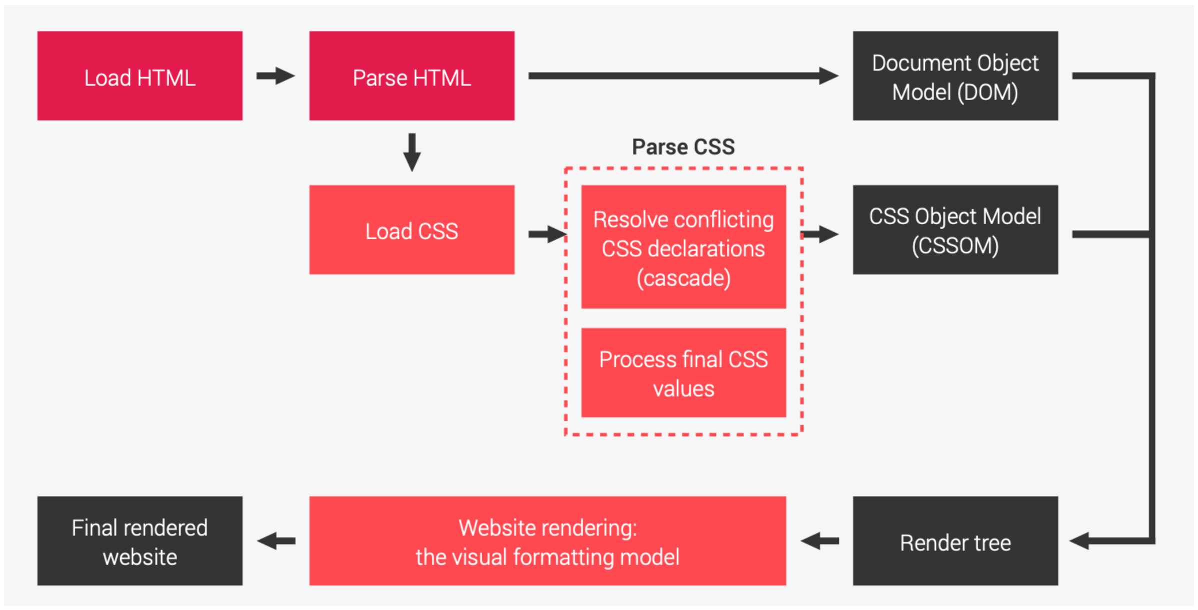
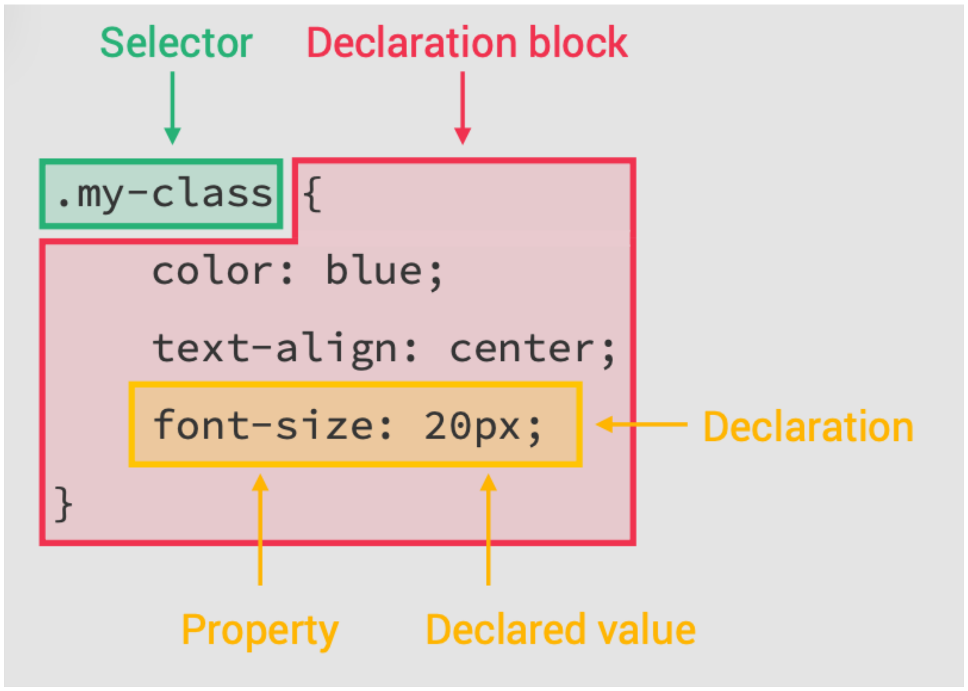
How CSS works behind the scene

