From Friends-of-Friends to Scaling APIs

Running Neo4j on AWS with Fargate

Why this talk?

- Apps = connections (friends, groups, patients).
- Relational joins → complex & slow.
- Graph DBs (Neo4j) → natural for these queries.
- But: How do you scale the API behind them?

The Challenge: When Relationships Get Complex

Traditional Approach

```
SELECT u1.name as friend, u3.name as
friend_of_friend FROM users u1 JOIN
friendships f1 ON u1.id = f1.user_id JOIN
users u2 ON f1.friend_id = u2.id JOIN
friendships f2 ON u2.id = f2.user_id JOIN
users u3 ON f2.friend_id = u3.id WHERE u1.id =
? AND u3.id != ?
```

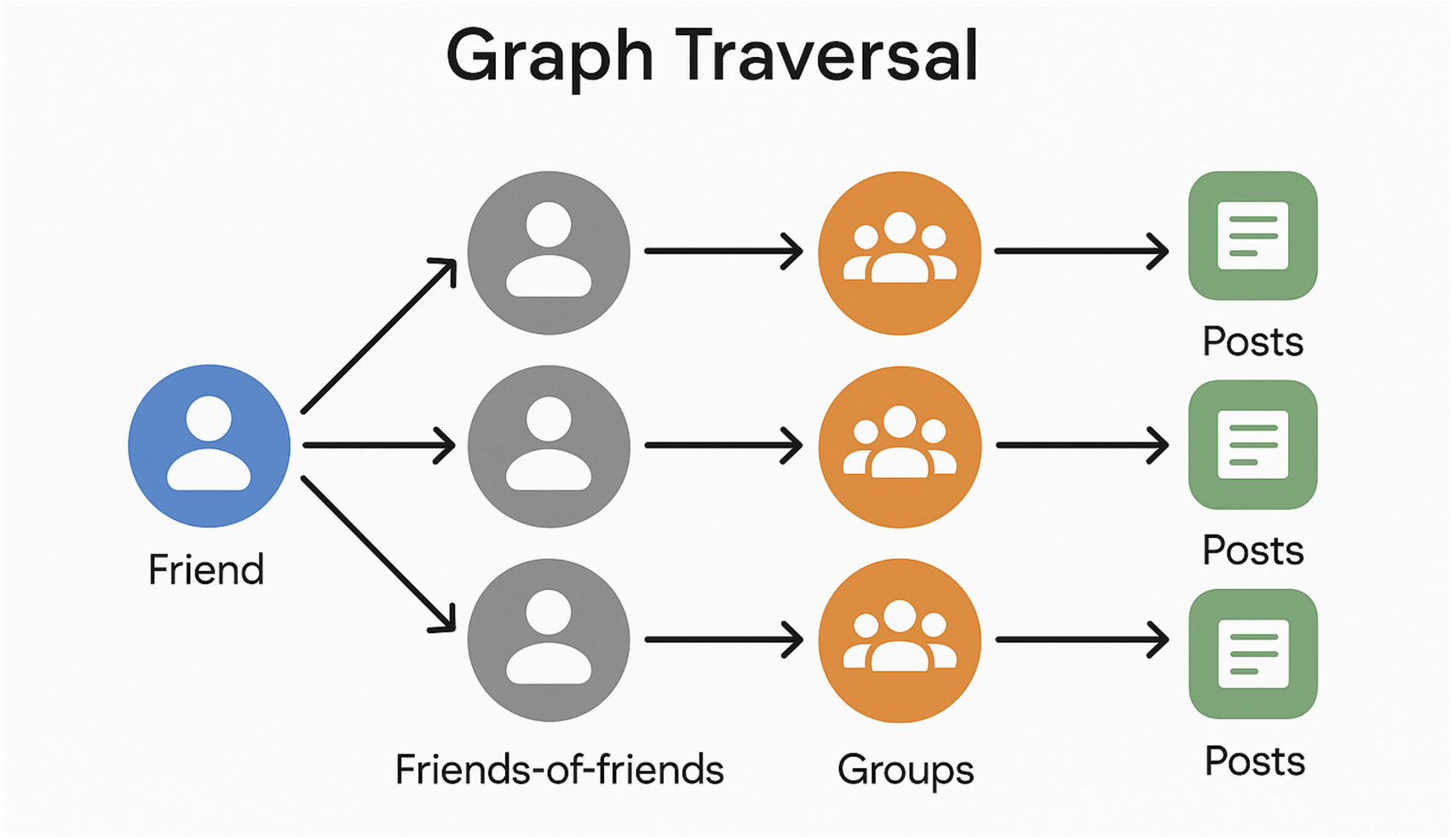
Imagine extending this to groups, posts, or patient-doctor relationships — joins explode.

