Cedar: a new language for expressive, fast, safe, and analyzable authorization

Emina Torlak

Sr Principal Applied Scientist, AWS
Affiliate Professor at the University of Washington

Joint work with

Craig Disselkoen, Aaron Eline, Shaobo He, Kyle Headley, Mike Hicks, Kesha Hietala, John Kastner, Anwar Mamat, Darin McAdams, Matt McCutchen, Neha Rungta, and Andrew Wells

What is authorization?

- Document authoring
- Social media
- Trouble Ticketing
- Payroll
- On-line gaming
- Project management
- Microservices

Determining ***who*** can do ***what*** in a multi-user application

What is hard about authorization?

The theory is known …

but hard to implement

- Role-based access control (**RBAC**)
- Attribute-based access control (**ABAC**)
- Relation-based access control (**ReBAC**)

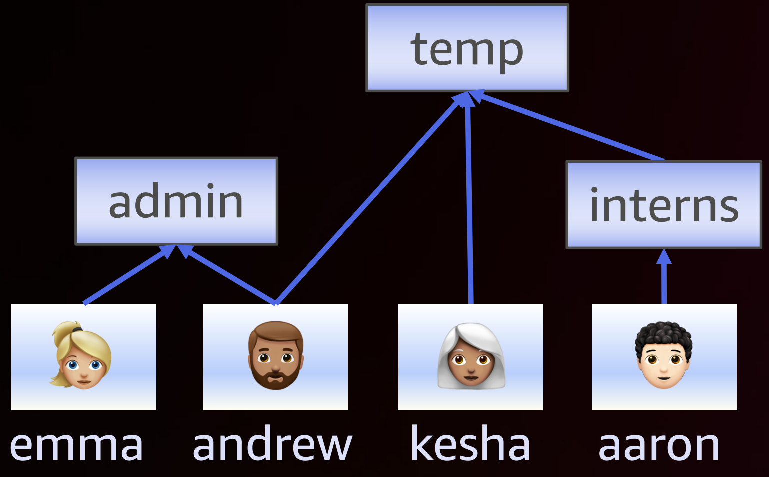
□ author
□ audit
□ maintain

What is hard about authorization? An example

TinyTodo✓

Allow **users and teams** to create, manage, and share **task lists**

- ❑ author
- ❑ audit
- ❑ maintain



List123

name: “Demo”
owner: User::"aaron"
editors: Team::”interns”
readers: Team::”temp”
tasks: […]

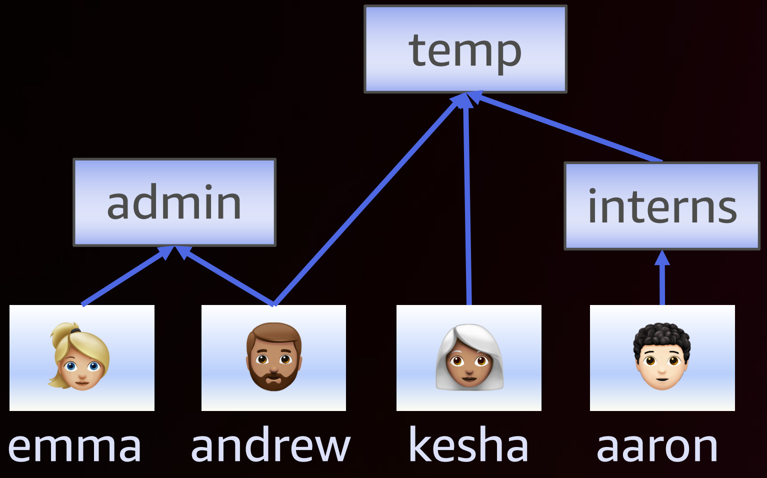
What is hard about authorization? An example

TinyTodo✓

Allow **users and teams** to create, manage, and share **task lists**

☒ author
☒ audit
☒ maintain

```
def get_list(request):
  if not request.user in db.query(admin):
   if db.query(request.listId).owner != request.user:
    if not request.user in db.query(request.listId).readers:
     if not request.user in db.query(request.listId).editors:
      return 'AccessDenied'
  list = db.query(request.listId)
  return { 'id': list.id, 'owner': list.owner, ... }
```



name: “Demo”
owner: User::"aaron"
editors: Team::”interns”
readers: Team::”temp”
tasks: […]

List123

Better authorization: policies as code

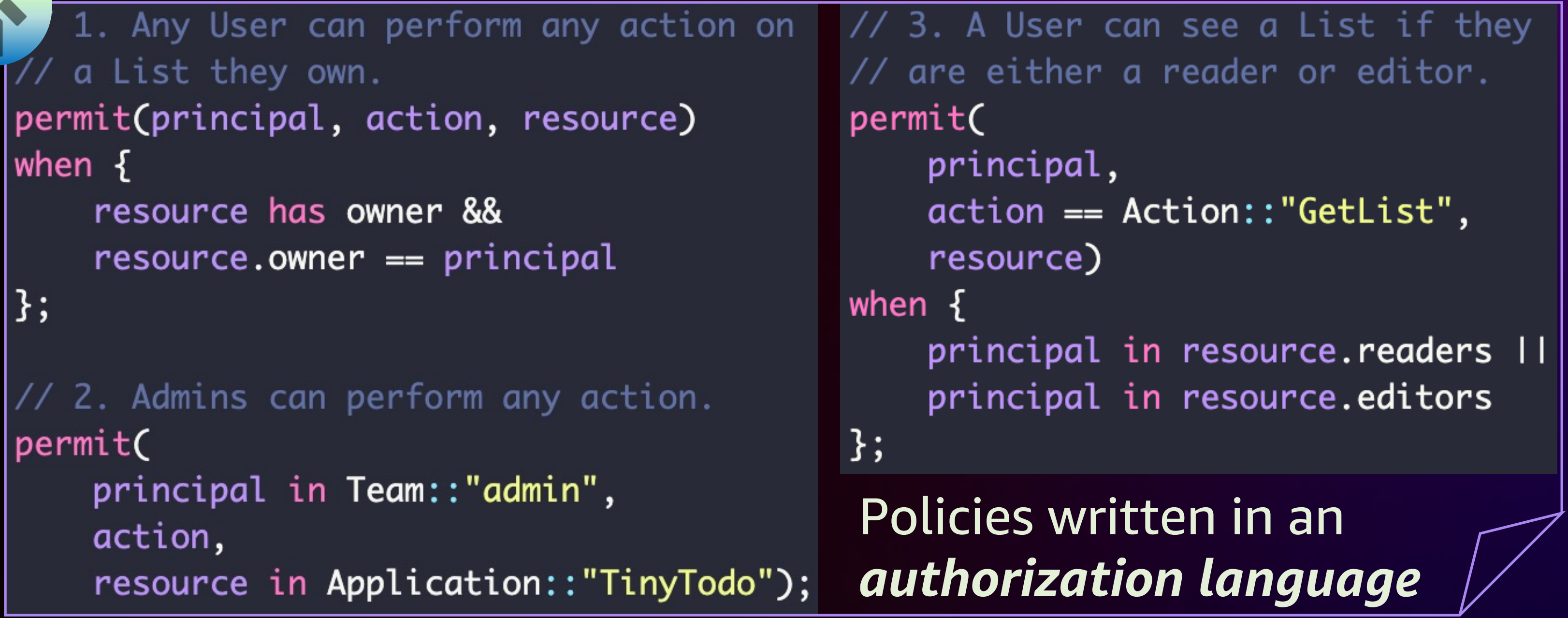
TinyTodo✓

Allow users and teams to create, manage, and share task lists

☑ author
☑ audit
☑ maintain

```
def get_list(request):
  if not is_authorized(request):
    return 'AccessDenied'
  list = db.query(request.listId)
  return { 'id': list.id, 'owner': list.owner, ... }
```

Delegate decision to an ***authorization engine***



```
   1. Any User can perform any action on
// a List they own.
permit(principal, action, resource)
when {
    resource has owner &&
    resource.owner == principal
};

// 2. Admins can perform any action.
permit(
    principal in Team::"admin",
    action,
    resource in Application::"TinyTodo");
```

```
// 3. A User can see a List if they
// are either a reader or editor.
permit(
    principal,
    action == Action::"GetList",
    resource)
when {
    principal in resource.readers ||
    principal in resource.editors
};
```

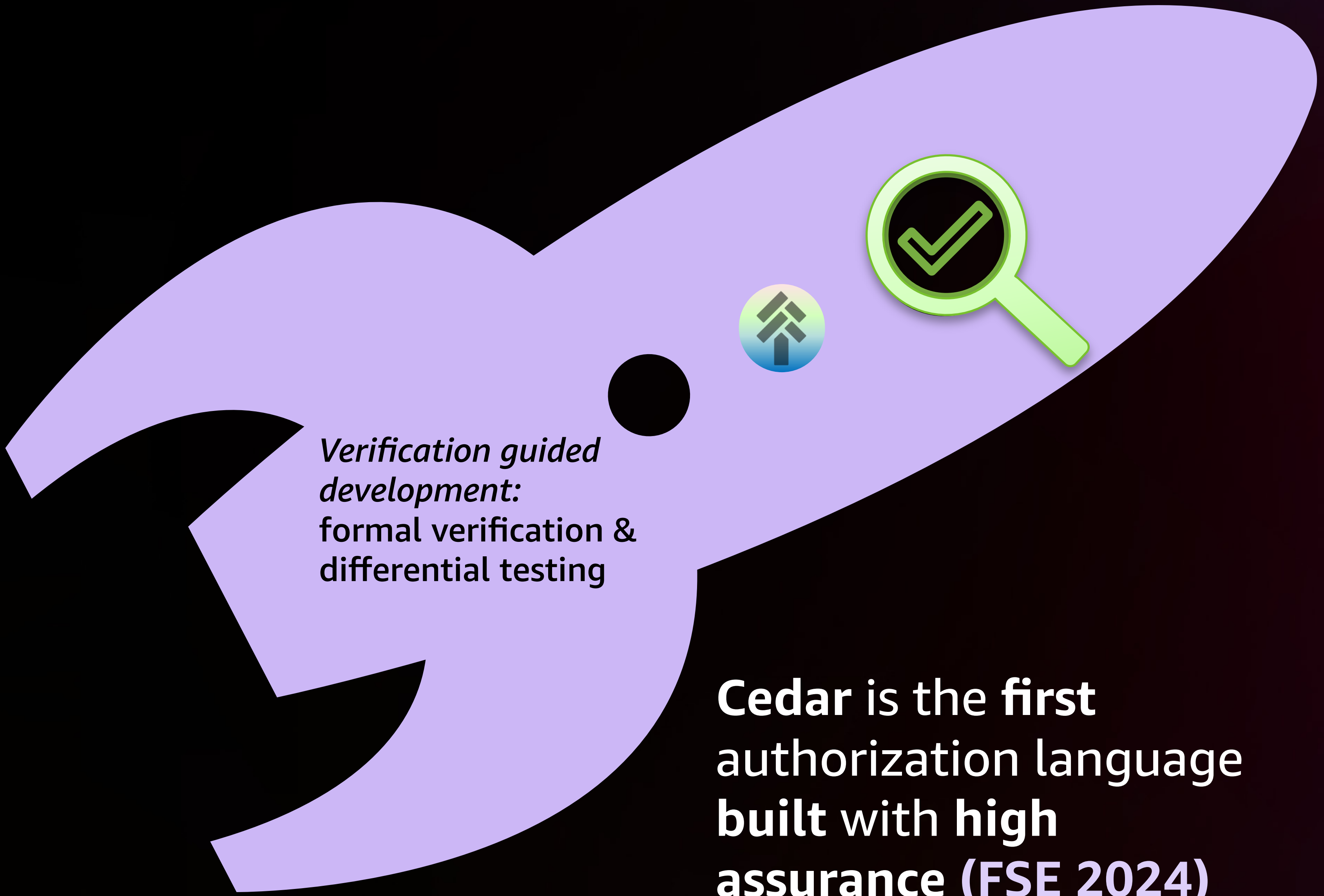
Policies written in an ***authorization language***

Cedar: a new authorization language

Powers Amazon Verified Permissions and AWS Verified Access

Open source at https://github.com/cedar-policy

What is unique about Cedar?



