



Université 
de Montréal
et du monde.

React - Code

IFT3225 – Technologie de l'Internet

Jamel Eddine Jridi

Steve Lévesque

Table des Matières

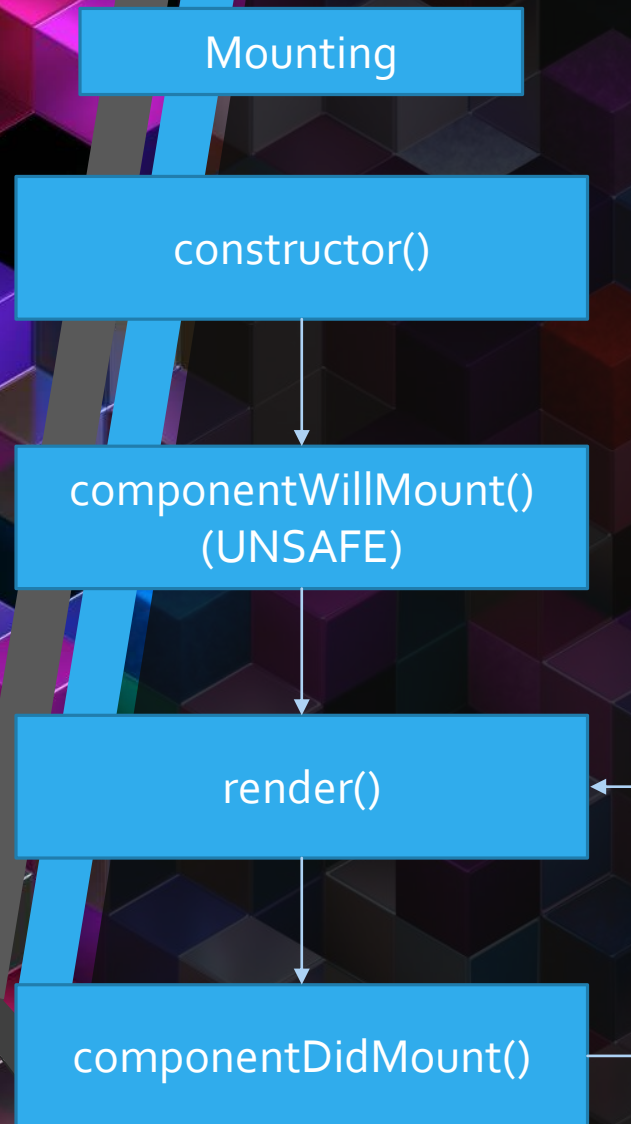
- Méthodologie React
- Création de « **Components** » réutilisables
- Passage des données avec les « **Props** »
- Changement de l'interface avec le « **State** »
- Flux de Données à Deux Sens
- Exemple d'Application

Méthodologie React

- En résumé :
 - Diviser pour régner (« **Components** »)
 - Construire une version statique (« **Props** »)
 - Rendre cette version dynamique (« **State** »)
 - Maîtriser le flux de données à deux sens (« two way Data-binding ») pour rendre le tout fonctionnel.

Méthodologie React

Types de rendues



- Une application React à un cycle de vie.
- Il est possible de lancer des opérations à différents moments de celui-ci.
- Bien comprendre chaque emplacement peut permettre de refactoriser du code et d'augmenter la performance.

