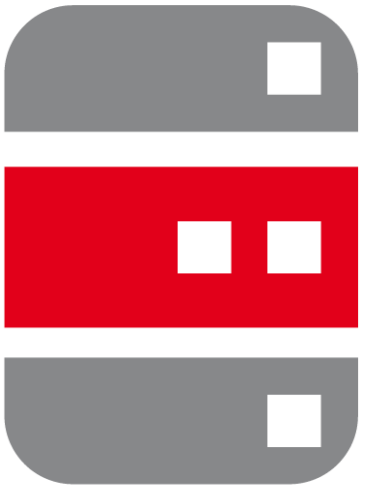
performing databases



Your reliability. Our concern.

performing databases
Your reliability. Our concern.

Full Throttle: Oracle SQL Tuning & the Ressource Consumption Approach

Martin Klier 

Performing Databases GmbH
Mitterteich / Germany

performing databases
Your reliability. Our concern.

The Self-Driving Database

or

„What We Can Learn From Speedy (SPD-13)“

Planet Mercury

Day: +427°C (800 F)
Night: -173°C (-279 F

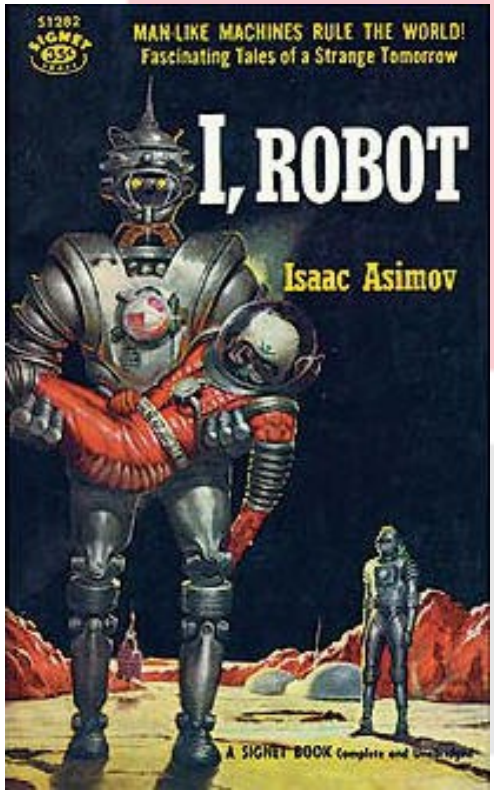
1 day on Mercury
= ½ year on Earth

1) A robot may not injure a human being
or, through inaction, allow a human being to come to harm.

2) A robot must obey the orders given it by human beings
except where such orders would conflict with the First Law.

3) A robot must protect its own existence
as long as such protection does not conflict with the First or Second Laws.

Isaac Asimov, „Runaround“, 1942

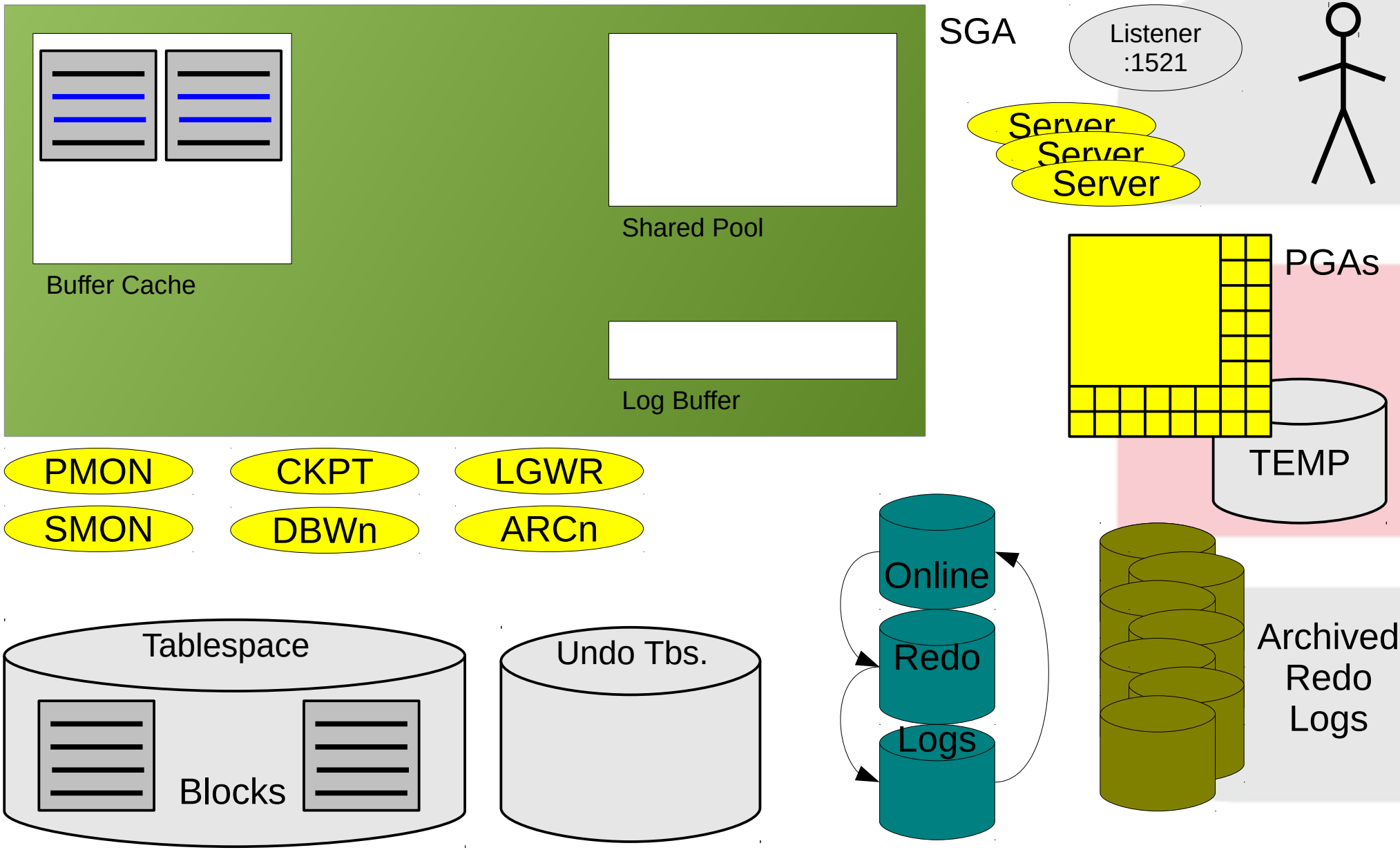


performing databases
Your reliability. Our concern.

