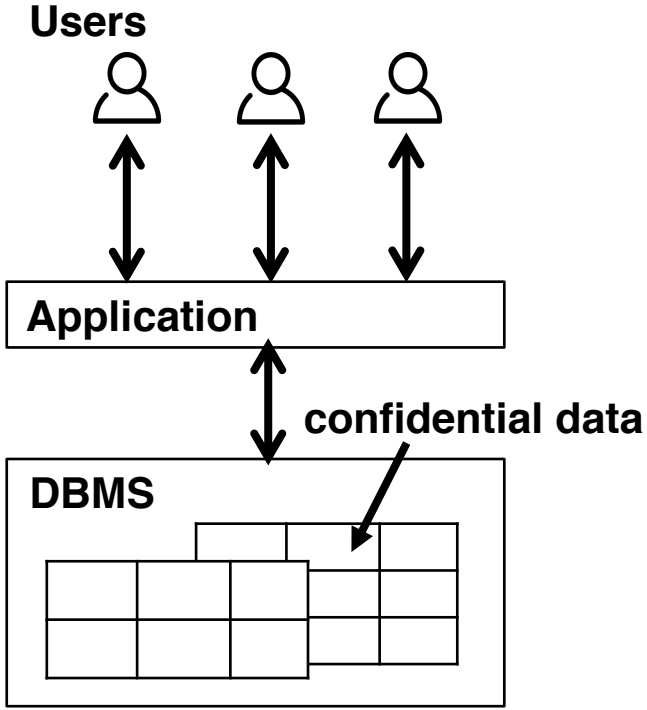
Qapla: Policy compliance for database-backed systems

Aastha Mehta[1], Eslam Elnikety[1], Katura Harvey[1,2], Deepak Garg[1], Peter Druschel[1]

[1]Max Planck Institute for Software Systems (MPI-SWS)

[2]University Of Maryland, College Park

Web applications store confidential data in DBMS



Application architecture

Healthcare systems
- patient records

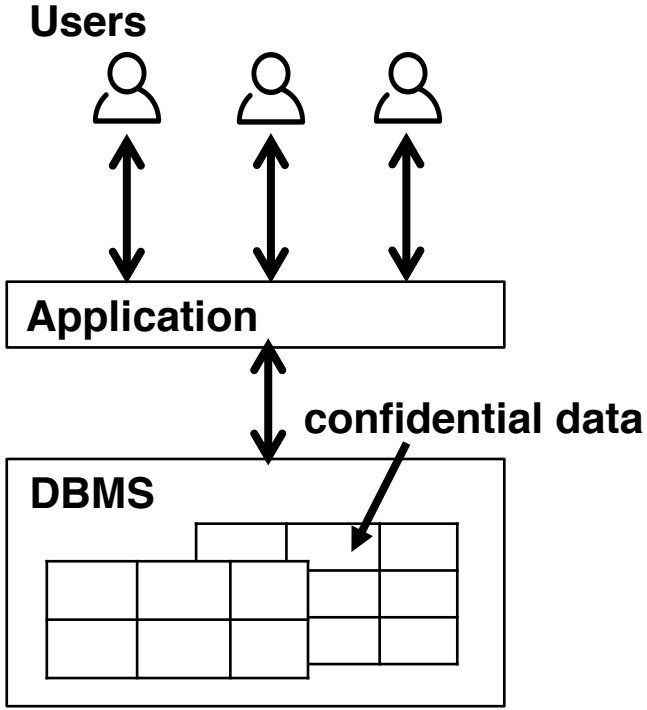
Personnel management systems
- salaries, ages

Conference management systems
- submissions, reviews

Requirement: applications must comply with data access policies

Challenge: specifying and enforcing complex data access policies is non-trivial

Web applications store confidential data in DBMS



Application architecture

Healthcare systems
- patient records

Personnel management systems
- salaries, ages

this talk: HotCRP

Conference management systems
- submissions, reviews

Requirement: applications must comply with data access policies

Challenge: specifying and enforcing complex data access policies is non-trivial

Current approach: enforcing policy in application code

Example from HotCRP (simplified for illustration)

Current application logic

review.php:

```
loadRows();

loadRows() {
  $getinfo[“paperId”] = $_REQUEST[“paperId”];
  if ($Me->isPC && $Conf->timePCReviewPreferences() || $Me->privChair)
    $getinfo[“reviewerPreference”] = true;
  $prow = $Conf->paperRow($getinfo, $Me);
  $rrows = $Conf->reviewRow($getinfo, $prow);
  if (!can_view_paper($prow))
    return null;
  if (!$Me->can_view_review($prow, $rrow) &&
      !$Me->privChair)
    return null;
  return $prow;
}

if (!$Me->can_view_review($prow
  report_error();

echo “<html>$prow</html>”;
```

load reviews

checks based on user roles, conference phase

fetch reviews for `paperId`

checks based on user roles, conflicts, conference phase

more checks ...

Policy checks inlined throughout application code

Application code

```
get_review($paper, $user) {
  $result = $db->query("SEL
       Review.paperId=".$pa

  $rrows = $result->fetchOb

  foreach ($rrows as $row)
    if (can_view_review($ro
       show($row->review);
  }
}
```

```
get_comments($reviewId, $user) {
  $result = $db->query("SELECT Reviews.* FROM Reviews
       WHERE Review.reviewId=".$reviewId
  $row = $result->fetchRow();
  if (!can_view_review($row, $user))
    return "";

  $result = $db->query("SELECT * FROM Co
  $rows = $result->fetchRows();
  foreach($rrows as $row) {
    if (can_view_comment($row, $user)) {
      comments[] = $row;
    }
  }
  return $comments;
}
```

```
get_pc_members($paper, $user) {
  $result = $db->query("SELECT name FROM Contacts
  WHERE roles=PC|CHAIR);

  $rrows = $result->fetchObject();

  foreach ($rrows as $row) {
    if (can_view_contact($row, $user))
      show($row->name);
  }
}
```

