CSS

Basic CSS Mechanisms

cascade, inheritance, block elements

doc. Ing. Radek Burget, Ph.D.

burgetr@vut.cz

Ing. Jiří Hynek, Ph.D.

hynek@vut.cz

Style Definition Process

- Basic style of elements (headings, paragraphs, tables, …)
  - Built-in user agent style
  - Author style sheet based on the graphical design
- Specification of details and specific cases
  - Layout of particular elements (header, footer, …)
  - Special cases (of tables, headings, …)
- We use the CSS mechanisms
  - Cascade of rules
  - Inheritance

Rule Application

- Multiple selectors usually apply to a single element

```
<p class="intro">Following text is
    <span class="imp">important</span></p>
```

```
p.intro span { color: red; }
p span { color: blue; font-weight: bold; }
```

- What color will be used to display the word “important”?

Side note: The `!important` directive (**very specific use**)

```
p.intro span { color: red; }
p span { color: blue !important; font-weight: bold; }
```

Cascade of Rules

- For every HTML element, the browser takes **all CSS rules** whose selector matches the element.
  - Each rule defines the values of some CSS properties.
- The rules are ordered to a cascade.
- The cascaded values are used for computing the final visual properties of the element.

Ordering criteria:

1. Rule origin
   - Author rules override the user agent styles.
2. **Rule selector specificity**
   - The more specific is the selector, the more important is the value
3. Order of specification
   - If the rule has **equal origin and specificity**, the latest specified value is used.

Rule specificity

- Specificity is a number `abcd`, where
  - `a=1` in case of inline style, `a=0` otherwise
  - `b` is the number of `id` attributes in selector
  - `c` is the number of classes in selector
  - `d` is the number of element names in selector
- Shortly:
  - `id` > `class` > `element`

```
#main p.intro span { … } /* 0 - 1 - 1 - 2 ~ 112 */
p.intro span { … }       /* 0 - 0 - 1 - 2 ~ 12 */
p span { … }             /* 0 - 0 - 0 - 2 ~ 2 */
```

Back to the Question

- What color will be used to display the word “important”?

```
<p class="intro">Following text is
    <span class="imp">important</span></p>
```

```
p.intro span { color: red; }
p span { color: blue; font-weight: bold; }
```

- More specific rule wins => ?

Back to the question (II)

- What color will be used to display the word “important”?

```
<p class="concl">Following text is
    <span class="imp">important</span></p>
```

```
p.intro span { color: red; }
p span { color: blue; font-weight: bold; }
```

- Only one candidate => ?

Back to the question (III)

- What color will be used to display the word “important”?

```
<p class="intro concl">Following text is
    <span class="imp">important</span></p>
```

```
p.intro span { color: blue; font-weight: bold; }
p.concl span { color: red; }
```

- Definition order decides => ?

Property inheritance

- Some property values are inherited by the child elements
  - When not explicitly specified

```
<p class="intro concl">Following text is
    <span class="imp">important</span></p>
```

```
p { color: red; border: 1px solid blue; }
```

- The text color applies to the child element too
- Only the parent element `<p>` has the border
- The CSS specification defines which properties are inherited

Typical Usage of Inheritance

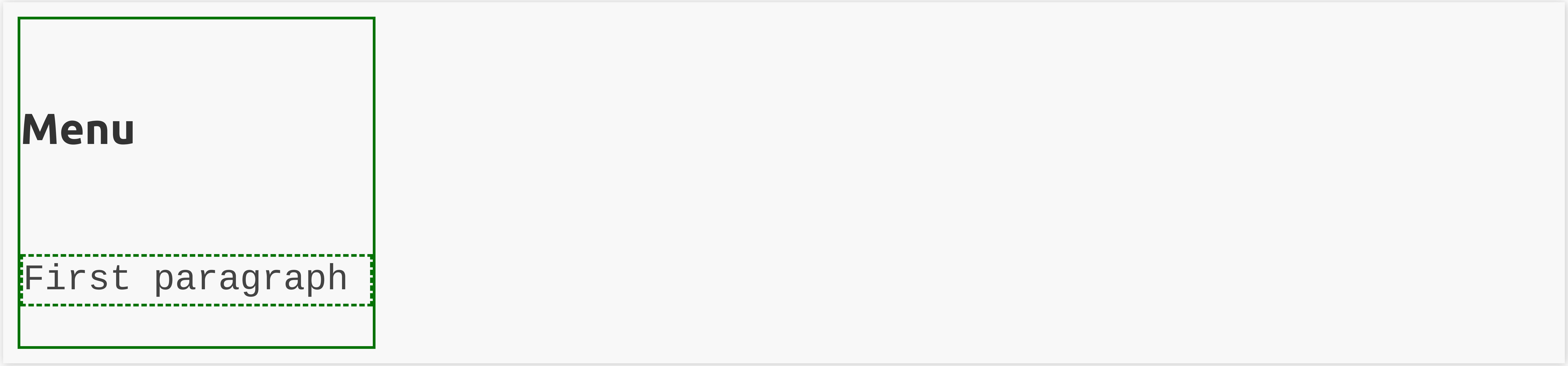
```
body {
    color: white;
    background-color: black;
}

h1 {
    color: red;
}
```

