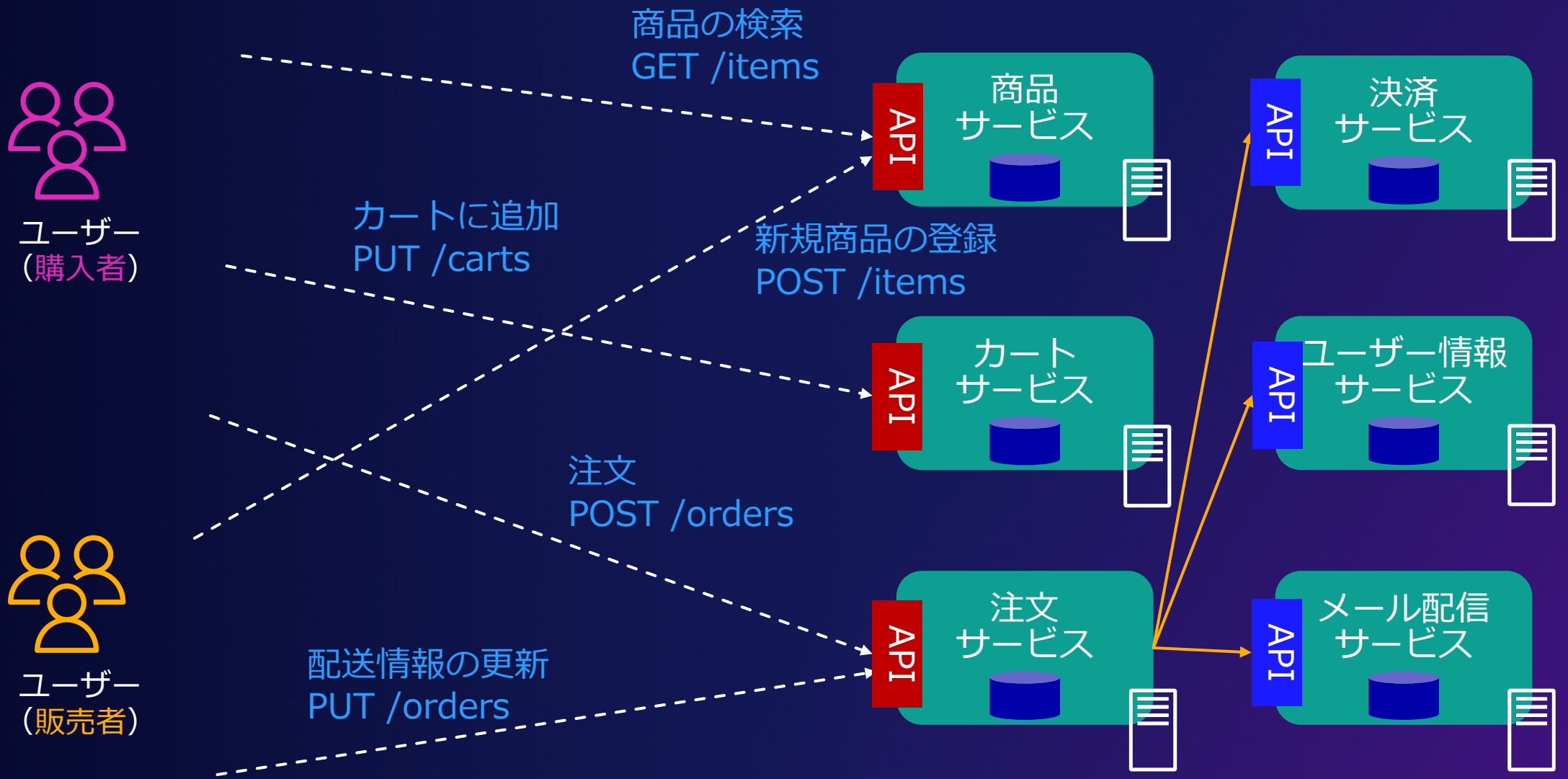
マイクロサービスな EC サイト



マイクロサービスな EC サイト



マイクロサービスな EC サイト



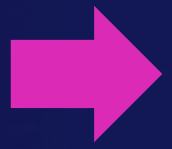
マイクロサービスがもたらすメリット

- 柔軟なスケーリング
- 一つのことに集中できるチーム
- 影響範囲の局所化
- 高速なデプロイ
- 自由な技術選定

etc.

マイクロサービスがもたらすメリット

- 柔軟なスケーリング
- 一つのことに集中できるチーム
- 影響範囲の局所化
- 高速なデプロイ
- 自由な技術選定



どうすれば継続的に安全に
スケーラブルに運用負荷をかけず
ユーザーが望む体験に
改善し続けられるだろうか 🤔

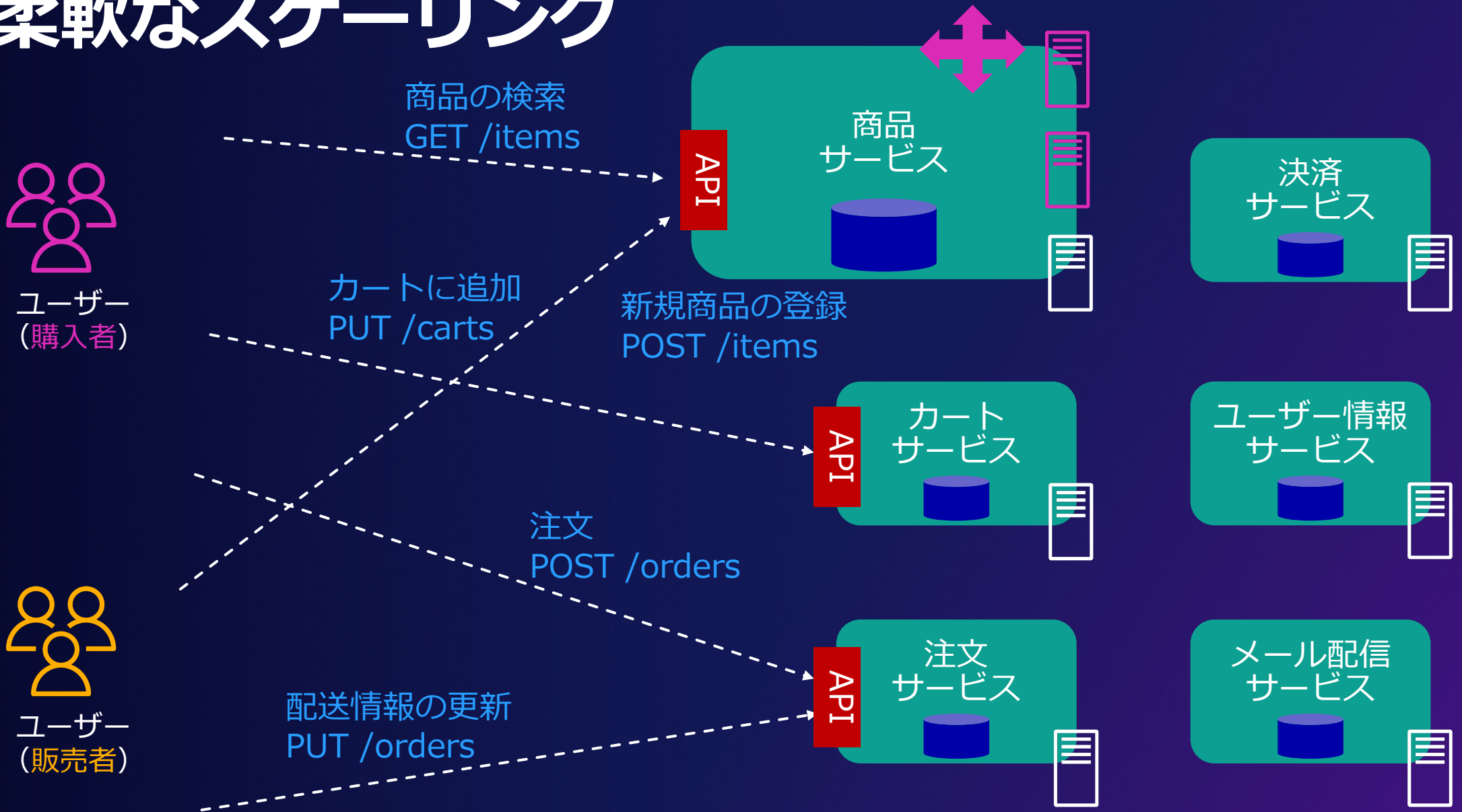
etc.

マイクロサービスがもたらすメリット

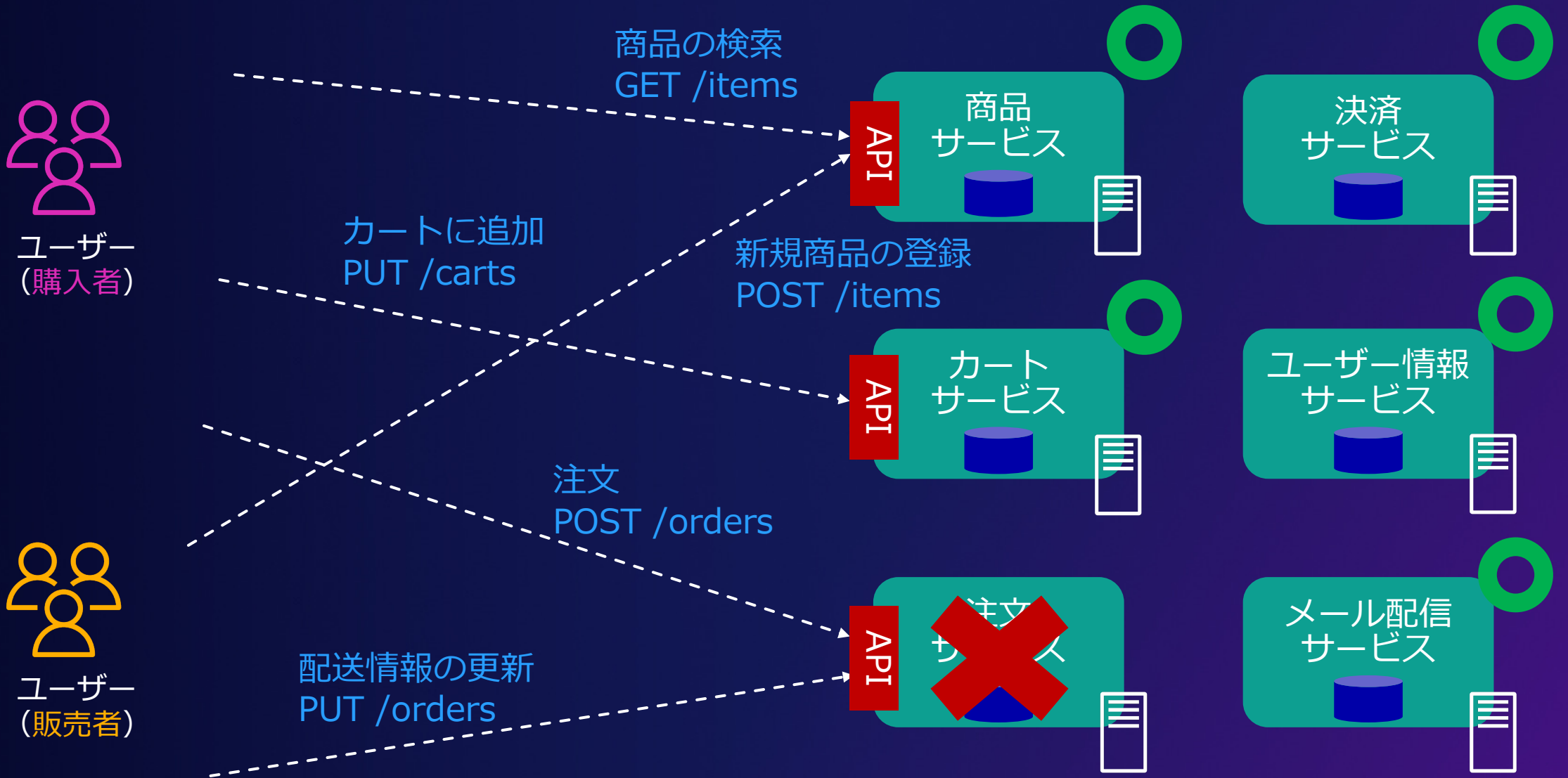
- 柔軟なスケーリング
- 一つのことに集中できるチーム
- 影響範囲の局所化
- 高速なデプロイ
- 自由な技術選定

etc.

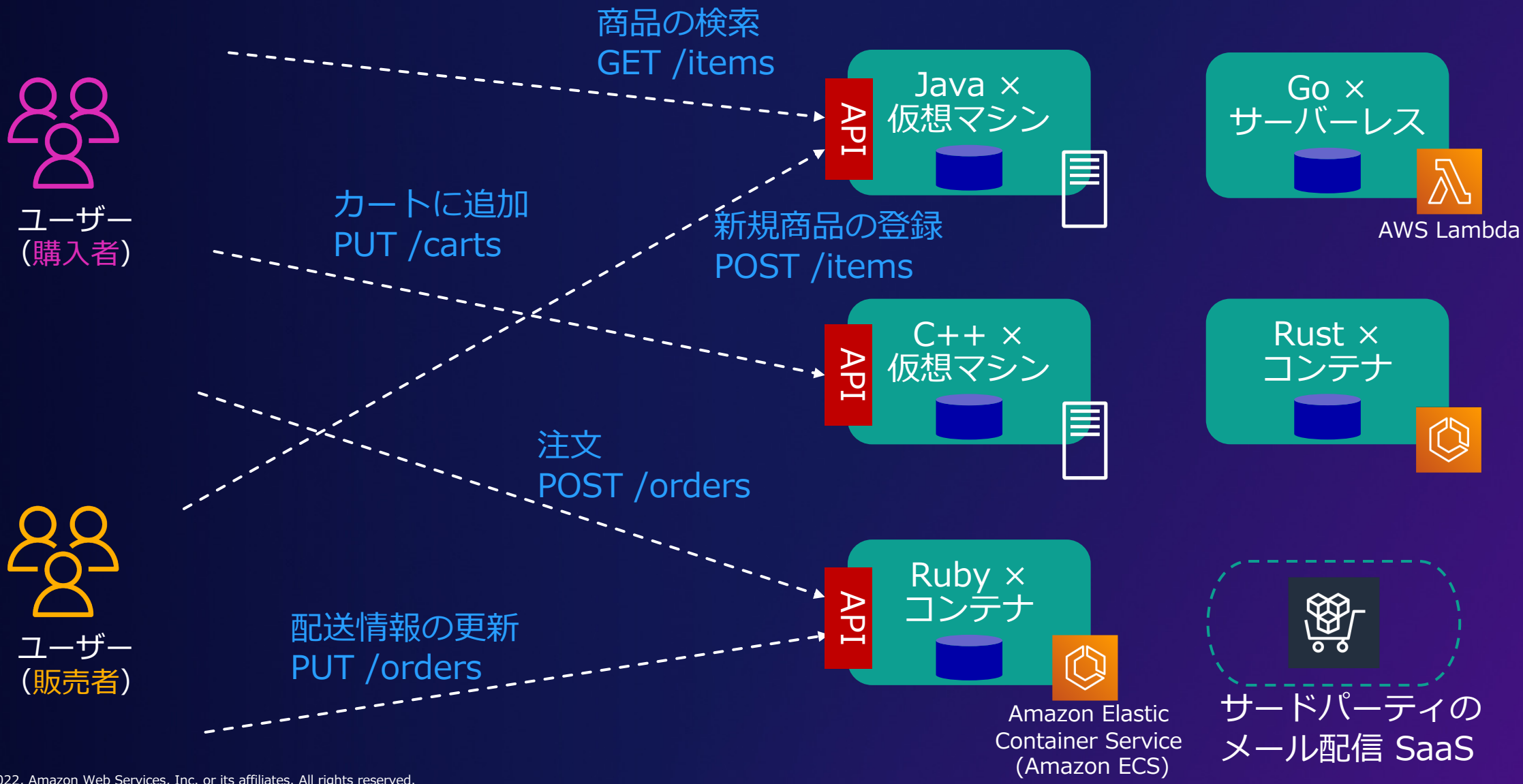
ex. 柔軟なスケーリング



ex. 高い耐障害性



ex. 自由な技術選定



マイクロサービスと SaaS の親和性

マイクロサービス

SaaS

小さいスコープで開発効率の向上

高速なイテレーション

デプロイメントの単位が小さくなる

影響範囲が局所化

サービス単位の柔軟なスケーリング

需要に沿ったコスト最適化

異なる技術スタックを選択できる

マイクロサービスと SaaS の親和性

マイクロサービス

SaaS

小さいスコープで開発効率の向上

高速なイテレーション

デプロイメントの単位が小さくなる

影響範囲が局所化

サービス単位の柔軟なスケーリング

需要に沿ったコスト最適化

異なる技術スタックを選択できる

マイクロサービスと SaaS の親和性

マイクロサービス

小さいスコープで開発効率の向上

高速なイテレーション

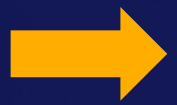
デプロイメントの単位が小さくなる

影響範囲が局所化

サービス単位の柔軟なスケーリング

需要に沿ったコスト最適化

異なる技術スタックを選択できる



SaaS

頻繁な実験とフィードバック

エクスペリエンスにフォーカスした
サービスの改善を高速化

マイクロサービスと SaaS の親和性

マイクロサービス

SaaS

小さいスコープで開発効率の向上

高速なイテレーション

デプロイメントの単位が小さくなる

影響範囲が局所化

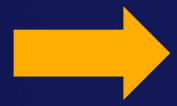
サービス単位の柔軟なスケーリング

需要に沿ったコスト最適化

異なる技術スタックを選択できる

頻繁な実験とフィードバック

エクスペリエンスにフォーカスした
サービスの改善を高速化



マイクロサービスと SaaS の親和性

マイクロサービス

SaaS

小さいスコープで開発効率の向上

高速なイテレーション

→

頻繁な実験とフィードバック

エクスペリエンスにフォーカスした
サービスの改善を高速化

デプロイメントの単位が小さくなる

影響範囲が局所化

→

サービスの信頼性向上

チャーンの防止や機会損失の削減

サービス単位の柔軟なスケーリング

需要に沿ったコスト最適化

異なる技術スタックを選択できる

マイクロサービスと SaaS の親和性

マイクロサービス

SaaS

小さいスコープで開発効率の向上

高速なイテレーション

→

頻繁な実験とフィードバック

エクスペリエンスにフォーカスした
サービスの改善を高速化

デプロイメントの単位が小さくなる

影響範囲が局所化

→

サービスの信頼性向上

チャーンの防止や機会損失の削減

サービス単位の柔軟なスケーリング

需要に沿ったコスト最適化

→

異なる技術スタックを選択できる

マイクロサービスと SaaS の親和性

マイクロサービス		SaaS
	小さいスコープで開発効率の向上 高速なイテレーション	頻繁な実験とフィードバック エクスペリエンスにフォーカスしたサービスの改善を高速化
	デプロイメントの単位が小さくなる 影響範囲が局所化	サービスの信頼性向上 チャーンの防止や機会損失の削減
	サービス単位の柔軟なスケーリング 需要に沿ったコスト最適化	機能ごとに細かく SLA を設定可能 テナントの規模に応じた最適なキャパシティ管理
	異なる技術スタックを選択できる	

