Oracle Database: Processing Engine or Persistence Layer?

John Clarke
Software Development Director
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.

What is Real-World Performance in 2018?

Bridging the Divide from Today's Performance to What is Possible

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Real-World Performance 2018

Who We Are

- Part of the Database Development Organization
- Global Team located in USA, Europe, Asia
- 350+ combined years of Oracle database experience
- Innovate to achieve exceptional Database Performance
- Our methods:
  - Use the product as it was designed to be used
  - Numerical and logical debugging techniques
  - Educate others about the best performance methods and techniques
  - Avoid and eliminate “tuning” by hacking/guessing/luck

Do you have the ability to *choose* your performance?

Do you move data to processing or processing to data?

Are you happy with incremental performance improvements?

Acknowledgements

- Toon Koppelaars, RWP
  - Performed much of the research & authored much of the content
  - Leads #SmartDB charge from Real-World Performance
- Bryn Llewellyn, PLSQL Product Manager
- Many other RWP engineers over the years
- Oracle's Fusion Applications/SaaS teams, designs, and solutions, which motivated us to do the research

Introduction

1. Introduction
2. Layered Architecture: History, Landscape, and Issues
3. Demos and Technical Stuff
4. Big Picture
5. Concluding Thoughts

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Application Architecture
How Do You Design apps?

- If you had to validate a single row on a web form, how would you design your app?
- What if you had to validate millions of rows that arrived as a batch?
- Do you do this?

Design(Many rows) = Design(One row) * #Rows

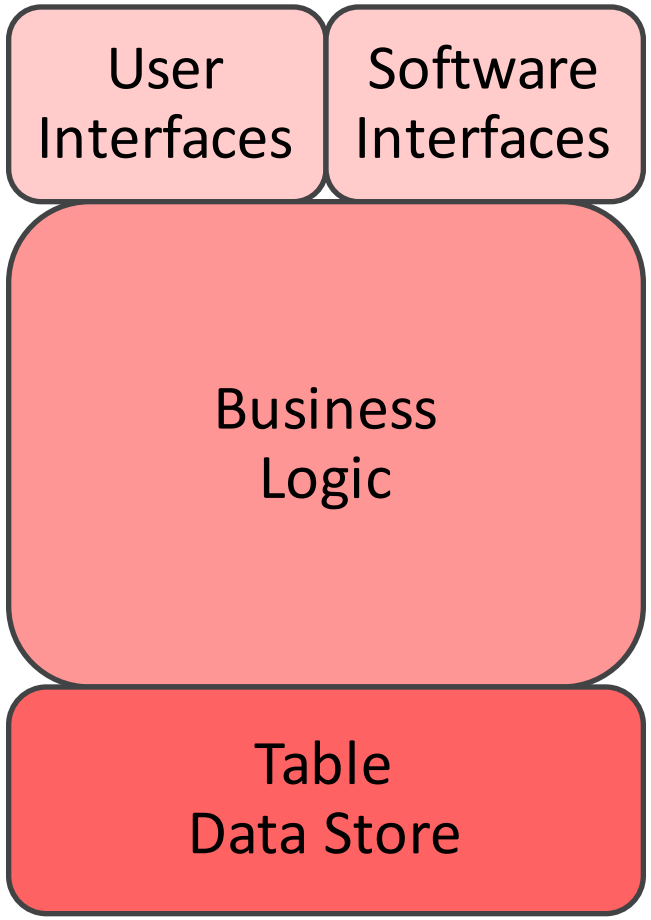
Some Context

- The database -> a processing engine, or persistence layer?
- OLTP: transactional enterprise applications
  - Data store as foundation, for which we use Oracle's RDBMS
  - Much/complex CRUD functionality on top
  - User interfaces, workflows, batch jobs, reports, API's to other applications
  - Potentially many users
- OLTP techniques for batch programming
- Not: data-warehouse, business-intelligence, big-data
  - But CSconcepts applicable

Transactional Enterprise Applications

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

Transactional Enterprise 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

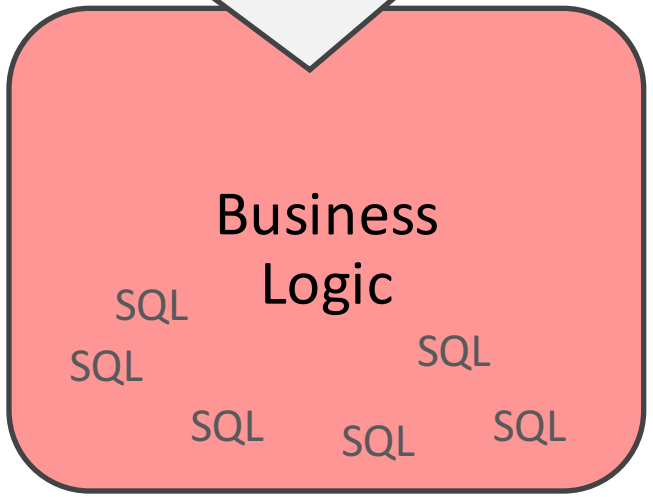
Wiki

Business logic

From Wikipedia, the free encyclopedia

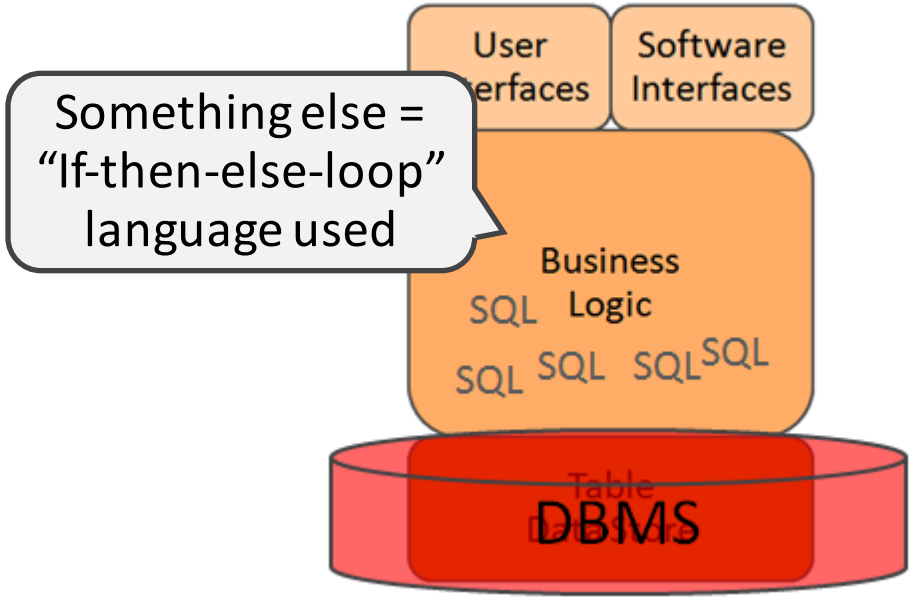
In computer software, **business logic** or **domain logic** is the part of the program that encodes the real-world business rules that determine how data can be created, displayed, stored, and changed. It is contrasted with the remainder of the software that might be concerned with lower-level details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

Conditional "If-then-else-loop" code with embedded data access statements in it
The way the business requires this to be done

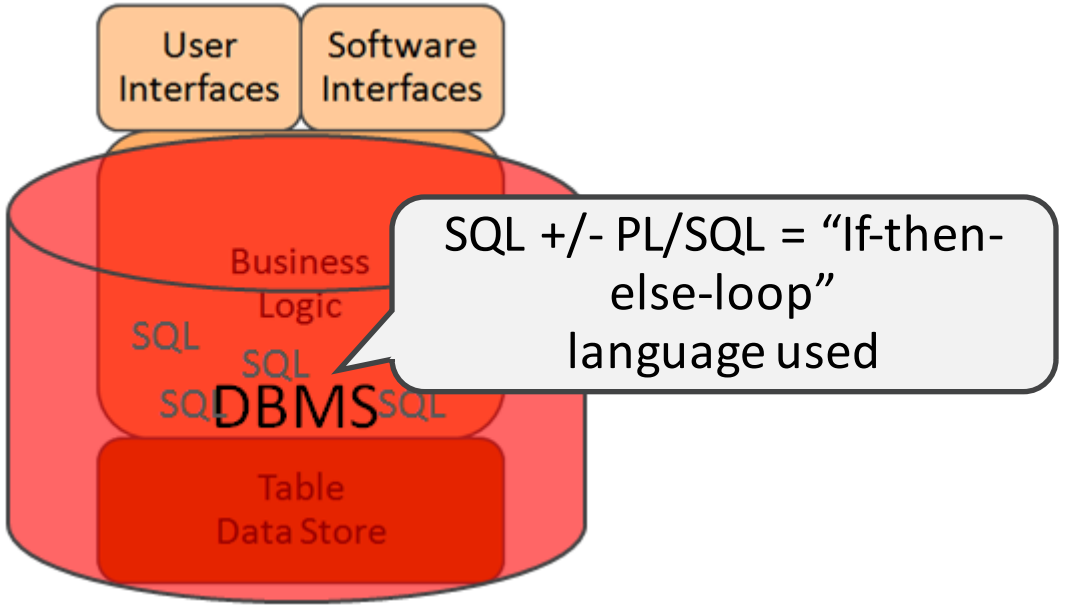


We See Two Mutually Distinct Approaches

RDBMS= Persistence Layer
"Layered Architecture"

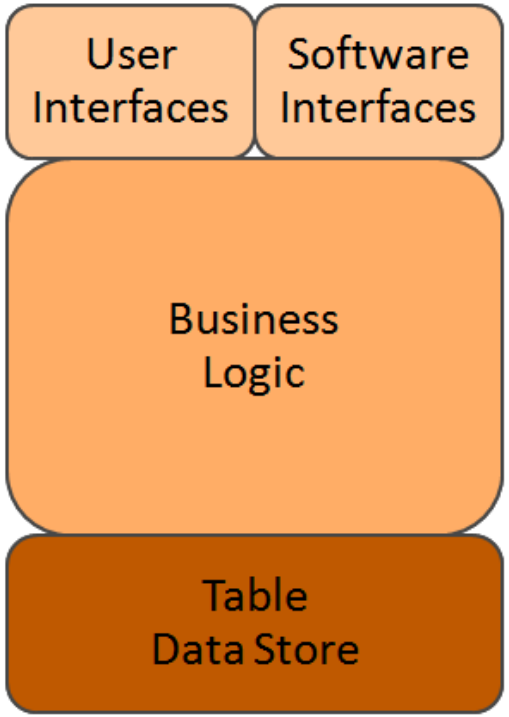


RDBMS = Processing Engine
"#SmartDB"



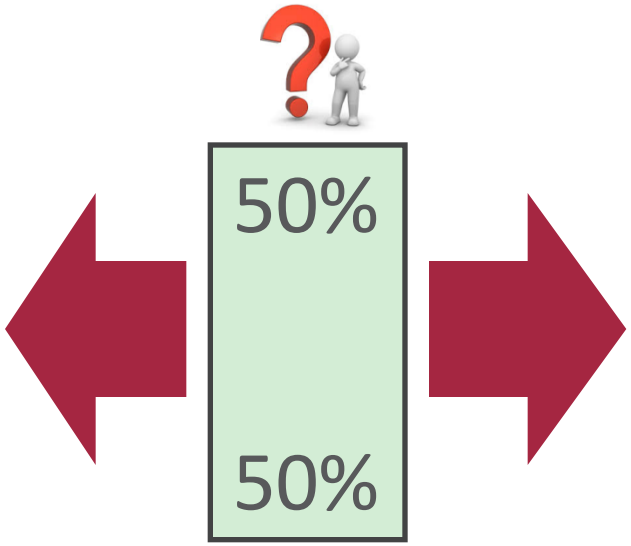
A quick poll

- In your typical DB application, what would be the distribution of CPU work between BL and SQL?



20%

80%



80%

20%

Ever Seen This?

Program Counter

Data manipulation

Clock-ticks required for instruction

81d0	mov r12, #0	i = 0	1
81d4	mul r1, r12, r12		
81d8	mov r2, r12, lsl #3		
81dc	mov r3, #15	r3 = 15	1
81e0	mla r3, r1, r3, r2	r3 = r1×r3 + r2	3
81e4	add r3, r3, #79	r3 = r3 + 79	1
81e8	str r3, [r0, r12, lsl #2]	dest[i] = r3	
81ec	add r12, r12, #1	i++	
81f0	cmp r12, #64	is i<64?	
81f4	bne 81d4	yes? branch 81d4	1
81f8	bx lr	return	1

Machine code mnemonics

Conditional branching/looping

Constitutes the *byte-stream* that is your executable program that gets run by the CPU

```
                    foo:
B8 22 11 00 FF      movl  $0xFF001122, %eax
01 CA               addl  %ecx, %edx
31 F6               xorl  %esi, %esi
53                  pushl %ebx
8B 5C 24 04         movl  4(%esp), %ebx
8D 34 48            leal  (%eax,%ecx,2), %esi
39 C3               cmpl  %eax, %ebx
72 EB               jnae  foo
C3                  retl
```

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Why Did I Show That?

- Re-introduction of term “if-then-else-loop language”
- Start thinking about how computers accomplish the tasks they’re asked to perform …

“If-then-else-loop” Languages

- Every compiled program translates to machine code which always is sequence of:
  - Moving data between main memory and on-core registers
  - Changing values in these registers
  - Computing new values using these registers
  - Comparing values in registers with literals and/or other registers
  - Conditional (or unconditional) branching
- All program languages (PL/SQL, Java, JavaScript, …) are just higher levels of abstraction to this
  = *if-then-else-loop languages*

PC	Instruction	Meaning	ET
81d0	mov r12, #0	i = 0	1
81d4	mul r1, r12, r12	r1 = i×i	3
81d8	mov r2, r12, lsl #3	r2 = i×8	1
81dc	mov r3, #15	r3 = 15	1
81e0	mla r3, r1, r3, r2	r3 = r1×r3 + r2	3
81e4	add r3, r3, #79	r3 = r3 + 79	1
81e8	str r3, [r0, r12, lsl #2]	dest[i] = r3	2
81ec	add r12, r12, #1	i++	1
81f0	cmp r12, #64	is i<64?	1
81f4	bne 81d4	yes? branch 81d4	1
81f8	bx lr	return	1

Machine code bytes	Assembly language statements
	foo:
B8 22 11 00 FF	movl $0xFF001122, %eax
01 CA	addl %ecx, %edx
31 F6	xorl %esi, %esi
53	pushl %ebx
8B 5C 24 04	movl 4(%esp), %ebx
8D 34 48	leal (%eax,%ecx,2), %esi
39 C3	cmpl %eax, %ebx
72 EB	jnae foo
C3	retl

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

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)

"if-then-else-loop" code

Primitive data access (single table, row-by-row)

Example Business Logic: All in SQL

Set-based data processing, aka "rich" SQL
Multiple tables referenced
Multiple rows processed

```
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)
```

You declare what needs to be done.
SQL exec engine & optimizer translate this into code equivalent with code on previous page

For now, we're going to assume this is how it appears

Point to be made → Business logic can appear:

- As code-lines in some programming language that issues simple (poor) SQL
- As part of (rich) SQL
- Both…

Layered Architecture: History, Landscape, and Issues

1. Introduction
2. Layered Architecture: History, Landscape, and Issues
3. Demos and Technical Stuff
4. Big Picture
5. Concluding Thoughts

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

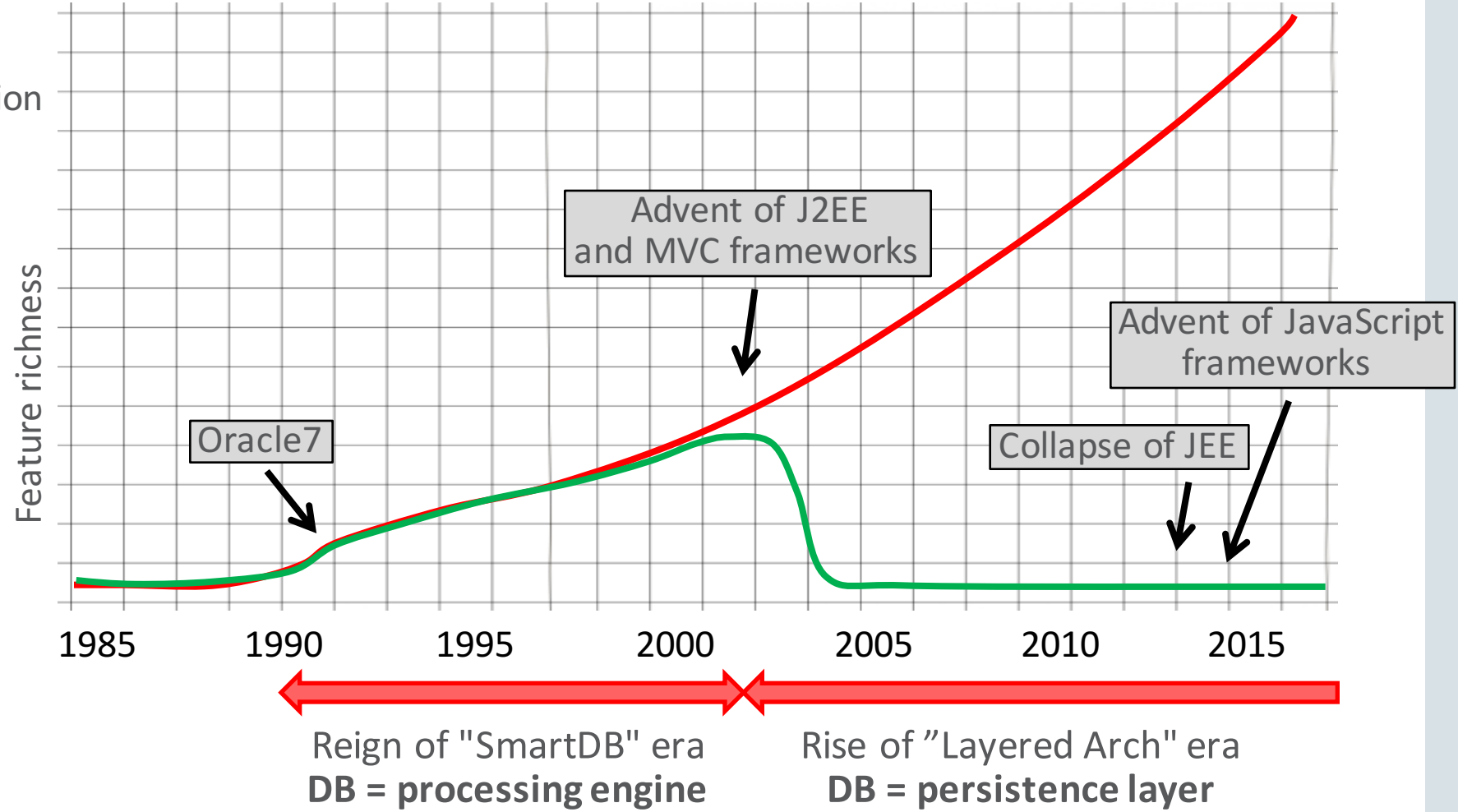
Oracle v4, v5, v6 Database Documentation

Oracle7, 8i: Database Documentation

History Observation: End of "SmartDB" Era

Features available in DB

DB-features used by application developers

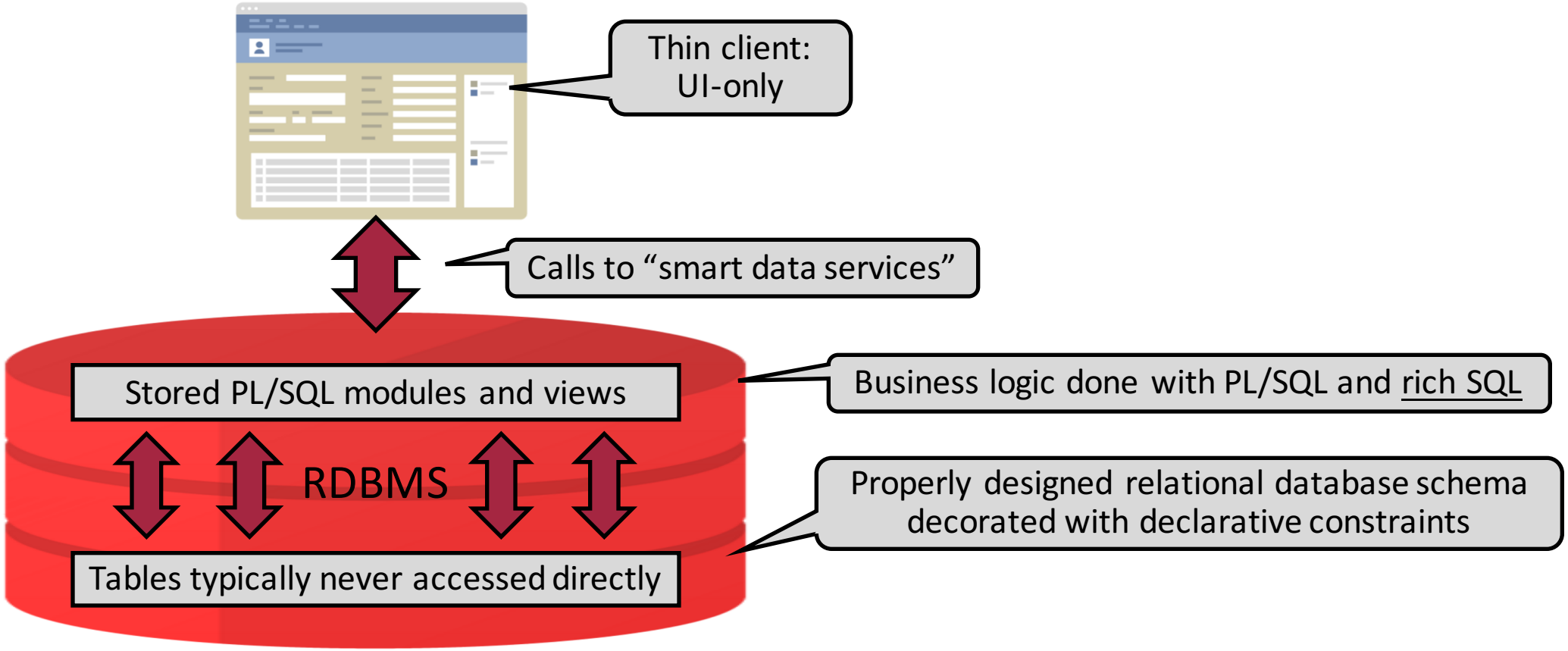


ORACLE

ORACLE
REAL-WORLD PERFORMANCE

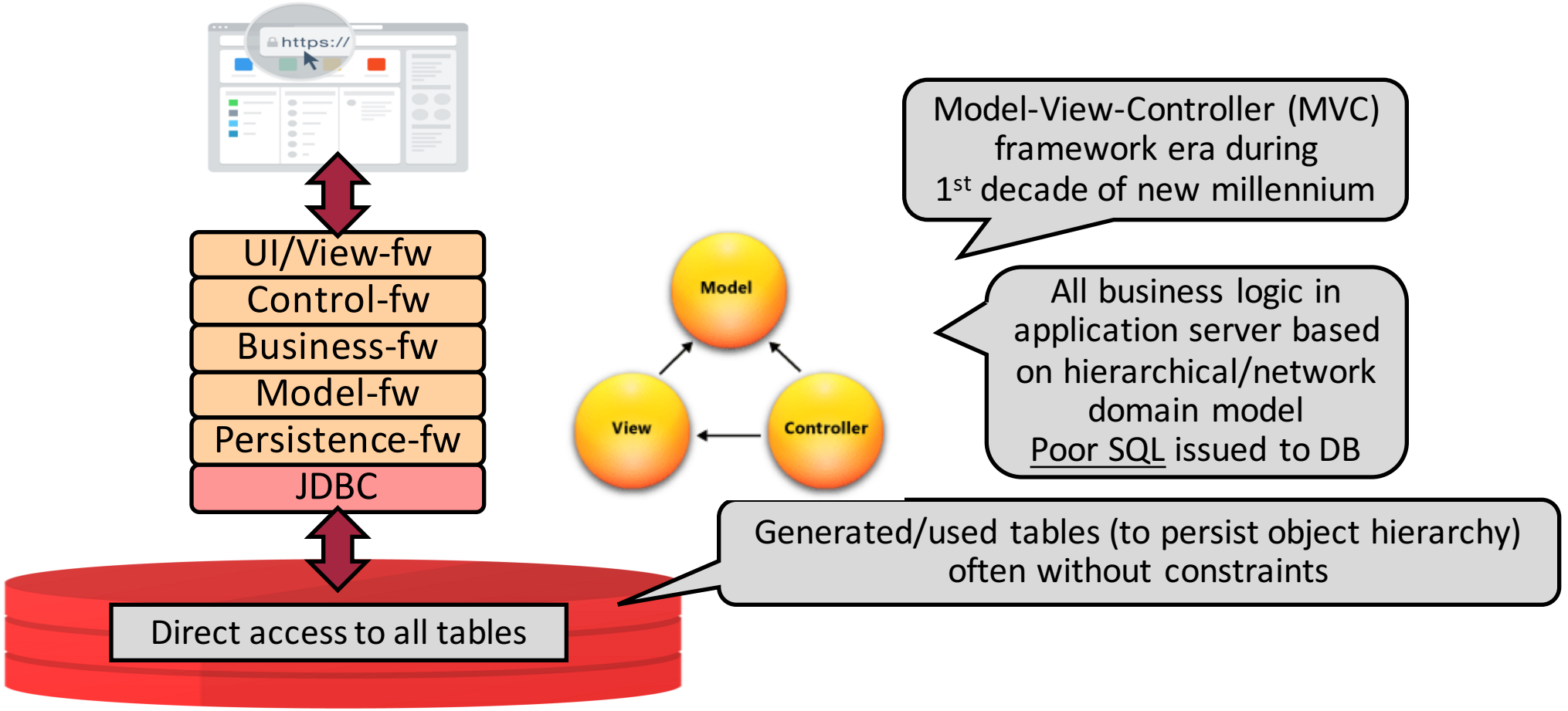
Where Were We at End of 1990's?

