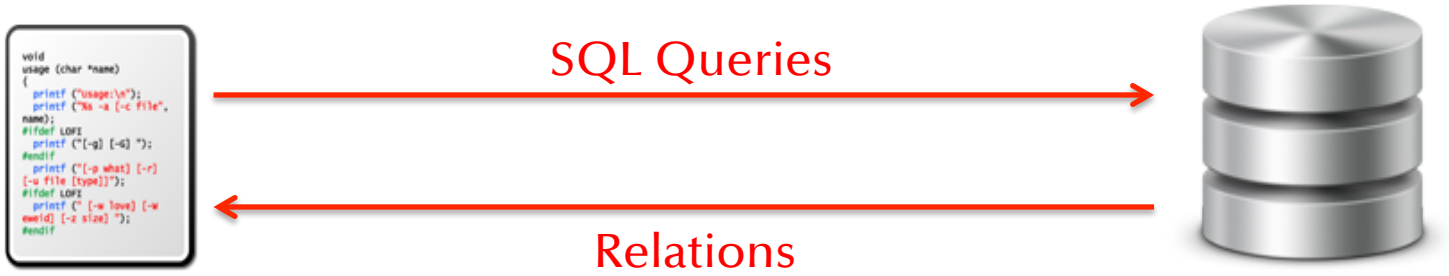
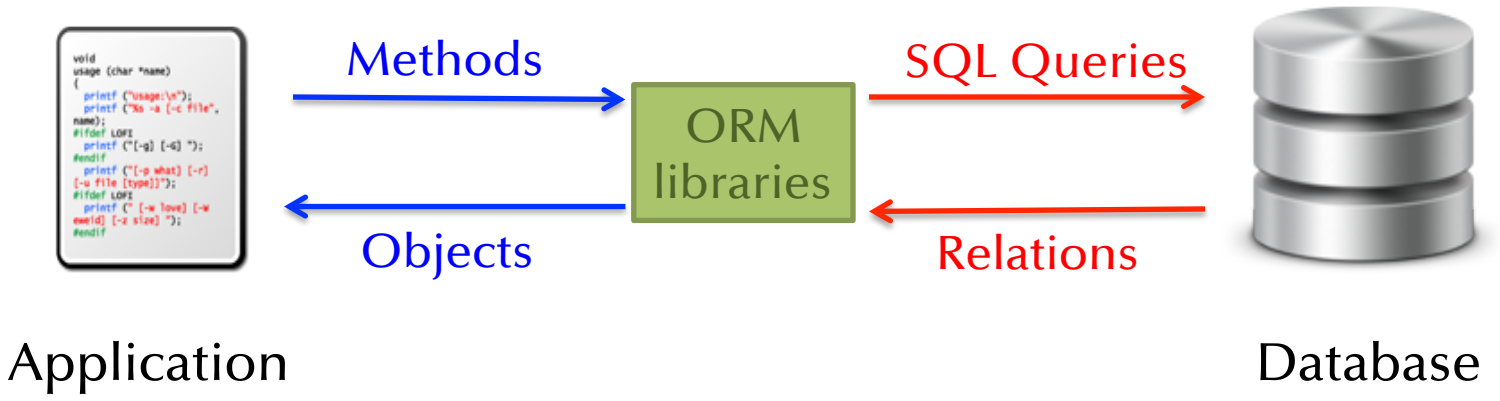
Optimizing Database-Backed Applications Using Query Synthesis

