BUILD AN ACCESSIBLE REACT TOOLTIP FROM SCRATCH

Enhanced step-by-step edition

Every step includes the complete cumulative JavaScript, with the code introduced in that step clearly marked.

React 19 | *CodePen* | *Accessibility* | *Portals* | *Auto Flip* | *Auto Placement*

- Keep all 29 original steps (Step 0 through Step 28)
- Full runnable checkpoint after every step
- Yellow code markers show exactly what changed in the current step
- Concept-only steps repeat the current full code and explicitly say that no code changed

How this enhanced edition works

This edition keeps the progression and concepts from the uploaded tutorial, but changes the code presentation: after every numbered step you get the complete JavaScript as it should look at that checkpoint. Lines or blocks introduced in the current step are surrounded by marker comments and highlighted in the PDF.

Marker legend

```
// >>> ADDED IN STEP N: short explanation
const somethingNew = ...;
// <<< END STEP N ADDITION
```

Important: When a step is conceptual (for example Step 17: clamping is not auto layout), the code checkpoint is intentionally unchanged. The PDF still repeats the complete cumulative code and labels that fact.

Original progression retained

Step 0 - CodePen Setup
Step 1 - Create the Smallest Possible Tooltip
Step 2 - Show the Tooltip Only on Hover
Step 3 - Remove the Wrapper Problem
Step 4 - Preserve Existing Event Handlers
Step 5 - Mouse Is Not Enough
Step 6 - Support Escape to Dismiss
Step 7 - Add Tooltip Semantics
Step 8 - We Need Access to the Real DOM
Step 9 - Render the Tooltip in a Portal
Step 10 - Measure the Trigger
Step 11 - Add Four Placements
Step 12 - Keep the Tooltip Inside the Viewport
Step 13 - Recalculate on Scroll and Resize
Step 14 - Make the Tooltip Hoverable
Step 15 - Store a Timer in a Ref
Step 16 - Let the Tooltip Cancel the Close
Step 17 - Understand Why Clamping Is Not Auto Layout
Step 18 - Measure Available Space
Step 19 - Determine How Much Space We Need
Step 20 - Add Auto Flip
Step 21 - Add placement="auto"
Step 22 - Requested Placement vs Actual Placement
Step 23 - The Final Positioning Pipeline
Step 24 - Final Complete Component
Step 25 - What Each React Feature Is Doing
Step 26 - The Final Architecture
Step 27 - Important Concept: React vs the Browser
Step 28 - Accessibility Boundary

Step 0 - CodePen Setup

Goal

Prepare the HTML, CSS, and Babel environment before introducing Tooltip logic.

What we are learning

- CodePen panels
- Babel
- Shared CSS

Logic

1. Add to the HTML panel.
2. Enable Babel in the JavaScript panel.
3. Load the shared CSS once so later steps can focus on React.

Why this matters: The environment should be boring and stable. We want every later behavior change to come from the React code, not from setup noise.

HTML panel - complete

```
<div id="root"></div>
```

CSS panel - complete

We intentionally use the final shared CSS from the beginning. Later JavaScript steps gradually make use of more of these rules.

```
* { box-sizing: border-box; }
body {
  margin: 0;
  font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", sans-serif;
  background: #f5f6f8;
  color: #222;
}
.app { min-height: 100vh; }
.demo {
  width: min(1100px, calc(100% - 32px));
  margin: 0 auto;
  padding: 48px 0 80px;
}
.tooltip {
  position: fixed;
  z-index: 1000;
  background-color: #1f1f1f;
  color: #fff;
  padding: 6px 10px;
  border-radius: 6px;
  font-size: 0.875rem;
  line-height: 1.4;
  font-weight: 400;
  max-width: 280px;
  white-space: normal;
  overflow-wrap: break-word;
  box-shadow:
    0 2px 4px rgb(0 0 0 / 0.15),
    0 4px 12px rgb(0 0 0 / 0.2);
  pointer-events: auto;
  transition: opacity 120ms ease, transform 120ms ease;
}
.trigger {
  padding: 9px 14px;
  border: 1px solid #aaa;
  border-radius: 6px;
  background: #fff;
  color: #222;
  font: inherit;
  cursor: pointer;
}
.trigger:focus-visible {
  outline: 3px solid #79a7ff;
  outline-offset: 3px;
}
@media (prefers-reduced-motion: reduce) {
  .tooltip { transition: none; }
}
```

JavaScript panel - full checkpoint after Step 0

```
        import React, { StrictMode } from "https://esm.sh/react@19";
        import { createRoot } from "https://esm.sh/react-dom@19/client";

        // >>> ADDED IN STEP 0: Minimal React bootstrapping for CodePen
function App() {
  return (
    <main className="demo">
      <h1>React Tooltip Tutorial</h1>
      <p>CodePen setup is ready.</p>
    </main>
  );
}
        // <<< END STEP 0 ADDITION

        const root = createRoot(document.getElementById("root"));
        root.render(
          <StrictMode>
```

```
          <div className="app"><App /></div>
        </StrictMode>
      );
```

Checkpoint: CodePen renders a basic React page. There is no Tooltip component yet.

Step 1 - Create the Smallest Possible Tooltip

Goal / problem

Start with a component API before introducing interaction.

New ideas in this step

- Components
- Props
- children
- Composition

Logic

1. ToolTip receives content and children.
2. Render children as the trigger.
3. Render content as the tooltip.
4. Accept that the tooltip is always visible for now.

Why this hook / API / idea?: Beginning with the smallest component makes the public API visible before hooks and browser geometry make the implementation more complex.

Full JavaScript checkpoint after Step 1

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, { StrictMode } from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";

// >>> ADDED IN STEP 1: Create the smallest reusable Tooltip with children + content
const ToolTip = ({ children, content }) => {
  return (
    <>
      {children}
      <div className="tooltip">{content}</div>
    </>
  );
};
// <<< END STEP 1 ADDITION

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <ToolTip content="I am a tooltip">
        <button className="trigger">Hover me</button>
      </ToolTip>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The Tooltip API exists and content is always visible.

Step 2 - Show the Tooltip Only on Hover

Goal / problem

The component now needs memory: is the pointer over the trigger?

New ideas in this step

- useState
- Mouse events
- Conditional rendering

Logic

1. Store hovered=false.
2. Set it true on mouse enter.
3. Set it false on mouse leave.
4. Render the tooltip only when hovered is true.

Why this hook / API / idea?: useState is appropriate because hovered directly changes what React renders.

Full JavaScript checkpoint after Step 2

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";

        const ToolTip = ({ children, content }) => {
          // >>> ADDED IN STEP 2: Remember whether the pointer is over the trigger
const [hovered, setHovered] = useState(false);
// <<< END STEP 2 ADDITION
          return (
            <span
              onMouseEnter={() => setHovered(true)}
              onMouseLeave={() => setHovered(false)}
            >
              {children}
              {hovered && <div className="tooltip">{content}</div>}
            </span>
          );
        };

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <ToolTip content="I am a tooltip">
        <button className="trigger">Hover me</button>
      </ToolTip>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The tooltip appears only while hovered.

Step 3 - Remove the Wrapper Problem

Goal / problem

The span owns behavior while the button is the real interactive element.

New ideas in this step

- cloneElement
- isValidElement
- Fragments

Logic

1. Validate that children is a React element.
2. Clone the child.
3. Attach mouse handlers to the child itself.
4. Return trigger + tooltip in a Fragment.

Why this hook / API / idea?: cloneElement lets the behavior live on the semantic trigger instead of adding a wrapper solely for event handling.

Full JavaScript checkpoint after Step 3

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";

const ToolTip = ({
  children,
  content
}) => {
  const [hovered, setHovered] = useState(false);

  const visible = hovered;

  // >>> ADDED IN STEP 3: Require a single React element because cloneElement needs an element
  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }
  // <<< END STEP 3 ADDITION

  // >>> ADDED IN STEP 3: Clone the real child and attach tooltip behavior instead of wrapping it
  const trigger = cloneElement(children, {
    onMouseEnter: event => {
      setHovered(true);
    },
    onMouseLeave: event => {
      setHovered(false);
    }
  });
  // <<< END STEP 3 ADDITION

  return (
    <>
      {trigger}
      {visible && (
        <div className="tooltip">{content}</div>
      )}
    </>
  );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <ToolTip content="I am a tooltip">
        <button className="trigger">Hover me</button>
      </ToolTip>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: Events now live on the actual child trigger, not an extra wrapper.

Step 4 - Preserve Existing Event Handlers

Goal / problem

A reusable component must not silently destroy handlers supplied by its caller.

New ideas in this step

- Handler composition
- Optional chaining

Logic

1. Call the child handler first if it exists.

2. Run tooltip behavior afterward.

3. Repeat the pattern for other handlers introduced later.

Why this hook / API / idea?: Reusable primitives should compose behavior rather than replace application behavior.

Full JavaScript checkpoint after Step 4

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";

const ToolTip = ({
  children,
  content
}) => {
  const [hovered, setHovered] = useState(false);

  const visible = hovered;

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  // >>> ADDED IN STEP 4: Compose existing child handlers instead of overwriting them
  const trigger = cloneElement(children, {
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      setHovered(false);
    }
  });
  // <<< END STEP 4 ADDITION

  return (
    <>
      {trigger}
      {visible && (
        <div className="tooltip">{content}</div>
      )}
    </>
  );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <ToolTip content="I am a tooltip">
        <button className="trigger">Hover me</button>
      </ToolTip>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: Existing child event handlers are preserved.

Step 5 - Mouse Is Not Enough

Goal / problem

Tabbing to the button should reveal the same description keyboard users need.

New ideas in this step

- Keyboard focus
- focus/blur
- Derived state

Logic

1. Add focused state.
2. Open on focus and stop focus visibility on blur.
3. Derive visible = hovered || focused.

Why this hook / API / idea?: focused is independent interaction state; visible is derived because it can be calculated from other state.

Full JavaScript checkpoint after Step 5

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";

const ToolTip = ({
  children,
  content
}) => {
  const [hovered, setHovered] = useState(false);

  // >>> ADDED IN STEP 5: Track keyboard focus separately from pointer hover
  const [focused, setFocused] = useState(false);
  // <<< END STEP 5 ADDITION

  // >>> ADDED IN STEP 5: Derive visibility instead of duplicating it in state
  const visible = hovered || focused;
  // <<< END STEP 5 ADDITION

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  // >>> ADDED IN STEP 5: Add focus and blur handlers for keyboard users
  const trigger = cloneElement(children, {
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      setHovered(false);
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    }
  });
  // <<< END STEP 5 ADDITION

  return (
    <>
      {trigger}
      {visible && (
        <div className="tooltip">{content}</div>
      )}
    </>
  );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <ToolTip content="I am a tooltip">
        <button className="trigger">Hover me</button>
      </ToolTip>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
```

