SmartDB: A Database Centric Approach to Application Development

Part 1: What?

Toon Koppelaars
Real-World Performance
Oracle Server Technologies

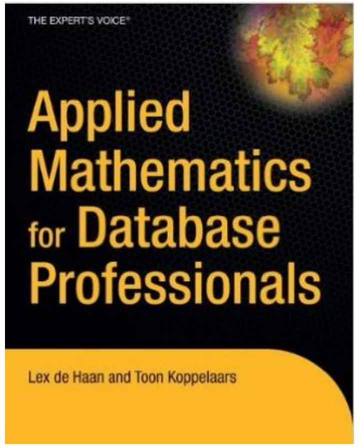
Safe Harbor Statement

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.

About Me

- Part of Oracle eco-system since 1987
  - Have done and seen quite a lot of application development
  - Database design, SQL and PL/SQL
- Big fan of “Using Database As a Processing Engine”
  - Not just as a persistence layer
- Member of Oracle’s Real-World Performance Group

@ToonKoppelaars


THE EXPERT'S VOICE
Applied Mathematics for Database Professionals
Lex de Haan and Toon Koppelaars

Terminology Over The Years

- Thick DB
  - Translated from Dutch, first used in “A First JDeveloper Project”, Oracle World 2002
- Fat DB
  - Because “thick” has other meanings
- “Phat” DB
  - More hip
- The Helsinki Declaration
  - Java-conference @Helsinki, resulted in TheHelsinkiDeclaration.blogspot.com
- Using database as “Processing Engine”
  - That’s what we call it inside Real-World Performance group
- SmartDB
  - Joint PM proposal new name

Agenda This Afternoon

- Part 1: SmartDB, What Is It and Why Would You Want to Consider It?
  - Break
- Part 1: SmartDB, What Is It and Why Would You Want to Consider It?
  - Break
- Part 2: SmartDB, How, What Are Critical Success Factors?

Roadmap Part 1

1. Business Logic
2. What Is SmartDB?
3. Some History and Observations
4. Issues With Other Approaches
5. Debunking Performance and Scalability Argument
6. Closing Remarks

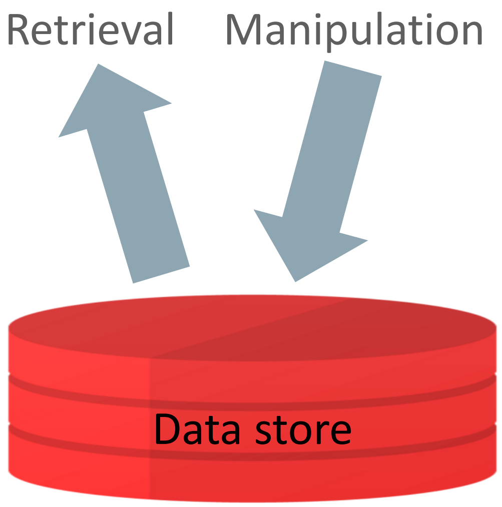
Roadmap Part 1

1. Business Logic
2. What Is SmartDB?
3. Some History and Observations
4. Issues With Other Approaches
5. Debunking Performance and Scalability Argument
6. Closing Remarks

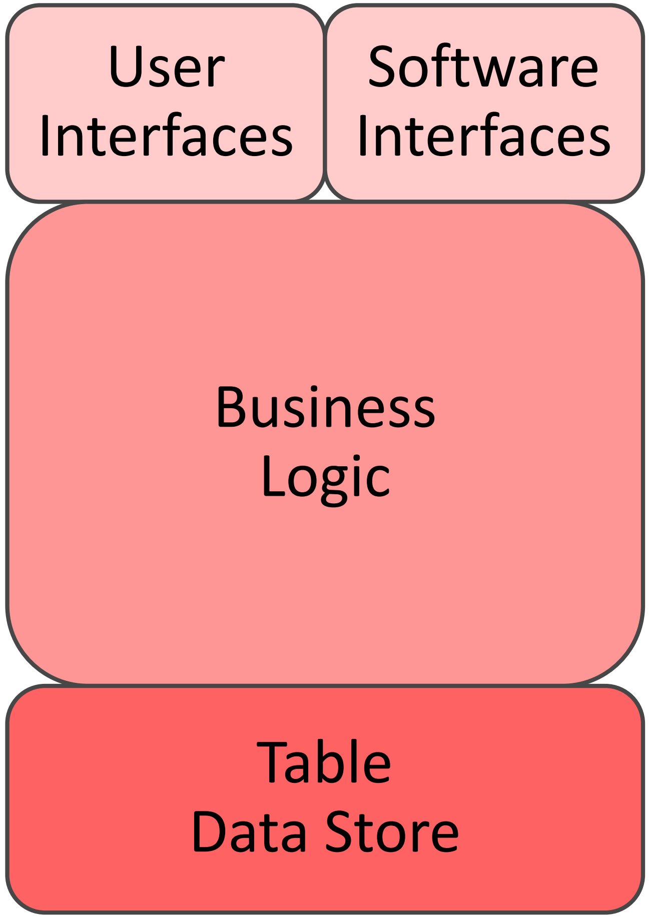
ORACLE

Context of This Presentation

- *Data intensive transactional business applications*
  - A **Data store** as foundation
  - Relational tables in Oracle database
  - Much business functionality on top
    - **Retrieval of data (select)**
    - **Manipulation of data (insert/update/delete)**
  - User interfaces, batches, reports, services to other application systems
  - Potentially many users



Transactional Business Applications



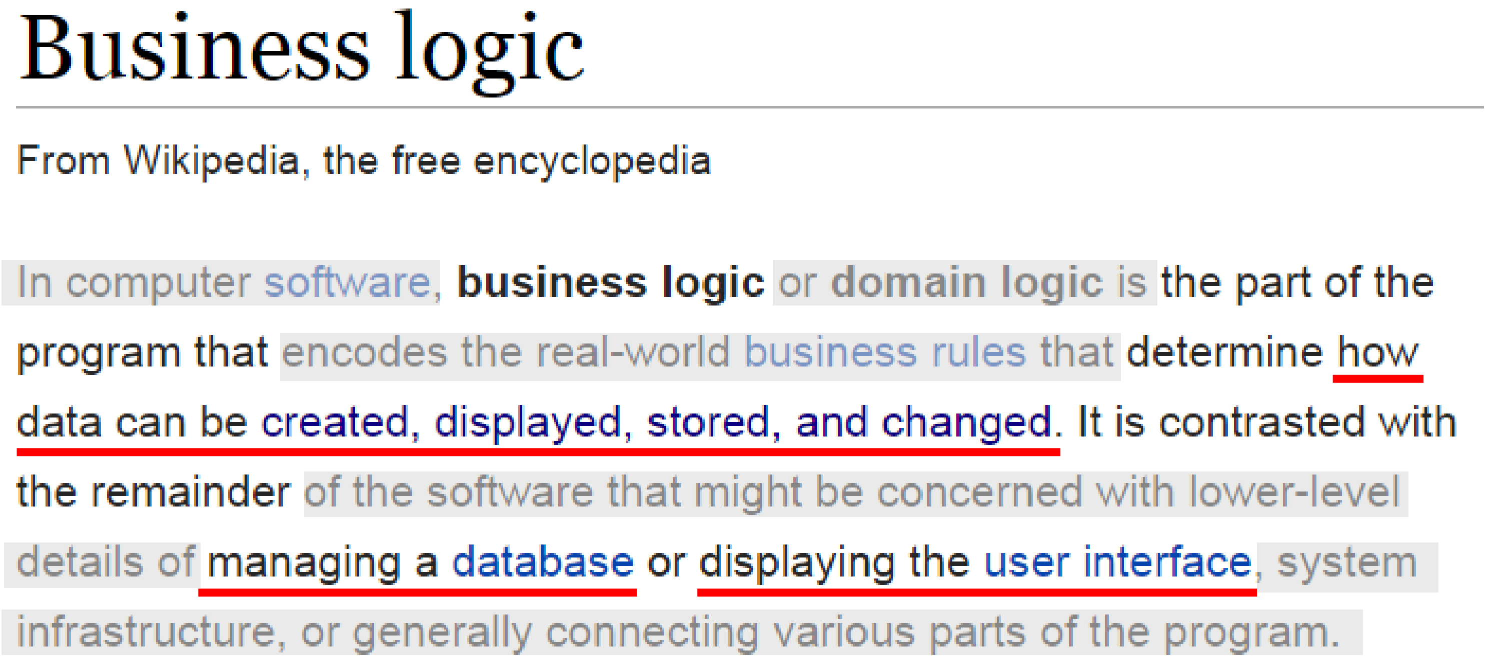
- Conceptually 3 tiers
  - Exposed functionality via services
    - GUI's for human interaction
    - REST, or otherwise, for software interaction
  - Internals
    - Business logic
    - Data store, relational database

Transactional Business Applications

- A big component of these applications is “Business Logic”

- What is “Business Logic”?

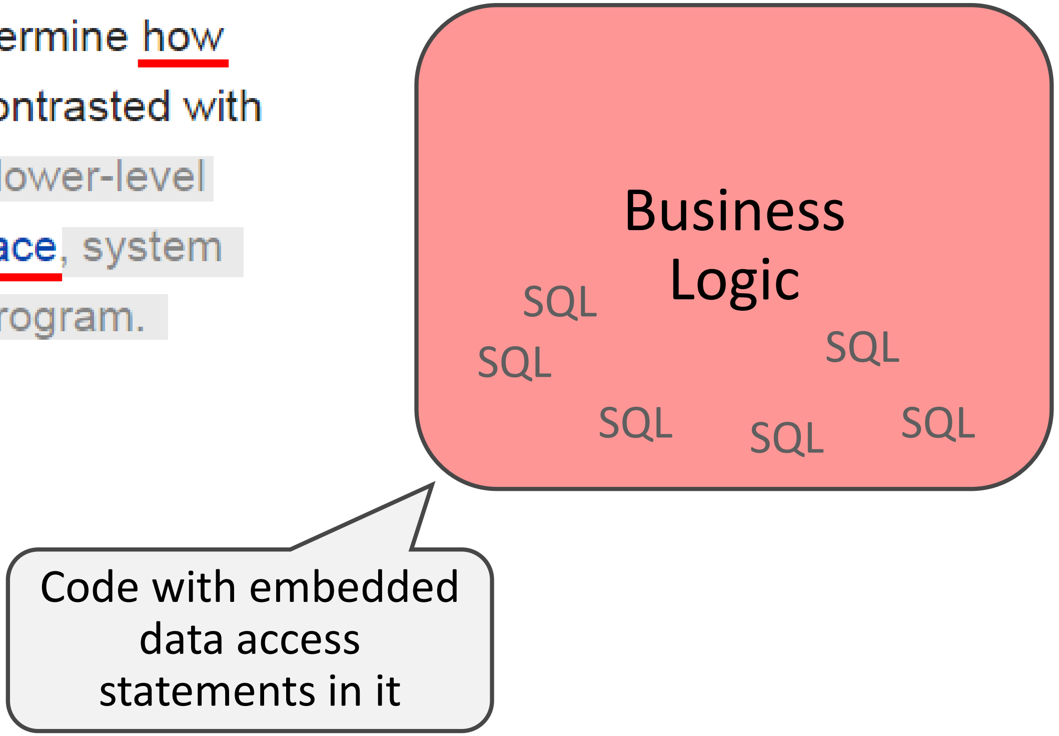
Wiki



Business logic

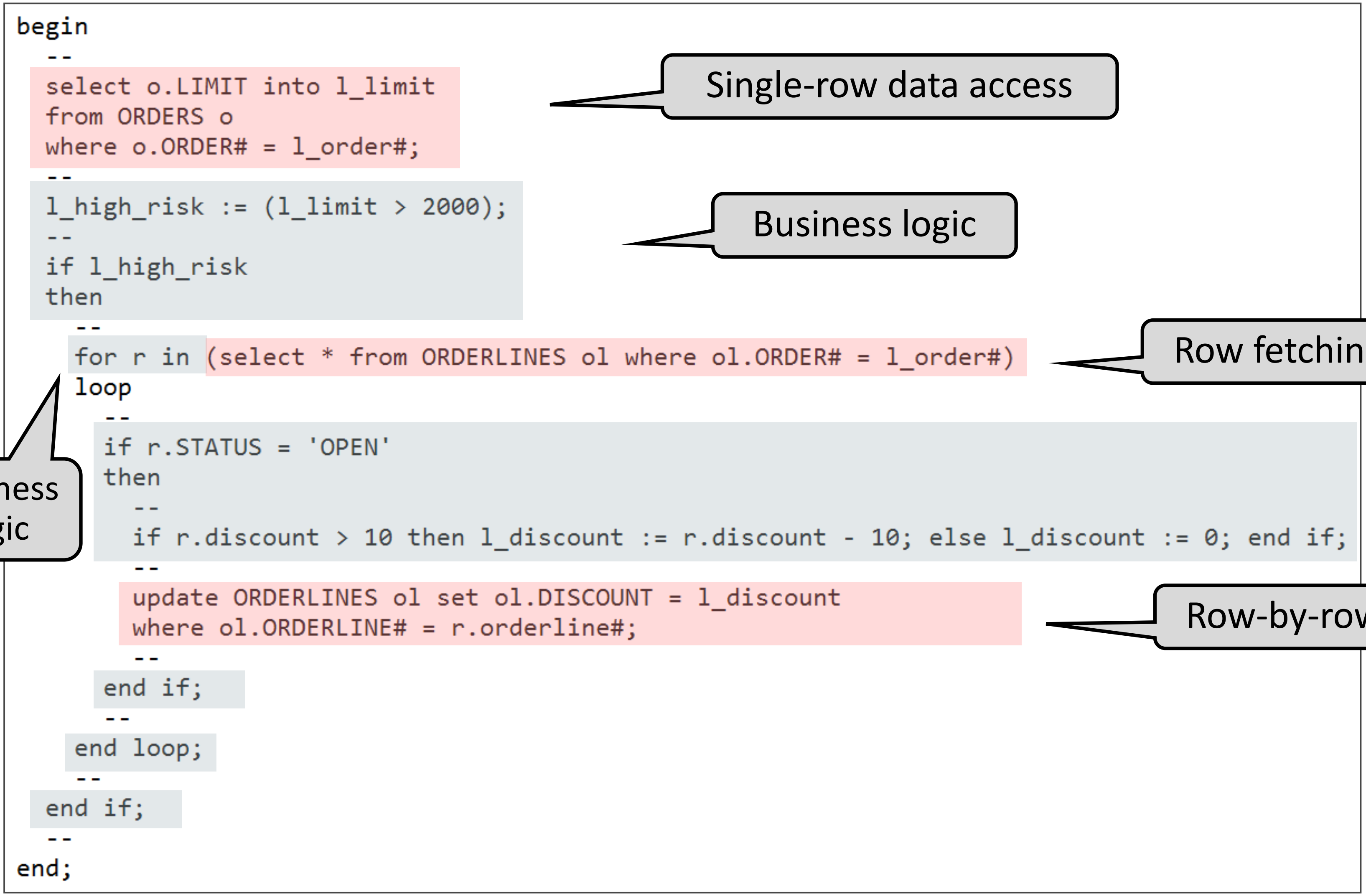
From Wikipedia, the free encyclopedia

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

Example Business Logic: Code With Embedded SQL



```
begin
 --
 select o.LIMIT into l_limit
 from ORDERS o
 where o.ORDER# = l_order#;
 --
 l_high_risk := (l_limit > 2000);
 --
 if l_high_risk
 then
  --
  for r in (select * from ORDERLINES ol where ol.ORDER# = l_order#)
  loop
   --
   if r.STATUS = 'OPEN'
   then
    --
    if r.discount > 10 then l_discount := r.discount - 10; else l_discount := 0; end if;
    --
    update ORDERLINES ol set ol.DISCOUNT = l_discount
    where ol.ORDERLINE# = r.orderline#;
    --
   end if;
   --
  end loop;
  --
 end if;
 --
end;
```

Single-row data access

Business logic

Row fetching (data access)

Business logic

Business logic

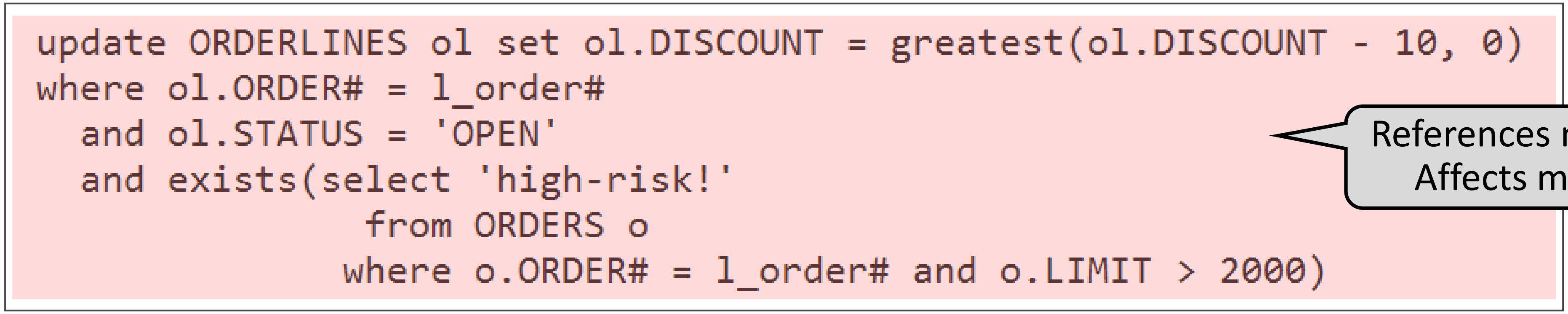
Row-by-row updating (data access)

Conditional if-then-else and looping logic

Primitive data access (single table, single row)

Example Business Logic: All in SQL

Set-based data processing
Aka “rich”SQL



```
update ORDERLINES ol set ol.DISCOUNT = greatest(ol.DISCOUNT - 10, 0)
where ol.ORDER# = l_order#
  and ol.STATUS = 'OPEN'
  and exists(select 'high-risk!'
             from ORDERS o
            where o.ORDER# = l_order# and o.LIMIT > 2000)
```

References multiple tables
Affects multiple rows

Point to be made:

- Business logic can appear:
  - As code-lines in some programming language that issues simple (poor) SQL
  - Inside (set-based) SQL itself

My Take on "Business Logic"

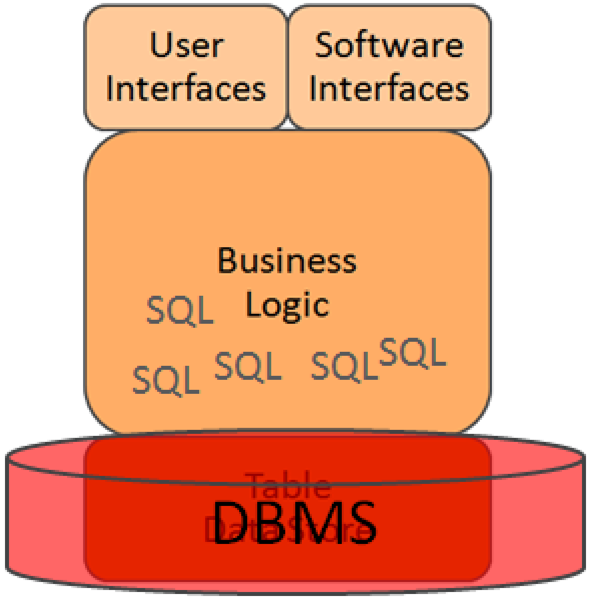
- Code that composes(*) queries and executes them
- Code that composes(*) transactions and executes them
  - *: The way the business requires this to be done

- Queries and transactions (sequence of DML statements) can be
  - Primitive: row-by-row, or
  - Rich: set-based

Roadmap

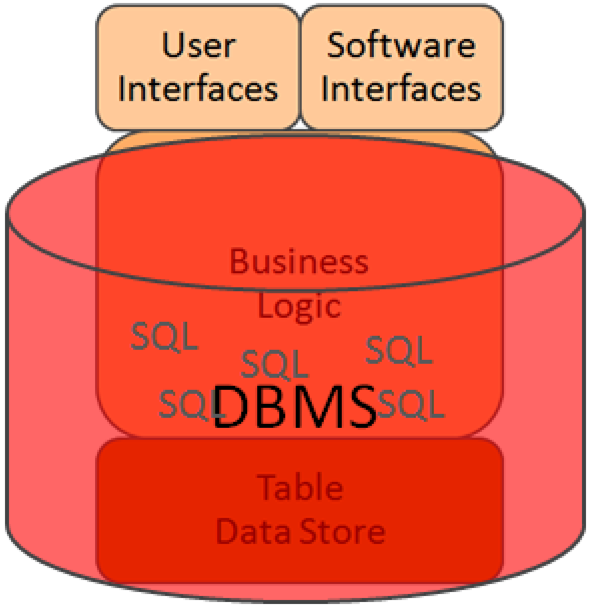
1. Business Logic
2. What Is SmartDB?
3. Some History and Observations
4. Issues With Other Approaches
5. Debunking Performance and Scalability Argument
6. Closing Remarks

We See Two Mutually Distinct Approaches



DBMS = Persistence Layer

"NoPlsql" Approach



DBMS = Processing Engine

"SmartDB" Approach

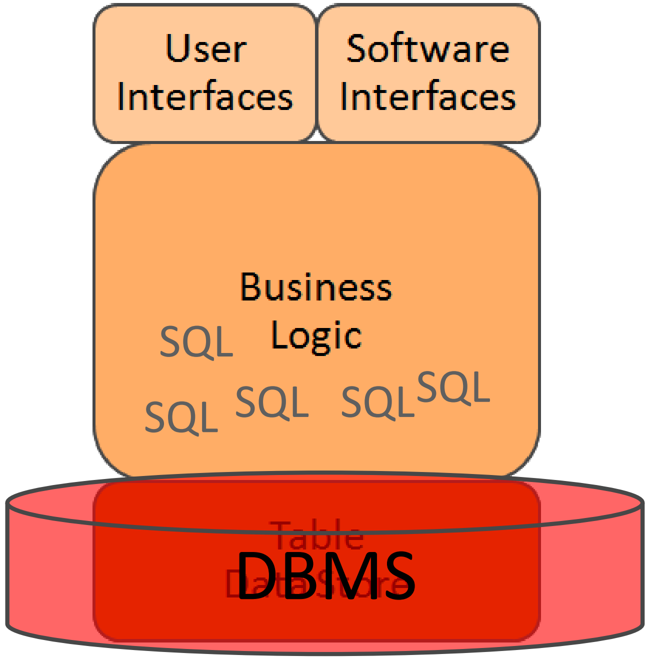
NoPlsql Approach

- Database = persistence layer

- No business logic in database
  - PL/SQL is not used
  - Set-based SQL is not used

- Some other language outside used for business logic
  - Java, .Net, JavaScript, PHP, …
  - Only primitive SQL-statements are submitted

    To persist and retrieve rows



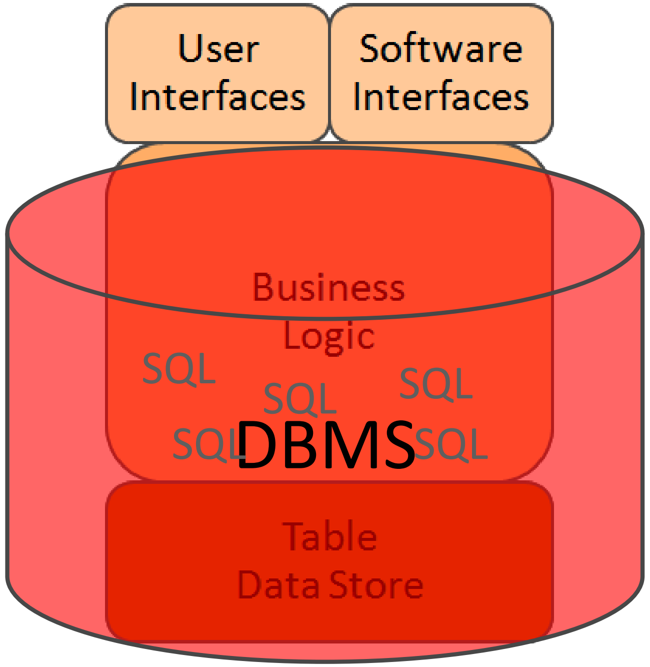
SmartDB Approach

- Database = processing engine

- Business logic is implemented via PL/SQL or complex SQL
  - All SQL, often set-based, executed from PL/SQL
  - Using database in ways it was designed to be used, ergo “SmartDB"

