Finding Application Errors and Security Flaws Using PQL: A Program Query Language

Michael Martin, Benjamin Livshits,
Monica S. Lam
Stanford University

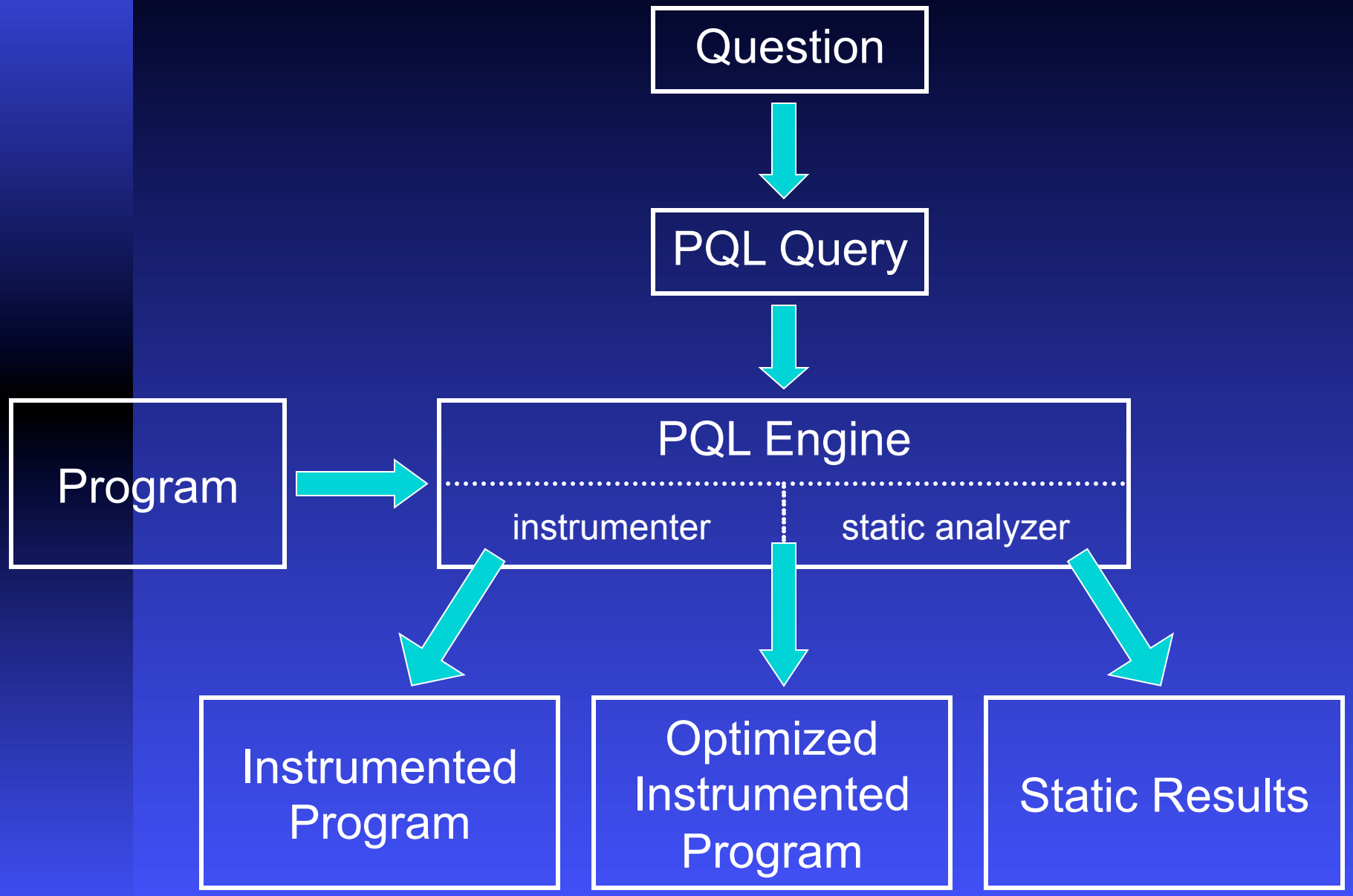
OOPSLA 2005

Modified for CMSC498L, Fall'12

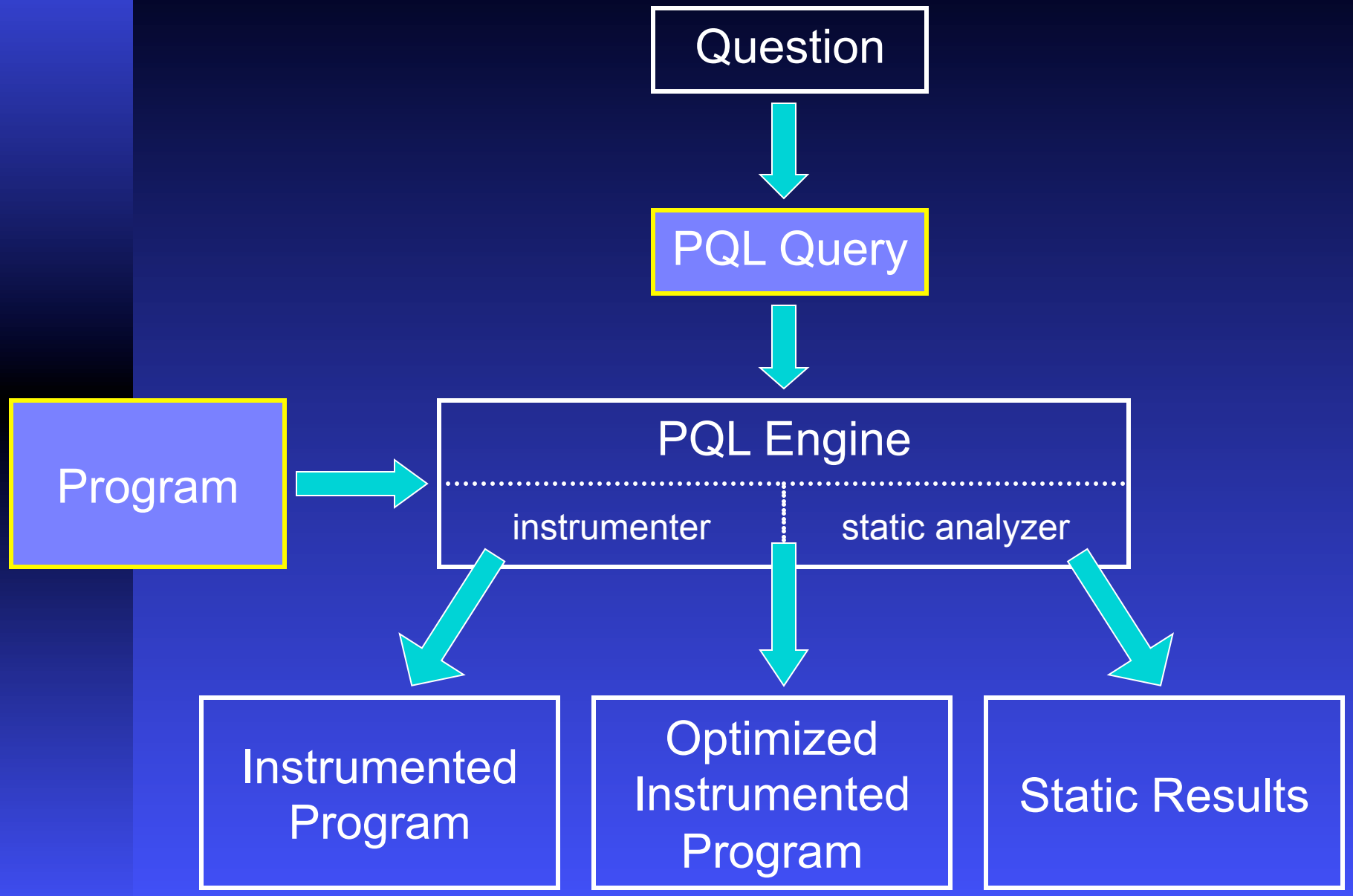
Program Query Language

- Queries operate on *program traces*
  - Sequence of events representing a run
  - Refers to object *instances*, not variables
  - Events matched may be widely spaced
- Patterns resemble Java code
  - Like a small matching code snippet
  - No references to compiler internals