Verification guided development:
formal verification &
differential testing

Ergonomics

Expressiveness

Safety

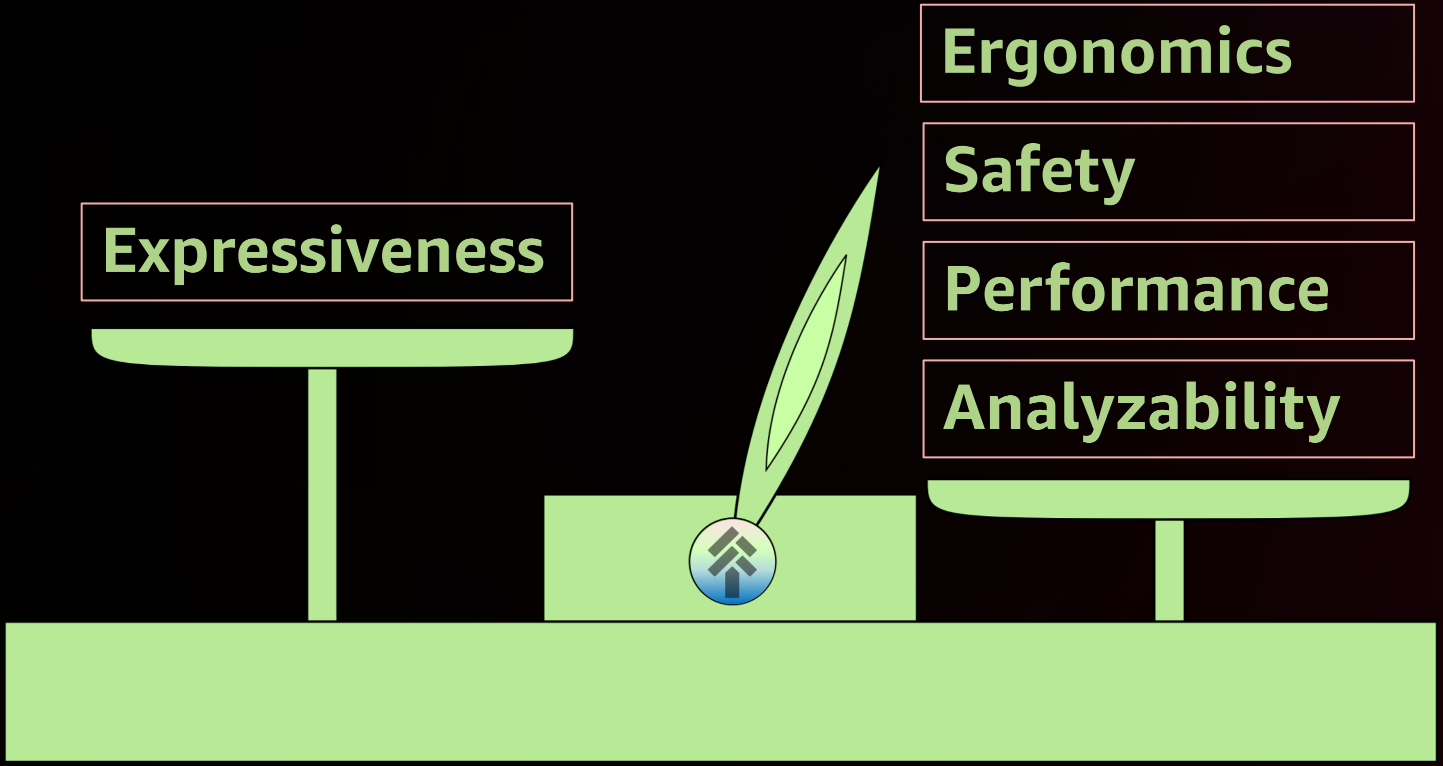
Performance

Analyzability

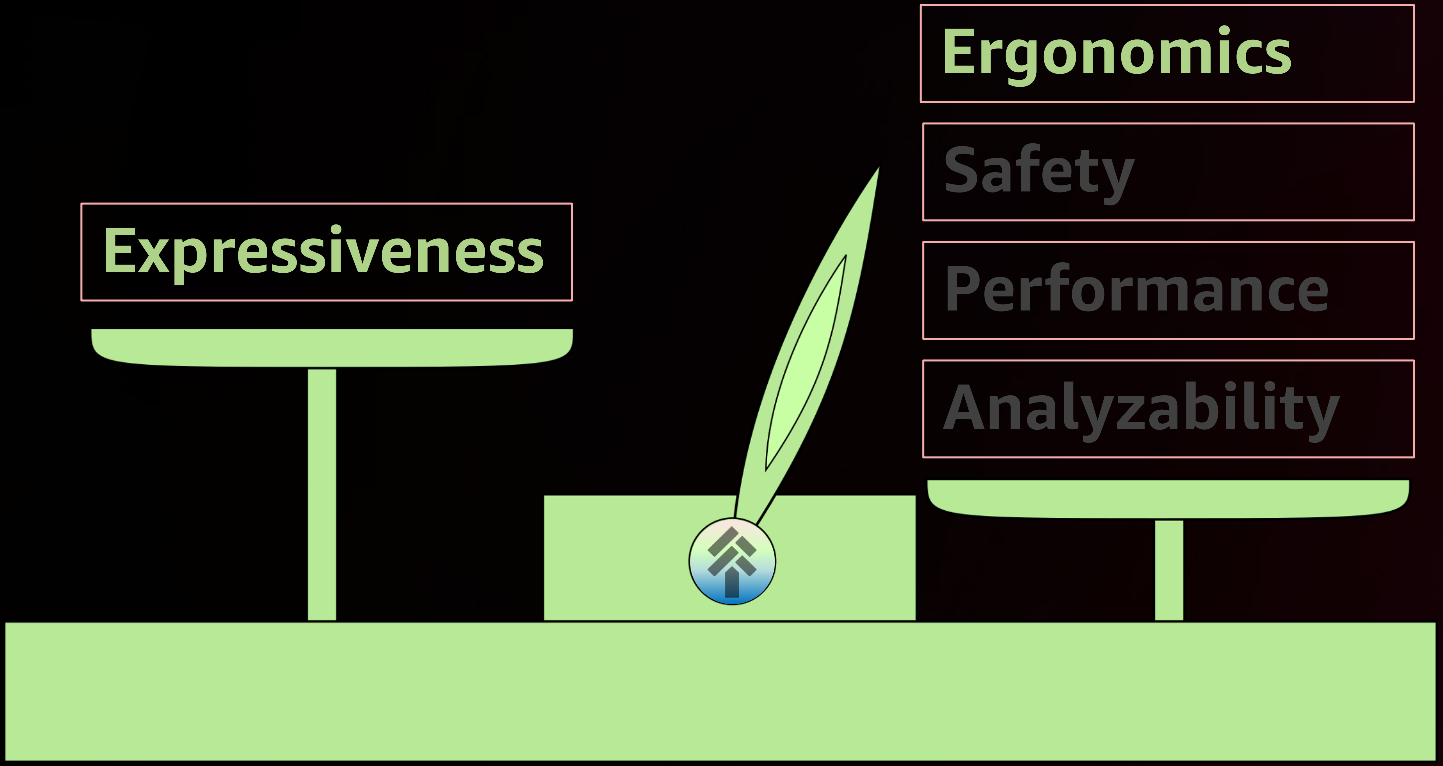
Cedar is the **first** authorization language **built** with **high assurance** (FSE 2024)

Cedar offers a **new way** to **balance** these criteria to achieve **analyzability** (OOPSLA 2024)

Cedar: design & development highlights



Cedar: design & development highlights



Expressiveness

Ergonomics

Safety

Performance

Analyzability

Syntax

Policy

```
permit (
  principal,
  action == Action::"GetList",
  resource)
when {
  principal in resource.readers ||
  principal in resource.editors
};
```

Effect: either `permit` or `forbid`

Scope: (optionally) constrains the `principal`, `action`, and `resource` using == and `in`

Condition(s): boolean expression prefixed by `when` or `unless` that further constrains access

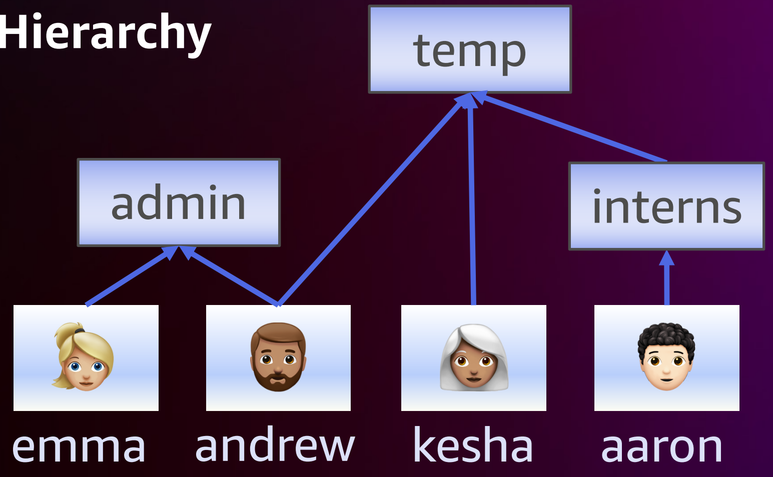
Syntax, data model

Policy

```
permit (
  principal,
  action == Action::"GetList",
  resource)
when {
  principal in resource.readers ||
  principal in resource.editors
};
```

Application ***entities*** with ***hierarchy*** and ***attributes***

Entities: Hierarchy



Entities: Attributes

List123

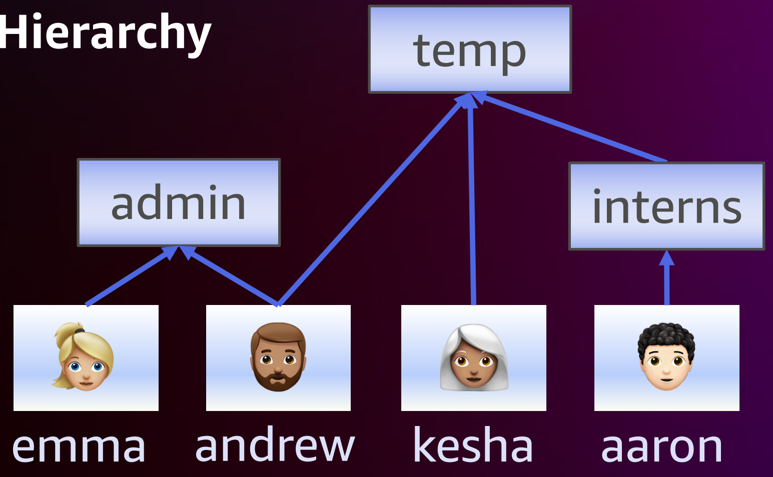
name: “Demo”
owner: User::"aaron"
editors: Team::”interns”
readers: Team::”temp”
tasks: […]

