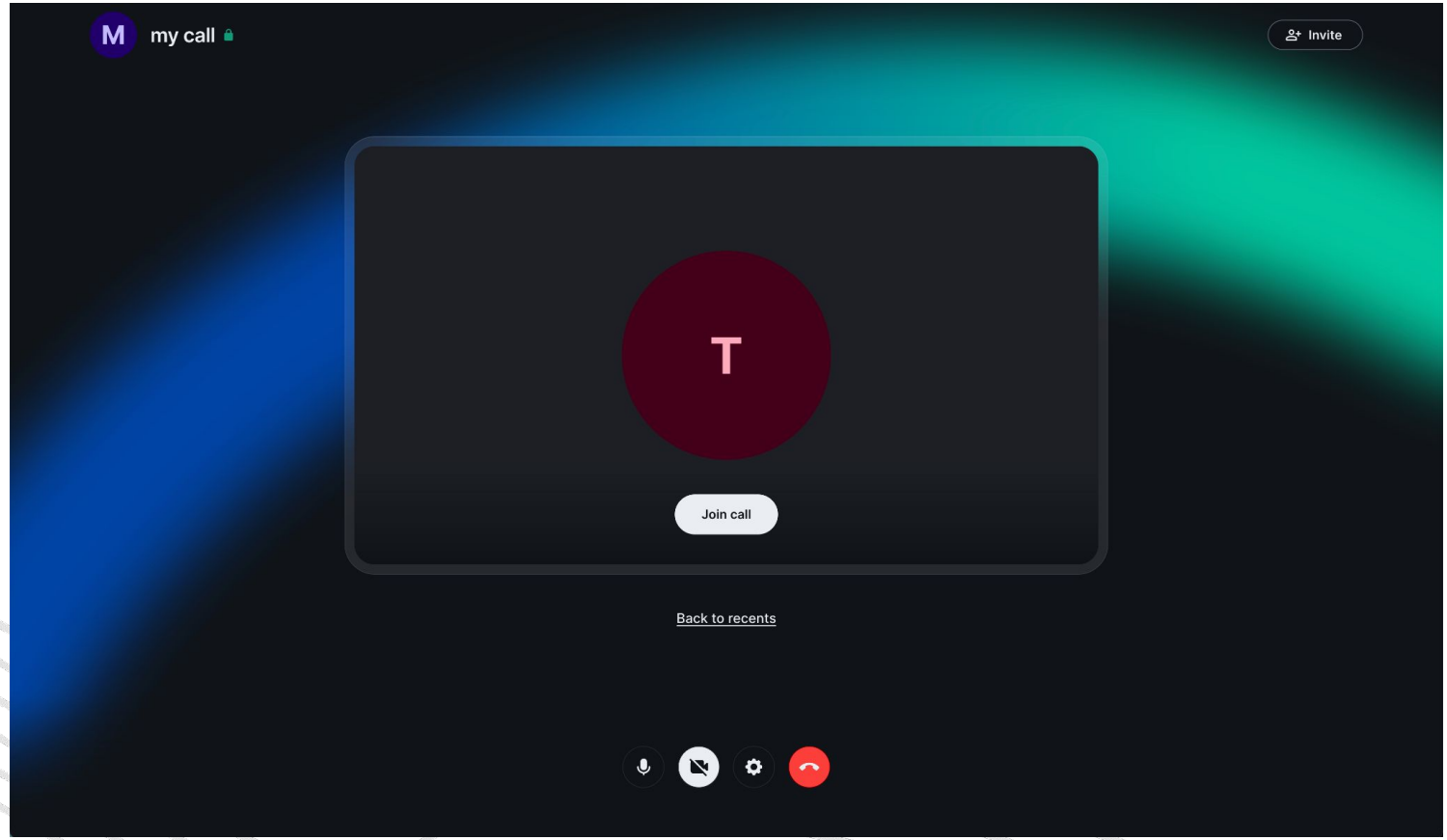
matrix

MatrixRTC: The Future of Matrix Calls

Famedly

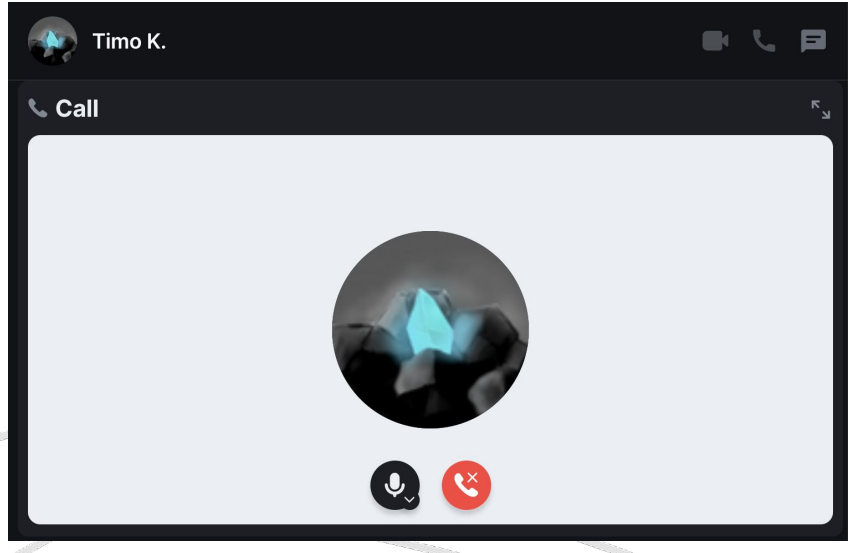
element

Element Call used matrixRTC under the hood for some time now



Why redesign it all?

There are already calls in matrix since ages!



Why rebuild it all?

There are already calls in matrix since ages!

There are issue, but why not fix those problems, why sth. new?

Limitations:

- By design only 1:1
- Very call specific (not designed as an infrastructure extending matrix)
- Signalling over room events (not fundamentally broken but hard to get right)

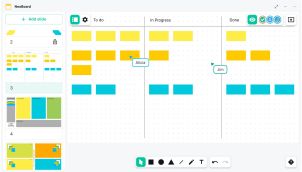
matrix

Making **calls** a great and central part of matrix

Think beyond calls: Expandable system that motivates other projects

third room

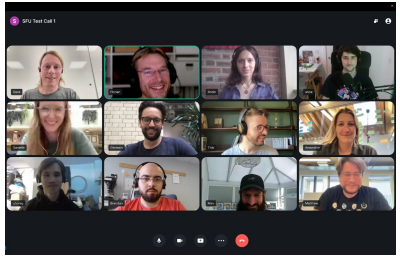
NeoBoard (Nordeck)



Pluggable RTC backend

e.g. Livekit

Large, secure group calls



Support many clients

Famedly

element

fluffychat

+

Others?

