



The
University
Of
Sheffield.



ALLEGHENY
COLLEGE

Virtual Mutation Analysis of Relational Database Schemas

Phil McMinn University of Sheffield

Gregory M. Kapfhammer Allegheny College

Chris J. Wright University of Sheffield

Relational Databases – Why Should We (Still) Care?

A vital component of many software systems

Despite the wave of interest in “NoSQL” technologies,
Relational Databases are still popular (and faster)

For developers: schemas provide self-documentation

Relational Databases – Why Should We (Still) Care?

A vital component of many software systems

Despite the wave of interest in “NoSQL” technologies,
Relational Databases are still popular (and faster)

For developers: schemas provide self-documentation

Relational Databases are still
important, popular and relevant

A Relational Database Schema

```
CREATE TABLE Station (  
    ID INTEGER PRIMARY KEY,  
    CITY CHAR(20),  
    STATE CHAR(2),  
    LAT_N INTEGER NOT NULL  
        CHECK (LAT_N BETWEEN 0 and 90),  
    LONG_W INTEGER NOT NULL  
        CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);  
  
CREATE TABLE Stats (  
    ID INTEGER REFERENCES STATION(ID),  
    MONTH INTEGER NOT NULL  
        CHECK (MONTH BETWEEN 1 AND 12),  
    TEMP_F INTEGER NOT NULL  
        CHECK (TEMP_F BETWEEN 80 AND 150),  
    RAIN_I INTEGER NOT NULL  
        CHECK (RAIN_I BETWEEN 0 AND 100),  
    PRIMARY KEY (ID, MONTH)  
);
```

A Relational Database Schema

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

Table

```
CREATE TABLE Stats (  
  ID INTEGER REFERENCES STATION(ID),  
  MONTH INTEGER NOT NULL  
    CHECK (MONTH BETWEEN 1 AND 12),  
  TEMP_F INTEGER NOT NULL  
    CHECK (TEMP_F BETWEEN 80 AND 150),  
  RAIN_I INTEGER NOT NULL  
    CHECK (RAIN_I BETWEEN 0 AND 100),  
  PRIMARY KEY (ID, MONTH)  
);
```

A Relational Database Schema

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

Table

```
CREATE TABLE Stats (  
  ID INTEGER REFERENCES STATION(ID),  
  MONTH INTEGER NOT NULL  
    CHECK (MONTH BETWEEN 1 AND 12),  
  TEMP_F INTEGER NOT NULL  
    CHECK (TEMP_F BETWEEN 80 AND 150),  
  RAIN_I INTEGER NOT NULL  
    CHECK (RAIN_I BETWEEN 0 AND 100),  
  PRIMARY KEY (ID, MONTH)  
);
```

Column and
data type ←

Integrity Constraints

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

Prevent invalid data
being entered into the
database

```
CREATE TABLE Stats (  
  ID INTEGER REFERENCES STATION(ID),  
  MONTH INTEGER NOT NULL  
    CHECK (MONTH BETWEEN 1 AND 12),  
  TEMP_F INTEGER NOT NULL  
    CHECK (TEMP_F BETWEEN 80 AND 150),  
  RAIN_I INTEGER NOT NULL  
    CHECK (RAIN_I BETWEEN 0 AND 100),  
  PRIMARY KEY (ID, MONTH)  
);
```

Encode domain logic

Integrity Constraints

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

Prevent invalid data
being entered into the
database

```
CREATE TABLE Stats (  
  ID INTEGER REFERENCES STATION(ID),  
  MONTH INTEGER NOT NULL  
    CHECK (MONTH BETWEEN 1 AND 12),  
  TEMP_F INTEGER NOT NULL  
    CHECK (TEMP_F BETWEEN 80 AND 150),  
  RAIN_I INTEGER NOT NULL  
    CHECK (RAIN_I BETWEEN 0 AND 100),  
  PRIMARY KEY (ID, MONTH)  
);
```

Encode domain logic

Testing the Schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', 30, 98);
```

Correctly **accepted** by the schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', NULL, 98);
```

Correctly **rejected** by the schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', 91, 98);
```

Correctly **rejected** by the schema

Testing the Schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', 30, 98);
```

Correctly **accepted** by the schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', NULL, 98);
```

Correctly **rejected** by the schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', 91, 98);
```

Correctly **rejected** by the schema

Testing the Schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', 30, 98);
```

Correctly **accepted** by the schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', NULL, 98);
```

Correctly **rejected** by the schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', 91, 98);
```

Correctly **rejected** by the schema

Testing the Schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', 30, 98);
```

Correctly **accepted** by the schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', NULL, 98);
```

Correctly **rejected** by the schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES (1, 'Austin', 'TX', 91, 98);
```

Correctly **rejected** by the schema

Why Do We Need to Do This?

Why Do We Need to Do This?

To trap common errors when designing a schema

For example: lack of uniqueness property on usernames, out of range values

Why Do We Need to Do This?

To trap common errors when designing a schema

For example: lack of uniqueness property on usernames, out of range values

To test development behaviour vs deployment

DBMSs have subtly different behaviors

Why Do We Need to Do This?

To trap common errors when designing a schema

For example: lack of uniqueness property on usernames, out of range values

To test development behaviour vs deployment

DBMSs have subtly different behaviors

Nobody throws away a database of data

To test the success of database migrations

Why Do We Need to Do This?

To trap common errors when designing a schema

For example: lack of uniqueness property on usernames, out of range values

To test development behaviour vs deployment

DBMSs have subtly different behaviors

Nobody throws away a database of data

To test the success of database migrations

Industry advice

Destroying database consistency can have huge cost implications

Mutation Analysis

Once a test suite has been created, its fault finding capability can be estimated with [mutation analysis](#).

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

For relational database schema testing, mutants are created by making small changes to the schema

Mutation Analysis

Once a test suite has been created, its fault finding capability can be estimated with [mutation analysis](#).

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

For relational database schema testing, mutants are created by making small changes to the schema

Mutation Analysis

Once a test suite has been created, its fault finding capability can be estimated with [mutation analysis](#).

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
  CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
  CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

For relational database schema testing, mutants are created by making small changes to the schema

Mutation Analysis

Once a test suite has been created, its fault finding capability can be estimated with [mutation analysis](#).

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20) UNIQUE,  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
  CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
  CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

For relational database schema testing, mutants are created by making small changes to the schema