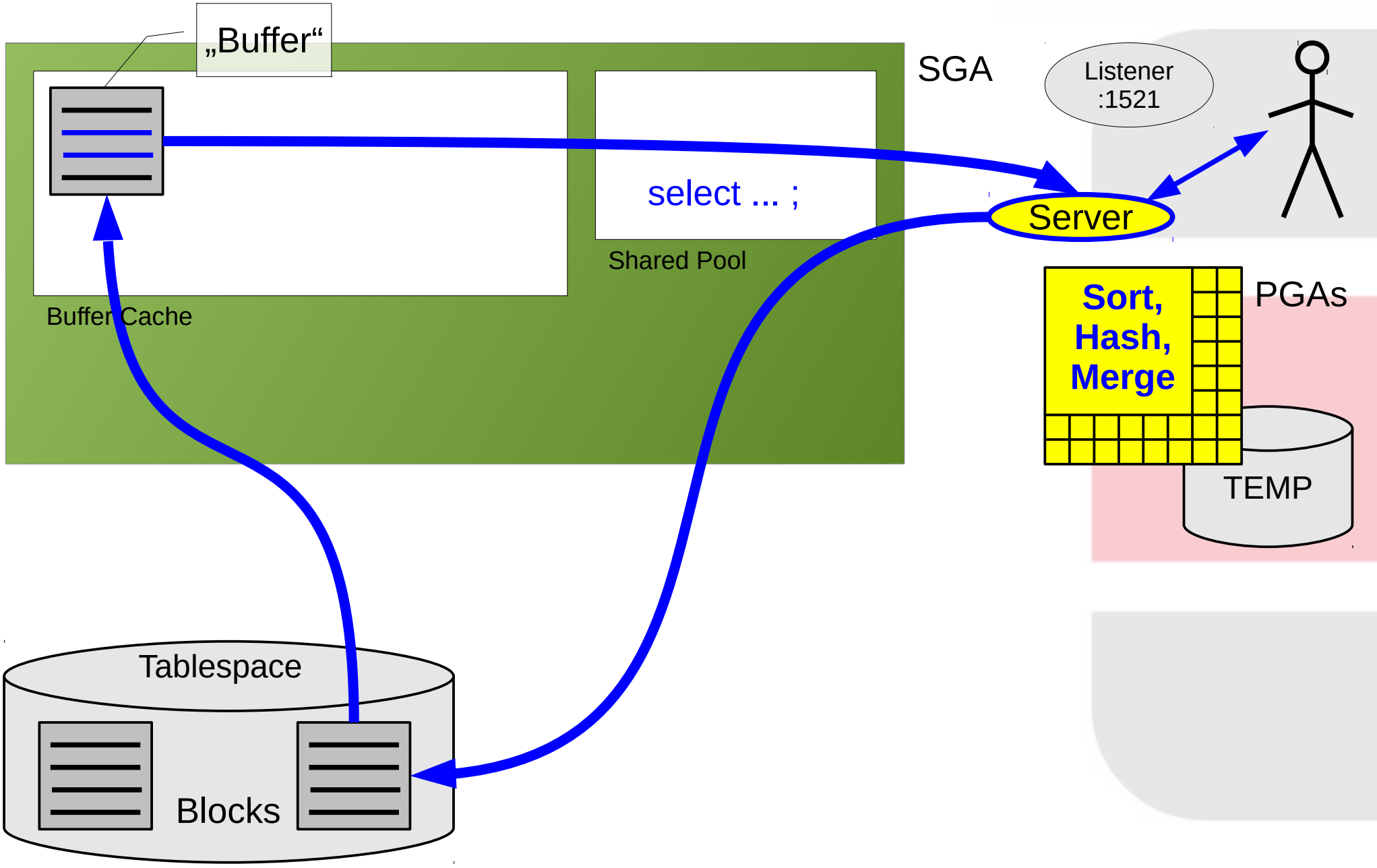
Basics

Oracle Architecture
(simplified)

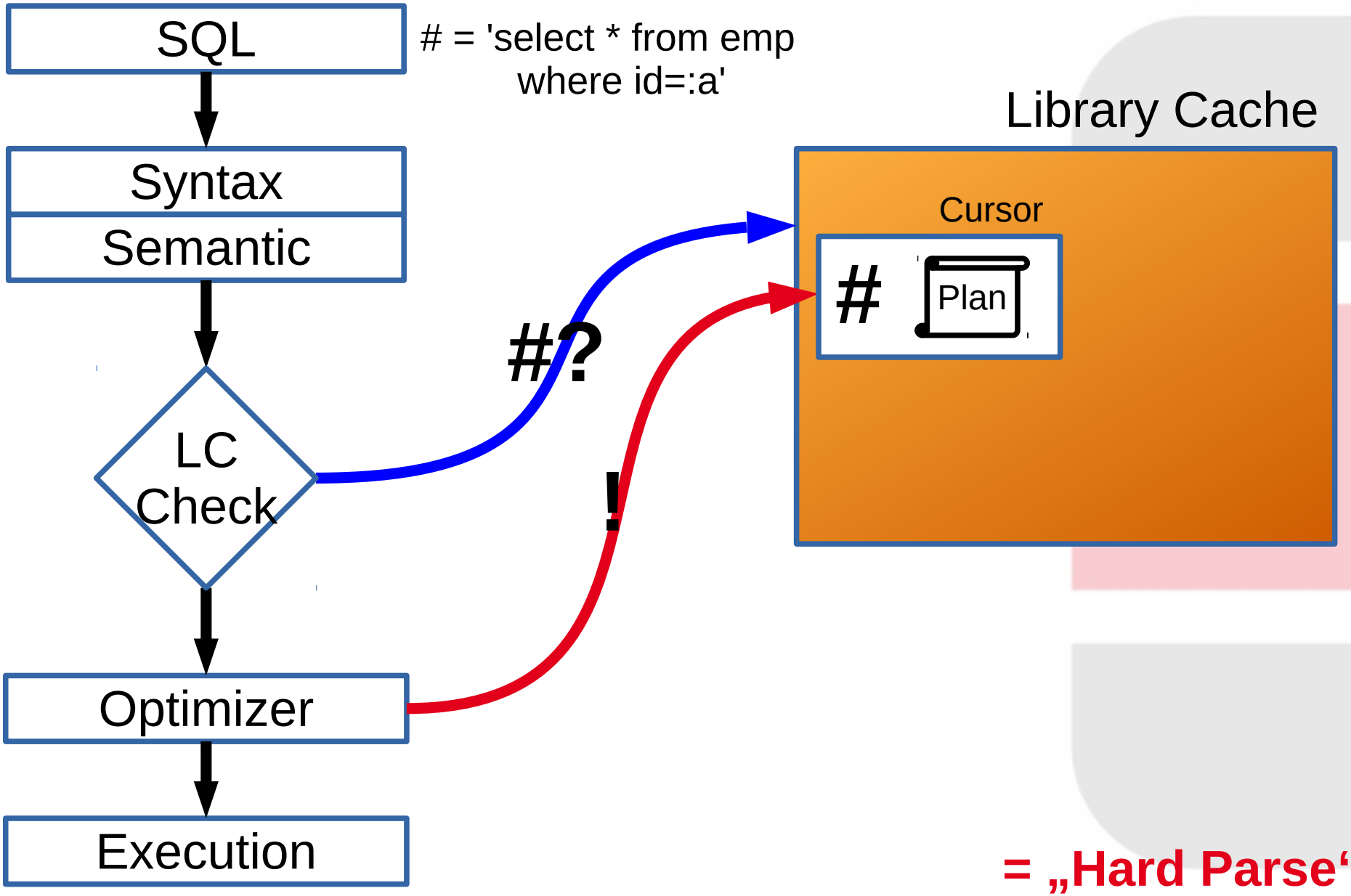
performing databases
Your reliability. Our concern.



