

RELAY AT UNICORN SCALE

Lessons from Productboard

FE Platform

The team selected and provided tooling

2.5 years

Serving as architect



productboard

Full rewrite of thick web client to thin with GraphQL & Relay



01

**QUICK INTRO
INTO RELAY**

02

**WINS & CHALLENGES
WE SAW**

03

**WHO SHOULD USE
RELAY?**

QUICK INTRO INTO RELAY

01

CREATED BY FACEBOOK

SAME TEAM THAT CREATED GRAPHQL

DEEPLY OPINIONATED

‘BUILT FOR SCALE’

‘GRAPHQL BEST PRACTICES BAKED IN’

Relay

The GraphQL client that scales with you.

Built for scale

Relay is designed for high performance at any scale. Relay keeps management of data-fetching easy, whether your app has tens, hundreds, or thousands of components. And thanks to Relay's incremental compiler, it keeps your iteration speed fast even as your app grows.

Keeps iteration quick

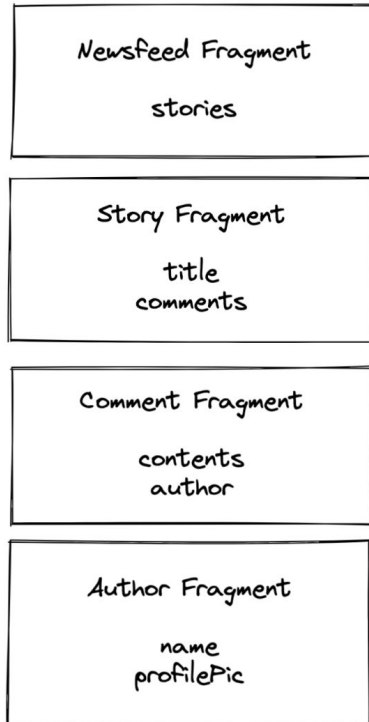
Relay is data-fetching turned **declarative**. Components declare their data dependencies, without worrying about how to fetch them. Relay guarantees that the data each component needs is fetched and available. This keeps components decoupled and promotes reuse.

With Relay, components and their data dependencies can be quickly modified without modifying other parts of the system. That means you won't accidentally break other components as you refactor or make changes to your app.

