
ORACLE®

#odddtky

Oracle DBA & Developer Days 2014

Oracle Database In-Memoryの導入／開発における考慮点

DBIM適用パターンに関する考察

佐藤裕之

Cloud & Big Data推進部長
データベース事業統括
製品戦略統括本部
日本オラクル株式会社

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

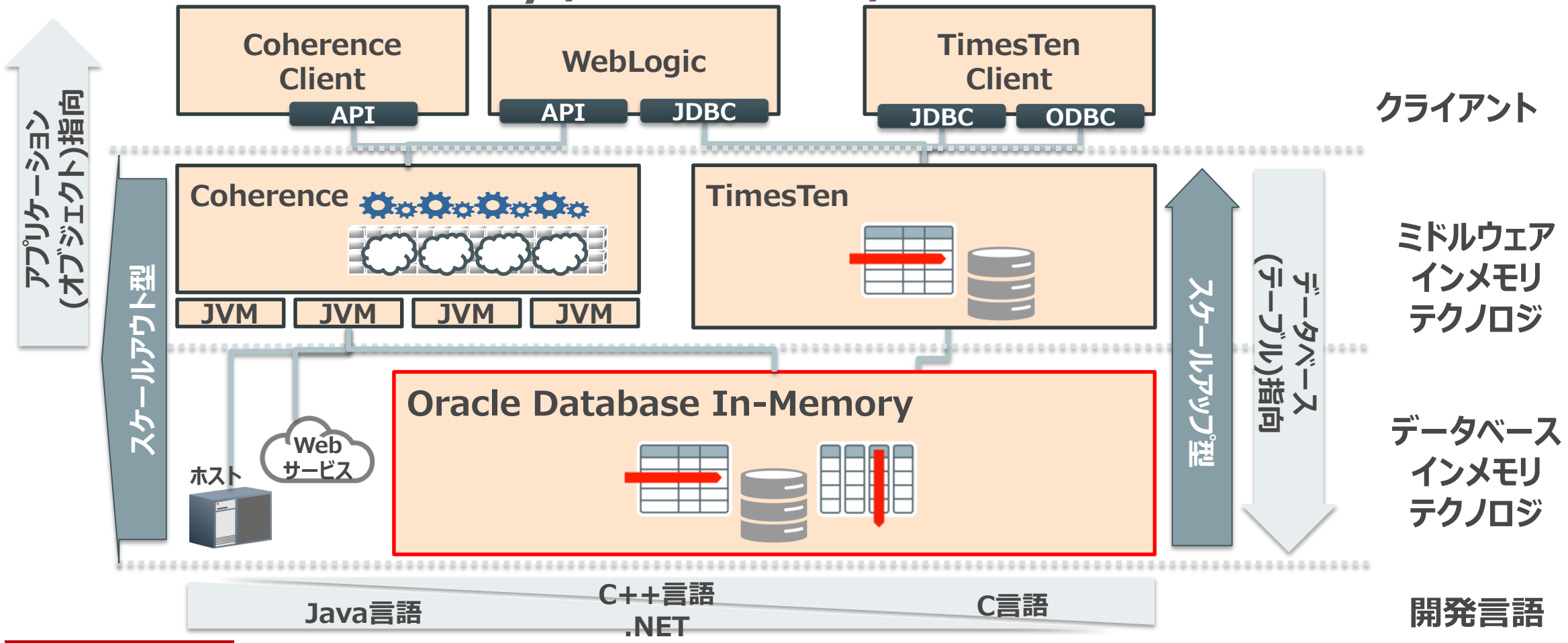
- 以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

アジェンダ

1. DBIMを総復習
2. DBIM適用パターンに関する考察

オラクルが提供するIn-Memory関連製品

Database In-Memory / TimesTen / Coherence



分析処理を高速化する新たな技術革新

DBにおける主要な2種類のフォーマット　– ロー型 vs カラム型

ロー (行)型

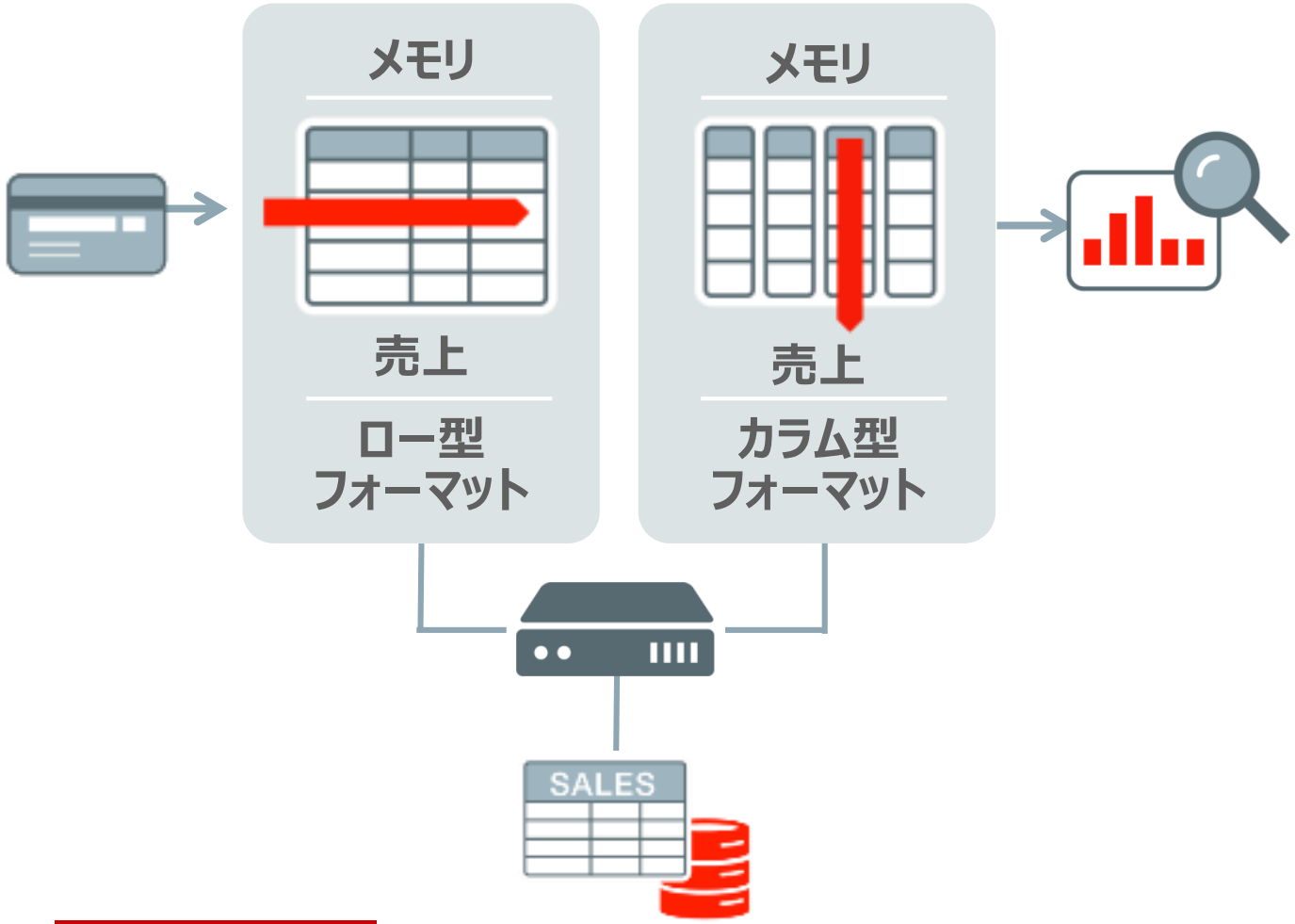
- OLTP処理を得意とするロー型
  - 例：注文データの挿入と検索
  - 少数の行(ロー)と多数の列(カラム)を高速処理

カラム(列)型

- 集計、分析処理を高速化するカラム型
  - 例：都道府県毎の売上合計のレポート
  - 少数の列(カラム)と多数の行(ロー)を高速処理

Oracle Database In-Memory テクノロジーは各特性を持つ、
２つのフォーマットを“両方同時に”メモリ上にロードし利用可能

分析処理を高速化する新たな技術革新
インメモリ・デュアル・フォーマット

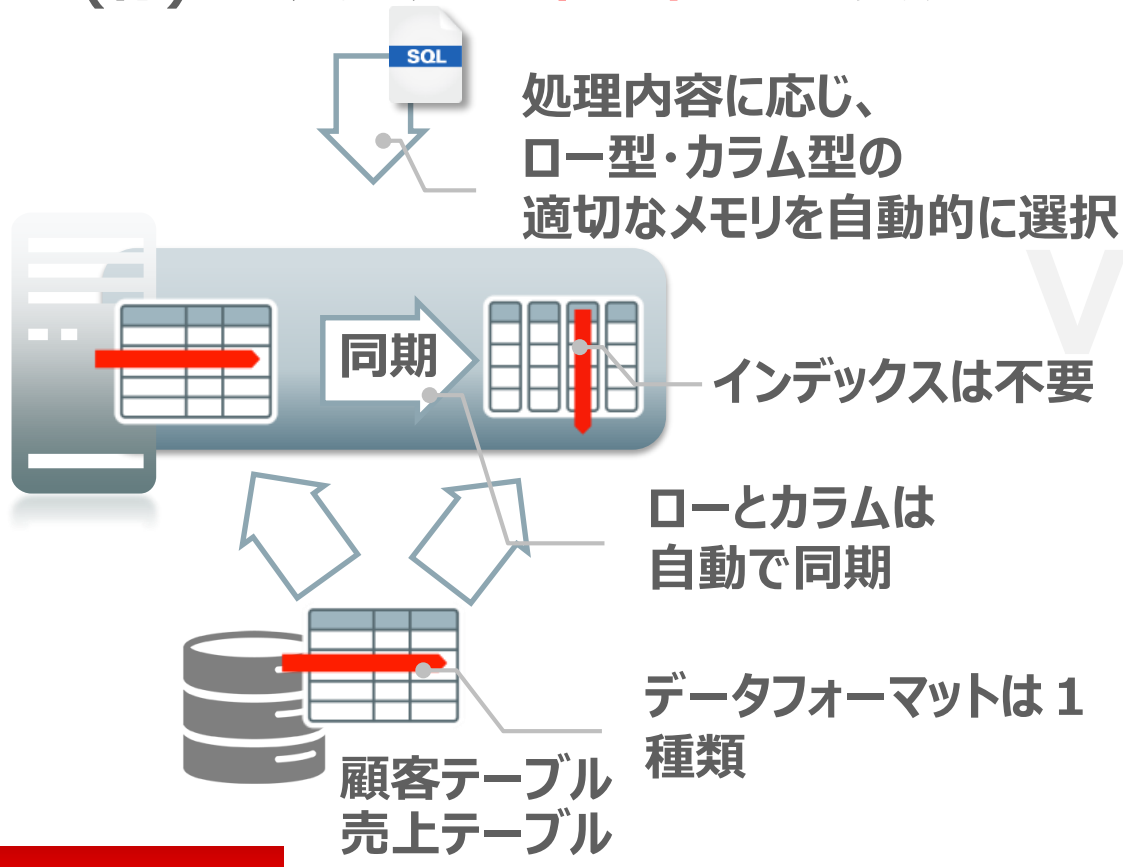


- 同一のデータをロー型、カラム型両方のフォーマットでメモリ上に保持
- 両方のフォーマットを同時に利用可能、トランザクションの一貫性も担保
- 2種類のフォーマットをオプティマイザが自動判別
  - 集計、レポート処理はカラム型フォーマットに対して実行
  - OLTP処理はロー型フォーマットに対して実行

OLTPと分析処理をリアルタイムに融合可能な“唯一の”仕組み
インメモリ・デュアル・フォーマット

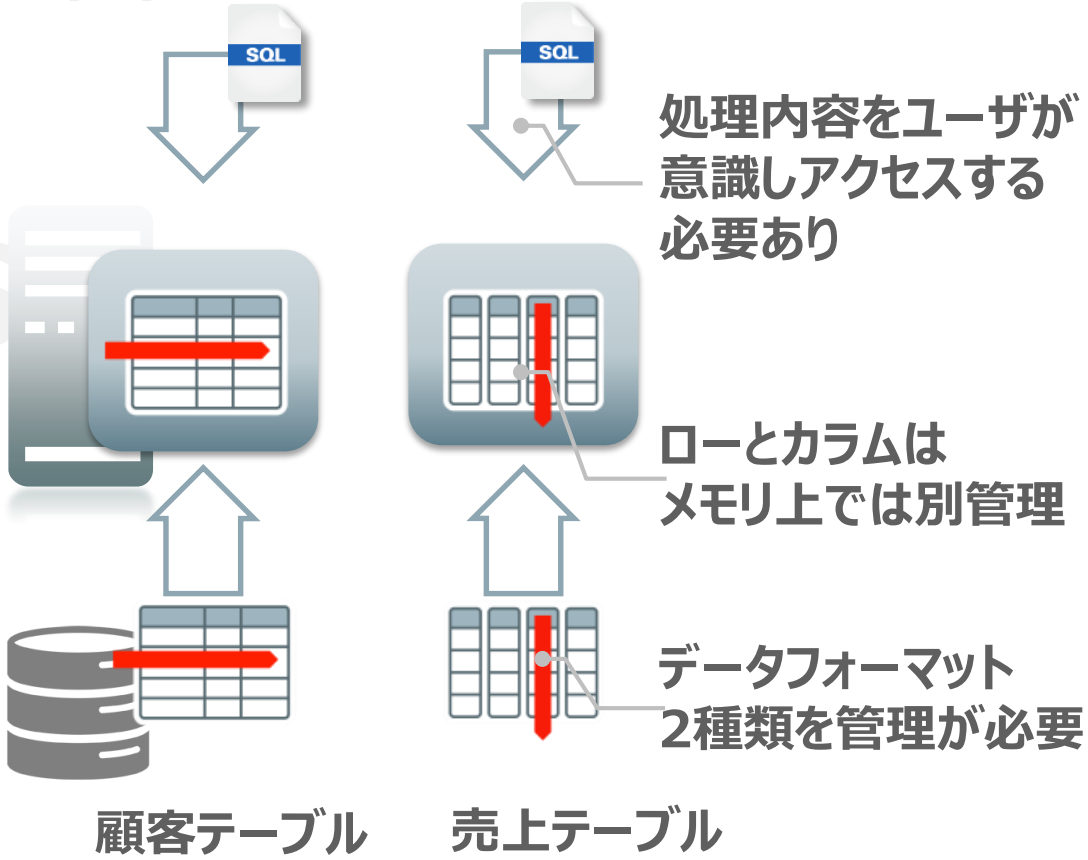
Oracle Database In-Memory

■ロー(行)型とカラム型を”両方同時に”実現



一般的なインメモリの仕組み

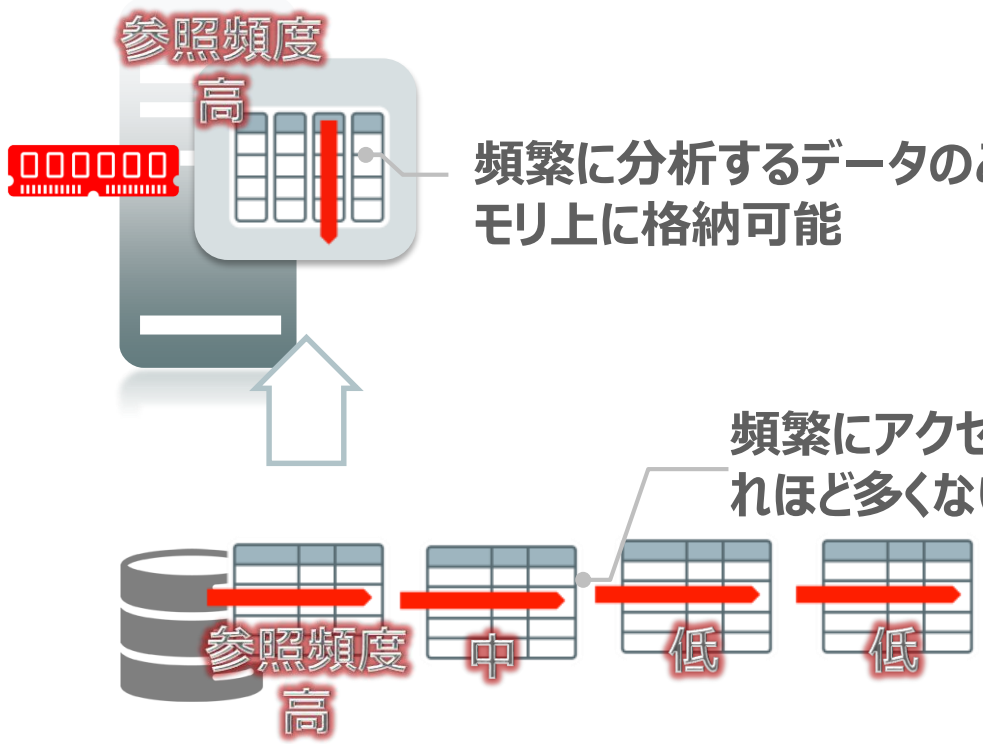
■ロー(行)型とカラム型の”どちらか一方を“選択



コスト効率性の高いメモリ上へのデータ配置

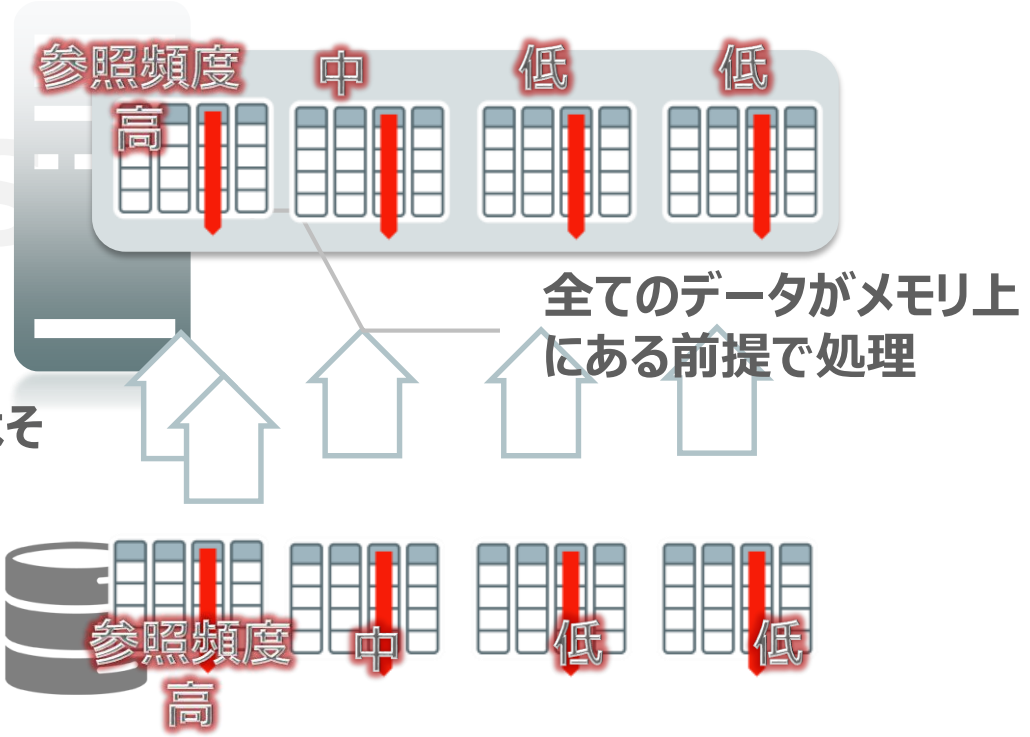
Oracle Database In-Memory

■高速化したいデータのみをメモリ上に展開



その他のインメモリ実装

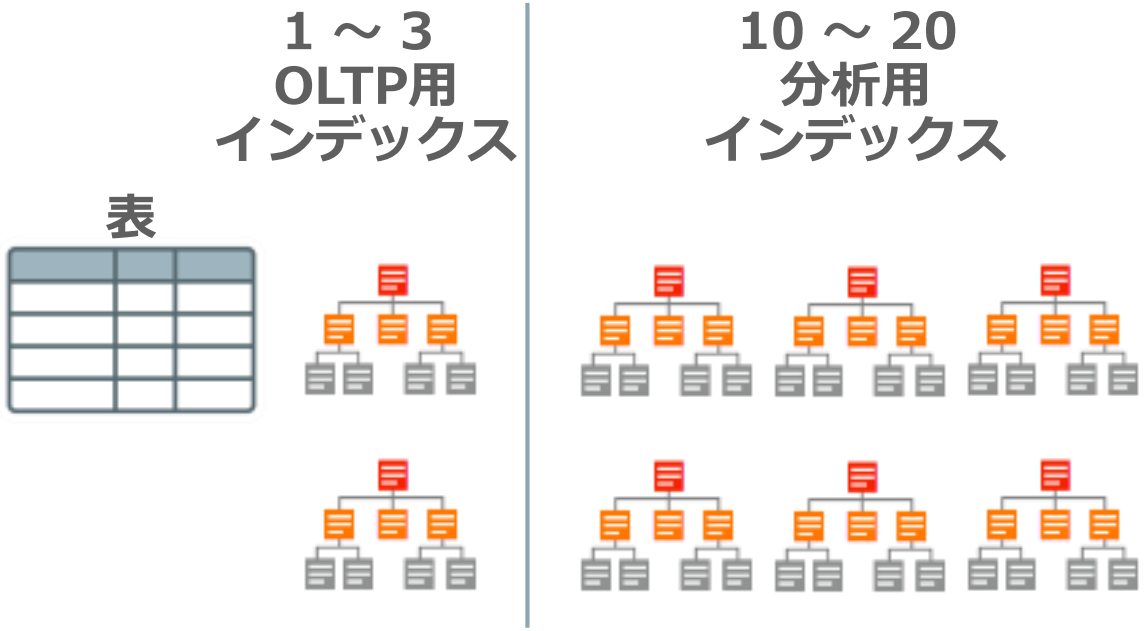
■ 基本的に全てのデータをメモリ上に展開



分析基盤におけるインデックスが不要

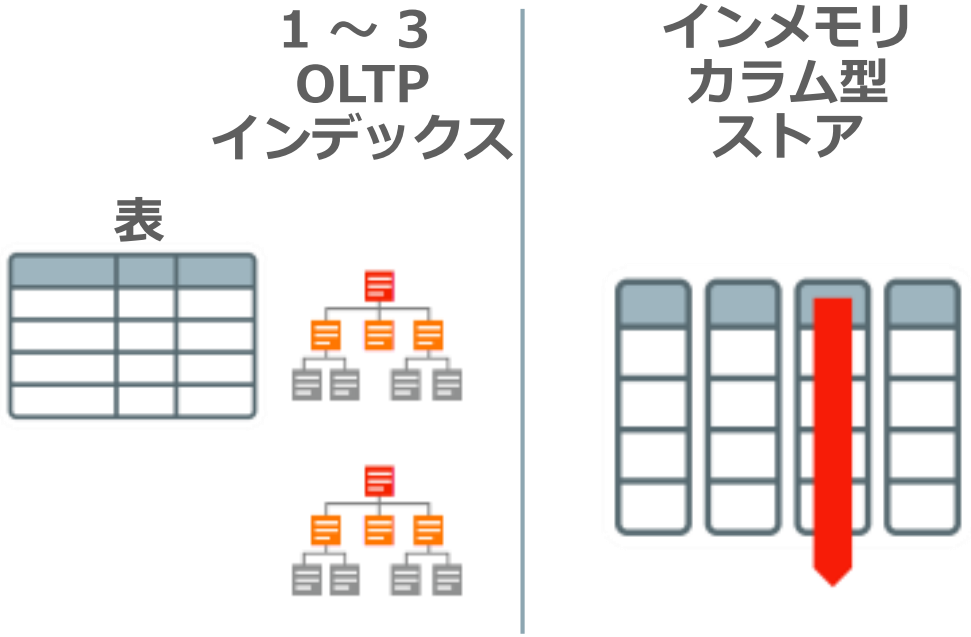
■ 既存Oracle Databaseによる分析基盤

- □ インデックスは事前予測可能なパターンのみ高速化
- □ 更新処理は10～20個のインデックスの更新が必要
  - ➢ パフォーマンス低下の原因



■ Oracle Database In-Memoryの適用

- □ インデックスなしですべてのカラム(列)の処理を高速化
- □ カラム型ストアはディスクI/Oなし

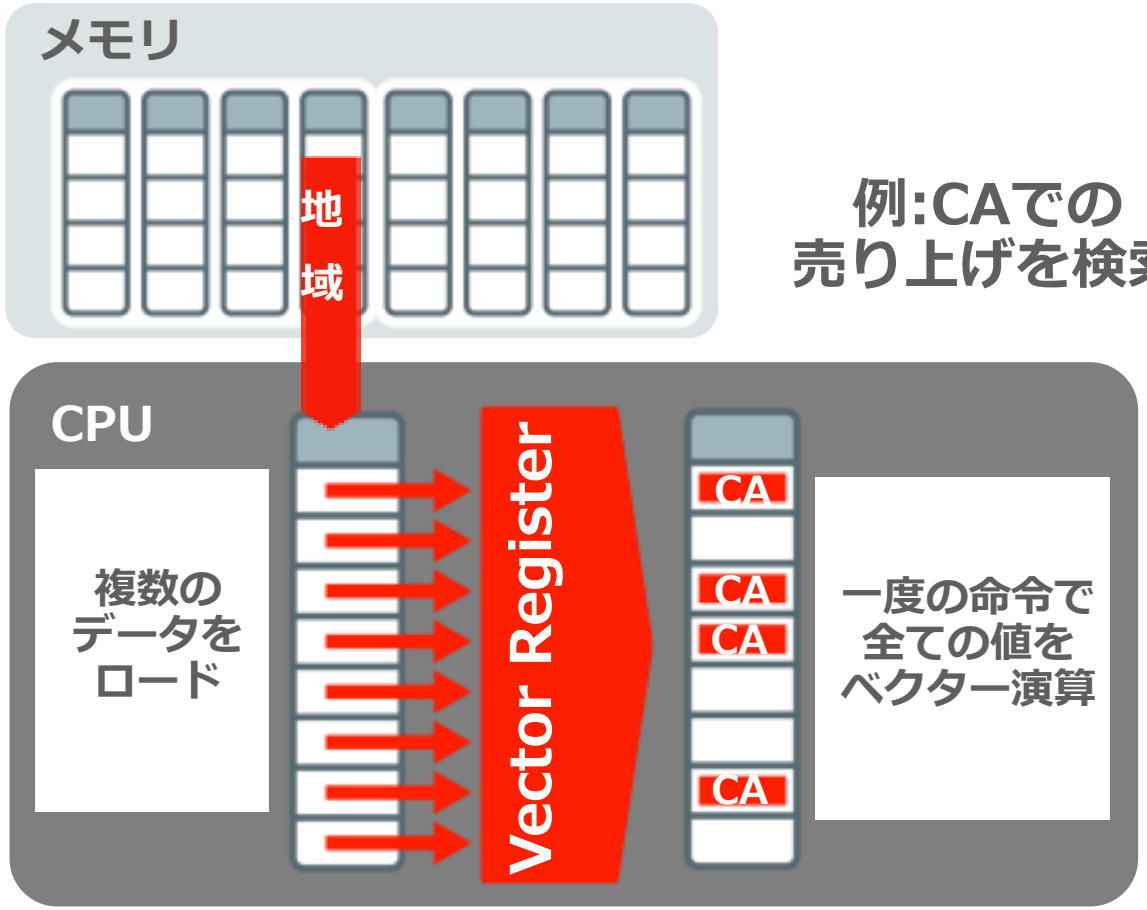


ユーザからは透過的

内部動作

CPUコア毎に数10億行/秒のスキャンを実施

メモリ内で超並列処理を支えるテクノロジ



例:CAでの売り上げを検索

- 各CPUコアが並列にデータを処理
- SIMDによる並列処理
- 各コアあたり数10億行/秒の処理能力を実現
  - ロー型フォーマットは数100万行/秒

＞ 100倍高速化

主要なDatabase In-Memoryの特徴

- 性能から見たDBIMの特徴
  - DWH(分析)処理の高速化
  - OLTP処理の一部高速化
- 運用管理からみたDBIMの特徴
  - アプリケーション変更必要なし
  - インデックスが基本的に不要
  - アドホックなクエリへの対応容易性
  - ストレージ容量削減
  - 既存の拡張性・可用性セキュリティの機能をそのまま利用可能

