

# Maximizing Cassandra's Potential: Tips on Schema, Queries, Parallel Access, and Reactive Programming

Hartmut Armbruster • [Thriving.dev](https://thriving.dev)

# Hartmut Armbruster

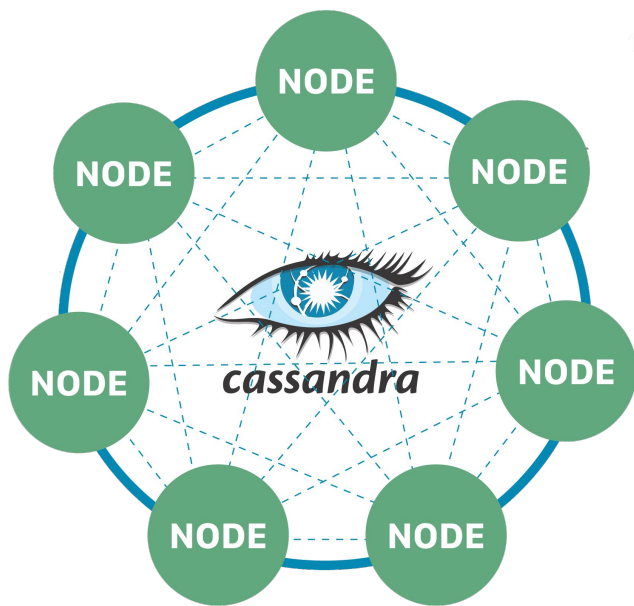
Software Architect, Developer,  
Independent Consultant

# Architecture • Data • Cloud-native •  
Distributed Systems • High-load/Scalability •  
Stream Processing • Backend • Web Front-End •  
Reactive Programming • Kotlin/Java • TS/JS •  
Vue.js • Nuxt.js • Kubernetes • GitOps

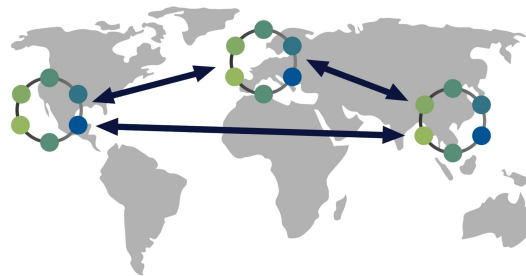


# Apache Cassandra in 1min

“When uptime and scale are non-negotiable, Cassandra delivers.”



- ❖ A highly scalable, distributed NoSQL database designed for massive data workloads
- ❖ Known for its **fault tolerance**, **high availability**, and **linear scalability**



# Agenda

- Use Case: 'Social Platform Feed'
- Access Patterns
  - Queries & Schema
  - Process Flow
- Iterative Refinement Process
- Non-blocking, Reactive Programming
- Code Example & Results
- Conclusions

The image shows a hand-drawn style screenshot of a social media feed. At the top, it says "thriving-dev / architecture" and "latest ↓". The first post is by a user with a profile icon, titled "RESTful API", with 42 likes. The second post is by "chloe" 8m ago, with 2.1k likes. The third post is by "Rodney" 43m ago, with 11k likes. Overlaid on the first post is a cyan box with "RESTful API", a yellow box with "<10ms!", and a pink box with "response time!!".

thriving-dev / architecture latest ↓

RESTful API 42

2.5k 57

<10ms!  
response time!!

chloe 8m ago 2.1k

137 3

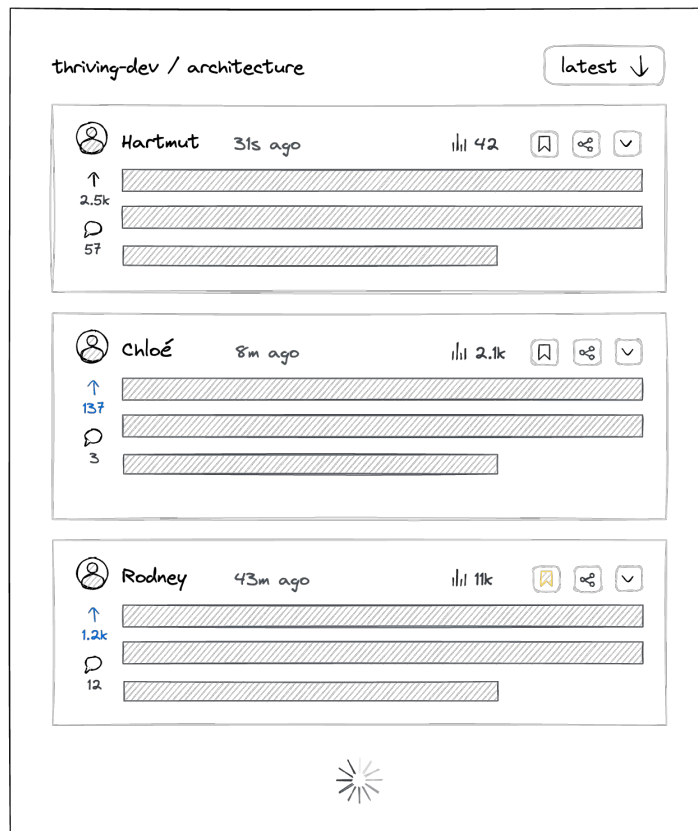
Rodney 43m ago 11k

# Functional Requirements

## RESTful API

## Social Platform Feed

- 'feeds' have 'posts'
- posts can be
  - listed in a given order
  - liked, bookmarked, commented
- "endless scroll"

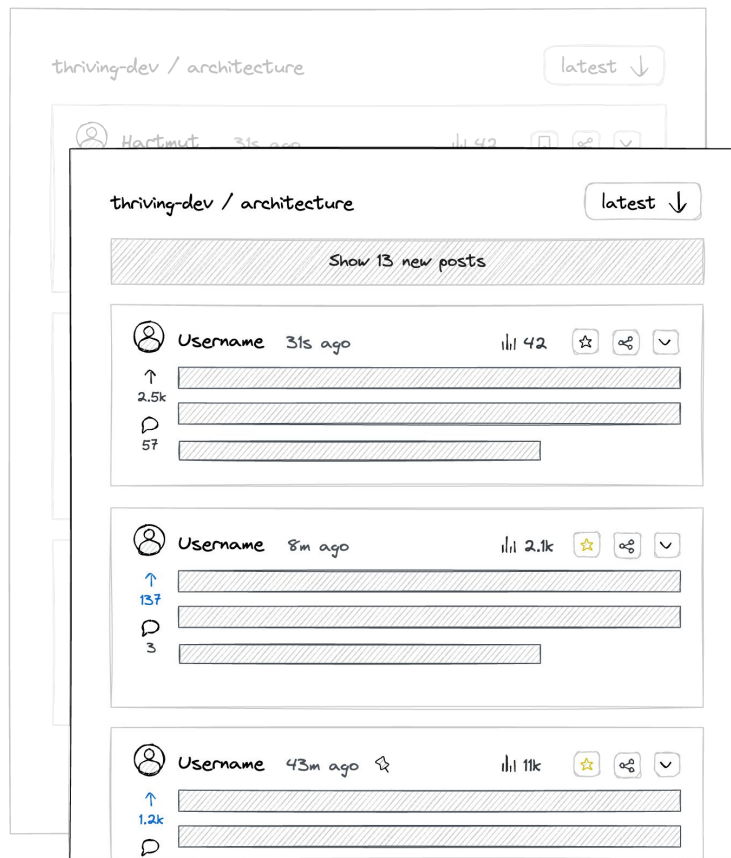


# Functional Requirements

## RESTful API

## Social Platform Feed

- 'feeds' have 'posts'
- posts can be
  - listed in a given order
  - liked, bookmarked, commented
- "endless scroll"
- "search-after" pagination
- "Show *n* new posts" indicator



# Social Platform Feed

RESTful API

Specification

GET /feeds/{feedId}/posts

?pageSize=20

&ltPostId=01HZMN4H77QBW1W8PS6W04ZH9V

Provides

- **all data** required to render the 'Social Platform Feed'
- for an authenticated **user** or guest
- with a **single request/response**