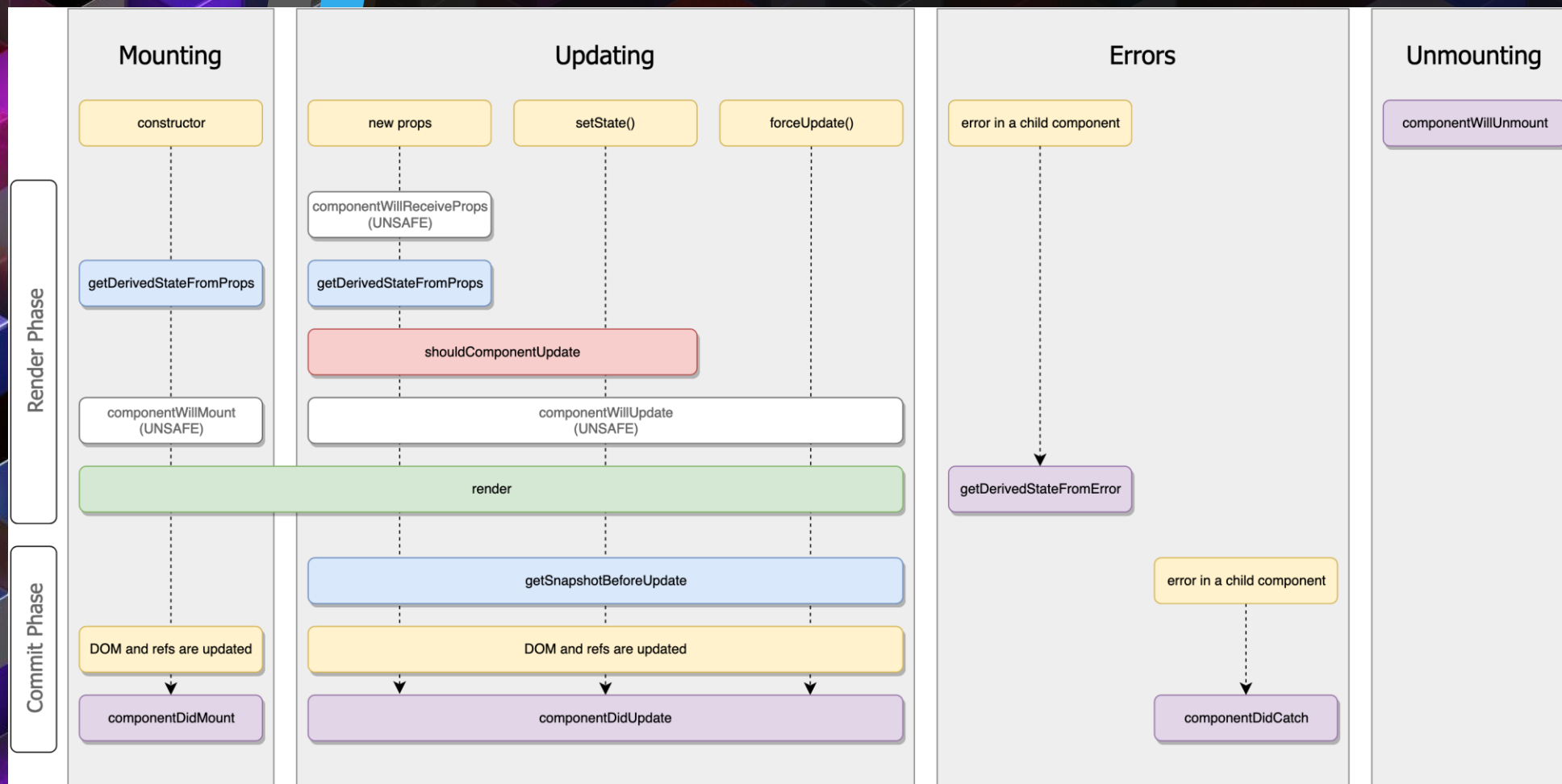
Méthodologie React

Types de rendues (Avancé)

- Voici les trois types de cycle de vie :
 - « Mounting » : Création et insertion du « **Component** » dans le DOM.
 - « Updating » : Lorsqu'une MAJ des « **Props** » et du « **State** » survient.
 - « Unmounting » : Lorsqu'un élément est enlevé du DOM.

Méthodologie React

Types de rendues (Avancé)



Création de « **Components** » réutilisables

```
class Course extends React.Component {  
  render() {  
    // Du code JavaScript (ES5+) peut être utilisé dans  
    // le render().  
    const sigle = "IFT3225"  
  
    // Retourne les balises HTML avec le(s) résultat(s).  
    return (  
      <div>Cours {sigle}</div>  
    )  
  }  
}  
  
ReactDOM.render(  
  <Course />,  
  document.getElementById('course')  
)
```

- L'idée est de ne jamais réinventer la roue.
- Un « **Component** » peut être réutilisé autant de fois que nécessaire.