(widget support EC)
Or any livekit sdk

Recap | Matrix

What matrix is really good at:

Open Standard

I guess kinda a given at FOSDEM

Encryption

Authentication / Verification

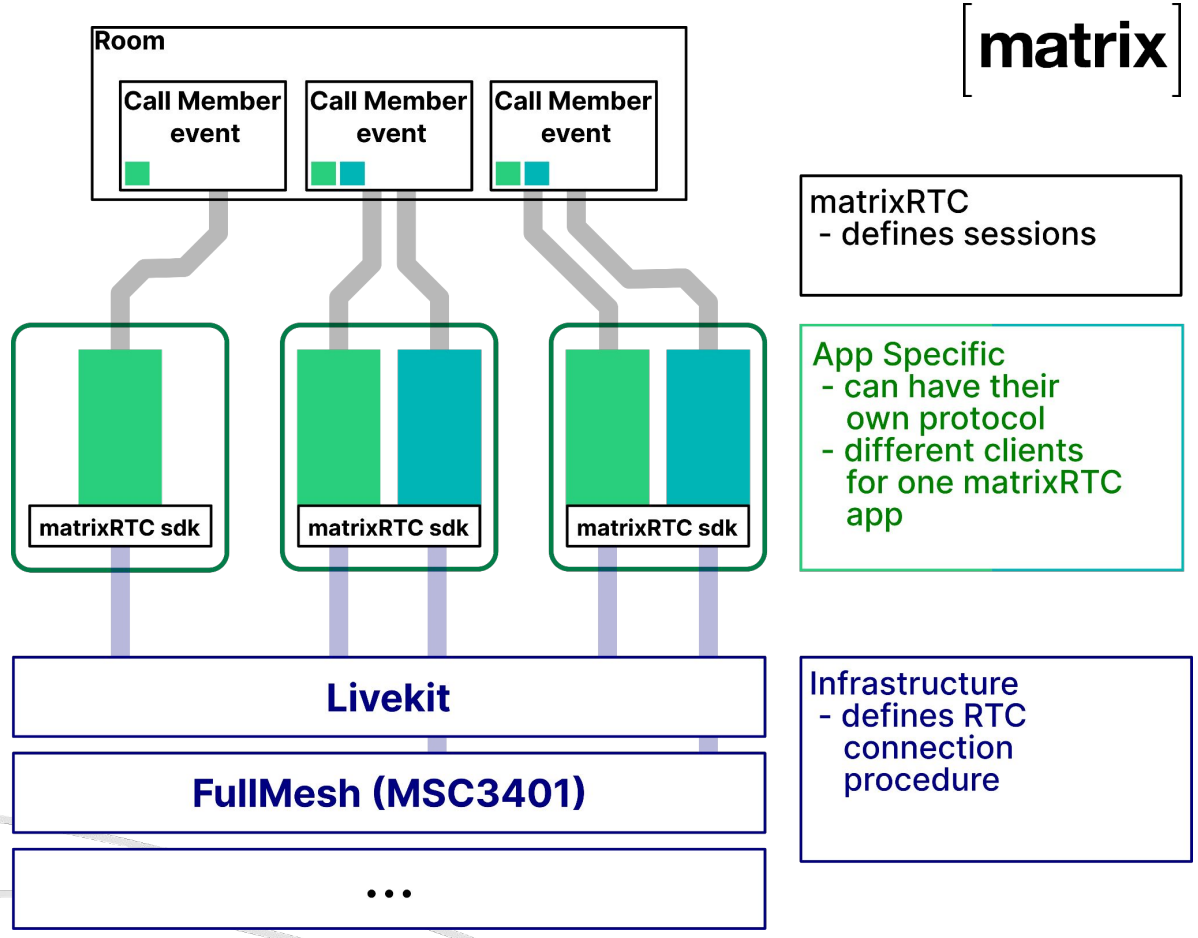
Federated

Persistent

How to use those parts the best way possible for a RTC infrastructure.

The Components of a MatrixRTC Call

- Matrix
- Client (Call app)
- RTC Infrastructure



Call Member Events

Call Member event

```
"content": {
"memberships": [
    {
```

MatrixRTC

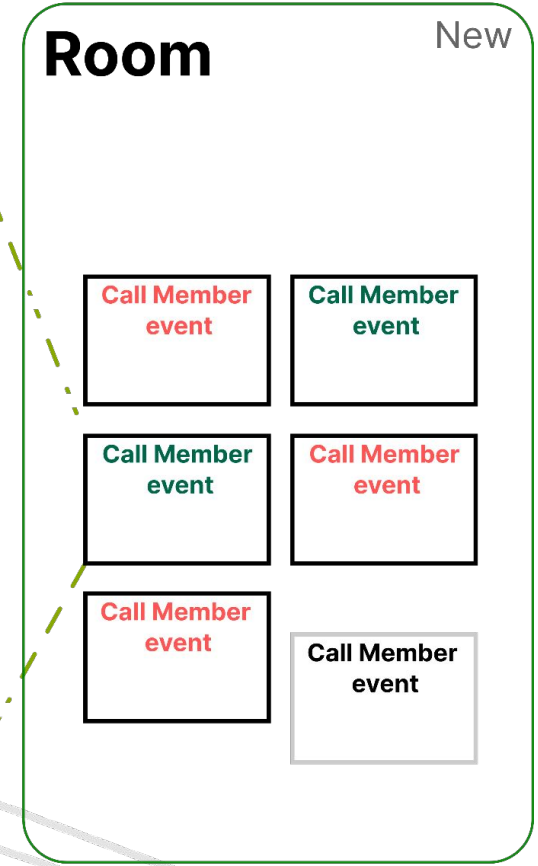
```
"device_id": "DEVICEID123",
"expires": 3600000,
"foci_active": [
    {
    "livekit_alias": "!NXHzTvNOwsFiZaAvTT:matrix.org",
    "livekit_service_url": "https://livekit-jwt.call.element.dev",
    "type": "livekit"
    }
],
"application": "m.call",
```

Application Specific
Fields defined by `m.call`

```
"call_id": "",
"scope": "m.room",

"linked_event":"$Azt5QD7kRbOq19IyqWoNUtmk6ulGsUSCgKLT6Bvzs-Y",
"other_shared_or_individual_data":"100,200"
```

```
    }
]
}
"state_key": "@user:matrix.org"
```



Call History

Custom event summarizing a call

- Who creates the event?
- It would be redundant data!
  - How to guarantee its in sync with the call.member.events?

