React Server Components vs. GraphQL

Jordan Eldredge — Meta — May 2026

About Me

Jordan Eldredge

- Nine years at Facebook/Meta
- Six years working on Relay
- Meta's GraphQL client for web and React Native

React Server Components

Compose?

Compete?

How? When?

What are React Server Components (RSC)?

React components that run on the server

```
async function UserName({ id }) {
  const user = await User.find(id);
  return <ClientButton>
    {user.fullName()}
  </ClientButton>;
}
```

What are React Server Components (RSC)?

React components that run on the server

Server →

```
async function UserName({ id }) {
  const user = await User.find(id);
  return <ClientButton>
Server →    {user.fullName()}
  </ClientButton>;
}
```

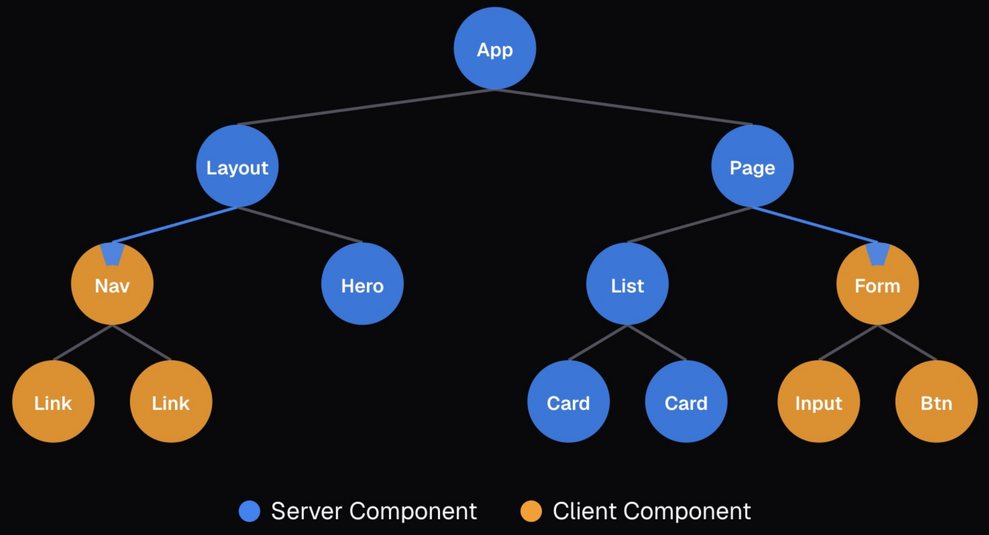
What are React Server Components (RSC)?

React components that run on the server

```
async function UserName({ id }) {
  const user = await User.find(id);
  return <ClientButton>                    ← Client
    {user.fullName()}
  </ClientButton>;                         ← Client
}
```

Server & Client Components

A single tree, split across two runtimes



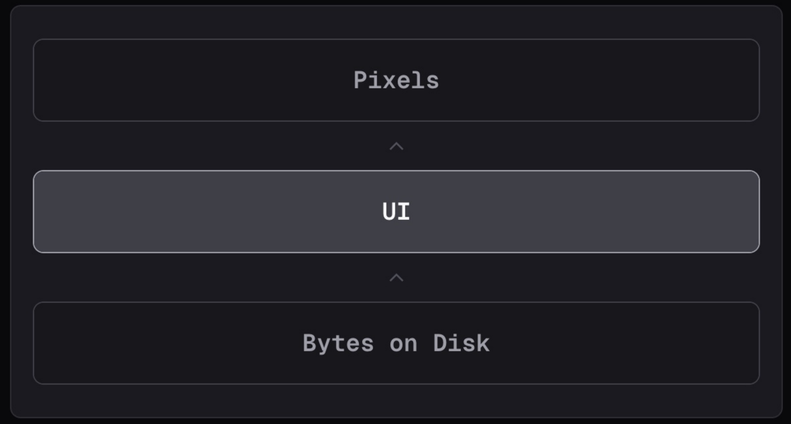
Forget Everything You Know

We're going to derive UI architecture from scratch

What even is UI?

pixels = f(disk)

