MIMI Protocol

Richard Barnes, Konrad Kohbrok & Travis Ralston
September 2023

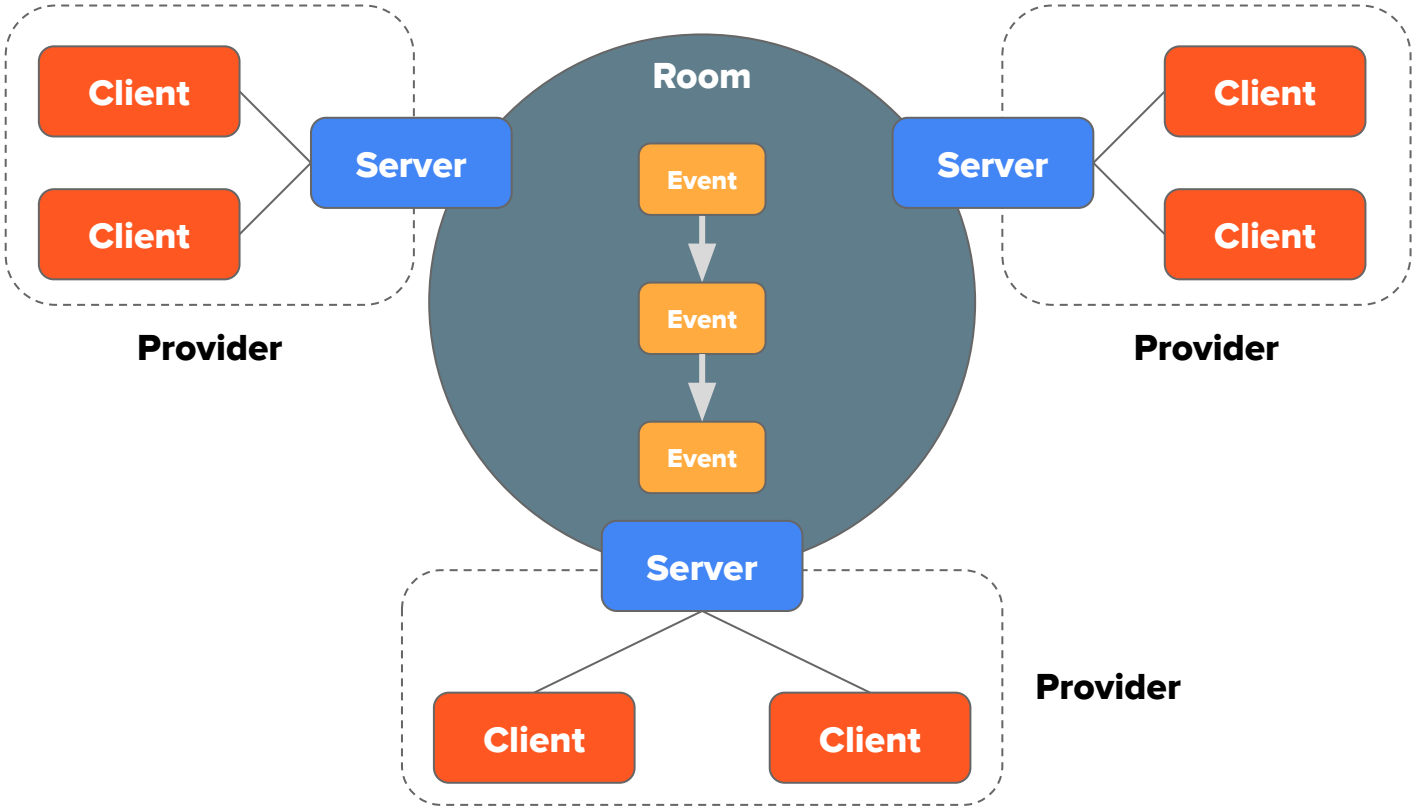
Agenda

- Architecture review
- The current document set
- Some worked example scenarios
  - “Alice starts a conversation with Bob”
  - “Alice leaves the room”

Architecture*

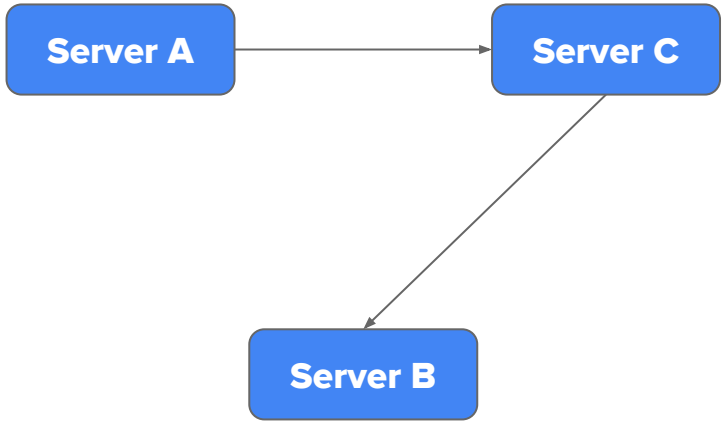
* draft-barnes-mimi-arch

CSSSC


Client
Client
Server
Provider
Room
Event
Event
Event
Server
Client
Client
Server
Provider
Client
Client
Server
Provider

CSSSC

- Each room is hosted at a hub server
- All communications go via the hub (C)
  - Messages
  - State changes
  - Ancillary things like KeyPackage fetches
- Anticipate that not every server will be willing to talk to every other server
  - A and B might not be willing to connect directly, even if they are willing to join a room on C
  - But if users from A and B are in a room hosted at C...
  - A needs to be able to talk to B via C



Terminology

The unit of MIMI functionality is a **room**

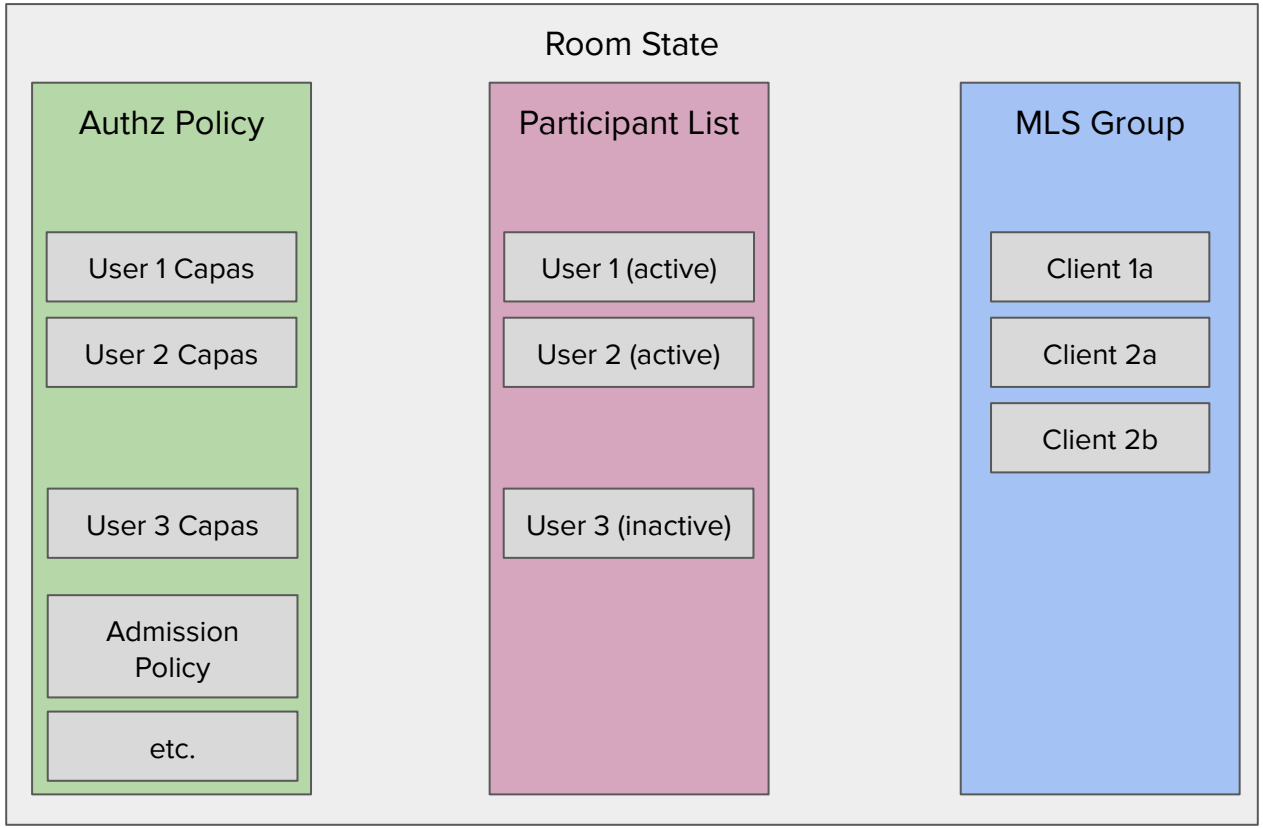
A room has a **state**, which has a few components:

- An **authorization policy**
- A list of **users** who are **participants** in the room
- An MLS **group**, including a list of **clients** who are **members** of the group

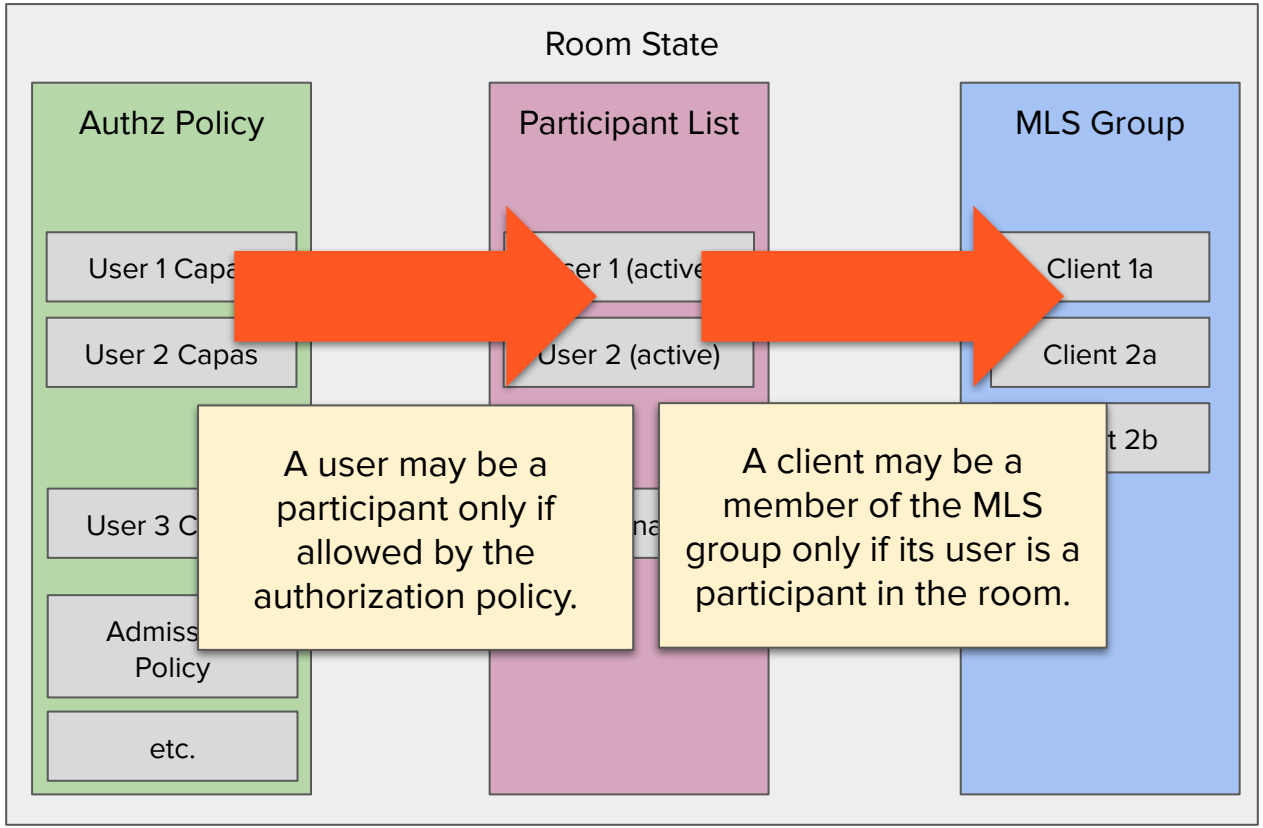
Participants can either be **active** or **inactive**. Active participants have at least one client that is a member of the MLS group.

