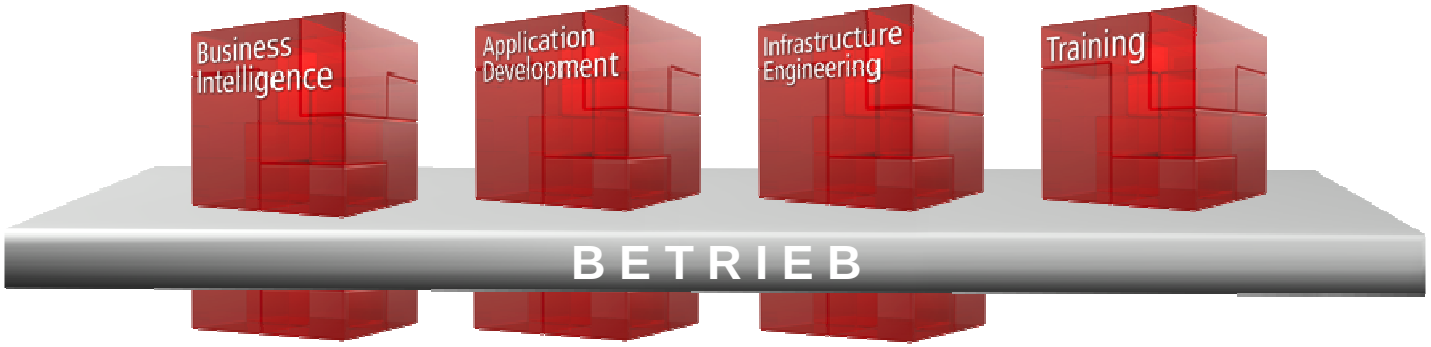
Pure SQL for batch processing

Andrej Pashchenko
Senior Consultant
Trivadis, Düsseldorf

Unser Unternehmen.

Trivadis ist führend bei der IT-Beratung, der Systemintegration, dem Solution Engineering und der Erbringung von IT-Services mit Fokussierung auf ORACLE - und Microsoft -Technologien in der Schweiz, Deutschland, Österreich und Dänemark. Trivadis erbringt ihre Leistungen aus den strategischen Geschäftsfeldern:



Trivadis Services übernimmt den korrespondierenden Betrieb Ihrer IT Systeme.

trivadis makes IT easier.

Mit über 600 IT- und Fachexperten bei Ihnen vor Ort.

- 14 Trivadis Niederlassungen mit über 600 Mitarbeitenden.
- Über 200 Service Level Agreements.
- Mehr als 4'000 Trainingsteilnehmer.
- Forschungs- und Entwicklungsbudget: CHF 5.0 Mio.
- Finanziell unabhängig und nachhaltig profitabel.
- Erfahrung aus mehr als 1'900 Projekten pro Jahr bei über 800 Kunden.

trivadis
makes IT easier.

Über mich

- Senior Consultant bei der Trivadis GmbH, Düsseldorf
- Schwerpunkt Oracle
  - Application Development
  - Application Performance
  - Data Warehousing
- 22 Jahre IT-Erfahrung, davon 16 Jahre mit Oracle DB
- Kurs-Referent „Oracle 12c New Features für Entwickler“ und „Beyond SQL and PL/SQL“

“If you can do it in a single SQL statement, by all means do it in a single SQL statement.” © Thomas Kyte

trivadis
makes IT easier.

Agenda

1. **Rahmenbedingungen: Was? Wann? Warum?**
2. **Vor- und Nachteile**
3. **Werkzeuge**
4. **Praxisfall**
5. **Fazit**

Batch Processing

Batch Processing in der Datenbank im Gegensatz zu OLTP

- nicht interaktive, asynchrone (**Massen-)**Datenverarbeitungsprozesse
- oft erfolgskritische Prozesse
- müssen in Verarbeitungsfenster passen
- lange Transaktionen

Beispiele:

- ETL, nicht nur im Data Warehousing Kontext
- Billing
- Einmalige Patch-Aktionen, Installation, Migration

Pure SQL?

Pure SQL? Was bedeutet das?

- Schwerpunkt der Geschäftslogik in (einem) SQL
- PL/SQL als Ausführungsumgebung für SQL
- PL/SQL für Transaktionssteuerung, Job-Steuerung, Instrumentierung, Logging, Fehlerbehandlung
- Kein SQL in PL/SQL-Schleifen
- Kein übermäßiges Nutzen von PL/SQL Funktionen in SQL-Statements
- Keine Trigger-Logik

Vorteile

Warum?