- Must update checks in several code paths as application or policies evolve
- Easy to miss or implement incorrect checks

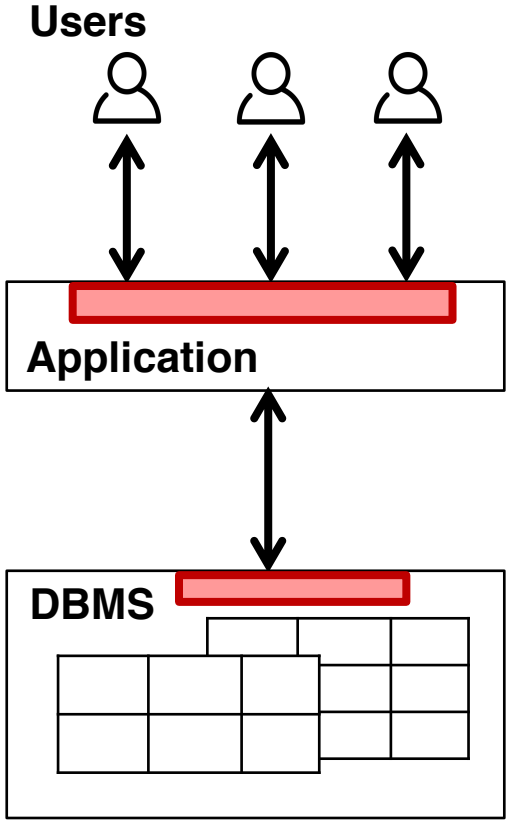
Examples of data leaks in HotCRP

HotCRP v2.58 23.Mar.2013
More **information leak plugging: explicit search for review fields that should be hidden from authors, and review rounds**. Reported by John Heidemann.

HotCRP v2.59 14.Jun.2013
Bug fix: "Monitor external reviews" works. Reported by Peter Sewell.
Information leak fixes: During response periods, don't notify authors of changes in PC-only fields. Don't allow searches on review rounds for conflicted papers. Don't show accept status via "Accepted papers" searches. Reported by Nickolai Zeldovich and Jeff Mogul.

Source: http://read.seas.harvard.edu/~kohler/hotcrp/news.html

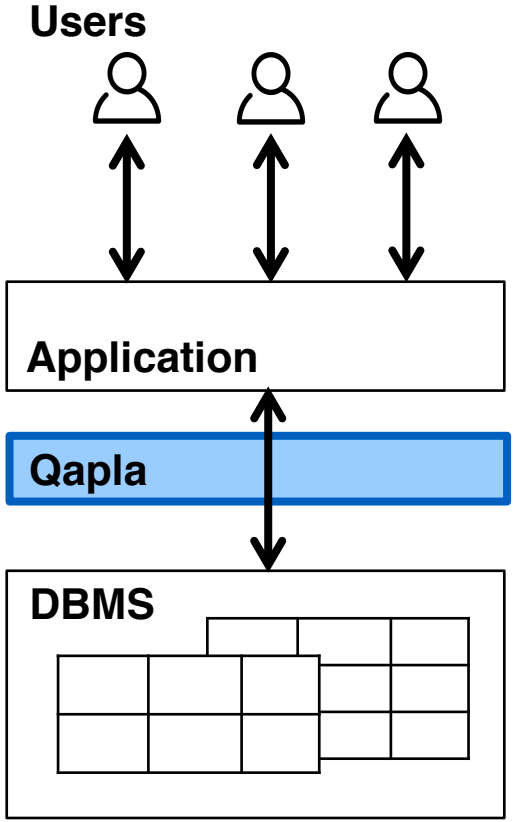
Limitations of existing approaches



Enforcing policy in application code:
Bugs in application code may lead to policy violations and cause data leaks

Using DBMS access control support:
Cannot support all policies without changes to DB schema or application queries

Our approach



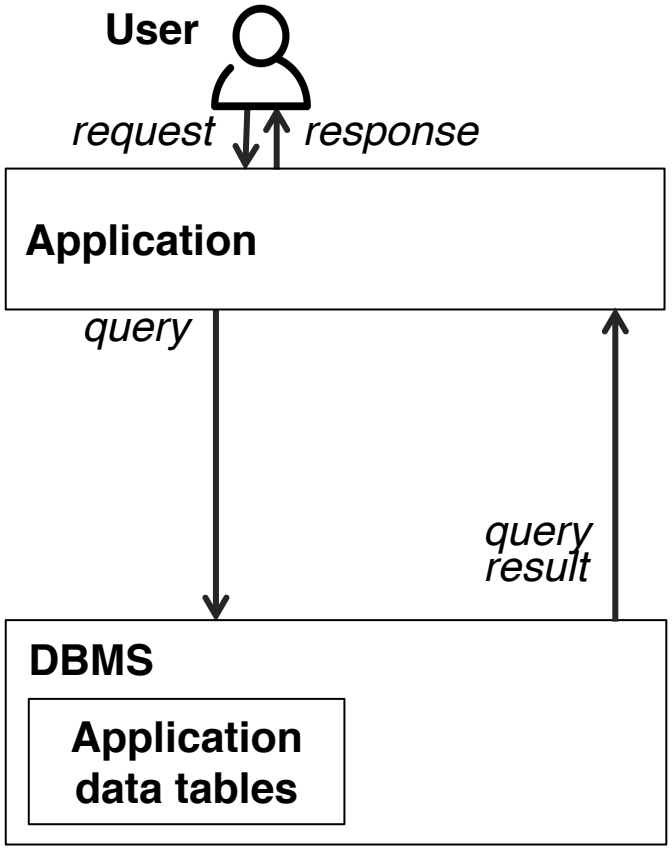
Goals:

- Separate policy compliance from application code and DBMS
- Support complex, fine-grained data access policies
- Add only moderate performance overheads for end users

