Building Modern Applications with GraphQL

@michael_staib

ahaslides.com/RHZFT

2012

RFC -- SuperGraph

Edit fields

Fields

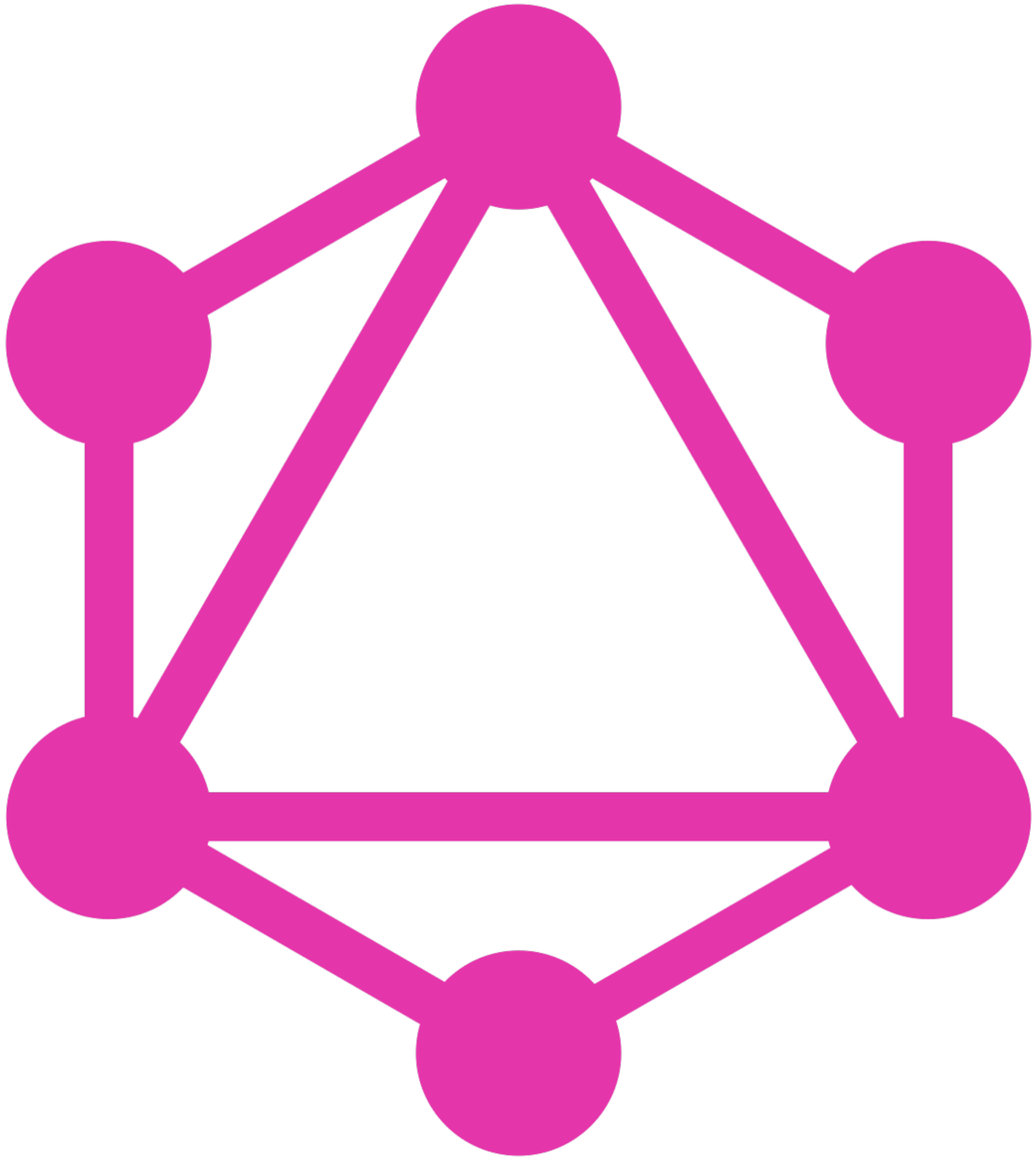
Chief Clown: ~~schrockn~~ · Created Feb 29, 2012 · Last Updated Mar 5, 2012 1:37 PM

Summary

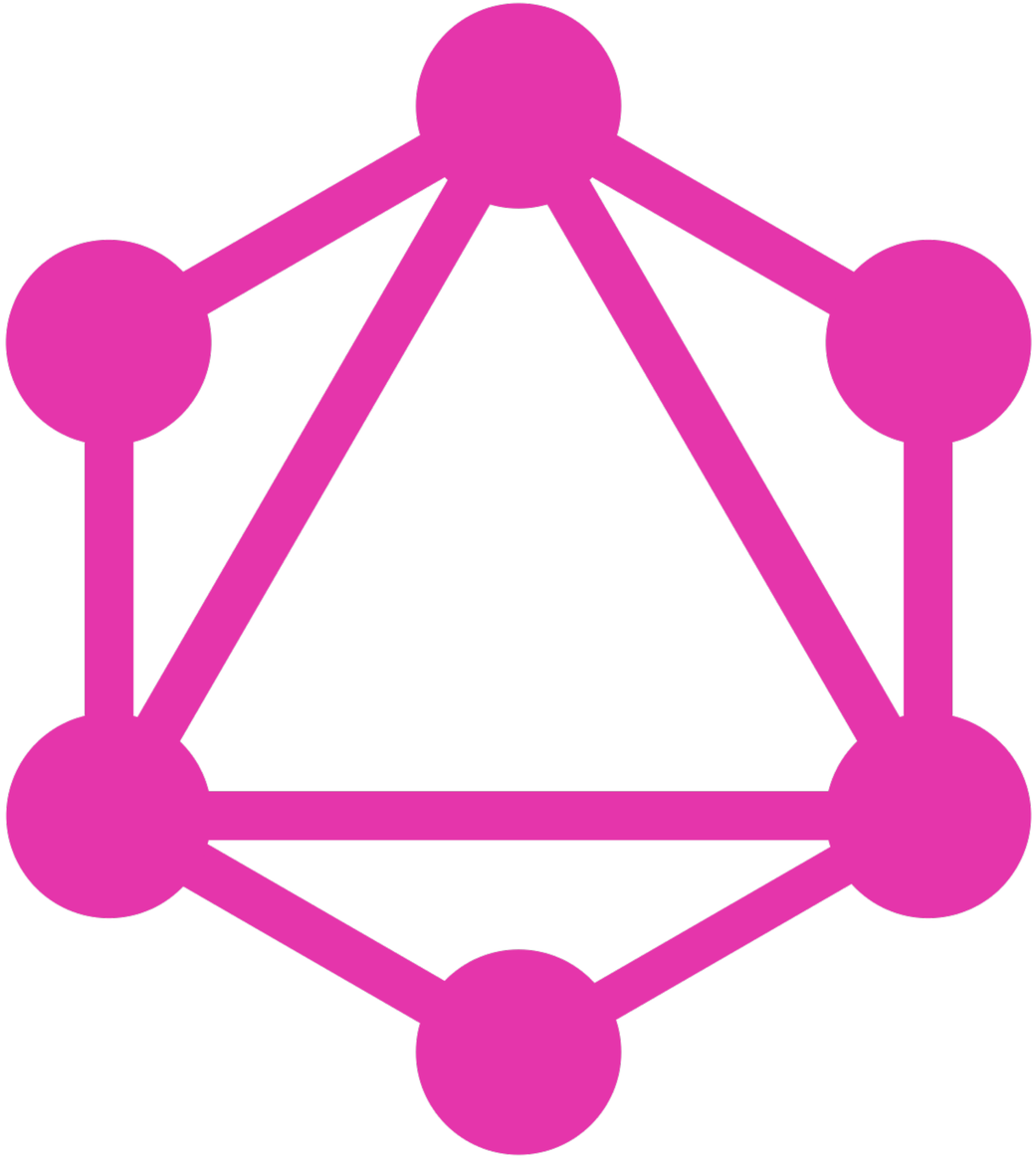
Here's a more concrete proposal of what we talked about last week.

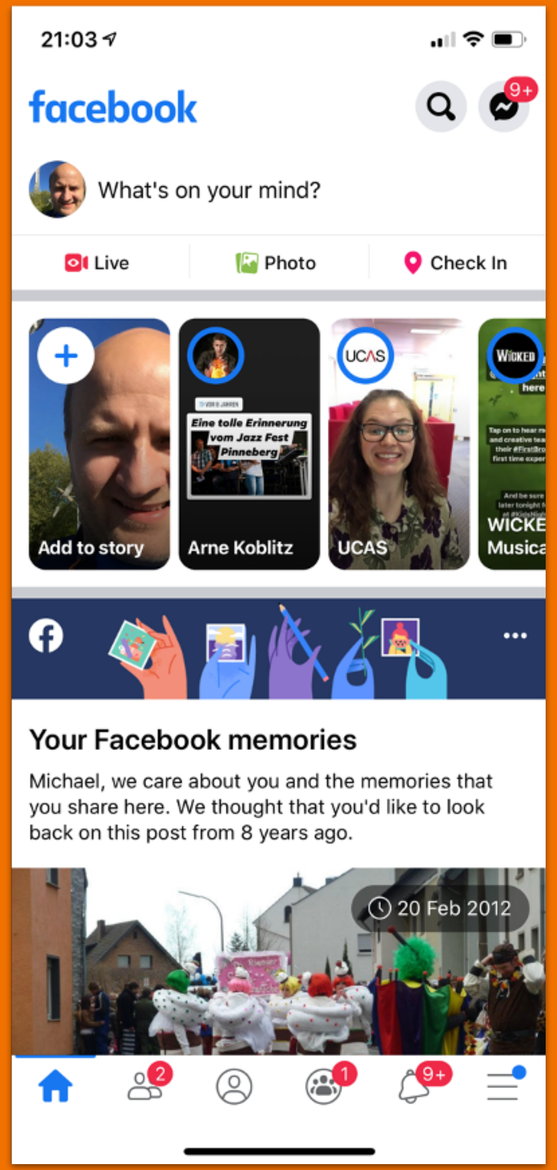
The idea here is a new engine for a souped up version of the graph API (SuperGraph, for lack of a better term). Rather than viewing graph as a wrapper of FQL, this instead maps directly to Ents and Ent queries. This allows for a lot more expressivity and power.

What is GraphQL?



Why GraphQL?

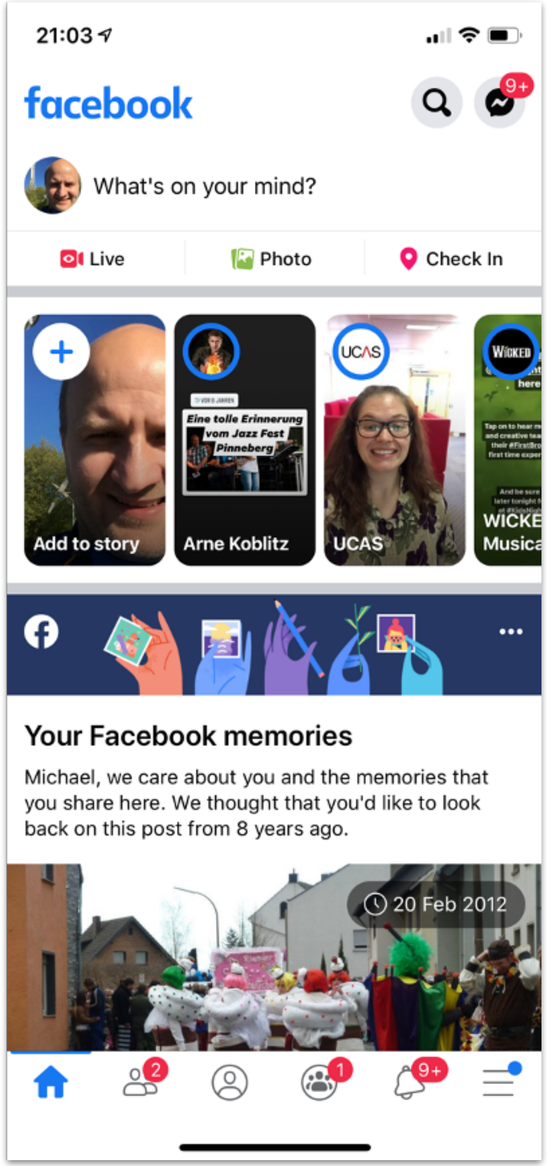



21:03
facebook
9+
What's on your mind?
Live
Photo
Check In
Add to story
Eine tolle Erinnerung vom Jazz Fest Pinneberg
Arne Koblitz
UCAS
Your Facebook memories
Michael, we care about you and the memories that you share here. We thought that you'd like to look back on this post from 8 years ago.
20 Feb 2012


21:04
Stefania Staib updated her cover photo.
8 hrs
Like
Comment
Share
Ariane Pieper and 16 others
Ariane Pieper
the little 3 stooges
6 h Like Reply
Kathleen Renken
You have adorable munchkins!
2 h Like Reply
Write a comment...

