drei Varianten (Typ+Klasse+Pseudo, Descendant, Child) → das deckt die Kernprinzipien perfekt ab.

Ganz genau, das hat einen Fachbegriff 👌

- `.app-footer` → **Klassenselektor**
- `footer` → **Typselektor (Elementselektor)**
- `footer.app-footer` → ein **kombinierter Selektor** (offiziell: **Typselektor + Klassenselektor** → auch „qualifizierter Klassenselektor“ genannt).

Terminologie in CSS-Spezifikation

- **Selektor**: alles, was vor den geschweiften Klammern steht.
- **Einfache Selektoren** (*simple selectors*): `footer`, `.app-footer`, `#id`, `[attr]`, `:hover` …
- **Kombinatoren**: Verknüpfen einfache Selektoren (`footer .app-footer`, `div > p`).
- **Selektorlisten**: mehrere Selektoren durch Komma getrennt (`h1, h2, .title`).

👉 `footer.app-footer` ist also **eine Selektorfolge** aus zwei einfachen Selektoren (`footer` und `.app-footer`), ohne Leerzeichen → das bedeutet **UND-Bedingung**: *„ein Element, das ein `<footer>` ist **und** die Klasse `app-footer` hat“*.

✅ Fachlich korrekt würdest du sagen:

- `.app-footer` ist ein **Klassenselektor**.
- `footer.app-footer` ist ein **qualifizierter Klassenselektor** (ein Elementselektor kombiniert mit einem Klassenselektor).

Spickzettel, wie der Browser Selektoren intern in einfache Bausteine zerlegt und verarbeitet.

CSS Selector Breakdown Cheat Sheet

Beispiel	Zerlegt in …	Typ Fachbegriff	Bedeutung im DOM
`div`	`div`	**Typselektor**	Matcht jedes `<div>`-Element
`.app-footer`	`.app-footer`	**Klassenselektor**	Matcht jedes Element mit `class="app-footer"`
`#main`	`#main`	**ID-Selektor**	Matcht das Element mit `id="main"`
`[disabled]`	`[disabled]`	**Attributselektor**	Matcht jedes Element mit Attribut `disabled`
`[type="text"]`	`[type="text"]`	**Attributselektor**	Matcht `<input type="text">`
`a:hover`	`a` + `:hover`	Typ + **Pseudoklasse**	Matcht `<a>`-Links im Hover-Zustand
`p::first-line`	`p` + `::first-line`	Typ + **Pseudoelement**	Matcht die erste Zeile im `<p>`
`footer.app-footer`	`footer` + `.app-footer`	Typ + Klasse	`<footer class="app-footer">`
`footer.app-footer:hover`	`footer` + `.app-footer` + `:hover`	Typ + Klasse + Pseudoklasse	`<footer class="app-footer">`, wenn Maus drüber
`ul > li`	`ul` + `>` + `li`	Typ + **Kind-Kombinator**	`<li>` direkt in `<ul>`