- All the text will be white
  - All the elements inherit their color from the `body` element
- Only the headings (and their descendant elements) will be red
  - We define red color for `h1`

The `inherit` value

- Any property can have a special `inherit` value
- The property value is then always inherited from the parent element

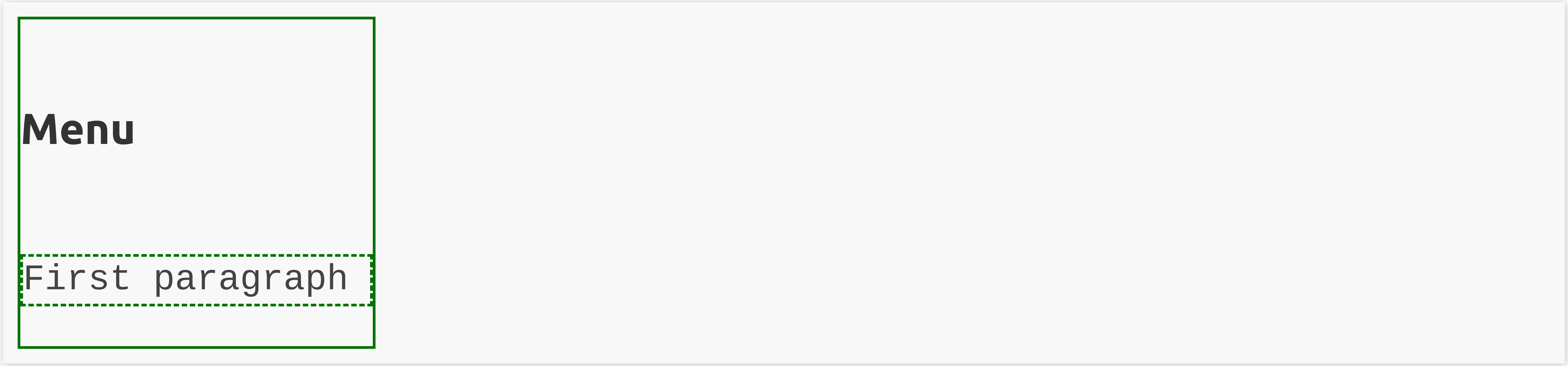
Example



```
<div id="menu">
    <h1>Menu</h1>
    <p>First paragraph</p>
    <p>Second paragraph</p>
</div>
```

```
#menu { border: 3px #057205ff solid; }
```

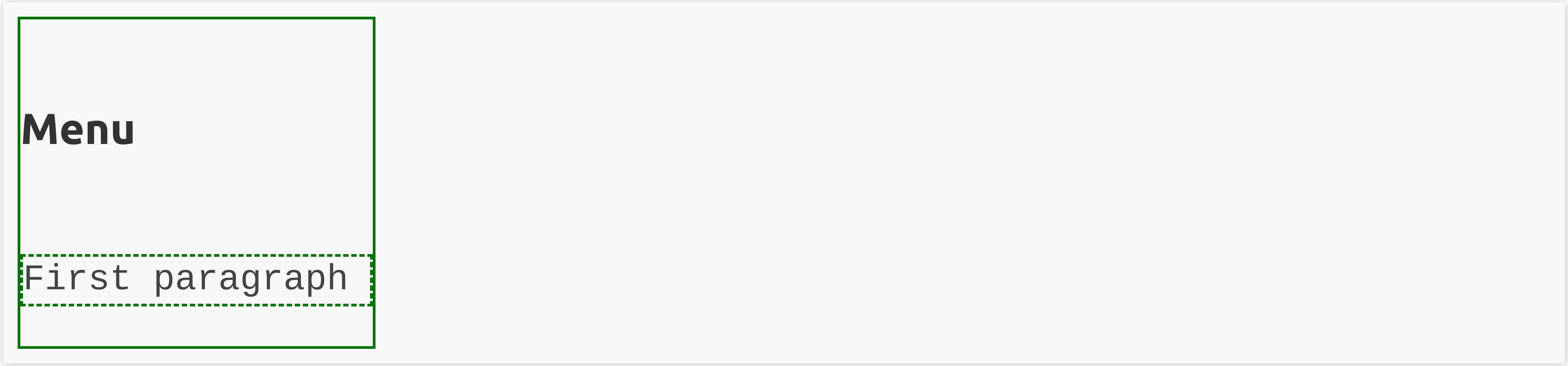
Example (II)



```
<div id="menu">
    <h1>Menu</h1>
    <p>First paragraph</p>
    <p>Second paragraph</p>
</div>
```

```
#menu { border: 3px #057205ff solid; }
#menu p { border: inherit; }
```