Main Issues

- Slow app start
- High data usage
- Drained batteries



GET http://api.facebook.com/news?userId=1&page=1

GET http://api.facebook.com/news?userId=1&page=1

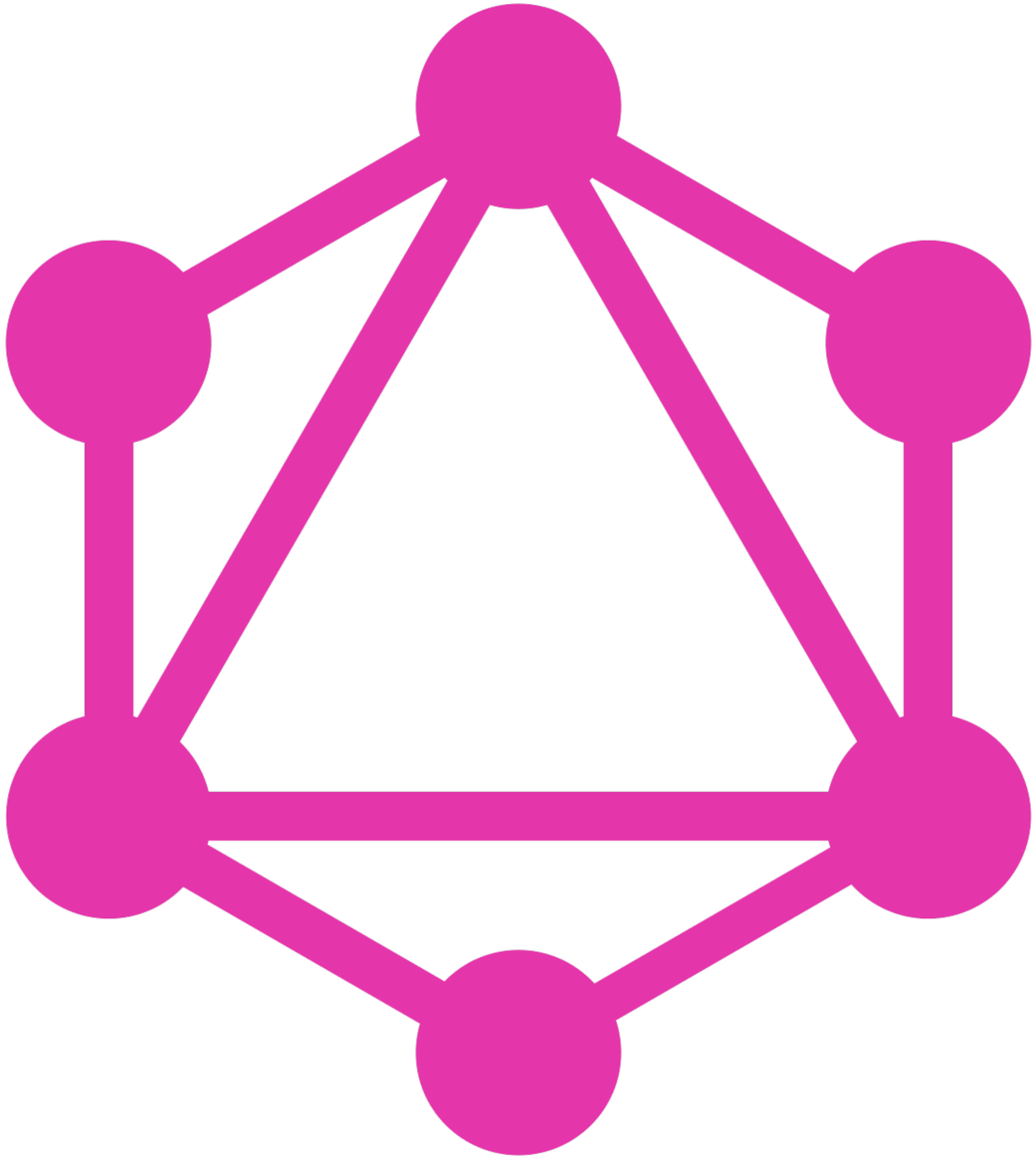
```
[
    {
      "id": 1,
      "title": "abcdef",
      "text": " ... ",
      "authorId": 5
       ...
    },
     ...
]
```

GET http://api.facebook.com/comments?newsId=1

GET http://api.facebook.com/comments?newsId=1

```
[
    {
      "id": 1,
      "text": "...",
      "newsId": 1,
      "authorId": 5
       ...
    },
     ...
]
```

200	preflight	Preflight	0 B	1.07 s
200	fetch	index.html:121	926 B	511 ms
200	preflight	Preflight	0 B	540 ms
200	preflight	Preflight	0 B	537 ms
200	preflight	Preflight	0 B	555 ms
200	preflight	Preflight	0 B	549 ms
200	preflight	Preflight	0 B	543 ms
200	fetch	index.html:121	1.3 kB	393 ms
200	fetch	index.html:121	1.3 kB	340 ms
200	fetch	index.html:121	1.3 kB	367 ms
200	fetch	index.html:121	1.4 kB	394 ms
200	fetch	index.html:121	1.4 kB	368 ms
200	fetch	index.html:121	860 B	402 ms
200	fetch	index.html:121	863 B	369 ms
200	preflight	Preflight	0 B	150 ms
200	fetch	index.html:121	872 B	372 ms
200	fetch	index.html:121	869 B	589 ms
200	preflight	Preflight	0 B	153 ms
200	fetch	index.html:121	866 B	383 ms
200	preflight	Preflight	0 B	150 ms
200	fetch	index.html:121	866 B	374 ms
200	fetch	index.html:121	849 B	844 ms
200	preflight	Preflight	0 B	243 ms
200	preflight	Preflight	0 B	158 ms
200	fetch	index.html:121	856 B	1.24 s
200	preflight	Preflight	0 B	148 ms



```
{
  me {
    name
  }
}
```

```
{
  me {
    name
  }
}
```

```
{
  "me": {
    "name": "Michael Staib"
  }
}
```

```
{
  me {
    name
    image {
      width
      height
      url
    }
  }
}
```

```
{
  me {
    name
    image {
      width
      height
      url
    }
  }
}
```

```
{
  "me": {
    "name": "Michael Staib",
    "image": {
      "width": 200,
      "height": 300,
      "url": "http://some/images/123.png"
    }
  }
}
```

```
{
  me {
    name
    lastSeen
    friends {
      name
      lastSeen
    }
  }
}
```

```
{
  me {
    name
    lastSeen
    friends {
      name
      lastSeen
    }
  }
}
```

```
{
  "me": {
    "name": "Michael Staib",
    "lastSeen": "2018-05-19T18:45",
    "friends": [
      {
        "name": "Rafael Staib",
        "lastSeen": "2018-05-24T12:37"
      },
      {
        "name": "Pascal Senn",
        "lastSeen": "2018-06-07T17:13"
      }
    ]
  }
}
```

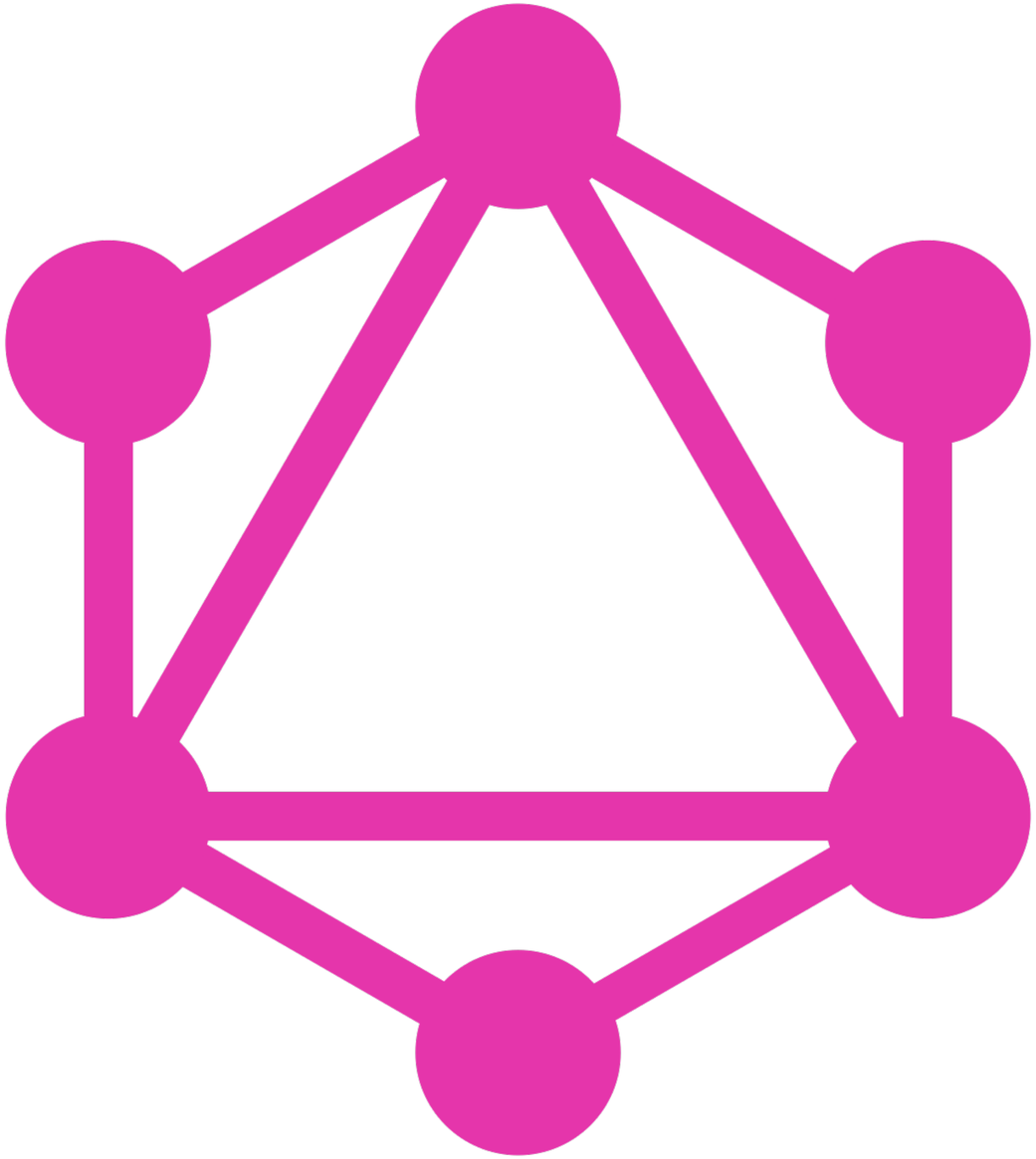
```
{
  me {
    … PersonInfo
    friends {
      … PersonInfo
    }
  }
}

fragment PersonInfo on Person {
  name
  lastSeen
}
```

```
{
  "me": {
    "name": "Michael Staib",
    "lastSeen": "2018-05-19T18:45",
    "friends": [
      {
        "name": "Rafael Staib",
        "lastSeen": "2018-05-24T12:37"
      },
      {
        "name": "Pascal Senn",
        "lastSeen": "2018-06-07T17:13"
      }
    ]
  }
}
```

GraphQL gives clients the power to ask for exactly what they need and nothing more.

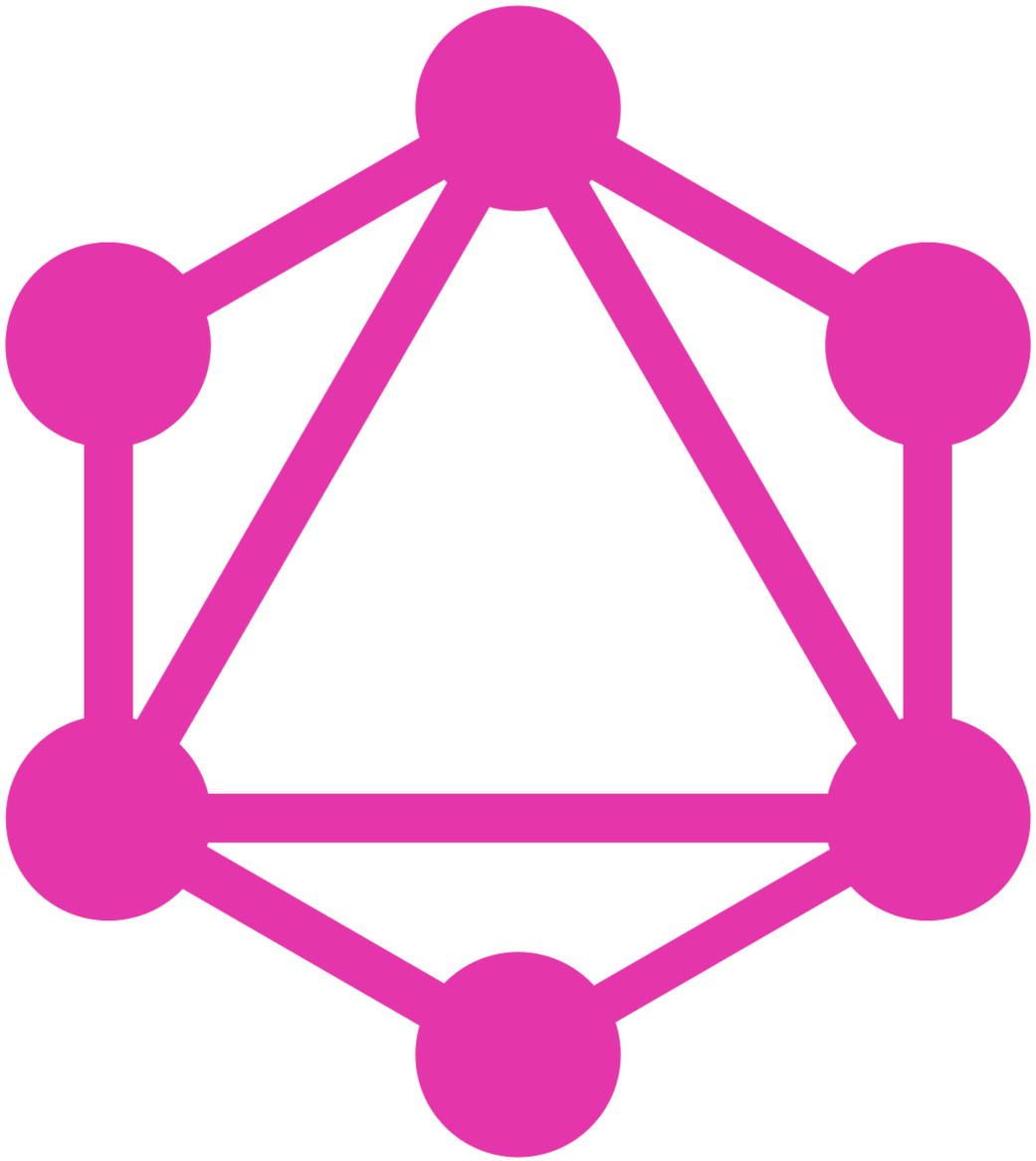
What is GraphQL?