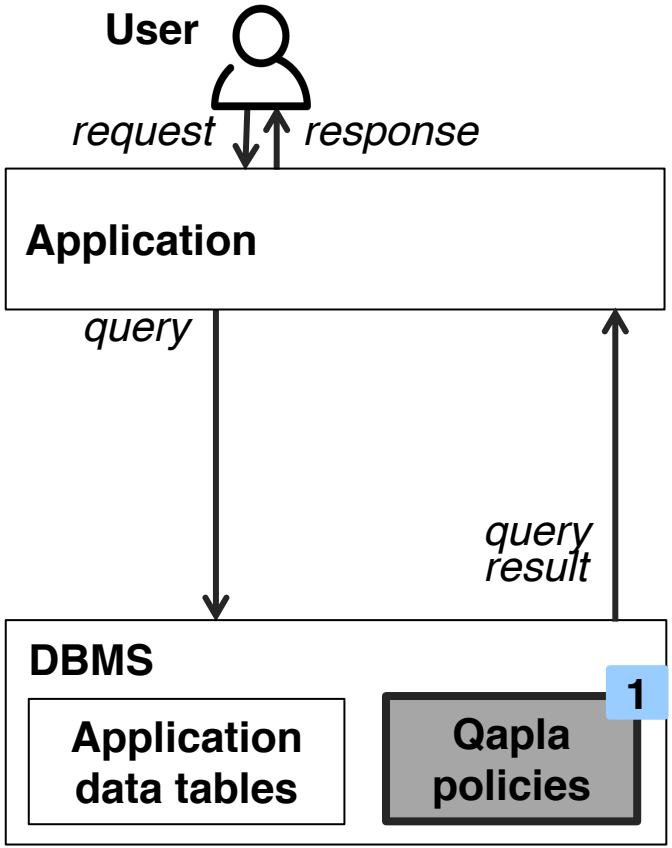
Outline

- Policy compliance today
  - Policy checks in application
  - DBMS access control
- Qapla
  - Design
  - Policy specification
  - Policy enforcement
- Evaluation

Qapla: compliance independent of application code and DBMS

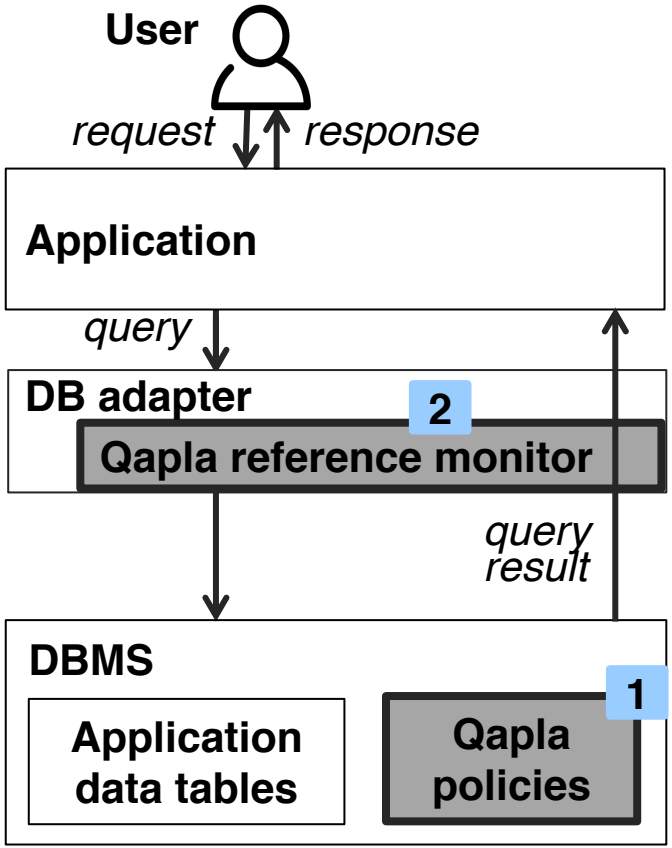


Qapla: compliance independent of application code and DBMS



1 **Declarative policies** associated with the DB schema, stored in the DBMS

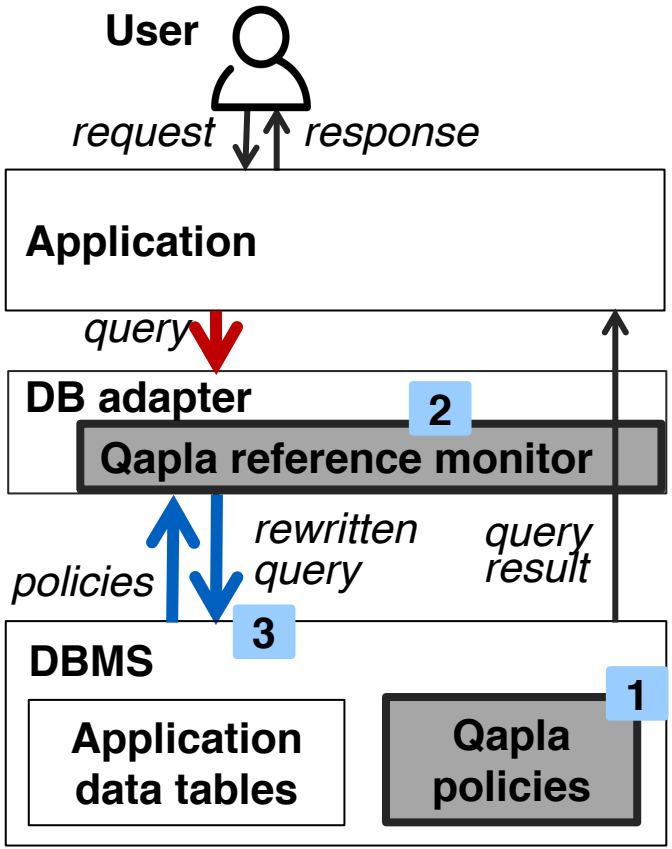
Qapla: compliance independent of application code and DBMS