1. Overview

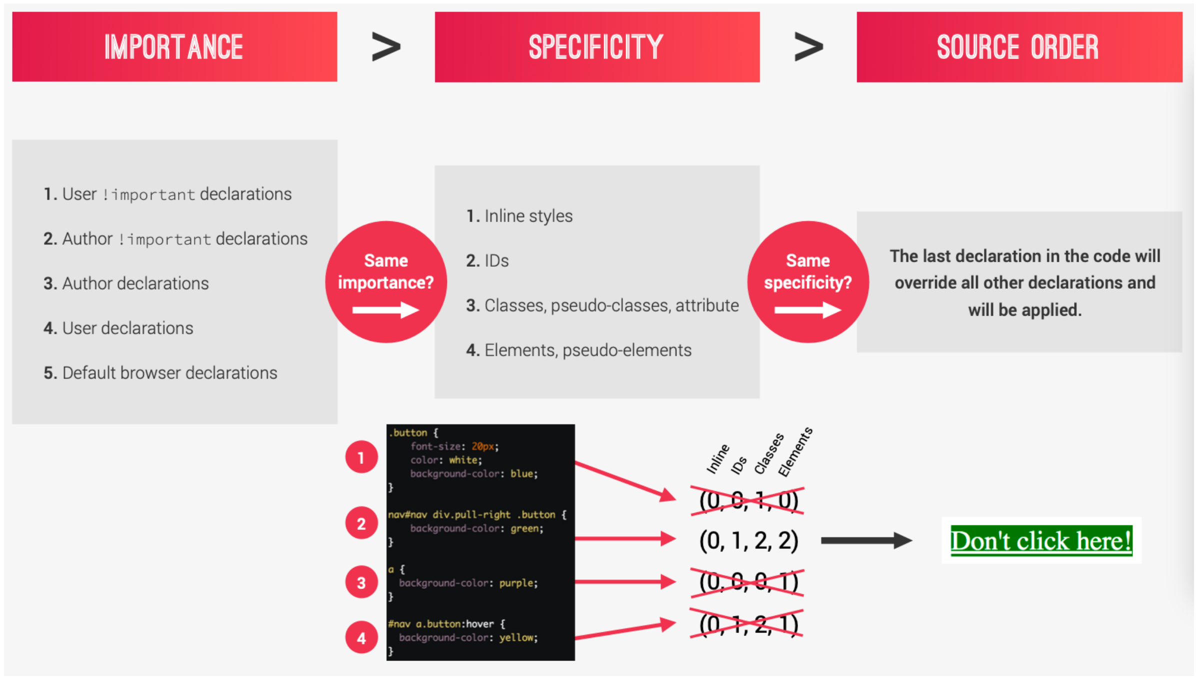

Load HTML
Parse HTML
Document Object Model (DOM)
Parse CSS
Load CSS
Resolve conflicting CSS declarations (cascade)
Process final CSS values
CSS Object Model (CSSOM)
Final rendered website
Website rendering: the visual formatting model
Render tree

2. CSS Terminology (CSS rules)



```
.my-class {
    color: blue;
    text-align: center;
    font-size: 20px;
}
```

3. CSS Parsing: the Cascade (The ‘C’ in CSS)
    a. Process of `combining differenct stylesheets` and `resolving conflicts between different CSS rules and declarations`, when more than one rule applices to a certain element.



1. User `!important` declarations
2. Author `!important` declarations
3. Author declarations
4. User declarations
5. Default browser declarations

1. Inline styles
2. IDs
3. Classes, pseudo-classes, attribute
4. Elements, pseudo-elements

The last declaration in the code will override all other declarations and will be applied.