- Query language for your API
- Runtime to fulfill your queries



What is GraphQL?

- One Endpoint
- One Request
- No over- or under-fetching
- Type System
- Predictable
- Real-Time



From zero to hero 🚀

GraphQL Operations

Operation	GraphQL	REST
Read	Query	GET
Write	Mutation	PUT, POST, PATCH, DELETE
Events	Subscription	N/A

GraphQL Operations

Operation	GraphQL	REST
Read	Query	GET
Write	Mutation	PUT, POST, PATCH, DELETE
Events	Subscription	N/A

Demo 1

Getting started with GraphQL

```
public class Query
{
    public string Hello(string name = "World")
        => $"Hello, {name}!";
}
```

```
public class Query
{
    public string Hello(string name = "World")
        ⇒ $"Hello, {name}!";
}
```

Resolver

```
public class Query
{
    public string Hello(string name = "World")
        ⇒ $"Hello, {name}!";
}
```

Resolver

```
type Query {
    hello(name: String! = “World"): String!
}
```

Annotation-Based Approach

```
public class Query
{
    public string Hello(string name = "World")
        ⇒ $"Hello, {name}!";
}
```

Resolver

```
type Query {
    hello(name: String! = “World"): String!
}
```

```
public class Query
{
    public string Hello(string name = "World")
        ⇒ $"Hello, {name}!";
}
```

Resolver

```
type Query {
    hello(name: String! = “World"): String!
}
```

Field

```
public class Query
{
    public string Hello(string name = "World")
        ⇒ $"Hello, {name}!";
}
```

Resolver

```
type Query {
    hello(name: String! = “World"): String!
}
```

Field **Argument**

```
public class Query
{
    public string Hello(string name = "World")
        ⇒ $"Hello, {name}!";
}
```

Resolver

```
type Query {
    hello(name: String! = “World"): String!
}
```

Field **Argument** **Default Value**

```
public class Query
{
    public string Hello(string name = "World")
        ⇒ $"Hello, {name}!";
}
```

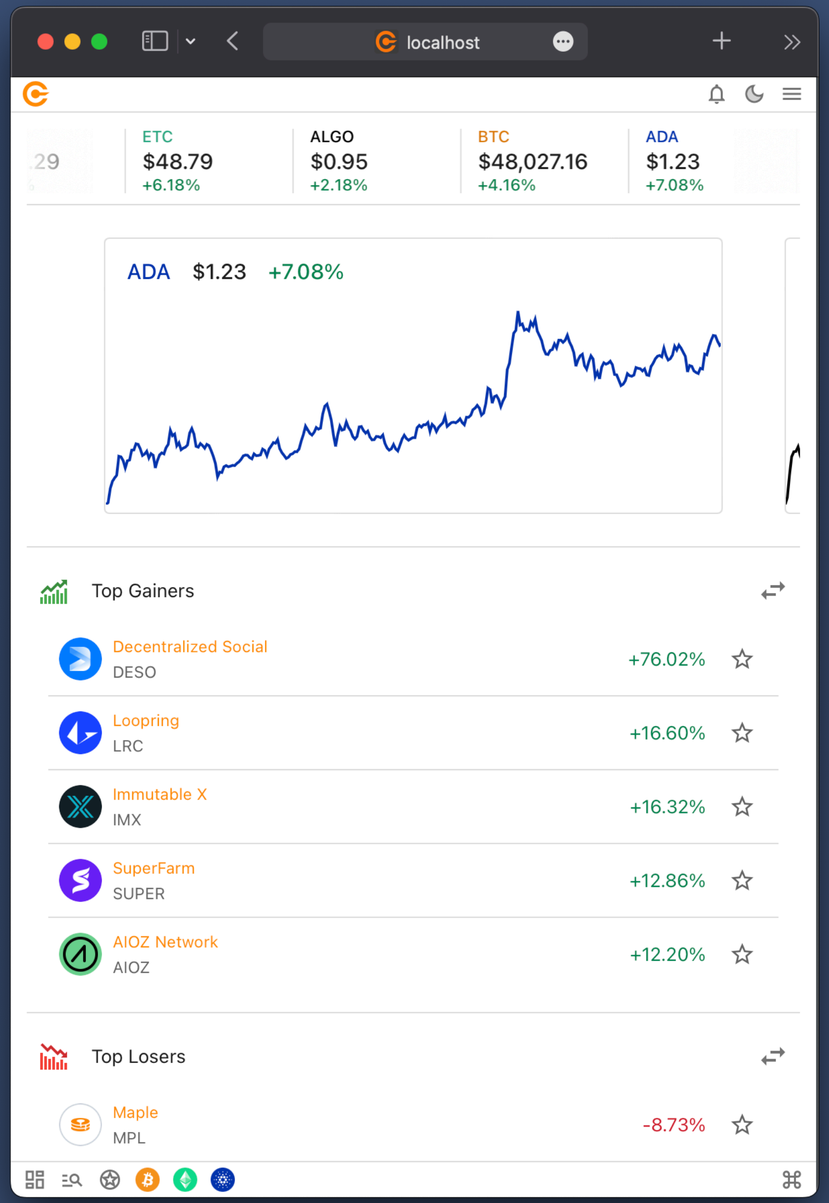
Resolver

```
type Query {
    hello(name: String! = “World"): String!
}
```

Field Argument Default Value Non-Null

GraphQL Transport

- Transport Agnostic
- **HTTP POST** most common for **Query** and **Mutation**
- **HTTP GET** often used for **Query** when using persisted queries
- **WebSockets** most common for **Subscription**
- **Server Side Events** can be used for **Subscription**
- **gRPC** used in some instances


localhost
ETC
$48.79
+6.18%
ALGO
$0.95
+2.18%
BTC
$48,027.16
+4.16%
ADA
$1.23
+7.08%
ADA $1.23 +7.08%
Top Gainers
Decentralized Social
DESO
+76.02%
Loopring
LRC
+16.60%
Immutable X
IMX
+16.32%
SuperFarm
SUPER
+12.86%
AIOZ Network
AIOZ
+12.20%
Top Losers
Maple
MPL
-8.73%

Goals

- Faster Iteration
- Strong Contracts
- Static Typing
- Efficient Data Fetching
- Empower the **Consumer** of our **Backend**
- Reactive

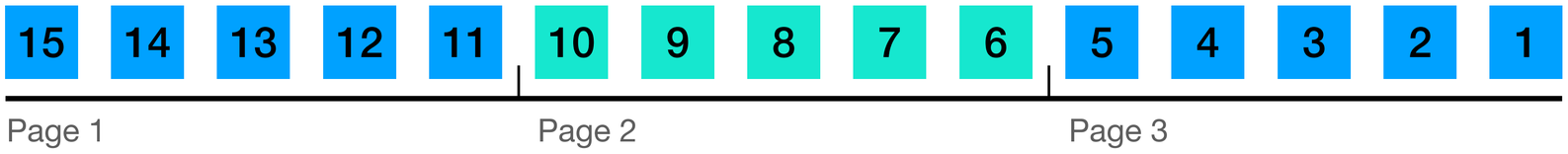
Demo 2

Fetching data with GraphQL

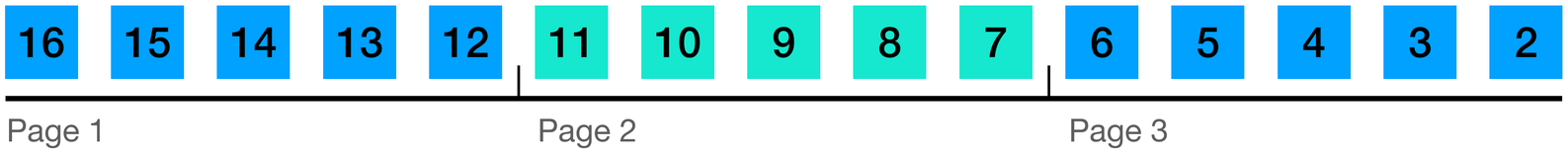
Pagination

Offset based pagination eg: Skip: 5, Take: 5

Offset based pagination eg: Skip: 5, Take: 5

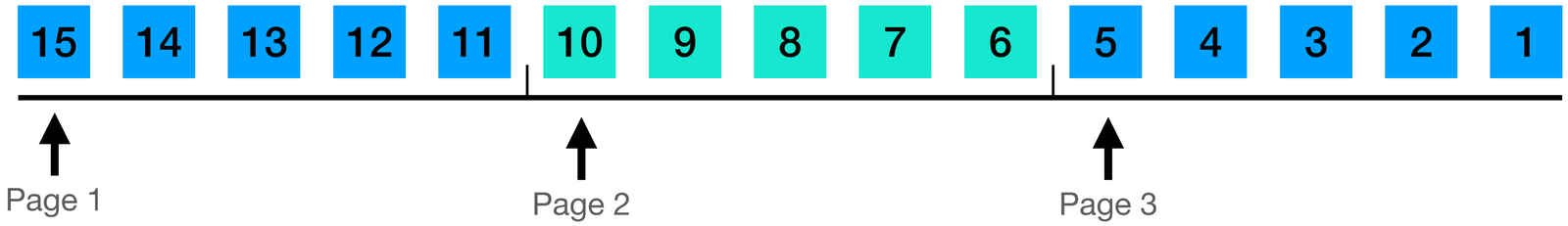


Offset based pagination eg: Skip: 5, Take: 5

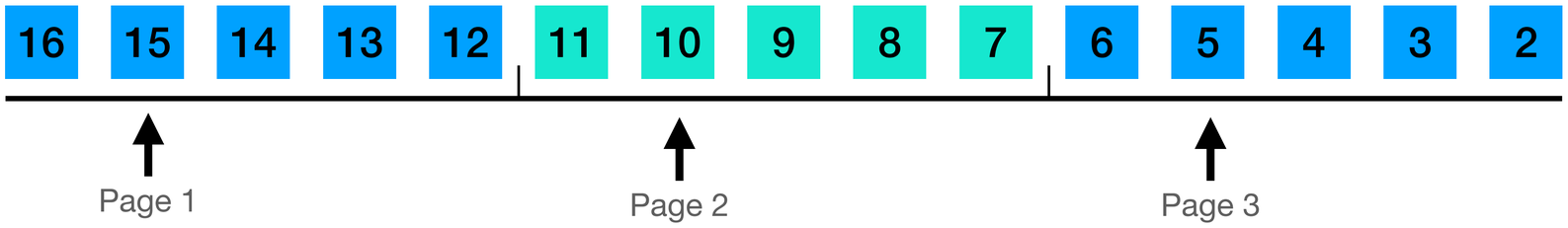


Cursor based pagination eg: after: id=5, Take: 5

Cursor based pagination eg: after: id=5, Take: 5



Cursor based pagination eg: after: id=5, Take: 5



Demo 2

Implementing cursor pagination with GraphQL

Field Middleware

```
public class Query
{
    [UsePaging]
    public IQueryable<Asset> GetAssets(AssetContext context)
        ⇒ context.Assets;
}
```

```
public class Query
{
    [UsePaging]
    public IQueryable<Asset> GetAssets(AssetContext context)
        ⇒ context.Assets;
}
```

Resolver

```
public class Query
{
    [UsePaging]
    public IQueryable<Asset> GetAssets(AssetContext context)
        ⇒ context.Assets;
}
```