Call History

The call.member.events can be parsed as **Join** and **Leave** events:

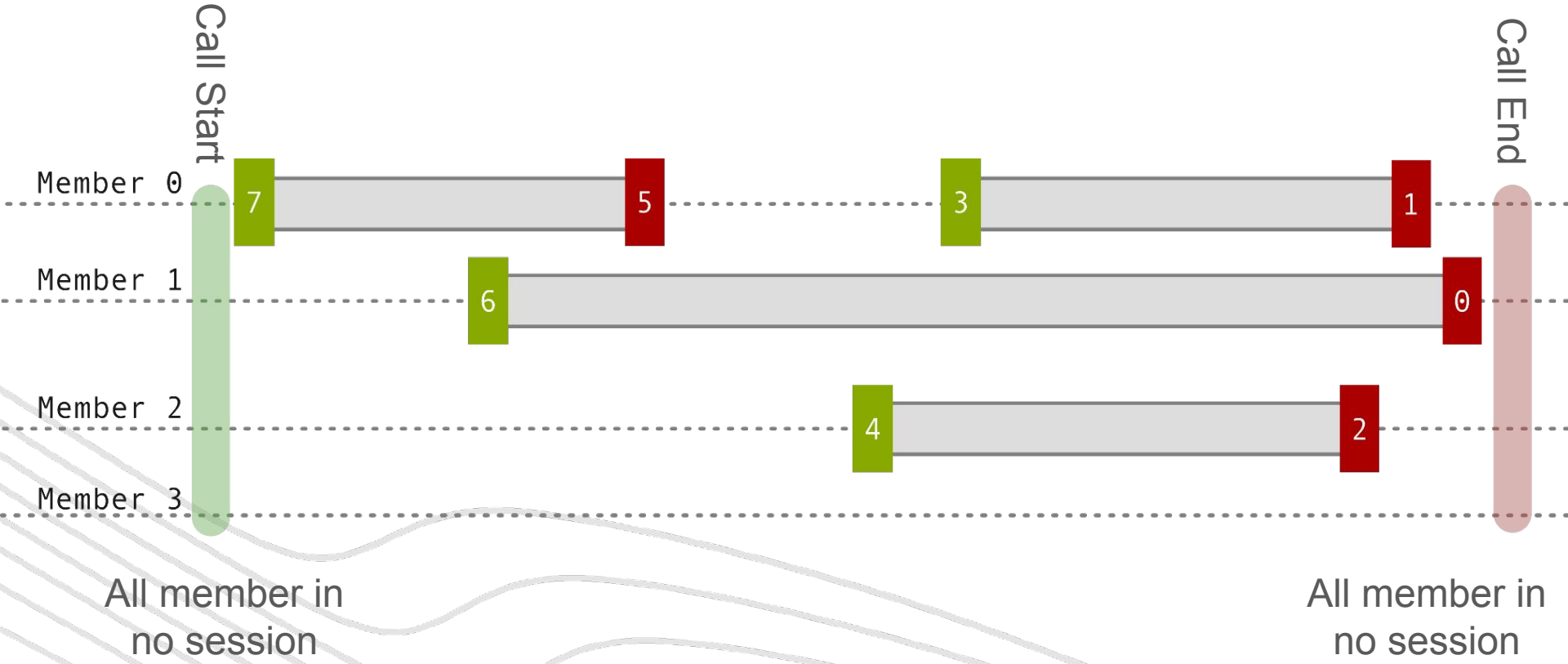
```
"content": {
    "memberships": [
        {
        "device_id": "1",
        "foci_active": [],
        "application": "...",
        ...
        }
    ]
},
"unsigned": {
    "prev_content": {
        "memberships": []
    },
}
```

Join

```
"content": {
    "memberships": []
},
"unsigned": {
    "prev_content": {
        "memberships": [
            {
            "device_id": "1",
            "foci_active": [],
            "application": "...",
            ...
            }
        ]
    },
}
```

Leave

Call History



Give overview about MSCs

- New global MatrixRTC MSC
- Implementing MatrixRTC using the mesh backend MSC
- Implementing MatrixRTC using the livekit backend MSC
- Possibly have the HS setup the pc for you somehow?
-

Existing MSCs

Calls

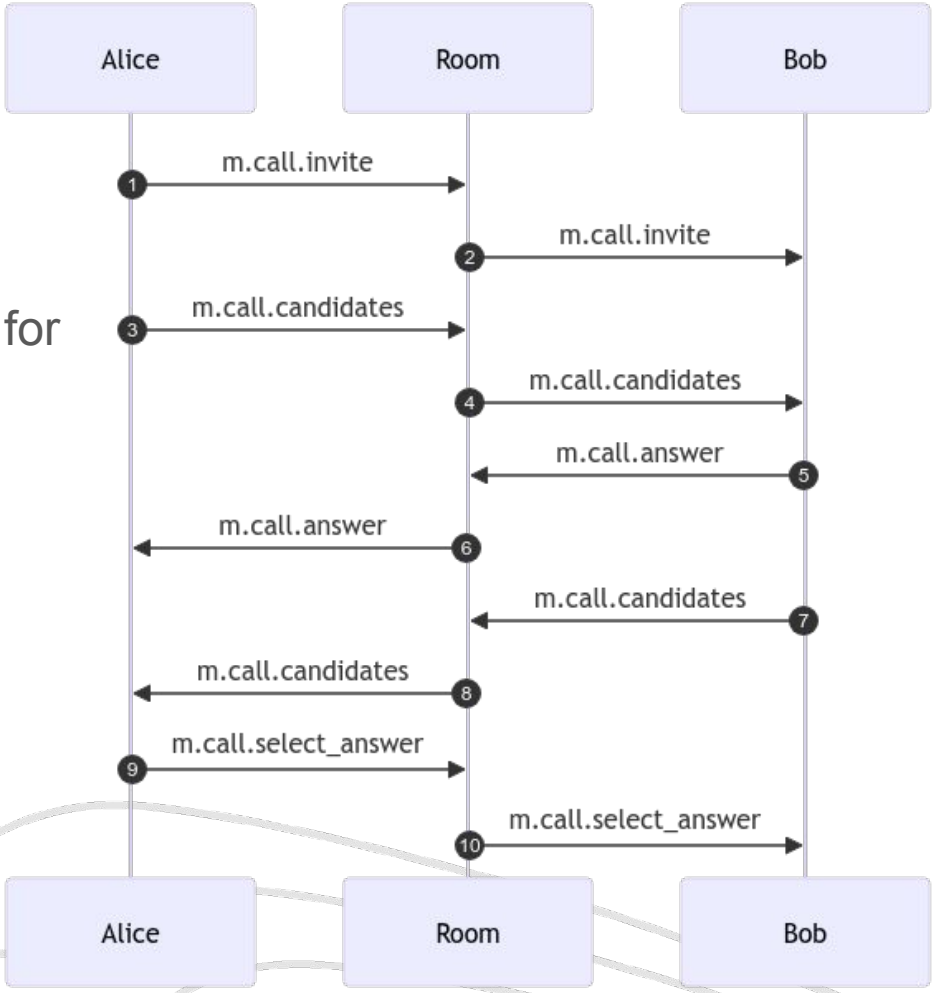
- Native webrtc
- Most platforms have good webrtc libraries already
- m.call events (invite, candidates, answer, hangup)
- MSC2746 added events reject, select_answer, negotiate, glare handling, hold/unhold, DTMF
- MSC3077 added stream metadata and ability to differentiate streams
- MSC3291 added the ability to mute and unmute streams