```
);
```

Checkpoint: Mouse and keyboard focus both reveal the tooltip.

Step 6 - Support Escape to Dismiss

Goal / problem

Hover/focus content should be dismissible without forcing pointer or focus movement.

New ideas in this step

- Keyboard events
- Escape
- Derived state

Logic

1. Track dismissed state.
2. Set dismissed=true on Escape.
3. Make visible depend on !dismissed.
4. Reset dismissal when a fresh hover/focus interaction begins.

Why this hook / API / idea?: A dedicated dismissed flag represents an explicit user action that differs from simply losing hover or focus.

Full JavaScript checkpoint after Step 6

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";

const ToolTip = ({
  children,
  content
}) => {
  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  // >>> ADDED IN STEP 6: Track explicit Escape dismissal
  const [dismissed, setDismissed] = useState(false);
  // <<< END STEP 6 ADDITION

  const visible = (hovered || focused) && !dismissed;

  // >>> ADDED IN STEP 6: Dismiss the tooltip with Escape
  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };
  // <<< END STEP 6 ADDITION

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  // >>> ADDED IN STEP 6: Reset dismissal on new interaction and handle Escape
  const trigger = cloneElement(children, {
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      setHovered(false);
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });
  // <<< END STEP 6 ADDITION

  return (
    <>
      {trigger}
      {visible && (
        <div className="tooltip">{content}</div>
      )}
    </>
  );
};

function App() {
  return (
```

```
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <ToolTip content="I am a tooltip">
        <button className="trigger">Hover me</button>
      </ToolTip>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: Escape can dismiss the tooltip without moving focus.

Step 7 - Add Tooltip Semantics

Goal / problem

Visual proximity is not enough for screen readers.

New ideas in this step

- useId
- aria-describedby
- role="tooltip"

Logic

1. Generate a unique ID.
2. Preserve any existing aria-describedby value.
3. Append the tooltip ID.
4. Give the tooltip matching id and role="tooltip".

Why this hook / API / idea?: useId gives each Tooltip instance a stable relationship without hard-coded or duplicated IDs.

Full JavaScript checkpoint after Step 7

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useId,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";

const ToolTip = ({
  children,
  content
}) => {
  // >>> ADDED IN STEP 7: Generate a stable ID for aria-describedby
  const id = useId();
  // <<< END STEP 7 ADDITION

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const visible = (hovered || focused) && !dismissed;

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  // >>> ADDED IN STEP 7: Preserve existing descriptions and append the tooltip ID
  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");
  // <<< END STEP 7 ADDITION

  // >>> ADDED IN STEP 7: Connect the trigger to the tooltip with aria-describedby
  const trigger = cloneElement(children, {
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      setHovered(false);
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });
  // <<< END STEP 7 ADDITION

              return (
                <>
                  {trigger}
```

```
                {visible && (
                  {/* >>> ADDED IN STEP 7: Give the floating element an ID and role="tooltip" */}
  <div
              id={id}
              role="tooltip" className="tooltip">{content}</div>
  {/* <<< END STEP 7 ADDITION */}
                )}
              </>
            );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <ToolTip content="I am a tooltip">
        <button className="trigger">Hover me</button>
      </ToolTip>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The trigger and tooltip now have an accessible semantic relationship.

Step 8 - We Need Access to the Real DOM

Goal / problem

Positioning requires actual pixels from the browser.

New ideas in this step

- useRef
- DOM nodes

Logic

1. Create triggerRef and tooltipRef.

2. Attach triggerRef to the cloned trigger.

3. Attach tooltipRef to the tooltip element.

Why this hook / API / idea?: Refs are the right place for DOM nodes because reading/changing a ref does not itself need to trigger a render.

Full JavaScript checkpoint after Step 8

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useId,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";

const ToolTip = ({
  children,
  content
}) => {
  const id = useId();

  // >>> ADDED IN STEP 8: Hold references to the real trigger and tooltip DOM nodes
  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);
  // <<< END STEP 8 ADDITION

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const visible = (hovered || focused) && !dismissed;

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  // >>> ADDED IN STEP 8: Attach triggerRef to the actual interactive element
  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      setHovered(false);
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });
  // <<< END STEP 8 ADDITION

              return (
                <>
                  {trigger}
                  {visible && (
```

```
                  {/* >>> ADDED IN STEP 8: Attach tooltipRef to the floating DOM element */}
  <div
              id={id}
              role="tooltip"
              ref={tooltipRef} className="tooltip">{content}</div>
  {/* <<< END STEP 8 ADDITION */}
                  )}
                </>
              );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <ToolTip content="I am a tooltip">
        <button className="trigger">Hover me</button>
      </ToolTip>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The component can access the real DOM nodes through refs.

Step 9 - Render the Tooltip in a Portal

Goal / problem

A tooltip inside overflow:hidden can be clipped by an ancestor.

New ideas in this step

- createPortal
- React tree vs DOM tree

Logic

1. Keep the trigger where it is.
2. Render the floating tooltip under document.body.
3. Keep state and events logically inside ToolTip.

Why this hook / API / idea?: Portals change physical DOM placement without breaking React ownership, context, or event semantics.

Full JavaScript checkpoint after Step 9

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useId,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const visible = (hovered || focused) && !dismissed;

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      setHovered(false);
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

              return (
                <>
                  {trigger}
                  {visible &&
                    // >>> ADDED IN STEP 9: Render the tooltip under document.body with a React Portal
  createPortal(
              <div
```

```
                id={id}
                ref={tooltipRef}
                role="tooltip"
                className="tooltip"
              >
                {content}
              </div>,
              document.body
            )
  // <<< END STEP 9 ADDITION
                }
              </>
            );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <ToolTip content="I am a tooltip">
        <button className="trigger">Hover me</button>
      </ToolTip>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The tooltip is physically rendered under document.body.

Step 10 - Measure the Trigger

Goal / problem

We can finally position the tooltip relative to the real browser geometry.

New ideas in this step

- getBoundingClientRect
- useLayoutEffect
- Layout measurement

Logic

1. Render the tooltip.
2. Measure trigger and tooltip rectangles.
3. Calculate a top position.
4. Store top/left in state before paint.

Why this hook / API / idea?: useLayoutEffect is designed for DOM measurement that must affect layout before the browser paints a visible frame.

Full JavaScript checkpoint after Step 10

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  // >>> ADDED IN STEP 10: Store calculated fixed-position coordinates
  const [style, setStyle] = useState({});
  // <<< END STEP 10 ADDITION

  const visible = (hovered || focused) && !dismissed;

  // >>> ADDED IN STEP 10: Measure DOM and position before browser paint
  useLayoutEffect(() => {
    if (!visible) return;
    const trigger = triggerRef.current;
    const tooltip = tooltipRef.current;
    if (!trigger || !tooltip) return;
    const triggerRect = trigger.getBoundingClientRect();
    const tooltipRect = tooltip.getBoundingClientRect();
    const gap = 8;
  // >>> ADDED IN STEP 10: Calculate the first placement: TOP
    const top = triggerRect.top - tooltipRect.height - gap;
    const left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
  // <<< END STEP 10 ADDITION
    setStyle({ left, top });
  }, [visible]);
  // <<< END STEP 10 ADDITION

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      setHovered(true);
    },
```

```
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      setHovered(false);
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      style={style}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <ToolTip content="I am a tooltip">
        <button className="trigger">Hover me</button>
      </ToolTip>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The tooltip is measured and positioned above its trigger.

Step 11 - Add Four Placements

Goal / problem

One hard-coded formula is not a reusable positioning API.

New ideas in this step

- placement prop
- Geometry
- switch

Logic

1. Add placement="top" as a default.
2. Calculate top/bottom/left/right with the same measured rectangles.
3. Center along the cross-axis.

Why this hook / API / idea?: A placement prop separates caller intent from the geometry implementation.

Full JavaScript checkpoint after Step 11

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content,
  placement = "top"
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const trigger = triggerRef.current;
    const tooltip = tooltipRef.current;
    if (!trigger || !tooltip) return;
    const triggerRect = trigger.getBoundingClientRect();
    const tooltipRect = tooltip.getBoundingClientRect();
    const gap = 8;
  // >>> ADDED IN STEP 11: Generalize the position math to four placements
    let top;
    let left;
    switch (placement) {
      case "bottom":
        top = triggerRect.bottom + gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        break;
      case "left":
        left = triggerRect.left - tooltipRect.width - gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "right":
        left = triggerRect.right + gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "top":
      default:
        top = triggerRect.top - tooltipRect.height - gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
    }
  // <<< END STEP 11 ADDITION
    setStyle({ left, top });
  }, [visible, placement]);

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
```

```
    id
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      setHovered(false);
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      style={style}
    >
      {content}
    </div>,
    document.body
  )
      }
    </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Tooltip prefers top" placement="top">
          <button className="trigger">Top</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers bottom" placement="bottom">
          <button className="trigger">Bottom</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers left" placement="left">
          <button className="trigger">Left</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers right" placement="right">
          <button className="trigger">Right</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: All four explicit placements work.

Step 12 - Keep the Tooltip Inside the Viewport

Goal / problem

A correct side can still place part of the tooltip outside the visible window.

New ideas in this step

- Collision detection
- Clamping
- Math.min/Math.max

Logic

1. Define a viewport margin.
2. Clamp left to the horizontal bounds.
3. Clamp top to the vertical bounds.

Why this hook / API / idea?: Clamping is a simple form of shift middleware: it nudges coordinates without changing the chosen side.

Full JavaScript checkpoint after Step 12

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content,
  placement = "top"
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const trigger = triggerRef.current;
    const tooltip = tooltipRef.current;
    if (!trigger || !tooltip) return;
    const triggerRect = trigger.getBoundingClientRect();
    const tooltipRect = tooltip.getBoundingClientRect();
    const gap = 8;
    const margin = 8;
    let top;
    let left;
    switch (placement) {
      case "bottom":
        top = triggerRect.bottom + gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        break;
      case "left":
        left = triggerRect.left - tooltipRect.width - gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "right":
        left = triggerRect.right + gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "top":
      default:
        top = triggerRect.top - tooltipRect.height - gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
    }
  // >>> ADDED IN STEP 12: Shift/clamp the final coordinates so the tooltip stays in the viewport
    left = Math.max(
      margin,
      Math.min(left, window.innerWidth - tooltipRect.width - margin)
    );
    top = Math.max(
      margin,
      Math.min(top, window.innerHeight - tooltipRect.height - margin)
    );
  // <<< END STEP 12 ADDITION
    setStyle({ left, top });
  }, [visible, placement]);

  const handleKeyDown = event => {
```

```
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      setHovered(false);
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      style={style}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Tooltip prefers top" placement="top">
          <button className="trigger">Top</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers bottom" placement="bottom">
          <button className="trigger">Bottom</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers left" placement="left">
          <button className="trigger">Left</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers right" placement="right">
          <button className="trigger">Right</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: Coordinates are shifted back inside the viewport.

Step 13 - Recalculate on Scroll and Resize

Goal / problem

Measured coordinates become stale as the viewport or scroll position changes.

New ideas in this step

- Event listeners
- Effect cleanup
- Scroll capture

Logic

1. Move geometry into updatePosition().
2. Run it immediately.
3. Listen for resize and captured scroll.
4. Remove the listeners in cleanup.

Why this hook / API / idea?: Any side effect that subscribes to an external system needs a matching cleanup path.

Full JavaScript checkpoint after Step 13

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content,
  placement = "top"
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      let top;
      let left;
      switch (placement) {
        case "bottom":
          top = triggerRect.bottom + gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
          break;
        case "left":
          left = triggerRect.left - tooltipRect.width - gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "right":
          left = triggerRect.right + gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "top":
        default:
          top = triggerRect.top - tooltipRect.height - gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
      }
      left = Math.max(
        margin,
        Math.min(left, window.innerWidth - tooltipRect.width - margin)
      );
      top = Math.max(
        margin,
        Math.min(top, window.innerHeight - tooltipRect.height - margin)
      );
      setStyle({ left, top });
    };
  // >>> ADDED IN STEP 13: Recalculate immediately, on resize, and on nested scroll
```

```
  updatePosition();
  window.addEventListener("resize", updatePosition);
  window.addEventListener("scroll", updatePosition, true);
  return () => {
    window.removeEventListener("resize", updatePosition);
    window.removeEventListener("scroll", updatePosition, true);
  };
  // <<< END STEP 13 ADDITION
  }, [visible, placement]);

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      setHovered(false);
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      style={style}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Tooltip prefers top" placement="top">
          <button className="trigger">Top</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers bottom" placement="bottom">
          <button className="trigger">Bottom</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers left" placement="left">
          <button className="trigger">Left</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers right" placement="right">
          <button className="trigger">Right</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: Position stays current while scrolling or resizing.

Step 14 - Make the Tooltip Hoverable

Goal / problem

Immediate close on trigger mouseleave prevents the pointer from reaching the floating tooltip.

New ideas in this step

- WCAG hover persistence
- Interaction design

Logic

1. Observe the gap between trigger and tooltip.
2. Recognize that immediate setHovered(false) is too aggressive.
3. Prepare to introduce a delayed close.

Why this hook / API / idea?: This step deliberately frames the UX/accessibility bug before adding the timer solution in the next steps.

Observe before coding: Move the pointer from the trigger toward the tooltip. The immediate mouseleave behavior makes the floating content disappear before the pointer reaches it.

Full JavaScript checkpoint after Step 14

This step adds understanding rather than JavaScript. The complete runnable checkpoint is repeated below and begins with a NO CODE CHANGE note.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

// STEP 14 CHECKPOINT: No JavaScript change in this conceptual step.
// The complete cumulative code is repeated below so the tutorial remains runnable.

const ToolTip = ({
  children,
  content,
  placement = "top"
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      let top;
      let left;
      switch (placement) {
        case "bottom":
          top = triggerRect.bottom + gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
          break;
        case "left":
          left = triggerRect.left - tooltipRect.width - gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "right":
          left = triggerRect.right + gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "top":
        default:
          top = triggerRect.top - tooltipRect.height - gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
      }
      left = Math.max(
        margin,
        Math.min(left, window.innerWidth - tooltipRect.width - margin)
      );
      top = Math.max(
        margin,
```

```
        Math.min(top, window.innerHeight - tooltipRect.height - margin)
      );
      setStyle({ left, top });
    };
  updatePosition();
  window.addEventListener("resize", updatePosition);
  window.addEventListener("scroll", updatePosition, true);
  return () => {
    window.removeEventListener("resize", updatePosition);
    window.removeEventListener("scroll", updatePosition, true);
  };
  }, [visible, placement]);

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      setHovered(false);
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      style={style}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Tooltip prefers top" placement="top">
          <button className="trigger">Top</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers bottom" placement="bottom">
          <button className="trigger">Bottom</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers left" placement="left">
          <button className="trigger">Left</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers right" placement="right">
          <button className="trigger">Right</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The hover persistence problem is clearly identified before the timer solution.

Step 15 - Store a Timer in a Ref

Goal / problem

We need a short grace period when the pointer leaves the trigger.

New ideas in this step

- setTimeout
- Timer ref
- Delayed close

Logic

1. Store the timeout ID in closeTimerRef.
2. cancelClose() clears a pending timeout.
3. scheduleClose() waits 120ms before clearing hover.
4. Use scheduleClose on trigger mouseleave.

Why this hook / API / idea?: A timer ID is mutable bookkeeping; storing it in state would cause unnecessary renders.

Full JavaScript checkpoint after Step 15

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content,
  placement = "top"
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  // >>> ADDED IN STEP 15: Store the delayed-close timer without causing a render
  const closeTimerRef = useRef(null);
  // <<< END STEP 15 ADDITION

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      let top;
      let left;
      switch (placement) {
        case "bottom":
          top = triggerRect.bottom + gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
          break;
        case "left":
          left = triggerRect.left - tooltipRect.width - gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "right":
          left = triggerRect.right + gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "top":
        default:
          top = triggerRect.top - tooltipRect.height - gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
      }
      left = Math.max(
        margin,
        Math.min(left, window.innerWidth - tooltipRect.width - margin)
      );
      top = Math.max(
        margin,
        Math.min(top, window.innerHeight - tooltipRect.height - margin)
```

```
      );
      setStyle({ left, top });
    };
  updatePosition();
  window.addEventListener("resize", updatePosition);
  window.addEventListener("scroll", updatePosition, true);
  return () => {
    window.removeEventListener("resize", updatePosition);
    window.removeEventListener("scroll", updatePosition, true);
  };
  }, [visible, placement]);

  // >>> ADDED IN STEP 15: Delay close so the pointer can travel from trigger to tooltip
  const cancelClose = () => {
    clearTimeout(closeTimerRef.current);
  };
  const scheduleClose = () => {
    cancelClose();
    closeTimerRef.current = setTimeout(() => {
      setHovered(false);
    }, 120);
  };
  // <<< END STEP 15 ADDITION

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  // >>> ADDED IN STEP 15: Use delayed closing on mouse leave
  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      cancelClose();
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      scheduleClose();
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });
  // <<< END STEP 15 ADDITION

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      style={style}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Tooltip prefers top" placement="top">
          <button className="trigger">Top</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers bottom" placement="bottom">
          <button className="trigger">Bottom</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers left" placement="left">
          <button className="trigger">Left</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers right" placement="right">
          <button className="trigger">Right</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
```

```
    </div>
  </StrictMode>
);
```

