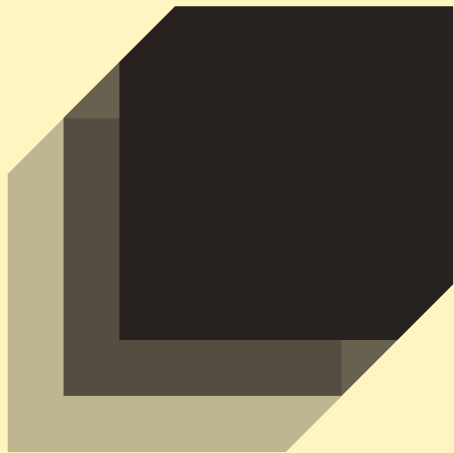




Paradigm *shift*

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

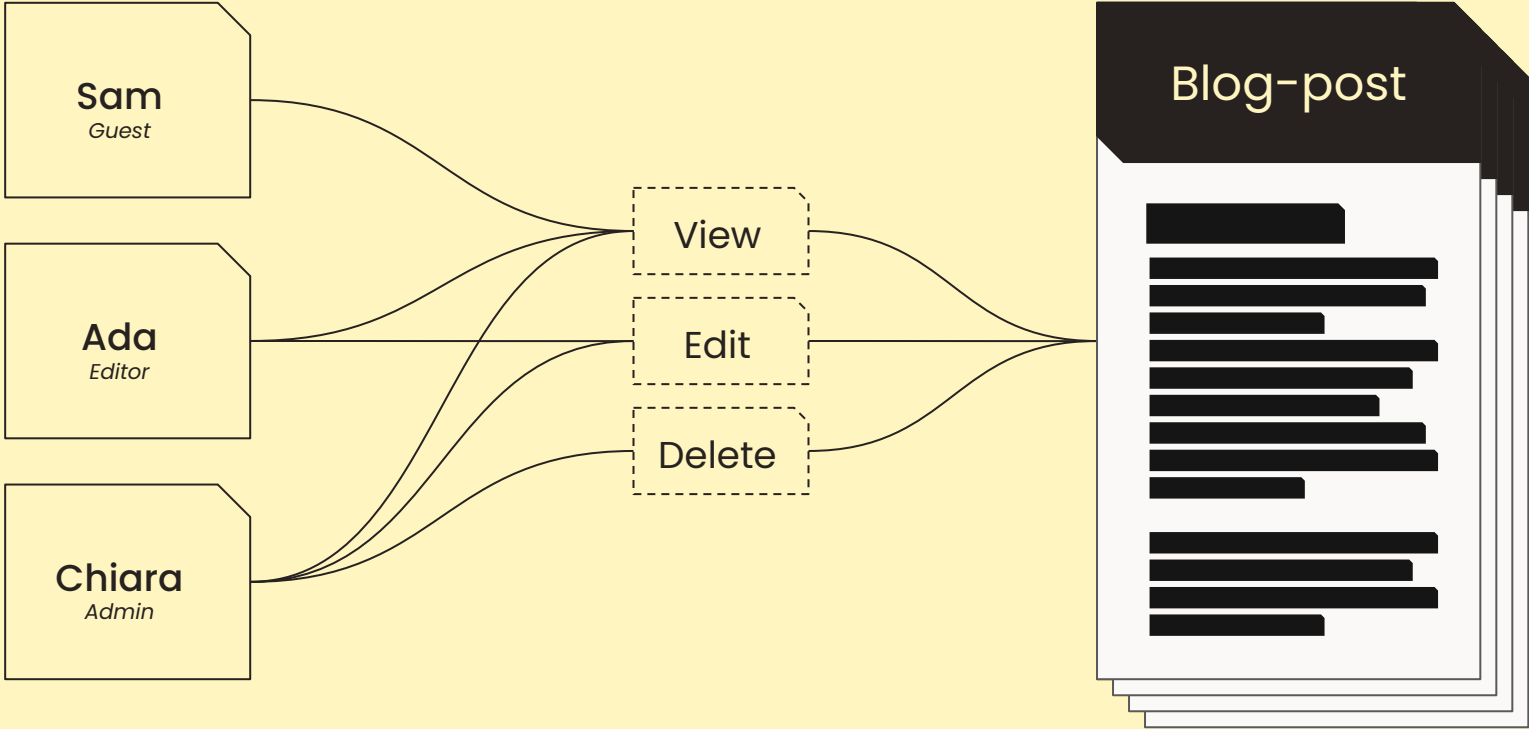


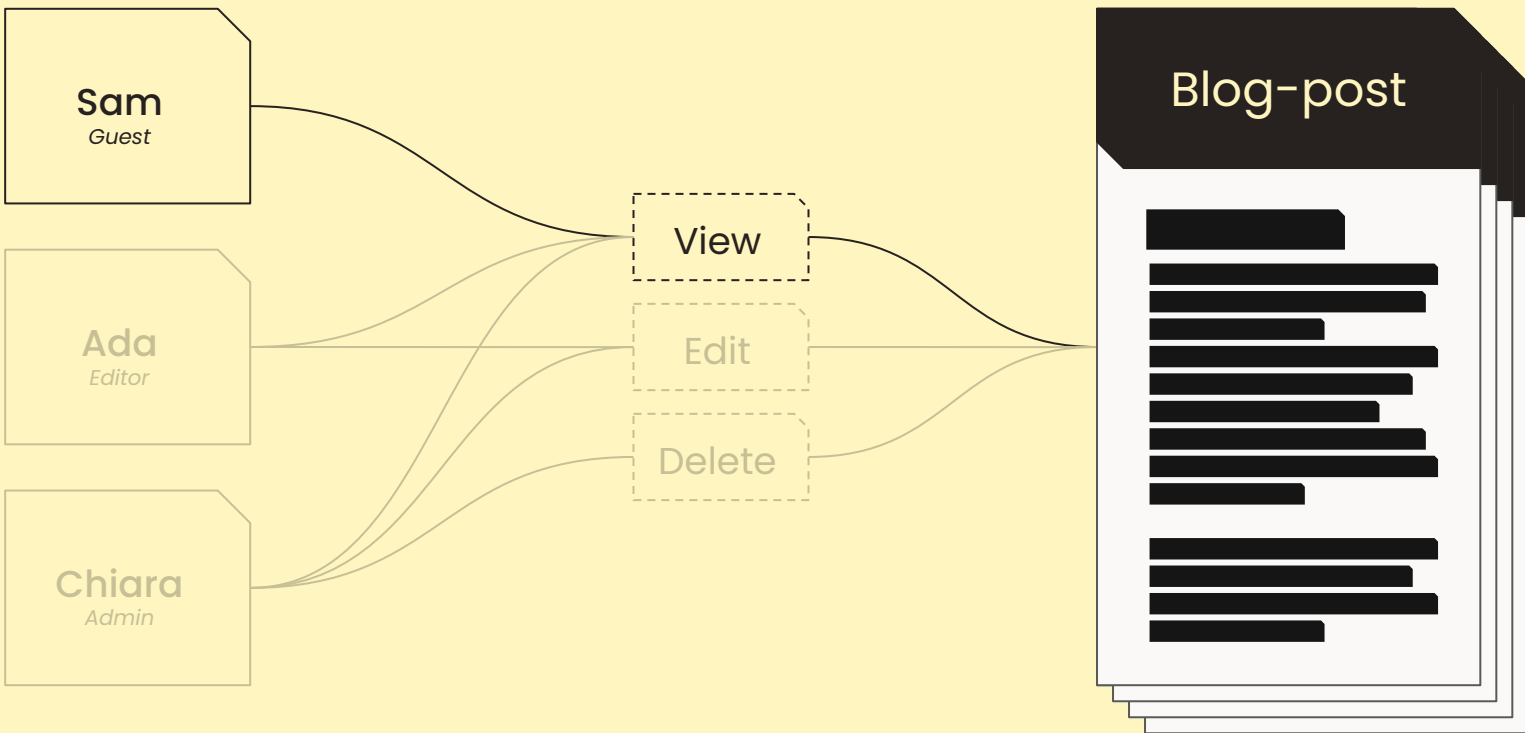
Access Control?

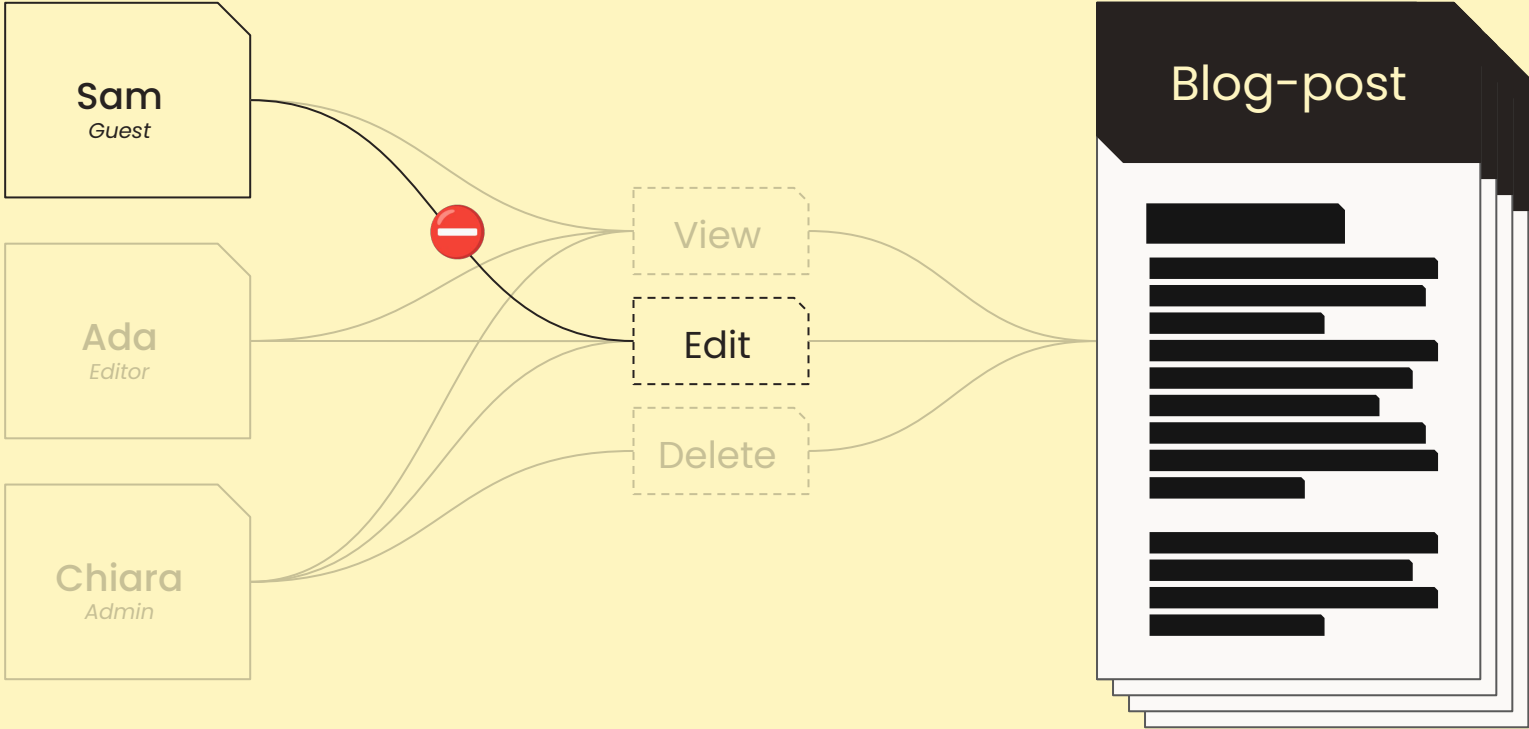
Can *this user* do *that action*?



