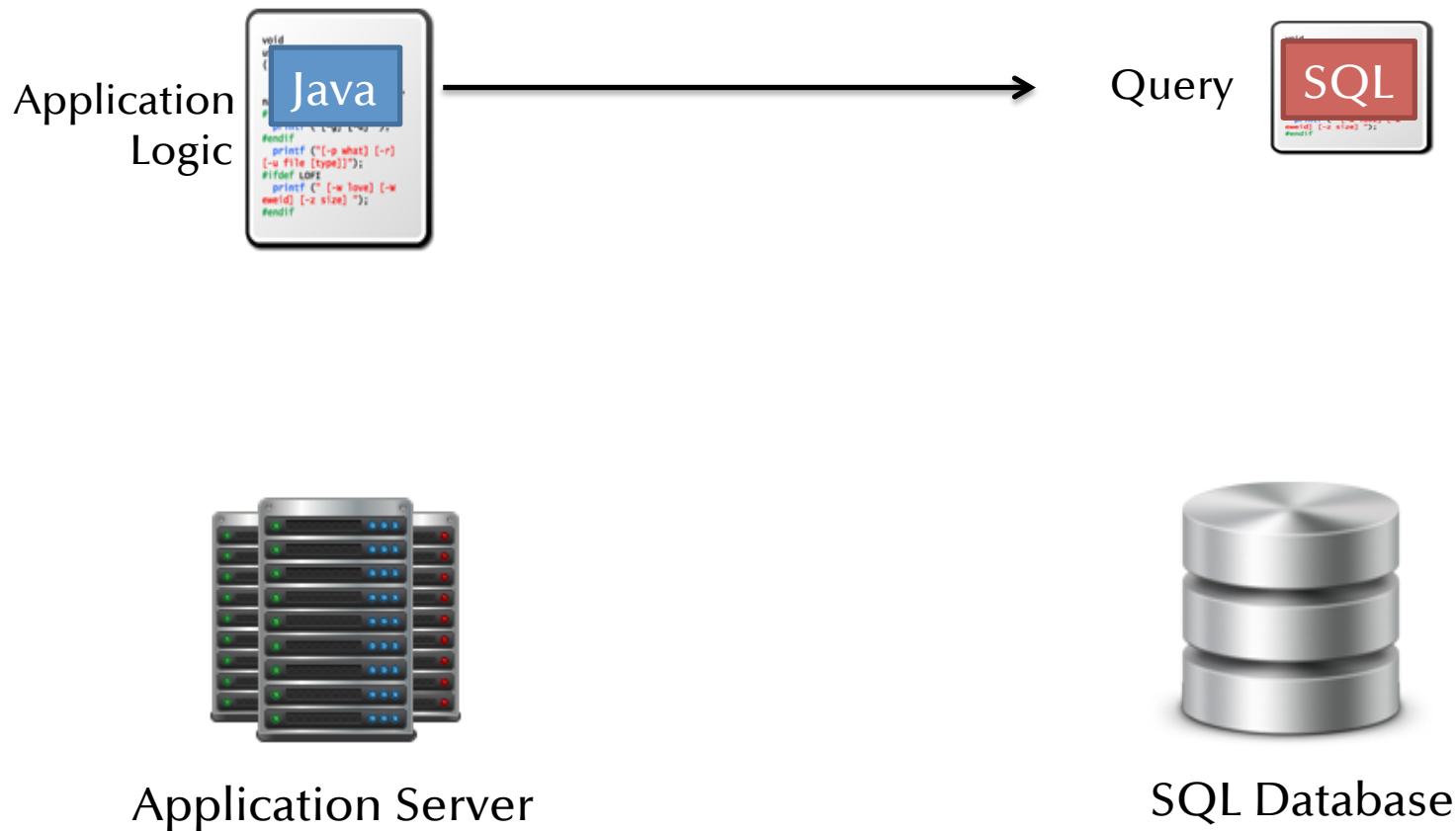


# StatusQuo: Making Familiar Abstractions Perform Using Program Analysis

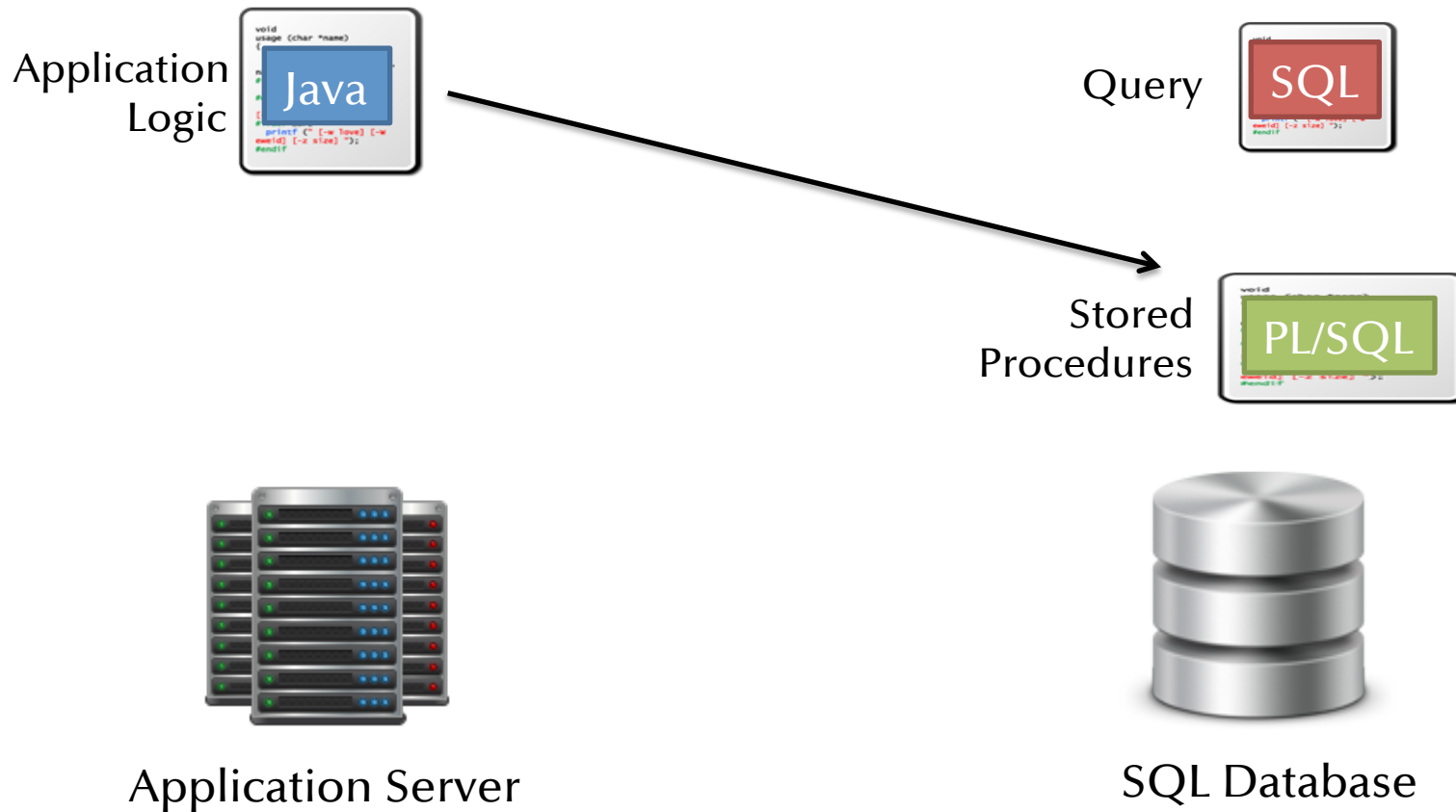
Alvin Cheung  
Samuel Madden  
Armando Solar-Lezama  
MIT

Owen Arden  
Andrew C. Myers  
Cornell

# Developing Database Applications



# Developing Database Applications



# Developing Database Applications

Application  
Logic

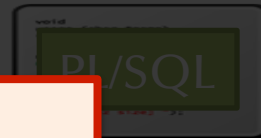


Query



## Language Choice for Application Logic

Stored



Program analysis to the rescue!

