

**[matrix]**

# Decentralised Persistent Communication

[matthew@matrix.org](mailto:matthew@matrix.org)

<http://www.matrix.org>

**Open  
Decentralised  
Persistent  
Eventually Consistent  
Cryptographically Secure  
Messaging Database  
with JSON-over-HTTP API.**

**Matrix is for:**

**Group Chat (and 1:1)  
WebRTC Signalling  
Bridging Comms Silos  
Internet of Things Data**

**...and anything else which needs to  
pubsub persistent data to the world.**

**1<sup>st</sup> law of Matrix:**

**Conversation history and Group  
comms are the 1<sup>st</sup> class citizens.**

**2<sup>nd</sup> law of Matrix:**

**No single party own your  
conversations – they are shared  
over all participants.**

**SIP was built to initiate 1:1 sessions.**

**(inspired by the use cases  
of the PSTN)**

**XMPP was built to pass messages.**

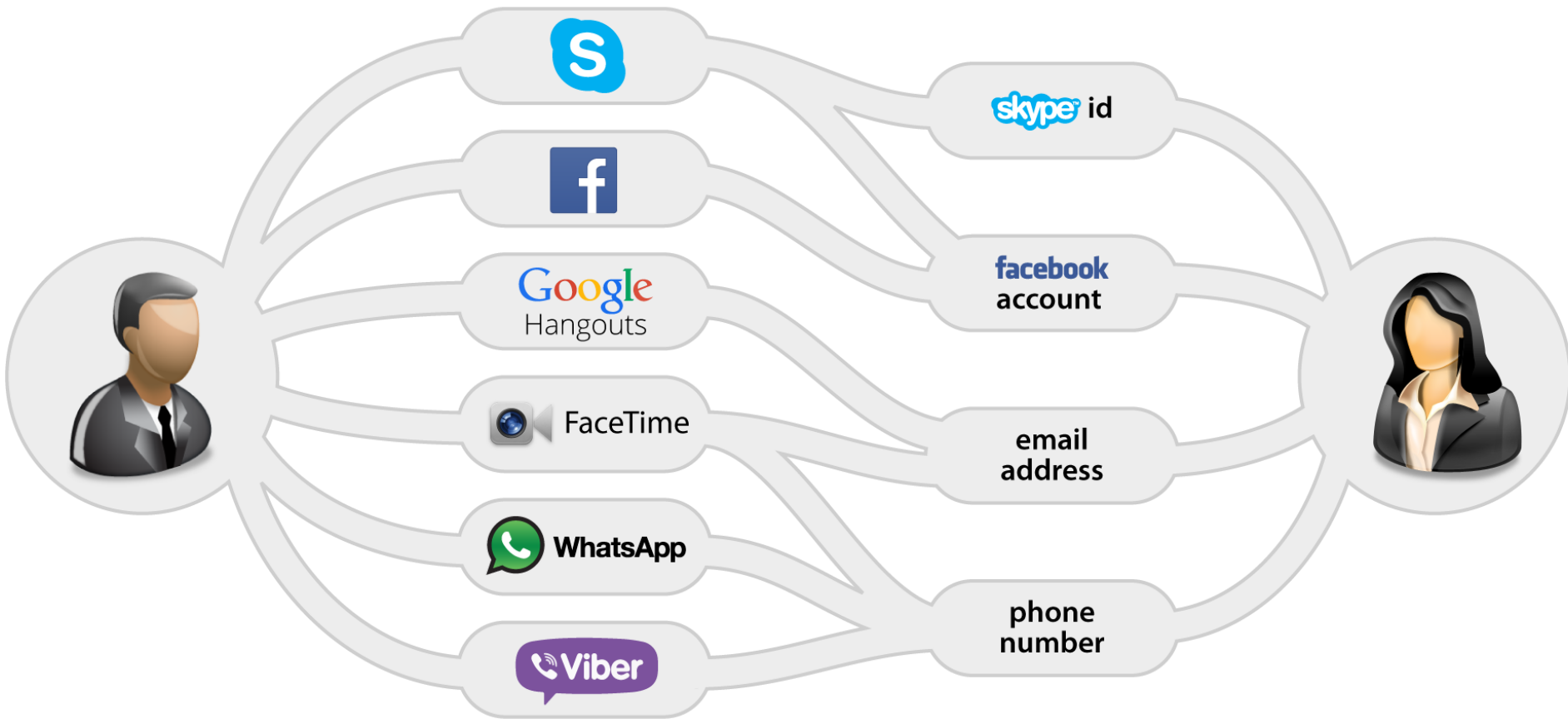
**(inspired by the use cases  
of AIM, ICQ, MSN)**

**Matrix was built to liberate and  
synchronize conversation history.**

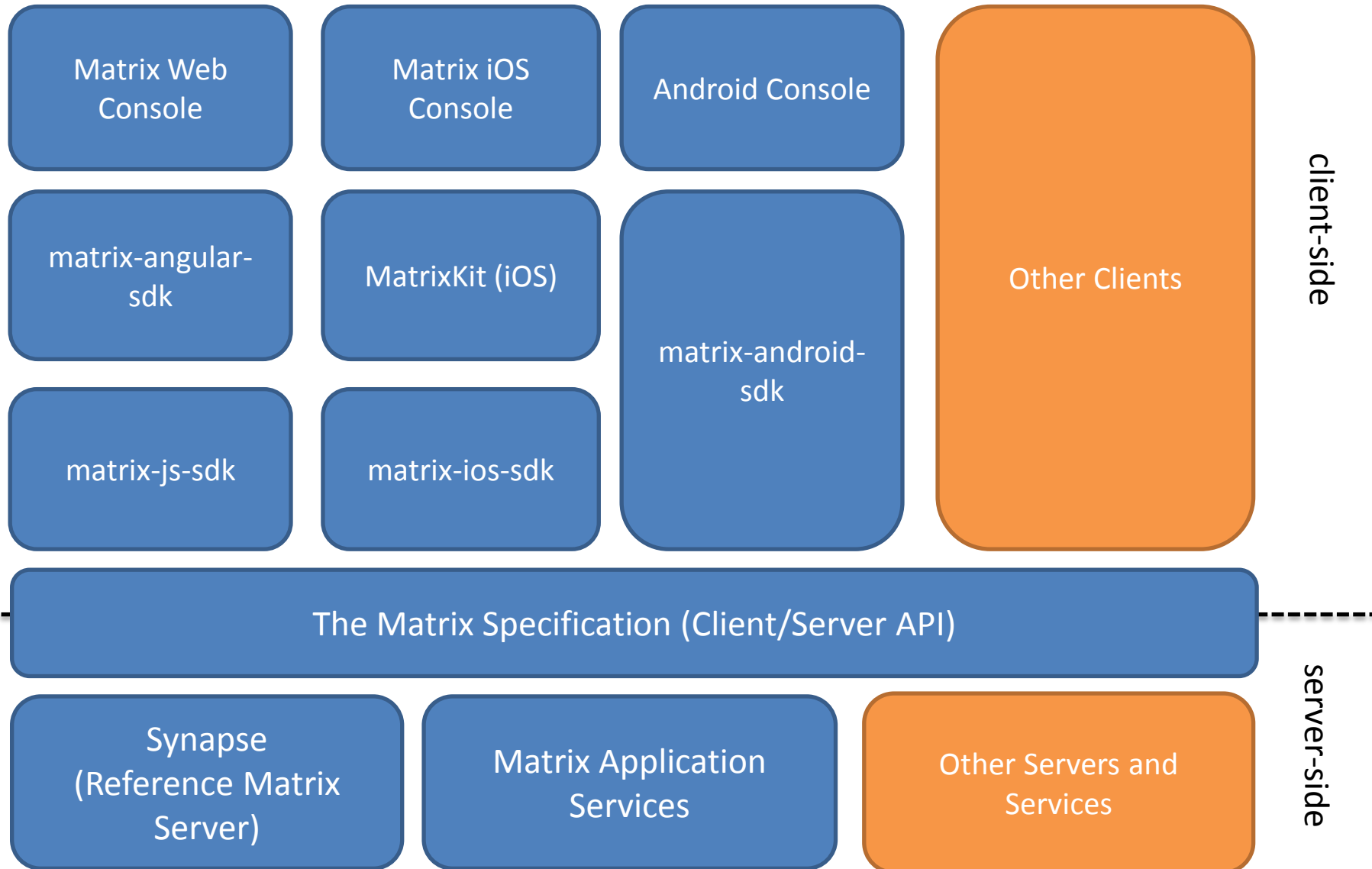
**(inspired by the use cases  
of Slack, Hangouts, Lync, FB, WhatsApp)**



**Why?**



# The Matrix Ecosystem



# Matrix is:

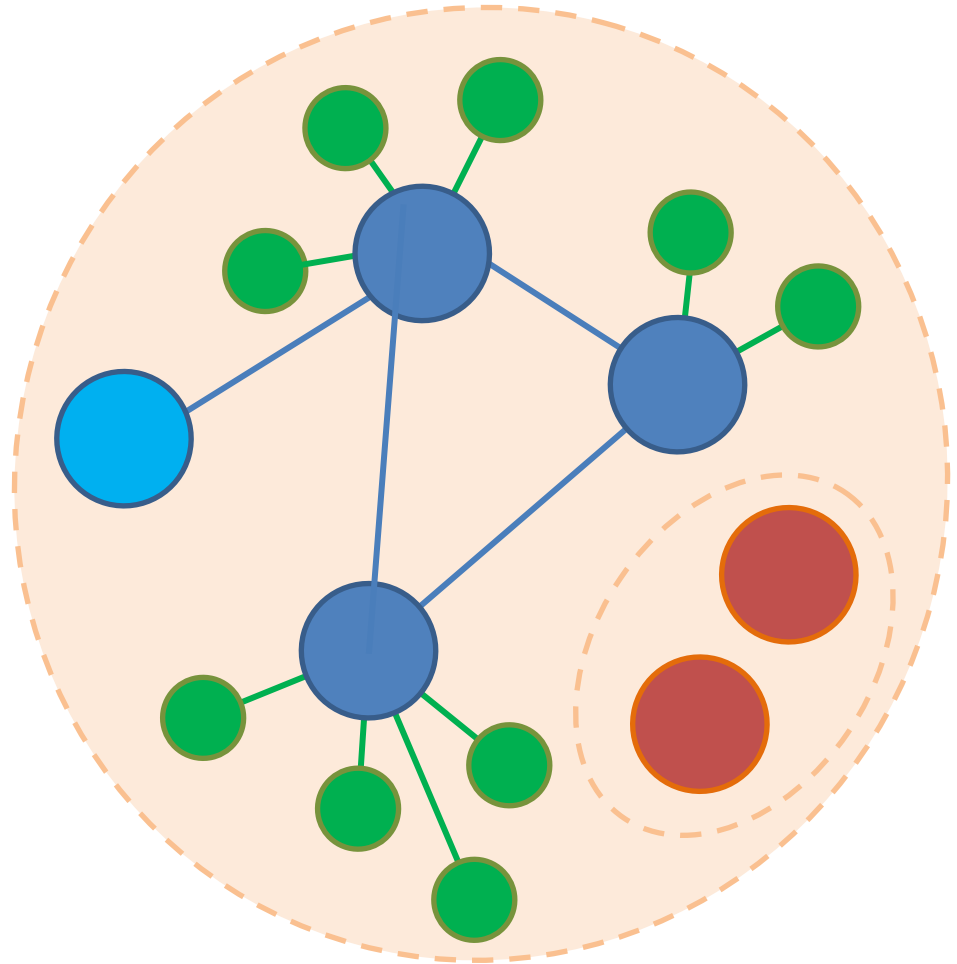
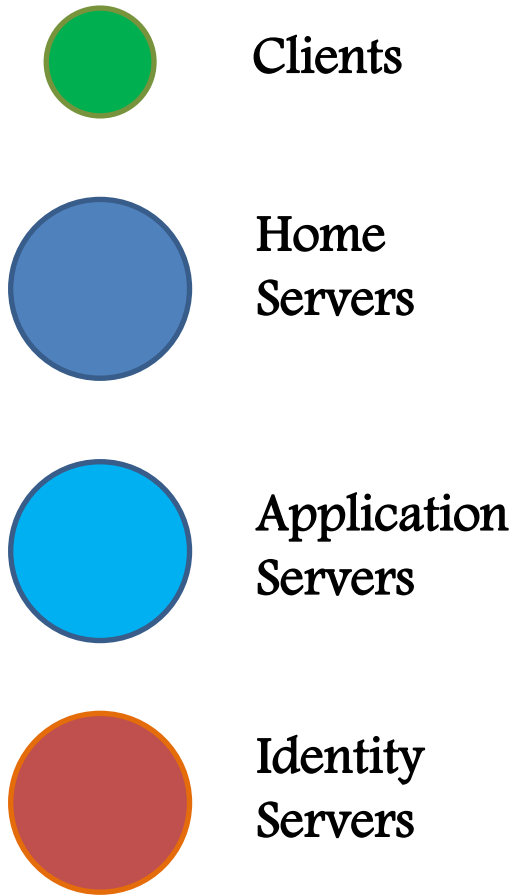
- Non-profit **Open Source Project**
- De-facto **Open Standard HTTP APIs**:
  - Client <-> Server
  - Server <-> Server
  - Application Services <-> Server
- Apache-Licensed Open Source **Reference Impls**
  - **Server** (Python/Twisted)
  - **Client SDKs** (iOS, Android, JS, Angular, Python, Perl)
  - **Clients** (Web, iOS, Android)
  - **Application Services** (IRC, SIP, XMPP, Lync bridges)
- A whole ecosystem of 3<sup>rd</sup> party servers, clients & services

**What does it look like?**

# Demo time!

<http://matrix.org/beta>

# Matrix Architecture



# Functional Responsibility

- **Clients:** Talks simple HTTP APIs to homeservers to push and pull messages and metadata. May be as thin or thick a client as desired.
- **Homeservers:** Stores all the data for a user – the history of the rooms in which they participate; their public profile data.
- **Identity Servers:** Trusted clique of servers (think DNS root servers): maps 3<sup>rd</sup> party IDs to **matrix** IDs.
- **Application Services:** Optional; delivers application layer logic on top of Matrix (Gateways, Conferencing, Archiving, Search etc). Can actively intercept messages if required.



# How does it work?

<http://matrix.org/#about>

# The client-server API

To send a message:

```
curl -XPOST -d '{"msgtype":"m.text", "body":"hello"}'  
"https://alice.com:8448/_matrix/client/api/v1/rooms/ROOM_  
ID/send/m.room.message?access_token=ACCESS_TOKEN"
```

```
{  
  "event_id": "YUwRidLecu"  
}
```

# The client-server API

To set up a WebRTC call:

```
curl -XPOST -d '{\n  "version": 0, \n  "call_id": "12345", \n  "offer": {\n    "type" : "offer",\n    "sdp" : "v=0\r\no=- 658458 2 IN IP4 127.0.0.1..."\n  }\n}'\nhttps://alice.com:8448/_matrix/client/api/v1/rooms/ROOM_ID/send/m.call.invite?access_token=ACCESS_TOKEN"

{ "event_id": "ZruiCZBu" }
```

# Basic 1.1 VoIP Matrix Signalling

```

    Caller                                Callee
m.call.invite ----->
m.call.candidate ----->
[more candidates events]
                                User answers call
                                <----- m.call.answer
[media flows]
                                <----- m.call.hangup
```

# The client-server API

To persist some MIDI:

```
curl -XPOST -d '{\
  "note": "71",\
  "velocity": 68,\
  "state": "on",\
  "channel": 1,\
  "midi_ts": 374023441\
}'
"https://alice.com:8448/_matrix/client/api/v1/rooms/ROOM_
ID/send/org.matrix.midi?access_token=ACCESS_TOKEN"

{ "event_id": "ORzcZn2" }
```

# The server-server API

```
curl -XPOST -H 'Authorization: X-Matrix origin=matrix.org,key="898be4...",sig="j7JXfIcPFDWl1pdJz..."' -d '{
  "ts": 1413414391521,
  "origin": "matrix.org",
  "destination": "alice.com",
  "prev_ids": ["e1da392e61898be4d2009b9fecce5325"],
  "pdu": [{
    "age": 314,
    "content": {
      "body": "hello world",
      "msgtype": "m.text"
    },
    "context": "!fkILCTRBTHhftNYgkP:matrix.org",
    "depth": 26,
    "hashes": {
      "sha256": "MqVORj mjauxBDBzSyN2+Yu+KJxw0oxrrJyuPW8NpELs"
    },
    "is_state": false,
    "origin": "matrix.org",
    "pdu_id": "rKQFuZQawa",
    "pdu_type": "m.room.message",
    "prev_pdus": [
      ["PaBNREEuZj", "matrix.org"]
    ],
    "signatures": {
      "matrix.org": {
        "ed25519:auto": "jZXTwAH/7EZbjHFhIFg8Xj6HGoSI+j7JXfIcPFDWl1pdJz+JJPMHTDIZRha75oJ7l7UM+CnhNAayHWZsUY3Ag"
      }
    },
    "origin_server_ts": 1413414391521,
    "user_id": "@matthew:matrix.org"
  ]
}' https://alice.com:8448/_matrix/federation/v1/send/916d630ea616342b42e98a3be0b74113
```

# Application Services (AS)

- Extensible custom application logic
- They have privileged access to the server (granted by the admin).
- They can subscribe to wide ranges of server traffic (e.g. events which match a range of rooms, or a range of users)
- They can masquerade as 'virtual users'.
- They can lazy-create 'virtual rooms'
- They can receive traffic by push.

# Uses for AS API

- Gateways to other comms platforms
- Data manipulation
  - Filtering
  - Translation
  - Indexing
  - Mining
  - Visualisation
  - Orchestration
- Application Logic (e.g. bots, IVR services)
- ...



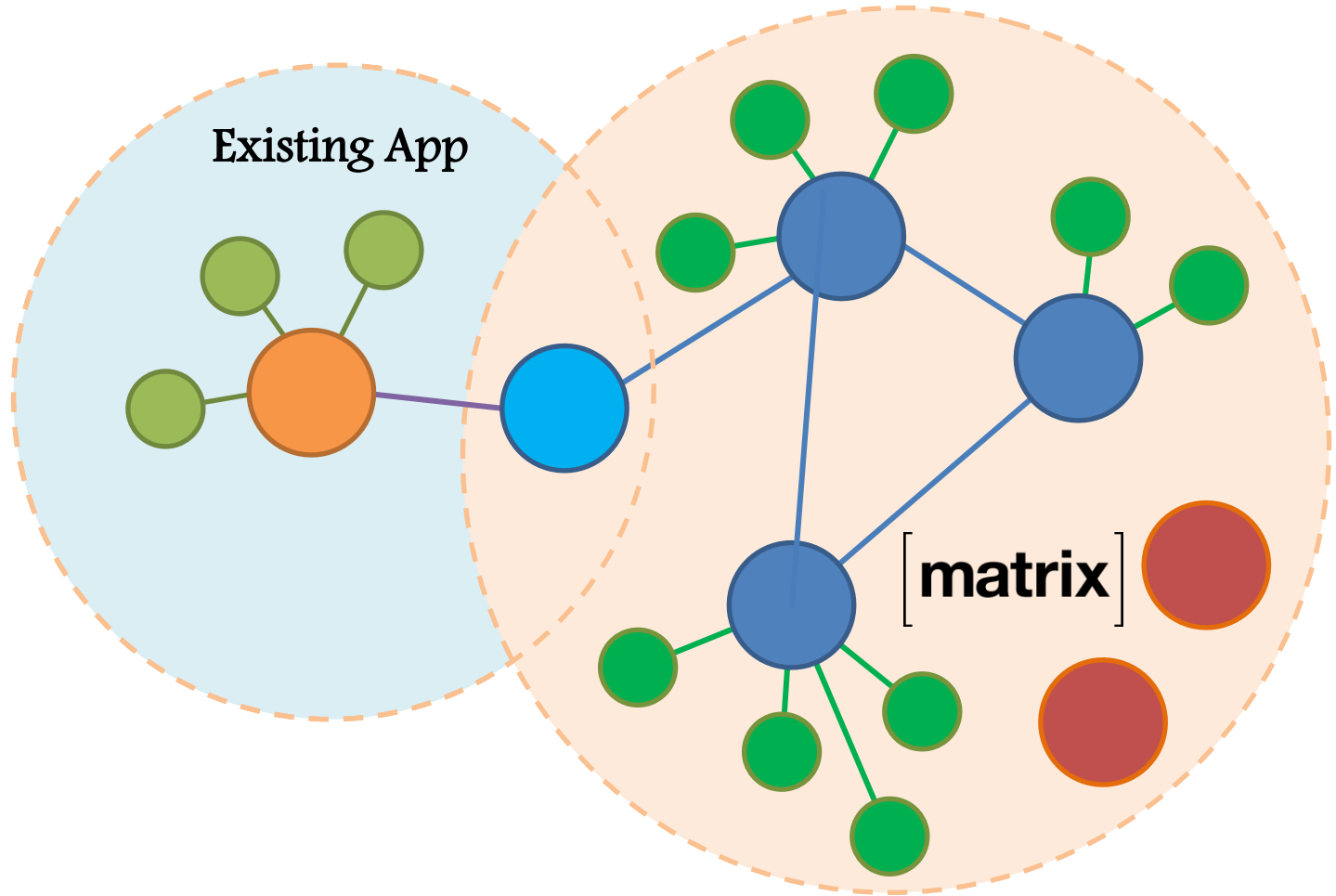
# A trivial application service

```
import json, requests # we will use this later
from flask import Flask, jsonify, request
app = Flask(__name__)

@app.route("/transactions/<transaction>", methods=["PUT"])
def on_receive_events(transaction):
    events = request.get_json()["events"]
    for event in events:
        print "User: %s Room: %s" % (event["user_id"], event["room_id"])
        print "Event Type: %s" % event["type"]
        print "Content: %s" % event["content"]
    return jsonify({})

if __name__ == "__main__":
    app.run()
```

# Matrix Bridging with ASes



# Kamailio SIP<->Matrix GW

SIP->Matrix ingress call:

```
-----  
-> SIP INVITE  
<- SIP 100  
    create Matrix room inviting other party (reuse existing one if available)  
    -> PUT m.call.invite  
    <- PUT m.call.answer  
<- 200 OK  
-> ACK  
-----  
-> SIP BYE  
    m.call.hangup  
<- SIP 200 OK  
-----  
<- m.call.hangup  
    -> BYE  
    <- 200 OK  
-----
```

# Kamailio SIP<->Matrix GW

- Use utils and json modules with `http_query()` or OEJ's curl module to relay to Matrix
- Use xhttp module and `event_route[xhttp:request]` to receive traffic from Matrix and relay to SIP via `txmt_reply_callid()` or `dlg_bye()`?

# Current Progress

- Funded May 2014
- Launched alpha Sept 2014
- Entered beta Dec 2014
- Stable v0.9 Beta May 2014
- July 2014: v1.0 release?!
- Remaining:
  - Build more gateways
  - Polish spec
  - End-to-End Encryption
  - v2 Client-Server API

# What's next?

- End-to-end encryption
- Reusable web UI components and improving the web client
- Multi-way VoIP
- Lots more Application Services
- Landing V2 APIs
- Use 3rd party IDs by default
- Yet more performance work
- Spec polishing
- New server implementations!

**We need help!!**

- We need people to try running their own servers and join the federation.
- We need people to run gateways to their existing services
- We need feedback on the APIs.
- Consider native Matrix support for new apps
- Follow @matrixdotorg and spread the word!





# [matrix]

Thank you!

matthew@matrix.org

<http://matrix.org>

@matrixdotorg

# Federation Design #1

- No single point of control for chat rooms.
- Any homeserver can publish a reference to a chat room (although typically the address is the homeserver of the user who created the room).
- Room addresses look like:

**#matrix:matrix.org**

(pronounced hash-matrix-on-matrix-dot-org)

- The IP of the matrix.org homeserver is discovered through DNS (SRV \_matrix record if available, otherwise looks for port 8448 of the A record).

# Federation Design #2

- When a user joins a room, his HS queries the HS specified in the room name to find a list of participating homeservers via a simple GET
- Messages form a directed acyclic graph (DAG) of chronologicity, each crypto-signed by the origin HS
- The user's HS pulls in messages via GETs from participating HSs by attempting to walk the DAG
- Each HS caches as much history as its users (or admin) desires
- When sending a message, the HS PUTs to participating homeservers (currently full mesh, but fan-out semantics using cyclical hashing in development)

# Identity Design

- We don't want to be yet another identity system (e.g. JIDs)
- So we aggregate existing 3<sup>rd</sup> party IDs (3PID) and map them to **matrix** IDs (MXIDs) by **Identity Servers**, whose use in public is strictly optional.
- And so login and user discovery is typically done entirely with 3<sup>rd</sup> party IDs.
- ID servers validate 3<sup>rd</sup> party IDs (e.g. email, MSISDN, Facebook, G+) and map them to MXIDs. MXIDs look like:

**@matthew:matrix.org**

# Security Design #1

- Server-server traffic is mandatorily TLS from the outset
- Can use official CA certs, but automagically self-sign and submit certs to **matrix** ID servers as a free but secure alternative
- Server-client traffic mandates transport layer encryption other than for tinkering
- Clients that support PKI publish their public keys, and may encrypt and sign their messages for E2E security.
- "Well behaved" clients should participate in key escrow servers to allow private key submission for law enforcement.
- End-to-end encryption for group chat is supported through a per-room encryption key which is shared 1:1 between participating members

# Security Design #2

- SPAM is contained by mandating invite handshake before communication
- Invite handshakes are throttled per user
- Homeservers and users may be blacklisted on identity servers
- ID servers authenticating 3PIDs are obligated to mitigate bulk registration of users via CAPTCHAs or domain-specific techniques (e.g. 2FA SMS for MSISDNs)

# Application Services: Spec & API

- Still in development; some early prototypes
- "Passive AS-API" Builds on the client-server API
  - Service registers a URL for inbound events to be PUT to
  - Allows a service to register for traffic on behalf of a namespace of virtual users and virtual rooms
  - Adds "superuser" permissions to subscribe to arbitrary filters of events on the homeserver, and inject arbitrary events
  - Modeled loosely after IRC Services
- Also: Active AS API for intercepting inbound events on a HS, and Storage API for exposing existing conversation DBs to Matrix via a HS.

# AS Example: Matrix/SMS Gateway

- matrix.org runs a homeserver.
- Matrix/SMS gw AS is registered to the homeserver, masquerading for the 'sms.matrix.org' domain.
- @447968722968.sms.matrix.org routes to the homeserver from anywhere in Matrix, which passes events for \*.sms.matrix.org through to the AS
- Matrix/SMS Gateway then relays via SMS aggregators to send SMS to +447968722968
- The reverse path is symmetrical, with the Matrix/SMS AS injecting events into the HS on behalf of @447968722968.sms.matrix.org



## AS Example: Matrix/SIP Gateway

- Similarly, AS can implement a SIP gateway, posing as a range of virtual matrix users.
- Events such as 'm.call.invite' and 'm.call.candidates' are PUT to the AS by the HS
- AS converts directly into SIP signalling (reINVITEing to advertise new ICE candidates)
- Media flows out-of-band to Matrix as typical WebRTC SRTP.
- We've already written a basic Matrix/Verto gateway (using client-service API – see [matrix.org/blog](https://matrix.org/blog))

# Why not XMPP?

- We used to use XMPP (ejabberd, OpenFire, Spectrum, psyced, Psi, Pidgin, ASmack, Spark, XMPP.Framework)
- We built an alternative because:
  - Single server per MUC is single point of control
  - Synchronised history is a very 2<sup>nd</sup> class citizen
  - Stanzas aren't framed or reliably delivered
  - XMPP stacks are not easy to implement in a web environment
  - Jingle is complicated and exotic
  - XML is needlessly verbose and unwieldy
  - The baseline feature-set is too minimal
  - JIDs haven't taken off like Email or MSISDNs
  - Not designed for mobile use cases (e.g. push; low bw)
  - Well documented spam and identity/security issues
  - ejabberd