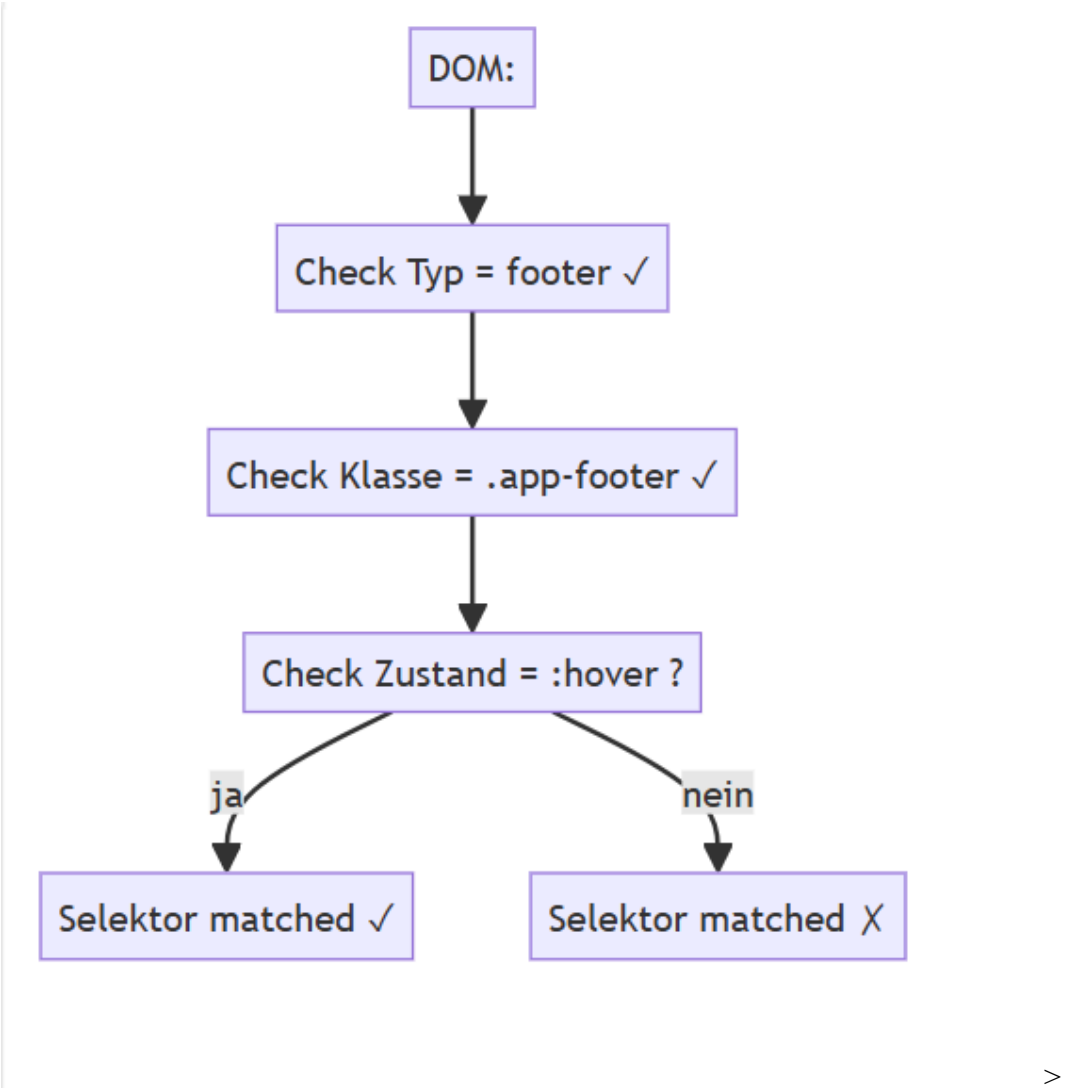
Beispiel	Zerlegt in …	Typ Fachbegriff	Bedeutung im DOM
`ul li`	`ul` + + `li`	Typ + **Descendant-Kombinator**	jedes `<li>` irgendwo innerhalb von `<ul>`
`h1, h2, .title`	`h1` , `h2` , `.title`	**Selektorliste**	Mehrere Selektoren gleichzeitig

---

Wie der Browser denkt

- **Selektorfolge** = Liste von einfachen Selektoren (Typ, Klasse, ID, Attribut, Pseudoklasse, Pseudoelement), die alle auf *dasselbe Element* zutreffen müssen.
- **Kombinatoren** (, >, +, ~) verbinden mehrere Selektoren über **Eltern-Kind- oder Nachbarschafts-Beziehungen**.
- **Selektorliste** (Komma-getrennt) = OR-Verknüpfung → reicht, wenn einer passt.

Visualisierung Beispiel `footer.app-footer:hover`



>

Hier eine kleine Übersicht der **Selektortypen in CSS** mit Beispielen und Fachbegriffen:

🎯 Grundtypen

Selektor	Beispiel	Fachbegriff laut Spezifikation	Bedeutung
`footer`	`footer { … }`	**Typselektor** (Type selector)	Alle `<footer>`-Elemente
`.app-footer`	`.app-footer { … }`	**Klassenselektor** (Class selector)	Alle Elemente mit `class="app-footer"`
`#main`	`#main { … }`	**ID-Selektor** (ID selector)	Element mit `id="main"` (sollte nur 1x vorkommen)
`*`	`* { … }`	**Universalselektor** (Universal selector)	Alle Elemente