- Applications capitalized on database being a processing engine



What Has Happened Since?

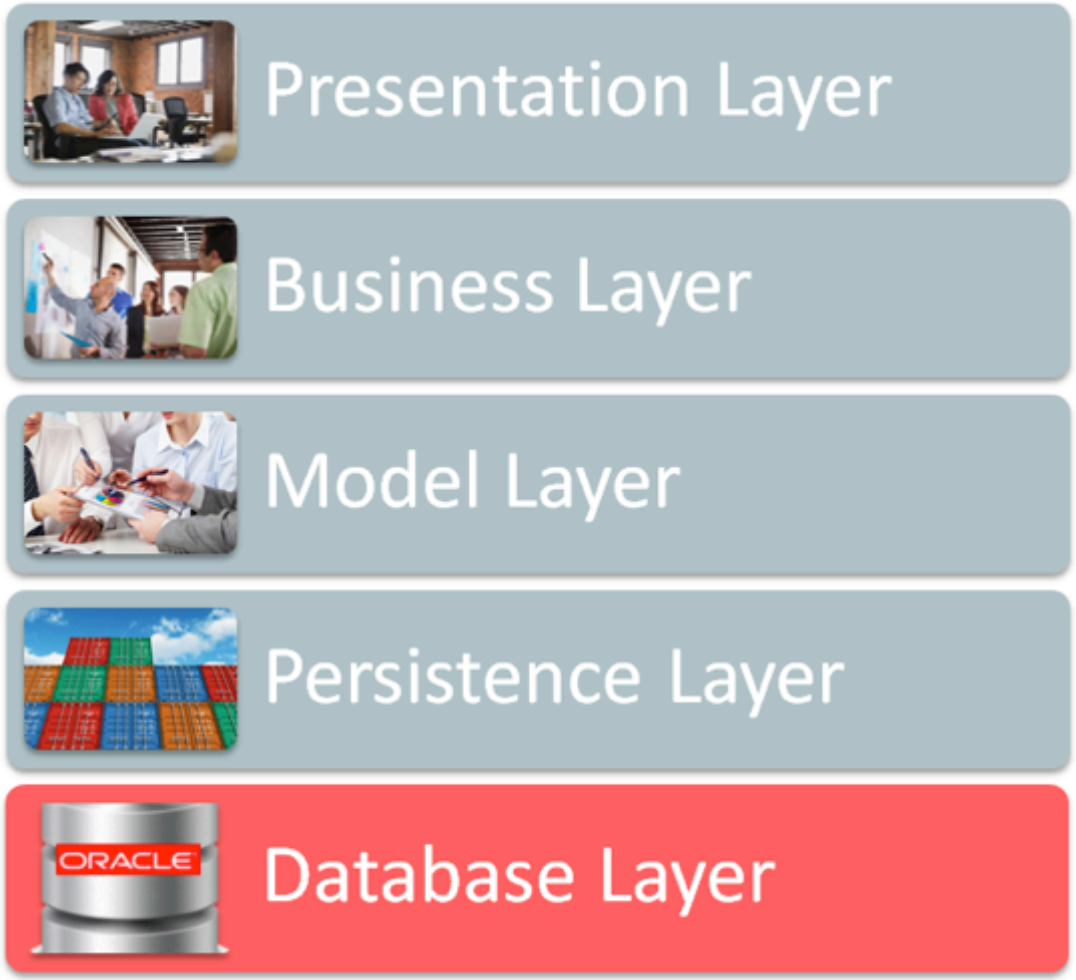
- Database to only fulfill persistence layer role (bit bucket)



What is a Layered Software Architecture?

- Relatively common architecture
- "n-Tier" architecture
- Standard for most Java EE applications
- Widely used by architects, designers, developers

What is a Layered Software Architecture?

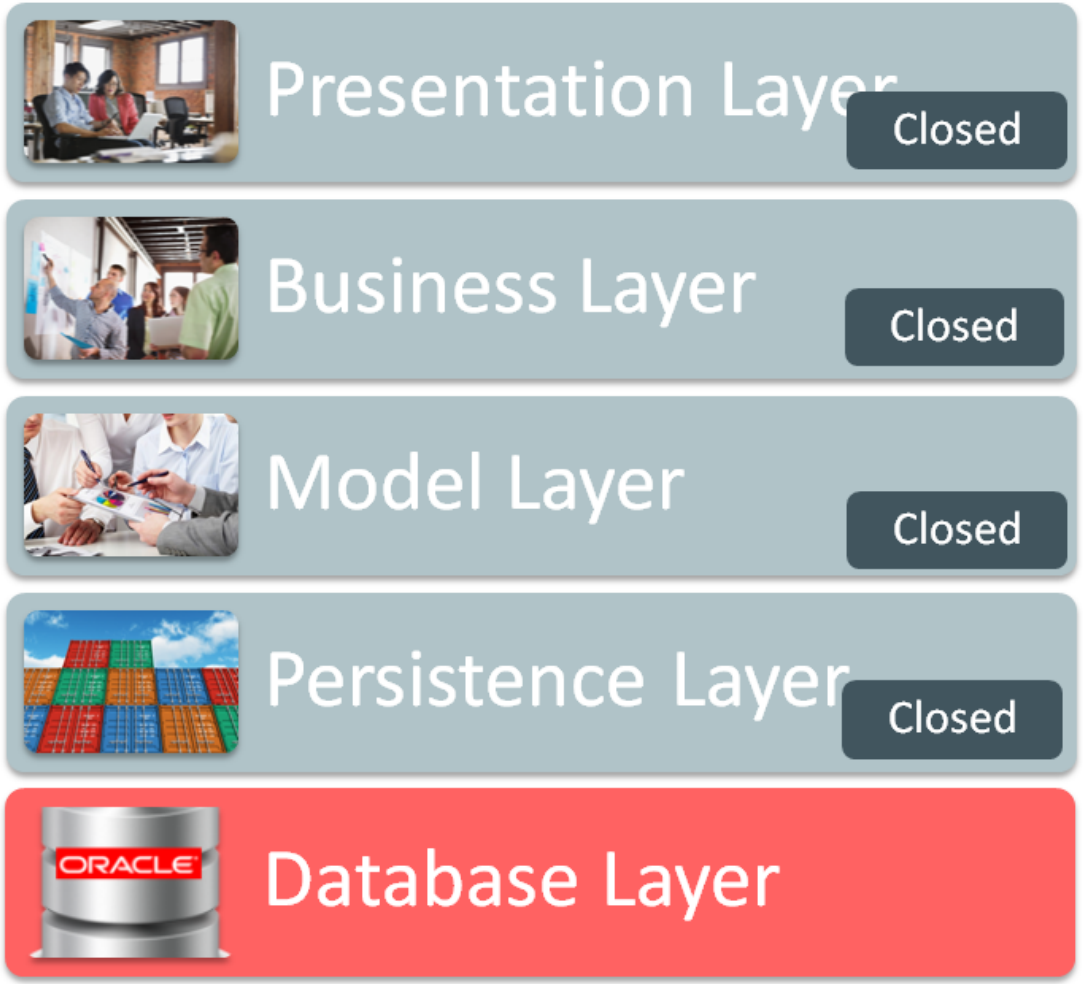


- Organized into horizontal layers
- Each layer performs specific role
- Most consist of 4-5 major layers
- Layers generally include presentation, business logic, model, persistence, and database

ORACLE

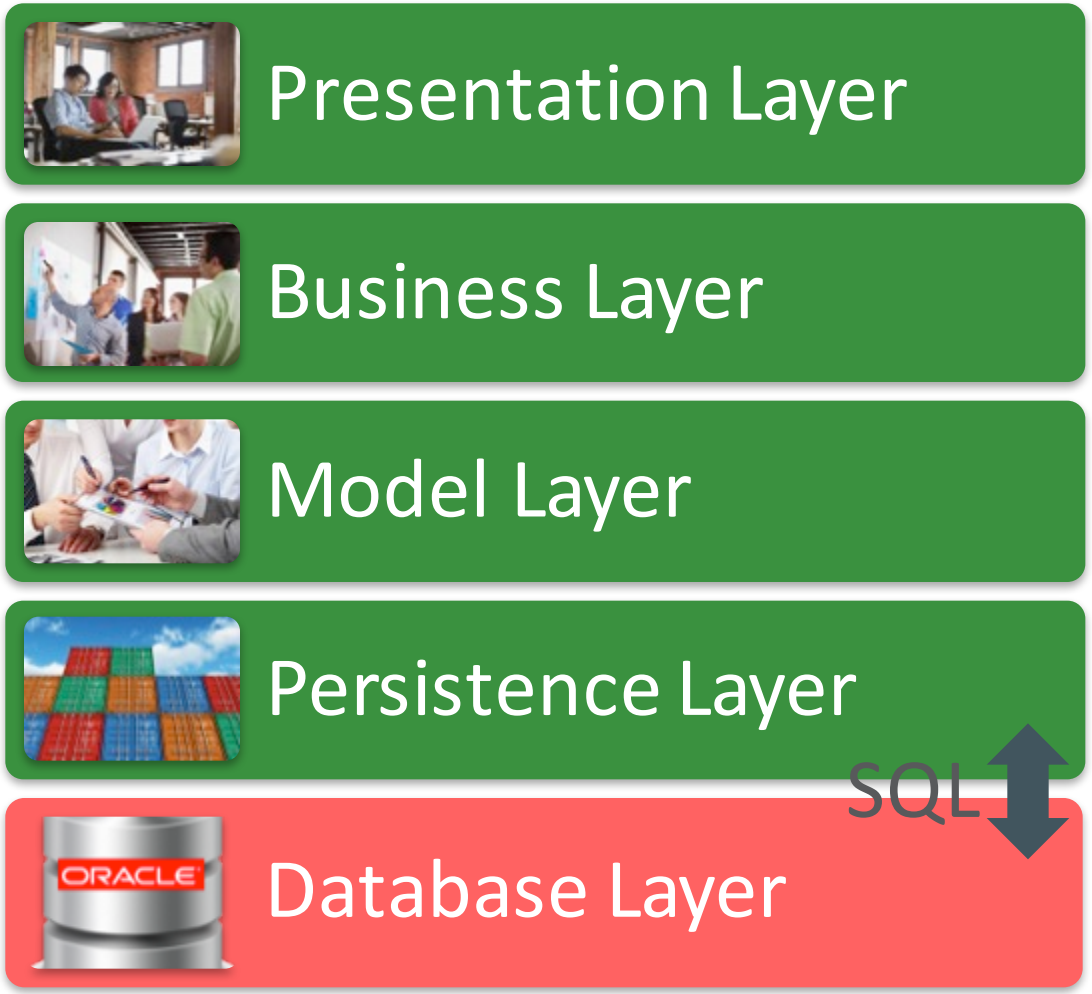
ORACLE
REAL-WORLD PERFORMANCE

Why a Layered Software Architecture?



- Layers are typically *closed* and expose API's for invocation
- Each layer must go through layer API's directly below it; enforces isolation
- Isolation enables separation of concerns and layer independence
- Makes it easy to re-use, build, and (*theoretically*) test and maintain application

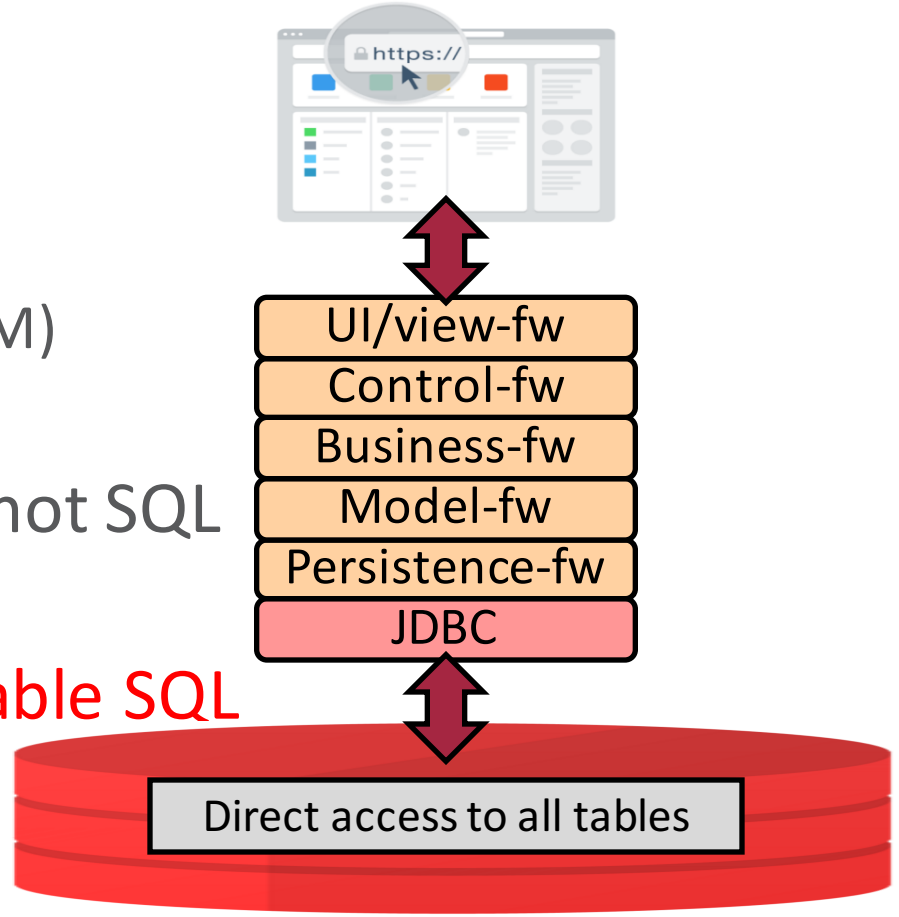
Implications of a Layered Software Architecture



- All application logic built in layers outside database
- SQL is hidden for/from developers
- Bottom layers translate GUI structures to relational data, often generating row-by-row SQL statements
- Database = (dumb) table store or "bit bucket", not a processing engine

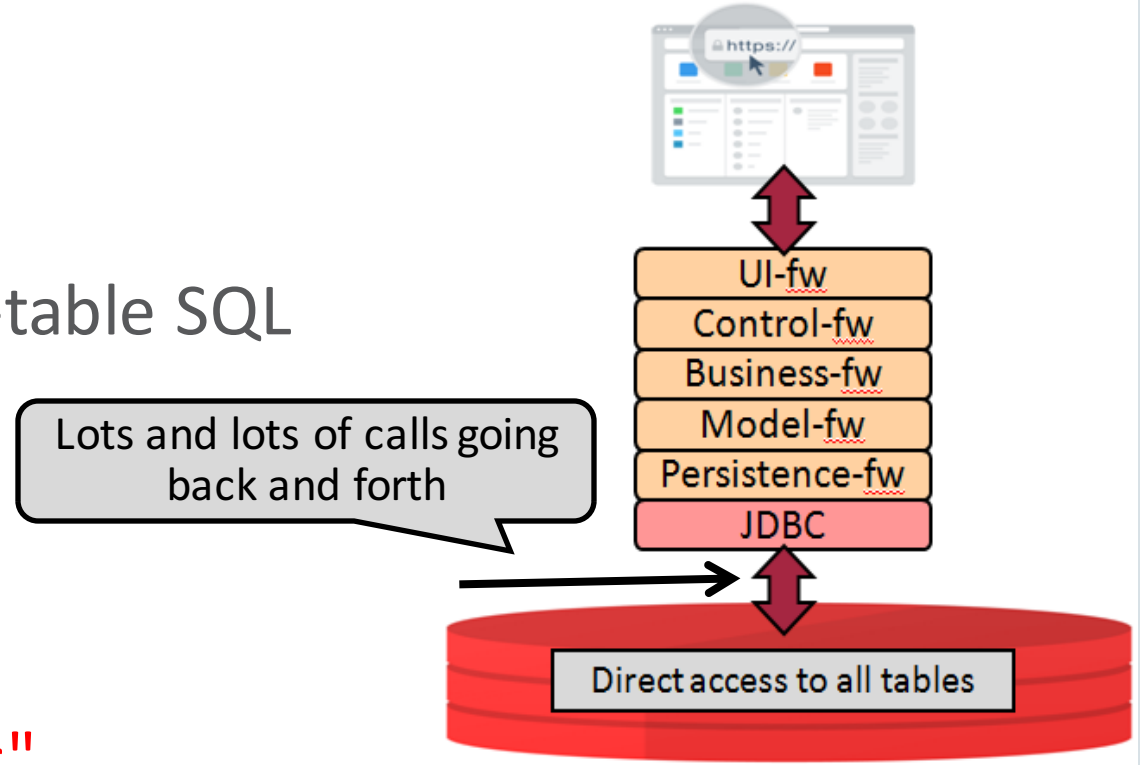
Important Points to Make (1/2)

- 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
- Common for Modern developers to know Java, not SQL
- ORM's produce single-row (or column), single-table SQL
  - In contrast to rich-SQL



Important Points to Make (2/2)

- Doing everything with single-row, single-table SQL results in "chatty" applications
  - Especially for batch processes
  - Think: 50K-200K DB-calls/second

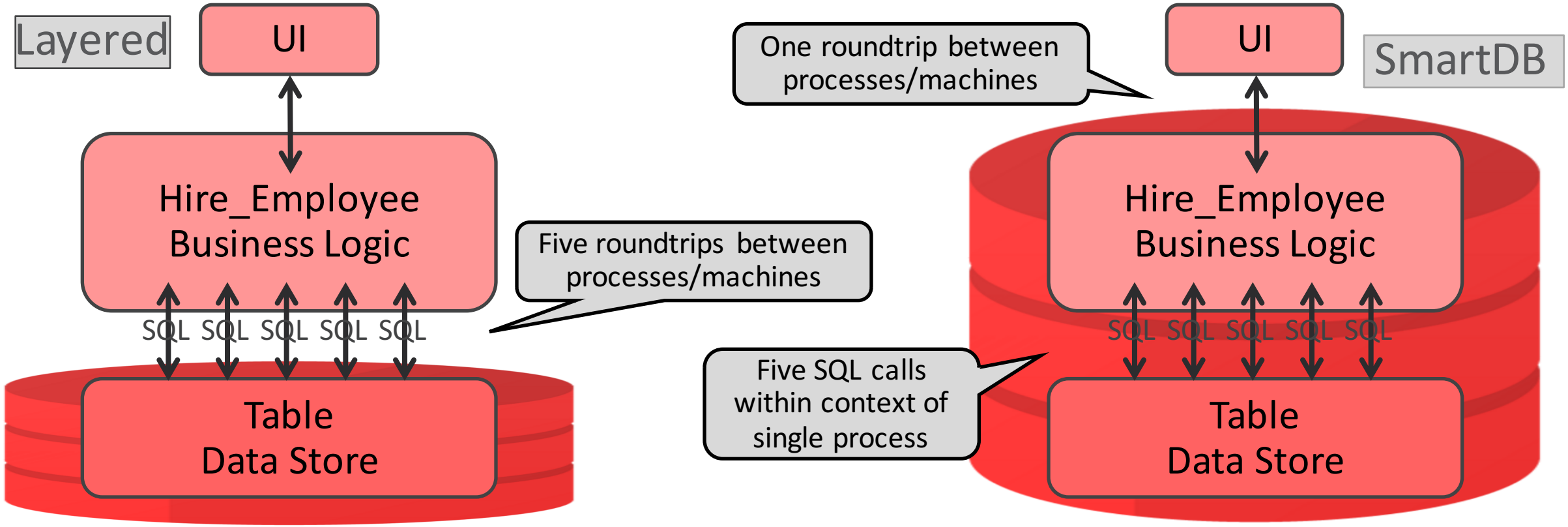


- In '90s we referred to this as "roundtrips"
  - Roundtrips were bad (for performance) then, and still are today
    - Bad then because of network-latency (now: because of CPU required)
  - Oracle7, with stored PL/SQL, helped us mitigate this
  - By moving business logic into database

