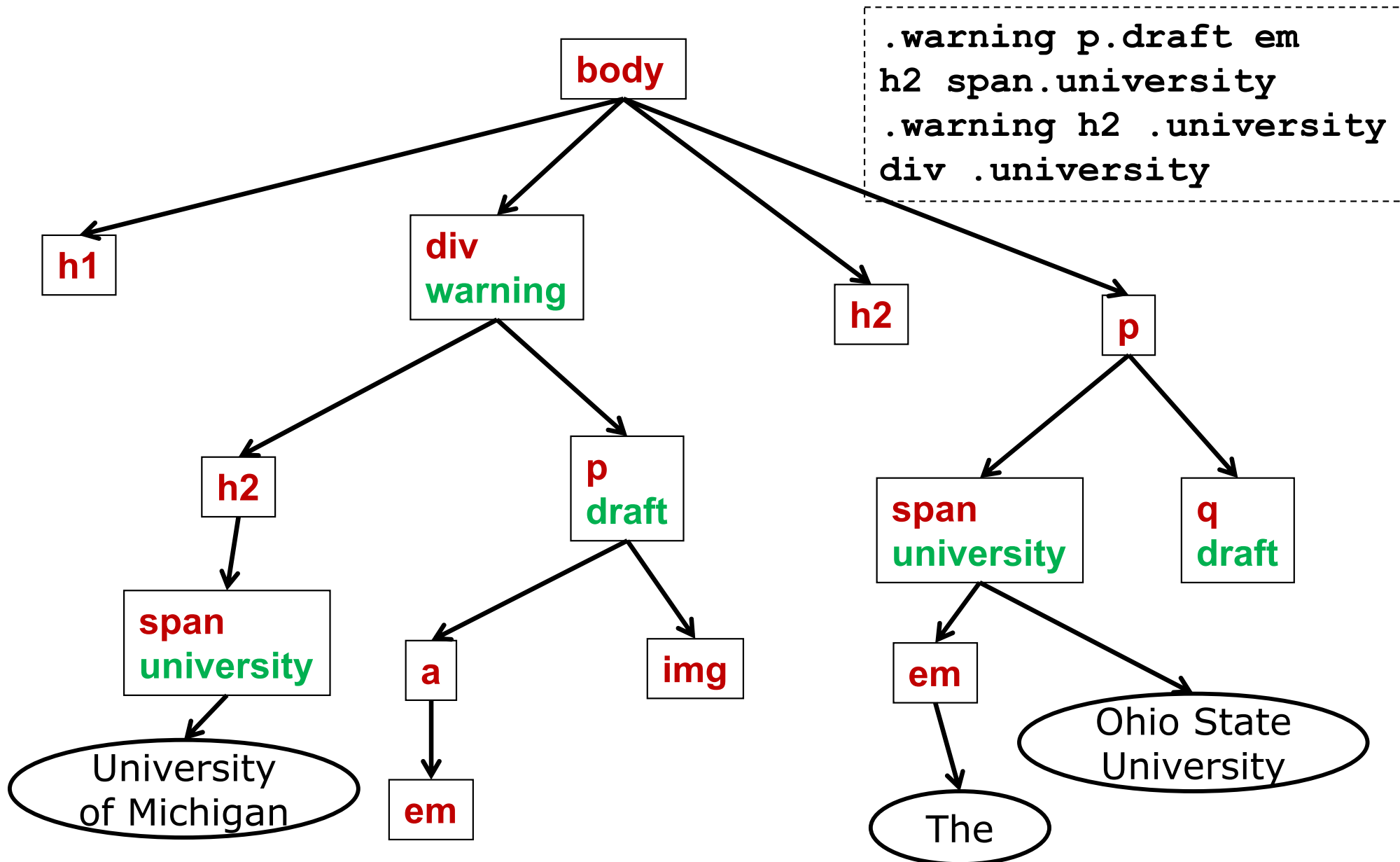


CSS Cont'd: Cascading Style Sheets

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 selectors that match children
- ❑ But conflicts can arise: *multiple* selectors match *the same* element
 - Multiple rules with same selector
 - Element part of 2 different classes
 - Two different paths (ancestors) match
 - Different sources of css (author vs user)

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

- 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 (defaults, eg blue underline)
- Priority order (high to low):
 1. Author (direct, head style, linked)
 2. User
 3. Browser