⚠️ Earlier discussions covered a “policy envelope”. Needs more refinement.

Terminology Illustrated



Preemption


Room State
Authz Policy
Participant List
MLS Group
User 1 Capa
User 2 Capas
User 3 C
Admiss
Policy
etc.
er 1 (active
User 2 (active)
Client 1a
Client 2a
t 2b
A user may be a participant only if allowed by the authorization policy.
A client may be a member of the MLS group only if its user is a participant in the room.

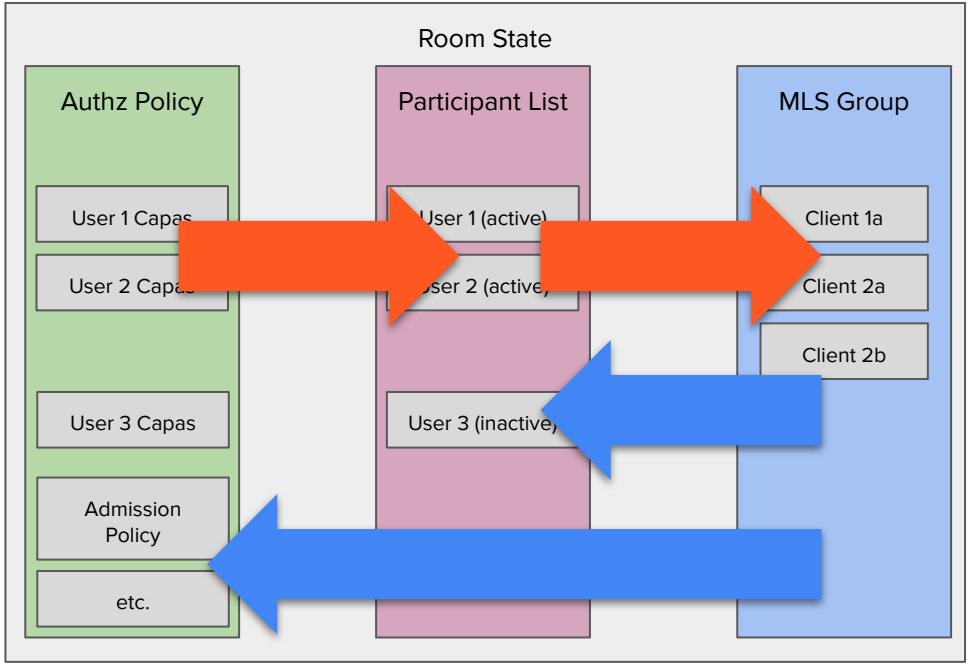
Confirmation

Different aspects of the state are managed via separate “control” sub-protocols

To ensure everyone agrees on state, it is included in the MLS key schedule

- E.g., as a GroupContext extension
- If clients don’t agree, they can’t communicate

Each Commit must reflect the current room state.

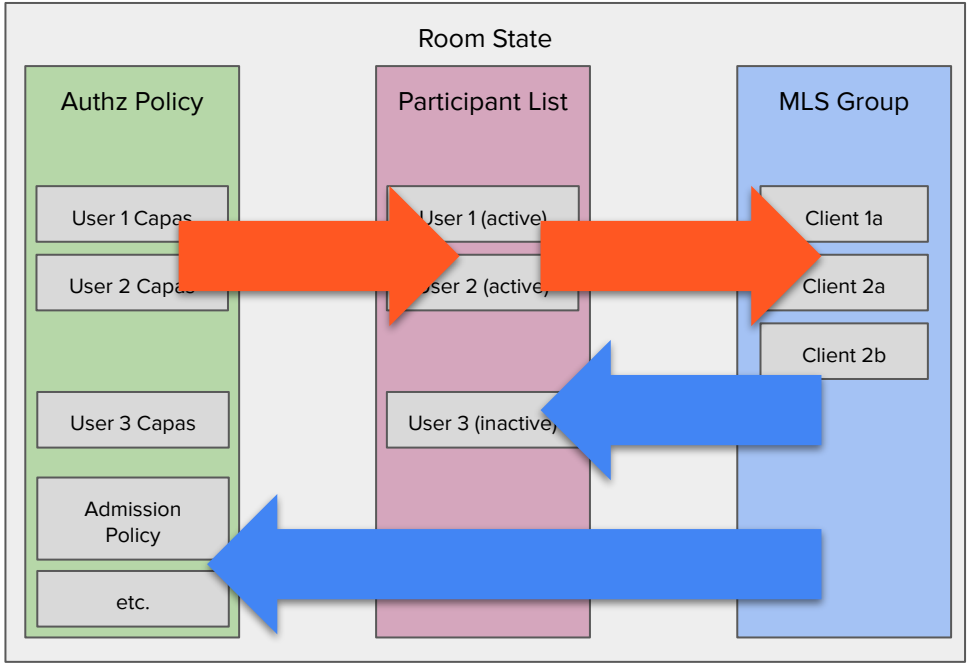


Order of Operations

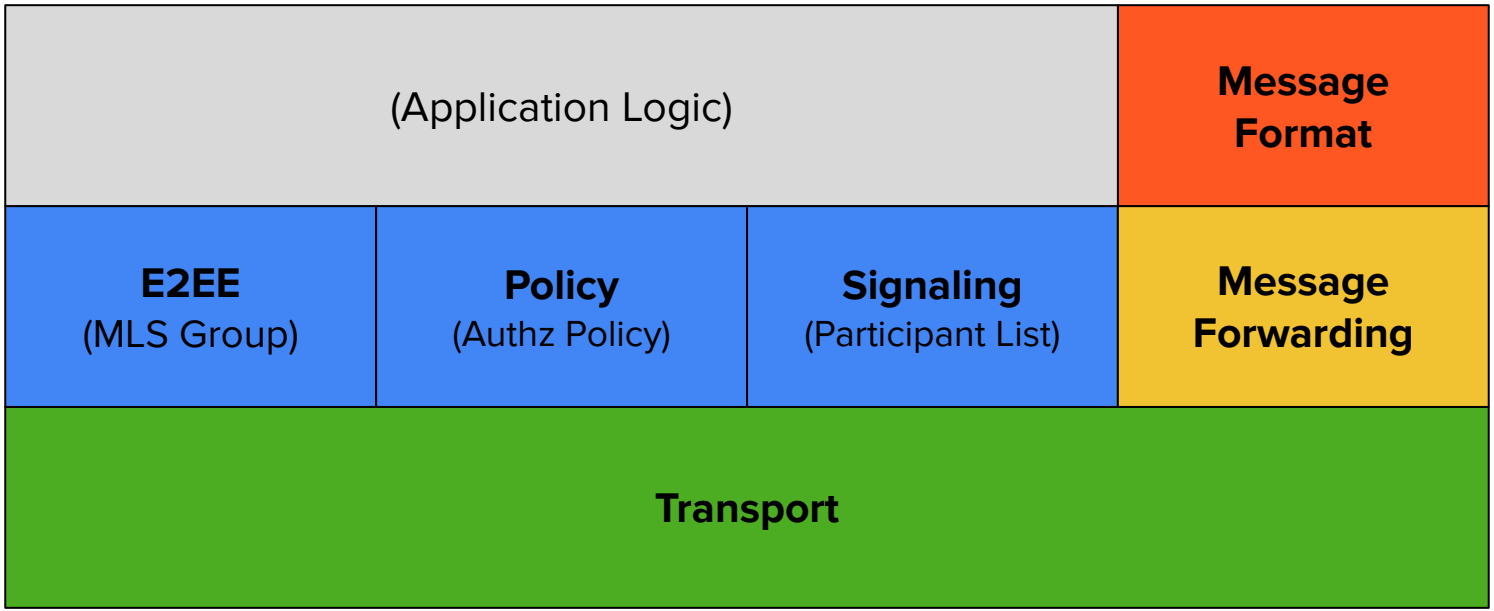
Preemption + Confirmation require some coupling between control protocols

- User must be on Participant List before clients added to MLS
- Clients must be removed from MLS before user leaves Participant List

Sending control messages together will help keep things organized.