I'll come back to this shortly

“Chatty” Applications

- Moving business logic into stored PL/SQL, reduces #roundtrips



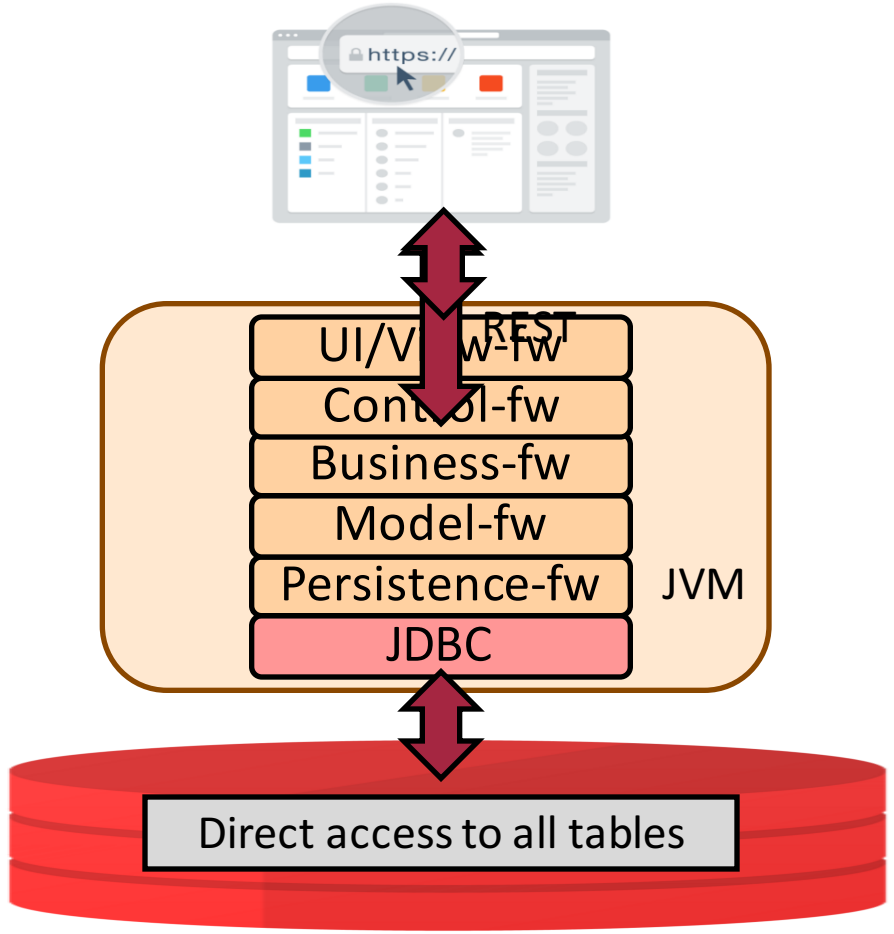
New Paradigm Shift Happening: Java → JavaScript

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

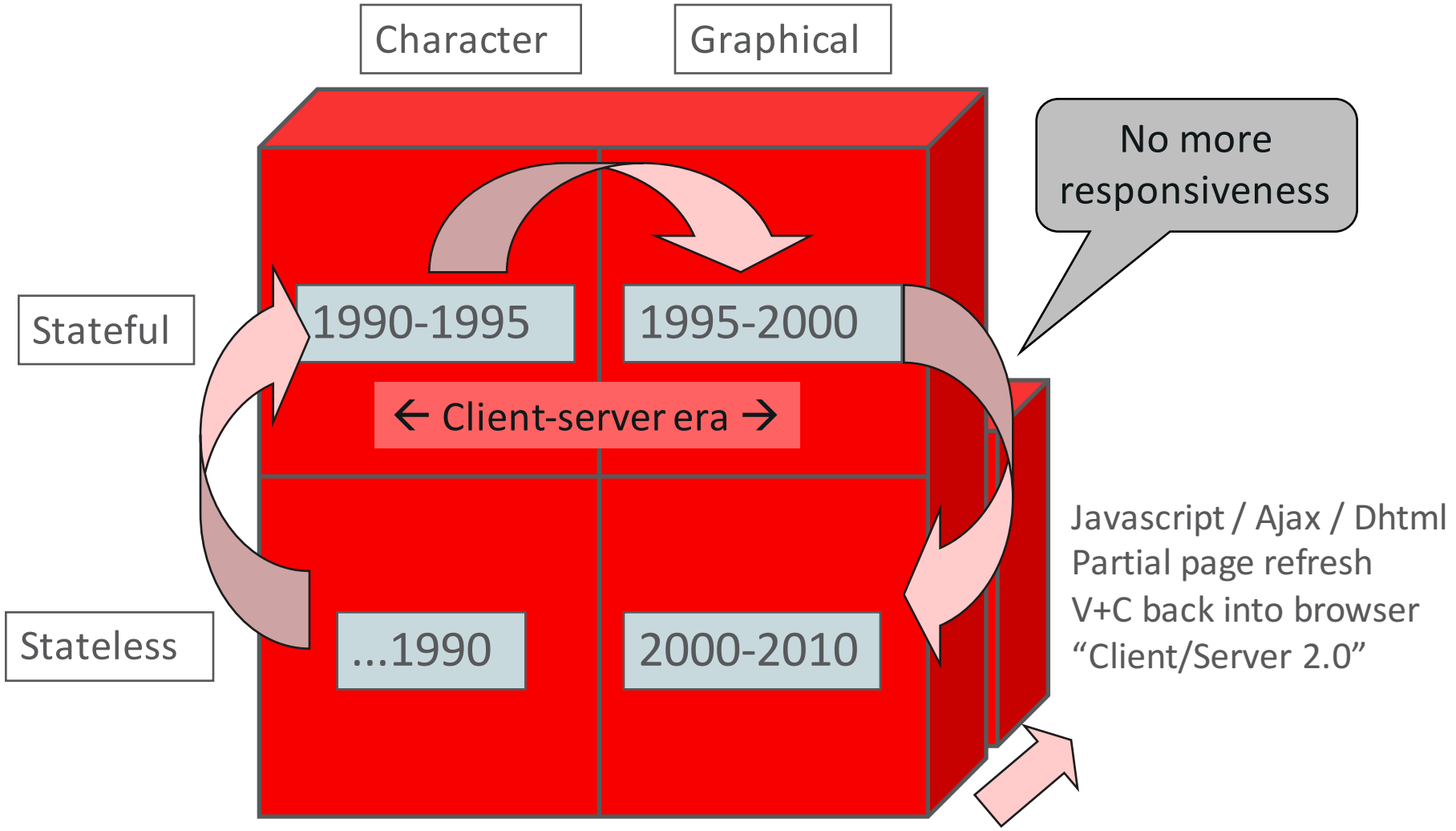
- New architecture is arising:
  - Browser-side JavaScript (V+C)
  - Server-side JavaScript (M)
  - REST to glue it together

  - Database *still* as persistence layer

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



We've Now Truly Gone Full-Circle…

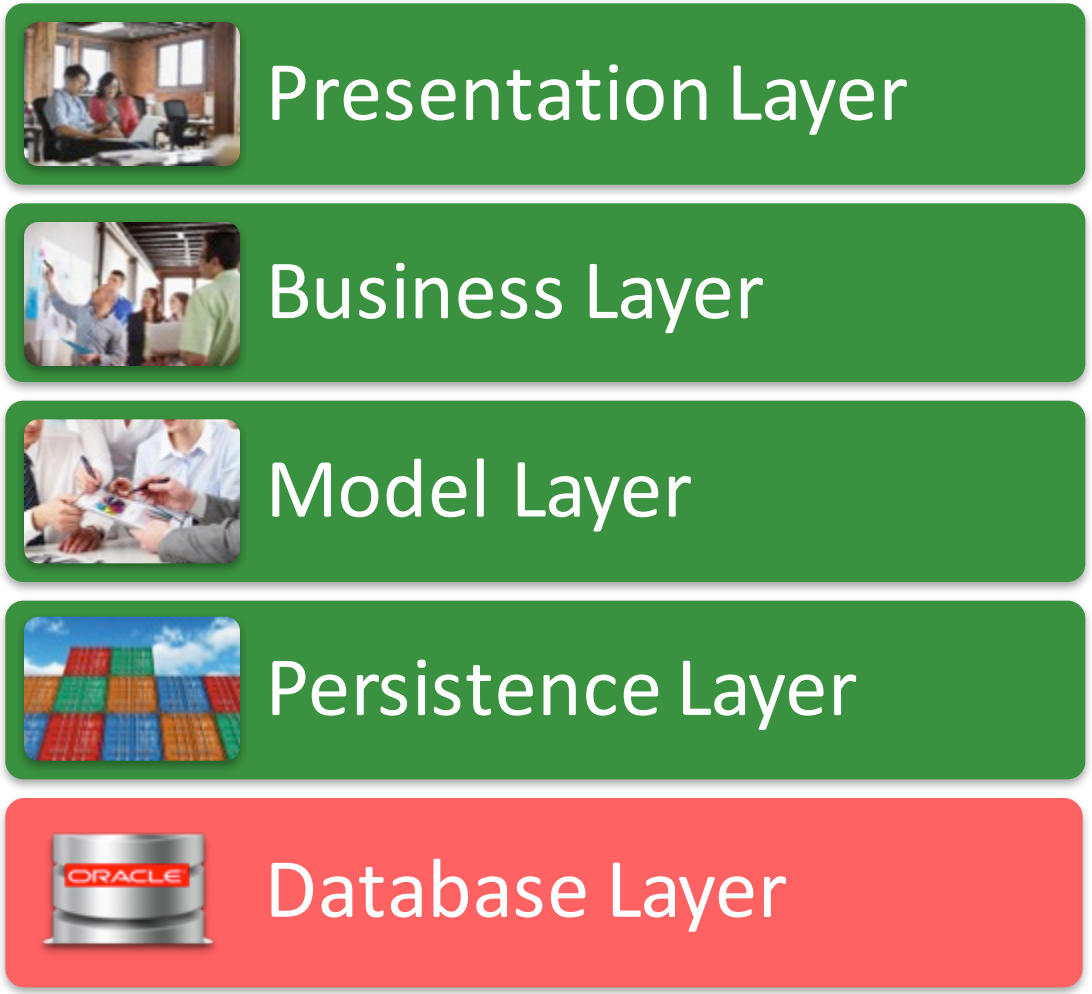


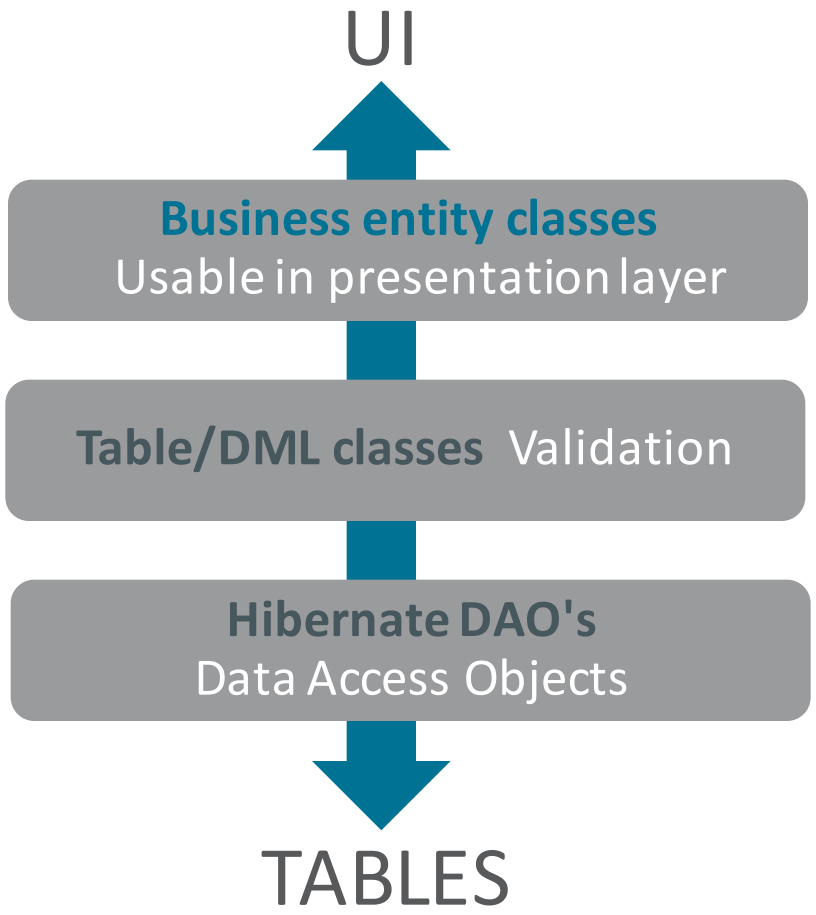
Real World Example

Development Framework Decisions

There are a number of layered development frameworks available

Layered Architecture with Hibernate





ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Layered Architecture with Oracle ADF

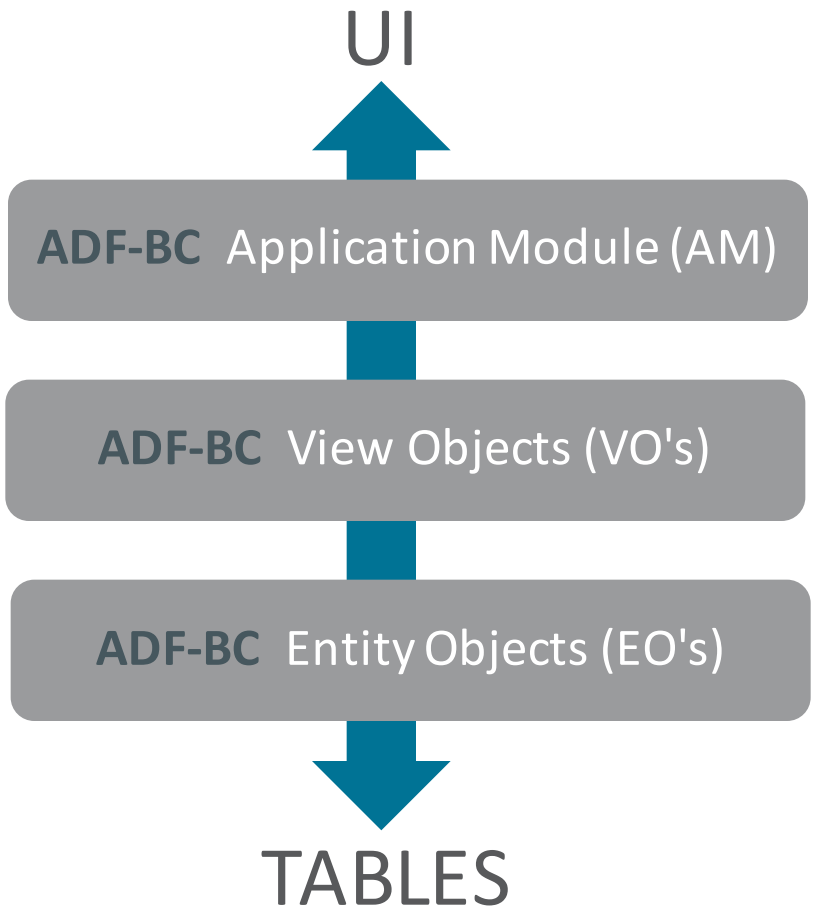

Presentation Layer


Business Layer


Model Layer


Persistence Layer

Database Layer



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Issues with Layered Architectures

1. Stability of technology stack
2. Development and maintenance cost
3. Performance and scalability

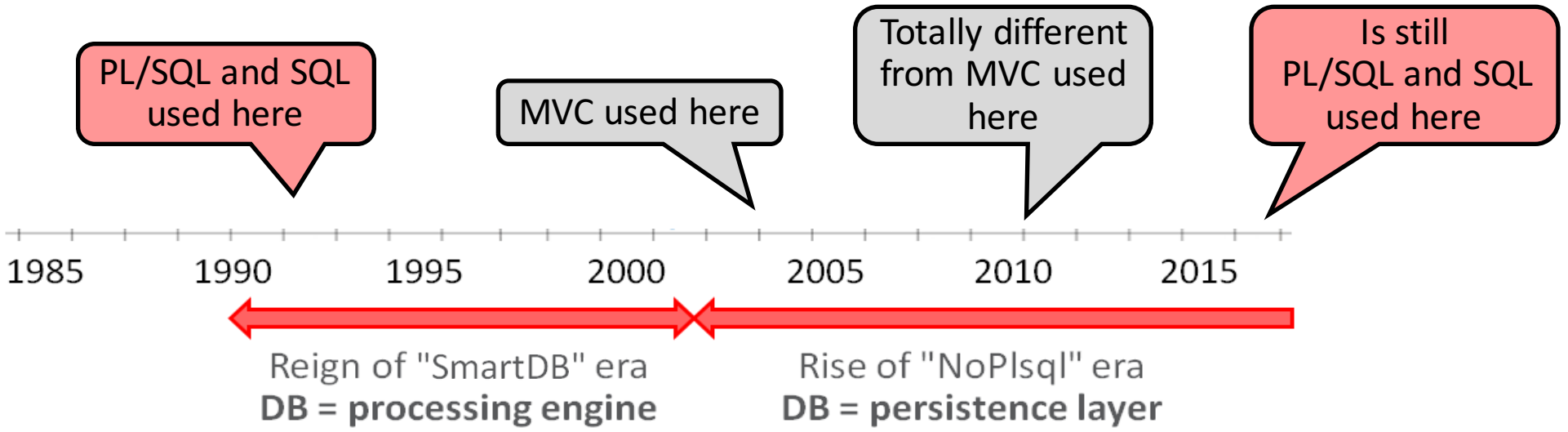
Reminder…

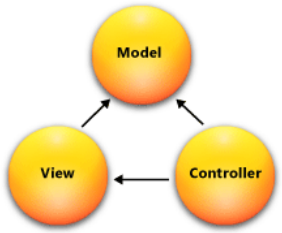
- OLTP: transactional enterprise applications
  - Data store as foundation, for which we use Oracle's RDBMS
  - Much/complex CRUD functionality on top

This is the "stable" factor" over the years - everything else on top of it, has not been…

Stability of Technology Stack – **Java MVC**

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





Google Web Toolkit®

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Stability of Technology Stack – **JavaScript MVC**

MVC used here

Is totally different from MVC used here

JavaScript used here

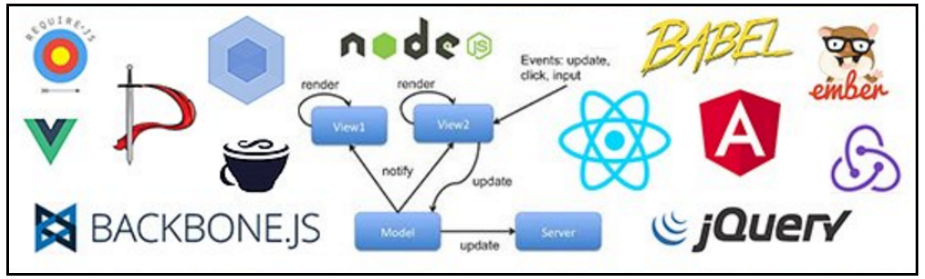
Is totally different from JavaScript used here

1985 1990 1995 2000 2005 2010 2015 2020

Reign of "SmartDB" era
DB = processing engine

Rise of "NoPlsql" era
DB = persistence layer

Rise of JavaScript everywhere



Technology volatility currently worse in emerging new world

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

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 → it'll be easier/cheaper to deal with the volatility

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

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

Issue 2: Development and Maintenance Costs

- Issue is multifaceted
  - Layered technology stacks are complex
  - Wheels are reinvented
  - Is OO a good fit for business logic of database applications?

Hibernate…

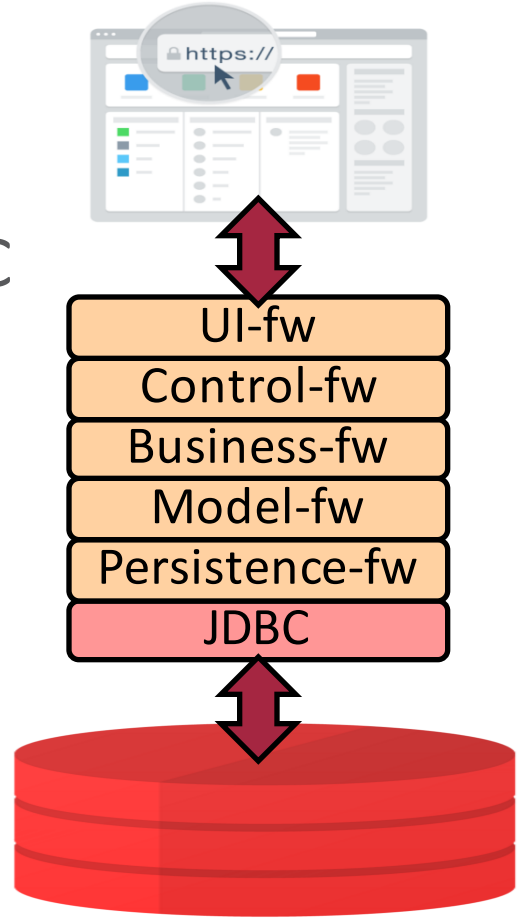
- “I had to learn *Hibernate* architecture, configuration, logging, naming strategies, tuplizers, entity name resolvers, enhanced identifier generators, identifier generator optimization, union-subclasses, XDoclet markup, bidirectional associations with indexed collections, ternary associations, idbag, 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.”

