心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

料金系基幹システムへのPostgreSQL導入事例

～成功までの道のり～

2015年9月11日

NTTコムウェア株式会社　朝倉 佑貴
NTT OSSセンタ　山田 達朗

目次

各社のご紹介

NTTグループ全体で「TCO削減」のためOSSを積極活用

事業会社・SIer・NTT OSSセンタが連携して、OSSを活用したシステムを開発

NTTコミュニケーションズ

「Global ICT Partner」として、最新のテクノロジーと安全で信頼性の高いICTサービスを提供

- Arcstar Universal One (VPNサービス)
- OCNモバイル ONE for Business (モバイル通信サービス)
- Arcstar IP Voice (IP電話サービス)

など、法人のお客さまにサービスを展開。
また、個人のお客さまにも多くのサービスを展開

協力

NTTコムウェア

プロジェクト
- 開発
- 運用、保守

支援

品質生産性技術本部
- 開発支援
- ノウハウ展開

ミッションクリティカルシステムの更改実績をベースに、高品質なICT基盤を提供・運用

相互連携

NTT OSSセンタ

OSS活用によるNTTグループ全体のシステムのTCO削減を目的に活動

開発連携 → OSSコミュニティ

連携 → 研究所 等

NTT

心をつなぐ、社会をつなぐ NTTコムウェア

Innovative R&D by NTT

NTTコムウェアにおけるOSSへの取り組み

NTTの通信ネットワークや、顧客サービス業務を支えてきた技術力を基に、
NTTグループ内外のお客様へのSIerとして、幅広く活動

大規模・高信頼領域への適用を視野に、2000年代初めから、PostgreSQLを
はじめとする、OSSへの取り組みを積極的に推進

2000年　Linuxセンタ設立
　　Linuxに関するサポート開始

2004年　PostgreSQLのサポート開始(PostgreSQL 7.4)

2010年　NTT事業会社 中規模注文管理システムへ
　　PostgreSQL含むOSSを全面導入

2013年　NTT事業会社
　　大規模履歴管理システムへのPostgreSQL導入

ご紹介事例

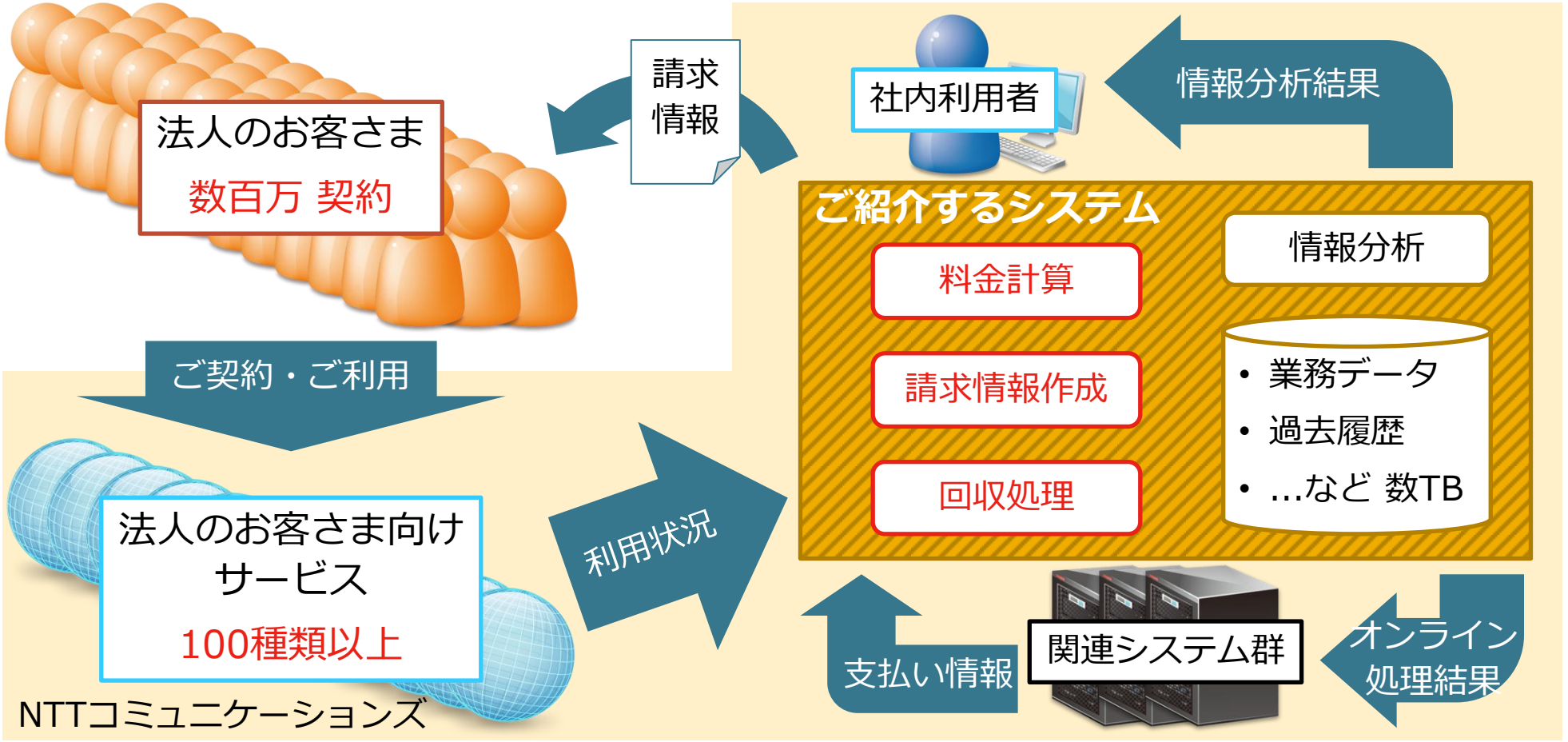
2015年　NTTコミュニケーションズ
　　料金系基幹システムへのPostgreSQL導入

心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

ご紹介するシステムの概要

NTTコミュニケーションズにおいて、経営基盤ともなる料金系の基幹システム

主に数百万契約の法人のお客さまが利用する100種類以上のサービスに関して、料金計算や請求情報作成などを行い、多様な情報分析も行う

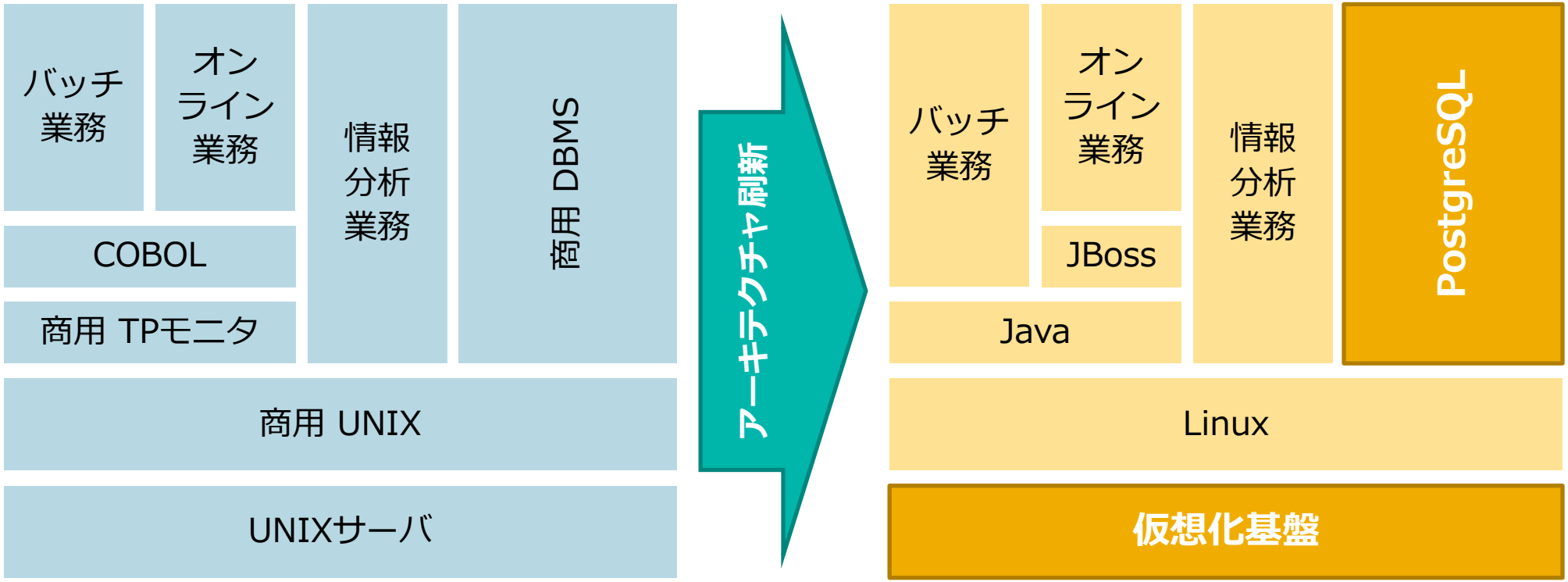


今回の開発内容

商用運用されているレガシーシステムのアーキテクチャを刷新

TCO削減と今後の適用領域拡大に向けた礎として、戦略的にPostgreSQLを採用

契約数の伸びに応じた拡張性が必要なため、NTT Comの仮想化基盤を採用

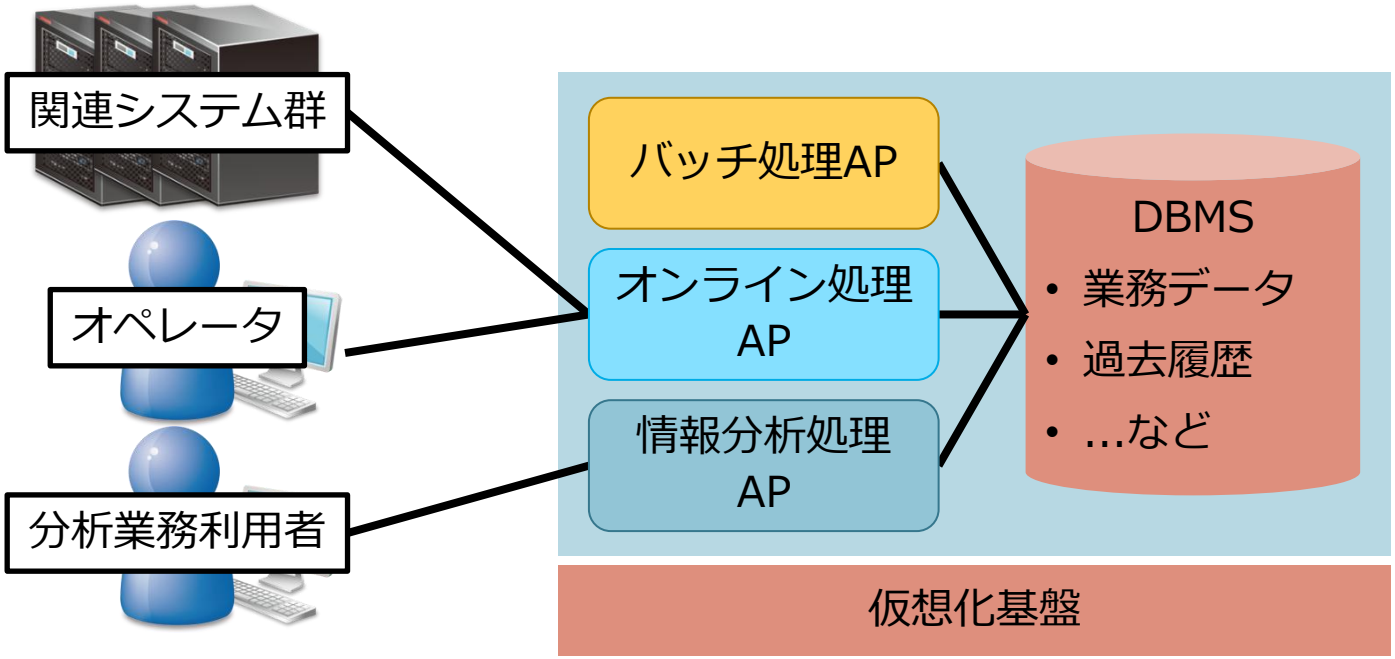


AP構造も刷新、テーブル構成や処理ロジックなど含め完全な作り直しを行った

開発プロジェクトの最大のミッション

最大のミッション、それは**「性能の担保」**

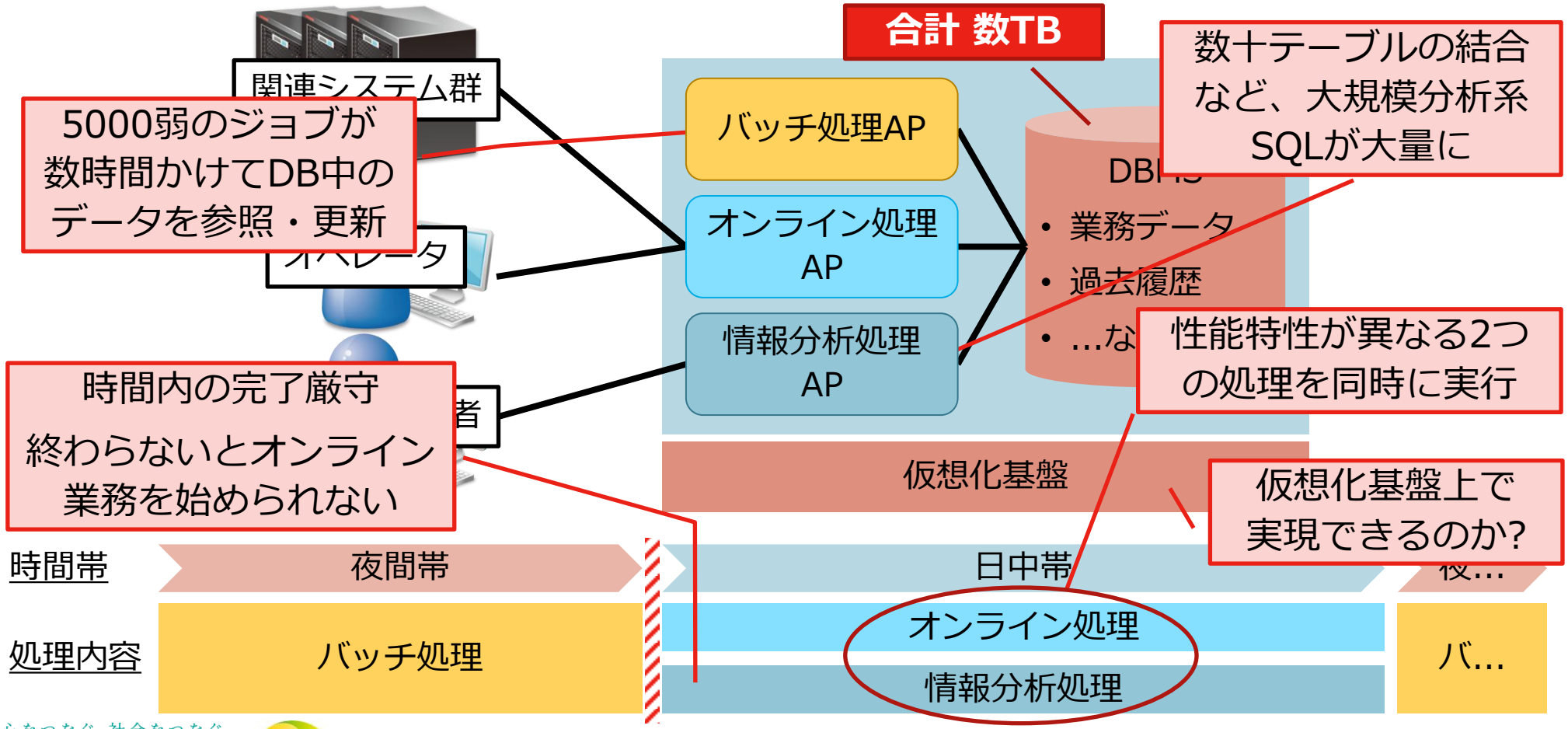
- 大量データの計算処理を確実に時間内で完了しつつ、処理時間の安定化も図れ
- 大規模な分析処理を行いつつ、関連システムからのオンライン処理をさばけ



