Blockaid

Data-access Policy Enforcement for Web Applications

Wen Zhang Eric Sheng Michael Chang
Aurojit Panda Mooly Sagiv Scott Shenker

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

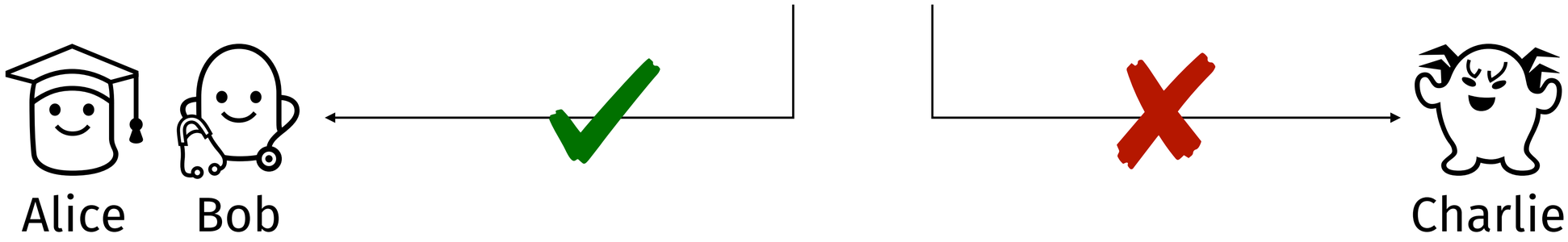
NETSYS

Web Applications Are Everywhere

Web Applications Serve Sensitive Information

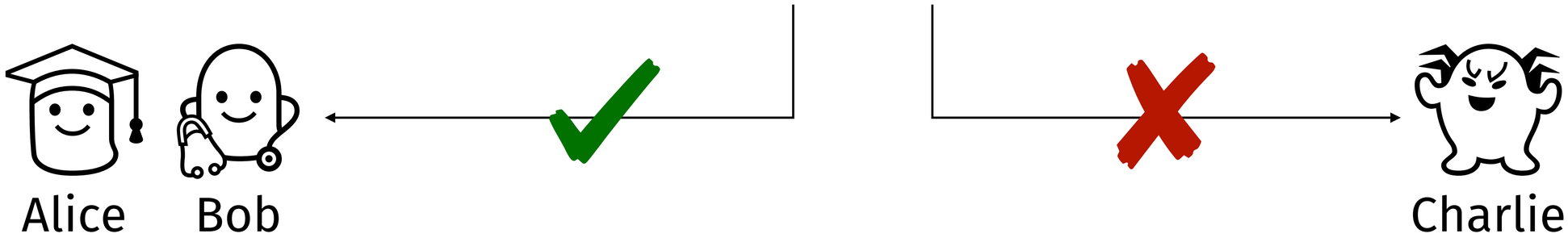
Private Message

From: Alice
To: Bob
Content:
Here’s my little secret…



Web Applications Serve Sensitive Information

Sensitive information be released
only to parties that should have access to it



What information can be accessed by which users?

Data-access Policy

- A **direct message** is accessible only to its **participants**.
- ...

Data-access Policy

Enforced using access checks

- A **direct message** is accessible only to its **participants**.
- ...

Every time a message is displayed...

```
⋮
if not message.has_participant(curr_user):
    return "Error"

return message.content
⋮
```

Access Checks Are Hard to Get Right

Missing/incorrect checks ➡ inadvertent data leaks in production software

How to systematically ensure an application reveals only information allowed by its data-access policy?

How to systematically ensure
an application reveals only information
allowed by its data-access policy?

Blockaid

Run-time data-access policy enforcer for web applications

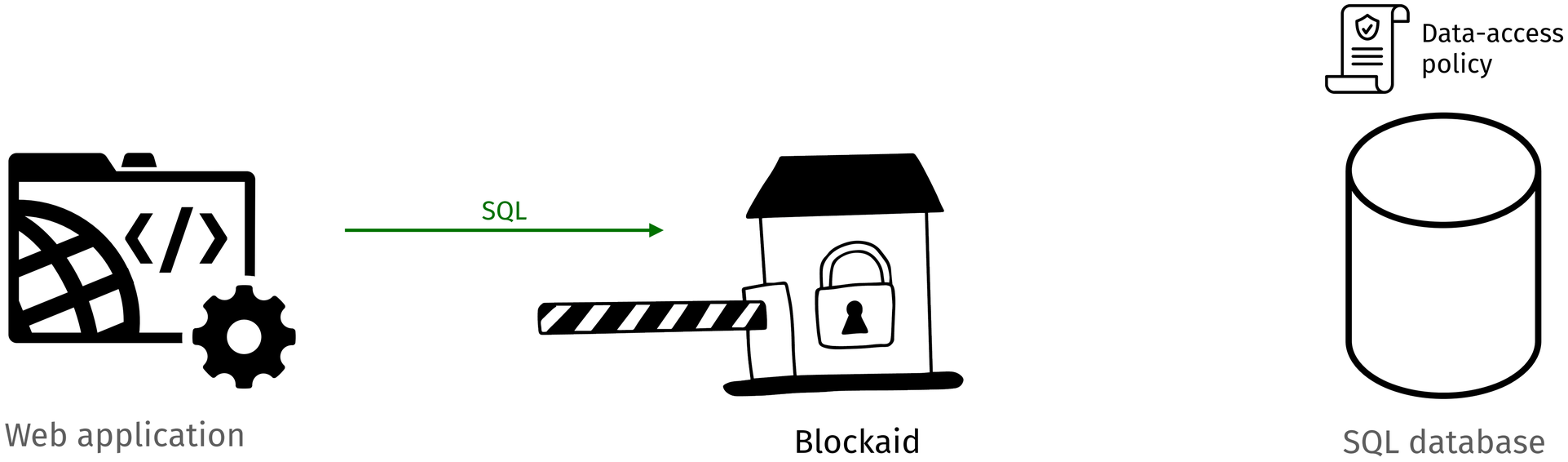
Coming Up Next...

