

ORACLE®



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。

OracleとJavaは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。

# プロフィール

- Oracle Exadata リリース当初から、お客様のSQLやデータを使用したPoC (Proof of Concept) を実施して7年
  - 本番稼働しているたくさんのシステムのパフォーマンス・チューニングを経験
- 2010年には米オラクルの開発部門の一員としてサンフランシスコのヘッド・クォーターで勤務
  - 米国のお客様のPoCを実施しつつ、そこから見えてきたOracle Database のパフォーマンス課題の解決に取り組む

# 動かないデータウェアハウス

## 性能を10倍遅くさせる理由

- ハードウェア購入プロセスの間違い
- OLTPの経験を基にしたシリアル実行手法
- 索引やデータマートによるデータベースの肥大化
- 遅すぎるETLやローディング
- 無制御なアドホック・クエリー
- CPUリソースが取得できないほど高すぎる同時実行数

# アジェンダ

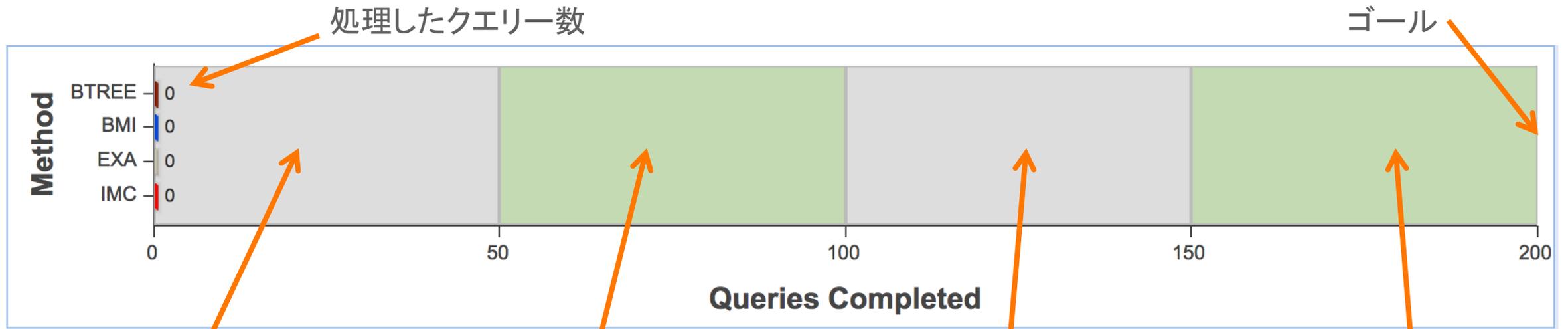
- 1 パフォーマンス・レース！
- 2 データウェアハウス・デザインのルールとフレームワーク
- 3 実行方式を選ぼう
- 4 複数ユーザー・パフォーマンス・レース！
- 5 プラットフォームを選ぼう
- 6 ハイパフォーマンスを実現するために一番大事なこと

# パフォーマンス・レース！

- スター・スキーマで構成されたデータ・モデル
- 4人のレーサー:  
Bツリー索引 / ビットマップ索引  
Exadata表スキャン / In-Memory
- 200個の異なるクエリーは  
序盤は抽出行数が少なく  
終盤になるにつれて多くなり  
それぞれのレーサーが同じ順序で実行



# パフォーマンス・レース！



ファクトから  
1~10行  
抽出するクエリー

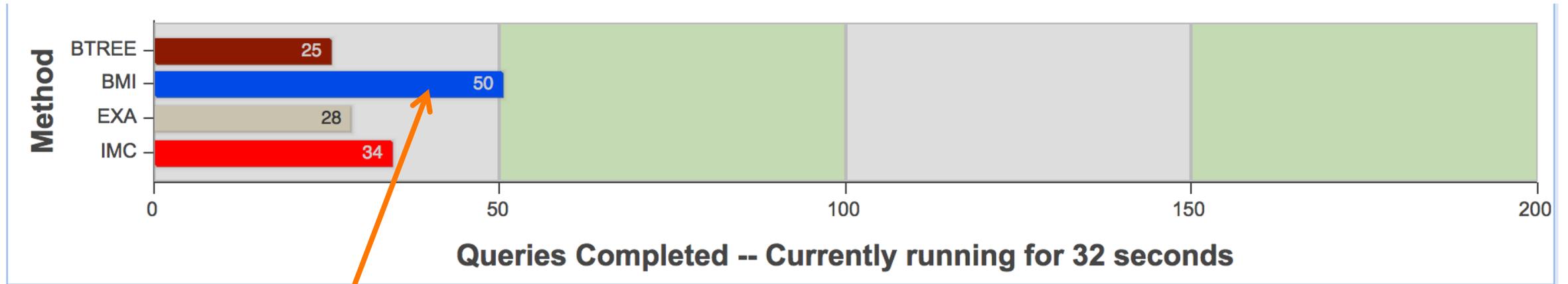
ファクトから  
10~1,000行  
抽出するクエリー

ファクトから  
1,000~100,000行  
抽出するクエリー

ファクトから  
100,000行以上  
抽出するクエリー

# パフォーマンス・レース！

## 10行抽出の結果

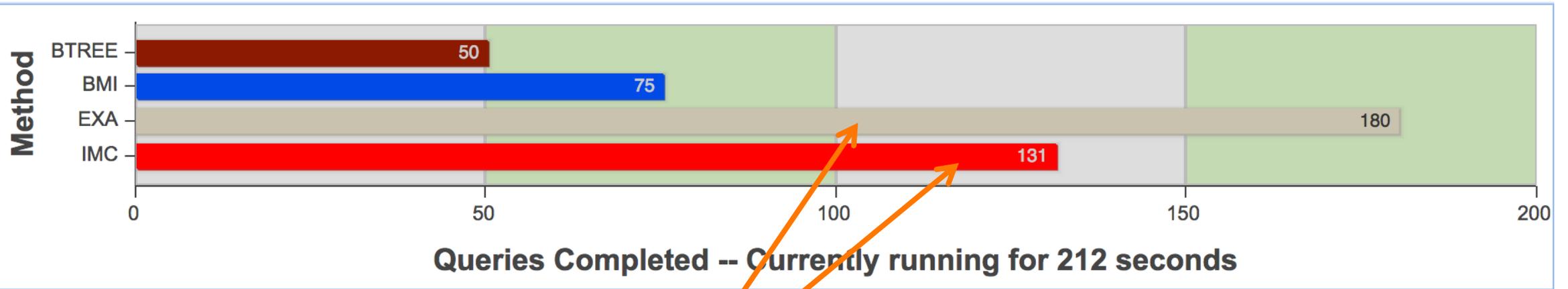


最初に50個のクエリーを処理し終えたのはビットマップ索引。  
抽出行が少ない場合はビットマップ索引が良い

- Bツリー索引 
- ビットマップ索引 
- Exadata表スキャン 
- In-Memory 

# パフォーマンス・レース！

## 100万行抽出の結果



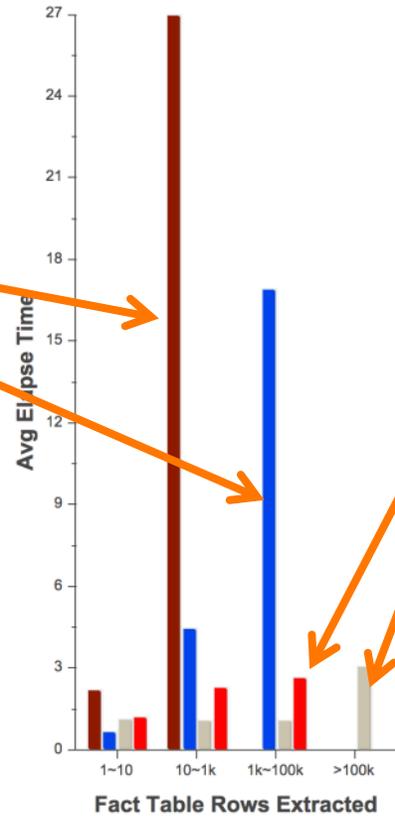
多くの行が返ってくるようになると  
表スキャン方式が逆転

- Bツリー索引
- ビットマップ索引
- Exadata表スキャン
- In-Memory

# パフォーマンス・レース！

## 平均レスポンス時間

抽出行が多くなるほど  
索引は遅くなる

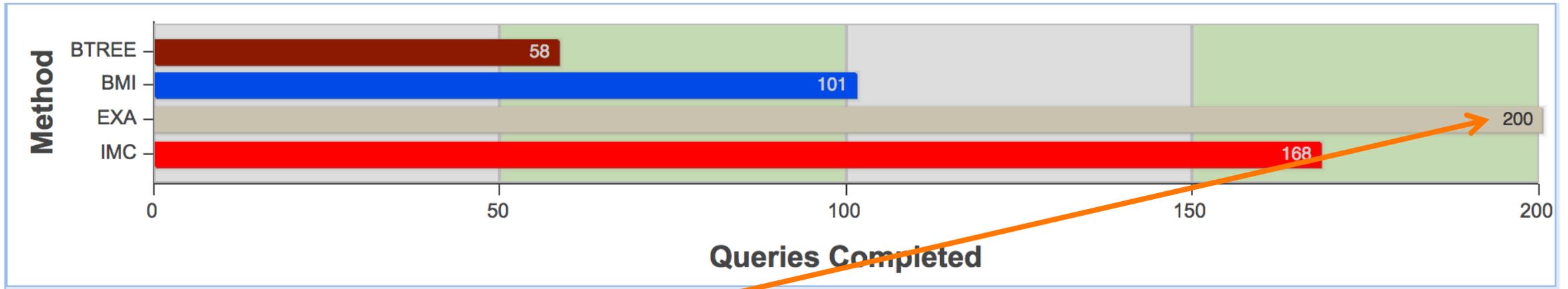


抽出行が増えても  
表スキャンは結果時間が安定している

- Bツリー索引
- ビットマップ索引
- Exadata表スキャン
- In-Memory

# パフォーマンス・レース！

## シリアル実行の結果



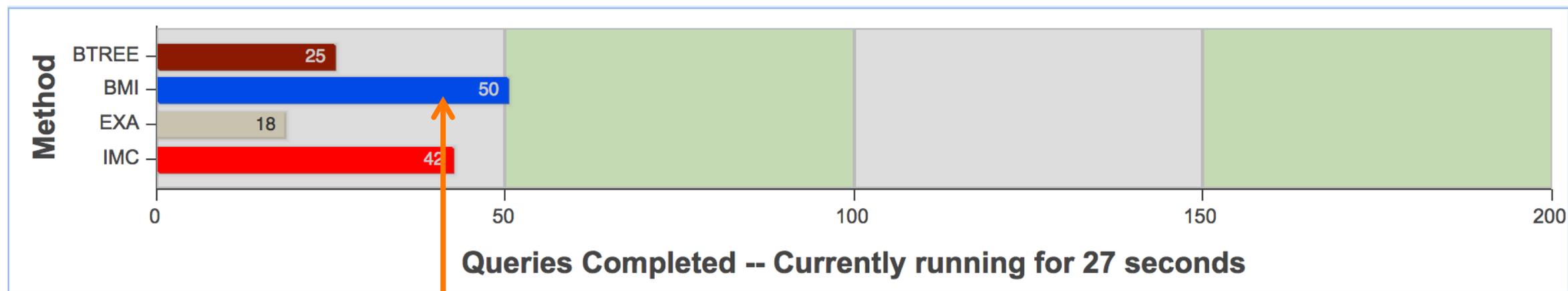
Exadataの勝ち！

シリアル実行では  
Storage Server のCPUも使えるExadataのほうが  
CPU **合計数** でIn-Memoryより有利

- Bツリー索引 
- ビットマップ索引 
- Exadata表スキャン 
- In-Memory 

# パフォーマンス・レース！

パラレル実行 (DoP=4)



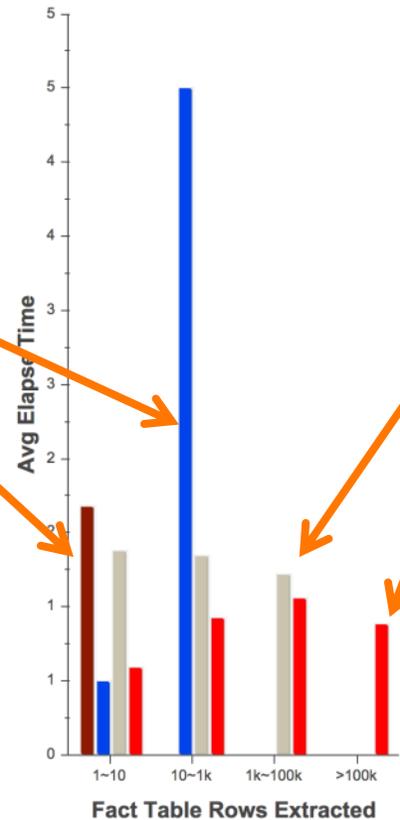
抽出行が少ない場合は引きつづきビットマップ索引が良い。  
ただし差は縮まった

Bツリー索引   
ビットマップ索引   
Exadata表スキャン   
In-Memory 

# パフォーマンス・レース！

## パラレル実行時の平均レスポンス時間

シリアル実行同様  
抽出行が多くなるほど  
索引は遅くなる

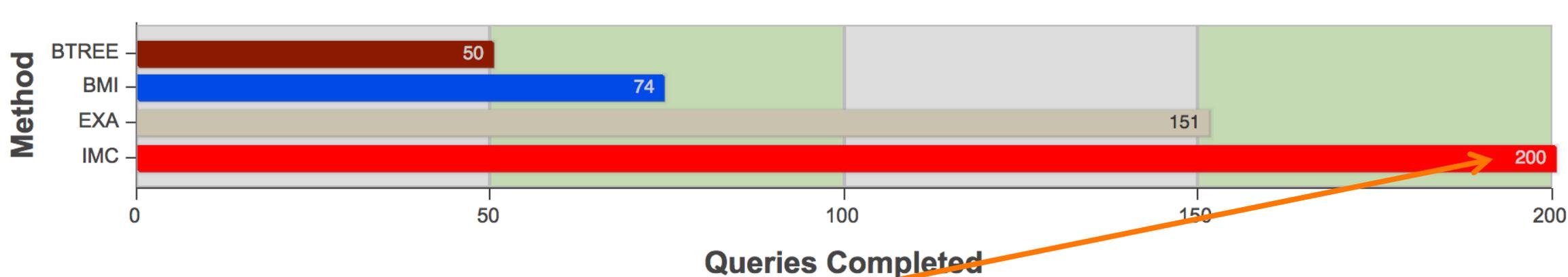


抽出行が増えても  
表スキャンは結果時間が安定している

- Bツリー索引
- ビットマップ索引
- Exadata表スキャン
- In-Memory

# パフォーマンス・レース！

## パラレル実行の結果



In-Memoryの勝ち！

ただし、Exadata表スキャンとIn-Memoryの差は大きくない。  
この2者は異なる方法で同じ問題を解決している



# 表スキャン時の高速手法

スキャン



ハードウェア:  
CPU、ディスク本数、  
PCIeフラッシュ、InfiniBand

ソフトウェア:  
Smart Scan、HCC、ストレージ索引

フィルタリング



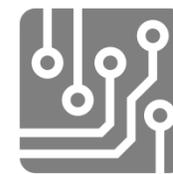
Storage Server での  
ブルーム・フィルタ実行



Exadata

In-Memory Columnar Layout  
SIMDベクトル演算

列ストアでの  
ブルーム・フィルタ実行



In-Memory

# Database In-Memory が効くところ

Operation	Name	Lin...	Estimated R...	Cost	Timeline(24s)	Executi...	Actual Rows	Memory (...)	Temp (Max)	O...	IO Reque...	IO B...	Activity %
SELECT STATEMENT		0				5	12						
PX COORDINATOR		1				5	12						
PX SEND QC (RANDOM)	:TQ10002	2	9	646K		2	12						
HASH GROUP BY		3	9	646K		2	12	2MB					
PX RECEIVE		4	9	646K		2	24						
PX SEND HASH	:TQ10001	5	9	646K		2	24						
HASH GROUP BY		6	9	646K		2	24	3MB					10
HASH JOIN		7	8,994K	646K		2	9,148K	4MB					2.08
JOIN FILTER CREATE	:BF0000	8	20K	233		2	40K						
TABLE ACCESS FULL	SUPPLIER	9	20K	233		2	40K						
HASH JOIN		10	44M	646K		2	9,841K	2MB					8.33
JOIN FILTER CREATE	:BF0001	11	366	21		2	732						
PX RECEIVE		12	366	21		2	732						
PX SEND BROADCAST	:TQ10000	13	366	21		2	732						
PX SELECTOR		14				2	366						
TABLE ACCESS FULL	DATE_DIM	15	366	21		2	366						
JOIN FILTER USE	:BF0000	16	300M	646K		2	9,841K						
JOIN FILTER USE	:BF0001	17	300M	646K		2	9,841K						
PX BLOCK ITERATOR		18	300M	646K		2	9,841K						
TABLE ACCESS FULL	LINEORDER	19	300M	646K		35	9,841K				33K	33GB	79



