We should probably start brainstorming how applications that do not

require total ordering guarantees (which the resource machine provides by default) can be built on the resource machine (and on Anoma), and in particular how they can be built in a way which takes advantage of their looser ordering requirements, including:

- ensuring that writes to CRDT-resources do not conflict, and instead perform the appropriate merge
- supporting local / compositionally-merged views which do not need to wait for the ultimate controller's sign-off in order to display updated data

I think the basic template here will look something like the following:

- CRDTs can be represented as regular resources, with the CRDT merge function checked by the resource logic. Transaction functions updating CRDTs will read the latest version and update that with changes in the transaction function (thereby avoiding conflicts). In order to avoid large amounts of duplicated storage (and extra bandwidth), updates will often want to include only changes and refer to most of the pre-existing data structure by hash - presumably CRDT theorists have already thought about these problems, as they aren't special to our model. CRDTs may also often recursively decompose into many resources (e.g. for sub-components of the data structure), which themselves may be sharded across controllers and/or visible to different parties. CRDT resources may allow themselves to be copied and split across controllers, then later merged when desired.
- User interfaces for CRDT-based applications can connect to many controllers (e.g. local node, local network, local area, and some final / global controller), and display updates as soon as they are made available by any controller, simply updating the display as more updates are received. The CRDT merging guarantees should provide corresponding guarantees of user interface consistency (both across users and across time).

Let's take the example of a simple forum (h/t@tg-x), with categories, topics, comments, and likes (roughly, in fact, like Discourse itself)! We want to implement the following logic:

- Categories, kept in an overall index, each associated with a list of topics, modifiable only by admins.
- Topics, each Markdown data, with an authenticated author, and nested (threaded) replies. Topics and comments can be deleted by the topic creator and administrators.
- Likes, authenticated to users, which are just a counter associated with each comment.

This can be implemented with (approximately) the following resource logics:

- AdminIndex

, which keeps a set (CRDT) of administrator identities.

- CategoryIndex

, which keeps a set (CRDT) of the current categories - modifiable only by identities in the AdminIndex

.

- Category

, which keeps a name and a set (CRDT) of the topics in that category.

- Topic

, which keeps a name, a topic creator, and a set (CRDT) of replies.

- Comment

, which keeps a Markdown body, an authenticated author, and a (possibly empty) set (CRDT) of replies.

- CommentLikes

, which keeps a reference to a Comment

and a simple incrementing counter (CRDT).

The only non-convergent operation here is changing the administrator index (after which certain operations might fail which would have succeeded before the change, or vice-versa). Everything else should be immediately displayable based on local consensus information, and can gradually update over time as more transactions become known.