The University Of Sheffield.

Allegheny College
Meadville, Pennsylvania

The Impact of Equivalent, Redundant and Quasi Mutants on Database Schema Mutation Analysis

Chris J. Wright

Gregory M. Kapfhammer

Phil McMinn

The Impact of Equivalent, Redundant and Quasi Mutants on Database Schema Mutation Analysis

Chris J. Wright

Gregory M. Kapfhammer

Phil McMinn

The University Of Sheffield.

Allegheny College
MEADVILLE, PENNSYLVANIA

The Impact of Equivalent, Redundant and Quasi Mutants on Database Schema Mutation Analysis

Chris J. Wright
Gregory M. Kapfhammer
Phil McMinn

The Impact of Equivalent, Redundant and Quasi Mutants on Database Schema Mutation Analysis

Chris J. Wright

Gregory M. Kapfhammer

Phil McMinn

The Impact of Equivalent, Redundant and Quasi Mutants on Database Schema Mutation Analysis

The University Of Sheffield.

Allegheny College
Meadville, Pennsylvania

Chris J. Wright

Gregory M. Kapfhammer

Phil McMinn

The Impact of Equivalent, Redundant and Quasi Mutants on Database Schema Mutation Analysis

Chris J. Wright
Gregory M. Kapfhammer
Phil McMinn

Database Schema

Database Schema

```sql
CREATE TABLE T  (
        A CHAR, B CHAR, C CHAR,
        CONSTRAINT UniqueOnColsAandB UNIQUE (A, B)
);

CREATE TABLE S (
        X CHAR, Y CHAR, Z CHAR,
        CONSTRAINT RefToColsAandB FOREIGN KEY (X, Y)
        REFERENCES T (A, B)
);
```

Database Schema

Tables

```
CREATE TABLE T  (
      A CHAR, B CHAR, C CHAR,
      CONSTRAINT UniqueOnColsAandB UNIQUE (A, B)
);

CREATE TABLE S (
      X CHAR, Y CHAR, Z CHAR,
      CONSTRAINT RefToColsAandB FOREIGN KEY (X, Y)
      REFERENCES T (A, B)
);
```

Database Schema

Columns

```
CREATE TABLE T  (
      A CHAR, B CHAR, C CHAR,
      CONSTRAINT UniqueOnColsAandB UNIQUE (A, B)
);

CREATE TABLE S (
      X CHAR, Y CHAR, Z CHAR,
      CONSTRAINT RefToColsAandB FOREIGN KEY (X, Y)
      REFERENCES T (A, B)
);
```

Database Schema

Constraints

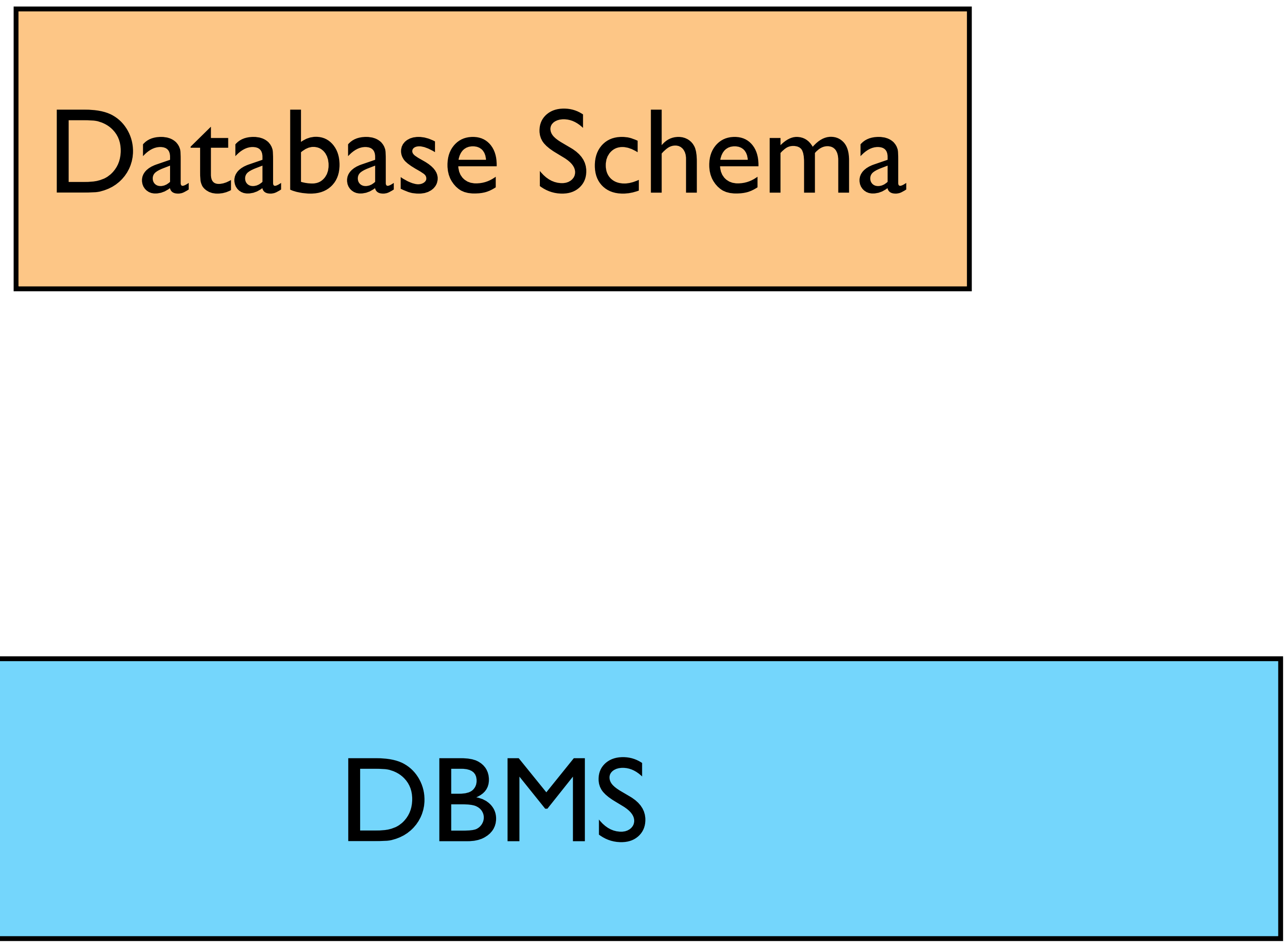
```
CREATE TABLE T  (
        A CHAR, B CHAR, C CHAR,
        CONSTRAINT UniqueOnColsAandB UNIQUE (A, B)
);

CREATE TABLE S (
        X CHAR, Y CHAR, Z CHAR,
        CONSTRAINT RefToColsAandB FOREIGN KEY (X, Y)
        REFERENCES T (A, B)
);
```

Why Test a Database Schema?

