

# Introduction to GraphQL

Krishna Regmi

# Before we start!!

- First and foremost, it's an abstraction.
    - You can do everything with or without GraphQL, but having GraphQL saves you time and effort.
    - On top of code abstraction, GraphQL also gives you a thinking model that enables better collaboration with other developers.
  - GraphQL ultimately makes it easier to build and maintain web applications.
  - There are applications where GraphQL is not the correct tool for the job.
-

# Why Industry Leaders are using GraphQL

- Facebook (also the creator)
    - GraphQL APIs has been powering their news feed since 2012
  - Capital One
    - Uses GraphQL as a way of wiring up various sources of data in their data analytics team.
  - Intuit
    - Uses GraphQL to connect all of their micro services together
  - Tinder
    - Uses GraphQL to power their Blog called Swipe Life
  - GitLab
    - Planning on moving completely to GraphQL based data API because of flexibility it provides
  - Expedia / Airbnb
    - Use GraphQL to power their Frontends (website, app, client specific sites)
-

# Why did we decide to use GraphQL with CXP?

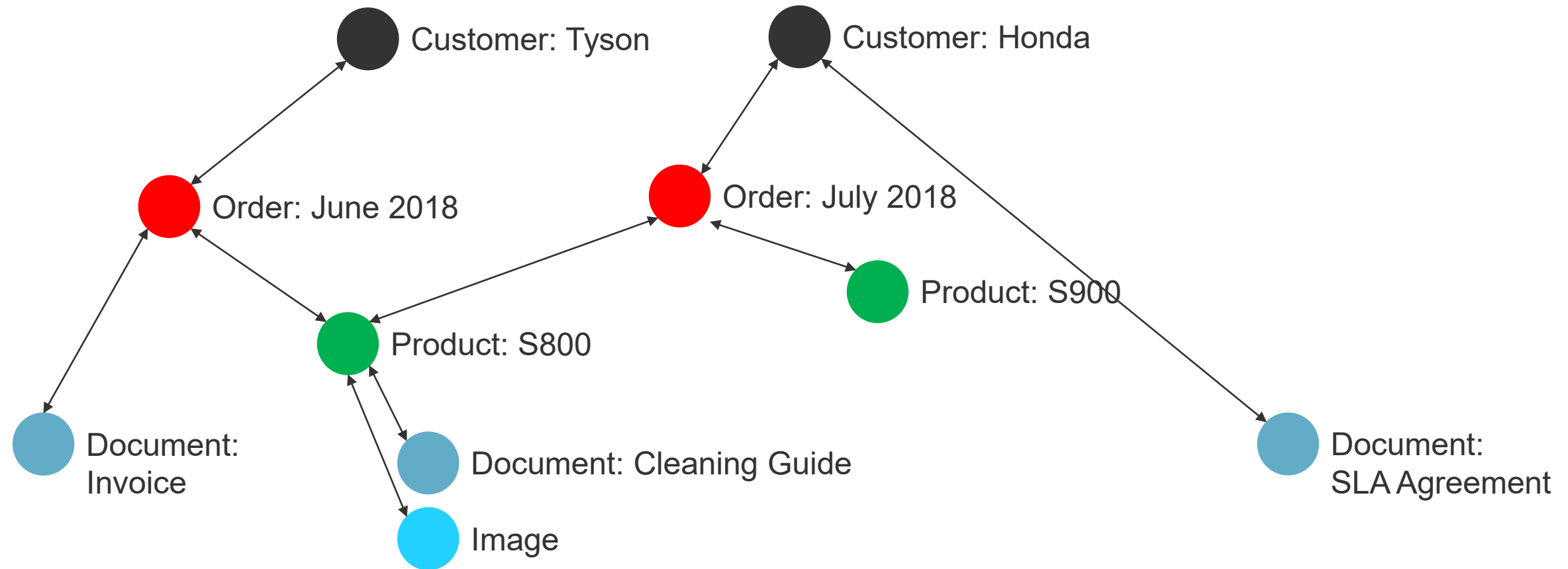
- Complex web of data sources that need to be wired up together
  - CXP needs will continue evolving. This enables us to not have to change our backend, even as frontend evolves
  - Faster development time because of self-documenting API and queries
-