Graph Database Approach

```
MATCH (user:Person {id: $userId}) -
[:FRIENDS_WITH*2]-> (friendOfFriend:Person)
WHERE NOT (user)-[:FRIENDS_WITH]->
(friendOfFriend) RETURN friendOfFriend.name
```

Adding new signals (shared groups, same city) is just more MATCH lines.

What happens as relationships deepen?

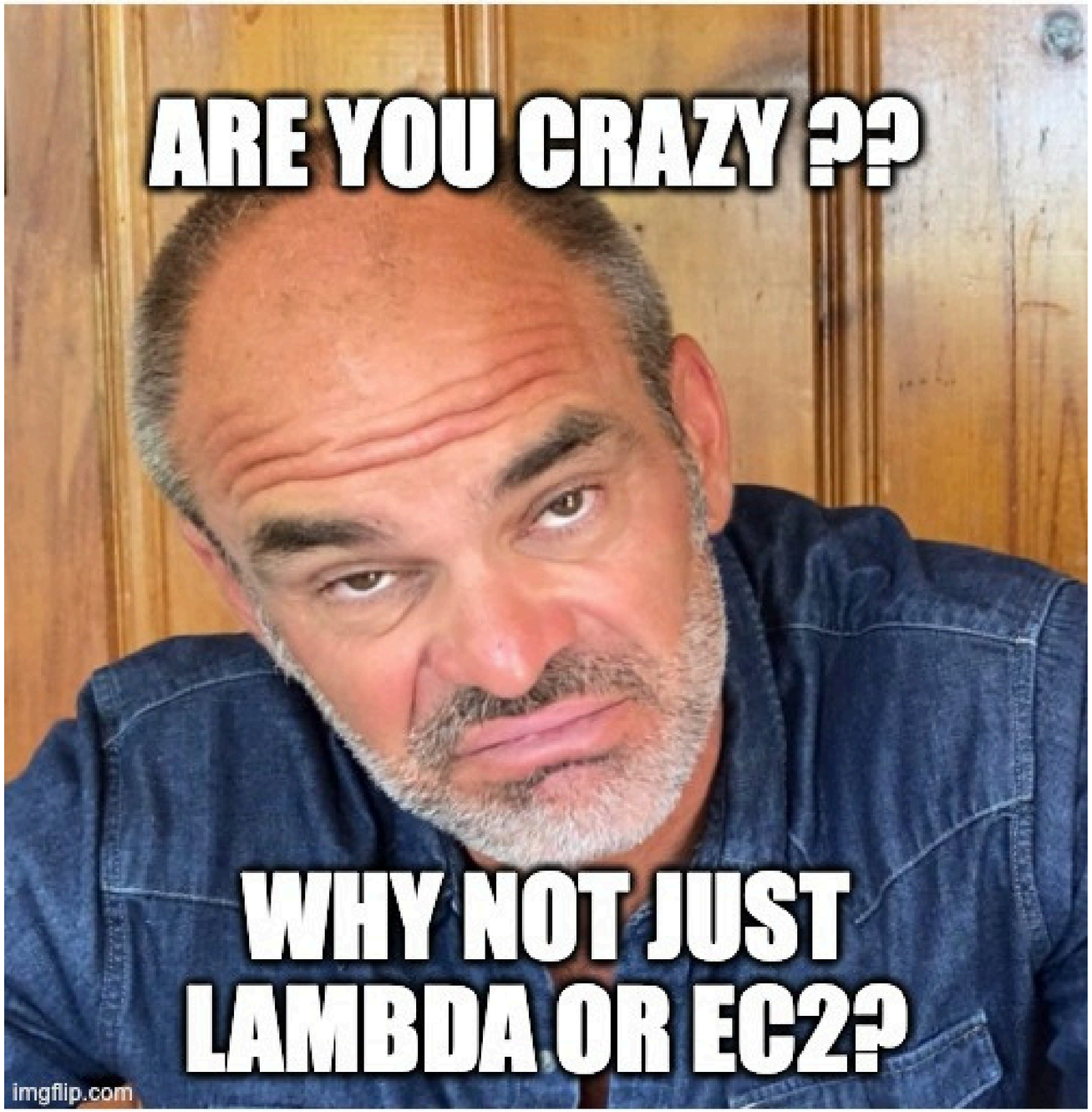


Each hop = one more JOIN in SQL

When to use Neo4j on AWS

Official AWS Blog

Lets use Fargate for API



Why Fargate for the API?

- Always-on connections → Neo4j driver stays warm; no churn like in Lambda.
- Predictable scaling → ECS Service Auto Scaling on CPU/p95 latency.
- No execution cap → good for APIs and batch precompute (vs Lambda 15-min).
- Serverless containers → no EC2 to manage, but more control than Lambda.

