HCSS 2023

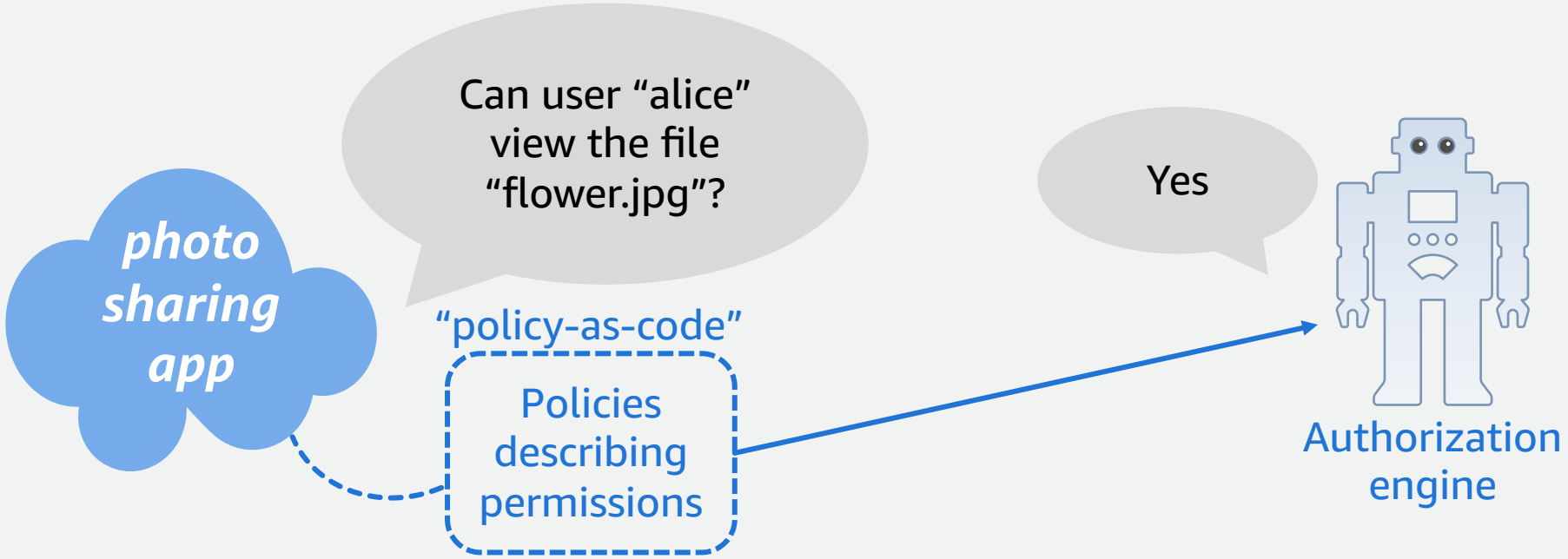
Verification-Guided Development of the Cedar Authorization Language

Kesha Hietala (she/her)

Applied Scientist
Amazon Web Services

Premise

- Authorization answers the question "*who has access to what*"?



- At Amazon we've been building CEDAR, a ***language*** for writing authorization policies and an ***engine*** for evaluating those policies.

Verification-guided development

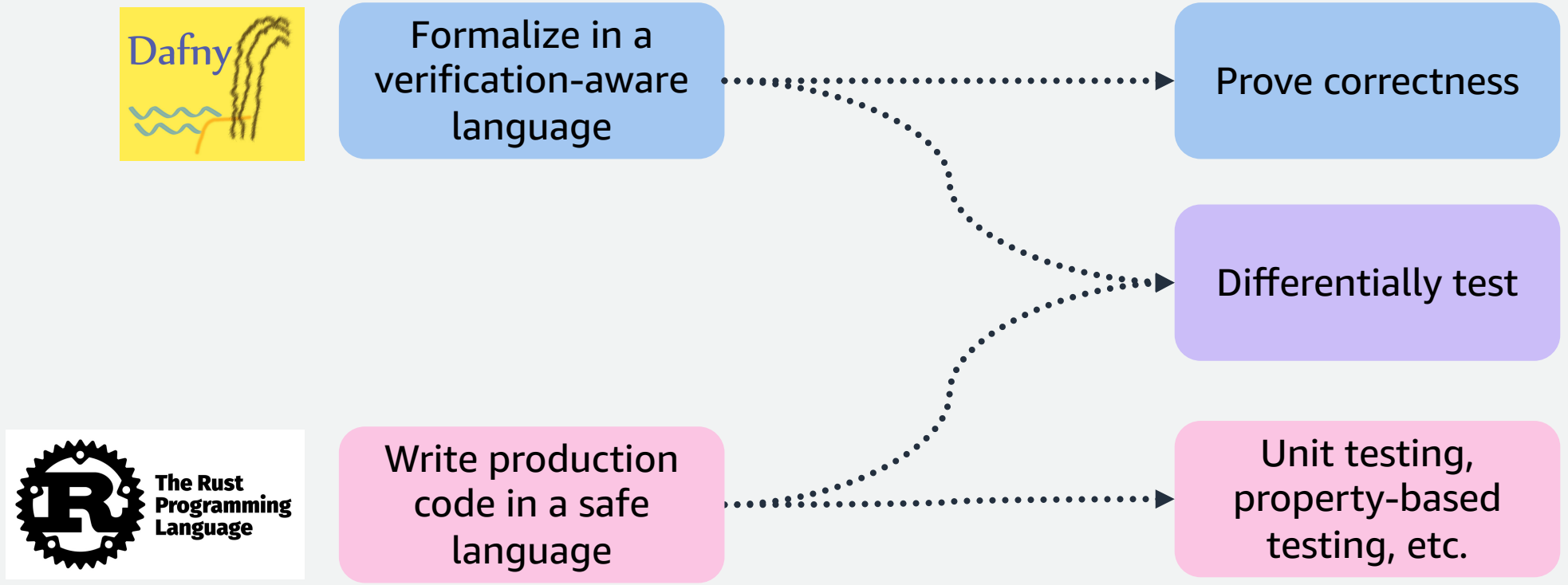
- Cedar is tasked with authorizing security-sensitive requests. How can we gain confidence that it does so correctly?