- 4GL (SQL) vs. 3GL (PL/SQL) und Set-based vs. Row-based
  - wir müssen keine eigenen Algorithmen entwickeln
  - es kann effizientere Algorithmen jetzt und in Zukunft geben (kein Ändern vom SQL Code nötig)
  - nicht künstlich die DB zur Einzelsatzverarbeitung zwingen
- Umgebungswechsel (context switch) zwischen SQL und PL/SQL vermeiden
  - BULK-Operationen und 12c-Neuerungen minimieren den Einfluss, man kann ihn aber ausschließen
- Probleme mit Lesekonsistenz beim Einsatz von PL/SQL werden oft übersehen: Default ist READ COMMITTED auf Statement-Level

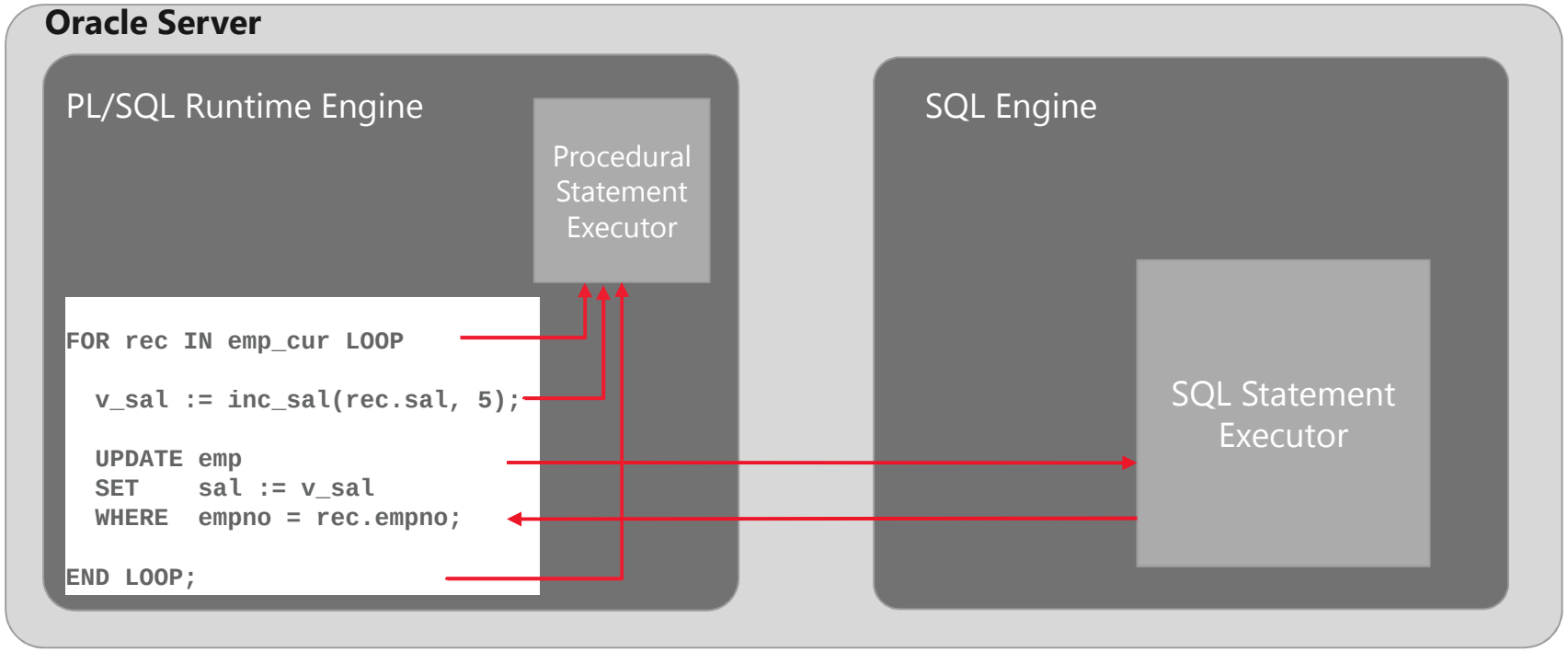
Nachteile

Warum nicht?

- komplizierte Logik lässt sich schwer abbilden
- Know-How, Erfahrung
- kein Logging, Protokollierung der Zwischenergebnisse führen möglicherweise zum erhöhten Testaufwand
- begrenzte Ressourcen (undo, temp)
- interne Richtlinien und Vorschriften (z.B. SQL-Länge)

