Roles & Permissions

Erik Hetzner & Zach Dennis

The goal of this slide deck is to introduce you to how the updated Roles & Permissions will work.

We hope it provides just enough information to provide basic understanding of R&P and how its pieces fit together.

Disclaimer: It will provide realistic-ish examples, but they are meant to showcase R&P, not to be representative of what production R&P data will look like.

Overarching Goal of R&P

- Support authorization on both the backend and the front-end of Aperta in a consistent, generic, and maintainable manner.

Backend Goal

- Allow the backend to be the authority on access to behavior and information
- E.g. don’t let a user create a task on a paper they can’t access
- E.g. don’t send information a user can’t see to the client (browser).

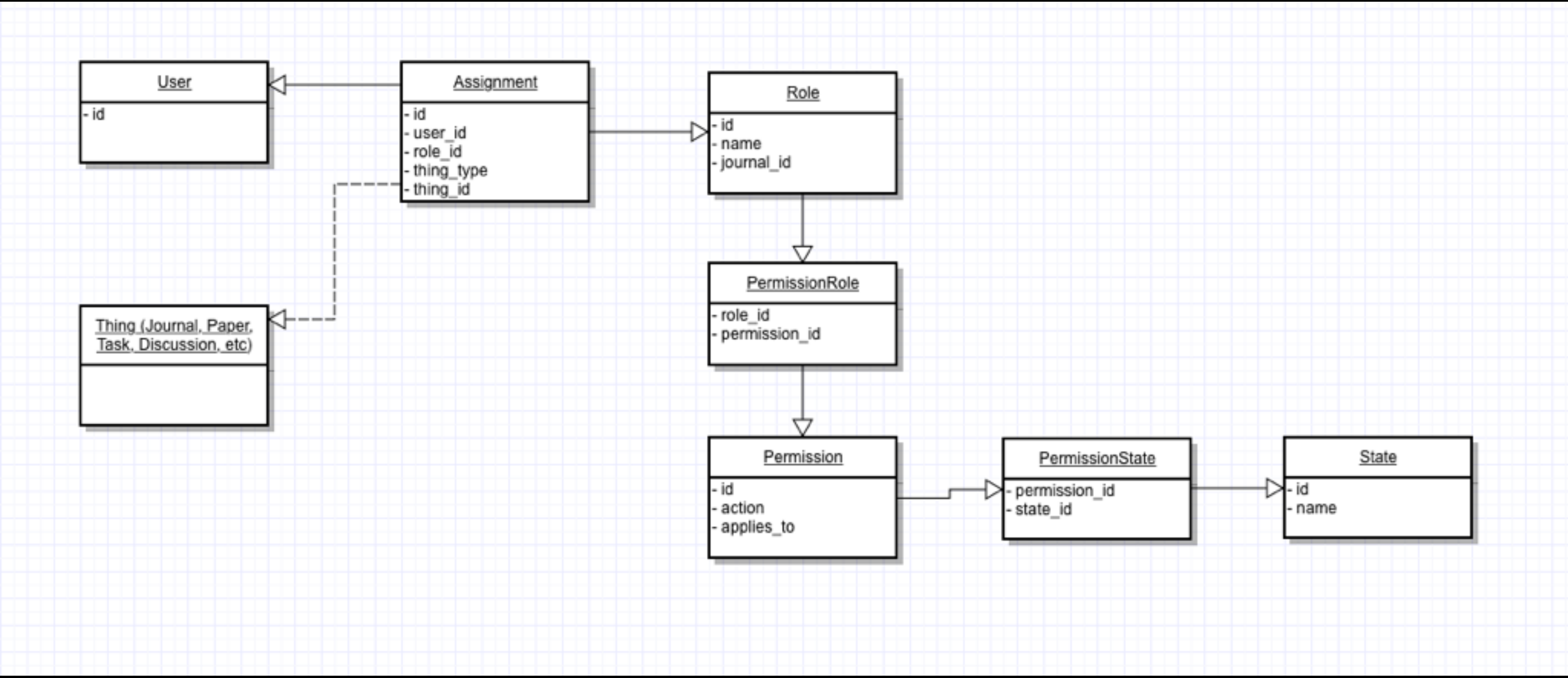
Front-end Goal

- Present contextually relevant information and functionality to the user
- E.g. Don’t render a section of content the user can’t see
- E.g. Don’t show a user a button they can’t click.

