Smart Database Architecture

Getting Great Performance by Using The Database As a Processing Engine

Toon Koppelaars
Real-World Performance
Oracle Server Technologies

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

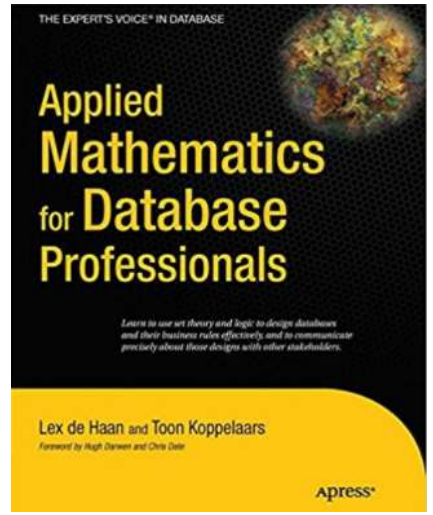
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Who I Am

@ToonKoppelaars

- Part of Oracle eco-system since 1987
  - Developer, DBA, Architect, Problem Solver
- Author of TheHelsinkiDeclaration.blogspot.com
  - http://thehelsinkideclaration.blogspot.com/2009/03/start-of-this-blog.html
  - Mr. Use-Database-As-Processing-Engine
- Coauthor of “Applied Mathematics for Database Professionals”
  - Mathematical Foundation Of The Relational Data Model
- Now in Real-World Performance group
  - Advising Application Developers How to Use Database in Smart Way



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Real-World Performance

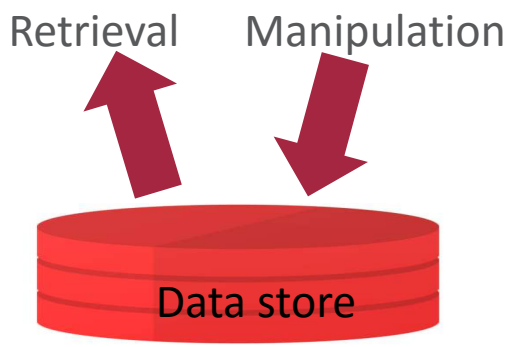
Who We Are

- Part of the database development organization
- Global team located in USA, Europe, Asia
- 300+ Combined years of Oracle database experience
- Our methods:
  - Use the product as it was designed to be used
  - Numerical and logical debugging techniques
  - Avoid and eliminate “tuning” by hacking/guessing/luck
  - Educate others about the best performance methods and techniques

Opening Preliminaries

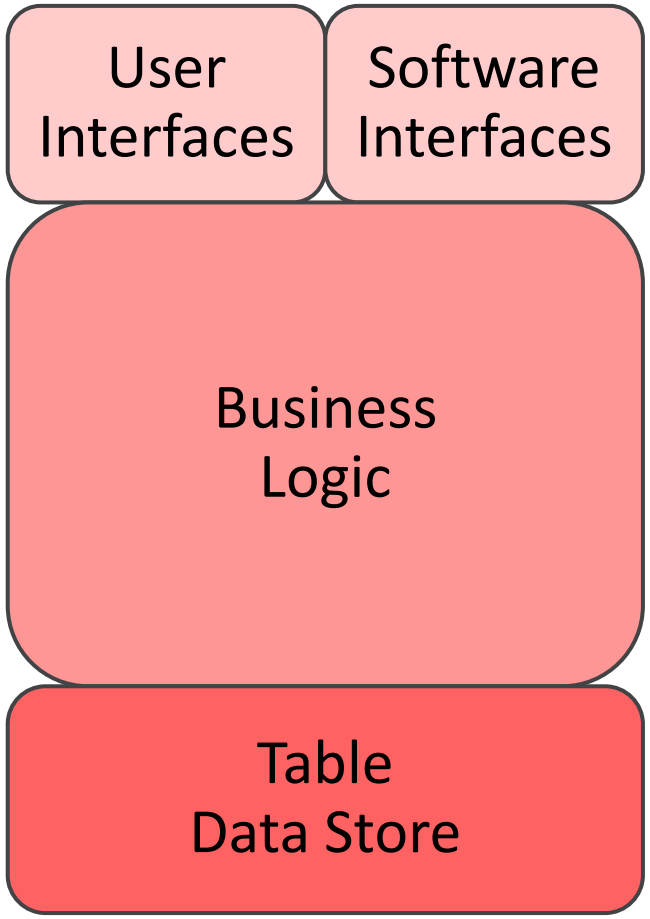
- Getting Great Performance by Using The Database As a Processing Engine

Opening Preliminaries: Our Context



- Getting great performance for OLTP applications
- Transactional business applications
  - Data store as foundation
  - Much/complex CRUD functionality on top
  - User interfaces, reports, workflows, event-driven background processing
  - Provision of API's for other applications
  - Potentially many (web) users

Opening Preliminaries: Business Applications



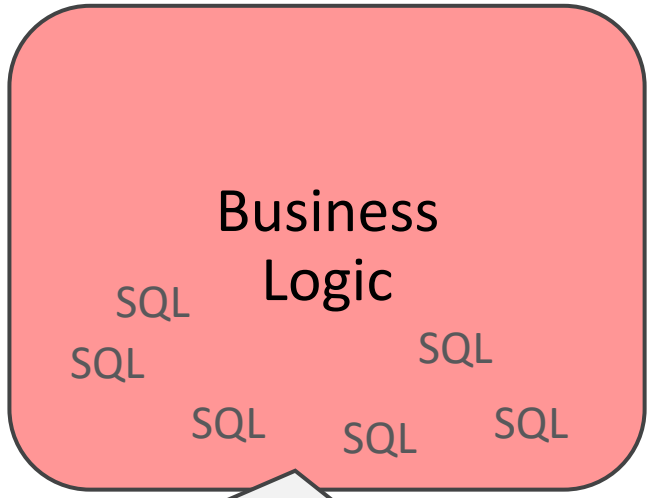
- Conceptually three tiers
  - Functionality exposed via interfaces
    - GUI's for human interaction
    - REST, Soap or otherwise, for software interaction
  - Business logic
  - Data store, relational database

Opening Preliminaries: Business Logic

Business logic

From Wikipedia, the free encyclopedia

In computer software, **business logic** or **domain logic** is the part of the program that encodes the real-world business rules that determine how data can be created, displayed, stored, and changed. It is contrasted with the remainder of the software that might be concerned with lower-level details of managing a database or displaying the user interface, system infrastructure, or generally connecting various parts of the program.



Code with embedded data access statements in it
The way the business requires this to be done

ORACLE

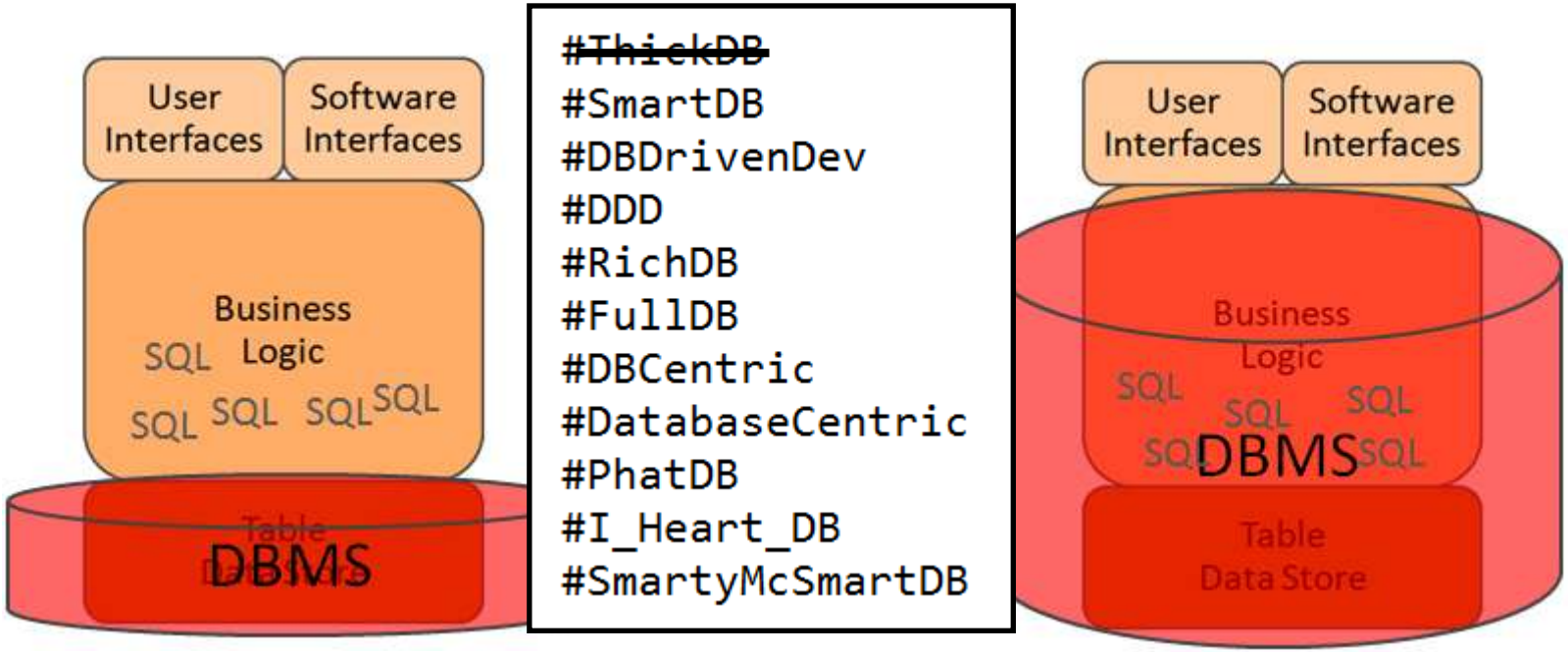
ORACLE
REAL-WORLD PERFORMANCE

Opening Preliminaries: Two Approaches

- Database: Processing Engine or Persistence layer?

