Rethinking the Application-Database Interface

Alvin Cheung
MIT CSAIL || Univ of Washington

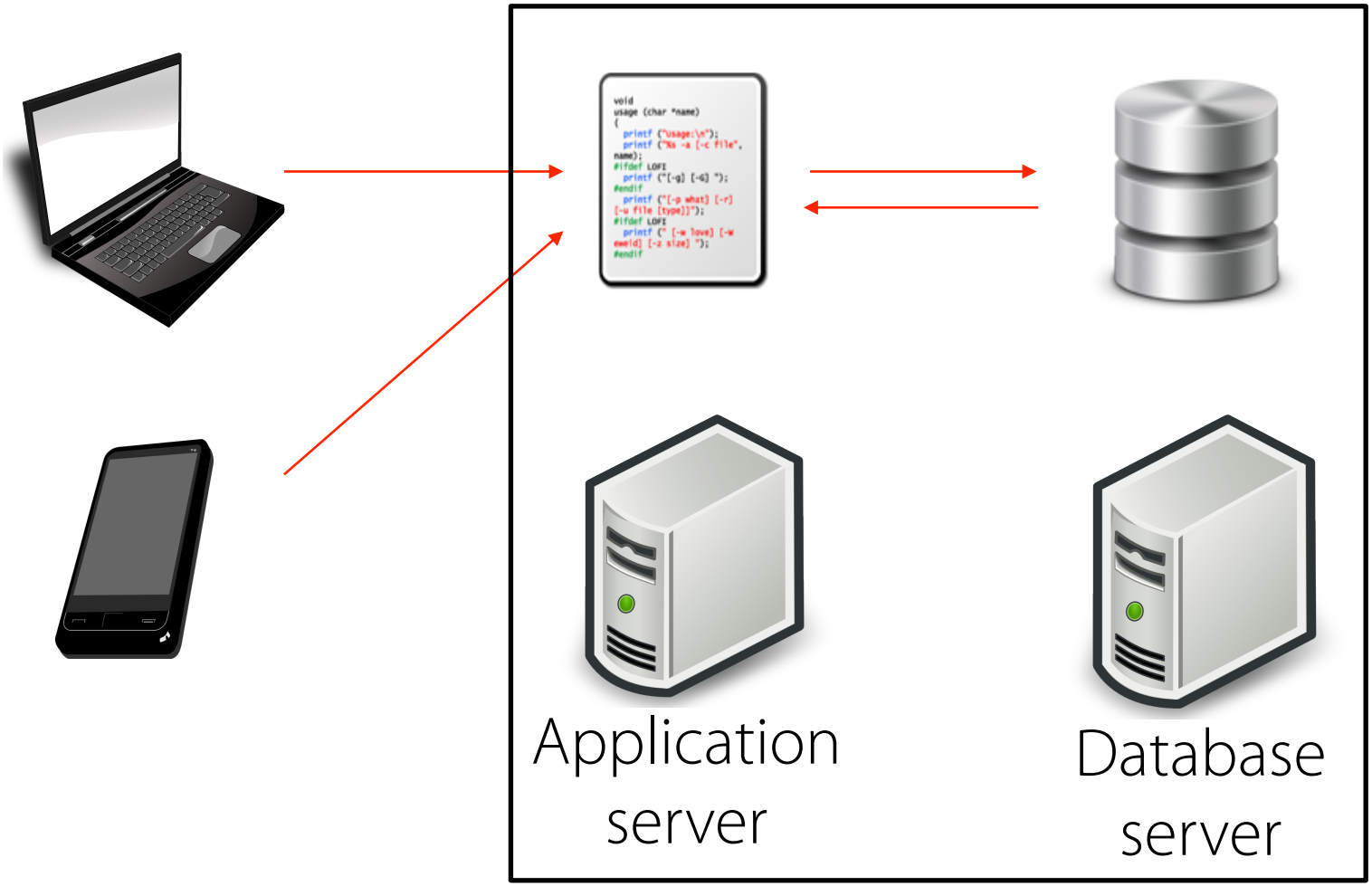

SAP

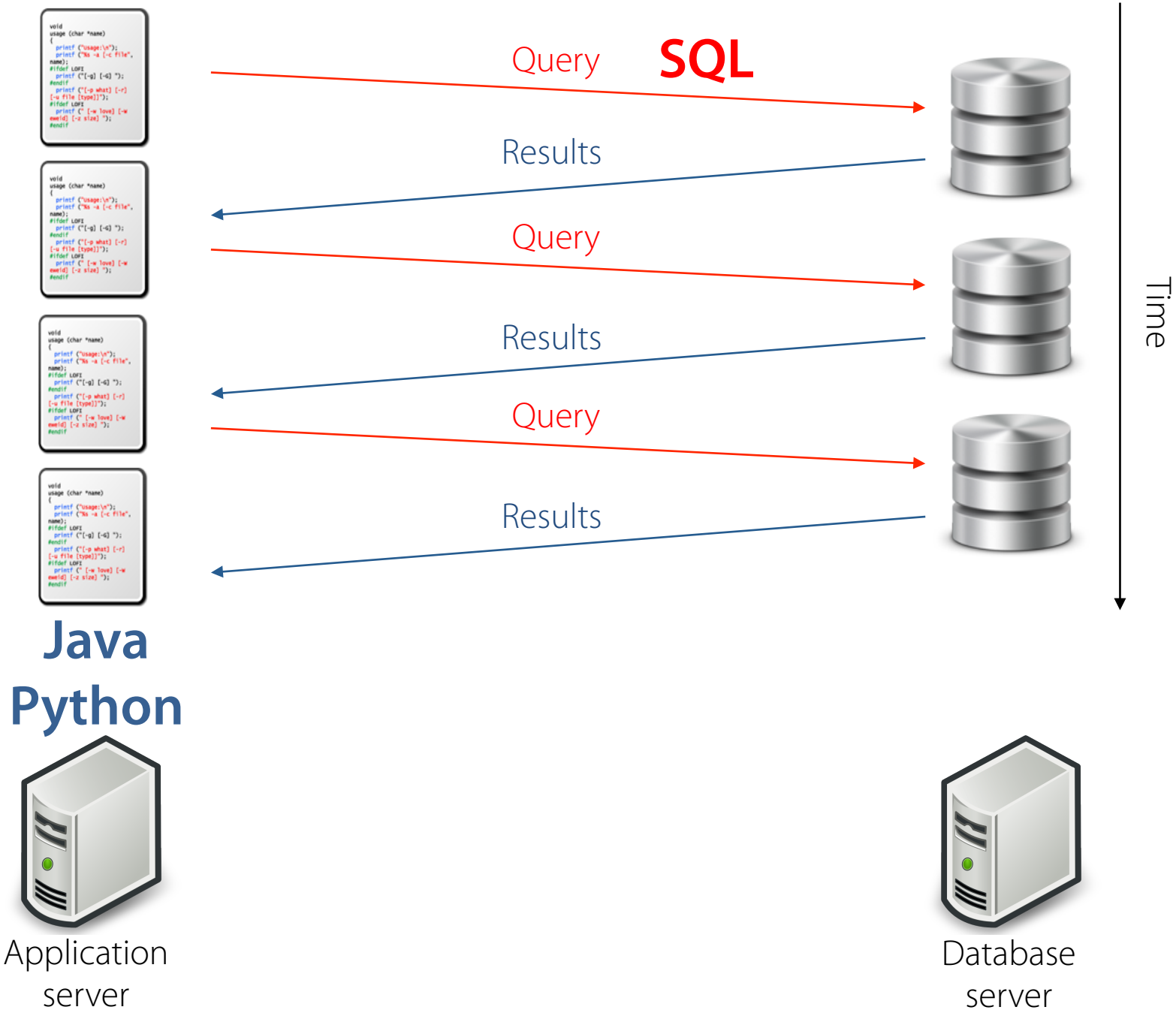

EVERGREEN


OpenMRS
MEDICAL RECORD SYSTEM


salesforce


AA


Application server
Database server

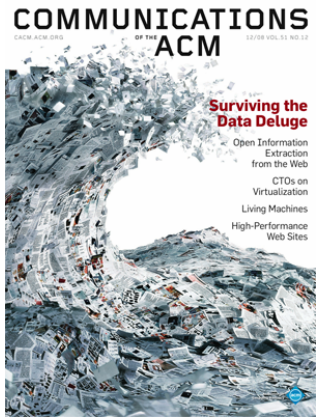

Query
SQL
Results
Query
Results
Query
Results
Time
Java
Python
Application server
Database server

Akamai Study on Travel Site Performance

Loading time – 57% of online shoppers will wait 3 seconds or less before abandoning the site.

MARISSA MAYER @ WEB 2.0

Half a second delay caused a 20% drop in traffic.
Half a second delay killed user satisfaction.



IT now 10 percent of world's electricity consumption
New analysis finds IT power suck has eclipsed aviation

The Register, August 2013

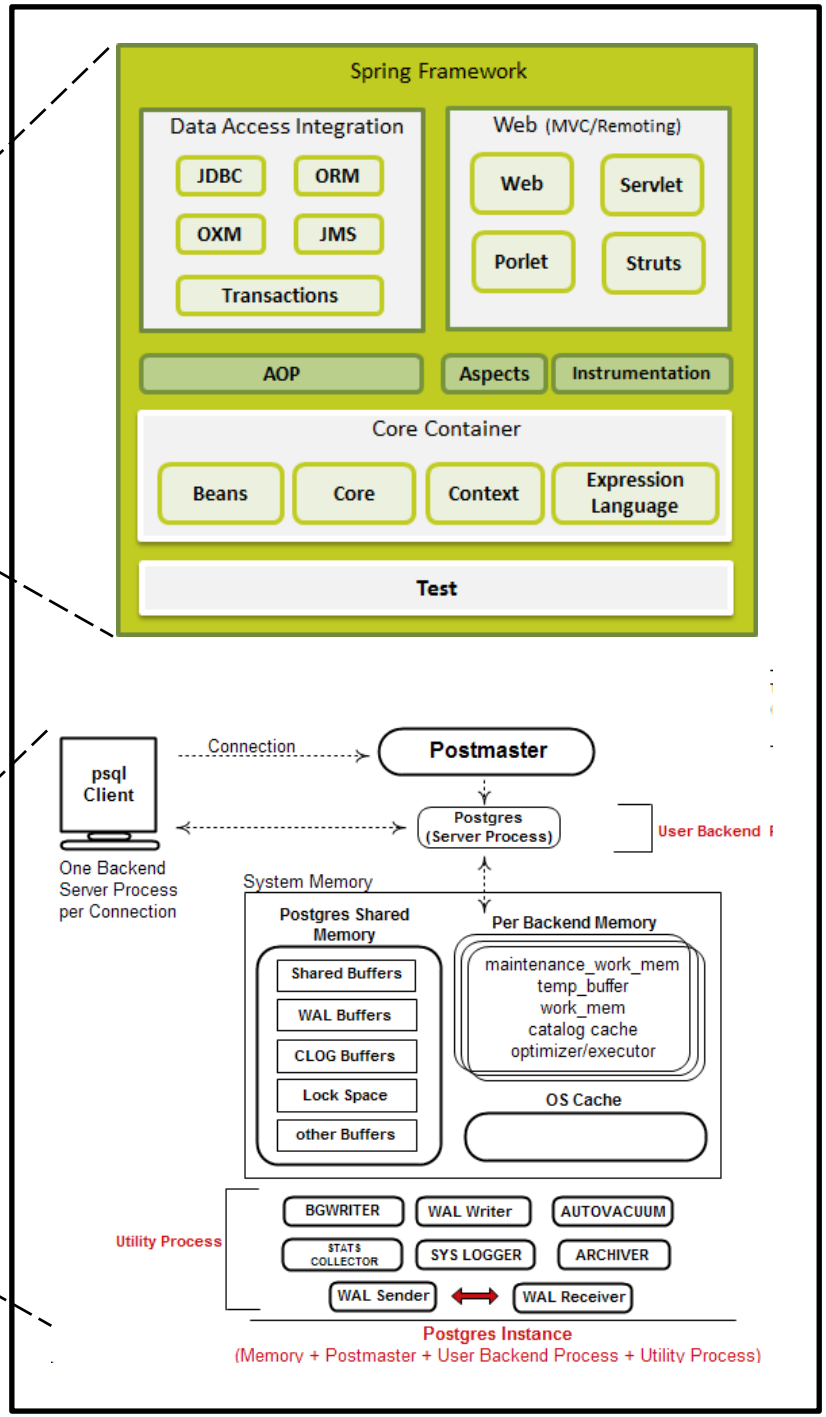
Program logic

Framework

Runtime

OS & Networking

Persistent storage



Program logic

Framework

spring

django

Runtime

Java

python

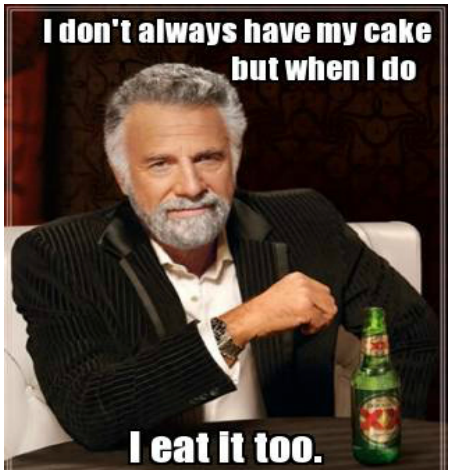
OS & Networking

Persistent storage

Program as specification

Program analysis	Program synthesis	Verification
Data management	Systems	

Multiple orders-of-magnitude performance improvement



StatusQuo: Making Familiar Abstractions Perform Using Program Analysis

CIDR 13 [Best Paper Award]

QUERY By SYNTHESIS

PLDI 13

Seamless transformation of code representation

HOW

WHERE

PYXIS

VLDB 12
SIGMOD 13

Automatic computation migration for best performance

How

Where

```
List getUsersWithRoles () {
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();
  for (User u : users) {
   for (Role r : roles) {
      if (u.roleId == r.id)
         results.add(u);
    }
  }
  return results;
}
```

```
List users = this.userDao.getUsers();
```

```
List getUsers () {
  List users = this.getHibernateTemplate().loadAll(User.class);
  return users;
}

HibernateTemplate org.springframework.org.hibernate3.support.
                  HibernateDaoSupport.getHibernateTemplate () {
  return this.hibernateTemplate;
}

List org.springframework.orm.hibernate3.
     HibernateTemplate.loadAll (Class entityClass) {
  Criteria criteria = session.createCriteria(entityClass);
  criteria.setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY);
  prepareCriteria(criteria);
  return criteria.list();
}

public List org.hibernate.impl.CriteriaImpl.list() {
  return session.list( this )
}
```