Requirements

- ✓ **Executable**
- ✗ **Fast** enough to run in production
- ✗ **Maintainable** by developers
- ✗ Supports utilities like parsing and file I/O

Verification-guided development

- Cedar is tasked with authorizing security-sensitive requests. How can we gain confidence that it does so correctly?



Outline

What is Cedar?

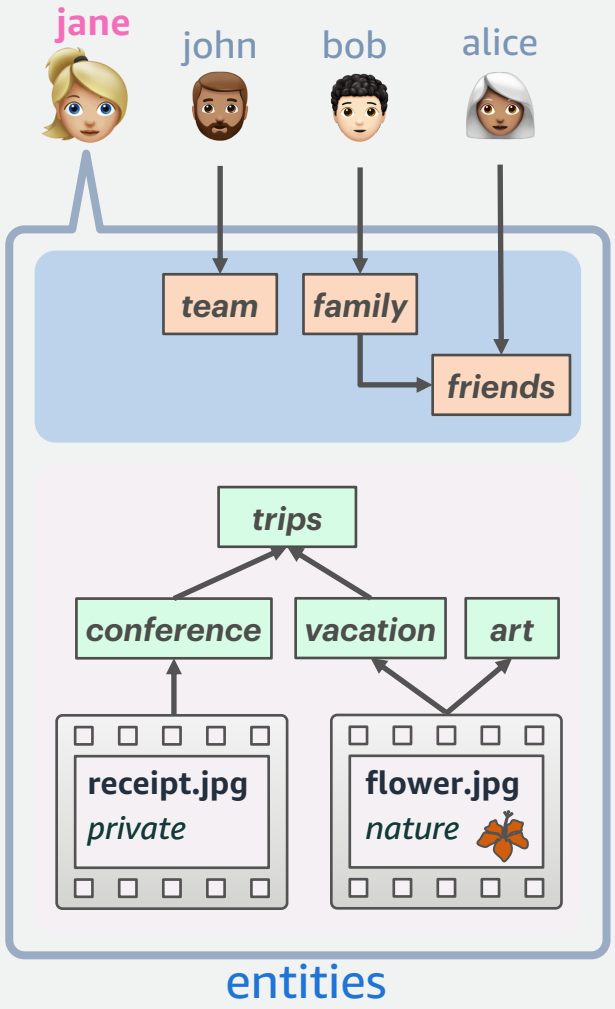
Verification-guided development

Formalization in Dafny

Differential random testing

Wrapping up

Example: Authorization with Cedar



entities

photo sharing app

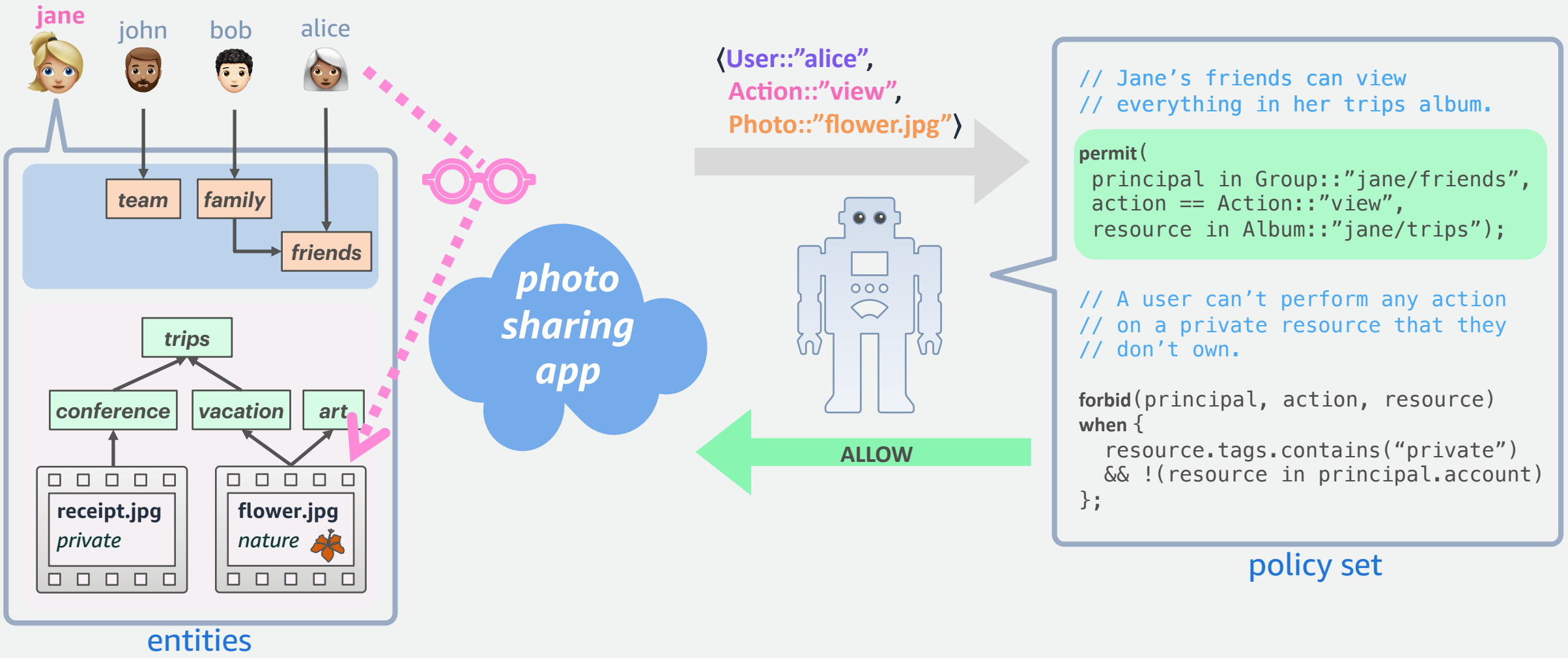
REQUEST

ALLOW / DENY

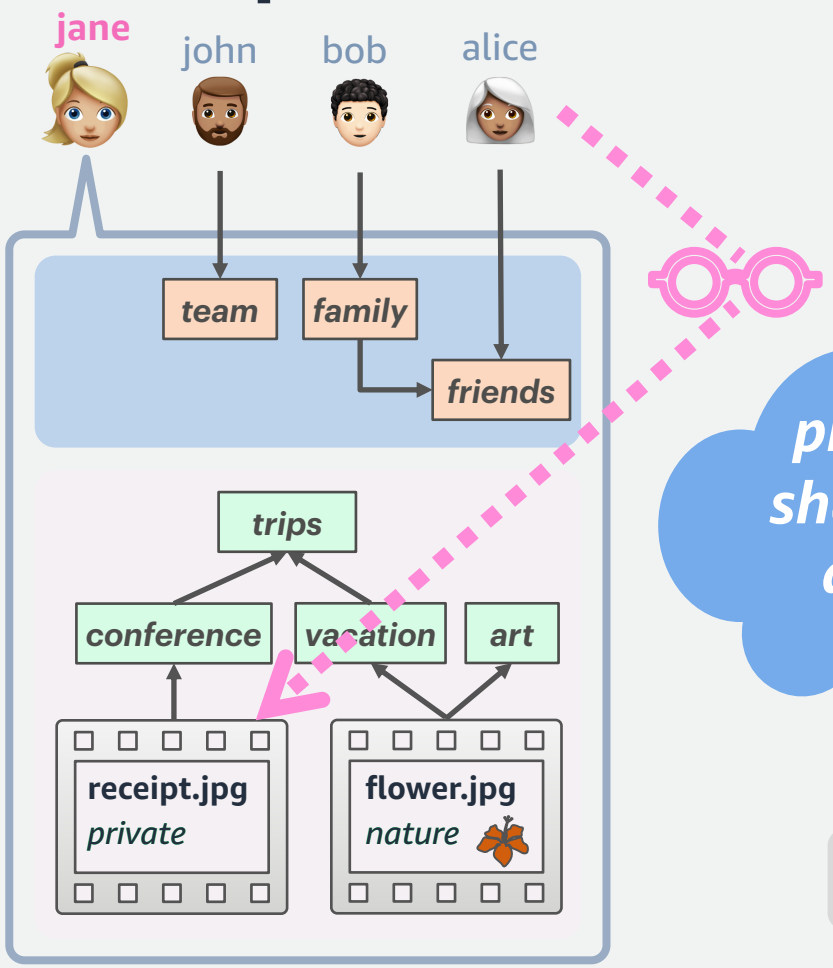
```
// Jane's friends can view
// everything in her trips album.

permit(
 principal in Group::"jane/friends",
 action == Action::"view",
 resource in Album::"jane/trips");


// A user can't perform any action
// on a private resource that they
// don't own.

forbid(principal, action, resource)
when {
  resource.tags.contains("private")
  && !(resource in principal.account)
};
```

policy set

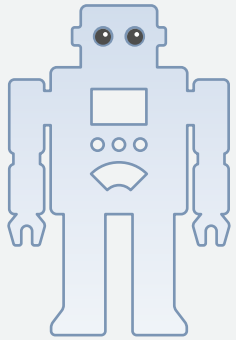
Example: Authorization with Cedar



Example: Authorization with Cedar



⟨User::"alice",
Action::"view",
Photo::"receipt.jpg"⟩



DENY

forbid overrides **permit**

```
// Jane's friends can view
// everything in her trips album.
permit(
 principal in Group::"jane/friends",
 action == Action::"view",
 resource in Album::"jane/trips");

// A user can't perform any action
// on a private resource that they
// don't own.
forbid(principal, action, resource)
when {
  resource.tags.contains("private")
  && !(resource in principal.account)
};
```

policy set

Preventing errors with policy validation

When a policy contains an **error**, it is **ignored** during evaluation

Policy validation checks that a policy is consistent with a user-provided schema

The **schema** describes the types of entities used in the application

```
permit(                Group
  principal in ~~Album~~::"jane/friends",
  action == Action::"view",
  resource in ~~Albom~~::"jane/trips");
                       Album
```

