Database Performance Core Principles

Toon Koppelaars
Consulting Member of Technical Staff
Real-World Performance
Server Technologies

About Me

Part of Oracle eco-system since 1987

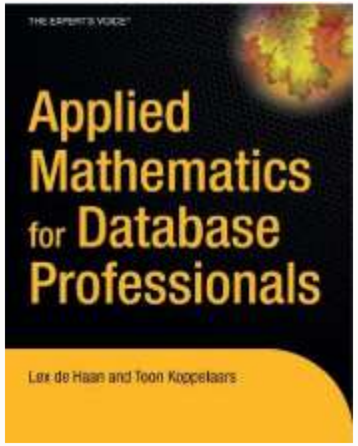
- Have done and seen quite a lot of application development
- Database design, SQL and PL/SQL

Big fan of "Using Database As a Processing Engine"

- Not just as a layer to persist data

Member of Oracle's Real-World Performance Group

@ToonKoppelaars



ORACLE
REAL-WORLD PERFORMANCE

My Dilemma…

- That what you are about to hear, is probably not going to be of much help to you
- Why?
- Because you are stuck with applications that can't be changed
- Those applications could run faster *and* use less computing resources, yet would require re-architecting
- I can only create awareness of this
- My hope is that you can use this talk's knowledge to influence architectural decisions of future applications

Performance Core Principles: Why?

- If I have a single CPU how many processes can be active at any one time?
  - And if I have four CPU’s?

- What’s the implication if there are more processes wanting to run than there are CPU’s?

- If my process needs to access an external resource (network, I/O) what happens to it?
  - Once request has finished, what decides when my process gets back onto CPU and how much is it allocated?

- What is the difference between user and sys CPU?

Agenda

Database Performance Core Principles

- Implications of Oracle's Process Based Architecture
- End User Response Time, Throughput, and DB Time

What happened last two decades with application development?

- It fundamentally moved away from these principles

Database Performance Core Principles

Oracle instance = Process based architecture:

Foreground process is serving your database calls

To perform efficiently:

1. process needs to get on CPU as quickly as possible

Process needs to stay on CPU as long as possible:

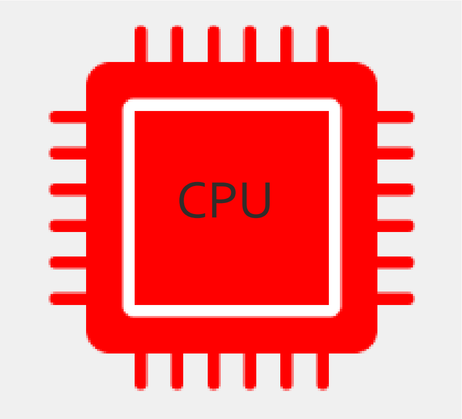
2. process shouldn't go to sleep voluntarily a lot
3. process should experience as few involuntary sleeps as possible

1: Get On CPU

Idle/sleeping

Oracle Foreground Process

Idle



1: Get On CPU

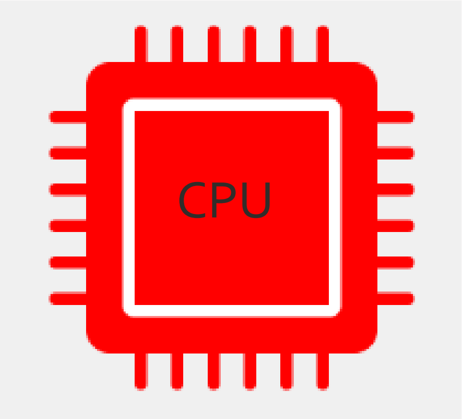
Application issues database call

Idle/sleeping

Oracle Foreground Process

OS receives interrupt: figures out which process to wake-up
Scheduler needs to get this process on CPU

Idle



All work performed by OS and scheduler is reported as SYS cpu

1: Get On CPU

- Only one process can be on a core at a time (it's either 1 or 0)
- How quickly you can get on core depends on current utilization
- Higher busy rate ➔ higher chance of spending time on run-queue before getting scheduled
- Less straightforward with multiple cores

CPU Utilization	Chance of getting scheduled immediately
50%	1 in 2
66%	1 in 3
75%	1 in 4
80%	1 in 5
90%	1 in 10

Impact of CPU Utilization on Database Call Response Time

- Not only does it take longer to get on CPU, moreover:
  - Busy rate has measurable impact on DB-call response time

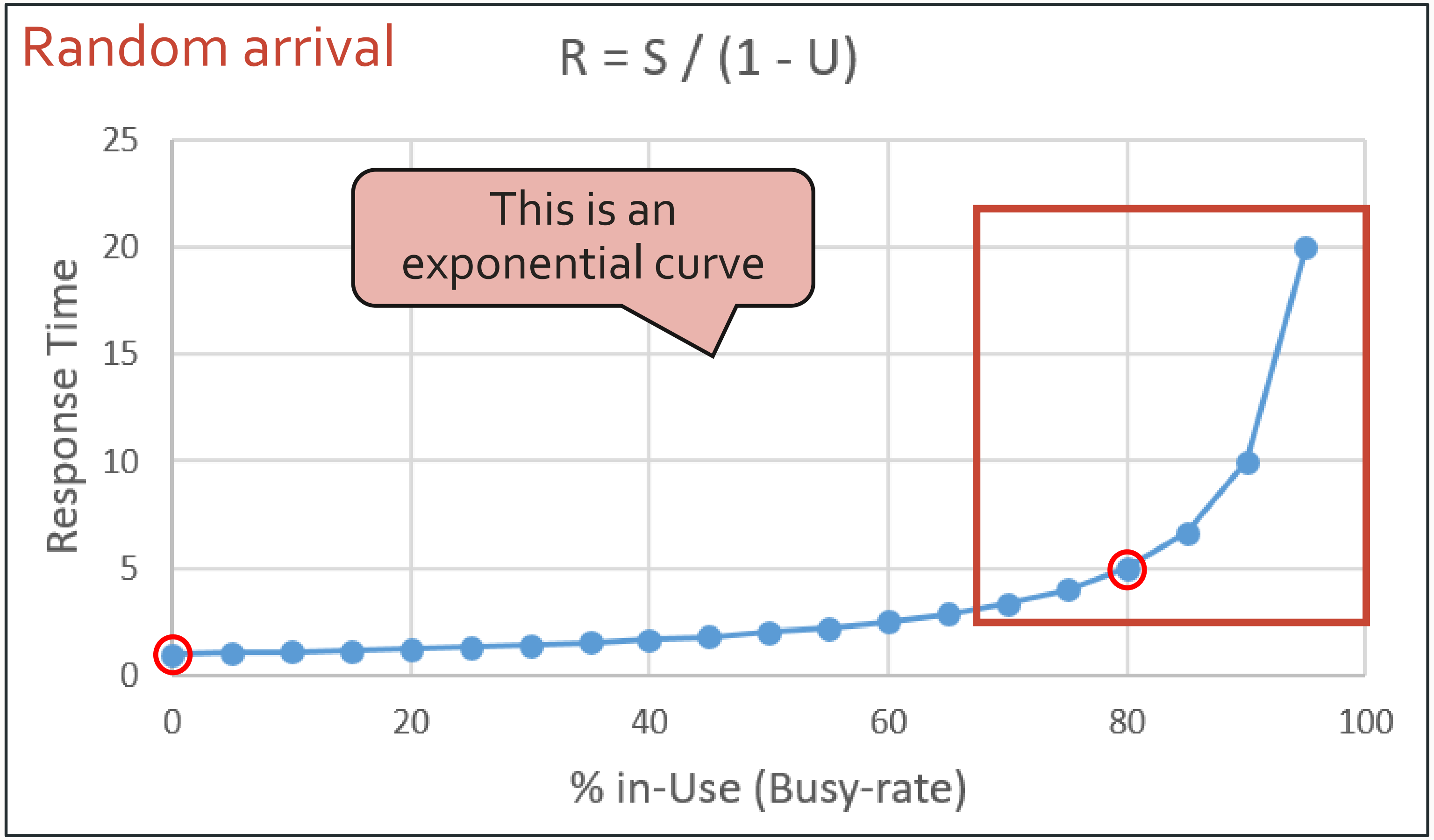
- Which becomes noticeable when CPU has 60-65+% busy rate
  - This is in essence basic queueing theory, nothing new…

Database Call Response Time, Single Core

$R = S / (1 - U)$

Response time
Service time
Utilization%

R – S = queue-time while resource is servicing others

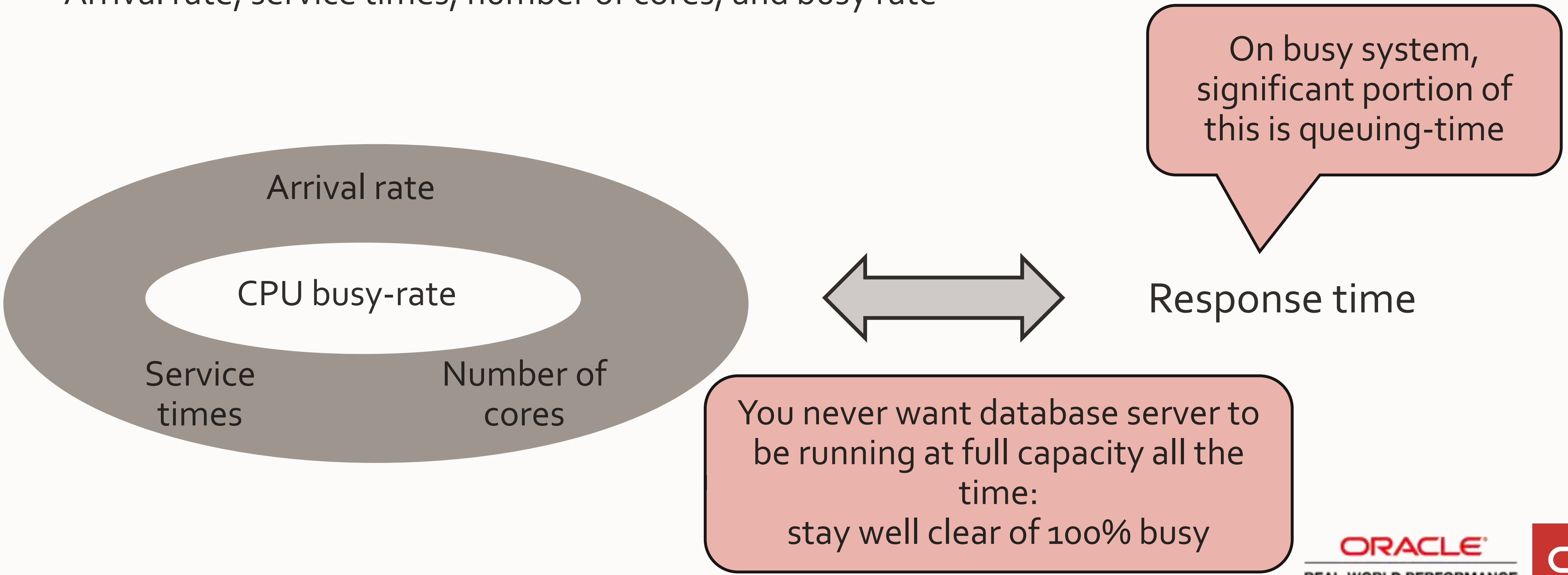


More Complex With Multiple Cores

Every system starts queueing at some point, impacting response times

This depends upon:

- Arrival rate, service times, number of cores, and busy rate

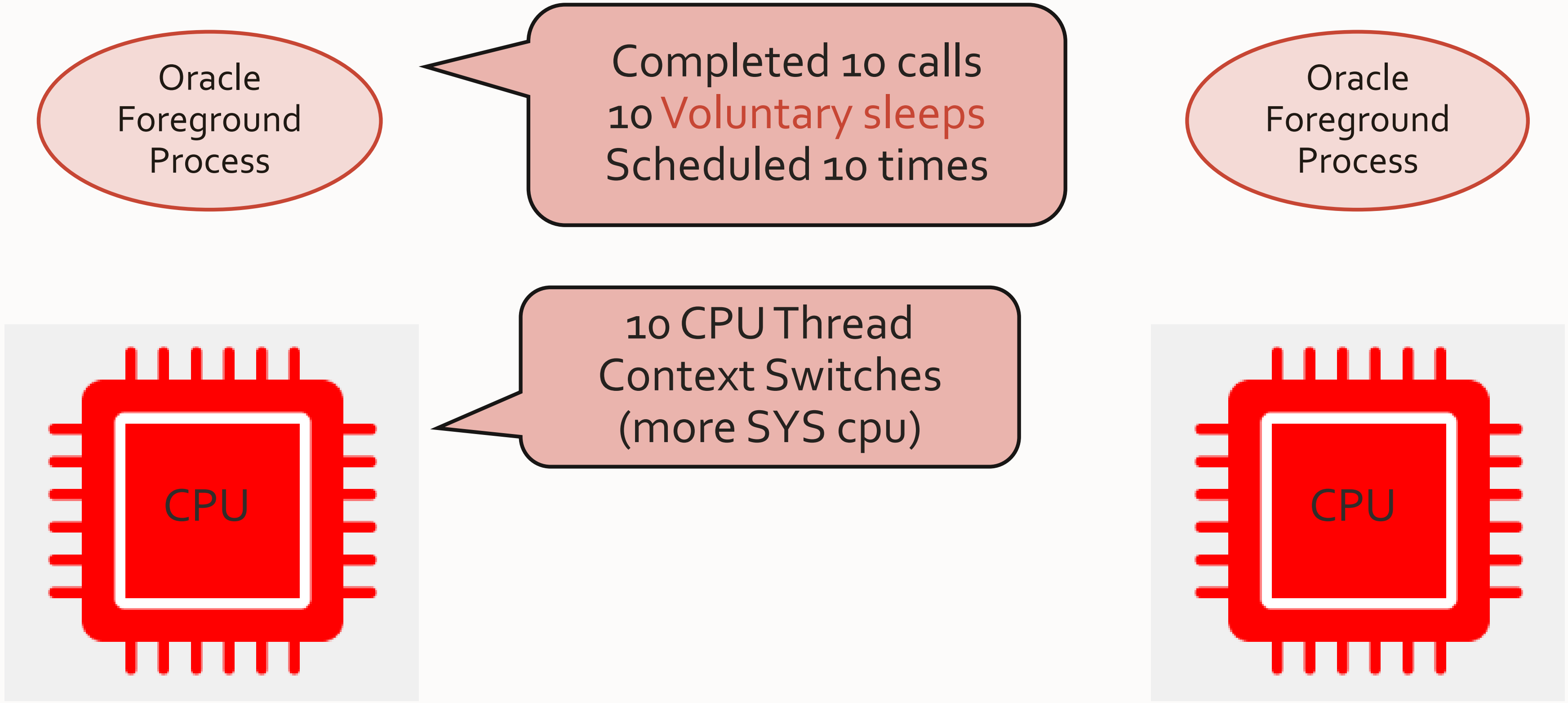


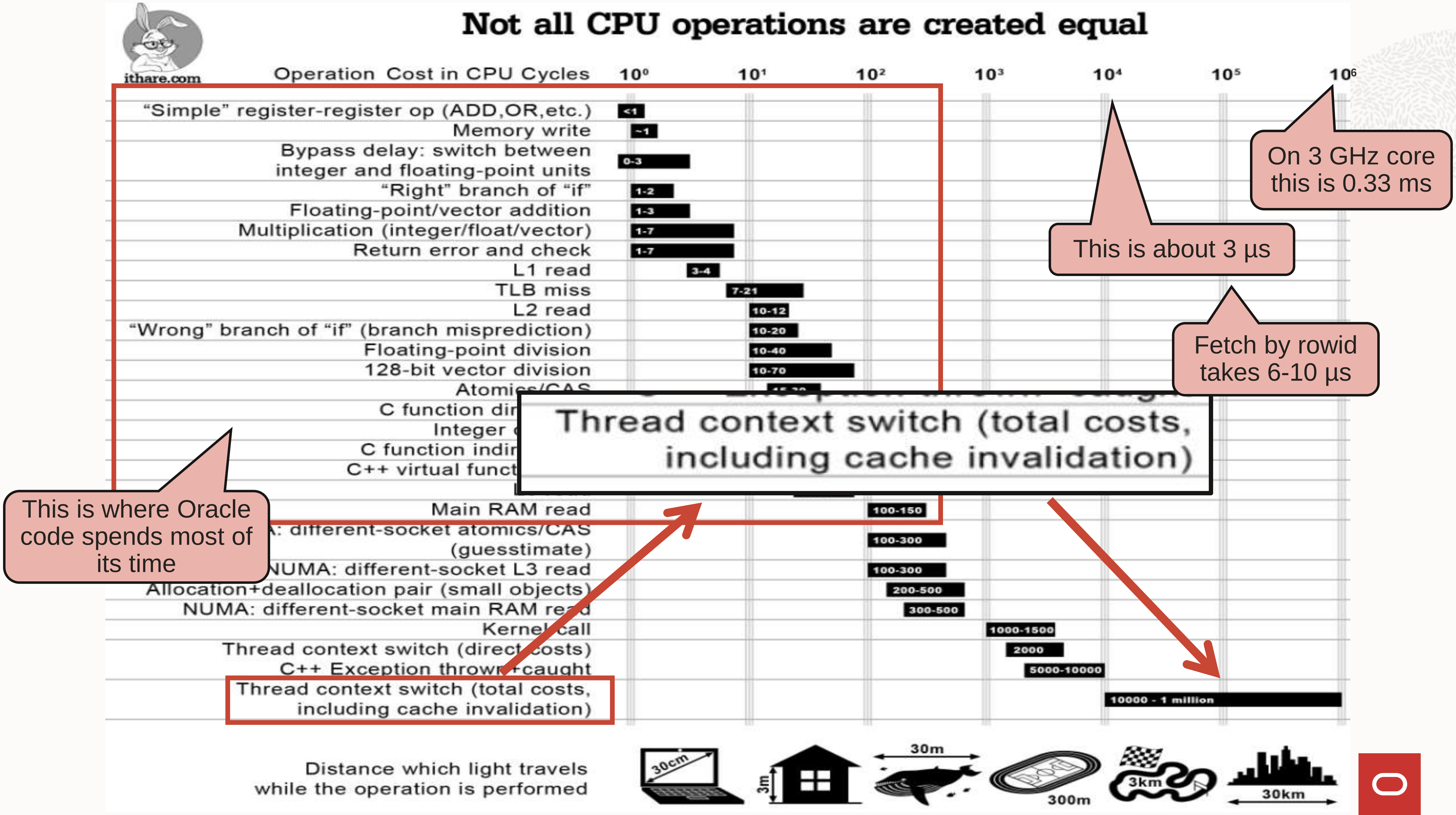
2: Stay on CPU, not Go to Sleep Voluntarily a Lot

Insert 10 rows

Ten database calls ← row-by-row

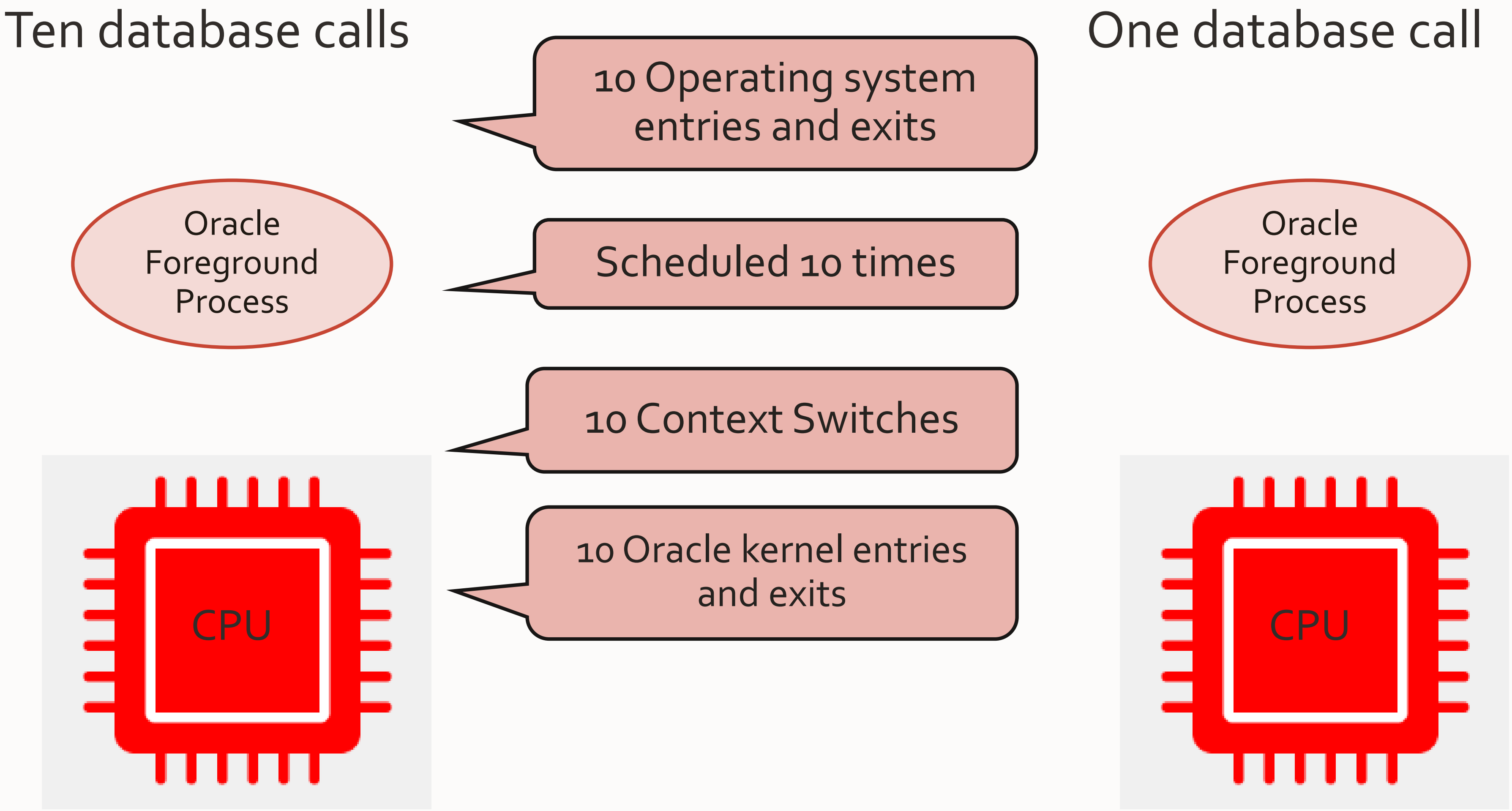
Array-based or stored proc → One database call





Not all CPU operations are created equal
ithare.com
Operation Cost in CPU Cycles
10⁰ 10¹ 10² 10³ 10⁴ 10⁵ 10⁶
"Simple" register-register op (ADD,OR,etc.) — <1
Memory write — ~1
Bypass delay: switch between integer and floating-point units — 0-3
"Right" branch of "if" — 1-2
Floating-point/vector addition — 1-3
Multiplication (integer/float/vector) — 1-7
Return error and check — 1-7
L1 read — 3-4
TLB miss — 7-21
L2 read — 10-12
"Wrong" branch of "if" (branch misprediction) — 10-20
Floating-point division — 10-40
128-bit vector division — 10-70
Main RAM read — 100-150
(guesstimate) — 100-300
NUMA: different-socket L3 read — 100-300
Allocation+deallocation pair (small objects) — 200-500
NUMA: different-socket main RAM read — 300-500
Kernel call — 1000-1500
Thread context switch (direct costs) — 2000
C++ Exception thrown+caught — 5000-10000
Thread context switch (total costs, including cache invalidation) — 10000 - 1 million
Distance which light travels while the operation is performed
30cm
3m
30m
300m
3km
30km
Thread context switch (total costs, including cache invalidation)
On 3 GHz core this is 0.33 ms
This is about 3 µs
Fetch by rowid takes 6-10 µs
This is where Oracle code spends most of its time

2: Stay on CPU, not Go to Sleep Voluntarily a Lot



Oracle Kernel Entry and Exit

- Like a thread context switch, this too is measurable overhead
  If your calls are small and fast

