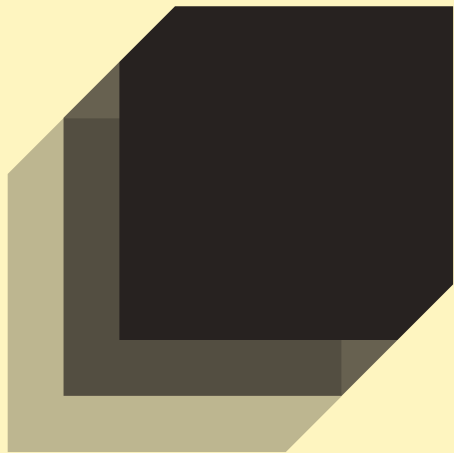


# *Can't* Touch This!

Moving beyond roles and permissions  
to a *fine-grained* access control



# Paradigm *shift*

Moving beyond roles and permissions  
to a *fine-grained* access control

# Safe harbor

This presentation contains "forward-looking statements" within the meaning of the "safe harbor" provisions of the Private Securities Litigation Reform Act of 1995, including but not limited to, statements regarding our financial outlook, business strategy and plans, market trends and market size, opportunities and positioning. These forward-looking statements are based on current expectations, estimates, forecasts and projections. Words such as "expect," "anticipate," "should," "believe," "hope," "target," "project," "goals," "estimate," "potential," "predict," "may," "will," "might," "could," "intend," "shall" and variations of these terms and similar expressions are intended to identify these forward-looking statements, although not all forward-looking statements contain these identifying words. Forward-looking statements are subject to a number of risks and uncertainties, many of which involve factors or circumstances that are beyond our control. For example, global economic conditions have in the past and could in the future reduce demand for our products; we and our third-party service providers have in the past and could in the future experience cybersecurity incidents; we may be unable to manage or sustain the level of growth that our business has experienced in prior periods; our financial resources may not be sufficient to maintain or improve our competitive position; we may be unable to attract new customers, or retain or sell additional products to existing customers;

customer growth has slowed in recent periods and could continue to decelerate in the future; we could experience interruptions or performance problems associated with our technology, including a service outage; we and our third-party service providers have failed, or were perceived as having failed, to fully comply with various privacy and security provisions to which we are subject, and similar incidents could occur in the future; we may not achieve expected synergies and efficiencies of operations from recent acquisitions or business combinations, and we may not be able to successfully integrate the companies we acquire; and we may not be able to pay off our convertible senior notes when due. Further information on potential factors that could affect our financial results is included in our most recent Quarterly Report on Form 10-Q and our other filings with the Securities and Exchange Commission. The forward-looking statements included in this presentation represent our views only as of the date of this presentation and we assume no obligation and do not intend to update these forward-looking statements.

Any products, features or functionality referenced in this presentation that are not currently generally available may not be delivered on time or at all. Product roadmaps do not represent a commitment, obligation or promise to deliver any product, feature or functionality, and you should not rely on them to make your purchase decisions.



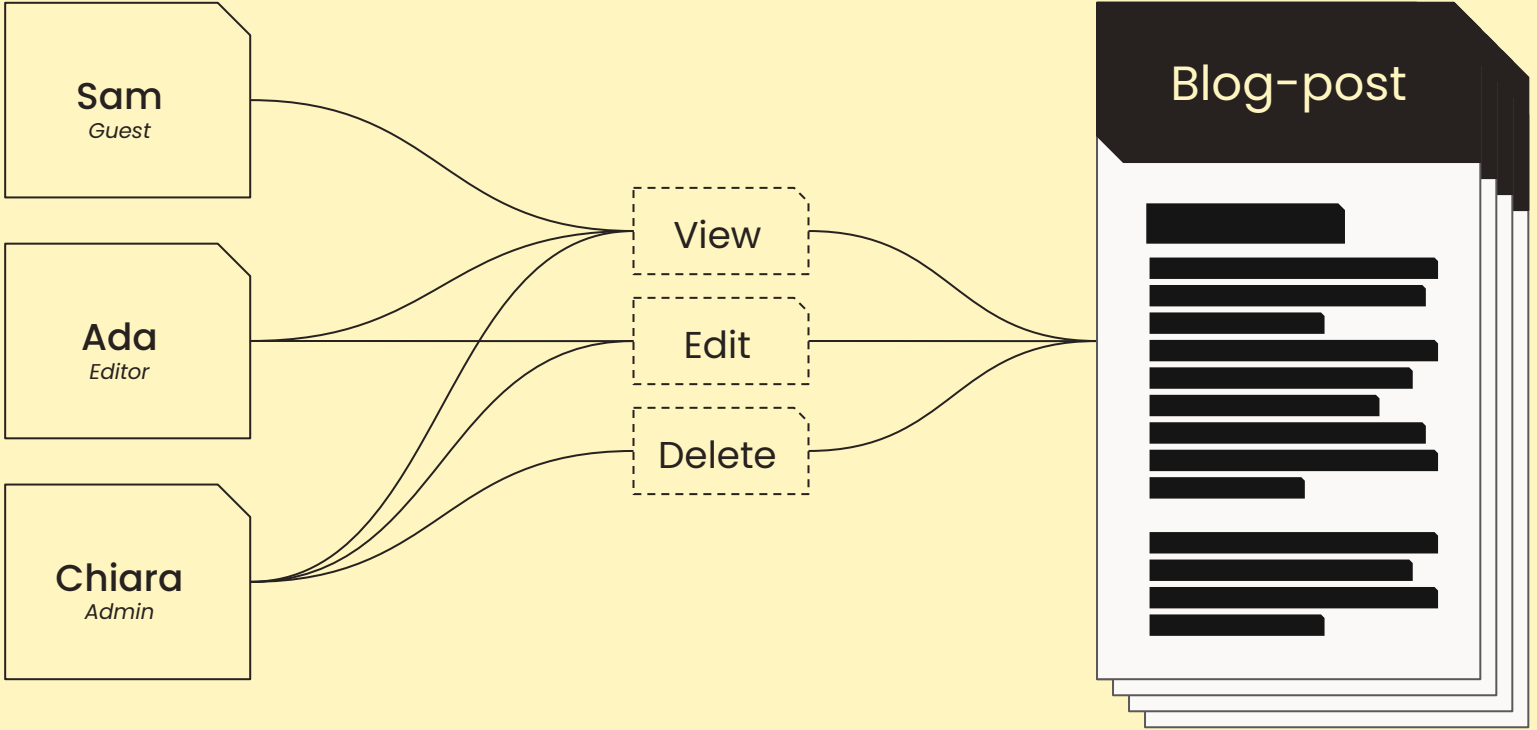


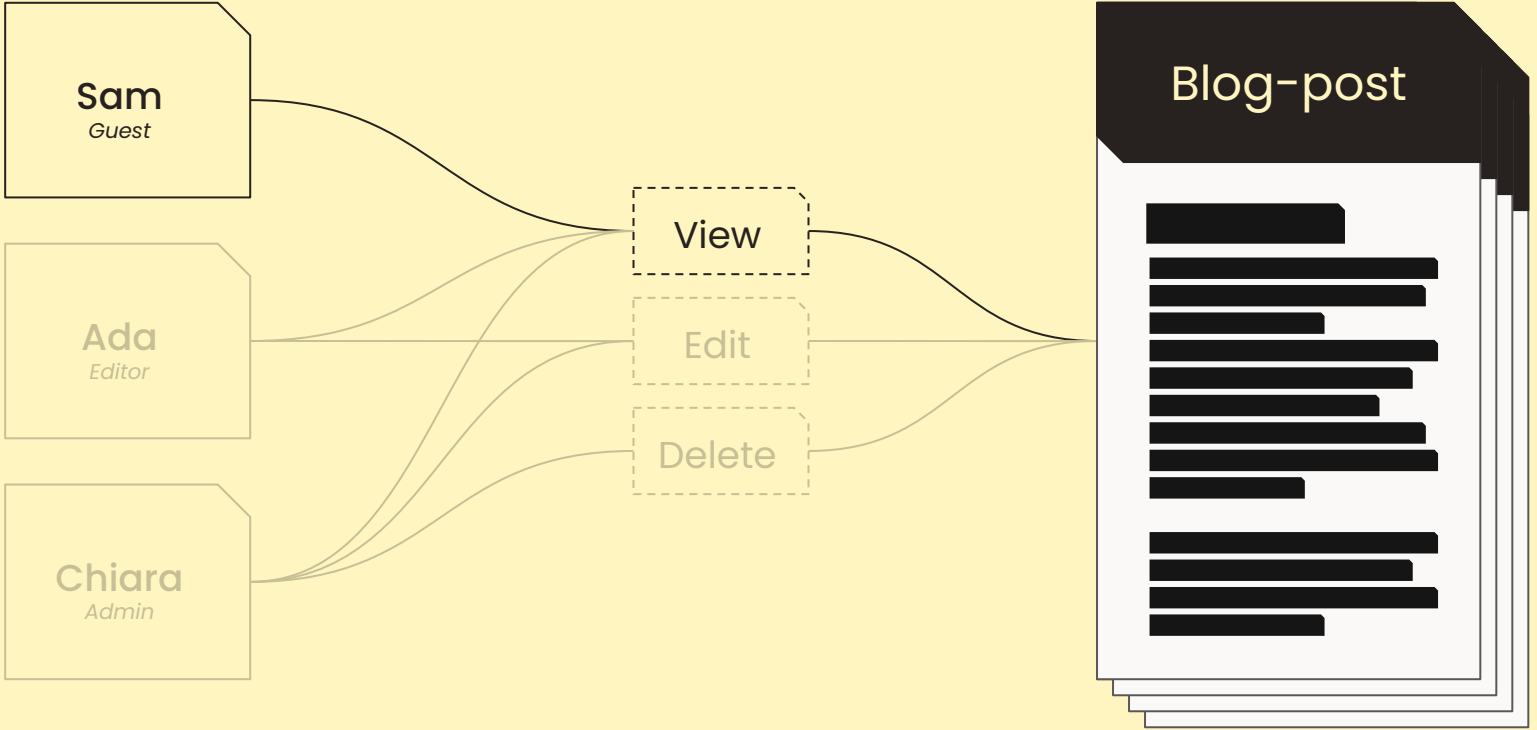
# Access Control?

