Architecting for Scale

A case-study in utilizing a sharded architecture for infinite* scalability

Sumner Evans

27 September 2022

Beeper

A bit about me

My name is Sumner, I'm a **software engineer at Beeper**.

- I graduated from Colorado School of Mines in 2018 with my bachelor's in CS and 2019 with a master's in CS.
- I am an adjunct professor. Currently I'm teaching CSCI 400. I've taught 406, and 564 in the past as well.
- I enjoy skiing, volleyball, and soccer.
- I'm a 4th degree black belt in ATA taekwondo.

Overview

1. A bit about Beeper
2. What we've built
3. What is software architecture?
4. A bit about Beeper's current architecture
5. Our new architecture

This talk is interactive!

If you have questions at any point, feel free to interrupt me.

A bit about Beeper

Beeper's mission

Our mission is to:

> *make it easy for everyone on Earth to chat with each other.*

We specifically chose the word "chat" rather than "communicate" because we are focusing on *people talking to one another*.

Beeper's mission

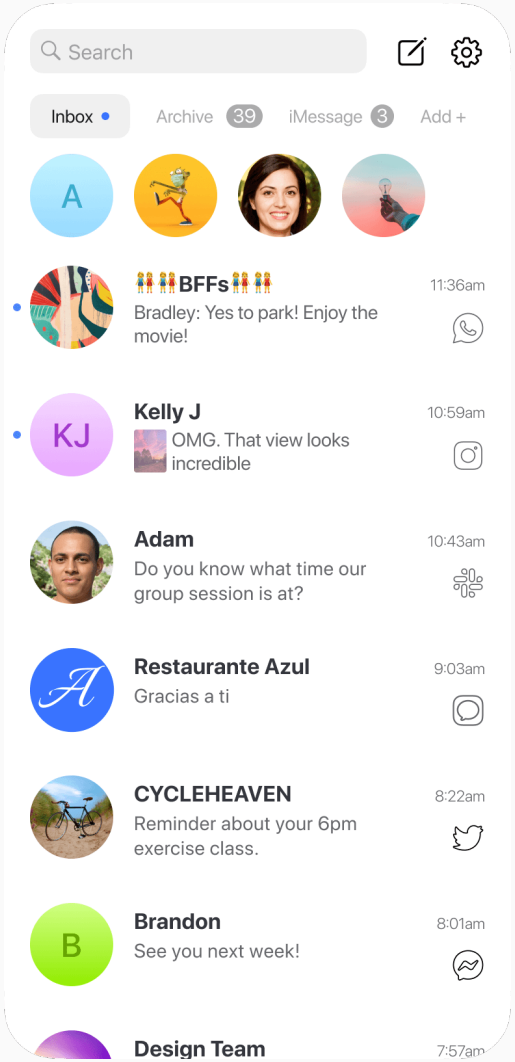
Our mission is to:

> *make it easy for everyone on Earth to chat with each other.*

We specifically chose the word "chat" rather than "communicate" because we are focusing on *people talking to one another*.

What is Beeper?

Beeper is an app that brings all of your chat networks together into a single inbox.



What is Beeper?

Beeper allows you to consolidate messages from 15+ chat apps into a single inbox.

- Beeper is available on on macOS, Windows, Linux iPhone, iPad, Android and Chrome OS.
- Beeper is built on top of the Matrix protocol.
- Beeper encrypts all chats, including bridged chats, by default[1].

[1] We have to momentarily decrypt your chat messages to translate from the other network to Matrix, but we never log or store your unencrypted messages.

What is Beeper?

Beeper allows you to consolidate messages from 15+ chat apps into a single inbox.

- Beeper is available on on macOS, Windows, Linux iPhone, iPad, Android and Chrome OS.
- Beeper is built on top of the Matrix protocol.
- Beeper encrypts all chats, including bridged chats, by default[1].

[1] We have to momentarily decrypt your chat messages to translate from the other network to Matrix, but we never log or store your unencrypted messages.

We are targeting individuals, not enterprises

We want to beat WhatsApp, not Slack.

Our core competency is the 50th highest priority at the big tech companies.

Most chat networks are an afterthought within the ecosystem of a larger company. At Beeper, chat is all we care about.

We are targeting individuals, not enterprises

We want to beat WhatsApp, not Slack.

Our core competency is the 50th highest priority at the big tech companies.

Most chat networks are an afterthought within the ecosystem of a larger company. At Beeper, chat is all we care about.

How we make money

We charge a flat $10/month fee to use Beeper. This fee allows users to connect as many chat networks as they want.

No ads. No data mining. We take advantage of Matrix's E2EE to add additional privacy guarantees.

The transaction is simple:
users pay us money for a service.

How we make money

We charge a flat $10/month fee to use Beeper. This fee allows users to connect as many chat networks as they want.

No ads. No data mining. We take advantage of Matrix's E2EE to add additional privacy guarantees.

The transaction is simple:
users pay us money for a service.

How we make money

We charge a flat $10/month fee to use Beeper. This fee allows users to connect as many chat networks as they want.

No ads. No data mining. We take advantage of Matrix's E2EE to add additional privacy guarantees.

The transaction is simple:
users pay us money for a service.

Bridging to a glorious Matrix future

We see bridges as a way to onboard users into the Matrix ecosystem.

Users can switch to a new app without loosing a single conversation!

We encourage our users to connect *all* of their chat networks to Beeper.

Bridging to a glorious Matrix future

We see bridges as a way to onboard users into the Matrix ecosystem.