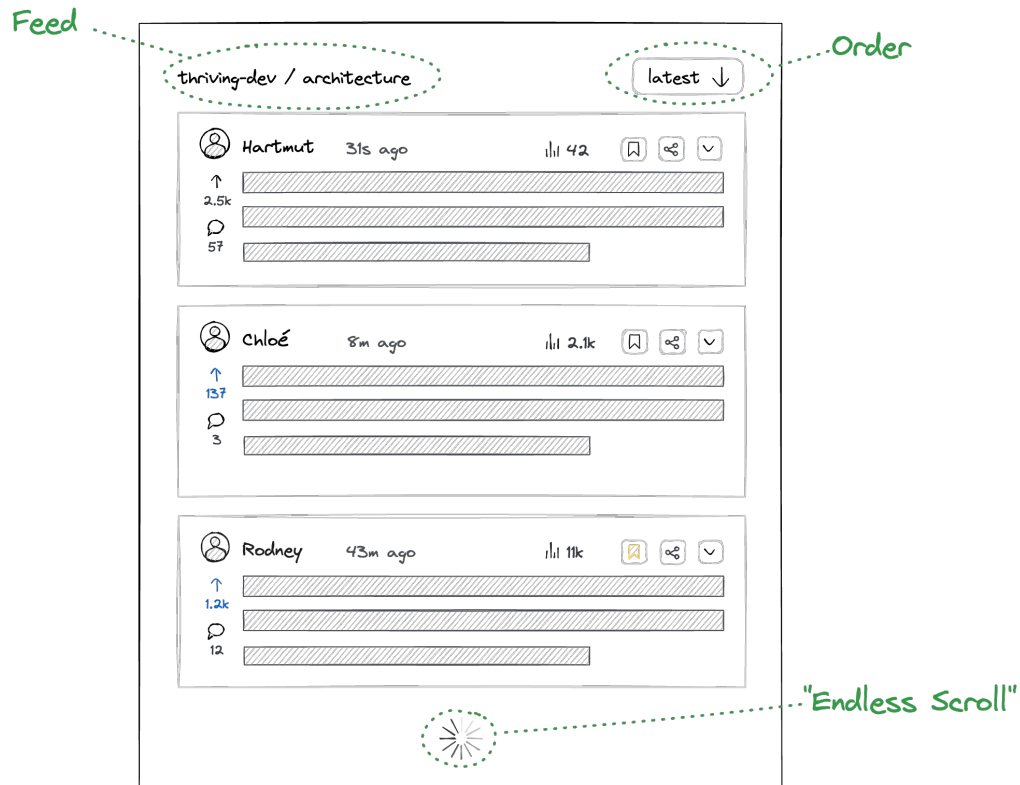
```
1 {
2   "meta": {
3     "feedId": "01HPWG2T821KXLA7TECS8W74W6",
4     "ltPostId": "01HZMN4H77QBW1W8PS6W04ZH9V",
5     "sessionId": "01HPWG2T869DBCS8W74W6XAVTE",
6     "pageSize": 20
7   },
8   "posts": [
9     {
10      "feedId": "01HPWG2T821KXLA7TECS8W74W6",
11      "postId": "01HQRE72RQ8RPHYDEAPGD9PZ8E",
12      "author": {
13        "userId": "01HQRE2YKE2628J739ZKDNMRSY",
14        "username": "tom",
15        "name": "Tom"
16      },
17      "text": "WeAreDevelopers World Congress is the best place to get a
18      complete overview of recent insights and future trends in modern
19      software development. Take the opportunity to grow your expertise and
20      elevate your capabilities in crafting remarkable software and
21      products.",
22      "impressions": 8965,
23      "bookmarked": 3,
24      "likes": 42,
25      "replies": 7,
26      "sessionUserRel": [
27        "BOOKMARK",
28        "LIKE"
29      ],
30      "createdAt": "2024-06-16T09:29:02.848Z"
31    },
32    {
33      "feedId": "01HPWG2T821KXLA7TECS8W74W6",
34      "postId": "01H0RE7282ZV9NVJ0H1SV2BY90"
```

# Data

## Entities & Relationships

### Feed

feed_id	text
name	text
created_at	timestamp





# Data

## Entities & Relationships

Post	
post_id	text
feed_id	text
user_id	text
text	text
created_at	timestamp
impressions	number
likes	number
comments	number

thriving-dev / architecture latest ↓

**Post 1:** Hartmut 31s ago 42 likes 2.5k impressions 57 comments

**Post 2:** Chloé 8m ago 2.1k likes 137 comments 3 comments

**Post 3:** Rodney 43m ago 11k likes 12 comments

Labels: Likes, Comments, Post, Impressions

# Data

## Entities & Relationships

### User Likes Post

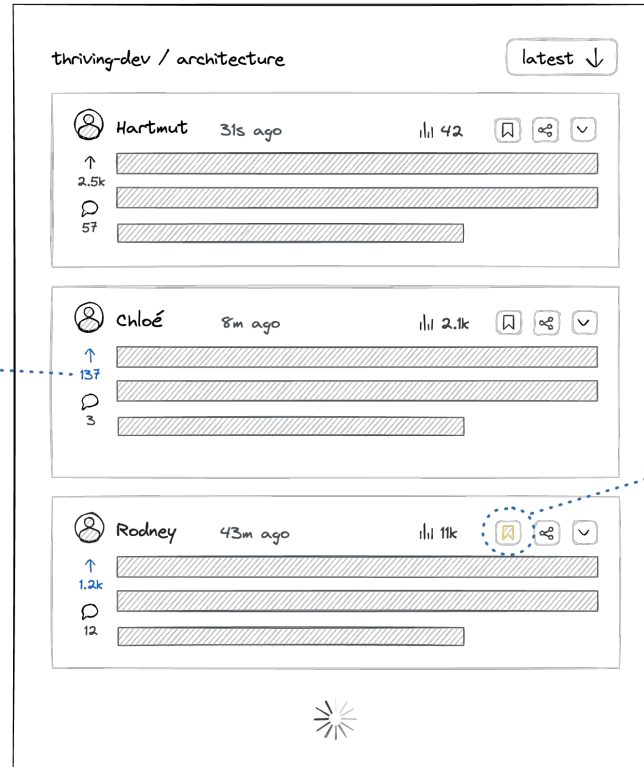
user_id	text
post_id	text
created_at	timestamp

### User Bookmarked Post

user_id	text
post_id	text
created_at	timestamp

Liked  
by me

Bookmarked  
by me



# Data

## Entities & Relationships

User	
user_id	text
username	text
name	text
avatar_url	text
created_at	timestamp
updated_at	timestamp

thriving-dev / architecture latest ↓

Hartmut 31s ago 111 42 🔖 🔗 ⌵

↑ 2.5k

🗨️ 57

Chloé 8m ago 111 2.1k 🔖 🔗 ⌵

↑ 137

🗨️ 3

Rodney 43m ago 111 11k 🔖 🔗 ⌵

↑ 1.2k

🗨️ 12

⚙️

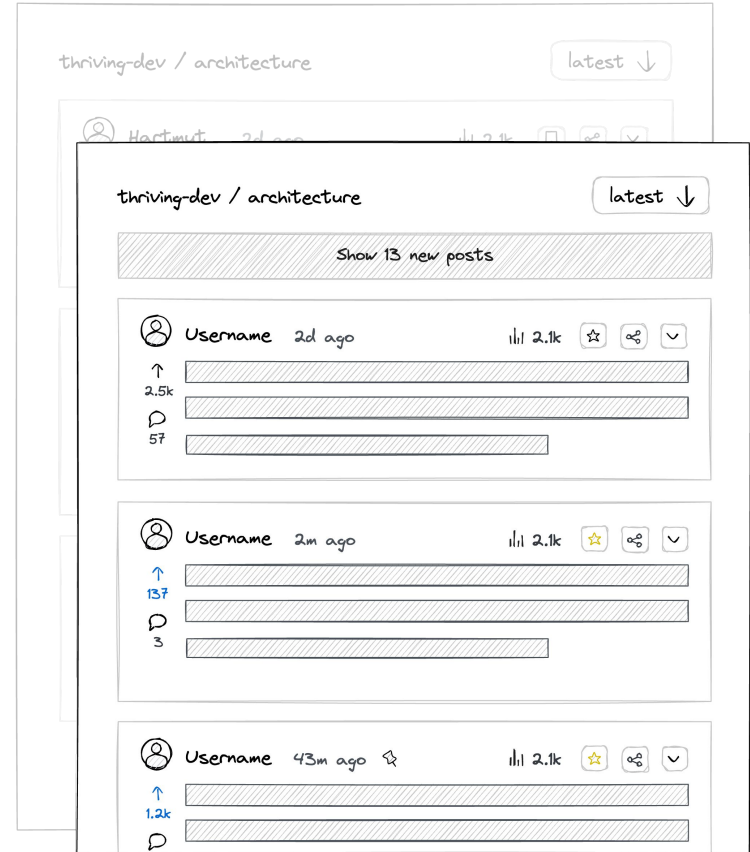
User  
(Author)