- Database exposes API's for user-interfaces

  More on this in part 2



Roadmap

1 Business Logic

2 What Is SmartDB?

3 Some History and Observations

4 Issues With Other Approaches

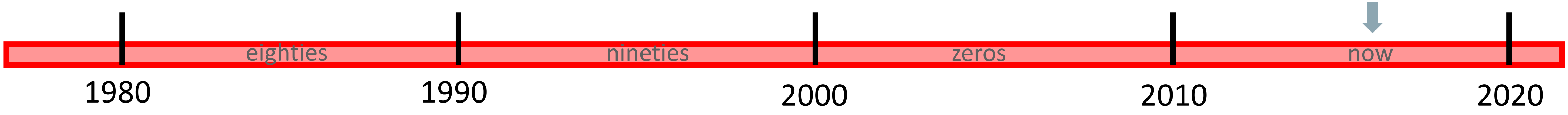
5 Debunking Performance and Scalability Argument

6 Closing Remarks

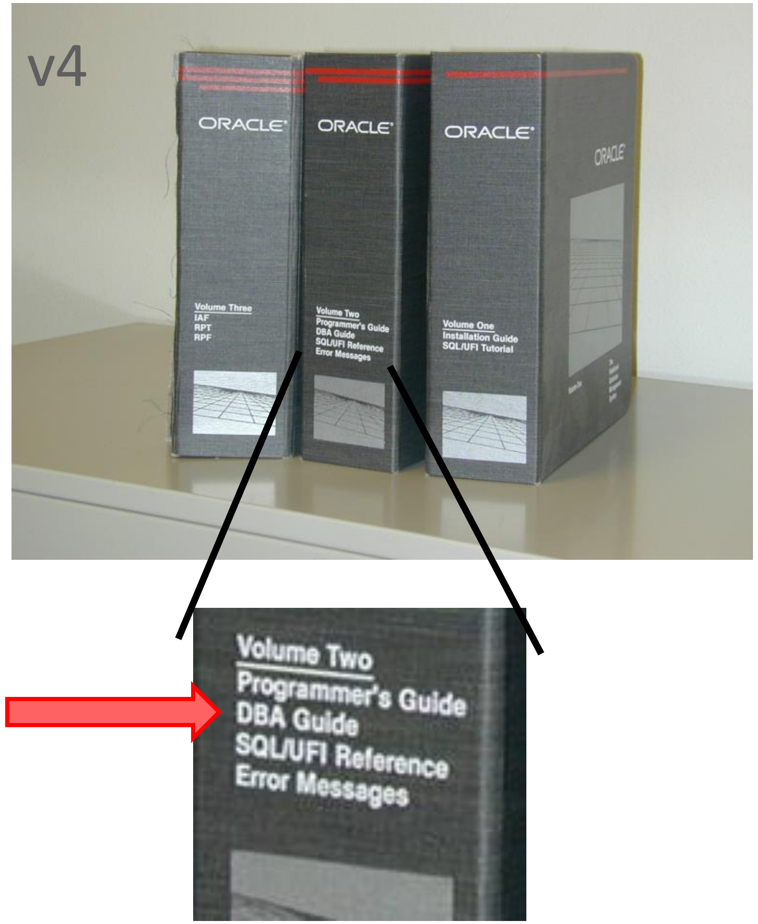
My Ride Through Wonderful World of IT

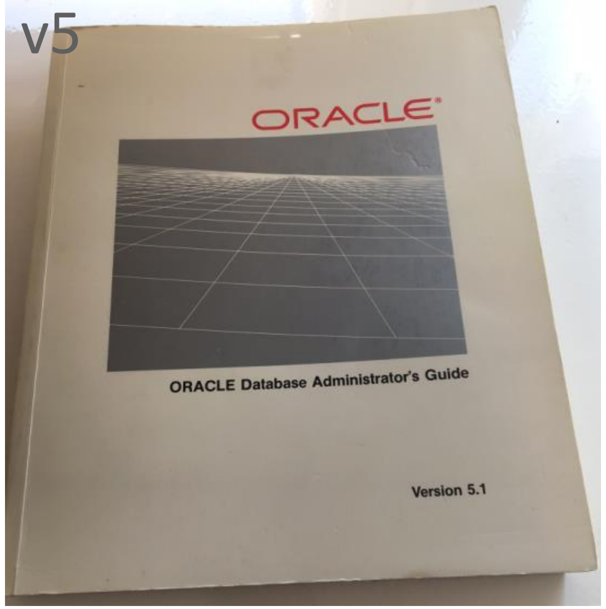
Terminal/host Character-mode GUI's client/server

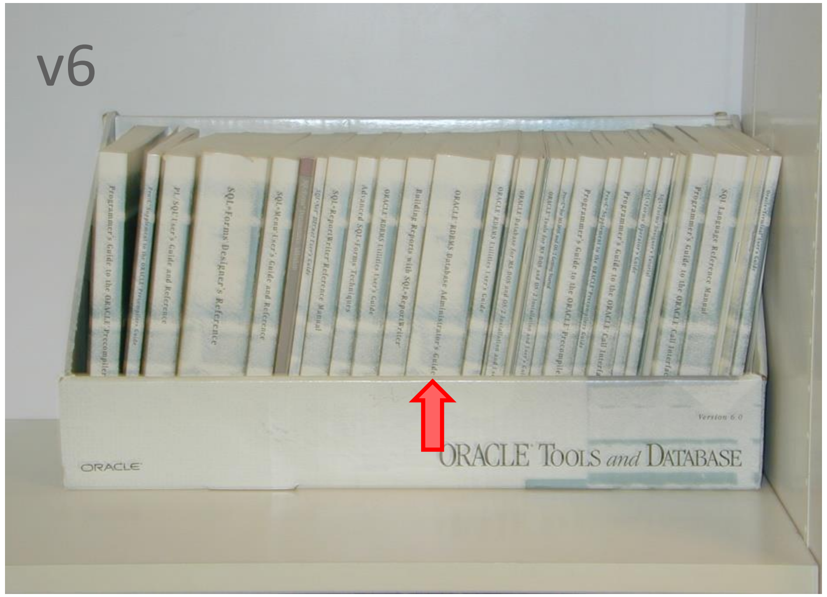
Block-mode/stateless Stateful client programs Stateless browser Many devices/mobile/always connected



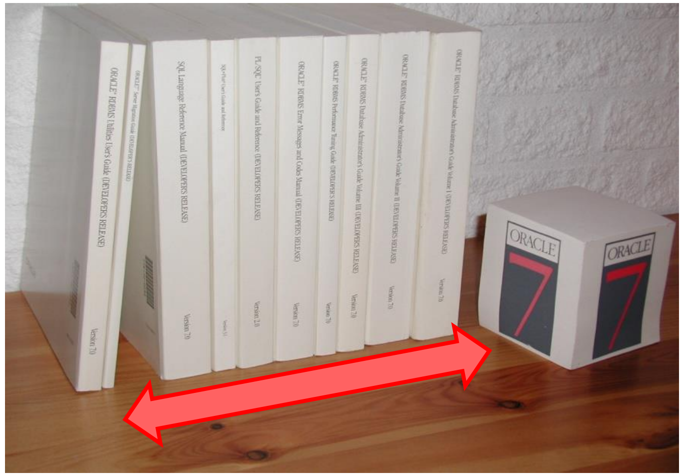
Oracle v4, v5, v6 Database Documentation

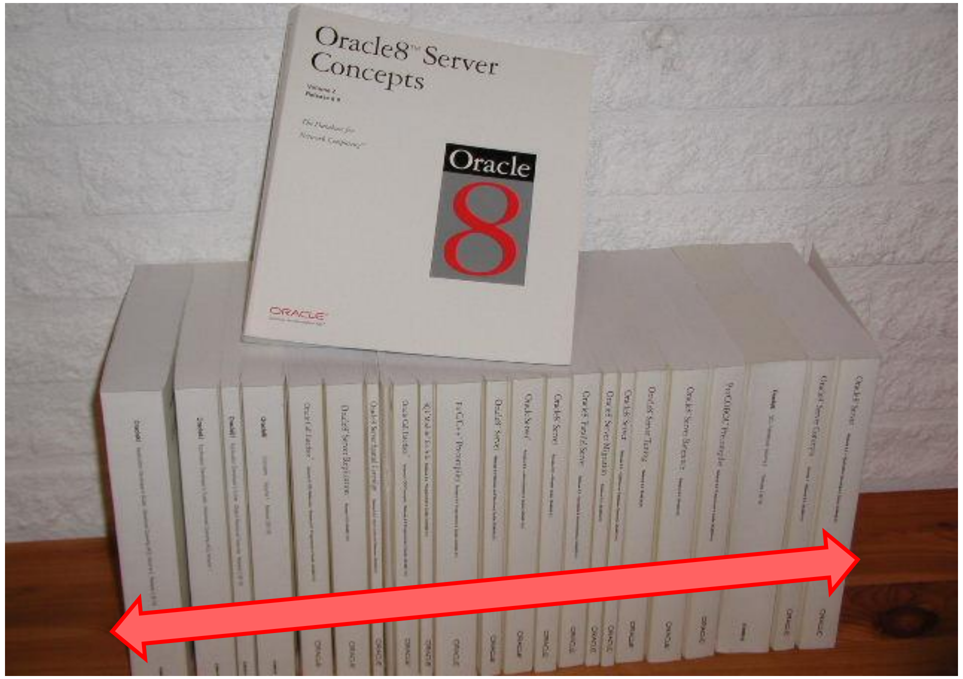




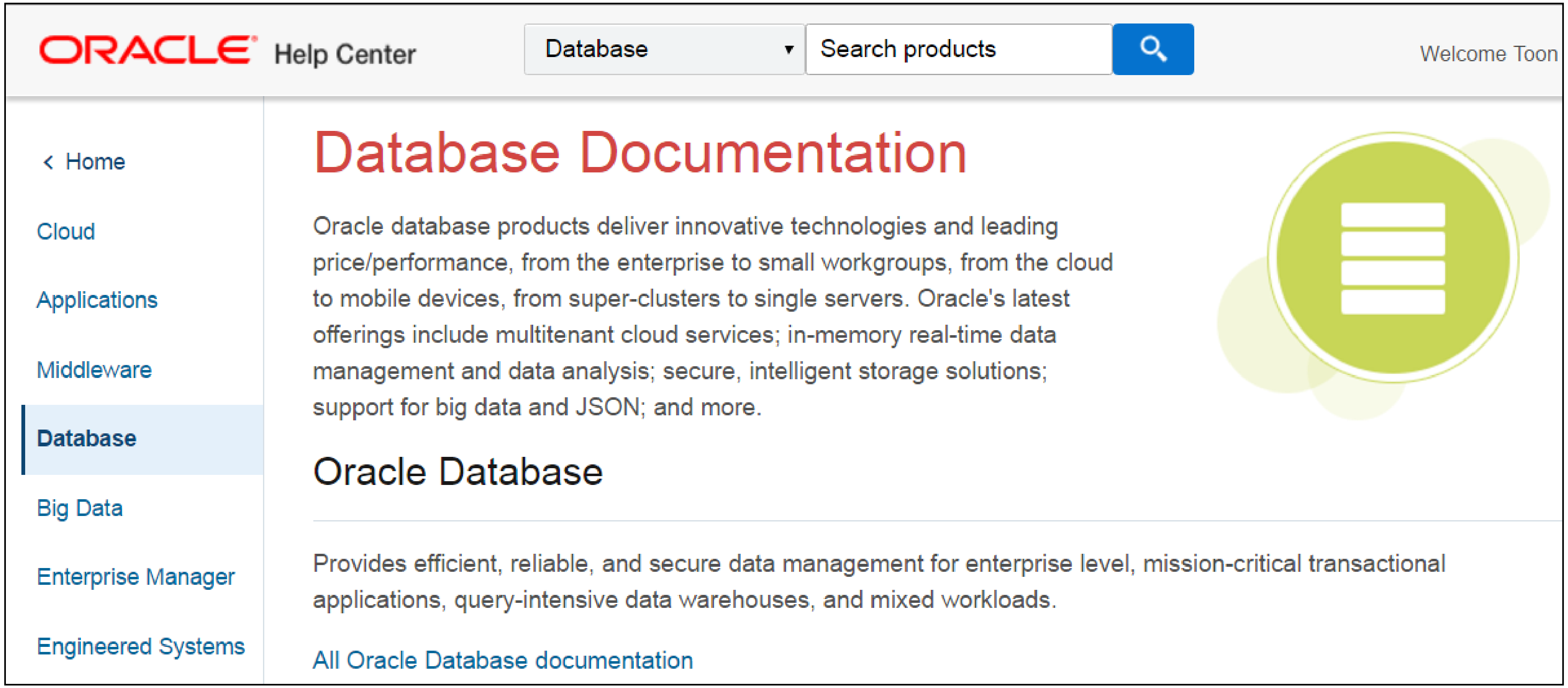


Oracle7, 8i: Database Documentation





http://docs.oracle.com/en/database/



Database Documentation

Oracle database products deliver innovative technologies and leading price/performance, from the enterprise to small workgroups, from the cloud to mobile devices, from super-clusters to single servers. Oracle's latest offerings include multitenant cloud services; in-memory real-time data management and data analysis; secure, intelligent storage solutions; support for big data and JSON; and more.

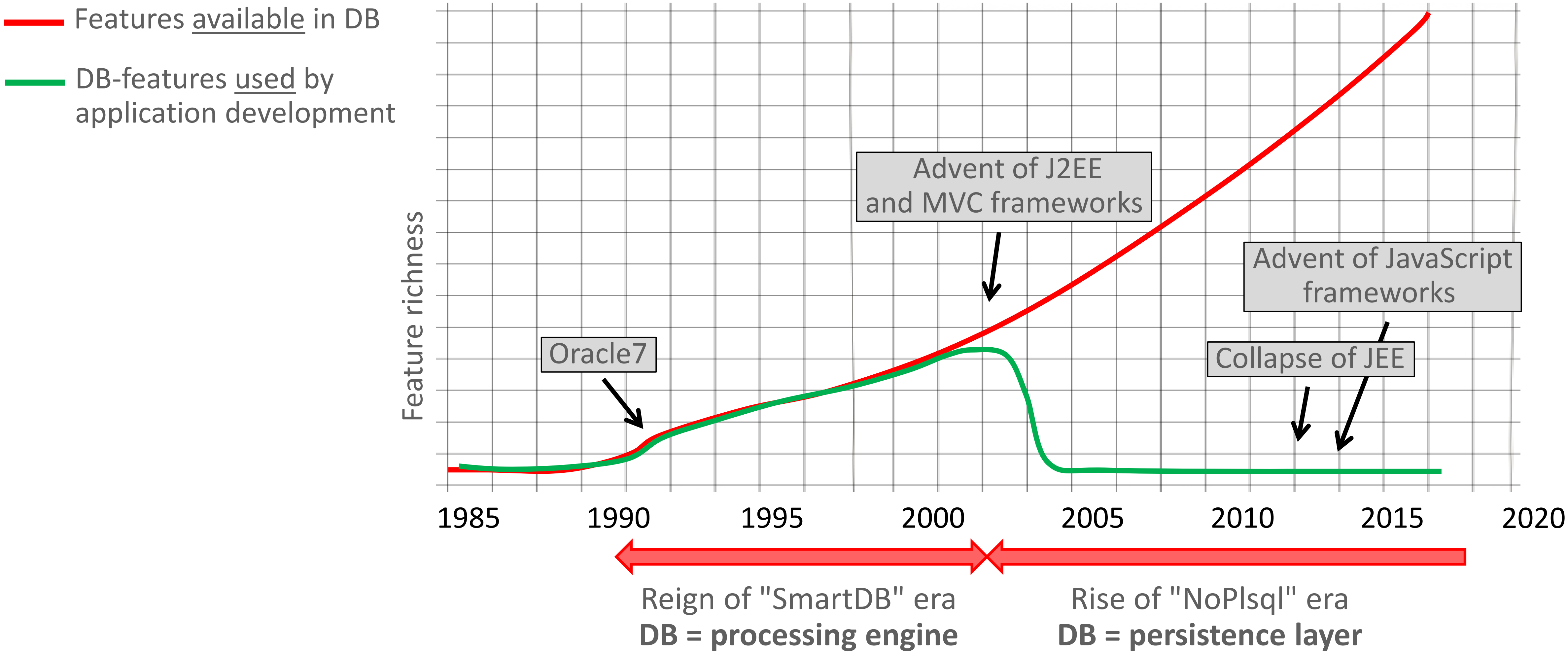
Oracle Database

Provides efficient, reliable, and secure data management for enterprise level, mission-critical transactional applications, query-intensive data warehouses, and mixed workloads.

All Oracle Database documentation

And of course lots of blogs out there on the internet

History Observation: End of "SmartDB" Era



Why Did Java Become so Popular?

- Many programming languages available on Windows in '90-s
  - From Microsoft, and other vendors
  - Industry was experiencing ugly 16-bit to 32-bit conversion
- Java seemed simple, had WORA, and developers were cheap
  - Object orientated programming (OOP) promised code reusability
  - IDE's with method-auto-completion
  - Programmer friendly naming conventions and no header files
  - C-like syntax, lowering bar for C-programmers
  - Offered garbage collection, relieving task of memory management
  - Introduced mainstream exceptions

Write Once
Run Anywhere

See also: https://www.youtube.com/watch?v=QM1iUe6IofM "OOP is bad"

Why Did Java Become so Popular?

Then Sun released **J2EE design pattern** which
included thin browser, fat mid-tier, DB-as-persistence-layer architecture

Promised scalability by offloading code from DBMS

Everybody (vendors, community, and academia) jumped on that bandwagon

The rest is history…

Why Did Java Become so Popular?

- Important note to make:

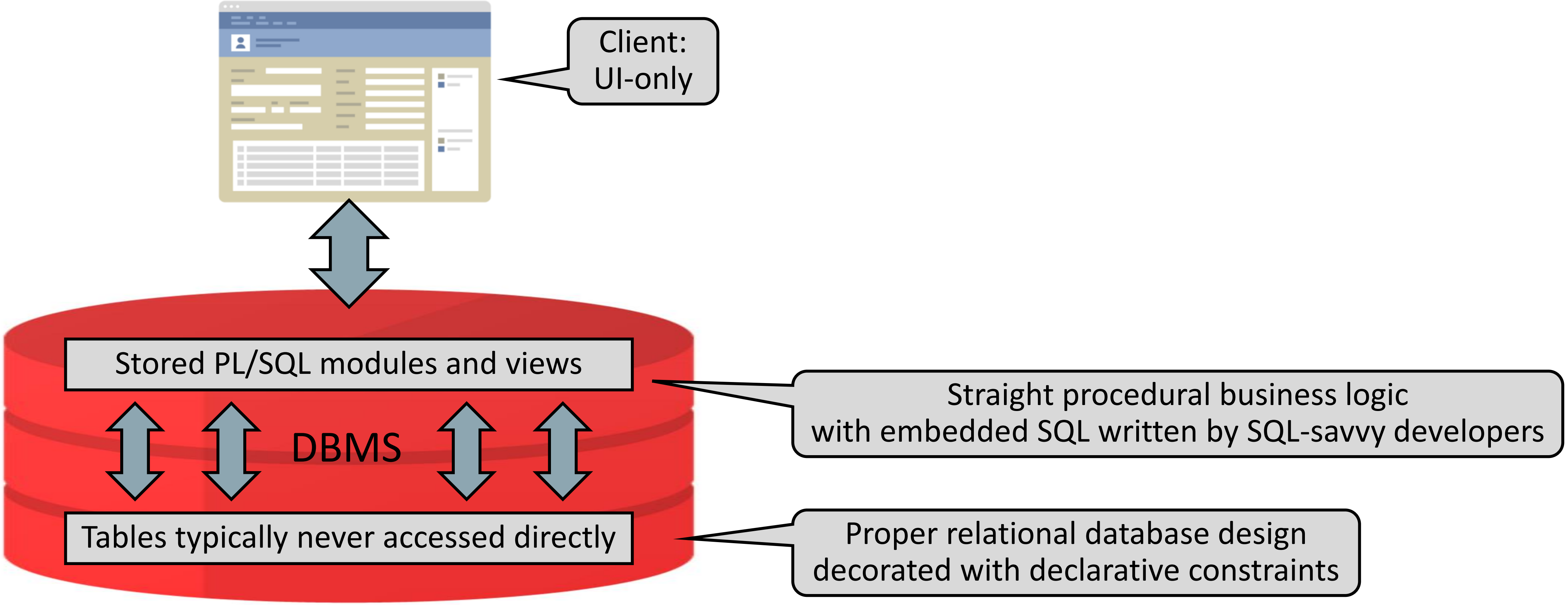
  "Java is a good fit for developing data-intensive business logic" is **not** in that list…

- All of SmartDB goodness was simply discarded in new millennium

- Only real counter-argument was: "Database is always bottleneck"

Where Were We at End of 1990's?

- Applications capitalized on database being a processing engine



Client: UI-only

Stored PL/SQL modules and views

DBMS

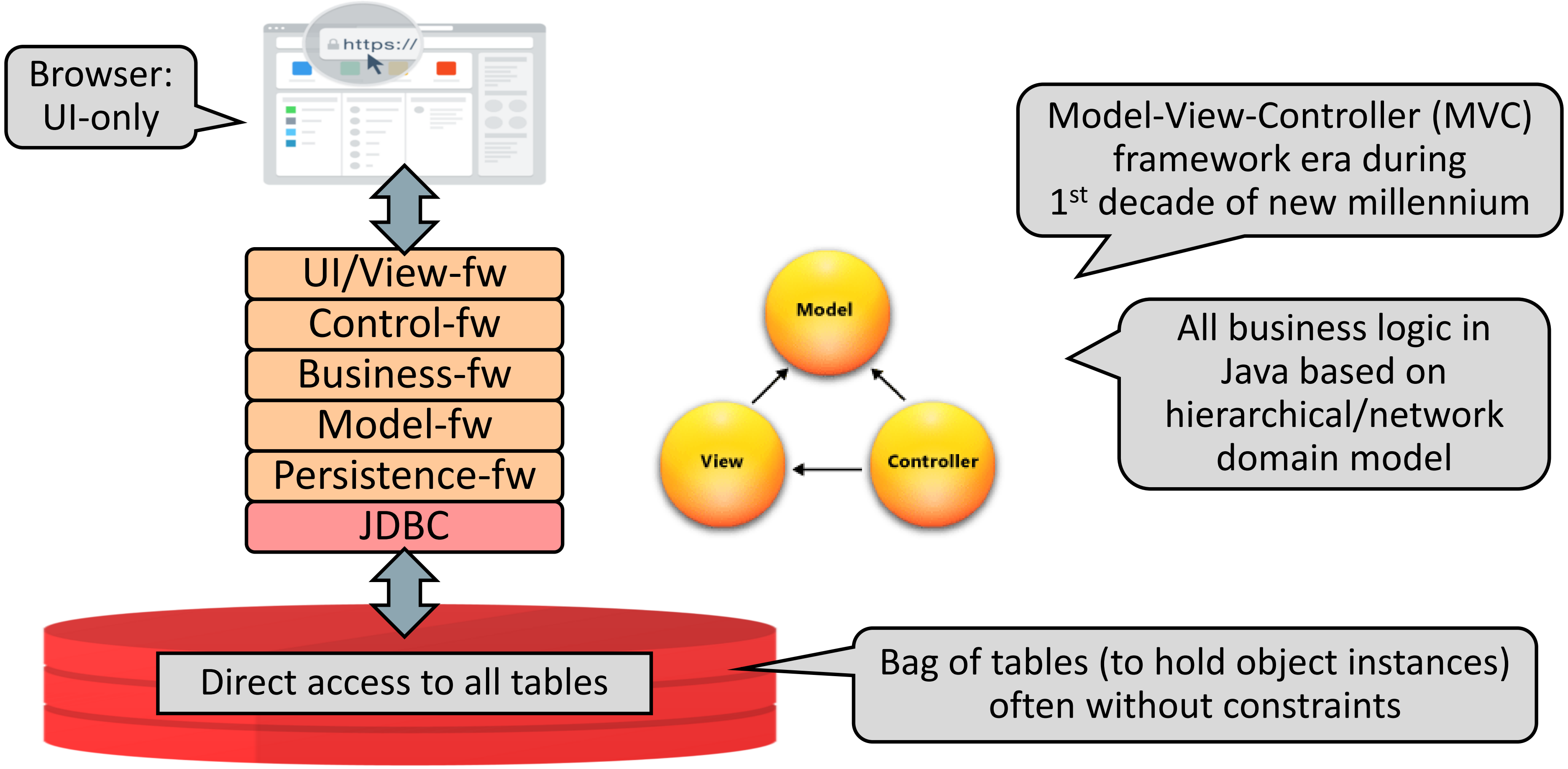
Tables typically never accessed directly

Straight procedural business logic with embedded SQL written by SQL-savvy developers

Proper relational database design decorated with declarative constraints

What Has Happened Since?