Passage des données avec les « Props »

```
class Course extends React.Component {  
  render() {  
    // Nul besoin de passer le props dans une constante  
    // ou variable si aucune manipulation n'est faite.  
  
    // Retourne les balises HTML avec le(s) résultat(s).  
    return (  
      <div>Cours {this.props.sigle}</div>  
    )  
  }  
}  
  
ReactDOM.render(  
  <Course />,  
  document.getElementById('course')  
)
```

- Il est possible de passer des données en attributs.
- Le « Props » aura le nom de l'attribut et sera disponible dans le « Component ».
- Il est possible de découpler les « Props » dans d'autres « Components ».

Changement de l'interface avec le « State »

```
class Course extends React.Component {  
  constructor(props) {  
    super(props);  
    // La déclaration "this.state" valable  
    // seulement dans le constructeur.  
    this.state = {  
      sigle: 'Une Valeur'  
    }  
  }  
  
  componentDidMount() {  
    this.setState({sigle: 'IFT3225'})  
  }  
  
  render() {  
    return (  
      <div>Cours {this.state.sigle}</div>  
    )  
  }  
}  
  
ReactDOM.render(  
  <Course />,  
  document.getElementById('course')  
)
```

- Le « State » est utile pour donner vie à notre application. Contrairement aux données des « Props » qui ne sont pas pour changer radicalement, le « State » peut changer lors de sa durée de vie notamment avec les entrées des utilisateurs.

*Le code montre l'utilisation d'un « State » à une seule direction.

Le « **State** », Comment l'Utiliser?

```
class Course extends React.Component {
  constructor(props) {
    super(props);
    // La déclaration "this.state" valable
    // seulement dans le constructeur.
    this.state = {
      sigle: 'Une Valeur'
    }
  }

  componentDidMount() {
    // Synchrone
    // Bon
    this.setState({sigle: 'IFT3225'})
    // Mauvais
    this.state.sigle = 'IFT3225'

    // Asynchrone
    // Bon - Permet d'attendre après "props.title".
    this.setState((state, props) => ({
      result: this.state.sigle + " - " + this.props.title
    })))
    // Mauvais - La valeur de "props.title" peut être vide.
    this.setState({
      result: this.state.sigle + " - " + this.props.title
    })
  }
  ...
}
```

- Un state ne peut être déclaré ou changer de n'importe quelle manière.
- Cela assure une manipulation sécuritaire des données.
- Le « **State** » devrait être dans le parent le plus haut/global possible.
- Un « **setState()** » ne peut être mis dans un « **constructor()** », logiquement puisque l'on peut tout simplement y désigner nos valeurs directement dans le « **this.state = ...** ».

Un « State » ou Non?

- Il est très important de ne pas transformer toutes les données de notre application en « State ». Il faut le faire seulement lorsque nécessaire, voici 3 questions à toujours se poser sur nos données :
 - Est-ce passé par un parent via « Props »?
 - Est-ce inchangé durant toute la durée de vie de l'application?
 - Est-ce qu'un résultat peut être calculé à partir d'un autre « State » ou via des « Props » dans le même « Component »?

Si la réponse pour toutes ses questions est NON, un « State » est adéquat.

Le « **State** », dans quel « **Component** »?

- De plus, il est important de choisir le ou les bon(s) « **Component(s)** » où le « **State** » devrait résider.
 - Identifier les « **Components** » qui offre un rendu basé sur les données du « **State** ».
 - Identifier un « **Component** » parent le plus haut hiérarchiquement (celui qui découle ses données au plus grand nombre de « **Components** » enfants, si la réponse n'est pas l'entièreté).
 - Après réflexion des points ci-dessus, si le « **State** » est sujet à être dans un grand nombre de « **Components** », un « **Component** » parent peut être créé avec seule responsabilité de tenir le « **State** ».

