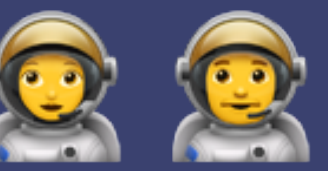
Let's build the future of forms with GraphQL 

#whoami

Charly POLY - Senior Software Engineer at algolia

🔗 Writing about Software Eng. on honest.engineering

 Writing about TypeScript and GraphQL on Medium

@whereischarly

Forms on the web ecosystems

Forms on the web ecosystems

- Subject tackled by most ecosystems
  - Rails simple_form
  - Symfony Forms
  - ASP.NET MVC

Forms on the web ecosystems

- Subject tackled by most ecosystems
  - Rails simple_form
  - Symfony Forms
  - ASP.NET MVC
- Form as a Service
  - TypeForm
  - Google Forms

Forms on the web ecosystems

- Subject tackled by most ecosystems
  - Rails simple_form
  - Symfony Forms
  - ASP.NET MVC
- Form as a Service
  - TypeForm
  - Google Forms

> *What about SPA, JavaScript ecosystem forms?*

Role of Forms in a Web Application

> *Forms handle the experience that user have with data*

Role of Forms in a Web Application

> *Forms handle the experience that user have with data*

Developer Experience

- Be flexible, customisable
- Avoid validation divergence with back-end
- Handle state

Role of Forms in a Web Application

> *Forms handle the experience that user have with data*

Developer Experience

- Be flexible, customisable
- Avoid validation divergence with back-end
- Handle state

User Experience

- Be the most precise as possible
- Make is easy and fast to fill complex informations
- Immediate feedback

Role of Forms in a Web Application

> *Forms handle the experience that user have with data*

Developer Experience

- Be flexible, customisable
- Avoid validation divergence with back-end
- Handle state

User Experience

- Be the most precise as possible
- Make is easy and fast to fill complex informations
- Immediate feedback

In short, fast and smooth experience

Current state of JS forms building

Vue.js forms | redux-form | Formik

Benchmark: Vue.js forms

Benchmark: Vue.js forms

What it solves:

- smart binding
- state related features (*lazy, numbers and debounce attributes*)
- (validations using vee-validate)

Benchmark: Vue.js forms

What it solves:

- smart binding
- state related features (*lazy, numbers and debounce attributes*)
- (validations using vee-validate)

What it doesn't solve:

- almost everything is done by hand ✍️
  - components, fields, state
- no Form state management
- no easily reusable ☠️
- non-standard validation format ☠️

Benchmark: redux-form

> *The best way to manage your form state in Redux.*

Benchmark: redux-form

> *The best way to manage your form state in Redux.*

What it solves:

- Error management
- Form state management
- Built-in components and helpers

Benchmark: redux-form

> *The best way to manage your form state in Redux.*

What it solves:

- Error management
- Form state management
- Built-in components and helpers

What it doesn't solve:

- lot of configuration ✍️
- validators by hand ✍️
- forms state in global state ☠️
- Field by Field building workflow ✍️

Benchmark: Formik

> *Build forms in React, without the tears* 😭

Benchmark: Formik

> *Build forms in React, without the tears* 😭

What it solves:

- Object-schema based validations with Yup
- Form state management (not global)
- "Functional" state management + render props pattern
- Built-in components and helpers

Benchmark: Formik

> *Build forms in React, without the tears* 😭

What it solves:

- Object-schema based validations with Yup
- Form state management (not global)
- "Functional" state management + render props pattern
- Built-in components and helpers

What it doesn’t solve:

- lot of configuration ✍️
- validators by hand ✍️
- non-standard validation format ☠️
- Field by Field building workflow ✍️

Benchmark results

Developer Experience

	Vue.js forms	redux-form	Formik
Manage state	✍️	✅	✅
Data validation	✍️	✍️ / ✅	✅
Theming	✍️	✍️	✍️
Error messages	✍️	✍️	✍️
Fields conf.	✍️	✍️	✍️

Benchmark results

Architecture

	Vue.js forms	redux-form	Formik
Productivity	⭐⭐	⭐⭐⭐⭐	⭐⭐⭐⭐
Performance	⭐⭐⭐⭐	⭐⭐⭐	⭐⭐⭐⭐
Flexibility	⭐⭐⭐⭐	⭐⭐⭐	⭐⭐⭐⭐
Components Reusability	⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐⭐⭐
Validation Isomorphism	*nodejs-only*	*nodejs-only*	*nodejs-only*

Observation

- **No matters the solution**, we have to duplicate:
  - fields list and definition
  - data validations

What about types?

The JavaScript eco-system new era:

- TypeScript: typed JavaScript
- GraphQL: typed data exchange

What about types?

The JavaScript eco-system new era:

- TypeScript: typed JavaScript
- GraphQL: typed data exchange

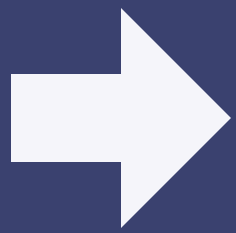
“ Since forms are about data,
why not build typed forms?

Feedback: Dropping redux-form

```
class Form extends ModuleForm {
    fields: FieldsDefinitions = {
        id: 'none',
        email: 'none',
        picture_path: {
            type: 'image', transformations: 'h_200,w_200,r_max,c_fill'
        },
        first_name: 'string*',
        last_name: 'string*',
        username: 'string*',
        job_title: 'string',
        company_name: 'string',
        language: {
            type: 'select*',
            component: LanguageSelectView,
            valueProperty: 'code',
            values: supportedLanguages,
            moduleName: 'attachment' // TODO: fix.
        }
    };

    constructor() {
        super('UserForm', 'user');
    }
}
```

Feedback: Dropping redux-form

```
class Form extends ModuleForm {
    fields: FieldsDefinitions = {
        id: 'none',
        email: 'none',
        picture_path: {
            type: 'image', transformations: 'h_200,w_200,r_max,c_fill'
        },
        first_name: 'string*',
        last_name: 'string*',
        username: 'string*',
        job_title: 'string',
        company_name: 'string',
        language: {
            type: 'select*',
            component: LanguageSelectView,
            valueProperty: 'code',
            values: supportedLanguages,
            moduleName: 'attachment' // TODO: fix.
        }
    };

    constructor() {
        super('UserForm', 'user');
    }
}
```



configured
redux-form

Solution: Mozilla react-jsonschema-form

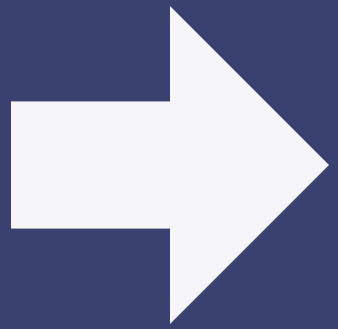
> *A React component for building Web forms from JSON Schema.*

Mozilla react-jsonschema-form

What is JSON Schema?

- "A Media Type for Describing JSON Documents"

```
interface Todo {
    id: String;
    name: String;
    completed: Boolean;
    color: Color;
}
```



```
{
    $schema: 'http://json-schema.org/draft-06/schema#',
    properties: {
        Todo: {
            type: 'object',
            properties: {
                id: { type: 'string' },
                name: { type: 'string' },
                completed: { type: 'boolean' },
                color: { $ref: '#/definitions/Color' },
            },
            required: ['id', 'name']
        }
    }
}
```

Mozilla react-jsonschema-form

✔ JSONSchema

```
{
  "title": "A registration form",
  "description": "A simple form example.",
  "type": "object",
  "required": [
    "firstName",
    "lastName"
  ],
  "properties": {
    "firstName": {
      "type": "string",
      "title": "First name"
    },
    "lastName": {
      "type": "string",
```

✔ UISchema

```
{
  "firstName": {
    "ui:autofocus": true
  },
  "age": {
    "ui:widget": "updown"
  },
  "bio": {
    "ui:widget": "textarea"
  },
  "password": {
    "ui:widget": "password",
    "ui:help": "Hint: Make it strong!"
  },
```

✔ formData

```
{
  "firstName": "Chuck",
  "lastName": "Norris",
  "age": 75,
  "bio": "Roundhouse kicking asses since 1940",
  "password": "noneed"
}
```

A registration form

A simple form example.

