
ORACLE®

#odddtky

Oracle DBA & Developer Days 2014

オラクル・コンサルが語る！
Oracle Coherence活用ノウハウ、そして最新ユースケース

日本オラクル株式会社
コンサルティングサービス事業統括
テクノロジーコンサルティング統括本部
米田 穣

Fusion Middleware事業統括
ビジネス推進本部
杉 達也

ORACLE

for your **Skill**

使える実践的なノウハウがここにある

Oracle
DBA & Developer Days
2014

- 以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

自己紹介

- 2007年より現職
  - 前職では、検索エンジンや医療システムの開発/保守を経験
- Coherence買収後の立ち上げメンバーとして参画して以来、歴7年目
  - アーキテクチャ設計、実装を得意としCohereceを活用したシステムの設計から実装工程までを重点的に支援
- 実績
  - 小売：家電量販店ECサイト
  - サービス：ショッピングモール
  - 航空：チケット予約・購入、空席・運賃照会システム
  - 他多数…

アジェンダ

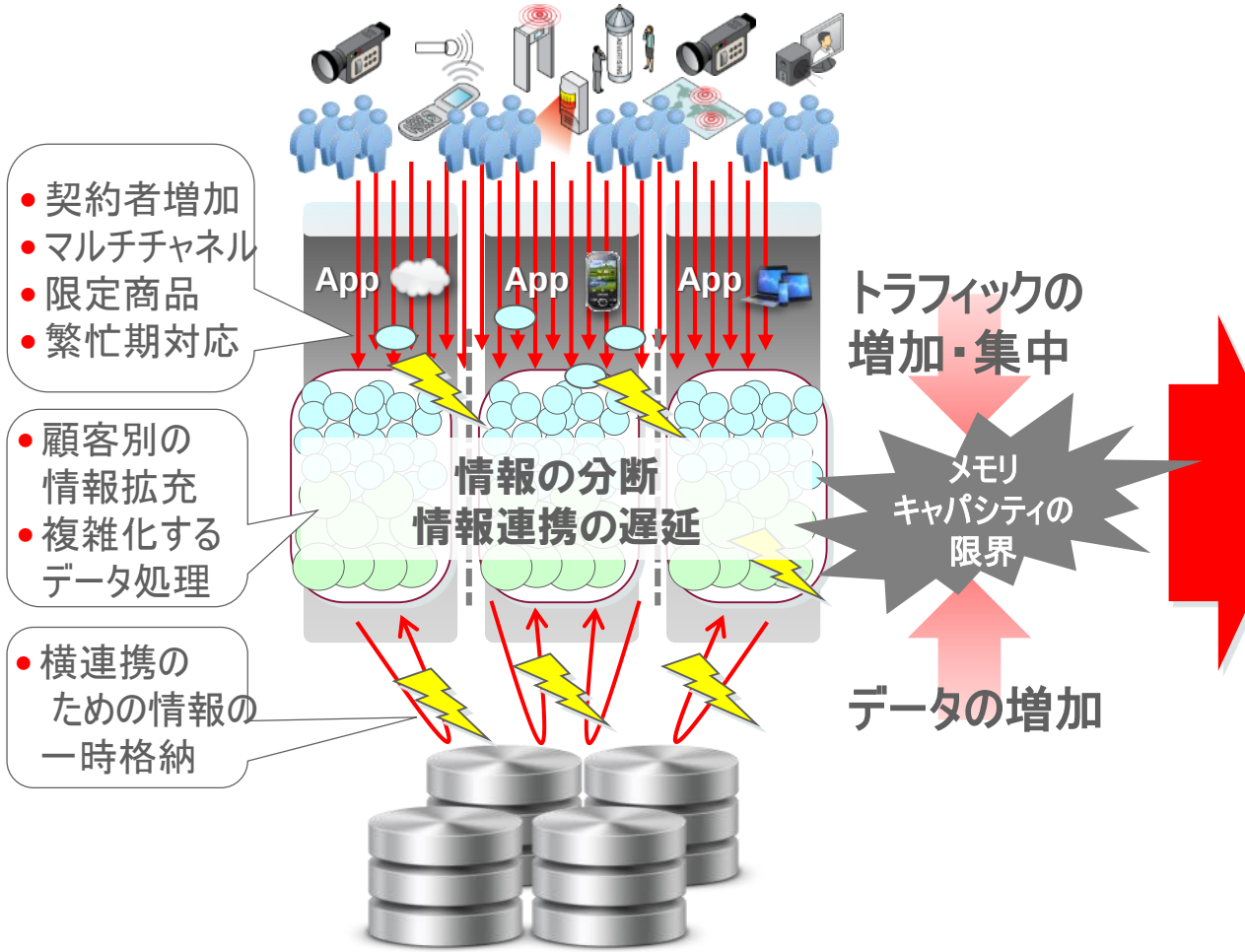
1. Oracle Coherenceチューニングの勘所
2. データ登録の種類と課題点
3. 実際のプロジェクトで用いた高速化アーキテクチャー