# Database In-Memory が効かないところ

Operation	Name	Lin...	Estimated R...	Cost	Timeline(24s)	Executi...	Actual Rows	Memory (...)	Temp (Max)	O...	IO Reque...	IO B...	Activity %
SELECT STATEMENT		0				5	12						
PX COORDINATOR		1				5	12						
PX SEND QC (RANDOM)	:TQ10002	2	9	646K		2	12						
HASH GROUP BY		3	9	646K		2	12	2MB					
PX RECEIVE		4	9	646K		2	24						
PX SEND HASH	:TQ10001	5	9	646K		2	24						
HASH GROUP BY		6	9	646K		2	24	3MB					10
HASH JOIN		7	8,994K	646K		2	9,148K	4MB					2.08
JOIN FILTER CREATE	:BF0000	8	20K	233		2	40K						
TABLE ACCESS FULL	SUPPLIER	9	20K	233		2	40K						
HASH JOIN		10	44M	646K		2	9,841K	2MB					8.33
JOIN FILTER CREATE	:BF0001	11	366	21		2	732						
PX RECEIVE		12	366	21		2	732						
PX SEND BROADCAST	:TQ10000	13	366	21		2	732						
PX SELECTOR		14				2	366						
TABLE ACCESS FULL	DATE_DIM	15	366	21		2	366						
JOIN FILTER USE	:BF0000	16	300M	646K		2	9,841K						
JOIN FILTER USE	:BF0001	17	300M	646K		2	9,841K						
PX BLOCK ITERATOR		18	300M	646K		2	9,841K						
TABLE ACCESS FULL	LINEORDER	19	300M	646K		35	9,841K				33K	33GB	79



# Database In-Memory が効かないところ

Operation	Name	Lin...	Estimated ...	Cost	Timeline(589s)	Execu...	Actual R...	Memory ...	Temp (...	O...	IO Requ...	IO ...	Activity %
[-] NESTED LOOPS		12	8,994K	925K		1	9,148K						.17
[-] BITMAP CONVERSION TO ROWIDS		13	8,994K	10K		1	9,148K						.17
[-] BITMAP AND		14				1	569						
[-] BITMAP MERGE		15				1	768	51MB					.52
[-] BITMAP KEY ITERATION		16				1	15K						
[-] VIEW	index\$_join\$_079	17	366	8		1	366						
[-] HASH JOIN		18				1	366	2MB					
[-] BITMAP CONVERSION TO R...		19	366	1		1	366						
[-] BITMAP INDEX SINGLE VAL...	DATE_DIM_YEAR_BMI3	20				1	1						
[-] INDEX FAST FULL SCAN	DATE_DIM_DATEKEY_U1	21	366	9		1	2,556						
[-] BITMAP INDEX RANGE SCAN	LINEORDER_DATE_BMI1	22				366	15K						
[-] BITMAP MERGE		23				1	1,381	239MB					4.33
[-] BITMAP KEY ITERATION		24				1	80K						
[-] VIEW	index\$_join\$_082	25	20K	213		1	20K						
[-] HASH JOIN		26				1	20K	2MB					
[-] BITMAP CONVERSION TO R...		27	20K	3		1	20K						
[-] BITMAP INDEX SINGLE VAL...	SUPPLIER_REGION_BMI3	28				1	5						
[-] INDEX FAST FULL SCAN	SUPPLIER_U1	29	20K	262		1	100K						
[-] BITMAP INDEX RANGE SCAN	LINEORDER_SUPPLIER_BMI2	30				20K	80K						
[-] TABLE ACCESS BY USER ROWID	LINEORDER	31	1	915K		16M	9,148K				2,592K	25GB	93

# パフォーマンス・レースで学んだこと

- 索引は抽出行が多くなればなるほど性能が悪化する
- 表スキャンは抽出行が増えても性能が安定している
- In-MemoryはExadataに比べて必ずしも圧倒的に速いわけではない

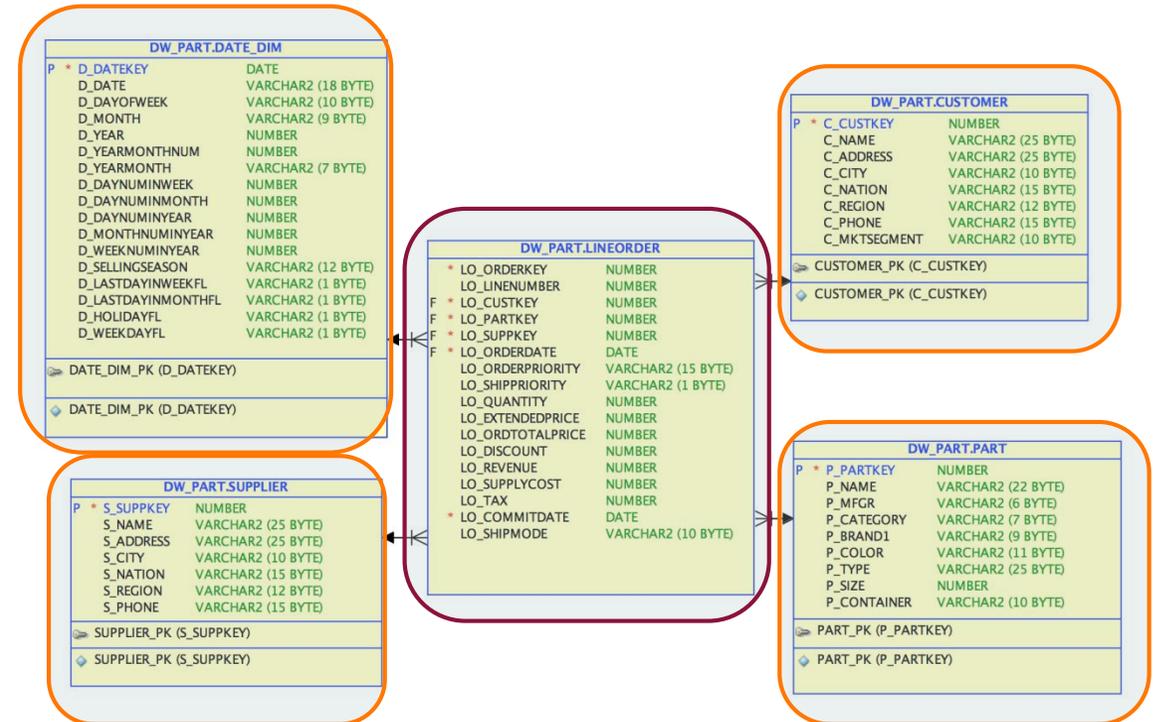


# アジェンダ

- 1 パフォーマンス・レース！
- 2 データウェアハウス・デザインのルールとフレームワーク
- 3 実行方式を選ぼう
- 4 複数ユーザー・パフォーマンス・レース！
- 5 プラットフォームを選ぼう
- 6 ハイパフォーマンスを実現するために一番大事なこと

# スター・スキーマとは

- ファクト表にはキーとメジャー(度量)が保存されている。第3正規形まで実施  
例: 注文数、小計
- ディメンション表にはファクトを表現するための属性が保存されている。第2正規形まで実施  
例: 顧客名、部品カテゴリー
- スター・クエリは  
スター・スキーマ用のクエリ



# 典型的なスター・クエリーの構造

```
SELECT      d_sellingseason, p_category, s_region,
            SUM(lo_extendedprice)
FROM        lineorder
            JOIN customer      ON lo_custkey = c_custkey
            JOIN date_dim     ON lo_orderdate = d_datekey
            JOIN part         ON lo_partkey = p_partkey
            JOIN supplier     ON lo_suppkey = s_suppkey
WHERE       d_year           IN (1993, 1994, 1995)
            AND p_container  IN ('JUMBO PACK')
GROUP BY   d_sellingseason, p_category, s_region
ORDER BY   d_sellingseason, p_category, s_region
```

- **ファクト**表を選択
- すべてのディメンション表を**結合**して、スターを作成
- 属性の**フィルタ**を選択
- 集合演算する**メジャー**を選択
- 付随する**項目**を選択
- **グルーピング単位**を選択
- **ソート順序**を選択

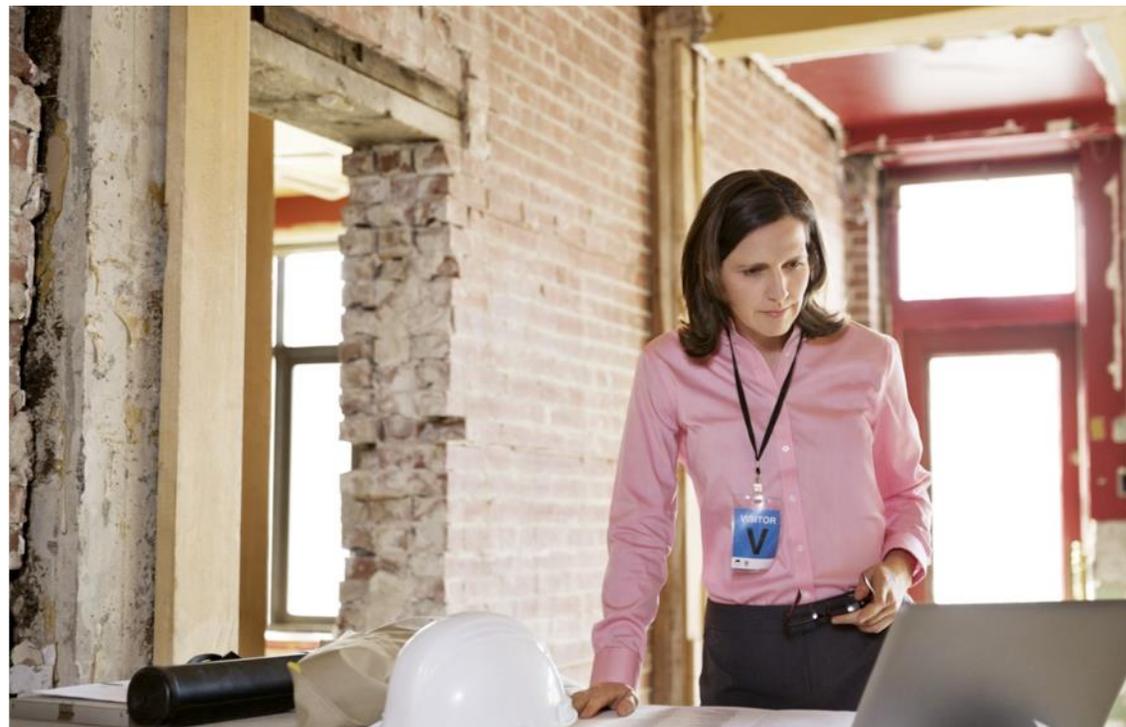
# ルールとフレームワーク

- 制約
- データ型
- データ加工
- 統計情報



# 制約

- 結合キーには NOT NULL 制約をつける
- ディメンション表の結合キーには主キー制約 (PK) をつける
- ファクト表の結合キーには参照整合性制約 (FK) をつける



# NOT NULL 制約

- lineorder表の各行は customer表から何行戻す？
- NOT NULL 制約がない状態で lo\_custkey列がNULLだと？
- lo\_custkey列が NOT NULL のとき、customer表の何行と結合する？
- NOT NULL 制約はコストがかからない

```
FROM      lineorder
JOIN      customer ON lo_custkey = c_custkey
```

```
SQL> DESC lineorder
Name          Null?         Type
-----
LO_CUSTKEY    NOT NULL     NUMBER
...
```

```
SQL> DESC customer
Name          Null?         Type
-----
C_CUSTKEY     NOT NULL     NUMBER
...
```

# PK制約とFK制約

- デイメンション表にはPK制約が必要
- ファクト表にはFK制約が必要
- 制約の検証のための索引を削除
- 制約の品質はETL/ELT中に保証する
- 制約にRELY句をつける
- オプティマイザにRELY付きの制約を信頼させる

```
ALTER TABLE customer  
ADD CONSTRAINT customer_pk  
PRIMARY KEY (c_custkey) RELY;
```

```
ALTER TABLE lineorder  
ADD CONSTRAINT lo_customer_pk  
FOREIGN KEY (lo_custkey)  
REFERENCES customer (c_custkey)  
RELY DISABLE NOVALIDATE;
```

```
ALTER SYSTEM  
SET QUERY_REWRITE_INTEGRITY = TRUSTED;
```

PK/FK制約がある状況では、ファクト表の1行は  
デイメンション表から必ず1行だけを返す

# ETL/ELT時のFK制約違反検索

- 制約が無効化された状態で  
どうやってデータを検証する？
- 制約がRELYモードのとき、ファクト表に  
INSERTされたデータをどう信頼する？
- 右のSQLは、customer表に  
存在しないlo\_custkey列の値を  
lineorder表から検索する。  
つまり、FK制約違反を見つける

```
SELECT *  
FROM   lineorder  
       LEFT OUTER JOIN customer  
         ON lo_custkey = c_custkey  
WHERE  c_custkey IS NULL;
```

# ETL/ELT時のFK制約違反検索

- 複数ディメンション表に対するlineorder表の同時検証も可能
- 各ディメンション表に存在しないキー列の値をlineorder表から検索する

```
SELECT *
FROM   lineorder
      LEFT OUTER JOIN customer
        ON lo_custkey = c_custkey
      LEFT OUTER JOIN date_dim
        ON lo_orderdate = d_datekey
      LEFT OUTER JOIN part
        ON lo_partkey = p_partkey
      LEFT OUTER JOIN supplier
        ON lo_suppkey = s_suppkey
WHERE  c_custkey IS NULL
      OR d_datekey IS NULL
      OR p_partkey IS NULL
      OR s_suppkey IS NULL;
```

# ETL/ELT時のPK制約違反検索

## 単純なPK制約違反 確認方法

```
SELECT  pk, COUNT(*)
FROM    dirty_data
GROUP BY pk
HAVING  COUNT(*) > 1;
```

## 重複データのROWIDの 1個を取得方法

```
SELECT  pk, COUNT(*),
        MAX(ROWID)
FROM    dirty_data
GROUP BY pk
HAVING  COUNT(*) > 1;
```

## ロード時に重複データを 排除する方法

```
SELECT column_list
FROM (
  SELECT
    d.*,
    ROW_NUMBER() OVER (
      PARTITION BY pk
      ORDER BY load_time DESC
    ) rowno
  FROM dirty_data d
)
WHERE rowno = 1;
```