Data Model @ 40,000 ft

- Role
- Permission
- State
- Assignment

Data Model Diagram



Role

- A **Role** is a name that represents a user’s part in the publishing process
- A user may have multiple roles as they may play multiple parts
- A role can be used to indicate at what a user is interested in as opposed to what they have access to.
    - E.g. PLOS Staff may have access to all papers on a journal, but they likely don’t want to see all papers on their dashboard

Permission

- A **Permission** is the most granular unit of access that Aperta knows how to enforce
- It’s a combination of a *action* describing what the permission is and what it *applies to*
- Aperta enforces permissions, not roles

State

- A **State** is a condition that must be met for a permission to be applicable
- There is a default state that represents that a permission is always applicable
- E.g. A reviewer cannot review a paper unless it is in the state of “in review”.
- This could be applied to things beyond paper states (e.g. task states), but we aren’t building that in, just noting that is a direction where this could be extended if it becomes necessary

Assignment

- An **Assignment** ties together a **user**, a **role**, and a **thing***
    - *thing* is polymorphic and can represent any entity in the system (e.g. Journal, Paper, Task, Discussion, etc)
- It is intended the single source of truth for how a user obtains any kind of access in Aperta

**thing* is subject to change as it is likely not the best name.

How Users Get Access To Things

@
20,000 ft.

The Basics

- A user gets access by being assigned to **something** (like a journal, a paper, a task, a discussion, etc) with a specific **role.**
- Since a role is largely a collection of permissions with a human-friendly name this is how a user gets permissions to things

Examples

- a user is assigned to the **PLOS BIO journal** as an **Internal Editor**
- a user is assigned to a **paper** as the **Author** when they create it
- a user is assigned to their **ReviewerReportTask** as a **Reviewer** when they accept the invitation to review

Authorization

@

10,000 ft.

Keep in mind

- An assignment says **how** you got access
- A permission says **what** you can access

Examples

To keep this simple we’re going to only use one permission: ***view***.

Assignments		Roles		Permissions
Lucy **is assigned to** PLOS BIO journal	→	Internal Editor	→	**view** which applies to **Journal** **view** which applies to **Paper** **view** which applies to **Task**
Bob **is assigned to** Some Paper	→	Author	→	**view** which applies to **Paper**
Karen **is assigned to** Reviewer Report Task on Some Paper	→	Reviewer	→	**view** which applies to **Task** **view** which applies to **Paper**

Assignments	Roles	Permissions
Lucy **is assigned to** PLOS BIO journal →	Internal Editor →	**view** which applies to **Journal** **view** which applies to **Paper** **view** which applies to **Task**
Bob **is assigned to** Some Paper →	Author →	**view** which applies to **Paper**
Karen **is assigned to** Reviewer Report Task on Some Paper →	Reviewer →	**view** which applies to **Task** **view** which applies to **Paper**

Lucy can view the PLOS Bio journal,
view any paper within the PLOS Bio journal,
and view any task within the PLOS Bio journal.

Assignments	Roles	Permissions
Lucy **is assigned to** PLOS BIO journal	Internal Editor	**view** which applies to **Journal** **view** which applies to **Paper** **view** which applies to **Task**
Bob **is assigned to** Some Paper	Author	**view** which applies to **Paper**
Karen **is assigned to** Reviewer Report Task on Some Paper	Reviewer	**view** which applies to **Task** **view** which applies to **Paper**

Bob can only view "Some Paper".

Assignments		Roles		Permissions
Lucy **is assigned to** PLOS BIO journal	→	Internal Editor	→	**view** which applies to **Journal** **view** which applies to **Paper** **view** which applies to **Task**
Bob **is assigned to** Some Paper	→	Author	→	**view** which applies to **Paper**
Karen **is assigned to** Reviewer Report Task on Some Paper	→	Reviewer	→	**view** which applies to **Task** **view** which applies to **Paper**

Karen can view the Reviewer Report Task she is assigned to.

