Localizing SQL Faults in Database Applications

Gregory M. Kapfhammer[†]

Sarah R. Clark[*], Jake Cobb[*],

James A. Jones[‡], and Mary Jean Harrold[*]

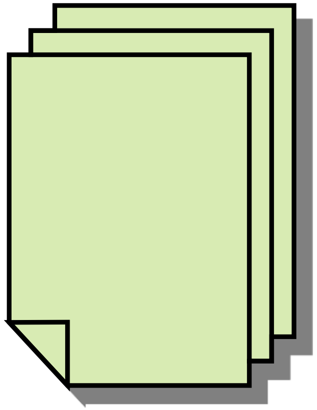
[*]Georgia Institute of Technology

[†]Allegheny College

[‡]University of California, Irvine

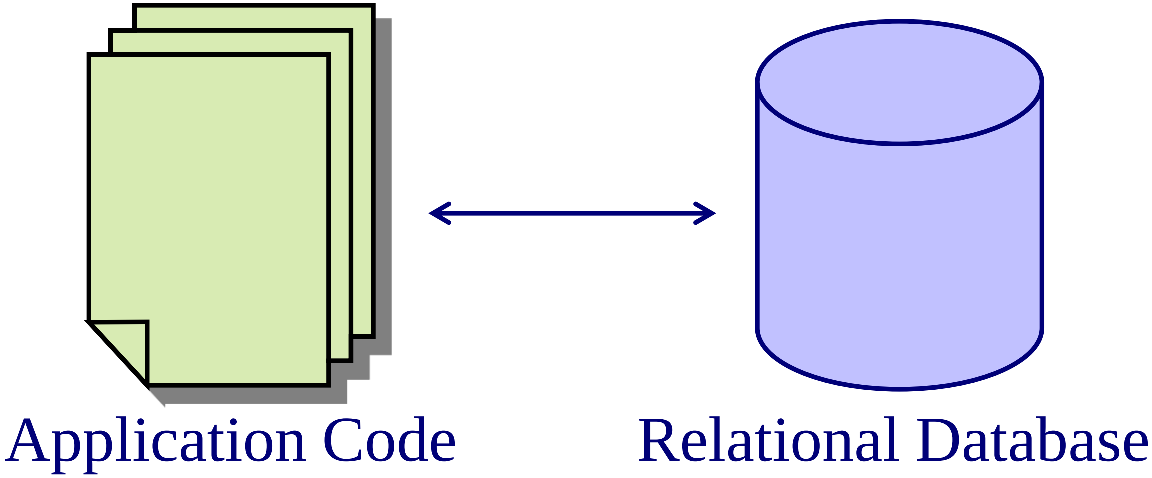
Supported by NSF CCF-1116943 and Google Faculty Research Award to UC Irvine, NSF CCF-0725202, CCF-0541048, IBM Software Quality Innovation Award, and InComm to Georgia Tech, and by SIGSOFT CAPS

