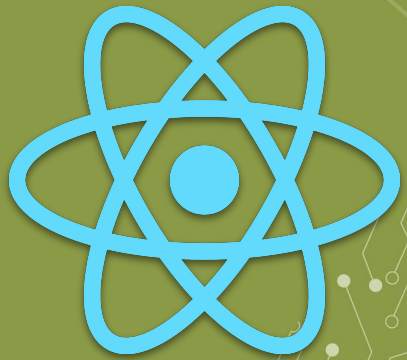
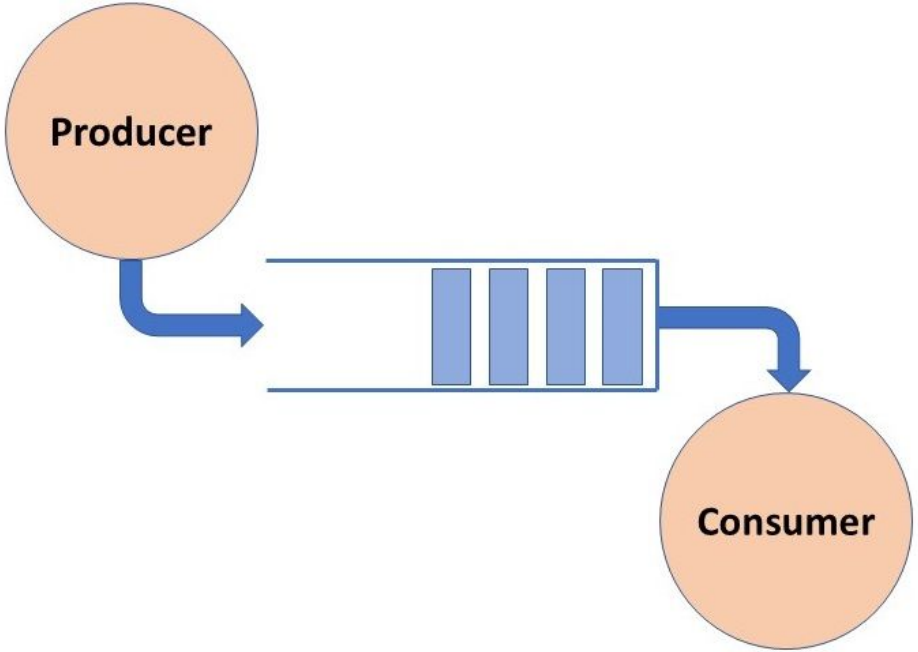
Gobierno del CHACO | Ministerio de la Producción y el Desarrollo Económico Sostenible