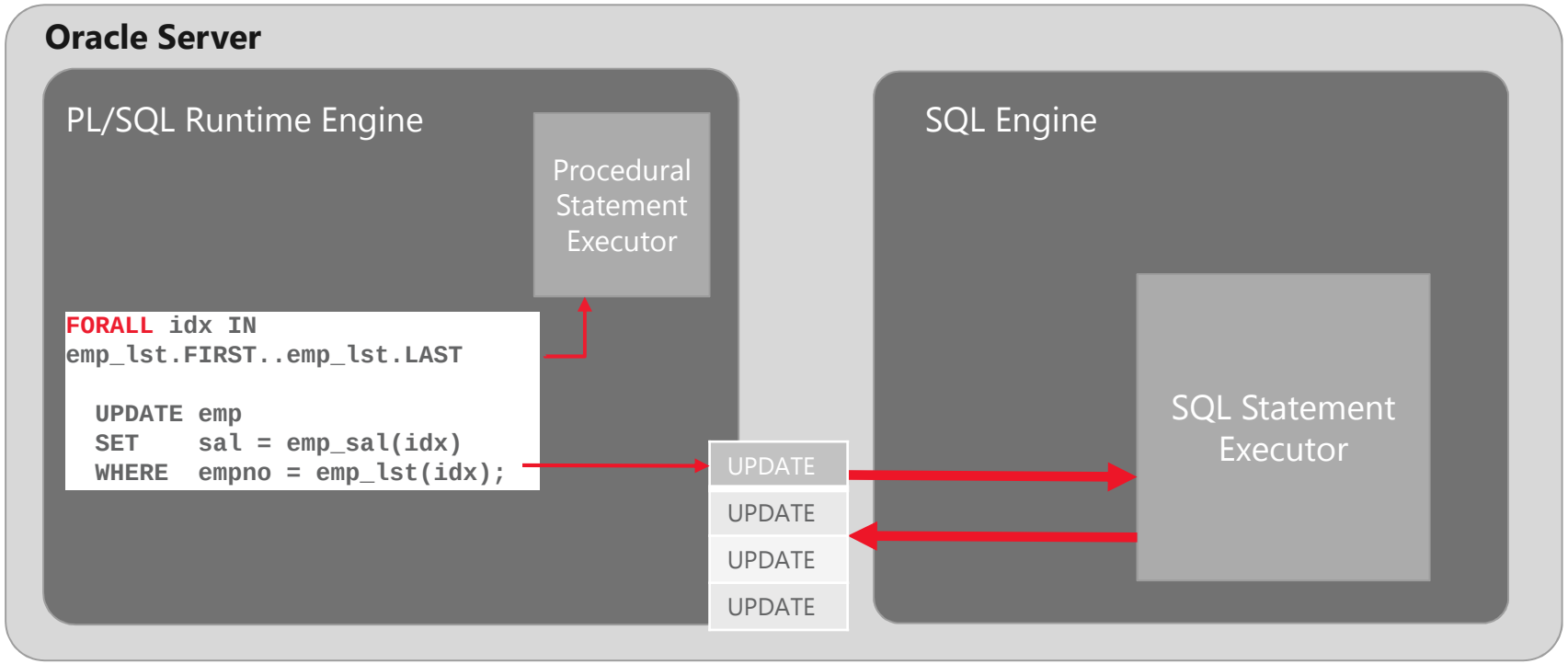
Context Switch

- SQL in der Schleife



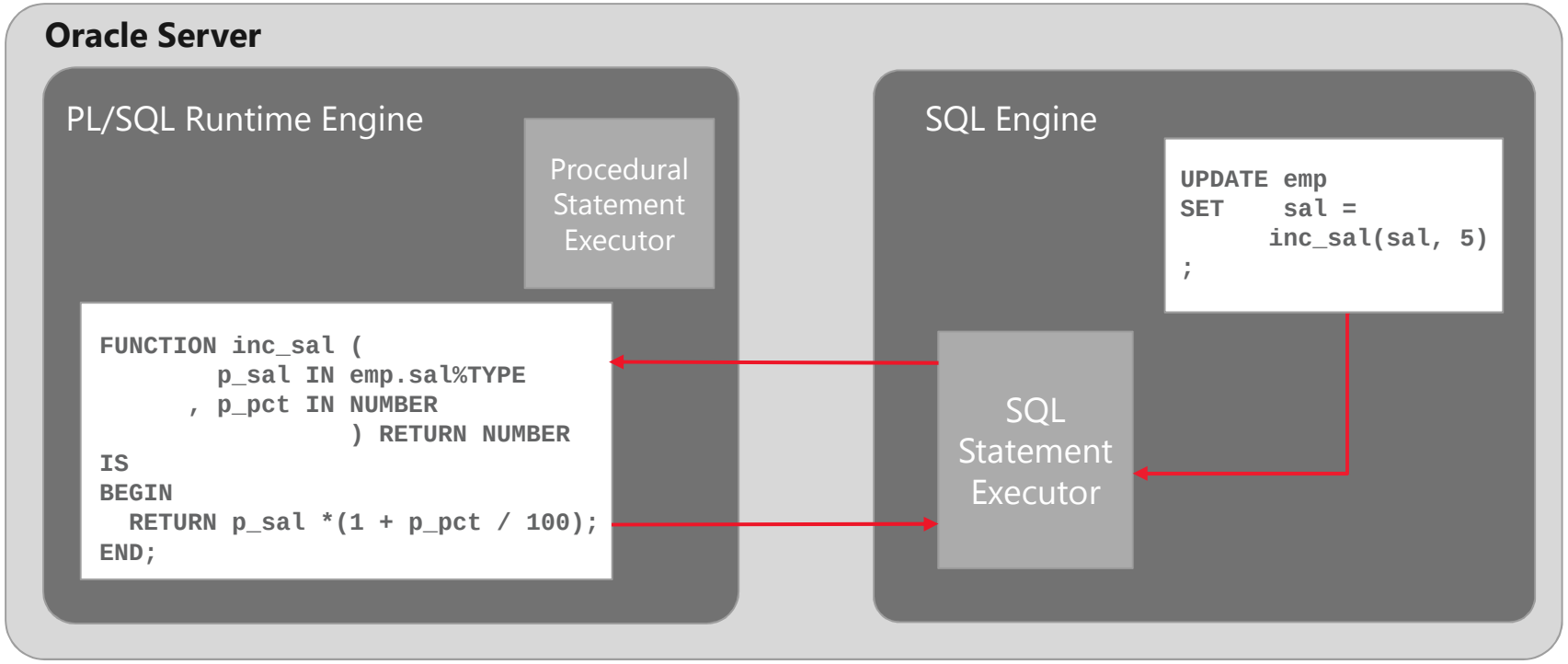
Context Switch

- SQL in der Schleife – Bulk Processing ist besser



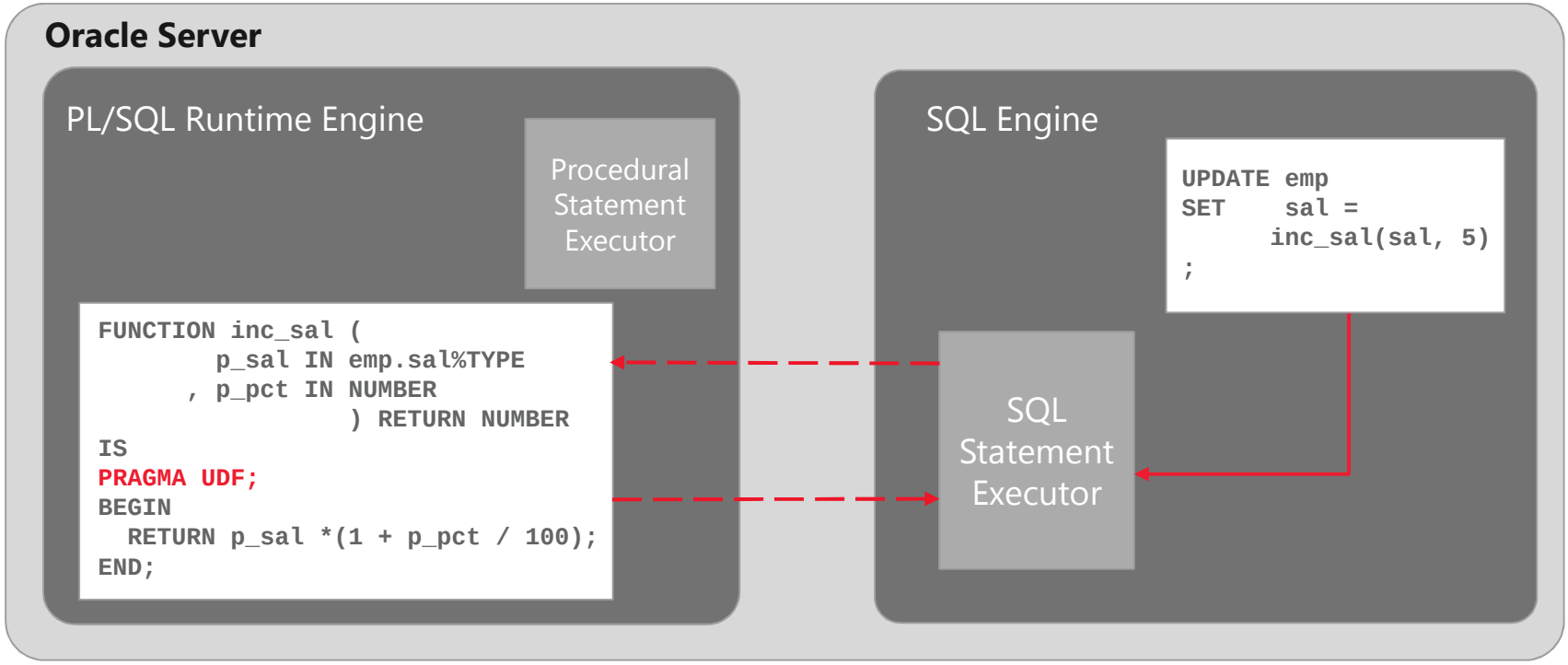
Context Switch

- Aufruf von PL/SQL aus SQL



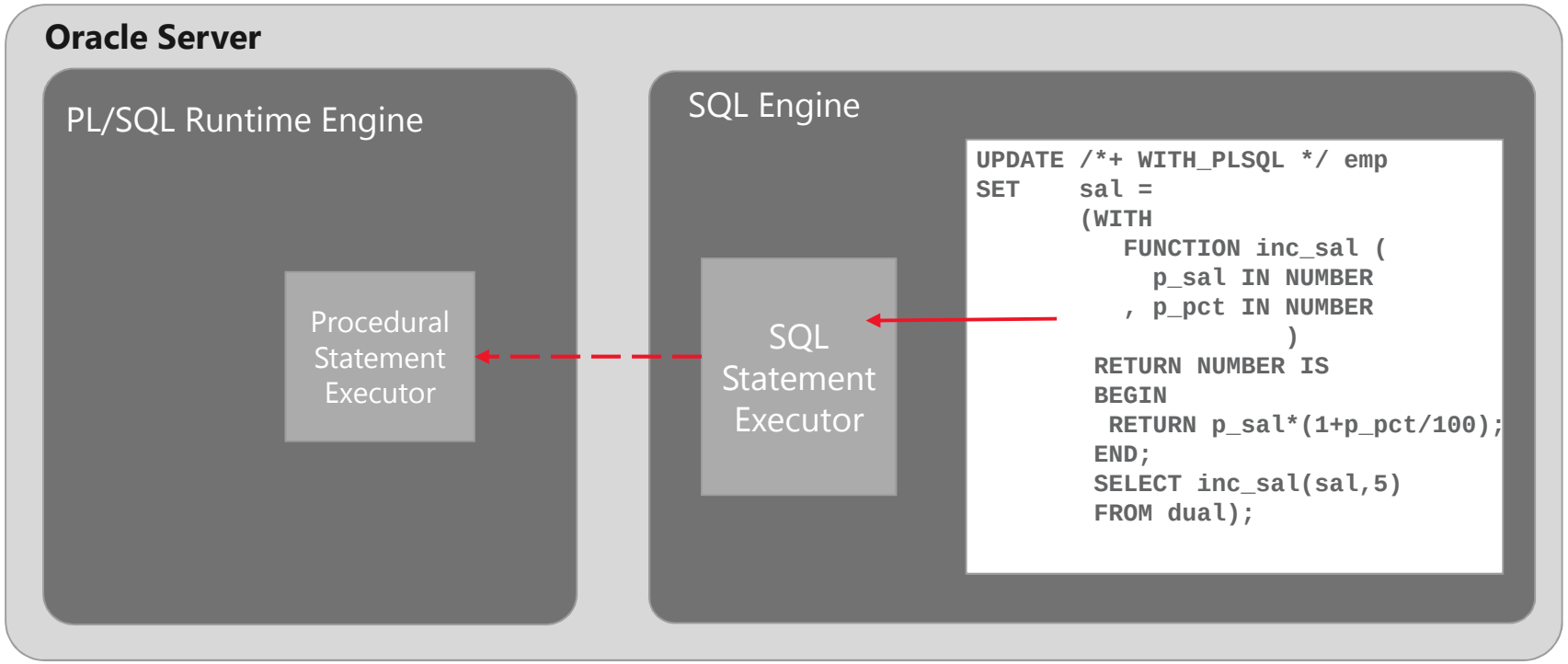
Context Switch

- 12c – Pragma UDF reduziert den Aufwand für Context Switch



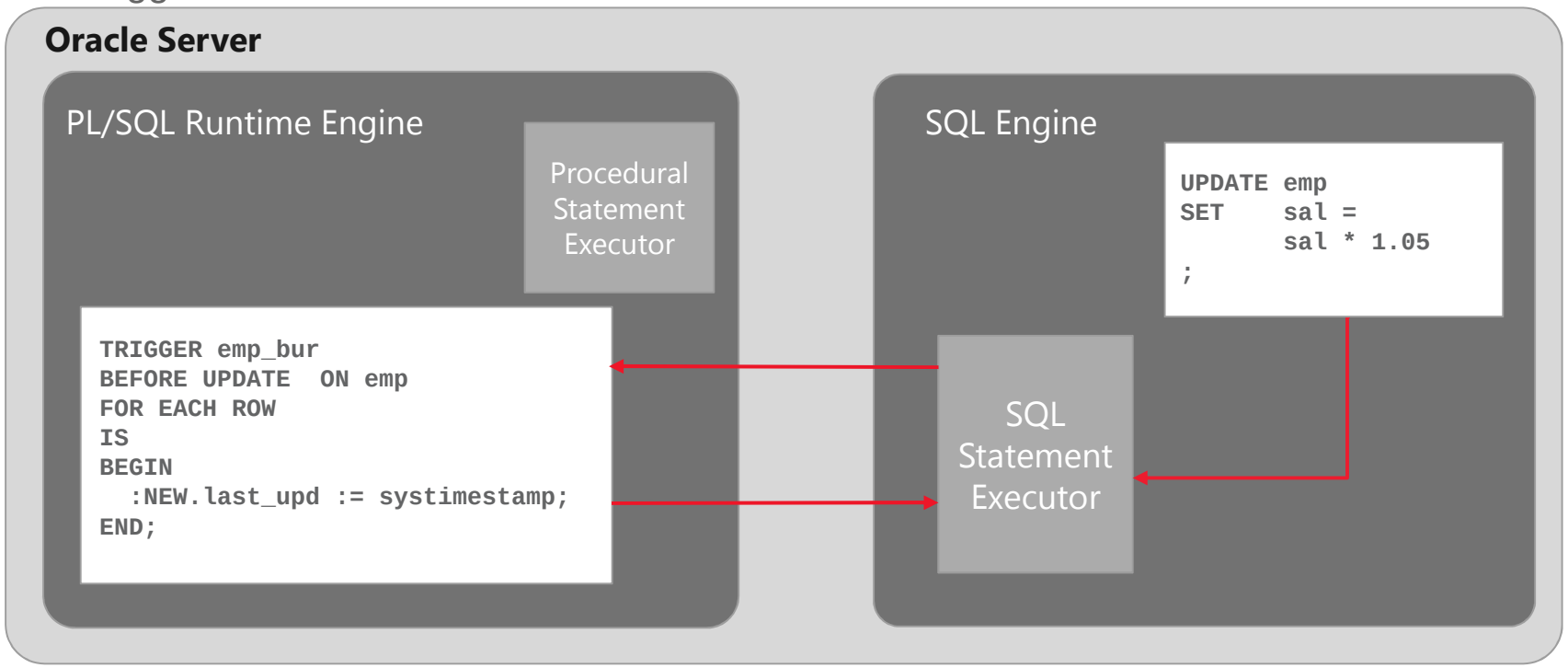
Context Switch

- 12c – Inline PL/SQL (WITH) reduziert den Aufwand für Context Switch



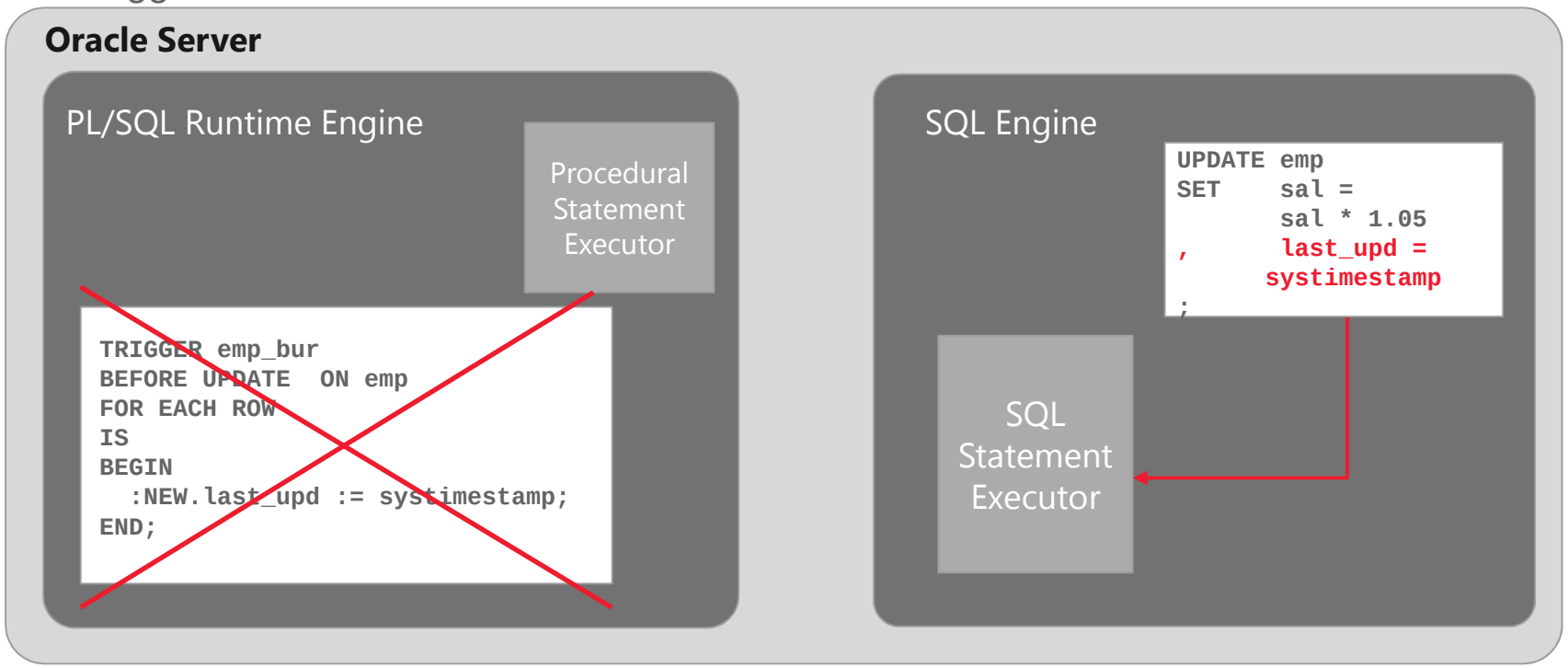
Context Switch

- Trigger führen wieder zum Context Switch



Context Switch