System Architecture



System Architecture



Running Example: SQL Injection

- Unvalidated user input passed to a database back-end for execution
- If SQL in embedded in the input, attacker can take over database
  - Unauthorized data reads
  - Unauthorized data modifications
- One of the top web security flaws

SQL Injection 1

```
HttpServletRequest req = /* ... */;
java.sql.Connection conn = /* ... */;
String q = req.getParameter("QUERY");
conn.execute(q);
```

```
1   CALL   o1.getParameter(o2)
2   RET    o2
3   CALL   o3.execute(o2)
4   RET    o4
```

SQL Injection 2

```
String read() {
  HttpServletRequest req = /* ... */;
    return req.getParameter("QUERY");
}
/* ... */
java.sql.Connection conn = /* ... */;
conn.execute(read());
```

```
1   CALL    read()
2   CALL    o1.getParameter(o2)
3   RET     o3
4   RET     o3
5   CALL    o4.execute(o3)
6   RET     o5
```

Essence of Pattern the Same

```
1 CALL o1.getParameter(o2)
2 RET  o3

3 CALL o4.execute(o3)
4 RET  o5
```

```
1 CALL read()
2 CALL o1.getParameter(o2)
3 RET  o3
4 RET  o3
5 CALL o4.execute(o3)
6 RET  o5
```

The object *returned by* **`getParameter`** is then *argument 1 to* **`execute`**

Translates Directly to PQL

```
query main()
uses String x;
matches {
    x = HttpServletRequest.getParameter(_);

    Connection.execute(x);
}
```

- Query variables → heap objects
- Instructions need not be adjacent in trace

Alternation

```
query main()
uses String x;
matches {
    x = HttpServletRequest.getParameter(_)
  | x = HttpServletRequest.getHeader(_);
    Connection.execute(x);
}
```

SQL Injection 3

```
HttpServletRequest req = /* ... */;
n = getParameter("NAME");
p = getParameter("PASSWORD");
conn.execute(
        "SELECT * FROM logins WHERE name=" +
        n +
        " AND passwd=" +
        p
    );
```

- Compiler translates string concatenation into operations on **`String`** and **`StringBuffer`** objects

SQL Injection 3

1	CALL	o_1.getParameter(o_2)
2	RET	o_3
3	CALL	o_1.getParameter(o_4)
4	RET	o_5
5	CALL	StringBuffer.<init>(o_6)
6	RET	o_7
7	CALL	o_7.append(o_8)
8	RET	o_7
9	CALL	o_7.append(o_3)
10	RET	o_7
11	CALL	o_7.append(o_9)
12	RET	o_7
13	CALL	o_7.append(o_5)
14	RET	o_7
15	CALL	o_7.toString()
16	RET	o_{10}
17	CALL	o_{11}.execute(o_{10})
18	RET	o_{12}

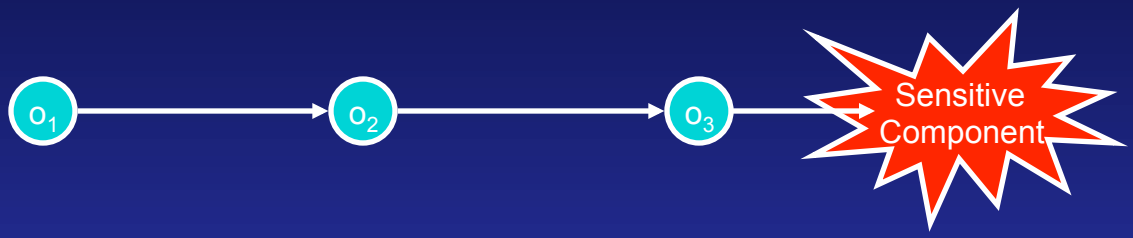
Old Pattern Doesn't Work

```
1    CALL    o1.getParameter(o2)
2    RET     o3
3    CALL    o1.getParameter(o4)
4    RET     o5

17   CALL    o11.execute(o10)
```

