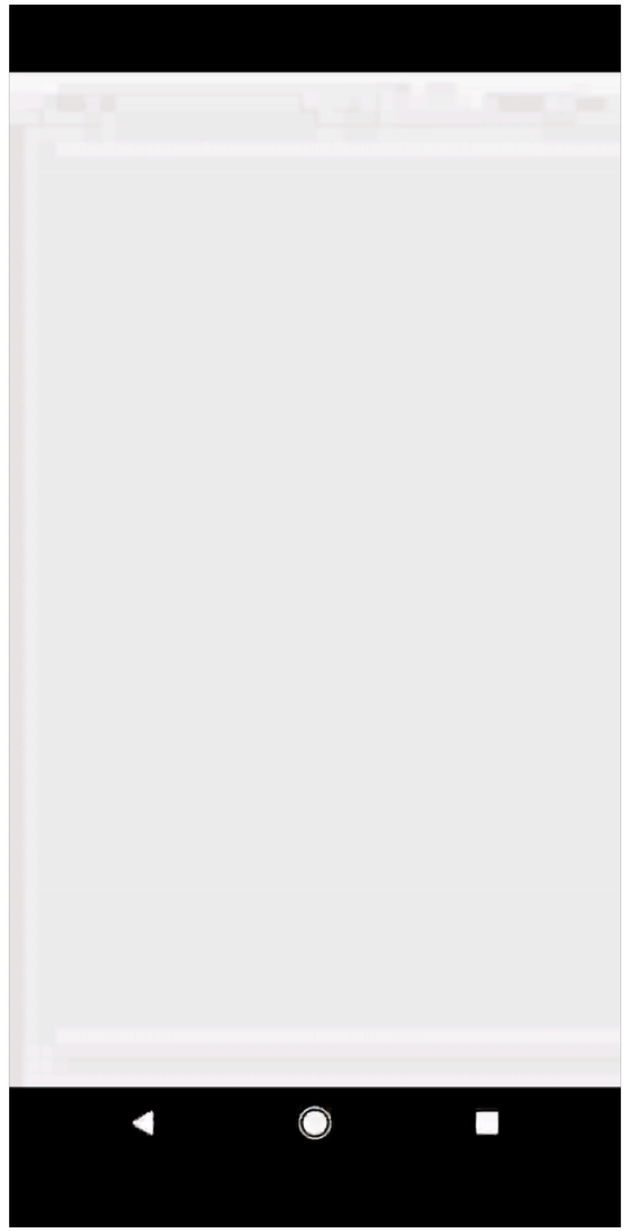
```
Query DroidConNYC{ slide(id: "1")
    {
     Title
     Authors
     Company
    }
}

{
  Title: Intro to GraphQL on Android,
  Authors:  [“Brian Plummer”,”Mike Nakhimovich],
  Company: New York Times
}
```

# We work at NYTimes

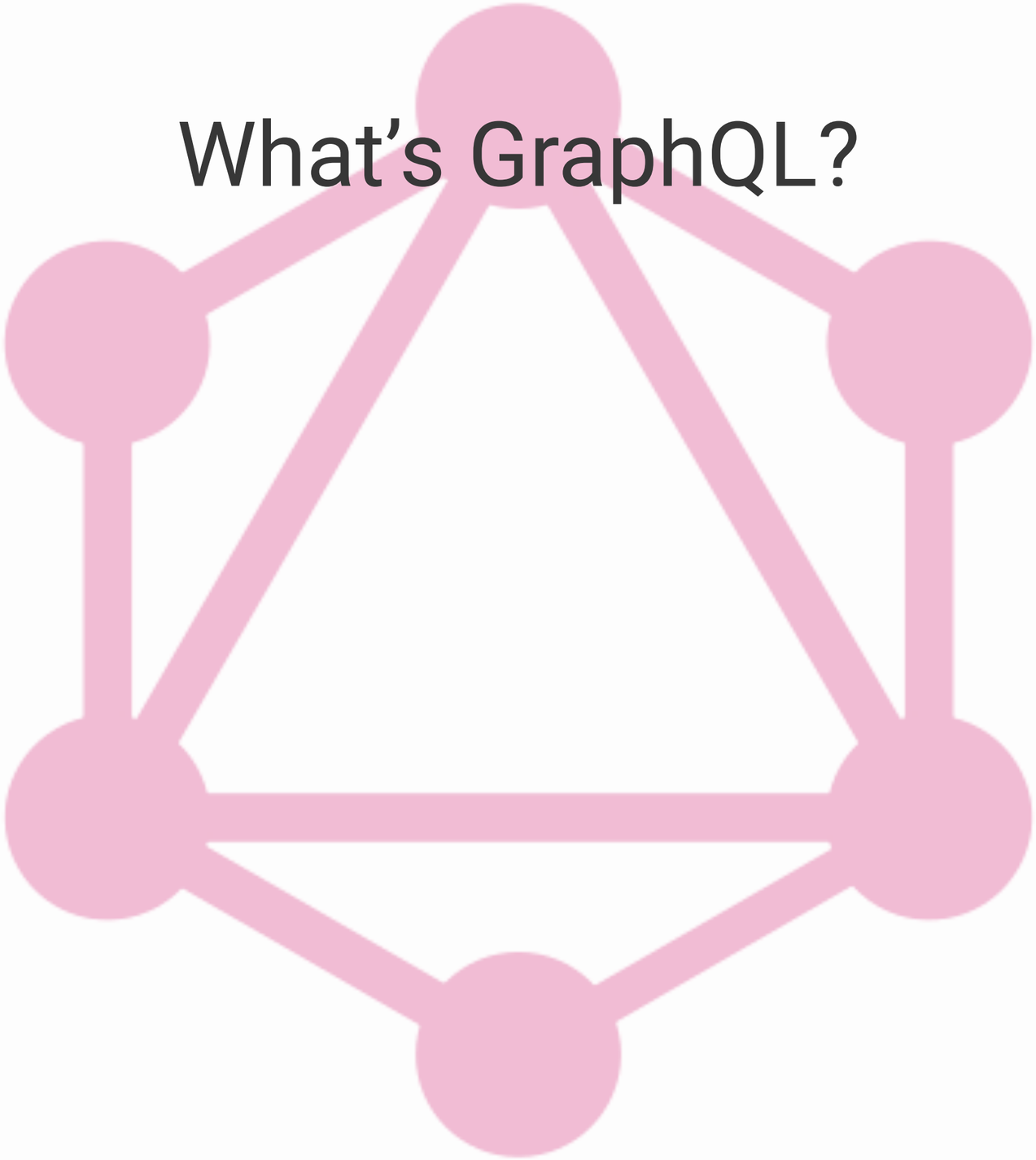
Where we do ***a lot*** of data loading



Recently our team began moving from Restful APIs to instead use GraphQL

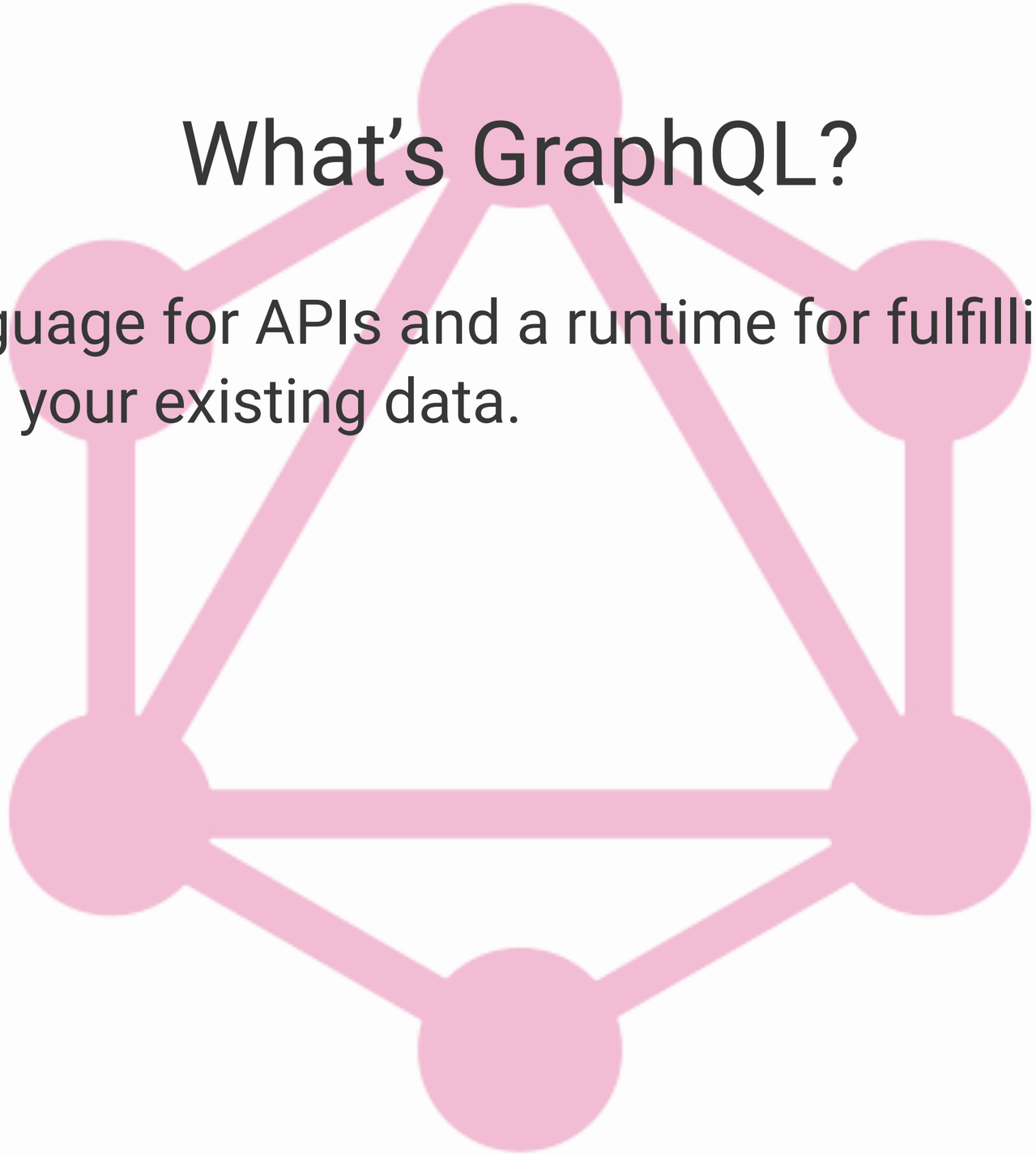
Before we dive into how we did it, lets start with a primer on what graphql is and why you should care

# What’s GraphQL?

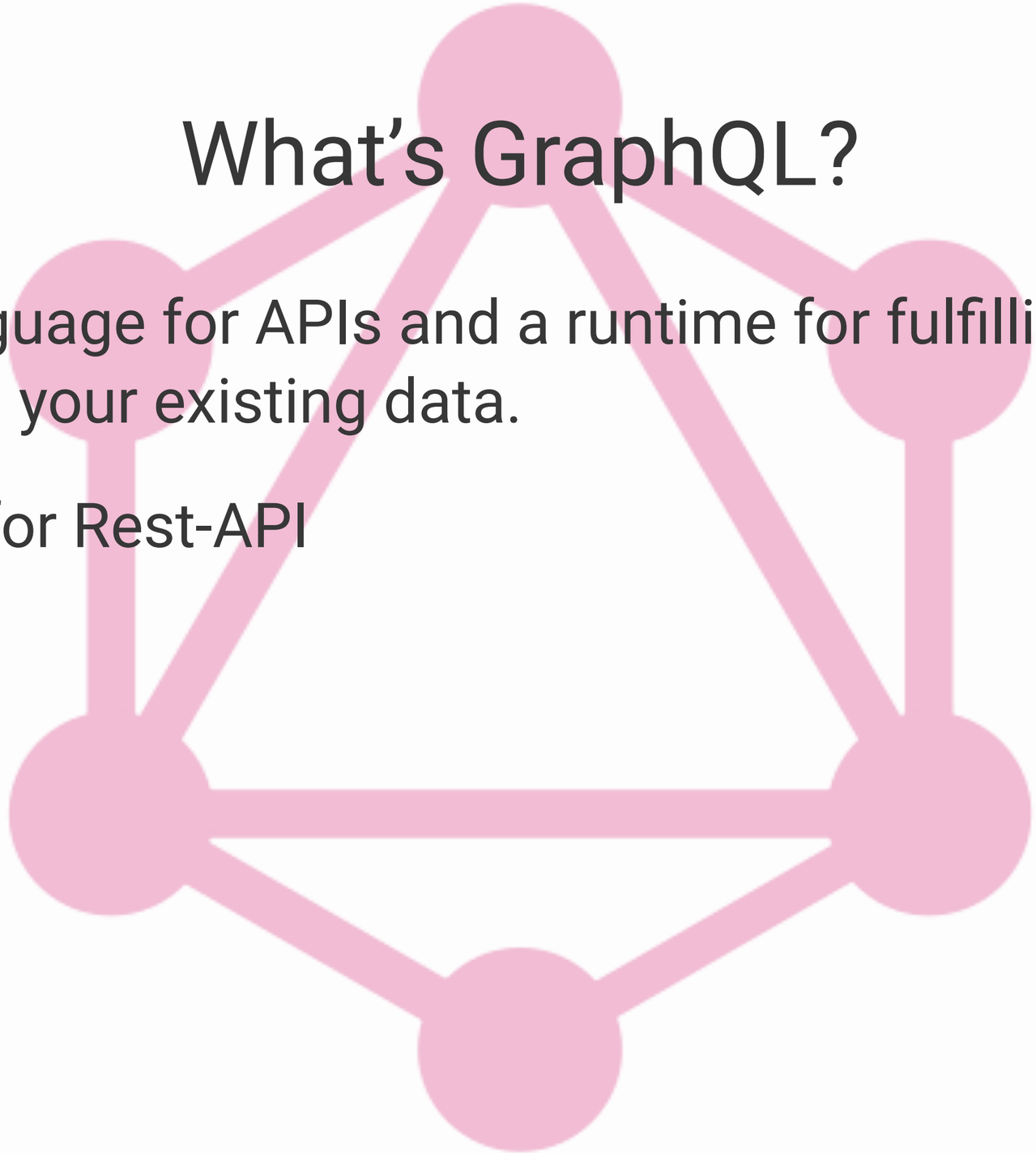


# What’s GraphQL?

- A query language for APIs and a runtime for fulfilling those queries with your existing data.