開発プロジェクトの最大のミッション

最大のミッション、それは**「性能の担保」**

- 大量データの計算処理を確実に時間内で完了しつつ、処理時間の安定化も図れ
- 大規模な分析処理を行いつつ、関連システムからのオンライン処理をさばけ



心をつなぐ、社会をつなぐ NTTコムウェア

Innovative R&D by NTT

本日のおはなし

最大のミッション**「性能の担保」**の達成に向け

全力で戦い抜いた壮絶(?)なストーリー

性能特性の異なる業務が

互いに影響しないよう

設計せよ

運用に耐えうるバッチ処理

時間を担保せよ

- 仮想化基盤上で性能要件を満たせ
- PostgreSQLのSQL実行時間を
  コントロールせよ

本日のおはなし

最大のミッション**「性能の担保」**の達成に向け

全力で戦い抜いた壮絶(?)なストーリー

性能特性の異なる業務が

互いに影響しないよう

設計せよ

運用に耐えうるバッチ処理

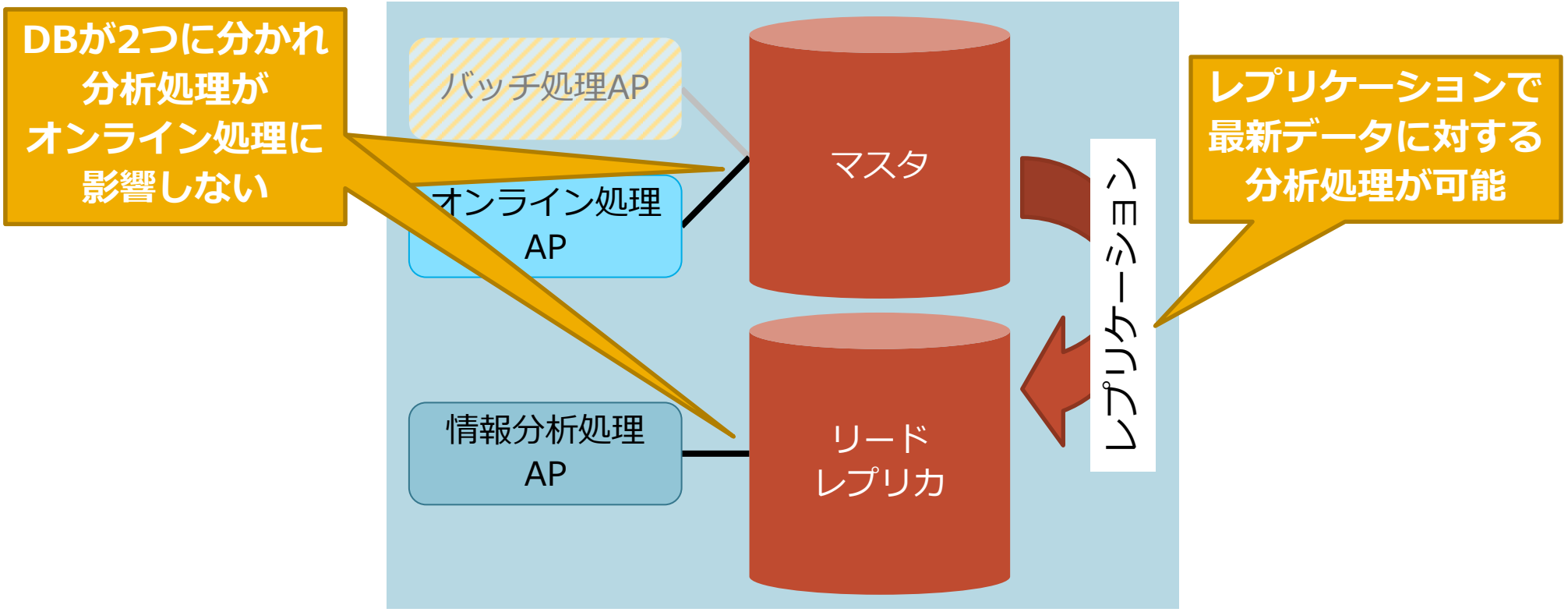
時間を担保せよ

- 仮想化基盤上で性能要件を満たせ
- PostgreSQLのSQL実行時間を
  コントロールせよ

性能特性の異なる業務が互いに影響しないよう設計せよ

常に**最新のデータ**に対して**大規模分析処理**を行いたい
でも、**オンライン処理**の性能に**影響を与えたくない**

ストリーミングレプリケーションで分析処理用のリードレプリカを構築



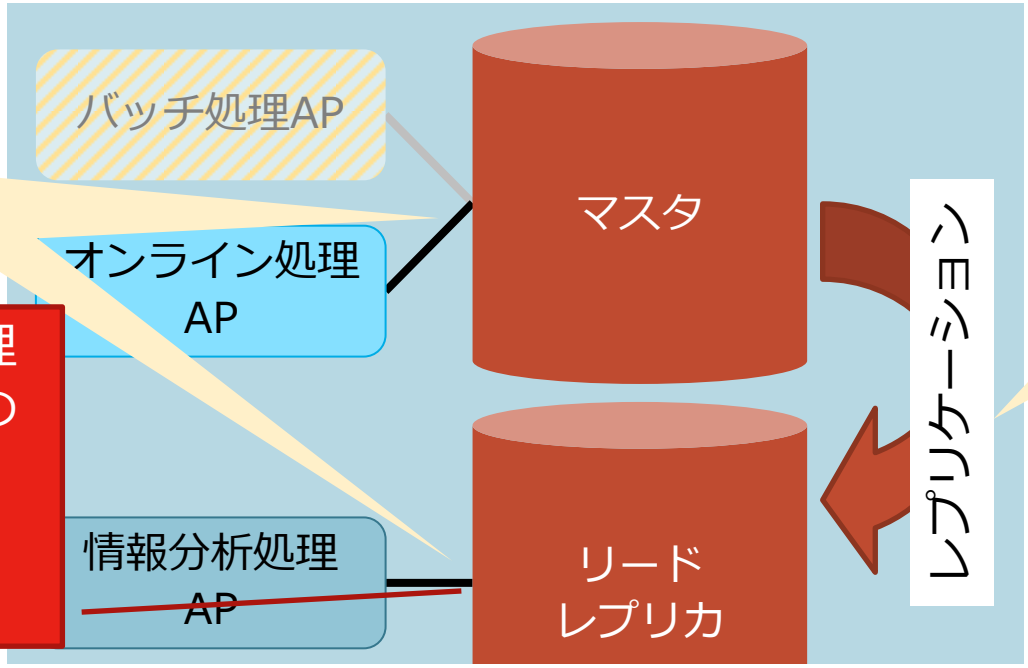
心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

性能特性の異なる業務が互いに影響しないよう設計せよ

常に**最新のデータ**に対して**大規模分析処理**を行いたい
でも、**オンライン処理**の性能に**影響を与えたくない**

ストリーミングレプリケーションで分析処理用のリードレプリカを構築

DBが2つに分かれ
分析処理が
オンライン処理に
影響しない



レプリケーションで
最新データに対する
分析処理が可能

レプリケーション処理
によって、分析処理の
SQLが実行中に中断
されないよう設定(*)
する必要あり

(*) max_standby_streaming_delayを-1とした。デフォルトでは、WALの再生に30秒以上かかるとレプリカ側のSQLが中断される。

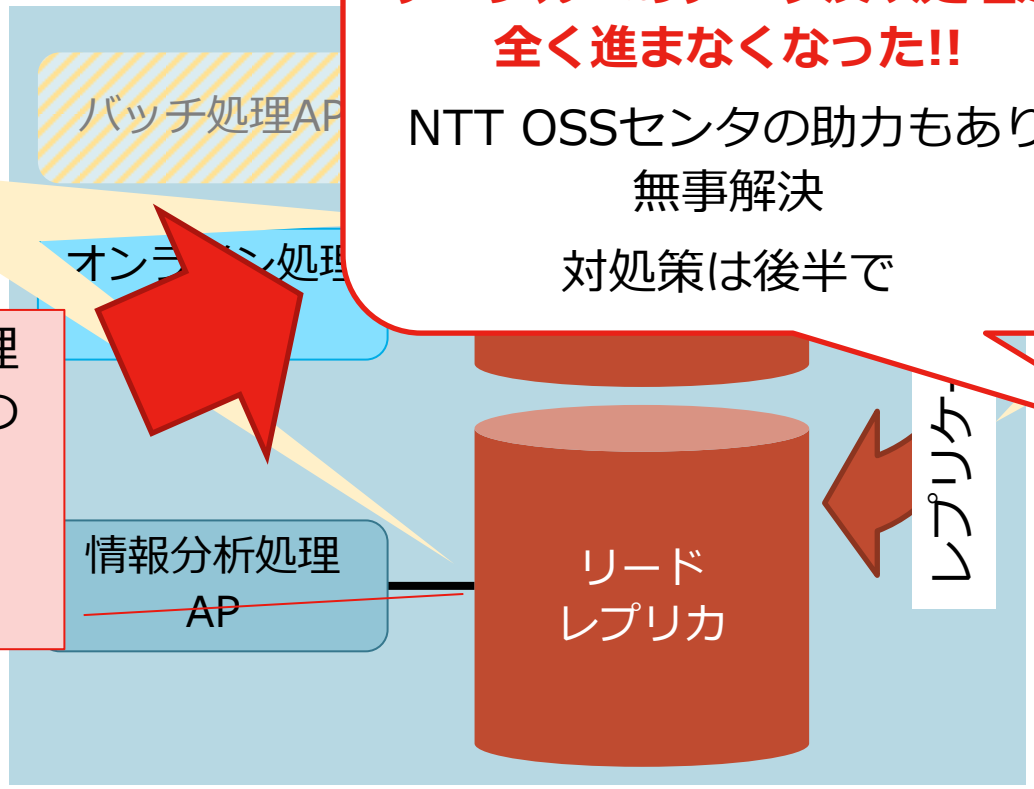
性能特性の異なる業務が互いに影響しないよう設計せよ

常に**最新のデータ**に対して**大規模分析処理**を行いたい
でも、**オンライン処理**の性能に**影響を与えたくない**

ストリーミングレプリケーシ　　**プリカを構築**

DBが2つに分かれ
分析処理が
オンライン処理に
影響しない

レプリケーション処理
によって、分析処理の
SQLが実行中に中断
されないよう設定(*)
する必要あり



レプリカ側で
テーブルへのデータ反映処理が
全く進まなくなった!!

NTT OSSセンタの助力もあり
無事解決

対処策は後半で

プリケーションで
新データに対する
分析処理が可能

心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

本日のおはなし

最大のミッション**「性能の担保」**の達成に向け

全力で戦い抜いた壮絶(?)なストーリー

性能特性の異なる業務が
互いに影響しないよう
設計せよ

運用に耐えうるバッチ処理
時間を担保せよ

- 仮想化基盤上で性能要件を満たせ
- PostgreSQLのSQL実行時間を
  コントロールせよ

運用に耐えうるバッチ処理時間を担保せよ
- 仮想化基盤上で性能要件を満たせ ～実機検証の必要性

PostgreSQL × 仮想化基盤 の採用には
性能面の懸念・課題があった

数十GBの表同士を結合・集計するSQLが多重で走るけど大丈夫?

広範囲のデータにアクセスするけど大丈夫?

マシンリソースを限界まで使い切って処理時間を縮めたいがどうしたらいい?