- o_{10} is neither o_3 nor o_5, so no match

Instance of Tainted Data Problem



- User-controlled input must be trapped and validated before being passed to database
- Objects ***derived*** from an initial input must also be considered user controlled
- Generalizes to many security problems: cross-site scripting, path traversal, response splitting, format string attacks...

Pattern Must Catch Derived `Strings`

```
1    CALL    o1.getParameter(o2)
2    RET     o3
3    CALL    o1.getParameter(o4)
4    RET     o5

9    CALL    o7.append(o3)
10   RET     o7
11   CALL    o7.append(o9)
12   RET     o7

15   CALL    o7.toString()
16   RET     o10
17   CALL    o11.execute(o10)
```

Pattern Must Catch Derived `Strings`

```
1    CALL    o1.getParameter(o2)
2    RET     o3
3    CALL    o1.getParameter(o4)
4    RET     o5

9    CALL    o7.append(o3)
10   RET     o7
11   CALL    o7.append(o9)
12   RET     o7

15   CALL    o7.toString()
16   RET     o10
17   CALL    o11.execute(o10)
```

Pattern Must Catch Derived `Strings`

```
1    CALL    o1.getParameter(o2)
2    RET     o3
3    CALL    o1.getParameter(o4)
4    RET     o5

9    CALL    o7.append(o3)
10   RET     o7
11   CALL    o7.append(o9)
12   RET     o7

15   CALL    o7.toString()
16   RET     o10
17   CALL    o11.execute(o10)
```

Derived String Query

```
query derived (Object x)
  uses Object temp;
  returns Object d;
matches {
  { temp.append(x); d := derived(temp); }
| { temp = x.toString(); d := derived(temp); }
| { d := x; }
}
```