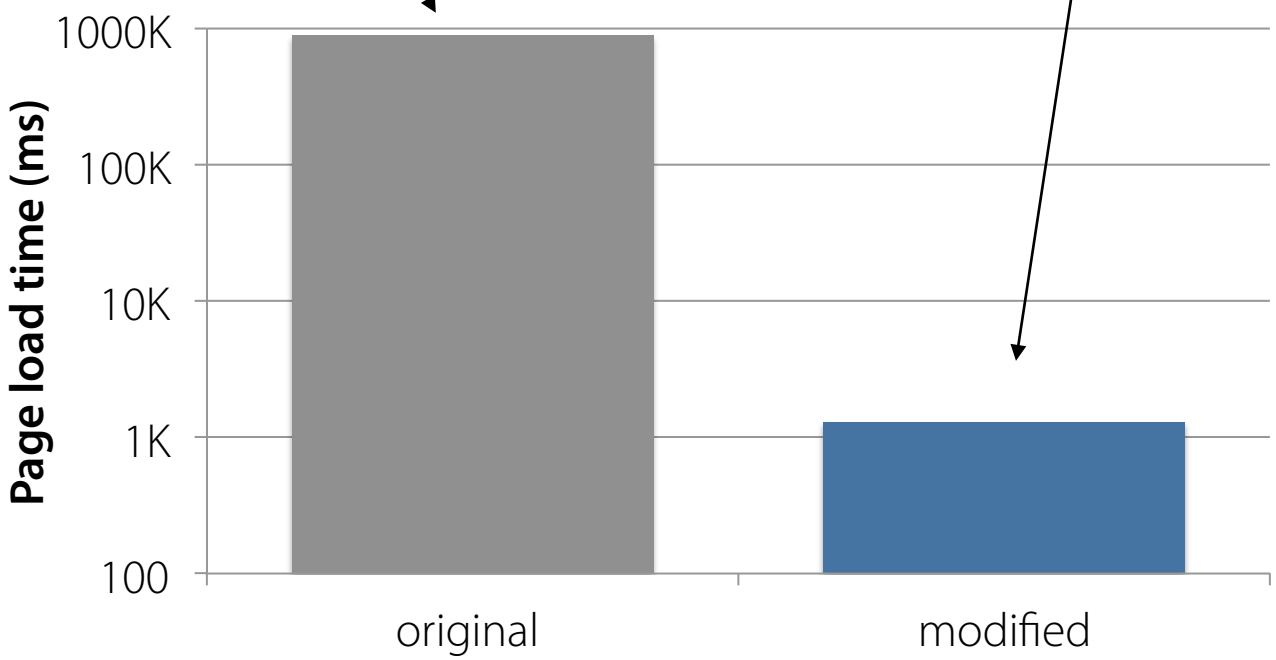
```
users = executeQuery("SELECT * FROM users");
```

```
public List org.hibernate.impl.
           SessionImpl.list (Criteria criteria) {
  checkTransactionSynchStatus();
  String[] implementors = factory.getImplementors(…);
  int size = implementors.length;

  CriteriaLoader[] loaders = new CriteriaLoader[size];
  Set spaces = new HashSet();
  for( int i=0; i <size; i++ ) {
    loaders[i] = new CriteriaLoader(
    getOuterJoinLoadable( implementors[i], … );
    spaces.addAll( loaders[i].getQuerySpaces() );
  }

  autoFlushIfRequired(spaces);

  List results = Collections.EMPTY_LIST;
  dontFlushFromFind++;
  boolean success = false;
  try {
    for( int i=0; i<size; i++ ) {
      final List currentResults = loaders[i].list(this);
      currentResults.addAll(results);
      results = currentResults;
    }
    success = true;
  }
  finally {
    dontFlushFromFind--;
    afterOperation(success);
  }
  return results;
}
```

```
List getUsersWithRoles () {
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();
  for (User u : users) {
     for (Role r : roles) {
       if (u.roleId == r.id)
          results.add(u);
    }}
  return results; }
```

```
executeQuery("SELECT * FROM Users");
executeQuery("SELECT * FROM Roles");
```

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



```
List getUsersWithRoles () {
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();
  for (User u : users) {
     for (Role r : roles) {
       if (u.roleId == r.id)
          results.add(u);
    }}
  return results; }
```

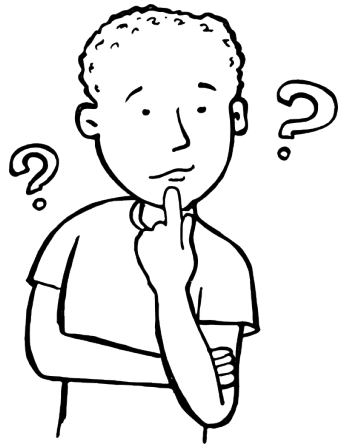
```
executeQuery("SELECT * FROM Users");
executeQuery("SELECT * FROM Roles");
```

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

How should computation be specified

Good software engineering

Inefficient



Optimized

Brittle

QUERY By SYNTHESIS

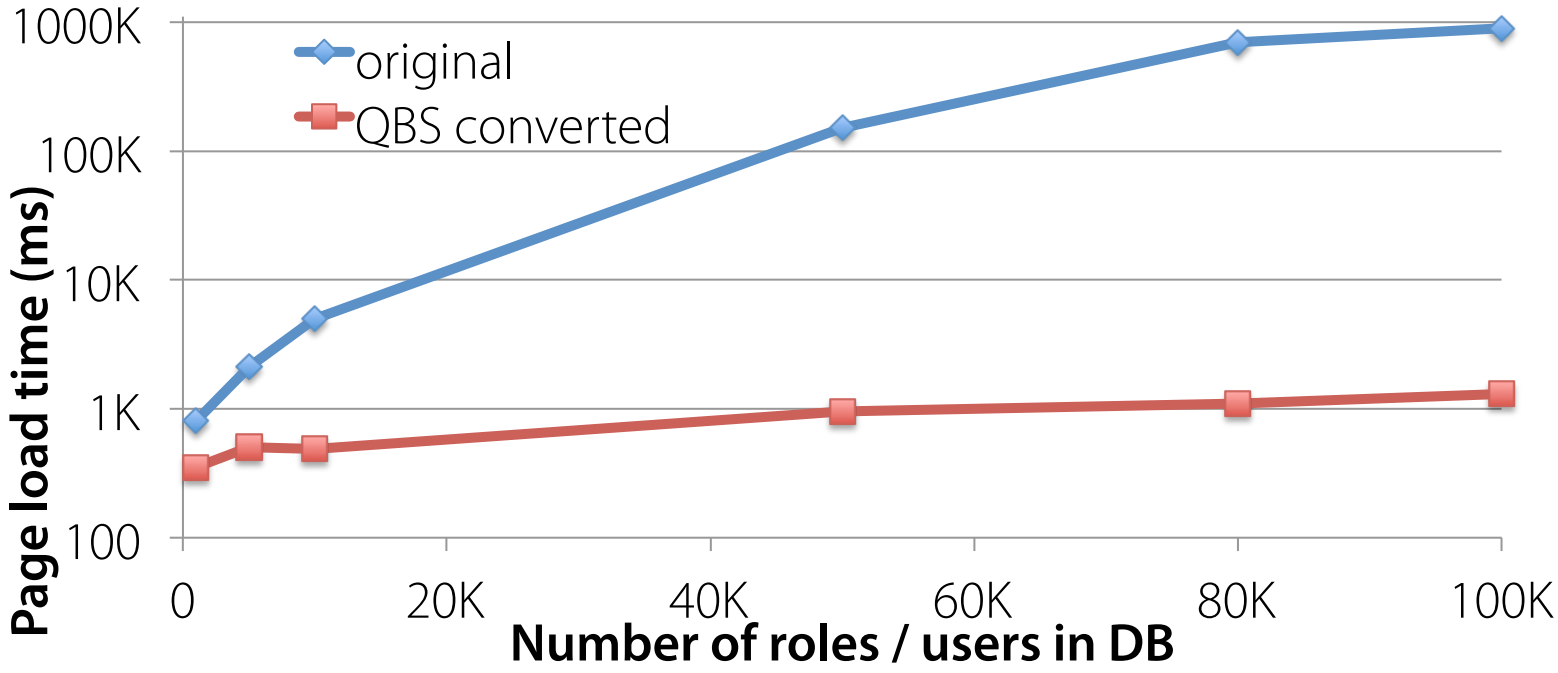
Seamless transformation of code representation

Optimizing Database-Backed Applications with Query Synthesis

PLDI 13

Wilos (project management application) – 62k LOC	
# fragments found	# fragments converted
33	**29**

iTracker (bug tracking system) – 61k LOC	
# fragments found	# fragments converted
16	**12**



```
List getUsersWithRoles () {
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();
  for (User u : users) {
     for (Role r : roles) {
       if (u.roleId == r.id)
          results.add(u);
    }}
  return results; }
```

output variable

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

Scan for code fragments

```
List getUsersWithRoles () {
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();
  for (User u : users) {
     for (Role r : roles) {
       if (u.roleId == r.id)
          results.add(u);
    }}
  return results; }
```

output variable

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

Scan for code fragments

Find SQL expressions for output variables

Verify found SQL expressions

```
List getUsersWithRoles () {
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();

  for (User u : users) {
    for (Role r : roles) {
      if (u.roleId == r.id)
        results.add(u);
    }
  }
```

results = π_{user}(users $\bowtie_{\text{roleId} = \text{id}}$ roles)

```
  return results;
}
```

???

Scan

Find

Verify

$\bowtie$: join of two lists

π : projection of fields

Hoare-style program verification

```
for (int i = 0; i < users.size(); ++i) {
  for (int j = 0; j < roles.size(); ++j)
  {
    if (users[i].roleId == roles[j].id)
        results.add(users[i]);
  }
```

$\bowtie$: list / element join
π : projection of fields

invariant (program state $_i$)

results $_{j+1}$ = results $_j$:
users[i] $\bowtie_{roleId = id}$ roles [0..j]

results$_{i+1}$ = results$_i$: users[i] $\bowtie_{roleId = id}$ roles → post-condition (program state $_{final}$)

Verification Conditions

Invariant is true on loop entry:
first enter loop → invariant (program state $_{initial}$)

Invariant is preserved:
loop continues ∧ invariant (program state $_i$) → invariant (program state $_{i+1}$)

Invariant implies post-condition on loop exit:
loop terminates ∧ invariant (program state $_i$) → post-condition (program state $_{final}$)

Scan
Find
Verify

```
for (int i = 0; i < users.size(); ++i) {
  for (int j = 0; j < roles.size(); ++j)
  {
    if (users[i].roleId == roles[j].id)
        results.add(users[i]);
  }
```

$\bowtie$: list / element join
π : projection of fields

invariant (program state $_i$)
results $_{j+1}$ = results $_j$:
users[i] $\bowtie_{roleId = id}$ roles [0..j]

results$_{i+1}$ = results$_i$: users[i] $\bowtie_{roleId = id}$ roles $\rightarrow$ post-condition (program state $_{final}$)

Invariant is preserved:

loop continues $\wedge$ invariant (program state $_i$) $\rightarrow$ invariant (program state $_{i+1}$)

---

j < size(roles) $\wedge$ results $_{j+1}$ = results $_j$: users[i] $\bowtie_{roleId = id}$ roles [0..j] $\rightarrow$

[π_{id}(roles[j]) = π_{roleId}(users[i]) $\wedge$ invariant (program state $_{i+1}$)]

$\vee$

[π_{id}(roles[j]) $\neq$ π_{roleId}(users[i]) $\wedge$ invariant (program state $_{i+1}$)]

```
for (int i = 0; i < users.size(); ++i) {
  for (int j = 0; j < roles.size(); ++j)
  {
    if (users[i].roleId == roles[j].id)
        results.add(users[i]);
  }
```

$\bowtie$: list / element join
π : projection of fields

invariant (program state $_i$)
results $_{j+1}$ = results $_j$:
users[i] $\bowtie_{roleId = id}$ roles [0..j]

results$_{i+1}$ = results$_i$: users[i] $\bowtie_{roleId = id}$ roles $\rightarrow$ post-condition (program state $_{final}$)

Invariant is preserved:

loop continues $\wedge$ invariant (program state $_i$) $\rightarrow$ invariant (program state $_{i+1}$)

---

j < size(roles) $\wedge$ results $_{j+1}$ = results $_j$: users[i] $\bowtie_{roleId = id}$ roles [0..j] $\rightarrow$

[π_{id}(roles[j]) = π_{roleId}(users[i]) $\wedge$ results : users[i] = …]

$\vee$

[π_{id}(roles[j]) $\neq$ π_{roleId}(users[i]) $\wedge$ invariant (program state $_{i+1}$)]

```
for (int i = 0; i < users.size(); ++i) {
  for (int j = 0; j < roles.size(); ++j)
  {
    if (users[i].roleId == roles[j].id)
        results.add(users[i]);
  }
```

$\bowtie$: list / element join
π : projection of fields

invariant (program state $_i$)
results $_{j+1}$ = results $_j$:
users[i] $\bowtie_{roleId = id}$ roles [0..j]

results$_{i+1}$ = results$_i$: users[i] $\bowtie_{roleId = id}$ roles $\rightarrow$ post-condition (program state $_{final}$)

Invariant is preserved:

loop continues $\wedge$ invariant (program state $_i$) $\rightarrow$ invariant (program state $_{i+1}$)

---

j < size(roles) $\wedge$ results $_{j+1}$ = results $_j$: users[i] $\bowtie_{roleId = id}$ roles [0..j] $\rightarrow$

[π_{id}(roles[j]) = π_{roleId}(users[i]) $\wedge$ results : users[i] = results : users[i] $\bowtie_{roleId = id}$ roles [0..j+1]]

$\vee$

[π_{id}(roles[j]) $\neq$ π_{roleId}(users[i]) $\wedge$ invariant (program state $_{i+1}$)]

```
for (int i = 0; i < users.size(); ++i) {
  for (int j = 0; j < roles.size(); ++j)
  {
    if (users[i].roleId == roles[j].id)
        results.add(users[i]);
  }
```

$\bowtie$: list / element join
π : projection of fields

invariant (program state $_i$)

results $_{j+1}$ = results $_j$:
users[i] $\bowtie_{roleId = id}$ roles [0..j]

results$_{i+1}$ = results$_i$: users[i] $\bowtie_{roleId = id}$ roles $\rightarrow$ post-condition (program state $_{final}$)

Invariant is preserved:

loop continues $\wedge$ invariant (program state $_i$) $\rightarrow$ invariant (program state $_{i+1}$)

---

j < size(roles) $\wedge$ results $_{j+1}$ = results $_j$: users[i] $\bowtie_{roleId = id}$ roles [0..j] $\rightarrow$

[π_{id}(roles[j]) = π_{roleId}(users[i]) $\wedge$ results : users[i] = results : users[i] $\bowtie_{roleId = id}$ roles [0..j+1]]

$\vee$

[π_{id}(roles[j]) $\neq$ π_{roleId}(users[i]) $\wedge$ results = …]

```
for (int i = 0; i < users.size(); ++i) {
  for (int j = 0; j < roles.size(); ++j)
  {
    if (users[i].roleId == roles[j].id)
        results.add(users[i]);
  }
```

$\bowtie$: list / element join
π : projection of fields

invariant (program state $_i$)
results $_{j+1}$ = results $_j$:
users[i] $\bowtie_{roleId = id}$ roles [0..j]

results$_{i+1}$ = results$_i$: users[i] $\bowtie_{roleId = id}$ roles $\rightarrow$ post-condition (program state $_{final}$)

Invariant is preserved: ✅

loop continues $\wedge$ invariant (program state $_i$) $\rightarrow$ invariant (program state $_{i+1}$)

---

j < size(roles) $\wedge$ results $_{j+1}$ = results $_j$: users[i] $\bowtie_{roleId = id}$ roles [0..j] $\rightarrow$

[π_{id}(roles[j]) = π_{roleId}(users[i]) $\wedge$ results : users[i] = results : users[i] $\bowtie_{roleId = id}$ roles [0..j+1]]

$\vee$

[π_{id}(roles[j]) $\neq$ π_{roleId}(users[i]) $\wedge$ results = results : users[i] $\bowtie_{roleId = id}$ roles [0..j+1]]

```
for (int i = 0; i < users.size(); ++i) {
  for (int j = 0; j < roles.size(); ++j)
  {
    if (users[i].roleId == roles[j].id)
        results.add(users[i]);
  }
```

$\bowtie$: list / element join
π : projection of fields

invariant (program state $_i$)

results $_{j+1}$ = results $_j$:
users[i] $\bowtie_{roleId = id}$ roles [0..j]

results$_{i+1}$ = results$_i$: users[i] $\bowtie_{roleId = id}$ roles → post-condition (program state $_{final}$)

Verification Conditions

Invariant is true on loop entry:
first enter loop → invariant (program state $_{initial}$)

Invariant is preserved:
loop continues ∧ invariant (program state $_i$) → invariant (program state $_{i+1}$)

Invariant implies post-condition on loop exit:
loop terminates ∧ invariant (program state $_i$) → post-condition (program state $_{final}$)

Scan
Find
Verify

Hoare style reasoning

loop continues $\wedge$ invariant (program state $_i$) $\rightarrow$
invariant (program state $_{i+1}$)
...