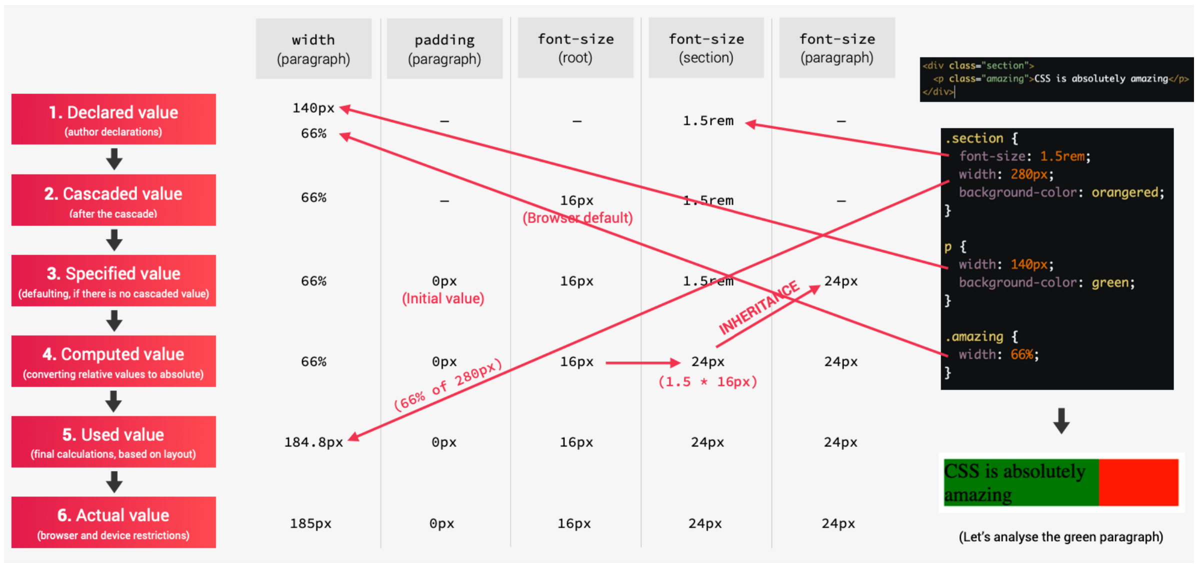
```
.button {
    font-size: 20px;
    color: white;
    background-color: blue;
}

nav#nav div.pull-right .button {
    background-color: green;
}

a {
  background-color: purple;
}

#nav a.button:hover {
  background-color: yellow;
}
```

1 ~~(0, 0, 1, 0)~~
2 (0, 1, 2, 2)
3 ~~(0, 0, 0, 1)~~
4 ~~(0, 1, 2, 1)~~

- Summary

- CSS declarations marked with !important have the highest priority.
- But only use !important as a last resource. It’s better to use correct specificities - more maintainable code!
- Inline styles will always have priority over styles in external stylesheets
- A selector that contains 1 ID is more specific than one with 1000 classes
- A selector that contains 1 class is more specific than one with 1000 elements
- The universal selector * has no specificity value (0,0,0,0)
- **Rely more on specificity** than on the order of selectors
- But rely on order when using 3rd-party stylesheets - **always put your author stylesheet last**