Checkpoint: Leaving the trigger starts a short delayed close.

Step 16 - Let the Tooltip Cancel the Close

Goal / problem

If the pointer reaches the tooltip within the grace period, it should stay visible.

New ideas in this step

- Tooltip pointer events
- useEffect cleanup

Logic

1. Cancel the close timer on tooltip mouseenter.
2. Keep hovered=true while pointer is over tooltip.
3. Schedule close on tooltip mouseleave.
4. Clear pending timer when ToolTip unmounts.

Why this hook / API / idea?: The trigger and tooltip now behave as one hover region even though the portal places them far apart in the DOM.

Full JavaScript checkpoint after Step 16

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content,
  placement = "top"
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      let top;
      let left;
      switch (placement) {
        case "bottom":
          top = triggerRect.bottom + gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
          break;
        case "left":
          left = triggerRect.left - tooltipRect.width - gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "right":
          left = triggerRect.right + gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "top":
        default:
          top = triggerRect.top - tooltipRect.height - gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
      }
      left = Math.max(
        margin,
        Math.min(left, window.innerWidth - tooltipRect.width - margin)
      );
      top = Math.max(
        margin,
        Math.min(top, window.innerHeight - tooltipRect.height - margin)
      );
      setStyle({ left, top });
```

```
    };
  updatePosition();
  window.addEventListener("resize", updatePosition);
  window.addEventListener("scroll", updatePosition, true);
  return () => {
    window.removeEventListener("resize", updatePosition);
    window.removeEventListener("scroll", updatePosition, true);
  };
  }, [visible, placement]);

  // >>> ADDED IN STEP 16: Clean up a pending timeout when the component unmounts
  useEffect(() => {
    return () => clearTimeout(closeTimerRef.current);
  }, []);
  // <<< END STEP 16 ADDITION

  const cancelClose = () => {
    clearTimeout(closeTimerRef.current);
  };
  const scheduleClose = () => {
    cancelClose();
    closeTimerRef.current = setTimeout(() => {
      setHovered(false);
    }, 120);
  };

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      cancelClose();
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      scheduleClose();
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

              return (
                <>
                  {trigger}
                  {visible &&
                    // >>> ADDED IN STEP 16: Let pointer entry cancel the close timer so tooltip is hoverable
  createPortal(
                <div
                  id={id}
                  ref={tooltipRef}
                  role="tooltip"
                  className="tooltip"
                  style={style}
                  onMouseEnter={() => { cancelClose(); setHovered(true); }}
                  onMouseLeave={scheduleClose}
                >
                  {content}
                </div>,
                document.body
              )
  // <<< END STEP 16 ADDITION
                  }
                </>
              );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Tooltip prefers top" placement="top">
          <button className="trigger">Top</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers bottom" placement="bottom">
          <button className="trigger">Bottom</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers left" placement="left">
          <button className="trigger">Left</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers right" placement="right">
          <button className="trigger">Right</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
```

```
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The pointer can reach and hover the tooltip itself.

Step 17 - Understand Why Clamping Is Not Auto Layout

Goal / problem

Staying inside the viewport is different from choosing a better side.

New ideas in this step

- Flip vs shift
- Auto layout reasoning

Logic

1. Compare requested top near the top edge.

2. Notice clamping keeps the side conceptually top.

3. Define the desired behavior: try the opposite side when preferred space is insufficient.

Why this hook / API / idea?: This conceptual distinction prevents us from calling simple coordinate clamping "auto placement".

Flip vs shift mental model

```
Requested TOP
    |
    v
Calculate TOP
    |
    v
Clamp coordinates    <-- shift only

Auto layout instead asks:
Requested TOP
    |
    v
Does TOP fit?
  /     \
 yes     no
 |        |
 TOP    BOTTOM / other fallback
```

Full JavaScript checkpoint after Step 17

This step adds understanding rather than JavaScript. The complete runnable checkpoint is repeated below and begins with a NO CODE CHANGE note.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

// STEP 17 CHECKPOINT: No JavaScript change in this conceptual step.
// The complete cumulative code is repeated below so the tutorial remains runnable.

const ToolTip = ({
  children,
  content,
  placement = "top"
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      let top;
      let left;
      switch (placement) {
        case "bottom":
          top = triggerRect.bottom + gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
          break;
        case "left":
          left = triggerRect.left - tooltipRect.width - gap;
```

```
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "right":
          left = triggerRect.right + gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "top":
        default:
          top = triggerRect.top - tooltipRect.height - gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
      }
      left = Math.max(
        margin,
        Math.min(left, window.innerWidth - tooltipRect.width - margin)
      );
      top = Math.max(
        margin,
        Math.min(top, window.innerHeight - tooltipRect.height - margin)
      );
      setStyle({ left, top });
    };
  updatePosition();
  window.addEventListener("resize", updatePosition);
  window.addEventListener("scroll", updatePosition, true);
  return () => {
    window.removeEventListener("resize", updatePosition);
    window.removeEventListener("scroll", updatePosition, true);
  };
  }, [visible, placement]);

  useEffect(() => {
    return () => clearTimeout(closeTimerRef.current);
  }, []);

  const cancelClose = () => {
    clearTimeout(closeTimerRef.current);
  };
  const scheduleClose = () => {
    cancelClose();
    closeTimerRef.current = setTimeout(() => {
      setHovered(false);
    }, 120);
  };

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      cancelClose();
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      scheduleClose();
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      style={style}
      onMouseEnter={() => { cancelClose(); setHovered(true); }}
      onMouseLeave={scheduleClose}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Tooltip prefers top" placement="top">
          <button className="trigger">Top</button>
```

```
        </ToolTip>
        <ToolTip content="Tooltip prefers bottom" placement="bottom">
          <button className="trigger">Bottom</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers left" placement="left">
          <button className="trigger">Left</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers right" placement="right">
          <button className="trigger">Right</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: We now distinguish shifting from flipping.

Step 18 - Measure Available Space

Goal / problem

Auto placement requires a model of the free space around the trigger.

New ideas in this step

- Viewport geometry
- Available space

Logic

1. Measure top space from trigger to viewport edge.
2. Do the same for bottom, left, and right.
3. Subtract margin and gap.

Why this hook / API / idea?: The auto-layout decision should be based on measured constraints, not guesses.

Full JavaScript checkpoint after Step 18

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content,
  placement = "top"
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
  // >>> ADDED IN STEP 18: Measure free space on all four sides of the trigger
      const availableSpace = {
        top: triggerRect.top - margin - gap,
        bottom: window.innerHeight - triggerRect.bottom - margin - gap,
        left: triggerRect.left - margin - gap,
        right: window.innerWidth - triggerRect.right - margin - gap
      };
  // <<< END STEP 18 ADDITION
      let top;
      let left;
      switch (placement) {
        case "bottom":
          top = triggerRect.bottom + gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
          break;
        case "left":
          left = triggerRect.left - tooltipRect.width - gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "right":
          left = triggerRect.right + gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "top":
        default:
          top = triggerRect.top - tooltipRect.height - gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
      }
      left = Math.max(
        margin,
        Math.min(left, window.innerWidth - tooltipRect.width - margin)
      );
```

```
      top = Math.max(
        margin,
        Math.min(top, window.innerHeight - tooltipRect.height - margin)
      );
      setStyle({ left, top });
    };
  updatePosition();
  window.addEventListener("resize", updatePosition);
  window.addEventListener("scroll", updatePosition, true);
  return () => {
    window.removeEventListener("resize", updatePosition);
    window.removeEventListener("scroll", updatePosition, true);
  };
  }, [visible, placement]);

  useEffect(() => {
    return () => clearTimeout(closeTimerRef.current);
  }, []);

  const cancelClose = () => {
    clearTimeout(closeTimerRef.current);
  };
  const scheduleClose = () => {
    cancelClose();
    closeTimerRef.current = setTimeout(() => {
      setHovered(false);
    }, 120);
  };

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      cancelClose();
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      scheduleClose();
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      style={style}
      onMouseEnter={() => { cancelClose(); setHovered(true); }}
      onMouseLeave={scheduleClose}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Tooltip prefers top" placement="top">
          <button className="trigger">Top</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers bottom" placement="bottom">
          <button className="trigger">Bottom</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers left" placement="left">
          <button className="trigger">Left</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers right" placement="right">
          <button className="trigger">Right</button>
        </ToolTip>
      </div>
    </main>
  );
}
```

```
const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The component measures free space on all four sides.

Step 19 - Determine How Much Space We Need

Goal / problem

Available space only matters when compared with the tooltip dimensions.

New ideas in this step

- requiredSpace
- fits predicate

Logic

1. Top/bottom require tooltip height.
2. Left/right require tooltip width.
3. Create fits(side) to make later algorithms readable.

Why this hook / API / idea?: A small predicate turns geometry into a reusable decision rule.