- This is all about where you implement “Business Logic” code
  - Inside database → PL/SQL issues all SQL
  - Outside database → Something else (Java, JavaScript) issues all SQL

We See Two Mutually Distinct Approaches



DBMS = Persistence Layer

"NoPlsql" Approach

DBMS = Processing Engine

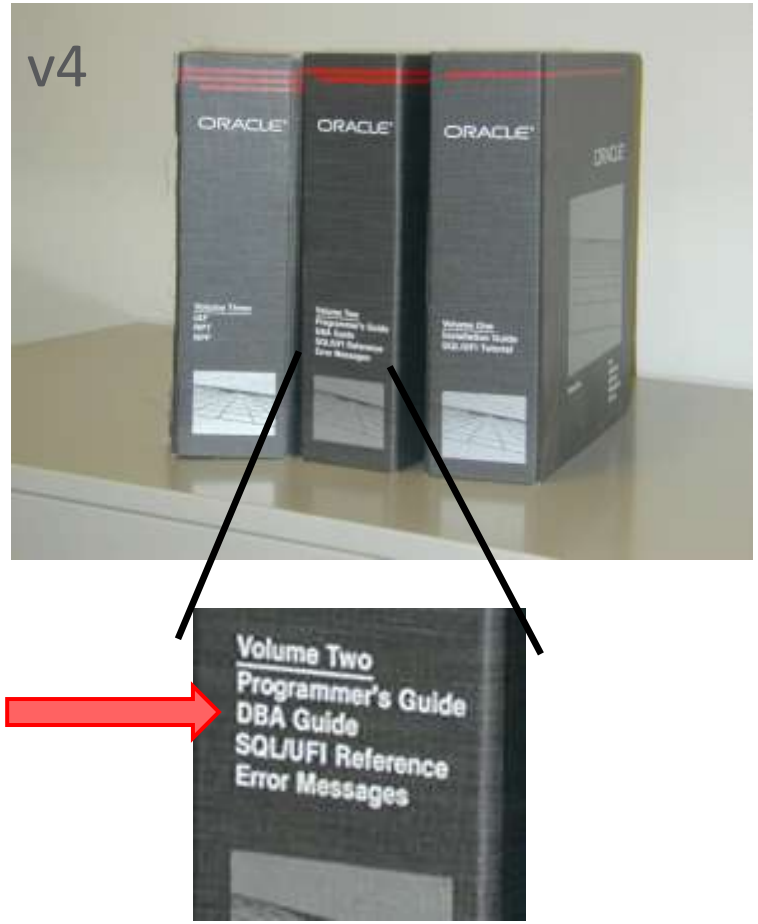
"SmartDB" Approach

Roadmap

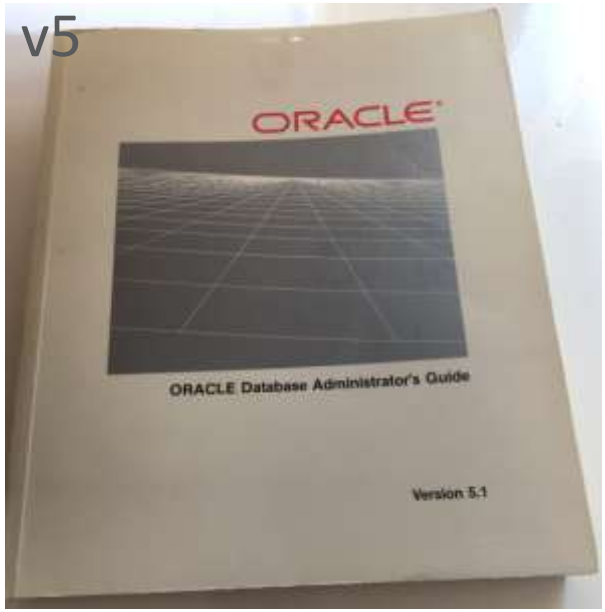
1. A bit of history and observations
2. Debunking performance and scalability argument
3. Some closing remarks