**What's in the Name?**

**Graph + QL**

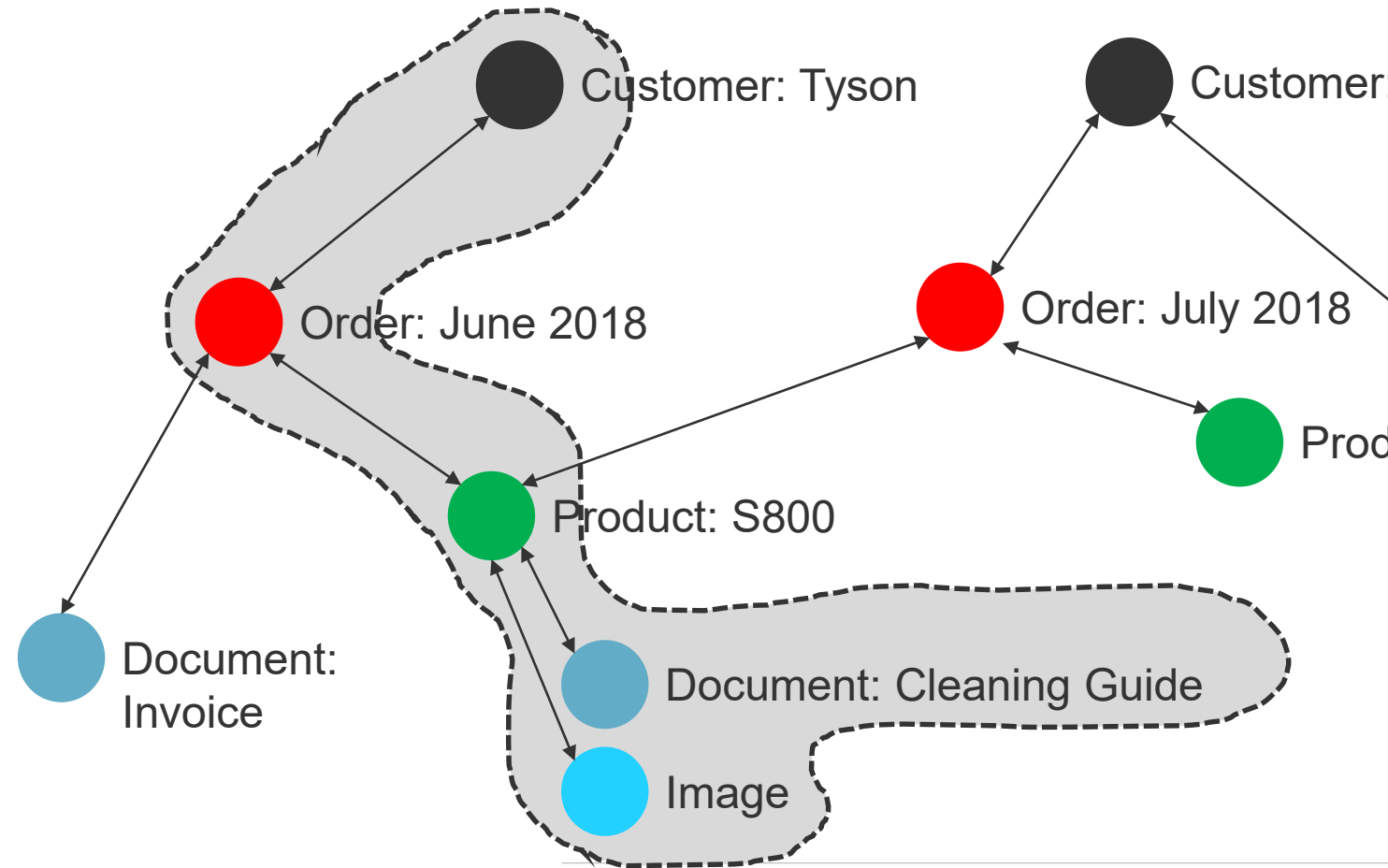
# Graph

Consider your application data as a graph of nodes that are connected.