1 **Declarative policies** associated with the DB schema, stored in the DBMS

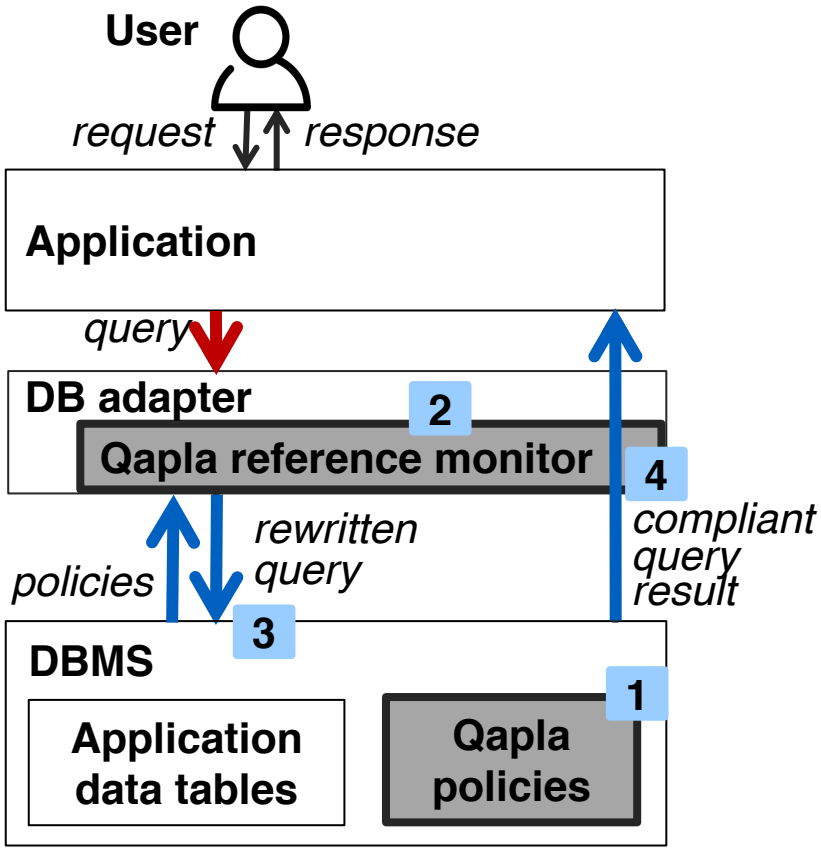
2 Policies enforced by a **reference monitor** integrated with a DB adapter

Qapla: compliance independent of application code and DBMS



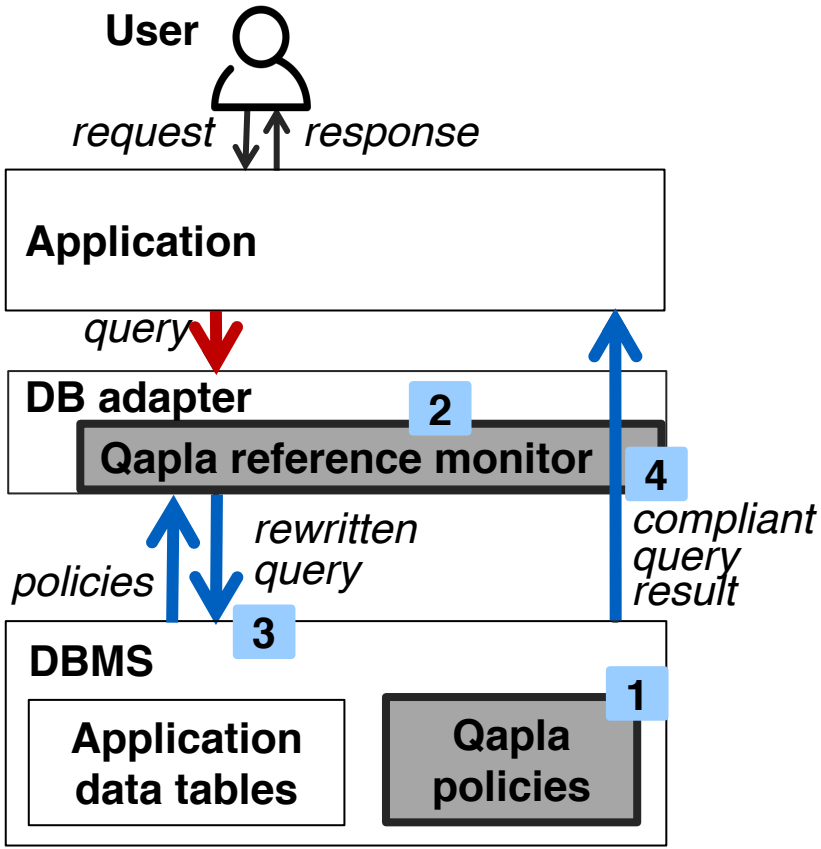
1. **Declarative policies** associated with the DB schema, stored in the DBMS
2. Policies enforced by a **reference monitor** integrated with a DB adapter
3. Reference monitor **rewrites query** with applicable policies

Qapla: compliance independent of application code and DBMS



1. **Declarative policies** associated with the DB schema, stored in the DBMS
2. Policies enforced by a **reference monitor** integrated with a DB adapter
3. Reference monitor **rewrites query** with applicable policies
4. Forwards **compliant query results** to the application