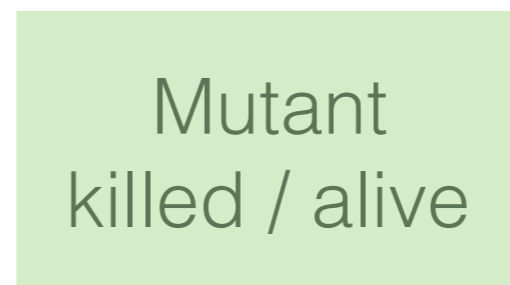
Mutation Analysis is Costly



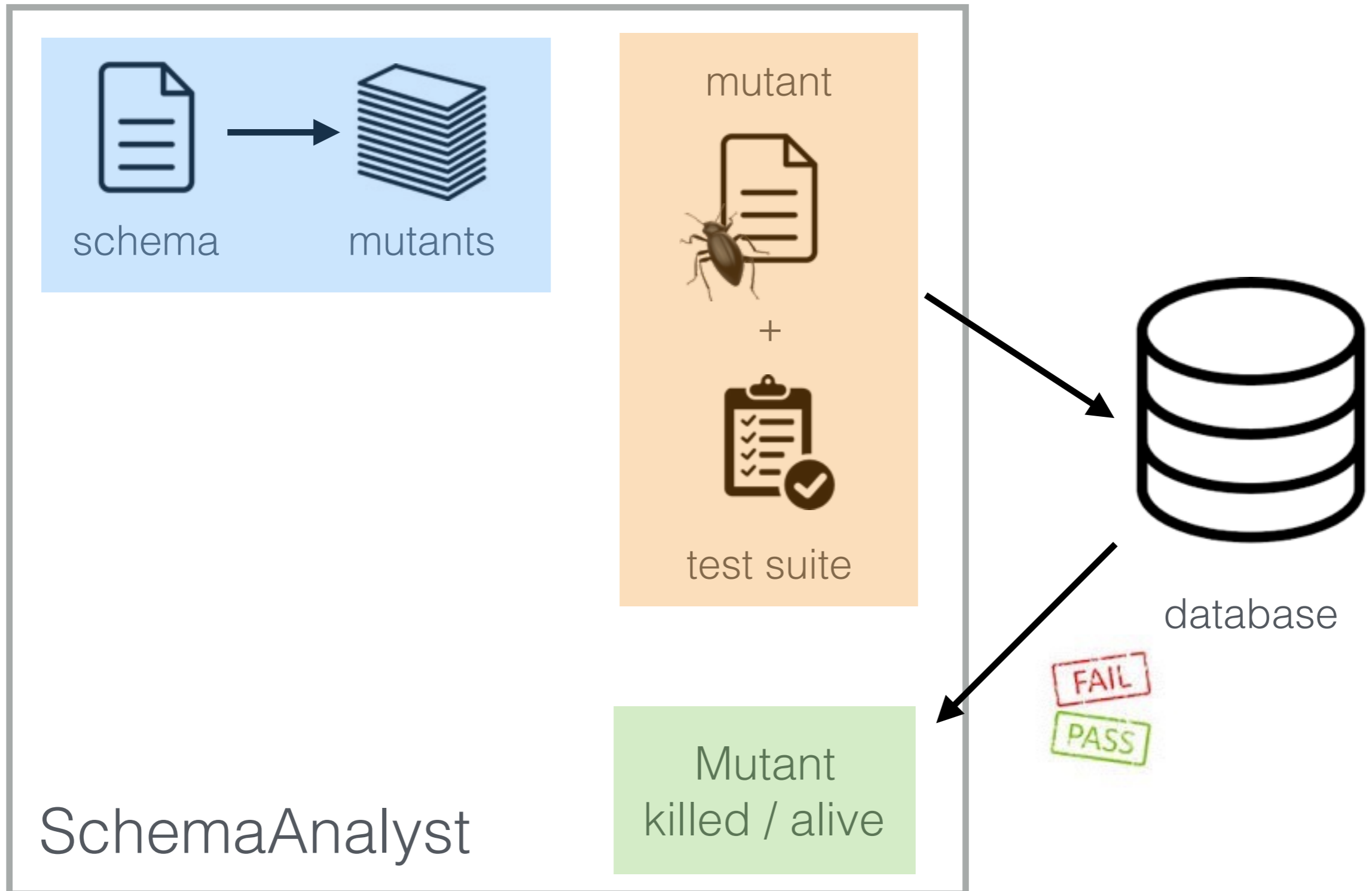
Mutation Analysis is Costly



database



Mutation Analysis is Costly



Mutation Analysis is Costly

Mutation Analysis is Costly

DO FEWER

Mutation Analysis is Costly

DO FEWER

DO SMARTER

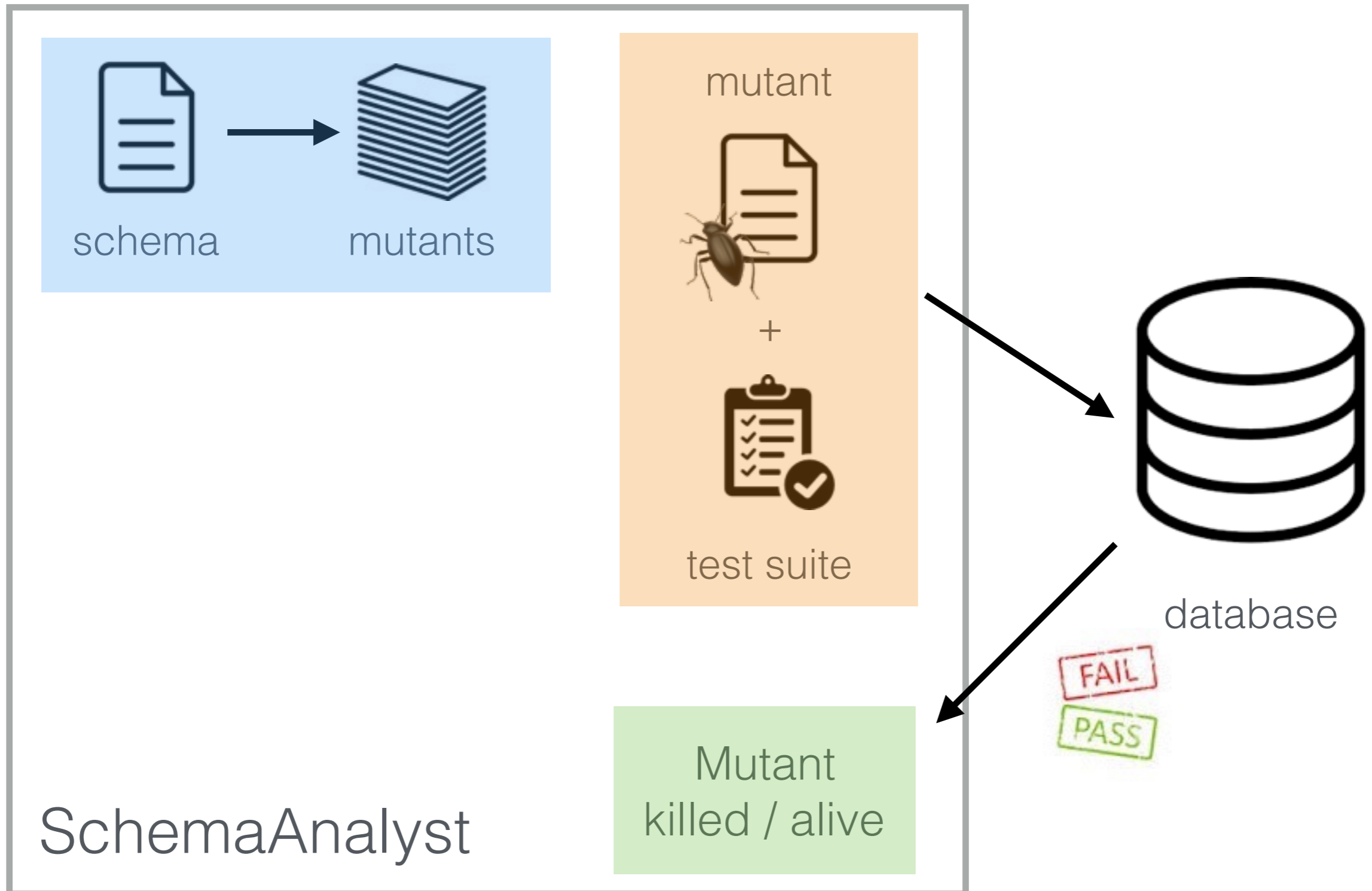
Mutation Analysis is Costly

DO FEWER

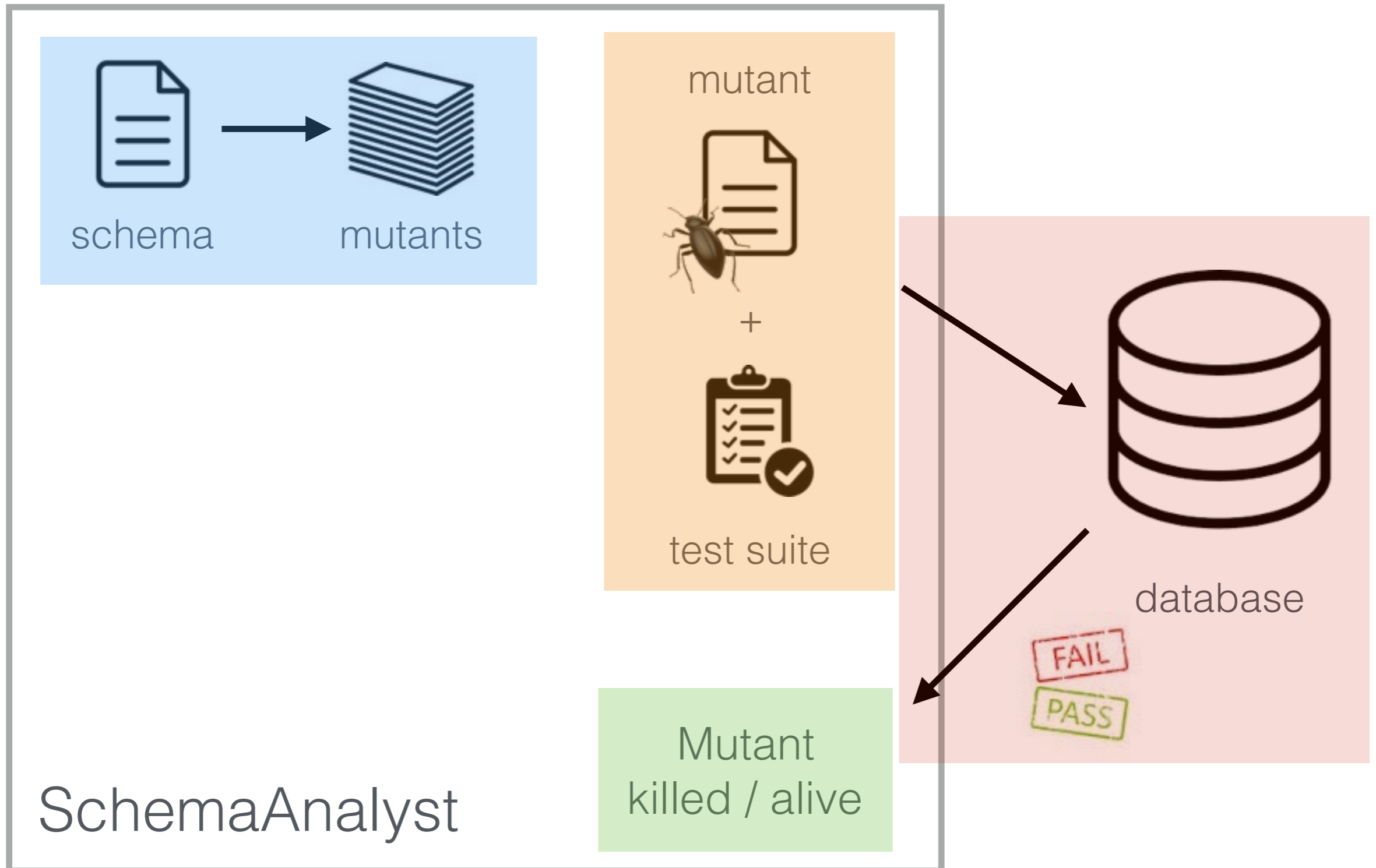
DO SMARTER

DO FASTER

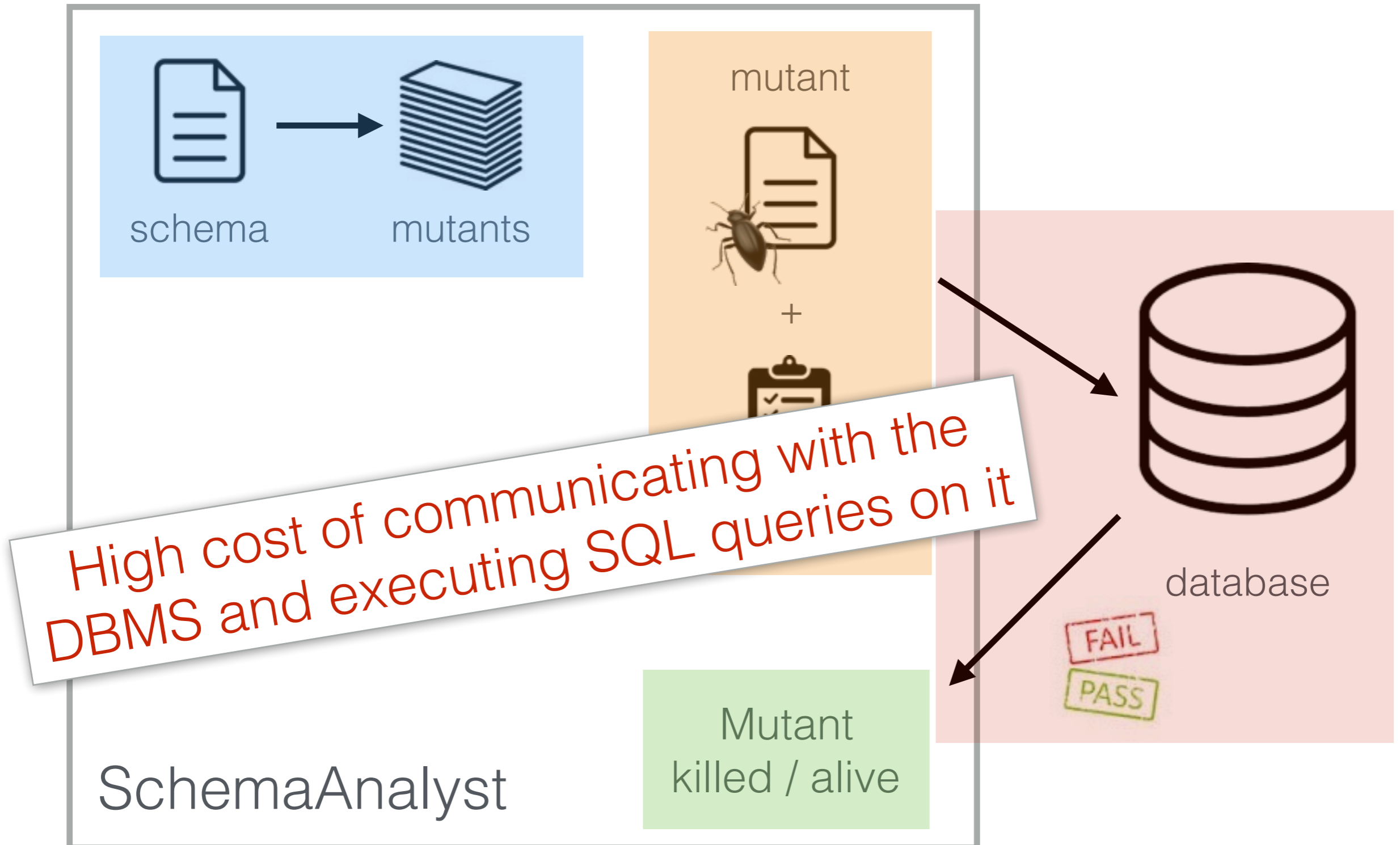
Mutation Analysis is Costly



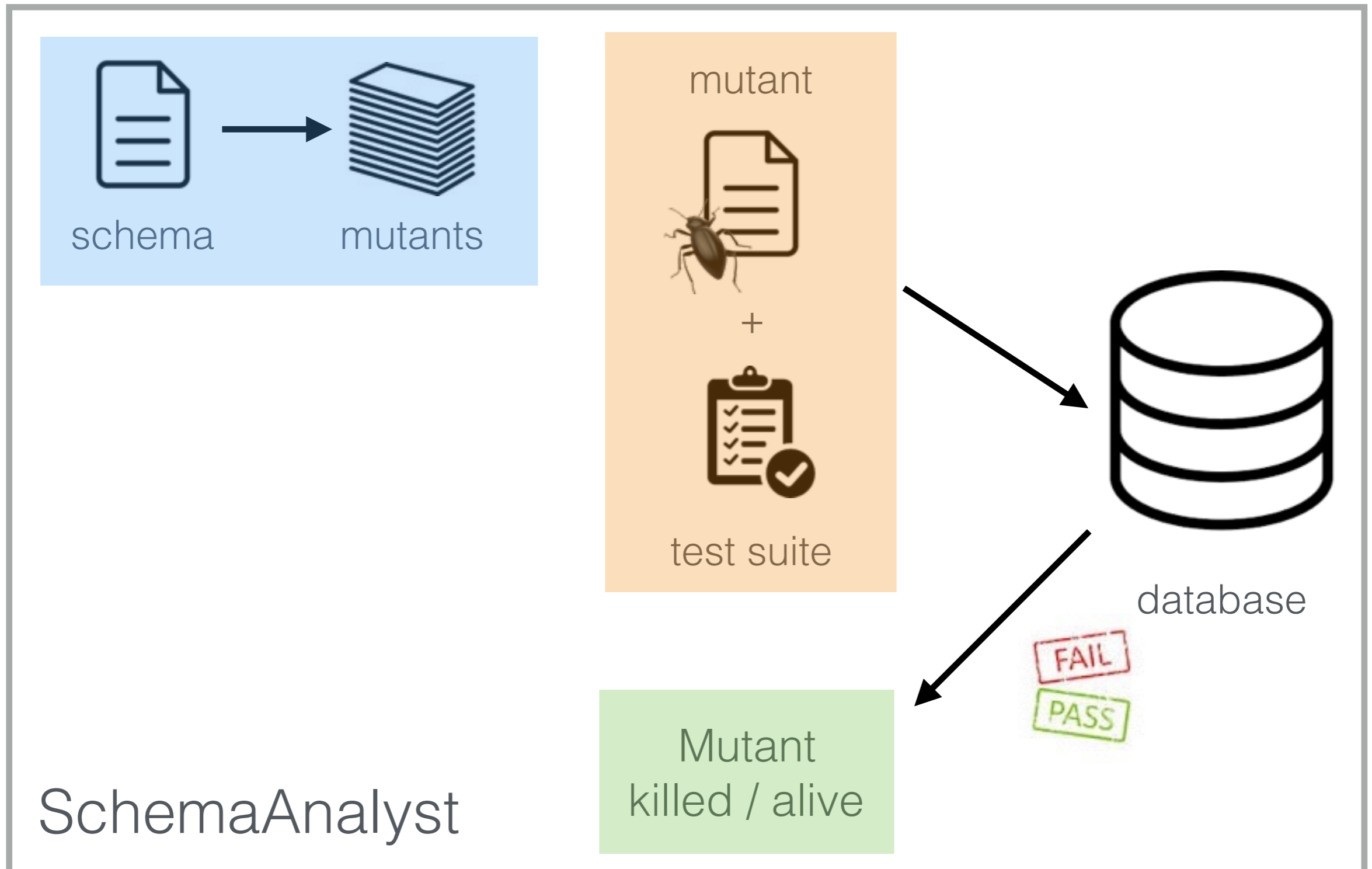
Mutation Analysis is Costly



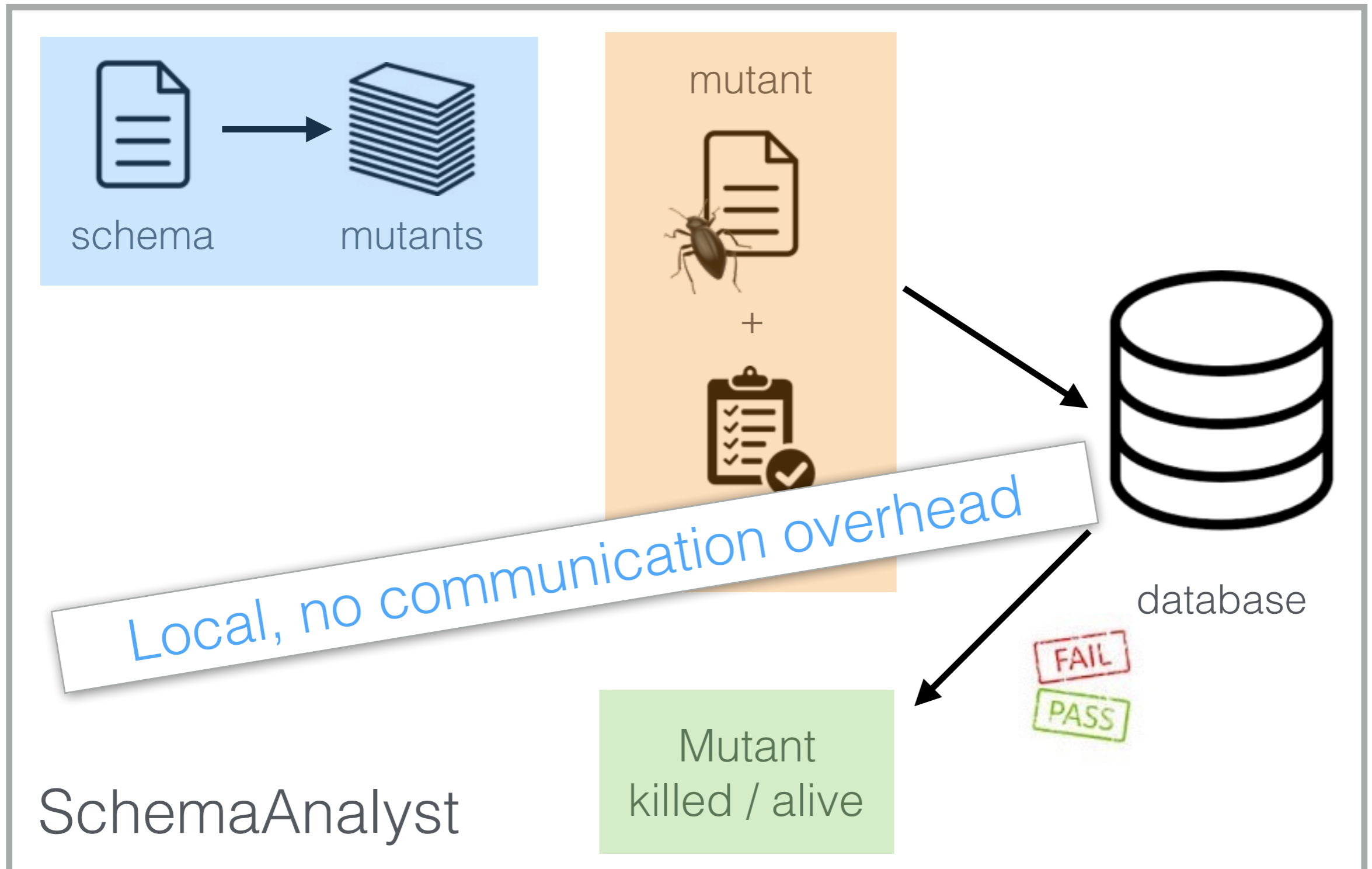
