playgrounds

Learn
Play
Discover
Grow

Providing advanced on-chain data analytics tools, interactive education hubs, and protocol simulation environments.

playgrounds

Content

- Motivations for Subgrounds, Recap and some GraphQL basics.
- FieldPaths: What are they and how do we combine them to create subgrounds requests?
- subgrounds.query method: the simplest way to get data.
- subgrounds.query_timeseries method: query and return regularized timeseries data.
- Creating and using SyntheticFields.
- Querying non-Subgraph APIs with Subgrounds.

playgrounds

Our Journey

Q2 2021

Protean Labs receives grants from The Graph Foundation to develop Mesh Engine.

Q3 2021

Mesh Engine evolves into what will become Subgrounds

Q4 2021

Protean Labs joins forces with Playgrounds to revolutionize Web3 data science. Mesh Engine rebrands to Subgrounds

Q1 2022

Playgrounds starts offering subgrounds powered data infrastructures to clients

Motivation

Why did we build Subgrounds?

- Leverage The Graph and use its vast trove of pre-modeled data.
- Leverage Python for its immense data science and analytics ecosystem.
- Recover the Web2 data science stack in Web3.
- Empower data scientists, analysts, engineers, and hobbyists with an advanced yet accessible and familiar set of tools for on-chain data analytics.

playgrounds

The Graph protocol, through subgraphs, organizes smart contract data in a meaningful way or in a manner that best represents the expected behaviors of the smart contract.

The Graph protocol saves developers the tremendous overhead required to create reliable and efficient data feeds from the blockchain.

The Graph offers developers easy access to open-source API subgraphs and data models across multiple networks, about 24 networks today.

The Graph eliminates the deep technical knowledge required to index on-chain data.

The Graph

The Graph is the foundation for advanced multi-chain data analytics in Web3

playgrounds

Python

Python is essentially the lingua franca of data science and analytics.

Python provides simple syntax which lowers the entry barrier for data analytics and financial modeling

Python and data science are tightly interwoven with libraries such as Pandas, NumPy, sci-kit learn essentially built in to every Python development environment

Python enables the creation of light weight and scalable data analytics programs

Python Naturally extends itself to robust machine learning tools

playgrounds

Subgrounds

Subgrounds enables advanced yet accessible and familiar set of tools for on-chain data analytics.

- Highly extensible, modular, and provides continuity with existing data analytics tools.
- Built in Plotly wrappers enable model based transformation and visualization of on-chain data.
- Enables manipulations and reflect the domain in which they are defined.
- Minimally verbose and significantly reduces on-chain analytics learning curve.
- Provides accessible dashboards which can either be auto generated or customized to varying degrees.
- Entirely based on subgraph schemas made available through The Graph,

playgrounds

GraphQL Basics

playgrounds

GraphQL Basics

Schema to Response

```
type Query {
  pair(id: ID!): Pair
}

type Pair {
  id: ID!
  token0: Token!
  token1: Token!
}

type Token {
  id: ID!
  symbol: String!
  name: String!
}
```

```
query {
  pair(id: "0xb4e16d0168e52d35cacd2c6185b44281ec28c9dc") {
    token0 {
      name
      symbol
    }
    token1 {
      name
      symbol
    }
  }
}
```

```
{
  "data": {
    "pair": {
      "token0": {
        "name": "USD//C",
        "symbol": "USDC"
      },
      "token1": {
        "name": "Wrapped Ether",
        "symbol": "WETH"
      }
    }
  }
}
```

playgrounds

GraphQL Basics

GraphQL vs REST

	GraphQL	REST
Architecture	client-driven	server-driven
Organized in terms of	schema & type system	endpoints
Operations	Query Mutation Subscription	Create, Read, Update, Delete
Data fetching	specific data with a single API call	fixed data with multiple API calls
Community	growing	large
Performance	fast	multiple network calls take up more time
Development speed	rapid	slower
Learning curve	difficult	moderate
Self-documenting	✓	—
File uploading	—	✓
Web caching	(via libraries built on top)	✓
Stability	less error prone, automatic validation and type checking	better choice for complex queries
Use cases	multiple microservices, mobile apps	simple apps, resource-driven apps

playgrounds

Demo

playgrounds

Subgrounds

Under the hood

playgrounds

```
sg.query([
  uniswapV2.Query.pairs.token0.symbol
])
```

Subgrounds

```
query {
  pairs {
    token0 {
      symbol
    }
  }
}
```

GraphQL

FieldPaths

Subgrounds and GraphQL

playgrounds

```
sg.query([
  uniswapV2.Query.pairs.token0.symbol
])
```

Subgrounds

```
1  query {
2    pairs {
3      token0 {
4        symbol
5      }
6    }
7  }
```

GraphQL

FieldPaths

Subgrounds and GraphQL

playgrounds

```
pairs = uniswapV2.Query.pairs(
  orderBy=uniswapV2.Pair.timestamp,
  orderDirection='desc',
  first=10
)

sg.query([
  pairs.token0.symbol,
  pairs.token1.symbol
])
```

