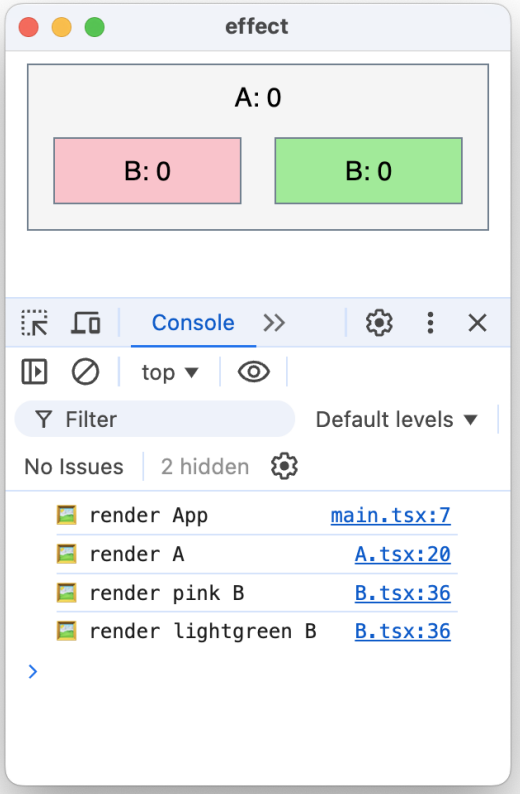
Effects

- Side Effects
- Controlled and Uncontrolled Input
- Canvas Component

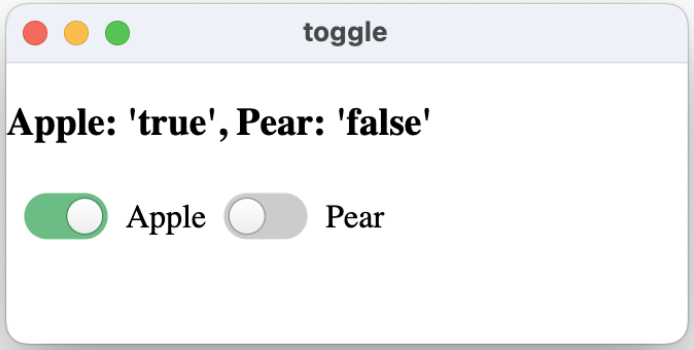
effect

- review of useState
- When component functions are called for rendering
- **Demos**
  - Examine “renders” of A
  - Add two B children and examine renders



toggle

- CSS creates look
  - imported style, hides real checkbox
- emulates input checkbox
  - `checked` boolean prop, `onChange` Event prop
- has optional `label` string prop for label text
- **Demos**
  - Two ways to save state: useState and signal
  - Why do both Toggles render when only one changes?
  - When *all* state local, only that component renders

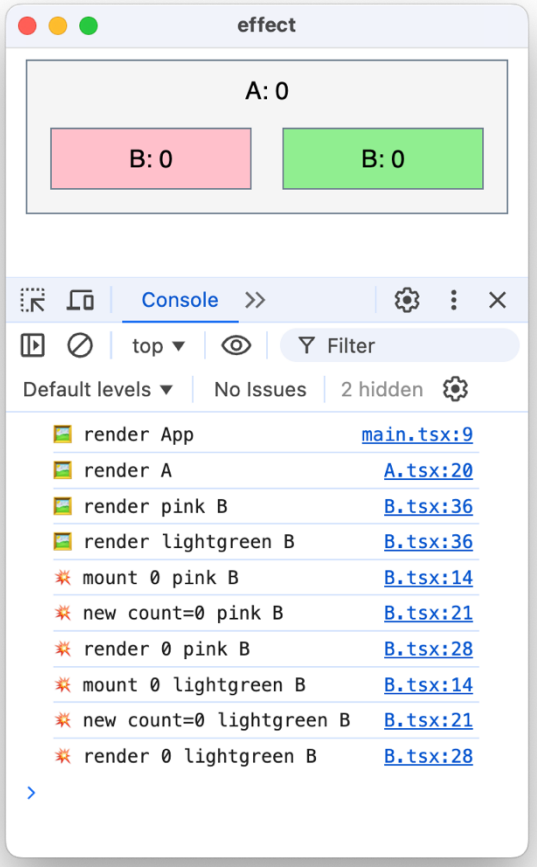


Side Effects

- Side effects are changes outside the Vdom render
  - Code is run as a result of Vdom change
- Examples of side effects
  - Logging
  - DOM manipulation (e.g. classlist.AddClass, global listeners)
  - Fetching data
  - Drawing in canvas graphics context
- Side effect hooks
  - useEffect (and useLayoutEffect)
  - useMemo
  - useCallback
  - useRef

