Extracting Database Access-Control Policies From Web Applications

Wen Zhang Dev Bali Jamison Kerney

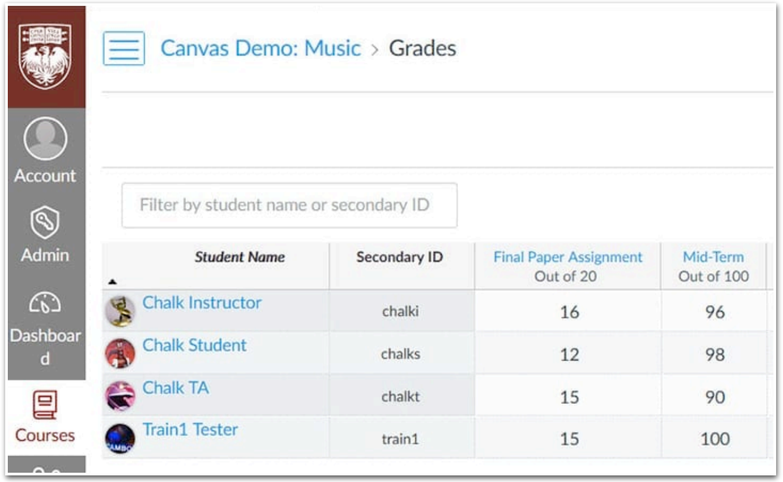
Aurojit Panda Scott Shenker



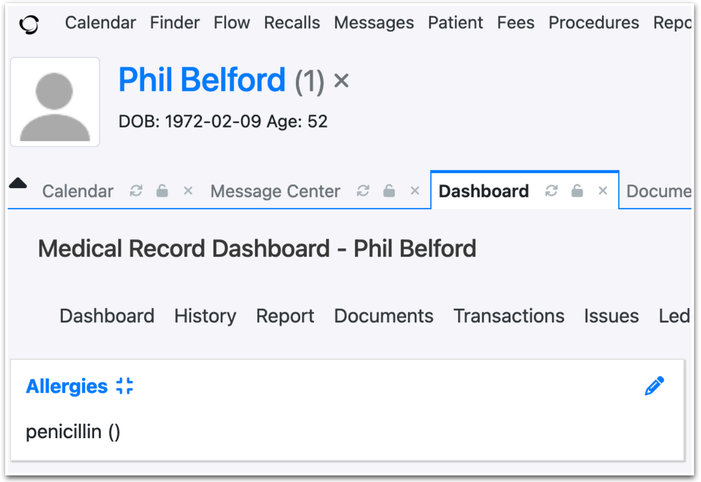


Web apps store and serve sensitive user data

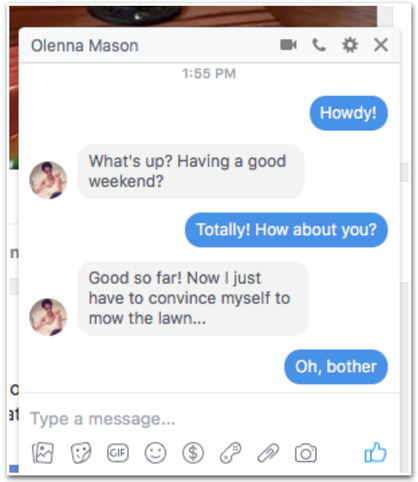
Education records



Medical records



Private messages



I do not want my data to be leaked!

How do web applications today protect sensitive data?

Control data access in application code

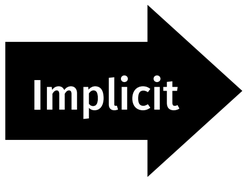
```
if not curr_user.is_attending(id):
    raise Http404("Event not found")

event = Event.find(id)
return format_html(event)
```

Access-control code implicitly defines a policy

```
if not curr_user.is_attending(id):
    raise Http404("Event not found")

event = Event.find(id)
return format_html(event)
```



Access-Control Policy

1. A user can see the calendar events that they attend.
2. A user can see their own profile.
3. ⋮

The access-control policy is implicit!

- Most web applications today have **no explicit policy** written down.
- "Actual policy" lives in the developer's head!
  - Implicitly reflected in code that enforces it…
  - Scattered across the codebase.

Access-Control Policy

1. A user can see the calendar events that they attend.
2. A user can see their own profile.
3. ⋮

Goal of this work

Take the implicit access-control policy and make it explicit.

But what's wrong with leaving the policy implicit?

What's wrong with an implicit policy?

1. It is hard to understand exactly what is being enforced.
2. Scattered access checks in business logic are error-prone.
   - *Potential solution: centralized access control outside business logic.*
   - *Many systems offer this: Jacqueline, Qapla, STORM, Blockaid, …*
3. Without an explicit policy, can't externally enforce access control!

Goal of this work

Take the implicit access-control policy and make it explicit.

Just ask the developer to write an explicit policy?

Writing a policy by hand is really hard!

I once hand-wrote policies for three existing web applications.

Speaking from experience…

The human needs help creating explicit access-control policies!

Ote: Access-Control Policy Extraction

Helps a human **create an explicit policy** for existing web application—

by extracting the policy embedded in its code.