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





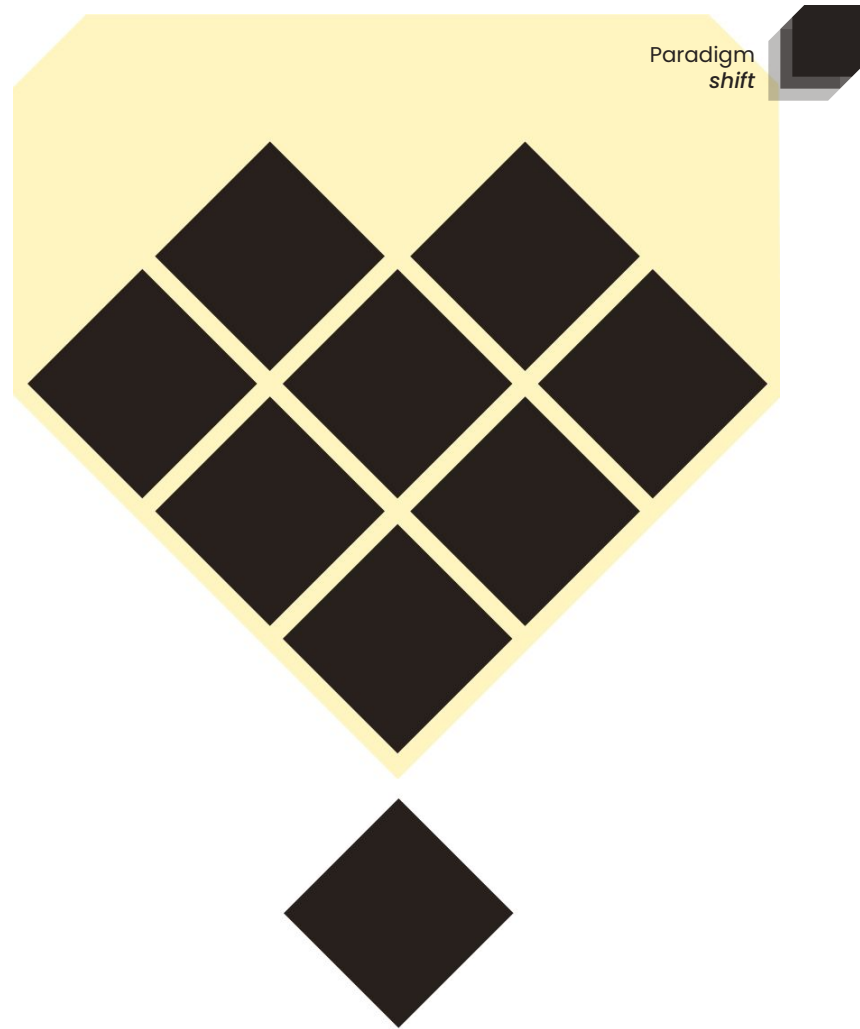




Fine-grained Access control

Take back control!

Coarse
grained





Access control decisions
are made based on
resource type



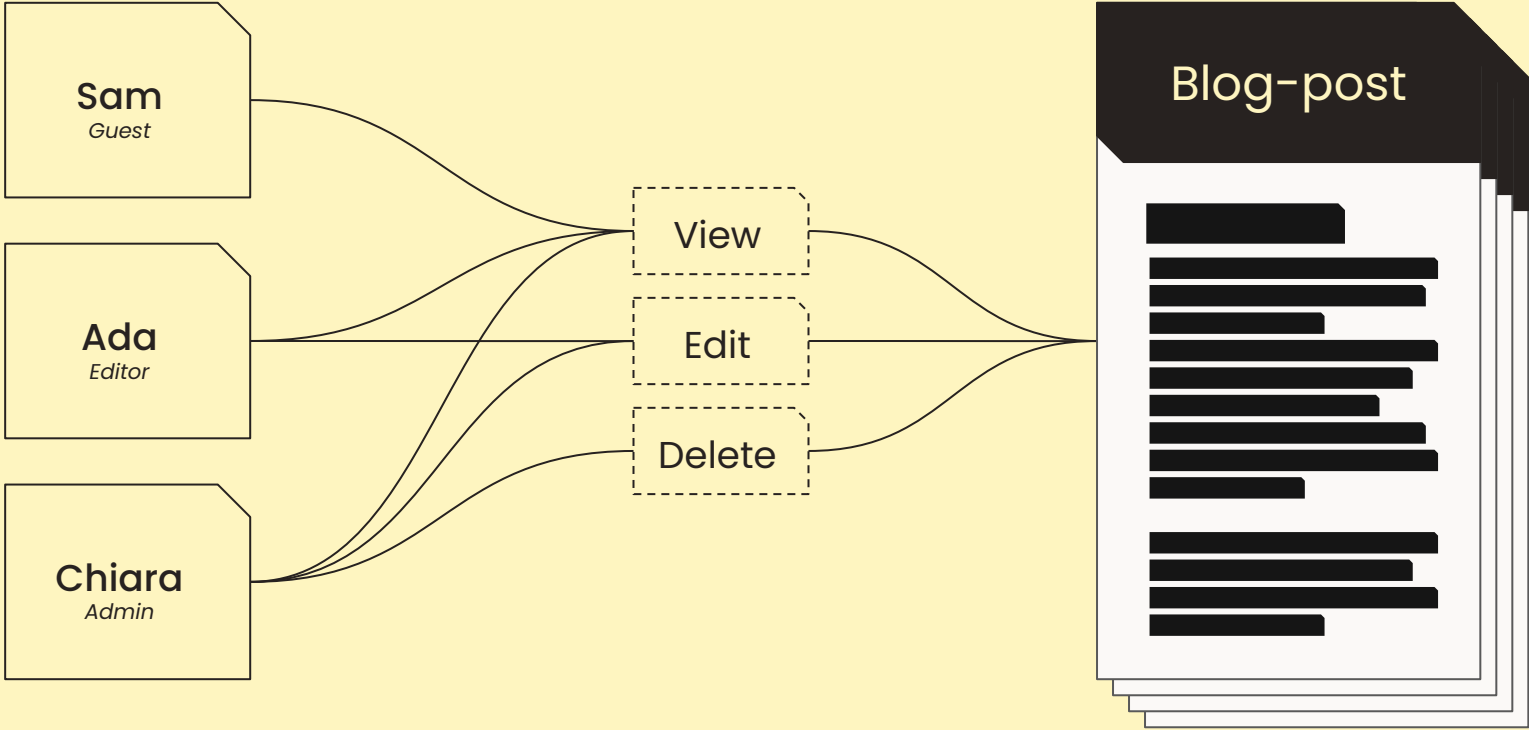
Example

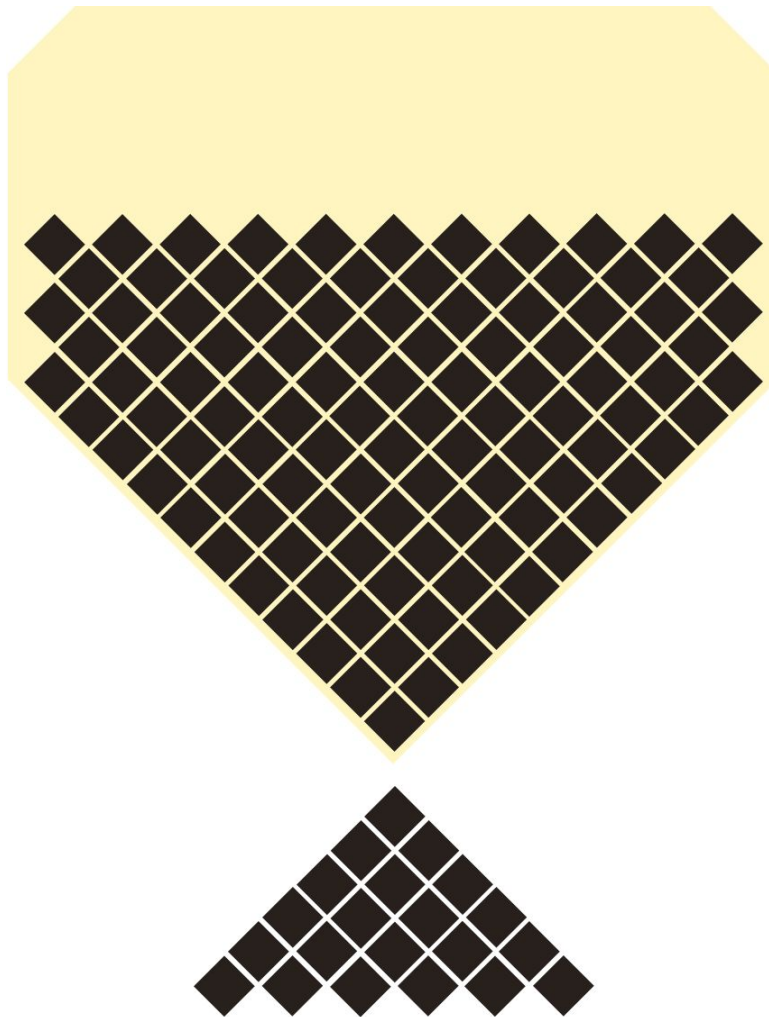
An editor can edit
all blog-posts



Example

An admin can delete
all blog-posts





Fine
grained



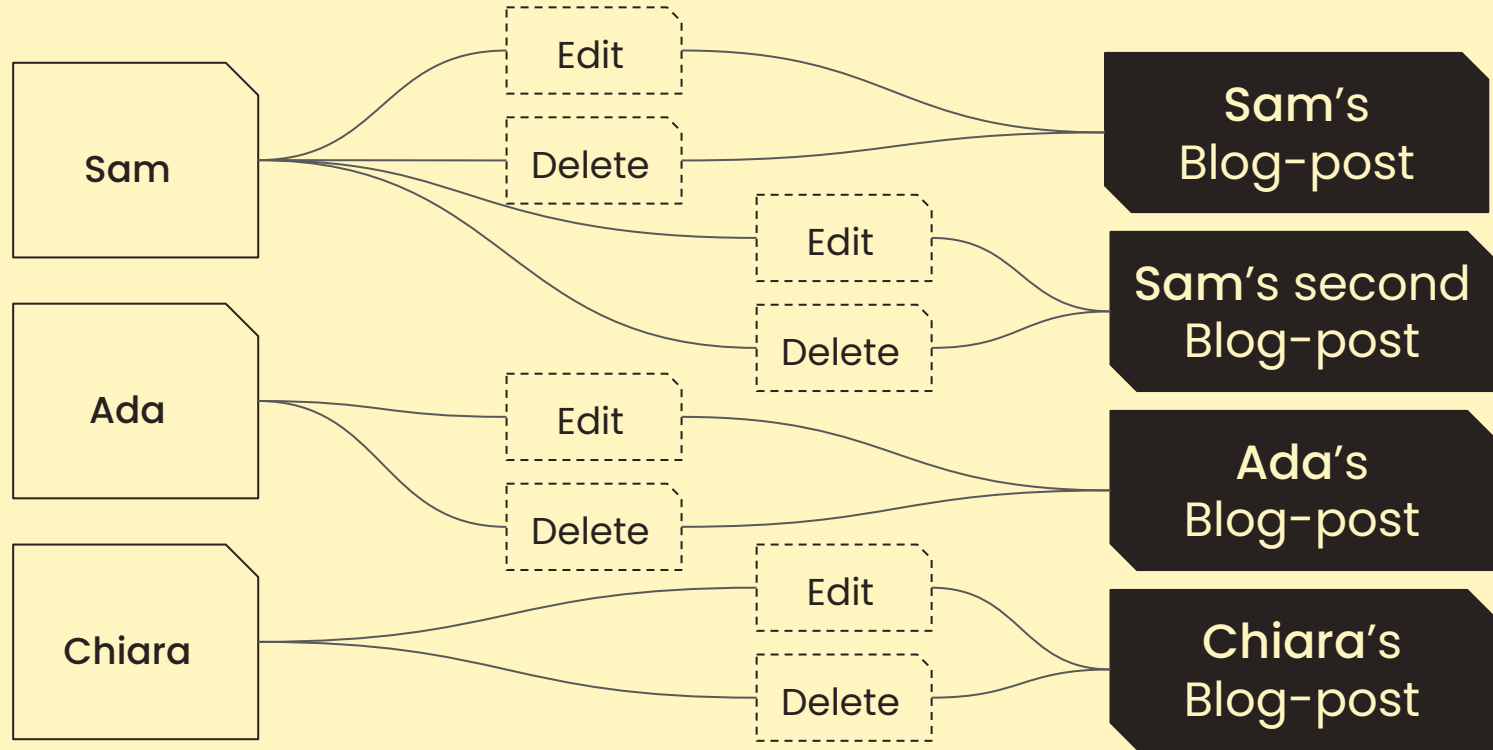
Access control decisions
are made for a specific
resource or situation