・・・

・・・

・・・

・・・

これほど大きなバッチ系システムをPostgreSQLと仮想化環境で実現するのは**初めて**

実機検証に基づいたハードウェアのサイジングを実施

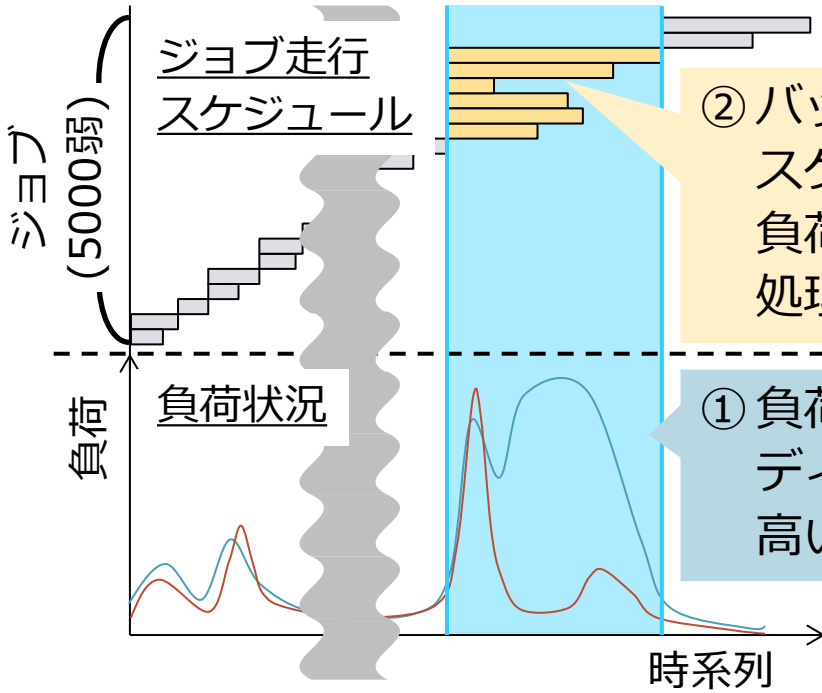
心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

運用に耐えうるバッチ処理時間を担保せよ
- 仮想化基盤上で性能要件を満たせ ～実機検証の内容

5000弱にもおよぶジョブ(*)のうち、特にCPUとディスクI/O負荷が高い6つのジョブに絞って、実機検証を効率化

(*) ジョブ: バッチ処理を構成するプログラムの一単位。多数のSQLで構成されている。

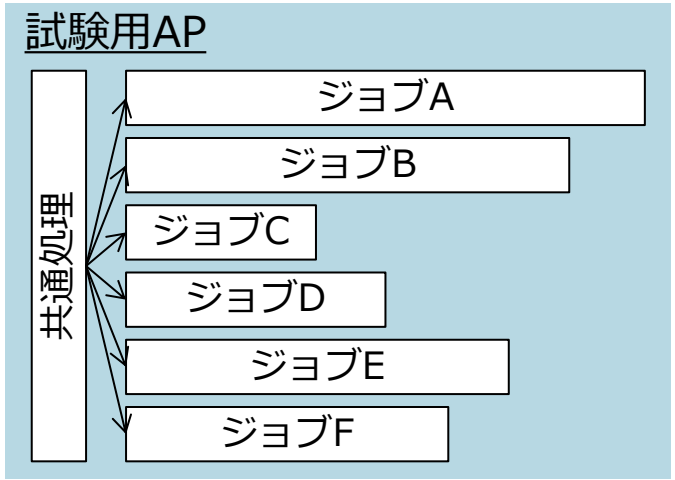
既存システムの負荷状況を詳細分析



② バッチジョブの走行スケジュールから負荷が高い時間帯の処理を抽出

① 負荷状況からCPUやディスクI/O負荷が高い時間帯を確認

試験に必要な最小限の6つの処理に絞って試験用APを作成



運用に耐えうるバッチ処理時間を担保せよ
- 仮想化基盤上で性能要件を満たせ ～実機検証の結果

- 実機検証の結果、特にディスクI/O リソースが多く必要と判明

WAL出力量削減など、今後のPostgreSQLの進化に期待

- 必要なリソース量が定量的に分かり、NTT Comで確保してもらえた

NTT Comには、マシンリソース確保に尽力してもらった

- PostgreSQL × 仮想化基盤において、十分な性能を得られる環境が整った

心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

本日のおはなし

最大のミッション**「性能の担保」**の達成に向け

全力で戦い抜いた壮絶(?)なストーリー

性能特性の異なる業務が
互いに影響しないよう
設計せよ

運用に耐えうるバッチ処理
時間を担保せよ

- 仮想化基盤上で性能要件を満たせ
- PostgreSQLのSQL実行時間を
  コントロールせよ

運用に耐えうるバッチ処理時間を担保せよ
- PostgreSQLのSQL実行時間をコントロールせよ ～バッチ性能への要求

運用に耐えうるバッチ性能への要求は2つ

1 時間内で確実にバッチ処理を完了せよ

朝までにバッチ処理を確実に完了しなければ、オンライン業務は開始できない。

5000弱の膨大なジョブを抱える本システムでは、たった一つのジョブの性能遅延が、命取りとなる。

2 バッチ性能の将来予測を可能にせよ

商品契約数の増加によって、バッチ処理時間も増加し続けるため、将来的にはマシンリソースの増強が必要になる。リソース増強時期を見極めるためには、安定した性能傾向が必要であり、バッチ処理時間をコントロールする必要がある。

バッチ性能の「コントロール」が命題

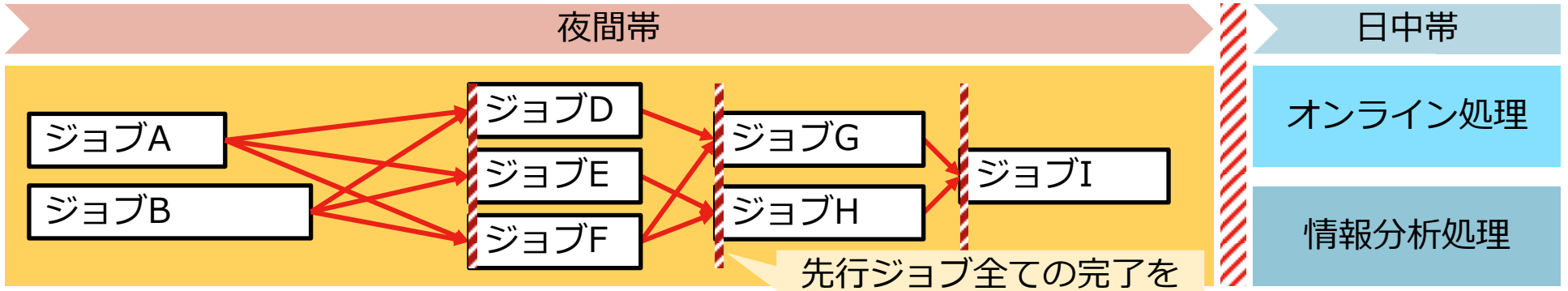
DB処理中心の本システムでは、SQL実行時間のコントロールが必須

運用に耐えうるバッチ処理時間を担保せよ
- PostgreSQLのSQL実行時間をコントロールせよ ～時間内に完了せよ

1 時間内で確実にバッチ処理を完了せよ

大規模バッチシステムの特徴は、多数の待ち合わせジョブが存在すること。

たった一つのSQL処理遅延により、後続ジョブのスタートが遅れ、バッチ処理時間オーバーに直結し、日中帯の処理を開始できなくなる。

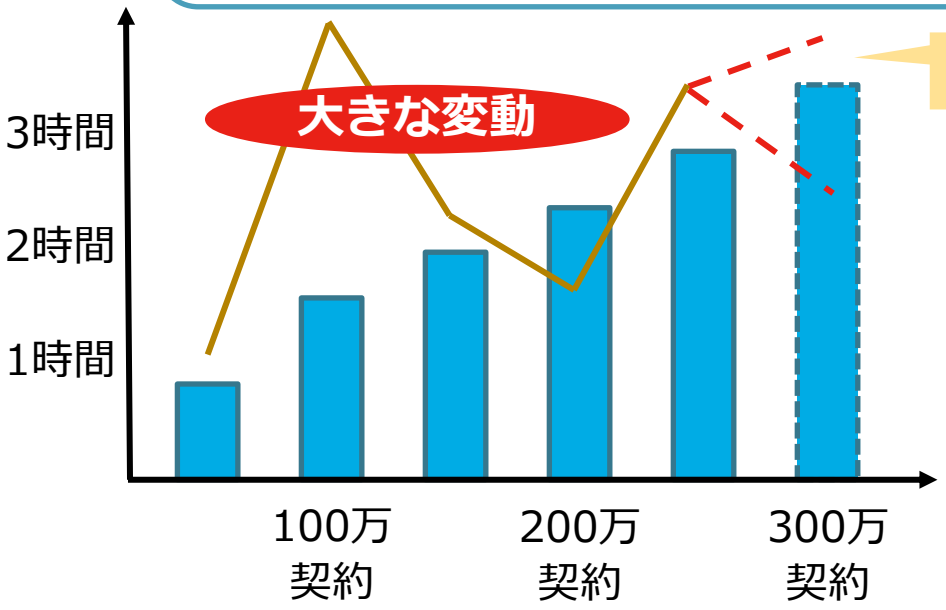


もし、処理遅延が発生したら…

運用に耐えうるバッチ処理時間を担保せよ
- PostgreSQLのSQL実行時間をコントロールせよ ～将来予測を可能にせよ

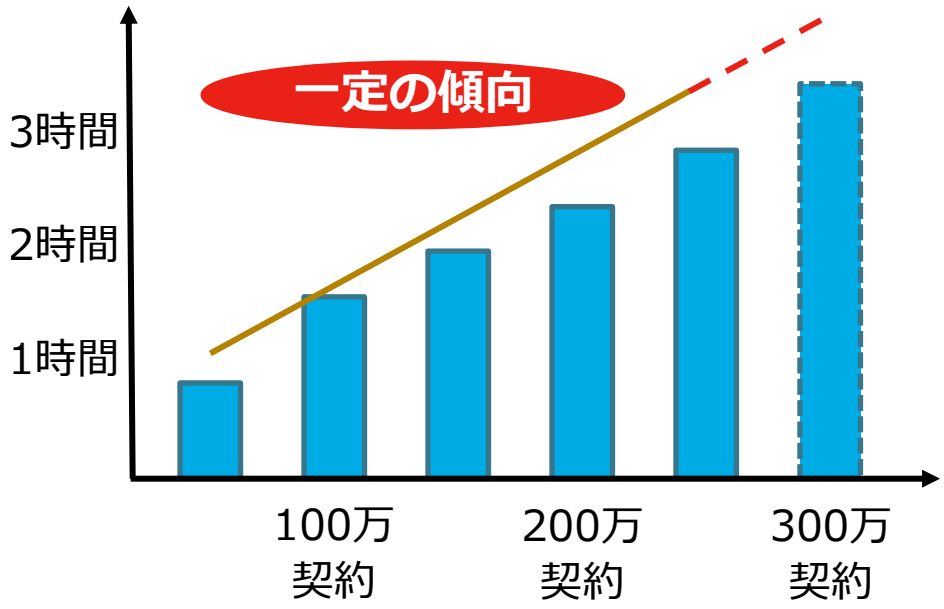
2 バッチ性能の将来予測を可能にせよ

商品契約数は常に一定量に増加。しかし、バッチ処理時間が毎回大きく異なれば、将来のバッチ処理時間予測が困難になる。バッチ処理時間の変動をコントロールし、一定の傾向を持たせ、将来予測を可能にする方法を模索した。