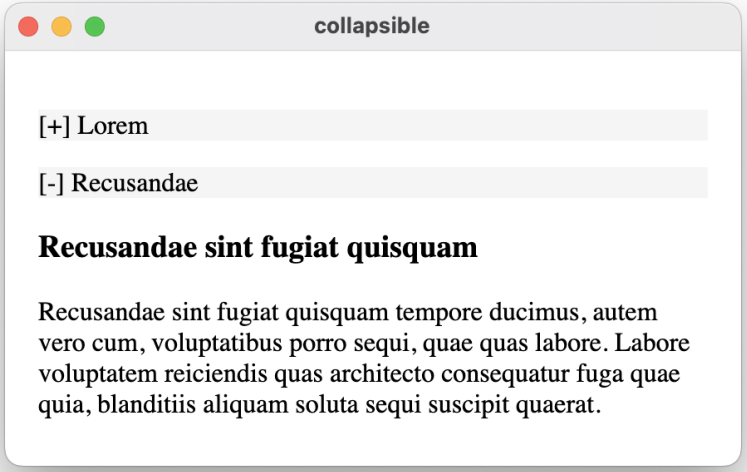
effect

- how useEffect works
- **Demos 1**
  - Just component A
  - Add a useEffect to A
- **Demos 2**
  - Add two B children
  - Examine renders when A or B changes
  - useEffect examples in B
  - *dependency array arg:* nothing, [], [count]
  - Returning a *cleanup function*



collapsible

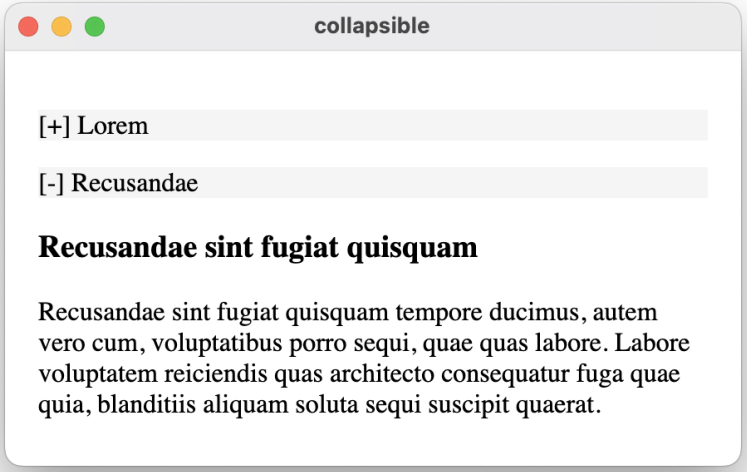
- useState for local component state
- children prop
- conditional rendering
- simple inline styling



inspired by: https://blog.openreplay.com/creating-a-collapsible-component-for-react/

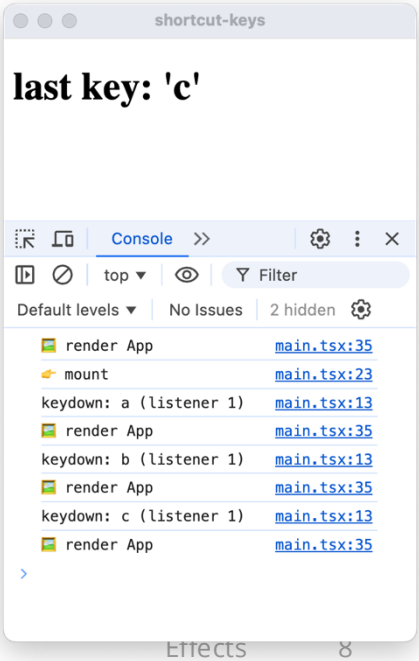
collapsible

- convert to a “local” signal
- **Demos**
  - ❌ Declaring signal as *local variable* doesn’t work
  - ❌ Declaring signal as *module variable* doesn’t work
    (try setting isOpen.value to startOpen)
    (try wrapping that in a useEffect)
  - ✅ *useMemo hook* to create local signal