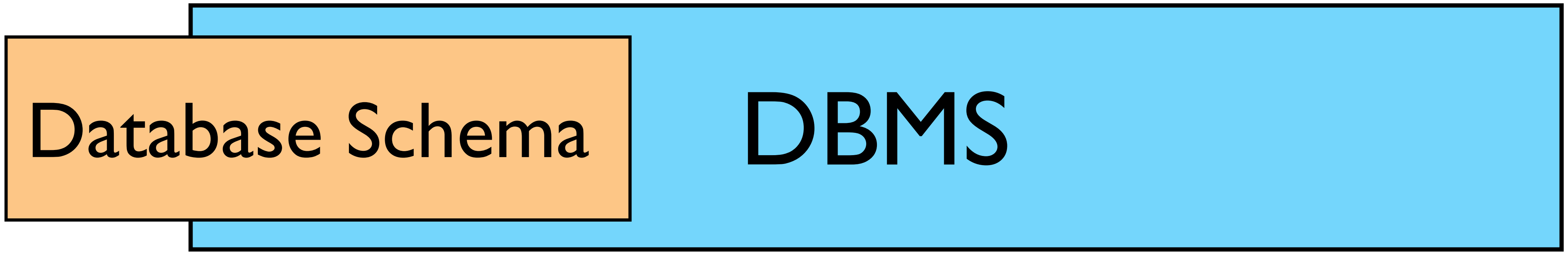
Why Test a Database Schema?

Database Schema

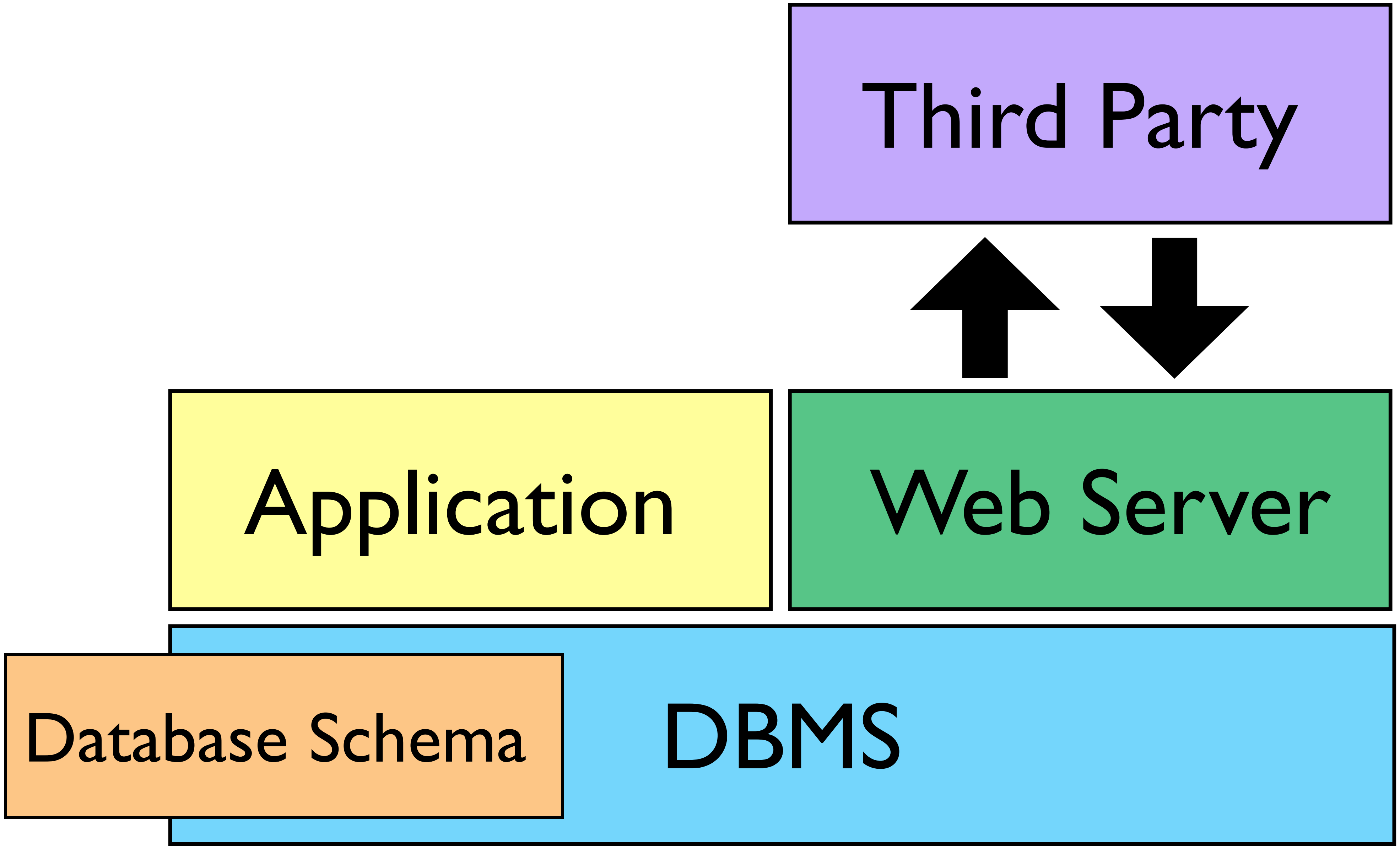
Why Test a Database Schema?



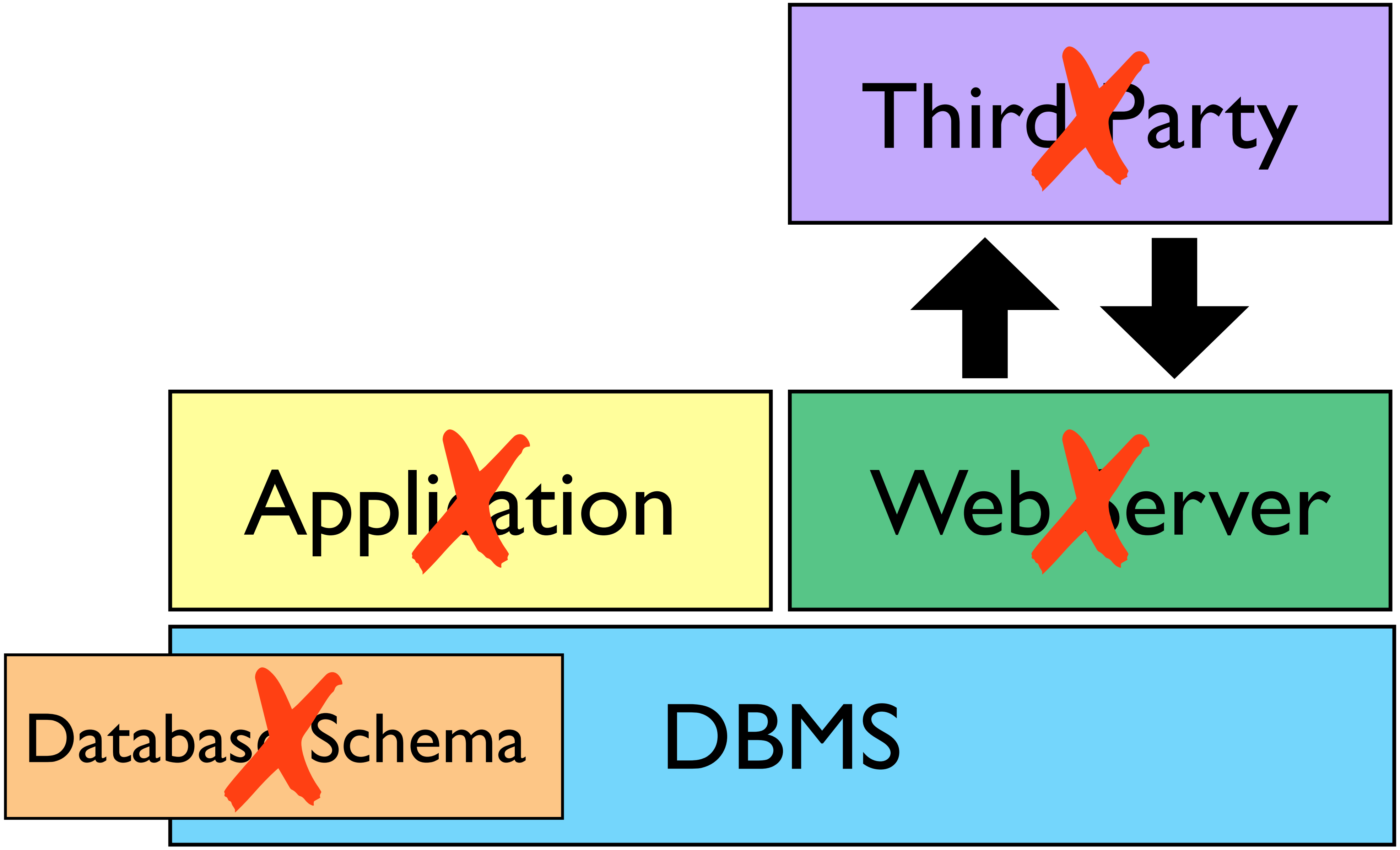
Why Test a Database Schema?