- Trigger führen wieder zum Context Switch



Context Switch

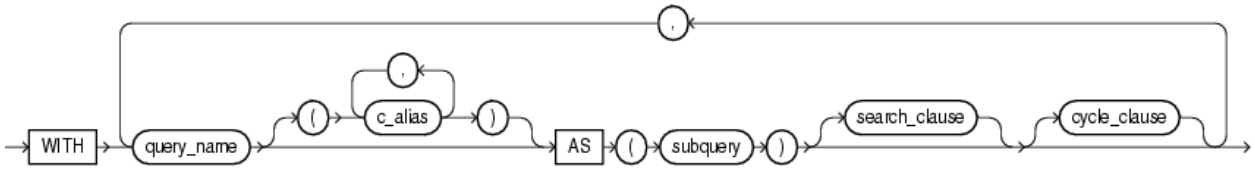
Update	Zeit, s
SQL im FOR-Loop	23,1
FORALL	13,4
SQL mit Funktion	7,5
SQL mit UDF-Funktion	5,8
SQL mit WITH-Funktion	6,2
SQL Pure	3,2
SQL mit aktivem Trigger	16,6
SQL ohne Trigger	3,8

Werkzeuge

Wie lässt sich komplexe Logik ohne 3GL Programmierstrukturen abbilden?

- Gutes Verständnis vom Datenmodell und vom erforderlichen Ergebnis
- Gute Grundkenntnisse in SQL: (Outer) Joins, Gruppierung, Set-Operatoren
- Top-Down-Ansatz für die Analyse und Dekomposition
- Bottom-Up-Ansatz bei der Implementierung