Reading

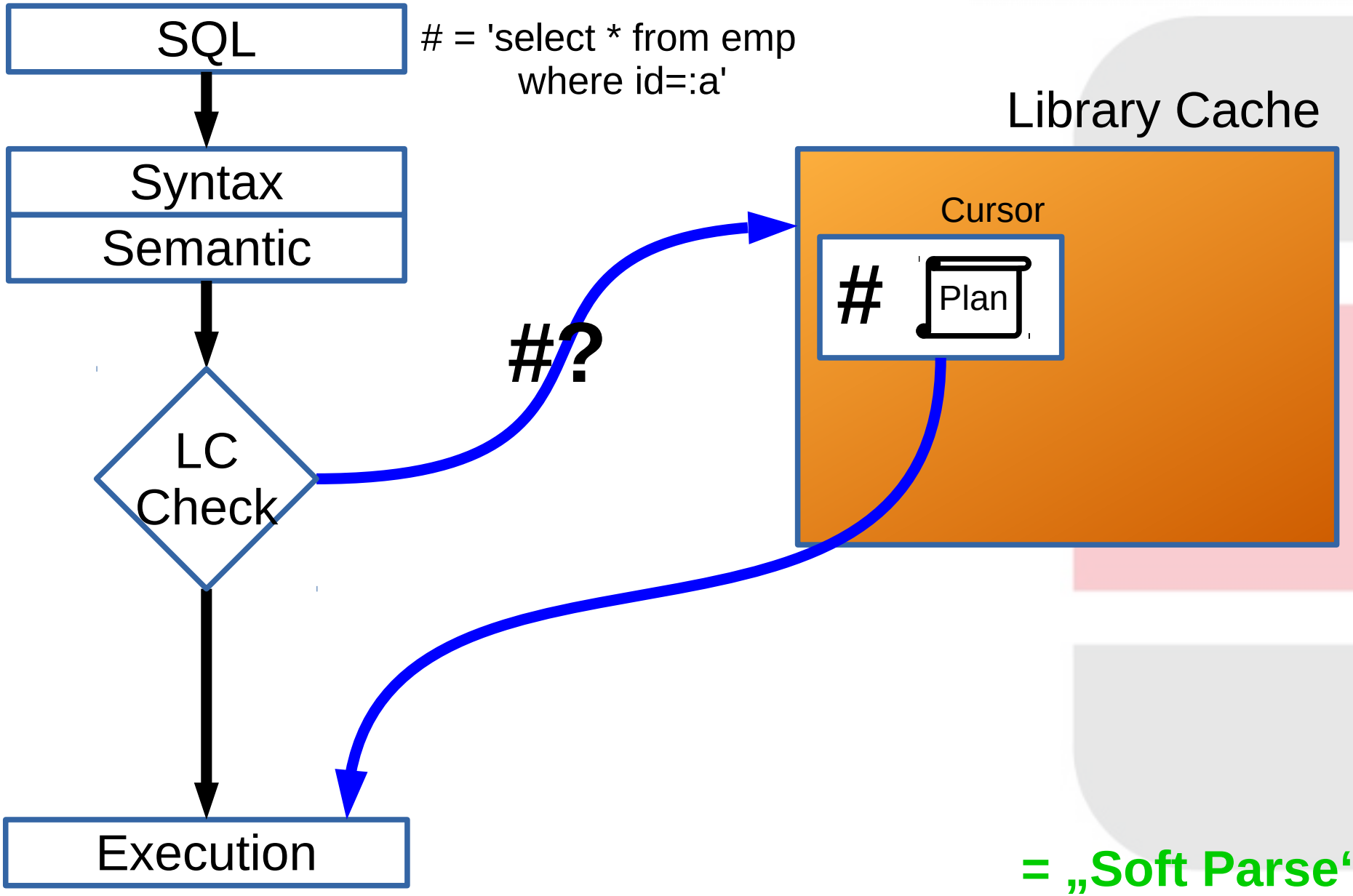

„Buffer“
SGA
Listener
:1521
select ... ;
Server
Shared Pool
Buffer Cache
Sort,
Hash,
Merge
PGAs
TEMP
Tablespace
Blocks