Qapla: compliance independent of application code and DBMS



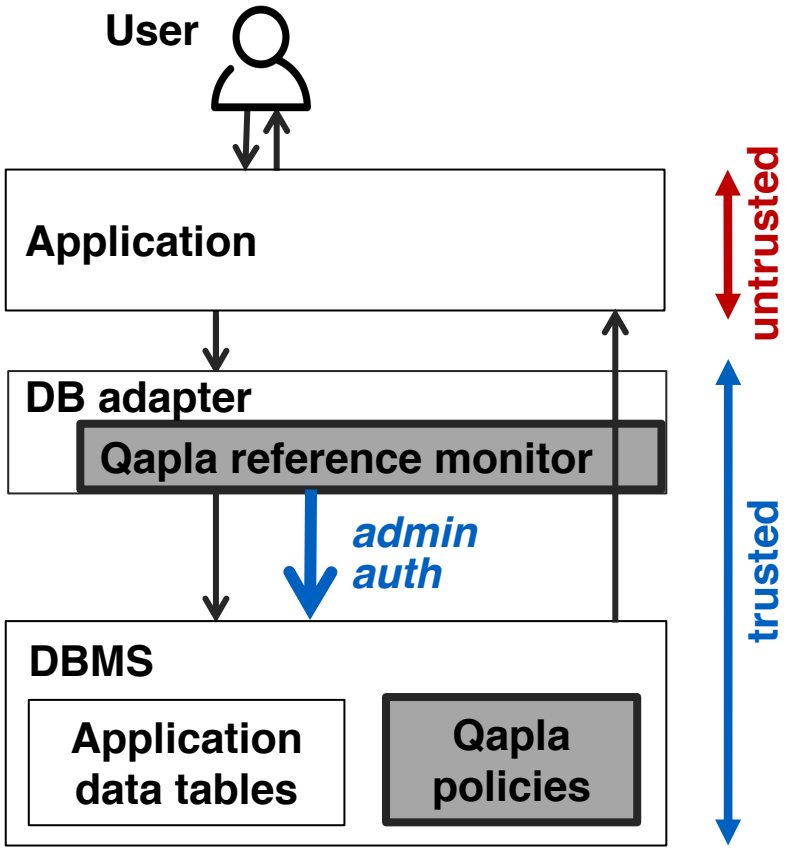
1. **Declarative policies** associated with the DB schema, stored in the DBMS
2. Policies enforced by a **reference monitor** integrated with a DB adapter
3. Reference monitor **rewrites query** with applicable policies
4. Forwards **compliant query results** to the application

Compliant application queries → **same results**

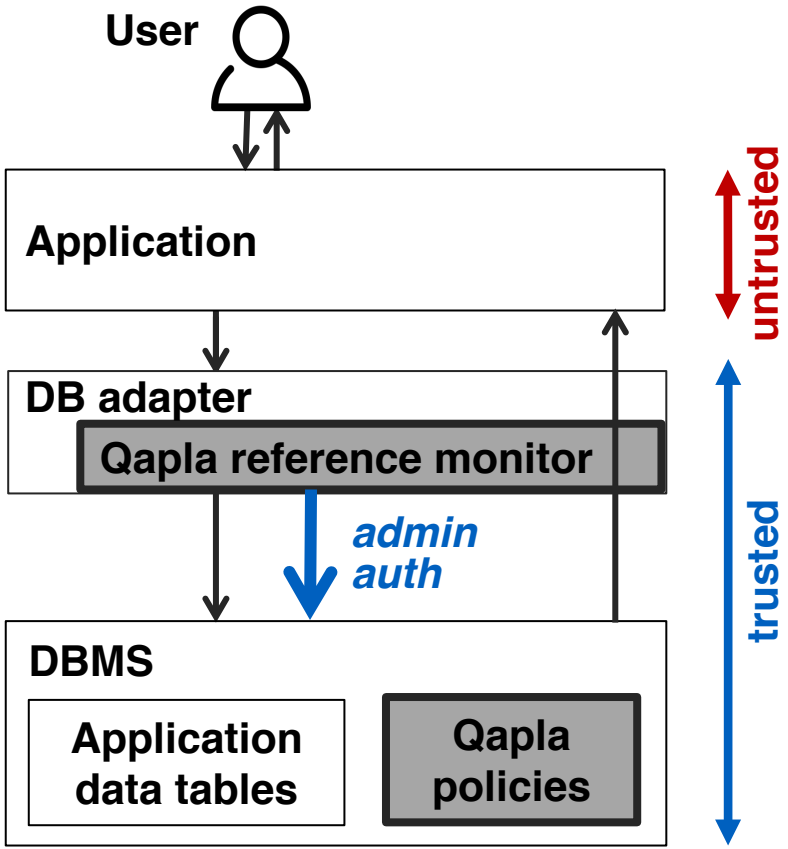
Non-compliant queries → **fewer results**

Threat model

Our goal: prevent inadvertent data leaks due to application bugs



Threat model



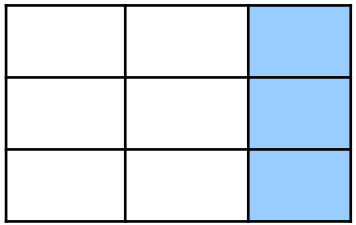
Our goal: prevent inadvertent data leaks due to application bugs

Prototype Limitations

- ! application does not circumvent the reference monitor
  - can use software fault isolation, process or address space isolation
- ! application sends correct user identity to reference monitor
  - user can directly authenticate with the reference monitor

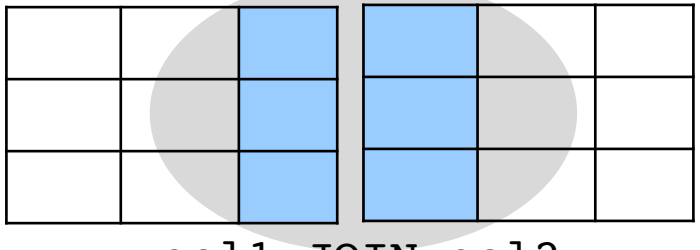
Policies on DB queries

1. single column policies



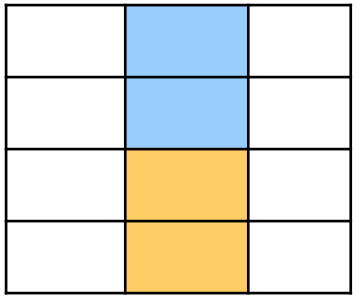
```
SELECT col
```

2. link policy (on joins, filters)