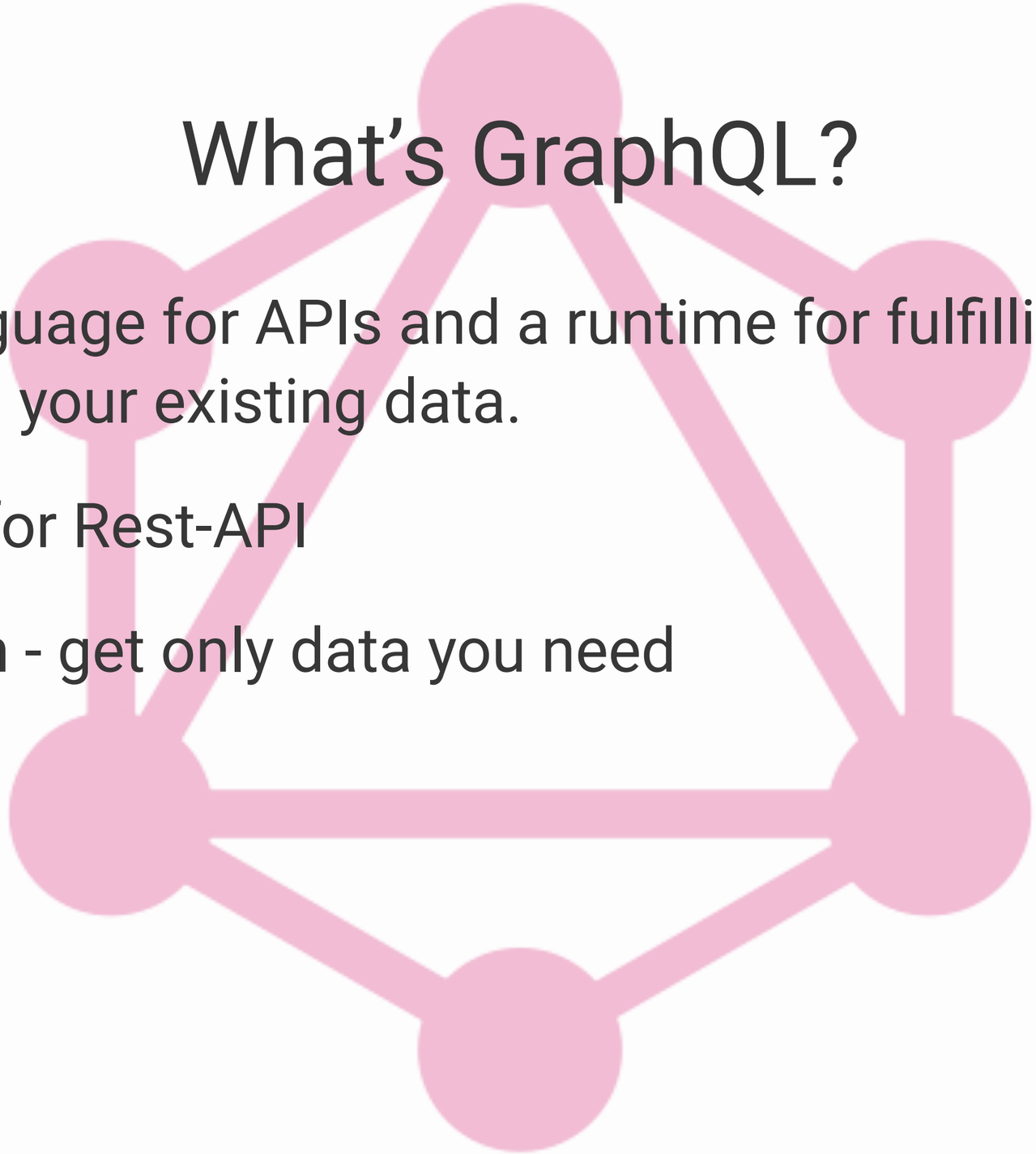
# What’s GraphQL?



- A query language for APIs and a runtime for fulfilling those queries with your existing data.
- Alternative for Rest-API

# What’s GraphQL?

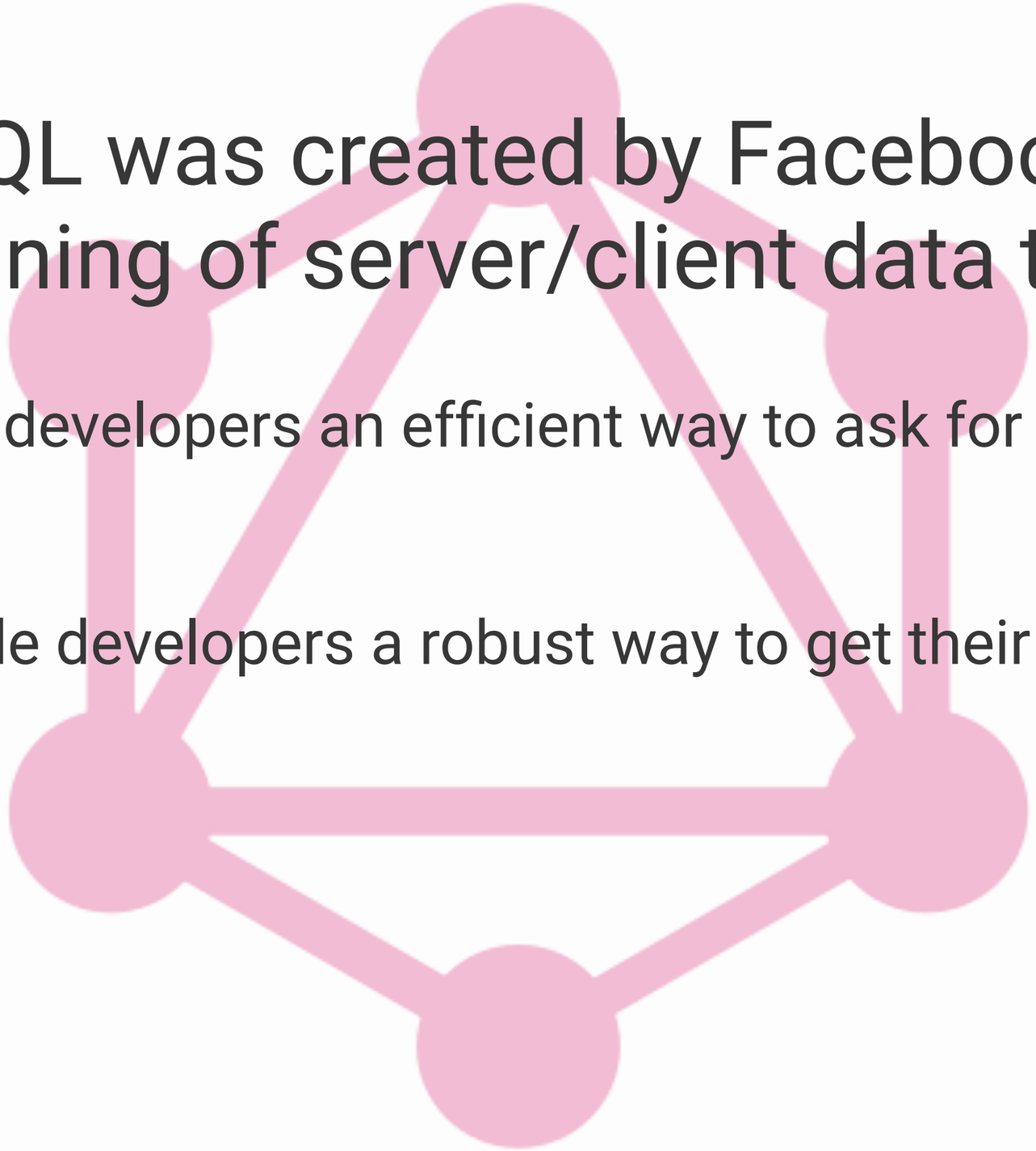
- A query language for APIs and a runtime for fulfilling those queries with your existing data.
- Alternative for Rest-API
- Client driven - get only data you need



# What’s GraphQL?

- A query language for APIs and a runtime for fulfilling those queries with your existing data.
- Alternative for Rest-API
- Client driven - get only data you need
- Works on iOS, Android, Web

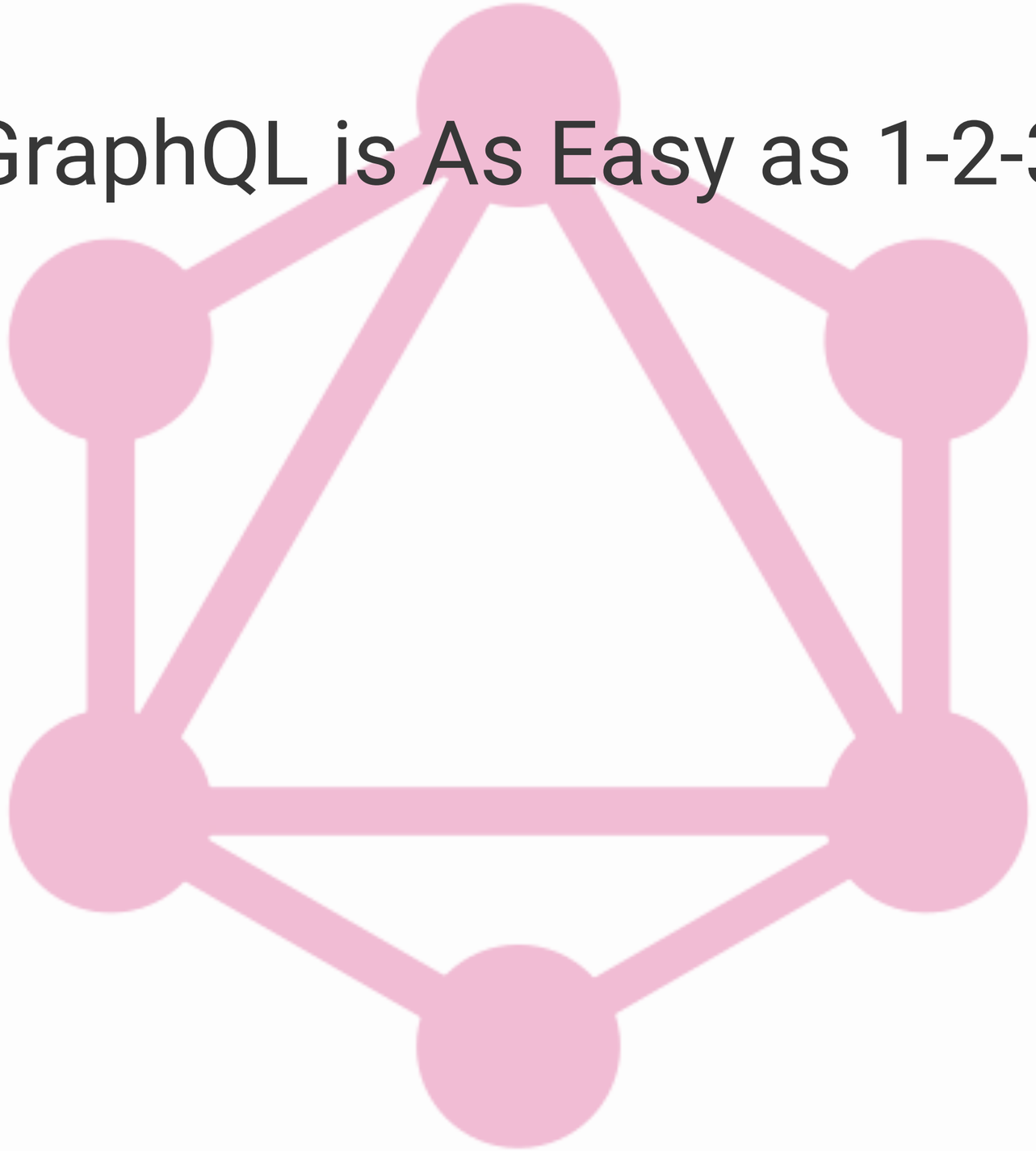
# GraphQL was created by Facebook as a reimagining of server/client data transfer