Importance

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

```
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 with caution! (eg for debugging)

Specificity

- 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)
- Compare specificity *lexicographically*
- More specific is larger = higher priority
 - $(2, 0, 0) > (1, 4, 3)$
 - $(1, 2, 0) > (1, 1, 5)$

Source Order

- ❑ Remaining ties broken by the order in which rules are encountered
- ❑ 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

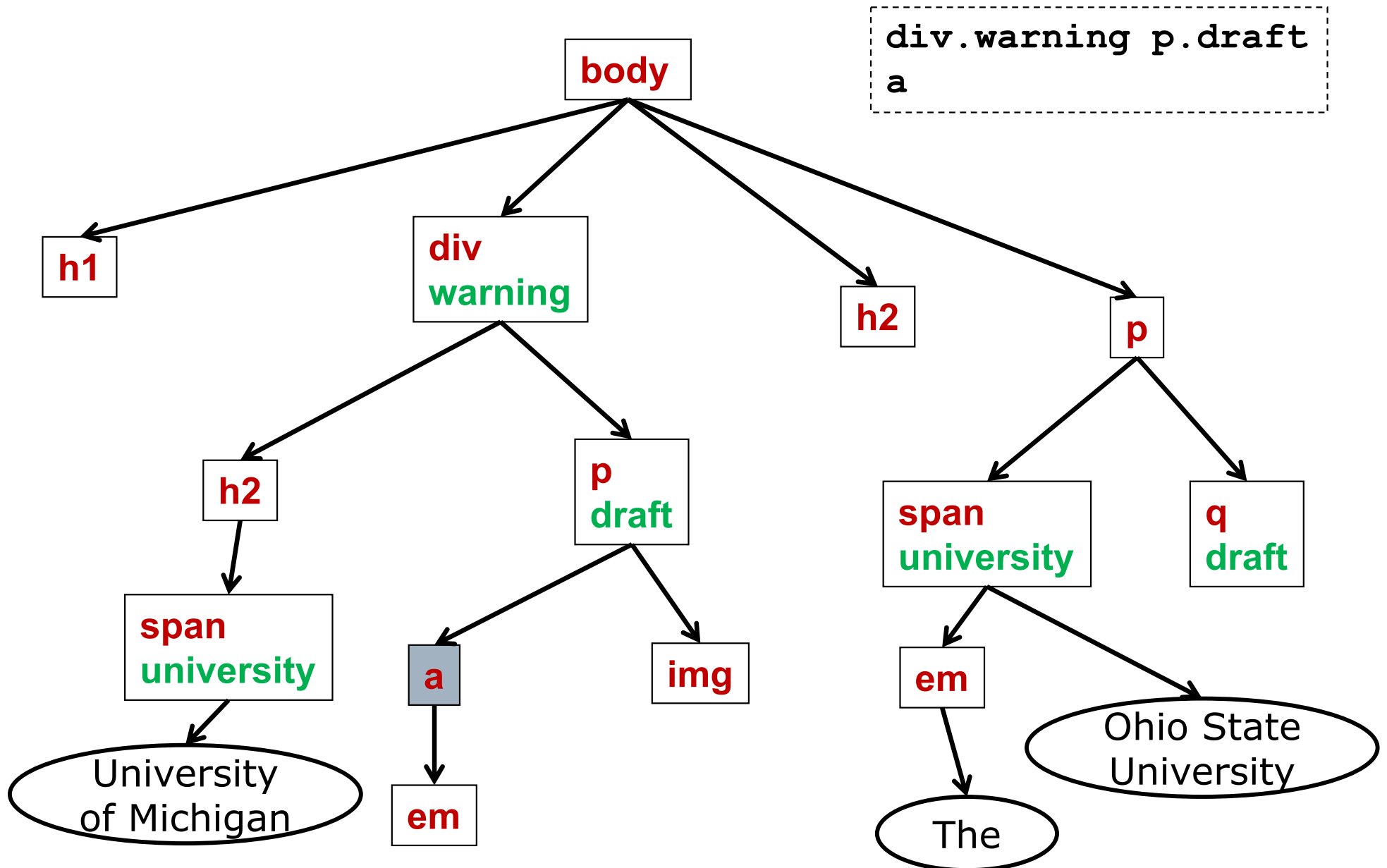
- Which rule has higher priority?

```
#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: Selectors Beat Inherit.