Oracle Kernel Entry and Exit

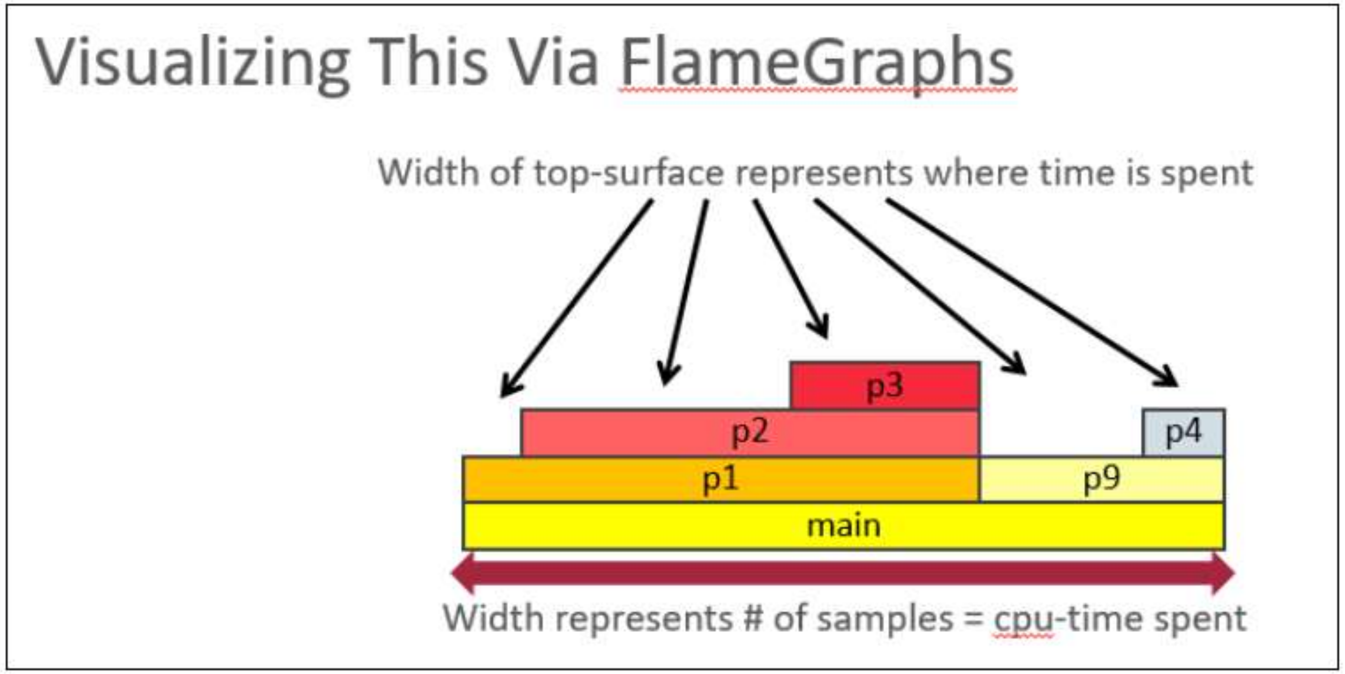
- We measured this in a lab environment

- In summary:
  - Simple batch process
  - Built in Java
  - Single threaded
  - Mix of single row selects, single row inserts and single row deletes
  - All issued (sequentially) over JDBC to database
  - Generating 9000 calls per second

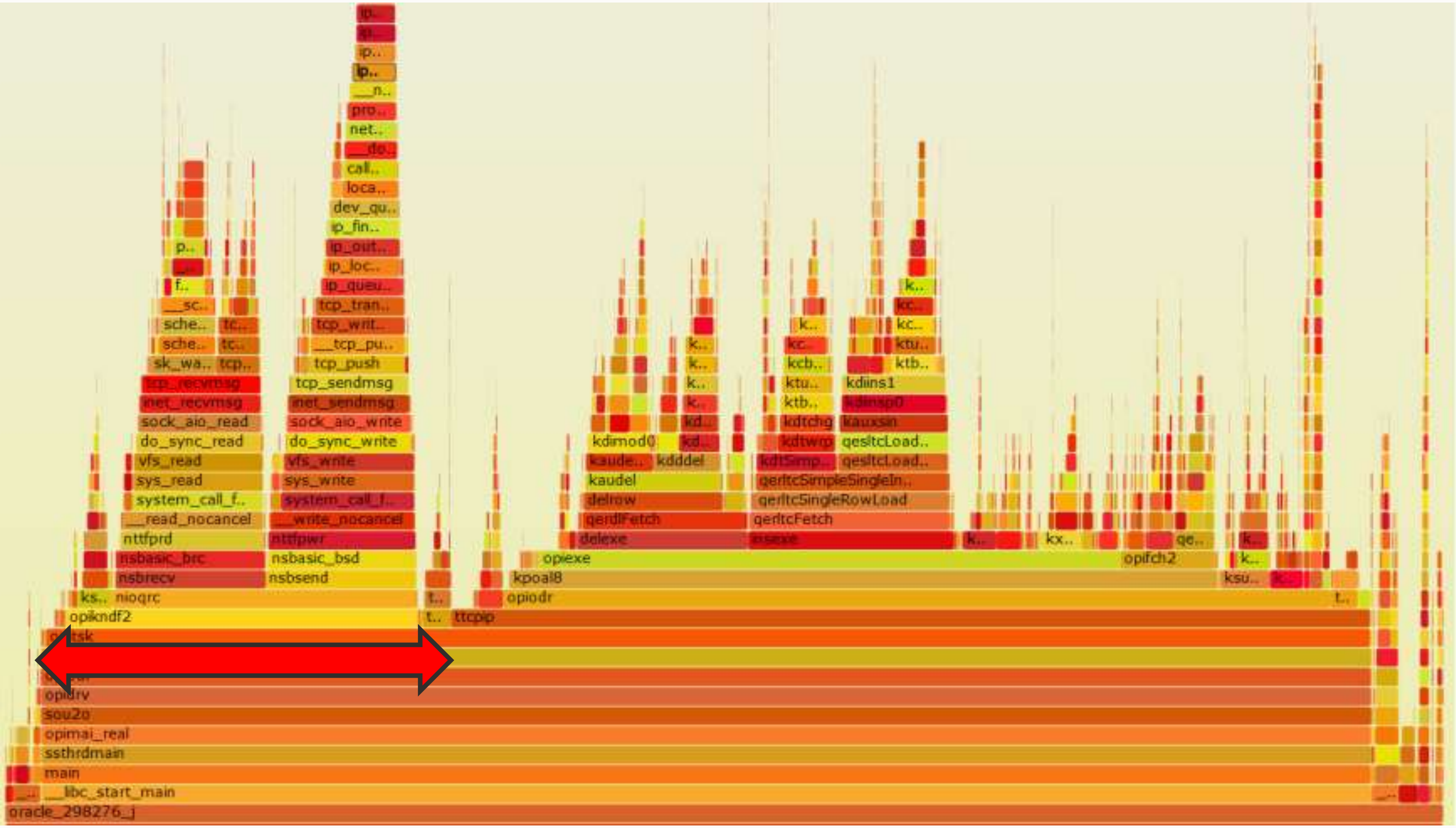
- Using "perf", we profiled foreground process to see where CPU time is spent

Visualizing CPU Usage With Flamegraph

- Perf samples n times per second, the call-stack of your process' execution thread
- Flamegraph visualizes this



Oracle Kernel Entry and Exit



Here, third of USER cpu is per-call overhead

How to Spot This in AWR Report

Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
DB CPU		28,2K		96.6	
log file sync	1,248,495	1244,9	1.00	4.3	Commit

Load Profile

	Per Second
DB Time(s):	19.1
DB CPU(s):	18.5
Background CPU(s):	0.1
Redo size (bytes):	5,900,374.3
Logical read (blocks):	1,165,406.1
Block changes:	71,814.5
Physical read (blocks):	77.6
Physical write (blocks):	707.4
Read IO requests:	77.0
Write IO requests:	302.9
Read IO (MB):	0.6
Write IO (MB):	5.5
IM scan rows:	0.0
Session Logical Read IM:	
User calls:	43,016.0
Parses (SQL):	11,703.7
Hard parses (SQL):	0.1
SQL Work Area (MB):	664.5
Logons:	1.3
Executes (SQL):	30,872.0
Rollbacks:	0.0
Transactions:	840.6

SQL*Net roundtrips to/from client	48,911,464

Time Model Statistics

- DB Time represents total time in user calls
- DB CPU represents CPU time of foreground processes

…oreground and background proces…
…"background" measure backgroun…
…descending order, followed by Stat…

DB CPU >>
sql execute elapsed time

Statistic Name	Time (s)	% of DB Time
DB CPU	28,214.31	96.64
sql execute elapsed time	17,972.94	61.56
PL/SQL execution elapsed time	1,384.76	4.74
parse time elapsed	572.33	
sequence load elapsed time	10.84	
hard parse elapsed time	8.28	0.03

10,000 seconds of USER cpu
spent on entry/exit

How to Spot This in AWR Report

SQL ordered by CPU Time

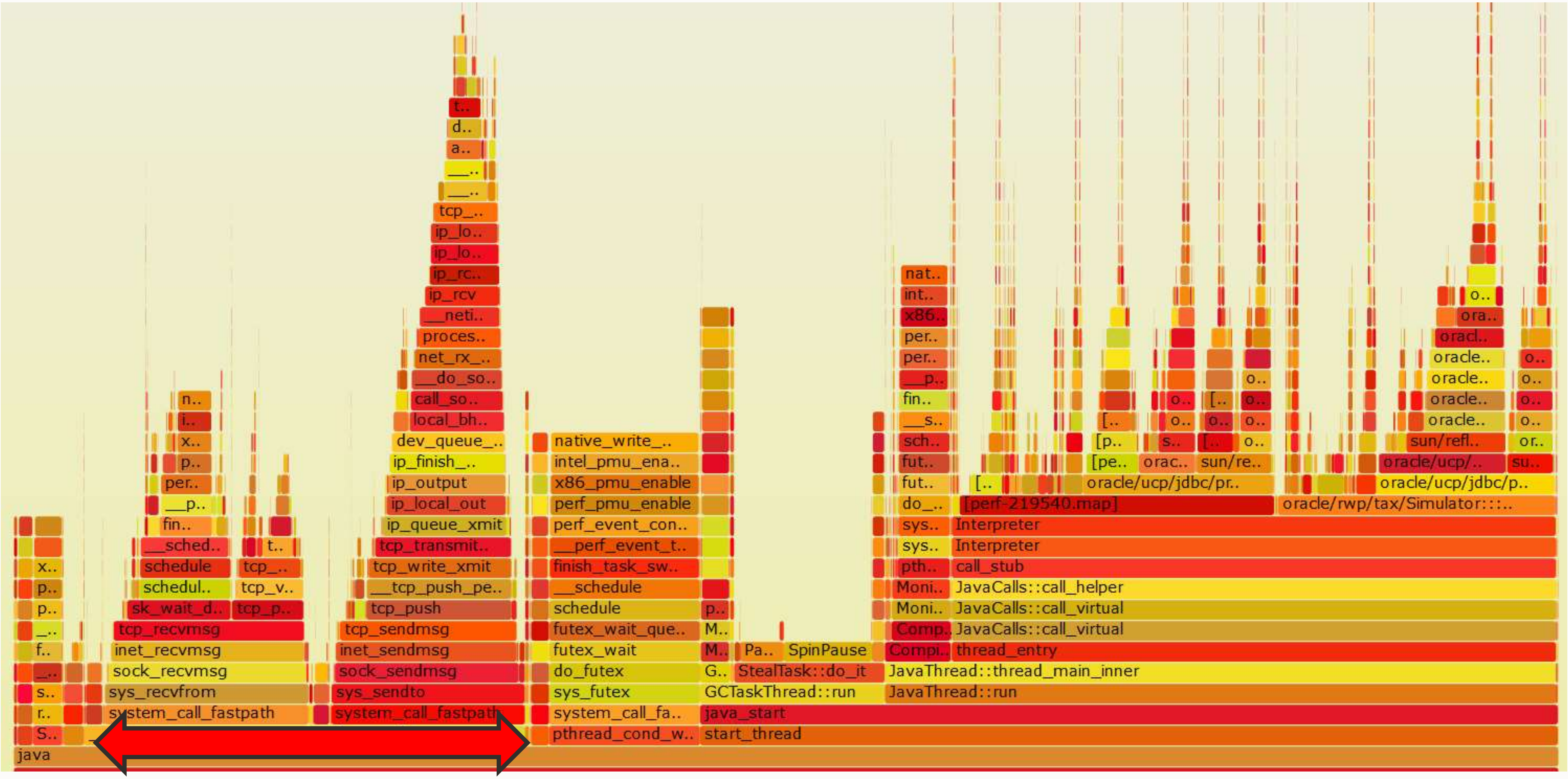
- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the co
- %Total - CPU Time as a percentage of Total DB CPU
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Captured SQL account for 39.1% of Total CPU Time (s): 28,214
- Captured PL/SQL account for 13.5% of Total CPU Time (s): 28,214

CPU Time (s)	Executions	CPU per Exec (s)	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Text
4,070.52	131,437	0.03	14.43	4,287.48	94.94	2.96	d2qrhp1kjtbxq	with claim_line_procedure_grou...
3,640.17	54,187	0.07	12.90	3,759.08	96.84	1.94	3jp1f1had5wxt	BEGIN pri_pprc_selection_pkg.c...
2,179.45	73,025	0.03	7.72	2,287.88	95.26	2.53	f6qcrhvva8nyh	with claim_line_procedure_grou...
742.84	54,193	0.01	2.63	748.79	99.21	0.00	dytzdjh1sdgr1	insert into pri$pprc_sel_proc1...
581.84	24,892	0.02	2.06	601.68	96.70	2.97	ddzqjy884hq5f	insert into pri$pprc_selectio...
312.81	49,410	0.01	1.11	353.63	88.46	0.00	cvfhbrhh6syu3	select t1.owner, t1.name, t1.q...
287.22	12,451	0.02	1.02	308.63	93.06	6.58	7dbwvdjb4af8m	insert into pri$pprc_selectio...
224.89	108,256	0.00	0.80	256.17	87.79	0.00	578aa1kxbhufu	select coch.* from rcl_combina...
195.68	425,789	0.00	0.69	250.85	78.00	0.00	b83y2q3dnbz61	SELECT * FROM (SELECT a.*, ROW...
187.06	1,036,606	0.00	0.66	355.82	52.57	0.00	1truyz26hjms5	SELECT ID, DYN_CHAR_001, DYN_C...