アジェンダ

1. Oracle Coherenceチューニングの勘所
2. データ登録の種類と課題点
3. 実際のプロジェクトで用いた高速化アーキテクチャー

Oracle Coherence とは

■ 従来構造で起こる典型的な課題

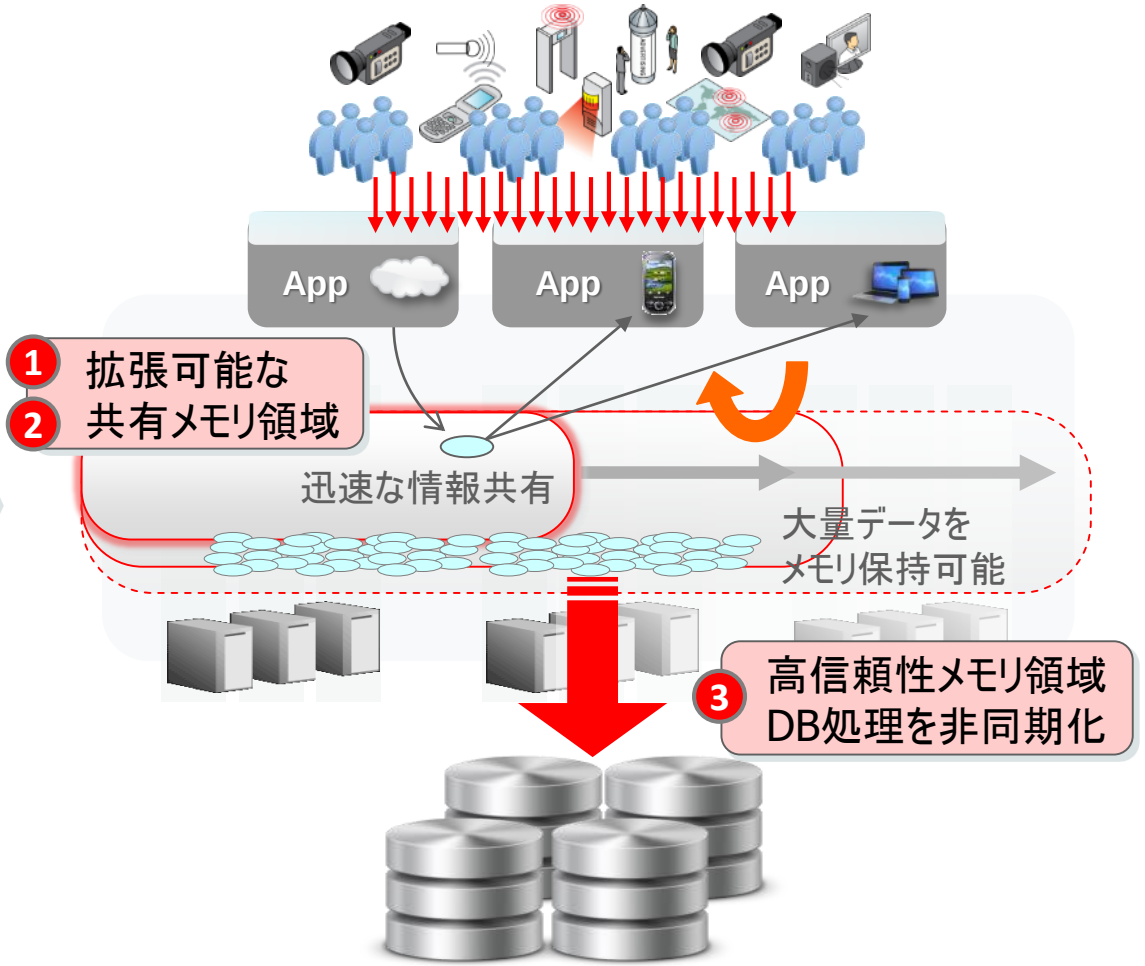


- 契約者増加
- マルチチャネル
- 限限定商品
- 繁忙期対応

- 顧客別の情報拡充
- 複雑化するデータ処理

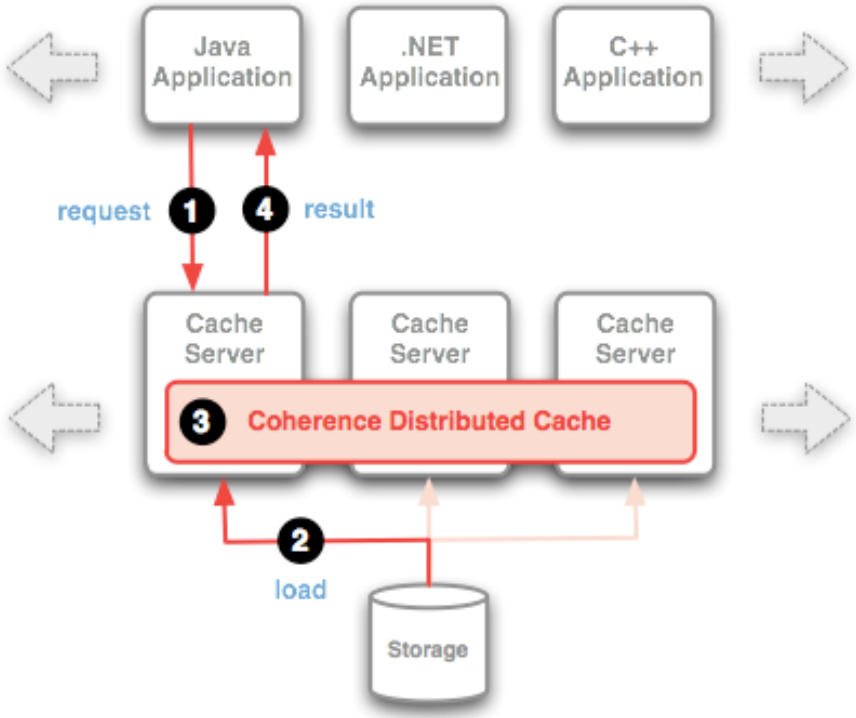
- 横連携のための情報の一時格納

■ Oracle Coherence 構成例

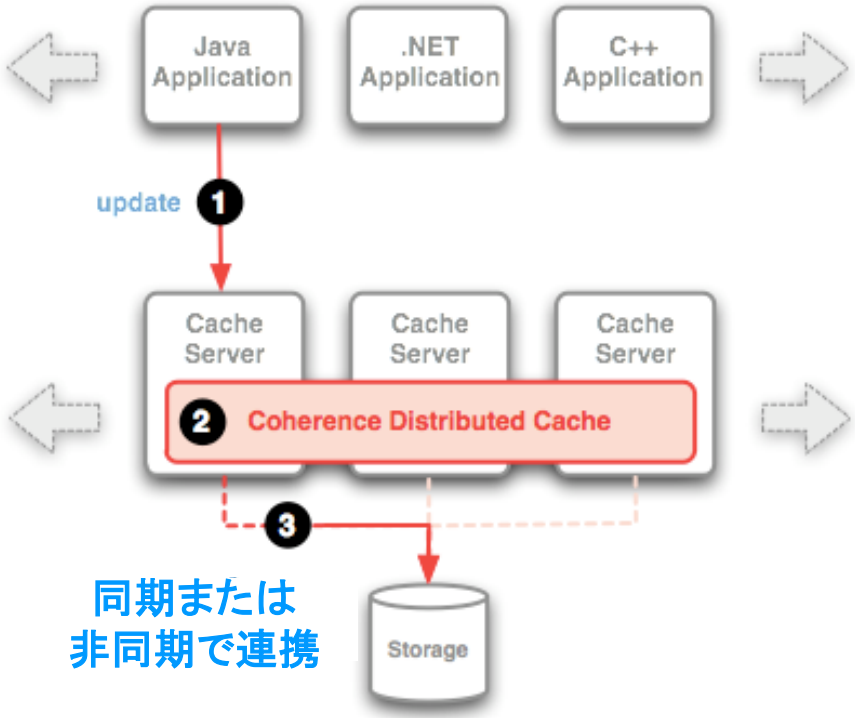


前提知識
バックエンドとの連携 – 同期/非同期連携

■ 読み取り処理
- Coherence経由でバックエンドから取得（Read-Through）
- 指定された時間間隔でキャッシュを自動更新（Refresh-Ahead）

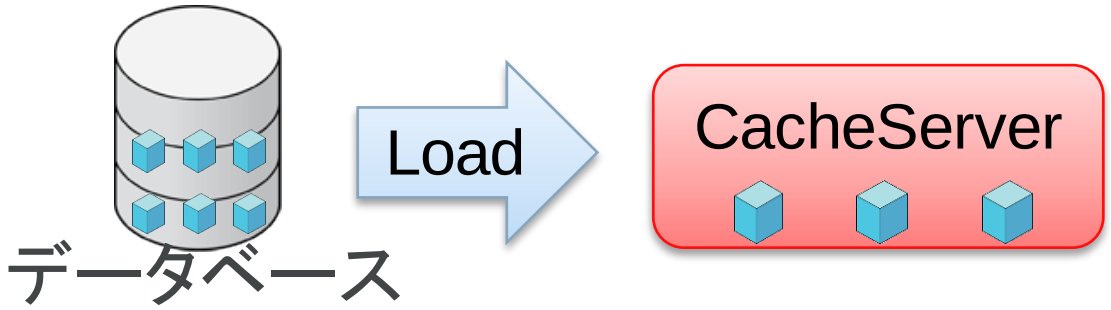


■ 書き込み処理
- Coherence経由で同期型で更新（Write-Through）
- 指定された時間間隔で非同期更新（Write-Behind）



Oracle Coherenceチューニングの勘所
本日の題材「初期データローダ」について

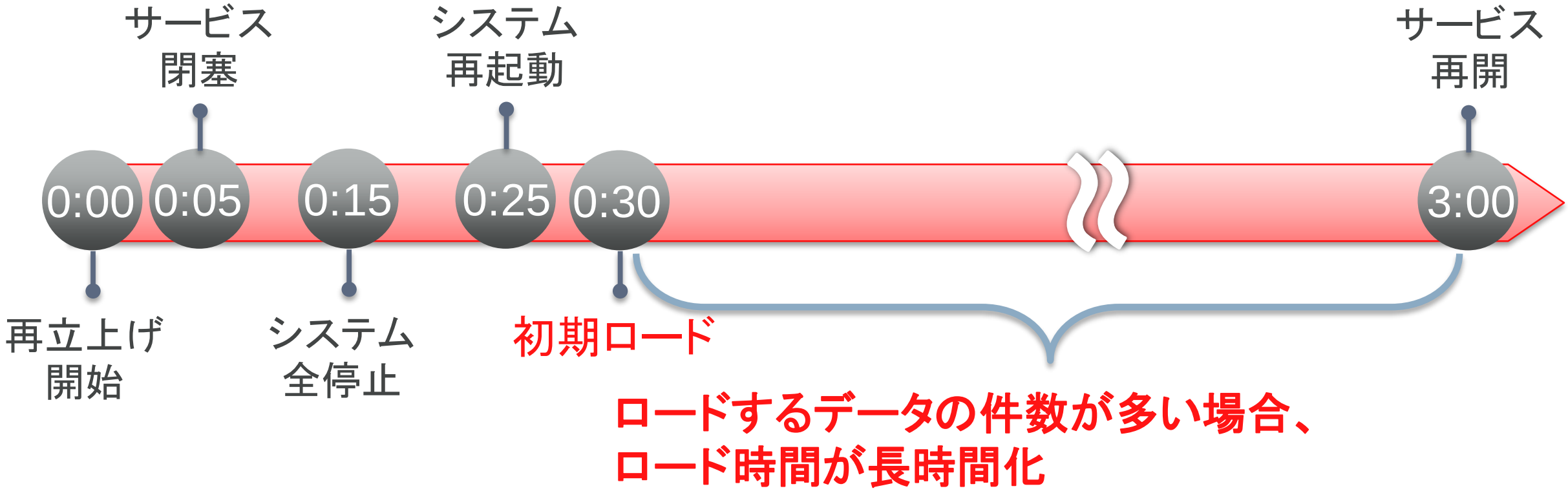
- データベース又はファイル等の永続化されたデータを、Coherence上にロードするためのツール
- システム起動時にCoherence上にデータをロードする際に使用
  - ユースケース：Coherenceを高速なデータストレージとして利用する場合
    - 例：マスタデータの格納
  - 非ユースケース：Coherenceをキャッシュとして利用する場合
    - 例:セッションデータの格納