Explicit Inheritance

- Problem: How to style `<a>`?
 - Children inherit color from parent (good)
 - But browser defines default color for `<a>`

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

```
a { color: black; }
```
 - But I want the color dictated by styling of *parent* of `<a>`

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

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

Pseudo-classes

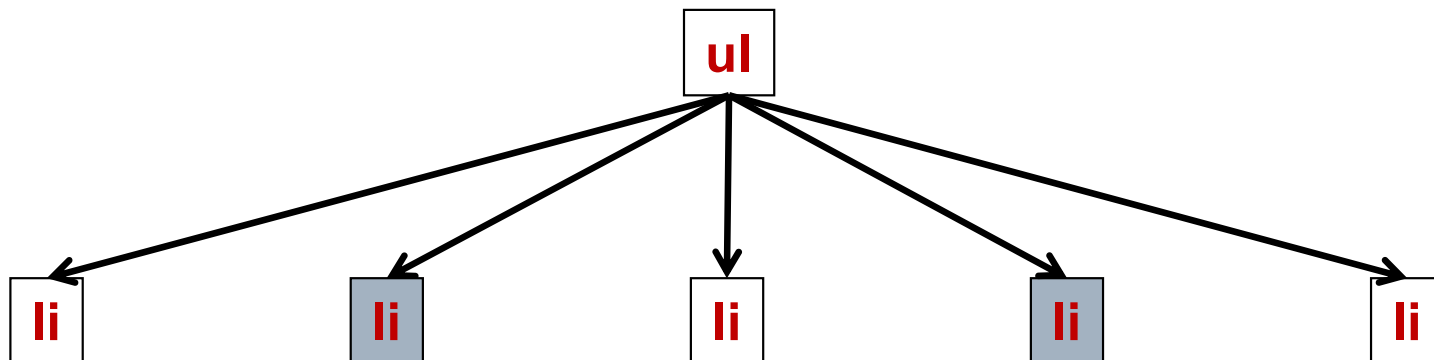
□ Virtual classes

- Implicitly declared (a few standard ones)
- Implicit membership (no class attribute)

□ CSS syntax: *elt:pseudo*

- Same specificity as (non-pseudo) class

```
ul li:nth-child(2n) {...}
```



Some Useful Pseudo-classes

- ❑ Classic
 - `:link, :visited, :active`
 - `:hover, :focus`
- ❑ Structural
 - `:nth-child(An+B), :nth-of-type(An+B)`
 - `: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)`

Examples

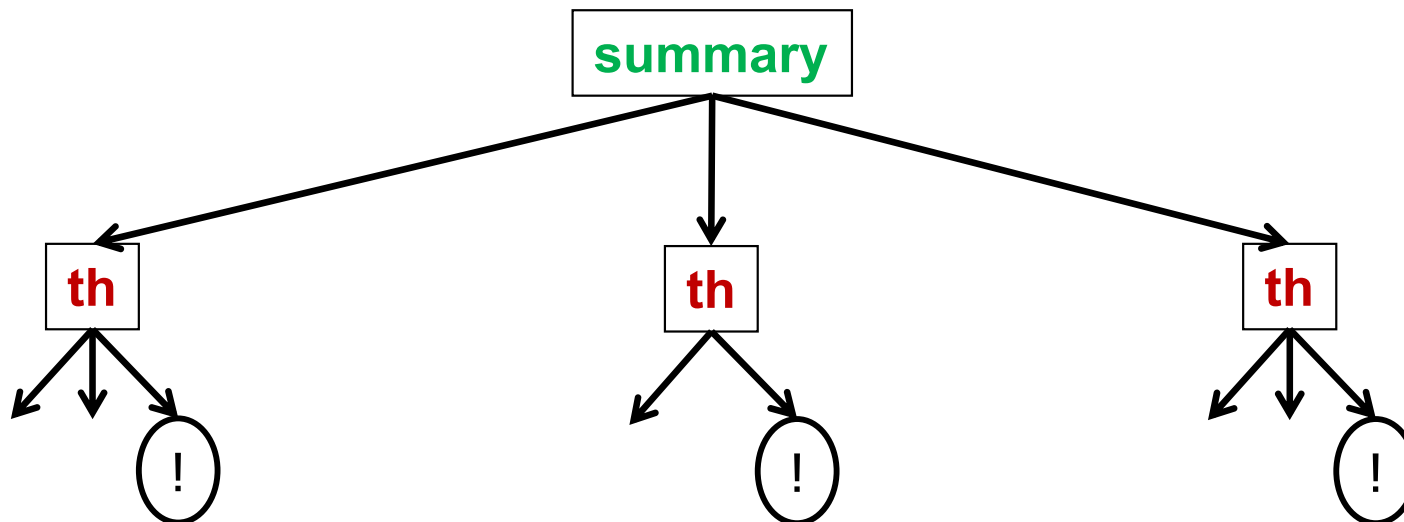
```
a.button:hover {  
    background: green;  
}
```

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

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`

□ Insert content

- `::before`, `::after`
- Inserted as (first/last) *child* of element
- Requires content property
- Beware using CSS to inject content!

Summary

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