FYI

- Some lesser known facts:
  - MSC2747 and MSC3086 adds transferring calls and asserted identities
  - MSC3635 adds the ability to send early media
  - MSC4075 adds the ability to configure notifications for MatrixRTC
- Use room events for the signalling

matrix

All p2p calls use normal room events for signalling.

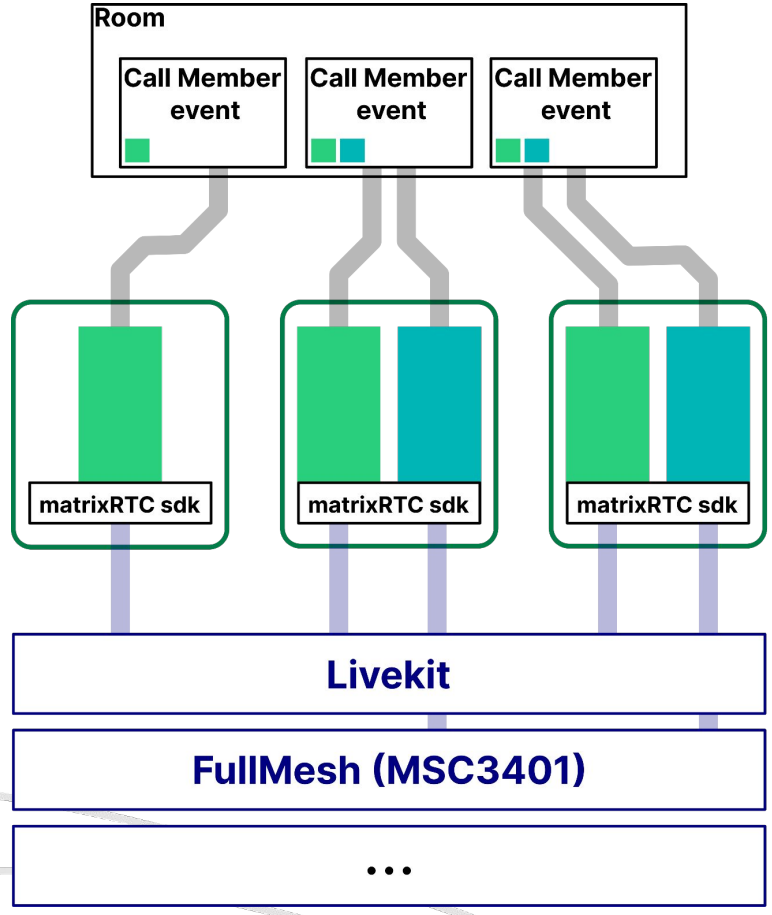


Implementing

RTC Infrastructure

The Components of a MatrixRTC Call

- Matrix
- Client
- RTC Infrastructure



matrixRTC
- defines sessions

App Specific
- can have their own protocol
- different clients for one matrixRTC app

Infrastructure
- defines RTC connection procedure

MSC3401 events

- m.call events

```
"content": {
  "m.intent": "m.prompt",
  "m.type": "m.video",
  "m.terminated": "call_ended"
},
```

- m.call.member events (old)

```
"m.call_id": "1703868462968EbLoPCKLD3u6VgLO",
"m.devices": [
  {
    "device_id": "SFAETNNELQ",
    "expires_ts": 1703868524905,
    "feeds": [
      {
        "purpose": "m.usermedia"
      }
    ],
"session_id": "oniVyQCHzgXo"
```

MSC3401 and its extension

- Do you really need m.call events though? (only hold the termination state)
- Just use m.call.member events
- Ability to pass in custom backends, scopes, applications
- Use to-device events for signalling
- MSC4018 adds ability to let the homeserver/SFU manage the member events
- Matrix now only helps with the call state and encryption for the sframe

