



HCSS 2023

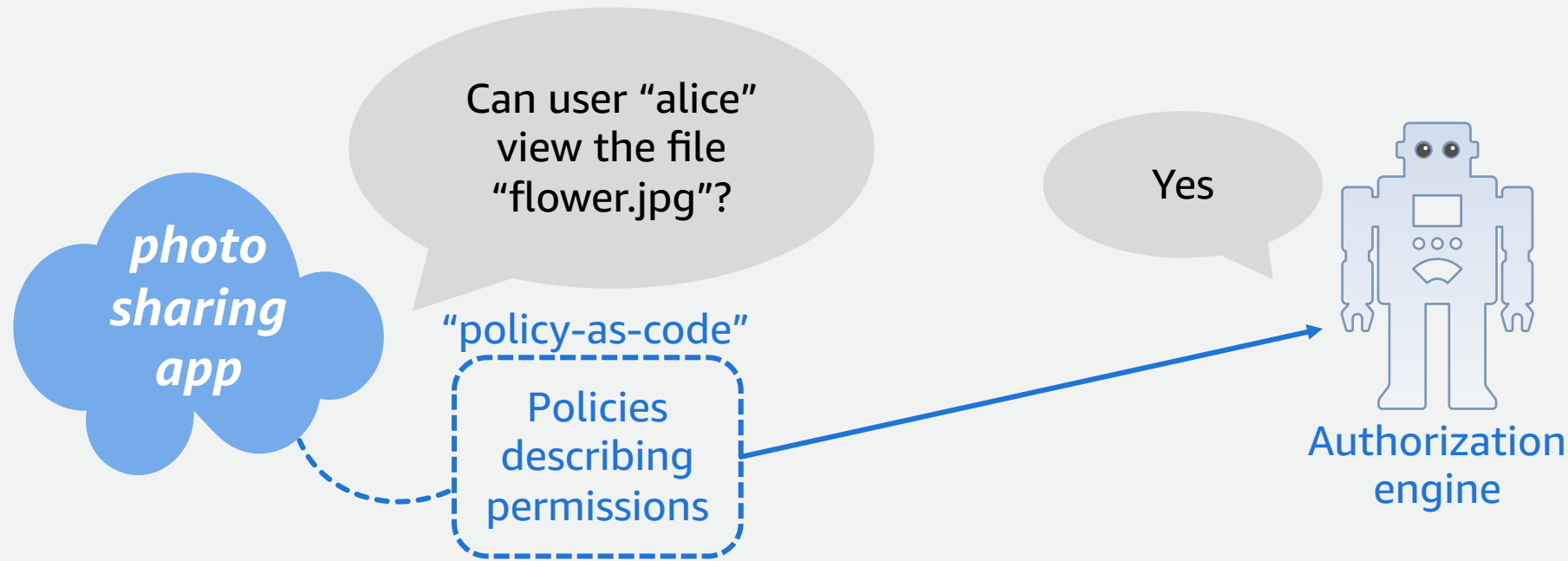
Verification-Guided Development of the Cedar Authorization Language

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?