Parsing

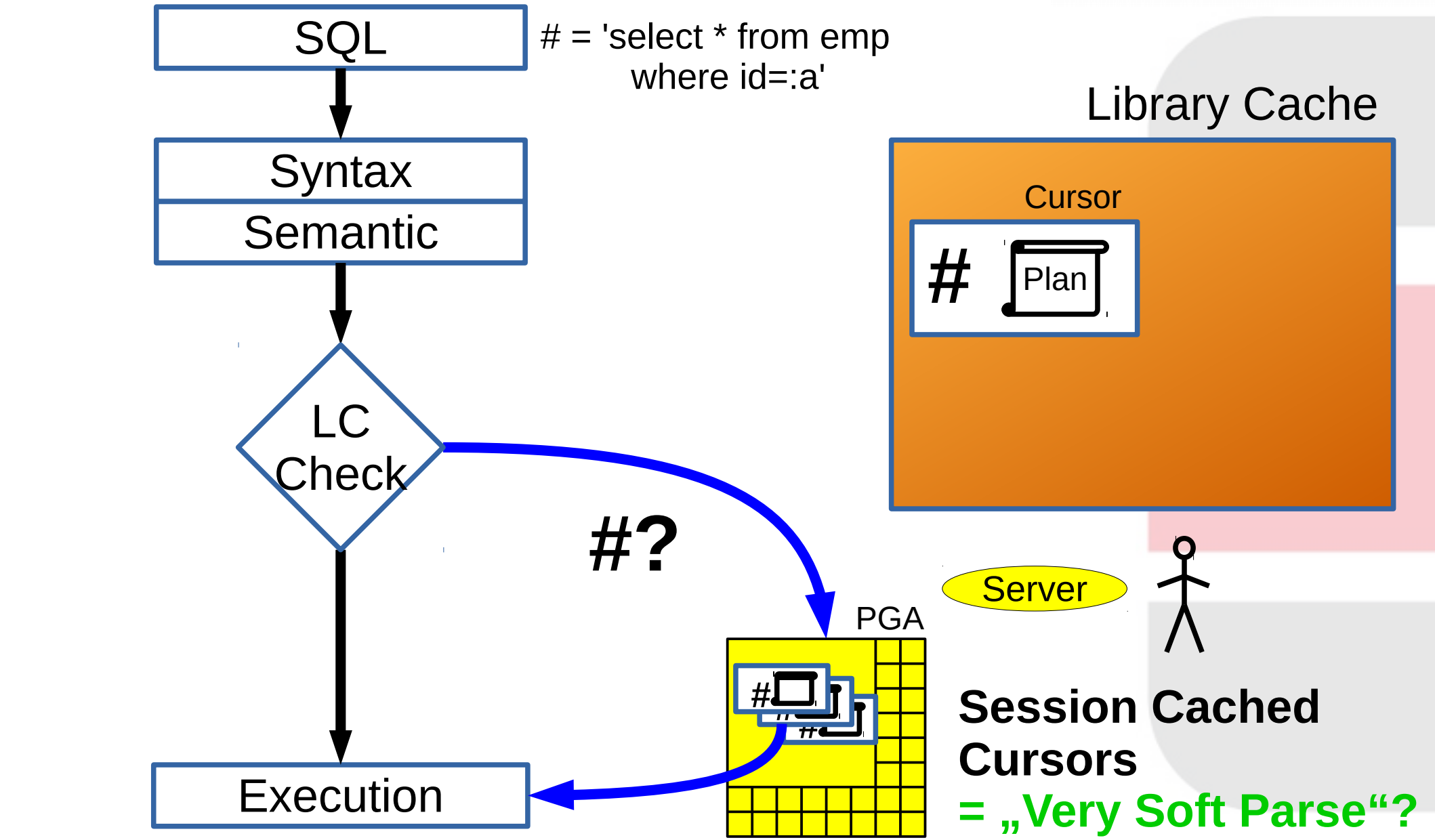


= „Hard Parse“

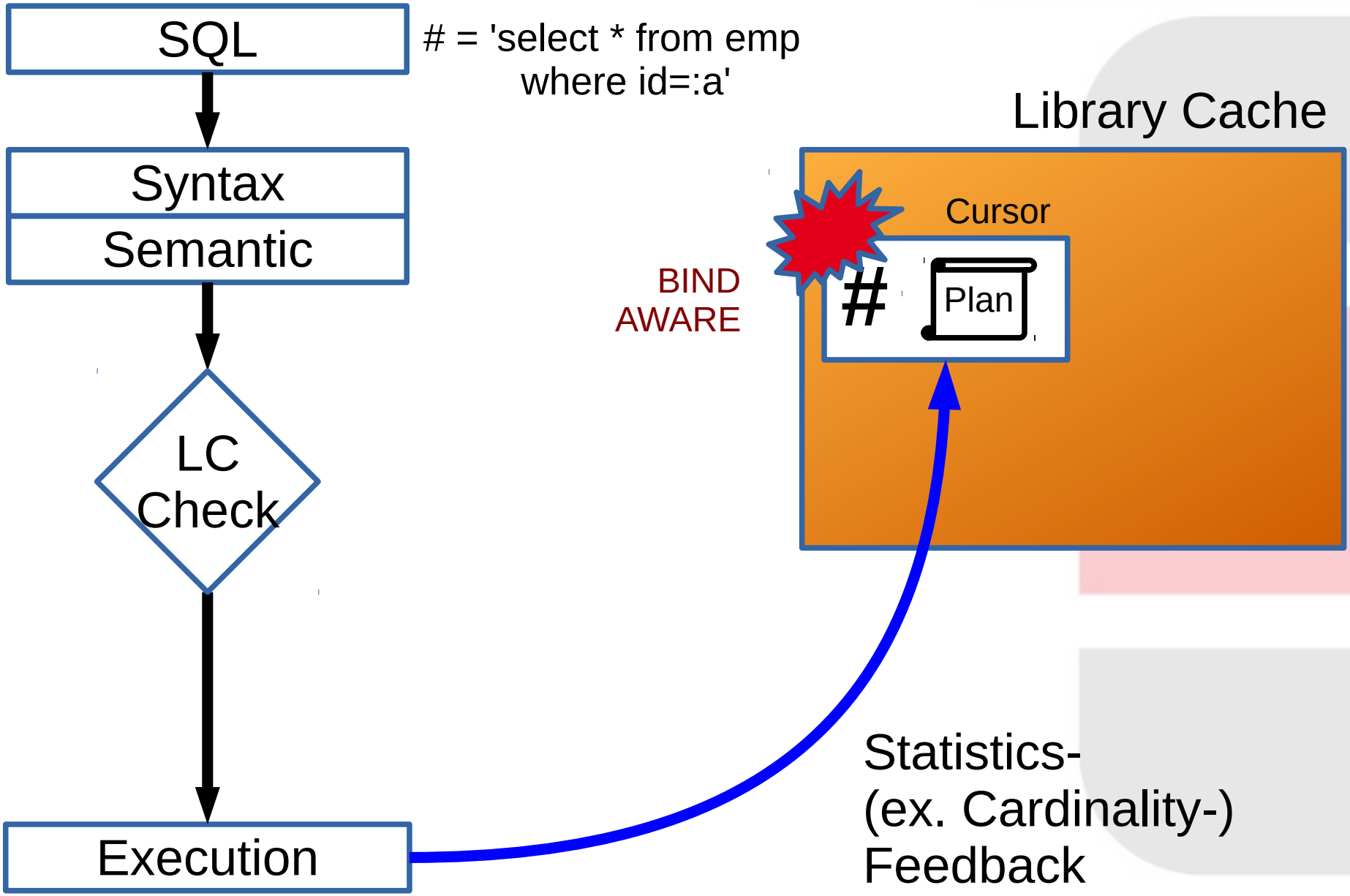
Cursor Sharing



Session Cached Cursors

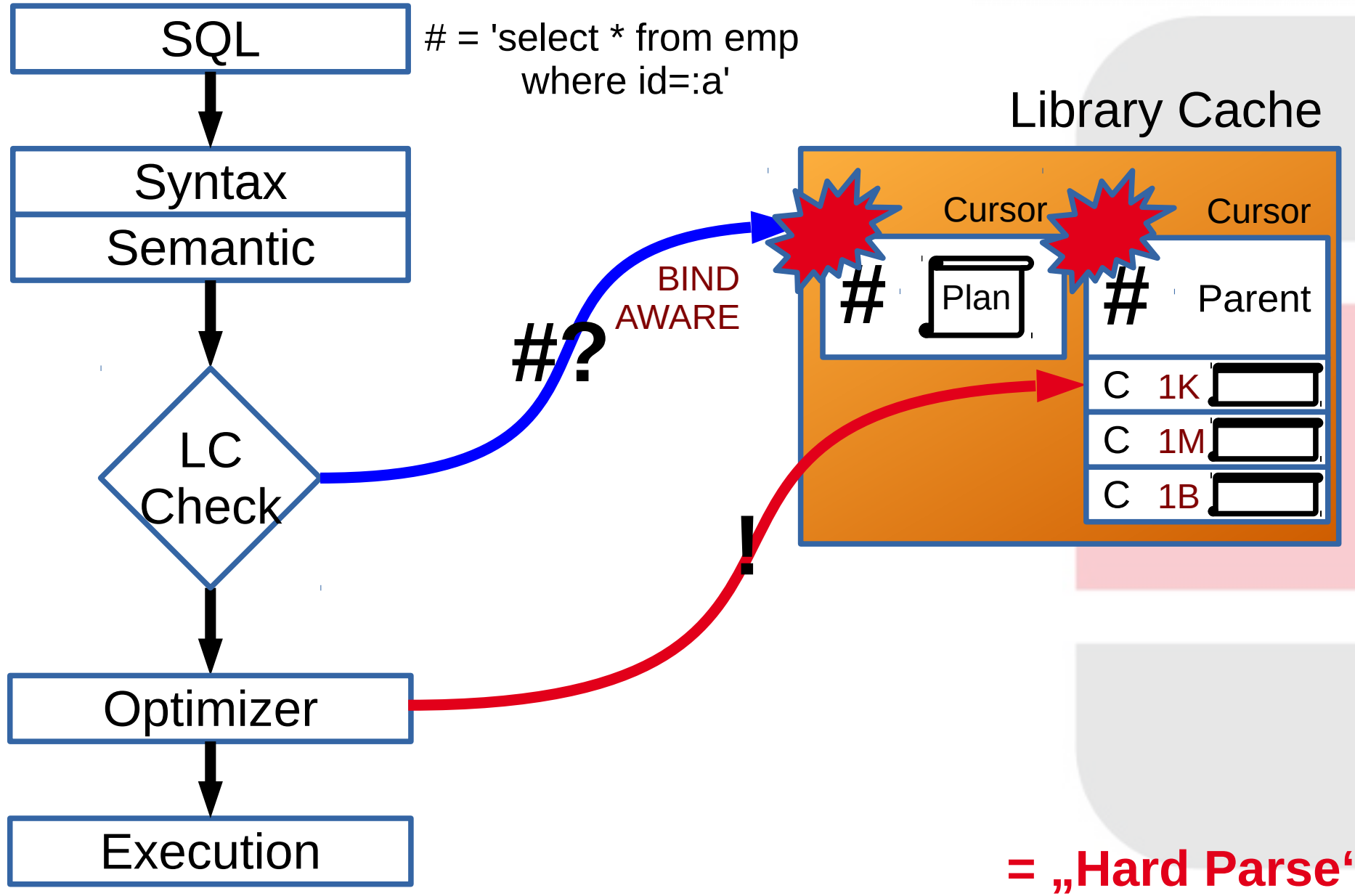


Statistics Feedback



Child Cursor Plans

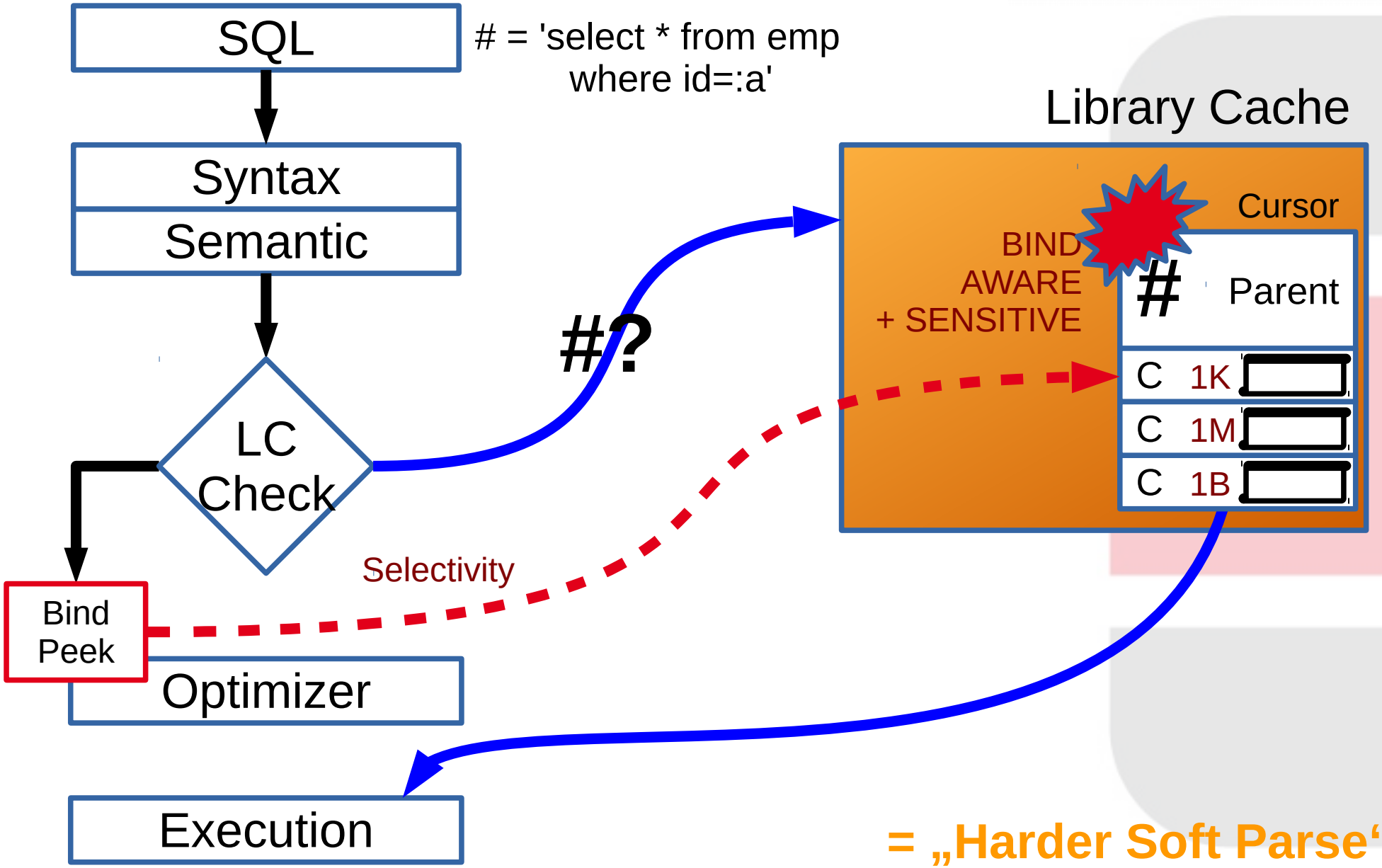
performing databases
Your reliability. Our concern.