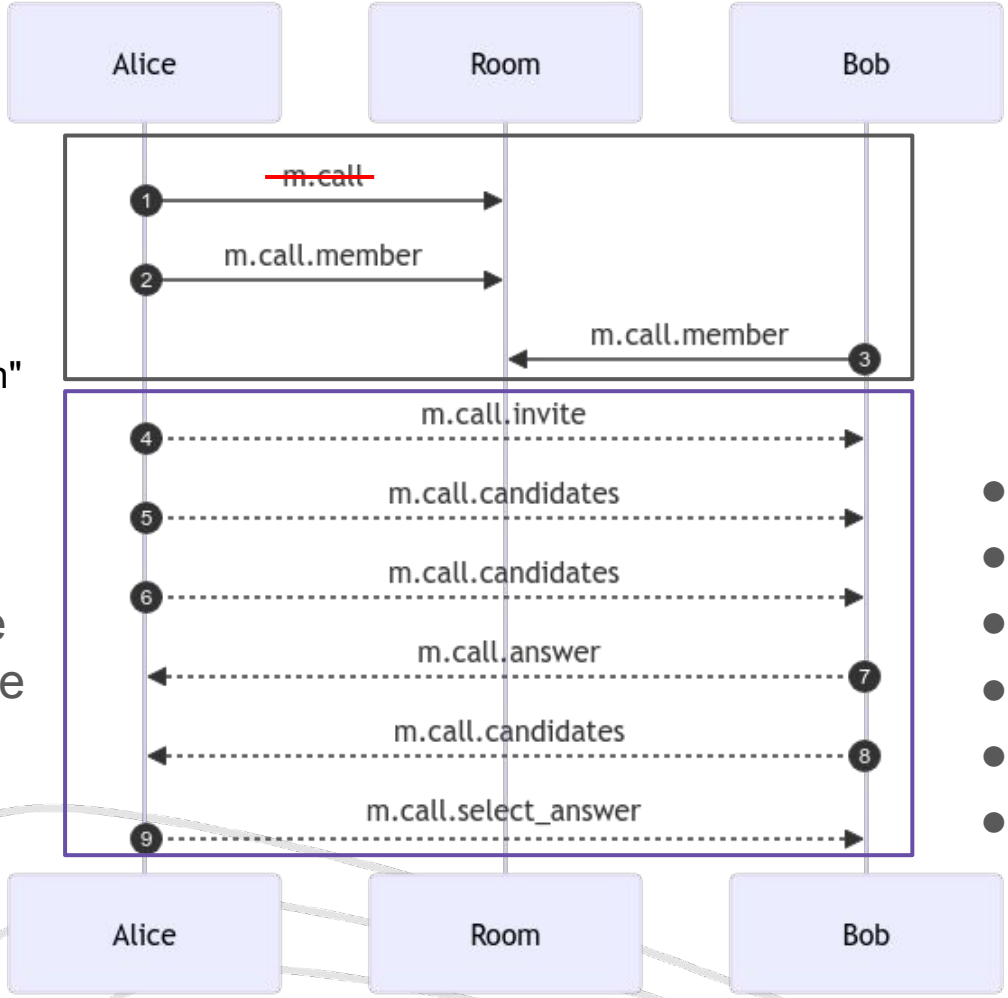
Full Mesh

matrix

MatrixRTC

```
foci_active: {
  "type": "mesh"
}
```

Infrastructure over to-device events



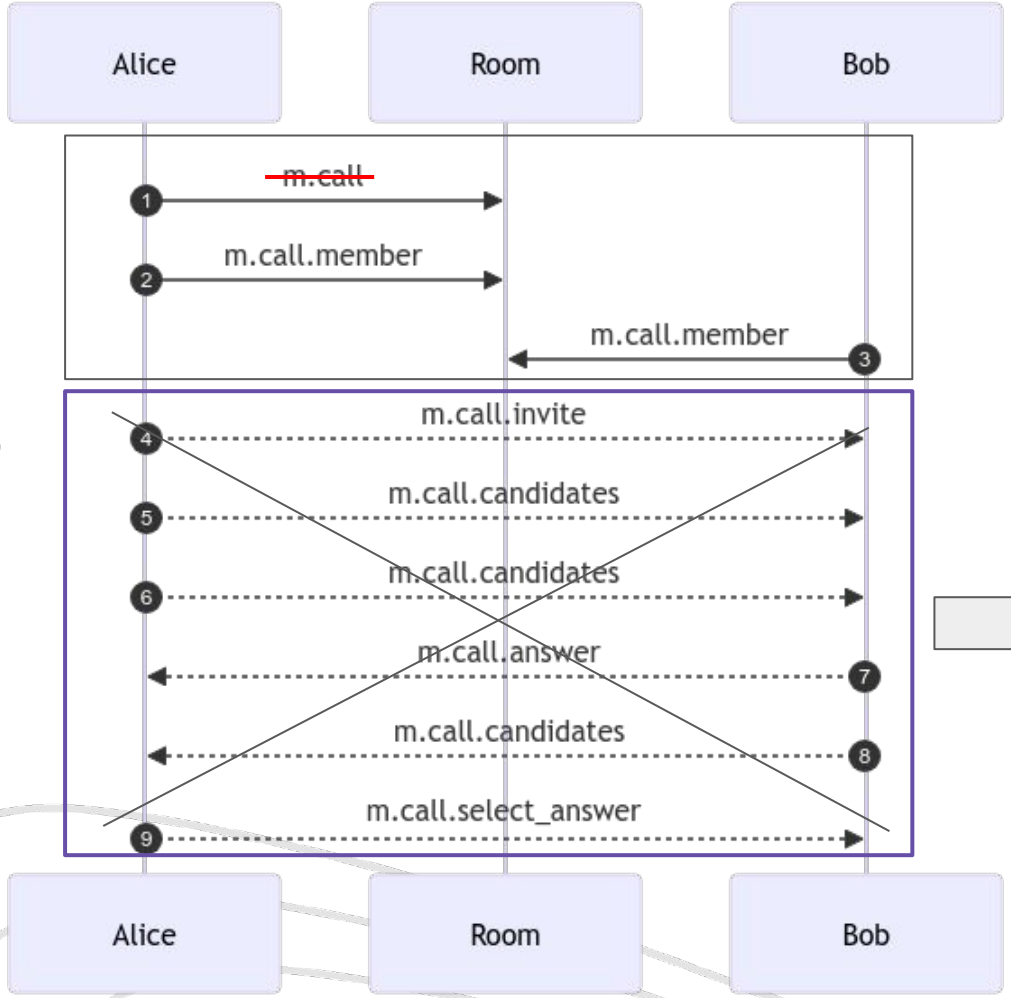
- MSC2746
- MSC2747
- MSC3401
- MSC3077
- MSC3291
- MSC3635

Livekit

MatrixRTC

```
foci_active: {
  "livekit_alias": "uwu",
  "livekit_service_url": "nyaa",
  "type": "livekit"
}
```

Infrastructure over websockts



matrix

Livekit

- JWT service:
  - Users need to be authenticated before getting access to the SFU
  - Currently the SDKs use a combination of the OpenID API and the federation userInfo endpoint to confirm if you are actually a user on the HS
  - In future, you would ideally also want to confirm if the user is even in the room and if they can join the group call.
  - The jwt sends you the livekit alias and the jwt token to authenticate with
- Uses websockets for webrtc signalling

Fancy stats

- Livekit docs: A `c2-standard-16` 16 core google VM should let you have video calls with around 150 members
- Personal server testing: Hetzner `CAX21` 4 ARM cores lets you have calls with around 70 participants