Real-World Software Applications

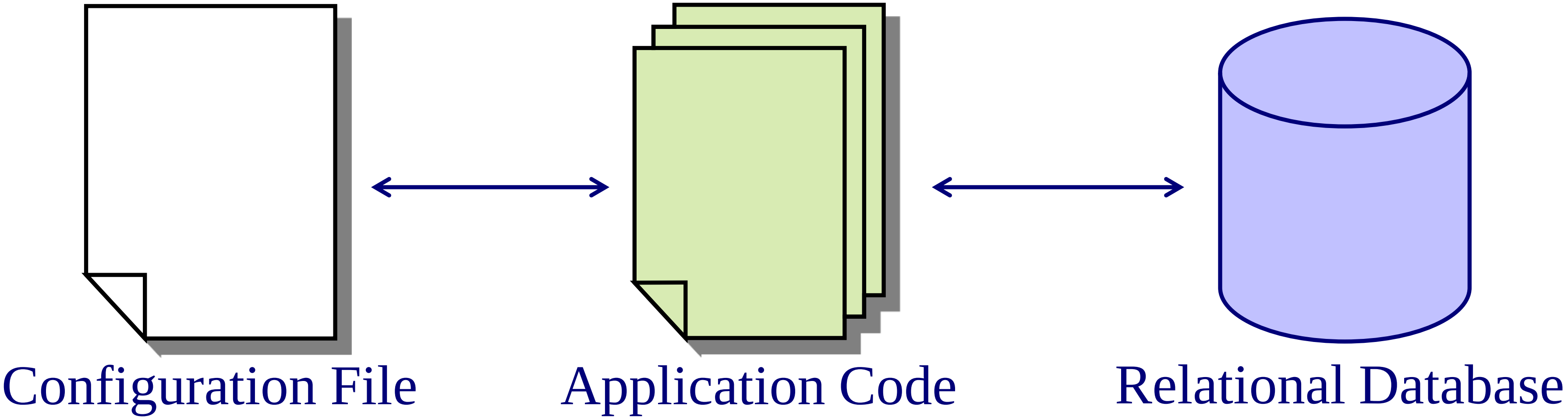


Application Code

Real-World Software Applications



Real-World Software Applications

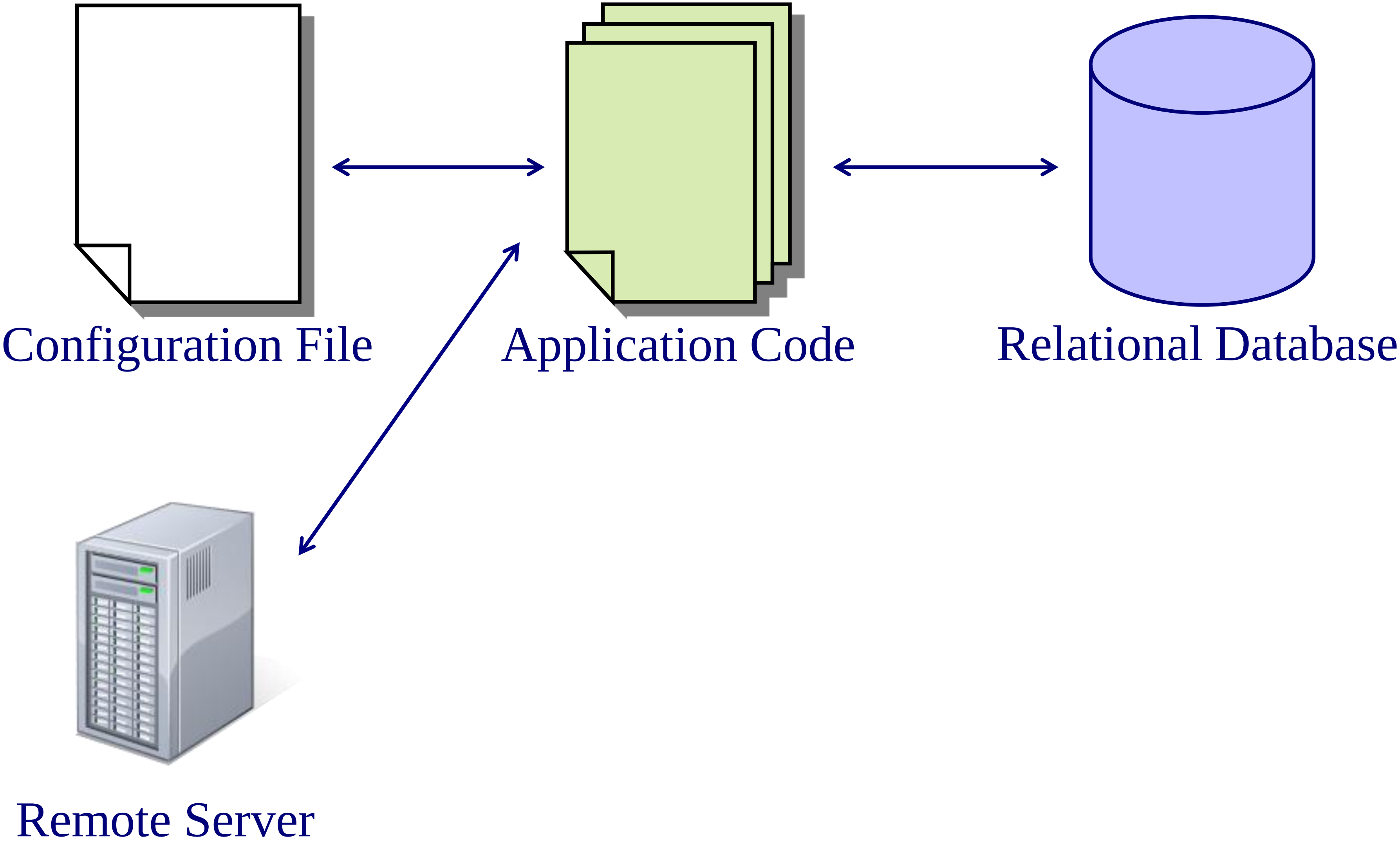


Configuration File

Application Code

Relational Database

Real-World Software Applications



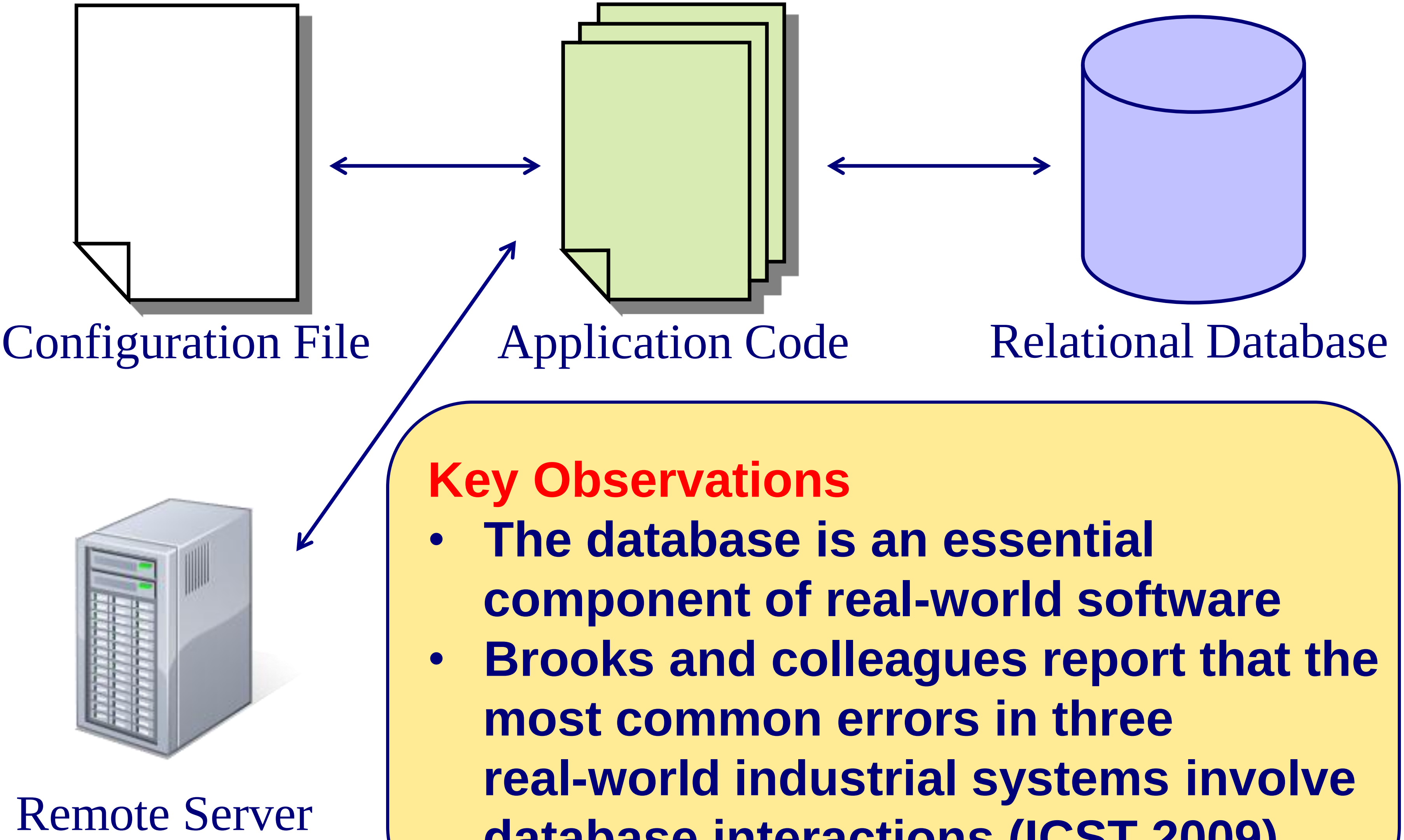
Configuration File

Application Code

Relational Database

Remote Server

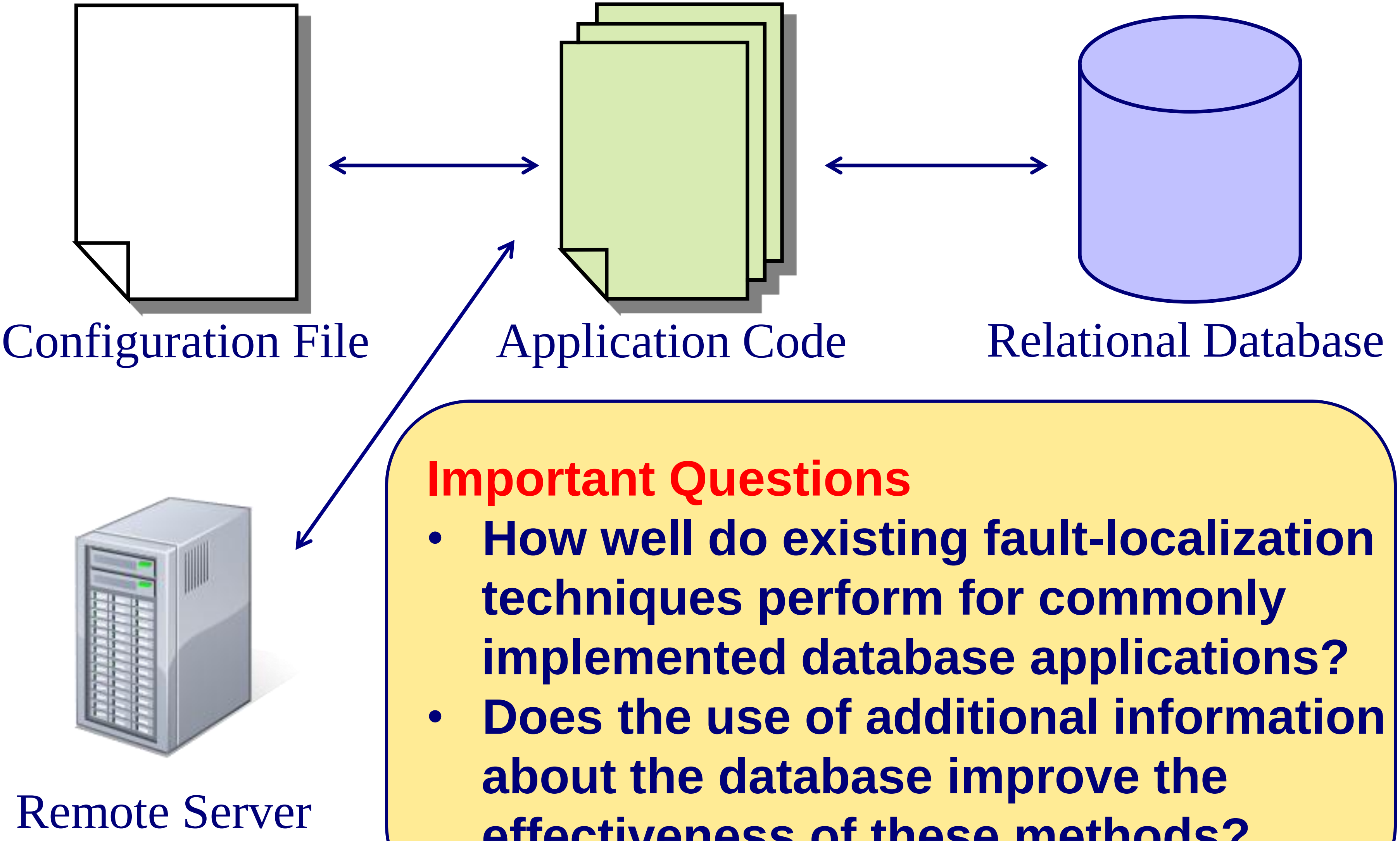
Real-World Software Applications



Key Observations

- **The database is an essential component of real-world software**
- **Brooks and colleagues report that the most common errors in three real-world industrial systems involve database interactions (ICST 2009)**

Real-World Software Applications



Important Questions

- **How well do existing fault-localization techniques perform for commonly implemented database applications?**
- **Does the use of additional information about the database improve the effectiveness of these methods?**

Motivating Example

```
printProdsold(String uType, String uID){
String attr=conf.getAttr(uType,uID);
String whereClause=conf.getWhere(uType,uID);
String SQL=“SELECT ”+attr+
           ”FROM Sale Where ”+whereClause;
PreparedStatement ps=new PreparedStatement();
ResultSet rs=ps.executeQuery(SQL);
printResultSet(rs);
}
```

Motivating Example

```
  printProdsold(String uType, String uID){
  String attr=conf.getAttr(uType,uID);
  String whereClause=conf.getWhere(uType,uID);
  String SQL=“SELECT ”+attr+
             ”FROM Sale Where ”+whereClause;
  PreparedStatement ps=new PreparedStatement();
  ResultSet rs=ps.executeQuery(SQL);
  printResultSet(rs);
}
```

Motivating Example

```
  printProdsold(String uType, String uID){
  String attr=conf.getAttr(uType,uID);
  String whereClause=conf.getWhere(uType,uID);
  String SQL=“SELECT ”+attr+
             ”FROM Sale Where ”+whereClause;
  PreparedStatement ps=new PreparedStatement();
  ResultSet rs=ps.executeQuery(SQL);
  printResultSet(rs);
}
```

Motivating Example

```
printProdsold(String uType, String uID){
String attr=conf.getAttr(uType,uID);
String whereClause=conf.getWhere(uType,uID);
String SQL=“SELECT ”+attr+
           ”FROM Sale Where ”+whereClause;
PreparedStatement ps=new PreparedStatement();
ResultSet rs=ps.executeQuery(SQL);
printResultSet(rs);
}
```

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

Database Table

Motivating Example

```
  printProdsold(String uType, String uID){
  String attr=conf.getAttr(uType,uID);
  String whereClause=conf.getWhere(uType,uID);
  String SQL=“SELECT ”+attr+
             ”FROM Sale Where ”+whereClause;
  PreparedStatement ps=new PreparedStatement();
  ResultSet rs=ps.executeQuery(SQL);
  printResultSet(rs);
}
```

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

Database Table

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Configuration File

Motivating Example

```
  printProdsold(String uType, String uID){
String attr=conf.getAttr(uType,uID);
String whereClause=conf.getWhere(uType,uID);
String SQL=“SELECT ”+attr+
           ”FROM Sale Where ”+whereClause;
PreparedStatement ps=new PreparedStatement();
ResultSet rs=ps.executeQuery(SQL);
printResultSet(rs);
}
```

Error in the whereClause!
>=
should be
=

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

Database Table

uType	Merchant (M)
attr	PRODUCT, PRICE
whereClause	MID>=uID
uType	Customer (C)
attr	PRODUCT, PRICE
whereClause	CID=uID

Configuration File

Statistical Fault Localization

```
printProdsold(String uType, String uID){
String attr=conf.getAttr(uType,uID);
String whereClause=conf.getWhere(uType,uID);
String SQL=“SELECT ”+attr+
           ”FROM Sale Where ”+whereClause;
PreparedStatement ps=new PreparedStatement();
ResultSet rs=ps.executeQuery(SQL);
printResultSet(rs);
}
```

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

Database Table

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Configuration File

Statistical Fault Localization

```
  printProdsold(String uType
String attr=conf.getAttr(u
String whereClause=conf.ge
String SQL=“SELECT ”+attr+
           ”FROM Sale Wher
PreparedStatement ps=new P
ResultSet rs=ps.executeQue
printResultSet(rs);
}
```

Techniques use:

Dynamic information

- **statements executed**
- **outcome (pass/fail)**

Statistical analysis

- **computes suspiciousness of each statement**

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

Database Table

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Configuration File

Statistical Fault Localization

printProdsold(String uType, String uID){	M,1	M,2	C,1	C,2	C,3
1:String attr=conf.getAttr(uType,uID);	•	•	•	•	•
2:String whereClause=conf.getWhere(uType,uID);	•	•	•	•	•
3:String SQL="SELECT "+attr+	•	•	•	•	•
"FROM Sale Where "+whereClause;	•	•	•	•	•
4:PreparedStatement ps=new PreparedStatement();	•	•	•	•	•
5:ResultSet rs=ps.executeQuery(SQL);	•	•	•	•	•
6:printResultSet(rs);	•	•	•	•	•
} **Pass/Fail Status**	**F**	**P**	**P**	**P**	**P**

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Statistical Fault Localization

$$suspiciousness(s) = \frac{failed(s)}{\sqrt{totalFailed(failed(s)+passed(s))}}$$

	M, 1	M, 2	C, 1	C, 2	C, 3
`printProdsold(String uType, String uID){`					
`1:String attr=conf.getAttr(uType,uID);`	●	●	●	●	●
`2:String whereClause=conf.getWhere(uType,uID);`	●	●	●	●	●
`3:String SQL=“SELECT ”+attr+`	●	●	●	●	●
`”FROM Sale Where ”+whereClause;`	●	●	●	●	●
`4:PreparedStatement ps=new PreparedStatement();`	●	●	●	●	●
`5:ResultSet rs=ps.executeQuery(SQL);`	●	●	●	●	●
`6:printResultSet(rs);`	●	●	●	●	●
`}` **Pass/Fail Status**	**F**	**P**	**P**	**P**	**P**

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Statistical Fault Localization

$$suspiciousness(s) = \frac{failed(s)}{\sqrt{totalFailed(failed(s) + passed(s))}}$$

	M, 1	M, 2	C, 1	C, 2	C, 3
`printProdsold(String uType, String uID){`					
`1:String attr=conf.getAttr(uType,uID);`	•	•	•	•	•
`2:String whereClause=conf.getWhere(uType,uID);`	•	•	•	•	•
`3:String SQL=“SELECT ”+attr+`	•	•	•	•	•
`”FROM Sale Where ”+whereClause;`	•	•	•	•	•
`4:PreparedStatement ps=new PreparedStatement();`	•	•	•	•	•
`5:ResultSet rs=ps.executeQuery(SQL);`	•	•	•	•	•
`6:printResultSet(rs);`	•	•	•	•	•
`}` **Pass/Fail Status**	**F**	**P**	**P**	**P**	**P**

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Statistical Fault Localization

$$suspiciousness(s) = \frac{failed(s)}{\sqrt{totalFailed(failed(s) + passed(s))}}$$

`printProdsold(String uType, String uID){`	M, 1	M, 2	C, 1	C, 2	C, 3
`1:String attr=conf.getAttr(uType,uID);`	●	●	●	●	●
`2:String whereClause=conf.getWhere(uType,uID);`	●	●	●	●	●
`3:String SQL="SELECT "+attr+`	●	●	●	●	●
`"FROM Sale Where "+whereClause;`	●	●	●	●	●
`4:PreparedStatement ps=new PreparedStatement();`	●	●	●	●	●
`5:ResultSet rs=ps.executeQuery(SQL);`	●	●	●	●	●
`6:printResultSet(rs);`	●	●	●	●	●
`}` **Pass/Fail Status**	F	P	P	P	P

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Statistical Fault Localization

$$suspiciousness(s) = \frac{failed(s)}{\sqrt{totalFailed(failed(s) + passed(s))}}$$

printProdsold(String uType, String uID){	M, 1	M, 2	C, 1	C, 2	C, 3
1:String attr=conf.getAttr(uType,uID);	●	●	●	●	●
2:String whereClause=conf.getWhere(uType,uID);	●	●	●	●	●
3:String SQL="SELECT "+attr+	●	●	●	●	●
"FROM Sale Where "+whereClause;	●	●	●	●	●
4:PreparedStatement ps=new PreparedStatement();	●	●	●	●	●
5:ResultSet rs=ps.executeQuery(SQL);	●	●	●	●	●
6:printResultSet(rs);	●	●	●	●	●
} **Pass/Fail Status**	**F**	**P**	**P**	**P**	**P**

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Statistical Fault Localization

$$\mathit{suspiciousness}(s) = \frac{\mathit{failed}(s)}{\sqrt{\mathit{totalFailed}(\mathit{failed}(s)+\mathit{passed}(s))}}$$

printProdsold(String uType, String uID){	M,1	M,2	C,1	C,2	C,3
1:String attr=conf.getAttr(uType,uID);	•	•	•	•	•
2:String whereClause=conf.getWhere(uType,uID);	•	•	•	•	•
3:String SQL=“SELECT ”+attr+	•	•	•	•	•
”FROM Sale Where ”+whereClause;	•	•	•	•	•
4:PreparedStatement ps=new PreparedStatement();	•	•	•	•	•
5:ResultSet rs=ps.executeQuery(SQL);	•	•	•	•	•
6:printResultSet(rs);	•	•	•	•	•
} Pass/Fail Status	F	P	P	P	P

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Statistical Fault Localization

$$\mathit{suspiciousness}(s) = \frac{\mathit{failed}(s)}{\sqrt{\mathit{totalFailed}(\mathit{failed}(s) + \mathit{passed}(s))}}$$

printProdsold(String uType, String uID){	M, 1	M, 2	C, 1	C, 2	C, 3
1:String attr=conf.getAttr(uType,uID);	•	•	•	•	•
2:String whereClause=conf.getWhere(uType,uID);	•	•	•	•	•
3:String SQL=“SELECT ”+attr+	•	•	•	•	•
”FROM Sale Where ”+whereClause;	•	•	•	•	•
4:PreparedStatement ps=new PreparedStatement();	•	•	•	•	•
5:ResultSet rs=ps.executeQuery(SQL);	•	•	•	•	•
6:printResultSet(rs);	•	•	•	•	•
} **Pass/Fail Status**	**F**	**P**	**P**	**P**	**P**

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Statistical Fault Localization

$$suspiciousness(s) = \frac{failed(s)}{\sqrt{totalFailed(failed(s) + passed(s))}}$$

printProdsold(String uType, String uID){	M, 1	M, 2	C, 1	C, 2	C, 3	suspiciousness
1:String attr=conf.getAttr(uType,uID);	•	•	•	•	•	
2:String whereClause=conf.getWhere(uType,uID);	•	•	•	•	•	
3:String SQL=“SELECT ”+attr+	•	•	•	•	•	
”FROM Sale Where ”+whereClause;	•	•	•	•	•	
4:PreparedStatement ps=new PreparedStatement();	•	•	•	•	•	
5:ResultSet rs=ps.executeQuery(SQL);	•	•	•	•	•	
6:printResultSet(rs);	•	•	•	•	•	
} **Pass/Fail Status**	F	P	P	P	P	