# Access Patterns

"Query-First Approach"

# Access Patterns

1. [feed] get posts
  - paginated, ordered (created DESC)
  - allow to pin query time for pagination
2. [post] get stats
  - (counters): no. of impressions, likes, comments
3. [post<>user] get session-user relations
  - likes the post
  - has bookmarked the post
4. [post] get author
  - by user\_id
5. [feed] get 'new posts' count



# Data Schema (1)

```
-- Tables related to posts
CREATE TABLE post (
  feed_id  text,
  post_id  text,
  user_id  text,
  text     text,
  created_at timestamp,
  PRIMARY KEY (feed_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);
```

```
-- 1.a. [feed] get posts, paginated (clustering key default order:
post_id DESC)
SELECT *
FROM post
WHERE feed_id='01HPWG2T821KXLA7TECS8W74W6'
LIMIT 20;

-- 1.b. [feed] get posts, paginated (clustering key default order:
post_id DESC)
--      - all posts 'before' (timestamp) a certain post_id
SELECT *
FROM post
WHERE feed_id='01HPWG2T821KXLA7TECS8W74W6'
  AND post_id<'01HQ0FDNEPZR1ES39JX8SHE54Q'
LIMIT 20;
```

# ULID



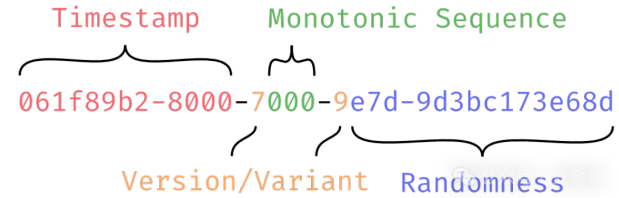
⇒ 0GWWXY2G84DFMRVWQNJ1SRYCMC

Universally Unique

**Lexicographically Sortable** Identifier

- **Canonically encoded** (26 chars)
- Case insensitive, URL safe
- 1.21e+24 unique ULIDs per millisecond

# UUID v7



UUIDv7

(RFC 9562)


- 128-bit compatibility with prev. UUID Standards
- Part of ongoing standardization efforts

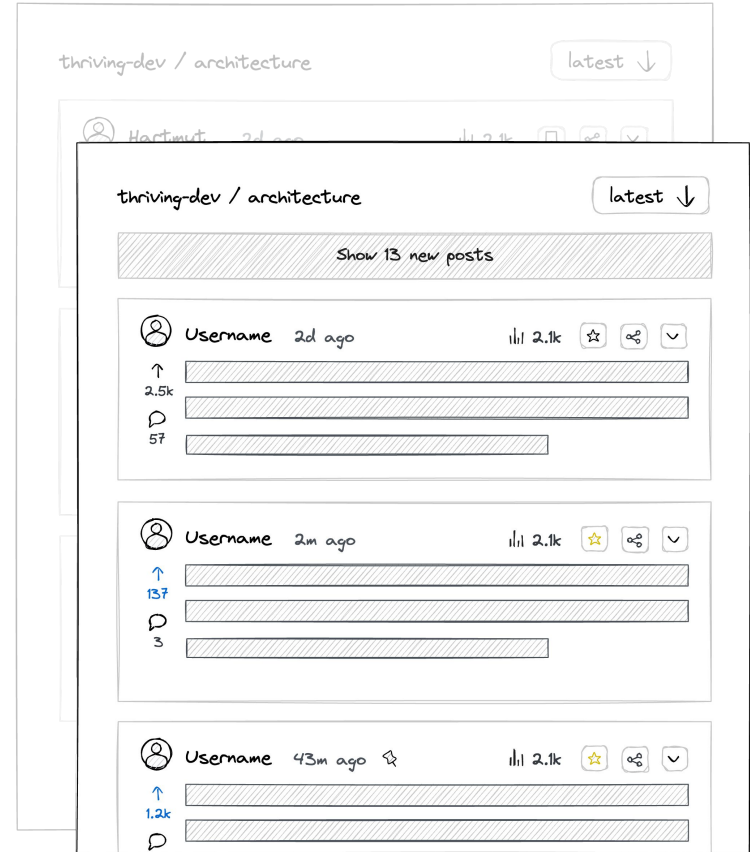
[0]: <https://github.com/ulid/spec>

[1]: <https://datatracker.ietf.org/doc/rfc9562/>

[2]: <https://itnext.io/why-uuid7-is-better-than-uuid4-as-clustered-index-edb02bf70056>

# Access Patterns

1. [feed] get posts 
  - paginated, ordered (created DESC)
  - allow to pin query time for pagination
2. [post] get stats
  - (counters): no. of impressions, likes, comments
3. [post<>user] get session-user relations
  - likes the post
  - has bookmarked the post
4. [post] get author
  - by user\_id
5. [feed] get 'new posts' count







## Data Schema (2)

```
-- Tables related to posts
CREATE TABLE post (
  feed_id  text,
  post_id  text,
  user_id  text,
  text     text,
  created_at timestamp,
  PRIMARY KEY (feed_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);

CREATE TABLE post_stats (
  post_id  text,
  impressions counter,
  bookmarked counter,
  likes    counter,
  replies  counter,
  PRIMARY KEY (post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);
```

```
-- 2.a. [post] get stats
SELECT *
FROM post_stats
WHERE post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';
```

# Access Patterns

1. [feed] get posts 
  - paginated, ordered (created DESC)
  - allow to pin query time for pagination
2. [post] get stats 
  - (counters): no. of impressions, likes, comments
3. [post<>user] get session-user relations
  - likes the post
  - has bookmarked the post
4. [post] get author
  - by user\_id
5. [feed] get 'new posts' count



# Data Schema (3)




```
-- Tables related to likes between
--      users and posts
CREATE TABLE user_likes_post (
  user_id    text,
  post_id    text,
  created_at timestamp,
  PRIMARY KEY (user_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);

-- Tables related to bookmarks between
--      users and posts
CREATE TABLE user_bookmarked_post (
  user_id    text,
  post_id    text,
  created_at timestamp,
  PRIMARY KEY (user_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);
```

```
-- 3.a. [post<>user] get session-user likes for post
SELECT *
FROM user_likes_post
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
      AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';

-- 3.b. [post<>user] get session-user has bookmarked post
SELECT *
FROM user_bookmarked_post
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
      AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';
```

# Access Patterns

1. [feed] get posts 
  - paginated, ordered (created DESC)
  - allow to pin query time for pagination
2. [post] get stats 
  - (counters): no. of impressions, likes, comments
3. [post<>user] get session-user relations 
  - likes the post
  - has bookmarked the post
4. [post] get author
  - by user\_id
5. [feed] get 'new posts' count



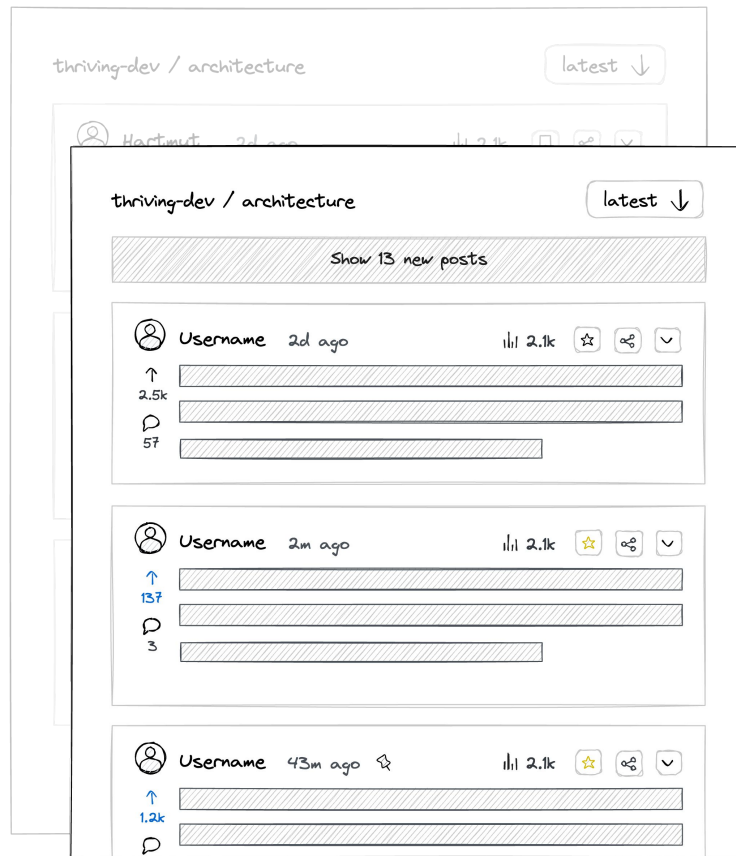
# Data Schema (4)

```
-- Tables related to users
CREATE TABLE user (
  user_id      text,
  username     text,
  password_hash text,
  name         text,
  avatar_url   text,
  created_at   timestamp,
  updated_at   timestamp,
  PRIMARY KEY (user_id)
);
```

```
-- 4.a. [post] get author
SELECT *
FROM user
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ';
```