Subgrounds

```
query {
  pairs(
    orderBy: timestamp,
    orderDirection: desc,
    first: 10
  ) {
    token0 {
      symbol
    }
    token1 {
      symbol
    }
  }
}
```

GraphQL

FieldPaths

Subgrounds and GraphQL

playgrounds

```
pairs = uniswapV2.Query.pairs(
  orderBy=uniswapV2.Pair.timestamp,
  orderDirection='desc',
  first=10
)

sg.query([
  pairs.token0.symbol,
  pairs.token1.symbol
])
```

Subgrounds

```
query {
  pairs(
    orderBy: timestamp,
    orderDirection: desc,
    first: 10
  ) {
    token0 {
      symbol
    }
    token1 {
      symbol
    }
  }
}
```

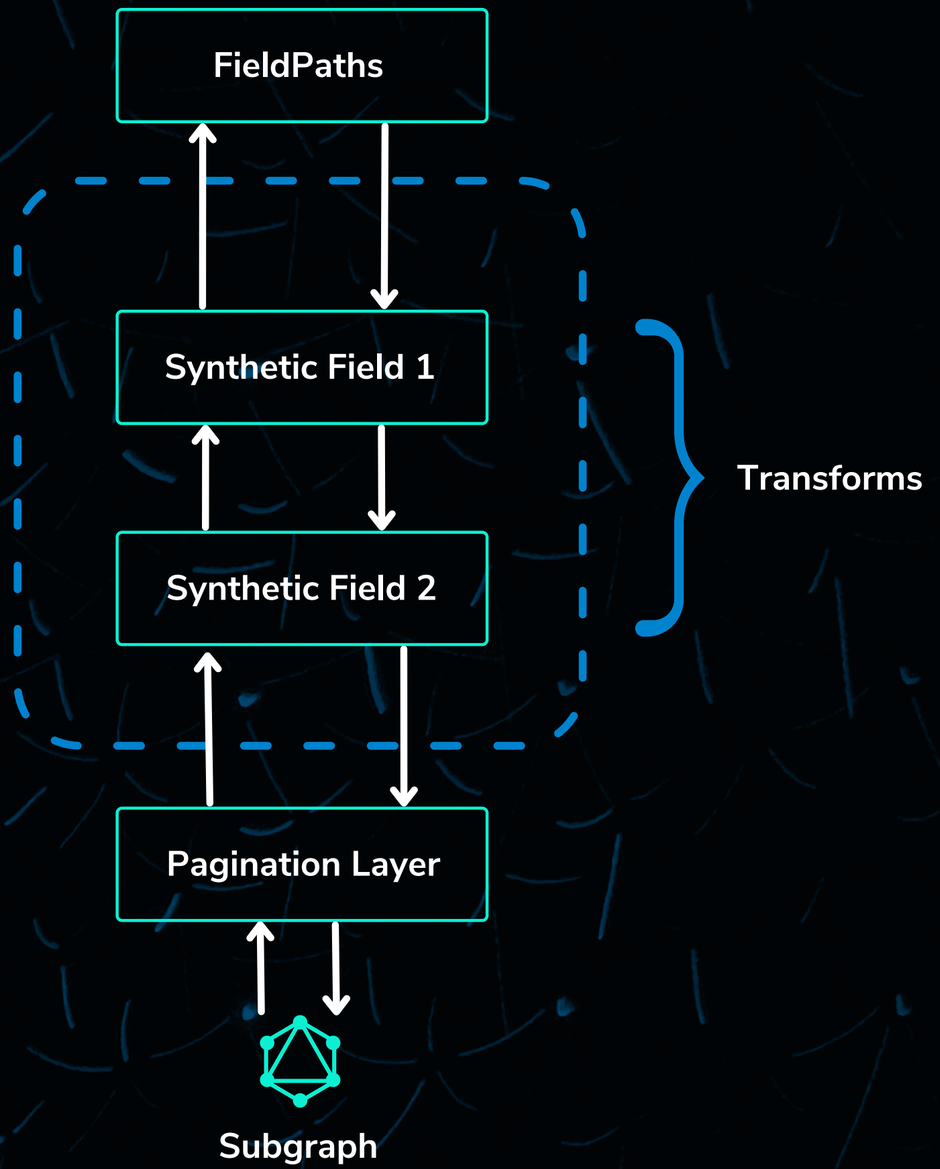
GraphQL

FieldPaths

Subgrounds and GraphQL

playgrounds

Anatomy of a Subgrounds request



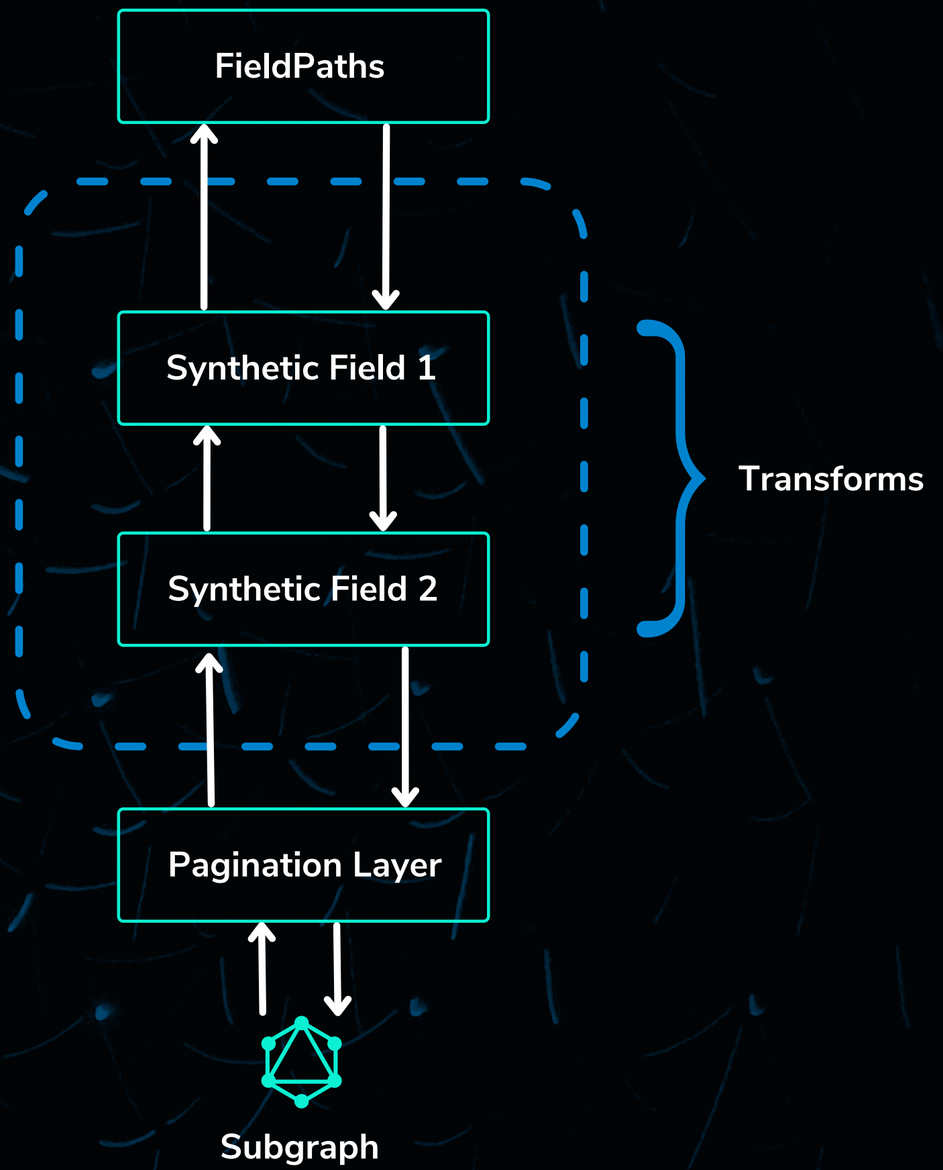
playgrounds

Subgraph

```
type Swap @entity {
  amount0: BigInt
  amount1: BigInt
}
```

Subgrounds

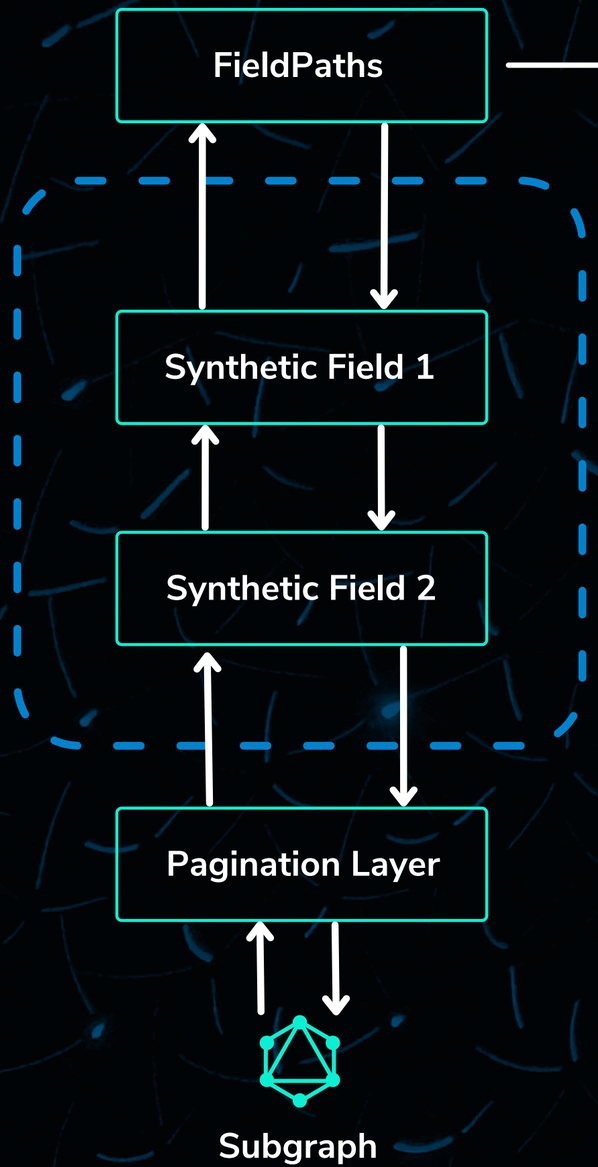
```
subgraph.Swap.price = subgraph.Swap.amount0 / subgraph.Swap.amount1
```