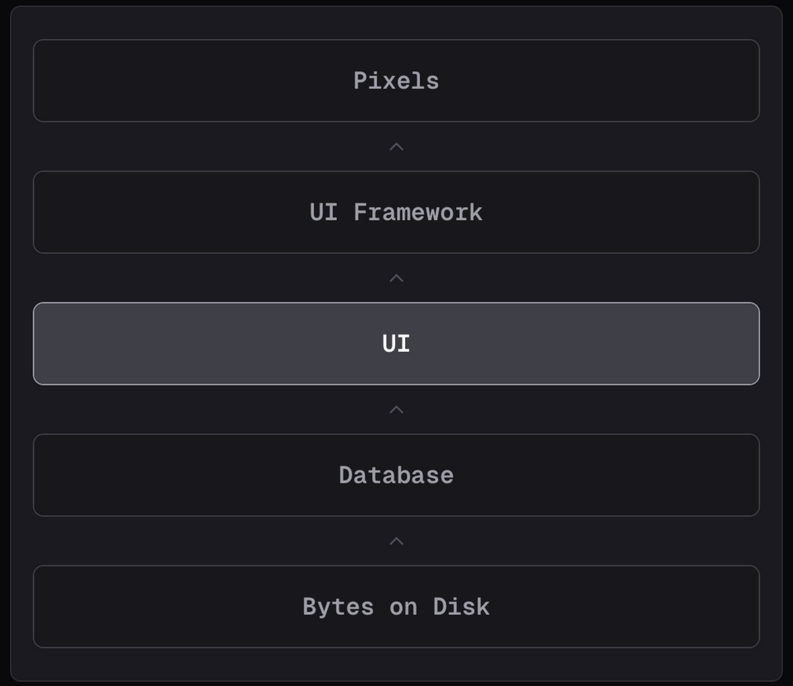
```
function App() {
  const data = readFile("./data");
  //... application logic
  return arrayOfPixels;
}
```



Database and UI Framework

Abstracting storage and presentation

```
function App() {
  const user = db.query(
    "SELECT * FROM users WHERE id = 4"
  );
  return <h1>
    {user.first_name} {user.last_name}
  </h1>;
}
```

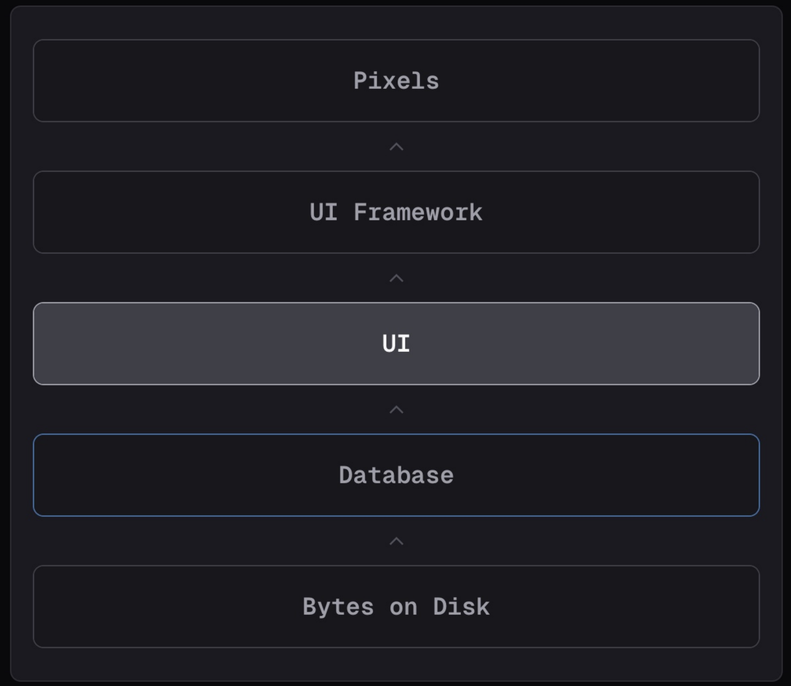


Database and UI Framework

Abstracting storage and presentation

Database →

```
function App() {
  const user = db.query(
    "SELECT * FROM users WHERE id = 4"
  );
  return <h1>
    {user.first_name} {user.last_name}
  </h1>;
}
```



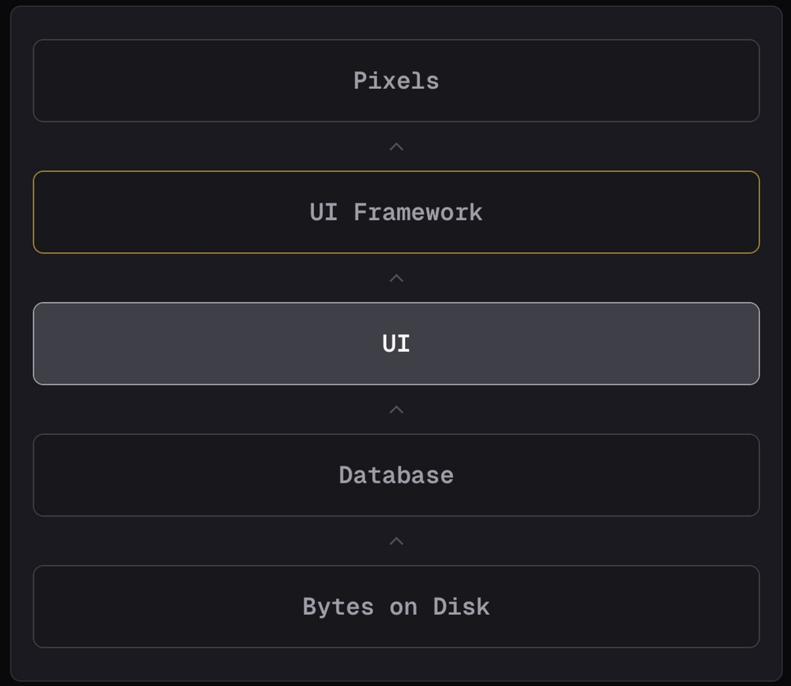
Database and UI Framework

Abstracting storage and presentation

```
function App() {
  const user = db.query(
    "SELECT * FROM users WHERE id = 4"
  );
  return <h1>
    {user.first_name} {user.last_name}
  </h1>;
}
```