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



Browser: UI-only

UI/View-fw

Control-fw

Business-fw

Model-fw

Persistence-fw

JDBC

Direct access to all tables

Model-View-Controller (MVC) framework era during 1st decade of new millennium

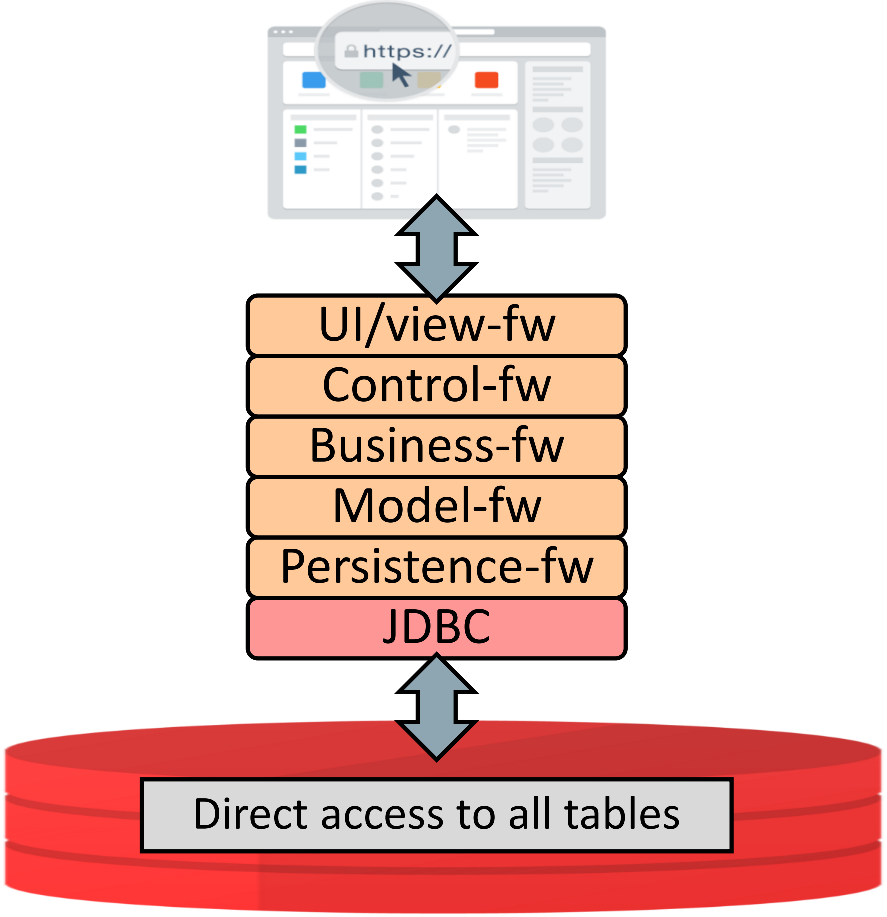
All business logic in Java based on hierarchical/network domain model

Bag of tables (to hold object instances) often without constraints

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 ORM tool

- ORM's produce single-row, single-table SQL
  - Consequence of this type of architecture
  - Which seems to have been accepted by everyone



UI/view-fw

Control-fw

Business-fw

Model-fw

Persistence-fw

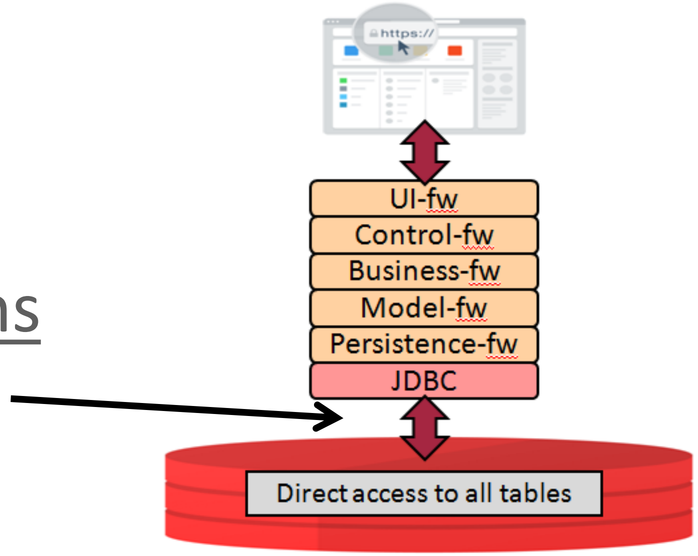
JDBC

Direct access to all tables

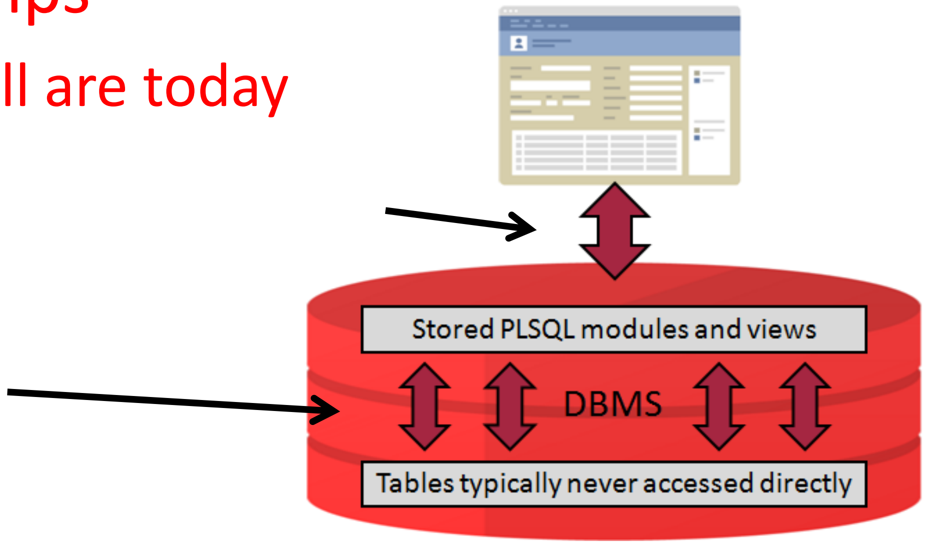
Important Points to Make

- Layered sw-architectures results in "chatty" applications

  Many small calls between JDBC and database



- In early 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


Stored PLSQL modules and views
DBMS
Tables typically never accessed directly

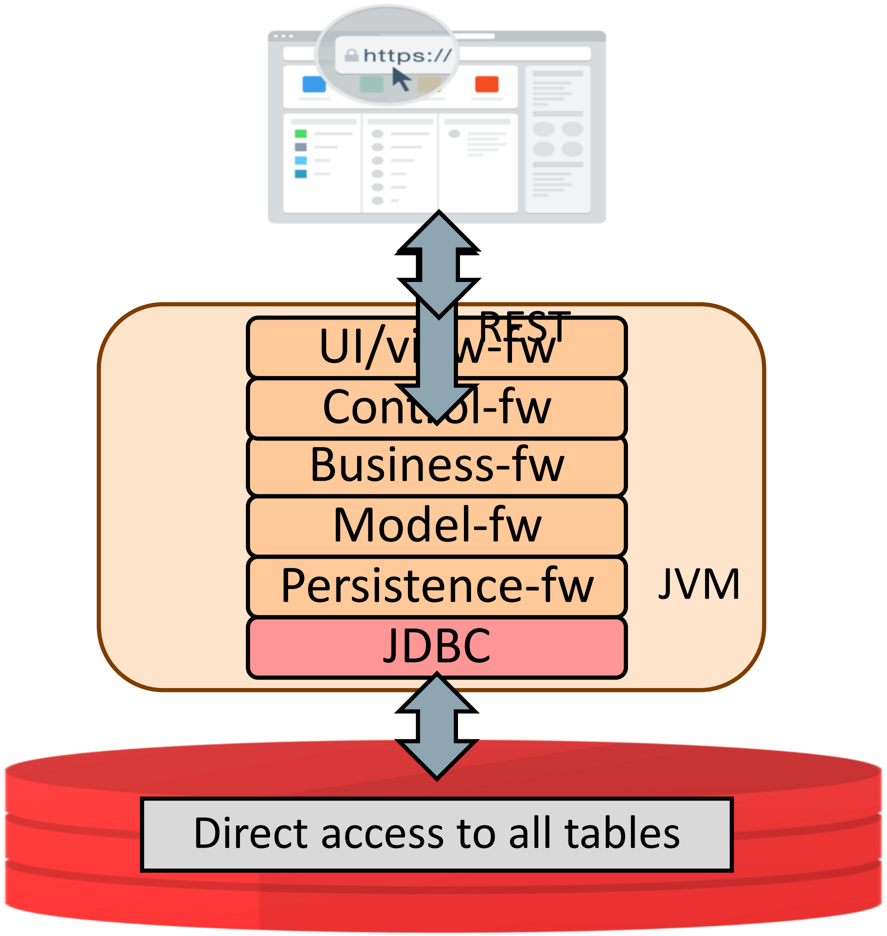
New Paradigm Shift Happening: Java → JavaScript

- 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

- In a sense, this is just client/server all over again
  - Responsive UI running on client (browser)
  - Smart data services running on server (JVM)



Hot Right Now: GraphQL

- Really?
- SQL in disguise
- Instead of n Rest calls from browser to various end-points
- GraphQL does one call to GraphQL “server”
- Server has “knowledge” about domain model
- Server dissects call into n Rest calls to end-points

- ?

Status Quo

- Layered Java MVC frameworks approach has been ubiquitous
  - Feature-rich DBMS acts as a persistence layer
  - All business logic implemented outside DBMS
    Submitting simple SQL only
- New JavaScript frameworks (MVVM), which come and go even more quickly, seem to maintain persistence layer role for DBMS

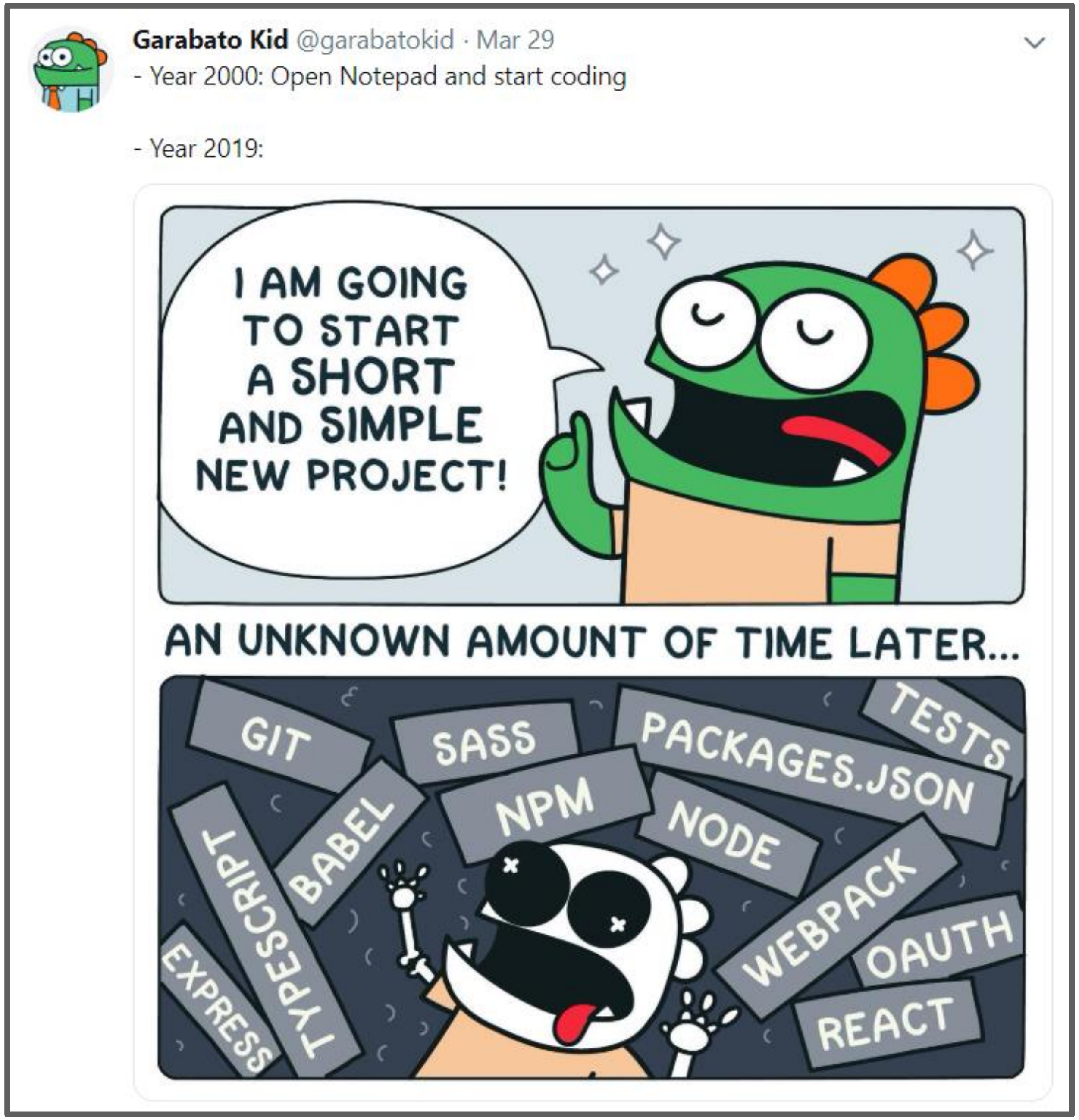
Roadmap

1. Business Logic
2. What Is SmartDB?
3. Some History and Observations
4. Issues With Other Approaches
5. Debunking Performance and Scalability Argument
6. Closing Remarks

Issues with NoPlsql

1. Stability of technology stack
2. Development and maintenance cost
3. Risk of compromised database security and integrity
4. Performance and scalability

Out of Control…



Garabato Kid @garabatokid · Mar 29

- Year 2000: Open Notepad and start coding

- Year 2019:

I AM GOING TO START A SHORT AND SIMPLE NEW PROJECT!
AN UNKNOWN AMOUNT OF TIME LATER...
GIT
SASS
PACKAGES.JSON
TESTS
NPM
NODE
BABEL
TYPESCRIPT
EXPRESS
WEBPACK
OAUTH
REACT

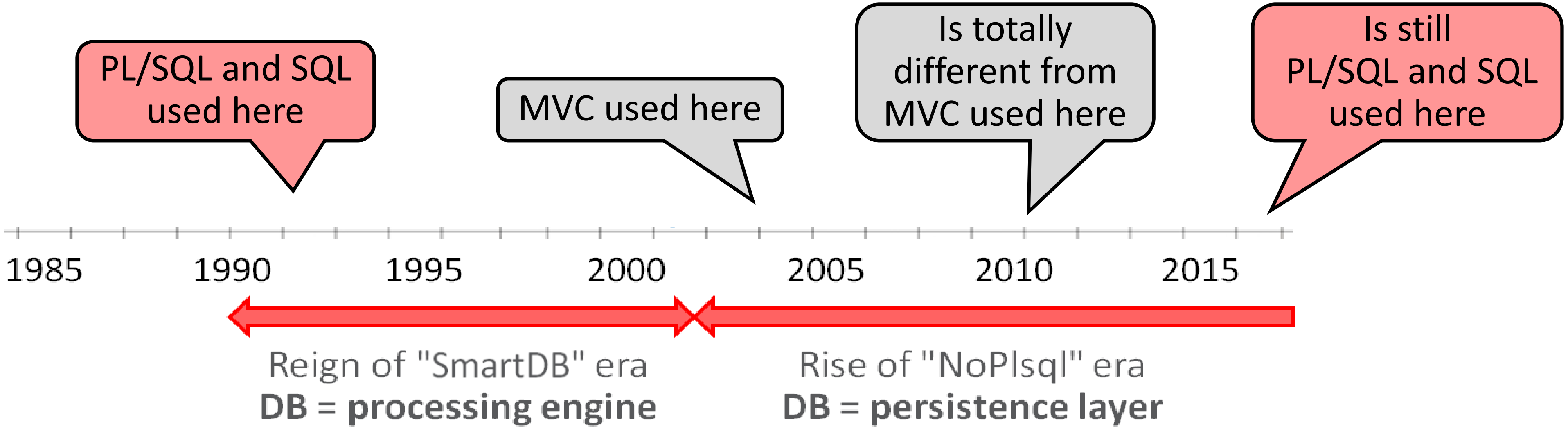
1 Stability of Technology Stack - MVC

- Choice of Java MVC frameworks heavily depended on
  1. Whom you hired or sought advice from
  2. What year + season it was

- Frameworks came and went much faster than did your applications https://en.wikipedia.org/wiki/Comparison_of_web_frameworks

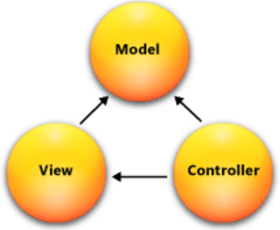
Stability of Technology Stack – **Java MVC**

- Java frameworks came and went much faster than did our applications


PL/SQL and SQL used here
MVC used here
Is totally different from MVC used here
Is still PL/SQL and SQL used here
1985
1990
1995
2000
2005
2010
2015

Reign of "SmartDB" era
DB = processing engine

Rise of "NoPlsql" era
DB = persistence layer



1 2

spring mvc

GRAILS

eclipse link

HIBERNATE

BC4J / ADF-BC

ADF Controller

vaadin }>

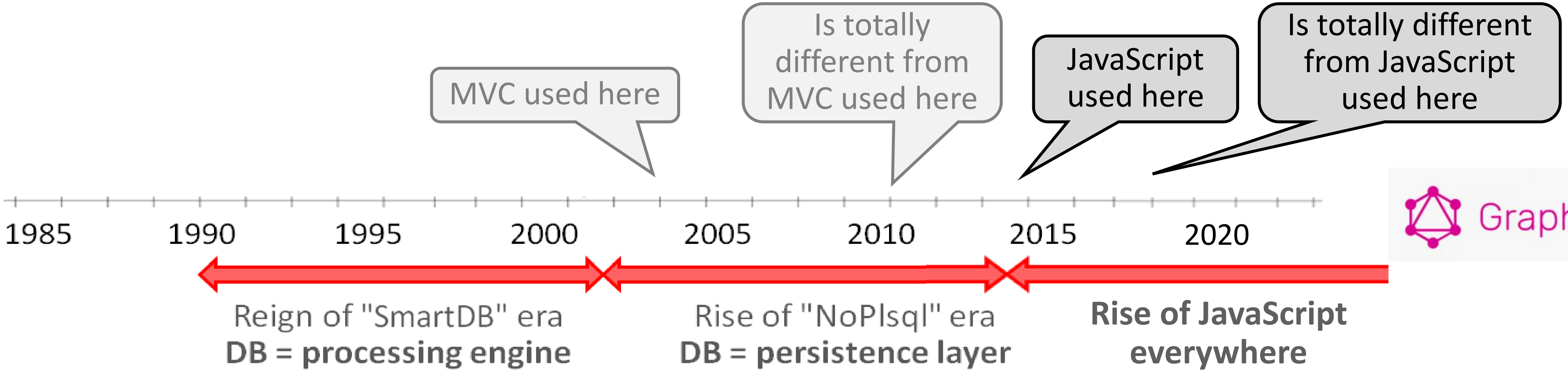
Google Web Toolkit®

tapestry

APACHE WICKET

https://www.youtube.com/watch?v=xsSnOQynTHs
https://www.youtube.com/watch?v=8aGhZQkoFbQ&t

Stability of Technology Stack – **JavaScript MVC**


MVC used here
Is totally different from MVC used here
JavaScript used here
Is totally different from JavaScript used here
1985
1990
1995
2000
2005
2010
2015
2020

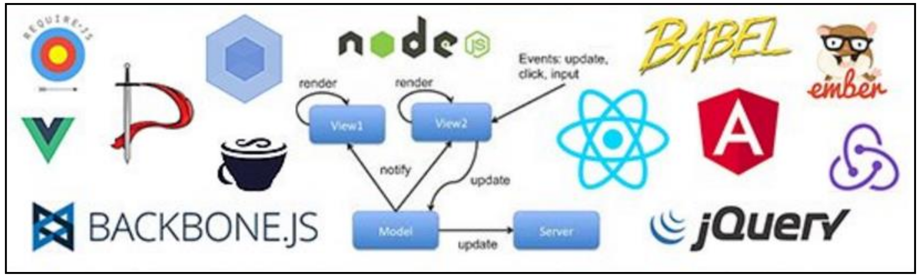
GraphQL

Reign of "SmartDB" era
DB = processing engine

Rise of "NoPlsql" era
DB = persistence layer

Rise of JavaScript everywhere

JavaScript
HTML
CSS



Technology volatility currently worse in emerging new world

ORACLE

Issue 1 Stability of Technology Stack

- Highly dynamic playing field
  - Frameworks outside database come and go fast
  - If business logic gets implemented within these frameworks

    → Danger of mandatory rewrites

    - Frameworks needed upgrading often <u>during</u> ongoing project
    - Frameworks going out-of-fashion

  - Alternatively: stay on old framework with decreasing available knowledge in marketplace

Issue 1: Stability of Technology Stack

- If layers in your chosen technology stacks are volatile…

  Then you ought to use them "thinly"
  - I.e. do not do business logic in them
  - Instead, push business logic further down into code-stack where stable layers exist

  Why? Enables agility. Prevents expensive technology stack upgrades/migrations.

- But nobody has been doing that…
  We have been creating maintenance nightmares in past 15 years

- Prediction:
  PL/SQL and SQL will still be here 10 years from now when JavaScript's reign ends

2 Speed of Development and Maintenance

- Issue is multi faceted
  a) Complex layered technology stacks
  b) Double work: domain model and database design
  c) Wheels are reinvented
  d) Is OO a good fit, given our context?

2a. Technology Stacks Are Complex

- The things you have to learn if you don’t want to “do SQL”:

2a. Technology Stacks Are Complex

- From: https://www.toptal.com/java/how-hibernate-ruined-my-career

> "I had to learn *Hibernate* architecture which included: its configuration, logging, naming strategies, tuplizers, entity name resolvers, enhanced identifier generators, identifier generator optimization, union-subclasses, XDoclet markup, bidirectional associations with indexed collections, ternary associations, idbag, mixing implicit polymorphism with other inheritance mappings, replicating object between two different datastores, detached objects and automatic versioning, connection release modes, stateless session interface, taxonomy of collection persistence, cache levels, lazy or eager fetching and many, many more."

2a. Plumbing Code and Architecture Discussions

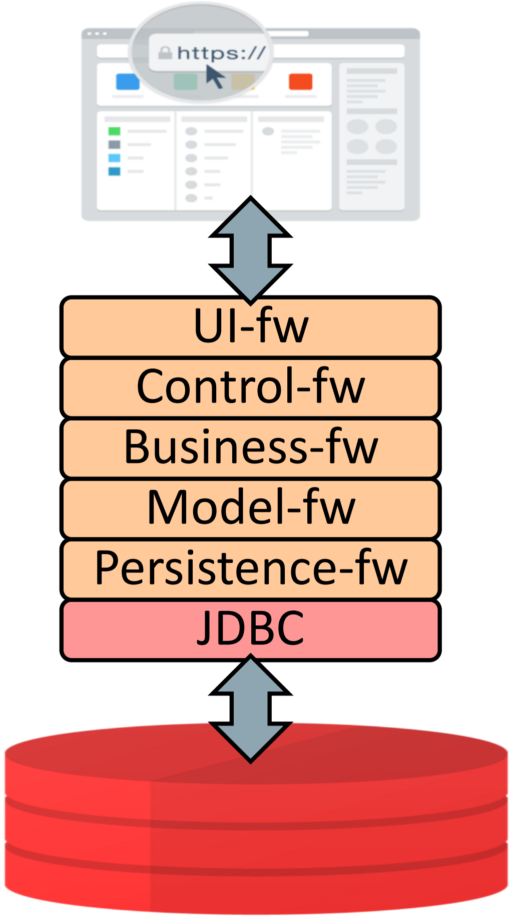
- Frameworks don't just work out of the box

  Need to be configured and glued together

- How exactly to do this results in debates at start of project
  - New role: "the architect"
  - Results in having to develop "plumbing code": glue and infrastructure code
  - Further refining and maintaining this, is ongoing cost

- A lot of time gets spent on above two topics

  Developers concentrate less on what is unique to application



2b. Double Work: Domain + Database Design

- Maybe not so much during initial build
  - As data model is likely just generated from domain model
    - Sub-optimal database designs (what about 3NF?)
    - Horrible SQL, performance issues
- Work needs to be put in, to cross the "object-relational impedance mismatch“
  - Resulting in more discussions and lost time
  - Extra work very much during ongoing maintenance when "something in the model needs changing "
- SmartDB developers proportionally spend more time on what end-users care for, and on what is unique to application

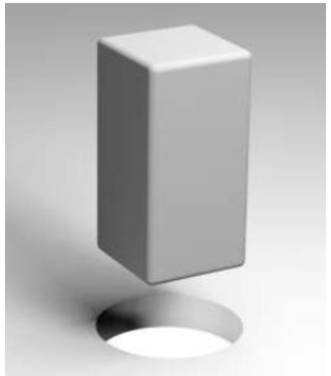
2c. Wheels Are Reinvented

- Both by frameworks as well as by developers
  - Transaction management, cache synchronization, read-consistency, security, …
  - Do-it-yourself: joining, set-operations, grouping, sorting, aggregation, …

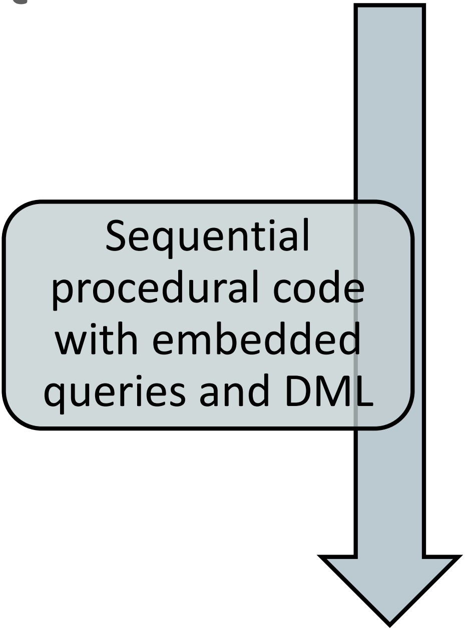
- All available out-of-the-box inside database, declaratively via SQL

ORACLE

2d. Is Object Orientation (OO) a Good Fit?