Give front end developers an efficient way to ask for minimal data

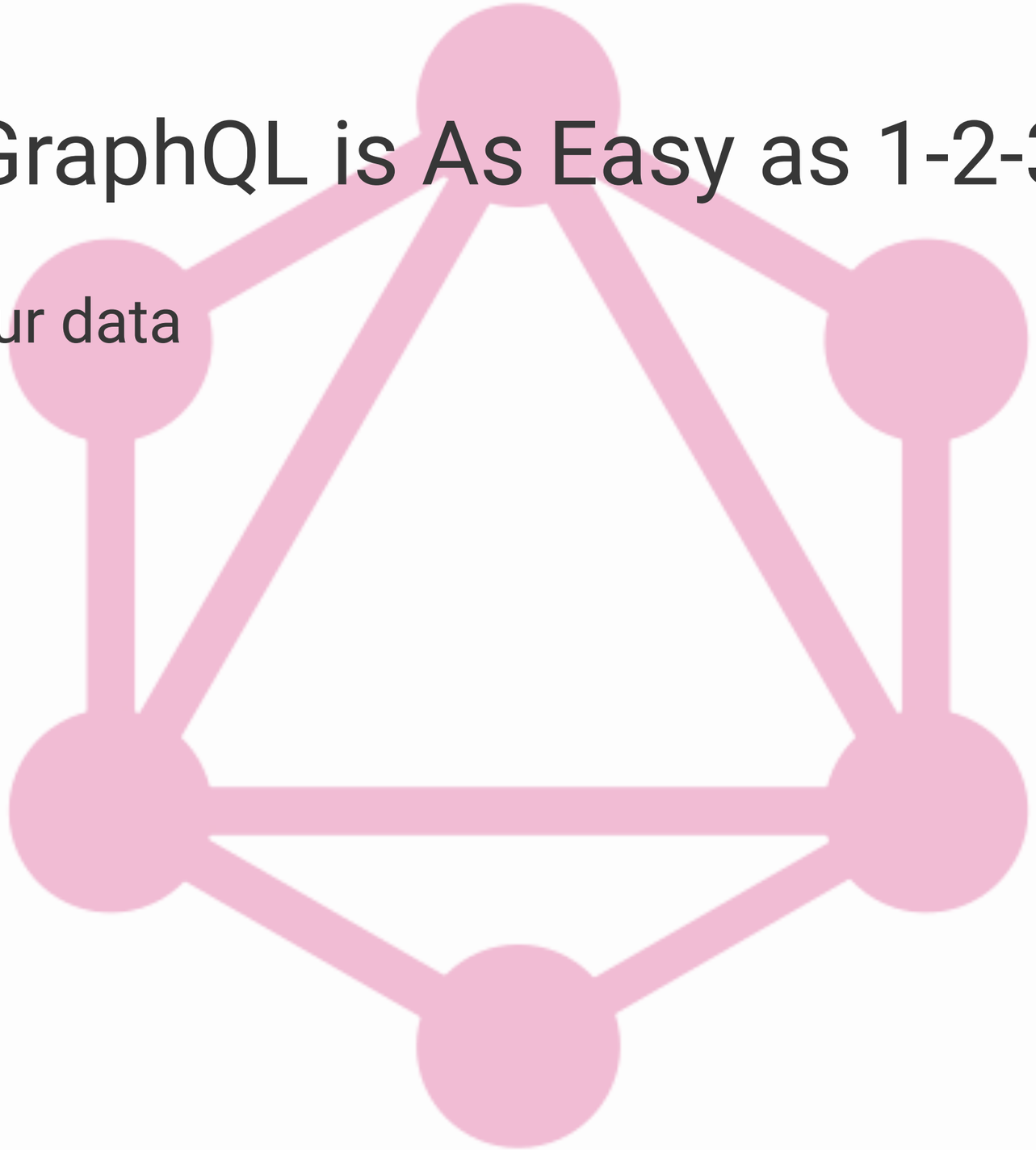
Give server-side developers a robust way to get their data out to their users.

# GraphQL is As Easy as 1-2-3



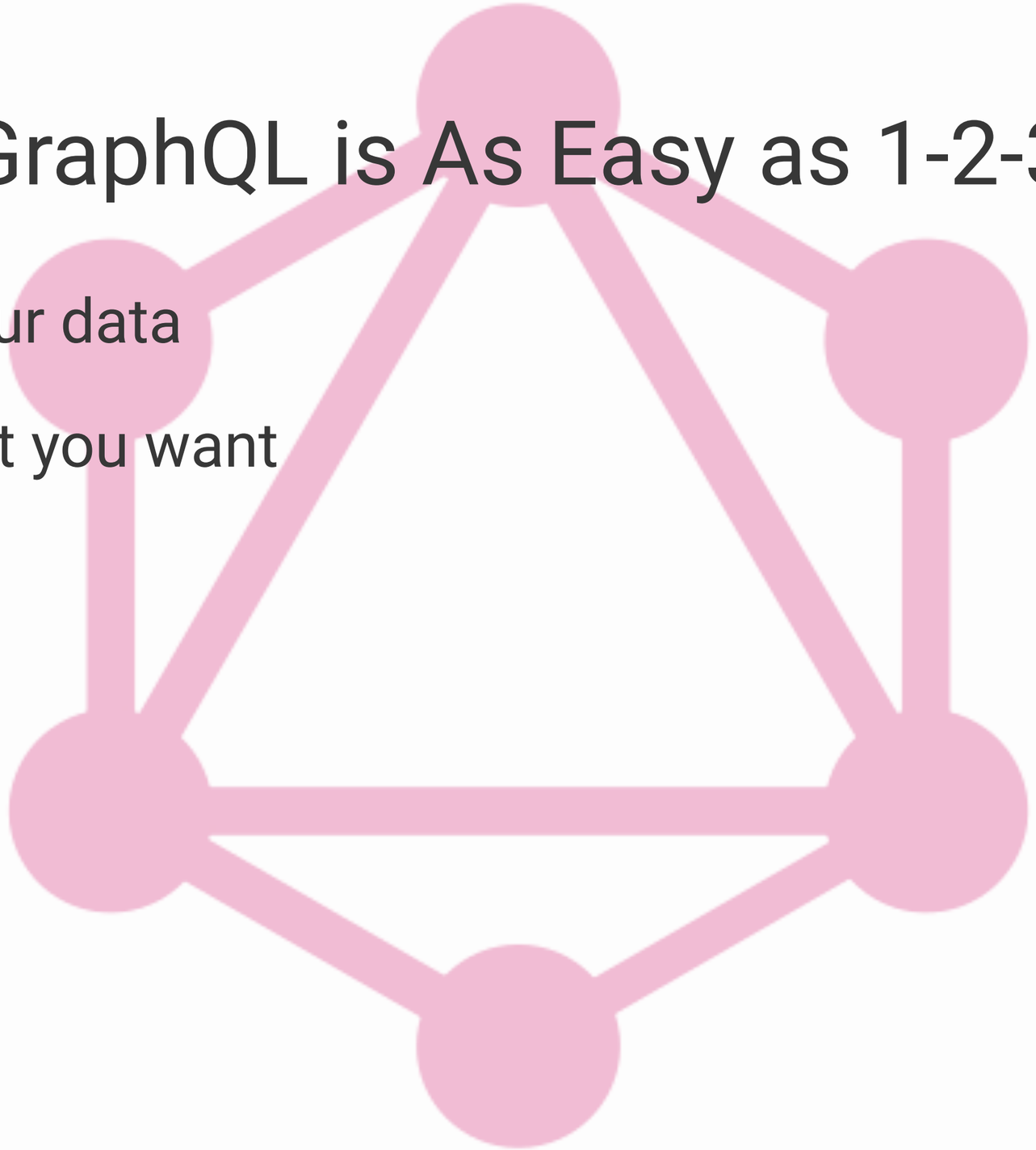
# GraphQL is As Easy as 1-2-3

- Describe your data



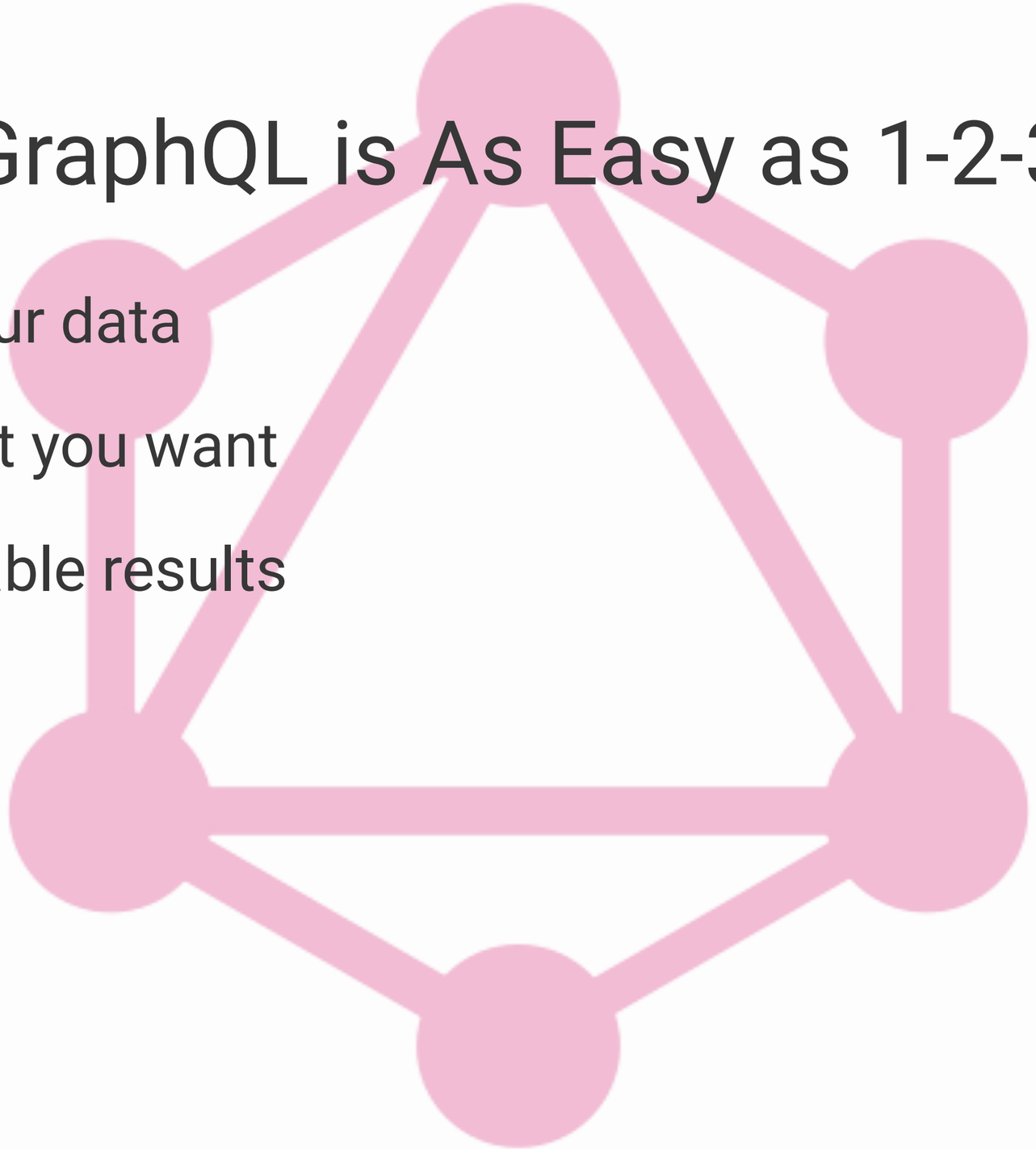
# GraphQL is As Easy as 1-2-3

- Describe your data
- Ask for what you want



# GraphQL is As Easy as 1-2-3

- Describe your data
- Ask for what you want
- Get predictable results



# Describe your data in a schema

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

# Describe your data in a schema

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

# Describe your data in a schema

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

- **Character** is a GraphQL Object Type, meaning it's a type with some fields. Most of the types in your schema will be object types.

# Describe your data in a schema

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

# Describe your data in a schema

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

- **name** and **appearsIn** are fields on the Character type. That means that name and appearsIn are the only fields that can appear in any part of a GraphQL query that operates on the Character type.

# Describe your data in a schema

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

# Describe your data in a schema

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

- **String** is one of the built-in scalar types - these are types that resolve to a single scalar object, and can't have sub-selections in the query.

# Graphql Example Schema

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

# Graphql Example Schema

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

- **String!** means that the field is non-nullable, meaning that the GraphQL service promises to always give you a value when you query this field.