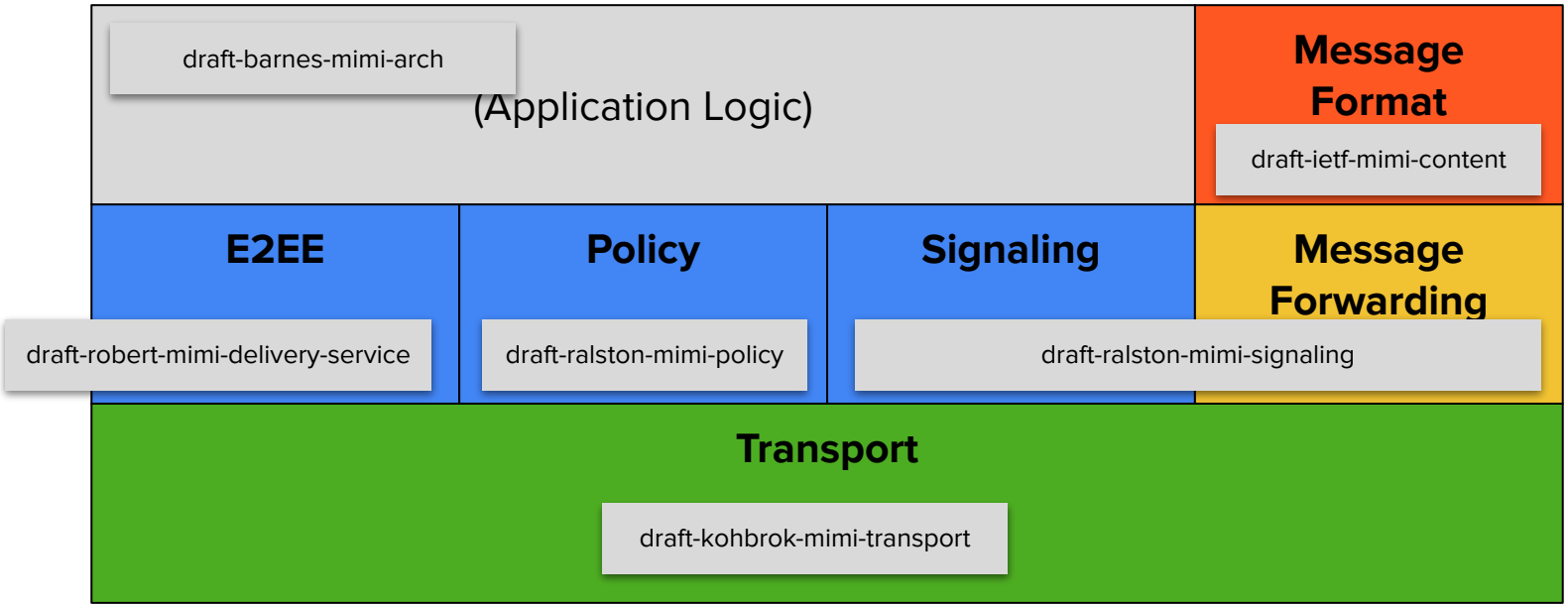


Lego* Bricks



* from Danish *leg godt* ≈ “play well” [Wikipedia]

Documents


draft-barnes-mimi-arch
(Application Logic)
Message Format
draft-ietf-mimi-content
E2EE
Policy
Signaling
Message Forwarding
draft-robert-mimi-delivery-service
draft-ralston-mimi-policy
draft-ralston-mimi-signaling
Transport
draft-kohbrok-mimi-transport

Scenario 1: Alice adds Bob

Assumptions

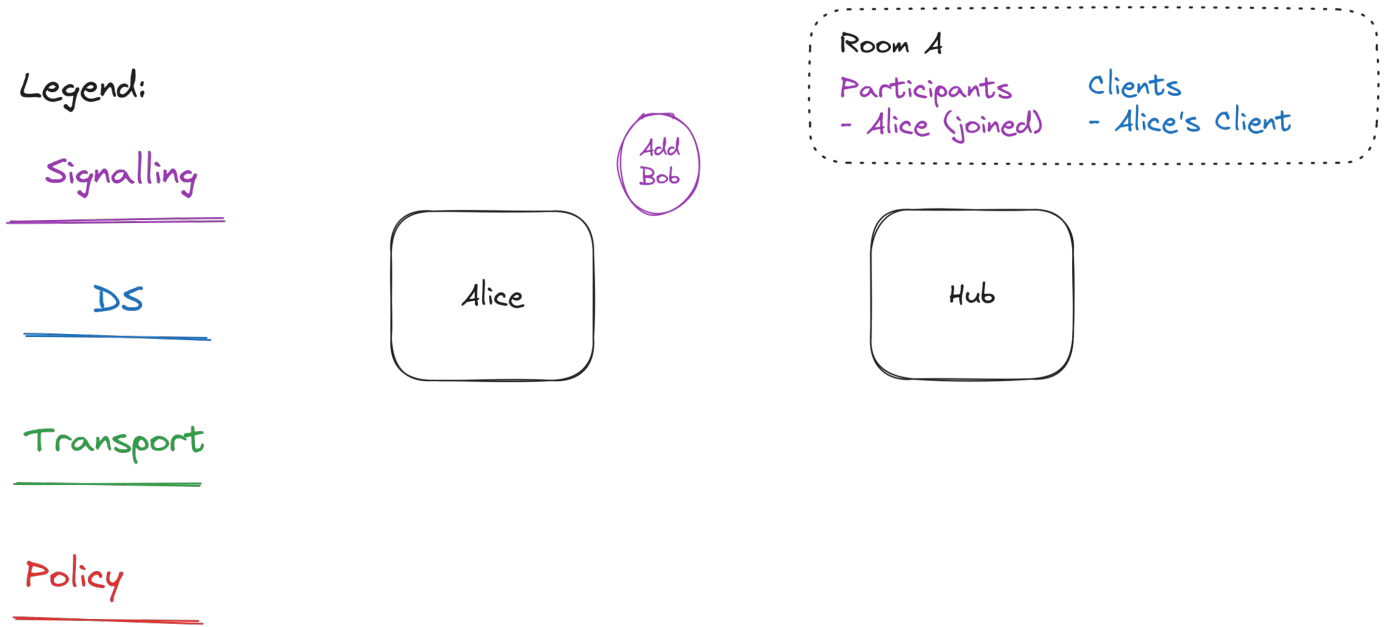
- Any discovery of Bob’s service-specific identifier has already happened
- Alice has any necessary consent to add Bob to a room
- There may be other join flows; this is just one example

Step 1: Alice creates group/room

1. Alice creates the MLS group and associated room within their messaging provider, independent of MIMI.
    a. Alice's server becomes the Hub server for the room
    b. Initial room state covers participant list (just Alice), policy, etc
2. (Optional) Alice adds their other clients to the MLS group
    a. Policy only requires that Alice's participation state be "joined" to do this. Alice can also become an inactive participant by having all of their clients removed from the group at any time.

Not a protocol operation, but initializes room state that drives the protocol later.

Alice creates* an invite event for Bob



*Alice could also commit a bunch of Add proposals here if using a Welcome flow, which skips invite and goes to join (in signaling).

Alice sends the invite to the hub

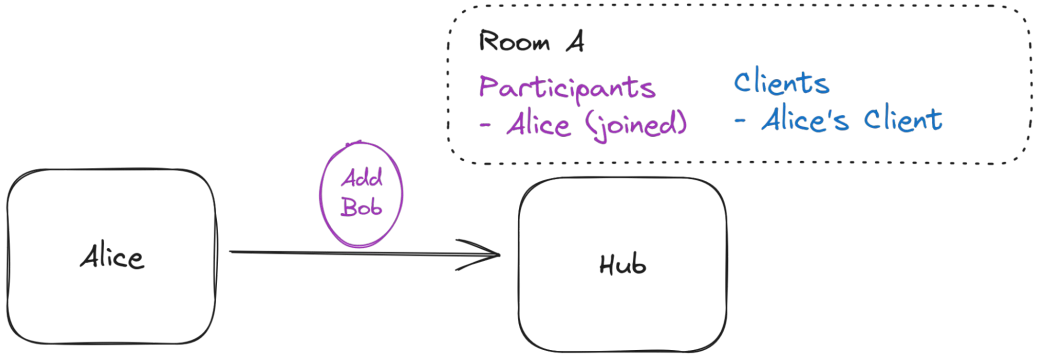
Legend:

Signalling

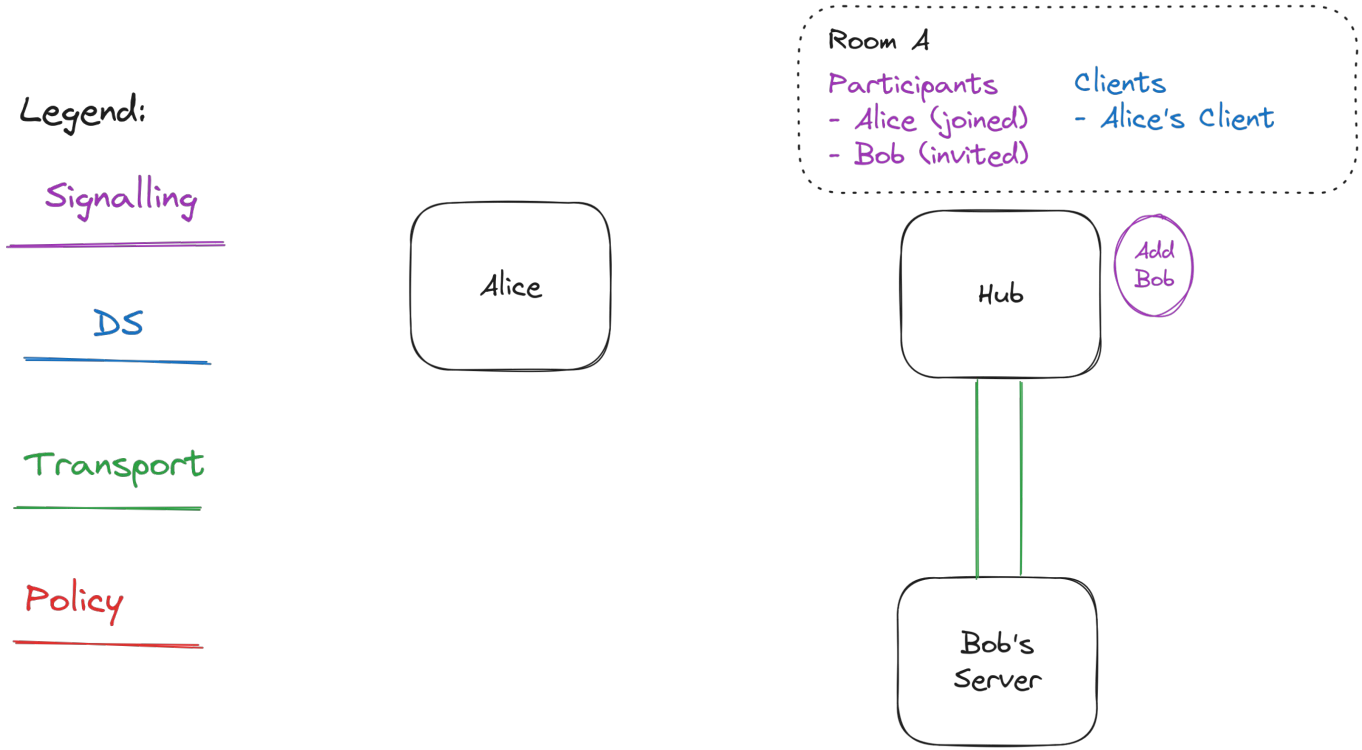
DS

Transport

Policy



Hub establishes secure transport



Hub fans out the signalling event

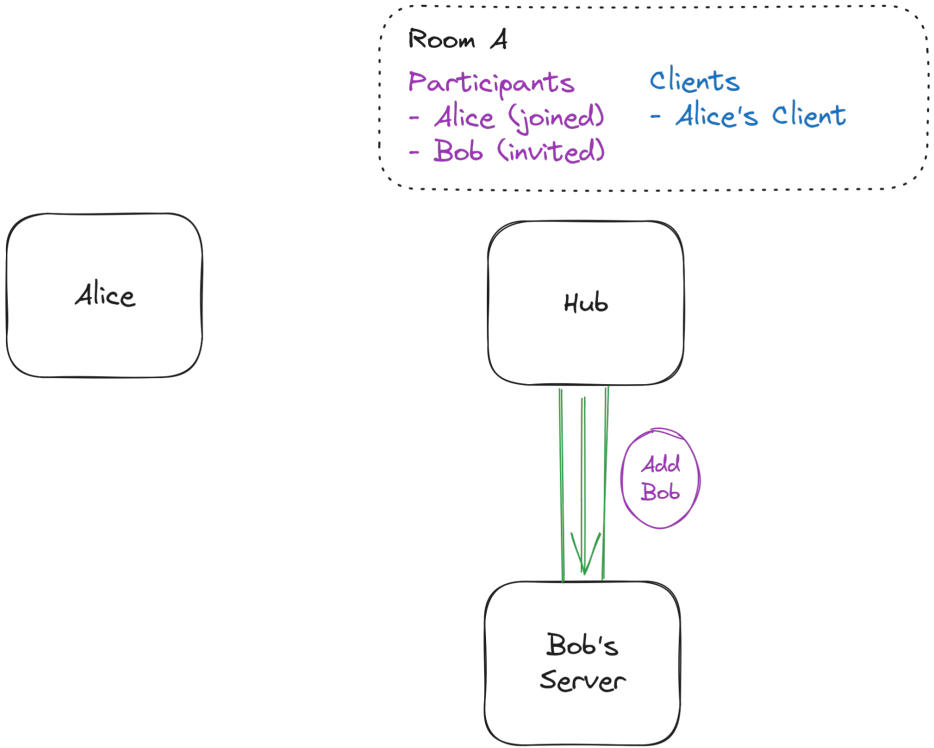
Legend:

Signalling

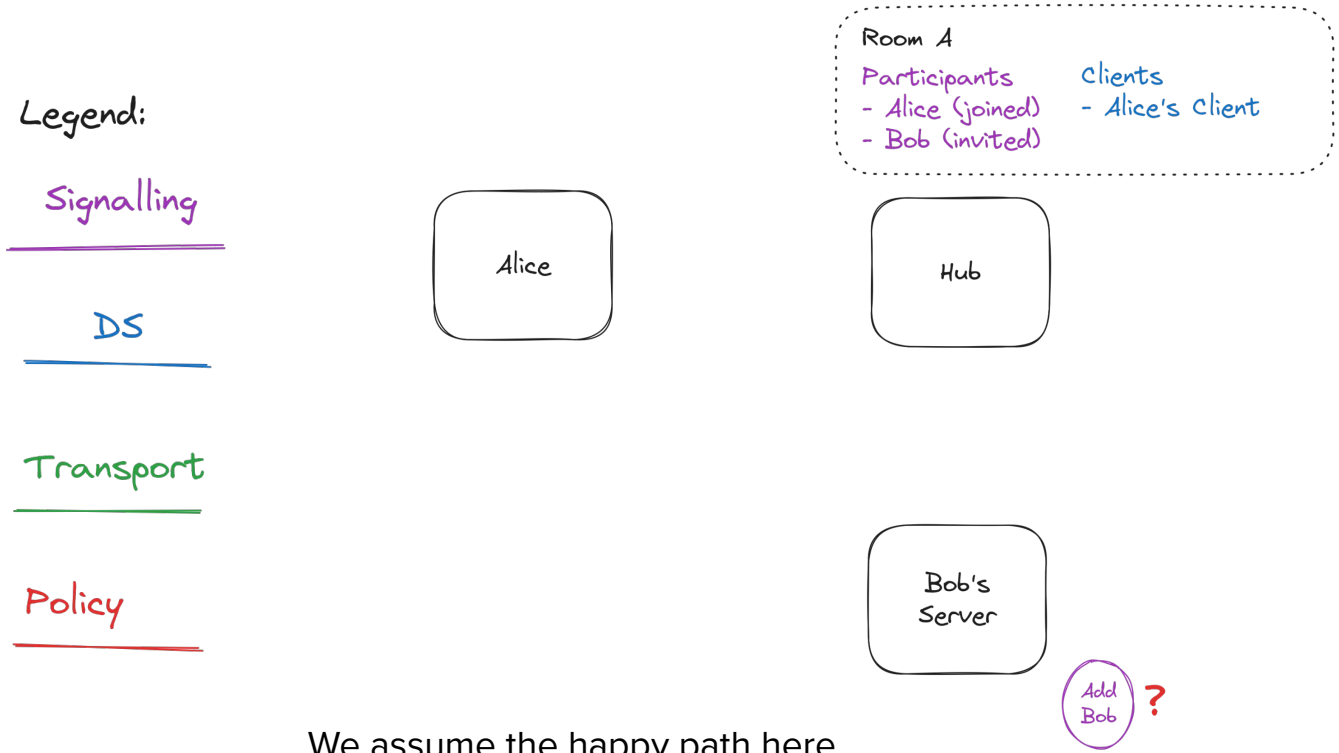
DS

Transport

Policy

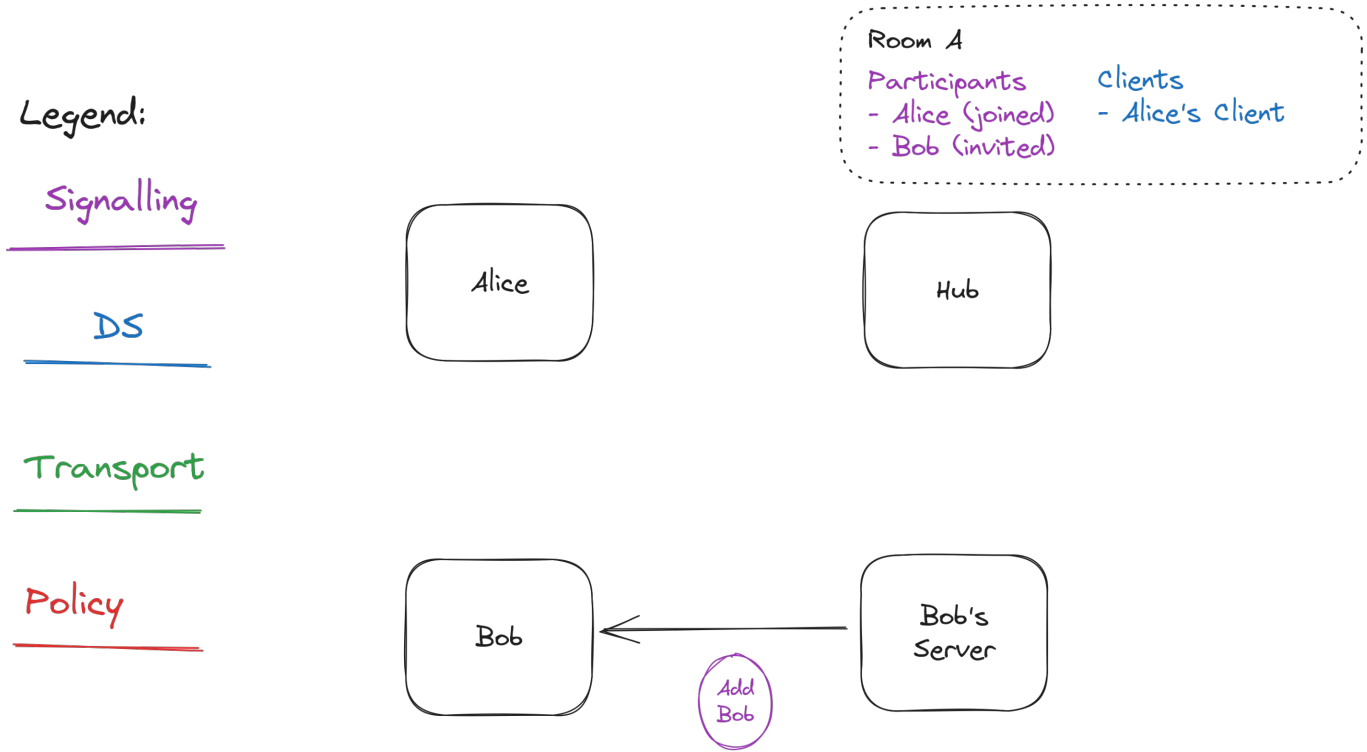


Bob's server checks policy support

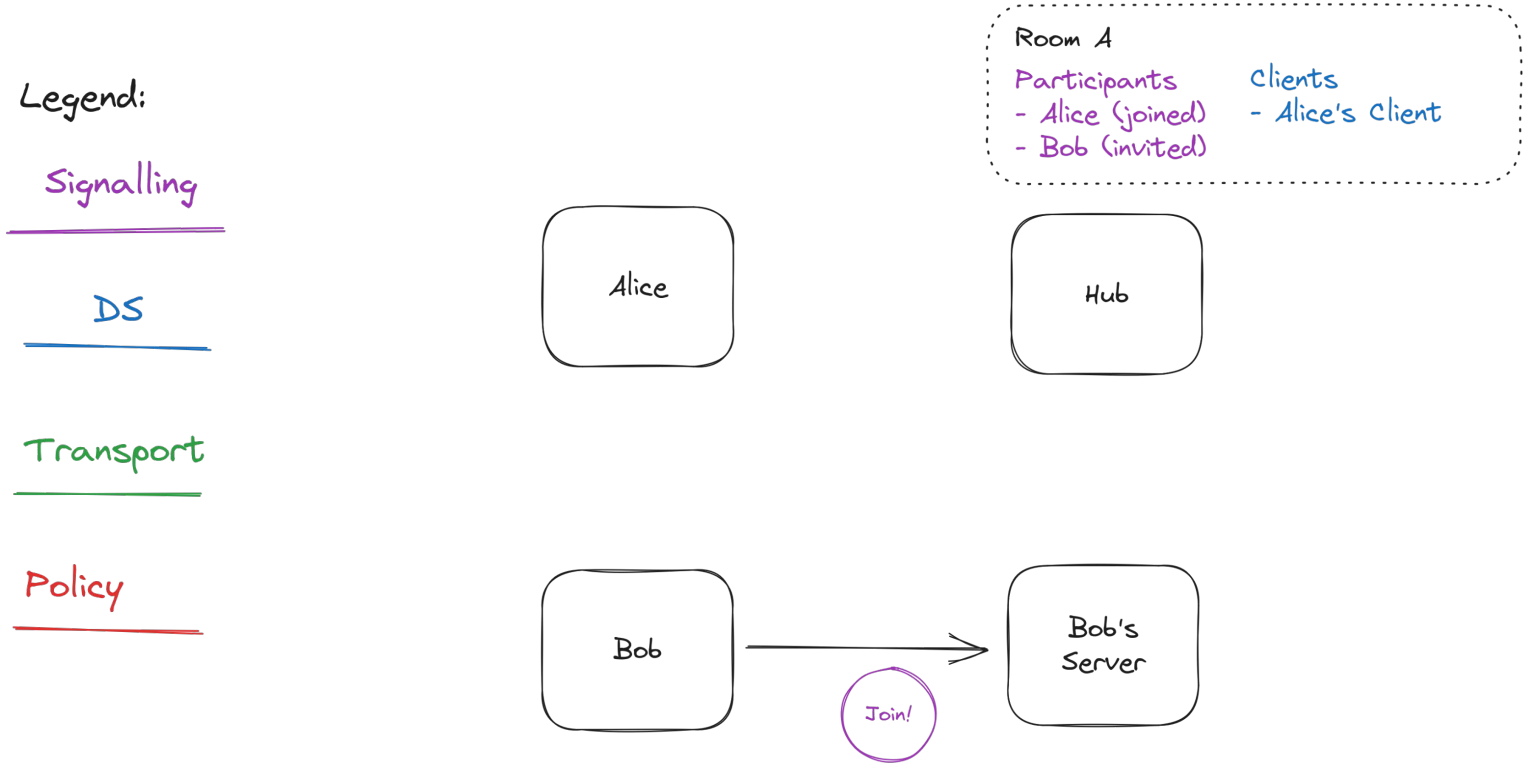


We assume the happy path here.