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

MVC Technology Stacks Are Complex

- Witnessed many projects: both #SmartDB and Layered/MVC
- Disproportional large amount of time spent on getting frameworks known and then set up and work all together

  Many discussions on "how to do this?", "how to do that?"
- (Counterargument?): Modern developers taught to ”plug”
- Much time is spent on "plumbing" code
  - Code that doesn’t add value to the business

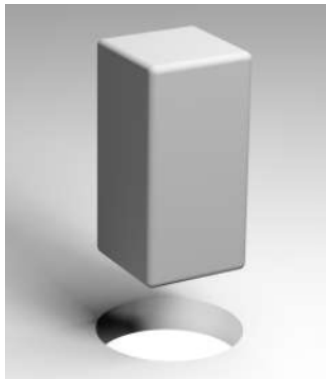


- SmartDB developers proportionally spend more time on what end-users care for, on what is unique to application: its business logic

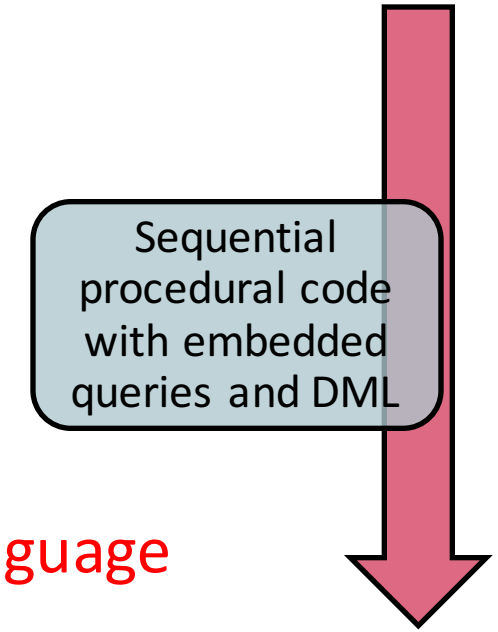
Wheels Are Reinvented

- 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 or declaratively via SQL

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
  Natural fit = implementation via some if-then-else-loop language
  Preferably one that does SQL really well: think PL/SQL



Issue 3: Performance and Scalability

- "Database is always bottleneck", so here's the Layered/ORM/MVC promise:
  - Get data from DB once into mid-tier caches
  - Then re-use many times in business logic running in horizontally scalable mid-tier servers
  - Write data back to DB once
- Always major argument to reject SmartDB approach

  “SmartDB approach (running BL in DB) will saturate database real quick“
  “It won’t scale”

Performance and Scalability

- However in real-world:
  - Hardly ever see these cached data "re-uses“ → these applications are *always* chatty
  - Data read + manipulated once, then written back, and not used again while in cache
  - Where's the advantage then?
- Also:
  - Instantiating objects for rows, takes a lot of memory and CPU
    Data is always cached in multiple layers: JDBC, ORM, model framework, Business Logic modules
  - Cached data volumes become so big that caches need to flush data pre-maturely
- Also, part II:
  - Where is my time being spent? More on this later …
- Btw: data is already cached very efficiently at database tier…

Demos and Technical Stuff

1. Introduction
2. Layered Architecture: History, Landscape, and Issues
3. Demos and Technical Stuff
4. Big Picture
5. Concluding Thoughts

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Figure out

Summary Here

- Original story at Oracle Learning Library channel on youtube

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

Search: "toon koppelaars"

Our (First) Experiment

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

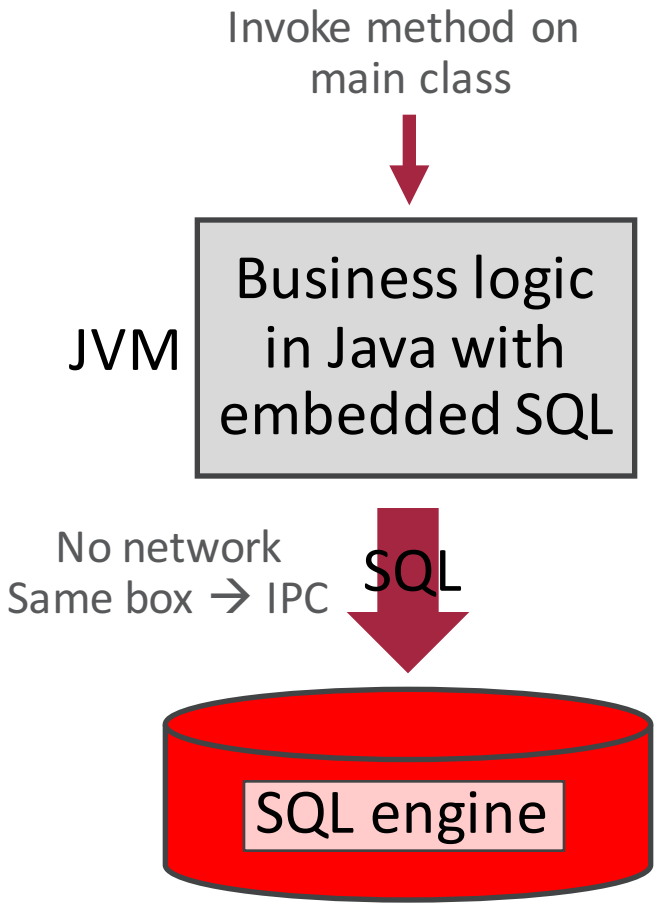
  Both built using row-by-row simple/poor SQL pattern

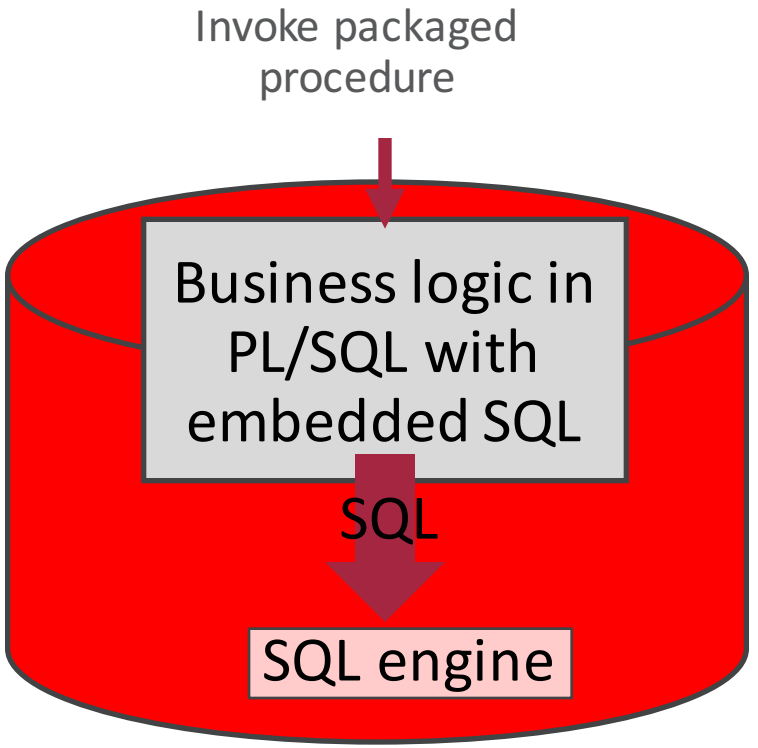
- Straight-forward business logic: if-then-else, looping

- 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



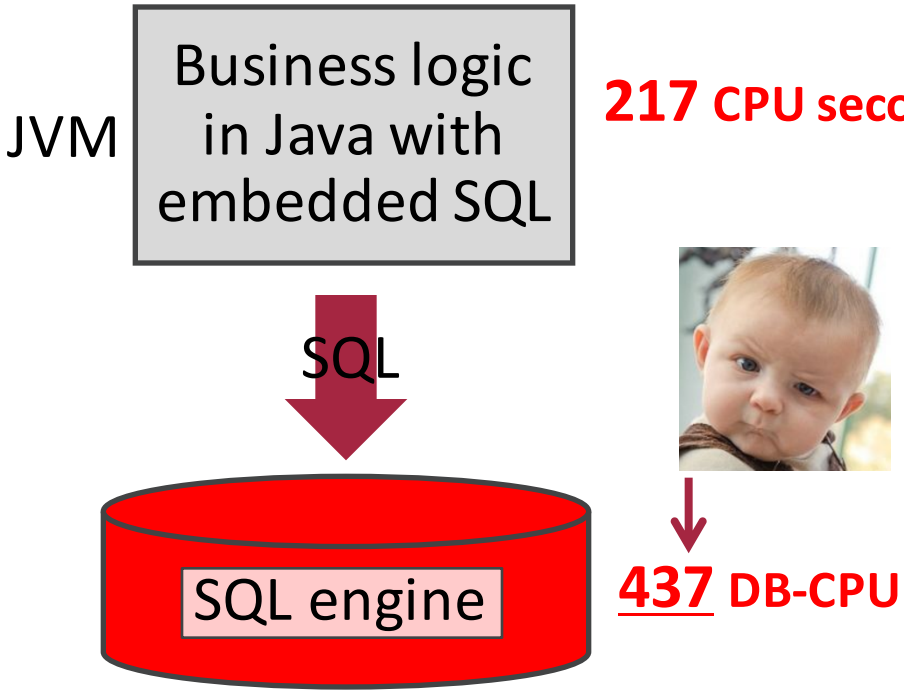


ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Java/JDBC versus PL/SQL

Elapsed-time: **11 minutes**

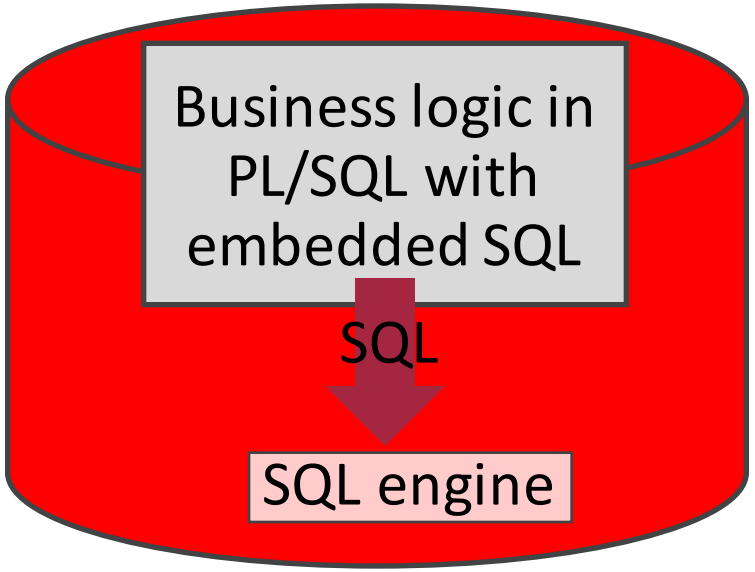


217 CPU seconds

204 DB-CPU seconds

437 DB-CPU seconds

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



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

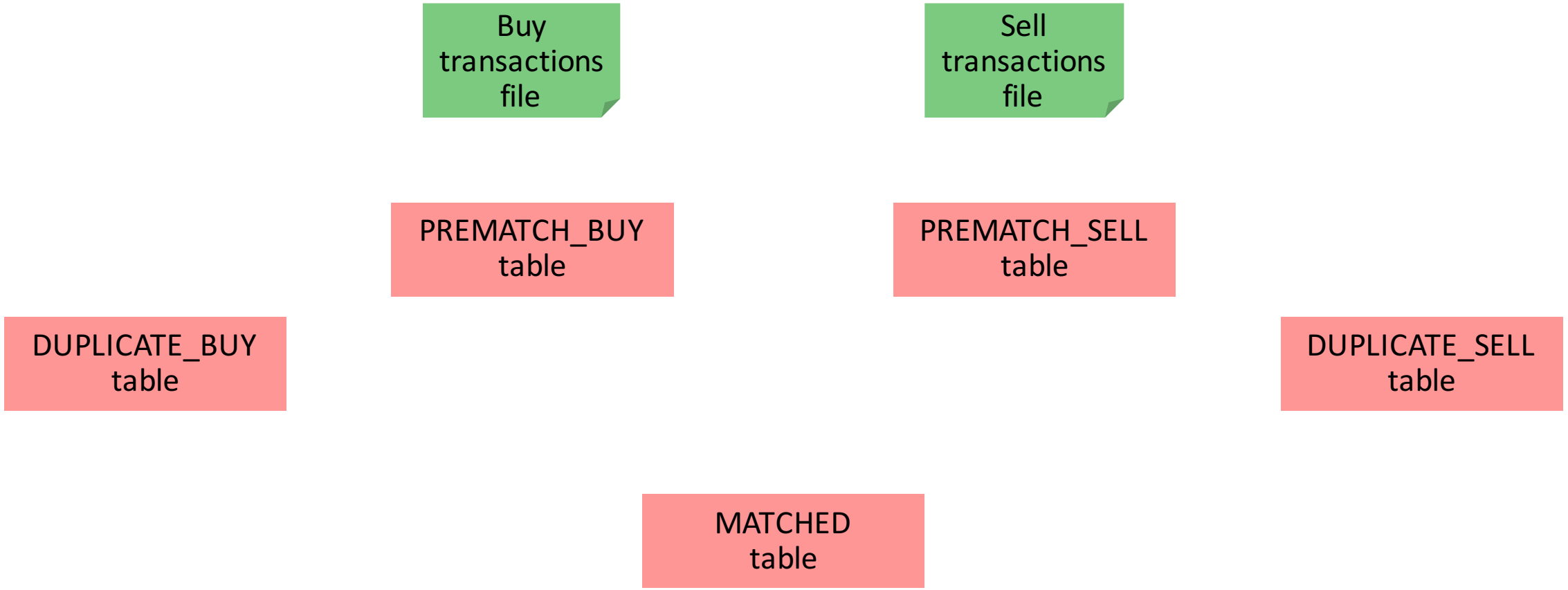
Real World Example

Daily Batch Program for Tax Matching: Briefing

- VAT Clearing/Matching
  - Company A sells to Company B
  - Company B buys from Company A
  - Each report VAT on purchased goods and sold goods; one business can claim the VAT
- Buy and Sell records must be matched
- Batches received daily in sets of 2: buy records and sell records
- Possible exceptions:
  - Duplicates
  - Out of sync records; i.e., Buy and Sell records in different batches

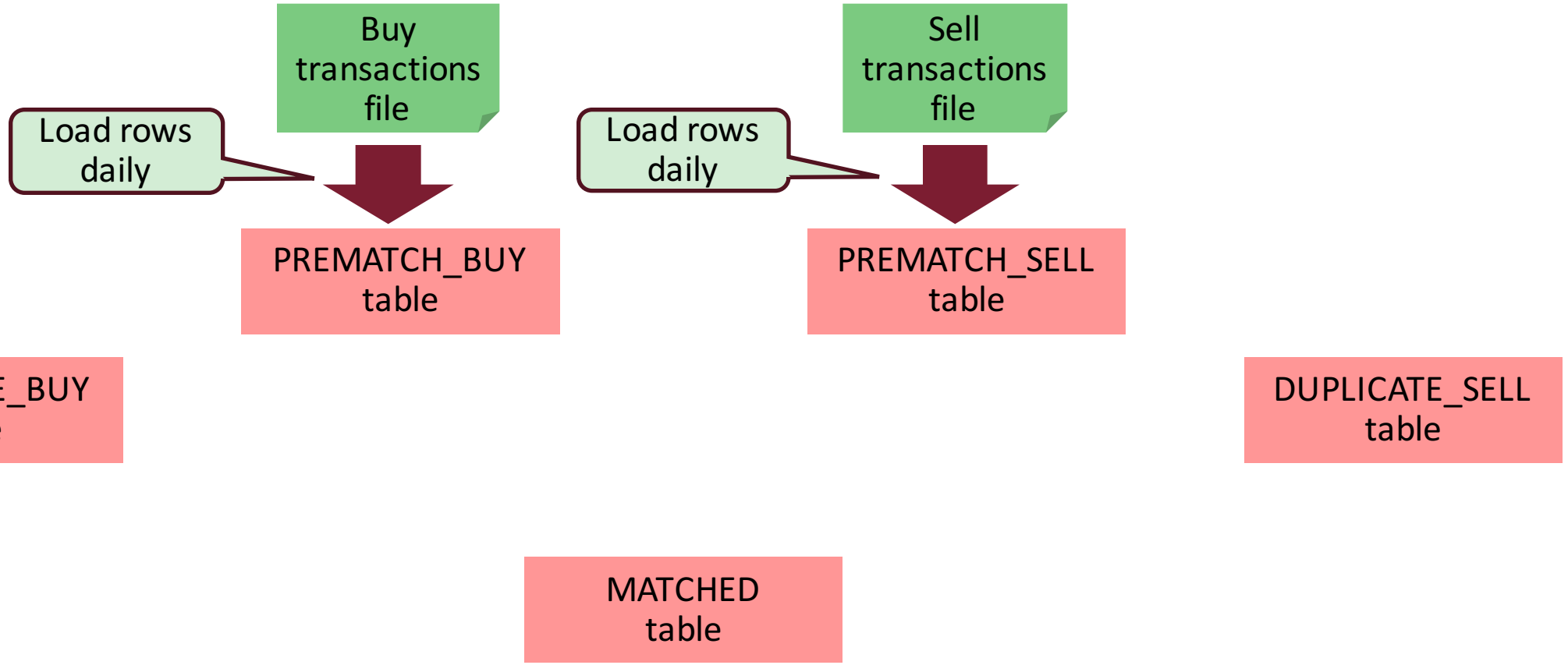
Real World Example

Involved Objects



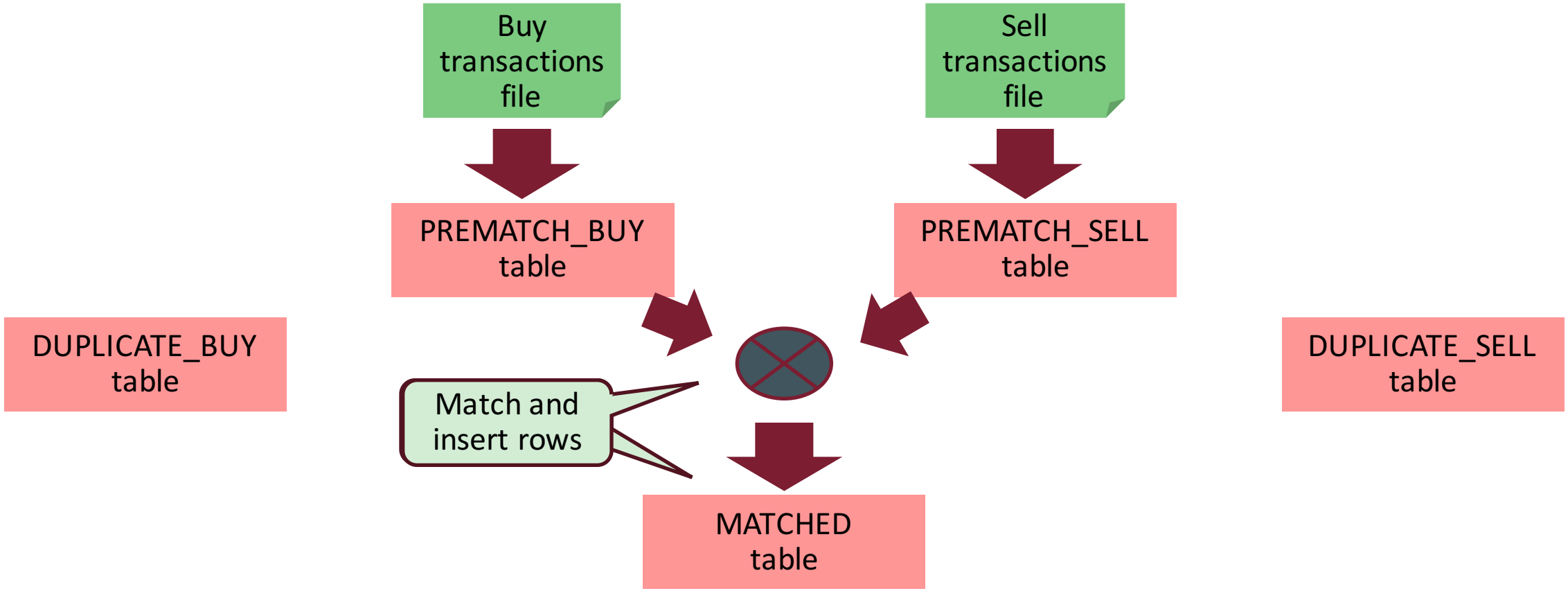
Real World Example

Data Flow



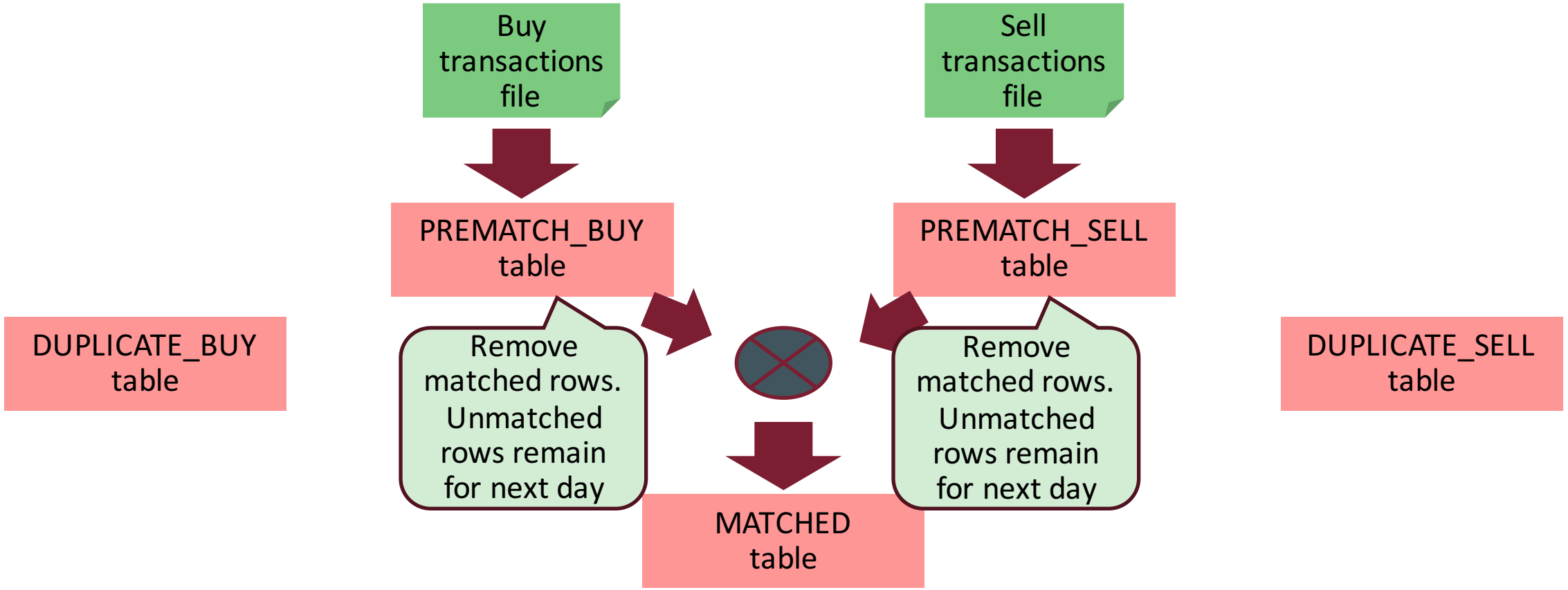
Real World Example

Data Flow



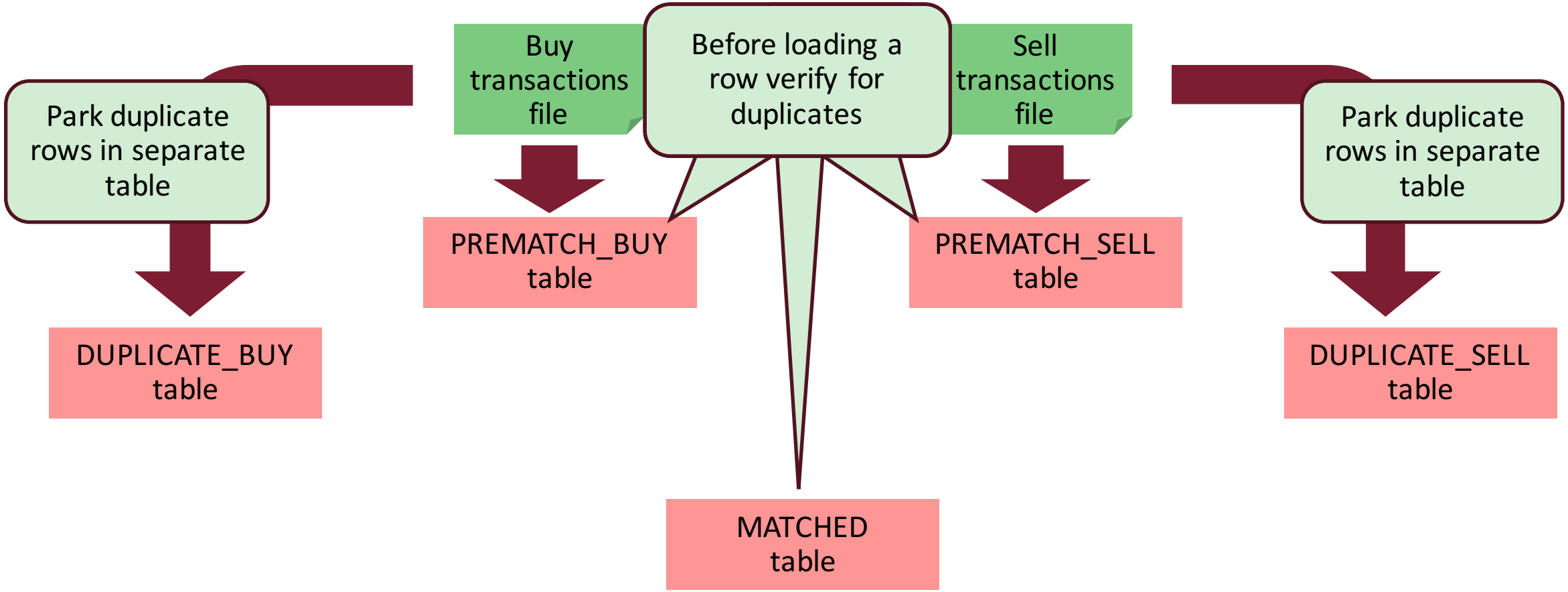
Real World Example

Data Flow



Real World Example

Additional Business Logic: Discard Duplicates



Real World Example

Load Profile

Does a bit of everything:

- Inserts into five tables
- Performs indexed lookups for de-duplication and matching
- Deletes from PREMATCH tables
- Index maintenance
- Some business logic (if-then-else) to de-dupe and match

- **We already have a Web UI that does VAT matching for single buy/sell record**
- **We'll reuse its business/model/persistence layers**

DEMO

Tax Demo – Run #1

Config	Threads	File 1 Seconds	File 2 Seconds	File 3 Seconds	File 4 Seconds	File 5 Seconds	Total Seconds
ADF	1	1,674	2,081	1,561	1,111	2,131	8,558
ADF	4	460	562	425	302	579	2,328
ADF	16	158	192	145	109	198	802
PL/SQL	1	337	421	315	224	430	1,727

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Tax Demo – Run #1

Load Profile

ADF 1 Thread

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	0.4	0.0	0.00	0.00
DB CPU(s):	0.4	0.0	0.00	0.00

ADF job spends less time in the database

Load Profile

PL/SQL 1 Thread

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	1.0	0.0	0.00	0.23
DB CPU(s):	1.0	0.0	0.00	0.23

Tax Demo – Run #1 (Continued)

Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	5.7	0.1	0.00	0.00
DB CPU(s):	5.6	0.1	0.00	0.00

ADF 16 threads

With 16 threads, our ADF job is only active in the database 5.7 seconds per second

Why did we show PL/SQL again?