UI Framework →

UI Framework →

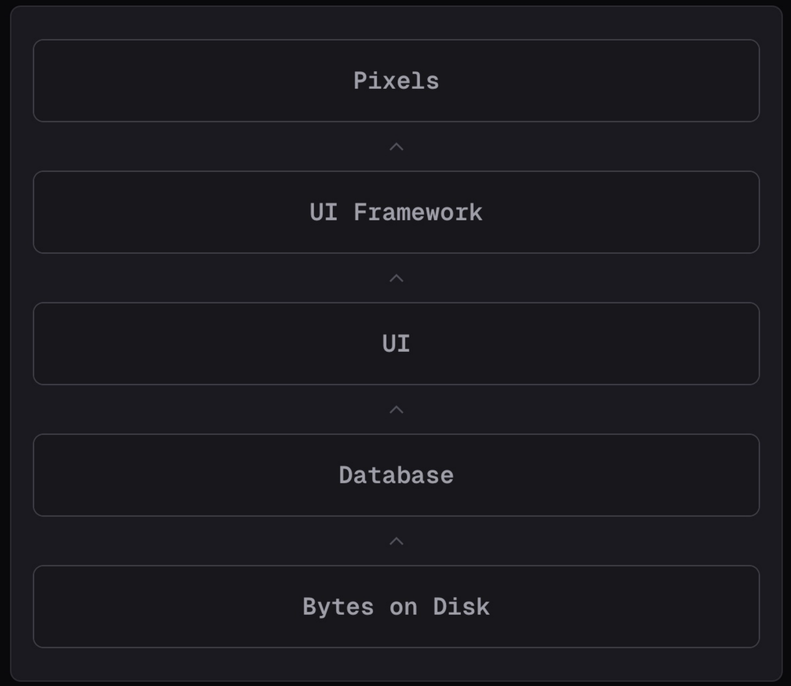


Database and UI Framework

Abstracting storage and presentation

Business logic? →

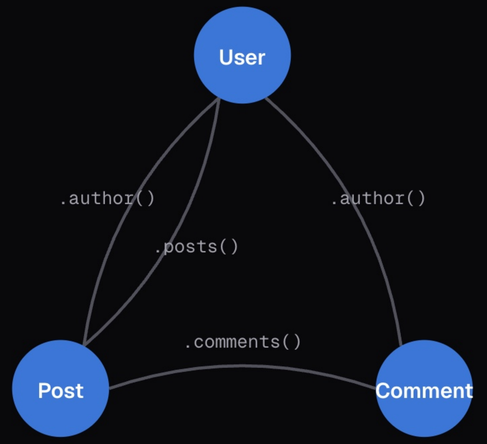
```
function App() {
  const user = db.query(
    "SELECT * FROM users WHERE id = 4"
  );
  return <h1>
    {user.first_name} {user.last_name}
  </h1>;
}
```



Models

Defining the mental model

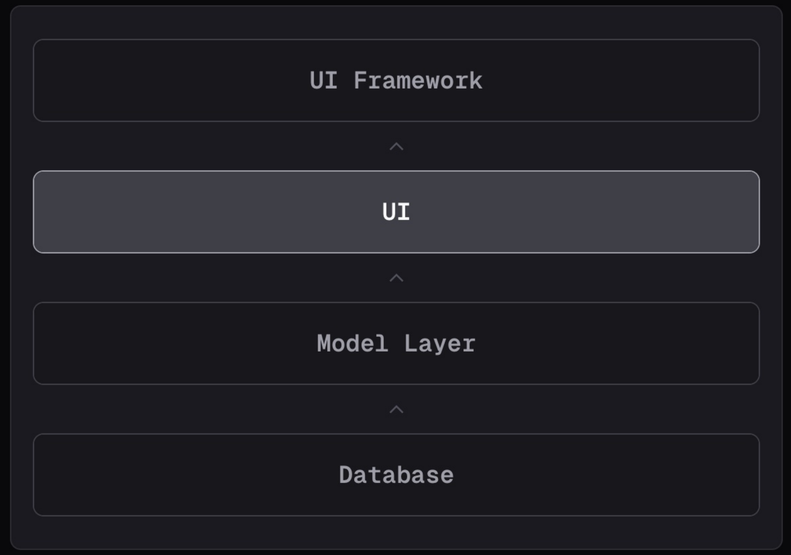
```
class User {
  static find(id): User {.. }
  fullName(): string {.. }
  posts(): Post[] {
    //... complex post visibility logic
  }
}
```



The Platonic Ideal

UI = f(Models)

```
function App() {
  const user = User.find(4);
  return <h1>
    {user.fullName()}
  </h1>;
}
```