4. CSS Parsing: Value processing



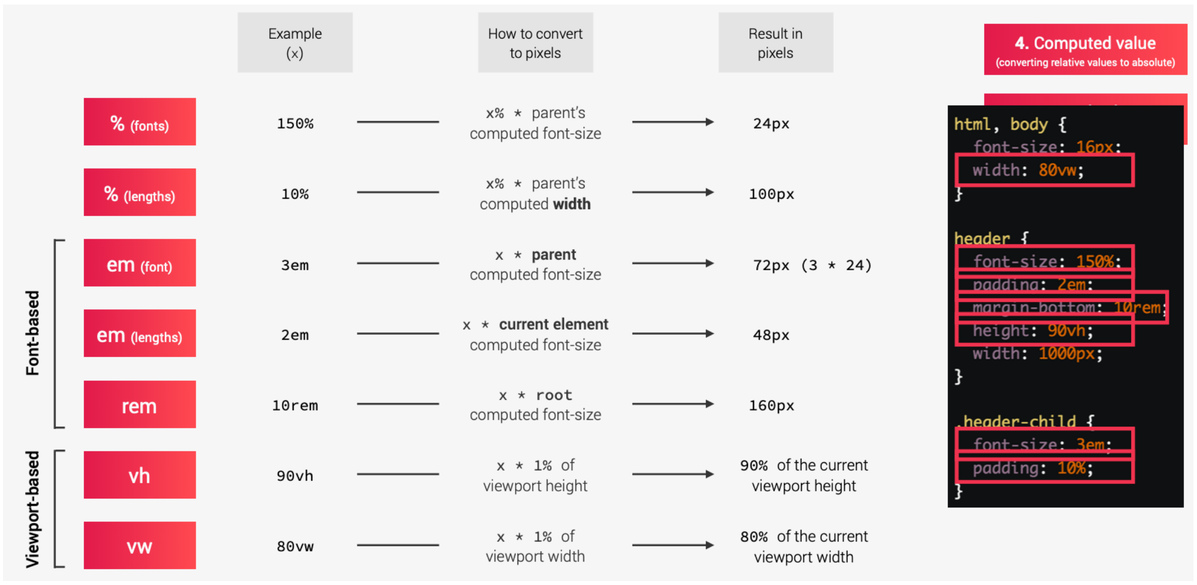
	`width` (paragraph)	`padding` (paragraph)	`font-size` (root)	`font-size` (section)	`font-size` (paragraph)
1. Declared value (author declarations)	140px 66%	–	–	1.5rem	–
2. Cascaded value (after the cascade)	66%	–	16px (Browser default)	1.5rem	–
3. Specified value (defaulting, if there is no cascaded value)	66%	0px (Initial value)	16px	1.5rem	24px
4. Computed value (converting relative values to absolute)	66%	0px	16px	24px (1.5 * 16px)	24px
5. Used value (final calculations, based on layout)	184.8px	0px	16px	24px	24px
6. Actual value (browser and device restrictions)	185px	0px	16px	24px	24px

```html
<div class="section">
  <p class="amazing">CSS is absolutely amazing</p>
</div>
```

```css
.section {
  font-size: 1.5rem;
  width: 280px;
  background-color: orangered;
}

p {
  width: 140px;
  background-color: green;
}

.amazing {
  width: 66%;
}
```

(Let’s analyse the green paragraph)