Do not sum up to 100%, while we are cpu bound

This Overhead Also on Application Servers!

Every process has this overhead!



Full Story Here

- Youtube video: https://www.youtube.com/watch?v=8jiJDflpw4Y
  "Koppelaars database"

Status sofar,

- Process architectures have per call overheads:
  1. OS interrupt figuring out which process to start (sys)
  2. Scheduler doing its work (sys)
  3. On-chip microcode execution to re-instantiate process state (sys)
  4. Entering Oracle code (user)

Database Performance Core Principles

Oracle instance = Process based architecture:

Foreground process is serving your database calls

To perform efficiently:

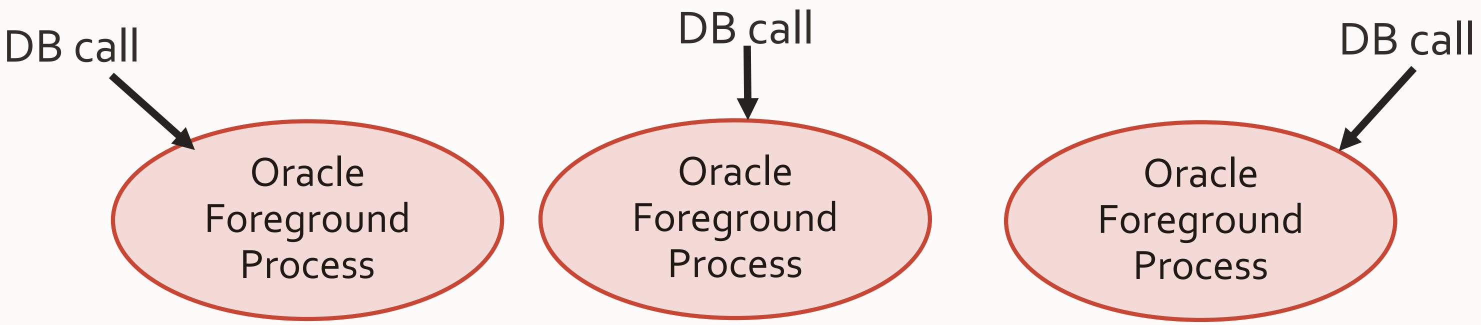
1. process needs to get on CPU as quickly as possible

Process needs to stay on CPU as long as possible:

2. process shouldn't go to sleep voluntarily a lot
3. process should experience as few involuntary sleeps as possible

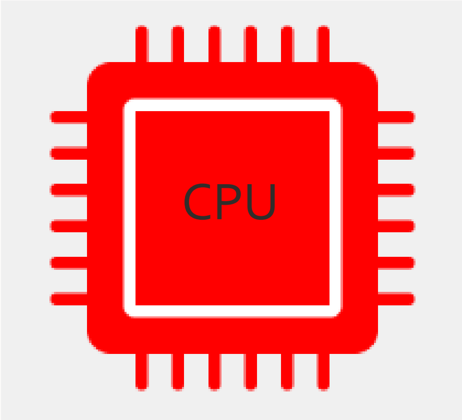
3: Stay on CPU, minimal Number of Involuntary Sleeps

Too many processes



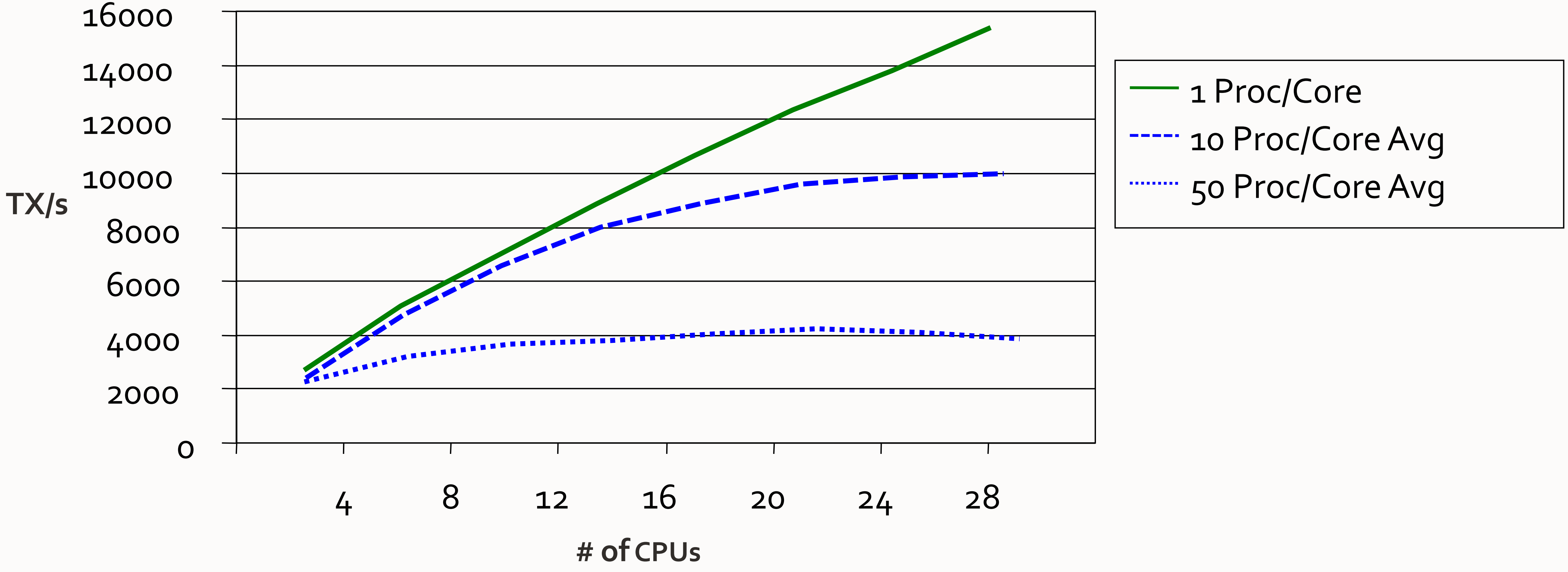
Three processes all want to insert 10 rows
(array-based or stored procedure)

Better to have one process insert 30 rows



OS scheduler **de-schedules** processes while they still have more CPU work to do = **involuntary context switch (sleep)**

Impact of Too Many Processes



ORACLE
REAL-WORLD PERFORMANCE

Database Performance Core Principles

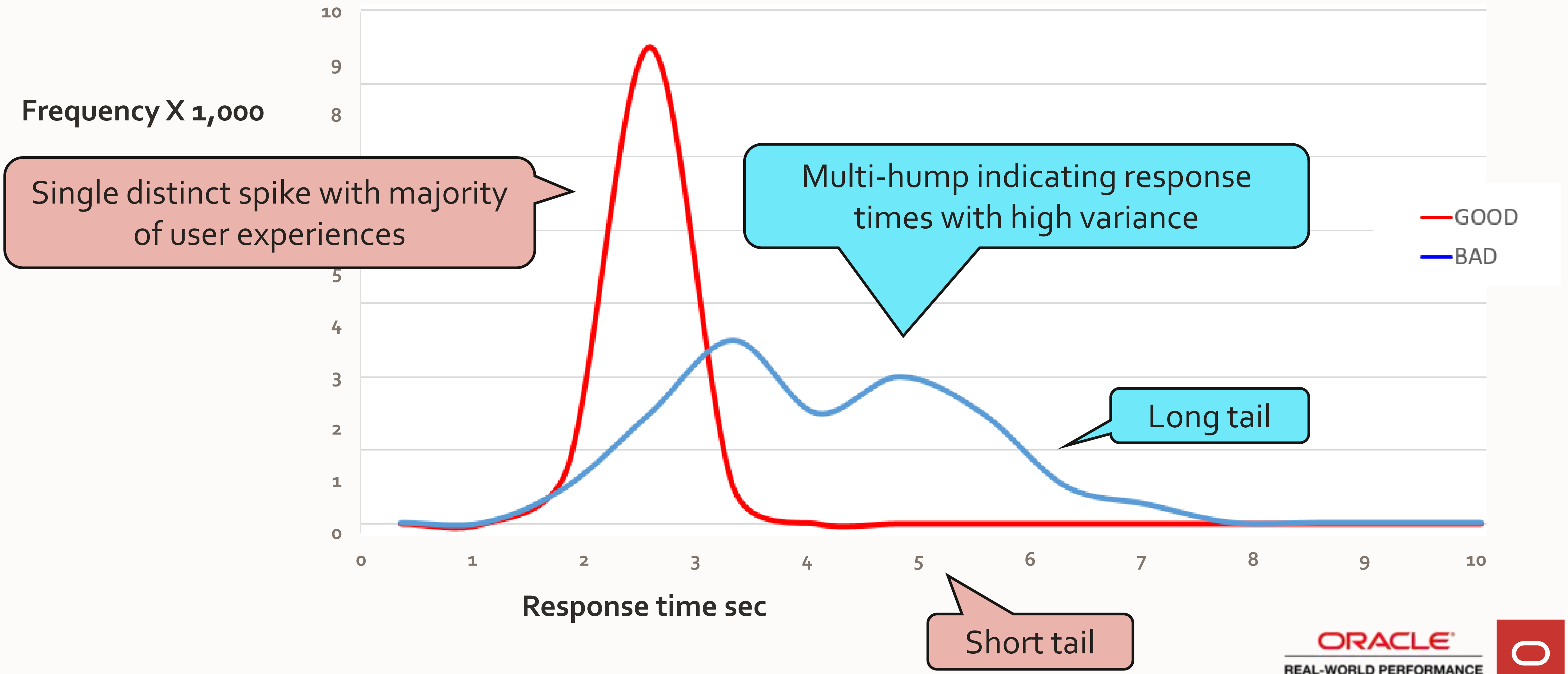
Database Performance Core Principles

- Implications of Oracle's Process Based Architecture
- User Response Time, Throughput and DB Time

User Response Time

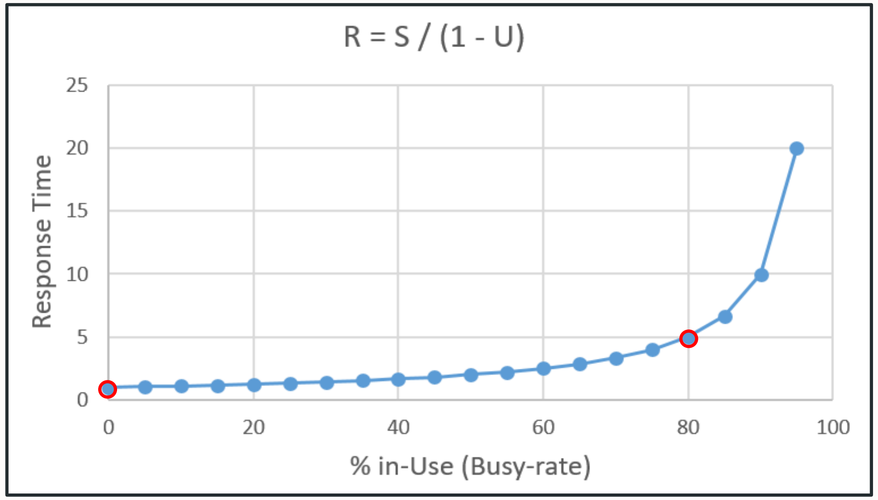
- Actual duration is measure of performance quality (shorter = better)
- Consistency of response time is an equally important measure of performance quality
- Variance in response times will not delight your users

