APOLLO

AUTOMATIC DETECTION AND DIAGNOSIS OF PERFORMANCE REGRESSIONS IN DATABASE SYSTEMS

Jinho Jung, Hong Hu, Joy Arulraj,
Taesoo Kim, Woonhak Kang*

Georgia Tech

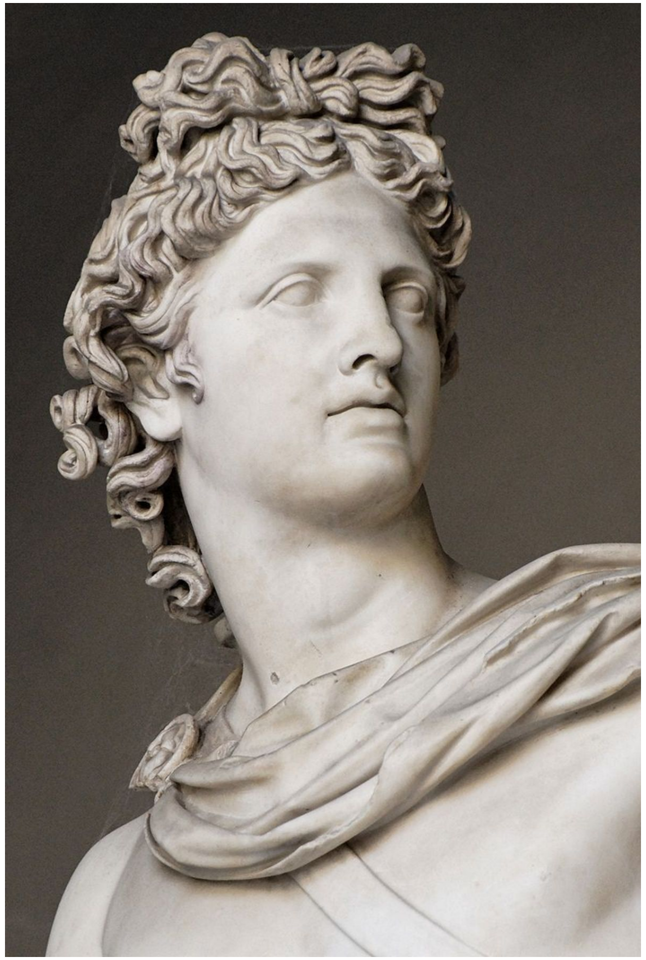
ebay*

APOLLO

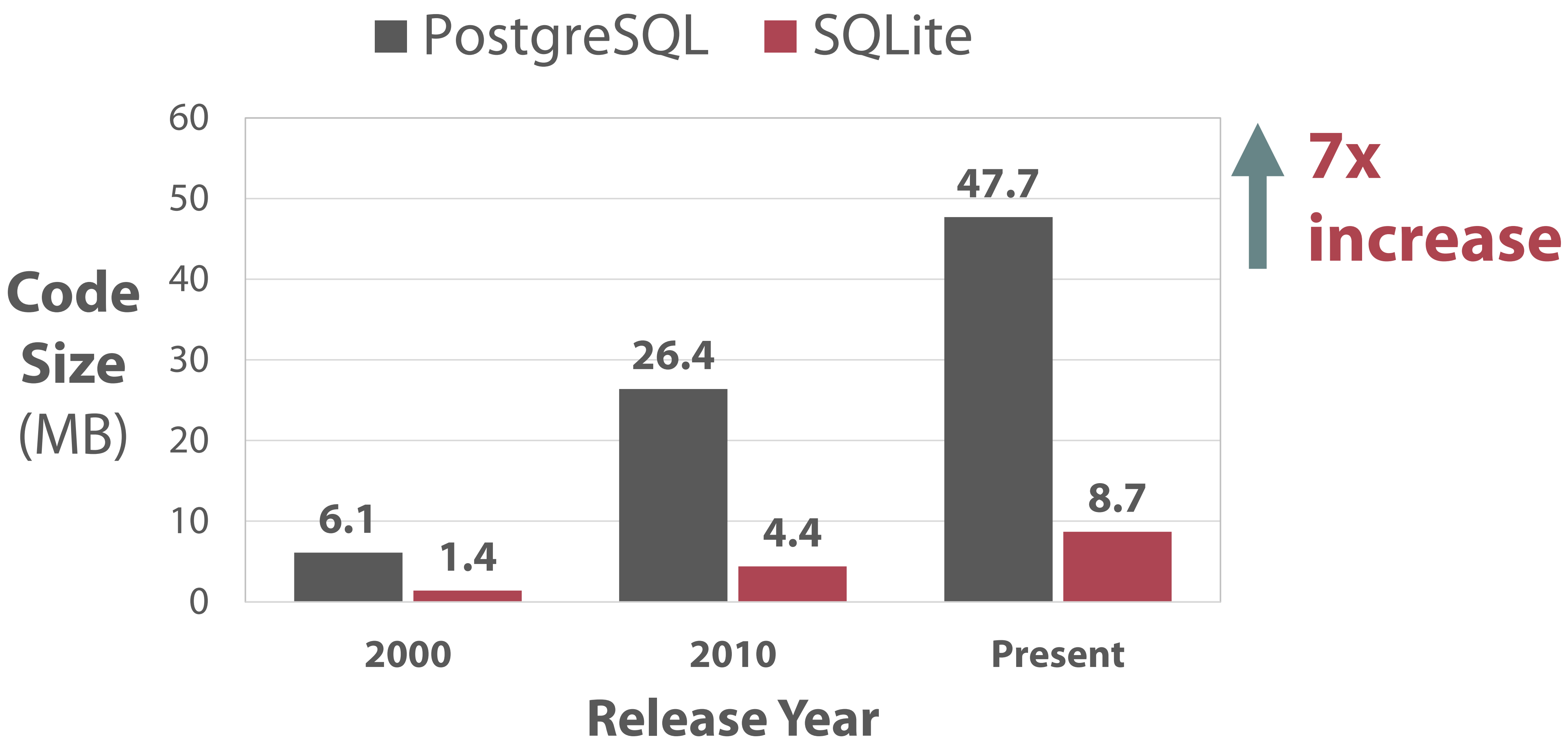
- Holistic toolchain for debugging DBMS

1 AUTOMATICALLY FIND SQL QUERIES EXHIBITING PERFORMANCE REGRESSIONS

2 AUTOMATICALLY DIAGNOSE THE ROOT CAUSE OF PERFORMANCE REGRESSIONS



MOTIVATION: DBMS COMPLEXITY



MOTIVATION: PERFORMANCE REGRESSIONS

CHALLENGING TO BUILD SYSTEM WITH *PREDICTABLE* PERFORMANCE

RE: Query became very slow after 9.6

From: "Alex Ignatov" <a(dot)ignatov(at)postgrespro(dot)ru>
To: "'Dmitry Shalashov'" <skaurus(at)gmail(dot)com>
Cc: <pgsql-performance(at)postgresql(dot)org>
Subject: RE: Query became very slow after 9.6 -> 10 upgrade
Date: 2017-11-22 14:44:18
Message-ID: 137da01d363a0$603cc970$20b65c50$@postgrespr
Views: Raw Message | Whole Thread | Download mbox | R
Thread: 2017-11-22 14:44:18 from "Alex Ignatov" <a(dot)ig
Lists: pgsql-hackers pgsql-performance

Query: https://pastebin.com/9b953tT7

It was running under 3 seconds (it's our default timeout) an

MySQL

Login / R

Developer Zone | Bugs Home | Report a bug | Statistics | Advanced search | Saved searches | Tags

Bug #87164	Queries running much slower in version 5.7 versus 5.6		
Submitted:	21 Jul 2017 16:28	Modified:	19 Apr 2018 5:46
Reporter:	Alok Pathak	Email Updates:	Subscribe
Status:	Won't fix	Impact on me:	None Affects Me
Category:	MySQL Server: Optimizer	Severity:	S3 (Non-critical)
Version:	5.7, 5.7.18, 5.7.19	OS:	CentOS (6 & 7)
Assigned to:		CPU Architecture:	Any
Tags:	regression		

View | Add Comment | Files | Developer | Edit Submission | View Progress Log | Contributions

[21 Jul 2017 16:28] Alok Pathak

Description:
After upgrading to MySQL 5.7, some queries are running very slow, taking abnormally long time in statistics s

When I run this query in 5.6, it finish in less than a second but on MySQL 5.7 it's taking approx 3 minutes.