# Access Patterns

1. [feed] get posts 
  - paginated, ordered (created DESC)
  - allow to pin query time for pagination
2. [post] get stats 
  - (counters): no. of impressions, likes, comments
3. [post<>user] get session-user relations 
  - likes the post
  - has bookmarked the post
4. [post] get author 
  - by user\_id
5. [feed] get 'new posts' count



# Data Schema (5)

```

-- Tables related to posts
CREATE TABLE post (
  feed_id  text,
  post_id  text,
  user_id  text,
  text     text,
  created_at timestamp,
  PRIMARY KEY (feed_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);

```

```

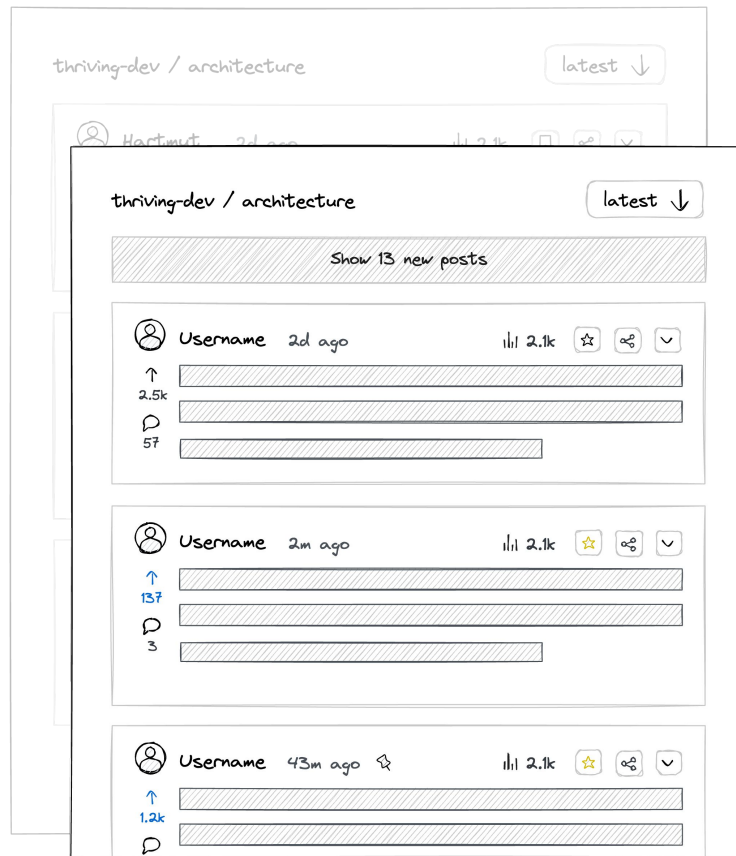
-- 5.a. [feed] get 'new posts' count
-- ⚠ DON'T DO THIS AT HOME: using COUNT()
-- (note: see better query 5.b. below)
SELECT COUNT(*) 01HPWG2T821KXLA7TECS8W74W6'
FROM post_id='01HQ0FDNEPZR1ES39JX8SHE54Q';
WHERE feed_id='01HPWG2T821KXLA7TECS8W74W6'
AND post_id='01HQ0FDNEPZR1ES39JX8SHE54Q';

-- 5.b. [feed] get 'new posts' count
-- 🙄 avoids COUNT() -> query IDs with upper LIMIT,
-- manually count rows returned in client
SELECT post_id
FROM post
WHERE feed_id='01HPWG2T821KXLA7TECS8W74W6'
AND post_id>='01HQ0FDNEPZR1ES39JX8SHE54Q'
LIMIT 1000; -- display like: '1'; '721'; '>1000'

```

# Access Patterns

1. [feed] get posts 
  - paginated, ordered (created DESC)
  - allow to pin query time for pagination
2. [post] get stats 
  - (counters): no. of impressions, likes, comments
3. [post<>user] get session-user relations 
  - likes the post
  - has bookmarked the post
4. [post] get author 
  - by user\_id
5. [feed] get 'new posts' count





# API Handler Process Flow (v1)

The most basic approach...

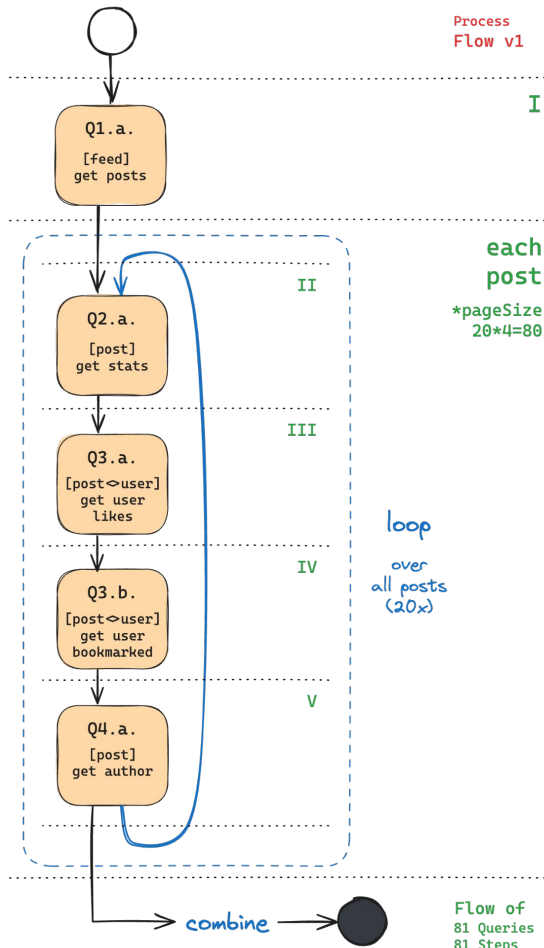
...fetch a 'page' of 20 posts  
...iterate and enrich

- > **81 queries** to the DB
- > **81 subsequent steps** (IO)



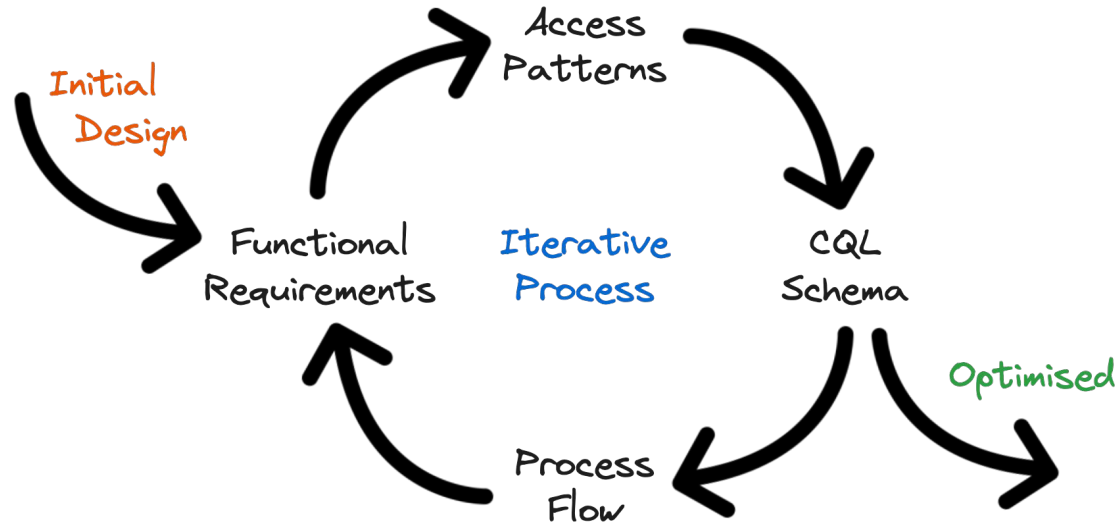
🤔 surely we can do better...??