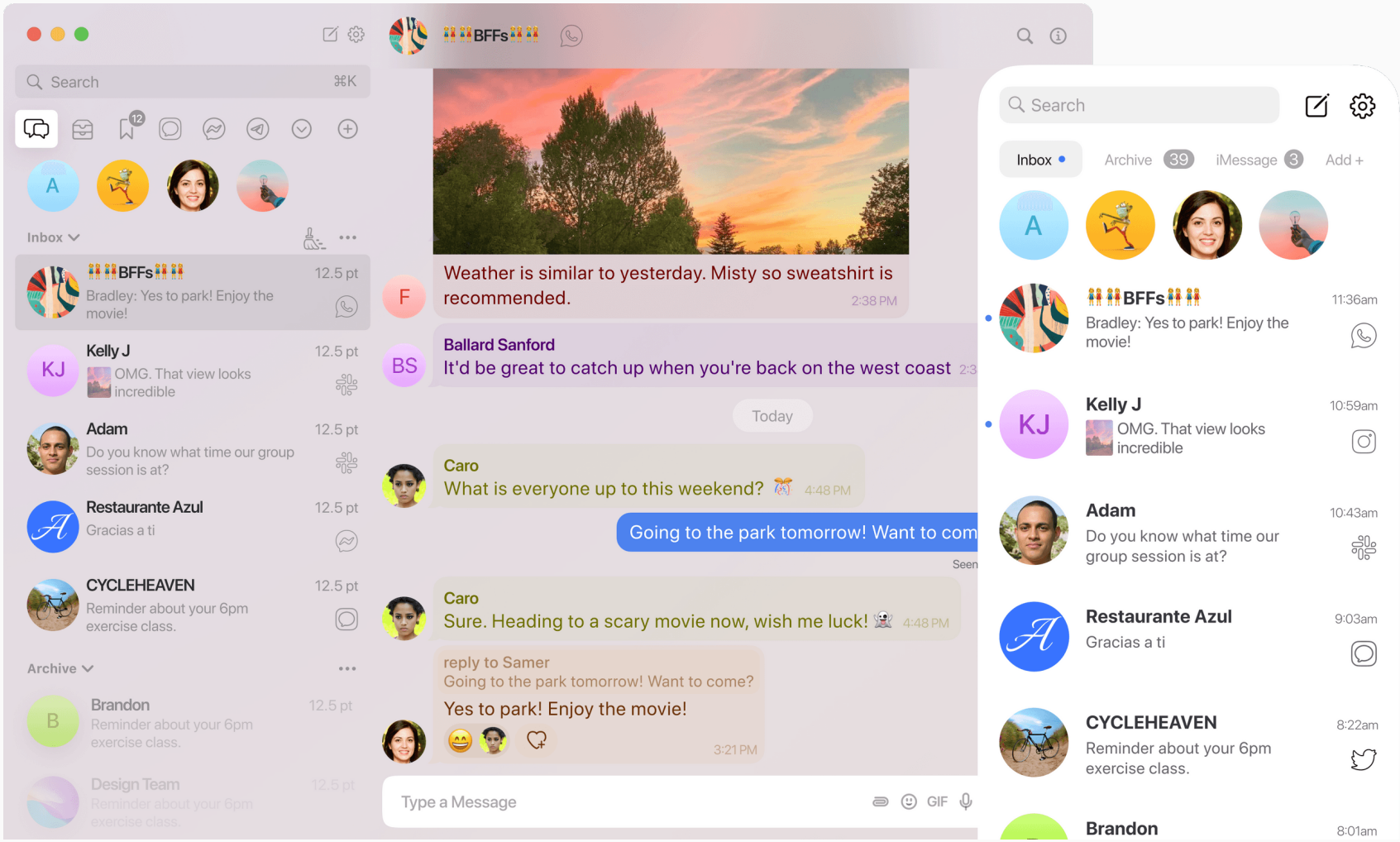
Users can switch to a new app without loosing a single conversation!

We encourage our users to connect *all* of their chat networks to Beeper.

What we’ve built

Custom clients for desktop and mobile

We forked Element, and added custom features.



Bridges, bridges, and more bridges!

We have built bridges to 10 networks:

- iMessage
- WhatsApp
- Facebook Messenger
- SMS (Android)
- Telegram
- Signal
- LinkedIn
- Instagram
- Twitter
- Google Chat

We are actively developing new bridges to Discord and Slack.

You can also connect to IRC (via the Libera.chat bridge) and the rest of the Matrix federation.

Bridges, bridges, and more bridges!

We have built bridges to 10 networks:

- iMessage
- WhatsApp
- Facebook Messenger
- SMS (Android)
- Telegram
- Signal
- LinkedIn
- Instagram
- Twitter
- Google Chat

We are actively developing new bridges to Discord and Slack.

You can also connect to IRC (via the Libera.chat bridge) and the rest of the Matrix federation.

Bridges, bridges, and more bridges!

We have built bridges to 10 networks:

- iMessage
- WhatsApp
- Facebook Messenger
- SMS (Android)
- Telegram
- Signal
- LinkedIn
- Instagram
- Twitter
- Google Chat

We are actively developing new bridges to Discord and Slack.

You can also connect to IRC (via the Libera.chat bridge) and the rest of the Matrix federation.

What I work on at Beeper

I am on the newly created *Scaling* team.

Our current objective is to prepare Beeper for rocket-ship growth.

I was previously part of the *Bridges* team.

Notable projects included:

- Writing the LinkedIn bridge
- Adding bridge status reporting and message send status
- Implementing massive stability improvements in the Signal bridge
- Implementing incremental infinite backfill in our WhatsApp and Facebook bridges

What I work on at Beeper

I am on the newly created *Scaling* team.

Our current objective is to prepare Beeper for rocket-ship growth.

I was previously part of the *Bridges* team.

Notable projects included:

- Writing the LinkedIn bridge
- Adding bridge status reporting and message send status
- Implementing massive stability improvements in the Signal bridge
- Implementing incremental infinite backfill in our WhatsApp and Facebook bridges

What is software architecture?

What do you think software architecture is?

- What questions do software architects ask?
- What concerns do software architects have to take into account?
- What tradeoffs do software architects have to consider?

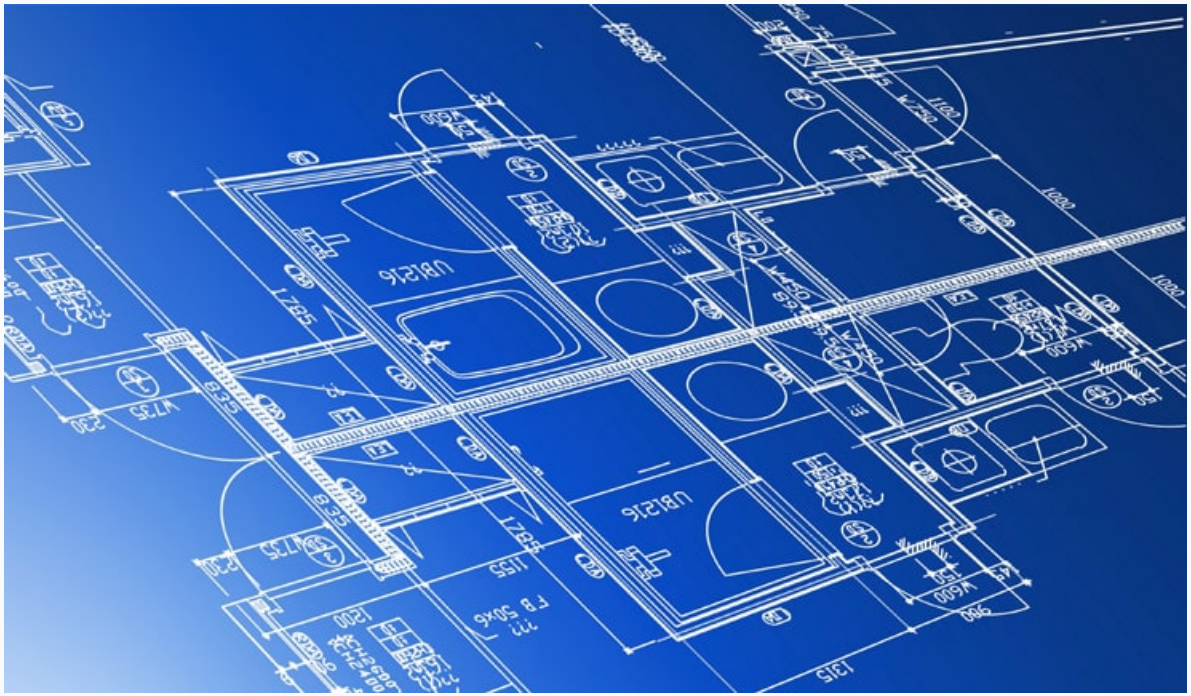
What do you think software architecture is?