Flux de Données à Deux Sens

- Par défaut, React ne permet pas aux données de prendre une direction inverse. Nous devons mettre en place le nécessaire nous même.
- Cela peut sembler intimidant pour les nouveaux arrivants de React. Par contre, cette décision permet de forcer le programmeur à bien comprendre ce qu'il fait, à garder le tout sécuritaire et clair au côté du code.

Flux de Données à Deux Sens



Écrire...
Cours

The diagram shows two input fields connected by a double-headed arrow, indicating bidirectional data flow. The top field is labeled 'Écrire... Cours' and the bottom field is labeled 'IFT3225 Cours IFT3225'. A blue line runs vertically through the center of the diagram, and a grey line runs horizontally across the bottom.

IFT3225
Cours IFT3225

- Pour que le flux fonctionne, il faut :
 - Ajouter un événement (`onChange`, `onClick`, etc.).
 - Lui ajouter une méthode de gestion d'événements qui est lié dans le « `constructor()` ».
 - Créer la méthode avec la logique ainsi que le « `setState()` »

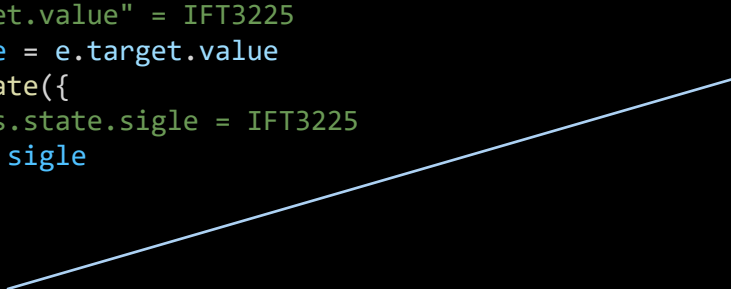
Flux de Données à Deux Sens

```
class Course extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      sigle: ''
    }

    // Méthode gestion d'événements
    this.handleChange = this.handleChange.bind(this)
  }

  handleChange(e) {
    // "e.target.value" = IFT3225
    const sigle = e.target.value
    this.setState({
      // this.state.sigle = IFT3225
      sigle: sigle
    })
  }

  render() {...}
}
```



```
render() {
  return (
    <div>
      <form>
        <input
          type="text"
          placeholder="Écrire..."
          value={this.state.sigle}
          // Événement pris en compte.
          // Ex: on écrit "IFT3225"
          onChange={this.handleChange}
        />
      </form>
      { /* Lorsqu'on écrit IFT3225, le tout va être à jour. */ }
      <div>Cours {this.state.sigle}</div>
    </div>
  )
}
```

Exemple d'Application Maquette et Planification

Page web avec React - List de Cours

☐ Monter les cours ouverts aux inscriptions seulement.

Cours : IFT3225 - Ouvert : Oui

Cours : IFT1015 - Ouvert : Oui

Cours : ECN1015 - Ouvert : Non

Cours : HIS1000 - Ouvert : Non

- Il est important d'avoir une bonne idée des séparations que l'on veut faire.
- Avec les séparations, on peut décider des relations parent/enfant des « **Components** ».
- En partant de zéro, on peut toujours faire un dessin de rendu.

Exemple d'Application

Diviser pour régner (« Components »)

```
class Course extends React.Component { ...}  
  
class CourseList extends React.Component { ... }  
  
class SearchBar extends React.Component { ... }  
  
class FilterableCourseList extends React.Component { ... }  
  
const courses = [ ... ]  
  
ReactDOM.render(  
  <FilterableCourseList courses={courses} />,  
  document.getElementById('courses')  
)
```

- Représentation des « Components »
 - Rouge : Élément normal extérieur (<p>).
 - Vert : Élément global de l'exemple. (<FilterableCourseList>)
 - Bleu : Obtiens les données de recherche. (<SearchBar>)
 - Mauve : Liste contenant la logique de filtrage. (<CourseList>)
 - Orange : Représente un élément de cours. (<Course>)

Exemple d'Application

Version Statique (« Props »)