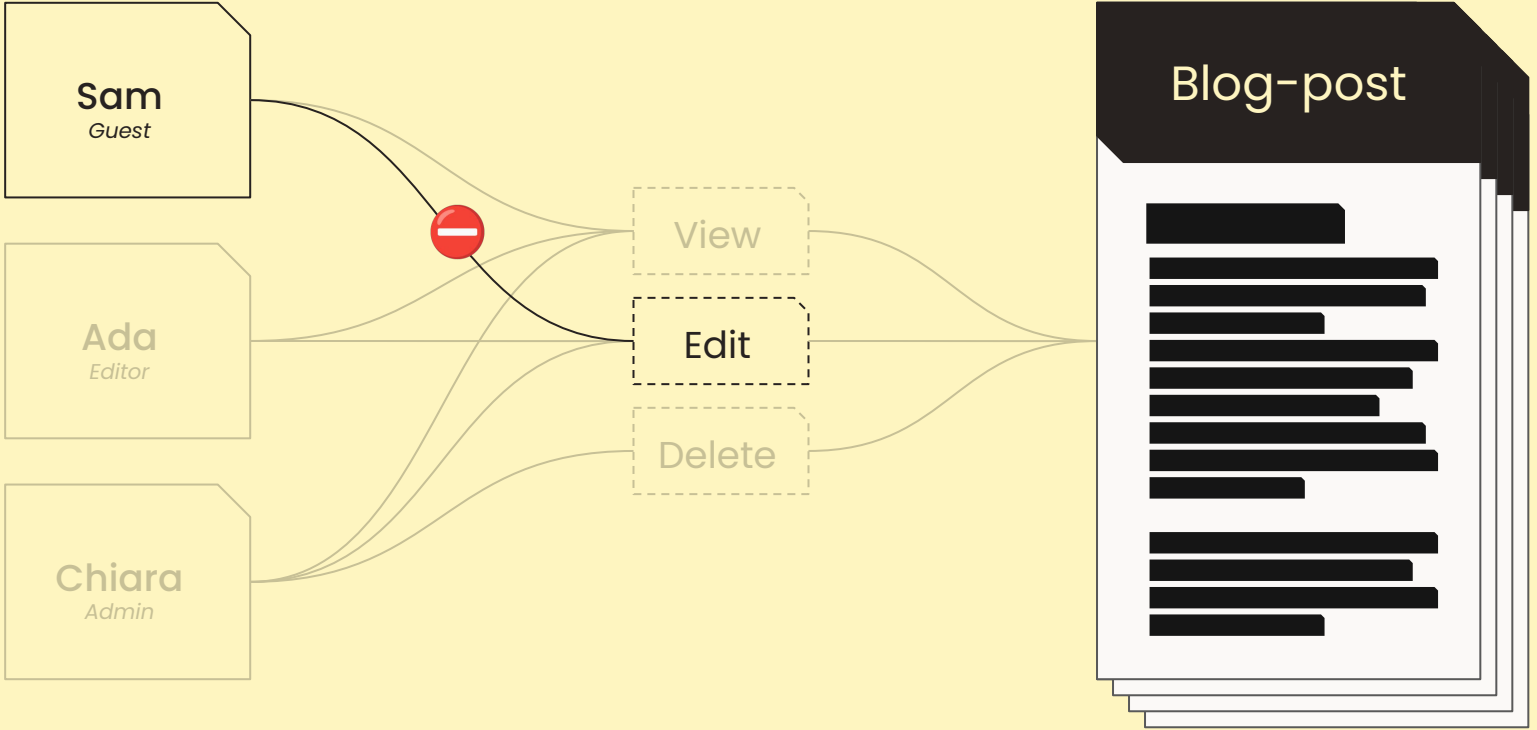
*Can this user do that action?*



An *access control method* decides whether **an action** **is allowed or not**







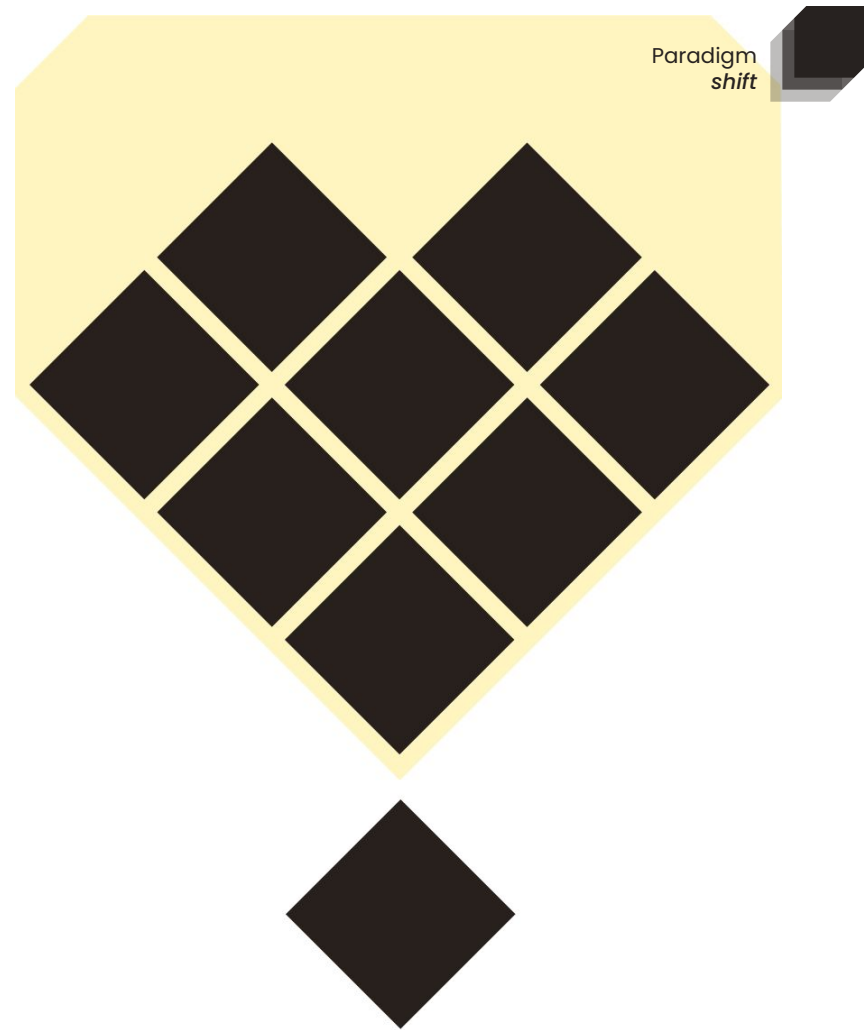




# *Fine-grained* Access control

Take back control!

*Coarse  
grained*





*Access control decisions*  
are made based on  
**resource type**



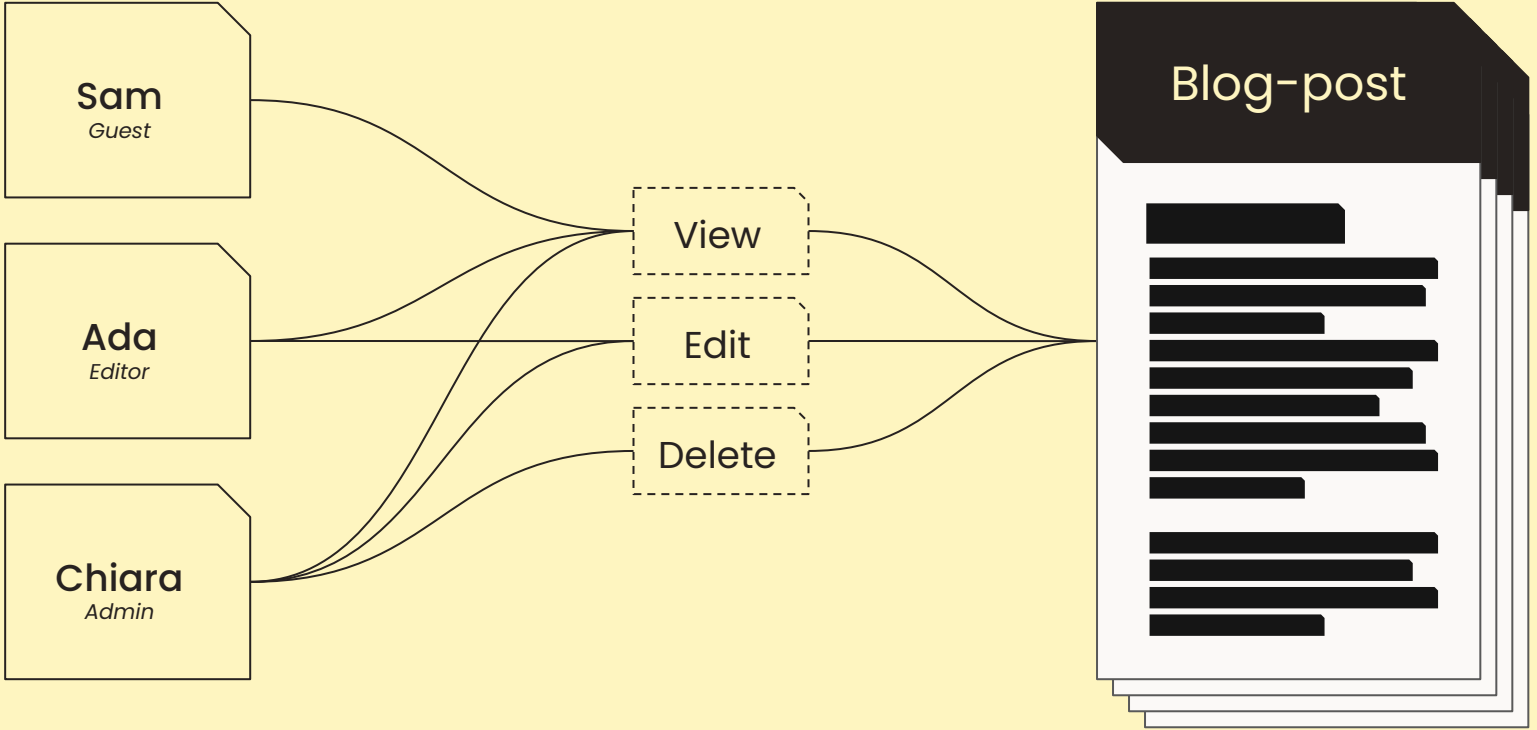
Example

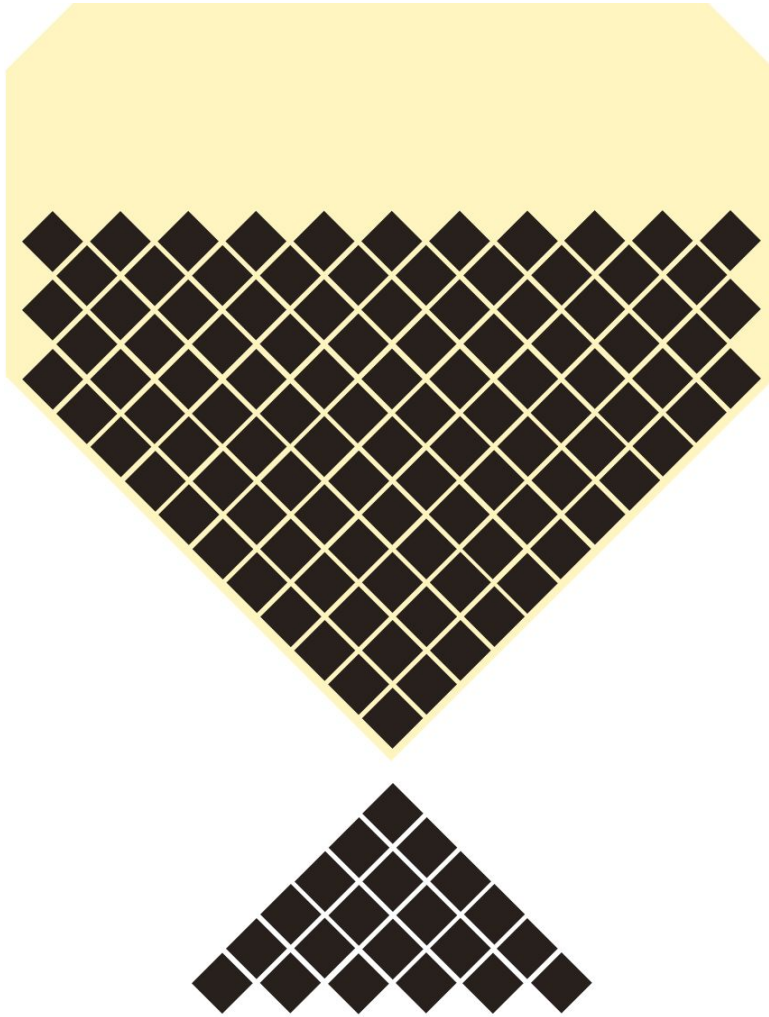
An editor can edit  
all blog-posts



Example

An **admin** can delete  
**all blog-posts**





*Fine*  
grained



*Access control decisions*  
are made for a **specific**  
**resource or situation**