Syntax, data model

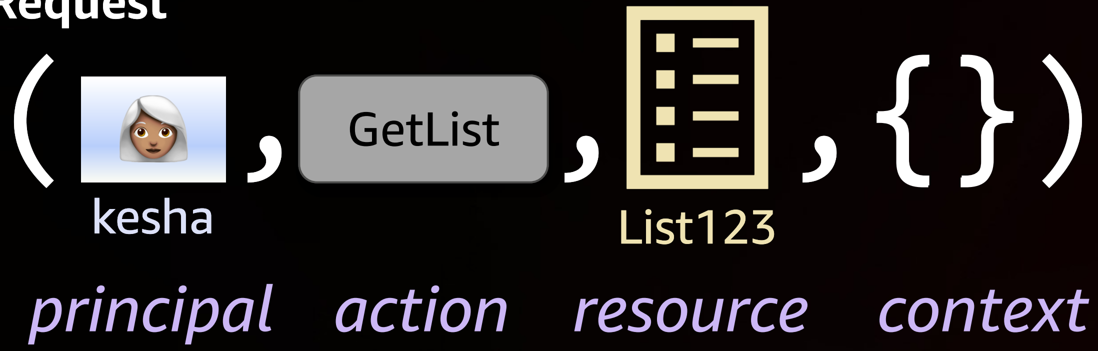
Policy

```
permit (
  principal,
  action == Action::"GetList",
  resource)
when {
  principal in resource.readers ||
  principal in resource.editors
};
```

Entities: Hierarchy



Request



Entities: Attributes

List123

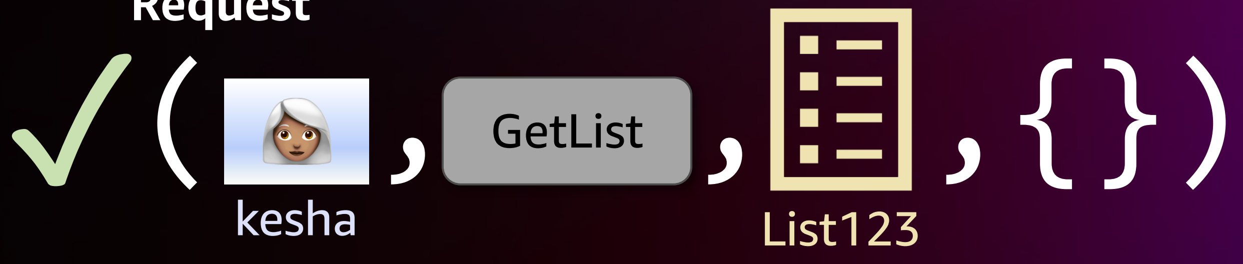
name: “Demo”
owner: User::"aaron"
editors: Team::”interns”
readers: Team::”temp”
tasks: […]

Syntax, data model, and semantics

Policy

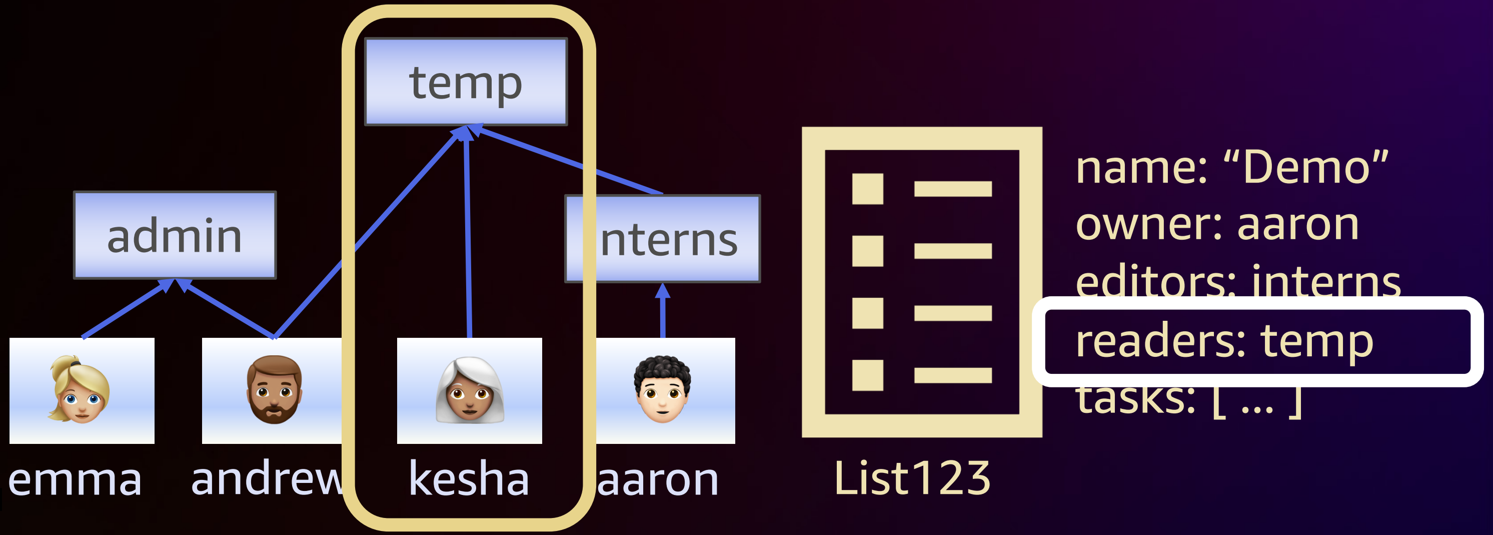
```
permit (
  principal,
  action == Action::"GetList",
  resource)
when {
  principal in resource.readers ||
  principal in resource.editors
};
```

Request



Request allowed when:

- it satisfies *at least one* **permit**
- and *no* **forbid** policies



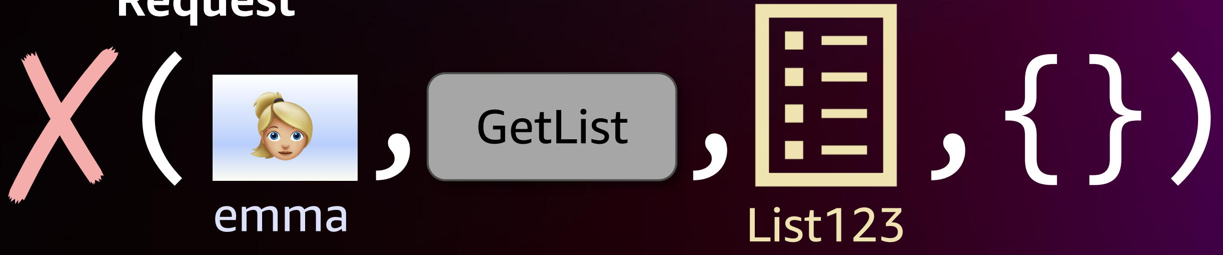
name: “Demo”
owner: aaron
editors: interns
readers: temp
tasks: […]

Syntax, data model, and semantics

Policy

```
permit (
  principal,
  action == Action::"GetList",
  resource)
when {
  principal in resource.readers ||
  principal in resource.editors
};
```

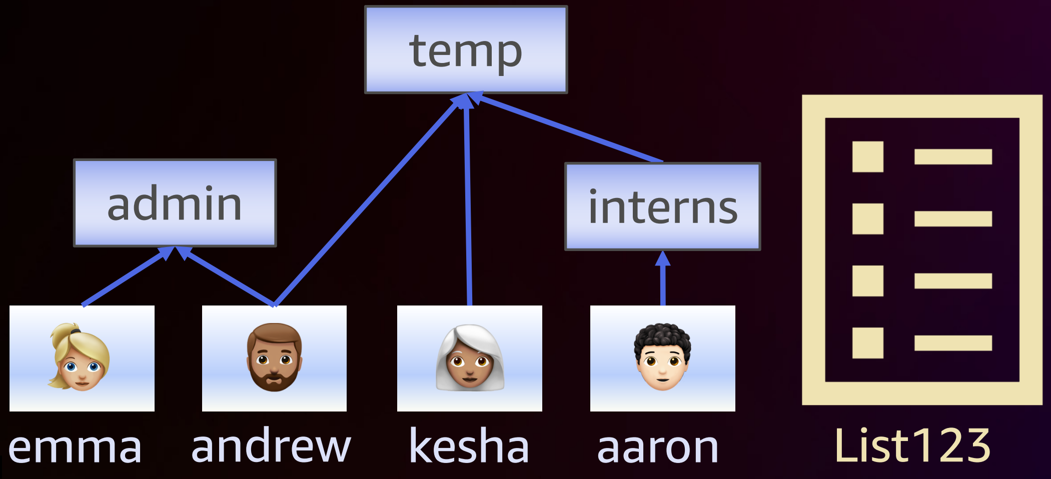
Request



(Default deny)

Request allowed when:

- it satisfies *at least one* **permit**
- and *no* **forbid** policies



name: “Demo”
owner: aaron
editors: interns
readers: temp
tasks: […]

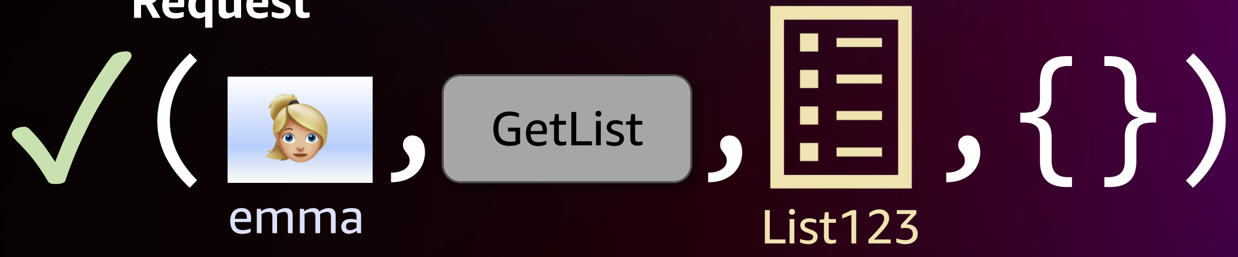
Syntax, data model, and semantics

Policy

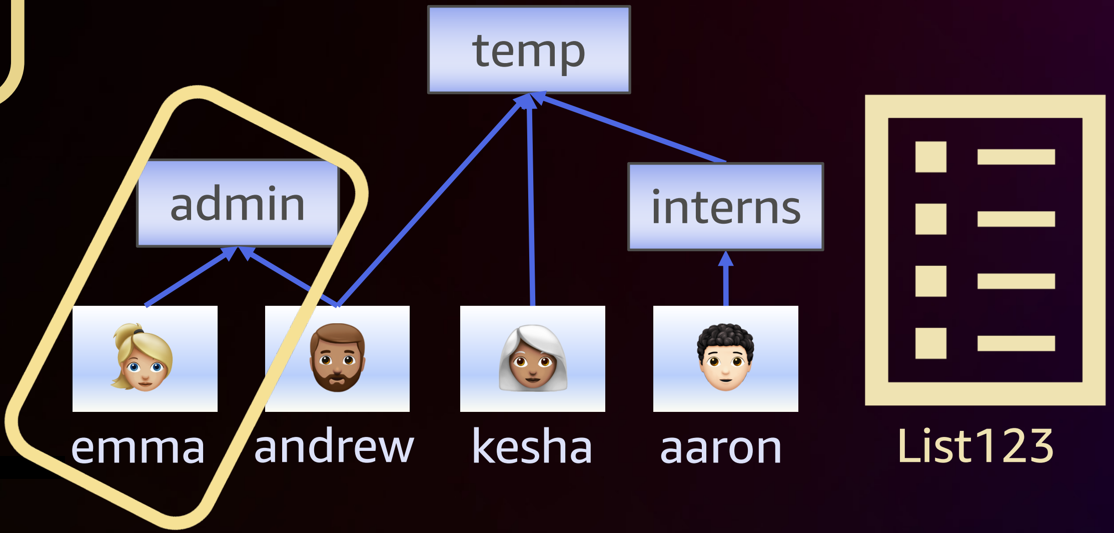
```
permit (
  principal,
  action == Action::"GetList",
  resource)
when {
  principal in resource.readers ||
  principal in resource.editors
};
```

```
permit (
  principal in Team::"admin",
  action,
  resource);
```