300万契約での処理時間が予測困難

300万契約での処理時間が予測可能！

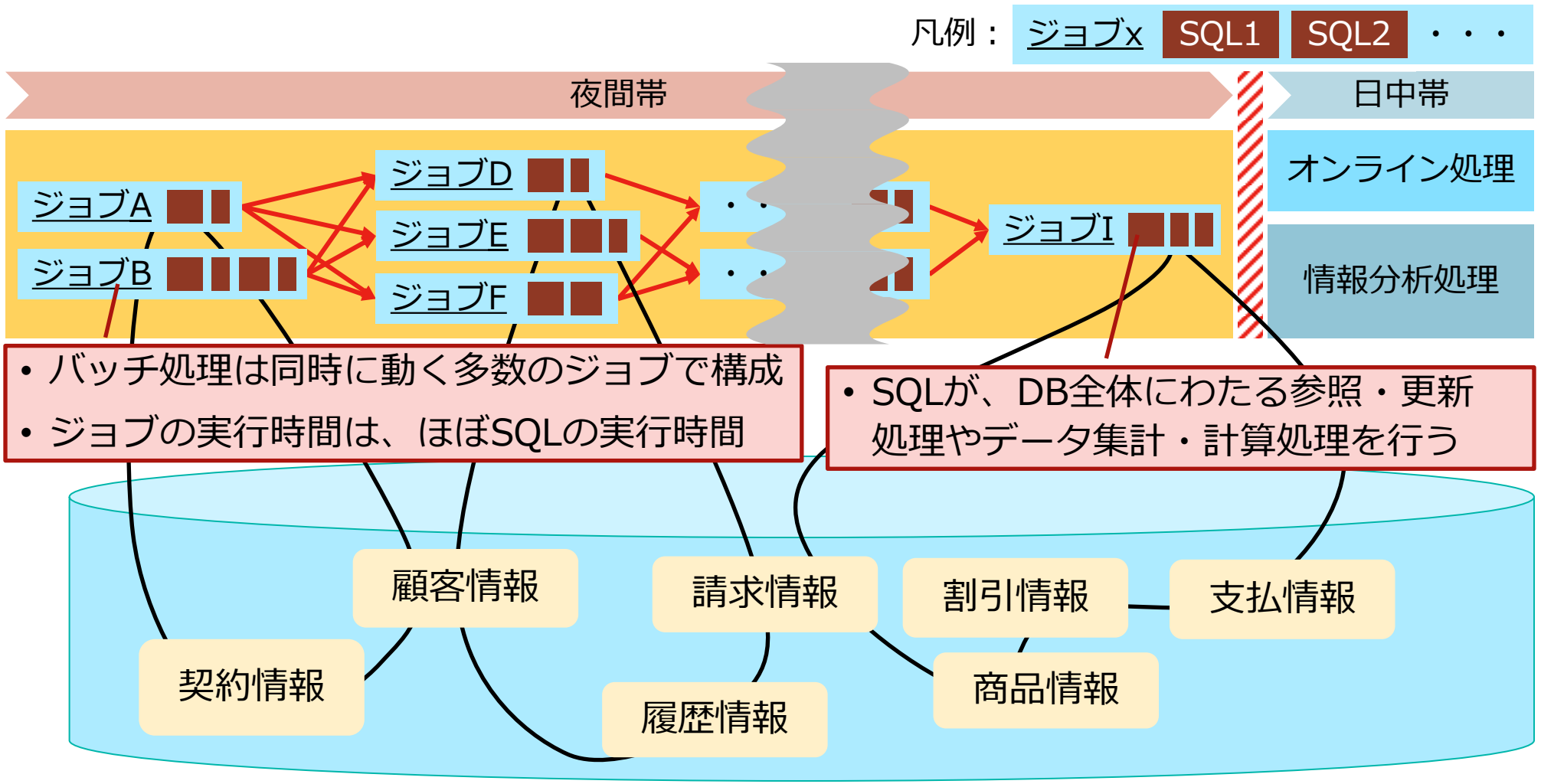


バッチ処理時間をコントロール

心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

運用に耐えうるバッチ処理時間を担保せよ
- PostgreSQLのSQL実行時間をコントロールせよ ～SQL実行時間の重要性

DB全体にわたる参照・更新処理はもちろんのこと、データの集計や計算処理もSQLによって行う。これらSQLの実行時間がバッチ処理性能の鍵を握っていた。

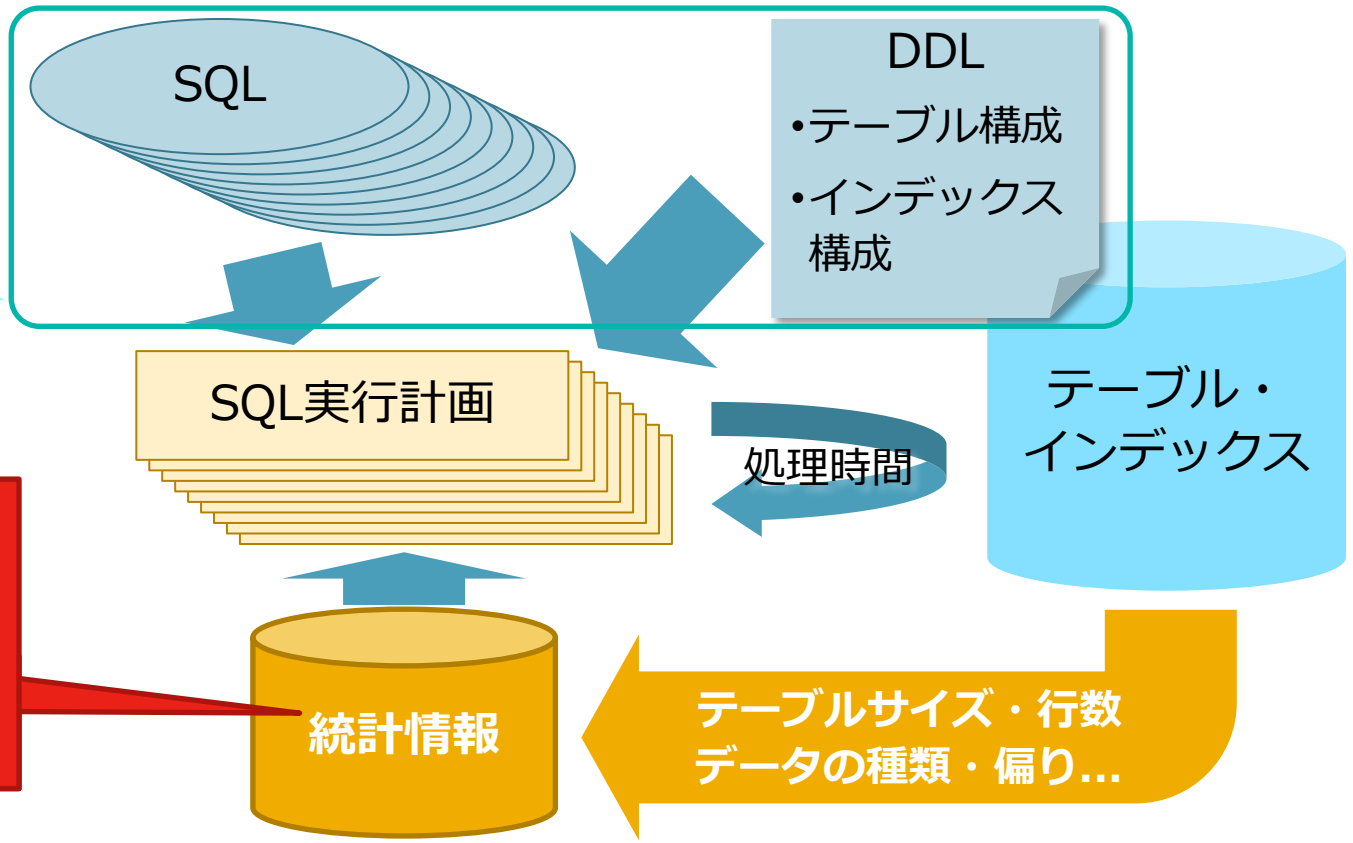


運用に耐えうるバッチ処理時間を担保せよ
- PostgreSQLのSQL実行時間をコントロールせよ ～SQL実行時間の制御

テーブルやインデックスの構成、SQLそのものに問題がなくても
完全には**SQL実行時間をコントロール**できない

SQL実行計画や、その元となる**統計情報**までもコントロールする必要がある

SQLやDDLは、
レビューや試験で
問題ないことを確認済み



作戦1: 統計情報を制御する

最新の状態が反映された
統計情報であることを
保証

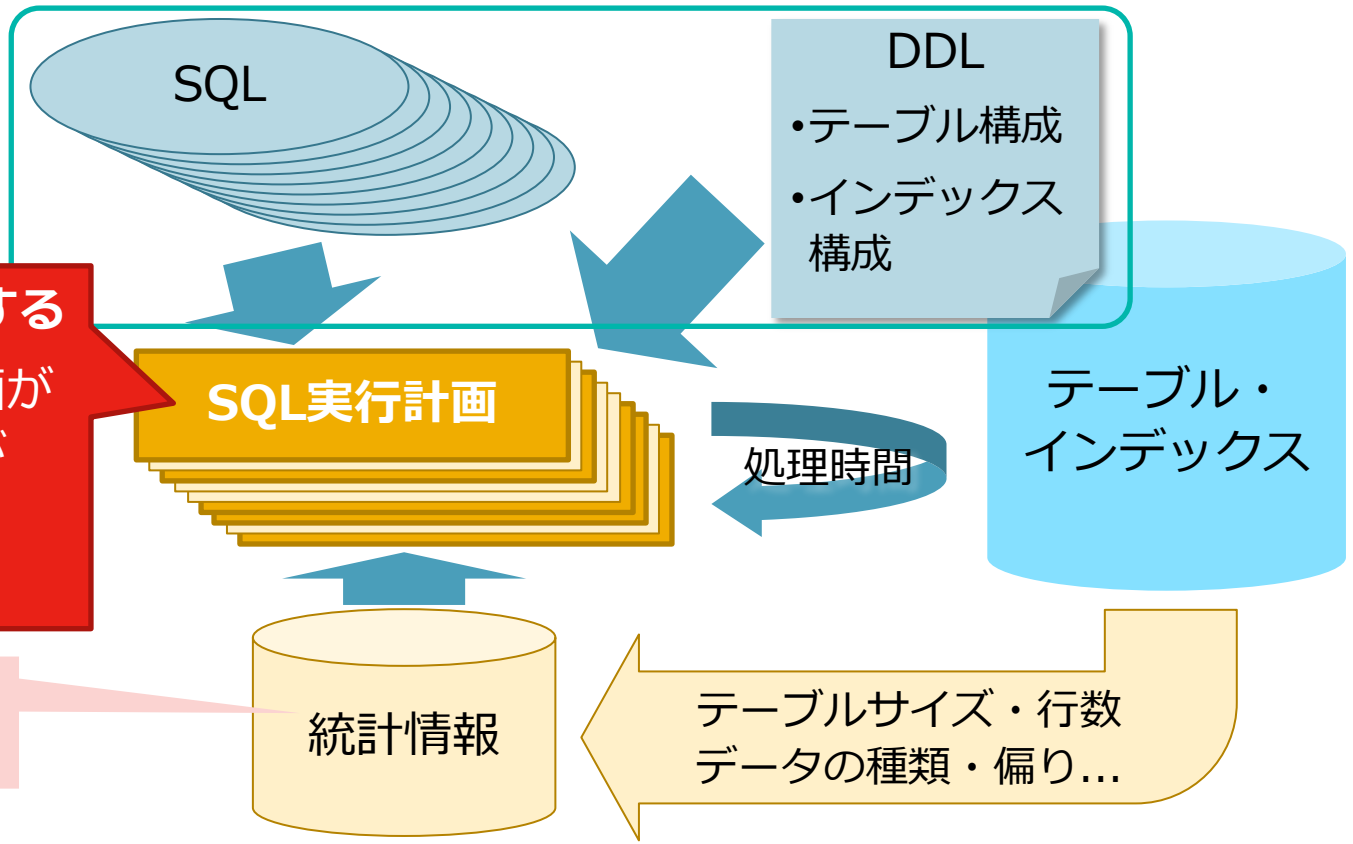
心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

運用に耐えうるバッチ処理時間を担保せよ
- PostgreSQLのSQL実行時間をコントロールせよ ～SQL実行時間の制御

テーブルやインデックスの構成、SQLそのものに問題がなくても
完全には**SQL実行時間をコントロール**できない

SQL実行計画や、その元となる**統計情報**までもコントロールする必要がある

SQLやDDLは、
レビューや試験で
問題ないことを確認済み



作戦2: SQL実行計画を制御する

実行するたびにSQL実行計画が
異なり、処理時間の変動が
大きなものについて、
SQL実行計画を固定化

作戦1: 統計情報を制御する

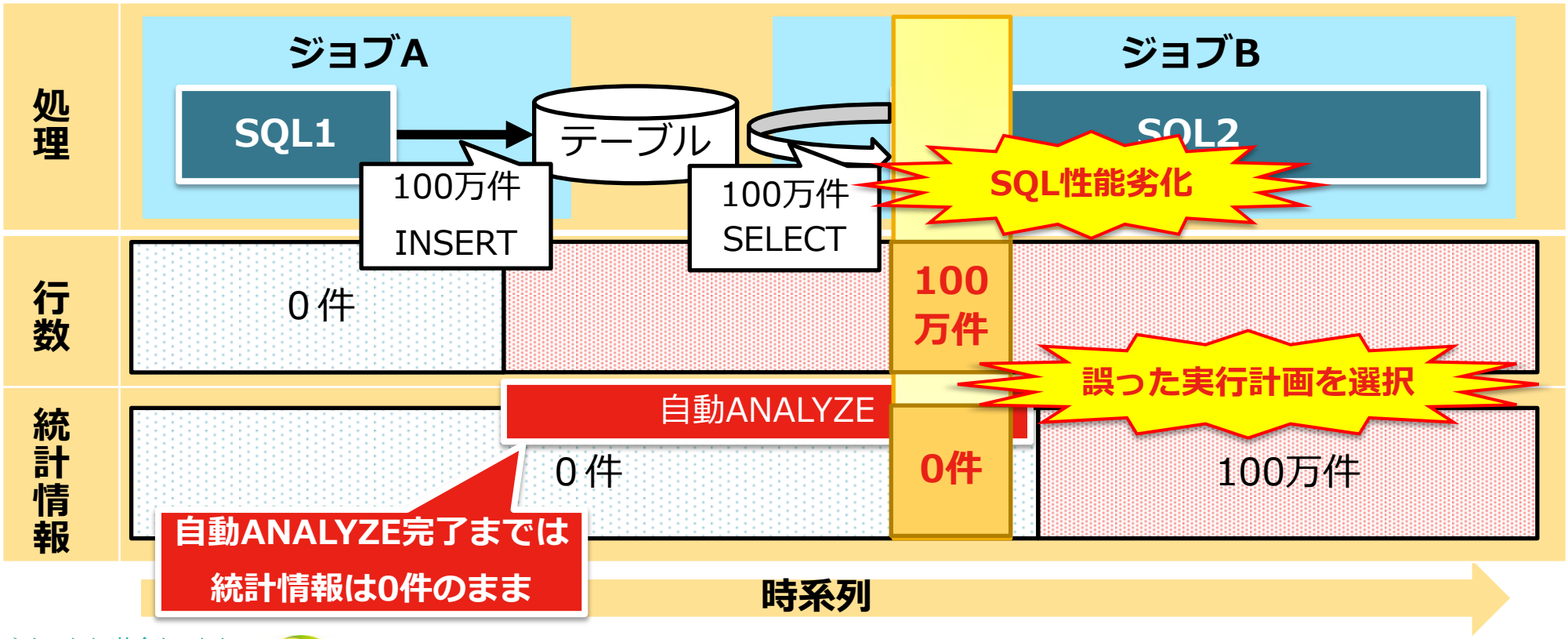
運用に耐えうるバッチ処理時間を担保せよ
- PostgreSQLのSQL実行時間をコントロールせよ ～統計情報の制御

作戦1

自動ANALYZEに頼らないことで、統計情報をコントロール

PostgreSQLが実行する自動ANALYZEでは、適切なタイミングで統計情報が収集されず、実データと統計情報にずれが発生した。

このため、正しいSQL実行計画が選択されなかった。



心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

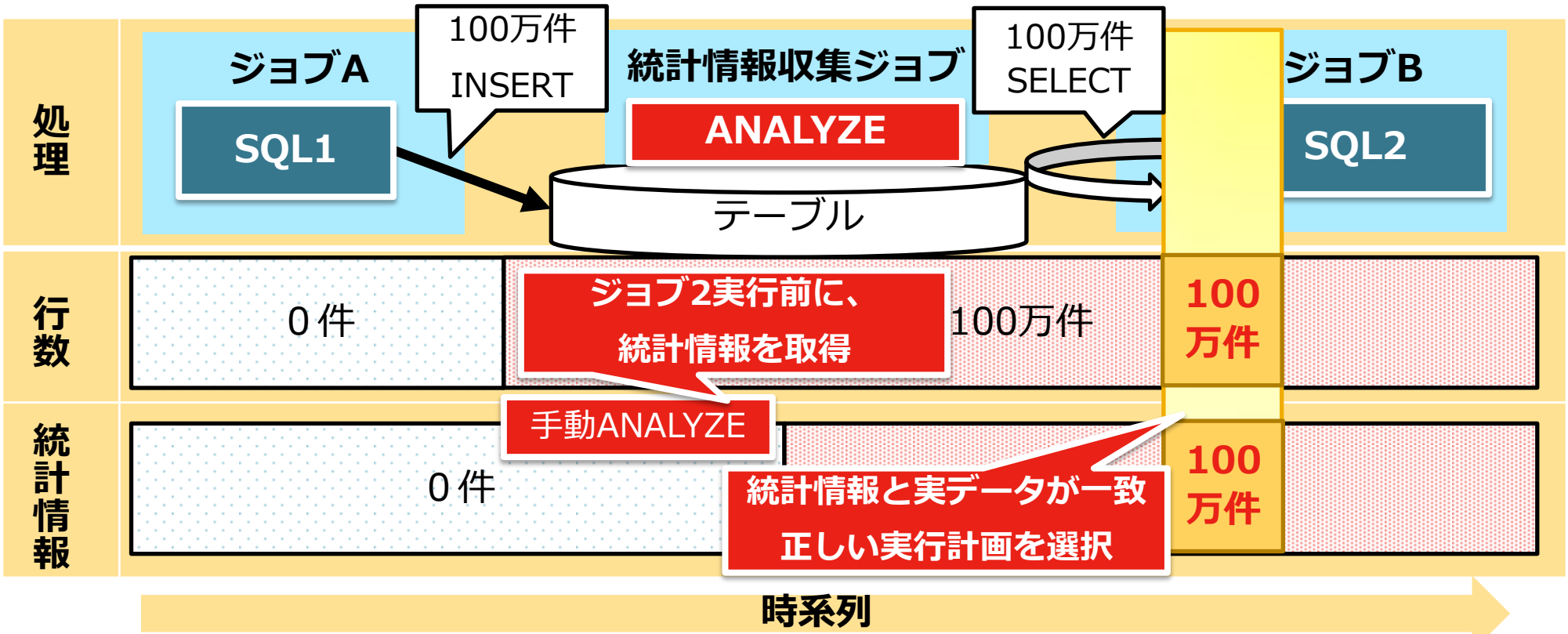
運用に耐えうるバッチ処理時間を担保せよ
- PostgreSQLのSQL実行時間をコントロールせよ ～統計情報の制御