---

🎯 Attribute & Pseudoklassen

Selektor	Beispiel	Fachbegriff	Bedeutung
`[disabled]`	`input[disabled]`	**Attributselektor** (Attribute selector)	`<input>` mit Attribut `disabled`
`[type="text"]`	`input[type="text"]`	**Attributselektor mit Wert**	Nur `<input type="text">`
`a:hover`	`a:hover { … }`	**Pseudoklasse** (Pseudo-class)	Links, wenn Maus drüber schwebt
`p::first-line`	`p::first-line { … }`	**Pseudoelement** (Pseudo-element)	Erste Zeile im Absatz

---

🎯 Kombinationen (Selektorfolgen & Kombinatoren)

Selektor	Beispiel	Fachbegriff	Bedeutung
`footer.app-footer`	`footer.app-footer { … }`	**Qualifizierter Klassenselektor** (Type + Class)	Nur `<footer class="app-footer">`
`div > p`	`div > p { … }`	**Kind-Kombinator** (Child combinator)	`<p>` direkt innerhalb von `<div>`
`ul li`	`ul li { … }`	**Nachfahrenselektor** (Descendant combinator)	Alle `<li>` irgendwo in `<ul>`
`h1, h2, .title`	`h1, h2, .title { … }`	**Selektorliste** (Selector list)	Mehrere Selektoren gleichzeitig
`button.primary:hover`	`button.primary:hover { … }`	**Selektorfolge** (Simple selectors in a sequence)	`<button class="primary">`, wenn Maus drüber ist

---

🎯 Spezifität (Wichtigkeit)

CSS-Browser-Berechnung (je „stärker“, desto höher gewinnt):

- Inline-Style (im `style="…"`) → am stärksten
- ID-Selektoren (`#id`) → stark
- Klassenselektoren, Attribute, Pseudoklassen (`.class`, `[attr]`, `:hover`) → mittel
- Typselektoren (`div`, `footer`) → schwach
- Universalselektor (`*`) → sehr schwach

✅ Damit hast du:

- `.app-footer` = **Klassenselektor**
- `footer.app-footer` = **qualifizierter Klassenselektor** (Type + Class)

Sehr gute Frage 👍 – so wie du es formulierst („eine Zeile nach der anderen vom Sourcecode?“) denkt man an einen sequentiellen Text-Parser.
Der Browser arbeitet aber anders:

---

1. DOM wird aus dem HTML gebaut

- Der HTML-Parser läuft tatsächlich **zeilenweise durch das Dokument**.
- Dabei baut er einen **Baum** auf: das **DOM** (Document Object Model).
- Das DOM ist nicht „Text mit Zeilen“, sondern eine **Baumstruktur von Knoten** (Elemente, Attribute, Textknoten).

---

2. CSS-Selektoren arbeiten nicht auf dem Quelltext

- Wenn der Browser Styles anwenden will, nimmt er **den fertigen DOM-Baum**.
- Selektoren wie `footer.app-footer:hover` werden als **Matcher** kompiliert.
- Dann wird geprüft: „Gibt es im DOM ein Element vom Typ `<footer>`, mit Klasse `app-footer`, das gerade im Zustand `:hover` ist?“

---

3. Matching-Strategie

- Intern wird meist **von rechts nach links** geprüft, nicht zeilenweise:
  - Bei `footer .tabs .active` schaut der Browser zuerst: „Welche Elemente haben `.active`?“
  - Dann prüft er, ob deren Eltern `.tabs` sind, und ob darüber ein `<footer>` ist.
- Grund: Das ist effizienter als „oben im DOM loslaufen und alles runterprüfen“.