# データ型

- PK列とFK列は同じデータ型でなければならない
- PK列とFK列のデータ型の精度も同じでなければならない
- データ型変換コストの削減のため

```
FROM lineorder
  JOIN customer ON lo_custkey = c_custkey
```

```
SQL> DESC lineorder
Name                Null?              Type
-----            -
LO_CUSTKEY          NOT NULL          NUMBER
...

SQL> DESC customer
Name                Null?              Type
-----            -
C_CUSTKEY           NOT NULL          NUMBER(11)
...
```

# データ加工 vs. データ修正

- データ加工

- 新しい表にデータを加工しながらINSERTする
- 安定していて予測可能な性能
- ダイレクト・パス・ロードと高効率な圧縮モードのサポート

- データ修正

- 同じ表内でデータを変更
- UPDATE、DELETE
- すでに存在しているデータ・ブロックの変更によるオーバーヘッド
- 限られた圧縮モード

# データ加工SQL

- ダイレクト・パス・ロードの使用方法
  - APPENDヒントで有効にする
  - CREATE TABLE ... AS SELECT ではデフォルト
- なぜダイレクト・パス・ロードなのか
  - バッファ・キャッシュのバイパス
  - REDO / UNDOの削減
  - ベーシック圧縮、Hybrid Columnar Compression に必須
  - パラレルINSERTに必須

# DELETEの書換え

```
ALTER SESSION ENABLE PARALLEL DML;
```

```
DELETE FROM tx_log  
WHERE symbol = 'JAVA';
```

```
COMMIT;
```

```
ALTER SESSION ENABLE PARALLEL DML;
```

```
INSERT /*+ APPEND */ INTO tx_log_new  
SELECT * FROM tx_log  
WHERE symbol <> 'JAVA';
```

```
COMMIT;
```



```
ALTER TABLE tx_log RENAME TO tx_log_old;  
ALTER TABLE tx_log_new RENAME TO tx_log;
```

または

```
ALTER TABLE tx_log  
EXCHANGE PARTITION part_201409  
WITH TABLE tx_log_new;
```

# UPDATEの書換え

```
ALTER SESSION ENABLE PARALLEL DML;

UPDATE sales_ledger
SET    tax_rate = 8
WHERE tax_rate = 5
      AND sales_date >= DATE '2014-04-01';

COMMIT;
```



```
ALTER SESSION ENABLE PARALLEL DML;

INSERT /*+ APPEND */ INTO
sales_ledger_new
SELECT col1, col2, col3,
      CASE
        WHEN tax_rate = 5
          AND sales_date >= DATE '2014-04-01'
        THEN 8
        ELSE tax_rate
      END tax_rate
FROM sales_ledger;
```

# 統計情報取得方法の基本

- DBMS\_STATSプロシージャを使用。ANALYZEコマンドは使用しない
- AUTO\_SAMPLE\_SIZEを使用
  - より正確なNDV (Number of Distinct Values) が、より高速に取得できる
  - 増分統計収集が可能になる
  - 同時統計収集が可能になる
  - 新しいヒストグラムが使用可能になる

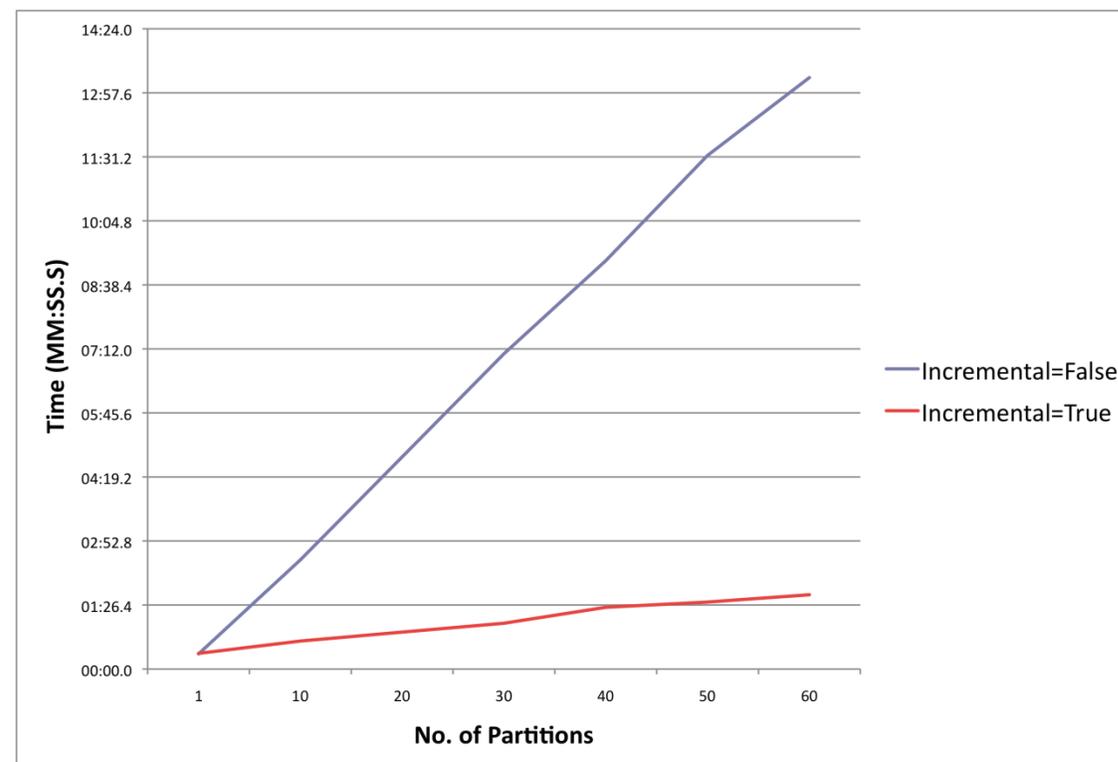
# 増分統計収集

- パーティション表はパーティション・レベルとグローバル・レベルの統計が必要
  - 1個のパーティションにプルーニングされるSQLの場合、パーティション・レベル統計を使用
  - 複数のパーティションにプルーニングされるSQLの場合、グローバル統計を使用
- 増分統計を使用した場合の内部動作
  - 変更されたパーティションのみの統計の収集がされ、シノプシスを作成
  - 作成されたシノプシスをグローバル統計にマージ

# 増分統計収集 vs. 非増分統計収集

- 日次ロードされる60日分の表
- 60パーティション
- 1日当たり100万行

```
BEGIN
  DBMS_STATS.SET_TABLE_PREFS(
    USER, 'table_name',
    'INCREMENTAL', 'TRUE'
  );
END;
/
```



# 同時統計収集

- デフォルトでは、パーティション表の各パーティションはシリアルに収集される
- 同時統計収集では各パーティションを同時に収集する
- パラレル実行するほどにはパーティションが大きくないときに有効

```
BEGIN
  DBMS_STATS.SET_GLOBAL_PREFS(
    'CONCURRENT', 'TRUE'
  );
END;
/
```

```
ALTER SYSTEM SET JOB_QUEUE_PROCESSES = 8;
```

# 拡張統計

- 2種類の拡張統計

- 列グループ統計

- 異なる列同士に相関性がある場合に、その情報をオプティマイザに提供する
      - 都道府県名と市区町村名
      - 自動車メーカーと自動車モデル

- 式統計

- WHERE句に式が使われているときに、式の結果の統計を取得する
      - `UPPER(emp_last_name) = 'SHIBATA'`

# 拡張統計

- イコール条件または IN句のときのみ使われる
- 元となる列のヒストグラムがあり列グループのヒストグラムがない場合、列グループ統計は使われない

```
SELECT
  DBMS_STATS.CREATE_EXTENDED_STATS(
    USER, 'cars', '(maker, model)'
  )
FROM dual;
```

```
SELECT
  DBMS_STATS.CREATE_EXTENDED_STATS(
    USER, 'emp', '(UPPER(emp_last_name))'
  )
FROM dual;
```

# 自動列グループ検出

1. DBMS\_STATS.  
GATHER\_TABLE\_STATS
2. DBMS\_STATS.SEED\_COL\_USAGE  
– 監視時間の設定および監視の開始
3. クエリーの実行
4. DBMS\_STATS.REPORT\_COL\_USAGE  
– 列使用情報レポートの作成
5. DBMS\_STATS.  
CREATE\_EXTENDED\_STATS  
(user, 'table\_name')  
– 3個目の引数のextension不要

```
COLUMN USAGE REPORT FOR SH.CUSTOMERS_TEST
.....
1. COUNTRY_ID : EQ
2. CUST_CITY : EQ
3. CUST_STATE_PROVINCE : EQ
4. (CUST_CITY, CUST_STATE_PROVINCE,
   COUNTRY_ID) : FILTER
5. (CUST_STATE_PROVINCE, COUNTRY_ID) : GROUP_BY
```

```
EXTENSIONS FOR SH.CUSTOMERS_TEST
.....
1. (CUST_CITY, CUST_STATE_PROVINCE, COUNTRY_ID)
   : SYS_STUMZ$C3AIHLPBROI#SKA58H_N created
2. (CUST_STATE_PROVINCE, COUNTRY_ID)
   : SYS_STU#S#WF25Z#QAHIE#MOFFMM_ created
```

# シバタツ流！ 初回の統計取得手順

1. CREATE TABLE
2. 空の表に対して統計を取得
3. DBMS\_STATS.SEED\_COL\_USAGE
4. 空の表に対してクエリー実行
  - ヒストグラムが必要な列の検出
5. 列グループの作成
  1. DBMS\_STATS.REPORT\_COL\_USAGE
  2. DBMS\_STATS.CREATE\_EXTENDED\_STATS
6. 増分統計の有効化
7. データ・ローディング
8. データの統計情報の取得
  - デフォルト値で
9. 索引の作成
  - もし本当に必要ならば.....

# アジェンダ

- 1 パフォーマンス・レース！
- 2 データウェアハウス・デザインのルールとフレームワーク
- 3 実行方式を選ぼう
- 4 複数ユーザー・パフォーマンス・レース！
- 5 プラットフォームを選ぼう
- 6 ハイパフォーマンスを実現するために一番大事なこと

# 実行方式を選ぼう



フィルタ後のファクト表から  
**少しの行**をどうやって抽出する？



フィルタ後のファクト表から  
**たくさんの行**をどうやって抽出する？

# ビットマップ索引によるスター型変換



- ファクト表から  
 少しの行を抽出する
- バッファ・キャッシュに  
 収まる程度に  
 索引が十分に小さい
- 同時実行数が多い
- ```
ALTER SESSION  
SET STAR_TRANSFORMATION_ENABLED  
= TRUE
```

# スター型変換

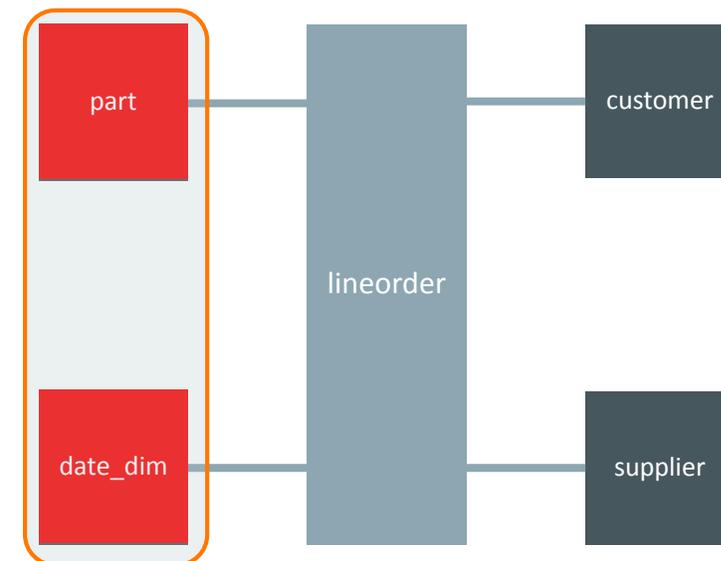
```
SELECT
    d_sellingseason, p_category, s_region,
    SUM(lo_extendedprice)
FROM
    lineorder
  JOIN customer ON lo_custkey = c_custkey
  JOIN date_dim ON lo_orderdate = d_datekey
  JOIN part ON lo_partkey = p_partkey
  JOIN supplier ON lo_suppkey = s_suppkey
WHERE
    d_year IN (1993, 1994, 1995)
  AND
    p_container IN ('JUMBO PACK')
GROUP BY
    d_sellingseason, p_category, s_region
ORDER BY
    d_sellingseason, p_category, s_region;
```



```
SELECT
    d_sellingseason, p_category, s_region,
    SUM(lo_extendedprice)
FROM
    lineorder
WHERE
    lo_orderdate IN (
        SELECT d_datekey
        FROM date_dim
        WHERE d_year IN (1993, 1994, 1995)
    )
  AND
    lo_partkey IN (
        SELECT p_partkey
        FROM part
        WHERE p_container IN ('JUMBO PACK')
    )
GROUP BY
    d_sellingseason, p_category, s_region
ORDER BY
    d_sellingseason, p_category, s_region;
```

# スター型変換: フィルタの作成

| Operation                   | Object Name                 | Predicate information          |
|-----------------------------|-----------------------------|--------------------------------|
| SELECT STATEMENT            |                             |                                |
| TEMP TABLE TRANSFORMATION   |                             |                                |
| LOAD AS SELECT              | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| TABLE ACCESS FULL           | DATE_DIM                    | D_YEAR IN ( 1993, 1994, 1995 ) |
| LOAD AS SELECT              | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| TABLE ACCESS FULL           | PART                        | P_CONTAINER = 'JUMBO PACK'     |
| SORT GROUP BY               |                             |                                |
| HASH JOIN                   |                             | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| HASH JOIN                   |                             | LO_ORDERDATE = D_DATEKEY       |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| HASH JOIN                   |                             | LO_SUPPKEY = S_SUPPKEY         |
| TABLE ACCESS FULL           | SUPPLIER                    |                                |
| VIEW                        | VW_ST_F981A0CC              |                                |
| NESTED LOOPS                |                             |                                |
| PARTITION RANGE SUBQUERY    |                             |                                |
| BITMAP CONVERSION TO ROWIDS |                             |                                |
| BITMAP AND                  |                             |                                |
| BITMAP MERGE                |                             |                                |
| BITMAP KEY ITERATION        |                             |                                |
| BUFFER SORT                 |                             |                                |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| BITMAP INDEX RANGE SCAN     | LO_DATE_B                   | LO_ORDERDATE = D_DATEKEY       |
| BITMAP MERGE                |                             |                                |
| BITMAP KEY ITERATION        |                             |                                |
| BUFFER SORT                 |                             |                                |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| BITMAP INDEX RANGE SCAN     | LO_PART_B                   | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS BY USER ROWID  | LINEORDER                   |                                |

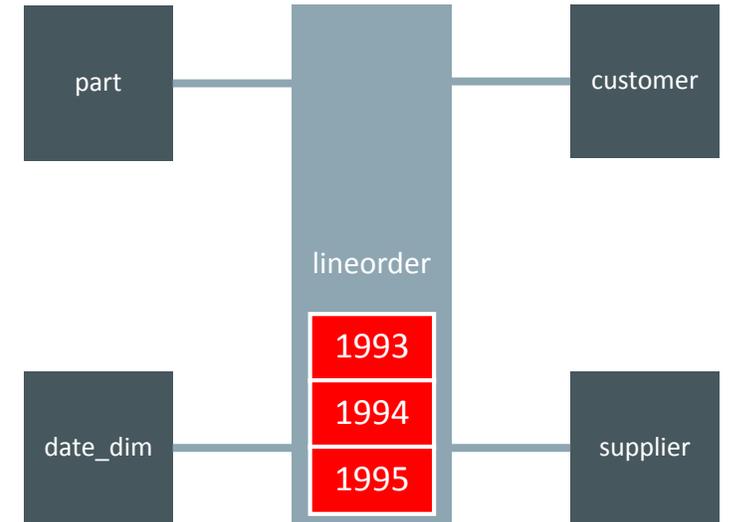


```

SELECT    d_sellingseason, p_category, s_region,
          SUM(lo_extendedprice)
FROM      lineorder
JOIN      customer ON lo_custkey = c_custkey
JOIN      date_dim  ON lo_orderdate = d_datekey
JOIN      part      ON lo_partkey = p_partkey
JOIN      supplier  ON lo_suppkey = s_suppkey
WHERE     d_year IN (1993, 1994, 1995)
AND       p_container IN ('JUMBO PACK')
GROUP BY d_sellingseason, p_category, s_region
ORDER BY d_sellingseason, p_category, s_region
    
```