マイクロサービスと SaaS の親和性

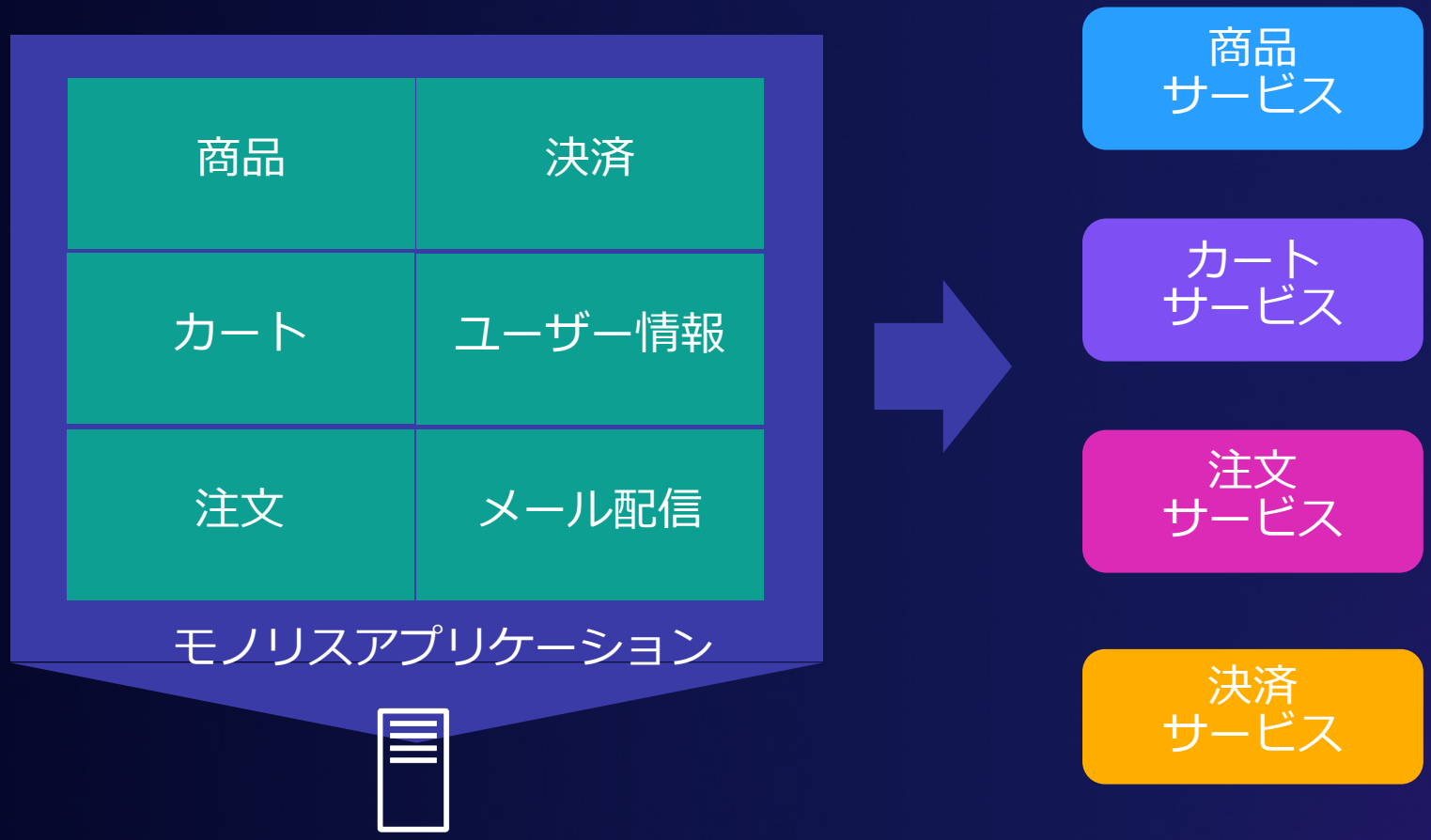
マイクロサービス		SaaS
	小さいスコープで開発効率の向上 高速なイテレーション	頻繁な実験とフィードバック エクスペリエンスにフォーカスしたサービスの改善を高速化
	デプロイメントの単位が小さくなる 影響範囲が局所化	サービスの信頼性向上 チャーンの防止や機会損失の削減
	サービス単位の柔軟なスケーリング 需要に沿ったコスト最適化	機能ごとに細かく SLA を設定可能 テナントの規模に応じた最適なキャパシティ管理
	異なる技術スタックを選択できる	

マイクロサービスと SaaS の親和性

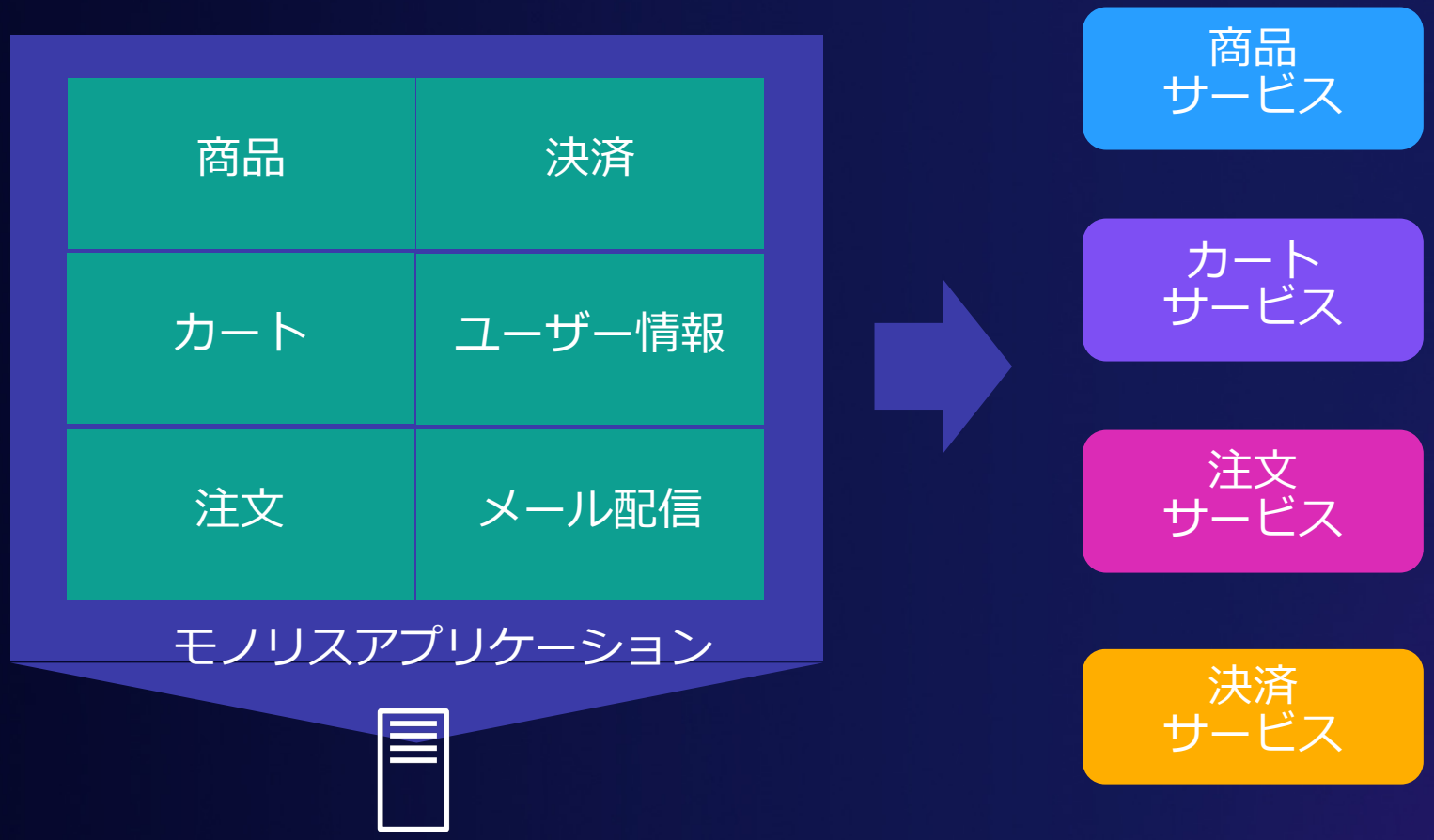
マイクロサービス		SaaS
	小さいスコープで開発効率の向上 高速なイテレーション	頻繁な実験とフィードバック エクスペリエンスにフォーカスしたサービスの改善を高速化
	デプロイメントの単位が小さくなる 影響範囲が局所化	サービスの信頼性向上 チャーンの防止や機会損失の削減
	サービス単位の柔軟なスケーリング 需要に沿ったコスト最適化	機能ごとに細かく SLA を設定可能 テナントの規模に応じた最適なキャパシティ管理
	異なる技術スタックを選択できる	様々なペルソナの要件に応じた構成 ビジネスの成長に合わせた最適なチューニング