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Statistical Fault Localization

$$suspiciousness(s) = \frac{failed(s)}{\sqrt{totalFailed(failed(s)+passed(s))}}$$

printProdsold(String uType, String uID){	M, 1	M, 2	C, 1	C, 2	C, 3	suspiciousness
1:String attr=conf.getAttr(uType,uID);	●	●	●	●	●	
2:String whereClause=conf.getWhere(uType,uID);	●	●	●	●	●	
3:String SQL="SELECT "+attr+	●	●	●	●	●	
"FROM Sale Where "+whereClause;	●	●	●	●	●	
4:PreparedStatement ps=new PreparedStatement();	●	●	●	●	●	
5:ResultSet rs=ps.executeQuery(SQL);	●	●	●	●	●	
6:printResultSet(rs);	●	●	●	●	●	
} Pass/Fail Status	F	P	P	P	P	

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Statistical Fault Localization

$$suspiciousness(s) = \frac{failed(s)}{\sqrt{totalFailed(failed(s)+passed(s))}}$$

	M, 1	M, 2	C, 1	C, 2	C, 3	suspiciousness
`printProdsold(String uType, String uID){`						
`1:String attr=conf.getAttr(uType,uID);`	●	●	●	●	●	0.45
`2:String whereClause=conf.getWhere(uType,uID);`	●	●	●	●	●	0.45
`3:String SQL="SELECT "+attr+`	●	●	●	●	●	0.45
`"FROM Sale Where "+whereClause;`	●	●	●	●	●	0.45
`4:PreparedStatement ps=new PreparedStatement();`	●	●	●	●	●	0.45
`5:ResultSet rs=ps.executeQuery(SQL);`	●	●	●	●	●	0.45
`6:printResultSet(rs);`	●	●	●	●	●	0.45
`}` **Pass/Fail Status**	F	P	P	P	P	

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Statistical Fault Localization

$$suspiciousness(s) = \frac{failed(s)}{\sqrt{totalFailed(failed(s) + passed(s))}}$$

printProdsold(String uType, String uID){	M, 1	M, 2	C, 1	C, 2	C, 3	suspiciousness
1:String attr=conf.getAttr(uType,uID);	•	•	•	•	•	0.45
2:String whe				•	•	0.45
3:String SQ					•	0.45
					•	0.45
4:Prepared					•	0.45
5:ResultSe					•	0.45
6:printRes					•	0.45
}					P	

Important Challenges to Overcome

- Statistical fault-localization assigns the same suspiciousness scores to all of the statements
- Existing methods do not consider the state or structure of the database

MID	CID		
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

whereClause	CE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Statistical Fault Localization

printProdsold(String uType, String uID){	M, 1	M, 2	C, 1	C, 2	C, 3	suspiciousness
1:String attr=conf.getAttr(uType,uID);	•	•	•	•	•	0.45
2:String whe				•	•	0.45
3:String SQ					•	0.45
					•	0.45
4:PreparedS					•	0.45
5:ResultSe					•	0.45
6:printRes					•	0.45
}					P	

Our database-aware fault-localization technique has two goals for SQL faults

1. **Localize on the faulty statement-SQL or statement-attribute tuple**
2. **Provide extra information about the SQL commands executed by tests**

MID	CID		
1	1		
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

	CE
whereClause	MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Outline for the Rest of the Presentation

- Our Technique
  - Definitions
  - Algorithm
- Empirical Studies
- Conclusion

Outline for the Rest of the Presentation

- Our Technique
    - Definitions
    - Algorithm
- Empirical Studies
- Conclusion

Our Technique—Definitions

```
printProdsold(String uType,
String attr=conf.getAttr(uT
String whereClause=conf.get
String SQL=‶SELECT ″+attr+
           ″FROM Sale Where ″+whereClause;
PreparedStatement ps=new PreparedStatement();
ResultSet rs=ps.executeQuery(SQL);
printResultSet(rs);
}
```

Database Interaction Point

Location in the source code where control and data transfer from the application to the database and back

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Our Technique—Definitions

Database Interaction Point

Location in the source code where control and data transfer from the application to the database and back

```
  printProdsold(String uType,
String attr=conf.getAttr(uT
String whereClause=conf.get
String SQL=“SELECT ”+attr+
           ”FROM Sale Where ”+     eClause;
PreparedStatement ps=new PreparedStatement();
ResultSet rs=ps.executeQuery(SQL);
printResultSet(rs);
}
```

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Our Technique—Definitions

Statement-SQL Tuple

- <*s*,*c*> where c is an SQL command executed by a statement *s*
- Record the set of <*s*,*c*> executed by each test case *t* in test suite *T*

```
  printProdsold(Stri
String attr=conf.g
String whereClause
String SQL="SELECT
           ”FROM Sale where "+whereClause;
PreparedStatement ps=new PreparedStatement();
ResultSet rs=ps.executeQuery(SQL);
printResultSet(rs);
}
```

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Our Technique—Definitions

Statement-SQL Tuple

- <*s*,*c*> where c is an SQL command executed by a statement *s*
- Record the set of <*s*,*c*> executed by each test case *t* in test suite *T*

```
  printProdsold(Stri
1:String attr=conf.g
2:String whereClause
3:String SQL="SELECT
           "FROM Sale Wh                 eClause;
4:PreparedStatement ps=new          edStatement();
5:ResultSet rs=ps.execute        (SQL);
6:printResultSet(rs);
}
```

<5, SELECT PRODUCT, PRICE FROM Sale WHERE MID>=?>

<5, SELECT PRODUCT, PRICE FROM Sale WHERE CID=?>

MID			
1			
1			
2	2	Hammer	5.00
2	3	Nails	0.50

	E
attr whereClause	PRODUCT, PRICE CID=uID

Our Technique—Definitions

Statement-Attribute Tuple

- <*s*,*a*> where *a* is an attribute appearing in one or more commands *c* executed at statement *s*
- Record the set of <*s*,*a*> executed by each test case *t* in test suite *T*
- Saved only when multiple unique SQL commands are executed at statement *s*

```
1
2                                            ));
3

4                                            ();
5:
6:printResultSet(rs);
}
```

MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	Merchant (M) PRODUCT, PRICE MID>=uID
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

Our Technique—Definitions

Statement-Attribute Tuple

- <*s*,*a*> where *a* is an attribute appearing in one or more commands *c* executed at statement *s*
- Record the set of <*s*,*a*> executed by each test case *t* in test suite *T*
- Saved only when multiple unique SQL commands are executed at statement *s*

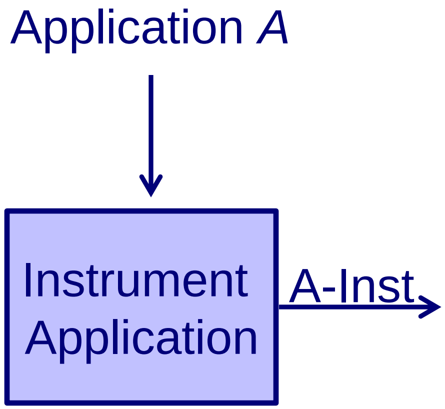
```
1
2                                              ));
3

4                                              ();
5:
6:printResultSet(rs);
}
```

<5, PRODUCT>
<5, PRICE>
<5,MID>
<5,CID>

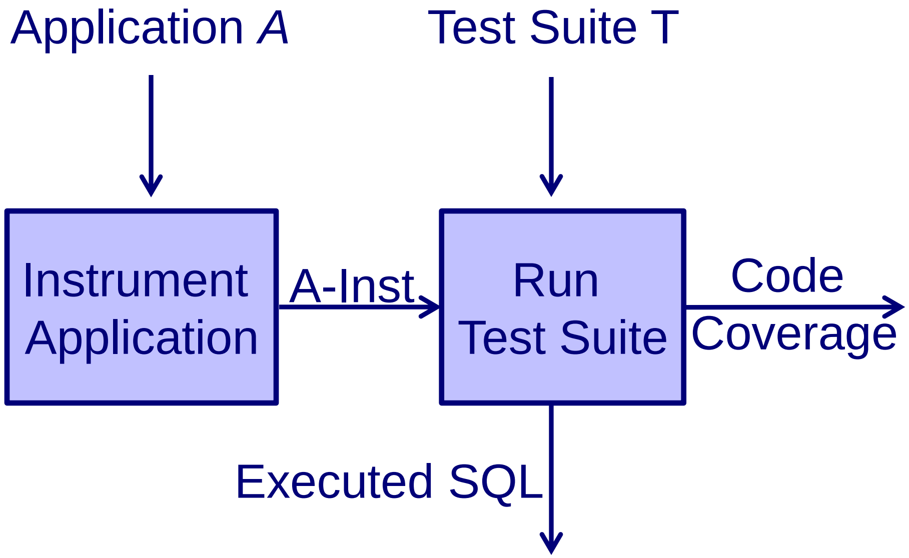
MID	CID	PROD	PRICE
1	1	Soda	$0.99
1	3	Cheese	3.99
2	2	Hammer	5.00
2	3	Nails	0.50

uType attr whereClause	M PR M
uType attr whereClause	Customer (C) PRODUCT, PRICE CID=uID