Example



```
<div id="menu">
    <h1>Menu</h1>
    <p>First paragraph</p>
    <p>Second paragraph</p>
</div>
```

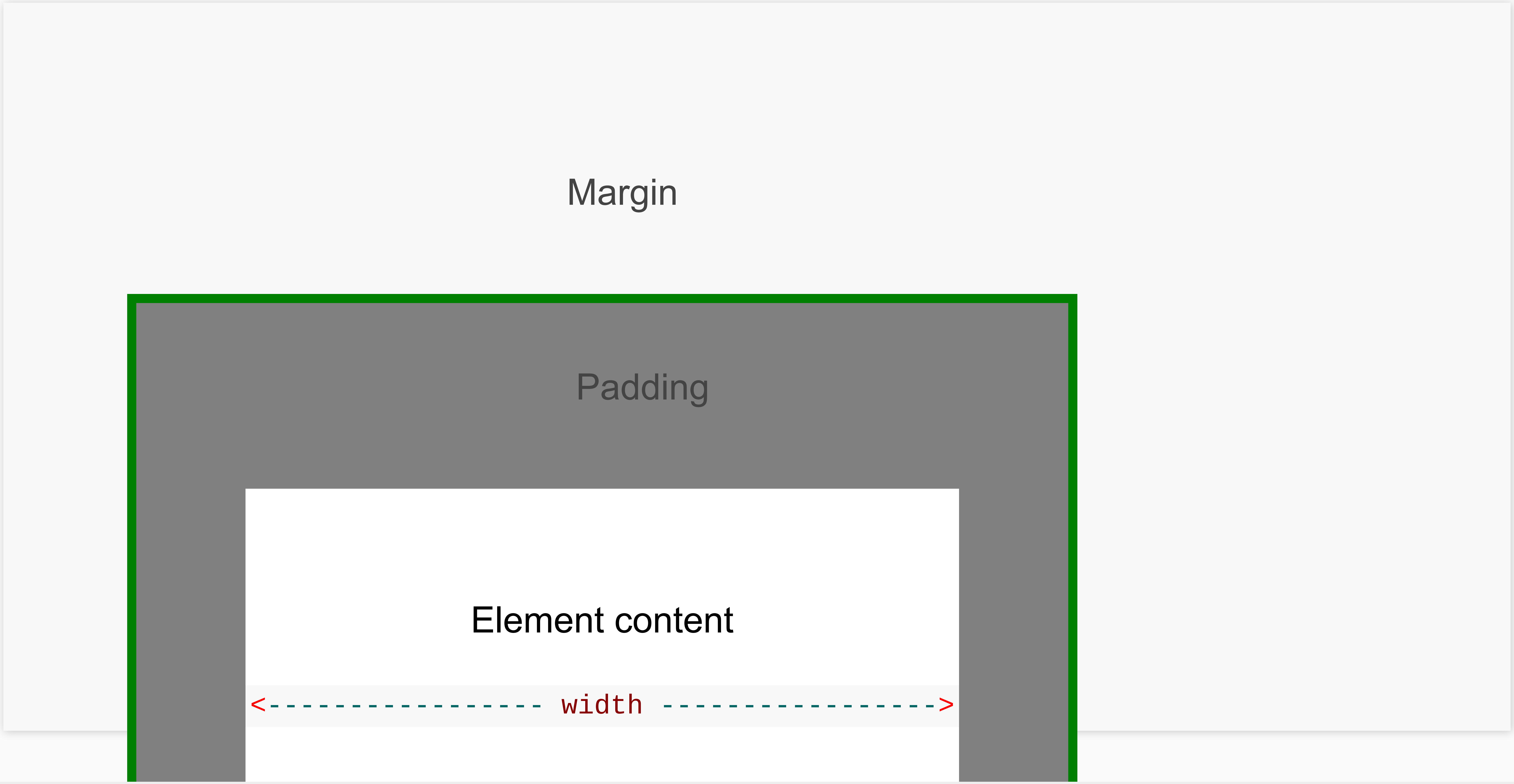
```
#menu { border: 3px #057205ff solid; }
#menu p { border-style: dashed; border-color: inherit; }
```