Response Time Experienced by User

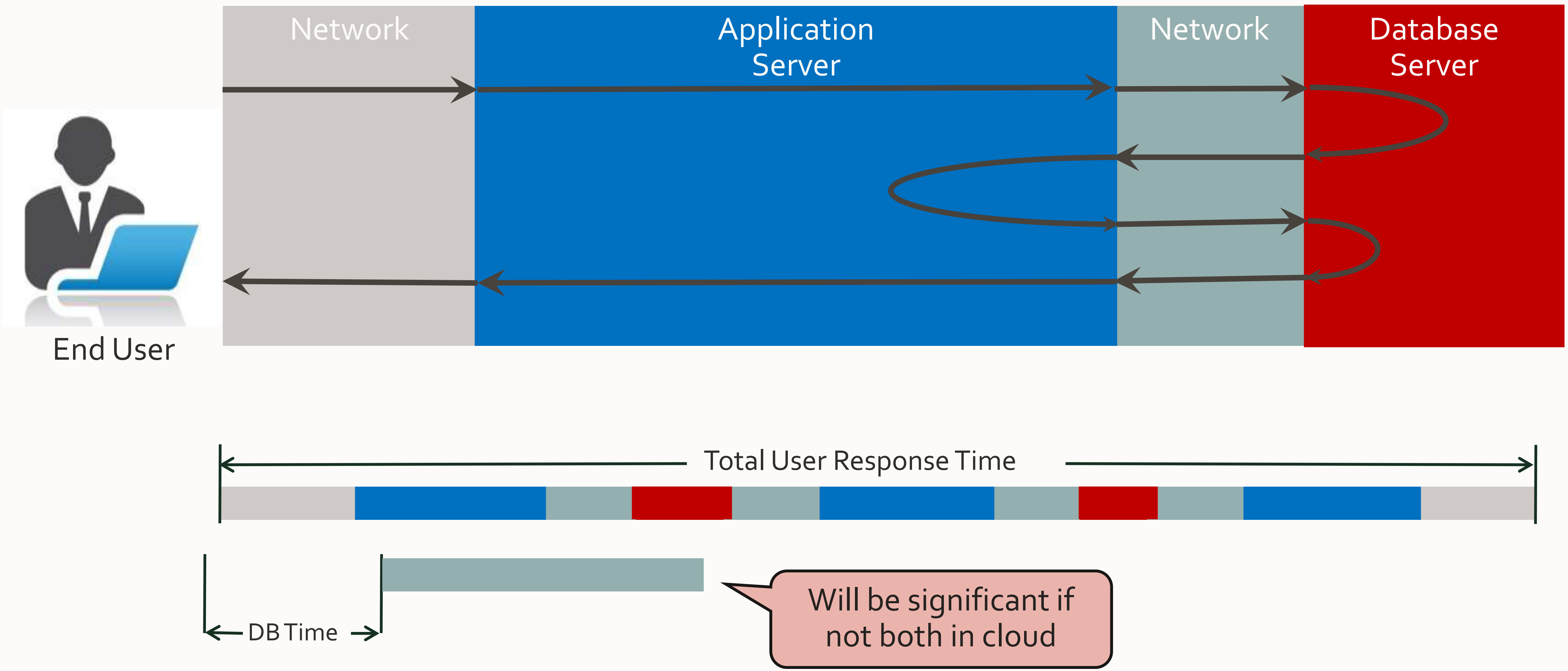


Throughput

- Throughput is the number of units of work processed within a period of time
- Throughput is not response time
- As the system gets busier, throughput increases (good), but the response time for each individual unit of work will also increase (bad)
- If you've witnessed throughput X at 40% busy, don't expect throughput 2*X at 80% busy