+

Relational algebra

σ , $\bowtie$, π , ...

||

Theory of ordered relations

Theory of ordered relations

Similar to relational algebra

Model relations as ordered lists

```
L := program var
   | []
   | L : L | L : e
   | L[0..e]
   | L ⋈_f L | σ_f(L)
   | π_f(L) | order_e(L)
```

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

Scan

Find

Verify

Theory of ordered relations

Similar to relational algebra

Model relations as ordered lists

```
                 ⎧ []       [0..i]      = []
L[0..i]          ⎪ L[0..i]              = []                  if i = 0
                 ⎩ ([e]:L)[0..i]        = [e]:L[0..i-1]       if i > 0
```

Scan

Find

Verify

```
List getUsersWithRoles ()
{
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();

  for (User u : users)
  {
    for (Role r : roles)
    {
      if (u.roleId == r.id)
          results.add(u);
    }
  }
  return results;
}
```

Scan

Find

Verify

outerInvariant(program state)

innerInvariant(program state)

post-condition(program state)

output variable → results = …

```
List getUsersWithRoles ()
{
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();

  for (User u : users)
  {
    for (Role r : roles)
    {
      if (u.roleId == r.id)
          results.add(u);
    }
  }
  return results;
}
```

Scan

Find

Verify

outerInvariant(program state)

innerInvariant(program state)

post-condition(program state)

↓

results = …

Synthesis to the rescue!

```
List getUsersWithRoles ()
{
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();

  for (User u : users)
  {
    for (Role r : roles)
    {
      if (u.roleId == r.id)
          results.add(u);
    }
  }
  return results;
}
```

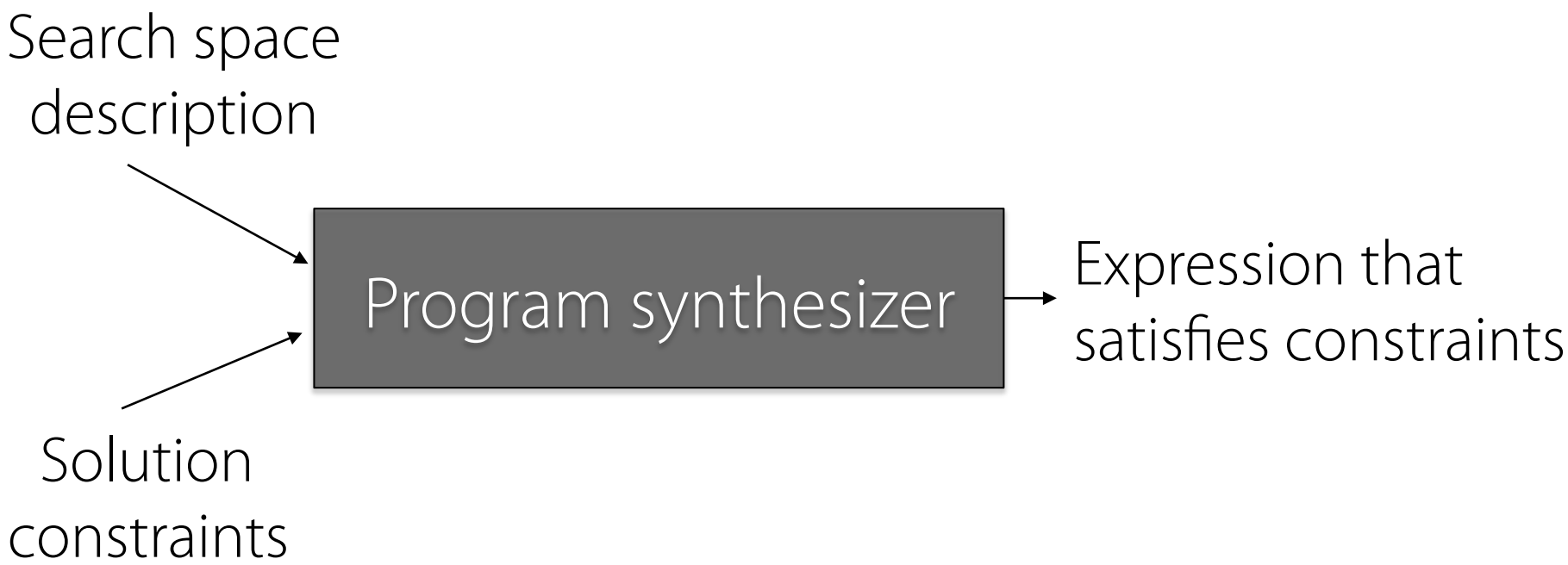
outerInvariant(program state)

innerInvariant(program state)

post-condition(program state)

↓

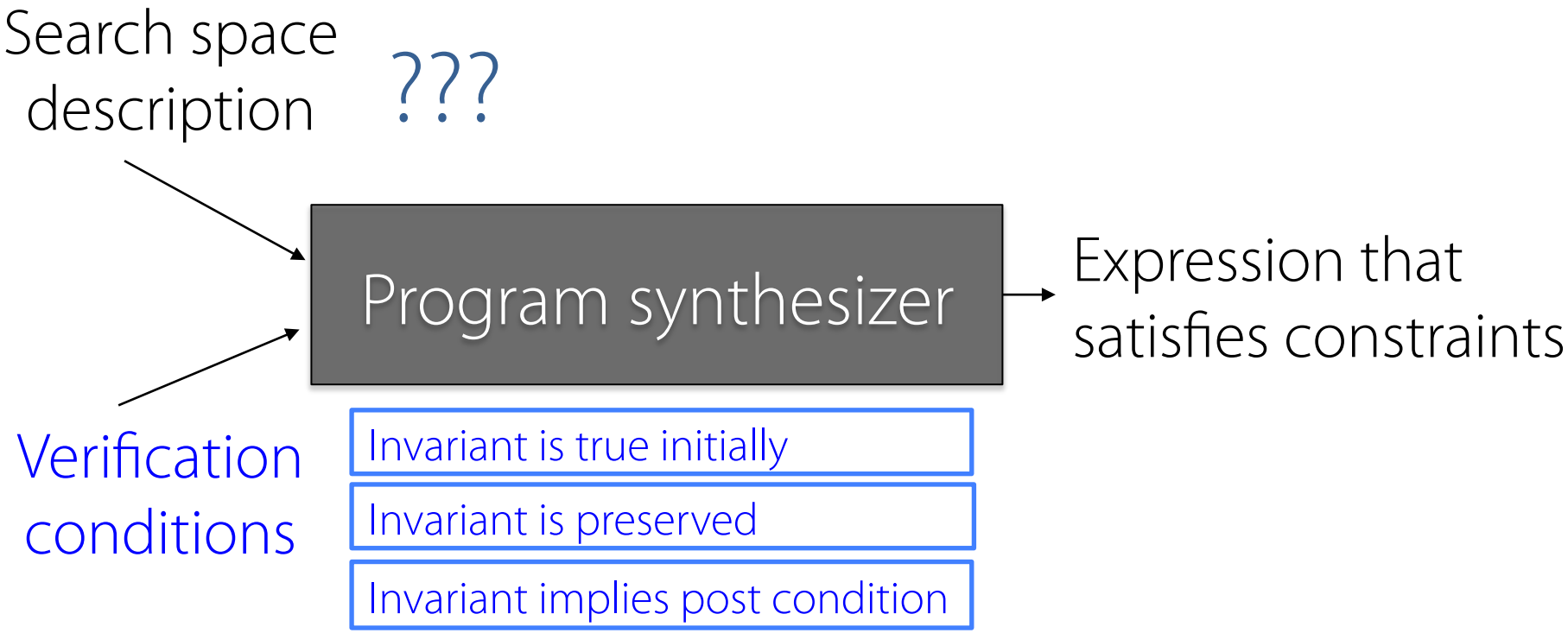
results = …



Scan

Find

Verify



Scan

Find

Verify

```
List getUsersWithRoles () {
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();

  for (User u : users) {

    for (Role r : roles) {
      if (u.roleId == r.id)
        results.add(u);
    }
  }

  return results;
}
```

Dataflow analysis:

users roles

Structural analysis:

σ $\bowtie$

Possible expressions in invariants and post condition:

σ_f(users) users[0..e]

π_f(users $\bowtie_g$ roles)

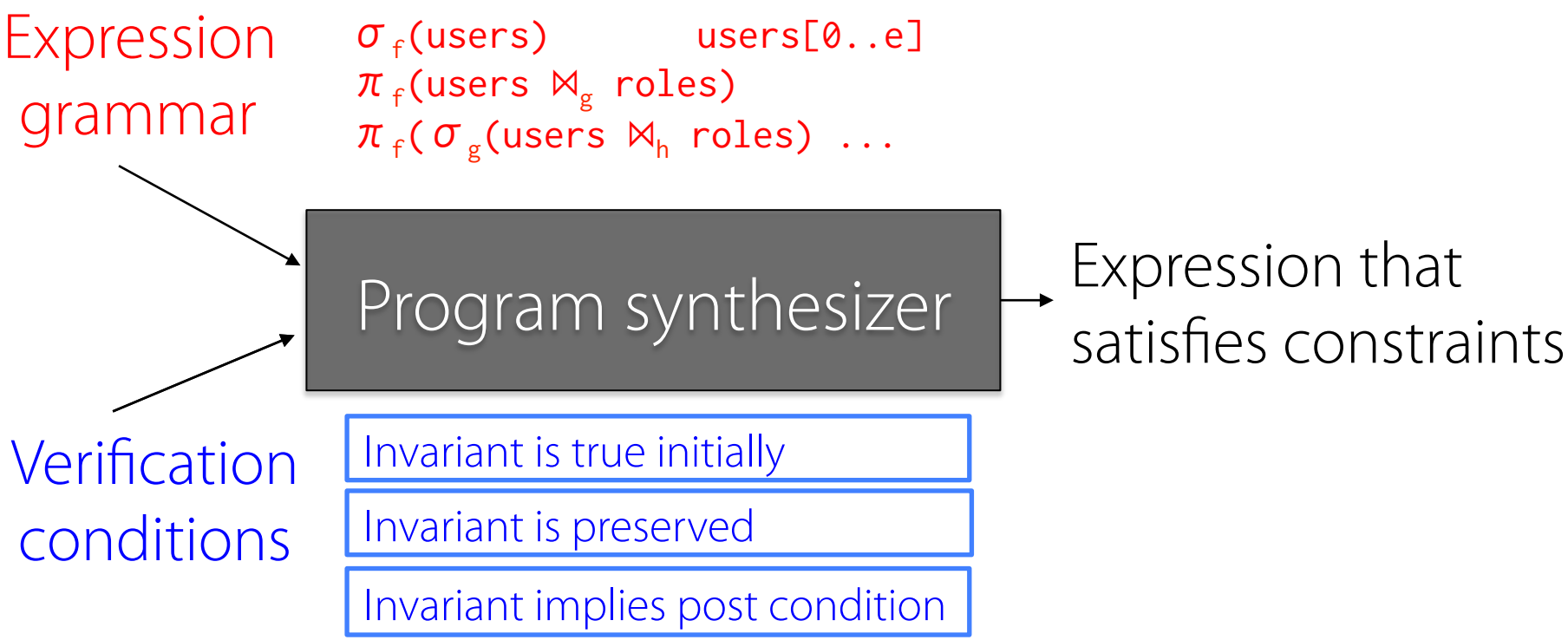
π_f(σ_g(users $\bowtie_h$ roles)

...

