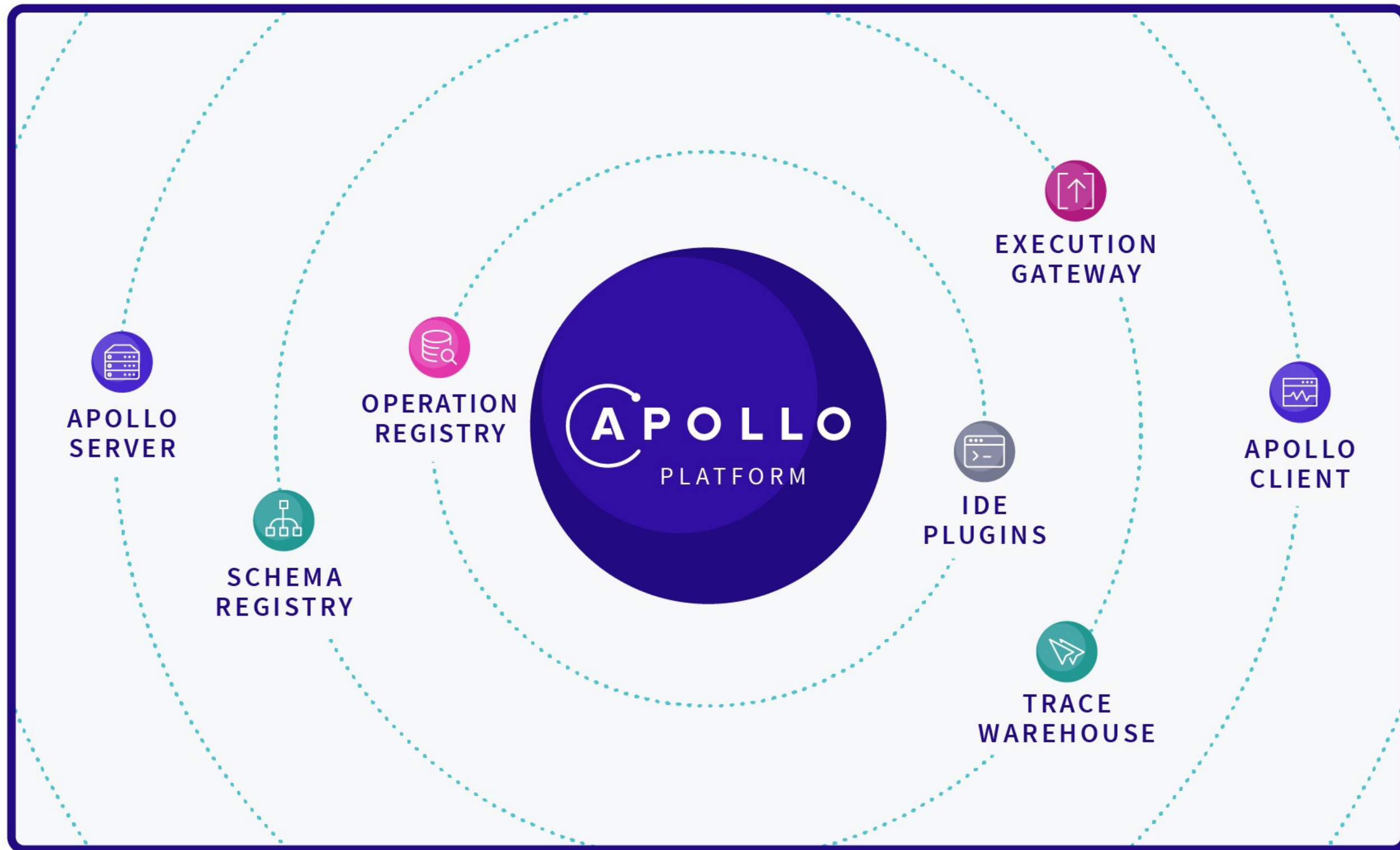


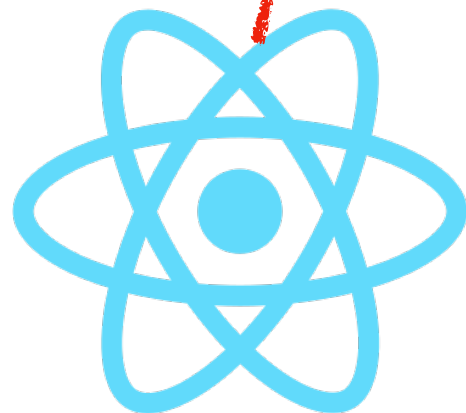
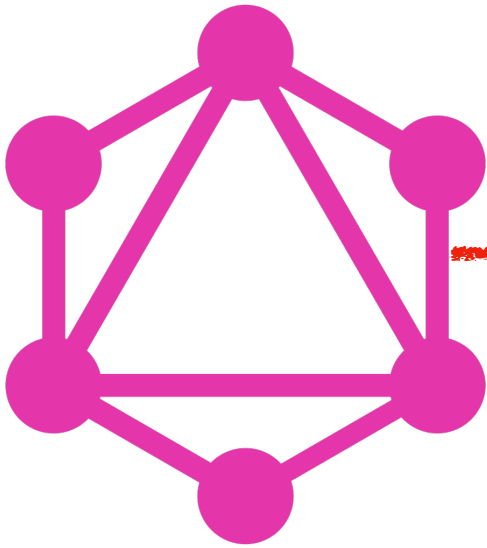
React Apollo

Apollo?