初期ローダの重要性

初期データローダ低速時の影響

- システムの立ち上げ時間からサービス開始までの時間短縮が求められる場合



Oracle Coherenceチューニングの勘所
高速化のポイント

『チューニング』というとパラメータや GC などが思い浮かびがちだが、以下の観点を“設計時点で織り込む”ことがポイント

- 如何に通信回数を少なくするか？
- 如何に並列に処理させるか？
- 如何にトラフィックを軽減させるか？

- ≪例≫ get→getAll、put→putAll、invoke→invokeAll

Oracle Coherenceチューニングの勘所

チューニング前後の計測時間の比較

- 測定結果

業種	データサイズ	チューニング以前	チューニング後
サービス業	40GB	6時間 →	15分
通信業	200GB	12時間 →	40分

- 効果
  - ✓ 計画停止や障害からの復旧時間の短縮
  - ✓ 初期本番移行時間の短縮
  - ✓ テスト時間の短縮

アジェンダ

1. Oracle Coherenceチューニングの勘所
2. **データ登録の種類と課題点**
3. 実際のプロジェクトで用いた高速化アーキテクチャー

データ登録の種類と課題点
データ登録の種類

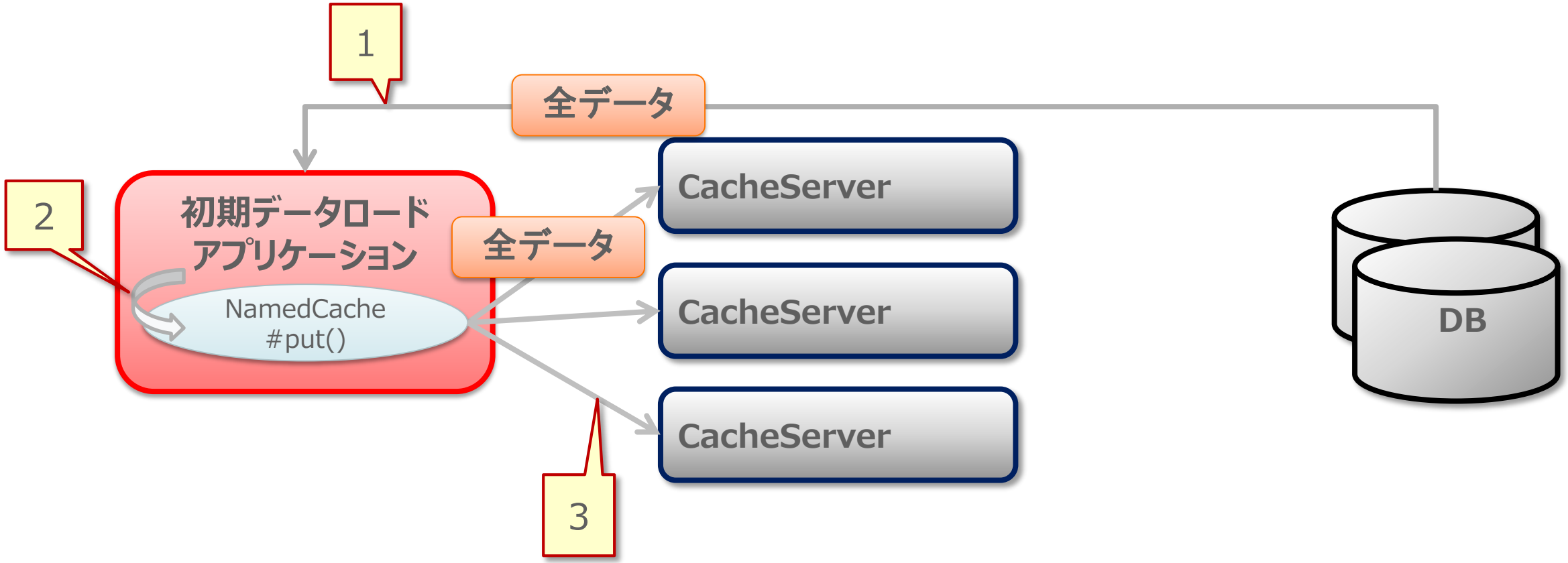
- Put APIを用いた独自アプリケーションを作成する方法
  1. putアプリケーション

- Put 以外のCoherence機能を有効活用する方法
  2. リードスルー・キャッシング
  3. PreloadProcessor
  4. pre-load