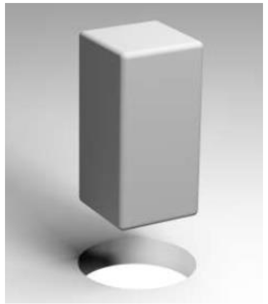


- Example use-case: funds transfer
  Inputs: source-account, target-account, transfer-amount
  - Perform validations on input values
  - Apply various "business rules"
    - Lookup customer-type and apply type specific policies
    - Lookup account-type and apply type specific policies
    - Validate enough funds available for transfer
  - Perform/transact funds transfer
  - Log transaction including policies applied
- In essence nothing OO-ish about business logic



Sequential procedural code with embedded queries and DML

2d. Molding Business Logic into OO-Form



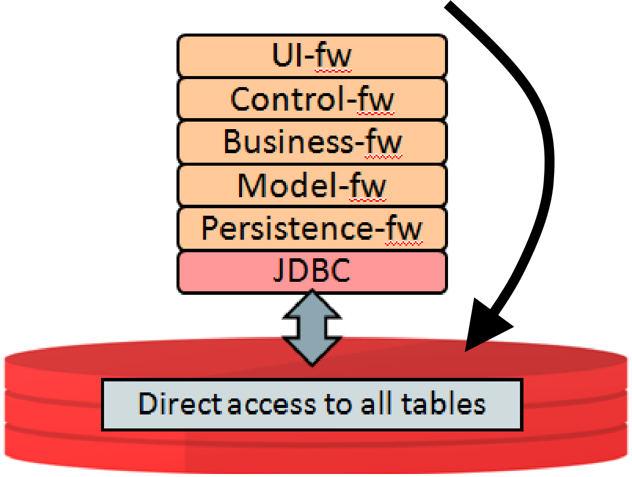
- Business logic = sequence of actions to be performed depending on outcome of embedded data accesses (SQL!) and/or supplied inputs
- Natural fit = Some language that can do SQL really good (think: PL/SQL)
- Hiding these actions into many layers of "abstraction" does not add value
  - Makes reading and understanding code more difficult
  - Makes maintaining code more expensive
  - Makes bug-hunting/providing support more difficult

OO is not silly
It has its use-cases, but doing data-intensive business logic with it, is not one of them

http://www.yegor256.com/2016/08/15/what-is-wrong-object-oriented-programming.html

3 Database Security and Integrity

- NoPlsql approach requires direct access to all tables
- All code to enforce data integrity and secure access is built outside database



- There always is need to access data other than via "the app"
- These accesses can easily compromise data integrity and security policies

4 Performance and Scalability

- "Database is always bottleneck", so here's NoPlsql's promise:
  - Get data from DB once into mid-tier cache
  - Then re-use many times in horizontally scalable mid-tier servers
  - Write data back to db once

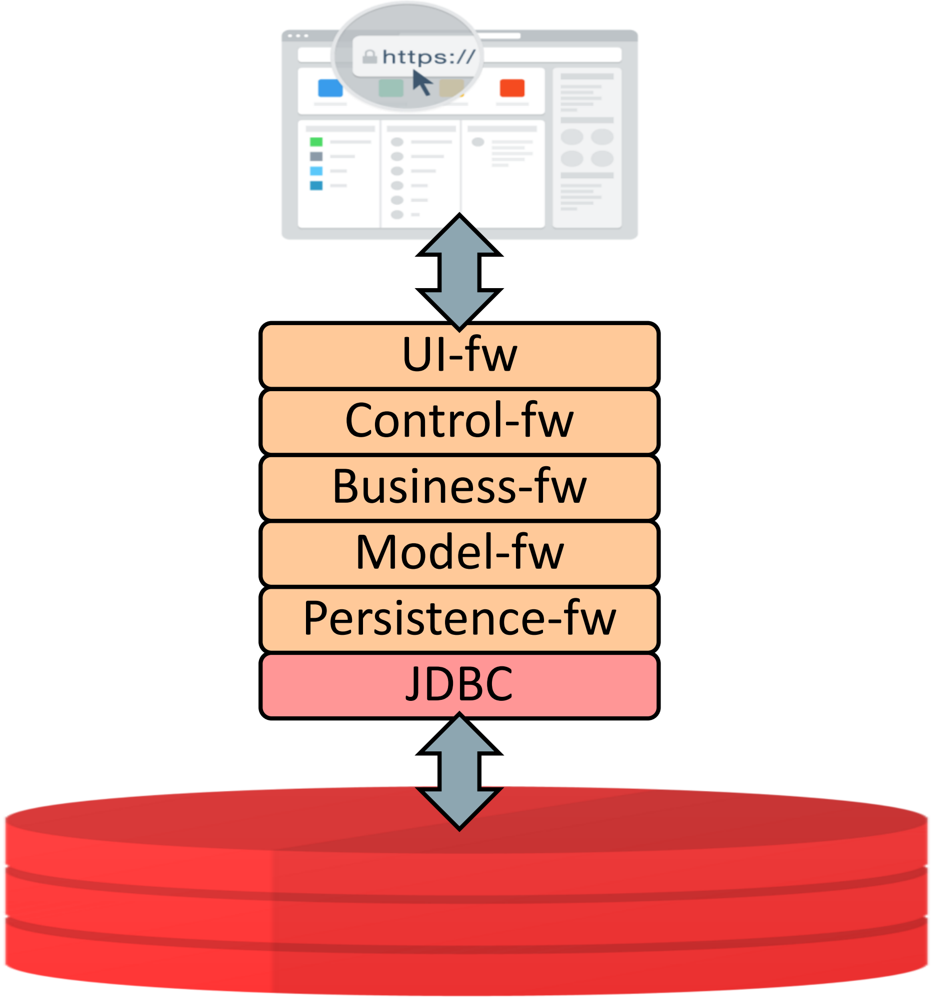
- This is often important argument used to reject SmartDB approach

4 Performance and Scalability

I know there are many realities out there. But this is specific scenario that I'm targeting, as it's the one I see most often

- However in real-world:
- Multiple re-uses of cached data hardly ever takes place
  - It is read + manipulated once, then written back, and not used again while in cache
  - Cached data volumes become so big that caches need to age-out data pre-maturely
- Instantiating objects for rows takes a lot of memory (and CPU)
  Data is always cached in multiple layers (jdbc, orm, …)

4 Scalability with Layered Architectures



UI-fw

Control-fw

Business-fw

Model-fw

Persistence-fw

JDBC

- Looks fantastic on white-board, right?
- Different layers, separation of concerns
  - Can hire expert for each layer
  - Working/tweaking in own layer
- But what happens for problems that require holistic approach like performance?
  - Q: Where is leverage with 6+ layers?
  - A: There is none

4 Scalability with Layered Architectures

- Q: So how do you scale?
- A: You use application parallelism (threading)

- Q: How much code do you need to write or run to make this work?
- A: A lot

Ties back into 'speed of development' issue

Roadmap

Next Section's goal:

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

1. Business Logic
2. What Is SmartDB?
3. Some History and Observations
4. Issues With Other Approaches
5. Debunking Performance and Scalability Argument
6. Closing Remarks

ORACLE®

Summary Here

- Full story at Oracle Learning Library channel on YouTube

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

  Search: "toon koppelaars"

The NoPlsql and Thick Database Approaches

Which One Do You Think Requires a Bigger Database Server?

What We Did

- Built Java batch program and measured performance
  - Using straight Java on top of JDBC (no frameworks)
  - With pattern we always see:
    - Chatty
    - All single-row, single-table SQL queries and DML
    - Get data into mid-tier, use-once, write data back to database
- Rebuilt batch program in PL/SQL also
  - Using same chatty row-by-row SQL behavior
  - Same SQL statements
  - Same business logic

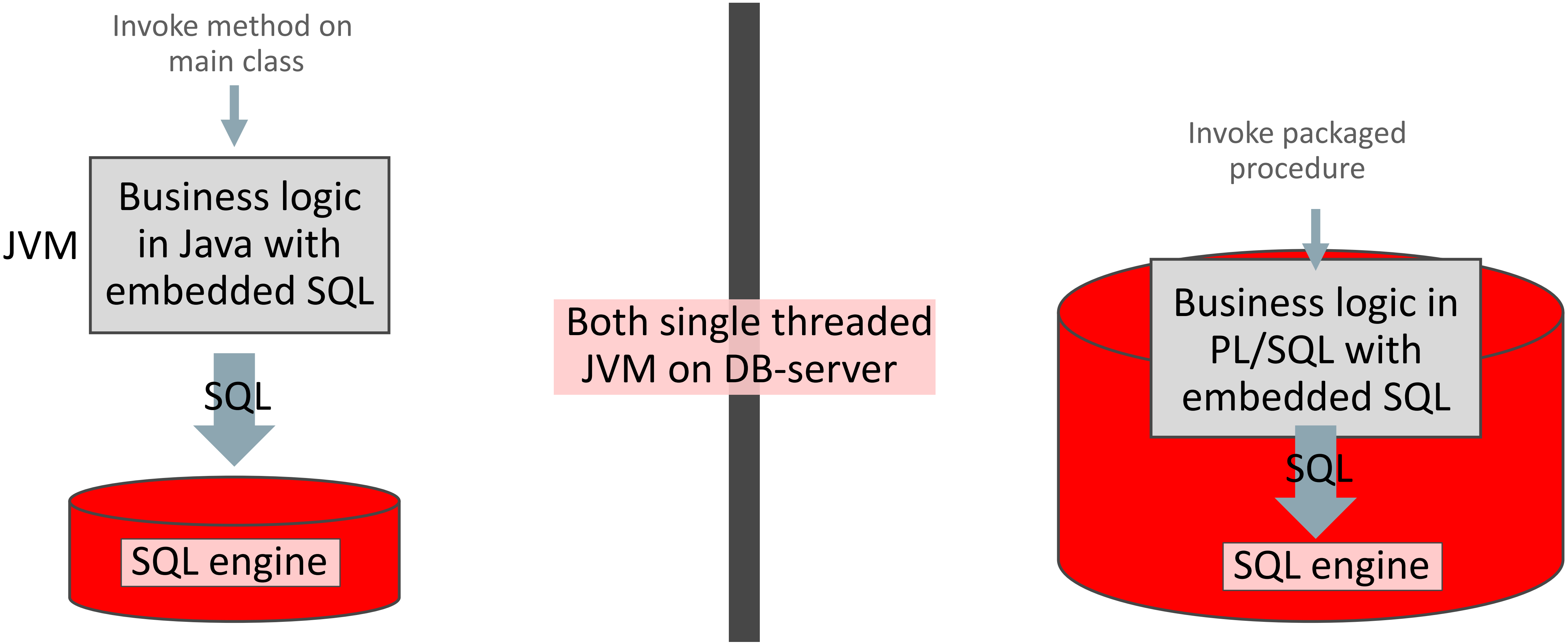
Load Profile

Program does a bit of everything:

- Single row inserts into five tables
- Indexed single row lookups from three tables
- Single row deletes from two tables
- Index maintenance on all involved tables
- Little bit of business logic
  - Row-by-row looping, if-then-else code

Executes 5+ million single-row SQL statements

Plain Java-on-JDBC vs. PLSQL (Both Row-by-Row)



Invoke method on main class

JVM

Business logic in Java with embedded SQL

SQL

SQL engine

Both single threaded JVM on DB-server

Invoke packaged procedure

Business logic in PL/SQL with embedded SQL

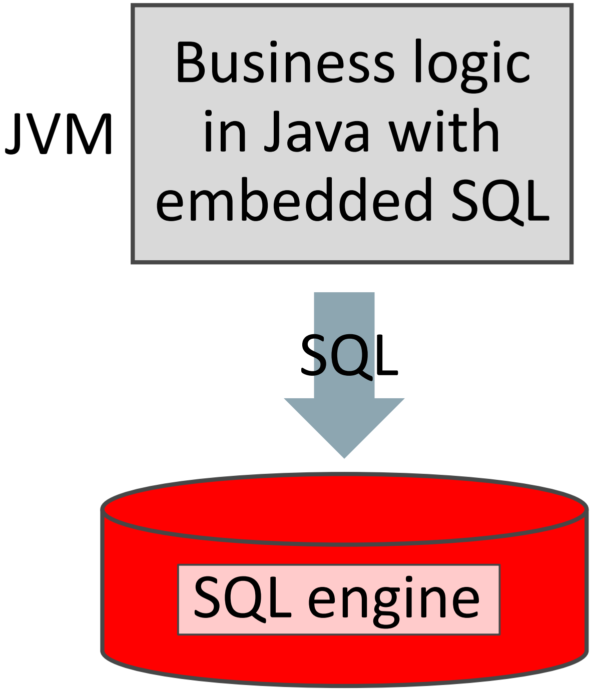
SQL

SQL engine

Plain Java-on-JDBC vs. PLSQL (Both Row-by-Row)

Elapsed-time: **11 minutes**

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


JVM

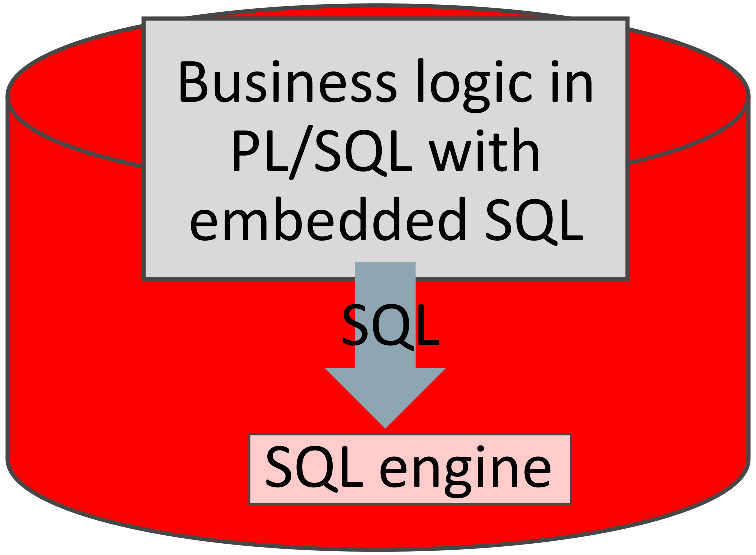
Business logic in Java with embedded SQL

SQL

SQL engine

437 DB-CPU seconds

204 DB-CPU seconds

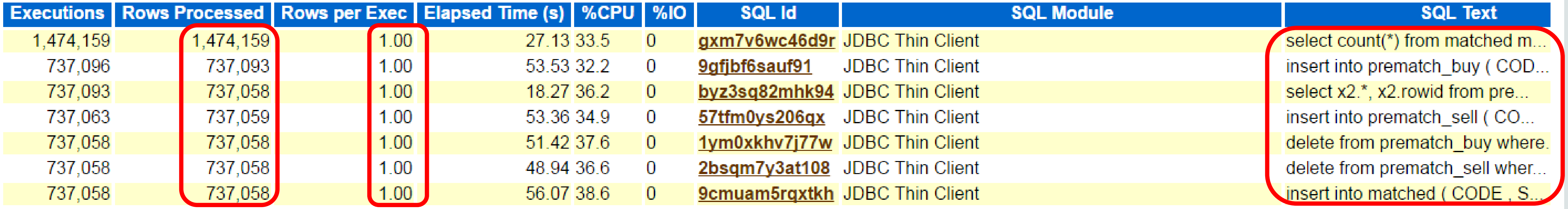

Business logic in PL/SQL with embedded SQL

SQL

SQL engine

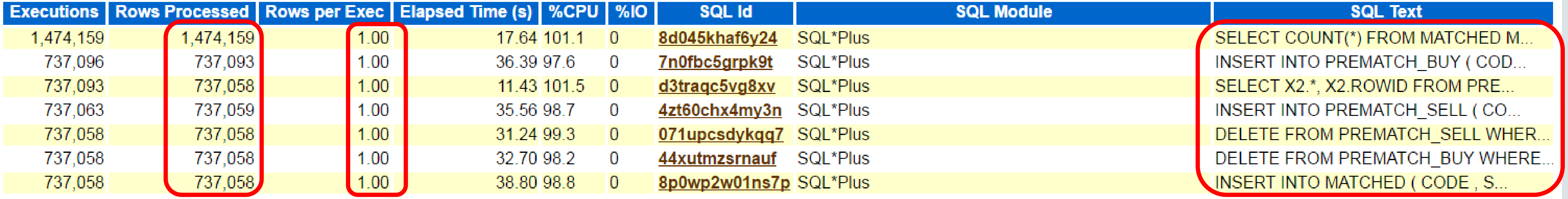
Execute Same SQL, Same Number of Times

Java/JDBC



Executions	Rows Processed	Rows per Exec	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
1,474,159	1,474,159	1.00	27.13	33.5	0	gxm7v6wc46d9r	JDBC Thin Client	select count(*) from matched m...
737,096	737,093	1.00	53.53	32.2	0	9gfjbf6sauf91	JDBC Thin Client	insert into prematch_buy (COD...
737,093	737,058	1.00	18.27	36.2	0	byz3sq82mhk94	JDBC Thin Client	select x2.*, x2.rowid from pre...
737,063	737,059	1.00	53.36	34.9	0	57tfm0ys206qx	JDBC Thin Client	insert into prematch_sell (CO...
737,058	737,058	1.00	51.42	37.6	0	1ym0xkhv7j77w	JDBC Thin Client	delete from prematch_buy where.
737,058	737,058	1.00	48.94	36.6	0	2bsqm7y3at108	JDBC Thin Client	delete from prematch_sell wher...
737,058	737,058	1.00	56.07	38.6	0	9cmuam5rqxtkh	JDBC Thin Client	insert into matched (CODE , S...

PLSQL

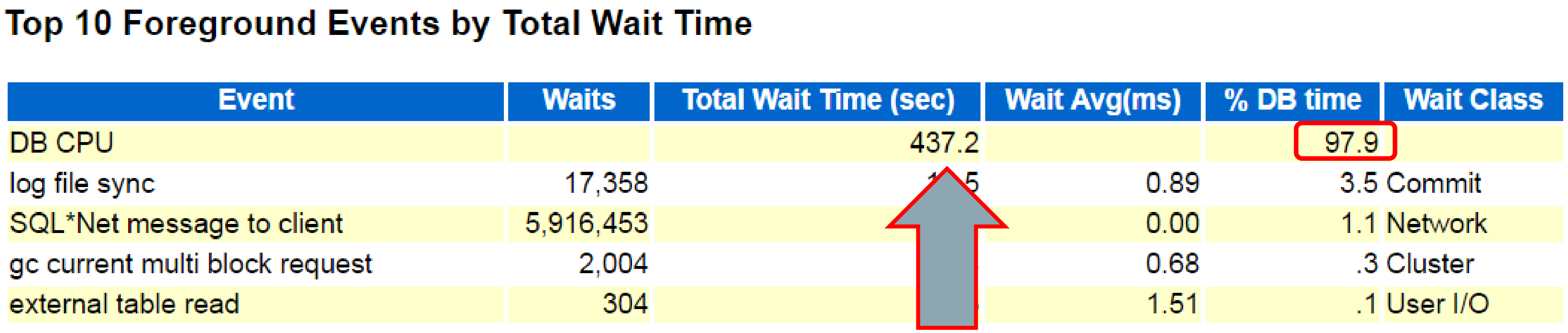


Executions	Rows Processed	Rows per Exec	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
1,474,159	1,474,159	1.00	17.64	101.1	0	8d045khaf6y24	SQL*Plus	SELECT COUNT(*) FROM MATCHED M...
737,096	737,093	1.00	36.39	97.6	0	7n0fbc5grpk9t	SQL*Plus	INSERT INTO PREMATCH_BUY (COD...
737,093	737,058	1.00	11.43	101.5	0	d3traqc5vg8xv	SQL*Plus	SELECT X2.*, X2.ROWID FROM PRE...
737,063	737,059	1.00	35.56	98.7	0	4zt60chx4my3n	SQL*Plus	INSERT INTO PREMATCH_SELL (CO...
737,058	737,058	1.00	31.24	99.3	0	071upcsdykqq7	SQL*Plus	DELETE FROM PREMATCH_SELL WHER...
737,058	737,058	1.00	32.70	98.2	0	44xutmzsrnauf	SQL*Plus	DELETE FROM PREMATCH_BUY WHERE...
737,058	737,058	1.00	38.80	98.8	0	8p0wp2w01ns7p	SQL*Plus	INSERT INTO MATCHED (CODE , S...

REAL-WORLD PERFORMANCE

AWR's Do Not Show Abnormalities: Both CPU Bound

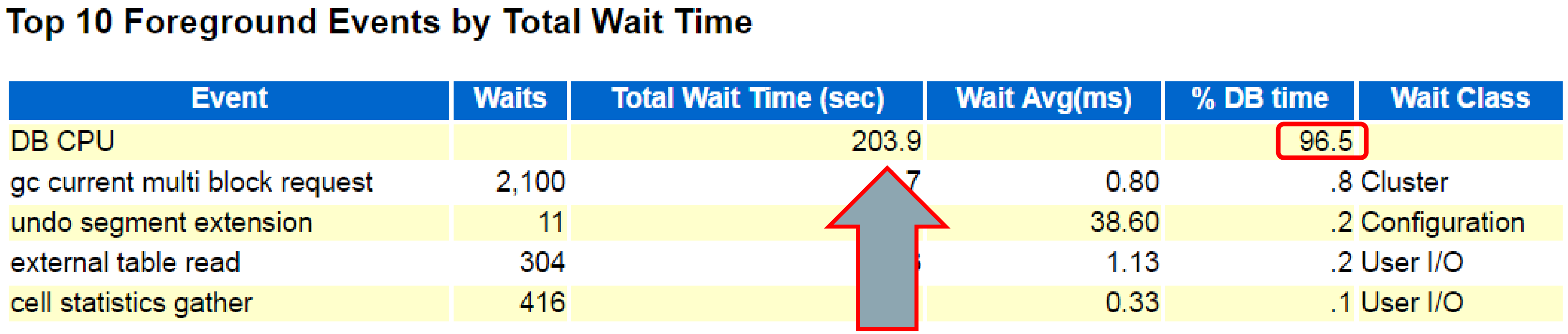
Java/JDBC



Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
DB CPU		437.2		97.9	
log file sync	17,358	[illegible]	0.89	3.5	Commit
SQL*Net message to client	5,916,453	[illegible]	0.00	1.1	Network
gc current multi block request	2,004	[illegible]	0.68	.3	Cluster
external table read	304	[illegible]	1.51	.1	User I/O

PL/SQL



Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
DB CPU		203.9		96.5	
gc current multi block request	2,100	[illegible]	0.80	.8	Cluster
undo segment extension	11	[illegible]	38.60	.2	Configuration
external table read	304	[illegible]	1.13	.2	User I/O
cell statistics gather	416	[illegible]	0.33	.1	User I/O

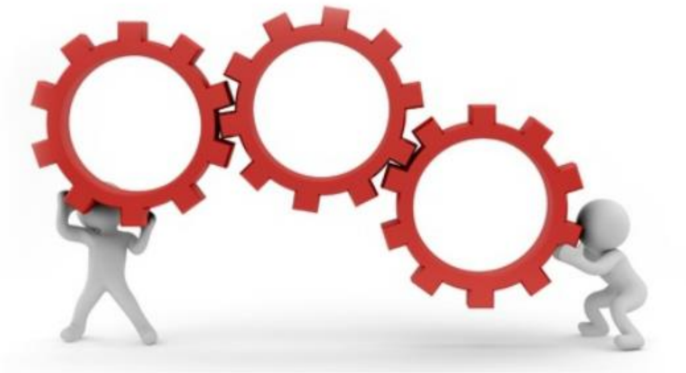
ORACLE REAL-WORLD PERFORMANCE

We Moved From NoPlsql to SmartDB

- Elapsed drops by 3X → #SmartDB is <u>faster</u>
- DB-CPU drops by 2X → #SmartDB is <u>more scalable</u>

Gets work done faster while at same time using less CPU

- Seems like “SmartDB approach will saturate database" is false?