SaaS の目線で見るサービス境界

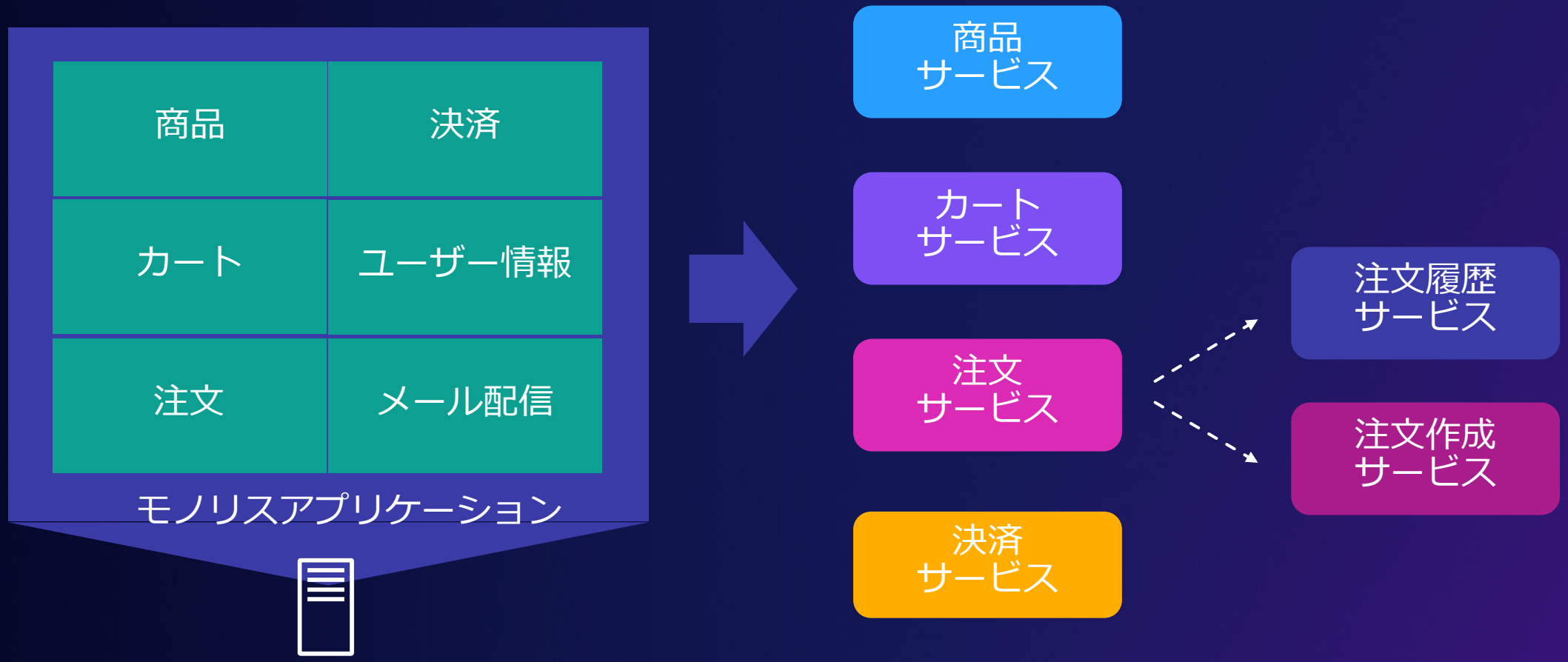
サービス境界をどう区切るか？



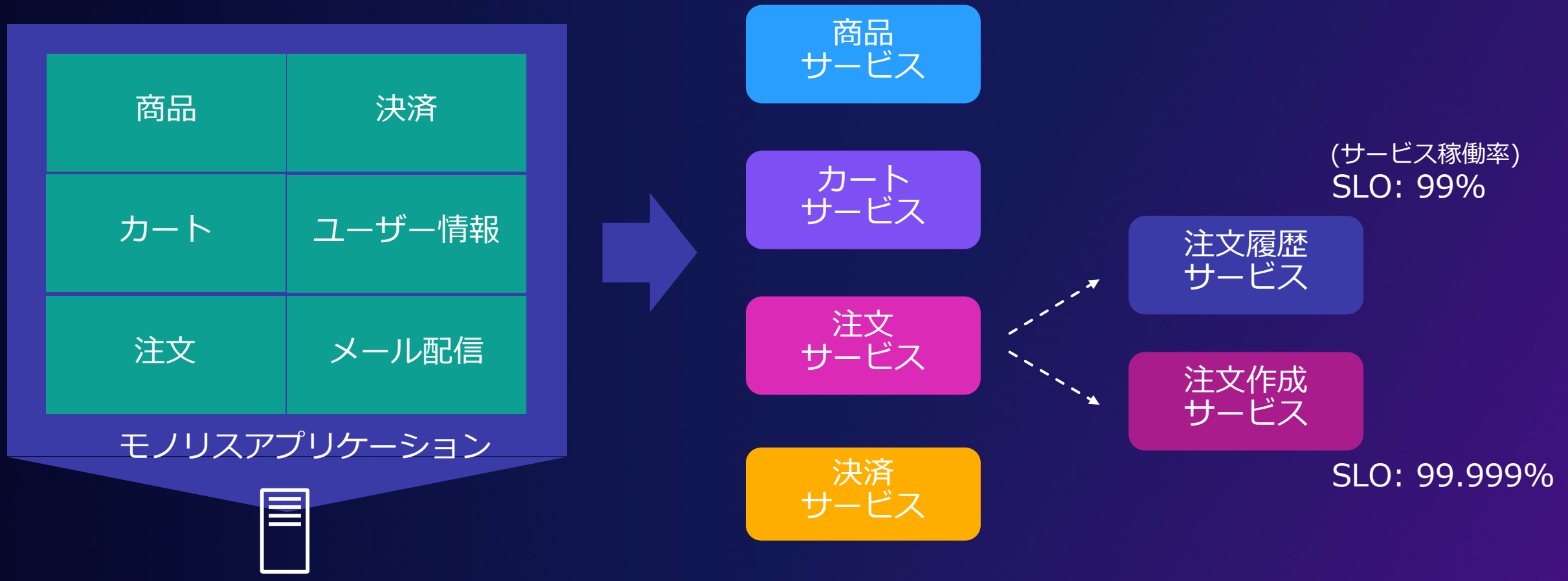
1. 可用性要件を考慮に入れる



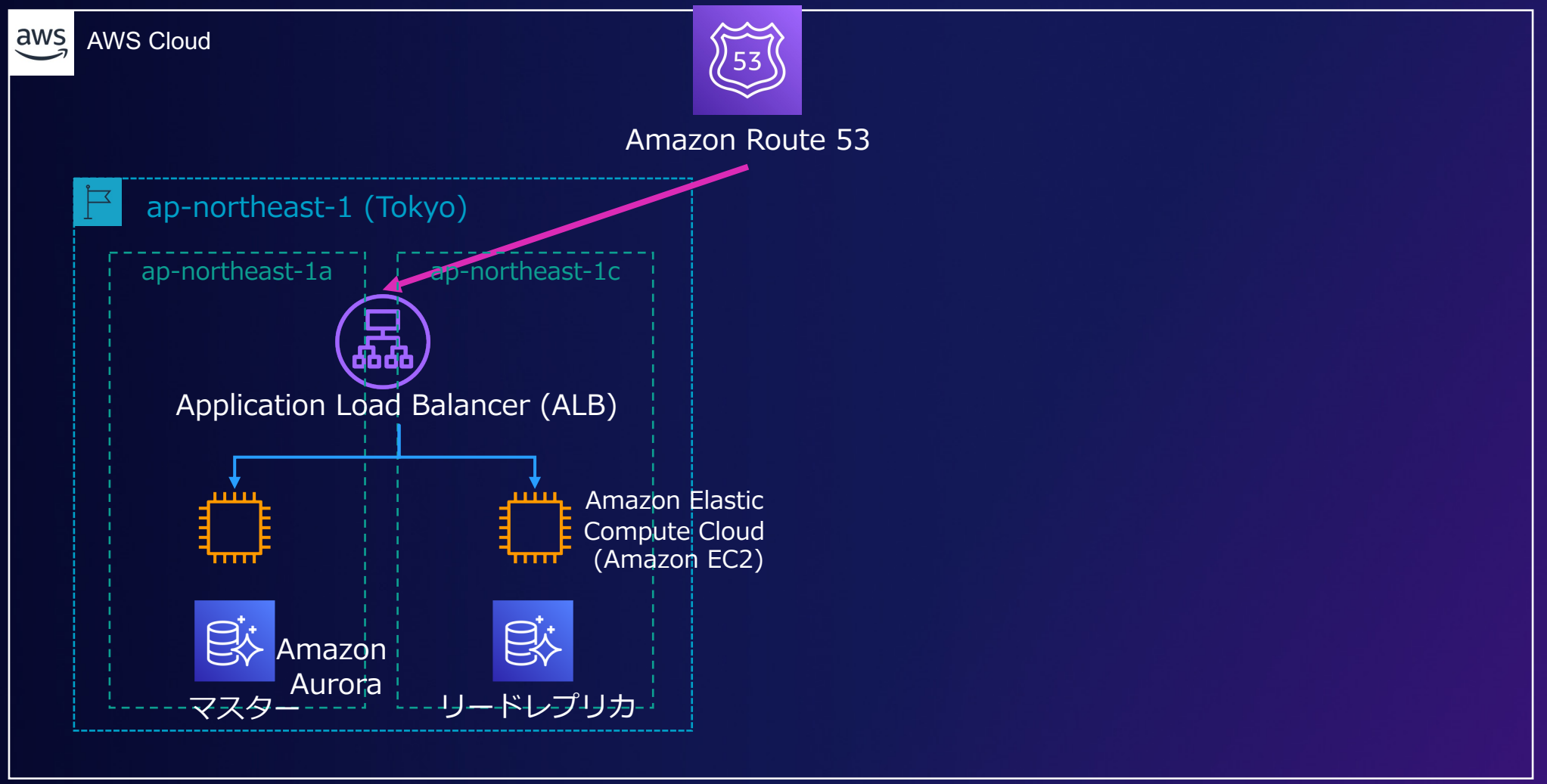
1. 可用性要件を考慮に入れる



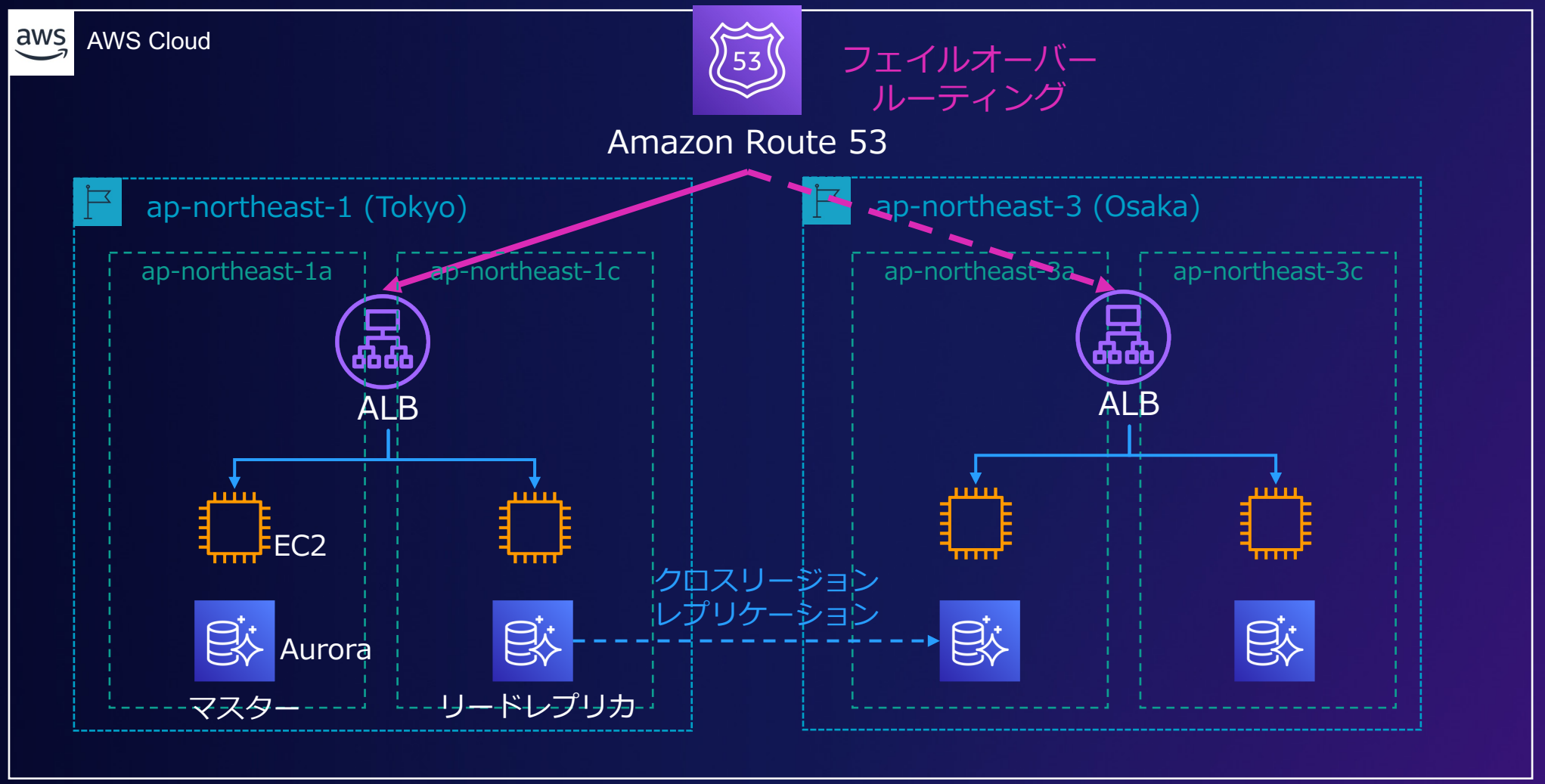
1. 可用性要件を考慮に入れる



ex. 注文履歴サービスはマルチ AZ に作る



ex. 注文作成サービスはマルチリージョンに作る

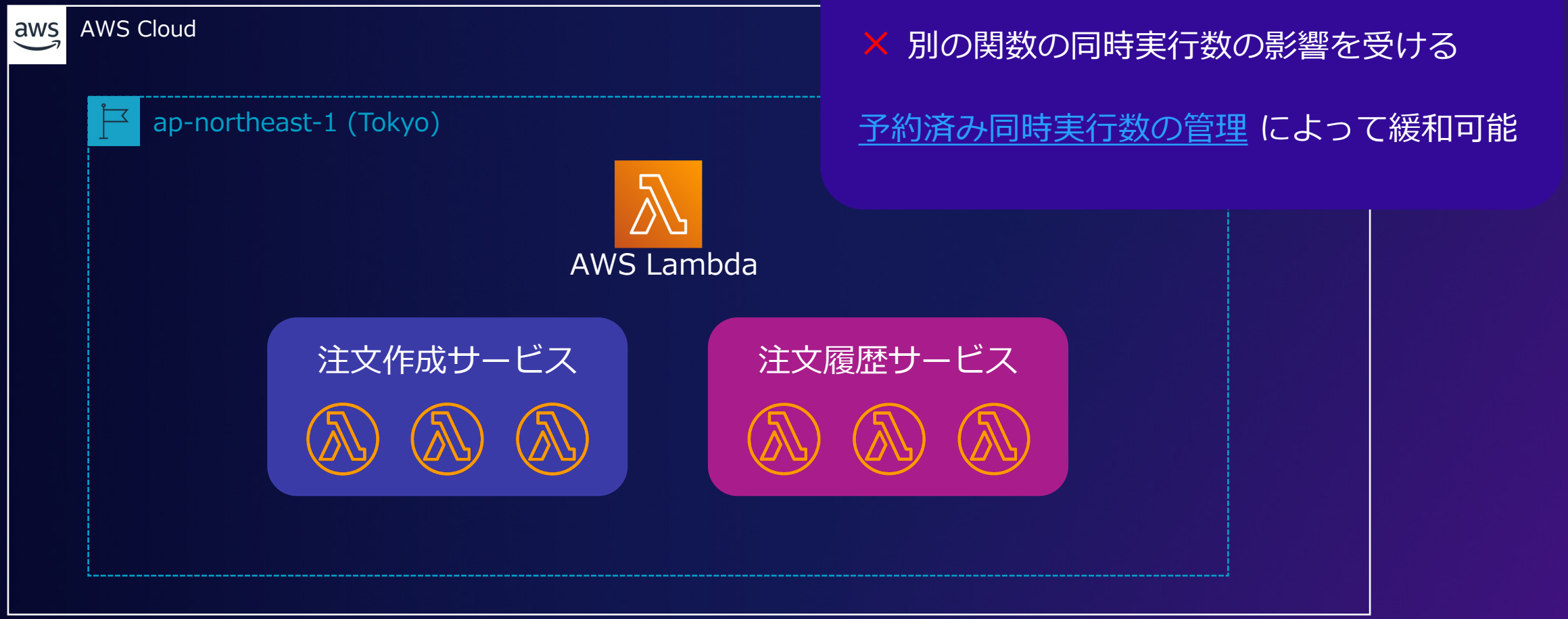


ex. サーバーレスの場合

各関数は

✕ 別の関数の同時実行数の影響を受ける

予約済み同時実行数の管理 によって緩和可能