Oracle v4, v5, v6 Database Documentation

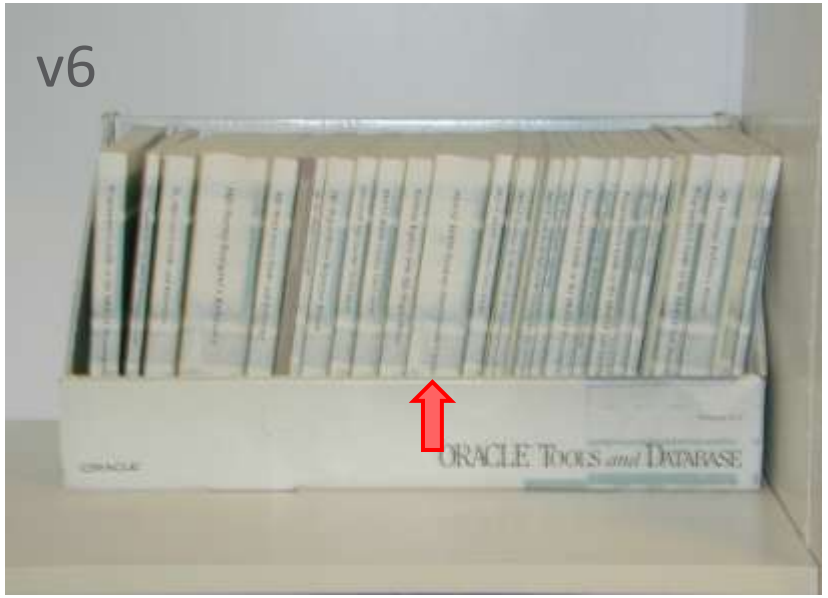
v4



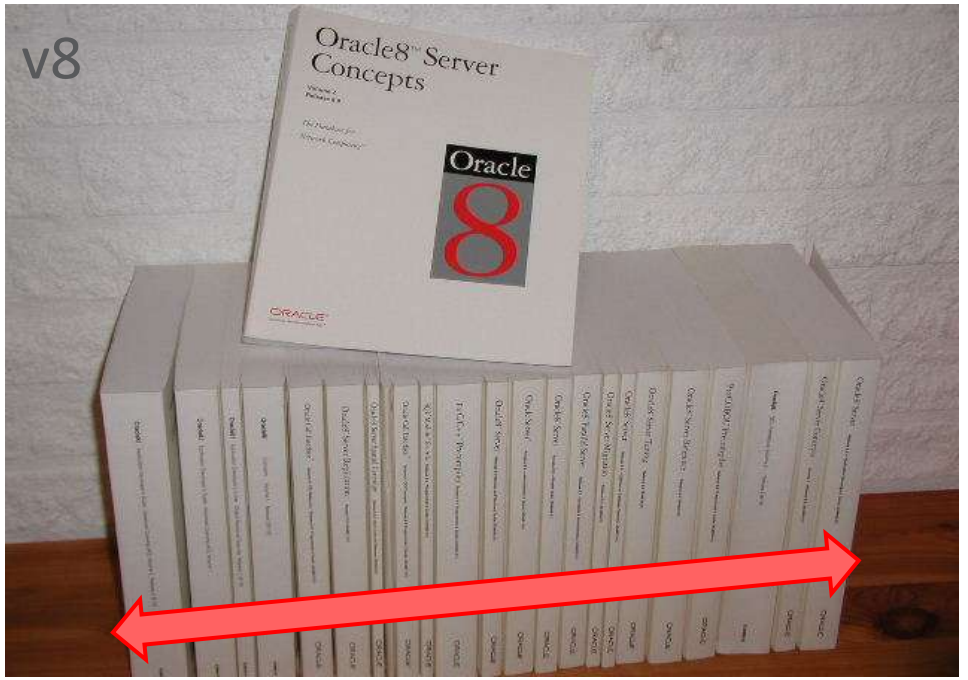
v5



v6



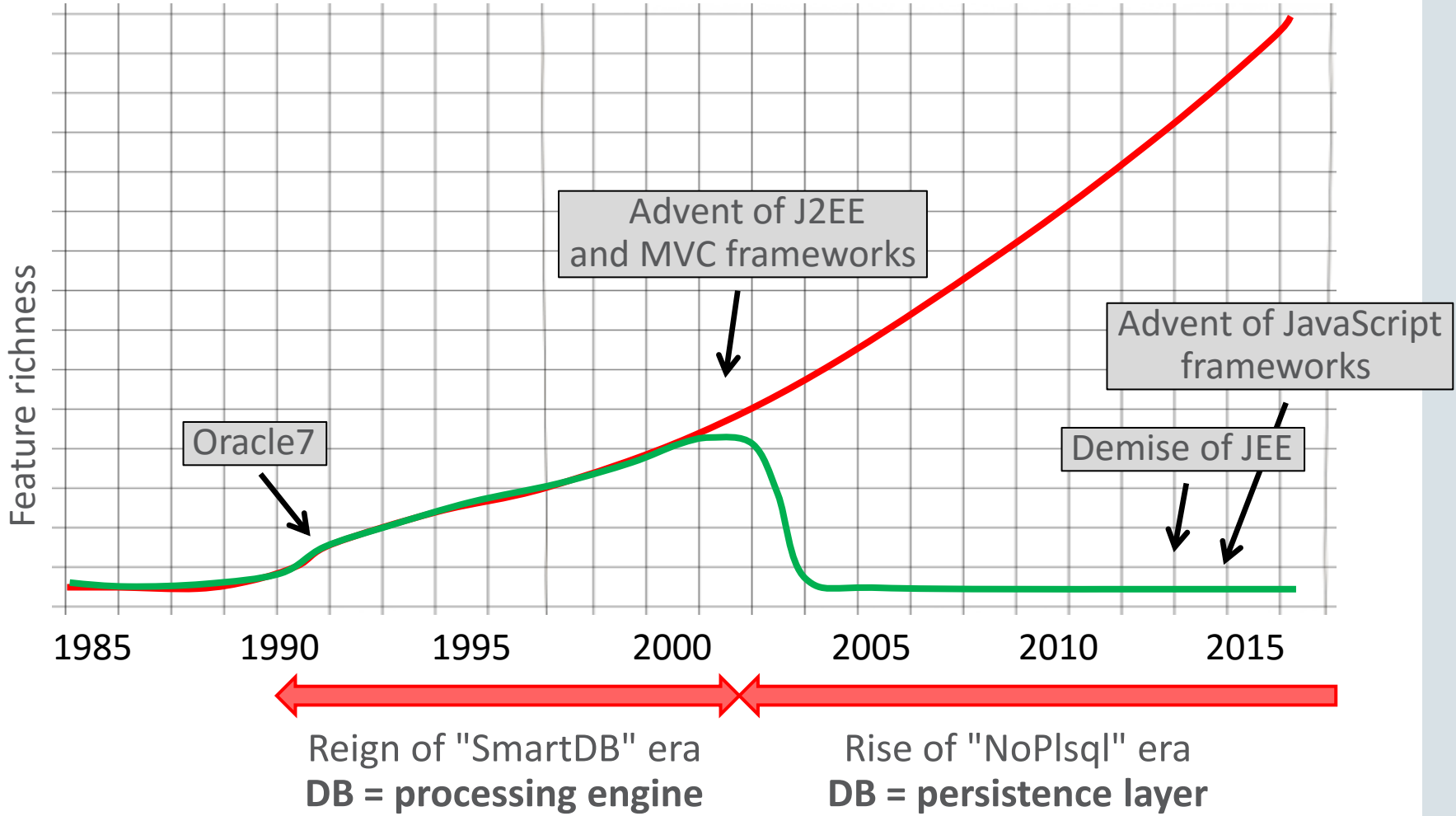
Oracle7, 8i: Database Documentation



History Observation: End of "SmartDB" Era

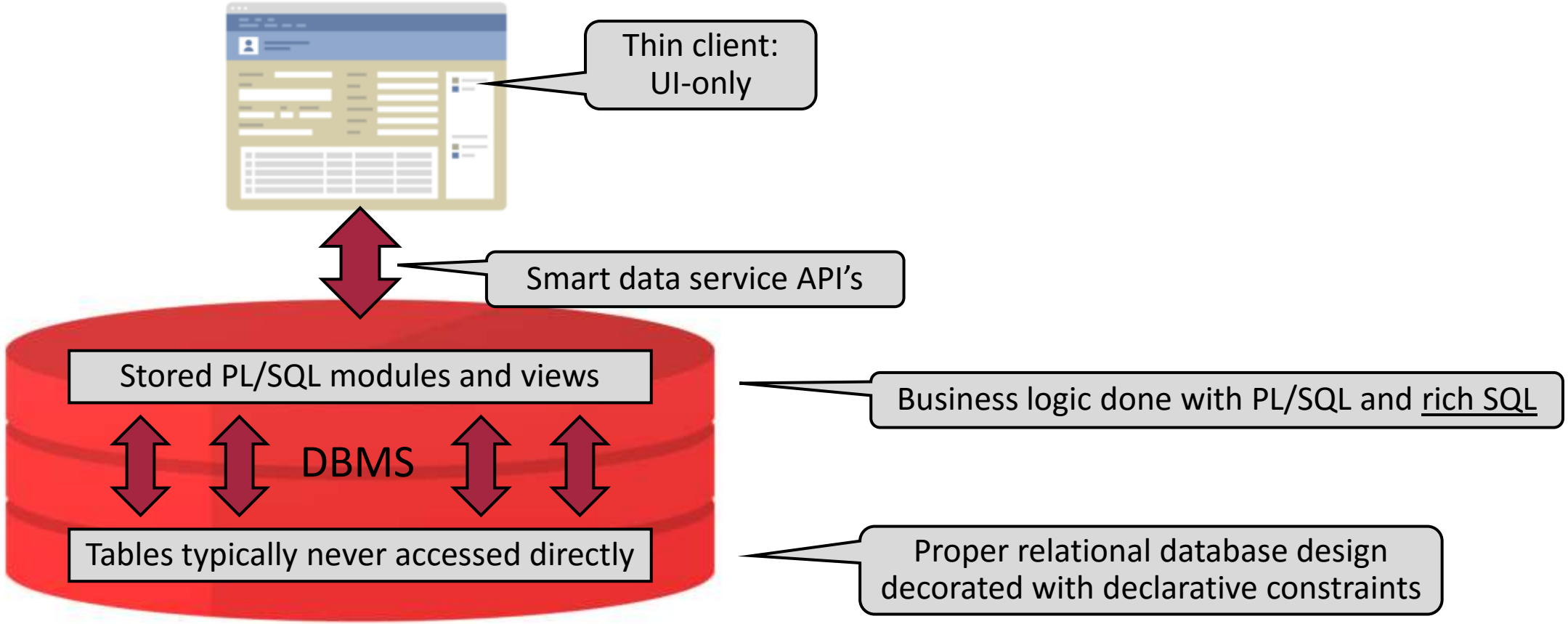
Features available in DB

DB-features used by application development



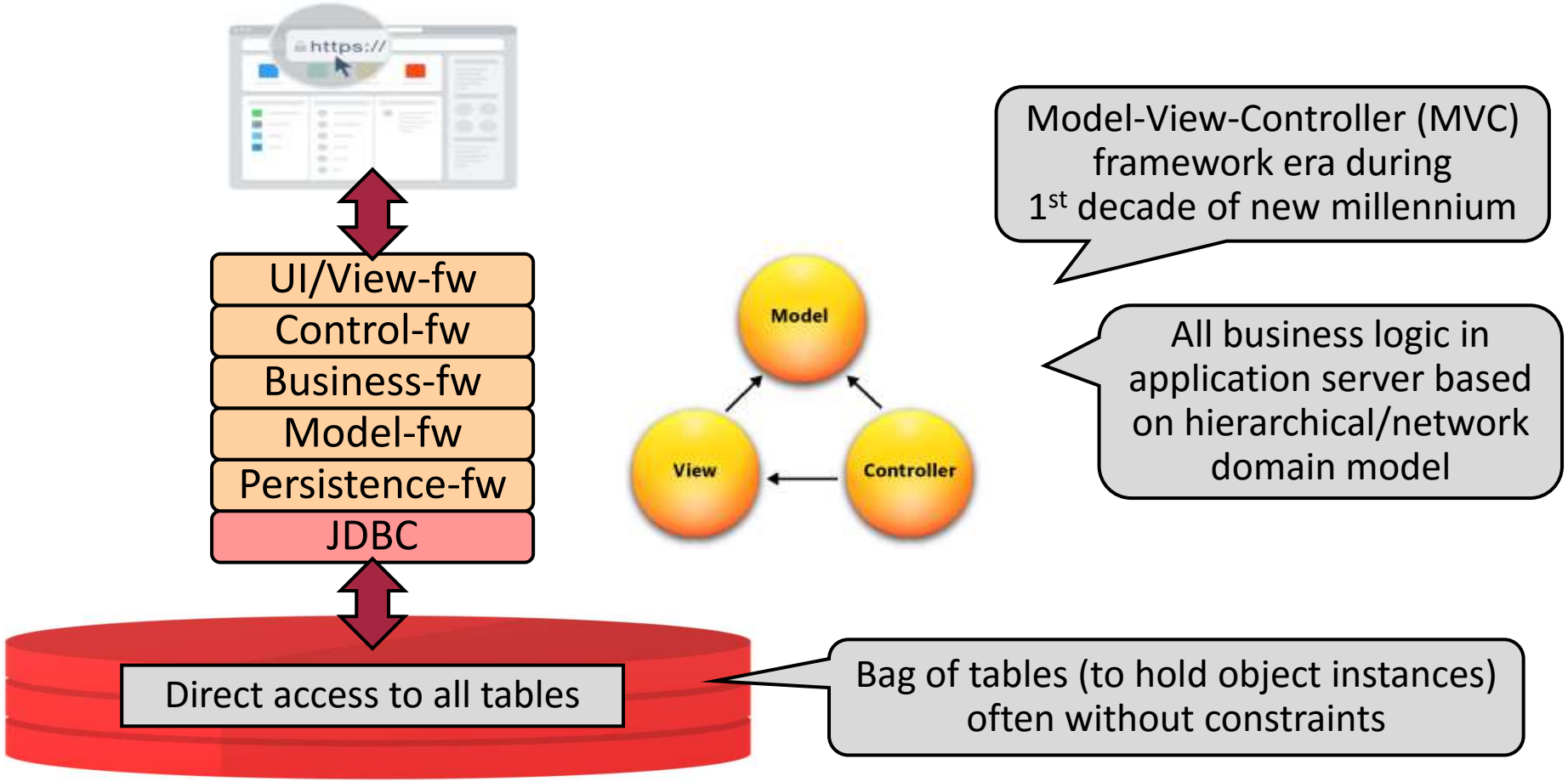
Where Were We at End of 1990's?

- Applications capitalized on database being a processing engine



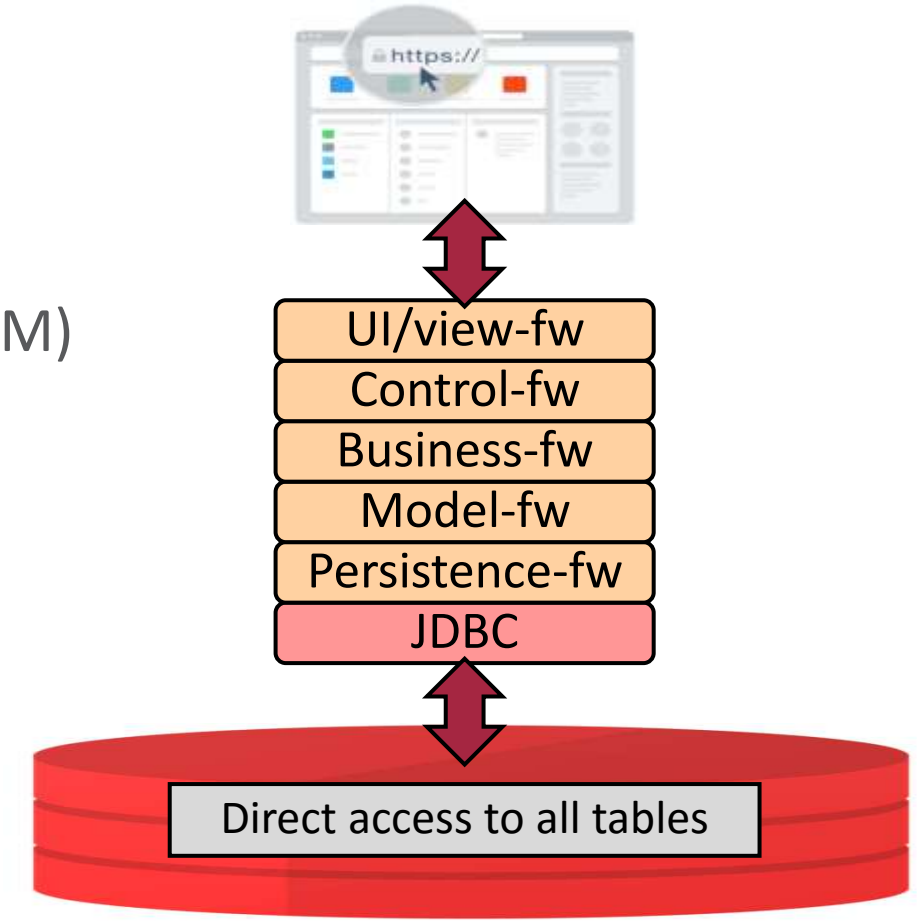
What Has Happened Since?