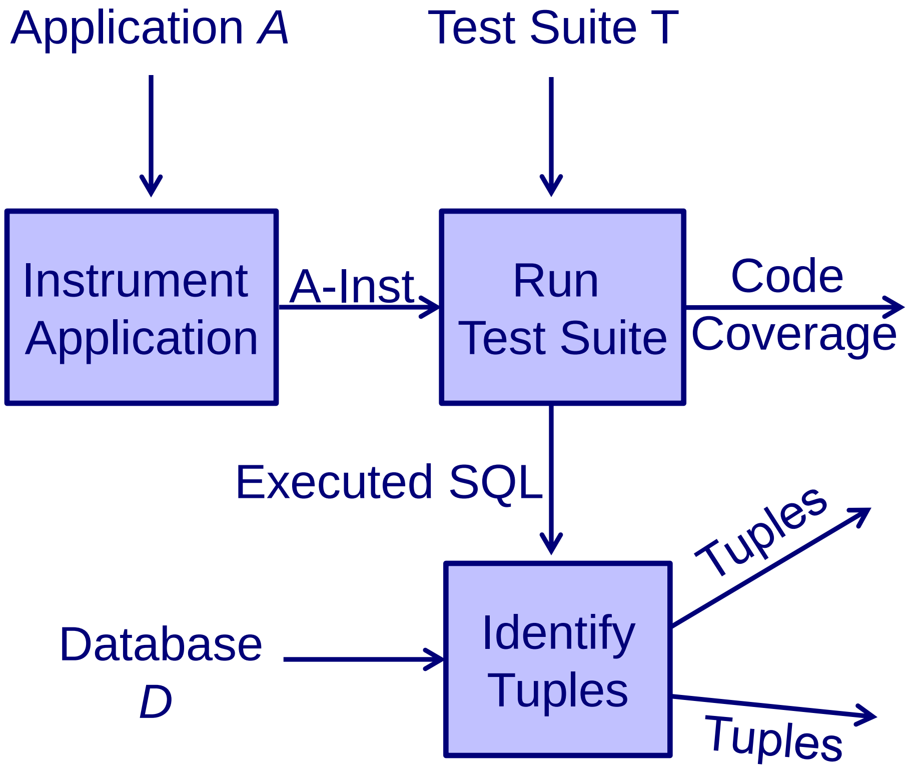
Our Technique—Algorithm



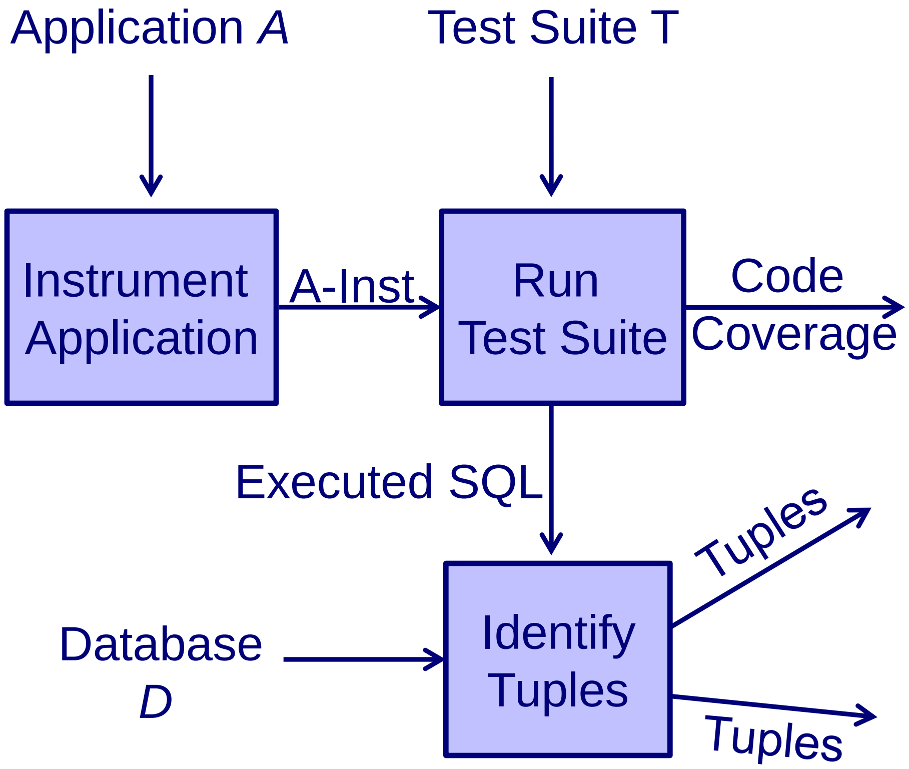
Our Technique—Algorithm



Our Technique—Algorithm



Our Technique—Algorithm



Revisiting the Example

Our Technique—Algorithm Example

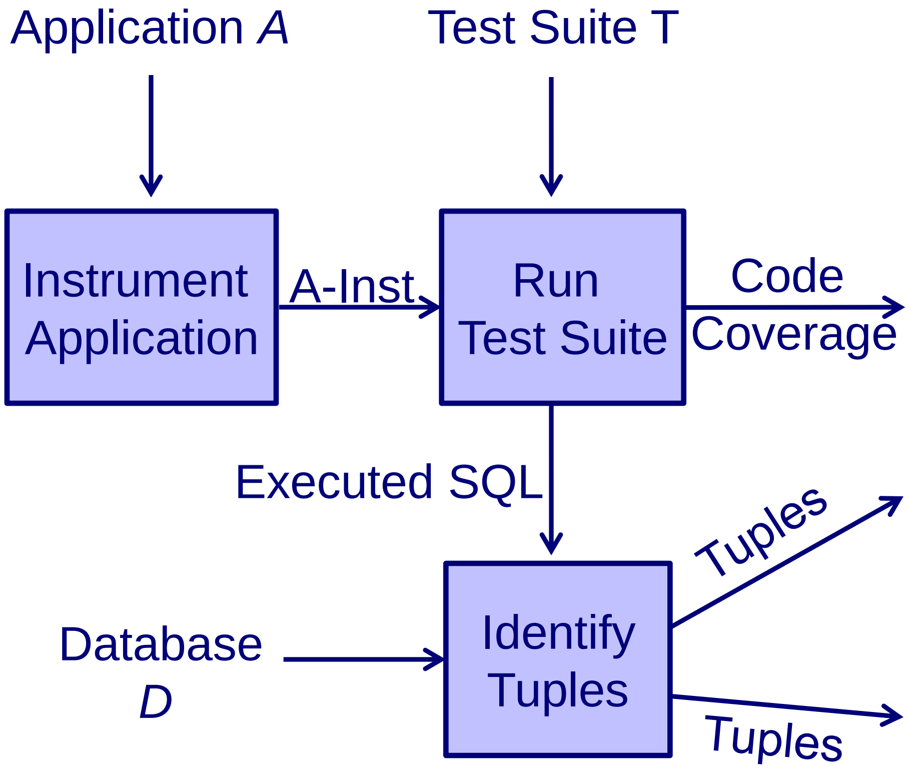
	M, 1	M, 2	C, 1	C, 2	C, 3
printProdsold(String uType, String uID){					
1:String attr=conf.getAttr(uType,uID);	●	●	●	●	●
2:String whereClause=conf.getWhere(uType,uID);	●	●	●	●	●
3:String SQL=“SELECT ”+attr+	●	●	●	●	●
”FROM Sale Where ”+whereClause;	●	●	●	●	●
4:PreparedStatement ps=new PreparedStatement();	●	●	●	●	●
5:ResultSet rs=ps.executeQuery(SQL);	●	●	●	●	●
<5,SELECT…WHERE MID>=?>	●	●			
<5,SELECT…WHERE CID=?>			●	●	●
<5,PRODUCT>	●	●	●	●	●
<5,PRICE>	●	●	●	●	●
<5,MID>	●	●			
<5,CID>			●	●	●
6:printResultSet(rs);	●	●	●	●	●
} Pass/Fail Status	F	P	P	P	P

Identify Statement-SQL and Statement-Attribute Tuples

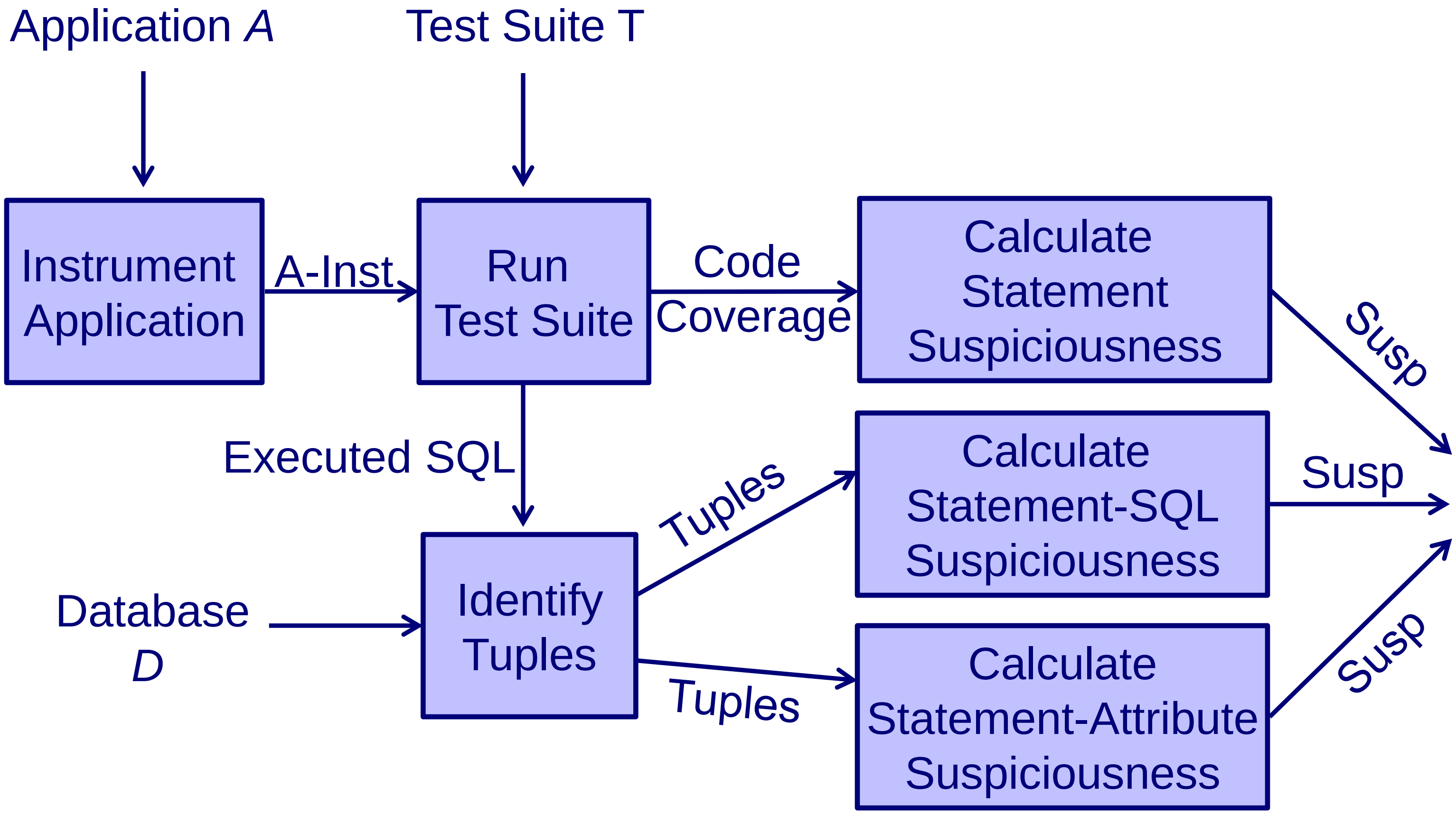
<5, SELECT PRODUCT, PRICE FROM Sale WHERE MID>=?>

