Access Control for Database Applications: Beyond Policy Enforcement

Wen Zhang
UC Berkeley

Aurojit Panda
NYU

Scott Shenker
UC Berkeley & ICSI

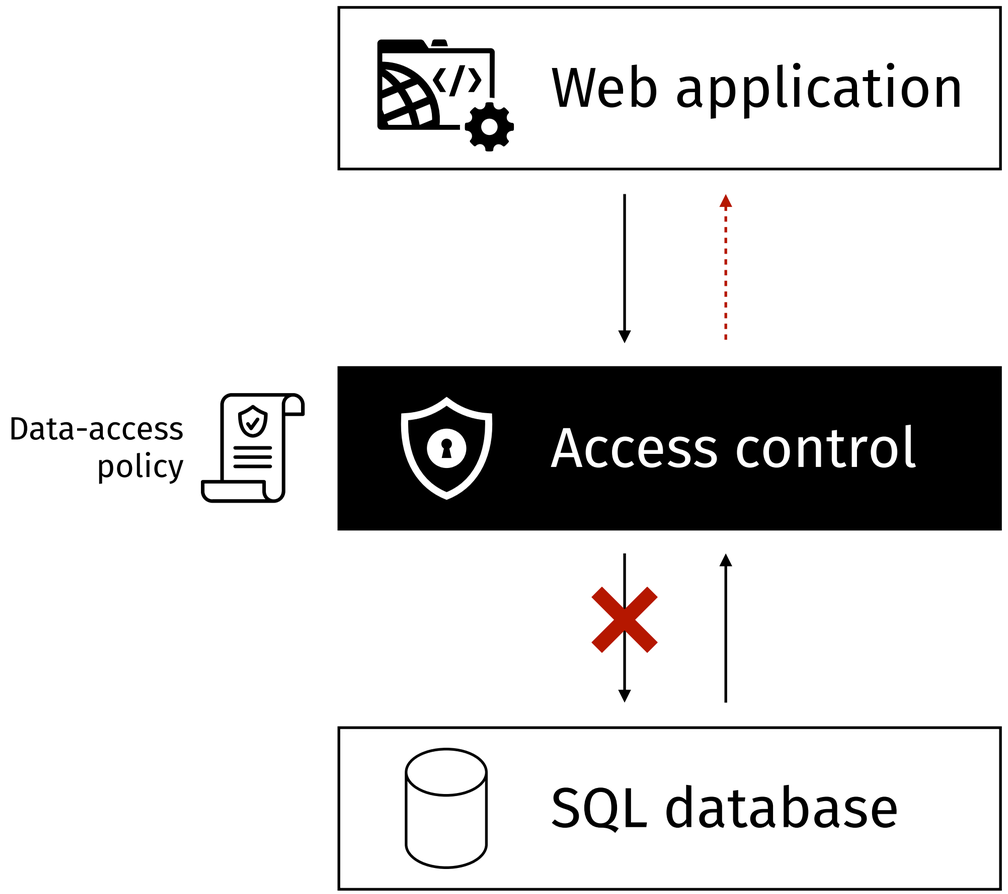
Apps must take care when revealing sensitive data

To ensure sensitive data is revealed only to authorized parties, developers insert permission checks + query filters in their code.

Permission checks and query filters are easy to miss or get wrong.

When this happens, sensitive data is leaked to unauthorized parties.

Access control to the rescue



Enforcement mechanism

Enforcement: Focus of access-control literature

Limiting Disclosure in Hippocratic Databases — VLDB '04

Kristen LeFevre[†*] Rakesh Agrawal[†] Vuk Ercegovac[*] Raghu Ramakrishnan[*]
Yirong Xu[†] David DeWitt[*]

[†]IBM Almaden Research Center, San Jose, CA 95120
[*]University of Wisconsin, Madison, WI 53706

Static Checking of Dynamically-Varying Security Policies in Database-Backed Applications — OSDI '10

Adam Chlipala
Impredicative LLC

Precise, Dynamic Information Flow for Database-Backed Applications — PLDI '16

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

Qapla: Policy compliance for database-backed systems — USENIX Sec '17

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

[1]Max Planck Institute for Software Systems (MPI-SWS), Saarland Informatics Campus
[2]University of Maryland, College Park