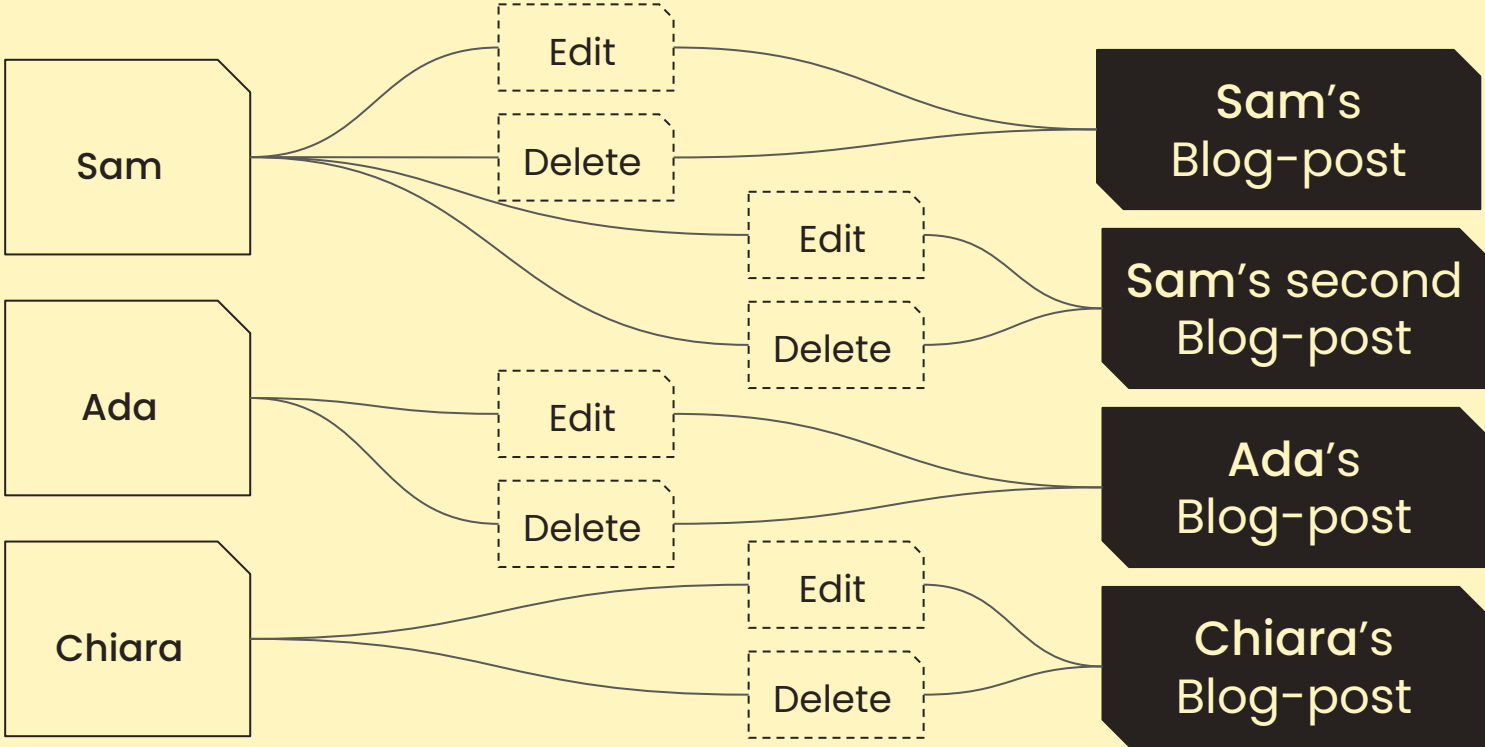


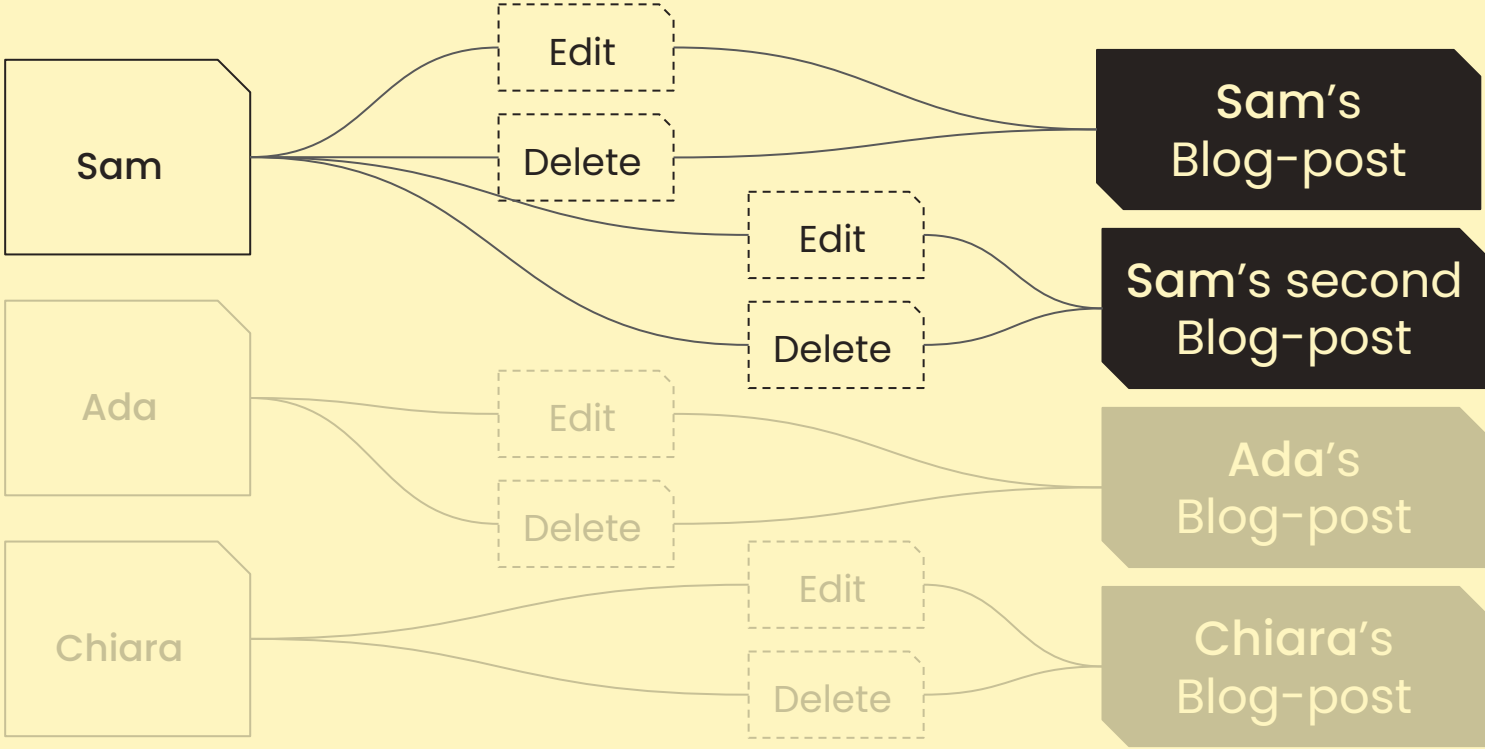


Example

An owner of a post can  
delete their own post

Fine-grained



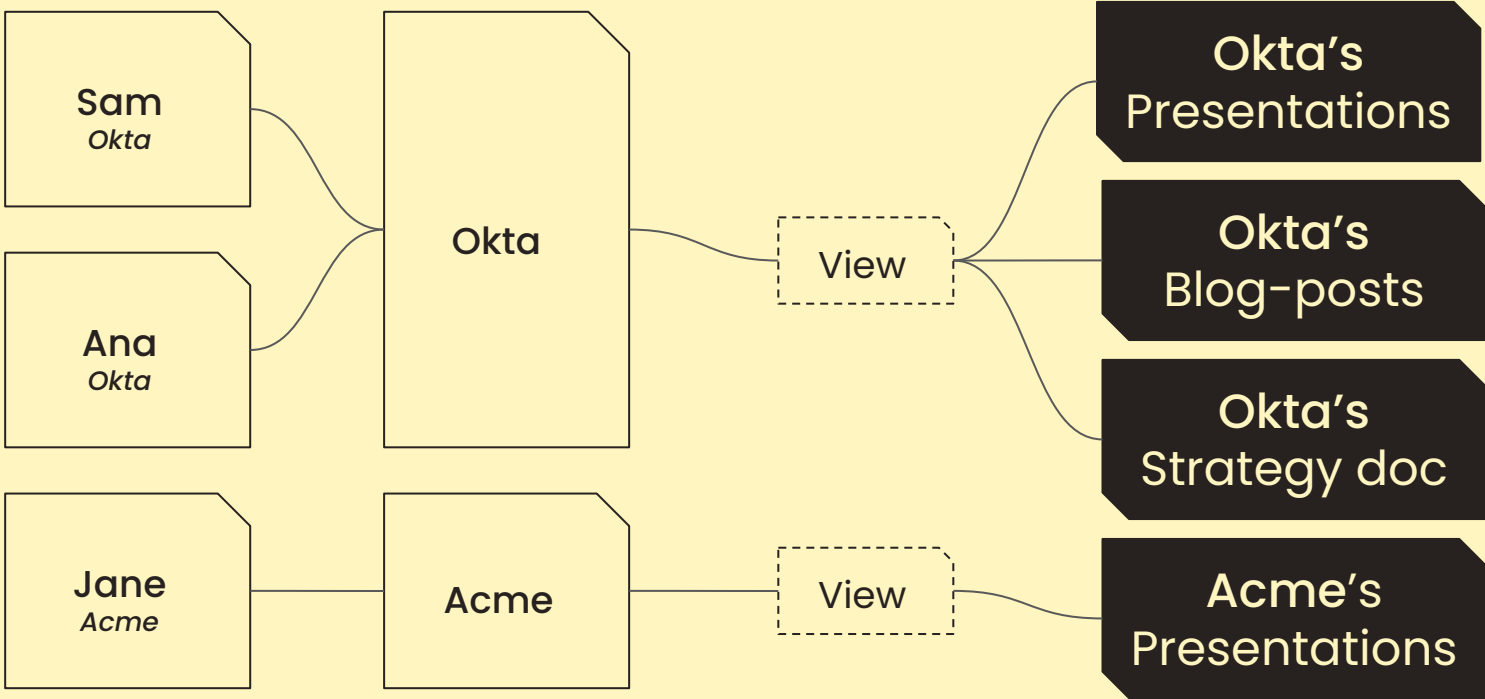


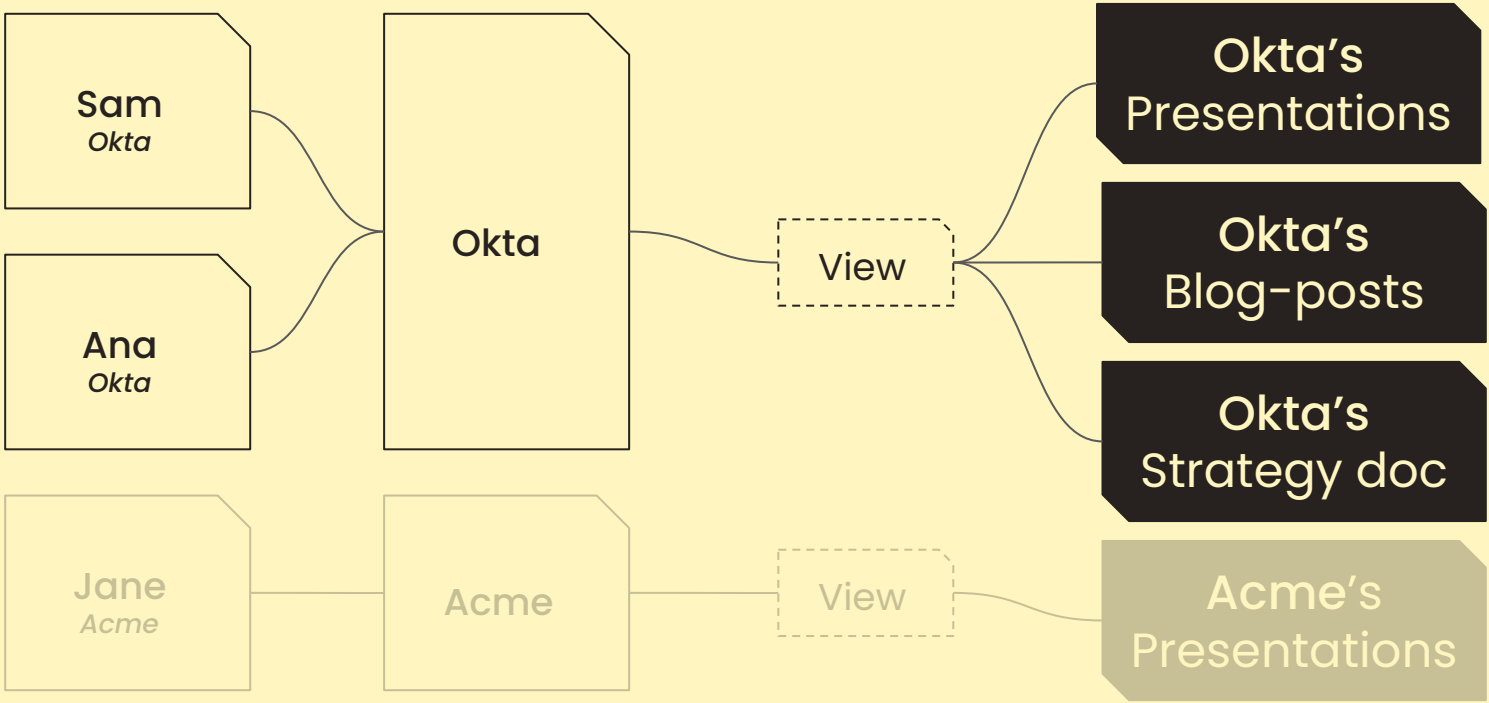


Example

A member of an organization can view all files belonging to that organization

Fine-grained



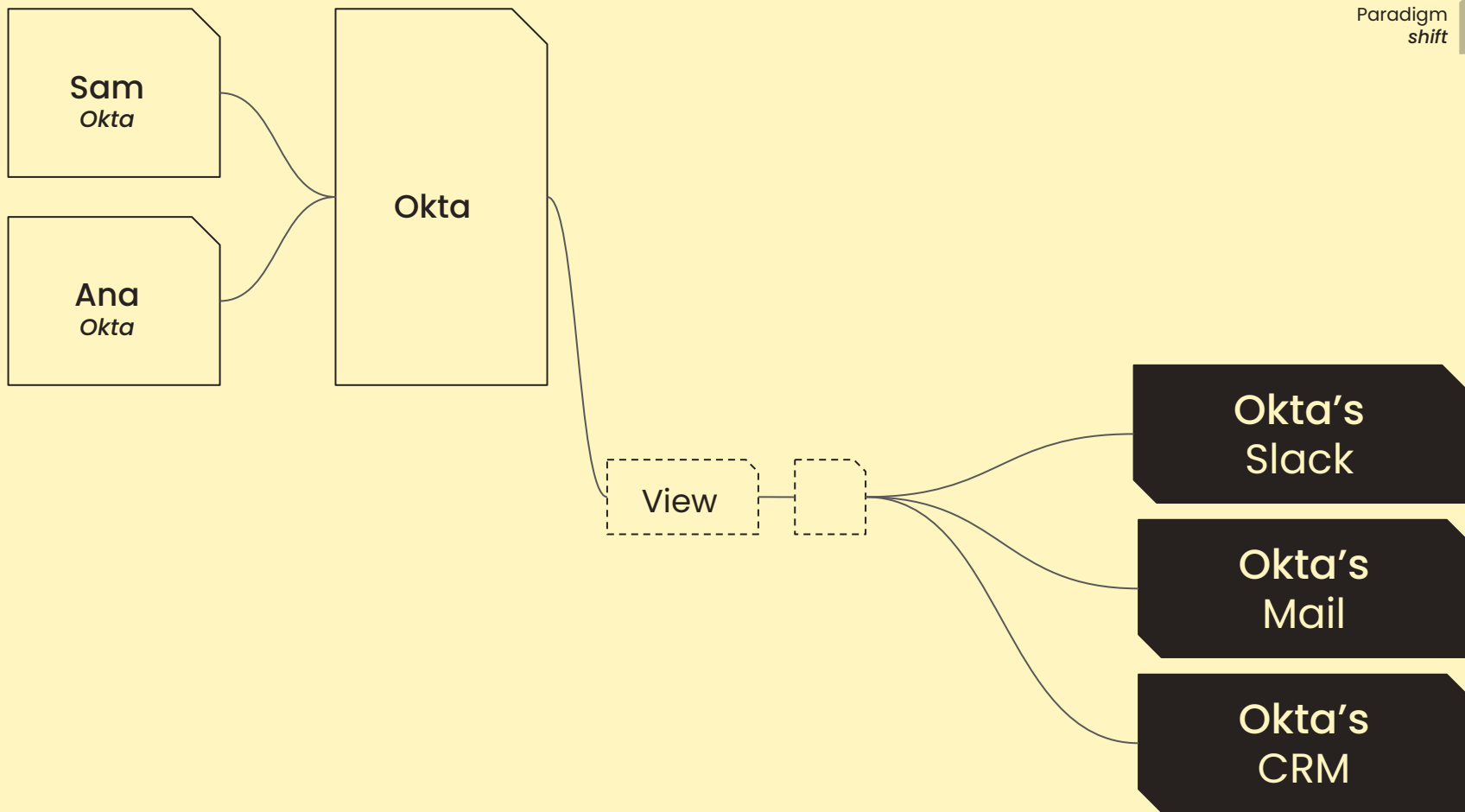




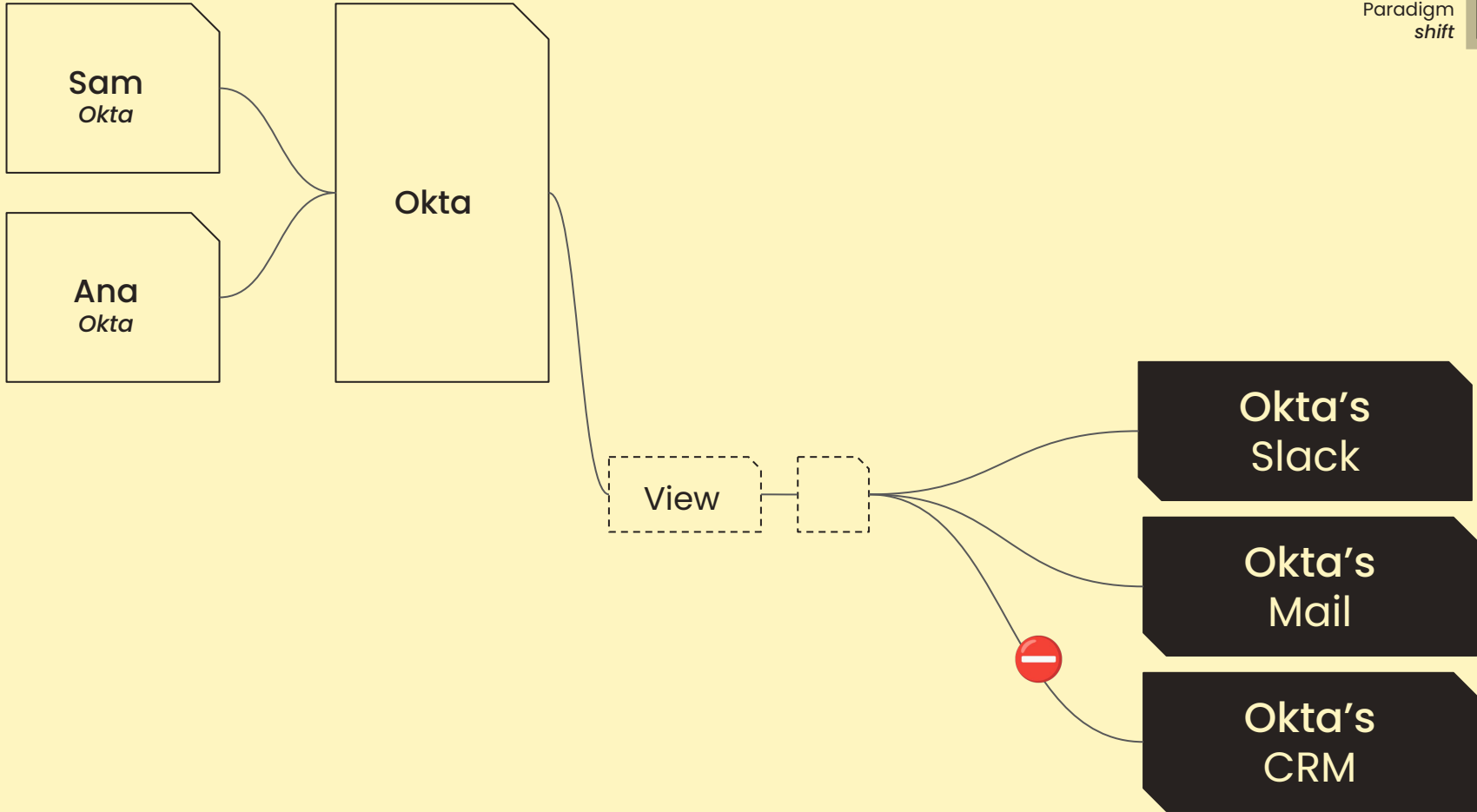
Example

An **employee** can access  
the CRM **during office**  
**hours**

Fine-grained









# *Coarse* grained

Smaller applications  
No user-generated content

# *Fine* grained

User-generated content  
Multi tenant applications  
Sharing resources  
Sensitive data  
Contextual or environmental  
decisions



RBAC

# *Role-based* Access control

*Roles and permissions* define access



*Decisions* are based on  
**permissions** assigned to a  
user via **roles**



# Roles

Guest

Admin

Editor

Paying customer

Enterprise



# Permissions

View

Create

Edit

Delete



# Auth0

## *Roles and permissions*

*RBAC* straight out of the box

role Details

manage.auth0.com/dashboard/eu/sambego/roles/rol\_t9pk4wDmPp3RxdvZ/permissions

sambego

Search Discuss your needs Documentation

Activity Applications Authentication Organizations User Management Users Roles Branding Security Actions Auth Pipeline Monitoring Marketplace Extensions Settings

Get support

← Back to Roles

### Cat person

Role ID: rol\_t9pk4wDmPp3RxdvZ

Settings **Permissions** Users

Add Permissions to this Role. Users who have this Role will receive all Permissions below that match the API of their login request. [Add Permissions](#)

Permission	Description	API
read:cats	See all cats	nestjs-api

Manage *roles* for your users in the user management section of the dashboard



The screenshot shows the 'role details' page for a role named 'Cat person'. The page is divided into three tabs: 'Settings', 'Permissions', and 'Users'. The 'Permissions' tab is active, showing a table of permissions. A circular callout highlights the 'Permissions' tab and the table content.