- Database to only fulfill persistence layer role (bit bucket)



Important Points to Make

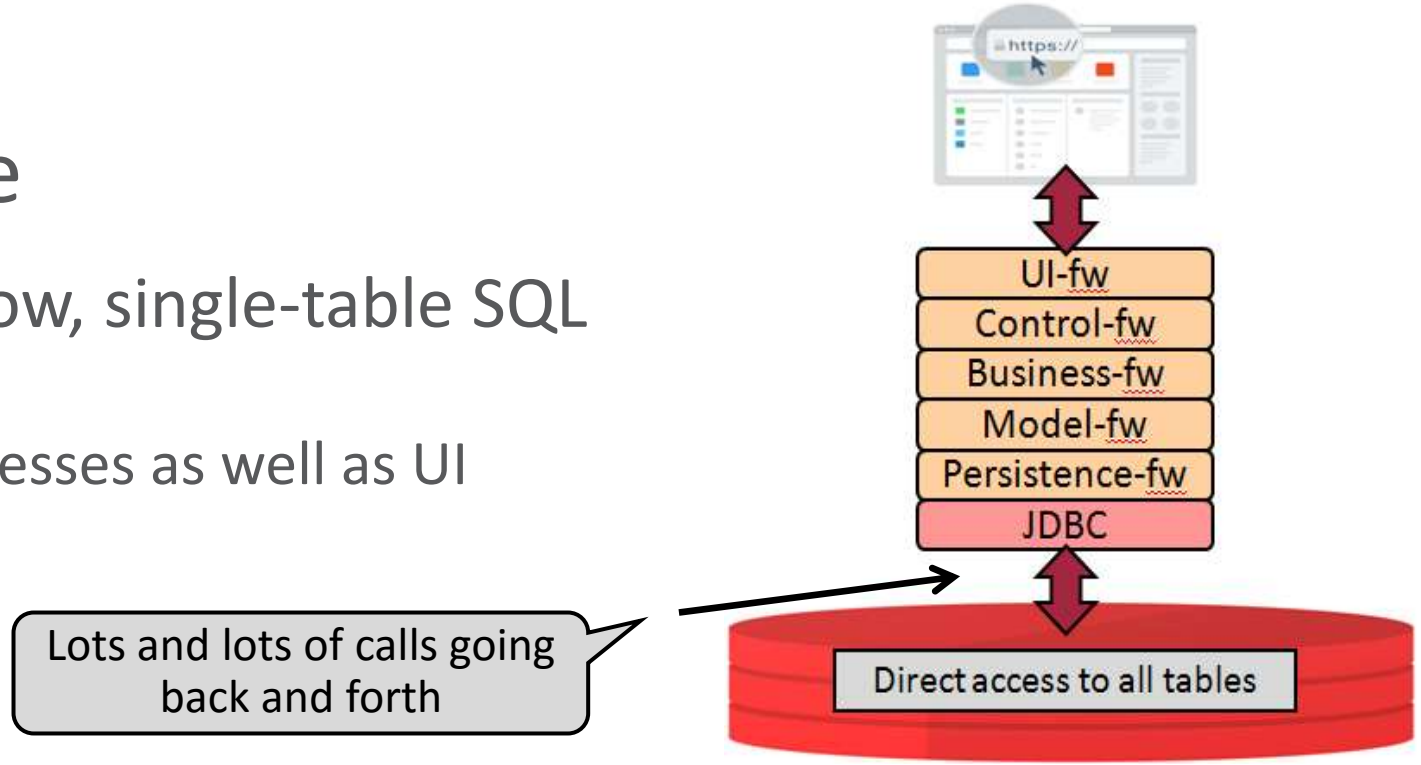
- In layered MVC approach SQL is invisible
- Almost always SQL is hidden from developers
  - Object oriented domain models are used
  - Developers invoke methods on objects
  - Objects map to tables via persistence framework (ORM)
    - Object Relational Mapping tools
- ORM's produce single-row, single-table SQL
  - In contrast to rich-SQL



Important Points to Make

- Doing everything with single-row, single-table SQL results in "chatty" applications

  Both for batches/event-driven processes as well as UI



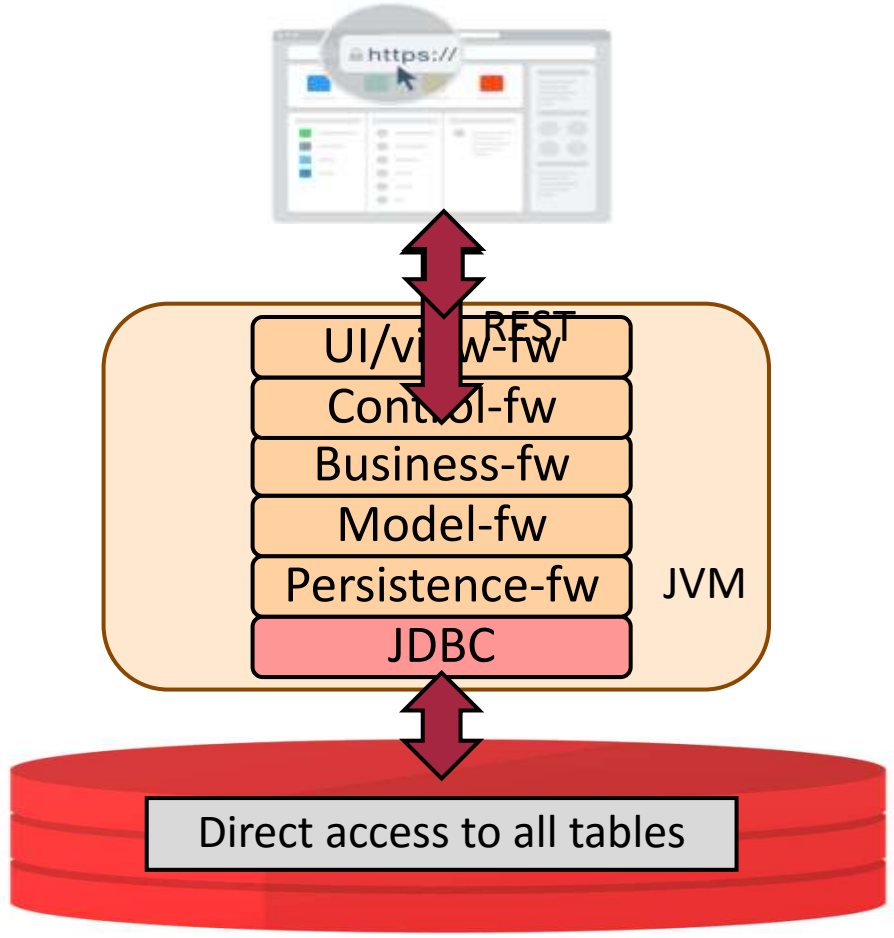
- In nineties we referred to this as "roundtrips"
  - Roundtrips were bad (for performance) then, and still are today
  - Oracle7, with stored PL/SQL, helped us mitigate this
  - By moving business logic into database

New Paradigm Shift Happening: Java → JavaScript

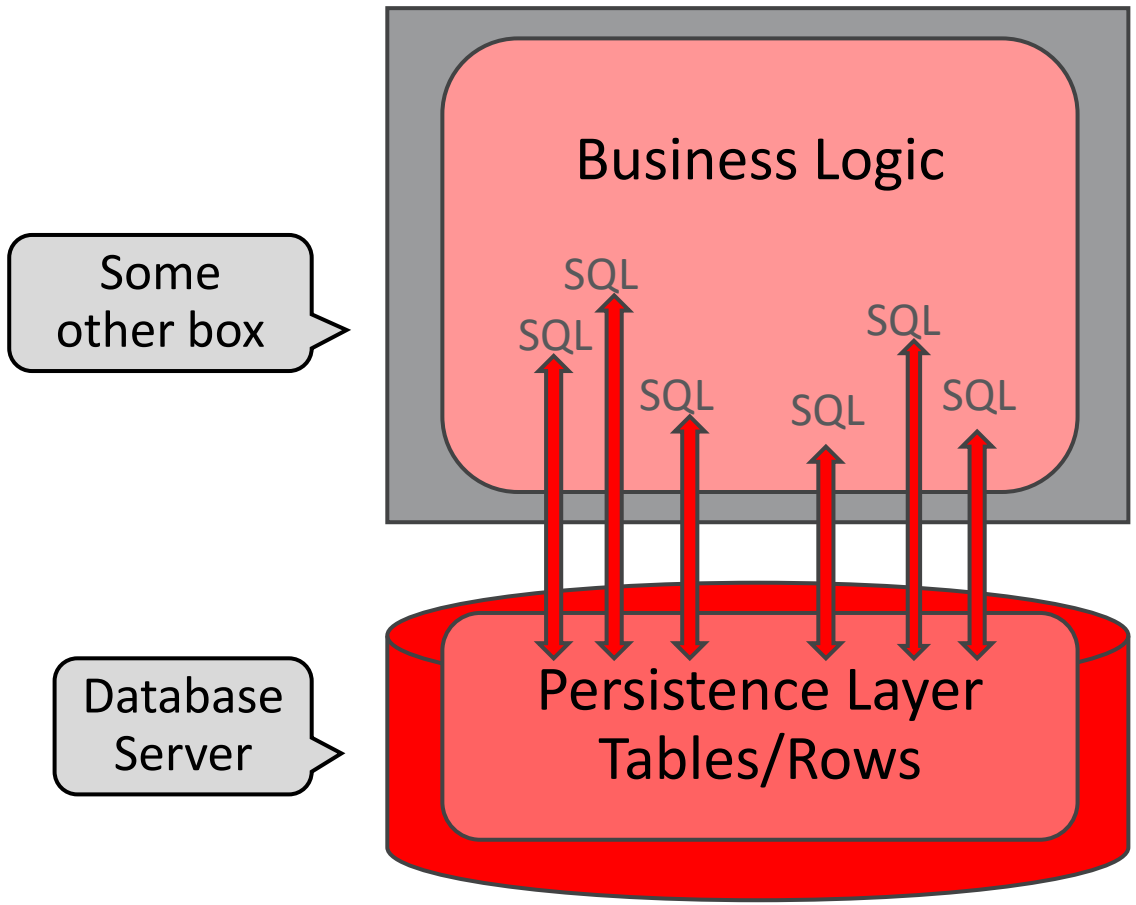
- Server-side Java MVC-frameworks approach has been ubiquitous