Scan

Find

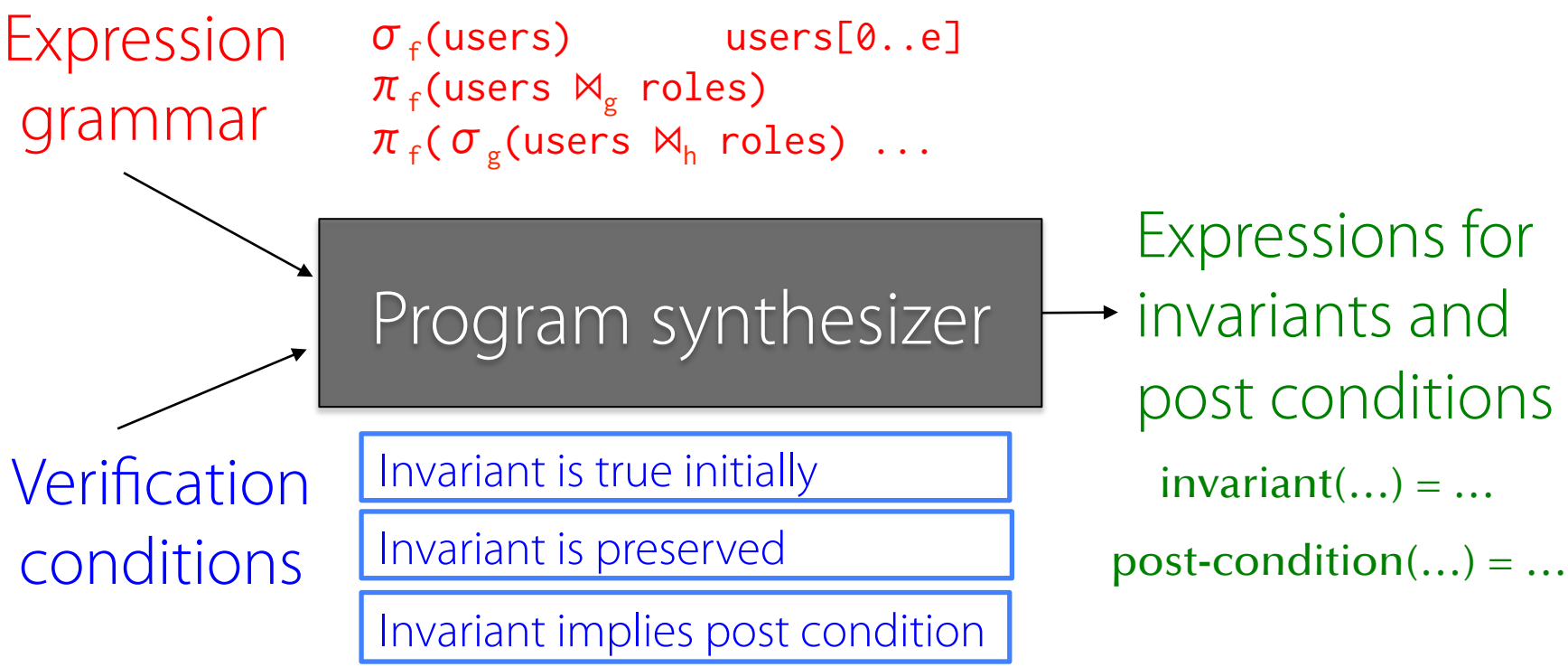
Verify



Scan

Find

Verify



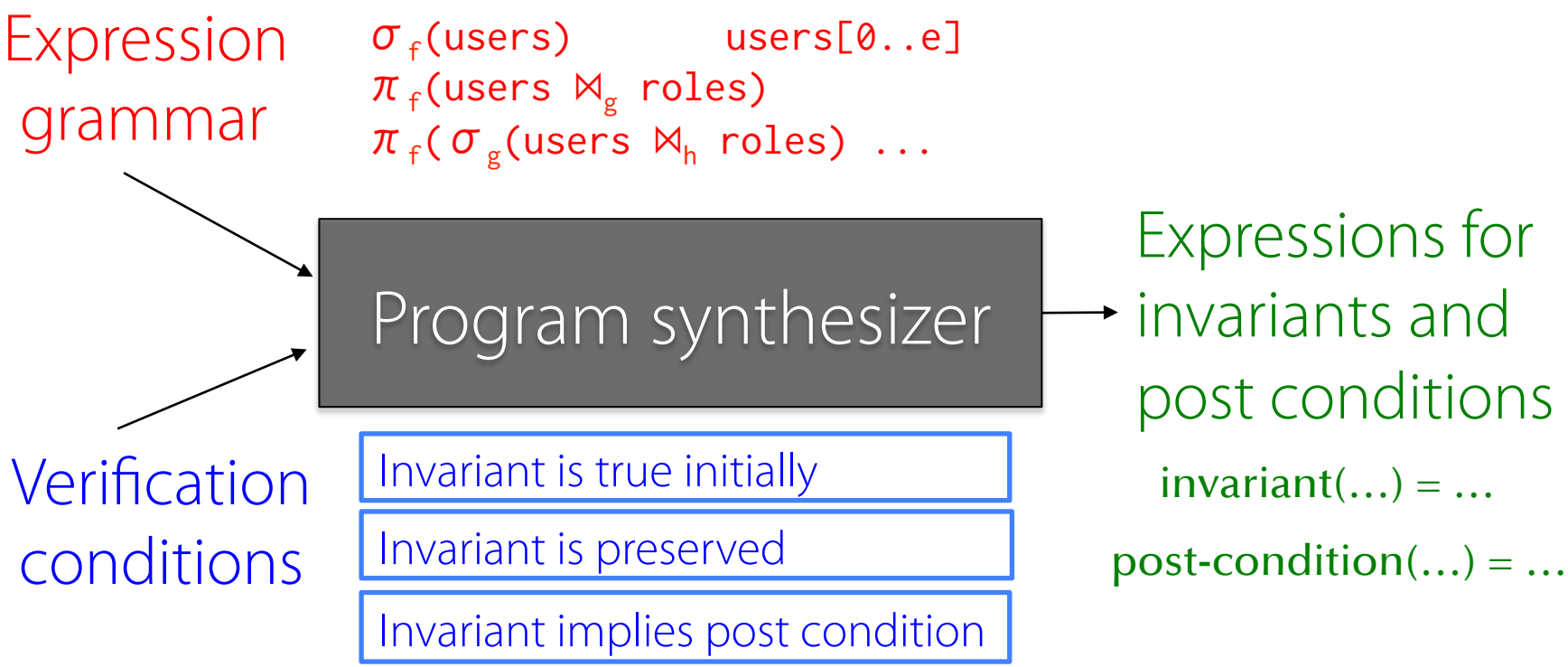
Use the Sketch synthesizer

symbolic manipulation of search space

Scan

Find

Verify



Solve incrementally

Increase complexity of expression grammar iteratively

Break symmetries

Use relational equivalences

$\sigma_f(\sigma_g(L)) = \sigma_g(\sigma_f(L))$

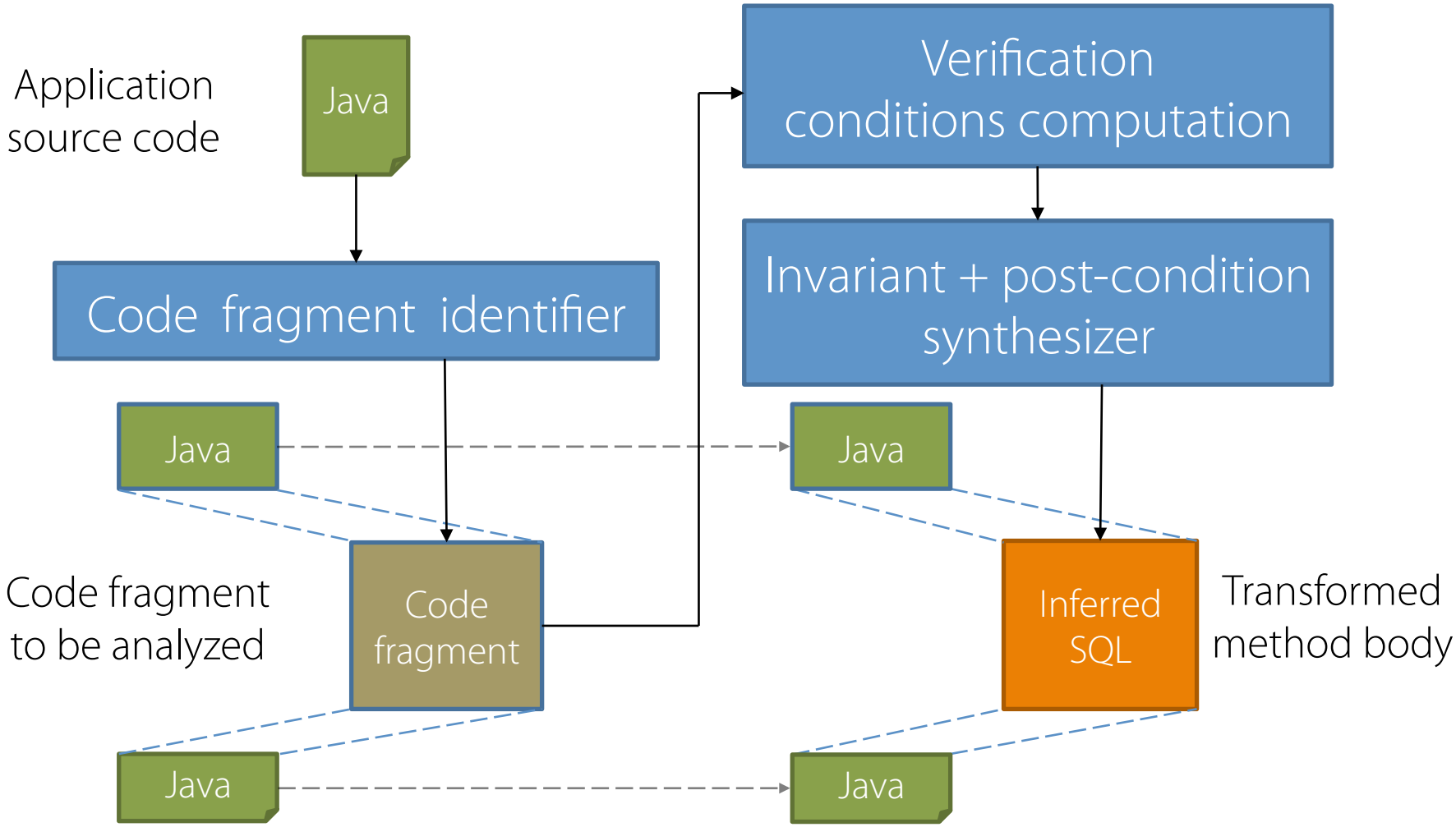
grammar only include one of the expressions

Scan

Find

Verify

Query By Synthesis toolchain



Experiments

Two large-scale open source web apps

- Developed by teams of programmers
- Use variety of idioms to access persistent data

Generate benchmarks from code

Goals

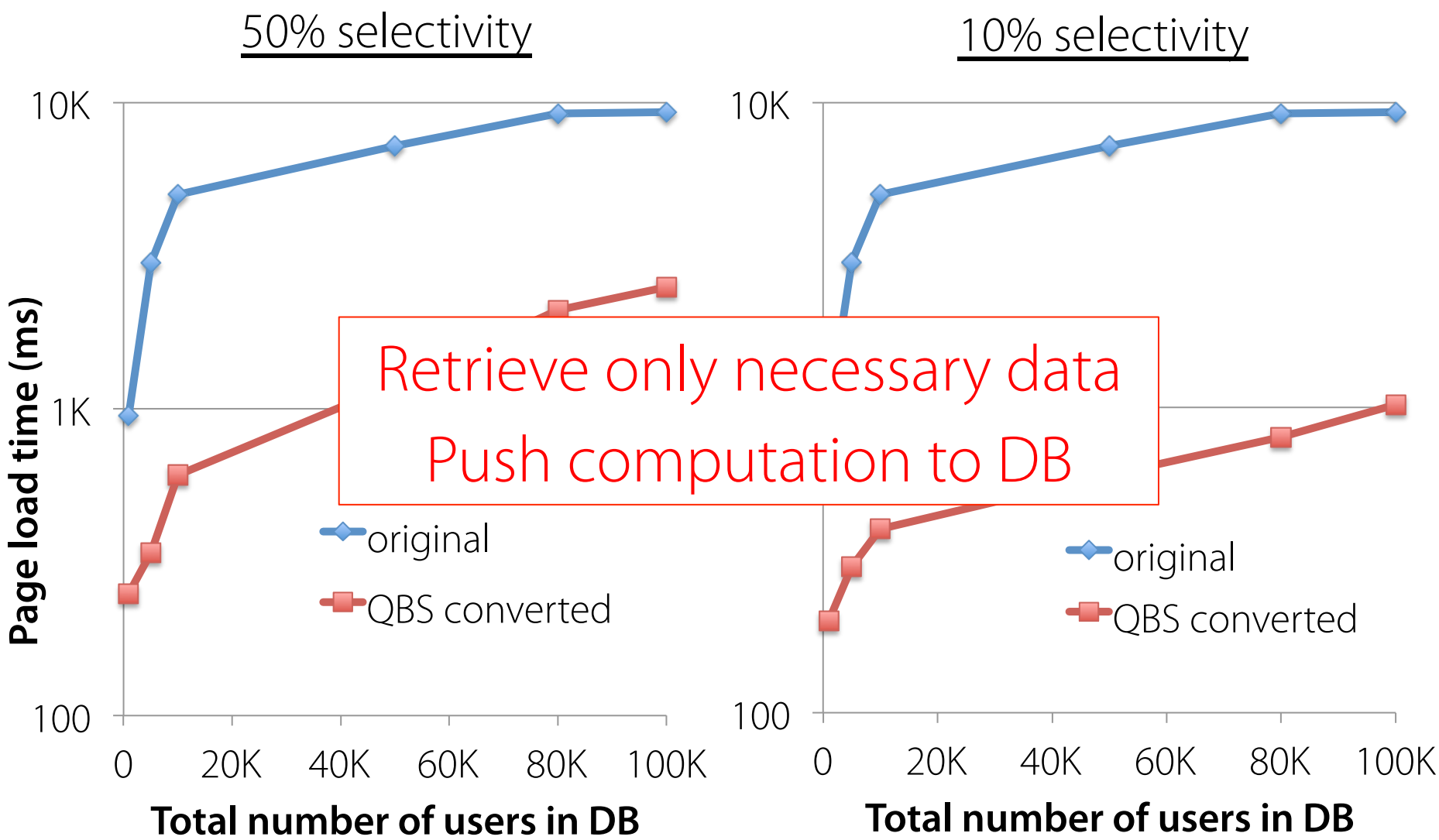
- Number of cases successfully converted
- Effect of conversions

Wilos (project management application) – 62k LOC		
Operation type	# fragments found	# fragments converted
Projection	2	2
Selection	13	10
Join	7	7
Aggregation	11	10
Total	**33**	**29**
iTracker (bug tracking system) – 61k LOC		
Projection	3	2
Selection	3	2
Join	1	1
Aggregation	9	7
Total	**16**	**12**

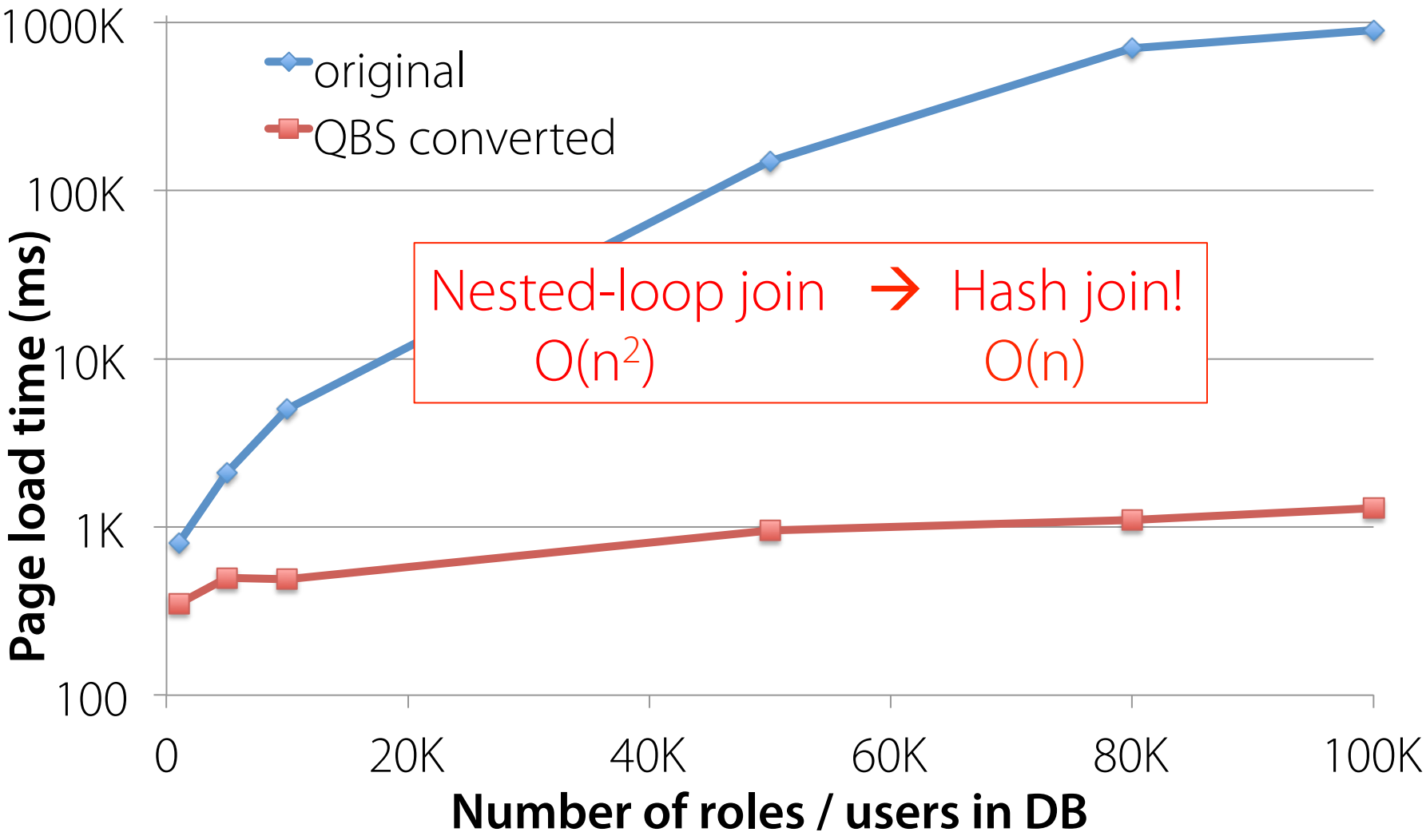
Max code fragment size: ~200 lines of code

