GraphQL

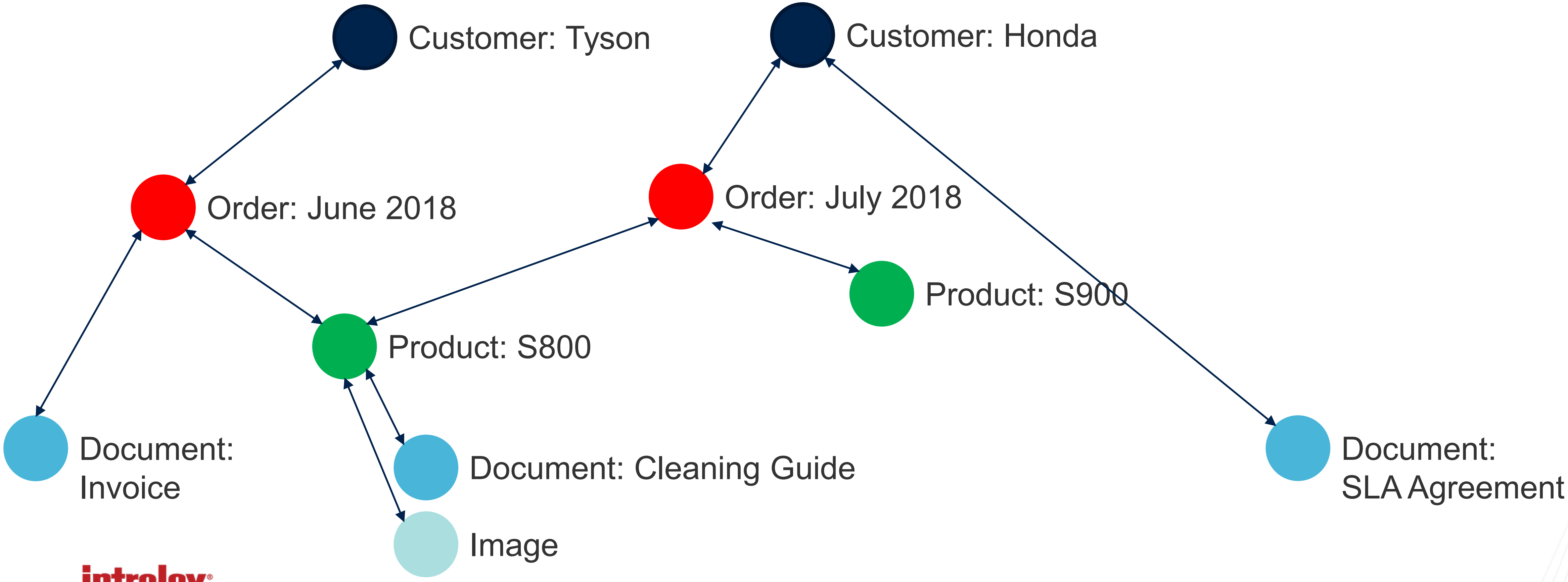
2/18/2020

Before we start!!

- First and foremost, it’s a abstraction.
  - You can do everything under the sun without graphql, but having graphQL saves you time and effort.
  - It also gives you a thinking model which enables better collaboration with other developers.
  - Finally – GraphQL users choose graphql because of its numerous benefits, but stay because it makes it easier to build applications. There are other benefits, but development experience seems to be the primary reason of its stickiness.
  - There are applications where GraphQL is not the correct tool.

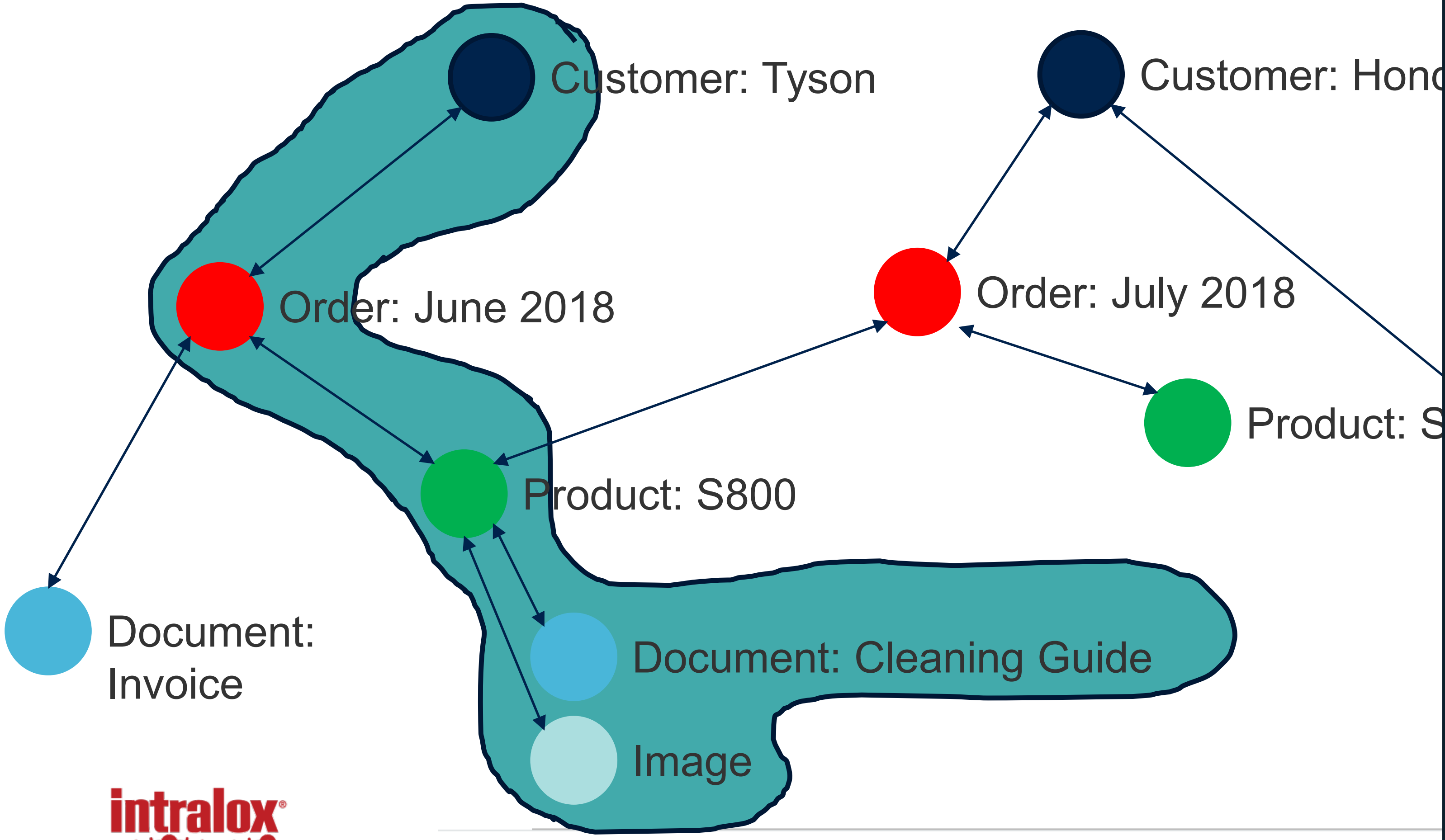
What’s in the Name?