Bob's server stores-and-forwards invite



Bob accepts invite (eventually)



Bob's server forwards join even to hub

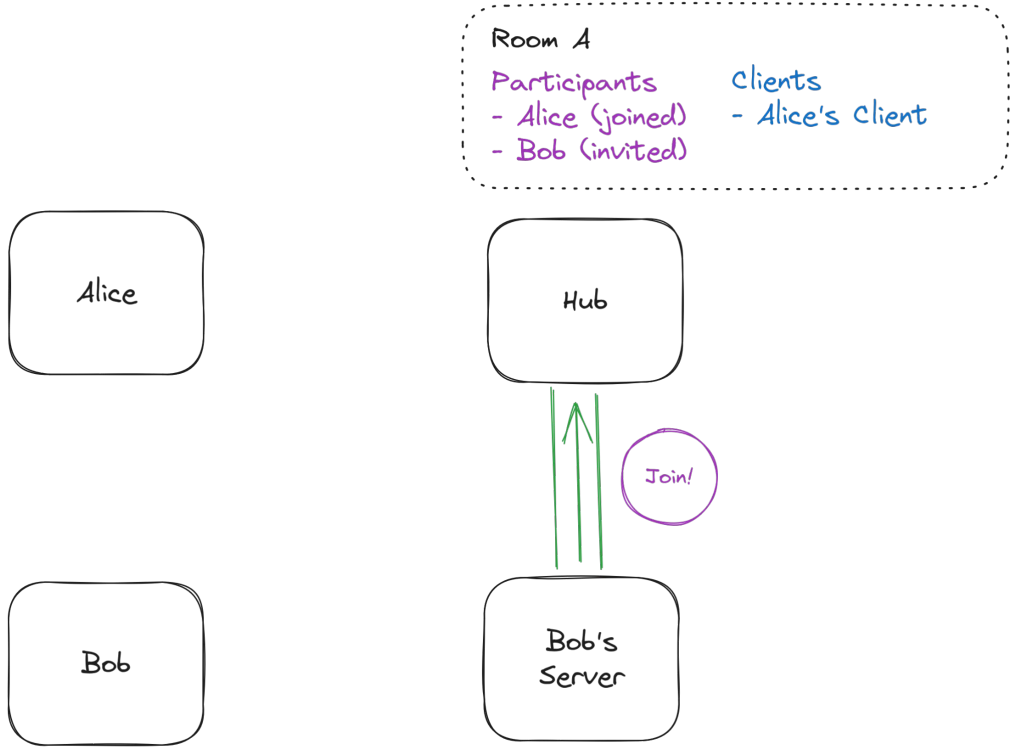
Legend:

Signalling

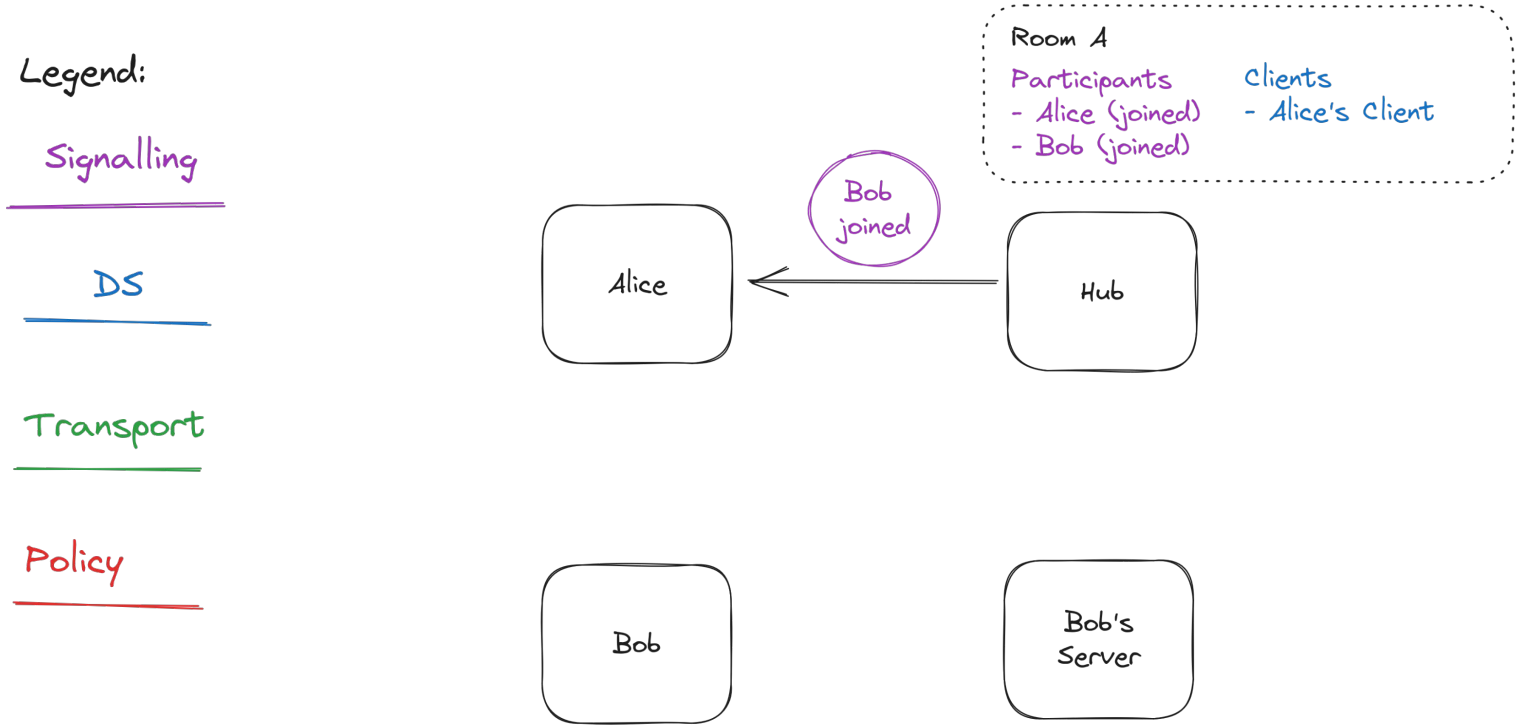
DS

Transport

Policy



Hub fans out join event



Hub sends GroupInfo to Bob's server

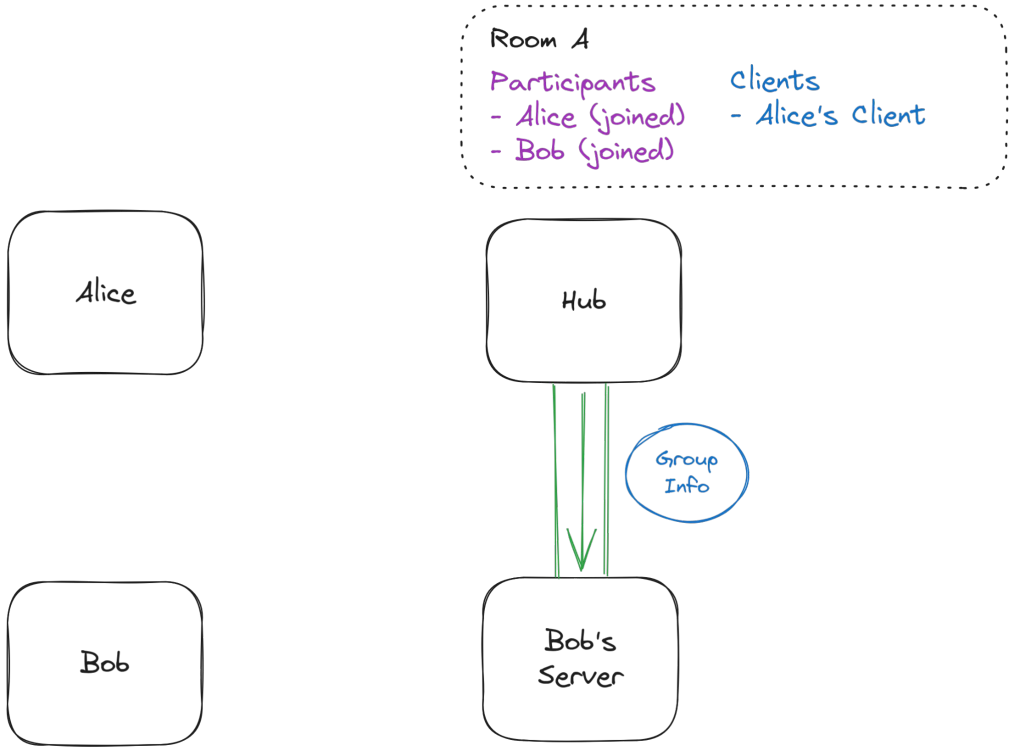
Legend:

Signalling

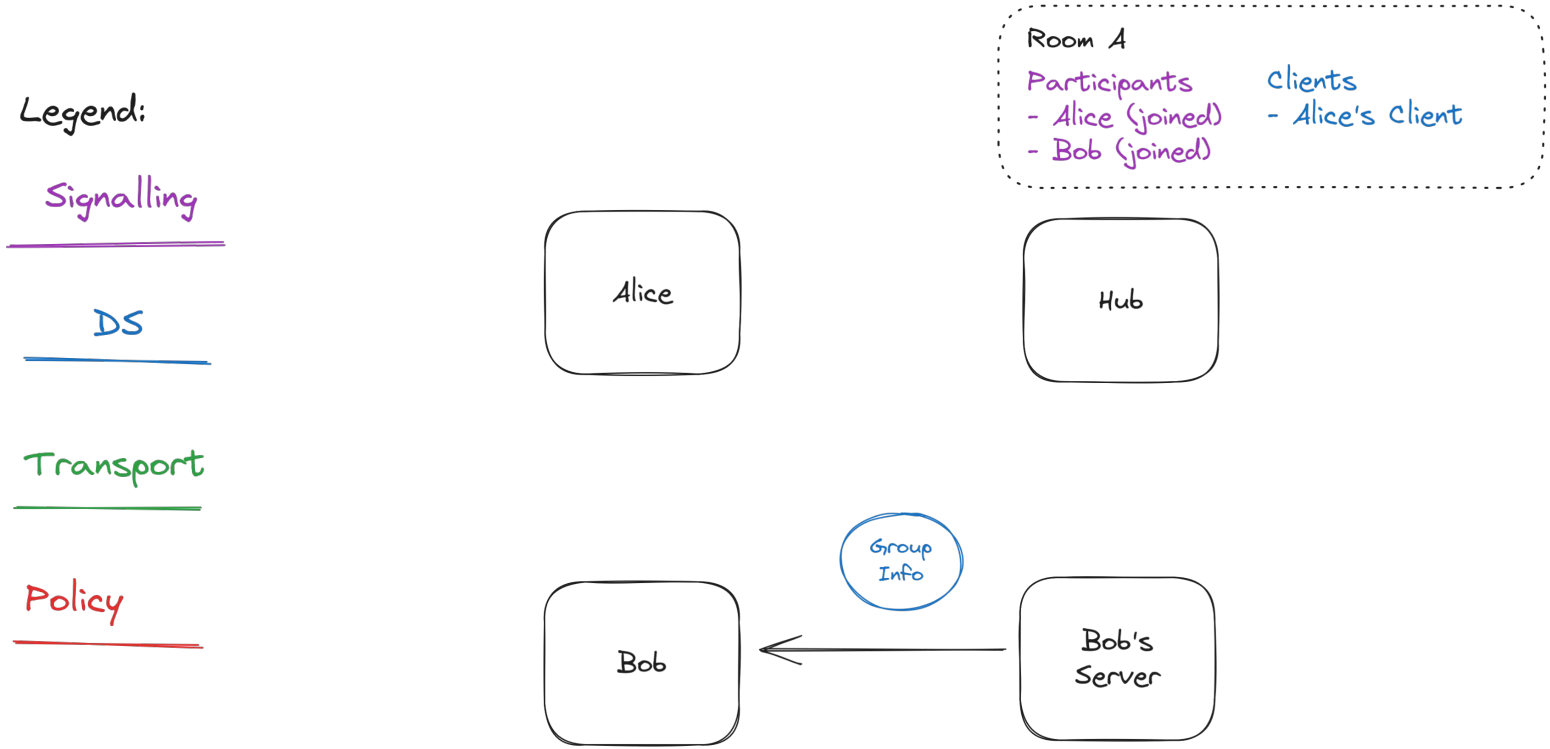
DS

Transport

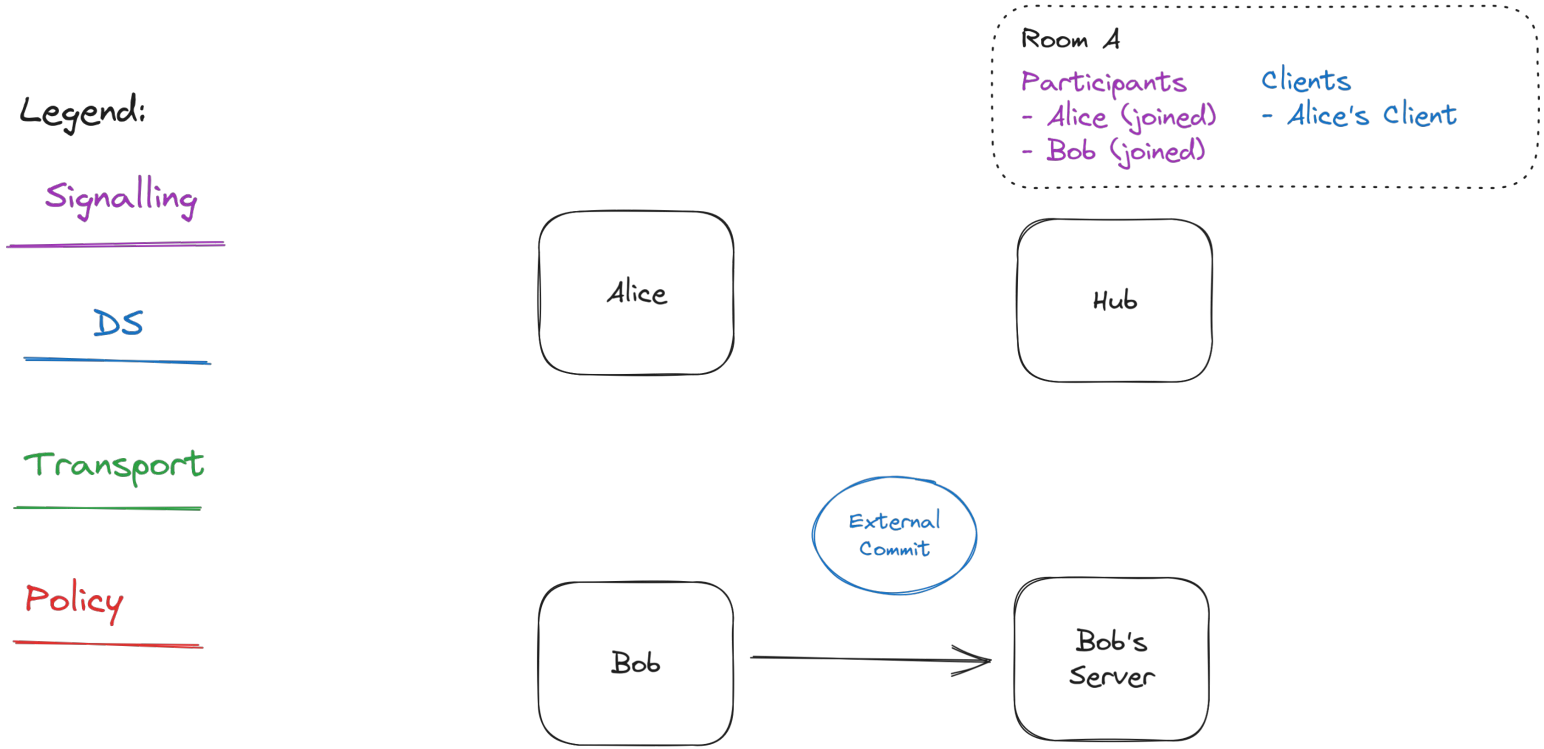
Policy



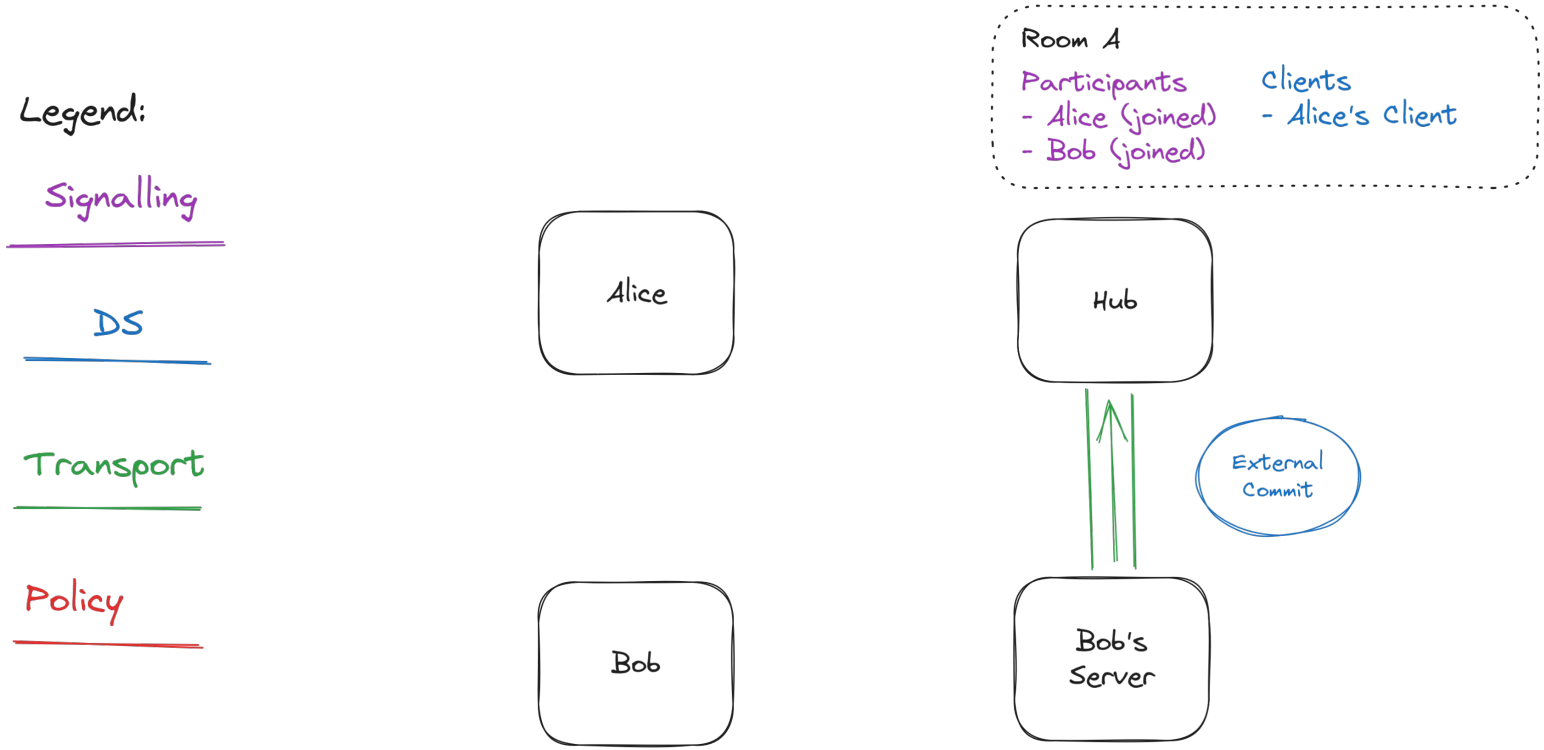
Hub sends GroupInfo to Bob's server



Bob joins group via External Commit



Bob's server forwards External Commit



Hub fans out the External Commit

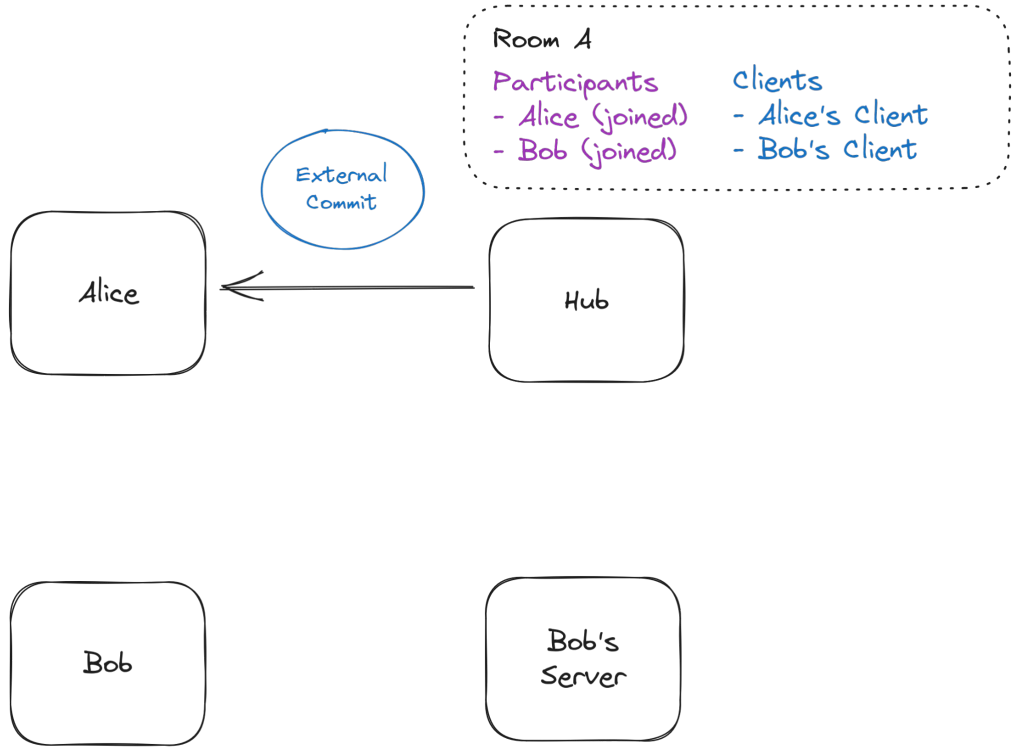
Legend:

Signalling

DS

Transport

Policy



Hello world

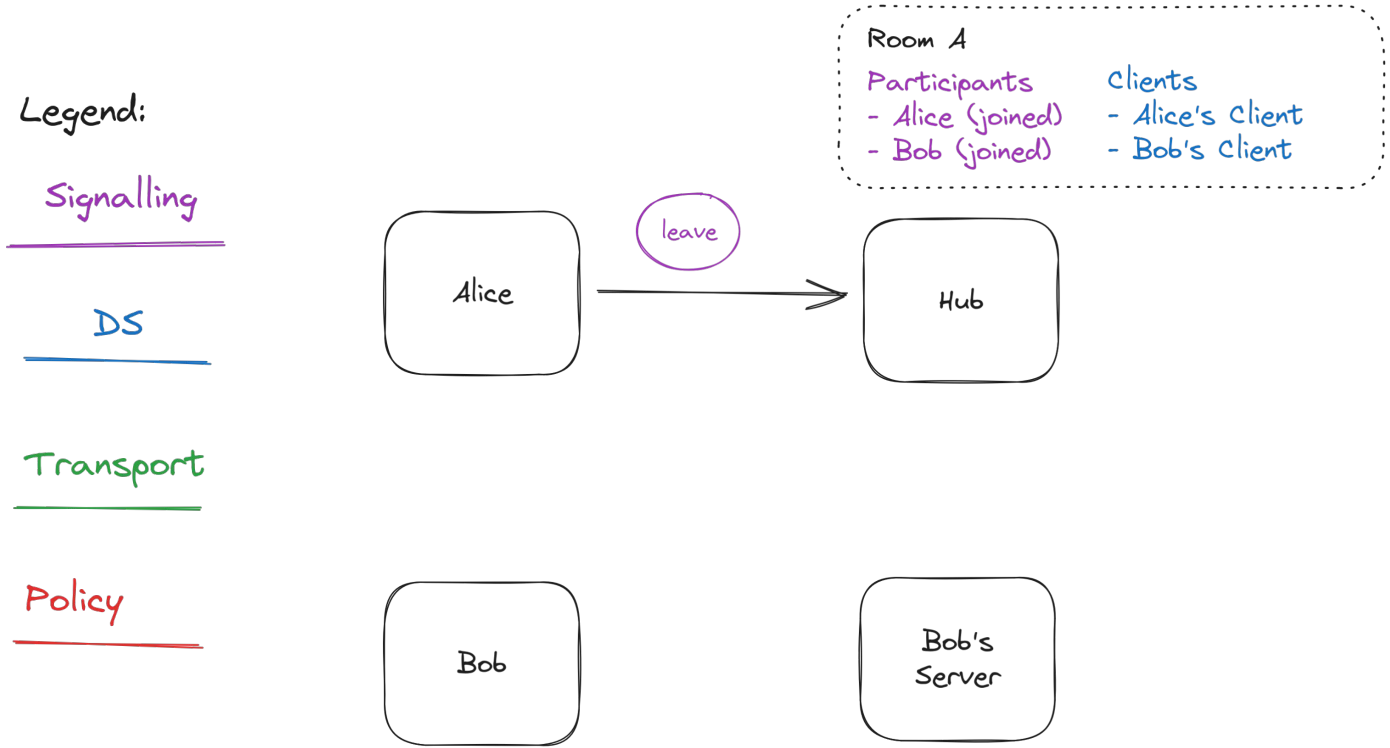
Alice and Bob can converse! 🎉

Scenario 2: Alice wants to leave

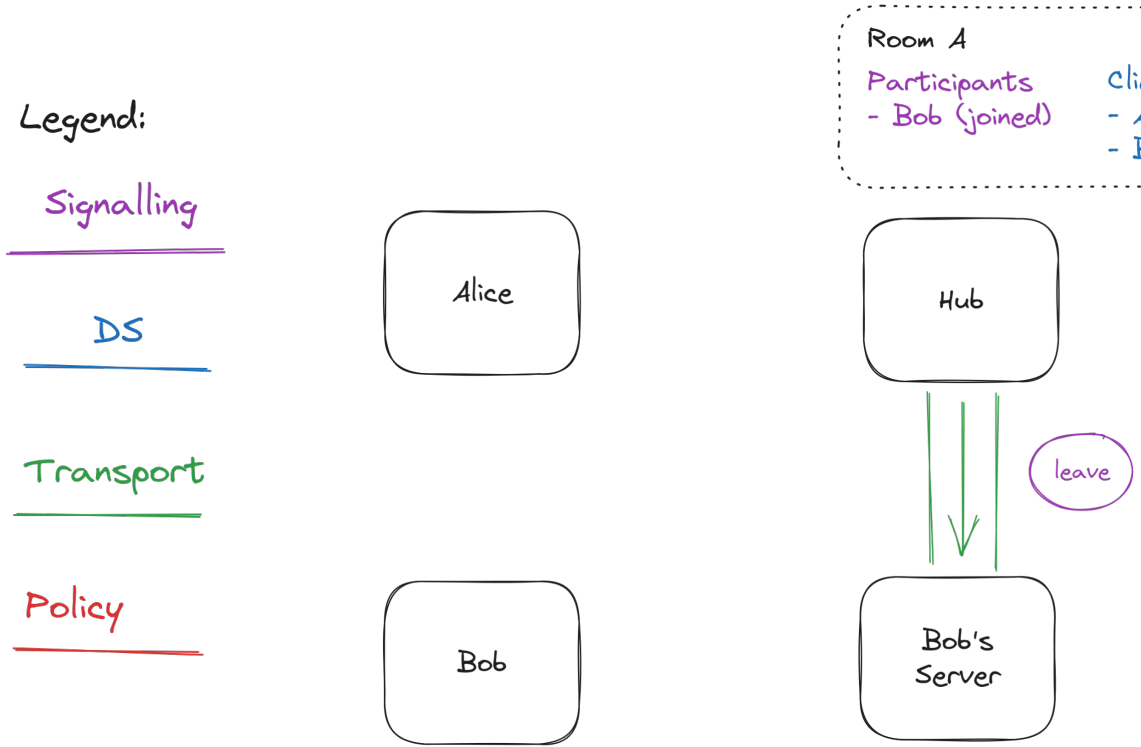
Assumptions

- This scenarios is built on top of the previous scenario
- Alice and Bob share a room, and both have clients in the group
- Alice’s server is the Hub for the room