## Application Distribution

Application Server

SQL Database

# StatusQuo

- Express application logic in ways that programmers are comfortable with
- Job of compiler & runtime to determine the most efficient implementation

# Two Key Technologies

- Infer queries from imperative code
- Migrate computation between servers for optimal performance

# Relational Operations in Imperative Code

```
List getUsersWithRoles () {  
  List users = getUsersFromDB();  
  List roles = getRolesFromDB();  
  List results = new ArrayList();  
  for (User u : users) {  
    for (Role r : roles) {  
      if (u.roleId == r.id)  
        results.add(u);  
    }  
  }  
  return results;  
}
```

SELECT \* FROM user

SELECT \* FROM role



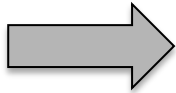
convert to

```
List getUsersWithRoles () {  
  return executeQuery(  
    "SELECT u FROM users u, roles r  
    WHERE u.roleId == r.id  
    ORDER BY u.roleId, r.id";  
  );  
}
```

# Relational Operations in Imperative Code

```
List getUsersWithRoles () {  
  List users = getUsersFromDB();  
  List roles = getRolesFromDB();  
  List results = new ArrayList();  
  for (User u : users) {  
    for (Role r : roles) {  
      if (u.roleId == r.id)  
        results.add(u);  
    }  
  }  
  return results; ← post-condition variable
```

Goal  
Find a variable that  
we can rewrite into a  
SQL expression

  
convert to

```
List getUsersWithRoles () {  
  return executeQuery(  
    "SELECT u FROM users u, roles r  
    WHERE u.roleId == r.id  
    ORDER BY u.roleId, r.id";  
  );  
}
```



# Query By Synthesis (QBS)

- Identify potential code fragments
  - i.e., regions of code that fetches persistent data and return values
- Find SQL expressions for post-condition variables
- Try to prove that those expressions preserve program semantics
  - if so, convert the code!

# Initial Code Fragments Identification

- Find program points that retrieve persistent data
- Run an inter-procedural analysis that:
  - determine where persistent data are used
  - delimit code fragment to analyze

# Search for Post-Condition Expressions

```
List getUsersWithRoles () {  
  List users = query(select * from users);  
  List roles = query(select * from roles);  
  List results = [];  
  for (User u : users) {  
    for (Role r : roles) {  
      if (u.roleId == r.id)  
        results = results : [] }}  
  return results; }
```

Relations involved:

users, roles

**Infinite search space size!**

Possible expressions to consider for results:

$\sigma_f(\text{users})$        $\text{top}_f(\text{users})$        $\pi_f(\text{users} \bowtie_g \text{roles})$   
 $\pi_f(\sigma_g(\text{users}) \bowtie_h \text{roles})$       other expressions involving users, roles

# Constraints for Post-Condition Expressions

```
List getUsersWithRoles () {  
  List users = query(select * from users);  
  List roles = query(select * from roles);  
  List results = [];  
  for (User u : users) {  $\longrightarrow$  results =  $\pi_{\text{user}}(\text{users}[0..i] \bowtie_{\text{roleId}=\text{id}} \text{roles})$   
    for (Role r : roles) { outer loop invariant  
      if (u.roleId == r.id)  
        results = results : [] } results =  $\pi_{\text{user}}(\text{users} \bowtie_{\text{roleId}=\text{id}} \text{roles})$   
  } post-condition expression  
  return results; }  $\longrightarrow$ 
```