React + Apollo



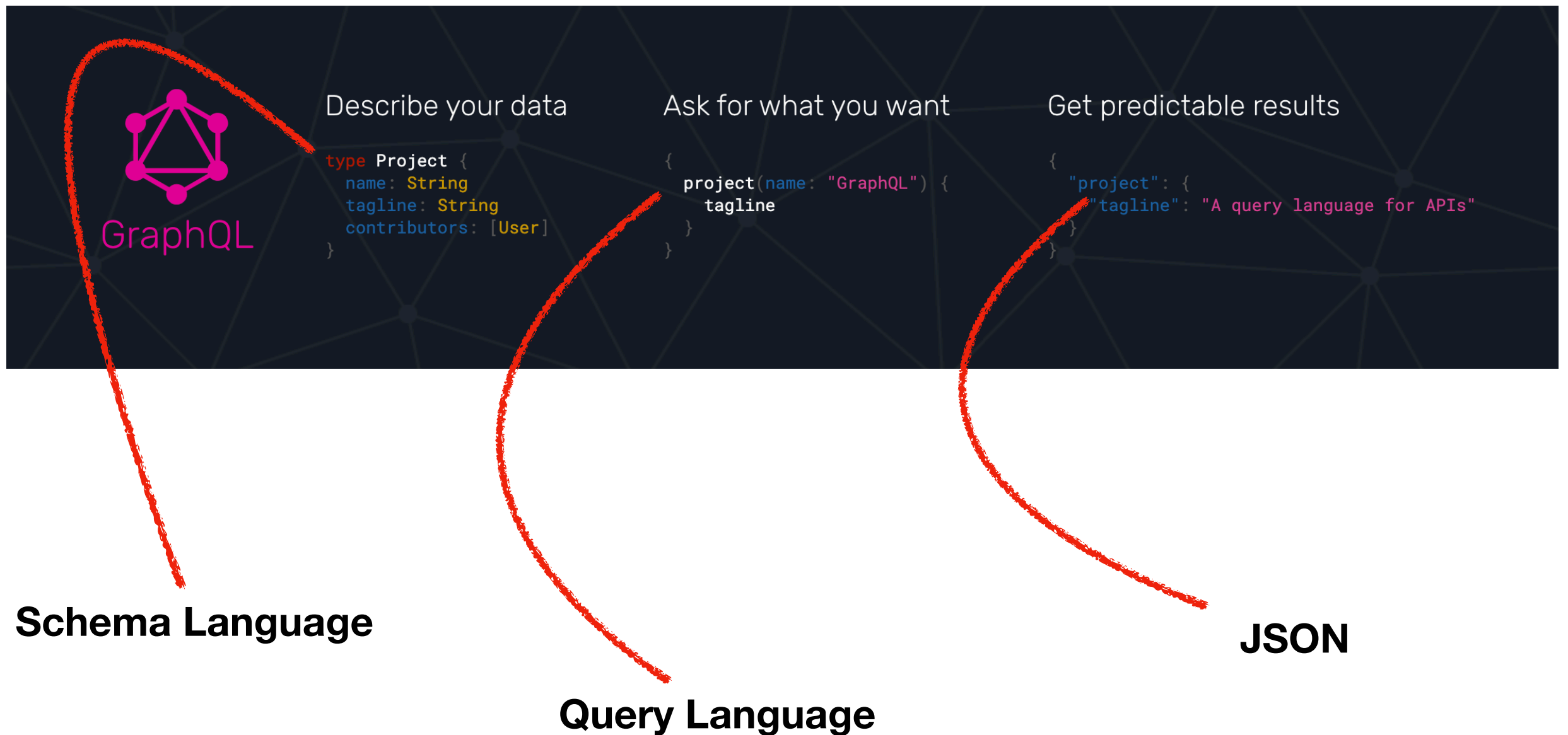
```
import React from 'react'
import { Query } from 'react-apollo'
import gql from 'graphql-tag'

const QUERY = gql`
query ($id: ID!) { user(id: $id) { id name } }
`;

const App: React.FC = () => (
  <Query query={QUERY} variables={{ id: "7" }}>
    ({ { data, loading, error }: any}) => {
      if (loading)
        return <p>Loading...</p>;
      else if (error || !data)
        return <p>Error!</p>;
      else
        return <p>{data.user.name}</p>;
    }
  </Query>
);

export default App
```

GraphQL?



Schema

- 제공할 데이터를 묘사합니다
- 주로 서버에서 사용합니다

```
type Query {  
  user(id: ID!): User  
  message(id: ID!): Message  
}  
  
type User {  
  id: ID!  
  name: String!  
}  
  
type Message {  
  id: ID!  
  payload: String!  
  createdAt: String!  
  author: User!  
}
```

Query

- 요청할 데이터를 묘사합니다
- 클라이언트에서 사용합니다

```
query AwesomeQuery {  
  user(id: "1") {  
    id  
    name  
  }  
  firstMessage: message(id: "0") {  
    id  
    payload  
    createdAt  
    author {  
      id  
      name  
    }  
  }  
}
```

GraphQL!