- Explores code paths through the application via concolic execution.
- Generalizes the encountered queries and conditions into a policy.

✓ Can extract a policy *within ≈hours*.

✓ *Saves human effort* and *avoids human errors!*

Coming up next

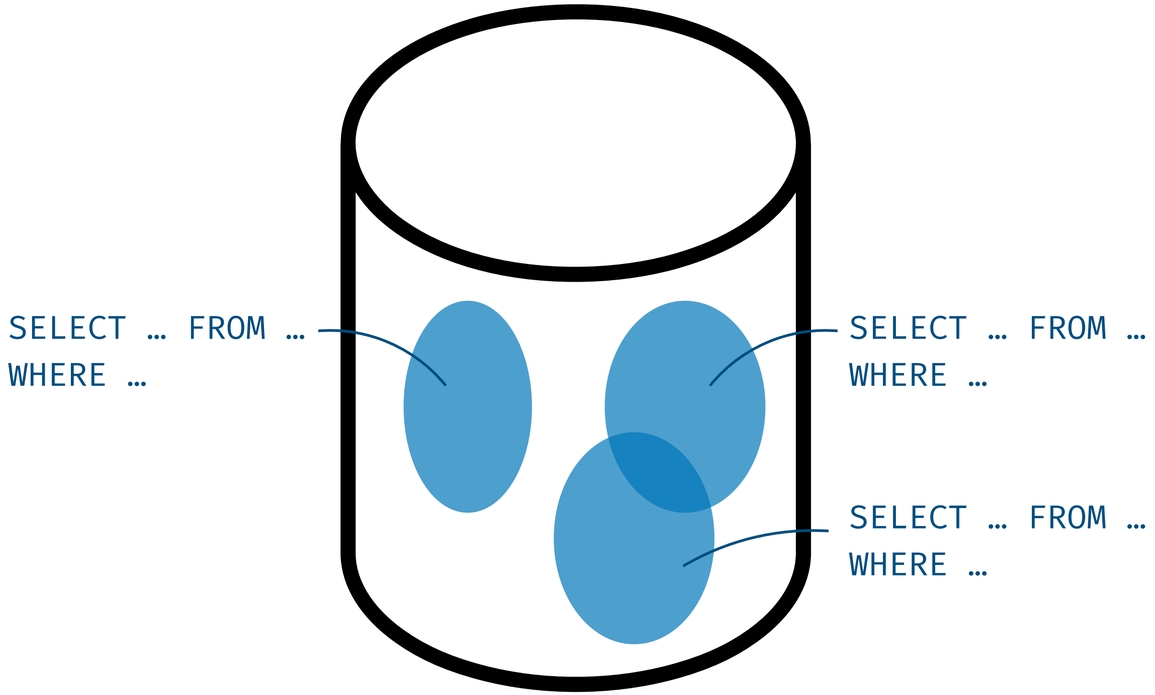
1. What form does a policy take in Ote?
2. What is the workflow for using Ote to extract a policy?
3. How does Ote extract a policy?
4. What did we learn from applying Ote?

Coming up next

1. What form does a policy take in Ote?
2. What is the workflow for using Ote to extract a policy?
3. How does Ote extract a policy?
4. What did we learn from applying Ote?

