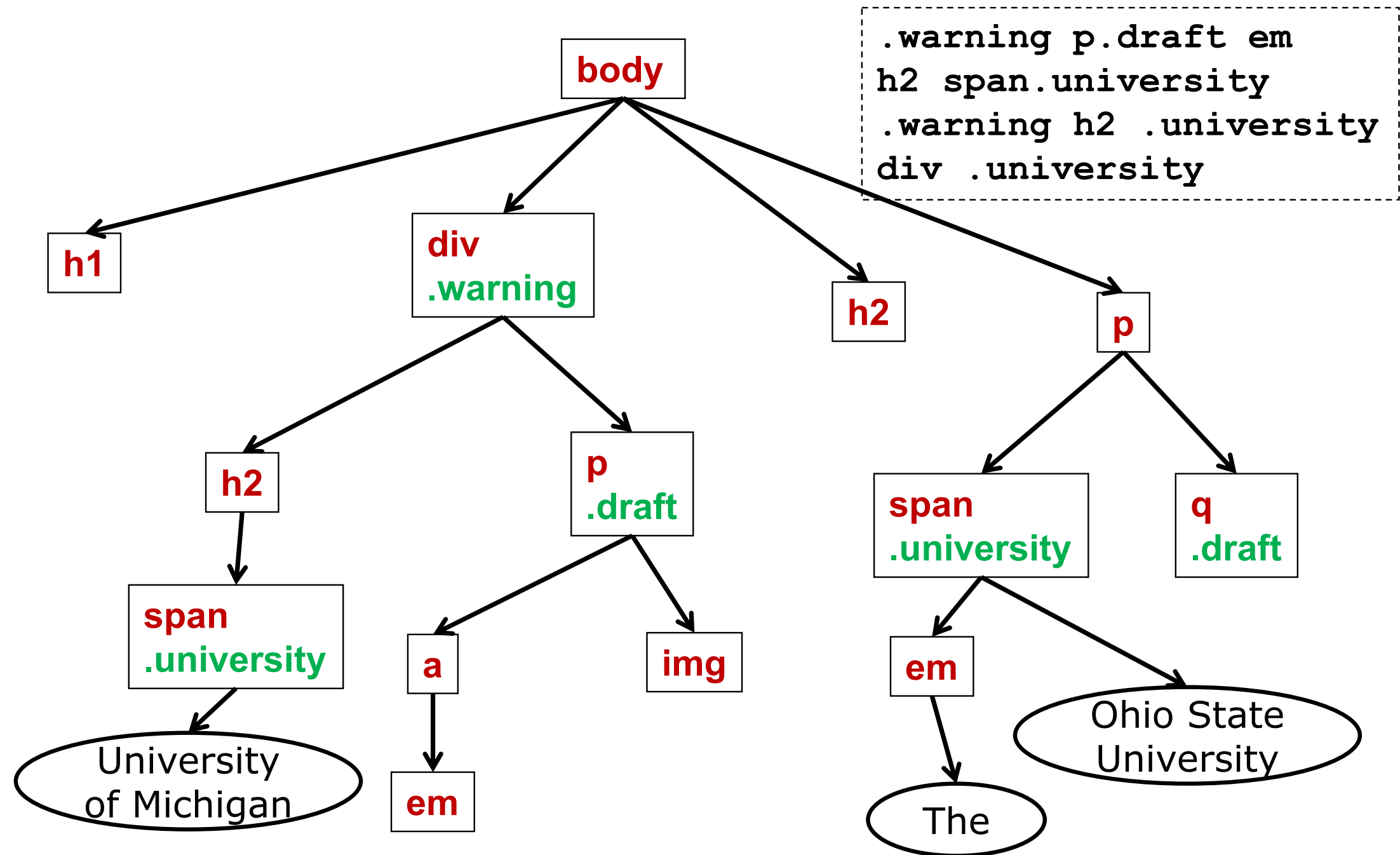


CSS: Conflict Resolution

Computer Science and Engineering ■ College of Engineering ■ The Ohio State University

Lecture 18

Recall: Example



Resolving Conflicts

- Generally, (text) styles are inherited
- Inherited styles are overridden by any selectors that match children
- But conflicts can arise: *multiple* selectors match *the same* element
 - Multiple rules with same selector

```
h1 { ... }  
h1 { ... }
```
 - One element in two different classes

```
<p class="draft warning">
```
 - Selectors with different paths match

```
.warning img  
.draft img
```
 - Different sources of css (document author, user agent)