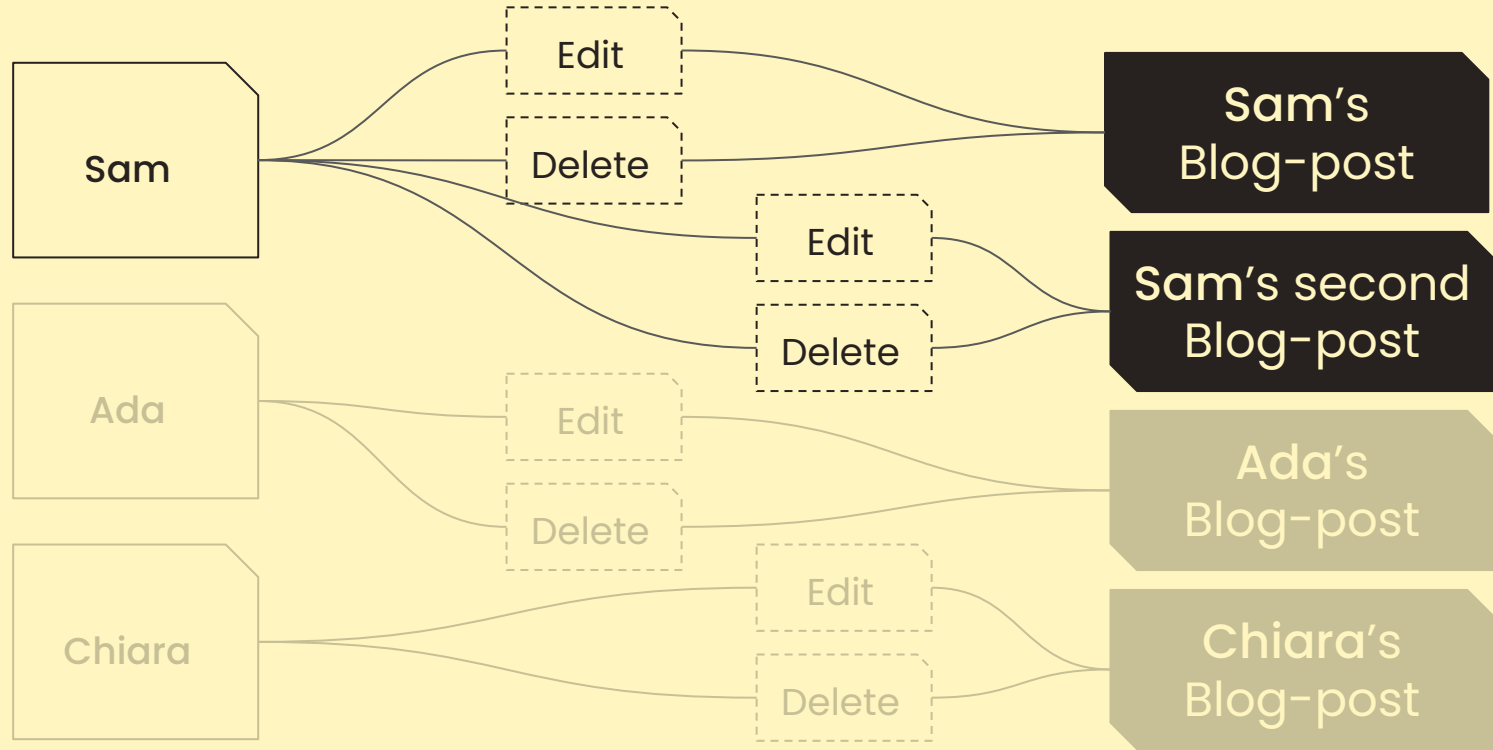


Example

An owner of a post can
delete their own post

Fine-grained



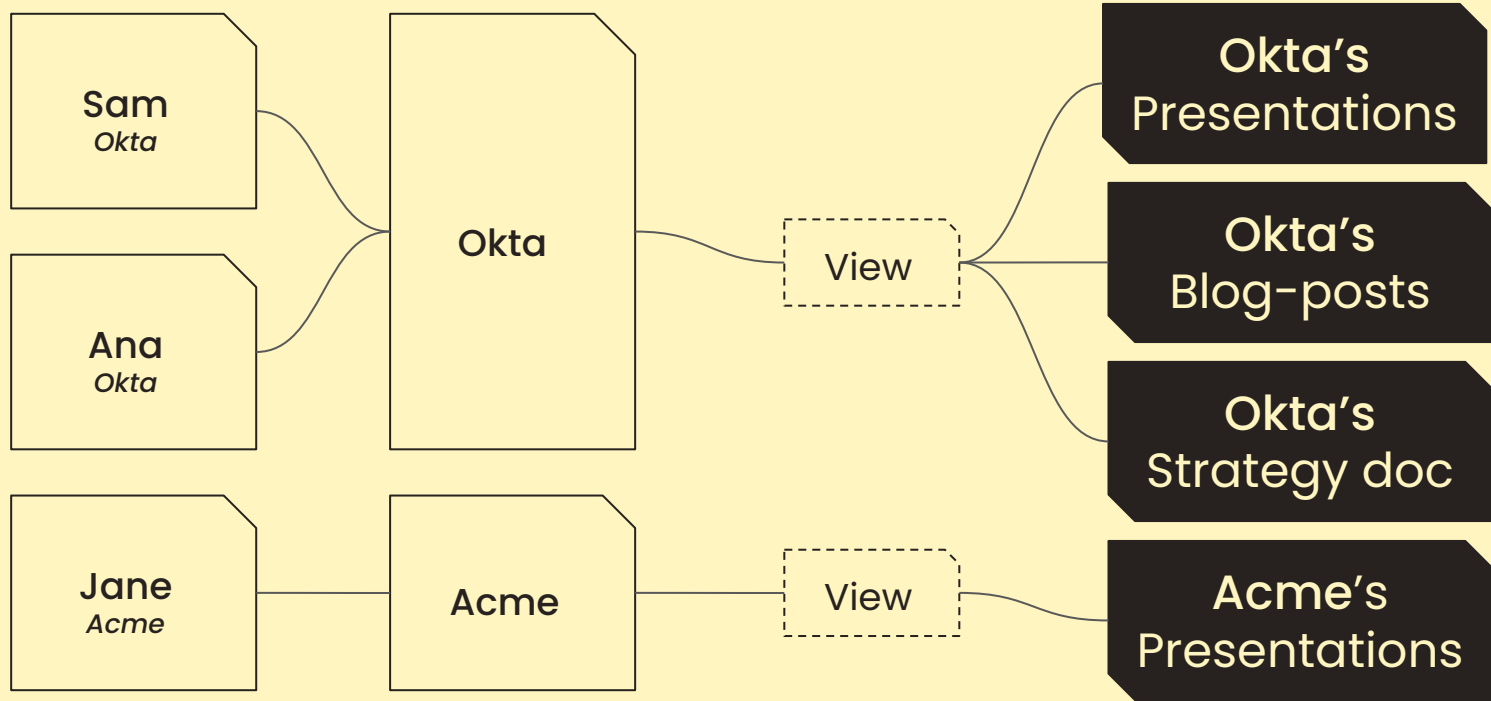


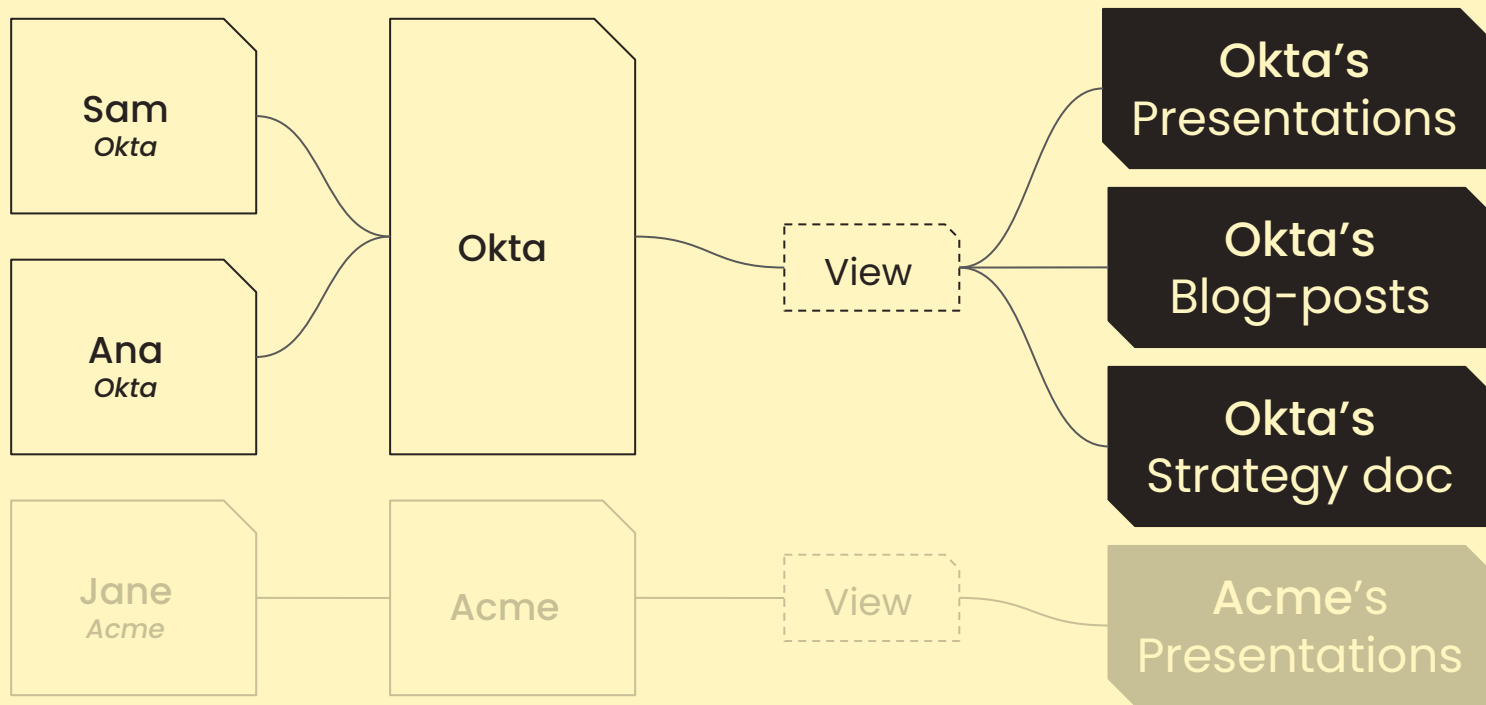


Example

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