# スター型変換: ファクト表からの抽出

| Operation                   | Object Name                 | Predicate information          |
|-----------------------------|-----------------------------|--------------------------------|
| SELECT STATEMENT            |                             |                                |
| TEMP TABLE TRANSFORMATION   |                             |                                |
| LOAD AS SELECT              | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| TABLE ACCESS FULL           | DATE_DIM                    | D_YEAR IN ( 1993, 1994, 1995 ) |
| LOAD AS SELECT              | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| TABLE ACCESS FULL           | PART                        | P_CONTAINER = 'JUMBO PACK'     |
| SORT GROUP BY               |                             |                                |
| HASH JOIN                   |                             | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| HASH JOIN                   |                             | LO_ORDERDATE = D_DATEKEY       |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| HASH JOIN                   |                             | LO_SUPPKEY = S_SUPPKEY         |
| TABLE ACCESS FULL           | SUPPLIER                    |                                |
| VIEW                        | VW_ST_F981A0CC              |                                |
| NESTED LOOPS                |                             |                                |
| PARTITION RANGE SUBQUERY    |                             |                                |
| BITMAP CONVERSION TO ROWIDS |                             |                                |
| BITMAP AND                  |                             |                                |
| BITMAP MERGE                |                             |                                |
| BITMAP KEY ITERATION        |                             |                                |
| BUFFER SORT                 |                             |                                |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| BITMAP INDEX RANGE SCAN     | LO_DATE_B                   | LO_ORDERDATE = D_DATEKEY       |
| BITMAP MERGE                |                             |                                |
| BITMAP KEY ITERATION        |                             |                                |
| BUFFER SORT                 |                             |                                |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| BITMAP INDEX RANGE SCAN     | LO_PART_B                   | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS BY USER ROWID  | LINEORDER                   |                                |

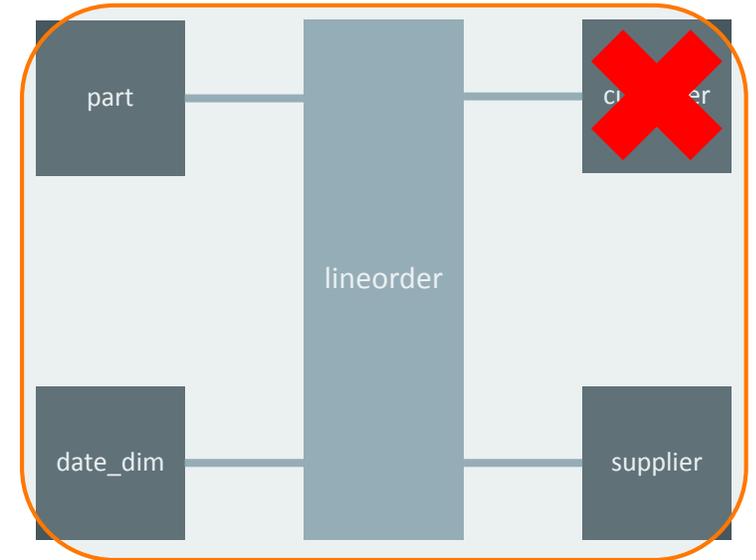


```

SELECT  d_sellingseason, p_category, s_region,
        SUM(lo_extendedprice)
FROM    lineorder
JOIN    customer ON lo_custkey = c_custkey
JOIN    date_dim  ON lo_orderdate = d_datekey
JOIN    part      ON lo_partkey = p_partkey
JOIN    supplier  ON lo_suppkey = s_suppkey
WHERE   d_year IN (1993, 1994, 1995)
AND     p_container IN ('JUMBO PACK')
GROUP BY d_sellingseason, p_category, s_region
ORDER BY d_sellingseason, p_category, s_region
    
```

# スター型変換: デイメンション表との結合

| Operation                   | Object Name                 | Predicate information          |
|-----------------------------|-----------------------------|--------------------------------|
| SELECT STATEMENT            |                             |                                |
| TEMP TABLE TRANSFORMATION   |                             |                                |
| LOAD AS SELECT              | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| TABLE ACCESS FULL           | DATE_DIM                    | D_YEAR IN ( 1993, 1994, 1995 ) |
| LOAD AS SELECT              | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| TABLE ACCESS FULL           | PART                        | P_CONTAINER = 'JUMBO PACK'     |
| SORT GROUP BY               |                             |                                |
| HASH JOIN                   |                             | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| HASH JOIN                   |                             | LO_ORDERDATE = D_DATEKEY       |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| HASH JOIN                   |                             | LO_SUPPKEY = S_SUPPKEY         |
| TABLE ACCESS FULL           | SUPPLIER                    |                                |
| VIEW                        | VW_ST_F981A0CC              |                                |
| NESTED LOOPS                |                             |                                |
| PARTITION RANGE SUBQUERY    |                             |                                |
| BITMAP CONVERSION TO ROWIDS |                             |                                |
| BITMAP AND                  |                             |                                |
| BITMAP MERGE                |                             |                                |
| BITMAP KEY ITERATION        |                             |                                |
| BUFFER SORT                 |                             |                                |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| BITMAP INDEX RANGE SCAN     | LO_DATE_B                   | LO_ORDERDATE = D_DATEKEY       |
| BITMAP MERGE                |                             |                                |
| BITMAP KEY ITERATION        |                             |                                |
| BUFFER SORT                 |                             |                                |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| BITMAP INDEX RANGE SCAN     | LO_PART_B                   | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS BY USER ROWID  | LINEORDER                   |                                |

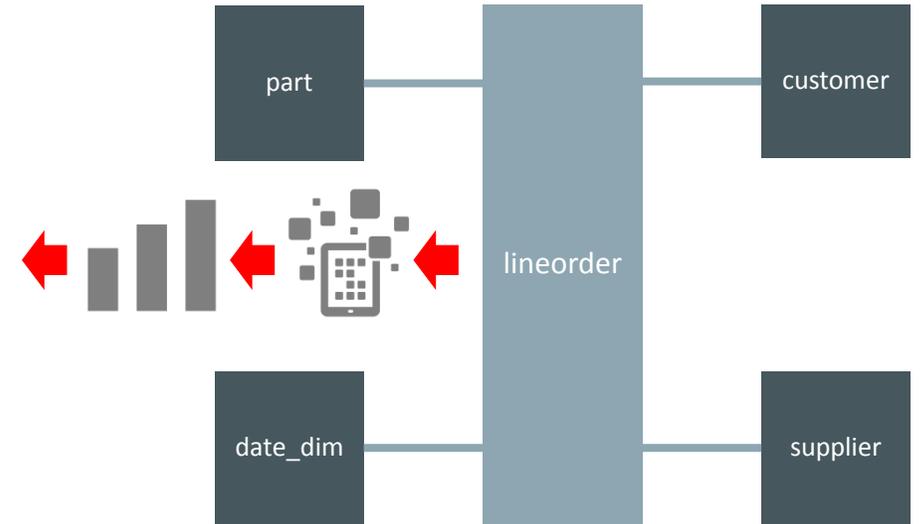


```

SELECT  d_sellingseason, p_category, s_region,
        SUM(lo_extendedprice)
FROM    lineorder
JOIN    customer ON lo_custkey = c_custkey
JOIN    date_dim    ON lo_orderdate = d_datekey
JOIN    part        ON lo_partkey = p_partkey
JOIN    supplier    ON lo_suppkey = s_suppkey
WHERE   d_year IN (1993, 1994, 1995)
AND     p_container IN ('JUMBO PACK')
GROUP BY d_sellingseason, p_category, s_region
ORDER BY d_sellingseason, p_category, s_region
    
```

# スター型変換: 集合演算とソート

| Operation                   | Object Name                 | Predicate information          |
|-----------------------------|-----------------------------|--------------------------------|
| SELECT STATEMENT            |                             |                                |
| TEMP TABLE TRANSFORMATION   |                             |                                |
| LOAD AS SELECT              | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| TABLE ACCESS FULL           | DATE_DIM                    | D_YEAR IN ( 1993, 1994, 1995 ) |
| LOAD AS SELECT              | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| TABLE ACCESS FULL           | PART                        | P_CONTAINER = 'JUMBO PACK'     |
| SORT GROUP BY               |                             |                                |
| HASH JOIN                   |                             | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| HASH JOIN                   |                             | LO_ORDERDATE = D_DATEKEY       |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| HASH JOIN                   |                             | LO_SUPPKEY = S_SUPPKEY         |
| TABLE ACCESS FULL           | SUPPLIER                    |                                |
| VIEW                        | VW_ST_F981A0CC              |                                |
| NESTED LOOPS                |                             |                                |
| PARTITION RANGE SUBQUERY    |                             |                                |
| BITMAP CONVERSION TO ROWIDS |                             |                                |
| BITMAP AND                  |                             |                                |
| BITMAP MERGE                |                             |                                |
| BITMAP KEY ITERATION        |                             |                                |
| BUFFER SORT                 |                             |                                |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| BITMAP INDEX RANGE SCAN     | LO_DATE_B                   | LO_ORDERDATE = D_DATEKEY       |
| BITMAP MERGE                |                             |                                |
| BITMAP KEY ITERATION        |                             |                                |
| BUFFER SORT                 |                             |                                |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| BITMAP INDEX RANGE SCAN     | LO_PART_B                   | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS BY USER ROWID  | LINEORDER                   |                                |



```

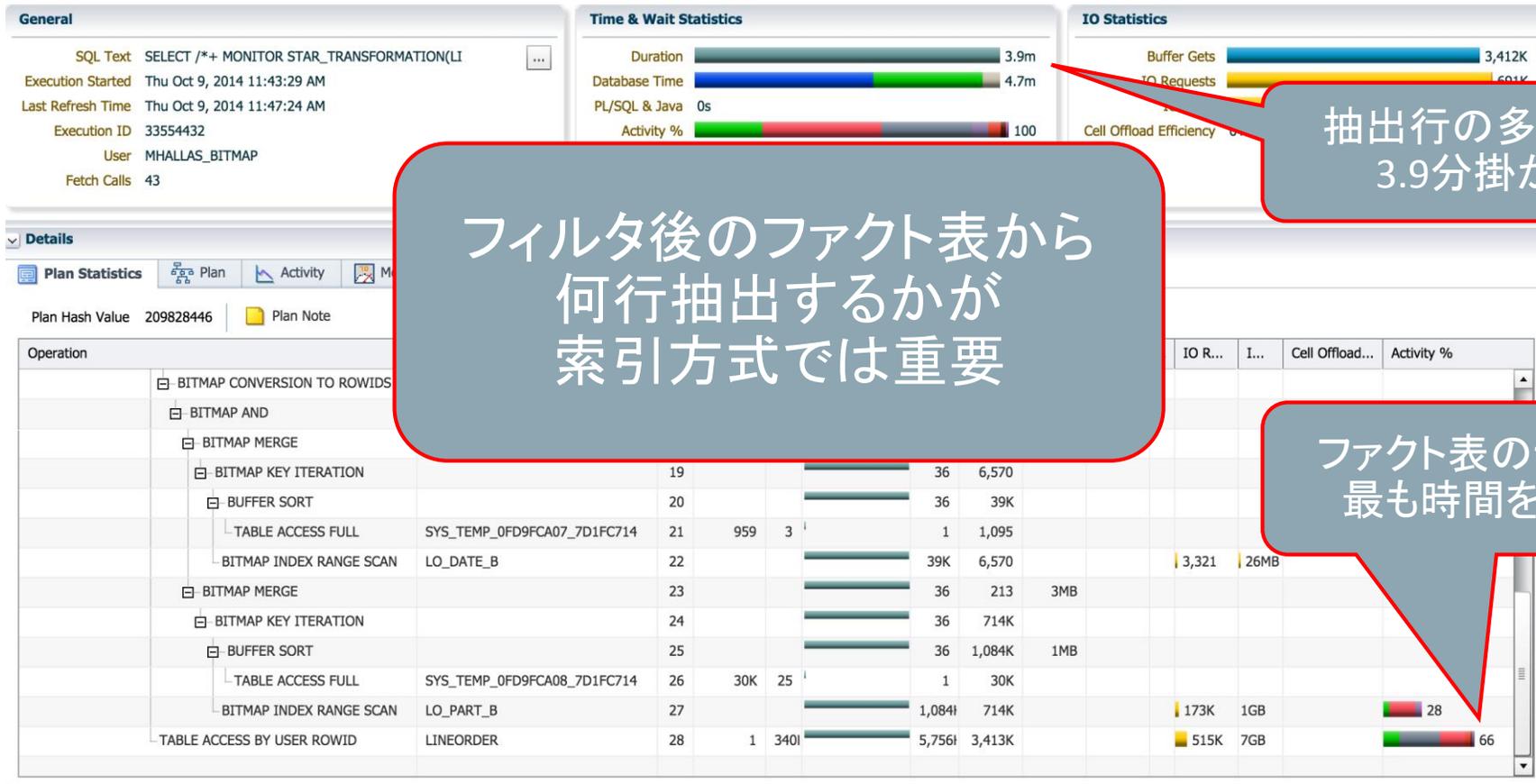
SELECT      d_sellingseason, p_category, s_region,
            SUM(lo_extendedprice)
FROM        lineorder
JOIN        customer ON lo_custkey = c_custkey
JOIN        date_dim  ON lo_orderdate = d_datekey
JOIN        part      ON lo_partkey = p_partkey
JOIN        supplier  ON lo_suppkey = s_suppkey
WHERE       d_year IN (1993, 1994, 1995)
AND         p_container IN ('JUMBO PACK')
GROUP BY   d_sellingseason, p_category, s_region
ORDER BY   d_sellingseason, p_category, s_region
    
```

# スター型変換で気をつけなければいけないこと

| Operation                   | Object Name                 | Predicate information          |
|-----------------------------|-----------------------------|--------------------------------|
| SELECT STATEMENT            |                             |                                |
| TEMP TABLE TRANSFORMATION   |                             |                                |
| LOAD AS SELECT              | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| TABLE ACCESS FULL           | DATE_DIM                    | D_YEAR IN ( 1993, 1994, 1995 ) |
| LOAD AS SELECT              | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| TABLE ACCESS FULL           | PART                        | P_CONTAINER = 'JUMBO PACK'     |
| SORT GROUP BY               |                             |                                |
| HASH JOIN                   |                             | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| HASH JOIN                   |                             | LO_ORDERDATE = D_DATEKEY       |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| HASH JOIN                   |                             | LO_SUPPKEY = S_SUPPKEY         |
| TABLE ACCESS FULL           | SUPPLIER                    |                                |
| VIEW                        | VW_ST_F981A0CC              |                                |
| NESTED LOOPS                |                             |                                |
| PARTITION RANGE SUBQUERY    |                             |                                |
| BITMAP CONVERSION TO ROWIDS |                             |                                |
| BITMAP AND                  |                             |                                |
| BITMAP MERGE                |                             |                                |
| BITMAP KEY ITERATION        |                             |                                |
| BUFFER SORT                 |                             |                                |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA09_7D1FC714 |                                |
| BITMAP INDEX RANGE SCAN     | LO_DATE_B                   | LO_ORDERDATE = D_DATEKEY       |
| BITMAP MERGE                |                             |                                |
| BITMAP KEY ITERATION        |                             |                                |
| BUFFER SORT                 |                             |                                |
| TABLE ACCESS FULL           | SYS_TEMP_0FD9FCA0A_7D1FC714 |                                |
| BITMAP INDEX RANGE SCAN     | LO_PART_B                   | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS BY USER ROWID  | LINEORDER                   |                                |