← Back to Roles

### Cat person

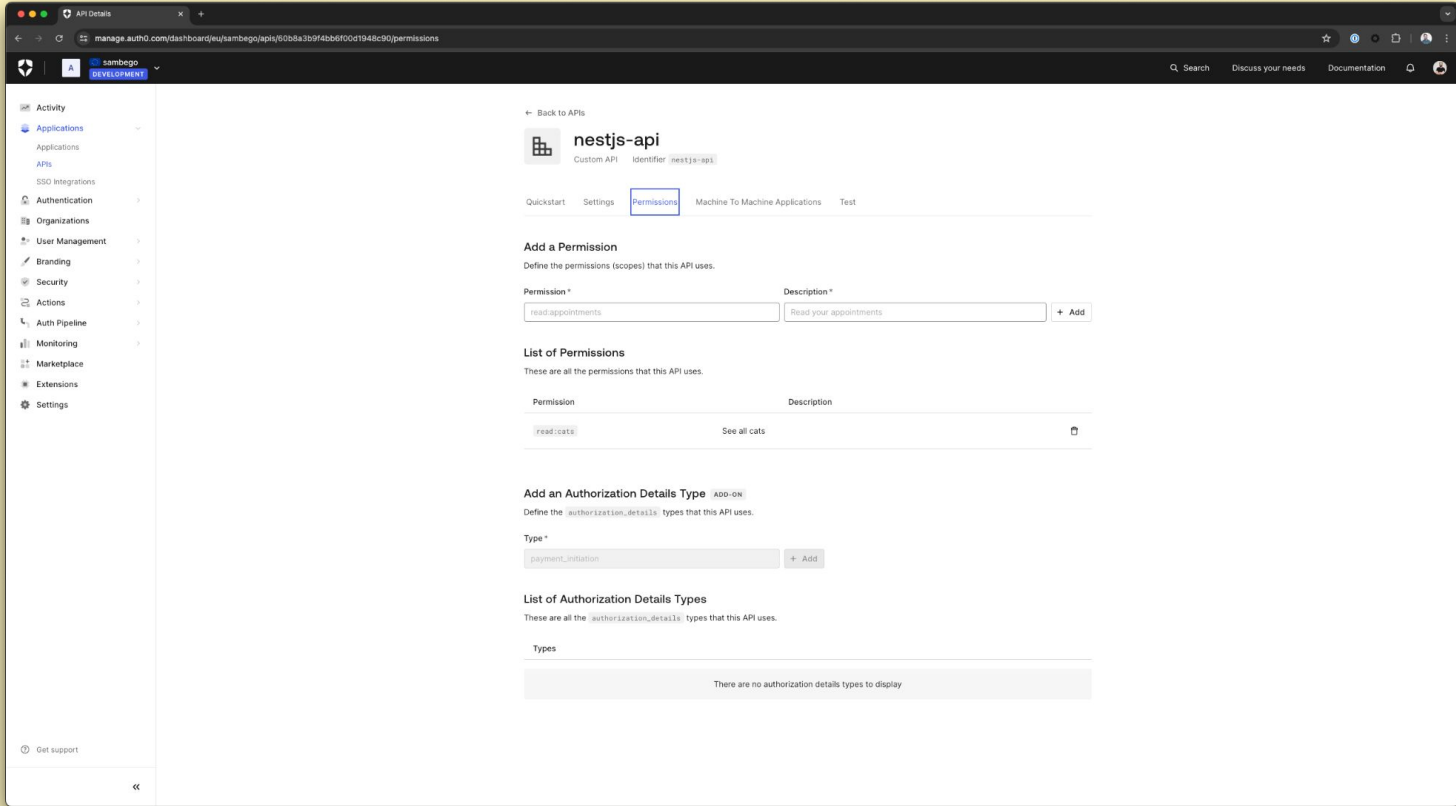
Role ID: `roL_t9pK4wDnVp3R8x9vZ`

Settings **Permissions** Users

Add Permissions to this Role. Users who have this Role will receive all Permissions below that match the request.

Permission ^	Description
<code>read:cats</code>	See all cats

Manage *roles* for your users in the user management section of the dashboard

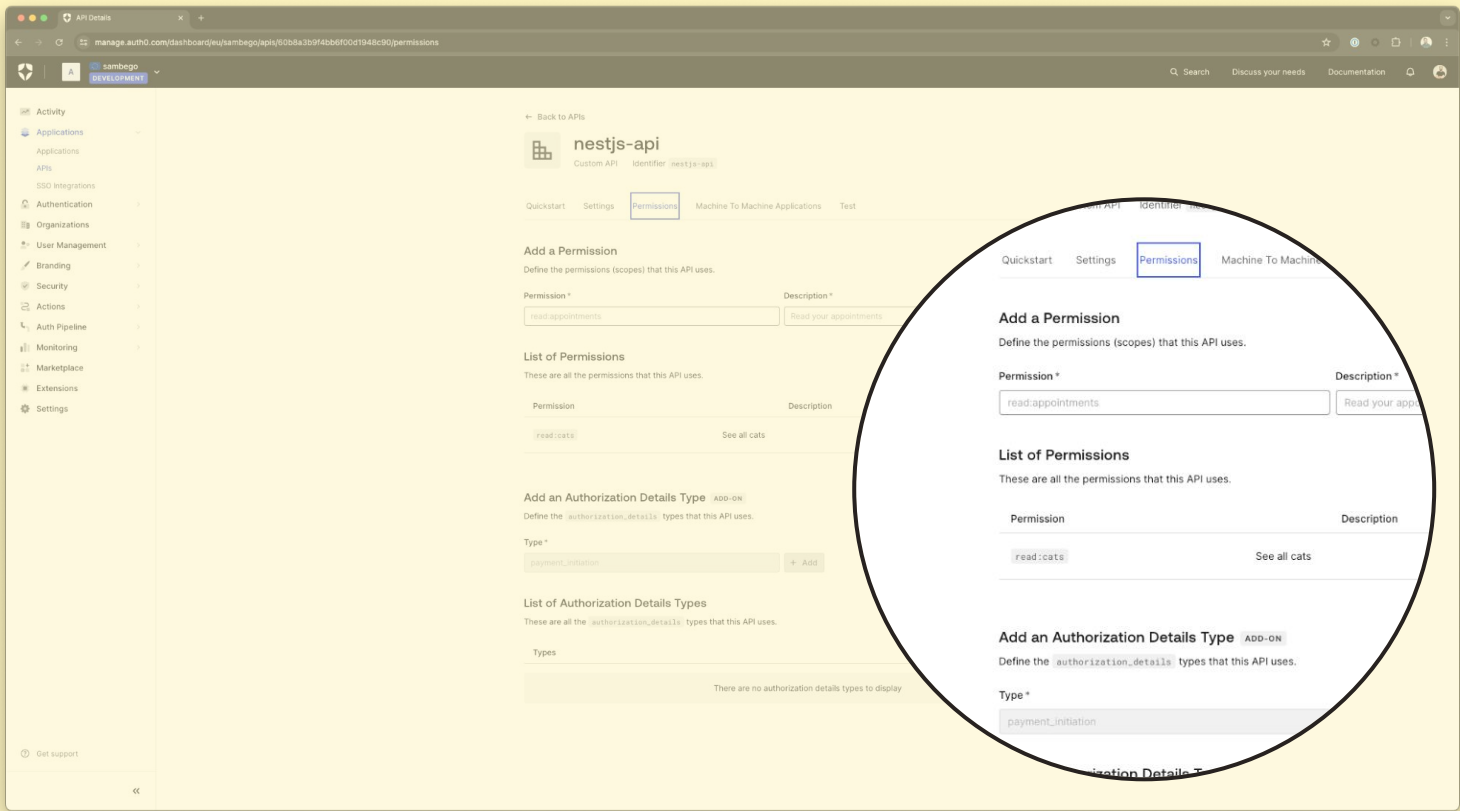


The screenshot shows the 'Permissions' configuration page for the 'nestjs-api' in the API management console. The page is divided into several sections:

- Navigation:** A sidebar on the left contains menu items like Activity, Applications, APIs, SSO Integrations, Authentication, Organizations, User Management, Branding, Security, Actions, Auth Pipeline, Monitoring, Marketplace, Extensions, and Settings.
- API Details:** At the top, it identifies the API as 'nestjs-api' (Custom API) with the identifier 'nestjs-api'.
- Permissions Section:**
  - Add a Permission:** A form to define permissions with fields for 'Permission \*' (containing 'read:appointments') and 'Description \*' (containing 'Read your appointments').
  - List of Permissions:** A table showing existing permissions. One entry is visible:
 

Permission	Description
read:cats	See all cats
- Authorization Details Section:**
  - Add an Authorization Details Type:** A form to define authorization details types. One entry is visible: 'payment\_initiation'.
  - List of Authorization Details Types:** A section indicating that there are no authorization details types to display.

Manage *permissions* for your APIs



Manage *permissions* for your APIs



## RBAC Settings

### Enable RBAC



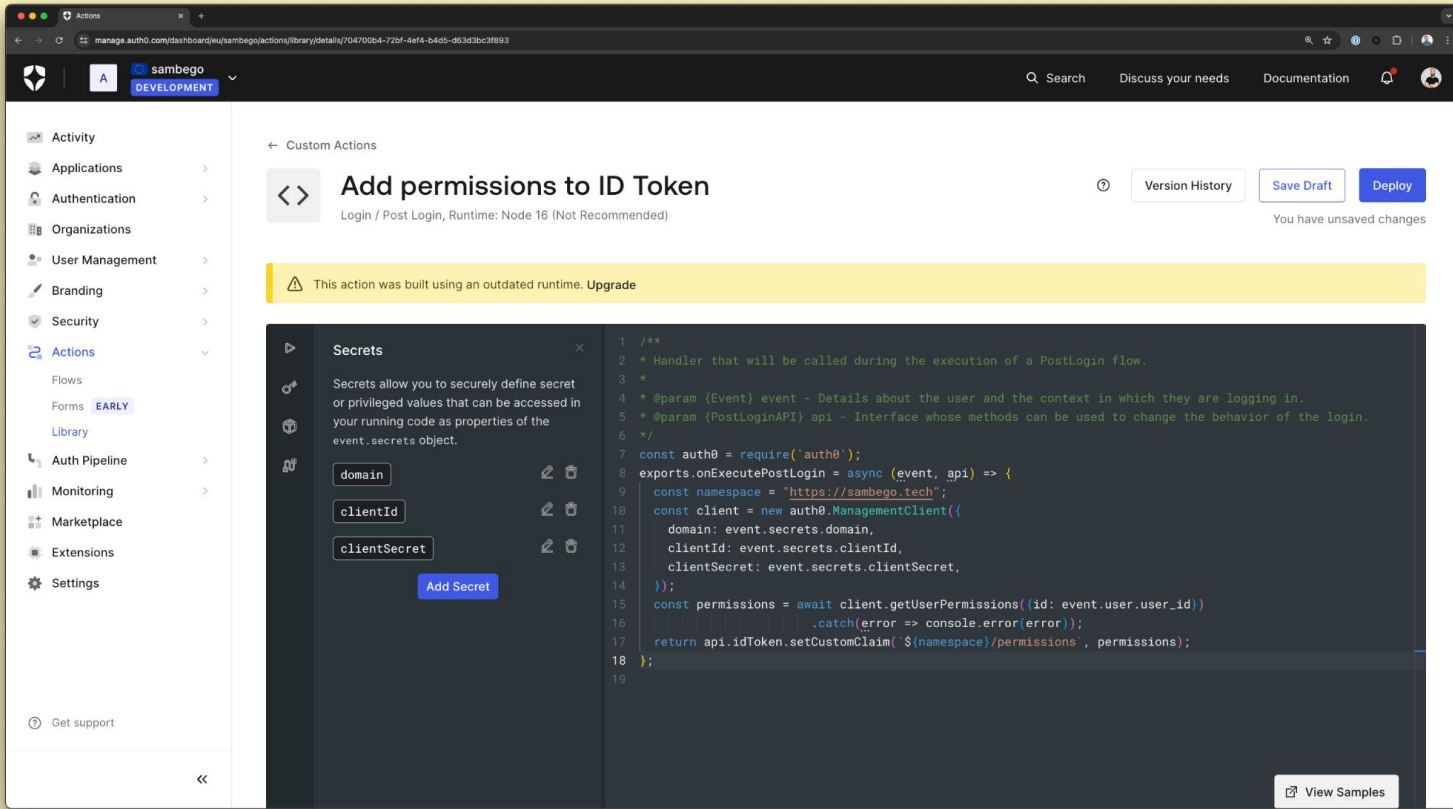
If this setting is enabled, RBAC authorization policies will be enforced for this API. [Role](#) and permission assignments will be evaluated during the login transaction.