<5, PRODUCT>, <5, PRICE>, <5,MID>, <5,CID>

Our Technique—Algorithm



Our Technique—Algorithm



Application *A*

Test Suite T

Instrument Application

A-Inst

Run Test Suite

Code Coverage

Calculate Statement Suspiciousness

Susp

Executed SQL

Tuples

Calculate Statement-SQL Suspiciousness

Susp

Database *D*

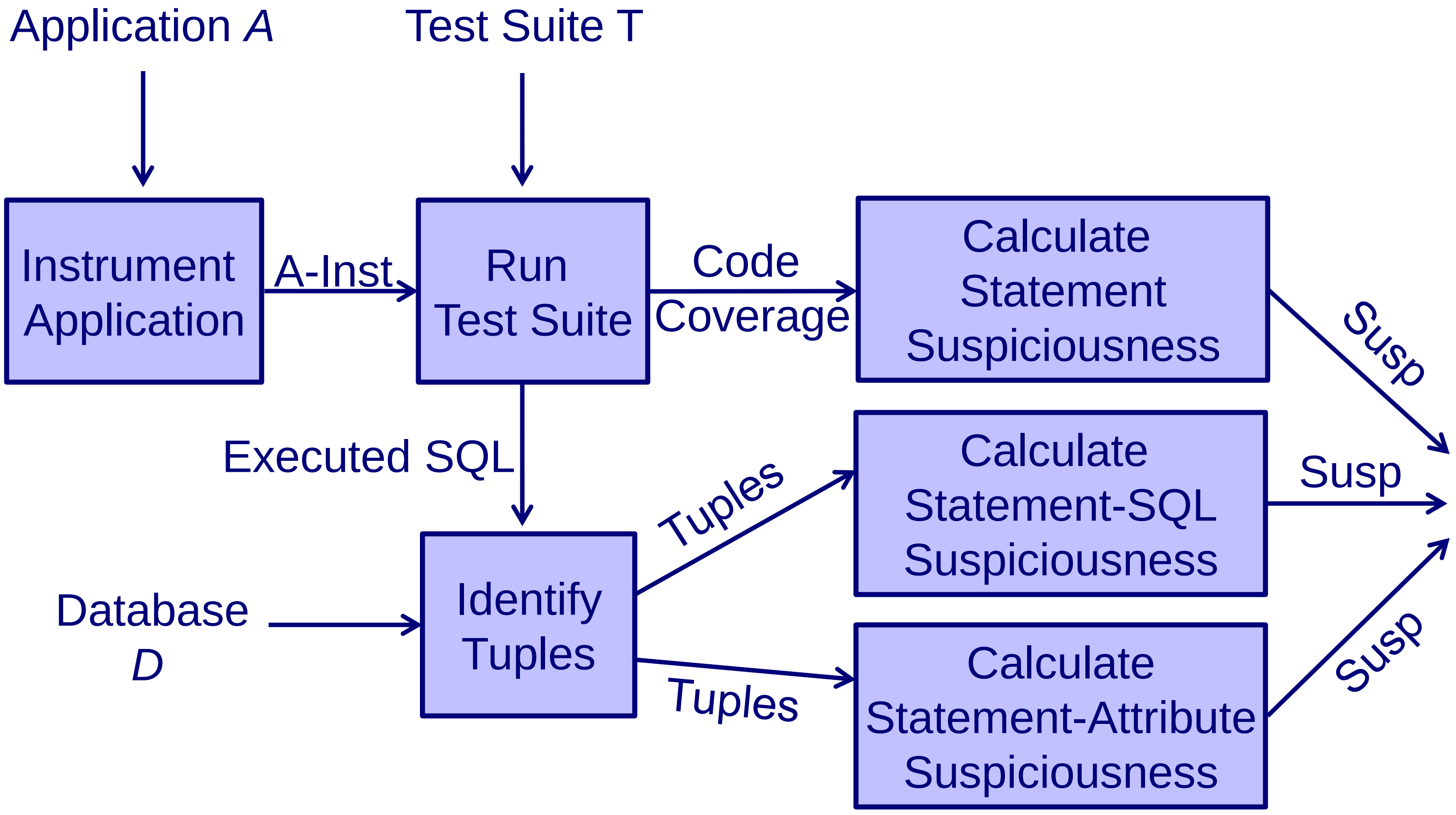
Identify Tuples

Tuples

Calculate Statement-Attribute Suspiciousness

Susp

Our Technique—Algorithm



Revisiting the Example

Our Technique—Algorithm Example

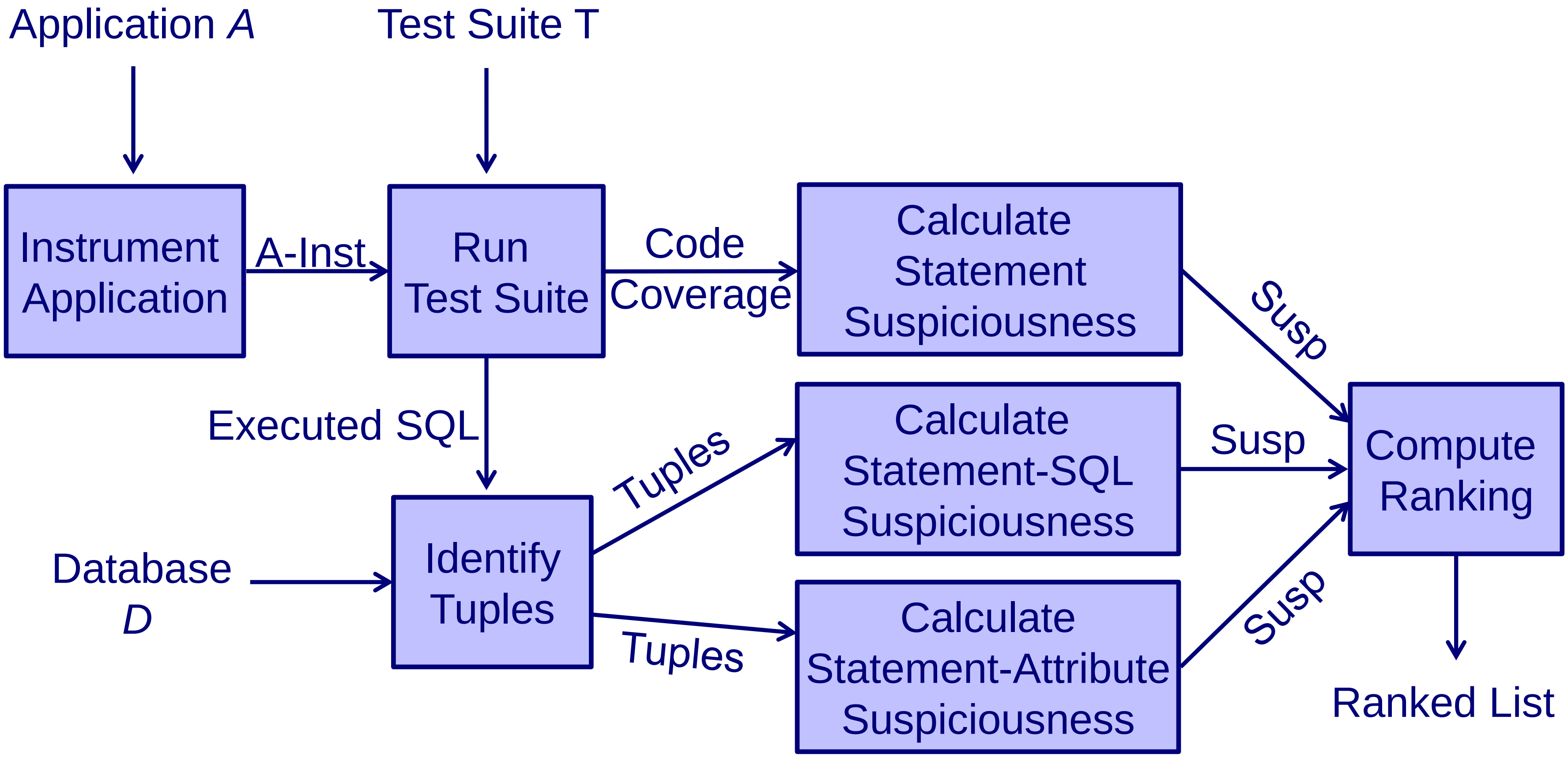
printProdsold(String uType, String uID){	M, 1	M, 2	C, 1	C, 2	C, 3	suspiciousness
1:String attr=conf.getAttr(uType,uID);	•	•	•	•	•	0.45
2:String whereClause=conf.getWhere(uType,uID);	•	•	•	•	•	0.45
3:String SQL=“SELECT ”+attr+	•	•	•	•	•	0.45
”FROM Sale Where ”+whereClause;	•	•	•	•	•	0.45
4:PreparedStatement ps=new PreparedStatement();	•	•	•	•	•	0.45
5:ResultSet rs=ps.executeQuery(SQL);	•	•	•	•	•	0.45
<5,SELECT…WHERE MID>=?>	•	•				0.71
<5,SELECT…WHERE CID=?>			•	•	•	0.00
<5,PRODUCT>	•	•	•	•	•	0.45
<5,PRICE>	•	•	•	•	•	0.45
<5,MID>	•	•				0.71
<5,CID>			•	•	•	0.00
6:printResultSet(rs);	•	•	•	•	•	0.45
} Pass/Fail Status	F	P	P	P	P	

Calculate Suspiciousness

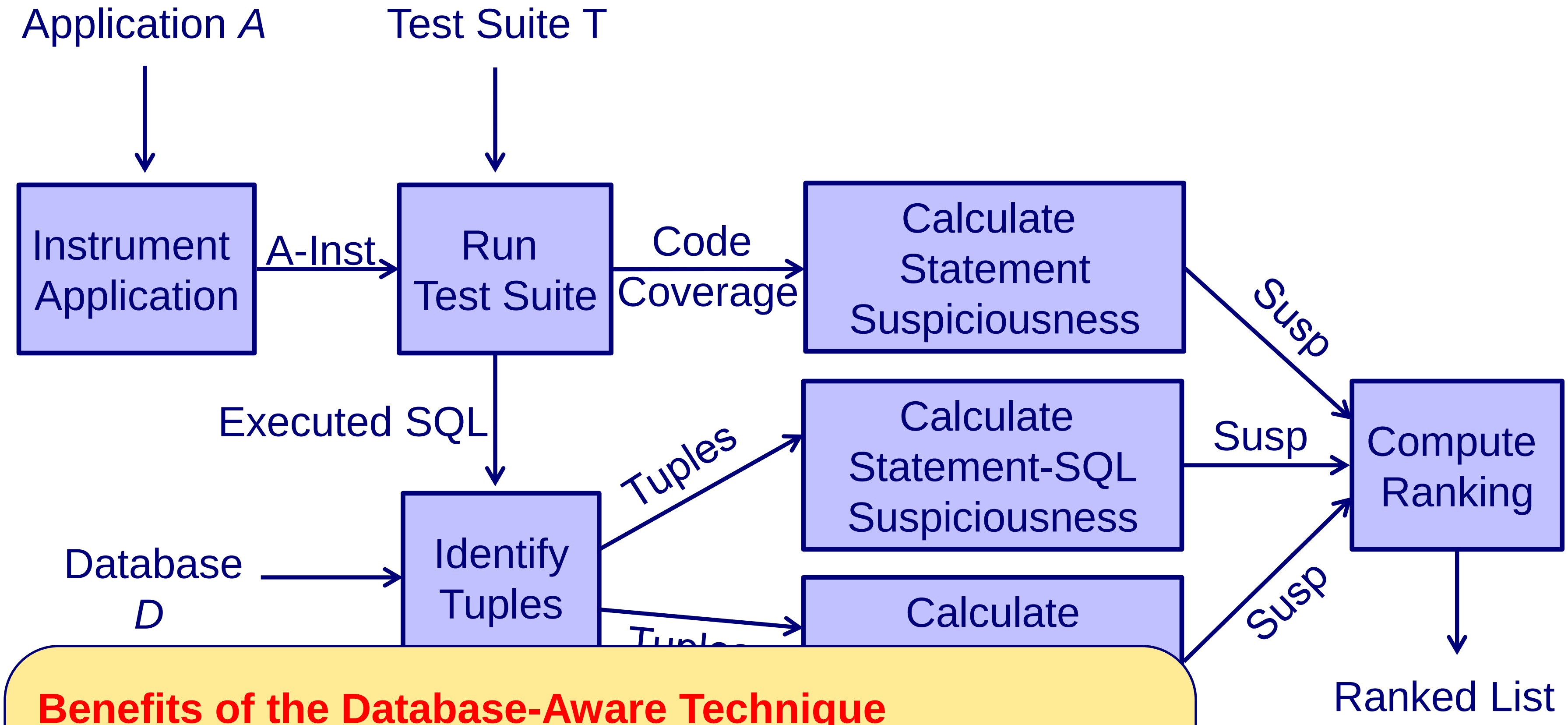
<5, MID>: Passed=1, Failed=1, Total Failed=1

Suspiciousness=1/sqrt(1(1+1)=0.71

Our Technique—Algorithm



Our Technique—Algorithm



Benefits of the Database-Aware Technique

Finds the faulty

1. Database interaction point

2. SQL command

3. Attribute in the SQL clause

Outline for the Rest of the Presentation

- Our Technique
  - Definitions
  - Algorithm
- Empirical Studies
- Conclusion

Empirical Studies

Implementation

- Cobertura: Collect per-test case coverage reports
- P6Spy: Record the executed SQL statements
- Unity: Parse statements in multiple versions of SQL

Empirical Studies

Implementation

- Cobertura: Collect per-test case coverage reports
- P6Spy: Record the executed SQL statements
- Unity: Parse statements in multiple versions of SQL

Subjects	Java LOC	Test Cases	Tables (DB)	Interaction Points (DB)	Type (DB)	Description
MessageSwitch	3672	80	15	16	Oracle	Transaction processing system
JWhoisServer	6684	79	10	2	HSQLDB	Open source WHOIS server
iTrust	25517	802	30	157	MySQL	Medical application (NC State)

Empirical Studies

Setup

- Identified types of mutants
  - Code mutants—code faults in the application
  - SQL mutants—SQL faults in the application
- Created the mutants manually
  - Existing tools couldn’t process our subjects
  - Followed an established approach (*IST* 49(4), 2007)

Empirical Studies

Setup

- Identified types of mutants
  - Code mutants—code faults in the application
  - SQL mutants—SQL faults in the application
- Created the mutants manually
  - Existing tools couldn’t process our subjects
  - Followed an established approach (*IST* 49(4), 2007)
- Resulting mutants

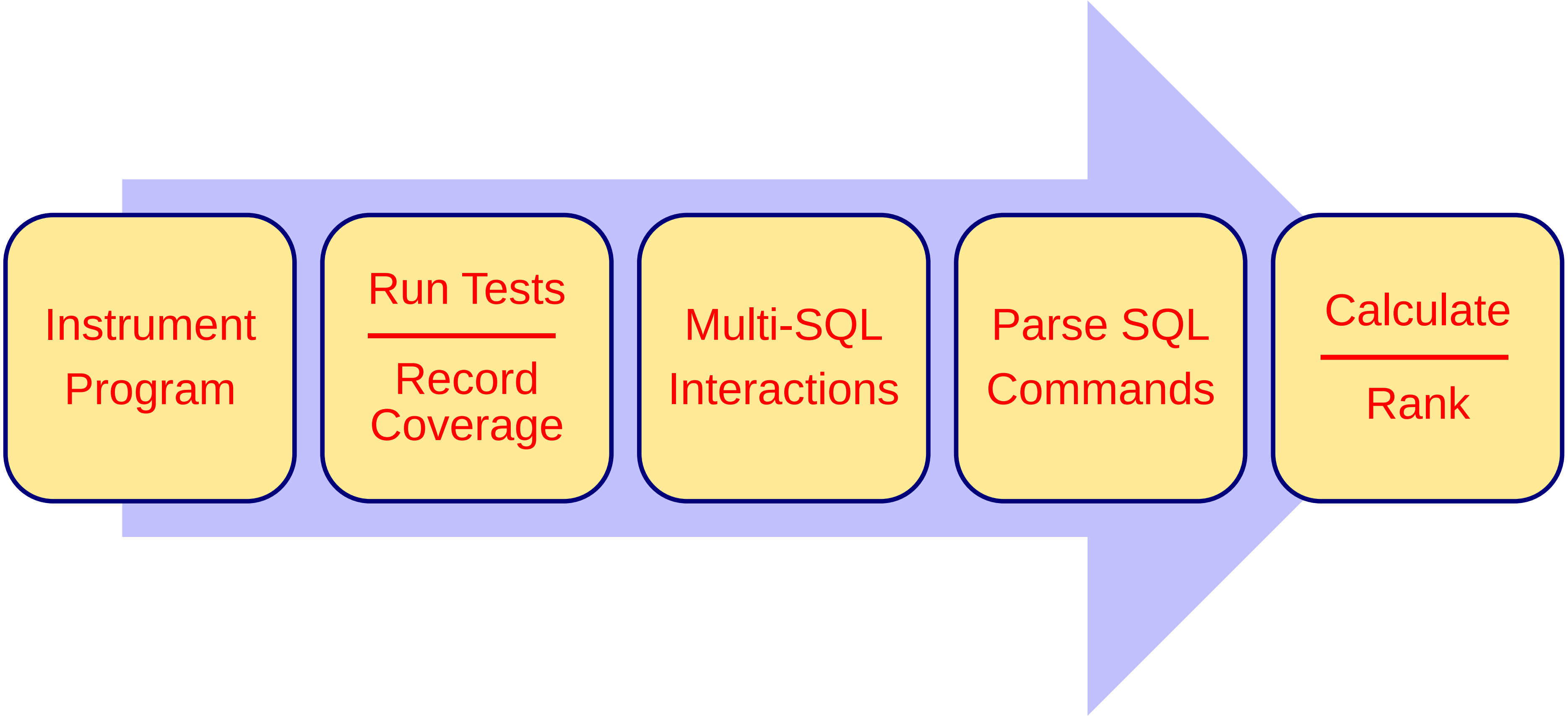
Subjects	Code Mutants	SQL Mutants
MessageSwitch	100	15
JWhoisServer	50	10
iTrust	25	30

Study 1—Effectiveness

- Goal Compare the database-aware approach to statement-based fault localization for SQL and code faults

Study 1—Effectiveness

- Goal Compare the database-aware approach to statement-based fault localization for SQL and code faults
- Method For each mutant in the program



Study 1—Results

Subject	Fault Type	Statement 99%	Database 99%	Statement 90%	Database 90%
MessageSwitch	SQL	50%	67%	100%	100%
	Code	26%	26%	68%	68%
	All	32%	36%	76%	76%
JWhoisServer	SQL	0%	95%	87%	100%
	Code	17%	13%	61%	61%
	All	7%	63%	77%	85%
iTrust	SQL	94%	94%	100%	100%
	Code	98%	98%	98%	100%
	All	97%	97%	98%	100%