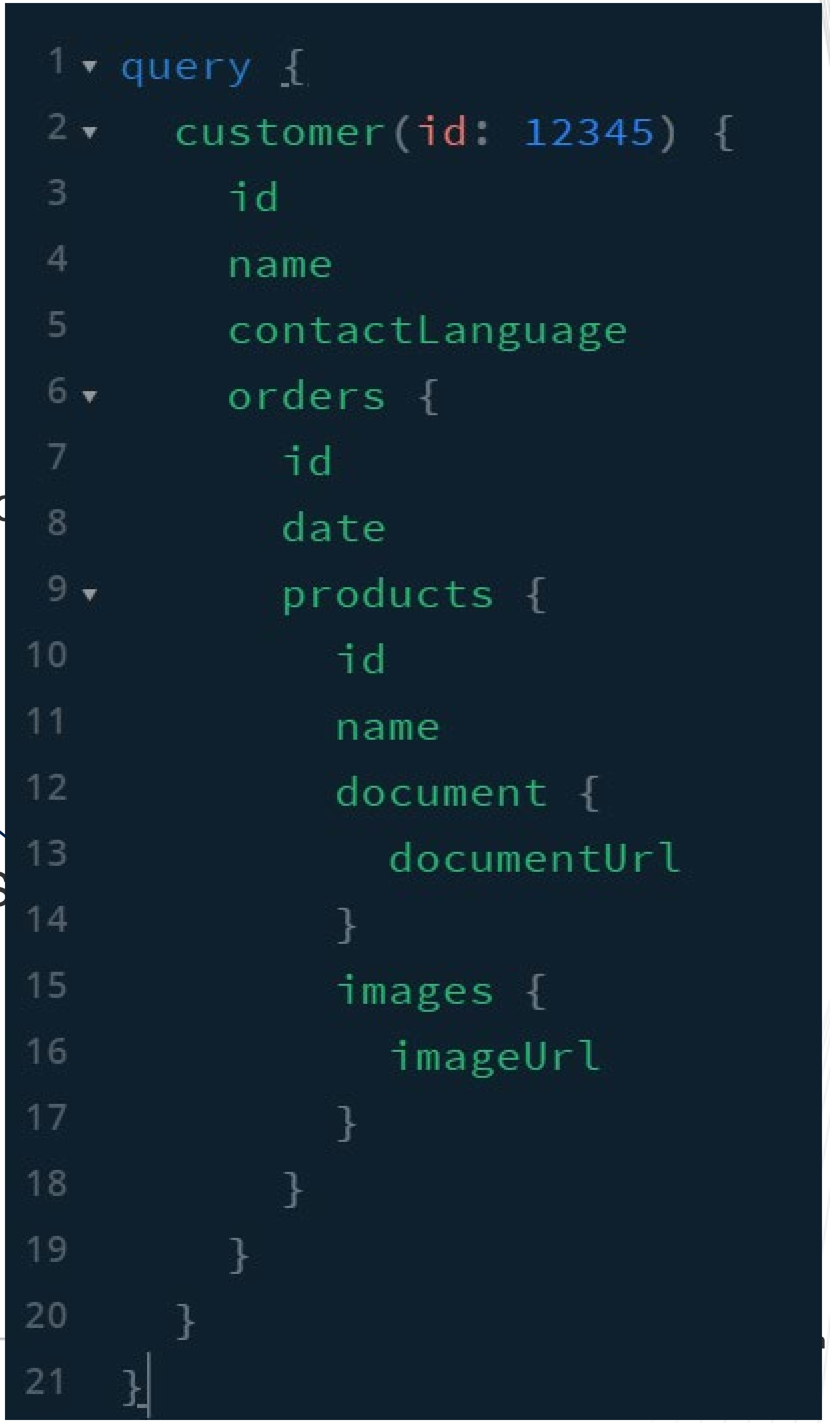
Graph: Think of application data as nodes that are connected to each other.



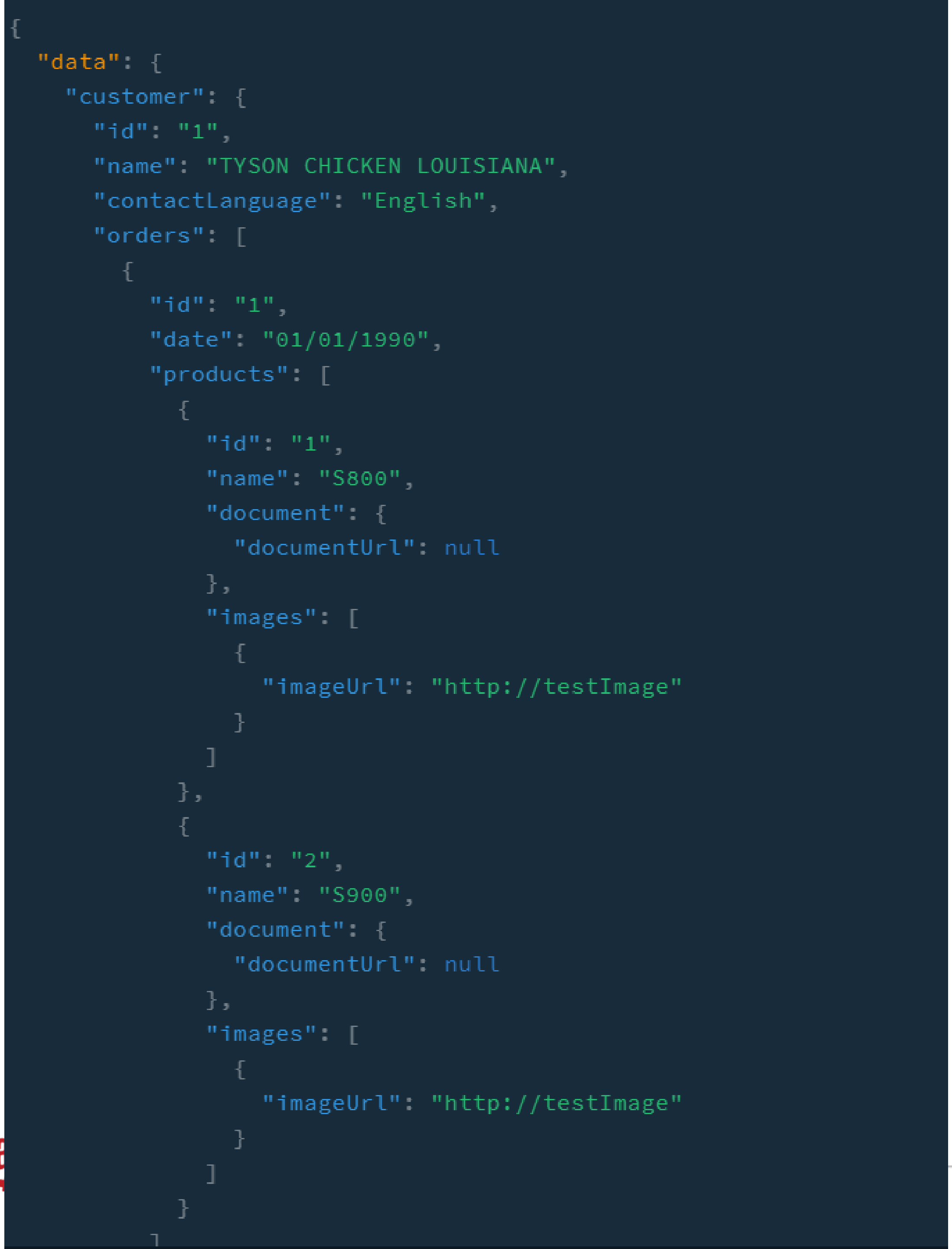
What’s in the Name?

- QL: Query Language

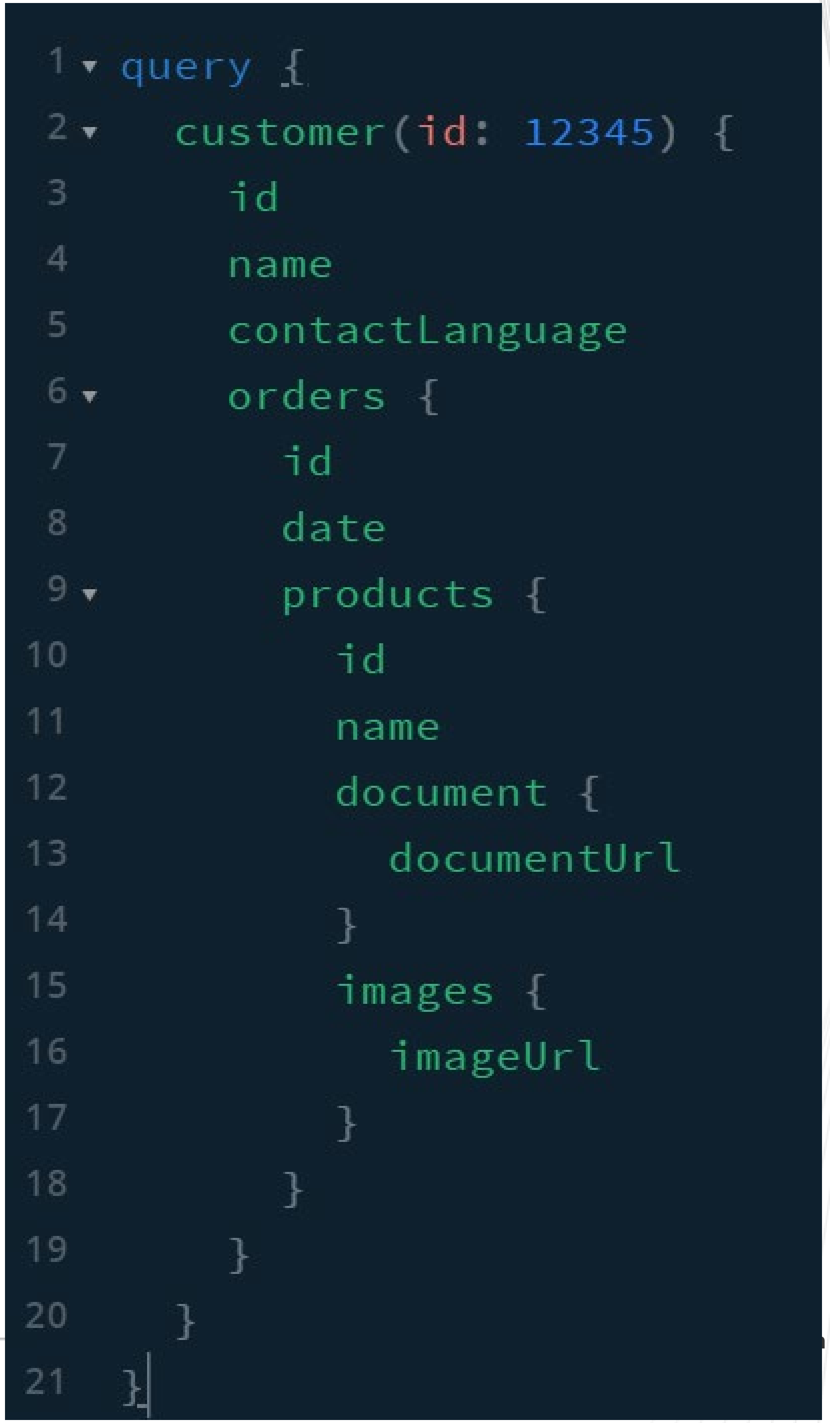




```
query {
  customer(id: 12345) {
    id
    name
    contactLanguage
    orders {
      id
      date
      products {
        id
        name
        document {
          documentUrl
        }
        images {
          imageUrl
        }
      }
    }
  }
}
```



```
{
  "data": {
    "customer": {
      "id": "1",
      "name": "TYSON CHICKEN LOUISIANA",
      "contactLanguage": "English",
      "orders": [
        {
          "id": "1",
          "date": "01/01/1990",
          "products": [
            {
              "id": "1",
              "name": "S800",
              "document": {
                "documentUrl": null
              },
              "images": [
                {
                  "imageUrl": "http://testImage"
                }
              ]
            },
            {
              "id": "2",
              "name": "S900",
              "document": {
                "documentUrl": null
              },
              "images": [
                {
                  "imageUrl": "http://testImage"
                }
              ]
            }
          ]
```