- New architecture is arising:
  - Browser-side JavaScript (V+C)
  - Server-side JavaScript (M)
  - REST to glue it together

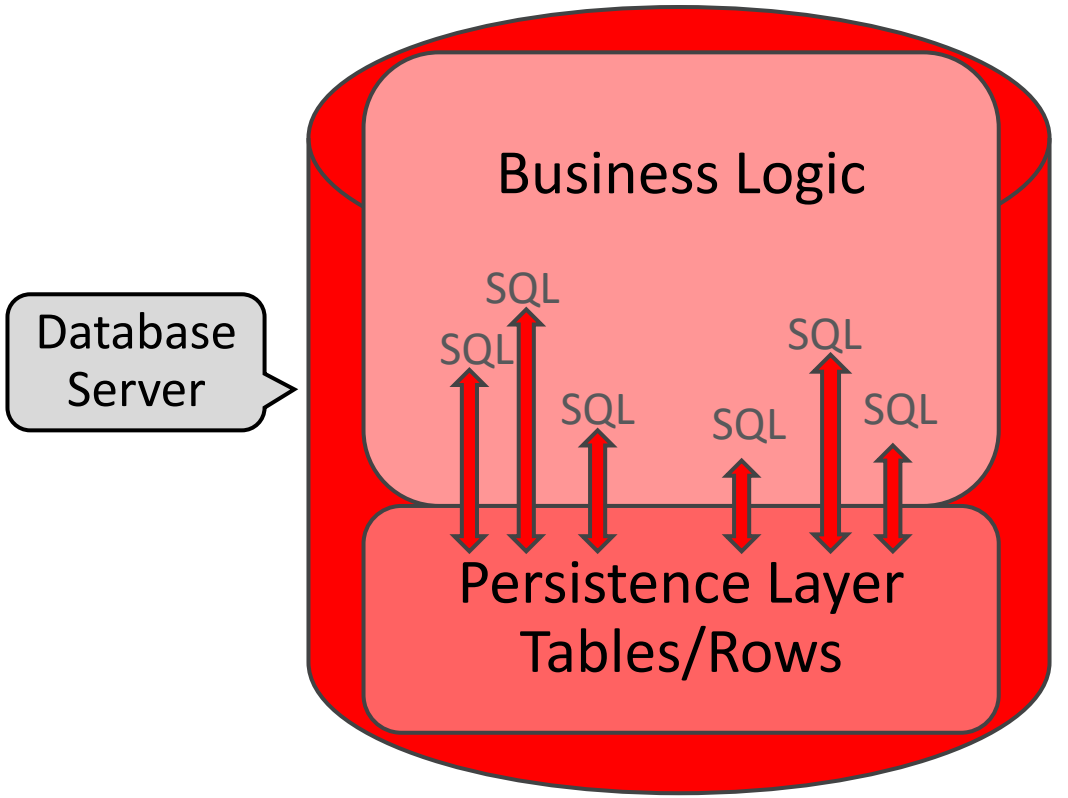
  - Database still as persistence layer



Persistence Layer Only vs. SmartDB



Processing a request causes inter-process communication per SQL-statement between application server process and database process



Processing a request takes place within a single (database) process

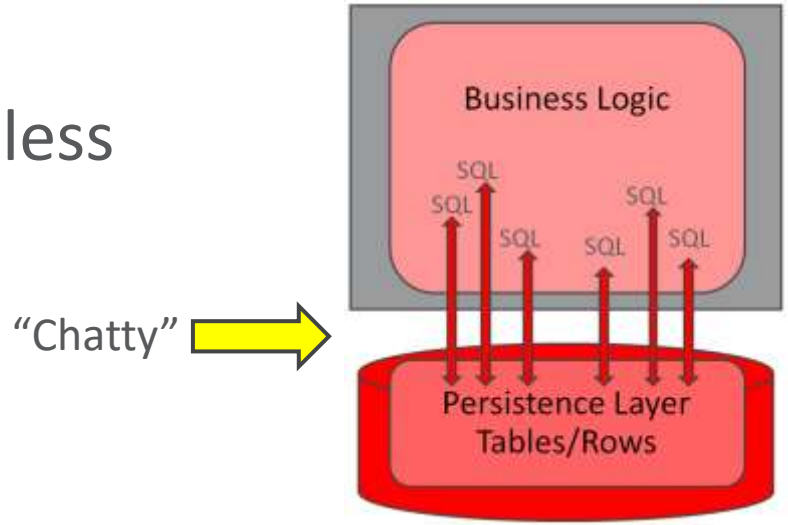
ORACLE

ORACLE
REAL-WORLD PERFORMANCE

The Performance/Scalability Argument

- "By moving business logic out of database, we put less load on database"

  → Sounds feasible, right?

"Chatty" 

- However:
  We introduce a lot of communication between major components of our infrastructure

  → Is this for free?

Performance and Scalability

- NoPlsql's promise:
  - Yes, it’s more chatty, but we’re mitigating that
  - We’ll get the data from DB once into mid-tier caches
  - Then re-use cached data many times in horizontally scalable mid-tier servers
  - Write data back to db once

- However in real-world:
  - Most NoPlsql applications seem unable to deliver on above promise
  - Applications *always* chatty: not able to use caches effectively?
  - Where's the advantage then?

Next up

1. A bit of history and observations
2. Debunking performance and scalability argument
3. Some closing remarks

Next Section's goal:

Create awareness of rather huge inefficiencies introduced by using the database just as a persistence layer

Summary Here

- Full story at Oracle Learning Library channel on youtube

  https://www.youtube.com/watch?v=8jiJDflpw4Y

  Search: “Toon Koppelaars"

Our Experiment

- Event-processing module based on real-world example
  - Built using SmartDB approach: PL/SQL stored procedure with embedded SQL
  - Built using NoPlsql approach: Java with embedded SQL on top of (thin) JDBC

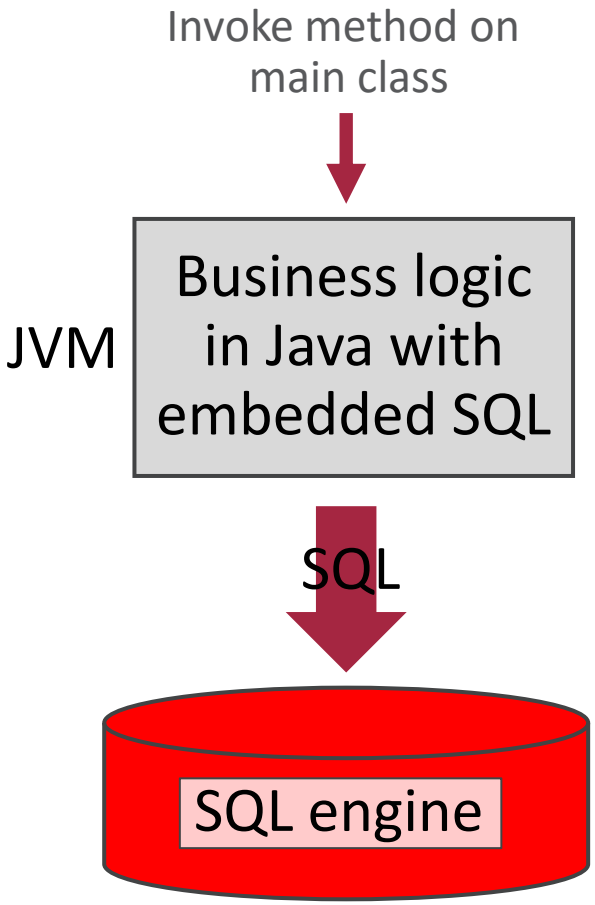
  Both built using row-by-row pattern

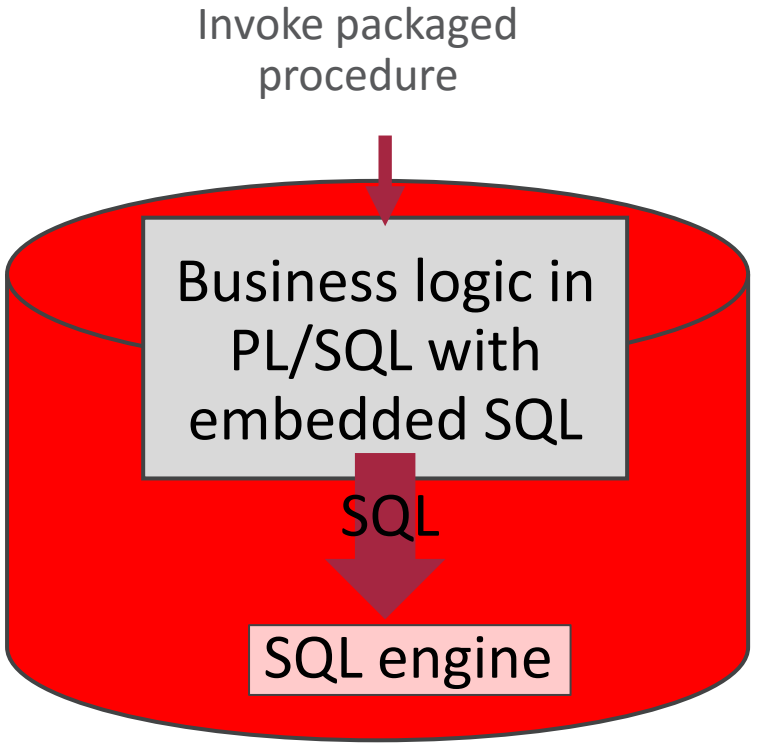
- Simple business logic: if-then-else, looping

- With typical load profile that we see all the time:
  - Many single-row SQL statements, mix of reads and writes
  - Index maintenance

Both runs execute the same 5M SQL-statements

Java/JDBC versus PL/SQL





ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Java/JDBC versus PL/SQL

Elapsed-time: **11 minutes**

JVM

Business logic in Java with embedded SQL

217 CPU seconds

SQL

SQL engine

204 DB-CPU seconds

437 DB-CPU seconds

Elapsed-time: **3 minutes 30 seconds**

Business logic in PL/SQL with embedded SQL

SQL

SQL engine

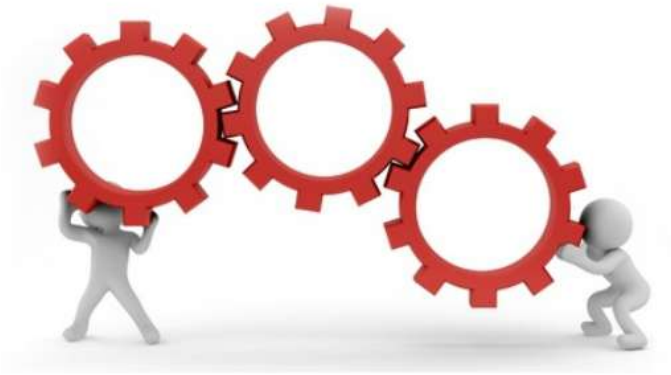
ORACLE

ORACLE REAL-WORLD PERFORMANCE

Reversed Effect…

- Runtime 3X more → #NoPlsql performs worse
- DB resource-usage 2X more → #NoPlsql scales worse

- Seems “performance/scalability” argument is plain wrong?