Styling the Block Elements

Box model

- A model that describes the dimensions of any object on the page
- Box model parts:
  - Content width and height
  - Border width
  - Margins
- There are corresponging CSS properties for each part

Box model



Content width and height

- **Inline elements**
  - Always computed automatically
- **Block elements**
  - Set explicitly using the `width` a `height` properties
  - Computed from remaining features:
    - Size of the parent element
    - Content size
    - Margins and borders
- **Root element** (`<html>`)
  - The size is given by the browser window size

Content width and height

- Width

```
.block {
  width: auto; /* default */
  width: 120px;
  width: 60%;

  min-width: 20em;
  min-height: 50em;
}
```

- Height

```
.block {
  height: auto; /* default */
  height: 30px;
  height: 80%;

  min-height: 20em;
  max-height: 50em;
}
```

```
.block {
  box-sizing: content-box | border-box; /* which box the width and height applies to? */
}
```

- Application order
  - `width` -> `max-width` -> `min-width`

Margin

- For individual sides

```
.block {
  margin-top: 2em;
  margin-right: 10px;
  margin-bottom: 2em;
  margin-left: 2em;
}
```

```
.block {
  margin: 2em;
  margin-right: 10px;
}
```

- At once

```
.block {
  margin: 2em 1em 3em 2em; /* top, right, bottom, left */
  margin: 2em 1em 1em;     /* top, right&left, bottom */
  margin: 2em 1.5em;       /* top&bottom, right&left */
  margin: 1em;             /* all */
}
```

- Automatic margin: `margin: auto`

Padding

- For individual sides

```css
.block {
  padding-top: 2em;
  padding-right: 10px;
  padding-bottom: 2em;
  padding-left: 2em;
}
```

- At once

```css
.block {
  padding: 2em 1em 3em 2em; /* top, right, bottom, left */
  padding: 2em 1em 1em;     /* top, right&left, bottom */
  padding: 2em 1.5em;       /* top&bottom, right&left */
  padding: 1em;             /* all */
}
```

- Padding cannot be automatic.

`auto` Values

- The `margin`, `width` and `height` properties can be set to `auto`
  - `width` and `height` are set to `auto` by default
  - For `margin` the default is 0 but `auto` can be used
- The real values for the `auto` properties are computed automatically
- The algorithm depends on the layout mode

Layout Modes

- **Normal flow** (default)
  - So called in-flow elements
  - Elements are laid out in the document order
  - Inline elements on the lines, block elements below each other
- Other layout modes
  - Floating blocks
  - Positioned elements
  - Flexbox, Grid layout
  - … will be explained later