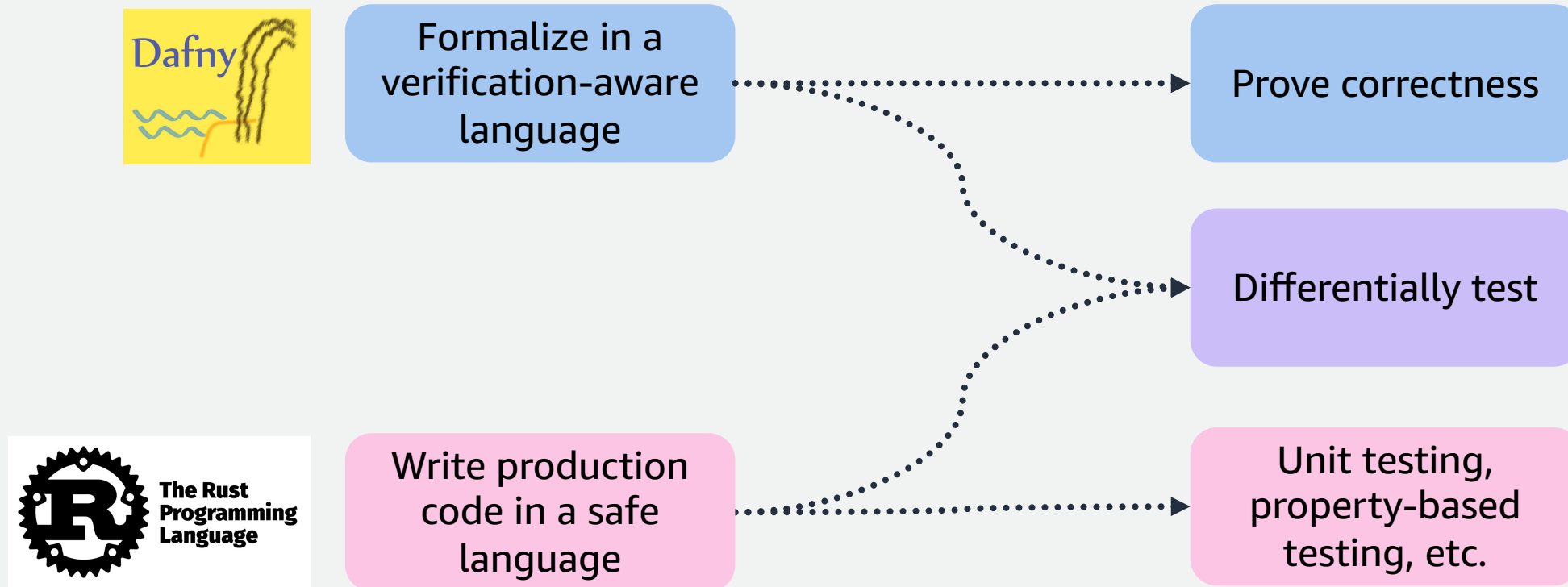


Requirements

- ✓ **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?

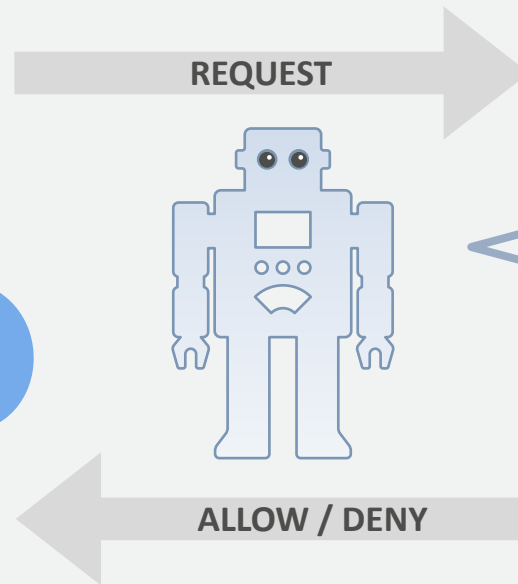
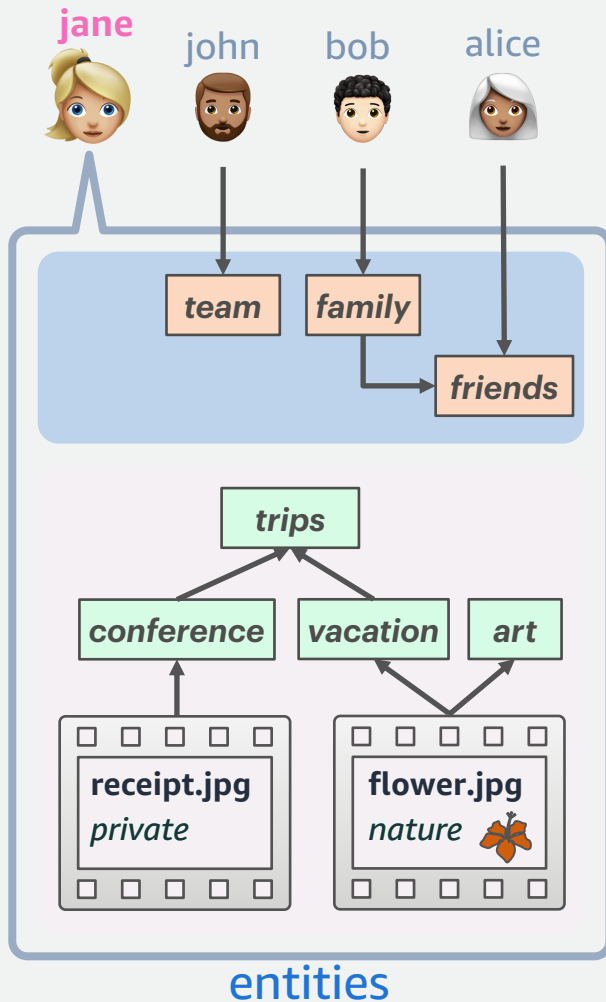
Verification-guided development