- Prioriser le découlement correct des données dans tout les « Components » parent et enfant.
- On s'assurerait que les données de cours soient bien affichés et transmises dans les « Components » enfants de `<FilterableCourseList>`

```
class Course extends React.Component {
  render() {
    return (
      <li class="list-group-item">
        Cours : {this.props.sigle} -
        Ouvert : {(this.props.opened) ? "Oui" : "Non"}
      </li>
    )
  }
}

class CourseList extends React.Component {
  render() {
    const filterText = this.props.filterText;
    const openedOnly = this.props.openedOnly;
    const rows = []

    this.props.courses.forEach(course => {
      if (openedOnly && !course.opened)
        return
      if (!course.sigle.includes(filterText))
        return
      rows.push(
        <Course
          sigle={course.sigle}
          opened={course.opened}
        />
      )
    })

    return (
      <ul class="list-group">
        {rows}
      </ul>
    )
  }
}
```

FilterableCourseList

CourseList

Course



Exemple d'Application

Version dynamique (« **State** »)

Flux à sens unique

Données

- Liste des cours.
- Texte de recherche.
- Valeur de la case à cocher
- La liste de cours filtré.

```
class FilterableCourseList extends React.Component {  
  constructor(props) {  
    super(props)  
    this.state = {  
      filterText: '',  
      openedOnly: false  
    }  
  }  
}
```

- On commence à réfléchir sur quelle donnée serait sujet à convertir en « **State** ».
- En se posant les questions d'évaluations, nous pouvons conclure que le texte de recherche et la case à cocher réside dans le « **State** » et que celui-ci est dans le « **Component** » parent le plus haut `<FilterableCourseList>`.

Exemple d'Application

Version dynamique (« **State** »)

Flux à sens double

Page web avec React - List de Cours Ne fonctionne pas?

☐ Monter les cours ouverts aux inscriptions seulement.

Cours : IFT3225 - Ouvert : Oui
Cours : IFT1015 - Ouvert : Oui
Cours : ECN1015 - Ouvert : Non
Cours : HIS1000 - Ouvert : Non

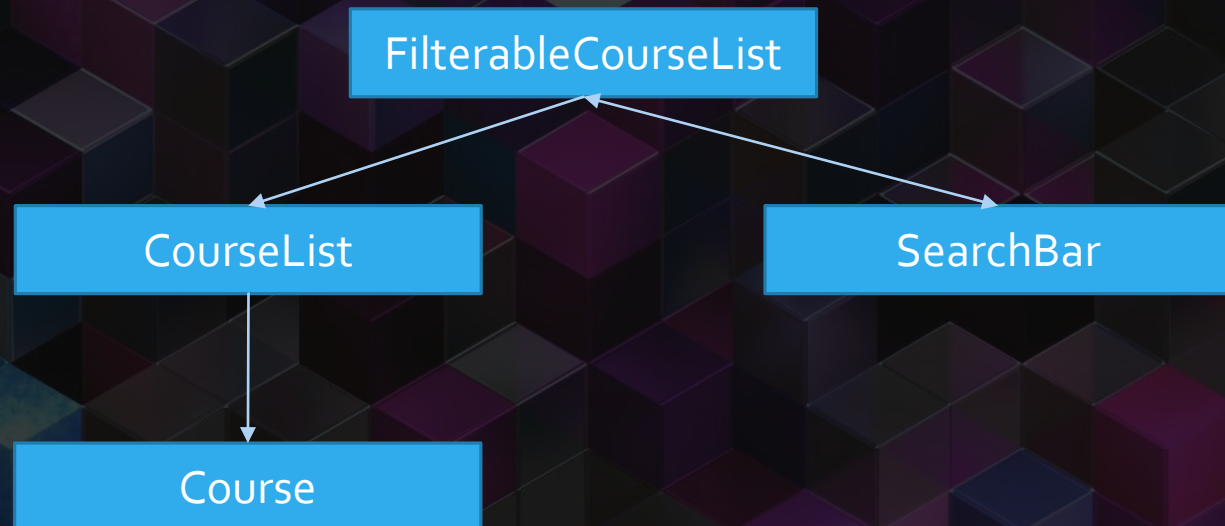
- Lorsque nous utilisons l'application, la barre de recherche et la case à cocher ne fonctionnent pas...
- En effet, il faut ajouter le flux à sens double pour que notre interface reflète les changements.