MOTIVATION: PERFORMANCE REGRESSIONS

CHALLENGING TO BUILD SYSTEM WITH *PREDICTABLE* PERFORMANCE

- Scenario: User upgrades a DBMS installation
  - Query suddenly takes 10 times longer to execute
  - Due to unexpected interactions between different components
  - Refer to this behavior as a <u>performance regression</u>
- Performance regression can hurt user productivity
  - Can easily convert an interactive query to an overnight one

MOTIVATION: PERFORMANCE REGRESSIONS

```
SELECT R0.S_DIST_06
FROM PUBLIC.STOCK AS R0
WHERE (R0.S_W_ID < CAST(LEAST(0, 1) AS INT8))
```

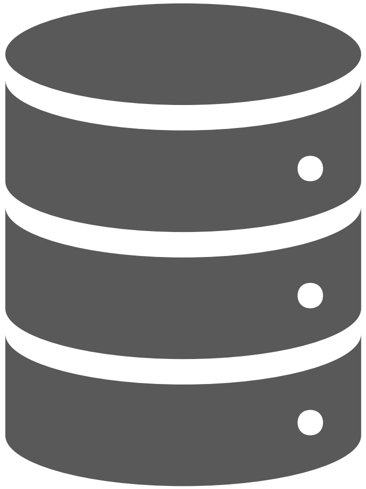
> 10,000x slowdown

LATEST VERSION OF POSTGRESQL

- Due to a recent optimizer update
  - New policy for choosing the scan algorithm
  - Resulted in over-estimating the number of rows in the table
  - Earlier version: Fast bitmap scan
  - Latest version: Slow sequential scan

MOTIVATION: DETECTING REGRESSIONS

1 HOW TO DISCOVER QUERIES EXHIBITING REGRESSIONS?

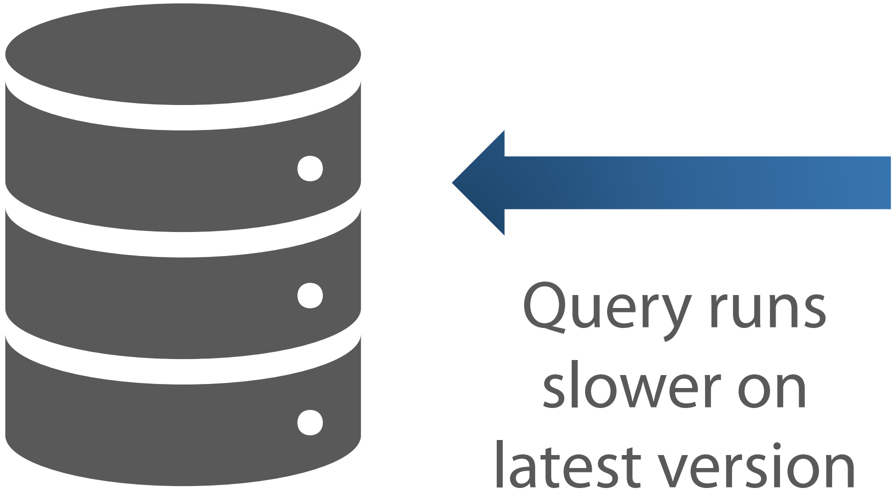


Query runs slower on latest version

```sql
SELECT NO FROM ORDER AS R0
WHERE EXISTS (
  SELECT CNT FROM SALES AS R1
  WHERE EXISTS (
    SELECT ID FROM HISTORY AS R2
    WHERE (R0.INFO IS NOT NULL));
```

MOTIVATION: REPORTING REGRESSIONS

2 HOW TO SIMPLIFY QUERIES FOR REPORTING REGRESSION?



Query runs slower on latest version

```
SELECT NO FROM ORDER AS R0
WHERE EXISTS (
  SELECT CNT FROM SALES AS R1
  WHERE EXISTS (
    SELECT ID FROM HISTORY AS R2
    WHERE (R0.INFO IS NOT NULL));
```

MOTIVATION: DIAGNOSING REGRESSIONS

3 HOW TO DIAGNOSE THE ROOT CAUSE OF THE REGRESSION?

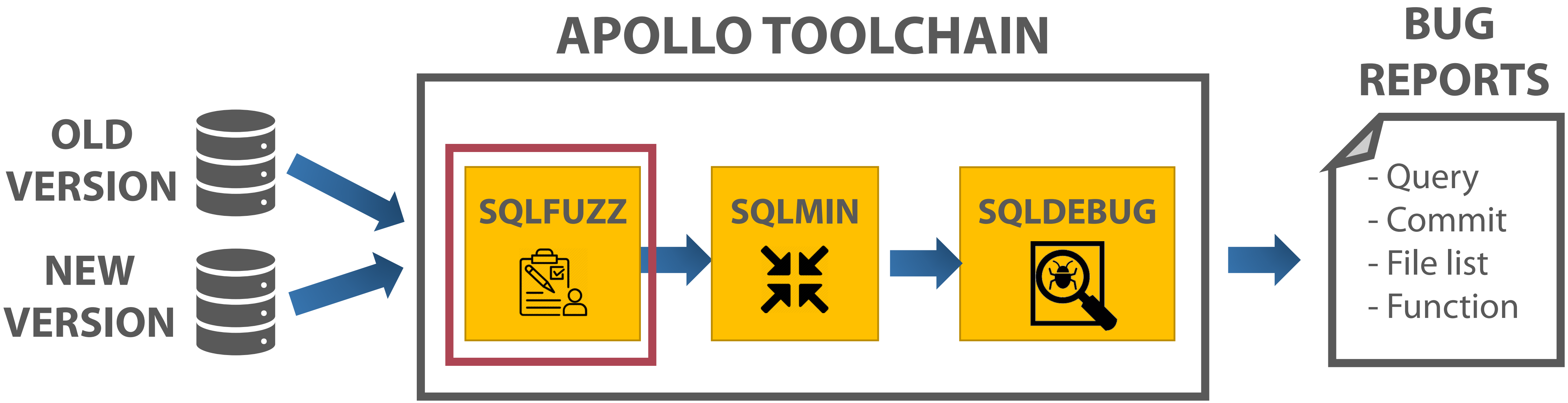
Query runs
slower on
latest version

```
SELECT NO FROM ORDER AS R0
WHERE EXISTS (
  SELECT CNT FROM SALES AS R1
  WHERE EXISTS (
    SELECT ID FROM HISTORY AS R2
    WHERE (R0.INFO IS NOT NULL));
```