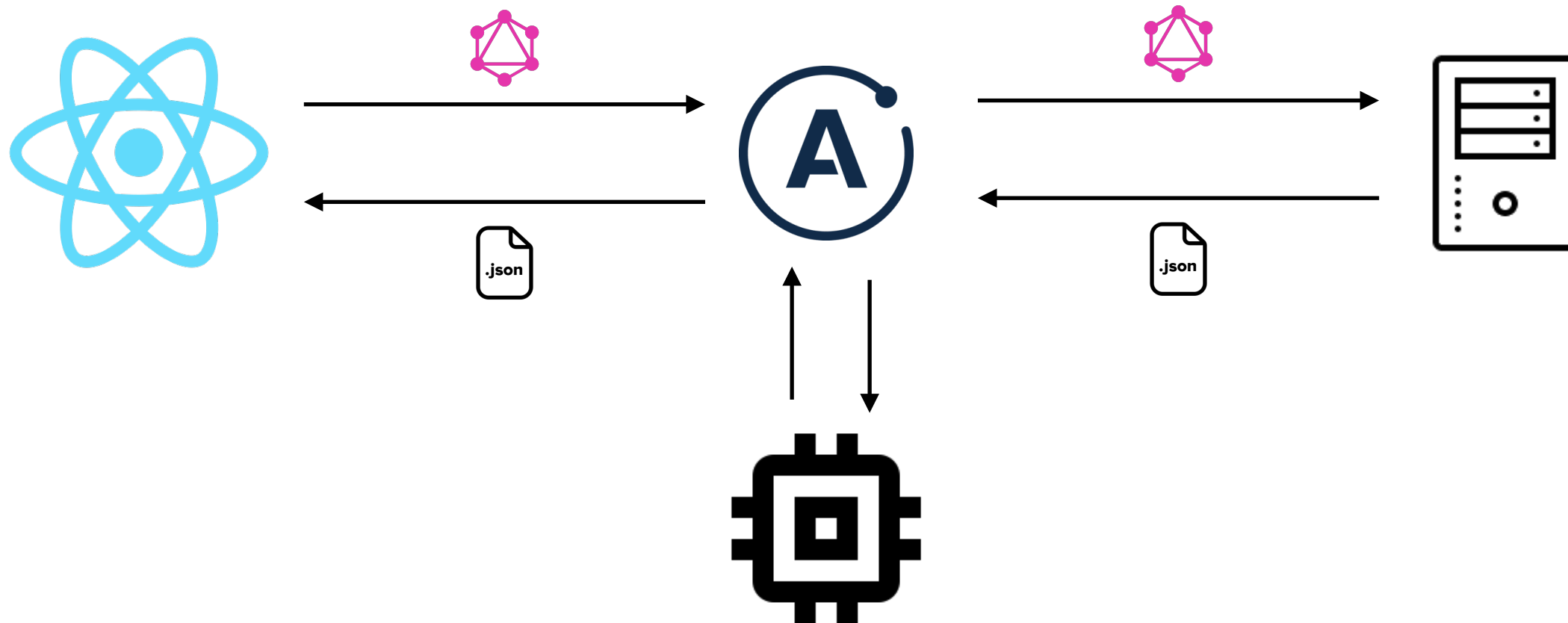
```
type Query {  
  user(id: ID!): User  
  message(id: ID!): Message  
}  
  
type User {  
  id: ID!  
  name: String!  
}  
  
type Message {  
  id: ID!  
  payload: String!  
  createdAt: String!  
  author: User!  
}
```

```
query AwesomeQuery {  
  user(id: "1") {  
    id  
    name  
  }  
  firstMessage: message(id: "0") {  
    id  
    payload  
    createdAt  
    author {  
      id  
      name  
    }  
  }  
}
```

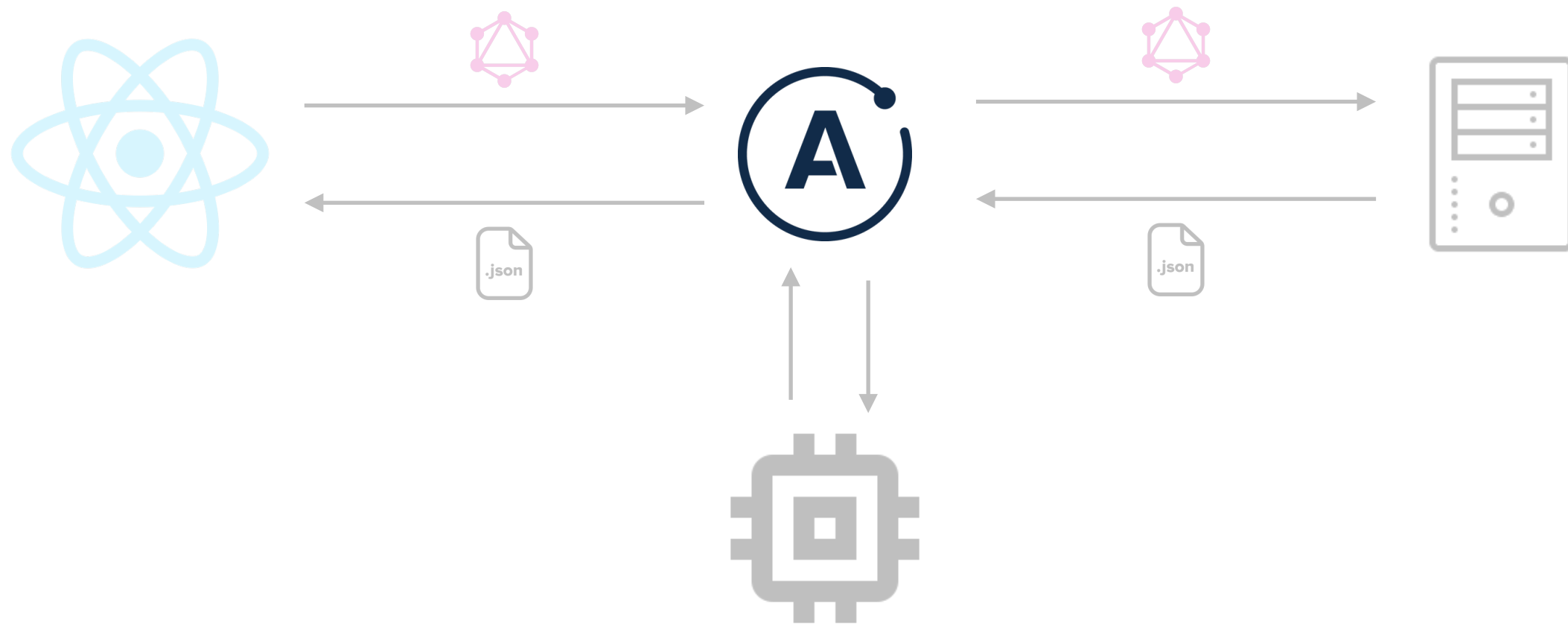
```
{  
  "data": {  
    "user": {  
      "id": "1",  
      "name": "Ian Funk"  
    },  
    "firstMessage": {  
      "id": "0",  
      "payload": "Animi aut aut.",  
      "createdAt": "1557676158000",  
      "author": {  
        "id": "393",  
        "name": "Zelma Runolfsson"  
      }  
    }  
  }  
}
```


Playground!