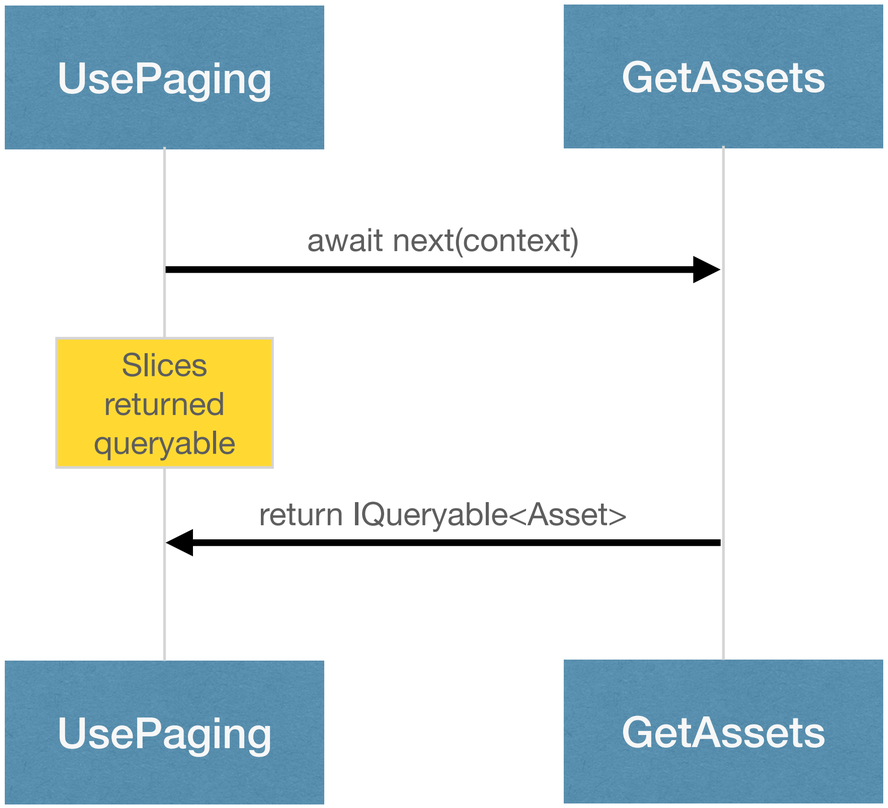
Middleware

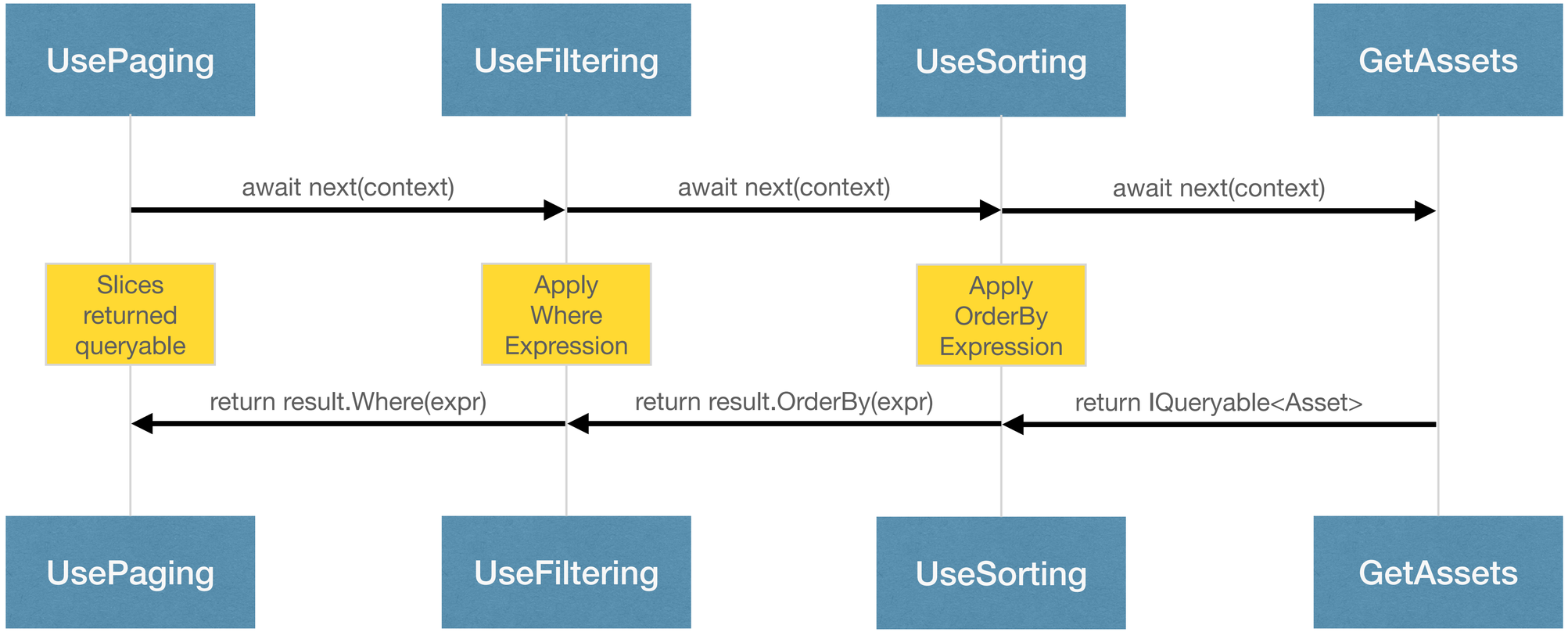
Resolver

```
public class Query
{
    [UsePaging]
    [UseProjection]
    [UseFiltering]
    [UseSorting]
    public IQueryable<Asset> GetAssets(AssetContext context)
        => context.Assets;
}
```

Middleware

Resolver


UsePaging
GetAssets
await next(context)
Slices returned queryable
return IQueryable<Asset>
UsePaging
GetAssets


UsePaging
UseFiltering
UseSorting
GetAssets
await next(context)
await next(context)
await next(context)
Slices returned queryable
Apply Where Expression
Apply OrderBy Expression
return result.Where(expr)
return result.OrderBy(expr)
return IQueryable<Asset>
UsePaging
UseFiltering
UseSorting
GetAssets

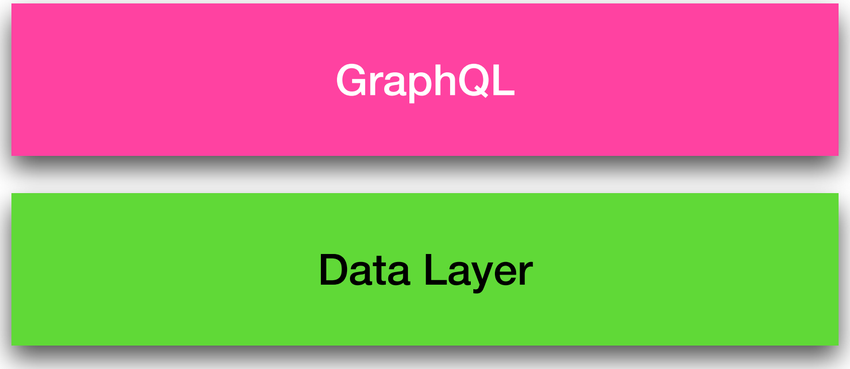
```
query {
  students(where: { OR: [{ lastName: "Bar" }, { lastName: "Baz" }] }) {
    firstMidName
    lastName
    enrollments {
      course {
        title
      }
    }
  }
}
```

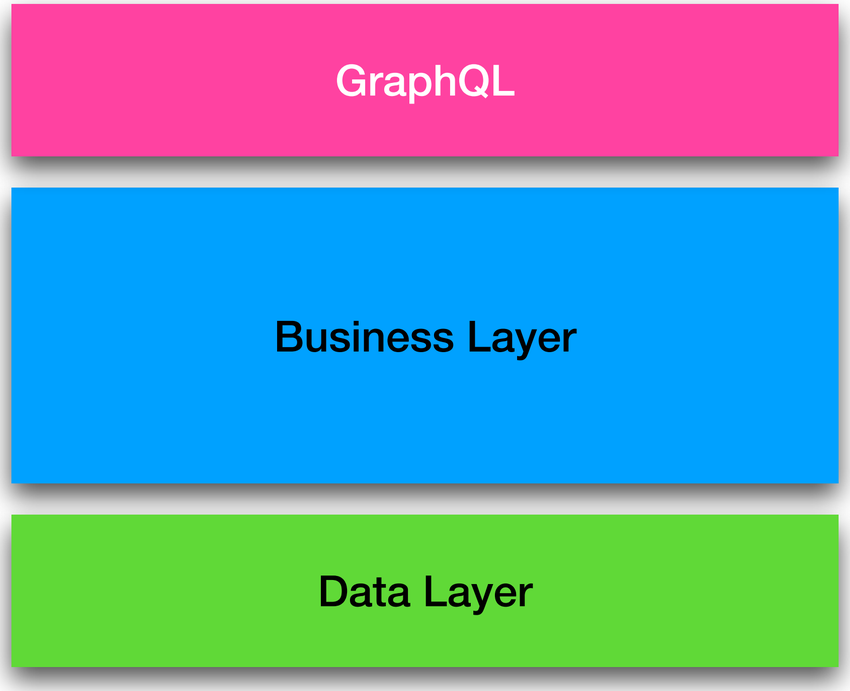
```
SELECT "s"."FirstMidName", "s"."LastName", "s"."Id", "t"."Title",
       "t"."EnrollmentId", "t"."CourseId"
    FROM "Students" AS "s"
    LEFT JOIN (
        SELECT "c"."Title", "e"."EnrollmentId", "c"."CourseId", "e"."StudentId"
        FROM "Enrollments" AS "e"
        INNER JOIN "Courses" AS "c" ON "e"."CourseId" = "c"."CourseId"
    ) AS "t" ON "s"."Id" = "t"."StudentId"
    WHERE ("s"."LastName" = 'Bar') OR ("s"."LastName" = 'Baz')
    ORDER BY "s"."Id", "t"."EnrollmentId", "t"."CourseId"
```