Max compilation time: 5 minutes

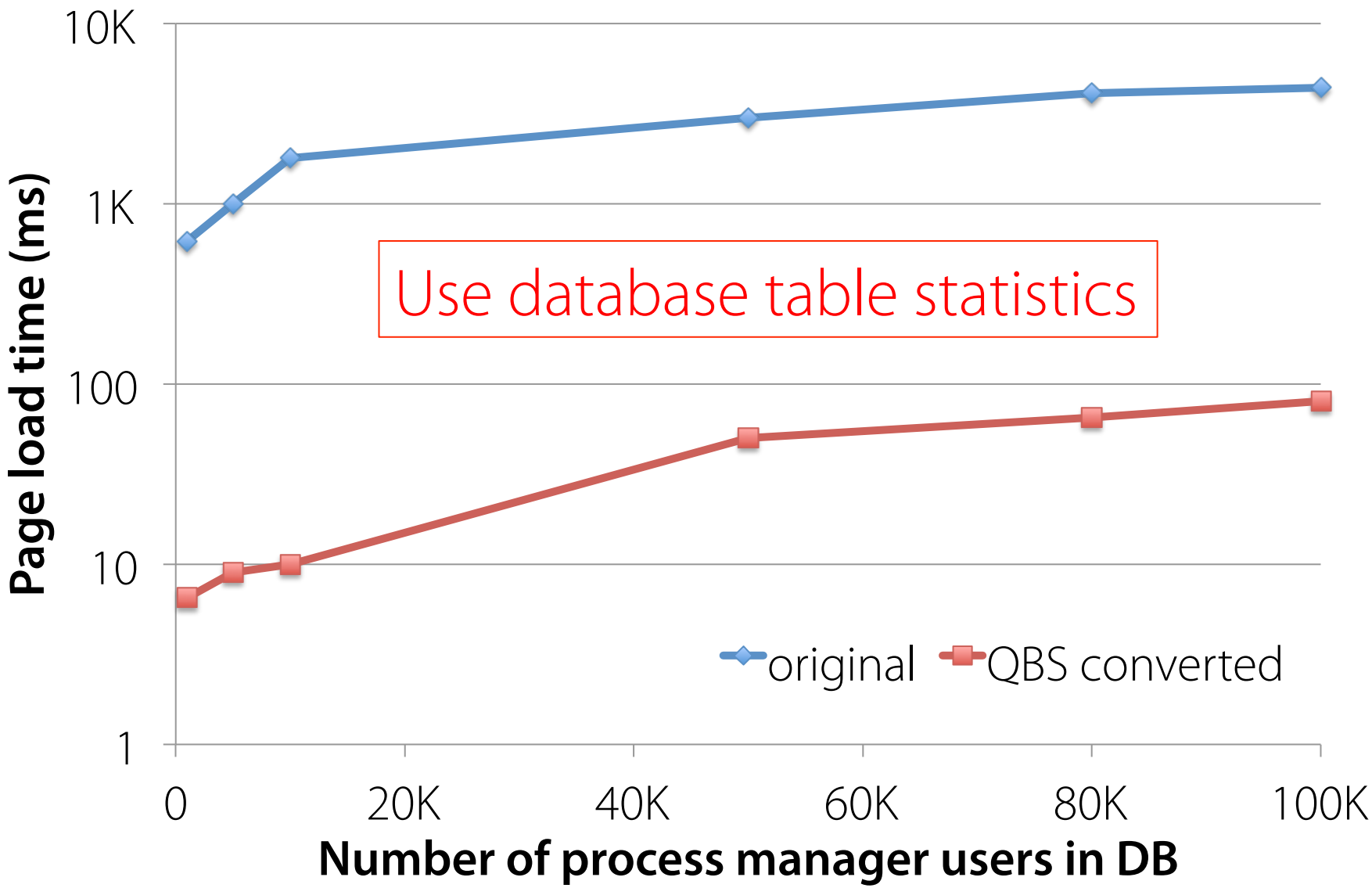
QBS code runs 10x faster for selections



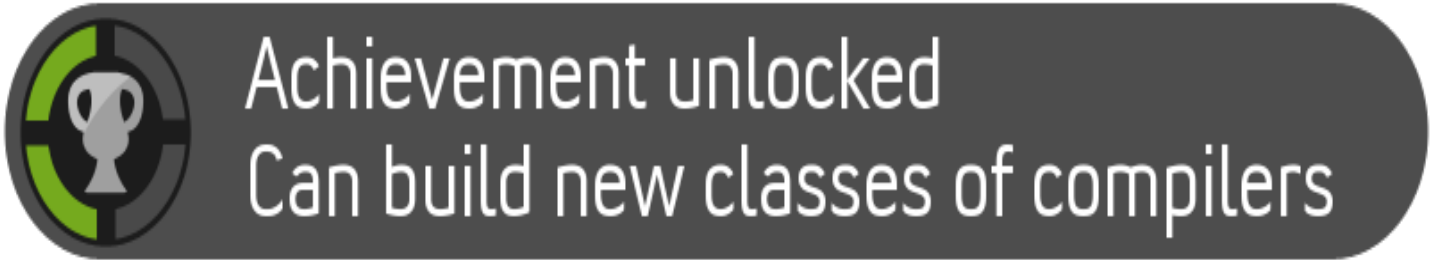
QBS code runs asymptotically faster for joins



QBS code runs 100x faster for aggregates




Achievement unlocked
Can translate code representations


Achievement unlocked
Can build new classes of compilers

Program logic

Framework

spring

django

Runtime

Java

python

Networking

Persistent storage

Program as specification

Program analysis

Program synthesis

Verification

Data management

Systems

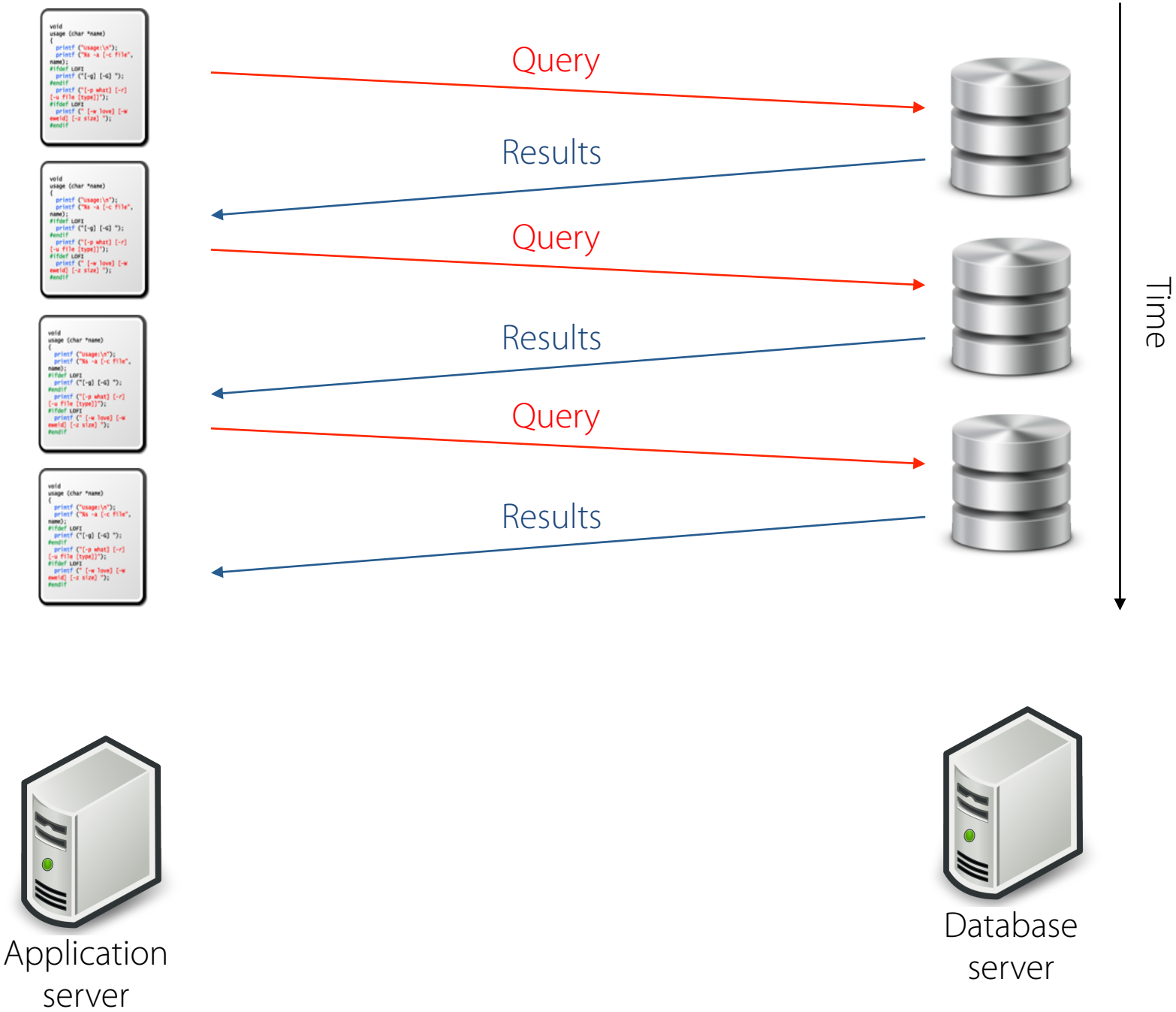
Multiple orders-of-magnitude performance improvement

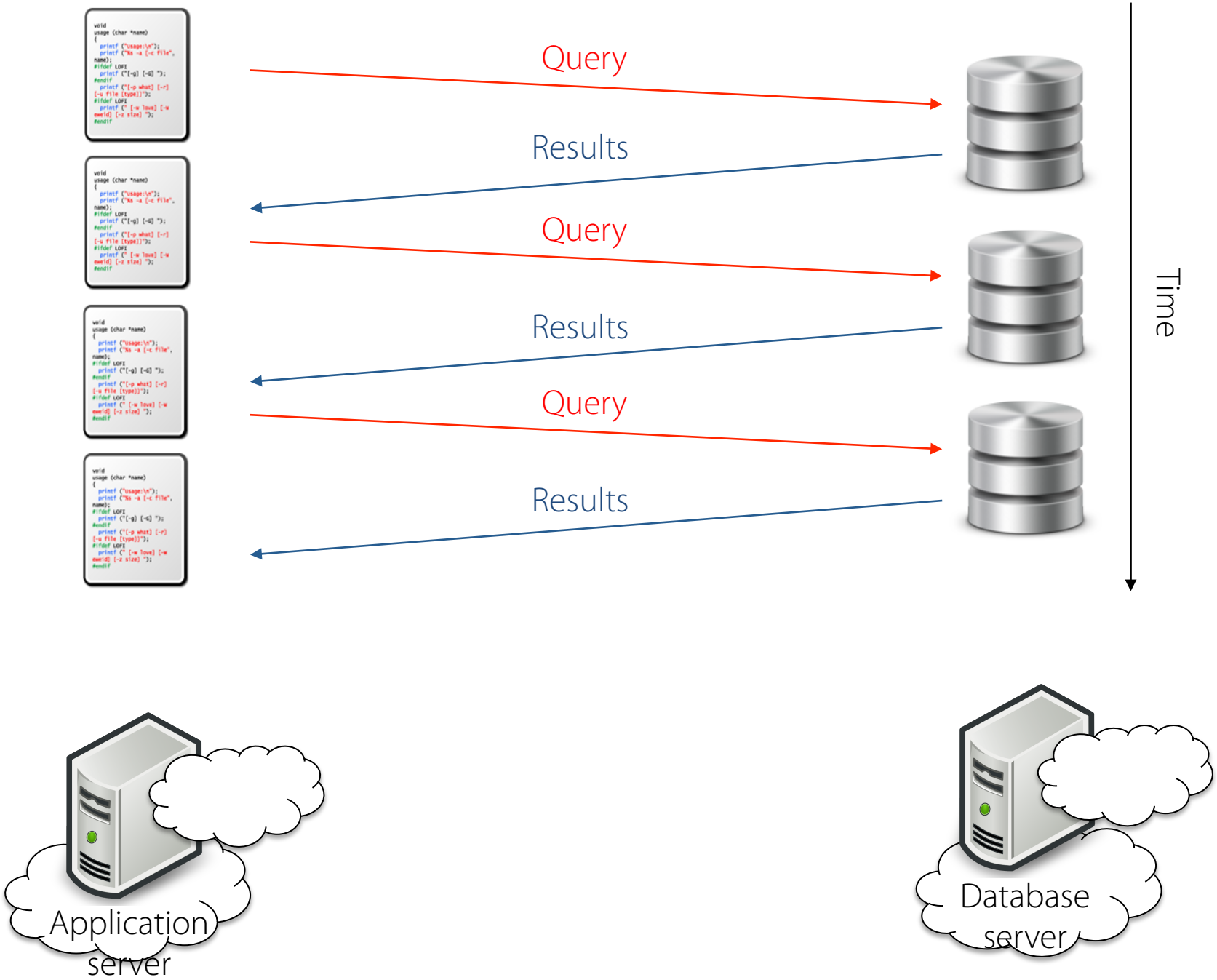
StatusQuo: Making Familiar Abstractions Perform Using Program Analysis

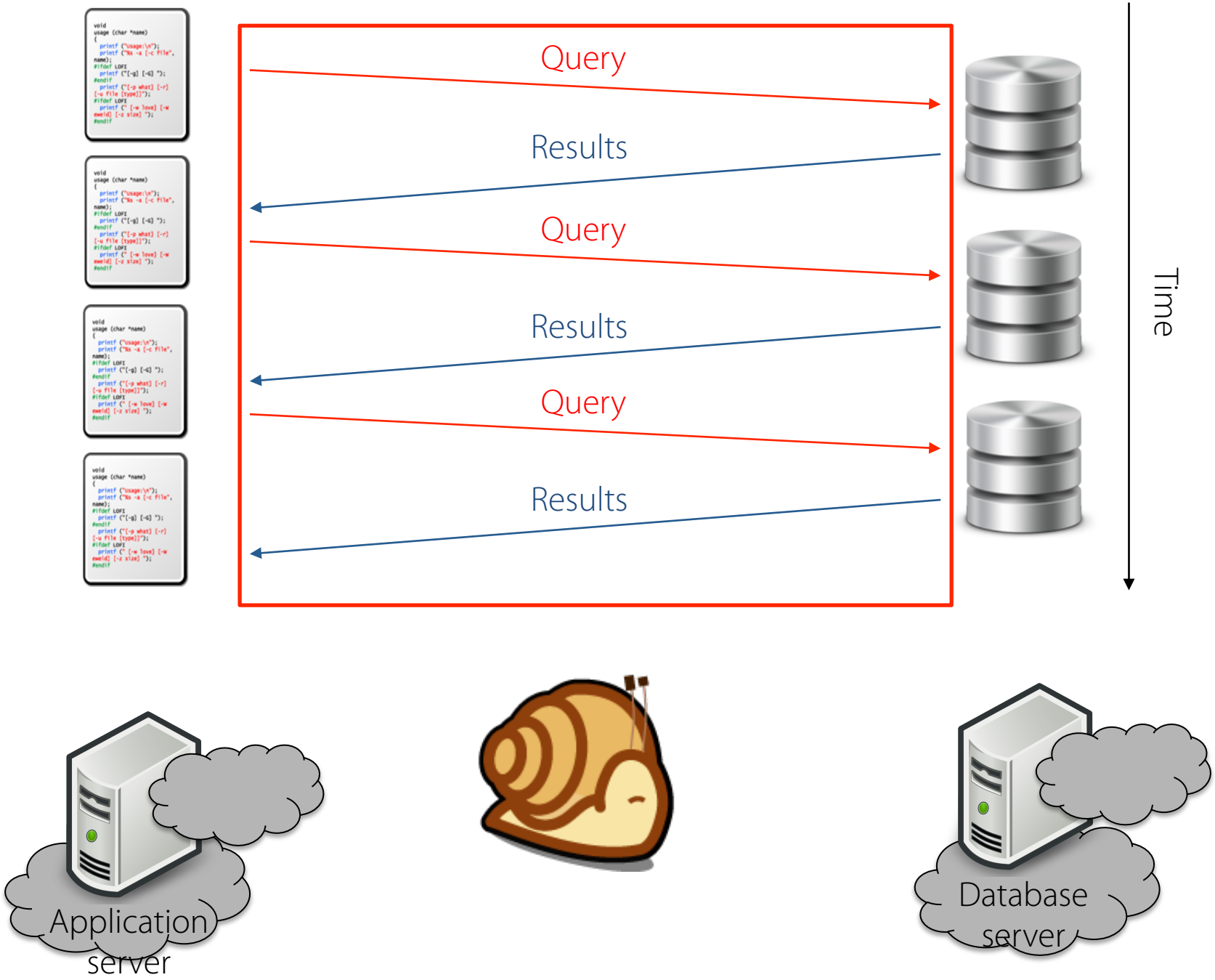
CIDR 13 [Best Paper Award]

How

Where


Query
Results
Query
Results
Query
Results
Time
Application server
Database server


Query
Results
Query
Results
Query
Results
Time
Application server
Database server


Query
Results
Query
Results
Query
Results
Time
Application server
Database server

```
discount = query("SELECT... FROM customer");

total = orderTotal * (1 - discount);

credit = query("SELECT... FROM customer");

if (credit < total)
  print("Only: " + credit + "in acct");
else
  update("UPDATE... SET credit = " + (credit - total));
```