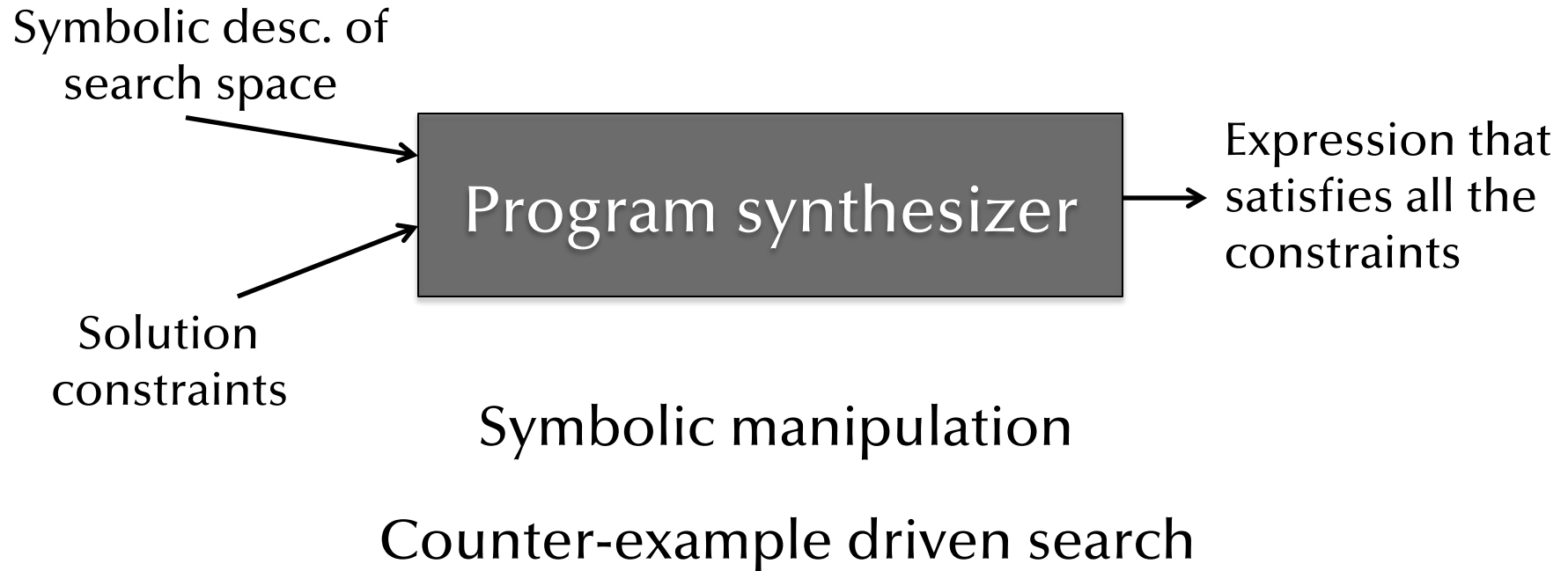
If **outer loop invariant** is true and **outer loop terminates**  
then **post-condition expression** is true