Request



✓ (emma , GetList , List123 , {})



temp

admin

interns

emma

andrew

kesha

aaron

List123

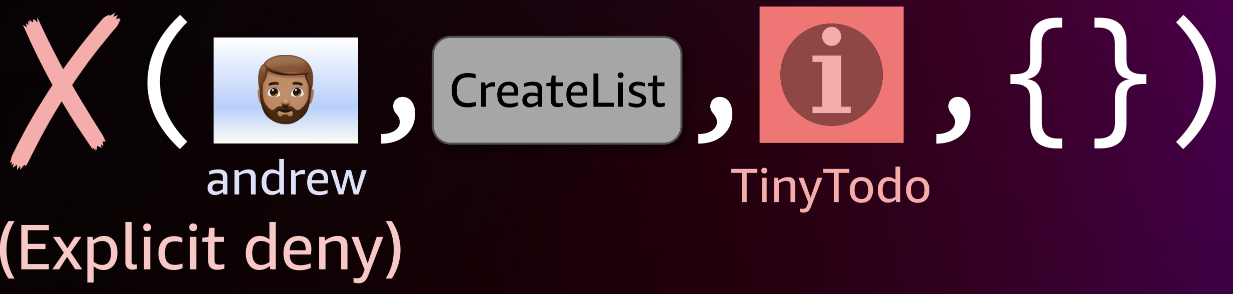
name: “Demo”
owner: aaron
editors: interns
readers: temp
tasks: […]

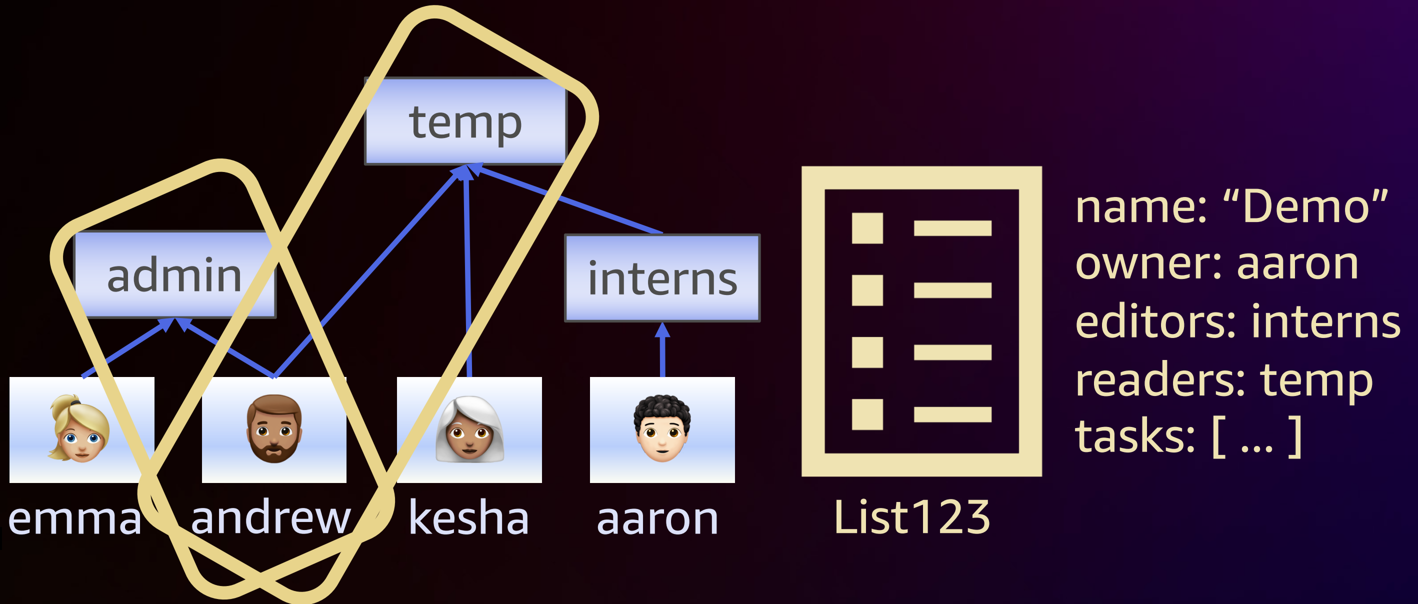
Syntax, data model, and semantics

Policy

```
permit (
  principal,
  action == Action::"GetList",
  resource)
when {
  principal in resource.readers ||
  principal in resource.editors
};
permit (
  principal in Team::"admin",
  action,
  resource);
forbid (
  principal in Team::"temp",
  action == Action::"CreateList",
  resource == Application::"TinyTodo";
```

Request





Syntax, data model, and semantics for …

- role-based access control (RBAC)
- attribute-based access control (ABAC)
- relation-based access control (ReBAC)

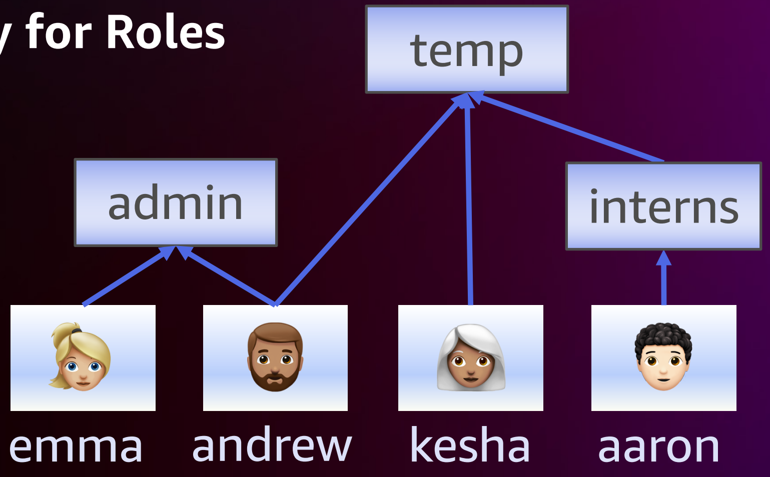
Syntax, data model, and semantics: RBAC

```
permit (
  principal in Team::"admin",
  action,
  resource);

forbid (
  principal in Team::"temp",
  action == Action::"CreateList",
  resource == Application::"TinyTodo");
```

in: transitive membership

Hierarchy for Roles



Key idea for O(1) **in** checks: it operates on the *transitive closure* of the entity hierarchy, given as a map from entities to their ancestors (sets of entities).

name: “Demo”
owner: User::"aaron"
editors: Team::”interns”
readers: Team::”temp”
tasks: […]

List123

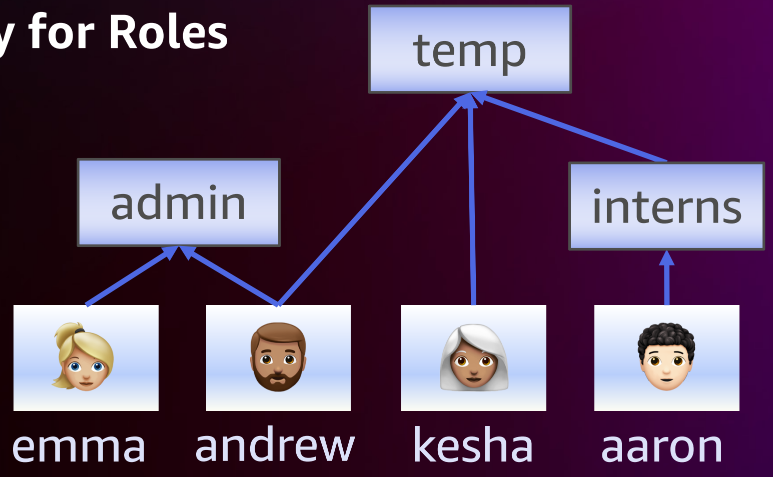
Syntax, data model, and semantics: ABAC

```
permit (
  principal,
  action,
  resource)
when {
  resource has owner &&
  resource.owner == principal
};
```

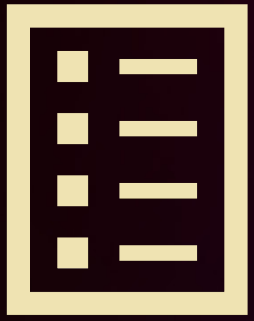
Conditions are **pure, loop-free expressions**: **==** , **in**, set membership, conditionals, **!**, **&&**, **||**, wildcard matching, …

Evaluation time **O(n)** typical, **O(n³)** worst case

Hierarchy for Roles



Attributes for Conditions

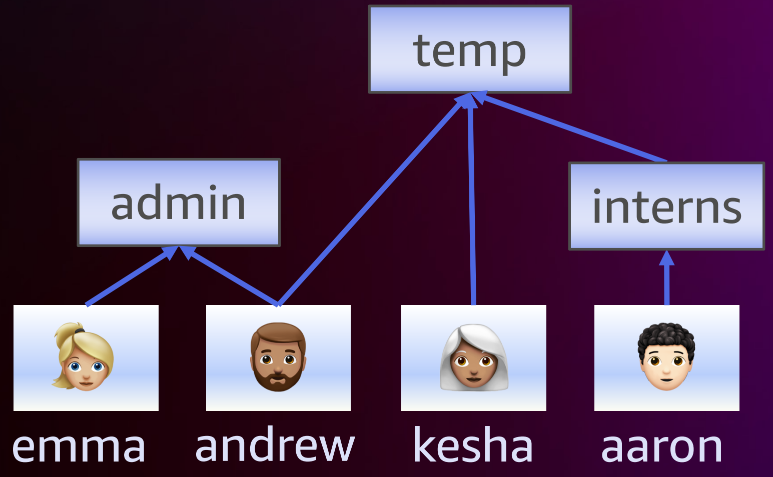


name: “Demo”
owner: User::"aaron"
editors: Team::”interns”
readers: Team::”temp”
tasks: […]

Syntax, data model, and semantics: ReBAC

```
permit (
  principal,
  action == Action::"GetList",
  resource)
when {
  principal in resource.readers ||
  principal in resource.editors
};
```