---

4. Zustände (Pseudo-Klassen)

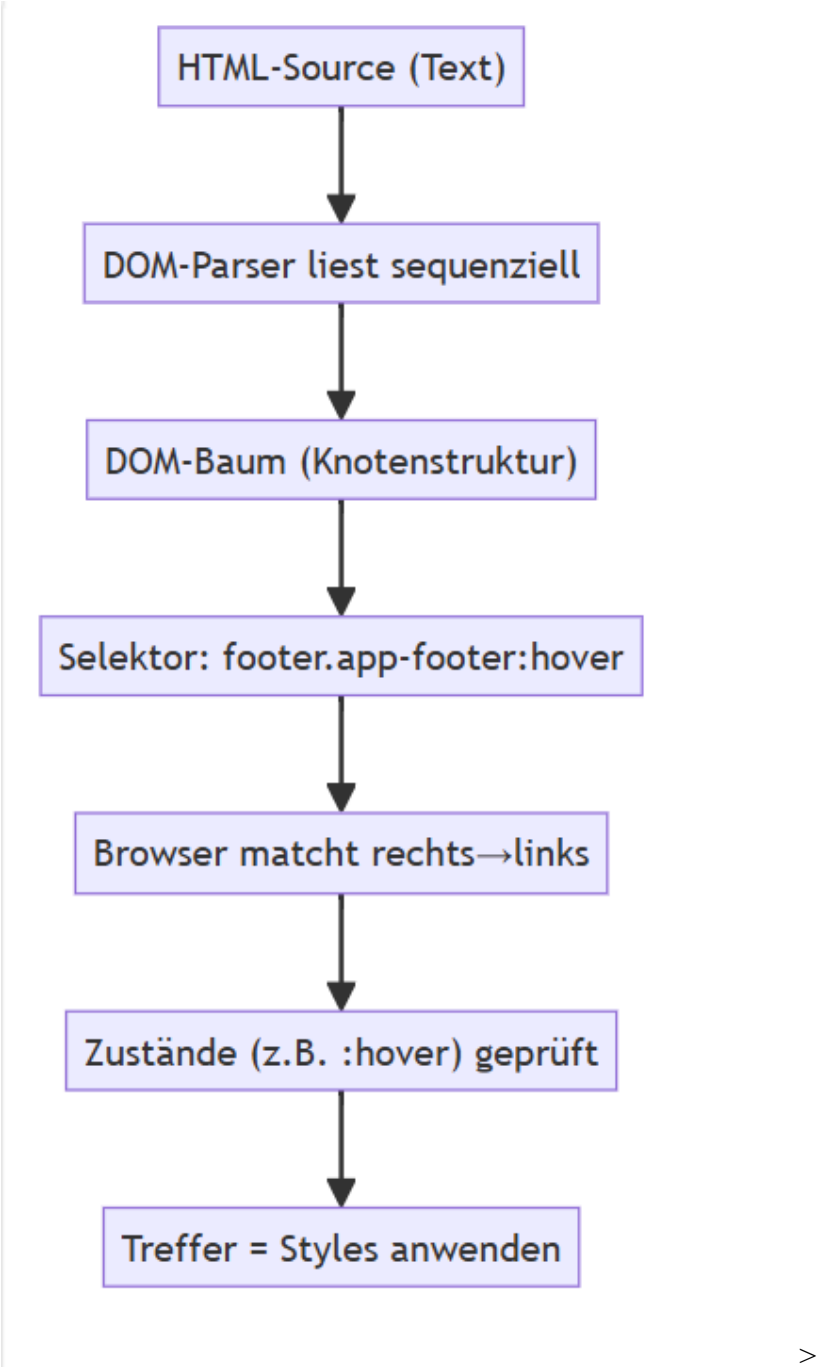
- Sachen wie `:hover`, `:focus`, `:checked` kommen **dynamisch dazu**.
- Der DOM bleibt gleich, aber der „Render Tree“ merkt: *dieses Element ist gerade hovered* → Styles anwenden.

Also kurz gesagt:

- **DOM Aufbau**: sequenziell aus dem Quelltext → ergibt Baum.
- **Selektor Matching**: nicht zeilenweise, sondern **baumorientiert und von rechts nach links**.
- **Zustände**: dynamisch ergänzt.

So läuft es ab

1. **HTML-Source**: Browser liest Text (Zeile für Zeile).
2. **DOM-Parser**: erstellt eine Baumstruktur (Nodes).
3. **DOM-Baum**: ist die Basis für CSS.
4. **Selektor Matching**: Browser schaut *von rechts nach links*.
    - `:hover` → Zustand?
    - `.app-footer` → hat Klasse?
    - `footer` → ist Typ `<footer>`?
5. **Treffer**: Styles anwenden.