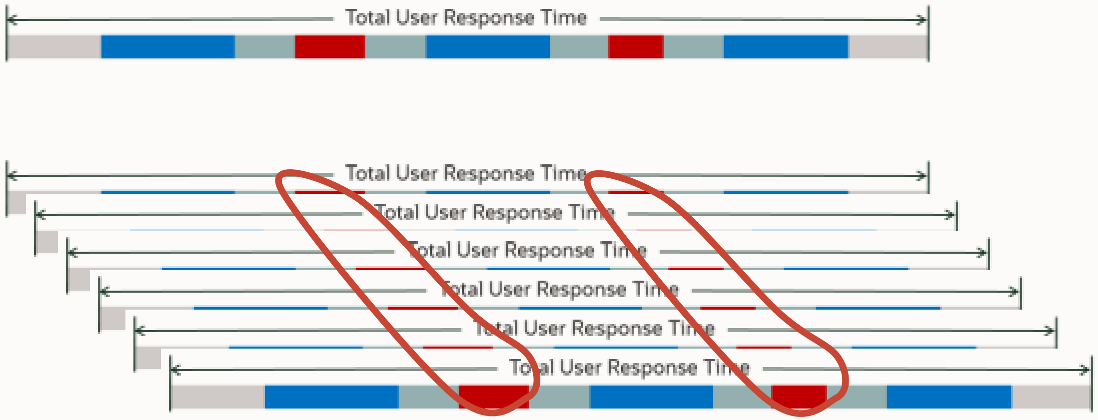


Response Time Versus DB Time



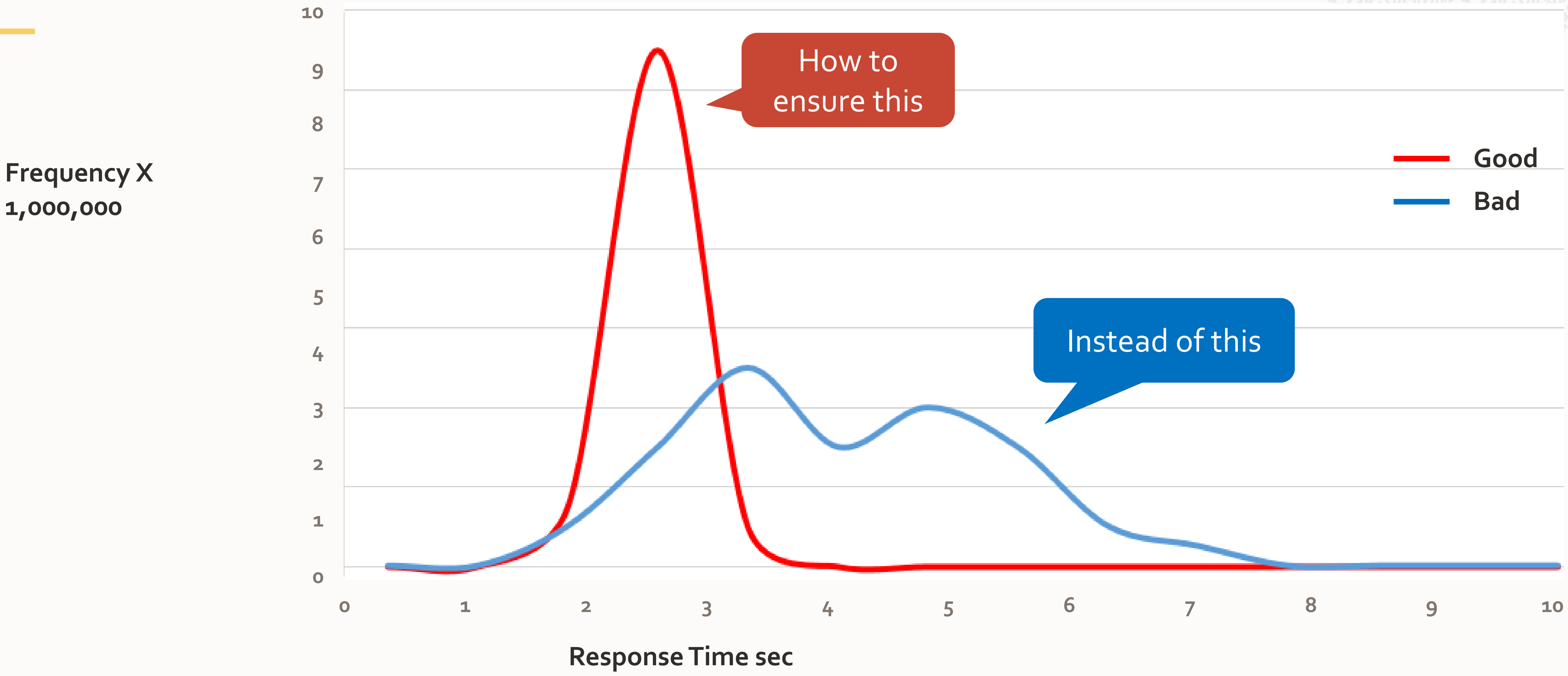
Remember

- Response time
- Throughput
- DB Time

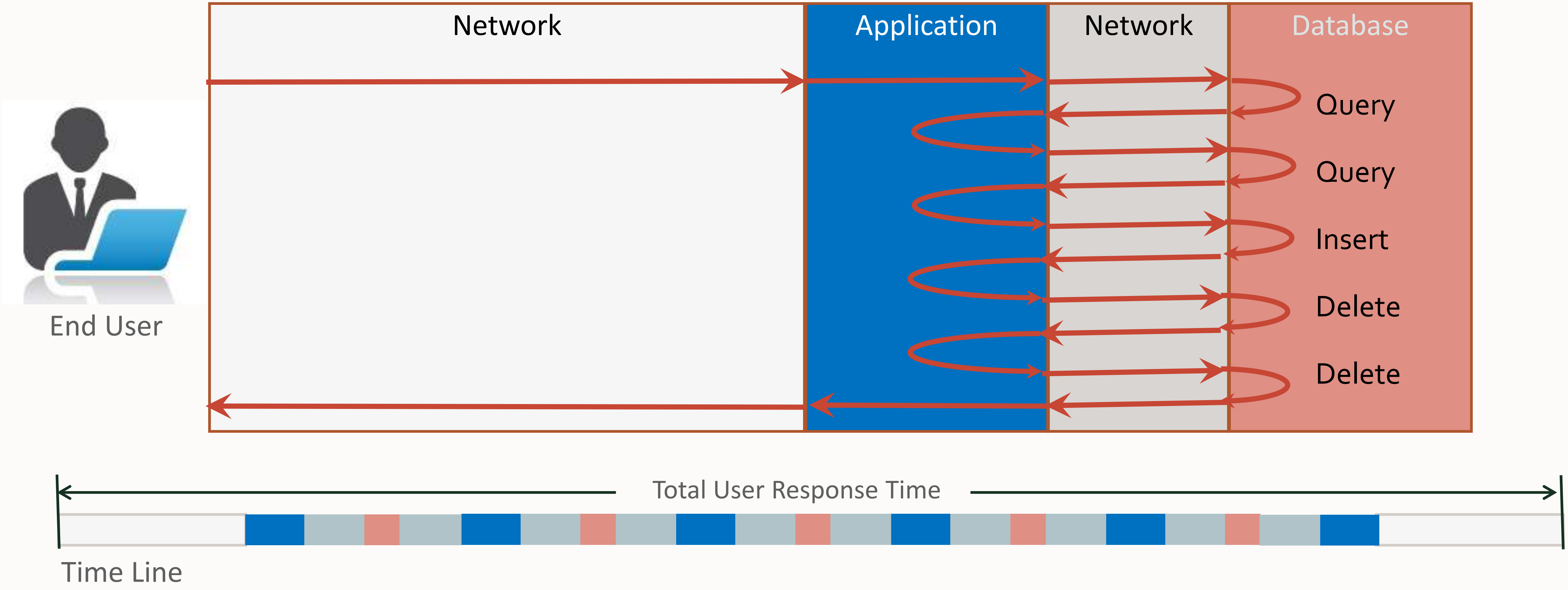


Three different things

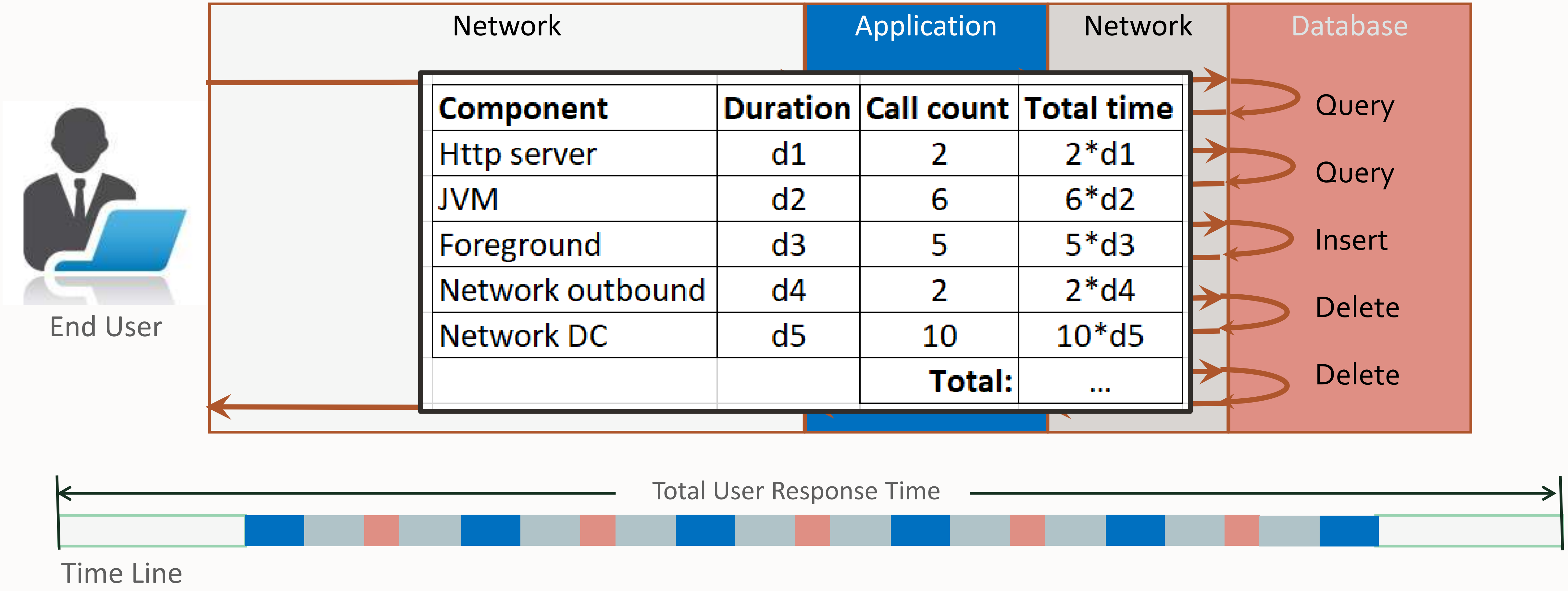
Response Time Core Principle



User Experience Response Time: Breakdown



User Experience Response Time: Breakdown



Component	Duration	Call count	Total time
Http server	d1	2	2*d1
JVM	d2	6	6*d2
Foreground	d3	5	5*d3
Network outbound	d4	2	2*d4
Network DC	d5	10	10*d5
		Total:	…

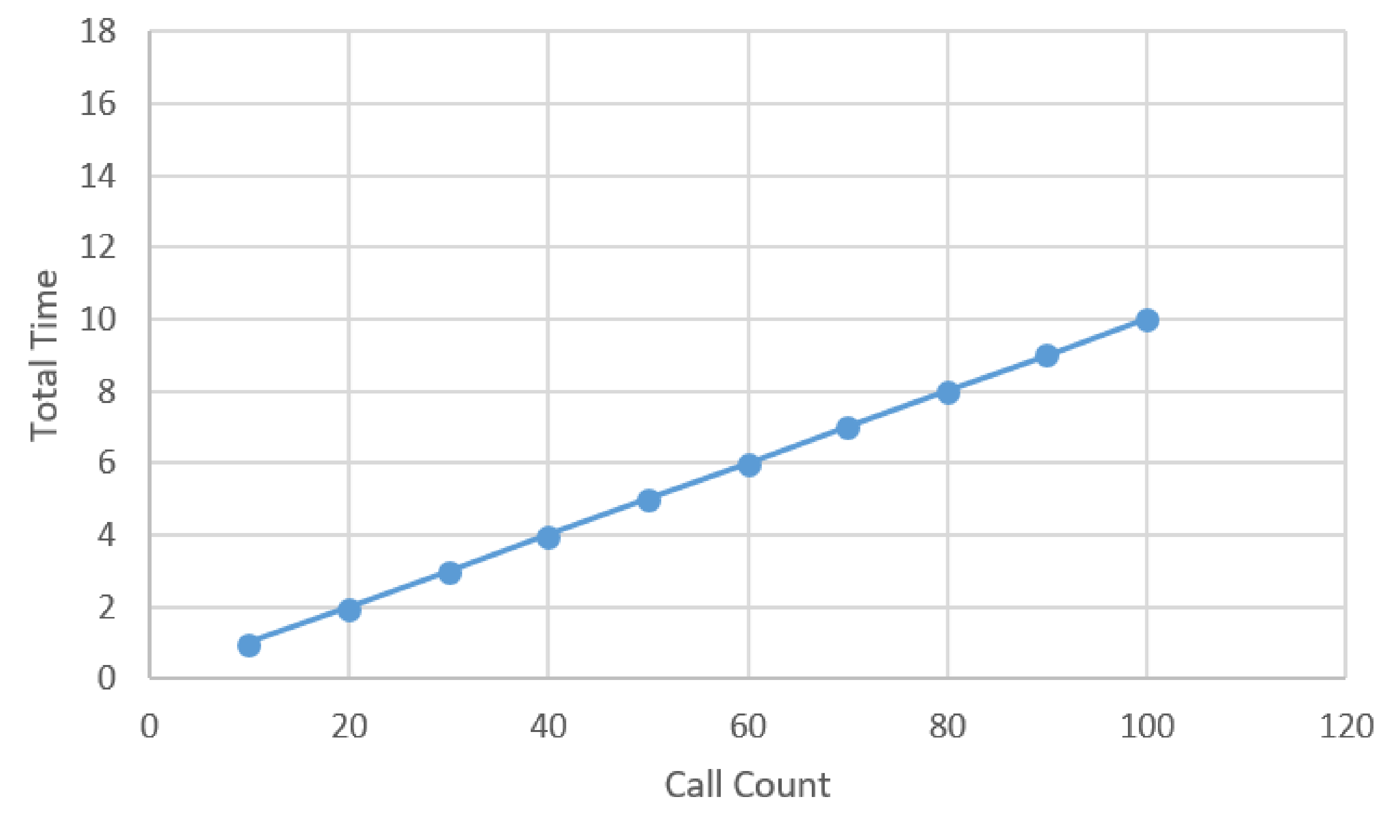
User Experience Response Time: Variance

Again: %-Busy and Call Counts matter

Breakdown of response time:

Component	Duration	Call count	Total time
Http server	d1	2	2*d1
JVM	d2	6	6*d2
Foreground	d3	5	5*d3
Network outbound	d4	2	2*d4
Network DC	d5	10	10*d5
		Total:	...

Linear relation?



ORACLE
REAL-WORLD PERFORMANCE

Ensuring Single Spike

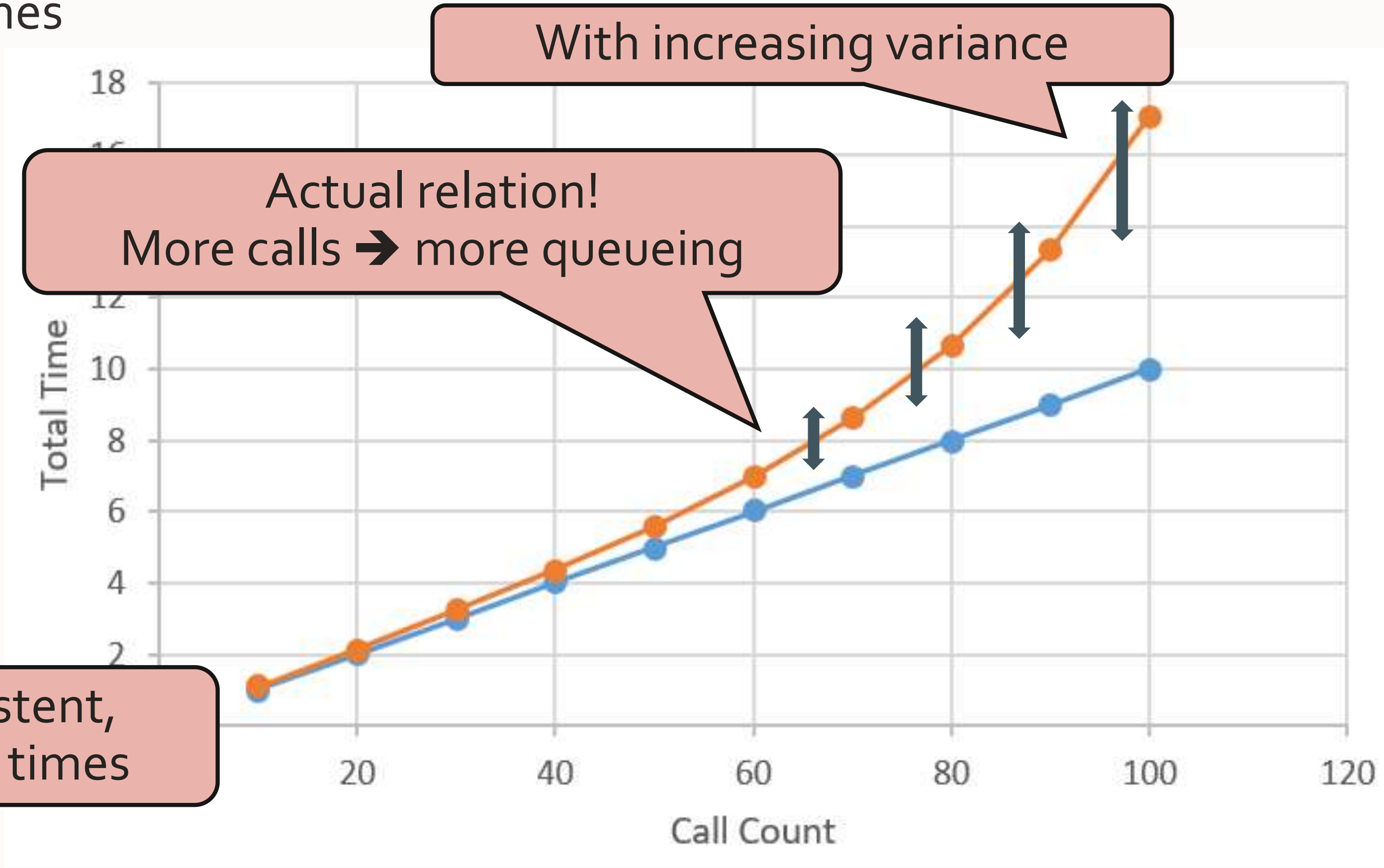
How to prevent multi-hump response times

Breakdown of response time:

Component	Duration	Call Count	Total Time
A1	x1	c1	x1 * c1
A2	x2	c2	x2 * c2
A3	x3	c3	x3 * c3
A4	x4	c4	x4 * c4
A5	x5	c5	x5 * c5
		Total:	………

$$\sum_{i=1}^{n} c_i$$

Smaller = More consistent, single spike, response times



DB Performance Core Principles: Summary

The Oracle database is a process based architecture

To perform efficiently each process requires:

1. to get on CPU asap, and stay on CPU
2. to not go to sleep voluntarily a lot
3. to experience as few involuntary sleeps as possible

You want to stay well clear of CPU's being 100% busy

Better to call database with more work than to call with little work

Use architecture that requires small number of processes

Consistent response times require minimization of call counts across components (processes)

So, If We See These…

High Call Counts per Second

Instance Activity Stats

- Ordered by statistic name

Statistic	Total	per Second	per Trans
SQL*Net roundtrips to/from client	290,410,469	23,473.70	12.10

High number of voluntary sleeps per second

Load Profile

	Per Second
DB Time(s):	30.5
DB CPU(s):	9.2
Background CPU(s):	1.2
Redo size (bytes):	16,944,137.9
ad (blocks):	332,564.8
nges:	88,414.9
ead (blocks):	1,059.0
rite (blocks):	3,247.4
equests:	1,032.0
equests:	1,872.5
MB):	8.3
Write IO (MB):	25.4
IM scan rows:	0.0
Session Logical Read IM:	0.0
Global Cache blocks received:	2,960.8
Global Cache blocks served:	2,771.9
User calls:	50,490.2
Parses (SQL):	16,556.7
Hard parses (SQL):	37.8
SQL Work Area (MB):	26.2
Logons:	0.7
Executes (SQL):	21,400.0
Rollbacks:	29.4
Transactions:	1,939.9

Little Work per Call (Row-by-Row Processing)

SQL ordered by Executions

- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Total Executions: 264,754,563
- Captured SQL account for 51.3% of Total