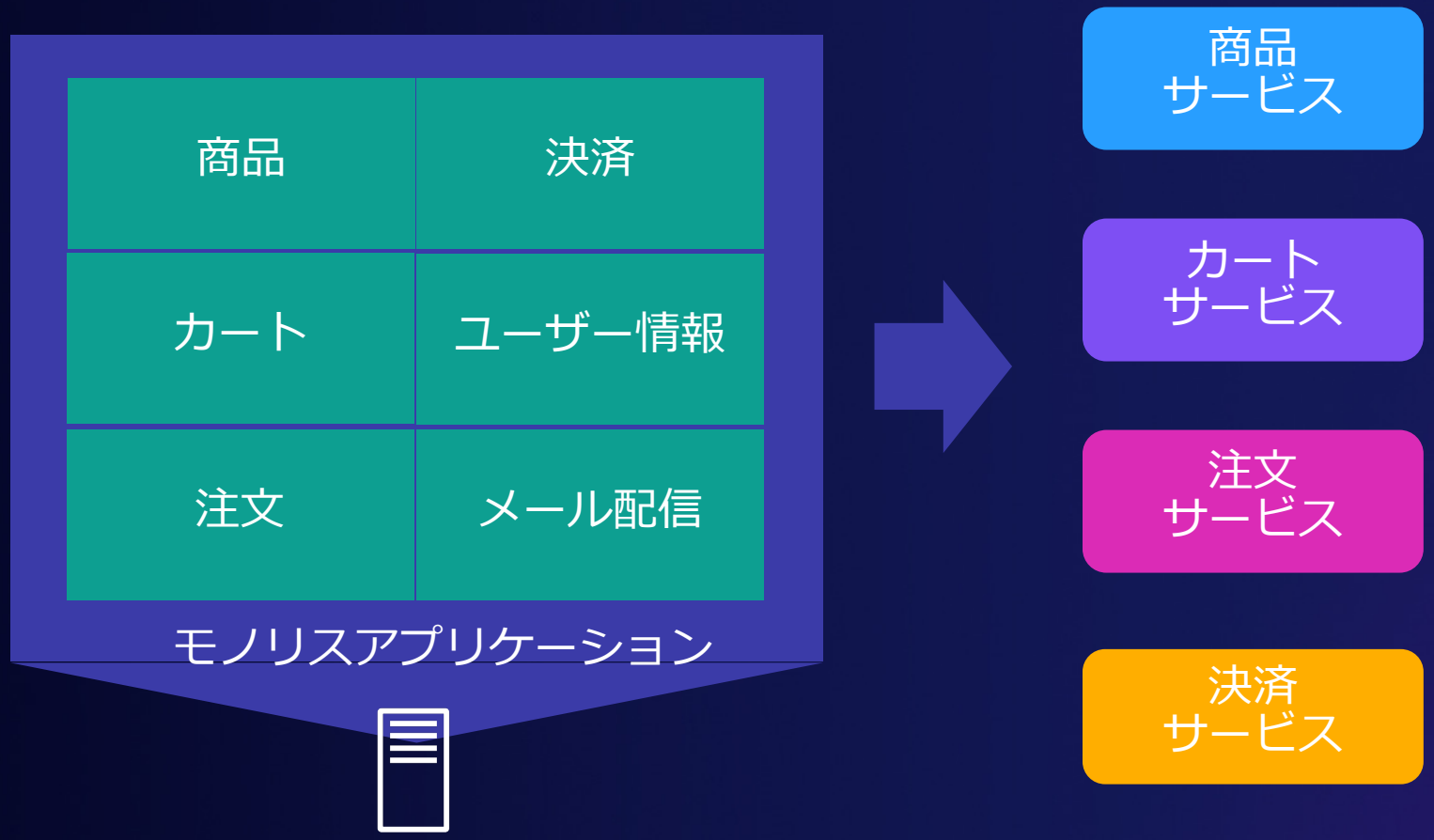
ex. サーバーレスの場合

各関数は

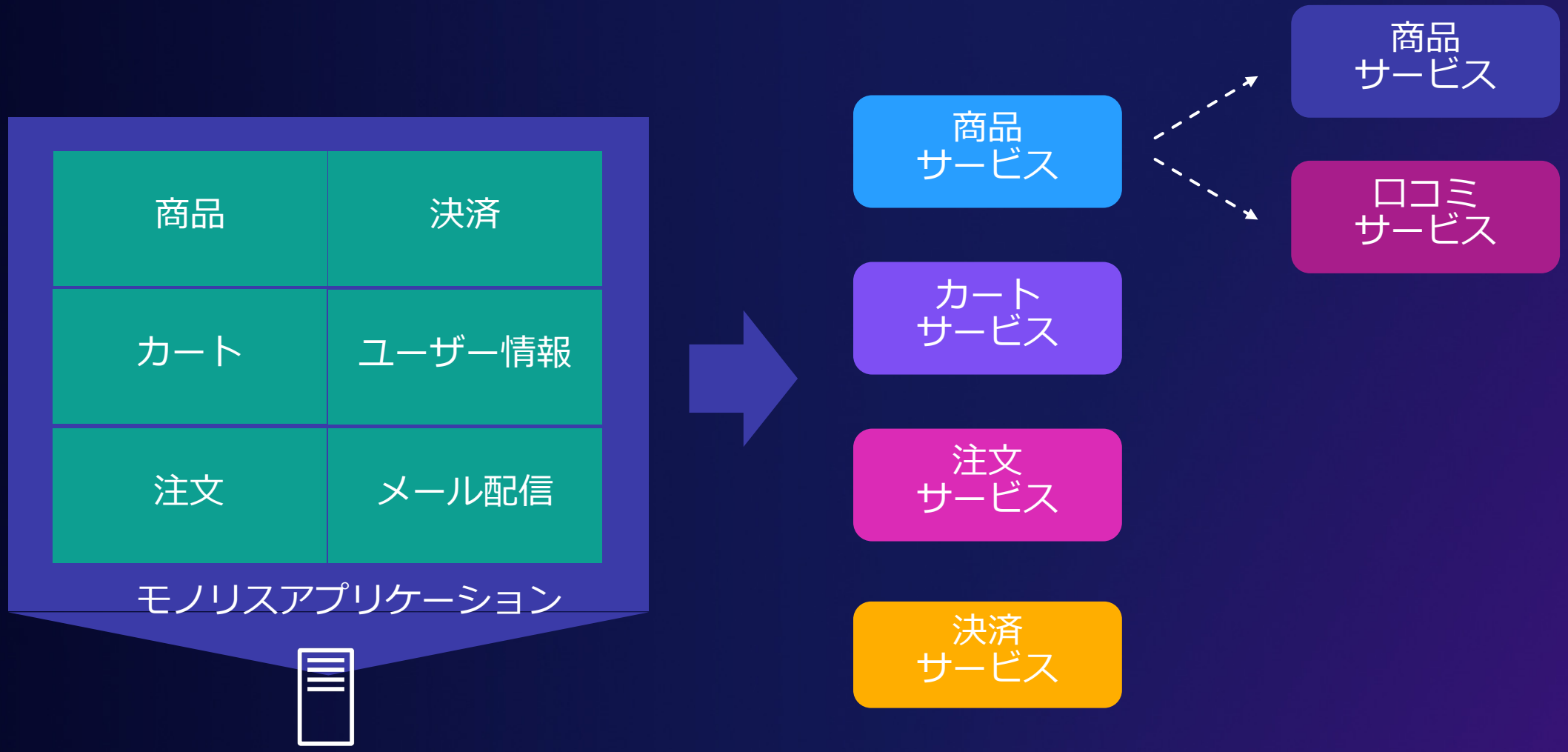
✕ 別の関数の同時実行数の影響を受ける

予約済み同時実行数の管理 によって緩和可能

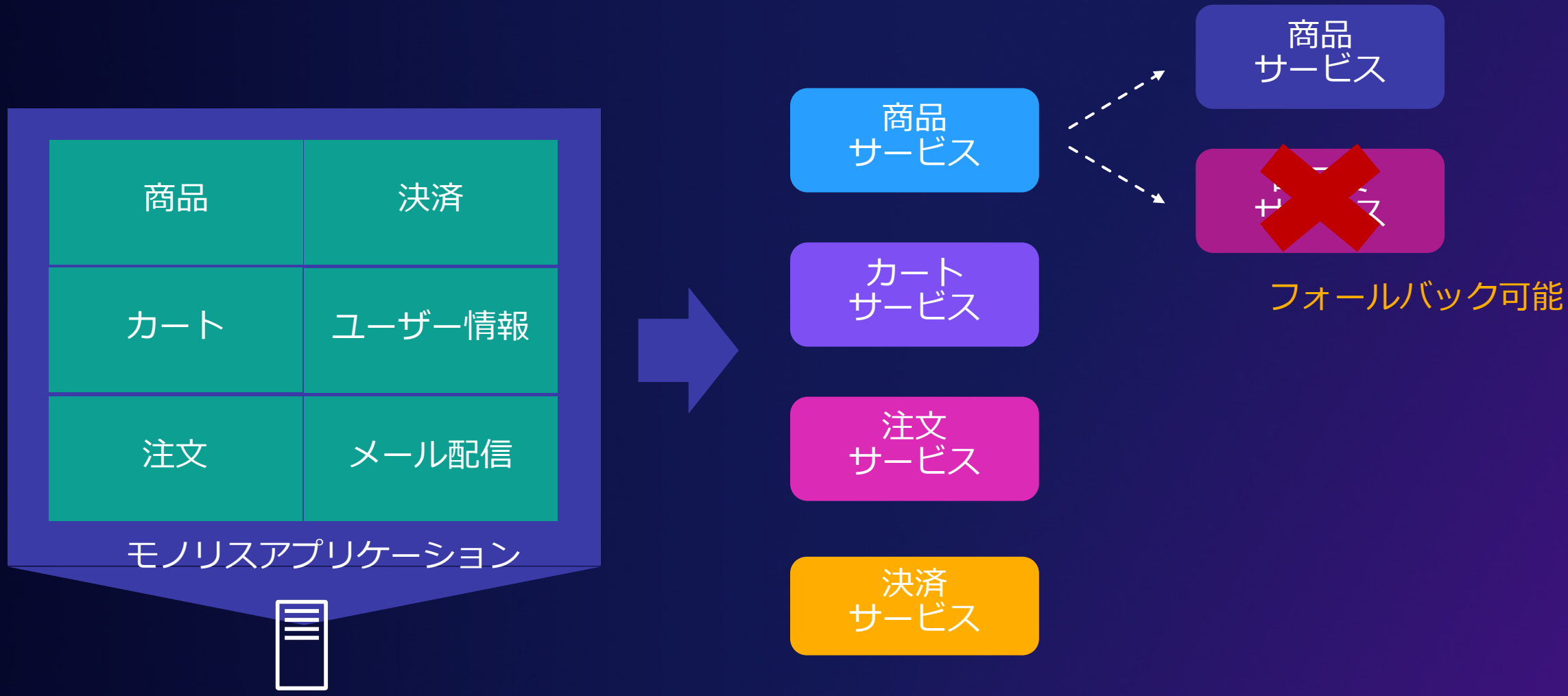
2. 耐障害性を考慮に入れる



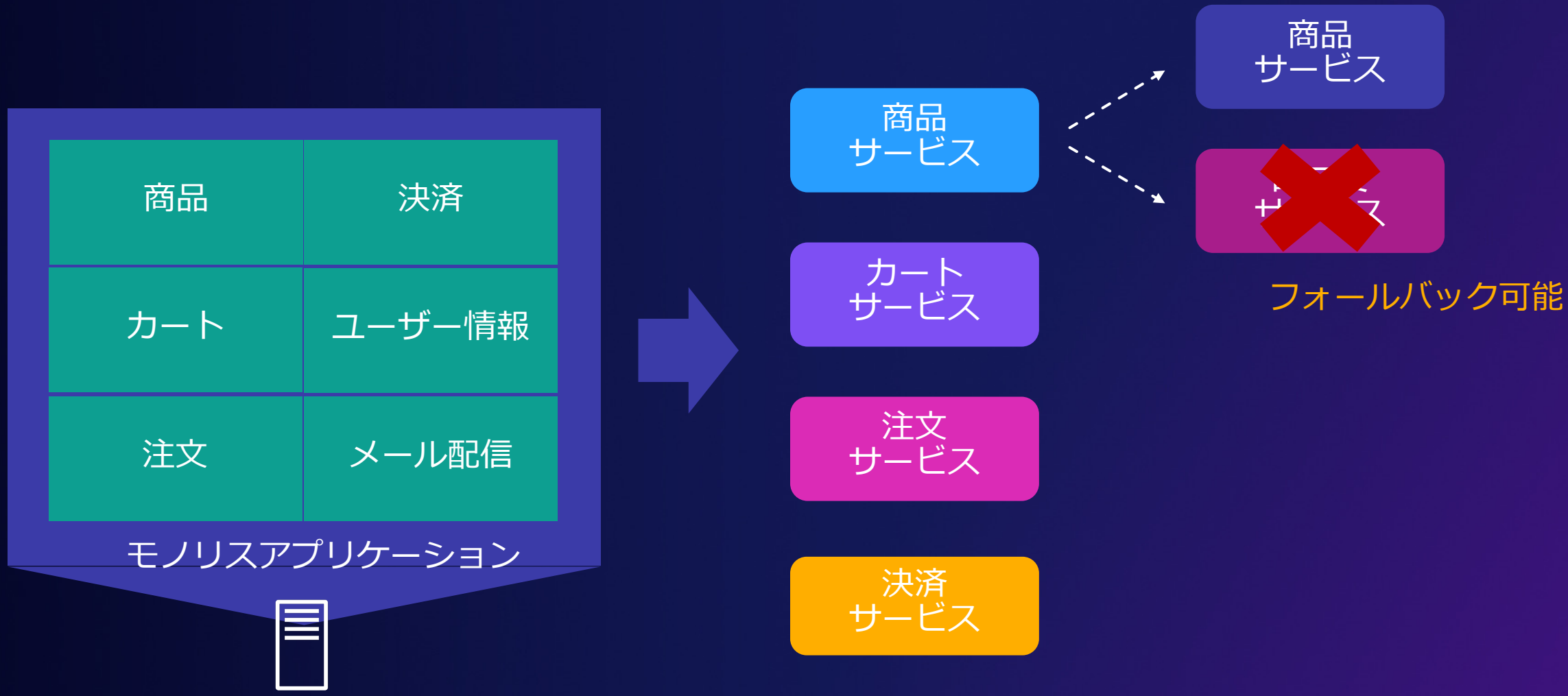
2. 耐障害性を考慮に入れる



2. 耐障害性を考慮に入れる

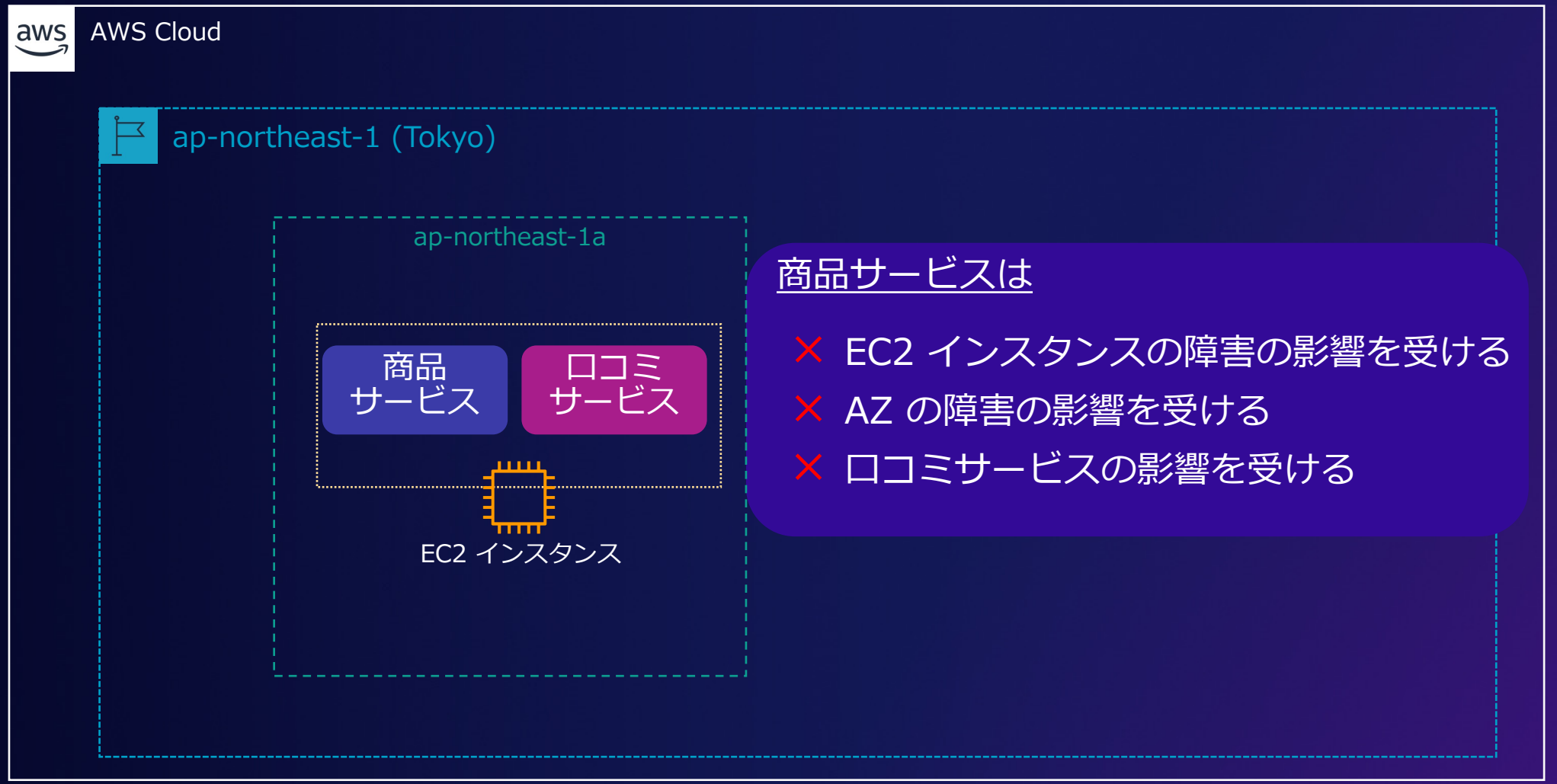


2. 耐障害性を考慮に入れる

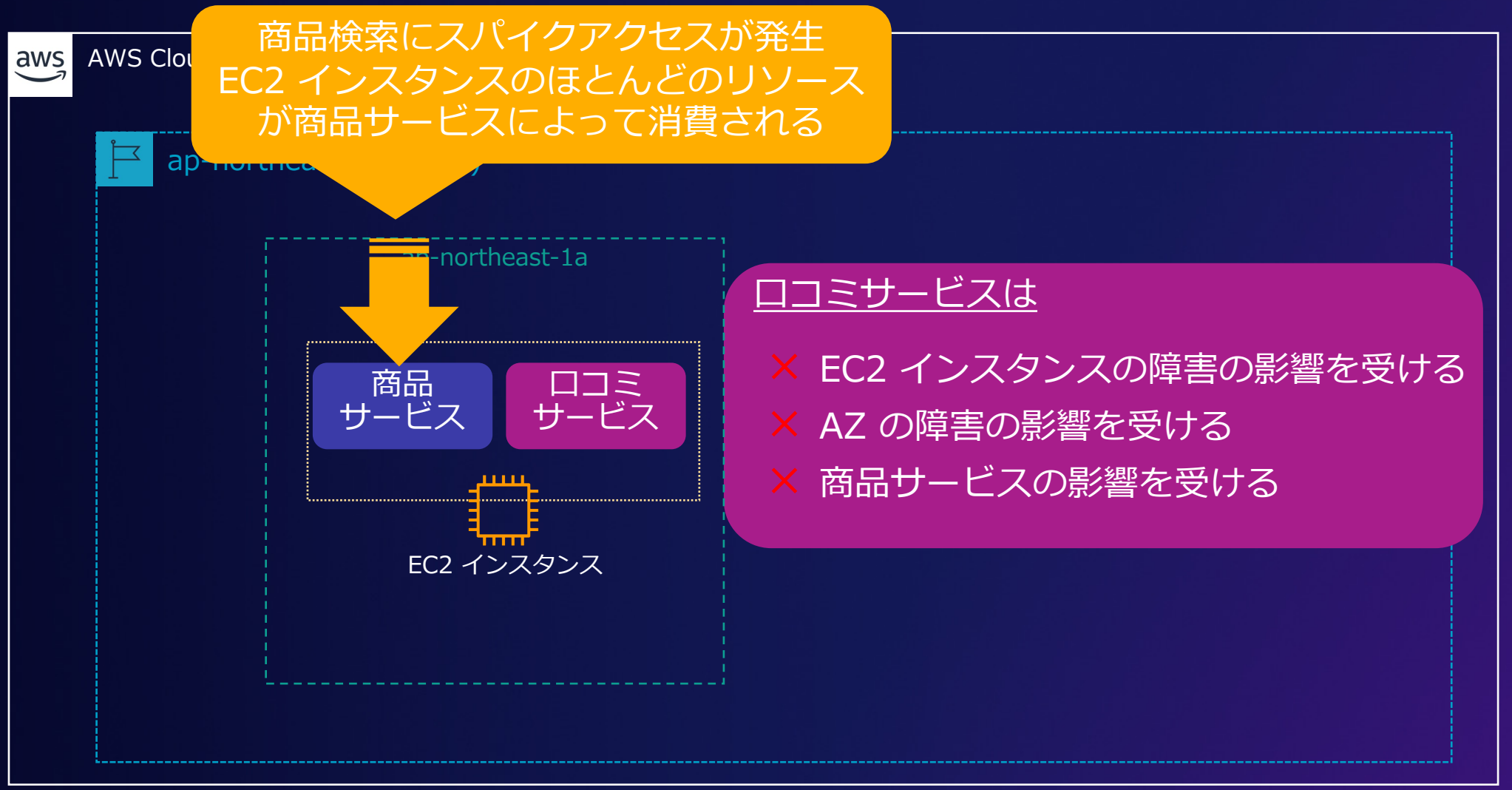


Blast radius (爆発半径) を意識する