# QL

## Query Language



```
1 query {
2   customer(id: 1) {
3     id
4     name
5     contactLanguage
6     orders {
7       id
8       date
9       products {
10        id
11        name
12        document {
13          documentUrl
14        }
15        images {
16          imageUrl
17        }
18      }
19    }
20  }
21 }
```

```

{
  "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"
                }
              ]
            }
          ]
        }
      ]
    }
  }
}

```



```

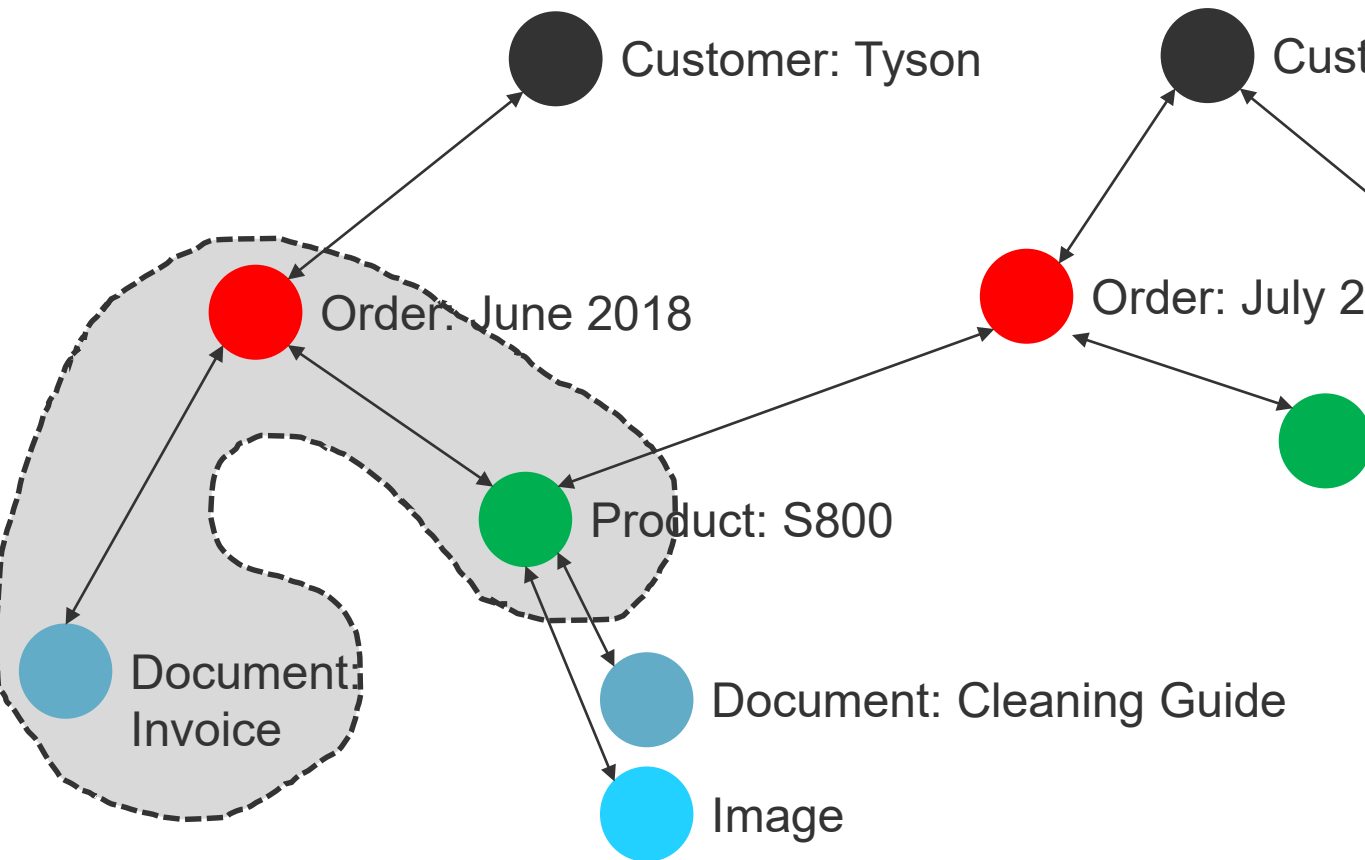
1 query {
2   customer(id: 1) {
3     id
4     name
5     contactLanguage
6     orders {
7       id
8       date
9       products {
10        id
11        name
12        document {
13          documentUrl
14        }
15        images {
16          imageUrl
17        }
18      }
19    }
20  }
21 }