GET /feeds/{feedId}/posts



# Data Modelling

## Iterative Process of Schema Refinement



## Data Schema (6)

```

CREATE TABLE user_likes_post (...)
CREATE TABLE user_bookmarked_post (...)

-- Tables related to relations between
--   users and posts
CREATE TABLE user_x_post_rel (
  user_id text,
  rel_type text,
  post_id text,
  created_at timestamp,
  PRIMARY KEY (user_id, rel_type, post_id)
) WITH CLUSTERING ORDER
  BY (rel_type ASC, post_id DESC);

```

```

-- 3.c. [post<>user] get specific session-user relations for post
SELECT *
FROM user_x_post_rel
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
  AND rel_type='LIKE'
  AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';

-- 3.d. [post<>user] get all session user relations for post
SELECT *
FROM user_x_post_rel
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
  AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';

-- 3.d. [post<>user] get all session-user relations for post
SELECT *
FROM user_x_post_rel
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
  AND rel_type IN ('LIKE', 'BOOKMARK')
  AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';

```

GET /feeds/{feedId}/posts

# API Handler Process Flow (v2)

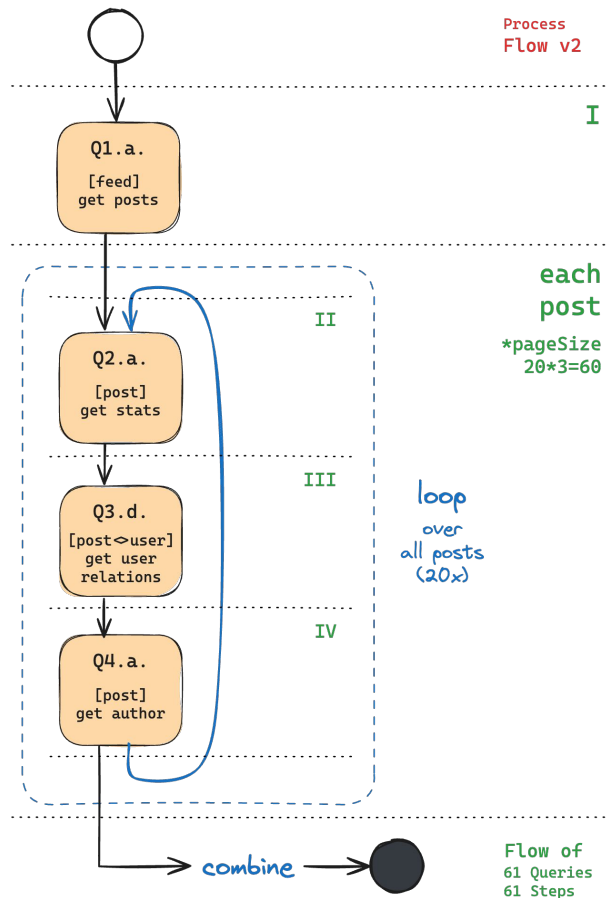
Still a basic approach...

...fetch a 'page' of 20 posts  
...iterate and enrich x3

- > **61 queries** to the DB
- > **61 subsequent** steps (IO)



...still have to do  
better!!



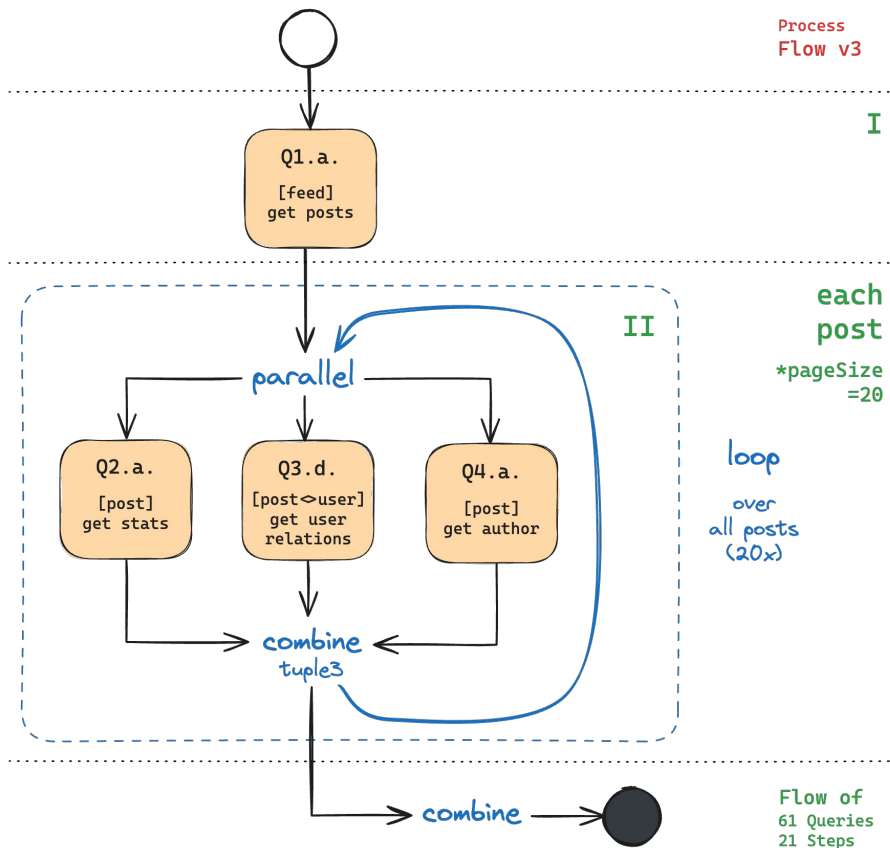
# API Handler Process Flow (v3)

## Parallel Processing

...fetch a 'page' of 20 posts  
...iterate and enrich **in parallel**

- > **61 queries** to the DB
- > **21 subsequent** steps (IO)

GET /feeds/{feedId}/posts



# Data Schema (7)

```

CREATE TABLE post_stats (
  post_id text,
  impressions counter,
  bookmarked counter,
  likes counter,
  replies counter,
  PRIMARY KEY (post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);

```

```

CREATE TABLE post_stats (
  feed_id text,
  post_id text,
  impressions counter,
  bookmarked counter,
  likes counter,
  replies counter,
  PRIMARY KEY (feed_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);

```

```

-- 2.a. [post] get stats
SELECT *
FROM post_stats
WHERE post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';

-- 2.b. [post] get stats for a list of posts
SELECT *
FROM post_stats
WHERE feed_id='01HPWG2T821KXLA7TECS8W74W6'
  AND post_id IN ('01HQ0FDQWBDSK3BZG99NR7JSE6',
                 '01HQ0FDPY867VHKMQGCMVFJFQW',
                 '01HQ0FDP3HVV04TFESMY5DJJA5');

```

# Data Schema (8)

```

-- Tables related to relations between
-- users and posts
CREATE TABLE user_x_post_rel (
  user_id    text,
  rel_type   text,
  post_id    text,
  created_at timestamp,
  PRIMARY KEY (user_id, rel_type, post_id)
) WITH CLUSTERING ORDER
  BY (rel_type ASC, post_id DESC);

```

```

-- 3.d. [post<>user] get any session user relations for post
SELECT *
FROM user_x_post_rel
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
AND rel_type IN ('LIKE', 'BOOKMARK')
AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6');

-- 3.e. [post<>user] get session-user relations
-- fetch for all posts in current 'page' at once
SELECT *
FROM user_x_post_rel
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
  AND rel_type IN ('LIKE', 'BOOKMARK')
  AND post_id IN ('01HQ0FDQWBDSK3BZG99NR7JSE6',
                  '01HQ0FDPY867VHKMQGCMVFJFQW',
                  '01HQ0FDP3HVV04TFESMY5DJ1A5');
```