1. Overview and goals.
2. Policy specification.
3. Policy enforcement.
4. Evaluation.

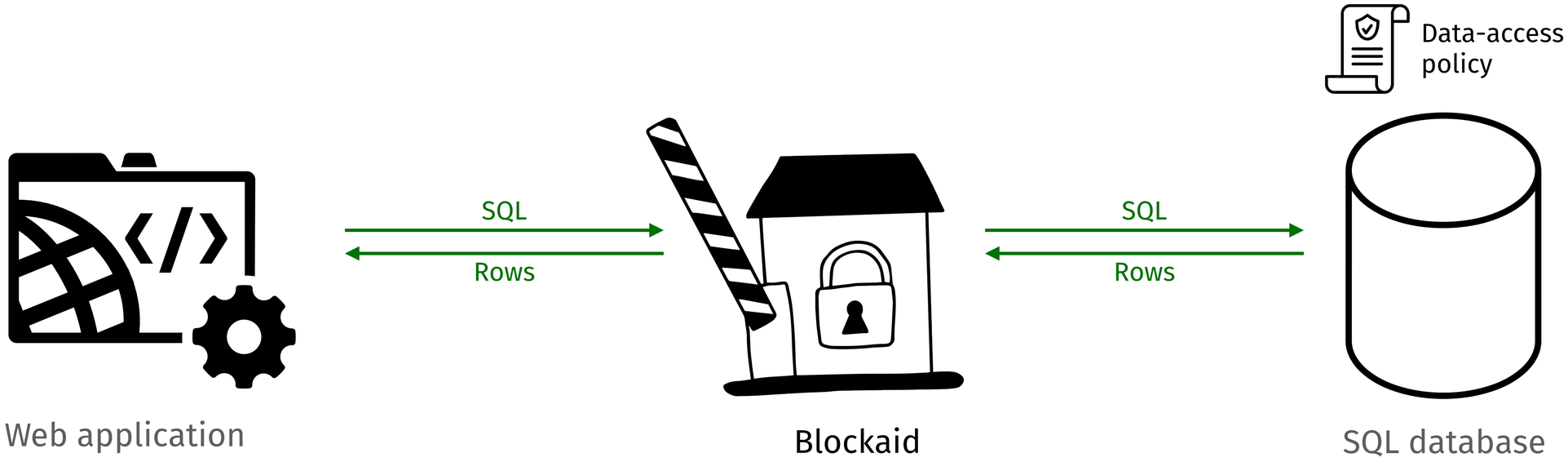
Coming Up Next...

1. Overview and goals.
2. Policy specification.
3. Policy enforcement.
4. Evaluation.

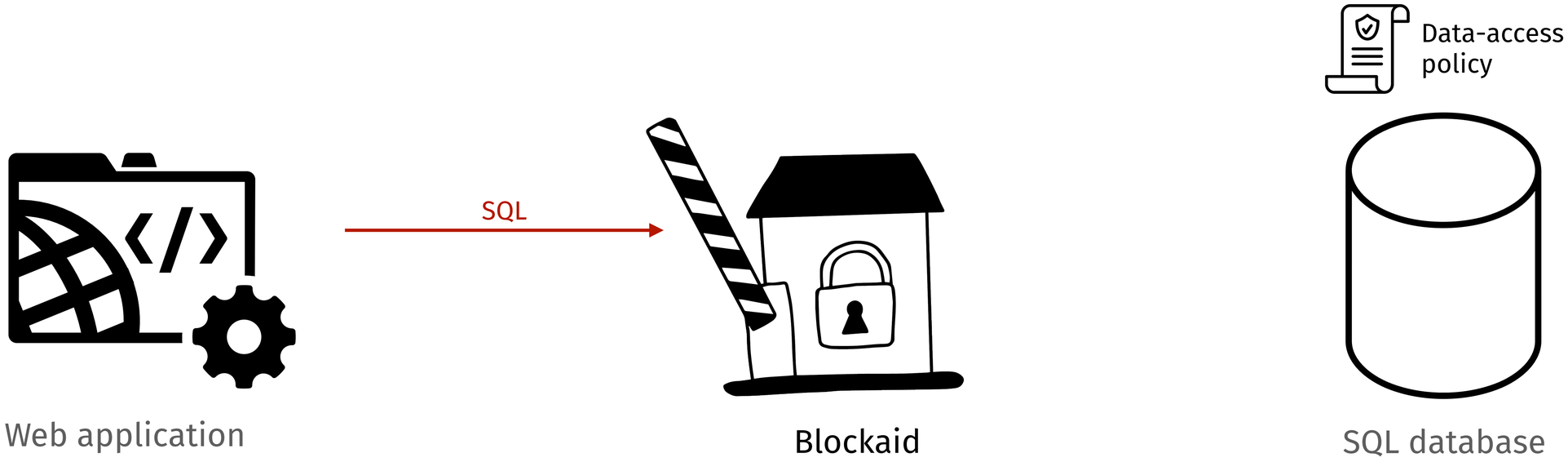
Blockaid: Overview



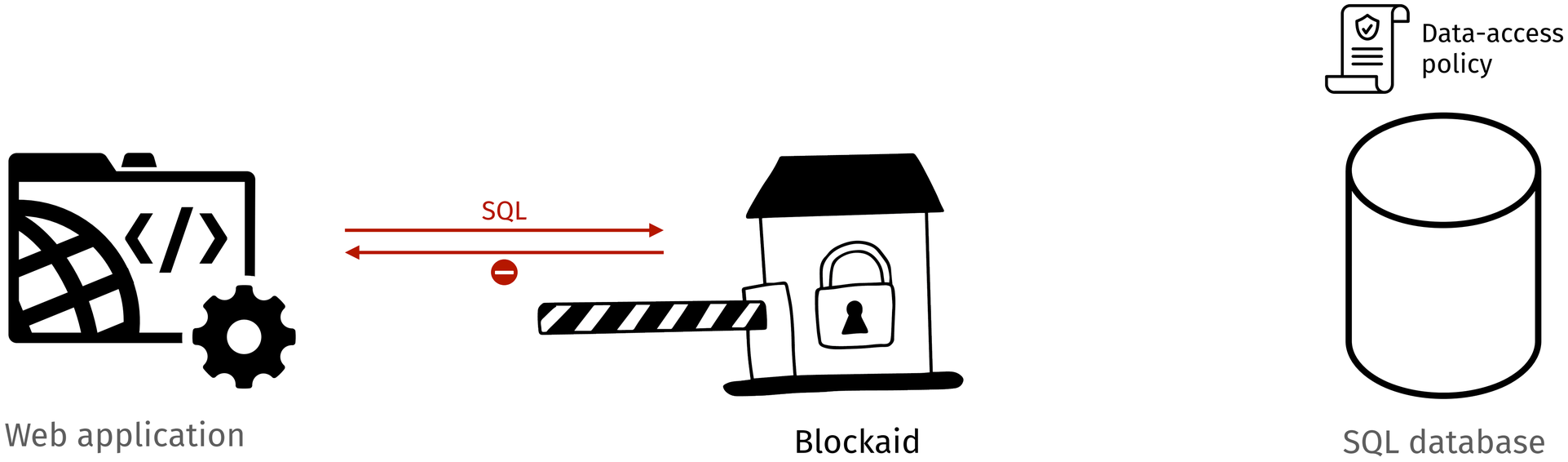
Blockaid: Overview



Blockaid: Overview



Blockaid: Overview



Goals

Ensure application reveals only information that the policy allows the user to access.