Wow…

- Single-row SQL from NoPlsql consumes 2X DB-CPU?
  - 437 CPU seconds vs. 204 CPU seconds

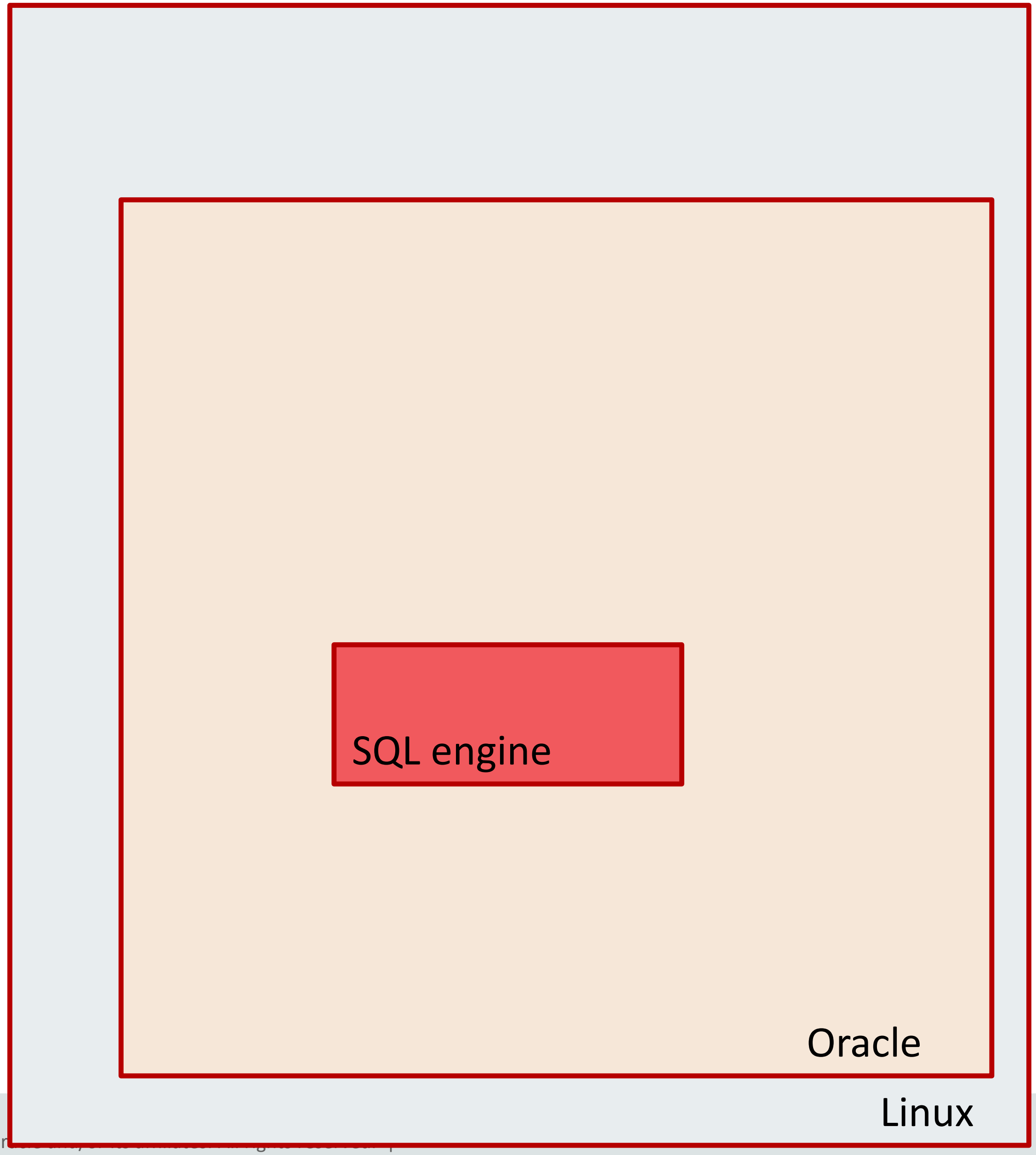
- Why is that?
  1. More code path
  2. Worse "IpC"

Why Is SmartDB So Much More Efficient in Executing SQL?

- "The Living Room" analogy
  - Living room is where SQL engine resides
  - PL/SQL is already in living room
- All other technologies have to enter through front-door
  - Traverse hallway
  - And only then enter living room

The Living Room

- SQL engine



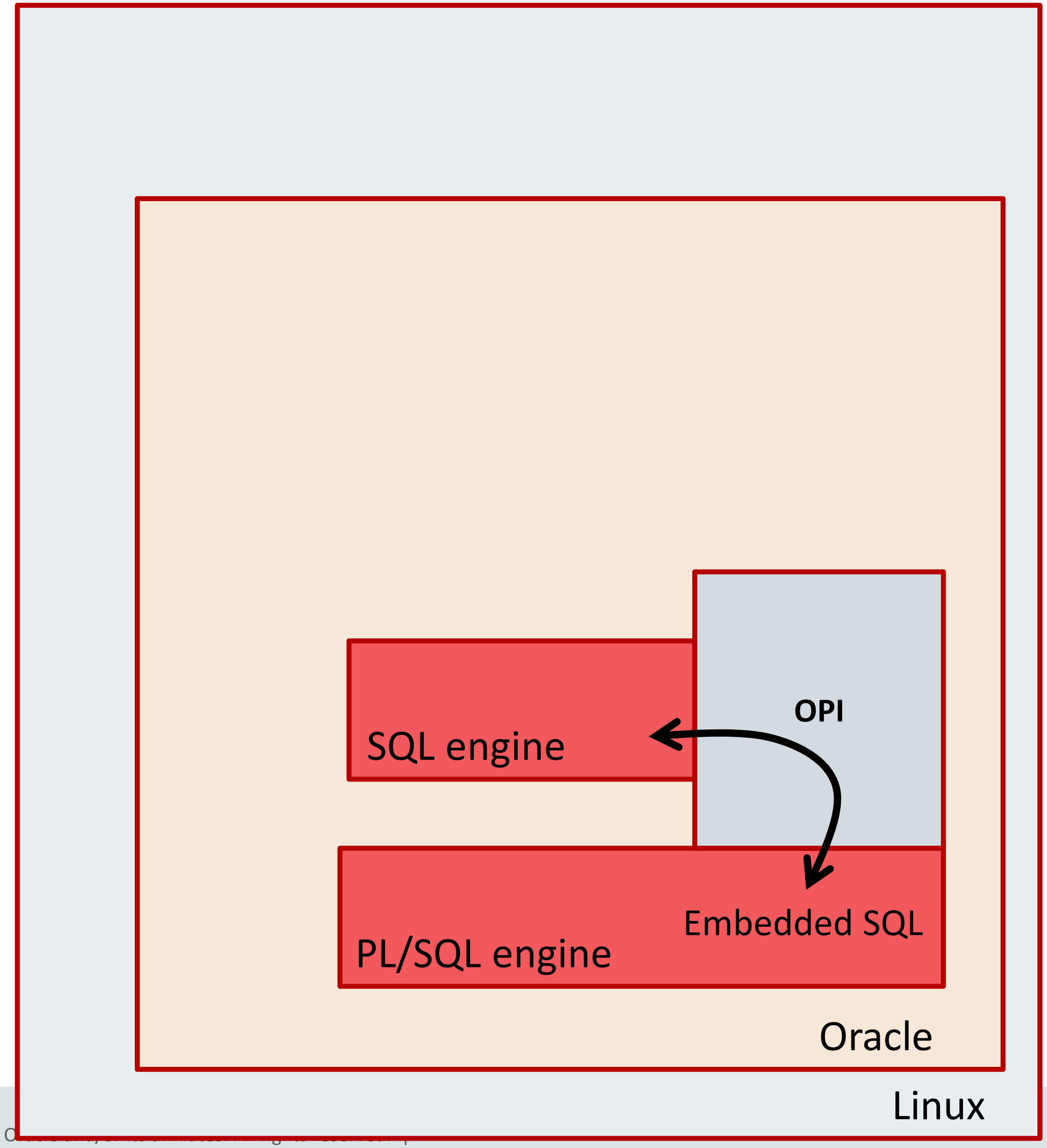
SQL engine

Oracle

Linux

The Living Room

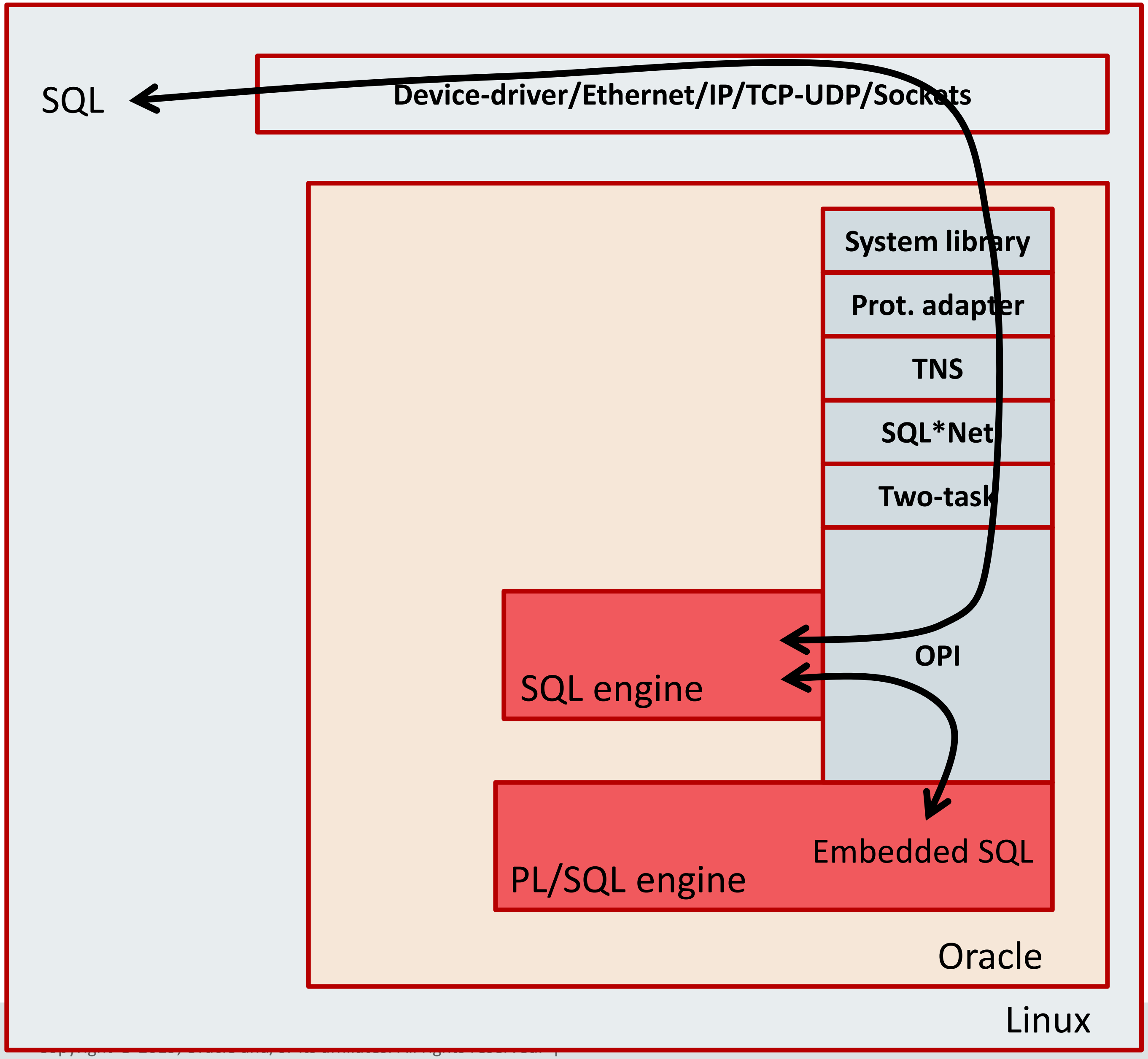
- SQL engine
  - Accessible via OPI layer
  - Oracle Program Interface
- PL/SQL directly calls OPI



The Living Room

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

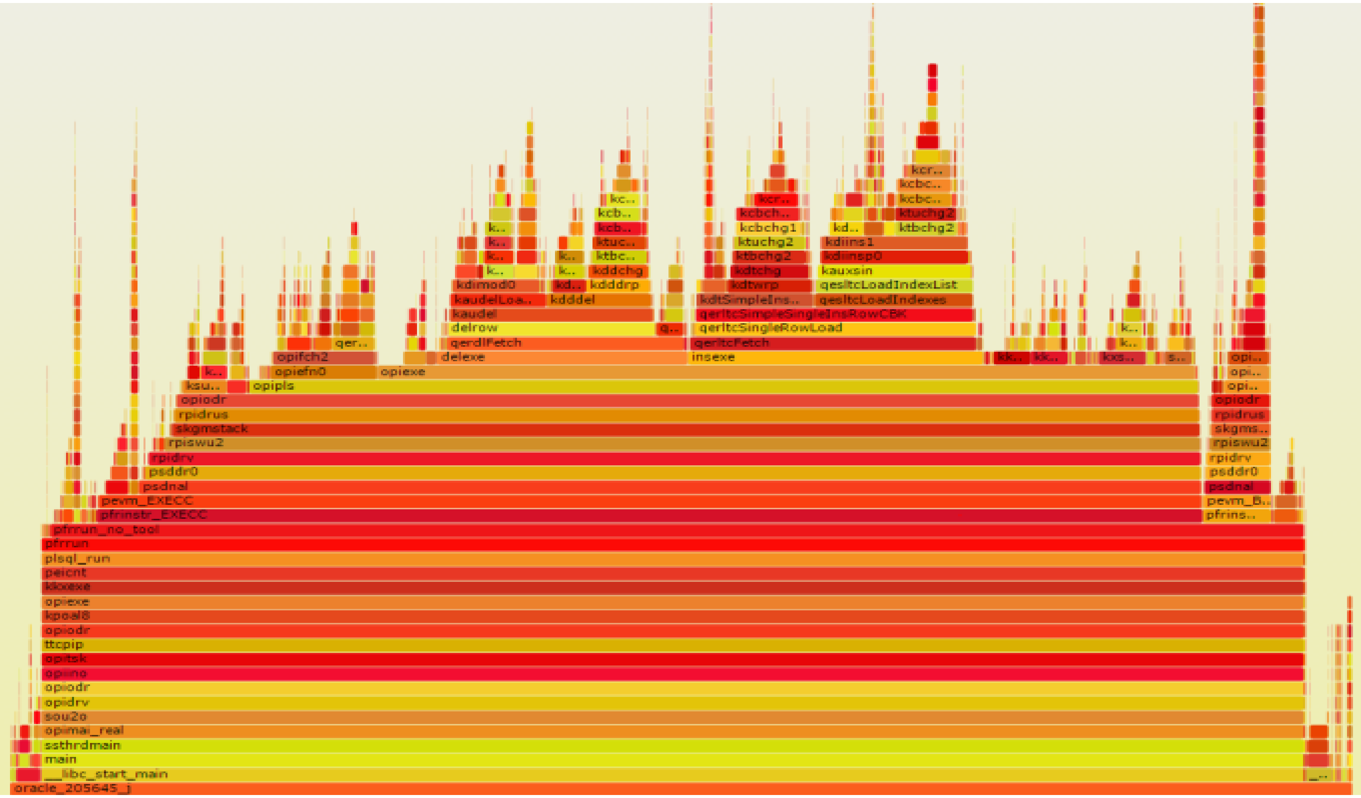
➔ More code path:
which you start noticing
for single-row/single-table SQL



ORACLE

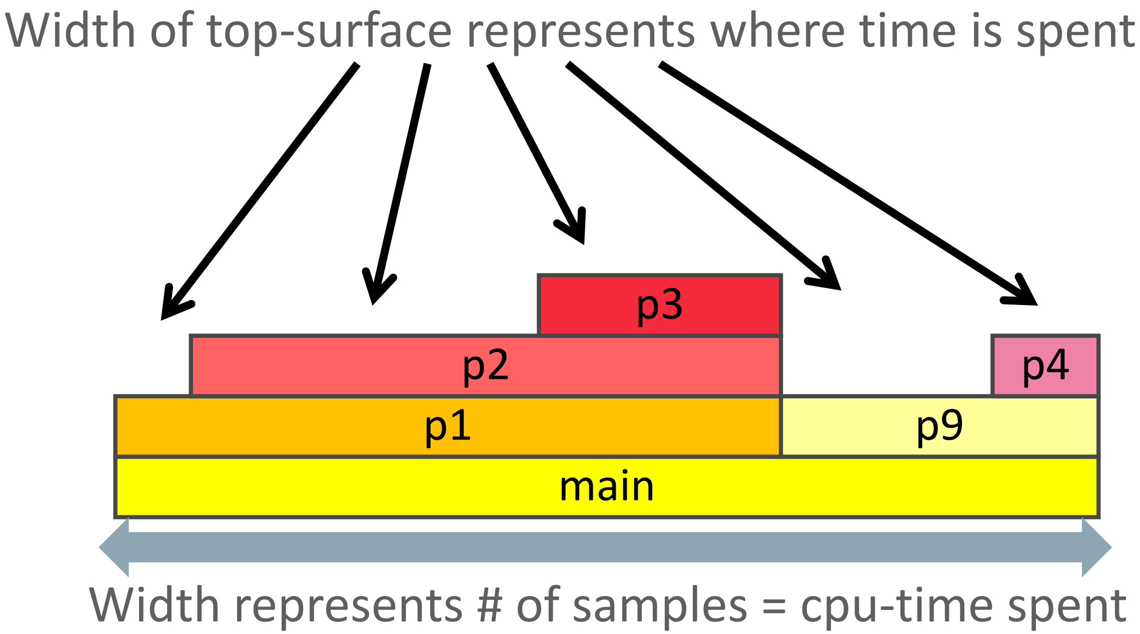
ORACLE
REAL-WORLD PERFORMANCE

Investigated This Via FlameGraphs



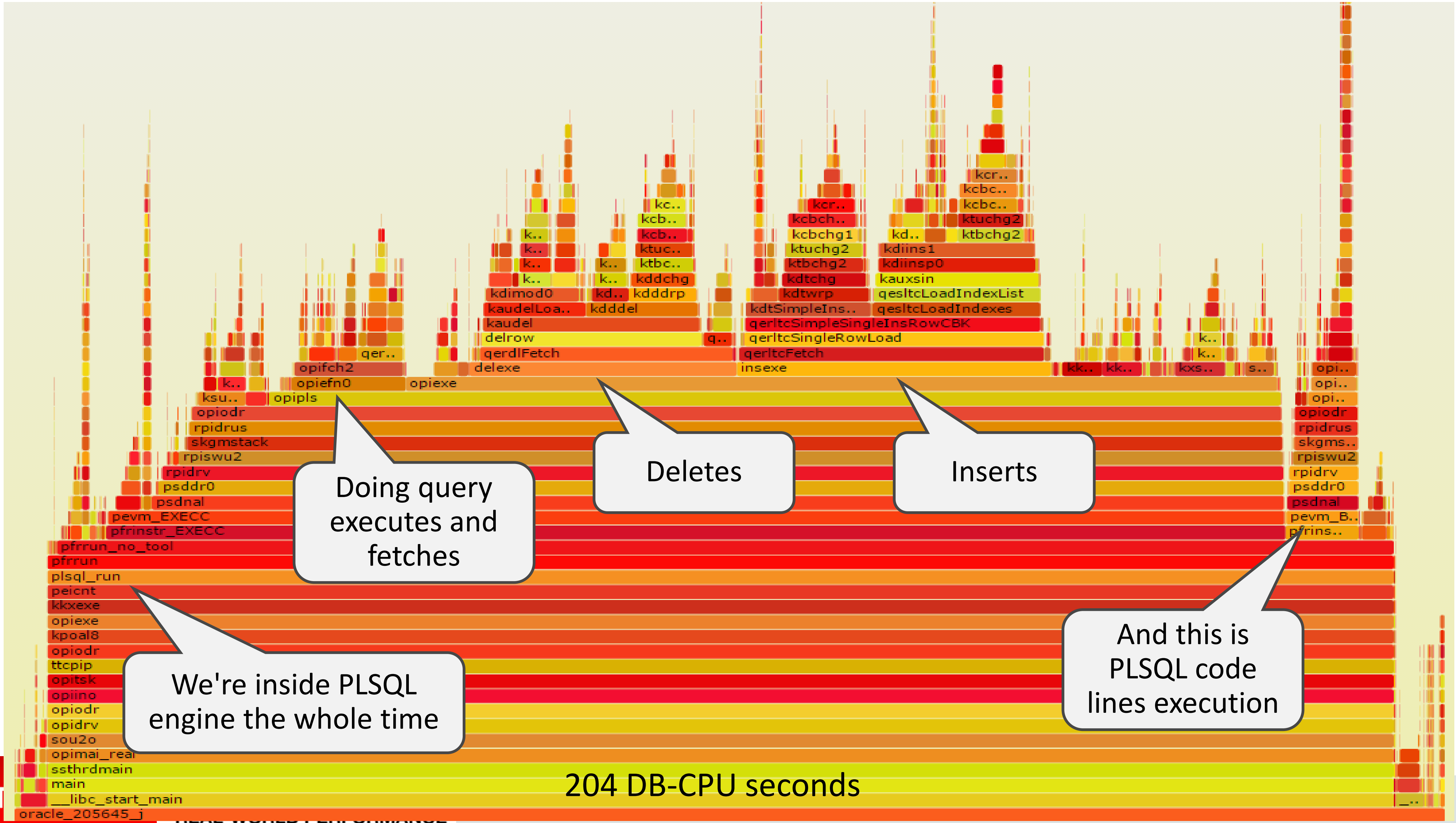
FlameGraph visualizes code-stacks where process has spent its time

Visualizing This Via FlameGraphs


Width of top-surface represents where time is spent
p3
p2
p4
p1
p9
main
Width represents # of samples = cpu-time spent

- More info:
  - https://github.com/brendangregg/FlameGraph
- See also Luca Canali's blog
  - http://externaltable.blogspot.nl/2014/05/flame-graphs-for-oracle.html