# Graphql Example Schema

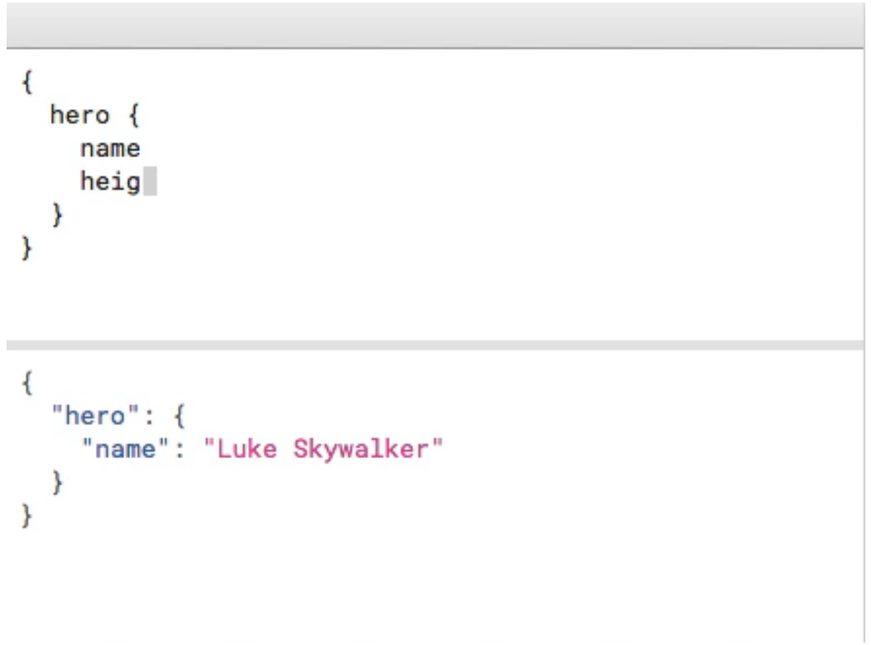
```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

# Graphql Example Schema

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

- **[Episode]!** represents an array of Episode objects. Since it is also non-nullable, you can always expect an array (with zero or more items) when you query the appearsIn field.

# Ask for what you need



```
{
  hero {
    name
    heig
  }
}
```

```
{
  "hero": {
    "name": "Luke Skywalker"
  }
}
```

# get predictable results

# GraphQL let's you combine resources in one request

```
{
  hero {
    name
    # Queries can have comments!
    friends {
      name
    }
  }
}
```

# GraphQL let's you combine resources in one request


```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
    }
  }
}
```

You can also reuse fields in a fragment insert fragment example

# **Walkthrough Time!**:

Loading data from Github using Rest vs GraphQL

# On Any Platform Data Loading can be broken down into

# On Any Platform Data Loading can be broken down into

1. Model Data

# On Any Platform Data Loading can be broken down into

1. Model Data
2. Network

# On Any Platform Data Loading can be broken down into

1. Model Data
2. Network
3. Transform

# On Any Platform Data Loading can be broken down into

1. Model Data
2. Network
3. Transform
4. Persist

# How does REST look on Android?

# Like a lot of dependencies

| **Data Modeling** | **Networking** | **Storage** | **Transform** |
|---|---|---|---|
| Immutables | OKhttp | Store | Gson |
| Curl | Retrofit | SqlDelight | RxJava |

Yes those are all needed 😂

```
},
{
  "url": "https://api.github.com/repos/vmg/redcarpet/issues/607",
  "repository_url": "https://api.github.com/repos/vmg/redcarpet",
  "labels_url": "https://api.github.com/repos/vmg/redcarpet/issues/607/labels{/name}",
  "comments_url": "https://api.github.com/repos/vmg/redcarpet/issues/607/comments",
  "events_url": "https://api.github.com/repos/vmg/redcarpet/issues/607/events",
  "html_url": "https://github.com/vmg/redcarpet/issues/607",
  "id": 211546203,
  "number": 607,
  "title": "Low Level Rendering `entity` what how when?",
  "user": {
    "login": "dgsan",
    "id": 313910,
    "avatar_url": "https://avatars2.githubusercontent.com/u/313910?v=4",
    "gravatar_id": "",
    "url": "https://api.github.com/users/dgsan",
    "html_url": "https://github.com/dgsan",
    "followers_url": "https://api.github.com/users/dgsan/followers",
    "following_url": "https://api.github.com/users/dgsan/following{/other_user}",
    "gists_url": "https://api.github.com/users/dgsan/gists{/gist_id}",
    "starred_url": "https://api.github.com/users/dgsan/starred{/owner}{/repo}",
    "subscriptions_url": "https://api.github.com/users/dgsan/subscriptions",
    "organizations_url": "https://api.github.com/users/dgsan/orgs",
    "repos_url": "https://api.github.com/users/dgsan/repos",
    "events_url": "https://api.github.com/users/dgsan/events{/privacy}",
    "received_events_url": "https://api.github.com/users/dgsan/received_events",
    "type": "User",
    "site_admin": false
  },
```


# Start with Inspection

***curl -i "https://api.github.com/repos/vmg/redcarpet/issues?state=closed"***

# Model Your Data

```
interface Issue {
    User user();
    String url();


    interface User {
        long id();
        String name();
    }
}
```

**Error Prone even with Code Generation**

# Data Modeling with Immutables

```
@Value.Immutable
interface Issue {
    User user();
    String url();

    @Value.Immutable
    interface User {
        long id();
        String name();
    }
}
```

**Error Prone even with Code Generation**

# Data Parsing with Gson

```
@Gson.TypeAdapters
@Value.Immutable
interface Issue {
    User user();
    String url();

    @Value.Immutable
    interface User {
        long id();
        String name();
    }
}
```

# Networking

```
open fun provideRetrofit(gson: Gson, okHttpClient: OkHttpClient): GithubApi {
        return Retrofit.Builder()
                .client(okHttpClient)
                .baseUrl(BuildConfig.BASE_URL)
                .addConverterFactory(GsonConverterFactory.create(gson))
                .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
                .build()
                .create(GithubApi::class.java!!)}
```

# Storage

```
CREATE TABLE issue (
    _id LONG  PRIMARY KEY AUTOINCREMENT,
    id LONG NOT NULL,
    url STRING,
    title STRING,
    comments INT NOT NULL
}
```

# Storage

```
public abstract class Issue implements IssueModel {
    public static final Mapper<Issue> MAPPER =
            new Mapper<>((Mapper.Creator<Issue>)
            ImmutableIssue::of);

    public static final class Marshal extends IssueMarshal {
    }
}
```

# Storage

```
long insertIssue(Issue issue) {
        if (recordExists(Issue.TABLE_NAME, Issue.ID, String.valueOf(issue.id()))) {
            return 0;
        }

        return db.insert(Issue.TABLE_NAME, new Issue.Marshal()
                .url(issue.url())
                .id(issue.id()));
}
```

# Storage - Memory