- 1回のランダムIOに5ミリ秒かかるシステム
- ファクト表から5行を抽出し、それらがバッファ・キャッシュに乗っていなかったら何秒かかる？
- 100万行を抽出するなら何秒かかる？

# スター型変換で気をつけなければいけないこと



フィルタ後のファクト表から何行抽出するかが索引方式では重要

抽出の多いクエリーで3.9分掛かっている

ファクト表のランダムIOが最も時間を占めている

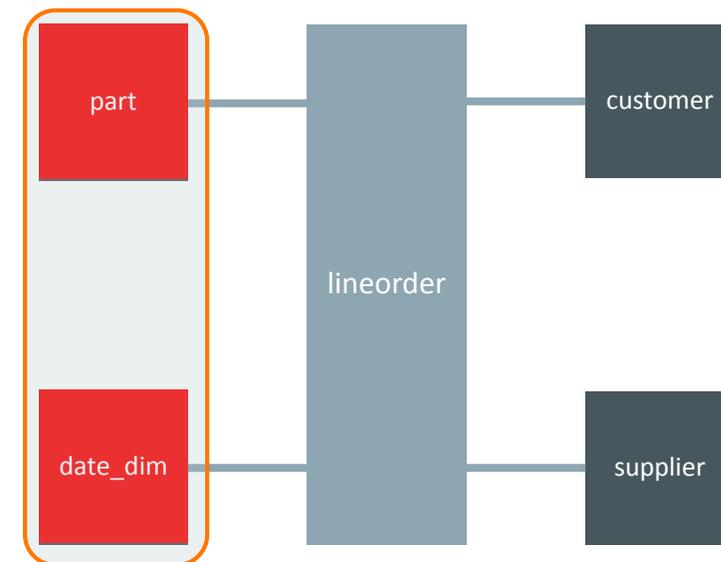
# 賢いフィルタリングを伴う表スキャン



- ファクト表から  
たくさんの行を抽出する
- データベース・サイズが  
大きい
- Exadata or In-Memory

# 表スキャン: フィルタの作成

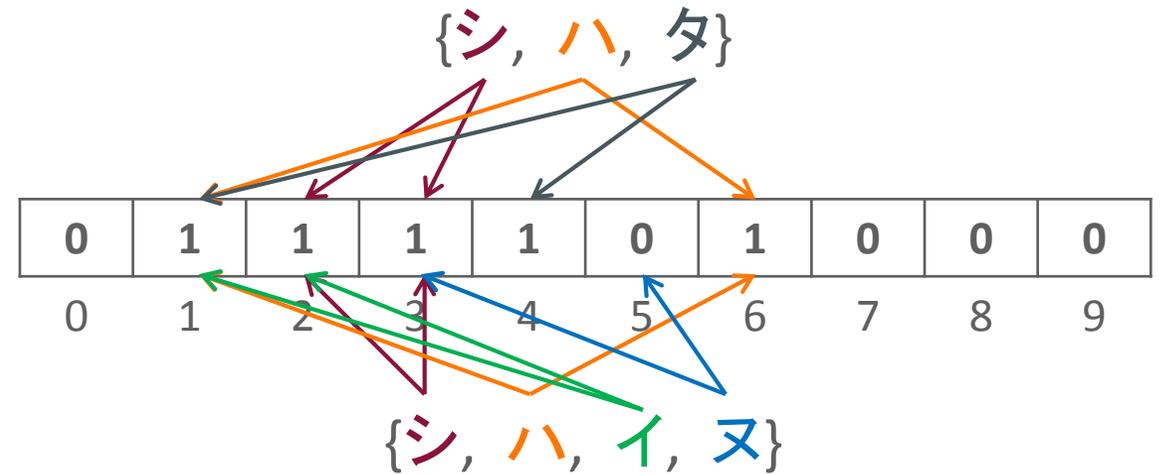
| Operation                   | Object Name | Predicate information          |
|-----------------------------|-------------|--------------------------------|
| SELECT STATEMENT            |             |                                |
| SORT GROUP BY               |             |                                |
| HASH JOIN                   |             | LO_SUPPKEY = S_SUPPKEY         |
| TABLE ACCESS STORAGE FULL   | SUPPLIER    |                                |
| HASH JOIN                   |             |                                |
| JOIN FILTER CREATE          | :BF0001     |                                |
| PART JOIN FILTER CREATE     | :BF0000     | LO_ORDERDATE = D_DATEKEY       |
| TABLE ACCESS STORAGE FULL   | DATE_DIM    | D_YEAR IN ( 1993, 1994, 1995 ) |
| HASH JOIN                   |             |                                |
| JOIN FILTER CREATE          | :BF0002     | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS STORAGE FULL   | PART        | P_CONTAINER = 'JUMBO PACK'     |
| JOIN FILTER USE             | :BF0001     |                                |
| JOIN FILTER USE             | :BF0002     |                                |
| PARTITION RANGE JOIN-FILTER |             |                                |
| TABLE ACCESS STORAGE FULL   | LINEORDER   | :BF0000                        |



```
SELECT    d_sellingseason, p_category, s_region,
          SUM(lo_extendedprice)
FROM      lineorder
  JOIN    customer ON lo_custkey = c_custkey
  JOIN    date_dim  ON lo_orderdate = d_datekey
  JOIN    part      ON lo_partkey = p_partkey
  JOIN    supplier  ON lo_suppkey = s_suppkey
WHERE     d_year IN (1993, 1994, 1995)
AND       p_container IN ('JUMBO PACK')
GROUP BY d_sellingseason, p_category, s_region
ORDER BY d_sellingseason, p_category, s_region
```

# ブルーム・フィルタとは

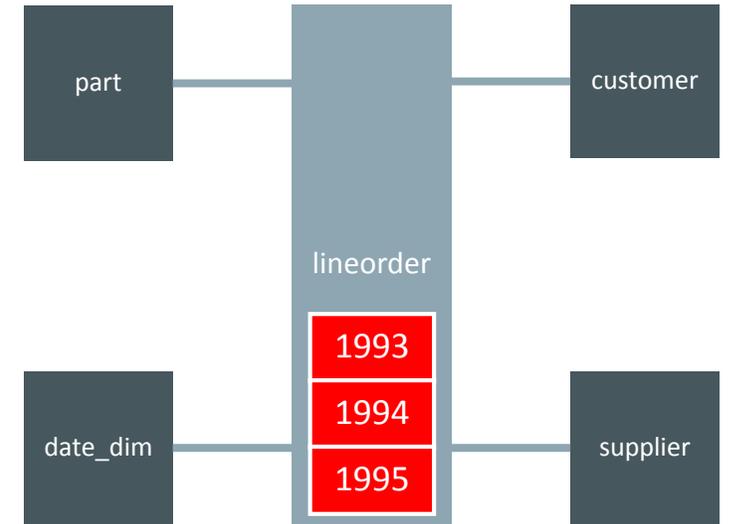
- 偽陽性 (False Positive) はあるが偽陰性 (False Negative) はない効率の良い(軽い)フィルタ
- {A, B, C, D, E, F} からAとBとCだけ残したい
  - {A, B, C} のディメンション表を使って {A, B, C, D, E, F} のファクト表をフィルタしたい
- ブルーム・フィルタを適用すると {A, B, C, E} になるかもしれないが {A, C} には決してならない



- シとハとイは {シ, ハ, タ} の集合に含まれているかもしれない
- ヌは {シ, ハ, タ} の集合に決して含まれない
- {シ, ハ, イ, ヌ} は {シ, ハ, イ} にフィルタできる
- ブルーム・フィルタには実データが存在しない

# 表スキャン: ファクト表からの抽出

| Operation                   | Object Name | Predicate information          |
|-----------------------------|-------------|--------------------------------|
| SELECT STATEMENT            |             |                                |
| SORT GROUP BY               |             |                                |
| HASH JOIN                   |             | LO_SUPPKEY = S_SUPPKEY         |
| TABLE ACCESS STORAGE FULL   | SUPPLIER    |                                |
| HASH JOIN                   |             |                                |
| JOIN FILTER CREATE          | :BF0001     |                                |
| PART JOIN FILTER CREATE     | :BF0000     | LO_ORDERDATE = D_DATEKEY       |
| TABLE ACCESS STORAGE FULL   | DATE_DIM    | D_YEAR IN ( 1993, 1994, 1995 ) |
| HASH JOIN                   |             |                                |
| JOIN FILTER CREATE          | :BF0002     | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS STORAGE FULL   | PART        | P_CONTAINER = 'JUMBO PACK'     |
| JOIN FILTER USE             | :BF0001     |                                |
| JOIN FILTER USE             | :BF0002     |                                |
| PARTITION RANGE JOIN-FILTER |             |                                |
| TABLE ACCESS STORAGE FULL   | LINEORDER   | :BF0000                        |

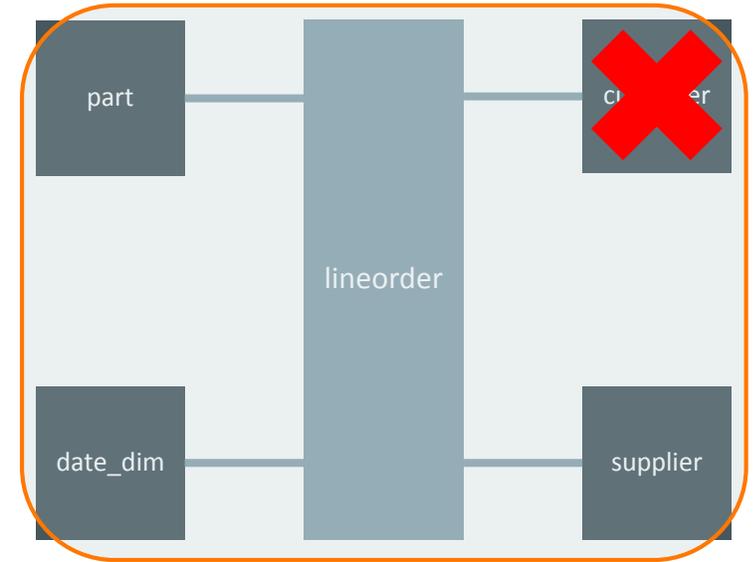


```

SELECT  d_sellingseason, p_category, s_region,
        SUM(lo_extendedprice)
FROM    lineorder
JOIN    customer ON lo_custkey = c_custkey
JOIN    date_dim  ON lo_orderdate = d_datekey
JOIN    part      ON lo_partkey = p_partkey
JOIN    supplier  ON lo_suppkey = s_suppkey
WHERE   d_year IN (1993, 1994, 1995)
AND     p_container IN ('JUMBO PACK')
GROUP BY d_sellingseason, p_category, s_region
ORDER BY d_sellingseason, p_category, s_region
    
```

# 表スキャン: ディメンション表との結合

| Operation                   | Object Name | Predicate information          |
|-----------------------------|-------------|--------------------------------|
| SELECT STATEMENT            |             |                                |
| SORT GROUP BY               |             |                                |
| HASH JOIN                   |             | LO_SUPPKEY = S_SUPPKEY         |
| TABLE ACCESS STORAGE FULL   | SUPPLIER    |                                |
| HASH JOIN                   |             |                                |
| JOIN FILTER CREATE          | :BF0001     |                                |
| PART JOIN FILTER CREATE     | :BF0000     | LO_ORDERDATE = D_DATEKEY       |
| TABLE ACCESS STORAGE FULL   | DATE_DIM    | D_YEAR IN ( 1993, 1994, 1995 ) |
| HASH JOIN                   |             |                                |
| JOIN FILTER CREATE          | :BF0002     | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS STORAGE FULL   | PART        | P_CONTAINER = 'JUMBO PACK'     |
| JOIN FILTER USE             | :BF0001     |                                |
| JOIN FILTER USE             | :BF0002     |                                |
| PARTITION RANGE JOIN-FILTER |             |                                |
| TABLE ACCESS STORAGE FULL   | LINEORDER   | :BF0000                        |

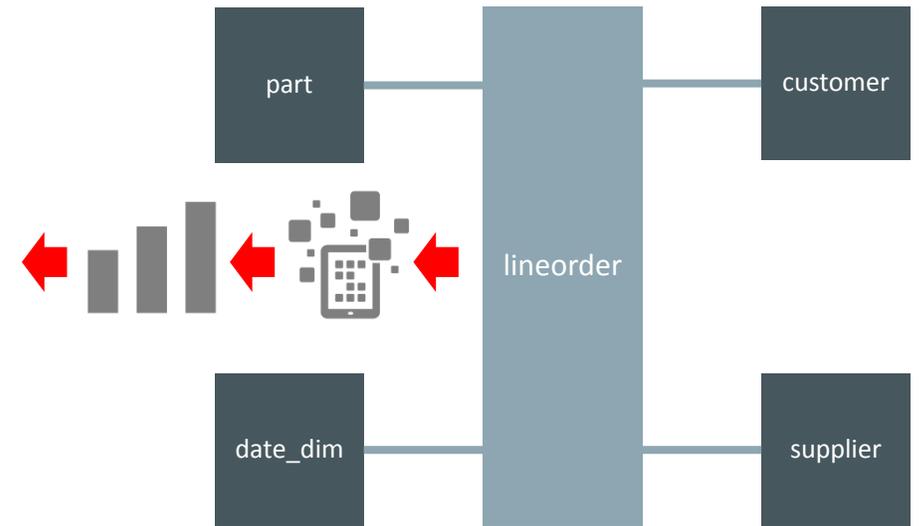


```

SELECT  d_sellingseason, p_category, s_region,
        SUM(lo_extendedprice)
FROM    lineorder
JOIN    customer ON lo_custkey = c_custkey
JOIN    date_dim    ON lo_orderdate = d_datekey
JOIN    part        ON lo_partkey = p_partkey
JOIN    supplier    ON lo_suppkey = s_suppkey
WHERE   d_year IN (1993, 1994, 1995)
AND     p_container IN ('JUMBO PACK')
GROUP BY d_sellingseason, p_category, s_region
ORDER BY d_sellingseason, p_category, s_region
    
```

# 表スキャン: 集合演算とソート

| Operation                   | Object Name | Predicate information          |
|-----------------------------|-------------|--------------------------------|
| SELECT STATEMENT            |             |                                |
| SORT GROUP BY               |             |                                |
| HASH JOIN                   |             | LO_SUPPKEY = S_SUPPKEY         |
| TABLE ACCESS STORAGE FULL   | SUPPLIER    |                                |
| HASH JOIN                   |             |                                |
| JOIN FILTER CREATE          | :BF0001     |                                |
| PART JOIN FILTER CREATE     | :BF0000     | LO_ORDERDATE = D_DATEKEY       |
| TABLE ACCESS STORAGE FULL   | DATE_DIM    | D_YEAR IN ( 1993, 1994, 1995 ) |
| HASH JOIN                   |             |                                |
| JOIN FILTER CREATE          | :BF0002     | LO_PARTKEY = P_PARTKEY         |
| TABLE ACCESS STORAGE FULL   | PART        | P_CONTAINER = 'JUMBO PACK'     |
| JOIN FILTER USE             | :BF0001     |                                |
| JOIN FILTER USE             | :BF0002     |                                |
| PARTITION RANGE JOIN-FILTER |             |                                |
| TABLE ACCESS STORAGE FULL   | LINEORDER   | :BF0000                        |



```

SELECT    d_sellingseason, p_category, s_region,
          SUM(lo_extendedprice)
FROM      lineorder
  JOIN    customer ON lo_custkey = c_custkey
  JOIN    date_dim  ON lo_orderdate = d_datekey
  JOIN    part      ON lo_partkey = p_partkey
  JOIN    supplier  ON lo_suppkey = s_suppkey
WHERE    d_year IN (1993, 1994, 1995)
  AND    p_container IN ('JUMBO PACK')
GROUP BY d_sellingseason, p_category, s_region
ORDER BY d_sellingseason, p_category, s_region
    
```