Priority of Styling

- Rough sketch:
 - Place conflicting rules into *categories*
 - Within category, most *specific* rule wins
 - Break remaining ties with *order of declaration*
- More detail: There are 3 stages, made from 4 factors:
 1. Location and Importance
 2. Specificity
 3. Declaration order

Location (1)

- Three sources of CSS rules:
 - Author of document
 - Direct style attribute on element (ugly)
 - `<style>` in head element
 - `<link>` to CSS style sheets in header
 - User (rare)
 - Browser, aka user agent (defaults, eg blue underline)
- Priority order (high to low):
 1. Author (direct, head style, linked)
 2. User
 3. Browser

Importance (1)

- ❑ Preference given to document author
- ❑ But some users *really* need control
- ❑ Solution: **!important** modifier on style

```
h1 { font-family: arial !important; }
```
- ❑ Priority order of categories (high to low):
 1. Browser **!important**
 2. User **!important**
 3. Author **!important**
 4. Author (normal)
 5. User (normal)
 6. Browser (normal)
- ❑ Use sparingly and with caution (eg for debugging)

Specificity (2)

- Within a given category, *most specific* rule has highest priority
- Specificity of selector: a triple (x, y, z)
 - x = no. of id's
 - y = no. of classes (and pseudo-classes)
 - z = no. of elements (and pseudo-elts)
- Specificity triples are compared *lexicographically*
- Larger triple is more specific = higher priority
 - $(2, 0, 0) > (1, 4, 3)$
 - $(1, 2, 0) > (1, 1, 5)$

Source Order (3)

- ❑ Any remaining ties are broken by the order in which rules are encountered
- ❑ Later rule wins, ie later rule overrides previous one
- ❑ Example (order matters)

```
h1, h2 { padding: 25px; }  
h2 { padding-left: 10px; }
```

- ❑ Example (order matters)

```
p {  
  padding: 25px;  
  padding-left: 80px;  
}
```

Your Turn: Priority Calculation

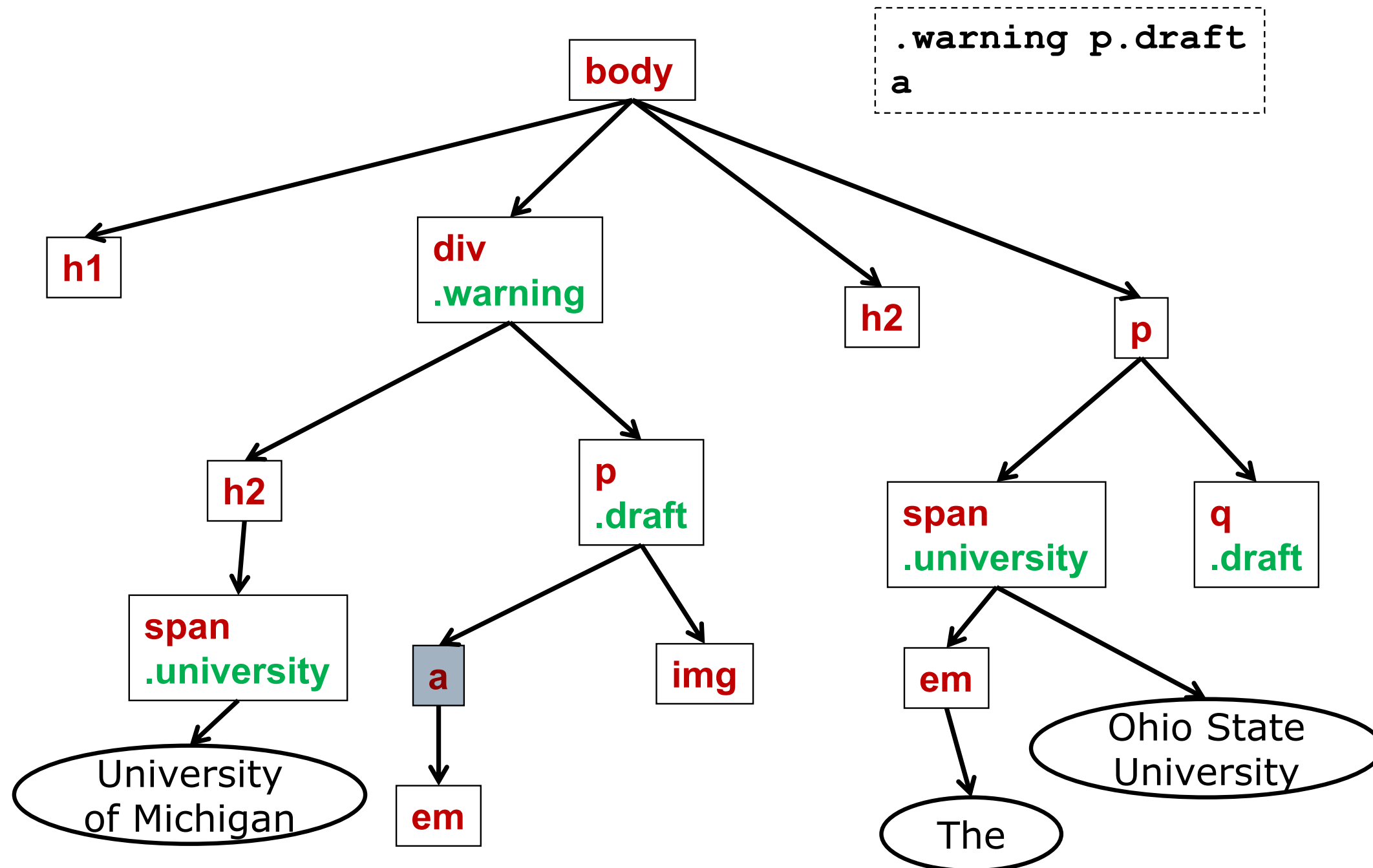
- Which rule has higher priority?

```
#main li { }  
.draft ul li { }
```