Oracle Server FlameGraph SmartDB



Doing query executes and fetches

Deletes

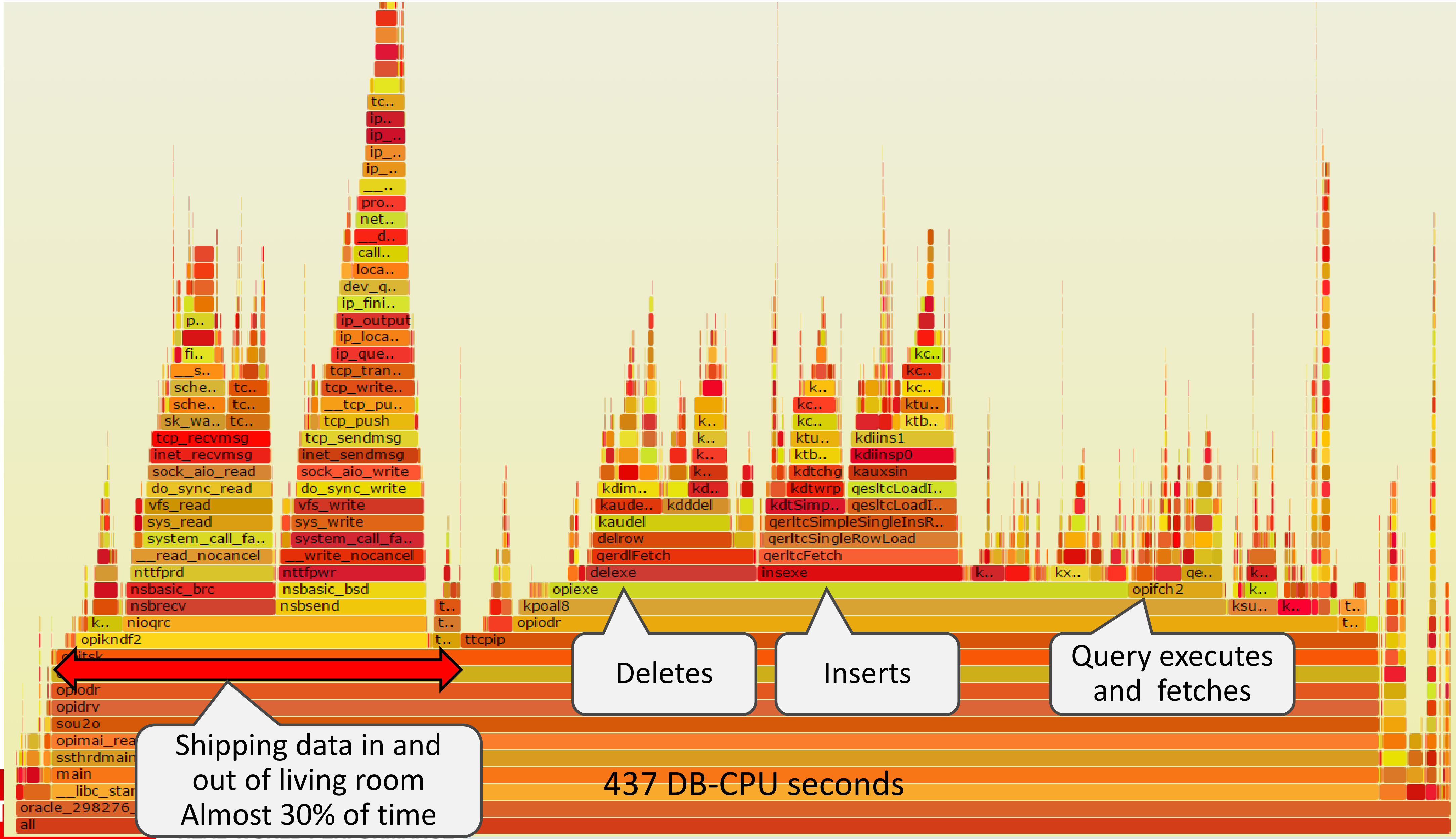
Inserts

We're inside PLSQL engine the whole time

And this is PLSQL code lines execution

204 DB-CPU seconds

Oracle Server Flamegraph Java/JDBC



Deletes

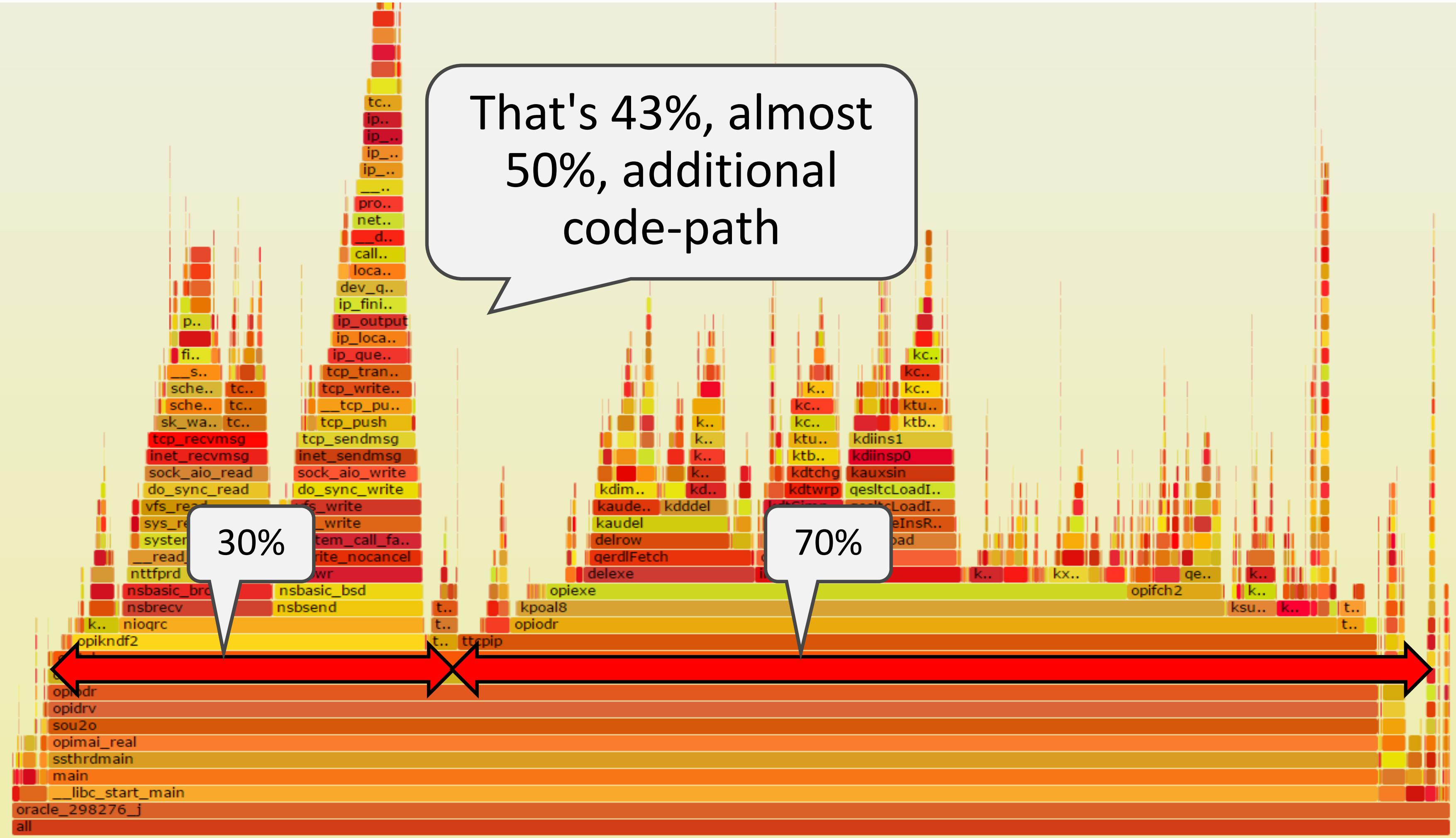
Inserts

Query executes and fetches

Shipping data in and out of living room Almost 30% of time

437 DB-CPU seconds

That 30% Is in Fact a 43% Increase on Top Of the 70%

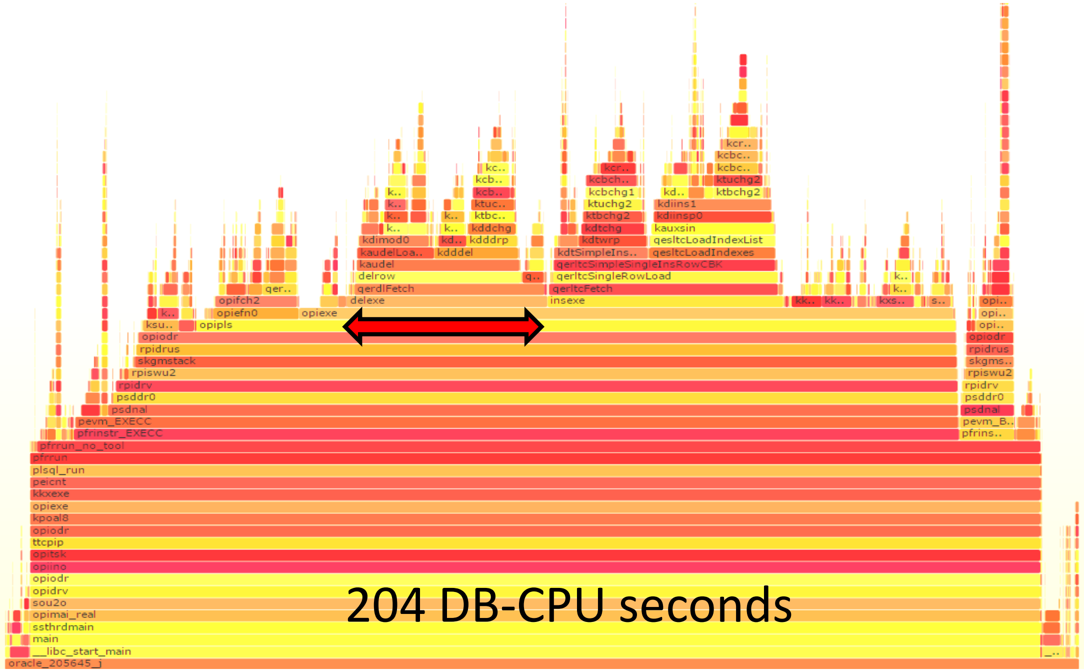


That's 43%, almost 50%, additional code-path

30%

70%

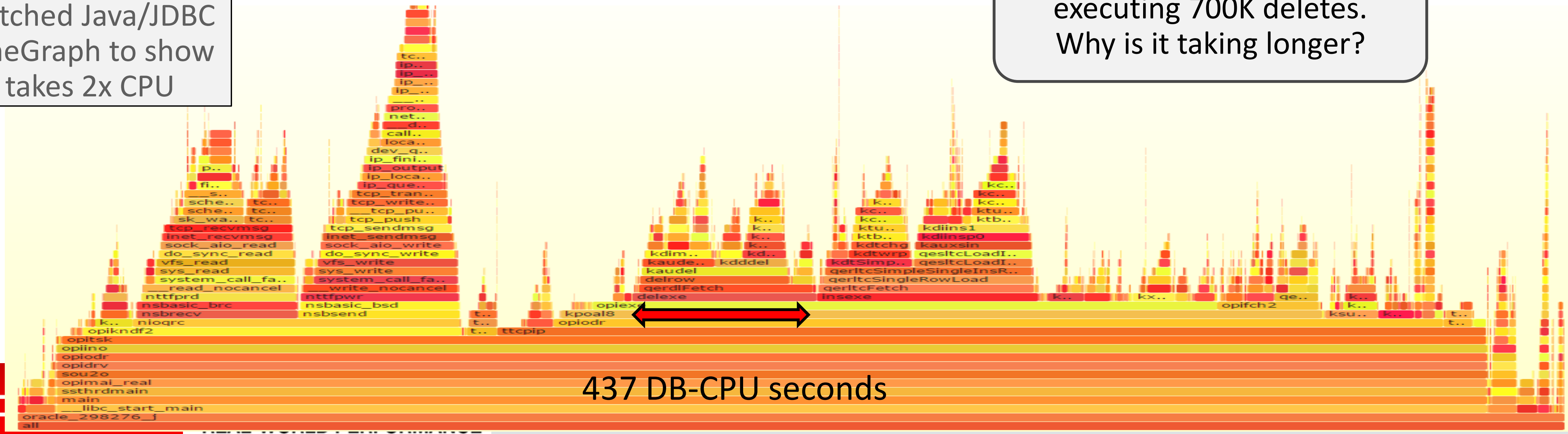
Digging Deeper



204 DB-CPU seconds

That's exactly the same code executing 700K deletes. Why is it taking longer?

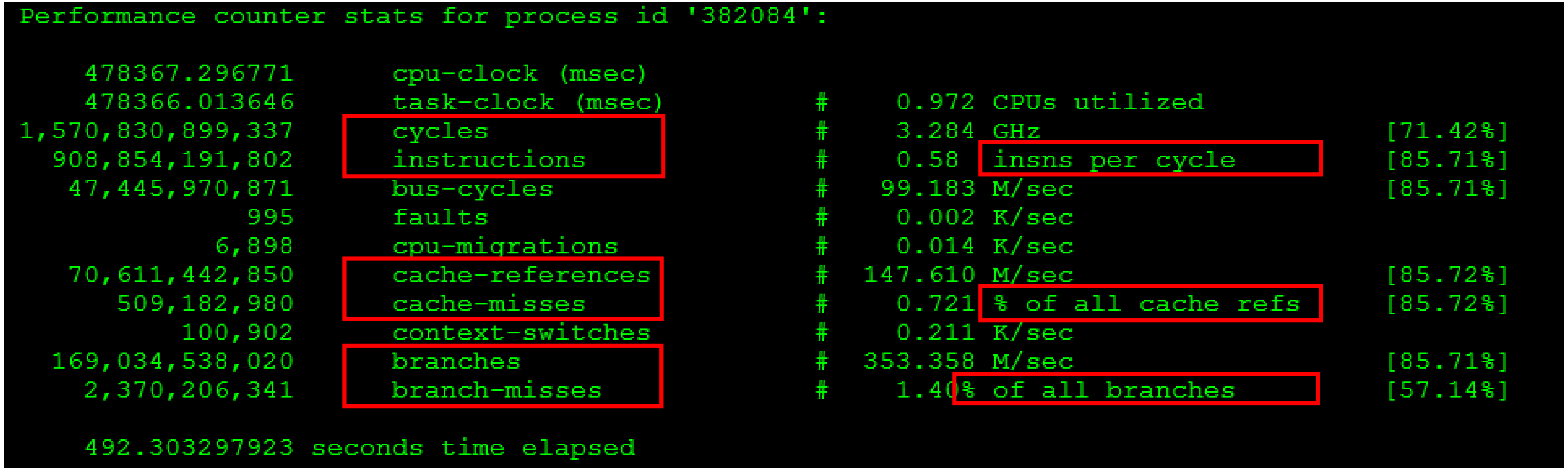
Stretched Java/JDBC FlameGraph to show it takes 2x CPU



437 DB-CPU seconds

Digging Deeper: CPU Efficiency

- Why is same code using more CPU cycles for NoPlsql?
- Let's use "perf stat" to get some insight here
  – Reports CPU usage of a pid

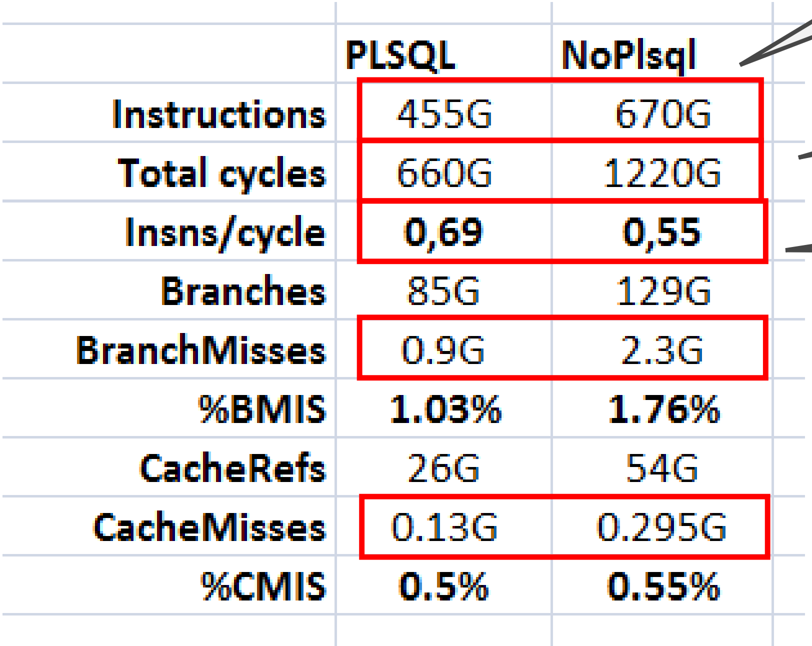


```
Performance counter stats for process id '382084':

    478367.296771      cpu-clock (msec)
    478366.013646      task-clock (msec)          #    0.972 CPUs utilized
1,570,830,899,337      cycles                     #    3.284 GHz                     [71.42%]
  908,854,191,802      instructions               #    0.58  insns per cycle         [85.71%]
   47,445,970,871      bus-cycles                 #   99.183 M/sec                   [85.71%]
              995      faults                     #    0.002 K/sec
            6,898      cpu-migrations             #    0.014 K/sec
   70,611,442,850      cache-references           #  147.610 M/sec                   [85.72%]
      509,182,980      cache-misses               #    0.721 % of all cache refs     [85.72%]
          100,902      context-switches           #    0.211 K/sec
  169,034,538,020      branches                   #  353.358 M/sec                   [85.71%]
    2,370,206,341      branch-misses              #    1.40% of all branches         [57.14%]

    492.303297923 seconds time elapsed
```

perf stat -e cpu-clock,task-clock,cycles,instructions,bus-cycles,faults,cpu-migrations,cache-references,cache-misses,context-switches,branches,branch-misses -p <pid>

NoPlsql Consistently Results in Worse IPC (insns per cycle)

"Perf stat" output summary
for duration of each run



	PLSQL	NoPlsql
Instructions	455G	670G
Total cycles	660G	1220G
Insns/cycle	**0,69**	**0,55**
Branches	85G	129G
BranchMisses	0.9G	2.3G
%BMIS	**1.03%**	**1.76%**
CacheRefs	26G	54G
CacheMisses	0.13G	0.295G
%CMIS	**0.5%**	**0.55%**

50% more instructions

Requiring 90% more CPU

Considerable worse IPC
<u>Basically means: you run on a slower CPU</u>

Caused by more branch misses

And more cache misses

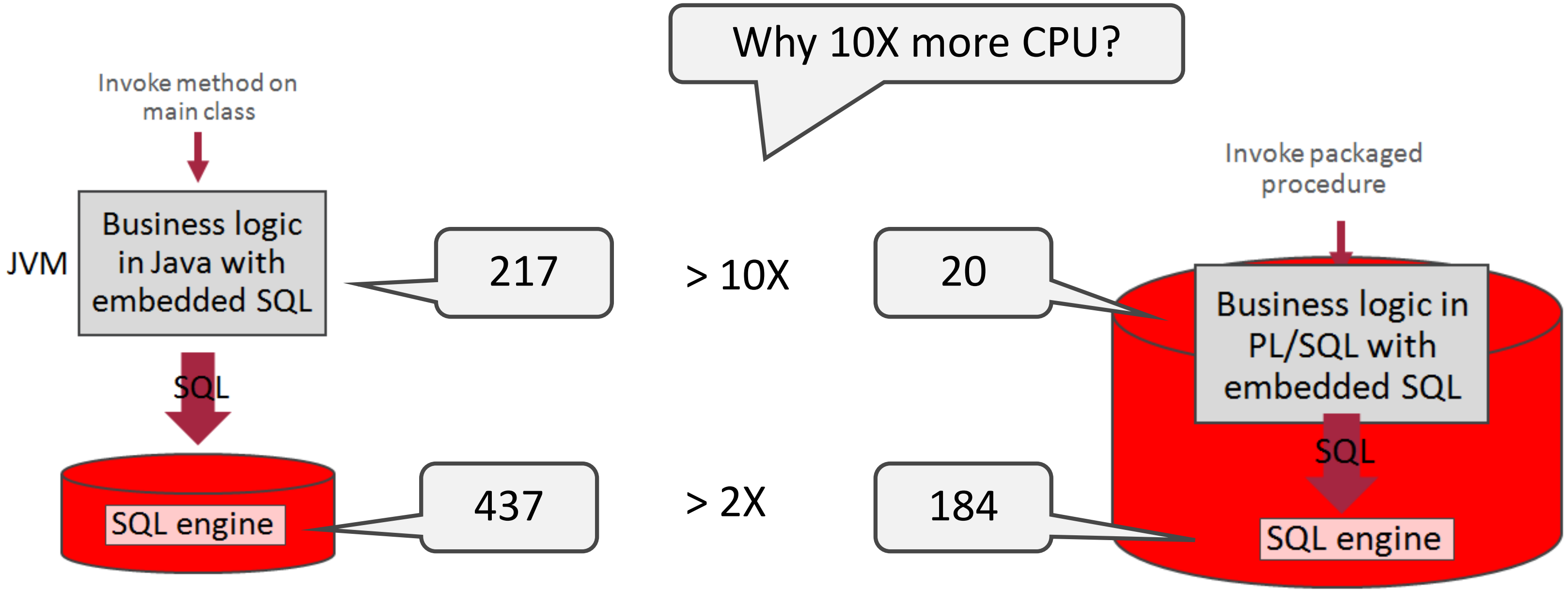
Any NoPlsql Approach Will Suffer From This

- OS has to wake up for every incoming SQL call
  - To service the network interrupt, find process associated with socket
- Schedule that dedicated server process to start running
- Once it runs, hopefully on same core as previous call, code+data caches likely full with other PID's stuff

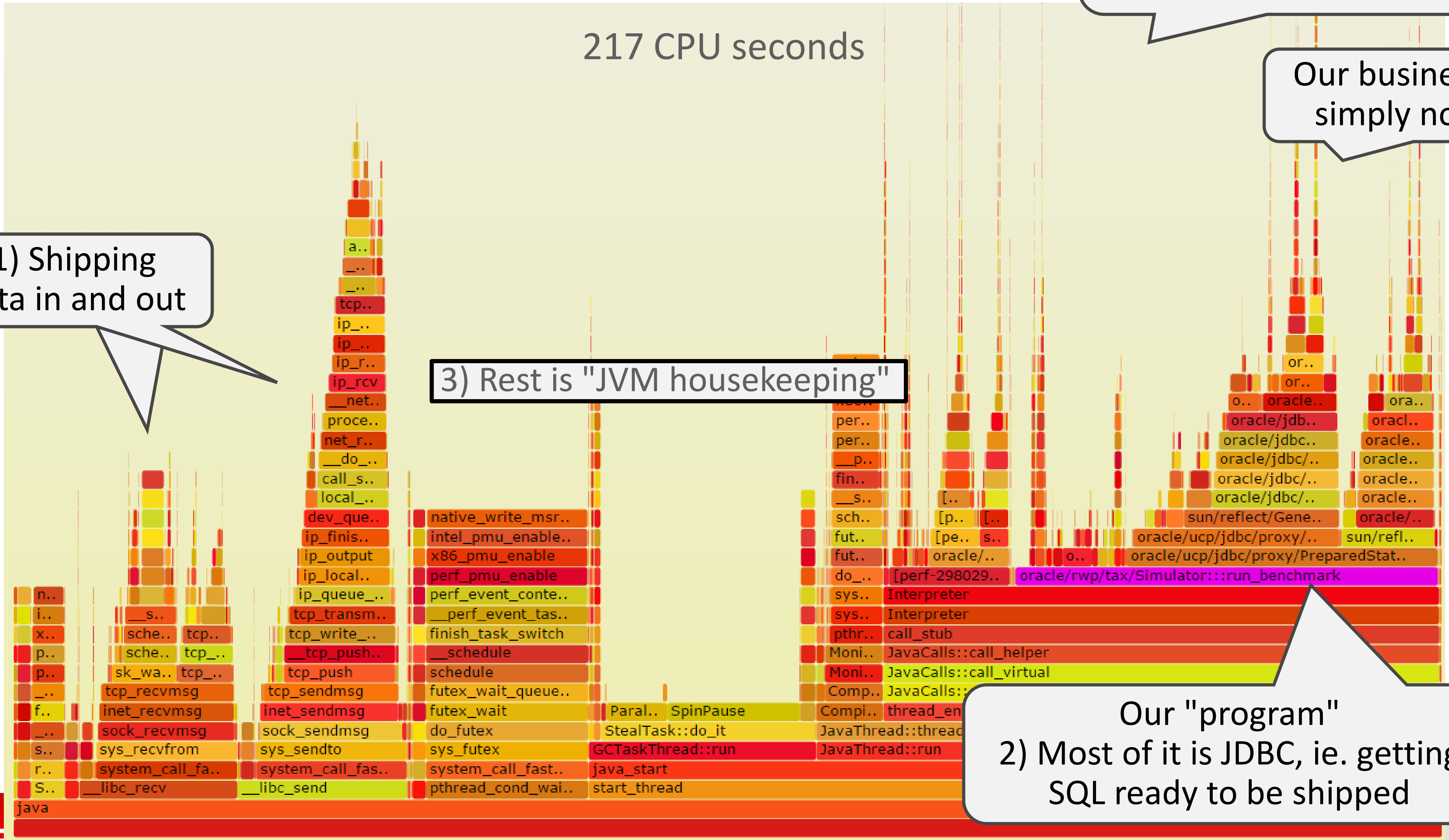
- These tests were on idle server: on busy server expect this phenomenon to become worse

What About Executing the Business Logic?