Fine-grained



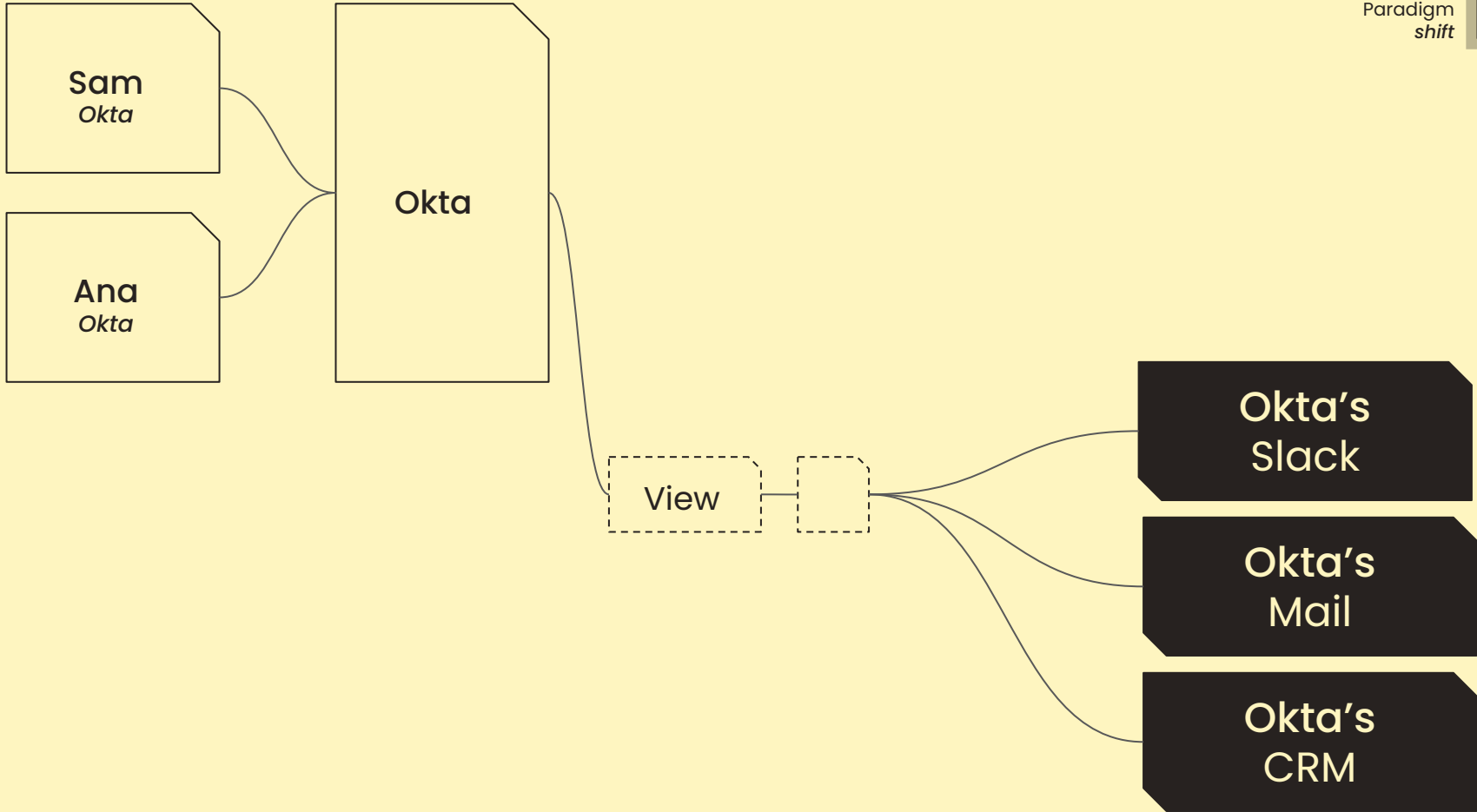


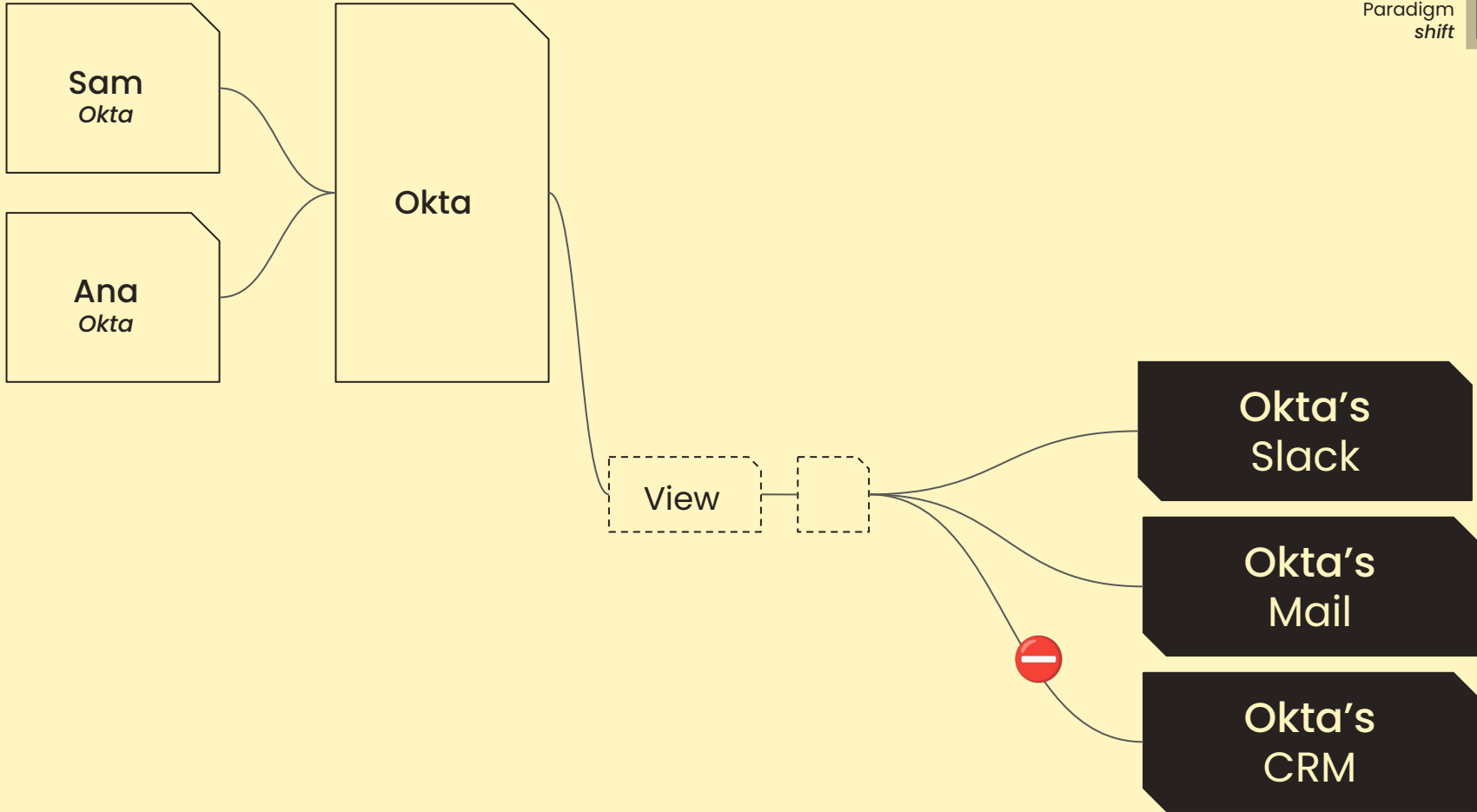


Example

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

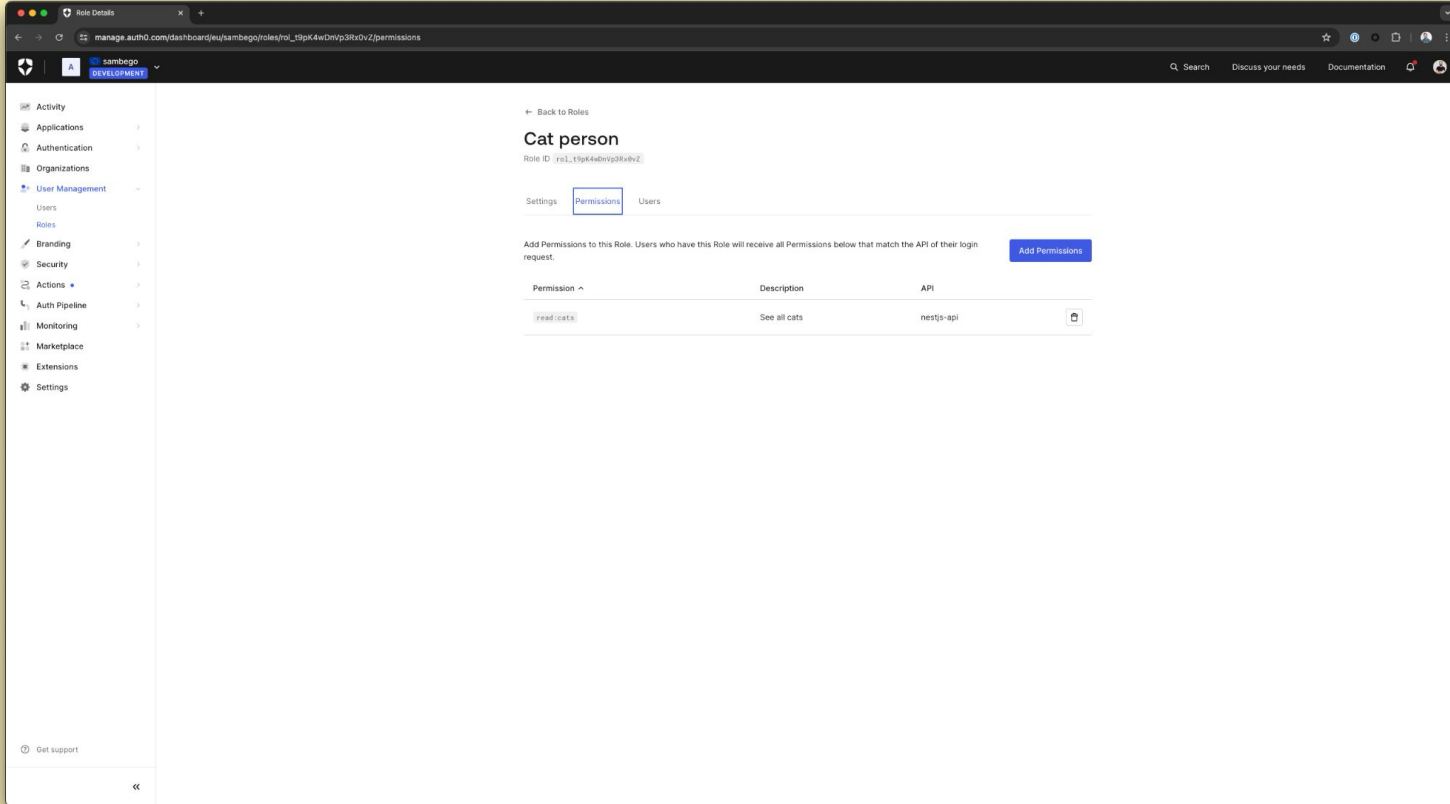
Edit

Delete