First name*

Chuck

Last name*

Norris

Age

75

Bio

Roundhouse kicking asses since 1940

Password

••••••

Hint: Make it strong!

Submit

Mozilla react-jsonschema-form

Mozilla react-jsonschema-form

1. Provide a JSON Schema **describing the data**

Mozilla react-jsonschema-form

1. Provide a JSON Schema **describing the data**
2. Provide a JSON Schema **describing the UI**

Mozilla react-jsonschema-form

1. Provide a JSON Schema **describing the data**
2. Provide a JSON Schema **describing the UI**
3. Provide initial data

Mozilla react-jsonschema-form

1. Provide a JSON Schema **describing the data**
2. Provide a JSON Schema **describing the UI**
3. Provide initial data

Mozilla react-jsonschema-form

1. Provide a JSON Schema **describing the data**
2. Provide a JSON Schema **describing the UI**
3. Provide initial data

Form React component

Mozilla react-jsonschema-form

Architecture

	react-jsonschema-form
Productivity	⭐⭐
Performance	⭐⭐⭐⭐
Flexibility	⭐⭐
Components Reusability	⭐⭐
Validation Isomorphism	⭐⭐

Developer Experience

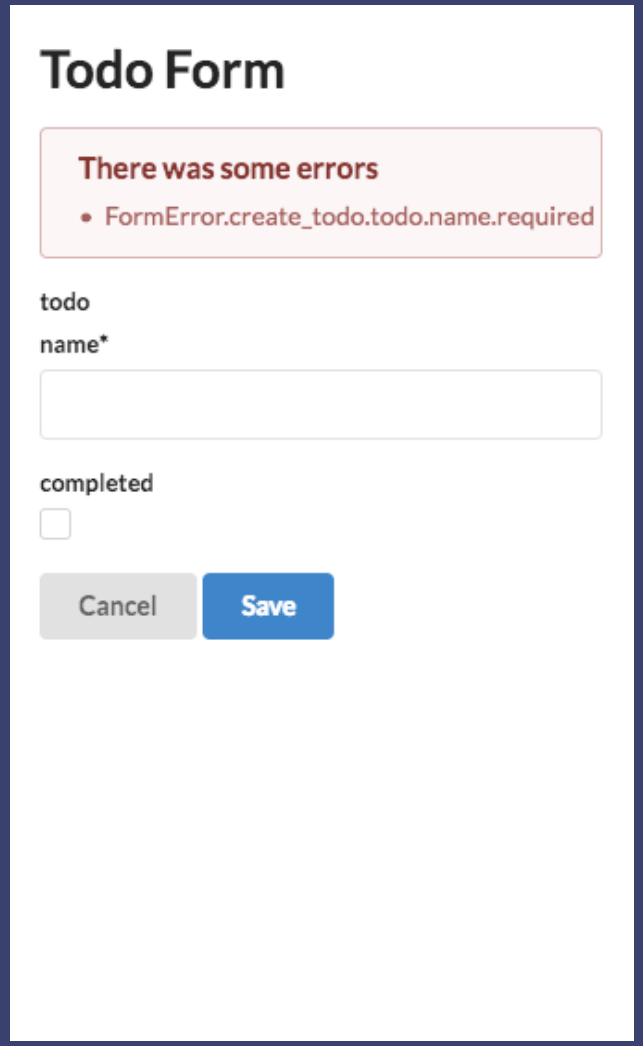
	react-jsonschema-form
Manage state	✅
Data validation	✍️
Theming	✅
Error messages	✅
Fields conf.	✍️

<ApolloForm> 👨‍🚀 👨‍🚀

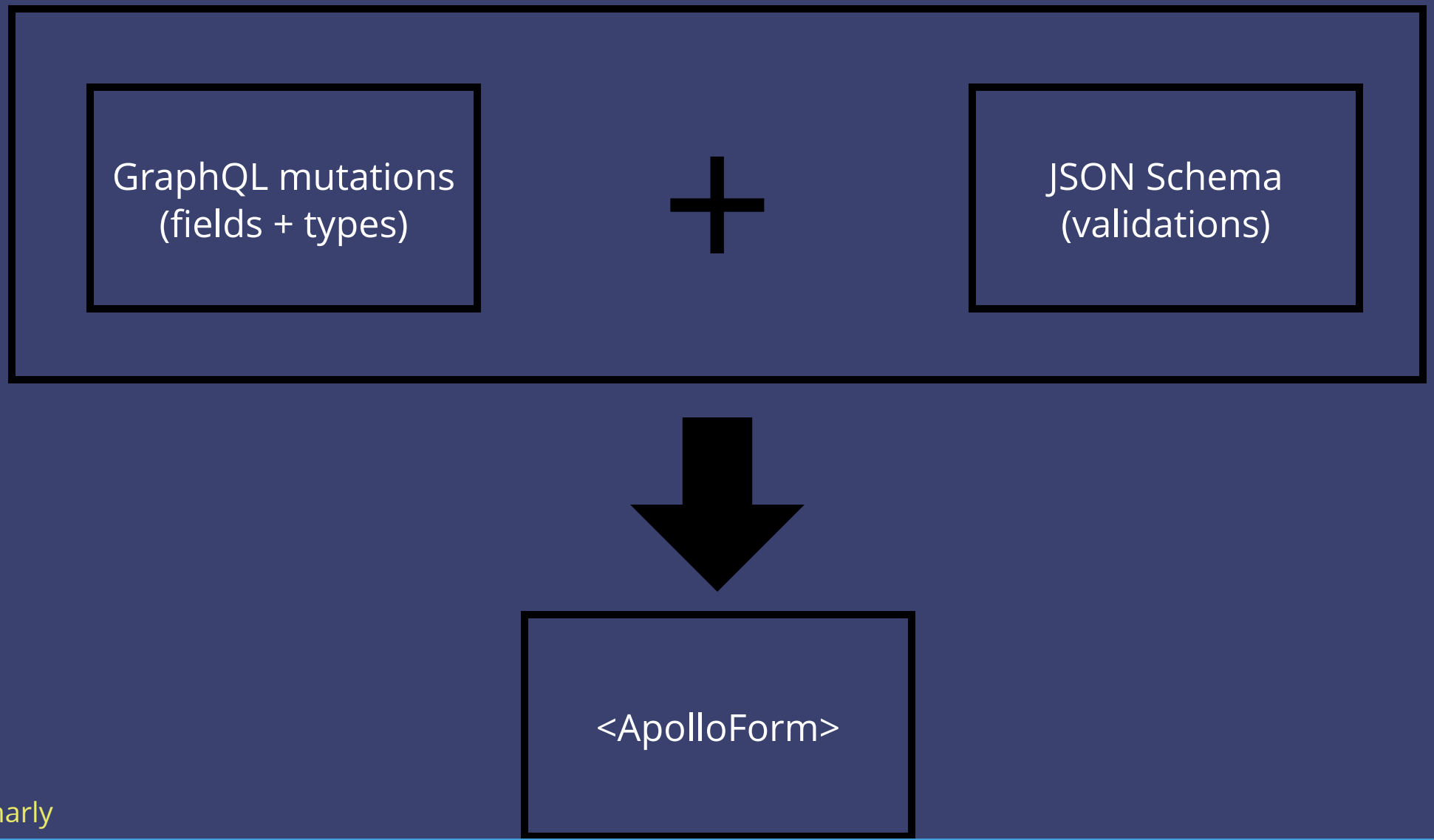
```
import * as React from 'react';
import gql from 'graphql-tag';
import { configure } from 'react-apollo-form';
import { client } from './apollo';
import { applicationFormTheme } from './core/forms/themes/application';


const jsonSchema = require('./core/apollo-form-json-schema.json');

export const ApplicationForm = configure<ApolloFormMutationNames>({
    // tslint:disable-next-line:no-any
    client: client as any,
    jsonSchema,
    theme: applicationFormTheme
});

<ApplicationForm
    config={{
        mutation: {
            name: 'create_todo',
            document: gql`mutation {...}`
        }
    }}
    data={{}}
/>
```