Exemple d'Application

Version dynamique (« **State** »)

Flux à sens double (suite)

- Le « **State** » réside dans `<FilterableCourseList>` pour pouvoir passer la valeur à `<CourseList>` ainsi que faire les échanges nécessaires avec `<SearchBar>`.



Exemple d'Application

Version dynamique (« State »)

Flux à sens double (suite)

```
class FilterableCourseList extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      filterText: '',
      openedOnly: false
    }

    this.handleFilterTextChange =
      this.handleFilterTextChange.bind(this)

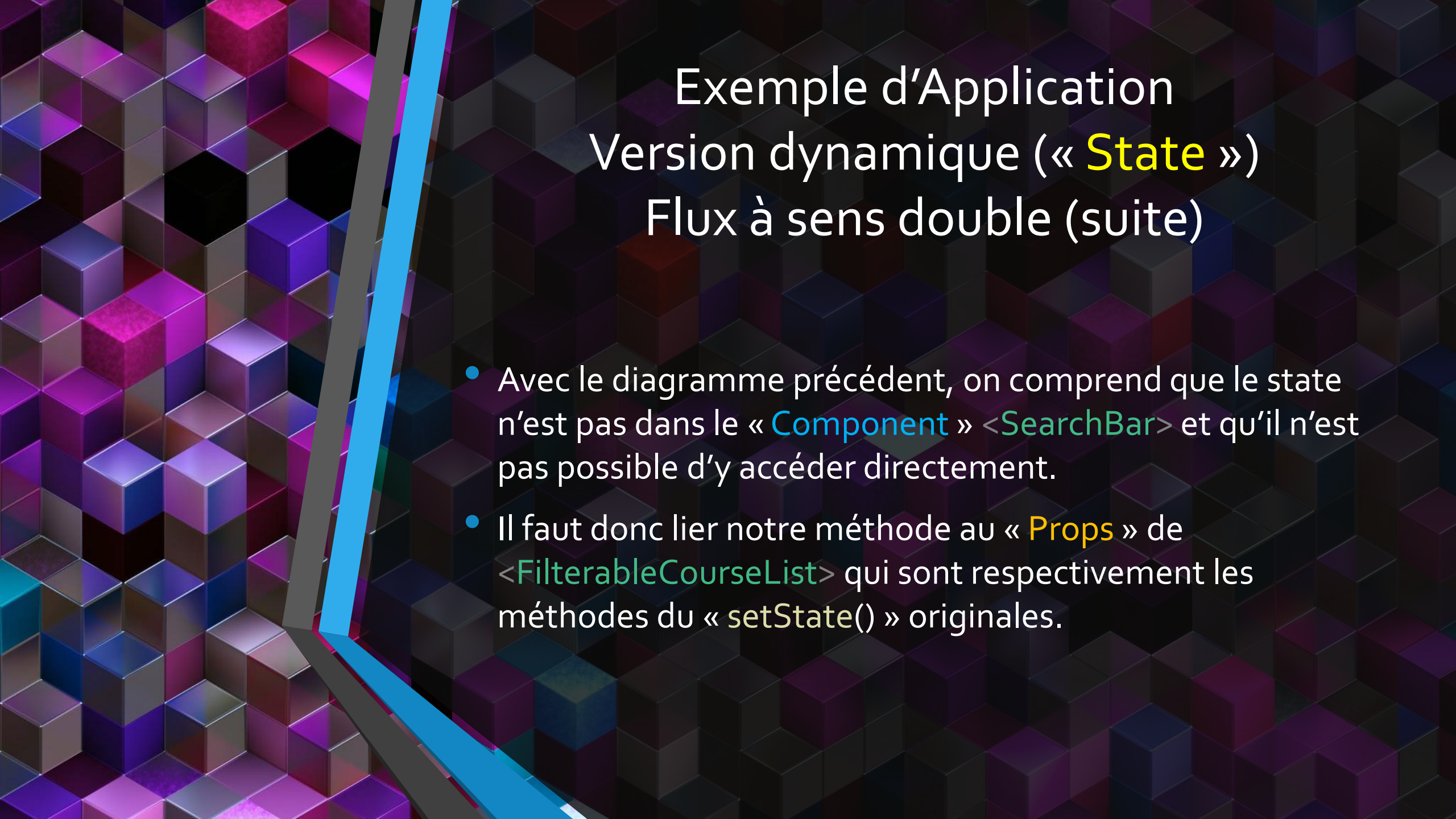
    this.handleOpenedChange =
      this.handleOpenedChange.bind(this)
  }

  handleFilterTextChange(filterText) {
    this.setState({
      filterText: filterText
    })
  }

  handleOpenedChange(openedOnly) {
    this.setState({
      openedOnly: openedOnly
    })
  }

  render() { ... }
}
```

```
render() {
  return (
    <div>
      <SearchBar
        filterText={this.state.filterText}
        openedOnly={this.state.openedOnly}
        // On passe nos méthodes de changement de "state"
        // dans note "component" enfant.
        onFilterTextChange={this.handleFilterTextChange}
        onOpenedChange={this.handleOpenedChange}
      />
      <CourseList
        courses={this.props.courses}
        filterText={this.state.filterText}
        openedOnly={this.state.openedOnly}
      />
    </div>
  )
}
```