Mutation Analysis is Costly



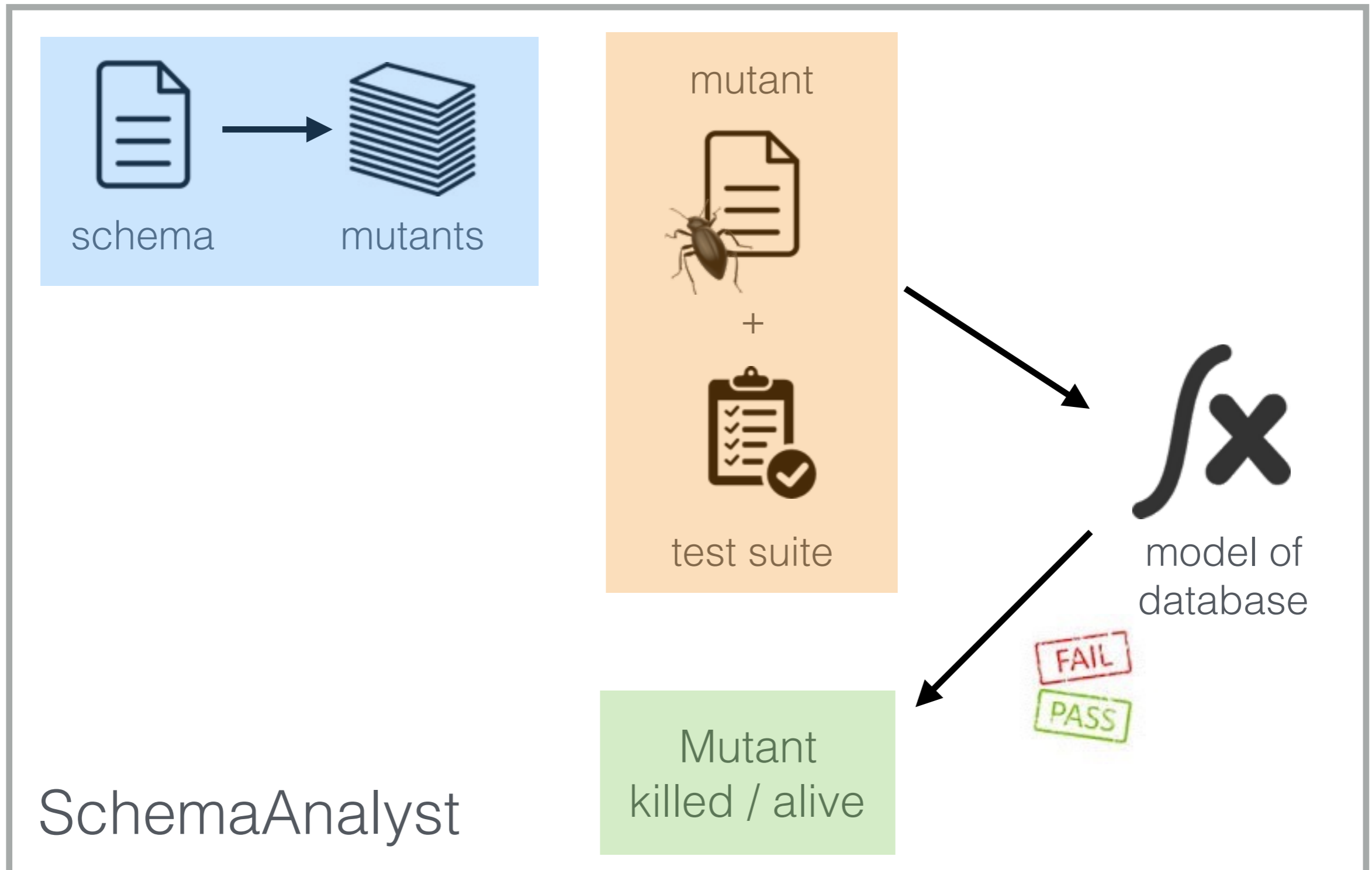
Reducing the Cost



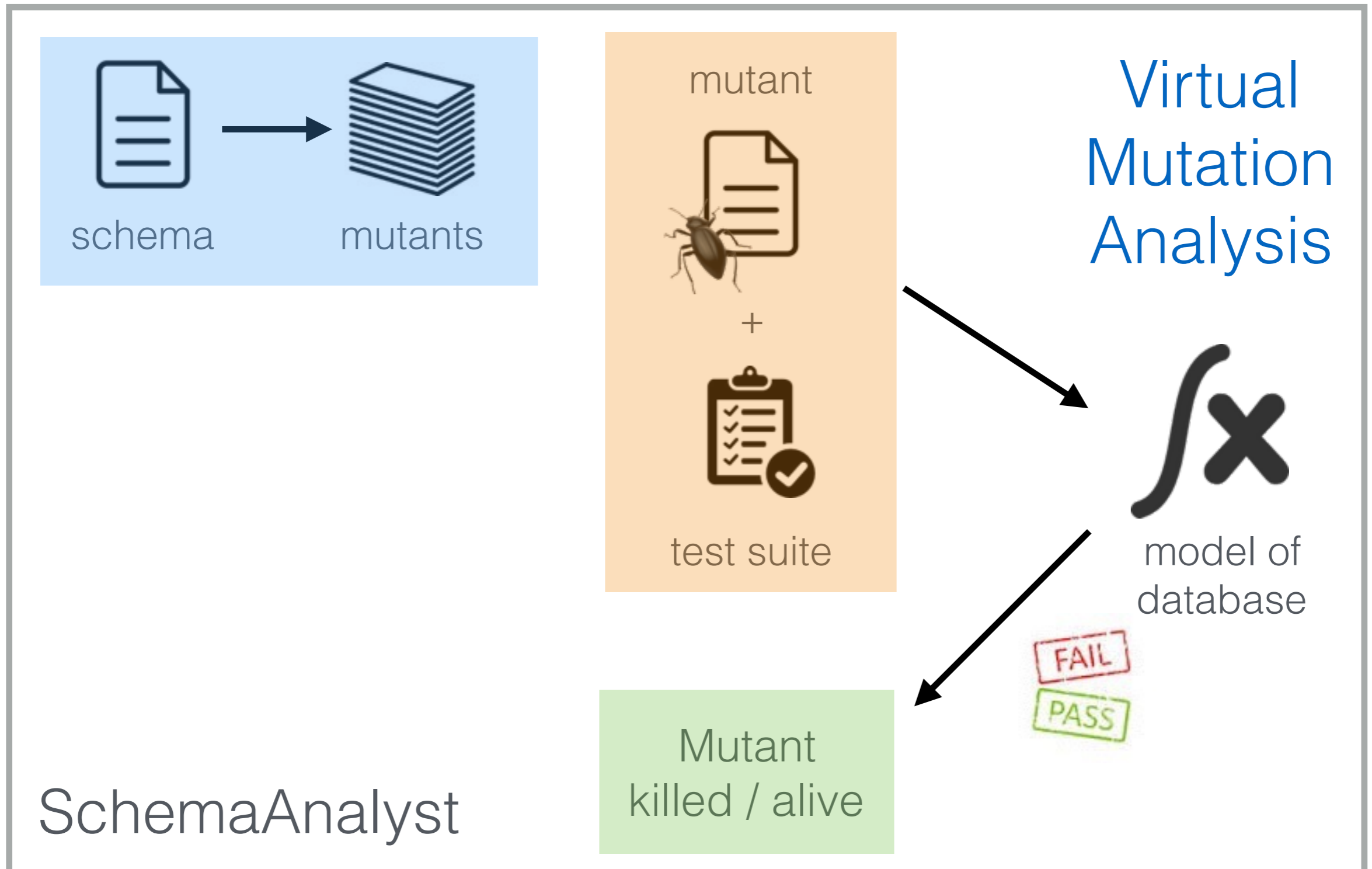
Reducing the Cost



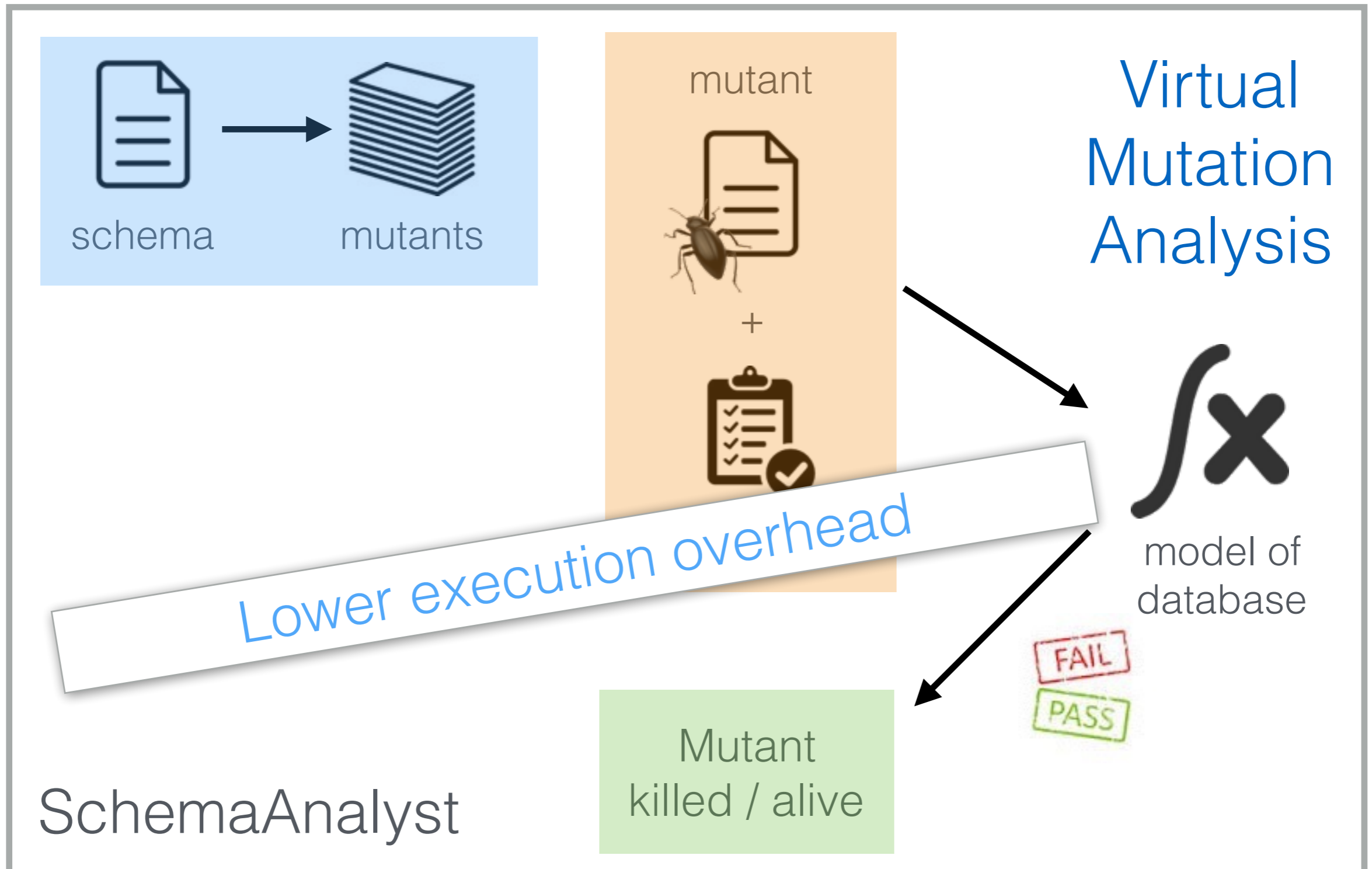
Reducing the Cost



Reducing the Cost



Reducing the Cost



The Model

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

The Model

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

Integrity constraint predicate *icp1*
 $(nr(ID) \neq NULL) \wedge (\forall er \in Station : nr(ID) \neq er(ID))$

The Model

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,   
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
  CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
  CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

Integrity constraint predicate *icp1*
 $(nr(ID) \neq NULL) \wedge (\forall er \in Station : nr(ID) \neq er(ID))$

$(nr(LAT_N) \neq NULL)$ *icp2*
 $(nr(LAT_N) \geq 0 \wedge nr(LAT_N) \leq 90)$ *icp3*

The Model

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,   
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
  CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
  CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

Integrity constraint predicate *icp1*
 $(nr(ID) \neq NULL) \wedge (\forall er \in Station : nr(ID) \neq er(ID))$

$(nr(LAT_N) \neq NULL)$ *icp2*
 $(nr(LAT_N) \geq 0 \wedge nr(LAT_N) \leq 90)$ *icp3*
icp4
icp5

The Model

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,   
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
  CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
  CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

Integrity constraint predicate *icp1*
 $(nr(ID) \neq NULL) \wedge (\forall r \in \text{Station} : nr(ID) \neq er(ID))$

$(nr(LAT_N) \neq NULL)$ *icp2*
 $(nr(LAT_N) \geq 0 \wedge nr(LAT_N) \leq 90)$ *icp3*
icp4
icp5

Form an acceptance predicate for the table:

$$ap = icp1 \wedge icp2 \wedge icp3 \wedge icp4 \wedge icp5$$

The Model

```
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,   
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
  CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
  CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```

Integrity constraint predicate *icp1*
 $(nr(ID) \neq NULL) \wedge (\forall er \in Station : nr(ID) \neq er(ID))$

$(nr(LAT_N) \neq NULL)$ *icp2*
 $(nr(LAT_N) \geq 0 \wedge nr(LAT_N) \leq 90)$ *icp3*
icp4
icp5

Form an acceptance predicate for the table:

$$ap = icp1 \wedge icp2 \wedge icp3 \wedge icp4 \wedge icp5$$

True when DBMS would accept the data

False otherwise

Virtual DBMS Models



HyperSQL

HSQldb - 100% Java Database

Empirical Study

RQ1. What is the relative **efficiency** of the virtual approach?

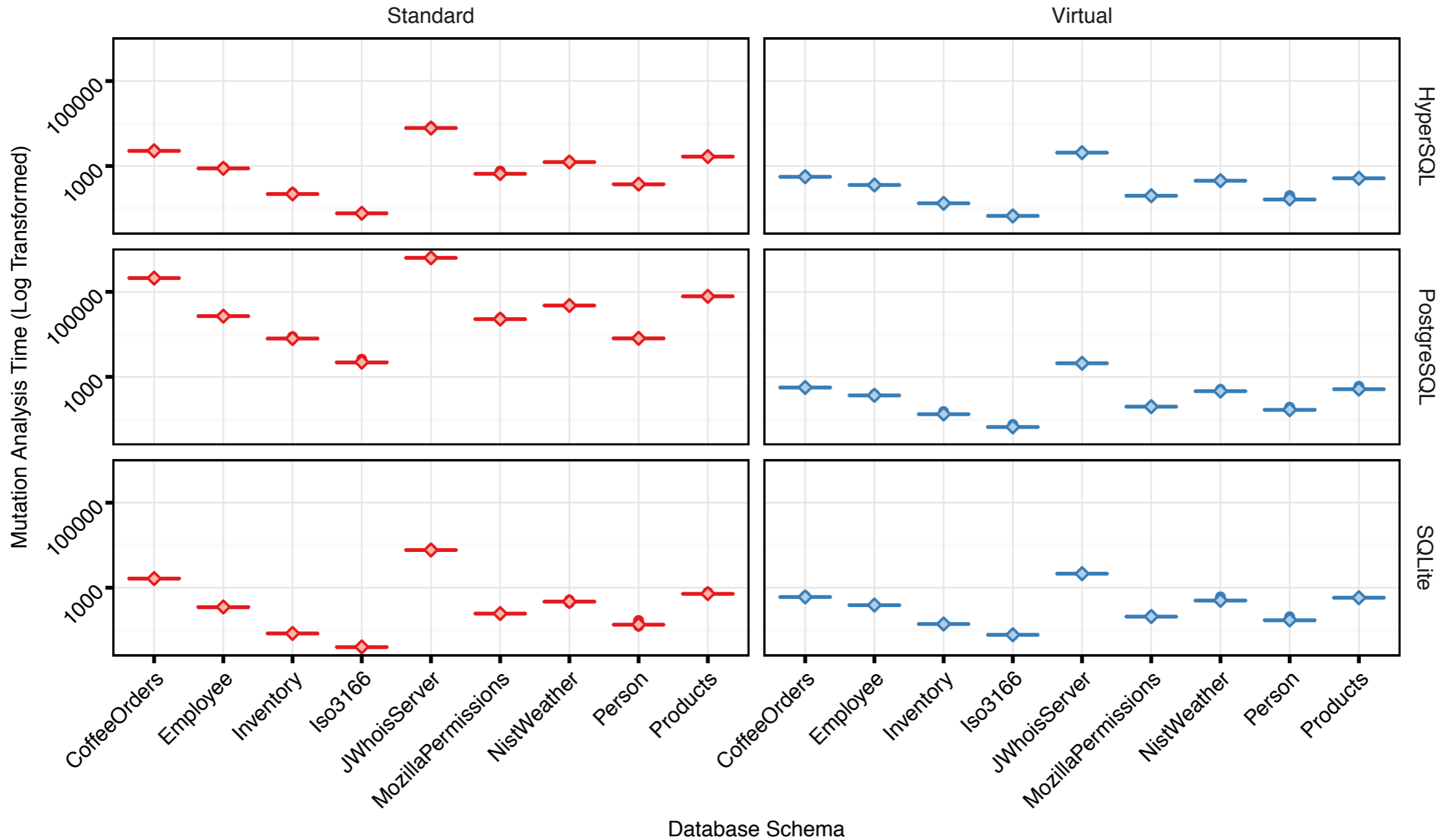
RQ2. What are the **time savings**?

RQ3. How do **mutation scores** compare when the standard approach is run for as long as the virtual one?

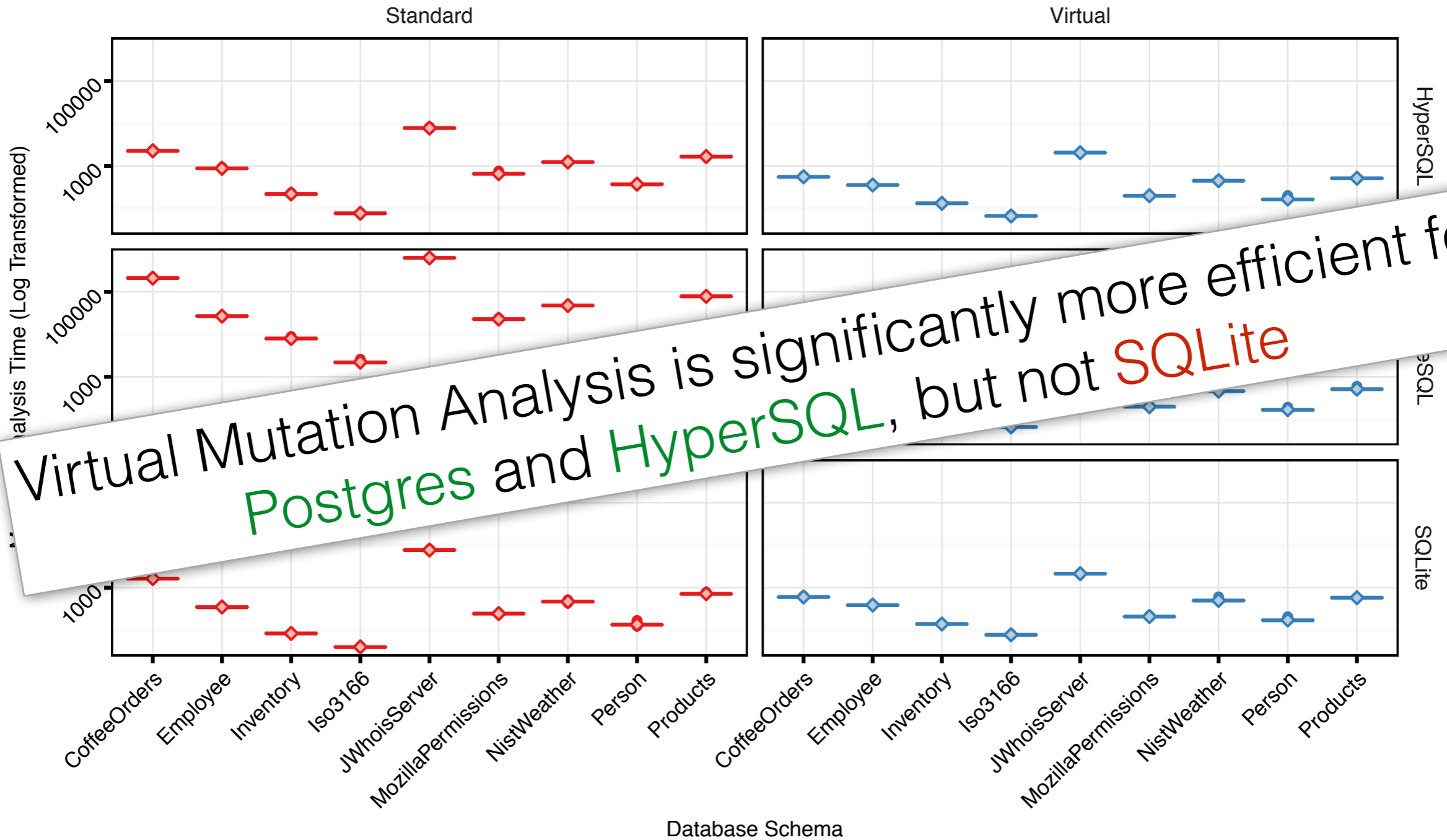
Subject Schemas

Schema	Tables	Columns	Checks	Foreign Keys	Not Nulls	Primary Keys	Uniques	Constraints
CoffeeOrders	5	20	0	4	10	5	0	19
Employee	1	7	3	0	0	1	0	4
Inventory	1	4	0	0	0	1	1	2
Iso3166	1	3	0	0	2	1	0	3
JWhoisServer	6	49	0	0	44	6	0	50
MozillaPermissions	1	8	0	0	0	1	0	1
NistWeather	2	9	5	1	5	2	0	13
Person	1	5	1	0	5	1	0	7
Products	3	9	4	2	5	3	0	14
Total	21	114	13	7	71	21	1	113

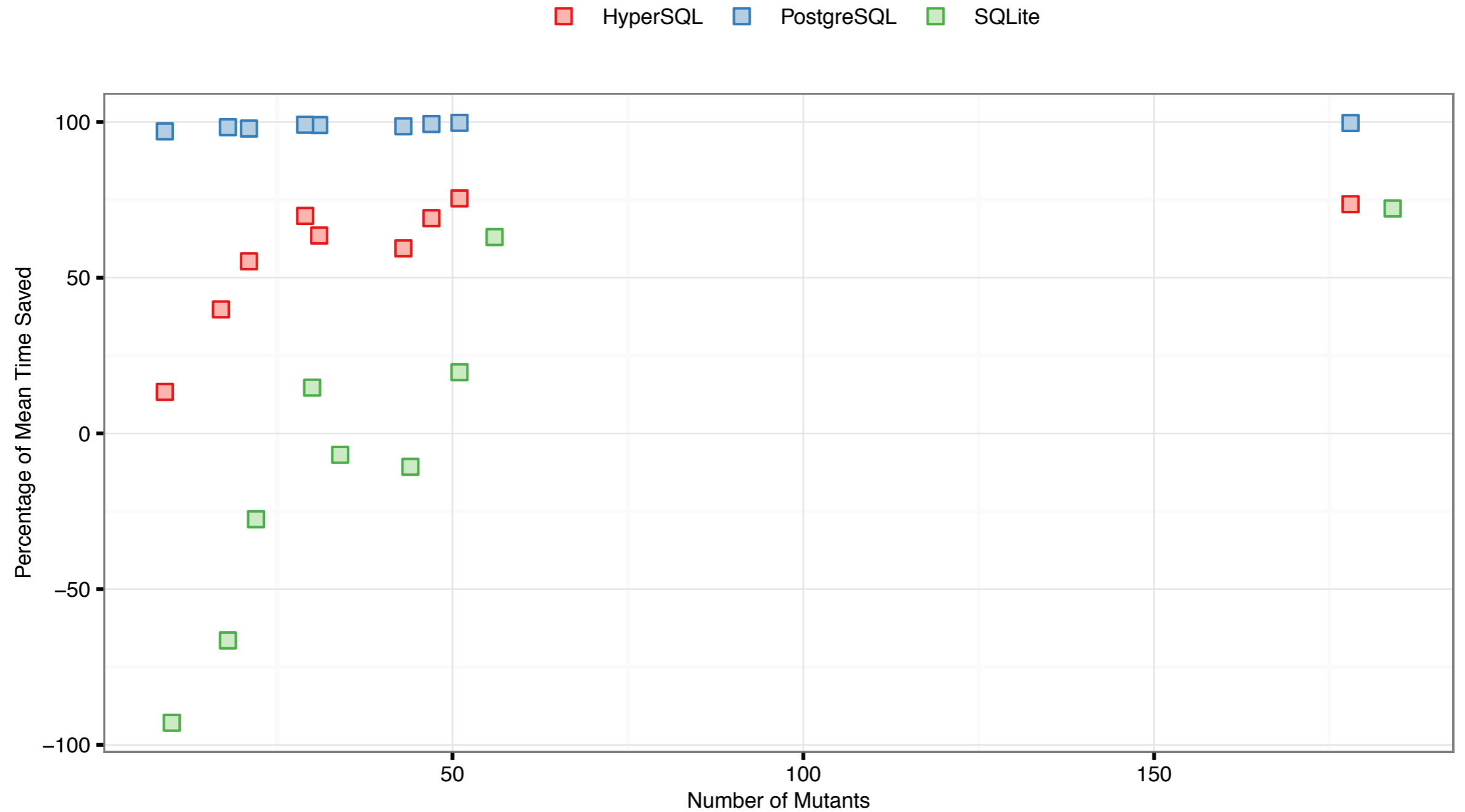
RQ1: Efficiency



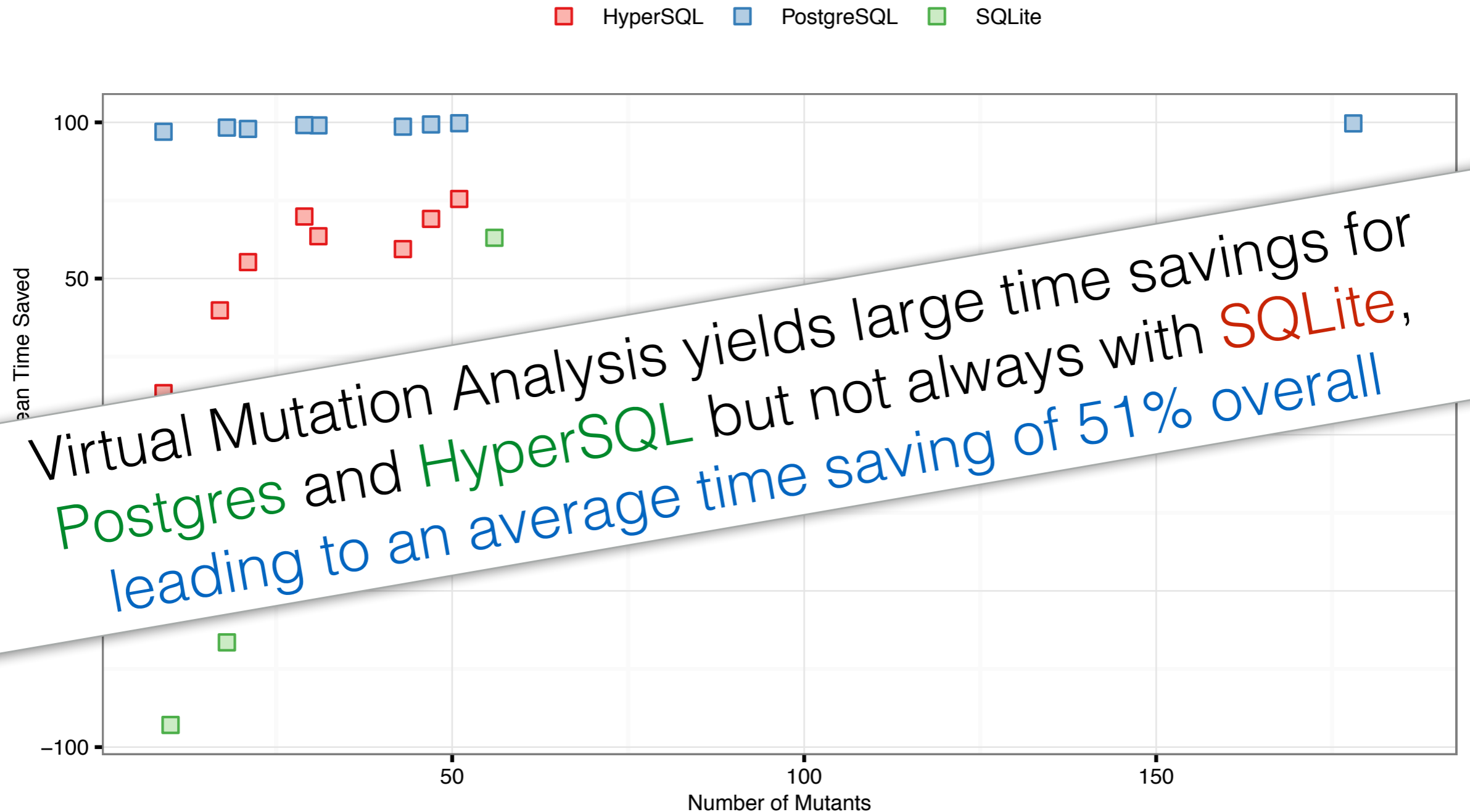