Executions	Rows Processed	Rows per Exec	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	
4,428,782	4,416,496	1.00	7,839.34	12.9	7.1	42v88dga8b6ym	JDBC Thin Client	SELECT
3,984,410	3,984,402	1.00	5,939.65	17.3	.7	fq1j38zxwk35x	JDBC Thin Client	UPDATE
3,644,892	3,644,799	1.00	2,063.83	30.2	25.4	ajzh9q11a8wa5	JDBC Thin Client	INSERT
3,600,643	3,600,300	1.00	1,823.46	20.2	0	8k9s18v6c6g22	JDBC Thin Client	SELECT
3,410,561	3,410,467	1.00	1,718.03	28.6	.9	0pyjdmfrzmt6g	JDBC Thin Client	INSERT
3,154,501	3,154,312	1.00	913.85	36.8	2.2	c8dasr4jjdd5r	JDBC Thin Client	SELECT
3,140,508	3,140,114	1.00	494.44	37.9	0	430vvx7gpt6t3	JDBC Thin Client	SELECT
3,116,220	3,115,952	1.00	564.03	27.1	40.3	7phddk8yy5zk0	JDBC Thin Client	SELECT
3,077,007	3,077,004	1.00	3,076.86	16.8	.4	8yf3smfz8ubk5	JDBC Thin Client	UPDATE
2,929,431	2,929,430	1.00	2,009.00	23.2	1	3yyhw6ssbg6j1	JDBC Thin Client	UPDATE
2,929,413	2,929,401	1.00	3,124.07	19	.3	2f6fcbhhg478n	JDBC Thin Client	UPDATE
2,800,813	2,800,512	1.00	1,099.73	20	1	fsawc6c853385	JDBC Thin Client	SELECT

Not wanting to stay on CPU long

High Process-to-Core Ratio + Busy System

Host Name	Platform	CPUs	Cores
rac1.mycompany.aws	Linux x86 64-bit	40	20

	Snap Id	Snap Time	Sessions
Begin Snap:	209	17-Apr-18 10:00:32	2112
End Snap:	213	17-Apr-18 13:26:44	2131

OS Scheduler doing what it does best: de-scheduling

Host Name	Platform	CPUs	Cores
xxxx.xxxx.xx.xxx	Linux x86 64-bit	88	44

	Snap Id	Snap Time	Sessions
Begin Snap:	93142	28-Aug-18 00:00:24	6218
End Snap:	93143	28-Aug-18 00:15:27	6233

We Pretty Much Know That

Despite lots of CPU being used, not much work is getting done

→ Majority of CPU time is spent on per-call overhead across all layers

With high busy rates on database and application servers

→ Response times probably aren't consistent

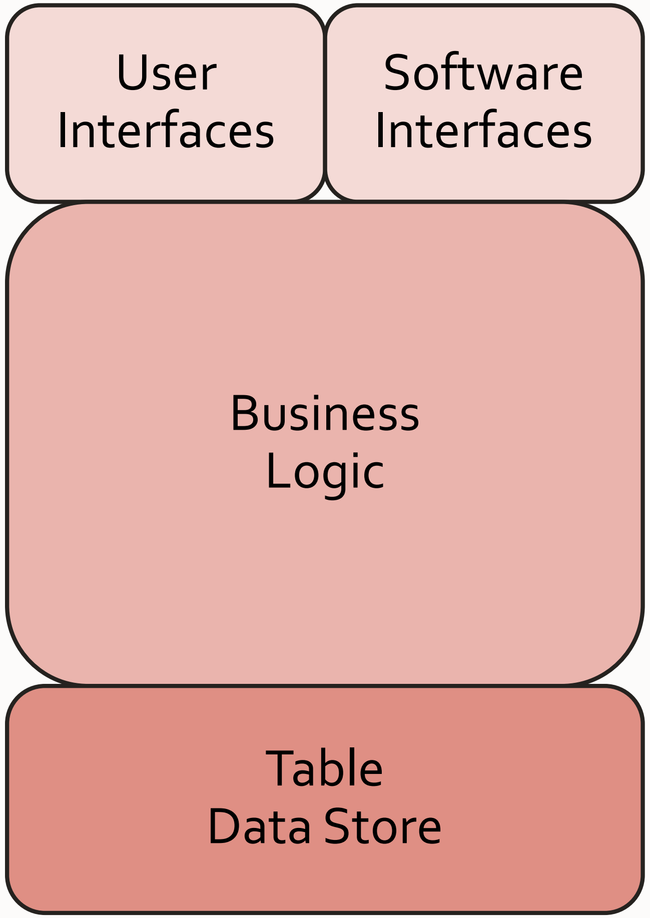
Last Topic…

What happened last two decades with application development?

- It fundamentally moved away from these principles

- We stopped using database as a processing engine

High Level Breakdown of OLTP Application



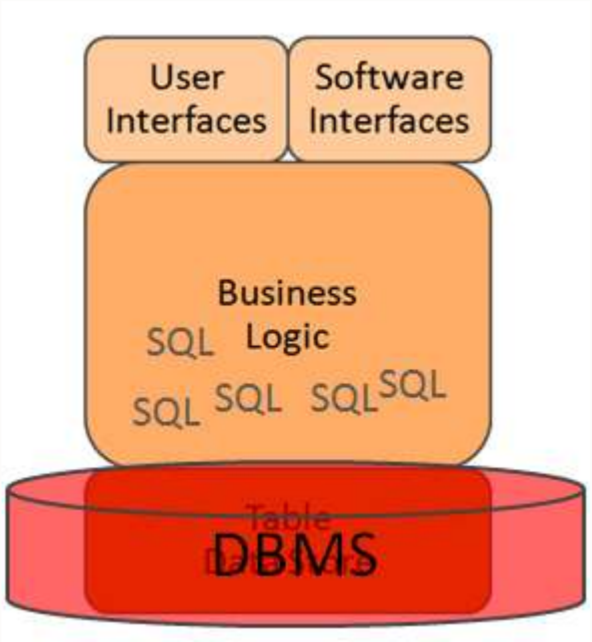
Conceptually 3 tiers

- Exposed functionality
  - GUI's for human interaction
  - REST, or otherwise, for software interaction

Internals

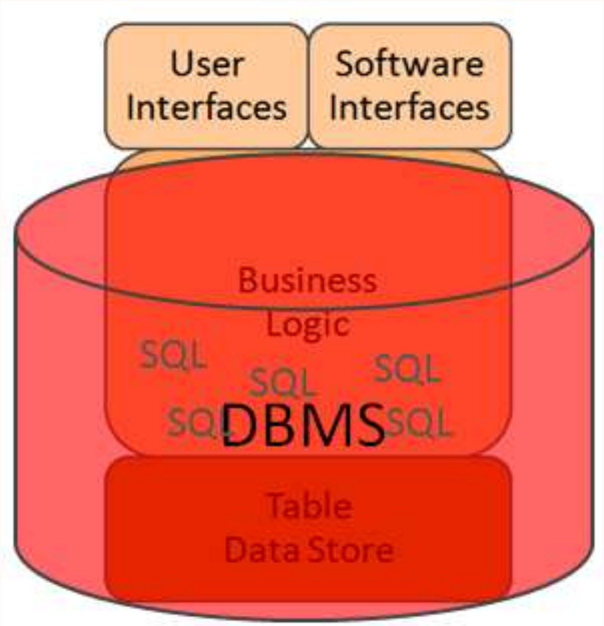
- Business logic
- Data store, relational database