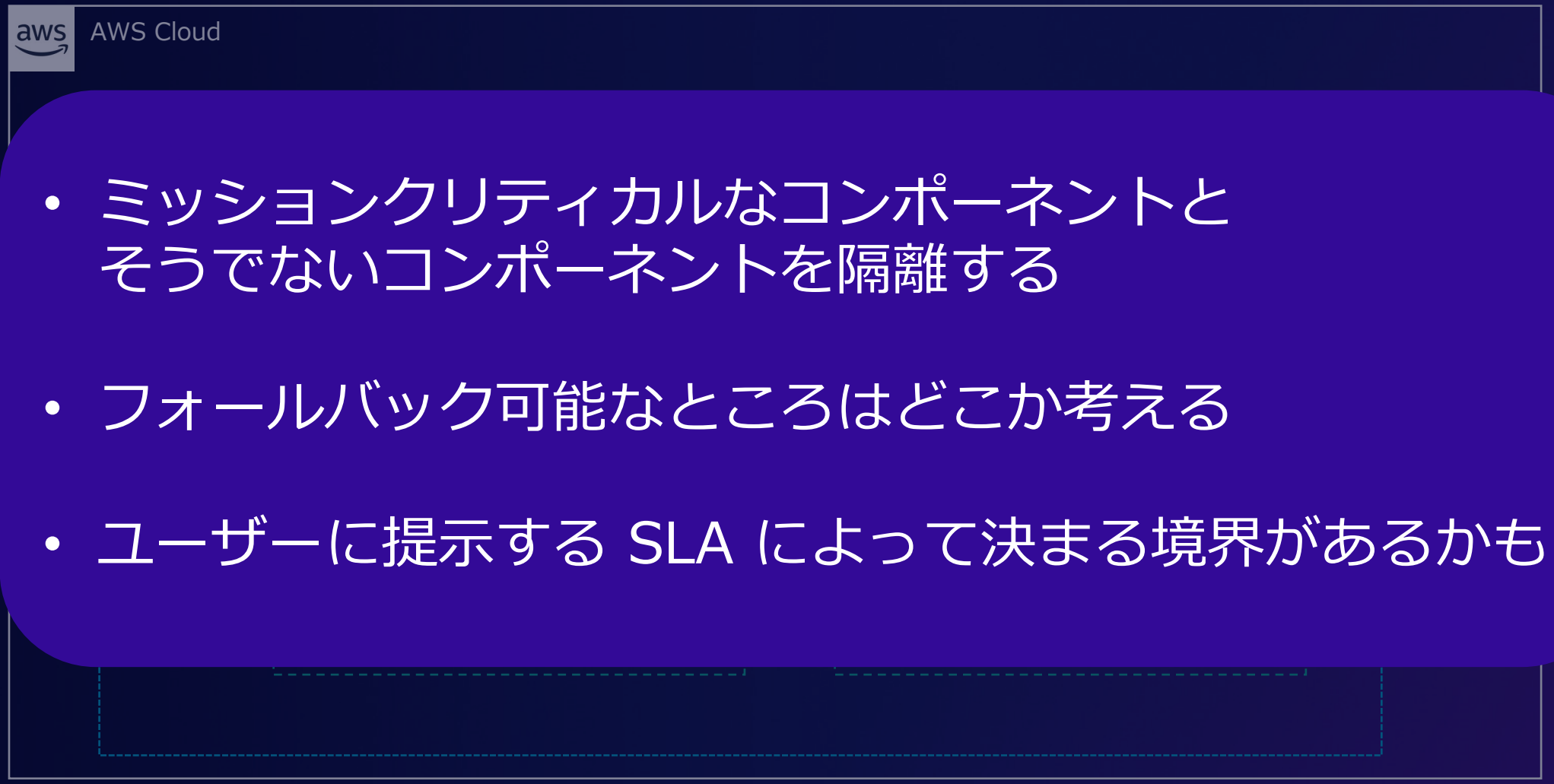
Blast radius (爆発半径) を意識する



Blast radius (爆発半径) を意識する



Blast radius (爆発半径) を意識する

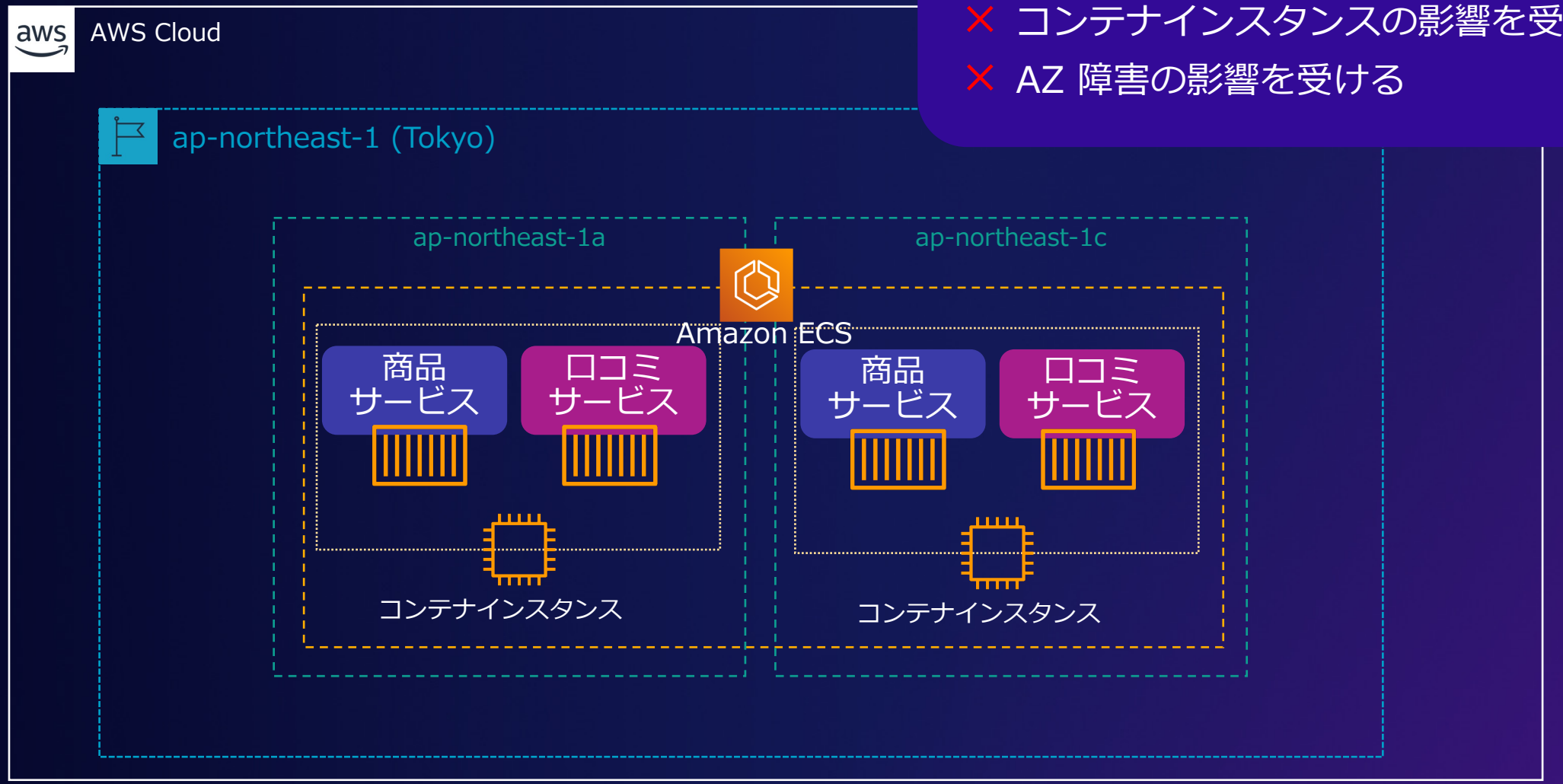


- ミッションクリティカルなコンポーネントと
  そうでないコンポーネントを隔離する
- フォールバック可能なところはどこか考える
- ユーザーに提示する SLA によって決まる境界があるかも

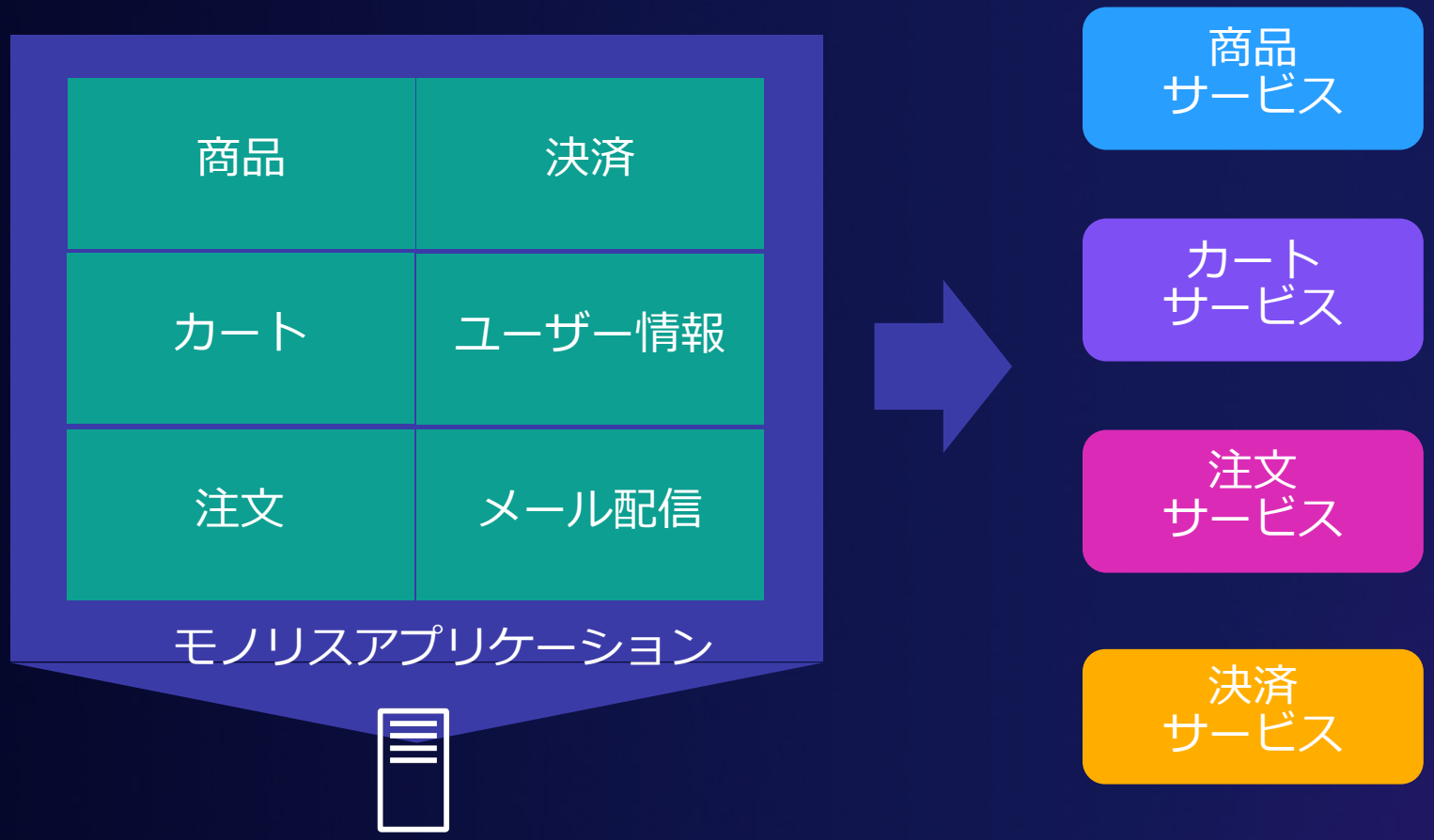
ex. コンテナの場合

各サービスは

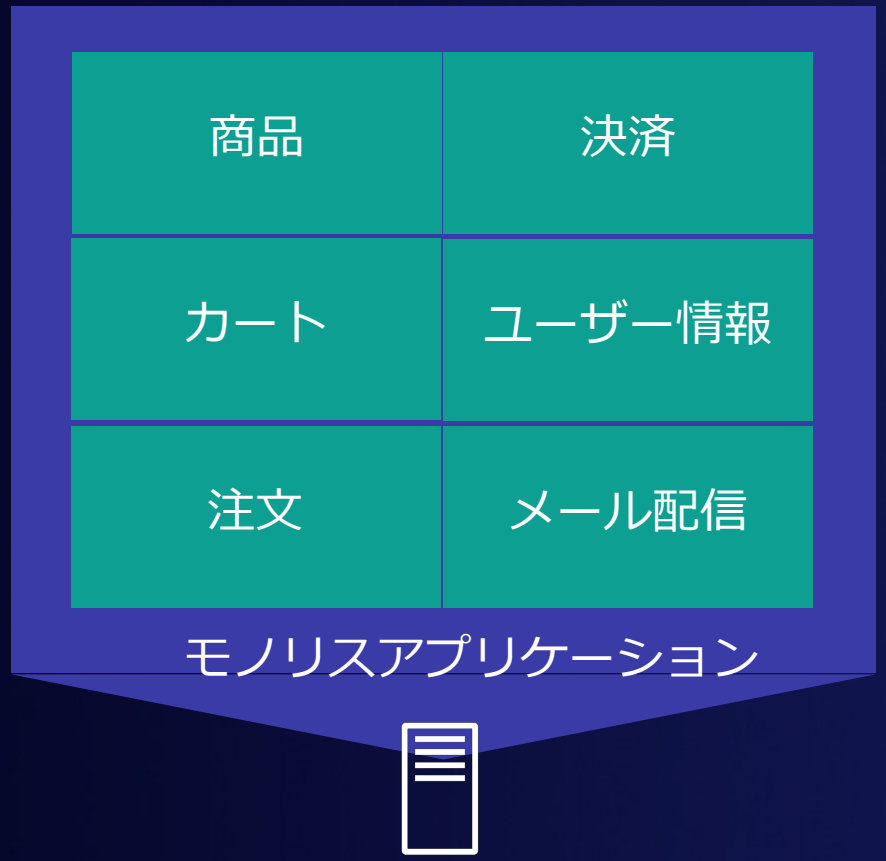
- ✕ タスク内の別コンテナの影響を受ける
- ✕ コンテナインスタンスの影響を受ける
- ✕ AZ 障害の影響を受ける