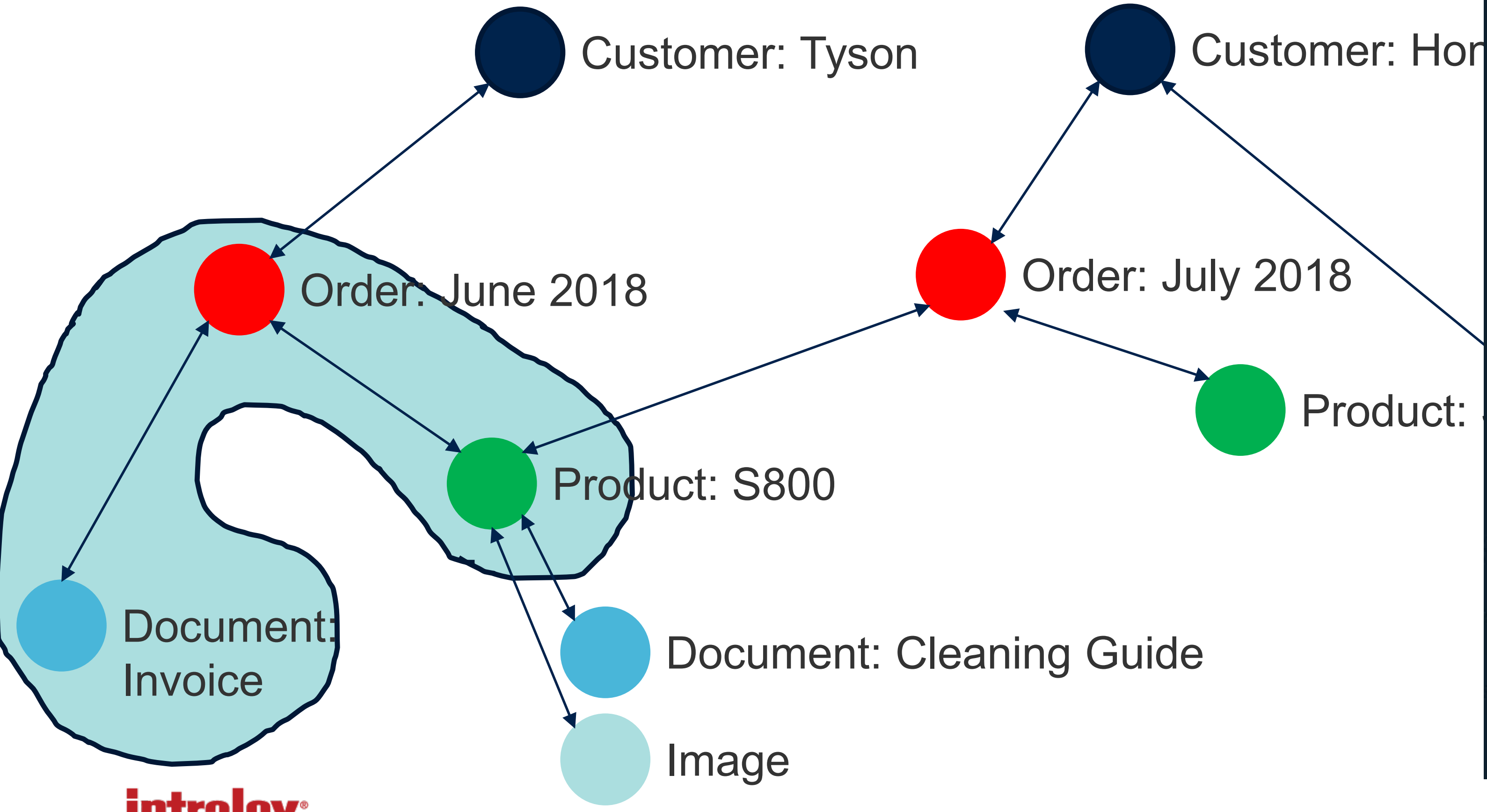


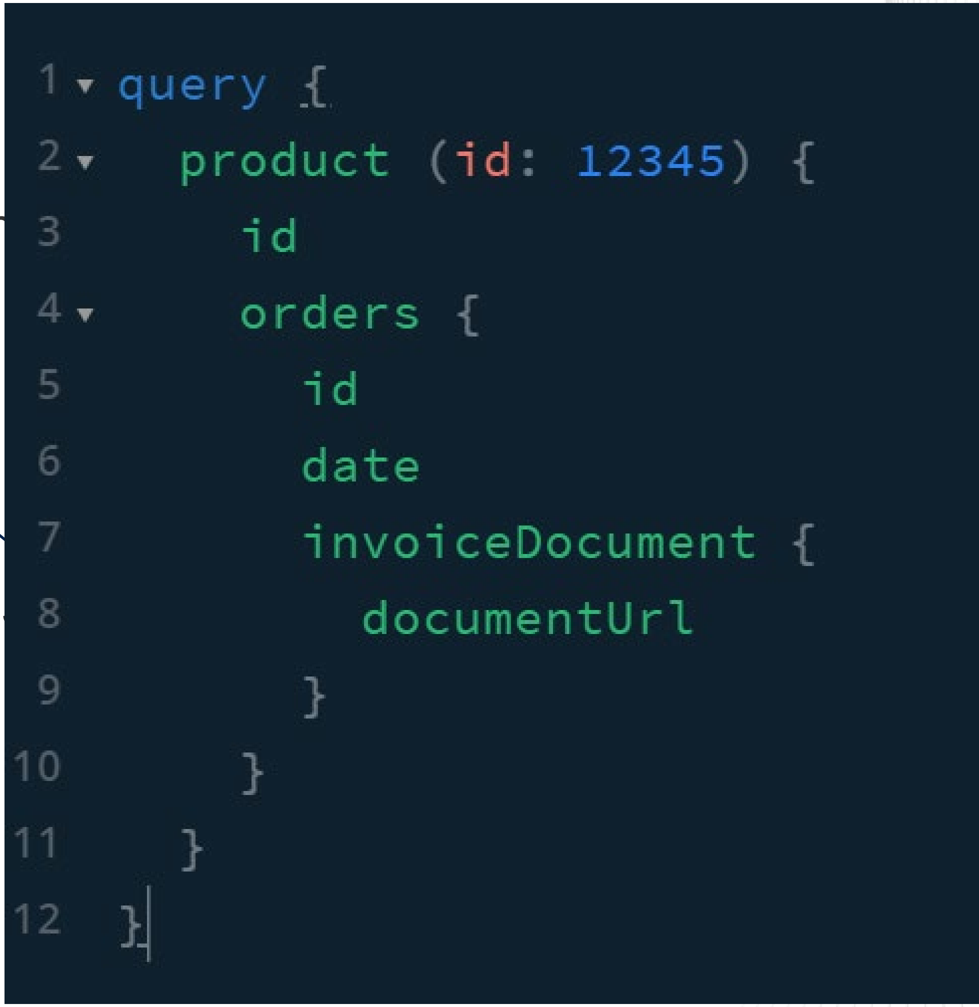
```
query {
  customer(id: 12345) {
    id
    name
    contactLanguage
    orders {
      id
      date
      products {
        id
        name
        document {
          documentUrl
        }
        images {
          imageUrl
        }
      }
    }
  }
}
```

GraphQL lets you start at Any Node

- Graphql lets you configure your API such that you can start at any node, and get other related information.
  - Eg: Start at product and get all orders that include this specific product.