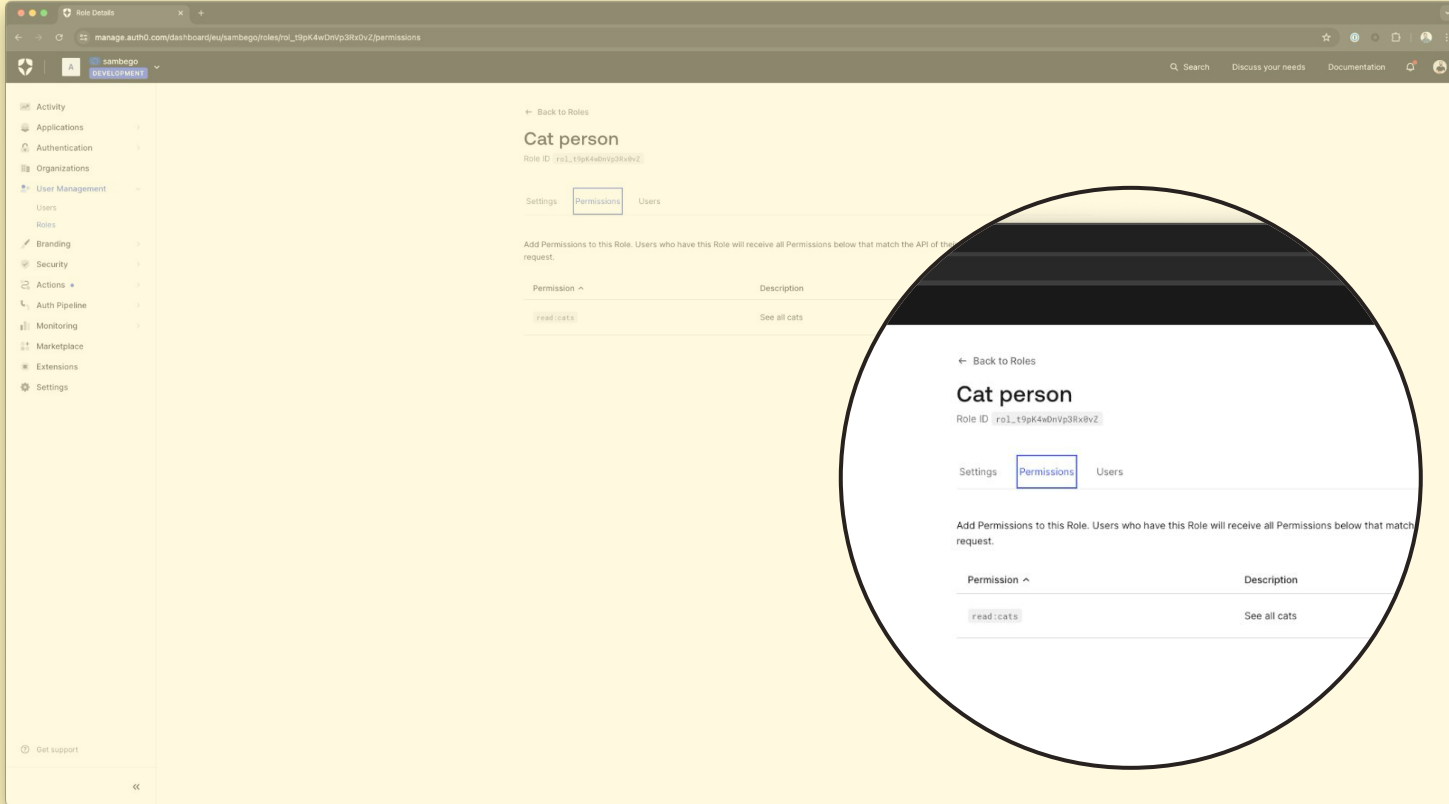
Auth0

Roles and permissions

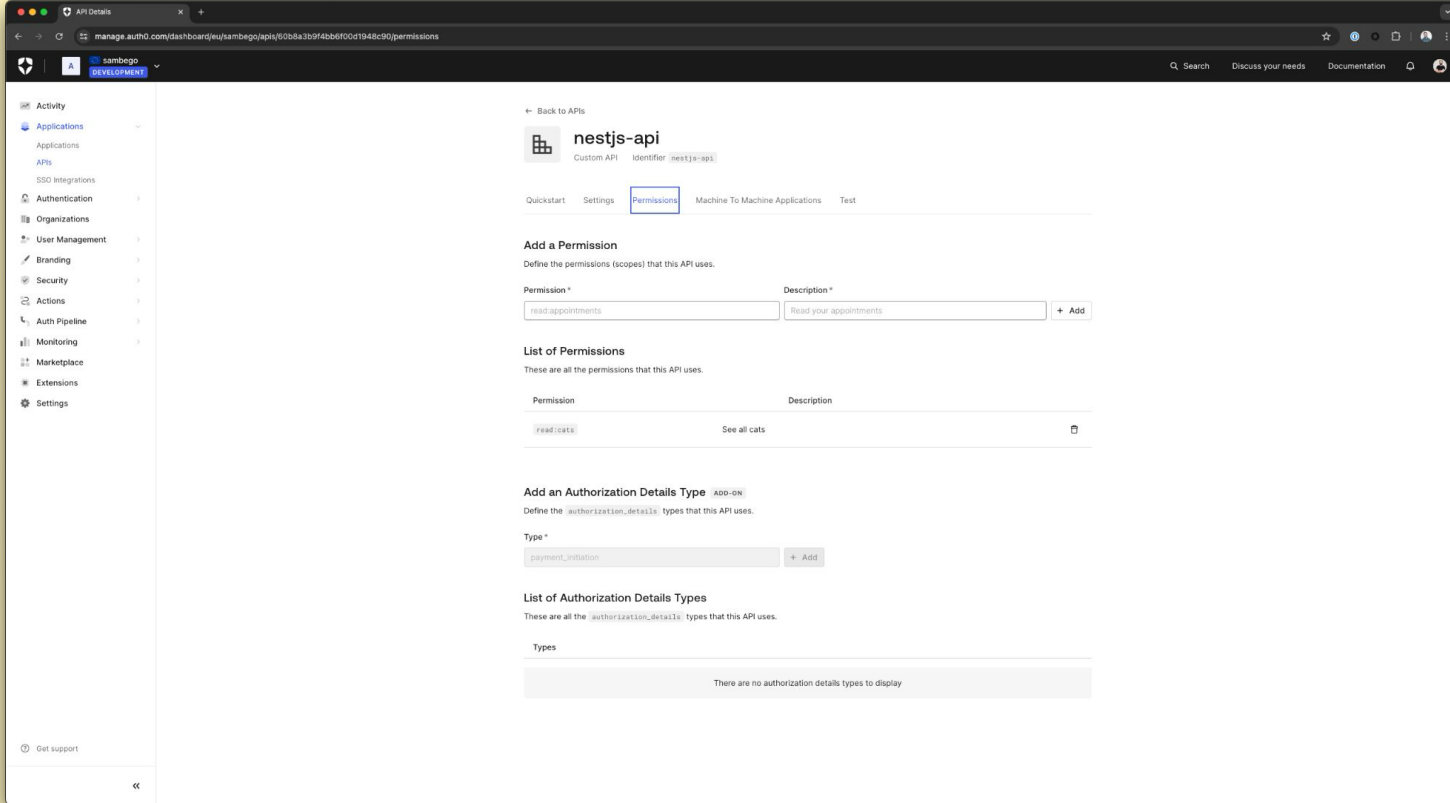
RBAC straight out of the box



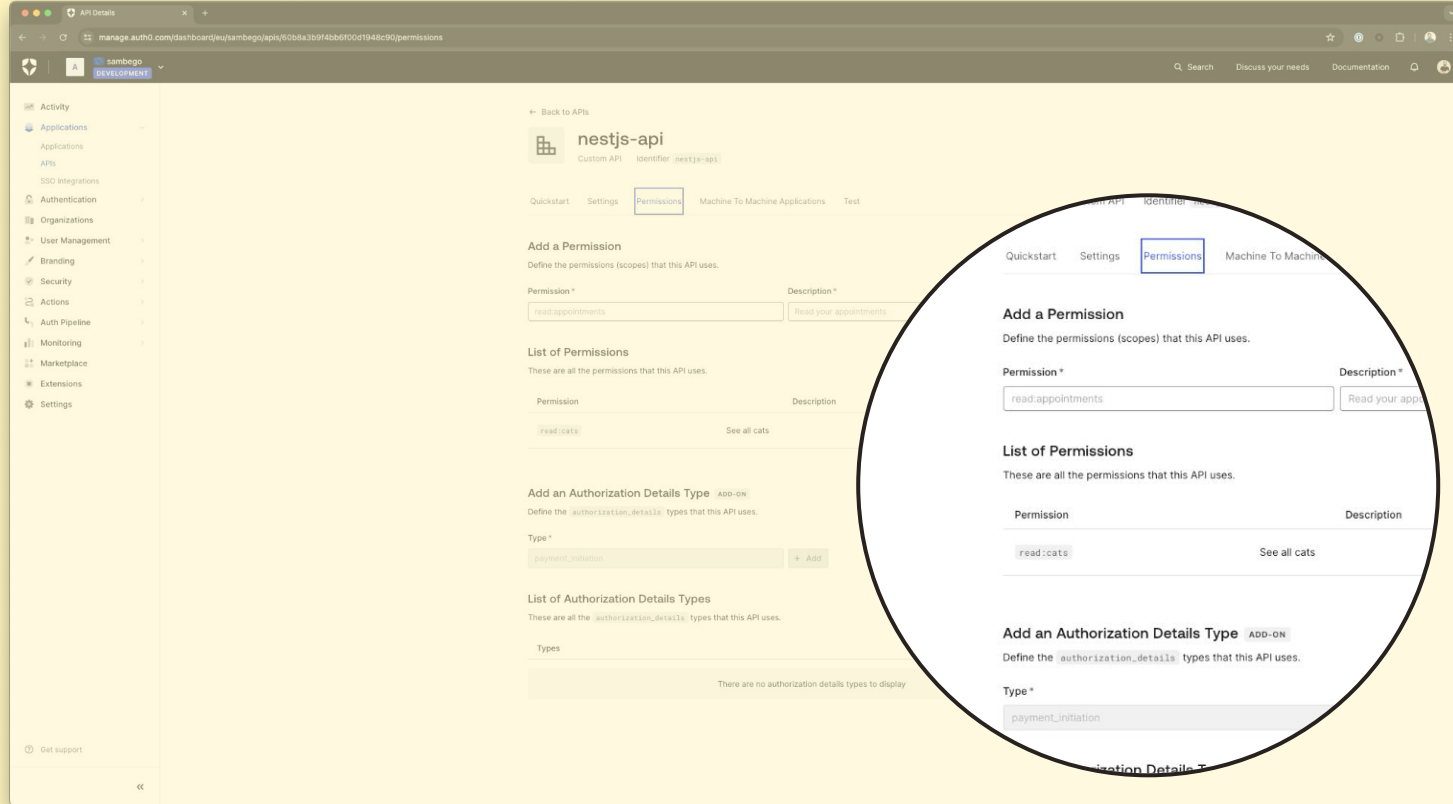
Manage *roles* for your users in the user management section of the dashboard