# API Handler Process Flow (v4)

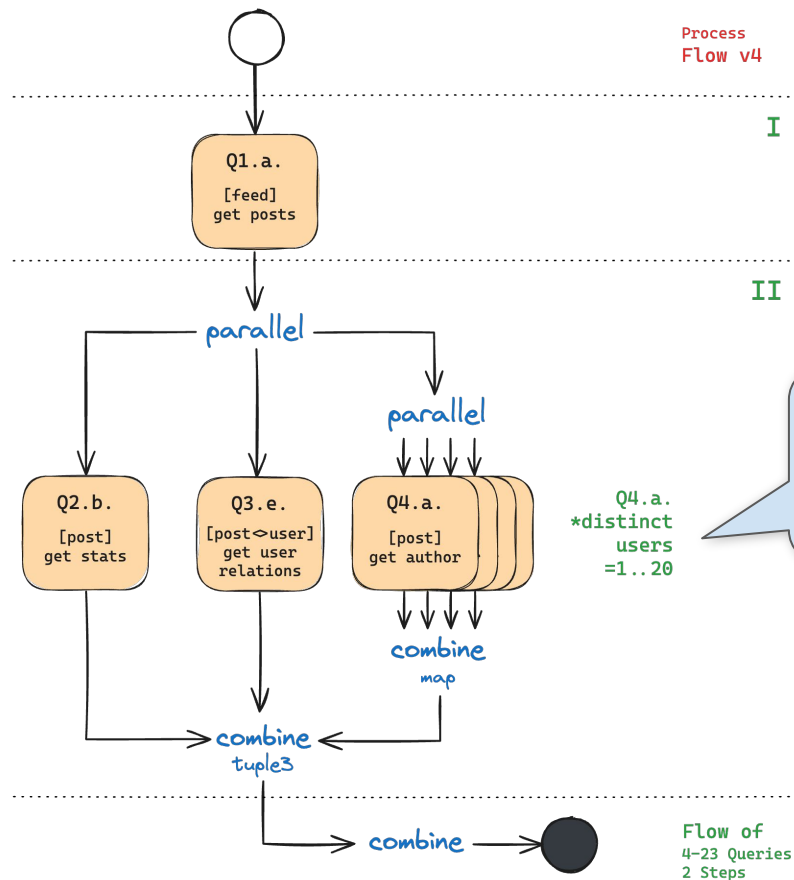
## Optimal Parallelisation

...fetch a 'page' of 20 posts  
 ...iterate **bulk query** enrichment  
 ...enrich in **parallel**  
 ...lookup **distinct users**

-> **4...23 queries** to the DB  
 -> **2 subsequent** steps (IO)



GET /feeds/{feedId}/posts







**HAVING COMPLETED  
THE REFINEMENT  
PROCESS**

# Application Design

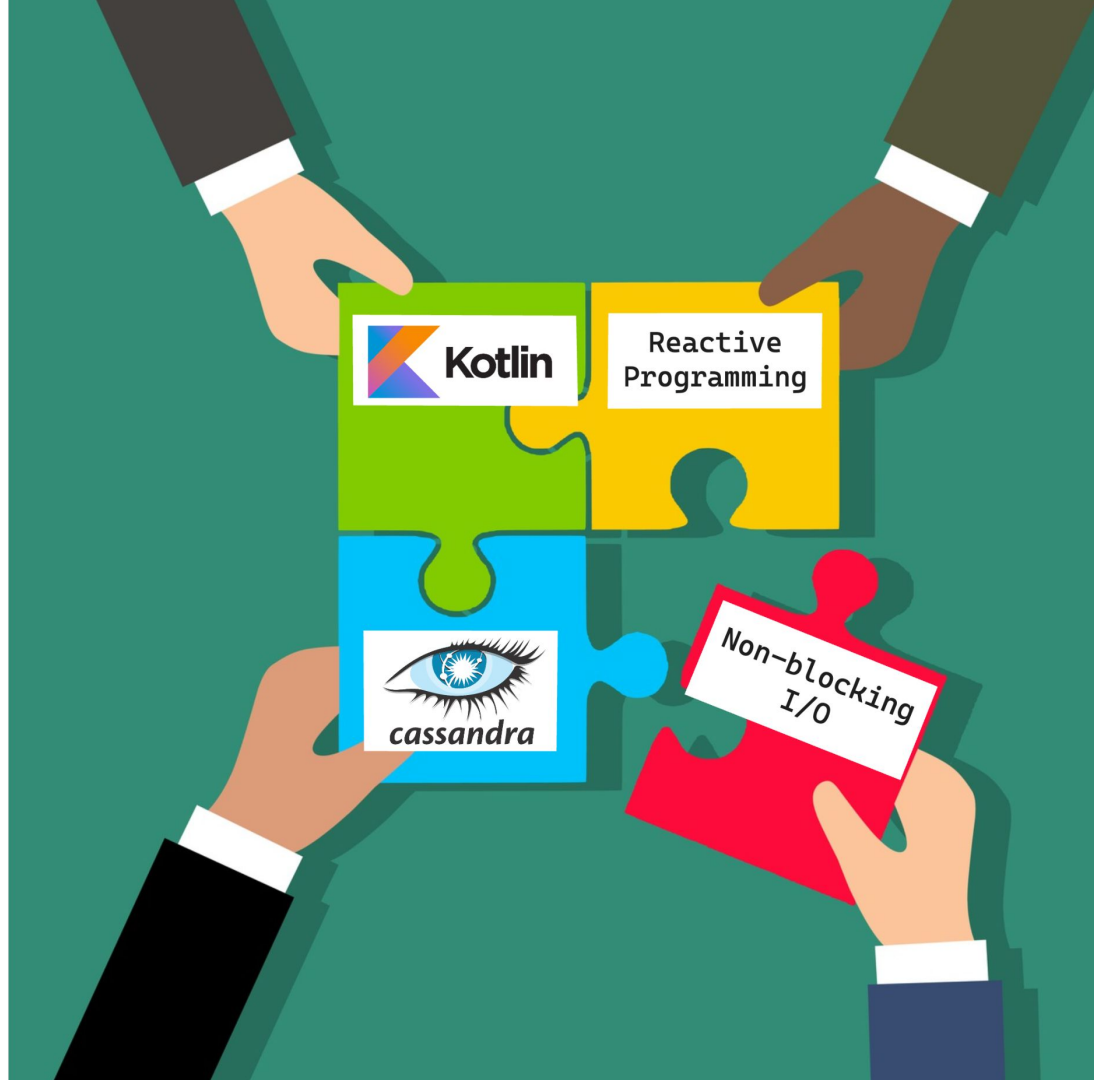
Backend

# Application Tech Stack

🚀 Non-blocking IO

✨ Reactive Programming

🦄 Kotlin



# Reactive Programming

## Key Benefits

- Asynchronous & Non-Blocking Operations
- Declarative Code
- Resilience and Fault Tolerance
- Responsive Systems



Project  
Reactor



RxJava

**MUTINY!**

Simply put, it's  
all about **Data Streams**



- Filter
- Map / Transform
- Reduce
- Collect

POC

# Implementation



QUARKUS

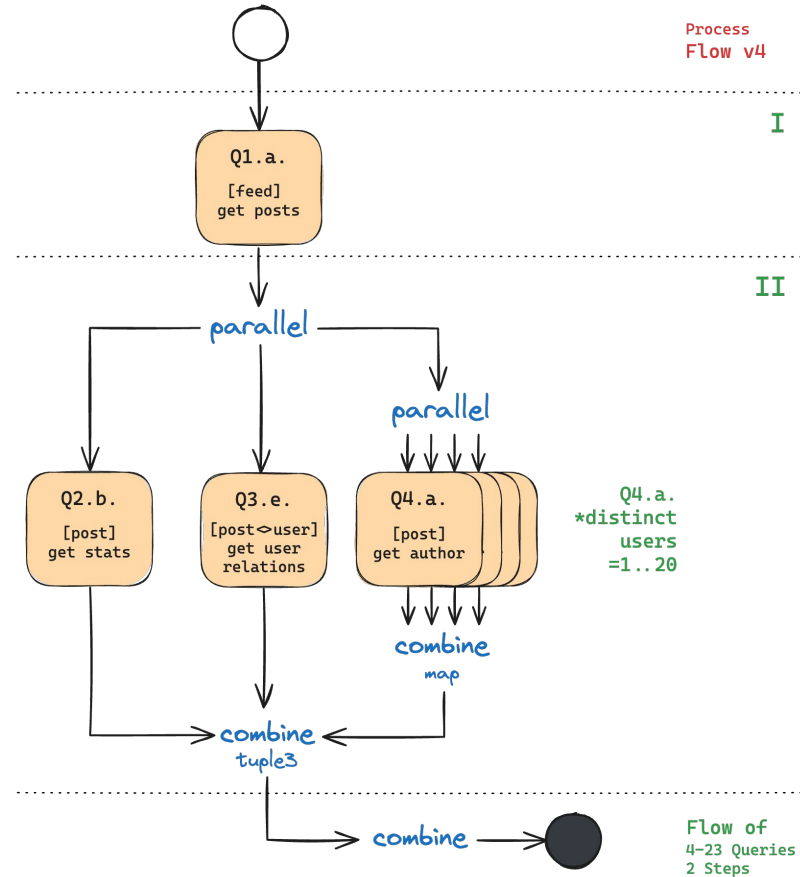
***MUTINY!***



**Kotlin**

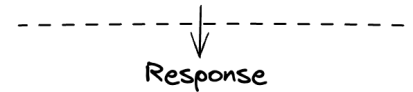
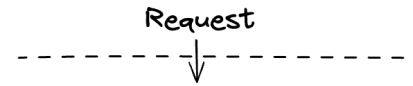
# API Handler Process Flow (v4)