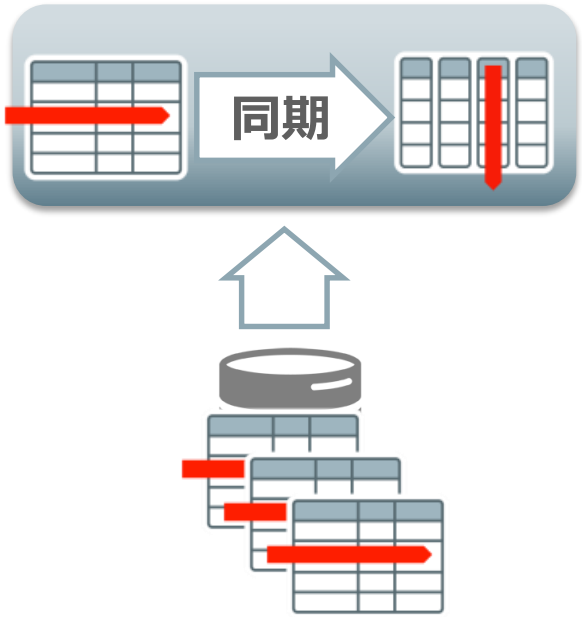
アジェンダ

1. DBIMを総復習
2. DBIM適用パターンに関する考察
    - １）基本構成からみたDBIM想定適用パターン
    - ２）全体アーキテクチャからみたDBIM想定適用パターン

基本構成からみたDBIM想定適用パターン①

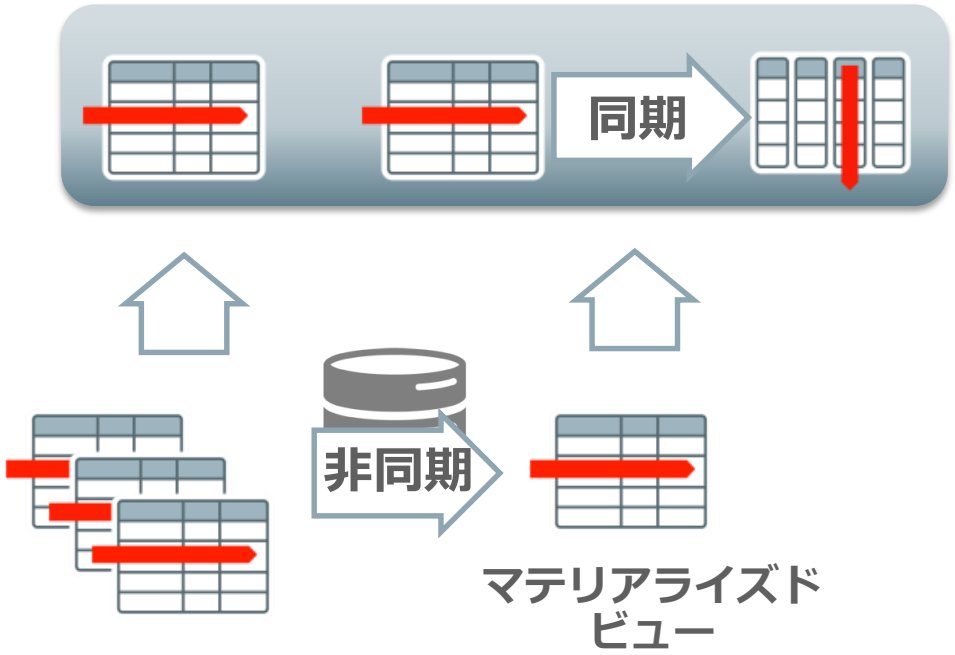
同一筐体内での構成例

1) 同一筐体内で同期



- アドホッククエリの簡素な高速化
- リソースを共有
- 更新をリアルタイムに同期
- 一部のデータを分析に活用可能

2) 同一筐体内で非同期

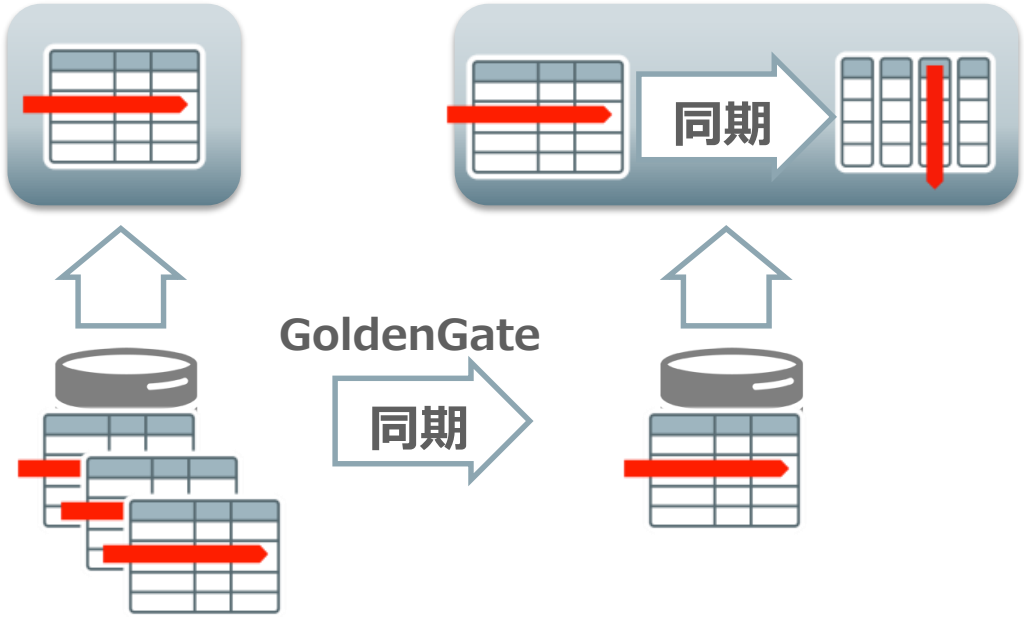


- アドホッククエリの簡素な高速化
- リソースを共有
- **更新の同期タイミングを制御可能**
- 一部のデータを分析に活用可能

基本構成からみたDBIM想定適用パターン②

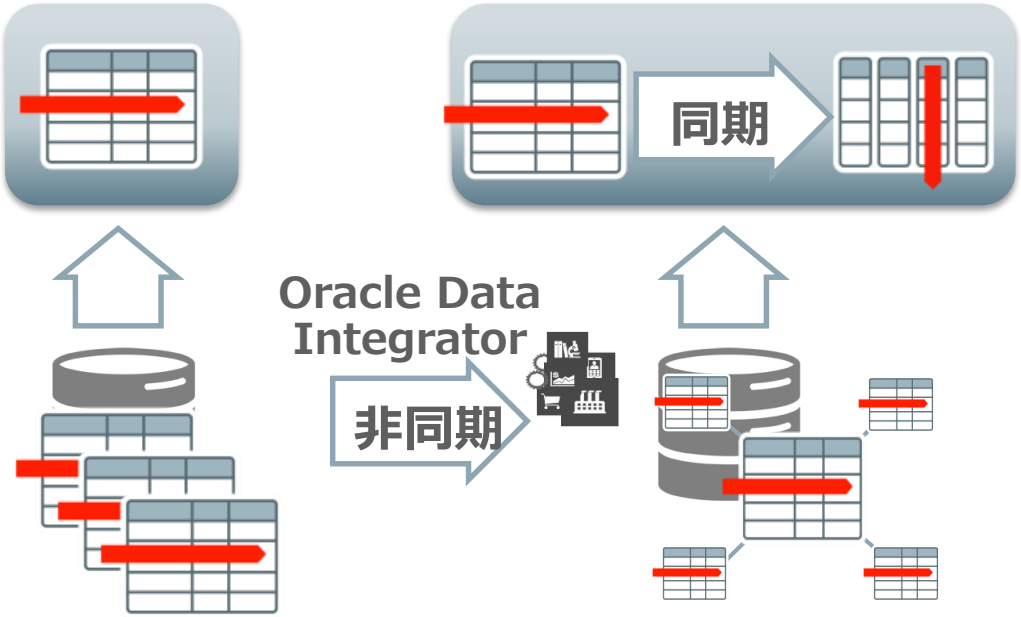
別筐体での構成例

3) 別筐体間で同期



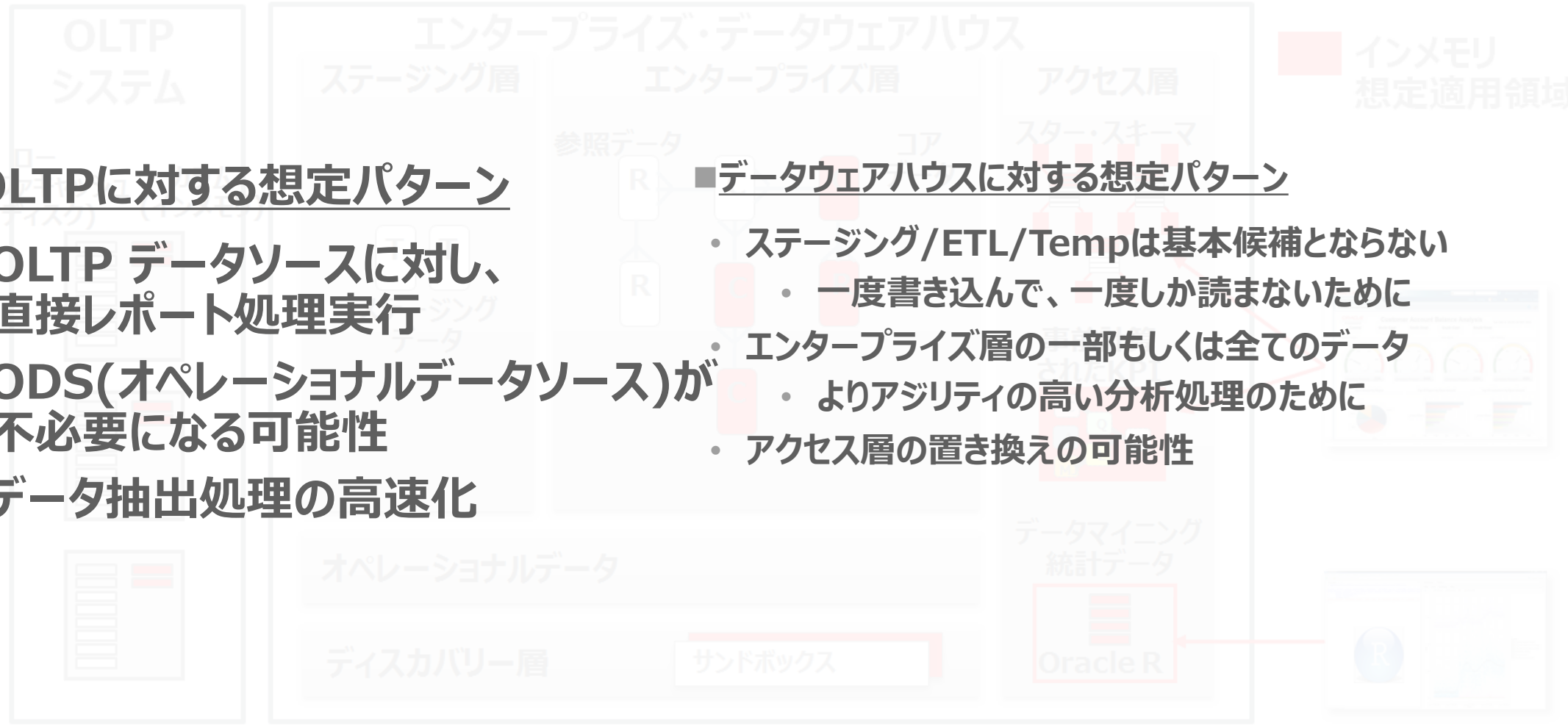
・アドホッククエリの簡素な高速化
・リソースを分離
・更新をリアルタイムに同期
・異なるバーションでも同期が可能
・一部のデータを分析に活用可能

4) 別筐体間で非同期



・アドホッククエリの簡素な高速化
・リソースを分離
・更新の同期タイミングを制御可能
・データの加工を柔軟に可能
（スタースキーマ・スノーフレーク等）

全体アーキテクチャからみたDBIM想定適用パターン①



■OLTPに対する想定パターン

- OLTP データソースに対し、直接レポート処理実行
- ODS(オペレーショナルデータソース)が不必要になる可能性
- データ抽出処理の高速化

■データウェアハウスに対する想定パターン

- ステージング/ETL/Tempは基本候補とならない
  - 一度書き込んで、一度しか読まないために
- エンタープライズ層の一部もしくは全てのデータ
  - よりアジリティの高い分析処理のために
- アクセス層の置き換えの可能性

全体アーキテクチャからみたDBIM想定適用パターン②

OLTP
システム

ロー
(バッファキャッシュ
＋ディスク)

カラム
(インメモリ)

エンタープライズ・データウェアハウス

ステージング層

T T

ステージング
データ

エンタープライズ層

参照データ

コア
データ

R C C R C R C

アクセス層

スター・スキーマ

事前計算
されたKPI

C Q A A L MJ

データマイニング
統計データ

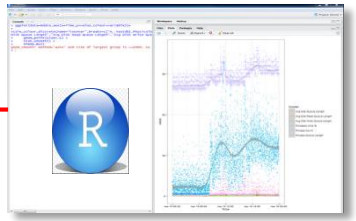
Oracle R

オペレーショナルデータ

ディスカバリー層

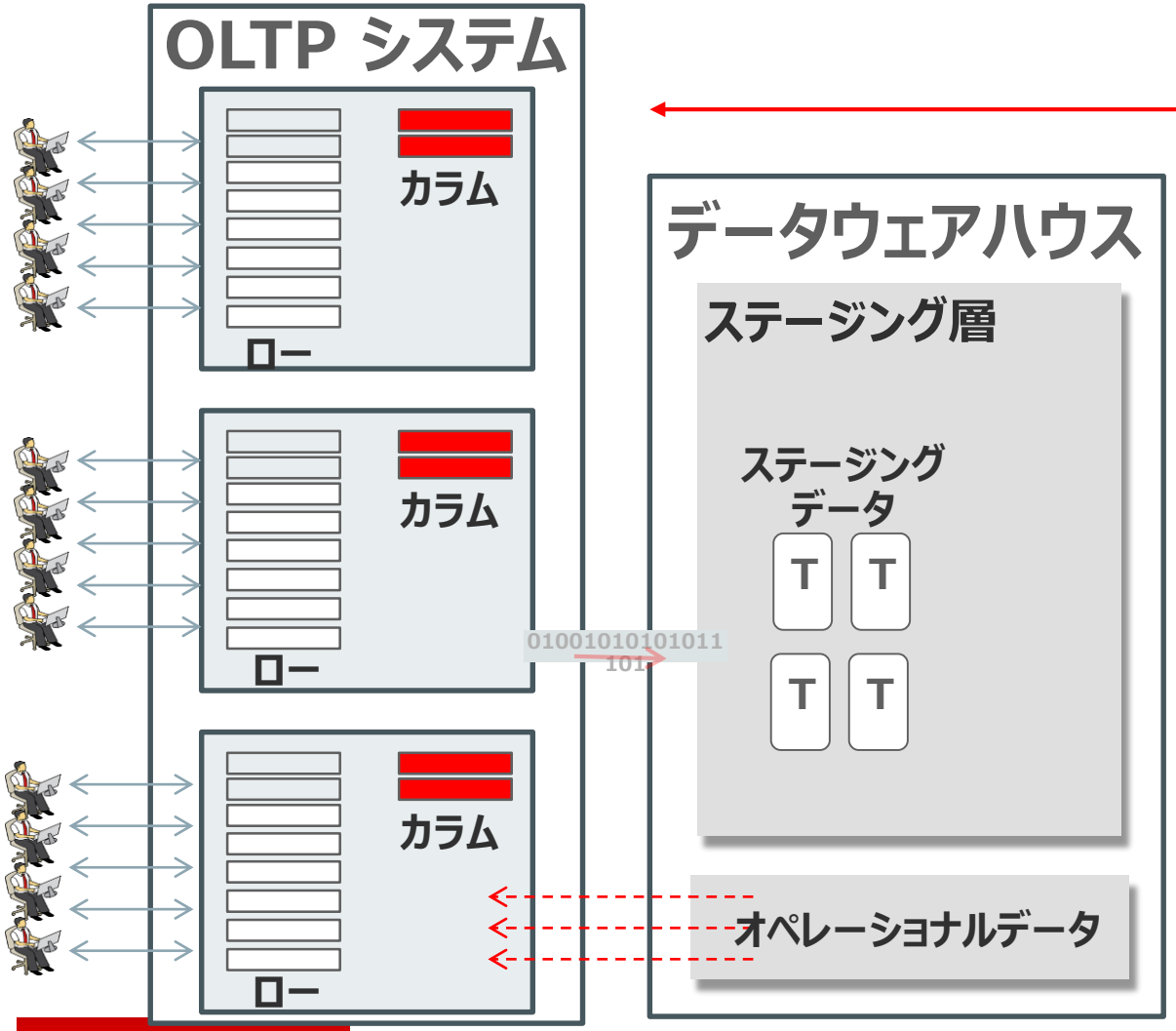
サンドボックス

インメモリ
想定適用領域



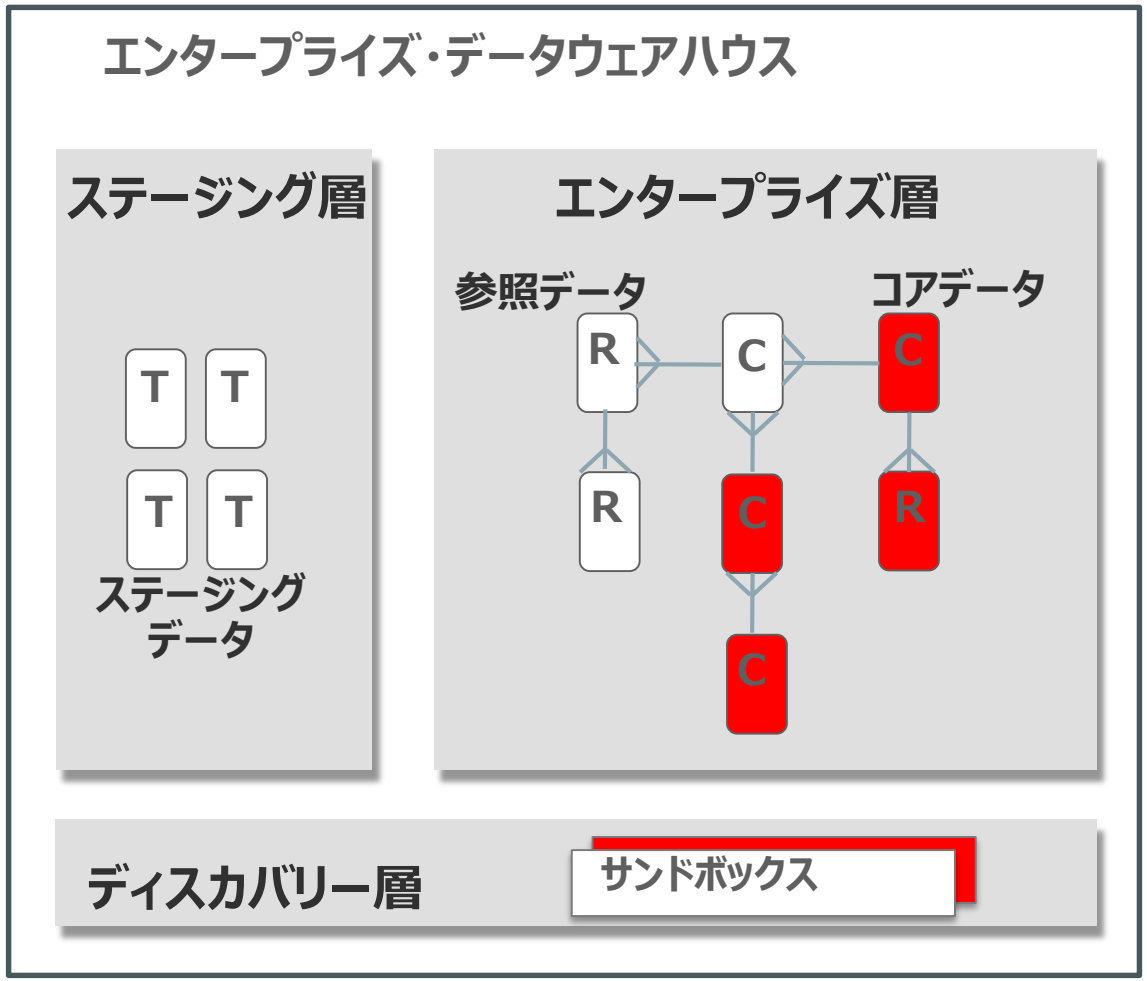
全体アーキテクチャからみたDBIM想定適用パターン③

OLTPシステムにおけるDBIMの想定適用例



- OLTPシステムに対し
  直接リアルタイム・レポート処理を実行
- バッファキャッシュ(ロー)を使いつつ、
  インメモリ化されたトランザクション
  データをETL処理の抽出処理に利用
- ODS(オペレーショナルデータストア)が
  不要な可能性あり