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



Manage *permissions* for your APIs



Manage *permissions* for your APIs



RBAC Settings

Enable RBAC



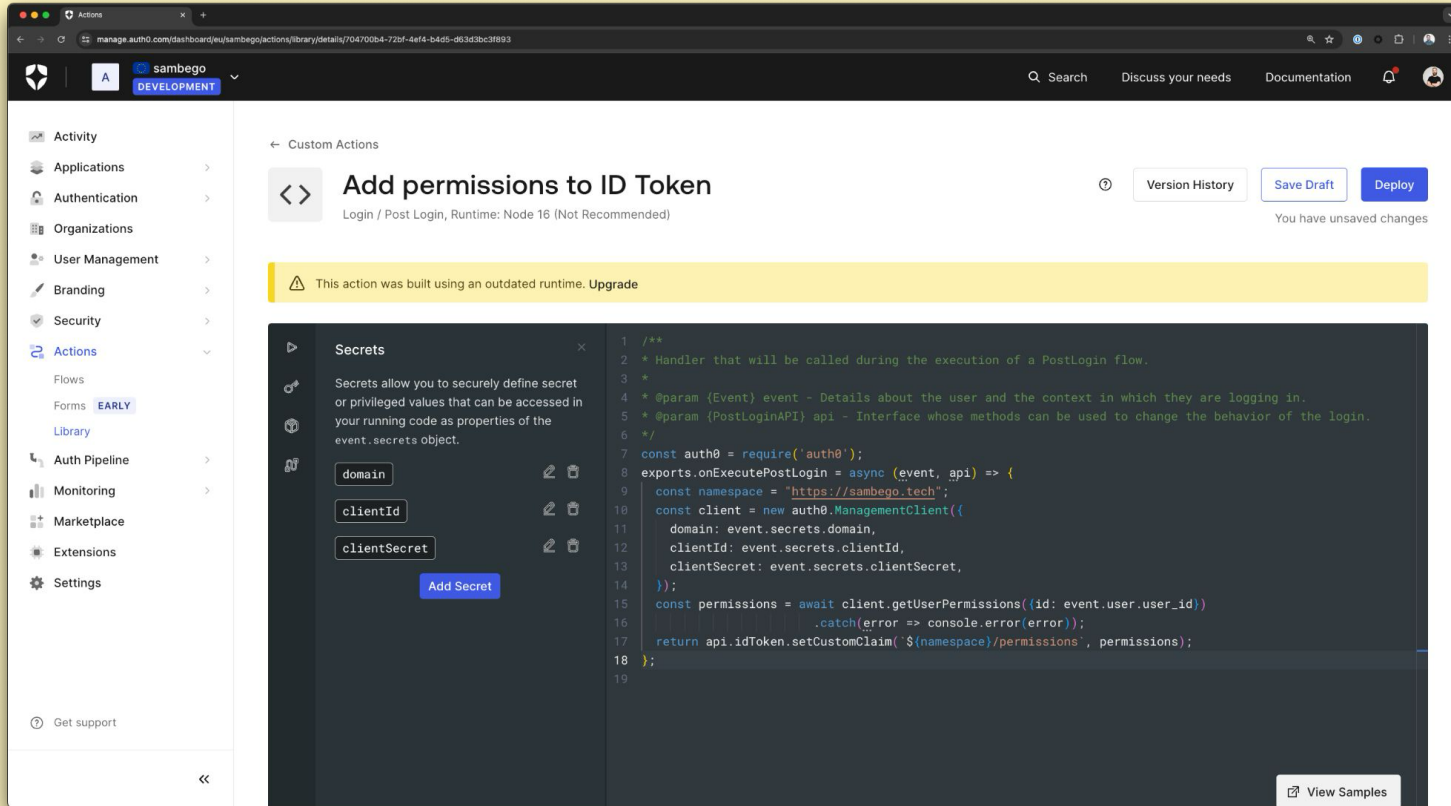
If this setting is enabled, RBAC authorization policies will be enforced for this API. [Role](#) and permission assignments will be evaluated during the login transaction.

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 sidebar with navigation links: Activity, Applications, Authentication, Organizations, User Management, Branding, Security, Actions (selected), Flows, Forms (EARLY), Library, Auth Pipeline, Monitoring, Marketplace, Extensions, and Settings. At the bottom of the sidebar is a 'Get support' link.

The main content area is titled 'Custom Actions' and shows the configuration for an action named 'Add permissions to ID Token'. The action is associated with the 'Login / Post Login' event and uses 'Runtime: Node 16 (Not Recommended)'. There are buttons for 'Version History', 'Save Draft', and 'Deploy'. A message states 'You have unsaved changes'.

A yellow warning banner indicates: 'This action was built using an outdated runtime. Upgrade'.

Below the banner, there is a 'Secrets' section. It explains that secrets allow you to securely define secret or privileged values that can be accessed in your running code as properties of the event.secrets object. It lists three existing secrets: 'domain', 'clientId', and 'clientSecret', each with an edit and delete icon. There is an 'Add Secret' button.

The right side of the configuration shows the JavaScript code for the action:

```
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
10   const namespace = "https://sambego.tech";
11   const client = new auth0.ManagementClient({
12     domain: event.secrets.domain,
13     clientId: event.secrets.clientId,
14     clientSecret: event.secrets.clientSecret,
15   });
16   const permissions = await client.getUserPermissions({id: event.user.user_id})
17     .catch(error => console.error(error));
18   return api.idToken.setCustomClaim(`${namespace}/permissions`, permissions);
19 };
```

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

Add *permissions* to the ID token trough
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

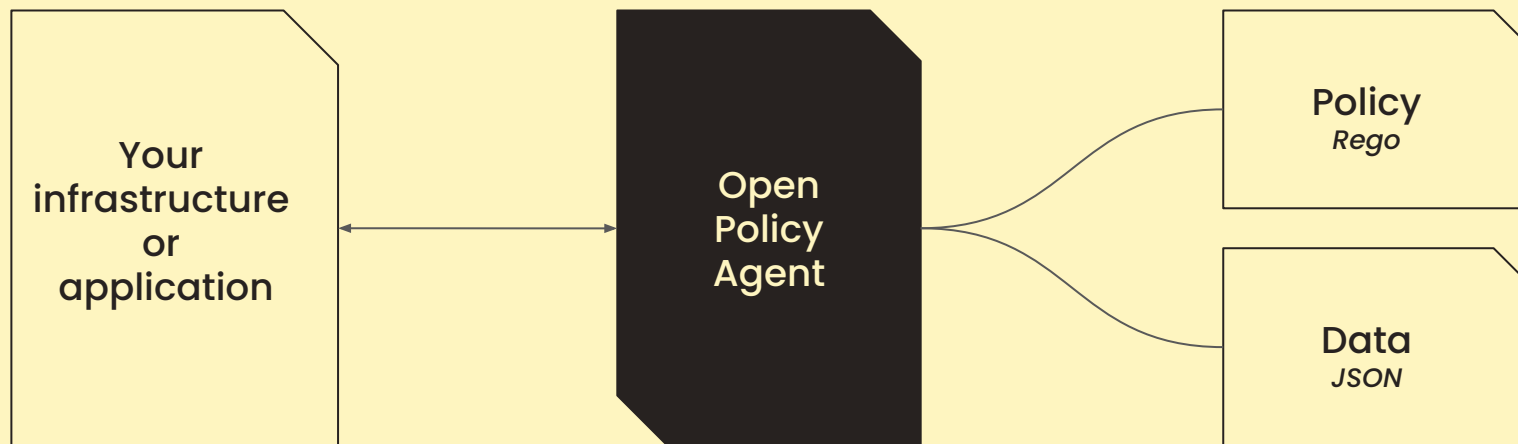
Open-source *policy engine*



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.