The problem with phone apps

Native Mobile Apps

Client-server tensions

Platonic Ideal

```
function App() {
  const user = User.find(4);
  return <h1>
    {user.fullName()}
  </h1>;
}
```

Native Mobile

```
function App() {
  const user = User.find(4);
  return <ClientTitle>
    {user.fullName()}
  </ClientTitle>;
}
```

Native Mobile Apps

Client-server tensions

Platonic Ideal

```
function App() {
  const user = User.find(4);
  return <h1>
    {user.fullName()}
  </h1>;
}
```

Native Mobile

```
function App() {
  const user = User.find(4);
  return <ClientTitle>
    {user.fullName()}
  </ClientTitle>;
}
```

Near database (server) →

Near database (server) →

Native Mobile Apps

Client-server tensions

Platonic Ideal

```
function App() {
  const user = User.find(4);
  return <h1>
    {user.fullName()}
  </h1>;
}
```

Native Mobile

```
function App() {
  const user = User.find(4);
  return <ClientTitle>
    {user.fullName()}
  </ClientTitle>;
}
```

← Compiled into app (client)

← Compiled into app (client)

Enter GraphQL

Model layer on the client

Platonic Ideal

```
function App() {

  const user = User.find(4);

  return <h1>
    {user.fullName()}
  </h1>;
}
```

GraphQL

```
function App() {
  const { user } = useQuery(graphql`
    query { user(id: 4) { fullName } }
  `);
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

Enter GraphQL

Model layer on the client

Platonic Ideal

```
function App() {

  const user = User.find(4);

  return <h1>
    {user.fullName()}
  </h1>;
}
```

Server →

GraphQL

```
function App() {
  const { user } = useQuery(graphql`
    query { user(id: 4) { fullName } }
  `);
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

Enter GraphQL

Model layer on the client

Platonic Ideal

```
function App() {

  const user = User.find(4);

  return <h1>
    {user.fullName()}
  </h1>;
}
```

GraphQL

```
function App() {
  const { user } = useQuery(graphql`
    query { user(id: 4) { fullName } }
  `);
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

← Client

Enter GraphQL

Model layer on the client

Platonic Ideal

```
function App() {

  const user = User.find(4);

  return <h1>
    {user.fullName()}
  </h1>;
}
```

GraphQL

```
function App() {
  const { user } = useQuery(graphql`
    query { user(id: 4) { fullName } }
  `);
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

Enter GraphQL

Model layer on the client

Platonic Ideal

```
function App() {

  const user = User.find(4);

  return <h1>
    {user.fullName()}
  </h1>;
}
```

GraphQL

```
function App() {
  const { user } = useQuery(graphql`
    query { user(id: 4) { fullName } }
  `);
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

- UI can access model layer

Enter GraphQL

Model layer on the client

Platonic Ideal

```
function App() {

  const user = User.find(4);

  return <h1>
    {user.fullName()}
  </h1>;
}
```

GraphQL

```
function App() {
  const { user } = useQuery(graphql`
    query { user(id: 4) { fullName } }
  `);
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

- UI can access model layer
- Cross-language type safety

Enter GraphQL

Model layer on the client

Platonic Ideal

```
function App() {

  const user = User.find(4);

  return <h1>
    {user.fullName()}
  </h1>;
}
```