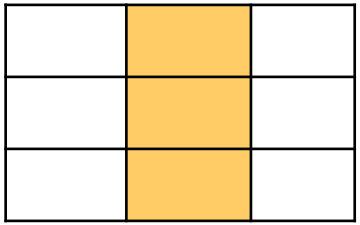
```
col1 JOIN col2
```

3. policies on aggregate, group by



```
SELECT COUNT(col)
GROUP BY col
```

4. policies on user defined functions (UDF)



in the paper

```
SELECT UDF(col)
```

Policy on queries that read single column

Authors can see their reviews only after decision notification

Reviews

paperId	contactId	review
123	9	

Papers

paperId	author	outcome
123	Alice	

Policy on queries that read single column

Authors can see their reviews only after decision notification

Reviews

paperId	contactId	review
123	9	

Papers

paperId	author	outcome
123	Alice	

Reviews.review :-

Policy on queries that read single column

Authors can see their reviews only after decision notification

Reviews

paperId	contactId	review
123	9	

Papers

paperId	author	outcome
123	Alice	

```
Reviews.review :-
CURRENT_DATE() >= 22-06-2017                    /* date after decision notification */
AND
EXISTS(SELECT 1 FROM Papers
WHERE paperId = Reviews.paperId                 /* user is an author of paper with
AND author = $user)                                paperId = Reviews.paperId */
```

Policy on queries that read single column

Authors can see their reviews only after decision notification

cell-level access control

Reviews

paperId	contactId	review
123	9	

Papers

paperId	author	outcome
123	Alice	

column-level condition

```
Reviews.review :-
CURRENT_DATE() >= 22-06-2017
AND
EXISTS(SELECT 1 FROM Papers
WHERE paperId = Reviews.paperId
AND author = $user)
```

/ date after decision notification */*

cell-level condition

/ user is an author of paper with paperId = Reviews.paperId */*

Policy on queries that link multiple columns

Authors can see PC **names** and **reviews** independently, but cannot **link** them

Reviews

paperId	contactId	review
123	9	

Contacts

contactId	name	role
9	Bob	PC

Papers

paperId	author	outcome
123	Alice	

Policy on queries that link multiple columns

Authors can see PC **names** and **reviews** independently, but cannot **link** them

{Reviews.review, Reviews.contactId, Contacts.name, Contacts.contactId} :-

Reviews

paperId	contactId	review
123	9	

Contacts

contactId	name	role
9	Bob	PC

Papers

paperId	author	outcome
123	Alice	

Policy on queries that link multiple columns

Authors can see PC **names** and **reviews** independently, but cannot **link** them

{Reviews.review, Reviews.contactId, Contacts.name, Contacts.contactId} :-

```
NOT EXISTS(SELECT 1 FROM
Papers WHERE author=$user)
```

author cannot access these columns together

Reviews

paperId	contactId	review
123	9	

Contacts

contactId	name	role
9	Bob	PC

Papers

paperId	author	outcome
123	Alice	

Policy on queries that link multiple columns

Authors can see PC **names** and **reviews** independently, but cannot **link** them

{Reviews.review, Reviews.contactId, Contacts.name, Contacts.contactId} :-

```
NOT EXISTS(SELECT 1 FROM
Papers WHERE author=$user)
```

author cannot access these columns together

authors can see their reviews only after notification

Reviews

paperId	contactId	review
123	9	

all users can see PC names

Contacts

contactId	name	role
9	Bob	PC

Papers

paperId	author	outcome
123	Alice	

Policy on aggregate queries

authors can see their outcome only after notification

Papers

paperId	author	outcome
		accept
123	Alice	accept
		reject

Policy on aggregate queries

Anyone can see the number of submitted and accepted papers after the decision notification

Papers

paperId	author	outcome
		accept
123	Alice	accept
		reject

{Papers.outcome**[COUNT, GROUP BY]**} :-

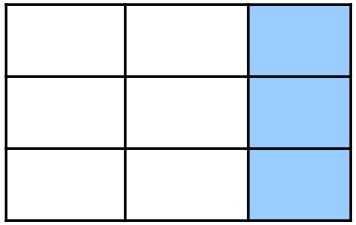
```
CURRENT_DATE() >= 22-06-2017
```

/ allow COUNT query with GROUP BY on outcome only after notification */*

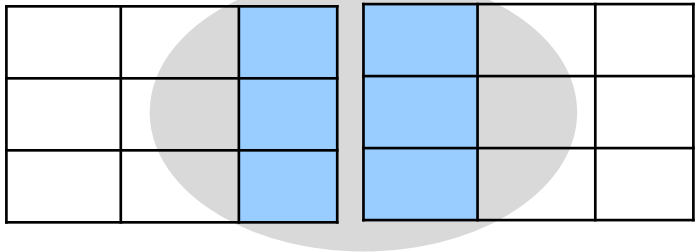
Summary: Qapla policy specification framework

Qapla policy
set of columns `:-` SQL `WHERE` clauses on tables

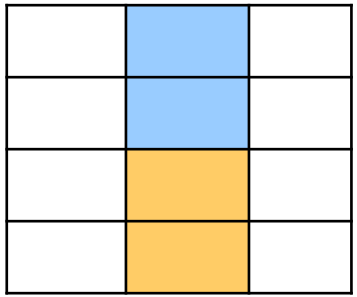
1. single column policies



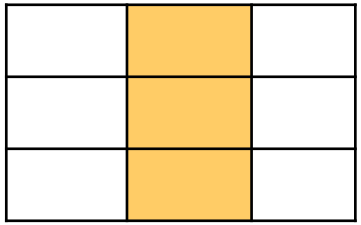
2. link policy (on joins, filters)



3. policies on aggregate, group by



4. policies on UDFs



Policy enforcement

T1

col1	col2

Policies:

{col1} :- P1
{col2} :- P2
{col1,col2} :- P3

Query on single column

```
Q: SELECT col1 FROM T1
```

Policies = {**P1**}

```
T1': SELECT * FROM T1
       WHERE P1
Qr: SELECT col1 FROM T1'
```