Exact Same Row-by-row SQL: Why The Huge Difference?

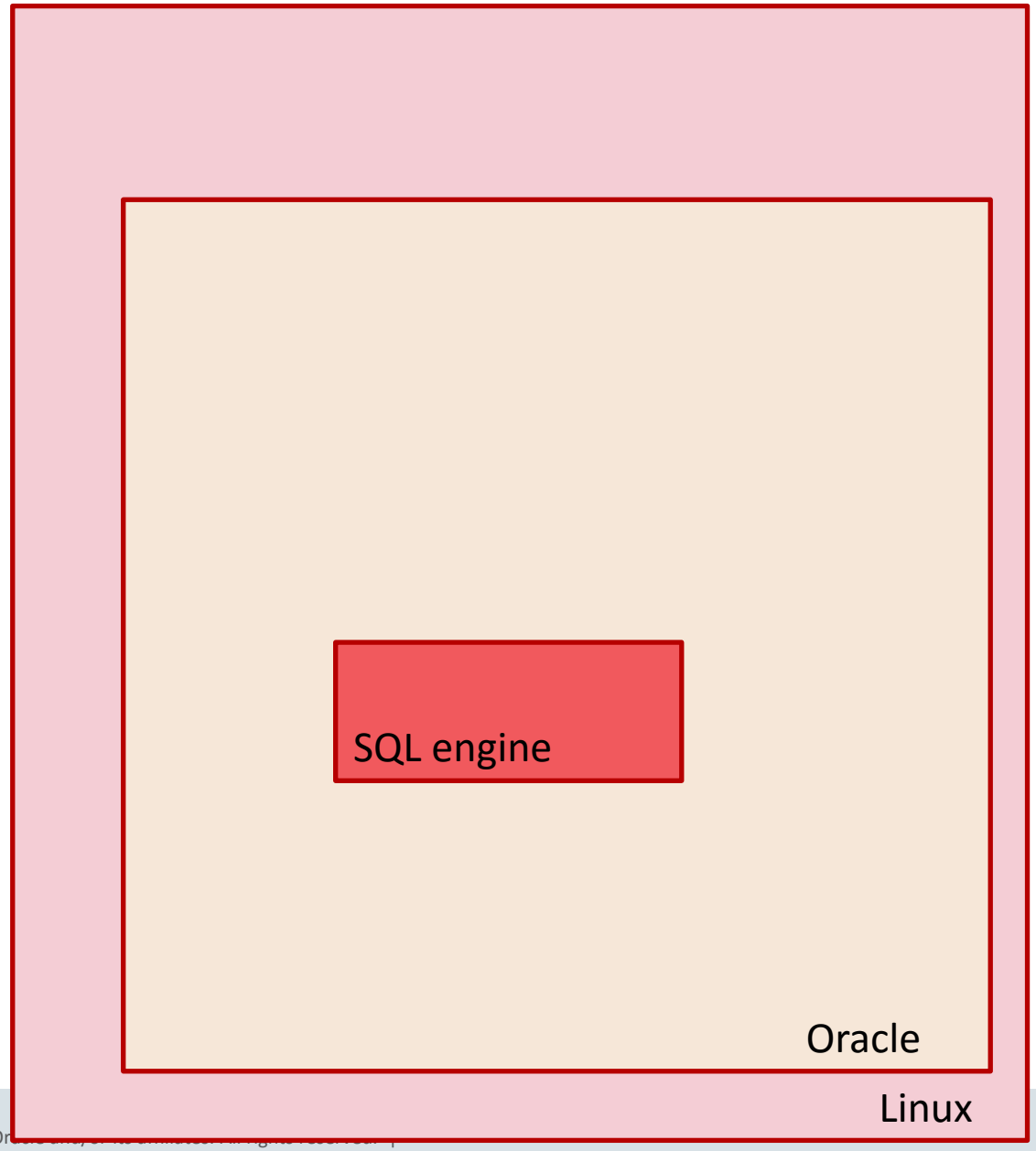
- "The Living Room" analogy
- With SmartDB:
  - PL/SQL is already in living room, which is where SQL lives

- All other languages need to enter from “outside”
  - Go through front door, traverse hall, enter living room