### Add Permissions in the Access Token



If this setting is enabled, the Permissions claim will be added to the access token. Only available if RBAC is enabled for this API.

Add *permissions* to the access tokens issued by Auth0



The screenshot shows the Auth0 dashboard interface. On the left is a navigation sidebar with categories like Activity, Applications, Authentication, Organizations, User Management, Branding, Security, Actions, Auth Pipeline, Monitoring, Marketplace, Extensions, and Settings. The main content area is titled 'Custom Actions' and features a header for 'Add permissions to ID Token' with buttons for 'Version History', 'Save Draft', and 'Deploy'. A yellow warning banner states: 'This action was built using an outdated runtime. Upgrade'. Below this is a 'Secrets' panel with input fields for 'domain', 'clientId', and 'clientSecret', and an 'Add Secret' button. To the right of the secrets panel is a code editor showing the following JavaScript code:

```

1 /**
2  * Handler that will be called during the execution of a PostLogin flow.
3  *
4  * @param {Event} event - Details about the user and the context in which they are logging in.
5  * @param {PostLoginAPI} api - Interface whose methods can be used to change the behavior of the login.
6  */
7 const auth0 = require('auth0');
8 exports.onExecutePostLogin = async (event, api) => {
9   const namespace = "https://sambego.tech";
10  const client = new auth0.ManagementClient({
11    domain: event.secrets.domain,
12    clientId: event.secrets.clientId,
13    clientSecret: event.secrets.clientSecret,
14  });
15  const permissions = await client.getUserPermissions({id: event.user.user_id})
16    .catch(error => console.error(error));
17  return api.idToken.setCustomClaim(`${namespace}/permissions`, permissions);
18 };
19 
```

At the bottom right of the code editor, there is a 'View Samples' button.

Add *permissions* to the ID token through  
Auth0 actions



*Watch out* for **token bloat!**



ABAC

# *Attribute-based* Access control

Different kinds of *attributes*  
influence the decision



*Decisions* are based on  
**attributes** related to the  
action being evaluated





# Attributes

User

Environment

Resource

Action



# User

Role

Organization

Security clearance

# Environment

Time of day

Location of data

Code-freeze

Current threat level

# Resource

Creation date

Owner

Data Sensitivity



# Action

Read

Write

Delete



PBAC

# *Policy-based* Access control

Combine attributes  
in *policies*



Declare scenarios with  
*attributes* in one or more  
policy



*A policy engine will*  
**evaluate access control**  
**decisions**





Example

**An accountant can upload an invoice, if it is during their working hours.**



Example

An accountant can upload  
an invoice, if it is during  
their working hours.

User



Example

An accountant can upload  
an invoice, if it is during  
their working hours.

Action



Example

An accountant can upload  
an invoice, if it is during  
their working hours.

Resource



Example

An accountant can upload  
an invoice, if it is during  
their working hours.

Environment



Example

**Any engineer can write to any file, when we are not having a code freeze.**



Example

Any engineer can write to  
any file, when we are not  
having a code freeze.

User



Example

Any engineer can write to  
any file, when we are not  
having a code freeze.

Action





Example

Any engineer can write to  
any file, when we are not  
having a code freeze.

Resource



Example

Any engineer can write to  
any file, when we are not  
having a code freeze.

Environment



**ABAC** is often used for  
managing infrastructure  
access



Scenarios where decisions are made on a **limited set of data**, the *attributes*



OPA

# Open *Policy* Agent

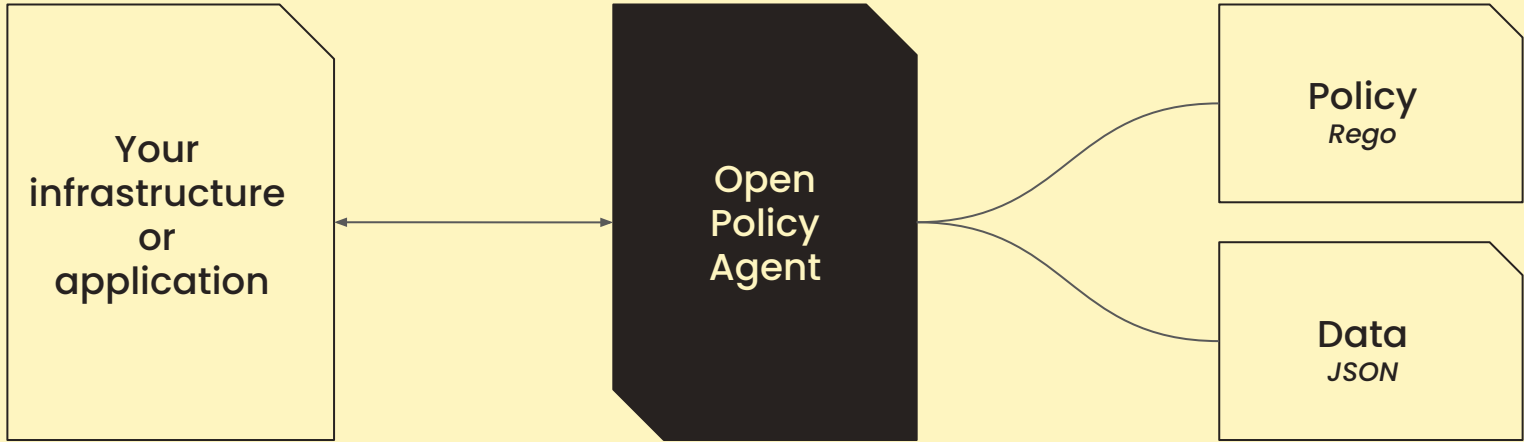
Open-source *policy engine*



*OPA* is a **decision-engine**  
that provides a way of  
declaratively writing  
**policies as code.**



It uses these *policies* as  
part of a **decision-making**  
**process.**







Example

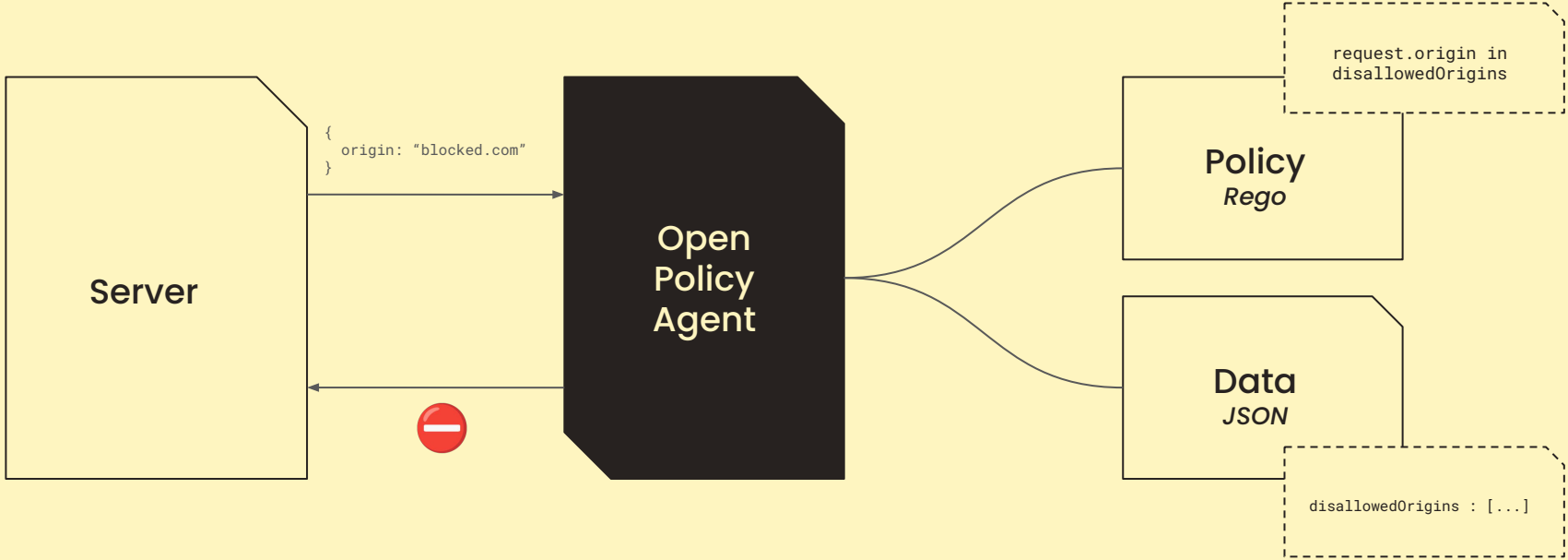
Block request coming from  
an origin on our block-list.

Environment



```
deny[reason] {  
    disallowedOrigins := ["blocked.com", "malicious.com"]  
  
    some input.request.origin in disallowedOrigins  
    reason := sprintf("origin %s has been blocked", [input.request.origin])  
}
```

*A policy* written in the Rego language  
used by OPA





***ABAC*** isn't optimized for  
large amounts of data  
that's continually  
changing.



ReBAC

# *Relationship-based* Access control

Look for *relationships* between  
resources



***Access control decisions***  
are based on relationships  
between a consumer and  
a resource



***A consumer*** can be a **user,**  
**group, folder, another**  
**resource...**



Example

**Sam is the owner of this  
slidedeck**





Example

**Sam** is the owner of this  
**slidedeck**

Consumer



Example

Sam is the owner of this  
slidedeck

Resource



Example

Sam is the **owner** of this  
slidedeck

Relation



There are **direct** and  
**indirect** *relationships*.



*Direct relationships* are  
explicitly defined



*Indirect relationships* are  
derived from the direct  
relationships



Example

Sam is a member of Auth0,  
and can therefore view all  
slide decks in our  
organisation.



Example

Sam is indirectly related to  
this slide deck.





# *OpenFGA*

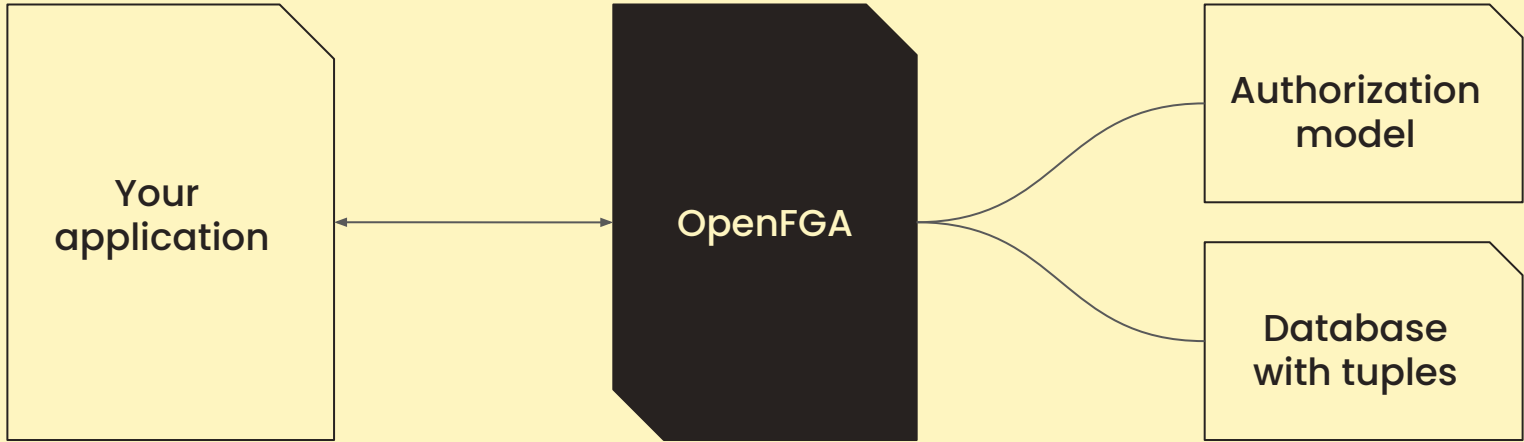
*An open-source ReBAC solution, a  
CNCf sandbox project*



*OpenFGA* is a decision-engine that makes decisions based on an authorization model in combination with tuples saved in a database.