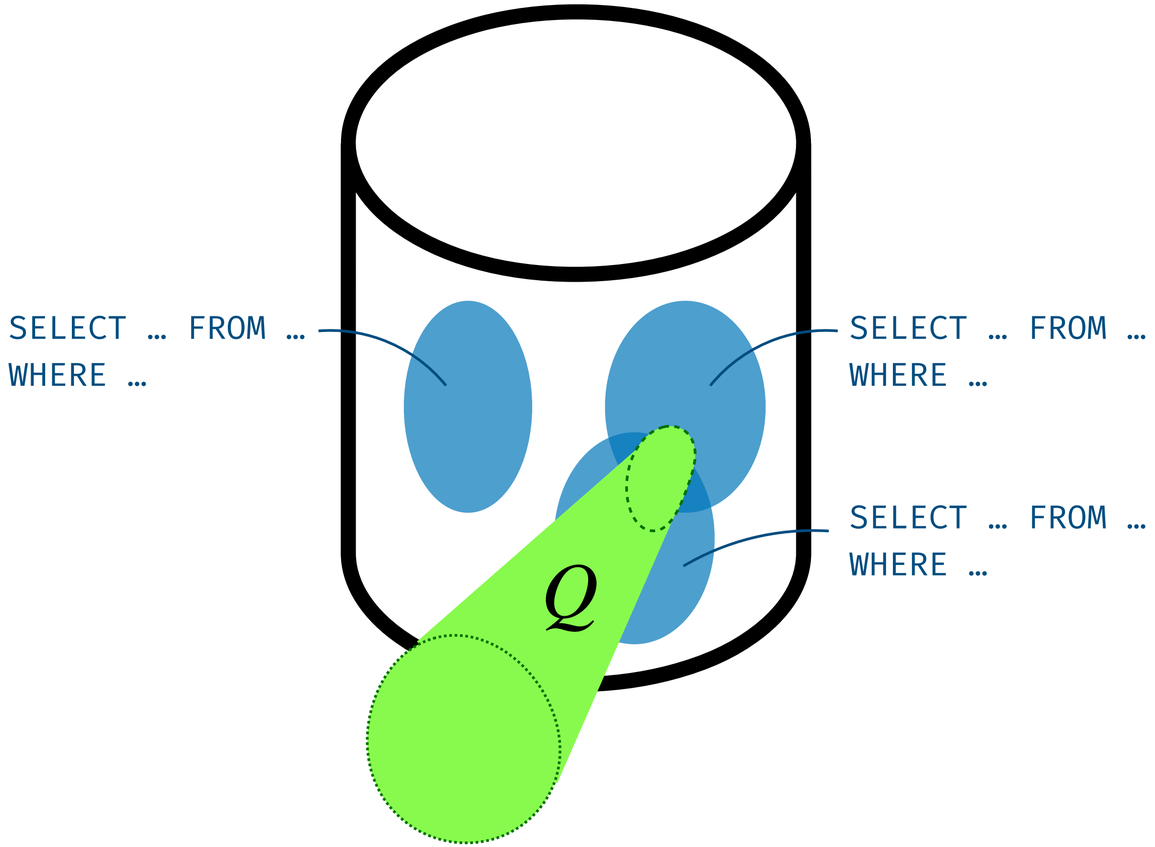
Specify a policy using database views

View: **SQL SELECT statement** defining data accessible to a user.



Specify a policy using database views

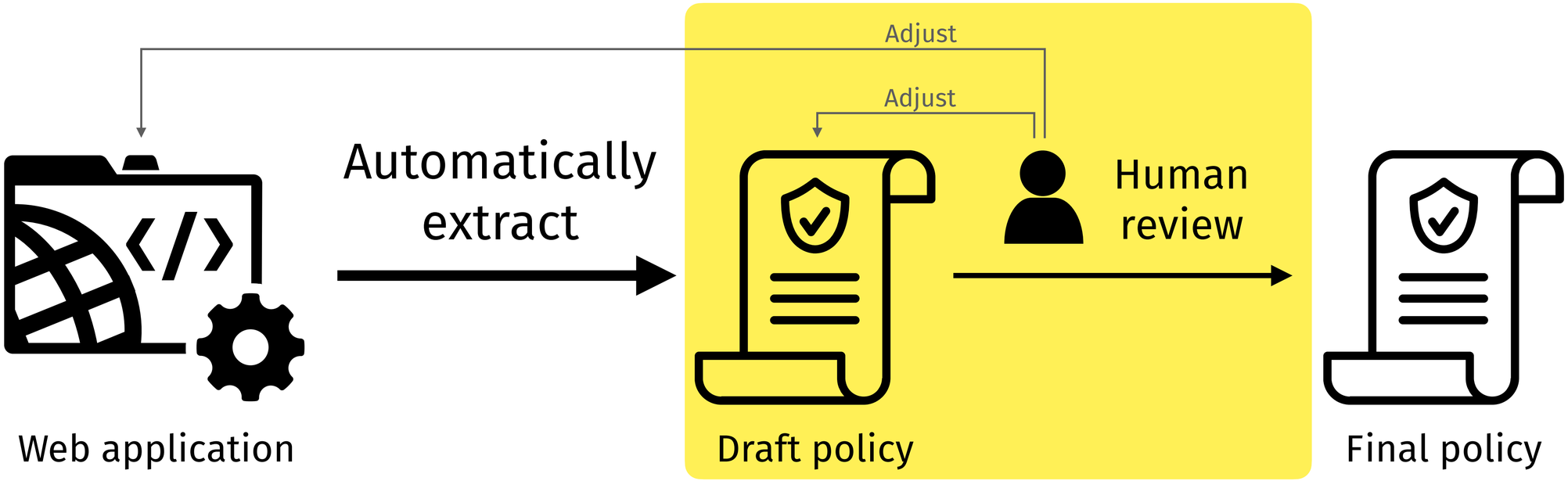
A query is **allowed** if it reveals only information exposed by the views.