GET /feeds/{feedId}/posts



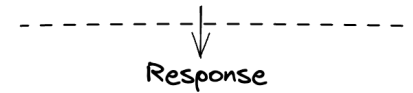
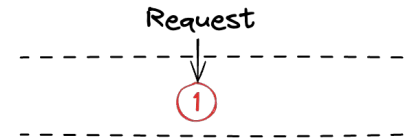
# Code Example

```
@GET
@Path("/feeds/{feedId}/posts")
fun posts(
    feedId: String,
    @QueryParam("ltPostId") ltPostId: String,
    @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
```



# Code Example

```
@GET
@Path("/feeds/{feedId}/posts")
fun posts(
    feedId: String,
    @QueryParam("ltPostId") ltPostId: String,
    @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
    ① getPostsByFeedIdLtPostId(feedId, ltPostId, pageSize)
```



① (1.b.) `SELECT post[]`

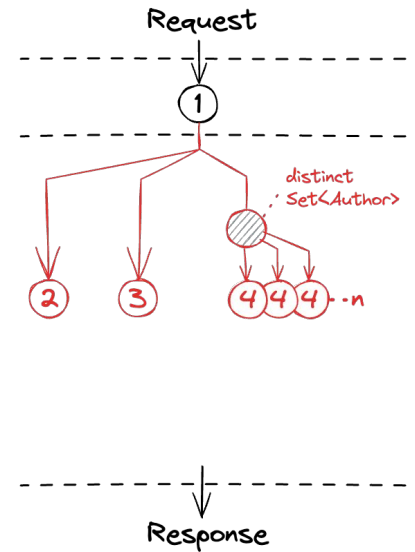


# Code Example

```

@GET
@Path("/feeds/{feedId}/posts")
fun posts(
    feedId: String,
    @QueryParam("ltPostId") ltPostId: String,
    @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
    ① getPostsByFeedIdLtPostId(feedId, ltPostId, pageSize)
        .transformToUni { posts: List<Post> ->
            Uni.combine().all().unis(
                ② getPostStatsListByFeed(feedId, posts),
                ③ getUserRelationsByPostIdsAsMap(STATIC_SESSION_USER_ID, posts),
                ④ getUsersByIdsAsMap(posts),
            )
        }
}

```



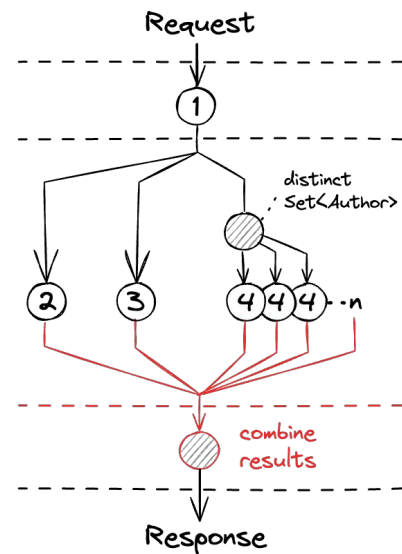
- ① (1.b.) SELECT post[]
- ② (2.b.) SELECT post\_stats[]
- ③ (3.e.) SELECT user\_x\_post\_rel[]
- ④ (4.a.) SELECT user

# Code Example

```

@GET
@Path("/feeds/{feedId}/posts")
fun posts(
    feedId: String,
    @QueryParam("ltPostId") ltPostId: String,
    @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
    ① getPostsByFeedIdLtPostId(feedId, ltPostId, pageSize)
        .transformToUni { posts: List<Post> ->
            Uni.combine().all().unis(
                ② getPostStatsListByFeed(feedId, posts),
                ③ getUserRelationsByPostIdsAsMap(STATIC_SESSION_USER_ID, posts),
                ④ getUsersByIdsAsMap(posts),
            ).asTuple().map { tuple3 -> ← combine results
                ok(toPostsResponseDto(feedId, ltPostId, pageSize, posts, tuple3))
            }
        }
}

```



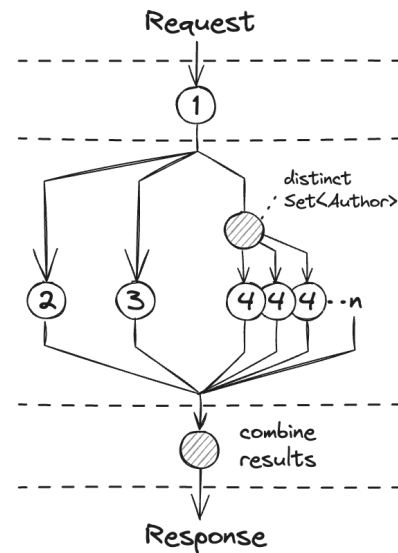
- ① (1.b.) SELECT post[]
- ② (2.b.) SELECT post\_stats[]
- ③ (3.e.) SELECT user\_x\_post\_rel[]
- ④ (4.a.) SELECT user

# Code Example

```

@GET
@Path("/feeds/{feedId}/posts")
fun posts(
    feedId: String,
    @QueryParam("ltPostId") ltPostId: String,
    @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
    ① getPostsByFeedIdLtPostId(feedId, ltPostId, pageSize)
        .transformToUni { posts: List<Post> ->
            Uni.combine().all().unis(
                ② getPostStatsListByFeed(feedId, posts),
                ③ getUserRelationsByPostIdsAsMap(STATIC_SESSION_USER_ID, posts),
                ④ getUsersByIdsAsMap(posts),
            ).asTuple().map { tuple3 -> ← combine results
                ok(toPostsResponseDto(feedId, ltPostId, pageSize, posts, tuple3))
            }
        }
    }.onFailure(NotFoundException::class.java) ← handle 404
        .recoverWithItem(notFound())

```



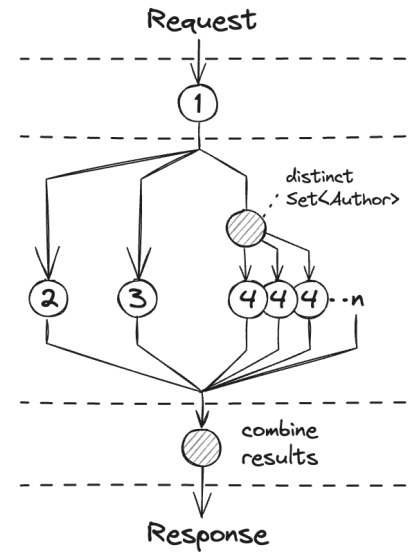
- ① (1.b.) SELECT post[]
- ② (2.b.) SELECT post\_stats[]
- ③ (3.e.) SELECT user\_x\_post\_rel[]
- ④ (4.a.) SELECT user

# Code Example

```

@GET
@Path("/feeds/{feedId}/posts")
fun posts(
    feedId: String,
    @QueryParam("ltPostId") ltPostId: String,
    @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
    ① getPostsByFeedIdLtPostId(feedId, ltPostId, pageSize)
        .transformToUni { posts: List<Post> ->
            Uni.combine().all().unis(
                ② getPostStatsListByFeed(feedId, posts),
                ③ getUserRelationsByPostIdsAsMap(STATIC_SESSION_USER_ID, posts),
                ④ getUsersByIdsAsMap(posts),
            ).asTuple().map { tuple3 -> ← combine results
                ok(toPostsResponseDto(feedId, ltPostId, pageSize, posts, tuple3))
            }
        }
    }.onFailure(NotFoundException::class.java) ← handle 404
        .recoverWithItem(notFound())

```



- ① (1.b.) SELECT post[]
- ② (2.b.) SELECT post\_stats[]
- ③ (3.e.) SELECT user\_x\_post\_rel[]
- ④ (4.a.) SELECT user

