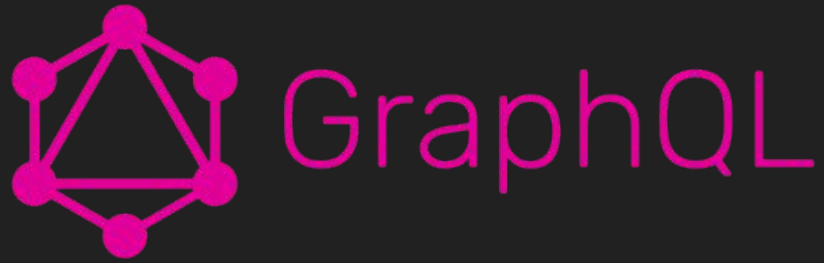
Automatic Persisted Queries: why does it matter?

@aymericbeaumet



Open-source **Q**uery and manipulation **L**anguage

- *2012* Internal release at Facebook
- *2015* Public release
- *2018* GraphQL foundation

Public APIs: GitHub, Shopify, Yelp...

Clients/servers: Node.js, Go, Ruby, Python, Java...

Everything is a trade-off.

Benefits

- Prevent under/overfetching
- Give control back to the clients
- React integration

Drawbacks

- Not yet widely adopted
- Rate-limiting
- Caching
- **Expensive** for the uplink bandwidth (exhaustivity)

Expensive you said?

GitHub v3 (REST) *vs* GitHub v4 (GraphQL)

Hey GitHub, give me all the repositories for this specific user!

GitHub API v3 (REST)

```
GET /users/aymericbeaumet/repos
```

```
curl -s -o /dev/null -w 'Request: %{size_request}B + Payload: %{size_upload}B'
'https://api.github.com/users/aymericbeaumet/repos'
```

Request: 104B + Payload: 0B

Total: 104B

GitHub API v4 (GraphQL)

`POST /graphql`

```
{
  user(login: "aymericbeaumet") {
    repositories(privacy: PUBLIC, last: 20) {
      nodes {
        id
        name
        url
        description
        isFork
        owner {
          id
          # ...
        }
        # ...
      }
    }
  }
}
```

```
$ curl -s -o /dev/null -w 'Request: %{size_request}B + Payload: %{size_upload}B'
'https://graphql-explorer.githubapp.com/graphql/proxy' -H 'content-type:
application/json' --data '{"query":"{\n  user(login: \"aymericbeaumet\") {\n
repositories(privacy: PUBLIC, last: 20) {\n    nodes {\n      id\n      name\n
url\n    description\n      isFork\n      owner {\n          id\n          # ...\n
    }\n      # ...\n  }\n  }\n  }\n}\n","variables":{},"operationName":null}'
```

Request: 485B + Payload: 324B

Total: 809B (~8x)