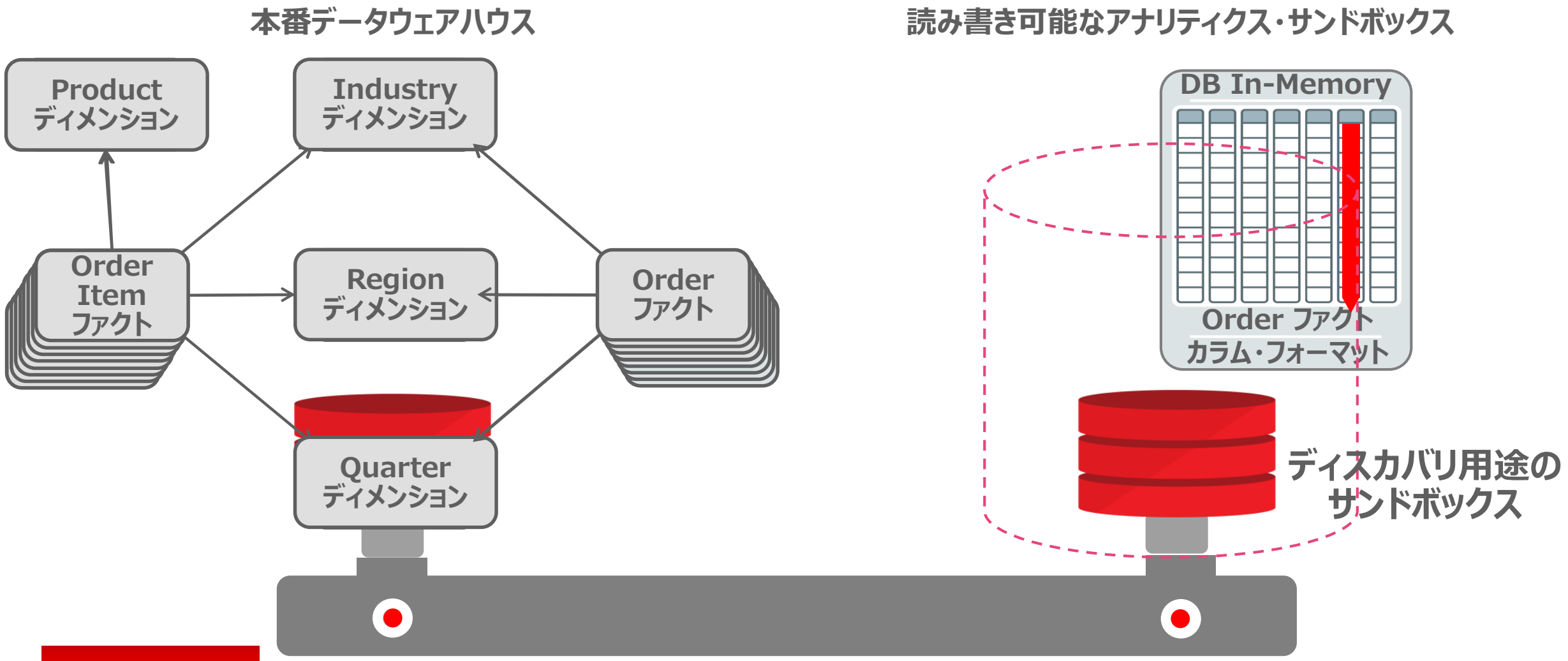
全体アーキテクチャからみたDBIM想定適用パターン④
第3正規形(3NF)データウェアハウスとDBIM想定適用例



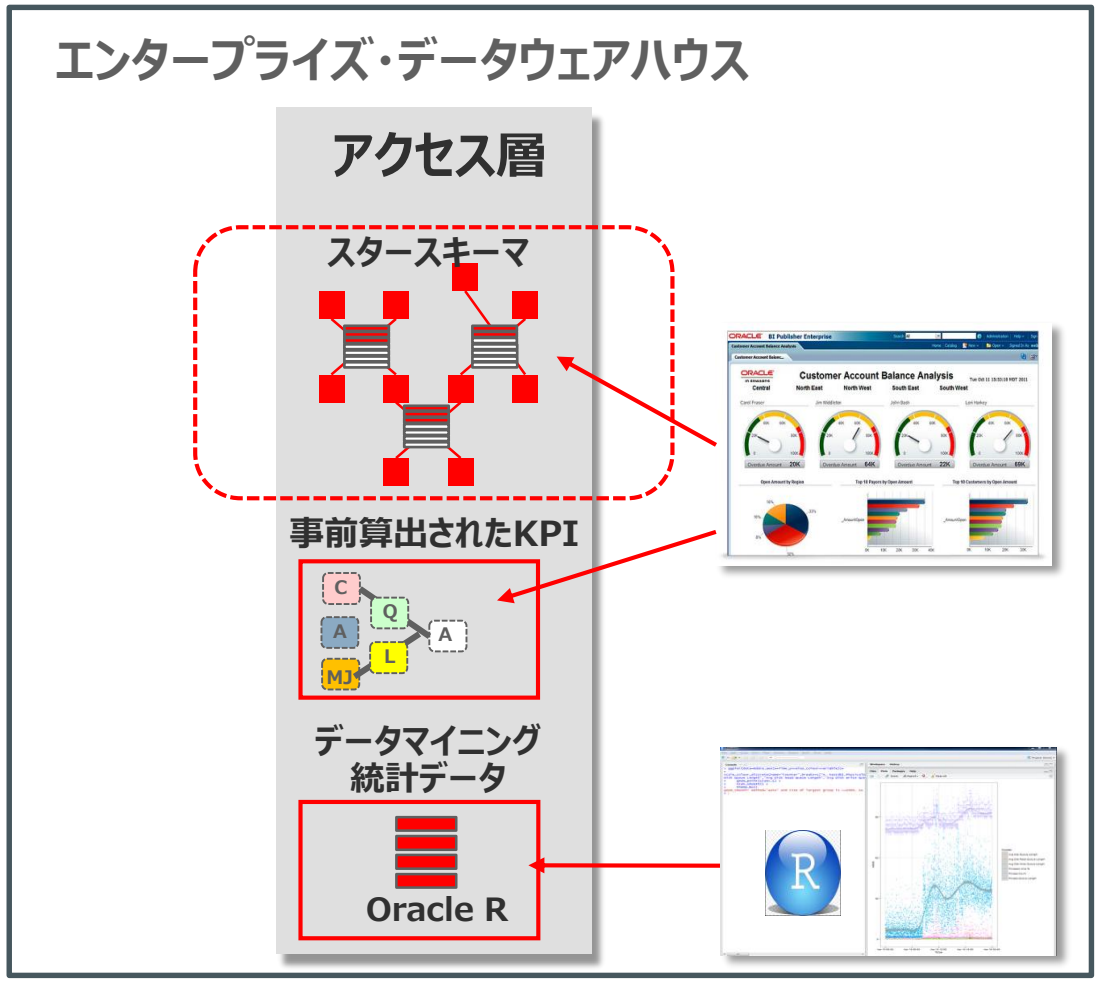
- エンタープライズ層は、トランザクションデータの完全なコピーを保持
- パフォーマンスが必要とされたデータのみをインメモリへポピュレート
- サンドボックス環境内で優先ユーザへDBIMを割り当て（アドホッククエリのパフォーマンス向上）

全体アーキテクチャからみたDBIM想定適用パターン⑤

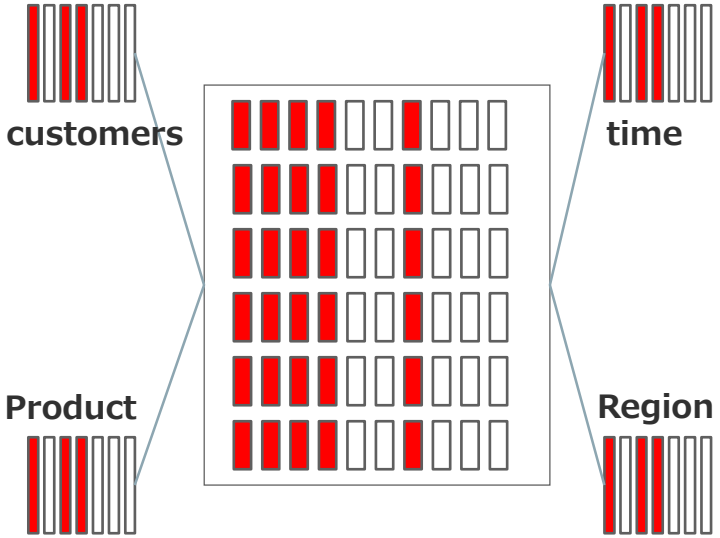
サンドボックスにおけるDBIMとマルチテナント想定適用例



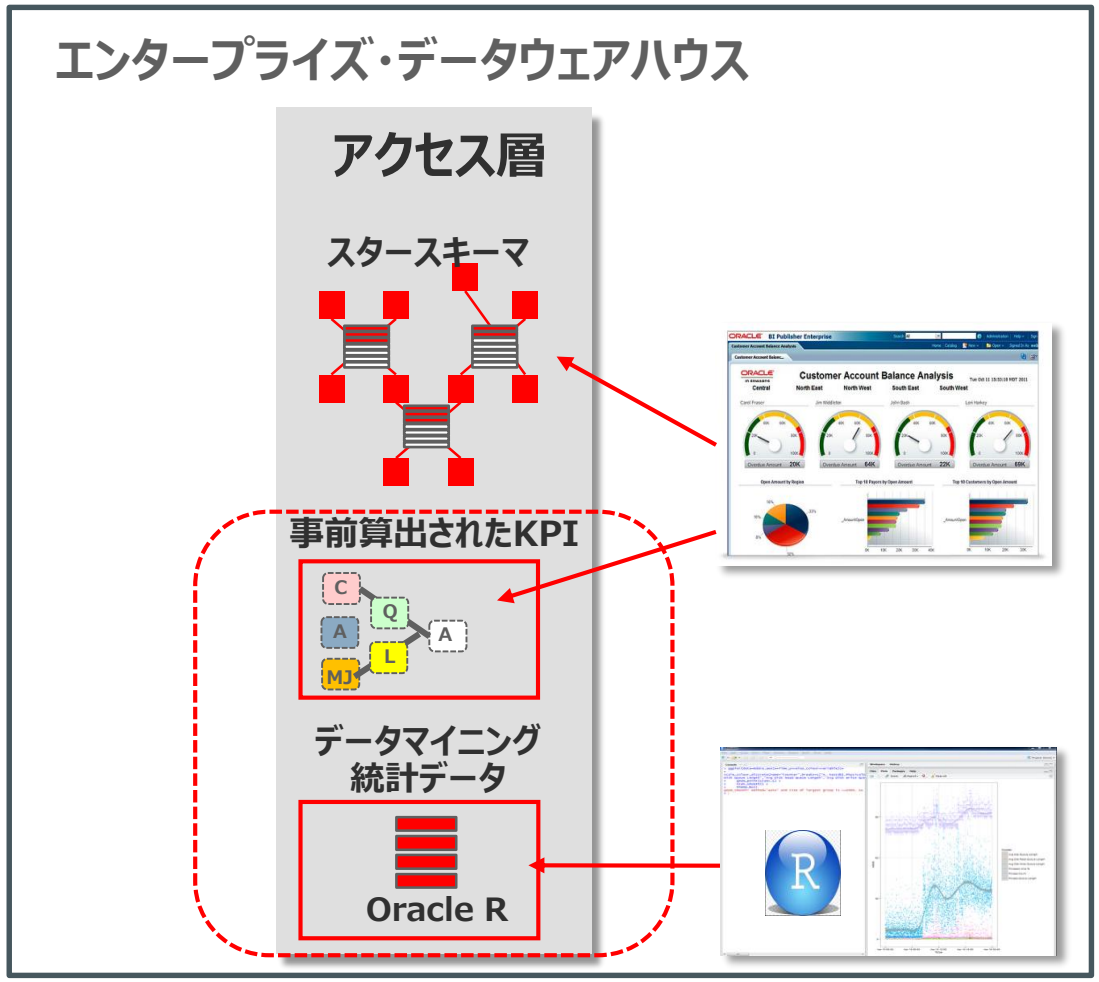
全体アーキテクチャからみたDBIM想定適用パターン⑥
アクセス層：スタースキーマにおけるDBIM適用例



- ユーザにとり扱いやすい既存のスタースキーマをそのまま利用
- 全体もしくは部分的にインメモリ化し高速化可能

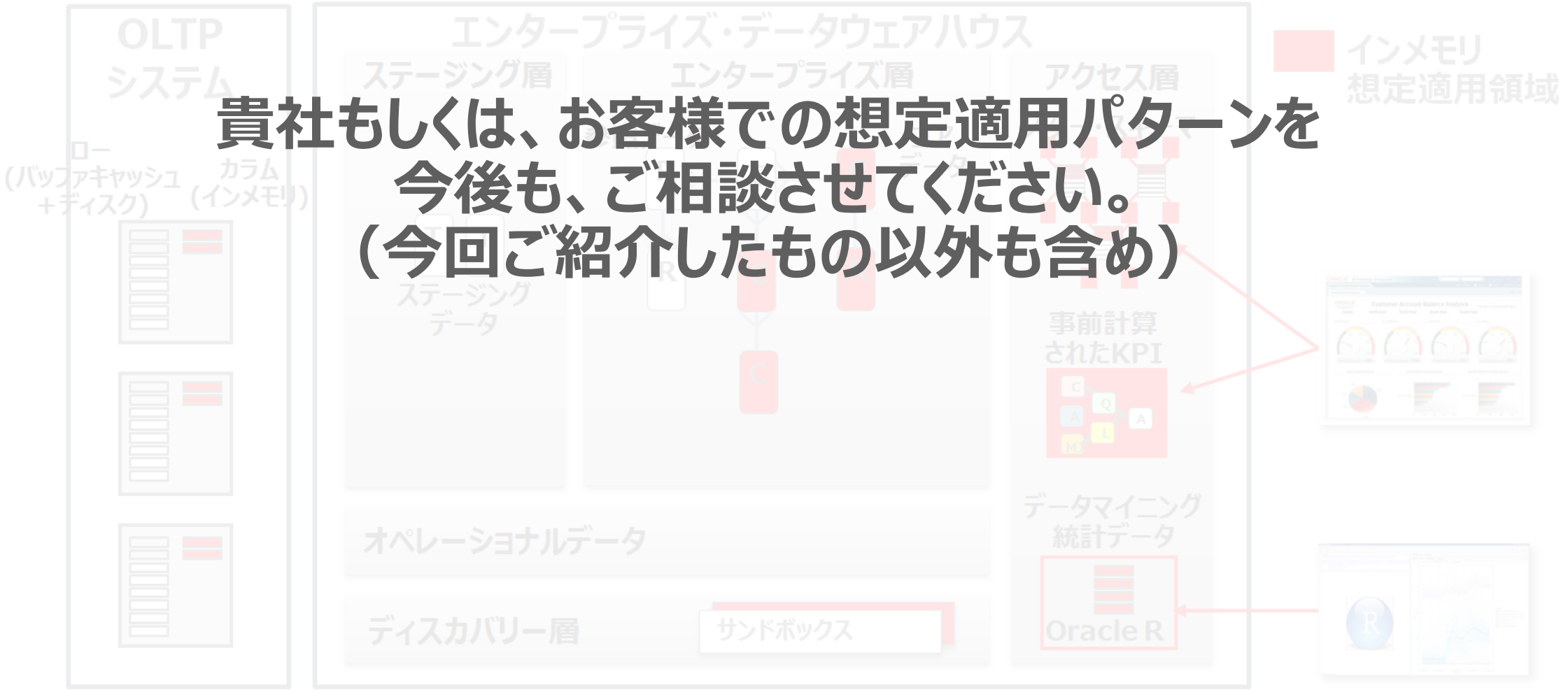


全体アーキテクチャからみたDBIM想定適用パターン⑦
アクセス層：事前に算出したデータへのDBIM適用例



- BIアナリティクスの多くは、予測可能で事前に算出が可能
- 事前に算出されたKPIを例えばマテリアライズド・ビュー経由で参照
- インメモリへポピュレートしたマテリアライズド・ビューによりエンドユーザダッシュボードの高速化
- R言語による統計解析データ取得を高速化するためのインメモリ化

まとめ：DBIM適用パターン



貴社もしくは、お客様での想定適用パターンを
今後も、ご相談させてください。
（今回ご紹介したもの以外も含め）

まとめ

1. DBIMを総復習
2. DBIM適用パターンに関する考察

#odddtky

Oracle DBA & Developer Days 2014

Oracle Database In-Memoryの導入／開発における考慮点

アーキテクチャとパフォーマンス分析実践

丹羽勝久
担当ディレクター

日本オラクル株式会社
データベースエンジニアリング本部

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

Oracle Database 12c Release 1 (12.1.0.2) コアテクセミナー（OTNセミナー・オンデマンド）

http://www.oracle.com/technetwork/jp/ondemand/od12c-coretech-aug2014-2283256-ja.html

YouTubeを使った動画セミナーにより
Oracle Database In-Memoryの製品
機能を解説しています

1. 概要
2. マルチテナント・アーキテクチャ
3. Upgrade / 12.1.0.2における変更点
4. Rapid Home Provisioning
5. Grid Infrastructure
6. JSON対応
7. Oracle Database In-Memory概要
8. Oracle Database In-Memory (2)
9. Oracle Database In-Memory (3)
10. Oracle Database In-Memory (4)
11. Oracle Database In-Memory (5)
12. Systems
13. Data Warehouse (DWH) 関連機能
14. Enterprise Manager

アジェンダ

1. Oracle Database In-Memory（DBIM）概要
2. インメモリ・カラム・ストアの詳細
3. DBIMのインメモリ・スキャン
4. DBIMの導入効果が高い処理／高くない処理
5. DBIMのパフォーマンス統計情報

高速な分析をリアルタイム化する新たな技術革新

DBにおける主要な2種類のフォーマット　– ロー型 vs カラム型

ロー(行)型

- OLTP処理を得意とするロー型
  - 例：注文データの挿入と検索
  - 少数の行(ロー)と多数の列(カラム)を高速処理

カラム(列)型

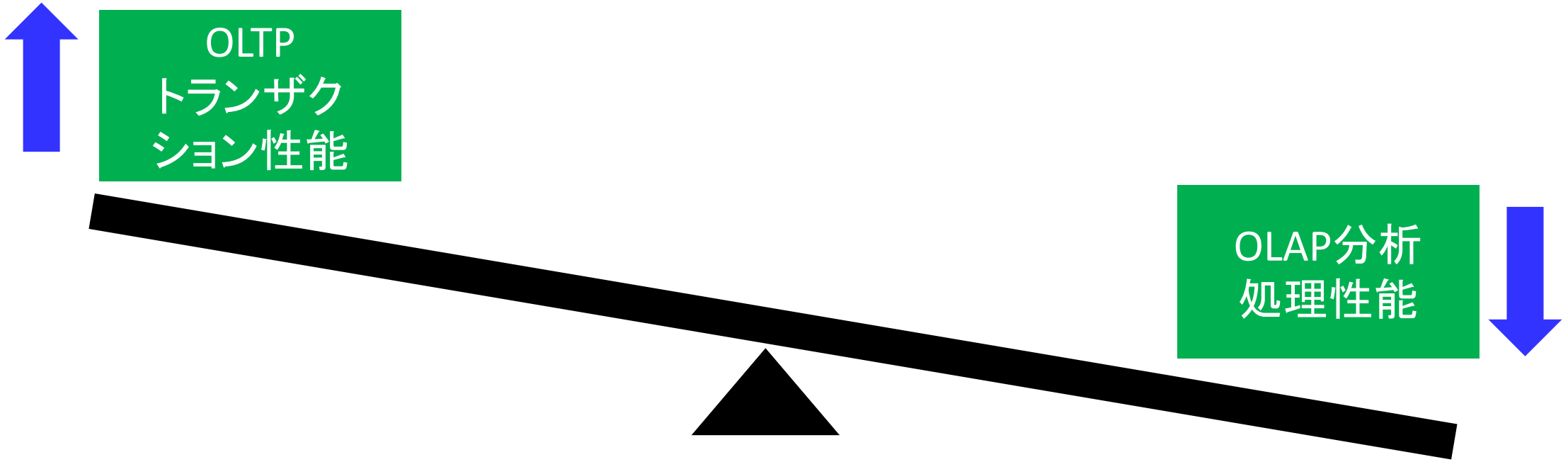
- 集計、分析処理を高速化するカラム型
  - 例：都道府県毎の売上合計のレポート
  - 少数の列(カラム)と多数の行(ロー)を高速処理

Oracle Database In-Memory テクノロジーは、各特性を持つ
2つのフォーマットを“両方同時に”メモリー上にロードし利用可能

OLTPとOLAPの性能向上はトレードオフ

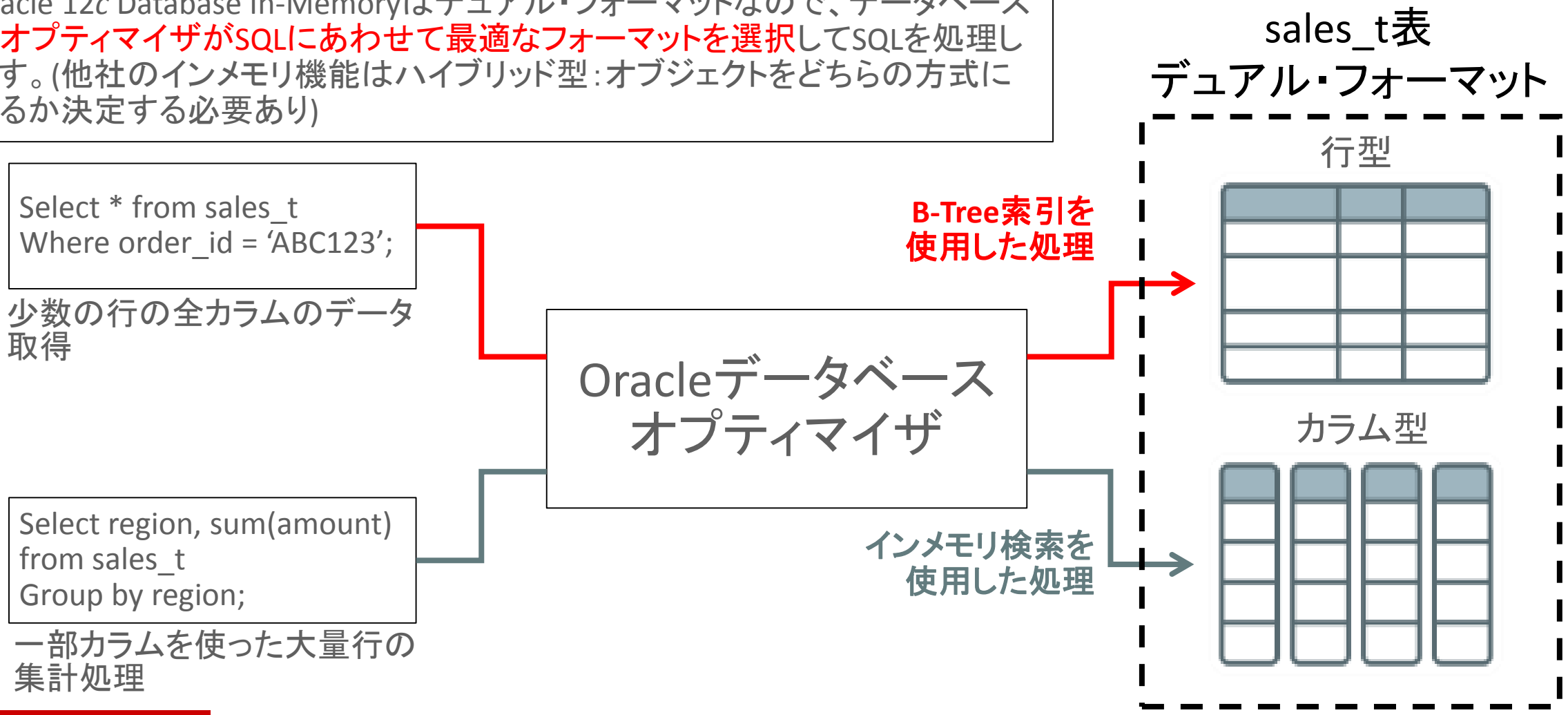
どちらかを性能向上するとどちらかにオーバーヘッドが発生

OLTPとOLAPを1つのデータベースで
共存することは難しい



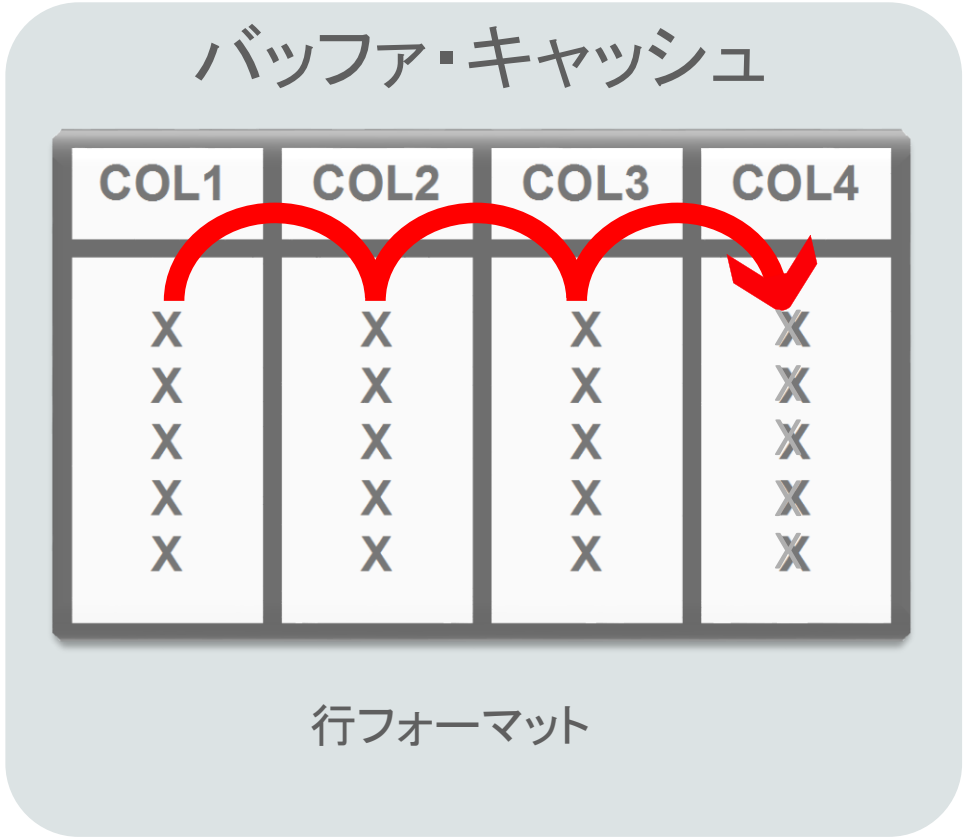
Oracle 12c Database In-Memory：デュアル・フォーマット