Study 1—Results

Subject	Fault Type	Statement 99%	Database 99%	Statement 90%	Database 90%
MessageSwitch	SQL	50%	67%	100%	100%
	Code	26%	26%	68%	68%
	All	32%	36%	76%	76%
JWhoisServer	SQL	0%	95%	87%	100%
	Code	17%	13%	61%	61%
	All	7%	63%	77%	85%
iTrust	SQL	94%	94%	100%	100%
	Code	98%	98%	98%	100%
	All	97%	97%	98%	100%

For each case study application, measure fault localization effectiveness for SQL and code faults

Study 1—Results

Subject	Fault Type	Statement 99%	Database 99%	Statement 90%	Database 90%
MessageSwitch	SQL	50%	67%	100%	100%
	Code	26%	26%	68%	68%
	All	32%	36%	76%	76%
JWhoisServer	SQL	0%	95%	87%	100%
	Code	17%	13%	61%	61%
	All	7%	63%	77%	85%
iTrust	SQL	94%	94%	100%	100%
	Code	98%	98%	98%	100%
	All	97%	97%	98%	100%

Measured the percentage of faults found without examining 99% and 90% of the subject’s source code

Study 1—Results

Subject	Fault Type	Statement 99%	Database 99%	Statement 90%	Database 90%
MessageSwitch	SQL	50%	67%	100%	100%
	Code	26%	26%	68%	68%
	All	32%	36%	76%	76%
JWhoisServer	SQL	0%	95%	87%	100%
	Code	17%	13%	61%	61%
	All	7%	63%	77%	85%
iTrust	SQL	94%	94%	100%	100%
	Code	98%	98%	98%	100%
	All	97%	97%	98%	100%

Higher values indicate a more effective fault localization method

Study 1—Discussion

	Fault	Statement	Database	Statement	Database
Subject	Type	99%	99%	90%	90%
MessageSwitch	SQL	50%	67%	100%	100%
	Code	26%	26%	68%	68%
	All	32%	36%	76%	76%
JWhoisServer	SQL	0%	95%	87%	100%
	Code	17%	13%	61%	61%
	All	7%	63%	77%	85%
iTrust	SQL	94%	94%	100%	100%
	Code	98%	98%	98%	100%
	All	97%	97%	98%	100%

Statement-based fault localization finds 0% of the SQL faults without examining 99% of statements

Study 1—Discussion

Subject	Fault Type	Statement 99%	Database 99%	Statement 90%	Database 90%
MessageSwitch	SQL	50%	67%	100%	100%
	Code	26%	26%	68%	68%
	All	32%	36%	76%	76%
JWhoisServer	SQL	0%	95%	87%	100%
	Code	17%	13%	61%	61%
	All	7%	63%	77%	85%
iTrust	SQL	94%	94%	100%	100%
	Code	98%	98%	98%	100%
	All	97%	97%	98%	100%

Database-aware fault localization finds 95% of the SQL faults without examining 99% of statements

Study 1—Discussion

Subject	Fault Type	Statement 99%	Database 99%	Statement 90%	Database 90%
MessageSwitch	SQL	50%	67%	100%	100%
	Code	26%	26%	68%	68%
	All	32%	36%	76%	76%
JWhoisServer	SQL	0%	95%	87%	100%
	Code	17%	13%	61%	61%
	All	7%	63%	77%	85%
iTrust	SQL	94%	94%	100%	100%
	Code	98%	98%	98%	100%
	All	97%	97%	98%	100%

Statement-based fault localization works well for applications with static database interactions

Study 1—Discussion

Subject	Fault Type	Statement 99%	Database 99%	Statement 90%	Database 90%
MessageSwitch	SQL	50%	67%	100%	100%
	Code	26%	26%	68%	68%
	All	32%	36%	76%	76%
JWhoisServer	SQL	0%	95%	87%	100%
	Code	17%	13%	61%	61%
	All	7%	63%	77%	85%
iTrust	SQL	94%	94%	100%	100%
	Code	98%	98%	98%	100%
	All	97%	97%	98%	100%

When improvement is unlikely, database-aware fault localization does not degrade effectiveness

Study 1—Discussion

Subject	Fault Type	Statement 99%	Database 99%	Statement 90%	Database 90%
MessageSwitch	SQL	50%	67%	100%	100%
	Code	26%	26%	68%	68%
	All	32%	36%	76%	76%
JWhoisServer	SQL	0%	95%	87%	100%
	Code	17%	13%	61%	61%
	All	7%	63%	77%	85%
iTrust	SQL	94%	94%	100%	100%
	Code	98%	98%	98%	100%
	All	97%	97%	98%	100%

The database-aware technique is most useful for database applications with dynamic interactions

Study 2—Qualitative Case Study

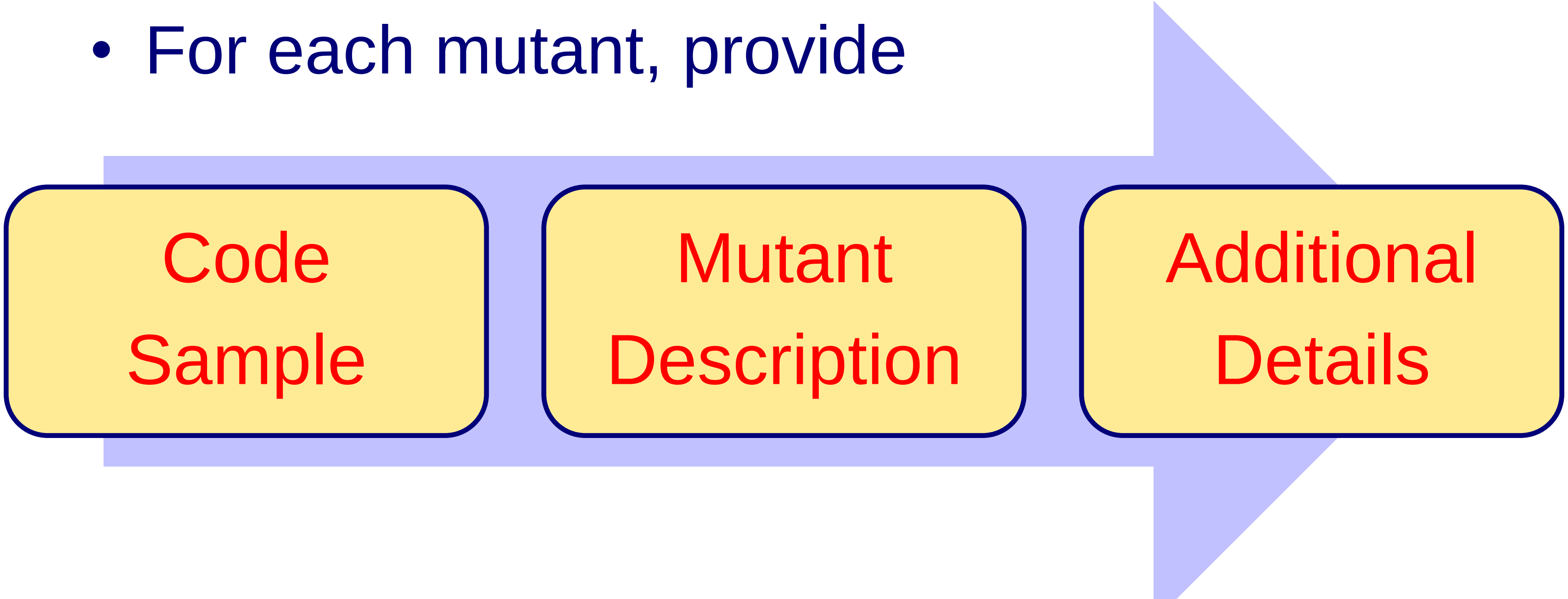
- Goal Evaluate the additional benefits of our technique that are difficult to quantify

Study 2—Qualitative Case Study

- Goal Evaluate the additional benefits of our technique that are difficult to quantify
- Method
  - Assume developer has found suspicious code
  - Select one mutant for each subject

Study 2—Qualitative Case Study

- Goal Evaluate the additional benefits of our technique that are difficult to quantify
- Method
  - Assume developer has found suspicious code
  - Select one mutant for each subject
  - For each mutant, provide



Code Sample

Mutant Description

Additional Details

Study 2 – JWhoisServer

```
private final synchronized ResultSet
          execPST(PreparedStatement pst)
          throws SQLException {
    ResultSet res = pst.executeQuery();
    return res;
}
```

Fault Localization Challenge

Database interaction point does not contain the faulty SQL command

Study 2 – JWhoisServer

```
protected final String getWherePart() {
    Vector<String> qv = this.getQfield();
    final String qf = this.getQfield().get(0);
    StringBuilder ret = new StringBuilder(
        "WHERE "+qf+" <= ? "
         +"AND inetnumend >= ? "
         +"AND "+this.bytelengthField+" = ? ");
     if (this.getWhereaddition().length() > 0) {
         if(!this.getWhereaddition().startsWith(" ")) {
              ret.append(" ");
         }
         ret.append(this.getWhereaddition());
     }
     ret.append("ORDER BY "+qf+" ASC, inetnumend ASC");
     return ret.toString();
}
```

Study 2 – JWhoisServer

```
protected final String getWherePart() {
    Vector<String> qv = this.getQfield();
    final String qf = this.getQfield().get(0);
    StringBuilder ret = new StringBuilder(
        "WHERE "+qf+" <= ? "
         +"AND inetnumend >= ? "
         +"AND "+this.bytelengthField+" = ? ");
     if (this.getWhereaddition().length() > 0) {
         if(!this.getWhereaddition().startsWith(" ")) {
              ret.append("
         }
         ret.append(thi
     }
     ret.append("OR
     return ret.toStrin
}
```

Fault Localization Challenge

JWhoisServer constructs the SQL command in a dynamic fashion

Study 2 – JWhoisServer

External Configuration File

```
db.inetnum.table=inetnum
db.inetnum.objectlookup=inetnum;inet
db.inetnum.qfield=inetnumstart
db.inetnum.key=descr
db.inetnum.bytelength=bytelength
db.inetnum.display=netname AS network;
bytelength;inetnumstart;inetnumend;descr;source
db.inetnum.recurse.person=admin_c;tech_c
```

Study 2 – JWhoisServer

External Configuration File

```
db.inetnum.table=inetnum
db.inetnum.objectlookup=inetnum;inet
db.inetnum.qfield=inetnumstart
db.inetnum.key=descr
db.inetnum.bytelength=bytelength
db.inetnum.display=netname AS network;
bytelength;inetnumstart;inetnumend;descr;source
db.inetnum.recurse.person=admin_c;tech_c
```

Suspicious Database Interaction Point

Statement: dbpool.java:631

SQL Command: select descr, netname as network, bytelength,
inetnumstart, inetnumend, source from inetnum
where inetnumstart <= ? and inetnumend >= ?
and bytelength = ?
order by inetnumstart asc, inetnumend asc

Suspiciousness: 0.91

Study 2 – JWhoisServer

Additional Information

The SQL command connected to a specific test case and its pass/fail status

External Configuration File

```
db.inetnum.table=inetnum
db.inetnum.objectlookup=inetnum;inet
db.inetnum.qfield=inetnumstart
db.inetnum.key=descr
db.inetnum.bytelength=bytelength
db.inetnum.display=netname AS network;
bytelength;inetnumstart;inetnumend;descr;source
db.inetnum.recurse.person=admin_c;tech_c
```

Suspicious Database Interaction Point

Statement: dbpool.java:631

SQL Command: select descr, netname as network, bytelength,
inetnumstart, inetnumend, source from inetnum
where inetnumstart <= ? and inetnumend >= ?
and bytelength = ?
order by inetnumstart asc, inetnumend asc

Suspiciousness: 0.91

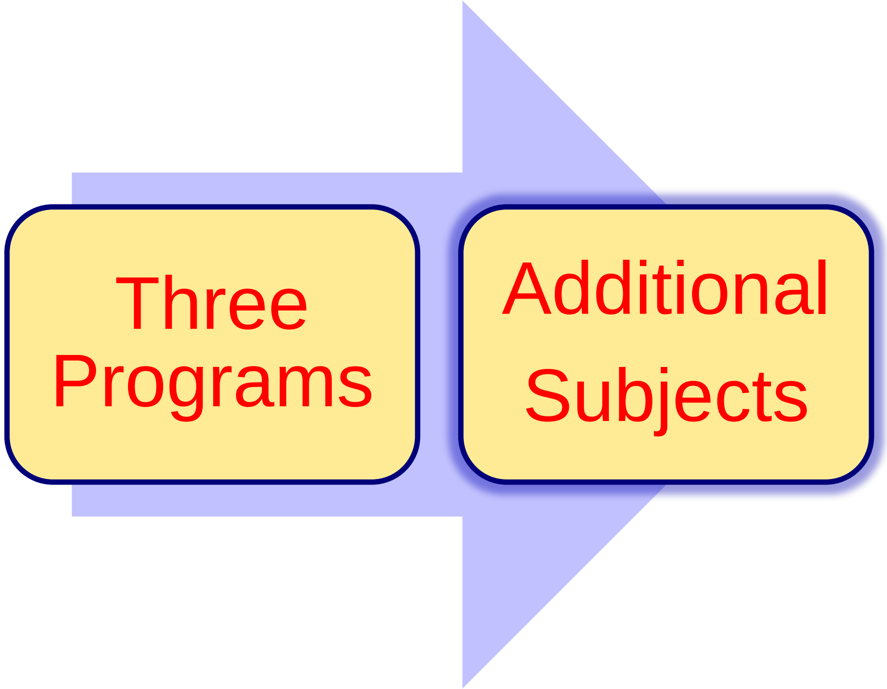
Study 2 – JWhoisServer

- **Standard method *does not***
  - **Identify the faulty database interaction point as highly suspicious**
  - **Extract the complete SQL command**
- **Database-aware technique provides a precise ranking *and* the full SQL command, thereby eliminating manual developer effort**