= „Hard Parse“

Adaptive Cursor Sharing

performing databases
Your reliability. Our concern.



= „Harder Soft Parse“

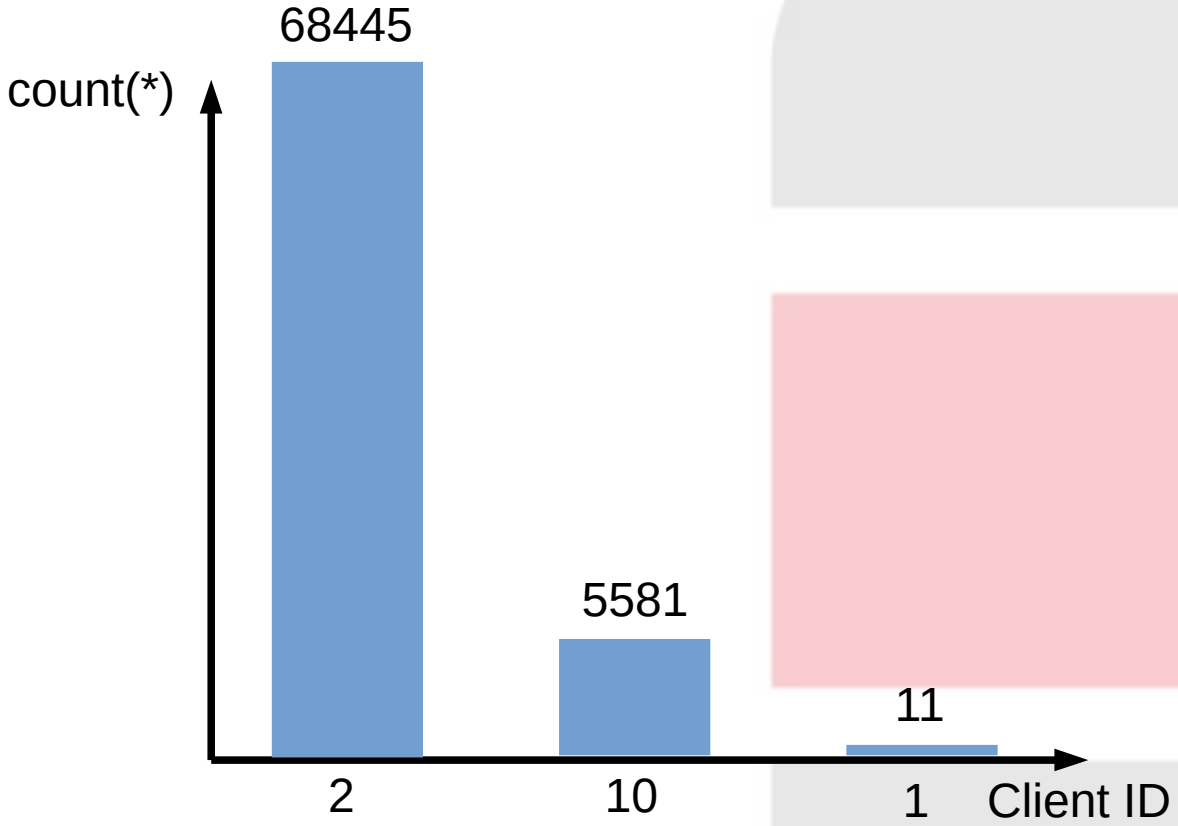
performing databases
Your reliability. Our concern.

Adaptive Cursor Sharing Example

Live-case Adaptive Cursor

Real-life example, Mar 30, 2016, Germany

```
select *
    from COLLI
    where status <> :3
        and moddat < :2
        and clientid = :1
;
```



In our three cases called with status <> 3 and moddat < sysdate

Index usage for ACS

performing databases
Your reliability. Our concern.

```
30
31  select owner,index_name,table_name,blevel,leaf_blocks,AVG_LEAF_BLOCKS_PER_KEY,distinct_keys,AVG_DATA_BLOCKS_PER_KEY,CLUSTERING_FACTOR
32  from DBA_IND_STATISTICS
33  where owner='WAGOZL5K_STT_QA'
34    and INDEX_NAME in ('XIF133COLLI','XAK1COLLI');
35
```

Query Result 4 x

SQL | All Rows Fetched: 2 in 0.249 seconds

	OWNER	INDEX_NAME	TABLE_NAME	BLEVEL	LEAF_BLOCKS	AVG_LEAF_BLOCKS_PER_KEY	DISTINCT_KEYS	AVG_DATA_BLOCKS_PER_KEY	CLUSTERING_FACTOR
1	WAGOZL5K_STT_QA	XIF133COLLI	COLLI	2	886	126	7	6175	43229
2	WAGOZL5K_STT_QA	XAK1COLLI	COLLI	2	478	1	148232	1	124170

XIF133COLLI = Blevel 2 886 LeafBl 43k ClusteringF

XAK1COLLI = Blevel 2 478 LeafBl 124k ClusteringF

ACS Cursor 3

performing databases
Your reliability. Our concern.

```
SQL_ID  bzbbg21catfrc, child number 3
-------------------------------------

select * from colli where status <> :3 and moddat < :2 and clientid = :1

Plan hash value: 4000139639

-------------------------------------------------------------------------------
| Id  | Operation         | Name  | E-Rows |E-Bytes| Cost (%CPU)| E-Time   |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |        |       |  9482 (100)|          |
|*  1 |  TABLE ACCESS FULL| COLLI |   5552 |  3307K|  9482   (1)| 00:00:01 |
-------------------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------

   1 - :1 (NUMBER): 2
   2 - :2 (TIMESTAMP): [Not Printable]
   3 - (NUMBER): 3

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(("STATUS"<>:3 AND "CLIENTID"=:1 AND "MODDAT"<:2))
```

Customer ID 2

Real # = 68445

ACS Cursor 0

performing databases
Your reliability. Our concern.

```
SQL_ID  bzbbg21catfrc, child number 0
-------------------------------------

select * from colli where status <> :3 and moddat < :2 and clientid = :1

Plan hash value: 1092566166
-----------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name        | E-Rows |E-Bytes| Cost (%CPU)| E-Time   |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |             |        |       |   196 (100)|          |
|*  1 |  TABLE ACCESS BY INDEX ROWID BATCHED  | COLLI       |     31 | 18910 |   196   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                    | XIF133COLLI |    652 |       |     6   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------

   1 - :1 (NUMBER): 10
   2 - :2 (TIMESTAMP): [Not Printable]
   3 - (NUMBER): 3

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(("STATUS"<>:3 AND "MODDAT"<:2))
   2 - access("CLIENTID"=:1)
```

Customer ID 10

Real # = 5581

XIF133COLLI = 886 LeafBl 43k ClusteringF

ACS Cursor 8

performing databases
Your reliability. Our concern.

```
SQL_ID  bzbbg21catfrc, child number 8
-------------------------------------

select * from colli where status <> :3 and moddat < :2 and clientid = :1

Plan hash value: 1188170921

-------------------------------------------------------------------------------------------
| Id  | Operation                                | Name       | E-Rows |E-Bytes| Cost (%CPU)|
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |            |        |       |     1 (100)|
|*  1 |  TABLE ACCESS BY INDEX ROWID BATCHED| COLLI      |      1 | 17265 |     0   (0)|
|*  2 |   INDEX RANGE SCAN                       | XAK1COLLI  |      1 |       |     0   (0)|
-------------------------------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------

   1 - :1 (NUMBER): 1
   2 - :2 (TIMESTAMP): [Not Printable]
   3 - (NUMBER): 3

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(("MODDAT"<:2 AND "STATUS"<>:3))
   2 - access("CLIENTID"=:1)
```

Customer ID 1

Real # = 11

XAK1COLLI = 478 LeafBl 124k ClusteringF

performing databases
Your reliability. Our concern.

Ressource Consumption

Makes you vulnerable

performing databases
Your reliability. Our concern.

Short: AWR Reports

Reading a Report

performing databases
Your reliability. Our concern.