Towards Multiverse Databases — HotOS '19

Alana Marzoev — *MIT CSAIL*
Lara Timbó Araújo[†] — *[†] MIT CSAIL and Airbnb*
Malte Schwarzkopf — *MIT CSAIL*
Samyukta Yagati — *MIT CSAIL*
Eddie Kohler[‡] — *[‡] Harvard University*
Robert Morris — *MIT CSAIL*
M. Frans Kaashoek — *MIT CSAIL*
Sam Madden — *MIT CSAIL*

STORM: Refinement Types for Secure Web Applications — OSDI '21

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*

Blockaid: Data Access Policy Enforcement for Web Applications — OSDI '22

Wen Zhang[1] Eric Sheng[2,*] Michael Chang[1] Aurojit Panda[3] Mooly Sagiv[4] Scott Shenker[1,5]
[1]*UC Berkeley* [2]*Yugabyte* [3]*NYU* [4]*Tel Aviv University* [5]*ICSI*

The full life-cycle of access control

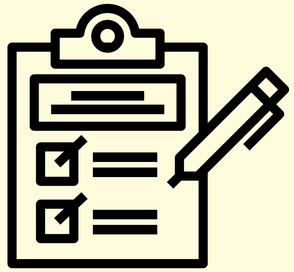
You're a web admin. You want to deploy access control on your web app.

Many challenges—**we outline a path to addressing them.**

Create a policy

"What data does the application need to function?"

and then...



Evaluate the policy

"Does this policy adequately protect sensitive data?"

and then...

Enforce the policy

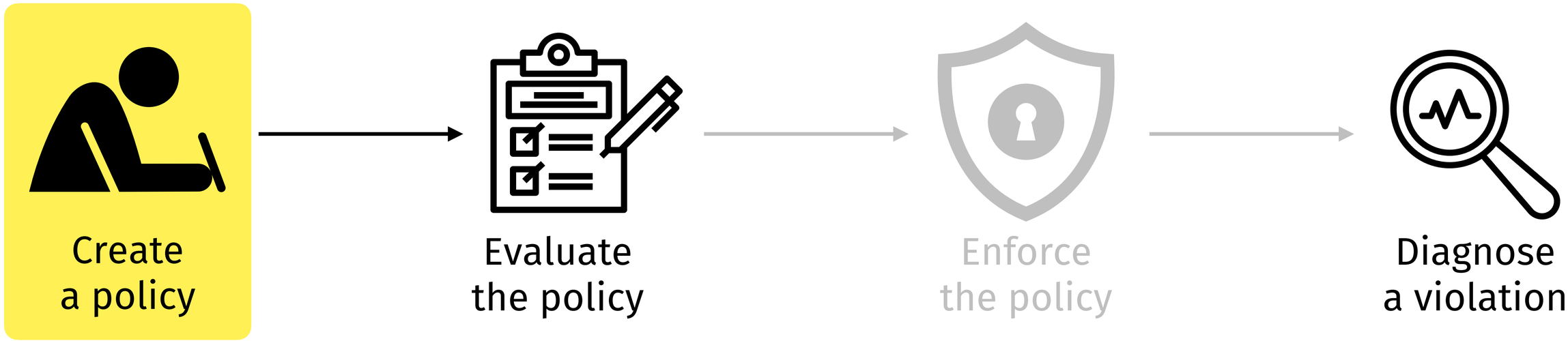
Policy violation

and then...

Diagnose a violation

"Why was my query rejected, and how do I fix this?"