<ApolloForm> 👨‍🚀 👨‍🚀

```
import * as React from 'react';
import gql from 'graphql-tag';
import { configure } from 'react-apollo-form';
import { client } from './apollo';
import { applicationFormTheme } from './core/forms/themes/application';


const jsonSchema = require('./core/apollo-form-json-schema.json');

export const ApplicationForm = configure<ApolloFormMutationNames>({
    // tslint:disable-next-line:no-any
    client: client as any,
    jsonSchema,
    theme: applicationFormTheme
});

<ApplicationForm
    config={{
        mutation: {
            name: 'create_todo',
            document: gql`mutation {...}`
        }
    }}
    data={{}}
/>
```



<ApolloForm> 👩‍🚀 👨‍🚀


GraphQL mutations
(fields + types)
+
JSON Schema
(validations)
<ApolloForm>

<ApolloForm> flexible API

- Schema options
- Theming options
- Error messages
- Callbacks
- Render props

```
import * as React from 'react';
import gql from 'graphql-tag';
import { configure } from 'react-apollo-form';
import { client } from './apollo';
import { applicationFormTheme } from './core/forms/themes/application';


const jsonSchema = require('./core/apollo-form-json-schema.json');

export const ApplicationForm = configure<ApolloFormMutationNames>({
    // tslint:disable-next-line:no-any
    client: client as any,
    jsonSchema,
    theme: applicationFormTheme
});

<ApplicationForm
    config={
        /* schema options */
    }
    data={{
        /* data */
    }}
    ui={
        /* UI options */
    }
/>
```

<ApolloForm> flexible API

Schema options

<ApolloForm> flexible API

Schema options

- "Manual mode"

```
<ApplicationForm
    title={'Todo Form'}
    config={{
        name: 'todo',
        schema: schema({
            todo: {
                name: types.type('string', { required: true
                completed: types.type('boolean')
            }
        }),
        saveData: data => {
            console.log('save !', data);
        }
    }}
    data={{}}
    ui={{}}
/>
```

<ApolloForm> flexible API

Schema options

- "Mutation mode"

```
<ApplicationForm
    config={{
        mutation: {
            name: 'create_todo',
            document: gql`mutation {...}`
        }
    }}
    data={{}}
/>
```

<ApolloForm> flexible API

Schema options

- "Mutation mode"

```
<ApplicationForm
    config={{
        mutation: {
            name: 'create_todo',
            document: gql`mutation {...}`
        },
        ignoreFields: ['user.image'],
        updateFields: {
          'user.phone_number': { pattern: '(^[0-9\+]{5,}$)|(^[a-zA-Z0-9]{0}$)' }
        },
        requiredFields: ['user.email']
    }}
    data={{}}
/>
```

<ApolloForm> flexible API

Render props

```
export interface ApolloRenderProps {
    // renderers
    header: () => React.ReactNode; // render the form header (title by default)
    form: () => React.ReactNode; // render the inputs
    buttons: () => React.ReactNode; // render save and cancel buttons
    saveButton: () => React.ReactNode; // render save button
    cancelButton: () => React.ReactNode; // render cancel button
    // actions
    cancel: () => void; // trigger a cancel
    save: (args: any) => void; // trigger a save
    // state
    isDirty: boolean;
    isSaved: boolean;
    hasError: boolean;
    data: any;
}
```

```
<ApplicationForm
   /* props options */
>
    {
        form => (
            <div
                style={{
                    backgroundColor: '#FFF',
                    padding: '20px'
                }}>
                <h2>
                    My form!
                </h2>
                <br />
                {form.form()}
                {form.saveButton()}
            </div>
        )
    }
</ApplicationForm>
```

<ApolloForm> flexible API

Error messages

```
type ApolloFormUi = {
    // default: false, should we display errors list at the top ?
    showErrorsList?: boolean;
    // default: true, should we display errors at field level ?
    showErrorsInline?: boolean;
    // you can provide a custom component to display error list
    errorListComponent?: ErrorListComponent;
};
```

By default, <ApolloForm> do not display errors.

To enable it, you should provide some options to ui prop.

<ApolloForm> flexible API

Theming

```
interface ApolloFormConfigureTheme {
    templates?: ApolloFormTheme['templates'];
    widgets?: ApolloFormTheme['widgets'];
    fields?: ApolloFormTheme['fields'];
    renderers?: Partial<ApolloFormTheme['renderers']>;
}
```

- Templates render Fields and Widgets
- Renderers are <ApolloForm> specific:
  - <saveButton>
  - <cancelButton>
  - <header>