1. Policy expressiveness.
2. Compatibility with existing frameworks.
3. Semantic transparency.

   Complies with policy ➡ Maintain application behavior.

   Violates policy ➡ Raise error **visibly**.
4. Low performance overhead.

No Prior System Satisfies All Four Goals

Ensure application reveals only information that the policy allows the user to access.

1. Policy expressiveness.
2. Compatibility with existing frameworks.
3. Semantic transparency.

   Complies with policy ➡ Maintain application behavior.

   Violates policy ➡ Raise error **visibly**.
4. Low performance overhead.

Prior Systems Don't Meet Both Goals

Query Modification

Limiting Disclosure in Hippocratic Databases

Qapla: Policy compliance for database-backed systems

Towards Multiverse Databases

Precise, Dynamic Information Flow for Database-Backed Applications

Jean Yang — Carnegie Mellon University and Harvard Medical School, USA
Travis Hance — Dropbox, USA
Thomas H. Austin — San Jose State University, USA
Armando Solar-Lezama — Massachusetts Institute of Technology, USA
Cormac Flanagan — University of California, Santa Cruz, USA
Stephen Chong — Harvard University, USA

✅ Compatible with existing frameworks

❌ **Not** semantically transparent

Static Verification

Static Checking of Dynamically-Varying Security Policies in Database-Backed Applications

Adam Chlipala
Impredicative LLC

STORM: Refinement Types for Secure Web Applications

Nico Lehmann — *UC San Diego*
Rose Kunkel — *UC San Diego*
Jordan Brown — *Independent*
Jean Yang — *Akita Software*
Niki Vazou — *IMDEA Software Institute*
Nadia Polikarpova — *UC San Diego*
Deian Stefan — *UC San Diego*
Ranjit Jhala — *UC San Diego*

✅ Semantically transparent

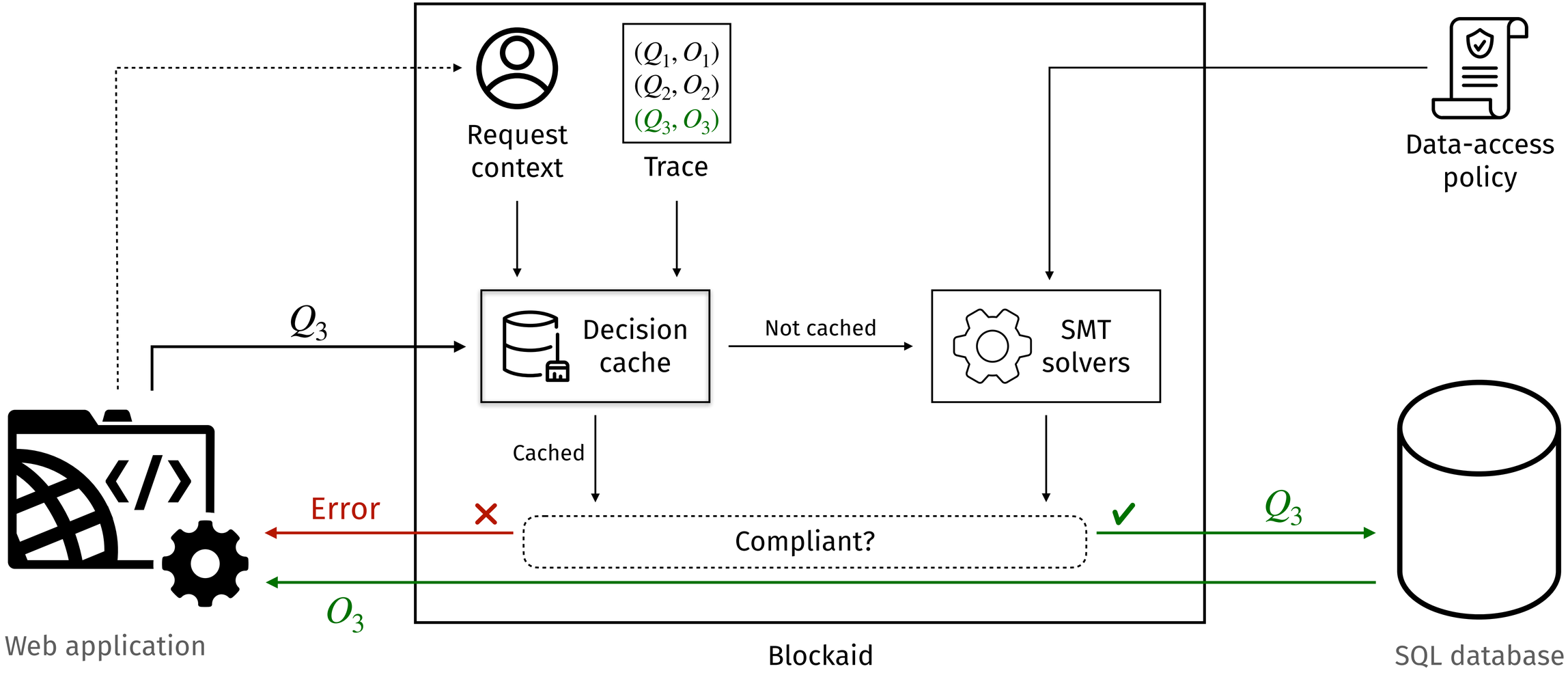
❌ **Incompatible** with existing frameworks

Blockaid

Run-time data-access policy enforcer for web applications

- ✅ Compatible with existing frameworks
- ✅ Semantically transparent
- ✅ Supports expressive policies
- ✅ Incurs low performance overhead