- Paretoprinzip (80 zu 20 Regel): schon wenige Tricks helfen bei der Lösung vieler Aufgaben
  - Struktur, modularer Aufbau: Subquery Factoring (WITH-Klausel)
  - Verzweigung: CASE, Analytische Funktionen, Verschachtelung
  - Schleifen: Row-Source-Generator

Struktur: Subquery Factoring (WITH Klausel)



- ab Oracle 9.2
- neue Art der Verschachtelung
- bringt Ordnung in komplexe Abfragen
- bei Wiederverwendung reduziert die Abfragelänge
- ab 11.2 auch rekursiv als ANSI Variante für CONNECT BY
- ab 12c PL/SQL Prozeduren und Funktionen
- Einfluss auf das Materialisieren der Zwischenergebnisse mit dem Hint /*+ materialize */ (undokumentiert!)

Struktur: Subquery Factoring (WITH Klausel)

- Testbarkeit: ab 11.2 keine Fehlermeldung mehr, einzelne Subqueries leicht testbar

```
WITH groups_marked AS (
    SELECT id
    ,      CASE
              WHEN id != LAG(id,1,id) OVER(ORDER BY id) + 1
              THEN 1
              ELSE 0
           END new_grp
    FROM   t_gaps)
,    sum_grp AS (
    SELECT id
    ,      SUM(new_grp) OVER(ORDER BY id) grp_sum
    FROM groups_marked )
SELECT * FROM groups_marked;
ORA-32035: unreferenced query name defined in WITH clause
ORA-01762: Vopdrv: View Query Block Not In From
```

Verzweigungen: CASE, Analytische Funktionen

"Analytic functions are the coolest thing to happen to SQL since the keyword SELECT"

Tom Kyte

- Seit Version 8i (1999), auch Windowing Functions genannt
- Ermöglichen Zugang zu mehreren Datensätzen des DataSets ohne einen Self-Join
- Erlauben die Logik, die früher prozeduralen Sprachen vorbehalten war
- Lassen sich nicht ineinander verschachteln und nur in SELECT- und ORDER BY-Liste verwenden – dies führt zu Verschachtelung der Abfragen
- Ein CASE als Argument oder umgekehrt analytische Funktionen in CASE-Bedingungen
- CASE auch in ORDER BY

CASE, Analytische Funktionen

Geordnet nach Item-ID die ausgelieferten Items zählen, an jedem das erste Auslieferungsdatum von Items im Status „ausgeliefert“ vermerken