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
CHIWPROD	1065804552	chiwprod	1	13-Feb-16 10:02	11.2.0.4.0	NO

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
CHLUSWI6001	Microsoft Windows x86 64-bit	12	12	2	127.87

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	18916	03-Mar-16 11:00:33	265	5.5
End Snap:	18918	03-Mar-16 12:00:40	255	5.6
Elapsed:		60.12 (mins)		
DB Time:		51.89 (mins)		

We observed 60min of clock time
We had 12 cores for 60min = 720min plus x of time
DB used only 51.89 min (nearly 100% of 1 core)

Can this be overload? Check for single-threaded problem.

Reading a Report

performing databases
Your reliability. Our concern.

Report Summary

Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	0.9	0.0	0.00	0.00
DB CPU(s):	0.8	0.0	0.00	0.00
Redo size (bytes):	388,752.8	5,645.8		
Logical read (blocks):	56,418.7	819.4		
Block changes:	2,130.6	30.9		
Physical read (blocks):	67.0	1.0		
Physical write (blocks):	88.4	1.3		
Read IO requests:	62.3	0.9		
Write IO requests:	56.6	0.8		
Read IO (MB):	0.5	0.0		
Write IO (MB):	0.7	0.0		
User calls:	4,905.5	71.2		
Parses (SQL):	2,046.4	29.7		
Hard parses (SQL):	1.1	0.0		
SQL Work Area (MB):	11.4	0.2		
Logons:	0.4	0.0		
Executes (SQL):	2,199.4	31.9		
Rollbacks:	30.3	0.4		
Transactions:	68.9			

CPU

IO

CPU

Reading a Report

performing databases
Your reliability. Our concern.

SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- %Total - Buffer Gets as a percentage of Total Buffer Gets
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Total Buffer Gets: 203,500,151
- Captured SQL account for 80.9% of Total

Buffer Gets	Executions	Gets per Exec	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL
39,836,966	294	135,499.88	19.58	180.92	98,4	0	c5bpttzczmj5r	iWACS	SELECT mailingId, m
39,161,524	290	135,039.74	19.24	172.27	99,9	0	403dzpcdzg3pt	iWACS	SELECT mailingId, m
9,589,820	121	79,254.71	4.71	16.63	99,5	0	drzjvrr83hu0q	rueckmto@DEBSCTC0098	SELECT goodsInPos
8,392,022	240,880	34.84	4.12	44.51	101,3	0	fvgsk34swhm9b	blumca@chluzctc1431	SELECT tuId, tuNo, t
6,211,310	240,891	25.78	3.05	14.68	102,5	0	70rxm16vmf54r	blumca@chluzctc1431	SELECT LU.SKUID F
6,071,773	7	867,396.14	2.98	23.96	99,7	0	d5msj347tupum	cukicjo@chluzcwm1108	SELECT colliId, ware
4,886,489	11,742	416.15	2.40	13.36	95,2	,2	brz36vgcc762v	iWACS	SELECT clientId, flag
4,209,341	60	70,155.68	2.07	17.45	100,8	0	g7s9p5dyx5uu2	iWACS	SELECT clientIdPrj, v
4,182,909	59,693	70.07	2.06	11.52	99,8	0	ag48gvq4rf6fb	iWACS	SELECT clientId, dtsI
3,341,004	63	53,031.81	1.64	12.23	99,7	0	b2q63rnz0xts1	iselich@chluzcwm1181	SELECT warehouseS
3,022,973	133	22,729.12	1.49	84.83	70,2	30,2	2301utna4pd58	nedderra@DEBSCTC0099	SELECT storageLoca
2,623,385	2,186	1,200.08	1.29	7.27	103,4	0	aj99nfbdyuvgt	blumca@chluzctc1431	SELECT tuId, storage
2,456,341	821,471	2.99	1.21	16.59	94,1	0	0xfa1r30mk0v3	iWACS	SELECT rackId, rackI
2,423,654	294	8,243.72	1.19	7.06	100,2	0	96v0ussgxx860	broemeha@DEBSCTC0139	SELECT goodsInId, v

Buffer Gets cause (CPU) load

=> Big ones first!

performing databases
Your reliability. Our concern.

Execution Plan

Example -1-

My #1 Tool

performing databases
Your reliability. Our concern.

```
SELECT *
FROM table(
          DBMS_XPLAN.DISPLAY_CURSOR(
              '&&SQL_ID',
              null,
              'COST,IOSTATS,LAST,ADVANCED,ADAPTIVE'
              )
          );
```

613K Buffer Gets

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-Time	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1			1832 (100)		70	00:00:04.48	613K
1	SORT ORDER BY		1	5	2590	1832 (1)	00:00:22	70	00:00:04.48	613K
2	VIEW	VIEWRTPICKING	1	5	2590	1831 (1)	00:00:22	70	00:00:04.48	613K
3	SORT UNIQUE		1	5	1376	1831 (75)	00:00:22	70	00:00:04.48	613K
4	UNION-ALL		1					70	00:00:04.48	613K
* 5	TABLE ACCESS BY INDEX ROWID	PICKORDERPOS	70	1	9	3 (0)	00:00:01	70	00:00:00.01	120
* 6	INDEX UNIQUE SCAN	XPKPICKORDERPOS	70	1		2 (0)	00:00:01	70	00:00:00.01	50
7	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	24	2 (0)	00:00:01	35	00:00:00.01	73
* 8	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
9	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	24	2 (0)	00:00:01	35	00:00:00.01	73
* 10	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
11	NESTED LOOPS		35					35	00:00:00.01	77
12	NESTED LOOPS		35	1	16	3 (0)	00:00:01	35	00:00:00.01	42
* 13	INDEX RANGE SCAN	I_STORAGELOCATION_TUNING_1	35	1	9	2 (0)	00:00:01	35	00:00:00.01	38
* 14	INDEX UNIQUE SCAN	XPKWAREHOUSE	35	1		0 (0)		35	00:00:00.01	4
15	TABLE ACCESS BY INDEX ROWID	WAREHOUSE	35	43	301	1 (0)	00:00:01	35	00:00:00.01	35
16	NESTED LOOPS		35	1	25	3 (0)	00:00:01	35	00:00:00.01	117
17	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	13	2 (0)	00:00:01	35	00:00:00.01	73
* 18	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
19	TABLE ACCESS BY INDEX ROWID	RACK	35	11276	132K	1 (0)	00:00:01	35	00:00:00.01	44
* 20	INDEX UNIQUE SCAN	XPKRACK	35	1		0 (0)		35	00:00:00.01	9
21	NESTED LOOPS		35	1	25	3 (0)	00:00:01	35	00:00:00.01	117
22	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	13	2 (0)	00:00:01	35	00:00:00.01	73
* 23	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
24	TABLE ACCESS BY INDEX ROWID	RACK	35	11276	132K	1 (0)	00:00:01	35	00:00:00.01	44
* 25	INDEX UNIQUE SCAN	XPKRACK	35	1		0 (0)		35	00:00:00.01	9
26	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	33	1	9	2 (0)	00:00:01	33	00:00:00.01	69
* 27	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	33	1		1 (0)	00:00:01	33	00:00:00.01	36
28	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	2	1	9	2 (0)	00:00:01	2	00:00:00.01	6
* 29	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	2	1		1 (0)	00:00:01	2	00:00:00.01	4
30	SORT AGGREGATE		1	1	8			1	00:00:00.01	376
* 31	INDEX FAST FULL SCAN	XAK1STORAGELOCATION	1	20257	158K	80 (2)	00:00:01	20773	00:00:00.01	376
32	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	9	2 (0)	00:00:01	35	00:00:00.01	73
* 33	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
34	HASH UNIQUE		36	1	8	145K (1)	00:29:09	36	00:00:04.48	610K
* 35	COUNT STOPKEY		36					36	00:00:04.45	610K
* 36	FILTER		36					36	00:00:04.45	610K
37	TABLE ACCESS FULL	PACKINGUNIT	36	28340	221K	210 (1)	00:00:03	556K	00:00:00.01	15311
38	NESTED LOOPS		556K					36	00:00:03.42	595K
39	NESTED LOOPS		556K	1	32	6 (0)	00:00:01	87	00:00:03.17	595K
40	TABLE ACCESS BY INDEX ROWID	LU	556K	1	19	4 (0)	00:00:01	87	00:00:03.00	595K
* 41	INDEX RANGE SCAN	I_TUNING_LU_3	556K	1		3 (0)	00:00:01	87	00:00:02.90	595K
* 42	INDEX UNIQUE SCAN	XPKTU	87	1		1 (0)	00:00:01	87	00:00:00.01	214
* 43	TABLE ACCESS BY INDEX ROWID	TU	87	1	13	2 (0)	00:00:01	36	00:00:00.01	87
44	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	8	2 (0)	00:00:01	35	00:00:00.01	73
* 45	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
46	NESTED LOOPS		1					70	00:00:00.01	1135
47	NESTED LOOPS		1	1	272	473 (1)	00:00:06	140	00:00:00.01	1088
48	NESTED LOOPS OUTER		1	1	263	469 (1)	00:00:06	70	00:00:00.01	1020
49	NESTED LOOPS		1	1	256	467 (1)	00:00:06	70	00:00:00.01	912
50	NESTED LOOPS		1	1	245	466 (1)	00:00:06	70	00:00:00.01	812
51	NESTED LOOPS		1	1	230	463 (0)	00:00:06	70	00:00:00.01	664
52	NESTED LOOPS		1	1	189	462 (0)	00:00:06	70	00:00:00.01	564
53	NESTED LOOPS		1	1	182	461 (0)	00:00:06	70	00:00:00.01	388
54	NESTED LOOPS		1	1	139	460 (0)	00:00:06	70	00:00:00.01	245
55	NESTED LOOPS		1	1	113	458 (0)	00:00:06	70	00:00:00.01	97
56	NESTED LOOPS		1	41	3116	344 (0)	00:00:05	6	00:00:00.01	52
57	NESTED LOOPS		1	41	2747	303 (0)	00:00:04	6	00:00:00.01	38
58	NESTED LOOPS		1	193	9457	27 (0)	00:00:01	6	00:00:00.01	26
59	NESTED LOOPS		1	5	195	4 (0)	00:00:01	6	00:00:00.01	11
60	NESTED LOOPS		1	1	22	2 (0)	00:00:01	1	00:00:00.01	4
* 61	INDEX UNIQUE SCAN	XPKWAREHOUSE	1	1	4	0 (0)		1	00:00:00.01	1
* 62	TABLE ACCESS BY INDEX ROWID	RACK	1	1	18	2 (0)	00:00:01	1	00:00:00.01	3
* 63	INDEX RANGE SCAN	IDXRACK_1	1	1		1 (0)	00:00:01	1	00:00:00.01	2
64	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	1	5	85	2 (0)	00:00:01	6	00:00:00.01	7
* 65	INDEX RANGE SCAN	XIF1STORAGELOCATION	1	5		1 (0)	00:00:01	6	00:00:00.01	2
66	TABLE ACCESS BY INDEX ROWID	TU	6	38	380	15 (0)	00:00:01	6	00:00:00.01	15
* 67	INDEX RANGE SCAN	XIF2TU	6	170		2 (0)	00:00:01	6	00:00:00.01	14
* 68	TABLE ACCESS BY INDEX ROWID	PICKTUPOS	6	1	18	3 (0)	00:00:01	6	00:00:00.01	12
* 69	INDEX RANGE SCAN	IDXPICKTUPOS_2	6	2		1 (0)	00:00:01	6	00:00:00.01	8
* 70	TABLE ACCESS BY INDEX ROWID	PICKTU	6	1	9	1 (0)	00:00:01	6	00:00:00.01	14