GraphQL lets you start at Any Node

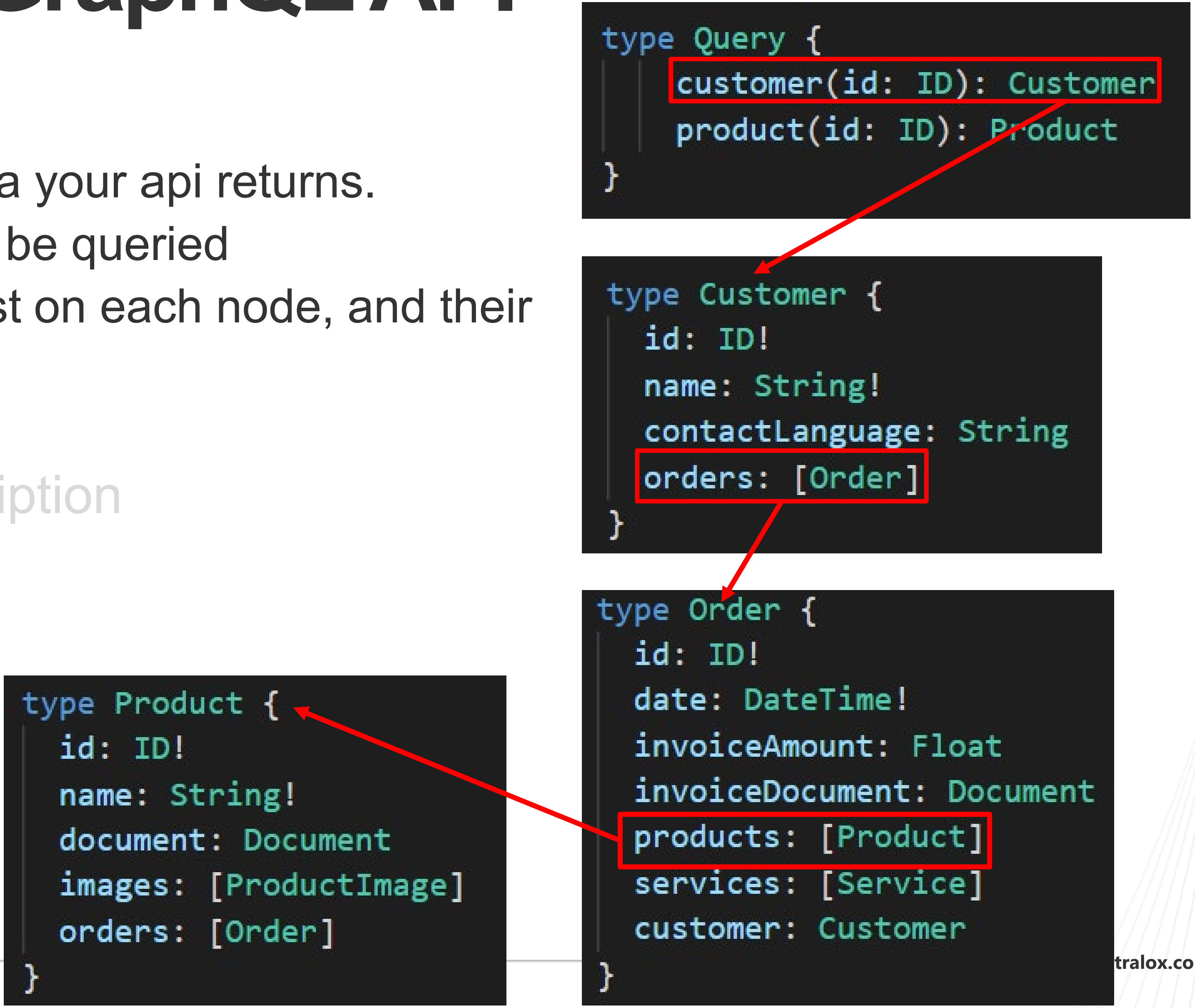




```
query {
  product (id: 12345) {
    id
    orders {
      id
      date
      invoiceDocument {
        documentUrl
      }
    }
  }
}
```

Components of GraphQL API

- Schema
  - Schema defines that data your api returns.
  - Declare what nodes can be queried
  - Declare what values exist on each node, and their types
- Resolvers
- Query / Mutation / Subscription



```
type Query {
  customer(id: ID): Customer
  product(id: ID): Product
}
```

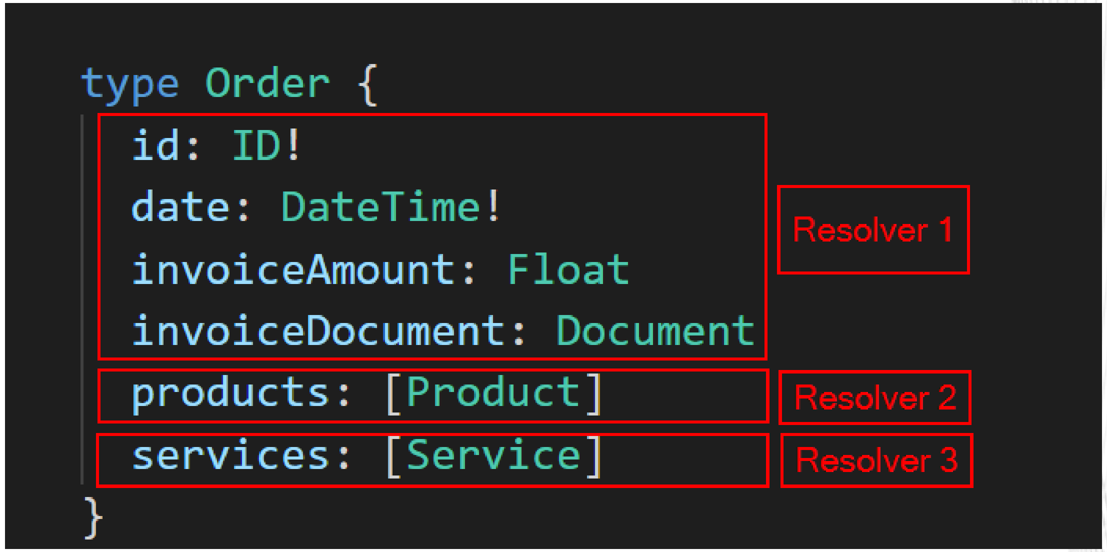
```
type Customer {
  id: ID!
  name: String!
  contactLanguage: String
  orders: [Order]
}
```

```
type Order {
  id: ID!
  date: DateTime!
  invoiceAmount: Float
  invoiceDocument: Document
  products: [Product]
  services: [Service]
  customer: Customer
}
```