Pre-querying

Defining a synthetic field

playgrounds



Transforms

Subgrounds

```
sg.query([
  subgraph.Query.swaps.price
])
```

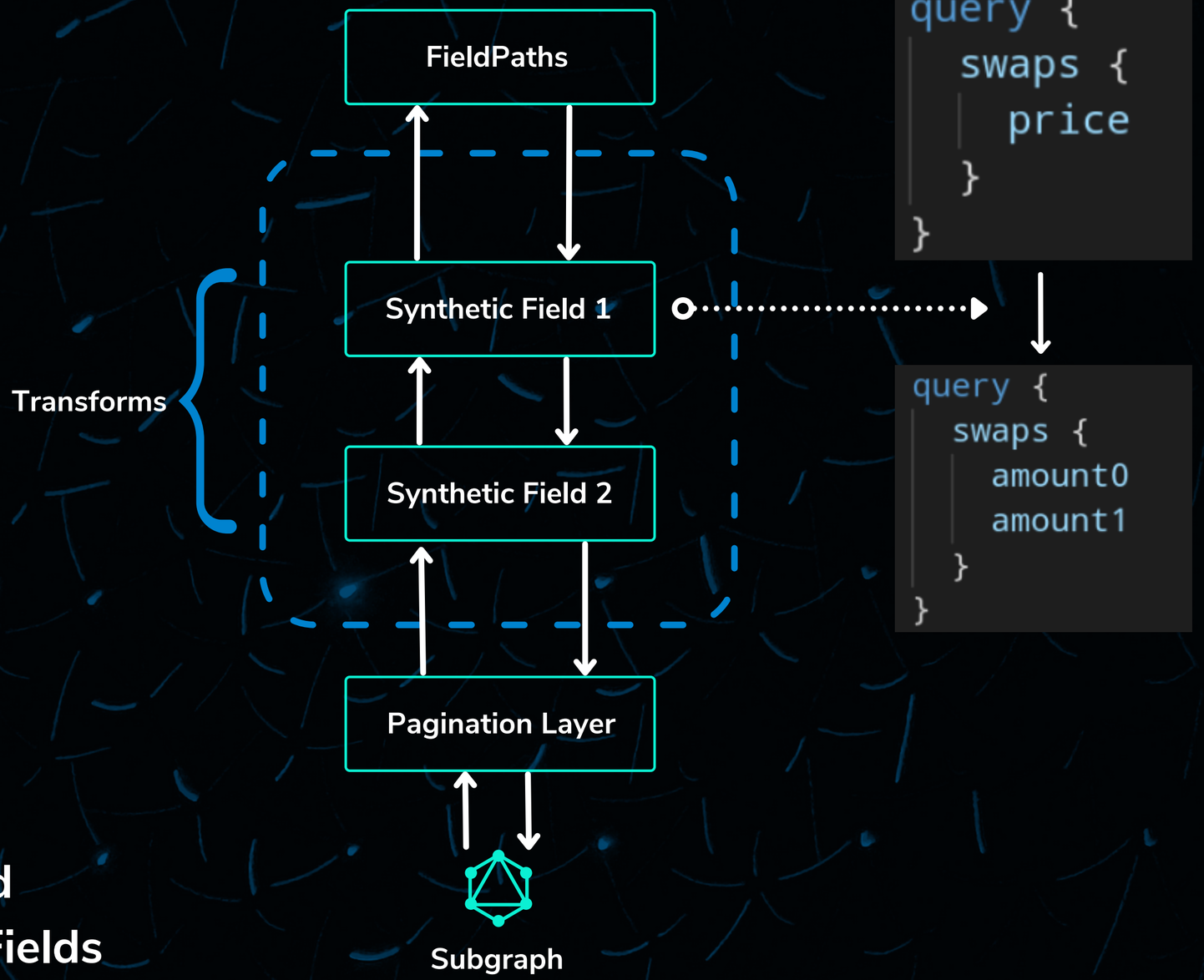
=

```
query {
  swaps {
    price
  }
}
```