3. データ要件を考慮に入れる



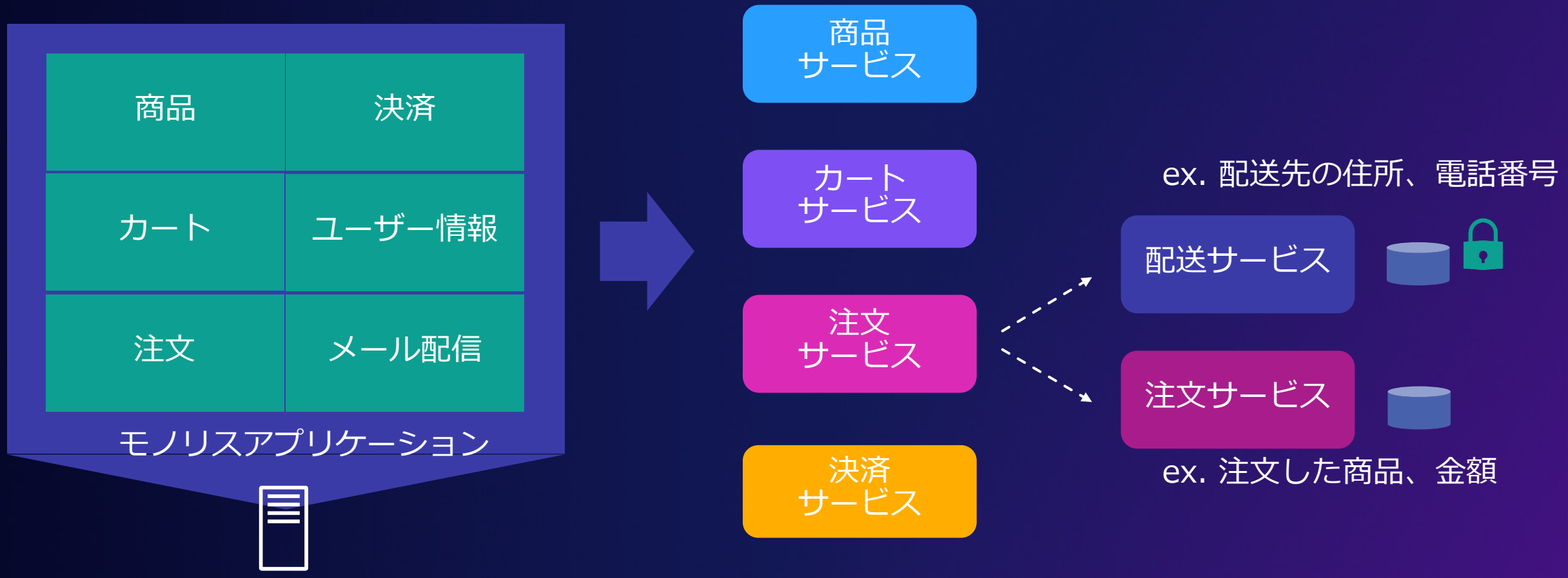
3. データ要件を考慮に入れる



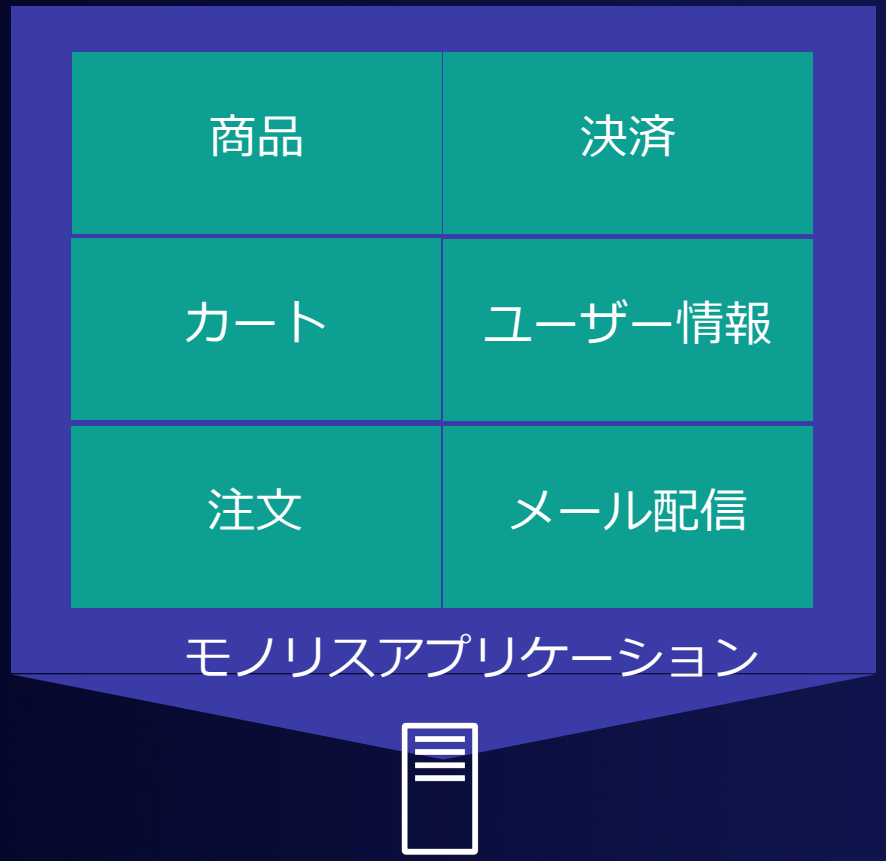
ex. 配送先の住所、電話番号

ex. 注文した商品、金額

3. データ要件を考慮に入れる



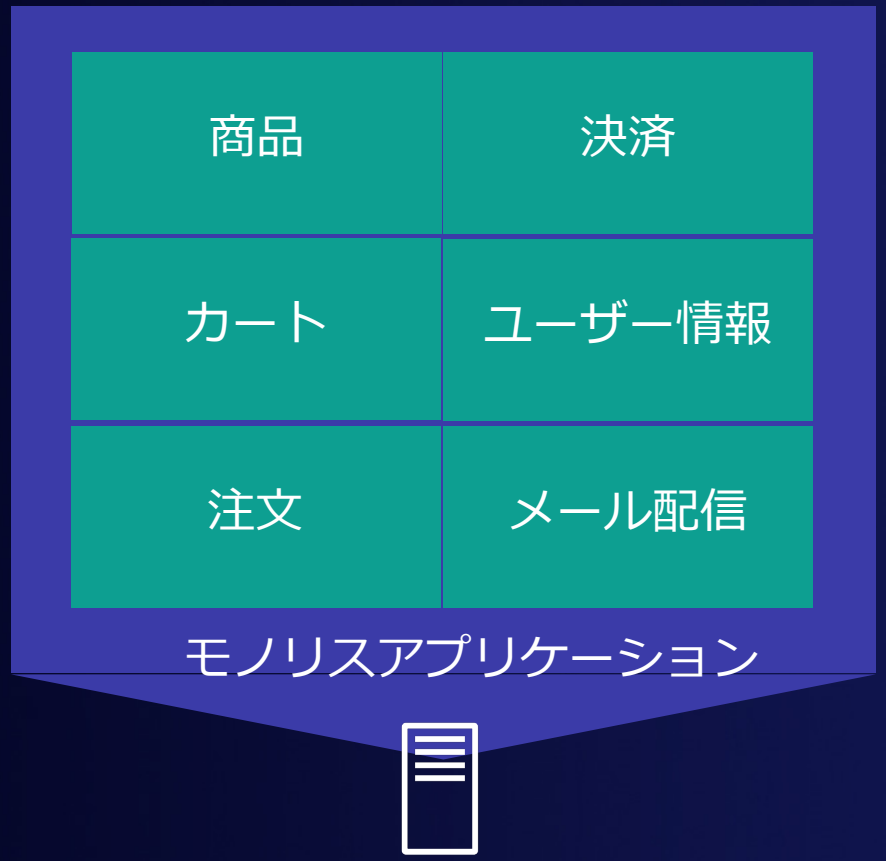
3. データ要件を考慮に入れる

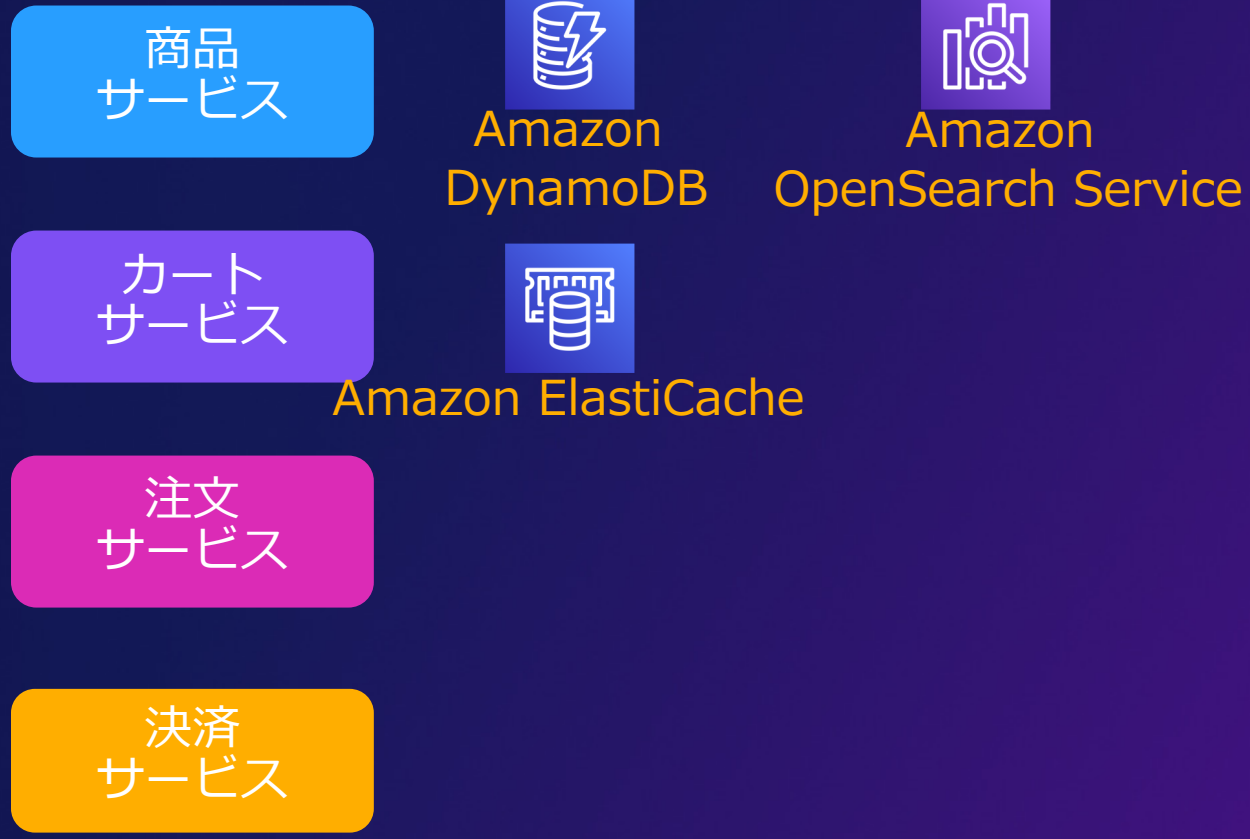


Amazon
DynamoDB

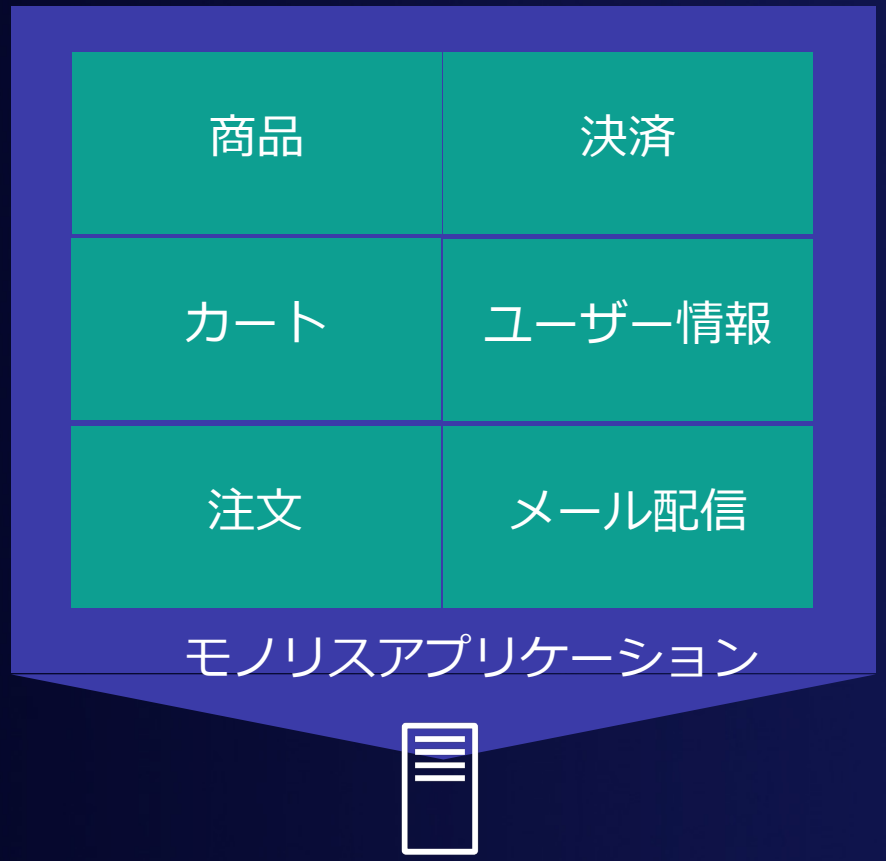
Amazon
OpenSearch Service

3. データ要件を考慮に入れる





3. データ要件を考慮に入れる



商品
サービス

カート
サービス

注文
サービス

決済
サービス

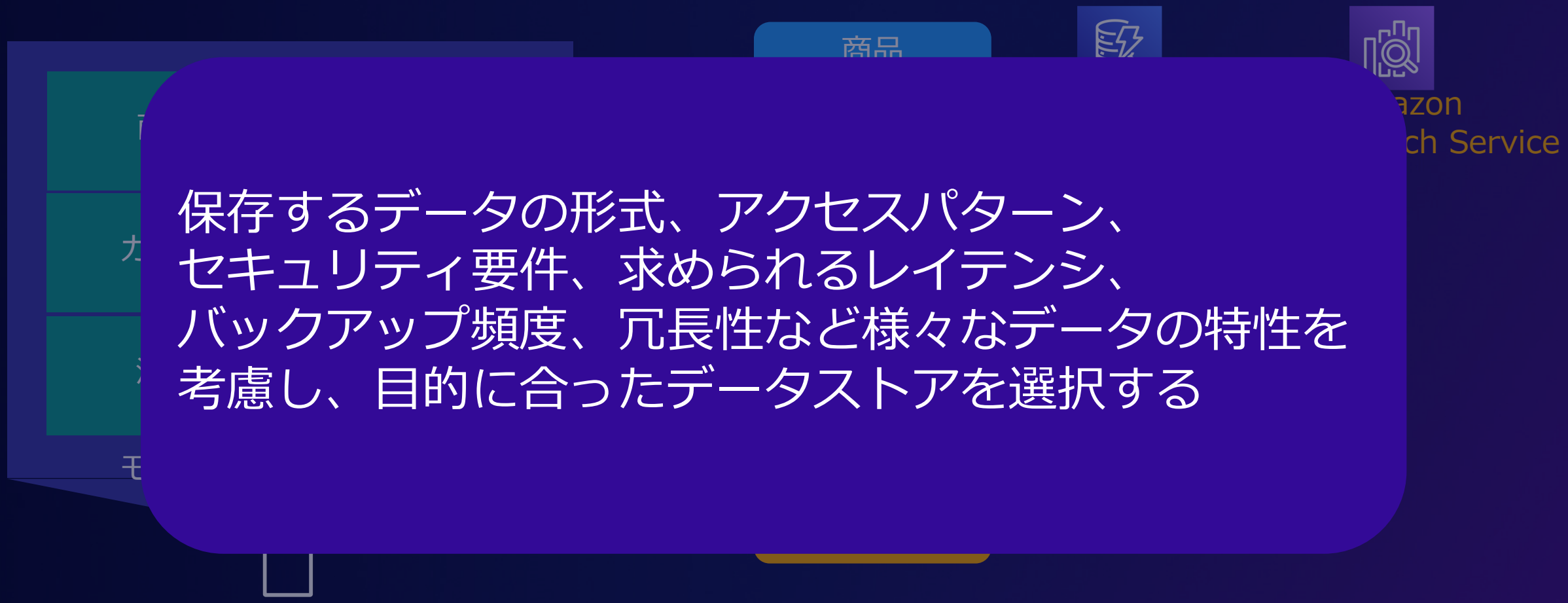
Amazon
DynamoDB

Amazon
OpenSearch Service

Amazon ElastiCache

Amazon Aurora

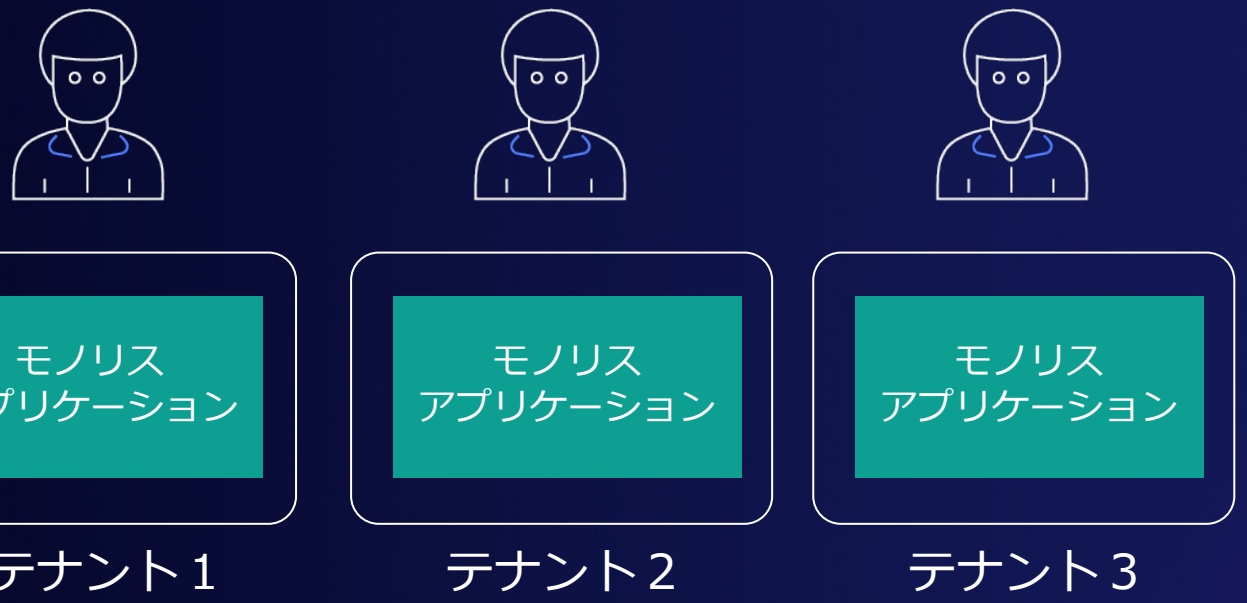
ex. データ要件を考慮に入れる



保存するデータの形式、アクセスパターン、
セキュリティ要件、求められるレイテンシ、
バックアップ頻度、冗長性など様々なデータの特性を
考慮し、目的に合ったデータストアを選択する