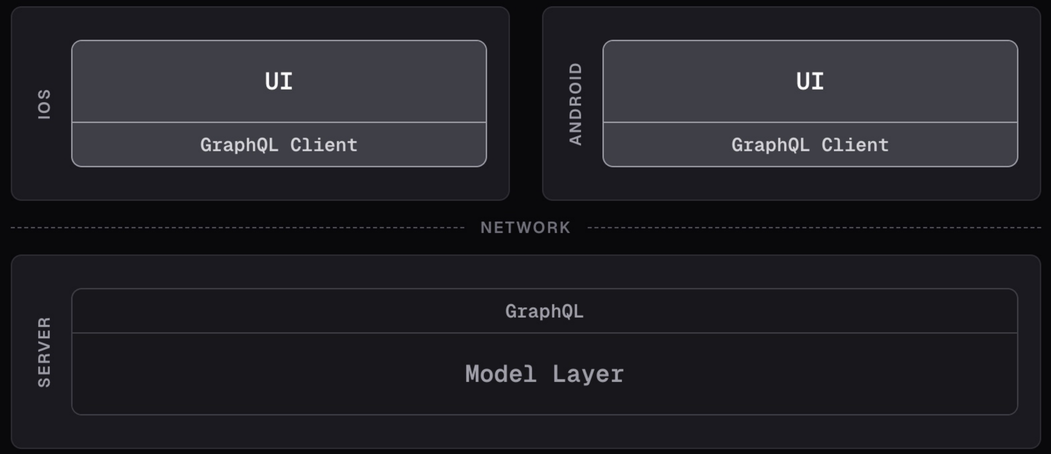
GraphQL

```
function App() {
  const { user } = useQuery(graphql`
    query { user(id: 4) { fullName } }
  `);
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

- UI can access model layer
- Cross-language type safety
- Avoid network waterfalls

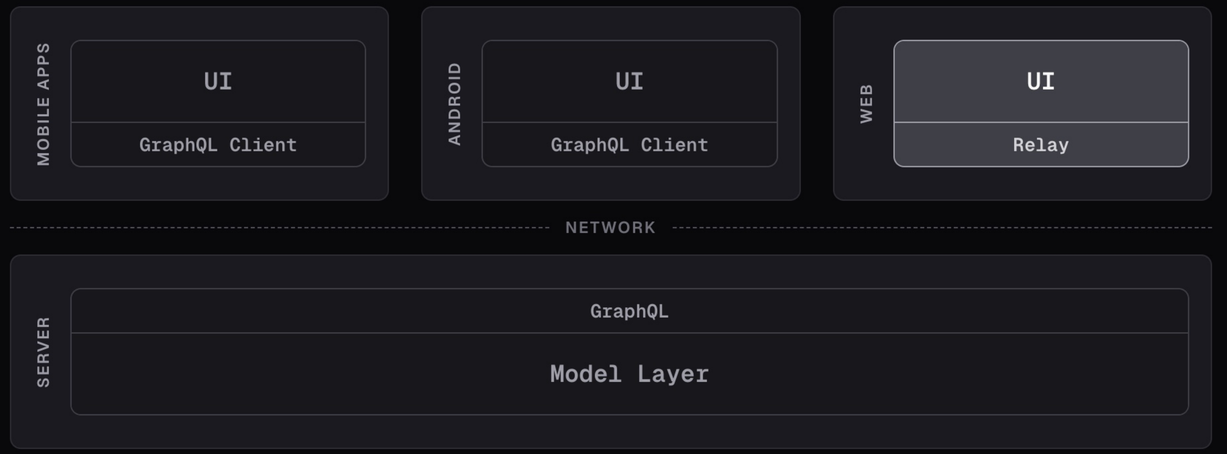
Multiple Clients

The business logic in the model layer is shared by all clients



Single Page Applications (SPAs)

Bringing the native client architecture to the web



Two Architectures

Neither is ideal

Server-Rendered HTML

```
function App() {

  const user = User.find(4);

  return <h1>
    {user.fullName()}
  </h1>;
}
```

- ✓ Avoids heavy JS (download/parse/execute)
- ✗ Interactivity requires switching languages

Single Page Application

```
function App() {
  const { user } = useQuery(graphql`
    query { user(id: 4) { fullName } }
  `);
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

- ✓ Easy to make interactive
- ✗ Challenging to avoid bloated JS
- ✗ Computation on the server requires switching languages

React Server Components

The best of both architectures

Platonic Ideal

```
async function App() {
  const user = await User.find(4);
  return <ClientTitle>
    {user.fullName()}
  </ClientTitle>;
}
```

GraphQL vs. RSC

Tradeoffs for the Web

GraphQL SPA

```
function App() {
  const { user } = useQuery(graphql`
    query { user(id: 4) { fullName } }
  `);
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

React Server Components

```
async function App() {

  const user = await User.find(4);

  return <ClientTitle>
    {user.fullName()}
  </ClientTitle>;
}
```

GraphQL vs. RSC

Tradeoffs for the Web

GraphQL SPA

```
function App() {
  const { user } = useQuery(graphql`
    query { user(id: 4) { fullName } }
  `);
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

- ✅ UI can access model layer
- ✅ Avoid network waterfalls
- ⚠️ Domain-specific language
- ⚠️ Heavy bundle cost
- ⚠️ Awkward to move compute