Apollo?

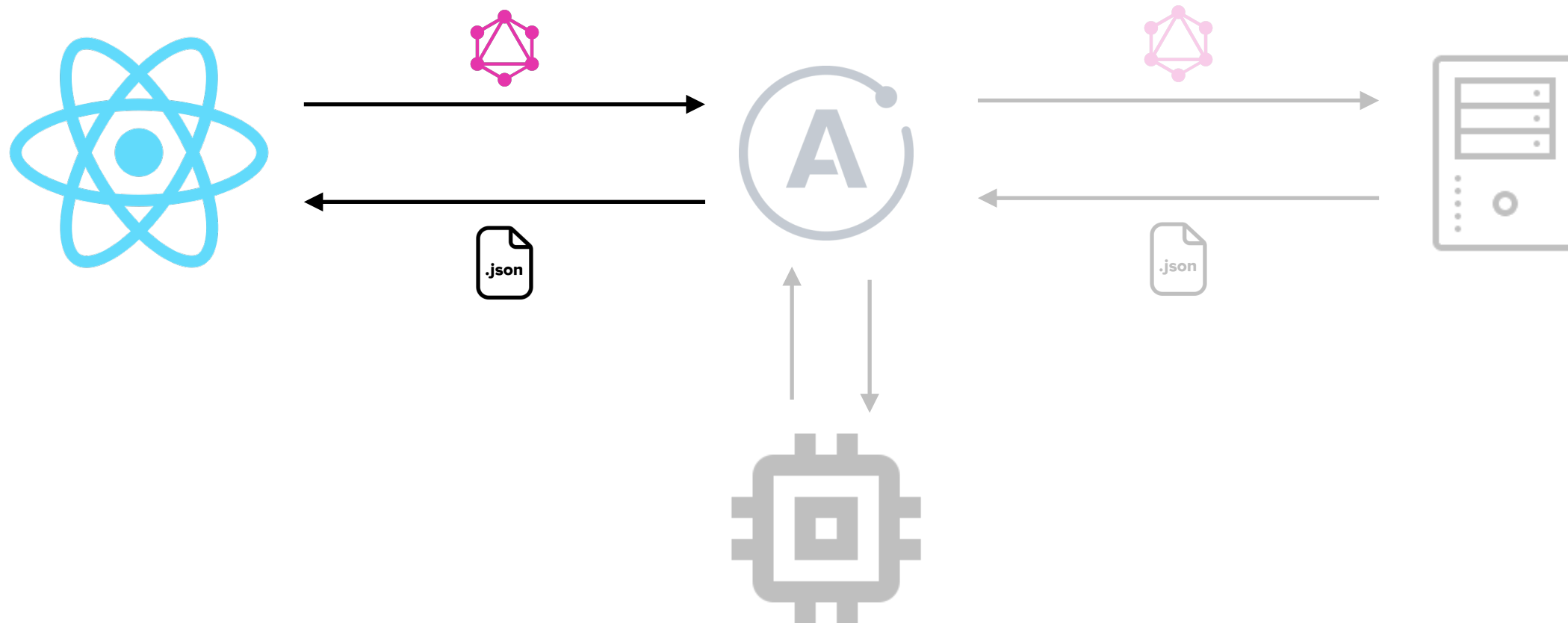


Apollo Client



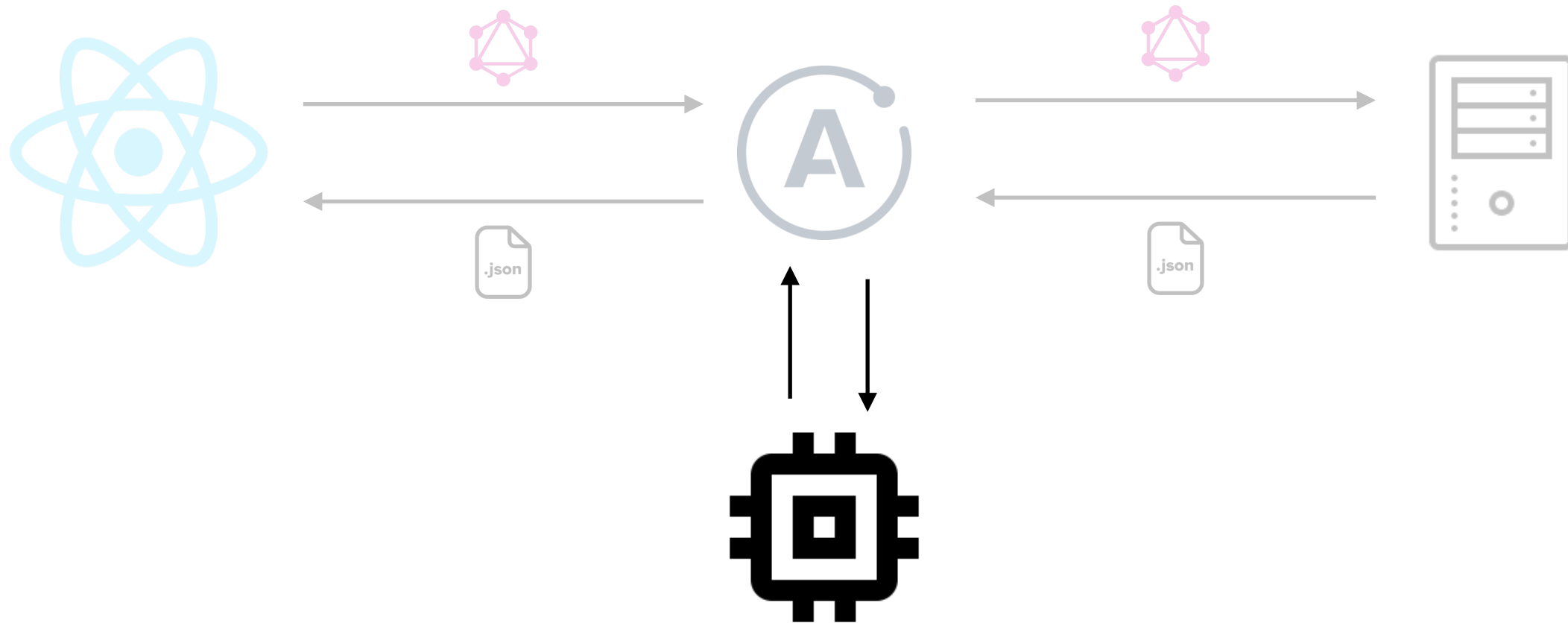
Apollo의 핵심이 되는 부분입니다
여러 기능을 한 곳에 모으고 관리합니다