- Our layered (ADF) application forced us down a row-by-row path
- We can easily do row-by-row processing in PL/SQL

Tax Demo – Run #1

Elapsed Time (s)	Executions	Elapsed Time per Exec (s)	%Total	%CPU	%IO			SQL Text
907.39	5	181.48	50.64	99.05	0.05	8		BEGIN matching_thread.match(1-...
409.09	5	81.82	22.83	99.14	0.29	3		BEGIN matching_thread.prematch...
408.78	5	81.76	22.81	98.97	0.46	b		BEGIN matching_thread.prematch...
193.61	3,777,832	0.00	10.81	99.36	0.06	44x	lauf	SQL*Plus DELETE FROM PREMATCH_BUY WHERE...
187.07	3,777,832	0.00	10.44	99.81	0.0		dpcsdykqq7	SQL*Plus DELETE FROM PREMATCH_SELL WHER...
165.56	3,777,832	0.00	9.24	98.5	0.16	8p0wp2w01ns7p		SQL*Plus INSERT INTO MATCHED (CODE , S...
148.01	3,777,961	0.00	8.26	99.35	0.08	7n0fbc5grpk9t		SQL*Plus INSERT INTO PREMATCH_BUY (COD...
147.45	3,777,839	0.00	8.23	99.28	0.61	4zt60chx4my3n		SQL*Plus INSERT INTO PREMATCH_SELL (CO...
141.24	7,555,884	0.00	7.88	100.95	0.04	8d045khaf6y24		SQL*Plus SELECT COUNT(*) FROM MATCHED M...
89.55	3,778,257	0.00	5.00	100.82	0.00	d3traqc5vg8xv		SQL*Plus SELECT X2.*, X2.ROWID FROM PRE...
53.39	0		2.98	99.88	0.00	dzzzrbrj43and		JDBC Thin Client BEGIN :1 := mon_db.startup(int...
38.89	1,765	0.02	2.17	99.53	0.00	a4mcjs95j4gnn		JDBC Thin Client SELECT SUM(VALUE) FROM V$SYSST...
25.40	5	5.08	1.42	96.72	3.82	229axchrcvjuz		SQL*Plus SELECT * FROM V_EXT_SELL WHERE...
25.19	5	5.04	1.41	96.37	3.94	ak9gptbkj80xk		SQL*Plus SELECT * FROM V_EXT_BUY WHERE ...

Will SQL tuning help?

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Tax Demo – Run #1

This is what row-by-row processing looks like

Executions	Rows Processed	Rows per Exec	Elapsed Time (s)	%CPU			le	SQL Text
7,555,884	7,555,884	1.00	141.24	100.9	0			SELECT COUNT(*) FROM MATCHED M...
3,778,257	3,777,832	1.00	89.55	100.8	0	[illegible]	SQL*Plus	SELECT X2.*, X2.ROWID FROM PRE...
3,777,961	3,777,958	1.00	148.01	99.4		7n0fbc5grpk9t	SQL*Plus	INSERT INTO PREMATCH_BUY (COD...
3,777,839	3,777,833	1.00	147.45	[illegible]	.6	4zt60chx4my3n	SQL*Plus	INSERT INTO PREMATCH_SELL (CO...
3,777,832	3,777,832	1.00	[illegible]	99.8	0	071upcsdykqq7	SQL*Plus	DELETE FROM PREMATCH_SELL WHER...
3,777,832	3,777,832	1.00	193.61	99.4	.1	44xutmzsrnauf	SQL*Plus	DELETE FROM PREMATCH_BUY WHERE...
3,777,832	3,777,832	1.00	165.56	98.5	.2	8p0wp2w01ns7p	SQL*Plus	INSERT INTO MATCHED (CODE , S...
88,550	88,550	1.00	16.25	99	0	8wps5gfjrm6su	SQL*Plus	MERGE INTO TAX1.TAX_RUN_CONTRO...
88,550	88,550	1.00	2.96	105.6	0	d7pav1dwuxk9j	SQL*Plus	SELECT RACE_STATUS FROM TAX1.T...
1,765	1,765	1.00	38.89	99.5	0	a4mcjs95j4gnn	JDBC Thin Client	SELECT SUM(VALUE) FROM V$SYSST...

Layered Arch's Promise? → Opposite Effect

- Runtime longer → Layered approach performs worse
- DB resource-usage more → Layered approach scales worse
- Layered approach uses more CPU
- Seems "performance/scalability" argument is wrong?

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

DEMO

Tax Demo – Run #2

Config	Threads	Array Size	File 1 Seconds	File 2 Seconds	File 3 Seconds	File 4 Seconds	File 5 Seconds	Total Seconds
ADF	16	1	158	192	145	109	198	802
PL/SQL	1	256	163	210	115	113	210	811
PL/SQL	16	1	37	43	31	23	44	178
PL/SQL	16	256	26	31	24	17	33	131

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

DEMO

ADF vs. PL/SQL CPU Efficiency

Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
DB CPU		4608.3		99.0	
SQL*Net message to client	38,050,060	33.3	0.00	.7	Network
log file sync	65,093	28.5	0.44	.6	Commit
external table read	24,992	27.1	1.09	.6	User I/O
cursor: pin S	16,841	18	1.07	.4	Concurrency
cell single block physical read	32,533	17.6	0.54	.4	User I/O
external table open	320	6.9	21.61	.1	User I/O
cell multiblock physical read	5,535	6	1.09	.1	User I/O
SQL*Net more data to client	81,920	5.7	0.07	.1	Network
buffer busy waits	177,977	5.6	0.03	.1	Concurrency

ADF (16 Threads)

The ADF program uses more CPU

PL/SQL (16 Threads)

Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
DB CPU		2351.2		88.2	
buffer busy waits	316,103	108.5	0.34	4.1	Concurrency
enq: HW - contention	7,169	75.4	10.51	2.8	Configuration
log file switch (checkpoint incomplete)	85	28.6	336.17	1.1	Configuration
external table read	24,992	27.9	1.12	1.0	User I/O
cursor: pin S	13,210	14	1.06	.5	Concurrency
undo segment extension	802	12.4	15.41	.5	Configuration
KSV master wait	6,916	7.2	1.03	.3	Other
control file sequential read	8,402	6.2	0.74	.2	System I/O
DFS lock handle	1,489	6	4.05	.2	Other

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

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

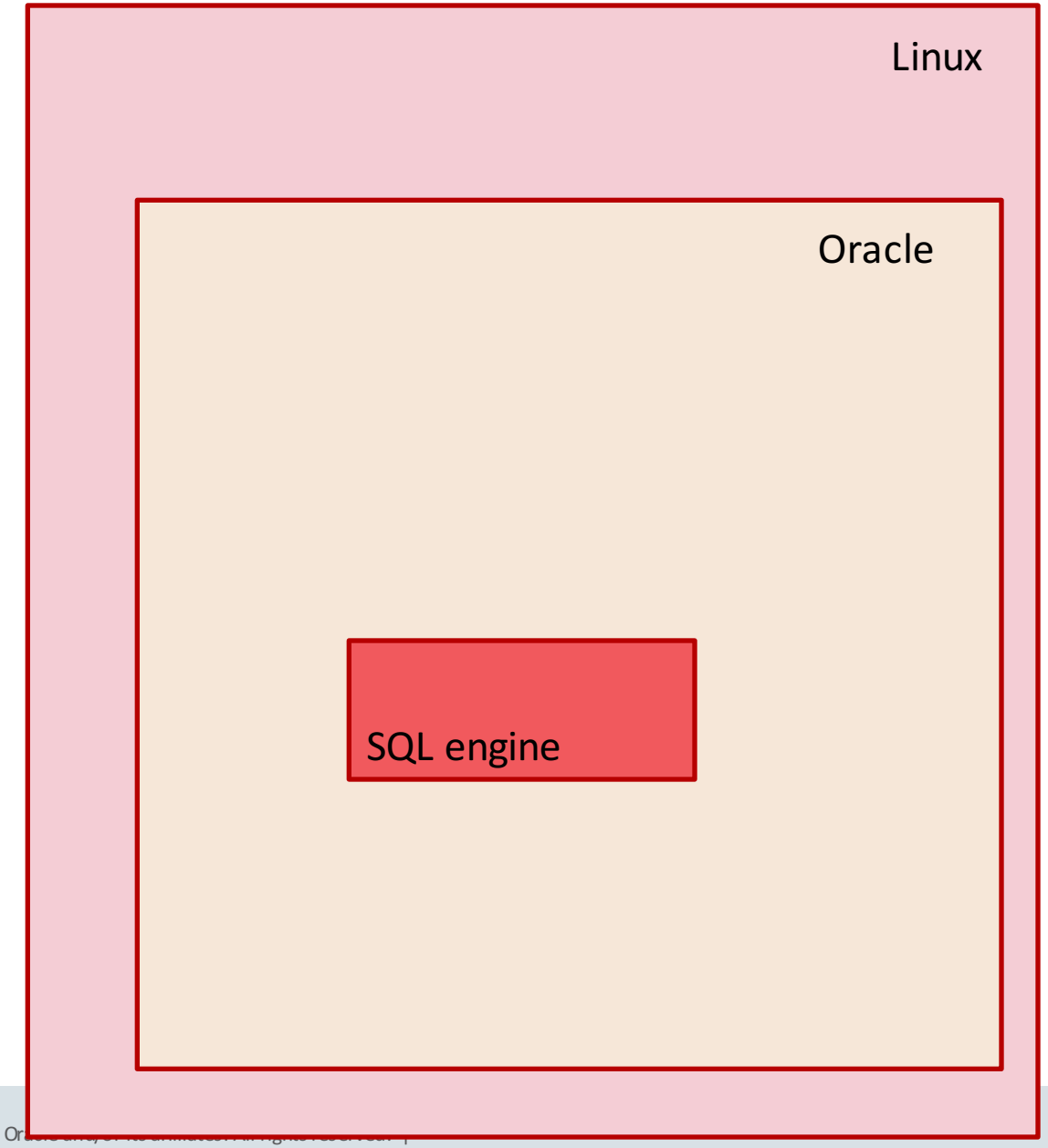
- "The Living Room" analogy
- With SmartDB:
  - PL/SQL is already in living room, which is where SQL-engine lives

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

  And apparently this is *not* for free if you execute lots of DB calls

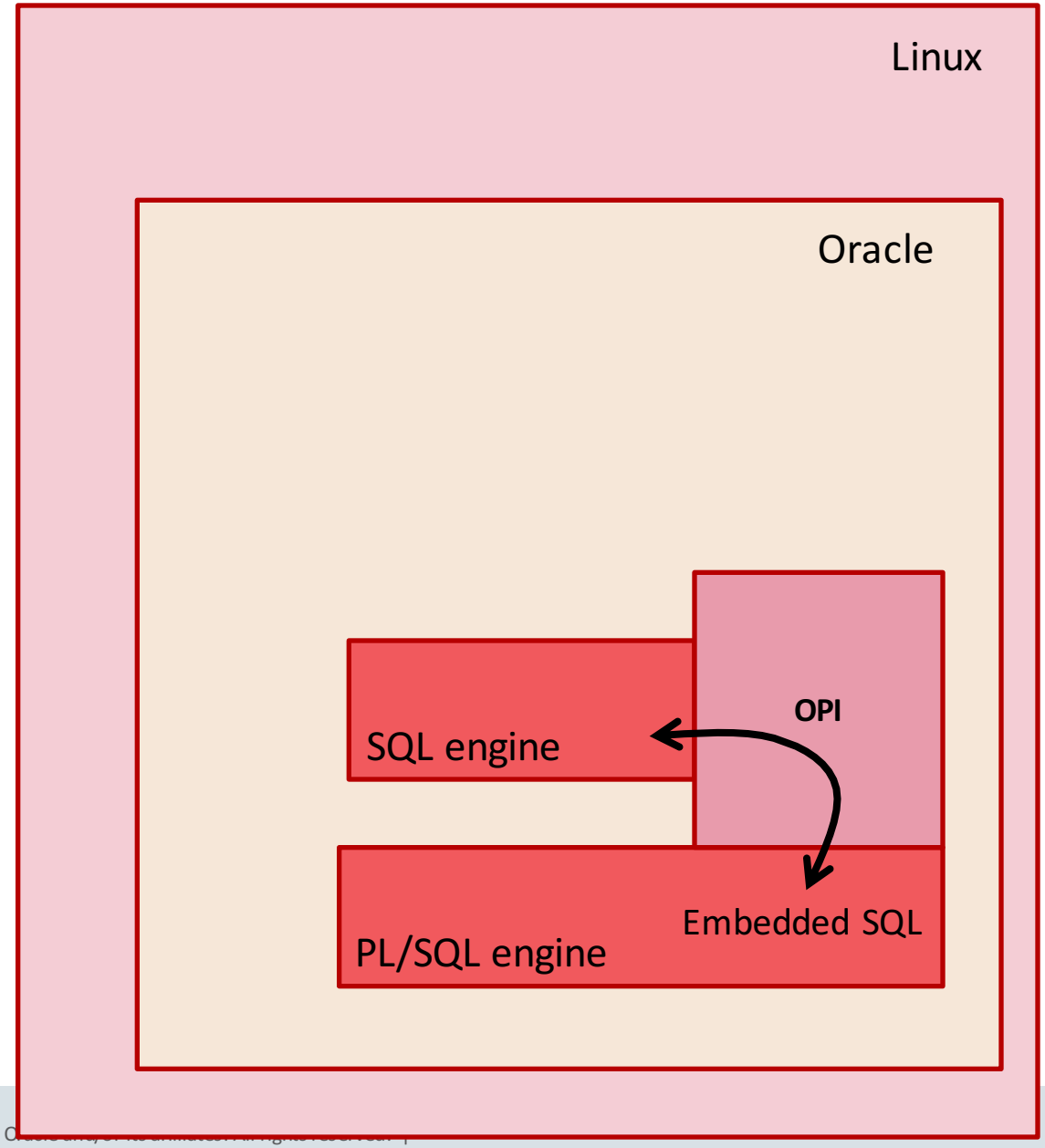
The Living Room

- SQL engine



The Living Room

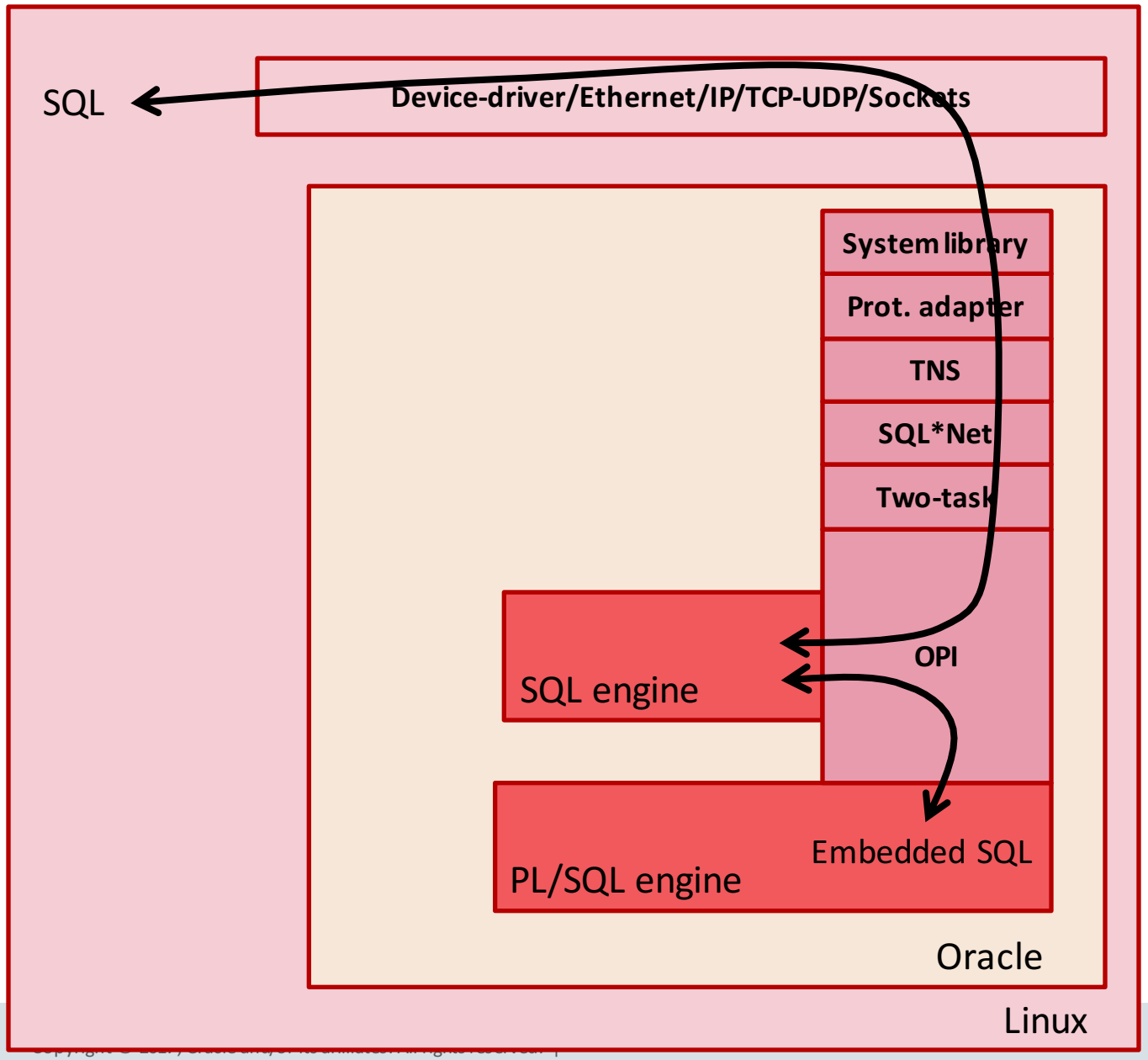
- SQL engine
  - Accessible via OPI layer
  - Oracle Program Interface
- PL/SQL directly calls OPI



The Living Room

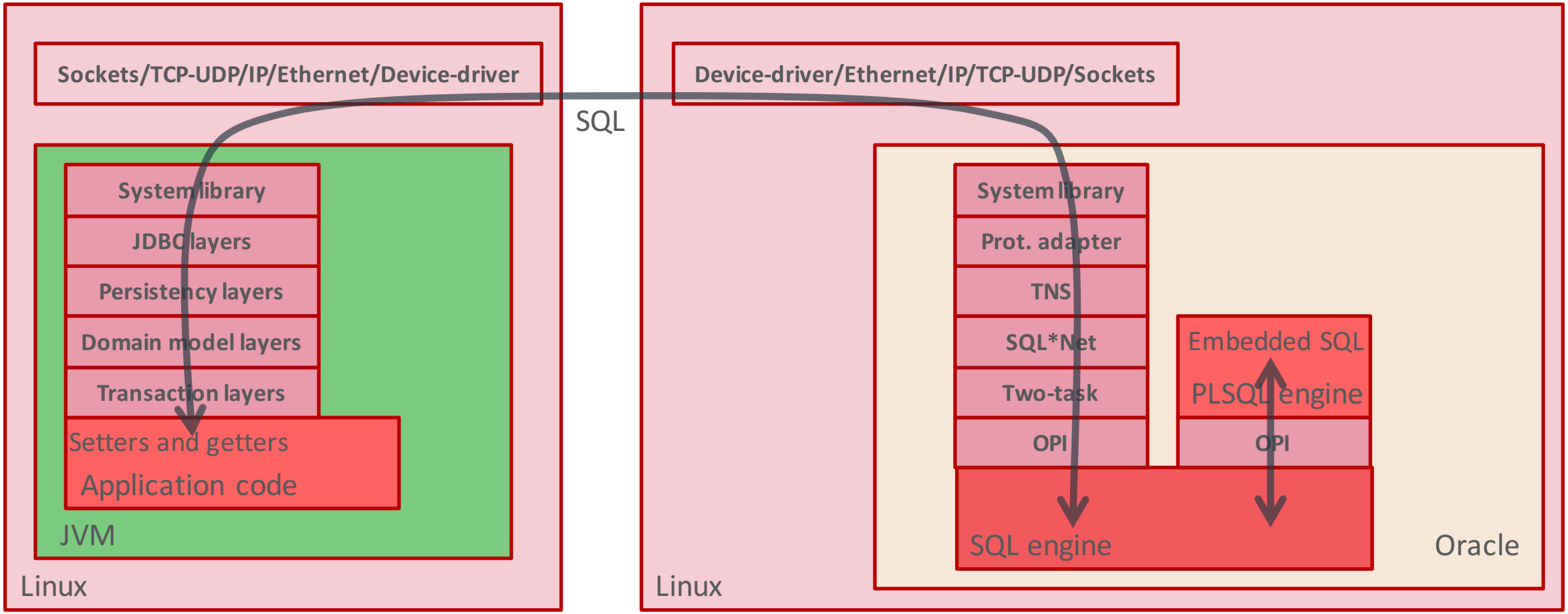
- Outside SQL route:
  - OS network/ipc layers
    - Front door, doormat
  - Net/TNS/TT layers
    - Hallway
  - OPI

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



ADF vs. PL/SQL CPU Efficiency

Why is Row-by-Row PL/SQL Using Less CPU?