Coming up next

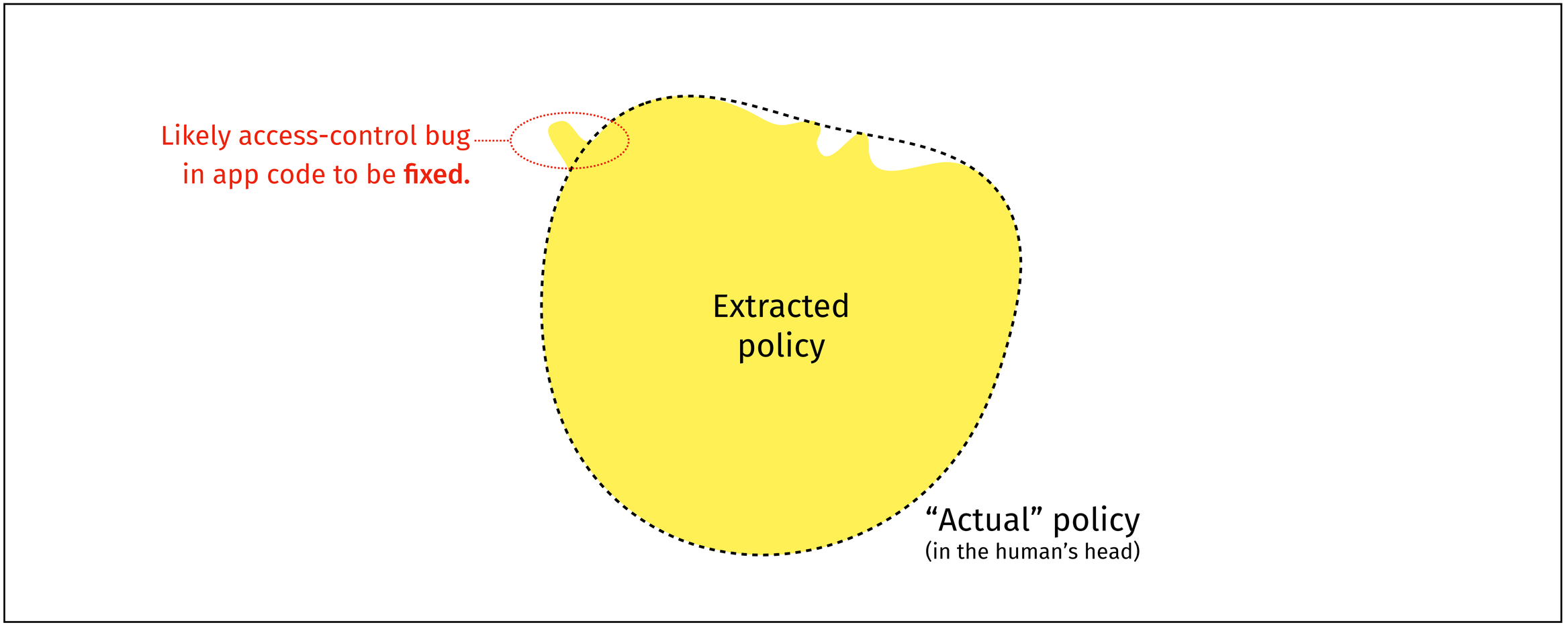
1. What form does a policy take in Ote?
2. What is the workflow for using Ote to extract a policy?
3. How does Ote extract a policy?
4. What did we learn from applying Ote?

Ote workflow



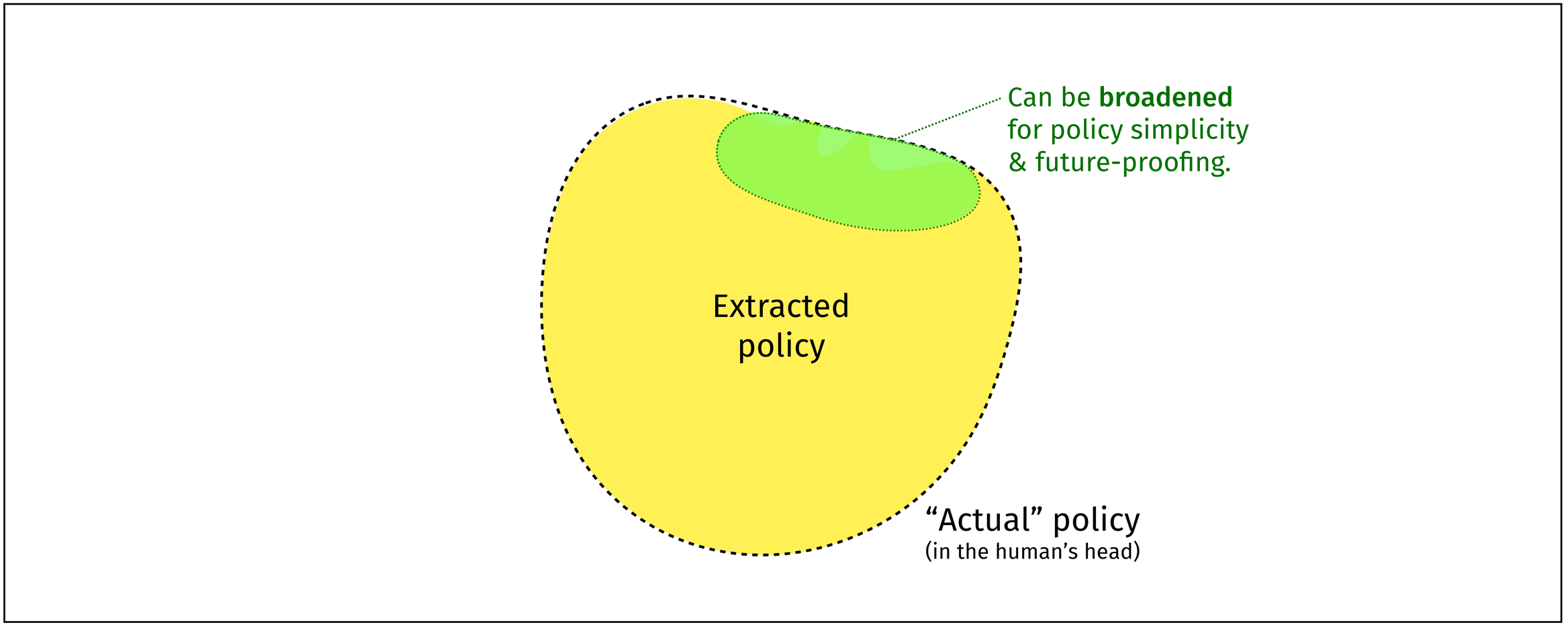
Goal: To help the human create the desired policy.