New Main Query

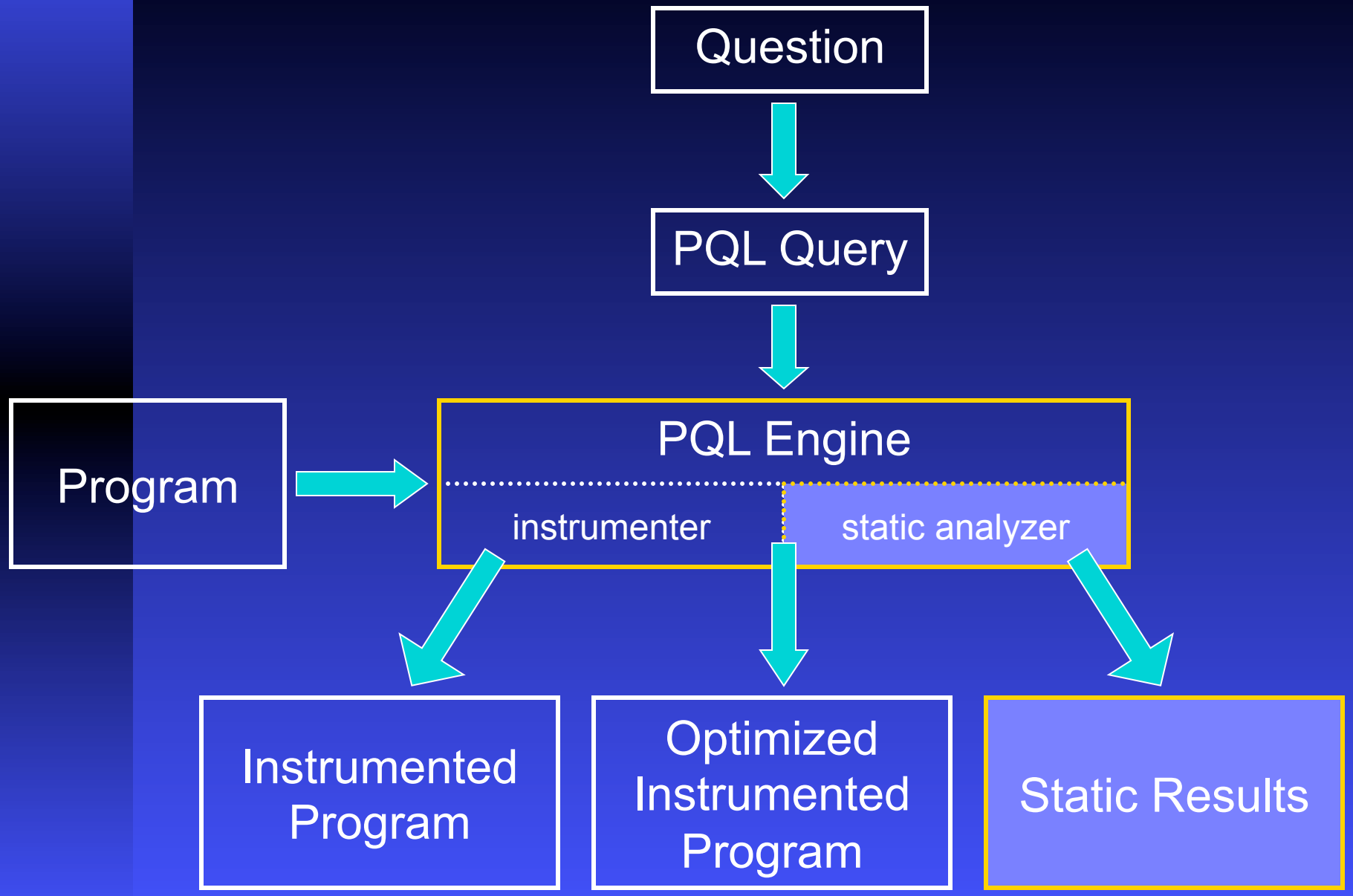
```
query main()
uses String x, final;
matches {
    x = HttpServletRequest.getParameter(_)
  | x = HttpServletRequest.getHeader(_);
    final := derived(x);
    Connection.execute(final);
}
```

Full derived() Query

```
query derived (Object x)
returns Object y;
uses Object temp;
matches {
  y := x
| { temp = StringProp(x);
    y := derived(temp); }
}
```

```
query StringProp (Object * x)
returns Object y;
uses Object z;
matches {
  y.append(x, ...)
| x.getChars(_, _, y, _)
| y.insert(_, x)
| y.replace(_, _, x)
| y = x.substring(...)
| y = new java.lang.String(x)
| y = new java.lang.StringBuffer(x)
| y = x.toString()
| y = x.getBytes(...)
| y = _.copyValueOf(x)
| y = x.concat(_)
| y = _.concat(x)
| y = new java.util.StringTokenizer(x)
| y = x.nextToken()
| y = x.next()
| y = new java.lang.Number(x)
| y = x.trim()
| { z = x.split(...); y = z[]; }
| y = x.toLowerCase(...)
| y = x.toUpperCase(...)
| y = _.replaceAll(_, x)
| y = _.replaceFirst(_, x);
}
```

System Architecture



Static Analysis

- “Can this program match this query?”
- Undecidable in general
- We give a conservative approximation
- No matches found = None possible

Static Analysis

- PQL query automatically translated to query on pointer analysis results
- Pointer analysis is *sound* and *context-sensitive*
  - 10^{14} contexts in a good-sized application
  - Exponential space represented with BDDs
  - Analyses given in Datalog
- Whaley/Lam, PLDI 2004 (*bddbddb*) for details

Static Results

- Sets of objects and events that could represent a match

OR

- Program points that could participate in a match

- No results = no match possible!