She can also view the paper that is related to the Reviewer Report Task.

Karen can view the paper that is related to the Reviewer Report Task?

Assignments **Roles** **Permissions**

Karen is assigned to **Reviewer Report Task** on Some Paper → Reviewer → view which applies to **Task**, view which applies to **Paper**

This provides the context, the "how" you get permissions.

This provides "what" you have to permission to in that context

How does the “applies to” know how to match up w/ the “thing” in the assignment context?

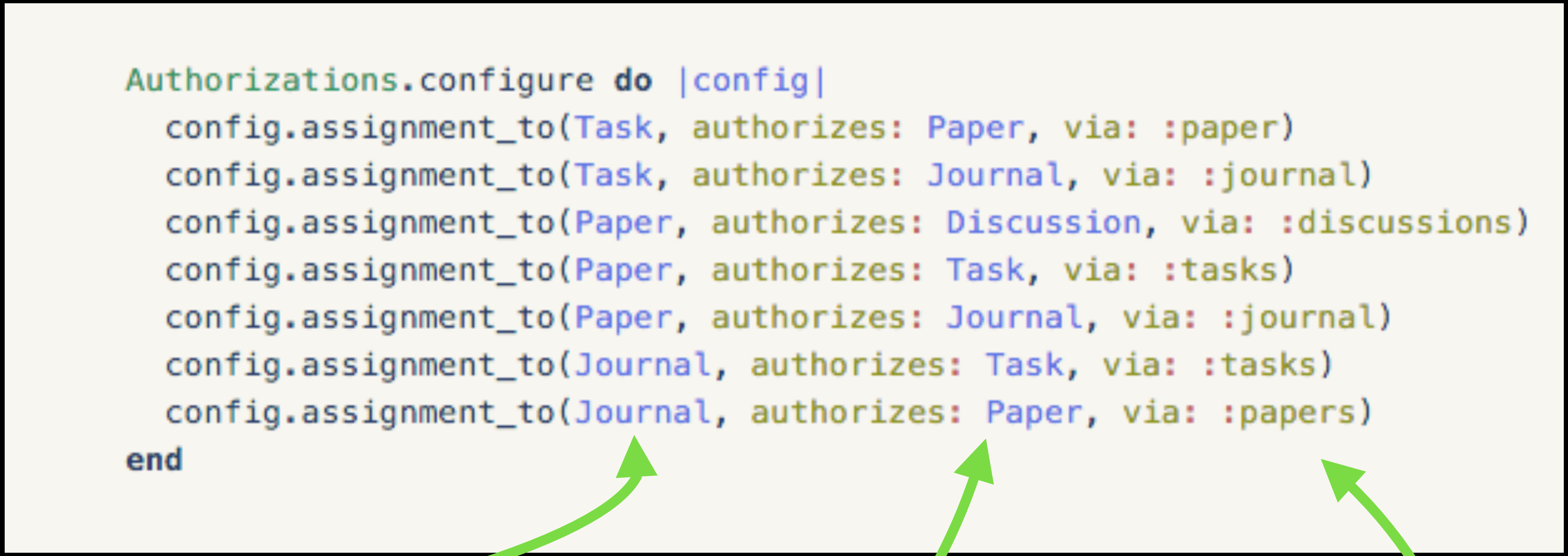
Assignments → **Roles** → **Permissions**

Karen is assigned to **Reviewer Report Task** on Some Paper → Reviewer → view which applies to **Task**
view which applies to **Paper**

This provides the context, the “how” you get permissions.

This provides “what” you have to permission to in that context

There’s a configuration file that spells out how things relate to one another.



```ruby
Authorizations.configure do |config|
  config.assignment_to(Task, authorizes: Paper, via: :paper)
  config.assignment_to(Task, authorizes: Journal, via: :journal)
  config.assignment_to(Paper, authorizes: Discussion, via: :discussions)
  config.assignment_to(Paper, authorizes: Task, via: :tasks)
  config.assignment_to(Paper, authorizes: Journal, via: :journal)
  config.assignment_to(Journal, authorizes: Task, via: :tasks)
  config.assignment_to(Journal, authorizes: Paper, via: :papers)
end
```

What you’re assigned to.

