Trixnity

One Matrix SDK for (almost) everything written in Kotlin

Who am I?

- **@benedict:imbitbu.de**
- **Privacy and IT security enthusiast**
- **4 years of development and admin experience with Matrix**
- **Maintainer of Trixnity**
- **Own Startup: connect2x**
  - Developing Timmy, a TI-Messenger for the medical sector in Germany

What is Trixnity?

- Matrix SDK
- For Client, Bot, Appservice and Server development
- Multiplatform: JVM, JS, Native
- Written in Kotlin
- High test coverage (including integration tests)
- Licence: Apache 2
- Repository: https://gitlab.com/trixnity/trixnity

Why another SDK?

- Back in January 2020 there were few multiplatform SDKs to choose from
- Mostly no strict typing of events and REST endpoints
- No or difficult extensibility (own event types)
- No generic application purpose (Client, Server, etc.)

Why Kotlin?

- Statically typed programming language
- Compiles to JVM, JS and Native → no bindings needed
- Common code base between platforms
- Platform-specific implementations possible
  - Access to platform-specific APIs
  - Use of platform-specific libraries (Maven, npm, C)
- Allows you to create your own DSLs

Architecture

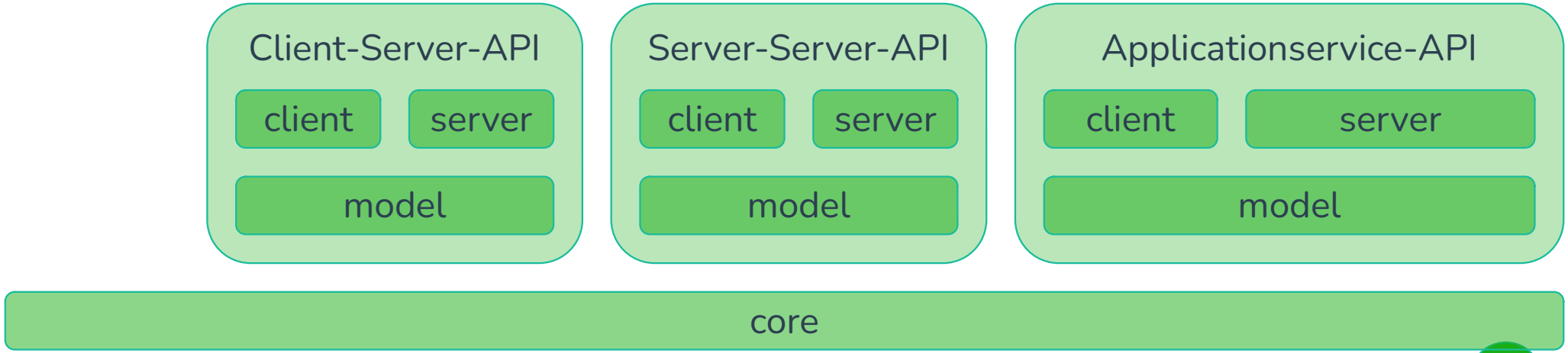
core

Custom Events

```
interface EventContentSerializerMappings {
    val message: Set<SerializerMapping<out MessageEventContent>>
    val state: Set<SerializerMapping<out StateEventContent>>
    val ephemeral: Set<SerializerMapping<out EphemeralEventContent>>
    val ephemeralDataUnit: Set<SerializerMapping<out EphemeralDataUnitContent>>
    val toDevice: Set<SerializerMapping<out ToDeviceEventContent>>
    val globalAccountData: Set<SerializerMapping<out GlobalAccountDataEventContent>>
    val roomAccountData: Set<SerializerMapping<out RoomAccountDataEventContent>>
}

object CustomEventContentSerializerMappings : BaseEventContentSerializerMappings {
    override val message: Set<SerializerMapping<out MessageEventContent>> = setOf(
        of<CatEventContent>("m.room.cat"),
    )
}
```

Architecture



Defining Matrix Endpoints

```
@Serializable
@Resource("/_matrix/client/v3/rooms/{roomId}/leave")
@HttpMethod(POST)
data class LeaveRoom(
    @SerialName("roomId") val roomId: RoomId,
    @SerialName("user_id") val asUserId: UserId? = null
) : MatrixEndpoint<LeaveRoom.Request, Unit> {
    @Serializable
    data class Request(
        @SerialName("reason") val reason: String?
    )
}
```

Using Matrix Endpoints

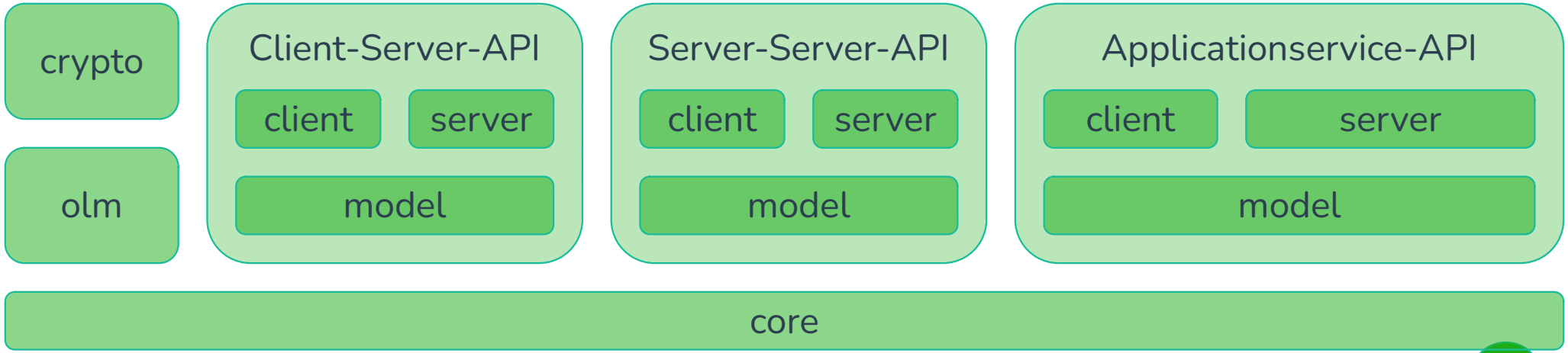
```
// as client (low level)
matrixApiClient.request(LeaveRoom(roomId), LeaveRoom.Request(reason))

// as client (high level)
roomsApiClient.leaveRoom(roomId, reason)

// as server (low level)
matrixEndpoint<LeaveRoom, LeaveRoom.Request, Unit>(json, contentMappings){ context ->
    // handle request
}

// as server (high level)
class RoomsApiHandlerImpl : RoomsApiHandler {
    override suspend fun leaveRoom(context: MatrixEndpointContext<LeaveRoom, LeaveRoom.Request, Unit>){
        // handle request
    }
}
```

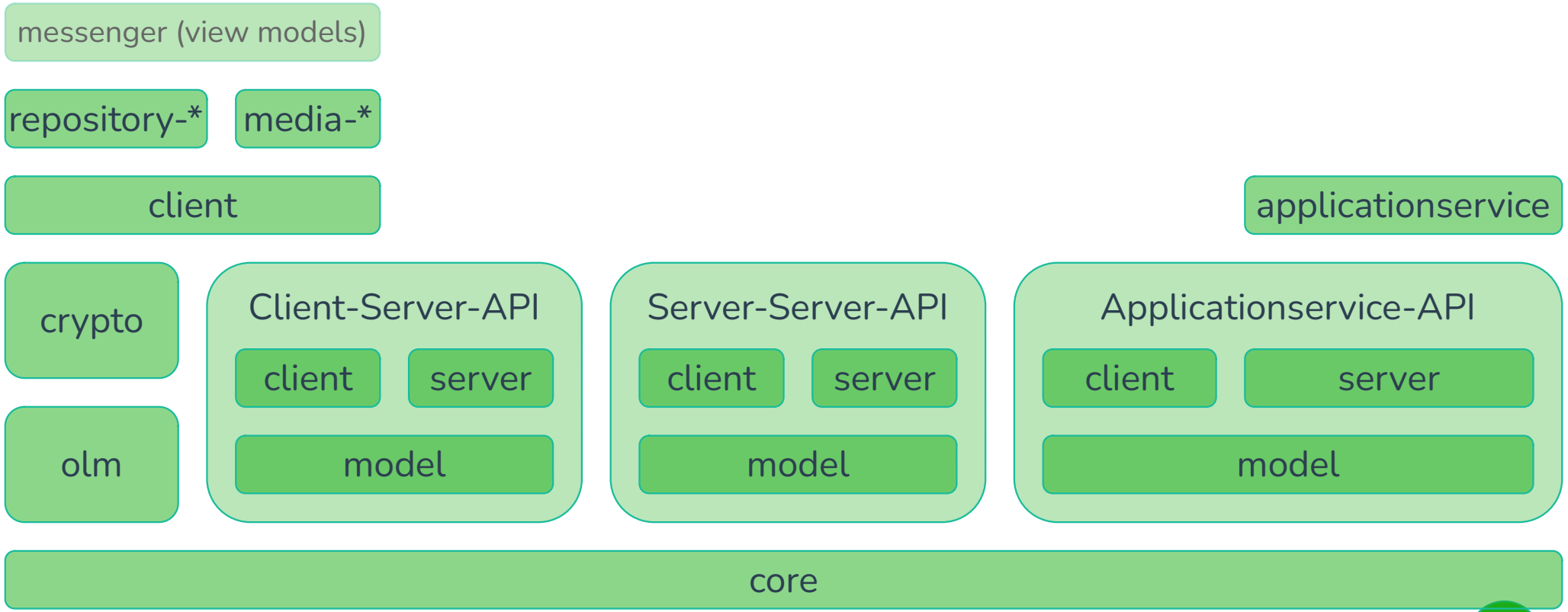
Architecture



End-To-End-Encryption

- **trixnity-olm:**
  - Libolm Wrapper for Kotlin JVM (via JNA), JS (via WASM) and Native (via C-Interop)
  - With packaged libolm binaries
  - Planed: migration to vodozemac
    (see also: https://gitlab.com/trixnity/uniffi-kotlin-multiplatform-bindings)
- **trixnity-crypto**
  - Key Management
  - Encryption and decryption of Events
  - Planed: migrate all crypto related stuff out of trixnity-client

Architecture



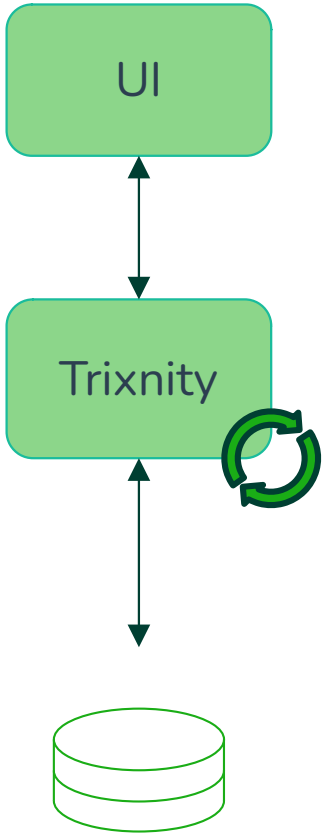
Client

- Contains most features from Matrix v1.5
- Exchangeable database
- Exchangeable media store
- extremely fast reactive cache on top of the database
- async transactions
- E2E including verification, cross signing, key backup, etc.
- Everything is reactive: rooms, timelines, users, outbox and more
- Notifications
- Thumbnail generation
- Redactions and relations
- ...

Currently implemented Media-Store-Wrappers

- Filesystem via okio for all targets (except Browsers)
- IndexedDB for JS (browser only)
- In memory for all targets (recommended for testing only)

How I accidentially created a reactive Cache



The challenge:

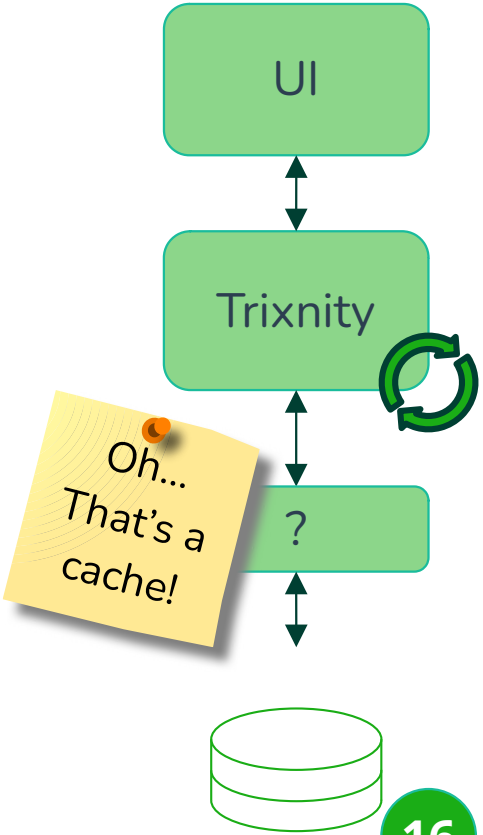
- UI should have access to reactive data

The problem:

- Exchangeable database vs. database with listeners

The solution:

- An intermediate layer based on Kotlin Flows
- Read values from database
- Write changed values into database
- Keep vaues there as long as they are used (+delay)



Everything is reactive

```
matrixClient.user.getAll(roomId)                          // Flow<Set<RoomUser>?>
matrixClient.user.canInviteUser(userId, roomId)           // Flow<Boolean>
matrixClient.user.getAccountData<DirectEventContent>()    // Flow<DirectEventContent>
matrixClient.room.getTimelineEvent(eventId)               // Flow<TimelineEvent?>
matrixCLient.room.getTimelineEvents(startFromEventId)     // Flow<Flow<TimelineEvent>>
matrixClient.verification.getSelfVerificationMethods()    // Flow<SelfVerificationMethods>
matrixClient.notifications.getNotifications()             // Flow<Notification>
matrixClient.key.getTrustLevel(userId)                    // Flow<UserTrustLevel>
...
```

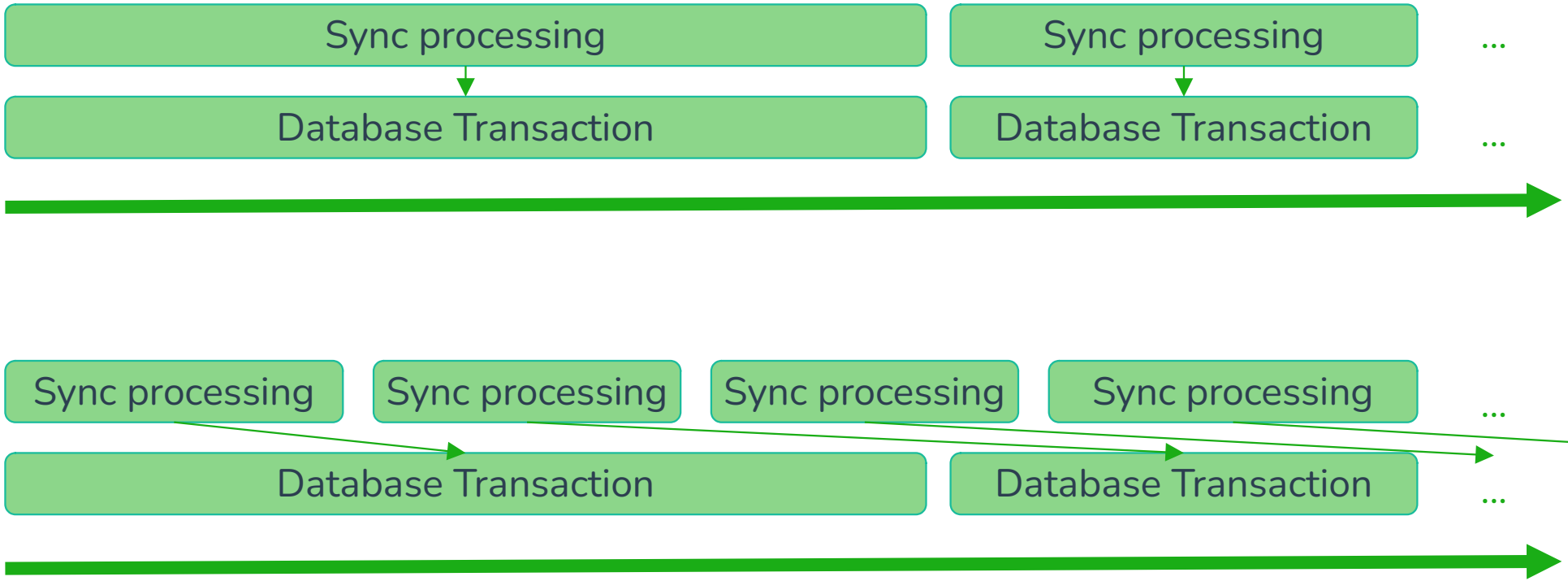
Currently implemented Database-Wrappers

- SQL based databases via Exposed for JVM based targets
- Realm via realm-kotlin for JVM based and native targets
- IndexedDB for JS (browser only)
- In memory for all targets (recommended for testing only)

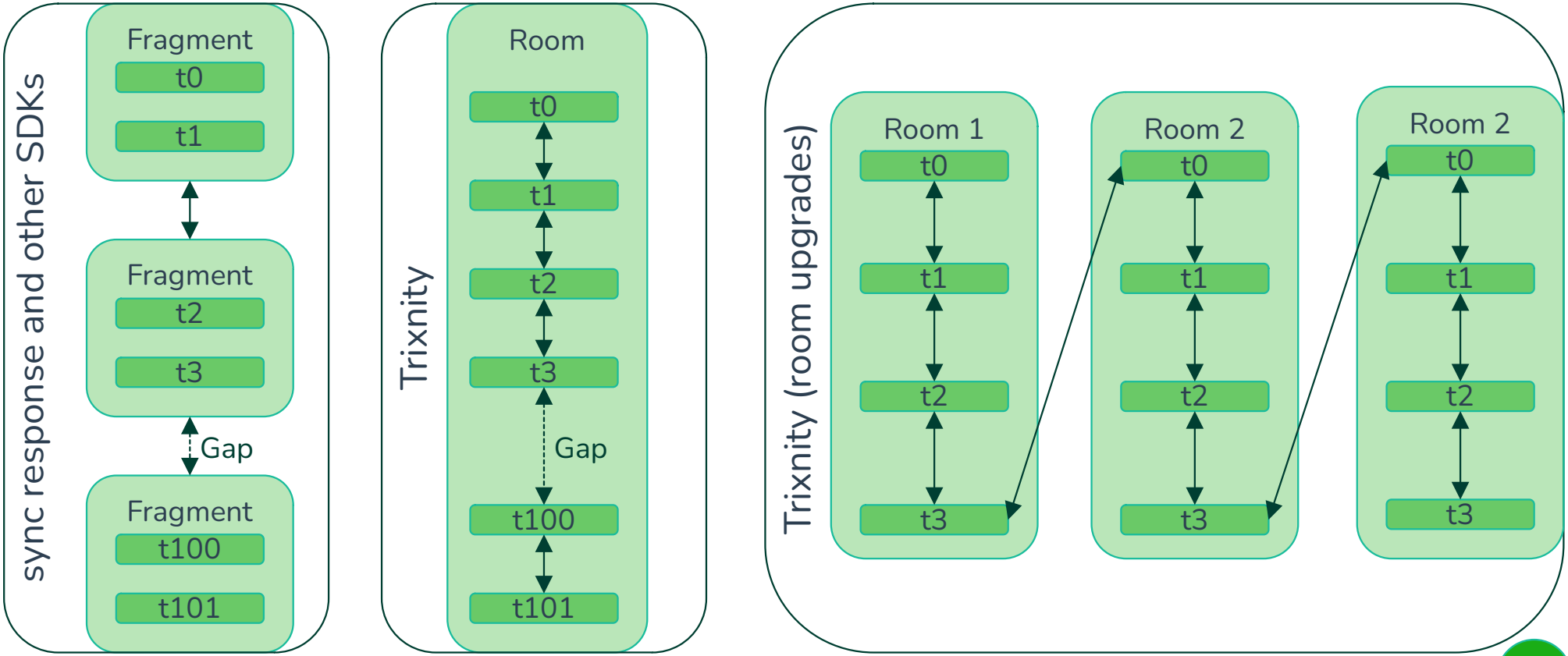
Transactions

… and then there was Realm …

Async Transactions



The Timeline



Bot example

```
matrixClient.room.getTimelineEventsFromNowOn().collect { timelineEvent ->
    val content = timelineEvent.content?.getOrNull()
    if (content is TextMessageEventContent) {
        when {
            content.body.lowercase().startsWith("ping") -> {
                matrixClient.room.sendMessage(timelineEvent.roomId) {
                    text("pong")
                    reply(timelineEvent)
                }
            }
        }
    }
}
```

Domain Specific Language

Projects using Trixnity (the ones I know about)

- Spotify control bot: https://github.com/VaiTon/matrixfy
- Mensa bot: https://github.com/dfuchss/MensaBot
- Some extensions: https://gitlab.com/Doomsdayrs/trixnityx
- Bot command extensions: https://gitlab.com/Doomsdayrs/trixnityx-commands
- Trixnity-Examples (E2EE enabled ping bot running on all (!) targets): https://gitlab.com/trixnity/trixnity-examples
- Timmy messenger (not Open Source yet!): https://timmy-messenger.de

Try it out!

#trixnity:imbitbu.de

@benedict:imbitbu.de