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Researched This Through FlameGraphs

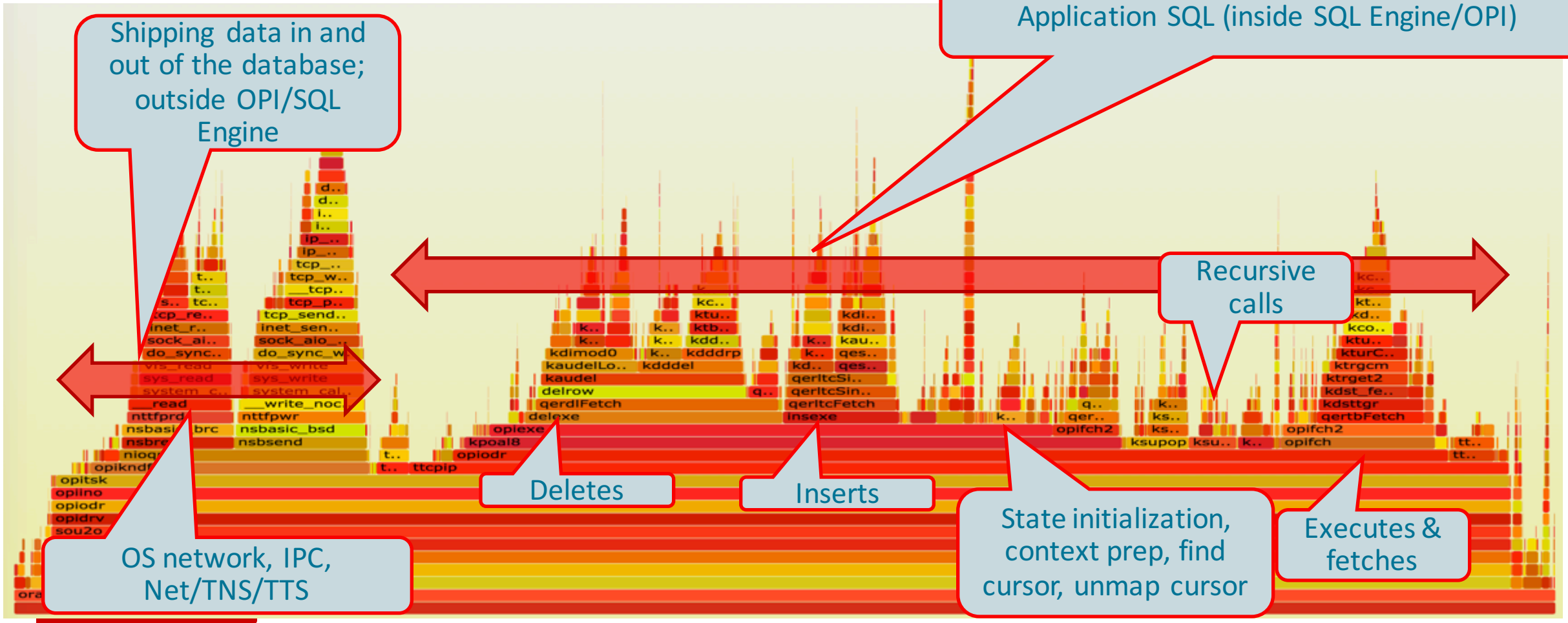
- Flamegraphs visualize proportionally where, in the code, a program spends its time

Generating a FlameGraph

- Is really easy
- All you need is two perl scripts, which you can download from
  - https://github.com/brendangregg/FlameGraph

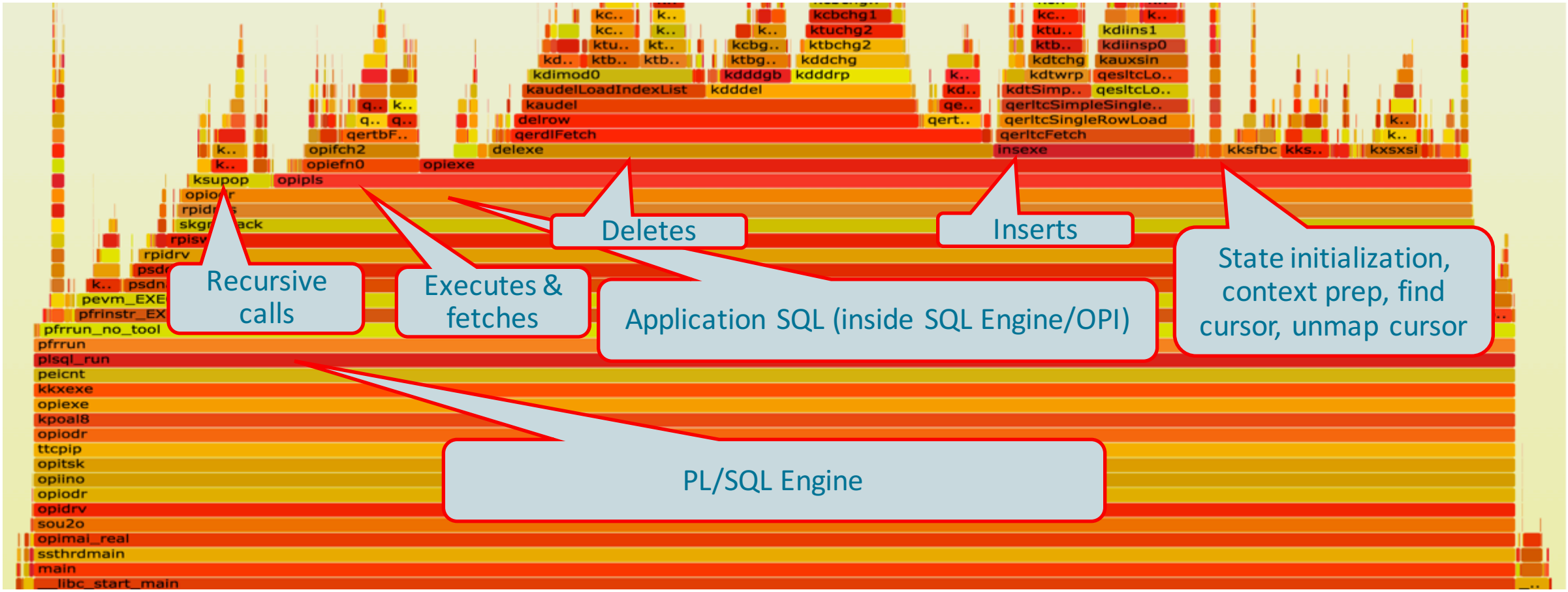
PL/SQL vs. ADF: CPU Impact

ADF Oracle call stack with Flame Graph

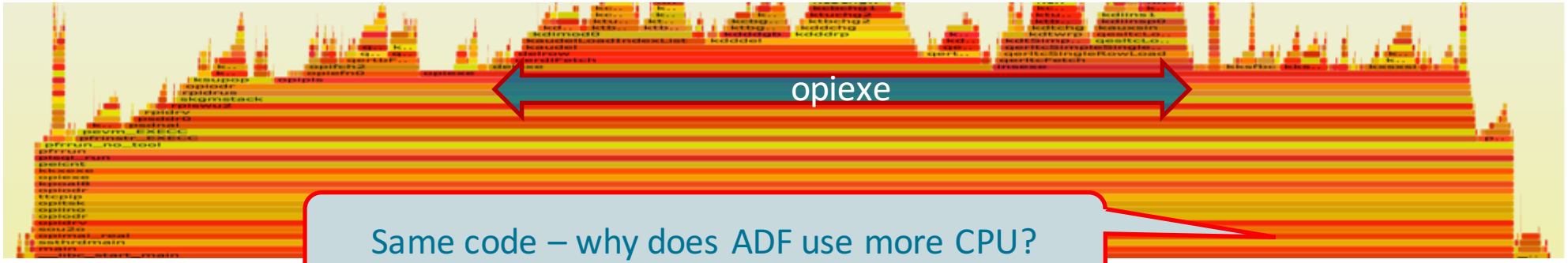


PL/SQL vs. ADF: CPU Impact

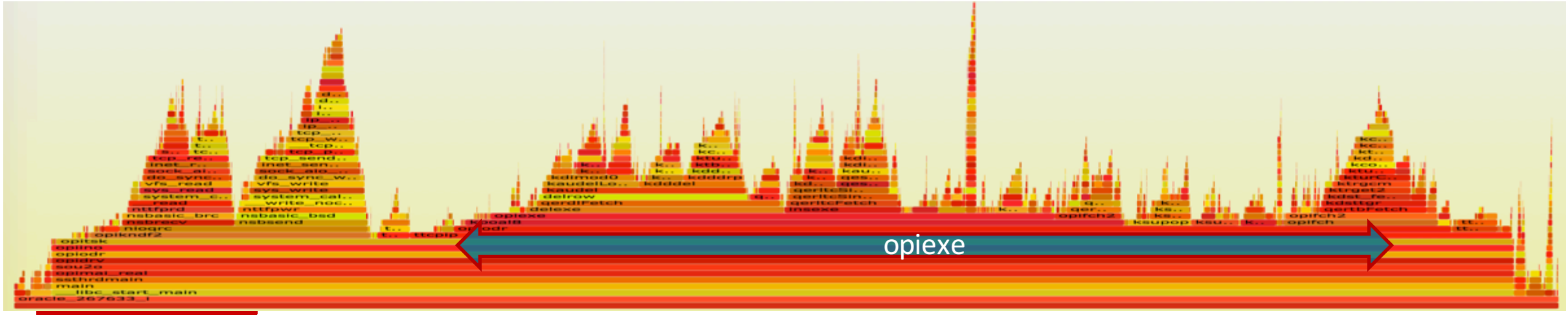
PL/SQL Oracle call stack with Flame Graph



PL/SQL vs. ADF: CPU Impact



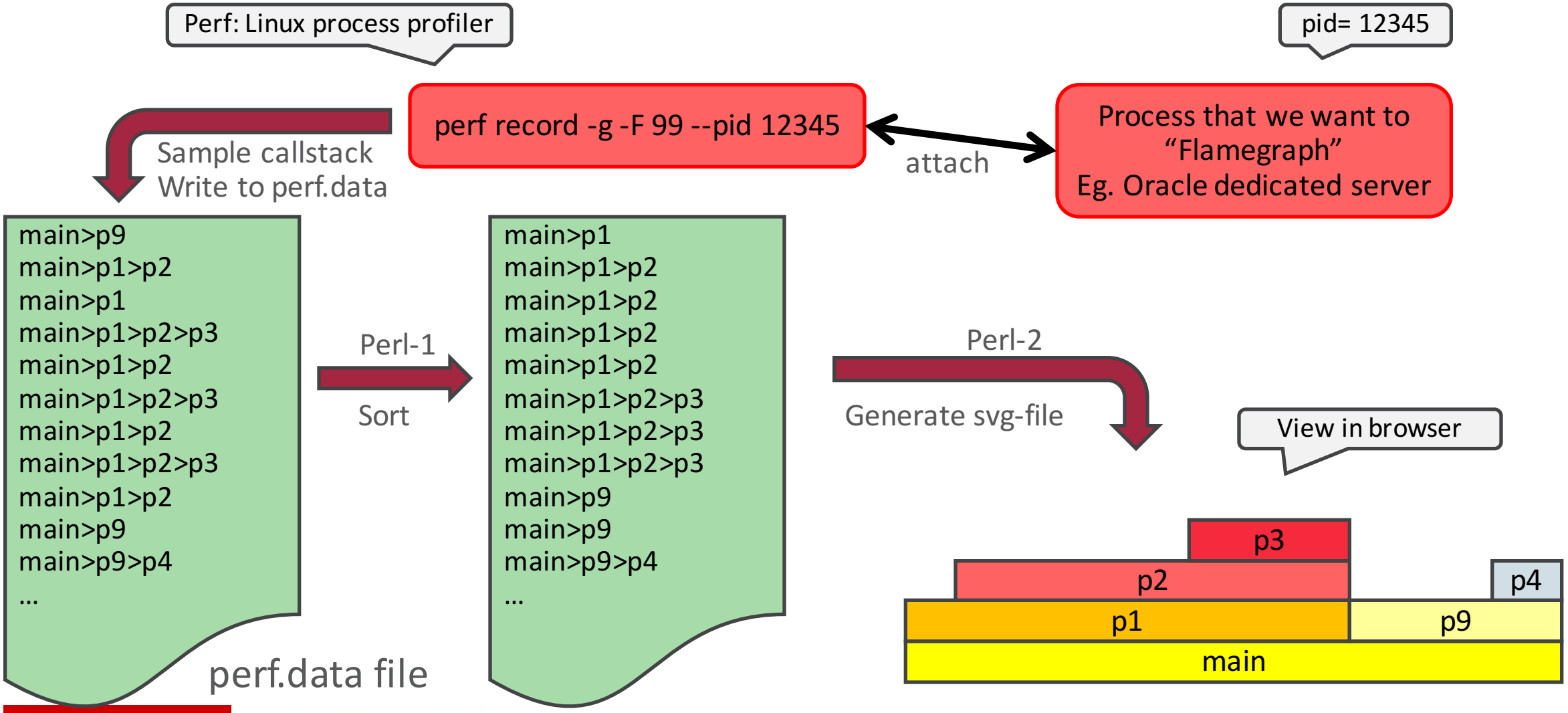
Same code – why does ADF use more CPU?



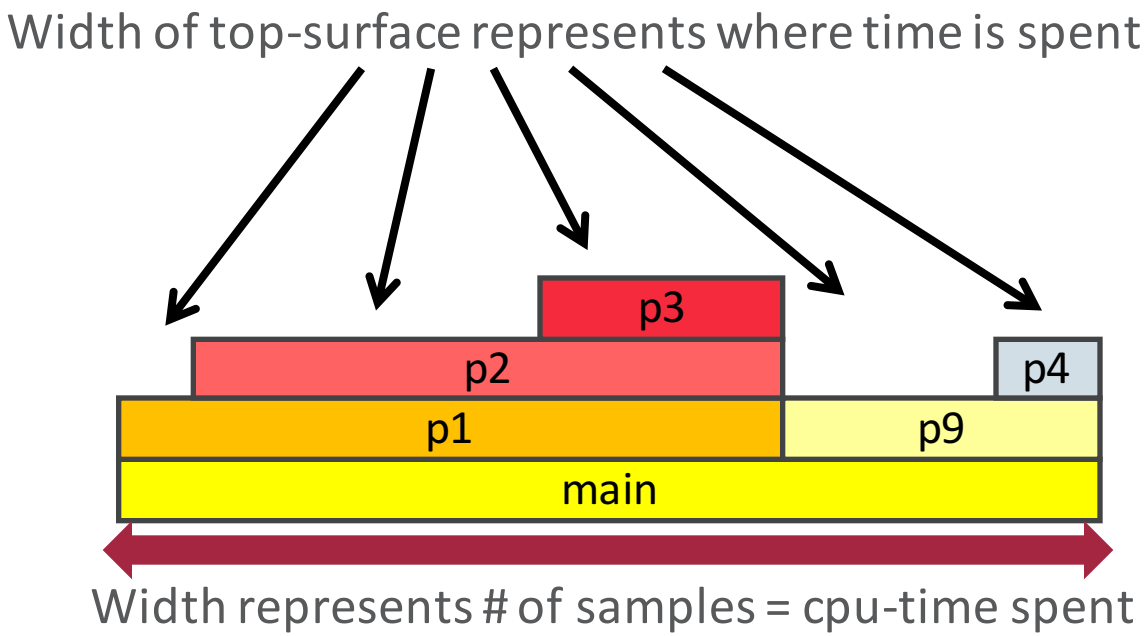
ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Generating a FlameGraph



How to Read a Flamegraph



- More info: https://github.com/brendangregg/FlameGraph

Reason #1

- With layered architecture approach, every database call/SQL-statement incurs a RDBMS-entry taks

- 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

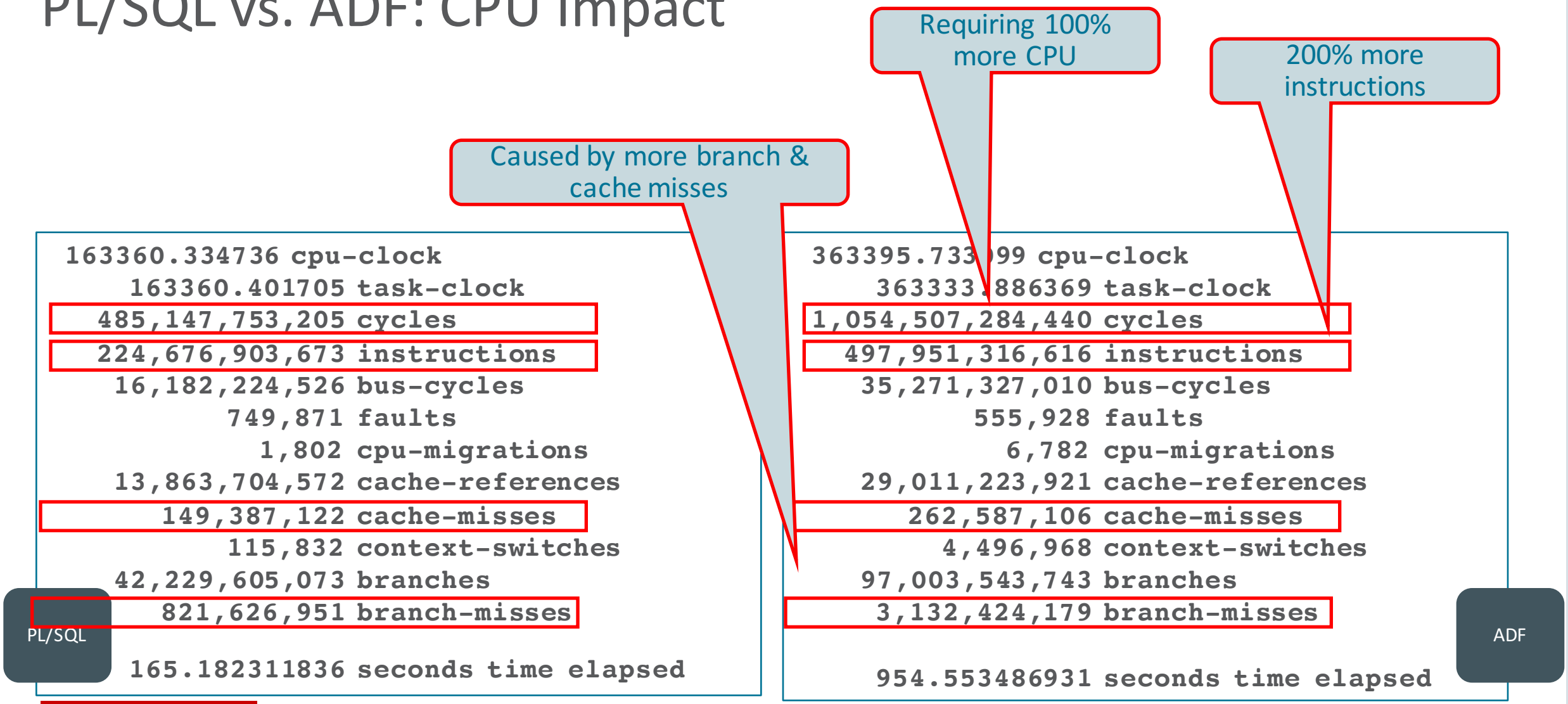
- Modern CPU cores are complex factories (just like RDBMS)
  - As a thread, it's best to stay in factory as long as possible
  - Getting off and back on CPU is very expensive in terms of required clock cycles
  - Process context-switching is most expensive CPU operation

- SmartDB approach has fewer process context-switches – stays on the CPU
- Layered approach deschedules many more millions of times per SQL statement, causing CPU to have to execute additional micro-operations

Researched through CPU profiling

- What this means is: Layered arch. approaches use *more* DB CPU
- What this means is: You use less CPU if you can "stay on it" to get your BL done

PL/SQL vs. ADF: CPU Impact



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Combined: Layered approach Puts >2X Load On RDBMS

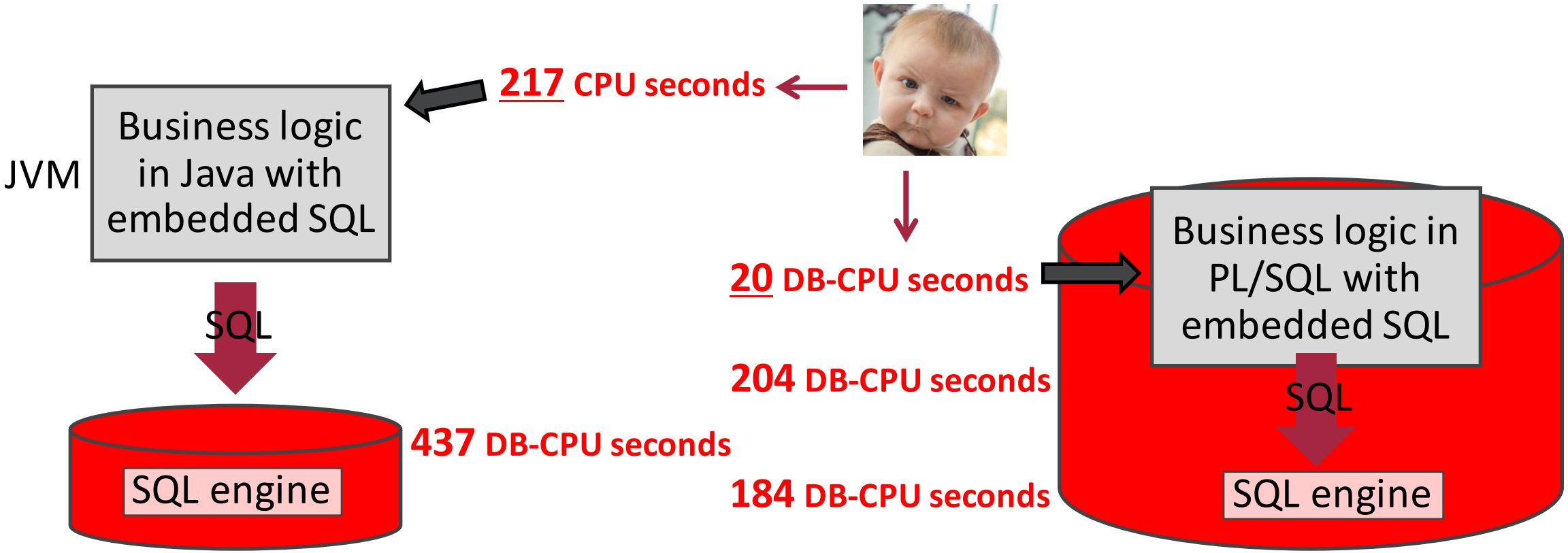
- Reason #1: Overhead in Oracle kernel per SQL statement
- Reason #2: Overhead at CPU-level due to stack traversal, scheduling, context-switching, etc.