Implementing

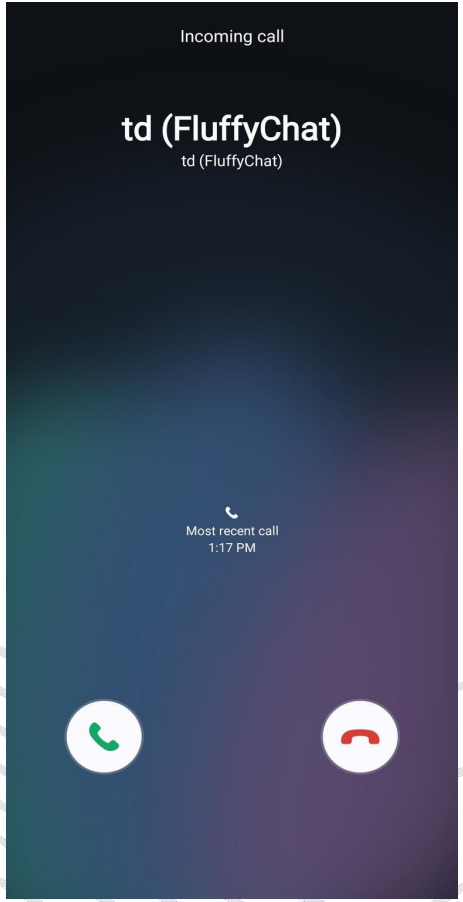
Ringing

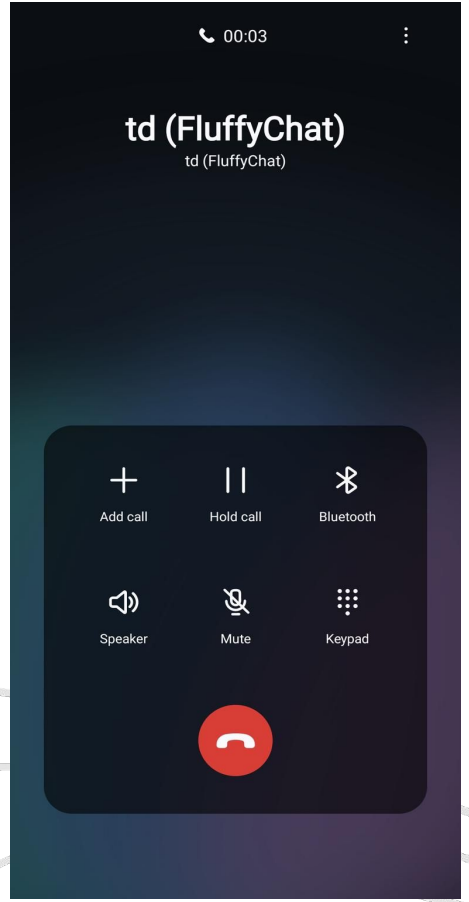
Ringing

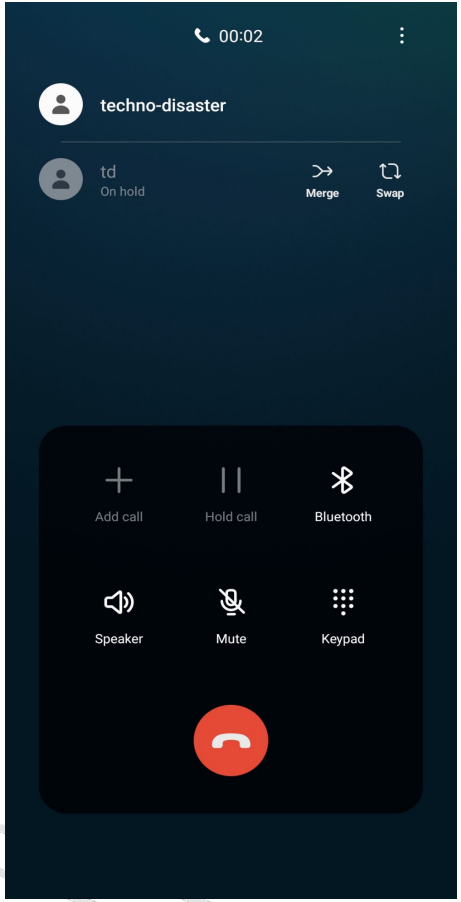
- Started as a GSoC’22 Project
- Cases:
  - Foreground
  - Background
  - Terminated
- Pivoted 3 times to get a stable way to handle ringing

Try 1: What we could have had! (ConnectionService API)

matrix







matrix

Try 2:

- Just hack it, use showOnLockScreen, appear on top and a custom flutter screen. Does not work on terminated apps.

matrix

Try 2:

- Just hack it, use showOnLockScreen, appear on top and a custom flutter screen. Does not work on terminated apps.

Try 3:

- Oh, push notifications, they already start the right flutter bits to decrypt stuff, yes we can just flutter engine started by those push plugin job workers!
- iOS still needs work and afaik the only possible way to do this there is using CallKit.

MSC4075 for call notifications

- Using intentional mentions and push rules to do rings instead of call invites
- Event type `m.call.notify` has metadata and instructs client how to ring on stuff
- Completely independent of the calls, can be used to signal users about calls even before the call.

```
{
  "content": {
        "application": "m.call | m.other_session_type…",
        "m.mentions": {"user_ids": [], "room": true | false},
        "notify_type": "ring | notification",
        // Application specific data,
        // optional fields to disambiguate which session
        // this notify event belongs to:
        // for application = "m.call":
        "call_id": "some_id",
  }
}
```

Implementing

SFrame key sharing

SFrames

- SFUs need another lock on top of SRTP/DTLS
- We use SFrames, most of the browsers supp
- Could do key sharing over room events or to-device events