DB
```
discount = query("SELECT... FROM customer");
```

APP
```
total = orderTotal * (1 - discount);
```

DB
```
credit = query("SELECT... FROM customer");
```

APP
```
if (credit < total)
  print("Only: " + credit + "in acct");
else
```

DB
```
  update("UPDATE... SET credit = " + (credit - total));
```

DB
```
discount = query("SELECT... FROM customer");
```

network communication

APP
```
total = orderTotal * (1 - discount);
```

network communication

DB
```
credit = query("SELECT... FROM customer");
```

network communication

APP
```
if (credit < total)
   print("Only: " + credit + "in acct");
```

network communication

```
else
```

DB
```
   update("UPDATE... SET credit = " + (credit - total));
```

Move computation to the data

DB

```
discount = query("SELECT... FROM customer");

total = orderTotal * (1 - discount);

credit = query("SELECT... FROM customer");

if (credit < total)
```

Stored procedure

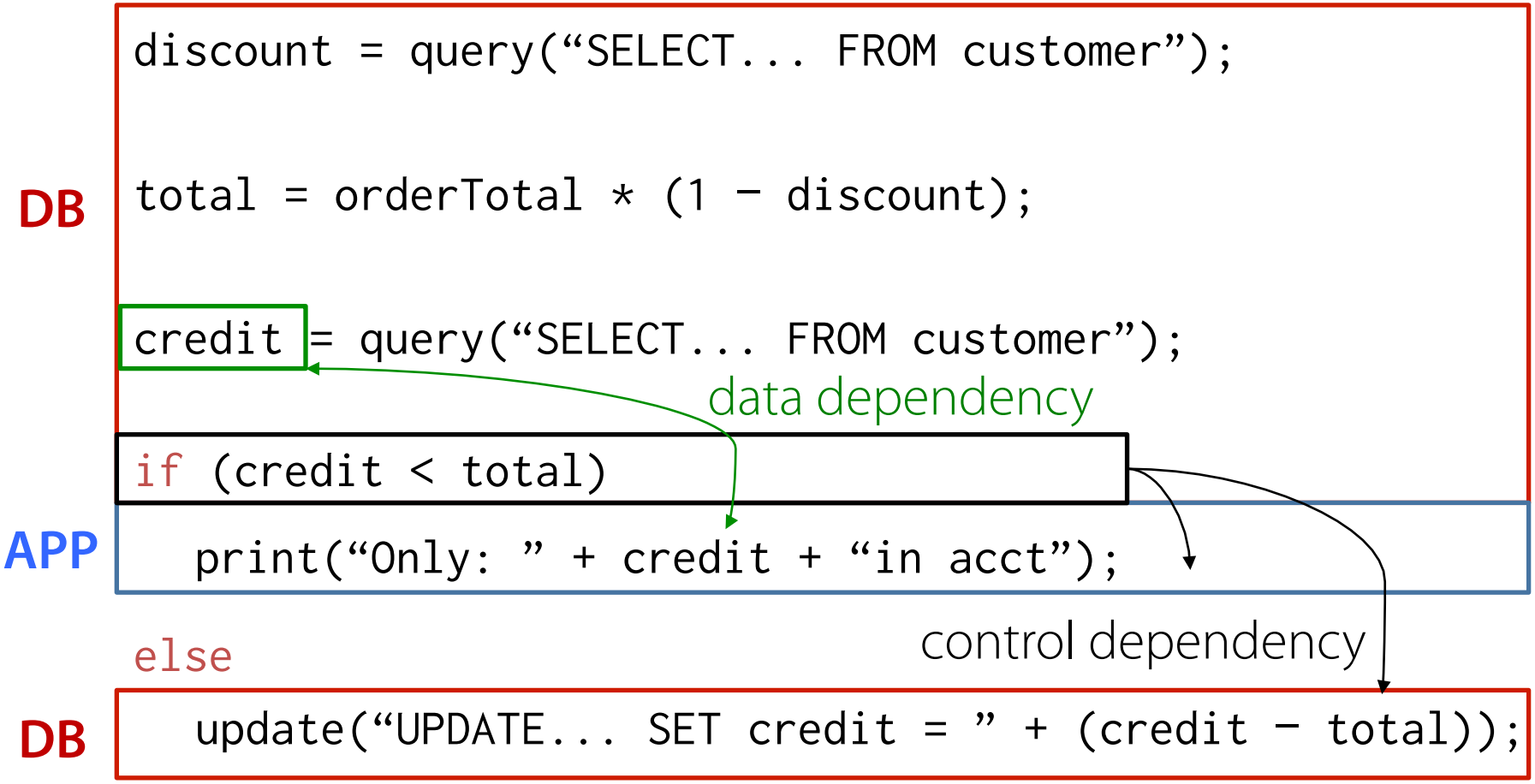
APP

```
  print("Only: " + credit + "in acct");
```

```
else
```

DB

```
  update("UPDATE... SET credit = " + (credit - total));
```


DB
discount = query("SELECT... FROM customer");
total = orderTotal * (1 - discount);
credit = query("SELECT... FROM customer");
data dependency
if (credit < total)
APP
print("Only: " + credit + "in acct");
control dependency
else
DB
update("UPDATE... SET credit = " + (credit - total));

DB

```
discount = query("SELECT... FROM customer");

total = orderTotal * (1 - discount);

credit = query("SELECT... FROM customer");
```

data dependency

APP

```
if (credit < total)
   print("Only: " + credit + "in acct");
```

control dependency

DB

```
else
   update("UPDATE... SET credit = " + (credit - total));
```

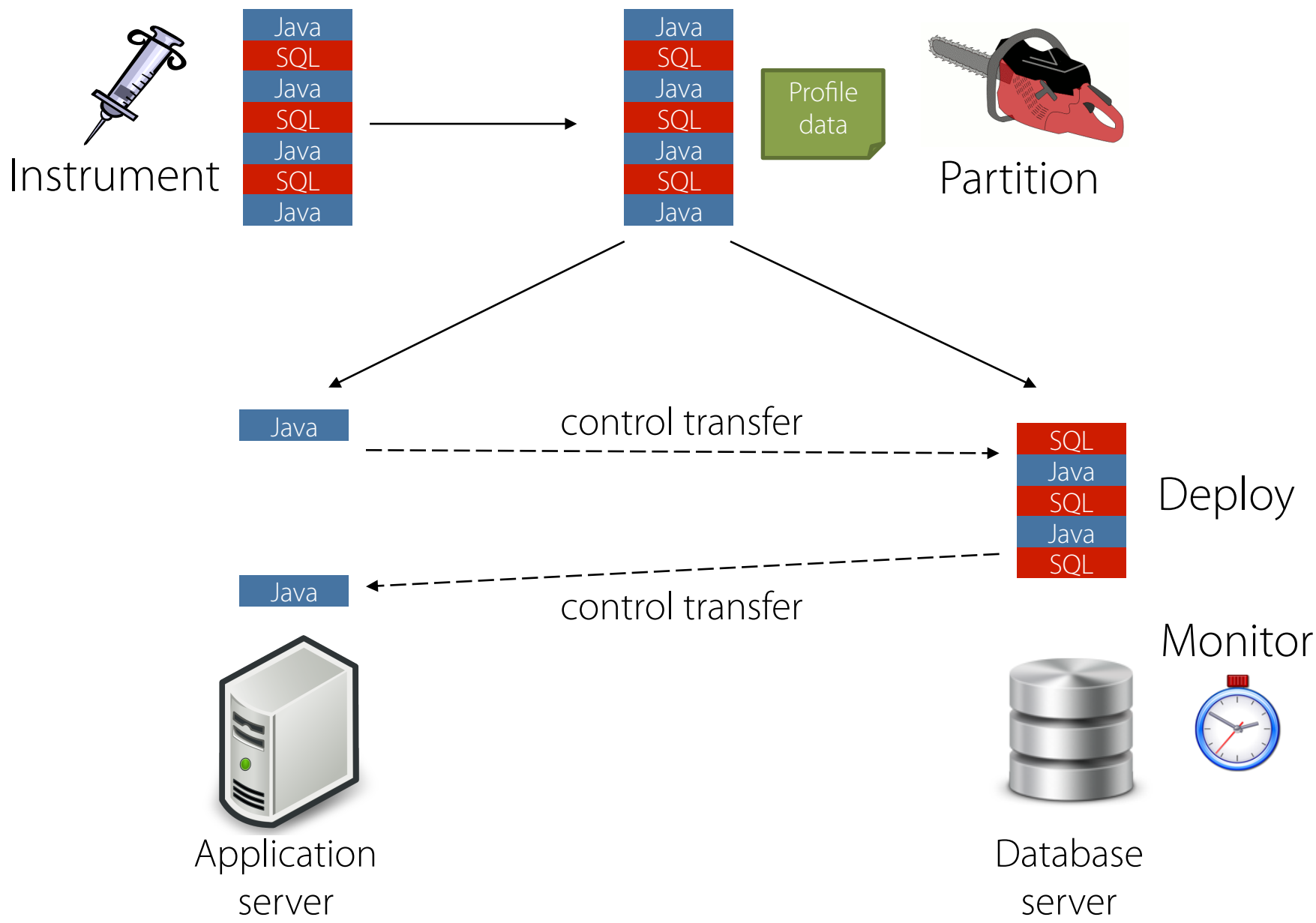
Pyxis

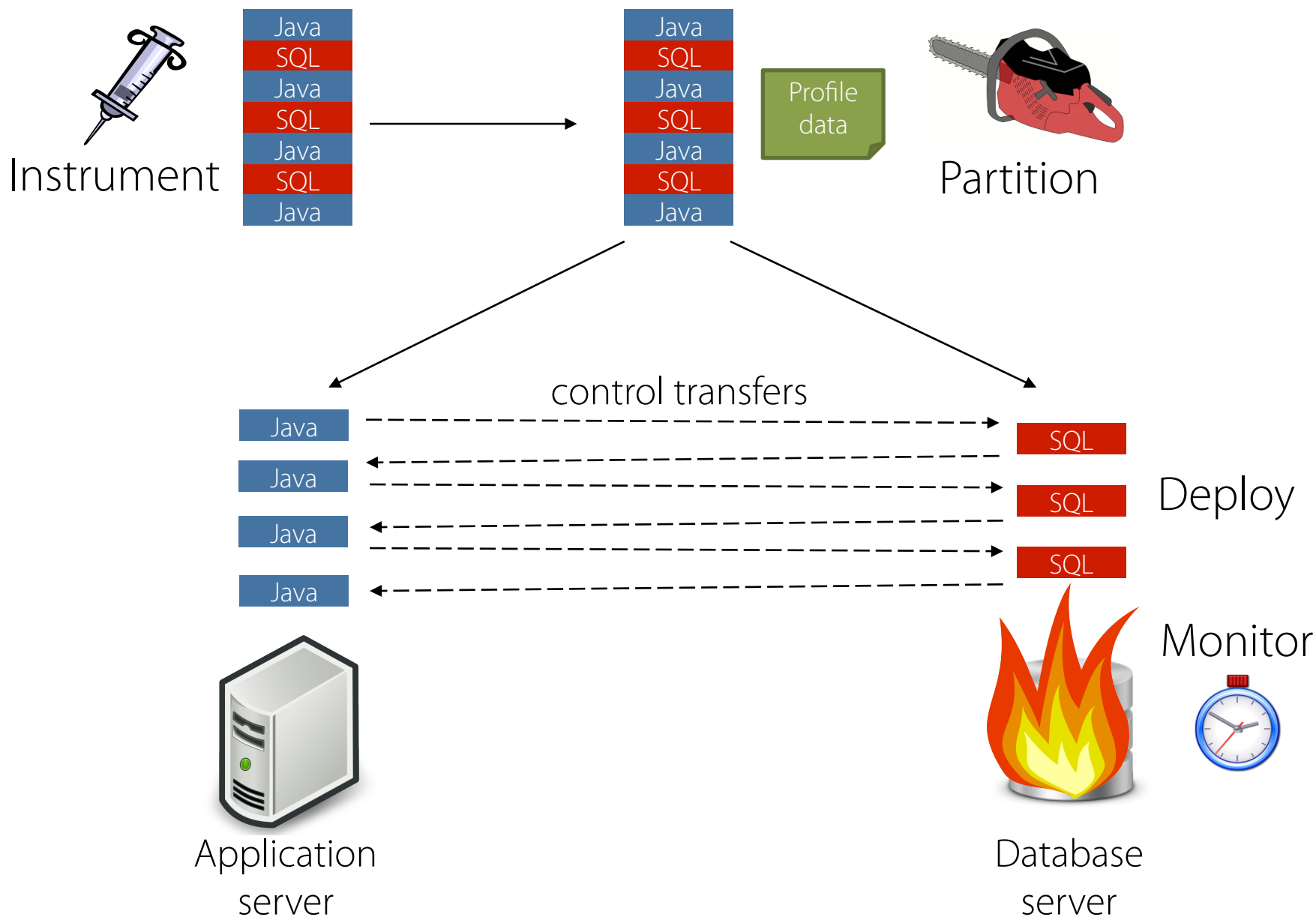
Automatically pushes code to the database

Adaptively controls the amount of code pushed for optimal performance

Automatic Partitioning of Database Applications
VLDB 12

Speeding up Database Applications with Pyxis
SIGMOD 13

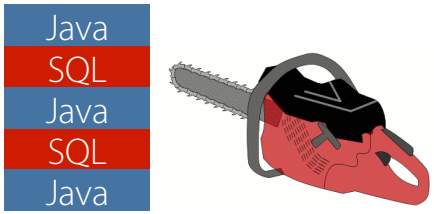

Instrument
Java
SQL
Java
SQL
Java
SQL
Java
Java
SQL
Java
SQL
Java
SQL
Java
Profile data
Partition
Java
control transfer
SQL
Java
SQL
Java
SQL
Deploy
Java
control transfer
Monitor
Application server
Database server


Instrument
Java
SQL
Java
SQL
Java
SQL
Java
Java
SQL
Java
SQL
Java
SQL
Java
Profile data
Partition
control transfers
Java
Java
Java
Java
SQL
SQL
SQL
Deploy
Monitor
Application server
Database server

Execution counts

```
100   discount = query("SELECT... FROM customer");
100   total = orderTotal * (1 - discount);
100   credit = query("SELECT... FROM customer");
100   if (credit < total)
 25     print("Only: " + credit + "in acct");
      else
 75     update("UPDATE... SET credit = " +
                (credit - total));
```

Program dependence graph

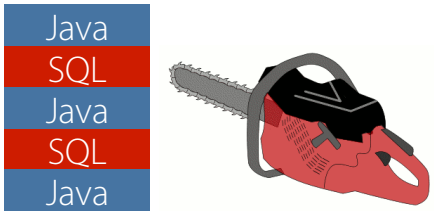


Partition

```
discount = query("SELECT... FROM customer");
total = orderTotal * (1 - discount);
credit = query("SELECT... FROM customer");
credit < total
print("Only: " + credit + "in acct");
update("UPDATE... SET credit = " + (credit - total));
```

→ Control dep

Program dependence graph



Partition

```
discount = query("SELECT... FROM customer");
```

discount

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

```
credit = query("SELECT... FROM customer");
```

```
credit < total
```

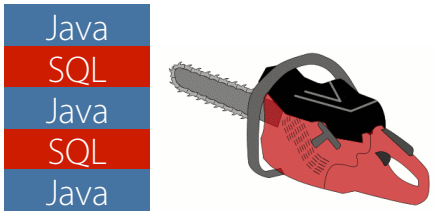
```
print("Only: " + credit + "in acct");
```

```
update("UPDATE... SET credit = " + (credit - total));
```

Control dep

Data dep

Program dependence graph



Partition

```
discount = query("SELECT... FROM customer");
```

discount

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

total

```
credit = query("SELECT... FROM customer");
```

credit

credit

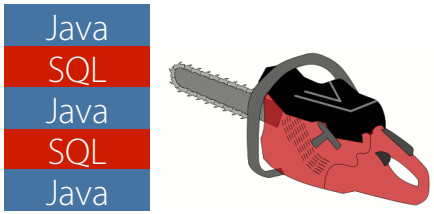
```
credit < total
```

```
print("Only: " + credit + "in acct");
```