One More Point to Make…

Apart from spending time executing SQL …

- We're also spending time executing the if-then-else-loop language from which SQL gets submitted
  - In Layered approach we're spending time in Java doing business logic
  - In #SmartDB approach we're spending time in PL/SQL doing business logic

Required CPU For Getting the Business Logic Done



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

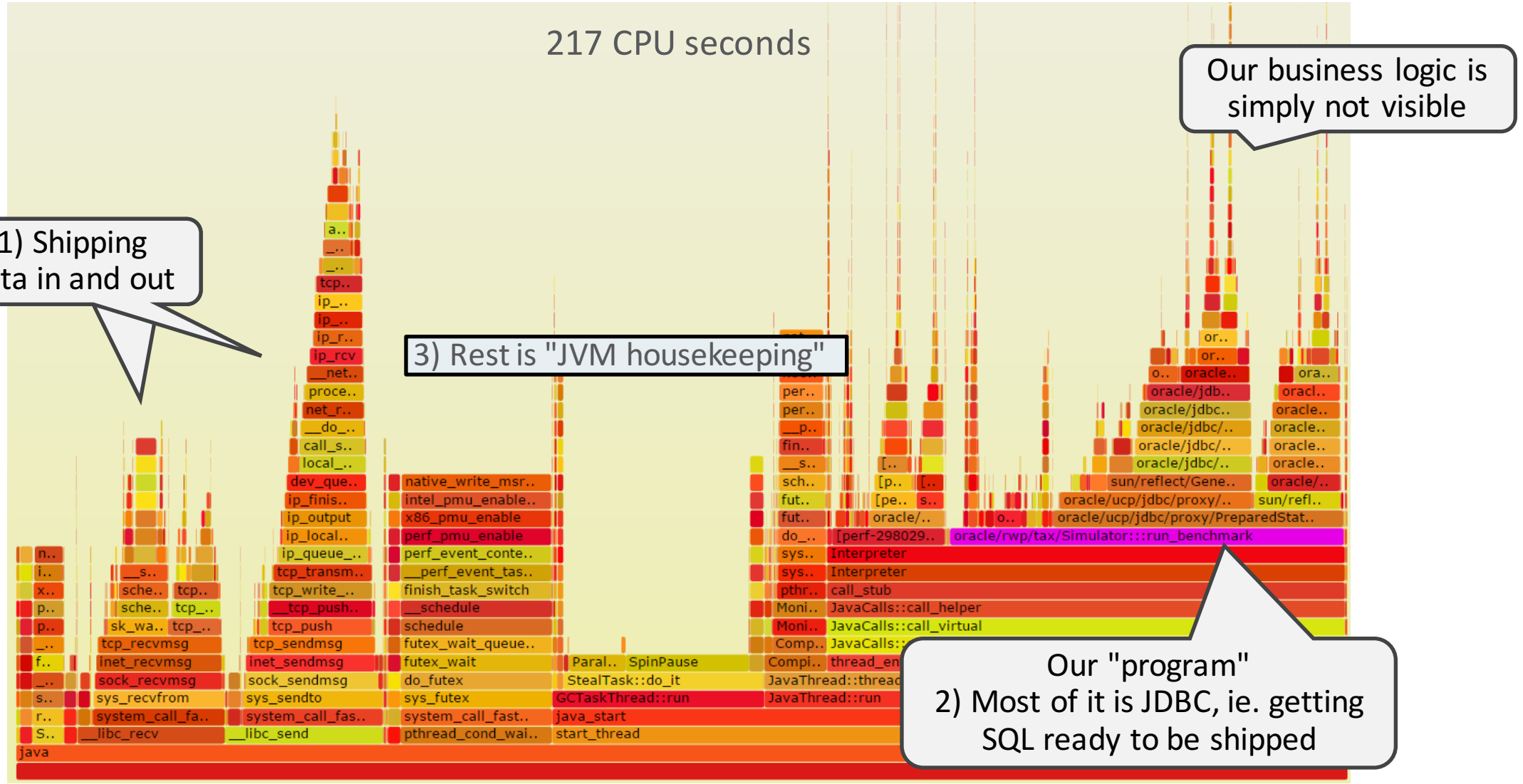
10X!

- 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

- 90% 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

FlameGraph of JVM



In Summary: Row-by-Row Layered vs. Row-by-Row SmartDB

- **If** you have your SQL generated by persistence/ORM frameworks,
- **Then** you'll get chatty, row-by-row, applications
- **Which then** results in hugely inefficient use of resources
  - Both at DB-server (2X) **and** JVM side (10X)
- The more chatty your application is, the worse this will be
  - Also, the greater the latency, the worse this will be
  - Also, the busier your DB server or app server, the worse this will be
- Roundtrips
  - Cause massive increase in required CPU power
  - Nowadays this is probably worse than the "time spent on network"

DEMO

What Does the PL/SQL Array Interface Do for You?

Array Size 1

Statistic Name	Time (s)	% of DB Time	% of Total CPU Time
sql execute elapsed time	1,788.01	99.79	
DB CPU	1,775.04	99.06	87.65
PL/SQL execution elapsed time	163.99	9.15	
parse time elapsed	1.11	0.06	
connection management call elapsed time	0.57	0.03	
hard parse elapsed time	0.28	0.02	
PL/SQL compilation elapsed time	0.09	0.01	
failed parse elapsed time	0.05	0.00	
hard parse (sharing criteria) elapsed time	0.04	0.00	
repeated bind elapsed time	0.00	0.00	
DB time	1,791.80		
background elapsed time	256.98		
background cpu time	250.16		12.35
background IM trickle repopulation elapsed time	1.01		
background IM trickle repopulation cpu time	0.83		0.04
total CPU time	2,025.20		

Array Size 256

Statistic Name	Time (s)	% of DB Time	% of Total CPU Time
sql execute elapsed time	880.24	99.79	
DB CPU	863.03	97.84	82.31
PL/SQL execution elapsed time	65.70	7.45	
connection management call elapsed time	0.38	0.04	
parse time elapsed	0.33	0.04	
hard parse elapsed time	0.21	0.02	
PL/SQL compilation elapsed time	0.09	0.01	
failed parse elapsed time	0.02	0.00	
hard parse (sharing criteria) elapsed time	0.01	0.00	
repeated bind elapsed time	0.00	0.00	
DB time	882.12		
background cpu time	185.45		17.69
background elapsed time	161.75		
total CPU time	1,048.48		

Array processing uses less CPU

DEMO

Tax Demo – Run #3

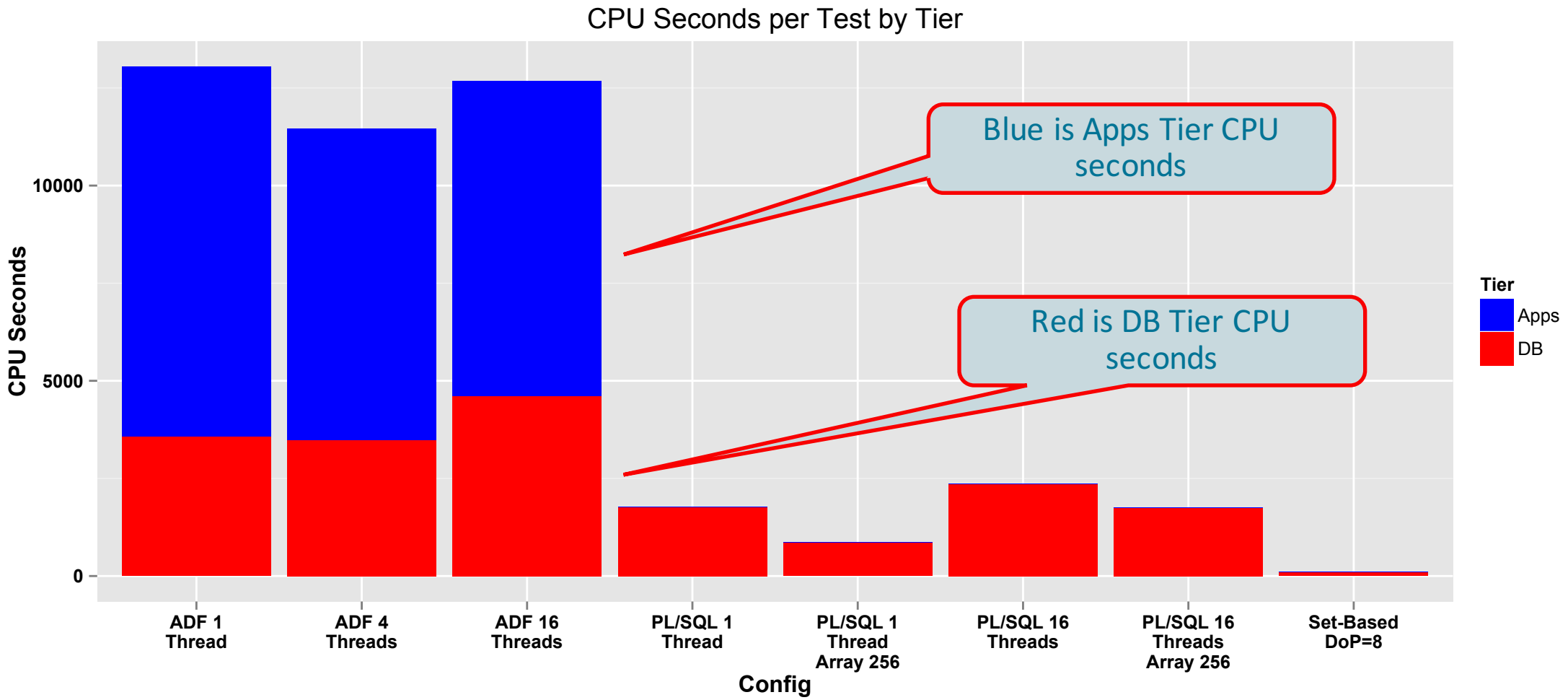
Config (Thread)	Threads/DoP	Array Size	File 1 Seconds	File 2 Seconds	File 3 Seconds	File 4 Seconds	File 5 Seconds	Total Seconds
ADF	16	1	158	192	145	109	198	802
PL/SQL	16	1	37	43	31	23	44	178
PL/SQL	16	256	26	31	24	17	33	131
Set	8	N/A	3	3	2	3	3	14

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

DEMO

CPU Usage



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

DEMO

CPU Usage

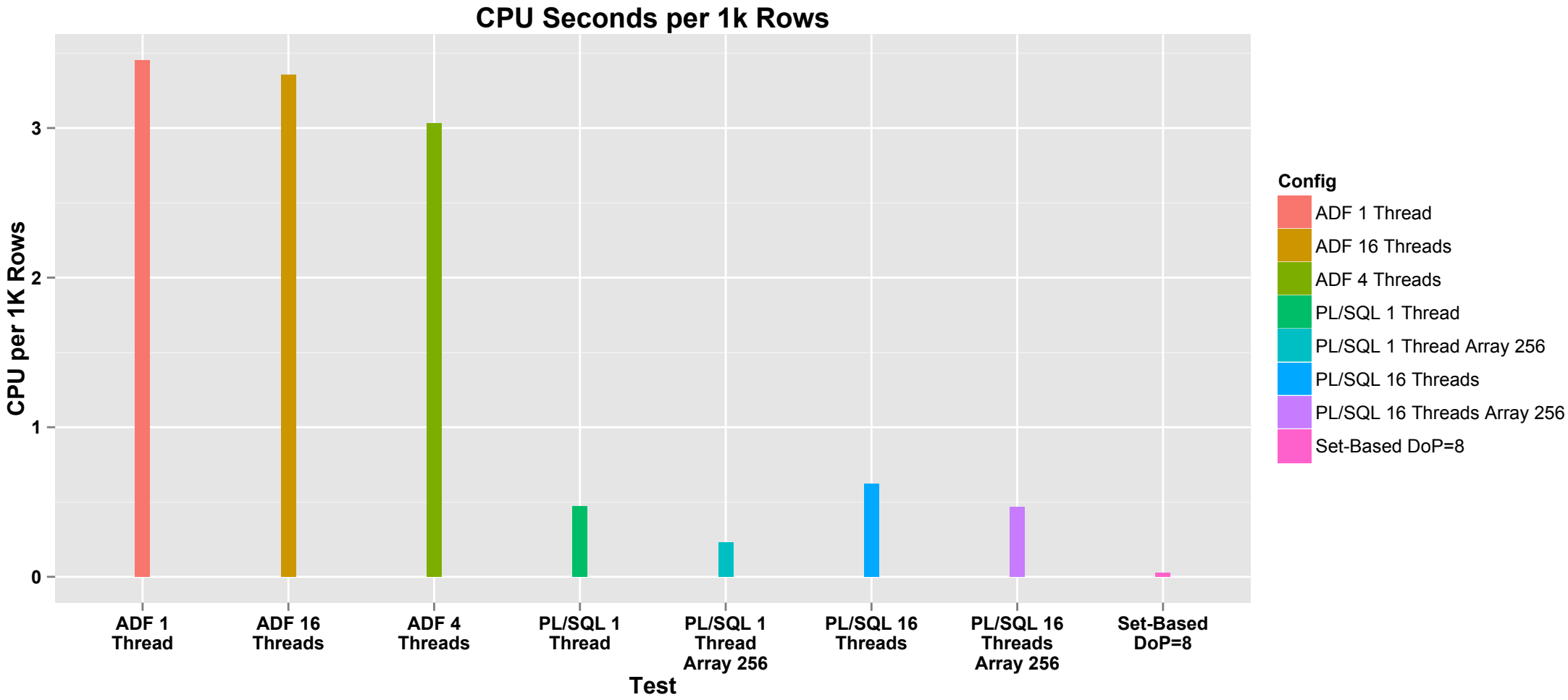
Config	Threads/DoP	Array Size	DB CPU	Apps Tier CPU	Total CPU	CPU per 1k Rows Processed
ADF	1	1	3,583.18	9,464.46	13,047.65	3.45
ADF	4	1	3,476.46	7,330.17	11,454.98	3.03
ADF	16	1	4,608.08	8,039.97	12,699.14	3.35
PL/SQL	1	1	1,775.04	0.00	1,775.04	0.47
Pl/SQL	1	256	863.03	0.00	863.03	0.22
PL/SQL	16	1	2,351.20	0.00	2,351.20	0.62
PL/SQL	16	256	1,755.99	0.00	1,755.99	0.46
Set	16	N/A	101.22	0.00	101.22	0.02

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

DEMO

CPU Usage



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Set-Based Processing

Exploit the power of SQL

- Techniques include:
  - Database Parallelism
  - DDL instead of DML
  - Multi-table inserts with Common Table Expressions
  - Window functions for row labeling
- **Faster**, uses **less CPU**, and **easier** to code and support
- Set-based SQL **moves processing to data** – Oracle DB is a processing engine
- With Set-Based SQL, we can ***choose* our performance** with DoP

Embracing Set-Based SQL

- Once you're in PL/SQL opportunities for set-based SQL open up naturally
  - You, the developer, write the SQL
  - Layered software architectures usually prevent this as, by design, SQL is invisible
- Very often there is opportunity to embrace set-based SQL
  - You specify the “what” and Oracle figures out “how”
    - Speedup of development
  - Replacing row-by-row with set-based SQL also delivers execution speedups
    - Coming from row-by-row Layered architectures, 10X-100X or more often achievable
- You move business logic into SQL (rich SQL)

Original Example: Batch Program

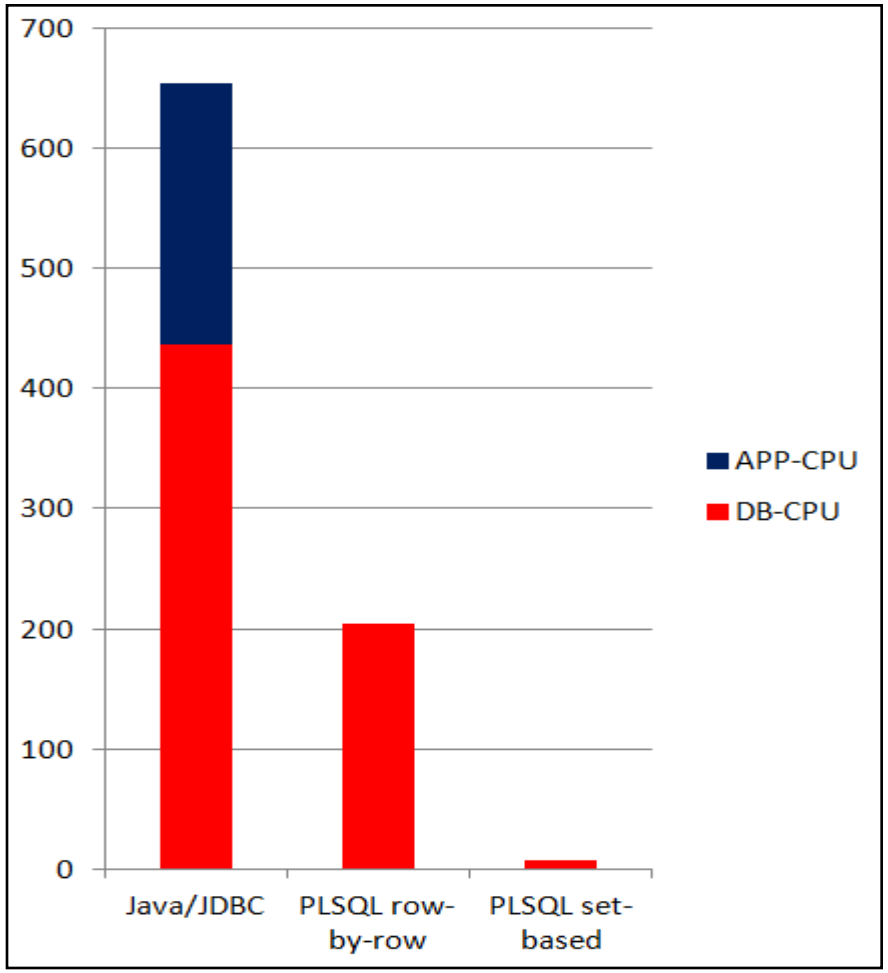
- Able to rewrite using set-based multi-table insert statements (MTI)

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

Original Results Visualized

	Java/JDBC	PL/SQL row-by-row	PL/SQL set-based
DB-CPU	437	204	7
APP-CPU	217	0 (n/a)	0 (n/a)
Total	654	204	7

Almost 100X
Just think about this…



- If you choose Layered, you've committed to:
  - Having to purchase a lot of hardware and DB licenses
  - Blaming database for your performance/scalability issues

  Whereas you should blame your chosen architecture

ORACLE
ORACLE
REAL-WORLD PERFORMANCE

Big Picture

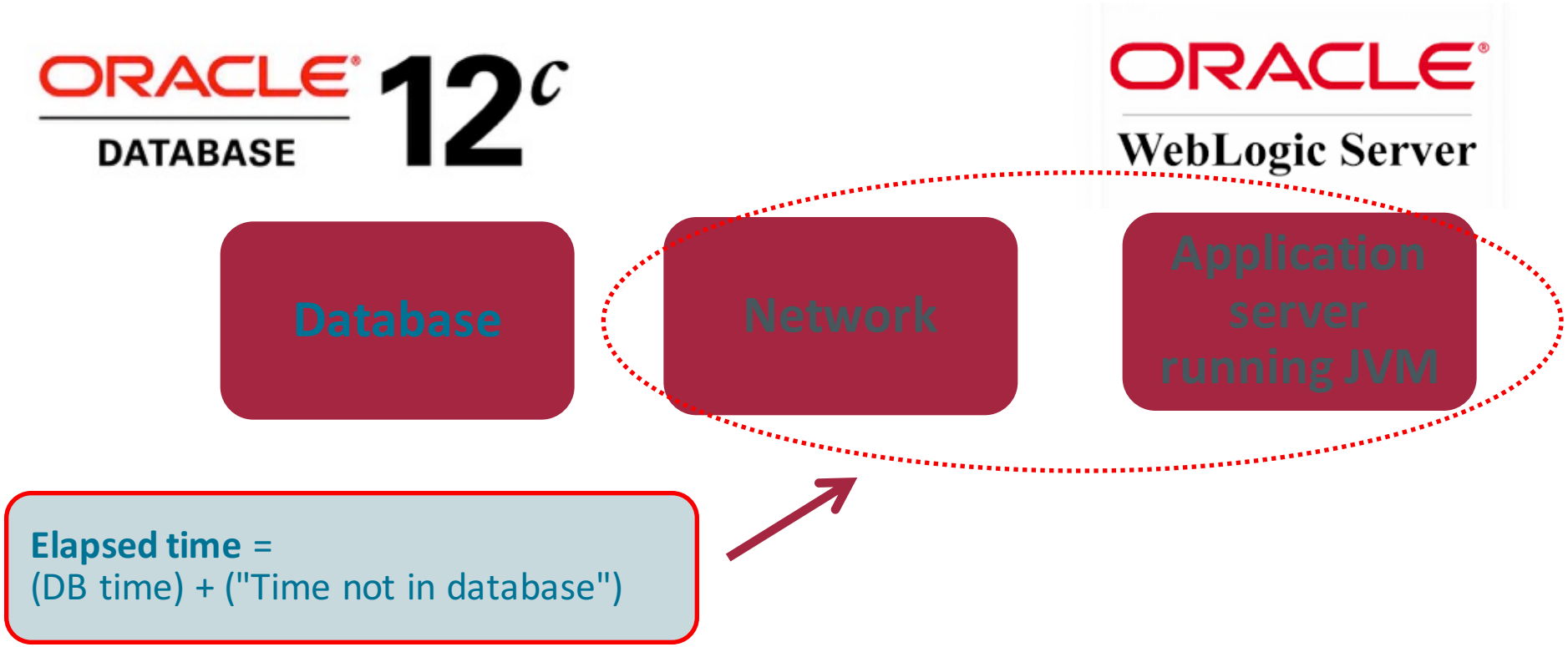
1. Introduction
2. Layered Architecture: History, Landscape, and Issues
3. Demos and Technical Stuff
4. Big Picture
5. Concluding Thoughts