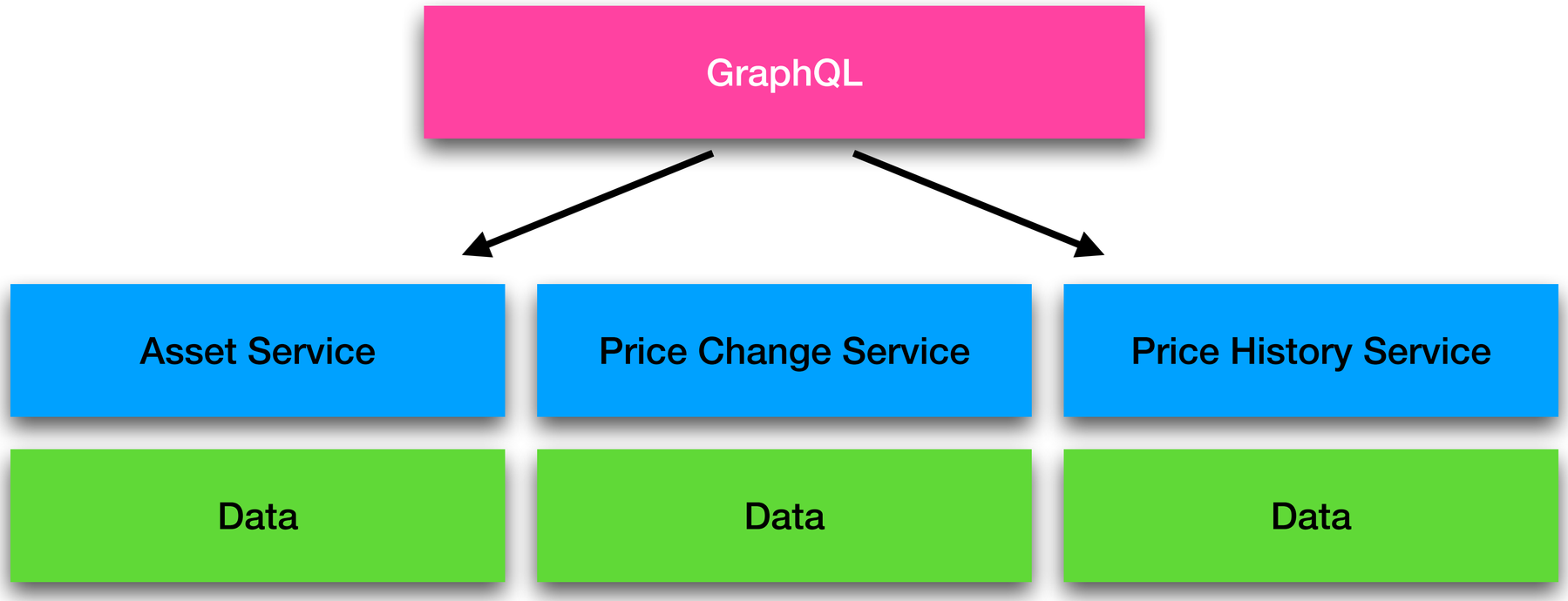
Demo 2

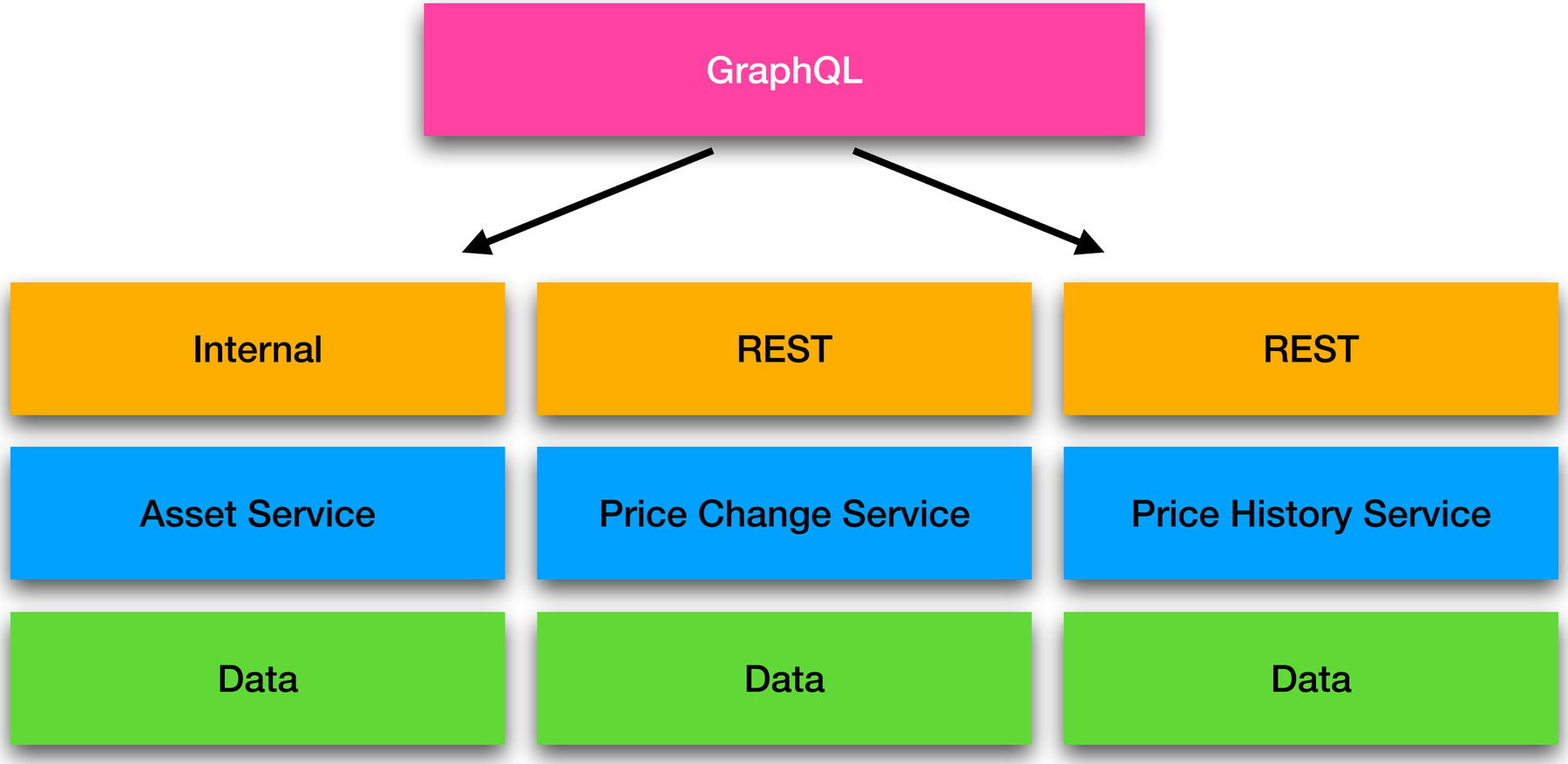
Applying Hot Chocolate Data Middleware

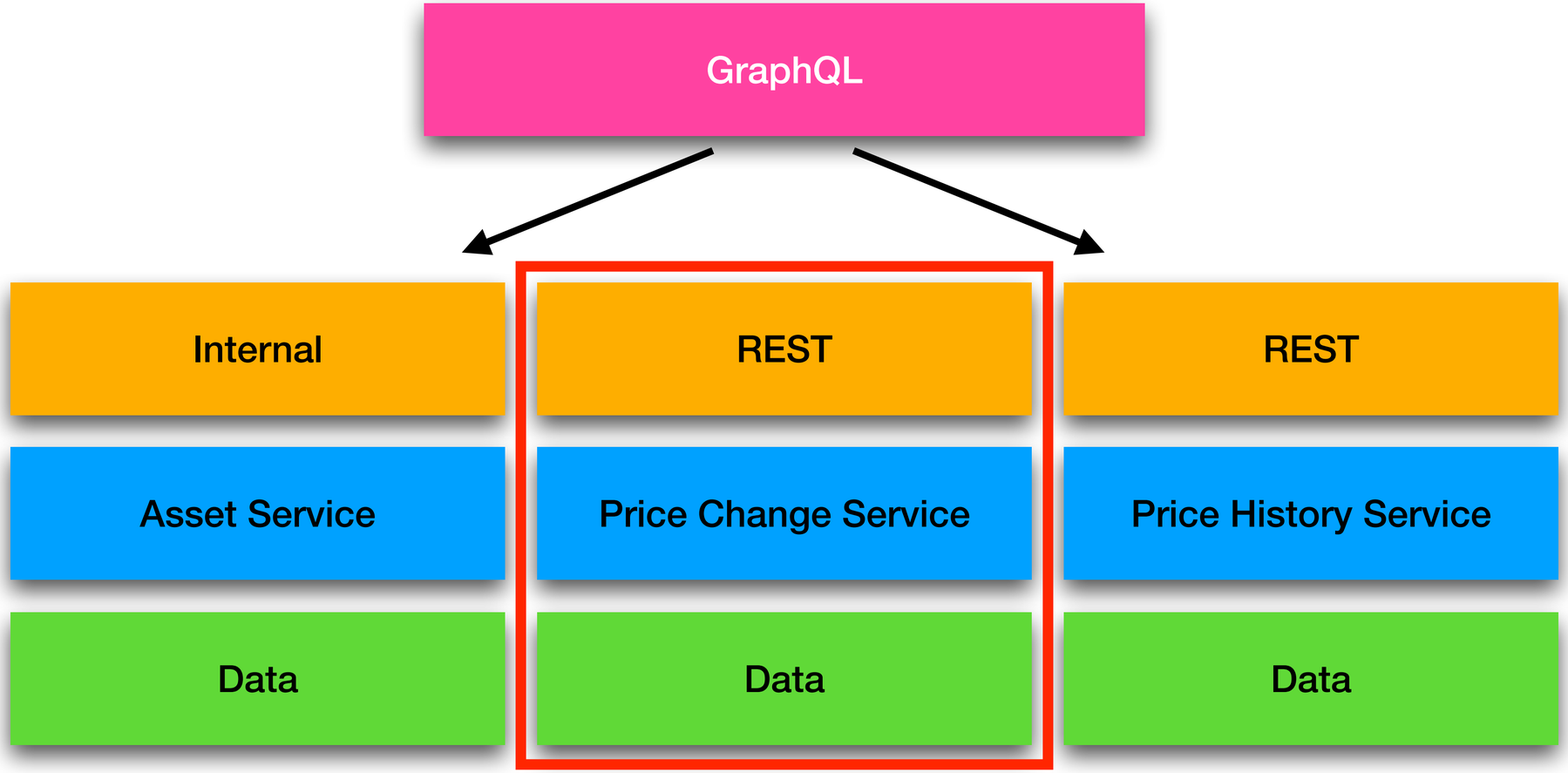
REST Integration

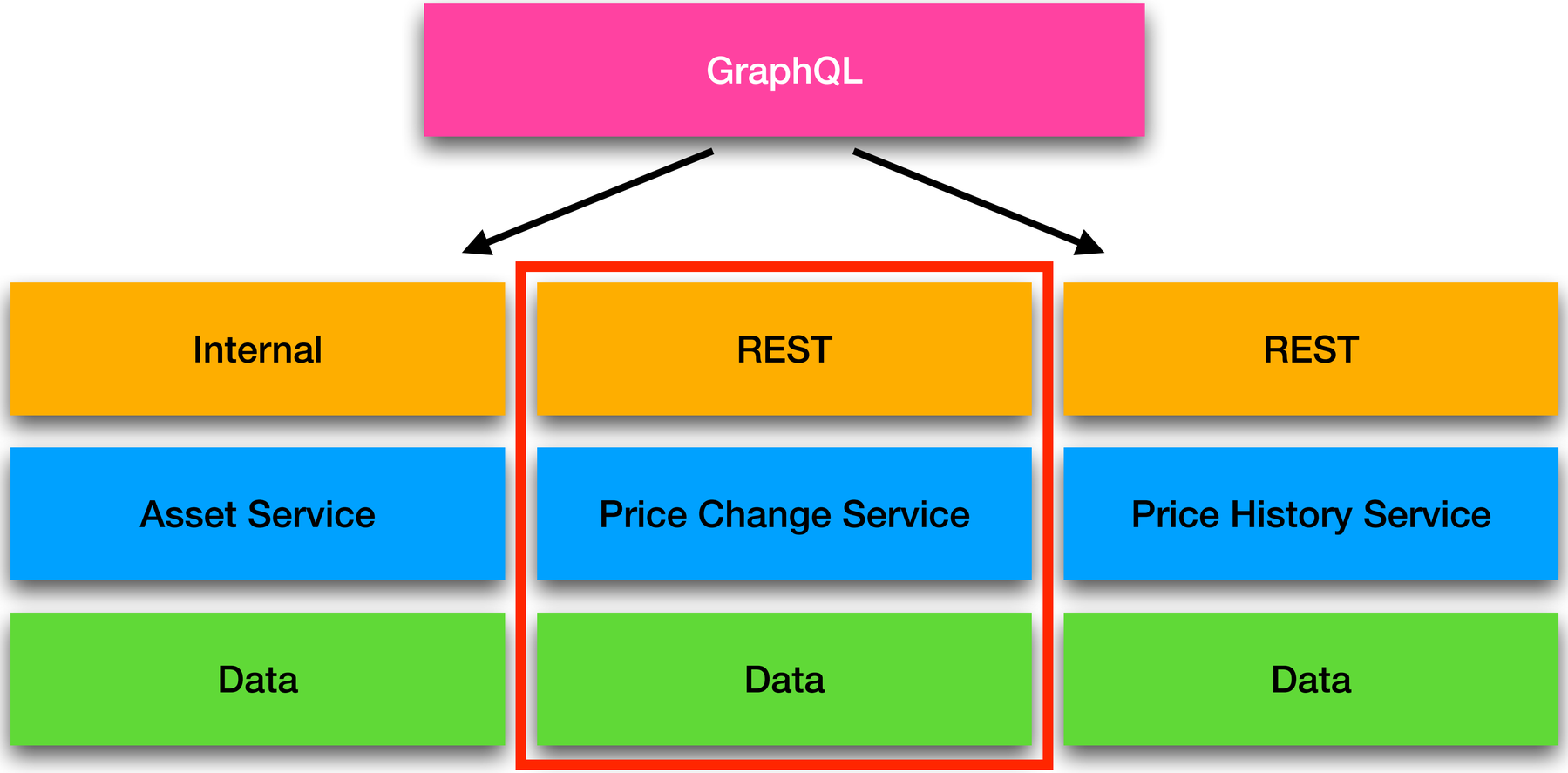

GraphQL
Data Layer


GraphQL
Business Layer
Data Layer


GraphQL
Asset Service
Price Change Service
Price History Service
Data
Data
Data


GraphQL
Internal
REST
REST
Asset Service
Price Change Service
Price History Service
Data
Data
Data

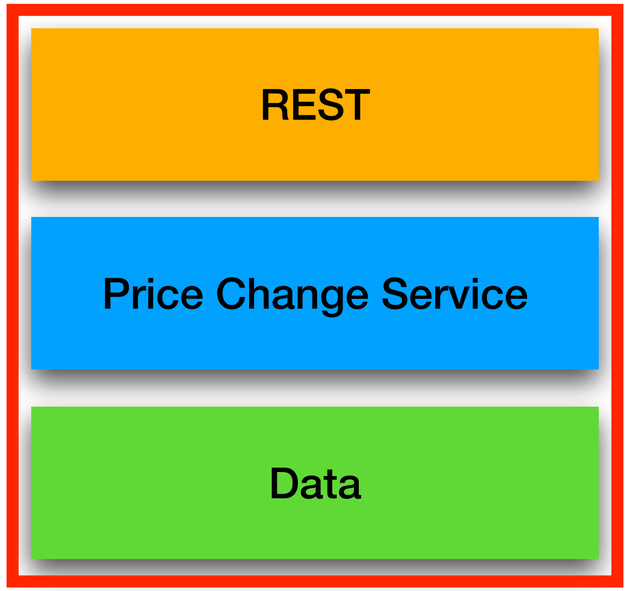

GraphQL
Internal
REST
REST
Asset Service
Price Change Service
Price History Service
Data
Data
Data



https://server/api/asset/price/change?symbol=BTC&span=Month

https://server/api/asset/price/change?symbol=BTC&span=Month


```
{
    "symbol": "BTC",
    "span": "Month",
    "percentageChange": 0.10344501837363199
}
```