Why Test a Database Schema?



Why Test a Database Schema?



How to Test a Database Schema

How to Test a Database Schema

- Generate test data – SQL **`INSERT`** statements

How to Test a Database Schema

- Generate test data – SQL **`INSERT`** statements

```sql
INSERT INTO T(a, b)
  VALUES('a', 'b');
```

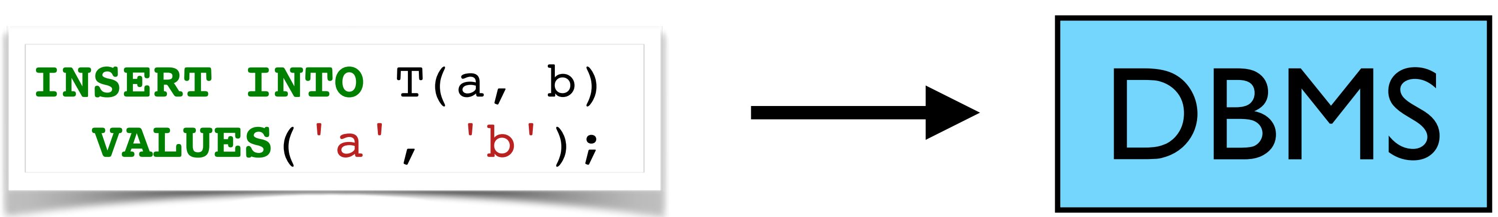
How to Test a Database Schema

- Generate test data – SQL **`INSERT`** statements

```
INSERT INTO T(a, b)
  VALUES('a', 'b');
```

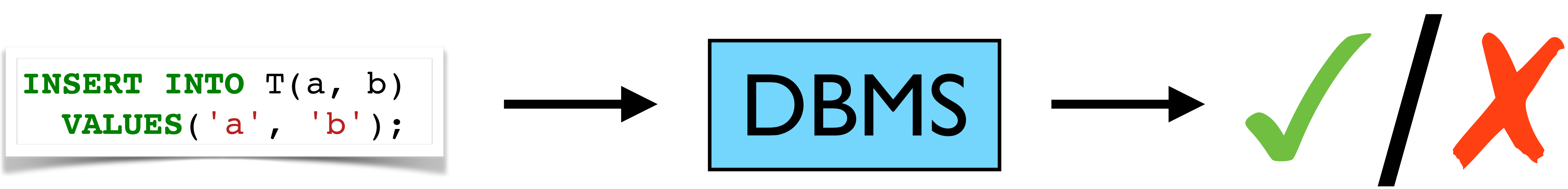
How to Test a Database Schema

- Generate test data – SQL
- Execute the data against the database



How to Test a Database Schema

- Generate test data – SQL
- Execute the data against the database
- Examine the acceptance of statements