Uplink footprint matters

REST

- Specialized approach
- Lightweight requests

GraphQL

- Generic approach
- Heavier requests

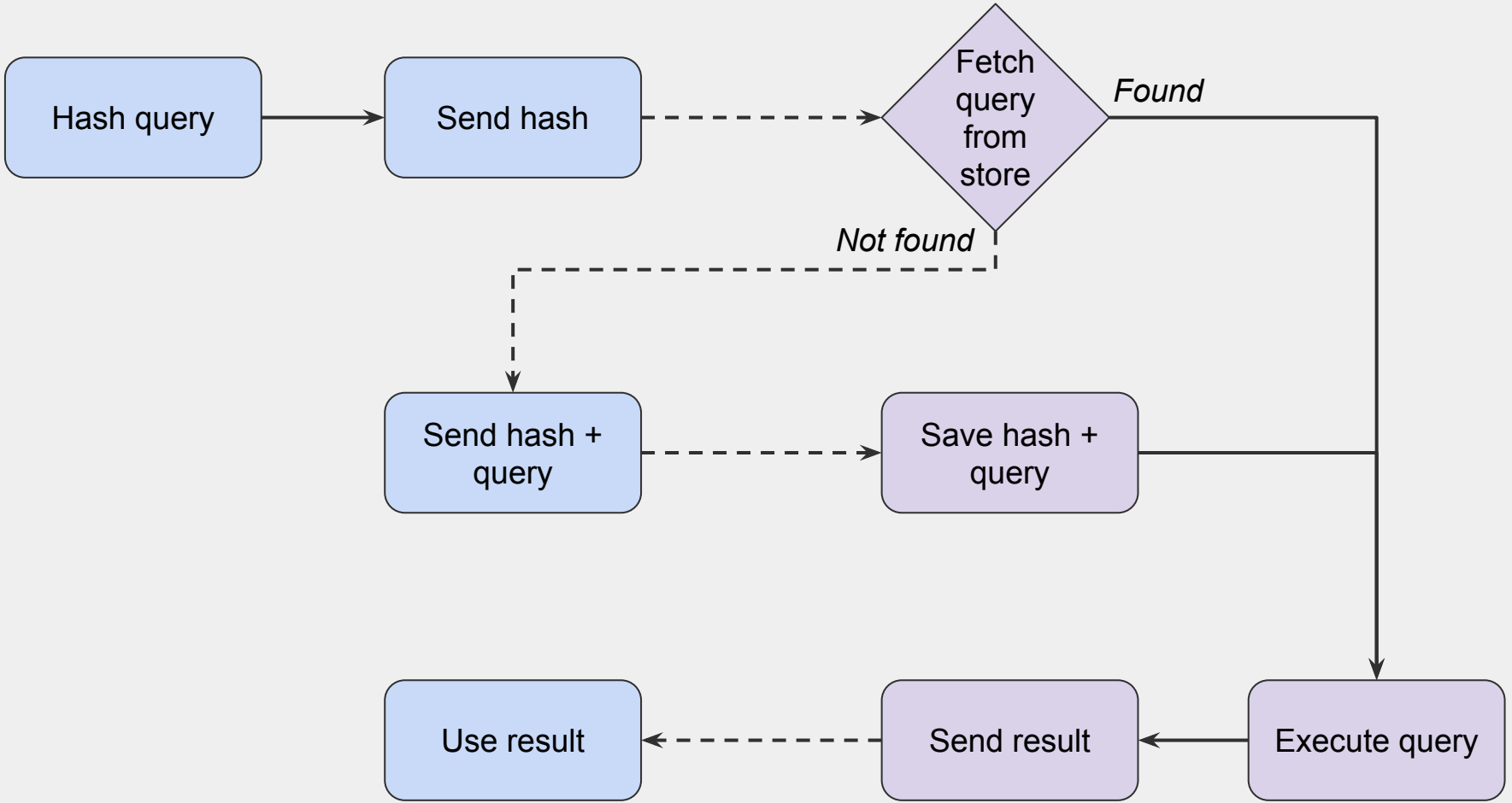
We want the *flexibility* of GraphQL
and the *lightweightness* of REST.