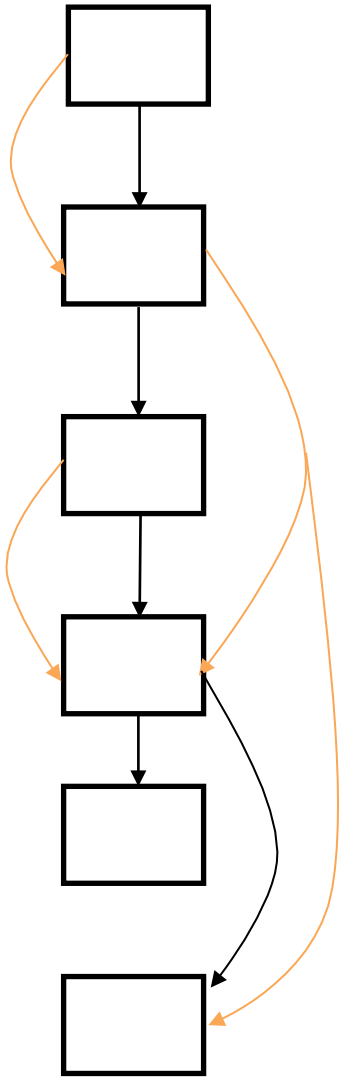
```
update("UPDATE... SET credit = " + (credit - total));
```

Control dep

Data dep



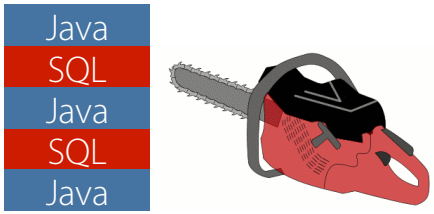
Partition



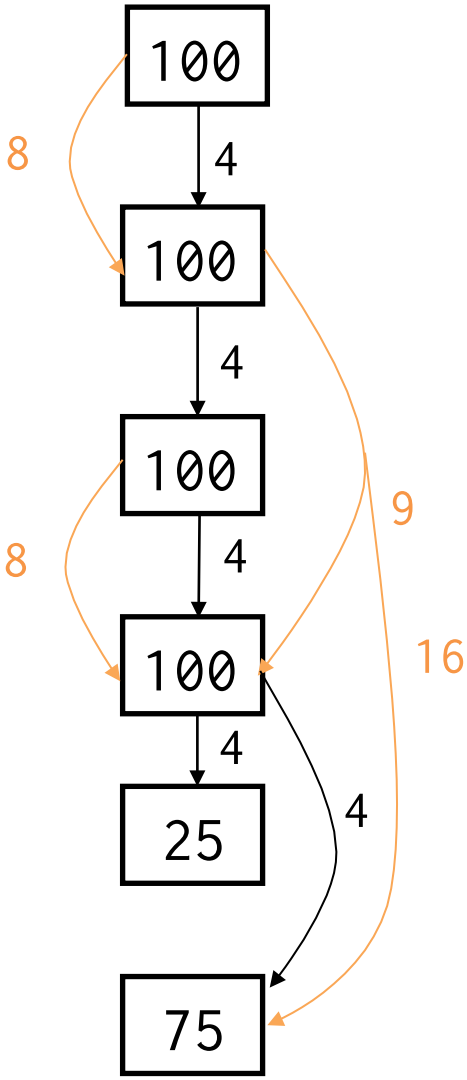
Profile data

Control dep

Data dep



Partition



Profile data

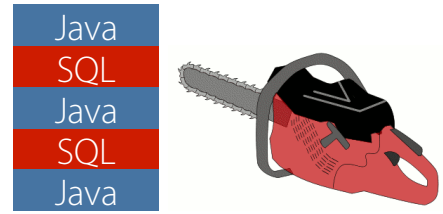
Control dep

Data dep

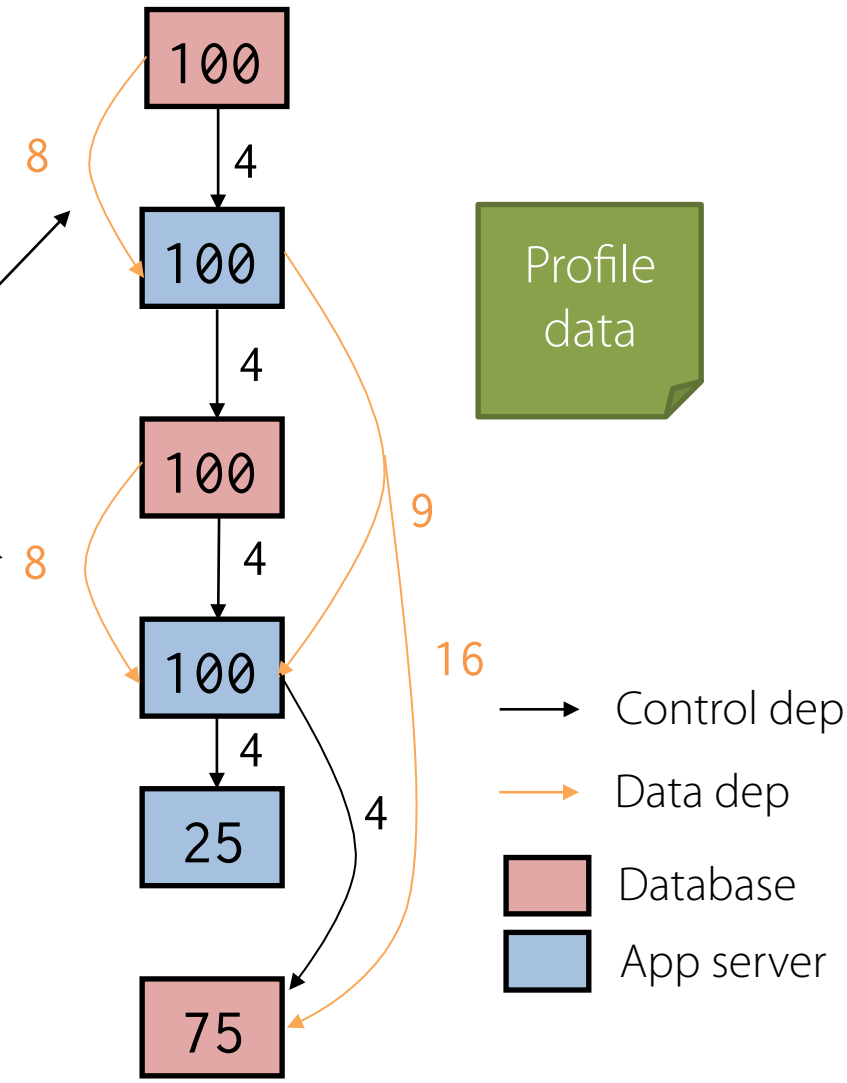
Integer Linear Program

Minimize:
cut edge weights

Subject to:
resources on DB
= 100%



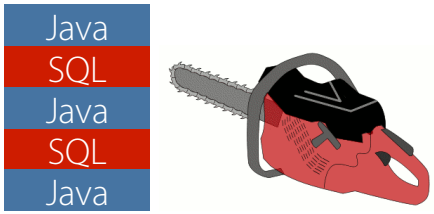
Partition



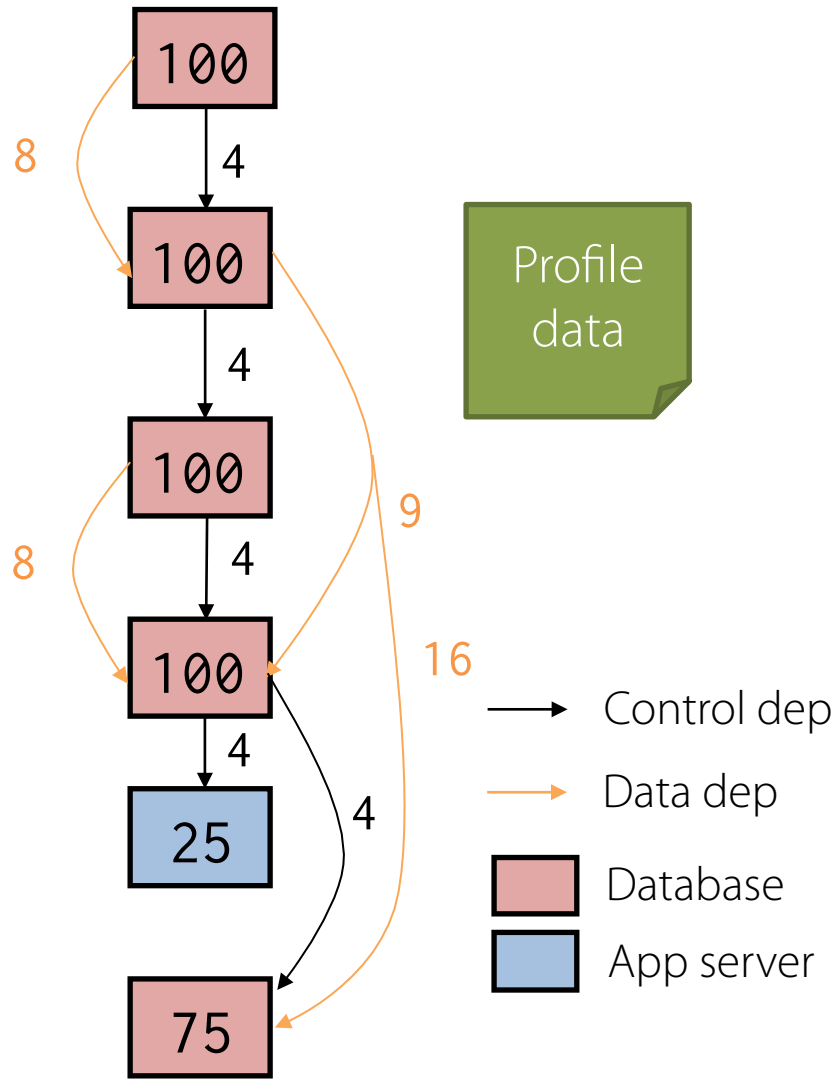
Integer Linear Program

Minimize:
cut edge weights

Subject to:
resources on DB
= 100%



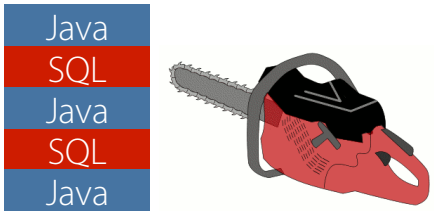
Partition



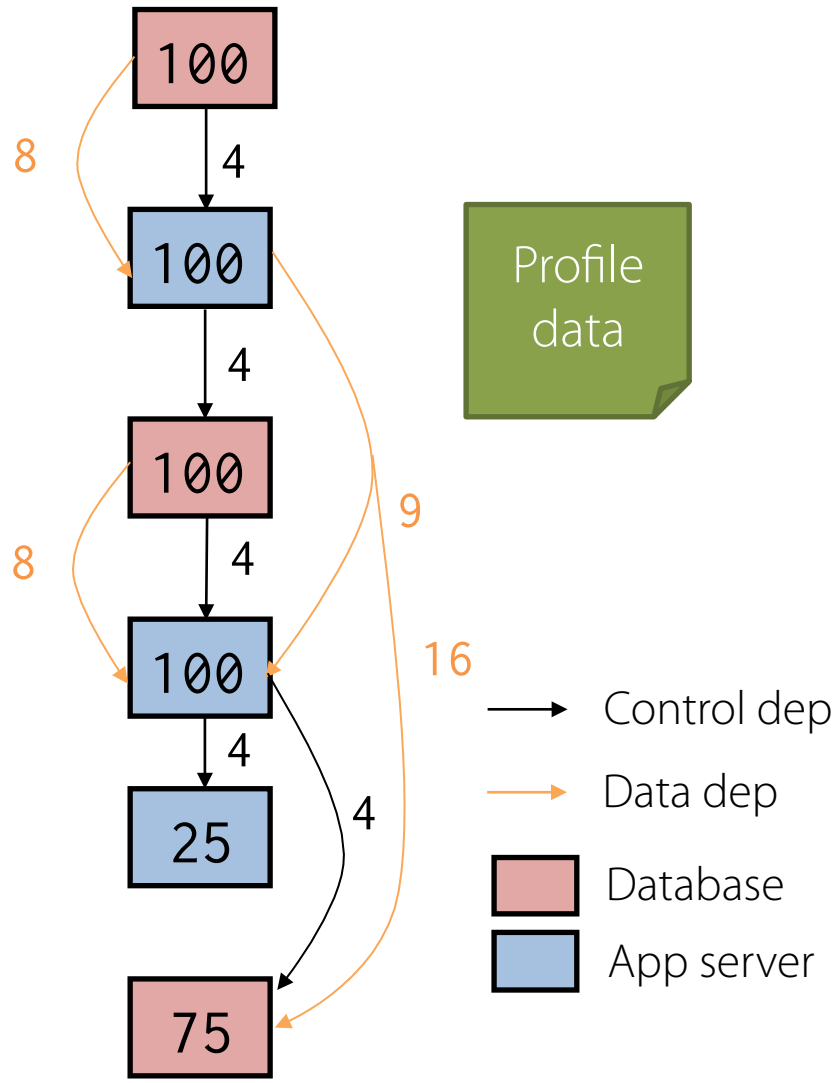
Integer Linear Program

Minimize:
cut edge weights

Subject to:
resources on DB
= 25%



Partition



Experiments

TPC-C Java implementation

20 concurrent users issuing transactions

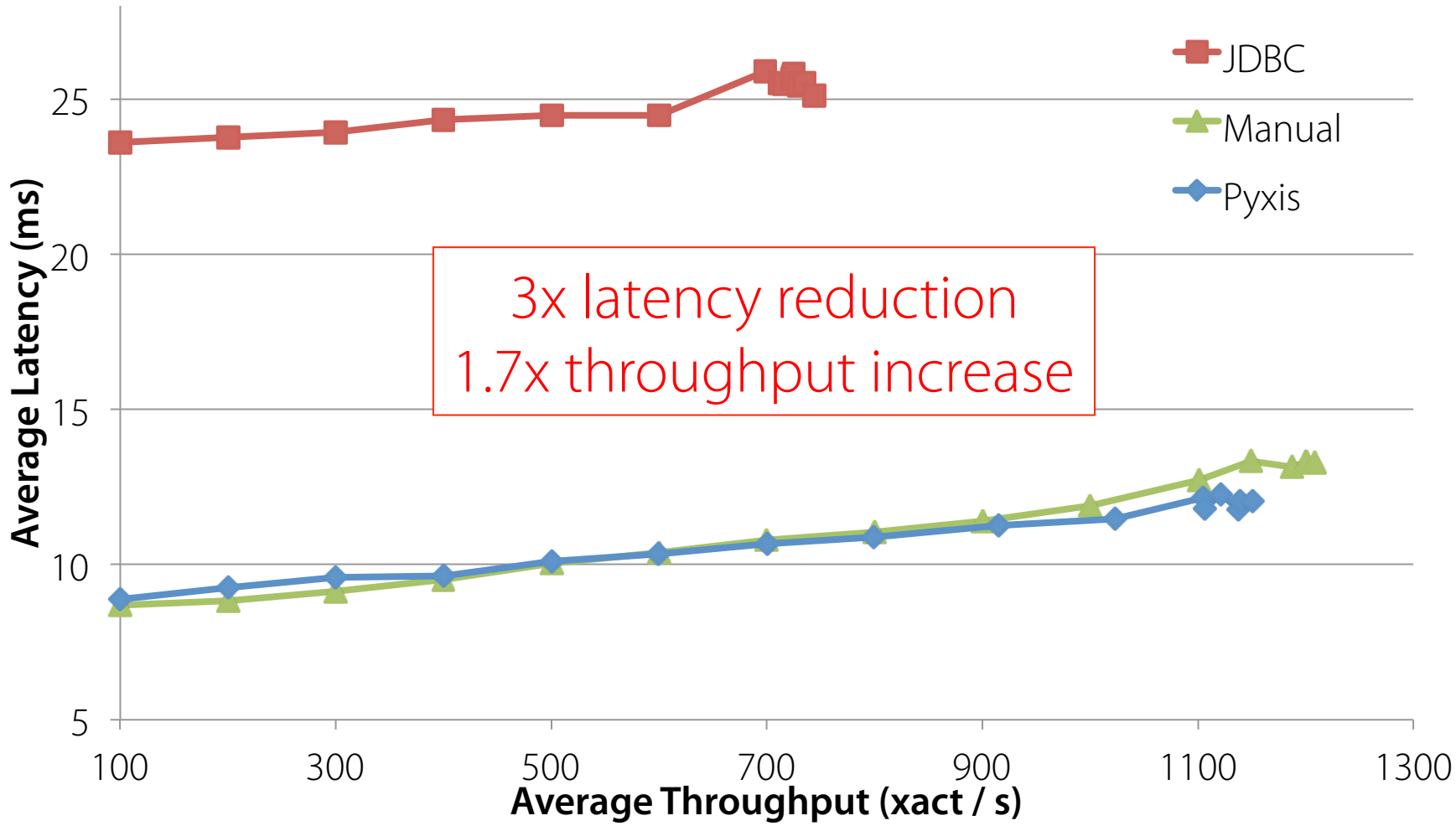
16 cores DB, 0.5ms latency between servers