Formalization in Dafny

Differential random testing

Wrapping up

Example: Authorization with Cedar



```

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

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

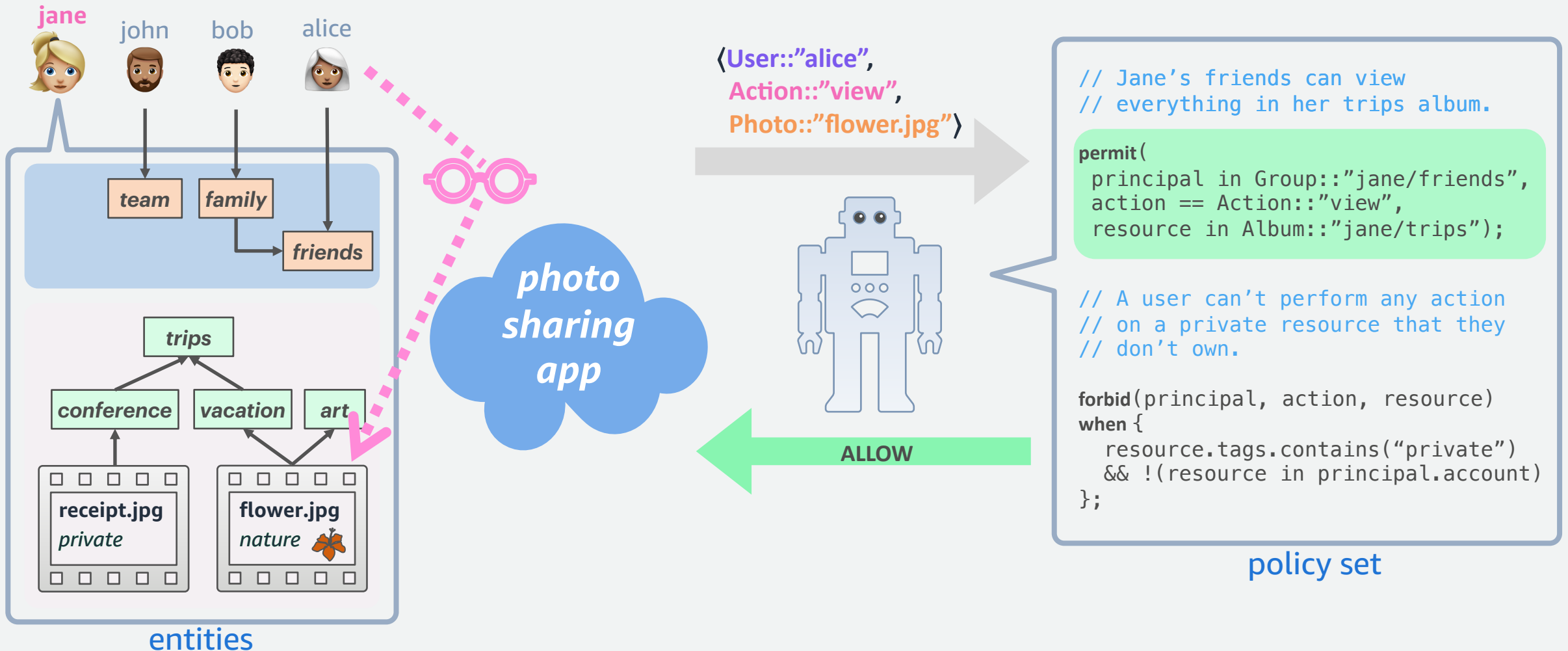
// A user can't perform any action
// on a private resource that they
// don't own.

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

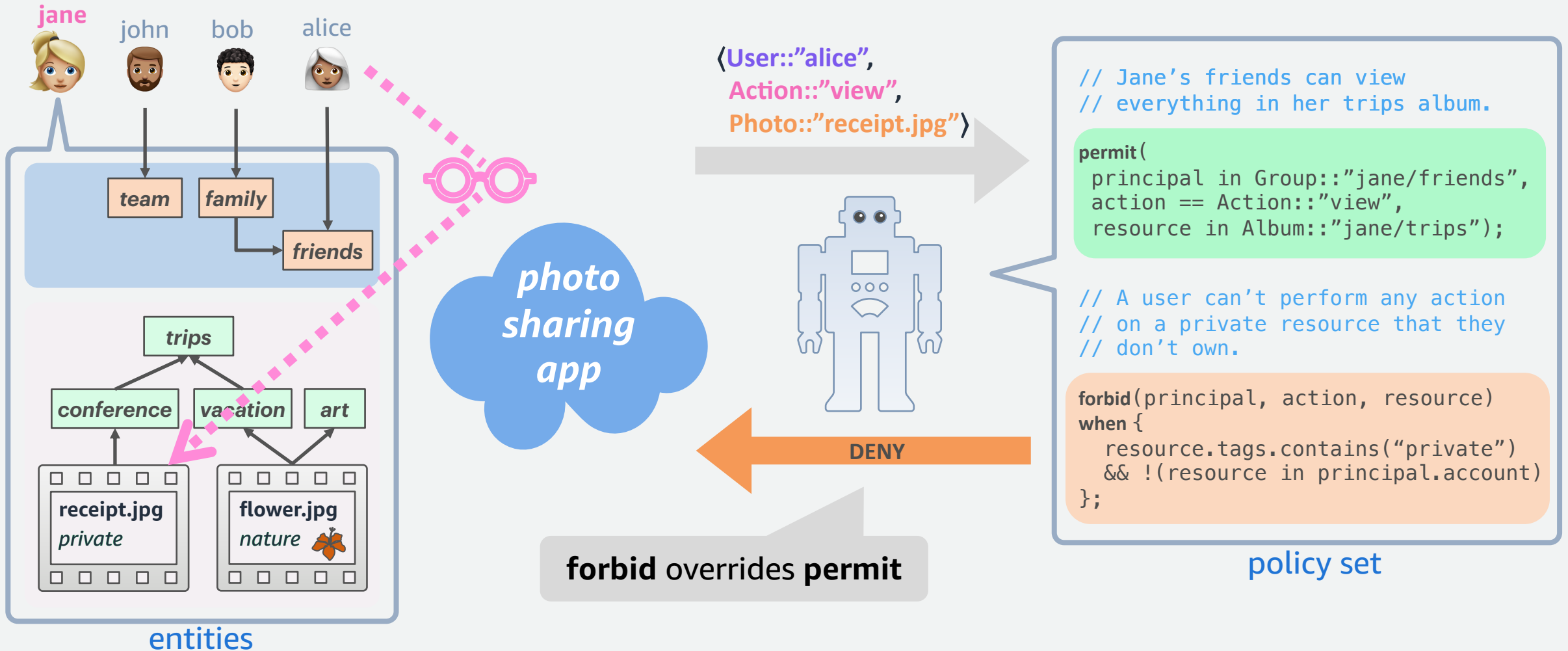
```