RQ1: Efficiency



RQ2: Time Savings

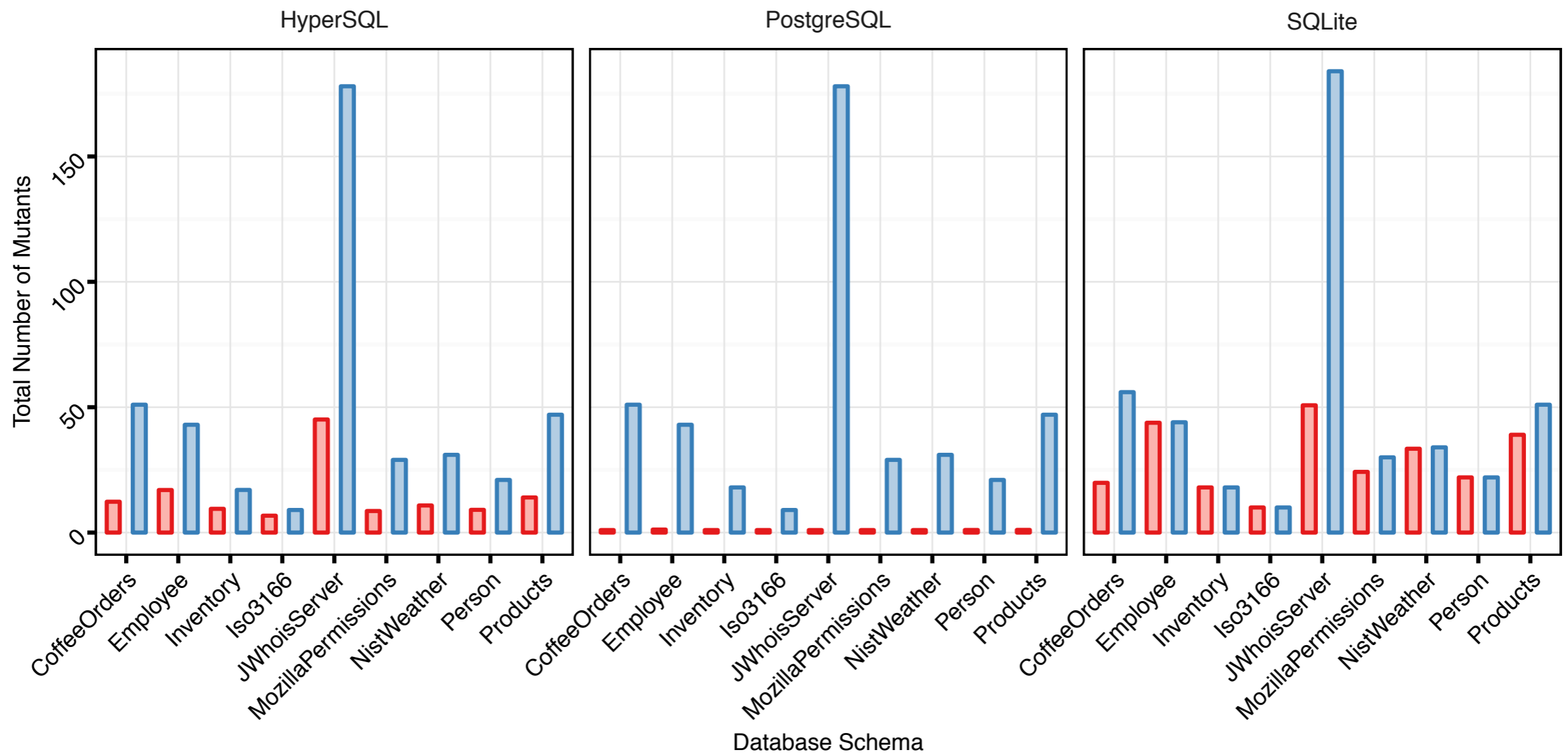


RQ2: Time Savings



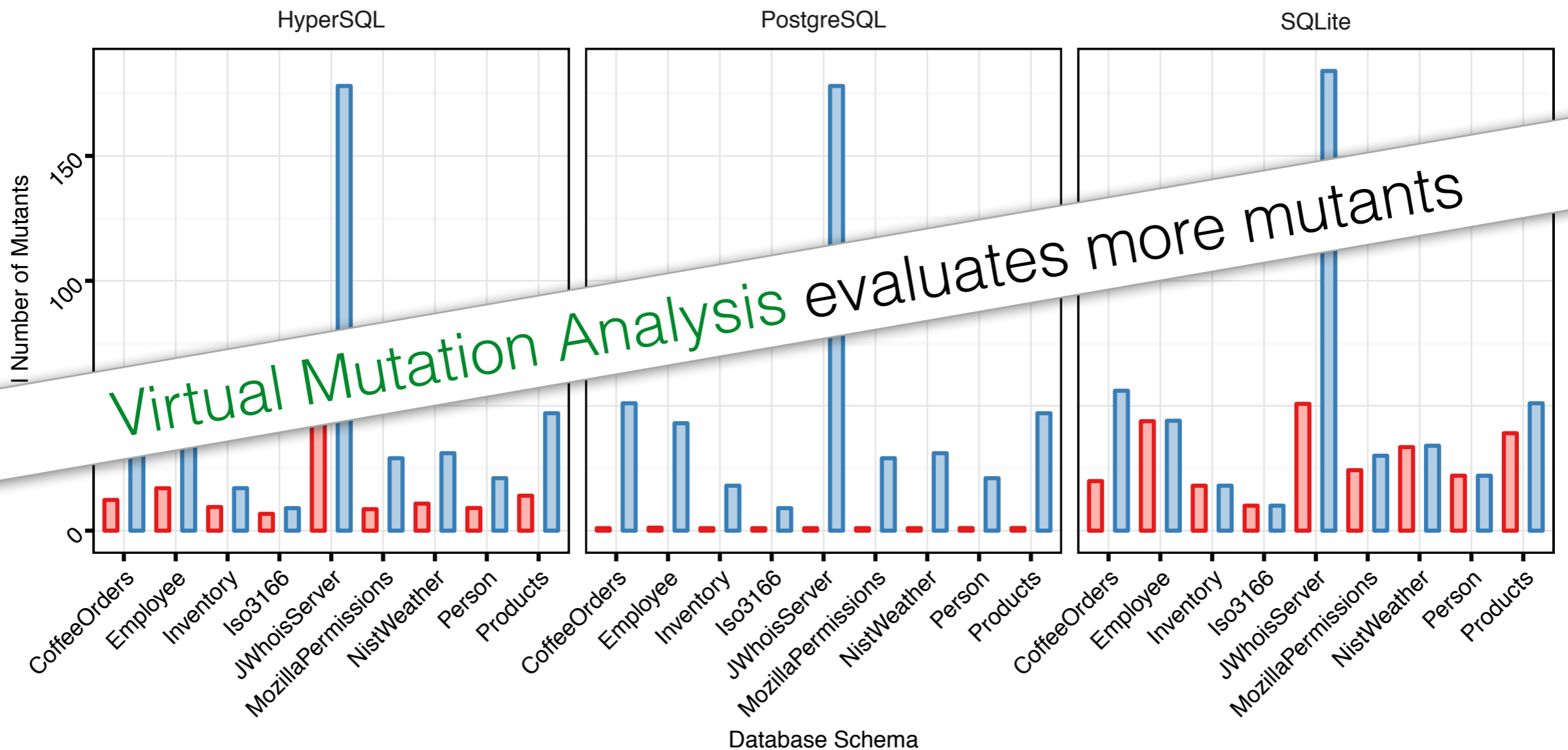
Virtual Mutation Analysis yields large time savings for PostgreSQL and HyperSQL but not always with SQLite, leading to an average time saving of 51% overall

RQ3: Comparison

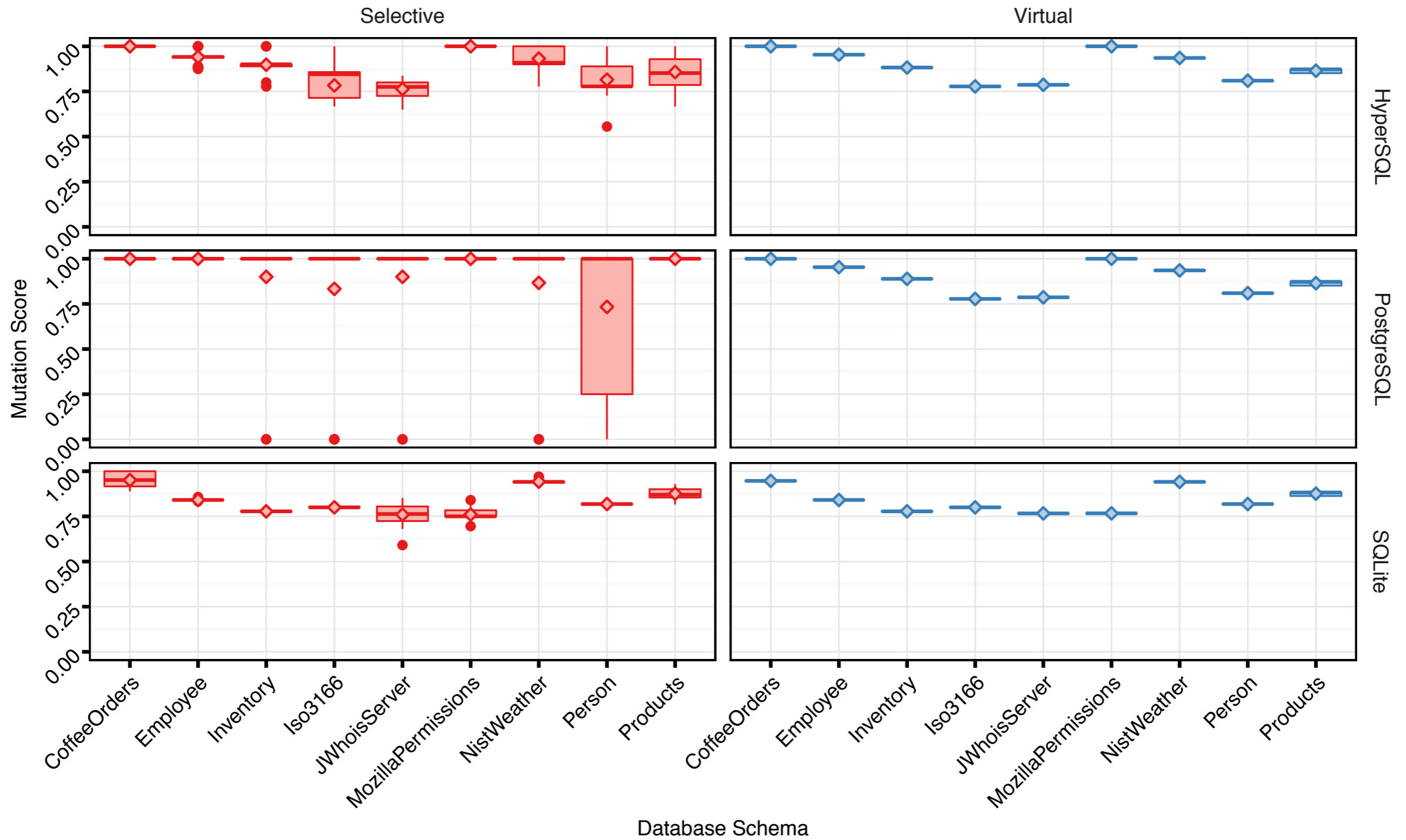


RQ3: Comparison

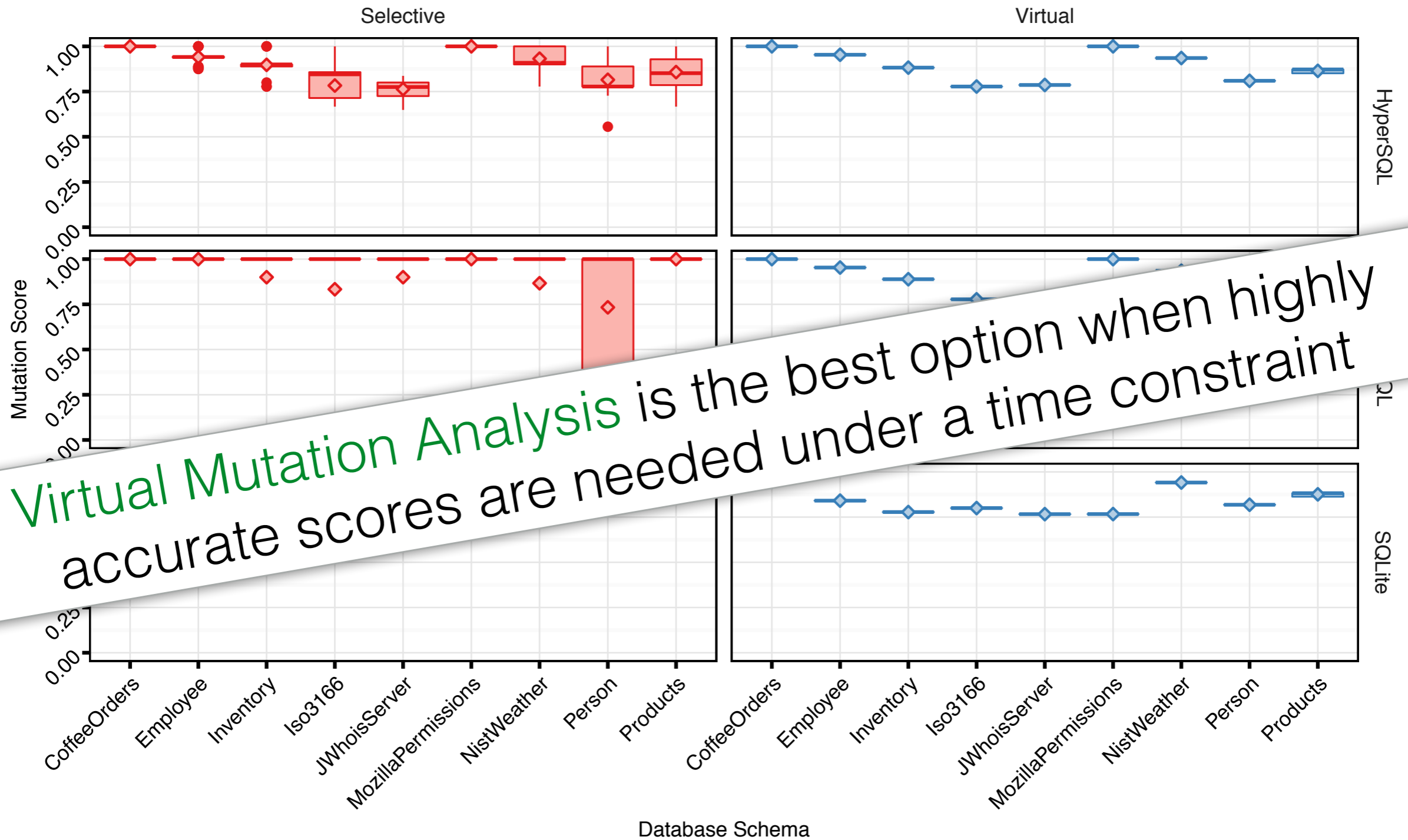
Selective Virtual



RQ3: Comparison



RQ3: Comparison



Conclusions

Virtual Mutation Analysis Technique:

Removes the need to use a real DBMS for relational database schema mutation testing

More cost-effective while still being accurate:

- More efficient for 22 of 27 configurations studied
- Yields time savings of 13 to 99%