# アジェンダ

- 1 パフォーマンス・レース！
- 2 データウェアハウス・デザインのルールとフレームワーク
- 3 実行方式を選ぼう
- 4 複数ユーザー・パフォーマンス・レース！
- 5 プラットフォームを選ぼう
- 6 ハイパフォーマンスを実現するために一番大事なこと

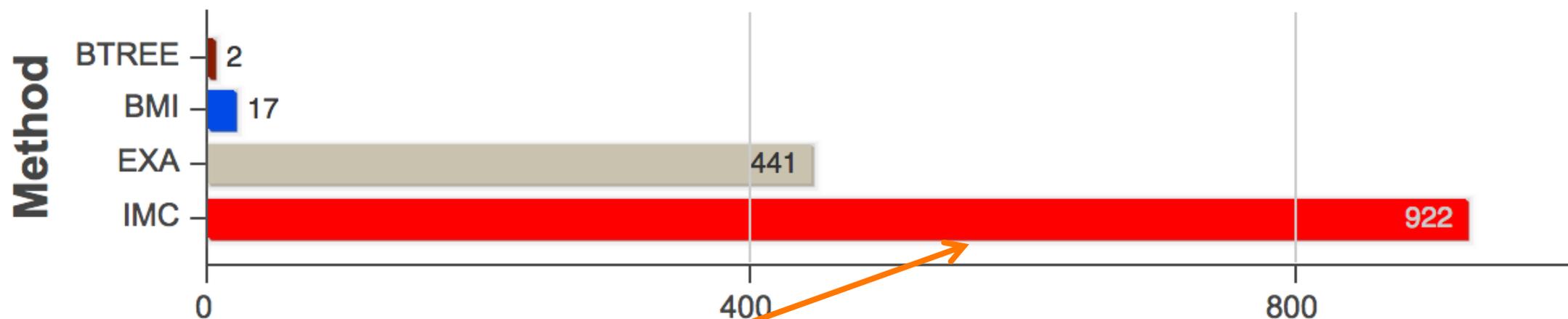
# 複数ユーザー・パフォーマンス・レース！

- スター・スキーマで構成されたデータモデル
- 4人のレーサー:  
Bツリー索引 / ビットマップ索引  
Exadata表スキャン / In-Memory
- 4同時ユーザー(同時実行)から開始



# 複数ユーザー・パフォーマンス・レース！

4同時ユーザー (DoP=2)



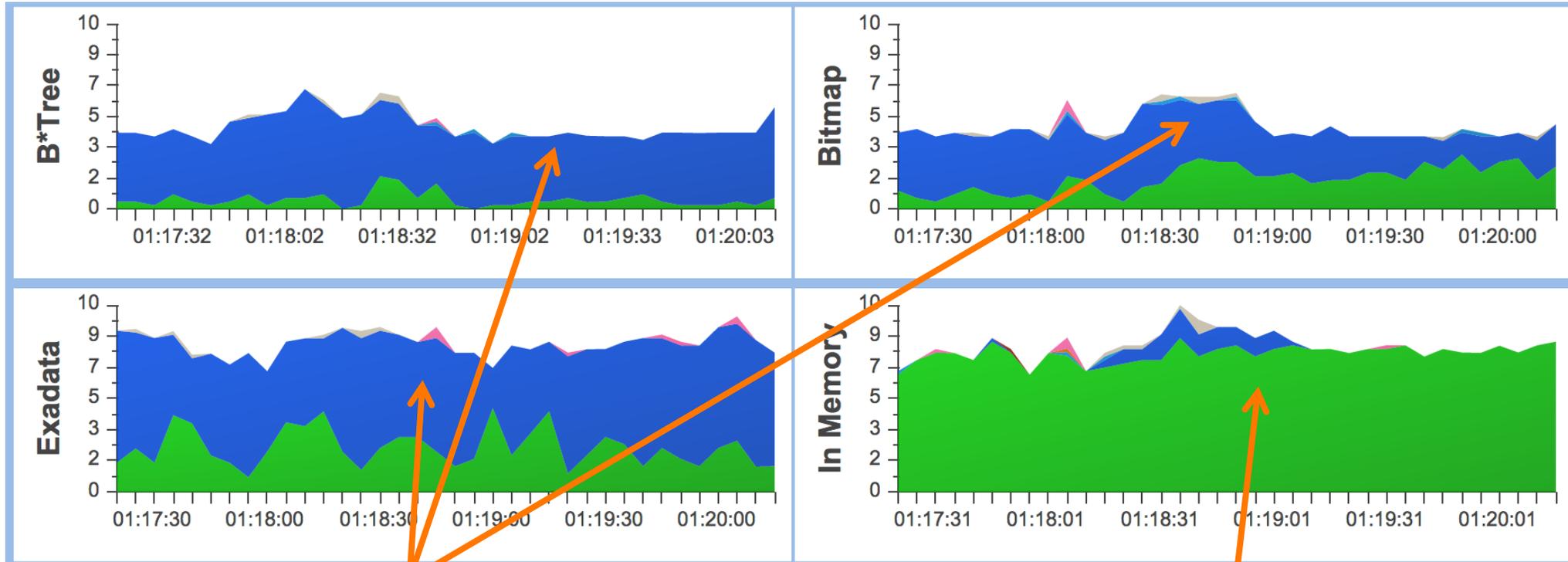
In-Memoryが  
Exadata表スキャンに  
2倍強の差をつけてゴール

Queries Completed



# 複数ユーザー・パフォーマンス・レース！

4同時ユーザー (DoP=2)

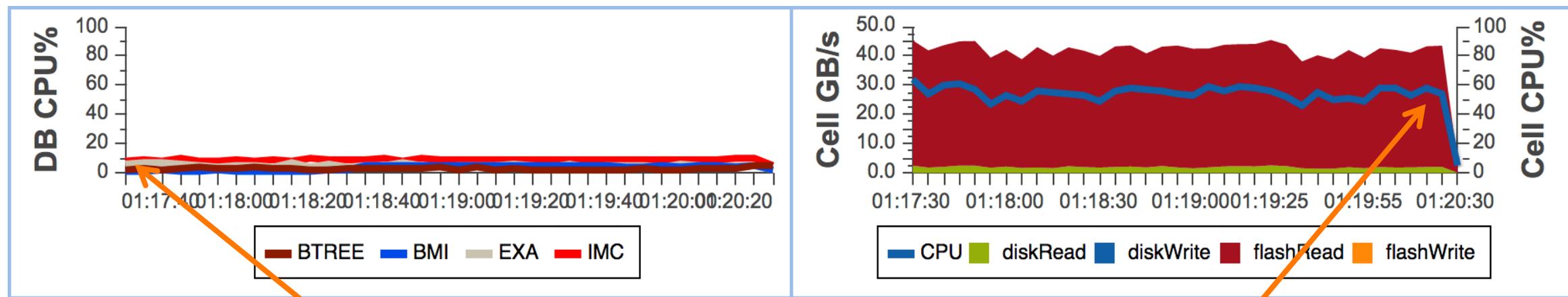


Exadataと索引は  
IOの待機イベントが多い

In-MemoryはほぼCPU

# 複数ユーザー・パフォーマンス・レース！

4同時ユーザー (DoP=2)



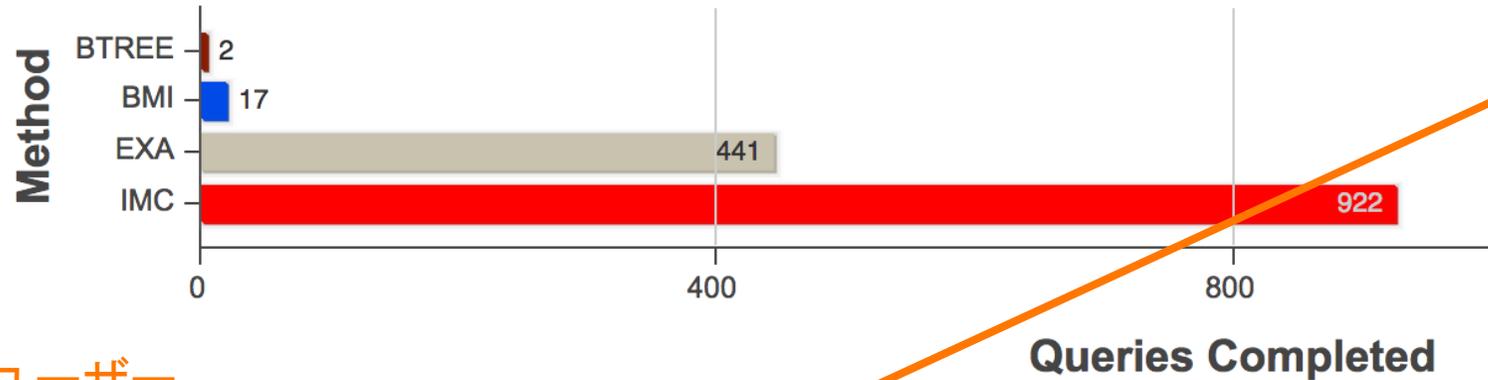
どのレーサーも  
CPU使用率にかなり余裕があるので  
8同時ユーザーに増やしてみる

Exadata Storage Server の  
CPU使用率は  
4同時ユーザーの時点で50%前後

# 複数ユーザー・パフォーマンス・レース！

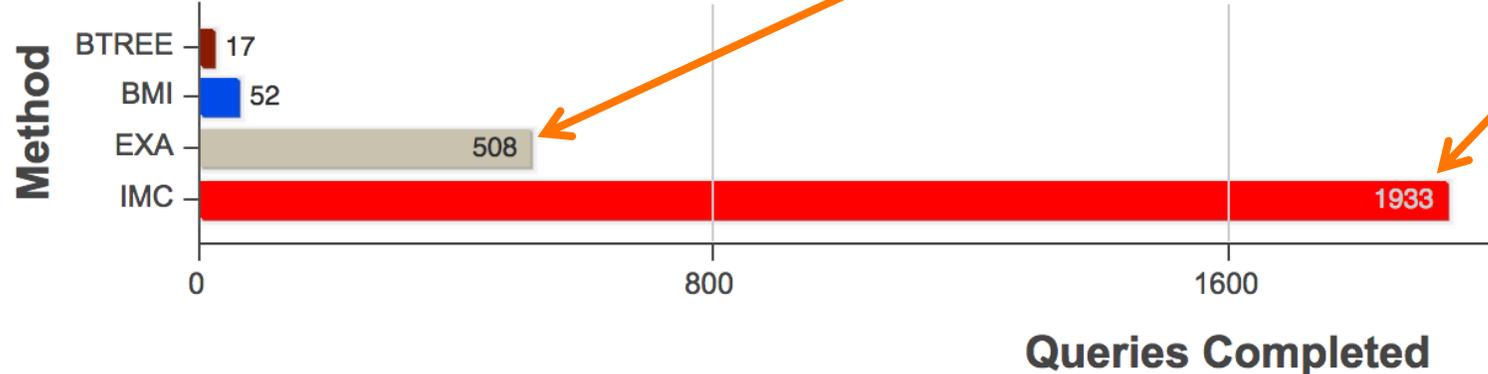
## 4 vs. 8同時ユーザー (DoP=2)

### 4同時ユーザー



Exadataは  
2倍のユーザー数で  
15%増程度の  
クエリー処理量

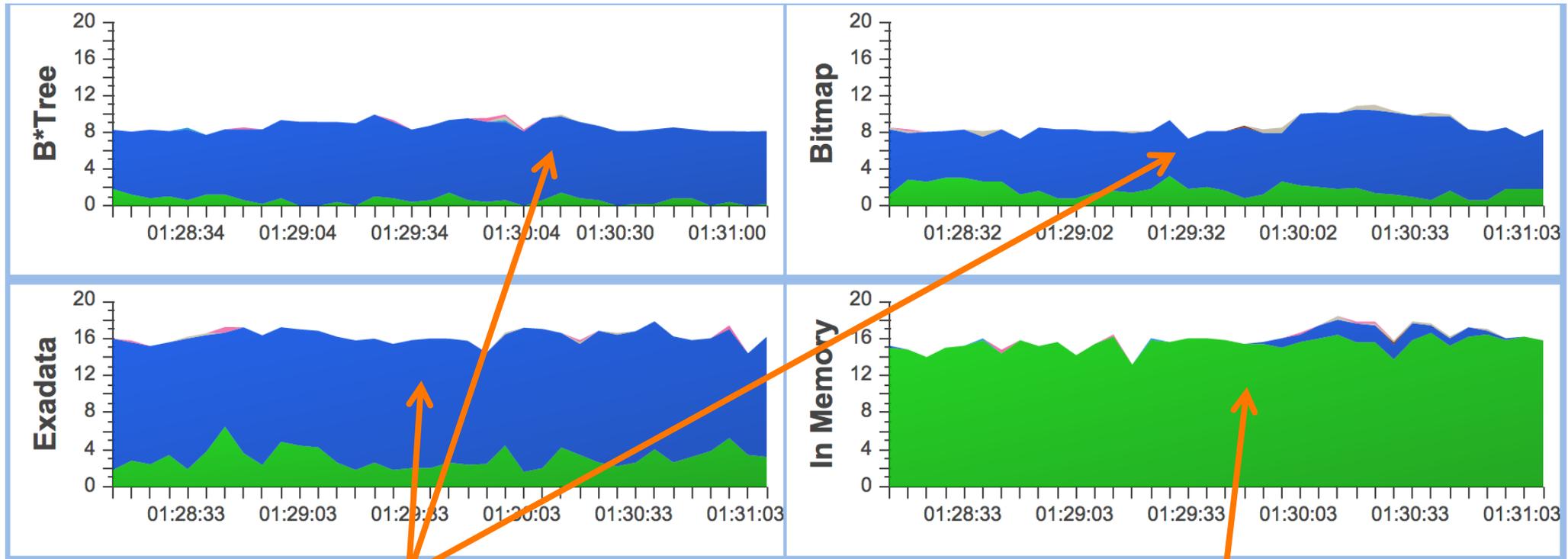
### 8同時ユーザー



In-Memoryは  
2倍のユーザー数で  
2倍のクエリー処理量

# 複数ユーザー・パフォーマンス・レース！

8同時ユーザー (DoP=2)

