

Positioning & Layouts

Frontend mentoring program. Brest.

March 4, 2015



Menu and content
dynamic



Menu fixed, Content
dynamic



Menu and content
dynamic



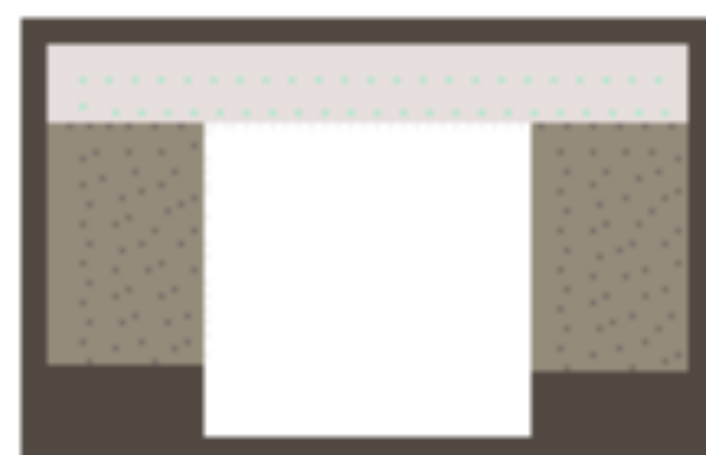
3 columns, all
dynamic



4 columns, all
dynamic



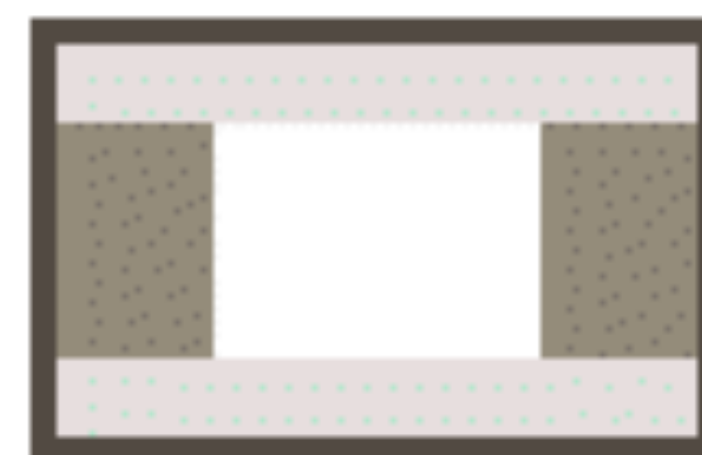
Menu floating



Menu fixed, content
& header dynamic



3 columns fixed
centered



dynamic with
header and footer

CSS

IS

AWESOME

Visual formatting model

- The CSS visual formatting model is the *algorithm* used to process a document and display it on a visual media.
- The visual formatting model will transform each element of the document and generate zero, one or several *boxes* conforming to the CSS box model.

The layout of each box will be defined by

- box dimensions and type
- positioning scheme (normal flow, float, and absolute positioning)
- relationships between elements in the document tree.
- external information (e.g., viewport size, intrinsic dimensions of images, etc.)