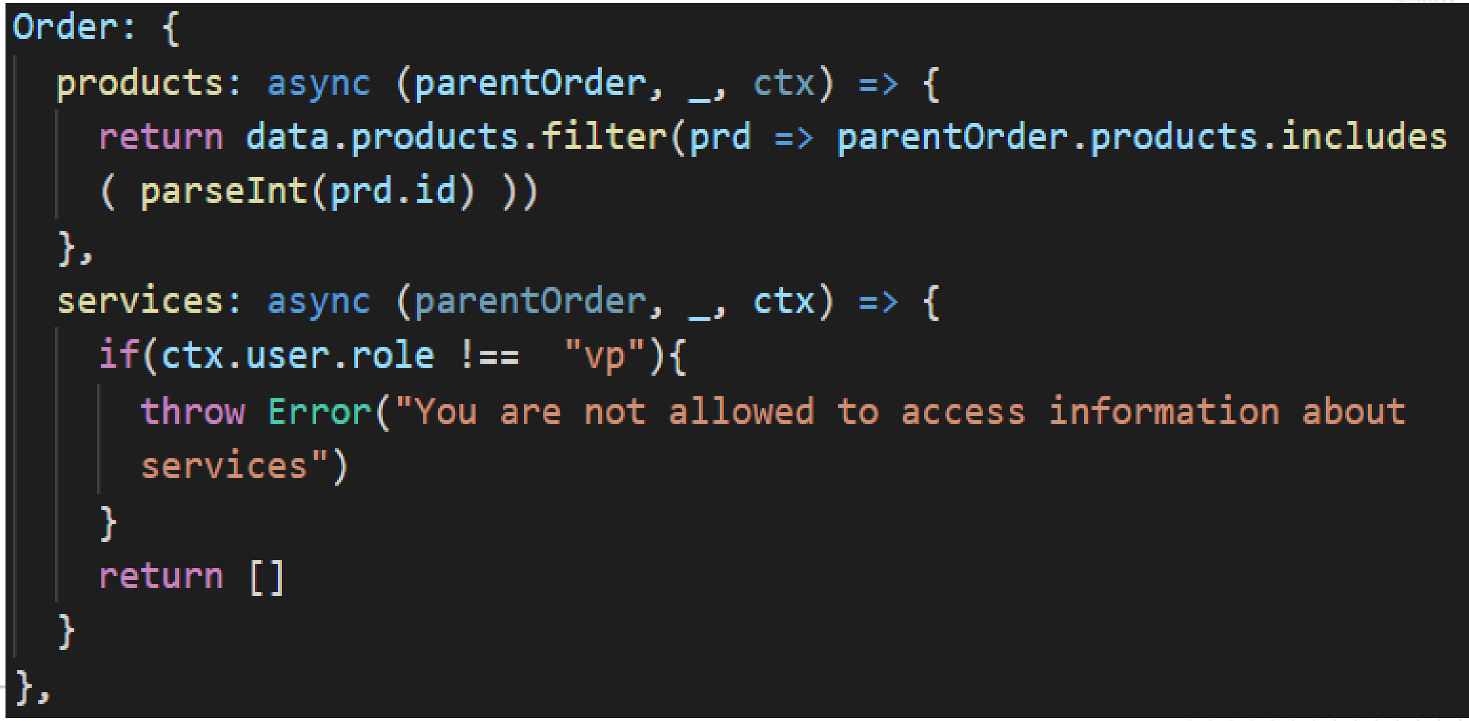
```
type Product {
  id: ID!
  name: String!
  document: Document
  images: [ProductImage]
  orders: [Order]
}
```

Components of GraphQL API

- Schema
- Resolvers
  - A function that knows how to obtain data
  - GraphQL allows the flexibility to define resolver on a data type or individual fields of a data type.
  - GraphQL runtime does the plumbing required. Application developers just write functions
  - GraphQL runtime only runs resolvers if necessary
  - Permissions can be applied on each resolver function
- Query / Mutation / Subscription



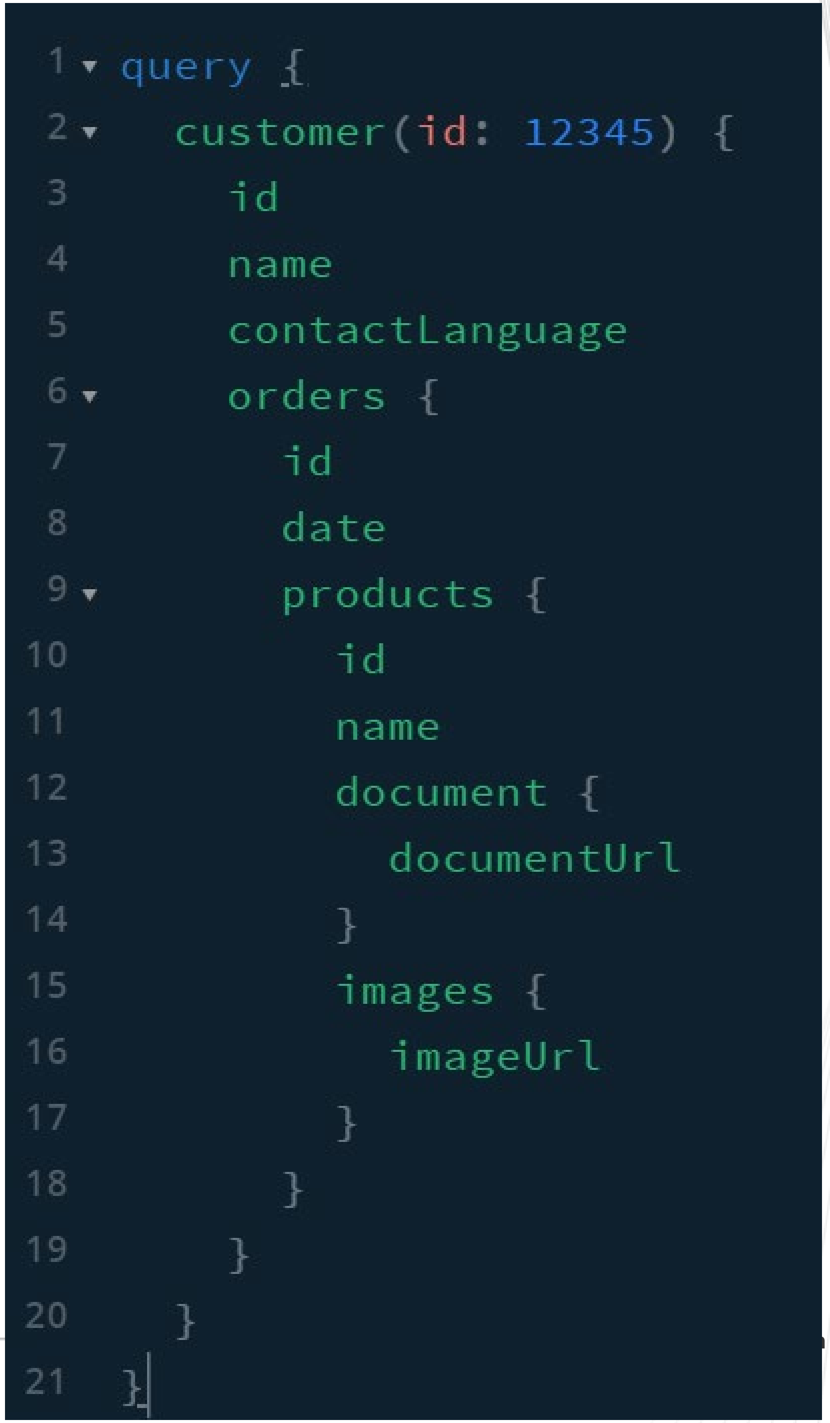
```
type Order {
  id: ID!
  date: DateTime!
  invoiceAmount: Float
  invoiceDocument: Document
  products: [Product]
  services: [Service]
}
```



```
Order: {
  products: async (parentOrder, _, ctx) => {
    return data.products.filter(prd => parentOrder.products.includes
    ( parseInt(prd.id) ))
  },
  services: async (parentOrder, _, ctx) => {
    if(ctx.user.role !==  "vp"){
      throw Error("You are not allowed to access information about
      services")
    }
    return []
  }
},
```

Components of GraphQL API

- Schema
- Resolvers
- Query / Mutation / Subscription
  - Query: Request for specific data
  - Mutation: Edit/Create data
  - Subscriptions: Real Time data
  - Both need to follow query language



```
query {
  customer(id: 12345) {
    id
    name
    contactLanguage
    orders {
      id
      date
      products {
        id
        name
        document {
          documentUrl
        }
        images {
          imageUrl
        }
      }
    }
  }
}
```

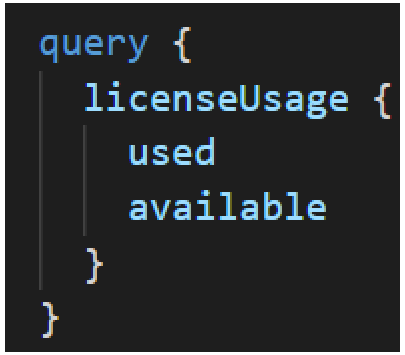
GraphQL enables efficient data loading

- Get only what you ask for
  - Only required amount of data is transferred through the network
  - Only required data is pulled from sources in the backend
  - Only one request to the backend no matter how you traverse the data graph
  - Additionally, multiple queries can be combined into one request
- Side effect
  - Once API has been designed, backend doesn’t have to change based on frontend’s needs.
  - This also enables much faster iteration on the frontend since you do not have to write new api end points or make changes to the api.
  - Speed to features