Why Fargate for the API?

Axis	Fargate (ECS)	Lambda	EC2
Process model	Always-on, warm pools	Ephemeral, cold starts	Always-on, manual mgmt
Scaling	Auto scaling by CPU/mem/latency	Event concurrency scaling	Auto Scaling Groups
Limits	No cap	15-min timeout	No cap
Ops	Serverless containers	Fully serverless	Full server mgmt

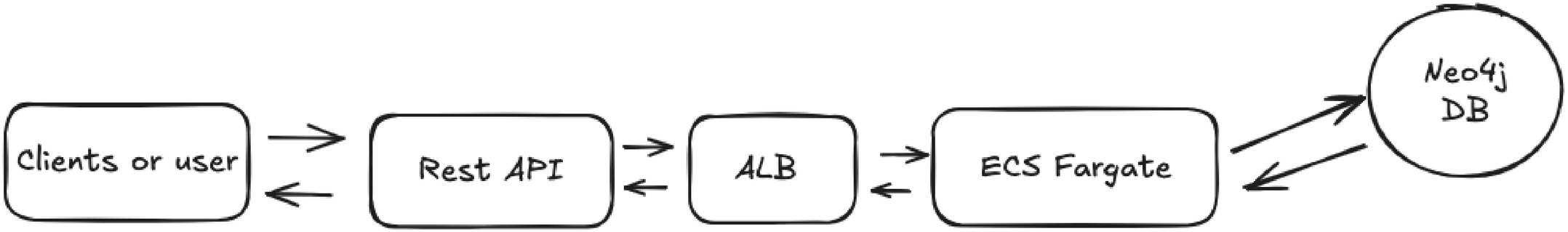
Hypothesis

- Graph queries are natural in Neo4j.
- Fargate can scale the API tier reliably under load.

Questions I wanted to test

- How does API latency behave under spiky traffic?
- Does Fargate scaling protect the app tier while graph queries get heavier?
- What are the trade-offs of this design (Fargate vs Lambda vs EC2)?

Architecture Overview



Dataset Overview

Users — basic profiles (name, city).