<ApolloForm> internals

<ApolloForm> internals

Build tools

At runtime

<ApolloForm> internals

Build tools

GraphQL Schema

At runtime

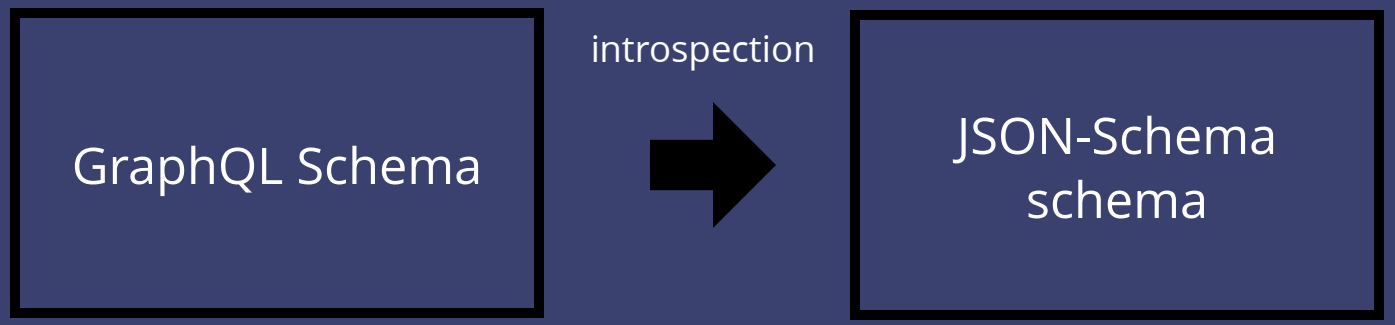
<ApolloForm> internals

Build tools

At runtime

<ApolloForm> internals

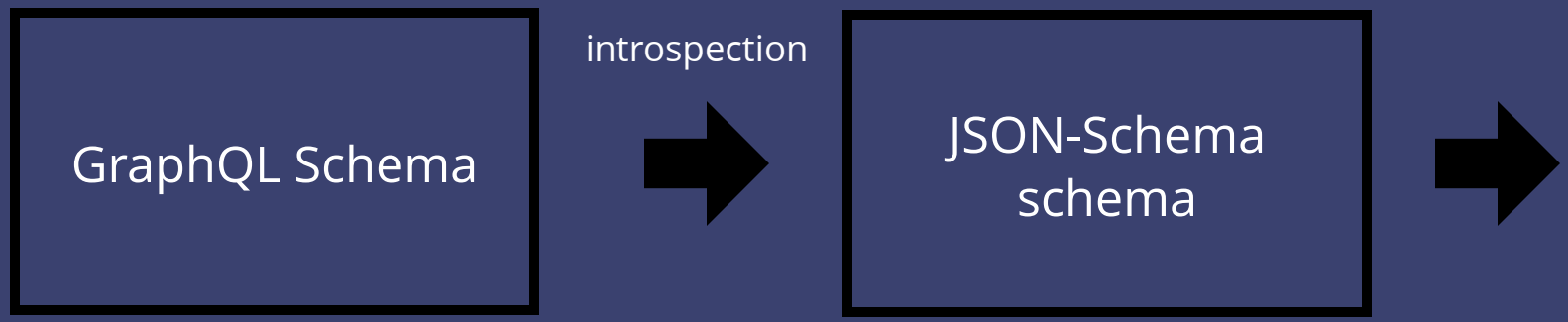
Build tools



At runtime

<ApolloForm> internals

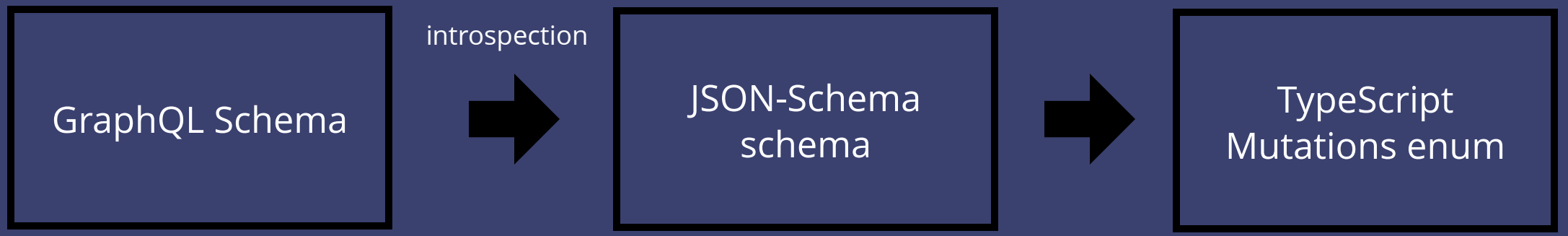
Build tools



At runtime

<ApolloForm> internals

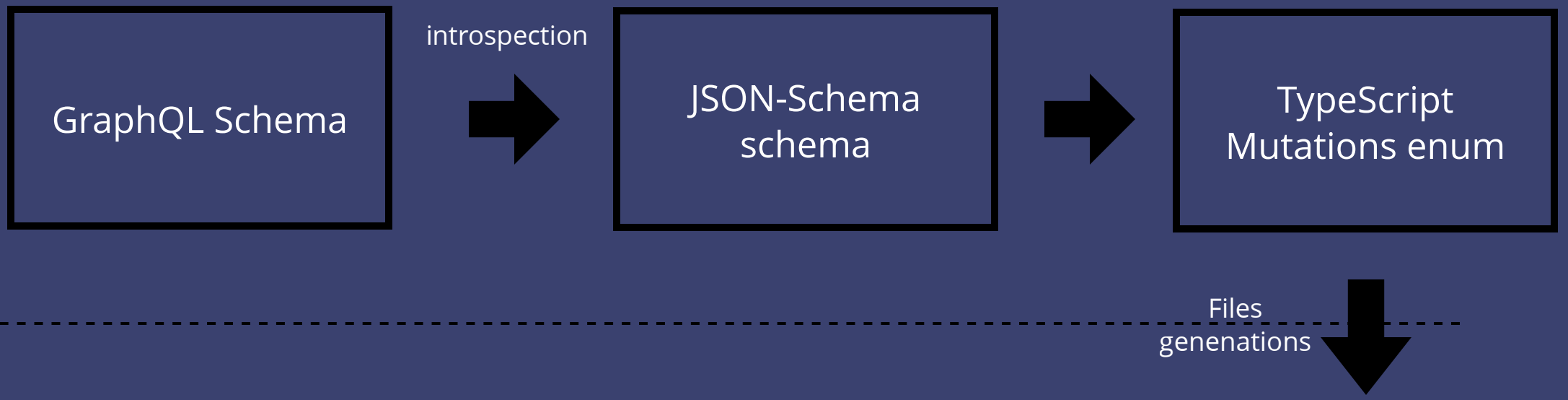
Build tools



At runtime

<ApolloForm> internals

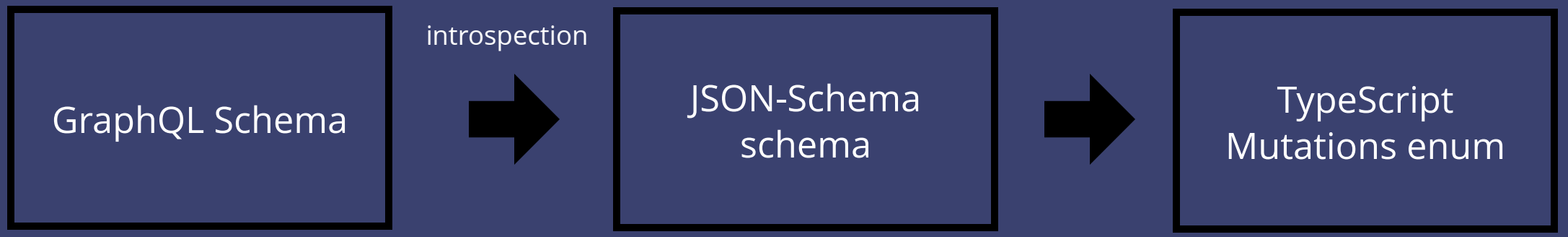
Build tools



At runtime

<ApolloForm> internals

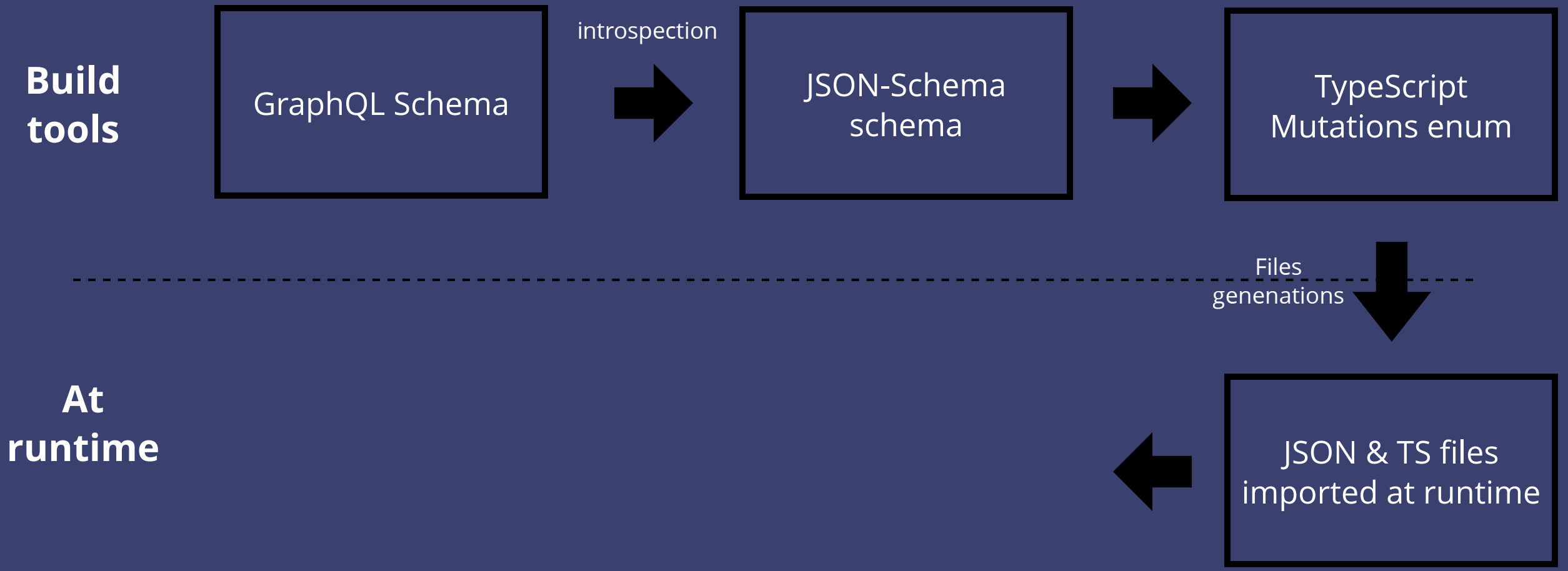
Build tools



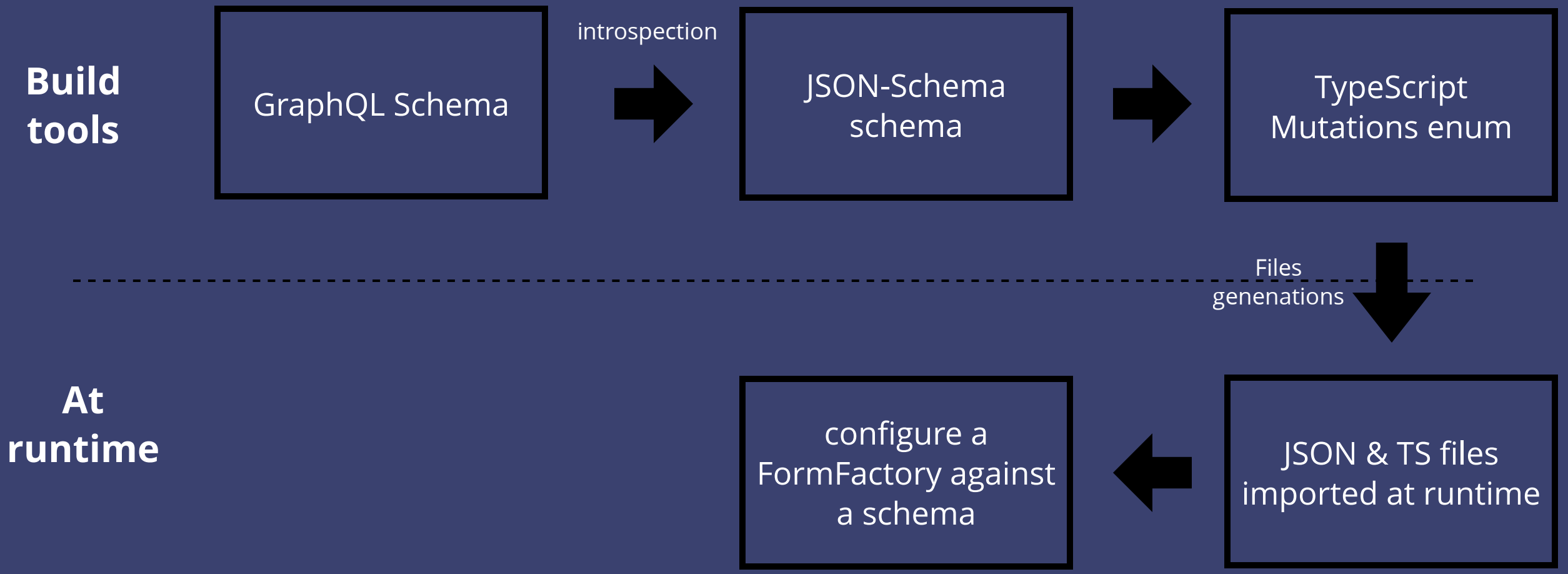
At runtime

JSON & TS files imported at runtime

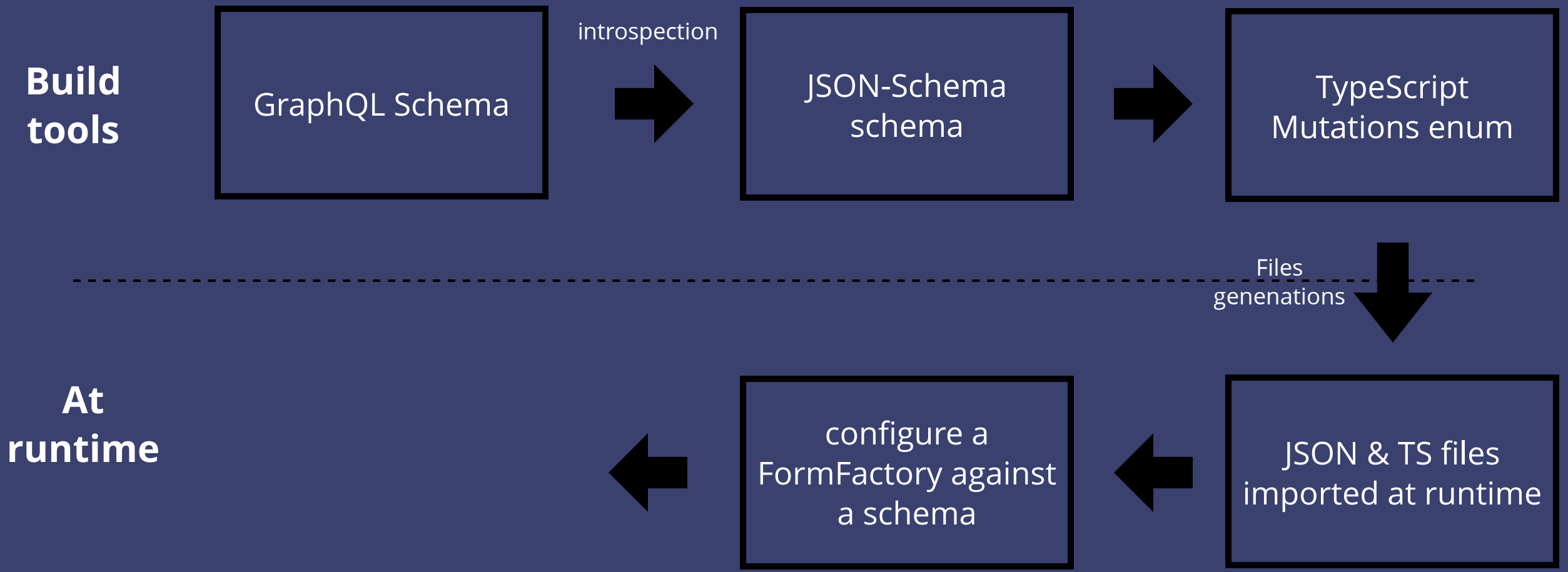
<ApolloForm> internals



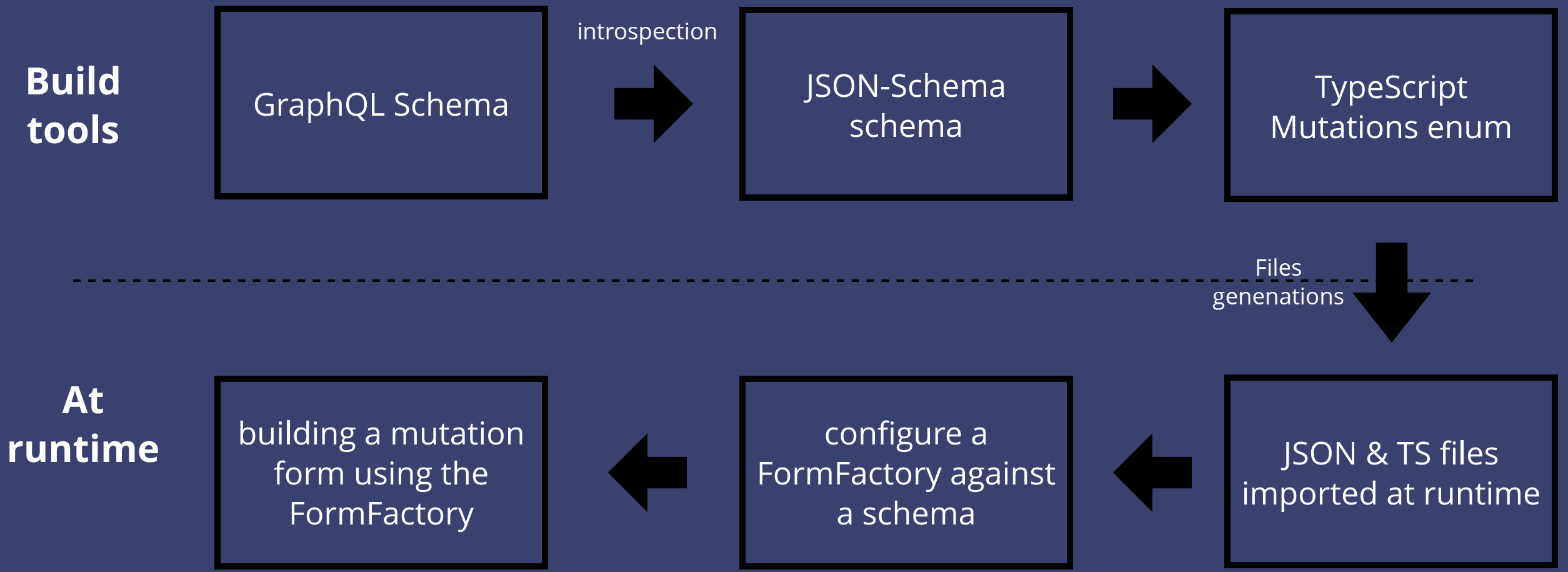
<ApolloForm> internals



<ApolloForm> internals

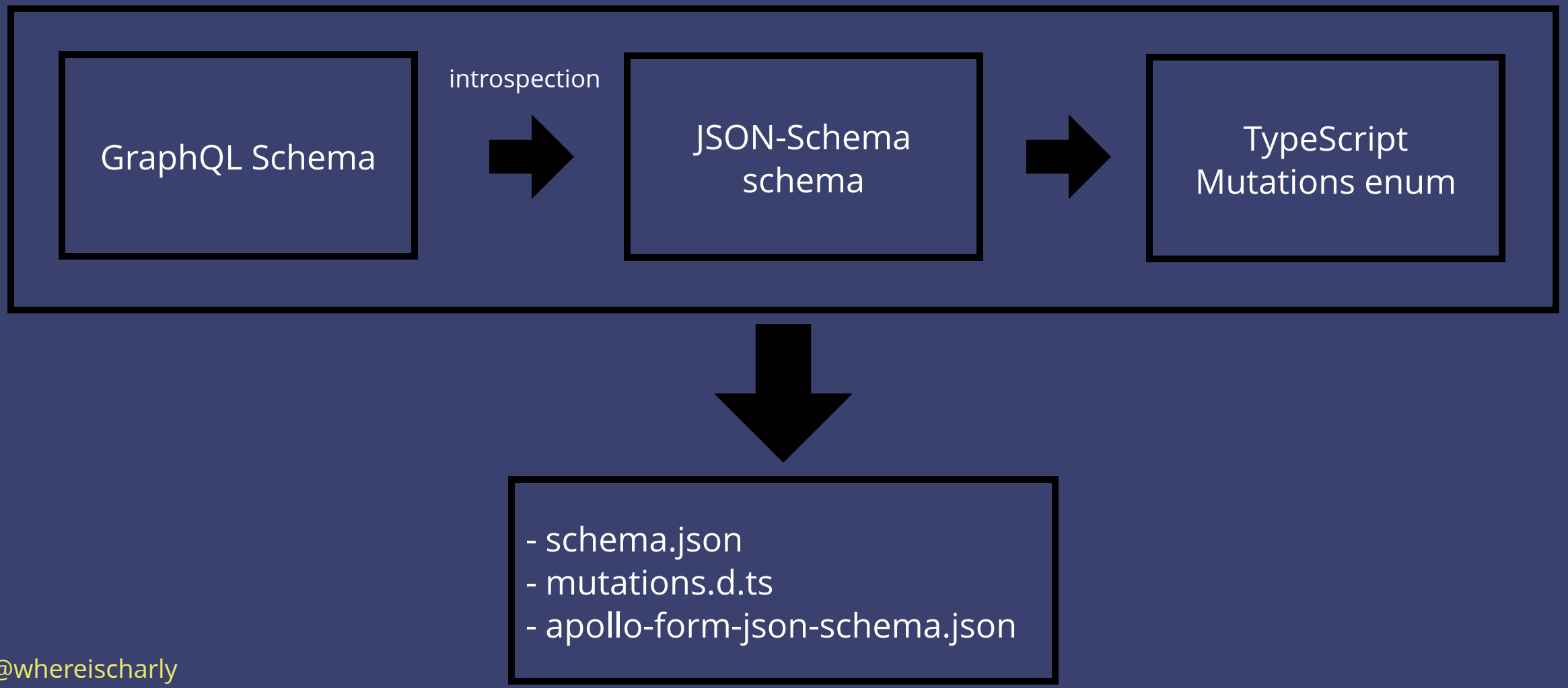


<ApolloForm> internals



<ApolloForm> internals : Build tools

```
react-apollo-form fetch-mutations <graphqlEndpoint> <outpurDir>
```



<ApolloForm> internals : Build tools

What is an introspection query?

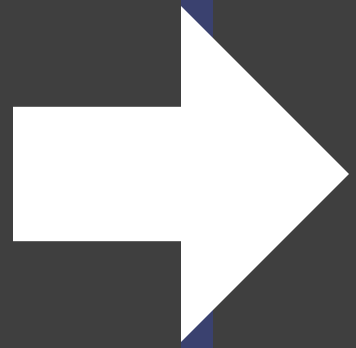
```
query IntrospectionQuery {
      __schema {
        queryType { name }
        mutationType { name }
        subscriptionType { name }
        types {
          ...FullType
        }