Two Fundamentally Different Architectures



DBMS = Persistence Layer

"NoPlsql" Approach



DBMS = Processing Engine

"SmartDB" Approach

You'll Always See: Row-by-Row, Single Table Approach

```
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

Row-by-row updating (data access)

"if-then-else-loop" code PL/SQL here for convenience

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

You'll Never See This: Set-Based SQL

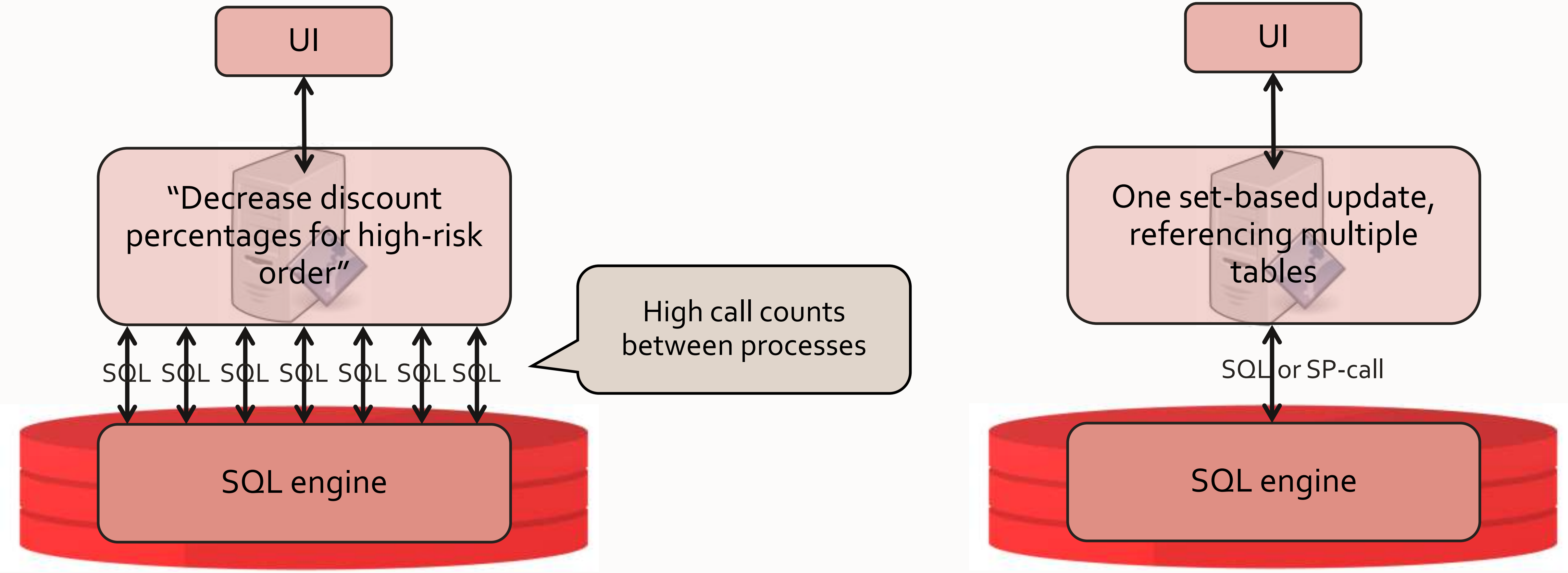
```
update ORDERLINES ol set ol.DISCOUNT = greatest(ol.DISCOUNT - 10, 0)
where ol.ORDER# = l_order#
  and ol.STATUS = 'OPEN'
  and exists(select 'high-risk!'
                from ORDERS o
               where o.ORDER# = l_order# and o.LIMIT > 2000)
```

Decrease the discount percentage on all (still open) order lines, of a given high-risk order.

References >1 table
Affects >1 row

Why not?
Because persistence frameworks aren't capable of generating this

“Chatty” Applications When Business Logic Outside DB



Two Fundamentally Different Architectures

Database as Bit Bucket

Aka #NoPlsql

All business logic outside database

"Layered Software Architectures"

- Many small calls to database leading to high rate of voluntary sleeps
- Noticeable per-call overhead, as call is single row statement
- Requires large number of FG-processes risking high rate of involuntary sleeps

Database as Processing Engine

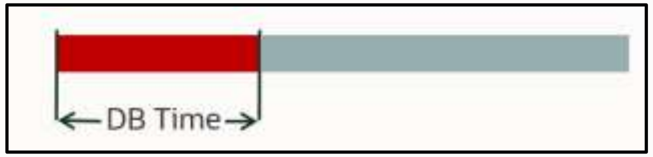
Aka #SmartDB

All business logic inside database

Only UI outside database

- One user experience = one database call, reducing voluntary sleeps
- Negligible per-call overhead, as call involves substantial amount of work
- Requires small number of FG-processes, reducing involuntary sleeps

Youtube: "asktom connection pool"

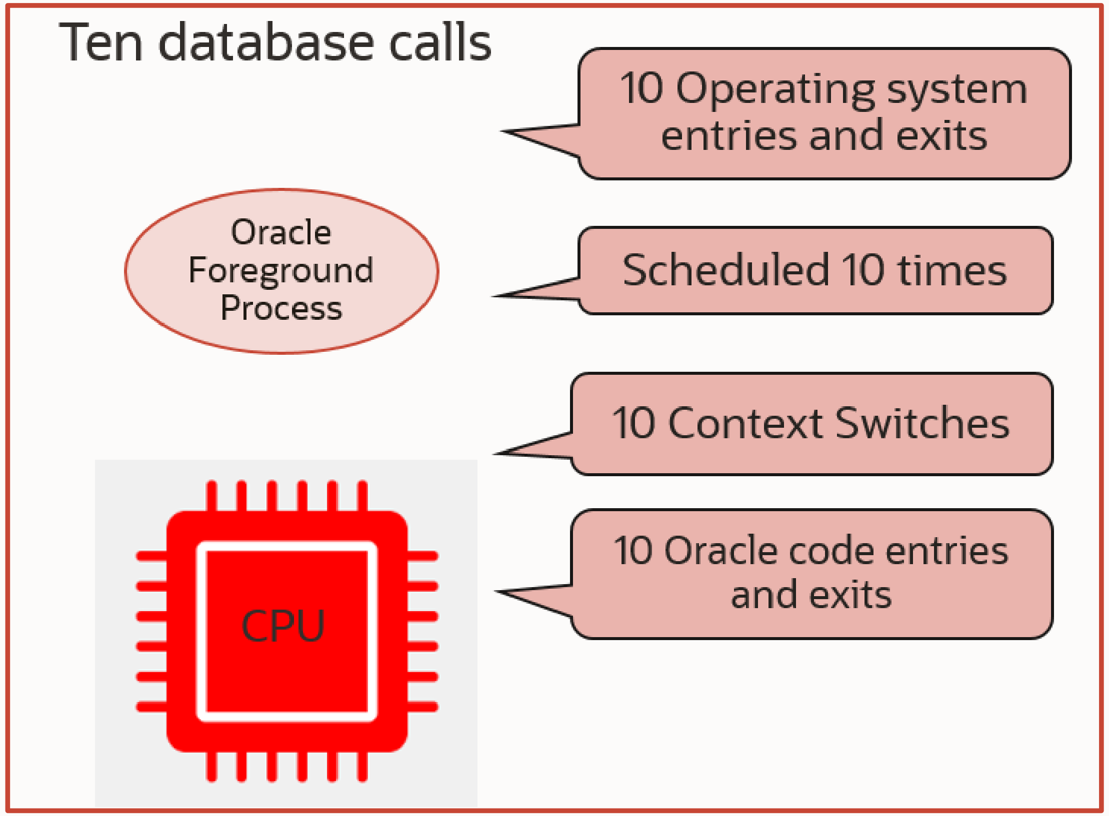


Implication For Your Computing Resources Footprint

Every SQL statement submitted from application server

- Requires OS entry
- Requires OS scheduling
- Requires CPU context switch
- Requires Oracle kernel entry

Before it arrives at SQL-engine

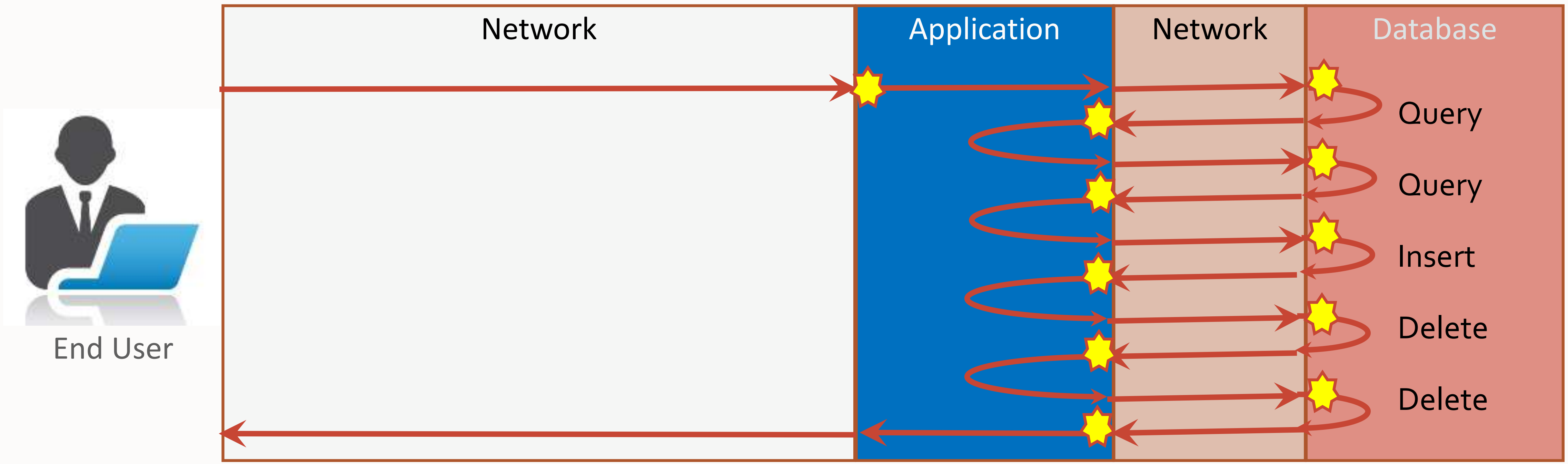


Sum of these fixed CPU overheads is very real for row-by-row SQL!

- Could be 2X CPU knock-on to SQL execution

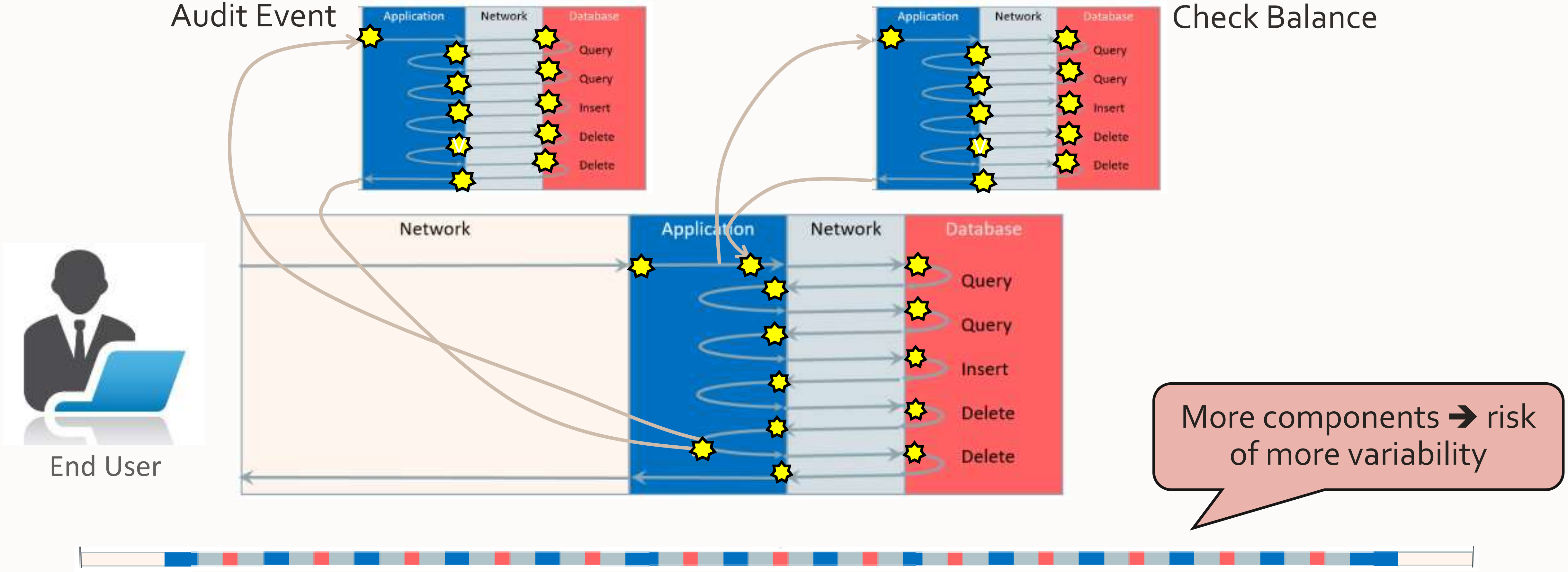
: Thread context switch

Implication For Your Footprint: Affects Application Server too



Not just on database server: on application server as well

Micro Services Architecture: Rife With Context Switches



Time Line: breakdown consists of many more components

Implication of Moving Business Logic Out of DBMS

Introduce ridiculous inefficient use of available CPU resources

We see database cores spending up to 60% of CPU-cycles
on OS and CPU context-switches and getting in/out database kernel

And application servers spending majority of time descending
and ascending up JDBC and framework layers

Only way to improve this ➔ move away from row-by-row processing

This requires hand-written set-based SQL

Want to Get an Idea of Inefficiency on Your Server?

Operating System Statistics - Det

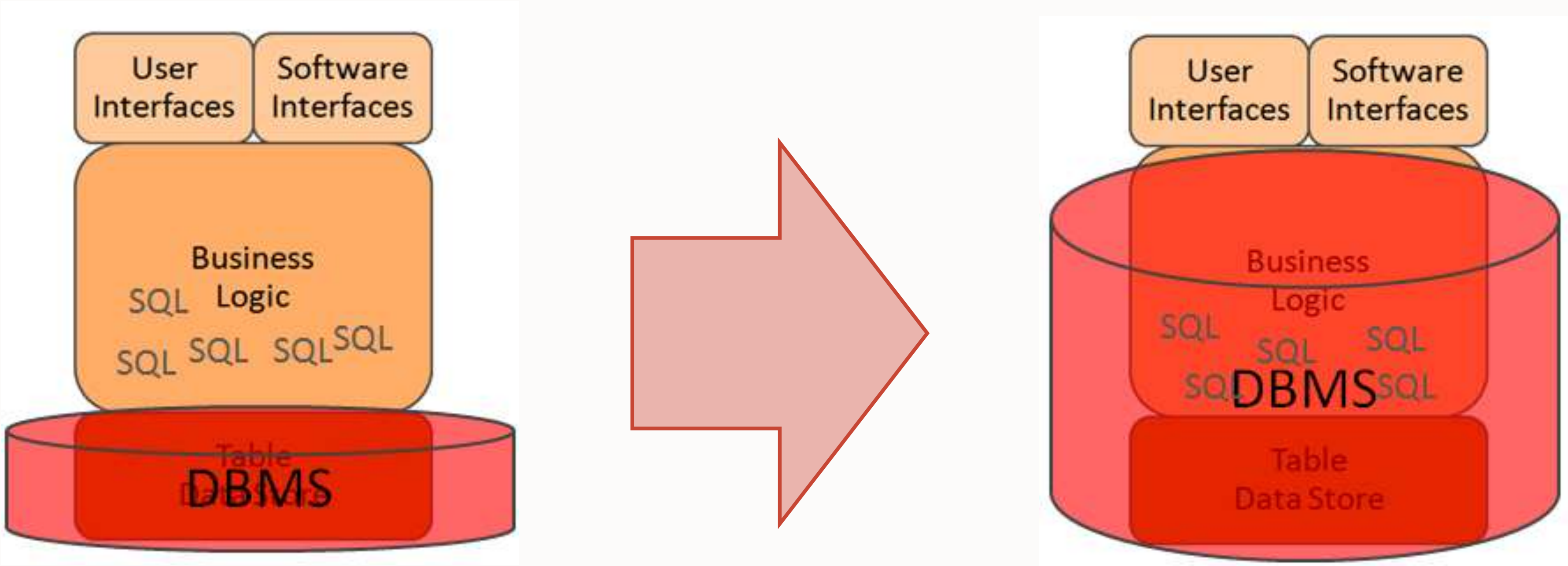
Snap Time	Load	%busy	%user	%sys	%idle	%iowait
16-Aug 14:00:27	104.09					
16-Aug 15:00:07	167.55	49.27	32.42	16.85	50.73	0.00

High Sys-to-User ratio is good indicator

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

You're not using DBMS as it was designed to be used

How to Reduce Your Computing Resources Footprint?



DBMS = Persistence Layer

More busy

DBMS = Processing Engine

Less busy

Here Is The, Counter Intuitive, Core Fact

The #SmartDB gains of:

- Massive reduction in call counts, greatly diminishes all aforementioned overheads per call
- Outweigh additional work you bring into database by implementing your business logic inside SQL and PL/SQL

In Summary

- Majority of current applications have large computing resources footprints
- Why? Because they were built using layered software architectures and have all business logic execute on application servers
- These architectures cause very high call counts across components (processes), and thereby they violate all database core performance principles

In Summary

- If you want to reduce your computing resources footprint
  ➔ Don't bring data to code, bring code to data, use smart SQL

- Focus on minimizing chattiness, process context switches, call-counts, etc.

- You'll reduce both your database-server footprint, as well as your application-server footprints

- And your user experiences will be consistent

- Questions: email Toon.Koppelaars@oracle.com, twitter: @toonkoppelaars