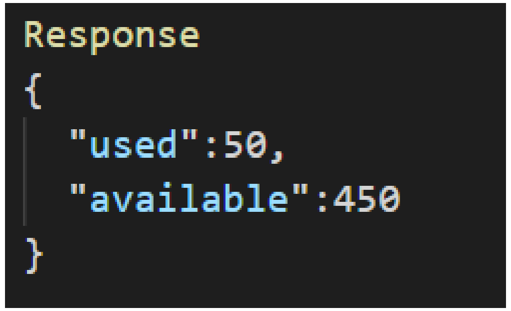
GraphQL enables efficient data loading

- Example from GraphQL API

API to get just license Usage

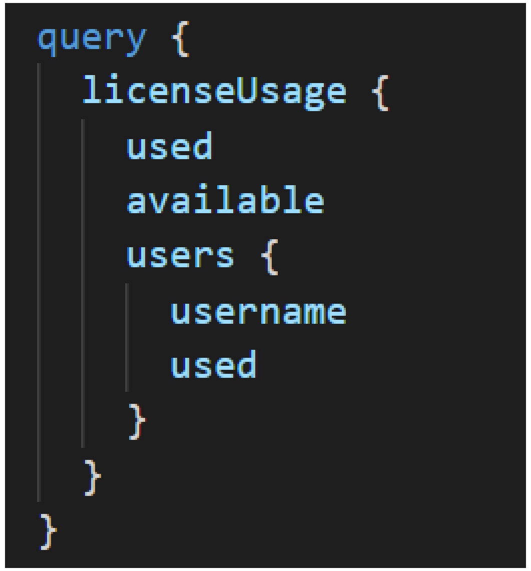


```
query {
  licenseUsage {
    used
    available
  }
}
```

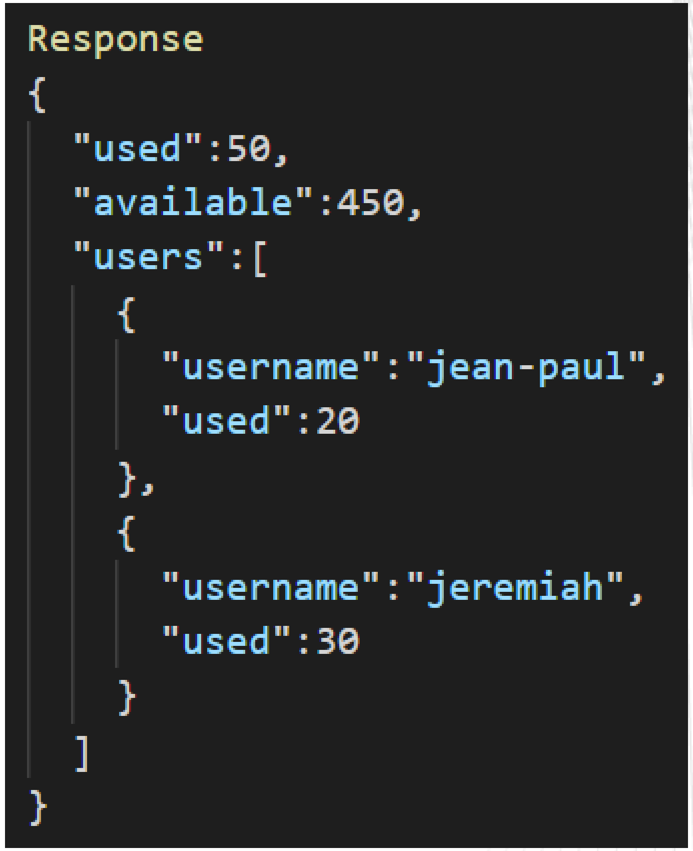


```
Response
{
  "used":50,
  "available":450
}
```

API to get just detailed license Usage



```
query {
  licenseUsage {
    used
    available
    users {
      username
      used
    }
  }
}
```

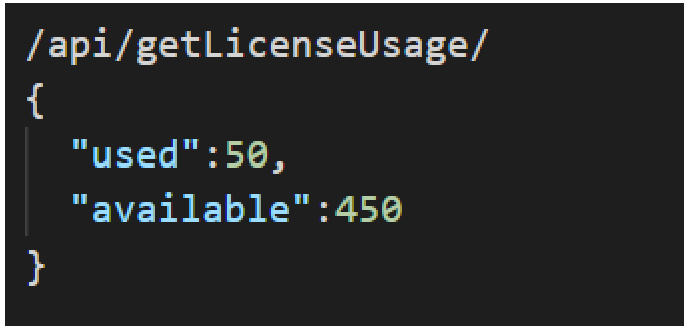


```
Response
{
  "used":50,
  "available":450,
  "users":[
    {
      "username":"jean-paul",
      "used":20
    },
    {
      "username":"jeremiah",
      "used":30
    }
  ]
}
```

GraphQL enables efficient data loading

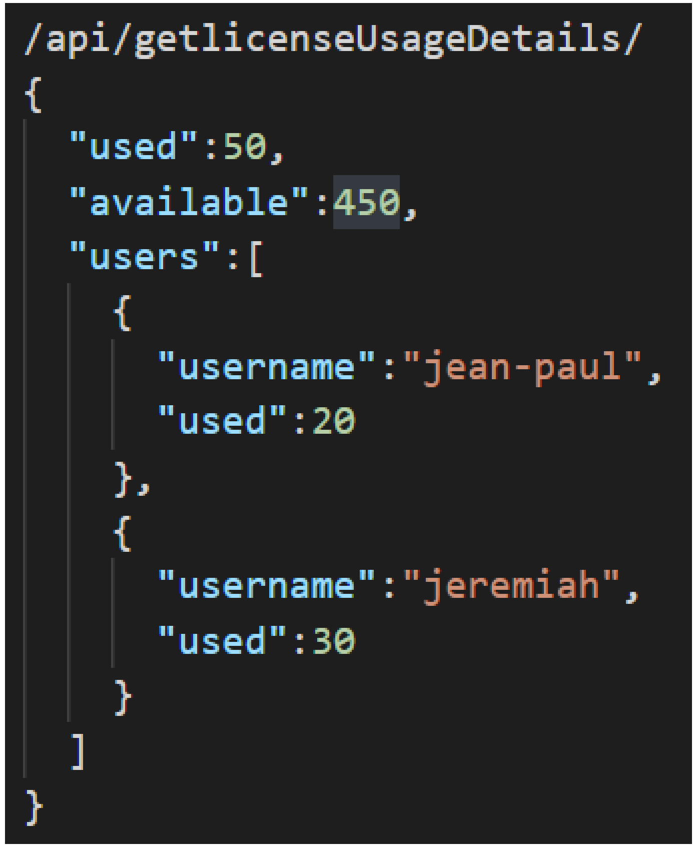
- Compare with Example from Simulation Portal

API to get just license Usage



```
/api/getLicenseUsage/
{
  "used":50,
  "available":450
}
```

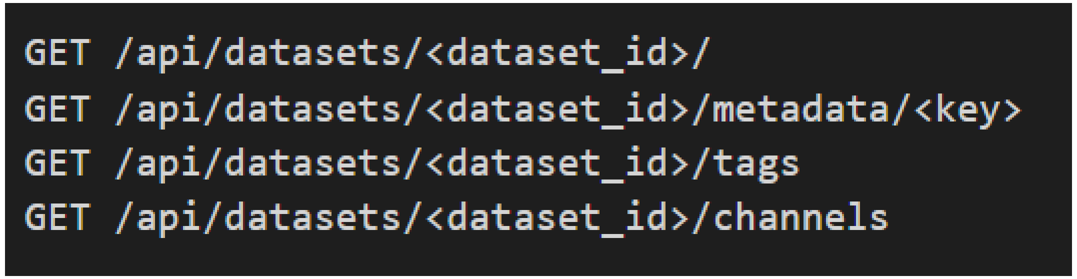
API to get just detailed license Usage



```
/api/getlicenseUsageDetails/
{
  "used":50,
  "available":450,
  "users":[
    {
      "username":"jean-paul",
      "used":20
    },
    {
      "username":"jeremiah",
      "used":30
    }
  ]
}
```

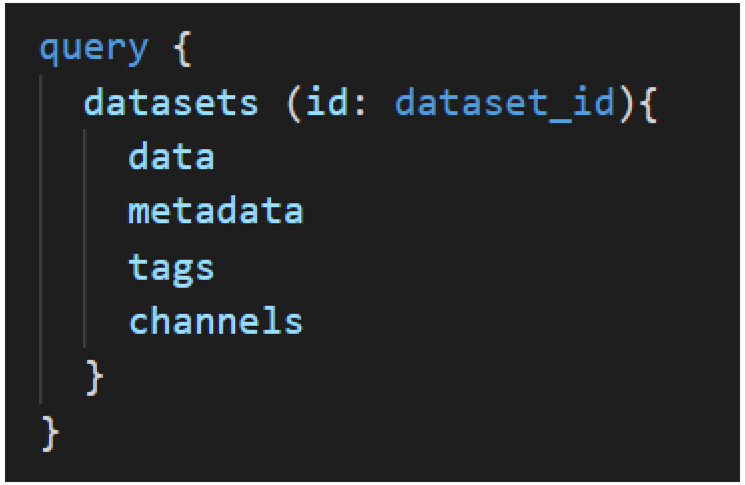
GraphQL enables efficient data loading

- Compare with Example from Sensing Portal



```
GET /api/datasets/<dataset_id>/
GET /api/datasets/<dataset_id>/metadata/<key>
GET /api/datasets/<dataset_id>/tags
GET /api/datasets/<dataset_id>/channels
```