Full JavaScript checkpoint after Step 19

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content,
  placement = "top"
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      const availableSpace = {
        top: triggerRect.top - margin - gap,
        bottom: window.innerHeight - triggerRect.bottom - margin - gap,
        left: triggerRect.left - margin - gap,
        right: window.innerWidth - triggerRect.right - margin - gap
      };
  // >>> ADDED IN STEP 19: Compare available space with the tooltip size
      const requiredSpace = {
        top: tooltipRect.height,
        bottom: tooltipRect.height,
        left: tooltipRect.width,
        right: tooltipRect.width
      };
      const fits = placementName =>
        availableSpace[placementName] >= requiredSpace[placementName];
  // <<< END STEP 19 ADDITION
      let top;
      let left;
      switch (placement) {
        case "bottom":
          top = triggerRect.bottom + gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
          break;
        case "left":
          left = triggerRect.left - tooltipRect.width - gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "right":
          left = triggerRect.right + gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "top":
        default:
```

```
          top = triggerRect.top - tooltipRect.height - gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        }
        left = Math.max(
          margin,
          Math.min(left, window.innerWidth - tooltipRect.width - margin)
        );
        top = Math.max(
          margin,
          Math.min(top, window.innerHeight - tooltipRect.height - margin)
        );
        setStyle({ left, top });
      };
    updatePosition();
    window.addEventListener("resize", updatePosition);
    window.addEventListener("scroll", updatePosition, true);
    return () => {
      window.removeEventListener("resize", updatePosition);
      window.removeEventListener("scroll", updatePosition, true);
    };
    }, [visible, placement]);

    useEffect(() => {
      return () => clearTimeout(closeTimerRef.current);
    }, []);

    const cancelClose = () => {
      clearTimeout(closeTimerRef.current);
    };
    const scheduleClose = () => {
      cancelClose();
      closeTimerRef.current = setTimeout(() => {
        setHovered(false);
      }, 120);
    };

    const handleKeyDown = event => {
      if (event.key === "Escape") {
        setDismissed(true);
      }
    };

    if (!isValidElement(children)) {
      throw new Error("ToolTip requires a single React element.");
    }

    const describedBy = [
      children.props["aria-describedby"],
      id
    ].filter(Boolean).join(" ");

    const trigger = cloneElement(children, {
      ref: triggerRef,
      "aria-describedby": describedBy,
      onMouseEnter: event => {
        children.props.onMouseEnter?.(event);
        setDismissed(false);
        cancelClose();
        setHovered(true);
      },
      onMouseLeave: event => {
        children.props.onMouseLeave?.(event);
        scheduleClose();
      },
      onFocus: event => {
        children.props.onFocus?.(event);
        setDismissed(false);
        setFocused(true);
      },
      onBlur: event => {
        children.props.onBlur?.(event);
        setFocused(false);
      },
      onKeyDown: event => {
        children.props.onKeyDown?.(event);
        handleKeyDown(event);
      }
    });

      return (
        <>
          {trigger}
          {visible &&
            createPortal(
      <div
        id={id}
        ref={tooltipRef}
        role="tooltip"
        className="tooltip"
        style={style}
        onMouseEnter={() => { cancelClose(); setHovered(true); }}
        onMouseLeave={scheduleClose}
      >
        {content}
      </div>,
      document.body
    )
          }
        </>
      );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Tooltip prefers top" placement="top">
          <button className="trigger">Top</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers bottom" placement="bottom">
          <button className="trigger">Bottom</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers left" placement="left">
          <button className="trigger">Left</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers right" placement="right">
```

```
          <button className="trigger">Right</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The component can answer whether a side fits.

Step 20 - Add Auto Flip

Goal / problem

A preferred side should win when it fits, but the component should recover when it does not.

New ideas in this step

- autoFlip prop
- Opposite placement
- Fallback ordering
- Overflow

Logic

1. Try the requested placement first.
2. Try its opposite second.
3. Try remaining sides.
4. If none fit fully, choose the side with least overflow.

Why this hook / API / idea?: Auto flip preserves caller preference while avoiding obviously broken placement near edges.

Full JavaScript checkpoint after Step 20

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content,
  placement = "top",
  autoFlip = true
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      const availableSpace = {
        top: triggerRect.top - margin - gap,
        bottom: window.innerHeight - triggerRect.bottom - margin - gap,
        left: triggerRect.left - margin - gap,
        right: window.innerWidth - triggerRect.right - margin - gap
      };
      const requiredSpace = {
        top: tooltipRect.height,
        bottom: tooltipRect.height,
        left: tooltipRect.width,
        right: tooltipRect.width
      };
      const fits = placementName =>
        availableSpace[placementName] >= requiredSpace[placementName];
  // >>> ADDED IN STEP 20: Choose the actual placement (auto / preferred + flip / fixed)
      const overflow = placementName =>
        Math.max(0, requiredSpace[placementName] - availableSpace[placementName]);
      const allPlacements = ["top", "bottom", "right", "left"];
      const oppositePlacement = { top: "bottom", bottom: "top", left: "right", right: "left" };
      let finalPlacement;
      if (autoFlip) {
        const opposite = oppositePlacement[placement];
        const candidates = [
          placement,
          opposite,
```

```
        ...allPlacements.filter(
          item => item !== placement && item !== opposite
        )
      ];
      finalPlacement = candidates.find(fits);
      if (!finalPlacement) {
        finalPlacement = [...candidates].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else {
      finalPlacement = placement;
    }
// <<< END STEP 20 ADDITION
    let top;
    let left;
    switch (finalPlacement) {
      case "bottom":
        top = triggerRect.bottom + gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        break;
      case "left":
        left = triggerRect.left - tooltipRect.width - gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "right":
        left = triggerRect.right + gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "top":
      default:
        top = triggerRect.top - tooltipRect.height - gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
    }
    left = Math.max(
      margin,
      Math.min(left, window.innerWidth - tooltipRect.width - margin)
    );
    top = Math.max(
      margin,
      Math.min(top, window.innerHeight - tooltipRect.height - margin)
    );
    setStyle({ left, top });
  };
updatePosition();
window.addEventListener("resize", updatePosition);
window.addEventListener("scroll", updatePosition, true);
return () => {
  window.removeEventListener("resize", updatePosition);
  window.removeEventListener("scroll", updatePosition, true);
};
}, [visible, placement, autoFlip]);

useEffect(() => {
  return () => clearTimeout(closeTimerRef.current);
}, []);

const cancelClose = () => {
  clearTimeout(closeTimerRef.current);
};
const scheduleClose = () => {
  cancelClose();
  closeTimerRef.current = setTimeout(() => {
    setHovered(false);
  }, 120);
};

const handleKeyDown = event => {
  if (event.key === "Escape") {
    setDismissed(true);
  }
};

if (!isValidElement(children)) {
  throw new Error("ToolTip requires a single React element.");
}

const describedBy = [
  children.props["aria-describedby"],
  id
].filter(Boolean).join(" ");

const trigger = cloneElement(children, {
  ref: triggerRef,
  "aria-describedby": describedBy,
  onMouseEnter: event => {
    children.props.onMouseEnter?.(event);
    setDismissed(false);
    cancelClose();
    setHovered(true);
  },
  onMouseLeave: event => {
    children.props.onMouseLeave?.(event);
    scheduleClose();
  },
  onFocus: event => {
    children.props.onFocus?.(event);
    setDismissed(false);
    setFocused(true);
  },
  onBlur: event => {
    children.props.onBlur?.(event);
    setFocused(false);
  },
  onKeyDown: event => {
    children.props.onKeyDown?.(event);
    handleKeyDown(event);
  }
});

  return (
    <>
      {trigger}
      {visible &&
        createPortal(
  <div
    id={id}
```

```
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      style={style}
      onMouseEnter={() => { cancelClose(); setHovered(true); }}
      onMouseLeave={scheduleClose}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Tooltip prefers top" placement="top">
          <button className="trigger">Top</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers bottom" placement="bottom">
          <button className="trigger">Bottom</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers left" placement="left">
          <button className="trigger">Left</button>
        </ToolTip>
        <ToolTip content="Tooltip prefers right" placement="right">
          <button className="trigger">Right</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: Preferred placement automatically flips/falls back when needed.

Step 21 - Add placement="auto"

Goal / problem

Sometimes the caller has no preferred side and simply wants the best available location.

New ideas in this step

- Auto placement
- Sorting candidates

Logic

1. Find all sides that fit.

2. Choose the fitting side with the most free space.

3. If nothing fits, choose least overflow.

Why this hook / API / idea?: Auto placement removes manual side selection while keeping the decision deterministic and explainable.

Full JavaScript checkpoint after Step 21

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content,
  placement = "top",
  autoFlip = true
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      const availableSpace = {
        top: triggerRect.top - margin - gap,
        bottom: window.innerHeight - triggerRect.bottom - margin - gap,
        left: triggerRect.left - margin - gap,
        right: window.innerWidth - triggerRect.right - margin - gap
      };
      const requiredSpace = {
        top: tooltipRect.height,
        bottom: tooltipRect.height,
        left: tooltipRect.width,
        right: tooltipRect.width
      };
      const fits = placementName =>
        availableSpace[placementName] >= requiredSpace[placementName];
  // >>> ADDED IN STEP 21: Choose the actual placement (auto / preferred + flip / fixed)
      const overflow = placementName =>
        Math.max(0, requiredSpace[placementName] - availableSpace[placementName]);
      const allPlacements = ["top", "bottom", "right", "left"];
      const oppositePlacement = { top: "bottom", bottom: "top", left: "right", right: "left" };
      let finalPlacement;
      if (placement === "auto") {
        const fittingPlacements = allPlacements.filter(fits);
        if (fittingPlacements.length > 0) {
          finalPlacement = [...fittingPlacements].sort(
            (a, b) => availableSpace[b] - availableSpace[a]
          )[0];
        } else {
          finalPlacement = [...allPlacements].sort(
            (a, b) => overflow(a) - overflow(b)
          )[0];
        }
```

```
    } else if (autoFlip) {
      const opposite = oppositePlacement[placement];
      const candidates = [
        placement,
        opposite,
        ...allPlacements.filter(
          item => item !== placement && item !== opposite
        )
      ];
      finalPlacement = candidates.find(fits);
      if (!finalPlacement) {
        finalPlacement = [...candidates].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else {
      finalPlacement = placement;
    }
// <<< END STEP 21 ADDITION
    let top;
    let left;
    switch (finalPlacement) {
      case "bottom":
        top = triggerRect.bottom + gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        break;
      case "left":
        left = triggerRect.left - tooltipRect.width - gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "right":
        left = triggerRect.right + gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "top":
      default:
        top = triggerRect.top - tooltipRect.height - gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
    }
    left = Math.max(
      margin,
      Math.min(left, window.innerWidth - tooltipRect.width - margin)
    );
    top = Math.max(
      margin,
      Math.min(top, window.innerHeight - tooltipRect.height - margin)
    );
    setStyle({ left, top });
  };
updatePosition();
window.addEventListener("resize", updatePosition);
window.addEventListener("scroll", updatePosition, true);
return () => {
  window.removeEventListener("resize", updatePosition);
  window.removeEventListener("scroll", updatePosition, true);
};
}, [visible, placement, autoFlip]);

useEffect(() => {
  return () => clearTimeout(closeTimerRef.current);
}, []);

const cancelClose = () => {
  clearTimeout(closeTimerRef.current);
};
const scheduleClose = () => {
  cancelClose();
  closeTimerRef.current = setTimeout(() => {
    setHovered(false);
  }, 120);
};

const handleKeyDown = event => {
  if (event.key === "Escape") {
    setDismissed(true);
  }
};

if (!isValidElement(children)) {
  throw new Error("ToolTip requires a single React element.");
}

const describedBy = [
  children.props["aria-describedby"],
  id
].filter(Boolean).join(" ");

const trigger = cloneElement(children, {
  ref: triggerRef,
  "aria-describedby": describedBy,
  onMouseEnter: event => {
    children.props.onMouseEnter?.(event);
    setDismissed(false);
    cancelClose();
    setHovered(true);
  },
  onMouseLeave: event => {
    children.props.onMouseLeave?.(event);
    scheduleClose();
  },
  onFocus: event => {
    children.props.onFocus?.(event);
    setDismissed(false);
    setFocused(true);
  },
  onBlur: event => {
    children.props.onBlur?.(event);
    setFocused(false);
  },
  onKeyDown: event => {
    children.props.onKeyDown?.(event);
    handleKeyDown(event);
  }
});

  return (
    <>
```

```
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      style={style}
      onMouseEnter={() => { cancelClose(); setHovered(true); }}
      onMouseLeave={scheduleClose}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <p>Hover or Tab to a button. Press Escape to dismiss.</p>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Prefer top, but flip if needed" placement="top">
          <button className="trigger">Top + Auto Flip</button>
        </ToolTip>
        <ToolTip content="Prefer bottom, but flip if needed" placement="bottom">
          <button className="trigger">Bottom + Auto Flip</button>
        </ToolTip>
        <ToolTip content="The component chooses the best available side" placement="auto">
          <button className="trigger">Auto Placement</button>
        </ToolTip>
        <ToolTip content="I stay on top even when space is limited" placement="top" autoFlip={false}>
          <button className="trigger">Fixed Top</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: placement="auto" chooses the best side.