作戦1

自動ANALYZEに頼らないことで、統計情報をコントロール

正しい実行計画を選択させるために、自動ANALYZEの使用をやめ、バッチ中に、適切なタイミングで統計情報を収集するための「手動ANALYZE」ジョブを作成。

正しい実行計画が選択されるように、統計情報をコントロールした。

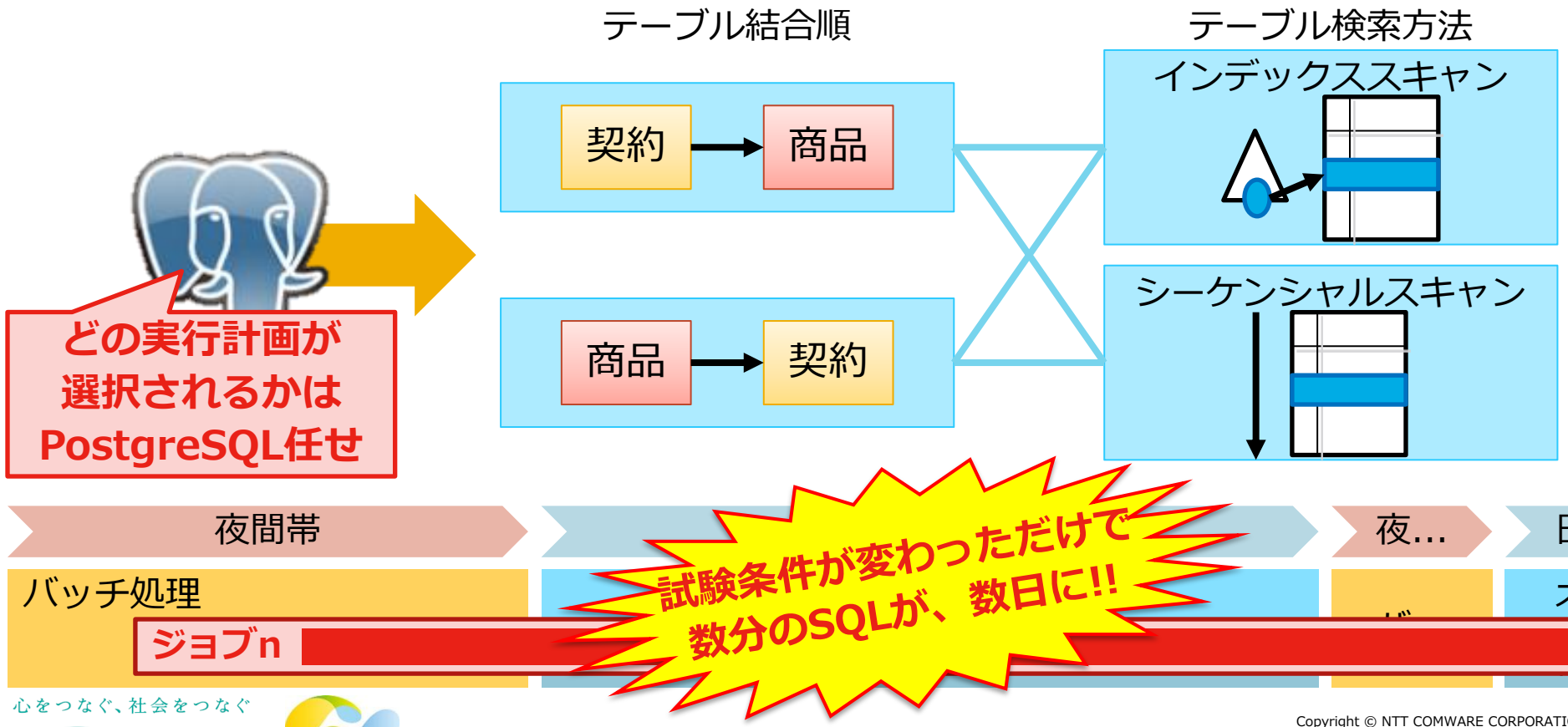


運用に耐えうるバッチ処理時間を担保せよ
- PostgreSQLのSQL実行時間をコントロールせよ ～SQL実行計画の制御

作戦2

pg_hint_planにより、SQL実行計画をコントロール

pg_hint_planは、SQLヒント句により、PostgreSQLのSQL実行計画を思うままに、コントロールするツール。特に処理時間の変動が大きかったSQLについて、pg_hint_planによって実行計画を固定し、処理時間を安定させた。



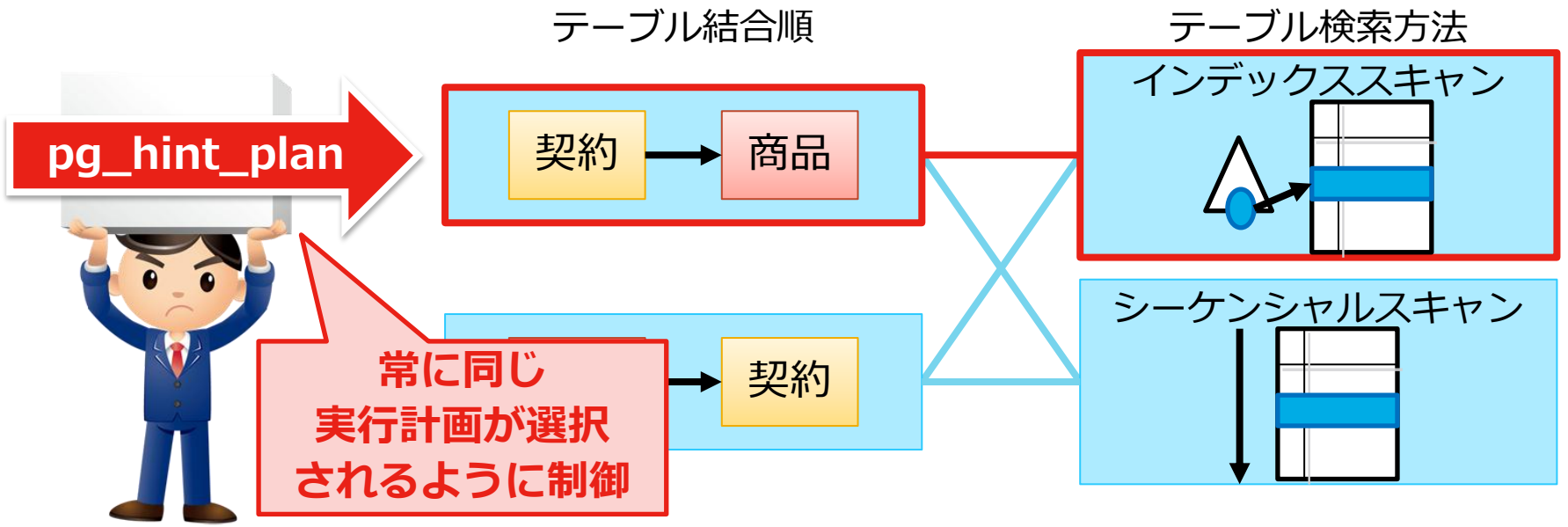
心をつなぐ、社会をつなぐ NTTコムウェア

Innovative R&D by NTT

運用に耐えうるバッチ処理時間を担保せよ
- PostgreSQLのSQL実行時間をコントロールせよ ～SQL実行計画の制御

作戦2 pg_hint_planにより、SQL実行計画をコントロール

pg_hint_planは、SQLヒント句により、PostgreSQLのSQL実行計画を思うままに、コントロールするツール。特に処理時間の変動が大きかったSQLについて、pg_hint_planによって実行計画を固定し、処理時間を安定させた。



心をつなぐ、社会をつなぐ NTTコムウェア Innovative R&D by NTT

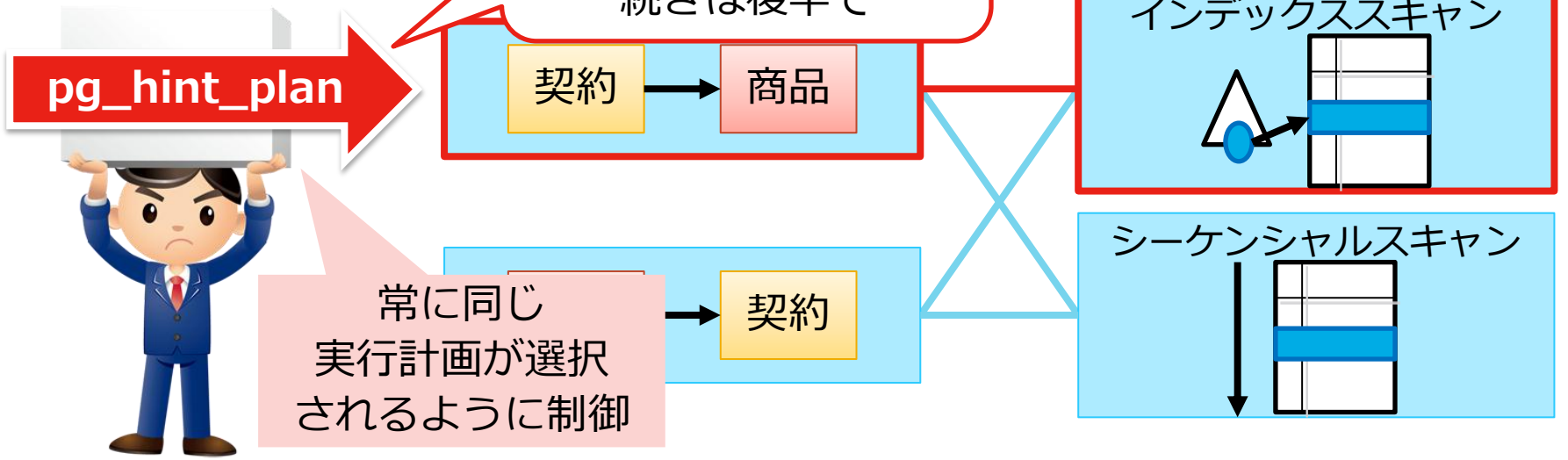
運用に耐えうるバッチ処理時間を担保せよ
- PostgreSQLのSQL実行時間をコントロールせよ ～SQL実行計画の制御

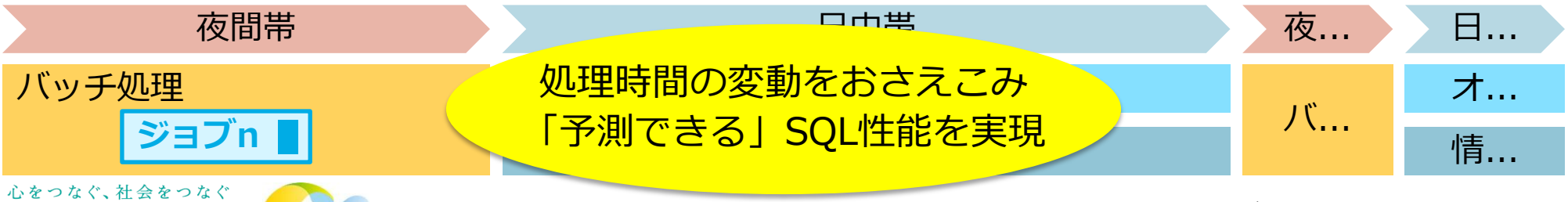
作戦2 pg_hint_planにより、SQL実行計画をコントロール

pg_hint_planは、SQLヒント句により、PostgreSQLのSQL実行計画を思うままに、コントロールするツール。特に処理時間の変動が大きかったSQLについて、pg_hint_planによって[illegible]時間を安定させた。

NTT OSSセンタ 開発ツール

続きは後半で





心をつなぐ、社会をつなぐ NTTコムウェア

Innovative R&D by NTT

壮絶(?)な戦いを振り返って、今思うこと

1 他DBMSでのノウハウをPostgreSQLを使った開発にも活かすべし

性能担保のために取り組んだことは、すべて、他DBMSでの数多の開発で培ったノウハウをベースにしたものだった。他DBMSでの**「当たり前にやるべきこと」**を、PostgreSQLを使った開発にも大いに活かすべき。

2 PostgreSQLでも、SQL性能はコントロールできる

SQL性能をコントロールできるかは、性能要求が厳しいミッションクリティカルシステムを扱う弊社では、DBMS選定の大きなファクターの一つ。

PostgreSQLは、**pg_hint_plan**という武器を手にし、適用範囲が大きく広がった。

3 今後のPostgreSQLには大規模・高負荷システムを意識した進化を期待

大規模・高負荷なミッションクリティカルシステムへの適用を加速させるには、ディスクI/O量削減やパーティショニング機能改善といった、**PostgreSQLのさらなる進化が必要**。より強力で使いやすいDBMSへの発展を期待する。