APOLLO TOOLCHAIN

1 HOW TO DISCOVER QUERIES EXHIBITING REGRESSIONS?

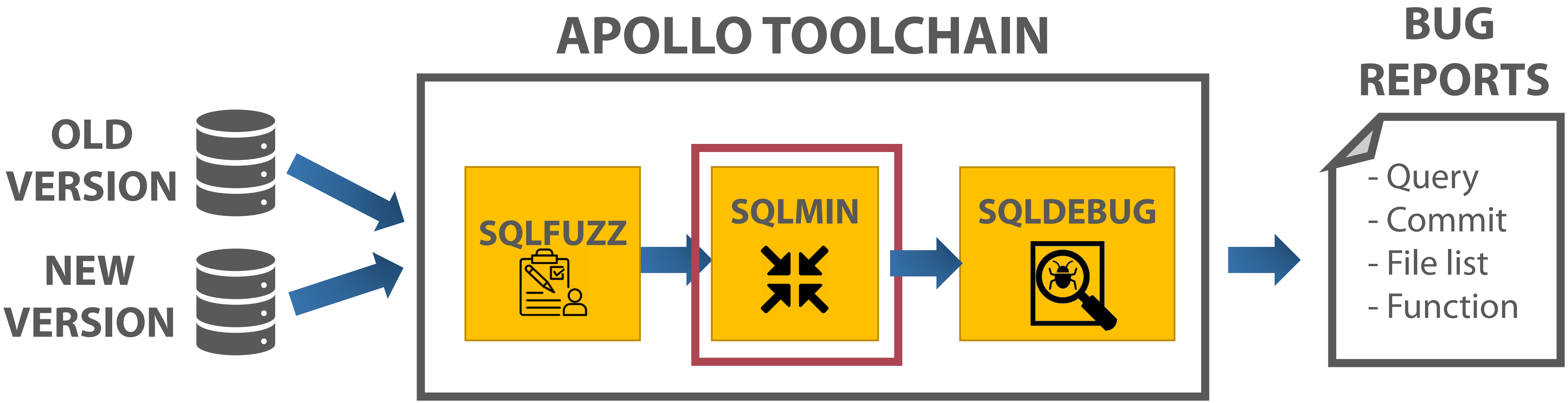
SQLFUZZ: FEEDBACK-DRIVEN FUZZING



APOLLO TOOLCHAIN

2 HOW TO SIMPLIFY QUERIES FOR REPORTING REGRESSION?

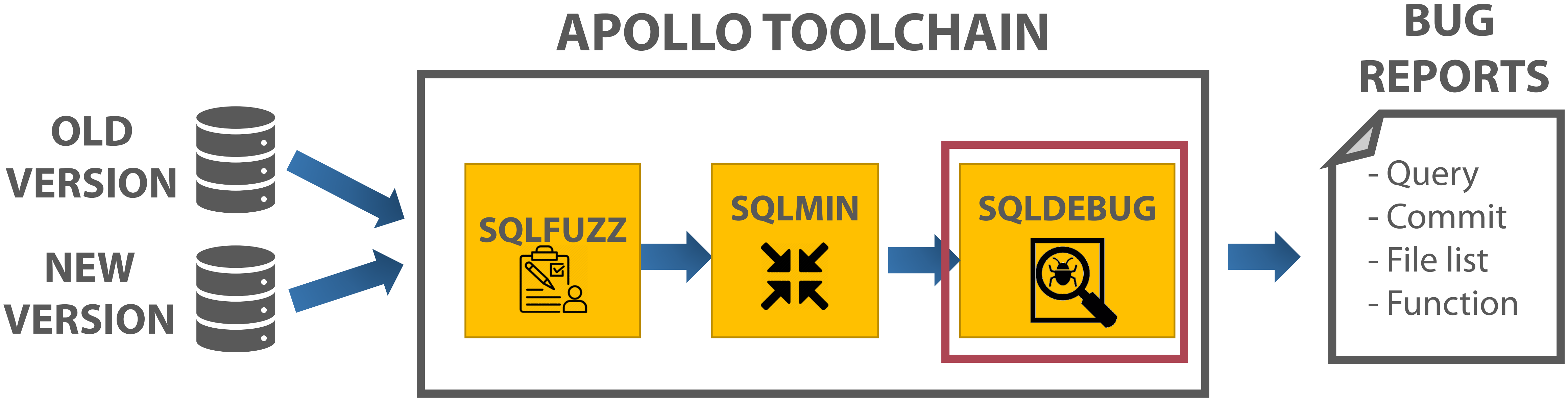
SQLMIN: BI-DIRECTIONAL QUERY REDUCTION ALGORITHMS



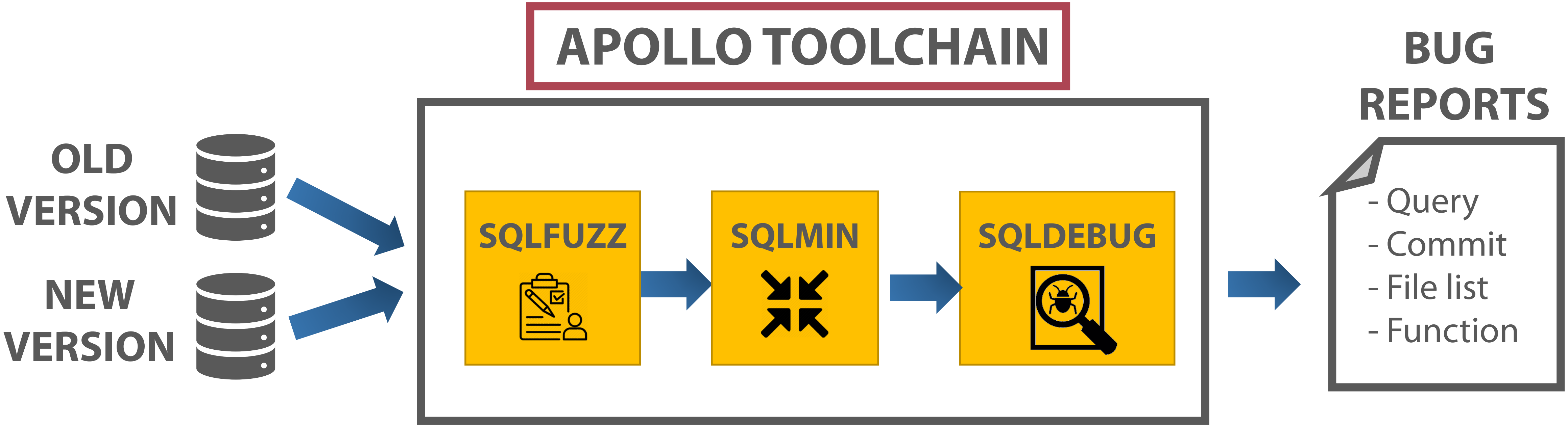
APOLLO TOOLCHAIN

3 HOW TO DIAGNOSE THE ROOT CAUSE OF THE REGRESSION?

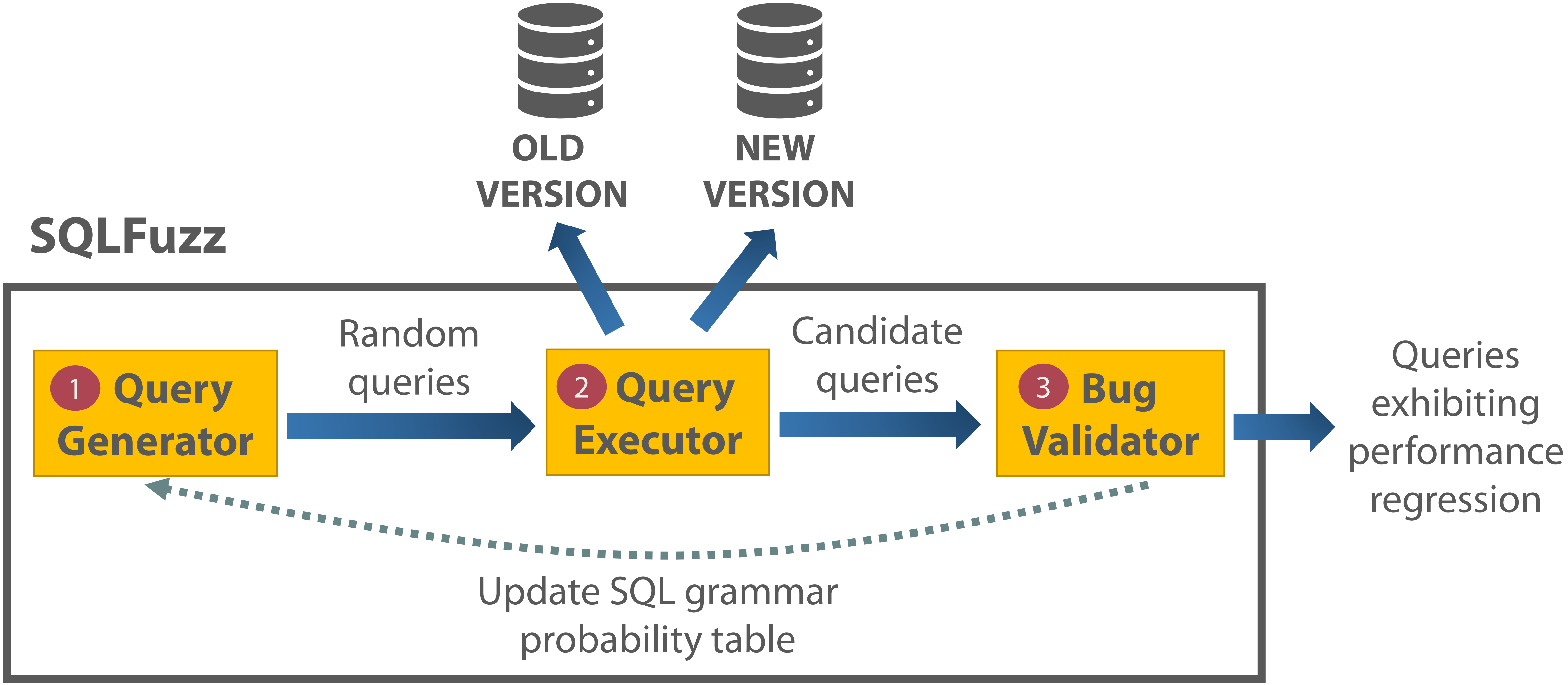
SQLDEBUG: STATISTICAL DEBUGGING + COMMIT BISECTION



TALK OVERVIEW

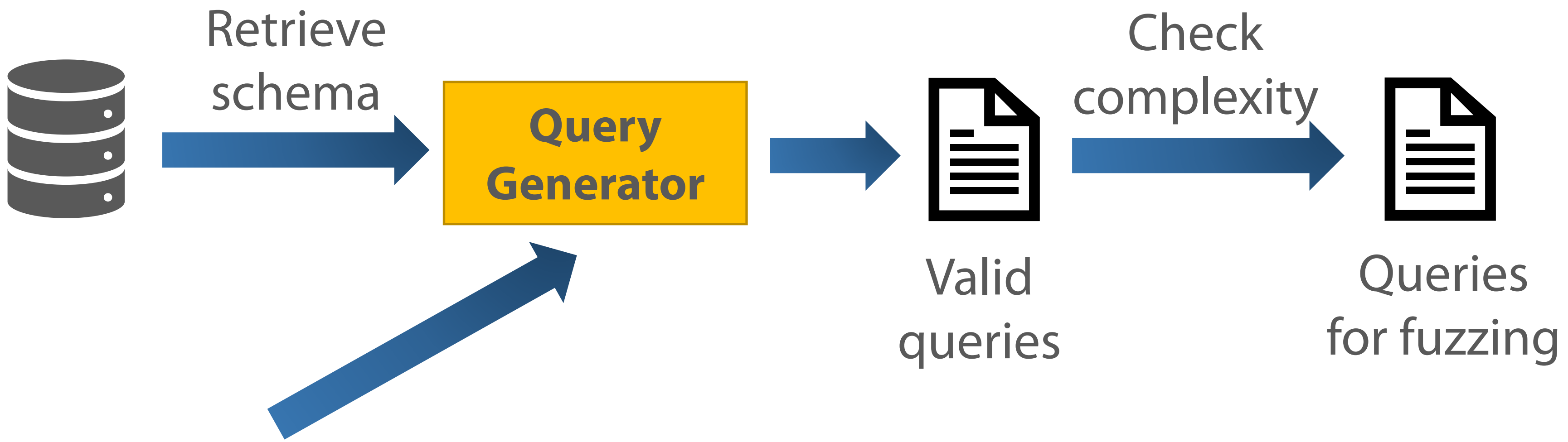


#1: SQLFUZZ — DETECTING REGRESSIONS



#1: SQLFUZZ — DETECTING REGRESSIONS

1 QUERY GENERATOR: RANDOM QUERY GENERATION

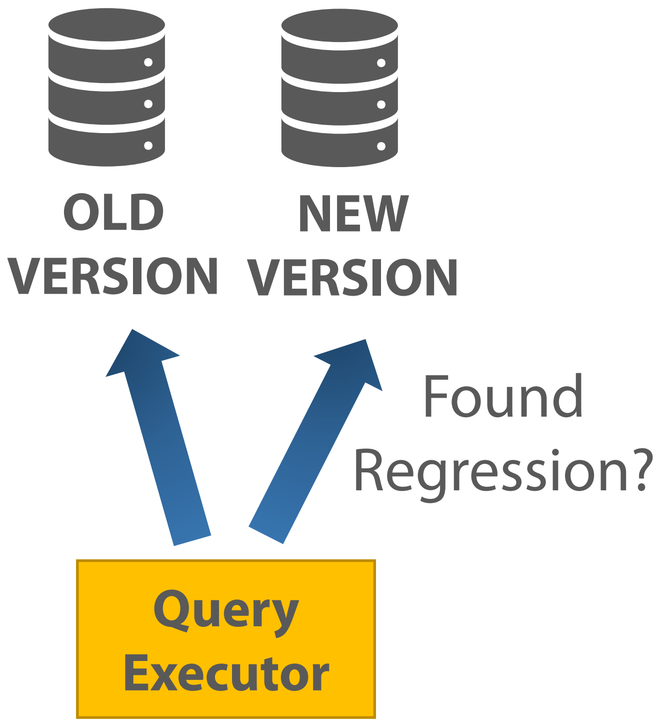


SELECT	0.3	LEFT JOIN	0.3
LIMIT	0.2	CAST	0.2

SQL grammar probability table

#1: SQLFUZZ — DETECTING REGRESSIONS

❷ QUERY EXECUTOR: FEEDBACK-DRIVEN FUZZING




```sql
SELECT R0.S_DIST_06
FROM PUBLIC.STOCK AS R0
WHERE (R0.S_W_ID <
CAST (LEAST(0, 1) AS INT8))
```

Update table

CASE		LEFT JOIN	
LIMIT		CAST	+0.1

SQL grammar probability table

#1: SQLFUZZ — DETECTING REGRESSIONS

3 REGRESSION VALIDATOR: REDUCING FALSE POSITIVES

Regression Query

Filtering rules

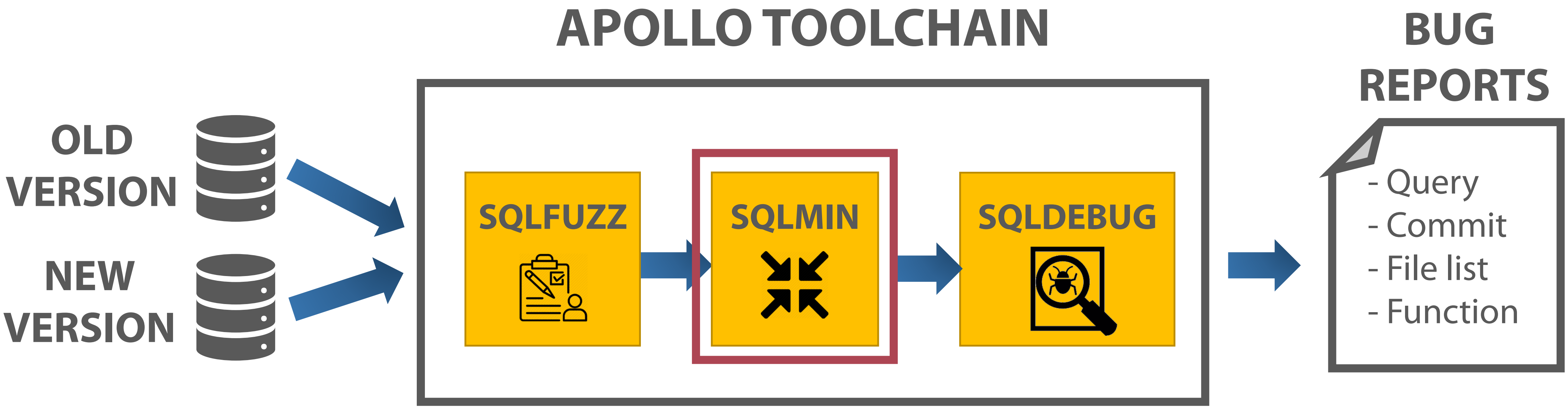
1	Non-deterministic behavior?
2	Non-executed plan?
3	Usage of catalog statistics?
4	Enough memory?
5	Limit statement?
6	Query is too complex?
7	…

Report

Developers

Updated filtering rules

TALK OVERVIEW