shortcut-keys

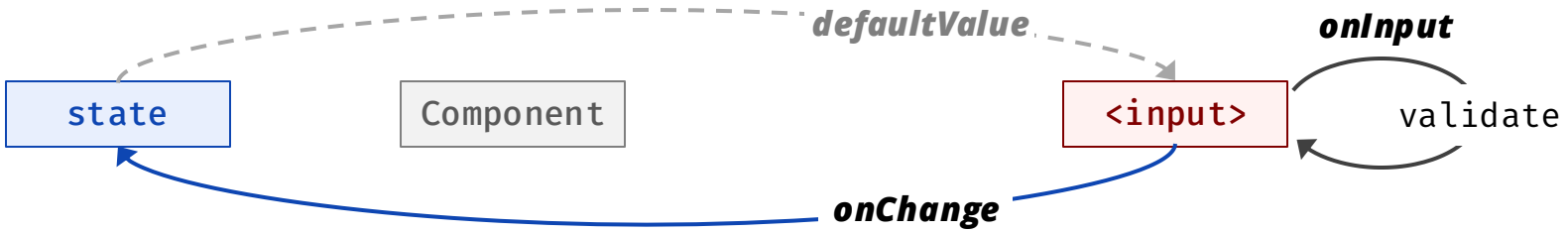
- Best practice to add global shortcut keys
- useCallback to ”memorize” the callback function
- useEffect to count number of times handler changes
- useEffect to add listener and remove listener
- **Demos**
  - Declare handler without useCallback
  - Set listener in function body (without useEffect)



Component Data Flow

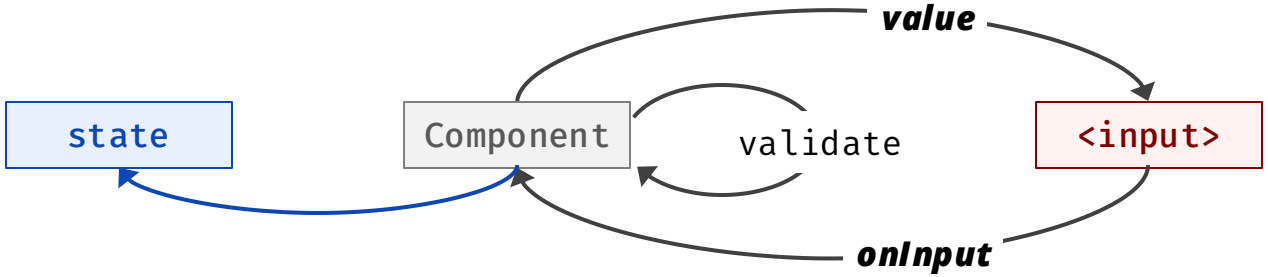
- How HTML Form Element values are associated with application state
- **Uncontrolled**
  - HTML Form Elements manage their own data values

```
<input defaultValue={someValue} onChange={myEventHandler} />;
```



- **Controlled**
  - Application state manages HTML Form Element data values

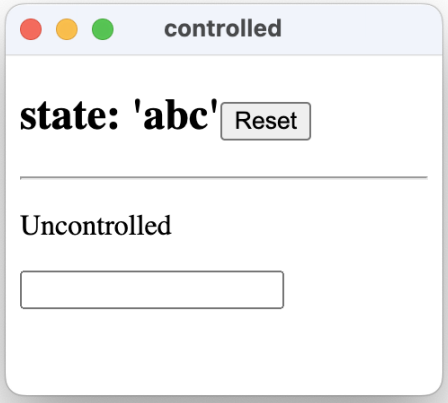
```
<input value={someValue} onInput={myEventHandler} />;
```



https://preactjs.com/guide/v10/forms

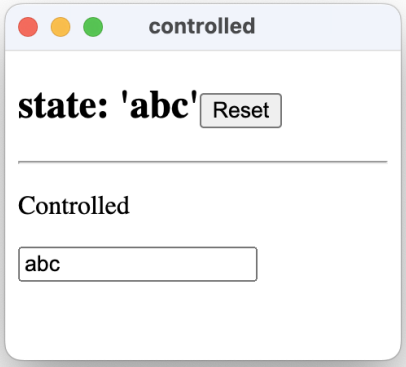
controlled < Uncontrolled />

- App state is a string signal (in “state.ts”)
- Uncontrolled component
  - <input type=”text” …
  - OnChange event sets state on focus loss
- **Demos**
  - Use defaultValue to initialize to state
  - Use OnInput event to set state on each keystroke
  - Add HTML DOM validation for `[abc]*`