Mutation Analysis

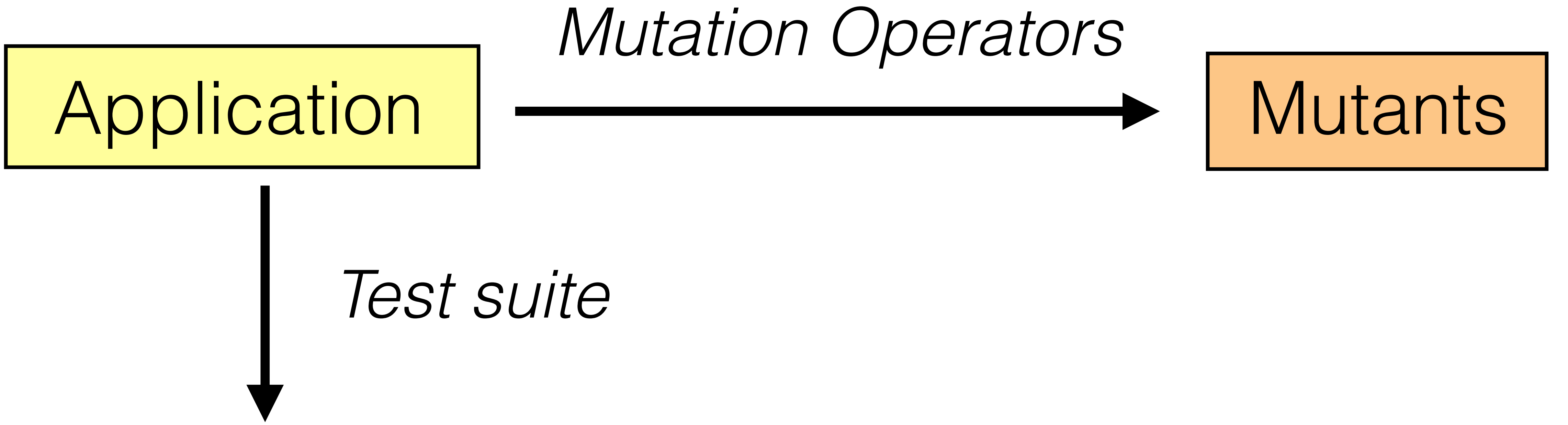
Mutation Analysis

Application

Mutation Analysis

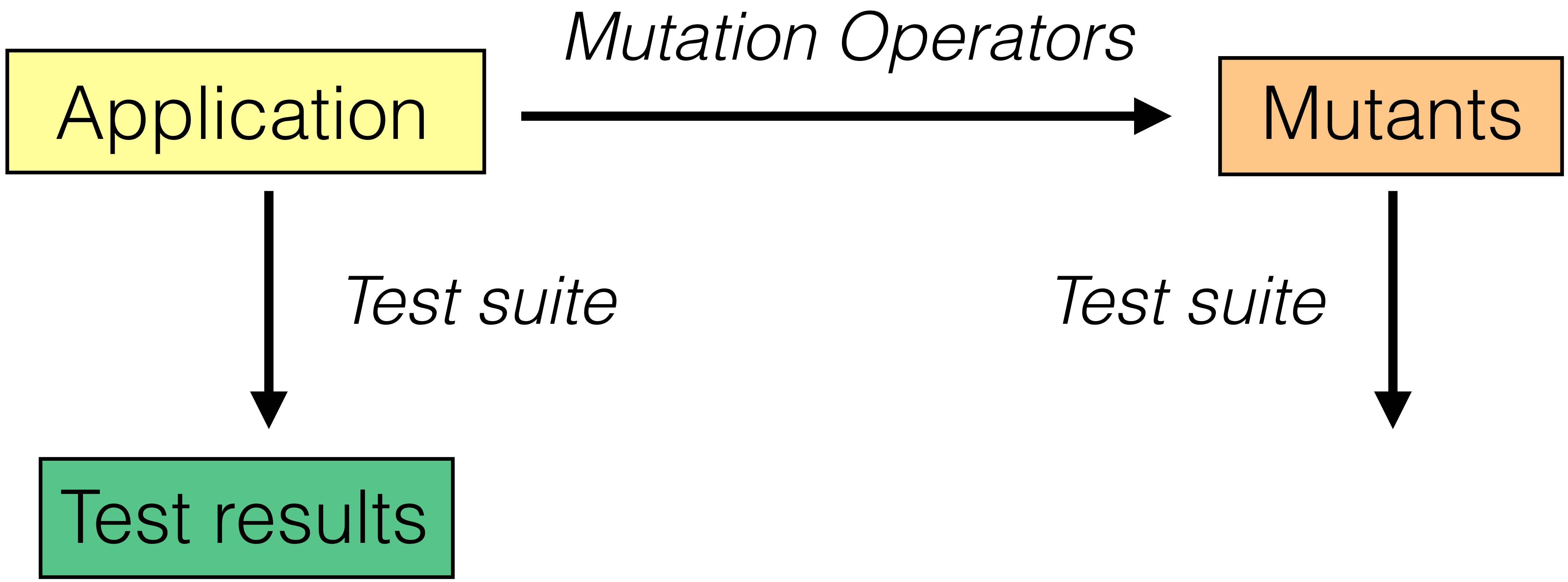
Mutation Analysis

Mutation Analysis

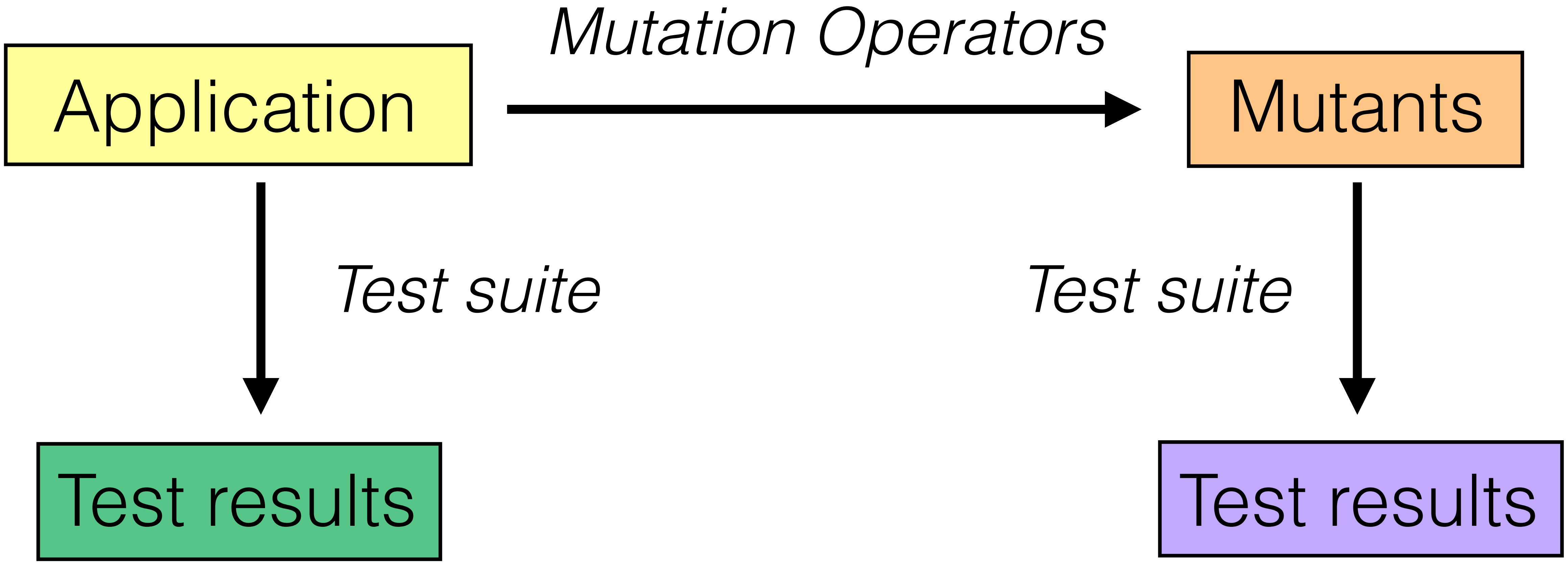


Mutation Analysis

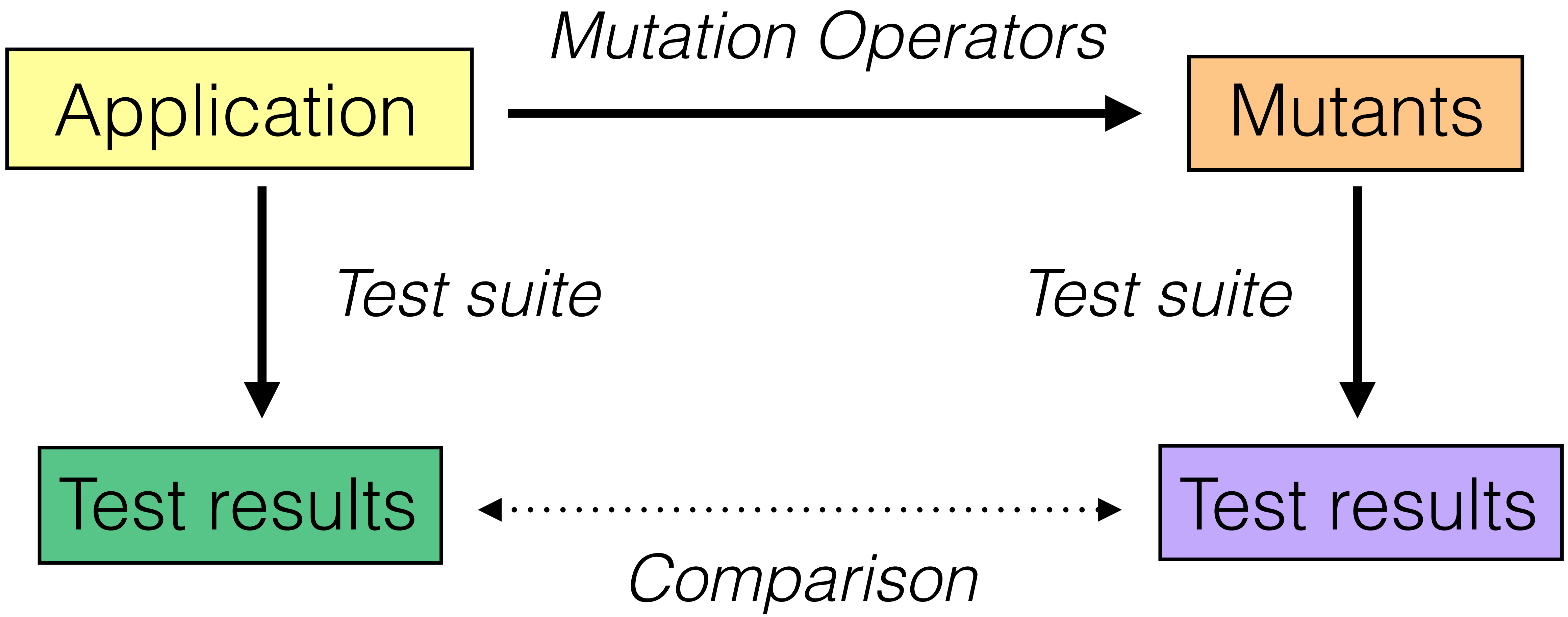
Mutation Analysis



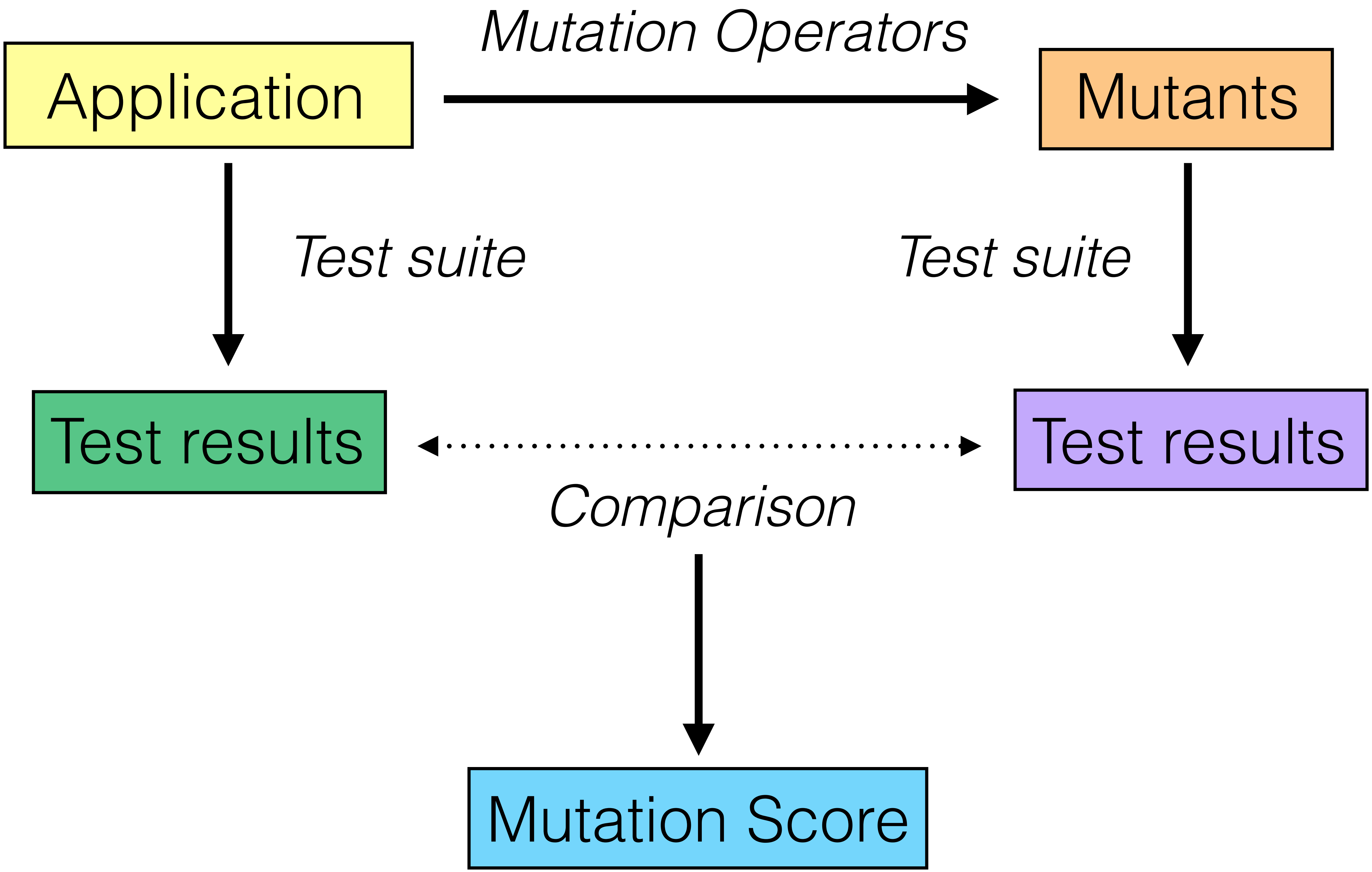
Mutation Analysis



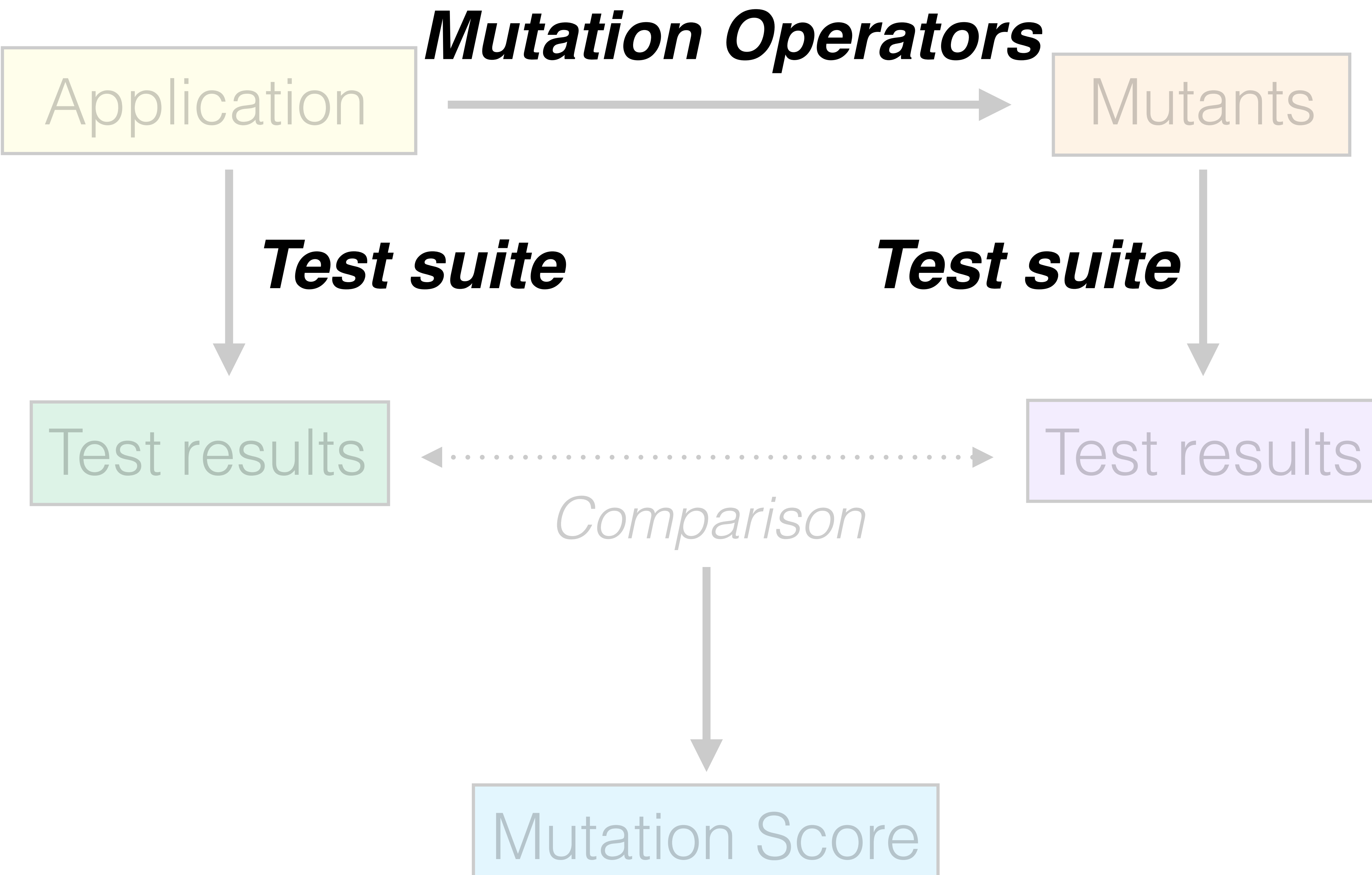
Mutation Analysis



Mutation Analysis



Mutation Analysis



Database Schema Mutation Operators

Database Schema Mutation Operators

Primary Key

Foreign Key

Unique

Not Null

Check

Database Schema Mutation Operators

Primary Key

Foreign Key

Unique

Not Null

Check

×

Database Schema Mutation Operators

Primary Key

Foreign Key

Unique

Not Null

Check

×

Column Addition

Database Schema Mutation Operators

Primary Key

Foreign Key

Unique

Not Null

Check

×

Column Addition

Column Removal

Database Schema Mutation Operators

Primary Key

Foreign Key

Unique

Not Null

Check

×

Column Addition

Column Removal

Column Exchange

Database Schema Mutation Operators

Database Schema Mutation Operators

Primary Key Column Addition

Database Schema Mutation Operators

Primary Key Column Addition

```
CREATE TABLE T  (
  A CHAR, B CHAR,
  PRIMARY KEY (A)
);
```

Database Schema Mutation Operators

Primary Key Column Addition

```
CREATE TABLE T  (
  A CHAR, B CHAR,
  PRIMARY KEY (A)
);
```

→

```
CREATE TABLE T  (
  A CHAR, B CHAR,
  PRIMARY KEY (A, B)
);
```

Database Schema Mutation Operators

Primary Key Column Exchange

```
CREATE TABLE T  (
  A CHAR, B CHAR,
  PRIMARY KEY (A)
);
```

→

```
CREATE TABLE T  (
  A CHAR, B CHAR,
  PRIMARY KEY (B)
);
```

Database Schema Mutation Operators

Primary Key Column Removal

```
CREATE TABLE T (
  A CHAR, B CHAR,
  PRIMARY KEY (A)
);
```

→

```
CREATE TABLE T (
  A CHAR, B CHAR,

);
```

Mutation Analysis – Challenges

Mutation Analysis – Challenges

- Special classes of mutants

Mutation Analysis – Challenges

- Special classes of mutants
    - Equivalent

Mutation Analysis – Challenges

- Special classes of mutants