Blockaid: A Close Look

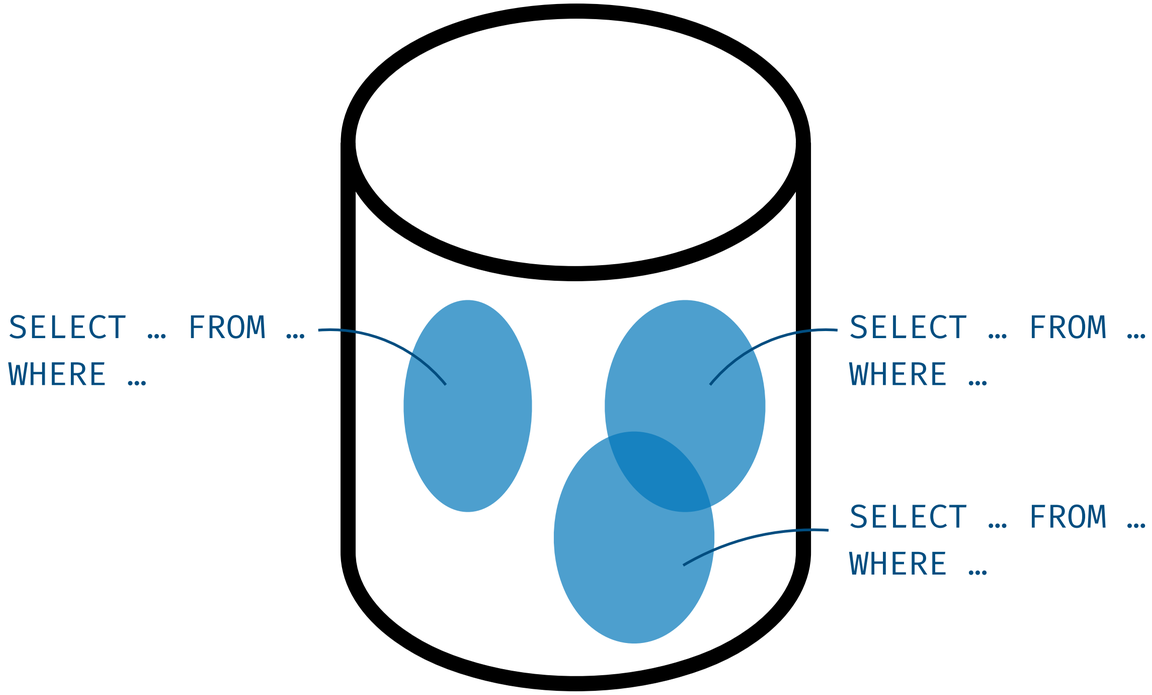


Coming Up Next...

1. Overview and goals.
2. Policy specification.
3. Policy enforcement.
4. Evaluation.

Specify Policy Using Database Views

Views: **SQL SELECT statements** defining data accessible to a user.



Example: Calendar Application

```
Users(UId, Name)
Events(EId, Title, Date)
Attendance(UId, EId)
```

1. `SELECT * FROM Users`

   Each user can view information on all users.

2. `SELECT EId FROM Attendance WHERE UId = ?MyUId`

   Each user can view IDs of events they attend.

3.
```
SELECT *
  FROM Events e
  JOIN Attendance a ON e.EId = a.EId
 WHERE a.UId = ?MyUId
```

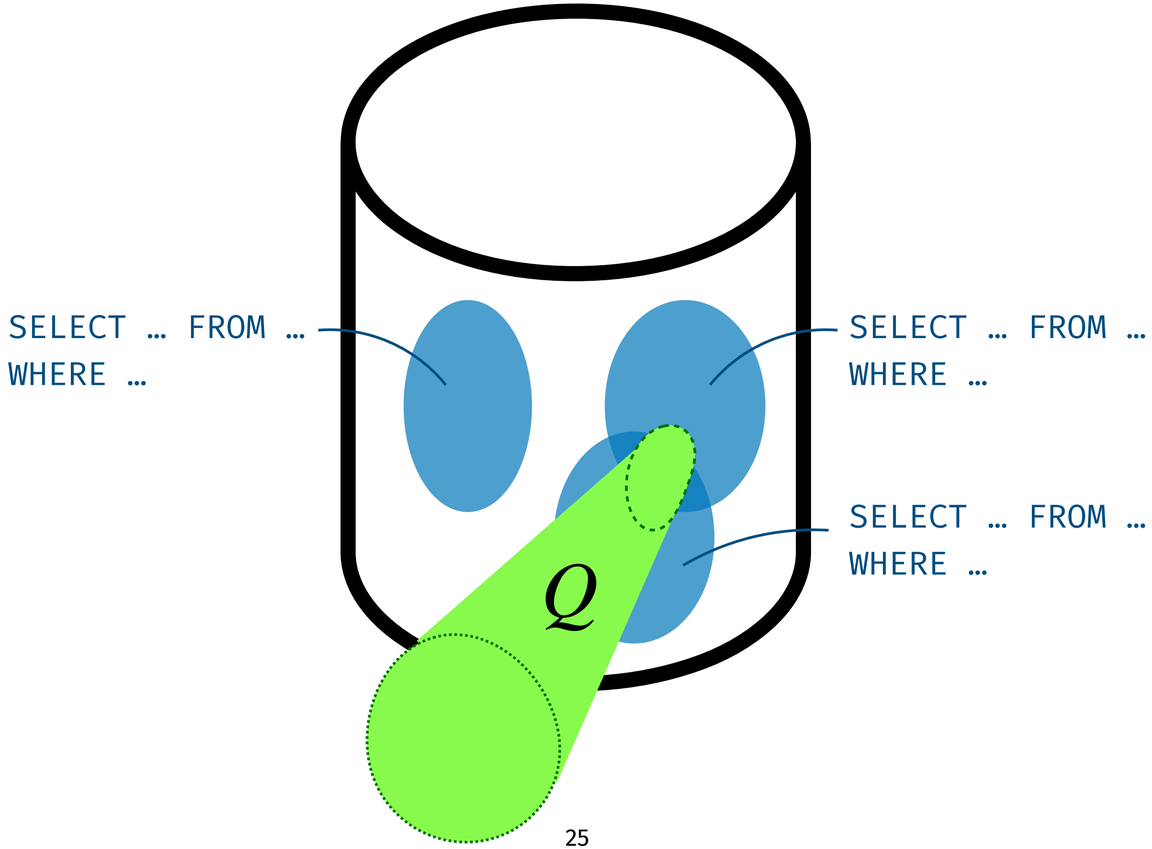
   Each user can view details of events they attend.

Coming Up Next...

1. Overview and goals.
2. Policy specification.
3. Policy enforcement.
4. Evaluation.

Query Compliance

A **compliant query** reveals only information exposed by the views.