- Order the following from high to low:

```
.draft div .warning li { }  
.draft div #main li { !important; }  
.draft div #main ul li { }  
.draft .warning ul li { }
```

Problem: Any Selector Beats Inheritance



Explicit Inheritance

- Problem: How can we style <a>?
 - Children inherit color from parent (usually a good thing)
 - But browser defines default color for <a>

```
a { color: blue;
    text-decoration: underline; }
```
 - Author styling wins against a browser rule

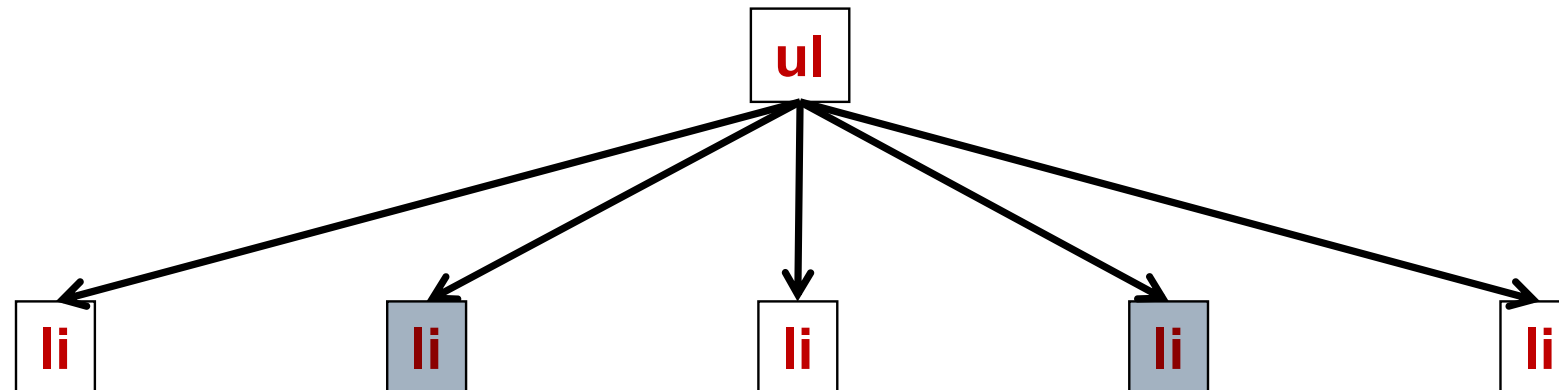
```
a { color: black; }
```
 - But what if we want the color of <a> to be the same as its parent?

```
.warning { color: darkred; }
.warning a { color: darkred; }
```
- Solution: explicit inheritance

```
a { color: inherit; }
```

Pseudo-classes

- Virtual classes
 - Implicit declaration (several standard ones exist)
 - Implicit membership (no need to set class in HTML)
 - CSS syntax: *elt:pseudo*
 - Same specificity score as (non-pseudo) class
- ```
ul li:nth-child(2n) {...}
```