        directives {
          name
          description
          locations
          args {
            ...InputValue
          }
        }
      }
    }
    fragment FullType on __Type {
      kind
      name
      description
      fields(includeDeprecated: true) {
        name
        description
        args {
          ...InputValue
        }
        type {
          ...TypeRef
        }
        isDeprecated
        deprecationReason
      }
      inputFields {
        ...InputValue
```

```
{
  "data": {
    "__schema": {
      "queryType": {
        "name": "Query"
      },
      "mutationType": null,
      "subscriptionType": null,
      "types": [
        {
          "kind": "OBJECT",
          "name": "Query",
          "description": "the schema allows the following query:",
          "fields": [
            {
              "name": "me",
              "description": "",
              "args": [],
              "type": {
                "kind": "OBJECT",
                "name": "PrivateUser",
                "ofType": null
              },
              "isDeprecated": false,
              "deprecationReason": null
            },
            {
              "name": "user",
              "description": "",
              "args": [
                {
                  "name": "id",
                  "description": "",
                  "type": {
                    "kind": "NON_NULL",
                    "name": null,
                    "ofType": {
```

<ApolloForm> internals : Build tools

graphql-2-json-schema package

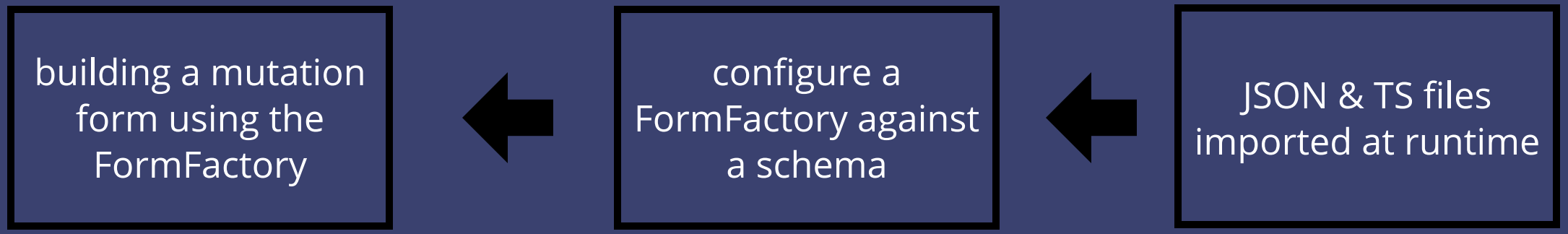
```
type Todo {
    id: String!
    name: String!
    completed: Boolean
    color: Color
}

input TodoInputType {
    name: String!
    completed: Boolean
    color: Color
}

type Mutation {
    update_todo(id: String!, todo: TodoInputType!): Todo
    create_todo(todo: TodoInputType!): Todo
}
```



```
{
    $schema: 'http://json-schema.org/draft-06/schema#',
    properties: {
        Mutation: {
            type: 'object',
            properties: {
                update_todo: {
                    type: 'object',
                    properties: {
                        arguments: {
                            type: 'object',
                            properties: {
                                id: { type: 'string' },
                                todo: { $ref: '#/definitions/
                            },
                            required: ['id', 'todo']
                        },
                        return: {
                            $ref: '#/definitions/Todo'
                        }
                    },
                    required: []
                },
                // ...
            }
        },
    },
    definitions: {
        'Todo': {
            type: 'object',
```