What that assignment gives you access to.

What method/association/scope to use to get access.*

** this is currently expected to be (or return) an ActiveRecord association/scope and not just a Ruby method. May change later.*

This file lives on its own so we can:

- Further decouple authorization logic from model logic
- Raise visibility and discoverability of how access is granted (or not) to developers
- Not bury this deep inside some complex method or SQL query too never be found or understood again
- Allow “how” authorization is wired up to be tested in isolation on its own
- Let models change independently of authorization and vise-versa

So back to Karen.

Can view the paper that is related to the Reviewer Report Task? Yes, yes she can.

An Absurd Example

$$\begin{tikzcd}
\textbf{Assignments} & \textbf{Roles} & \textbf{Permissions} \\
\text{Gary is assigned to Foo Paper} \arrow[r] & \text{Unfortunate One} \arrow[r] & \boxed{\text{view which applies to } \textbf{Task}}
\end{tikzcd}$$

Gary cannot view the paper even though he's assigned to it.
He doesn't have the right permission.
He only has the permission to view Tasks.

A Plausible Example

Assignments		**Roles**		**Permissions**
Grace is assigned to Genetics journal	→	Internal Editor	→	view which applies to **Paper**
Grace is assigned to the Hopper Files Paper on the PLOS One journal	→	Author	→	view which applies to **Paper**

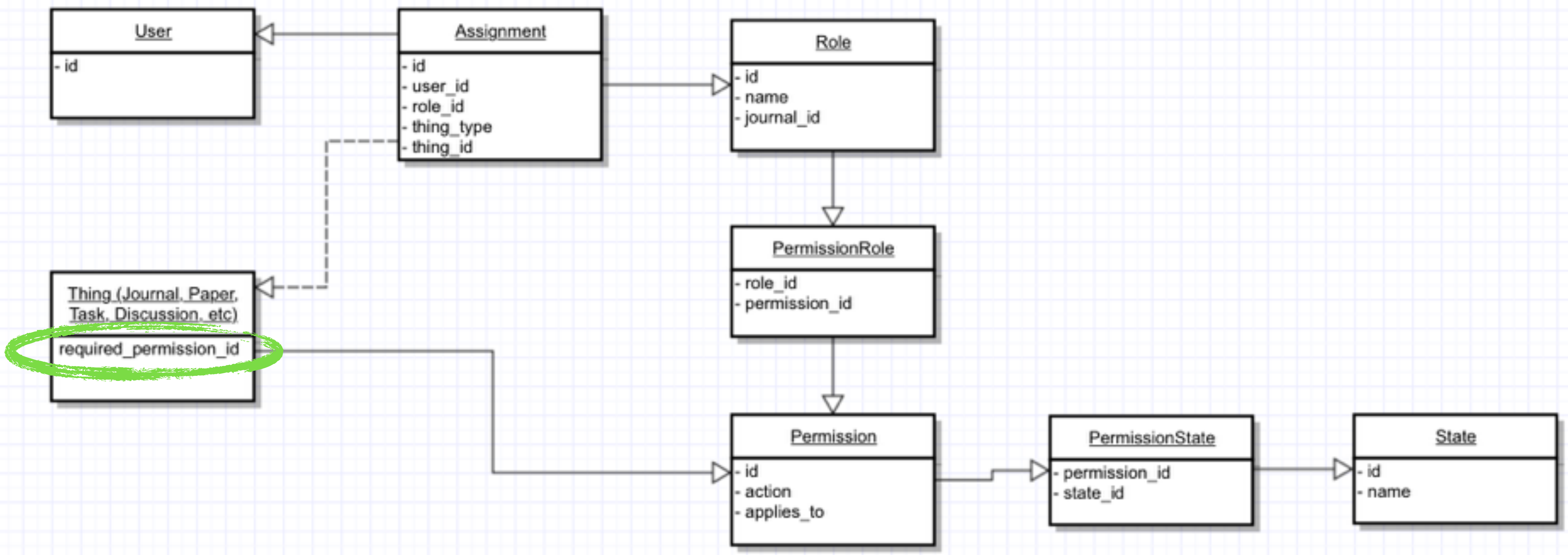
Grace can view all the papers in the Genetics Journal.
She can also view her paper on the PLOS One journal.
She cannot view any other papers in the PLOS One journal.
She cannot view any tasks on either journal.