#2: SQLMIN — REPORTING REGRESSIONS

- Bottom-up Query Reduction
  - Extract valid sub-query

- Top-down Query Reduction
  - Iteratively removes unnecessary expressions

#2: SQLMIN — REPORTING REGRESSIONS

```
SELECT S1.C2
FROM (
  SELECT
    CASE WHEN EXISTS (
        SELECT S0.C0
        FROM ORDER AS R1
        WHERE ((S0.C0 = 10) AND (S0.C1 IS NULL))
    ) THEN S0.C0 END AS C2,
  FROM (
     SELECT R0.I_PRICE AS C0, R0.I_DATA AS C1,
        (SELECT ID FROM ITEM) AS C2
    FROM ITEM AS R0
    WHERE R0.PRICE IS NOT NULL
        OR (R0.PRICE IS NOT S1.C2)
    LIMIT 1000) AS S0) AS S1;
```

#2: SQLMIN — REPORTING REGRESSIONS

```
SELECT S1.C2
FROM (
  SELECT
    CASE WHEN EXISTS (
        SELECT S0.C0
        FROM ORDER AS R1
        WHERE ((S0.C0 = 10) AND (S0.C1 IS NULL))
    ) THEN S0.C0 END AS C2,
  FROM (
     SELECT R0.I_PRICE AS C0, R0.I_DATA AS C1,
        (SELECT ID FROM ITEM) AS C2
    FROM ITEM AS R0
    WHERE R0.PRICE IS NOT NULL
       [redacted]
    LIMIT 1000) AS S0) AS S1;
```

BOTTOM-UP REDUCTION

EXTRACT SUB-QUERY

Remove dependencies

#2: SQLMIN — REPORTING REGRESSIONS

```
SELECT S1.C2
FROM (
  SELECT
    CASE WHEN EXISTS (
        SELECT S0.C0
        FROM ORDER AS R1
        WHERE ((S0.C0 = 10) [redacted]
    ) THEN S0.C0 END AS C2,
  FROM (
     SELECT R0.I_PRICE AS C0, [redacted]
        [redacted]
    FROM ITEM AS R0
    WHERE R0.PRICE IS NOT NULL
        [redacted]
   [redacted] AS S0) AS S1;
```