Alvin Cheung
Armando Solar-Lezama Samuel Madden
MIT

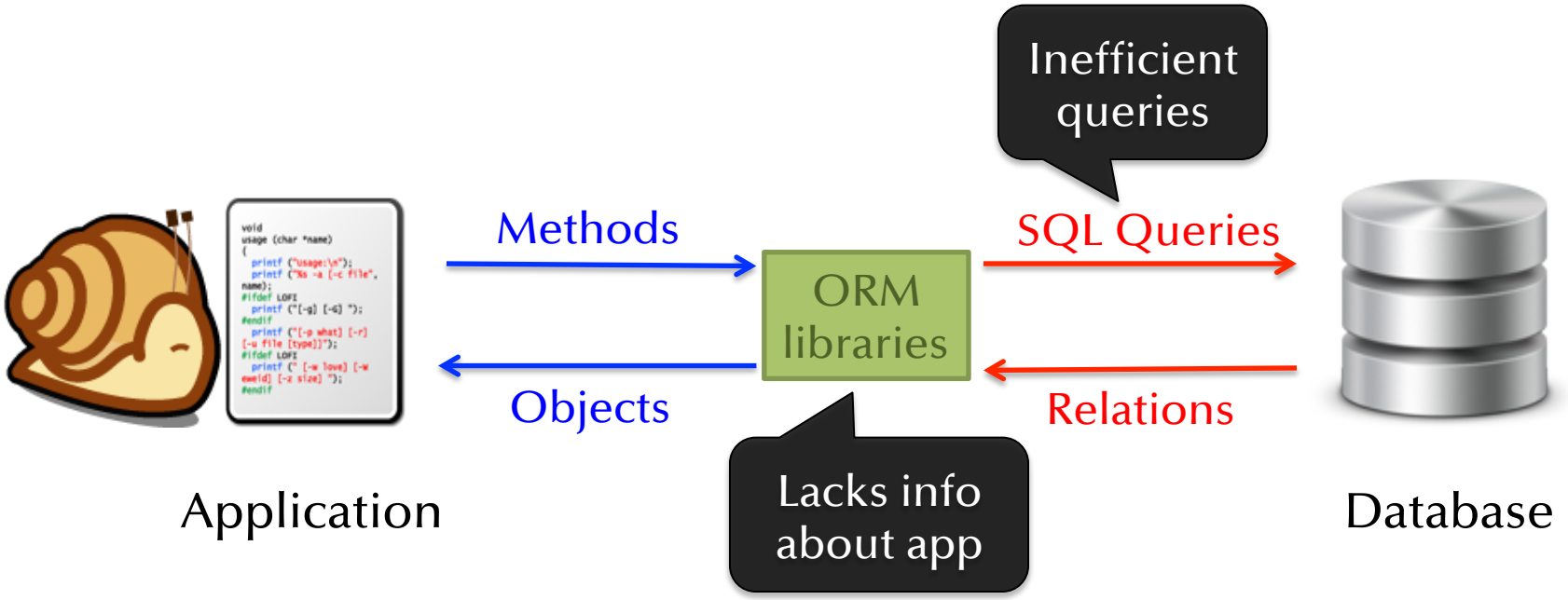
Developing Database Applications



Developing Database Applications



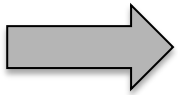
Developing Database Applications



Relational Operations in Imperative Code

```
List getUsersWithRoles () {
  List users = User.getAllUsers();
  List roles = Role.getAllRoles();
  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

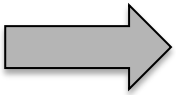
```
List getUsersWithRoles () {
  return executeQuery(
   "SELECT   u FROM user u, role r
    WHERE    u.roleId == r.id
    ORDER BY u.roleId, r.id"; }
```

Relational Operations in Imperative Code

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

output variable

Goal
Find a post-condition that we can rewrite into a SQL expression



convert to

```
List getUsersWithRoles () {
  return executeQuery(
   "SELECT   u FROM user u, role 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 exprs for output variables
- Try to prove that those expressions preserve program semantics
  - if so, convert the code!