policy set

Example: Authorization with Cedar



Example: Authorization with Cedar



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 Album::"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"  
};
```

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**



Free AWS Training | Advance your career with AWS Cloud Practitioner Essentials—a free, six-hour, foundational course »

« Security, Identity & Compliance

Amazon Verified Permissions (Preview)

Manage fine-grained permissions and authorization within custom applications

Sign up for the preview

Cedar powers **Amazon Verified Permissions**

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

Accelerate application development by decoupling authorization from business logic.

Save time and resources with centralized permissions and policy lifecycle management.

Simplify compliance audits at scale using automated analysis to confirm that permissions work as intended.

Build applications that support Zero Trust architectures with dynamic real-time authorization decisions.

Browser address bar: <https://aws.amazon.com/verified-access/>

Navigation: Products Solutions Pricing Documentation Learn Partner Network AWS Marketplace Customer Enablement Events Explore More

Account: Contact Us Support English My Account Sign In [Create an AWS Account](#)

Sub-navigation: **AWS Verified Access** Overview Features Pricing Partners Customers

Banner: Free AWS Training | Advance your career with AWS Cloud Practitioner Essentials—a free, six-hour, foundational course »

« [Networking](#)

AWS Verified Access

Provide secure access to corporate applications without a VPN

[Get started with Verified Access](#)

Cedar powers AWS Verified Access

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

Improve security posture by evaluating each access request in real time against predefined requirements.

Deliver a seamless user experience through virtual access to corporate applications without a VPN.

Define a unique access policy for each application, with conditions based on identity data and device posture.

Outline

What is Cedar?

Verification-guided development

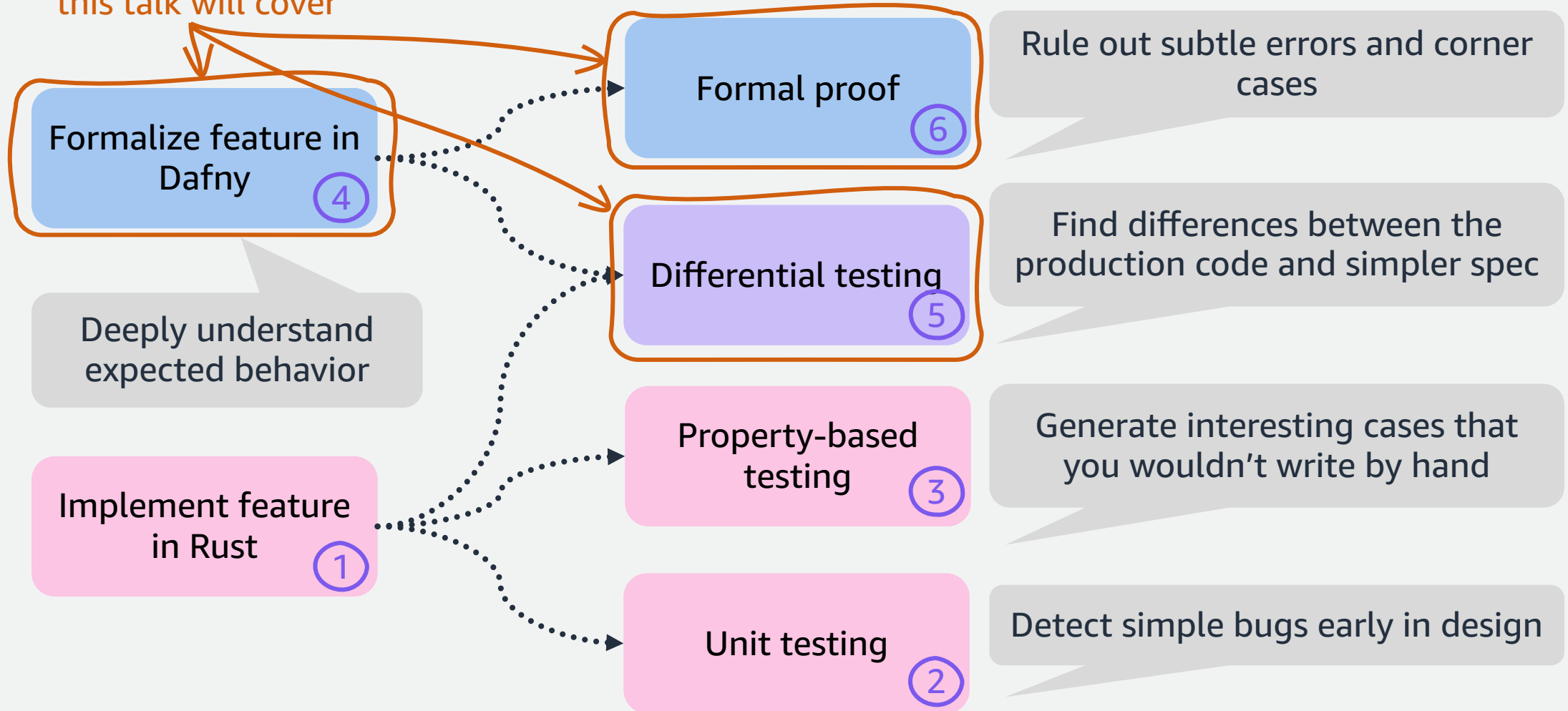
Formalization in Dafny

Differential random testing

Wrapping up

Verification-guided development

this talk will cover



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