Preventing errors with policy validation

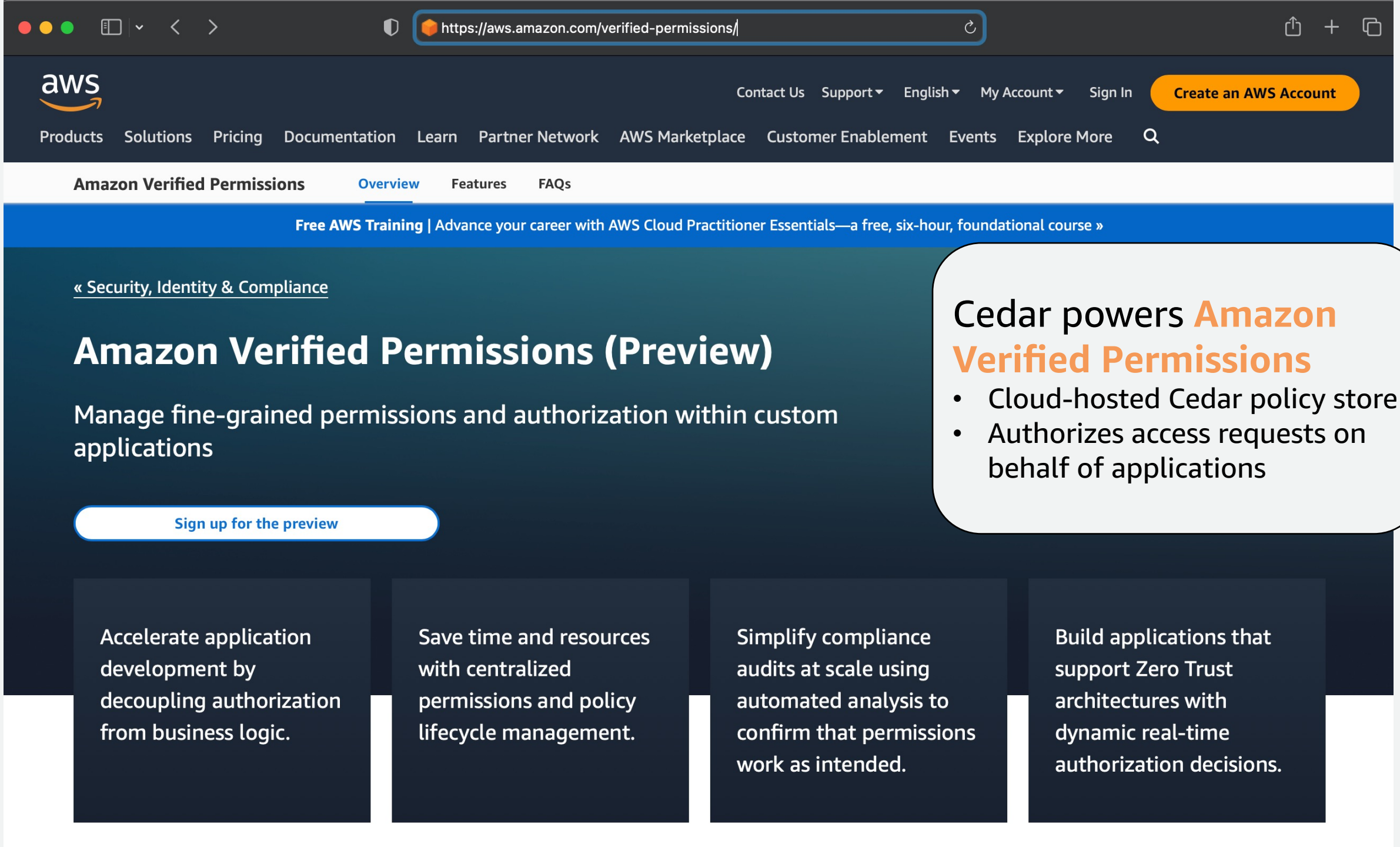
When a policy contains an **error**, it is **ignored** during evaluation

Policy validation checks that a policy is consistent with a user-provided schema

The **schema** describes the types of entities used in the application

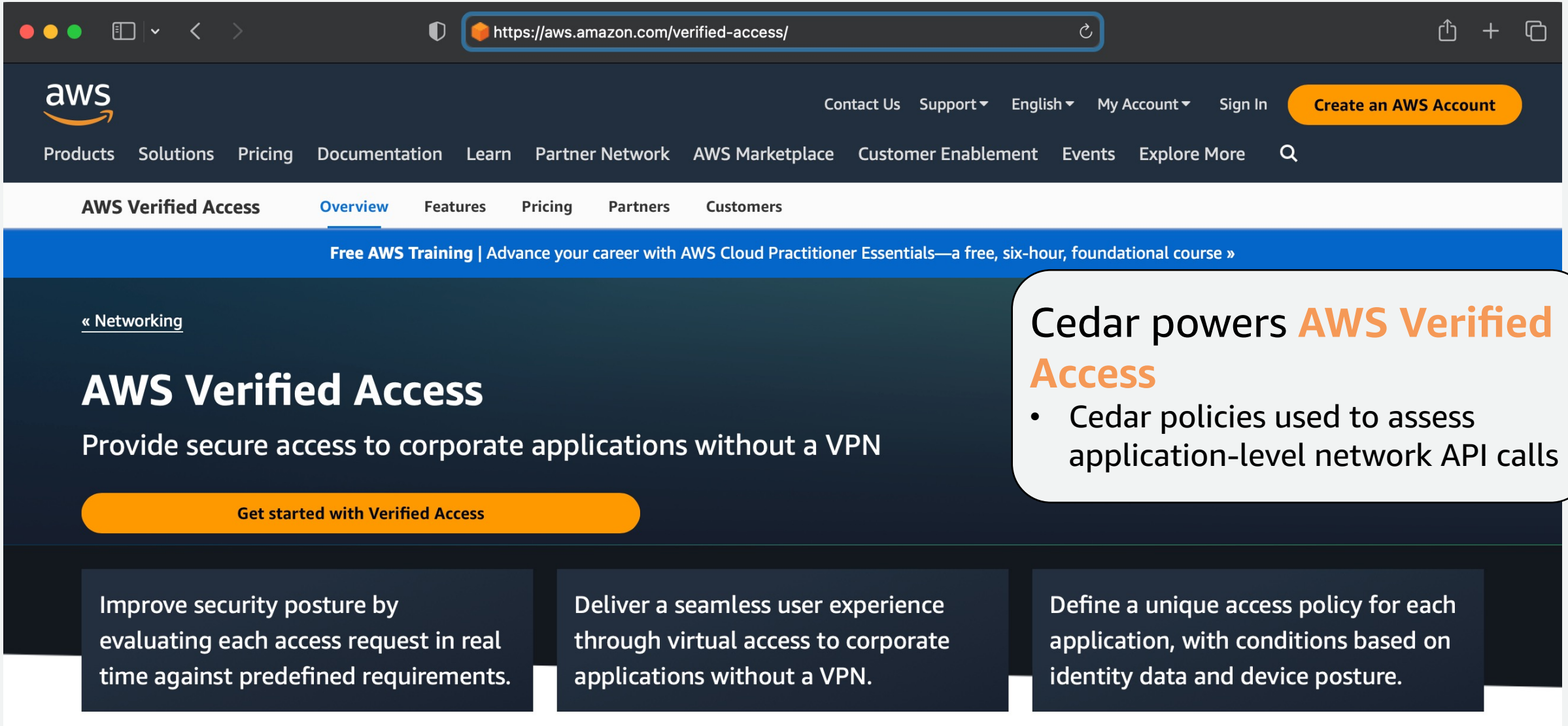
```
permit(
 principal in Group::"jane/friends",
 action == Action::"view",
 resource in Album::"jane/trips")
when {
 principal.jobLevel > "5"
};                    5
```