Query on multiple columns

```
Q: SELECT col1 FROM T1 WHERE
                col2 = expr1
```

Policies = {**P3**}

```
T1': SELECT * FROM T1
       WHERE P3
Qr: SELECT col1 FROM T1' WHERE
              col2 = expr1
```

Policy enforcement

T1

col1	col2

Policies:

```
{col1}      :- P1
{col2}      :- P2
{col1,col2} :- P3
```

Step1: Identify the policies applicable to the set of columns in the query

Query on multiple columns

```
Q: SELECT col1 FROM T1 WHERE
                col2 = expr1
```

Policies = {**P3**}

Step2: Replace each table with a **subquery** containing the applicable polices

```
T1': SELECT * FROM T1
       WHERE P3
Qr: SELECT col1 FROM T1' WHERE
           col2 = expr1
```

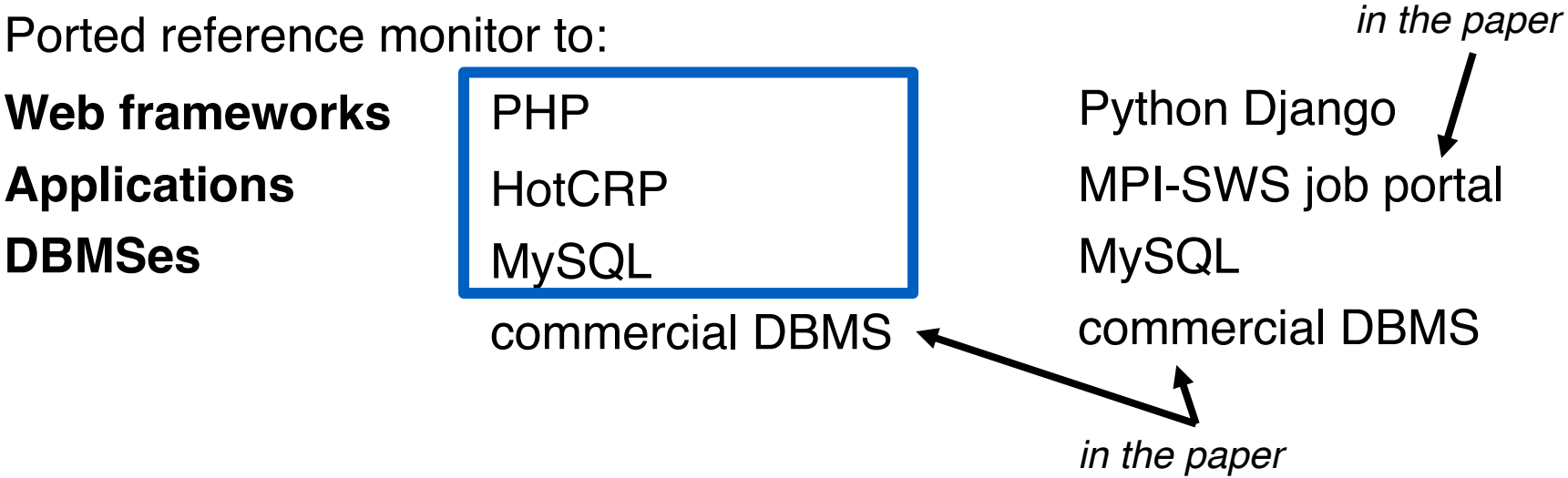
Outline

- Policy compliance today
  - Policy checks in application
  - DBMS access control
- Qapla
  - Design
  - Policy specification
  - Policy enforcement
- Evaluation

Reference monitor implementation

- ~20K lines of C code (+ MySQL parser library)
- identifying set of columns, rewriting query with applicable policies
- API: create and set policies on columns

Ported reference monitor to:



Policy compliance in HotCRP

- Schema: 22 tables, 215 columns
- Anonymized data from past conference hosting
- Implemented 30 policies for a typical configuration
  - double blind reviewing
  - chair conflict handling
  - review process with no rebuttal

- Application changes
  - overly general queries return fewer results with Qapla
  - changed ~150 out of ~52K LoC in application (< 0.3%)