```
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)
{ ... }
```

formal proofs of
key properties

Our proofs are
~2700 LOC

Outline

What is Cedar?

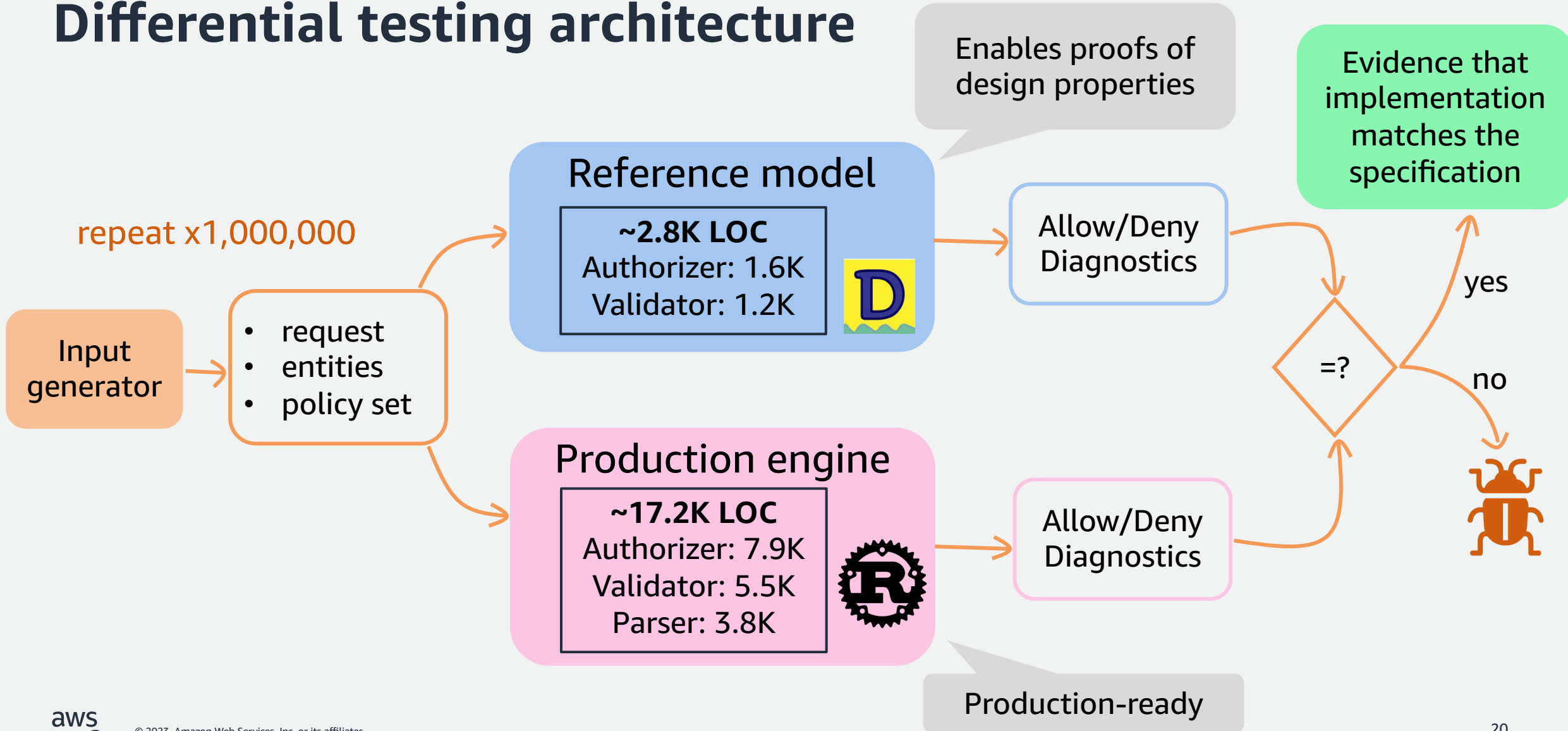
Verification-guided development

Formalization in Dafny

Differential random testing

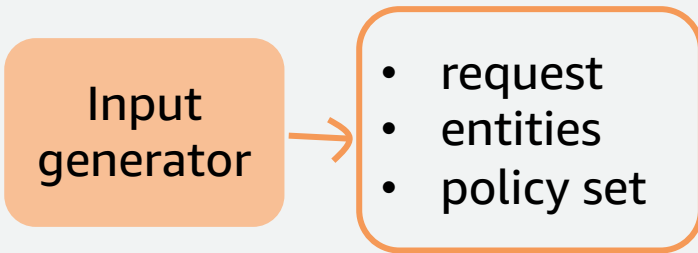
Wrapping up

Differential testing architecture



Input generation

repeat x1,000,000