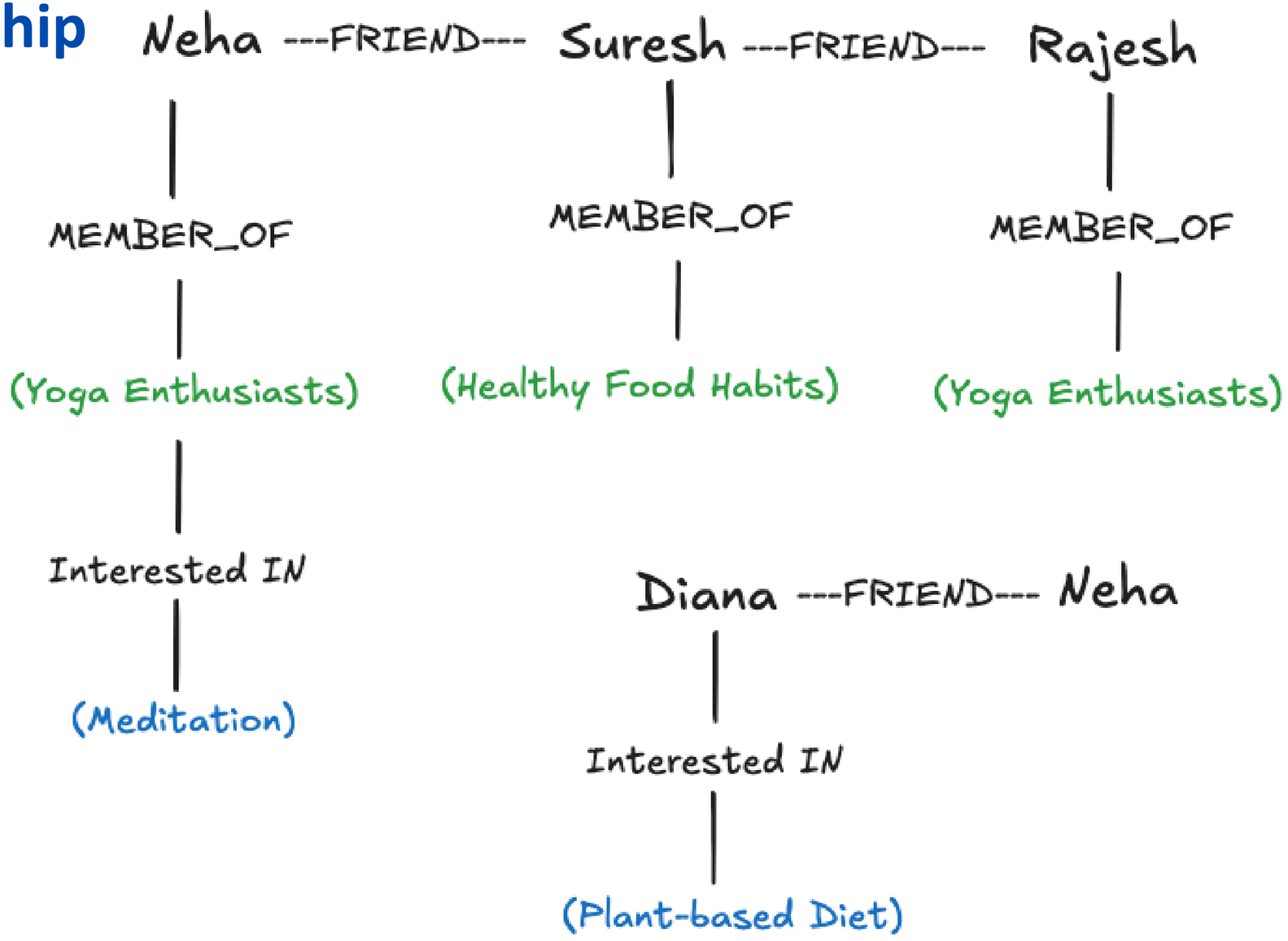
Relationships:

- FRIEND → user ↔ user.
- MEMBER_OF → user → group.
- INTEREST_IN → user → interest/topic.
- LIKES → user → post.

Dataset Overview

- Designed to mimic real-world patterns:
- Friend-of-friend recommendations.
- Group/community discovery.
- Shared interests boosting recommendations.