What about roles like Billing Staff?

Billing Staff Primer

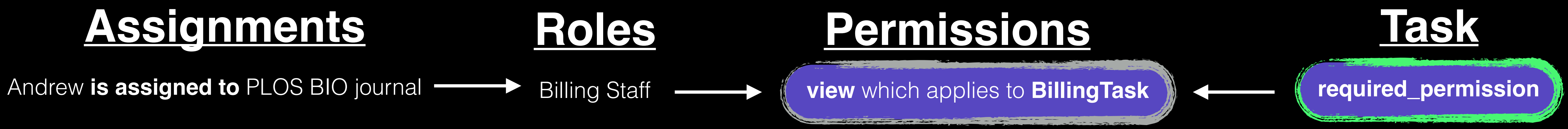
- Billing Staff are assigned at the journal level
- They can view all billing tasks/cards for their journal
- They cannot view any other tasks
- They may be able to do other things like view the paper, but the tricky thing is how to limit them to just the billing task

We can introduce the concept of a required permission that points to a specific permission



A required permission is optional but when it's set it can be used to enforce filtering to a particular a thing (in this case BillingTask).

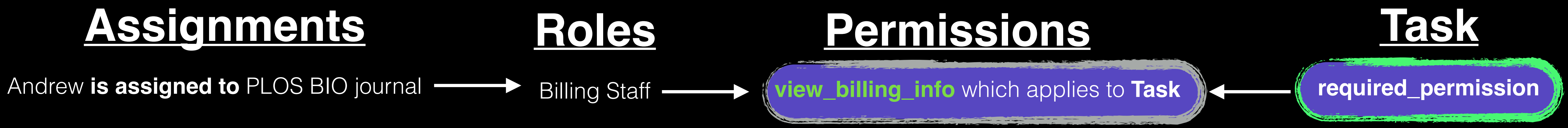
Example #1: use same *view* permission



Assignments | **Roles** | **Permissions** | **Task**

Andrew **is assigned to** PLOS BIO journal → Billing Staff → **view** which applies to **BillingTask** ← **required_permission**

Example #2: OR make new permission



Assignments | **Roles** | **Permissions** | **Task**

Andrew **is assigned to** PLOS BIO journal → Billing Staff → **view_billing_info** which applies to **Task** ← **required_permission**

required permission cont.

- **Where does the required_permission comes from?** It likely comes from the task template or whatever factory is responsible for creating things that have specific requirements.
- **Can it be NULL?** Currently, yes.
- **What does NULL indicate?** That there's no specific permission requirement.
- **Why not just create a permission that "applies_to" BillingTask and omit the required permission?** There needs to be a way to exclude specific kinds of Task(s) or even specific Tasks (maybe ad-hoc tasks). E.g. if someone has a "view" permission that applies to "Task" we don't want that to grant permission on a BillingTask. By having a required permission on BillingTask we can exclude it.

Using Authorization

@

0 ft.

Back-end Usage

- Minimal API
- If you’ve used CanCan the API will feel familiar
- Doesn’t use CanCan
- Is data driven

Checking Permission

```
user.can?(:view, paper)
```

Permission

Object you're checking permissions on

Retrieving a list of things you can access

```
user.filter_authorized(:view, Paper.all)
```

The filter.

Permission

The objects you're filtering against.

user.filter_authorized(:view, Paper.all)

- filter_authorized **has *some* smarts**
- It **will not *load*** all Papers in memory even though you said Paper.all
- It is **smart enough to build queries** based on what you give it
- E.g. an Array of objects, an ActiveRecord::Relation, a chained set of ActiveRecord::Relation(s), etc.