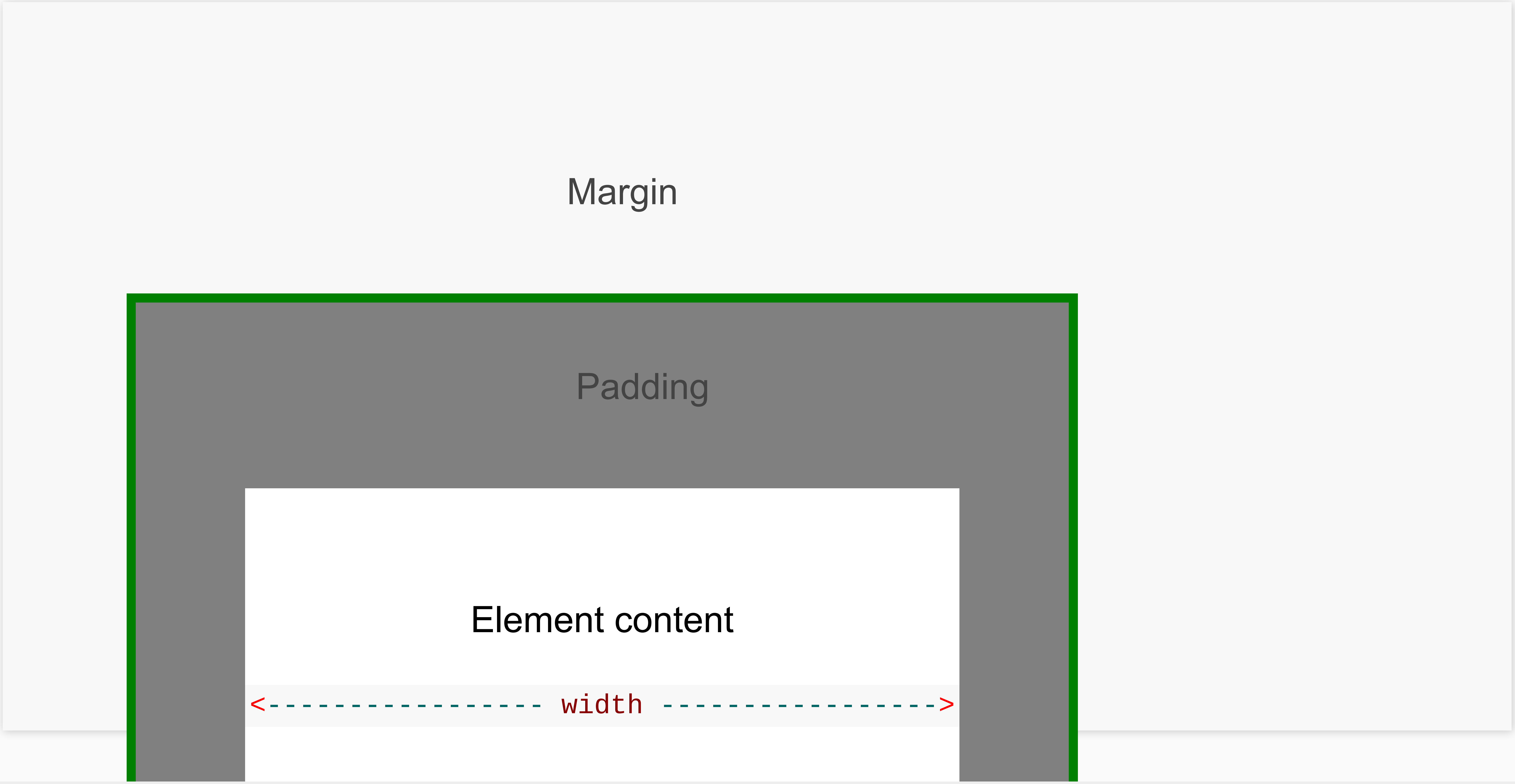
Block Width in Normal Flow

- The block always occupies the whole width of its containing block (roughly its parent block)
  - The whole viewport width for the root element
- It always holds that:

```
margin-left + border-left-width + padding-left
+ width
+ padding-right + border-right-width + margin-right
= containing_block_width
```

- This allows computing the eventual `auto` values
- If none of the values is `auto`, the right margin specfication is ignored and computed automatically

Box model



Computation of `auto` values

- When a single value is `auto`, it is computed from the equation
- If the width is not set (`width=auto`), the `auto` values of margins are interpreted as `0`
- If both the left and right margins are `auto` and the width is set, their final values are equal (`margin-left = margin-right`) => block centering

Margins – Example

Block height

- When `height=auto`, the height is computed automatically so that all the content fits the block
  - Normally, only in-flow content is considered
  - This may be changed by setting the `overflow` property (explained later)
- **Use the `height` limits with care**
  - The text can easily overflow the content box
- For `margin-top` and `margin-bottom`, the `auto` value is always interpreted as `0`

Margin Collapsing

- **Horizontal** margins are **never collapsed**
- **Vertical** adjacent margins are collapsed
  - For two **non-floating** blocks placed below each other
  - For two nested blocks (top or bottom margin)
    - Top margin only if the nested object has `clear: none`
  - Empty blocks (top and bottom margin)
- The resulting margin is the **maximum** of the margins being collapsed

