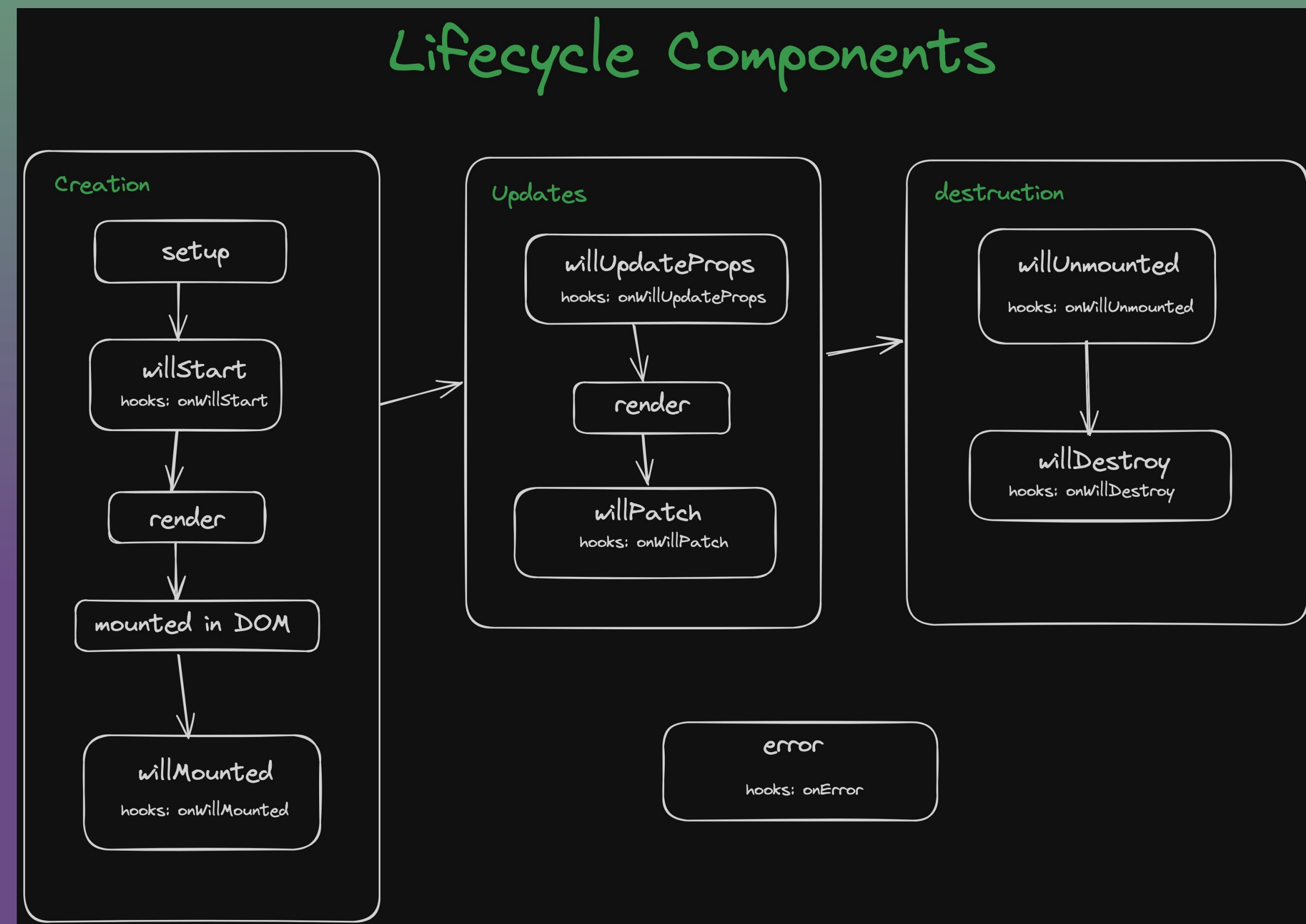



# Odoo OWL Lifecycle Components using hooks



# setup()

hook: None

Inicializa el componente configurando el estado y las propiedades. Este método se llama una vez, antes de que se realice cualquier renderizado o actualización. En setup, puedes definir el estado inicial, vincular datos reactivos, o preparar cualquier lógica que necesite estar lista antes de que el componente comience a interactuar con el usuario.




```
1  import { Component, useState } from '@odoo/owl';
2
3  class MyComponent extends Component {
4      setup() {
5          this.state = useState({ count: 0 });
6      }
7  }
```