It’s more likely that you’ll have specific objects in mind, not “<Model>.all”

- user.filter_authorized(:view, journal.papers)
- user.filter_authorized(:view, paper.tasks)
- user.filter_authorized(:view, paper.tasks.where(foo: true))

Front-end Usage Examples

- Uses ember-can for API (basically ***free*** ember integration).
- Replaces ember-can's default lookup-strategy
- Expects the server to give it a lookup table of permissions for a user

ember-can in a view

```
{{#if (can “participate_in_discussion”, paper)}}
  <button {{action “new”}}>Add Reply</button>
{{else}}
  You can't talk in this discussion!
{{/if}}
```

“can expectations in Aperta

```
(can “participate_in_discussion”, paper)
```

Permission (required)

The object you’re checking permission for. (required)

Use the CanMixin to use permissions elsewhere

```
import Ember from 'ember';
import { CanMixin } from 'ember-can';

export default Ember.Route.extend(CanMixin, {
  beforeModel: function() {
    if (!this.can(‘view’, paper)) {
      this.transitionTo('index');
    }
  }
});
```

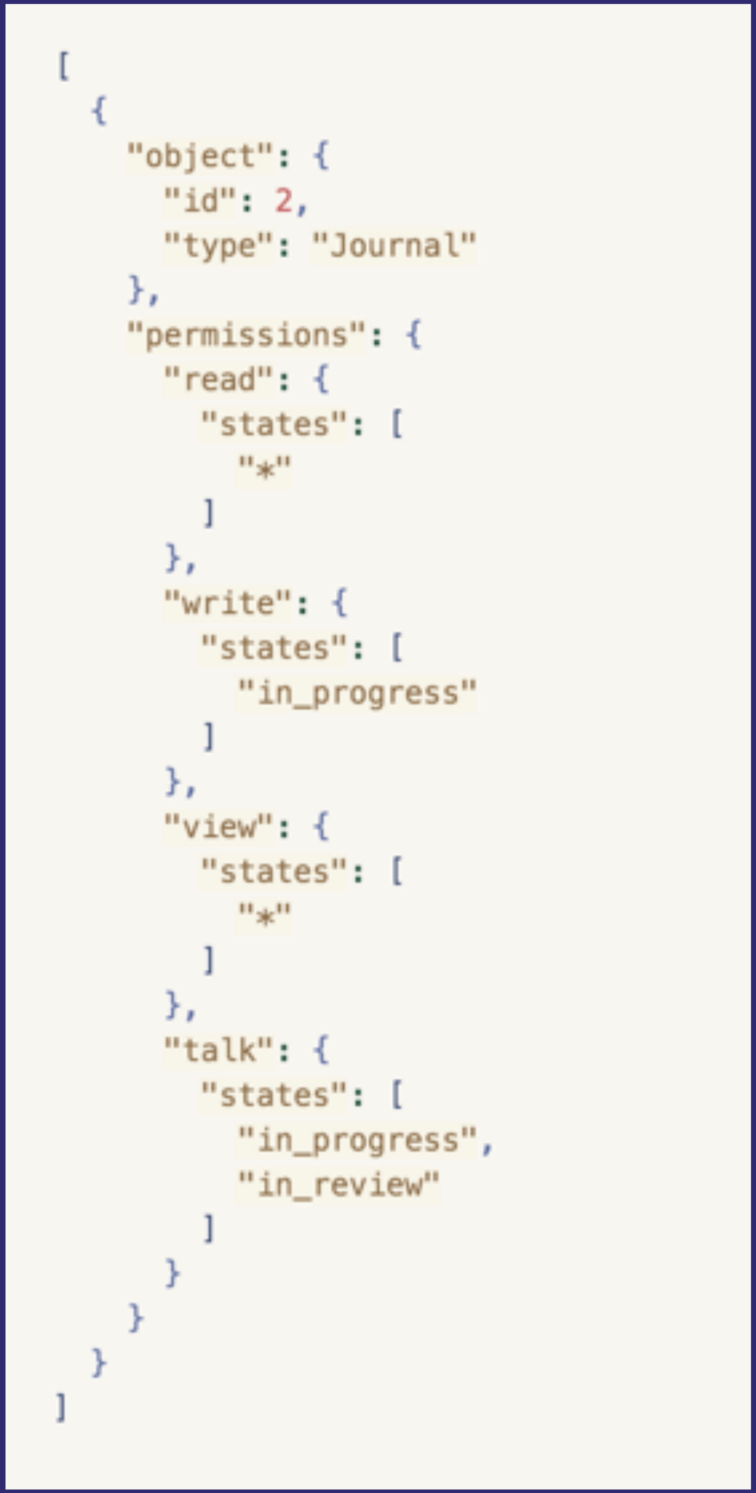
can(…) has some smarts

- Only persisted objects (with an id) actually do a permission lookup
- Objects without ids are assumed to be something the user is creating so they **can**(…) will always return true
- These are Aperta defined smarts, not ember-can smarts