Step 22 - Requested Placement vs Actual Placement

Goal / problem

After flipping, the requested side and actual side may differ.

New ideas in this step

- resolvedPlacement
- data-placement

Logic

1. Store the chosen side in state.

2. Use resolvedPlacement for data-placement.

3. Keep placement as the caller request.

Why this hook / API / idea?: This distinction becomes essential for arrows, animations, styling, tests, and debugging.

Full JavaScript checkpoint after Step 22

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

const ToolTip = ({
  children,
  content,
  placement = "top",
  autoFlip = true
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  // >>> ADDED IN STEP 22: Remember the side actually chosen after auto layout
  const [resolvedPlacement, setResolvedPlacement] = useState(
    placement === "auto" ? "top" : placement
  );
  // <<< END STEP 22 ADDITION

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      const availableSpace = {
        top: triggerRect.top - margin - gap,
        bottom: window.innerHeight - triggerRect.bottom - margin - gap,
        left: triggerRect.left - margin - gap,
        right: window.innerWidth - triggerRect.right - margin - gap
      };
      const requiredSpace = {
        top: tooltipRect.height,
        bottom: tooltipRect.height,
        left: tooltipRect.width,
        right: tooltipRect.width
      };
      const fits = placementName =>
        availableSpace[placementName] >= requiredSpace[placementName];
      const overflow = placementName =>
        Math.max(0, requiredSpace[placementName] - availableSpace[placementName]);
      const allPlacements = ["top", "bottom", "right", "left"];
      const oppositePlacement = { top: "bottom", bottom: "top", left: "right", right: "left" };
      let finalPlacement;
      if (placement === "auto") {
        const fittingPlacements = allPlacements.filter(fits);
        if (fittingPlacements.length > 0) {
          finalPlacement = [...fittingPlacements].sort(
            (a, b) => availableSpace[b] - availableSpace[a]
          )[0];
```

```
      } else {
        finalPlacement = [...allPlacements].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else if (autoFlip) {
      const opposite = oppositePlacement[placement];
      const candidates = [
        placement,
        opposite,
        ...allPlacements.filter(
          item => item !== placement && item !== opposite
        )
      ];
      finalPlacement = candidates.find(fits);
      if (!finalPlacement) {
        finalPlacement = [...candidates].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else {
      finalPlacement = placement;
    }
// >>> ADDED IN STEP 22: Expose the actual placement for styling, arrows, testing, and debugging
    setResolvedPlacement(finalPlacement);
// <<< END STEP 22 ADDITION
    let top;
    let left;
    switch (finalPlacement) {
      case "bottom":
        top = triggerRect.bottom + gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        break;
      case "left":
        left = triggerRect.left - tooltipRect.width - gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "right":
        left = triggerRect.right + gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "top":
      default:
        top = triggerRect.top - tooltipRect.height - gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
    }
    left = Math.max(
      margin,
      Math.min(left, window.innerWidth - tooltipRect.width - margin)
    );
    top = Math.max(
      margin,
      Math.min(top, window.innerHeight - tooltipRect.height - margin)
    );
    setStyle({ left, top });
  };
updatePosition();
window.addEventListener("resize", updatePosition);
window.addEventListener("scroll", updatePosition, true);
return () => {
  window.removeEventListener("resize", updatePosition);
  window.removeEventListener("scroll", updatePosition, true);
};
}, [visible, placement, autoFlip]);

useEffect(() => {
  return () => clearTimeout(closeTimerRef.current);
}, []);

const cancelClose = () => {
  clearTimeout(closeTimerRef.current);
};
const scheduleClose = () => {
  cancelClose();
  closeTimerRef.current = setTimeout(() => {
    setHovered(false);
  }, 120);
};

const handleKeyDown = event => {
  if (event.key === "Escape") {
    setDismissed(true);
  }
};

if (!isValidElement(children)) {
  throw new Error("ToolTip requires a single React element.");
}

const describedBy = [
  children.props["aria-describedby"],
  id
].filter(Boolean).join(" ");

const trigger = cloneElement(children, {
  ref: triggerRef,
  "aria-describedby": describedBy,
  onMouseEnter: event => {
    children.props.onMouseEnter?.(event);
    setDismissed(false);
    cancelClose();
    setHovered(true);
  },
  onMouseLeave: event => {
    children.props.onMouseLeave?.(event);
    scheduleClose();
  },
  onFocus: event => {
    children.props.onFocus?.(event);
    setDismissed(false);
    setFocused(true);
  },
  onBlur: event => {
    children.props.onBlur?.(event);
    setFocused(false);
  },
  onKeyDown: event => {
```

```
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

              return (
                <>
                  {trigger}
                  {visible &&
                    // >>> ADDED IN STEP 22: Publish resolvedPlacement through data-placement
  createPortal(
              <div
                id={id}
                ref={tooltipRef}
                role="tooltip"
                className="tooltip"
                data-placement={resolvedPlacement}
                style={style}
                onMouseEnter={() => { cancelClose(); setHovered(true); }}
                onMouseLeave={scheduleClose}
              >
                {content}
              </div>,
              document.body
            )
  // <<< END STEP 22 ADDITION
                  }
                </>
              );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <p>Hover or Tab to a button. Press Escape to dismiss.</p>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Prefer top, but flip if needed" placement="top">
          <button className="trigger">Top + Auto Flip</button>
        </ToolTip>
        <ToolTip content="Prefer bottom, but flip if needed" placement="bottom">
          <button className="trigger">Bottom + Auto Flip</button>
        </ToolTip>
        <ToolTip content="The component chooses the best available side" placement="auto">
          <button className="trigger">Auto Placement</button>
        </ToolTip>
        <ToolTip content="I stay on top even when space is limited" placement="top" autoFlip={false}>
          <button className="trigger">Fixed Top</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The requested placement and actual resolved placement are separate.

Step 23 - The Final Positioning Pipeline

Goal / problem

Pause and name the algorithm we have built.

New ideas in this step

- Pipeline thinking
- Measure -> decide -> position -> shift

Logic

1. Measure trigger and tooltip.
2. Measure free space.
3. Choose or flip placement.
4. Calculate coordinates.
5. Shift inside viewport.

Why this hook / API / idea?: Naming the pipeline makes the code easier to reason about and connects the exercise to production positioning engines.

Final positioning pipeline

```
Visible
  -> Measure trigger + tooltip
  -> Measure available space
  -> Resolve placement (preferred / flip / auto)
  -> Calculate top + left
  -> Shift inside viewport
  -> Render portal
```

Full JavaScript checkpoint after Step 23

This step adds understanding rather than JavaScript. The complete runnable checkpoint is repeated below and begins with a NO CODE CHANGE note.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

// STEP 23 CHECKPOINT: No JavaScript change in this conceptual step.
// The complete cumulative code is repeated below so the tutorial remains runnable.

const ToolTip = ({
  children,
  content,
  placement = "top",
  autoFlip = true
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const [resolvedPlacement, setResolvedPlacement] = useState(
    placement === "auto" ? "top" : placement
  );

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      const availableSpace = {
        top: triggerRect.top - margin - gap,
        bottom: window.innerHeight - triggerRect.bottom - margin - gap,
        left: triggerRect.left - margin - gap,
        right: window.innerWidth - triggerRect.right - margin - gap
      };
      const requiredSpace = {
        top: tooltipRect.height,
        bottom: tooltipRect.height,
        left: tooltipRect.width,
```

```
      right: tooltipRect.width
    };
    const fits = placementName =>
      availableSpace[placementName] >= requiredSpace[placementName];
    const overflow = placementName =>
      Math.max(0, requiredSpace[placementName] - availableSpace[placementName]);
    const allPlacements = ["top", "bottom", "right", "left"];
    const oppositePlacement = { top: "bottom", bottom: "top", left: "right", right: "left" };
    let finalPlacement;
    if (placement === "auto") {
      const fittingPlacements = allPlacements.filter(fits);
      if (fittingPlacements.length > 0) {
        finalPlacement = [...fittingPlacements].sort(
          (a, b) => availableSpace[b] - availableSpace[a]
        )[0];
      } else {
        finalPlacement = [...allPlacements].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else if (autoFlip) {
      const opposite = oppositePlacement[placement];
      const candidates = [
        placement,
        opposite,
        ...allPlacements.filter(
          item => item !== placement && item !== opposite
        )
      ];
      finalPlacement = candidates.find(fits);
      if (!finalPlacement) {
        finalPlacement = [...candidates].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else {
      finalPlacement = placement;
    }
    setResolvedPlacement(finalPlacement);
    let top;
    let left;
    switch (finalPlacement) {
      case "bottom":
        top = triggerRect.bottom + gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        break;
      case "left":
        left = triggerRect.left - tooltipRect.width - gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "right":
        left = triggerRect.right + gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "top":
      default:
        top = triggerRect.top - tooltipRect.height - gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
    }
    left = Math.max(
      margin,
      Math.min(left, window.innerWidth - tooltipRect.width - margin)
    );
    top = Math.max(
      margin,
      Math.min(top, window.innerHeight - tooltipRect.height - margin)
    );
    setStyle({ left, top });
  };
updatePosition();
window.addEventListener("resize", updatePosition);
window.addEventListener("scroll", updatePosition, true);
return () => {
  window.removeEventListener("resize", updatePosition);
  window.removeEventListener("scroll", updatePosition, true);
};
}, [visible, placement, autoFlip]);

useEffect(() => {
  return () => clearTimeout(closeTimerRef.current);
}, []);

const cancelClose = () => {
  clearTimeout(closeTimerRef.current);
};
const scheduleClose = () => {
  cancelClose();
  closeTimerRef.current = setTimeout(() => {
    setHovered(false);
  }, 120);
};

const handleKeyDown = event => {
  if (event.key === "Escape") {
    setDismissed(true);
  }
};

if (!isValidElement(children)) {
  throw new Error("ToolTip requires a single React element.");
}

const describedBy = [
  children.props["aria-describedby"],
  id
].filter(Boolean).join(" ");

const trigger = cloneElement(children, {
  ref: triggerRef,
  "aria-describedby": describedBy,
  onMouseEnter: event => {
    children.props.onMouseEnter?.(event);
    setDismissed(false);
    cancelClose();
    setHovered(true);
  },
  onMouseLeave: event => {
```

```
      children.props.onMouseLeave?.(event);
      scheduleClose();
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      data-placement={resolvedPlacement}
      style={style}
      onMouseEnter={() => { cancelClose(); setHovered(true); }}
      onMouseLeave={scheduleClose}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <p>Hover or Tab to a button. Press Escape to dismiss.</p>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Prefer top, but flip if needed" placement="top">
          <button className="trigger">Top + Auto Flip</button>
        </ToolTip>
        <ToolTip content="Prefer bottom, but flip if needed" placement="bottom">
          <button className="trigger">Bottom + Auto Flip</button>
        </ToolTip>
        <ToolTip content="The component chooses the best available side" placement="auto">
          <button className="trigger">Auto Placement</button>
        </ToolTip>
        <ToolTip content="I stay on top even when space is limited" placement="top" autoFlip={false}>
          <button className="trigger">Fixed Top</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The complete positioning pipeline is named and understood.

Step 24 - Final Complete Component

Goal / problem

Combine every behavior into one final copy-paste component.

New ideas in this step

- Integration
- Production-style component

Logic

1. Keep all accessibility behavior.
2. Keep portal rendering.
3. Keep positioning and auto-layout.
4. Keep cleanup and handler composition.

Why this hook / API / idea?: This is the consolidation step: no new idea should appear without having been motivated earlier.