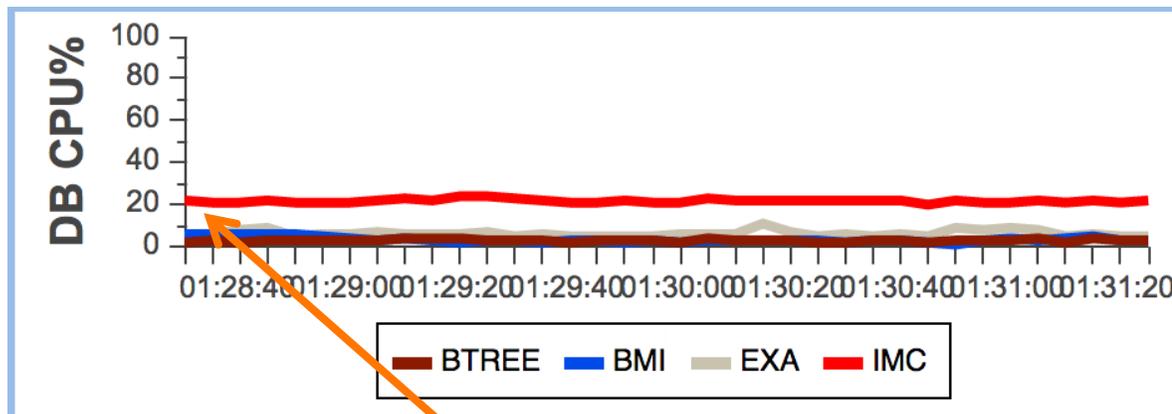


Exadataと索引は  
IOの待機イベントが多い

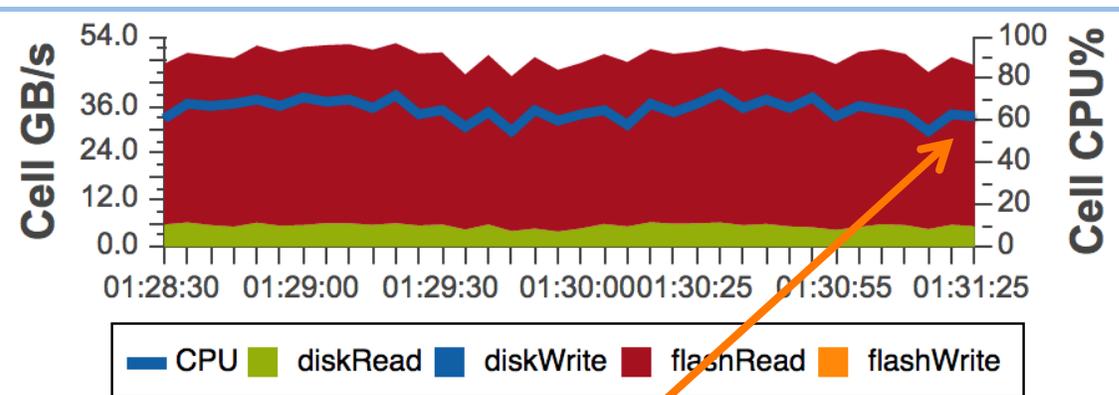
In-MemoryはほぼCPU

# 複数ユーザー・パフォーマンス・レース！

8同時ユーザー (DoP=2)



In-Memoryも  
CPU使用率にまだ余裕があるので  
32同時ユーザーに増やしてみる

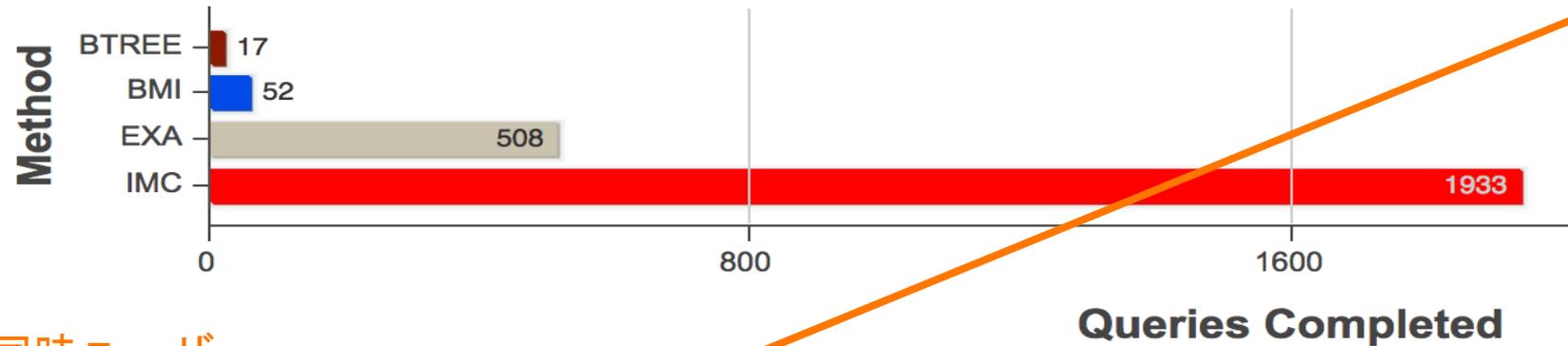


Exadata Storage Server の  
CPU使用率は  
8同時ユーザーで70%前後。  
IO帯域限界にも達しつつある

# 複数ユーザー・パフォーマンス・レース！

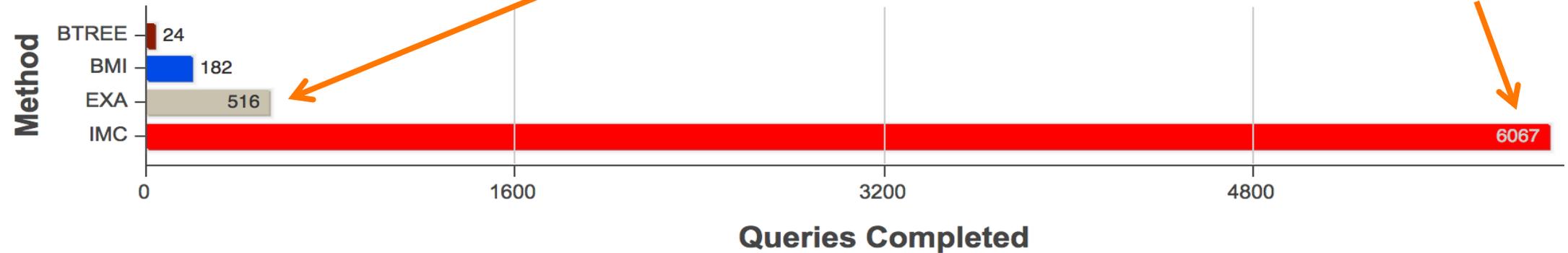
8 vs. 32同時ユーザー (DoP=2)

8同時ユーザー



Exadataは  
4倍のユーザー数なのに  
クエリー処理量が  
ほぼ変わらない

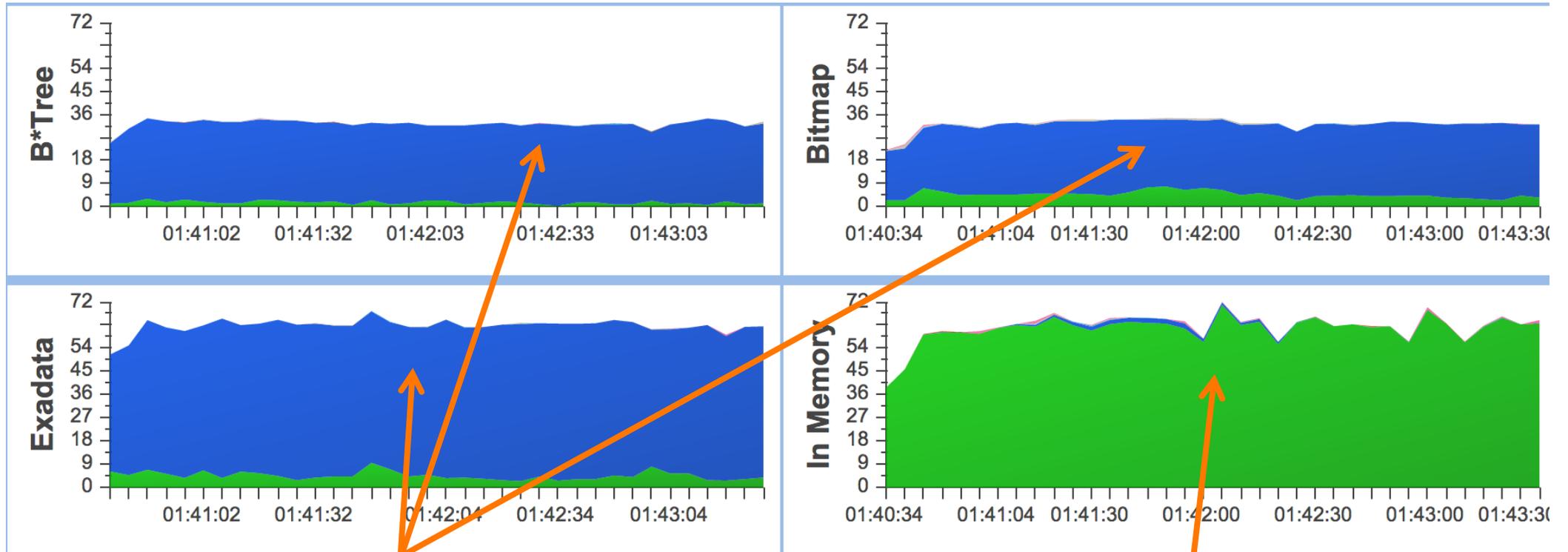
32同時ユーザー



In-Memoryは  
比較的にニアに  
増えている

# 複数ユーザー・パフォーマンス・レース！

32同時ユーザー (DoP=2)

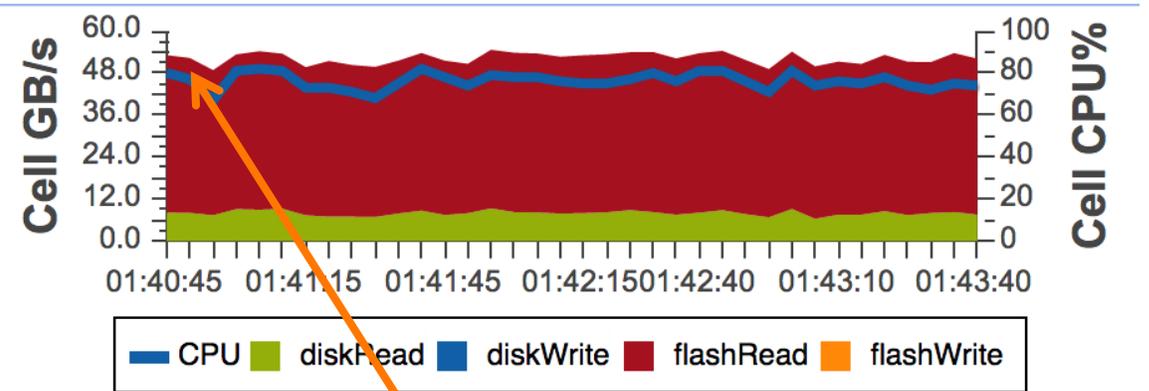
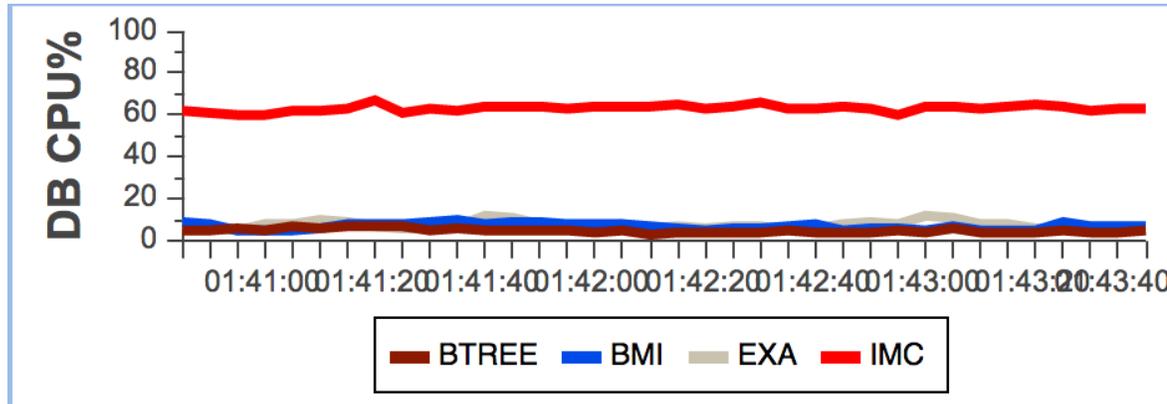


Exadataと索引は  
IOの待機イベントが多い

In-MemoryはほぼCPU

# 複数ユーザー・パフォーマンス・レース！

32同時ユーザー (DoP=2)



IO帯域限界に達している

# 複数ユーザー・パフォーマンス・レースで学んだこと

- 少数人数では、In-MemoryがExadata表スキャンに比べて多少良い
- ExadataはIOの限界に達するとスループットが伸び悩む
- 同時実行性能が求められるほどIn-Memoryのほうが性能が良い
- 単体ユーザーでの結果は複数ユーザーでの結果と異なる



# 複数ユーザー・パフォーマンス・レース 延長戦

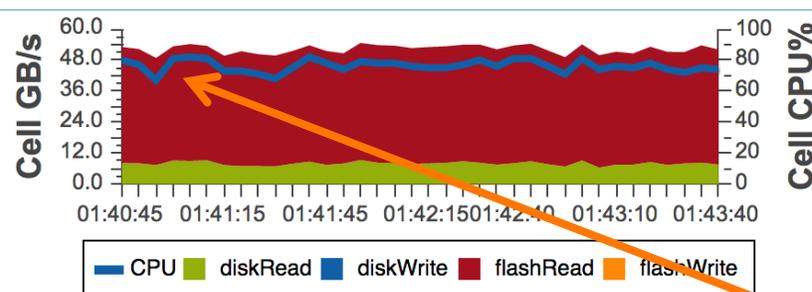
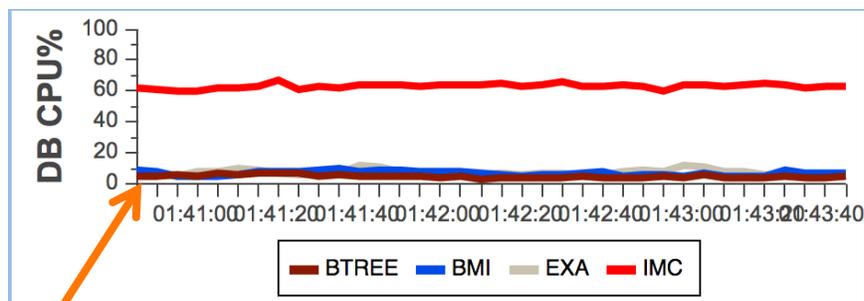
- クエリーのほとんどはIOネックだった
- Hybrid Columnar Compression を使い、  
正しくパーティションを設計し、  
IO量を減らせば  
Exadataのスループットも良くなるはず



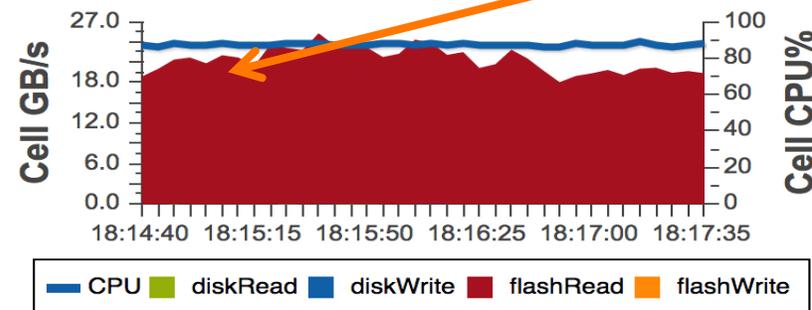
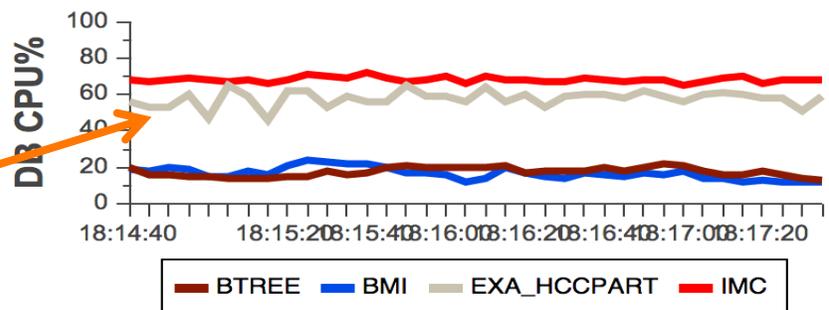
# 複数ユーザー・パフォーマンス・レース 延長戦