```



**GraphQL lets you start at any  
node in the graph**

# GraphQL lets you start at Any Node



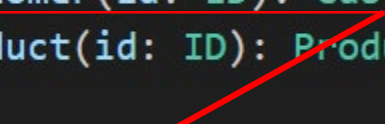
```
1 query {  
2   product (id: "S800") {  
3     id  
4     orders {  
5       id  
6       date  
7       invoiceDocument {  
8         documentUrl  
9       }  
10    }  
11  }  
12 }
```

# Components of GraphQL API


# Components of GraphQL API

- Schema
  - Schema defines the 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
    - Resolver is a function that gets the data requested
    - GraphQL runtime only runs resolvers necessary
    - Permissions can be applied on each resolver function
  - Query / Mutation / Subscription
-

# Components of GraphQL API

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

Resolver 1

Resolver 2

Resolver 3

```
const resolvers = {  
  Query: {  
    customer: async (_, {id}, ctx) => {  
      return data.customers.find(cust => cust.id == id)  
    },  
    product: async (_, {id}, ctx) => {  
      return data.products.find(p => p.id == id)  
    },  
    order: async (_, {id}, ctx) => {  
      return data.orders.find(ord => ord.id == id)  
    }  
  },  
}
```

Root Resolver

```
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

```
query {  
  order(id:1) {  
    id  
    invoiceAmount  
  }  
}
```

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

```
const resolvers = {  
  Query: {  
    customer: async (_, {id}, ctx) => {  
      return data.customers.find(cust => cust.id == id)  
    },  
    product: async (_, {id}, ctx) => {  
      return data.products.find(p => p.id == id)  
    },  
    order: async (_, {id}, ctx) => {  
      return data.orders.find(ord => ord.id == id)  
    }  
  },  
}
```

```
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

```
query {  
  order(id:1) {  
    id  
    invoiceAmount  
    products {  
      name  
    }  
  }  
}
```

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

```
const resolvers = {  
  Query: {  
    customer: async (_, {id}, ctx) => {  
      return data.customers.find(cust => cust.id == id)  
    },  
    product: async (_, {id}, ctx) => {  
      return data.products.find(p => p.id == id)  
    },  
    order: async (_, {id}, ctx) => {  
      return data.orders.find(ord => ord.id == id)  
    }  
  },  
}
```

```
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

```
query {  
  order(id:1) {  
    id  
    invoiceAmount  
    products {  
      name  
    }  
    services {  
      name  
    }  
  }  
}
```

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

```
const resolvers = {  
  Query: {  
    customer: async (_, {id}, ctx) => {  
      return data.customers.find(cust => cust.id == id)  
    },  
    product: async (_, {id}, ctx) => {  
      return data.products.find(p => p.id == id)  
    },  
    order: async (_, {id}, ctx) => {  
      return data.orders.find(ord => ord.id == id)  
    }  
  },  
}
```

```
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: Used to read data
  - Mutation: Used to Edit/Create data
  - Subscriptions: Used for Real-Time data
  - Use Query language for all three.

```
1 query {  
2   customer(id: 1) {  
3     id  
4     name  
5     contactLanguage  
6     orders {  
7       id  
8       date  
9       products {  
10        id  
11        name  
12        document {  
13          documentUrl  
14        }  
15        images {  
16          imageUrl  
17        }  
18      }  
19    }  
20  }  
21 }
```