4. テナント分離モデルが与える影響

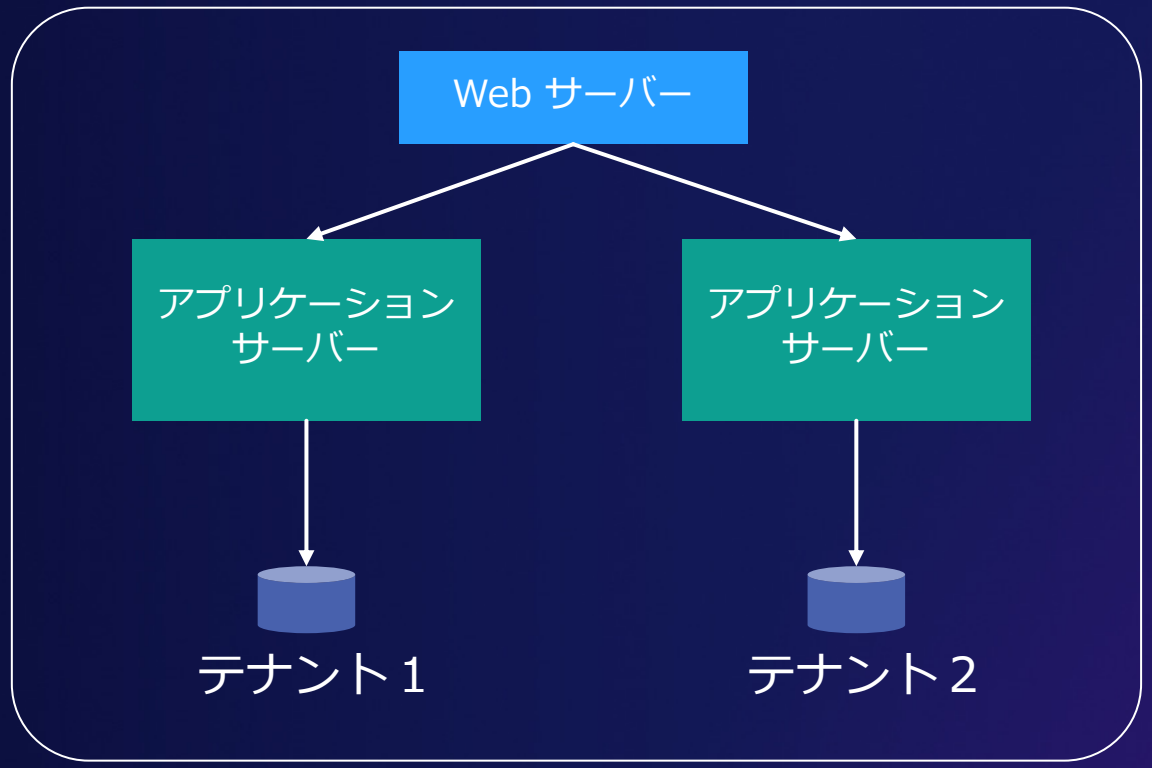
サイロモデル



プールモデル

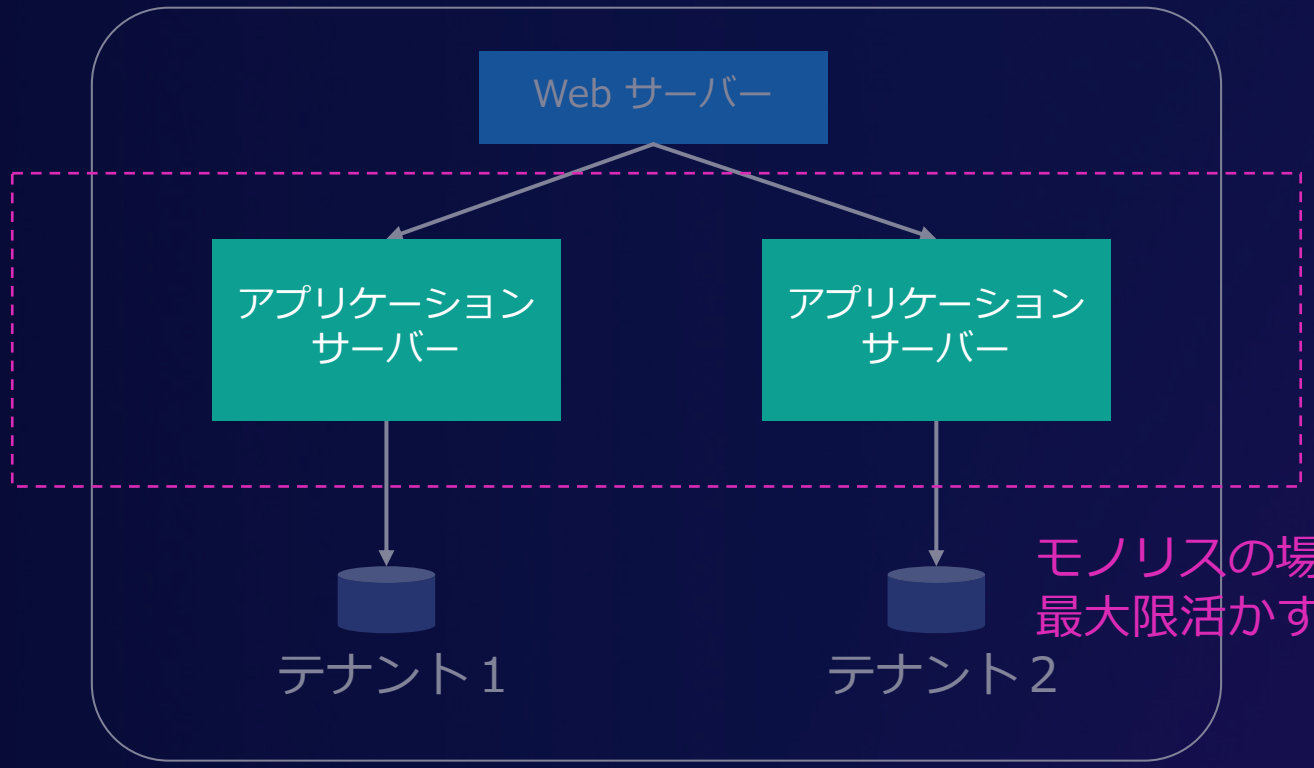
4. テナント分離モデルが与える影響

ブリッジモデル



4. テナント分離モデルが与える影響

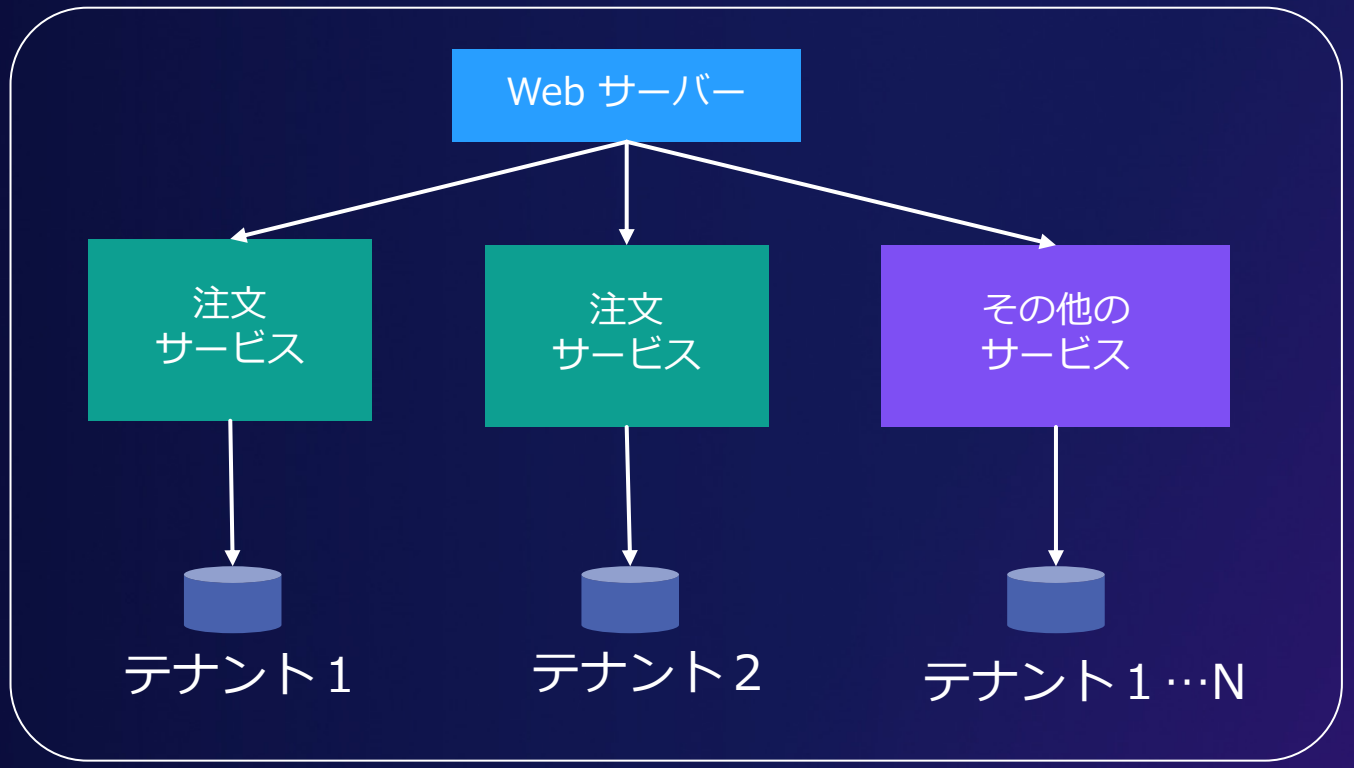
ブリッジモデル



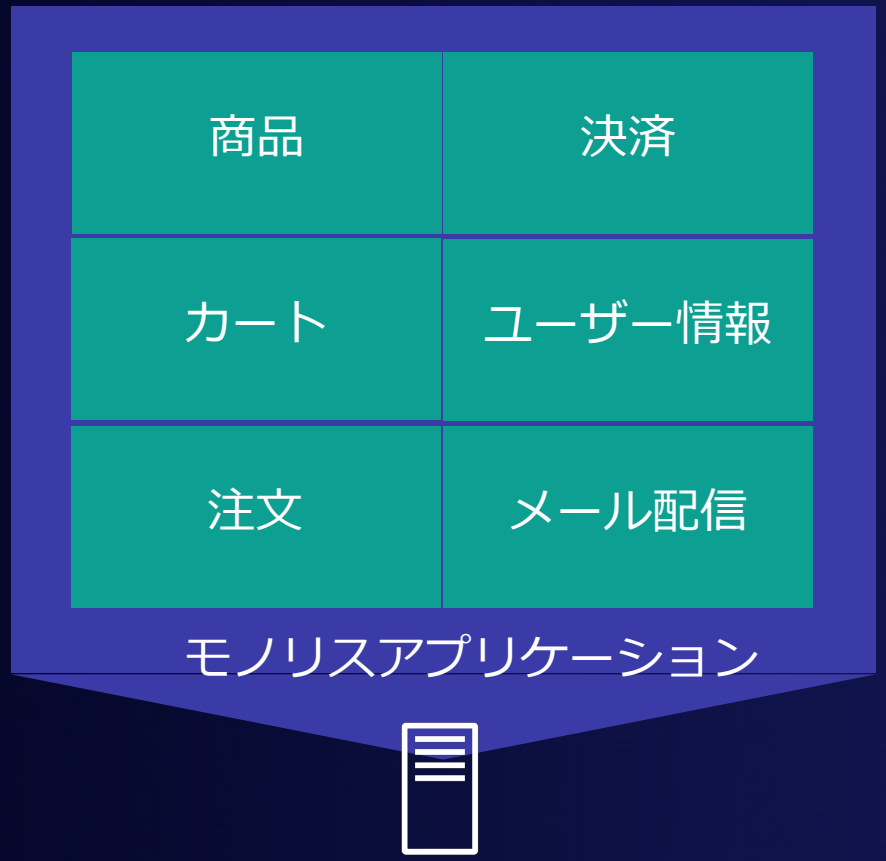
モノリスの場合、ブリッジモデルの利点を
最大限活かすことができない

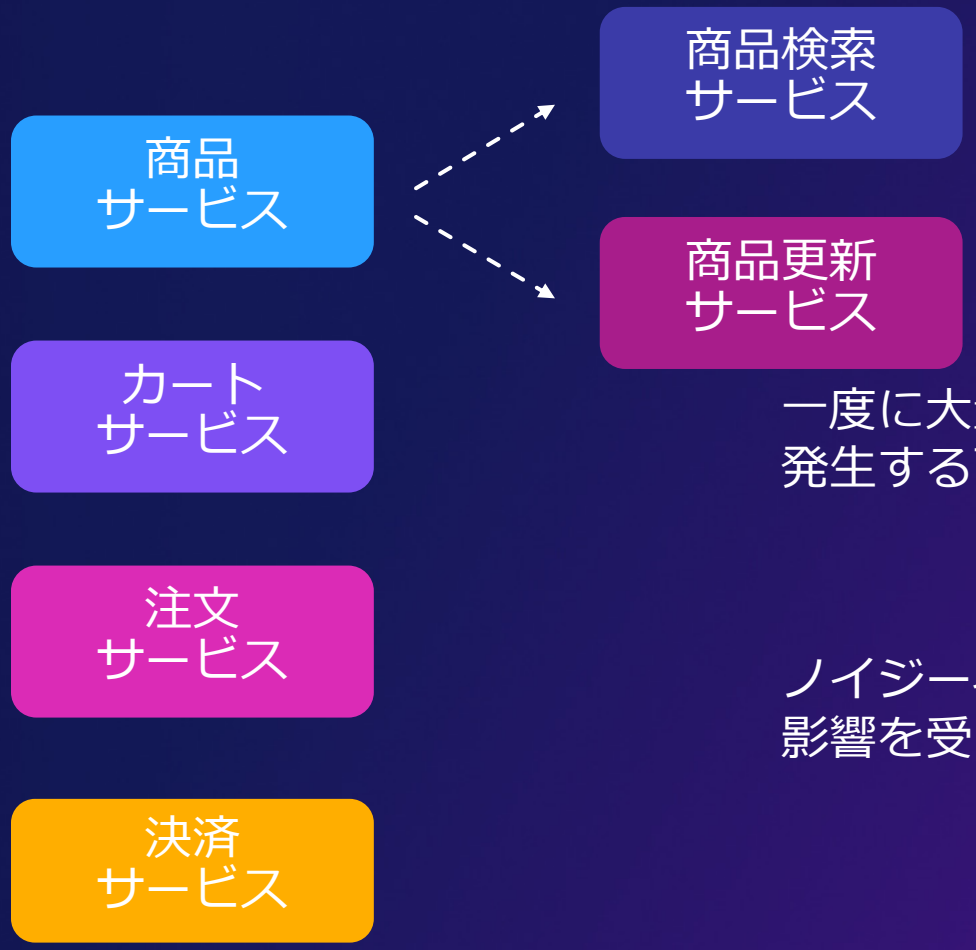
マイクロサービスごとにデプロイモデルを選択

ブリッジモデル



ex. パフォーマンス要件を考慮に入れる



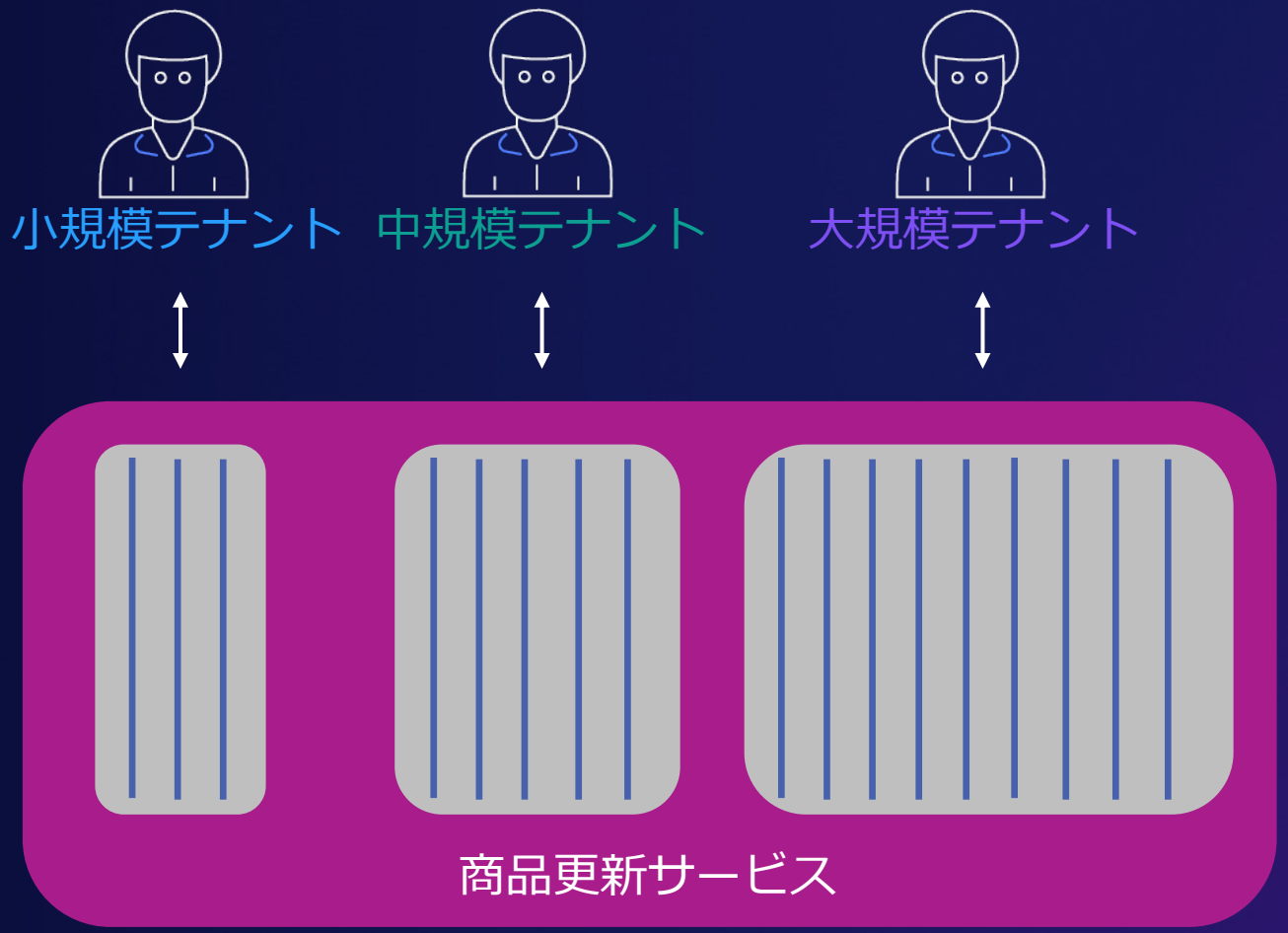


一度に大量の更新が
発生する可能性あり

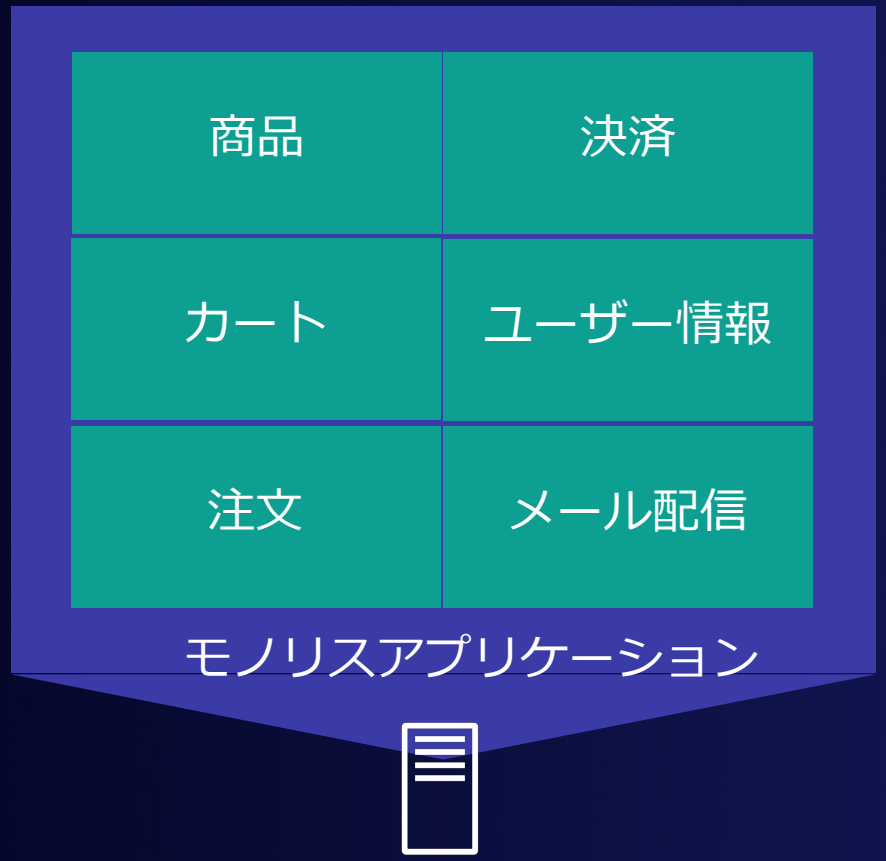
＋

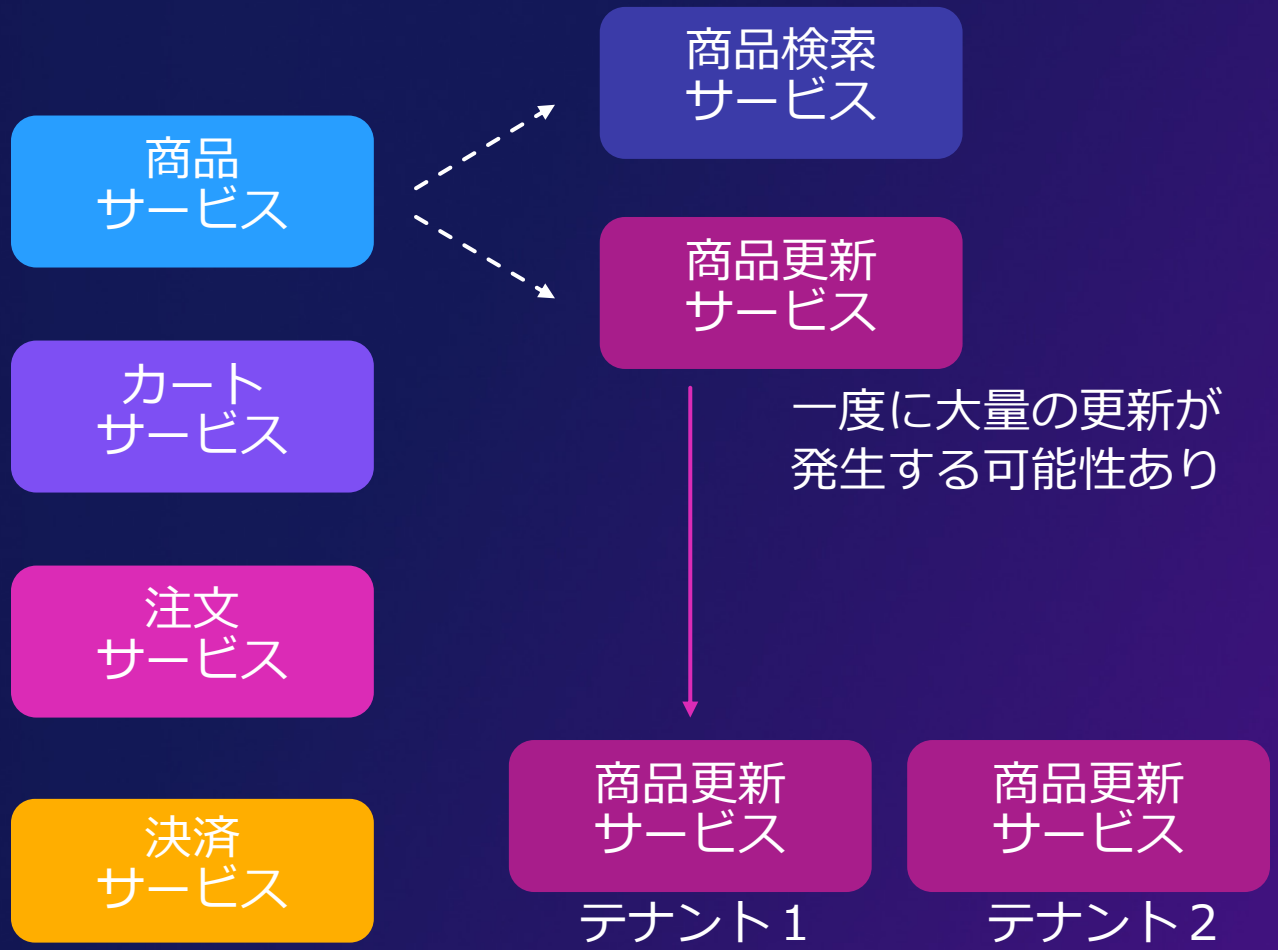
ノイジーネイバーの
影響を受けやすい

ノイジーネイバーのリスク



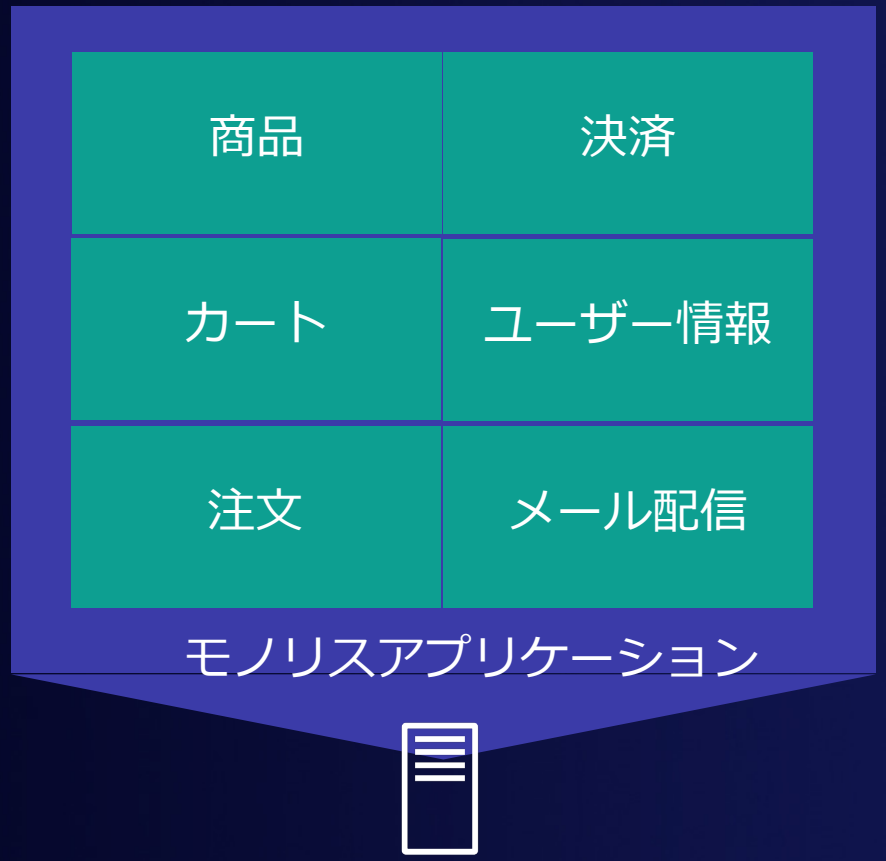
テナント分離要件がサービス境界となる

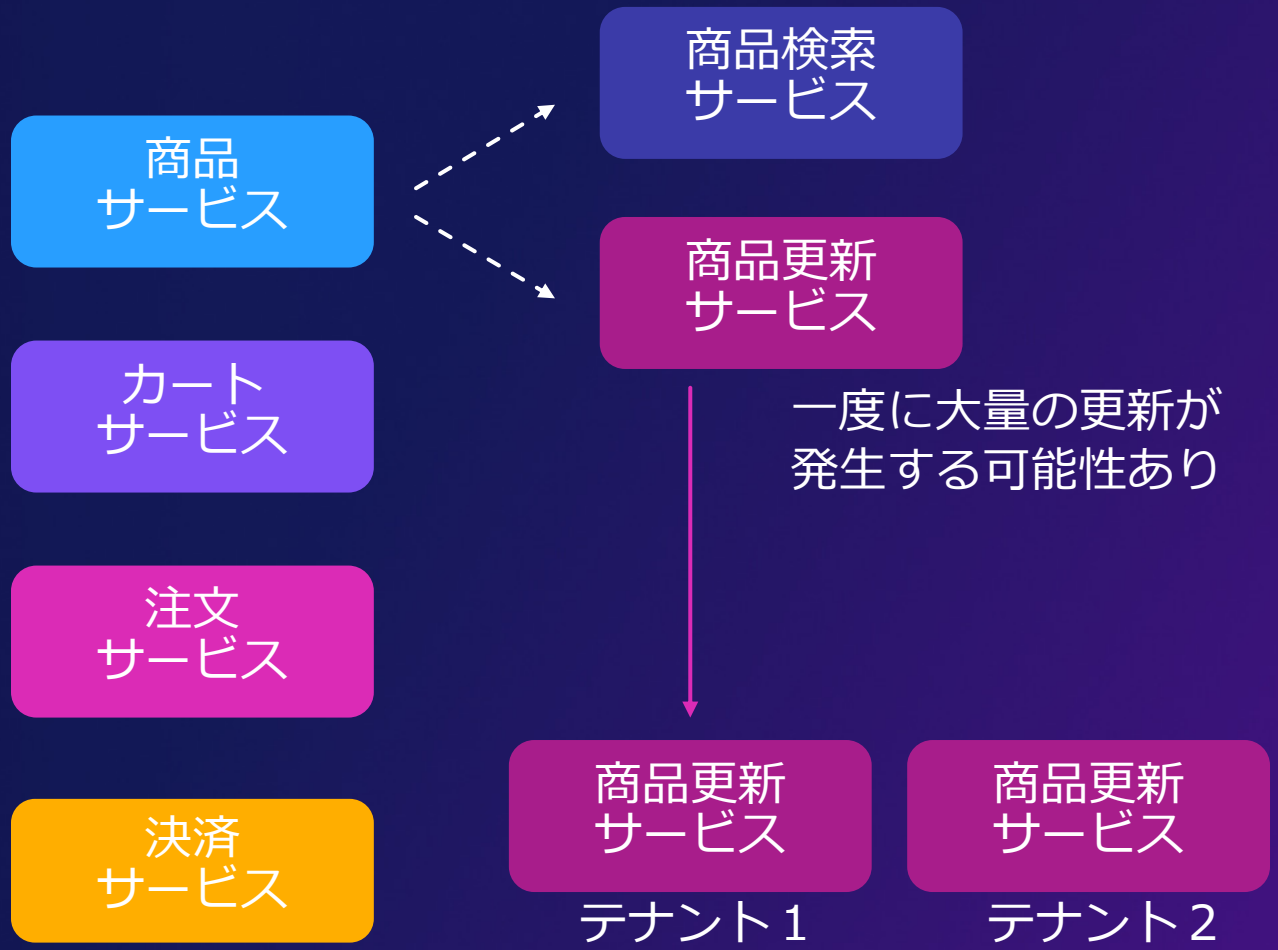




サイロモデルにすることによって
ノイジーネイバーのリスクを排除

テナント分離要件がサービス境界となる





サイロモデルにすることによって
ノイジーネイバーのリスクを排除

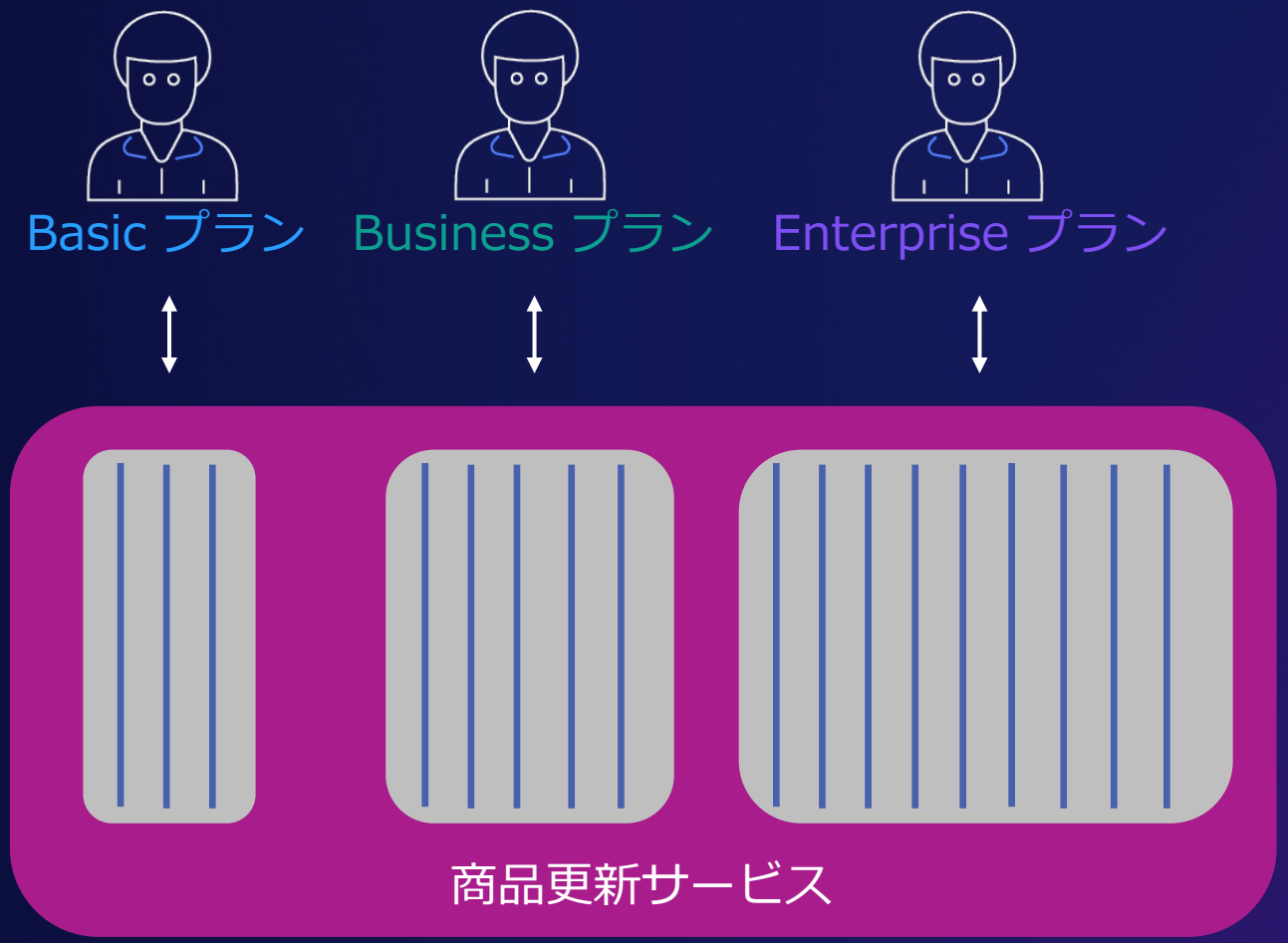
各ティアに提供する価値から考える

Basic プラン 月額 980 円	Business プラン 月額 2,980 円	Enterprise プラン 月額 9,980 円
最大 5 ユーザーまで利用可能	最大 15 ユーザーまで利用可能	最大 50 ユーザーまで利用可能
SLA なし	サービス稼働率 99.9 %を保証	サービス稼働率 99.999 %を保証
最大 100 商品まで登録可能	最大 500 商品まで登録可能	商品登録数の制限なし
一部機能の制限あり	全ての機能を利用可能 ※商品のバッチ更新は一度に 50 点まで	全ての機能を利用可能 ※商品のバッチ更新は一度に 500 点まで
カスタマーサポート 3 営業日以内に返信	カスタマーサポート 24 時間以内に返信	カスタマーサポート 1 時間以内に返信

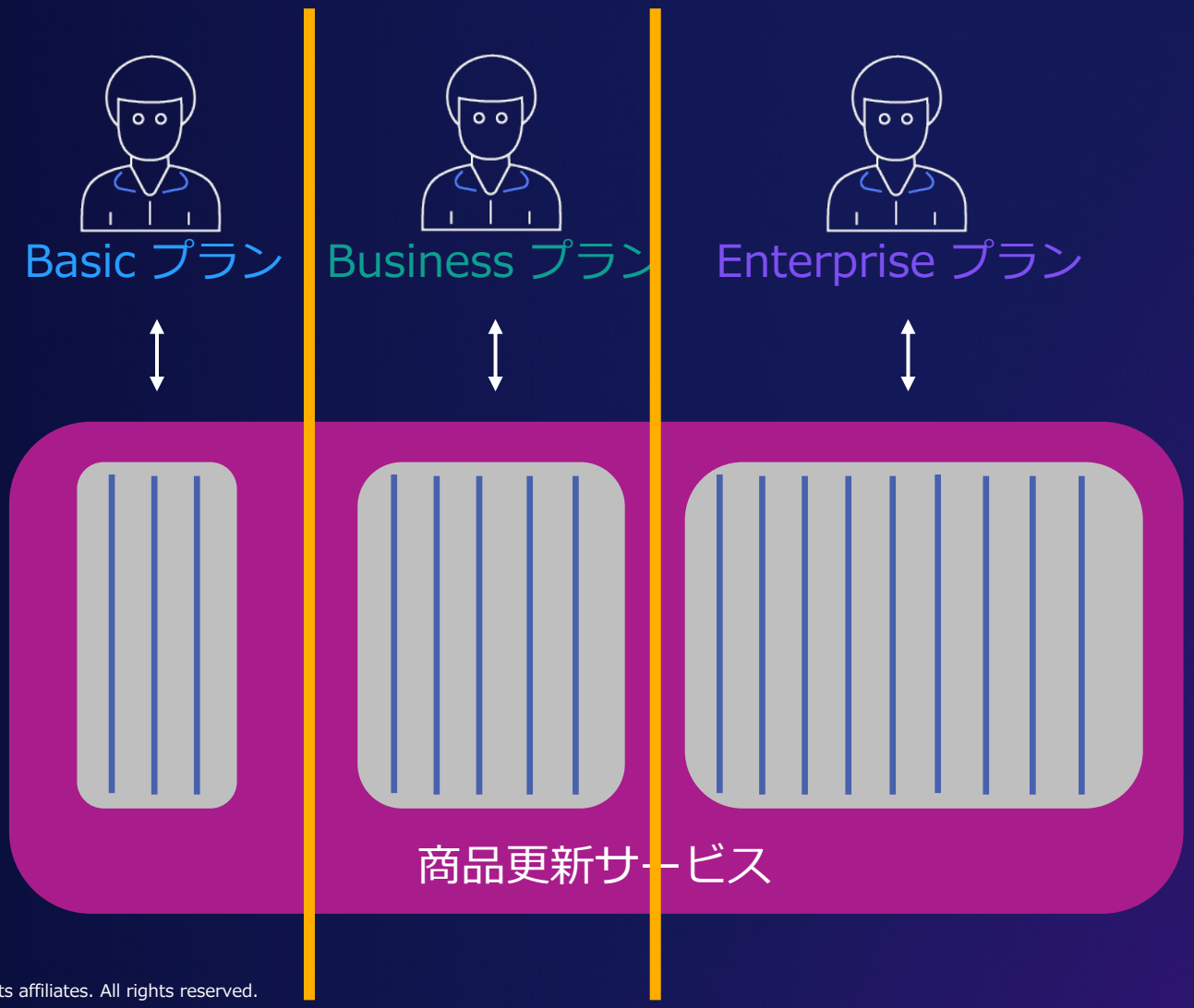
各ティアに提供する価値から考える

Basic プラン 月額 980 円	Business プラン 月額 2,980 円	Enterpriseプラン 月額 9,980 円