Step 1

Construct requests using FieldPaths

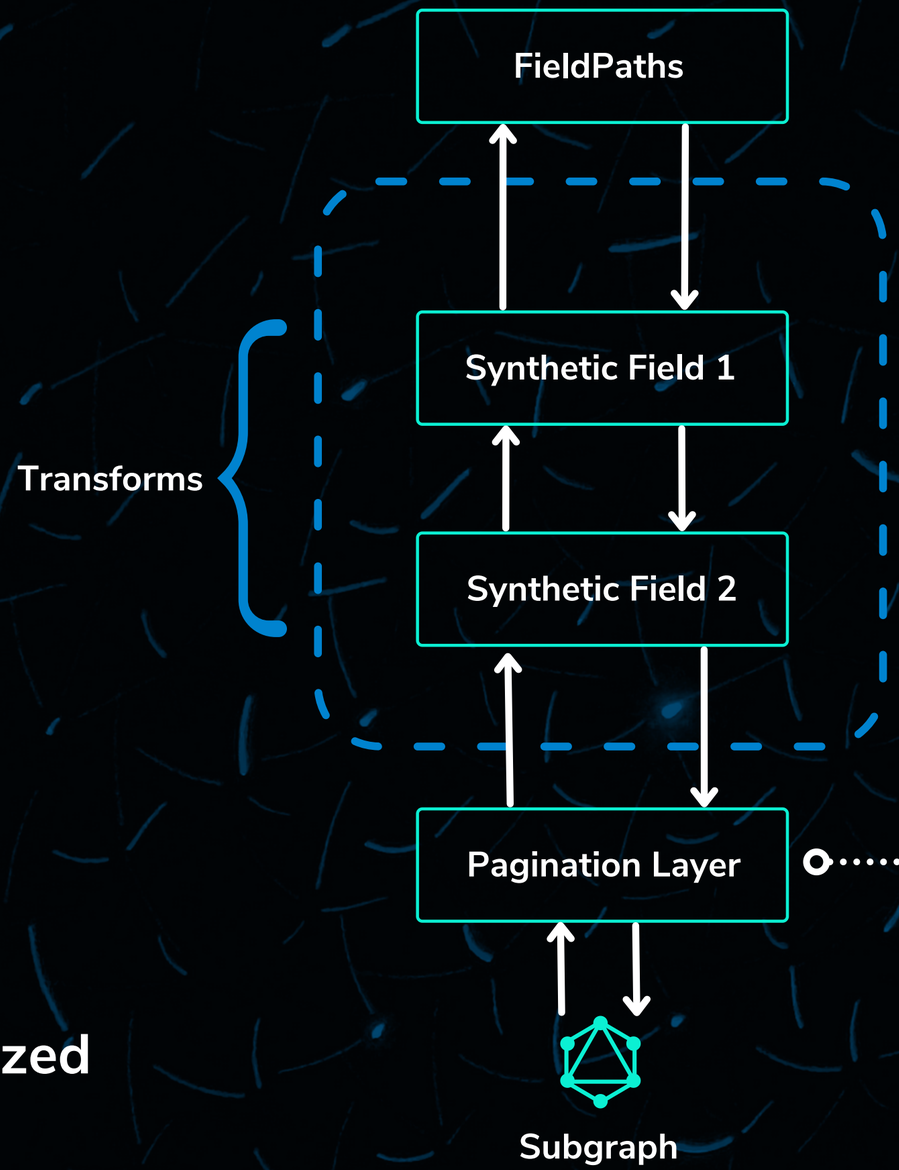
playgrounds



Step 2

Initial query transformed according to Synthetic Fields

playgrounds



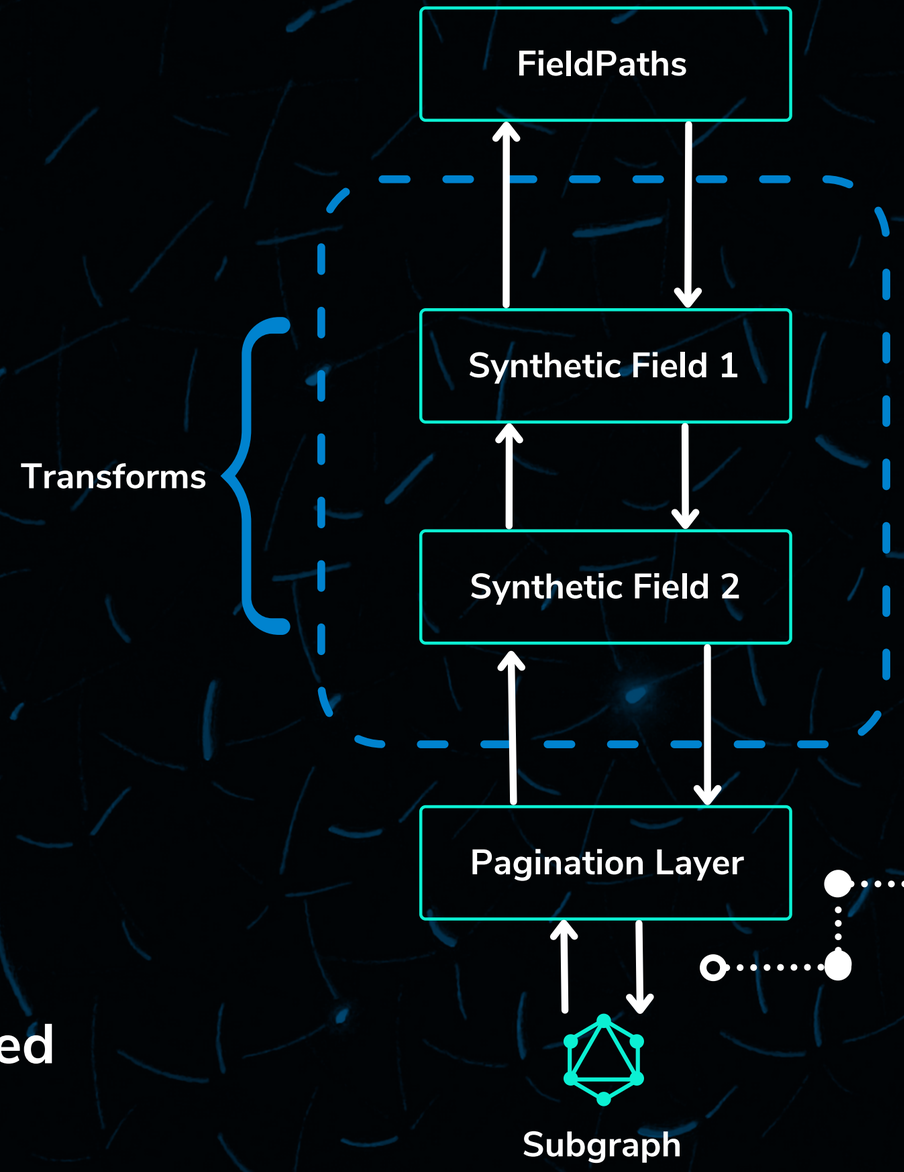

```
query {
  swaps {
    amount0
    amount1
  }
}
```

```
query(
  $first0: BigInt,
  $skip0: BigInt,
  $lastOrderingValue0: String
) {
  swaps(
    first: $first0,
    skip: $skip0,
    orderBy: id,
    orderDirection: asc
    where: {
      id_gt: $lastOrderingValue0
    }
  ) {
    id
    amount0
    amount1
  }
}
```

Step 3

Query normalized for pagination

playgrounds




```
query(
  $first0: BigInt,
  $skip0: BigInt,
  $lastOrderingValue0: String
) {
  swaps(
    first: $first0,
    skip: $skip0,
    orderBy: id,
    orderDirection: asc
    where: {
      id_gt: $lastOrderingValue0
    }
  ) {
    id
    amount0
    amount1
  }
}
```