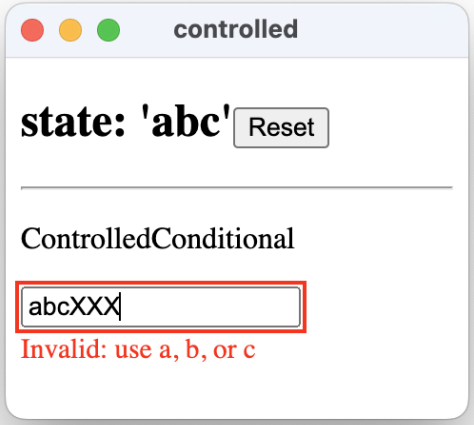
controlled < Controlled />

- Controlled component
  - <input type=”text” …
  - <input> value is set with state
  - No need for defaultValue to initialize to state
  - OnInput calls handler which does validation and updates state
- **Demos**
  - suppress input instead of validate



Controlled < ControlledConditional />

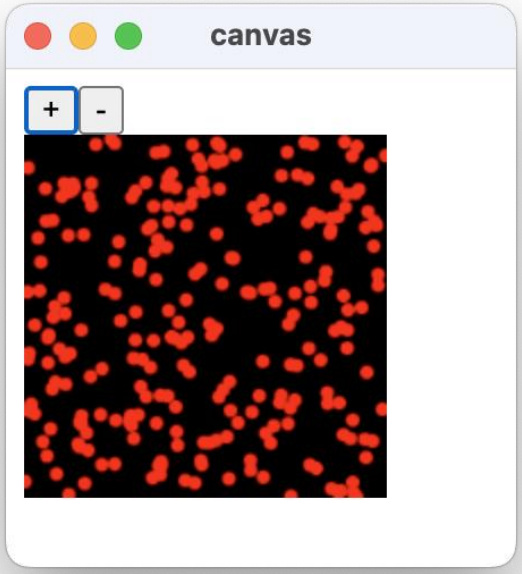
- More advanced method of handling text input with validation
  - Input text changes state only when valid, when not focused for editing, the input text matches state
- Key ideas
  - local state to display input value
  - useEffect to update local state when app state changes (e.g. "Reset")
  - only updates application state when valid input
  - leave text input value with valid value when exiting
  - only render error message <p> when invalid



Generally, you should use Controlled Components

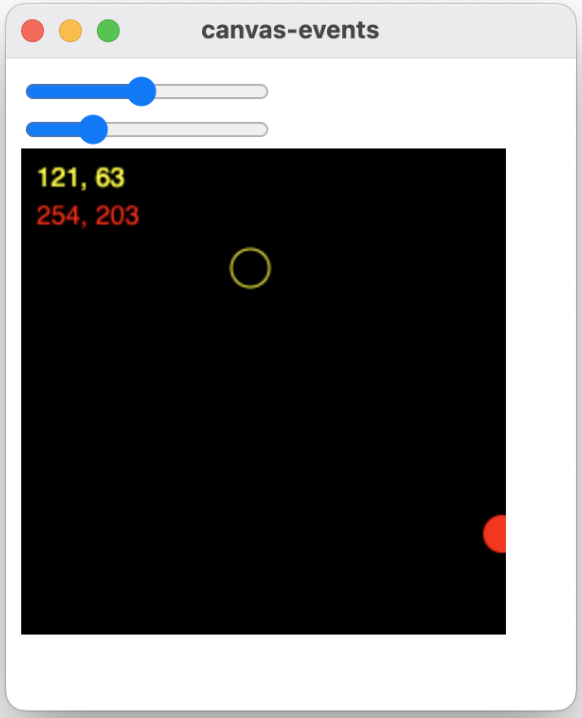
canvas

- In Canvas component
  - useRef
  - useLayoutEffect to draw
  - Draw function should look familiar
- **Demos**
  - Switch to useEffect and adjust size



canvas-events

- local state for point in App and movePoint in Canvas
  - remember to set new point object, don't just change object property
  - changing property point won't trigger a component update
- draw using useLayoutEffect when point local state changes
- handlers reference canvas
  - when canvas destroyed, the handlers are also destroyed



ResizeObserver

- HTML DOM feature
- Like window-resize for elements

https://barvysta.medium.com/resizeobserver-a-new-powerful-tool-for-responsive-web-f9a53ed71952

canvas-resize

- Try commenting out ResizeObserver update
  - canvas DOM size increases, but not graphics context (GC) size
- Try commenting out canvas width=100% and height=100%
  - see default 100 x 100 canvas
- Setup ResizeObserver in useEffect
  - so we can remove it when canvas is destroyed
- Local state for GC width and height
- Try changing useLayoutEffect for draw to useEffect