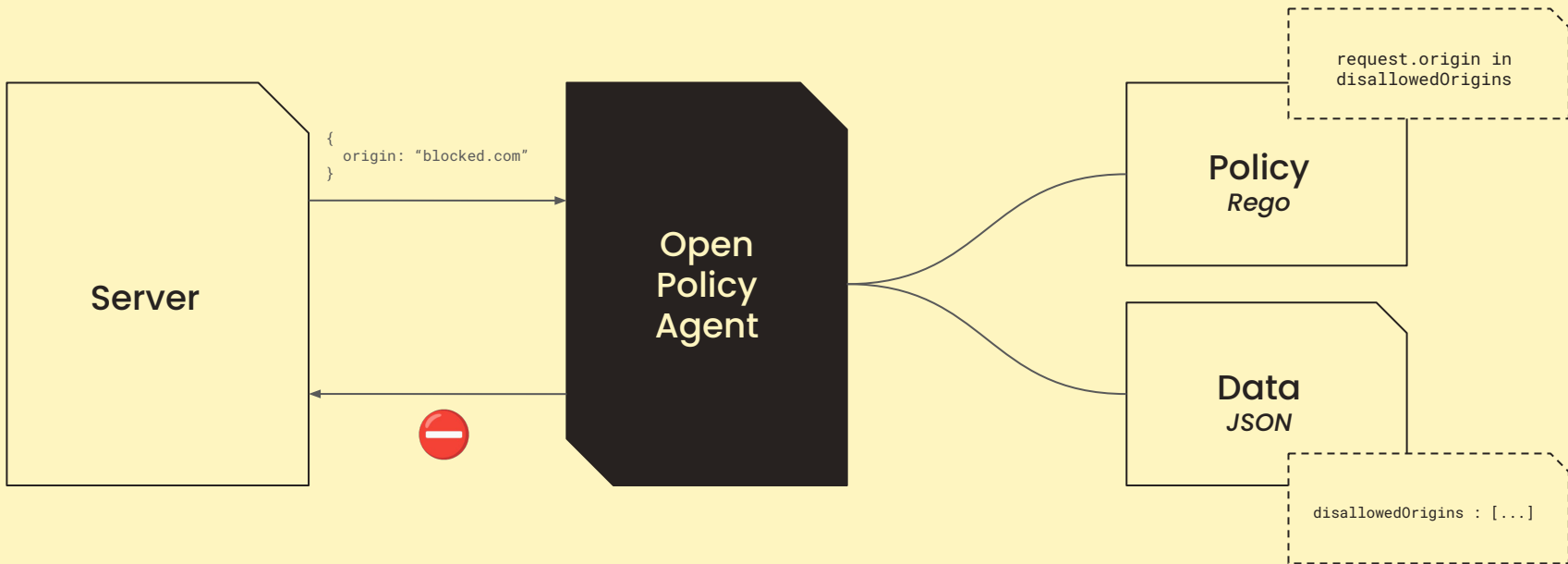


Environment



```
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.

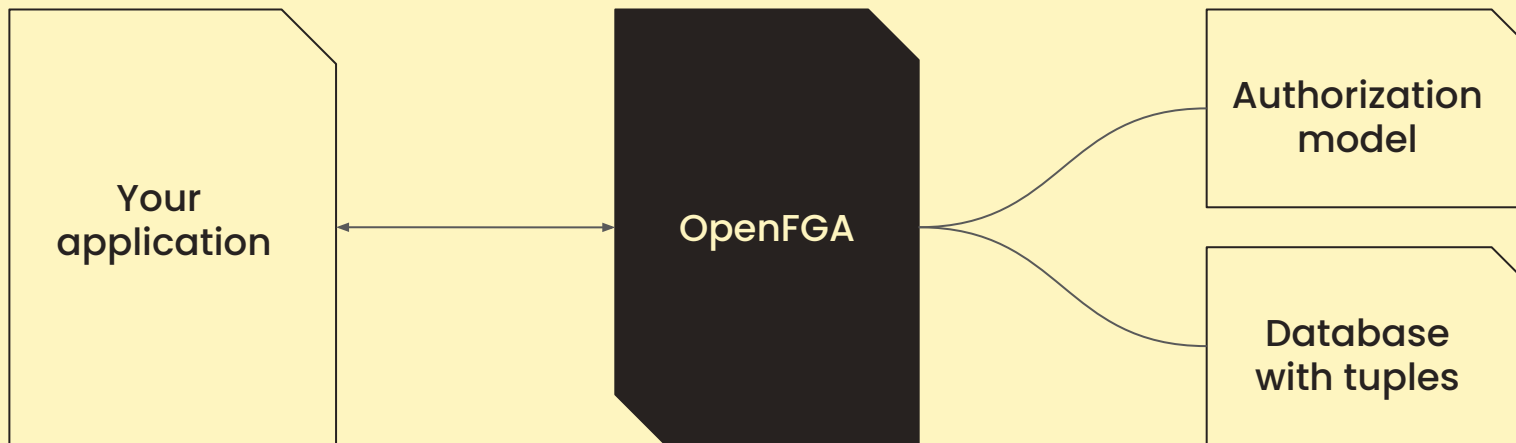
OpenFGA

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

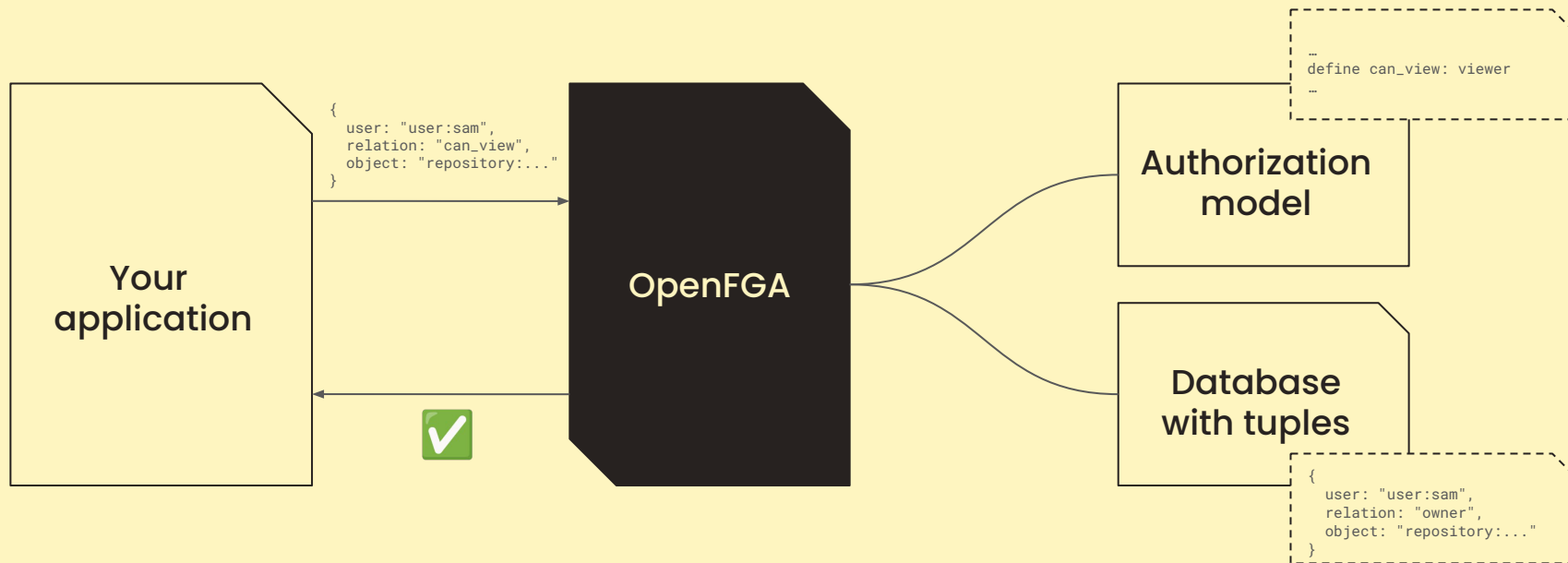
Relation



```
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

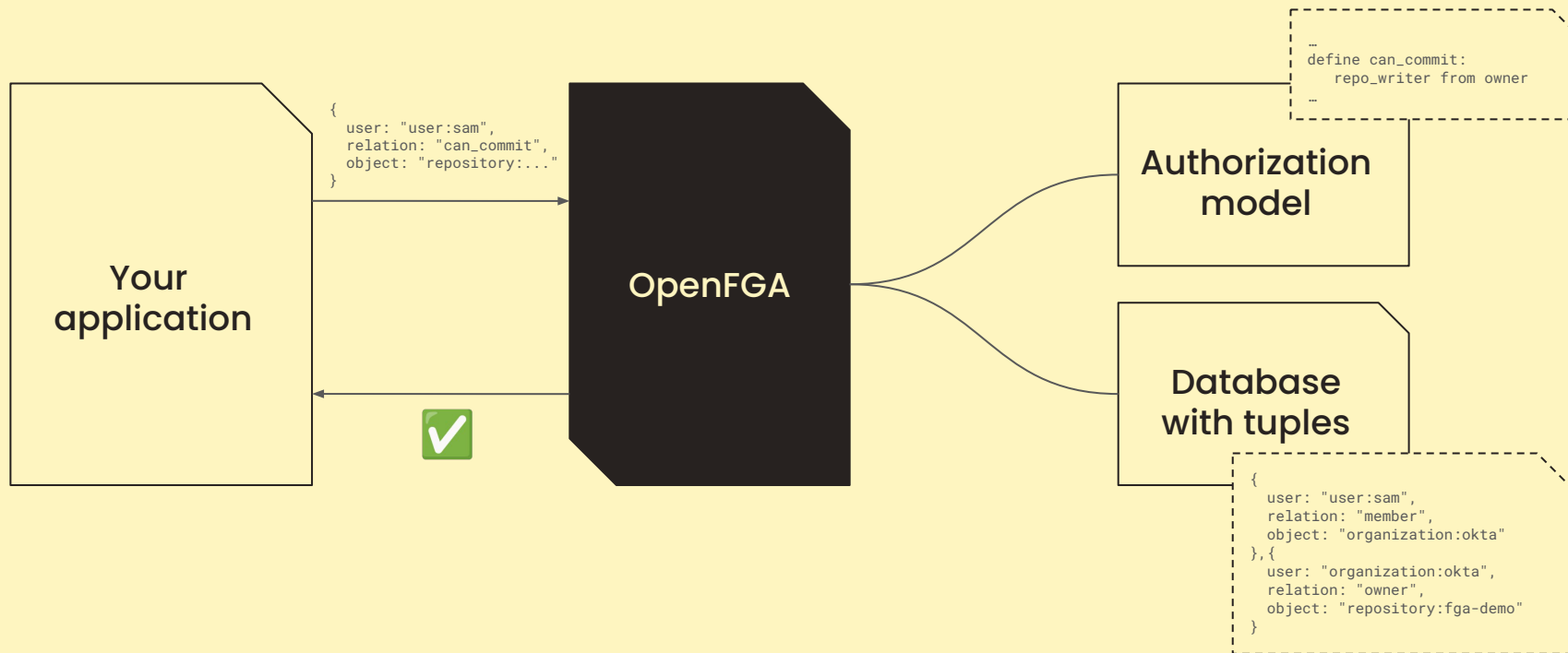
type repository
  relations
    define owner: [organization]
    define can_commit: repo_writer from owner
```

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
```

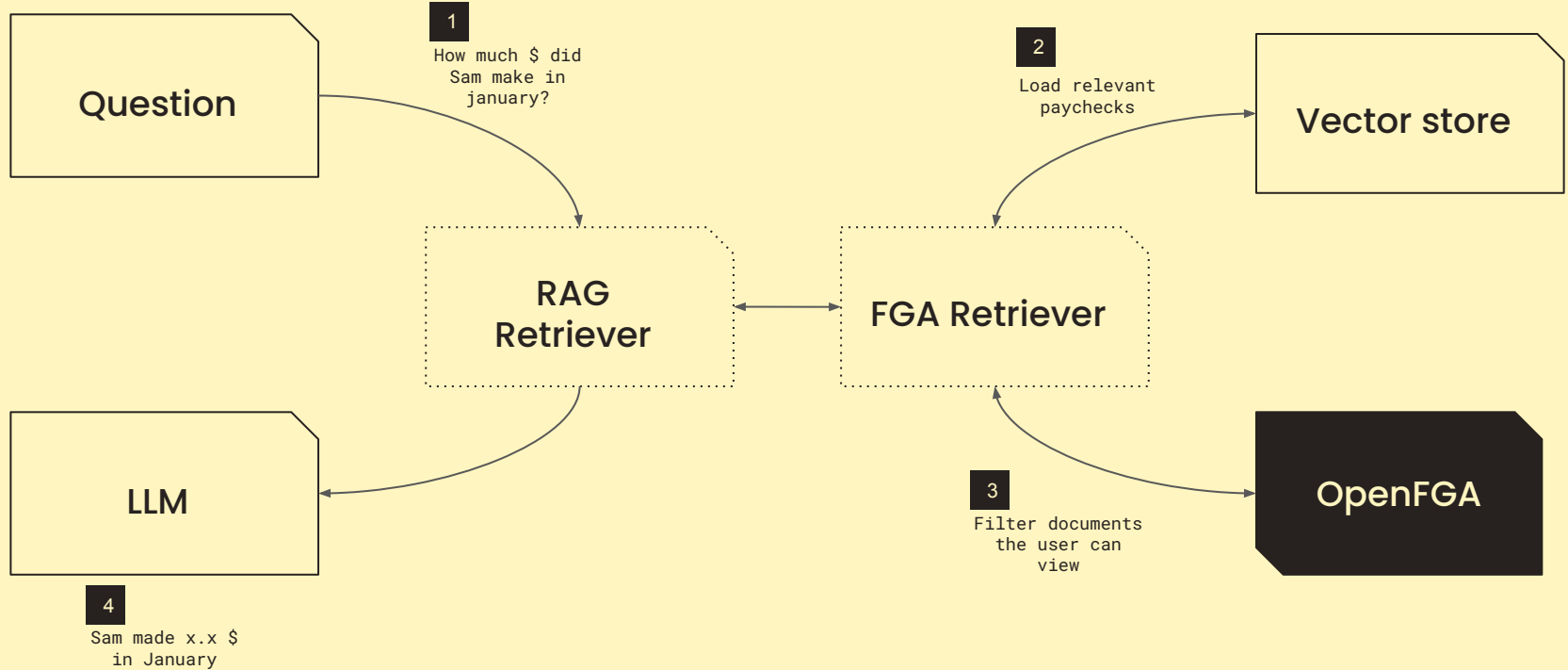
```
    define owner: [user]
```

```
    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*





There's an SDK for that!

a0.to/ato-fga-ai

Okta FGA

A managed OpenFGA service

Model Explorer [FGA]

[dashboard.fga.dev/customers/01FXQD5Y3PRB91EQ6CPSH9QFV/stores/01HR54JVWYF8GVTD14N6WPV2C4/models](#)

Fine Grained Authorization FREE TRIAL

sambego Manage Account Docs

SELECT STORE

drive

Getting Started

Model Explorer

Tuple Management

Developer Mode

Settings

Model Explorer

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

Model

Save Preview

```

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 owners: [user]
15    define parents: [folder]
16    define viewers: [user, user=]
17
18   type folder
19   relations
20     define can_create_files: owner or owner from parent
21     define can_create_folders: 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 owners: [user]
25     define parents: [folder]
26     define viewers: [user, user=] or owner or viewer from parent
27

```

Latest ID 01HST538X9QKEYRW7QKT056J24

store

type

relation

Zoom In

Zoom Out

Reset

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

Get support

Developer Mode | FGA

dashboard.fga.dev/customers/01FXQD5Y3PRB91EQG6CPSH9GFV/stores/01HRS4JVWYF8GVTD14N6WPVZC4/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](#)

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 owners: [user]
15    define parents: [folder]
16    define viewers: [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 owners: [user]
25     define parents: [folder]
26     define viewers: [user, user+] or owner or viewer from parent
27
```

Latest ID: 01HSTS3BX9QXEYRW7OKT0S6JZ4

Tuples

Type to filter tuples...