# willStart

*hook: onWillStart(async callback)*

Se utiliza para realizar tareas asíncronas antes del primer renderizado. Es ideal para inicializar datos que se necesitan para el primer renderizado del componente.



```
1  import { Component, onWillStart } from '@odoo/owl';
2
3  class MyComponent extends Component {
4      setup() {
5          onWillStart(async () => {
6              this.data = await fetchData();
7          });
8      }
9  }
```



# willRender

## hook: onWillRender(callback)

Se llama justo antes de que el componente sea renderizado. Puedes utilizar este método para preparar cualquier dato o lógica que debe estar lista justo antes de que el componente se dibuje.

```
1  import { Component, onWillRender } from '@odoo/owl';
2
3  class MyComponent extends Component {
4      setup() {
5          onWillRender(() => {
6              console.log("Component will render");
7          });
8      }
9  }
```

# rendered

## hook: `onRendered(callback)`

Se invoca justo después de que el componente ha sido renderizado. Puedes utilizar este método para realizar tareas que dependen de la existencia del DOM del componente, como la inicialización de librerías que interactúan con el DOM.



```
1  import { Component, onRendered } from '@odoo/owl';
2
3  class MyComponent extends Component {
4      setup() {
5          onRendered(() => {
6              console.log("Component has been rendered");
7          });
8      }
9  }
```

# mounted

## hook: *onWillMounted(callback)*

Se llama después de que el componente ha sido renderizado y agregado al DOM. Es útil para realizar manipulaciones del DOM o inicializar integraciones que requieran que el componente esté completamente en el DOM.

```
1  import { Component, onWillMounted } from '@odoo/owl';
2
3  class MyComponent extends Component {
4      setup() {
5          onWillMounted(() => {
6              this.el
7                  .querySelector('button')
8                  .addEventListener('click', this.increment.bind(this));
9          });
10     }
11 }
```



# willUpdateProps

hook: `onWillUpdateProps(callback)`

Se invoca antes de que se actualicen las propiedades del componente. Aquí puedes implementar lógica que reaccione a cambios en las propiedades antes de que el componente se vuelva a renderizar.

```
1 import { Component, onWillUpdateProps } from '@odoo/owl';
2
3 class MyComponent extends Component {
4     setup() {
5         onWillUpdateProps(nextProps => {
6             return this.loadData({id: nextProps.id});
7         });
8     }
9 }
```

# willPatch

*hook: onWillPatch(callback)*

Se llama justo antes de que el DOM sea parcheado con los nuevos datos. Puedes usar este método para preparar cualquier cambio en el DOM que necesite ser manejado justo antes de que OWL aplique las diferencias al DOM.