TOP-DOWN REDUCTION

REMOVE ELEMENTS

Remove condition

Remove columns

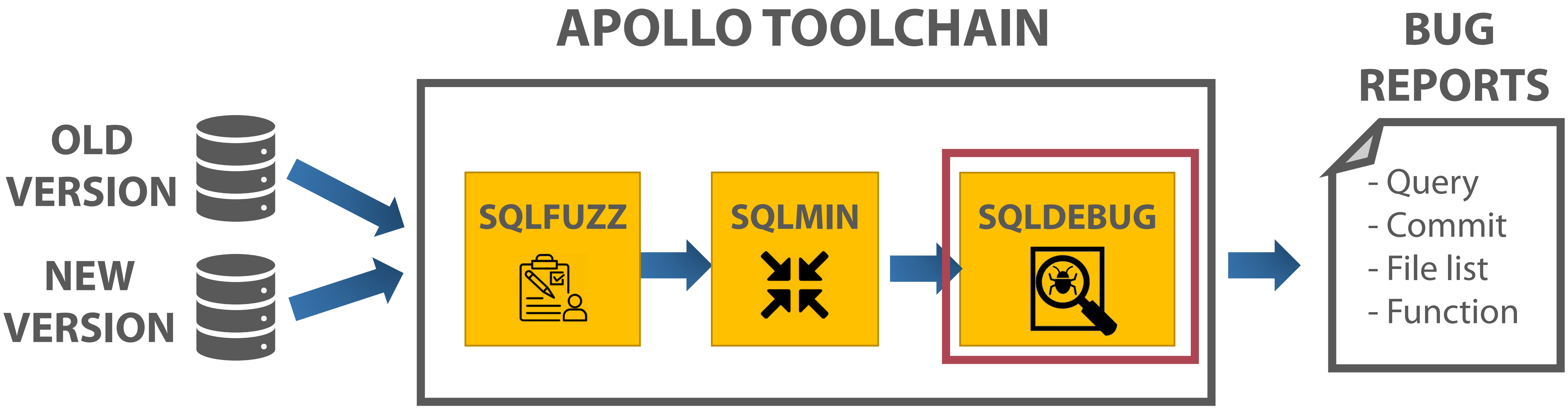
Remove sub-queries

Remove clause

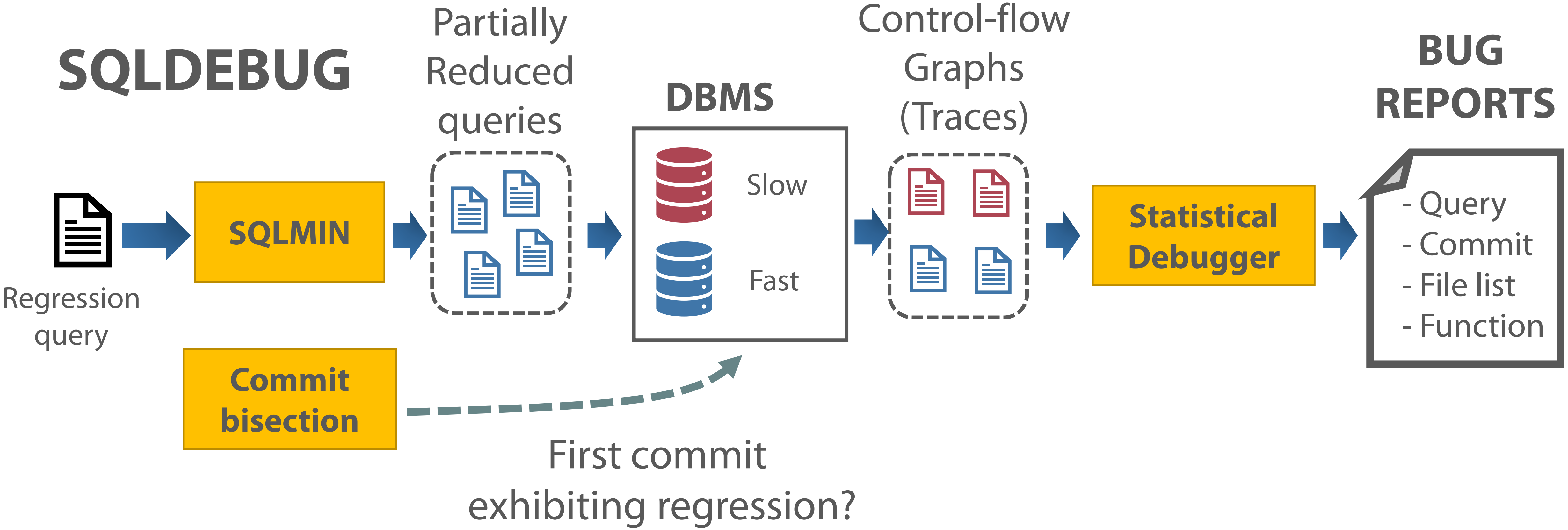
#2: SQLMIN — REPORTING REGRESSIONS

```
SELECT
  CASE WHEN EXISTS (
      SELECT S0.C0
      FROM ORDER AS R1
      WHERE ((S0.C0 = 10))
  ) THEN S0.C0 END AS C2,
FROM (
   SELECT R0.I_PRICE AS C0,
   FROM ITEM AS R0
   WHERE R0.PRICE IS NOT NULL) AS S0)
AS S1;
```

TALK OVERVIEW

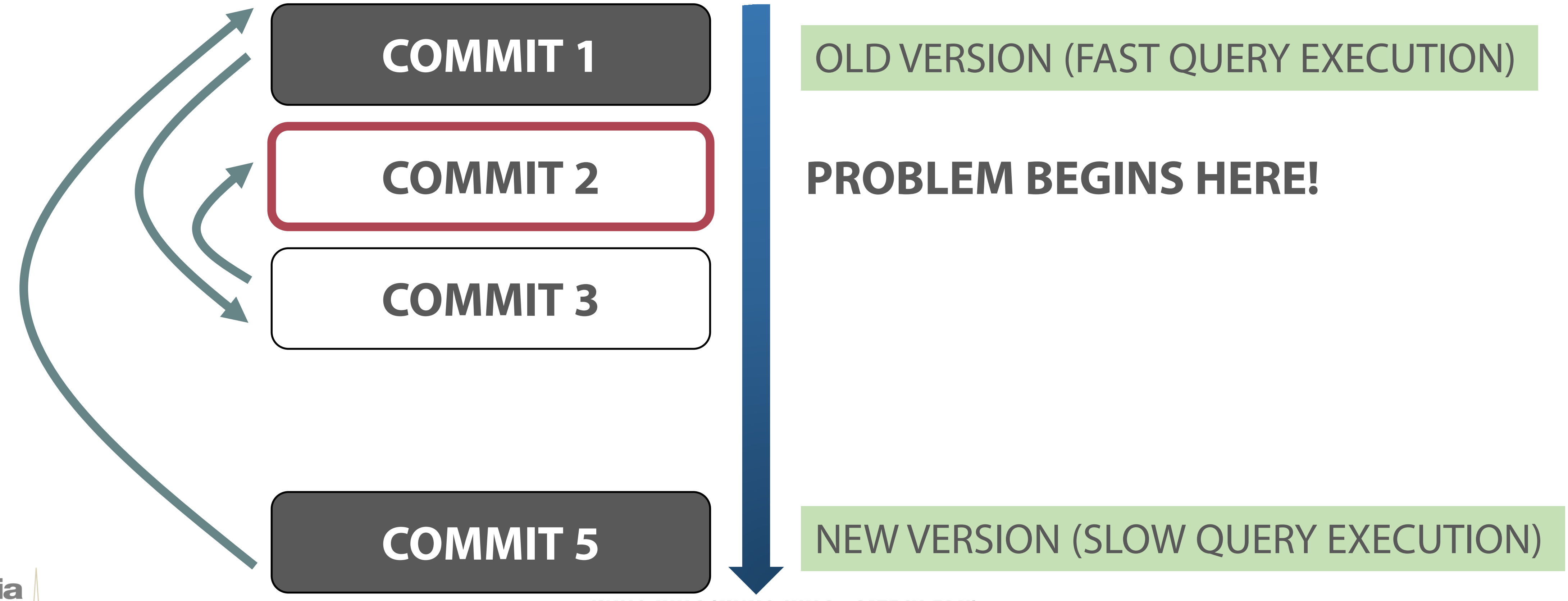


#3: SQLDEBUG — DIAGNOSING REGRESSIONS



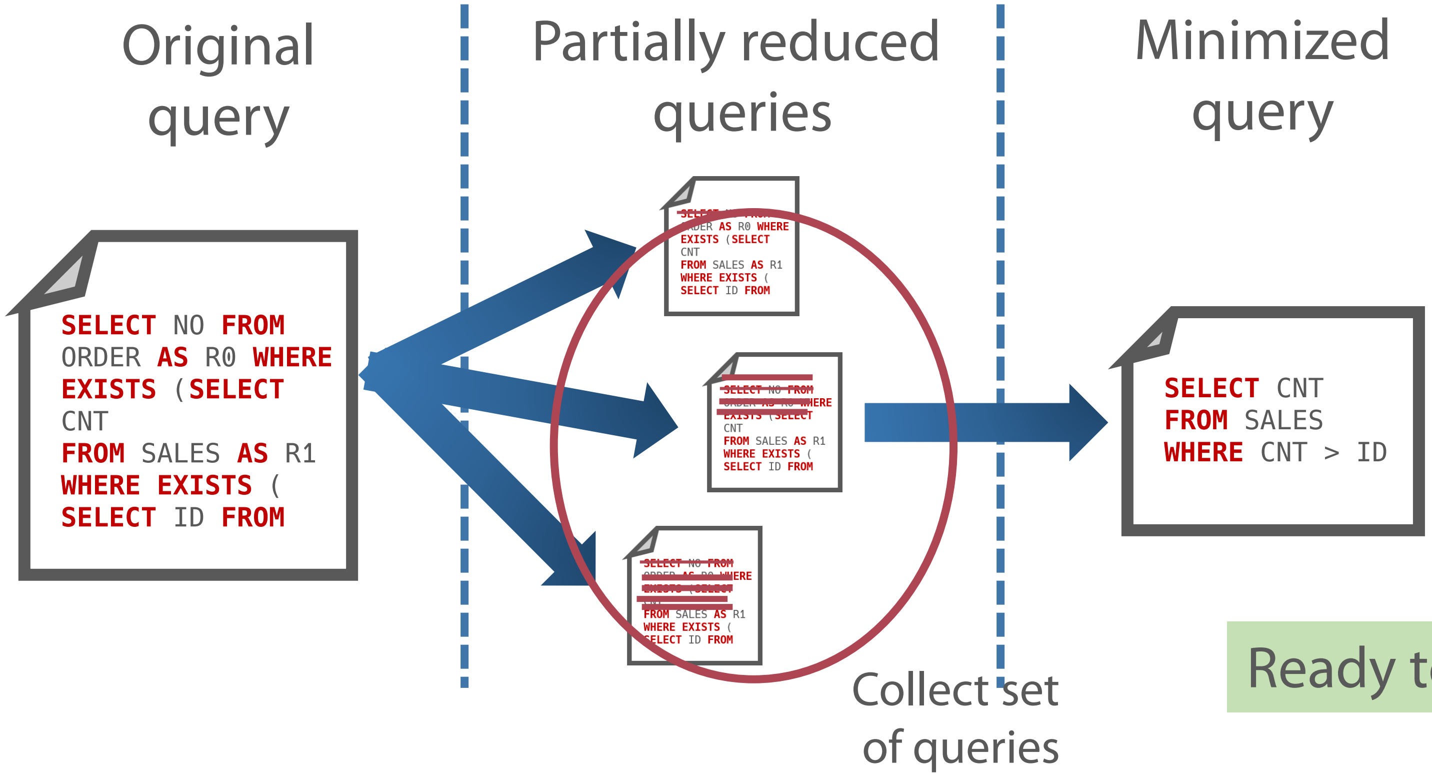
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

1 COMMIT BISECTION: FIND EARLIEST PROBLEMATIC COMMIT



#3: SQLDEBUG — DIAGNOSING REGRESSIONS

2 QUERY REDUCTION: PARTIALLY REDUCED QUERIES



Ready to use statistical debugging?