Draft policy and adjustments

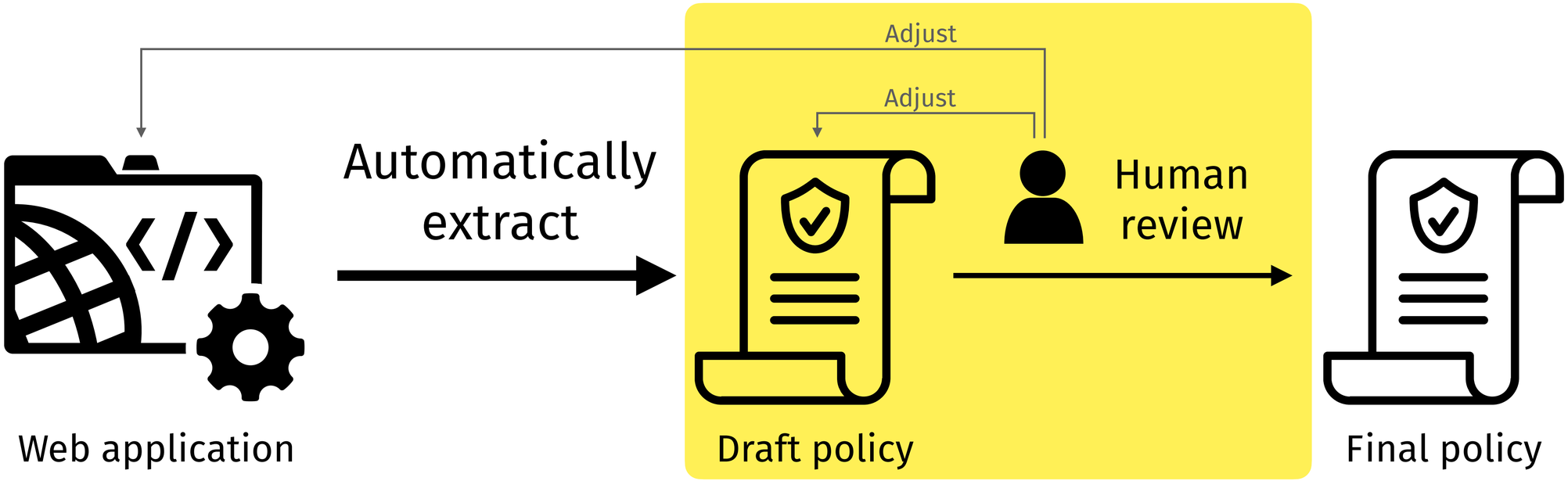


Draft policy and adjustments

All information in database



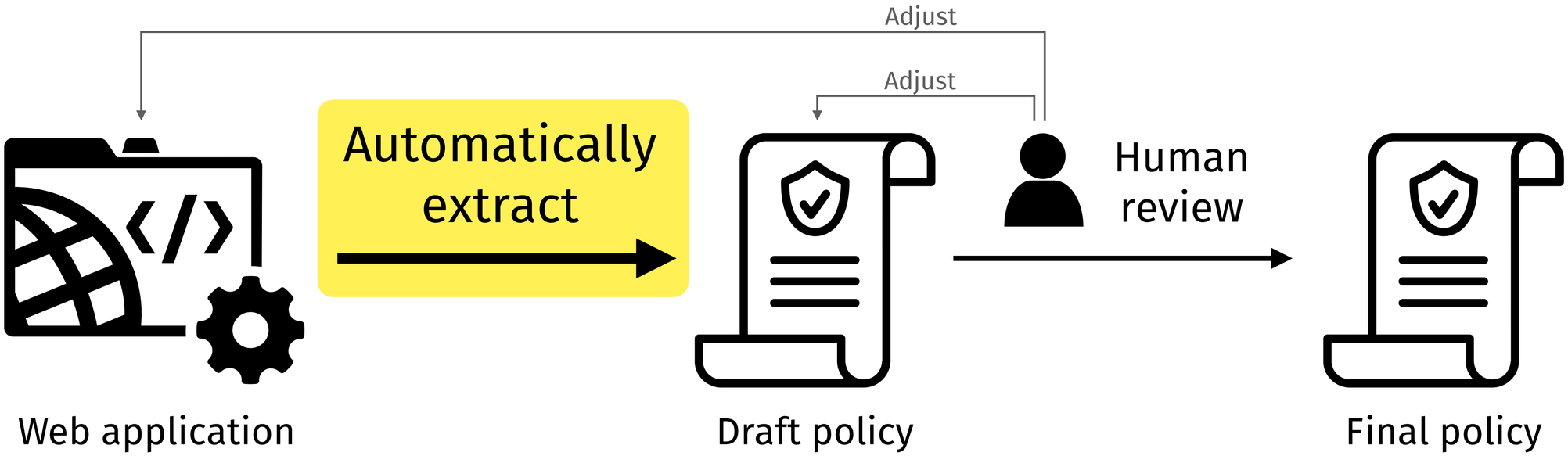
Ote workflow



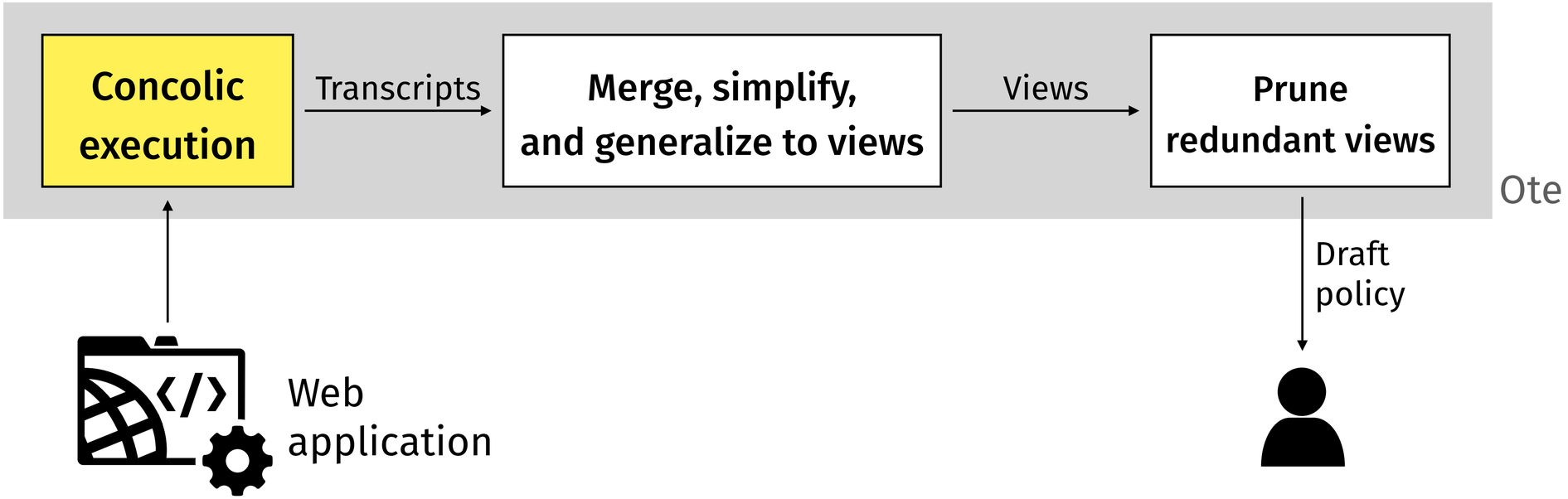
Coming up next

1. What form does a policy take in Ote?
2. What is the workflow for using Ote to extract a policy?
3. How does Ote extract a policy?
4. What did we learn from applying Ote?

Ote workflow

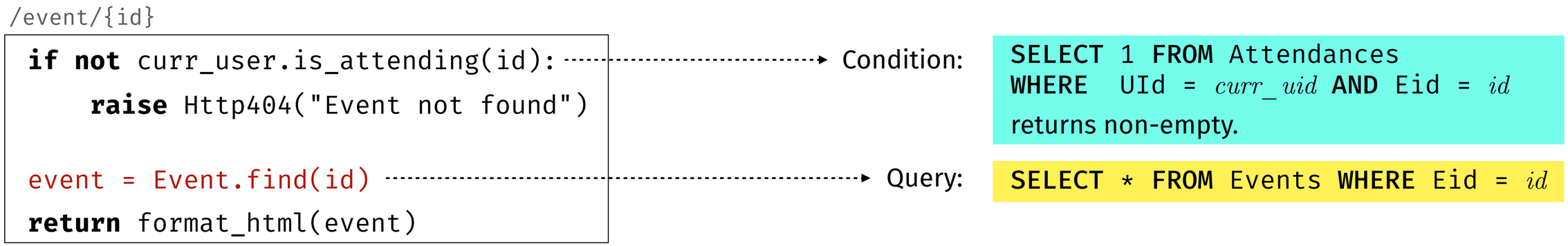


How does Ote extract a policy?



Generating transcripts from an application

Transcript contains: Which SQL queries are issued? Under what conditions?



Must track *data-flow* and *control-flow* into query-issuing statements.

Challenge: Web codebases are large and complex ➡ Hard to analyze!

Observation and approach

Observation: The query-issuing core tends to be simple [Chl10] [SR19].

↳ *Slice of the program that a SQL query may depend on.*

- This is the only part that Ote cares about!
- Catch: The core is scattered across the code base.

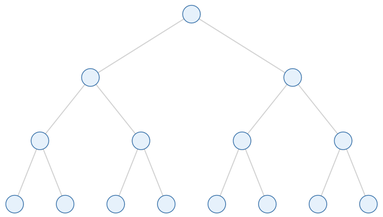
Approach: Use concolic execution to focus only on simple parts of the codebase.

- Symbolically instrument only the simple operations; the rest are ignored.

But: Even the simple parts have many branches!