Hoare-style program verification

Still need a smarter  
way to search

# Search for Post-Condition Expressions and Invariants

- Use program synthesis as search engine



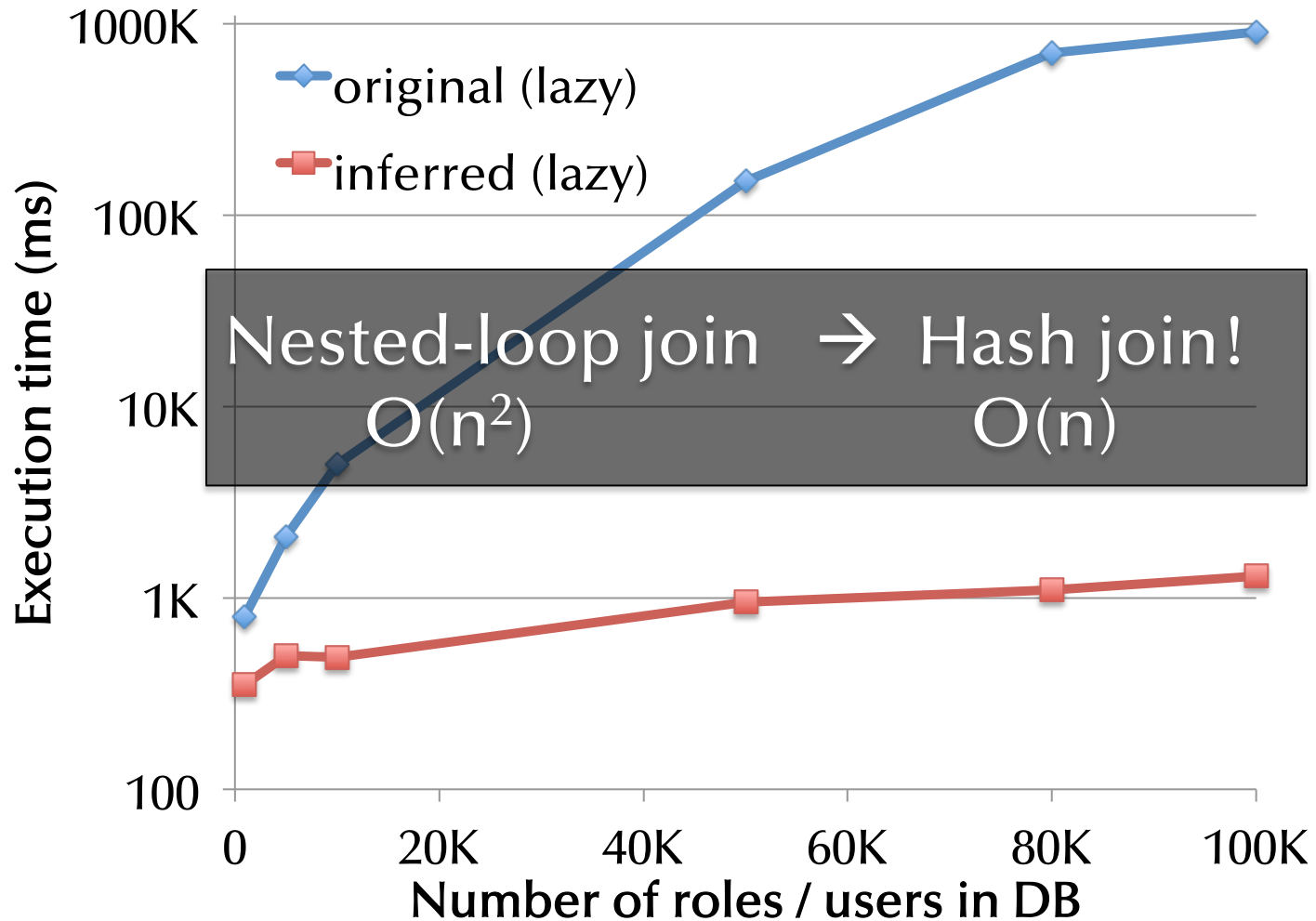
# Experiments

# Real-world Evaluation

Wilos (project management application) – 62k LOC

Operation type	# Fragments found	# Fragments converted
Projection	1	1
Selection	13	10
Join	7	7
Aggregation	11	10
<b>Total</b>	<b>33</b>	<b>28</b>

# Performance Evaluation: Join Query





# Developing Database Applications

Application  
Logic



Query



Stored  
Procedures



Application Server



SQL Database

## Application Distribution

# Running Example

```
discount = executeQuery("select discount from customers
                        where id = " + cid);

totalAmount = orderTotal * (1 - discount);

credit = executeQuery("select credit from customers
                     where id = " + cid);

if (credit < totalAmount)
    printToConsole("Only " + credit + " in account!");
else
    executeUpdate("update customer set credit = " +
                 (credit - totalAmount) + " where id = " + cid);
```

# Actual Execution

DB

```
discount = executeQuery("select discount from customers  
                        where id = " + cid);
```

APP

```
totalAmount = orderTotal * (1 - discount);
```

DB

```
credit = executeQuery("select credit from customers  
                     where id = " + cid);
```

APP

```
if (credit < totalAmount)  
    printToConsole("Only " + credit + " in account!");  
else
```

DB

```
executeUpdate("update customer set credit = " +  
             (credit - totalAmount) + " where id = " + cid);
```

# Actual Execution

DB

```
discount = executeQuery("select discount from customers  
                        where id = " + cid);
```



network communication

APP

```
totalAmount = orderTotal * (1 - discount);
```



network communication

DB

```
credit = executeQuery("select credit from customers  
                     where id = " + cid);
```



network communication

APP

```
if (credit < totalAmount)  
    printToConsole("Only " + credit + " in account!");
```

```
else
```



network communication

DB

```
executeUpdate("update customer set credit = " +  
             (credit - totalAmount) + " where id = " + cid);
```

# Speeding up Execution

```
discount = executeQuery("select discount from customers  
                        where id = " + cid);
```

```
totalAmount = orderTotal * (1 - discount);
```

DB

```
credit = executeQuery("select credit from customers  
                     where id = " + cid);
```

```
if (credit < totalAmount)
```

APP

```
    printToConsole("Only " + credit + " in account!");
```

```
else
```

DB

```
    executeUpdate("update customer set credit = " +  
                 (credit - totalAmount) + " where id = " + cid);
```

# Speeding up Execution

```
discount = executeQuery("select discount from customers  
                        where id = " + cid);
```

```
totalAmount = orderTotal * (1 - discount);
```

data dependency

```
credit = executeQuery("select credit from customers  
                    where id = " + cid);
```

control dependency

```
if (credit < totalAmount)
```

```
    printToConsole("Only " + credit + " in account!");
```

```
else
```

```
    executeUpdate("update customer set credit = " +  
                (credit - totalAmount) + " where id = " + cid);
```

DB

APP

DB

# Speeding up Execution

```
discount = executeQuery("select discount from customers  
                        where id = " + cid);
```

```
totalAmount = orderTotal * (1 - discount);
```

data dependency



DB Server

DB

```
credit = executeQuery("select credit from customers  
                     where id = " + cid);
```

control dependency

```
if (credit < totalAmount)
```

APP

```
    printToConsole("Only " + credit + " in account!");
```

```
else
```