Query By Synthesis (QBS)

- Identify potential code fragments
  - i.e., regions of code that fetches persistent data and return values

Code pre-processing

- Find SQL exprs for output variables

Language for verification conditions

- Try to prove that those expressions preserve program semantics
  - If so, convert the code i

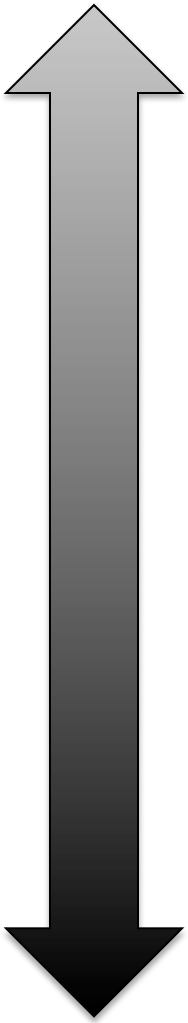
Compute verification conditions

Language for Verification Conditions

Language Requirements

- Compute verification conditions of imperative code fragment
- Handle relational operations as well
  - Order of records matter!
- Output variable expressions must be translatable to SQL
- Expressions should be easily synthesized

Language Design



- Relational Algebra
  - ✔ models database operations
  - ✔ translates to SQL
  - ✖ preserves record order
  - ✖ express VCs of imperative code
- First-Order Logic
  - ✔ models all operations
  - ✔ preserves record order
  - ✖ easy to synthesize
  - ? translates to SQL

Theory of Ordered Relations (TOR)

- Similar to relational algebra
- Model relations as ordered lists

L := program var
| []
| L : L | L : e
| $top_e(L)$
| $L \bowtie_f L$ | $\sigma_f(L)$
| $\pi_f(L)$ | $order_e(L)$

e := L[i]
| e *op* e
| max(L) | min(L)
| sum(L) | avg(L)
| size(L)

Using TOR

- Semantics defined using axioms, e.g.:

$$top_i([]) = []$$
$$top_i(L) = [] \quad \text{if } i = 0$$
$$top_i(h : L) = h : top_{i-1}(L) \quad \text{if } i > 0$$

Computing Verification Conditions

- Standard Hoare logic rules
- Treat loop invariants and post-conditions for output variables as function calls
  - Leave function bodies to be synthesized

Example

```
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; }
```

outerInvariant(users, roles, u, results, …)

innerInvariant(users, roles, u, r, results, …)

results = postCondition(users, roles)

Verification conditions

```
assume(preCondition = true)
preCondition →
   outerInvariant(users/query(…), results/[], …)
outerInvariant(…) ∧ outer loop terminates →
   results = postCondition(users, roles) …
```

Synthesizing Expressions

Synthesis Templates for Invariants and Post-conditions

- Template for invariants:

  $\bigwedge$(<variable in scope> = <TOR expr>)