React Apollo



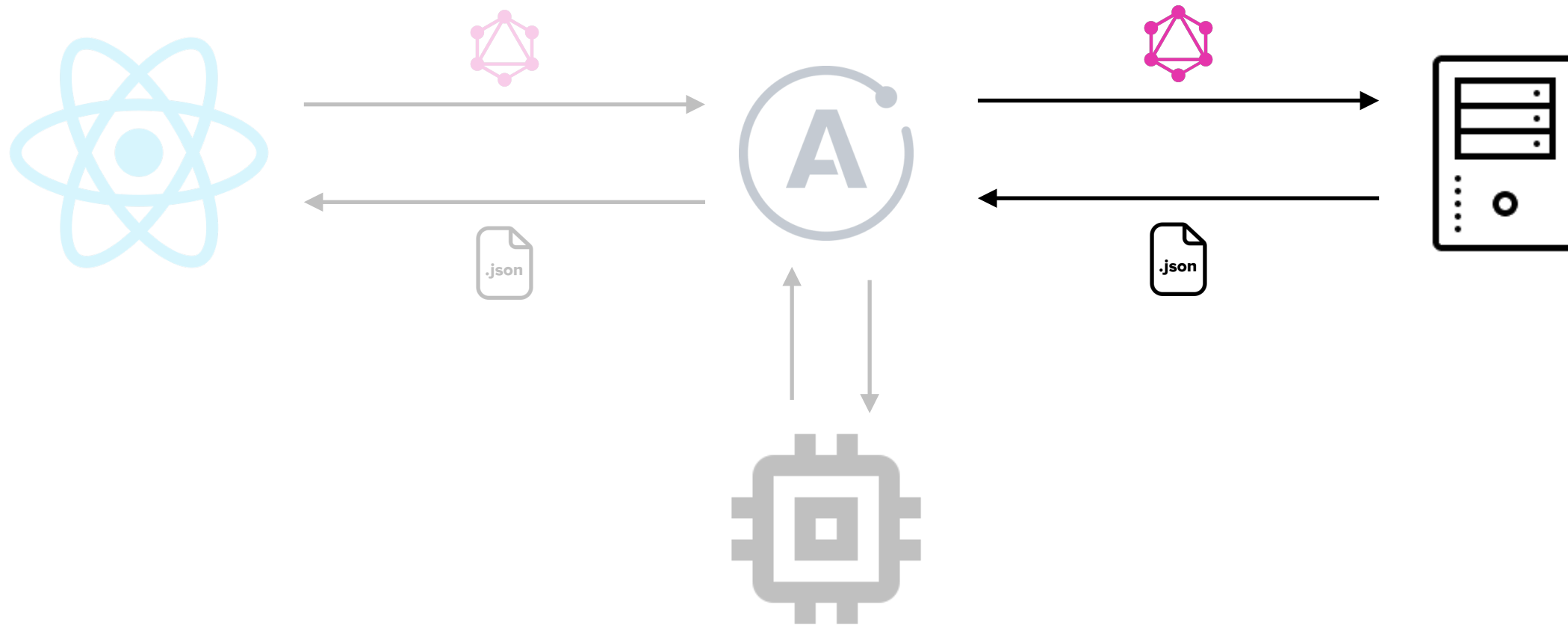
컴포넌트가 사용할 데이터를 GraphQL로 요청하고
JSON의 형태로 데이터를 얻습니다