```
StoreBuilder.parsedWithKey<GitHubOrgId, BufferedSource, Issues>()
                .fetcher(fetcher)
                .persister(persister)
                .parser(parser)
                .memoryPolicy(MemoryPolicy
                        .builder()
                        .setMemorySize(11L)
                        .setExpireAfterWrite(TimeUnit.HOURS.toSeconds(24))
                        .setExpireAfterTimeUnit(TimeUnit.SECONDS)
                        .build())
                .networkBeforeStale()
                .open()
```

# Thats a good architecture

## It's also not something we can expect a beginner to know

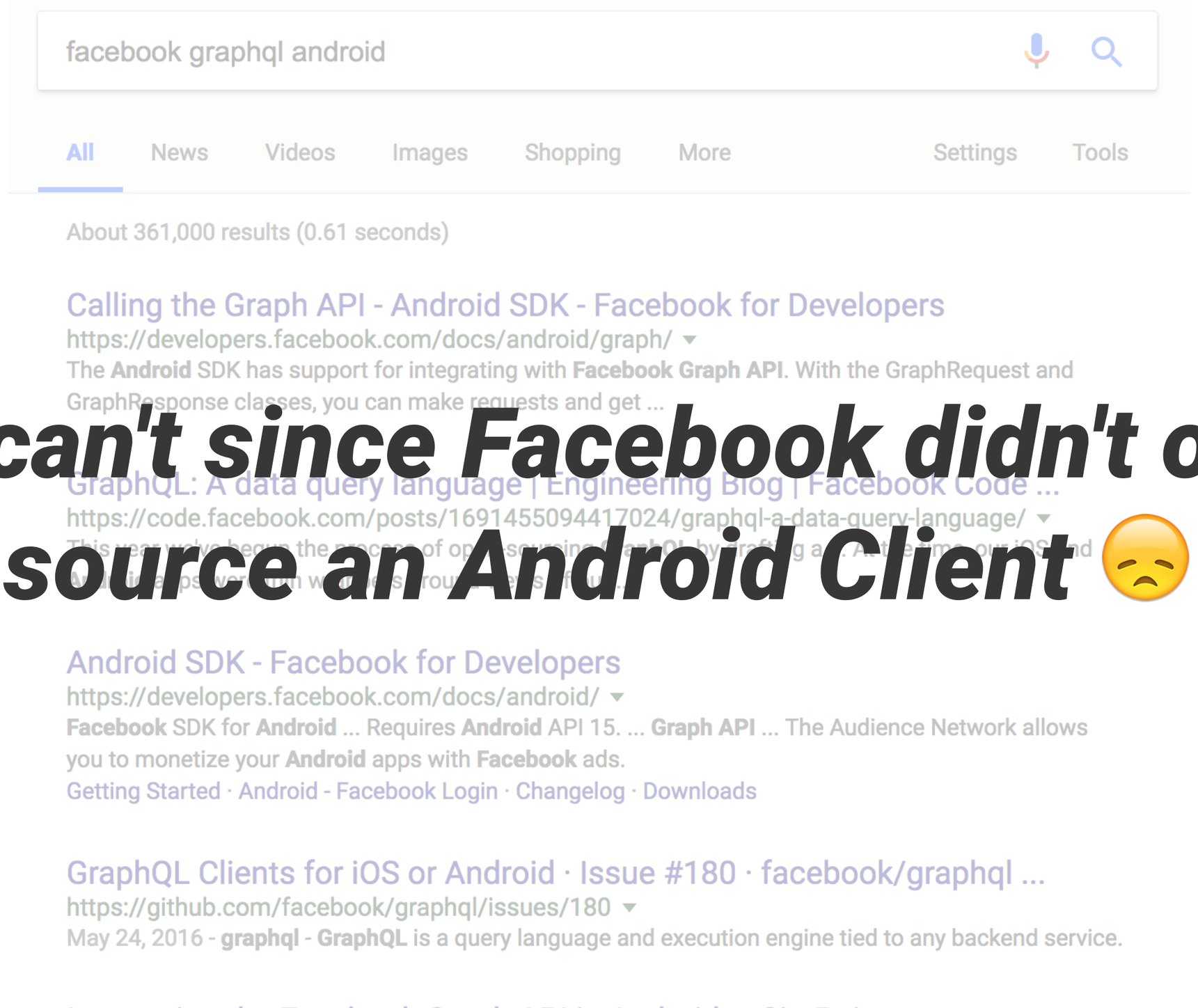
# REST feels like legacy tech

It reminds me of Java with all these great tools & hours of setup to do anything

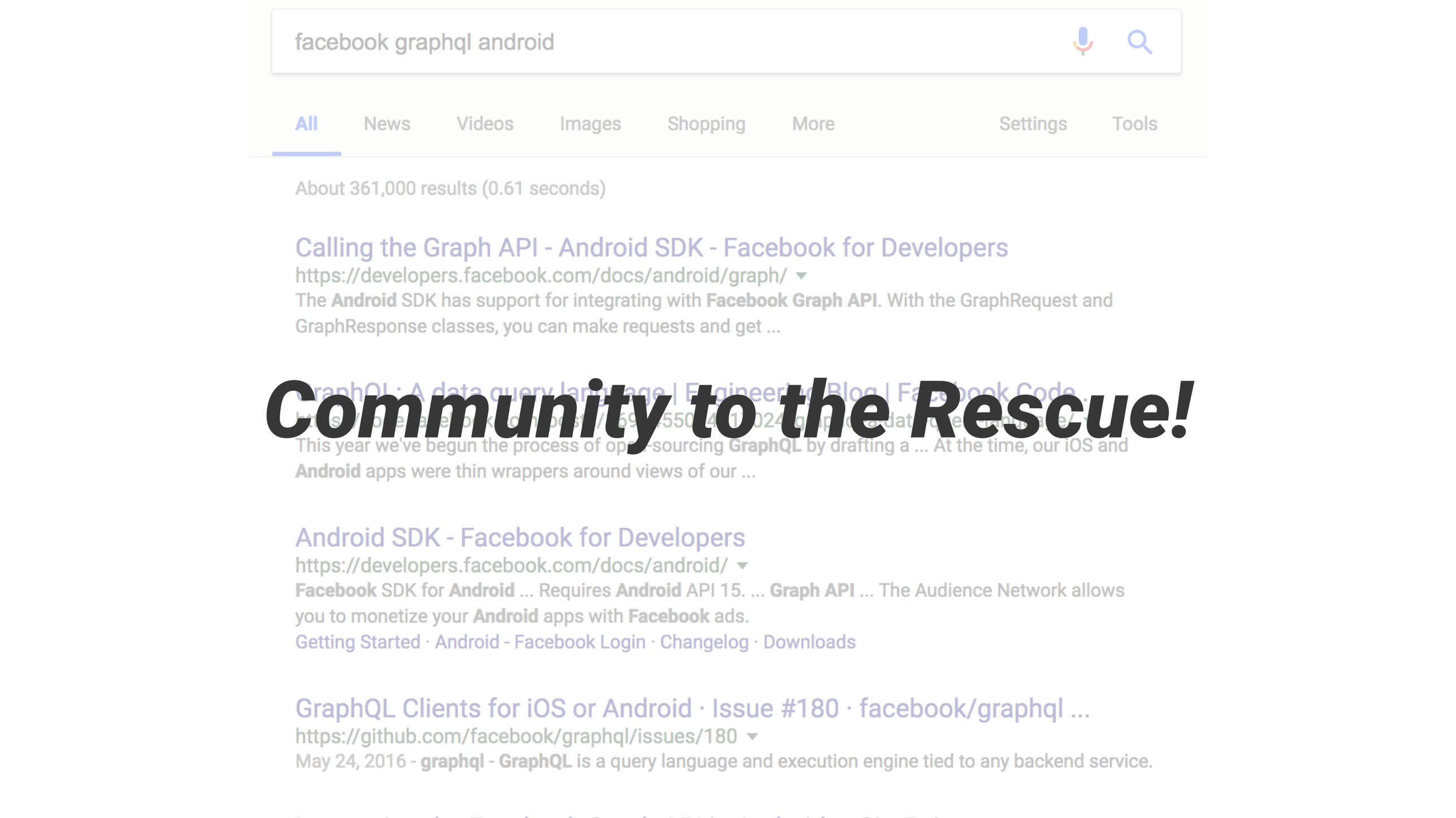
Like we said, there's a new kid on the block

# GraphQL

# Now lets see GraphQL on Android




# *We can't since Facebook didn't open source an Android Client* 😞




# Community to the Rescue!

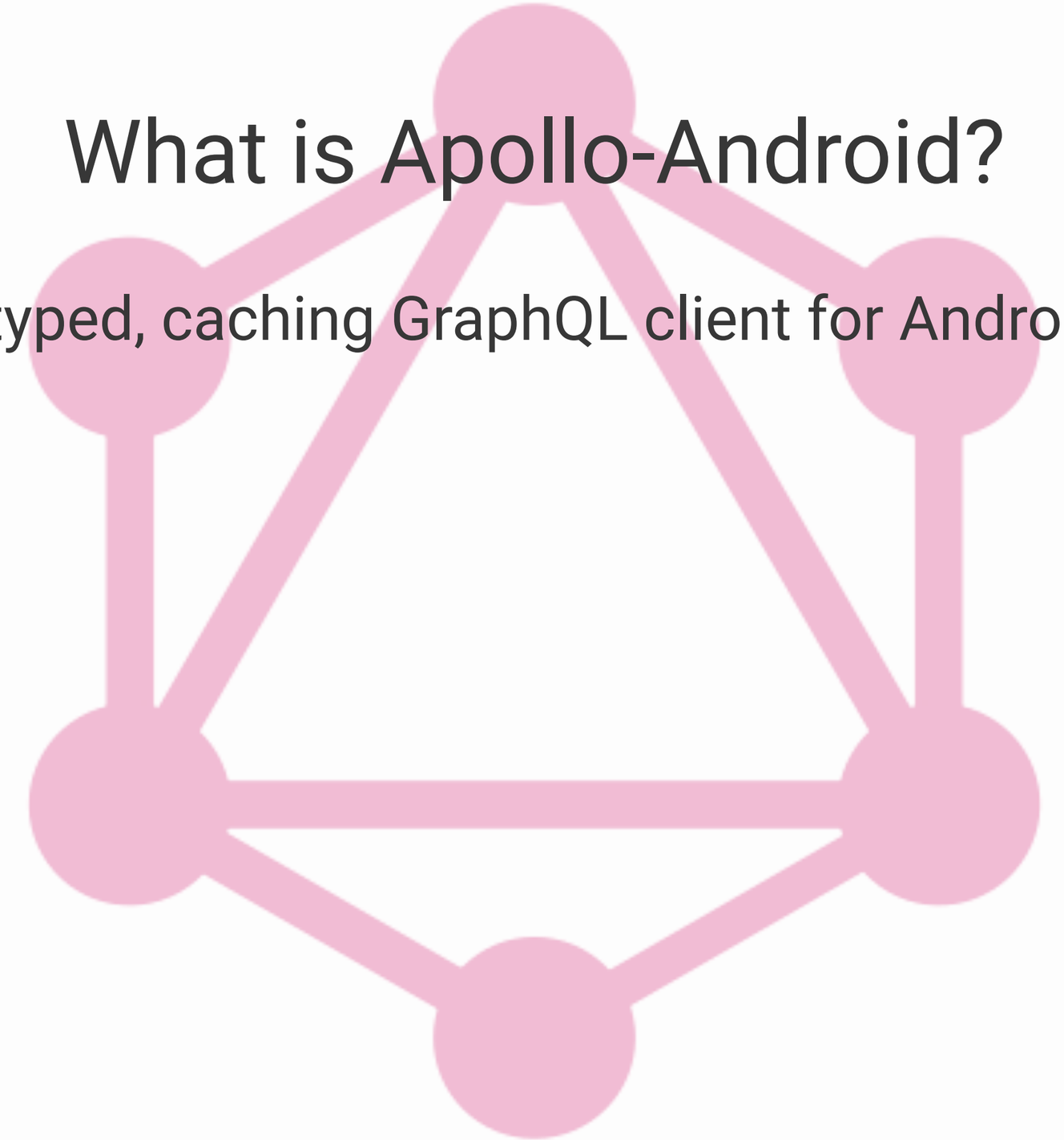
# Introducing Apollo-Android GraphQL


***Apollo Android*** was developed by Shopify, New York Times, & AirBnb as an **Open Source** culmination of tools, libraries, and patterns to assist in fetching data from GraphQL servers

# What is Apollo-Android?

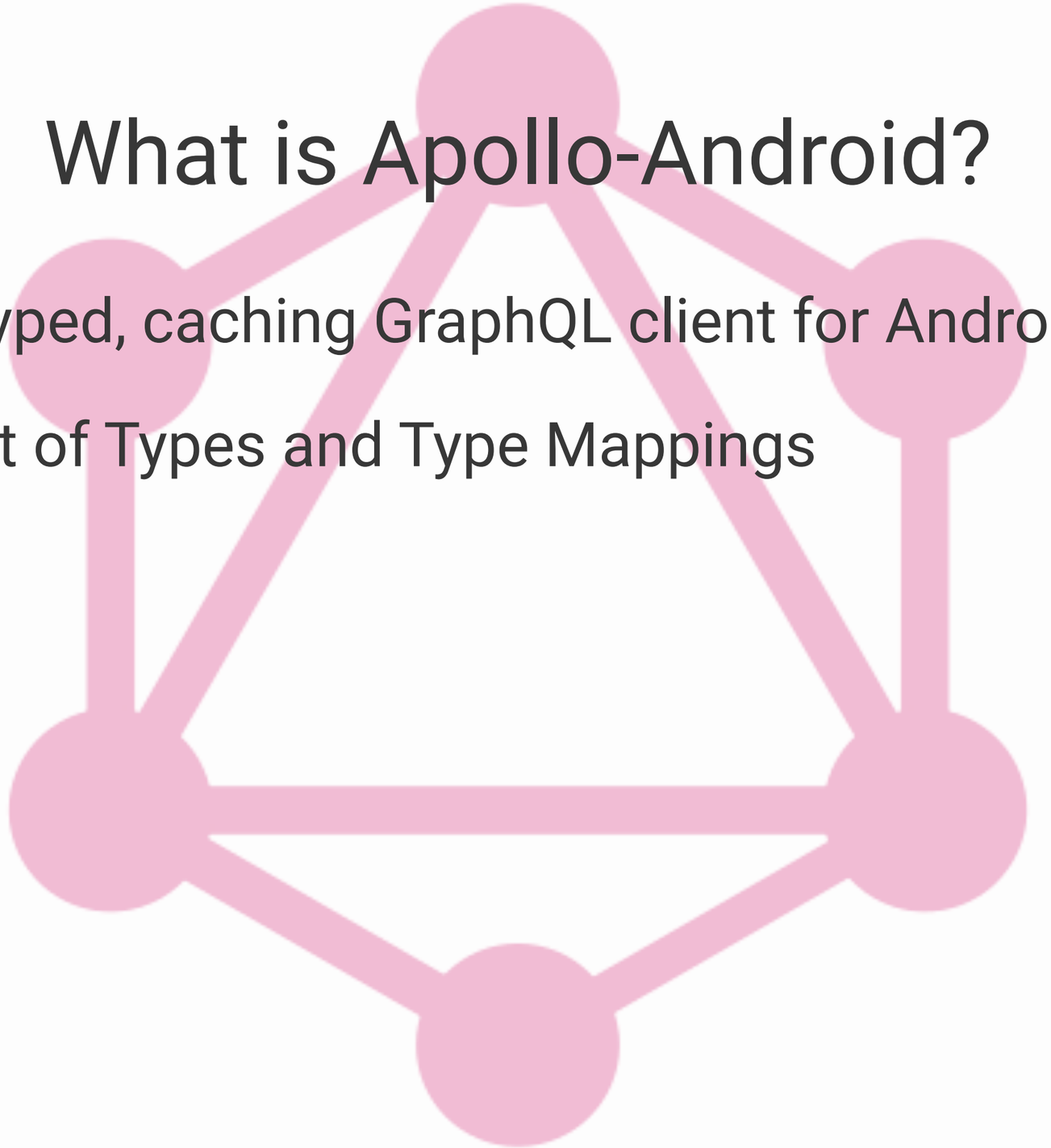