vs



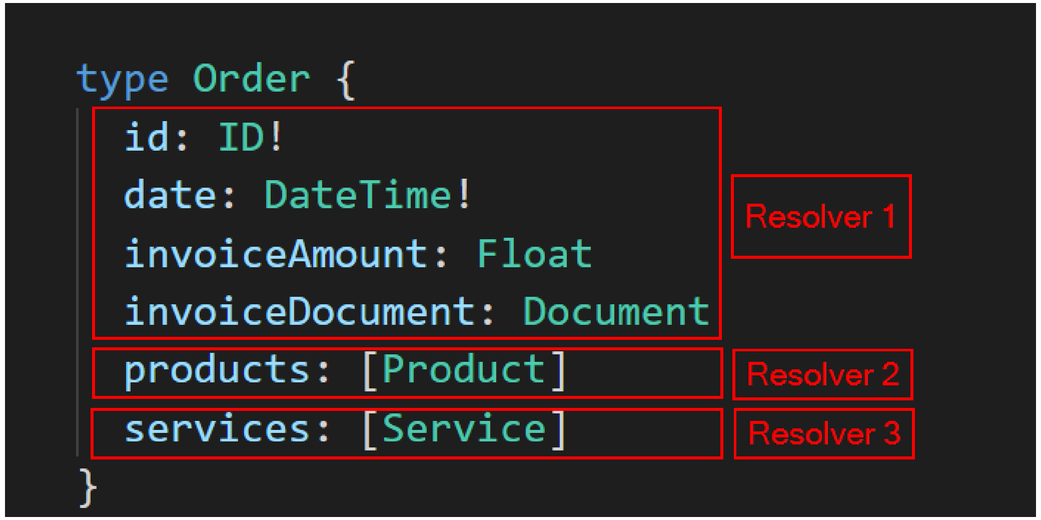
```
query {
  datasets (id: dataset_id){
    data
    metadata
    tags
    channels
  }
}
```

Four End points to get various info in REST

In GraphQL, you ask for what you need

GraphQL enables Fine Grain Permissions Checks

- Because we can write resolver for each node, it is possible to perform permission check on every node.

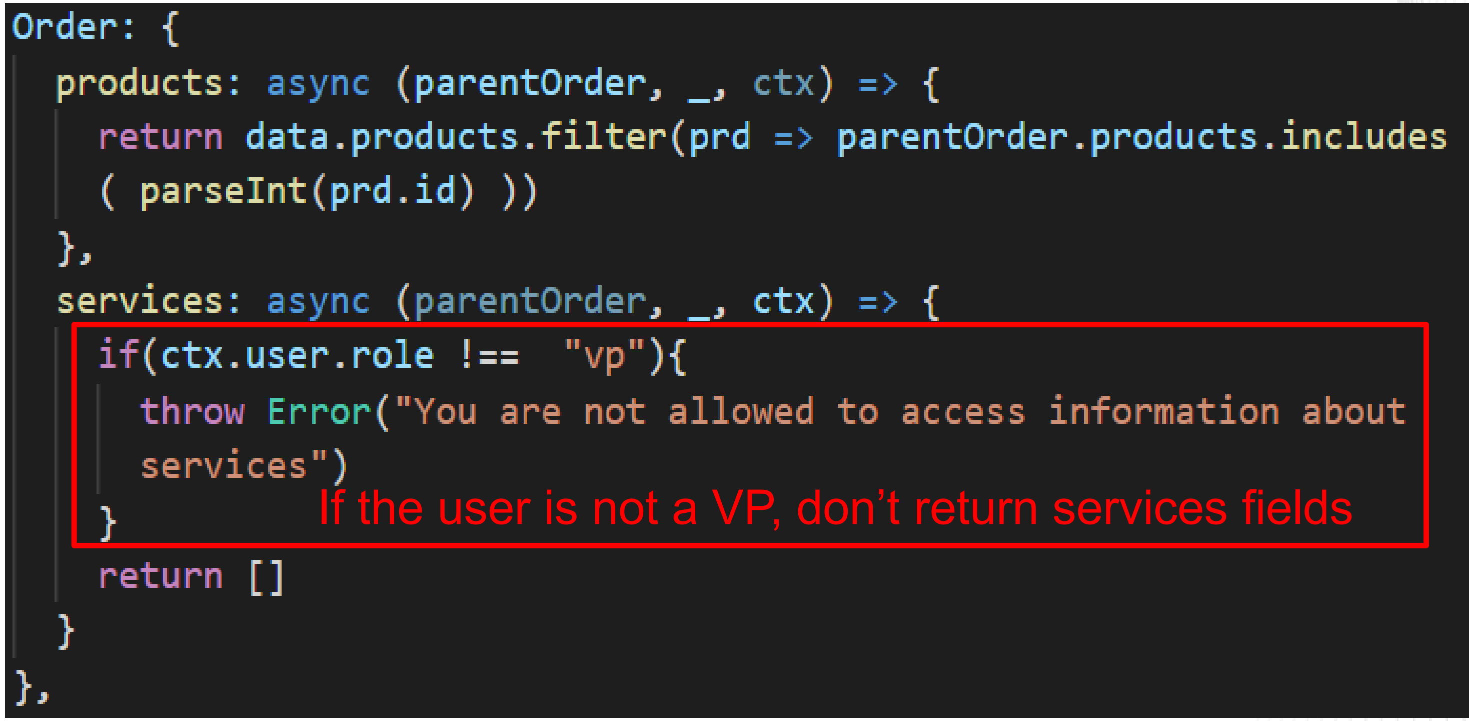


```
type Order {
  id: ID!
  date: DateTime!
  invoiceAmount: Float
  invoiceDocument: Document
  products: [Product]
  services: [Service]
}
```

Resolver 1

Resolver 2

Resolver 3

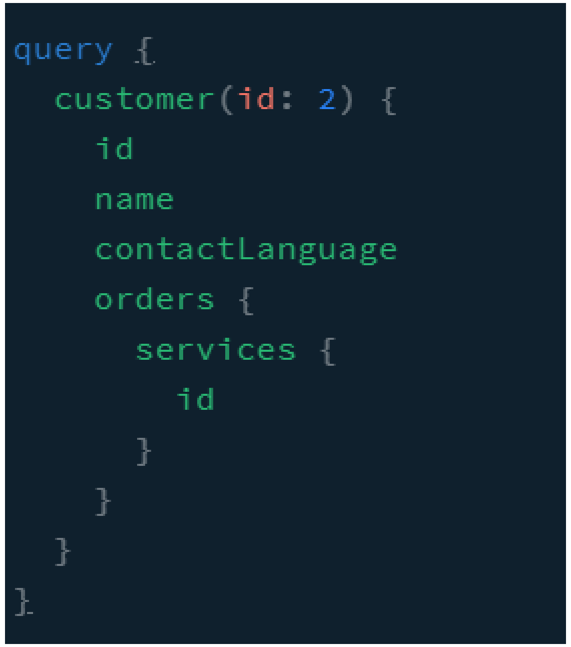


```
Order: {
  products: async (parentOrder, _, ctx) => {
    return data.products.filter(prd => parentOrder.products.includes
    ( parseInt(prd.id) ))
  },
  services: async (parentOrder, _, ctx) => {
    if(ctx.user.role !==  "vp"){
      throw Error("You are not allowed to access information about
      services")
    }
    return []
  }
},
```