Box Model

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

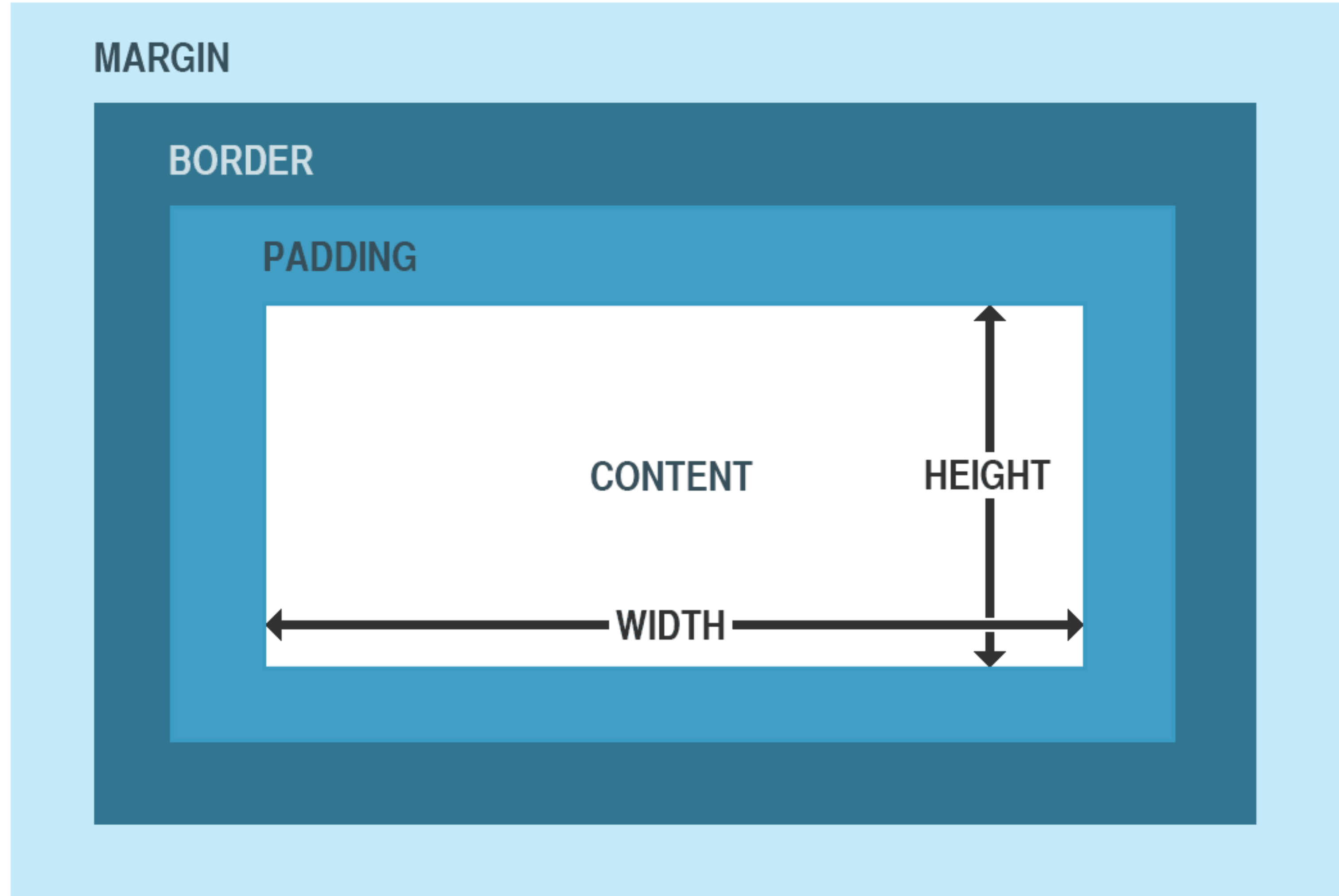


DuckDuckGo

A long, empty rectangular search input field with a thin border.

The search engine that doesn't track you. [Learn More](#)





Definitions

- **Content** - The content of the box, where text and images appear
- **Padding** - Clears an area around the content. The padding is transparent
- **Border** - A border that goes around the padding and content
- **Margin** - Clears an area outside the border. The margin is transparent

Box-sizing

The box-sizing CSS property is used to alter the default CSS box model used to calculate widths and heights of elements. It is possible to use this property to emulate the behavior of browsers that do not correctly support the CSS box model specification.