Apollo Cache



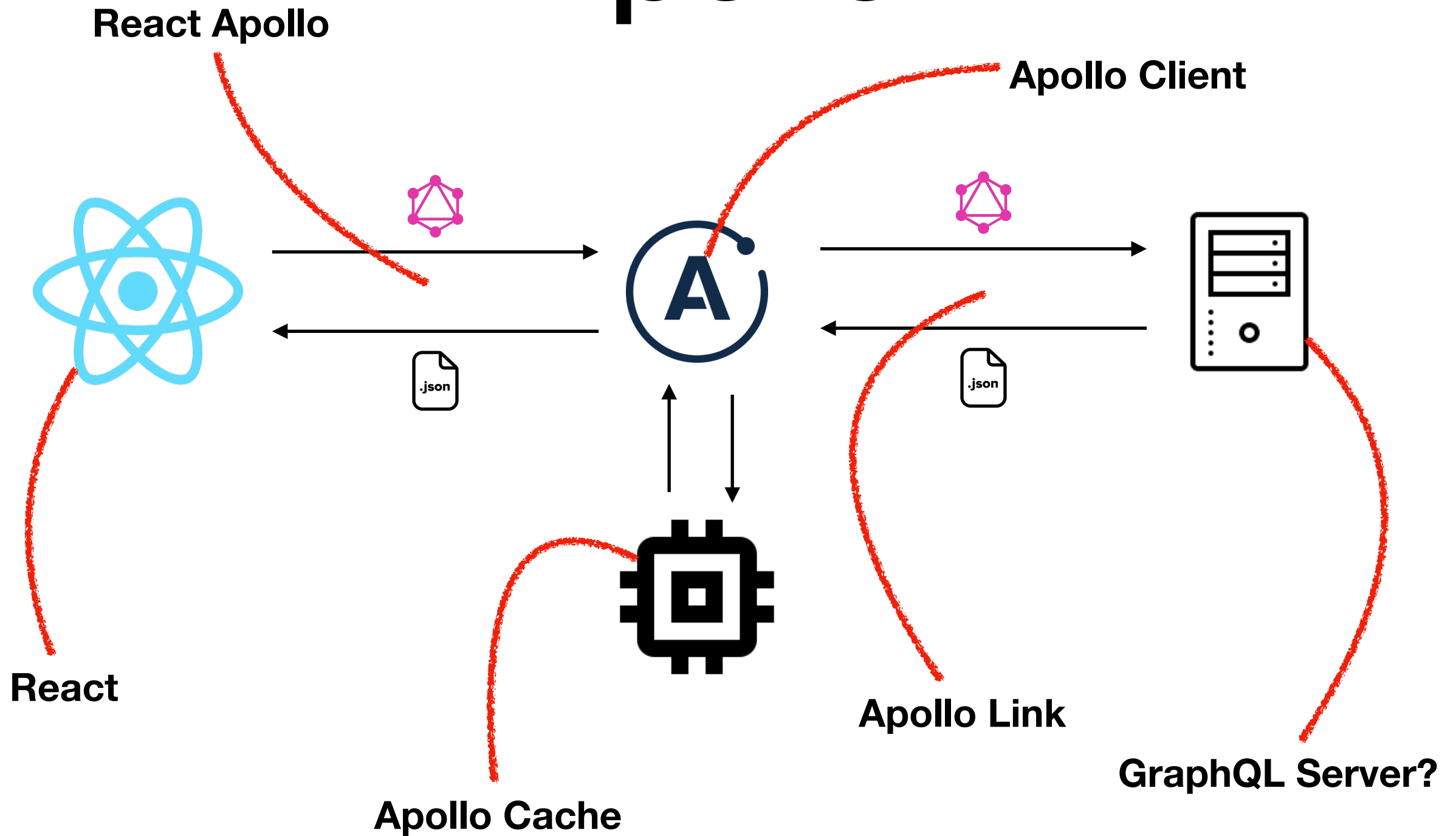
쿼리의 결과를 저장합니다
요청을 보내기 전에 Cache에 필요한 데이터가 있는지 확인하고 재사용합니다

Apollo Link



외부 GraphQL 요청에 대한 인터페이스를 제공합니다
통신 레이어를 추상화하고, 상황에 따라 서버없이 사용 할 수 있습니다

Apollo!



React Apollo Project



```
$ create-react-app <your project> # --typescript
```

```
$ cd <your project>
```

```
$ yarn add graphql apollo-boost react-apollo
```


Apollo Client

src/configureClient.ts

```
import { ApolloClient, InMemoryCache, HttpLink } from 'apollo-boost'

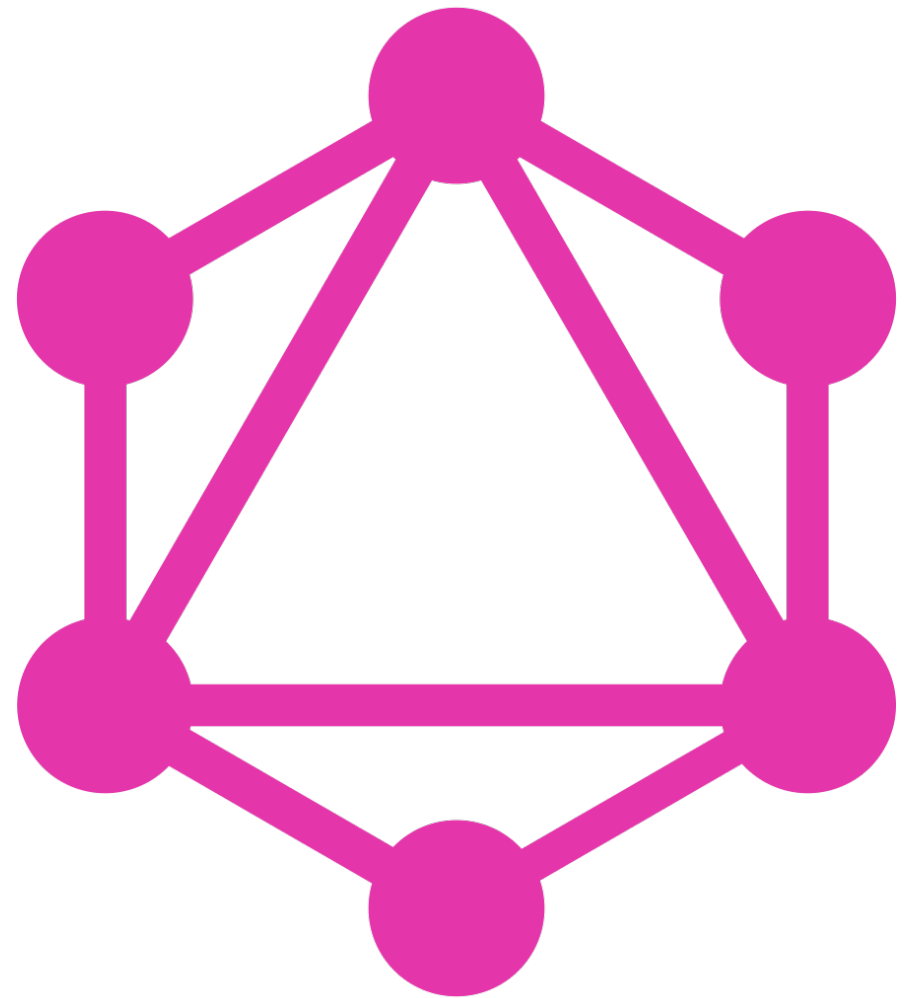
export default function configureClient() {
  // your GraphQL endpoint
  const uri = 'http://localhost:8888/graphql';
  const cache = new InMemoryCache();
  const link = new HttpLink({ uri });
  return new ApolloClient({ cache, link });
}
```