Example: Calendar Application

```
Users(UId, Name)
Events(EId, Title, Date)
Attendance(UId, EId)
```

1. `SELECT * FROM Users`

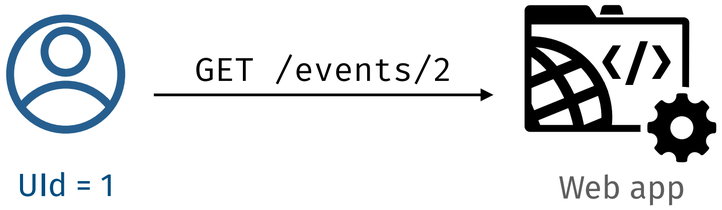
   Each user can view information on all users.

2. `SELECT * FROM Attendance`
   `WHERE UId = ?MyUId`

   Each user can view which events they attend.

3. `SELECT *`
   `FROM Events e`
   `JOIN Attendance a`
   `ON e.EId = a.EId`
   `WHERE a.UId = ?MyUId`

   Each user can view information on events they attend.



1. `SELECT * FROM Attendance`
   `WHERE UId = 1 AND EId = 2`

   ✅ Covered by View 2.

   ↩ Returns one row.

2. `SELECT * FROM Events`
   `WHERE EId = 2`

   ✅ Covered by View 3 *given Query 1 & result.*

Query Compliance, Formally

Given:

- A set $\mathscr{V}$ of views (for the current request context),
- A trace of query-result pairs: $(Q_1, O_1), \ldots, (Q_n, O_n)$,

We say query Q is **compliant** if for every *pair of databases* D, D':

Consistent with the observed trace $\quad Q_i(D) = Q_i(D') = O_i \qquad \forall 1 \leq i \leq n$

Contain the same accessible information $\quad V(D) = V(D') \qquad \forall V \in \mathscr{V}$

$$\Downarrow$$

$$Q(D) = Q(D')$$

Checking Compliance

$$\underbrace{\begin{array}{c} Q_i(D) = Q_i(D') = O_i \\ V(D) = V(D') \end{array}}_{\Downarrow \; Q(D) = Q(D')}$$

Compliance definition

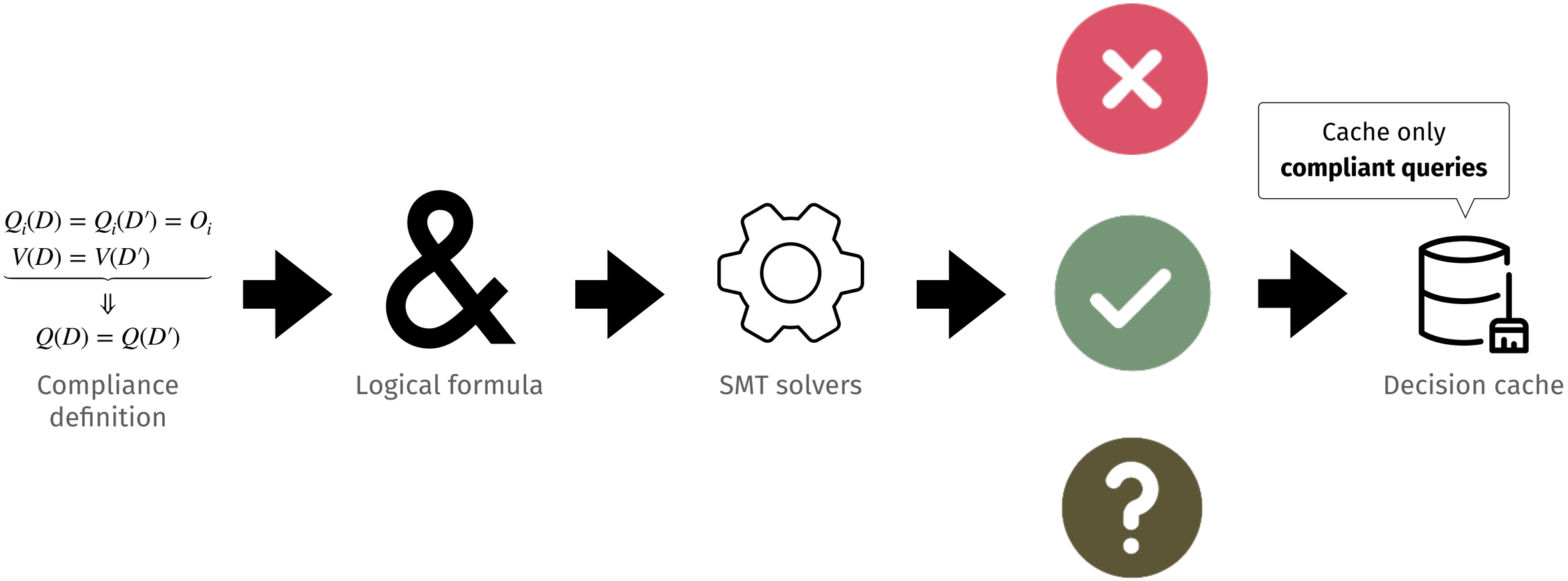
Translate SQL into first-order logic
via Codd's Theorem + approximation.

Logical formula

SMT solvers

SMT solvers can be **slow**!
100 ms per query ⇝ seconds per page

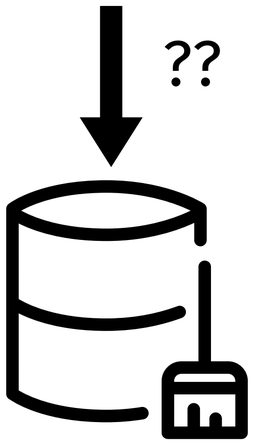
Checking Compliance Fast



Naive Caching ⇝ Low Cache Hit Rate

Assuming fixed database schema and policy...