  - Equivalent
  - Redundant

Mutation Analysis – Challenges

- Special classes of mutants
    - Equivalent
    - Redundant
    - Quasi-mutants

Mutation Analysis – Challenges

- Special classes of mutants
  - Equivalent
  - Redundant
  - Quasi-mutants

Equivalent Mutants

Equivalent Mutants

- Functionally identical to non-mutant

Equivalent Mutants

- Functionally identical to non-mutant
- …but syntactically different

Equivalent Mutants

- Functionally identical to non-mutant
- …but syntactically different
- Cannot be ‘killed’

Equivalent Mutants

- Functionally identical to non-mutant
- …but syntactically different
- Cannot be ‘killed’
- Artificially decrease mutation score

Equivalent Mutants

- Functionally identical to non-mutant
- …but syntactically different
- Cannot be ‘killed’
- Artificially decrease mutation score

Equivalent Mutants

Equivalent Mutants

Original:

```
CREATE TABLE T  (
        A CHAR,
        PRIMARY KEY (A)
);
```

Equivalent Mutants

Original:

```
CREATE TABLE T  (
        A CHAR,
        PRIMARY KEY (A)
);
```

Mutant:

```
CREATE TABLE T  (
        A CHAR NOT NULL,
        PRIMARY KEY (A)
);
```

Redundant Mutants

Redundant Mutants

- Functionally identical to **another mutant**

Redundant Mutants

- Functionally identical to
- …but syntactically different

Redundant Mutants

- Functionally identical to
- …but syntactically different
- May be ‘killed’

Redundant Mutants

- Functionally identical to
- …but syntactically different
- May be ‘killed’
- Artificially alters mutation score

Redundant Mutants

- Functionally identical to
- …but syntactically different
- May be ‘killed’
- Artificially alters mutation score
- Reduces efficiency

Redundant Mutants

- Functionally identical to
- …but syntactically different
- May be ‘killed’
- Artificially alters mutation score
- Reduces efficiency

Types of Equivalence

Types of Equivalence

- Structural

Types of Equivalence

- Structural
    - Functionally irrelevant syntactic differences

Types of Equivalence

- Structural
    - Functionally irrelevant syntactic differences