Understand relationship



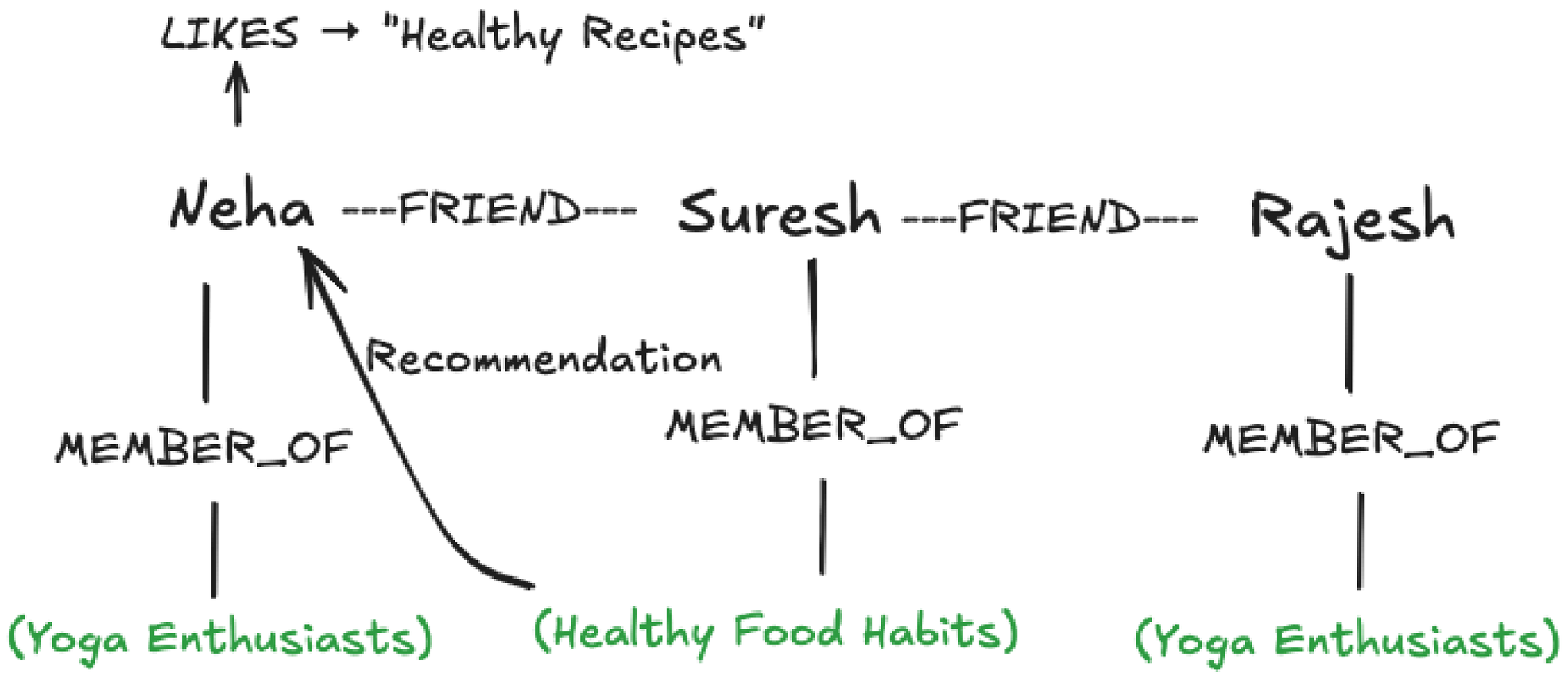
What does the LIKES relationship mean?

Neha → LIKES → "Healthy Recipes"

Rajesh → LIKES → "Morning Yoga Routine"

This is like Facebook/Instagram “likes” → a signal of interest.

How does this help recommendations?



How do we query this in Neo4j?

```
MATCH (u:User {name: "Neha"})-[:LIKES]->(p:Post)<-[:LIKES]-(other:User)-[:MEMBER_OF]->(g:Group)
WHERE NOT (u)-[:MEMBER_OF]->(g)
RETURN DISTINCT g.name AS RecommendedGroup
```

lets do demo

Lessons Learned

- Graph queries scale in complexity, not just size.
- Fargate is a good fit for the stateless API layer, not the DB.
- For heavy analytics (>15 min): run as ECS tasks or AWS Batch.
- For triggers/glue: use Lambda.

Implementation Tips

- Connection pooling is critical: Reuse database connections across requests
- Right-size your containers: Don't over-provision CPU/memory for graph queries
- Cache wisely: Graph queries can be expensive; cache results when appropriate
- Test realistic scenarios: Load test with actual graph traversal patterns
- Plan for growth: Design data models that scale with your graph size

Whats next ??


THANK YOU!
Connect with me to learn more!
AWS
FOR PRODUCT BUILDERS