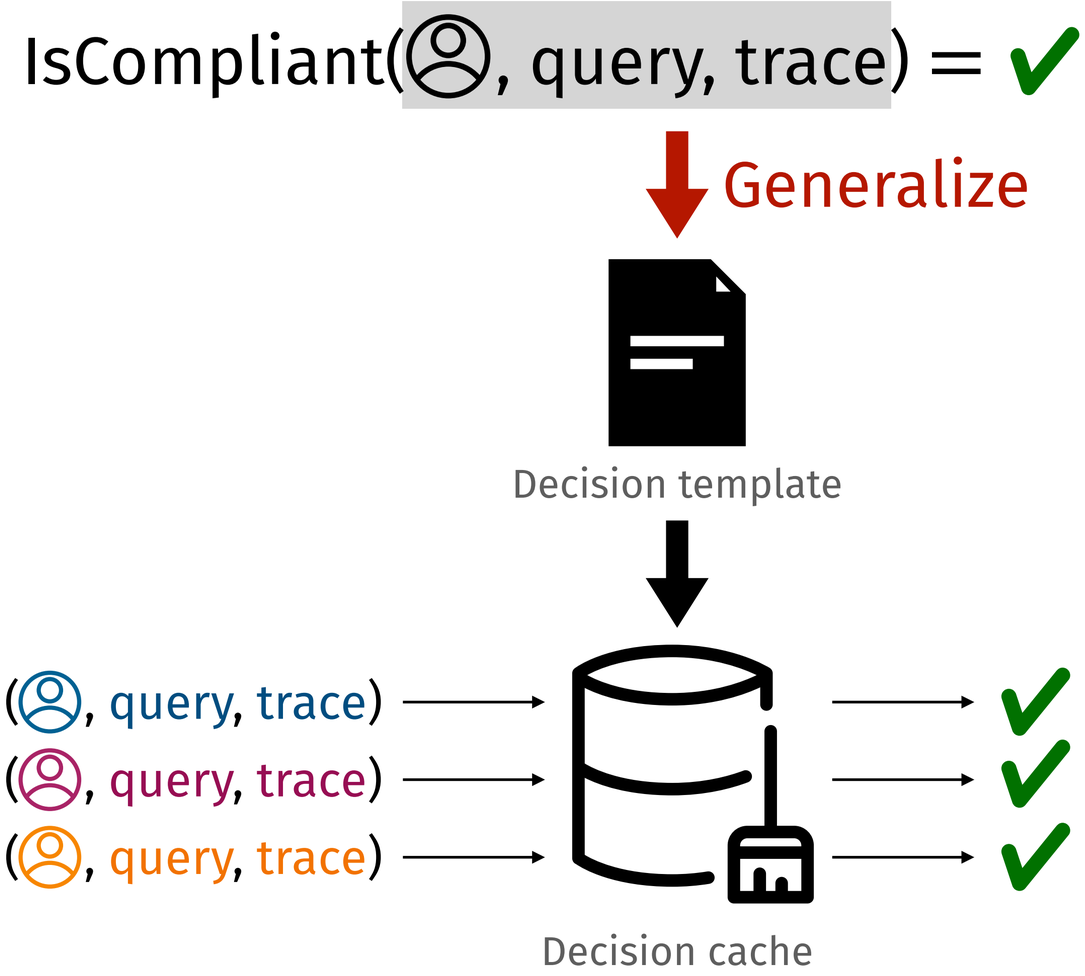
IsCompliant(👤, query, trace) = ✔



Decision cache

Each entry specific to user & URLs visited!

Blockaid Generalizes Compliance Determinations

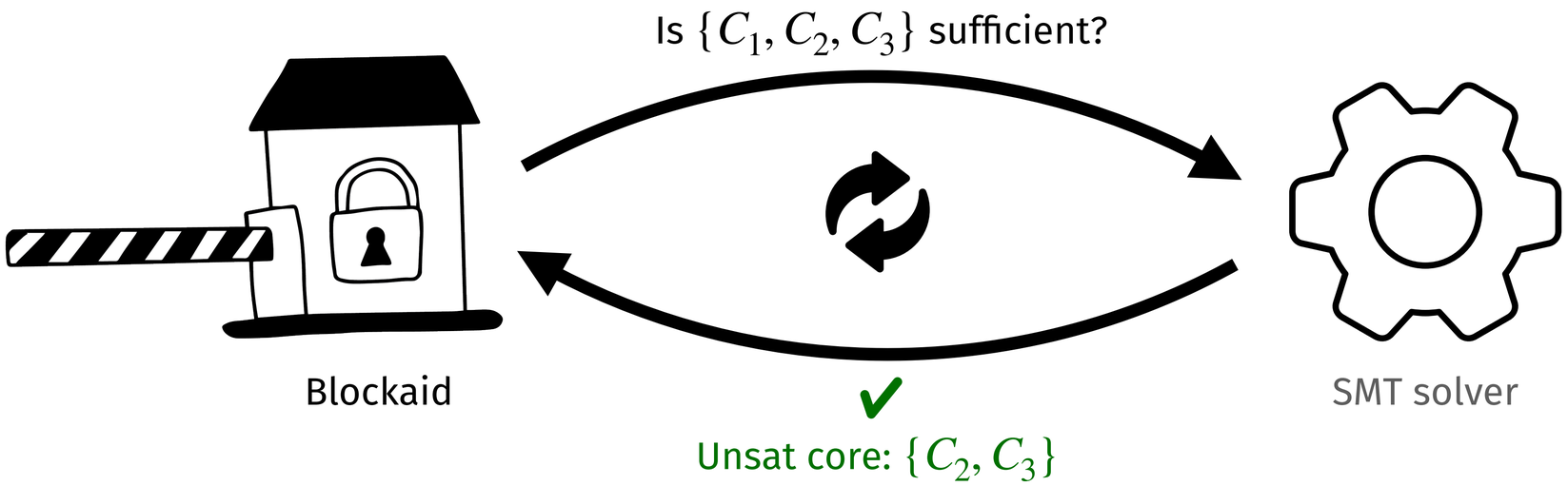


How Are Templates Generated?

Find a small set of constraints

on the request context, query, & trace

that is sufficient to guarantee compliance



Coming Up Next...