データ登録の種類と課題点

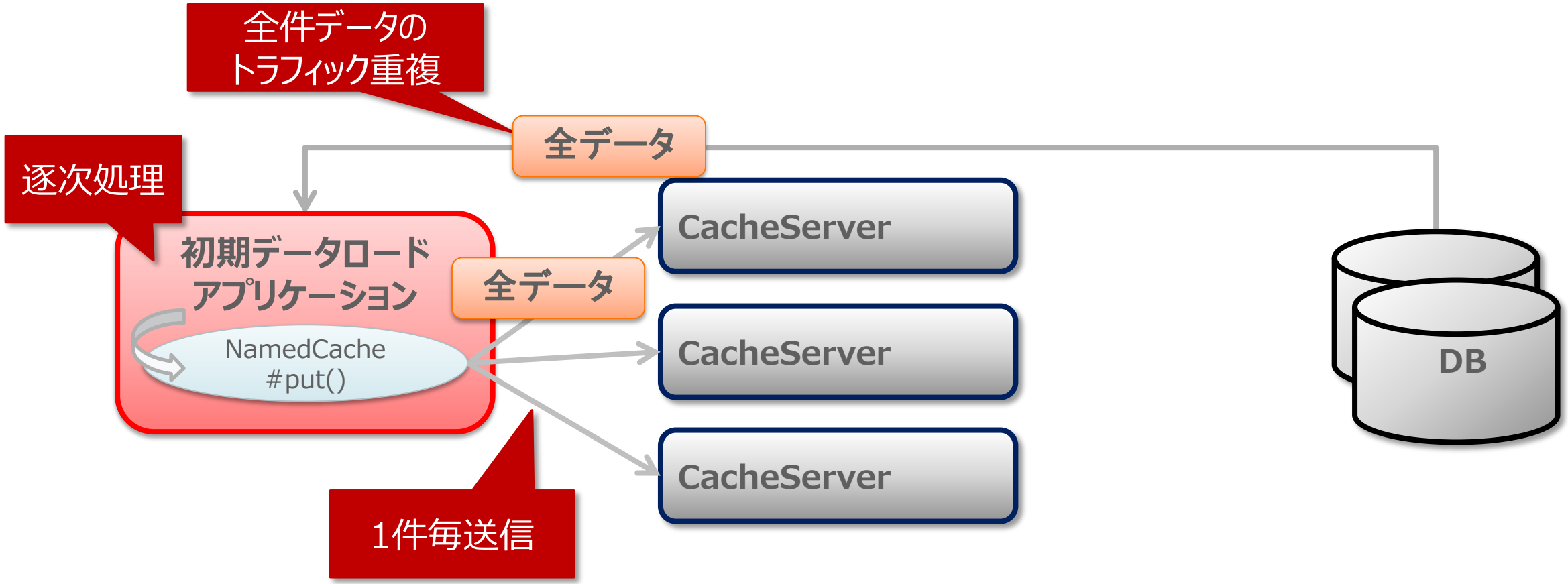
1. putアプリケーション

- JDBCとCoherenceAPIのputを用いた、最もシンプルな実装方法



データ登録の種類と課題点

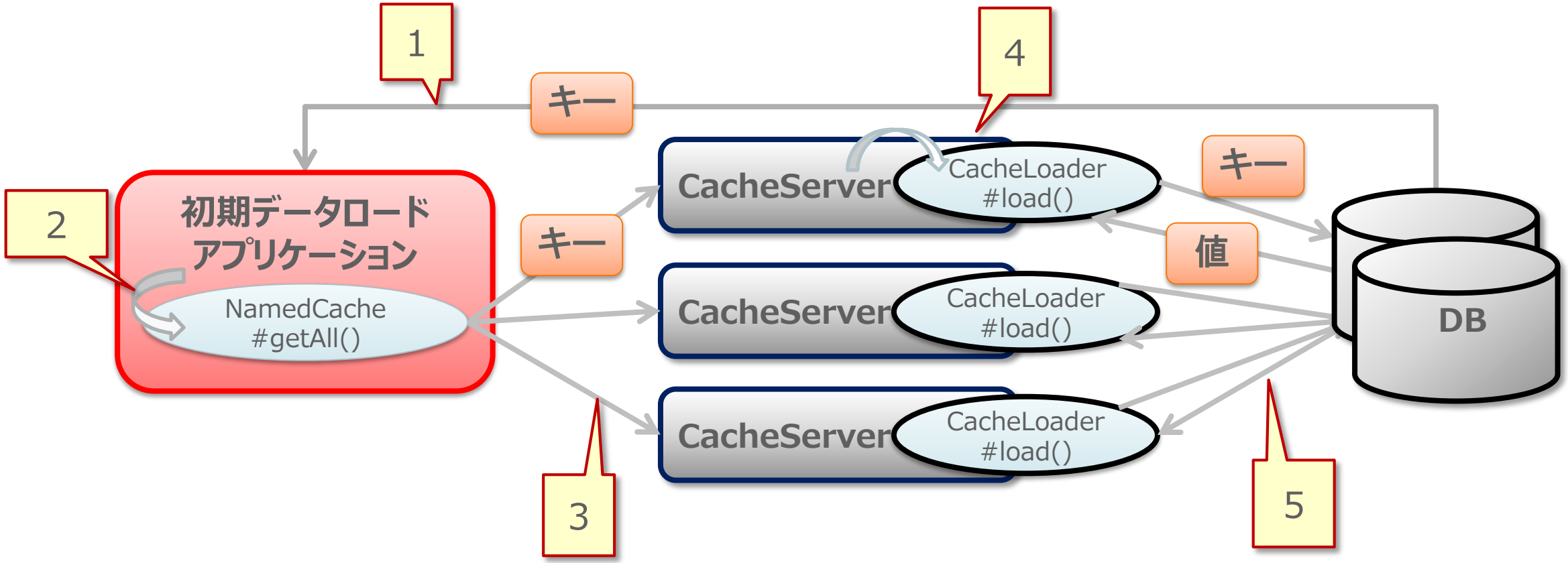
1. putアプリケーションの課題点



データ登録の種類と課題点

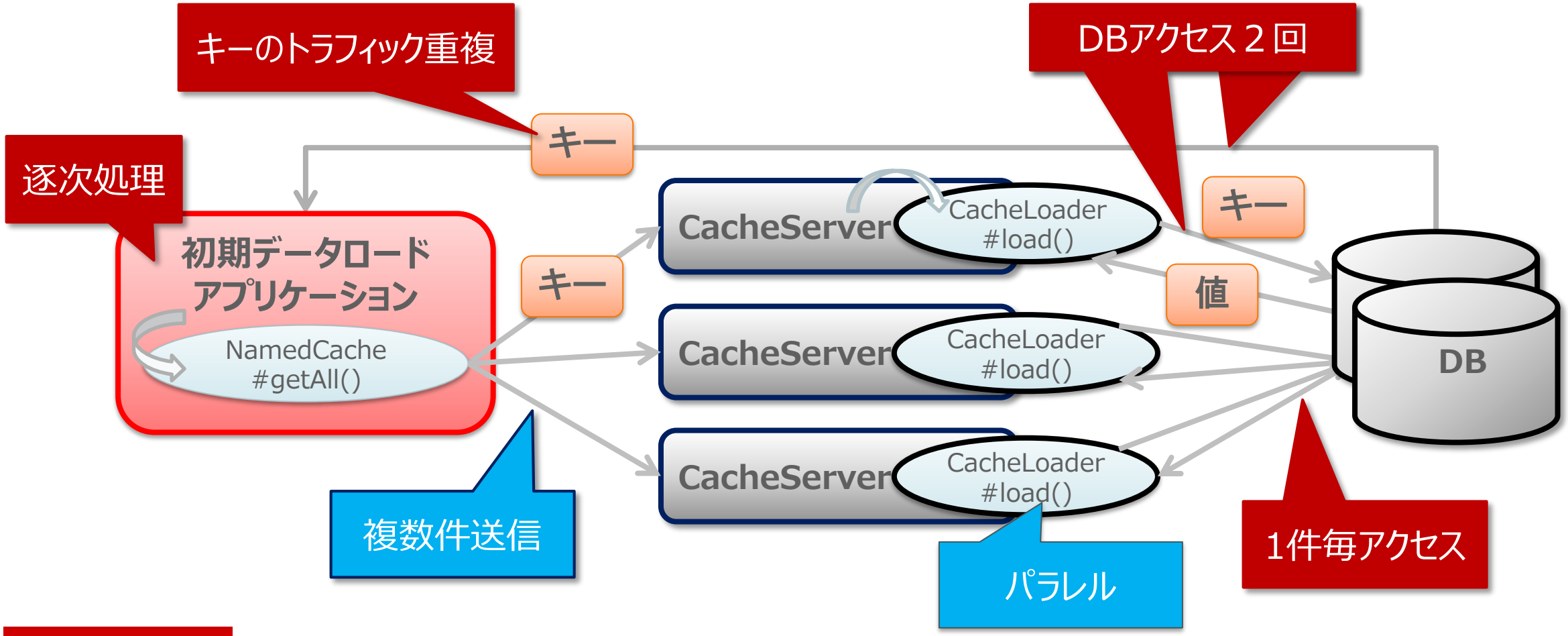
2. リードスルー・キャッシング

- CacheLoaderを活用し、各CacheServerでデータロードを並列実行



データ登録の種類と課題点

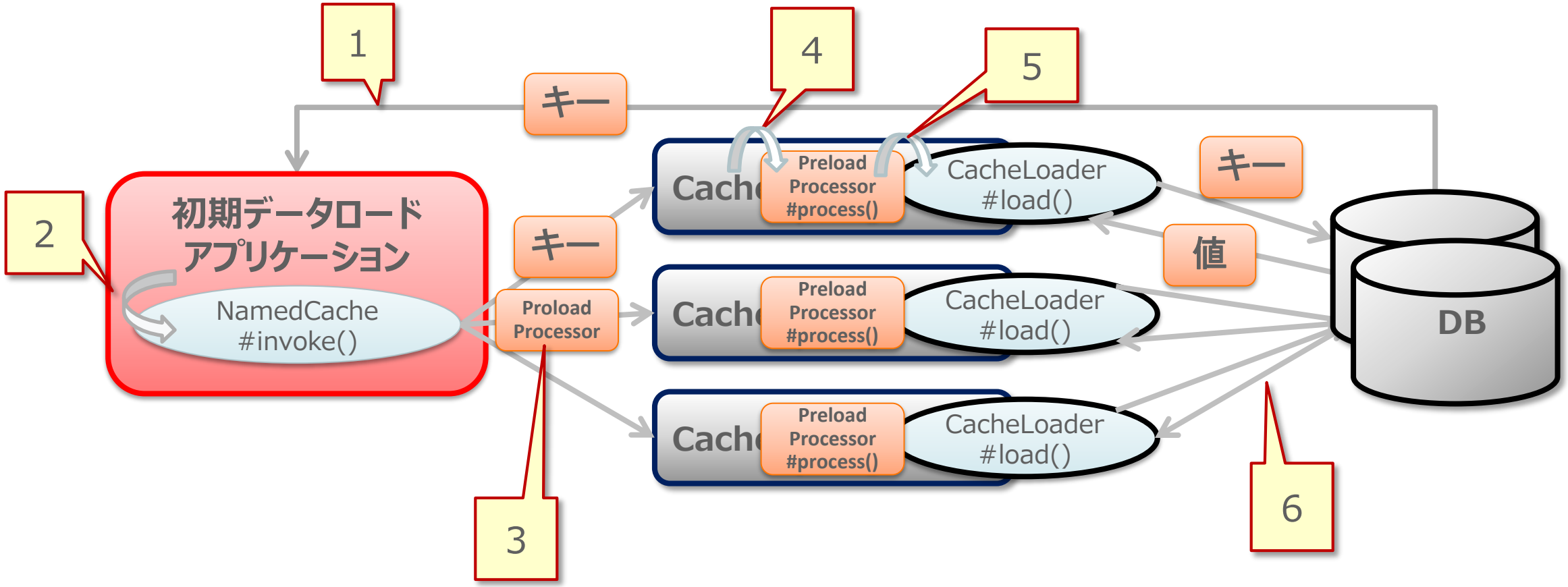
2. リードスルー・キャッシングの課題点



データ登録の種類と課題点