>

Perfekt 🙌 – hier eine kleine **DOM-Baum-Darstellung mit CSS-Matching** am Beispiel `footer.app-footer:hover`.

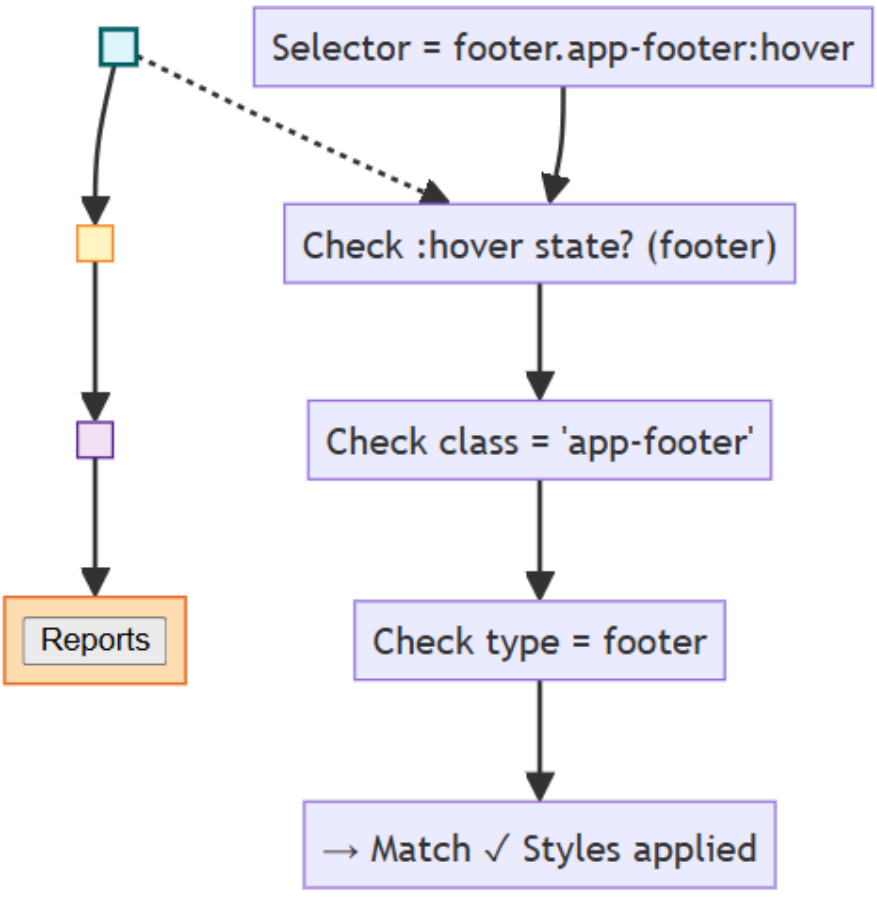
Beispiel-HTML

```
<footer class="app-footer">
  <nav class="bottom">
    <div class="tabs">
      <button class="active">Reports</button>
    </div>
  </nav>
</footer>
```

Mermaid-Diagramm (DOM-Baum + Matching)

Erklärung der Schritte

1. **DOM-Baum**: Footer → Nav → Div → Button.
2. **Selector `footer.app-footer:hover` prüft **rechts-nach-links**:
    - `:hover` → Ist das Footer-Element gerade im Hover-Zustand?
    - `.app-footer` → Hat es die Klasse `app-footer`?
    - `footer` → Ist es ein `<footer>`?
3. Wenn alles passt → **Match**, CSS-Regeln werden angewandt.



>

Sehr gut 👍 – hier die zweite Variante: diesmal der Selektor **`.tabs .active`**, also ein **Descendant-Selektor**: *ein Element mit Klasse `.active` innerhalb eines Elements mit Klasse `.tabs`*.