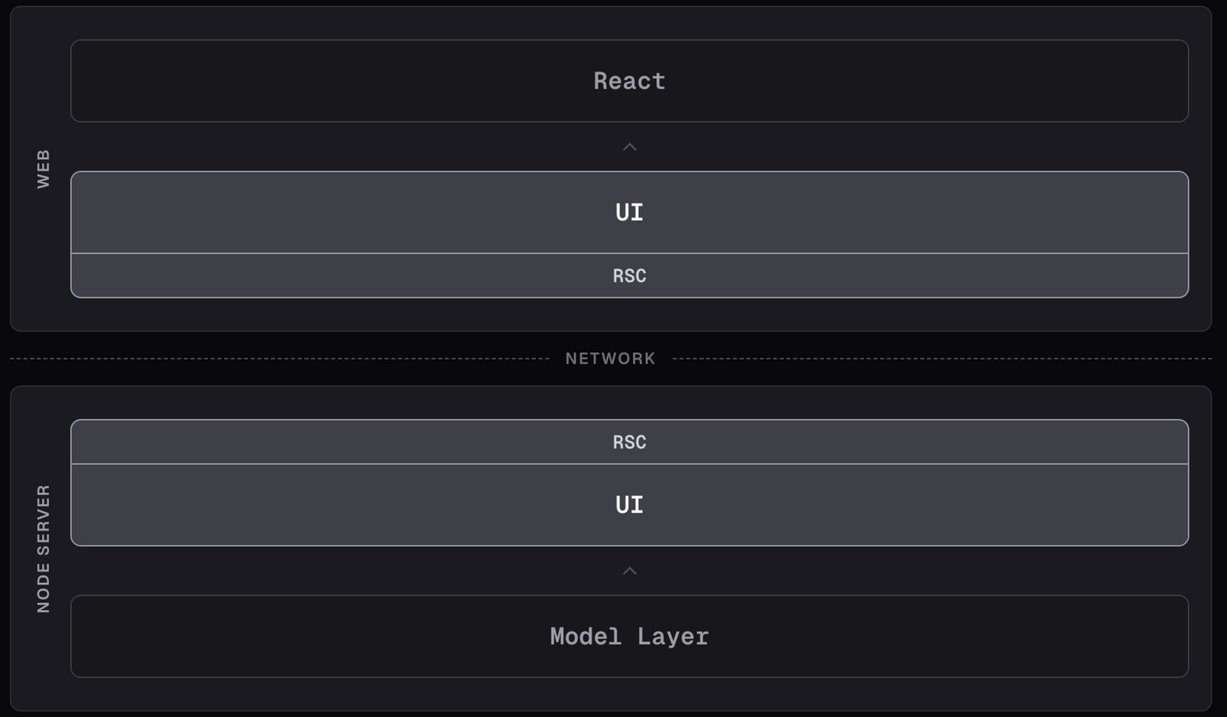
React Server Components

```
async function App() {

  const user = await User.find(4);

  return <ClientTitle>
    {user.fullName()}
  </ClientTitle>;
}
```

- ✅ UI can access model layer
- ✅ Avoid network waterfalls
- ✅ **Same language - just code ← !!**
- ✅ Optimized client bundle cost
- ✅ Effortlessly move compute