料金系基幹システムへのPostgreSQL導入における技術的チャレンジ

NTT OSSセンタ　　山田 達朗

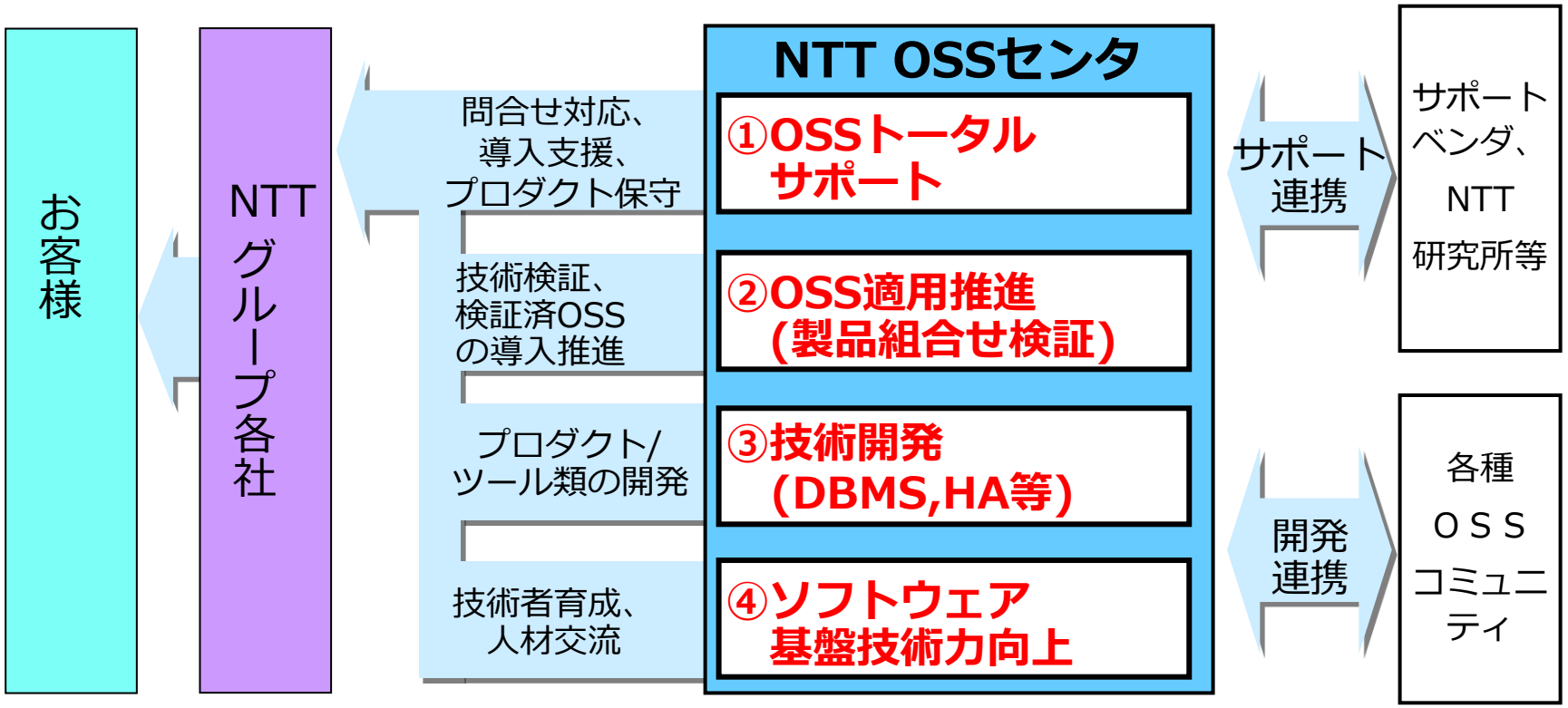
心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

NTT OSSセンタの紹介

◆目的

OSS活用によるNTTグループの**システムのTCO削減**

下記①～④の4つのミッションでグループ事業に貢献



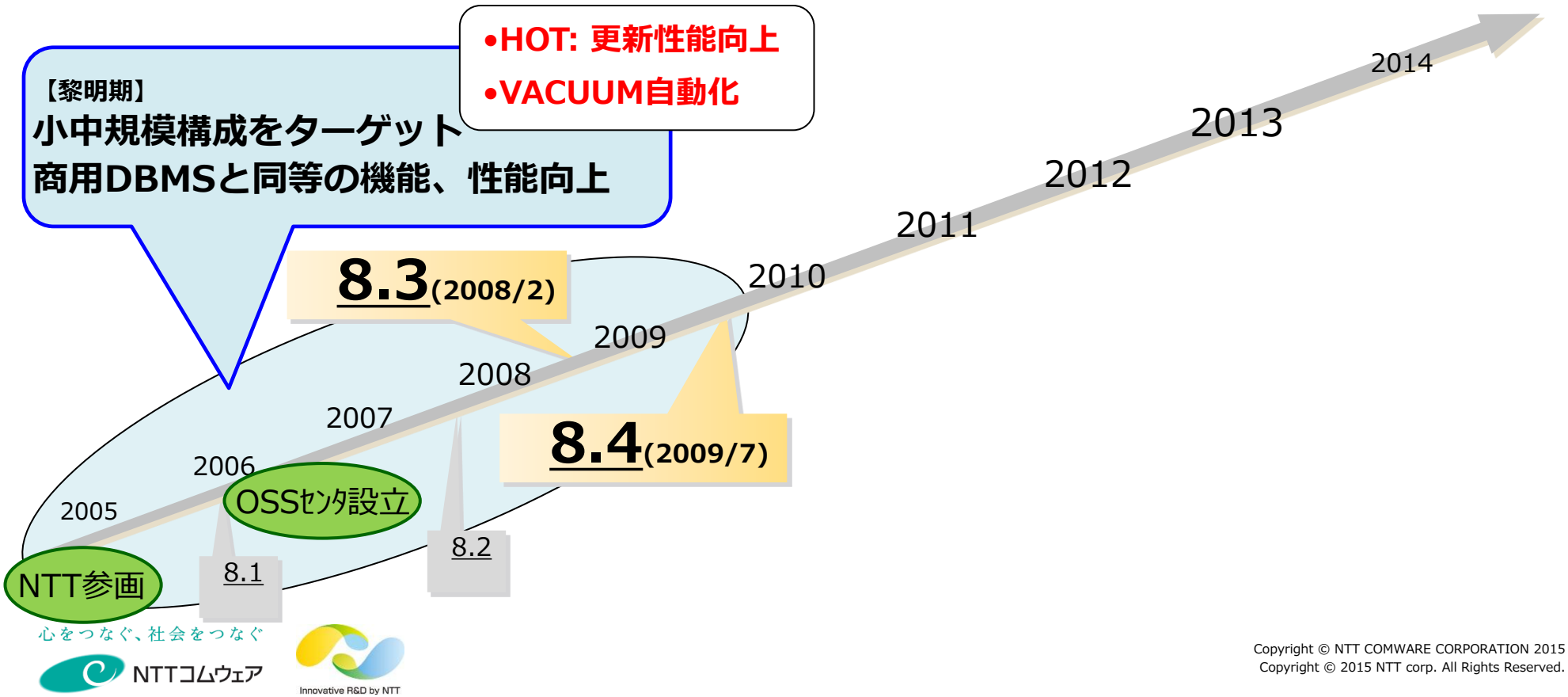
DBMSはPostgreSQLを推進

PostgreSQLの進化とOSSセンタの関わり

◆PostgreSQLのエンタープライズ適用に向けた進化を
OSSセンタの活動状況と合わせてご紹介

赤字：OSSセンタ貢献

➢Step1. 追いつけ！商用DBMS



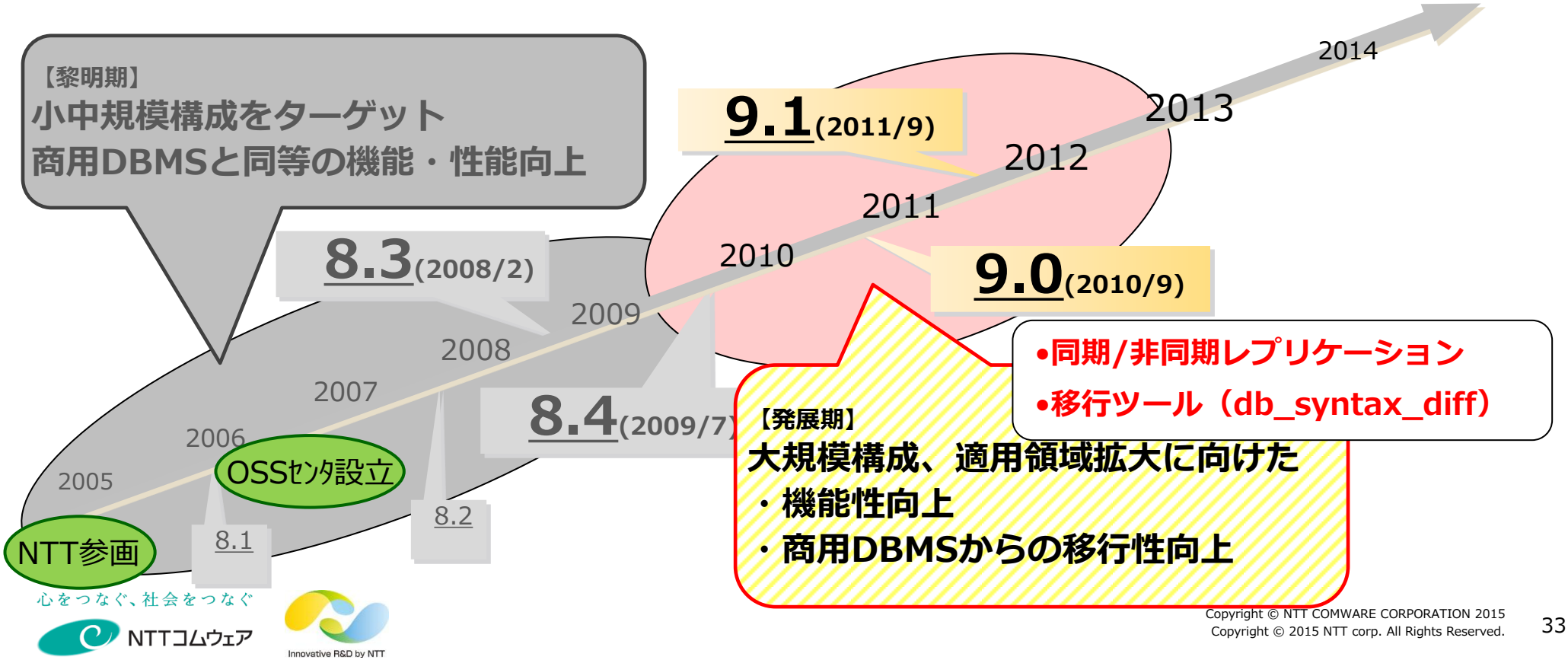
心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

PostgreSQLの進化とOSSセンタの関わり

◆PostgreSQLのエンタープライズ適用に向けた進化を
OSSセンタの活動状況と合わせてご紹介

赤字：OSSセンタ貢献

- Step1. 追いつけ！商用DBMS
- Step2. 信頼性/可用性、移行性の向上



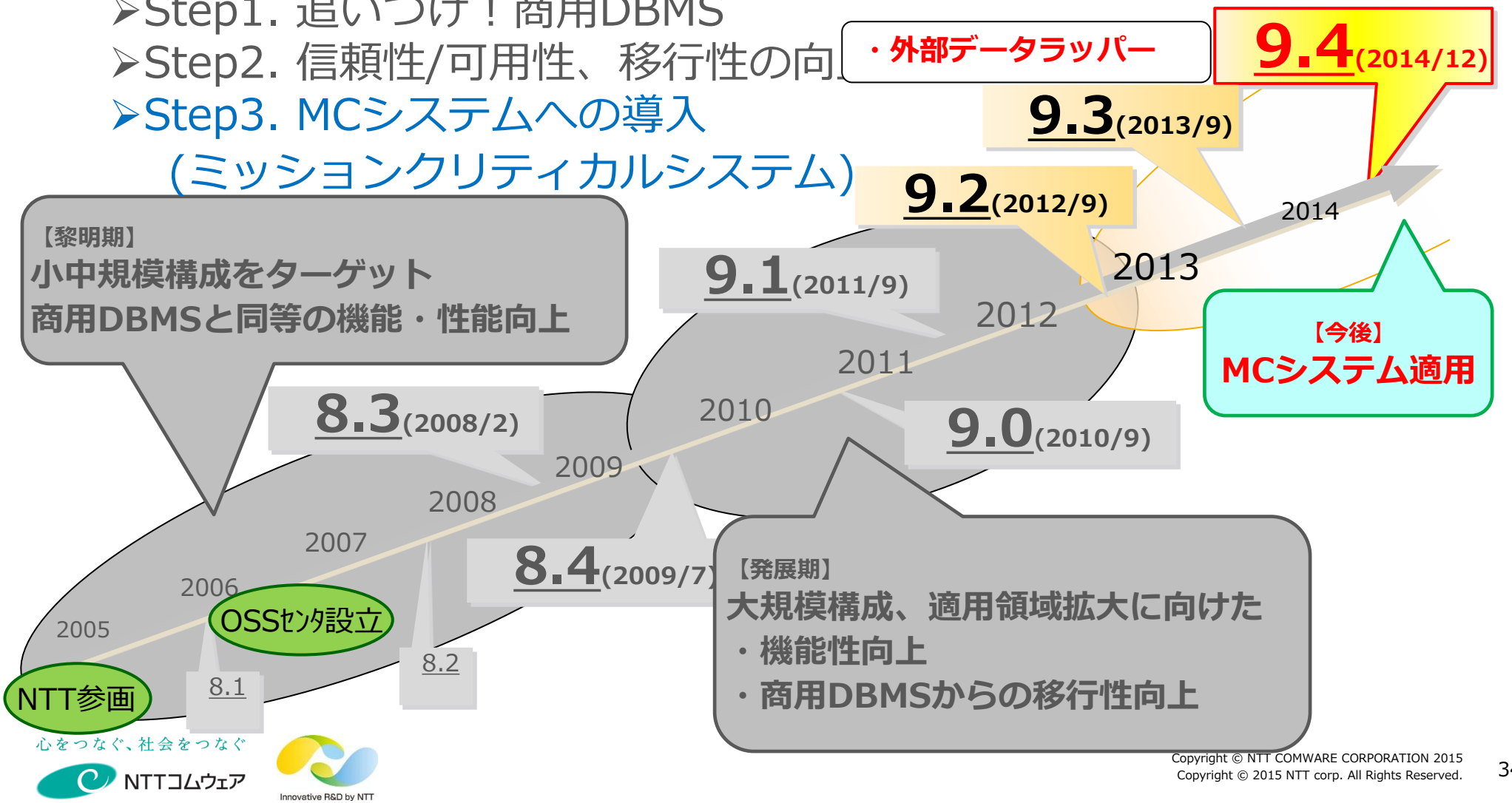
心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