Oracle 12c Database In-Memoryはデュアル・フォーマットなので、データベースのオプティマイザがSQLにあわせて最適なフォーマットを選択してSQLを処理します。(他社のインメモリ機能はハイブリッド型：オブジェクトをどちらの方式にするか決定する必要あり)



カラム型表は何故分析用クエリーが高速か? (1)

必要なカラムのみアクセス

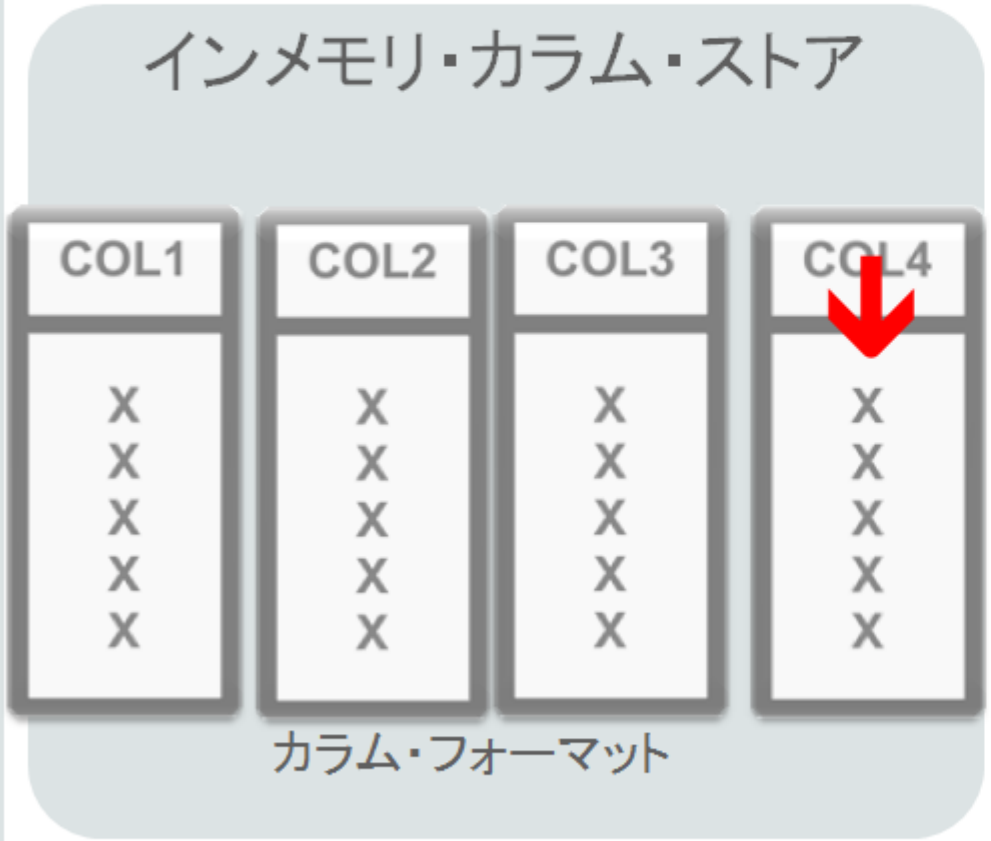



```
SELECT COL4 FROM MYTABLE;
```

X X X X X

カラム型表は何故分析用クエリーが高速か? (1)

必要なカラムのみアクセス




```
SELECT COL4 FROM MYTABLE;
```

結果

X X X X X

必要なカラムのみアクセス

データの読込量少ない

カラム型表は何故分析用クエリーが高速か? (2)

ディクショナリ圧縮

非圧縮

EMP表のJOB列

CLERK
SALESMAN
SALESMAN
MANAGER
SALESMAN
MANAGER
MANAGER
ANALYST
PRESIDENT
SALESMAN
CLERK
CLERK
ANALYST
CLERK

97 bytes

ディクショナリ圧縮

ディクショナリ(distinctされた値)

ソートされた値

カラム値	ディクショナリ値	ビット表現
ANALYST	0	000
CLERK	1	001
MANAGER	2	010
PRESIDENT	3	011
SALESMAN	4	100

カラム値サイズ合計＋ビット値合計
→ 36 bytes + 3bit * 5 = 38 bytes

エンコードされた各行の値 → 行

001	100	100	010	100	010	010	000	011	100	001	001	000	001

3bit * 14行 = 5.25bytes → 38 + 5.25 = **44 bytes (1/2.2 圧縮)**

ディクショナリ圧縮は圧縮した状態で検索が可能

Where job = 'MANAGER'

Where job = 010
に内部的に変換

※圧縮状態で検索可能

カラム型表は何故分析用クエリーが高速か? (3)

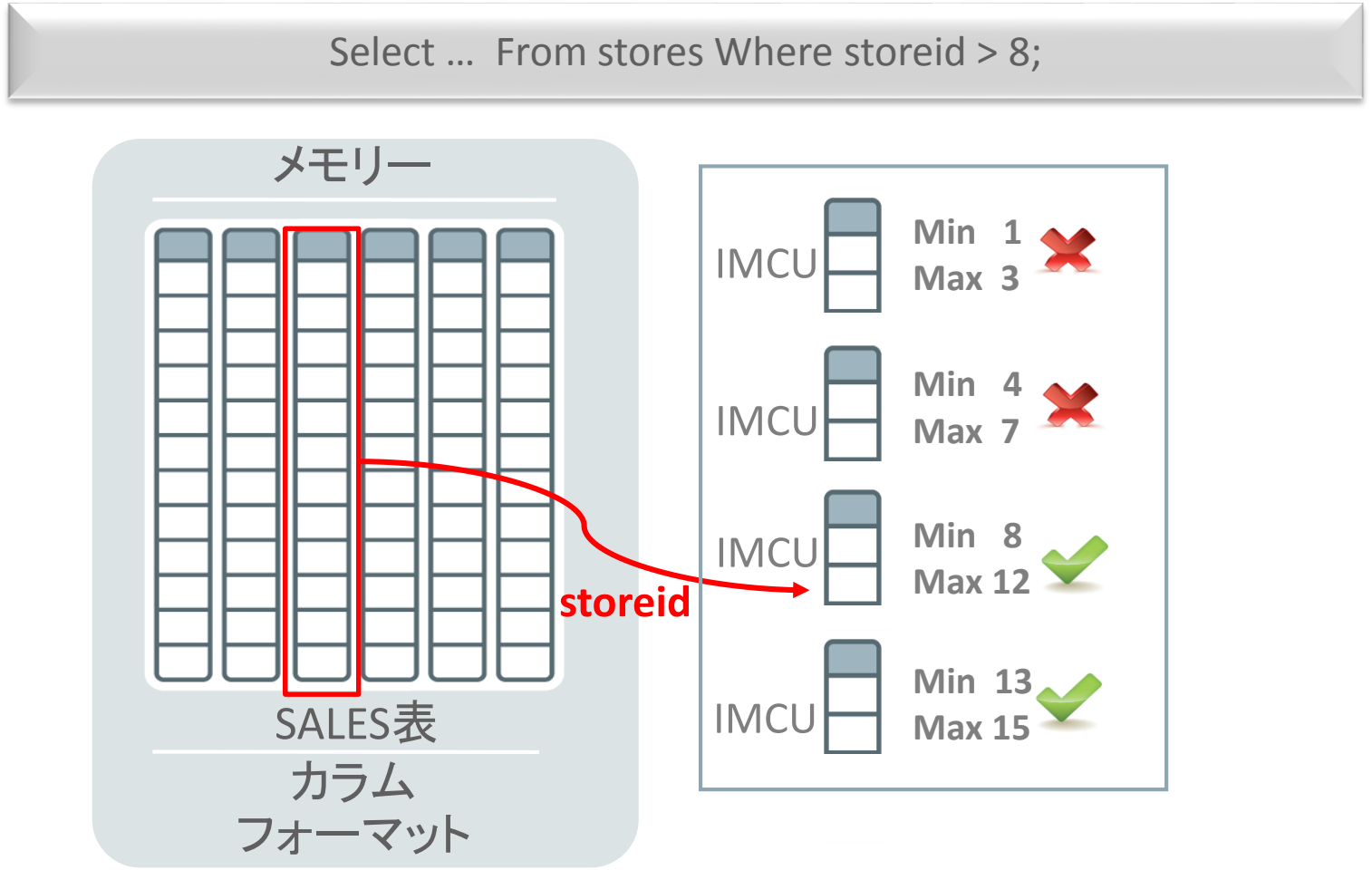
インメモリ・ストレージ索引 (※メモリー内に定義される)

各カラムは複数のカラム・ユニット(IMCU)で構成される

各IMCUで最小値/最大値を自動的に記録

WHERE句の条件に合致する領域だけを読み込み

すべての検索でパーティション・プルーニングと同様のパフォーマンスを提供



*1: IMCU - In-Memory Compression Unit

カラム型表は何故分析用クエリーが高速か? (4)

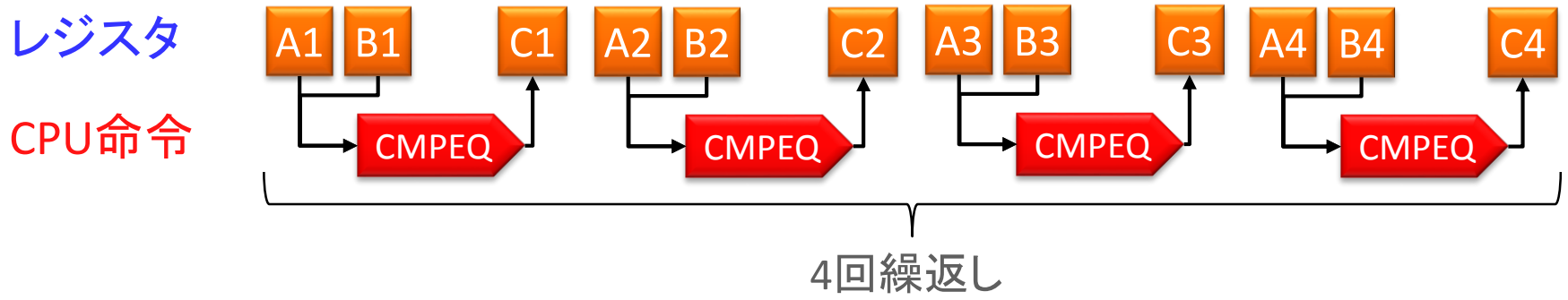
最新のプロセッサで搭載されているSIMD命令セットにより高速スキャン

SIMD : Single Instruction Multiple Data

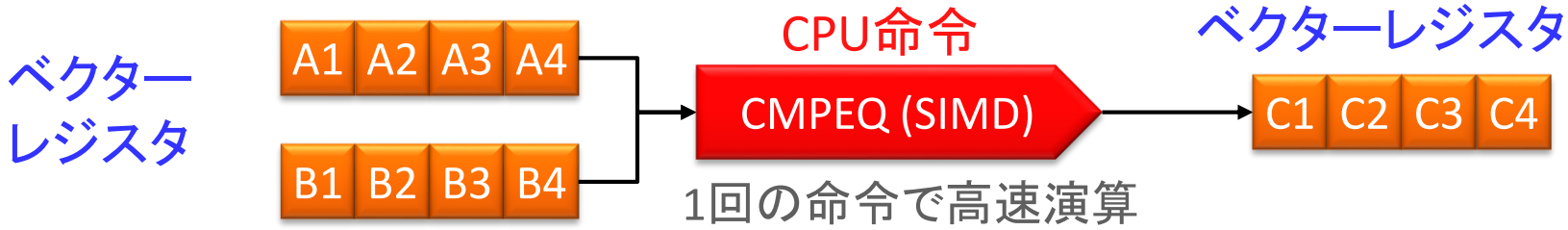
4回の一致
比較の場合

IF (A1 = B1) → C1
IF (A2 = B2) → C2
IF (A3 = B3) → C3
IF (A4 = B4) → C4

通常の命令セットの場合(1組のデータ演算から1つの結果を算出)



SIMD命令セットの場合(複数のデータを1回の演算命令で高速実行)



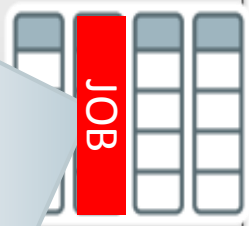
カラム型表は何故分析用クエリーが高速か? (4)

最新のプロセッサで搭載されているSIMDにより高速スキャン

インメモリ・カラム・ストア

EMP表

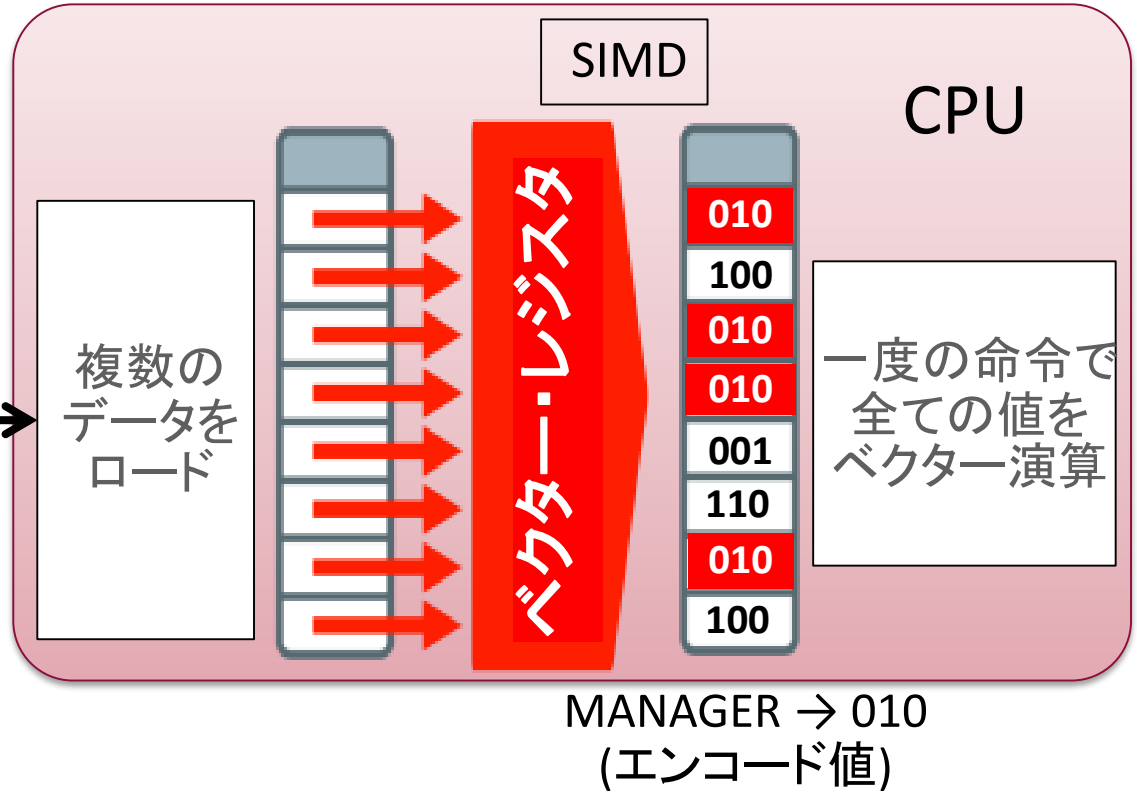
JOBカラム値	ディクショナリ値	ビット表現
ANALYST	0	000
CLERK	1	001
MANAGER	2	010
PRESIDENT	3	011
SALESMAN	4	100



001	100	100	010	100	010	010	000	011

ディクショナリ圧縮により
実データ値をビットデータとして扱うことで**より多くのデータをCPUレジスタにロード可能**

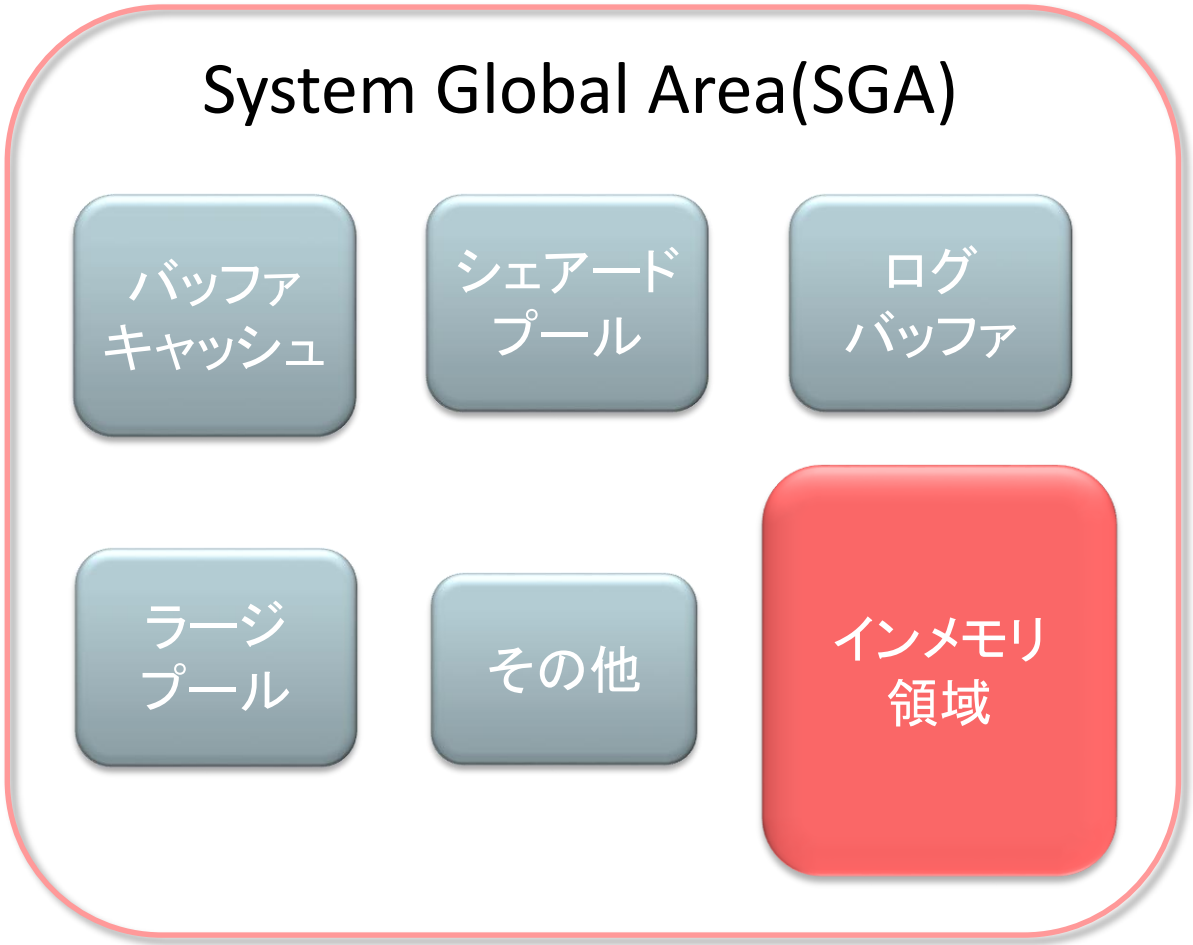
例:「MANAGER」職種を検索
(MANAGER → 010)



アジェンダ

1. Oracle Database In-Memory（DBIM）概要
2. インメモリ・カラム・ストアの詳細
3. DBIMのインメモリ・スキャン
4. DBIMの導入効果が高い処理／高くない処理
5. DBIMのパフォーマンス統計情報

インメモリ領域: SGA内の新しい領域



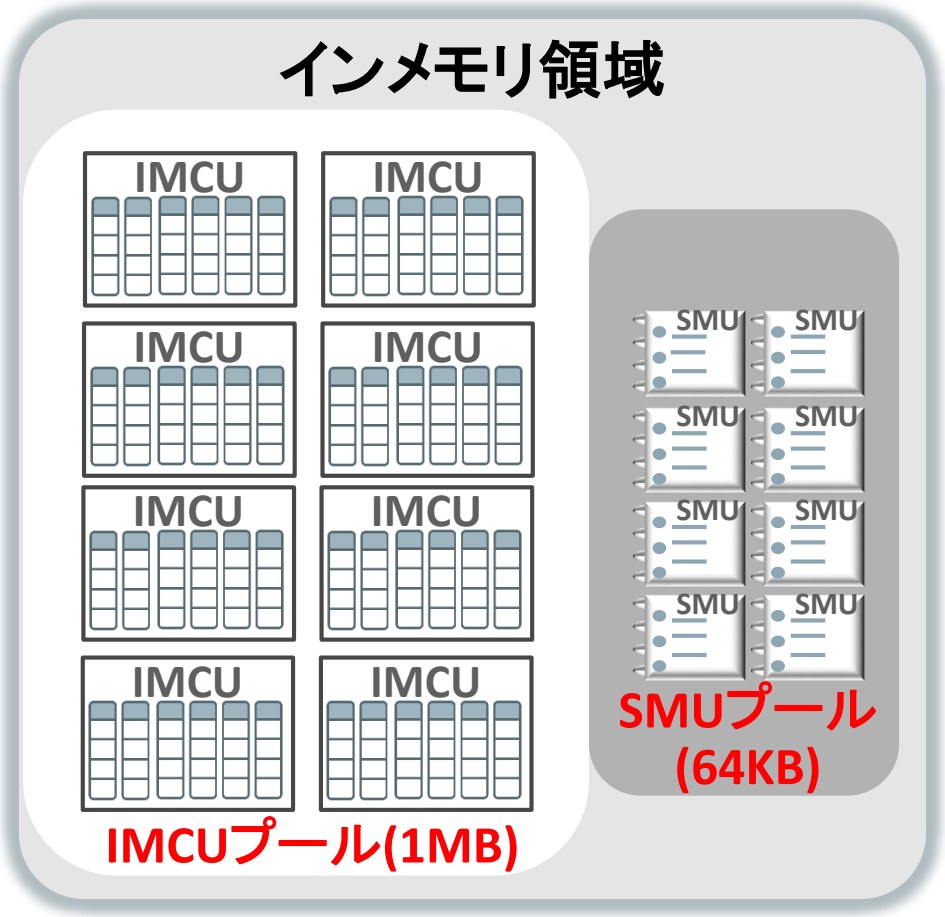
- 新しいインメモリ・カラム・フォーマットのデータを格納
- 初期化パラメータ「INMEMORY_SIZE」により設定
  - 最小サイズ: 100MB
- SGA_TARGETはこのインメモリ領域を格納できる十分大きな値の設定が必要
- 静的な領域で自動メモリー管理により増減しない

インメモリ領域: SGA内の新しい領域

確認方法

```
SELECT * FROM V$SGA;

NAME                             VALUE
--------------------      ----------
Fixed Size                   2927176
Variable Size              570426808
Database Buffers          4634022912
Redo Buffers                13848576
In-Memory Area          10244836480
```

インメモリ領域: 構成



- 2つのサブプールで構成:
  - IMCU(1MB)プール:
    IMCU(In-Memory Compression Units)を格納
  - SMU(64KB)プール:
    SMU(Snapshot Metadata Units)を格納
- IMCUはカラム書式のデータを格納
- SMUはメタデータとトランザクション情報を格納