## 32同時ユーザー (DoP=2) + パーティション + HCC

Exadata: パーティション + HCC 未使用



Exadata: パーティション + HCC 使用



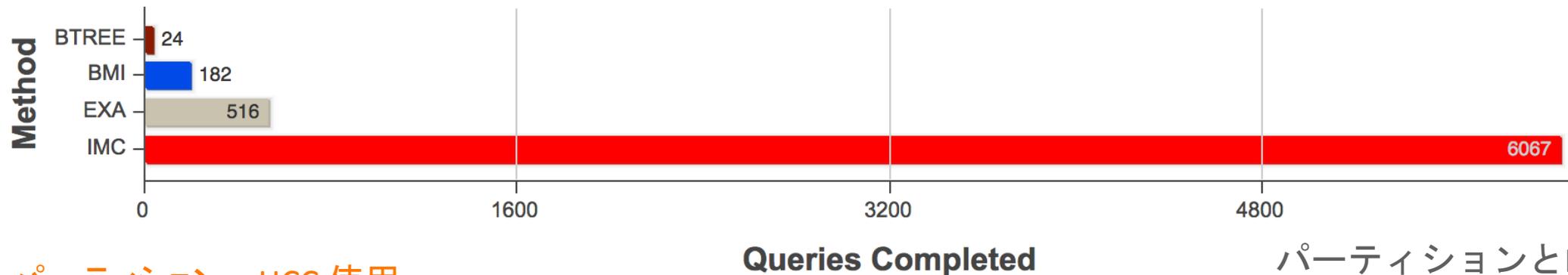
CPU使用率が  
上昇

IO量が減少

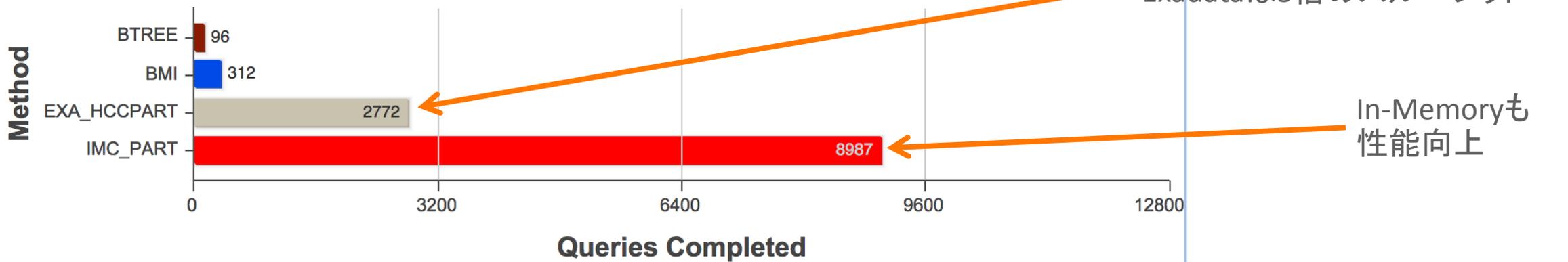
# 複数ユーザー・パフォーマンス・レース 延長戦

32同時ユーザー (DoP=2) + パーティション (Exadata / In-Memory) + HCC (Exadata)

パーティション + HCC 未使用



パーティション + HCC 使用



# アジェンダ

- 1 パフォーマンス・レース！
- 2 データウェアハウス・デザインのルールとフレームワーク
- 3 実行方式を選ぼう
- 4 複数ユーザー・パフォーマンス・レース！
- 5 プラットフォームを選ぼう
- 6 ハイパフォーマンスを実現するために一番大事なこと

# プラットフォームを選ぼう: スター型変換



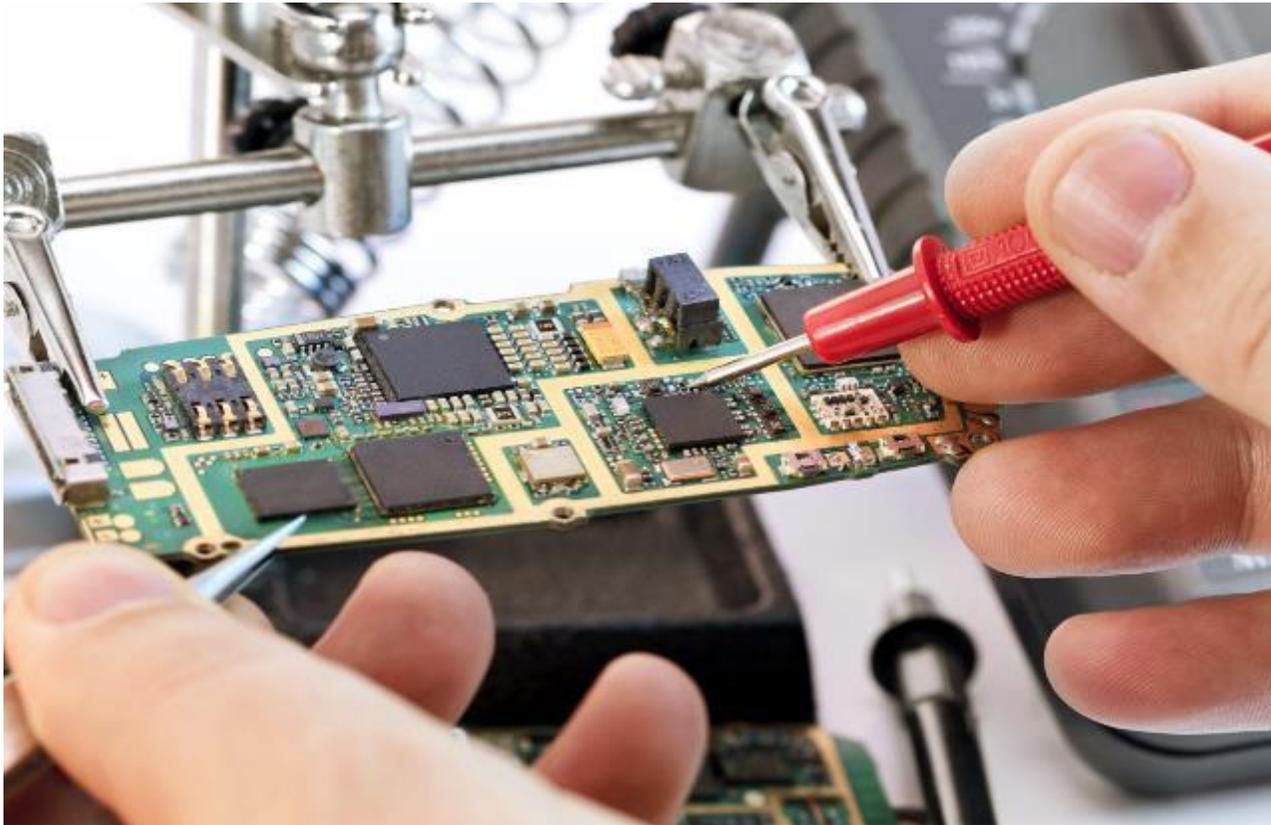
- ファクト表から  
 少しの行を抽出する
- バッファ・キャッシュに  
 収まる程度に  
 索引が十分に小さい
- 同時実行数が多い

# プラットフォームを選ぼう: Exadata表スキャン



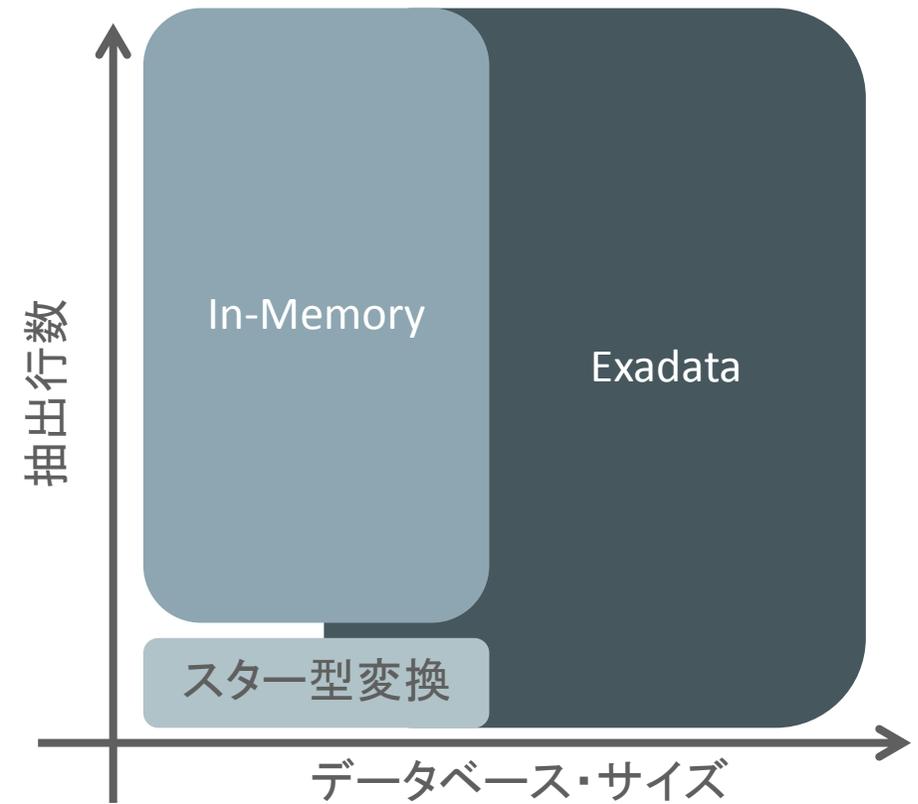
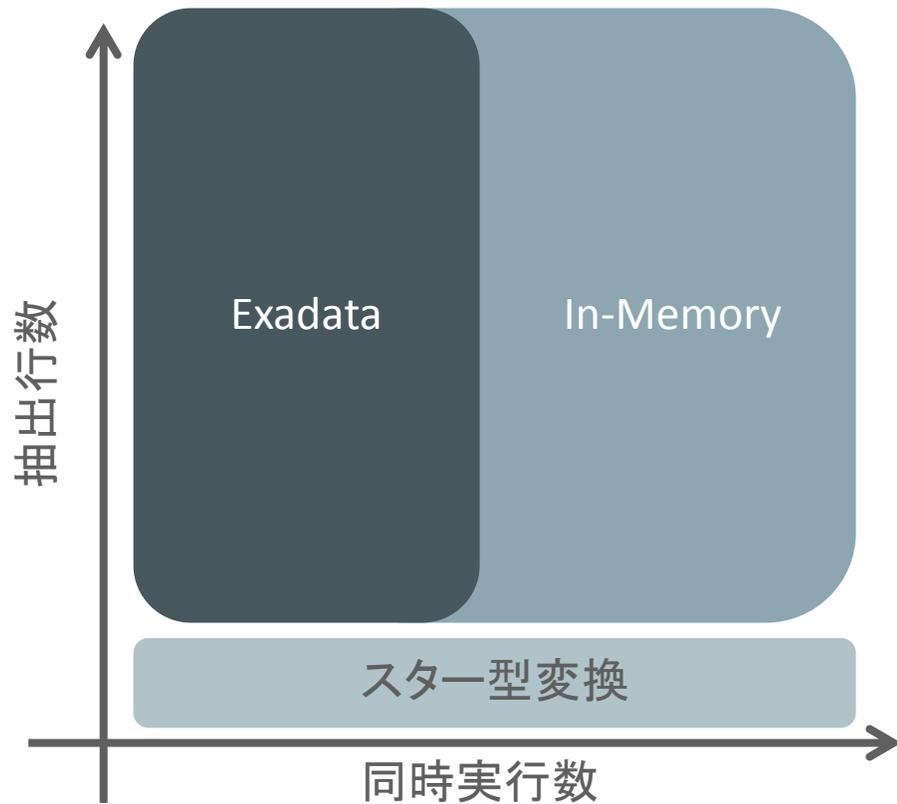
- ファクト表から  
たくさんの行を抽出する
- データベース・サイズが  
大きい
- 同時実行数が少ない

# プラットフォームを選ぼう: In-Memory



- ファクト表から  
たくさんの行を抽出する
- メモリーに乗る程度の  
データベース・サイズ
- 同時実行数が多い

# プラットフォームを選ぼう: まとめ



# アジェンダ

- 1 パフォーマンス・レース！
- 2 データウェアハウス・デザインのルールとフレームワーク
- 3 実行方式を選ぼう
- 4 複数ユーザー・パフォーマンス・レース！
- 5 プラットフォームを選ぼう
- 6 ハイパフォーマンスを実現するために一番大事なこと

# アーキテクトになろう！

- アドミニストレータから  
アーキテクトへの意識改革
- アーキテクチャを決めるのは  
あなた！



# アーキテクチャを変えよう！

- 根本原因の解決には  
“変化”が求められる
- “変化”することは怖い
- アーキテクトとして  
アーキテクチャを**変えよう**！



# Oracle Database 12c おすすめ研修コース

## Oracle Database 12c: SQL チューニング ワークショップ

|       |                                                                                                                                                                 |                                                                                                                                                                             |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 概要    | このコースでは、OracleのSQL文のチューニングや、Oracle Databaseに合わせて適切にチューニングされたSQL文を記述する方法を説明します。SQLトレース機能の使い方、実行計画の取得方法、オプティマイザ機能の活用方法などを、実機演習を通して習得することができます。                    |                                                                                                                                                                             |
| 学習項目  | <ul style="list-style-type: none"><li>■ Database Vaultの概要</li><li>■ Database Vaultの構成</li><li>■ 権限の分析 (12c 新機能)</li><li>■ レルムの構成</li><li>■ ルール・セットの定義</li></ul> | <ul style="list-style-type: none"><li>■ コマンド・ルールの構成</li><li>■ ルール・セットの拡張</li><li>■ セキュア・アプリケーション・ロールの構成</li><li>■ Database Vaultレポートによる監査</li><li>■ ベスト・プラクティスの実装</li></ul> |
| コース日数 | 3 日間 【トレーニングキャンパス赤坂】 2014/12/3-5                                                                                                                                |                                                                                                                                                                             |

## Oracle Database 12c: パフォーマンス・チューニング

|       |                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                 |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 概要    | このコースでは、OracleのSQL文のチューニングや、Oracle Databaseに合わせて適切にチューニングされたSQL文を記述する方法を説明します。SQLトレース機能の使い方、実行計画の取得方法、オプティマイザ機能の活用方法などを、実機演習を通して習得することができます。                                                                                                                                          |                                                                                                                                                                                                                                                                                 |
| 学習項目  | <ul style="list-style-type: none"><li>■ 基本チューニング診断</li><li>■ 自動ワークロード・リポジトリの使用</li><li>■ パフォーマンス問題の範囲の定義</li><li>■ メトリックとアラートの使用</li><li>■ ベースラインの使用</li><li>■ AWRベースのツールの使用</li><li>■ リアルタイム・データベース操作監視</li><li>■ アプリケーションの監視</li><li>■ 問題のあるSQL文の識別</li><li>■ オプティマイザへの影響</li></ul> | <ul style="list-style-type: none"><li>■ SQL操作のコストの削減</li><li>■ SQLパフォーマンス・アナライザの使用</li><li>■ SQLパフォーマンスの管理</li><li>■ データベース・リプレイの使用</li><li>■ 共有プールのチューニング</li><li>■ バッファ・キャッシュのチューニング</li><li>■ PGAおよび一時領域のチューニング</li><li>■ 自動メモリー管理の使用</li><li>■ パフォーマンス・チューニングのまとめ</li></ul> |
| コース日数 | 5 日間 【トレーニングキャンパス赤坂】 2015/1/19-23                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                 |

詳細は [Oracle University Webサイト](#) にてご確認ください。

# **Hardware and Software Engineered to Work Together**

**VISION 2020**

**#1 CLOUD**

**ORACLE JAPAN**

ORACLE®