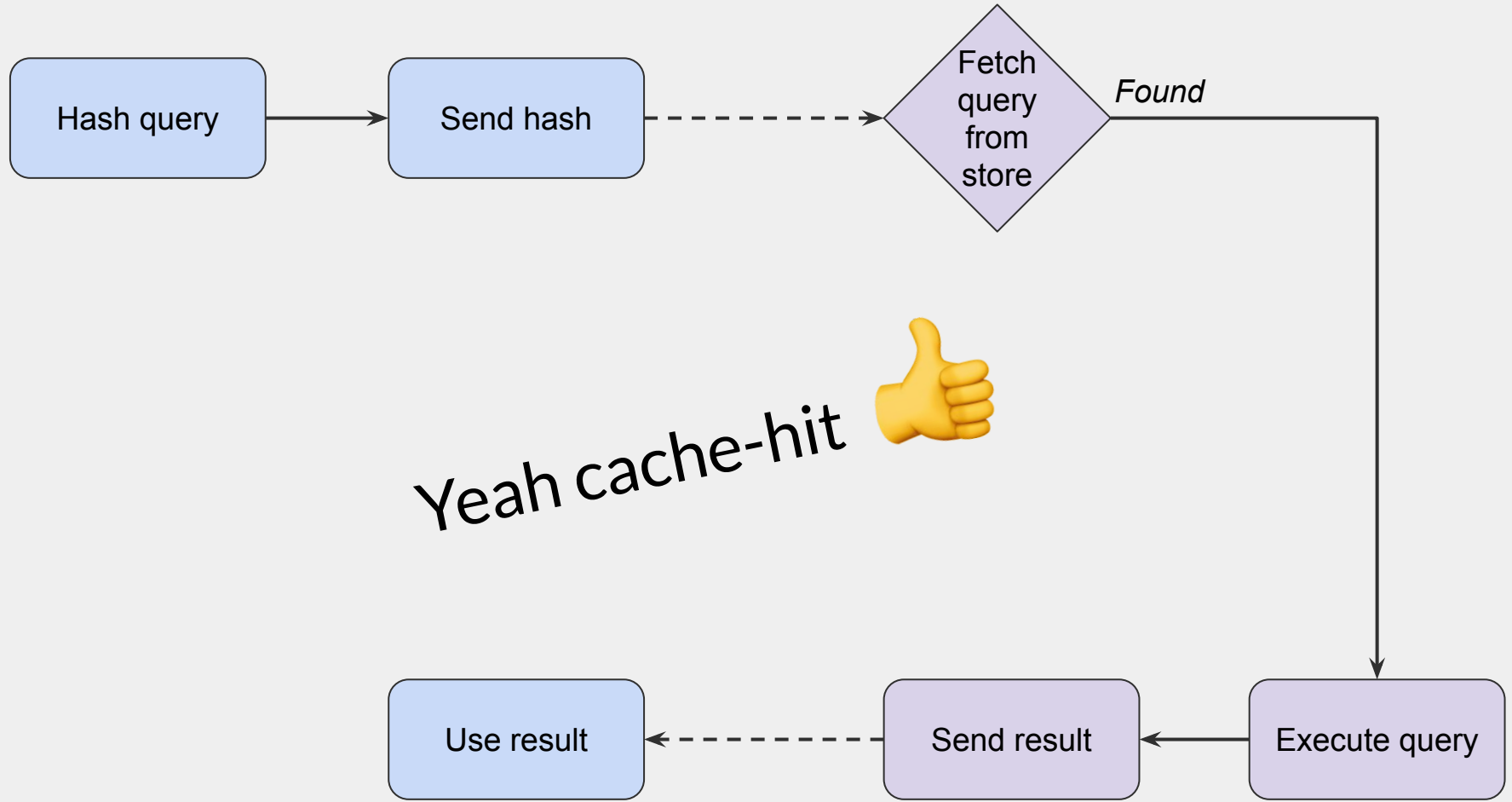
Automatic Persisted Queries

Instead of sending the query *each* time.

Hash it and *cache* it server-side.

Then, optimistically *leverage* the hash.


Hash query
Send hash
Fetch query from store
Found
Not found
Send hash + query
Save hash + query
Use result
Send result
Execute query


Hash query
Send hash
Fetch query from store
Found
Yeah cache-hit
Use result
Send result
Execute query

apollo-link-persisted-queries

```
import { createPersistedQueryLink } from "apollo-link-persisted-queries";
import { createHttpLink } from "apollo-link-http";
import { InMemoryCache } from "apollo-cache-inmemory";
import ApolloClient from "apollo-client";

const link = createPersistedQueryLink().concat(createHttpLink({ uri: "/graphql" }));
const client = new ApolloClient({
  cache: new InMemoryCache(),
  link: link,
});
```

Conclusion

- Bandwidth matters
- APQ reduces your uplink footprint
- Slower latency for the first request (across all your clients)
- Easy to implement with Apollo
- Give it a try!

Questions?