#3: SQLDEBUG — DIAGNOSING REGRESSIONS

3 CONTROL-FLOW GRAPH COMPARISON: ALIGN TRACES

Functions

OLD VERSION

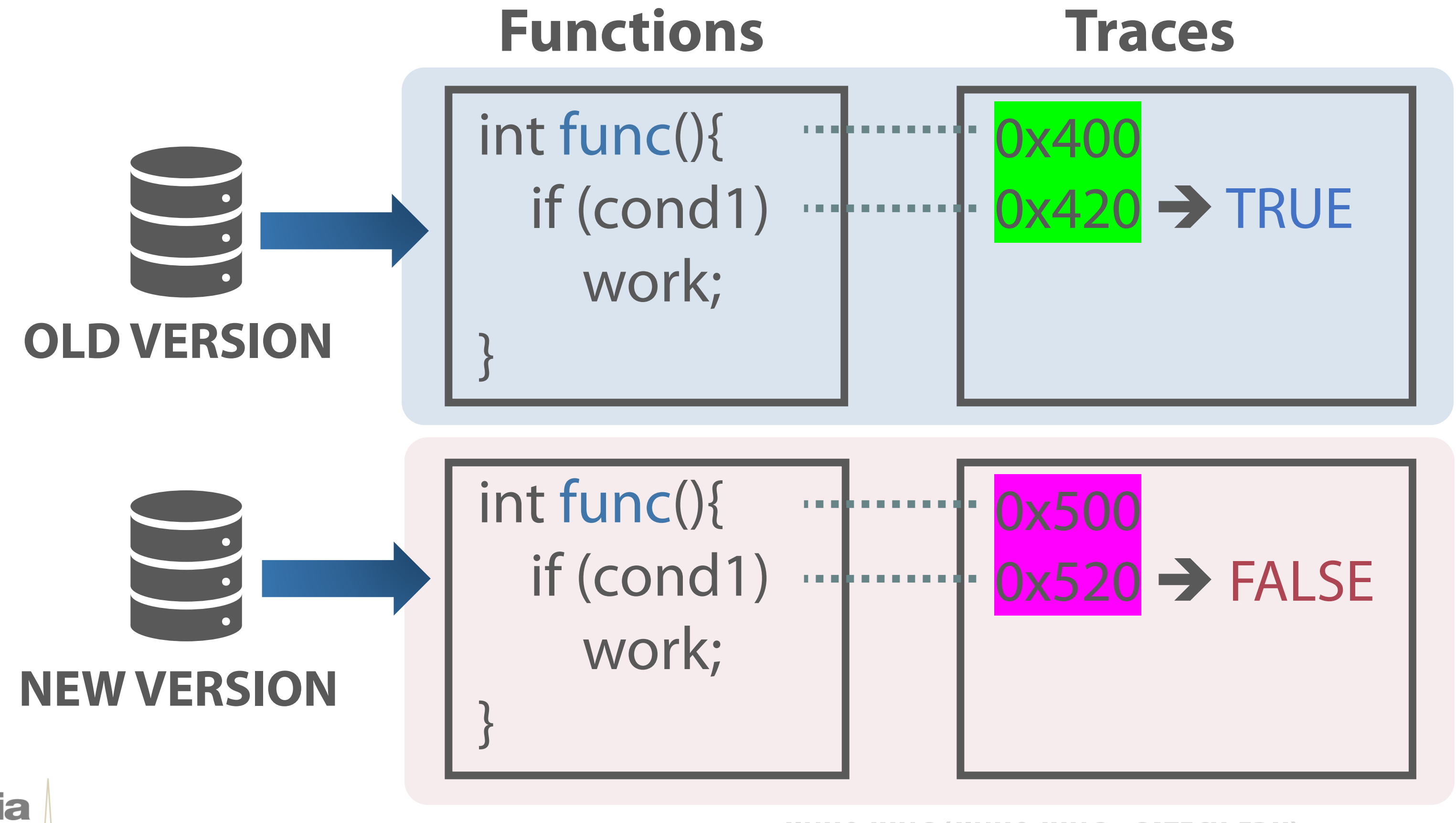
```
int func(){
   if (cond1)
      work;
}
```