Types of Equivalence

- Structural
    - Functionally irrelevant syntactic differences
- Behavioural

Types of Equivalence

- Structural
    - Functionally irrelevant syntactic differences
- Behavioural
    - Overlap within SQL features

Types of Equivalence

- Structural
  - Functionally irrelevant syntactic differences
- Behavioural
  - Overlap within SQL features

Behavioural Equivalence Patterns

Behavioural Equivalence Patterns

- **`NOT NULL`** in **`CHECK`** constraints
- **`NOT NULL ≅ CHECK(… IS NOT NULL)`**

Behavioural Equivalence Patterns

- **`NOT NULL`** in **`CHECK`** constraints
- **`NOT NULL ≅ CHECK(… IS NOT NULL)`**

```sql
CREATE TABLE T (
  A CHAR NOT NULL,
);
```

```sql
CREATE TABLE T (
  A CHAR,
  CHECK(A IS NOT NULL)
);
```

Behavioural Equivalence Patterns

Behavioural Equivalence Patterns

- `NOT NULL` on `PRIMARY KEY` columns

Behavioural Equivalence Patterns

- **`NOT NULL`** on **`PRIMARY KEY`** columns
- Implicit **`NOT NULL`** on **`PRIMARY KEY`**

Behavioural Equivalence Patterns

- **`NOT NULL`** on **`PRIMARY KEY`** columns
- Implicit **`NOT NULL`** on **`PRIMARY KEY`**
- (Only PostgreSQL and HyperSQL)

Behavioural Equivalence Patterns

- **`NOT NULL`** on **`PRIMARY KEY`** columns

Behavioural Equivalence Patterns

- **`NOT NULL`** on **`PRIMARY KEY`** columns

```
CREATE TABLE T (
  A CHAR,
  PRIMARY KEY (A)
);
```

```
CREATE TABLE T (
  A CHAR NOT NULL,
  PRIMARY KEY (A)
);
```

Behavioural Equivalence Patterns

Behavioural Equivalence Patterns

- **`UNIQUE`** and **`PRIMARY KEY`** with shared columns

Behavioural Equivalence Patterns

- **`UNIQUE`** and **`PRIMARY KEY`** with shared columns

```
CREATE TABLE T (
  A CHAR,
  PRIMARY KEY (A)
);
```

```
CREATE TABLE T (
  A CHAR,
  PRIMARY KEY (A),
  UNIQUE (A)
);
```

Quasi-mutants

Quasi-mutants

- Operators produce DBMS-agnostic mutants

Quasi-mutants

- Operators produce DBMS-agnostic mutants
- Some DBMSs have implicit constraints

Quasi-mutants

- Operators produce DBMS-agnostic mutants
- Some DBMSs have implicit constraints
- Valid for some DBMSs, invalid for others