Hierarchy
+ Attributes
= Relations

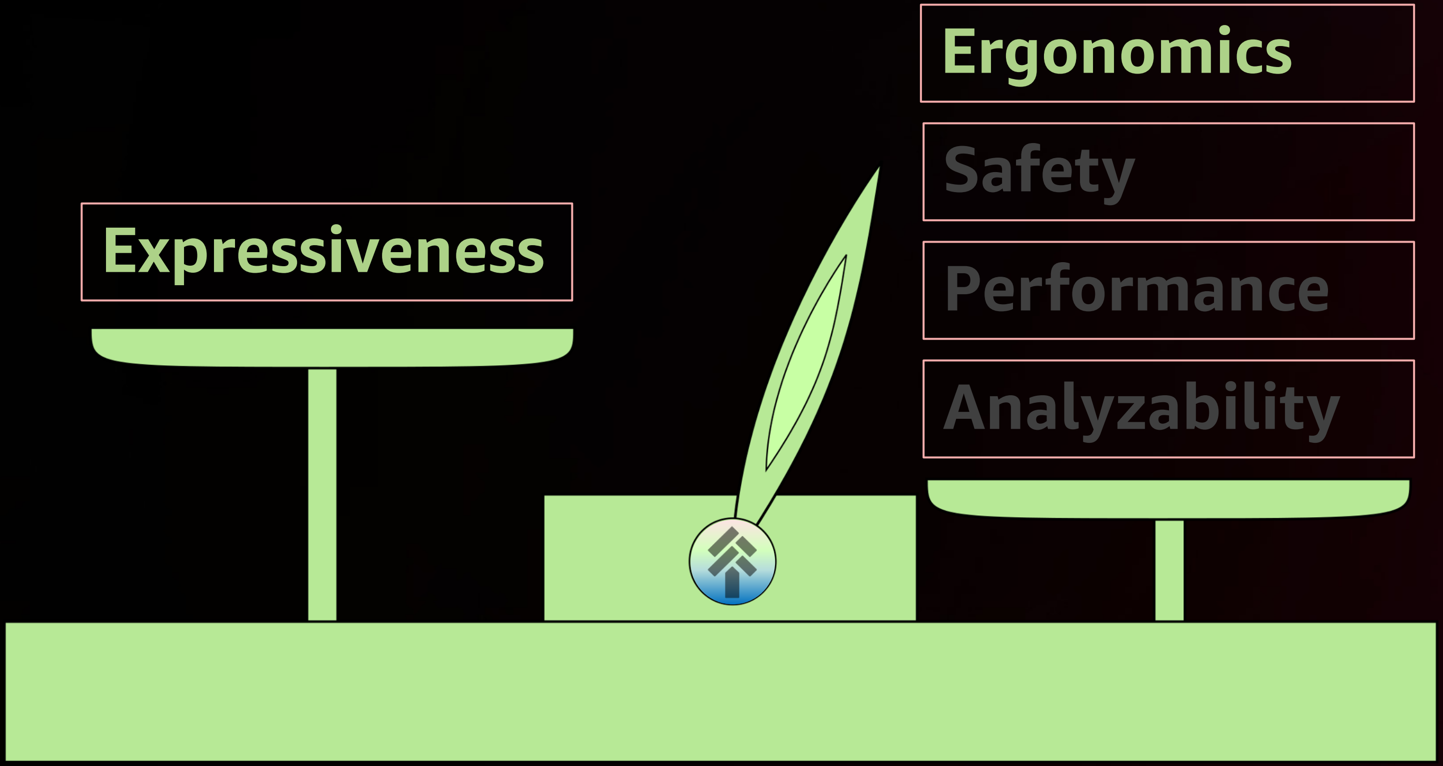


"principal *is related to* resource via the readers relation
or
principal *is related to* resource via the editors relation"

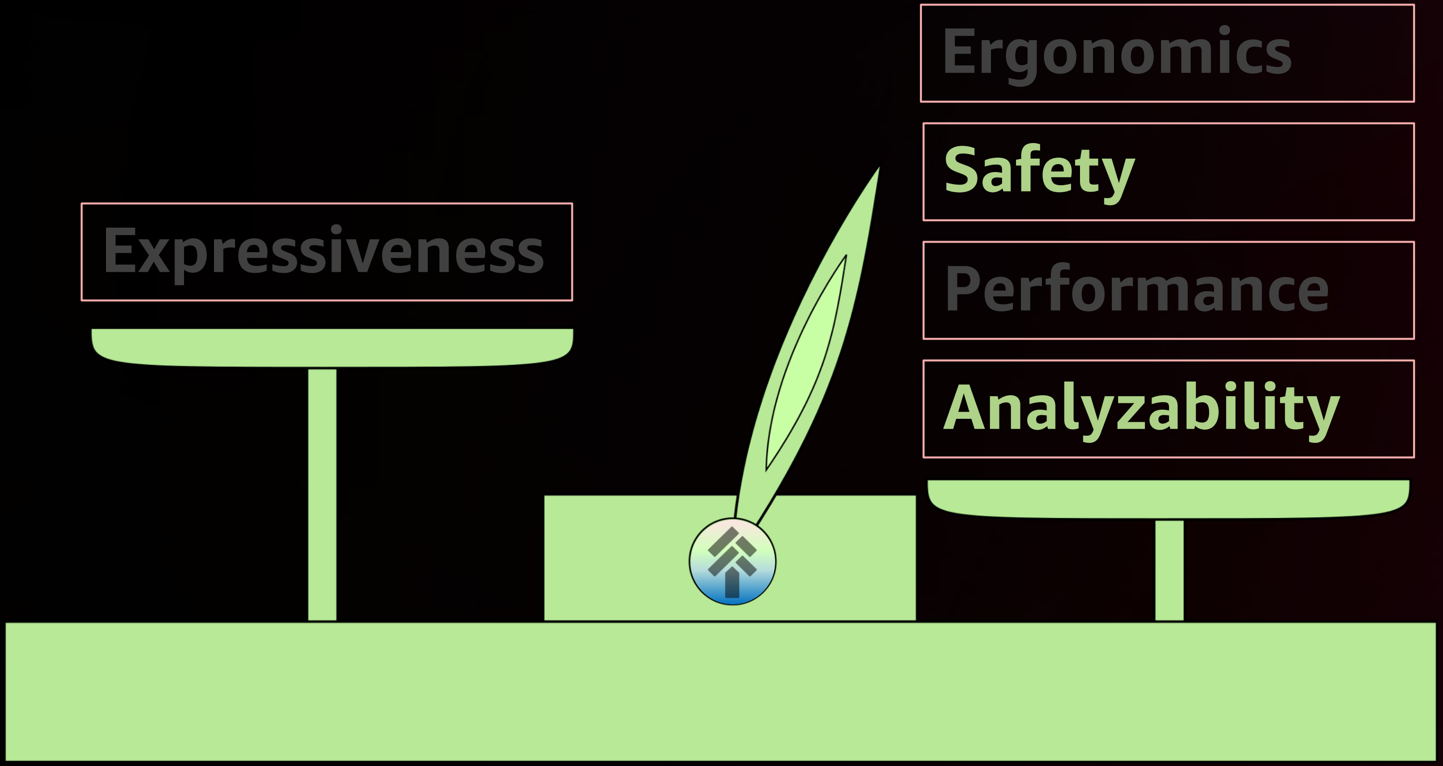
name: "Demo"
owner: User::"aaron"
editors: Team::"interns"
readers: Team::"temp"
tasks: [...]

List123

Cedar: design & development highlights



Cedar: design & development highlights



Policy validation for safety

```
permit(
 principal,
 action == Action::"GetList",
 resource)
when {
 principal in resource.readers ||
 principal in resource.editors
};
```

Valid?

```
entity Application;
entity Team, User in [Team];
entity List {
  readers: Team,
  editors: Team,
  owner: User,
  tasks: Set<Task>,
  name: String
};
action GetList appliesTo {
  principal: [User],
  resource: [List]
};
```

Schema

Can detect: improper entity relationship, misspelling of action, unrecognized attributes, illegal operations, …

Key features: path-sensitive request-type handling, flow-sensitive capability tracking, singleton types

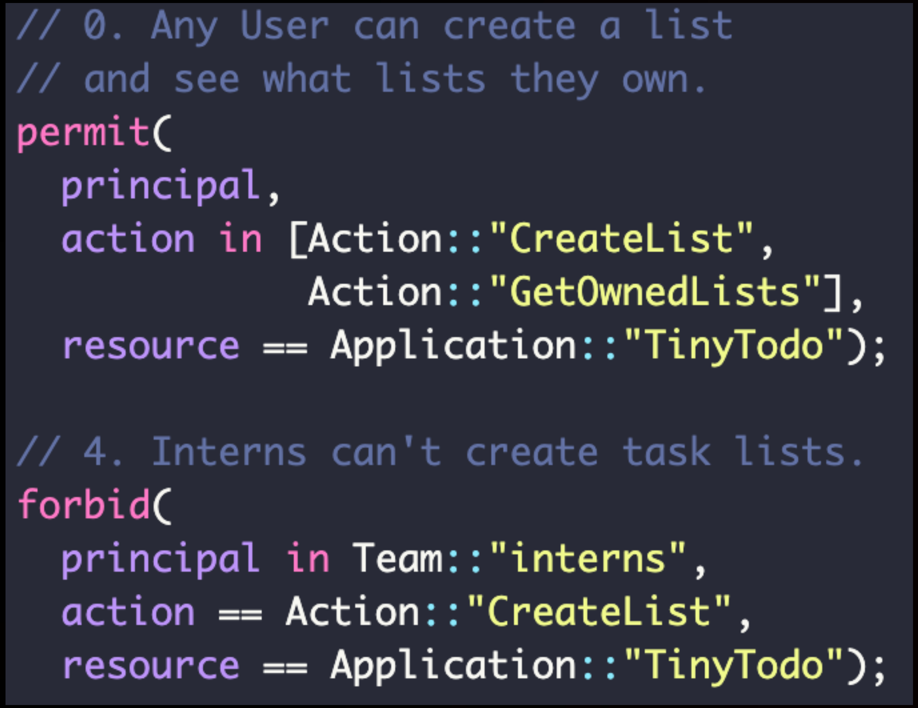
Theorem (soundness): If validation succeeds, policy evaluation will exhibit no run-time type errors.

Policy analysis for semantic reasoning

Answers ***universal*** questions about the behavior of policies on ***all*** possible inputs—all requests and entities

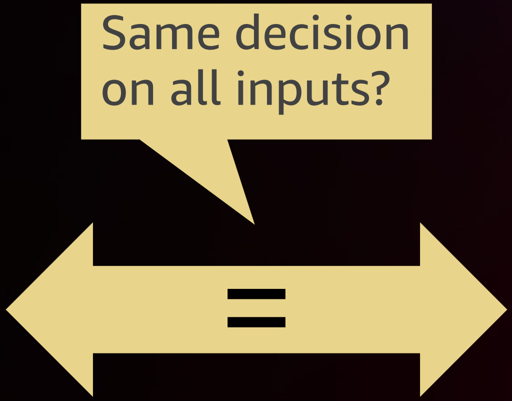
Example (equivalence): do two (sets of) policies produce the same decision on all inputs?