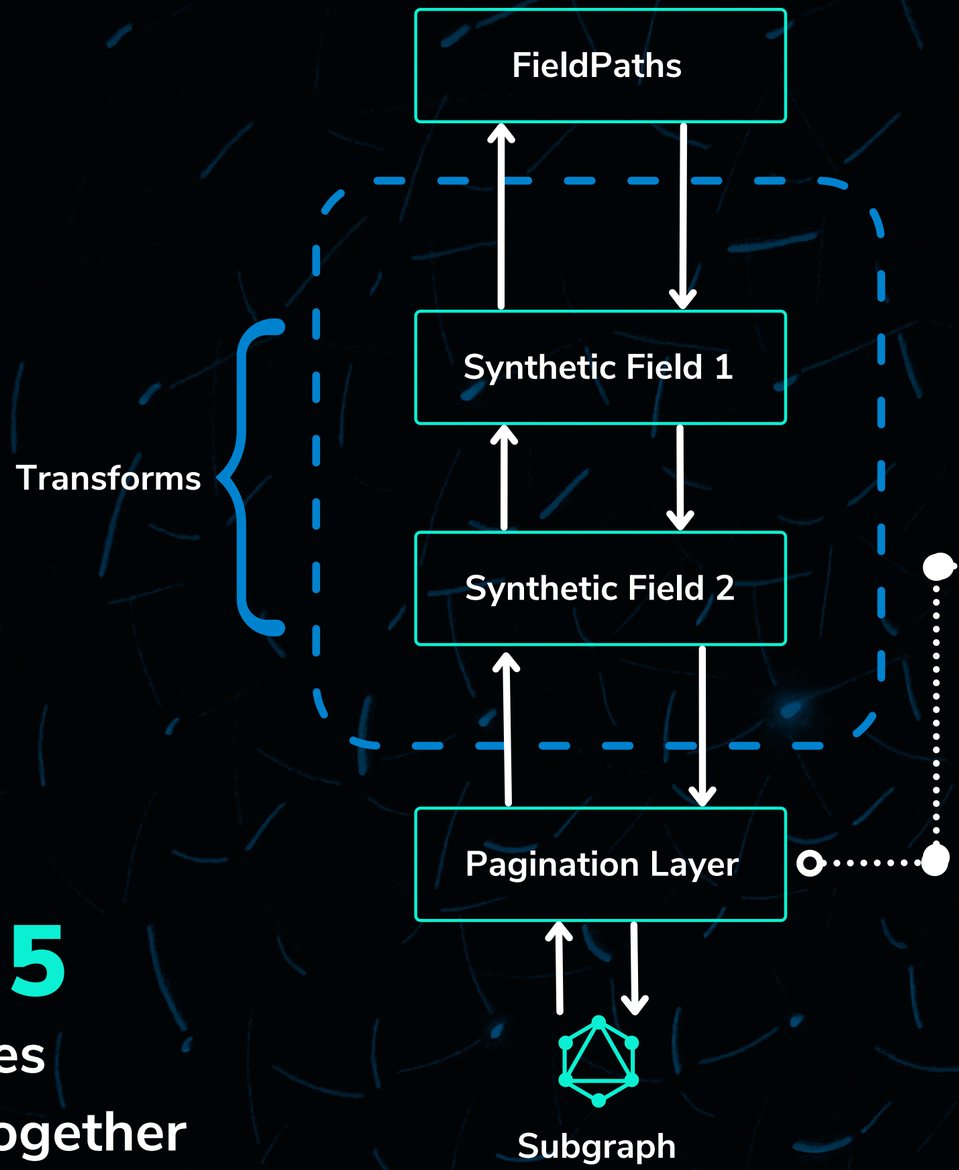
Req 1
Req 2
Req 3

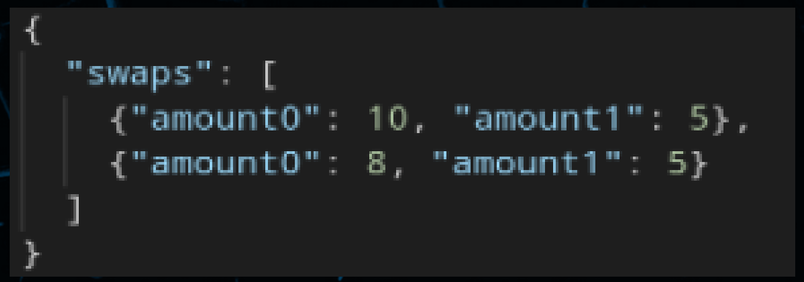
```
{
  "first0": ...,
  "skip0": ...,
  "lastOrderingValue0": ...
}
```