Inliners

```
select
      (select …) as
from a,b,c
where….
```

Main Join „Staircase“

Execution Plan Overview

Real-life example, Nov 5, 2017, Switzerland

performing databases
Your reliability. Our concern.

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-Time	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1			1832 (100)		70	00:00:04.48	613K
1	SORT ORDER BY		1	5	2590	1832 (1)	00:00:22	70	00:00:04.48	613K
2	VIEW	VIEWRTPICKING	1	5	2590	1831 (1)	00:00:22	70	00:00:04.48	613K
3	SORT UNIQUE		1	5	1376	1831 (75)	00:00:22	70	00:00:04.48	613K
4	UNION-ALL		1					70	00:00:04.48	613K
* 5	TABLE ACCESS BY INDEX ROWID	PICKORDERPOS	70	1	9	3 (0)	00:00:01	70	00:00:00.01	120
* 6	INDEX UNIQUE SCAN	XPKPICKORDERPOS	70	1		2 (0)	00:00:01	70	00:00:00.01	50
7	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	24	2 (0)	00:00:01	35	00:00:00.01	73
* 8	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
9	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	24	2 (0)	00:00:01	35	00:00:00.01	73
* 10	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
11	NESTED LOOPS		35					35	00:00:00.01	77
12	NESTED LOOPS		35	1	16	3 (0)	00:00:01	35	00:00:00.01	42
* 13	INDEX RANGE SCAN	I_STORAGELOCATION_TUNING_1	35	1	9	2 (0)	00:00:01	35	00:00:00.01	38
* 14	INDEX UNIQUE SCAN	XPKWAREHOUSE	35	1		0 (0)		35	00:00:00.01	4
15	TABLE ACCESS BY INDEX ROWID	WAREHOUSE	35	43	301	1 (0)	00:00:01	35	00:00:00.01	35
16	NESTED LOOPS		35	1	25	3 (0)	00:00:01	35	00:00:00.01	117
17	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	13	2 (0)	00:00:01	35	00:00:00.01	73
* 18	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
19	TABLE ACCESS BY INDEX ROWID	RACK	35	11276	132K	1 (0)	00:00:01	35	00:00:00.01	44
* 20	INDEX UNIQUE SCAN	XPKRACK	35	1		0 (0)		35	00:00:00.01	9
21	NESTED LOOPS		35	1	25	3 (0)	00:00:01	35	00:00:00.01	117
22	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	13	2 (0)	00:00:01	35	00:00:00.01	73
* 23	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
24	TABLE ACCESS BY INDEX ROWID	RACK	35	11276	132K	1 (0)	00:00:01	35	00:00:00.01	44
* 25	INDEX UNIQUE SCAN	XPKRACK	35	1		0 (0)		35	00:00:00.01	9
26	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	33	1	9	2 (0)	00:00:01	33	00:00:00.01	69
* 27	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	33	1		1 (0)	00:00:01	33	00:00:00.01	36
28	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	2	1	9	2 (0)	00:00:01	2	00:00:00.01	6
* 29	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	2	1		1 (0)	00:00:01	2	00:00:00.01	4
30	SORT AGGREGATE		1	1	8			1	00:00:00.01	376
* 31	INDEX FAST FULL SCAN	XAK1STORAGELOCATION	1	20257	158K	80 (2)	00:00:01	20773	00:00:00.01	376
32	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	9	2 (0)	00:00:01	35	00:00:00.01	73
* 33	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
34	HASH UNIQUE		36	1	8	145K (1)	00:29:09	36	00:00:04.48	610K
* 35	COUNT STOPKEY		36					36	00:00:04.45	610K
* 36	FILTER		36					36	00:00:04.45	610K
37	TABLE ACCESS FULL	PACKINGUNIT	36	28340	221K	210 (1)	00:00:03	556K	00:00:00.01	15311
38	NESTED LOOPS		556K					36	00:00:03.42	595K
39	NESTED LOOPS		556K	1	32	6 (0)	00:00:01	87	00:00:03.17	595K
40	TABLE ACCESS BY INDEX ROWID	LU	556K	1	19	4 (0)	00:00:01	87	00:00:03.00	595K
* 41	INDEX RANGE SCAN	I_TUNING_LU_3	556K	1		3 (0)	00:00:01	87	00:00:02.90	595K
* 42	INDEX UNIQUE SCAN	XPKTU	87	1		1 (0)	00:00:01	87	00:00:00.01	214
* 43	TABLE ACCESS BY INDEX ROWID	TU	87	1	13	2 (0)	00:00:01	36	00:00:00.01	87
44	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	35	1	8	2 (0)	00:00:01	35	00:00:00.01	73
* 45	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	35	1		1 (0)	00:00:01	35	00:00:00.01	38
46	NESTED LOOPS		1					70	00:00:00.01	1135
47	NESTED LOOPS		1	1	272	473 (1)	00:00:06	140	00:00:00.01	1088
48	NESTED LOOPS OUTER		1	1	263	469 (1)	00:00:06	70	00:00:00.01	1020
49	NESTED LOOPS		1	1	256	467 (1)	00:00:06	70	00:00:00.01	912
50	NESTED LOOPS		1	1	245	466 (1)	00:00:06	70	00:00:00.01	812
51	NESTED LOOPS		1	1	230	463 (0)	00:00:06	70	00:00:00.01	664
52	NESTED LOOPS		1	1	189	462 (0)	00:00:06	70	00:00:00.01	564
53	NESTED LOOPS		1	1	182	461 (0)	00:00:06	70	00:00:00.01	388
54	NESTED LOOPS		1	1	139	460 (0)	00:00:06	70	00:00:00.01	245
55	NESTED LOOPS		1	1	113	458 (0)	00:00:06	70	00:00:00.01	97

Overall Heat
613K Buffer Gets

Inliners
400 BG
=>„Small Game“

Inliner
Filter + Count +
Hash Unique
610K BG

Main Join
„Staircase“
~1k BG

Execution Plan Detail

performing databases
Your reliability. Our concern.