**GraphQL enables efficient data loading**

# GraphQL enables efficient data loading

- Get only what you ask for
  - Just the requested data is transferred through the network. No more, no less.
  - You only need one request to the backend no matter how you traverse the data graph
  - Multiple graph traversals can be combined into one request, reducing network delays
- As a result of the architecture:
  - Speed to features when data requirement changes
  - One can read the query in the frontend code and clearly understand what data comes back without having to be familiar with the backend.

```
1 query {  
2   customer(id:1){  
3     name  
4   }  
5   product(id:1){  
6     name  
7   }  
8 }
```

```
{  
  "data": {  
    "customer": {  
      "name": "TYSON CHICKEN LOUISIANA"  
    },  
    "product": {  
      "name": "S800"  
    }  
  }  
}
```

# GraphQL enables efficient data loading

PRODUCTS PLATFORMS INDUSTRIES NEWS & UPDATES SUPPORT

INTRALOX SELF-SERVE

ALL ORDERS CUSTOMER ORDERS OPEN ORDERS

MY DASHBOARD

MY CUSTOMERS

CASE CREATOR

CUSTOMER ORDERS

Order For **Superior Foods - Serbia**

Search For An Order

Related Documents

Purchase Order: 234897G Order Number: 234897G

Four sources of documents in 1 table

Document Name	Document Type	File Type	File Size
<a href="#">Invoice #234345</a>	Invoice	PDF	800kb
<a href="#">Invoice #345978</a>	Invoice	PDF	10kb
<a href="#">Delivery #234 Packing Slip</a>	Packing Slip	PDF	13kb
<a href="#">Delivery #234 Packing Slip</a>	Packing Slip	PDF	18kb
<a href="#">BPA Free Certificate</a>	General Document	PDF	138kb
<a href="#">Allergen Certificate</a>	General Document	PDF	13kb
<a href="#">Allergen Certificate</a>	General Document	PDF	59kb
<a href="#">Food Safe Certificate for Acetol</a>	Food Safe	PDF	420kb