GraphQL in JS

```
import { gql } from 'apollo-boost'

const MY_AWESOME_QUERY = gql`
query AwesomeQuery {
  ping
  user(id: "1") {
    id
    name
  }
  firstMessage: message(id: "0") {
    id
    payload
    createdAt
    author {
      id
      name
    }
  }
}
`;
```

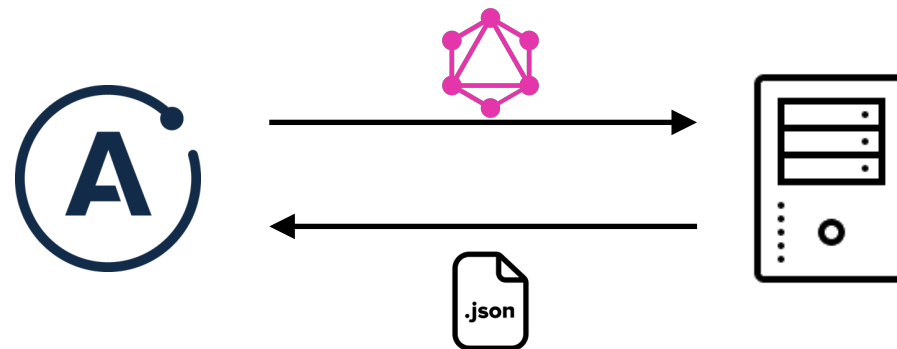


Query

```
import gql from 'graphql-tag'
import configureClient from './configureClient'

const query = gql`
query { ping }
`;

const client = configureClient();
client.query({ query })
  .then(console.log)
  .catch(console.error);
```



Variables

```
import gql from 'graphql-tag'
import configureClient from './configureClient'

const query = gql`
query ($id: ID!) {
  user(id: $id) {
    id
    name
  }
}
`;

const client = configureClient();
client.query({ query, variables: { id: "42" } })
  .then(console.log)
  .catch(console.error);
```

fetchPolicy



```
import gql from 'graphql-tag'
import configureClient from './configureClient'


const query = gql`
query { ping }
`;

const client = configureClient();
client.query({ query, /* fetchPolicy: 'cache-first' */ })
  .then(() => client.query({ query }))
  .then(() => client.query({ query }))
  .catch(console.error);
```

cache-first
cache-and-network
network-only
cache-only
no-cache



fetchPolicy



```
import gql from 'graphql-tag'
import configureClient from './configureClient'

const query = gql`
query { ping }
`

const client = configureClient();
client.query({ query, /* fetchPolicy: 'cache-first' */ })
  .then(() => client.query({ query }))
  .then(() => client.query({ query }))
  .catch(console.error);
```

link

cache

React Apollo

src/index.tsx

```
import React from 'react'
import { ApolloProvider } from 'react-apollo'
import ReactDOM from 'react-dom'
import App from './App'
import configureClient from './configureClient'

const client = configureClient();

const Root: React.FC = () => (
  <ApolloProvider client={client}>
    <App />
  </ApolloProvider>
);

ReactDOM.render(<Root />, document.getElementById('root'));
```

React Apollo를 사용하기 위해
client를 ApolloProvider에 전달해줍니다

Query Component

src/App.tsx

```
import React from 'react'
import { Query } from 'react-apollo'
import gql from 'graphql-tag'

const QUERY = gql`
query { ping }
`

const App: React.FC = () => (
  <Query query={QUERY}>
    ({ { data, loading, error }: any) => {
      if (loading)
        return <p>Loading...</p>;
      else if (error || !data)
        return <p>Error!</p>;
      else
        return <p>{data.ping}</p>
    }}
  </Query>
);

export default App
```

```
import React from 'react'
import { Query } from 'react-apollo'
import gql from 'graphql-tag'

const QUERY = gql`
query ($id: ID!) { user(id: $id) { id name } }
`

const App: React.FC = () => (
  <Query query={QUERY} variables={{ id: "7" }}>
    ({ { data, loading, error }: any) => {
      if (loading)
        return <p>Loading...</p>;
      else if (error || !data)
        return <p>Error!</p>;
      else
        return <p>{data.user.name}</p>;
    }}
  </Query>
);

export default App
```

query, variables, fetchPolicy 등
props를 사용할 수 있습니다

Typed Component

```
import React from 'react'
import { Query } from 'react-apollo'
import gql from 'graphql-tag'

const QUERY = gql`
query ($id: ID!) { user(id: $id) { id name } }
`

type QueryResult = { user: { id: string, name: string } };
type QueryVariables = { id: string };
class MyQuery extends Query<QueryResult, QueryVariables>{ }

const App: React.FC = () => (
  <MyQuery query={QUERY} variables={{ id: "7" }}>
    ({ { data, loading, error } }) => {
      if (loading)
        return <p>Loading...</p>;
      else if (error || !data)
        return <p>Error!</p>;
      else
        return <p>{data.user.name}</p>;
    }
  </MyQuery>
);

export default App
```