Full JavaScript checkpoint after Step 24

Replace the entire CodePen JavaScript panel with this cumulative version. Yellow marker comments identify the code introduced or structurally changed in this step.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

// >>> STEP 24: FINAL COMPLETE COMPONENT - all previous features combined
const ToolTip = ({
  children,
  content,
  placement = "top",
  autoFlip = true
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const [resolvedPlacement, setResolvedPlacement] = useState(
    placement === "auto" ? "top" : placement
  );

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      const availableSpace = {
        top: triggerRect.top - margin - gap,
        bottom: window.innerHeight - triggerRect.bottom - margin - gap,
        left: triggerRect.left - margin - gap,
        right: window.innerWidth - triggerRect.right - margin - gap
      };
      const requiredSpace = {
        top: tooltipRect.height,
        bottom: tooltipRect.height,
        left: tooltipRect.width,
        right: tooltipRect.width
      };
      const fits = placementName =>
        availableSpace[placementName] >= requiredSpace[placementName];
      const overflow = placementName =>
        Math.max(0, requiredSpace[placementName] - availableSpace[placementName]);
      const allPlacements = ["top", "bottom", "right", "left"];
      const oppositePlacement = { top: "bottom", bottom: "top", left: "right", right: "left" };
      let finalPlacement;
      if (placement === "auto") {
        const fittingPlacements = allPlacements.filter(fits);
        if (fittingPlacements.length > 0) {
          finalPlacement = [...fittingPlacements].sort(
```

```
          (a, b) => availableSpace[b] - availableSpace[a]
        )[0];
      } else {
        finalPlacement = [...allPlacements].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else if (autoFlip) {
      const opposite = oppositePlacement[placement];
      const candidates = [
        placement,
        opposite,
        ...allPlacements.filter(
          item => item !== placement && item !== opposite
        )
      ];
      finalPlacement = candidates.find(fits);
      if (!finalPlacement) {
        finalPlacement = [...candidates].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else {
      finalPlacement = placement;
    }
    setResolvedPlacement(finalPlacement);
    let top;
    let left;
    switch (finalPlacement) {
      case "bottom":
        top = triggerRect.bottom + gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        break;
      case "left":
        left = triggerRect.left - tooltipRect.width - gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "right":
        left = triggerRect.right + gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "top":
      default:
        top = triggerRect.top - tooltipRect.height - gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
    }
    left = Math.max(
      margin,
      Math.min(left, window.innerWidth - tooltipRect.width - margin)
    );
    top = Math.max(
      margin,
      Math.min(top, window.innerHeight - tooltipRect.height - margin)
    );
    setStyle({ left, top });
  };
updatePosition();
window.addEventListener("resize", updatePosition);
window.addEventListener("scroll", updatePosition, true);
return () => {
  window.removeEventListener("resize", updatePosition);
  window.removeEventListener("scroll", updatePosition, true);
};
}, [visible, placement, autoFlip]);

useEffect(() => {
  return () => clearTimeout(closeTimerRef.current);
}, []);

const cancelClose = () => {
  clearTimeout(closeTimerRef.current);
};
const scheduleClose = () => {
  cancelClose();
  closeTimerRef.current = setTimeout(() => {
    setHovered(false);
  }, 120);
};

const handleKeyDown = event => {
  if (event.key === "Escape") {
    setDismissed(true);
  }
};

if (!isValidElement(children)) {
  throw new Error("ToolTip requires a single React element.");
}

const describedBy = [
  children.props["aria-describedby"],
  id
].filter(Boolean).join(" ");

const trigger = cloneElement(children, {
  ref: triggerRef,
  "aria-describedby": describedBy,
  onMouseEnter: event => {
    children.props.onMouseEnter?.(event);
    setDismissed(false);
    cancelClose();
    setHovered(true);
  },
  onMouseLeave: event => {
    children.props.onMouseLeave?.(event);
    scheduleClose();
  },
  onFocus: event => {
    children.props.onFocus?.(event);
    setDismissed(false);
    setFocused(true);
  },
  onBlur: event => {
    children.props.onBlur?.(event);
    setFocused(false);
  },
  onKeyDown: event => {
```

```
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      data-placement={resolvedPlacement}
      style={style}
      onMouseEnter={() => { cancelClose(); setHovered(true); }}
      onMouseLeave={scheduleClose}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};
// <<< END FINAL COMPONENT

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <p>Hover or Tab to a button. Press Escape to dismiss.</p>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Prefer top, but flip if needed" placement="top">
          <button className="trigger">Top + Auto Flip</button>
        </ToolTip>
        <ToolTip content="Prefer bottom, but flip if needed" placement="bottom">
          <button className="trigger">Bottom + Auto Flip</button>
        </ToolTip>
        <ToolTip content="The component chooses the best available side" placement="auto">
          <button className="trigger">Auto Placement</button>
        </ToolTip>
        <ToolTip content="I stay on top even when space is limited" placement="top" autoFlip={false}>
          <button className="trigger">Fixed Top</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The production-style teaching component is complete.

Step 25 - What Each React Feature Is Doing

Goal / problem

Map each API to one concrete responsibility in the final component.

New ideas in this step

- Responsibility mapping
- Hook selection

Logic

1. Review state vs refs.
2. Review layout effect vs normal effect.
3. Review cloneElement and portal.
4. Review ARIA responsibilities.

Why this hook / API / idea?: A strong React mental model is not memorizing hooks; it is understanding which kind of problem each hook solves.

Feature	Responsibility
Component	Encapsulates Tooltip behavior
Props	Configure content, placement, and autoFlip
children	Allows a compatible element to become the trigger
useState	Stores interaction and calculated layout
Derived state	Calculates visibility without duplicated state
useRef	Holds DOM nodes and timer ID
useId	Creates accessible trigger-tooltip relationship
cloneElement	Enhances the actual trigger without a wrapper
useLayoutEffect	Measures and positions DOM before paint
useEffect	Cleans up delayed-close timer
createPortal	Renders floating UI under document.body
getBoundingClientRect	Measures real browser geometry
aria-describedby	Links trigger to descriptive content
role="tooltip"	Gives floating content tooltip semantics

Full JavaScript checkpoint after Step 25

This step adds understanding rather than JavaScript. The complete runnable checkpoint is repeated below and begins with a NO CODE CHANGE note.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

// STEP 25 CHECKPOINT: No JavaScript change in this conceptual step.
// The complete cumulative code is repeated below so the tutorial remains runnable.

const ToolTip = ({
  children,
  content,
  placement = "top",
  autoFlip = true
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const [resolvedPlacement, setResolvedPlacement] = useState(
```

```
  placement === "auto" ? "top" : placement
);

const visible = (hovered || focused) && !dismissed;

useLayoutEffect(() => {
  if (!visible) return;
  const updatePosition = () => {
    const trigger = triggerRef.current;
    const tooltip = tooltipRef.current;
    if (!trigger || !tooltip) return;
    const triggerRect = trigger.getBoundingClientRect();
    const tooltipRect = tooltip.getBoundingClientRect();
    const gap = 8;
    const margin = 8;
    const availableSpace = {
      top: triggerRect.top - margin - gap,
      bottom: window.innerHeight - triggerRect.bottom - margin - gap,
      left: triggerRect.left - margin - gap,
      right: window.innerWidth - triggerRect.right - margin - gap
    };
    const requiredSpace = {
      top: tooltipRect.height,
      bottom: tooltipRect.height,
      left: tooltipRect.width,
      right: tooltipRect.width
    };
    const fits = placementName =>
      availableSpace[placementName] >= requiredSpace[placementName];
    const overflow = placementName =>
      Math.max(0, requiredSpace[placementName] - availableSpace[placementName]);
    const allPlacements = ["top", "bottom", "right", "left"];
    const oppositePlacement = { top: "bottom", bottom: "top", left: "right", right: "left" };
    let finalPlacement;
    if (placement === "auto") {
      const fittingPlacements = allPlacements.filter(fits);
      if (fittingPlacements.length > 0) {
        finalPlacement = [...fittingPlacements].sort(
          (a, b) => availableSpace[b] - availableSpace[a]
        )[0];
      } else {
        finalPlacement = [...allPlacements].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else if (autoFlip) {
      const opposite = oppositePlacement[placement];
      const candidates = [
        placement,
        opposite,
        ...allPlacements.filter(
          item => item !== placement && item !== opposite
        )
      ];
      finalPlacement = candidates.find(fits);
      if (!finalPlacement) {
        finalPlacement = [...candidates].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else {
      finalPlacement = placement;
    }
    setResolvedPlacement(finalPlacement);
    let top;
    let left;
    switch (finalPlacement) {
      case "bottom":
        top = triggerRect.bottom + gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        break;
      case "left":
        left = triggerRect.left - tooltipRect.width - gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "right":
        left = triggerRect.right + gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "top":
      default:
        top = triggerRect.top - tooltipRect.height - gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
    }
    left = Math.max(
      margin,
      Math.min(left, window.innerWidth - tooltipRect.width - margin)
    );
    top = Math.max(
      margin,
      Math.min(top, window.innerHeight - tooltipRect.height - margin)
    );
    setStyle({ left, top });
  };
updatePosition();
window.addEventListener("resize", updatePosition);
window.addEventListener("scroll", updatePosition, true);
return () => {
  window.removeEventListener("resize", updatePosition);
  window.removeEventListener("scroll", updatePosition, true);
};
}, [visible, placement, autoFlip]);

useEffect(() => {
  return () => clearTimeout(closeTimerRef.current);
}, []);

const cancelClose = () => {
  clearTimeout(closeTimerRef.current);
};
const scheduleClose = () => {
  cancelClose();
  closeTimerRef.current = setTimeout(() => {
    setHovered(false);
  }, 120);
};
```

```
  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      cancelClose();
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      scheduleClose();
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      data-placement={resolvedPlacement}
      style={style}
      onMouseEnter={() => { cancelClose(); setHovered(true); }}
      onMouseLeave={scheduleClose}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <p>Hover or Tab to a button. Press Escape to dismiss.</p>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Prefer top, but flip if needed" placement="top">
          <button className="trigger">Top + Auto Flip</button>
        </ToolTip>
        <ToolTip content="Prefer bottom, but flip if needed" placement="bottom">
          <button className="trigger">Bottom + Auto Flip</button>
        </ToolTip>
        <ToolTip content="The component chooses the best available side" placement="auto">
          <button className="trigger">Auto Placement</button>
        </ToolTip>
        <ToolTip content="I stay on top even when space is limited" placement="top" autoFlip={false}>
          <button className="trigger">Fixed Top</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: Every React API in the final code has a named responsibility.

Step 26 - The Final Architecture

Goal / problem

View the final component as connected subsystems rather than one large function.

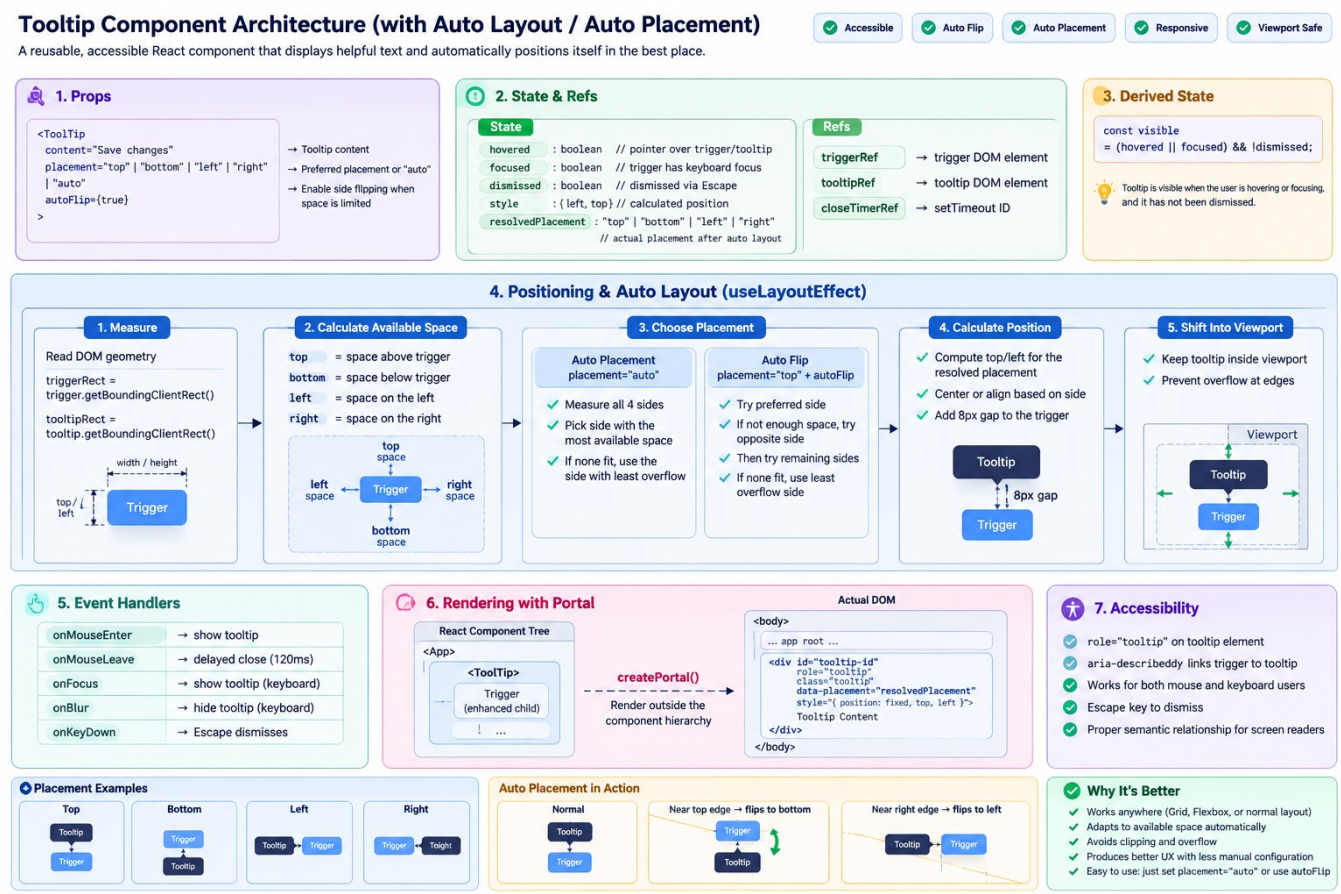
New ideas in this step

- Architecture
- Data/control flow

Logic

1. Props configure behavior.
2. State represents interaction and layout.
3. Refs bridge to DOM.
4. Events update state.
5. Effects measure/clean up.
6. Portal renders floating UI.