# What is Apollo-Android?

- A strongly-typed, caching GraphQL client for Android

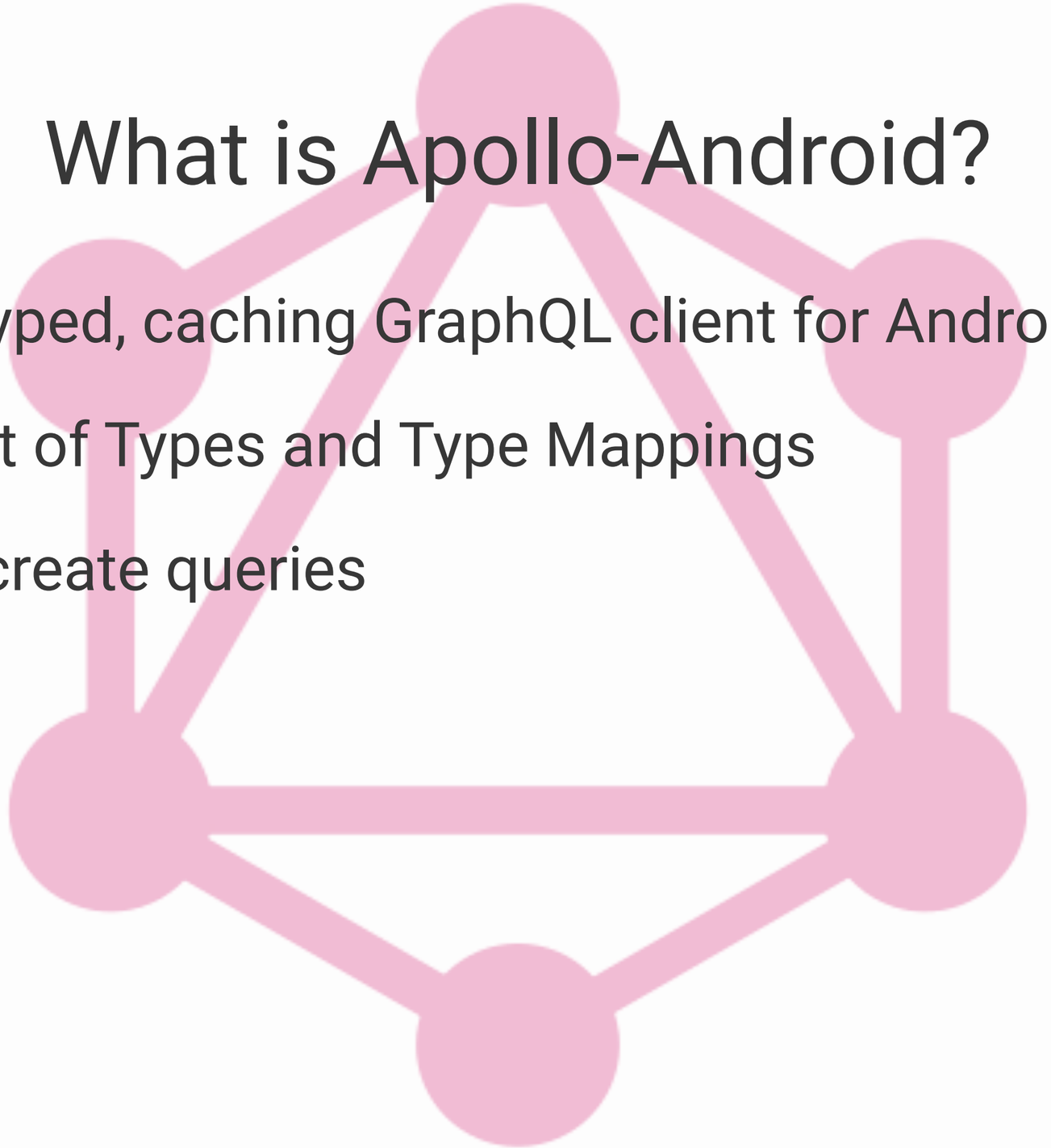


# What is Apollo-Android?

- A strongly-typed, caching GraphQL client for Android
- Rich support of Types and Type Mappings

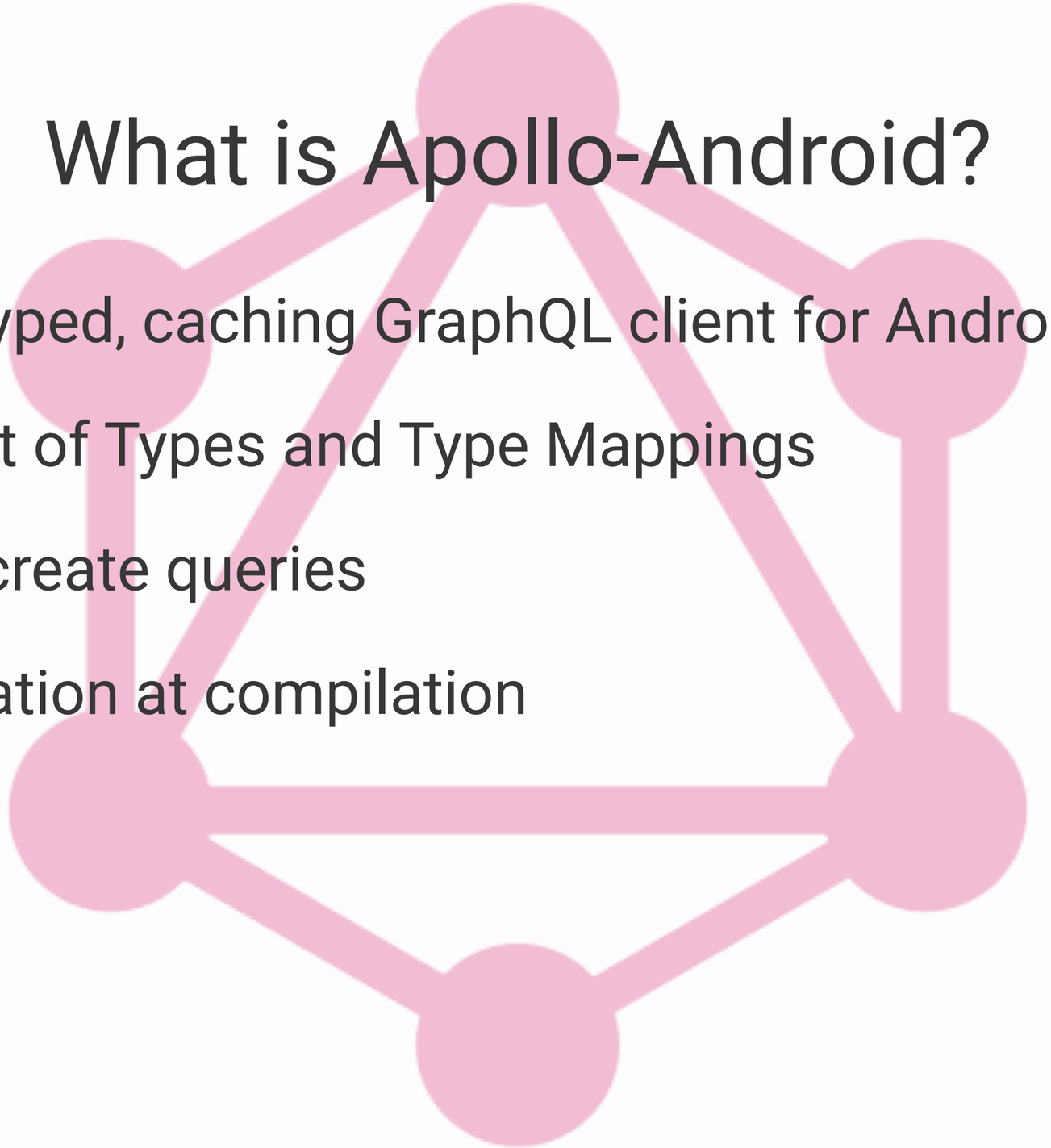


# What is Apollo-Android?



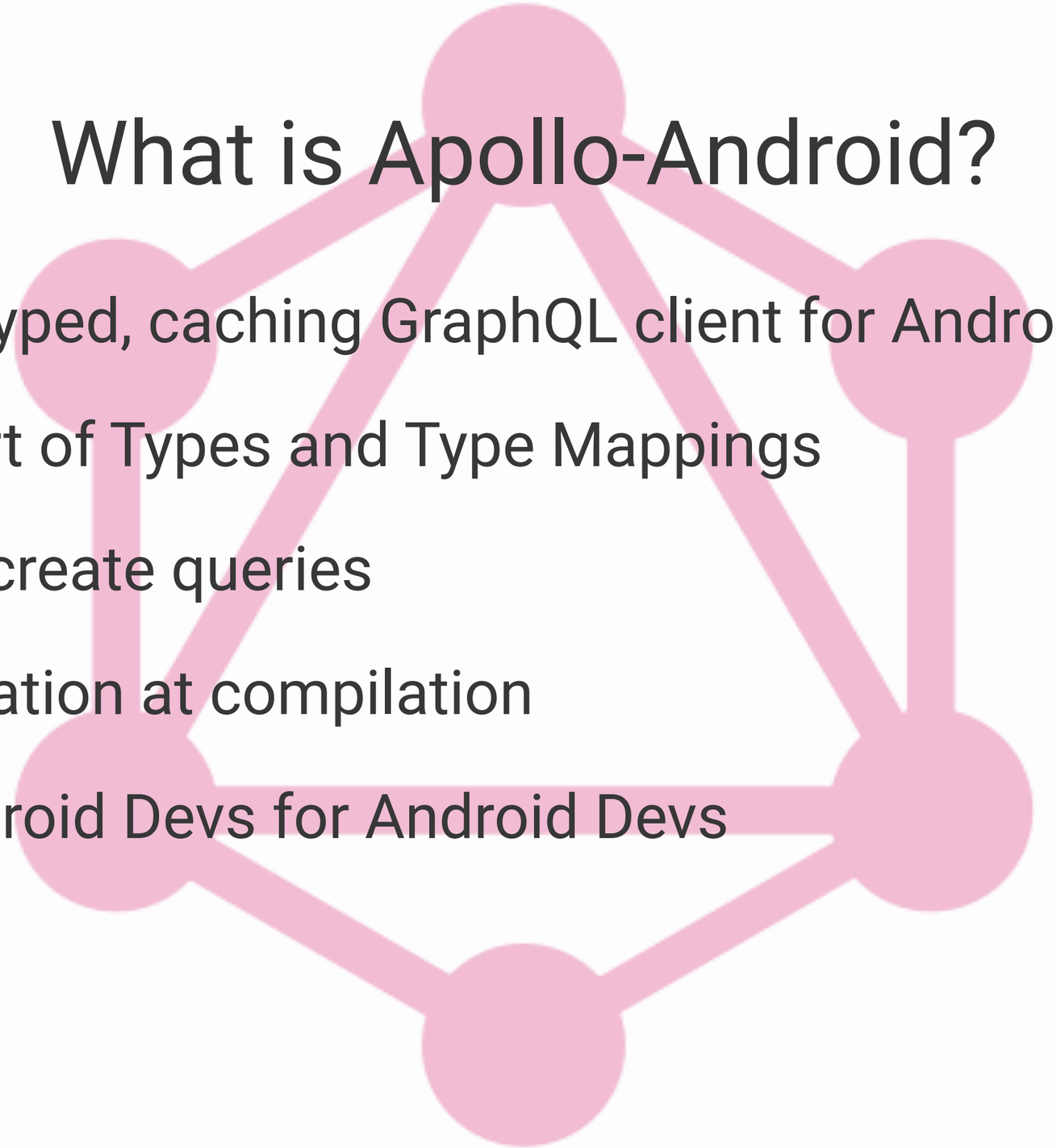
- A strongly-typed, caching GraphQL client for Android
- Rich support of Types and Type Mappings
- Builders to create queries

# What is Apollo-Android?



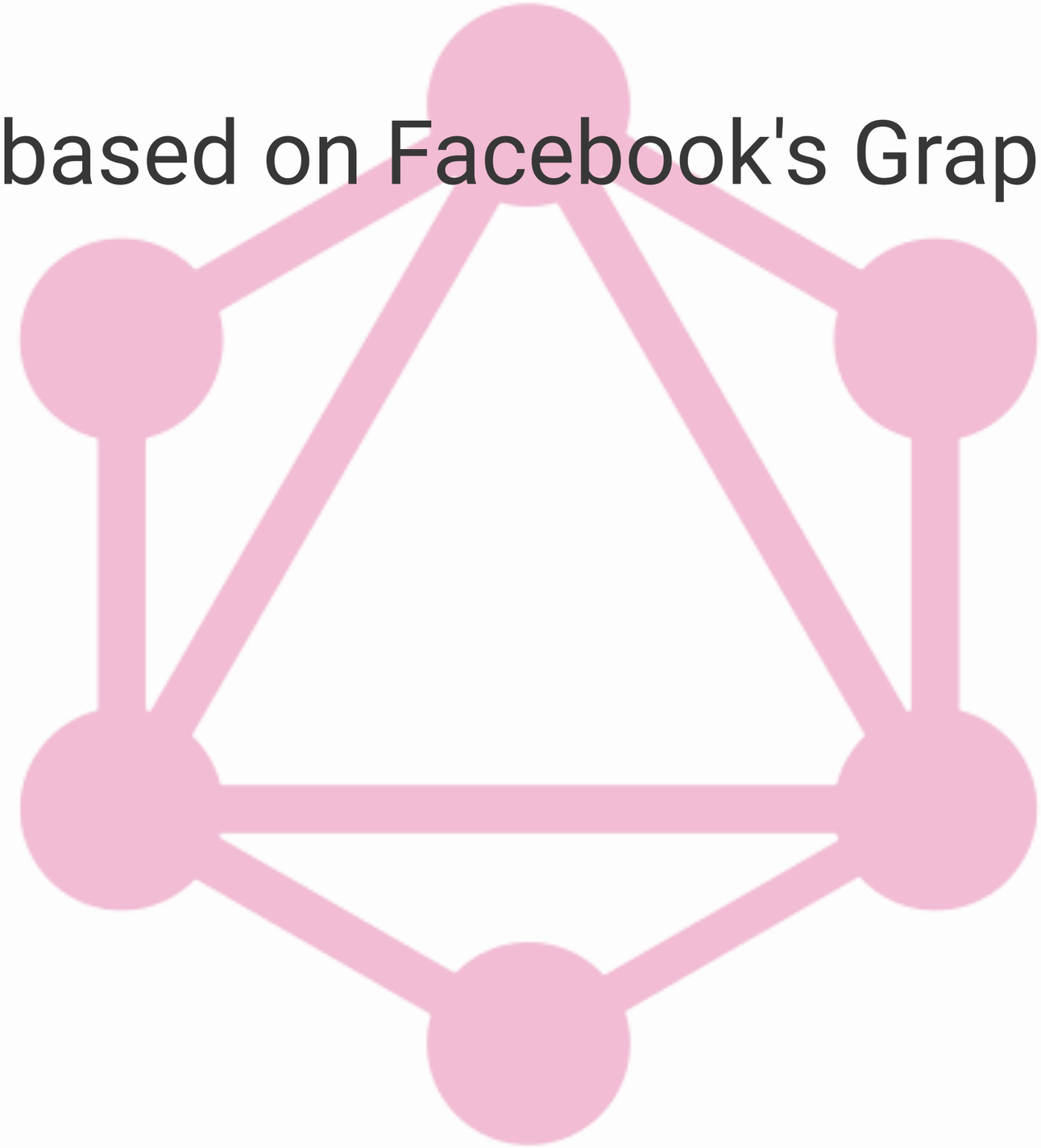
- A strongly-typed, caching GraphQL client for Android
- Rich support of Types and Type Mappings
- Builders to create queries
- Query Validation at compilation

# What is Apollo-Android?



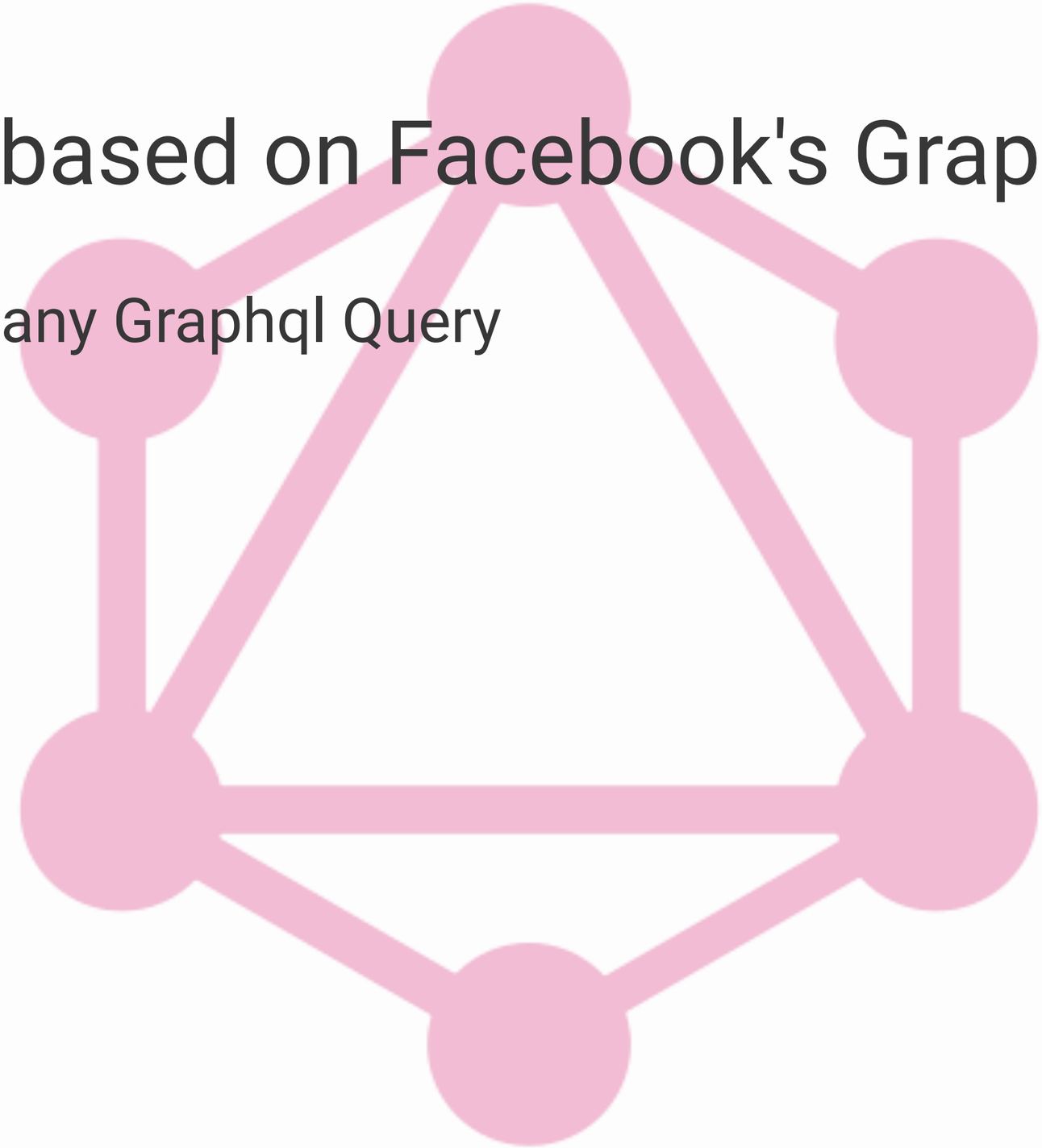
- A strongly-typed, caching GraphQL client for Android
- Rich support of Types and Type Mappings
- Builders to create queries
- Query Validation at compilation
- Built by Android Devs for Android Devs

Created based on Facebook's GraphQl Spec

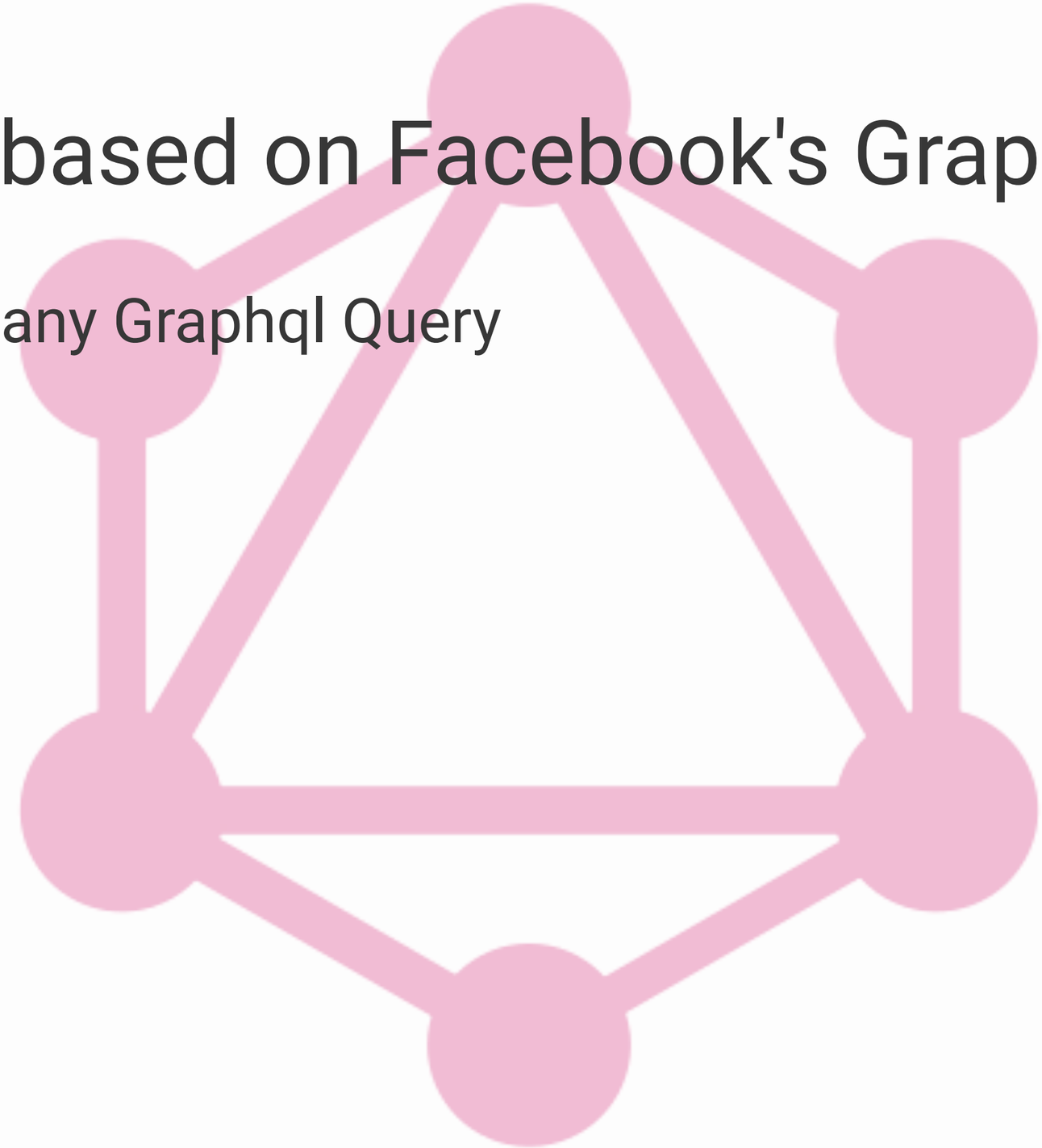


# Created based on Facebook's GraphQl Spec

- Works with any Graphql Query

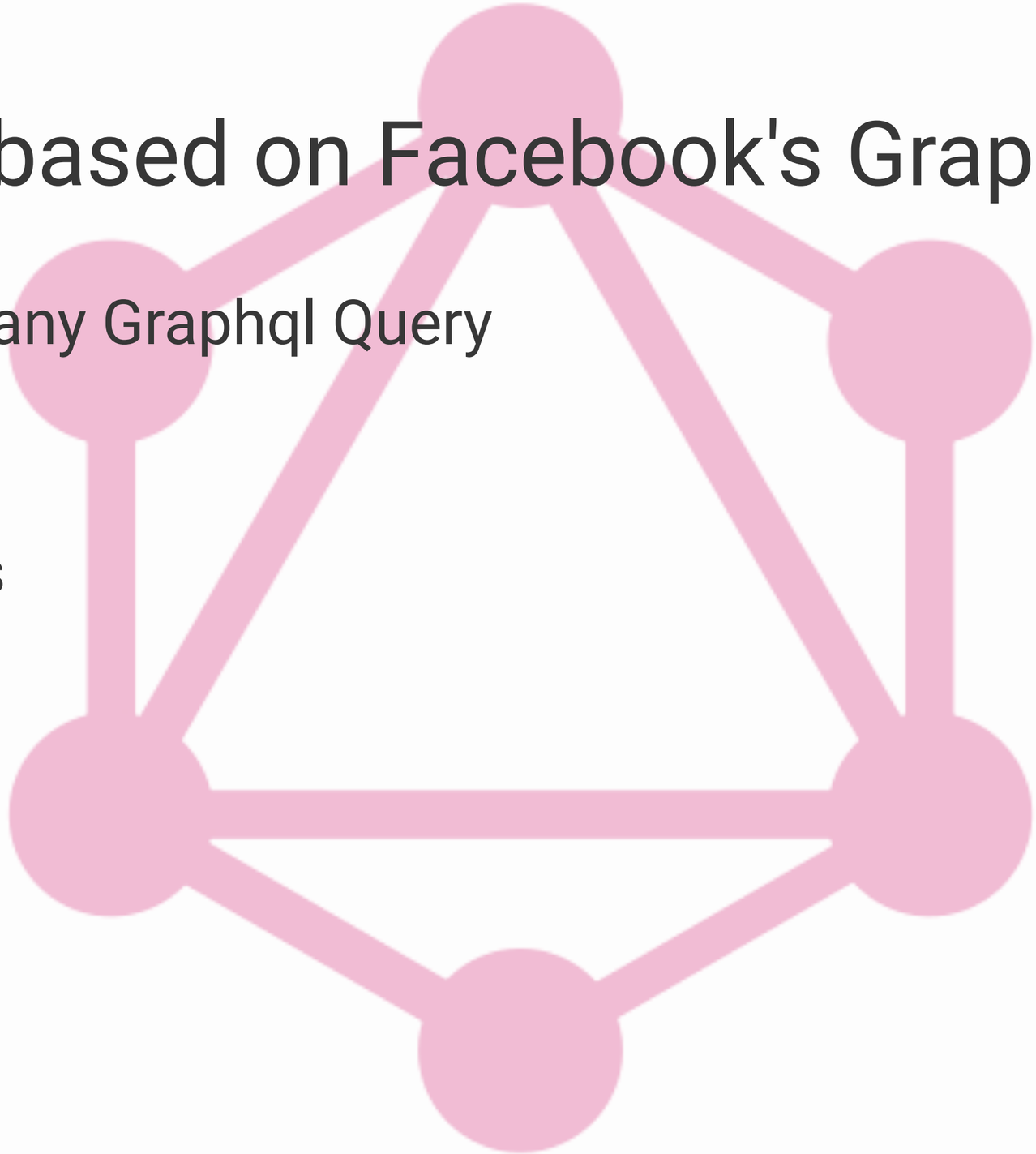