Example policy analysis: equivalence

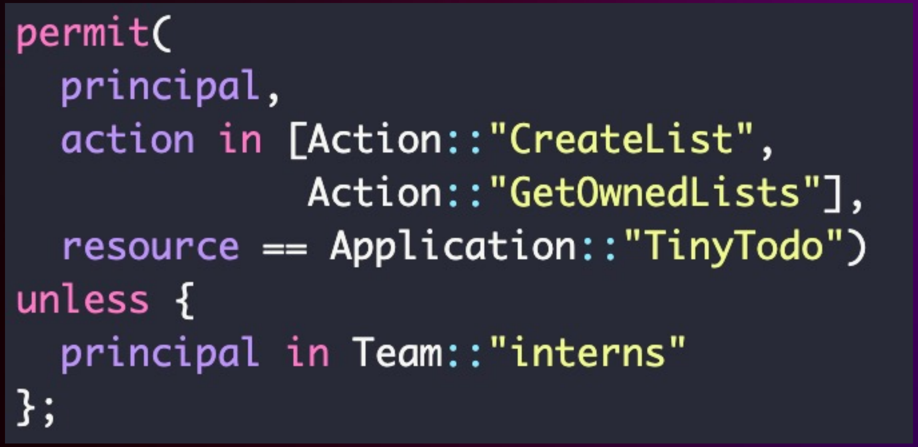


```
// 0. Any User can create a list
// and see what lists they own.
permit(
  principal,
  action in [Action::"CreateList",
             Action::"GetOwnedLists"],
  resource == Application::"TinyTodo");

// 4. Interns can't create task lists.
forbid(
  principal in Team::"interns",
  action == Action::"CreateList",
  resource == Application::"TinyTodo");
```

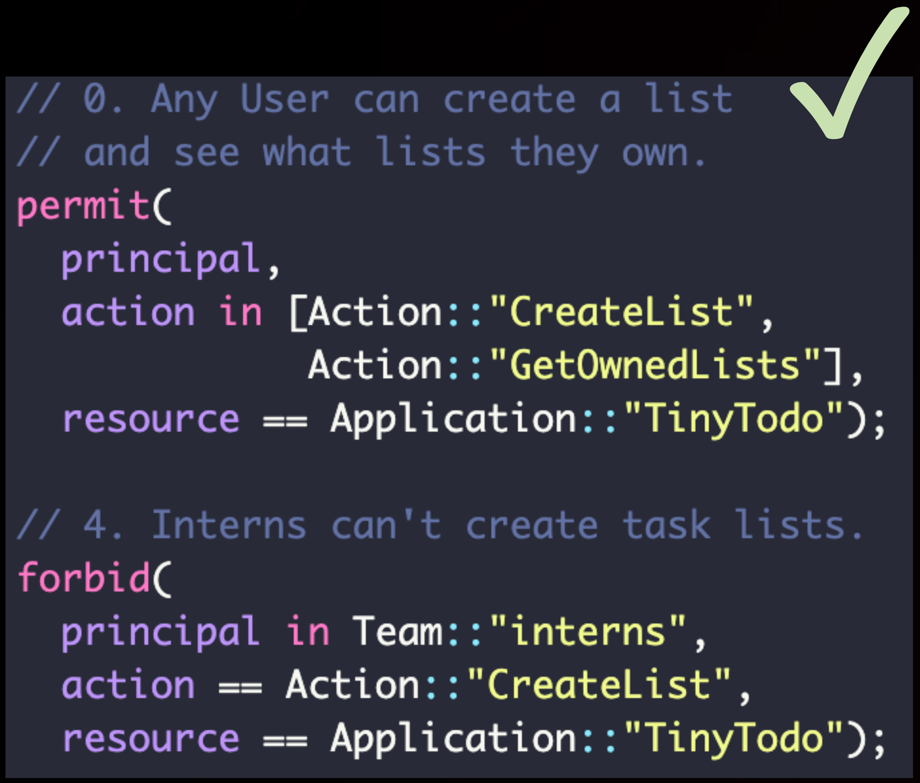


Same decision on all inputs?



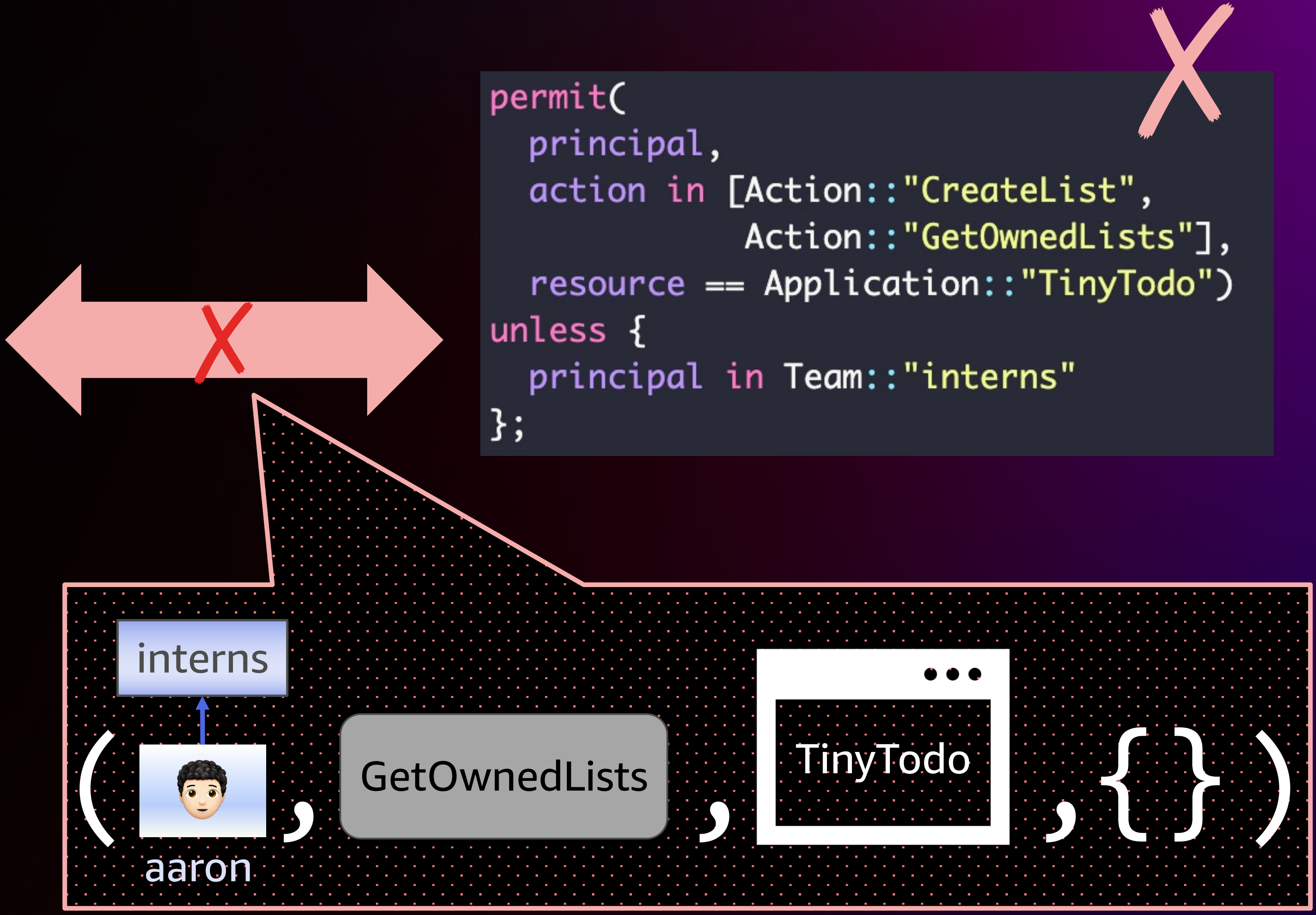
```
permit(
  principal,
  action in [Action::"CreateList",
             Action::"GetOwnedLists"],
  resource == Application::"TinyTodo")
unless {
  principal in Team::"interns"
};
```

Example policy analysis: equivalence



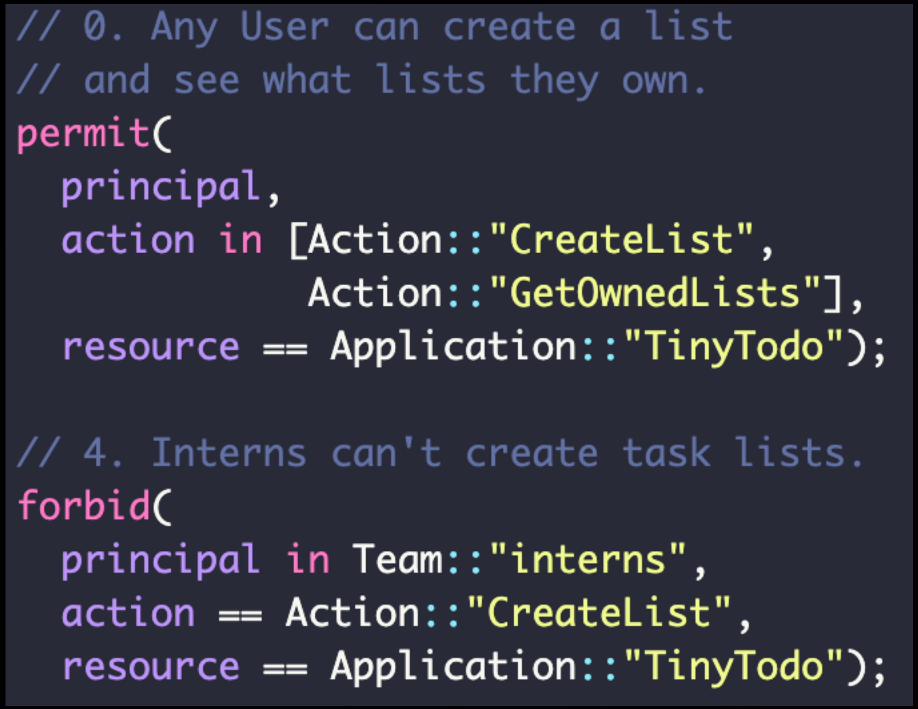
```
// 0. Any User can create a list
// and see what lists they own.
permit(
  principal,
  action in [Action::"CreateList",
             Action::"GetOwnedLists"],
  resource == Application::"TinyTodo");

// 4. Interns can't create task lists.
forbid(
  principal in Team::"interns",
  action == Action::"CreateList",
  resource == Application::"TinyTodo");
```



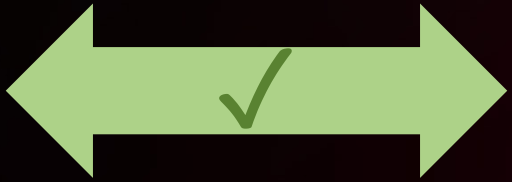
```
permit(
  principal,
  action in [Action::"CreateList",
             Action::"GetOwnedLists"],
  resource == Application::"TinyTodo")
unless {
  principal in Team::"interns"
};
```

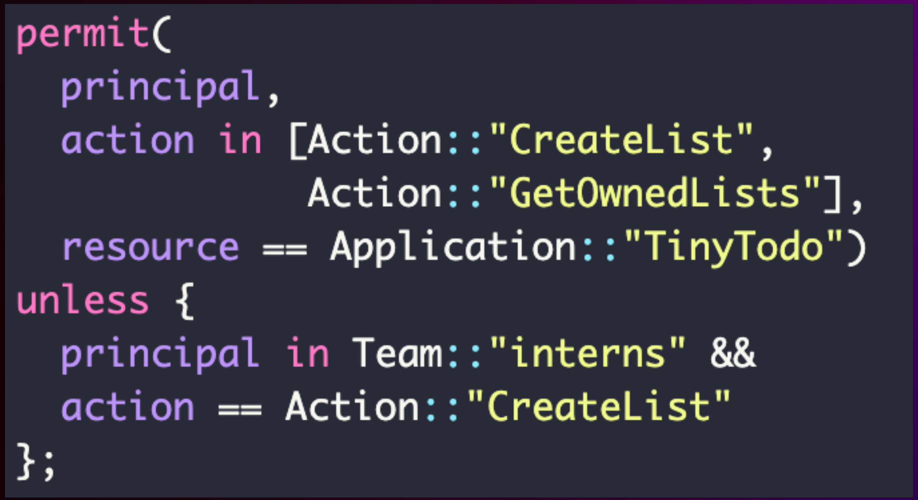
Example policy analysis: equivalence



```
// 0. Any User can create a list
// and see what lists they own.
permit(
  principal,
  action in [Action::"CreateList",
             Action::"GetOwnedLists"],
  resource == Application::"TinyTodo");

// 4. Interns can't create task lists.
forbid(
  principal in Team::"interns",
  action == Action::"CreateList",
  resource == Application::"TinyTodo");
```





```
permit(
  principal,
  action in [Action::"CreateList",
             Action::"GetOwnedLists"],
  resource == Application::"TinyTodo")
unless {
  principal in Team::"interns" &&
  action == Action::"CreateList"
};
```

Works by **symbolically compiling** policies to logical formulas, and using an **SMT solver** to check that the negation of the desired property is unsatisfiable.