5. CSS Parsing: Value processing (unit conversion)

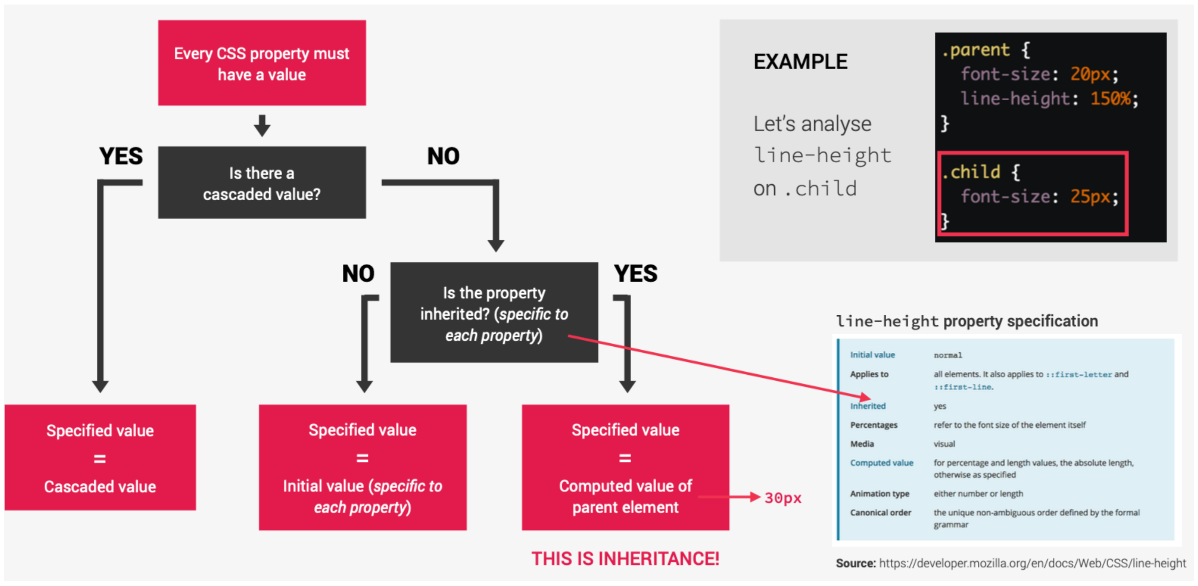

4. Computed value
(converting relative values to absolute)

		Example (x)	How to convert to pixels	Result in pixels
	% (fonts)	`150%`	`x%` * parent's computed font-size	`24px`
	% (lengths)	`10%`	`x%` * parent's computed **width**	`100px`
Font-based	em (font)	`3em`	`x` * **parent** computed font-size	`72px (3 * 24)`
Font-based	em (lengths)	`2em`	`x` * **current element** computed font-size	`48px`
Font-based	rem	`10rem`	`x` * `root` computed font-size	`160px`
Viewport-based	vh	`90vh`	`x` * `1%` of viewport height	`90%` of the current viewport height
Viewport-based	vw	`80vw`	`x` * `1%` of viewport width	`80%` of the current viewport width

```css
html, body {
  font-size: 16px;
  width: 80vw;
}

header {
  font-size: 150%;
  padding: 2em;
  margin-bottom: 10rem;
  height: 90vh;
  width: 1000px;
}

.header-child {
  font-size: 3em;
  padding: 10%;
}
```

- Summary
  - Each property has an initial value, used if nothing is declared (and if there is no inheritance — see next lecture)
  - Browsers specify a **root font-size** for each page (usually 16px)
  - Percentages and relative values are always converted to pixels
  - Percentages are measured relative to their parent's **font-size**, if used to specify font-size
  - Percentages are measured relative to their parent's **width**, if used to specify lengths
  - em are measured relative to their **parent** font-size, if used to specify font-size
  - em are measured relative to the **current** font-size, if used to specify lengths
  - rem are always measured relative to the **document's root** font-size
  - vh and vw are simply percentage measurements of the viewport's height and width.

6. CSS Parsing: Inheritance


Every CSS property must have a value
YES
Is there a cascaded value?
NO
NO
Is the property inherited? (specific to each property)
YES
Specified value = Cascaded value
Specified value = Initial value (specific to each property)
Specified value = Computed value of parent element
30px
THIS IS INHERITANCE!
EXAMPLE
Let's analyse line-height on .child

```
.parent {
  font-size: 20px;
  line-height: 150%;
}

.child {
  font-size: 25px;
}
```

`line-height` property specification

Initial value	`normal`
Applies to	all elements. It also applies to `::first-letter` and `::first-line`.
Inherited	yes
Percentages	refer to the font size of the element itself
Media	visual
Computed value	for percentage and length values, the absolute length, otherwise as specified
Animation type	either number or length
Canonical order	the unique non-ambiguous order defined by the formal grammar

Source: https://developer.mozilla.org/en/docs/Web/CSS/line-height

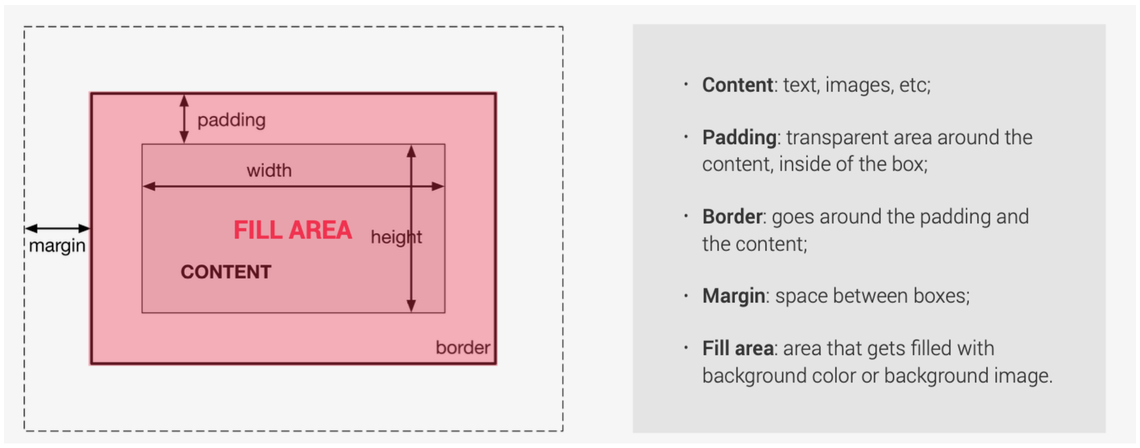
- Summary
    - Inheritance passes the values for some specific properties from parents to children — **more maintainable code**
    - Properties related to text are inherited: font-family, font-size, color, etc
    - **The computed value** of a property is what gets inherited, **not** the declared value.
    - Inheritance of a property only works if no one declares a value for that property
    - The **inherit keyword forces inheritance** on a certain property
    - The initial keyword resets a property to its initial value.