PostgreSQLの進化とOSSセンタの関わり

◆PostgreSQLのエンタープライズ適用に向けた進化を
OSSセンタの活動状況と合わせてご紹介

赤字：OSSセンタ貢献

- Step1. 追いつけ！商用DBMS
- Step2. 信頼性/可用性、移行性の向上
- Step3. MCシステムへの導入
  (ミッションクリティカルシステム)



心をつなぐ、社会をつなぐ
NTTコムウェア
Innovative R&D by NTT

PostgreSQLコミュニティへの貢献

◆NTT OSSセンタの貢献を一部ご紹介　(2014年度)

年間パッチ採用数

- PostgreSQL本体　　：**39**件
- PostgreSQL周辺ツール　：**30**件

講演

- PGCon cluster summit
- PGECons PostgreSQL事例セミナー
- JPUG PostgreSQLカンファレンス

PostgreSQLの開発面や利用面において**貢献**

ここからのおはなし

最大のミッション**「性能の担保」**の達成に向け
PostgreSQLエキスパートとして
全力で戦い抜いた壮絶(?)なストーリー

レプリケーションで
DWHを構築せよ

事例(1)
レプリカ側でテーブルへの
データ反映処理が全く
進まなくなった

バッチ処理時間を
担保せよ

事例(2)
pg_hint_planにより、
SQL実行計画をコントロール

最大のミッション「性能の担保」の達成に向け PostgreSQLエキスパートとして 全力で戦い抜いた壮絶(?)なストーリー

レプリケーションで
DWHを構築せよ

事例(1)
レプリカ側でテーブルへの
データ反映処理が全く
進まなくなった

バッチ処理時間を
担保せよ

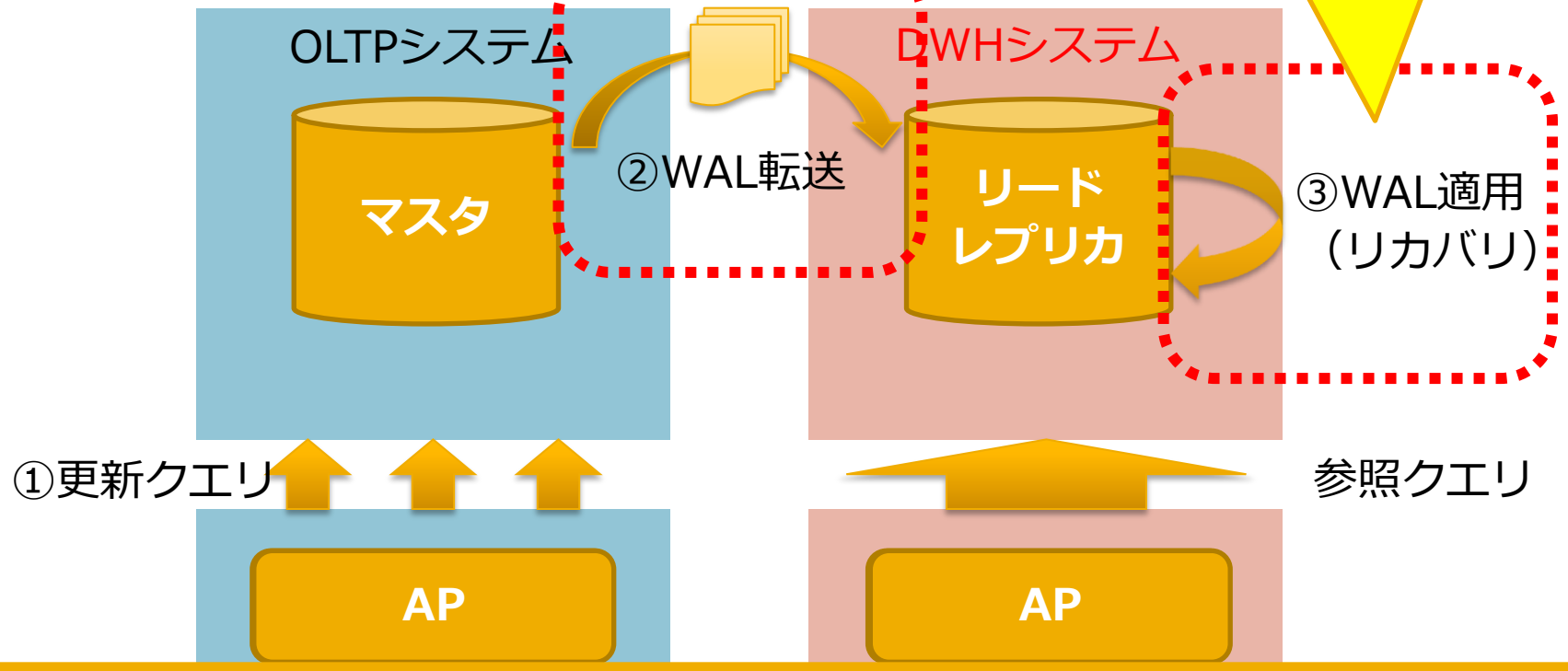
事例(2)
pg_hint_planにより、
SQL実行計画をコントロール

レプリケーションの仕組み

◆マスタとリードレプリカはWAL(更新ログ)を使用

- マスタの更新毎にリードレプリカ（スタンバイ）にWAL
- リードレプリカは受け取ったWALレコードを適用し

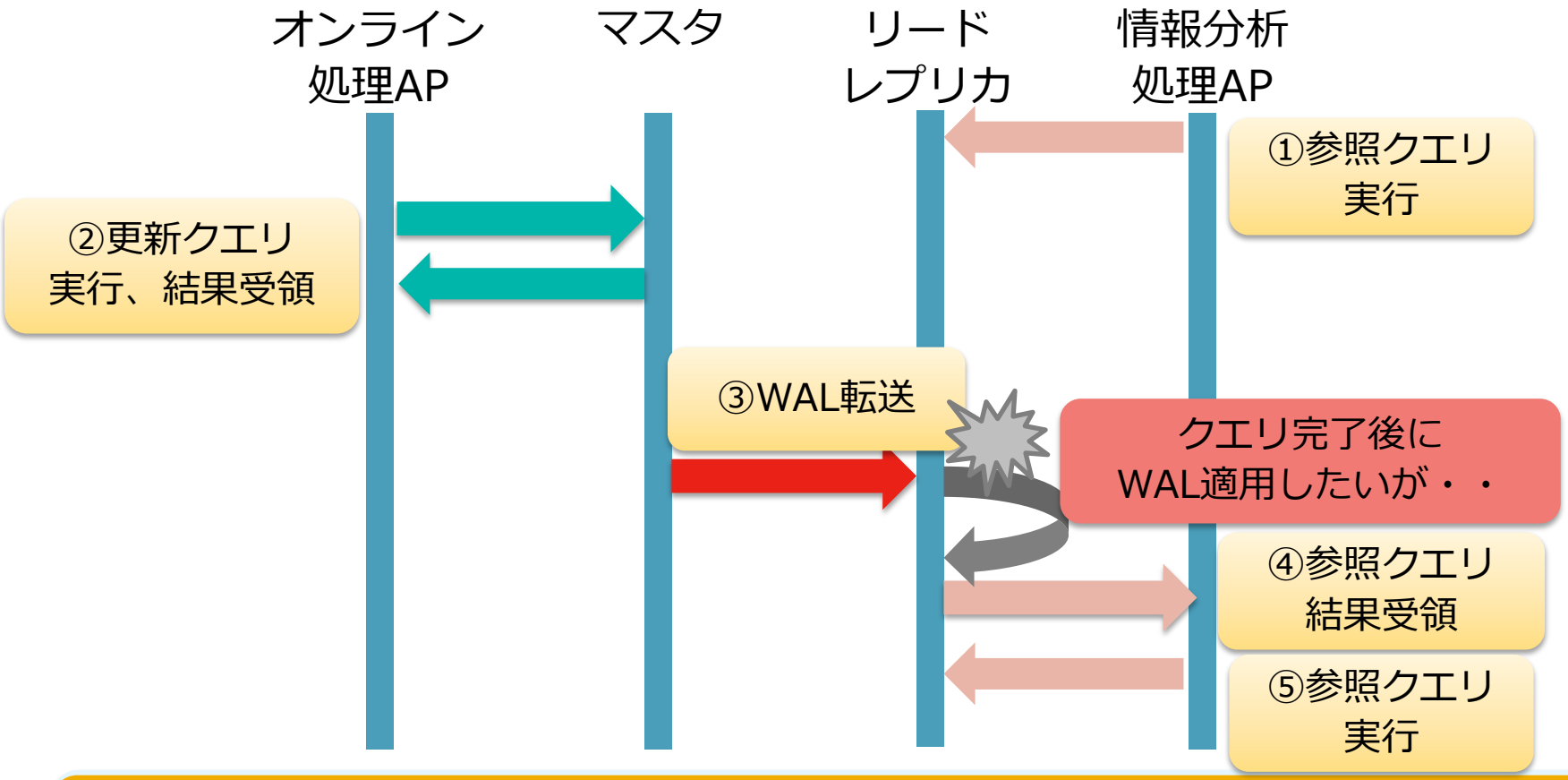
データ反映処理
②③のどちらかが
止まっている？



データ反映処理(同期)が全く進まなくなった原因は？

データ反映処理が全く進まなかった原因は？

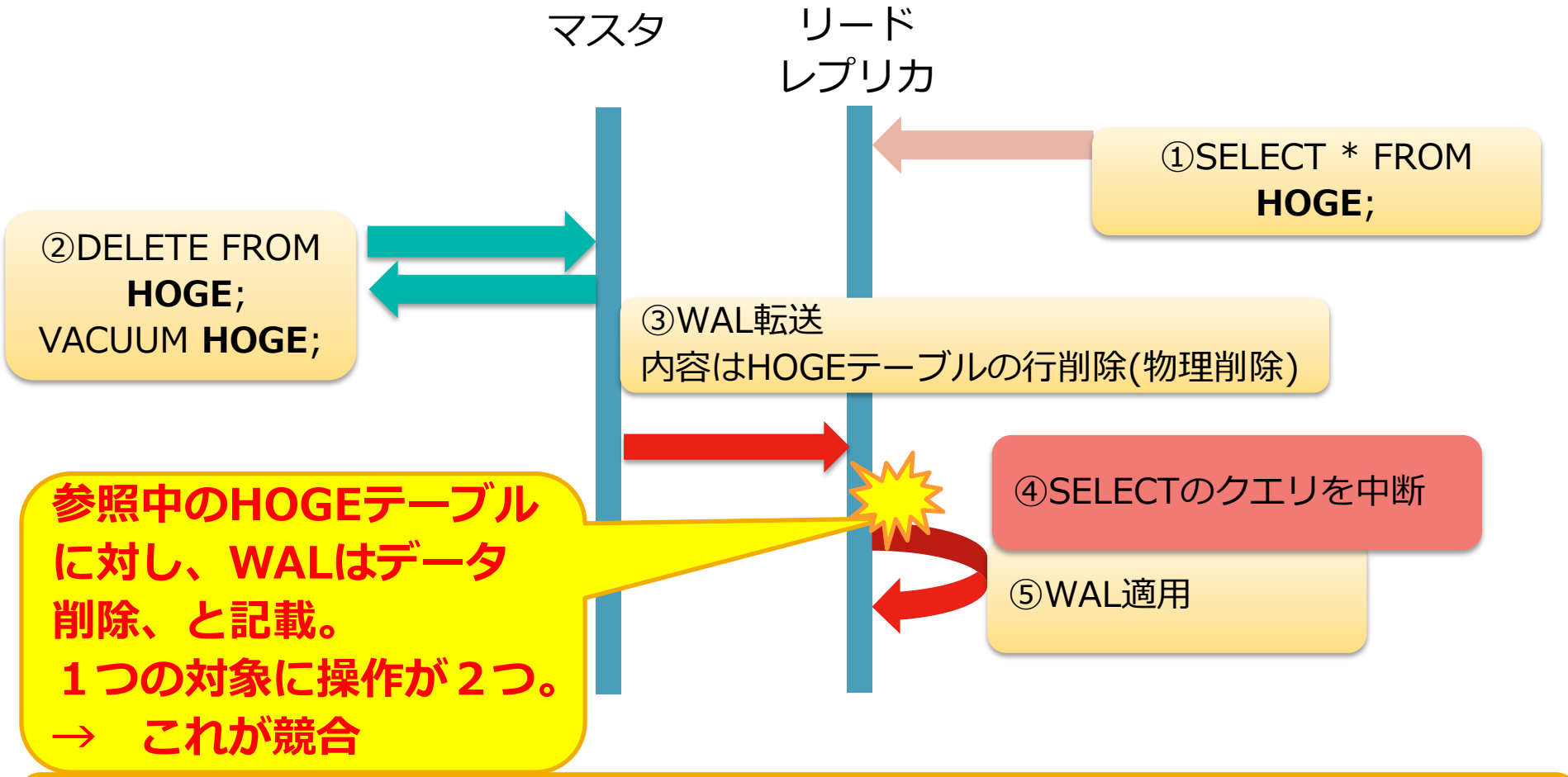
◆WALは転送されていない？ or 適用されていない？



WAL適用と競合する参照クエリが絶え間なく実行されている。
クエリ中断しない設定のため、WAL適用が見送られていた。

WAL適用と競合する参照クエリとは何か？

◆競合の典型的な例　クエリ実行中に行削除が行われた場合



競合発生の場合、 WAL適用が優先されるため、
実行中の参照クエリは中断される。　競合は回避できないか？

競合を回避するには？パラメータはあるか？

◆競合の原因である”削除(物理削除)のみ”を遅延させることが可能

hot_standby_feedback=OFF(デフォルト)/ON

マニュアル（抜粋）

現在処理を行っている問合せについてプライマリーにフィードバックを送るか否か。
レコードの後片付けに起因する問合せの取り消しを排除するために使用できます。 **？ ！**