DB

```
    executeUpdate("update customer set credit = " +  
                  (credit - totalAmount) + " where id = " + cid);
```

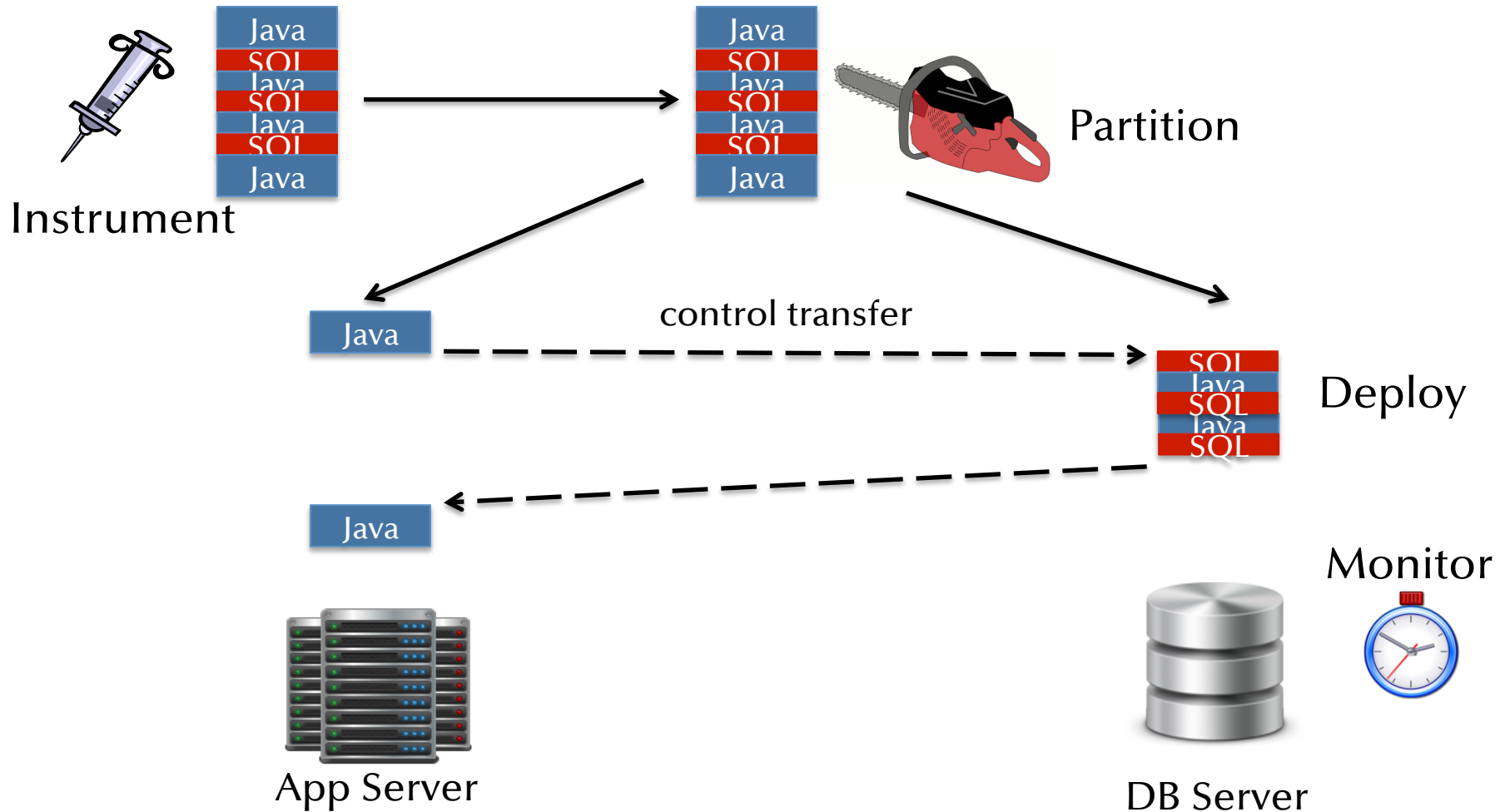
# Introducing Pyxis

- “Store-procedurizes” DB apps and pushes computation to the DB
- Adaptively controls the amount of computation pushed to DB for optimal performance
- No programmer intervention required