  And apparently this is *not* for free

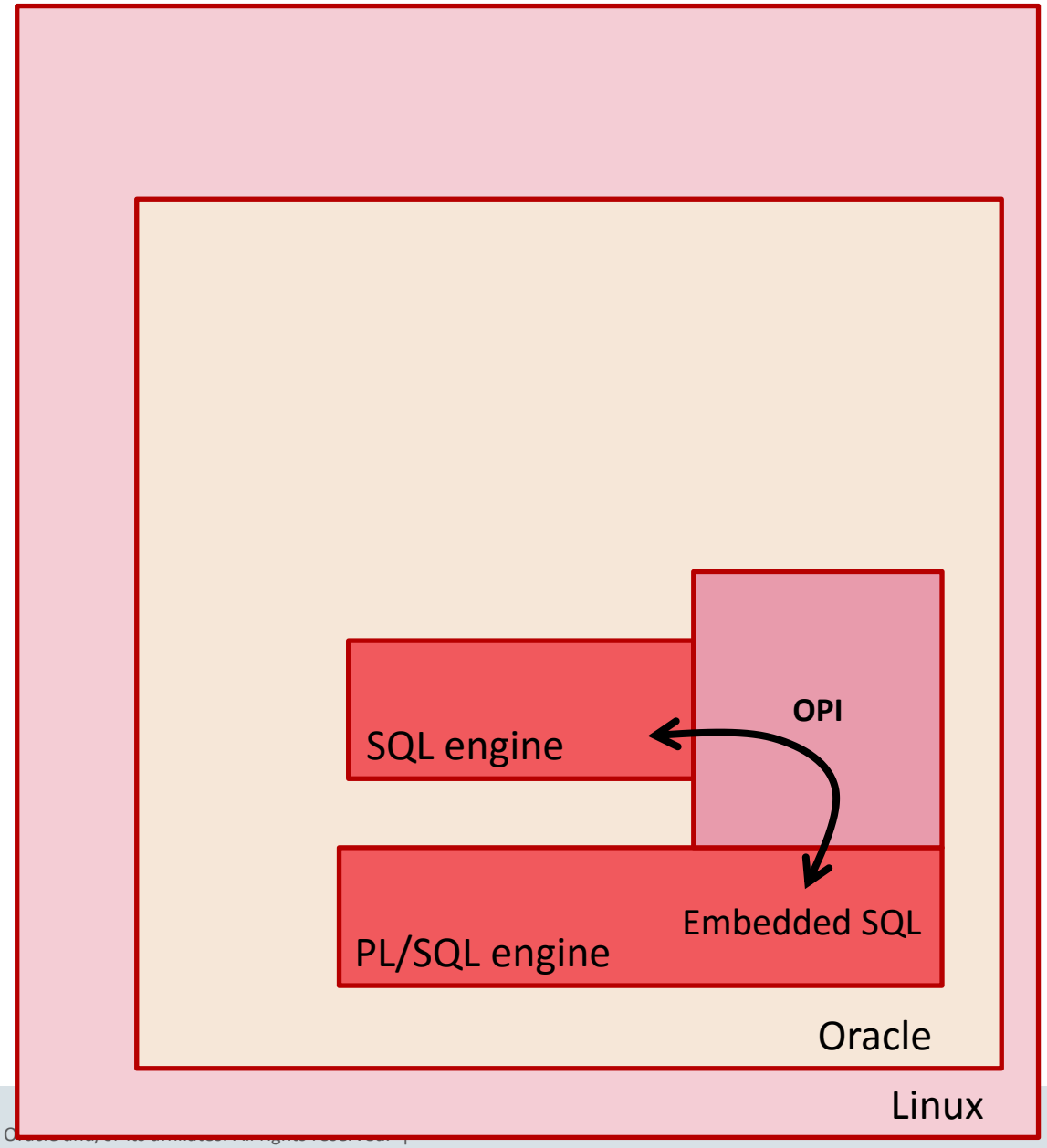
The Living Room

- SQL engine



The Living Room

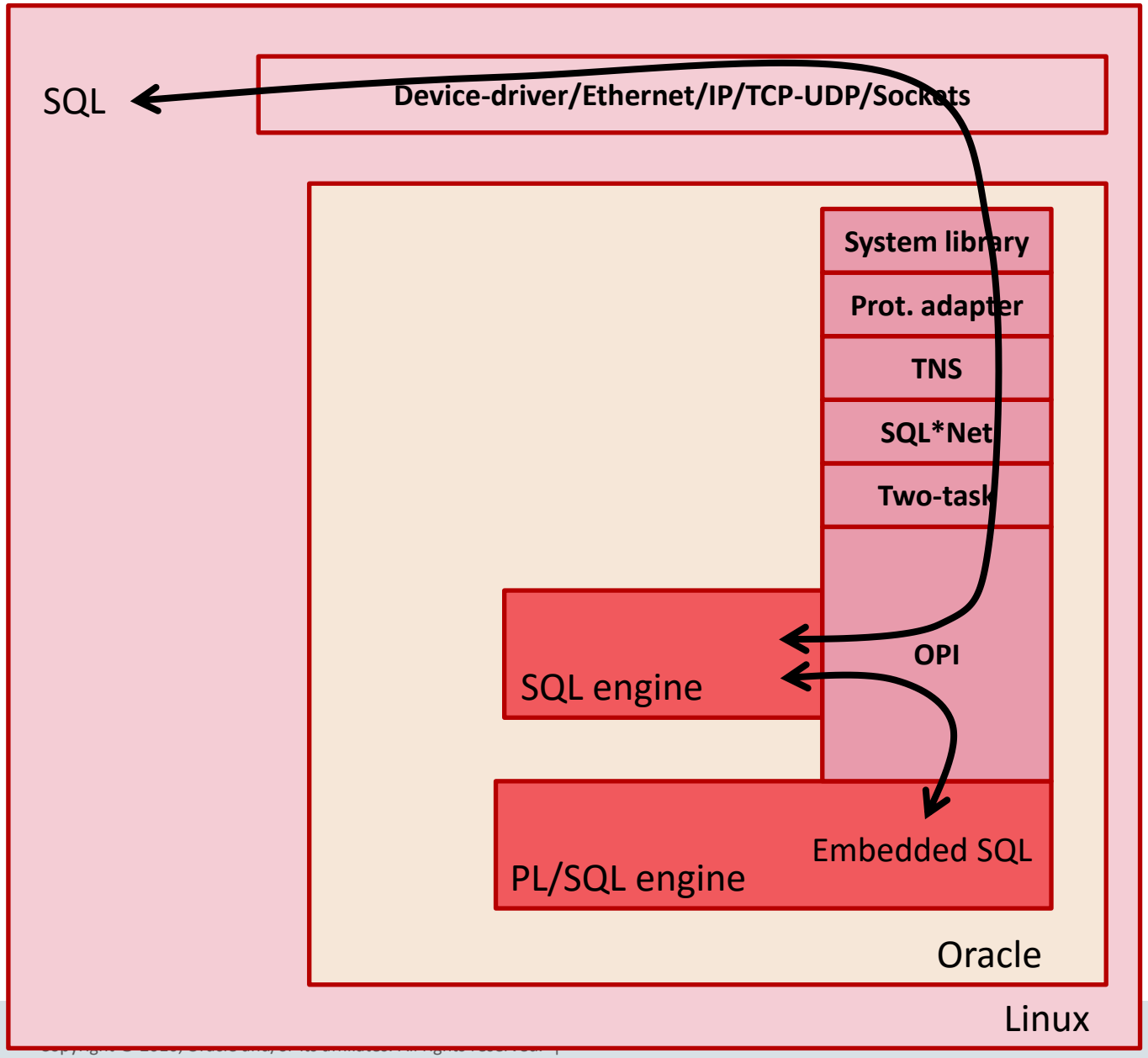
- SQL engine
  - Accessible via OPI layer
  - Oracle Program Interface
- PL/SQL directly calls OPI



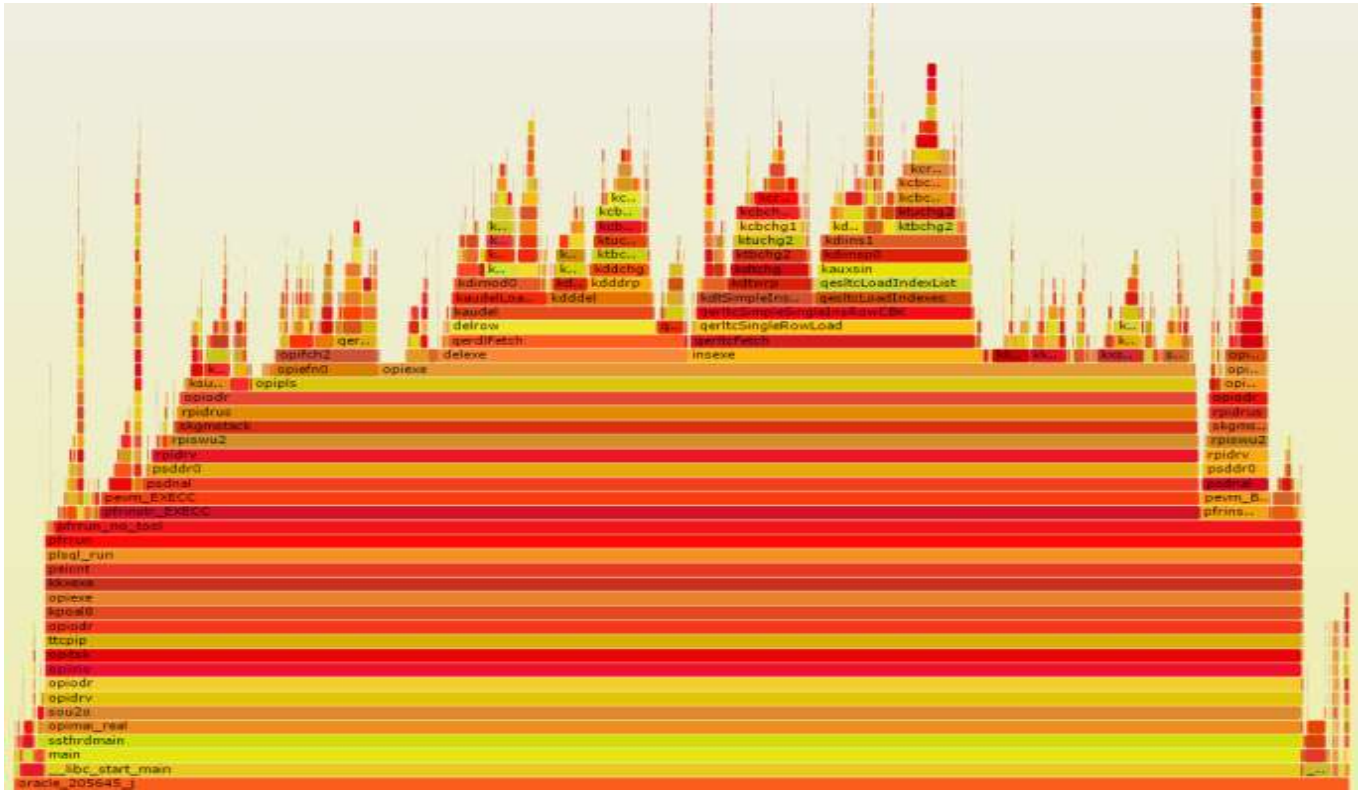
The Living Room

- Outside SQL route:
  - OS network/ipc layers
    - Front door, doormat
  - Net/TNS/TT layers
    - hallway
  - OPI

➔ More code path:
For row-by-row SQL,
you notice this overhead



Researched This Through FlameGraphs



See the video…

Reason #1

- In #NoPlsql every SQL-statement gets hit with fixed DBMS-entry cost
- In #SmartDB there is no entry required for SQL, it's already in there

Research showed: 40-50% addititional CPU-cycles per SQL-statement
Remember, we're dealing with row-by-row SQL here

- But that's not the observed >2X increase in DB-Time...

Reason #2: CPU Efficiency

Reason #2: CPU Efficiency

- Modern CPU cores are complex factories (just like RDBMS)
  - It's best to stay in factory as long as possible
  - Getting off and back on CPU is very expensive

    Process context-switching is actually most expensive CPU operation

- SmartDB approach always has the least process context-switches
- NoPlsql deschedules at least 5M times (once per SQL statement)

  Causing CPU to have to execute additional micro-operations

- What this means is: NoPlsql runs on slower DB CPU's

Researched through CPU profiling

Combined: NoPlsql Puts >2X Load On DBMS

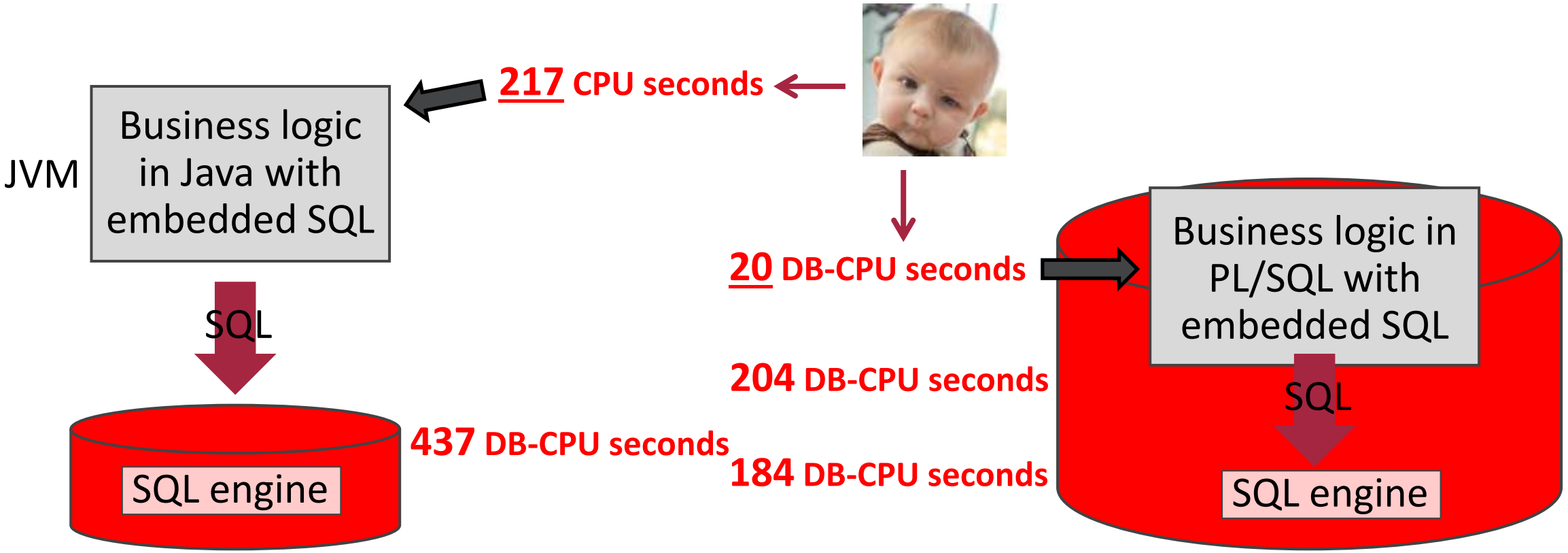
- Reason #1: Overhead per SQL statement
- Reason #2: Inefficient use of database CPU

One More Point to Make…

- Apart from spending time executing SQL

- We're also spending time executing the language from which SQL gets submitted
  - In #NoPlsql approach we're spending time in Java
  - In #SmartDB approach we're spending time in PL/SQL

Required CPU For Getting the Business Logic Done



10X!