7. Website rendering
    a. Algorithm that calculates boxes and determines the layout of theses boxes, for each element in the render tree, in order to determine the final layout of the page
        i. Dimensions of boxes: the box model
        ii. Box type: inline, bloc, inline-block
        iii. Positioning Scheme: floats and positioning
        iv. Stacking contexts

v. Other elements

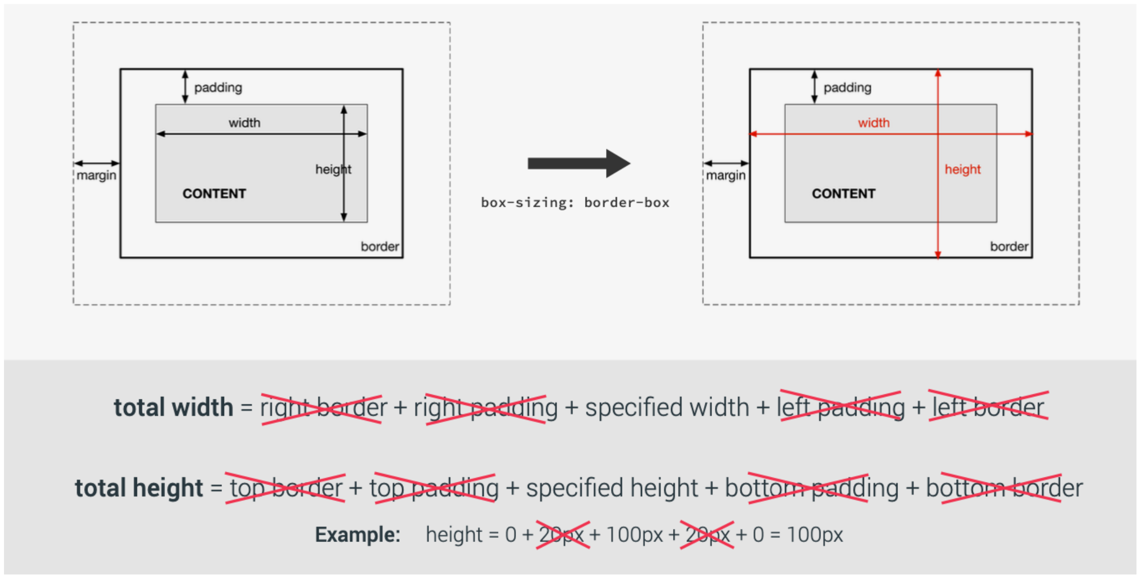
vi. Viewport size, dimensions of images, …

b. The box model



- **Content**: text, images, etc;
- **Padding**: transparent area around the content, inside of the box;
- **Border**: goes around the padding and the content;
- **Margin**: space between boxes;
- **Fill area**: area that gets filled with background color or background image.

1. * total width = right border + right padding + specified width + left padding + left border
   * total height = top border + top padding + specified height + bottom padding + bottom border