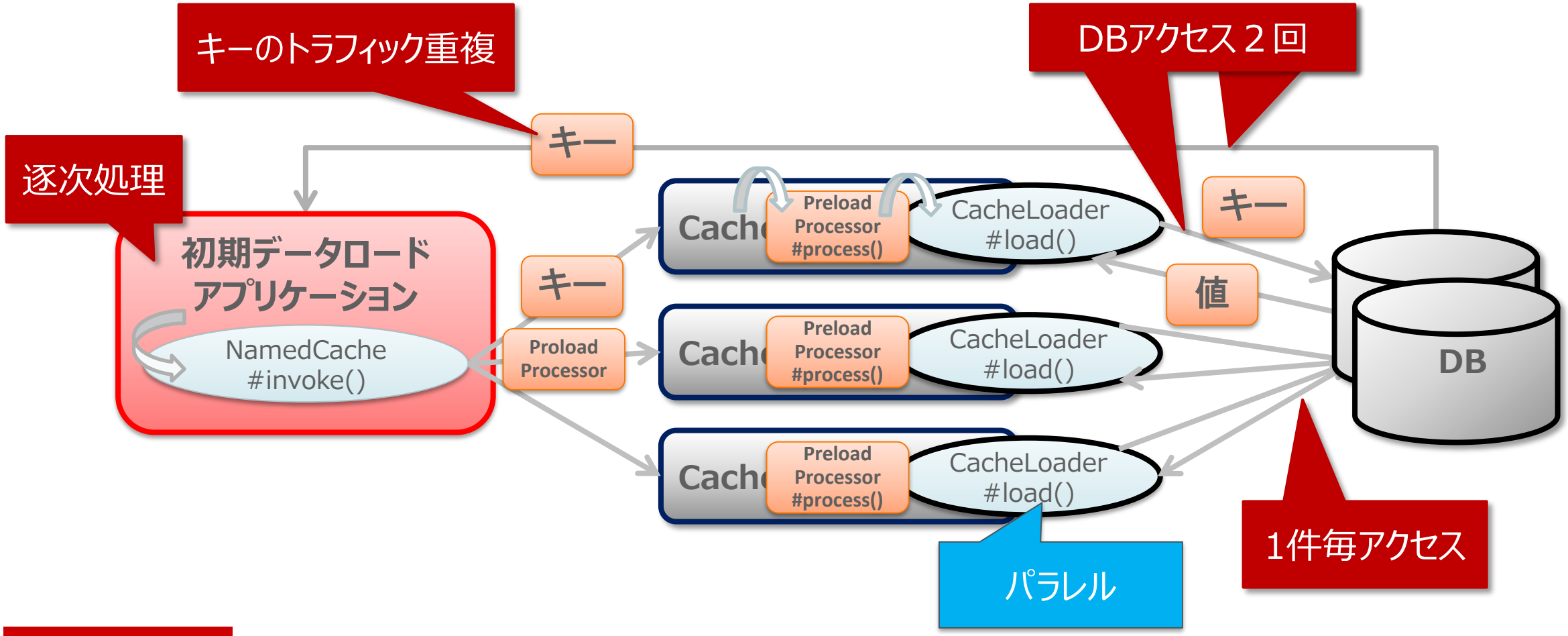
PreloadProcessor

- PreloadRequest を活用し、各CacheServerでデータロードを並列実行



データ登録の種類と課題点

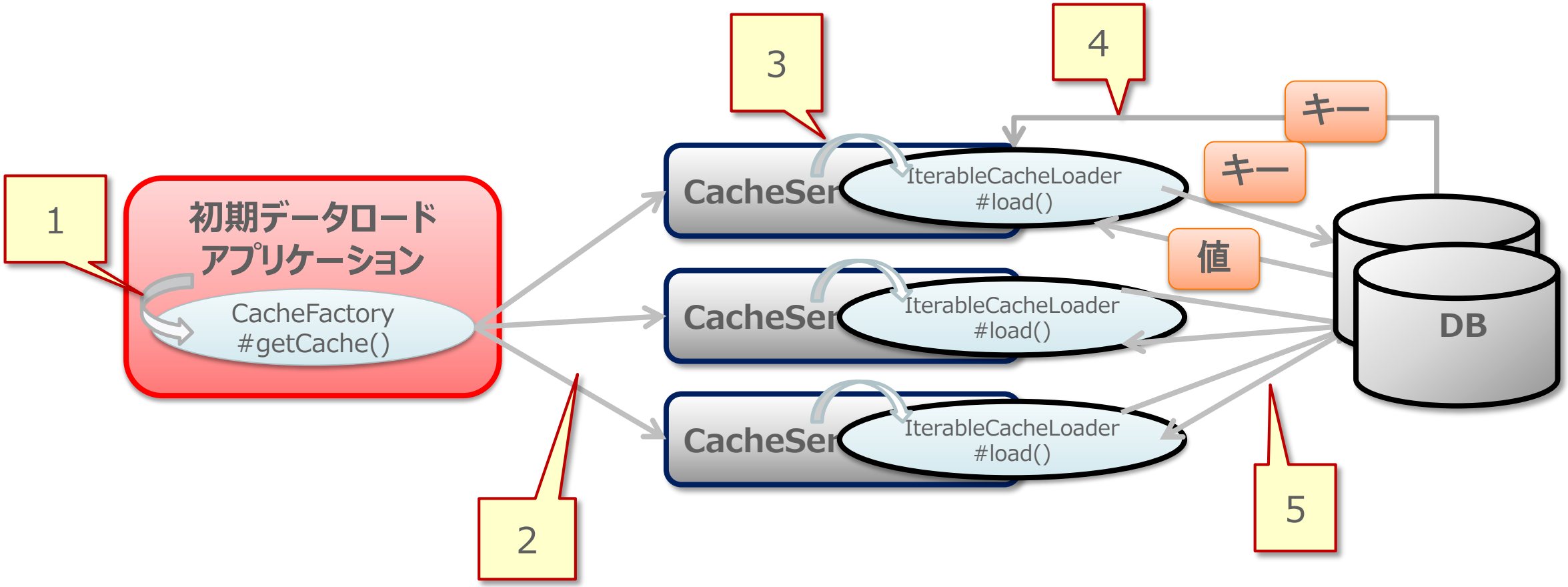
PreloadProcessorの課題点



データ登録の種類と課題点

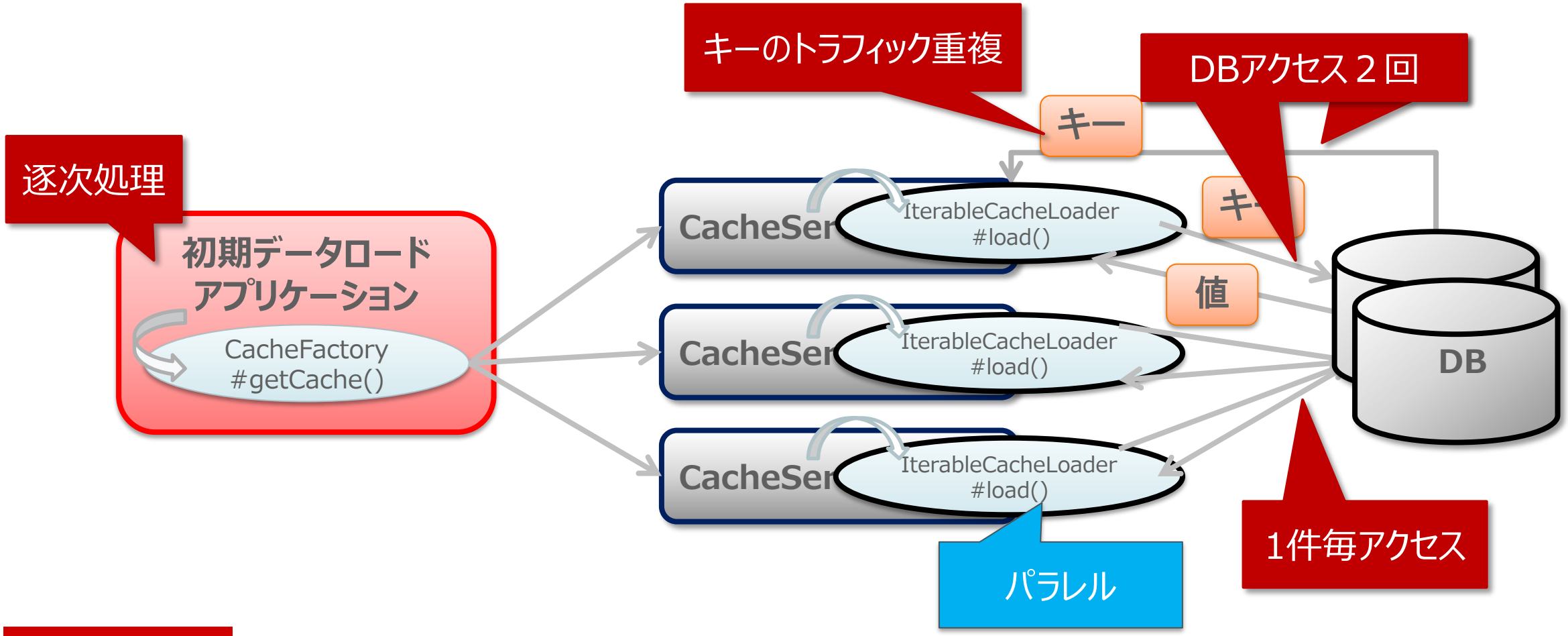
pre-load

- IterableCacheLoaderを活用し、各CacheServerでデータロードを並列実行



データ登録の種類と課題点

pre-loadの課題点



データ登録の種類と課題点

結局どれがいいのか？

- チューニングの観点からすると、まだまだ改善の余地がある