Theorem: If the policy set is valid according to the schema, and the schema is consistent with the entities and request, then evaluating the request will produce **no type errors**



Cedar powers **Amazon Verified Permissions**

- Cloud-hosted Cedar policy store
- Authorizes access requests on behalf of applications



Cedar powers **AWS Verified Access**

- Cedar policies used to assess application-level network API calls

Outline

What is Cedar?

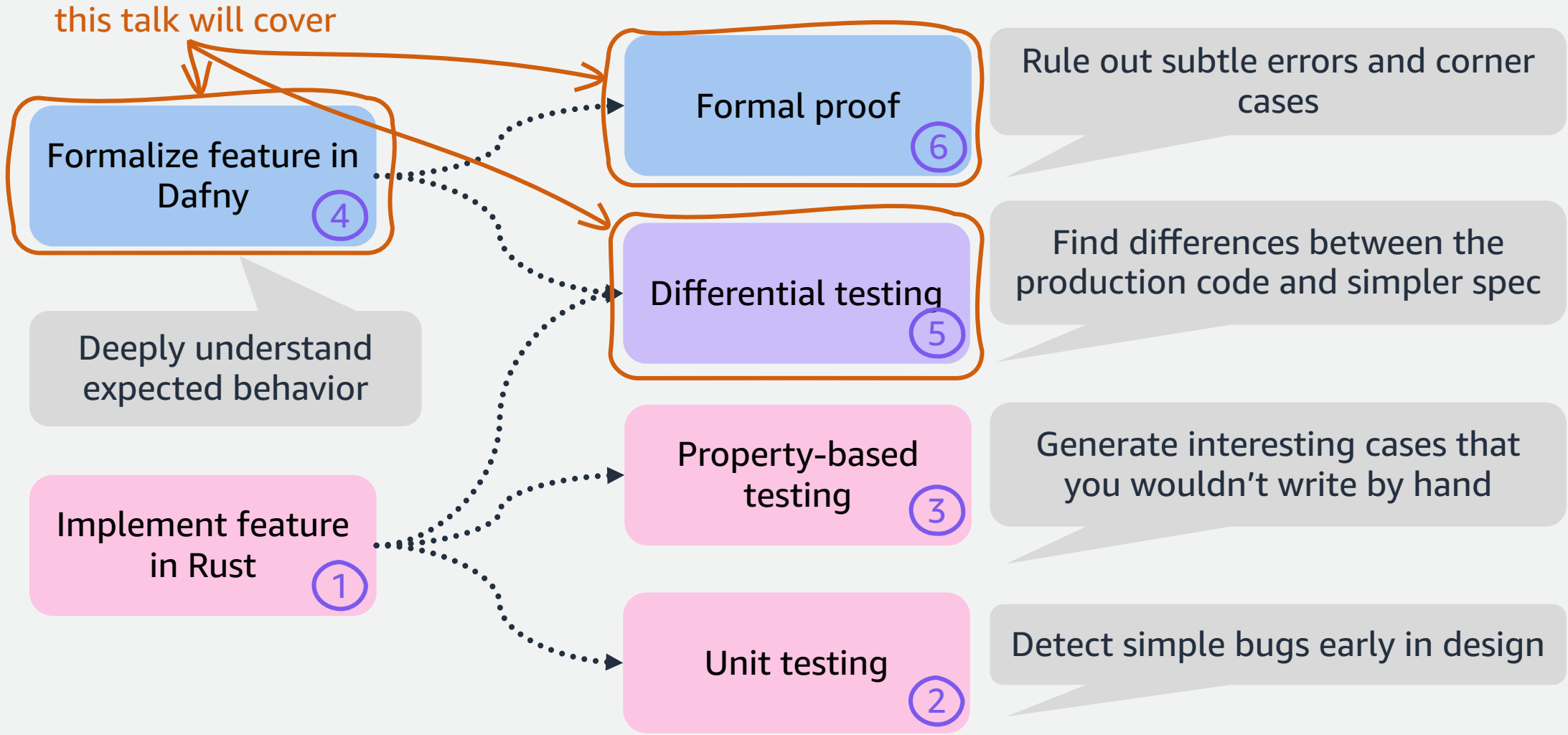
Verification-guided development

Formalization in Dafny

Differential random testing

Wrapping up

Verification-guided development



Outline

What is Cedar?