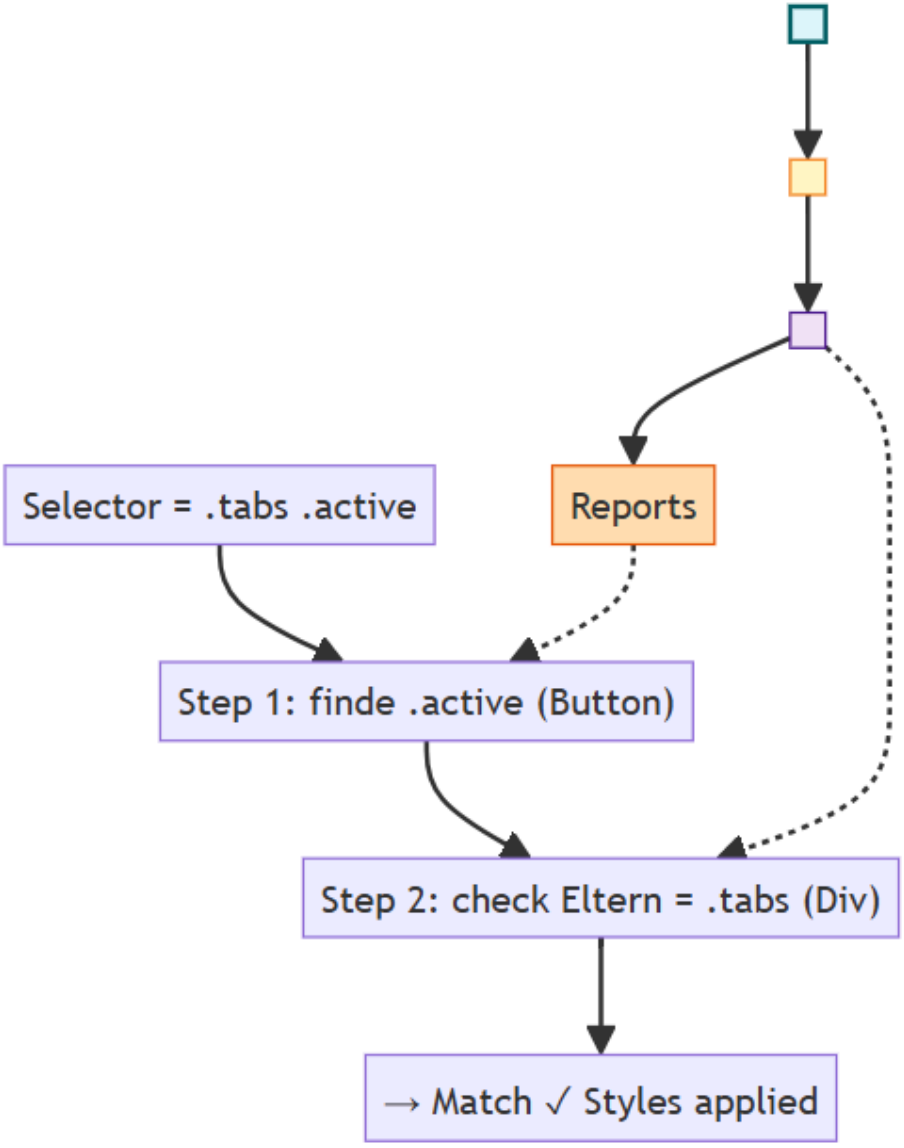
Beispiel-HTML

```
<footer class="app-footer">
  <nav class="bottom">
    <div class="tabs">
      <button class="active">Reports</button>
    </div>
  </nav>
</footer>
```

Mermaid-Diagramm (DOM-Baum + Matching `.tabs .active`)

Erklärung der Schritte

1. **Rechts-nach-links Matching**:
   - Browser sucht zuerst nach allen `.active` → findet `<button class="active">`.
   - Dann prüft er, ob dessen Eltern ein `.tabs`-Element ist → ja, `<div class="tabs">`.
2. → Der Selektor `.tabs .active` matcht.
3. Styles werden auf das **Button-Element** angewandt.



>

Sehr schön 👍 – jetzt die dritte Variante mit dem **Kind-Kombinator `>`**:

- Selektor: `.tabs > .active`
- Bedeutung: *ein Element mit Klasse `.active`, das **direkt** ein Kind von `.tabs` ist*
- Unterschied zu `.tabs .active`: dort darf `.active` auch tiefer verschachtelt sein.