© Intralox 2019

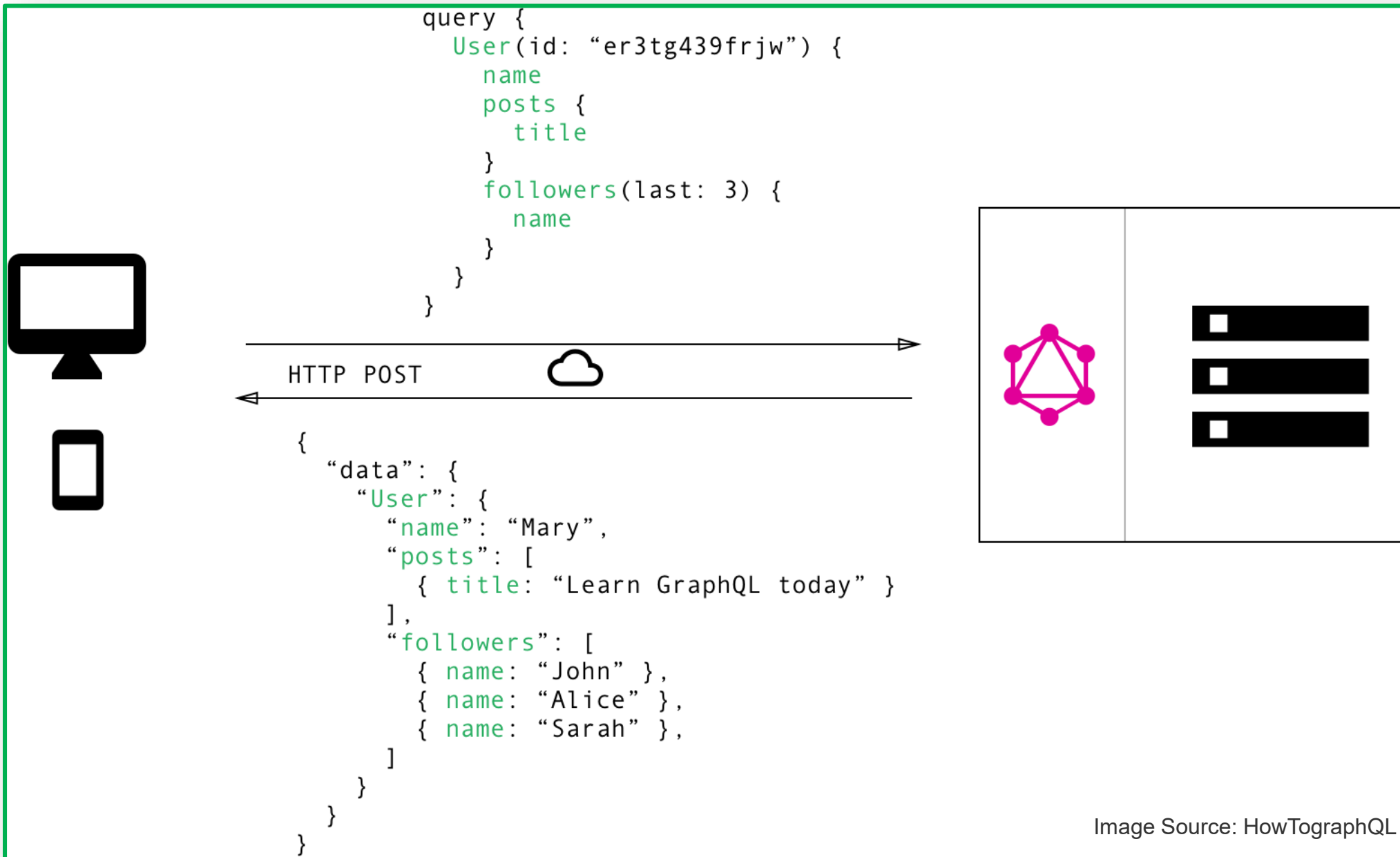
[Privacy Policy](#)

```
query {  
  invoices(orderNumber:"234897G"){  
    documentUrl  
    fileType  
    fileSize  
  }  
  packingSlips(orderNumber:"234897G"){  
    documentUrl  
    fileType  
    fileSize  
  }  
  generalDocuments {  
    documentUrl  
    fileType  
    fileSize  
  }  
  foodSafeCertificateDocument(Bom: {...}) {  
    documentUrl  
    fileType  
    fileSize  
  }  
}
```

# GraphQL enables efficient data loading



# GraphQL enables efficient data loading



# GraphQL enables efficient data loading

## REST Endpoints examples from Intralox

```
GET /api/feedback/beltconfigs/<id>  
GET /api/feedback/beltconfig/<id>/fealist/  
GET /api/feedback/beltconfig/<id>/feacheck/  
GET /api/feedback/beltconfig/<id>/loadcheck/
```

API from Simulation 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
```

API from Sensing Portal

---



**GraphQL enables Fine Grain Control**

# 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": [{"line": 10, "column": 10}],  
      "path": [{"line": 10, "column": 10}]  
    }  
  ]  
}
```

# 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 is Self Documenting**

PRETTIFYHISTORY

● http://localhost:4099/

```
1 query {
2   customer(id: 1) {
3     id
4     name
5     contactLanguage
6     orders {
7       id
8       date
9       products {
10        id
11        name
12        document {
13          documentUrl
14        }
15        images {
16          imageUrl
17        }
18      }
19    }
20  }
21 }
```

DOCS

SCHEMA

🔍 Search the docs ...