These *tuples* are the **direct**  
**relationships.**





Example

Check if a user can view a  
GitHub repository

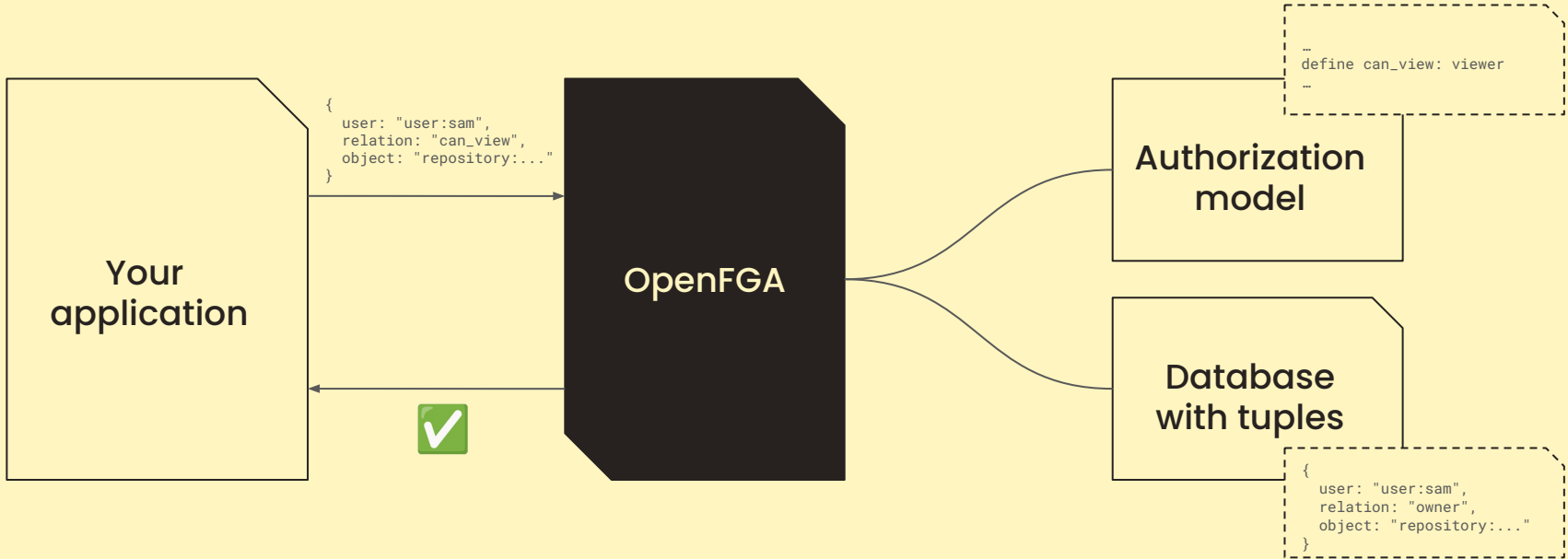
Relation



```
type user
type repository
  relations
    define owner: [user]
    define viewer: [user] or owner
    define can_view: viewer or owner
```

```
{
  user: "user:sam",
  relation: "owner",
  object: "repository:fga-demo"
}
```

*An OpenFGA authorization model and tuple*





Example

Check if a user  
can commit to a GitHub  
repository

Relation





```
type user
```

```
type organization
```

```
  relations
```

```
    define member: [user]
```

```
    define repo_writer: [user] or member
```

```
type repository
```

```
  relations
```

```
    define owner: [organization]
```

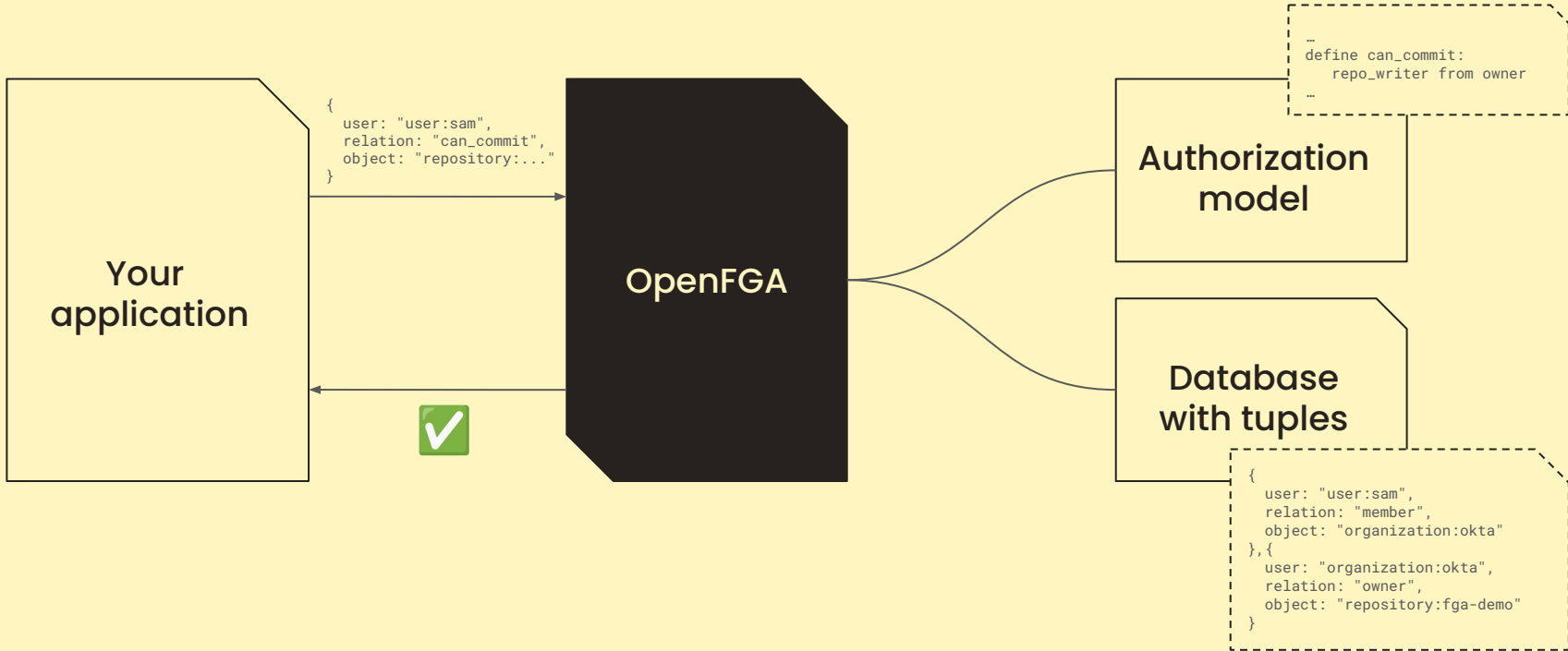
```
    define can_commit: repo_writer from owner
```

An OpenFGA *authorization model*



```
{  
  user: "user:sam",  
  relation: "member",  
  object: "organization:okta"  
}, {  
  user: "organization:okta",  
  relation: "owner",  
  object: "repository:fga-demo"  
}
```

OpenFGA *tuples*





Example

*Retrieval-augmented  
generation (RAG):* Retrieve  
financial documents  
assigned to a user

Relation



```
type user
```

```
type document
```

```
  relations
```

```
    define owner: [user]
```

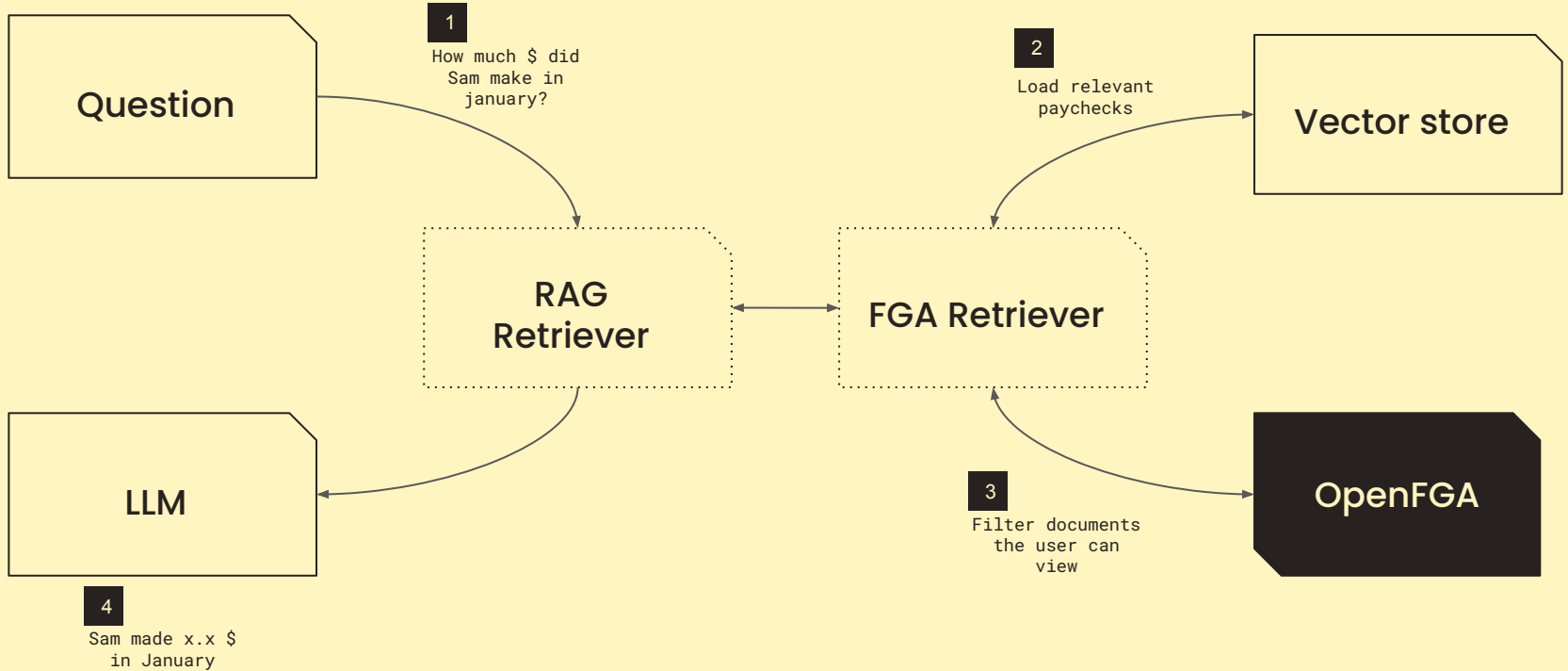
```
    define can_view: [user] or owner
```

An OpenFGA *authorization model*



```
{
  user: "user:sam",
  relation: "owner",
  object: "document:paycheck-jan-2024-sam"
},{
  user: "user:hr-rudy",
  relation: "can_view",
  object: "document:paycheck-jan-2024-sam"
},{
  user: "user:jane",
  relation: "owner",
  object: "document:paycheck-jan-2024-jane"
},{
  user: "user:hr-rudy",
  relation: "can_view",
  object: "document:paycheck-jan-2024-jane"
}
```