@whereischarly

<ApolloForm> internals : at runtime



<ApolloForm> internals : at runtime

```
import * as React from 'react';
import gql from 'graphql-tag';
import { configure } from 'react-apollo-form';
import { client } from './apollo';
import { applicationFormTheme } from './core/forms/themes/application';


const jsonSchema = require('./core/apollo-form-json-schema.json');

export const ApplicationForm = configure<ApolloFormMutationNames>({
    // tslint:disable-next-line:no-any
    client: client as any,
    jsonSchema,
    theme: applicationFormTheme
});

<ApplicationForm
    config={{
        mutation: {
            name: 'create_todo',
            document: gql`mutation {...}`
        }
    }}
    data={{}}
/>
```

<ApolloForm> internals : at runtime

```
import * as React from 'react';
import gql from 'graphql-tag';
import { configure } from 'react-apollo-form';
import { client } from './apollo';
import { applicationFormTheme } from './core/forms/themes/application';


const jsonSchema = require('./core/apollo-form-json-schema.json');

export const ApplicationForm = configure<ApolloFormMutationNames>({
    // tslint:disable-next-line:no-any
    client: client as any,
    jsonSchema,
    theme: applicationFormTheme
});

<ApplicationForm
    config={{
        mutation: {
            name: 'create_todo',
            document: gql`mutation {...}`
        }
    }}
    data={{}}
/>
```

1. import definitions files

<ApolloForm> internals : at runtime

```
import * as React from 'react';
import gql from 'graphql-tag';
import { configure } from 'react-apollo-form';
import { client } from './apollo';
import { applicationFormTheme } from './core/forms/themes/application';


const jsonSchema = require('./core/apollo-form-json-schema.json');

export const ApplicationForm = configure<ApolloFormMutationNames>({
    // tslint:disable-next-line:no-any
    client: client as any,
    jsonSchema,
    theme: applicationFormTheme
});

<ApplicationForm
    config={{
        mutation: {
            name: 'create_todo',
            document: gql`mutation {...}`
        }
    }}
    data={{}}
/>
```

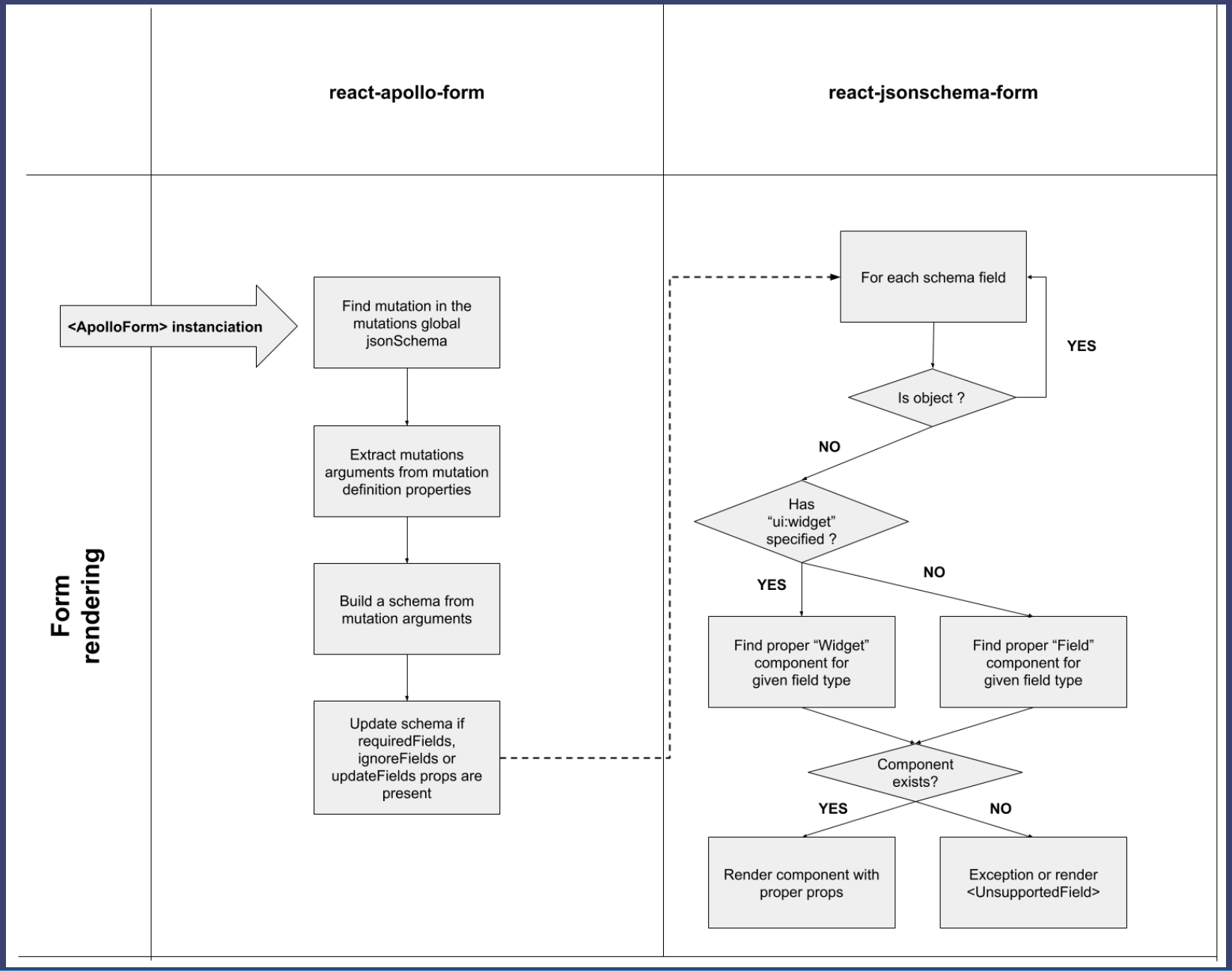
1. import definitions files
2. configure a Form Builder

<ApolloForm> internals : at runtime

```
import * as React from 'react';
import gql from 'graphql-tag';
import { configure } from 'react-apollo-form';
import { client } from './apollo';
import { applicationFormTheme } from './core/forms/themes/application';


const jsonSchema = require('./core/apollo-form-json-schema.json');

export const ApplicationForm = configure<ApolloFormMutationNames>({
    // tslint:disable-next-line:no-any
    client: client as any,
    jsonSchema,
    theme: applicationFormTheme
});

<ApplicationForm
    config={{
        mutation: {
            name: 'create_todo',
            document: gql`mutation {...}`
        }
    }}
    data={{}}
/>
```