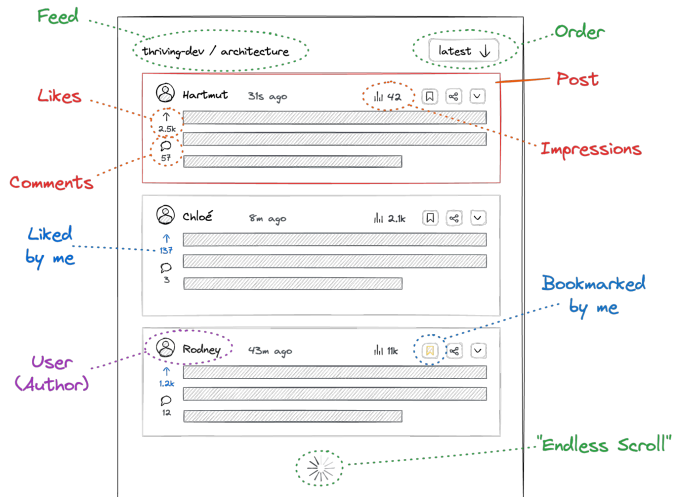
# Social Platform Feed

## RESTful API

GET /feeds/{feedId}/posts

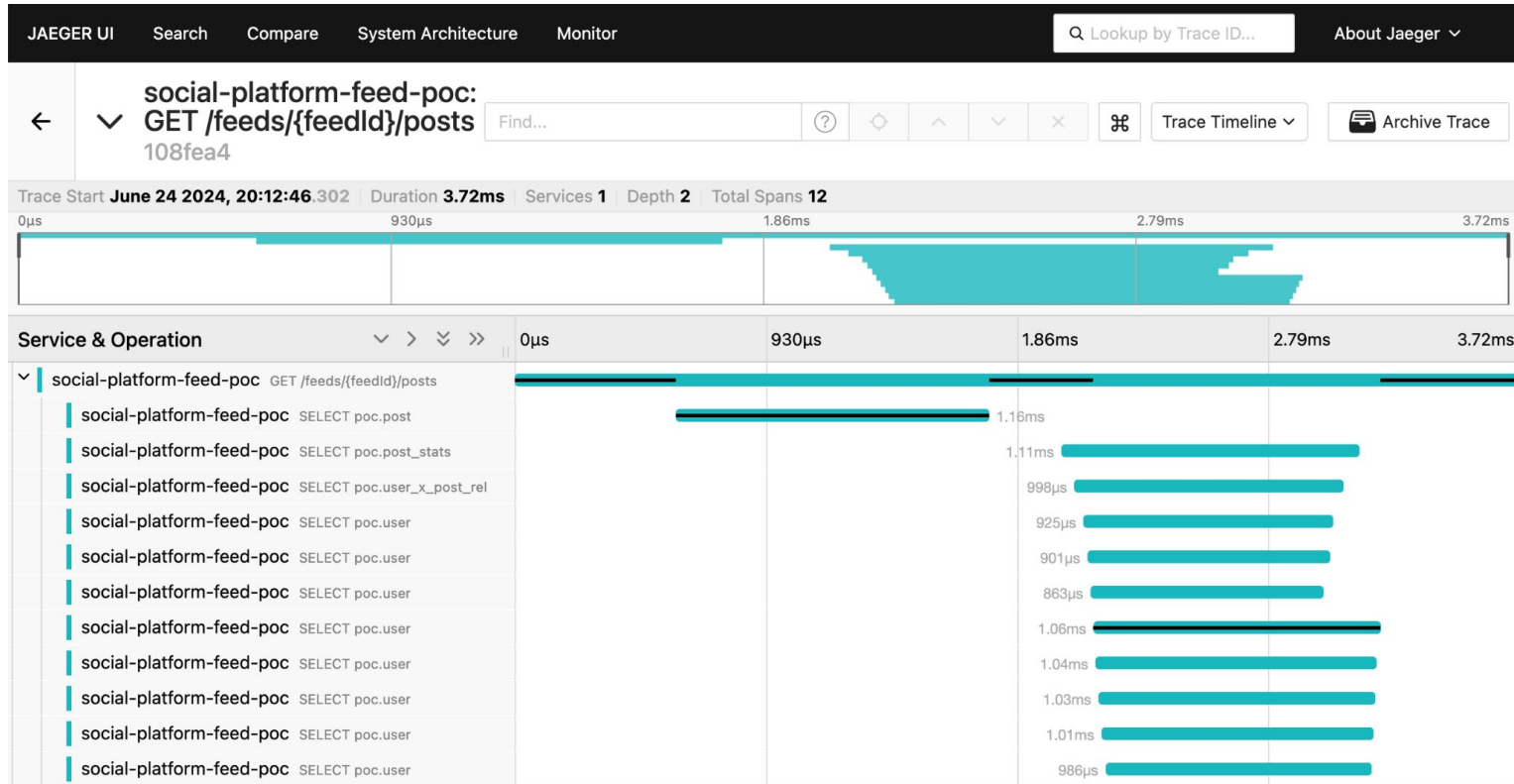
?pageSize=20

&ltPostId=01HZMN4H77QBW1W8PS6W04ZH9V

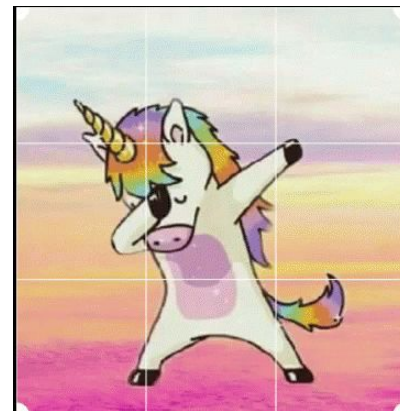
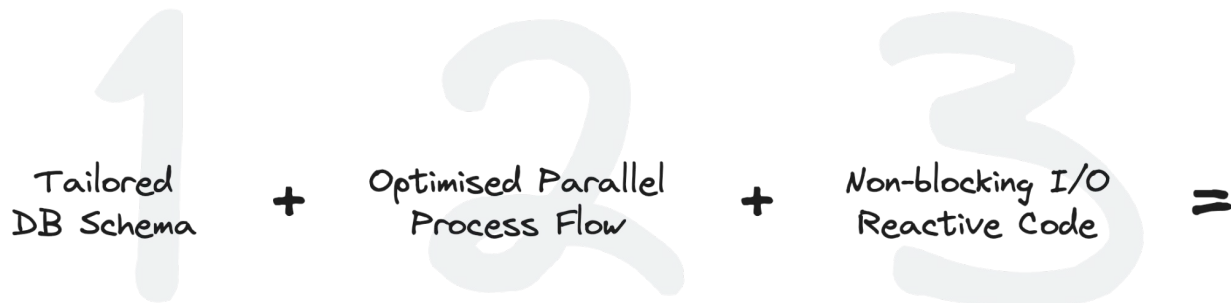


```
1 {
2   "meta": {
3     "feedId": "01HPWG2T821KXLA7TECS8W74W6",
4     "ltPostId": "01HZMN4H77QBW1W8PS6W04ZH9V",
5     "sessionId": "01HPWG2T869DBCS8W74W6XAVTE",
6     "pageSize": 20
7   },
8   "posts": [
9     {
10      "feedId": "01HPWG2T821KXLA7TECS8W74W6",
11      "postId": "01HQRE72RQ8RPHYDEAPGD9PZ8E",
12      "author": {
13        "userId": "01HQRE2YKE2628J739ZKDNMRSY",
14        "username": "tom",
15        "name": "Tom"
16      },
17      "text": "WeAreDevelopers World Congress is the best place to get a
18      complete overview of recent insights and future trends in modern
19      software development. Take the opportunity to grow your expertise and
20      elevate your capabilities in crafting remarkable software and
21      products.",
22      "impressions": 8965,
23      "bookmarked": 3,
24      "likes": 42,
25      "replies": 7,
26      "sessionUserRel": [
27        "BOOKMARK",
28        "LIKE"
29      ],
30      "createdAt": "2024-06-16T09:29:02.848Z"
31    },
32    {
33      "feedId": "01HPWG2T821KXLA7TECS8W74W6",
34      "postId": "01H0RE7282ZV9NVJ0H1SV2BY90"
35    }
36  ]
37 }
```

# Sample Trace (OpenTelemetry)



# Conclusions



# Conclusions; Beware of ... 🙅

- Choosing Cassandra should be an **informed decision!**
  - Data must be denormalized & indexed on write
  - Extra complexity of working with eventual consistency
  - Watch out for hot partitions -> "*bucketing*"
  - Migrations are expensive
- Reactive Programming
  - Training and Learning Curve
  - Testing, Debugging and Troubleshooting
- Universal patterns & solutions



# Example: Bluesky

v1 Architecture -> Scaling the database layer. \*



*Postgres was great early on because we didn't quite know exactly what questions we'd be asking of the data. It let us toss data into the database and figure it out from there.*

*Now we understand the data and the types of queries we need to run, it frees us up to **index it in Scylla in exactly the manner we need** and provide APIs for the exact queries we'll be asking.*

[1]: <https://newsletter.pragmaticengineer.com/p/bluesky>

# Questions?

 @hartmut\_co\_uk

 @hartmut-co-uk

 @hartmut-co-uk

 @thriving\_dev

 <https://thriving.dev>