Automatic optimizations

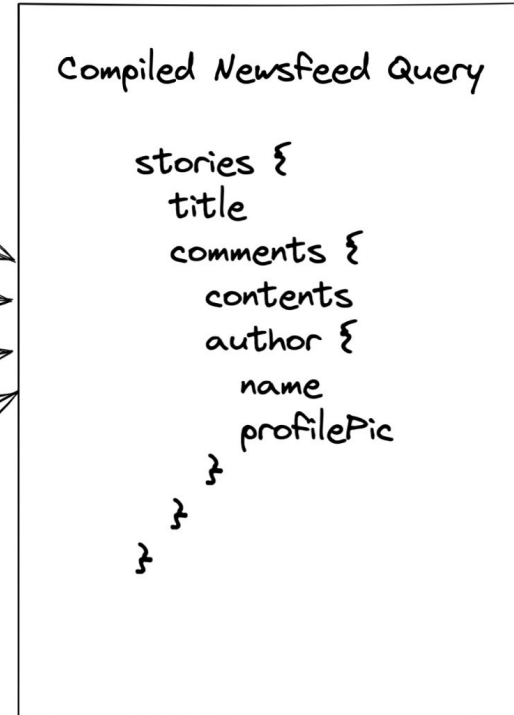
COLOCATION

Components declare their data dependencies



QUERY AGGREGATION

One request per page and other optimisations



DEMO

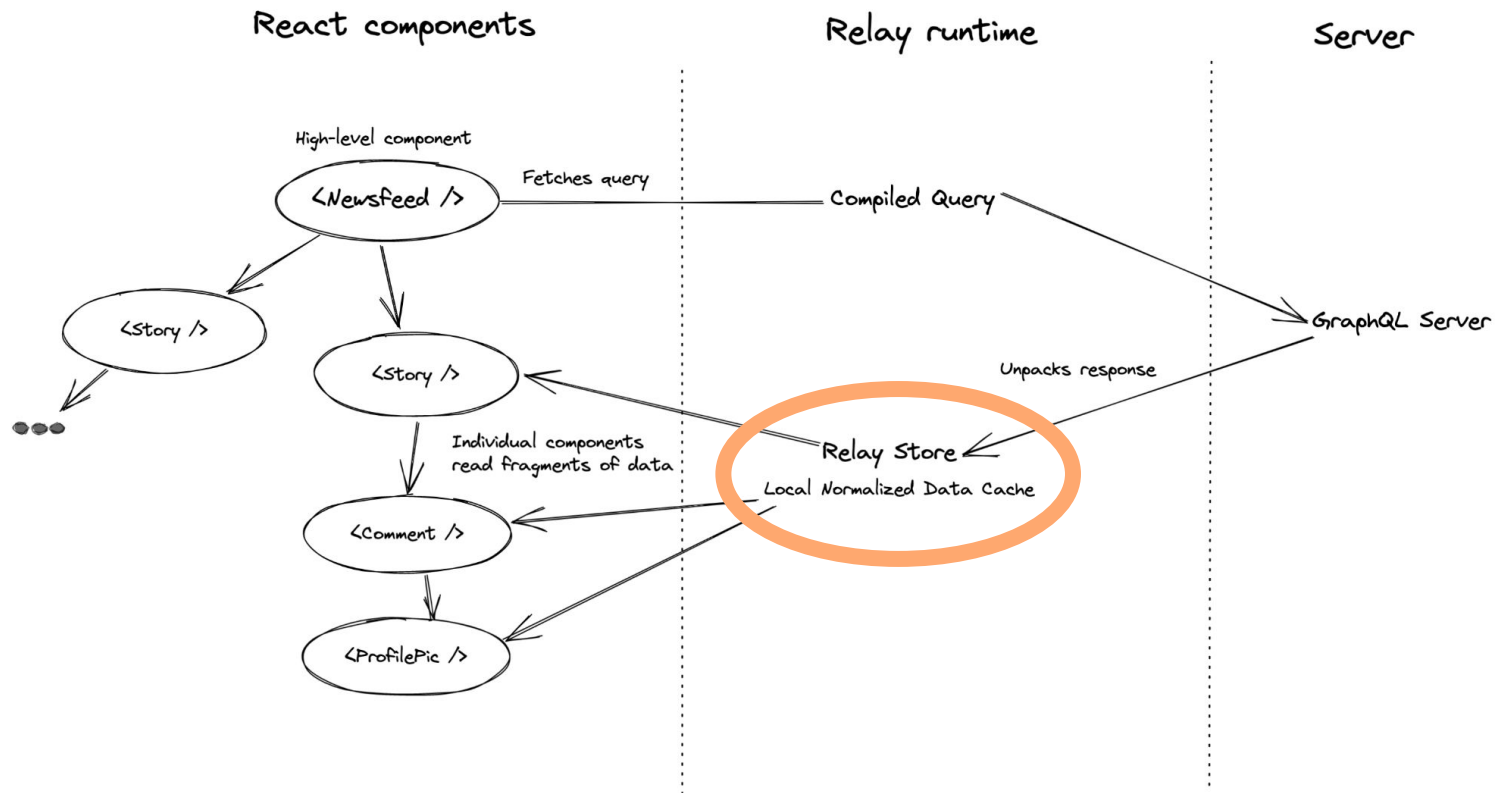
FOR FREE

Automatic request
deduplication

Built-in pagination
via Connections

Optimistic updates

THE MAGIC



NODE INTERFACE

Global Object Identification Spec

[facebook.com/10235697663175551](https://www.facebook.com/10235697663175551) → Photo

[facebook.com/1452660028278850](https://www.facebook.com/1452660028278850) → Group

Relay Store = One source of truth

Update once → reflects everywhere

```
interface Node {  
  id: ID!  
}  
  
type Story implements Node  
{  
  id: ID!  
  title: String!  
}  
  
type Query {  
  node(id: ID!): Node  
}
```

WINS & CHALLENGES WE SAW

02

DEMO

WINS

TYPE SAFETY END-TO-END

Catch breaking changes at build time. Full autocomplete in IDE. No manual type definitions.

ENUMS & FUTURE-PROOFING

Schema evolution without breaking clients

REFETCHABLE FRAGMENTS

Initial page load: one aggregated query. Need fresh data? Refetch just this fragment.

PAGINATION WITH @CONNECTION

UsePaginationFragment hook handles everything for standard schema

SUSPENSE INTEGRATION

React 18 Suspense works out of the box

OPTIMISTIC RESPONSES

UI updates instantly, rollback on error

CHALLENGES

DOCUMENTATION & COMMUNITY

Tutorial often outdated
Small community = fewer answers
Hard to Google solutions
Diving in source code was common

IMPACT ON ARCHITECTURE

Architecturally significant decision
Dictates best practises for schema
Node query needs ALL types
Federation + Node query = pain

INITIAL INVESTMENT

Shallow (!) learning curve
Server engineers needs to know it
Requires solid React knowledge
Tooling takes time to set up

**WHO SHOULD USE
RELAY?**

03

USE

Advanced data model

Large scale application

Need polished,
consistent UX

Team can invest in
learning & culture

SKIP

Simple CRUD

Independent pages

Unwilling to accept the
Relay way

Can't invest in learning
curve

RELAY IS FERRARI-LEVEL ENGINEERING

Not every team is ready. More should try it.



LET'S CONNECT