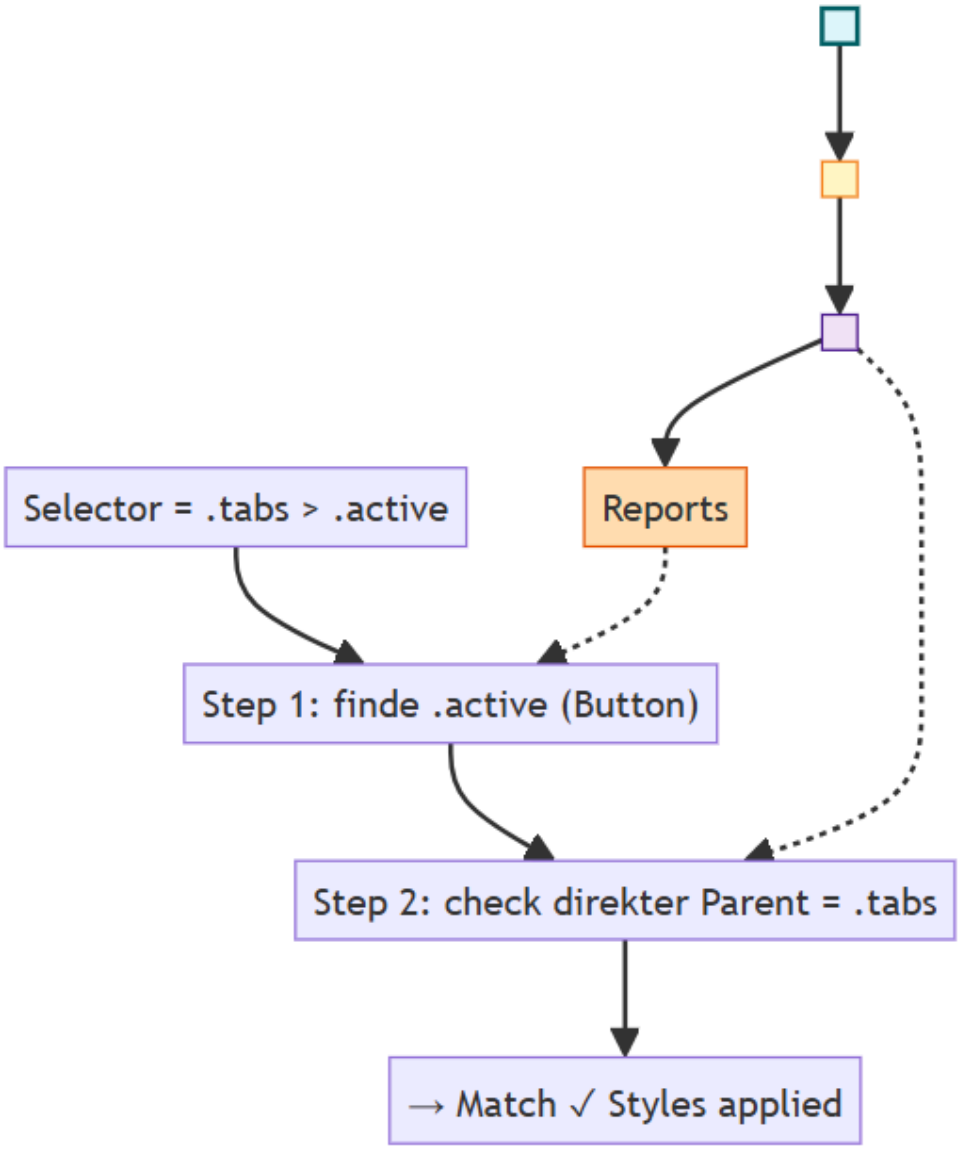
Beispiel-HTML

```
<footer class="app-footer">
  <nav class="bottom">
    <div class="tabs">
      <button class="active">Reports</button>
    </div>
  </nav>
</footer>
```

Mermaid-Diagramm (`.tabs > .active`)

Erklärung der Schritte

1. Browser sucht `.active` → findet den Button.
2. Prüft: ist der **direkte Eltern-Knoten** ein `.tabs`?
    - Ja → `<div class="tabs">` ist der direkte Parent.
3. Selektor matched → Styles werden angewandt.



>