インメモリ領域の確認

- V$INMEMORY_AREA:
  現状の各プールのサイズと状態を表示

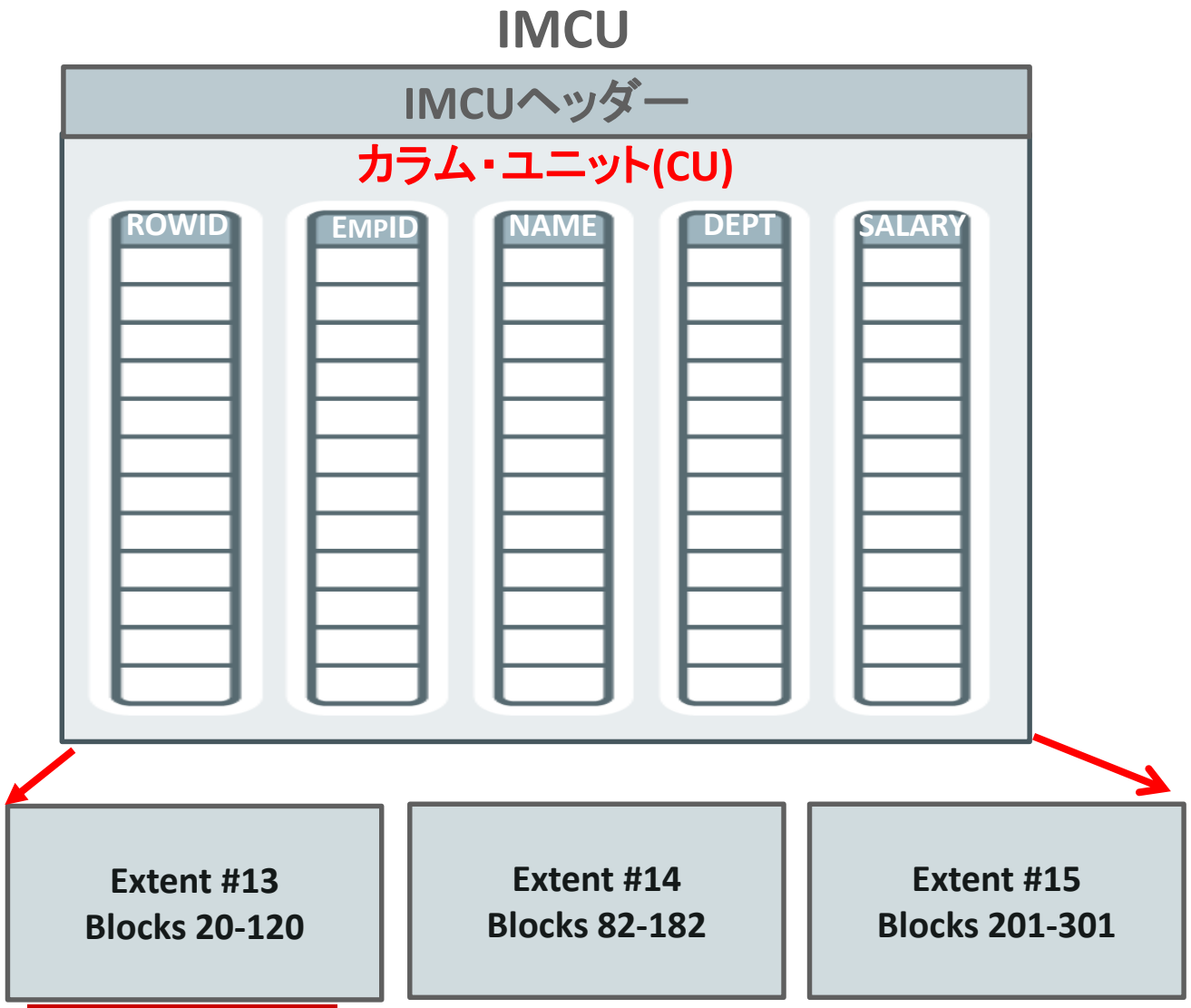
```
col pool for a10
col status for a20

select pool, alloc_bytes/1024/1024 alloc_MB,
    used_bytes/1024/1024 used_MB,
    (alloc_bytes - used_bytes)/1024/1024 free_MB,
    populate_status status
from v$inmemory_area;
```

割当領域　使用領域　空き領域

```
POOL          ALLOC_MB    USED_MB    FREE_MB STATUS
---------- ---------- ---------- ---------- --------------------
1MB POOL         57338      42153      15185 DONE
64KB POOL        14320   589.3125 13730.6875 DONE
```

IMCUプール
SMUプール

IMCU : In-Memory Compression Unit



- カラム型オブジェクトの管理単位
  - カラム型データをある程度の行数のセットで保持（例：50万行程度）
  - 格納される行は1つ以上の表エクステントから取得
- IMCUの実サイズは行サイズ、圧縮率等により変化（固定値ではない）
- カラム毎に分離／近接したカラム・ユニット(CU)として保存
  - Rowidも同様に1つのCUとして保存

IMCUの確認(1) – 各オブジェクトのIMCU数

- V$IM_HEADER:
  現在のインメモリ・カラムストア内のIMCU数の確認

```
col object_name for a20

SELECT OBJECT_NAME, count(*) NUM_IMCU
FROM  V$IM_HEADER i, DBA_OBJECTS o
WHERE i.objd = o.object_id
group by object_name order by 1;

OBJECT_NAME            NUM_IMCU
-------------------- ----------
CUSTOMER                      3
DATE_DIM                      1
LINEORDER                   563
PART                          3
SUPPLIER                      1
```

IMCUの確認(2)

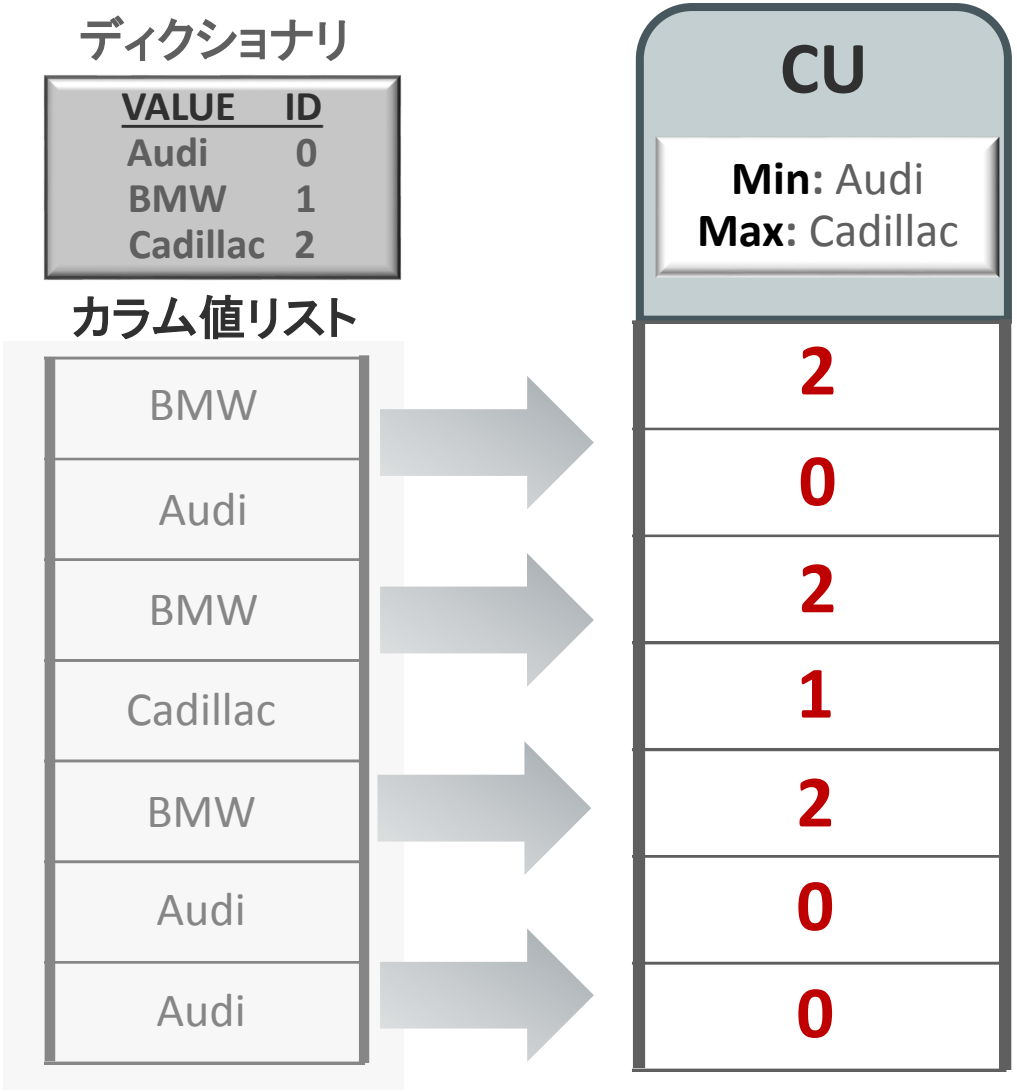
- V$IM_HEADER:
  現在のインメモリ・カラムストア内のIMCUのリスト

```
col object_name for a20
col tsname for a15

SELECT OBJECT_NAME, ts.NAME TSNAME, ALLOCATED_LEN/1024/1024 ALLOC_MB,
       NUM_ROWS, NUM_COLS
FROM  V$IM_HEADER i, DBA_OBJECTS o, v$tablespace ts
WHERE i.objd = o.object_id and i.tsn = ts.ts#
order by 1, 2;
```

IMCUサイズ IMCU内行数 カラム数

OBJECT_NAME	TSNAME	ALLOC_MB	NUM_ROWS	NUM_COLS
CUSTOMER	TS_DATA	39	495602	8
CUSTOMER	TS_DATA	33	415593	8
CUSTOMER	TS_DATA	47	588805	8
DATE_DIM	TS_DATA	1	2556	17
PART	TS_DATA	1	14807	9
PART	TS_DATA	14	591442	9
PART	TS_DATA	14	593751	9
SUPPLIER	TS_DATA	8	100000	7

CU：カラムユニット (Column Unit)



- IMCUに格納されている各カラム値の管理単位
- 全CUは自動的に最小／最大値を保存
  - インメモリ・ストレージ索引
- 圧縮フォーマット
  - 例）ディクショナリ圧縮
    CUは実際の値ではなく、サイズの小さいディクショナリIDをデータ値として格納
    → 圧縮した状態で検索が可能
  - 他の圧縮方式と組合わせることも可能

CUの確認方法

- V$IM_COL_CU：CUに関する情報の確認例

VARCHAR2型列：UTL_RAW.CAST_TO_VARCHAR2
DATE型列： DBMS_STATS.CONVERT_RAW_VALUE
（プロシジャ）

```
col obj_name for a20
select
   OBJECT_NAME, DICTIONARY_ENTRIES Dict_Entries,
   UTL_RAW.CAST_TO_NUMBER(MINIMUM_VALUE) MIN_VALUE,
   UTL_RAW.CAST_TO_NUMBER(MAXIMUM_VALUE) MAX_VALUE
from v$im_col_cu, dba_objects
where objd = object_id
and object_name = 'PART' and owner = 'SSB'
and column_number = 1
order by 1;

OBJECT_NAME          DICT_ENTRIES  MIN_VALUE  MAX_VALUE
-------------------- ------------ ---------- ----------
PART                        14807     927875     942681
PART                       591442     336433     927874
PART                       593751          1    1200000
```

表名とスキーマ名と
カラム番号の指定

アジェンダ

1. Oracle Database In-Memory（DBIM）概要
2. インメモリ・カラム・ストアの詳細
3. DBIMのインメモリ・スキャン
4. DBIMの導入効果が高い処理／高くない処理
5. DBIMのパフォーマンス統計情報

データベースの検索処理レイヤ

データベースの
検索処理

データベースの検索処理レイヤ

データ処理層

データスキャンで取得したデータの処理

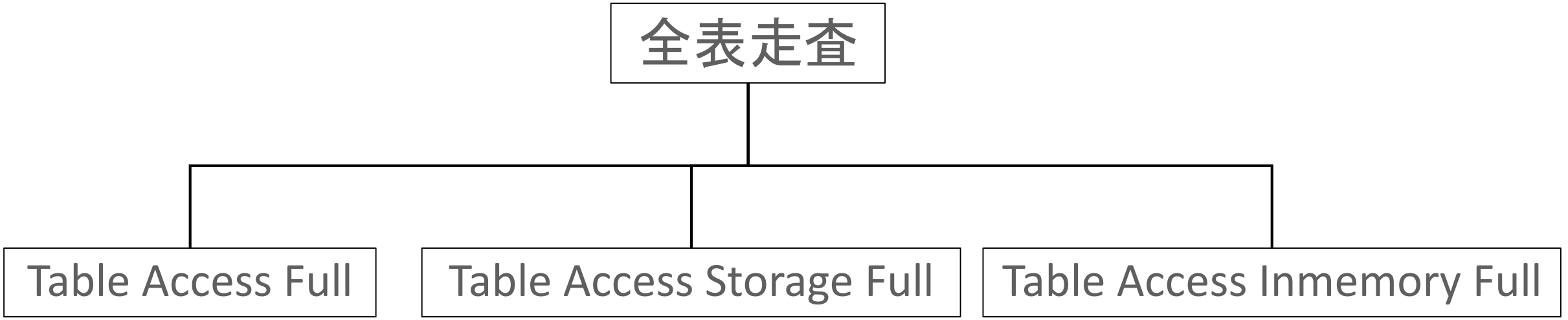
- 集計演算
- ジョイン
- ソート etc

データ・スキャン層

該当データの取得＋フィルタリング

- 全表走査 (Full Table Scan)
- 索引走査 (Index Range Scan)

全表走査の分類



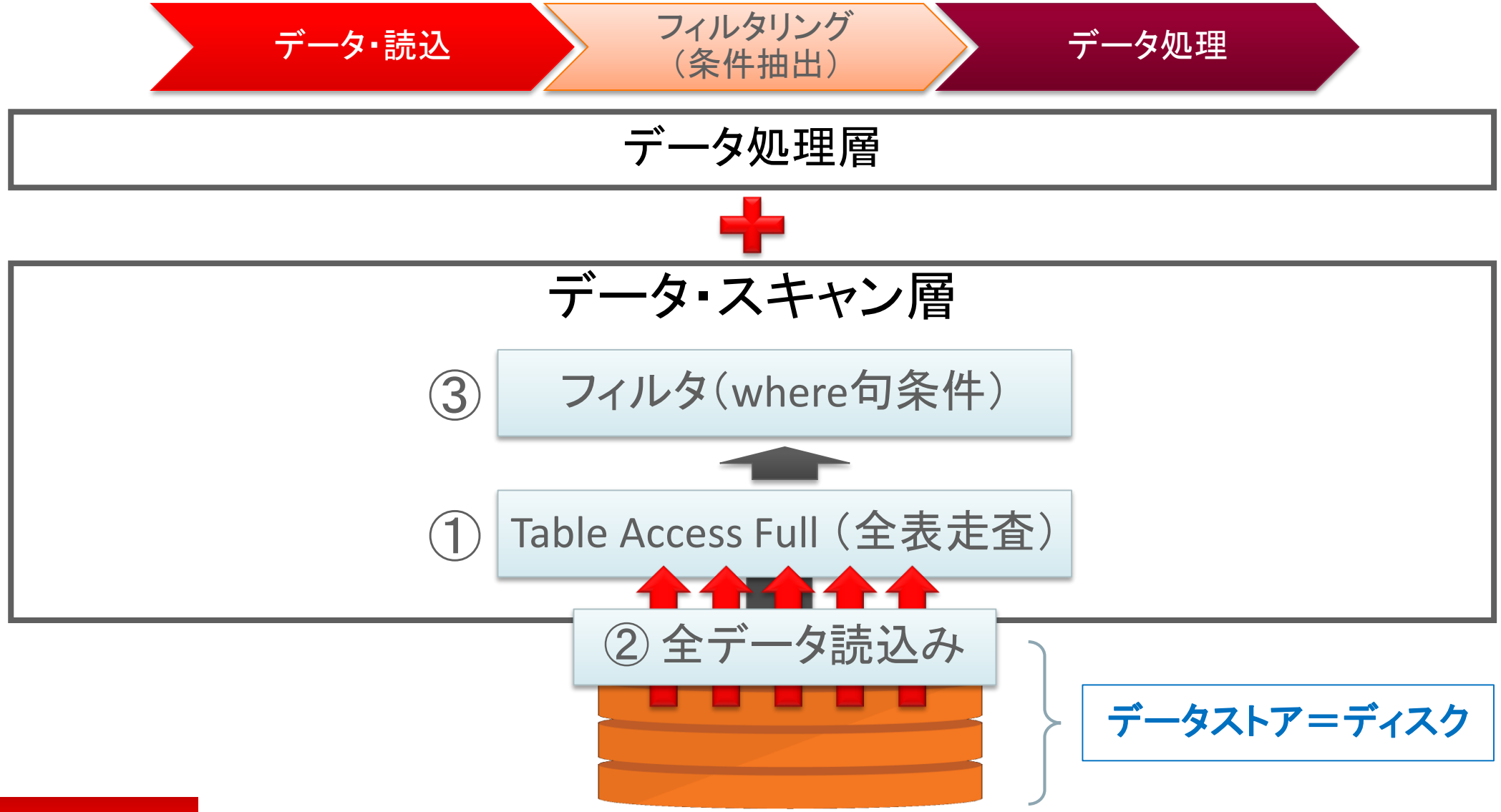
一般的な全表走査

Exadataの全表走査
（スマート・スキャン）

インメモリの全表走査
（インメモリ・スキャン）

インメモリ・スキャンは
全表走査の発展形

全表走査(Table Access Full)のイメージ



実行計画例 (Table Access Full)

SALES表の全表走査

```
select count(*) from sales
where quantity_sold > 30 or (quantity_sold < 10 and prod_id = 5000);
```

```
Plan hash value:672559287
-------------------------------------------------------------------
| Id  | Operation                    | Name        | Pstart| Pstop |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |       |       |
|   1 |  SORT AGGREGATE              |             |       |       |
|   2 |   PARTITION RANGE ALL        |             |     1 |    16 |
|*  3 |    TABLE ACCESS FULL         | SALES       |     1 |    16 |
-------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
 3 - filter("QUANTITY_SOLD">30 OR "PROD_ID"=5000 AND
           "QUANTITY_SOLD"<10)
```

全表走査して本条件による
フィルタリング実行

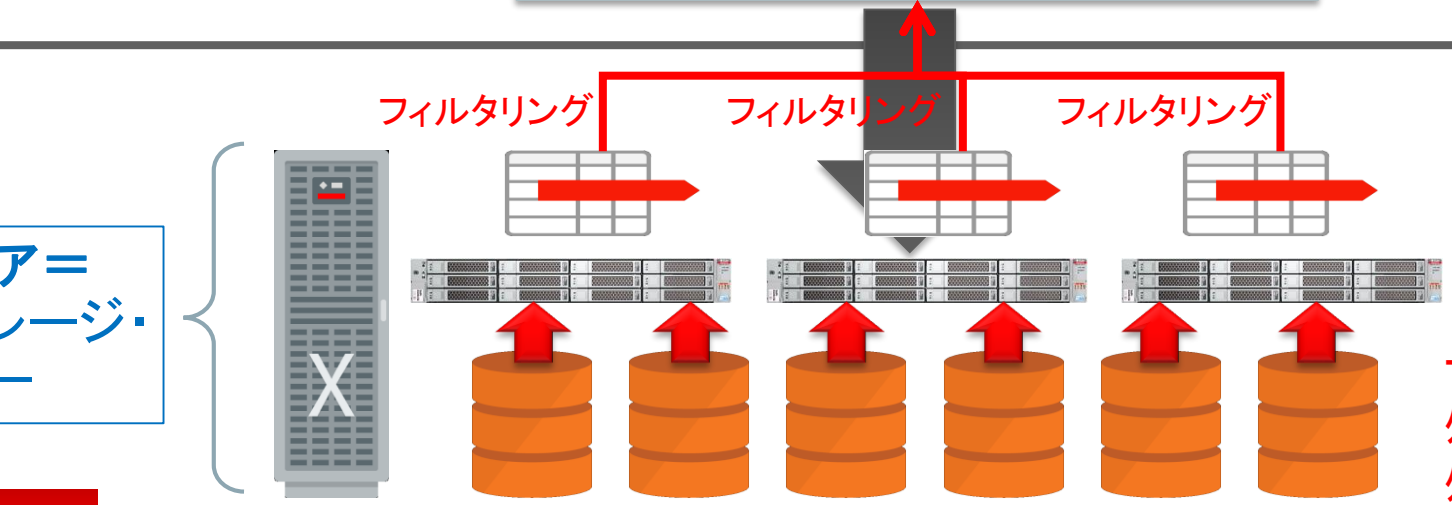
Exadataのスマート・スキャンのイメージ

データ処理層

＋

データ・スキャン層

① スマートスキャン
（Table Access Storage Full）



データストア＝
Exadata ストレージ・サーバー

② Exadataのスマート・スキャンによりストレージ層でフィルタリング

データスキャン層のフィルタリング処理をExadataのスマート・スキャン処理に「プッシュダウン」する

実行計画例 (Table Access Storage Full)

SALES表のExadata スマート・スキャン

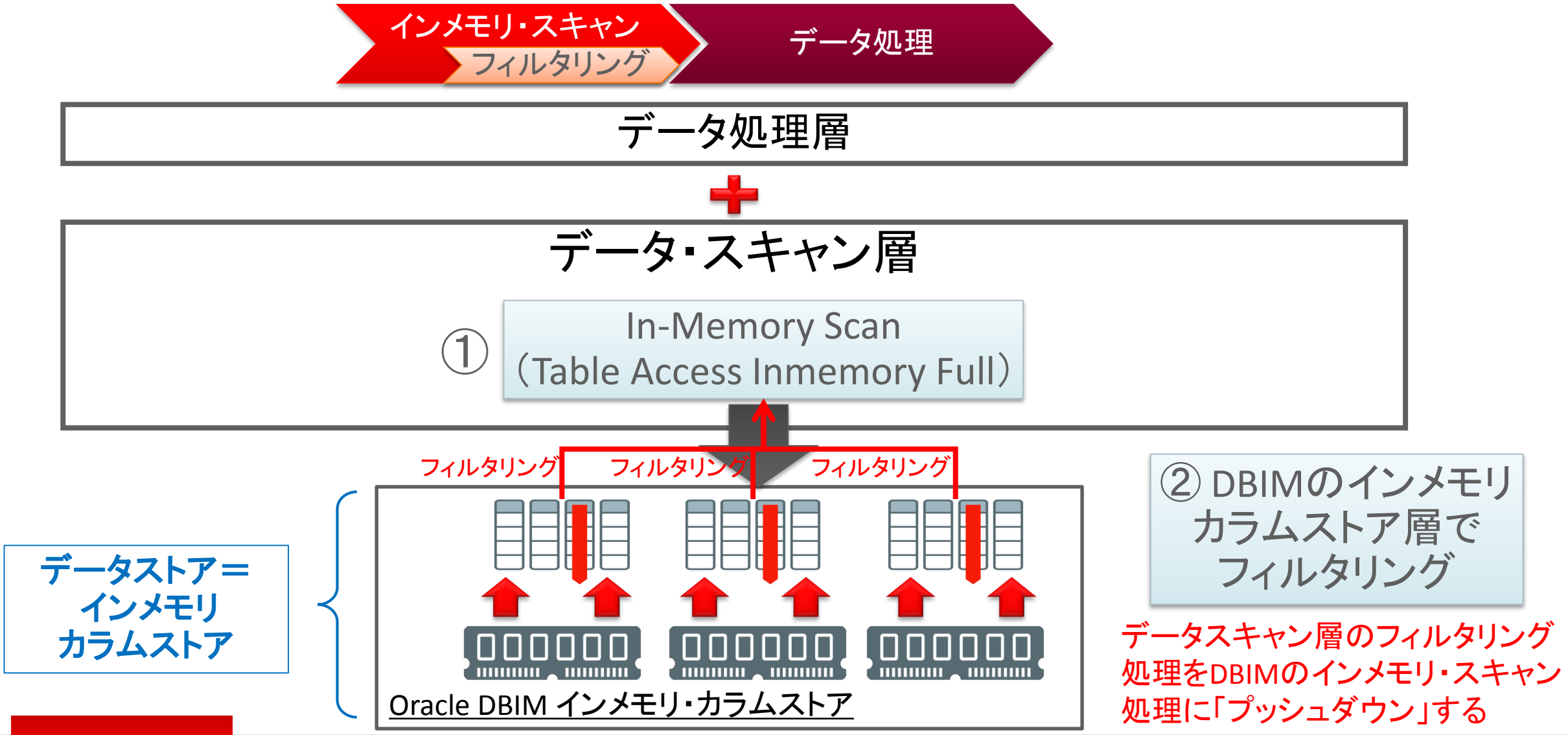
```
select count(*) from sales
where quantity_sold > 30 or (quantity_sold < 10 and prod_id = 5000);
```

```
Plan hash value:672559287
-------------------------------------------------------------------
| Id  | Operation                    | Name        | Pstart| Pstop |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |       |       |
|   1 |  SORT AGGREGATE              |             |       |       |
|   2 |   PARTITION RANGE ALL        |             |     1 |    16 |
|*  3 |    TABLE ACCESS STORAGE FULL | SALES       |     1 |    16 |
-------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
 3 - storage("QUANTITY_SOLD">30 OR "PROD_ID"=5000 AND
           "QUANTITY_SOLD"<10)
     filter("QUANTITY_SOLD">30 OR "PROD_ID"=5000 AND
           "QUANTITY_SOLD"<10)
```