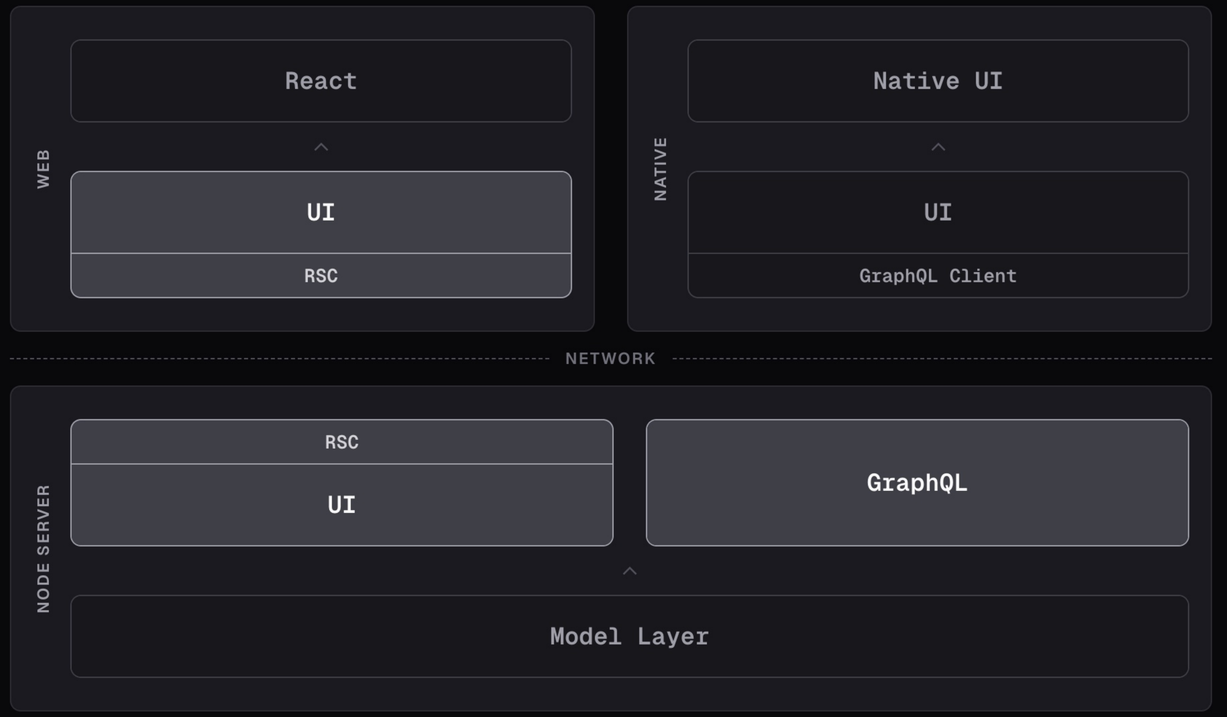
Node Server

RSC + Native Clients



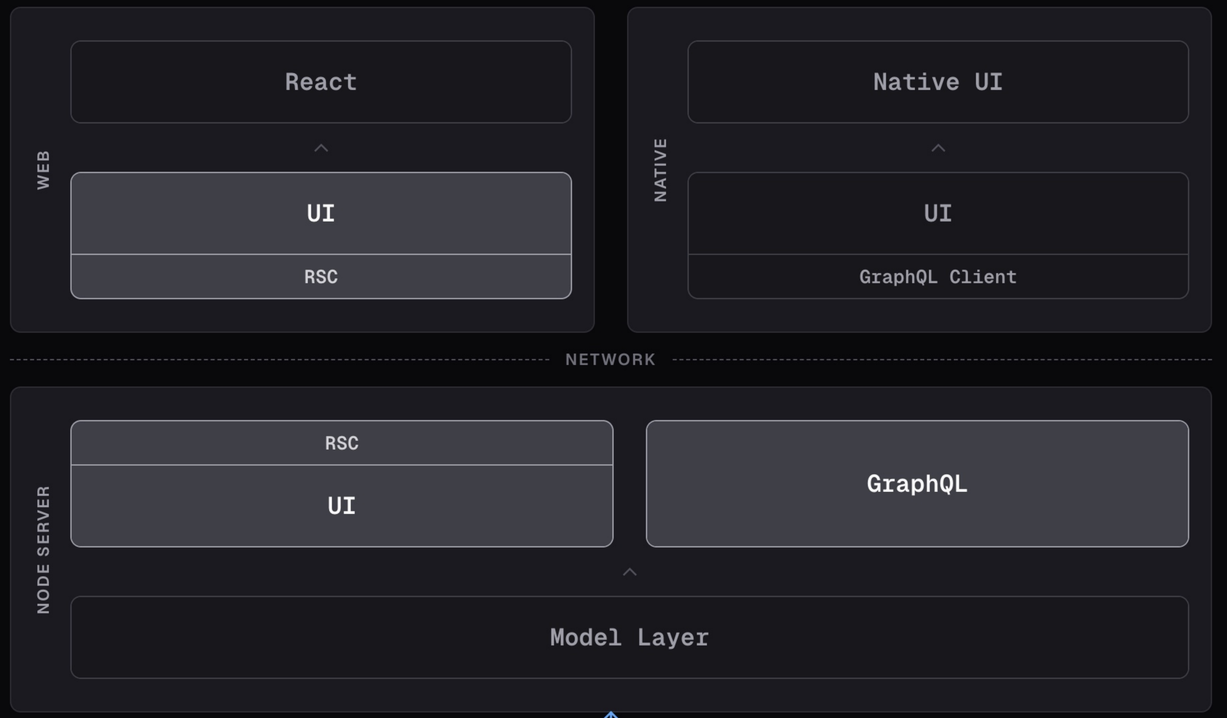
Node Server

RSC + Native Clients



Node Server

RSC + Native Clients



← Grats!

Caching and permissions belong here

RSC + GraphQL

Server components consuming GraphQL data

```
async function App() {
  const { user } = await serverFetchQuery(
    graphql`query {
     user(id: 4) { fullName }
    }`
  );
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

RSC + GraphQL

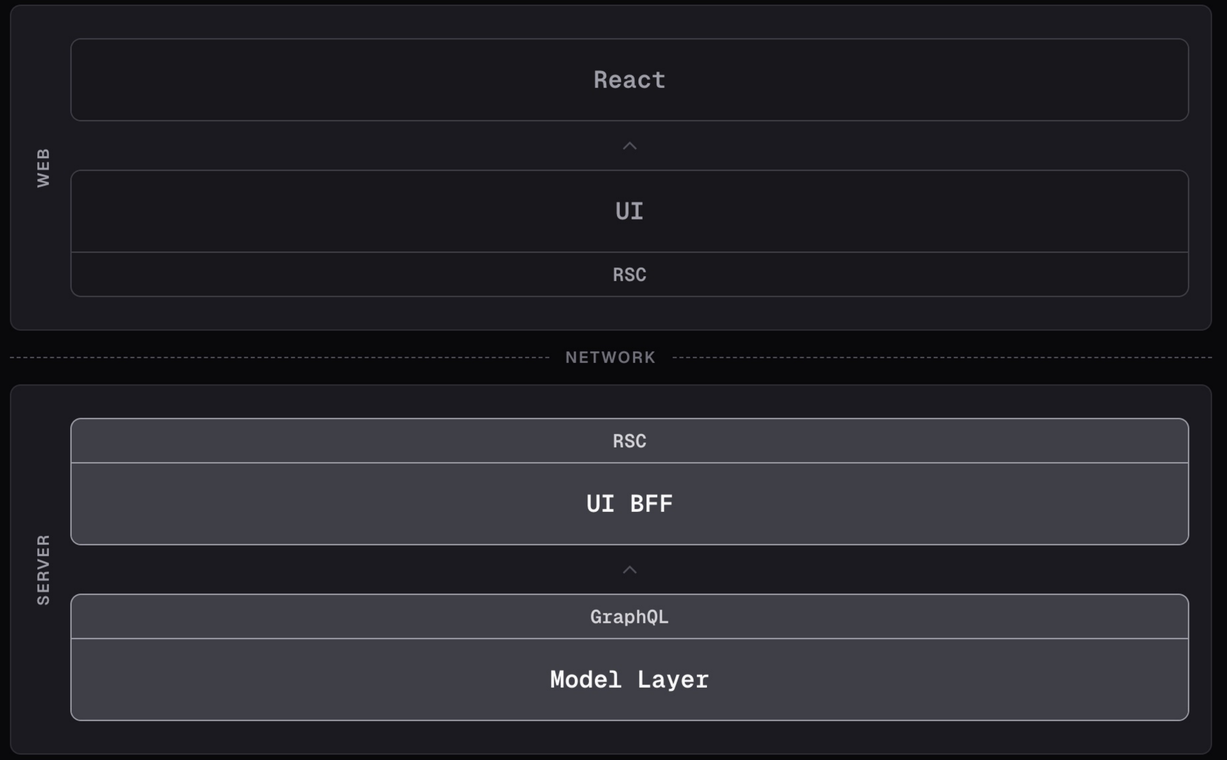
Server components consuming GraphQL data