  – Only consider expressions that type check

- Template for post-conditions:
  <output variable> = <TOR expr>

  – Limit to TOR expressions that are translatable to SQL

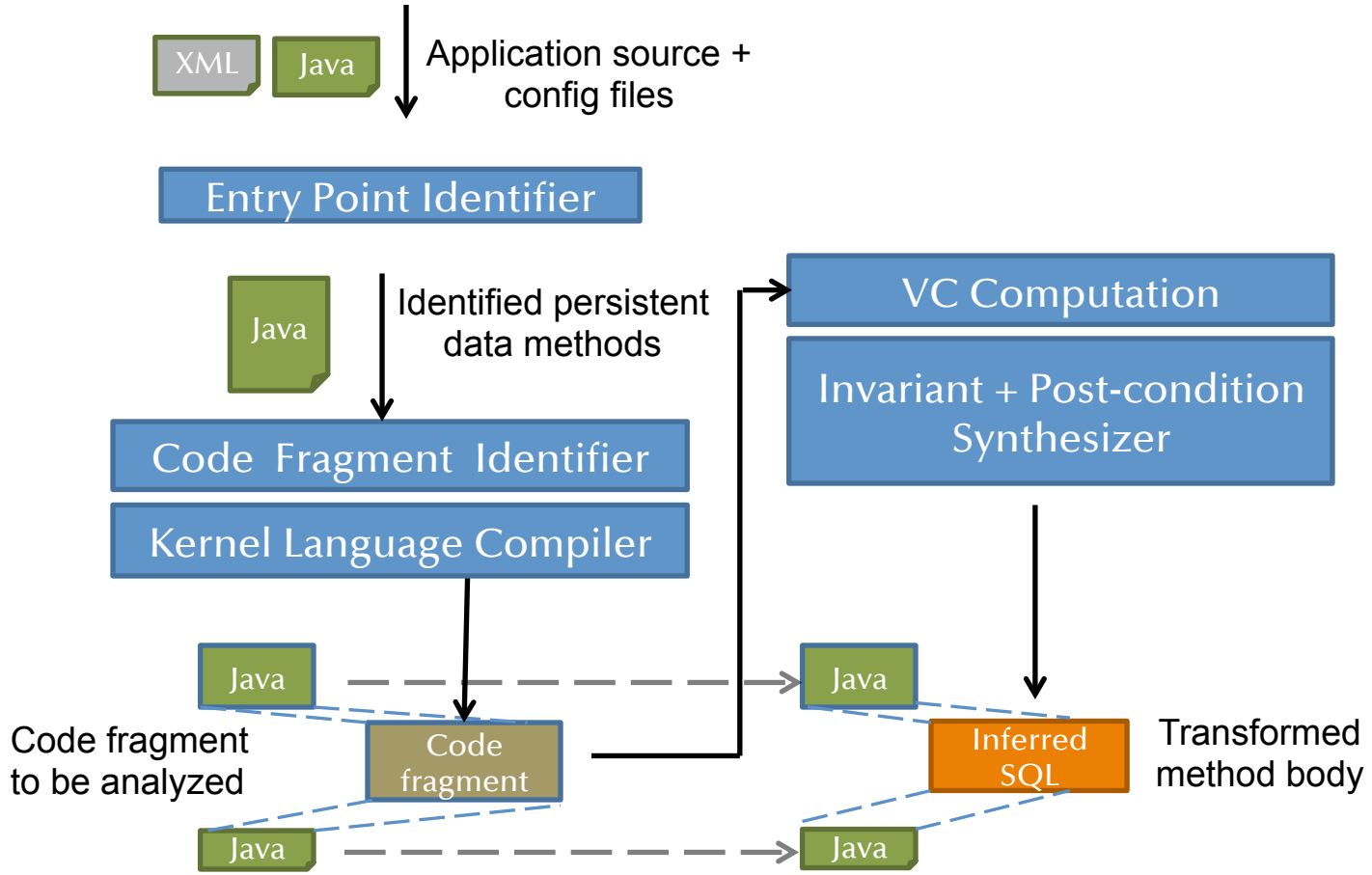
Speeding up Synthesis

- Solve incrementally
  - Increase complexity of expression templates iteratively
- Break symmetries
  - Use relational equivalences, e.g.:
    - $\sigma_f(\sigma_g(L)) = \sigma_g(\sigma_f(L))$
      template only include one of the expressions

Initial Code Fragments Identification

- Find program points that retrieve persistent data
- Run pointer analysis
- Determine where persistent data flow to
- Delimit start and end of code fragment to analyze
- Convert to kernel language

QBS Toolchain



Experiments

Experiment Setup

- No standard benchmarks available
- Experimented on two large scale open source web applications
  - How many code fragments can be converted
  - Difference in page load times while scaling to different database sizes