Verification-guided development

Formalization in Dafny

Differential random testing

Wrapping up

Step 1: Define reference model

concise specification

```
datatype Authorizer = Authorizer(request : Request, store: Store) {

    function evaluate(pid: PolicyID): Result<Value>
    { ... }

    function forbids(): set<PolicyID> {
      set pid | evaluate(pid) == Ok(Value.True) &&
        store.policies.policies[pid].effect == Forbid
    }

    function permits(): set<PolicyID> {
      set pid | evaluate(pid) == Ok(Value.True) &&
        store.policies.policies[pid].effect == Permit
    }

    function isAuthorized(): Response {
      var f := forbids();
      var p := permits();
      if f == {} && p != {} then
        Response(Allow, p)
      else
        Response(Deny, f)
    }
}
```

The authorizer is ~200 LOC

In total our spec is ~2800 LOC

Step 2: Prove properties

formal proofs of key properties

```
lemma ForbidOverridesPermit(request: Request, store: Store)
  requires // If some forbid policy evaluates to true, then
    exists f ::
      f in store.policies.policies.Keys &&
      store.policies.policies[f].effect == Forbid ::
      Authorizer(request, store).evaluate(f) == Ok(True)
  ensures // the request is denied.
    Authorizer(request, store).isAuthorized().decision == Deny
{ }
```

```
lemma TypeSoundness(pid: PolicyID, request : Request,
                    store: Store, schema: Schema)
  requires pid in store.policies.policies.Keys
  requires SatisfiesSchema(request, store.entities, schema)
  requires typecheck(pid, store.policies, schema).Ok?
  ensures
    var res := Authorizer(request, store).evaluate(pid);
    res != Err(TypeError)
{ ... }
```

Our proofs are ~2700 LOC

Outline

What is Cedar?