Challenges in the access-control life-cycle



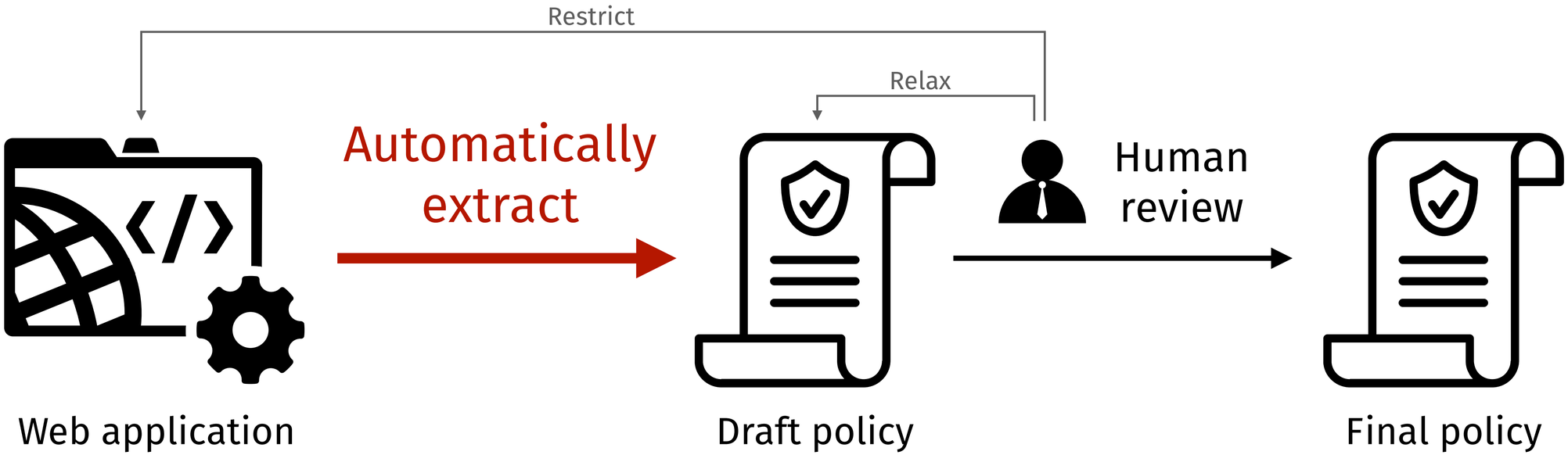
Writing a policy from scratch is challenging

- You're an admin, wanting to write a data-access policy for your application.
  1. You map out and understand all the intended data accesses…
  2. And write down a good policy.
- This is tedious and error-prone.
  - As you wade through thousands upon thousands of line of code…
  - You can easily miss an edge case, or make a typo in the policy.

Can we make policy creation easier?

- Writing a policy inherently requires human input.
  - Balance application needs vs social, regulatory requirements.
- But we should automate as much as possible.
- Assumption:
  - Compared to writing a policy from scratch…
  - Reviewing and refining a draft policy is much easier.

Proposal: Policy extraction

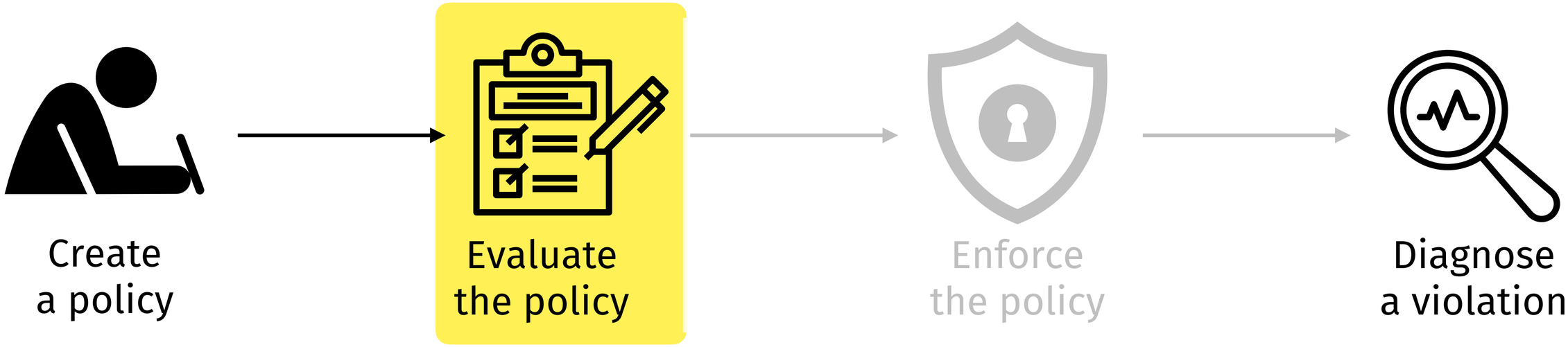


Ideally: Strictest policy that allows all possible access.

How to extract draft policy?

- Language-based: Symbolic execution.
  - Explore program paths ➡ Gather (symbolic) queries ➡ Construct policy.
  - **Challenge**: Web languages & frameworks are highly dynamic.
- Language-agnostic: (Run-time) Specification mining.
  - Run the application ➡ Observe (concrete) queries ➡ Construct policy.
  - **Challenge**: Must run application on comprehensive input suite.
  - **Challenge**: Must generalize concrete queries into policy.
    - Doctor #1 accesses Patient #10 ➡ Doctors can access patients they treat.