# Created based on Facebook's GraphQl Spec



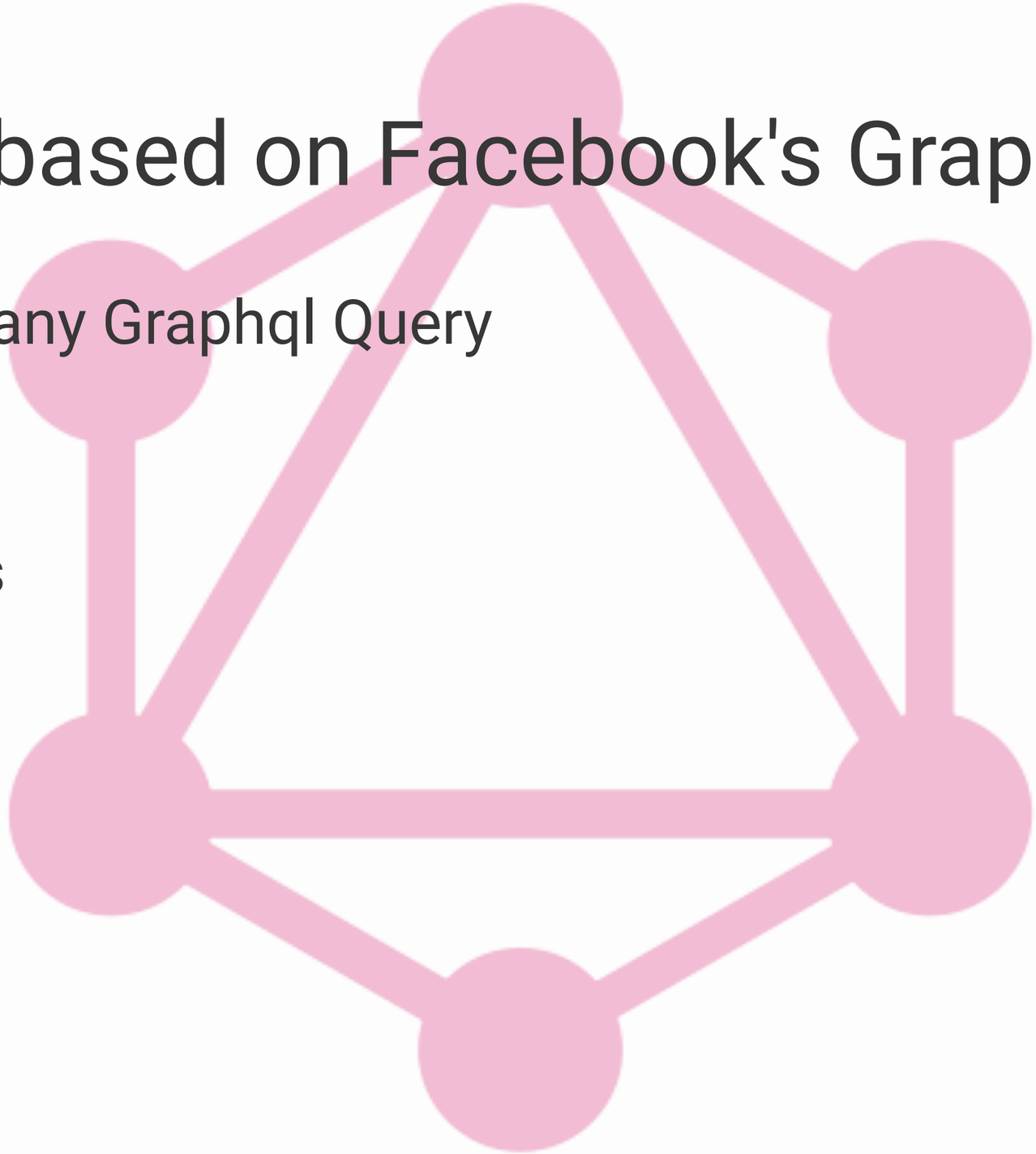
- Works with any Graphql Query
- Fragments

# Created based on Facebook's GraphQl Spec



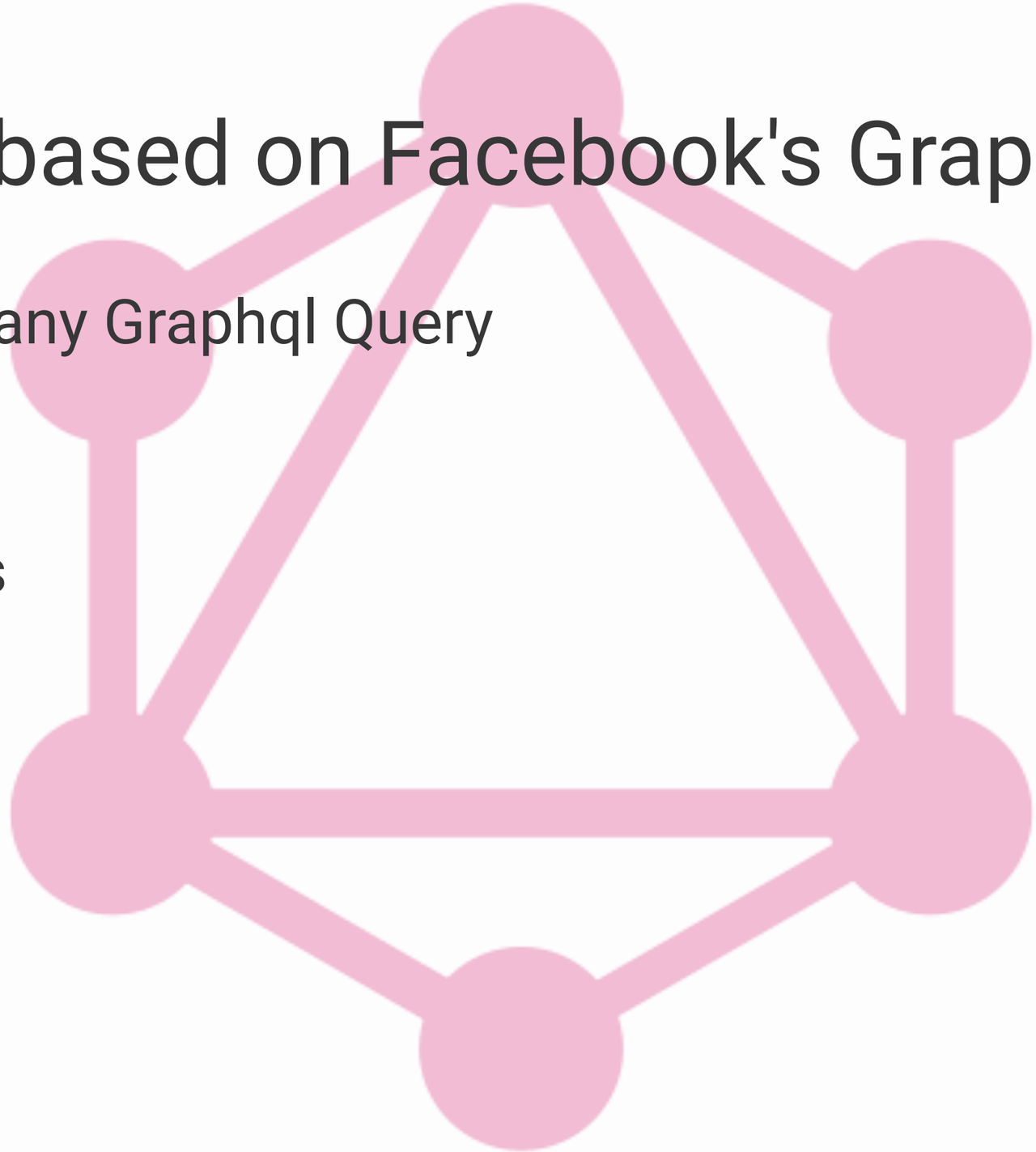
- Works with any Graphql Query
- Fragments
- Union Types

# Created based on Facebook's GraphQl Spec



- Works with any Graphql Query
- Fragments
- Union Types
- Nullability

# Created based on Facebook's GraphQl Spec



- Works with any Graphql Query
- Fragments
- Union Types
- Nullability
- Deprecation

# Apollo Reduces setup to work with a backend

| **Data Modeling** | **Networking** | **Storage** | **Transform** |
|---|---|---|---|
| Github Explorer | OKhttp | Apollo | RxJava |
| Apollo | Apollo | Apollo | Apollo |

You Ain't Gonna Need It

***~~Retrofit~~ | ~~Immutables~~| ~~Gson~~ | ~~Guava~~ | ~~SqlDelight/Brite~~ | ~~Store~~ | ~~Curl~~ | ~~JsonViewer.hu~~***

# Apollo-Android has **2 main parts**

# Apollo-Android has **2 main parts**

- **Gradle Plugin** Apollo Code Gen Plugin To generate code.

# Apollo-Android has **2 main parts**

- **Gradle Plugin** Apollo Code Gen Plugin To generate code.
- **Runtime** Apollo Client For executing operations

# Using Apollo

## like a boss

# Add Apollo dependencies