INFORMATORIO

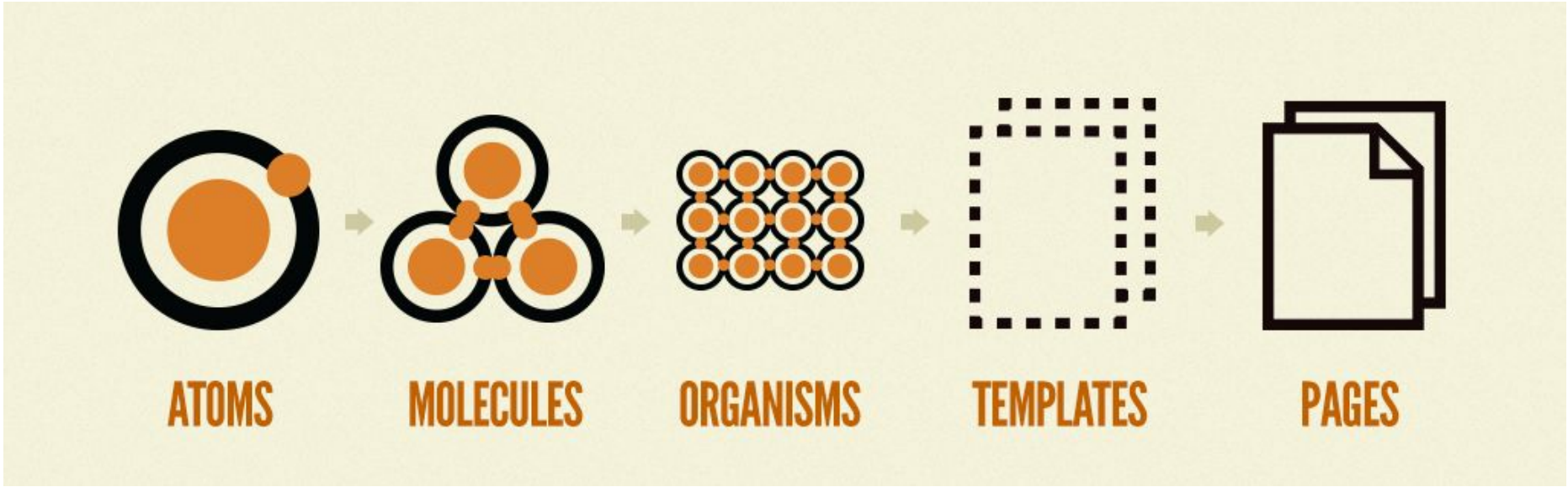
Patrones de diseño



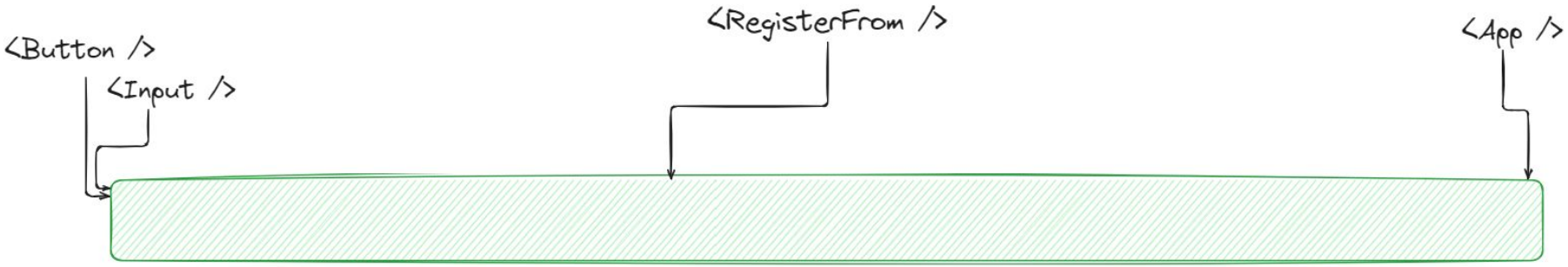
Producers vs Consumers



Atomic Design



Espectro de componentes o División de responsabilidades



Ejercicio

Ejercicio

Prop Delegation

```tsx
type TextInputProps = ComponentProps<'input'> & {
  label: string;
};
function TextInput({ label, ...delegated }: TextInputProps) {
  return (
    <div>
      <label htmlFor={delegated.id}>{label}</label>
      <input {...delegated} />
    </div>
  );
}
```

Conflictos

1. Si queremos permitir que el consumer sobreescriba los atributos, colocamos {...delegated} al final
2. Si no queremos permitir que el consumer sobreescriba los atributos, colocamos {...delegated} al principio
3. Si queremos utilizar ambos valores, necesitamos manejarlo nosotros mismos sin usar {...delegated}

Forwarding Ref

```
const Slider = forwardRef<HTMLInputElement, SliderProps>(function (
  { label, ...delegated },
  ref
) {
  const id = useId();
  return (
    <div>
      <label htmlFor={id}>{label}</label>
      <input {...delegated} ref={ref} type="range" id={id} />
    </div>
  );
});
```

Polimorfismo

```tsx
type ListProps = {
  children: ReactNode;
  as: 'ul' | 'ol';
};
const LIST_TAGS = {
  ul: 'ul',
  ol: 'ol',
};
function List({ children, as, ...delegated }: ListProps) {
  const Tag = LIST_TAGS[as];

  return <Tag {...delegated}>{children}</Tag>;
}
```

Compound Components

```
<Dropdown>
  <Dropdown.Toggle>
    Actions
  </Dropdown.Toggle>

  <Dropdown.Menu>
    <Dropdown.Item href="/change-email">
      Change Email
    </Dropdown.Item>
    <Dropdown.Item href="/reset-pwd">
    Reset Password
    </Dropdown.Item>
    <Dropdown.Item href="/delete">
    Delete account
    </Dropdown.Item>
  </Dropdown.Menu>
</Dropdown>
```

```
function Dropdown({ children }) {
  return <div>{children}</div>;
}

function DropdownToggle({ children }) {
  return <button>{children}</button>;
}
function DropdownMenu({ children }) {
  return <nav>{children}</nav>;
}
function DropdownItem({ href, children }) {
  return <a href={href}>{children}</a>;
}

Dropdown.Toggle = DropdownToggle;
Dropdown.Menu = DropdownMenu;
Dropdown.Item = DropdownItem;

export default Dropdown;
```