Scaling problem: Running time is exponential in #branches

Optimization: Prune irrelevant branches



Observation: Many branches have *nothing to do* with data access!

```
<% if submission.version == 0 %>
  <font size=-2>Unofficial</font>
<% else %>
  <%= submission.version %>
<% end %>
```

Irrelevant to data access.

Idea: Skip exploration of irrelevant branches.

But how to identify irrelevant branches?

- *Typically pretty obvious* to a human…

Approach: Ask an LLM coding agent 🤖

- Accurate, and a huge time saver!

Coming up next

1. What form does a policy take in Ote?
2. What is the workflow for using Ote to extract a policy?
3. How does Ote extract a policy?
4. What did we learn from applying Ote?

What did we learn from applying Ote?

Used Ote to extract policies for **three existing applications:**

diaspora* AUTØLAB



THE ODIN PROJECT

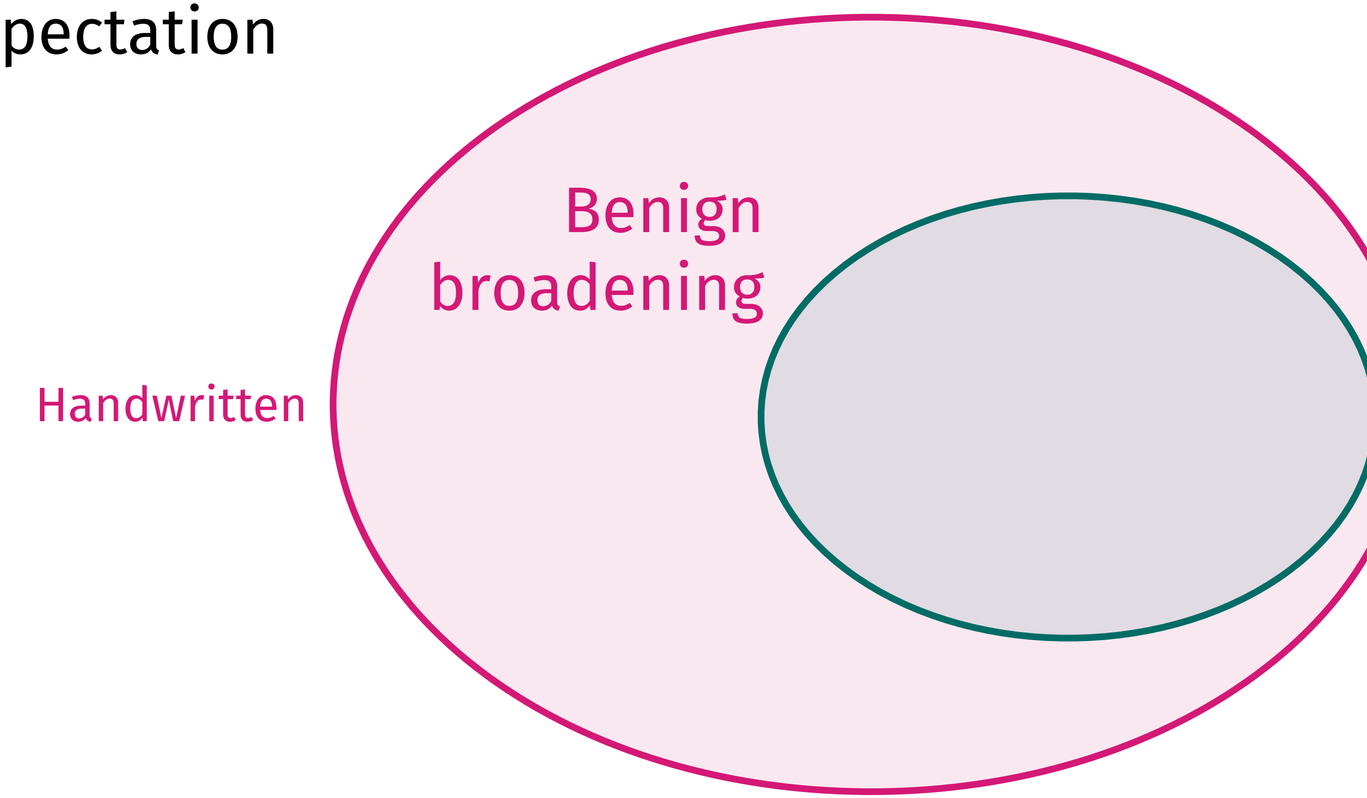
Previously wrote policies by hand.

Ote extracted each policy within five hours.

? Did I do a good job with the handwritten policies?