Step 4

Query normalized for pagination

playgrounds



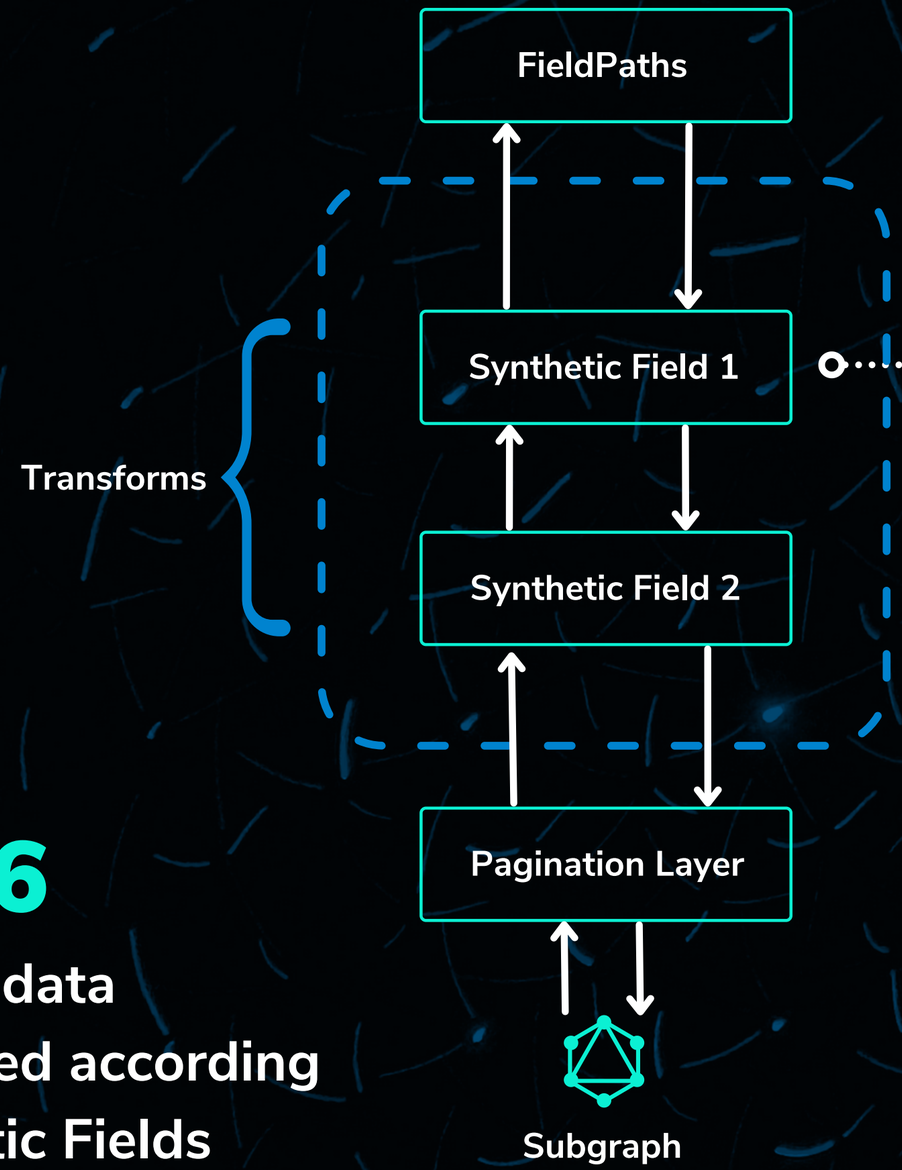


Combined response data

Step 5

Data pages merged together

playgrounds



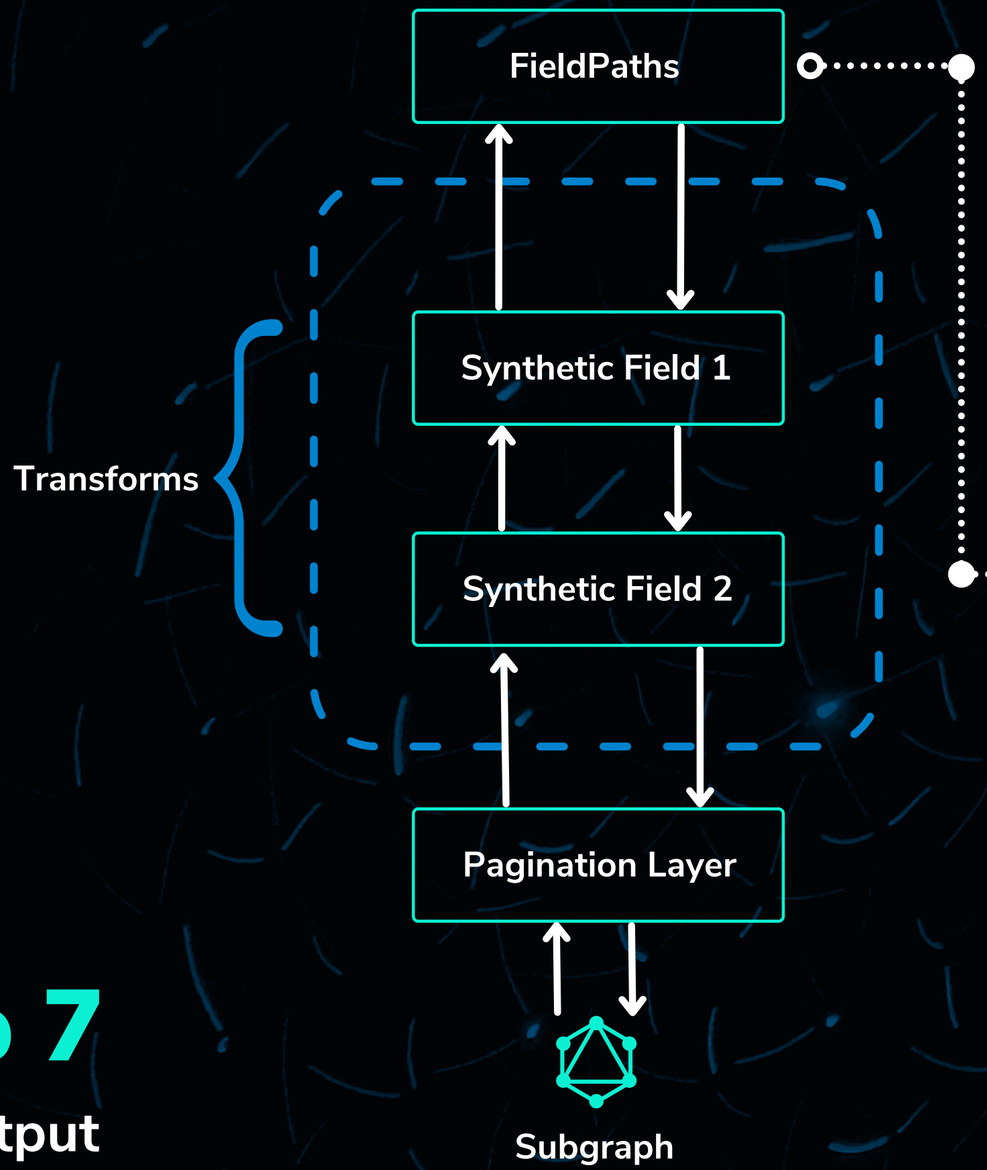

```
{
  "swaps": [
    {"amount0": 10, "amount1": 5},
    {"amount0": 8, "amount1": 5}
  ]
}
```

```
{
  "swaps": [
    {"price": 2},
    {"price": 1.6}
  ]
}
```

Step 6

Response data transformed according to Synthetic Fields

playgrounds




```
sg.query([
  subgraph.Query.swaps.price
])
```

✓ 0.4s

```
[2.0, 1.6]
```

Step 7

Data output

playgrounds

Demo

Bringing everything back together


playgrounds