Challenges in the access-control life-cycle



Does a policy adequately protect sensitive data?

- A policy must strike a balance between:
  - Application's need to access data,
  - Admin's need to protect sensitive information.
- **Where does a given policy fall in the balance?**
  - Given a policy for allowed queries, and a sensitive query…
  - Does the policy disclose too much about sensitive query's output?
- To discuss this, we need a metric for disclosure.

Possible metric: Reject the sensitive query.

Reject sensitive query ≠ Prevent disclosure

- In a medical-records management system, the policy allows analyst to view:
  1. The doctor assigned to each patient.
  2. The diseases treated by each doctor.
- Sensitive query: What disease is patient John being treated for?
  - Query is rejected under the policy…
  - But significant disclosure is possible from answers to allowed queries.
  - (E.g., if John is being treated by a doctor who treats only two diseases.)

State-of-the-art metric: Bayesian privacy

Given an adversary's prior belief over what the sensitive data might be...

Compute a posterior belief after seeing the allowed data.

Sensitive data **disclosure** = Difference between the prior and the posterior.

- Can seeing allowed data change an adversary's mind about sensitive data?
- **Issue**: Must estimate what the adversary believes *a priori*.
  - Hard to model realistically and validate empirically.

Deutsch, A. (2008). Privacy in Database Publishing: A Bayesian Perspective. In: Gertz, M., Jajodia, S. (eds) Handbook of Database Security.

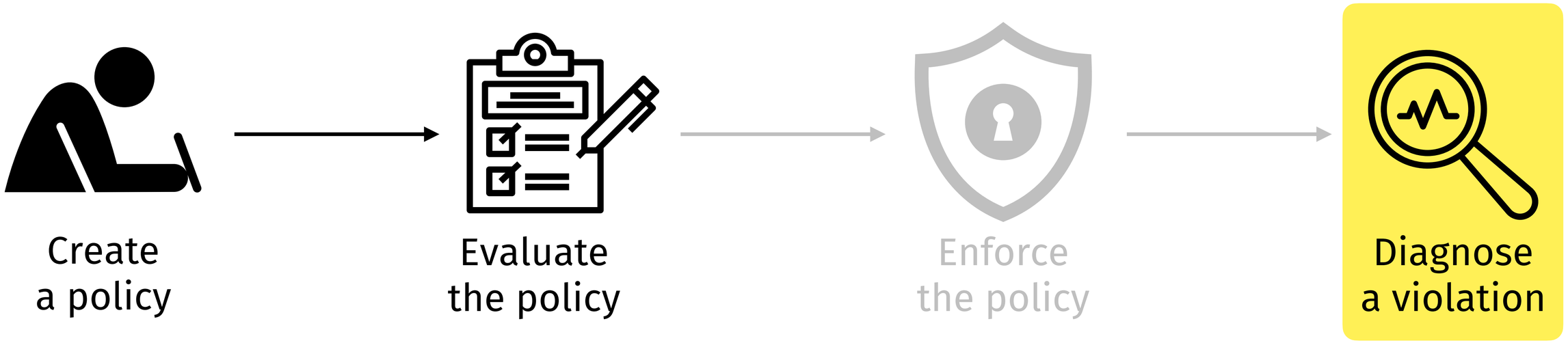
Proposal: Prior-agnostic privacy criteria

We should use metrics that do not require modeling priors.

- Two examples from theoretical literature:
  - PQI: Can allowed data imply John definitely has pneumonia?
  - NQI: Can allowed data imply John definitely does not have pneumonia?
- Coarser-grained than Bayesian criteria, but meaningful regardless of belief.
- **Challenges**:
  - How to measure prior-agnostic privacy?
  - How to present the result to the admin?

Michael Benedikt, Pierre Bourhis, Balder ten Cate, Gabriele Puppis, and Michael Vanden Boom. 2021. Inference from Visible Information and Background Knowledge. ACM Trans. Comput. Log. 22, 2 (2021).