同一のハッシュ値であり、実行計画的には全表走査と同じ扱い

「storage」は、Exadataのスマート・スキャンによりWhere条件のフィルタリングがストレージ・サーバーにオフロードされたことを示している（プッシュダウン）

DBIMのインメモリ・スキャンのイメージ



実行計画例 (Table Access Inmemory Full)

SALES表がインメモリ化されている場合:

```
select count(*) from sales
where quantity_sold > 30 or (quantity_sold < 10 and prod_id = 5000);
```

```
Plan hash value:672559287
-------------------------------------------------------------------
| Id  | Operation                          | Name        | Pstart| Pstop |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |             |       |       |
|   1 |  SORT AGGREGATE                    |             |       |       |
|   2 |   PARTITION RANGE ALL              |             |     1 |    16 |
|*  3 |    TABLE ACCESS INMEMORY FULL      | SALES       |     1 |    16 |
-------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - inmemory(("QUANTITY_SOLD">30 OR "QUANTITY_SOLD"<10) AND
                (“QUANTITY_SOLD”>30 OR “PROD_ID”=5000 AND
                 "QUANTITY_SOLD"<10))
       filter("QUANTITY_SOLD">30 OR "PROD_ID"=5000 AND
              "QUANTITY_SOLD"<10)
```

同一のハッシュ値であり、実行計画的には全表走査と同じ扱い

フィルタリングを効率化するための暗黙条件の付加

「inmemory」は、インメモリスキャンによりWhere条件のフィルタリングがオフロードされたことを示している（プッシュダウン）

フィルタリングを効率化するための暗黙条件の生成

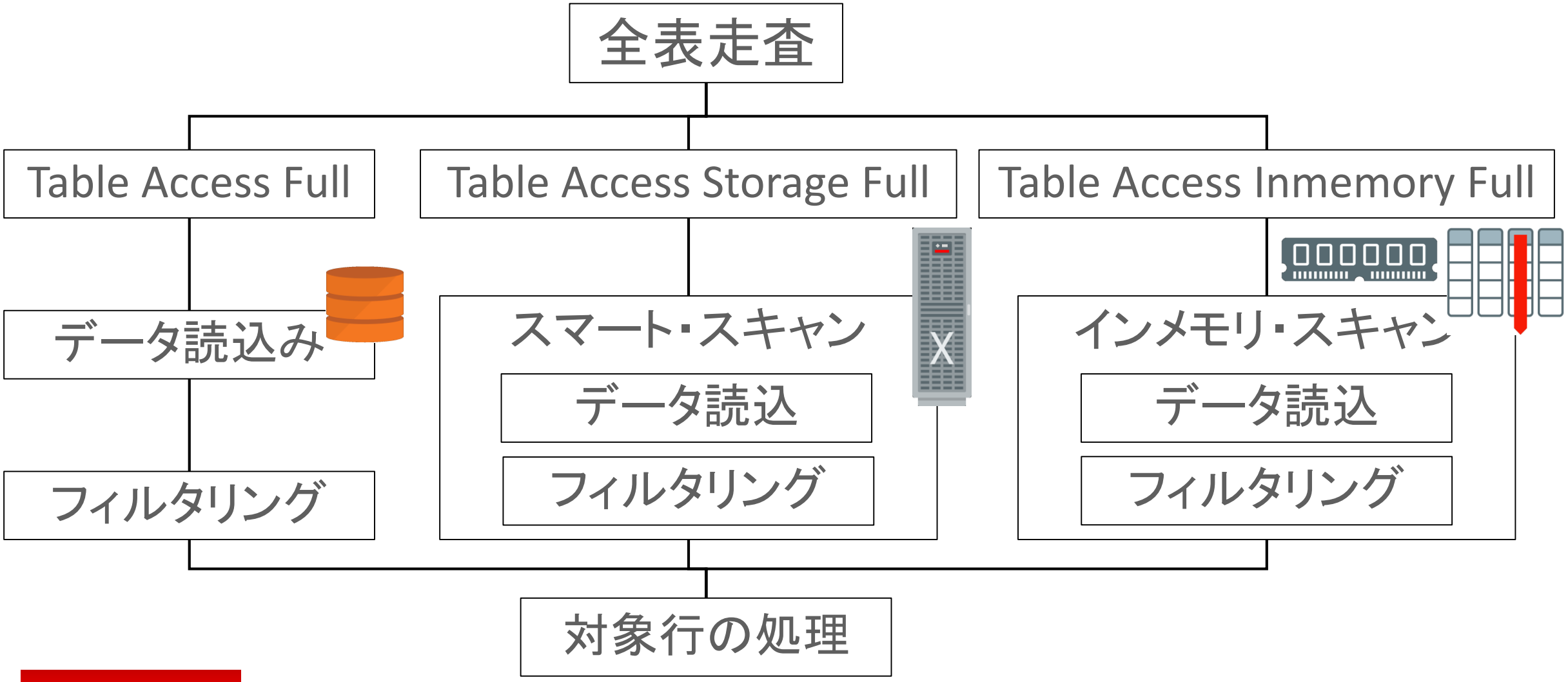
- 暗黙条件の生成

  例）

  - 実際の条件：「col1 = 2 OR (col1 = 1 AND col2 > 1) 」
    → 暗黙条件追加：「(**col1 IN (1, 2)**) AND (col1 = 2 OR (col1 = 1 AND col2 > 1)) 」
    ※ 生成された暗黙条件：「col1 IN (1, 2)」

  このcol1列に対する暗黙条件（col1 IN (1,2)）のフィルタリングをインメモリ・スキャンすることで、実際の条件（col1 = 2 OR (col1 = 1 AND col2 > 1) ）のフィルタリングを効率的に処理する

全表走査の分類(まとめ)



インメモリ・スキャンのフィルタリングが高速な理由

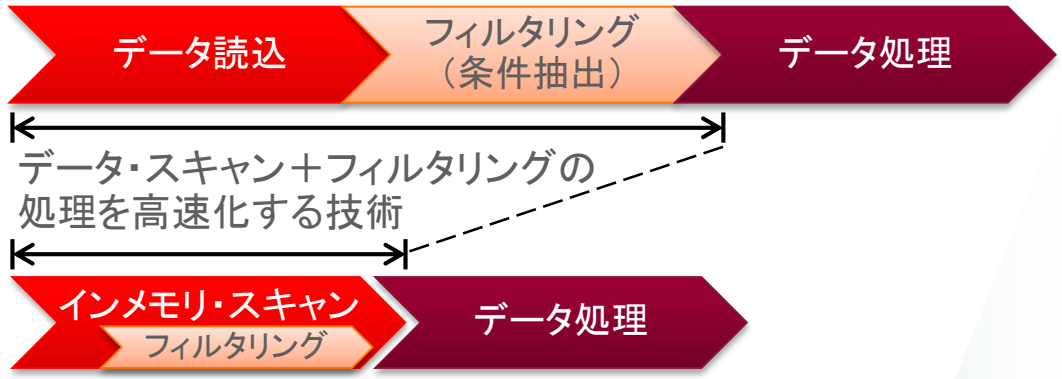
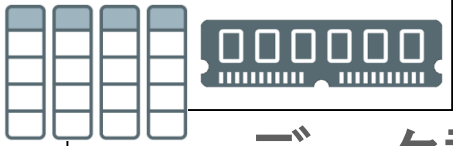
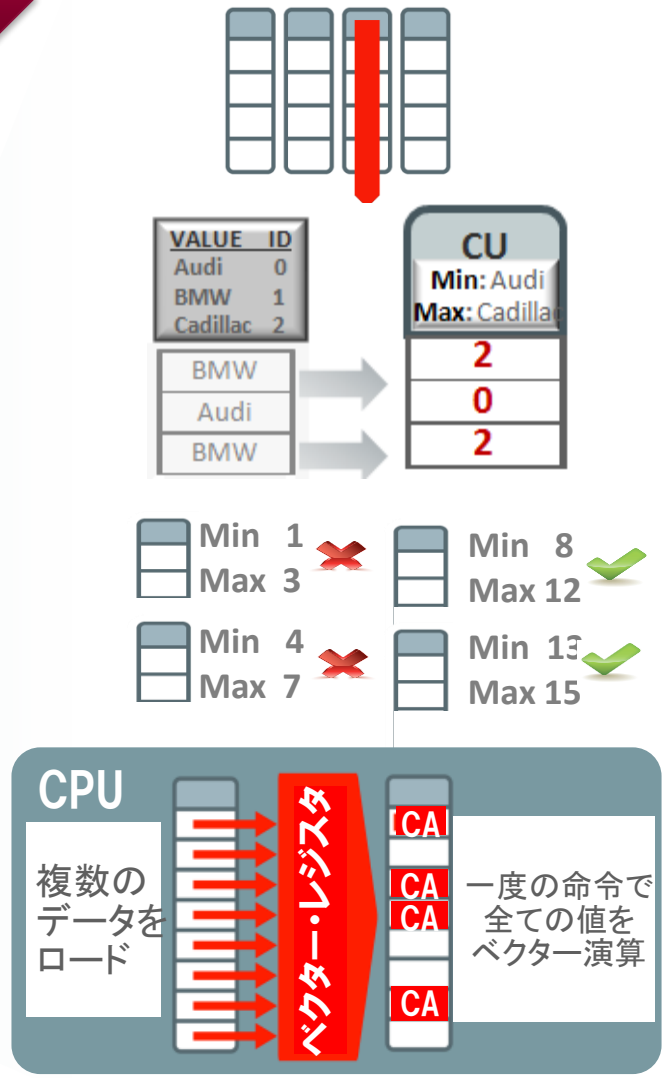


Table Access Inmemory Full



データ読込み
（インメモリ・スキャン）

フィルタリング



1. 必要なカラムのみアクセス

2. 効果的な圧縮技術により
圧縮した状態で検索が可能
(ディクショナリ圧縮)

3. インメモリ・ストレージ索引により
最小限のIMCUのみスキャン

4. 最新のプロセッサで搭載されて
いるSIMDにより高速スキャン
（ベクター・スキャン）

アジェンダ

1. Oracle Database In-Memory（DBIM）概要
2. インメモリ・カラム・ストアの詳細
3. DBIMのインメモリ・スキャン
4. DBIMの導入効果が高い処理／高くない処理
5. DBIMのパフォーマンス統計情報

DBIMの利用効果の大きい処理

少数列に対する大量行の処理に効果大

1. 処理特性
ディスクI/Oの物理読込（Physical Reads）量が大きいもの

- 大量データの全表走査（Full Table Scan）
- 大量データの索引走査（Index Range Scan/Bitmap Index Scan）

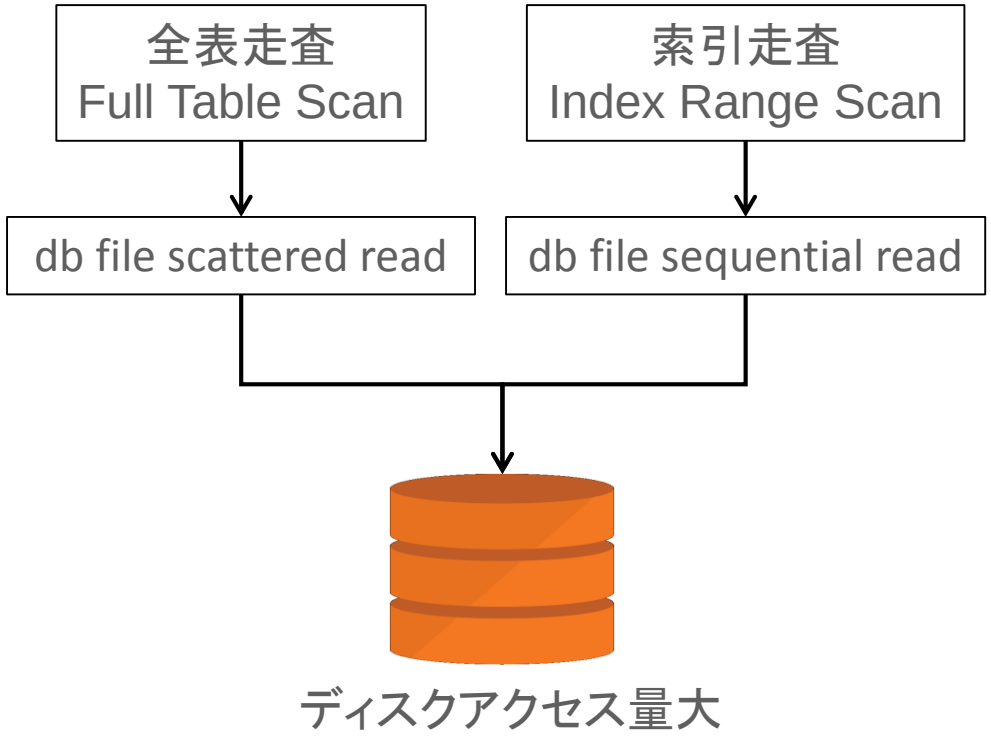
2. SQLの特徴
大量行の表を含む分析クエリ

- 複数表を利用した結合処理とフィルタ条件（WHERE xxx = :abc）
- 集計演算処理（特に MAX, MIN, COUNT）
- 中間一致検索（ユニーク値の列は除く）

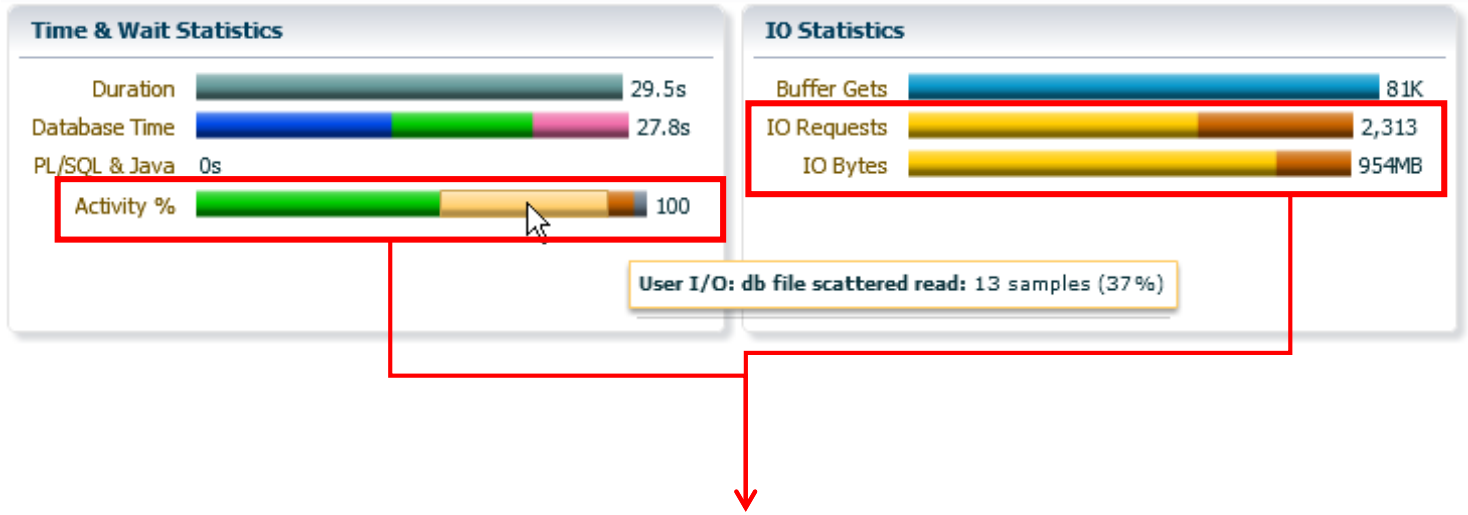
DBIMの利用効果の大きい処理

ディスクI/Oの物理読込（Physical Reads）量が大きいSQLとは？

インメモリ化によるパフォーマンス改善が
期待できる処理



SQL Monitorによる確認例



SQLの処理で多くのディスク・アクセスが発生している
(User I/O: db file scattered read)

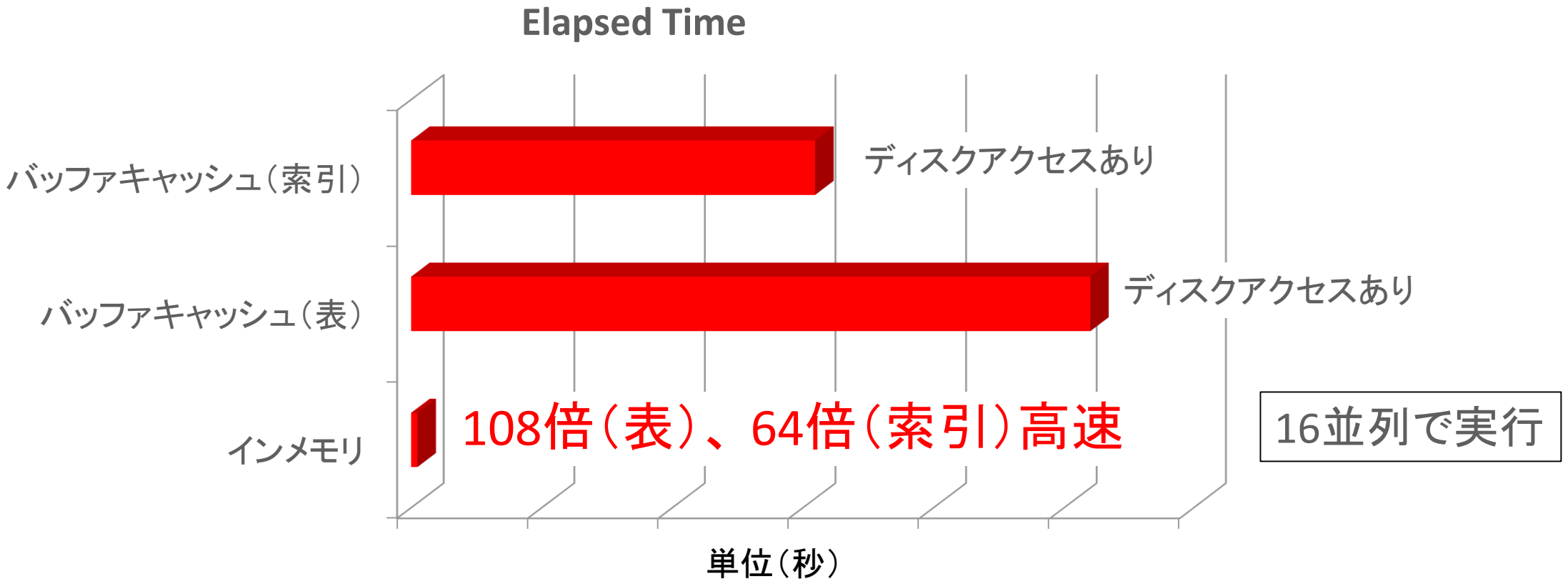
このようなSQLであればインメモリ化による
パフォーマンス改善の効果を期待できる

DBIMの利用効果の大きい処理

大量行の表を含むジョイン処理、集計処理でディスクアクセスがある場合と比較

```
select sum(lo_quantity) from lineorder, customer
where lo_custkey = c_custkey and c_nation in ( 'JAPAN', 'CHINA' ) ;
```

LINEORDER: 3億件
CUSTOMER: 150万件



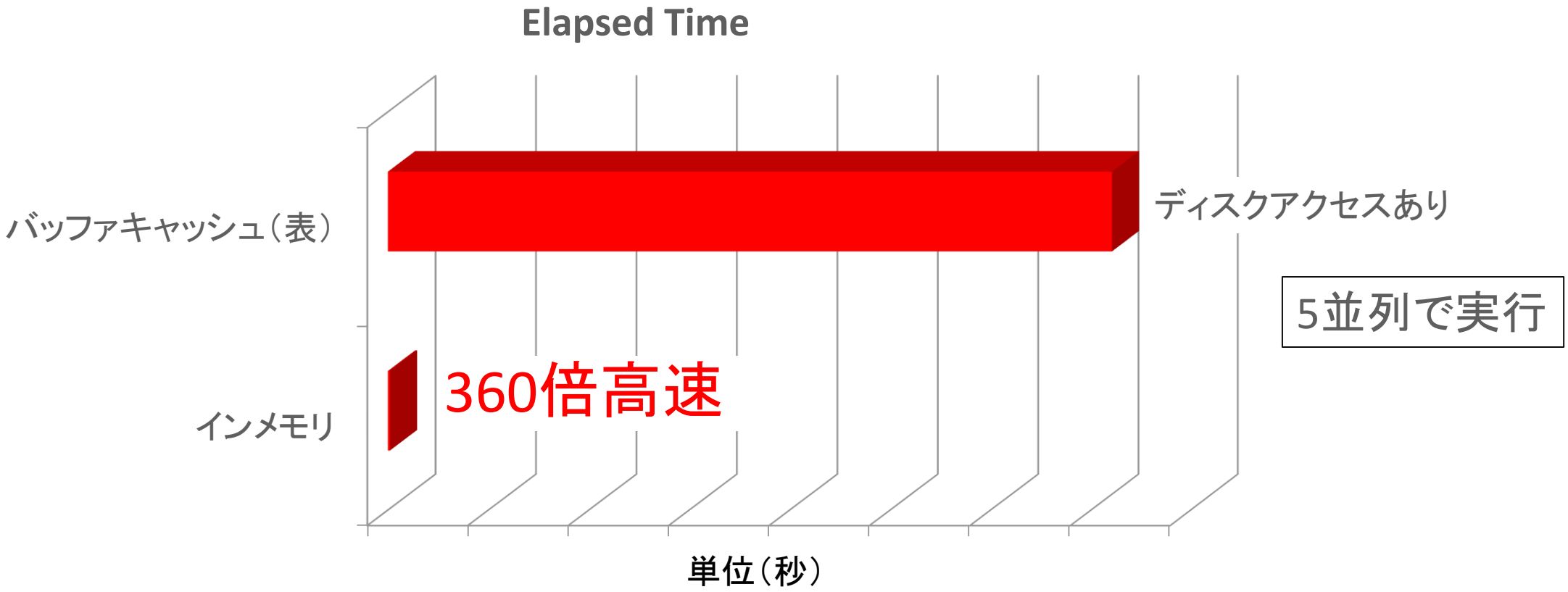
16並列で実行

DBIMの利用効果の大きい処理

大量行の表を含むジョイン処理、集計処理でディスクアクセスがある場合と比較

```
select max(lo_quantity) from lineorder
where lo_orderkey between 1 and 50000000 ;
```

LINEORDER: 3億件



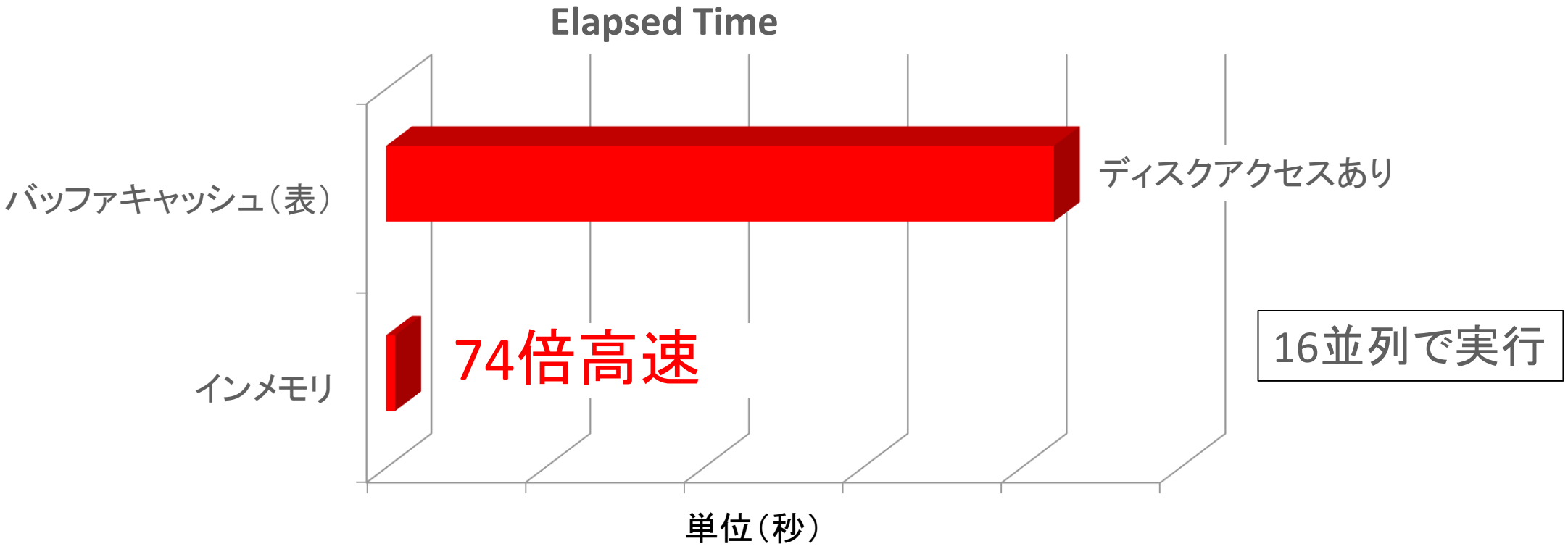
5並列で実行

DBIMの利用効果の大きい処理

大量行の表に対する中間一致検索でディスクアクセスあり

```
select sum(lo_quantity) from lineorder, customer
 where lo_custkey = c_custkey and c_region like '%RICA%' ;
```