```
build.gradle:
dependencies {
  classpath 'com.apollographql.apollo:gradle-plugin:0.4.1'
}

app/build.gradle:
apply plugin: 'com.apollographql.android'
.....
//optional RxSupport
compile 'com.apollographql.apollo:apollo-rx-support:0.4.1'
```

# Create a standard GraphQL query

## Queries have params and define shape of response

```
organization(login:”nyTimes”){
    repositories(first:6 {
            Name
    }
}
```

# No CURL Needed

Most Graphql Servers have a GUI

https://developer.github.com/v4/explorer/

# Explorer is for exploring schema and building queries

# Explorer is for exploring schema and building queries

- Shape of Response

# Explorer is for exploring schema and building queries

- Shape of Response
- Nullability Rules

# Explorer is for exploring schema and building queries

- Shape of Response
- Nullability Rules
- Enum values

# Explorer is for exploring schema and building queries

- Shape of Response
- Nullability Rules
- Enum values
- Types

## Add Schema & RepoQuery.graphql to project & compile

Image needed

# Apollo writes code so you don't have to

```
private fun CodeGenerationIR.writeJavaFiles(context: CodeGenerationContext, outputDir: File,
    outputPackageName: String?) {
  fragments.forEach {
    val typeSpec = it.toTypeSpec(context.copy())
    JavaFile.builder(context.fragmentsPackage, typeSpec).build().writeTo(outputDir)
  }

  typesUsed.supportedTypeDeclarations().forEach {
    val typeSpec = it.toTypeSpec(context.copy())
    JavaFile.builder(context.typesPackage, typeSpec).build().writeTo(outputDir)
  }

  if (context.customTypeMap.isNotEmpty()) {
    val typeSpec = CustomEnumTypeSpecBuilder(context.copy()).build()
    JavaFile.builder(context.typesPackage, typeSpec).build().writeTo(outputDir)
  }

  operations.map { OperationTypeSpecBuilder(it, fragments, context.useSemanticNaming) }
      .forEach {
        val packageName = outputPackageName ?: it.operation.filePath.formatPackageName()
        val typeSpec = it.toTypeSpec(context.copy())
        JavaFile.builder(packageName, typeSpec).build().writeTo(outputDir)
      }
}
```

# Query.Builder - For Creating your request instance

```
///api
val query = RepoQuery.builder.name("friendlyrobotnyc").build()

//Generated Code

public static final class Builder {
    private @Nonnull String name;

    Builder() {
    }

    public Builder name(@Nonnull String name) {
      this.name = name;
      return this;
    }

    public RepoQuery build() {
      if (name == null) throw new IllegalStateException("name can't be null");
      return new RepoQuery(name);
    }
  }
```

# Notice how our request param `name` is validated

```
///api
val query = RepoQuery.builder.name("friendlyrobotnyc").build()

//Generated Code

public static final class Builder {
    private @Nonnull String name;

    Builder() {
    }

    public Builder name(@Nonnull String name) {
      this.name = name;
      return this;
    }

    public RepoQuery build() {
      if (name == null) throw new IllegalStateException("name can't be null");
      return new RepoQuery(name);
    }
  }
```

# MyQuery.Data aka Effective Java Value Object

```
public static class Repositories {
   final @Nonnull String __typename;
   final int totalCount;
   final @Nullable List<Edge> edges;
   private volatile String $toString;
   private volatile int $hashCode;
   private volatile boolean $hashCodeMemoized;

   public @Nonnull String __typename() { return this.__typename; }

   //Identifies the total count of items in the connection.
   public int totalCount() {return this.totalCount;}

   //A list of edges.
   public @Nullable List<Edge> edges() {return this.edges;}

   @Override
   public String toString() {...}

   @Override
   public boolean equals(Object o) {  ... }

   @Override
   public int hashCode() {...}
```

# Query.Mapper - Reflection Free Parser

```
public static final class Mapper implements ResponseFieldMapper<Repositories> {
     final Edge.Mapper edgeFieldMapper = new Edge.Mapper();

     @Override
     public Repositories map(ResponseReader reader) {
       final String __typename = reader.readString($responseFields[0]);
       final int totalCount = reader.readInt($responseFields[1]);
       final List<Edge> edges = reader.readList($responseFields[2], new ResponseReader.ListReader<Edge>() {
         @Override
         public Edge read(ResponseReader.ListItemReader reader) {
           return reader.readObject(new ResponseReader.ObjectReader<Edge>() {
             @Override
             public Edge read(ResponseReader reader) {
               return edgeFieldMapper.map(reader);
             }
           });
         }
       });
       return new Repositories(__typename, totalCount, edges);
     }}
```

Can parse 20mb response without OOM

# We have our Data Models

## How do we get a response from Github?

# Building an Apollo Client

```
ApolloClient.builder()
                .serverUrl("https://api.github.com/graphql)
                .okHttpClient(okhttp)
                .build();
```

# Apollo’s api is very similar to Okhttp

## Stateless Apollo Client that can create an `ApolloCall`

```
query = RepoQuery.builder().name("friendlyrobotnyc").build()

ApolloQueryCall githubCall = apolloClient.query(query);

githubCall.enqueue(new ApolloCall.Callback<>() {
    @Override
    public void onResponse(@Nonnull Response<> response) {

    }

    @Override
    public void onFailure(@Nonnull ApolloException e) {

    }
});
```

# Apollo’s api is very similar to Okhttp

## Stateless Apollo Client that can create an `ApolloCall`

```
query = RepoQuery.builder().name("friendlyrobotnyc").build()

ApolloQueryCall githubCall = apolloClient.query(query);

githubCall.enqueue(new ApolloCall.Callback<>() {
    @Override
    public void onResponse(@Nonnull Response<> response) {

    }

    @Override
    public void onFailure(@Nonnull ApolloException e) {

    }
});
```

# Apollo’s api is very similar to Okhttp

## Stateless Apollo Client that can create an `ApolloCall`

```
query = RepoQuery.builder().name("friendlyrobotnyc").build()

ApolloQueryCall githubCall = apolloClient.query(query);

githubCall.enqueue(new ApolloCall.Callback<>() {
    @Override
    public void onResponse(@Nonnull Response<> response) {

    }

    @Override
    public void onFailure(@Nonnull ApolloException e) {

    }
});
```

# Storage with Apollo is done through Caches

# Storage with Apollo is done through Caches

- HTTP

# Storage with Apollo is done through Caches

- HTTP
- Normalized

# Http Caching

need code example of prefetch

# Http Caching

- Similar to OKHTTP Cache (LRU)

need code example of prefetch

# Http Caching

- Similar to OKHTTP Cache (LRU)
- Streams response to cache same time as parsing

# need code example of prefetch

# Http Caching

- Similar to OKHTTP Cache (LRU)
- Streams response to cache same time as parsing
- Can Set Max Cache Size

need code example of prefetch

# Http Caching

- Similar to OKHTTP Cache (LRU)
- Streams response to cache same time as parsing
- Can Set Max Cache Size
- Useful for background updating to prefill cache

# need code example of prefetch

HTTP Caching is about as well as you can do in REST

# Apollo Introduces a Normalized Cache - Apollo Store

# Apollo Store

# Apollo Store

- Caching is done Post Parsing

# Apollo Store

- Caching is done Post Parsing
- Each field is Cached Individually

# Apollo Store

- Caching is done Post Parsing
- Each field is Cached Individually
- Allows multiple queries to share same cached values

# Apollo Store

- Caching is done Post Parsing
- Each field is Cached Individually
- Allows multiple queries to share same cached values
- Great for things like Master/Detail

# Apollo Store

- Caching is done Post Parsing
- Each field is Cached Individually
- Allows multiple queries to share same cached values
- Great for things like Master/Detail
- Apollo ships with both an in memory and a disk implementation of an Apollo Store

# Apollo Store

-Each Object in Response will have its own record with ID
-All Scalars will be merged together as fields
-When we are reading from Apollo, it will seamlessly read from Apollo Store or network

# Settings Up Bi-Level Caching with Apollo Store

```
//Create DB
ApolloSqlHelper apolloSqlHelper = ApolloSqlHelper.create(context, "db_name");
//Create NormalizedCacheFactory
NormalizedCacheFactory normalizedCacheFactory = new LruNormalizedCacheFactory(EvictionPolicy.NO_EVICTION)
                                                    .chain(new SqlNormalizedCacheFactory(apolloSqlHelper));
//Create the cache key resolver
CacheKeyResolver<Map<String, Object>> resolver = {
            String id = (String) objectSource.get("id");
            if (id == null || id.isEmpty()) {
              return CacheKey.NO_KEY;
            }
            return CacheKey.from(id);
        }
//Build the Apollo Client
ApolloClient apolloClient = ApolloClient.builder()
                                    .serverUrl("/")
                                    .normalizedCache(cacheFactory, resolver)
                                    .okHttpClient(okHttpClient)
                                    .build();
```

# Don't like our Cache? BYO Cache

```
public abstract class NormalizedCache {

@Nullable public abstract Record loadRecord(@Nonnull String key, @Nonnull CacheHeaders cacheHeaders)

@Nonnull public Collection<Record> loadRecords(@Nonnull Collection<String> keys, @Nonnull CacheHeaders cacheHeaders)

@Nonnull public abstract Set<String> merge(@Nonnull Record record, @Nonnull CacheHeaders cacheHeaders)

public abstract void clearAll()

public abstract boolean remove(@Nonnull CacheKey cacheKey)
```

# Apollo Is Reactive

QueryWatcher will emit new response when there are changes to the normalized cache records this query depends on or when mutation call occurs

# Bonus: Includes RxJava Bindings

```
RxApollo.from(apolloClient.query(RepoQuery.builder().name("friendlyrobotnyc").build()))
        .map(dataResponse -> dataResponse
        .data()
        .organization()
        .repositories())
        .subscribe(view::showRepositories, view::showError)
```

# RxApollo response can be transformed into LiveData

# Version 1.0 ships today

380 commits

1000s of tests

18 contributors including devs from Shopify, Airbnb, NY Times