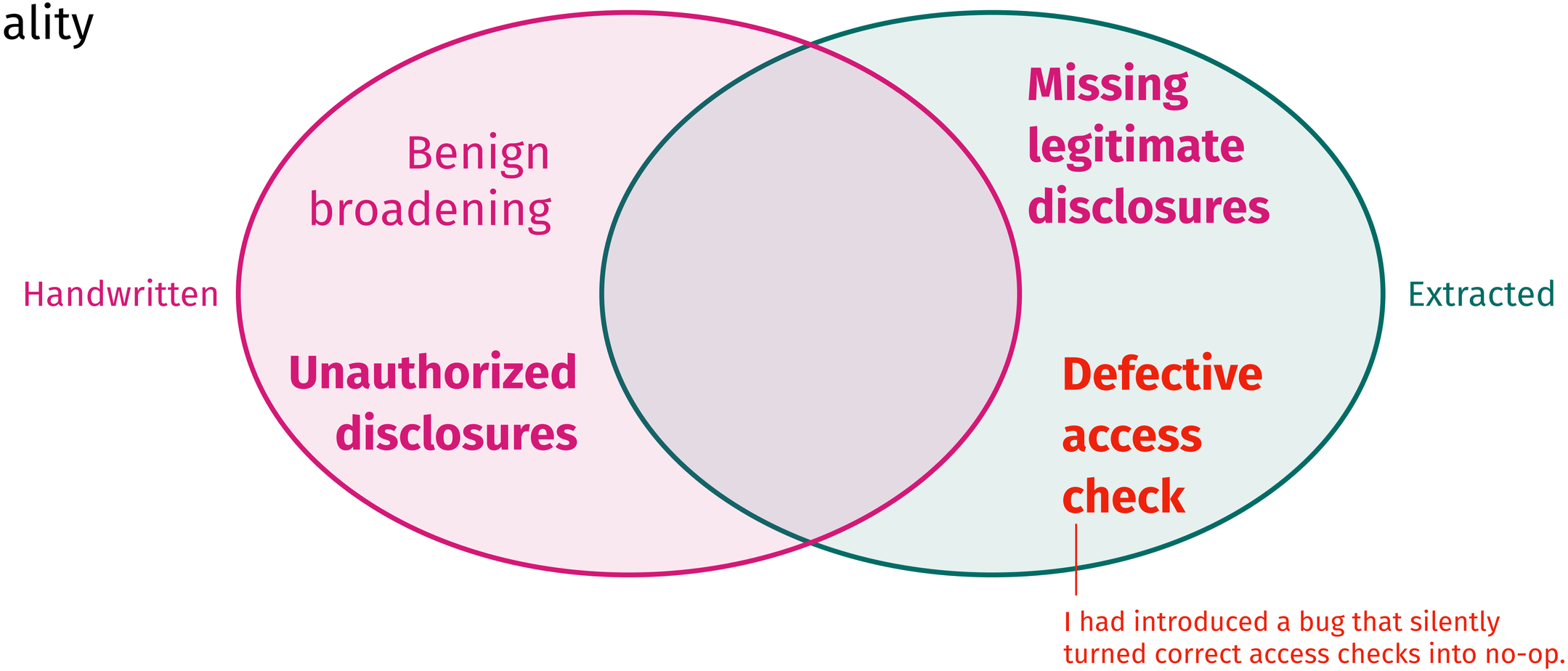
Handwritten vs extracted policy

Expectation

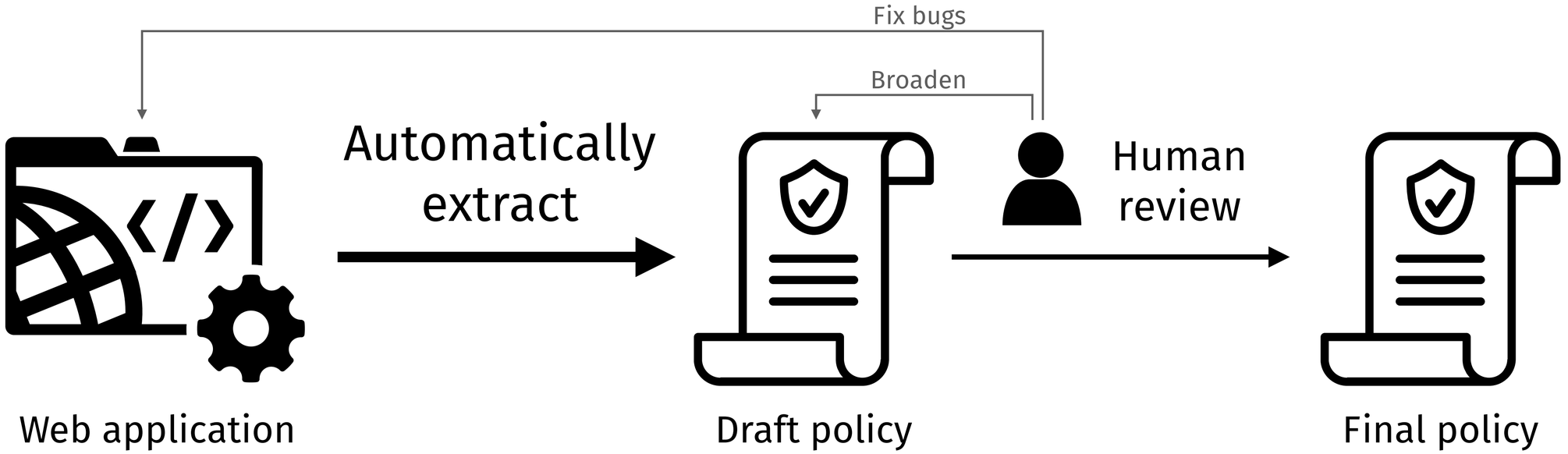


Handwritten vs extracted policy

Reality



Ote: Access-Control Policy Extraction



Thank you!