Why this hook / API / idea?: Architecture diagrams reduce implementation detail into relationships that are easier to teach and remember.



Full JavaScript checkpoint after Step 26

This step adds understanding rather than JavaScript. The complete runnable checkpoint is repeated below and begins with a NO CODE CHANGE note.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

// STEP 26 CHECKPOINT: No JavaScript change in this conceptual step.
// The complete cumulative code is repeated below so the tutorial remains runnable.

const ToolTip = ({
  children,
```

```
  content,
  placement = "top",
  autoFlip = true
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const [resolvedPlacement, setResolvedPlacement] = useState(
    placement === "auto" ? "top" : placement
  );

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      const availableSpace = {
        top: triggerRect.top - margin - gap,
        bottom: window.innerHeight - triggerRect.bottom - margin - gap,
        left: triggerRect.left - margin - gap,
        right: window.innerWidth - triggerRect.right - margin - gap
      };
      const requiredSpace = {
        top: tooltipRect.height,
        bottom: tooltipRect.height,
        left: tooltipRect.width,
        right: tooltipRect.width
      };
      const fits = placementName =>
        availableSpace[placementName] >= requiredSpace[placementName];
      const overflow = placementName =>
        Math.max(0, requiredSpace[placementName] - availableSpace[placementName]);
      const allPlacements = ["top", "bottom", "right", "left"];
      const oppositePlacement = { top: "bottom", bottom: "top", left: "right", right: "left" };
      let finalPlacement;
      if (placement === "auto") {
        const fittingPlacements = allPlacements.filter(fits);
        if (fittingPlacements.length > 0) {
          finalPlacement = [...fittingPlacements].sort(
            (a, b) => availableSpace[b] - availableSpace[a]
          )[0];
        } else {
          finalPlacement = [...allPlacements].sort(
            (a, b) => overflow(a) - overflow(b)
          )[0];
        }
      } else if (autoFlip) {
        const opposite = oppositePlacement[placement];
        const candidates = [
          placement,
          opposite,
          ...allPlacements.filter(
            item => item !== placement && item !== opposite
          )
        ];
        finalPlacement = candidates.find(fits);
        if (!finalPlacement) {
          finalPlacement = [...candidates].sort(
            (a, b) => overflow(a) - overflow(b)
          )[0];
        }
      } else {
        finalPlacement = placement;
      }
      setResolvedPlacement(finalPlacement);
      let top;
      let left;
      switch (finalPlacement) {
        case "bottom":
          top = triggerRect.bottom + gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
          break;
        case "left":
          left = triggerRect.left - tooltipRect.width - gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "right":
          left = triggerRect.right + gap;
          top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
          break;
        case "top":
        default:
          top = triggerRect.top - tooltipRect.height - gap;
          left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
      }
      left = Math.max(
        margin,
        Math.min(left, window.innerWidth - tooltipRect.width - margin)
      );
      top = Math.max(
        margin,
        Math.min(top, window.innerHeight - tooltipRect.height - margin)
      );
      setStyle({ left, top });
    };
  updatePosition();
  window.addEventListener("resize", updatePosition);
  window.addEventListener("scroll", updatePosition, true);
```

```
  return () => {
    window.removeEventListener("resize", updatePosition);
    window.removeEventListener("scroll", updatePosition, true);
  };
  }, [visible, placement, autoFlip]);

  useEffect(() => {
    return () => clearTimeout(closeTimerRef.current);
  }, []);

  const cancelClose = () => {
    clearTimeout(closeTimerRef.current);
  };
  const scheduleClose = () => {
    cancelClose();
    closeTimerRef.current = setTimeout(() => {
      setHovered(false);
    }, 120);
  };

  const handleKeyDown = event => {
    if (event.key === "Escape") {
      setDismissed(true);
    }
  };

  if (!isValidElement(children)) {
    throw new Error("ToolTip requires a single React element.");
  }

  const describedBy = [
    children.props["aria-describedby"],
    id
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      cancelClose();
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      scheduleClose();
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      data-placement={resolvedPlacement}
      style={style}
      onMouseEnter={() => { cancelClose(); setHovered(true); }}
      onMouseLeave={scheduleClose}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <p>Hover or Tab to a button. Press Escape to dismiss.</p>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Prefer top, but flip if needed" placement="top">
          <button className="trigger">Top + Auto Flip</button>
        </ToolTip>
        <ToolTip content="Prefer bottom, but flip if needed" placement="bottom">
          <button className="trigger">Bottom + Auto Flip</button>
        </ToolTip>
        <ToolTip content="The component chooses the best available side" placement="auto">
          <button className="trigger">Auto Placement</button>
        </ToolTip>
        <ToolTip content="I stay on top even when space is limited" placement="top" autoFlip={false}>
          <button className="trigger">Fixed Top</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
```

```
);
```

Checkpoint: The final architecture is visible as props/state/refs/events/effects/rendering.

Step 27 - Important Concept: React vs the Browser

Goal / problem

This component sits exactly at the boundary between declarative React and imperative browser geometry.

New ideas in this step

- Declarative UI
- Imperative DOM measurement

Logic

1. React renders from state.
2. The browser owns actual pixels.
3. Refs expose DOM nodes.
4. useLayoutEffect reads geometry and feeds calculated state back into React.

Why this hook / API / idea?: Tooltips are a good example of appropriate DOM access in React because their job inherently depends on browser layout.

React / browser feedback loop

```
React State
    |
    v
React Render
    |
    v
Browser DOM
    |
    v
DOM Measurement (refs + getBoundingClientRect)
    |
    v
Calculated React State
    |
    v
React Render
```

Full JavaScript checkpoint after Step 27

This step adds understanding rather than JavaScript. The complete runnable checkpoint is repeated below and begins with a NO CODE CHANGE note.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

// STEP 27 CHECKPOINT: No JavaScript change in this conceptual step.
// The complete cumulative code is repeated below so the tutorial remains runnable.

const ToolTip = ({
  children,
  content,
  placement = "top",
  autoFlip = true
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const [resolvedPlacement, setResolvedPlacement] = useState(
    placement === "auto" ? "top" : placement
  );

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      const availableSpace = {
        top: triggerRect.top - margin - gap,
        bottom: window.innerHeight - triggerRect.bottom - margin - gap,
```

```
      left: triggerRect.left - margin - gap,
      right: window.innerWidth - triggerRect.right - margin - gap
    };
    const requiredSpace = {
      top: tooltipRect.height,
      bottom: tooltipRect.height,
      left: tooltipRect.width,
      right: tooltipRect.width
    };
    const fits = placementName =>
      availableSpace[placementName] >= requiredSpace[placementName];
    const overflow = placementName =>
      Math.max(0, requiredSpace[placementName] - availableSpace[placementName]);
    const allPlacements = ["top", "bottom", "right", "left"];
    const oppositePlacement = { top: "bottom", bottom: "top", left: "right", right: "left" };
    let finalPlacement;
    if (placement === "auto") {
      const fittingPlacements = allPlacements.filter(fits);
      if (fittingPlacements.length > 0) {
        finalPlacement = [...fittingPlacements].sort(
          (a, b) => availableSpace[b] - availableSpace[a]
        )[0];
      } else {
        finalPlacement = [...allPlacements].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else if (autoFlip) {
      const opposite = oppositePlacement[placement];
      const candidates = [
        placement,
        opposite,
        ...allPlacements.filter(
          item => item !== placement && item !== opposite
        )
      ];
      finalPlacement = candidates.find(fits);
      if (!finalPlacement) {
        finalPlacement = [...candidates].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else {
      finalPlacement = placement;
    }
    setResolvedPlacement(finalPlacement);
    let top;
    let left;
    switch (finalPlacement) {
      case "bottom":
        top = triggerRect.bottom + gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        break;
      case "left":
        left = triggerRect.left - tooltipRect.width - gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "right":
        left = triggerRect.right + gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "top":
      default:
        top = triggerRect.top - tooltipRect.height - gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
    }
    left = Math.max(
      margin,
      Math.min(left, window.innerWidth - tooltipRect.width - margin)
    );
    top = Math.max(
      margin,
      Math.min(top, window.innerHeight - tooltipRect.height - margin)
    );
    setStyle({ left, top });
  };
updatePosition();
window.addEventListener("resize", updatePosition);
window.addEventListener("scroll", updatePosition, true);
return () => {
  window.removeEventListener("resize", updatePosition);
  window.removeEventListener("scroll", updatePosition, true);
};
}, [visible, placement, autoFlip]);

useEffect(() => {
  return () => clearTimeout(closeTimerRef.current);
}, []);

const cancelClose = () => {
  clearTimeout(closeTimerRef.current);
};
const scheduleClose = () => {
  cancelClose();
  closeTimerRef.current = setTimeout(() => {
    setHovered(false);
  }, 120);
};

const handleKeyDown = event => {
  if (event.key === "Escape") {
    setDismissed(true);
  }
};

if (!isValidElement(children)) {
  throw new Error("ToolTip requires a single React element.");
}

const describedBy = [
  children.props["aria-describedby"],
  id
].filter(Boolean).join(" ");

const trigger = cloneElement(children, {
  ref: triggerRef,
  "aria-describedby": describedBy,
```

```
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      cancelClose();
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      scheduleClose();
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      data-placement={resolvedPlacement}
      style={style}
      onMouseEnter={() => { cancelClose(); setHovered(true); }}
      onMouseLeave={scheduleClose}
    >
      {content}
    </div>,
    document.body
  )
      }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <p>Hover or Tab to a button. Press Escape to dismiss.</p>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Prefer top, but flip if needed" placement="top">
          <button className="trigger">Top + Auto Flip</button>
        </ToolTip>
        <ToolTip content="Prefer bottom, but flip if needed" placement="bottom">
          <button className="trigger">Bottom + Auto Flip</button>
        </ToolTip>
        <ToolTip content="The component chooses the best available side" placement="auto">
          <button className="trigger">Auto Placement</button>
        </ToolTip>
        <ToolTip content="I stay on top even when space is limited" placement="top" autoFlip={false}>
          <button className="trigger">Fixed Top</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The React-browser measurement feedback loop is understood.

Step 28 - Accessibility Boundary

Goal / problem

A tooltip is descriptive floating content, not an interactive mini-dialog.

New ideas in this step

- Tooltip vs popover/dialog/menu
- Accessibility boundary

Logic

1. Keep tooltip content informational.

2. Do not put focusable controls inside role="tooltip".

3. Use a popover/dialog/menu pattern when interaction is required.

Why this hook / API / idea?: Correct semantics depend on behavior. Once users must focus or operate content inside the floating surface, the component pattern changes.

Accessibility boundary: Informational floating content can be a tooltip. If the floating surface contains buttons, links, form controls, or requires focus management, use an appropriate popover/dialog/menu pattern instead.

Full JavaScript checkpoint after Step 28

This step adds understanding rather than JavaScript. The complete runnable checkpoint is repeated below and begins with a NO CODE CHANGE note.

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

// STEP 28 CHECKPOINT: No JavaScript change in this conceptual step.
// The complete cumulative code is repeated below so the tutorial remains runnable.

const ToolTip = ({
  children,
  content,
  placement = "top",
  autoFlip = true
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const [resolvedPlacement, setResolvedPlacement] = useState(
    placement === "auto" ? "top" : placement
  );

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
      const gap = 8;
      const margin = 8;
      const availableSpace = {
        top: triggerRect.top - margin - gap,
        bottom: window.innerHeight - triggerRect.bottom - margin - gap,
        left: triggerRect.left - margin - gap,
        right: window.innerWidth - triggerRect.right - margin - gap
      };
      const requiredSpace = {
        top: tooltipRect.height,
        bottom: tooltipRect.height,
        left: tooltipRect.width,
        right: tooltipRect.width
      };
      const fits = placementName =>
        availableSpace[placementName] >= requiredSpace[placementName];
      const overflow = placementName =>
        Math.max(0, requiredSpace[placementName] - availableSpace[placementName]);
      const allPlacements = ["top", "bottom", "right", "left"];
```

```
    const oppositePlacement = { top: "bottom", bottom: "top", left: "right", right: "left" };
    let finalPlacement;
    if (placement === "auto") {
      const fittingPlacements = allPlacements.filter(fits);
      if (fittingPlacements.length > 0) {
        finalPlacement = [...fittingPlacements].sort(
          (a, b) => availableSpace[b] - availableSpace[a]
        )[0];
      } else {
        finalPlacement = [...allPlacements].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else if (autoFlip) {
      const opposite = oppositePlacement[placement];
      const candidates = [
        placement,
        opposite,
        ...allPlacements.filter(
          item => item !== placement && item !== opposite
        )
      ];
      finalPlacement = candidates.find(fits);
      if (!finalPlacement) {
        finalPlacement = [...candidates].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else {
      finalPlacement = placement;
    }
    setResolvedPlacement(finalPlacement);
    let top;
    let left;
    switch (finalPlacement) {
      case "bottom":
        top = triggerRect.bottom + gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        break;
      case "left":
        left = triggerRect.left - tooltipRect.width - gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "right":
        left = triggerRect.right + gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "top":
      default:
        top = triggerRect.top - tooltipRect.height - gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
    }
    left = Math.max(
      margin,
      Math.min(left, window.innerWidth - tooltipRect.width - margin)
    );
    top = Math.max(
      margin,
      Math.min(top, window.innerHeight - tooltipRect.height - margin)
    );
    setStyle({ left, top });
  };
updatePosition();
window.addEventListener("resize", updatePosition);
window.addEventListener("scroll", updatePosition, true);
return () => {
  window.removeEventListener("resize", updatePosition);
  window.removeEventListener("scroll", updatePosition, true);
};
}, [visible, placement, autoFlip]);

useEffect(() => {
  return () => clearTimeout(closeTimerRef.current);
}, []);

const cancelClose = () => {
  clearTimeout(closeTimerRef.current);
};
const scheduleClose = () => {
  cancelClose();
  closeTimerRef.current = setTimeout(() => {
    setHovered(false);
  }, 120);
};

const handleKeyDown = event => {
  if (event.key === "Escape") {
    setDismissed(true);
  }
};

if (!isValidElement(children)) {
  throw new Error("ToolTip requires a single React element.");
}

const describedBy = [
  children.props["aria-describedby"],
  id
].filter(Boolean).join(" ");

const trigger = cloneElement(children, {
  ref: triggerRef,
  "aria-describedby": describedBy,
  onMouseEnter: event => {
    children.props.onMouseEnter?.(event);
    setDismissed(false);
    cancelClose();
    setHovered(true);
  },
  onMouseLeave: event => {
    children.props.onMouseLeave?.(event);
    scheduleClose();
  },
  onFocus: event => {
    children.props.onFocus?.(event);
    setDismissed(false);
    setFocused(true);
```

```
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      data-placement={resolvedPlacement}
      style={style}
      onMouseEnter={() => { cancelClose(); setHovered(true); }}
      onMouseLeave={scheduleClose}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <p>Hover or Tab to a button. Press Escape to dismiss.</p>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Prefer top, but flip if needed" placement="top">
          <button className="trigger">Top + Auto Flip</button>
        </ToolTip>
        <ToolTip content="Prefer bottom, but flip if needed" placement="bottom">
          <button className="trigger">Bottom + Auto Flip</button>
        </ToolTip>
        <ToolTip content="The component chooses the best available side" placement="auto">
          <button className="trigger">Auto Placement</button>
        </ToolTip>
        <ToolTip content="I stay on top even when space is limited" placement="top" autoFlip={false}>
          <button className="trigger">Fixed Top</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```

Checkpoint: The semantic boundary between tooltip and interactive overlays is clear.

Appendix - Final CodePen Panels

After completing the step-by-step progression, these are the final three CodePen panels in one place.

HTML