PostgreSQL ✗	HyperSQL ✗	SQLite ✓

Quasi-mutants

- Operators produce DBMS-agnostic mutants
- Some DBMSs have implicit constraints
- Valid for some DBMSs, invalid for others

PostgreSQL ✗ | HyperSQL ✗ | SQLite ✓

Quasi-mutants

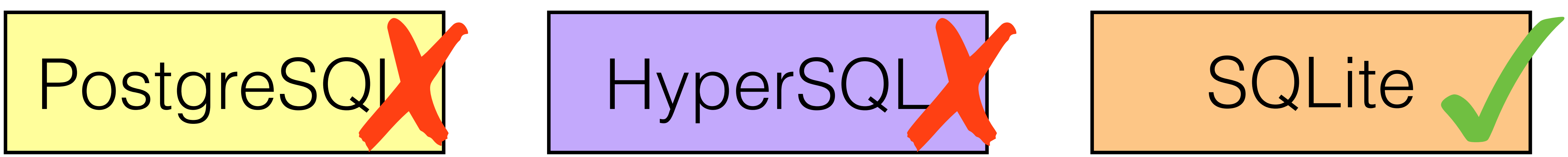
PostgreSQL ✗

HyperSQL ✗

SQLite ✓

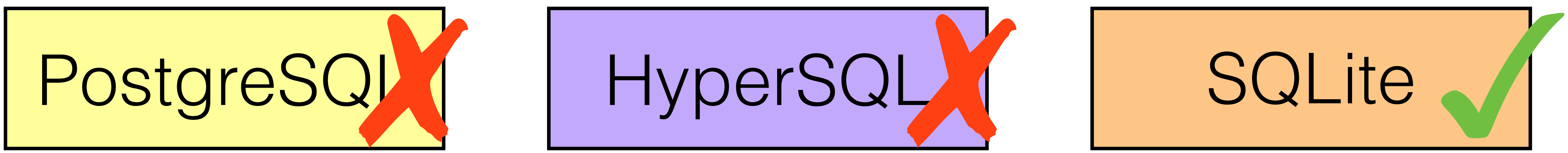
Quasi-mutants

- Cannot adversely affect mutation score



Quasi-mutants

- Cannot adversely affect mutation score
- …but may preclude some optimisations



Quasi-mutants

- Cannot adversely affect mutation score
- …but may preclude some optimisations
- Remove when DBMS will ‘reject’ them

SQLite ✓

PostgreSQL ✗ HyperSQL ✗

Types of Quasi-mutants

Types of Quasi-mutants

- Representative example

Types of Quasi-mutants

- Representative example
  - DBMS: PostgreSQL, HyperSQL

Types of Quasi-mutants

- Representative example
  - DBMS: PostgreSQL, HyperSQL
  - $\forall$ FK(reference columns) $\exists$
    (PK(reference columns) $\vee$
    Unique(reference columns))

Types of Quasi-mutants

- Representative example
  - DBMS: PostgreSQL, HyperSQL
  - $\forall$ FK(reference columns) $\exists$
    (PK(reference columns) $\vee$
    Unique(reference columns))

Types of Quasi-mutants

- Representative example
  - DBMS: PostgreSQL, HyperSQL
  - $\forall$ FK(reference columns) $\exists$
    (PK(reference columns) $\vee$
    Unique(reference columns))

Types of Quasi-mutants

- Representative example
  - DBMS: PostgreSQL, HyperSQL
  - $\forall$ FK(reference columns) $\exists$
    (PK(reference columns) $\vee$
    Unique(reference columns))

Types of Quasi-mutants

- Representative example
  - DBMS: PostgreSQL, HyperSQL
  - $\forall$ FK(reference columns) $\exists$
    (PK(reference columns) $\lor$
    Unique(reference columns))

Types of Quasi-mutants

- Representative example
  - DBMS: PostgreSQL, HyperSQL
  - ∀

    (PK(reference columns)

    Unique(reference

Detecting Quasi-mutants

Detecting Quasi-mutants

- Submit to DBMS

Detecting Quasi-mutants

- Submit to DBMS
    - 100% accurate

Detecting Quasi-mutants

- Submit to DBMS
  - 100% accurate
  - Convert representation to SQL, submit to database, inspect response

Detecting Quasi-mutants

- Submit to DBMS
    - 100% accurate
    - Convert representation to SQL, submit to database, inspect response
- Analyse statically

Detecting Quasi-mutants

- Submit to DBMS
  - 100% accurate
  - Convert representation to SQL, submit to database, inspect response
- Analyse statically
  - Operates directly on representation

Detecting Quasi-mutants

- Submit to DBMS
    - 100% accurate
    - Convert representation to SQL, submit to database, inspect response
- Analyse statically
    - Operates directly on representation
    - DBMS-specific implementation

Empirical Study

Empirical Study

1. Quasi-mutant detection – DBMS v Static Analysis

Empirical Study

1. Quasi-mutant detection – DBMS v Static Analysis
2. Equivalent, Redundant and Quasi-mutant removal – Efficiency?

Empirical Study

1. Quasi-mutant detection – DBMS v Static Analysis
2. Equivalent, Redundant and Quasi-mutant removal – Efficiency?
3. Equivalent, Redundant and Quasi-mutant removal – Effectiveness?

Empirical Study

Empirical Study

- 16 schemas

Empirical Study

- 16 schemas
- 2 DBMSs – PostgreSQL, HyperSQL

Empirical Study

- 16 schemas
- 2 DBMSs – PostgreSQL, HyperSQL
- 15 repeat trials

Empirical Study

- 16 schemas
- 2 DBMSs – PostgreSQL, HyperSQL
- 15 repeat trials

Empirical Study – Quasi-mutants

Empirical Study – Quasi-mutants

- 5 conditions:

Empirical Study – Quasi-mutants

- 5 conditions:
  - Postgres (with/without transactions)