Policy analysis by symbolic compilation to SMT

Schema

Expression

$$M,\Gamma \vdash e \downarrow t$$

Symbolic environment

Symbolic term

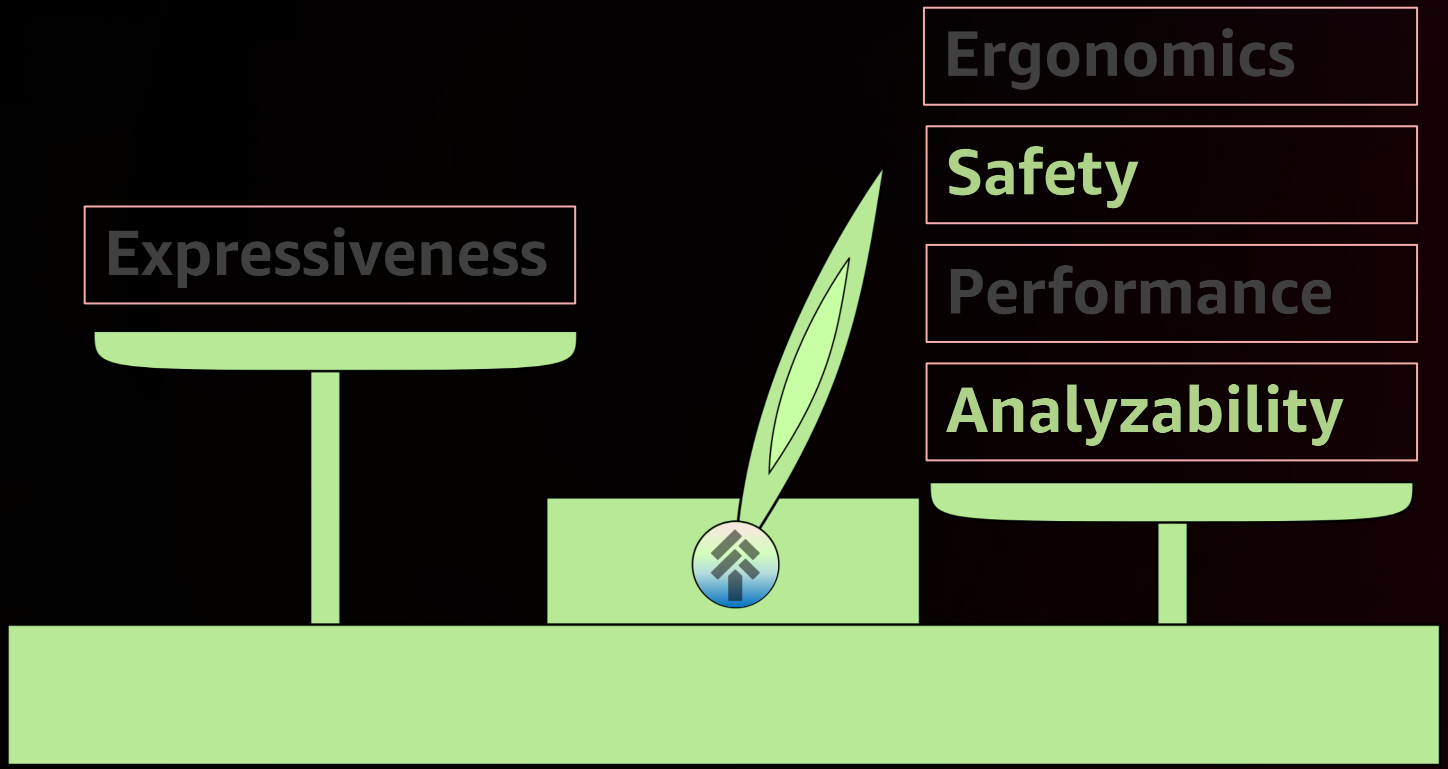
Key challenge: how to encode the fact that hierarchies are DAGs while remaining decidable (i.e., without transitive closure or quantifiers)?

Symbolic compilation: type-directed reduction to a ***decidable*** fragment of SMT (uninterpreted functions, bitvectors, strings, ADTs, and finite sets)

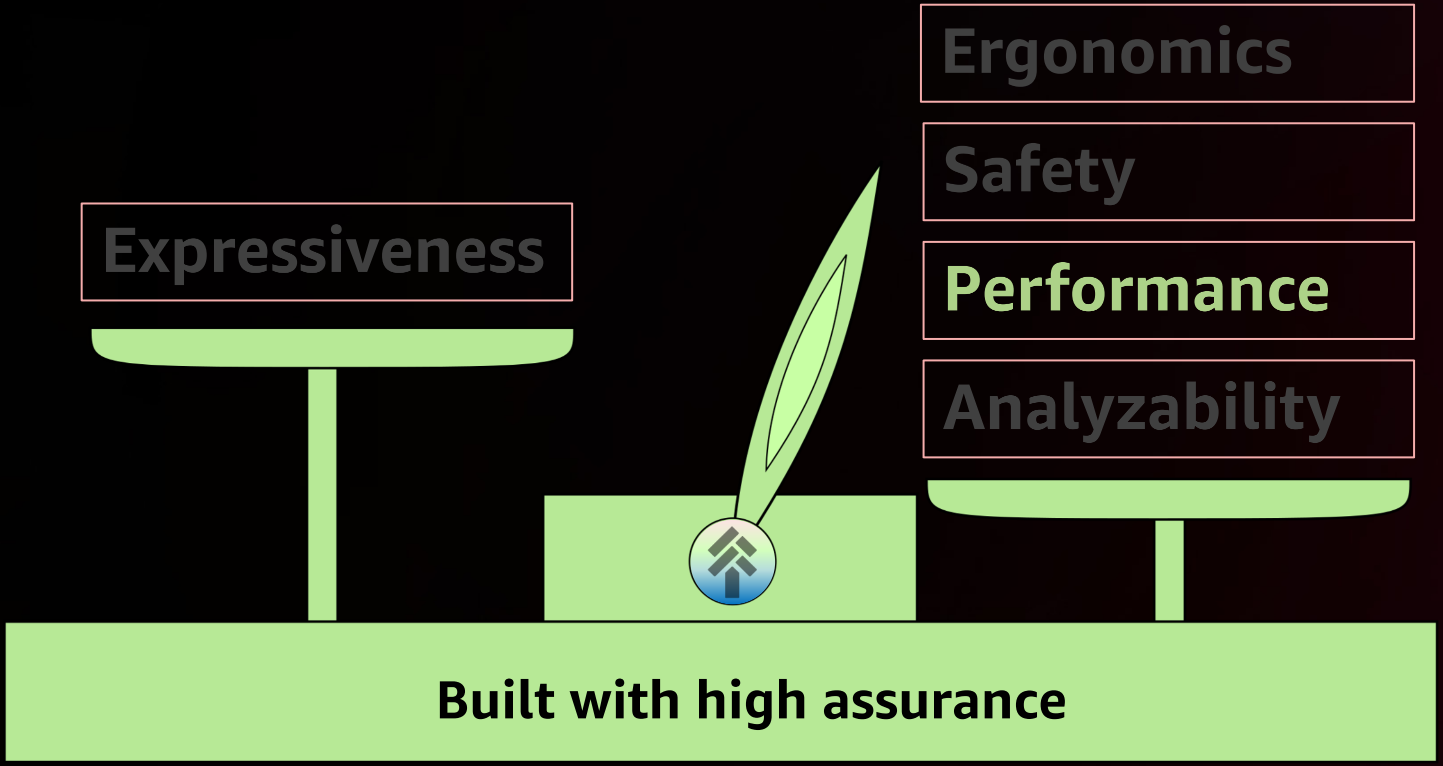
Solution: observe that an expression ***e*** can access only a finite set of entities. Compute an overapproximation of that set, and use it to ground acyclicity and transitivity constraints on hierarchies.

Theorem (soundness and completeness): policy analysis based on symbolic compilation produces no false negatives and no false positives.