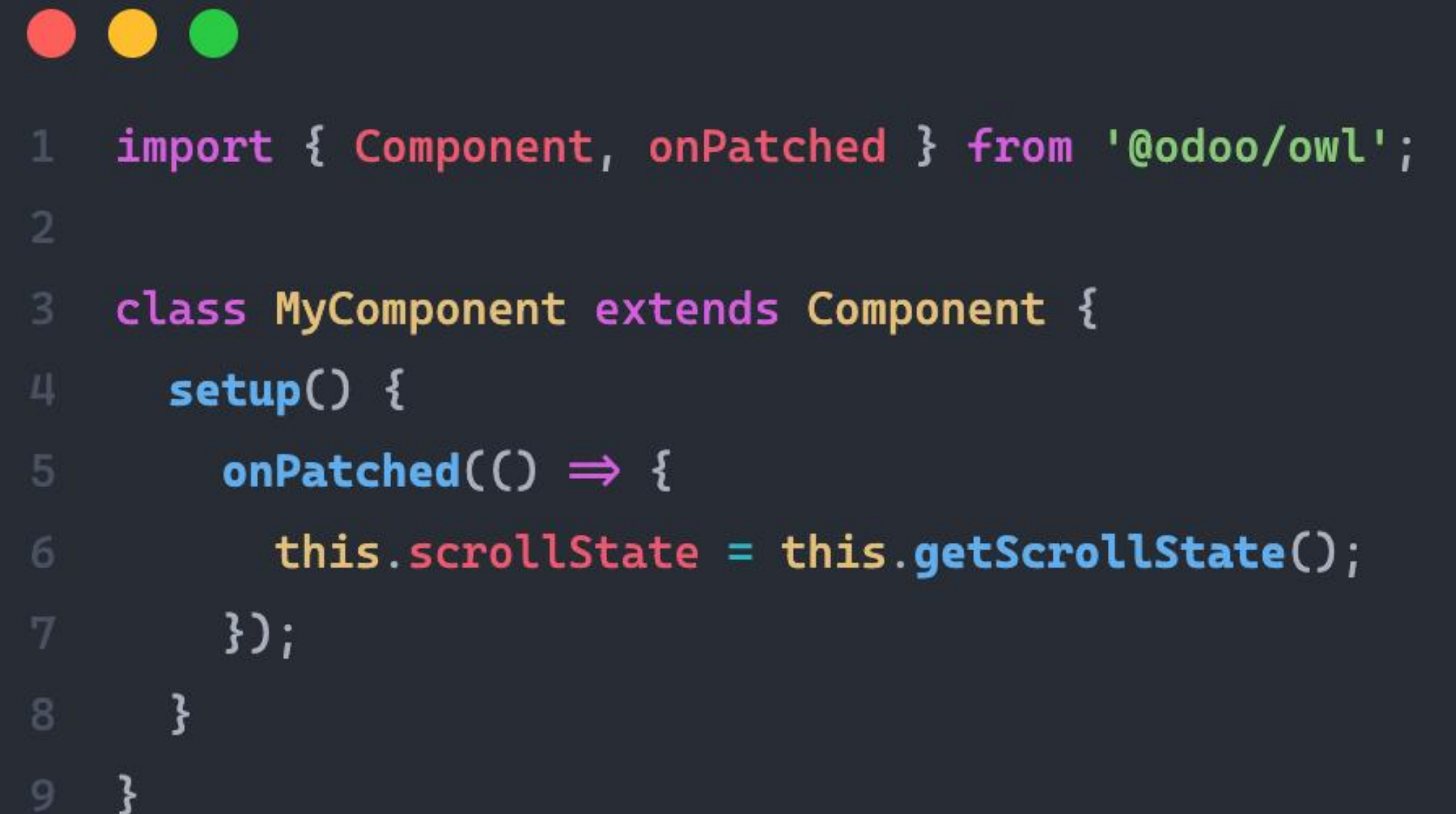
```
1  import { Component, onWillPatch } from '@odoo/owl';
2
3  class MyComponent extends Component {
4      setup() {
5          onWillPatch(() => {
6              this.scrollState = this.getScrollState();
7          });
8      }
9  }
```



# patched

## hook: onPatched(callback)

Se invoca justo después de que el DOM ha sido parcheado. Es útil para realizar ajustes finales o reacciones a la actualización del DOM.



```
1  import { Component, onPatched } from '@odoo/owl';
2
3  class MyComponent extends Component {
4      setup() {
5          onPatched(() => {
6              this.scrollState = this.getScrollState();
7          });
8      }
9  }
```

# willUnmount

*hook: onWillUnmount(callback)*

Se utiliza justo antes de que el componente sea eliminado del DOM. Es ideal para limpiar recursos o escuchadores de eventos que fueron configurados durante el ciclo de vida del componente.

```
1  import { Component, onWillUnmount } from '@odoo/owl';
2
3  class MyComponent extends Component {
4    setup() {
5      onWillUnmount(() => {
6        this.el
7          .querySelector('button')
8          .removeEventListener('click', this.increment.bind(this));
9      });
10   }
11 }
```

# willDestroy

*hook: onWillDestroy(callback)*

Se llama justo antes de que el componente sea destruido. Aquí puedes limpiar cualquier recurso que el componente haya creado o administrado, como conexiones a bases de datos, suscripciones, etc.




```
1  import { Component, onWillDestroy } from '@odoo/owl';
2
3  class MyComponent extends Component {
4      setup() {
5          onWillDestroy(() => {
6              console.log("Component will be destroyed");
7          });
8      }
9  }
```



# error

## hook: onError(callback)

Maneja errores que se producen durante el ciclo de vida del componente. Puedes usar este método para capturar y manejar errores de manera global en el componente.



```
1  import { Component, onError } from '@odoo/owl';
2
3  class MyComponent extends owl.Component {
4      onError(() => {
5          console.error("An error occurred:", error);
6      });
7  }
```