1. Overview and goals.
2. Policy specification.
3. Policy enforcement.
4. Evaluation.

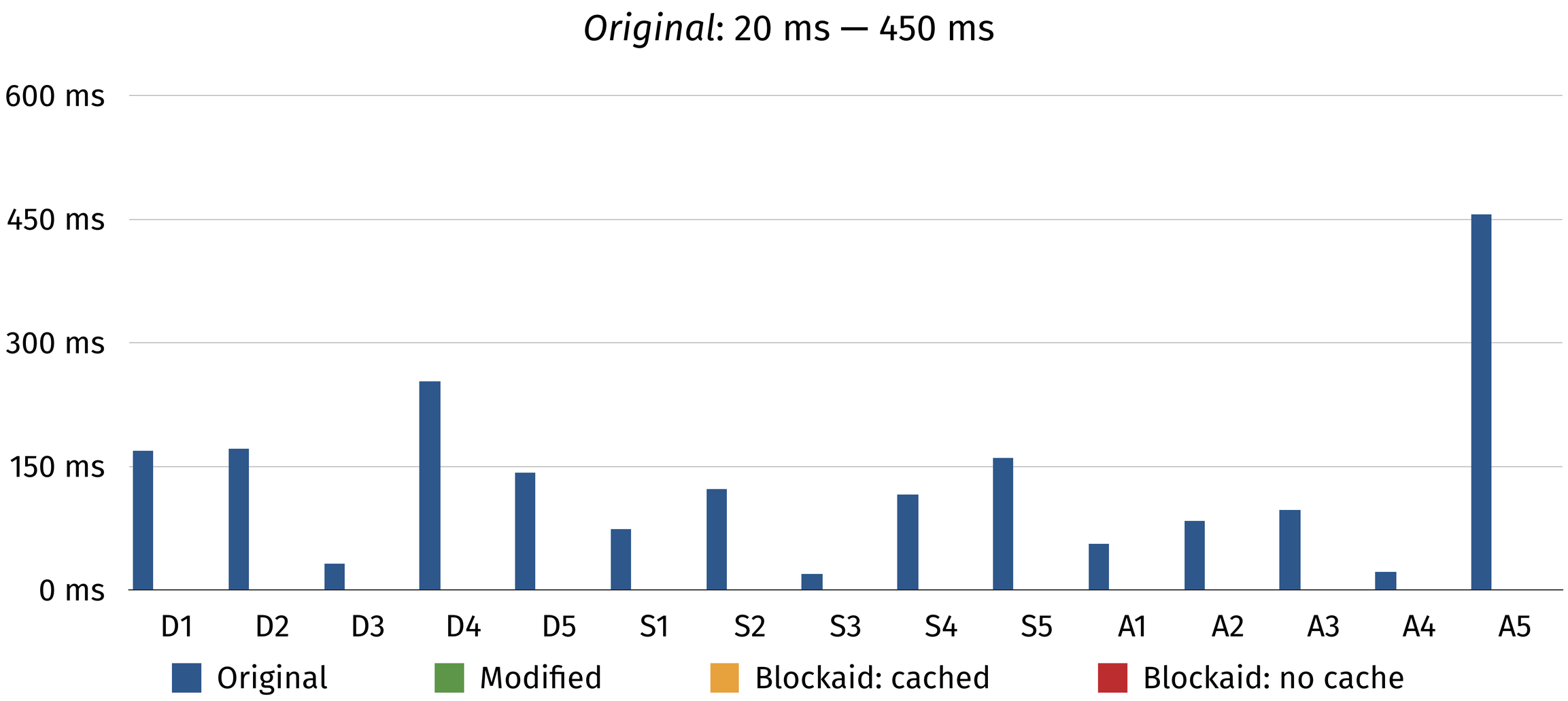
Setup

- Implemented Blockaid as **JDBC driver** wrapping a database connection.
- Applied Blockaid to **three applications** (with hand-crafted policies).

diaspora* spree AUTØLAB

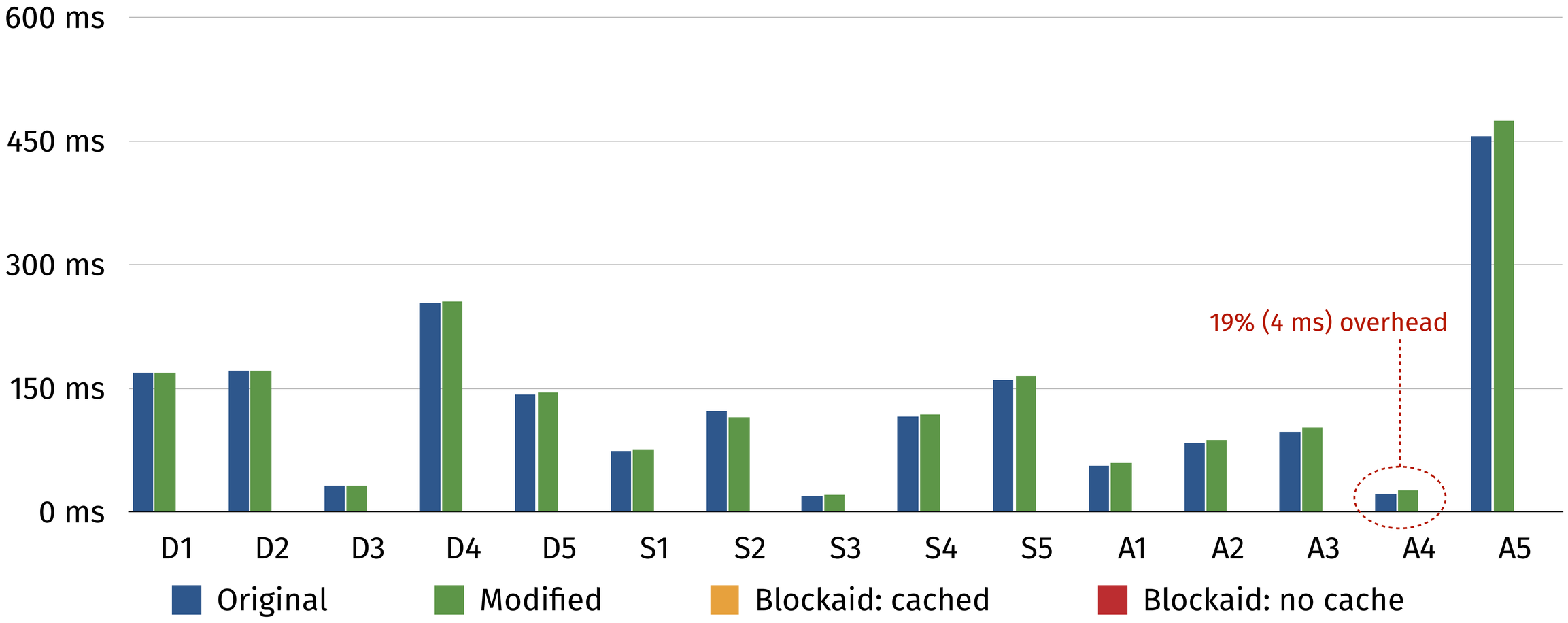
- Measured **page load time** of 5 URLs per application.
- Modified **~20 — 100 lines** of code per application.
  - To fetch only data that **can be revealed** to the user.