- What questions do software architects ask?
- What concerns do software architects have to take into account?
- What tradeoffs do software architects have to consider?

What do you think software architecture is?

- What questions do software architects ask?
- What concerns do software architects have to take into account?
- What tradeoffs do software architects have to consider?

Software architecture is about component interactions



Just like architects have to think about the interactions between plumbing, electrical, HVAC, etc., software architects have to think about the interactions of components in the software system.

“The stack”

No, not the one you learn in Data Structures.

In software architecture, when we refer to the stack we are referring to the tools, services, and languages used for the various components of the system.

The closer a system is to a user, the further *up* the stack it is.

“The stack”

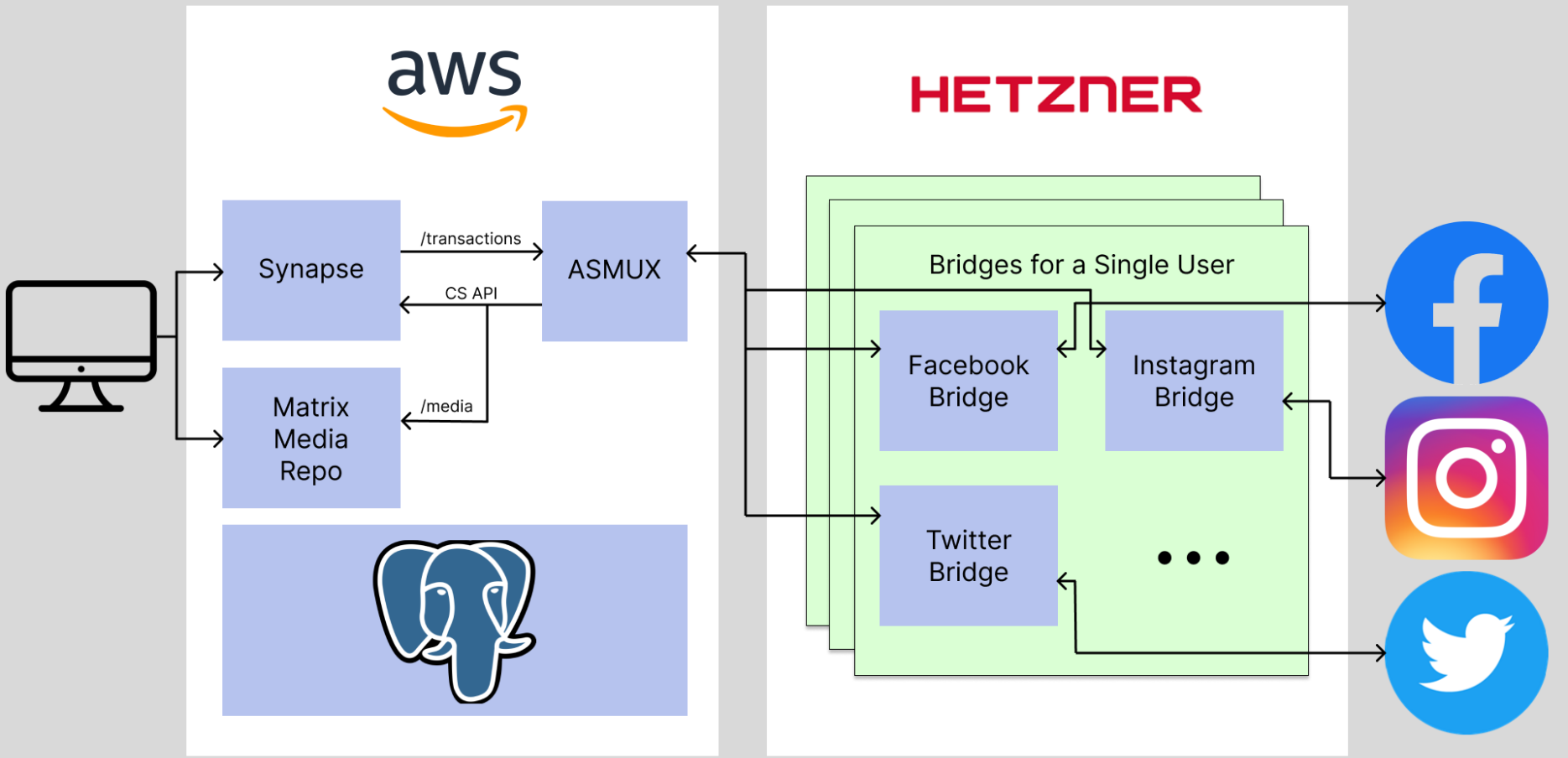
No, not the one you learn in Data Structures.

In software architecture, when we refer to the stack we are referring to the tools, services, and languages used for the various components of the system.

The closer a system is to a user, the further *up* the stack it is.

A bit about Beeper’s current architecture

A diagram



Each user gets their own bridge for each network they connect.

Advantages of our architecture

- ASMUX allows us to dynamically change what bridges are running
- Each users' bridge is **isolated**
- We can deploy **different bridge versions** to different sets of users
- Bridges can be stopped and started individually
- Double puppeting

Advantages of our architecture

- ASMUX allows us to dynamically change what bridges are running
- Each users' bridge is **isolated**
- We can deploy **different bridge versions** to different sets of users
- Bridges can be stopped and started individually
- Double puppeting

Advantages of our architecture

- ASMUX allows us to dynamically change what bridges are running
- Each users' bridge is **isolated**
- We can deploy **different bridge versions** to different sets of users
- Bridges can be stopped and started individually
- Double puppeting

Advantages of our architecture

- ASMUX allows us to dynamically change what bridges are running
- Each users' bridge is **isolated**
- We can deploy **different bridge versions** to different sets of users
- Bridges can be stopped and started individually
- Double puppeting

Advantages of our architecture

- ASMUX allows us to dynamically change what bridges are running
- Each users' bridge is **isolated**
- We can deploy **different bridge versions** to different sets of users
- Bridges can be stopped and started individually
- Double puppeting

Disadvantages of our architecture

- We have to run a **lot** of bridges
- Synapse is not designed for a dynamic numbers of bridges
- If two users join the same chat on an external network, we end up with two rooms with the same data
- We run Synapse on AWS which is relatively expensive
- Synapse becomes a bottleneck for all traffic

Disadvantages of our architecture

- We have to run a **lot** of bridges
- Synapse is not designed for a dynamic numbers of bridges
- If two users join the same chat on an external network, we end up with two rooms with the same data
- We run Synapse on AWS which is relatively expensive
- Synapse becomes a bottleneck for all traffic

Disadvantages of our architecture

- We have to run a **lot** of bridges
- Synapse is not designed for a dynamic numbers of bridges
- If two users join the same chat on an external network, we end up with two rooms with the same data
- We run Synapse on AWS which is relatively expensive
- Synapse becomes a bottleneck for all traffic