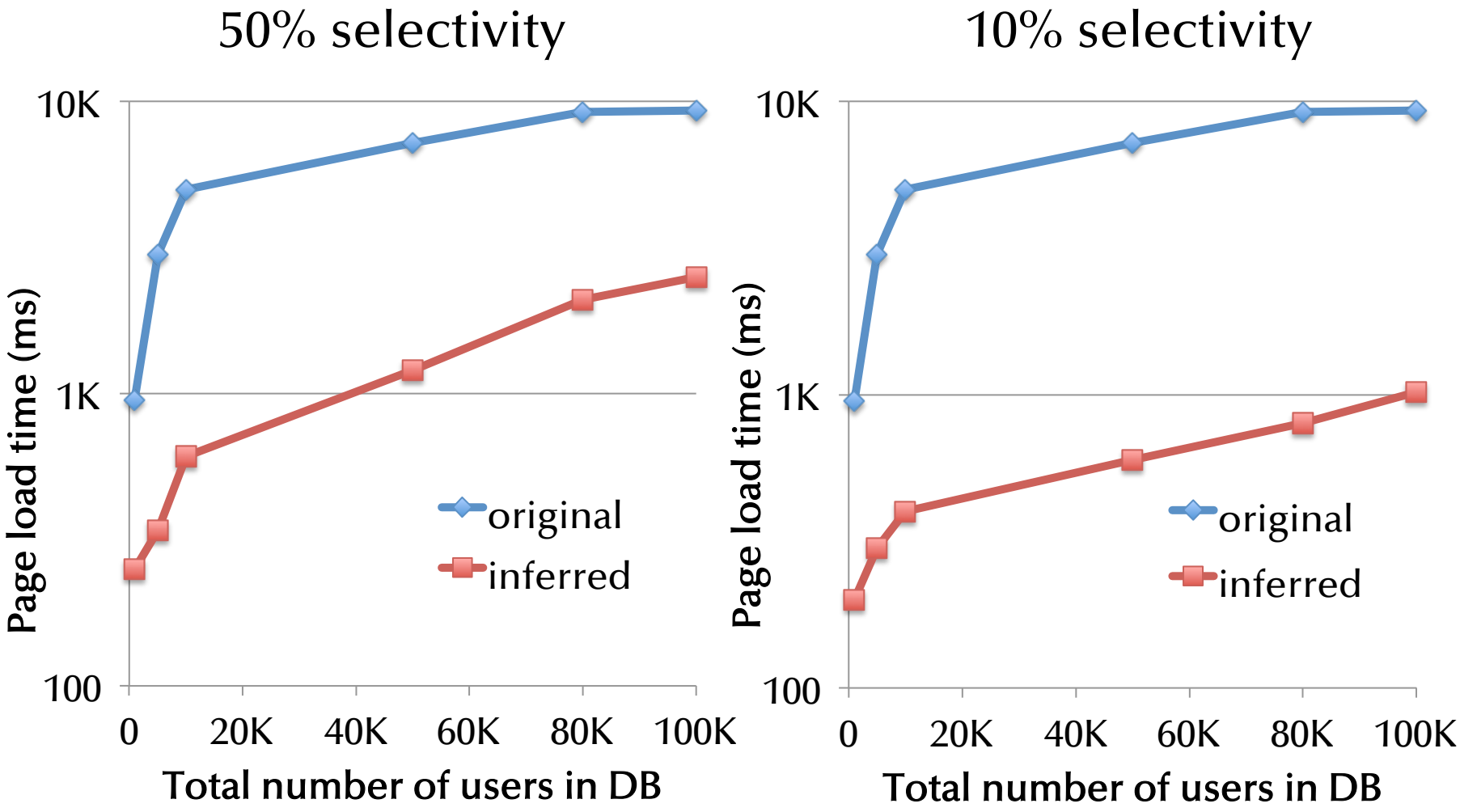
Real-world Evaluation

Wilos (project management application) – 62k LOC		
Operation type	# Fragments found	# Fragments converted
Projection	1	1
Selection	13	10
Join	7	7
Aggregation	11	10
Total	**33**	**28**