NEW VERSION

```
int func(){
   if (cond1)
      work;
}
```

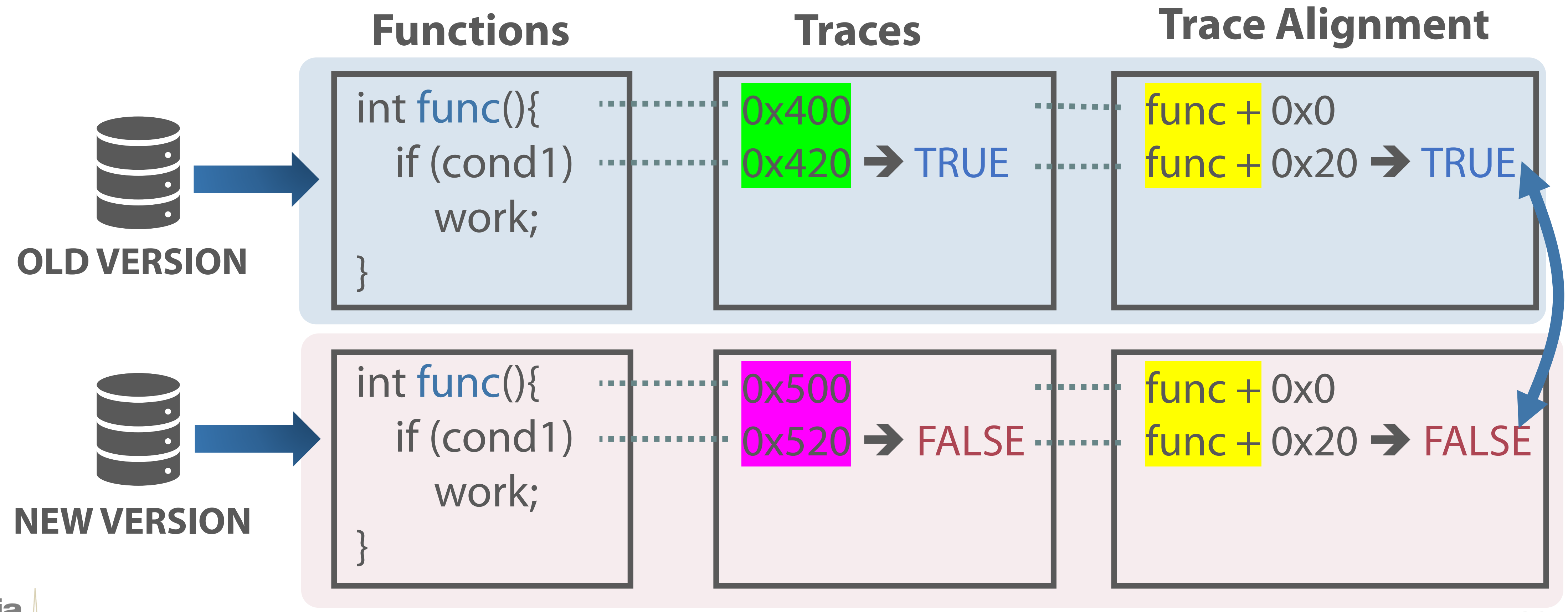
#3: SQLDEBUG — DIAGNOSING REGRESSIONS

3 CONTROL-FLOW GRAPH COMPARISON: ALIGN TRACES



#3: SQLDEBUG — DIAGNOSING REGRESSIONS

3 CONTROL-FLOW GRAPH COMPARISON: ALIGN TRACES



#3: SQLDEBUG — DIAGNOSING REGRESSIONS

4 STATISTICAL DEBUGGING: FAST AND SLOW QUERY TRACES

Fast query execution traces

PRED.	RESULT
1	TAKEN
2	TAKEN

Slow query execution traces

PRED.	RESULT
1	TAKEN
2	NOT TAKEN

Statistical model

#3: SQLDEBUG — DIAGNOSING REGRESSIONS

4 STATISTICAL DEBUGGING: FAST AND SLOW QUERY TRACES

Fast query execution traces

PRED.	RESULT
1	TAKEN
2	TAKEN

Slow query execution traces

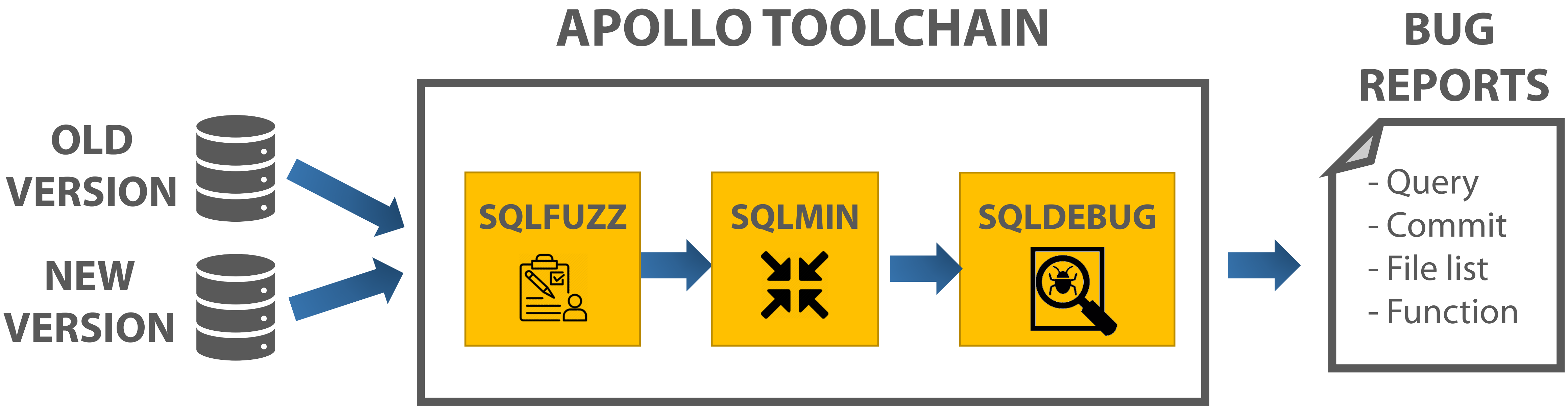
PRED.	RESULT
1	TAKEN
2	NOT TAKEN

Statistical model

RANK	FILE	FUNCTION	LINE
1	foo.c	bar()	2
…	…	…	…

Final report

RECAP



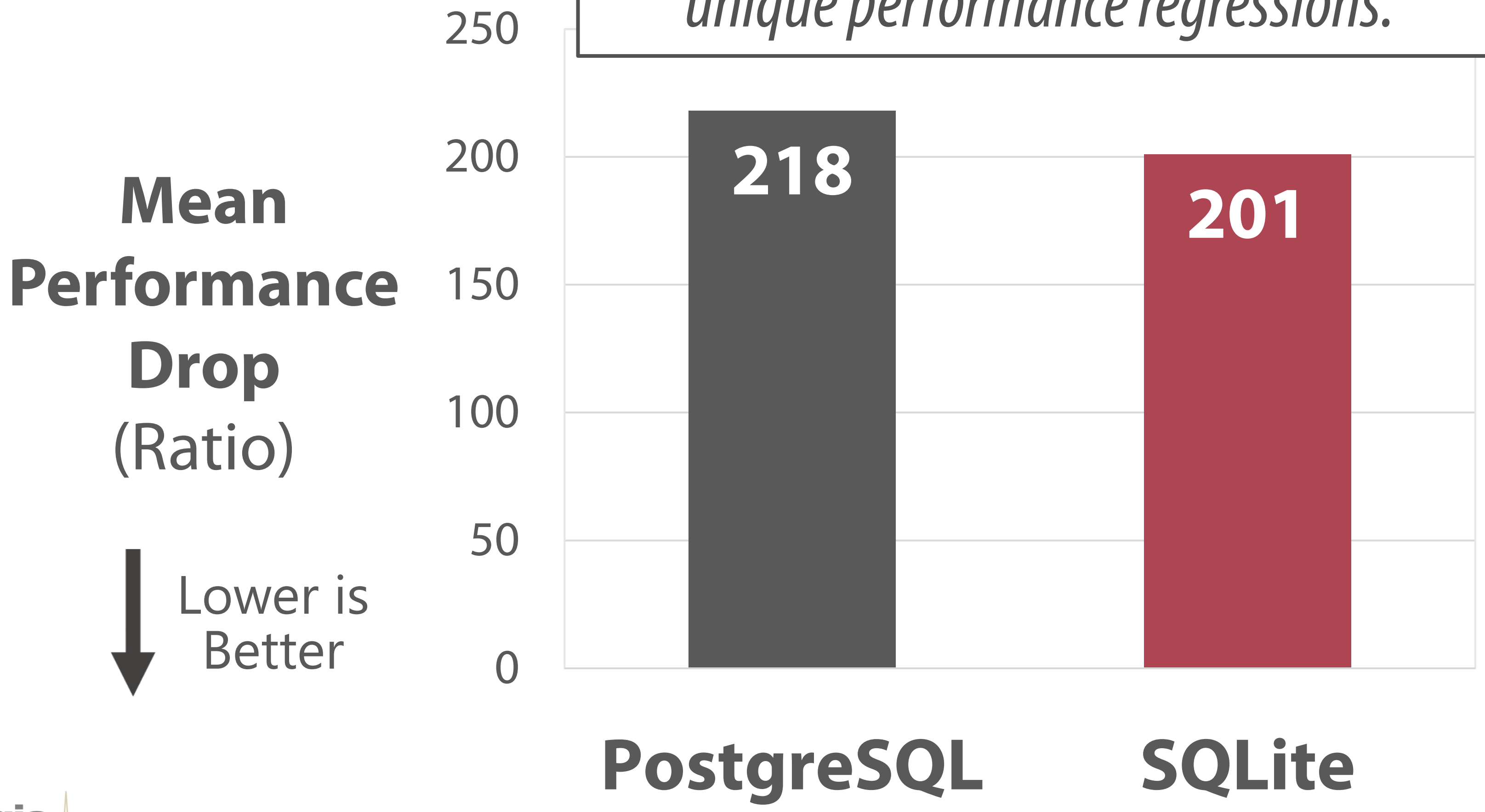
APOLLO TOOLCHAIN
BUG REPORTS
OLD VERSION
NEW VERSION
SQLFUZZ
SQLMIN
SQLDEBUG
- Query
- Commit
- File list
- Function

EVALUATION

- Tested database systems
  - PostgreSQL, SQLite
- Binary instrumentation to get control flow graphs
  - DynamoRIO instrumentation tool
- Evaluation
  - Efficacy of SQLFuzz in detecting regressions?
  - Efficacy of SQLMin in reducing queries?
  - Accuracy of SQLDebug in diagnosing regressions?

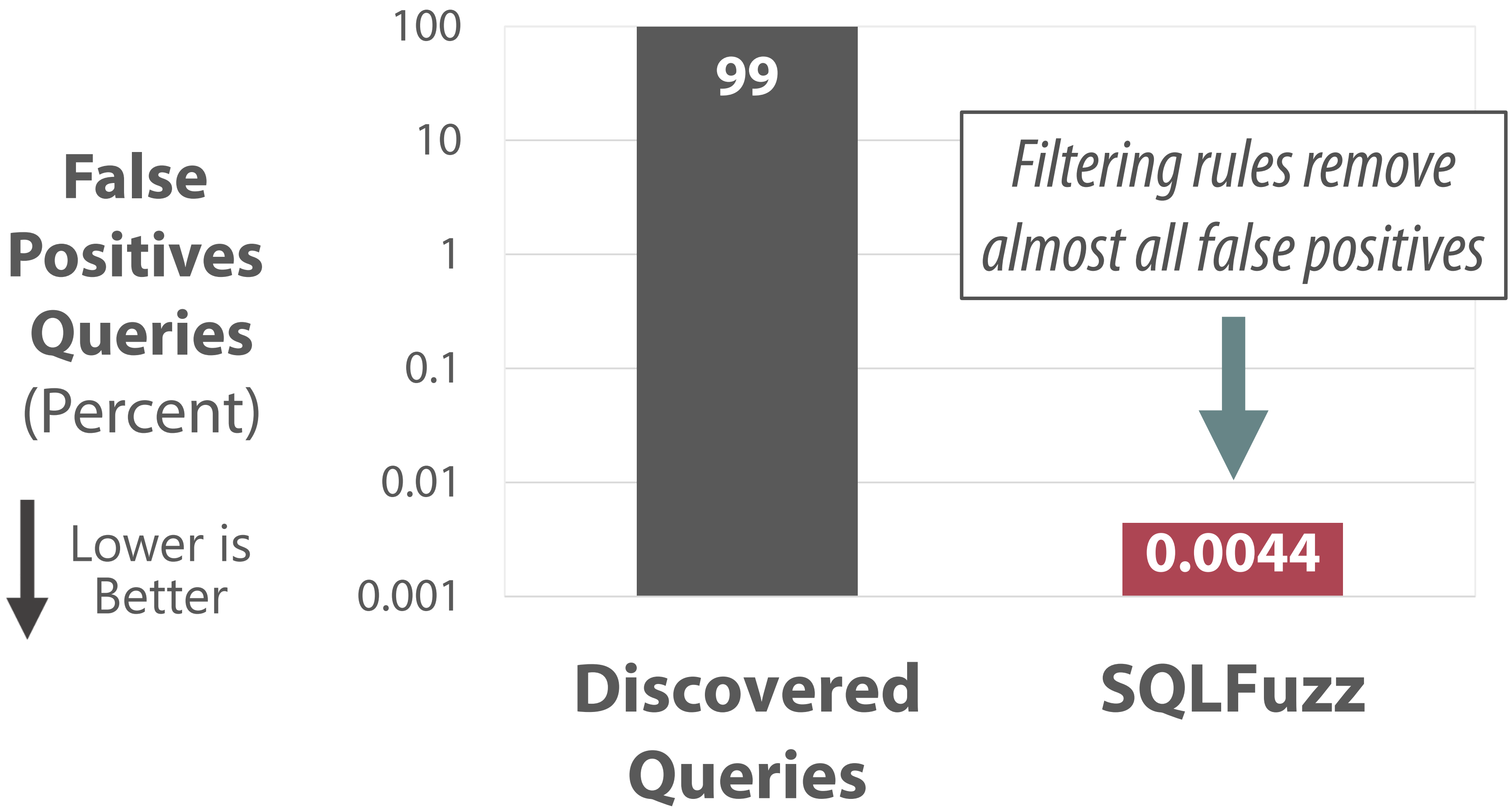
#1: SQLFUZZ — DETECTING REGRESSIONS

Discovered 10 previously unknown, unique performance regressions.