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-
34	HASH UNIQUE		36	1	8	145K (1)	00
* 35	COUNT STOPKEY		36				
* 36	FILTER		36				
37	TABLE ACCESS FULL	PACKINGUNIT	36	28340	221K	210 (1)	00
38	NESTED LOOPS		556K				
39	NESTED LOOPS		556K	1	32	6 (0)	00
40	TABLE ACCESS BY INDEX ROWID	LU	556K	1	19	4 (0)	00
* 41	INDEX RANGE SCAN	I_TUNING_LU_3	556K	1		3 (0)	00
* 42	INDEX UNIQUE SCAN	XPKTU	87	1		1 (0)	00
* 43	TABLE ACCESS BY INDEX ROWID	TU	87	1	13	2 (0)	00

```
35 - filter(ROWNUM=1)
36 - filter( IS NOT NULL) Filter = Intermediate Result :(
41 - access("LU"."SKUID"=:B1 AND "LU"."PACKINGUNITID"=:B2 AND "LU"."STATUS"<99)
     filter("LU"."PACKINGUNITID"=:B1)
42 - access("LU"."TUID"="TU2"."TUID")
43 - filter(("TU2"."STORAGELOCATIONID"=:B1 AND "TU2"."STATUS"<99))
```

Not all reducing predicates in an index

Major Problem:
- **JOIN grows big, because**
- **PACKINGUNIT does not work as a reduction here (not part of JOIN)**
- **Both have to be filtered afterwards (big heat)**

Analyzing the View

performing databases
Your reliability. Our concern.

This was used as an Inline Select (=dynamic field) of a MASSIVE join

```
(SELECT DISTINCT pu.qtyunit
   FROM packingunit pu
  WHERE pu.packingunitid IN (SELECT DISTINCT lu.packingunitid
                               FROM lu
                              WHERE lu.tuid IN (SELECT tu2.tuid
                                                  FROM tu tu2
                                                 WHERE tu2.storagelocationid = pop.storagelocationid
                                                   AND tu2.status < 99)
                                AND lu.skuid = sku.skuid
                                AND lu.status < 99) AND ROWNUM = 1) AS bqtyunit,
```

LU ist the largest table in the database.
Using IN() and DISTINCT makes it impossible to estimate the cardinality here or to Query-rewrite it to a JOIN!

Tables are de-facto referenced by Foreign Keys pointing to Primary Keys.
JOIN is an option! :)

Improving the View

performing databases
Your reliability. Our concern.

In many cases, the Cost-based Optimizer works best, if problems can be solved by USING JOIN OPERATIONS!

So why not just thinking + working the same way?

```
(SELECT pu.qtyunit
 FROM PackingUnit pu,
      Lu lu,
      Tu tu
 WHERE pu.packingunitid = lu.packingunitid
     AND lu.skuid = sku.skuid
     AND lu.status < 99
     AND lu.tuid = tu.tuid
     AND tu.status < 99
     AND tu.storagelocationid = pop.storagelocationid
     AND ROWNUM = 1) AS bqtyunit,
```

Just joining ...

Removed DISTINCT, since it's useless:
Join Condition (FK on PK) + ROWNUM=1 does the same here

Added 2-column index + Extended Statistics on TU

Execution Plan Improved

performing databases
Your reliability. Our concern.

1.700 Buffer Gets

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-
* 34	COUNT STOPKEY		0				
35	NESTED LOOPS		0	4	160	13 (8)	00
* 36	HASH JOIN		0	4	128	9 (12)	00
37	TABLE ACCESS BY INDEX ROWID	LU	0	4	76	5 (0)	00
* 38	INDEX RANGE SCAN	I_TUNING_LU_3	0	4		3 (0)	00
* 39	INDEX RANGE SCAN	I_TUNING_TU_2	0	68	884	3 (0)	00
* 40	INDEX RANGE SCAN	I_TUNING_PU_1	0	1	8	1 (0)	00
41	TABLE ACCESS BY INDEX ROWID	STORAGELOCATION	0	1	8	2 (0)	00
* 42	INDEX UNIQUE SCAN	XPKSTORAGELOCATION	0	1		1 (0)	00

```
34 - filter(ROWNUM=1)
36 - access("LU"."TUID"="TU"."TUID")
38 - access("LU"."SKUID"=:B1 AND "LU"."STATUS"<99)
39 - access("TU"."STORAGELOCATIONID"=:B1 AND "TU"."STATUS"<99)
40 - access("PU"."PACKINGUNITID"="LU"."PACKINGUNITID")
42 - access("STORAGELOCATIONID"=:B1)
```

What helped:
- **Straight join approach allows accurate Cardinality Estimates, based on**
- **Multi-Column (=Extended) Statistics**
- **Most results can be read from Multi-Column Indexes**

performing databases
Your reliability. Our concern.

Build Bridges of Gold

for the CBO!

performing databases
Your reliability. Our concern.

Still time for Execution Plan Example -2- ???

Example 2

Real-life example, October 25, 2017, Germany

performing databases
Your reliability. Our concern.

„Dynamic Data Model“ - joining into large table several times (to generate columns), part of MV refresh

Runtime 20min

See *BadPlan.txt*

Countermeasure: **Multi-Column Indexes** (to avoid table access)

See *BetterPlan.txt*

Example 2

Real-life example, October 25, 2017, Germany

performing databases
Your reliability. Our concern.

„Dynamic Data Model“ - joining a large table several times (to generate columns), part of MV refresh

Runtime <5min, way better, but still looking for improvement

Created MV logs on tables + Fast Refresh on demand

```
create materialized view MYMV refresh fast on commit
```

Hint: Use DBMS_MVIEW.EXPLAIN_MVIEW

Runtime improved to 2sec

Speaker

performing databases
Your reliability. Our concern.

- Martin Klier
- Solution Architect and Database Expert

- My focus
  - Performance Optimization
  - High Availability
  - Architecture DBMS

- Linux since 1997
- Oracle Database since 2003

ORACLE ACE Director

Speaker

performing databases
Your reliability. Our concern.

- Meet & Greet

il OUG
Israel Oracle User Group
ארגון משתמשי אורקל בישראל
Tel Aviv, January 2018

IOUG
Independent oracle users group
COLLABORATE
Las Vegas, April 2018

IT-Tage 2017
Frankfurt, December 2018

DOAG
Regionalgruppen
Fachkonferenzen

- Contact: martin.klier@performing-db.com
- Weblog: http://www.usn-it.de (English)

Performing Databases

performing databases
Your reliability. Our concern.

ORACLE Gold Partner

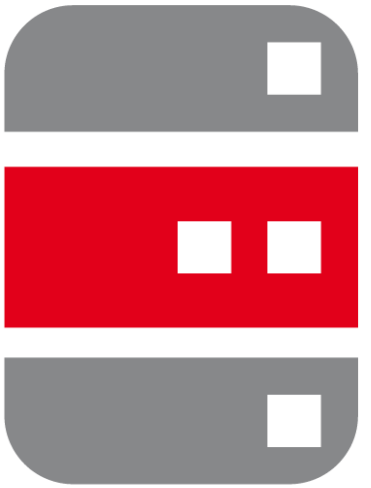
- Experts for Database Technology
  - Concept
  - Planning & Sizing
  - Licensing
  - Implementation and Troubleshooting
- Get in touch
  - Performing Databases GmbH
    Wiesauer Straße 27
    95666 Mitterteich, GERMANY
  - Web: http://www.performing-databases.com
  - Twitter: @PerformingDB

performing databases
Your reliability. Our concern.

Q & A

Download my Presentations and Whitepapers
http://www.performing-databases.com

performing databases



Your reliability. Our concern.