Cedar: design & development highlights

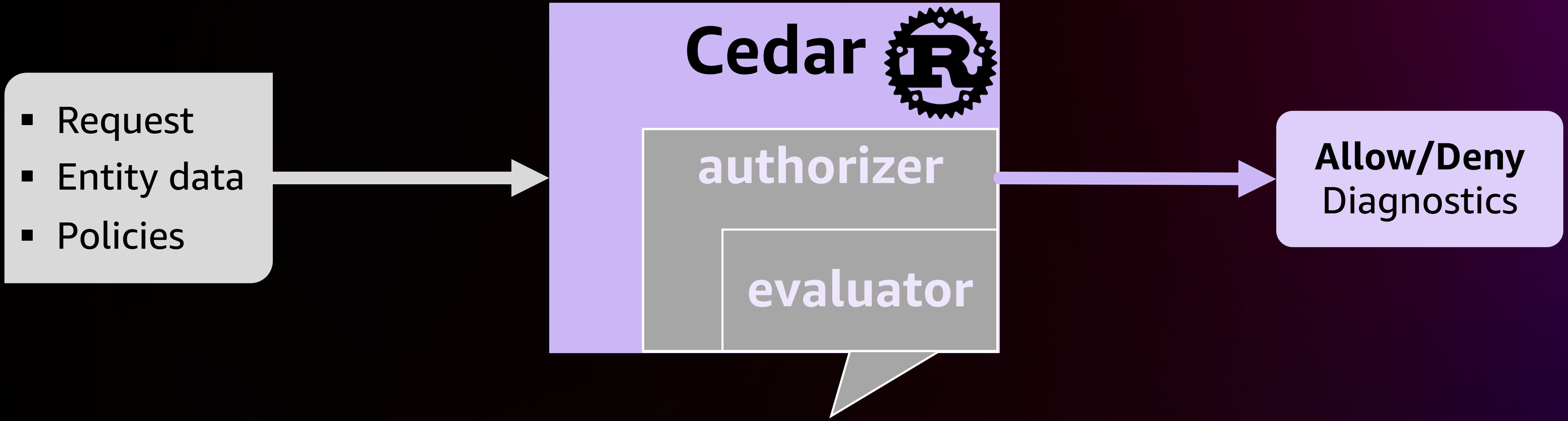


Cedar: design & development highlights



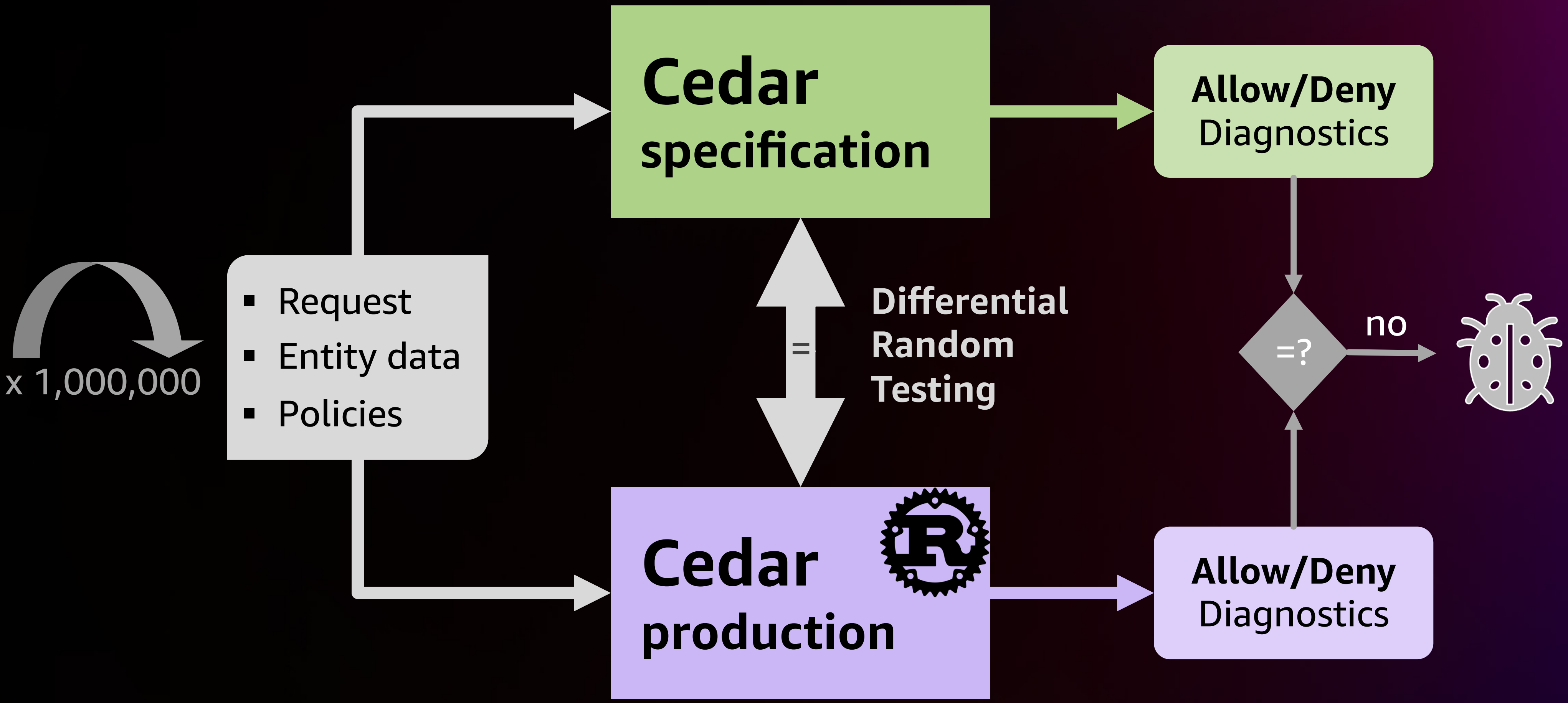
A fast runtime with Rust

42.8×-80.8× faster than OPA Rego
28.7×-35.2× faster than OpenFGA



< 10 µs for a typical input

A fast & safe runtime with Rust, DRT



x 1,000,000

- Request
- Entity data
- Policies

Cedar specification

Cedar production

=

Differential Random Testing

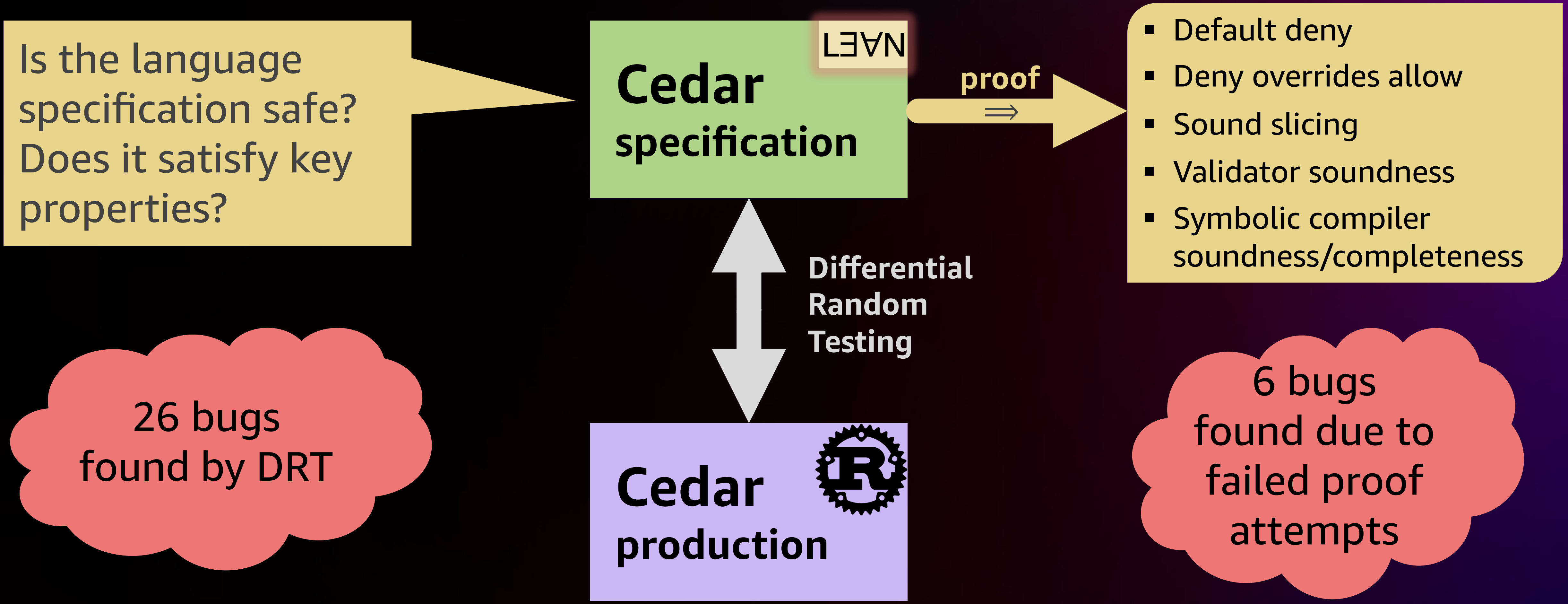
Allow/Deny Diagnostics

Allow/Deny Diagnostics

=?

no

A fast & safe runtime with Rust, DRT, and Lean



Is the language specification safe? Does it satisfy key properties?

Cedar specification

proof ⟹

- Default deny
- Deny overrides allow
- Sound slicing
- Validator soundness
- Symbolic compiler soundness/completeness

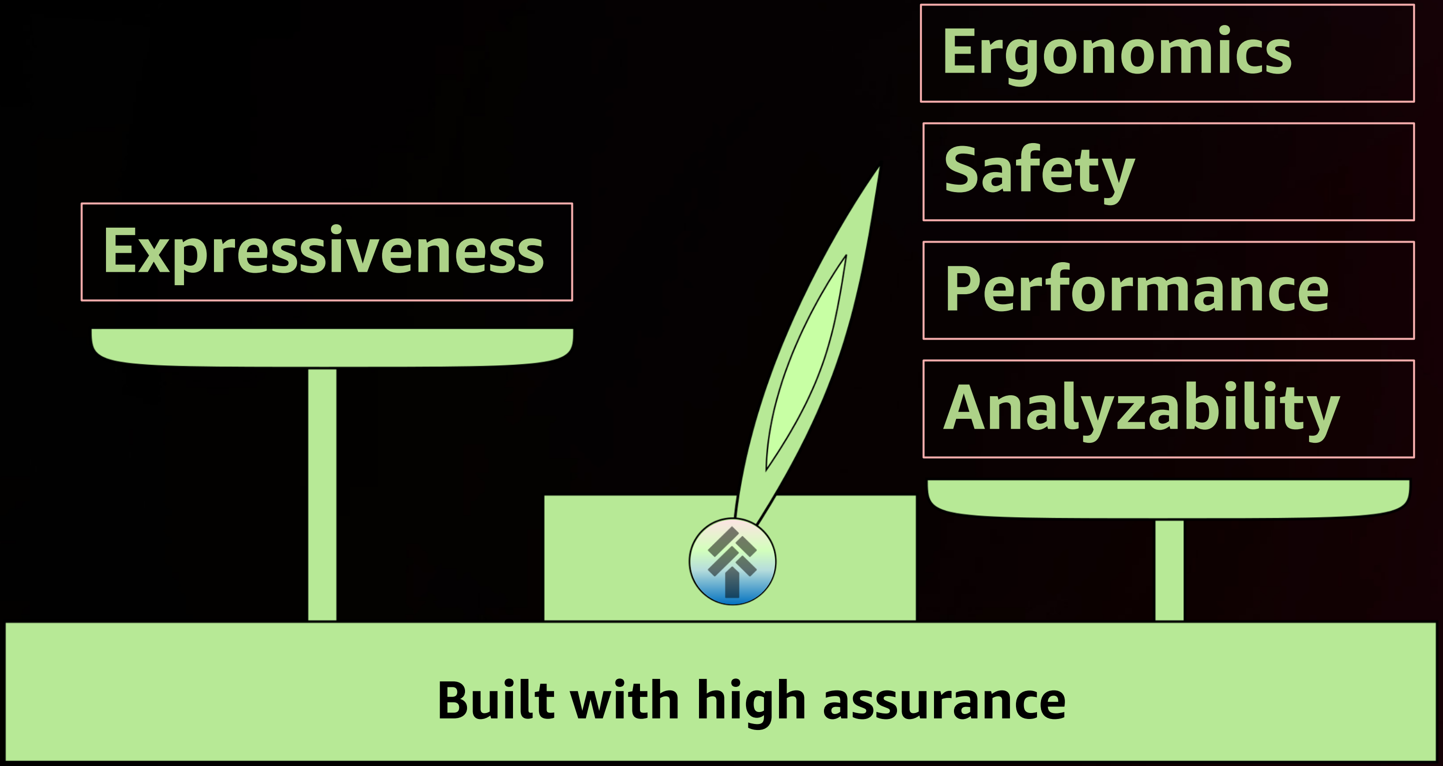
Differential Random Testing

Cedar production

26 bugs found by DRT

6 bugs found due to failed proof attempts

Cedar: expressive, fast, safe, analyzable authz


Expressiveness

Ergonomics

Safety

Performance

Analyzability

Built with high assurance

https://github.com/cedar-policy