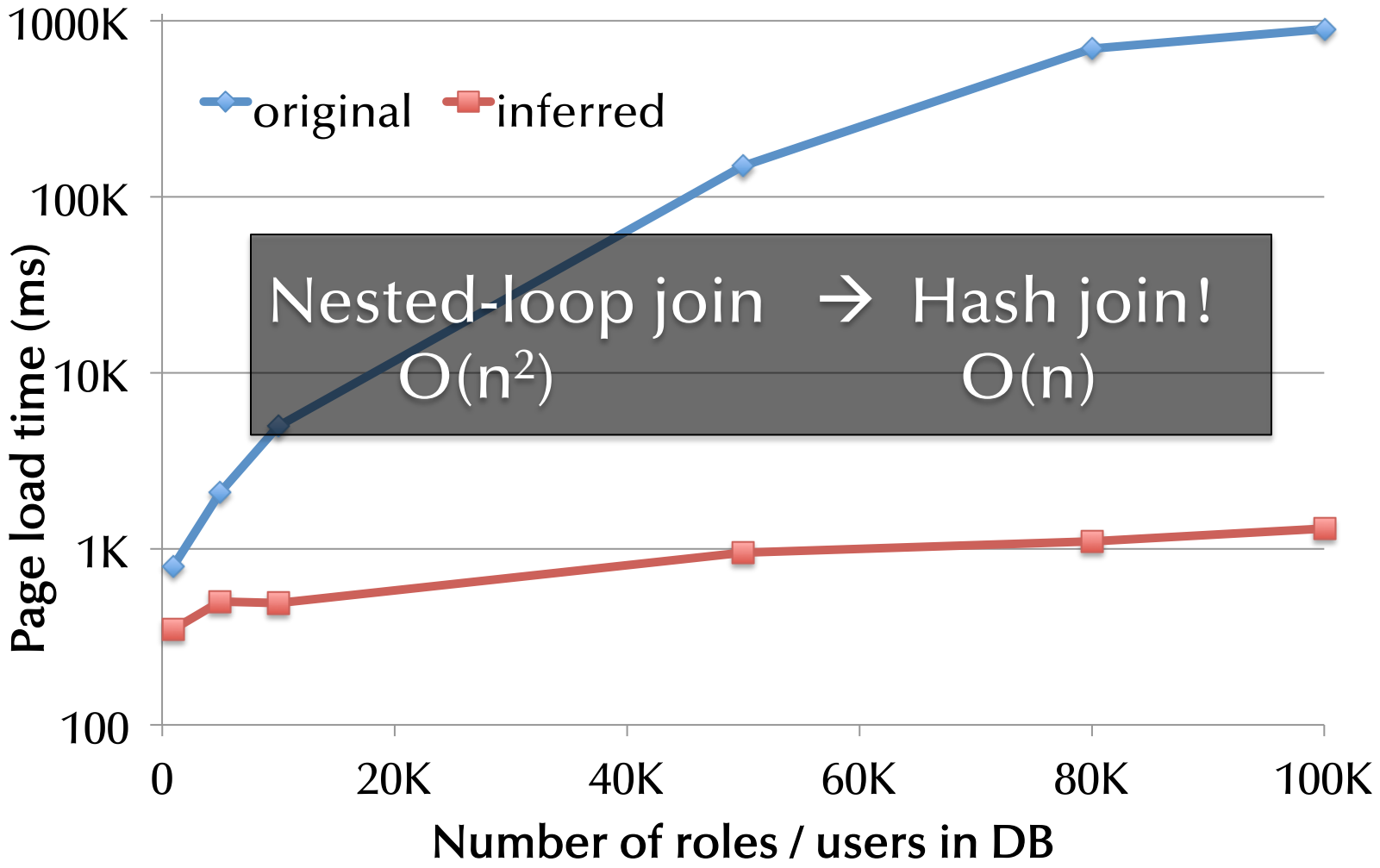
Real-world Evaluation

iTracker (bug tracking system) – 61k LOC		
Operation type	# Fragments found	# Fragments converted
Projection	3	2
Selection	3	2
Join	1	1
Aggregation	9	7
Total	**16**	**12**

Performance Evaluation: Selection Query



Performance Evaluation: Join Query



Failed Code Fragments

- Custom comparators
- Use database schema information

```
List records = Query("SELECT id FROM t");
List results = new ArrayList();
Collections.sort(records); // sort by id
int i = 0;
while (records.get(i).id < 10) {
  results.add(records.get(i)); ++i;
}
```

→ `SELECT id FROM t ORDER BY id LIMIT 10`

Query By Synthesis

Convert imperative program statements into declarative SQL

Shows substantial improvement in real-world applications

Illustrates power of synthesis in enabling complex optimizations