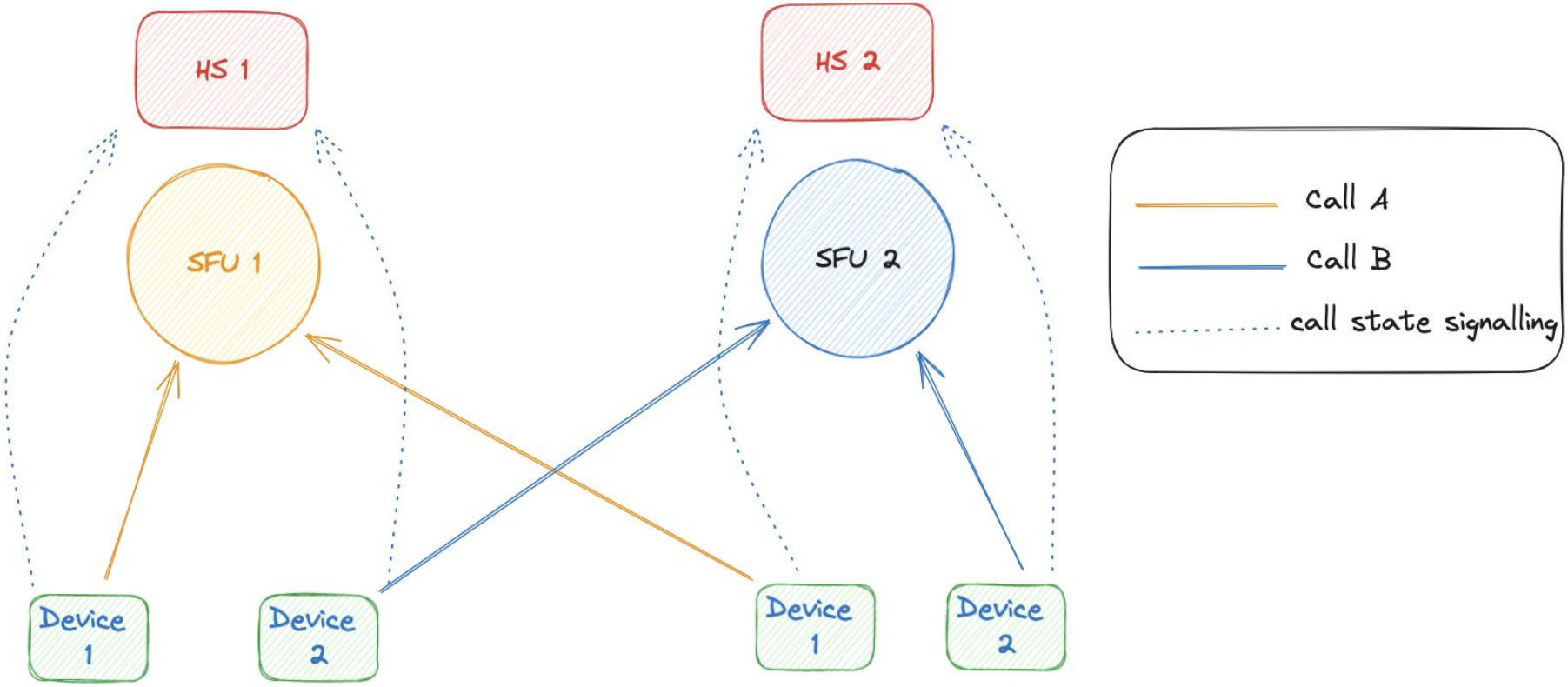
SFrames key sharing contd…

- Key sharing happens on leave/join
- But we also implement a pull/push feature to request and send keys.
  - call.encryption_keys event for key details
  - call.encryption_keys.request event for requesting a key (remember to do the necessary checks before sending a key)

Implementing

Cascading

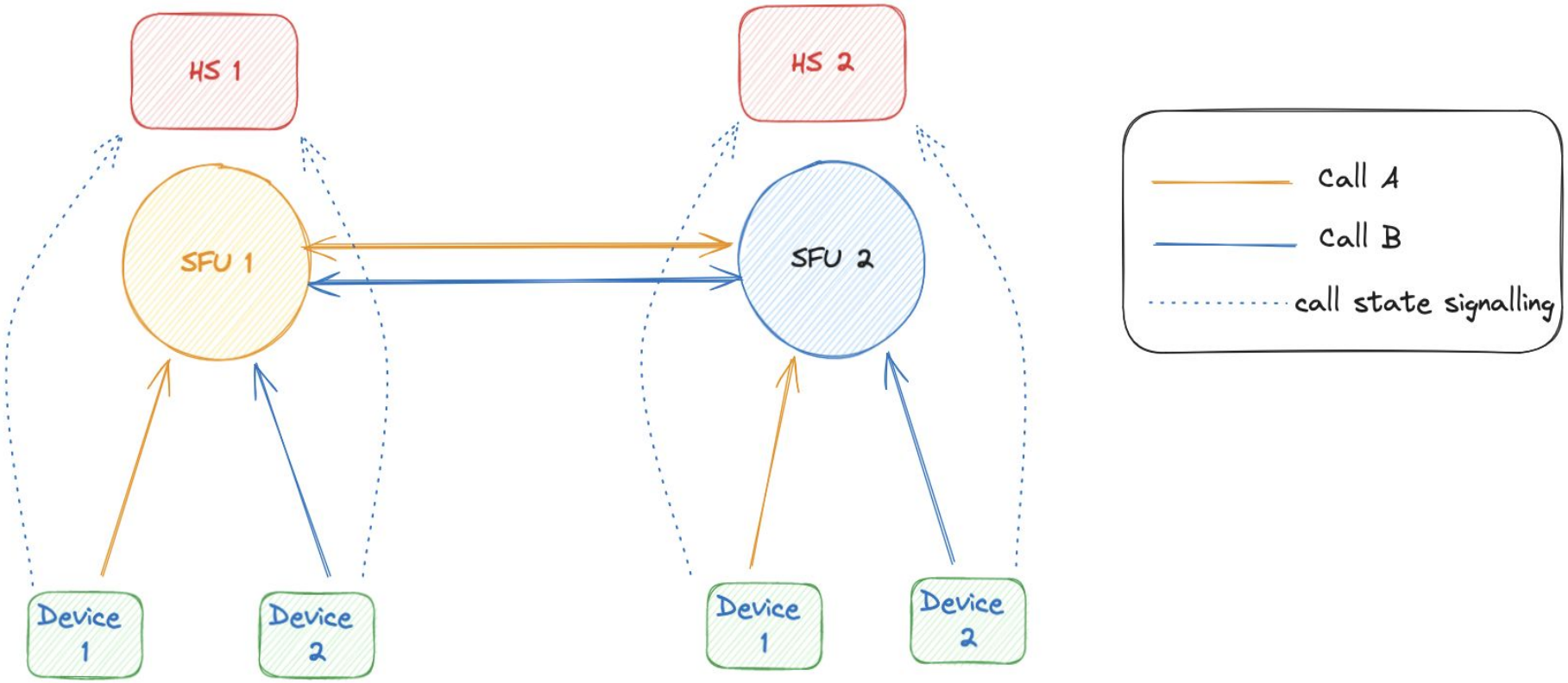
matrix


HS 1
HS 2
SFU 1
SFU 2
Device 1
Device 2
Device 1
Device 2
Call A
Call B
call state signalling