- The "app-server side": quite interesting too…
  - Java/JDBC : 217 CPU seconds (11 minutes busy 33% in JVM)
  - PL/SQL : 20 CPU seconds PLSQL execution time (Time Model in AWR)
- Ten times more expensive…


Why 10X more CPU?
Invoke method on main class
JVM
Business logic in Java with embedded SQL
SQL
SQL engine
217
437
> 10X
> 2X
20
184
Invoke packaged procedure
Business logic in PL/SQL with embedded SQL
SQL
SQL engine

JVM FlameGraph Java/JDBC



1) Shipping data in and out

3) Rest is "JVM housekeeping"

Why 10X more cpu? PLSQL doesn't have any of these three overheads

Our business logic is simply not visible

Our "program"
2) Most of it is JDBC, ie. getting SQL ready to be shipped

ORACLE

REAL-WORLD PERFORMANCE

Comparing NoPlsql and ThickDB

- If you execute many single-row, single-table SQL
  You start noticing overhead if SQL is not submitted from PL/SQL
  - Both at database server and at application server

- Layered (MVC) software architectures come with considerable CPU cost
  - Executing code through many object-oriented micro layers is not free

Embracing Set-Based SQL

- Once you're in PL/SQL opportunities for set-based SQL open up naturally
  - NoPlsql SW architectures simply prevent this as SQL is invisible
- Often parts of business logic can be rewritten into set-based SQL
  - This pushes business logic further down, from PL/SQL into SQL
- RWP's consistent experience has been:
  - From row-by-row to set-based ➔ speedups of up to 2 orders of magnitude
  - 100X faster is not uncommon

ORACLE

ORACLE REAL-WORLD PERFORMANCE

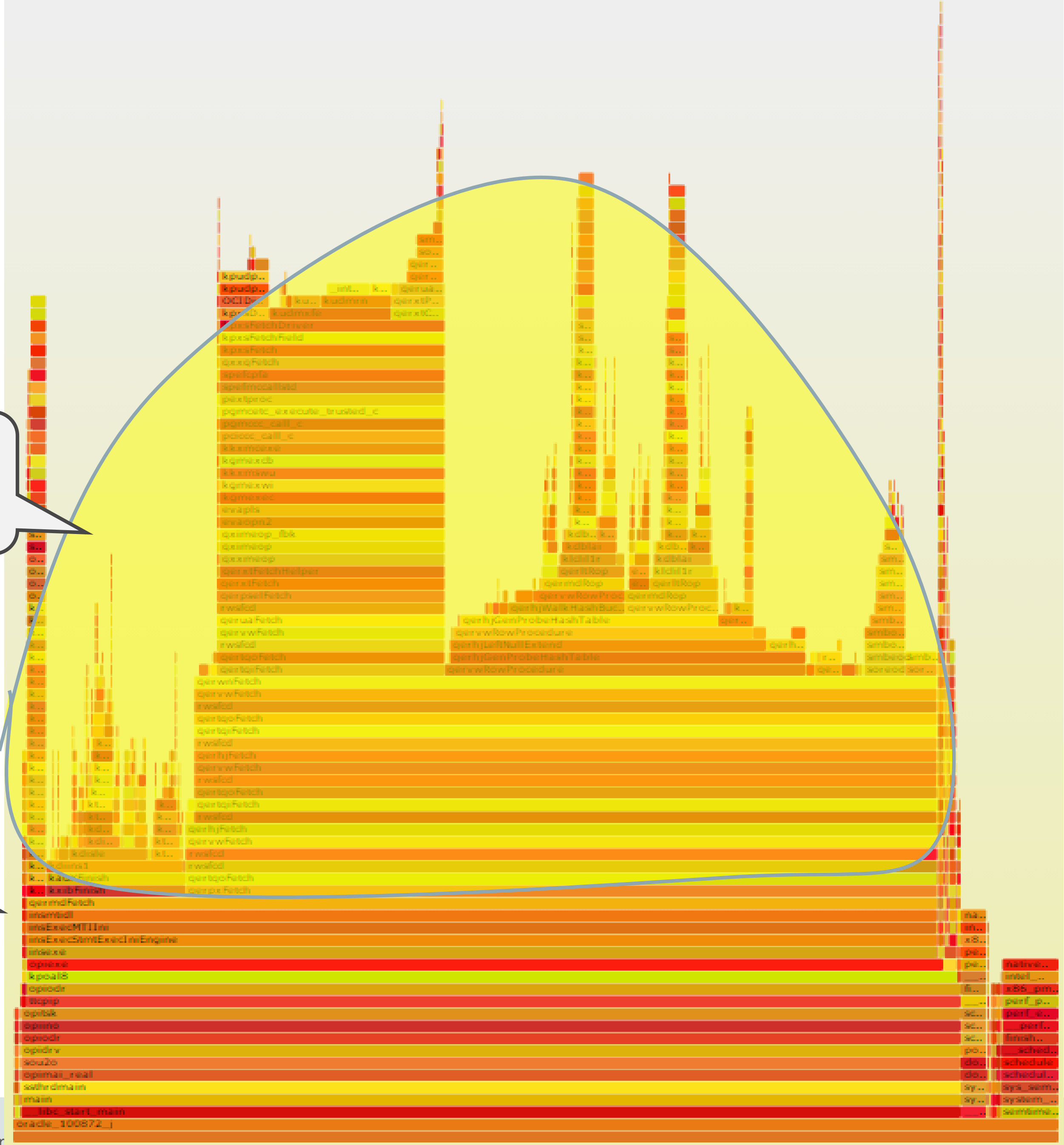
Our Example Batch Program

- Able to rewrite using set-based multi-table insert statements (MTI)

- Row-by-row Java/JDBC used : 437 DB-CPU seconds
- Row-by-row PLSQL used : 204 DB-CPU seconds
- Set-based uses : **7 DB-CPU seconds**

Flamegraph set-based

Set-based approaches
typically result in less hot-code size
Resulting in better cache-hit ratios
Resulting in better IPC



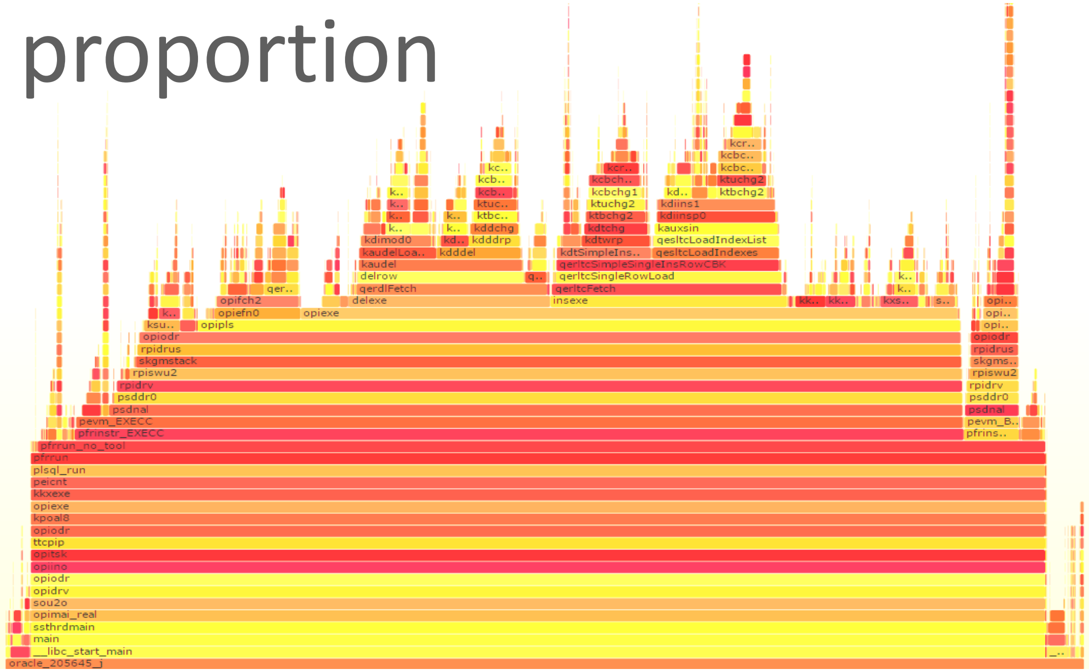
Less hot-code as MTI
fits our problem
Less cache-misses

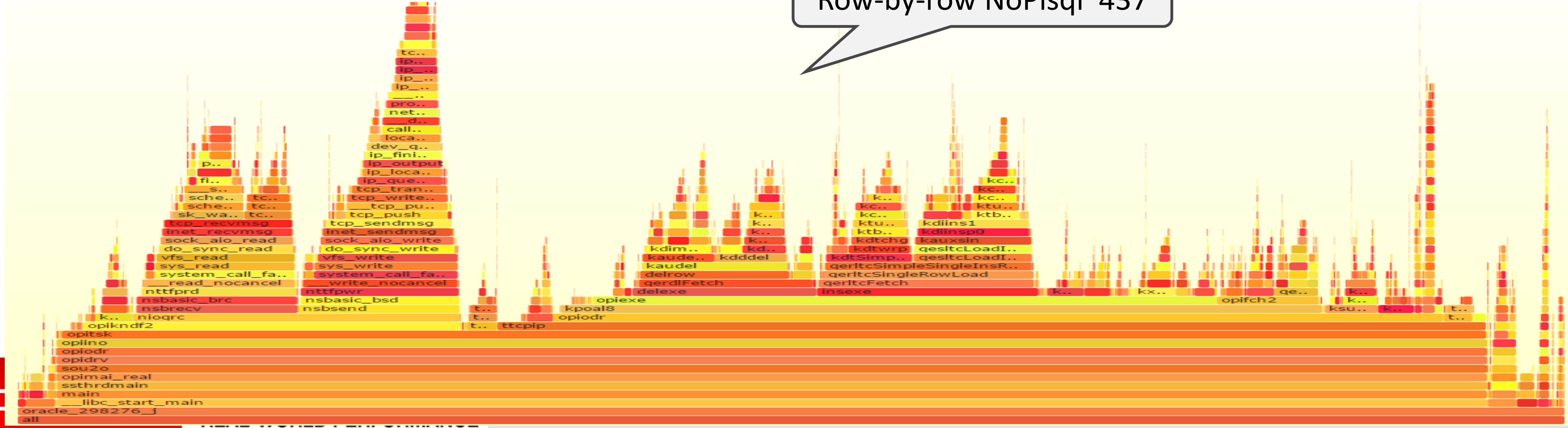
Inside MTI the
whole time

ORACLE

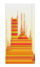
ORACLE
REAL-WORLD PERFORMANCE

In proportion





Set-based 7



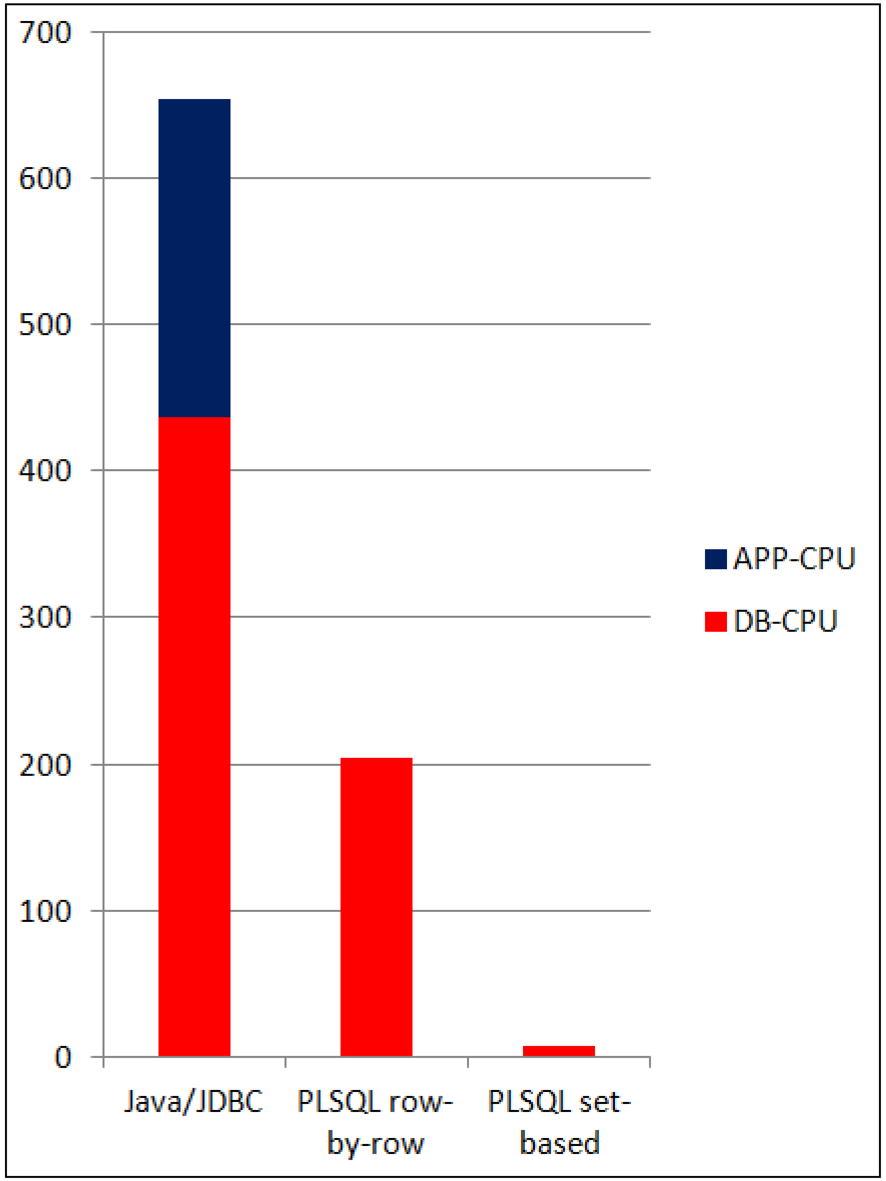
Row-by-row PLSQL 204

Row-by-row NoPlsql 437

Our Results Visualized

	Java/JDBC	PLSQL row-by-row	PLSQL set-based
DB-CPU	437	204	7
APP-CPU	217	0	0
Elapsed	660	204	7

Not a little faster…
Just think about this…



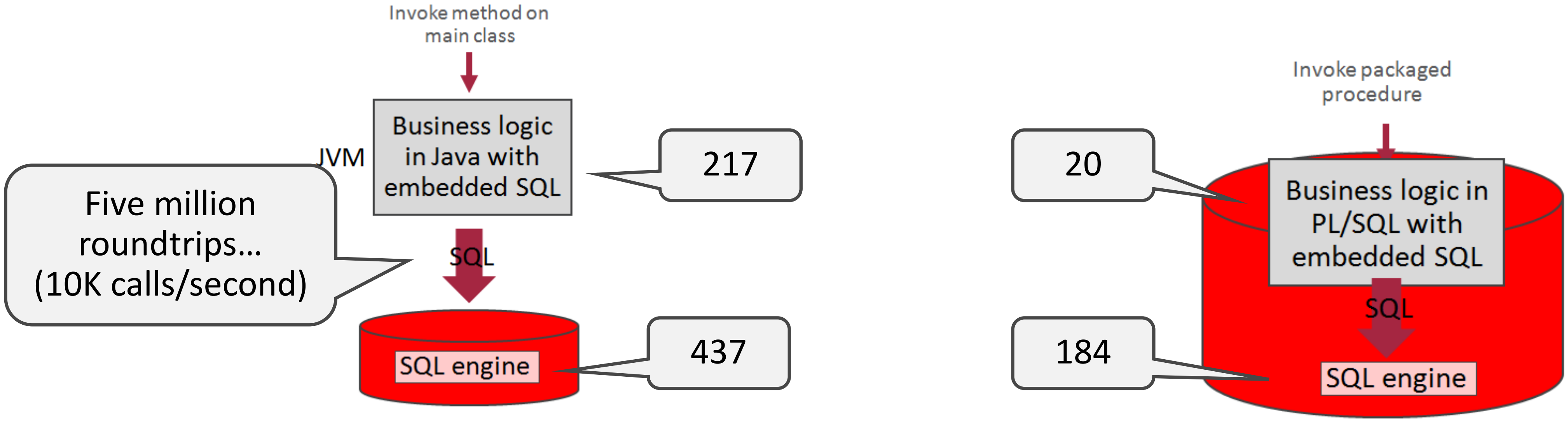
Set-Based Has Another Major Advantage

- If elapsed time of 7 seconds is still not fast enough…
- Just flip switch and have CBO generate a parallel execution plan

- In NoPlsql there's "Do it yourself parallelism" via threading
  - Requiring development time orders of magnitude more than flipping switch

Two additional points to be made (1/2)

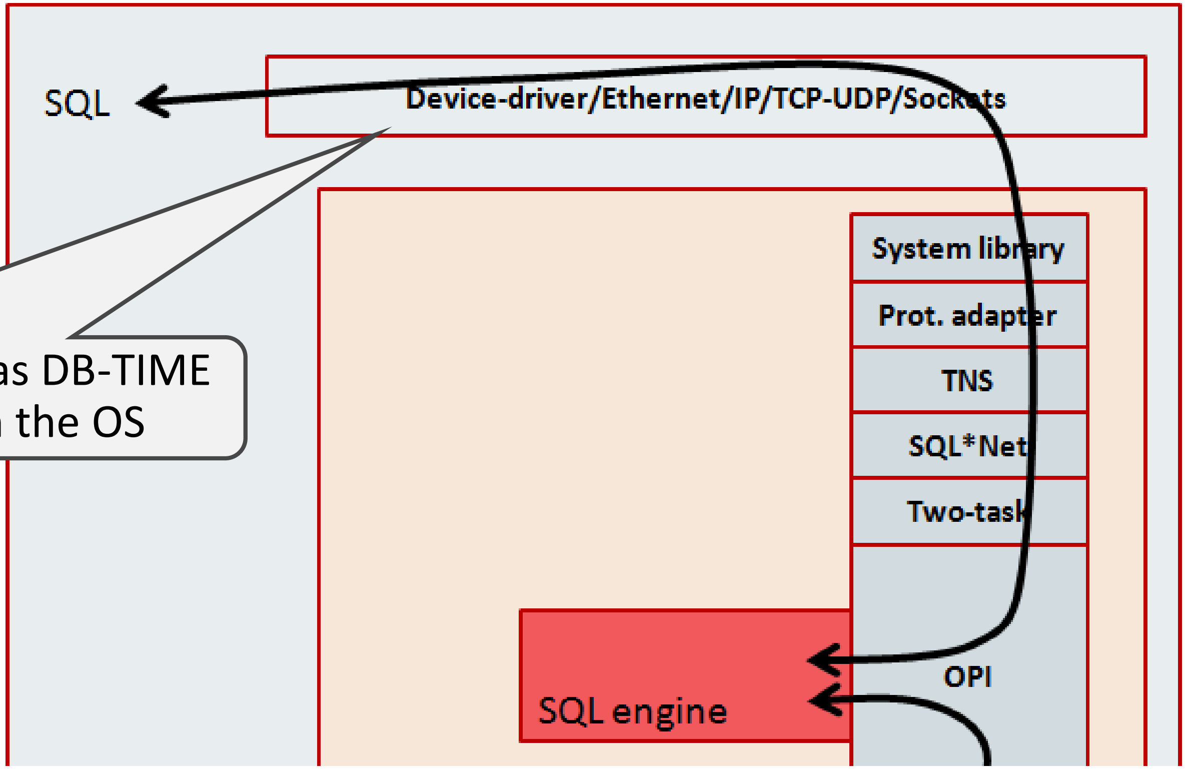
- If network were involved, elapsed time for NoPlsql would be seriously impacted


Invoke method on main class
JVM
Business logic in Java with embedded SQL
217
Five million roundtrips…
(10K calls/second)
SQL
SQL engine
437
Invoke packaged procedure
20
Business logic in PL/SQL with embedded SQL
SQL
SQL engine
184

- You’ll spend a lot of time on the wire

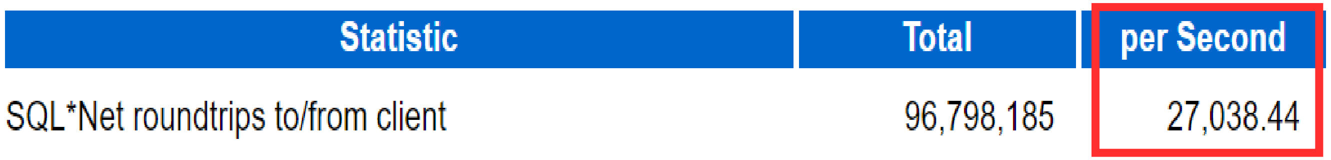
Two additional points to be made (2/2)

- NoPlsql row-by-row solutions suffer from additional sys-time in OS
  - Could easily be 5-10% additional cpu load on DB-server
  - Depends on chatty-ness of application

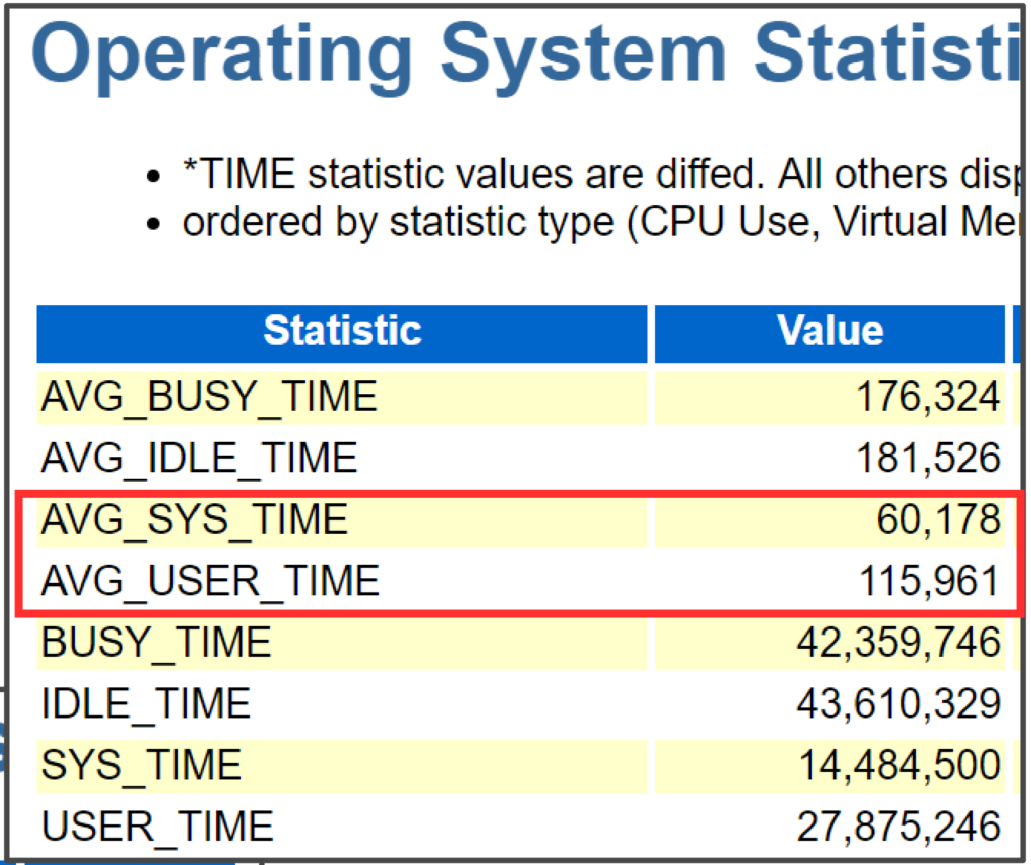


Time spent in these does not show up as DB-TIME
It shows up as additional sys-time in the OS

Check your SYS/USER CPU Ratio



Statistic	Total	per Second
SQL*Net roundtrips to/from client	96,798,185	27,038.44

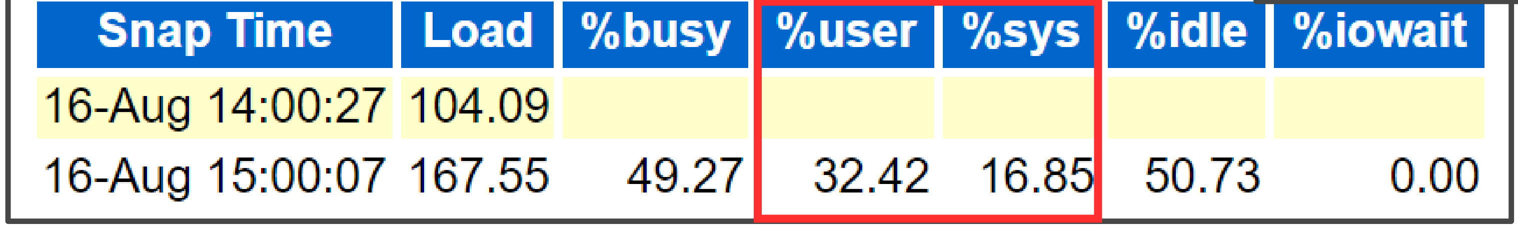


Operating System Statisti

- *TIME statistic values are diffed. All others disp
- ordered by statistic type (CPU Use, Virtual Me