登録方式	処理実行方式 (クライアント側)	トラフィック データの重複	データベース アクセス回数	データの 同時取得件数
1. putアプリケーション	逐次	キー、値	1回	複数
2. リードスルー・キャッシング	逐次	キー	2回	1件ずつ
3. PreloadProcessor	逐次	キー	2回	1件ずつ
4. pre-load	逐次	キー	2回	1件ずつ

高速化へのアプローチ

- いかに通信回数を少なくするか？
  → データベースアクセス回数の削減
  → １回のアクセスの処理効率化(複数件取得)

- いかに並列に処理させるか？
  → クライアント側の並行処理化

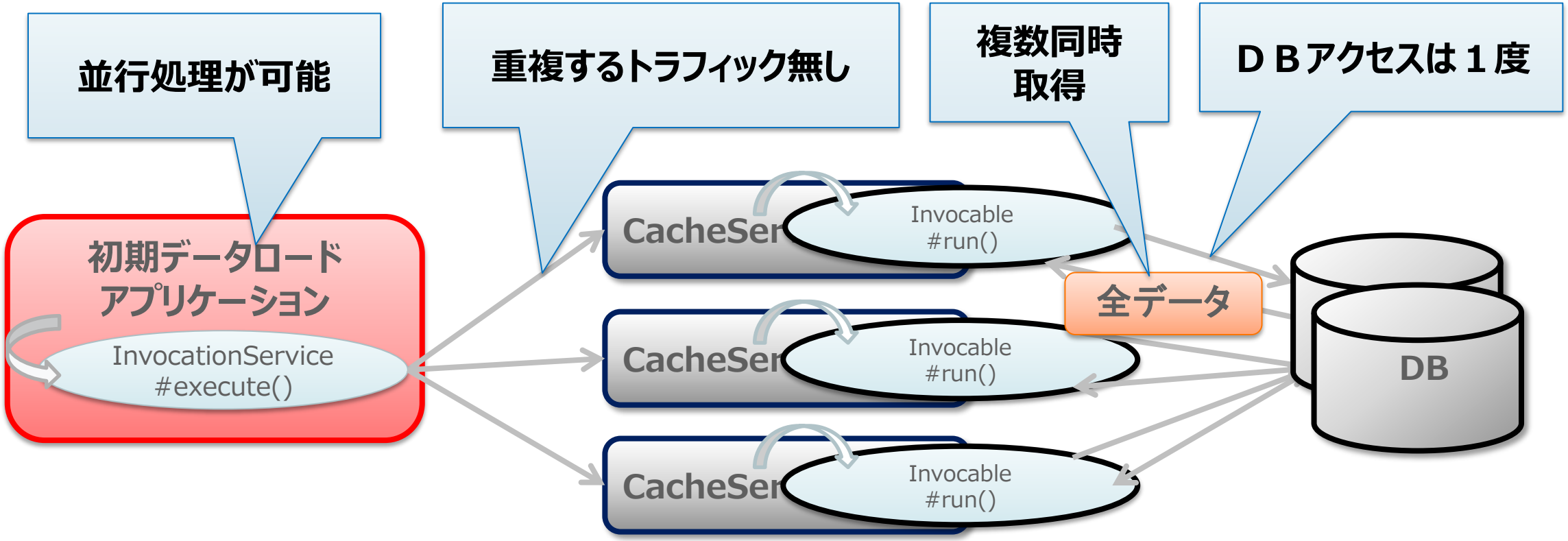
- いかにトラフィックを軽減させるか？
  → キー全件や、キーと値との両方が重複して送受信されないような方式の検討