End-to-end latency for user actions

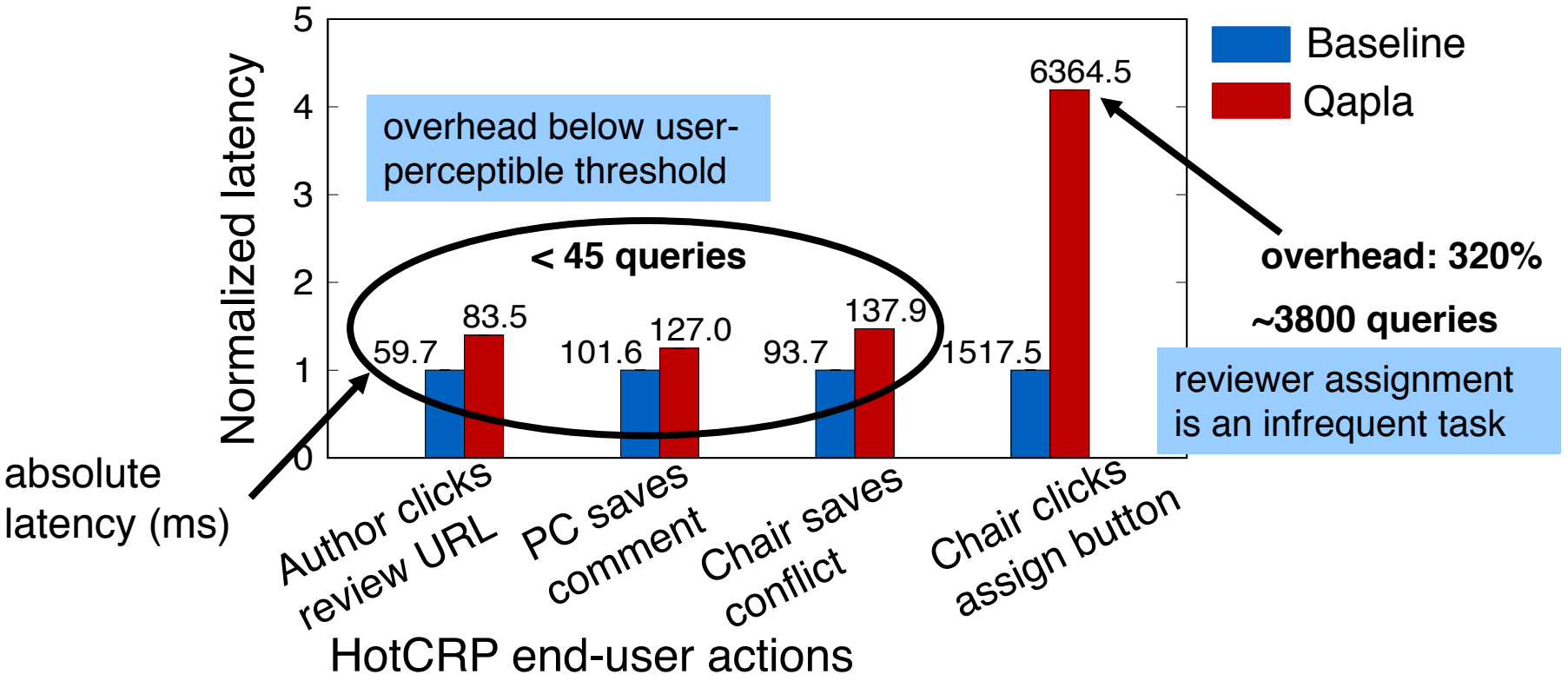
Author clicks review URL: an author clicks the URL of paper to view reviews after notification

PC saves comment: a PC member clicks a button to save comments on a paper during review phase

Chair sets conflict: Chair assigns conflict for a paper and a PC member

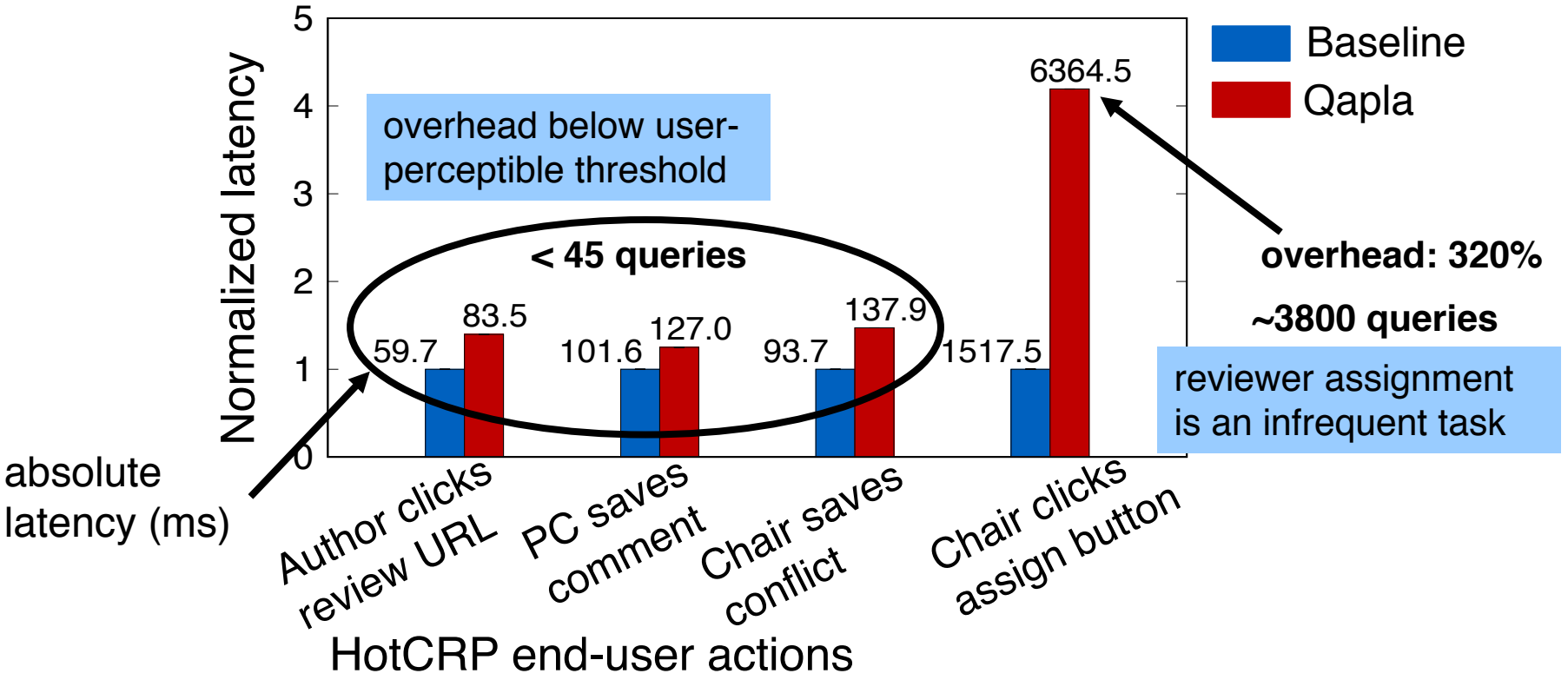
Chair clicks assign button: Chair clicks a button automatic review assignment

End-to-end latency for user actions



End-to-end latency for user actions

Most of Qapla's overhead corresponds to execution of rewritten queries



Summary

DB-backed systems 

data leaks due to policy violations

Qapla: an **effective policy compliance system**

- independent of DBMS support for access control
- independent of application code
- modest changes to application for functionality
- moderate overhead for end users

Questions?