LINEORDER: 3億件
CUSTOMER: 150万件



16並列で実行

Q)大きなサイズのバッファ・キャッシュを確保して、全データをキャッシュ内に保持して実行すればインメモリ検索と変わらないのか？

→ **No**

検証1）フル・キャッシュ（行） vs インメモリ（列）

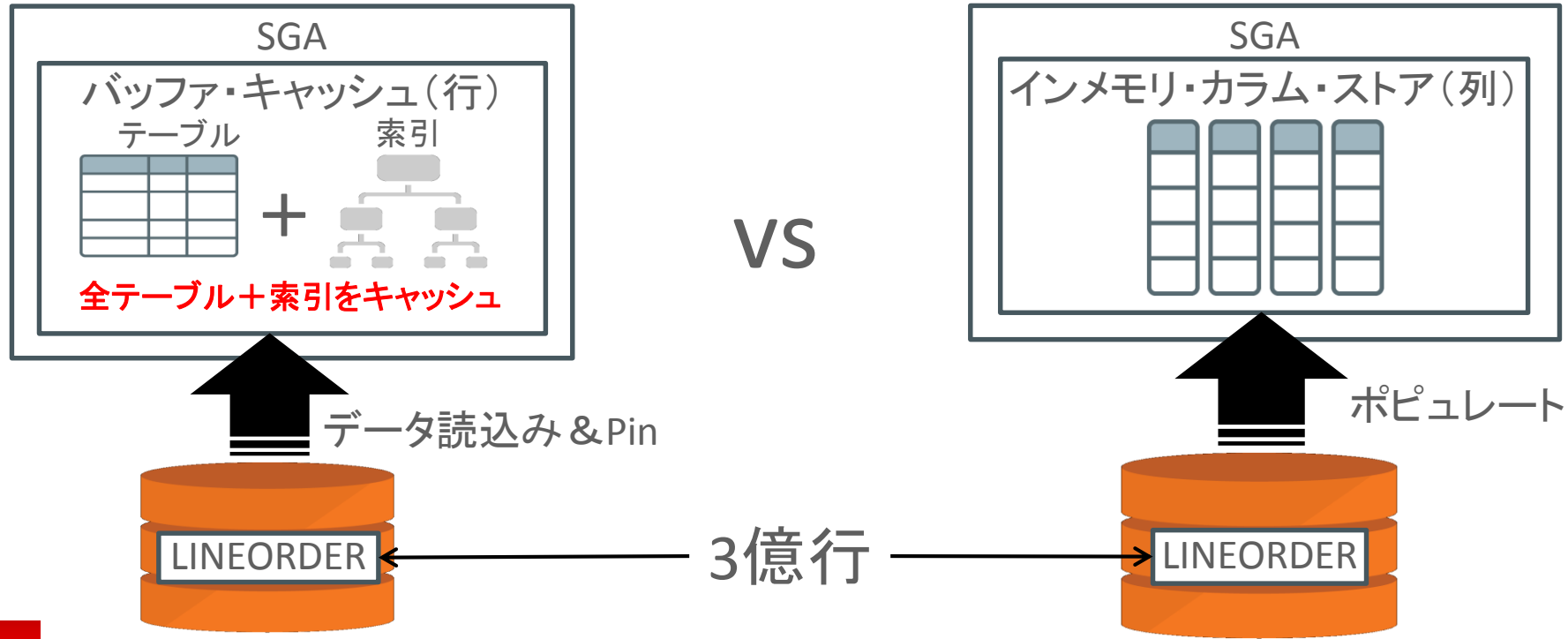
全表スキャン vs 索引スキャン vs インメモリ・スキャン

バッファ・キャッシュに表の全データがキャッシュされている場合とインメモリ検索の場合のパフォーマンスの違いを計測

実行SQL

3億行から5,000万行を抽出する

```
select sum(lo_quantity) from lineorder
where lo_orderkey between 1 and 50000000 ;
```



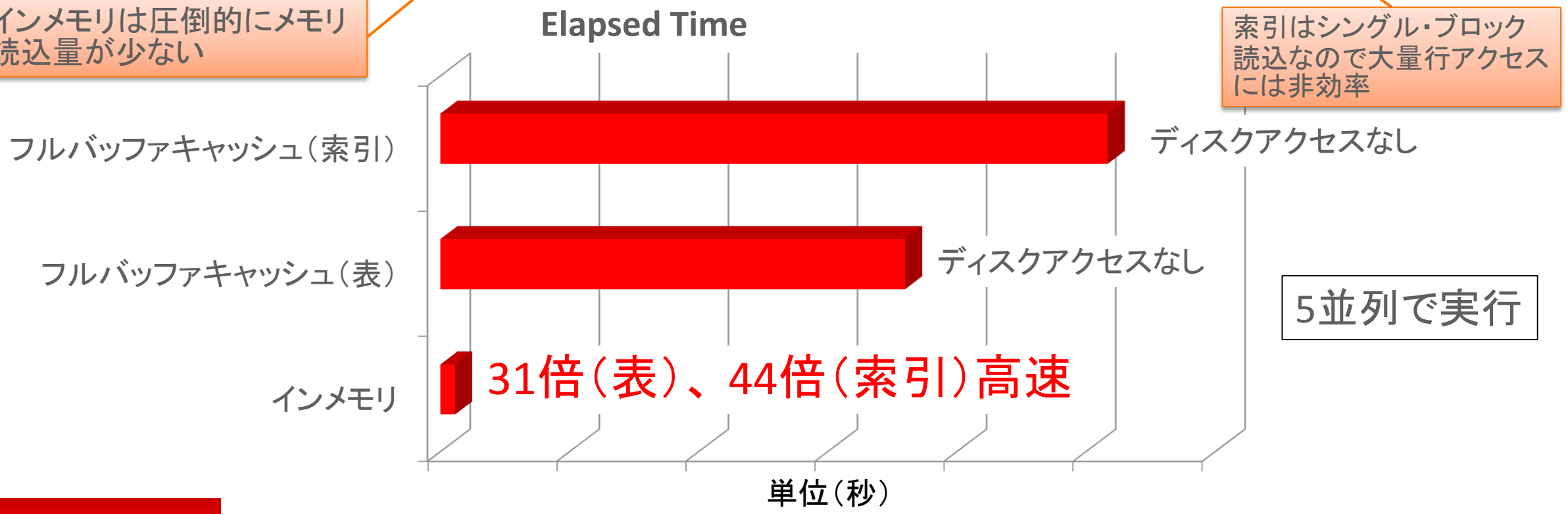
検証1）フル・キャッシュ（行）vs インメモリ（列）

3億行の表から5,000万行を抽出するSQL

	DBIM	表（フルキャッシュ）	索引（フルキャッシュ）
読込メモリブロック数 (consistent gets)	224	2,285,176	497,832

インメモリは圧倒的にメモリ読込量が少ない

索引はシングル・ブロック読込なので大量行アクセスには非効率



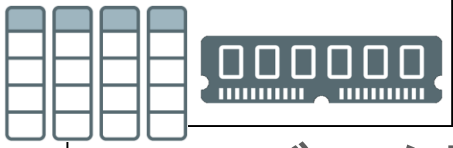
5並列で実行

5千万行のアクセスを高速にアクセス出来る理由

再掲）インメモリ・スキャンの高速フィルタリング

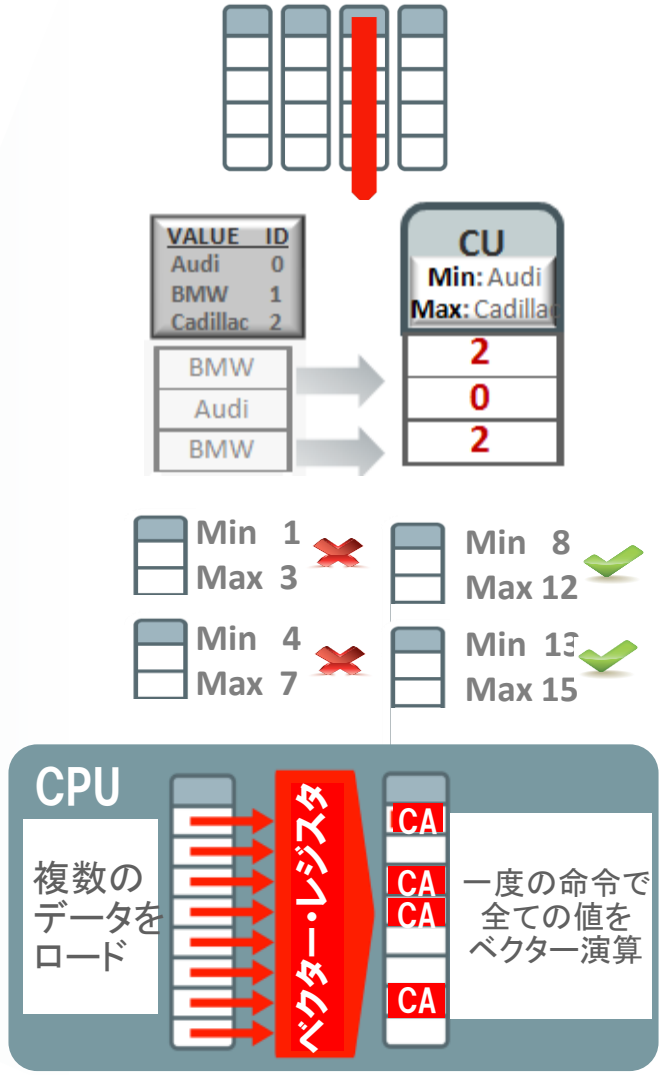
全表走査

Table Access Inmemory Full



データ読込み
（インメモリ・スキャン）

フィルタリング



1. 必要なカラムのみアクセス

2. 効果的な圧縮技術により圧縮した状態で検索が可能
(ディクショナリ圧縮)

3. インメモリ・ストレージ索引により
最小限のIMCUのみスキャン

4. 最新のプロセッサで搭載されているSIMDにより高速スキャン
（ベクター・スキャン）

Q)インメモリ化すると全ての検索処理を
高速化出来るのか？

→ **No**

DBIMの利用効果が大きくない処理

1. 索引により最適化されている
（大きな表から少ない行データを検索する）

- 数億／数千万行の表から1～数百行取得するような検索
- ハードウェア・リソースの観点からインメモリ検索の
並列度を高く設定出来ない

2. 集計値を別の仕組みで保持している

- マテリアライズド・ビューにより集計値を保持
- 同一内の別カラム、また別表に集計値を保持

少ない検索結果件数による比較

索引 vs インメモリ

- 6億行の表から530行を検索するSQL（シリアル実行）

```
select lo_quantity from lineorder where lo_custkey = 1499999 ;
```

	索引	インメモリ
Elapsed Time	0.04秒	0.13秒

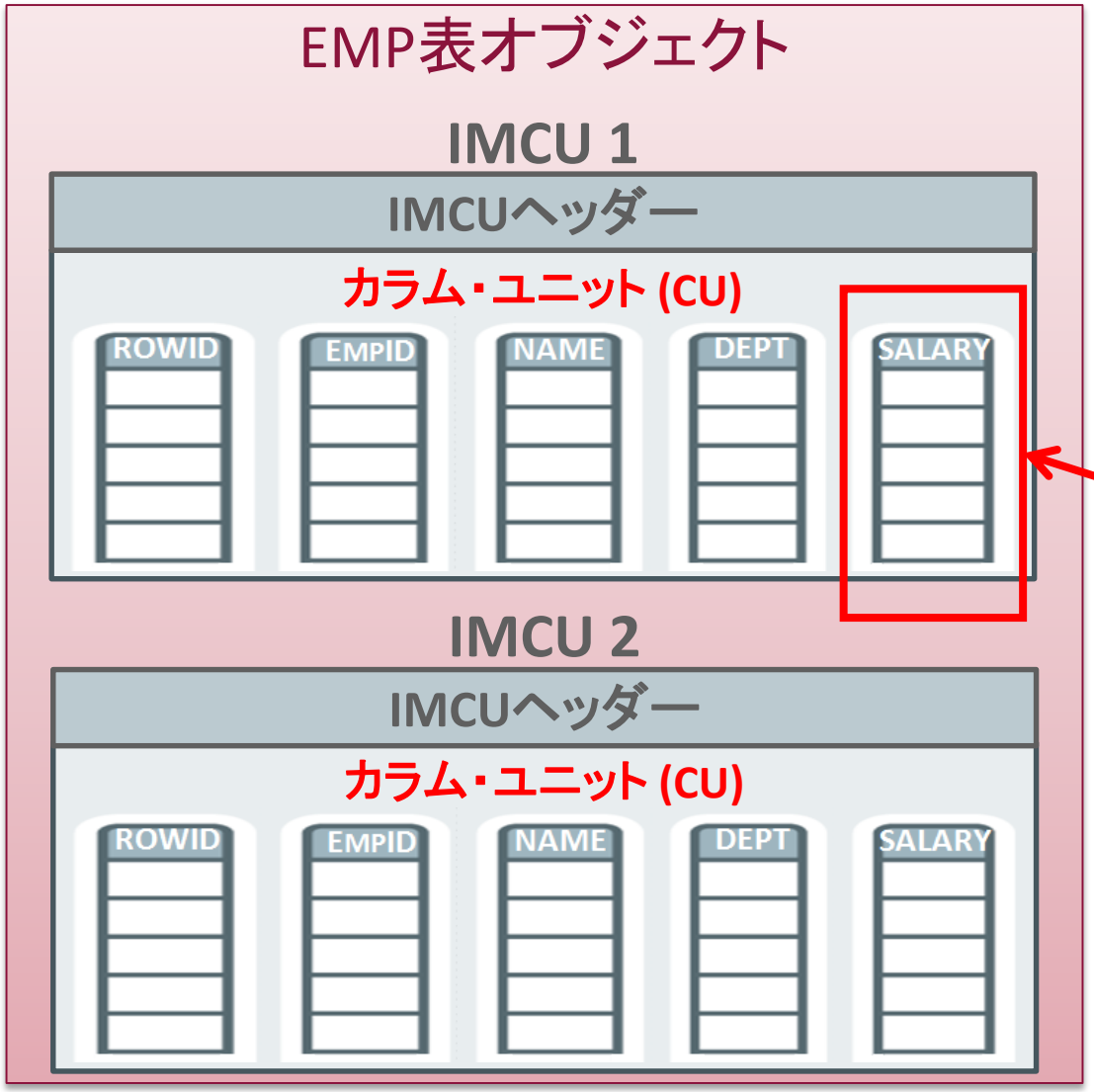
→ 表の格納行数が多く、検索結果件数が少ないほど索引が若干速くなるケースが多い

→ SQLヒントを指定しないと索引アクセス処理になる

アジェンダ

1. Oracle Database In-Memory（DBIM）概要
2. インメモリ・カラム・ストアの詳細
3. DBIMのインメモリ・スキャン
4. DBIMの導入効果が高い処理／高くない処理
5. DBIMのパフォーマンス統計情報

CU（カラム・ユニット）の確認



インメモリ・カラムストア内の
各オブジェクトを構成する
IMCU内の各カラムの構成
ユニット／カラム・データ部

インメモリ系の主要統計名（v$sysstat/v$mystat)

統計の確認方法）
v$sysstat/v$mystatビューとv$statnameで確認する

確認SQL例）

```
select display_name, value from v$mystat m, v$statname n
where m.STATISTIC#=n.STATISTIC# and display_name like 'IM scan %'
order by 1;
```

統計名	説明
IM scan CUs columns theoretical max	クエリーでアクセスされるであろう論理的な最大CU 数
IM scan CUs columns accessed	クエリーでアクセスしたCU 数
IM scan CUs pruned	ストレージ索引によって読み飛ばされたCU 数
IM scan rows	インメモリスキャンで論理的な読込対象となる行数
IM scan rows optimized	インメモリ・スキャンで実際に読み飛ばされた行数
IM scan rows projected	検索結果の処理対象行数（結果出力のための行数）

※多くのケースで「(IM scan rows) ≠ (IM scan rows projected) + (IM scan rows optimized) 」

効果の確認：必要なカラムのみアクセス (1)

CUSTOMER表の定義確認：300万件データ、8列の表

→ 1 IMCU中に8個のCU（列データ）×6個のIMCU = CU数の合計48

```
SQL> select count(*) from customer;

  COUNT(*)
----------
   3000000

SQL> desc customer
 Name                 Type
 -------------------- ------------
 C_CUSTKEY            NUMBER
 C_NAME               VARCHAR2(25)
 C_ADDRESS            VARCHAR2(25)
 C_CITY               CHAR(10)
 C_NATION             CHAR(15)
 C_REGION             CHAR(12)
 C_PHONE              CHAR(15)
 C_MKTSEGMENT         CHAR(10)
```

8列

IMCU数の確認）

```
SELECT OBJECT_NAME, count(*) NUM_IMCU
FROM  V$IM_HEADER i, DBA_OBJECTS o
WHERE i.objd = o.object_id
and object_name = 'CUSTOMER'
group by object_name;


OBJECT_NAME     NUM_IMCU
------------ ----------
CUSTOMER                6 ←合計 6 個のIMCU
```

効果の確認：必要なカラムのみアクセス (2)

テストSQL1の実行）※1カラムのみで処理

```
select c_region, count(*) from customer
group by c_region order by c_region;
```

実行結果）

```
C_REGION        COUNT(*)
------------ ----------
AFRICA             599436
AMERICA            599689
ASIA               600016
EUROPE             601031
MIDDLE EAST        599828
```

合計48CU中、6CUのみアクセス

統計名	値
IM scan CUs columns theoretical max	**48**
IM scan CUs columns accessed	**6**
IM scan CUs pruned	0
IM scan rows	3000000
IM scan rows optimized	0
IM scan rows projected	3000000

→ IMCU数 (6) × 8列数 /IMCU

→ 検索に利用する列は「C_REGION」のみ：IMCU数(6) x 1

→ 全件検索なのでストレージ索引利用なし

→ 全件検索なのでインメモリ・スキャン中に削減された行はなし

効果の確認：必要なカラムのみアクセス (3)

テストSQL2の実行） ※2カラムで処理

```
select c_region, sum(c_custkey)from customer
group by c_region order by c_region;
```

実行結果）

```
C_REGION      SUM(C_CUSTKEY)
------------ --------------
AFRICA             9.0032E+11
AMERICA            9.0016E+11
ASIA               8.9908E+11
EUROPE             9.0072E+11
MIDDLE EAST        8.9971E+11
```

合計48CU中、12CUのみアクセス

統計名	値
IM scan CUs columns theoretical max	**48**
IM scan CUs columns accessed	**12**
IM scan CUs pruned	0
IM scan rows	3000000
IM scan rows optimized	0
IM scan rows projected	3000000

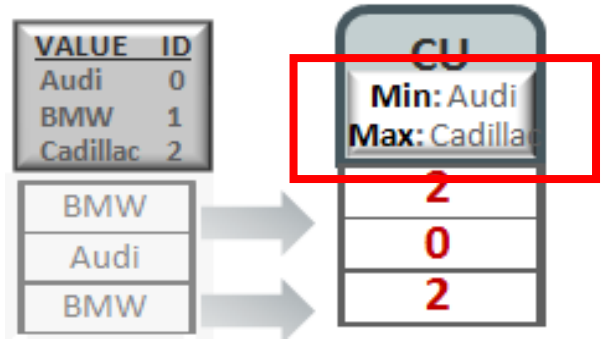
→ 6個のIMCU × 8列数 /IMCU

→「C_REGION」と「C_CUSTKEY」列を利用：IMCU数(6) × 2

→ 全件検索なのでストレージ索引利用なし

→ 全件検索なのでインメモリ・スキャン中に削減された行はなし

効果の確認：必要なカラムのみアクセス (4)



テストSQL3の実行） ※インメモリ・ストレージ索引を利用

```
select max(c_name)from customer;
```

MAX値の検索は、インメモリ・ストレージ索引のMAX値のみで検索が可能

実行結果）

```
MAX(C_NAME)
--------------------------
Customer#003000000
```

統計名	値
IM scan CUs columns theoretical max	48
IM scan CUs columns accessed	**6**
IM scan CUs pruned	0
IM scan rows	**3000000**
IM scan rows optimized	**3000000**
IM scan rows projected	**6**

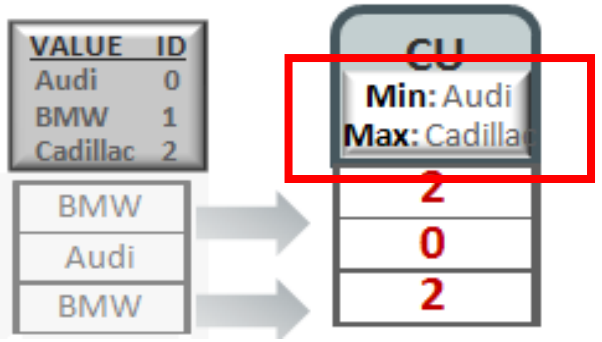
→ 6個のIMCU × 8列数 /IMCU

→「C_NAME」列のみを利用：IMCU数(6) × 1

→ インメモリ・ストレージ索引を利用するがCUスキップしない（全件検索のため）

→ インメモリ・ストレージ索引のみのアクセスなので全行の読込が削除(optimized)され、「C_NAME」のCUのMAX値の読込数である「6」がprojectedになる

効果の確認：インメモリ・ストレージ索引



テストSQL4の実行）※インメモリ・ストレージ索引によるCUスキップ

```
select count(*) from customer
where c_custkey between 1 and 5000;
```

実行結果）

```
COUNT(*)
----------
      5000
```

統計名	値
IM scan CUs columns theoretical max	48
IM scan CUs columns accessed	**1**
IM scan CUs pruned	**5**
IM scan rows	**3000000**
IM scan rows optimized	**2501589**
IM scan rows projected	5000

→ 6個のIMCU × 8列数 /IMCU

→「C_CUSTKEY」列を利用：IMCU数(6) × 1 – prunedの値 (5)

→ ストレージ索引により、5CUがスキップされた

→ インメモリ・スキャン対象の全行数（含むスキップ行）

→ インメモリ・スキャンにより読込スキップされた行

→ 最終検索結果対象の行数
※498411行(3000000-2501589)から5000行が条件マッチ

まとめ：本日お伝えしたこと

- Oracle DBIM概要
  - インメモリ・カラム・ストア／ディクショナリ圧縮／SIMD／インメモリ・ストレージ索引
- インメモリ・カラム・ストア詳細
  - IMCU / CU
  - V$INMEMORY_AREA, V$IM_HEADERS, V$IM_COL_CU
- インメモリ・スキャン
  - 全表走査の発展形 (TABLE ACCESS INMEMORY FULL)
  - フィルタリング処理をインメモリ・スキャンへプッシュ・ダウン
- DBIMの効果の高い処理／高くない処理
  - 効果が高い：少ない列の大量行アクセス処理
  - 効果が高くない：索引に最適化されている／集計値を別の仕組みで保持　※他の仕組みで既に高速化済
- DBIM関連のパフォーマンス統計

Appendix

オブジェクト毎のインメモリ定義の確認

- USER_TABLES: オブジェクト毎のインメモリ定義

```
column table_name format a20
column PRIORITY format a15
column DISTRIBUTE format a15
column COMPRESSION format a20

Select table_name,
       inmemory_priority priority,
       inmemory_distribute distribute,
       inmemory_compression compression
From user_tables;
```

	ポピュレート優先度	ノード間分散方式	圧縮モード
TABLE_NAME	PRIORITY	DISTRIBUTE	COMPRESSION
LINEORDER	CRITICAL	AUTO	FOR QUERY LOW
PART	NONE	AUTO	FOR QUERY LOW
CUSTOMER	NONE	AUTO	FOR QUERY LOW
SUPPLIER	NONE	AUTO	FOR QUERY LOW
DATE_DIM	NONE	AUTO	FOR QUERY LOW

オブジェクト毎のポピュレーションの状態確認

- V$IM_SEGMENTS: オブジェクト毎のポピュレーションの状態

```
column name format a20
column owner format a10
column status format a15
column No_Pop_Bytes format 999,999,999,999

SELECT v.owner, v.segment_name name, v.populate_status status,
inmemory_size/1024/1024 IM_MB, v.bytes_not_populated No_Pop_Bytes
FROM   v$im_segments v;
```