Demo 3

Integrating data from REST services

DataLoader

```
query GetPriceChanges {
  assets {
    nodes {
      symbol
      price {
        change(span: MONTH) {
          percentageChange
        }
      }
    }
  }
}
```

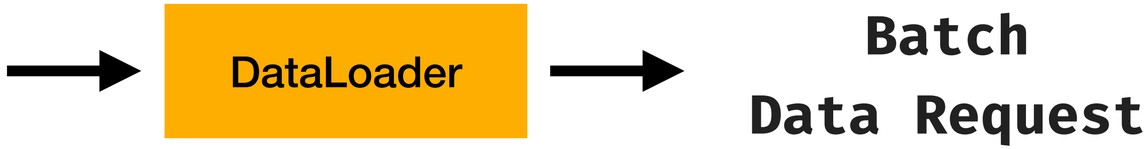
```
query GetPriceChanges {
  assets {
    nodes {
      symbol
      price {
        change(span: MONTH) {
          percentageChange
        }
      }
    }
  }
}
```

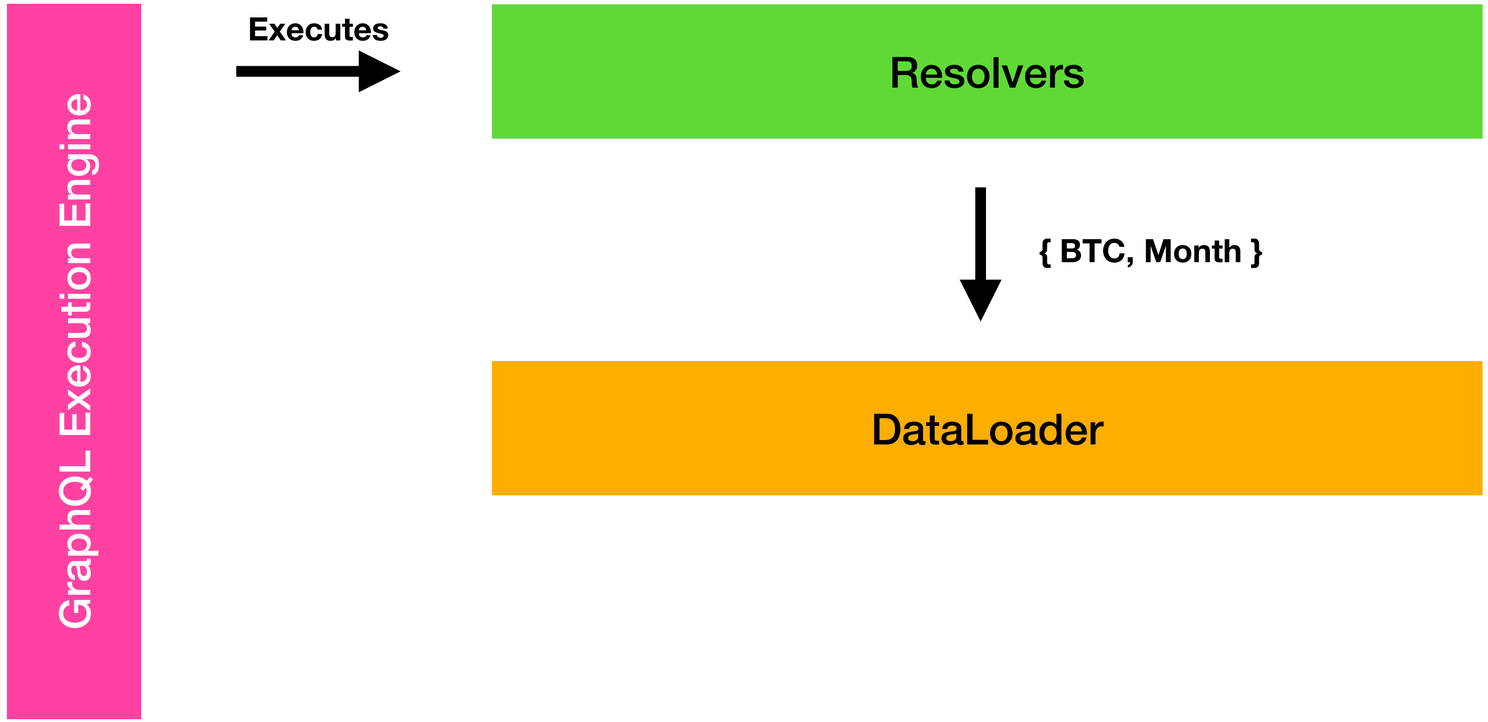
→ **HTTP Request**

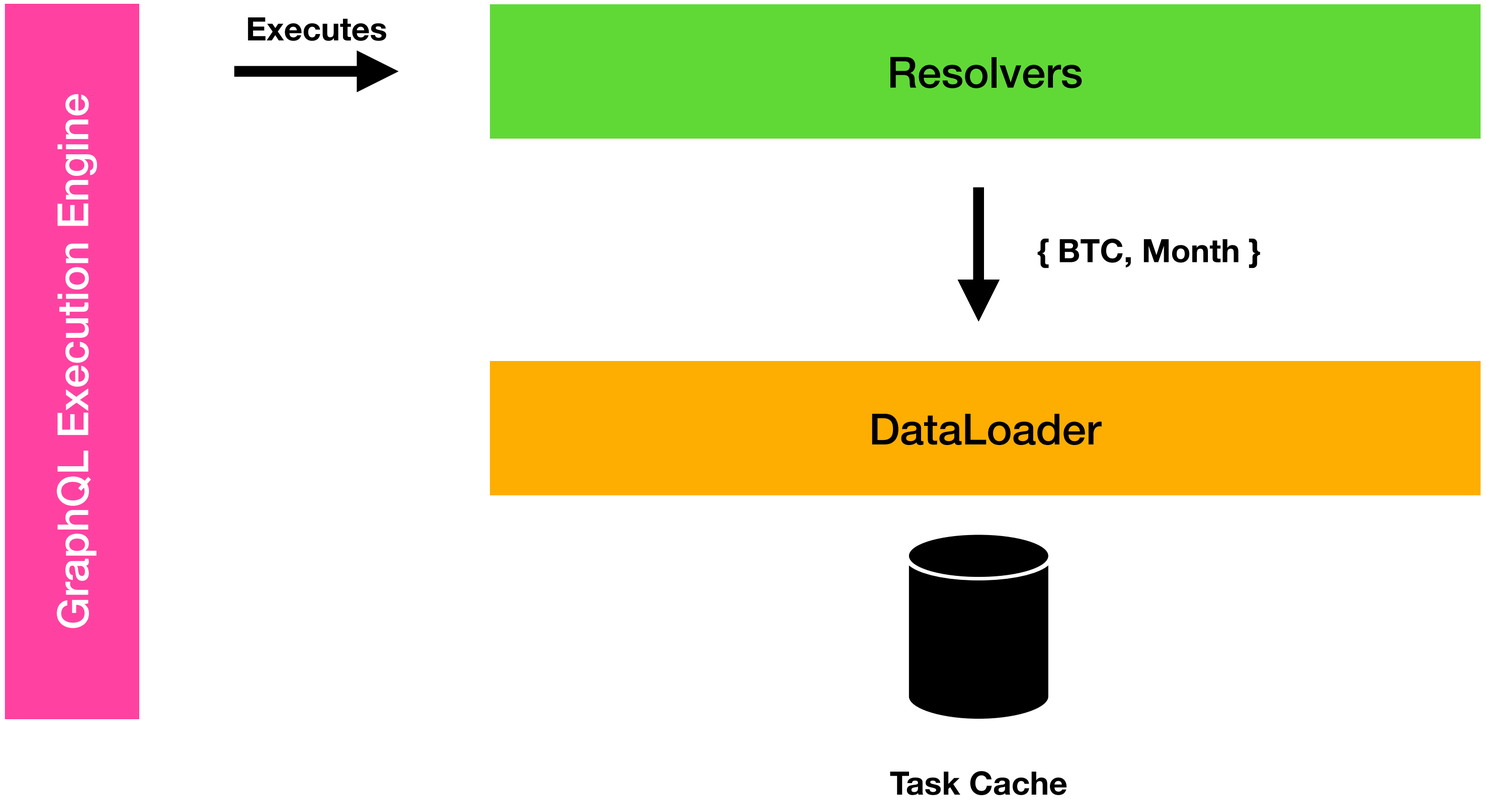
```
query GetPriceChanges {
  assets {
    nodes {
      symbol
      price {
        change(span: MONTH) {
          percentageChange
        }
      }
    }
  }
}
```