Box-sizing

content-box | padding-box | border-box

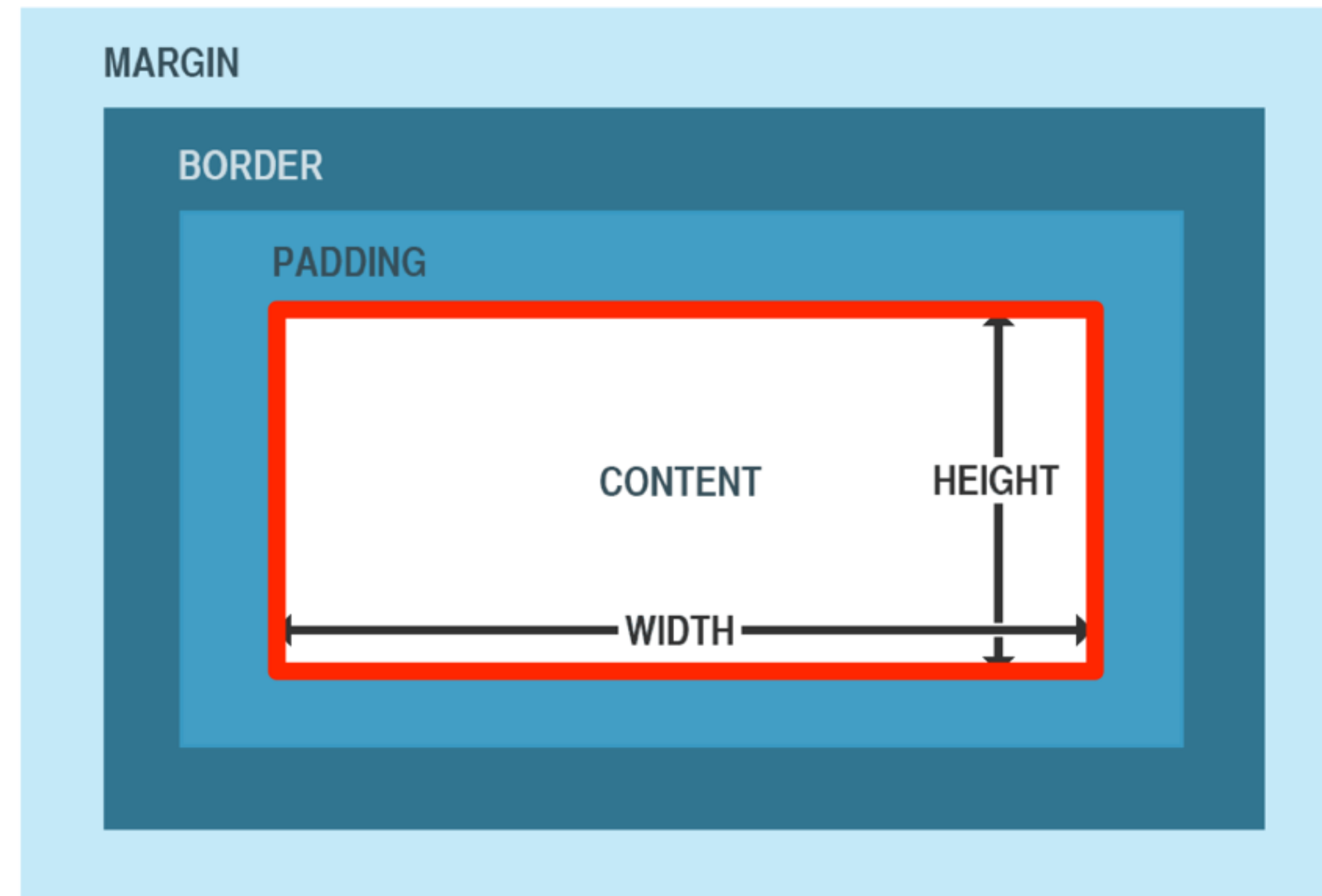
Examples:

```
box-sizing: content-box;
```

```
box-sizing: padding-box;
```