Hooks



```
import React from 'react'
import { ApolloProvider } from 'react-apollo-hooks'
import ReactDOM from 'react-dom'
import App from './App'
import configureClient from './configureClient'

const client = configureClient();

const Root: React.FC = () => (
  <ApolloProvider client={client}>
    <App />
  </ApolloProvider>
);

ReactDOM.render(<Root />, document.getElementById('root'));
```

useQuery Hook

```
import React from 'react'
import gql from 'graphql-tag'
import { useQuery } from 'react-apollo-hooks'

const QUERY = gql`
query ($id: ID!) { user(id: $id) { id name } }
`

const App: React.FC = () => {
  const { data, loading, error } =
    useQuery(QUERY, { variables: { id: "7" } });

  if (loading)
    return <p>Loading...</p>;
  else if (error || !data)
    return <p>Error!</p>;
  else
    return <p>{data.user.name}</p>;
};

export default App
```

Typed Hook

```
import React from 'react'
import gql from 'graphql-tag'
import { useQuery } from 'react-apollo-hooks'

const QUERY = gql`
query ($id: ID!) { user(id: $id) { id name } }
`

type QueryResult = { user: { id: string, name: string } };
type QueryVariables = { id: string };

const App: React.FC = () => {
  const { data, loading, error } =
    useQuery<QueryResult, QueryVariables>(QUERY, { variables: { id: "7" } });

  if (loading)
    return <p>Loading...</p>;
  else if (error || !data)
    return <p>Error!</p>;
  else
    return <p>{data.user.name}</p>
};

export default App
```

Persist Cache

```
import { ApolloClient, InMemoryCache, HttpLink } from 'apollo-boost'
import { persistCache } from 'apollo-cache-persist'

export default async function configureClient() {
  // your GraphQL endpoint
  const uri = 'http://localhost:8888/graphql';
  const cache = new InMemoryCache();
  const link = new HttpLink({ uri });

  const storage = window.localStorage as any;
  await persistCache({ cache, storage });

  return new ApolloClient({ cache, link });
}
```

영속적인 저장소에서 데이터를 가져옵니다

Persisted Query Link

```
import { ApolloClient, InMemoryCache, HttpLink } from 'apollo-boost'
import { createPersistedQueryLink } from 'apollo-link-persisted-queries'

export default function configureClient() {
  // your GraphQL endpoint
  const uri = 'http://localhost:8888/graphql';
  const cache = new InMemoryCache();

  const httpLink = new HttpLink({ uri });
  const persistedLink = createPersistedQueryLink()
  const link = persistedLink.concat(httpLink);

  return new ApolloClient({ cache, link });
}
```

GraphQL Query 대신 Hash로 요청합니다
Apollo Link는 여러 링크를 연결해서 사용할 수 있습니다

Persisted Query Link

```
{
  "operationName": null,
  "variables": {
    "id": "7"
  },
  "extensions": {
    "persistedQuery": {
      "version": 1,
      "sha256Hash": "<Your GraphQL Hash>"
    }
  }
}
```

Hash Request

```
{
  "operationName": null,
  "variables": {
    "id": "7"
  },
  "extensions": {
    "persistedQuery": {
      "version": 1,
      "sha256Hash": "<Your GraphQL Hash>"
    }
  },
  "query": "<Your GraphQL Code>"
}
```

Hash Request Fallback

Batch HTTP Link

```
import { ApolloClient, InMemoryCache } from 'apollo-boost'
import { createPersistedQueryLink } from 'apollo-link-persisted-queries'
import { BatchHttpLink } from 'apollo-link-batch-http'

export default function configureClient() {
  // your GraphQL endpoint
  const uri = 'http://localhost:8888/graphql';
  const cache = new InMemoryCache();

  const batchHttpLink = new BatchHttpLink({ uri })
  const persistedLink = createPersistedQueryLink()
  const link = persistedLink.concat(batchHttpLink)

  return new ApolloClient({ cache, link });
}
```

여러 GraphQL 요청을 한 번에 처리합니다
서버와의 통신 횟수를 줄입니다

Batch HTTP Link



```
{ "operationName": null, "variables": { "id": "0" }, "extensions": { } },  
{ "operationName": null, "variables": { "id": "1" }, "extensions": { } },  
{ "operationName": null, "variables": { "id": "2" }, "extensions": { } },  
{ "operationName": null, "variables": { "id": "3" }, "extensions": { } },  
{ "operationName": null, "variables": { "id": "4" }, "extensions": { } },  
{ "operationName": null, "variables": { "id": "5" }, "extensions": { } }
```

Many Requests (HttpLink)



```
[  
  { "operationName": null, "variables": { "id": "0" }, "extensions": { } },  
  { "operationName": null, "variables": { "id": "1" }, "extensions": { } },  
  { "operationName": null, "variables": { "id": "2" }, "extensions": { } },  
  { "operationName": null, "variables": { "id": "3" }, "extensions": { } },  
  { "operationName": null, "variables": { "id": "4" }, "extensions": { } },  
  { "operationName": null, "variables": { "id": "5" }, "extensions": { } }  
]
```

One Request (BatchHttpLink)

Batch HTTP Link

- 느린 Query가 전체에 영향을 줍니다
- 네트워크 수준 디버깅이 불편합니다
- CDN에서 캐시하기 어렵습니다
- HTTP2를 지원하면 의미가 없습니다