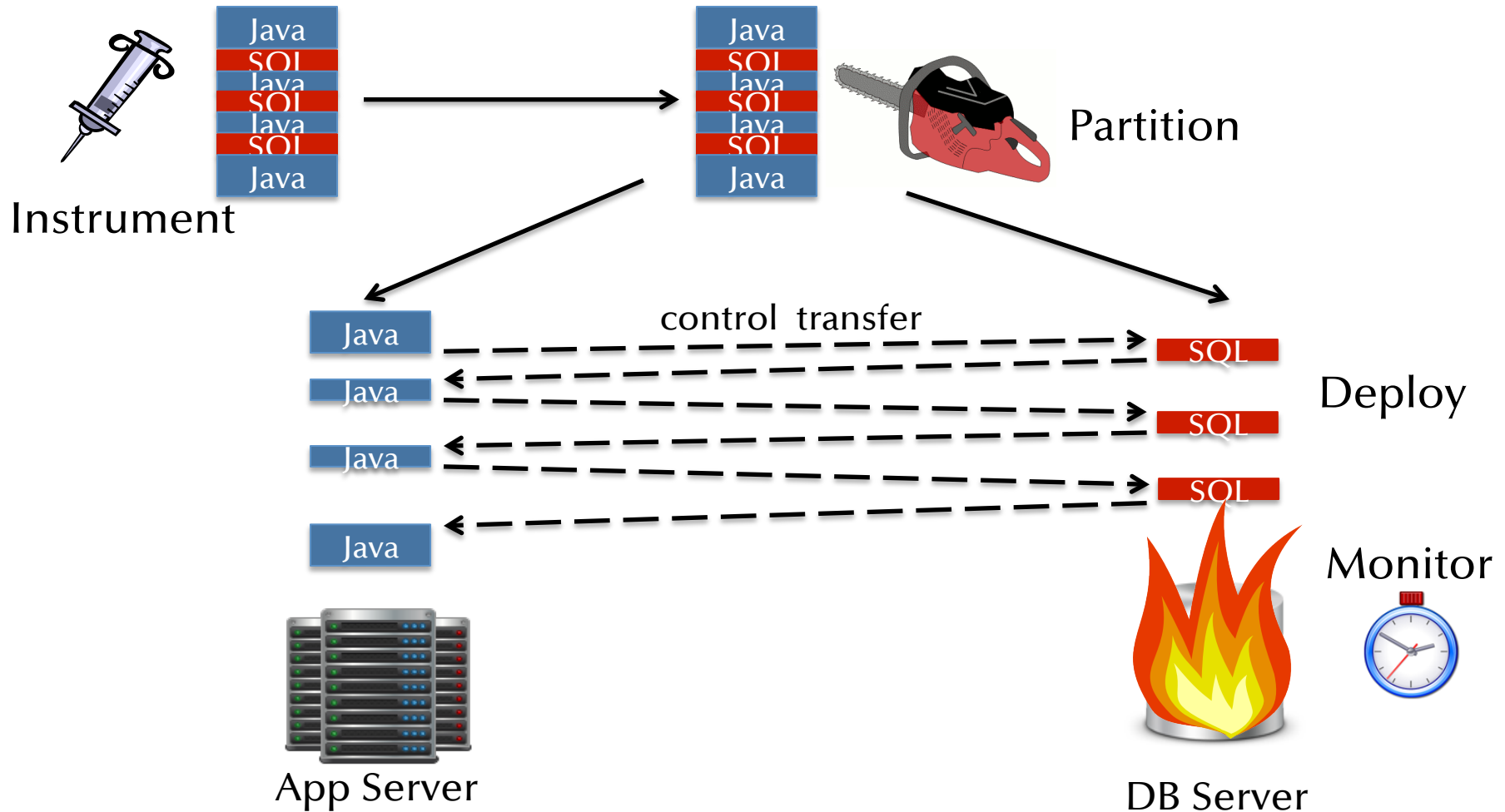


# Using Pyxis

# How Pyxis Works



# How Pyxis Works



# Generating Program Partitions

- Deploy and profile application as-is
- Construct a dependence graph of program statements
  - captures both control and data flow
- Formulate linear program from profile data and dependence graph
  - solution gives a partitioning of the source code