Challenges beyond enforcement

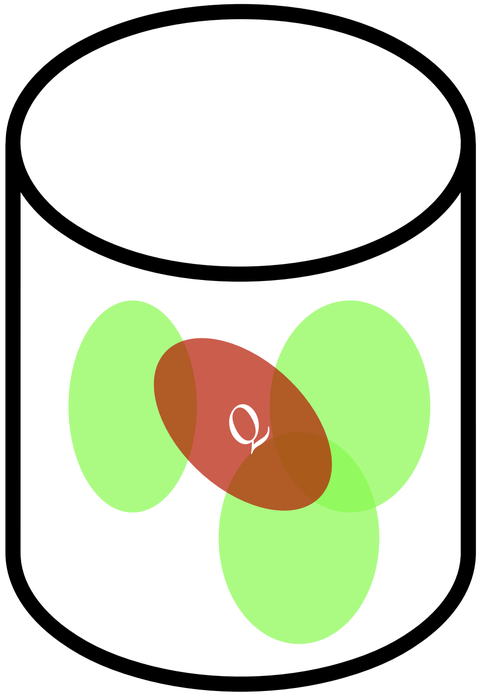


When a query gets rejected…

- One day, the application issues a query that gets rejected under the policy.
  - Why was the query rejected?
  - How do I fix this?
- You're shown: The policy, the offending query, a stack trace.
  - Diagnosing the violation is still difficult—*too much information!*
- **How to better assist the admin in diagnosing such a violation?**
  - Ideally: Give a small amount of feedback that the admin can act upon.

What would the ideal feedback look like?

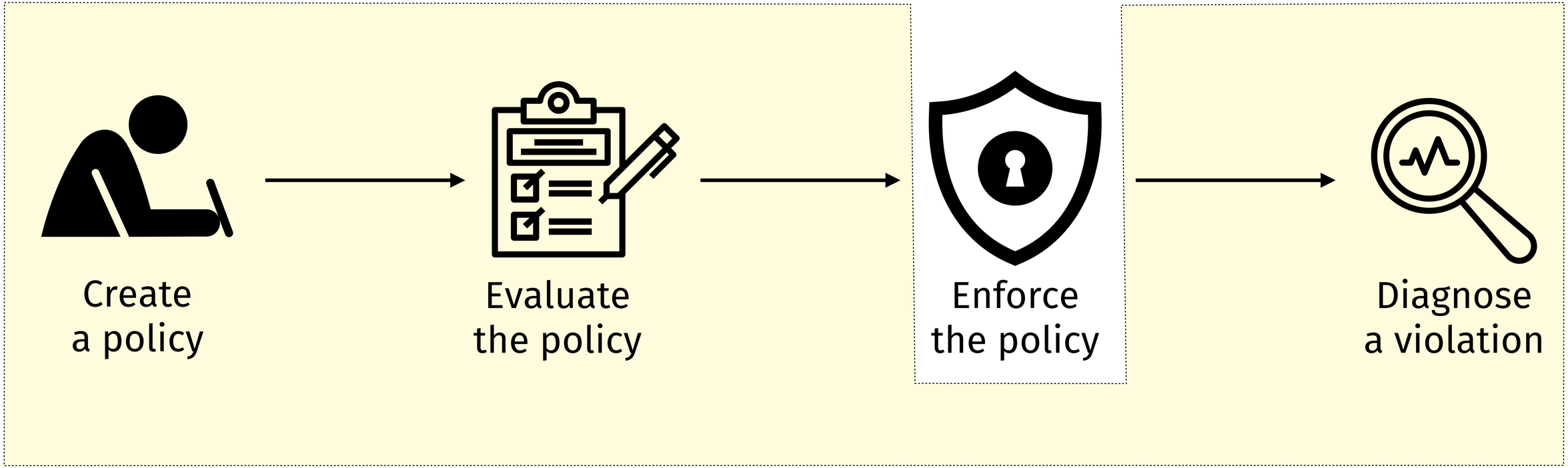
- It is unclear, especially for expressive policies.
- An allow-list policy specifies accessible information.
- A query gets rejected simply because the information it reveals is not contained in "accessible regions".
- No policy item—or subset of items—is *responsible* for the rejection.
  - Hard to *explain why* the query was rejected.



Proposal: Generate fixes, show to admin

- Fixing the policy: Grant access to more data.
  - **Approach**: Re-run policy extraction on updated source code / input suite.
- Fixing the application: Narrow down the query, or insert access check.
  - **Approach**: View-based query rewriting, abductive reasoning.

Addressing the full life-cycle of access control



Thank you!

Wen Zhang <zhangwen@cs.berkeley.edu>