ポピュレート未完了のオブジェクト

インメモリ内サイズ　　未完了のバイト数

OWNER	NAME	STATUS	IM_MB	NO_POP_BYTES
SSB	SUPPLIER	COMPLETED	16.125	0
SSB	LINEORDER	STARTED	23975.75	15,753,846,784
SSB	CUSTOMER	COMPLETED	240.4375	0
SSB	DATE_DIM	COMPLETED	1.125	0
SSB	PART	COMPLETED	34.25	0

オブジェクト毎のインメモリ内のサイズと圧縮率の確認

- V$IM_SEGMENTS: オブジェクト毎のインメモリサイズと圧縮率

```
column name format a20

Select v.segment_name              name,
       v.bytes/1024/1024           orig_MB,
       v.inmemory_size/1024/1024 in_mem_MB,
       v.bytes/v.inmemory_size     comp_ratio
From   v$im_segments v
where v.owner = 'SSB'
Order by 4;
```

元サイズ　インメモリサイズ　　圧縮率

```
NAME                    ORIG_MB  IN_MEM_MB COMP_RATIO
-------------------- ---------- ---------- ----------
CUSTOMER                    256   240.4375 1.06472576
PART                        128      34.25 3.73722628
SUPPLIER                     64     16.125 3.96899225
DATE_DIM                     64      1.125 56.8888889
```

DBIMのチューニング

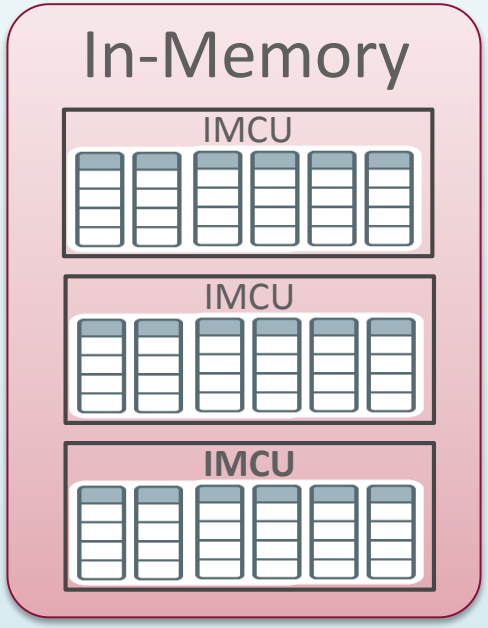
1. 基本的にほとんどチューニング項目はない
   - ポピュレート完了の状態、パフォーマンス統計などの確認等
2. パラレル度の指定とパーティションの活用方法
   - CPUのネックにならないように検索のパラレル度を調整する
   - 表サイズが大きい場合はパーティションを効果的に利用し
     インメモリスキャンのスキャンサイズを小さくする
3. Attribute Clusterも限定的に利用検討する
   - 検索条件頻度の高いカラムには「Attribute Cluster」の機能の適用を検討する
     - インメモリ・ストレージ索引の効果を高める可能性大
   - Attribute Clusterの仕様を理解したうえで利用する

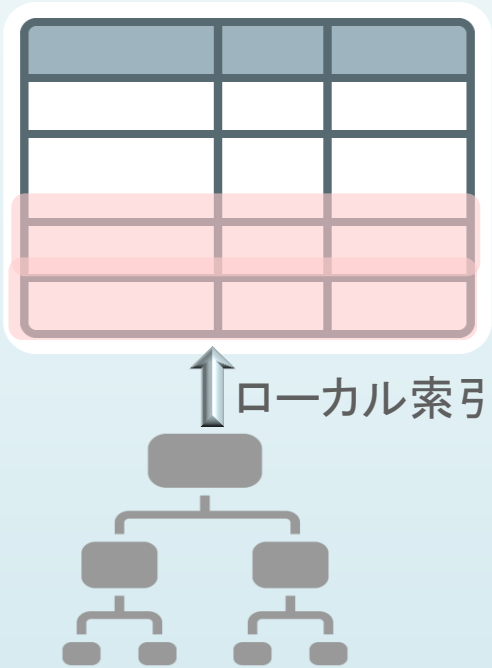
DBIMでもパーティション表は効果的

リストパーティション（ SALES表）

P1: ORDER_STATUS='OPEN'

P2: ORDER_STATUS='CLOSED'





P1: インメモリスキャンが最適

P2: 索引スキャンが最適

パーティション単位に
最適なSQL実行計画を
選択可能

→ 最適なパフォーマンス

DBIMでもパーティション表は効果的

SQL実行計画例

```
SELECT * FROM SALES where ORDER_DATE between‘2013-01-01’ and ‘2014-01-01’;
```

```
-------------------------------------------------------------------------
| Id  | Operation                          | Name        | Pstart| Pstop |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |             |       |       |
|   1 |  SORT AGGREGATE                    |             |       |       |
|   2 |   VIEW                             | VW_TE_2     |       |       |
|   3 |    UNION-ALL                       |             |       |       |
|   4 |     PARTITION RANGE SINGLE         |             |     1 |     1 |
|*  5 |      TABLE ACCESS INMEMORY FULL    | SALES       |     1 |     1 |
|   6 |     PARTITION RANGE SINGLE         |             |     2 |     2 |
|*  7 |      INDEX RANGE SCAN              | IDX_SALES   |     2 |     2 |
-------------------------------------------------------------------------
   5 - inmemory("ORDER_DATE“>=TO_DATE(' 2013-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDER_DATE"<=TO_DATE(' 2014-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
       filter("ORDER_DATE“>=TO_DATE(' 2013-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDER_DATE"<=TO_DATE(' 2014-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   7 - access("ORDER_DATE“>=TO_DATE(' 2013-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDER_DATE"<=TO_DATE(' 2014-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

実行したインメモリ検索の速度が妥当か？

物理I/Oアクセスがないかをチェック

- 対応が必要なもの
  - ポピュレートが未完了
  - PGA領域不足（大量ソート、ジョイン） ※通常のデータベース・チューニング
  - Oracle RAC構成の場合でインターノード・パラレルクエリ
    - パラレル度ポリシーの設定（AutoDOP）: PARALLEL_DEGREE_POLICY = AUTO
- 対応が特に不要なもの
  - ハード・パースによるアクセス（2回目以降なくなる）
  - Adaptive Plan、Dynamic Samplingによるアクセス（2，3回目以降なくなる）

ポピュレートが未完了

POPULATE_STATUSが「STARTED」で、BYTES_NOT_POPULATED列に値が表示される場合は未完了

オブジェクトのポピュレートがまだ完了していない場合

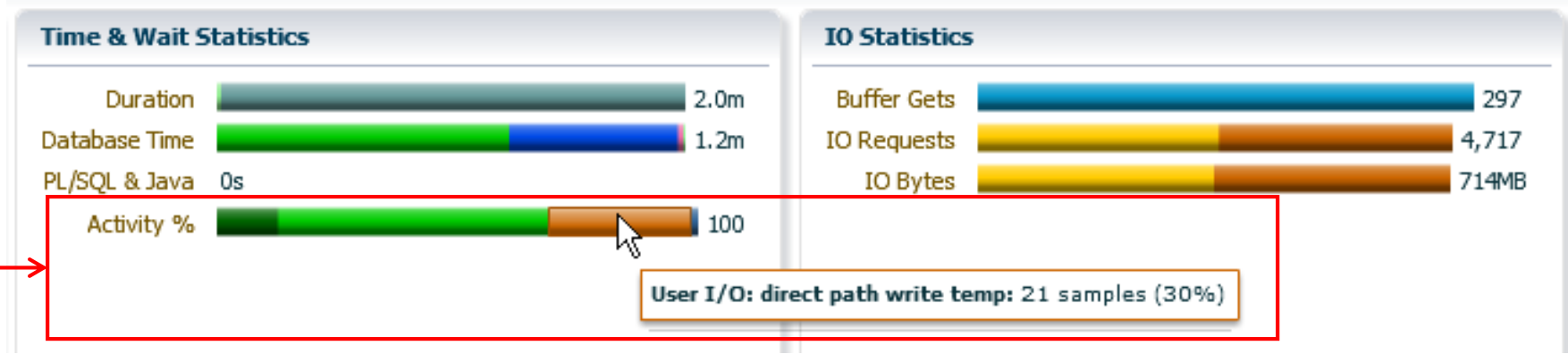
```
SQL> select segment_name, populate_status, inmemory_priority, inmemory_size,
bytes_not_populated from v$im_segments;

SEGMENT_NAME  POPULATE_STATUS  INMEMORY_PRIORITY  INMEMORY_SIZE  BYTES_NOT_POPULATED
------------  ---------------  -----------------  -------------  -------------------
ACCOUNTS      STARTED          HIGH               196606         2434886912
SALES         COMPLETED        CRITICAL           135790592      0
```

```
SQL> select pool, alloc_bytes, used_bytes, populate_status from v$inmemory_area;

POOL              ALLOC_BYTES  USED_BYTES  POPULATE_STATUS
----------------  -----------  ----------  ----------------
1MB POOL           8186232832  8186232832  OUT OF MEMORY
64KB POOL          2063597568    26345472  DONE
```

インメモリ・カラム・ストアの領域不足の場合に「OUT OF MEMORY」が表示される

PGA領域が不足している（大量ソート、ジョインなど）

SQL Monitorより



待機イベント・サマリで
TEMP領域の書込発生

Operation	Name	Lin...	IO Bytes	IO Req...	Activity %
SELECT STATEMENT		0			15
PX COORDINATOR		1			1.41
PX SEND QC (ORDER)	:TQ10001	2			4.23
SORT ORDER BY		3	368MB	1,765	37
PX RECEIVE		4			
PX SEND RANGE	:TQ10000	5			5.63
HASH JOIN		6	346MB	2,952	23
JOIN FILTER CREATE	:BF0000	7			
TABLE ACCESS INMEMORY FULL	CUSTOMER	8			
JOIN FILTER USE	:BF0000	9			1.41
PX BLOCK ITERATOR		10			
TABLE ACCESS INMEMORY FULL	LINEORDER	11			13

ジョインとソート処理で
ディスクI/Oあり

インメモリ・スキャンはCPUのみ

Attribute Clustering

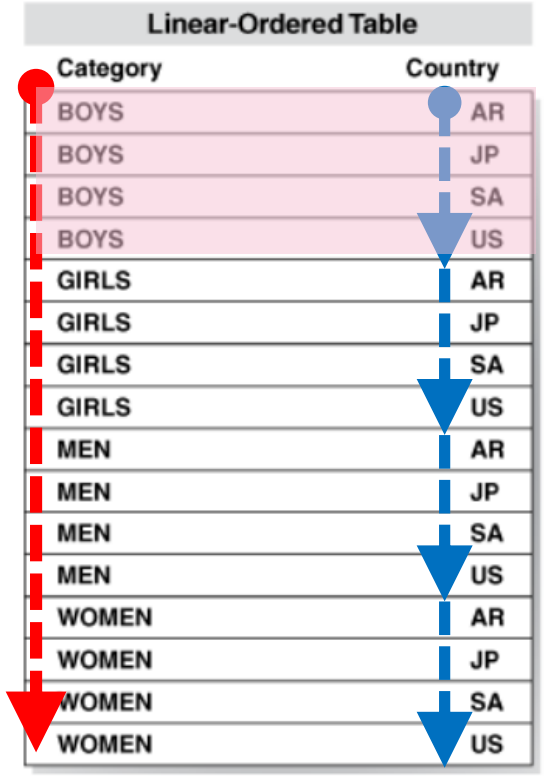
ユーザーの指定に基づきデータ属性より隣接した領域に格納する構成表

Attribute Clustering

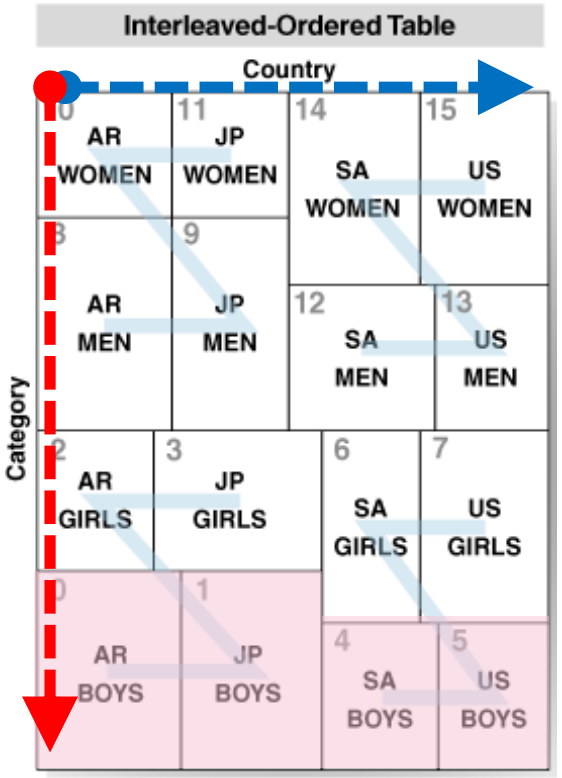
データ格納方式の改善による読込みコストの削減

- 概要
  - データを整えつつ物理的に格納
  - 物理的に近接したエリアに格納される
  - ユーザー定義による構成
- クラスタ・タイプ
  - Clustering by **Linear order**
    - 単一表での使用を推奨
  - Clustering by **Interleaved order**
    - 結合表での使用を推奨

Linear order



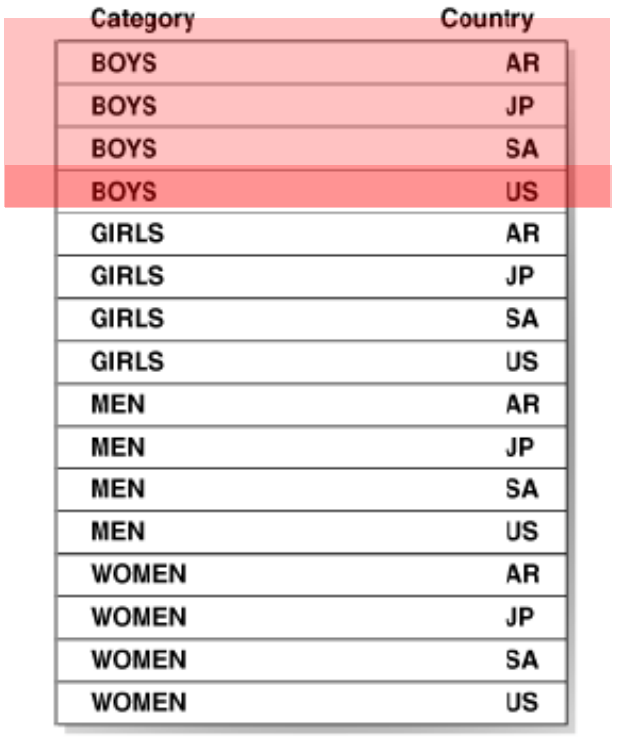
Interleaved order



Attribute Clustering

Example

- CLUSTERING BY LINEAR ORDER (category, country)



Category	Country
BOYS	AR
BOYS	JP
BOYS	SA
BOYS	US
GIRLS	AR
GIRLS	JP
GIRLS	SA
GIRLS	US
MEN	AR
MEN	JP
MEN	SA
MEN	US
WOMEN	AR
WOMEN	JP
WOMEN	SA
WOMEN	US

LINEAR ORDER

Pruning with:

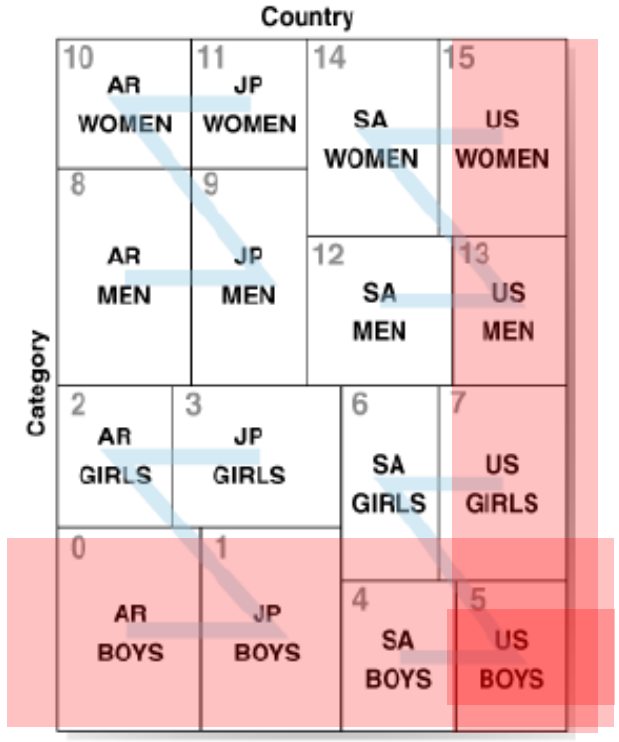
```
SELECT   ..
FROM     table
WHERE    category = 'BOYS';
```

```
SELECT   ..
FROM     table
WHERE    category = 'BOYS';
AND      country  = 'US'
```

Attribute Clustering

Example

- CLUSTERING BY INTERLEAVED ORDER (category, country)



INTERLEAVED ORDER

Pruning with:

```
SELECT  ..
FROM    table
WHERE   category = 'BOYS';

SELECT  ..
FROM    table
WHERE   country  = 'US'

SELECT  ..
FROM    table
WHERE   category = 'BOYS';
AND     country  = 'US'
```

Attribute Clustering

データを整えた状態で格納すること

CSEQ	C25	C100	C1K	C10K	
524	3	49	145	7964	
525	11	43	375	8597	
526	3	30	334	1954	
527	16	19	512	1641	
528	8	18	399	1720	
529	12	15	106	271	
530	14	43	991	8529	
531	10	88	511	2758	
532	19	55	496	7856	
533	17	23	864	7424	
534	19	68	395	3222	
535	22	3	686	4187	
536	20	96	425	6527	
537	10	31	780	7422	
538	3	56	946	4081	
539	20	34	45	4343	

Attribute Clustering対象カラム(C25,C100)

CSEQ	C25	C100	C1K	C10K	
8868	1	1	845	8183	
6574	1	1	826	6312	
7244	1	1	70	6631	
6295	1	1	973	1451	
2446	1	1	979	618	
2719	1	1	175	3150	
2819	1	1	54	6484	
7720	1	2	754	1486	
7773	1	2	1000	3907	
2935	1	2	859	7023	
2620	1	2	369	7428	
3575	1	2	371	5169	
6843	1	3	174	5763	
6475	1	3	884	8840	
2050	1	3	757	6988	
7721	1	4	505	7544	

Attribute Clustering

データを整えた状態で格納すること

CSEQ	C25	C100	C1K	C10K
8868	1	1	845	8183
6574	1	1	826	6312
7244	1	1	70	6631
6295	1	1	973	1451
2446	1	1	979	618
2719	1	1	175	3150
2819	1	1	54	6484
7720	1	2	754	1486
7773	1	2	1000	3907
2935	1	2	859	7023
2620	1	2	369	7428
3575	1	2	371	5169
6843	1	3	174	5763
6475	1	3	884	8840
2050	1	3	757	6988
7721	1	4	505	7544

データを整えるメリット

- ExadataのStorage Index
- Database In-Memoryインメモリストレージ索引
- Zone Maps (Exadata or SuperCluster)

上記機能への効果

Attribute Clustering / Syntax

–Linear

```
CREATE TABLE sales (
        category VARCHAR2(20),
        country  VARCHAR2(2),
        amount_sold  NUMBER(10,2)
        )
CLUSTERING BY LINEARORDER (
        category,
        country
        )
        YES ON LOAD
        YES ON DATA
        MOVEMENT;
```

–Interleaved

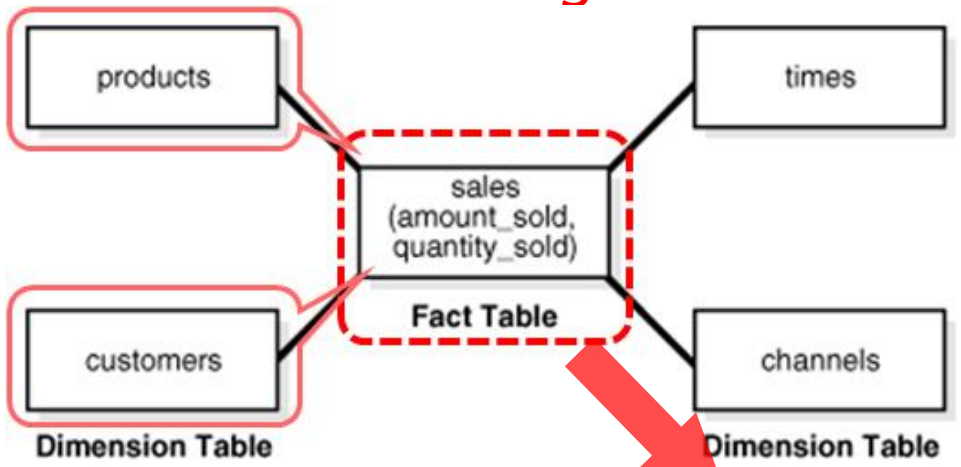
```
CREATE TABLE sales (
        prod_id  NOT NULL NUMBER,
        cust_id     NOT NULL  NUMBER,
        amount_sold NUMBER(10,2) …)
    CLUSTERING  SALES
        JOIN products ON
        (sales.prod_id = products.prod_id)
        JOIN customers ON
        (sales.cust_id = customers.cust_id)
    BY INTERLEAVED ORDER(
        (products.prod_category,
        products.prod_subcategory),
        (customers.country_id,
        customers.cust_state_province,
        customers.cust_city)
```

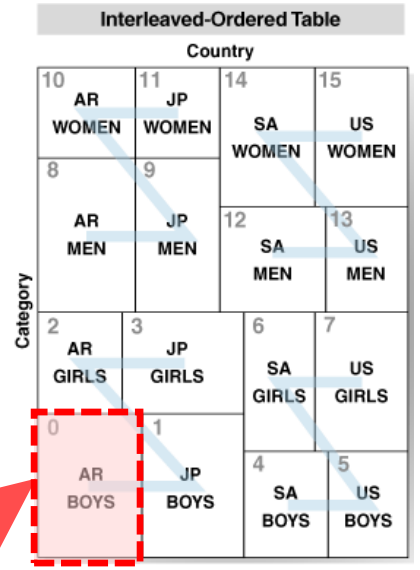
Attribute Clustering

Guidelines for Using Attribute Clustering

スター・スキーマなどの環境においては、頻繁に使用される2から3のディメンジョン表から検討する

Attribute Clustering





Sales					
products		customers			amount_sold
category	subcategory	country_id	cust_state_province	cust_city	

Attribute Clustering

Guidelines for Using Attribute Clustering

- データ属性の組み合わせによるI/Oコスト削減を目的とする
- 表サイズが大きい場合、カーディナリティの低いものを優先に構成する
- 頻繁に検索されるファクト表を中心に選択し構成する
- Linear Orderでは 接頭語・接尾語のあるリストカラムが優位
- 結合表などを検討し、カーディナリティの低いものを優先に構成する
- 4つ以上のディメンジョン表が存在する場合、頻繁に照会される2から3の表を選択し、クラスタリングを検討する
- 索引の代わりにカーディナリティの低いカラムを選択し、Attribute Clusteringを活用する
- ディメンジョン表の主キーは階層的なもの組み合わせではない外部キーが望ましい

  例： 年・四半期・月・日 などの組み合わせなど

Attribute Clustering

データの移行

- Direct-path insert operations

  ダイレクトロード時にデータのクラスタリングが行われる

  SQL> INSERT /*+ APPEND */ INTO sales SELECT * FROM sales_org ;

- Data movement operations

  以下のような表のmove操作に加え、オンライン表再定義を含むパーティション移動、マージ、分割、および結合によりデータのクラスタリングが行われる

  SQL> alter table sales move;

Attribute Clustering

Database In-Memory / IMCU と Attribute Clustering

	CSEQ	C25	C100	C1K	C10K
IMCU1	9897	1	7	999	6715
	8895	1	7	501	9786
	3419	1	7	73	3309
	466	1	7	935	7627
	7905	1	8	91	506
	1724	1	8	527	7966
	2389	1	8	13	9095
	2848	1	8	363	620
IMCU2	8793	1	9	797	7120
	6173	1	9	129	7411
	7496	1	9	328	7657
	8299	1	9	603	8827
	6049	1	9	720	4738
	3018	1	9	465	4999
	3389	1	9	766	4201
	9717	1	10	364	6601

C25 Min : 1 / Max : 1
C100 Min : 7 / Max : 8

C1K Min : 13 / Max : 999

C25 Min : 1 / Max : 1
C100 Min : 9 / Max : 10

C1K Min : 129 / Max : 797

Hardware and Software
Engineered to Work Together

VISION 2020

#1 CLOUD

ORACLE JAPAN


ORACLE®