OpenFGA *tuples*





# *Okta FGA*

*A managed* OpenFGA service



Model Explorer | FGA
dashbboard.fga.dev/customers/01FXQD5Y3PR8B1EQ6CPSH9QF1/stores/01HRS4\_JVWYF8GV1DT4N8WPV2C4/models

AutD by Okta
Fine Grained Authorization FREE TRIAL
sambego ▼ Manage Account Docs

**SELECT STORE**

drive ▼

---

**Getting Started**

- Model Explorer
- Tuple Management
- Developer Mode
- Settings

---

Upgrade to an Enterprise subscription to use Okta FGA in production. [Learn More](#)

Get support

## Model Explorer

Define your authorization model. Specify the types of objects in your application and the relations for each of those types. [Learn more](#)

Save
Preview

**Model**

```

1 model
2   schema 1.1
3
4 type user
5
6 type file
7 relations
8   define can_delete: owner or owner from parent
9   define can_share: owner or owner from parent
10  define can_view: viewer or owner or viewer from parent
11  define can_write: owner or owner from parent
12  define is_owned: owner
13  define is_shared: can_view but not owner
14  define owner: [user]
15  define parent: [folder]
16  define viewer: [user, user=+]
17
18 type folder
19 relations
20   define can_create_file: owner or owner from parent
21   define can_create_folder: owner or owner from parent
22   define can_share: owner or owner from parent
23   define can_view: viewer or owner or viewer from parent
24   define owner: [user]
25   define parent: [folder]
26   define viewer: [user, user=+] or owner or viewer from parent
27
                
```

Latest ID 01HSTS38X6Q2EYRW7QKT0S6J24

Developer Mode | FGA

dashboards.fga.dev/customers/01FXQDSY3PRB91EQ6CPSH9QF1ntores/01HRS4\_JVVYF8GV1DT4N8WPV2C4/dev-mode

Auto by Okta Fine Grained Authorization FREE TRIAL sambego Manage Account Docs

## Developer Mode

Iteratively develop your Okta FGA store. Use a single view to quickly shift between your authorization model, tuples and assertions. Tuples **should not** contain Personal Identifiable Information. [Learn more](#)

[Guided Tour](#)

**Model** Preview Q Save

```

1 model
2 schema 1.1
3
4 type user
5
6 type file
7 relations
8   define can_delete: owner or owner from parent
9   define can_share: owner or owner from parent
10  define can_view: viewer or owner or viewer from parent
11  define can_write: owner or owner from parent
12  define is_owned: owner
13  define is_shared: can_view but not owner
14  define owner: [user]
15  define parent: [folder]
16  define viewer: [user, user+]
17
18 type folder
19 relations
20   define can_create_file: owner or owner from parent
21   define can_create_folder: owner or owner from parent
22   define can_share: owner or owner from parent
23   define can_view: viewer or owner or viewer from parent
24   define owner: [user]
25   define parent: [folder]
26   define viewer: [user, user+] or owner or viewer from parent
27

```

Latest ID 01HSTS3BX9QXEYRW7OKT0S6JZ4

**Tuples** Preview Q Save

Type to filter tuples...

+ Add Tuple

USER	user:auth0 65faef64e18ff2c62b63cd9a	
OBJECT	folder:86f2bb93-a88c-48a8-9528-d99682...	
RELATION	viewer	
USER	user:auth0 65faef64e18ff2c62b63cd9a	
OBJECT	file:49384b83-13bc-483c-a4fe-8b68f72db...	
RELATION	viewer	
USER	user:google-oauth2 1135799221734285377...	
OBJECT	file:62f35382-c928-464d-a132-57953752...	
RELATION	owner	
USER	folder:61b2e898-c5bc-47db-a821-140c218...	
OBJECT	file:62f35382-c928-464d-a132-57953752...	
RELATION	parent	
USER	user:google-oauth2 1135799221734285377...	
OBJECT	file:e01f61d4-59fc-4332-85db-9bcfb9f73b...	
RELATION	owner	
USER	folder:61b2e898-c5bc-47db-a821-140c218...	
OBJECT	file:e01f61d4-59fc-4332-85db-9bcfb9f73b...	
RELATION	parent	

**Assertions** Query >

Type to filter assertions...

+ Add Assertion

Assertions allow you to save a list of statements or tests and run them - individually or as a group - to make it easy to iterate on your namespace configuration.

Click Add Assertion to get started

Run All Run 0 Selected



*ReBAC* is a great solution for applications that deal with user-generated content, that is constantly changing



# *(de)centralized* access control

*Where* should your authorization  
decisions be made?



***Modern ABAC and ReBAC solutions*** have a decision engine that centralizes decisions from your applications



This is useful for audits!



*All applications* will use the  
same decisions



# *Hybrid\**

# access control

Modern tools can support *multiple paradigms*





*Some access control solutions* have support for **multiple methods**, working around limitations inherent to their original strategy



Limitation

***ABAC*** does not work well  
with **dynamic data**



*OPAL* adds an  
administration layer to  
*OPA*, to keep policies and  
data in sync across agents



*TOPAZ* brings “real-time,  
policy based access  
control for applications  
and APIs” to *OPA*



Limitation

*ReBAC* does not take  
context or environment  
into consideration



*OpenFGA* has ABAC  
support through  
**contextual tuples and  
conditions**



# Standards?

*Available authorization standards*



***XACML***, extensible access  
control markup language





~~**XACML**, extensible access  
control markup language~~



*AuthZEN* provide standard mechanisms to communicate authorization related information from different entities



*AuthZEN* is currently a first  
implementers draft at the  
*OpenID Foundation*.



```
{
  "subject": {
    "type": "user",
    "id": "sam",
  },
  "action": {
    "name": "can_view",
  },
  "resource": {
    "type": "document",
    "id": "paycheck-jan-2024-sam",
  },
  "context": {
    ...
  }
}
```

AuthZEN evaluation payload



```
{  
  "decision": true  
}
```

AuthZEN evaluation response

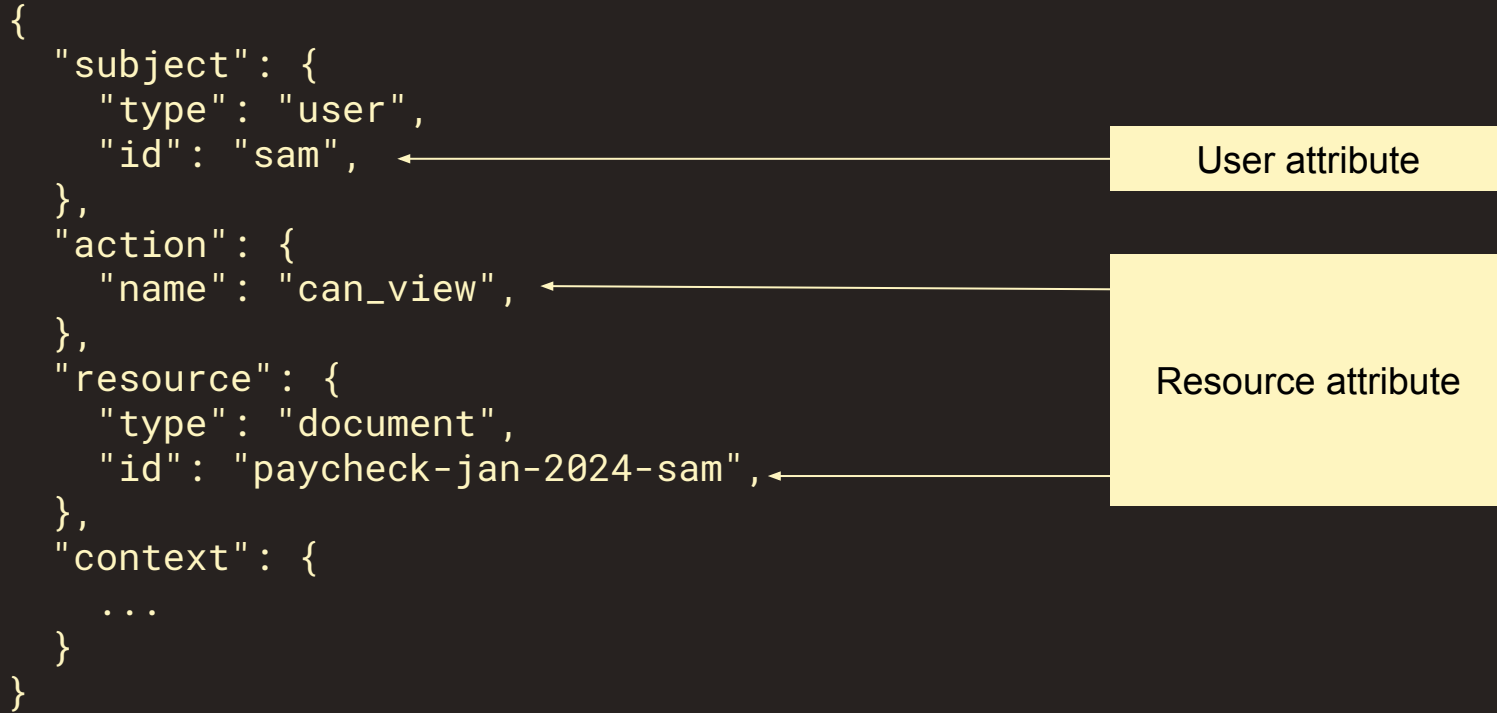


```
{
  "decision": false,
  "context": {
    "id": "0",
    "reason_admin": {
      "en": "No relation between user and object"
    },
    "reason_user": {
      "en-403": "Not allowed. Contact your administrator",
      "it-403": "Non consentito. Contatta l'amministratore"
    }
  }
}
```

AuthZEN evaluation response

```
{
  "subject": {
    "type": "user",
    "id": "sam", ← User
  },
  "action": {
    "name": "can_view", ← Relation
  },
  "resource": {
    "type": "document",
    "id": "paycheck-jan-2024-sam", ← Object
  },
  "context": {
    ...
  }
}
```

AuthZEN evaluation payload for  
OpenFGA



AuthZEN evaluation payload for OPA





# Let's *recap*

*TL;DR*



There are *different access control strategies*, they all have their purpose!



*RBAC* is good for applications that don't let their users generate content.



*ABAC* is great for  
fine-grained control based  
on a **limited set of**  
**attributes.**



*ReBAC* can evaluate a large database of direct relationships, and deduct indirect ones from this dataset.



This is *perfect* for complex applications with **an ever changing set of data** that influences the access control decision.



# Sam Bellen

*Principal developer advocate at Auth0*

@sambego  
sambego.tech



# Feedback

*[a0.to/jfokus-sam](https://a0.to/jfokus-sam)*

@sambego  
sambego.tech





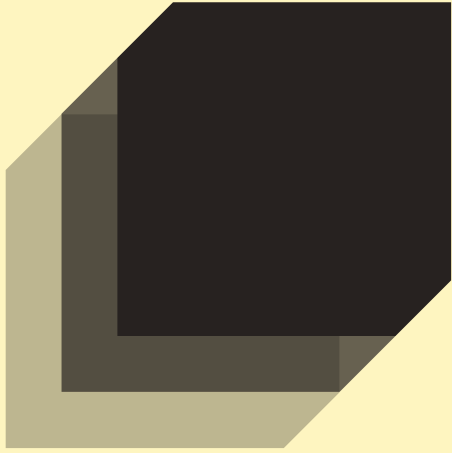
# Learn more

*[a0.to/fga-event](https://a0.to/fga-event)*

@sambego  
sambego.tech



Thank *you!*



# Paradigm *shift*

Moving beyond roles and permissions  
to a fine-grained access control