```
<div id="root"></div>
```

CSS

```
* { box-sizing: border-box; }
body {
  margin: 0;
  font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", sans-serif;
  background: #f5f6f8;
  color: #222;
}
.app { min-height: 100vh; }
.demo {
  width: min(1100px, calc(100% - 32px));
  margin: 0 auto;
  padding: 48px 0 80px;
}
.tooltip {
  position: fixed;
  z-index: 1000;
  background-color: #1f1f1f;
  color: #fff;
  padding: 6px 10px;
  border-radius: 6px;
  font-size: 0.875rem;
  line-height: 1.4;
  font-weight: 400;
  max-width: 280px;
  white-space: normal;
  overflow-wrap: break-word;
  box-shadow:
    0 2px 4px rgb(0 0 0 / 0.15),
    0 4px 12px rgb(0 0 0 / 0.2);
  pointer-events: auto;
  transition: opacity 120ms ease, transform 120ms ease;
}
.trigger {
  padding: 9px 14px;
  border: 1px solid #aaa;
  border-radius: 6px;
  background: #fff;
  color: #222;
  font: inherit;
  cursor: pointer;
}
.trigger:focus-visible {
  outline: 3px solid #79a7ff;
  outline-offset: 3px;
}
@media (prefers-reduced-motion: reduce) {
  .tooltip { transition: none; }
}
```

JavaScript (Babel)

```
import React, {
  StrictMode,
  cloneElement,
  isValidElement,
  useEffect,
  useId,
  useLayoutEffect,
  useRef,
  useState
} from "https://esm.sh/react@19";
import { createRoot } from "https://esm.sh/react-dom@19/client";
import { createPortal } from "https://esm.sh/react-dom@19";

// >>> STEP 24: FINAL COMPLETE COMPONENT - all previous features combined
const ToolTip = ({
  children,
  content,
  placement = "top",
  autoFlip = true
}) => {
  const id = useId();

  const triggerRef = useRef(null);
    const tooltipRef = useRef(null);

  const closeTimerRef = useRef(null);

  const [hovered, setHovered] = useState(false);

  const [focused, setFocused] = useState(false);

  const [dismissed, setDismissed] = useState(false);

  const [style, setStyle] = useState({});

  const [resolvedPlacement, setResolvedPlacement] = useState(
    placement === "auto" ? "top" : placement
  );

  const visible = (hovered || focused) && !dismissed;

  useLayoutEffect(() => {
    if (!visible) return;
    const updatePosition = () => {
      const trigger = triggerRef.current;
      const tooltip = tooltipRef.current;
      if (!trigger || !tooltip) return;
      const triggerRect = trigger.getBoundingClientRect();
      const tooltipRect = tooltip.getBoundingClientRect();
```

```
    const gap = 8;
    const margin = 8;
    const availableSpace = {
      top: triggerRect.top - margin - gap,
      bottom: window.innerHeight - triggerRect.bottom - margin - gap,
      left: triggerRect.left - margin - gap,
      right: window.innerWidth - triggerRect.right - margin - gap
    };
    const requiredSpace = {
      top: tooltipRect.height,
      bottom: tooltipRect.height,
      left: tooltipRect.width,
      right: tooltipRect.width
    };
    const fits = placementName =>
      availableSpace[placementName] >= requiredSpace[placementName];
    const overflow = placementName =>
      Math.max(0, requiredSpace[placementName] - availableSpace[placementName]);
    const allPlacements = ["top", "bottom", "right", "left"];
    const oppositePlacement = { top: "bottom", bottom: "top", left: "right", right: "left" };
    let finalPlacement;
    if (placement === "auto") {
      const fittingPlacements = allPlacements.filter(fits);
      if (fittingPlacements.length > 0) {
        finalPlacement = [...fittingPlacements].sort(
          (a, b) => availableSpace[b] - availableSpace[a]
        )[0];
      } else {
        finalPlacement = [...allPlacements].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else if (autoFlip) {
      const opposite = oppositePlacement[placement];
      const candidates = [
        placement,
        opposite,
        ...allPlacements.filter(
          item => item !== placement && item !== opposite
        )
      ];
      finalPlacement = candidates.find(fits);
      if (!finalPlacement) {
        finalPlacement = [...candidates].sort(
          (a, b) => overflow(a) - overflow(b)
        )[0];
      }
    } else {
      finalPlacement = placement;
    }
    setResolvedPlacement(finalPlacement);
    let top;
    let left;
    switch (finalPlacement) {
      case "bottom":
        top = triggerRect.bottom + gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
        break;
      case "left":
        left = triggerRect.left - tooltipRect.width - gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "right":
        left = triggerRect.right + gap;
        top = triggerRect.top + triggerRect.height / 2 - tooltipRect.height / 2;
        break;
      case "top":
      default:
        top = triggerRect.top - tooltipRect.height - gap;
        left = triggerRect.left + triggerRect.width / 2 - tooltipRect.width / 2;
    }
    left = Math.max(
      margin,
      Math.min(left, window.innerWidth - tooltipRect.width - margin)
    );
    top = Math.max(
      margin,
      Math.min(top, window.innerHeight - tooltipRect.height - margin)
    );
    setStyle({ left, top });
  };
updatePosition();
window.addEventListener("resize", updatePosition);
window.addEventListener("scroll", updatePosition, true);
return () => {
  window.removeEventListener("resize", updatePosition);
  window.removeEventListener("scroll", updatePosition, true);
};
}, [visible, placement, autoFlip]);

useEffect(() => {
  return () => clearTimeout(closeTimerRef.current);
}, []);

const cancelClose = () => {
  clearTimeout(closeTimerRef.current);
};
const scheduleClose = () => {
  cancelClose();
  closeTimerRef.current = setTimeout(() => {
    setHovered(false);
  }, 120);
};

const handleKeyDown = event => {
  if (event.key === "Escape") {
    setDismissed(true);
  }
};

if (!isValidElement(children)) {
  throw new Error("ToolTip requires a single React element.");
}

const describedBy = [
  children.props["aria-describedby"],
  id
```

```
  ].filter(Boolean).join(" ");

  const trigger = cloneElement(children, {
    ref: triggerRef,
    "aria-describedby": describedBy,
    onMouseEnter: event => {
      children.props.onMouseEnter?.(event);
      setDismissed(false);
      cancelClose();
      setHovered(true);
    },
    onMouseLeave: event => {
      children.props.onMouseLeave?.(event);
      scheduleClose();
    },
    onFocus: event => {
      children.props.onFocus?.(event);
      setDismissed(false);
      setFocused(true);
    },
    onBlur: event => {
      children.props.onBlur?.(event);
      setFocused(false);
    },
    onKeyDown: event => {
      children.props.onKeyDown?.(event);
      handleKeyDown(event);
    }
  });

    return (
      <>
        {trigger}
        {visible &&
          createPortal(
    <div
      id={id}
      ref={tooltipRef}
      role="tooltip"
      className="tooltip"
      data-placement={resolvedPlacement}
      style={style}
      onMouseEnter={() => { cancelClose(); setHovered(true); }}
      onMouseLeave={scheduleClose}
    >
      {content}
    </div>,
    document.body
  )
        }
      </>
    );
};
// <<< END FINAL COMPONENT

function App() {
  return (
    <main className="demo">
      <h1>Accessible React Tooltip</h1>
      <p>Hover or Tab to a button. Press Escape to dismiss.</p>
      <div style={{ display: "flex", flexWrap: "wrap", gap: "24px", marginTop: "80px" }}>
        <ToolTip content="Prefer top, but flip if needed" placement="top">
          <button className="trigger">Top + Auto Flip</button>
        </ToolTip>
        <ToolTip content="Prefer bottom, but flip if needed" placement="bottom">
          <button className="trigger">Bottom + Auto Flip</button>
        </ToolTip>
        <ToolTip content="The component chooses the best available side" placement="auto">
          <button className="trigger">Auto Placement</button>
        </ToolTip>
        <ToolTip content="I stay on top even when space is limited" placement="top" autoFlip={false}>
          <button className="trigger">Fixed Top</button>
        </ToolTip>
      </div>
    </main>
  );
}

const root = createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <div className="app">
      <App />
    </div>
  </StrictMode>
);
```