Outline for the Rest of the Presentation

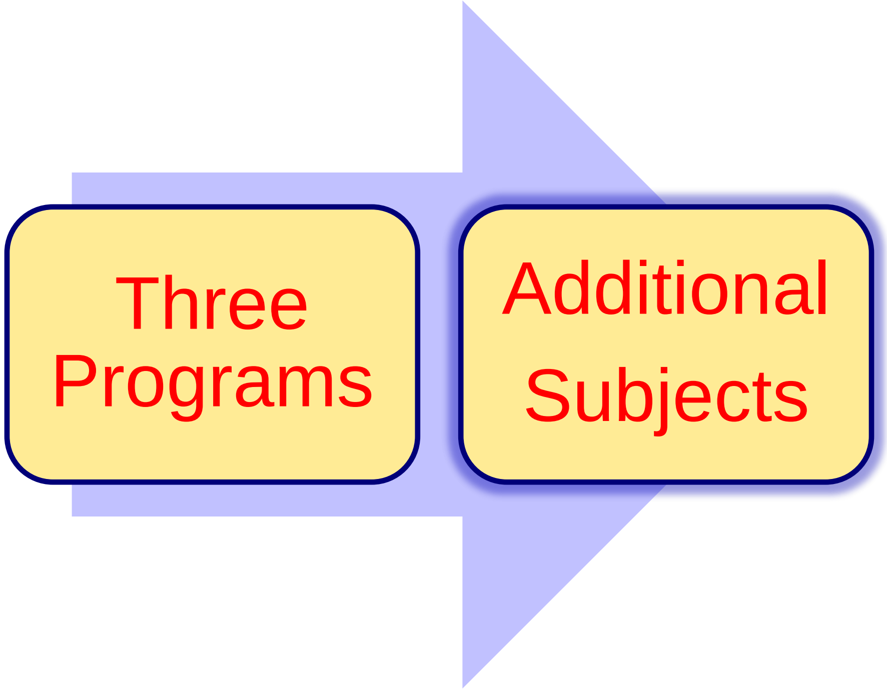
- Our Technique
  - Definitions
  - Algorithm
- Empirical Studies
- Conclusion

Future Work



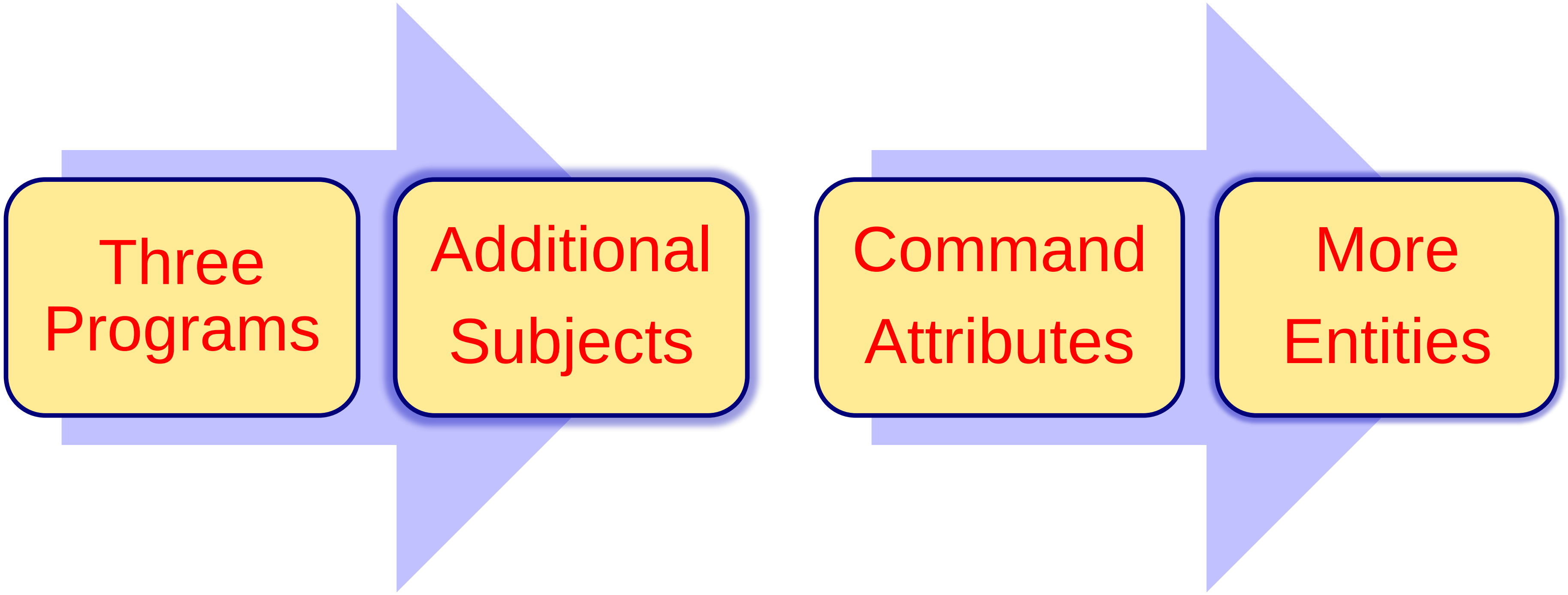
Used three subject programs – (1) from previous research, (2) open source, and (3) industrial

Future Work



Future Work: Incorporate other suitable subjects

Future Work



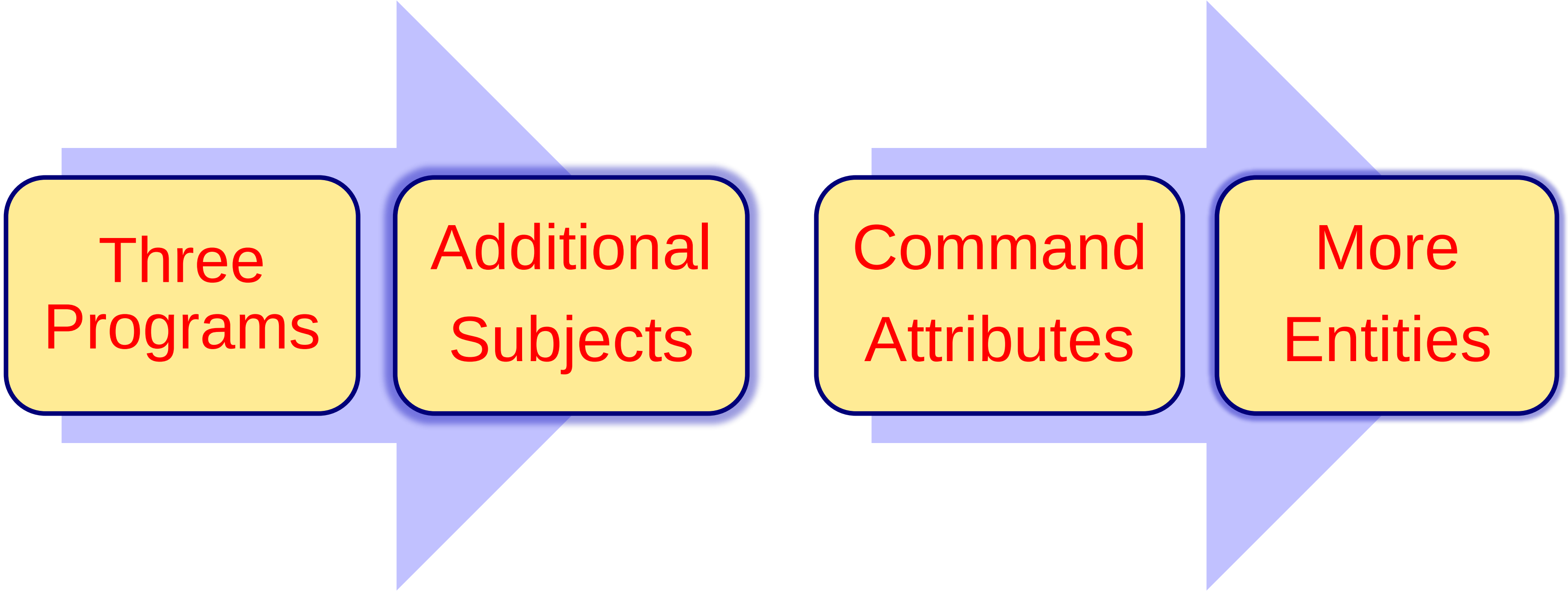
Three Programs

Additional Subjects

Command Attributes

More Entities

Future Work


Three Programs

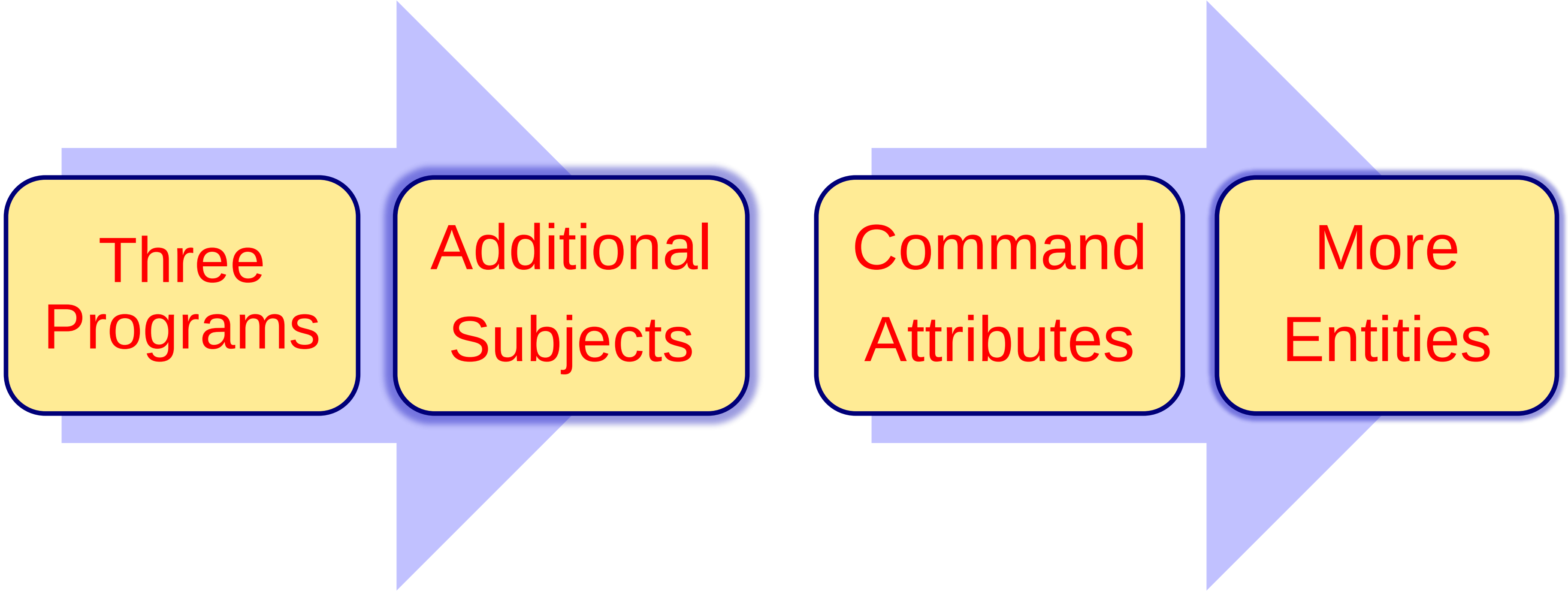
Additional Subjects

Command Attributes

More Entities

Focused on entities involving an SQL command and the attributes found in the relational database