# Executing Partitioned Programs

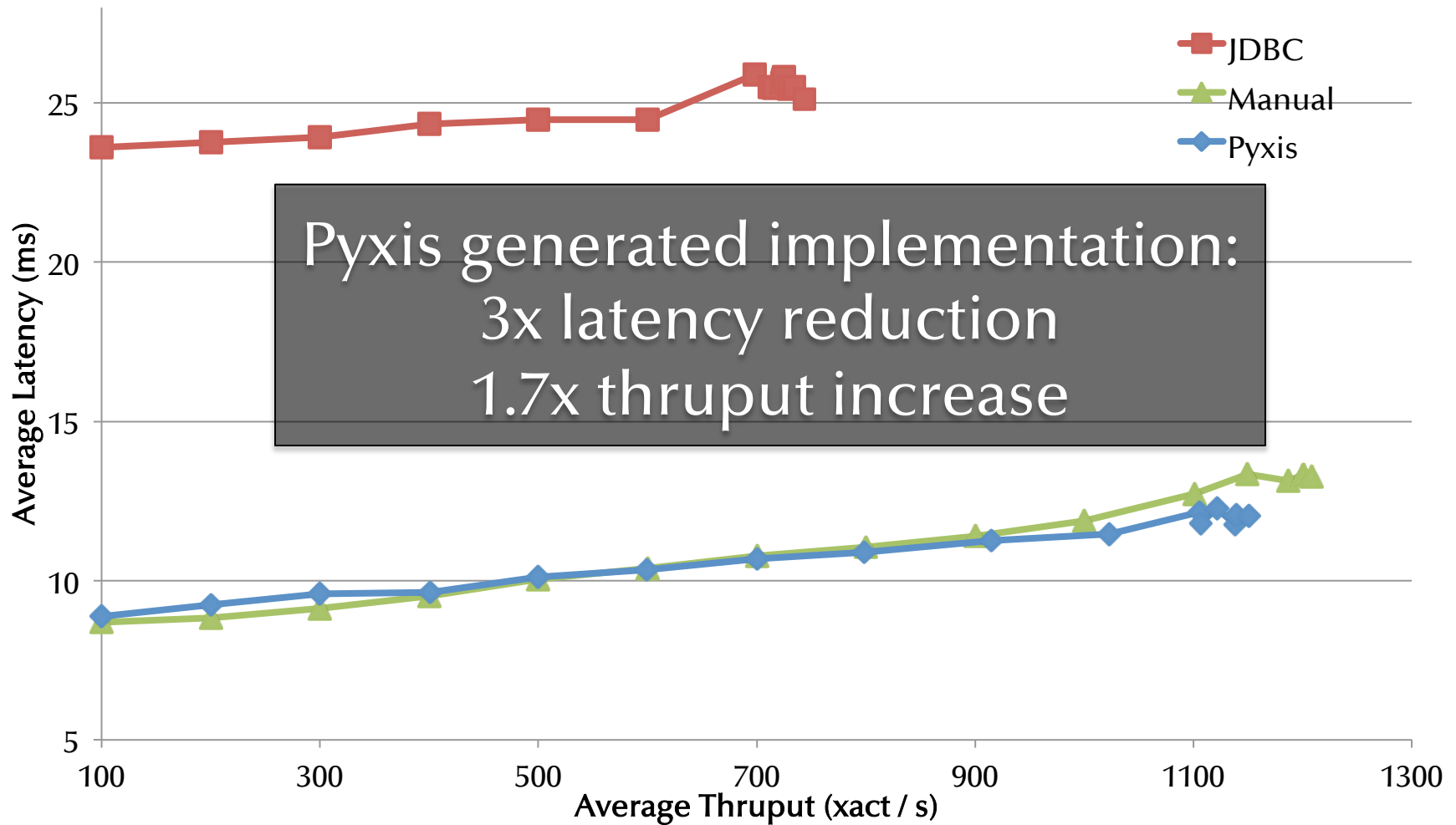
- Pyxis compiler translates partitioned code into standard Java code
- Pyxis runtime executes compiled Java code
  - runtime is just another Java program running on a standard JVM
  - includes monitoring component to determine partition switching

# Experiments

# Experiment Setup

- TPC-C Java implementation
  - 20 terminals issuing new order transactions
  - DB server has 16 cores total
  - Compared against two implementations:
    - JDBC: everything on app server except for JDBC stmts
    - Manual: custom “store procedurized” implementation where everything is on the DB server

# All Cores Available





# StatusQuo

Ease DB application development

Convert imperative program statements  
into declarative SQL

Fully automatic code partitioning using  
application and server characteristics

`db.csail.mit.edu/statusquo`