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

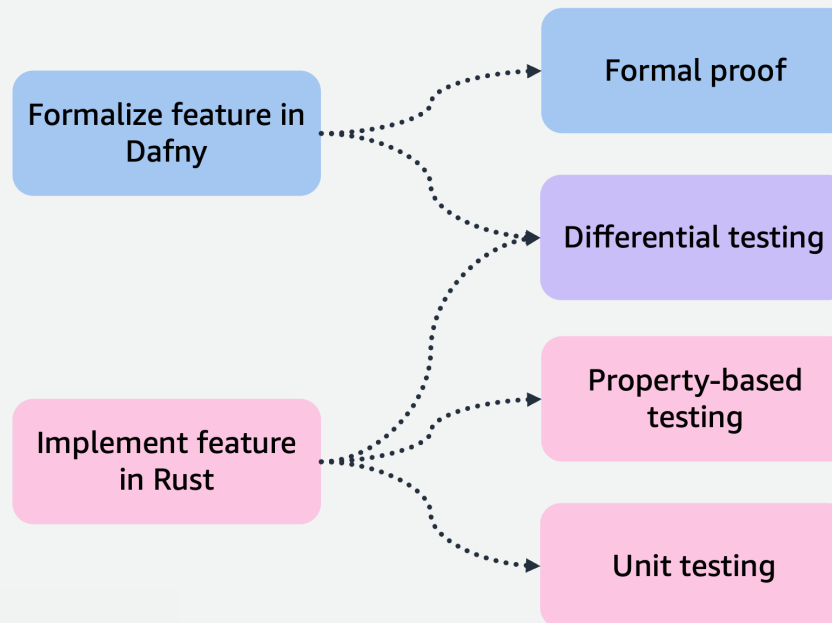
Differential random testing

Wrapping up

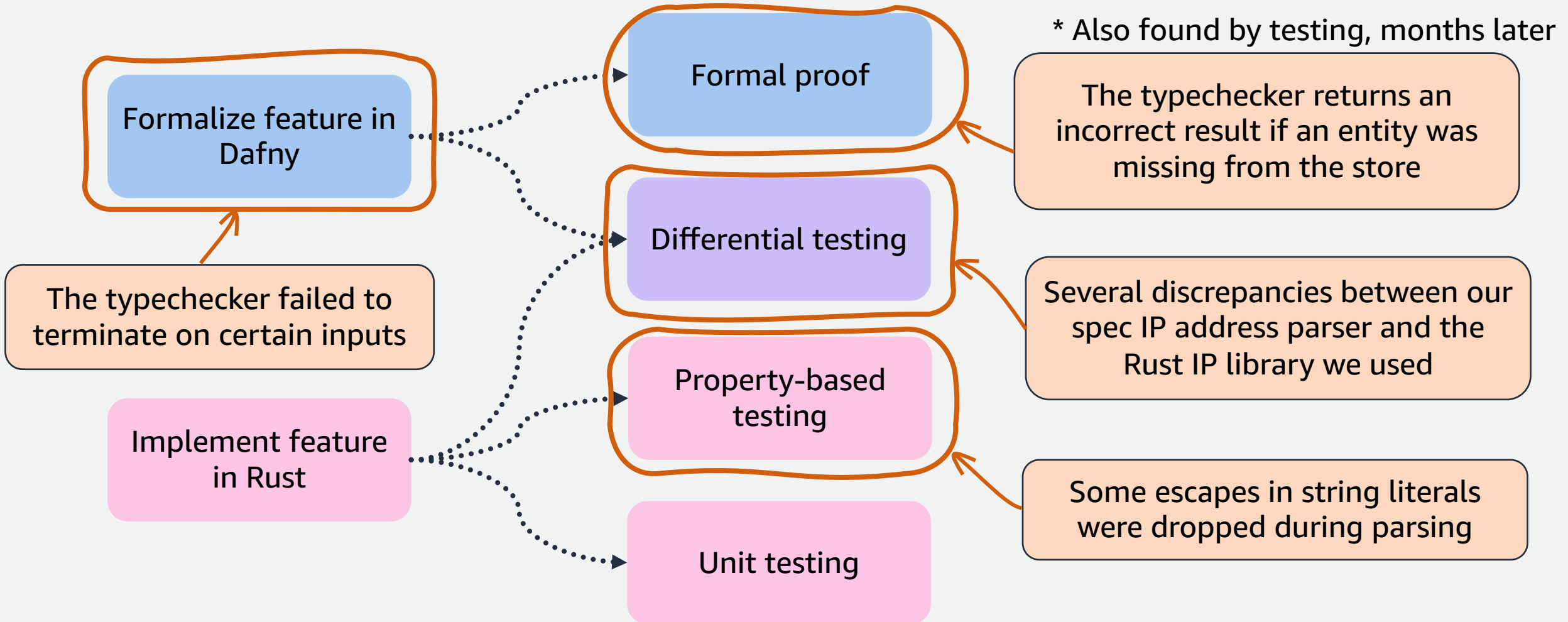
Summary

 **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





cedar-policy

Find a repository...

Type

Language

Sort

cedar-docs

Public

Apache-2.0 0 1 0 0 Updated 6 minutes ago

cedar-examples

Public

Rust Apache-2.0 1 1 0 0 Updated 9 minutes ago

cedar

Public

Rust Apache-2.0 3 12 1 0 Updated 1 hour ago

.github

Public

Forked from amzn/github
232 0 0 0 Updated 2 hours ago

cedar-spec

Public

Rust Apache-2.0 0 3 0 0 Updated 5 hours ago

cedar-java

Public

Java Apache-2.0 0 0 0 0 Updated 17 hours ago

Cedar is **open source**

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

Dafny spec +
DRT code





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