アジェンダ

1. Oracle Coherence概要
2. データ登録の種類と課題点
3. **実際のプロジェクトで用いた高速化アーキテクチャー**

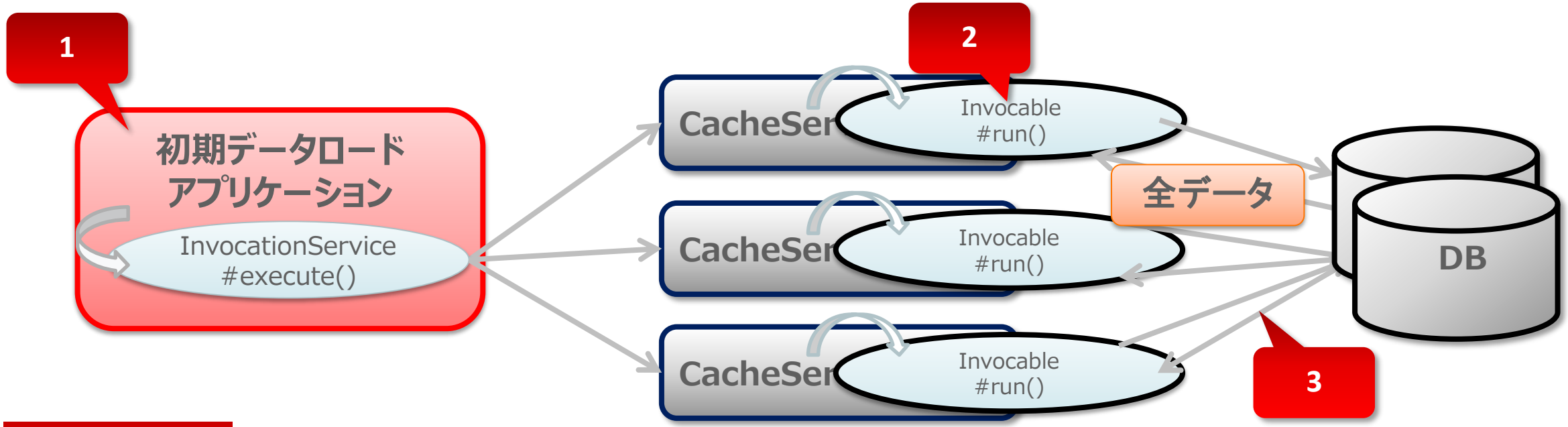
実際のプロジェクトで用いた高速化アーキテクチャー

新アーキテクチャーの改善点



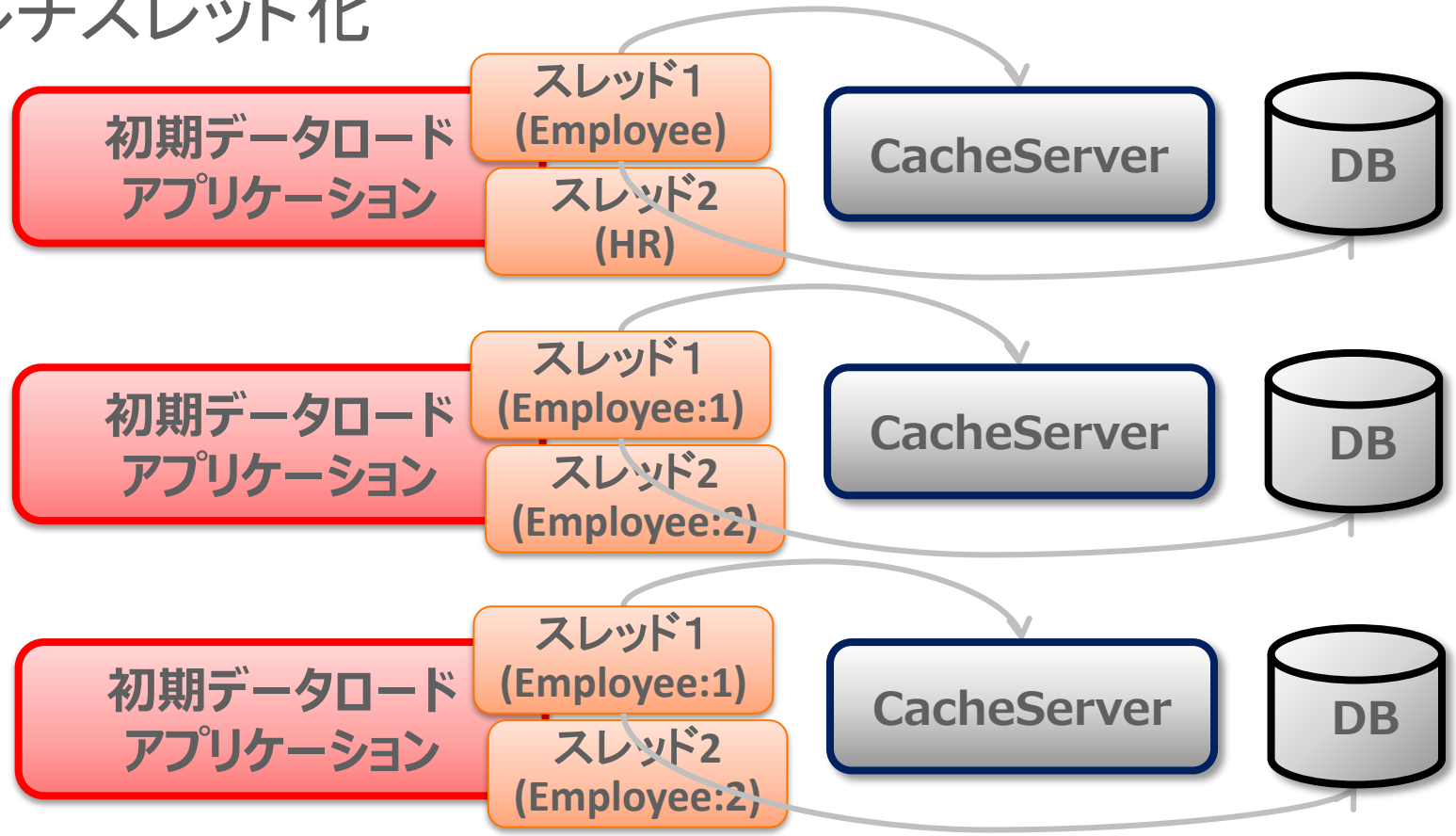
実装に向けた課題

1. クライアント側の処理並列化
2. 任意のロジックをCacheServer側で実行する方法
3. DBからのロードするデータをCacheServer毎に指示する方法



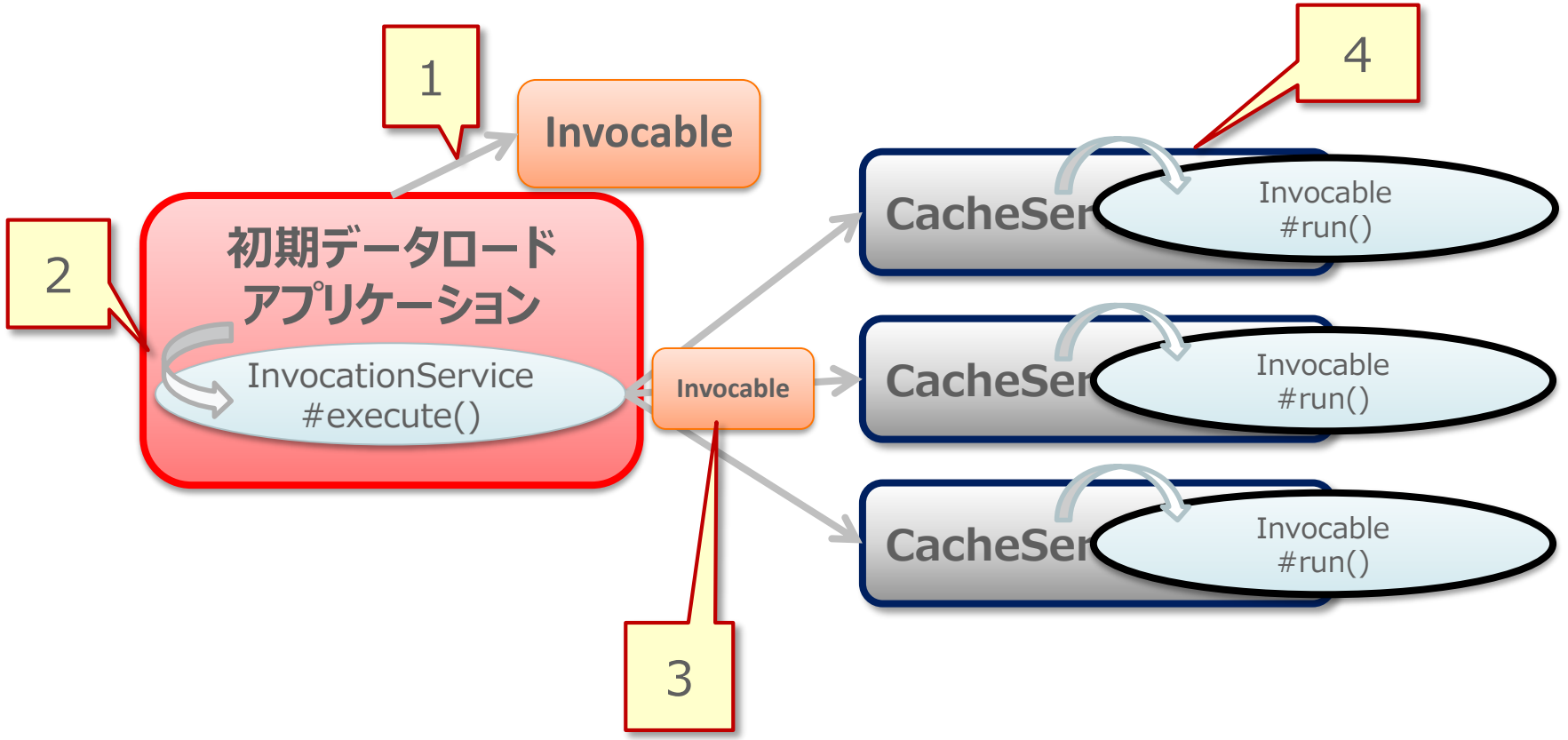
1. クライアント側の処理並列化

- クライアントとCacheServer間の処理を非同期化
- クライアント処理のマルチスレッド化
  - キャッシュ単位
  - キャッシュ分割単位
  - 役割単位