■既知のパラメータではあるが、実際に採用しても問題無いのか？

- 存在を知っている　≠　理解している（使いこなせる）
- マニュアルには詳細な説明は無い
- 制約はあるのか？

■次のアプローチで動きや制約等を確認した

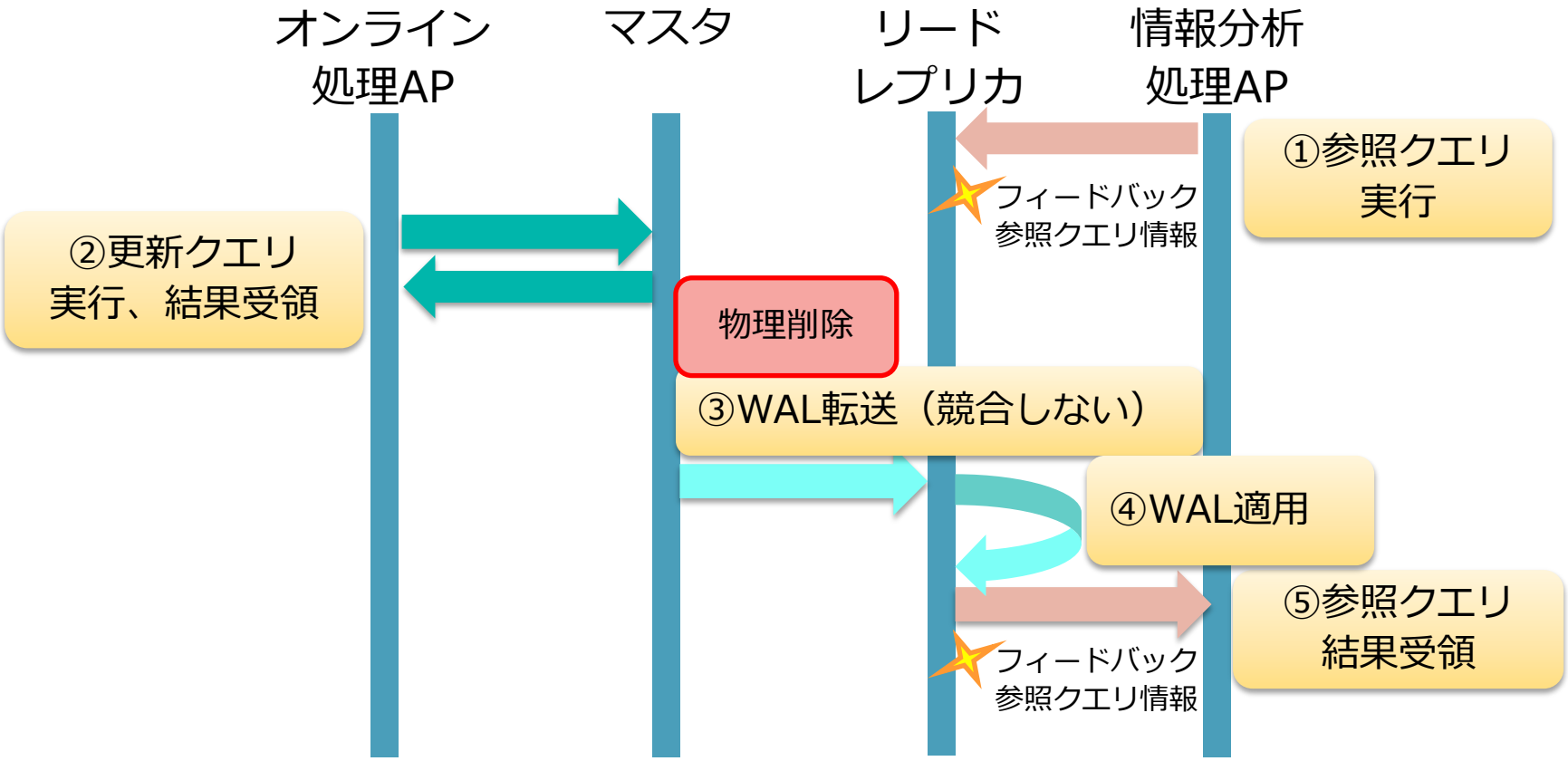
✓実機検証

✓ソース解析

→ **リードレプリカ側は、トランザクションIDをマスタ側にフィードバック。**
マスタ側は、競合する可能性があるレコードの物理削除を遅延させる。

パラメータを有効にした場合、フローはどうなるか？

◆パラメータが有効な場合、クエリと競合するWALは生成しない。
競合しないWALのみが転送されて適用される。



制約や留意点についてはどうか？

制約や留意点

◆パラメータの採用にあたっては、以下に注意

制約

1. NW断などによる**レプリケーション再開直後**はフィードバックが行われておらず、**クエリの中断が回避できない場合がある**。
2. XID周回防止のための**FREEZE処理**が発生した場合は**クエリの中断は回避できない**。

留意点

- マスタの不要行の物理削除が遅延するため、**表の肥大化が進む可能性**がある。

◆本案件においては特に問題無しと判断

制約1.問題なし　NW断などは発生頻度は低い、運用対処で対応

制約2.問題なし　autovacuum停止、手動VACUUMによる運用

留意点 問題なし　クエリ実行時のマスタ側の更新は比較的少ない＆手動VACUUM

フィードバックのパラメータを追加し解決へ！

◆本案件では以下のパラメータを組み合わせて採用

max_standby_streaming_delay=30秒　→　-1

- WALが到着してから何秒後にWAL適用を開始するかを設定する。
- -1　：　クエリ完了までWAL適用は行わない

hot_standby_feedback=OFF　→　ON　追加

- トランザクションIDをマスタに伝え、参照する可能性がある行の削除を待ってもらう。
- ON　：　有効

リードレプリカ側はクエリ中断を回避しつつ、
データ反映処理が行われるようになり、課題は解決

最大のミッション「性能の担保」の達成に向け PostgreSQLエキスパートとして 全力で戦い抜いた壮絶(?)なストーリー

レプリケーションで
DWHを構築せよ

事例(1)
レプリカ側でテーブルへの
データ反映処理が全く
進まなくなった

バッチ処理時間を
担保せよ

事例(2)
pg_hint_planにより、
SQL実行計画をコントロール

pg_hint_planの概要

◆pg_hint_planとは

NTT OSSセンタ製のヒント句を利用可能にするツール

(githubで公開中)

◆ヒント句とは

プラン（実行計画）を制御するためのアドバイス（ヒント）。

以下を柔軟に制御し、目的のプランに誘導することが可能。

・結合順番、結合方法の指定
・表スキャン方法の指定　など多数。

◆メリットは？なぜ作ったか？

プランナの見積り誤りによるプラン訂正や性能安定化が可能

チューニングの最終手段

pg_hint_planの使い方の例

◆ネスティッドループ結合をハッシュ結合に変更するサンプル

```
# EXPLAIN SELECT
#    FROM pgbench_branches b
#    JOIN pgbench_accounts a ON b.bid = a.bid
#   ORDER BY a.aid;
---------------------------
 Nested Loop (c
   Join Filter: (b.bid =
   -> Index Scan
   -> Materialize
        -> Seq Scan c
```

```
# /*+
#     HashJoin(a b)
#     SeqScan(a)
#  */
# EXPLAIN SELECT
#    FROM pgbench_branches b
#    JOIN pgbench_accounts a ON b.bid = a.bid
#   ORDER BY a.aid;
------------------------------------------------------------------------------------
 Sort  (cost=12320.84..12570.84 rows=100000 width=4)
   Sort Key: a.aid
   -> Hash Join  (cost=1.02..4016.02 rows=100000 width=4)
        Hash Cond: (a.bid = b.bid)
        -> Seq Scan on pgbench_accounts a  (cost=0.00..2640.00 rows=100000 width=8)
        -> Hash  (cost=1.01..1.01 rows=1 width=4)
             -> Seq Scan on pgbench_branches b  (cost=0.00..1.01 rows=1 width=4)
```

これがヒント句。
例は、ハッシュ結合と
シーケンシャルスキャンを指定。

本案件におけるpg_hint_planの必要性

◆本案件ではクエリの実行時間を担保するために様々な営みを実施

- SQLコーディングガイドによる机上チェック
- 早期からのプランチェック
- 業務アプリのロジック見直し（必要に応じて）
- 統計情報の手動実行、収集タイミングの管理
- 性能試験による評価（商用相当EOL時点のデータ量と分布で性能要件を満たすことを確認）

それでも

◆プラン変動や非効率なプラン選択が発生し、実行時間が変動。主な原因は以下

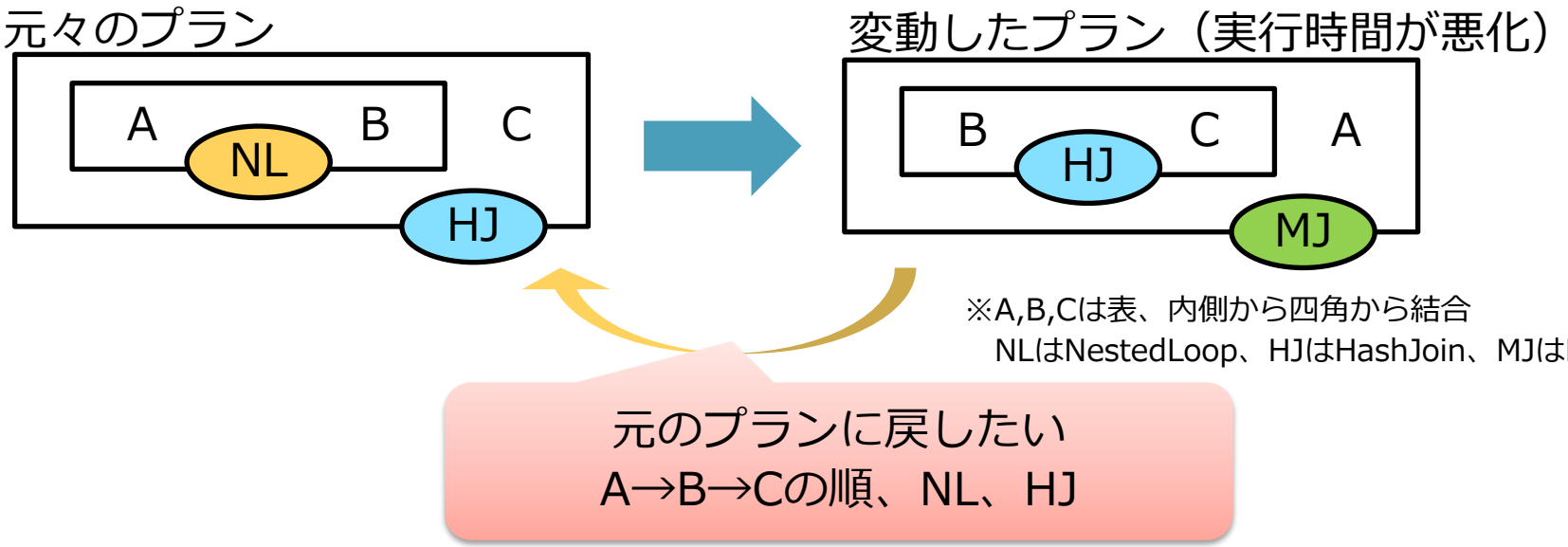
- あるタイミングでデータ分布が変わる　→　コスト見積りが変動
- クエリ内の結合数が多い　→　パターン数の上昇、選択ミス
- プランナが不得手なクエリ　→　コスト見積り誤り（仕様）

コストベースのプランナに全てを任せるのではなく、自らプランを制御する必要がある

具体的にどういうことを行ったのか？

◆典型的な例１　：　あるタイミングでデータ分布が変わる

結合順と結合方法が変動したため、実行時間が変動した



・PostgreSQLの基本機能では、結合順や結合方法を細かく制御できない
　例．set文で結合方法を指定すると、プラン内のすべての結合に影響

→　ヒント句ならば、結合順や結合方法を細かく制御可能！

具体的にどういうことを行ったのか？

◆典型的な例２　：　プランナが不得手なクエリ

Nested Loopが選択されたが、実はHash Joinの方が適切

見積りでは

NL　結合後の行数 10

実際はこのような結果だった

NL　結合後の行数 **1,000,000**

※NLはNestedLoop、HJはHashJoin

NL　は大量の処理に向いていない

HJ　の方が実行時間短縮の見込みがある

なお、ヒント句では、結合方法だけではなく、見積り行やコスト値も変更可能。

pg_hint_planにより、プランの細部までを制御可能。
プランの安定により、システムの安定性が向上！

ここまでのまとめ１

◆２つの課題解決の事例を紹介した。

事例１．レプリカ側でテーブルへのデータ反映処理が全く進まなくなった

レプリケーションによるDWH構築は、マスタ側の**物理削除を遅延させる**、
リードレプリカ側の**WAL適用タイミングを変更することが重要！**

（再掲）

最大のミッション、それは**「性能の担保」**

- 大規模な分析処理を行いつつ、関連システムからのオンライン処理をさばけ

達成

心をつなぐ、社会をつなぐ NTTコムウェア

Innovative R&D by NTT

ここまでのまとめ２

◆２つの課題解決の事例を紹介した。

事例２．pg_hint_planにより、SQL実行計画をコントロール

DB性能の安定化のためには、**ヒント句によるプラン制御が重要**！

（再掲）

最大のミッション、それは**「性能の担保」**

- 大量データの計算処理を確実に時間内で完了しつつ、処理時間の安定化も図れ

達成

全体のまとめ

◆本案件は難易度が高く、PostgreSQL導入における挑戦であった。

- 数百万回線、百種類以上のサービスを扱う顧客料金システム
- 24時間365日稼動
- 複雑かつ大規模な夜間バッチ処理の実行時間厳守、性能安定化
- PostgreSQL＋Linux＋仮想化環境（元々は商用DBMS＋UNIX＋物理環境）
- テラバイドオーダーのデータ量
- OLTPとOLAPの業務両立

◆それに対し、各社の総力を結集し、様々な工夫で課題を乗り越え、「性能の担保」を実現。　現在、システムは安定運用中である。

◆ミッションクリティカルである本システムへのPostgreSQL導入を無事成功に導くことができた。

おわりに

PostgreSQLは益々進化しており、
エンタープライズ利用が進んでいます。

みなさんもぜひ利用を検討してみてください。

ご清聴ありがとうございました

- Linux ®は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。
- その他、記載されている会社名、製品名、サービス名は、各社の商標または登録商標です。