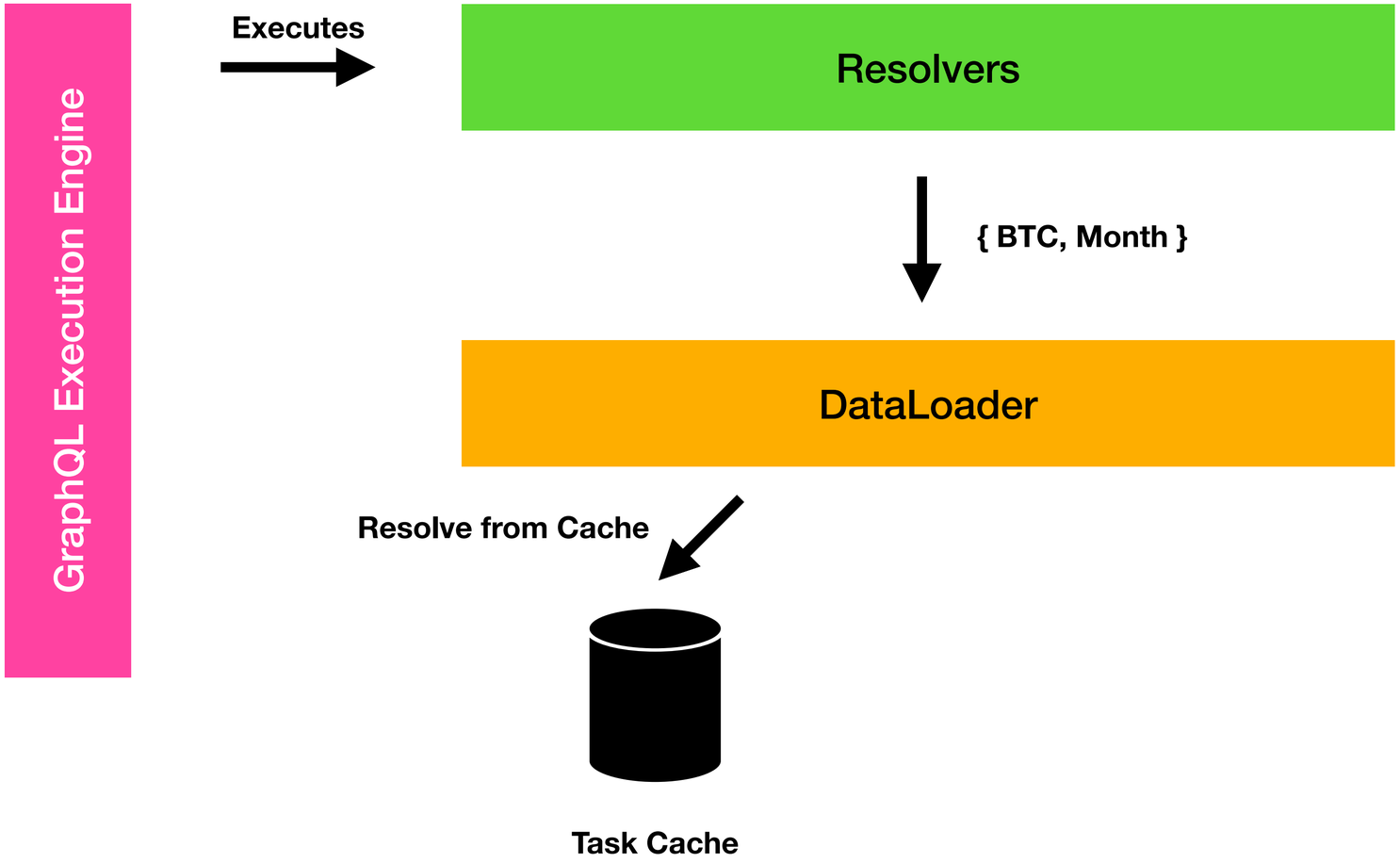
→ **Data Request**

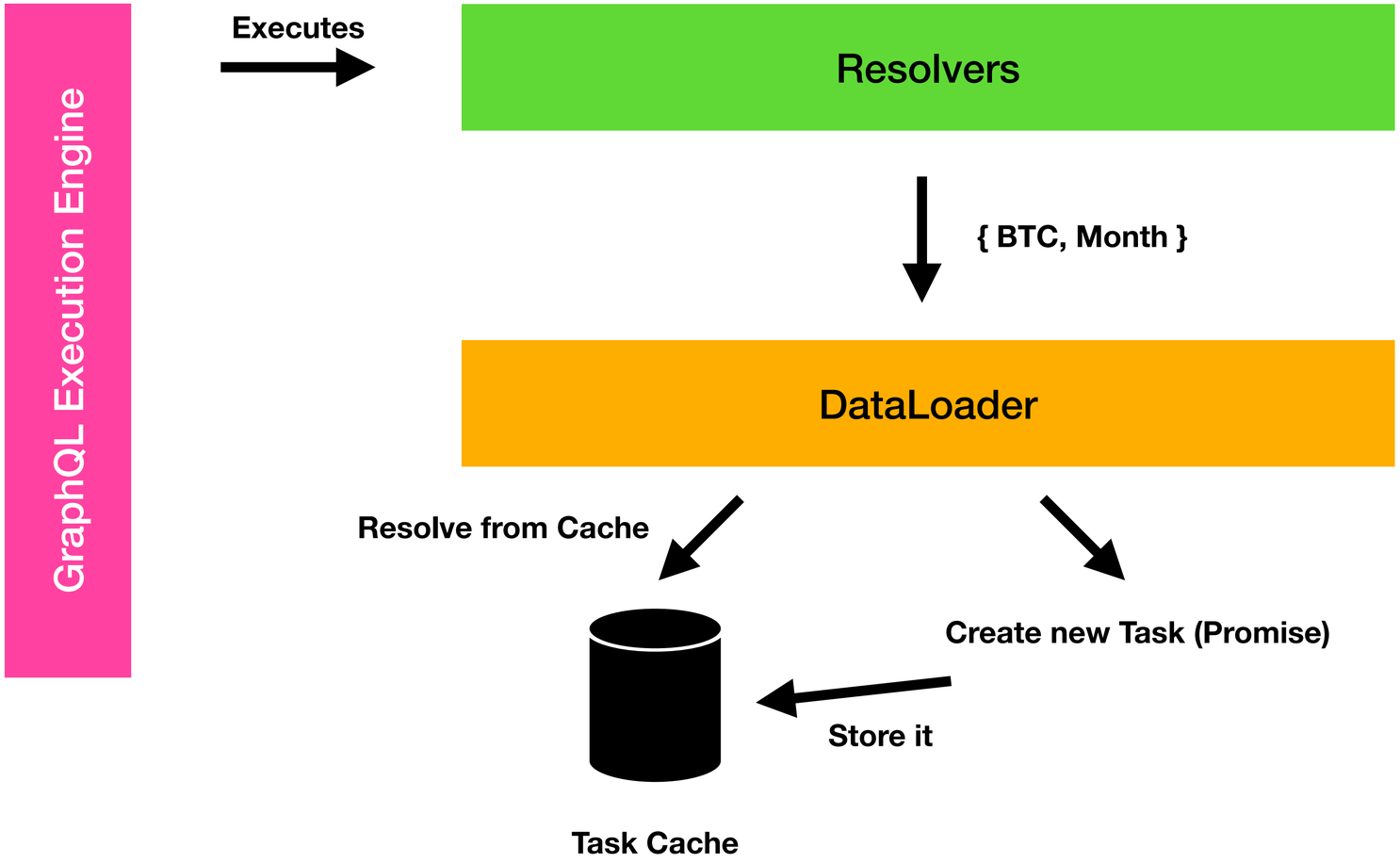
```
query GetPriceChanges {
  assets {
    nodes {
      symbol
      price {
        change(span: MONTH) {
          percentageChange
        }
      }
    }
  }
}
```