Margin collapsing – Example 1

Margin collapsing – Example 2

Overflowing content

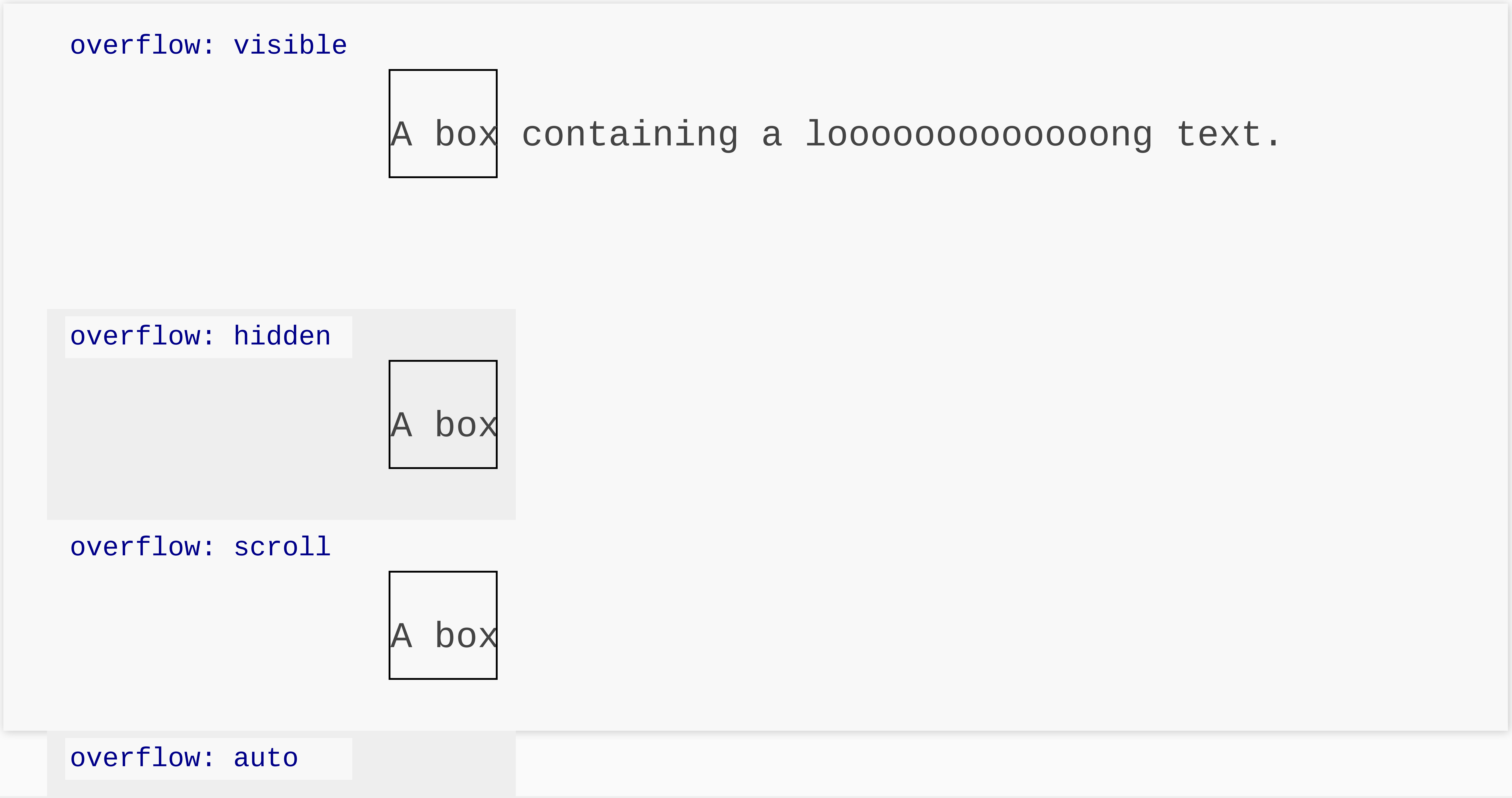
- The `overflow` property - what to do when the content overflows the content box

```
.block {
  overflow: visible; /* let overflow (default) */
  overflow: hidden;  /* trims the rest */
  overflow: scroll;  /* display scrollbars */
  overflow: auto;    /* display scrollbars if needed */
}
```

- Example

```
<div style="width: 5em; height: 6em; border: 2px solid white;">
  A box containing a looooooooooooong text.
</div>
```

Overflow examples



`overflow: visible`

A box containing a loooooooooooooong text.

`overflow: hidden`

A box

`overflow: scroll`

A box

`overflow: auto`

To be continued…

CSS – Block positioning and layout