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

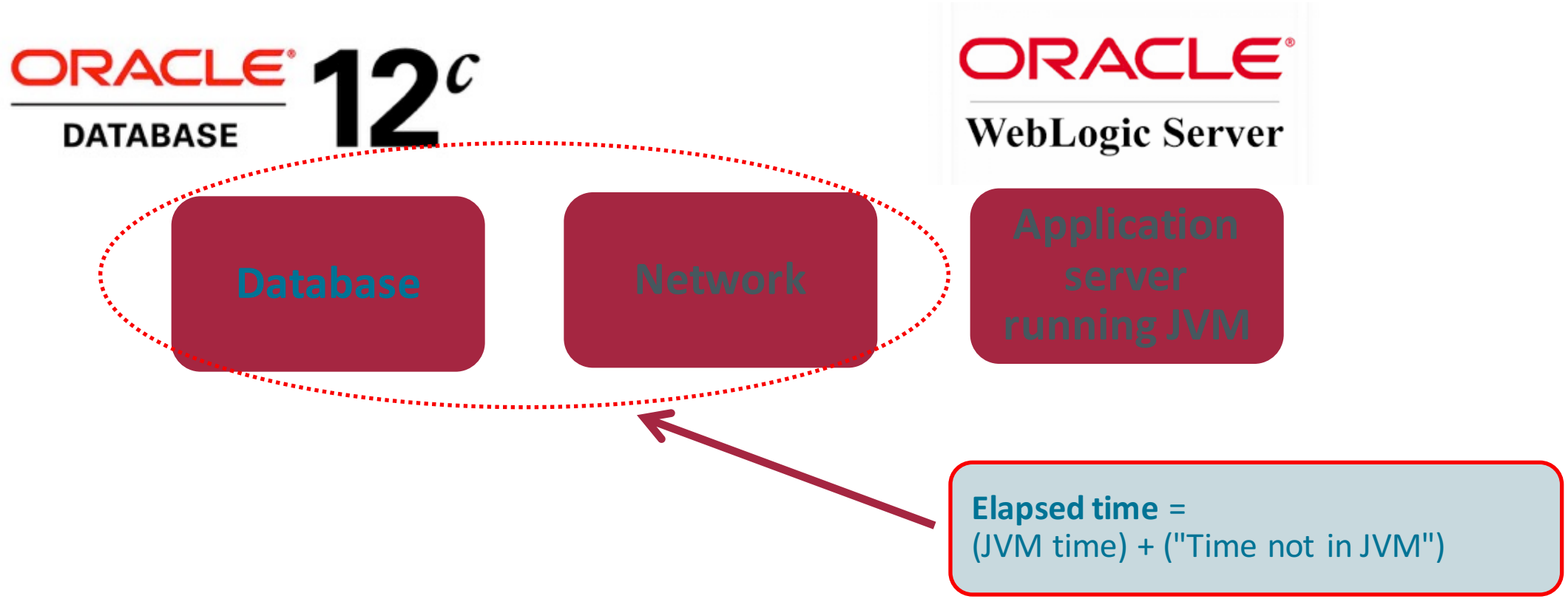
Performance with Layered Architectures

From database's perspective …



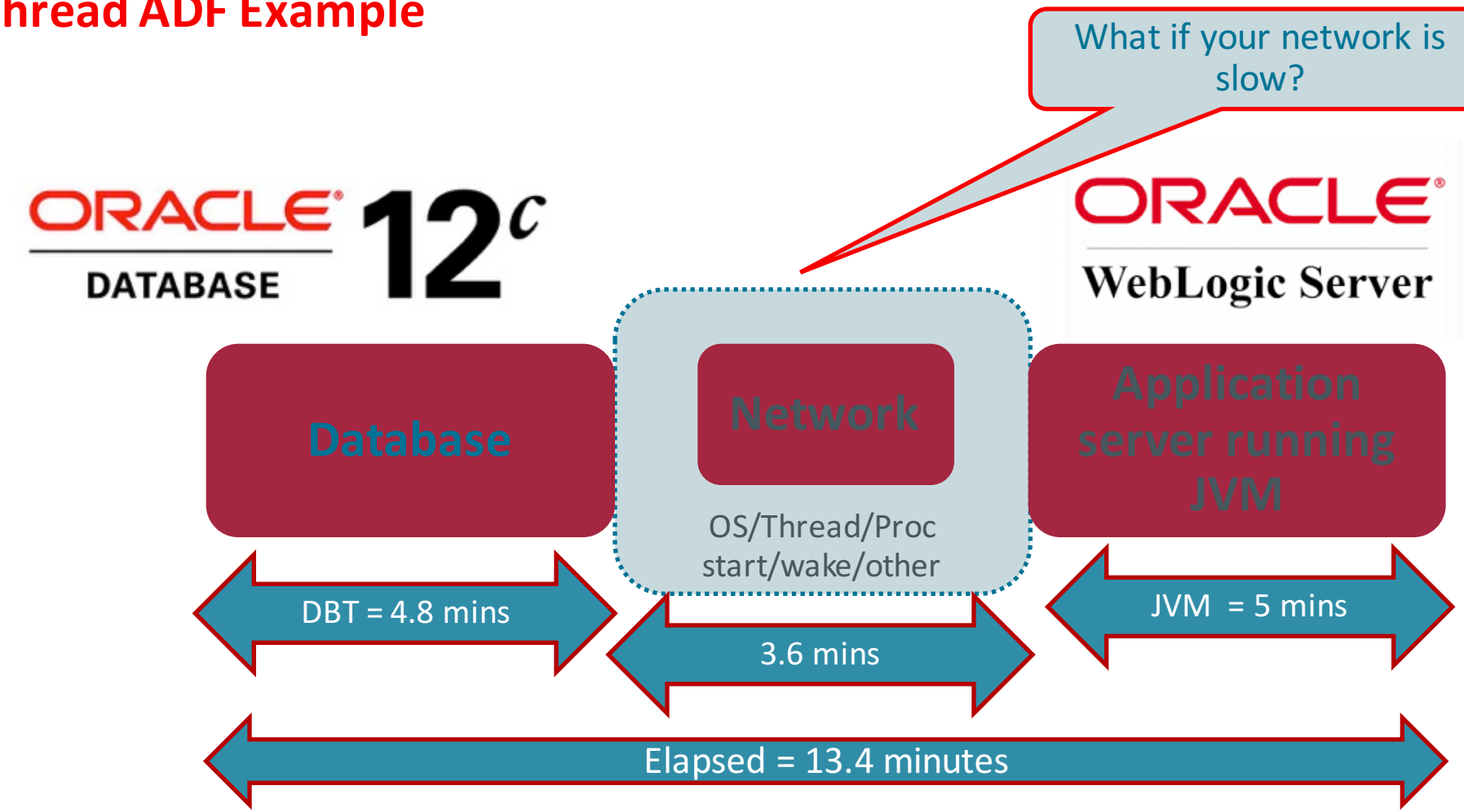
Performance with Layered Architectures

From apps server's perspective …



Performance with Layered Architectures

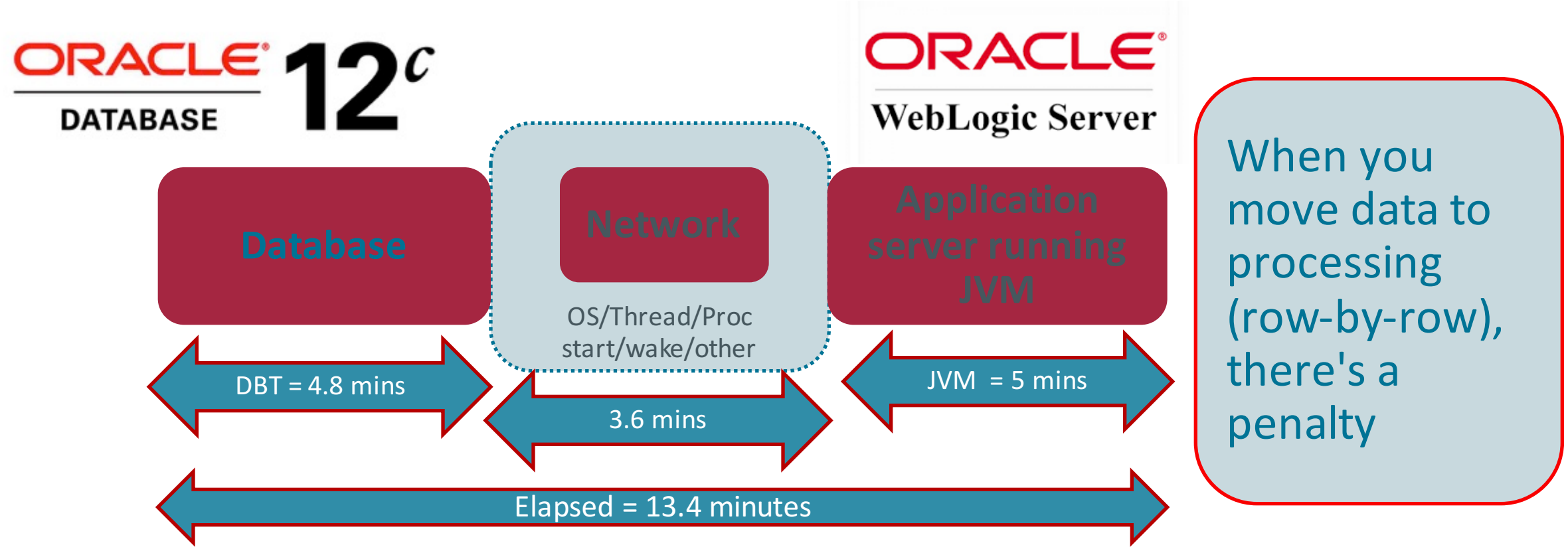
16 Thread ADF Example



ORACLE REAL-WORLD PERFORMANCE

Performance with Layered Architectures

16 Thread ADF Example

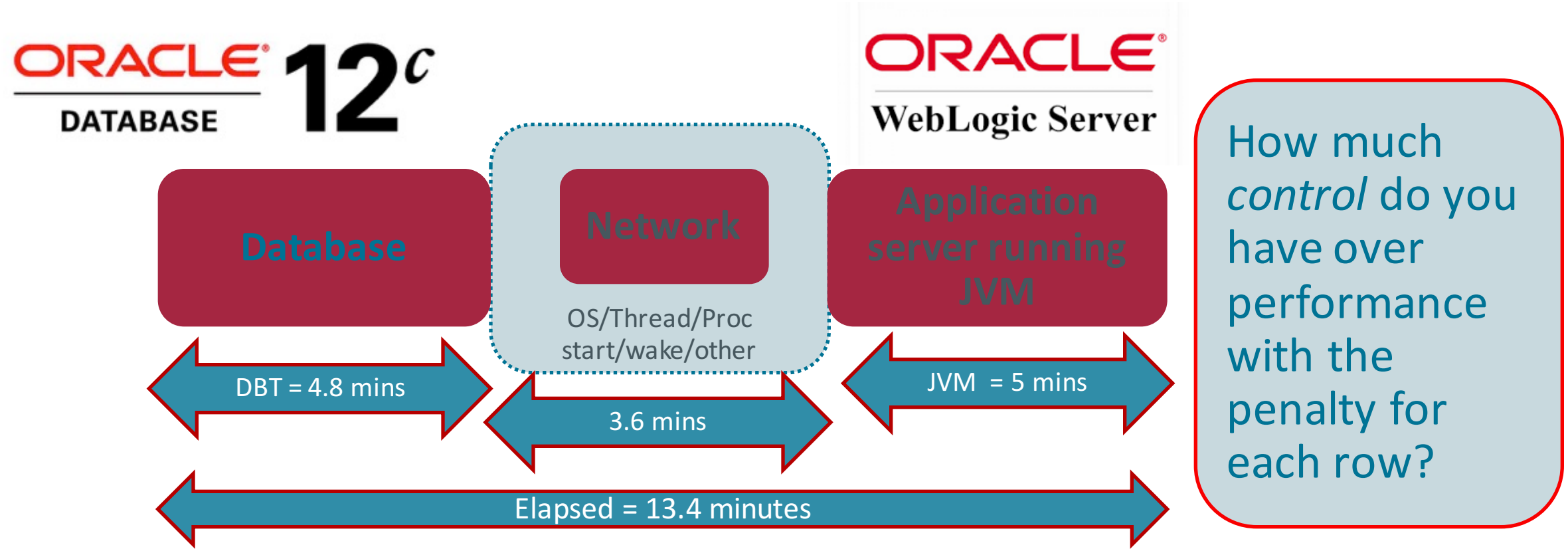


When you move data to processing (row-by-row), there's a penalty

ORACLE REAL-WORLD PERFORMANCE

Performance with Layered Architectures

16 Thread ADF Example



How much *control* do you have over performance with the penalty for each row?

ORACLE REAL-WORLD PERFORMANCE

Performance with Layered Architectures

Where is the Leverage?

Where do you start?

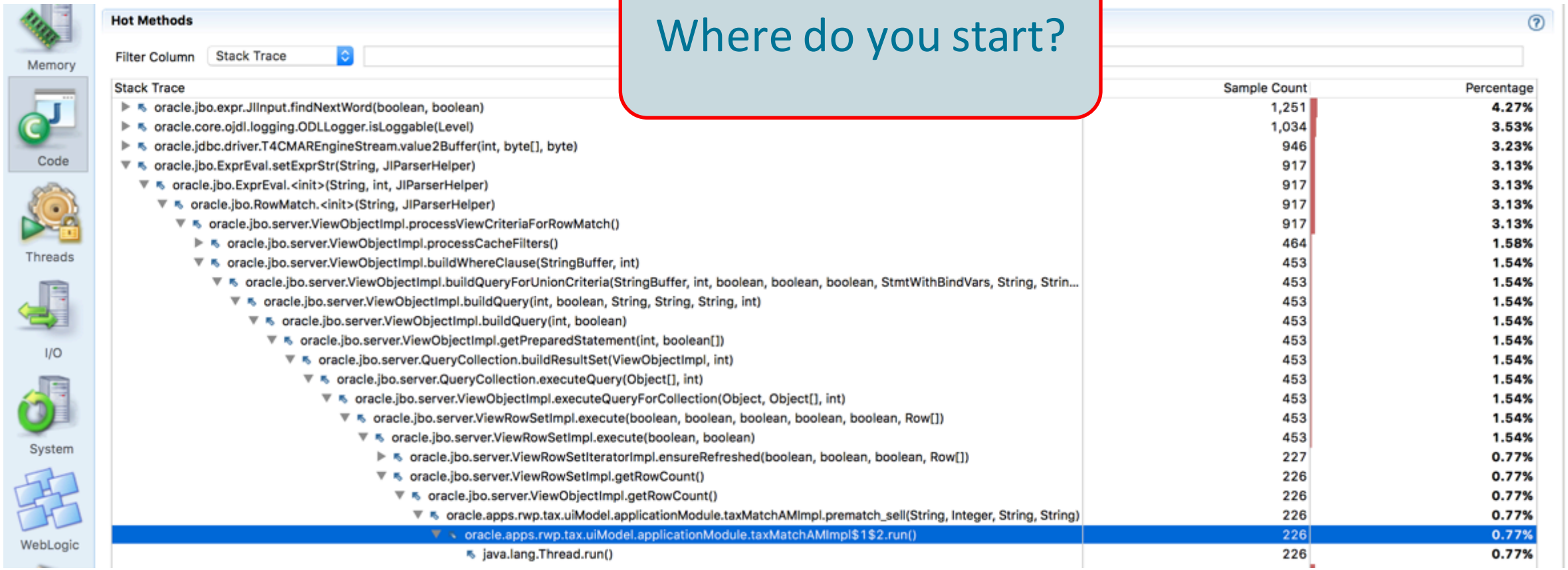
Elapsed Time (s)	Executions	Elapsed Time per Exec (s)	%Total	%CPU	%IO	SQL Id	SQL Module	SQL Text
316.77	3,777,402	0.00	6.80	36.42	0.12	gn7jdbsw9mcfm	JDBC Thin Client	DELETE FROM prematch_buy where...
288.07	3,777,501	0.00	6.19	34.59	0.05	15qbk6ss8aqkh	JDBC Thin Client	DELETE FROM prematch_sell wher...
263.29	3,777,369	0.00	5.66	36.14	0.15	7tmbmtzdb8kpu	JDBC Thin Client	INSERT INTO matched (CODE, SO...
247.46	3,777,509	0.00	5.32	36.33	0.79	7mhs1jc6jzhnj	JDBC Thin Client	SELECT PrematchSellEO.CODE, Pr...
238.00	3,776,857	0.00	5.11	33.06	3.05	2jbwsbn8mzzst	JDBC Thin Client	INSERT INTO prematch_buy (COD...
236.36	3,776,369	0.00	5.08	32.84	2.34	9q3c6x15qfb9r	JDBC Thin Client	INSERT INTO prematch_sell (CO...
200.39	7,551,830	0.00	4.30	28.19	0.09	9gjmmvu0t3brk	JDBC Thin Client	SELECT MatchedEO.CODE, Matched...

In the land of
incremental gains

Performance with Layered Architectures

Where is the Leverage?

Where do you start?



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Performance with Layered Architectures

Throwing application parallelism at the problem

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	95.2	0.1	0.00	0.47
DB CPU(s):	32.0	0.0	0.00	0.16

Over-processed

Contention

Event	Waits	Total Wait Time (s			s
DB CPU		71			
buffer busy waits	948,177	33			y
external table read	199,936	29			
latch: redo allocation	205,559	1644.5	8.00	7.7	Other
latch: ges resource hash list	94,334	977.3	10.36	4.6	Other
library cache: mutex X	239,359	894.2	3.74	4.2	Concurrency
log file switch (checkpoint incomplete)	932	659.6	707.68	3.1	Configuration
latch: cache buffers chains	115,010	414.1	3.60	1.9	Concurrency
enq: TX - index contention	45,531	354.4	7.78	1.7	Concurrency
cursor: pin S	72,982	238.2	3.26	1.1	Concurrency

Is this the point you start bottom-up analysis?

Performance with Layered Architectures

Throwing application parallelism at the problem

```
AdminServer-diagnostic.log:[2016-09-15T14:03:23.210-07:00] [AdminServer] [NOTIFICATION] [DFW-40101] [oracle.dfw.incident]
[tid: TaxMatch125] [userId: <anonymous>] [ecid: 47e4c839-a6ff-4dc8-b4f6-7155f95baca7-00000023,0:4:1548] [APP: RwpTax2] An
incident has been signalled with the incident facts: [problemKey=DFW-99998
[oracle.jbo.pool.ResourcePoolException][oracle.jbo.pool.ResourcePool.allocateResource][RwpTax2] incidentSource=SYSTEM
incidentTime=Thu Sep 15 14:03:23 PDT 2016 errorMessage=DFW-99998 executionContextId=null]
AdminServer-diagnostic.log:[2016-09-15T14:03:23.210-07:00] [AdminServer] [WARNING] [DFW-40125] [oracle.dfw.incident]
[tid: TaxMatch125] [userId: <anonymous>] [ecid: 47e4c839-a6ff-4dc8-b4f6-7155f95baca7-00000023,0:4:1548] [APP: RwpTax2]
incident flood controlled with Problem Key "DFW-99998
[oracle.jbo.pool.ResourcePoolException][oracle.jbo.pool.ResourcePool.allocateResource][RwpTax2]"
e21b0a54-30f2-4028-bf18-a1c62871c7d4-00000373,0:2:11:1556284] [errid: 349] [detailLoc: /nfs-
shared/wls/home/user_projects/domains/rwp_domain/servers/AdminServer/adr/diag/ofm/rwp_domain/AdminServer/incident/incdir_
349] [probKey: DFW-99997 [java.lang.OutOfMemoryError]] [APP: RwpTax2] incident 349 created with problem key "DFW-99997
```

What do you do when you run out of application server resources?

Can you really *choose* your performance this way?

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Concluding Thoughts

1. Introduction
2. Layered Architecture: History, Landscape, and Issues
3. Demos and Technical Stuff
4. Big Picture
5. Concluding Thoughts

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Remarks

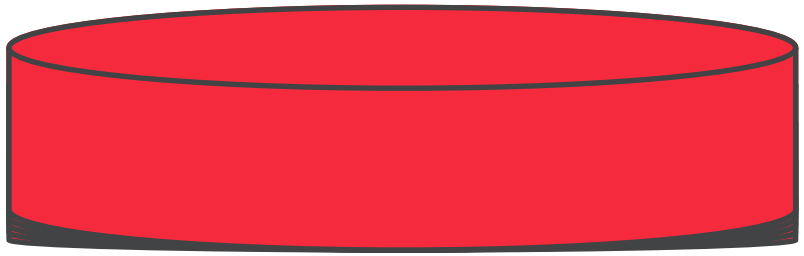
1. The implication of all this
2. SQL isn’t accidental
3. “My enterprise application is too complex”
4. Beware of risks if you go SmartDB

1: The Implication Of All This, Visualized

Layered

App server | App server | App server | App server | App server

500 TX/Sec



Database is always first bottleneck

SMARTDB

500 TX/Sec

With SmartDB you can process more with same DB hardware

With SmartDB you can process same with less DB licenses

Set-based SQL…

1: The Implication Of All This

- Moving poor SQL into PL/SQL likely frees up 50% of your DB-CPU time

- Moving business logic from layered sw-architecture in JVM's, to straightforward PL/SQL, makes it require 10X less CPU

- Your question 1: does the 10X less CPU fit in the 50% freed up DB-CPU's?

  In case, "no":

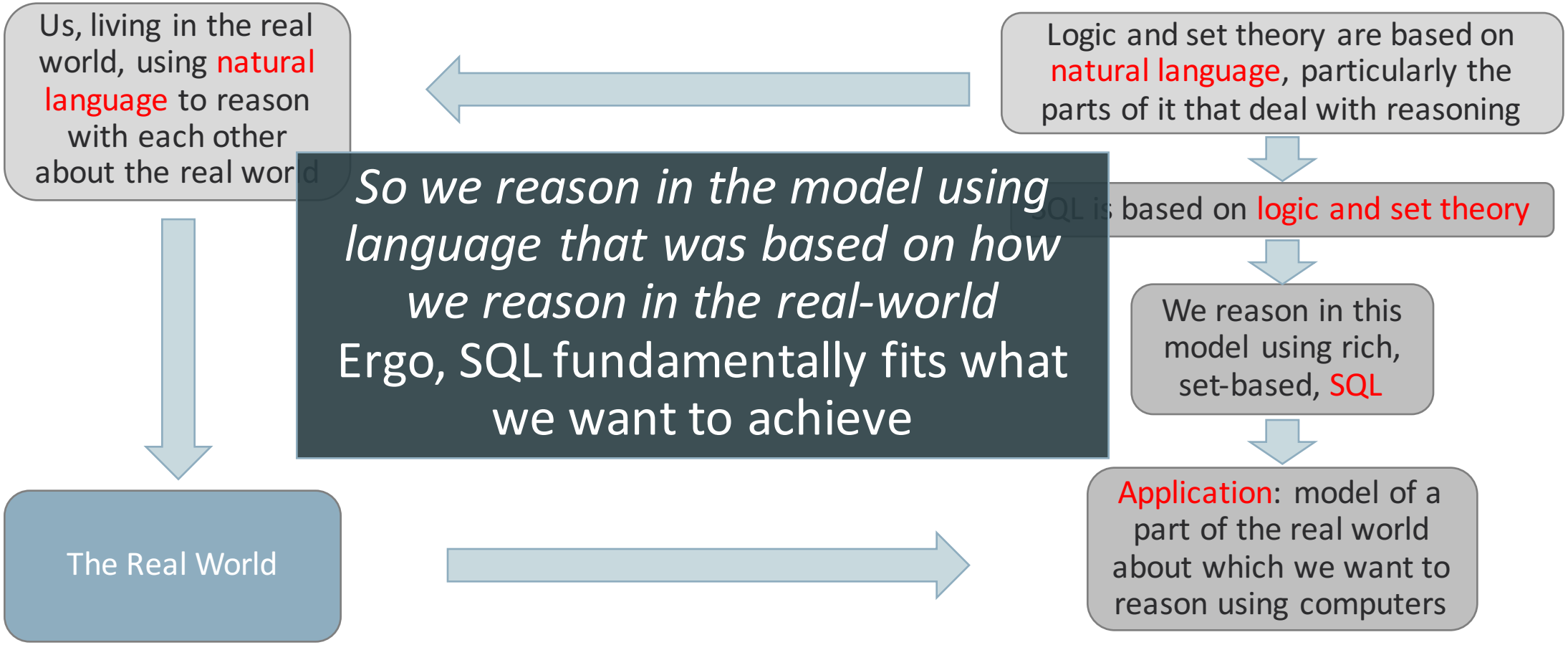
- Your question 2: where can I embrace set-based SQL to make it "yes"?

2: SQL Isn't Accidental, 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, It's Fundamental



3: My Enterprise 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 enterprise 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
- Counterpoint – be prepared for your developers to say "I don't *want* to move my application logic …"

3: Often This Is The Issue When SQL + PL/SQL Are Dismissed

- A mindshift is required:

- You need to start thinking in "processing data"
- Instead of "interacting with objects that have behavior"

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

4: 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

4: 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

Conclusions

Can You Choose Your Performance with Layered Architectures?

Layered architectures don't let you *choose* your performance

- Row-by-row algorithms across multiple stacks = more time, less throughput
- This per-row tax multiplies as data volume increases
- You can only control a small piece – the piece of code or infrastructure you own
- Increased CPU usage, which increases software/hardware/Cloud costs
- Easy to run into contention / bottlenecks and difficult to resolve – usually involves complex application code change

Conclusions

Moving Processing to Data

Layered architectures drive data to processing, not processing to data

- Data shipped to Apps tier at unit of lowest common denominator – a row (or column!)
- Database viewed as persistence layer, not processing engine
- Equates to:
  - Lots of round trips, lots of stack traversal, thread/process sleep/wakeup/start/stop
  - … means more CPU and less efficient, and
  - … both performance and resource usage dependent on # of rows/columns
- Increased CPU = higher software/hardware/Cloud

Conclusions

Incremental Performance Improvements

Layered architectures foster a bottom-up performance culture

- Layer owners only have visibility / responsibility over small piece of puzzle
- Owners spend lots of time "tuning" their piece, chasing percentage points
- Lack of ability to innovate
- "Race to the bottom" mindset – expert in each layer focuses on their layer and nobody looks holistically

If We Have Time …

- Going #SmartDB, or
- Connections and Connection Pools (it's relevant, I promise)

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Integrated Cloud

Applications & Platform Services

ORACLE

ORACLE
REAL-WORLD PERFORMANCE