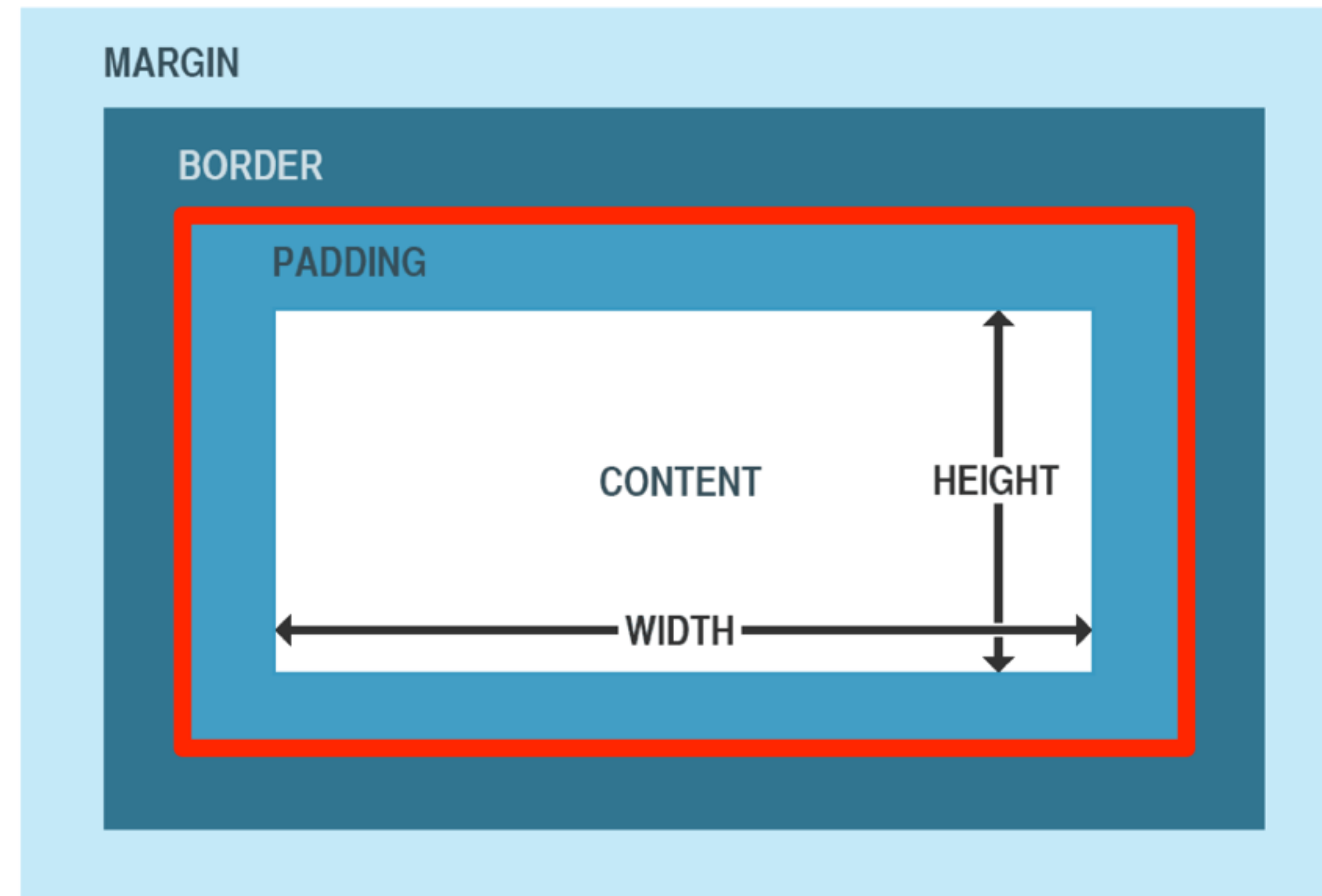
```
box-sizing: border-box;
```

```
box-sizing: inherit;
```



content-box

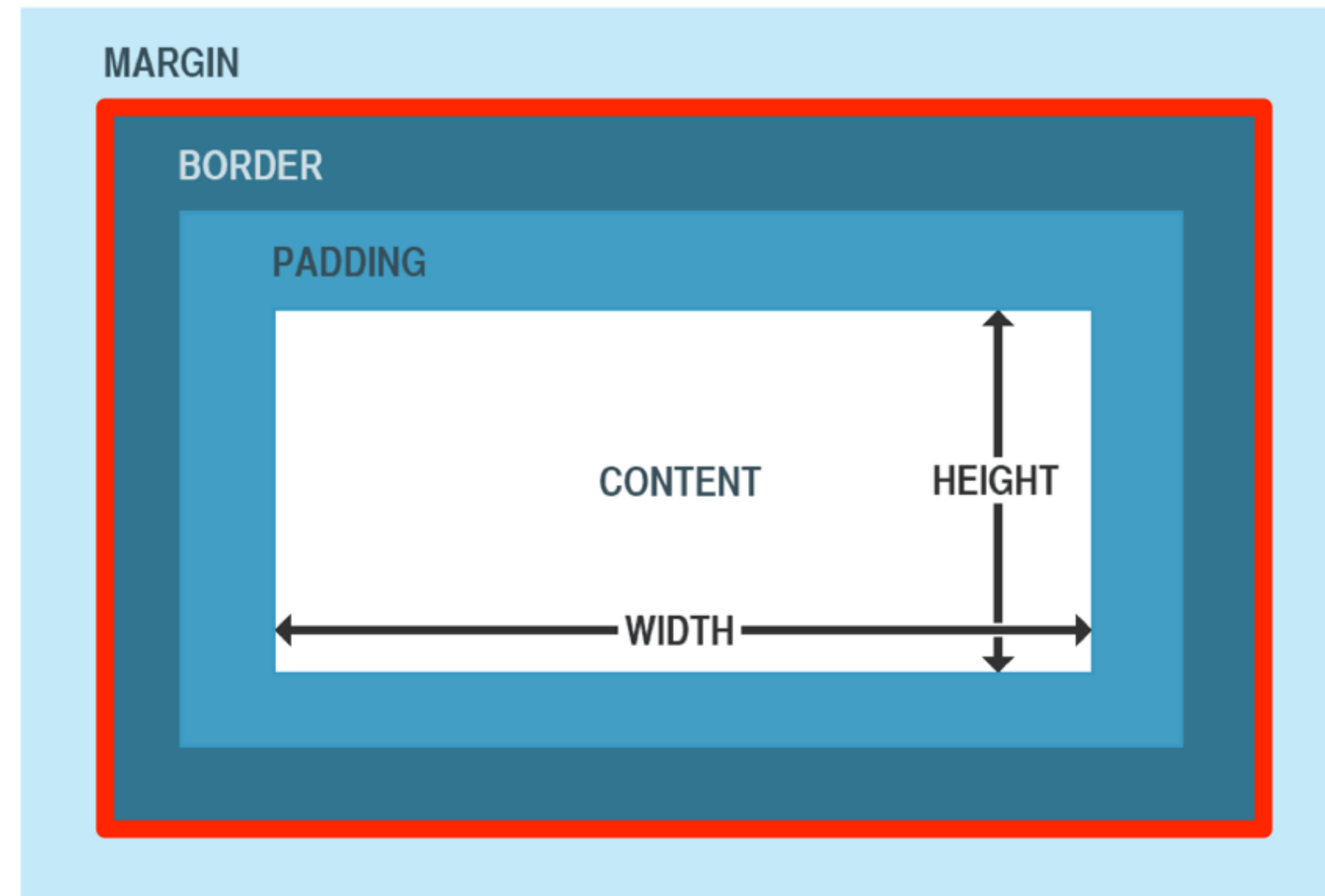
This is the **default** style as specified by the CSS standard. The width and height properties are measured including only the content, but not the padding, border or margin.