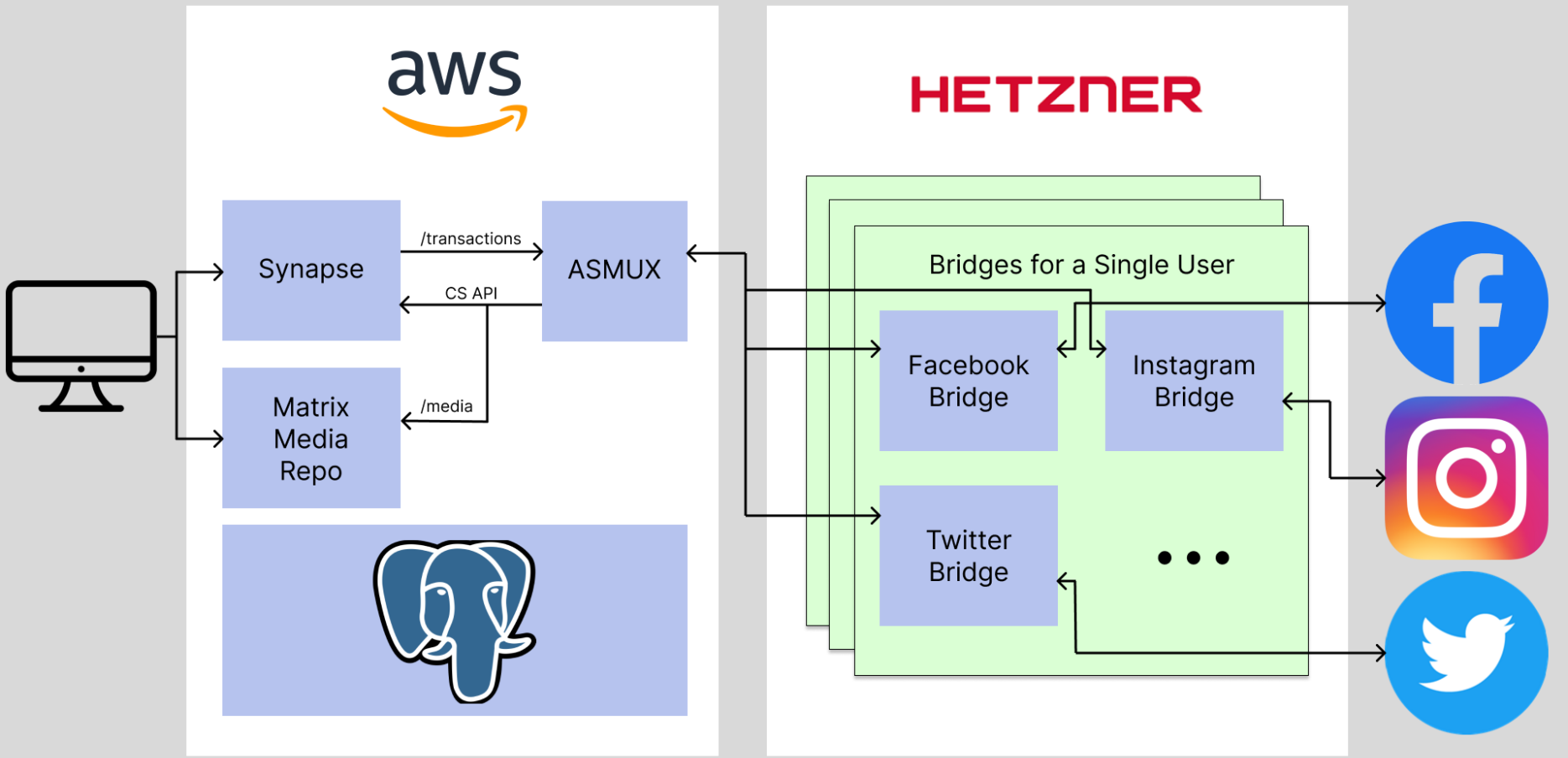
Disadvantages of our architecture

- We have to run a **lot** of bridges
- Synapse is not designed for a dynamic numbers of bridges
- If two users join the same chat on an external network, we end up with two rooms with the same data
- We run Synapse on AWS which is relatively expensive
- Synapse becomes a bottleneck for all traffic

Disadvantages of our architecture

- We have to run a **lot** of bridges
- Synapse is not designed for a dynamic numbers of bridges
- If two users join the same chat on an external network, we end up with two rooms with the same data
- We run Synapse on AWS which is relatively expensive
- Synapse becomes a bottleneck for all traffic

Back to the diagram



The core software in this architecture is Synapse.

Stats

Because we encourage users to connect every chat network, we have a lot of puppet users.

On average, each user brings 4716 puppeted users.

All of these puppets represent real people on other networks, so they each generate their own traffic.

On average, each user and their puppets collectively account for 100k events!

None of that traffic is federated, but it has to go to Synapse which is designed with federation front-and-center!

Stats

Because we encourage users to connect every chat network, we have a lot of puppet users.

On average, each user brings 4716 puppeted users.

All of these puppets represent real people on other networks, so they each generate their own traffic.

On average, each user and their puppets collectively account for 100k events!

None of that traffic is federated, but it has to go to Synapse which is designed with federation front-and-center!

Stats

Because we encourage users to connect every chat network, we have a lot of puppet users.

On average, each user brings 4716 puppeted users.

All of these puppets represent real people on other networks, so they each generate their own traffic.

On average, each user and their puppets collectively account for 100k events!

None of that traffic is federated, but it has to go to Synapse which is designed with federation front-and-center!

A few more notes

- **Synapse is aggressively federated**
  Synapse has no special handling of federated vs unfederated rooms.
- **Synapse does not handle message floods well**
  Synapse falls over when users bridge some large Telegram chats and Discord servers.
- **Deleting rooms from Synapse is hard**
  We end up with lots of *dead* rooms when users delete their bridges and never turn it back on (or need a hard bridge reset).

A few more notes

- **Synapse is aggressively federated**
  Synapse has no special handling of federated vs unfederated rooms.
- **Synapse does not handle message floods well**
  Synapse falls over when users bridge some large Telegram chats and Discord servers.
- **Deleting rooms from Synapse is hard**
  We end up with lots of *dead* rooms when users delete their bridges and never turn it back on (or need a hard bridge reset).

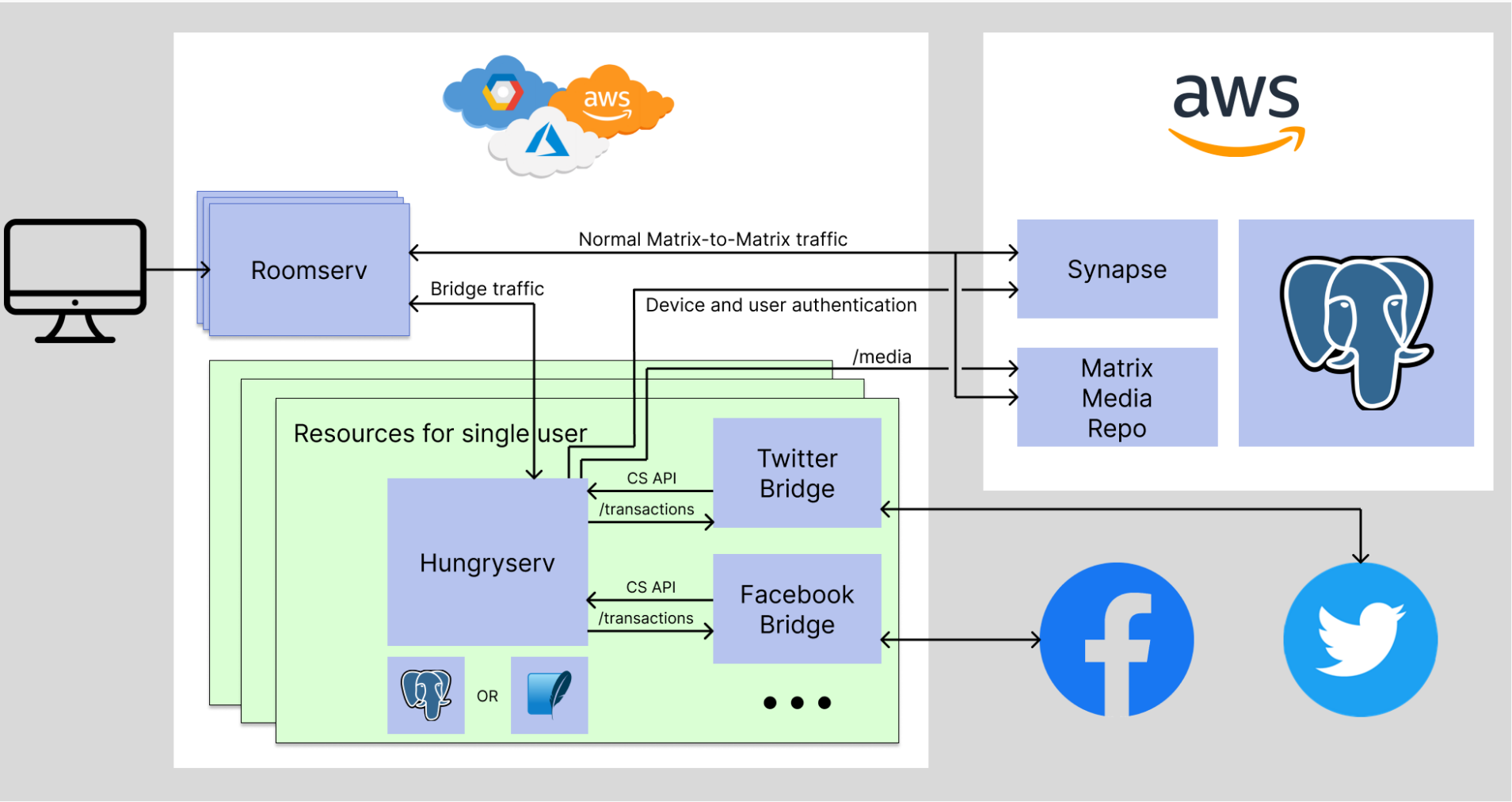
A few more notes

- **Synapse is aggressively federated**
  Synapse has no special handling of federated vs unfederated rooms.
- **Synapse does not handle message floods well**
  Synapse falls over when users bridge some large Telegram chats and Discord servers.
- **Deleting rooms from Synapse is hard**
  We end up with lots of *dead* rooms when users delete their bridges and never turn it back on (or need a hard bridge reset).

Solution: shard our homeserver based on bridged vs. unbridged traffic

Our new architecture

A sharded future



Let's talk about the differences from the old architecture...

Splitting the homeserver, keeping the protocol

This new architecture allows us to handle Matrix-native federated traffic separately from our high-throughput unfederated bridge traffic.

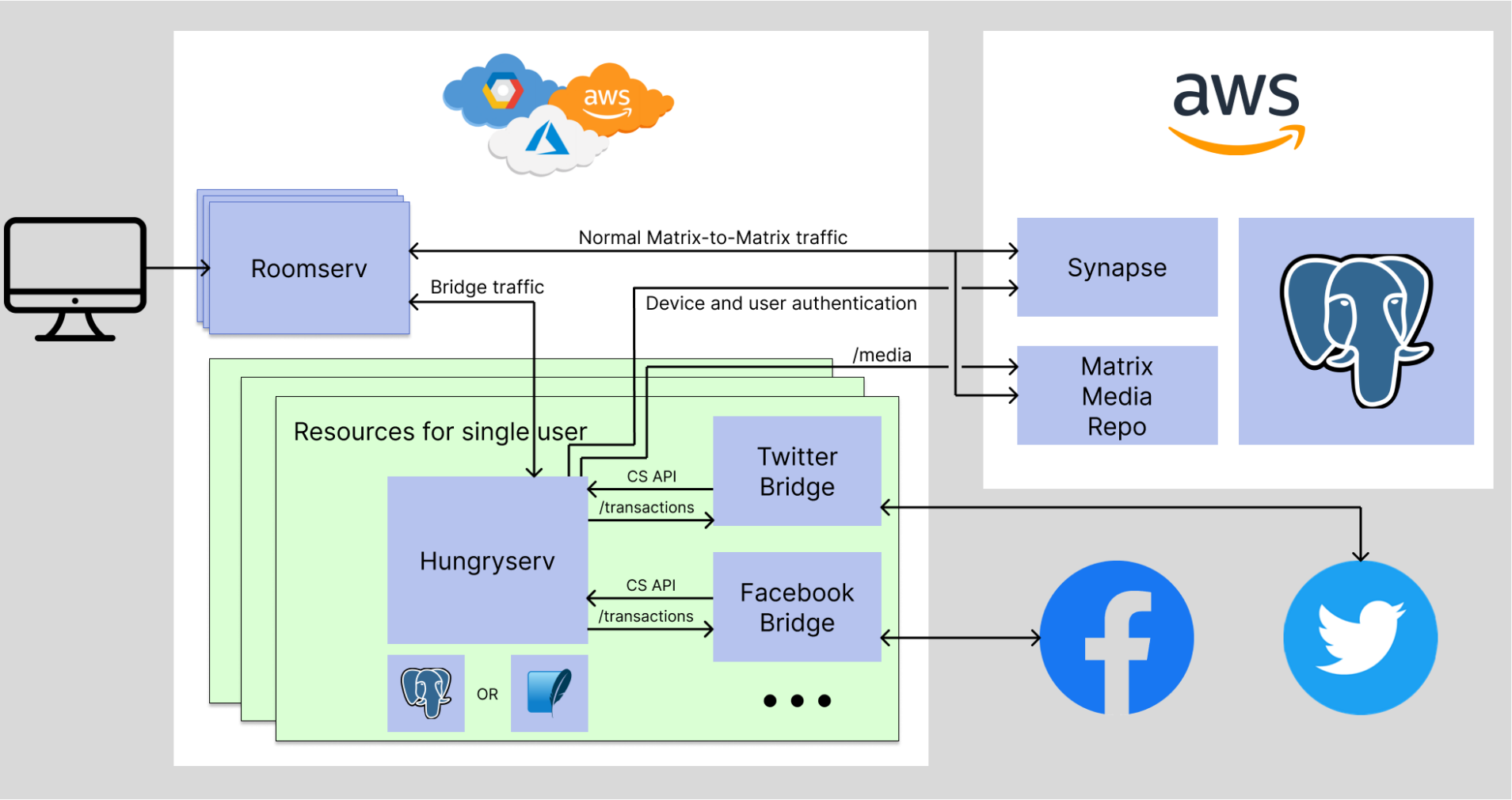
All while maintaining API compatibility with clients, bridges, and the Matrix federation!

Splitting the homeserver, keeping the protocol

This new architecture allows us to handle Matrix-native federated traffic separately from our high-throughput unfederated bridge traffic.

All while maintaining API compatibility with clients, bridges, and the Matrix federation!

A sharded future



Let's talk about the differences from the old architecture...

Hungryserv: why is it hungry?

Because it’s unfederated!

Hungryserv: why is it hungry?

Because it’s unfederated!

Hungryserv

We run a **Hungryserv** instance for every user to handle that user's bridge traffic.

- If there's a message flood from a bridge, it only affects that user.
- We don't have to store all of the bridge events and users in our Synapse database.
- Synapse is less resource-constrained and can do what it does best: federate with other Matrix homeservers.

We are also currently looking into ways of using **SQLite** in production using Litestream.

Hungryserv

We run a **Hungryserv** instance for every user to handle that user's bridge traffic.

- If there's a message flood from a bridge, it only affects that user.
- We don't have to store all of the bridge events and users in our Synapse database.
- Synapse is less resource constrained and can do what it does best: federate with other Matrix homeservers.

We are also currently looking into ways of using **SQLite** in production using Litestream.

Hungryserv

We run a **Hungryserv** instance for every user to handle that user's bridge traffic.

- If there's a message flood from a bridge, it only affects that user.
- We don't have to store all of the bridge events and users in our Synapse database.
- Synapse is less resource constrained and can do what it does best: federate with other Matrix homeservers.

We are also currently looking into ways of using **SQLite** in production using Litestream.

Hungryserv

We run a **Hungryserv** instance for every user to handle that user's bridge traffic.

- If there's a message flood from a bridge, it only affects that user.
- We don't have to store all of the bridge events and users in our Synapse database.
- Synapse is less resource constrained and can do what it does best: federate with other Matrix homeservers.

We are also currently looking into ways of using **SQLite** in production using Litestream.

Hungryserv

We run a **Hungryserv** instance for every user to handle that user's bridge traffic.

- If there's a message flood from a bridge, it only affects that user.
- We don't have to store all of the bridge events and users in our Synapse database.
- Synapse is less resource constrained and can do what it does best: federate with other Matrix homeservers.

We are also currently looking into ways of using **SQLite** in production using Litestream.

Hungryserv: aggressively unfederated

Each bridge belongs to a single user, so federation is unnecessary.

When your DAG is a linked-list, don't store it as a DAG.

Being unfederated means that:

- we can avoid implementing the state resolution algorithm,
- event storage requires less metadata,
- deletions can be optimized, and
- infinite backfill becomes an simple batch SQL operation.

Hungryserv: aggressively unfederated

Each bridge belongs to a single user, so federation is unnecessary.

> *When your DAG is a linked-list, don't store it as a DAG.*

Being unfederated means that:

- we can avoid implementing the state resolution algorithm,
- event storage requires less metadata,
- deletions can be optimized, and
- infinite backfill becomes an simple batch SQL operation.

Hungryserv: aggressively unfederated

Each bridge belongs to a single user, so federation is unnecessary.

> *When your DAG is a linked-list, don't store it as a DAG.*

Being unfederated means that:

- we can avoid implementing the state resolution algorithm,
- event storage requires less metadata,
- deletions can be optimized, and
- infinite backfill becomes an simple batch SQL operation.

Hungryserv: goals, and non-goals

Goals

- Be fast
- Be compatible with Beeper clients (and mostly compatible with Element clients)
- Be API-compatible with the Appservice API for bridges

Non-goals

- Be federated
- Be fully spec compliant

Hungryserv: goals, and non-goals

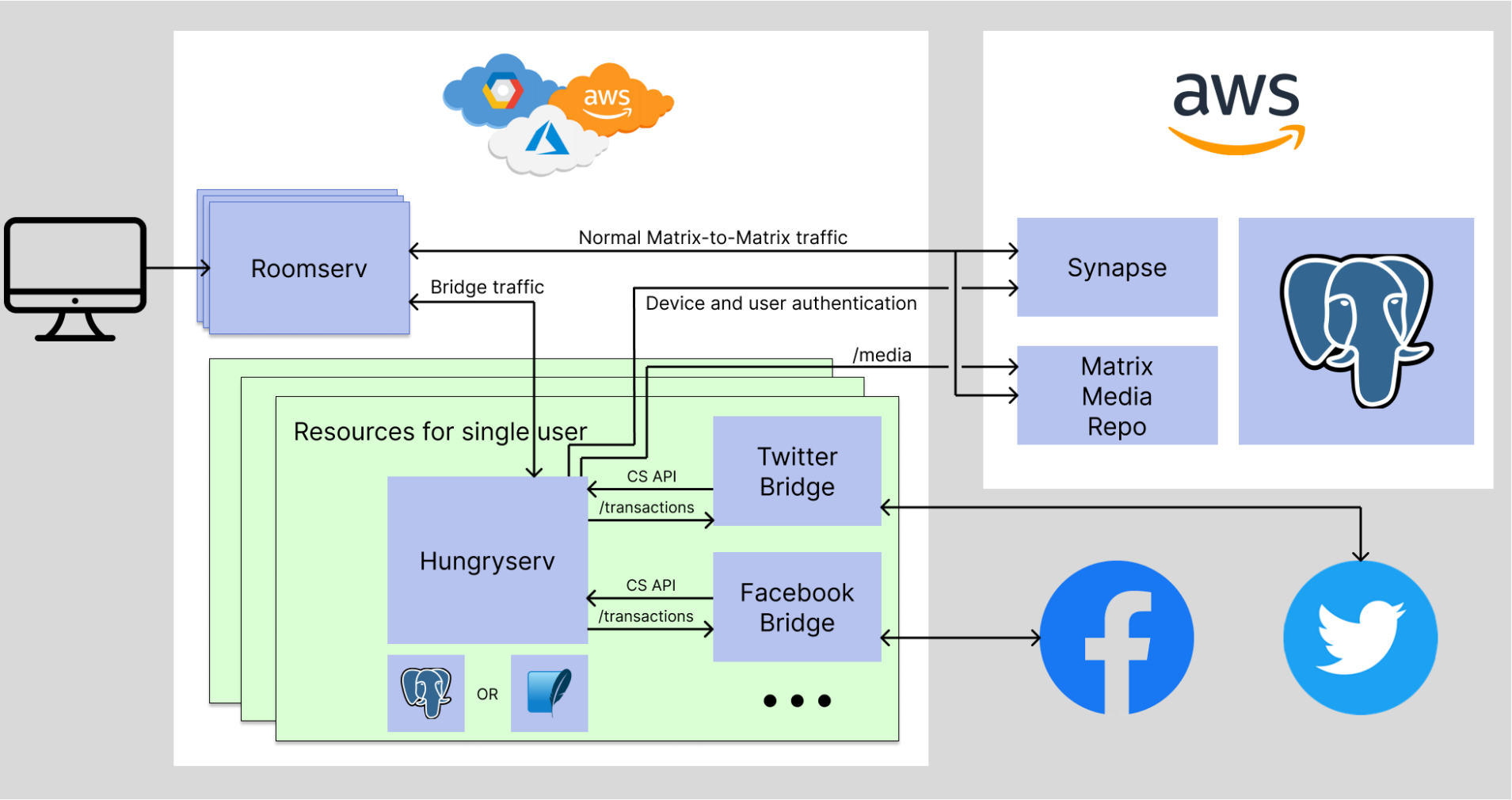
Goals

- Be fast
- Be compatible with Beeper clients (and mostly compatible with Element clients)
- Be API-compatible with the Appservice API for bridges

Non-goals

- Be federated
- Be fully spec compliant

A sharded future



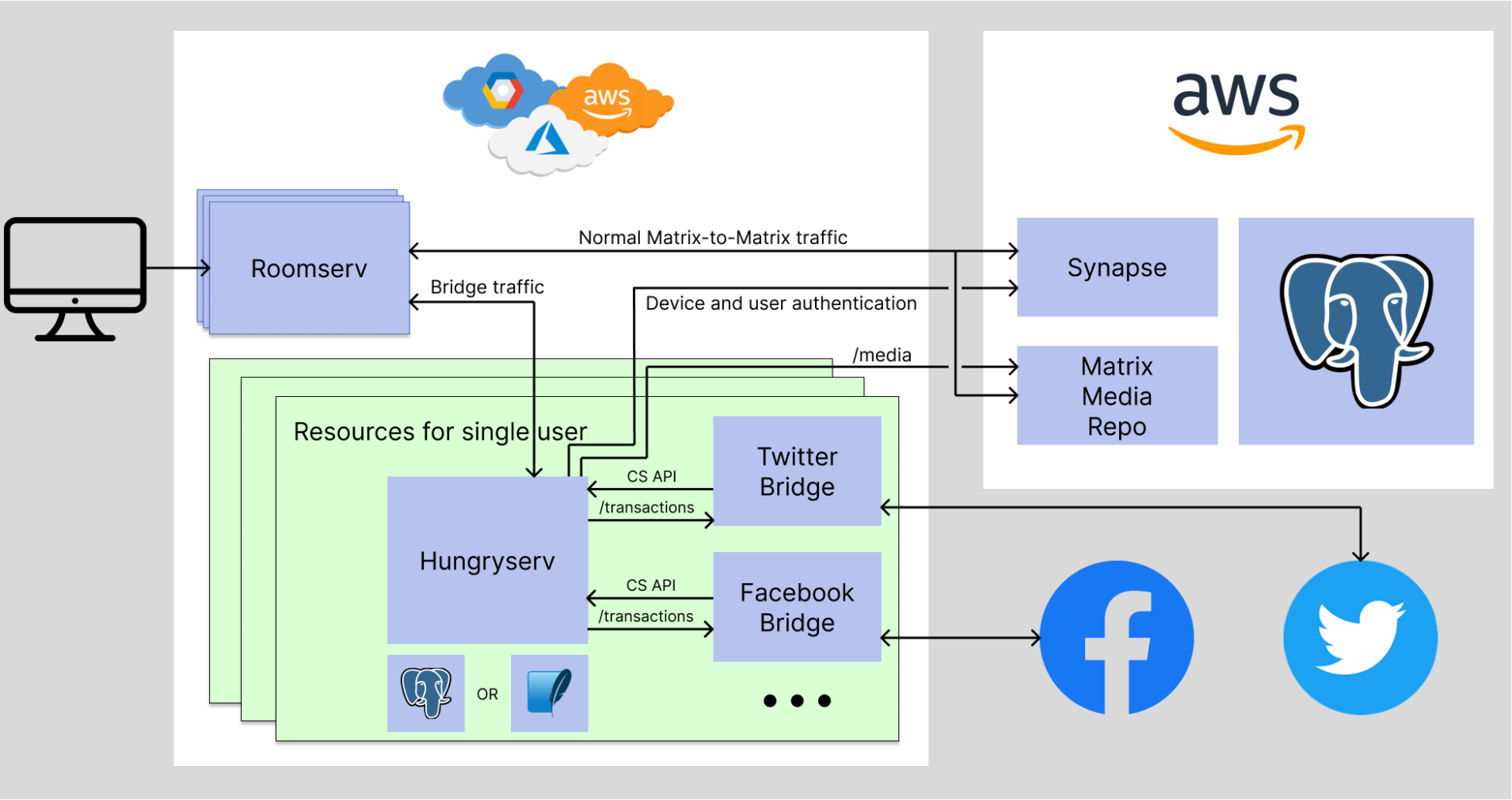
Let's talk about the differences from the old architecture...

Roomserv

Roomserv is a service which splits client requests between Synapse and Hungryserv.

Roomserv merges the responses from the two homeservers to make it appear as a single homeserver to the client.

A sharded future



Let's talk about the differences from the old architecture...

No more ASMUX

Hungryserv natively supports dynamic registration of app services, and knows how to route requests to the correct bridge.

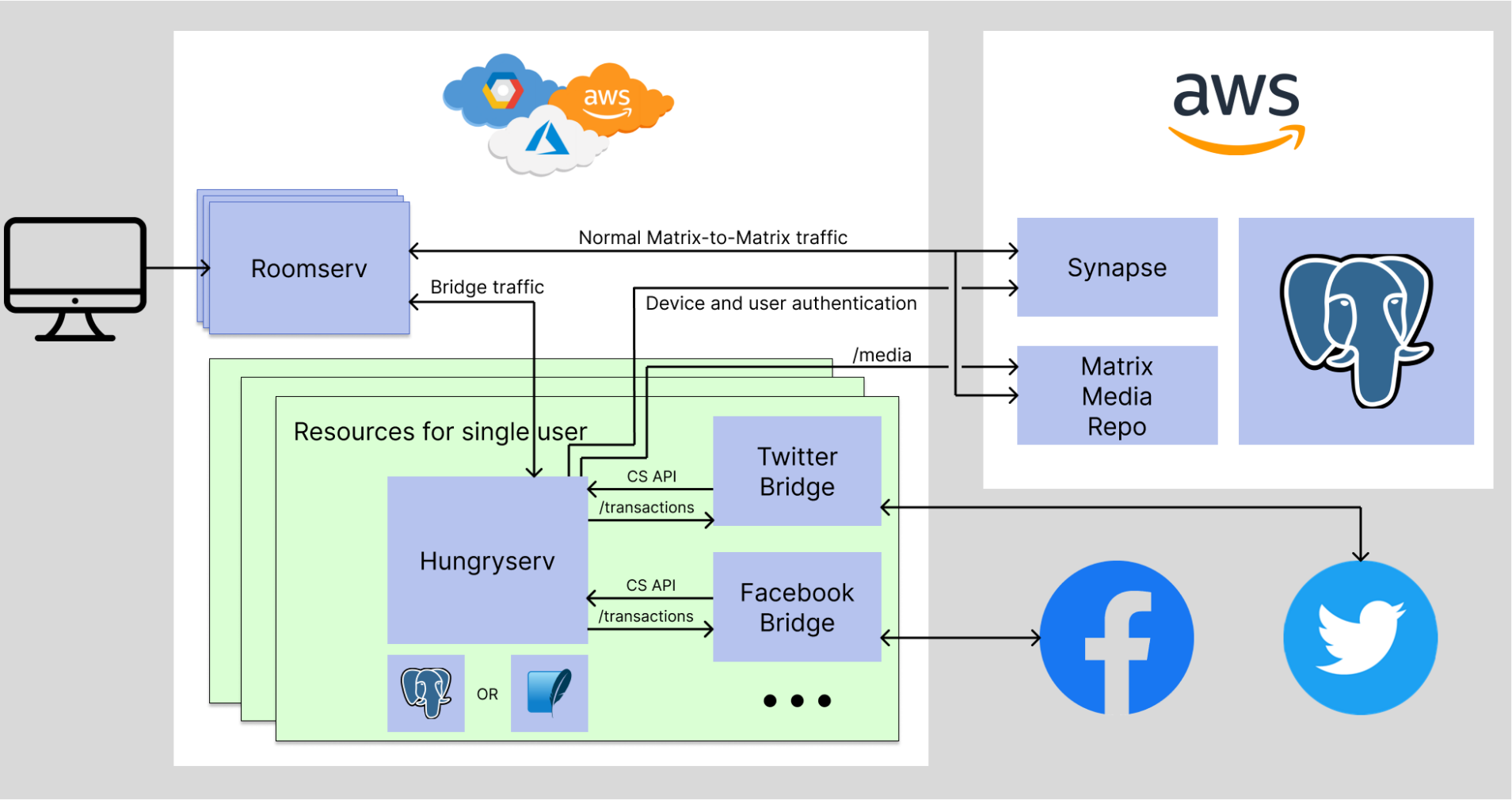
We no longer have to rely on the single transactions firehose from Synapse.

No more ASMUX

Hungryserv natively supports dynamic registration of app services, and knows how to route requests to the correct bridge.

We no longer have to rely on the single transactions firehose from Synapse.

A sharded future



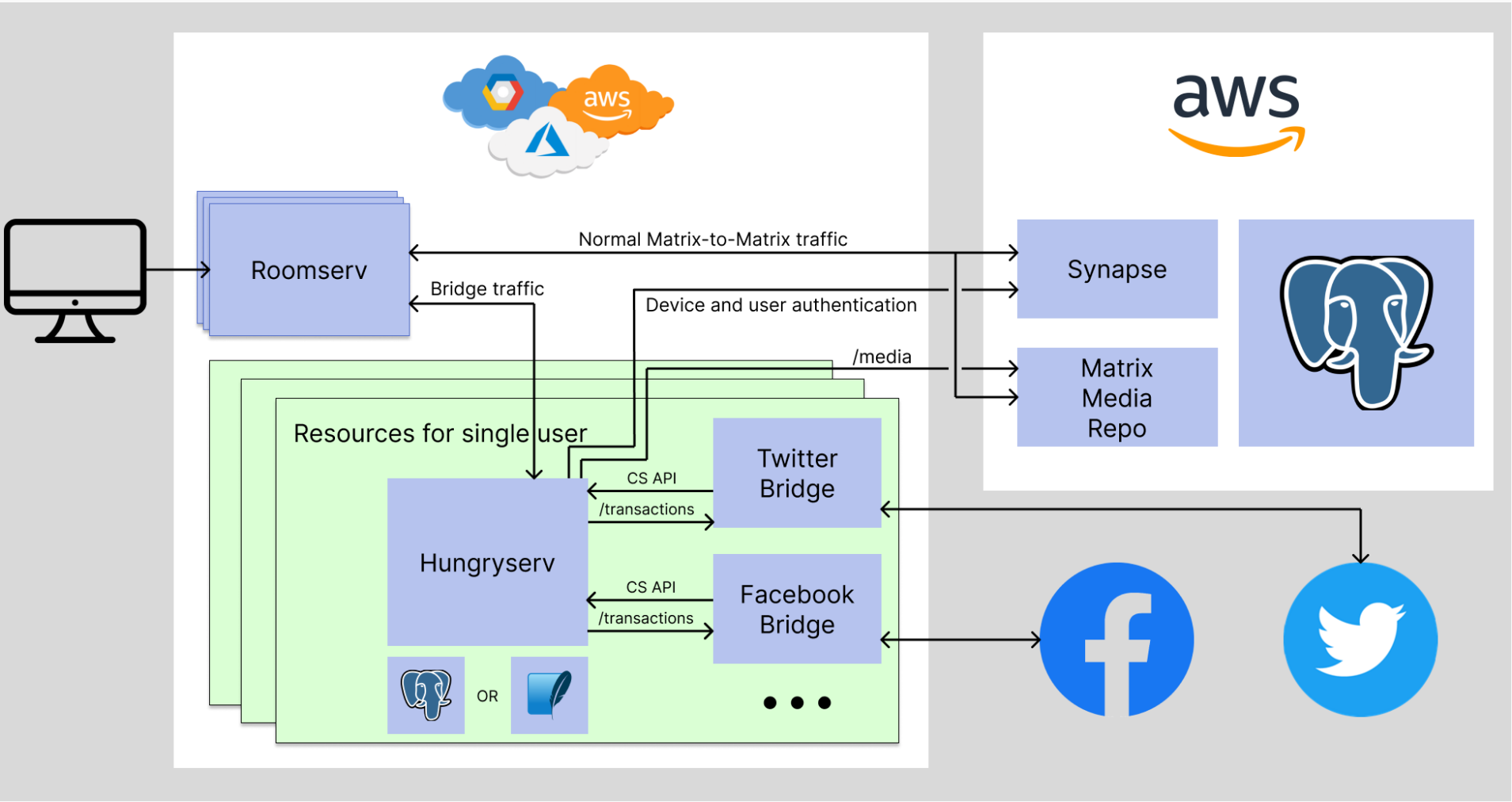
Let's talk about the differences from the old architecture...

A multi-cloud future?

The resources for each user can be moved closer to them using a multi-cloud, or multi-region strategy.

This will help reduce latency (especially to bridged networks) and allow for us to deploy in more flexible and cost-effective environments.

A sharded future



Let's talk about the differences from the old architecture...

Demo!

Questions?