Empirical Study – Quasi-mutants

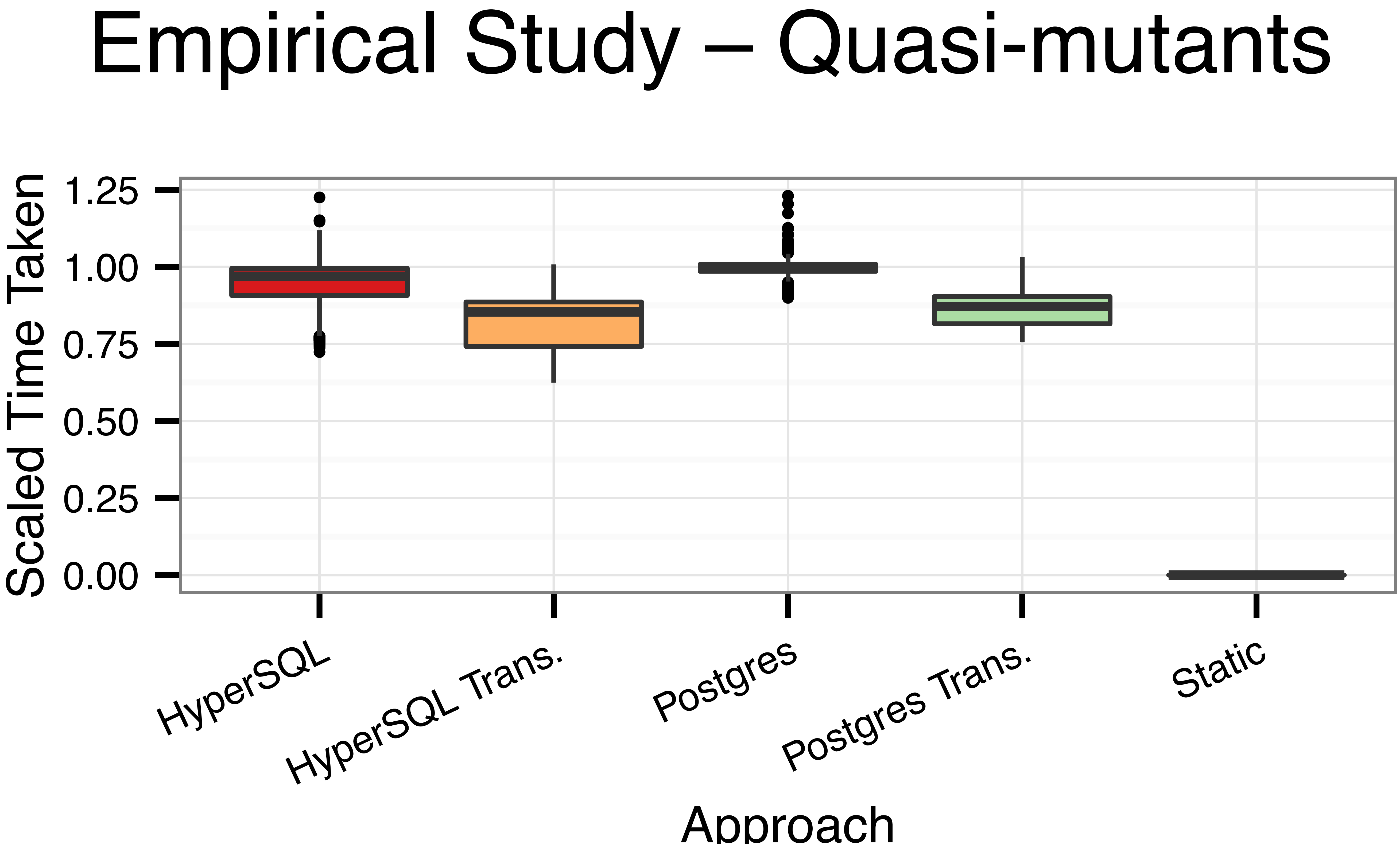
- 5 conditions:
  - Postgres (with/without transactions)
  - HyperSQL (with/without transactions)

Empirical Study – Quasi-mutants

- 5 conditions:
  - Postgres (with/without transactions)
  - HyperSQL (with/without transactions)
  - Static analysis

Empirical Study – Quasi-mutants

- 5 conditions:
  - Postgres (with/without transactions)
  - HyperSQL (with/without transactions)
  - Static analysis


Empirical Study – Quasi-mutants
Scaled Time Taken
1.25
1.00
0.75
0.50
0.25
0.00
HyperSQL
HyperSQL Trans.
Postgres
Postgres Trans.
Static
Approach

Empirical Study – Mutant Removal

Empirical Study – Mutant Removal

- 2 conditions – with and without removal

Empirical Study – Mutant Removal

- 2 conditions – with and without removal
- 2 metrics –

Empirical Study – Mutant Removal

- 2 conditions – with and without removal
- 2 metrics –
    - Time taken for mutation analysis

Empirical Study – Mutant Removal

- 2 conditions – with and without removal
- 2 metrics –
  - Time taken for mutation analysis
  - Mutation score

Empirical Study – Mutant Removal

Empirical Study – Mutant Removal

- HyperSQL – Time saved
    - Best case: 718ms (23.05%)
    - Worst case: -824ms (-9.71%)

Empirical Study – Mutant Removal

Empirical Study – Mutant Removal

- HyperSQL – Time saved

Empirical Study – Mutant Removal

- HyperSQL – Time saved
    - 9/16 mean time decrease (p < 0.05)

Empirical Study – Mutant Removal

- HyperSQL – Time saved
  - 9/16 mean time decrease (
  - 7/16 mean time increase ($p < 0.05$)

Empirical Study – Mutant Removal

- HyperSQL – Time saved
  - 9/16 mean time decrease (
  - 7/16 mean time increase (
  - Overall, decrease (1.6% mean, 1.4% median)

Empirical Study – Mutant Removal

Empirical Study – Mutant Removal

- PostgreSQL – Time saved
    - Best case: 317,208ms (33.71%)
    - Worst case: -3,086ms, (-0.33%)

Empirical Study – Mutant Removal

Empirical Study – Mutant Removal

- PostgreSQL – Time saved

Empirical Study – Mutant Removal

- PostgreSQL – Time saved
  - 14/16 mean time decrease ($p < 0.05$)

Empirical Study – Mutant Removal

- PostgreSQL – Time saved
    - 14/16 mean time decrease (
    - 2/16 mean time increase ($p < 0.05$)

Empirical Study – Mutant Removal

- PostgreSQL – Time saved
  - 14/16 mean time decrease (
  - 2/16 mean time increase (
  - Overall, decrease (12.7% mean, 11.8% median)

Empirical Study – Mutant Removal

DBMS	Time saved (ms)	
	Median	Mean

Empirical Study – Mutant Removal

DBMS	Time saved (ms)	
	Median	Mean
HyperSQL	36.2	7.5

Empirical Study – Mutant Removal

DBMS	Time saved (ms)	
	Median	Mean
HyperSQL	36.2	7.5
Postgres	8,071	50,880

Empirical Study – Mutant Removal

DBMS	Time saved (ms)	
	Median	Mean
HyperSQL	36.2	7.5
Postgres	8,071	50,880
Both	229.9	25,450

Empirical Study – Mutant Removal

DBMS	Time saved (%)	
	Median	Mean
HyperSQL	1.4	1.6
Postgres	12.7	11.8
Both	4.7	6.7

Empirical Study – Mutant Removal

Empirical Study – Mutant Removal

- HyperSQL – Mutation score
    - 75% Increased
        - 44% Adequate
    - 25% No change

Empirical Study – Mutant Removal

- PostgreSQL – Mutation score
  - 75% Increased
    - 44% Adequate
  - 25% No change

Empirical Study – Mutant Removal

DBMS	Scores changed (%)	
	Increased (adequate)	No change
HyperSQL	75 (44)	25
Postgres	75 (44)	25
Both	75 (44)	25

Conclusion

1. Quasi-mutant detection – DBMS v Static Analysis
2. Equivalent, Redundant and Quasi-mutant removal – Efficiency?
3. Equivalent, Redundant and Quasi-mutant removal – Effectiveness?

Conclusion

1. Quasi-mutant detection – DBMS v Static Analysis

2. Equivalent, Redundant and Quasi-mutant removal – Efficiency?

3. Equivalent, Redundant and Quasi-mutant removal – Effectiveness?

Conclusion

1. Quasi-mutant detection – DBMS v Static Analysis
2. Equivalent, Redundant and Quasi-mutant removal – Efficiency?
3. Equivalent, Redundant and Quasi-mutant removal – Effectiveness?

Conclusion

1. Quasi-mutant detection – ***Improved efficiency***

2. Equivalent, Redundant and Quasi-mutant removal – ***Improved efficiency***

3. Equivalent, Redundant and Quasi-mutant removal – ***Improved effectiveness***

Conclusion

1. Quasi-mutant detection – ***Improved efficiency***
2. Equivalent, Redundant and Quasi-mutant removal – ***Improved efficiency***
3. Equivalent, Redundant and Quasi-mutant removal – ***Improved effectiveness***

Chris J. Wright – aca08cw@sheffield.ac.uk