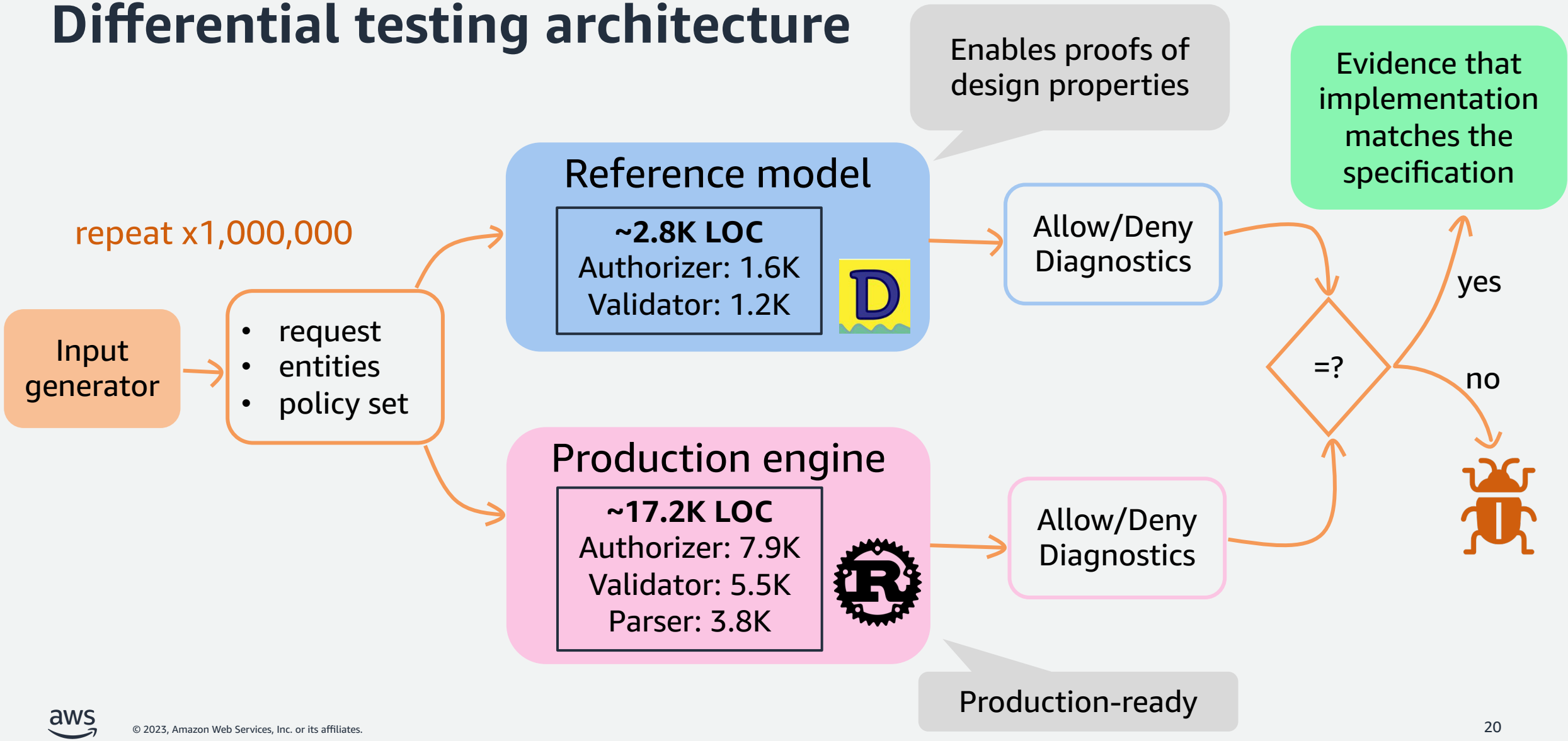
Verification-guided development

Formalization in Dafny

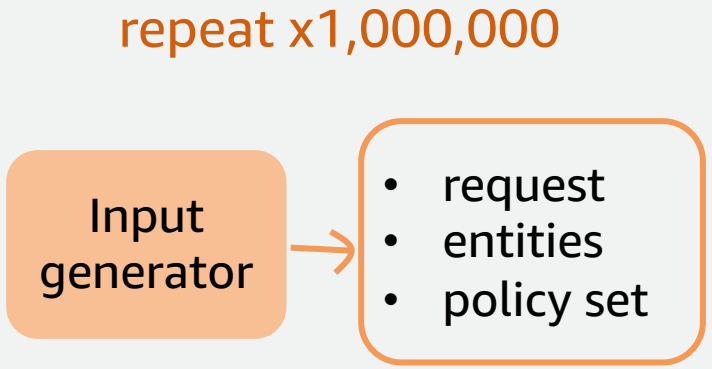
Differential random testing

Wrapping up

Differential testing architecture



Input generation



We use custom generators to produce random (but *well-formed*) inputs & use coverage-guided mutations

For some targets, we weight the generators to produce **well-typed** inputs to achieve better coverage

We run differential testing nightly for 6 hours, and generate on the order of **100M tests**

Outline

What is Cedar?