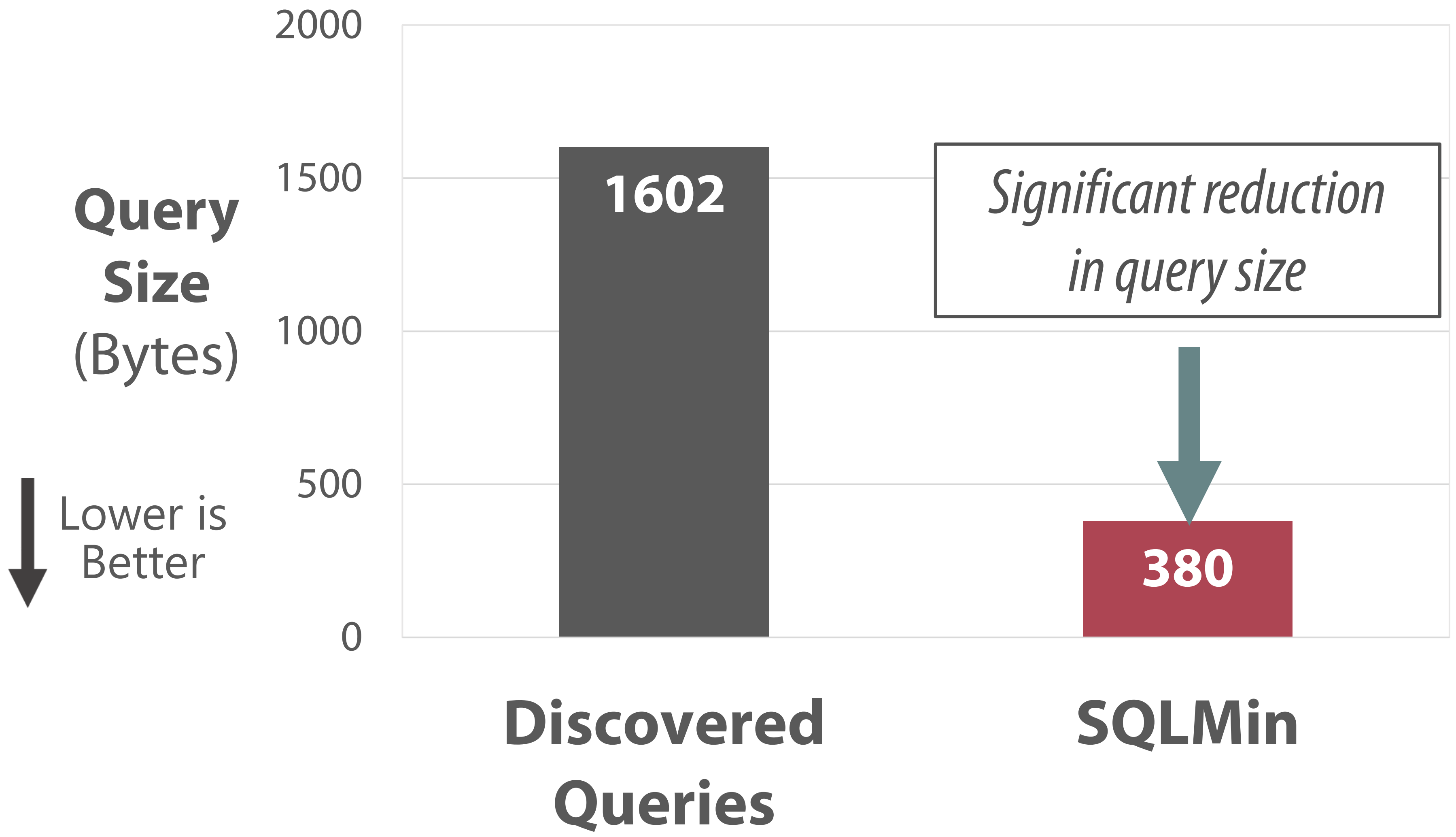


200x performance drop

#1: SQLFUZZ — FALSE POSITIVES

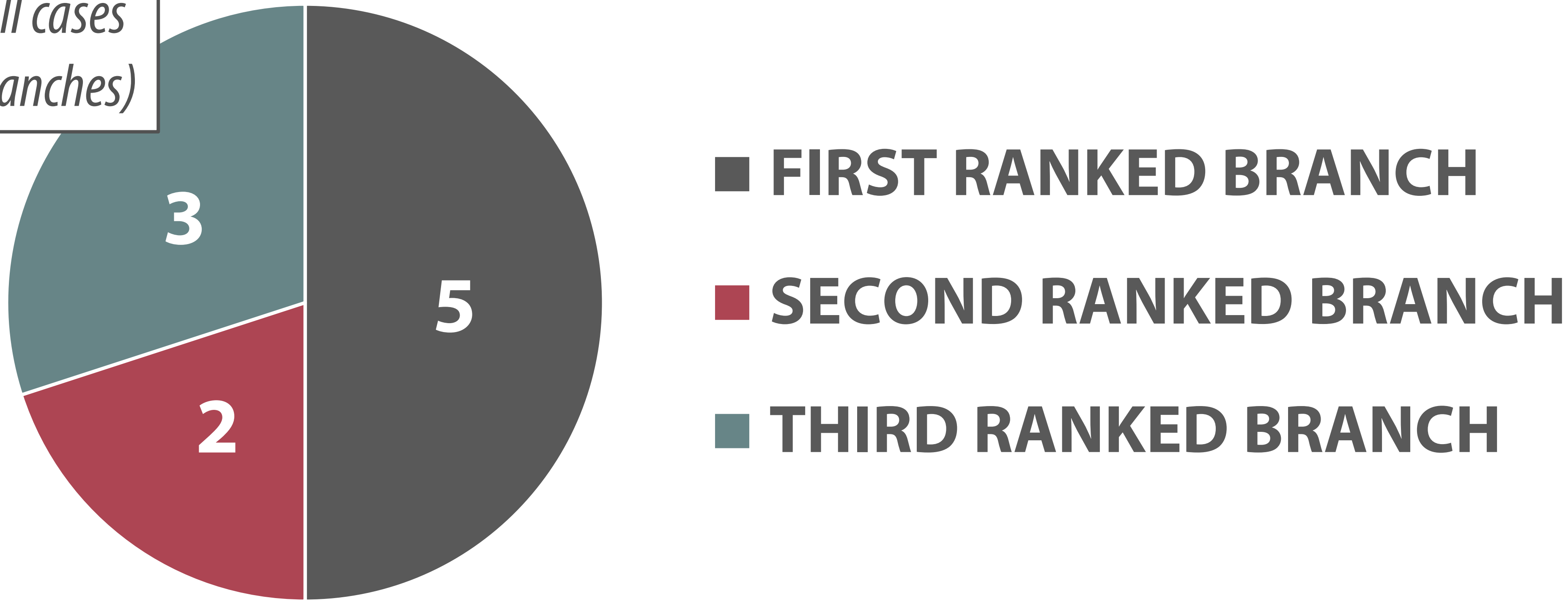


#2: SQLMIN — REPORTING REGRESSIONS



#3: SQLDEBUG — DIAGNOSING REGRESSIONS

Branch related to root cause
Correctly identified in all cases
(within top-3 ranked branches)



10 regressions

CASE STUDY #1: OPTIMIZER UPDATE

```
SELECT COUNT (*)
FROM (SELECT R0.ID
  FROM CUSTOMER AS R0 LEFT JOIN STOCK AS R1
  ON (R0.STREET = R1.DIST)
  WHERE R1.DIST IS NOT NULL) AS S0
WHERE EXISTS (SELECT ID FROM CUSTOMER);
```

> 1000x slow down

LATEST VERSION OF SQLITE

- Due to a bug fix (for a correctness bug)
  - Breaks query optimization
  - Optimizer no longer transforms the LEFT JOIN operator

CASE STUDY #2: EXECUTION ENGINE UPDATE

```
SELECT R0.ID FROM ORDER AS R0
WHERE EXISTS (SELECT COUNT(*)
  FROM (SELECT DISTINCT R0.ENTRY
    FROM CUSTOMER AS R1
    WHERE (FALSE)) AS S1);
```

3x slow down

LATEST VERSION OF POSTGRESQL

- Hashed aggregation executor update
  - Resulted in redundantly building hash tables

CONCLUSION

- APOLLO (v1.0)
  - Toolchain for detecting & diagnosing regressions
  - Open-sourced: https://github.com/sslab-gatech/apollo
- Adding support for other types of bugs (v2.0)
  - Correctness bugs
  - Performance bugs
  - Database corruption

CONCLUSION

- Interested in integrating APOLLO with more DBMSs
  - Discovered > 5 performance regressions in CockroachDB
  - Improve the toolchain based on developer feedback
- Automation will help reduce labor of developing DBMSs
  - Developers get to focus on more important problems

ACKNOWLEDGEMENTS

Supported by:

Developers:

END

JINHO.JUNG@GATECH.EDU