最大 5 ユーザーまで利用可能	最大 15 ユーザーまで利用可能	最大 50 ユーザーまで利用可能
SLA なし	サービス稼働率 99.9 %を保証	サービス稼働率 99.999 %を保証
最大 100 商品まで登録可能	最大 500 商品まで登録可能	商品登録数の制限なし
一部機能の制限あり	全ての機能を利用可能 ※商品のバッチ更新は一度に 50 点まで	全ての機能を利用可能 ※商品のバッチ更新は一度に 500 点まで
カスタマーサポート 3 営業日以内に返信	カスタマーサポート 24 時間以内に返信	カスタマーサポート 1 時間以内に返信

提供する体験を損なわないように境界を引く



提供する体験を損なわないように境界を引く



（再掲）多様なペルソナのサポート

ペルソナ 1：1000名規模のエンタープライズ企業

- データは他のテナントとは物理的に分離してほしい
- プライベートなネットワークでサービスを利用したい
- 可用性は 99.999 %以上が必要
- コストは惜しまない

SaaS

- セキュリティは最低限でOK
- 可用性は 99 %以上であればよい
- なるべくコストを抑えたい
- 一度に大量のデータをアップロードできるようにしてほしい

ペルソナ 2：十数名規模のスタートアップ

まとめ

まとめ

- マイクロサービスアーキテクチャと SaaS には高い親和性がある

まとめ

- マイクロサービスアーキテクチャと SaaS には高い親和性がある
- SaaS 特有の観点を考慮するとマイクロサービスの利点が増幅される

まとめ

- マイクロサービスアーキテクチャと SaaS には高い親和性がある
- SaaS 特有の観点を考慮するとマイクロサービスの利点が増幅される
- SaaS が提供すべき価値をベースに、サービス境界の引き方を考える

まとめ

- マイクロサービスアーキテクチャと SaaS には高い親和性がある
- SaaS 特有の観点を考慮するとマイクロサービスの利点が増幅される
- SaaS が提供すべき価値をベースに、サービス境界の引き方を考える
- ティアの設計とマイクロサービスの分割方法は互いに影響する

まとめ

- マイクロサービスアーキテクチャと SaaS には高い親和性がある
- SaaS 特有の観点を考慮するとマイクロサービスの利点が増幅される
- SaaS が提供すべき価値をベースに、サービス境界の引き方を考える
- ティアの設計とマイクロサービスの分割方法は互いに影響する
- 現在の企業のステージ、市場規模、目指すゴールに応じて、
  慎重にマイクロサービスを導入すべきか検討する

まとめ

- マイクロサービスアーキテクチャと SaaS には高い親和性がある
- SaaS
- SaaS
- ティ
- 現在
  慎重にマイクロサービスを導入すべきか検討する

Q. マイクロサービスは SaaS 開発者を救うか？

A. マイクロサービスは万能ではないが、
ARR 100 億円の規模までビジネスを
スケールさせる上で、重要な鍵となり得る

Thank you!