- Researched through FlameGraphing the JVM
- Analogy: not only do you have to come into (DB) house from outside
  You also first have to exit your (JVM) house for every SQL statement

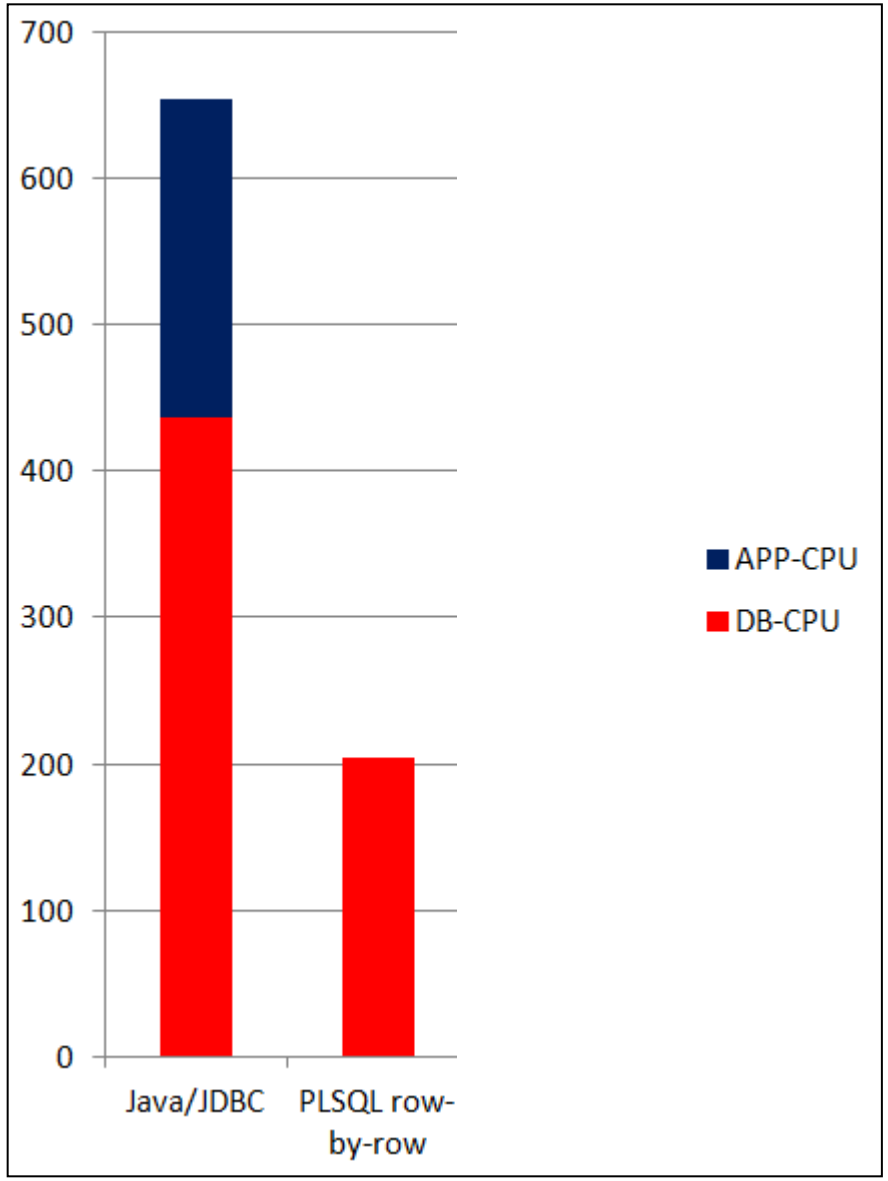
- Majority of time spent in JVM is in:
  1. Executing JDBC code-layers
  2. Getting in and out of JVM
  3. Other JVM-specifics (JIT compilation, Garbage Collection, ...)

All three simply do not exist in SmartDB approach

In Summary

	Java/JDBC	PLSQL row-by-row
DB-CPU	437	204
APP-CPU	217	0
Elapsed	660	204

You gain a little (-20),
but loose a lot (+250)



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Next up

1. A bit of history and observations
2. Debunking performance and scalability argument
3. Some closing remarks

Closing Remarks

- The implication of all this
- SQL isn’t accidental
- My application is too complex
- Database = API provider
- Beware of risks if you go SmartDB

The Implication Of All This

- "NoPlsql scales worse" → essential to understand this point

- For every application
  - There's always a bottleneck, right?
  - And it's almost always the database, right?

- Well,
  - With SmartDB, the database will be the bottleneck later
  - With SmartDB, you could run your application on less hardware
  - With SmartDB, you'll likely spend less on Oracle licenses

SQL Isn't Accidental: Au-Contraire, It's Fundamental

- I've not talked about moving from row-by-row to set-based SQL

- Very often there is opportunity to embrace set-based SQL
  - You specify the "what" and DBMS figures out "how"
    - Speedup of development
  - Replacing row-by-row with set-based SQL also delivers further speedups
    - 10X, 100X often achievable

- In a way you move business logic into SQL (rich SQL)
  - And stop using database as a record-store

SQL Isn't Accidental: Au-Contraire, It's Fundamental

- Why is there almost always opportunity to go set-based?
  - An application = model of part of the real-world about which we wish to reason
  - We reason via accessing the underlying database design through (rich) SQL

  Fact: SQL is based on logic and set theory

  Fact: Logic and set theory are based on natural language, particularly the part of it that deals with reasoning

- So we reason in model using language that was based on how we reason in the real-world
  - Ergo, SQL fundamentally fits what we want to achieve

My Application Is Too Complex

- "No fit with your experiment":
  - In experiment ratio business-logic⇔SQL was: 10⇔90
  - My application is the other way around: 90⇔10
    - 90⇔10, NoPlsql ? Remember that 10X overhead (btw. worse if frameworks used)...
    - 90⇔10, SmartDB ? I'd like to see that...

- "I cannot do my application logic in SQL and PL/SQL"
  - Both SQL and PL/SQL have become incredibly rich
  - Given our context (transactional business applications) and SQL's fundamental fit, it would be strange if your logic cannot be dealt with

Often This Is The Issue

- A mindshift is required:

- You need to start thinking in “processing data”
- Instead of “interacting with objects”

- A relational database design should be your frame of reference
- And not an (object oriented) domain model

Beware Of Risks If You Go SmartDB

- Make sure you involve people
  - Who've done this before
  - Who think "processing data"
  - Who are experienced in designing databases
  - Who know full power of SQL and PL/SQL

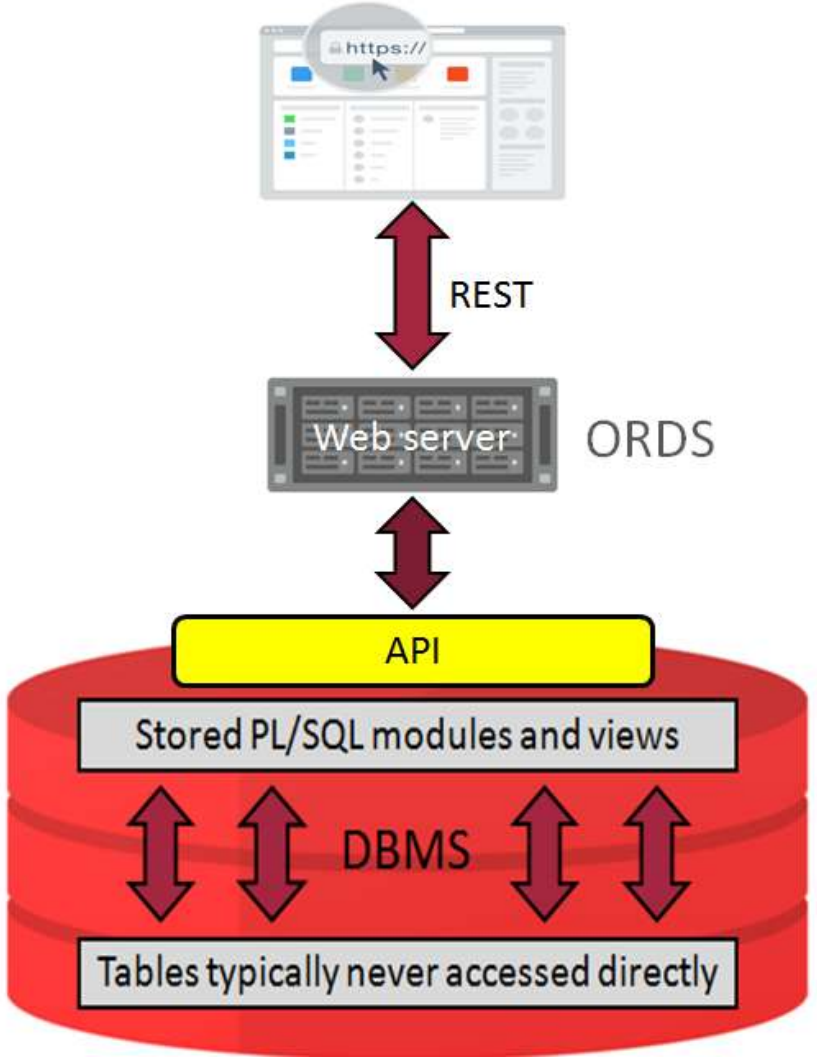
- If you're new to this: obviously start small

Reach Out to Active SmartDB Community

- Oracle Technology Network, Database, SQL and PL/SQL forums: https://community.oracle.com/welcome
- Ask The Oracle Masters: https://asktom.oracle.com
- Oracle Dev Gym: https://devgym.oracle.com/
- Stack Overflow: https://stackoverflow.com/questions/tagged/plsql https://stackoverflow.com/questions/tagged/oracle
- Oracle-l maillist: https://www.freelists.org/list/oracle-l

In Summary

- Turn DB into smart API provider:
  - Database to provide transactional and non-transactional endpoint data-services
  - Through packaged stored procedures
  - Which can easily be REST-enabled for consumption



Lot Not Covered

- Kept this sweet and short

- Main goal:

  Create awareness of rather huge inefficiencies introduced by using the database just as a persistence layer

- The performance and scalability argument almost always is a misconception

Integrated Cloud

Applications & Platform Services