```
async function App() {
  const { user } = await serverFetchQuery(
    graphql`query {
     user(id: 4) { fullName }
    }`
  );
  return <ClientTitle>
    {user.fullName}
  </ClientTitle>;
}
```

- Effortlessly move compute to the server
- Optimize JavaScript bundle size

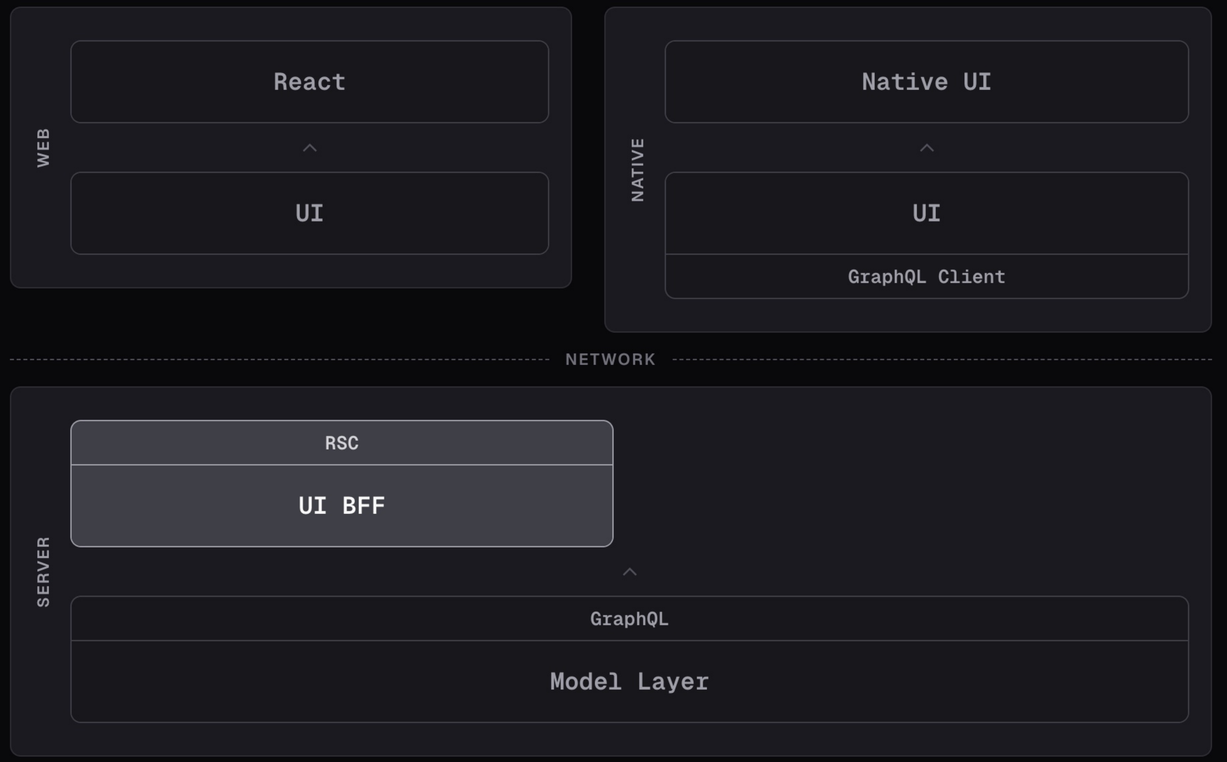
RSC with a Non-Node Server

RSC accesses the model layer via GraphQL



RSC with a Non-Node Server

RSC accesses the model layer via GraphQL



Current RSC Gaps

Relative to Relay GraphQL SPAs

- Challening to build highly dynamic apps
- No built-in mechanism for granular data updates
- No formally defined data boundary

	Node Server	Non-Node Server
Web	✅ RSC	⚠️ RSC + GraphQL ⚠️ SPA GraphQL
Native	✅ GraphQL	✅ GraphQL

Q&A

	Node Server	Non-Node Server
Web	✅ RSC	⚠️ RSC + GraphQL ⚠️ SPA GraphQL
Native	✅ GraphQL	✅ GraphQL