QUERIES

customer(...): Customer ▶

product(...): Product ▶

MUTATIONS

createOrder(...): Order ▶

customer(  
 id: ID  
): Customer

TYPE DETAILS

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

ARGUMENTS

id: ID ▶

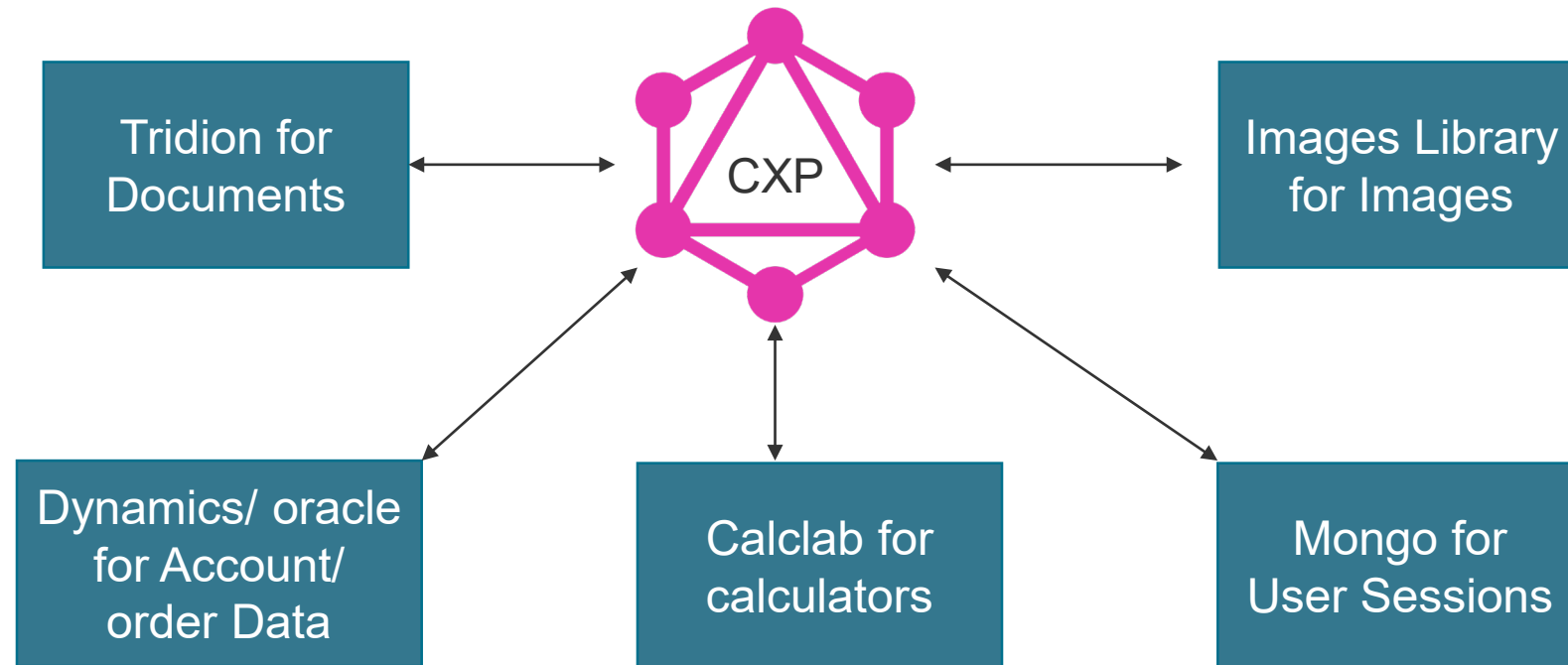
orders: [Order]

TYPE DETAILS

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

**GraphQL is a perfect fit for  
Complex Interconnected Systems**

# GraphQL enables easy ways to combine different data sources





# Real Demo + Questions?

- Demo Topics
    - Schema Review
    - Resolver Review
    - Quick chat about permissions, authorization etc.
-

# 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
-

# GraphQL enables more efficient caching

- With GraphQL, each Node is cached which creates efficiencies in caching
    - Compare with REST where resources are cached based on URL
  - Consideration:
    - Node level caching is generally more complex to implement than URL level caching
    - Luckily the community has already developed lots of libraries for caching.
-

# Consideration with GraphQL

- Simple web apps with 1 data source and limited API
  - Mature applications where REST has been working, and there are no plans for extending the application.
  - Applications where performance efficiency is the most important criteria of success
    - Especially important if you have one data source with complex queries
-