How the client & server communicate

Server sends a permission table

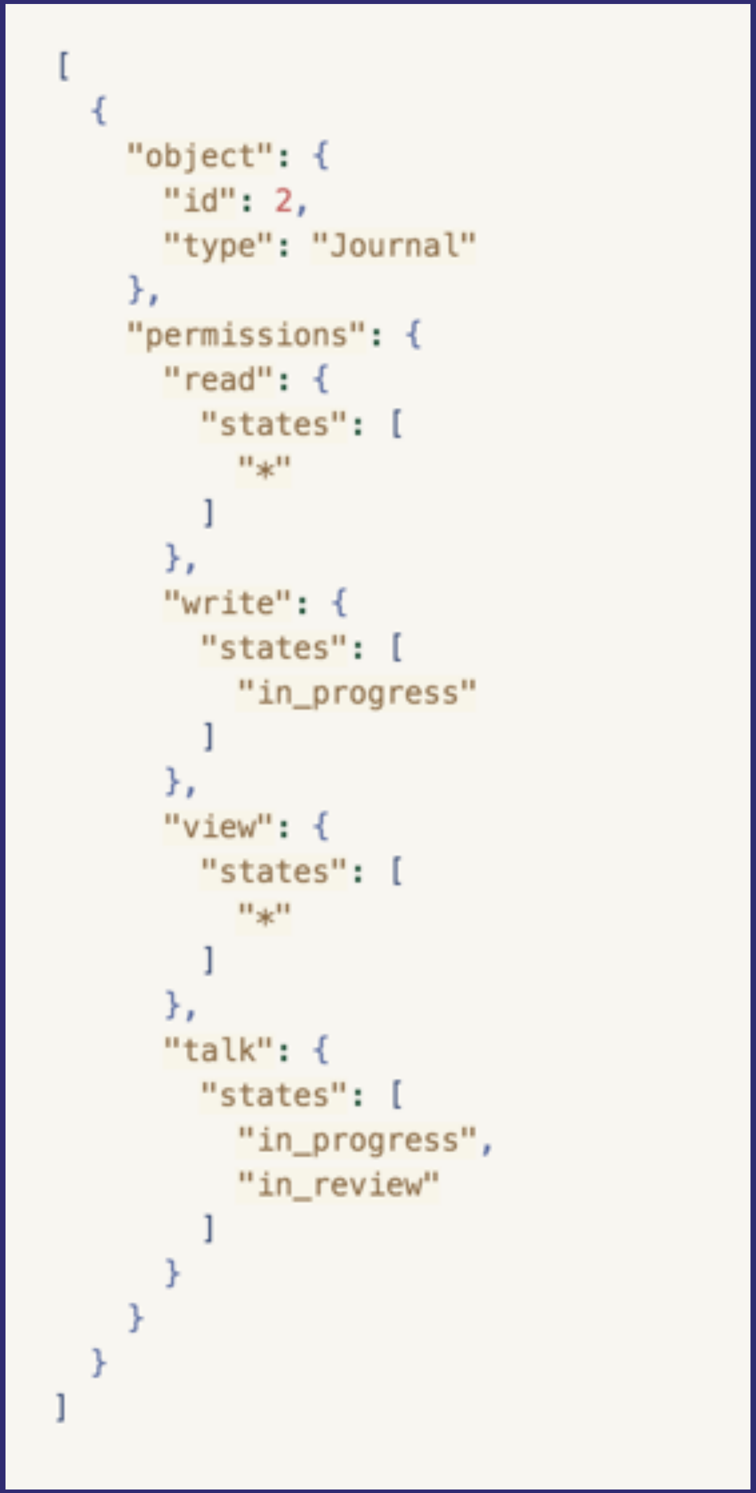
(below is one entry)



```json
[
  {
    "object": {
      "id": 2,
      "type": "Journal"
    },
    "permissions": {
      "read": {
        "states": [
          "*"
        ]
      },
      "write": {
        "states": [
          "in_progress"
        ]
      },
      "view": {
        "states": [
          "*"
        ]
      },
      "talk": {
        "states": [
          "in_progress",
          "in_review"
        ]
      }
    }
  }
]
```