Future Work



Three Programs

Additional Subjects

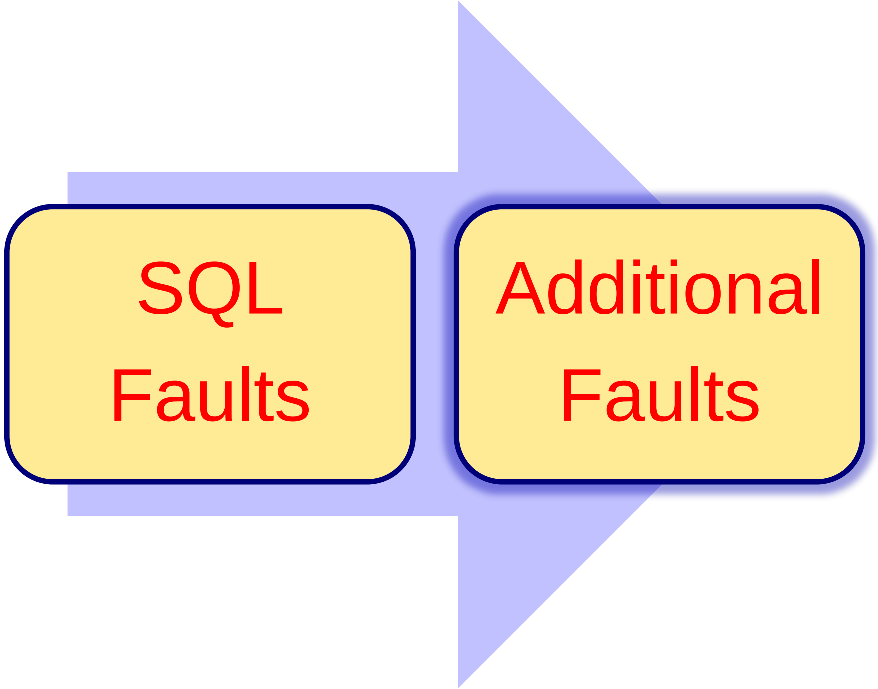
Command Attributes

More Entities

Future Work: WHERE and GROUP BY clauses

Future Work

Future Work

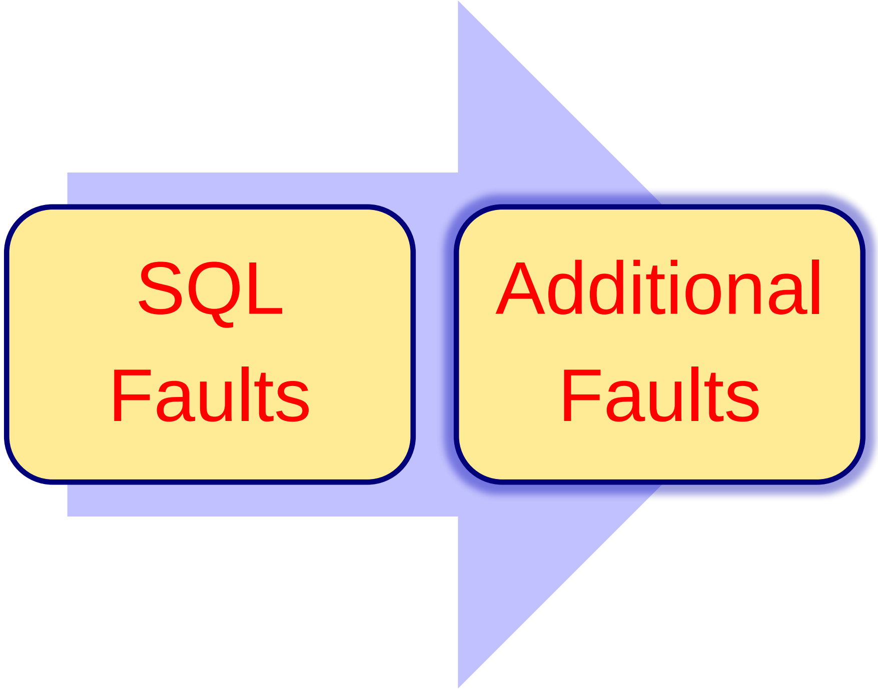


SQL Faults

Additional Faults

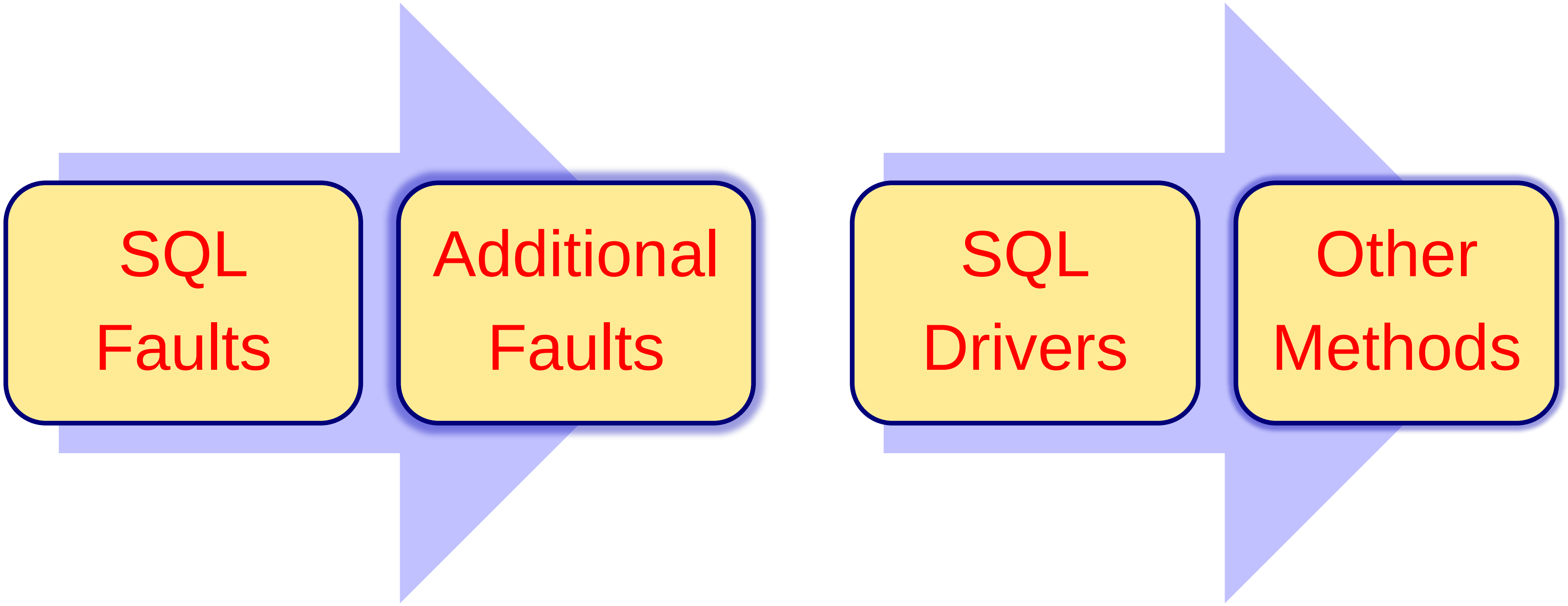
Localizing SQL faults that involve mistakes in querying and modifying the database

Future Work



Future Work: Consider data and schema faults

Future Work



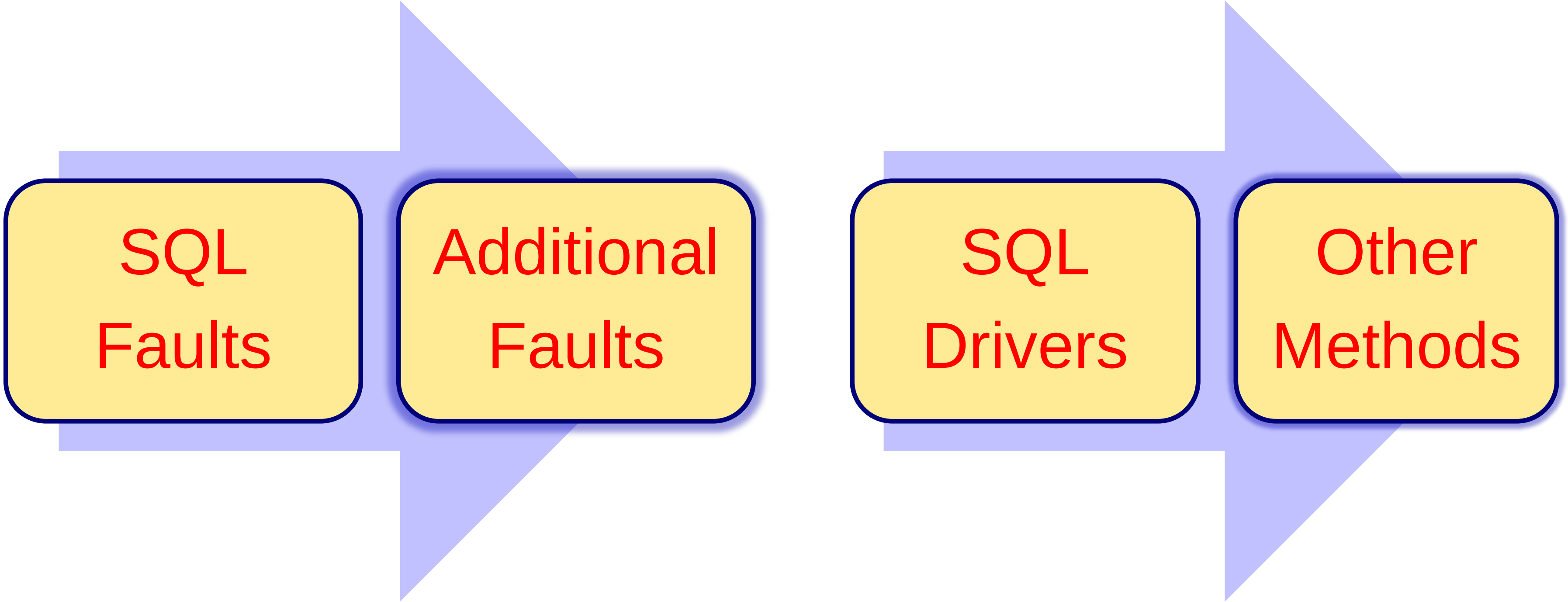
SQL Faults

Additional Faults

SQL Drivers

Other Methods

Future Work


SQL Faults

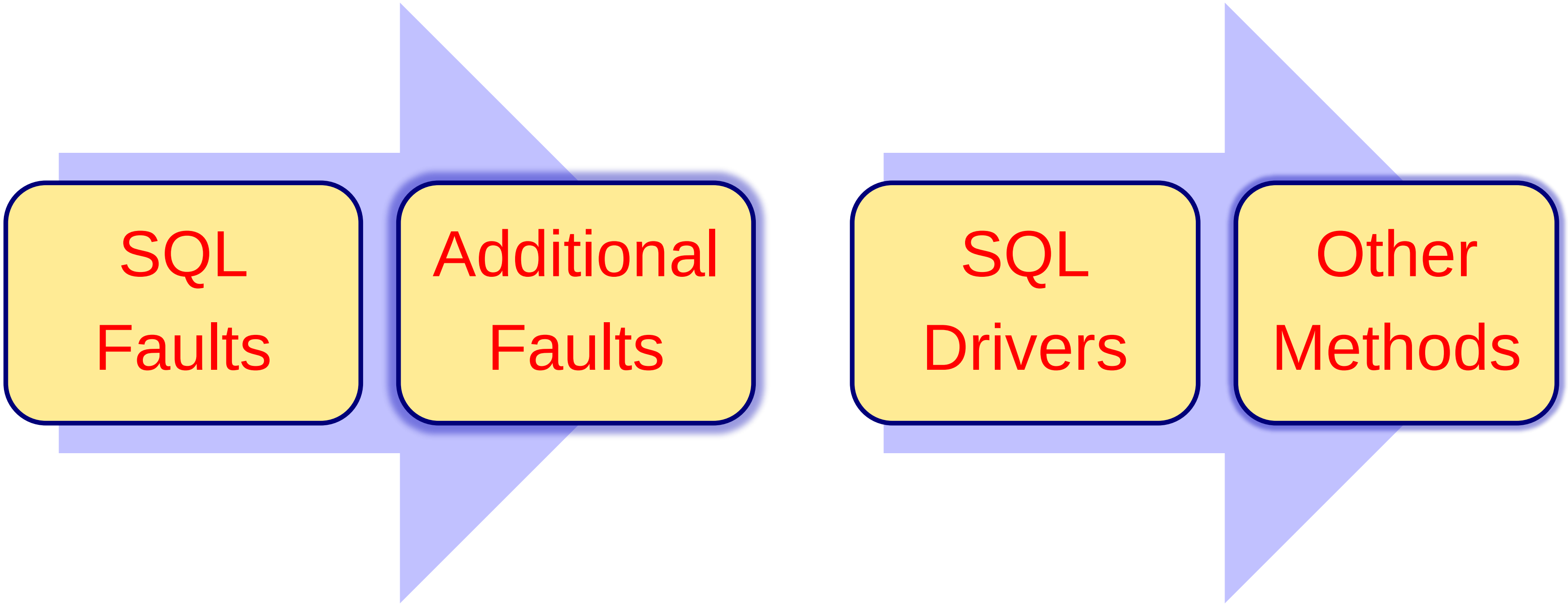
Additional Faults

SQL Drivers

Other Methods

Considered SQL commands that are encoded as strings and submitted through a database driver

Future Work



SQL Faults

Additional Faults

SQL Drivers

Other Methods

Future Work: Localize faults in stored procedures

Summary of Contributions

Key Motivators

- **Databases are an essential component of many software applications**
- **Real-world industrial faults result from incorrect interaction with a database**

Summary of Contributions

- Database-aware fault localization method that uses database-related information
- Prototype database-aware fault localization system that provides a ranking as well as the executed SQL commands

Summary of Contributions

- Database-aware fault localization method that uses database-related information
- Prototype database-aware fault localization system that provides a ranking as well as the executed SQL commands
- Empirical studies revealing that:
    - Statement-based methods work well for database applications with static interactions
    - Database-aware approach markedly improves fault localization for dynamic applications

Summary of Contributions

In summary, this paper

- **Shows the need for database-aware fault-localization methods**
- **Describes the first approach that calculates suspiciousness for program and database entities**

Summary of Contributions

In summary, this paper

- **Shows the need for database-aware fault-localization methods**
- **Describes the first approach that calculates suspiciousness for program and database entities**

The experimental study

- **Quantitatively and qualitatively evaluates the presented technique**
- **Shows improvements in the effectiveness of finding SQL faults by as much as 95% over existing methods**

Localizing SQL Faults in Database Applications

Gregory M. Kapfhammer[†]

Sarah R. Clark[*], Jake Cobb[*],

James A. Jones[‡], and Mary Jean Harrold[*]

[*]Georgia Institute of Technology

[†]Allegheny College

[‡]University of California, Irvine

Supported by NSF CCF-1116943 and Google Faculty Research Award to UC Irvine, NSF CCF-0725202, CCF-0541048, IBM Software Quality Innovation Award, and InComm to Georgia Tech, and by SIGSOFT CAPS