padding-box

The width and height properties include the padding size, and do not include the border or margin.

Experimental. Firefox only!



border-box

The width and height properties include the padding and border, but not the margin.

Best practice (old)

```
*, *:before, *:after {  
    box-sizing: border-box;  
}
```


Best practice (new)

```
html {  
    box-sizing: border-box;  
}  
  
*, *:before, *:after {  
    box-sizing: inherit;  
}
```

Display types

Every element on a web page is a rectangular box. The display property in CSS determines just how that rectangular box behaves.

Display property

inline | inline-block | block | none*

Examples:

```
display: inline;  
display: inline-block;  
display: block;  
display: none;
```

* - table, flex and grid are omitted for simplicity

Inline elements

- The default value for all elements is inline
- Most "User Agent stylesheets" reset many elements to "block"
- Think of elements like ``, ``, or ``
- An inline element will accept margin and padding
- Margin and padding will **only push other elements horizontally** away, not vertically
- An inline element will **not accept height and width**. It will just ignore it

Lorem ipsum dolor `inline sit amet`,
consectetur adipiscing elit. In urna ipsum,
scelerisque at orci sed, dictum dictum purus.
Aenean eget ligula blandit, gravida dolor
congue, sagittis magna. Pellentesque diam
ligula, hendrerit in imperdiet et, sodales eget
nisl. Ut at luctus massa. Praesent at erat

display: inline

Inline-block elements

- Very similar to inline
- Will set inline with the natural flow of text (on the "**baseline**")
- **Width and height will be respected**

Lorem ipsum dolor inline-block sit amet ,

consectetur adipiscing elit. In urna ipsum,
scelerisque at orci sed, dictum dictum purus.
Aenean eget ligula blandit, gravida dolor
congue, sagittis magna. Pellentesque diam
ligula, hendrerit in imperdiet et, sodales eget

display: inline-block

Block elements

- A number of elements are set to block by the browser UA stylesheet
- Usually container elements
- `<div>`, `<section>`, ``, `<p>` and `<h1>`
- Do not sit inline but **break** past them
- Take up **as much horizontal space as they can**
- `display:list-item` or `display:table` are also block items