Exemple d'Application

Version dynamique (« **State** »)

Flux à sens double (suite)

- Avec le diagramme précédent, on comprend que le state n'est pas dans le « **Component** » `<SearchBar>` et qu'il n'est pas possible d'y accéder directement.
- Il faut donc lier notre méthode au « **Props** » de `<FilterableCourseList>` qui sont respectivement les méthodes du « `setState()` » originales.

Exemple d'Application

Version dynamique (« State »)

Flux à sens double (suite)

```
class SearchBar extends React.Component {
  constructor(props) {
    super(props)
    this.handleFilterTextChange =
      this.handleFilterTextChange.bind(this)

    this.handleOpenedChange =
      this.handleOpenedChange.bind(this)
  }

  // Le "state" n'est pas local, il faut le changer
  // à partir du "props"
  // passé par le parent.
  handleFilterTextChange(e) {
    this.props.onFilterTextChange(e.target.value)
  }

  handleOpenedChange(e) {
    this.props.onOpenedChange(e.target.checked)
  }

  render() { ... }
}
```

```
render() {
  return (
    <form>
      <input
        type="text"
        placeholder="Rechercher"
        value={this.props.filterText}
        onChange={this.handleFilterTextChange}
      />
      <p>
        <input
          type="checkbox"
          checked={this.props.openedOnly}
          onChange={this.handleOpenedChange}
        />
        {' '}
        Monter les cours ouverts aux inscriptions seulement.
      </p>
    </form>
  )
}
```


Exemple d'Application Résultat

- Voilà! L'application est fonctionnelle et prête à être déployée.
- Le « **State** » de l'application peut désormais faire changer l'interface en fonction des manipulations faites par l'utilisateur.

Page web avec React - List de Cours

☐ Monter les cours ouverts aux inscriptions seulement.

Cours : IFT3225 - Ouvert : Oui

Cours : IFT1015 - Ouvert : Oui

Cours : ECN1015 - Ouvert : Non

Cours : HIS1000 - Ouvert : Non

Page web avec React - List de Cours

☒ Monter les cours ouverts aux inscriptions seulement.

Cours : IFT3225 - Ouvert : Oui

Cours : IFT1015 - Ouvert : Oui

Page web avec React - List de Cours

☒ Monter les cours ouverts aux inscriptions seulement.

Cours : IFT3225 - Ouvert : Oui

Bibliographie

- <https://reactjs.org>