Page Load Times (median)



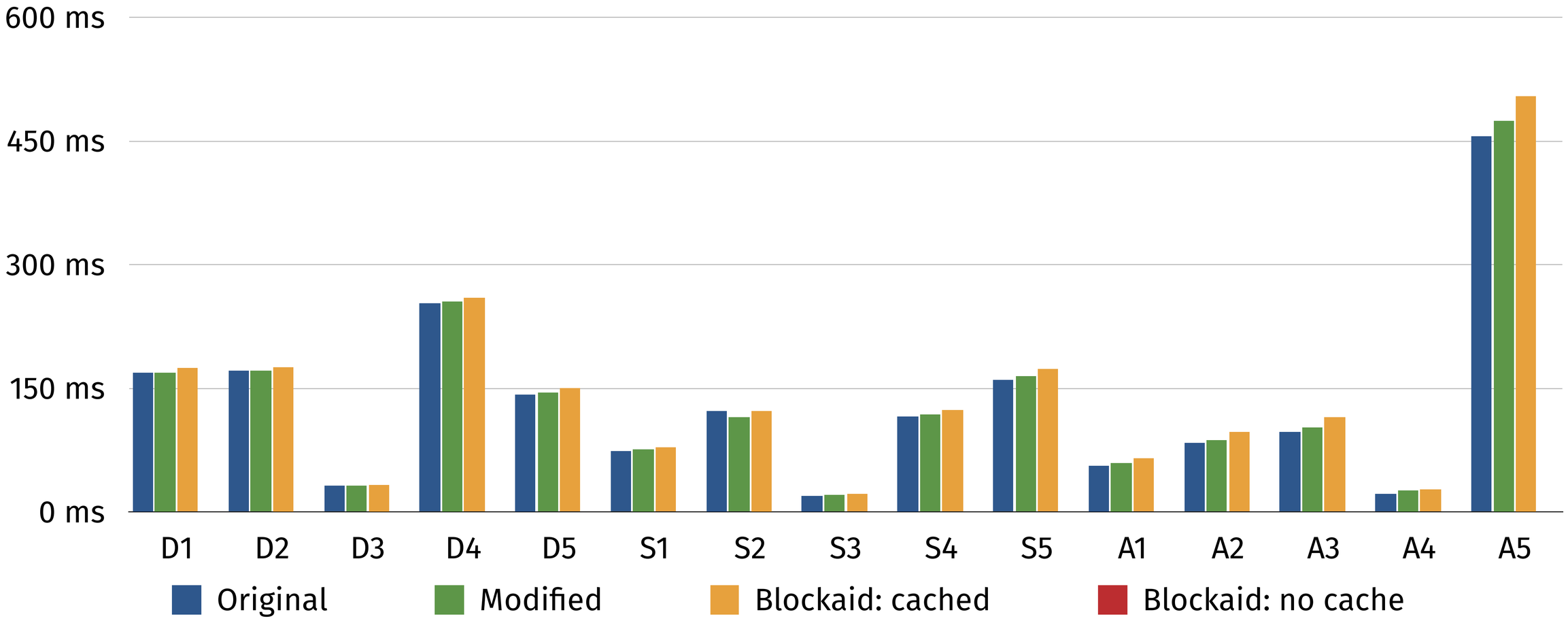
Page Load Times (median)

Modified: up to 6% overhead for all but one page



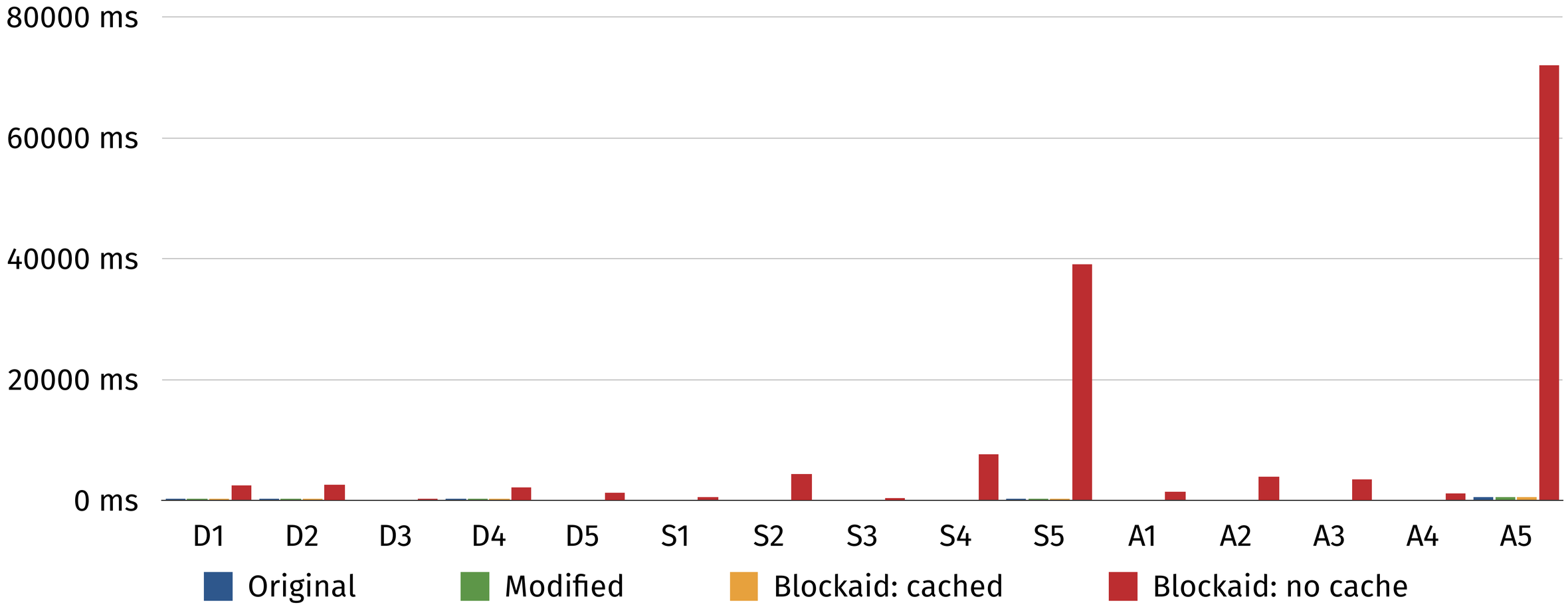
Page Load Times (median)

Cached: **up to 12% overhead** over *modified*



Page Load Times (median)

No cache: **up to 236x overhead** over *modified*



More Evaluation in the Paper

- Breakdown of page load times.
- Solver performance.
- Template generalization case study.

Blockaid

Data-access Policy Enforcement for Web Applications

- Compatible with existing frameworks ✚ semantically transparent.
- Uses SMT to verify query compliance with view-based policy.
- Generalization-based caching through decision templates.

https://github.com/blockaid-project/

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED