LOCAL-FIRST

STEPHEN WHITE

CADEC 2025.01.23 & 2025.01.29 | CALLISTAENTERPRISE.SE

CALLISTA

LOCAL-FIRST - CONTENTS

PROBLEM STATEMENT:

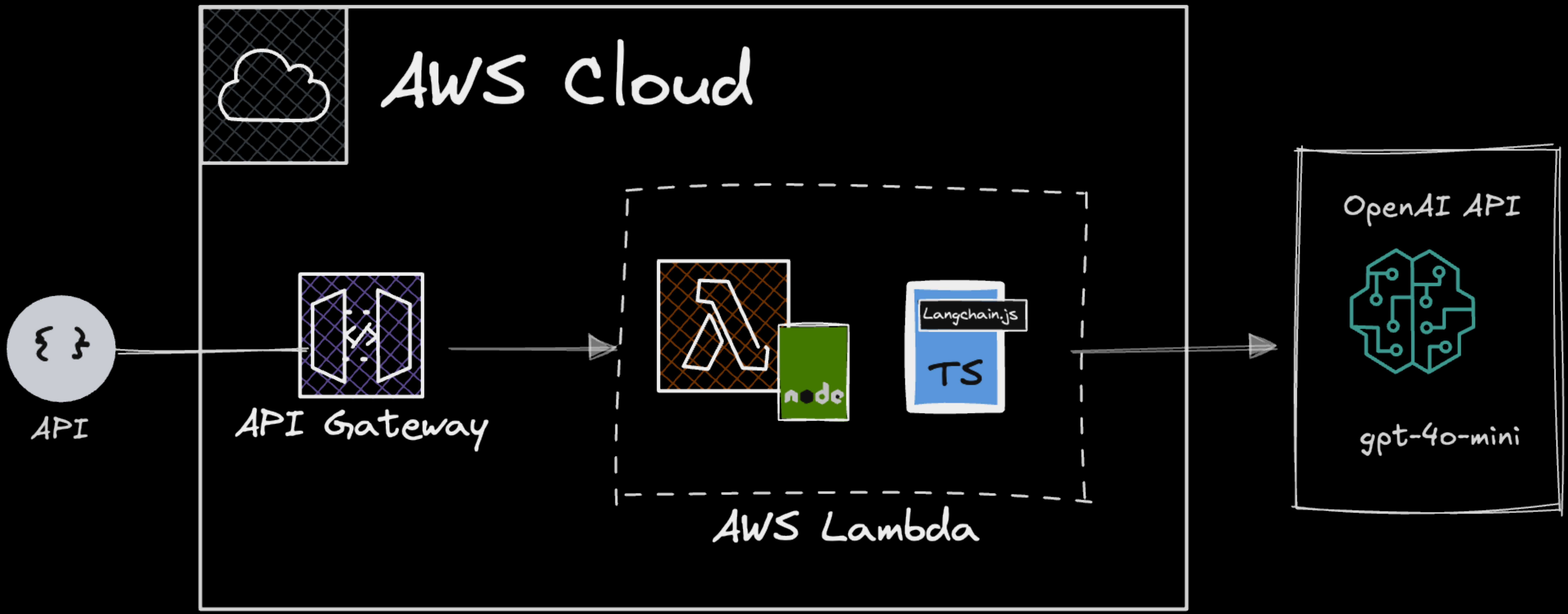
Q&A sessions after conference talks can sometimes get derailed by irrelevant or off-topic questions.

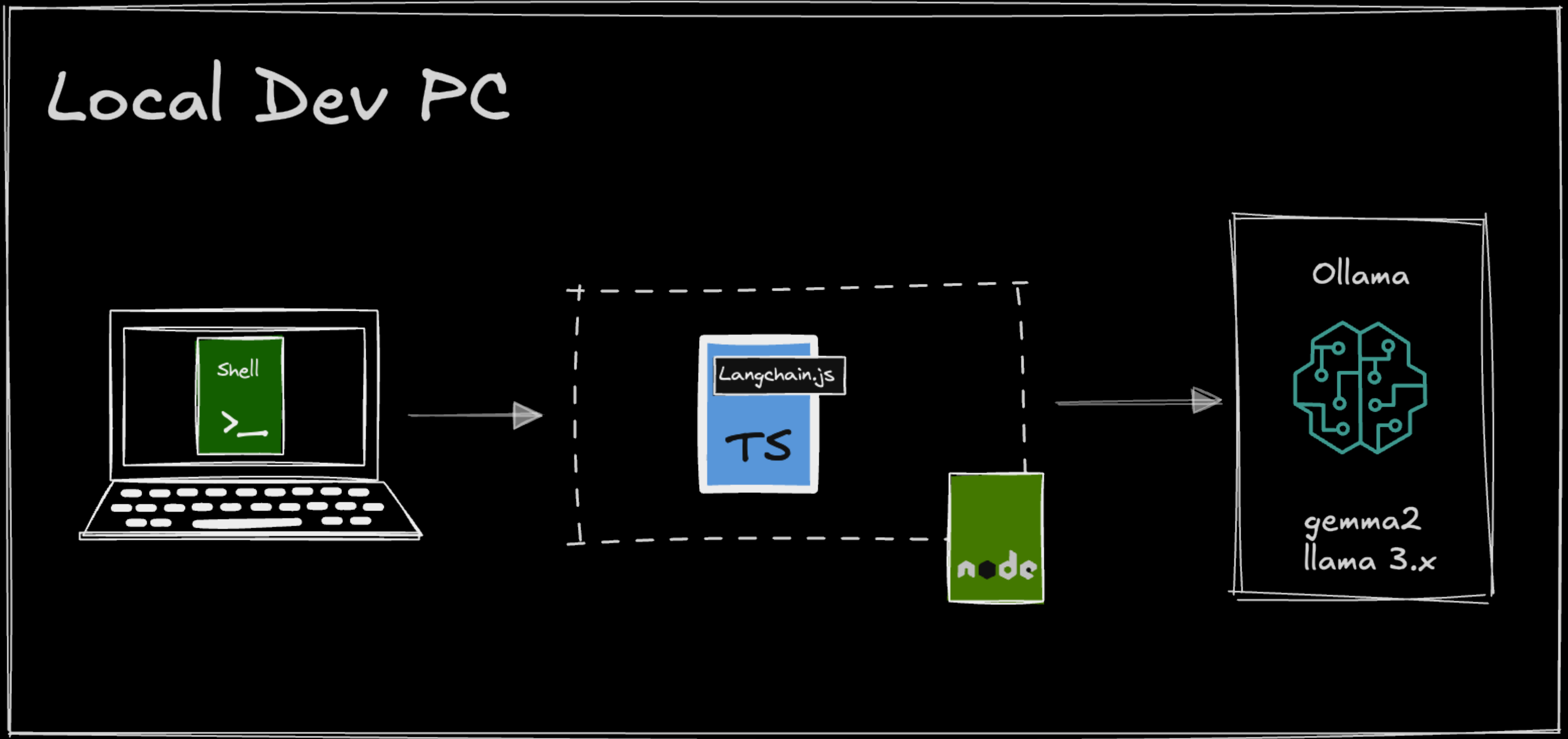
Using AI – is it possible to moderate Q&A questions automatically?

CADEC MODERATION APP

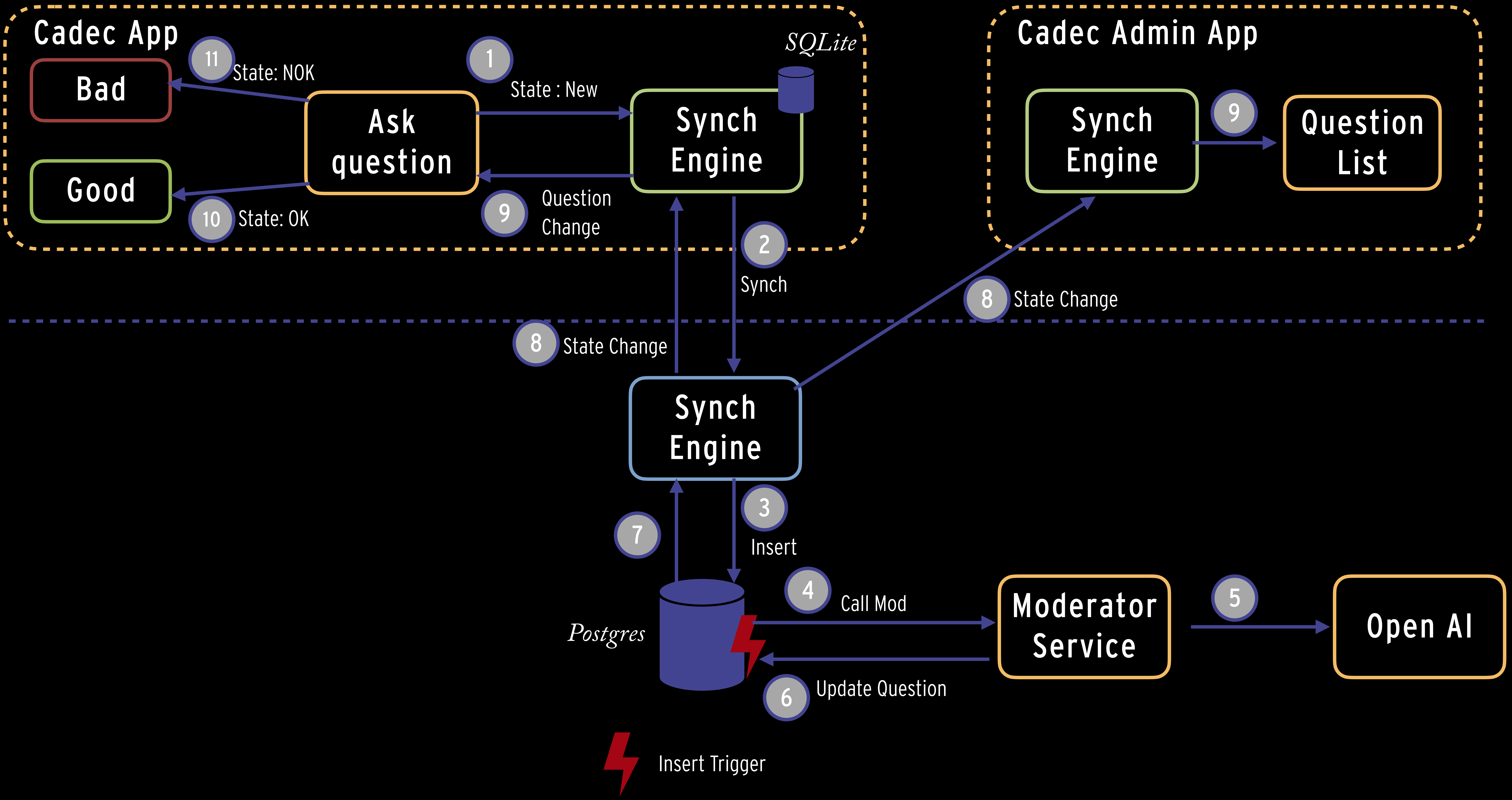
- The app analyzes submitted questions to check:
  - Whether the question contains profanity or offensive language.
  - **Relevance** to the talk topic
    - provides brief statement.
    - gives a relevance score (not visible in UI)
  - **Sentiment** (positive, neutral, negative).
  - Suggests a (brief) **answer.**

HOW IT WORKS


AWS Cloud
API
API Gateway
node
Langchain.js
TS
AWS Lambda
OpenAI API
gpt-4o-mini


Local Dev PC
Shell
>_
Langchain.js
TS
node
Ollama
gemma2
llama 3.x

CADEC APP


Cadec App
Bad
11
State: NOK
1
State : New
SQLite
Ask question
Synch Engine
Good
10
State: OK
9
Question Change
2
Synch
Cadec Admin App
Synch Engine
9
Question List
8
State Change
8
State Change
Synch Engine
3
Insert
7
4
Call Mod
Postgres
Moderator Service
5
Open AI
6
Update Question
Insert Trigger

WHAT - LOCAL-FIRST

a set of principles for software that enables both collaboration and ownership for users.

Local-first ideals include the ability to work offline and collaborate across multiple devices,

while also improving the security, privacy, long-term preservation, and user control of data.

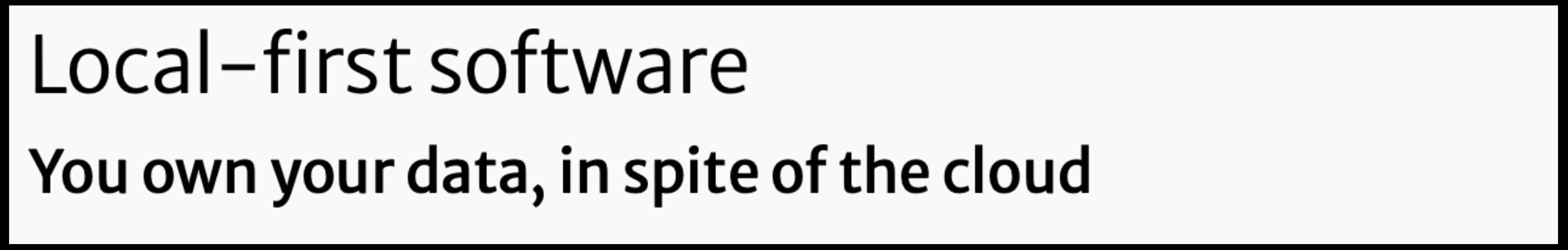
CALLISTA

https://www.inkandswitch.com/local-first/

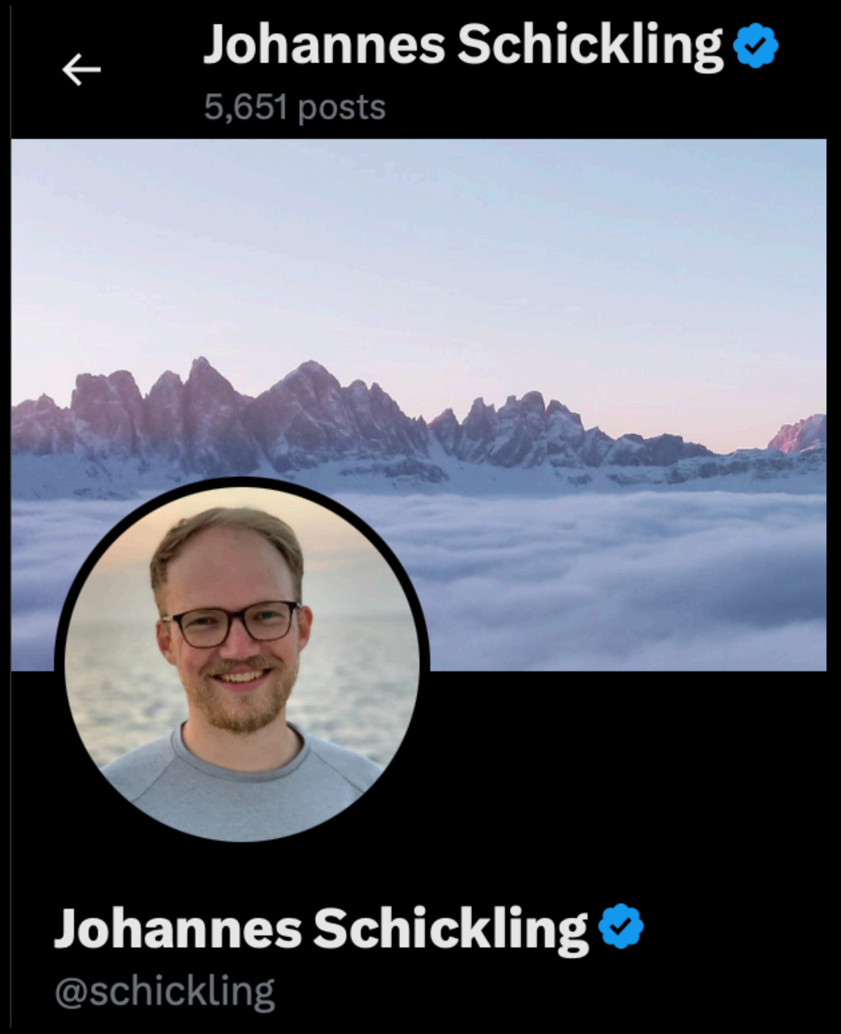
WHAT - WHO

Ink & Switch

An independent research lab exploring the future of tools for thought.


Local-first software

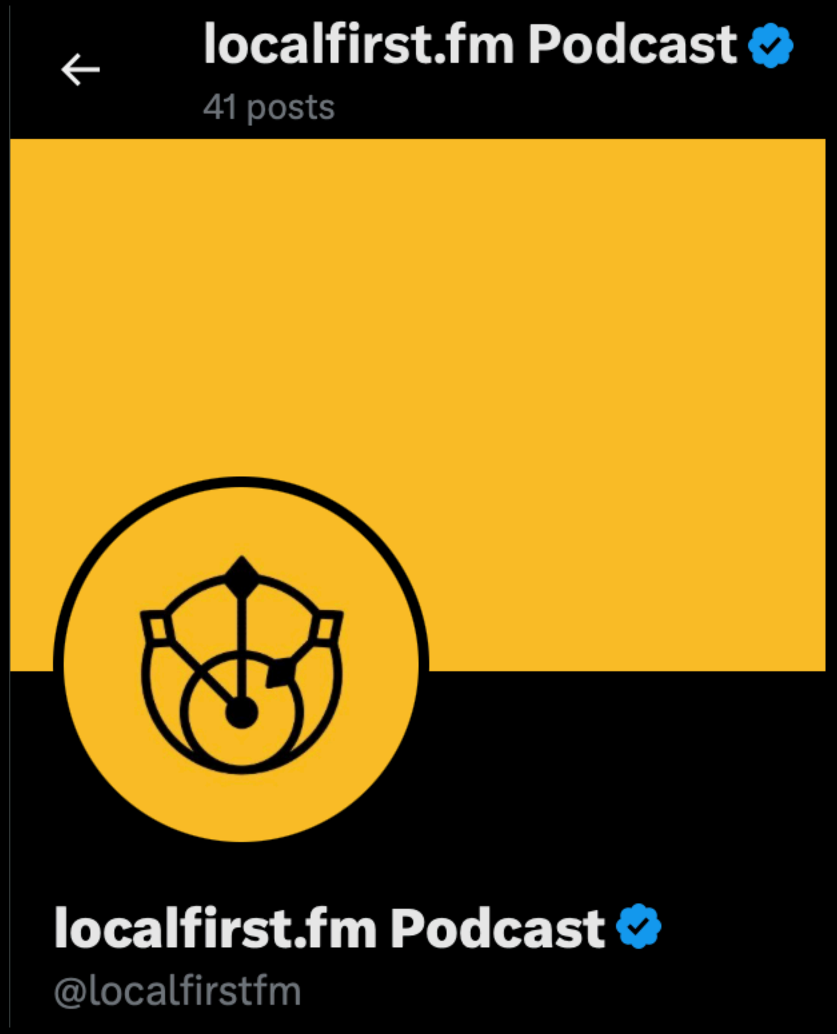
You own your data, in spite of the cloud


Johannes Schickling

5,651 posts

Johannes Schickling

@schickling


localfirst.fm Podcast

41 posts

localfirst.fm Podcast

@localfirstfm

WHAT - APPS

Google Docs

Goodnotes

miro

Linear

Figma

Apple notes

CADEC

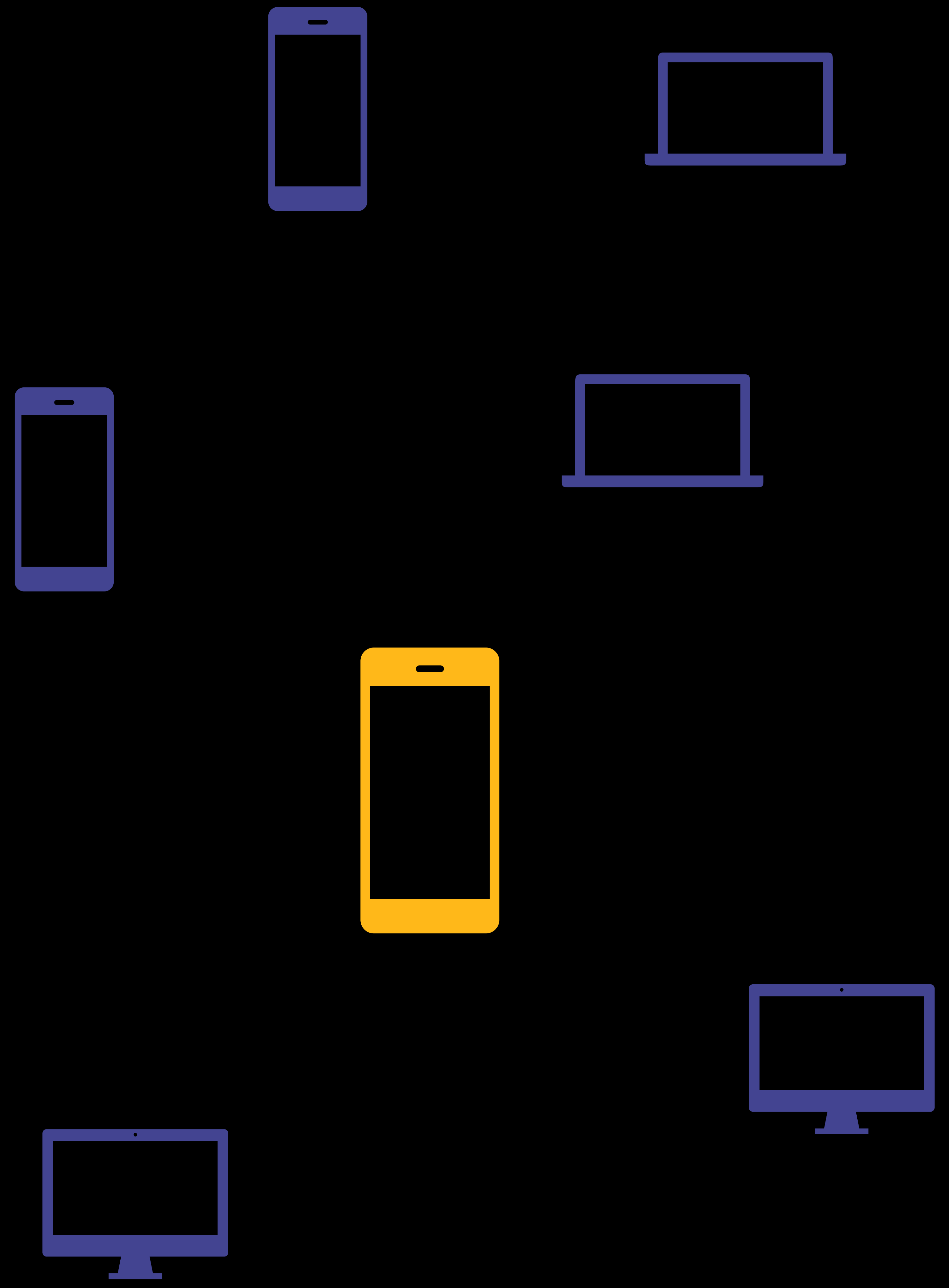
Cadec

WHAT - THE 7 POINTS OF LOCAL-FIRST

1. No spinners - *Fast*
2. Your work is not trapped on one device - *Multi-device*
3. The network is optional - *Offline*
4. Seamless collaboration with your colleagues - *Collaboration*
5. The Long now - *Longevity*
6. Security and privacy by default - *Privacy*
7. You retain ultimate ownership and control - *User control*

WHAT - THE 7 POINTS OF LOCAL-FIRST

1. No spinners - *Fast*
2. Your work is not trapped on one device - *Multi-device*
3. The network is optional - *Offline*
4. Seamless collaboration with your colleagues - *Collaboration*
5. The Long now - *Longevity*
6. Security and privacy by default - *Privacy*
7. You retain ultimate ownership and control - *User control*



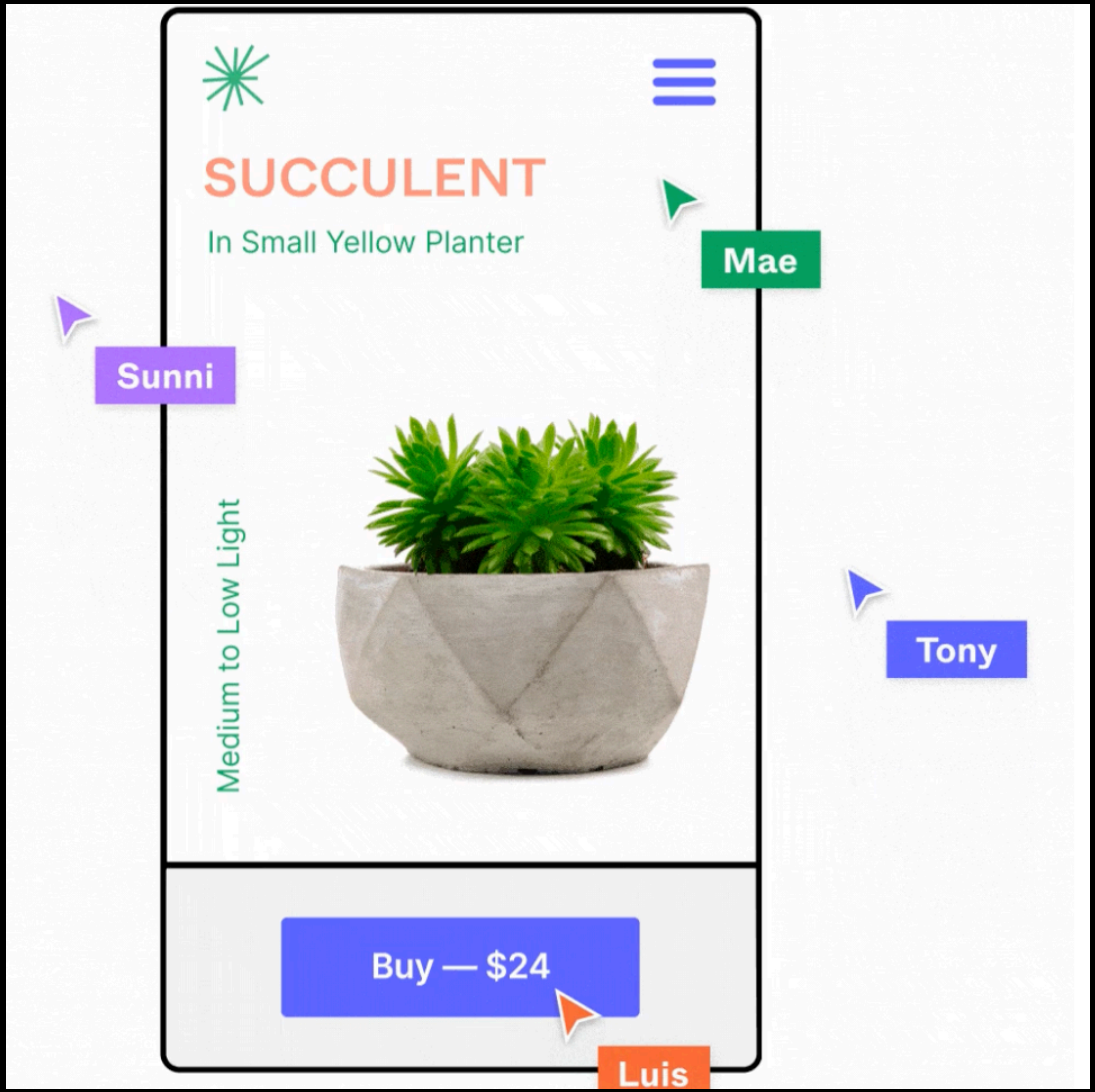
CALLISTA

WHAT - THE 7 POINTS OF LOCAL-FIRST

1. No spinners - *Fast*
2. Your work is not trapped on one device - *Multi-device*
3. The network is optional - *Offline*
4. Seamless collaboration with your colleagues - *Collaboration*
5. The Long now - *Longevity*
6. Security and privacy by default - *Privacy*
7. You retain ultimate ownership and control - *User control*

WHAT - THE 7 POINTS OF LOCAL-FIRST

1. No spinners - *Fast*
2. Your work is not trapped on one device - *Multi-device*
3. The network is optional - *Offline*
4. Seamless collaboration with your colleagues - *Collaboration*
5. The Long now - *Longevity*
6. Security and privacy by default - *Privacy*
7. You retain ultimate ownership and control - *User control*



CALLISTA

WHAT - THE 7 POINTS OF LOCAL-FIRST

1. No spinners - *Fast*
2. Your work is not trapped on one device - *Multi-device*
3. The network is optional - *Offline*
4. Seamless collaboration with your colleagues - *Collaboration*
5. The Long now - *Longevity*
6. Security and privacy by default - *Privacy*
7. You retain ultimate ownership and control - *User control*

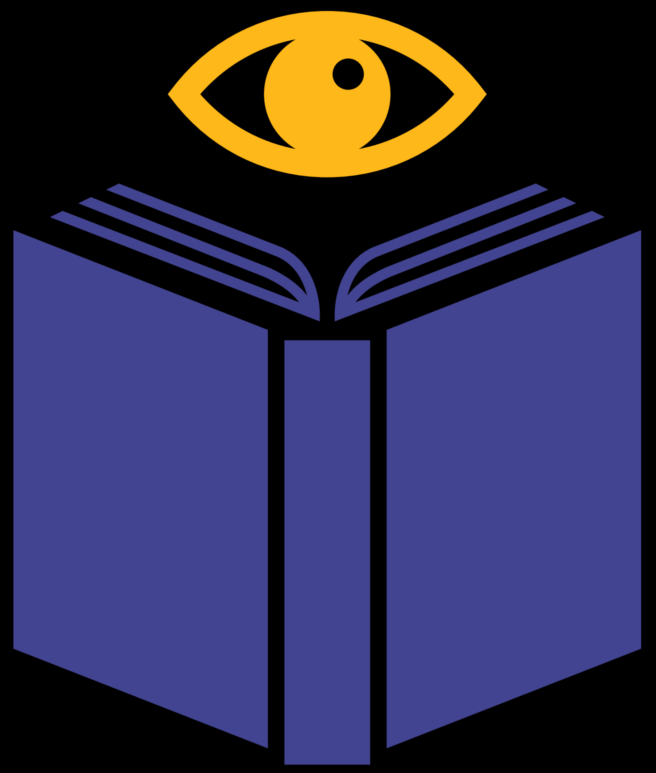
To install DOOM, type INSTALL and press ENTER
DOOM
REGISTERED
DISK 3
Created by id Software V 1.2
©1993 id Software, Inc. All Rights Reserved.
DISK 4
DISK 1
DISK 2

https://en.wikipedia.org/wiki/Digital_dark_age

CALLISTA

WHAT - THE 7 POINTS OF LOCAL-FIRST

1. No spinners - *Fast*
2. Your work is not trapped on one device - *Multi-device*
3. The network is optional - *Offline*
4. Seamless collaboration with your colleagues - *Collaboration*
5. The Long now - *Longevity*
6. Security and privacy by default - *Privacy*
7. You retain ultimate ownership and control - *User control*

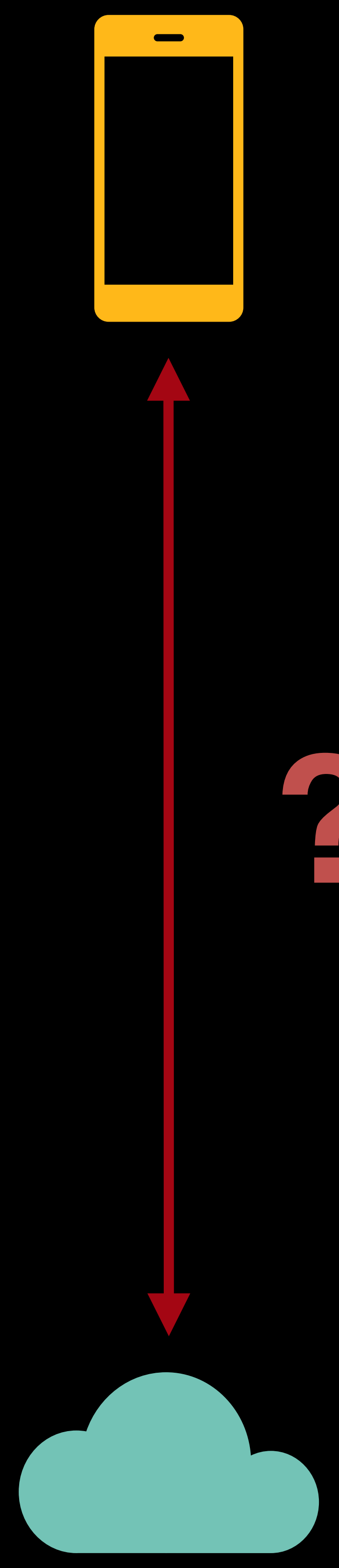


WHAT - THE 7 POINTS OF LOCAL-FIRST

1. No spinners - *Fast*
2. Your work is not trapped on one device - *Multi-device*
3. The network is optional - *Offline*
4. Seamless collaboration with your colleagues - *Collaboration*
5. The Long now - *Longevity*
6. Security and privacy by default - *Privacy*
7. You retain ultimate ownership and control - *User control*

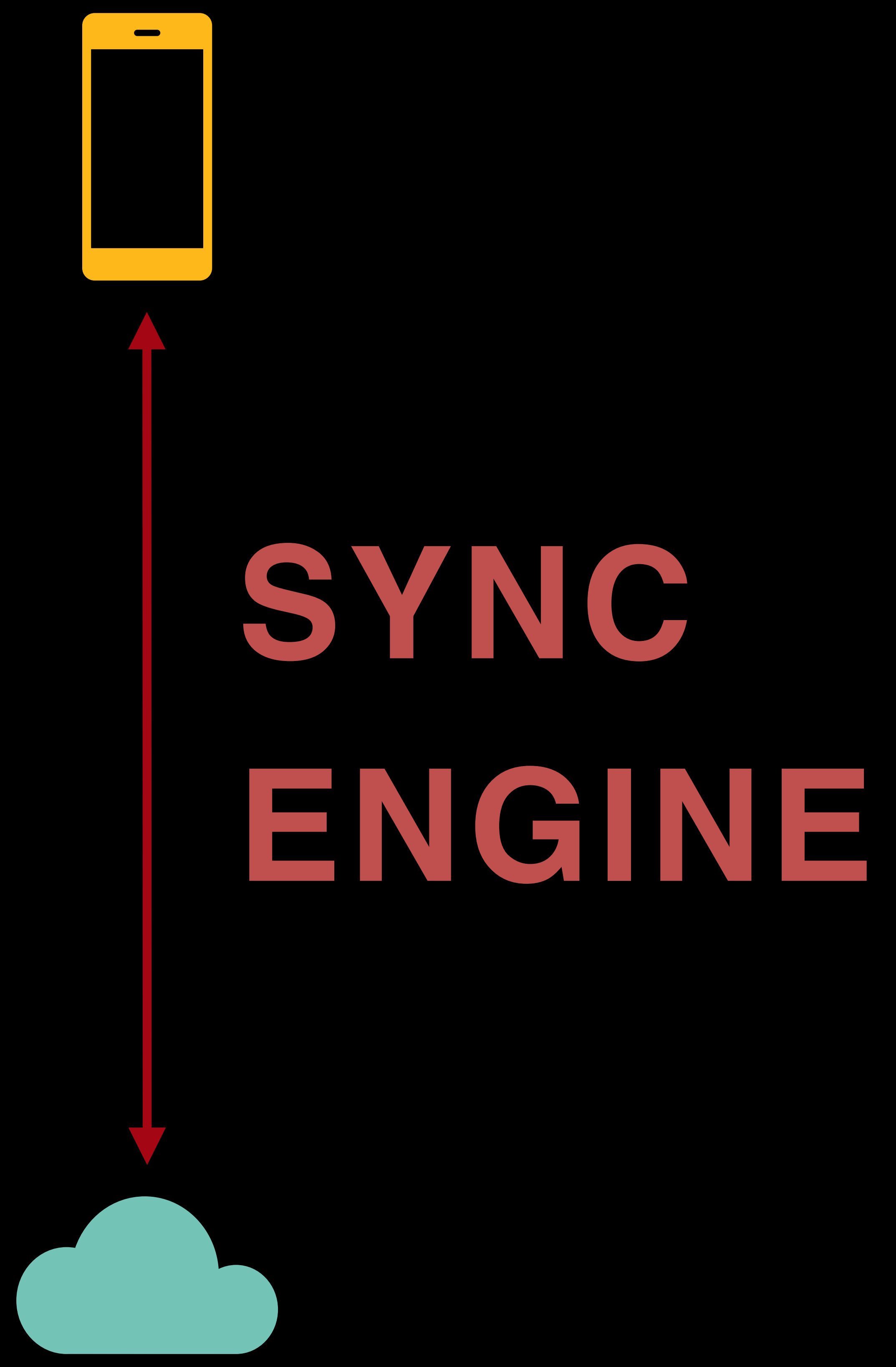
HOW - WHERE WE ARE NOW

- Request / reply
- Huge Innovations in both FE and BE
- Somethings missing?



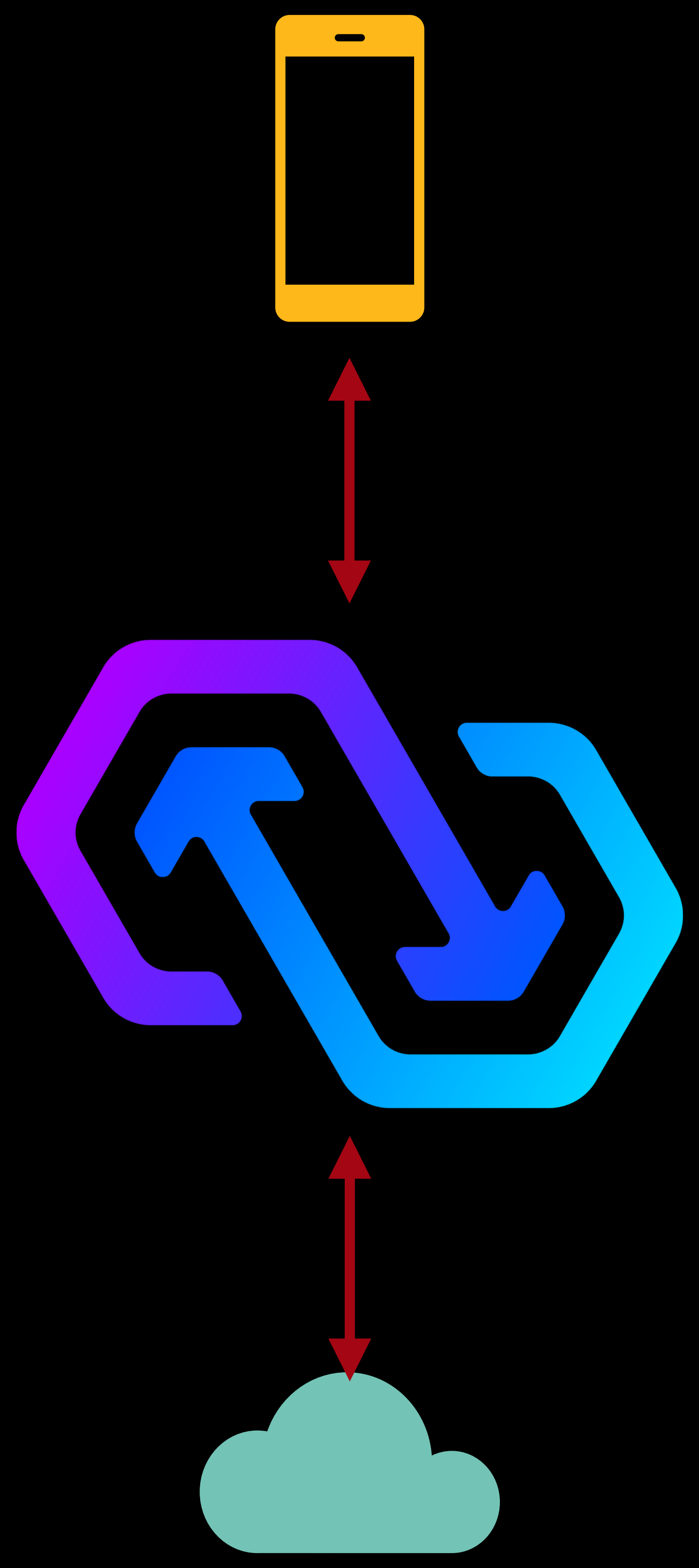
HOW - WHERE WE ARE NOW

- Request / reply
- Huge Innovations in both FE and BE
- Its the **Synch Engine!**



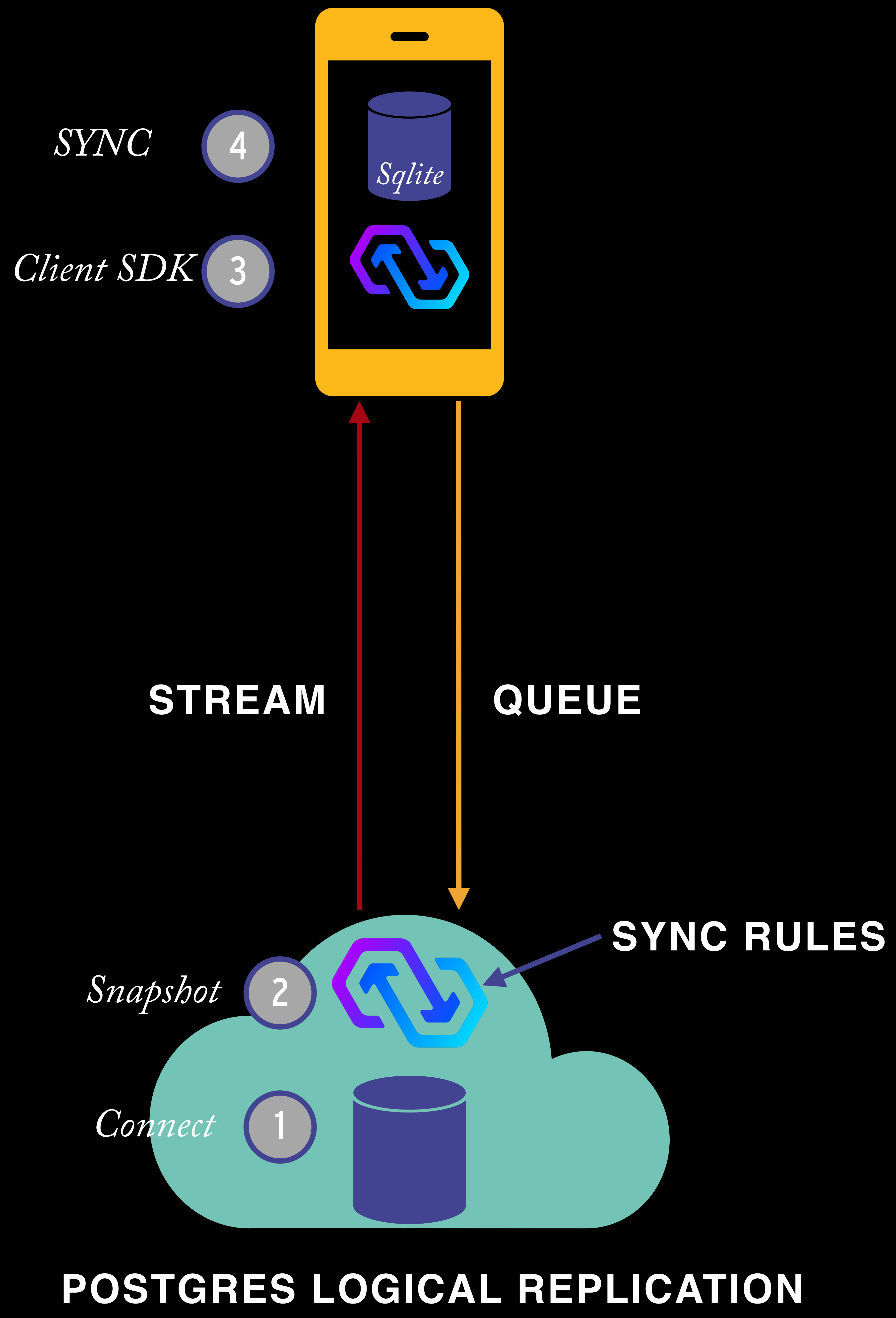
HOW - SYNCH ENGINE - POWER SYNCH

- Solves how we shuffle data around a system
- Merging
- Conflict Free Replicated Data Type (CRDTS)
  - CF - merge algorithms for different data types
    - » Json
    - » Sets
    - » Primitives
    - » Etc ..



HOW - POWER SYNCH - SYNCH RULES

1. How does data synch
2. Sync rules - Materialised View
    1. Data queries - specifies what data is included in a bucket
    2. Parameter queries - determines which buckets should be synched with the users device
    3. Combining queries
        1. Determine which user receives which bucket

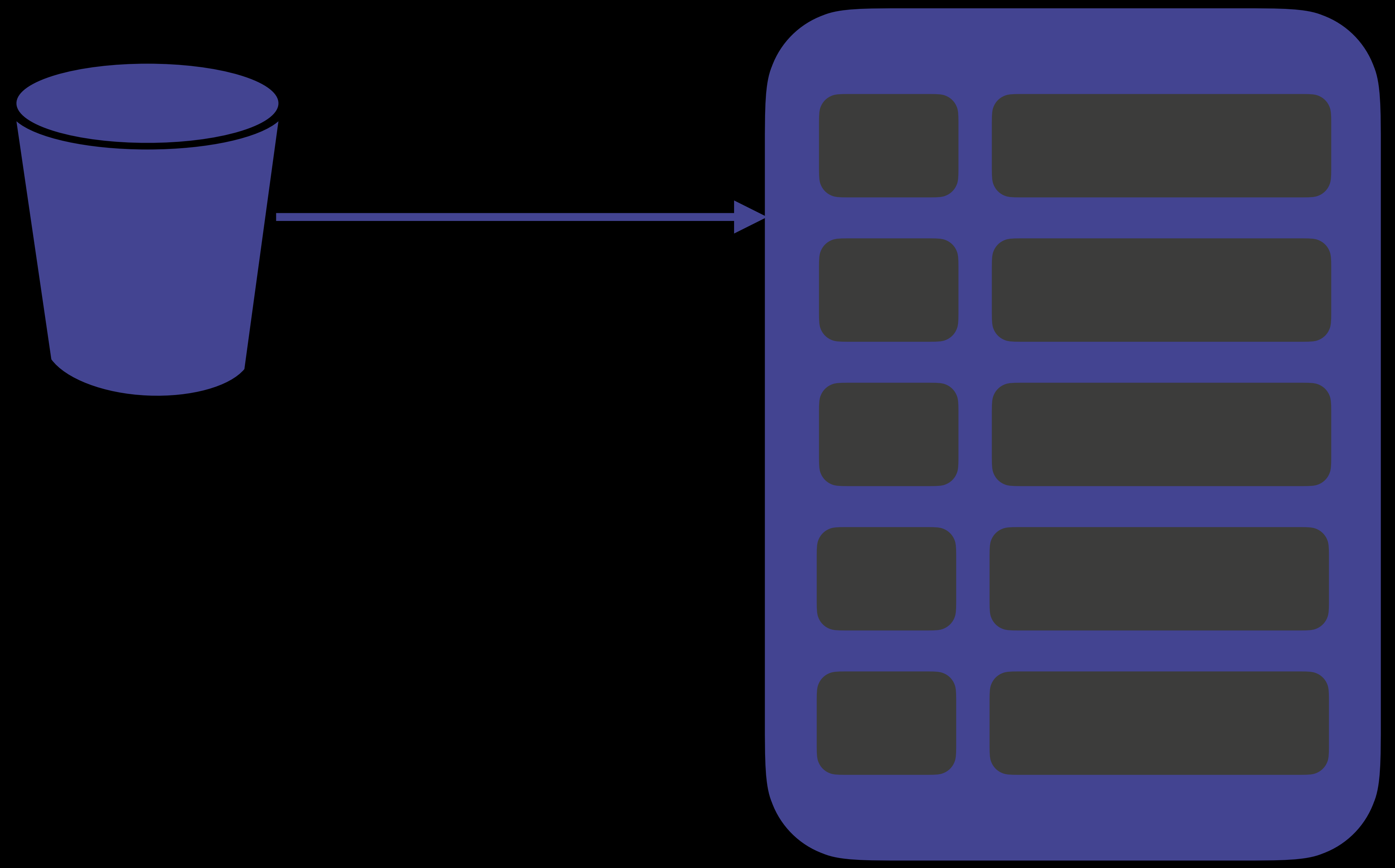


CALLISTA

HOW - POWER SYNCH - SYNCH RULES

1. How does data get on your device
2. Sync rules - Materialised View
    1. Data queries - specifies what data is included in a bucket
    2. Parameter queries - determines which buckets should be synched with the users device
    3. Combining queries
        1. Determine which user receives which bucket

```
# sync-rules.yaml
bucket_definitions:
  bucket_name: #name of bucket e.g. questions
    Parameters:# (optional) query used to determine which
                            buckets are synched
    Data: # query used to determine the data in each bucket
```

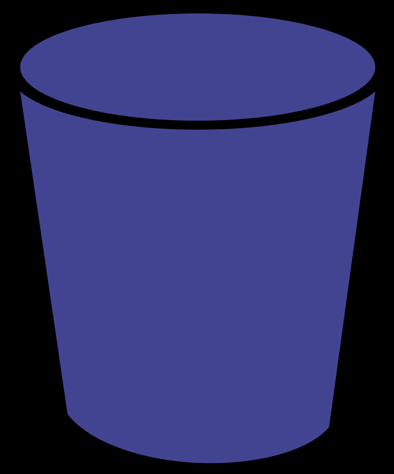


CALLISTA

HOW - POWER SYNCH - SYNCH RULES

1. How does data get on your device
2. Sync rules - Materialised View
   1. Data queries - specifies what data is included in a bucket
   2. Parameter queries - determines which buckets should be synched with the users device
   3. Combining queries
      1. Determine which user receives which bucket

```yaml
# sync-rules.yaml
bucket_definitions:
  Questions:
    data:
      - SELECT * FROM questions
```


QUESTIONS BUCKET

CALLISTA

HOW - POWER SYNCH - SYNCH RULES

1. How does data get on your device
2. Sync rules - Materialised View
    1. Data queries - specifies what data is included in a bucket
    2. Parameter queries - determines which buckets should be synched with the users device
    3. Combining queries
        1. Determine which user receives which bucket

```
# sync-rules.yaml
bucket_definitions:
  users_questions:
    Parameters:
      - SELECT request.user_id() AS id
```

From jwt token

User 1
User 2
User 3

CALLISTA

HOW - POWER SYNCH - SYNCH RULES

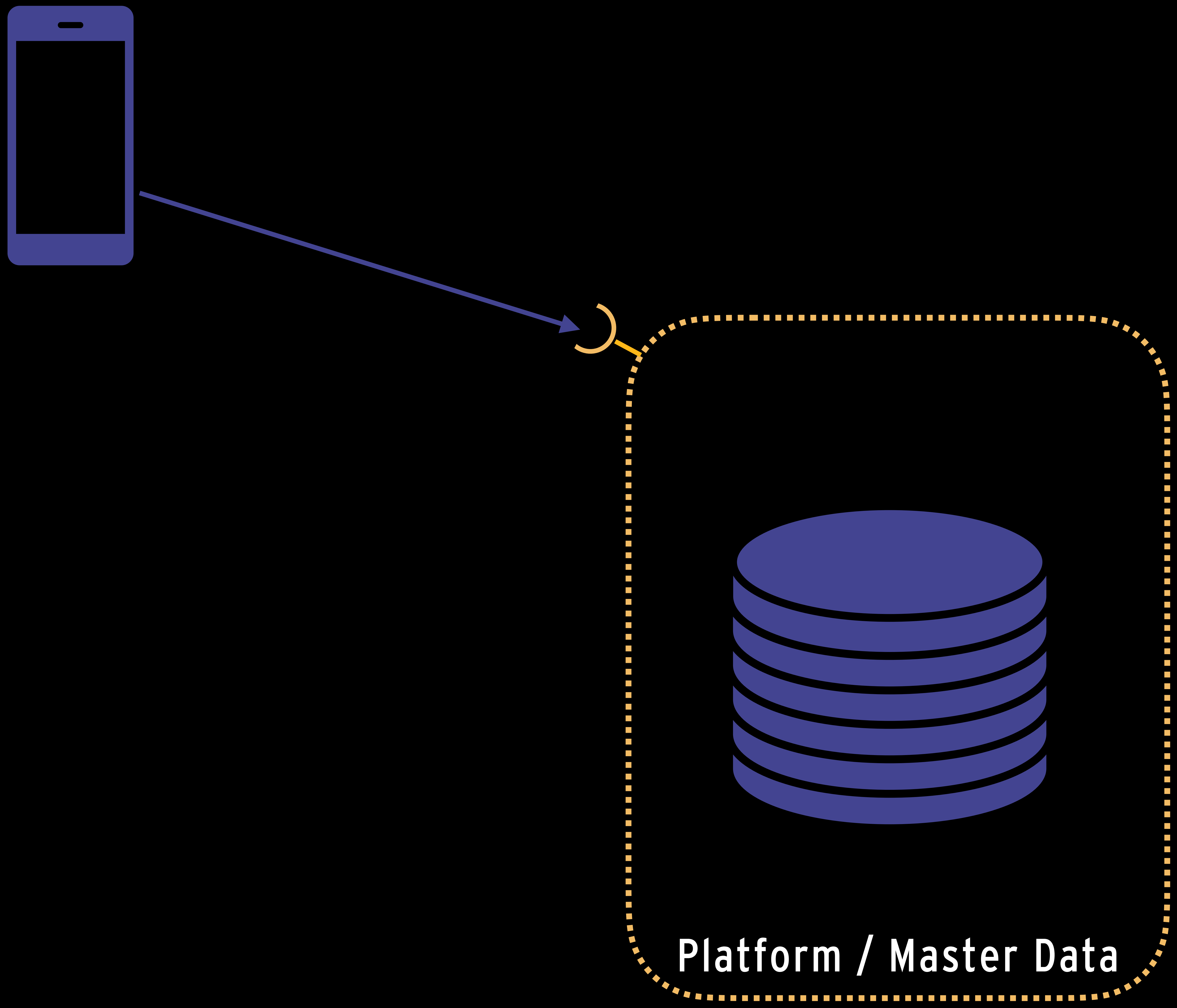
1. How does data get on your device
2. Sync rules - Materialised View
   1. Data queries - specifies what data is included in a bucket
   2. Parameter queries - determines which buckets should be synched with the users device
   3. Combining queries
      1. Determine which user receives which bucket

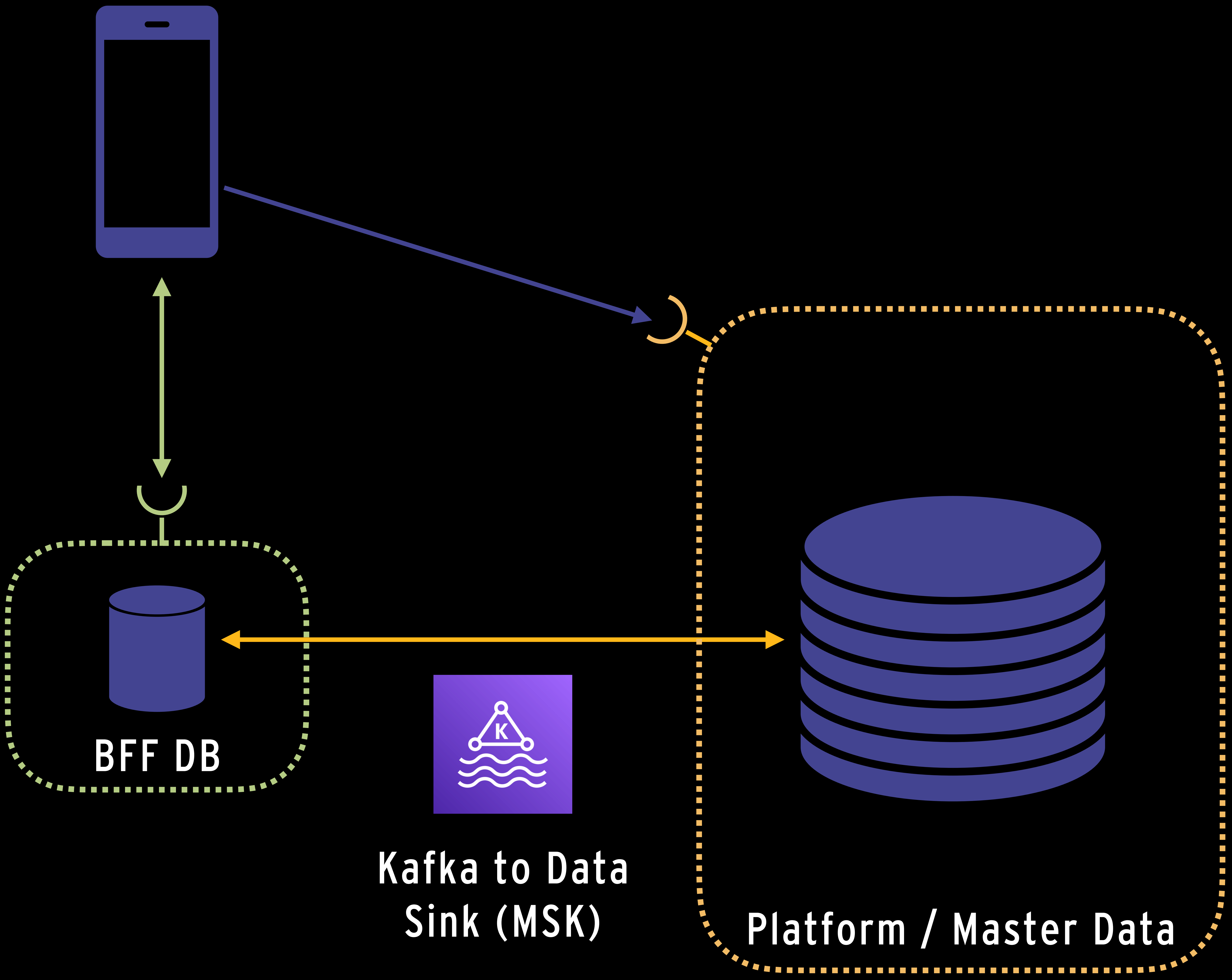
```
# sync-rules.yaml
bucket_definitions:
  users_questions:
    parameters:
      - SELECT request.user_id() AS id
    Data:
      - SELECT * from questions
            WHERE owner_id = bucket.id
```

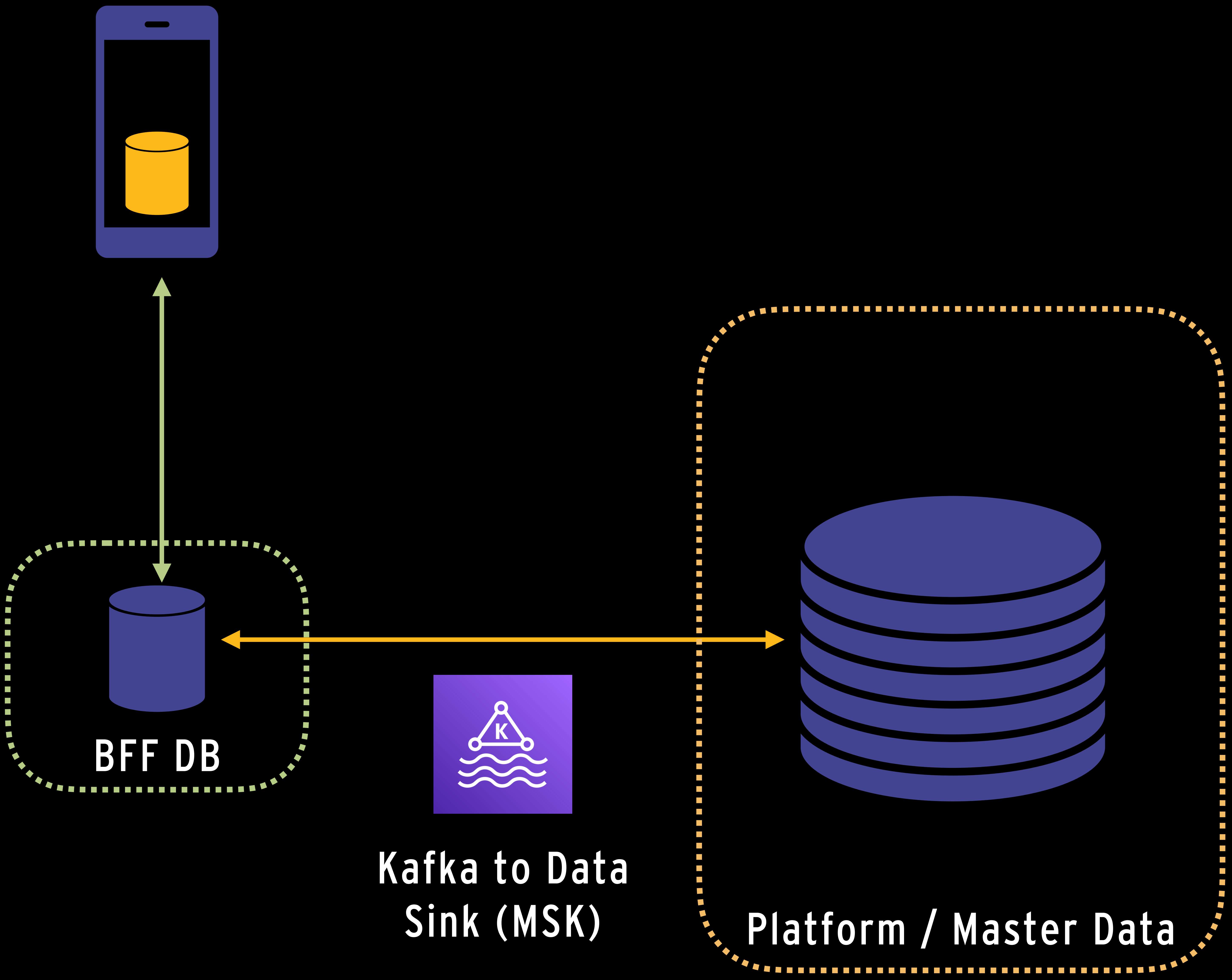
User 1
questions

User 2
questions

CALLISTA


Platform / Master Data


BFF DB
Kafka to Data Sink (MSK)
Platform / Master Data


BFF DB
Kafka to Data Sink (MSK)
Platform / Master Data

HOW - CADEC-APP - TECH STACK

ElectricSQL
ElectricSQL team

TinyBase
James Pearce

PouchDB
PouchDB contributors

Prisma

drizzle

Automerge
Ink & Switch and contributors

WatermelonDB
Radek Pietruszewski / Nozbe

Amplify DataStore
Amazon Web Services

Legend State
Jay Meistrich

Yjs
Kevin Jahns & contributors

PowerSync
JourneyApps

EXPO

LiveStore
Early Access

HOW - CADEC-APP - TECH STACK

EXPO

DRIZZLE-ORM

PowerSync

supabase

DEVELOPMENT EXPERIENCE

1. Drizzle Server -
    1. Define Schema
    2. Migrate / Seed
2. Power Synch - Define Synch Rules
3. Drizzle Client -
    1. Define Schema
    2. Create Hook
        1. Query
            1. Aggregate
        2. Mutation
    3. Use Hook in Component

```
export const talks = pgTable("talk", {
  id: uuid().defaultRandom().notNull().primaryKey(),
  created_at: timestamp().defaultNow(),
  title: varchar('250'),
  description: varchar('1000'),
});

export const talksRelations = relations(talks, ({ many }) =>
({
  locationsToTalks: many(locationsToTalks),
  speakers: many(speakers),
  questions: many(questions),
  votes: many(voteCounts),
}));
```

DEVELOPMENT EXPERIENCE

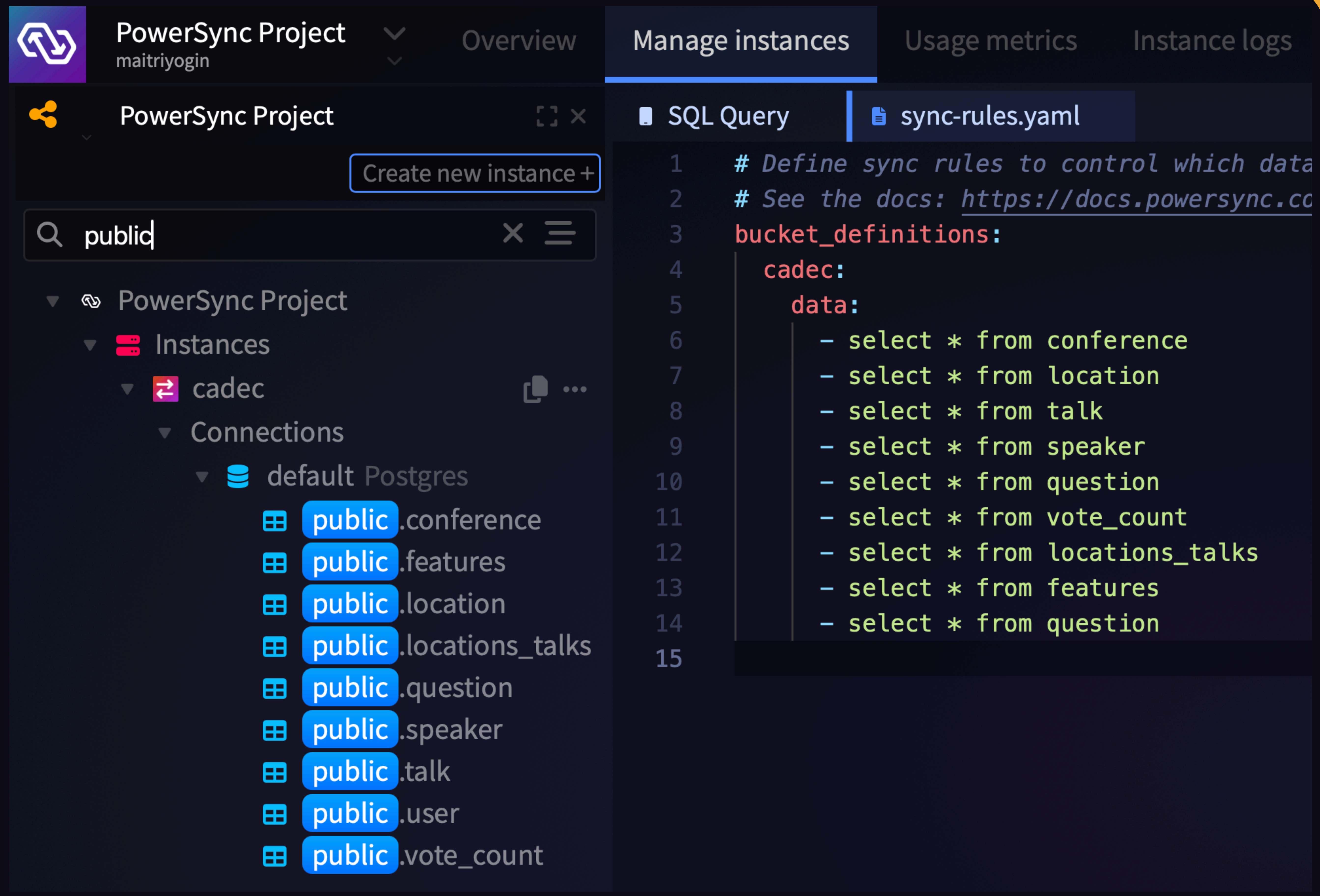
1. Drizzle Server -
   1. Define Schema
   2. Migrate / Seed
2. Power Synch - Define Synch Rules
3. Drizzle Client -
   1. Define Schema
   2. Create Hook
      1. Query
         1. Aggregate
      2. Mutation
   3. Use Hook in Component

```
> npx drizzle-kit migrate

export default defineConfig({
  out: "./drizzle",
  schema: "./src/db/schema.ts",
  dialect: "postgresql",
  dbCredentials: {
    url: process.env.DATABASE_URL!,
  },
});

> npx drizzle-kit push

> bun src/index.ts

// index.ts
async function main() {
  // clear db
  console.log("--- deleting data");
  await db.delete(speakers);
  …
  // insert
  console.log("--- inserting data");
  await db.insert(speakers)
          .values(data.speakers).onConflictDoNothing();

// data.ts
export const speakers: ISpeaker[] = [
  {
    id: ids.speaker1,
    talkId: ids.talk1,
    name: "Stephen White",
  },
```

CALLISTA

DEVELOPMENT EXPERIENCE

1. Drizzle Server -
    1. Define Schema
    2. Migrate / Seed
2. Power Synch - Define Synch Rules
3. Drizzle Client -
    1. Define Schema
    2. Create Hook
        1. Query
            1. Aggregate
        2. Mutation
    3. Use Hook in Component



```yaml
# Define sync rules to control which data
# See the docs: https://docs.powersync.co
bucket_definitions:
  cadec:
    data:
      - select * from conference
      - select * from location
      - select * from talk
      - select * from speaker
      - select * from question
      - select * from vote_count
      - select * from locations_talks
      - select * from features
      - select * from question
```

CALLISTA

DEVELOPMENT EXPERIENCE

1. Drizzle Server -
   1. Define Schema
   2. Migrate / Seed
2. Power Synch - Define Synch Rules
3. Drizzle Client -
   1. Define Schema
   2. Create Hook
      1. Query
         1. Aggregate
      2. Mutation
   3. Use Hook in Component

```
export const talks = sqliteTable("talk", {
  id: text("id").$defaultFn(uuid).notNull(),
  created_at: text("created_at")
    .default(sql`(datetime())`)
    .notNull(),
  title: text().notNull(),
  description: text().notNull(),
});

export const talksRelations = relations(talks, ({ one,
many }) => ({
  locationsToTalks: many(locationsToTalks),
  speakers: many(speakers),
  questions: many(questions),
  votes: many(voteCounts),
}));

export const locationsToTalks = sqliteTable(
  "locations_talks",
  {
    id: text("id").$defaultFn(uuid).notNull(),
    talkId: text("talk_id")
      .notNull()
      .references(() => talks.id),
    locationId: text("location_id")
      .notNull()
      .references(() => locations.id),
  },
  (t) => ({
    pk: primaryKey({ columns: [t.talkId, t.locationId] }),
  }),
);
```

DEVELOPMENT EXPERIENCE

1. Drizzle Server -
    1. Define Schema
    2. Migrate / Seed
2. Power Synch - Define Synch Rules
3. Drizzle Client -
    1. Define Schema
    2. Create Hook
        1. Query
            1. Aggregate
        2. Mutation
    3. Use Hook in Component

```typescript
export const useTalk = (talkId: string) => {
  const system = useSystem();
  const { locationId } = useFeatures();

  const result = system.db.query.talks.findFirst({
    with: {
      votes: true,
      speakers: true,
    },
    where: eq(talks.id, talkId),
  });
  const { data } = useQuery(toCompilableQuery(result));
  return data?.length > 0 ? data[0] : undefined;
};
```

DEVELOPMENT EXPERIENCE

1. Drizzle Server -
   1. Define Schema
   2. Migrate / Seed
2. Power Synch - Define Synch Rules
3. Drizzle Client -
   1. Define Schema
   2. Create Hook
      1. Query
         1. Aggregate
      2. Mutation
   3. Use Hook in Component

```
export const useTalkVoteCount = ({ talkId }: { talkId?:
string }) => {
  const system = useSystem();
  const { locationId } = useFeatures();

  if (!talkId) return 0;

  const countQuery = system.db
    .select({ voteCount: count(voteCounts.id) })
    .from(voteCounts)
    .where(
      and(
        eq(voteCounts.userId, system.userId),
        eq(voteCounts.locationId, locationId),
        eq(voteCounts.talkId, talkId),
      ),
    );

  const { data } = useQuery(toCompilableQuery(countQuery));
  return data?.[0]?.voteCount ?? 0;
};
```

DEVELOPMENT EXPERIENCE

```
const qs = `select q.id, q.state, t.id as talkId, t.title as talkTitle, q.question, q.user_id,
q.location_id,

      ( SELECT  COUNT(*) FROM vote_count vcq WHERE vcq.question_id = q.id AND
         vcq.user_id ='${system.userId}' AND
         vcq.location_id='${locationId}' ) AS yourQuestionVotes,
      ( SELECT  COUNT(*) FROM vote_count vcq WHERE vcq.question_id = q.id AND
         vcq.location_id='${locationId}' ) AS totalQuestionVotes,
      ( SELECT  COUNT(*) FROM vote_count vcq WHERE vcq.talk_id= t.id AND
         vcq.location_id='${locationId}' AND vcq.user_id='${system.userId}') AS yourTalkVotes,
      ( SELECT  COUNT(*) FROM vote_count vcq WHERE vcq.talk_id= t.id AND
         vcq.location_id='${locationId}') AS totalTalkVotes,

      from question q inner join talk t on t.id = q.talk_id

      where q.location_id='${locationId}' ${whereState} ${whereTalkId}

      order by talkId, totalQuestionVotes DESC;`;
```

DEVELOPMENT EXPERIENCE

1. Drizzle Server -
    1. Define Schema
    2. Migrate / Seed
2. Power Synch - Define Synch Rules
3. Drizzle Client -
    1. Define Schema
    2. Create Hook
        1. Query
            1. Aggregate
        2. Mutation
    3. Use Hook in Component

```
export const useInsertQuestion = ({ talkId }:
InsertQuestionsProps) => {
  const [question, updateQuestion] = useState("");
  const system = useSystem();
  const { locationId } = useFeatures();
  const insertQuestion = async () => {
    try {
      return system.db
        .insert(questions)
        .values({
          userId: system.userId!,
          question: question,
          talkId: talkId,
          locationId,
        })
        .returning();
    } catch (e) {
      console.log("######## INSERT QUESTION", e);
    }
  };
  return { insertQuestion, updateQuestion, question };
};
```

DEVELOPMENT EXPERIENCE

1. Drizzle Server -
   1. Define Schema
   2. Migrate / Seed
2. Power Synch - Define Synch Rules
3. Drizzle Client -
   1. Define Schema
   2. Create Hook
      1. Query
         1. Aggregate
      2. Mutation
   3. Use Hook in Component

```
export const QuestionsNewScreen:
React.FC<QuestionsNewScreenProps> = ({
  talkId,
  goBack,
}) => {

  const { insertQuestion } = useInsertQuestion({
    talkId,
  });

  const talk = useTalk(talkId);
```

FINAL THOUGHTS

PROS

1. Amazing DX!
2. Reduces cognitive API load!
3. The Domain is King!
4. Everything is Reactive!
5. You can be a Pioneer!
6. Just try it!

CONS

1. Pioneer Tax …
2. *Not a good match for all apps*
3. Can be hard to find a tech stack that suites your needs. (changing fast)

We believe that local-first is poised to become the default architecture for the majority of apps

Local-first apps feel instant to use because of the near-zero latency of working with a local in-app database

are functional even if the user's network connection is unreliable or unavailable

provide built-in multi-user real-time collaboration

RIFFLE PRINCIPLES

1. *DECLARATIVE QUERIES CLARIFY APPLICATION STRUCTURE*
2. *MANAGING ALL STATE IN ONE SYSTEM PROVIDES GREATER FLEXIBILITY*
3. *FAST REACTIVE QUERIES PROVIDE A CLEAN MENTAL MODEL*