Server sends a permission table

(below is one entry)



```
[
  {
    "object": {
      "id": 2,
      "type": "Journal"
    },
    "permissions": {
      "read": {
        "states": [
          "*"
        ]
      },
      "write": {
        "states": [
          "in_progress"
        ]
      },
      "view": {
        "states": [
          "*"
        ]
      },
      "talk": {
        "states": [
          "in_progress",
          "in_review"
        ]
      }
    }
  }
]
```

The object

Permission "read"

Any state

Permission "write"

Must be in state "in_progress"

Permission "view"

Any state

Permission "talk"

Must be in state "in_progress" OR "in_review"

Client reads look up table

- Client knows how to use the look up table
- Look-up table may be side loaded with requests
- Side-loaded requests will likely only include part of the permission table (for things you're looking at)
- If client cannot find the permission in a lookup table it falls back to asking the server

Trade-offs

Being data-driven

- Adds a level of structure and consistency to how authorization is modeled. **:)**
- Some ramp-up to understanding the model required. **:l**
- Makes it possible to send authorization tables to the client (didn’t exist prior) **:)**

Being data-driven

- A certain level of complexity will exist around building SQL queries **:(**
- Balancing SQL performance and Ruby/ActiveRecord performance adds interesting code **:|**
- Isolates complexity exposes a clean API. **:)**

Performance

- Asking the server to tell you all of the permissions for every **thing** that a user has access to can be **expensive.**

- Asking the server to tell you the permissions for specific kinds of objects given a starting point is **much more performant** (e.g. what are all the tasks for this paper I can access)

- It's bullet point #2 that we believe is the normal use-case for Aperta.

Side loading

- We think side loading permissions for things being returned in the response may be a good start
- It avoids umpteen requests to the server
- It avoids asking the server to compute permissions for every possible thing the user may touch

~~Negative~~ Permissions

- We’re planning to omit negative permissions
- E.g. what happens when a user has both a “can access” and a “cannot access” permission?
- Our current plan is to express more narrowly scoped “positive” permissions and to not assign them to roles.
- Trade-off: More permissions, but less complex system and likely more readable code.

Coupling

- We expect some roles to be coupled to the various parts of the system initially.
- E.g. Knowing that creating a paper assigns the Author role for the current journal.
- E.g. Knowing that the user who accepts a reviewer invitation gets the Reviewer role
- Adding even more configurability and flexibility comes at a cognitive cost. We think this is a piece to punt on for now.

What about ad-hoc tasks?

- Good question
- Maybe task templates have a default permission associated with them. Possibly by using the *required_permission.*
- Perhaps in the ad-hoc task UI a user can select what the permission is required to par-take in the task
- Or perhaps something else? Depends on what the PLOS ends up needing.
- We need more info, but we don’t think this will be a show stopper.

Closing Thoughts

- **More confident than not**: These ideas have been implemented, iterated on, and tested in a playground app (which pulled Aperta models). We've gone from 15% confident to probably 80 ~ 85%. we won't get that last 15% - 20% until we start putting it in place in Aperta.
- **Things to keep an eye on:** Performance on real data, edge cases we didn't think of, other ways of organizing information.
- **Encourage implementation evolution:** A minimal API, a non-leaky (or minimally leaky) R&P abstraction, a contract between client and server, and a solid test suite should support conversation tweaks and radical ideas to improve the underlying implementation while at the same time having minimal surface area affected by those changes.
- **Change is necessary:** We think we've got a really good starting point, but we expect change and iteration over time.
- **Feedback**: Ask questions and poke holes. We want you to grok this as we move forward and want to fill in holes as early as we can.

The End.