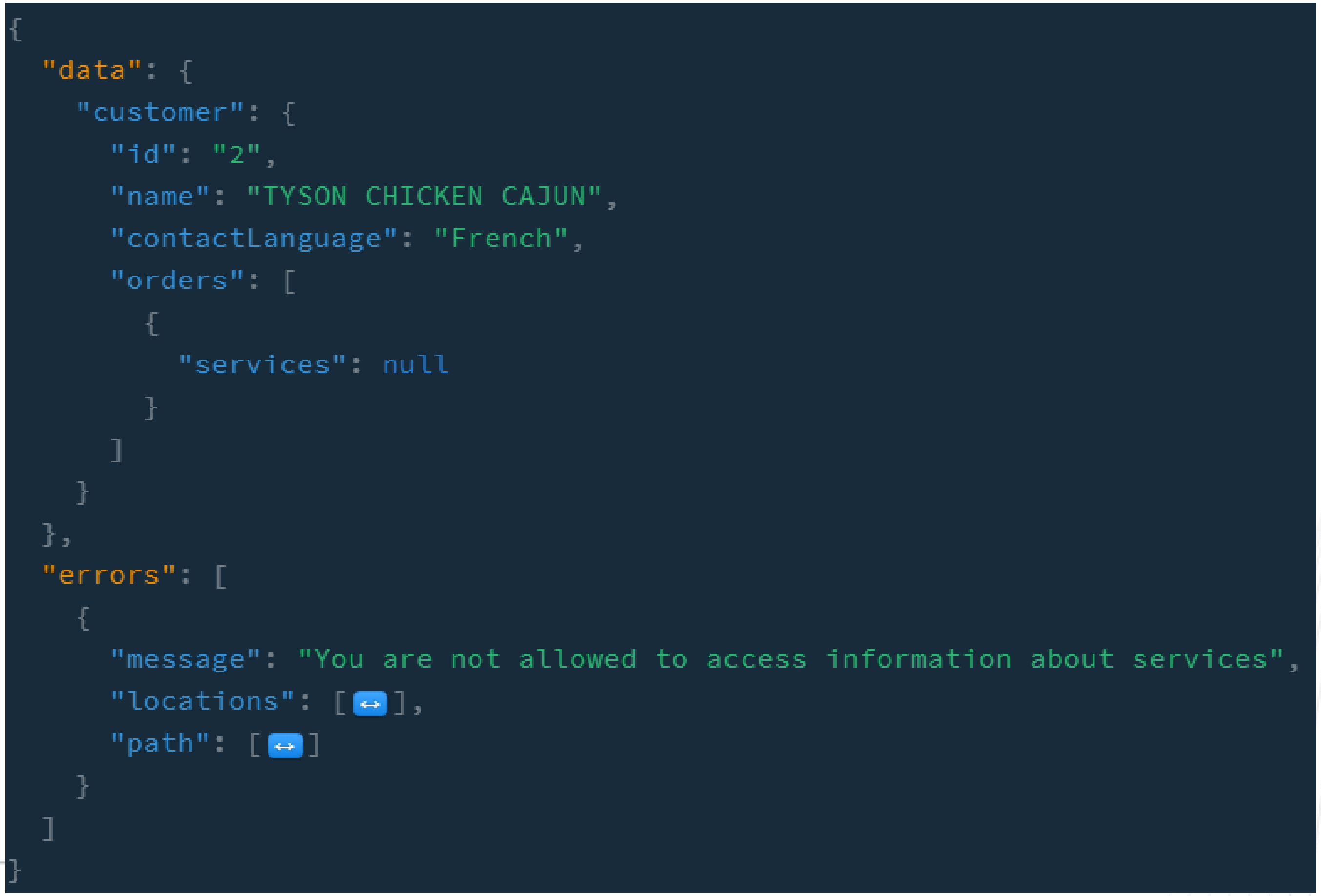
If the user is not a VP, don't return services fields

GraphQL enables Fine Grain Error Messages

- Because of permissions check, we also get fine grain error messages.
- Data that can be fetched is returned along with error message.



```
query {
  customer(id: 2) {
    id
    name
    contactLanguage
    orders {
      services {
        id
      }
    }
  }
}
```



```
{
  "data": {
    "customer": {
      "id": "2",
      "name": "TYSON CHICKEN CAJUN",
      "contactLanguage": "French",
      "orders": [
        {
          "services": null
        }
      ]
    }
  },
  "errors": [
    {
      "message": "You are not allowed to access information about services",
      "locations": [↔],
      "path": [↔]
    }
  ]
}
```

GraphQL enables more efficient caching

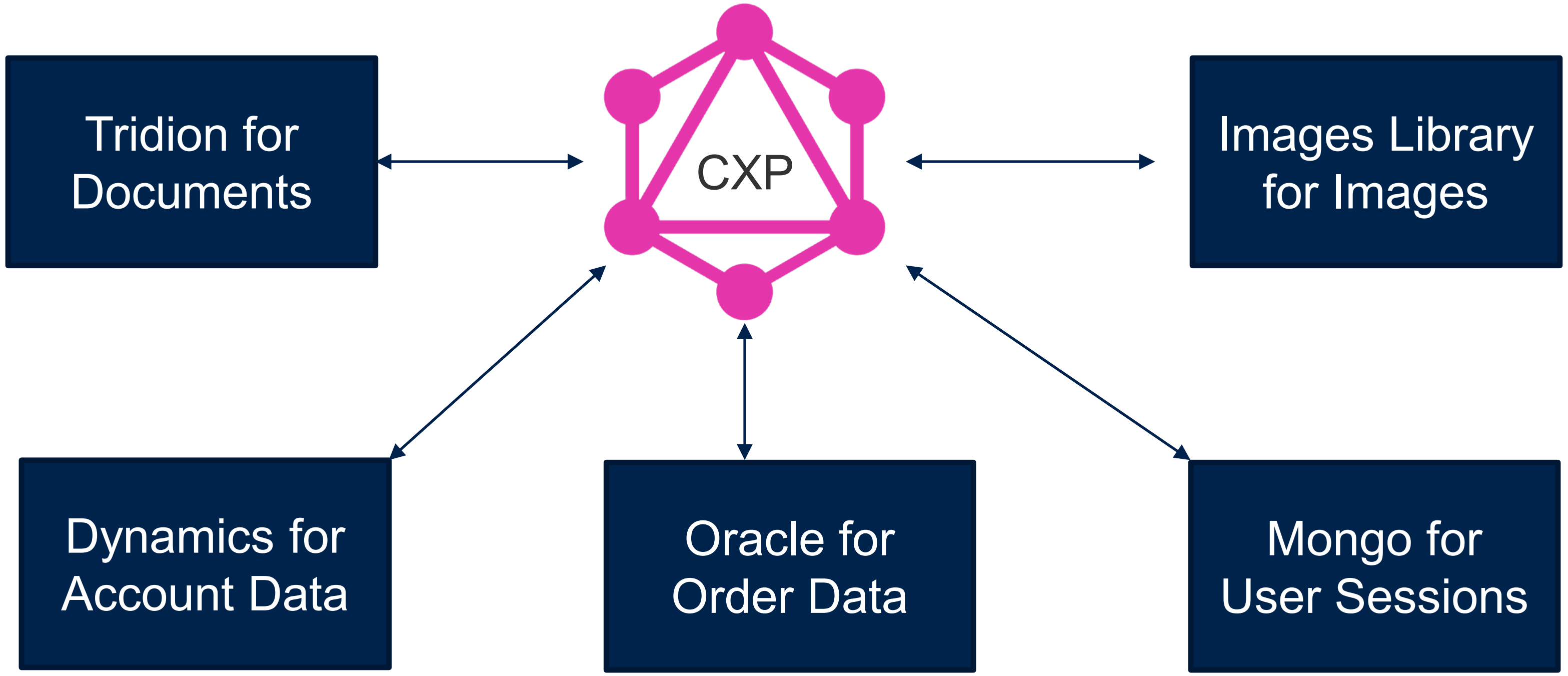
- With older technologies like REST, resource was cached based on URL.
  - This meant that if the same resource was accessed by two different endpoints, they were both saved in the cache as separate items
- With GraphQL, each Node cached which creates efficiencies in caching
- Disadvantage:
  - Node level caching is generally more complex to implement than url level caching
  - Luckily the community has already developed lots of libraries for caching.

GraphQL enables writing less code

- Data plumbing is taken care of by GraphQL runtime.
- Do not need any extra code for arbitrary ways of traversing the graph

GraphQL enables easy ways to mix different data sources

- Data can be accessed from various different sources. GraphQL makes it very easy to wire up different sources together.



Industry Leaders are using GraphQL

- Capital One
  - Uses GraphQL as a way of wiring up various sources of data in their data analytics team.
- GitLab
  - Planning on moving completely to GraphQL based data API because of flexibility it provides
- Expedia
  - Using Graphql to power all of their FrontEnds (website, app, client specific sites)
- Intuit
  - Using GraphQL to connect all of their micro services together
- Facebook
  - GraphQL apis power their news feed which changes over time

Why did we decide to use GraphQL with CXP?

- Lots of data sources need to be wired up together
- CXP needs will continue evolving. This enables us to not have to change our backend, even as frontend evolves
- Faster development time because of self documenting API

Real Demo + Questions?

Extra Slides Below

Benefit of thinking of app data as a graph

- REST is the current best alternative. To build out our two requirements here are the rest endpoints. Compare this with GraphQL in later slides

REST Endpoints	
/customers	Returns list of customers
/customers/id	Returns info about customers
/customers/id/orders	Returns orders for the customer
/orders/id/documents	Returns documents related to a specific order
/orders/id/images	Returns images related to a specific order
/products/id/images	Returns images related to a specific product
/products/id/orders	Returns orders for product with a certain id
/orders/id/documents	Returns documents for a specific order

Challenges with REST

- Have to know requirements ahead of time.
- As a project evolves, new APIs get added to fulfill specific data requirements of evolving application.
- Clients and Front end developers do not have a automated way to know what’s possible and how to query it.
- Always dealing with over-fetching or under-fetching