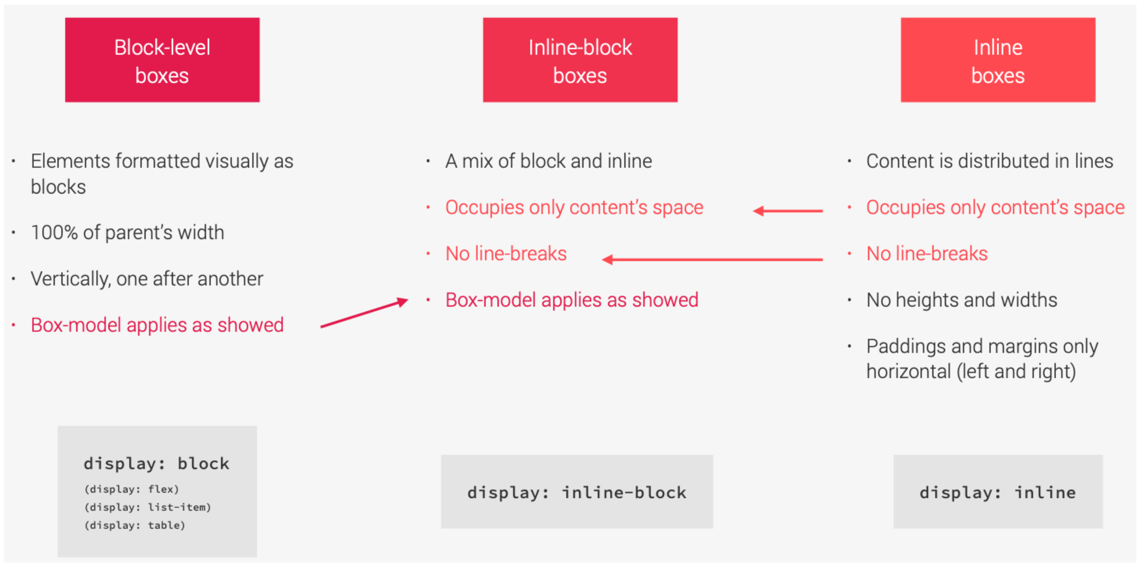


total width = ~~right border~~ + ~~right padding~~ + specified width + ~~left padding~~ + ~~left border~~

total height = ~~top border~~ + ~~top padding~~ + specified height + ~~bottom padding~~ + ~~bottom border~~

Example: height = 0 + ~~20px~~ + 100px + ~~20px~~ + 0 = 100px

c. The Box types


Block-level boxes

- Elements formatted visually as blocks
- 100% of parent's width
- Vertically, one after another
- Box-model applies as showed

```
display: block
(display: flex)
(display: list-item)
(display: table)
```

Inline-block boxes

- A mix of block and inline
- Occupies only content's space
- No line-breaks
- Box-model applies as showed

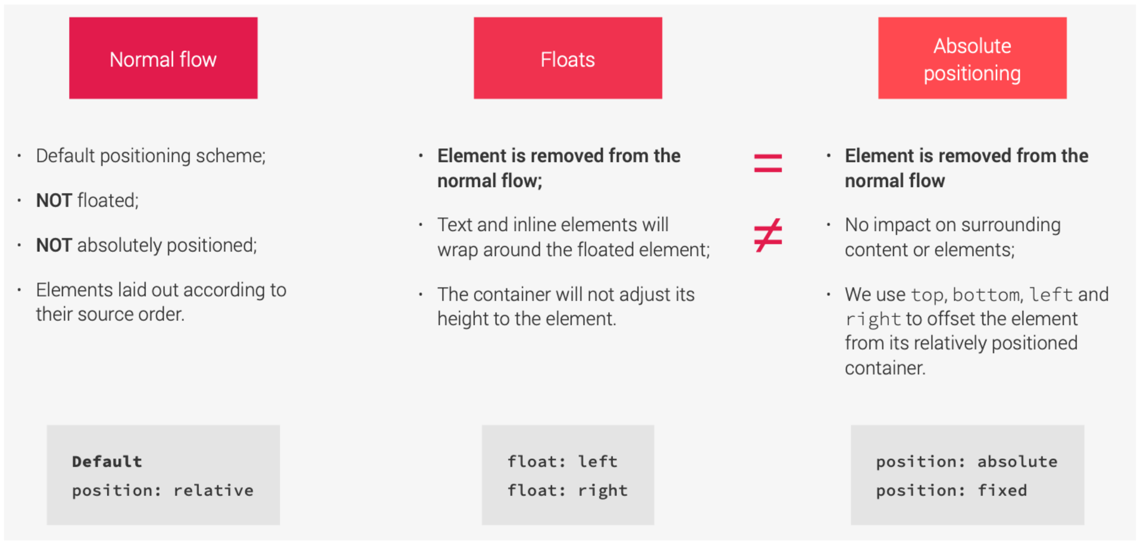
```
display: inline-block
```

Inline boxes

- Content is distributed in lines
- Occupies only content's space
- No line-breaks
- No heights and widths
- Paddings and margins only horizontal (left and right)

```
display: inline
```

d. Positioning schemes


Normal flow

- Default positioning scheme;
- **NOT** floated;
- **NOT** absolutely positioned;
- Elements laid out according to their source order.

```
Default
position: relative
```

Floats

- **Element is removed from the normal flow;**
- Text and inline elements will wrap around the floated element;
- The container will not adjust its height to the element.

```
float: left
float: right
```