1. import definitions files
2. configure a Form Builder
3. Instantiate a Form

<ApolloForm> internals : at runtime



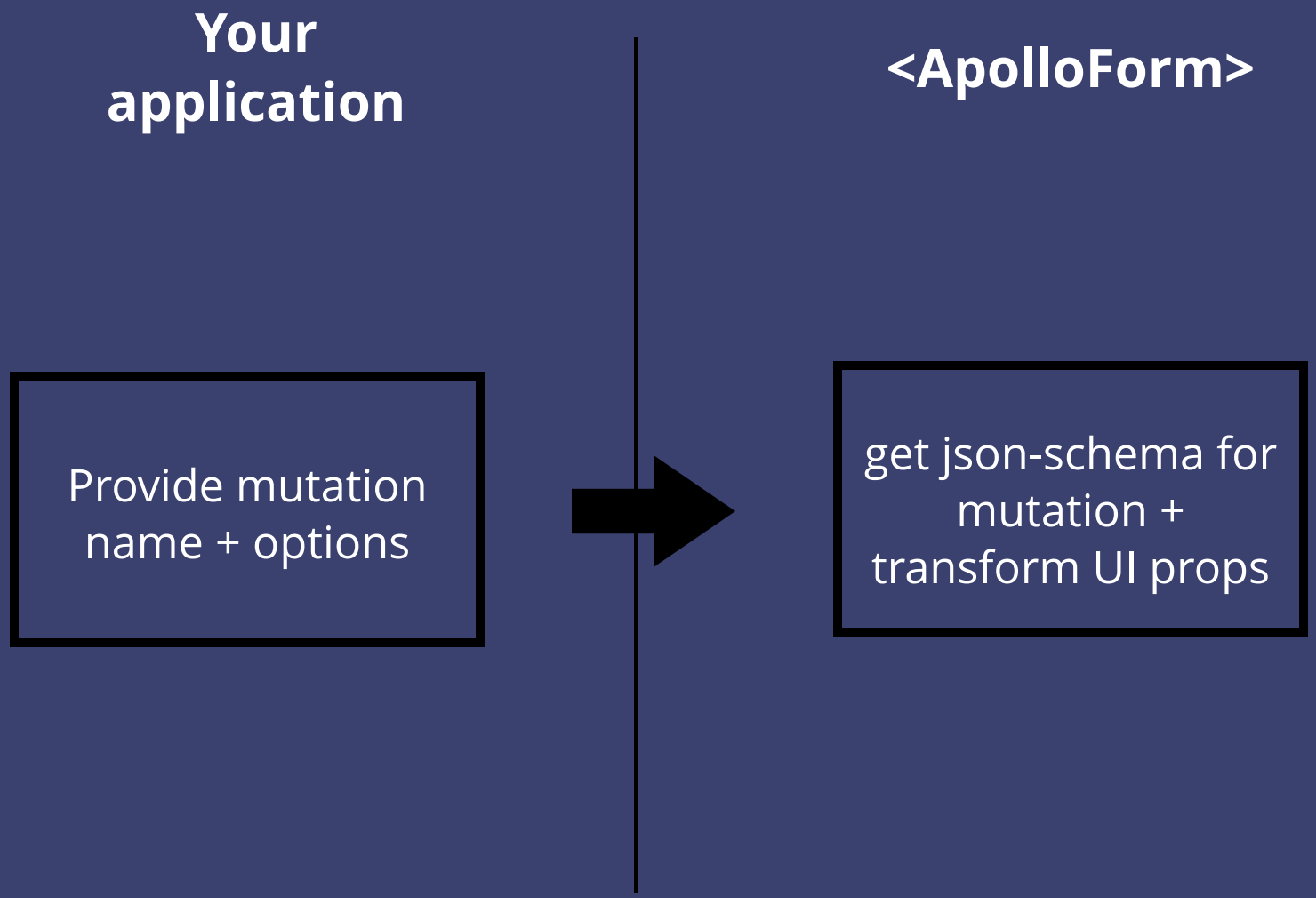
<ApolloForm> internals : at runtime

<ApolloForm> internals : at runtime

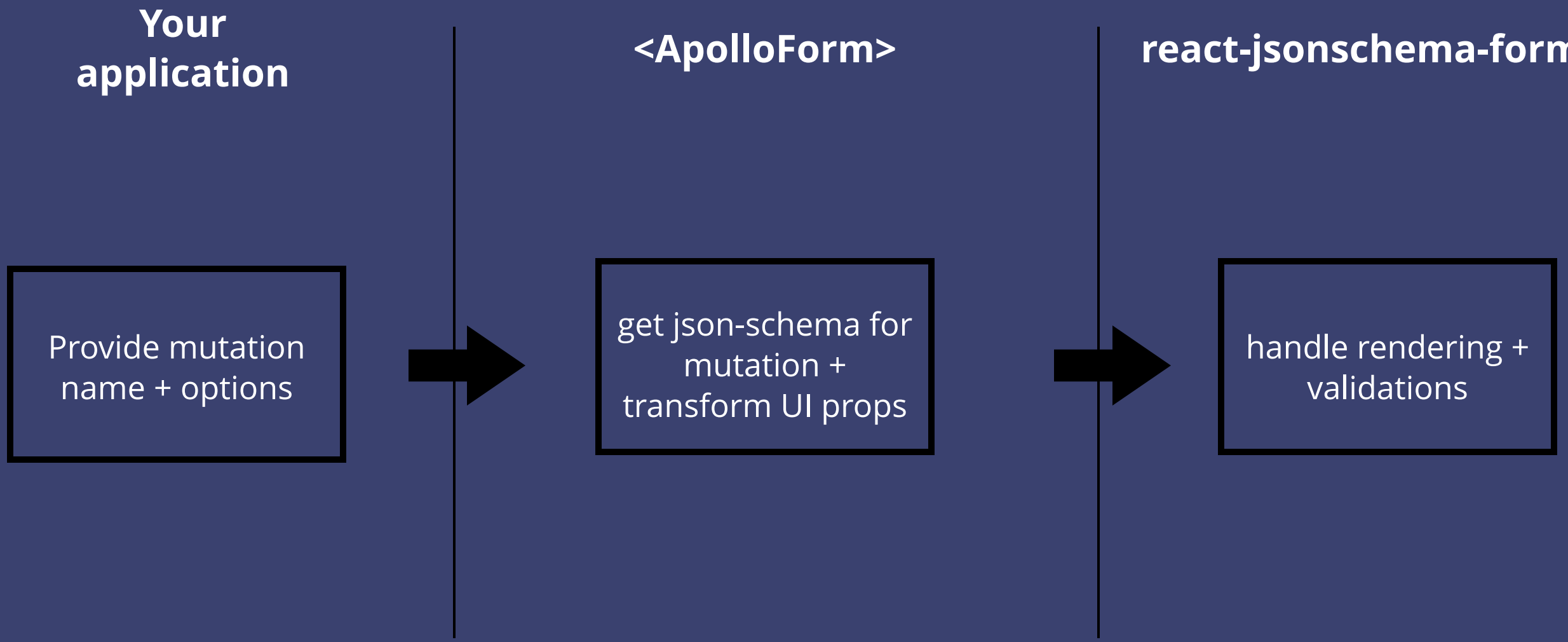
Your application

Provide mutation name + options

<ApolloForm> internals : at runtime



<ApolloForm> internals : at runtime



<ApolloForm> benchmark

<ApolloForm> benchmark

- react-jsonschema-form provides:
  - **Extendable,** with the fields and widgets system.
  - **State management and validation,**
    handled with local state and local validation using ajv.

<ApolloForm> benchmark

- react-jsonschema-form provides:
  - **Extendable,** with the fields and widgets system.
  - **State management and validation,** handled with local state and local validation using ajv.

- <ApolloForm> bring the last 2 key features:
  - **API/Client Interoperability:** generate the form JSON Schema from a mutation definition
  - **Simplicity:** provide some helpers to avoid the complexity of JSON Schema and enhance the extensibility of react-jsonschema-form.

Conclusion

Developer Experience

	Vue.js forms	redux-form	Formik	react-jsonschema-form	<ApolloForm>
Manage state	✍️	✅	✅	✅	✅
Data validation	✍️	✍️	✅	✅	✅
Theming	✍️	✍️	✍️	✅	✅
Error messages	✍️	✍️	✍️	✅	✅
Fields conf.	✍️	✍️	✍️	✍️	✅

Conclusion

Architecture

	Vue.js forms	redux-form	Formik	react-jsonschema-form	<ApolloForm>
Productivity	★★★	★★★★	★★★★	★★	★★★★
Performance	★★★★	★★★	★★★★	★★★★	★★★★
Flexibility	★★★★	★★★	★★★★	★★	★★★★
Components Reusability	★★★★	★★★★	★★★★	★★	★★★★
Validation Isomorphism	*nodejs-only*	*nodejs-only*	*nodejs-only*	★	★★★★

Thanks for listening!

Still in beta, looking for collaborators ➡️ react-apollo-form

🔗 honest.engineering

 @whereischarly

 /wittydeveloper