2.任意のロジックをCacheServer側で実行する方法

- InvocationService を用いると、Coherenceサイドでロジックを実行可能



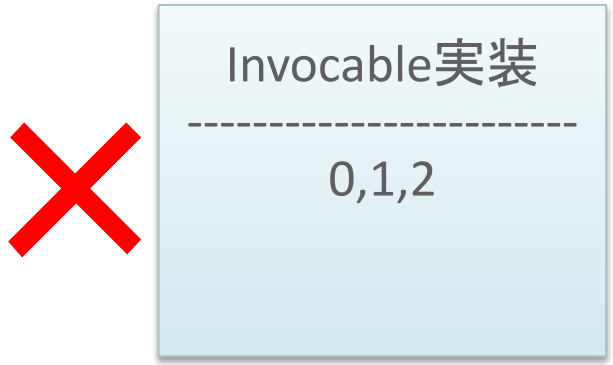
2.任意のロジックをCacheServer側で実行する方法

Invocable実装の注意点

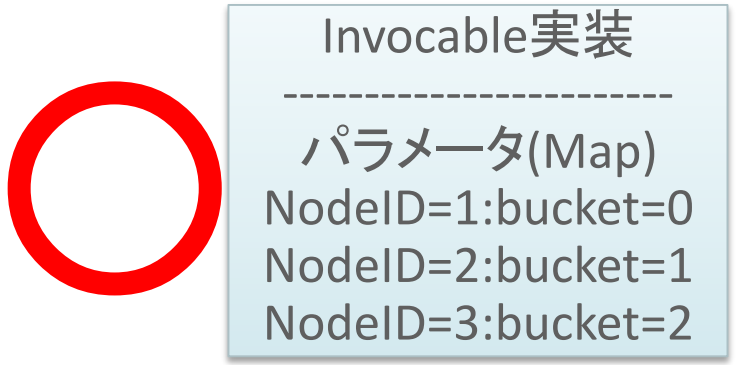
InvocationServiceは1つのInvocableを一度に全サーバにリクエストするため、1つのInvocableに全サーバ分のパラメータを詰め込む必要がある

- InvocableのパラメータはMapで与えるのが常套

  keyにCohereceのプロセス毎に異なる識別番号（通称NodeID）、valueに各Coherenceプロセスが処理する対象のバケット番号



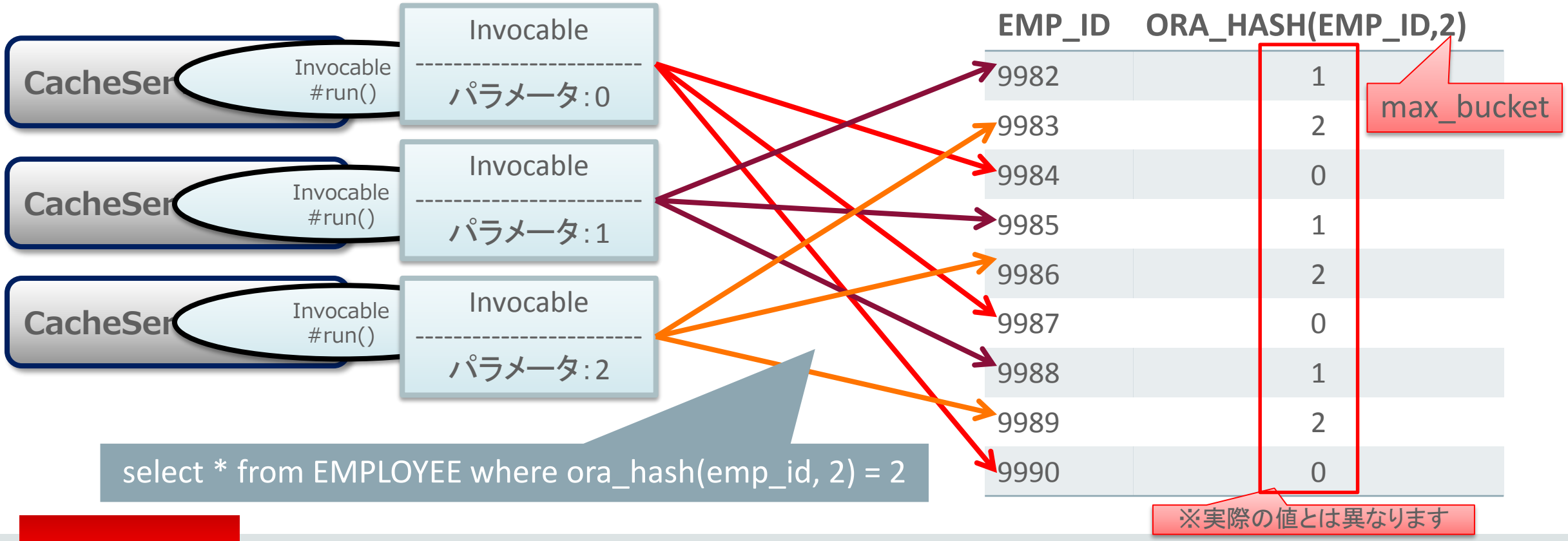
自身のみが処理すべき
値の区別がつかない



対象NodeIDと値が明確

3. DBからのロードするデータをCacheServer毎に指示する方法

- データベースへのデータ指定には、ハッシュ関数を使用する方法がある
- 下図はORA_HASH関数(OracleDatabaseの場合)を使用した例



便利なAPI #1/2

- クラスタ内の各Coherenceプロセスの情報を保持するインスタンスはMember
- 自身のMemberインスタンスを取得するAPI

```
Member localMember = CacheFactory.getCluster().getLocalMember();
```

- MemberからNodeIDを取得するAPI

```
Int nodeID = member.getId();
```

- クラスタ内のCoherenceプロセスをMember集合として取得するAPI

```
Set<Member> memberSet = CacheFactory.getCluster().getMemberSet();
```

- サービス名指定でクラスタ内のCoherenceプロセスのMemberを取得するAPI

```
Set<Member> memberSet= CacheFactory.getService(serviceName).getInfo().getServiceMembers();
```

便利なAPI #2/2

- 分散サービス名指定でクラスタ内のCoherenceプロセスのMemberを取得するAPI

```
PartitionedService service = (PartitionedService) CacheFactory.getService(serviceName);
Set<Member> memberSet= service.getOwnershipEnabledMembers();
```

- サービス名指定でInvocationServiceを取得する

```
InvocationService invocationService = (InvocationService) CacheFactory.getService(invocationServiceName);
```

- 例）localstorage=trueな分散サービス”DistributedCache”が動作しているCacheServerにてInvocableを実行する

```
PartitionedService service = (PartitionedService) CacheFactory.getService("DistributedCache");
Set<Member> memberSet= service.getOwnershipEnabledMembers();
InvocationService invocationService = (InvocationService) CacheFactory.getService("InvocationService");
invocationService.execute(invocable, memberSet, observer);
```

注意点

- InvocationServiceの実行は execute()を使用する
  query()メソッドはエラーを検知できないため使用すべきではない
- データベースに予めインデックスを作成しておく
  インデックスを作成しないと、速くならない

  例） create index EMP_ID_HASH_INDEX on EMPLOYEE (ORA_HASH(emp_id, 2))

- Coherenceへのデータ登録は、NamedCache.putAll(Map)で構わない
  ロードを実行するCoherenceにのみ登録されるように実装すると複雑化
  putAllを使用したことで問題が発生したことは現時点ではなし

まとめ

- いかに通信回数を少なくするか？
- いかに並列に処理させるか？
- いかにトラフィックを軽減させるか？

- チューニングのポイントは、上記を『いかに“設計時点で織り込むか？”』
- 他の案件でも、本日お話しした内容と同じアプローチで臨んでいます
- ご相談はオラクルコンサルティングサービスまで

ORACLE

Hardware and Software
Engineered to Work Together

VISION 2020

#1 CLOUD

ORACLE JAPAN


ORACLE®