Cascading

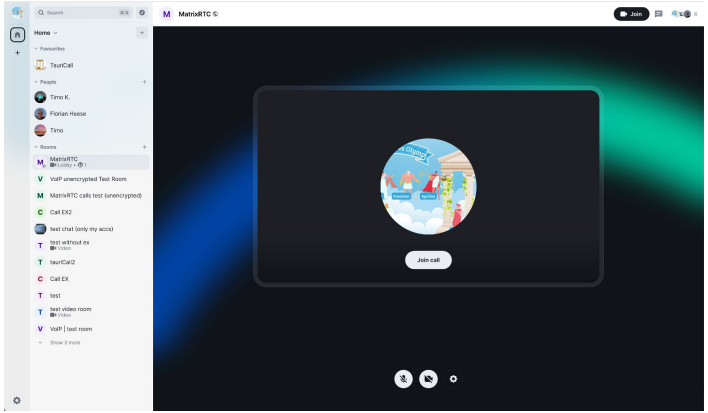
- As of today livekit calls are “in vibe” federated.
- You can have multiple calls in the same room each with a different SFU
- The livekit SFUs are currently mandate all the participants in a call to be on the same SFU
- In future each participant could bring their own cascading SFU and everything would just work (maybe even cross SFU!), f.ex:
  - Device 1: cascading sfu 1
  - Device 2: cascading sfu 2
  - Device 3: cascading sfu 3
- The jwt service could also finer control. Currently the server only validates that the oidc token is from a valid user

matrix

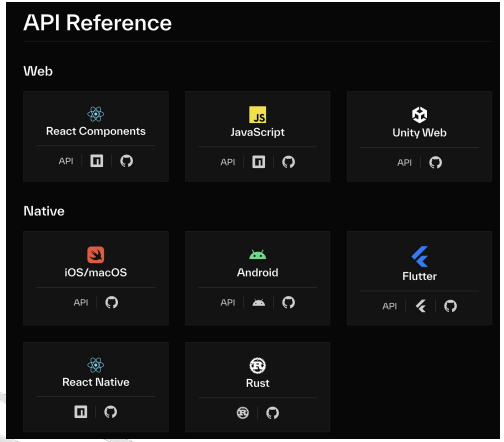

HS 1
HS 2
SFU 1
SFU 2
Device 1
Device 2
Device 1
Device 2
Call A
Call B
call state signalling

How you can implement this

- Widget (Matroska ElementCall)
  - Any client which supports the widget api can get MatrixRTC calls for free
  - The rust SDK and the react sdk already support widgets

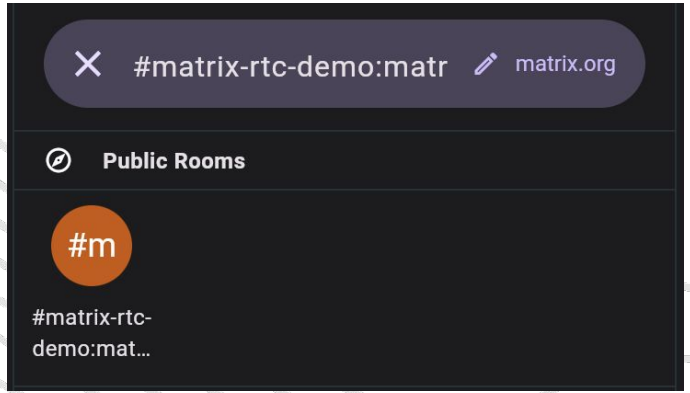


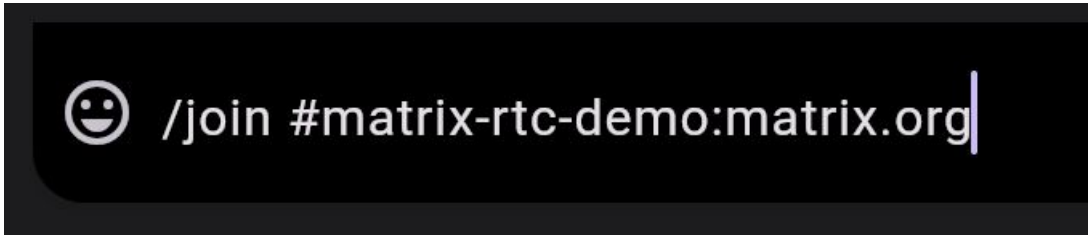
- Native livekit sdk f.ex the dart client sdk

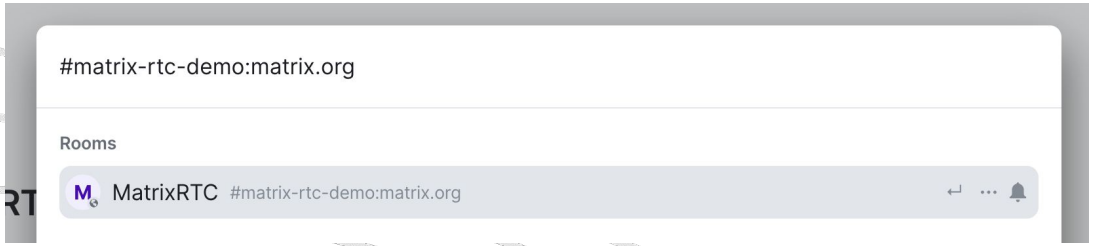


DEMOS

Showing the two implementation paths







- **Changes for fosdem interop**
  - create_ts + expires + membershipID -> expire_ts. (Mandatory) (DONE)
  - call_id should be empty, if m.room.
  - backend should be renamed to foci_active. (Mandatory)
  - we wont demo an encrypted call.
  - Use oldest member sfu (low prio for the demo)
- **Topics:**
  - What we had upto now (TD)
  - The vision of matrixRTC (google docs/whiteboard/real-time games/security cam,webcam) (Timo)
  - Livekit as the backend (full mesh as of 3401) (Timo)
  - Timeline Rendering, but keep it short (Timo)
    - Compute based on Member event history.
    - Let the SFU create timeline events.
  - Ringing Spec (Timo || TD)
  - Ringing Impl (TD)
  - Encryption. (TD)
    - to-device events vs room events.
  - Cascading (TD)
    - Start with stricter sfu access control
  - Implementation Strategies:
    - Widget
      - really easy to adopt for other clients -> webview shows calling view of a room (multi platform thanks to web stack)
      - More borders for device permissions
    - Native (example: Flutter)
      - Can be used in other flutter apps. (multi platfrom thanks to flutter)
  - Current challenges (Timo)
    - Historic session data
    - Member event permissions (we need one state event for all devices)
    - Member event update always comes from clients - crashes don’t result in dicsonncts (HS or SFU membership control)
- **Demos:**
  - Show call interoperability between element call and fluffychat/famedly. (both)
  - Calling between devices of the same user. (switch from device to device (continuity)). (TD)
  - Ringing (foreground, background, terminated). (TD)