Verification-guided development

Formalization in Dafny

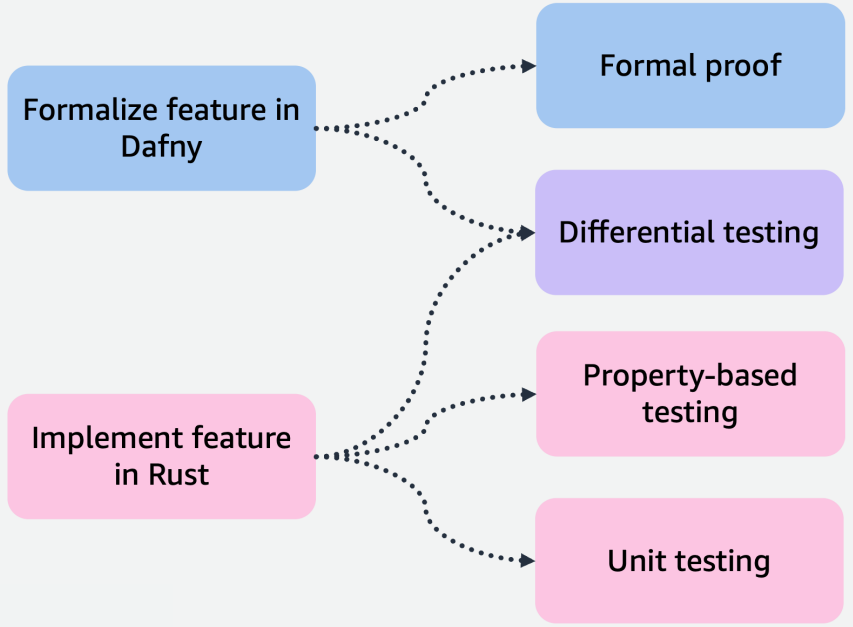
Differential random testing

Wrapping up

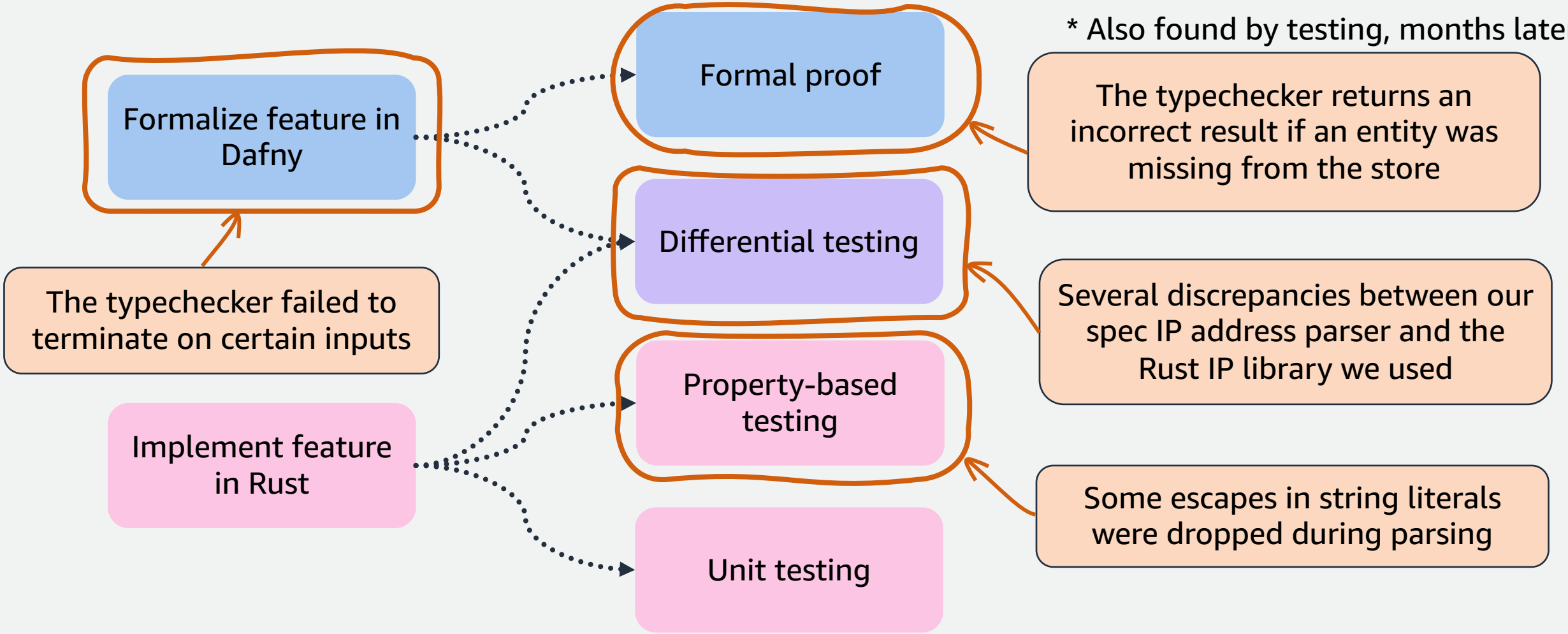
Summary

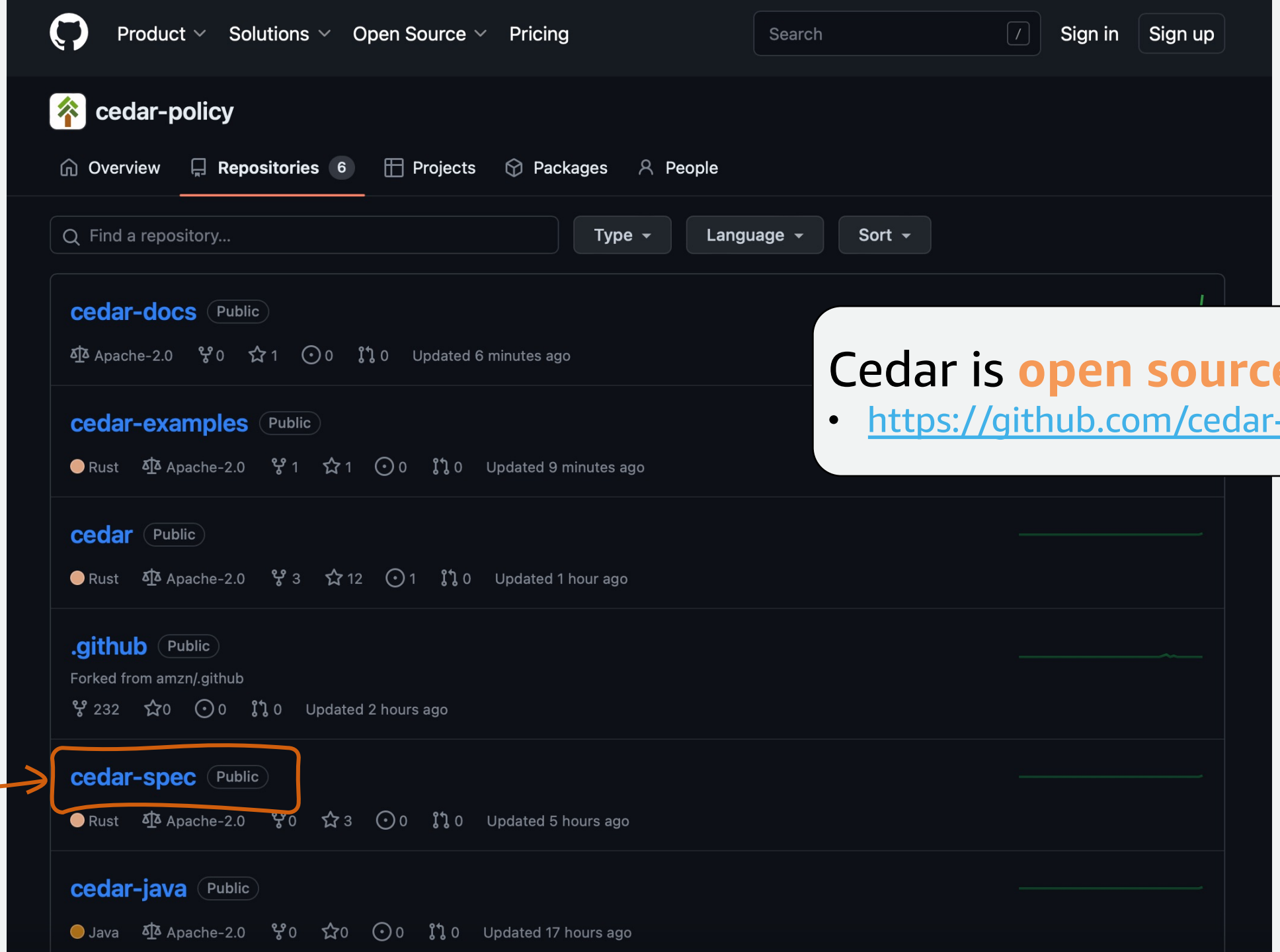
CEDAR is a ***language*** for writing authorization policies, and an ***engine*** for evaluating those policies.

We use *verification-guided development* to provide a high assurance implementation for the Cedar language.



Results: Bugs found and fixed





Dafny spec + DRT code

Cedar is **open source**

- https://github.com/cedar-policy

Thank you!

The Cedar Language Team

https://www.cedarpolicy.com/

https://github.com/cedar-policy

Craig Disselkoen
Aaron Eline
Shaobo He
Kesha Hietala
Mike Hicks
John Kastner
Anwar Mamat
Matt McCutchen
Emina Torlak
Andrew Wells