+ Add Tuple

USER	user:auth0 65faef64e18ff2c62b63cd9a	
OBJECT	folder:86f2bb93-aa8c-48a8-9528-d99682...	
RELATION	viewer	
USER	user:auth0 65faef64e18ff2c62b63cd9a	
OBJECT	file:49304d83-13bc-483c-a4fe-8668f72db...	
RELATION	viewer	
USER	user:google-oauth2 1135799221734285377...	
OBJECT	file:62f35382-c928-464d-a132-57953752...	
RELATION	owner	
USER	folder:6fb2e898-c5bc-47db-a821-140c210...	
OBJECT	file:62f35382-c928-464d-a132-57953752...	
RELATION	parent	
USER	user:google-oauth2 1135799221734285377...	
OBJECT	file:a01f61dd-59fc-4332-85db-9bcfb9f73b...	
RELATION	owner	
USER	folder:6fb2e898-c5bc-47db-a821-140c210...	
OBJECT	file:a01f61dd-59fc-4332-85db-9bcfb9f73b...	
RELATION	parent	

Assertions

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

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 fourth
(and final?) draft at the
OpenID Foundation.



```
{
  "subject": {
    "type": "user",
    "id": "sam",
  },
  "action": {
    "name": "can_view",
  },
  "resource": {
    "type": "document",
    "id": "paycheck-jan-2025-sam",
  },
  "context": {
    ...
  }
}
```

AuthZEN evaluation payload

`pdp.example.com/access/evaluation`



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

AuthZEN evaluation response

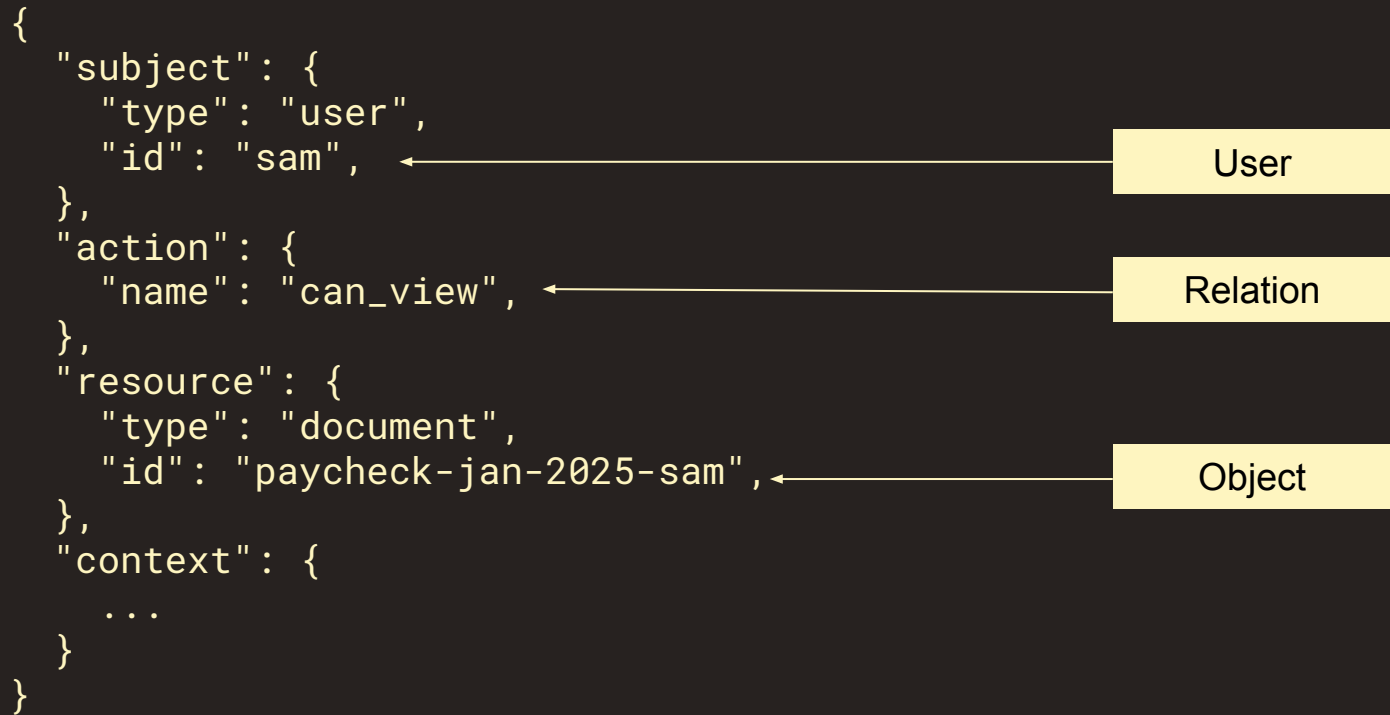
`pdp.example.com/access/evaluation`



```
{
  "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

/evaluation



AuthZEN evaluation payload for
OpenFGA

```
{
  "subject": {
    "type": "user",
    "id": "sam",
  },
  "action": {
    "name": "can_view",
  },
  "resource": {
    "type": "document",
    "id": "paycheck-jan-2025-sam",
  },
  "context": {
    ...
  }
}
```

User attribute

Resource attribute

AuthZEN evaluation payload for OPA



```
{
  "subject": {
    "type": "user",
  },
  "action": {
    "name": "can_view",
  },
  "resource": {
    "type": "document",
    "id": "paycheck-jan-2025-sam",
  },
  "context": {
    ...
  }
}
```

AuthZEN subject search payload

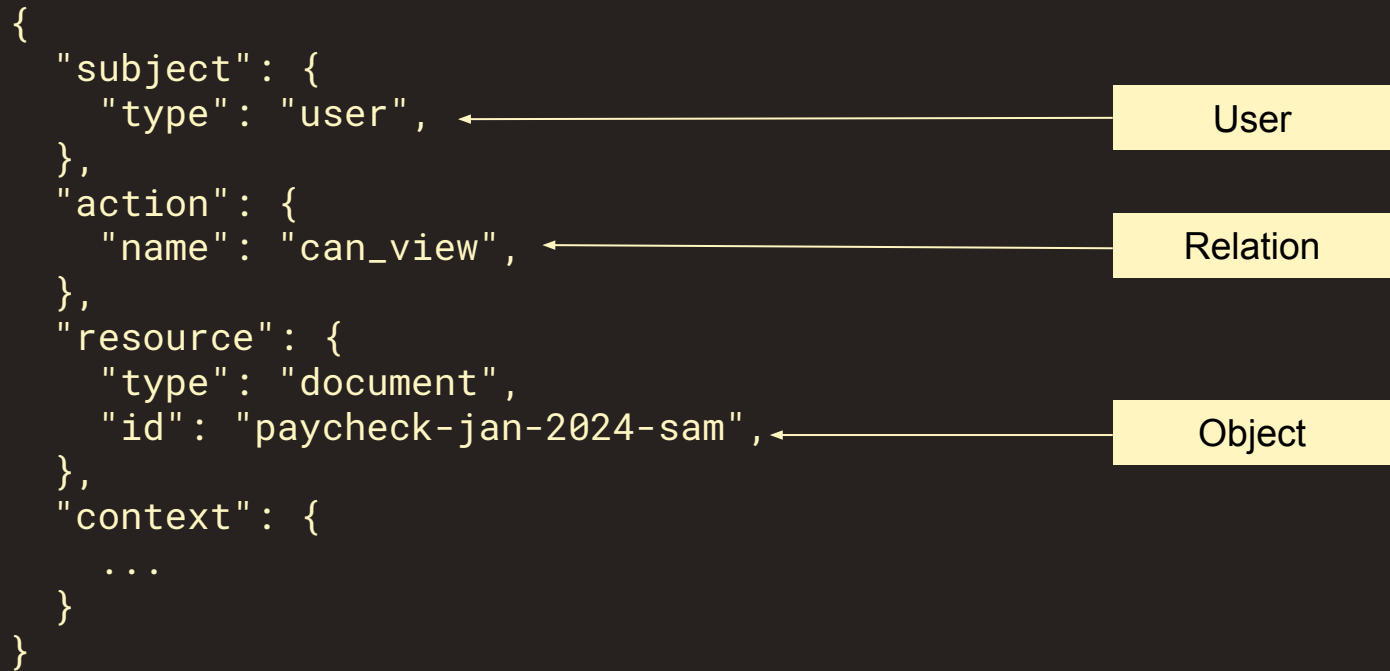
pdp.example.com/access/subject



```
{
  "results": [
    {
      "type": "user",
      "id": "sam"
    },
    {
      "type": "user",
      "id": "jane"
    }
  ]
}
```

AuthZEN subject search response

`pdp.example.com/access/subject`



AuthZEN subject search payload for
OpenFGA



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

The diagram illustrates the mapping of attributes from the JSON payload to external boxes. A yellow box labeled "User attribute" has an arrow pointing to the "type" field of the "subject" object. A larger yellow box labeled "Resource attribute" has two arrows: one pointing to the "name" field of the "action" object, and another pointing to the "id" field of the "resource" object.

AuthZEN subject search payload for
OPA



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

AuthZEN resource search payload

`pdp.example.com/access/resource`



```
{
  "results": [
    {
      "type": "document",
      "id": "paycheck-jan-2025-sam"
    },
    {
      "type": "document",
      "id": "paycheck-feb-2025-sam"
    }
  ]
}
```

AuthZEN resource search response

`pdp.example.com/access/resource`



```
{
  "subject": {
    "type": "user",
    "id": "sam"
  },
  "resource": {
    "type": "document",
    "id": "paycheck-jan-2025-sam",
  },
  "context": {
    ...
  }
}
```

AuthZEN action search payload

pdp.example.com/access/action



```
{  
  "results": [  
    {  
      "name": "can_view"  
    },  
  ]  
}
```

AuthZEN action search response

`pdp.example.com/access/action`



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.



AuthZEN is an **open standard** to provide a common way to relay access control decisions, **regardless of strategy** or decision engine



Sam Bellen

Principal developer advocate at Auth0

@sambego
sambego.tech



Learn more

a0.to/ato-fga

@sambego
sambego.tech

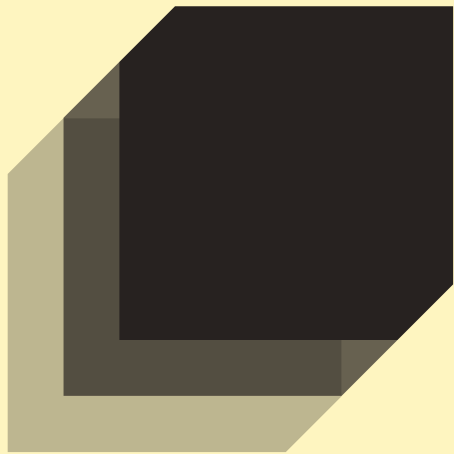


Find these slides

slides.sambego.tech/paradigm-shift

@sambego
sambego.tech

Thank *you*!



Paradigm *shift*

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