# Examples: Pseudo-classes

```
a.button: hover {
 background: green;
}
```

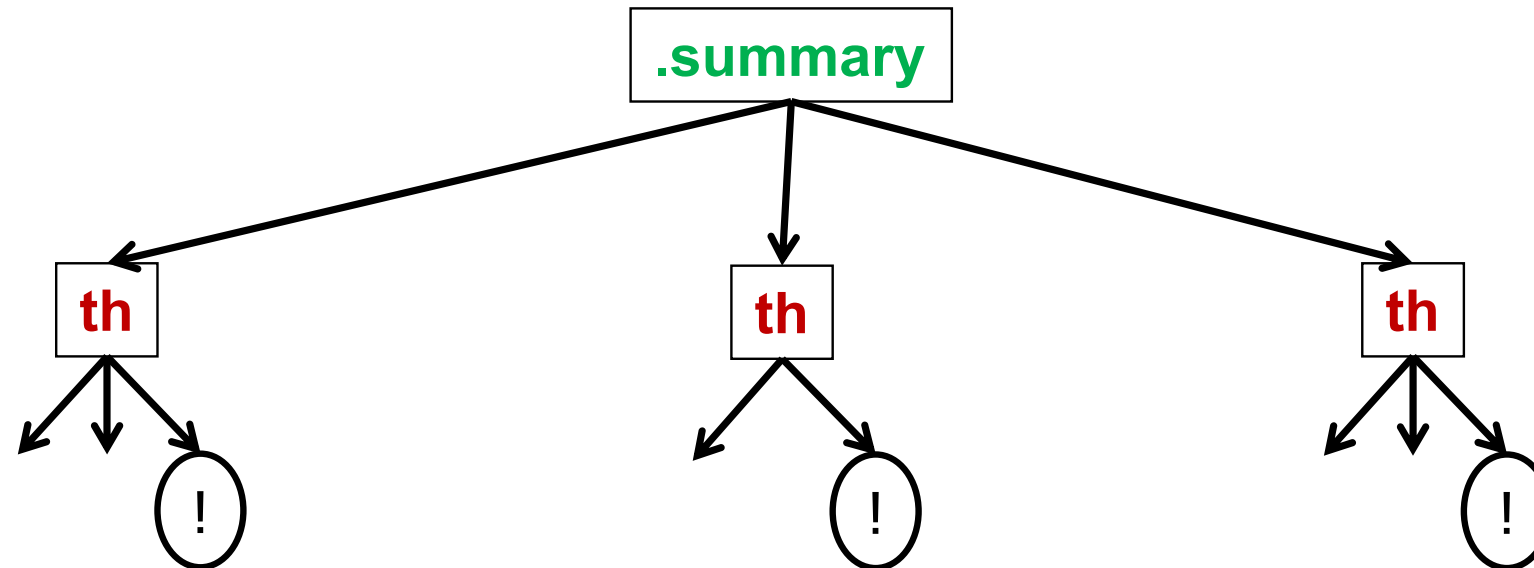
```
tbody tr: nth-of-type (odd) {
 background: #ccc;
}
```

# Some Useful Pseudo-classes

- Classic
  - `:link`, `:visited`, `:active`
  - `:hover`, `:focus`
- Structural
  - `:nth-child(An+B | even | odd)`, `:nth-of-type(An+B | even | odd)`
  - `:first-child`, `:last-child`, `:first-of-type`
  - `:only-child`, `:only-of-type`
  - `:empty`, `:root`
- State of UI elements
  - `:enabled`, `:disabled`
  - `:checked`
- Target
  - `:target` */\* elt whose id matches url fragment \*/*
- Negation
  - `:not(S)`

# Pseudo-elements

- ❑ Virtual elements
  - Implicitly exist
  - Not part of structural tree (just rendering)
- ❑ CSS syntax: *elt::pseudo*  
`.summary th::after { content: "!" ; }`



# Some Useful Pseudo-Elements

## □ Match start

- `::first-line, ::last-line`

- `::first-letter`

- `p::first-letter { font-size: 300% }`

## □ Insert content

- `::before, ::after`

- Inserted as (first/last) *child* of element

- Requires CSS content property to be set

- Beware using CSS to inject content!

# Summary: CSS Conflict Resolution

---

Computer Science and Engineering ■ The Ohio State University

- ❑ Classes and Ids
- ❑ Divs and Spans
- ❑ Selectors with ancestors, siblings
- ❑ Conflict resolution in CSS
- ❑ Pseudo-classes and pseudo-elements