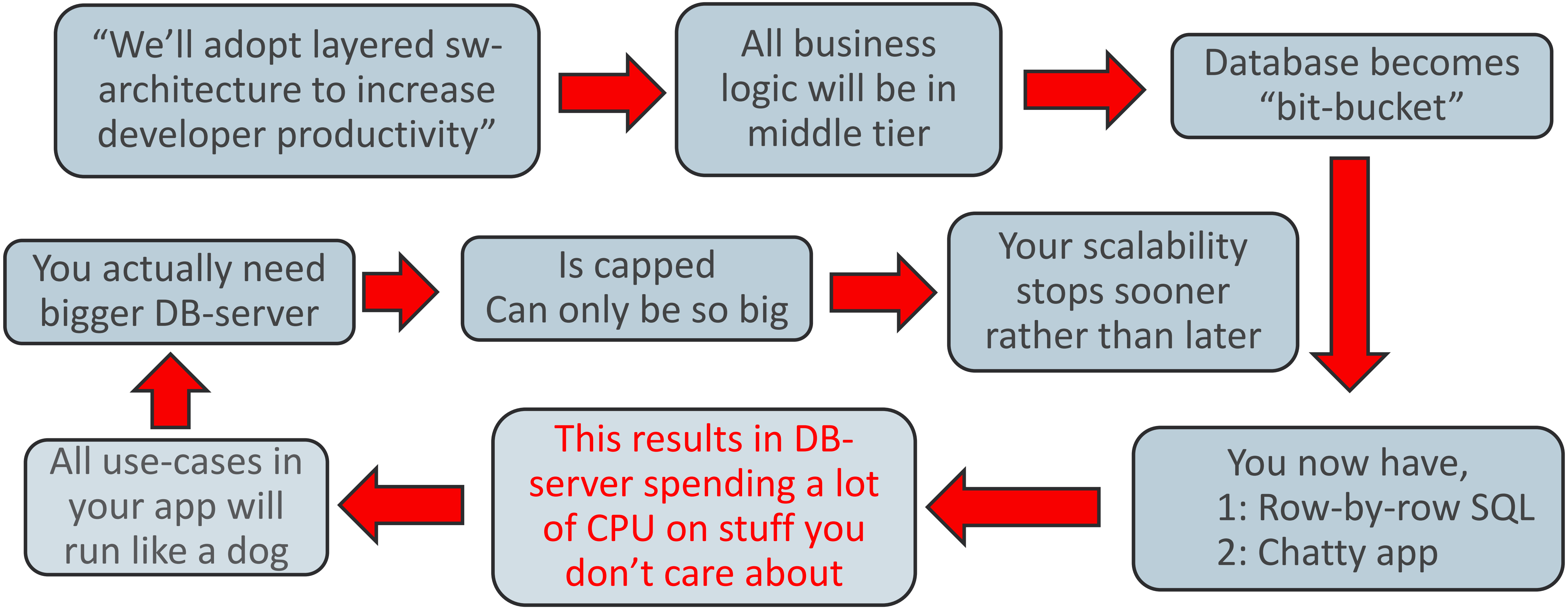
Statistic	Value
AVG_BUSY_TIME	176,324
AVG_IDLE_TIME	181,526
AVG_SYS_TIME	60,178
AVG_USER_TIME	115,961
BUSY_TIME	42,359,746
IDLE_TIME	43,610,329
SYS_TIME	14,484,500
USER_TIME	27,875,246

Operating System Statistics



Snap Time	Load	%busy	%user	%sys	%idle	%iowait
16-Aug 14:00:27	104.09					
16-Aug 15:00:07	167.55	49.27	32.42	16.85	50.73	0.00

Debunking “Keep Business Logic Outside Database”



“We’ll adopt layered sw-architecture to increase developer productivity”

All business logic will be in middle tier

Database becomes “bit-bucket”

You now have,
1: Row-by-row SQL
2: Chatty app

This results in DB-server spending a lot of CPU on stuff you don’t care about

All use-cases in your app will run like a dog

You actually need bigger DB-server

Is capped
Can only be so big

Your scalability stops sooner rather than later

Roadmap

1 Business Logic

2 What Is SmartDB?

3 Some History and Observations

4 Issues With Other Approaches

5 Debunking Performance and Scalability Argument

6 Closing Remarks

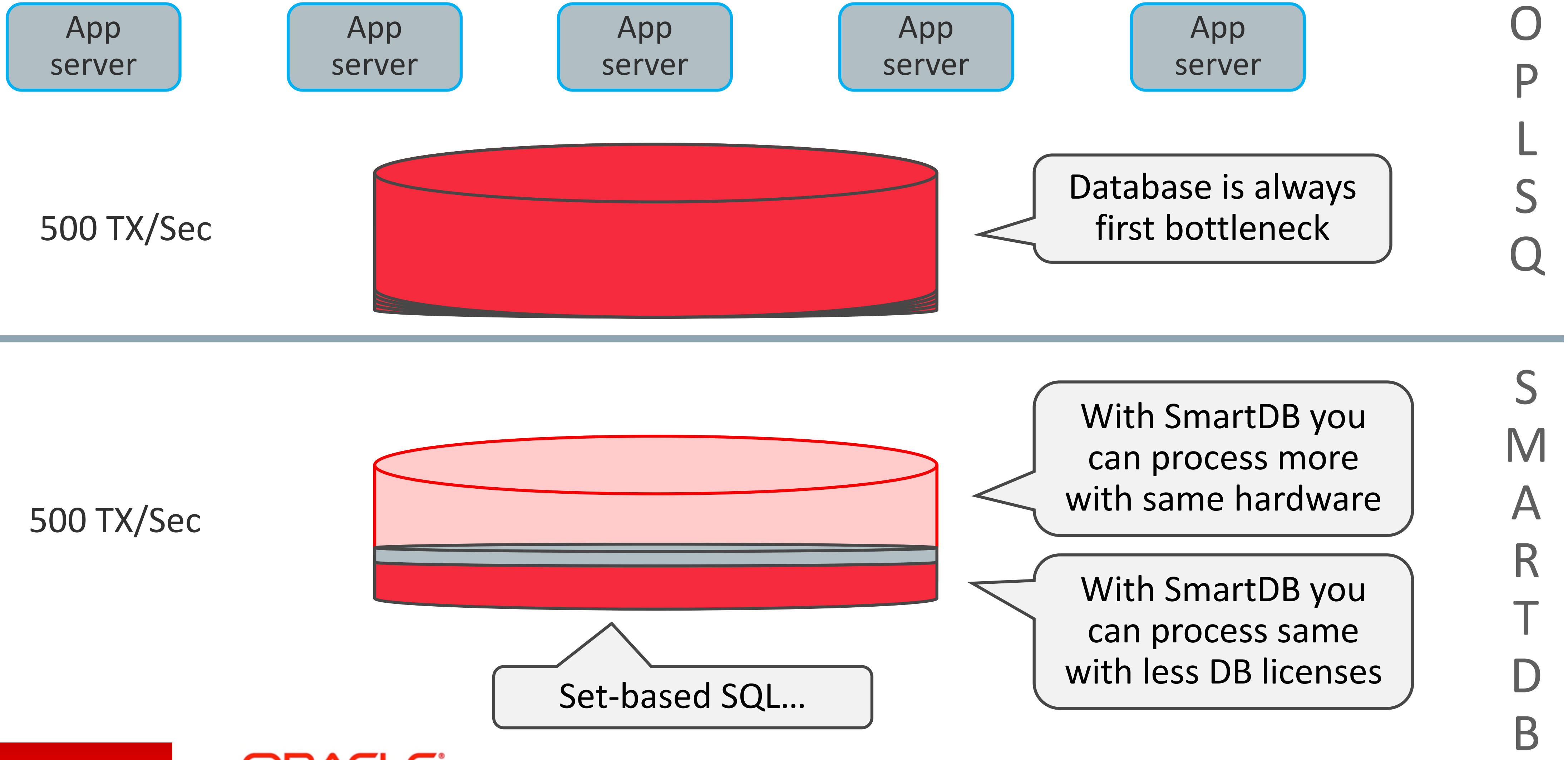
What does all this mean?

- Trying to scale your NoPlsql application via many cheap middle tier servers running BL <u>will saturate your database server way earlier</u> than when you had employed SmartDB approach for your application
- Or,
- You can service more application users on the same database server if you use the SmartDB approach

➔ Using database as processing engine saves you money

➔ Using database as bit-bucket costs you money

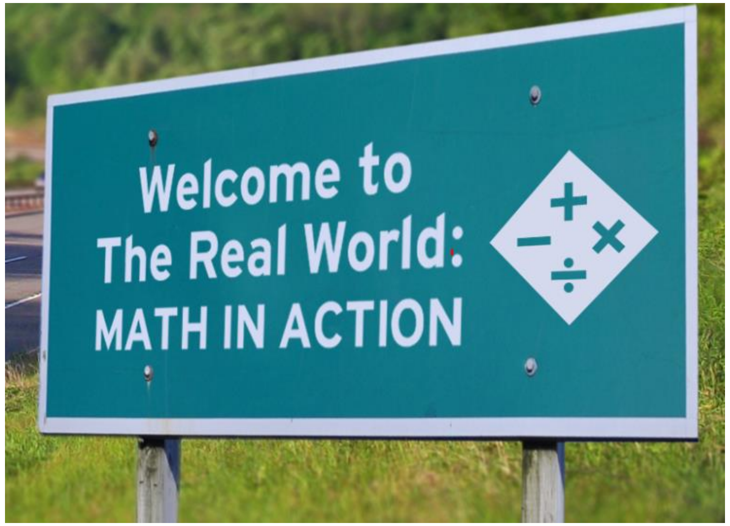
The Implication Of All This, Visualized


App server
App server
App server
App server
App server
500 TX/Sec
Database is always first bottleneck
N O P L S Q
500 TX/Sec
With SmartDB you can process more with same hardware
With SmartDB you can process same with less DB licenses
Set-based SQL…
S M A R T D B

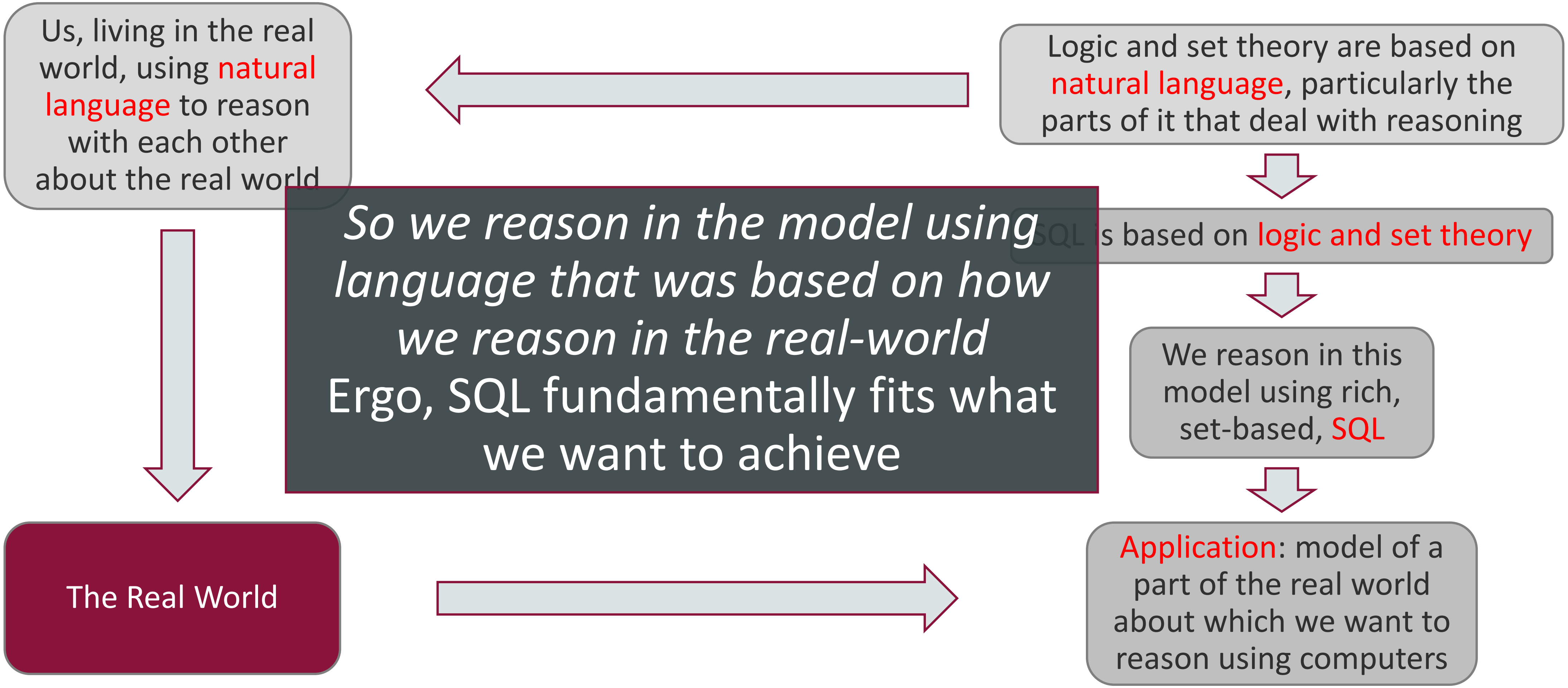
2: SQL Isn't Accidental: Au-Contraire, It's Fundamental

- There are nearly always opportunities for your business logic to be pushed into set-based SQL

- Why is this the case?
- There's a fundamental reason for this...



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



Us, living in the real world, using natural language to reason with each other about the real world

Logic and set theory are based on natural language, particularly the parts of it that deal with reasoning

SQL is based on logic and set theory

We reason in this model using rich, set-based, SQL

Application: model of a part of the real world about which we want to reason using computers

The Real World

So we reason in the 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

- “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

- Don’t underestimate width and depth of SQL and PL/SQL
- And all DB features surrounding these two languages

ORACLE REAL-WORLD PERFORMANCE

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

Finishing Up

- NoPlsql has had its reign
- Arguments for its rise,
  - Have either not been delivered (code reuse, speed of development)
  - Or, have been debunked (performance and scalability)
- Current JavaScript hype brings no new arguments to table

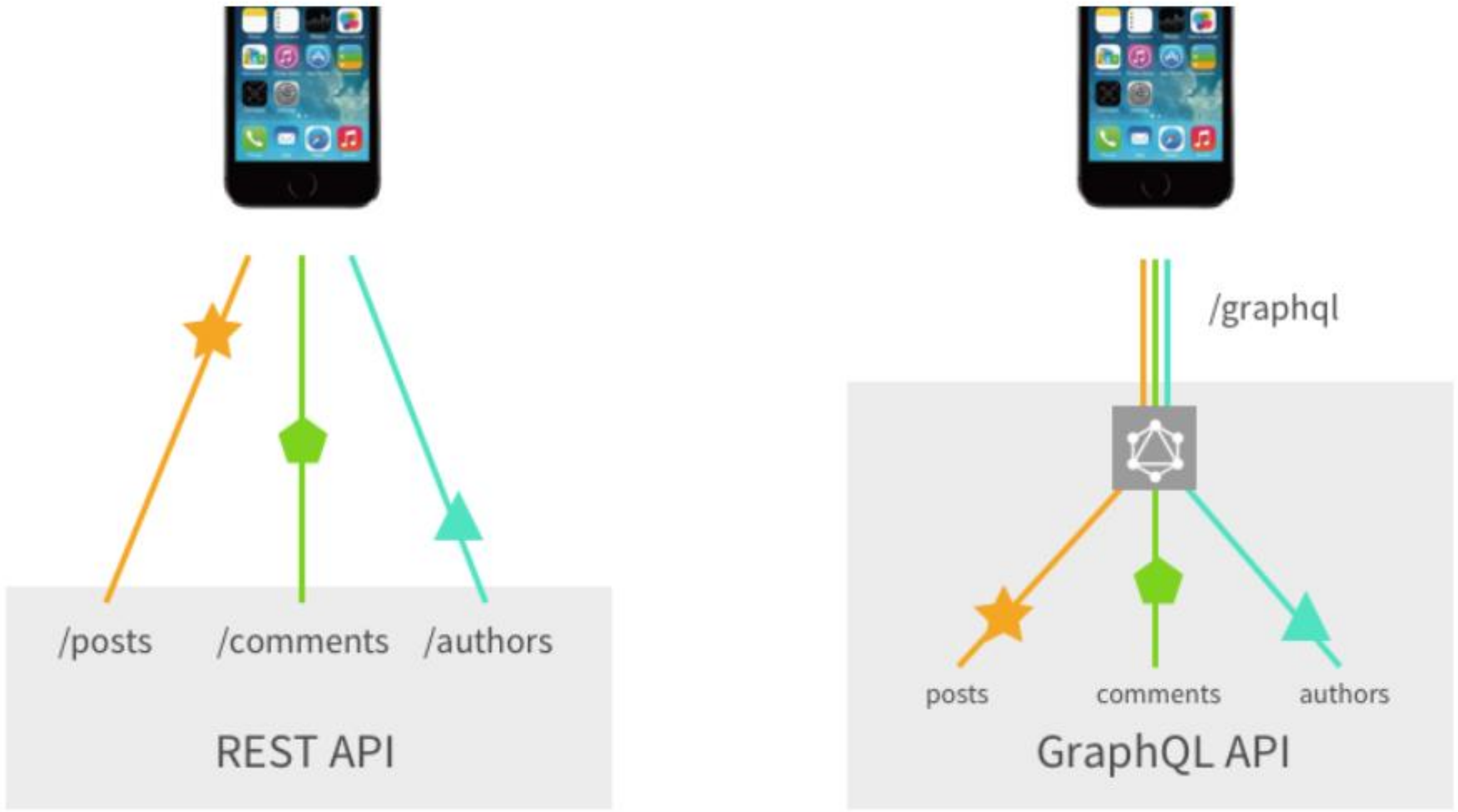
- SmartDB has survived in many (happy) pockets around the world
  - PL/SQL and SQL have moved forward a lot since 2001
  - It's high time for resurgence of using database as processing engine
  - In Part 2 we will discuss how to adopt SmartDB approach

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

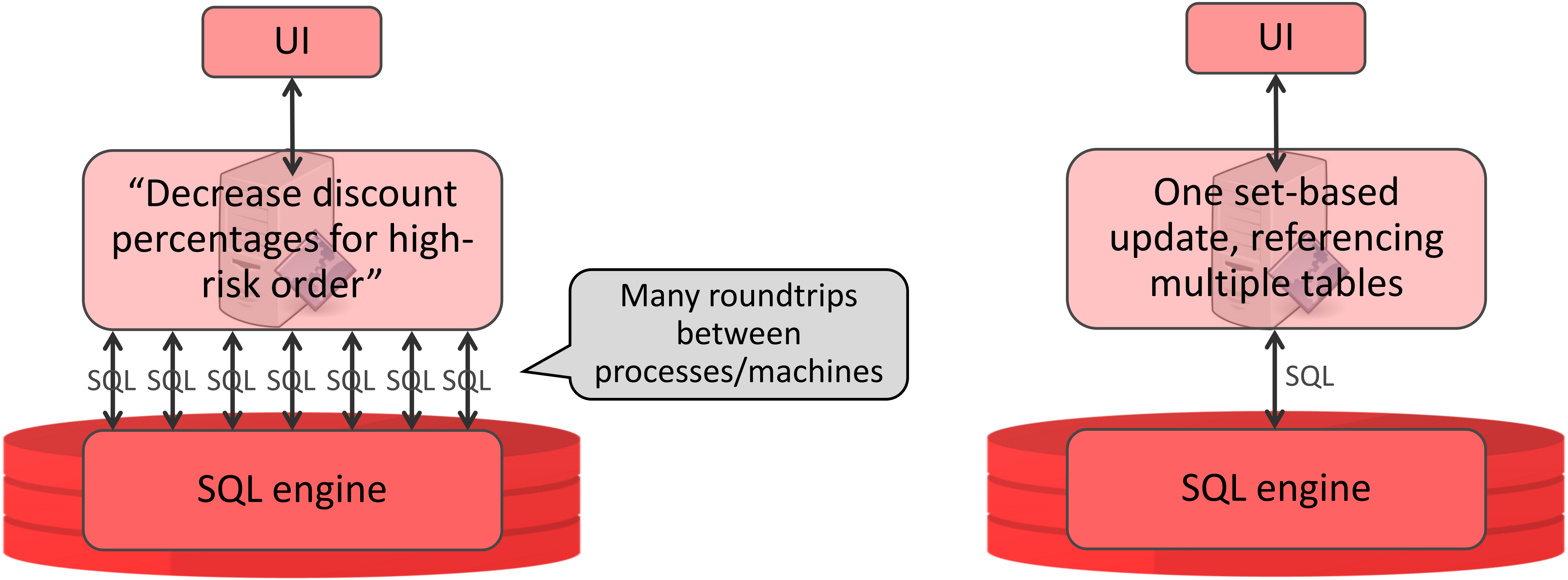
Integrated Cloud

Applications & Platform Services

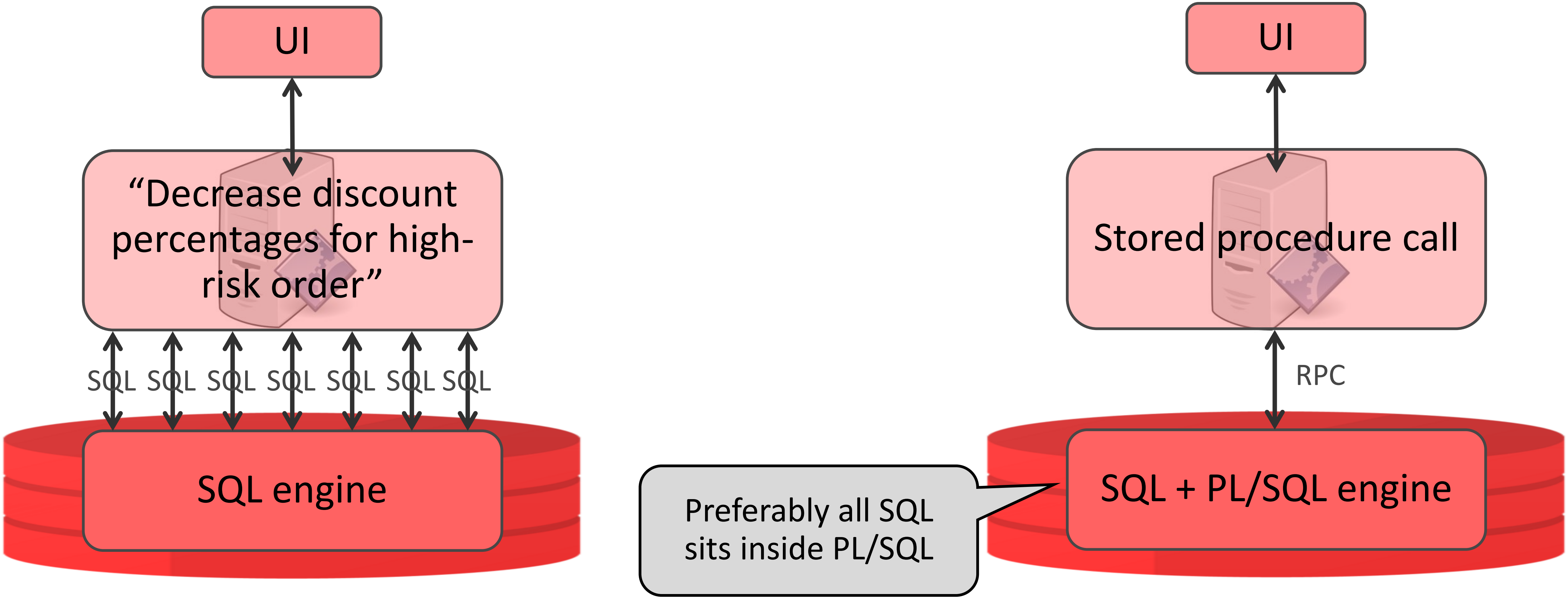


An artists' interpretation of fetching resources with multiple REST roundtrips vs. one GraphQL request

“Chatty” Applications


UI
“Decrease discount percentages for high-risk order”
SQL SQL SQL SQL SQL SQL SQL
SQL engine
Many roundtrips between processes/machines
UI
One set-based update, referencing multiple tables
SQL
SQL engine

All SQL Sits Inside PL/SQL



UI

“Decrease discount percentages for high-risk order”

SQL SQL SQL SQL SQL SQL SQL

SQL engine

UI

Stored procedure call

RPC

SQL + PL/SQL engine

Preferably all SQL sits inside PL/SQL