Alice sends leave event to the Hub

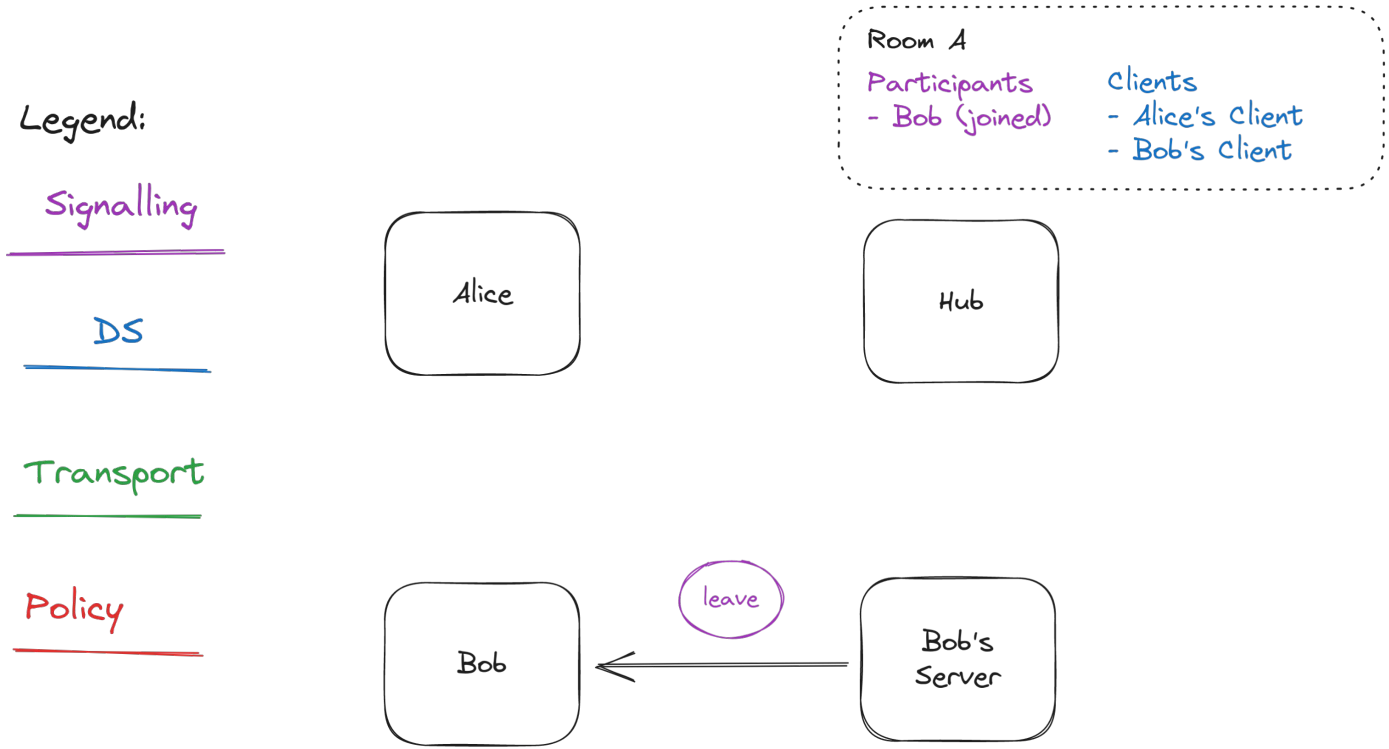


Hub fans out event to Bob's server

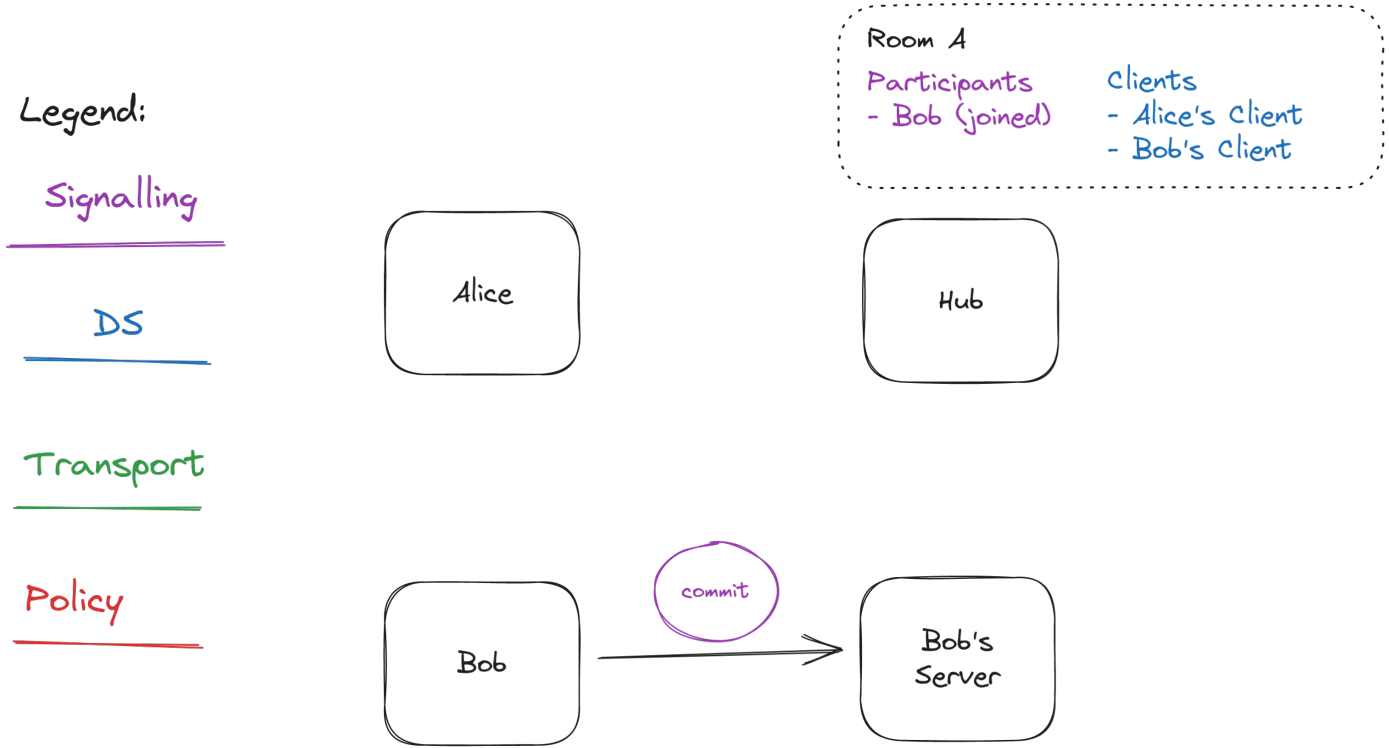


Now that Alice is gone, the Hub requires that Alice's client(s) be removed with the next commit.

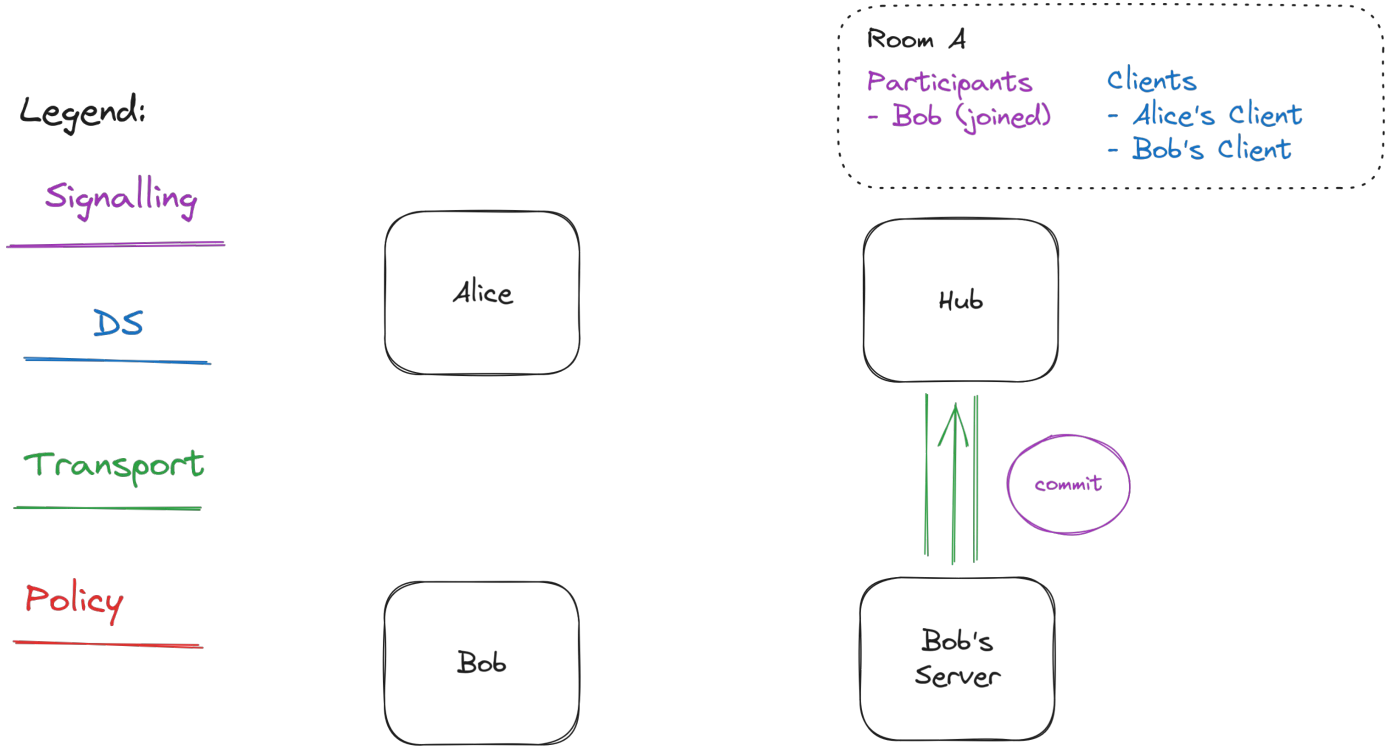
Bob receives signalling event



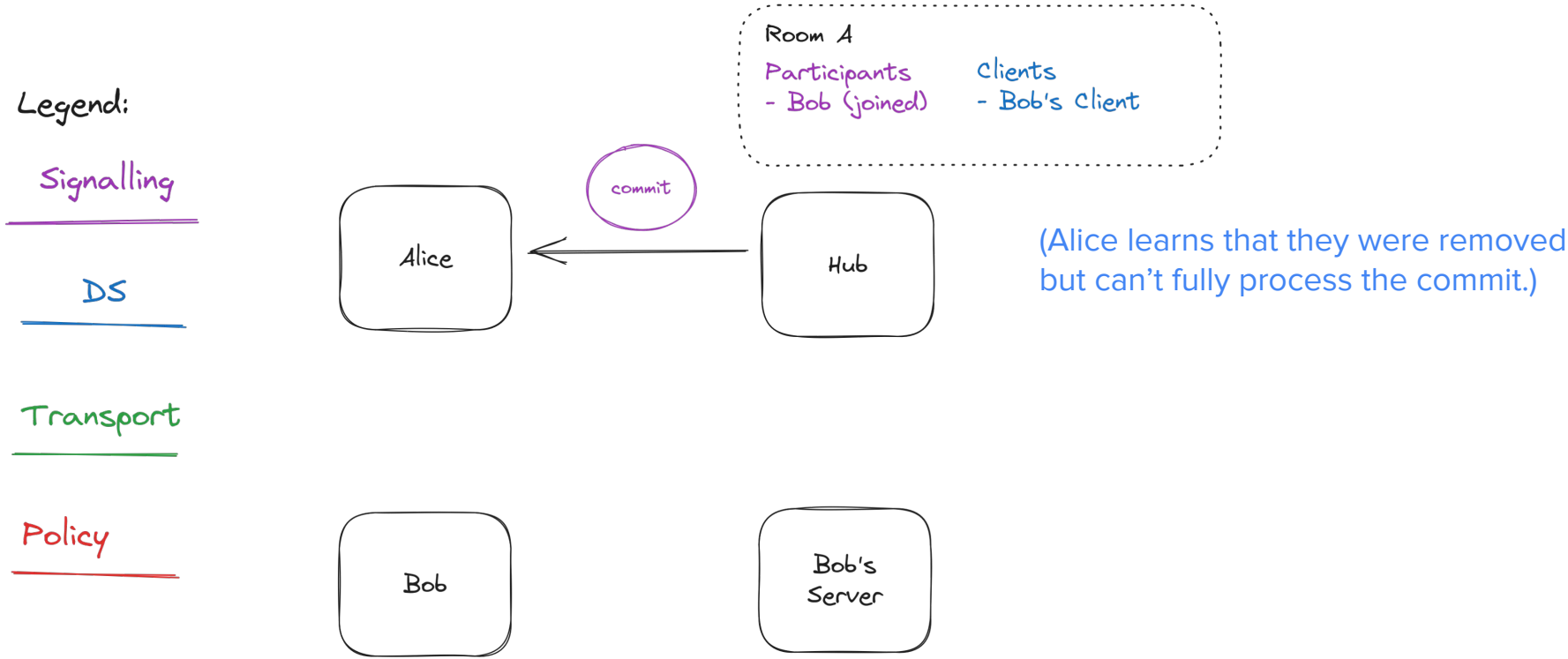
Bob commits to remove Alice's clients



Bob's server forwards commit to Hub



Bob's server forwards commit to Hub



(Alice learns that they were removed but can't fully process the commit.)

~~Hello world~~

Alice is no longer participating in the room or group, and cannot converse with Bob anymore.

⚠️ The Hub server role may need to move to a surviving server in the room.


DISCUSS!