ORDER_ID	ID	STATUS	SHIP_DATE
1	1	shipped	01.02.2016
1	2	shipped	01.01.2016
1	3	open	
1	4	returned	01.12.2015

```
SELECT  order_id, id, status
,       COUNT(CASE WHEN status='shipped' THEN id ELSE NULL END)
            OVER (PARTITION BY order_id ORDER BY id) anzahl_mit_count
,       SUM(CASE WHEN status='shipped' THEN 1 ELSE 0 END)
            OVER (PARTITION BY order_id ORDER BY id) anzahl_mit_sum
,       FIRST_VALUE(ship_dt)
            OVER (PARTITION BY order_id ORDER BY ship_dt) first_shipment_falsch
,       FIRST_VALUE(CASE WHEN status='shipped' THEN ship_dt ELSE NULL END)
            OVER (PARTITION BY order_id ORDER BY ship_dt) first_shipment_auch_falsch
,       FIRST_VALUE(ship_dt)
            OVER (PARTITION BY order_id
              ORDER BY CASE WHEN status='shipped' THEN ship_dt ELSE NULL END
              NULLS LAST) first_shipment
FROM   order_items oi;
```

Schleifen: Row-Source-Generator

- Eine Datenstruktur mit definierten Spalten und definierten (Mindest-) Anzahl Datensätze
- Ermöglicht eine Art “Schleifen” in SQL
- Früher oft eine große Tabelle + ROWNUM

```
SELECT rownum FROM all_objects WHERE rownum <= 20000;
```

- Heute oft über CONNECT BY mit DUAL

```
SELECT level lvl FROM dual CONNECT BY level <= 20000;
```

Schleifen: Row-Source-Generator

- Als Iterationshilfe: Rufnummer in Bestandteile zerlegen
- Statt drei UNION-Abfragen ein Join mit dem Row Source

```
WITH rufnummer AS (SELECT '(+49) 211 58666470' rn FROM   dual)
SELECT REGEXP_REPLACE(rufnummer.rn,  '(\(.*\)) (.*) (.*)', '\1') AS lkz
FROM   rufnummer
UNION ALL
SELECT REGEXP_REPLACE(rufnummer.rn,  '(\(.*\)) (.*) (.*)', '\2') AS onkz
FROM   rufnummer
UNION ALL
SELECT REGEXP_REPLACE(rufnummer.rn,  '(\(.*\)) (.*) (.*)', '\3') AS rufnummer
FROM   rufnummer;
```

```
WITH rufnummer AS (SELECT '(+49) 211 58666470' rn FROM   dual)
,    rs as (SELECT level lvl FROM dual CONNECT BY level <=3)
SELECT REGEXP_REPLACE(rufnummer.rn,  '(\(.*\)) (.*) (.*)', '\'||rs.lvl) AS ergebnis
FROM rs, rufnummer;
```

trivadis
makes IT easier.

Schleifen: Row-Source-Generator

- Als Raster: An welchen Tagen fielen Verkäufe an? vs. Wieviel pro Tag verkauft?

```
WITH alle_tage AS
-- Alle Tage aus dem aktuellen Jahr
   (SELECT TRUNC(sysdate-level+1) dt FROM DUAL
    WHERE TRUNC(sysdate-level+1) >= TRUNC(sysdate,'YYYY')
    CONNECT BY LEVEL <=366)
SELECT a.dt order_date, SUM(NVL(o.order_total,0)) total
FROM   alle_tage a
       LEFT JOIN orders o on (o.order_date = a.dt)
GROUP BY a.dt
ORDER BY a.dt;
```

Fall aus der Praxis

- Datenaustausch mit externen Datenabnehmern im Bereich Automotive
- Überführung in anderes Datenmodell
- Prozess bestehend aus mehreren Schritten
- Bei wesentlichen Veränderungen in den Anforderungen wurden die Teilprozesse nach und nach auf SQL umgestellt
- Auf das Logging der Zwischenergebnisse konnte man weitgehend verzichten

Fall aus der Praxis

- Performancegewinn Faktor drei bis fünf 
- Gut strukturierte und kommentierte SQL-Anweisungen sind wichtig 
- Optimizer stößt manchmal an seine Grenzen:
  - Zusatzaufwand für die Analyse 
  - Daten der Zwischenergebnisse physikalisch materialisieren, Statistiken 
  - Hints + Baselines 
- Ressourcenengpässe verursachen (z.B. Temp)  aber auch bekämpfen (redo) 
  - Daten der Zwischenergebnisse physikalisch materialisieren 
  - parallelisieren mit DBMS_PARALLEL_EXECUTE 
  - Einfluss auf redo über direct path load (APPEND-Hint) 

Fazit

- Bessere Performance
- Eingeschränkte Wiederverwendbarkeit
- Wartbarkeit: gut strukturierte und kommentierte SQL-Anweisungen sind wichtig
- Wichtig für die Entwickler – denken in Mengen, statt an einzelne Datensätze

Andrej Pashchenko
Senior Consultant

Tel. +49 211 58 66 64 70
andrej.pashchenko@trivadis.com

trivadis
makes IT easier.