Compared against two implementations

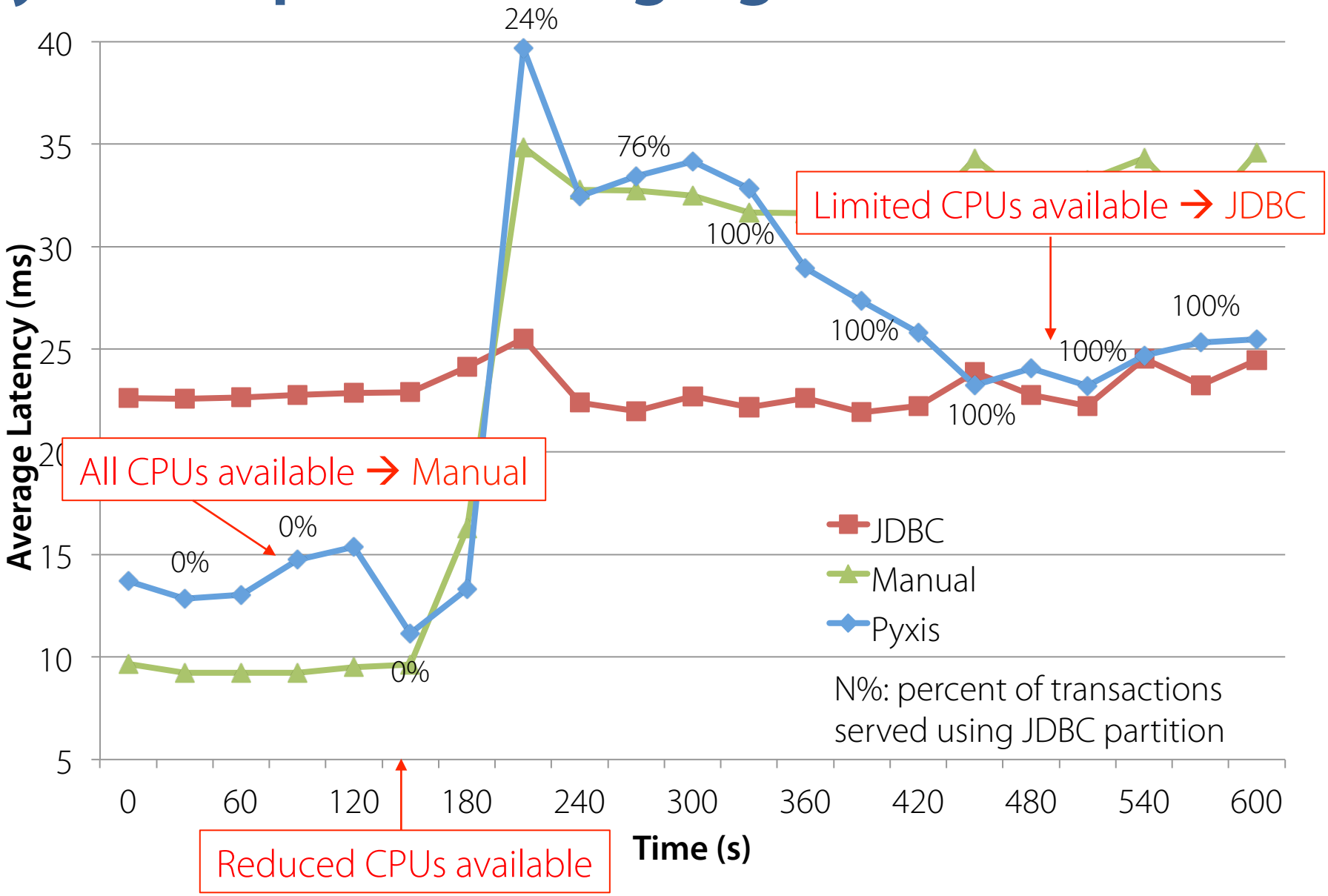
JDBC: all code on app server except for JDBC calls

Manual: hand-written "stored procedurized" version where most code is on the database server

Pyxis performs like manual when CPUs are available



Pyxis adapts to changing environments




Achievement unlocked
Can move computation for optimal performance

QUERY By SYNTHESIS

PLDI 13

PYXIS

VLDB 12
SIGMOD 13

Achievement unlocked
Can translate code representations

Achievement unlocked
Can build new classes of compilers

Achievement unlocked
Can move computation for optimal performance

SLOTH

3x latency reduction
2x throughput increase

Achievement unlocked
Can batch queries to reduce roundtrips

Sloth: Being Lazy is a Virtue (When Issuing Database Queries)
POPL 14, SIGMOD 14

Related Work

[Grust 09] FERRY: database-supported program execution

[Iu 10] HadoopToSQL: a MapReduce query optimizer

[Meijer 06] LINQ: Reconciling objects, relations and XML in the .NET framework

[Srivastava et al 09] Program verification using templates over predicate abstraction

[Tatlock 10] Bringing extensibility to verified compilers

[Wiedermann 07] Interprocedural query extraction for transparent persistence

---

[Chong 07] Secure web applications via automatic partitioning

[Chun 11] CloneCloud: elastic execution between mobile device and cloud

[Cuervo 10] Maui: making smartphones last longer with code offload

[Newton 09] Wishbone: Profile-based partitioning for sensornet applications

[Ra 11] Odessa: enabling interactive perception applications on mobile devices

[Yang 07] A unified platform for data driven web applications with automatic client-server partitioning

Program logic

Framework

spring

django

Runtime

Java

python

OS & Networking

Persistent storage

Program as specification

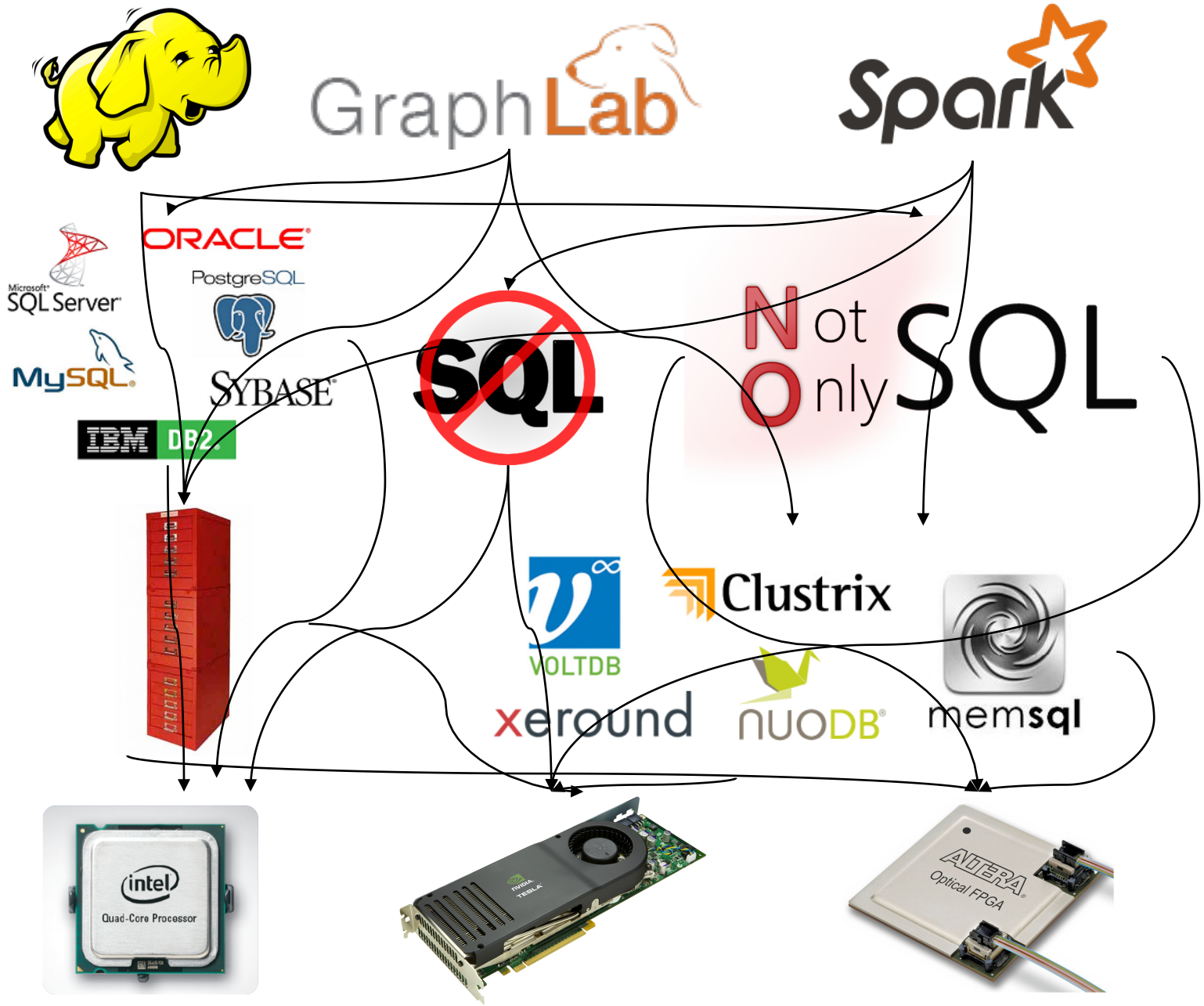
Program analysis	Program synthesis	Verification
Data management	Systems	

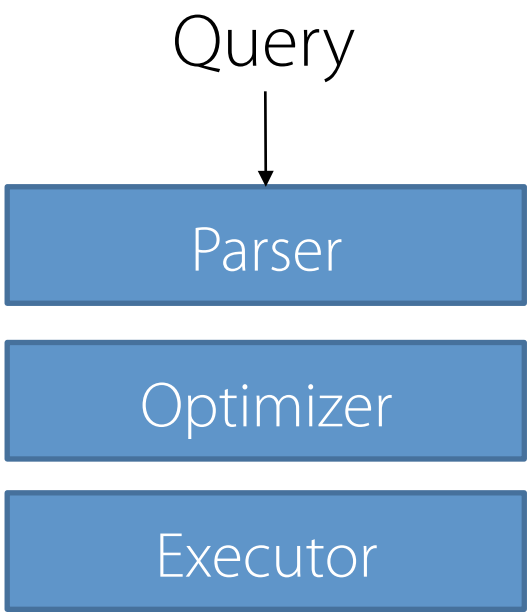
Multiple orders-of-magnitude performance improvement

RETHINK THE SOFTWARE STACK

StatusQuo: Making Familiar Abstractions Perform Using Program Analysis

CIDR 13 [Best Paper Award]

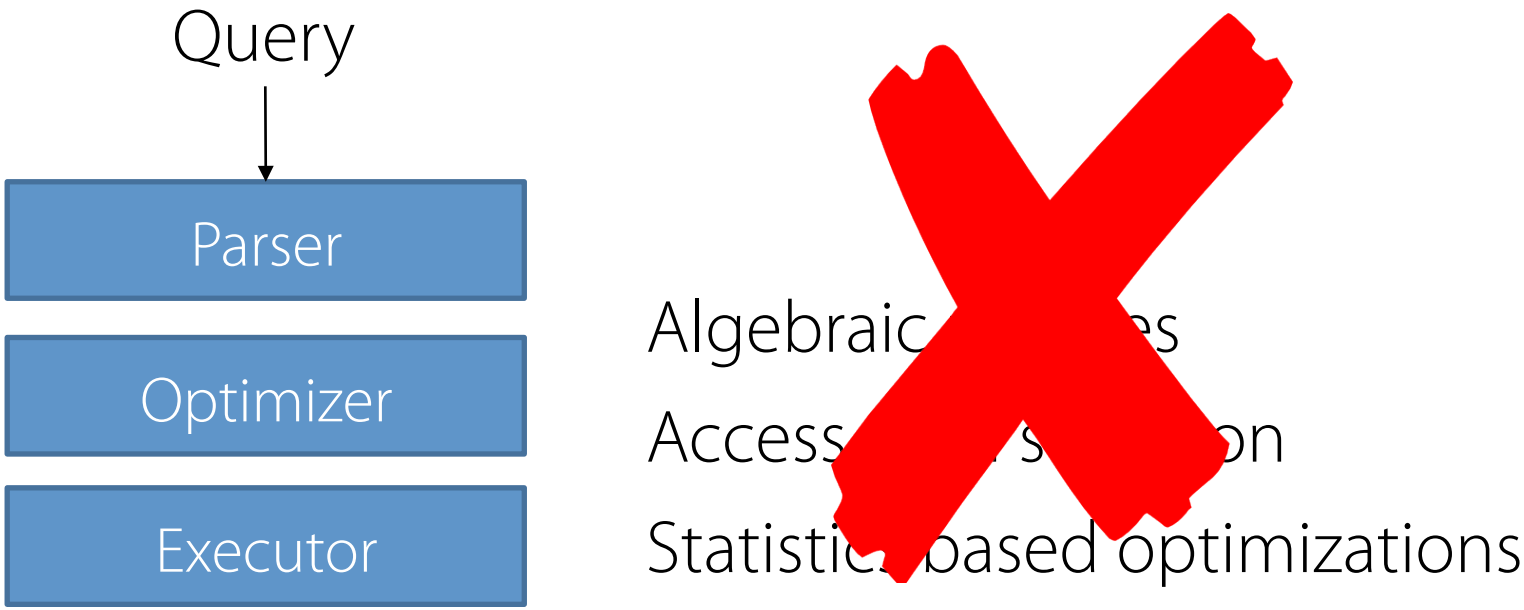

GraphLab
Spark
ORACLE
PostgreSQL
Microsoft SQL Server
MySQL
SYBASE
IBM DB2
SQL
Not Only SQL
VOLTDB
Clustrix
xeround
NUODB
memsql
intel
Quad-Core Processor
ALTERA
Optical FPGA

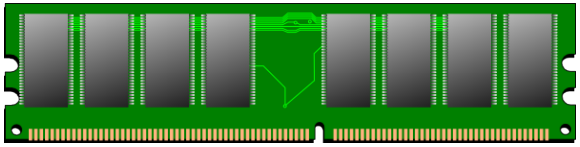


Algebraic rewrites

Access path selection

Statistics based optimizations


Query
Parser
Optimizer
Executor
Algebraic
es
Access
s
on
Statistic based optimizations




SSD
SLC
GREEN HOUSE

Application

↓

Query

↓

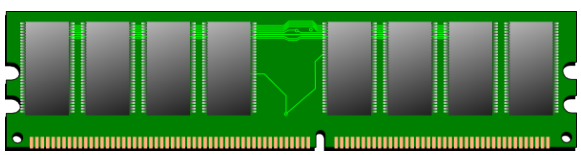
Application specific storage

Systems

Synthesis

Analysis

Languages



High-level storage specification

Rethinking the Application-Database Interface

QUERY By SYNTHESIS

PYXIS

SLOTH

Alvin Cheung

akcheung@cs.washington.edu