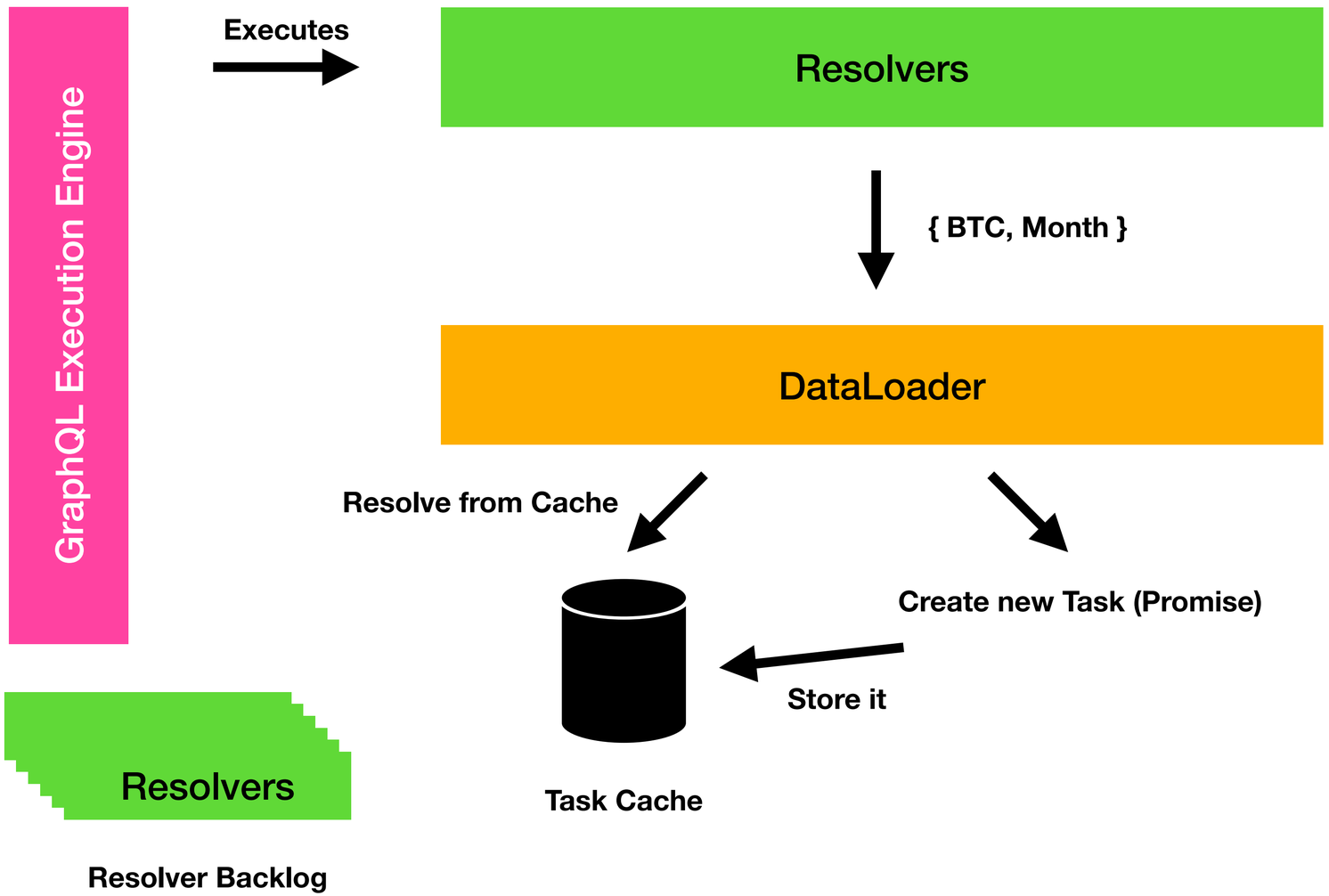


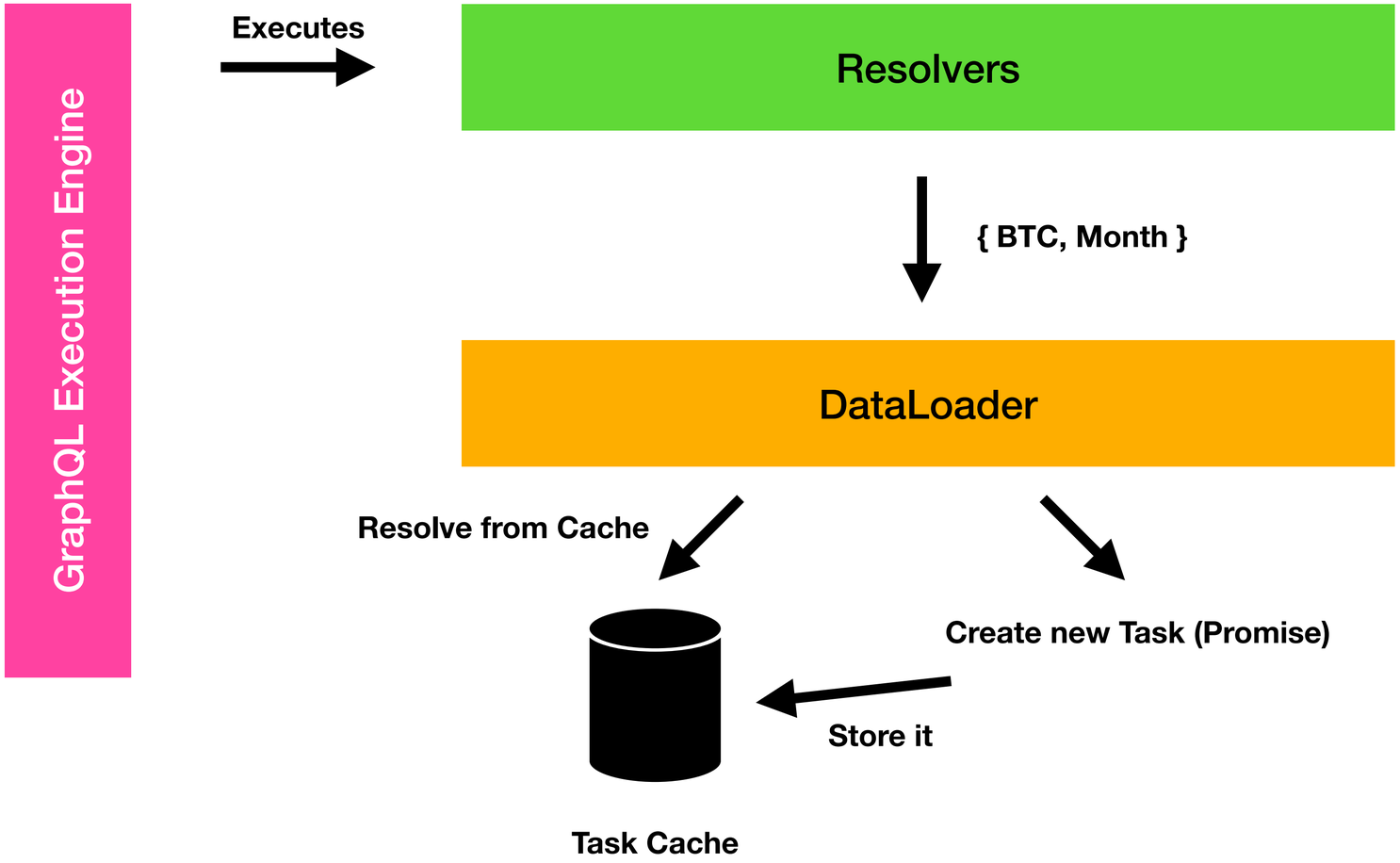

GraphQL Execution Engine
Executes
Resolvers
{ BTC, Month }
DataLoader

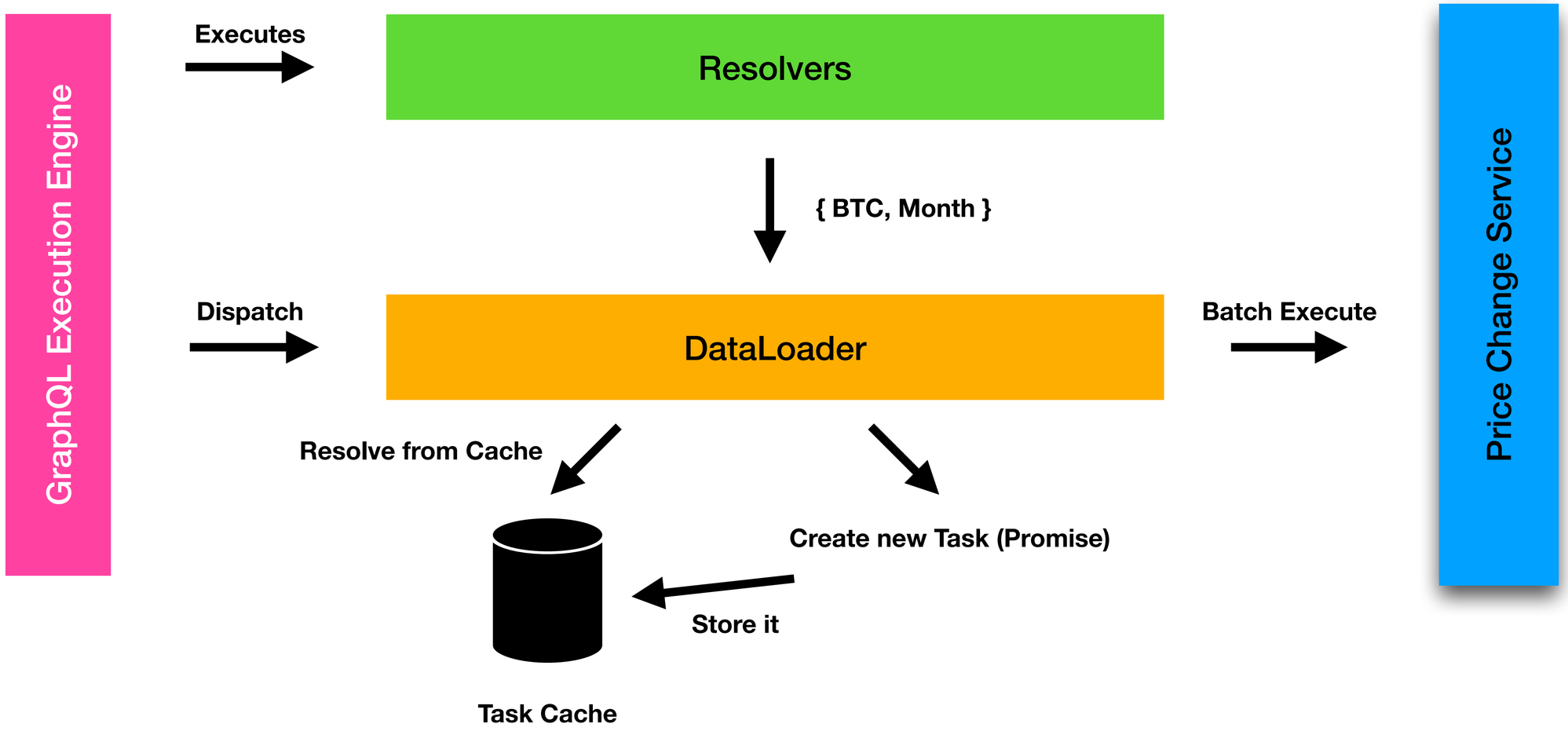

GraphQL Execution Engine
Executes
Resolvers
{ BTC, Month }
DataLoader
Task Cache


GraphQL Execution Engine
Executes
Resolvers
{ BTC, Month }
DataLoader
Resolve from Cache
Task Cache


GraphQL Execution Engine
Executes
Resolvers
{ BTC, Month }
DataLoader
Resolve from Cache
Create new Task (Promise)
Store it
Task Cache

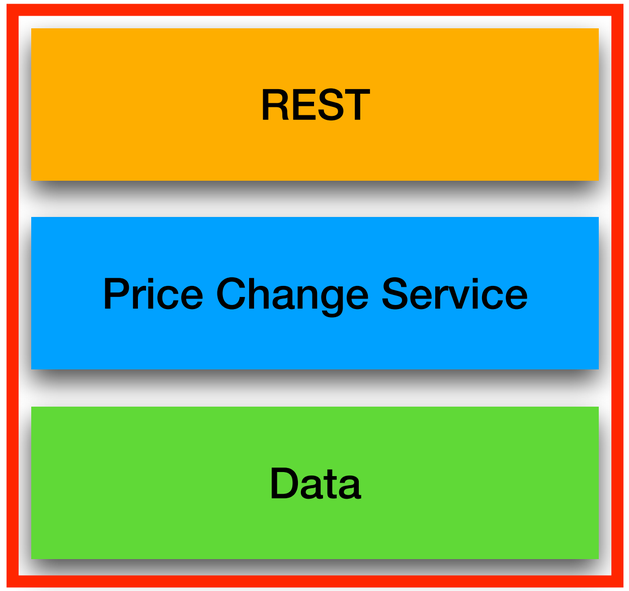

GraphQL Execution Engine
Executes
Resolvers
{ BTC, Month }
DataLoader
Resolve from Cache
Create new Task (Promise)
Store it
Task Cache
Resolvers
Resolver Backlog


GraphQL Execution Engine
Executes
Resolvers
{ BTC, Month }
DataLoader
Resolve from Cache
Create new Task (Promise)
Store it
Task Cache


GraphQL Execution Engine
Executes
Resolvers
{ BTC, Month }
Dispatch
DataLoader
Batch Execute
Price Change Service
Resolve from Cache
Create new Task (Promise)
Store it
Task Cache

https://server/api/asset/price/change?**symbols=BTC,ADA**&span=Month


```
[
  {
    "symbol": "BTC",
    "span": "Month",
    "percentageChange": 0.10344501837363199
  },
  {
    "symbol": "ADA",
    "span": "Month",
    "percentageChange": 0.24657676348547722
  }
]
```

DataLoader

- Improves Data Fetching
- Ensures Consistency

Example 3

Optimising Data Fetching with DataLoader

Real-Time Data

GraphQL Operations

Operation	GraphQL	REST
Read	Query	GET
Write	Mutation	PUT, POST, PATCH, DELETE
Events	Subscription	N/A

GraphQL Operations

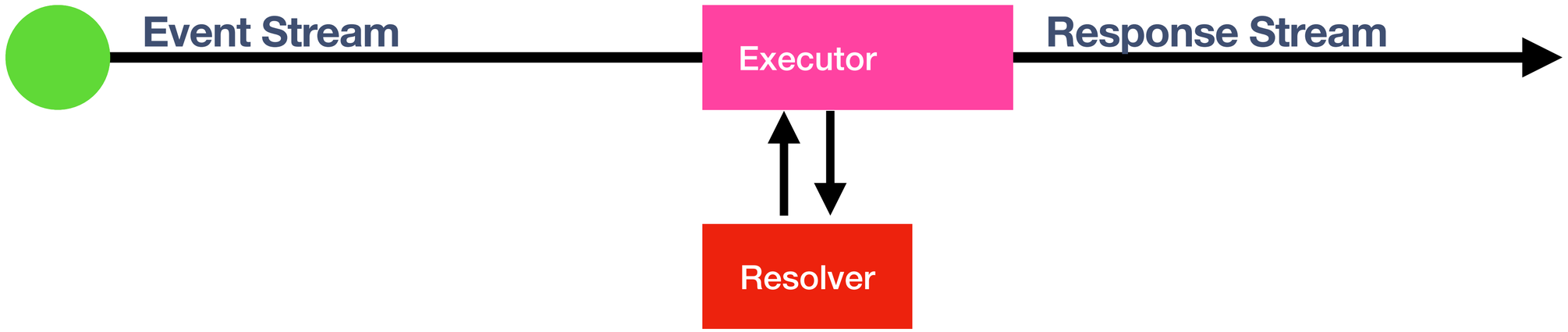
Operation	GraphQL	REST
Read	Query	GET
Write	Mutation	PUT, POST, PATCH, DELETE
→ Events	Subscription	N/A

Subscriptions

- Subscriptions instead of a single response return a response stream.
- Subscriptions do NOT work over standard HTTP requests
- Events, an event that happens somewhere
- Responses are created when an event is raised.

Event Stream

Executor



Demo 4

Making it real-time 🚀

Problems

- Multiple Subscriptions -> Multiplexing
- Scaling
- Throttling -> batching
- Quality of Service

@defer and @stream

```
query GetAssets {
  assets {
    nodes {
      name
      price {
        latestPrice
        change24hour
      }
    }
  }
}
```

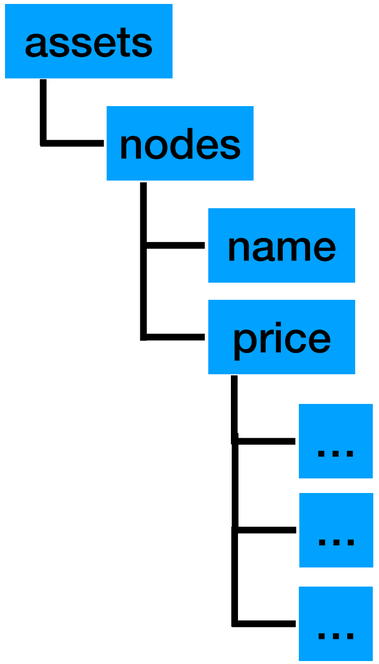
```
query GetAssets {
  assets {
    nodes {
      name
      price {
        latestPrice
        change24hour
      }
    }
  }
}
```

@defer and @stream allow the consumer to express what parts of the query can be **depriorized.**

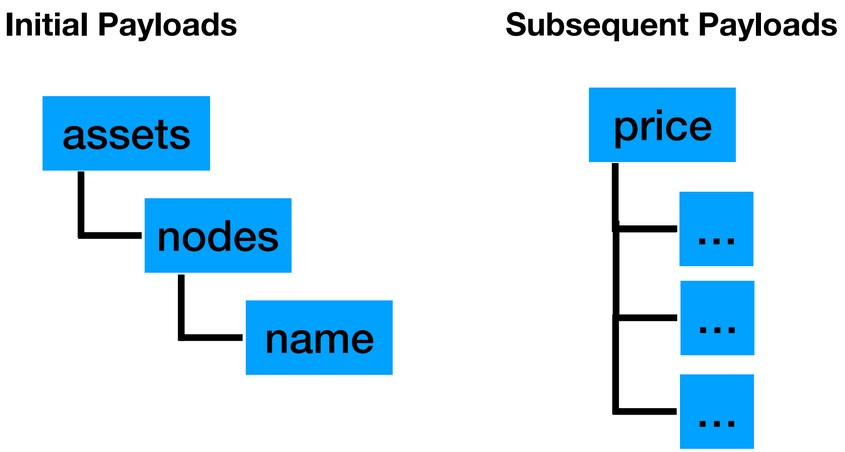
```
query GetAssets {
  assets {
    nodes {
      name
      ... @defer {
        price {
          latestPrice
          change24hour
        }
      }
    }
  }
}
```

```
query GetAssets {
  assets {
    nodes {
      name
      … Price @defer
    }
  }
}

fragment Price on AssetPrice {
  price {
    latestPrice
    change24hour
  }
}
```

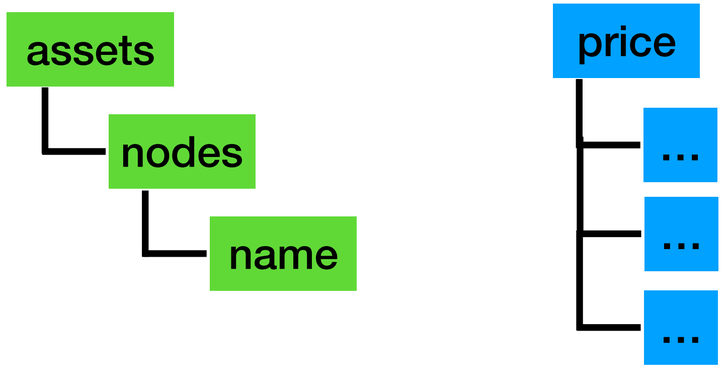
```graphql
query GetAssets {
  assets {
    nodes {
      name
      … Price @defer
    }
  }
}

fragment Price on AssetPrice {
  price {
    latestPrice
    change24hour
  }
}
```



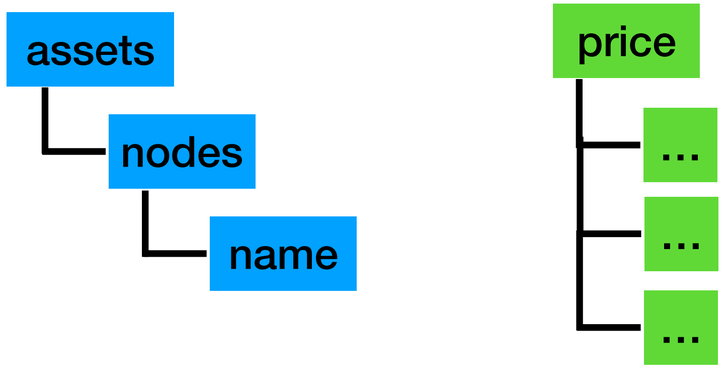
```
query GetAssets {
  assets {
    nodes {
      name
      … Price @defer
    }
  }
}

fragment Price on AssetPrice {
  price {
    latestPrice
    change24hour
  }
}
```



```
query GetAssets {
  assets {
    nodes {
      name
      … Price @defer
    }
  }
}

fragment Price on AssetPrice {
  price {
    latestPrice
    change24hour
  }
}
```



```
query GetAssets {
  assets {
    nodes {
      name
      … Price @defer
    }
  }
}

fragment Price on AssetPrice {
  price {
    latestPrice
    change24hour
  }
}
```



Conclusion

Conclusion

- GraphQL is evolving at a rapid pace.
- More control over data
  - @defer / @stream -> prioritisation
  - Error boundaries -> data quality
- Stronger Type System
  - @oneOf
  - Interfaces implement interfaces
  - Repeatable directives

Thank You

Social Media:
Follow me on GitHub: bit.ly/michaelGitHub
Follow me on Twitter: bit.ly/michaelTwitter
Connect on LinkedIn: bit.ly/michaelLinkedIn
Subscribe on YouTube: youtube.chillicream.com

Demos: bit.ly/reactor-examples

Help us by starring us on GitHub:
https://github.com/ChilliCream/hotchocolate

Community: slack.chillicream.com
Web: chillicream.com