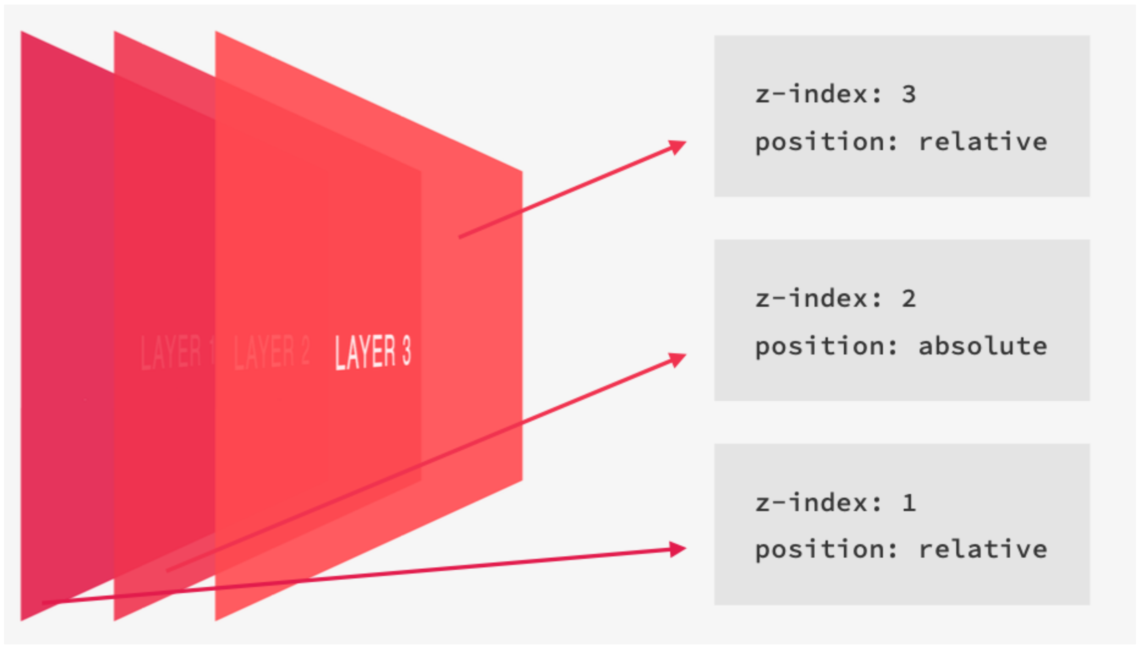
$=$ $\neq$

Absolute positioning

- **Element is removed from the normal flow**
- No impact on surrounding content or elements;
- We use `top`, `bottom`, `left` and `right` to offset the element from its relatively positioned container.

```
position: absolute
position: fixed
```

e. Stacking contexts (z-indexes)



8. CSS Architecture, Components and BEM
    a. Think : about the layout of your webpage or web app before writing code
    b. Build : your layout in HTML and CSS with a consistent structure for naming classes
    c. Architect : create a logical architecture for your CSS with files and folders
    d. Think: Component-driven design
        i. Modular building blocks that make up interfaces
        ii. Held together by the layout of the page
        iii. Re-usable across a project, and between different projects
        iv. Independent, allowing us to use them anywhere on the page
        - Atomic Design
            - Atoms
            - Molecules
            - Orgnisms (component)
            - Templates
            - Pages

e. Build: BEM

    i. Block Element Modifier

    ii. Block : standalone component that is meaningful on its own

    iii. Element : part of a block that has no standalone meaning

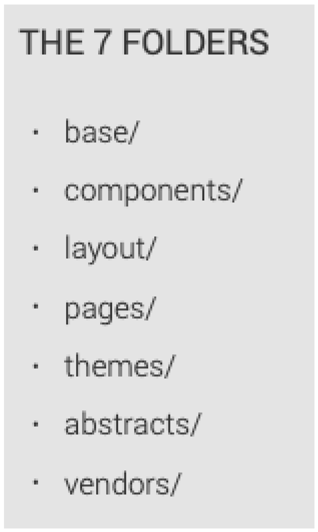
    iv. Modifier : a different version of a block or an element

    .block { } / .block__element{ } / .block__element—modifier { }

f. Architect

    i. The 7-1 Pattern

        7 different folders for patial Sass files, and 1 main Sass file to import all other files into a compiled CSS stylesheet



THE 7 FOLDERS

- base/
- components/
- layout/
- pages/
- themes/
- abstracts/
- vendors/