Inversión de Control

```
// Version 1: Sistema cerrado
<Dropdown
  label="Actions"
  options={[
    { href: '/change-email', label: "Change Email" },
    { href: '/reset-pwd', label: "Reset Password" },
    { href: '/delete', label: "Delete Account" },
  ]}
/>
```

```
// Version 2: Inversion de Control
<Dropdown>
  <DropdownToggle>
    Actions
  </DropdownToggle>

  <DropdownMenu>
    <DropdownItem href="/change-email">
      Change Email
    </DropdownItem>
    <DropdownItem href="/reset-pwd">
    Reset Password
    </DropdownItem>
    <DropdownItem href="/delete">
    Delete account
    </DropdownItem>
  </DropdownMenu>
</Dropdown>
```

Ventajas/Desventajas

1. Version 1 es mas facil de usar, pero más rígido. Lo cual no nos permite mucha customización del componente.
2. Version 2 lleva un poco mas de trabajo, pero es mucho más flexible. Podemos combinar las piezas de muchas maneras, o incluso sustituirlas por nuestros propios componentes.

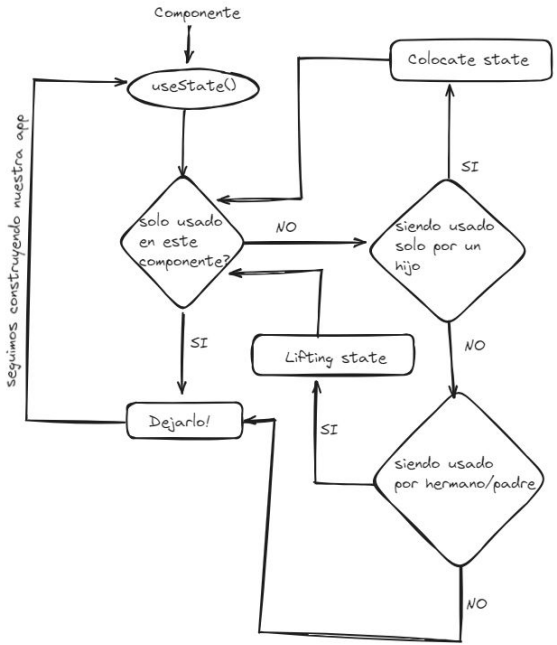
Render props

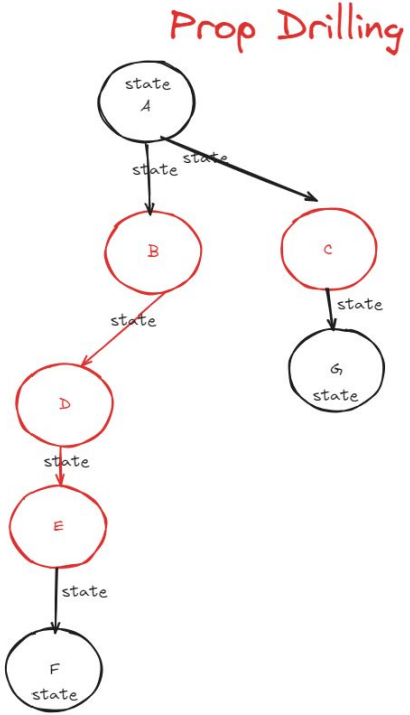
```tsx
function Input({ render }: { render: (value: string) => JSX.Element }) {
  const [value, setValue] = useState('');

  return (
    <>
      <input
        type="number"
        value={value}
        onChange={(event) => setValue(event.target.value)}
      />
      {render(value)}
    </>
  );
}
```

```tsx
export default function App() {
  return (
    <div>
      <h1>Conversor</h1>
      <Input
        render={(value) => {
          return (
            <div>
              <Celcius value={Number(value)} />
              <Fahrenheit value={Number(value)} />
            </div>
          );
        }}
      />
    </div>
  );
}
```

Context - Que soluciona?





Context - Sintaxis

1. Providing
2. Consuming

Context - Providing

Usamos un “Provider” para hacer un valor en particular disponible en todo el “context”. Esto lo hacemos envolviendo nuestra aplicación en un componente Provider, el cual nos brinda react cuando creamos un context.

```
// Creamos un nuevo context
export const FavouriteColorContext = createContext('');

export default function App() {
  const [favouriteColor, setFavouriteColor] = useState('#EBDEFB')

  // Envolvemos nuestra aplicación con el Provider
  return (
    <FavouriteColorContext.Provider value={favouriteColor}>
      <Home />
    </FavouriteColorContext.Provider>
  );
}
```

Context - Consuming

Para acceder al context, react nos brinda un hook para hacerlo facilmente. useContext

```
function Home() {
  const favouriteColor = useContext(FavouriteColorContext);
  return (
    <div>
      <h1>hola</h1>
      <p>Tu color favorito es el {favouriteColor}, gran color!</p>
    </div>
  );
}
```