Lorem ipsum dolor

block sit amet

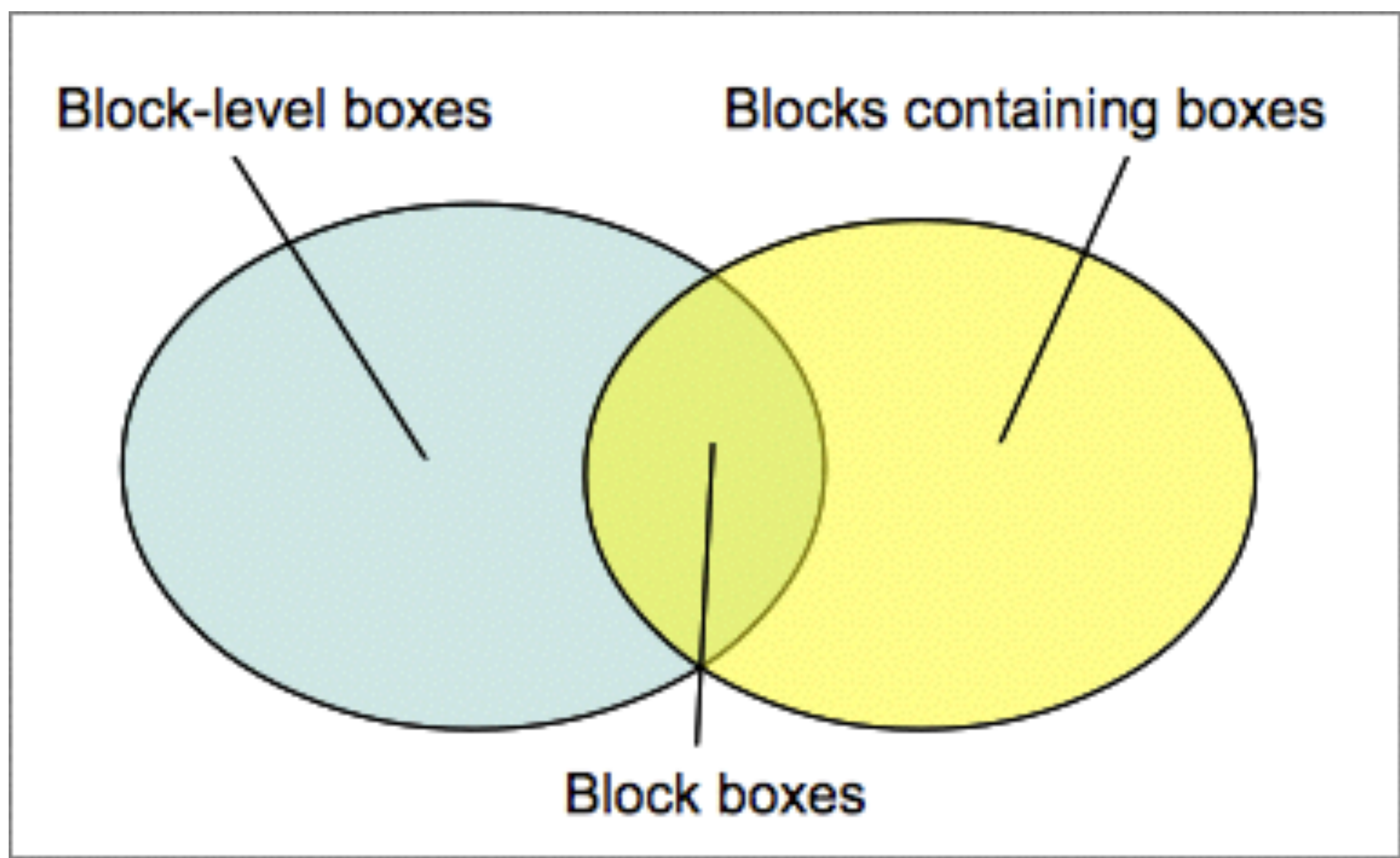
, consectetur adipiscing elit. In urna ipsum,
scelerisque at orci sed, dictum dictum purus.
Aenean eget ligula blandit, gravida dolor
congue, sagittis magna. Pellentesque diam

display: block

Box generation

A box is rendered *relatively* to the edge of its containing block. Usually a box establishes the containing block for its descendants. Note that a box is not constrained by its containing block; when its layout goes outside it, it is said to overflow.

Block-level elements
and block boxes



Block-level elements and block boxes

- An element is said to be block-level when the calculated value of its display CSS property is: *block*, *list-item* or *table*
- A block-level element is visually formatted as a block (e.g. paragraph), intended to be *vertically* stacked
- Each block-level element generates at least one block-level box, called the *principal block-level box*
- Block-level boxes are boxes that participate in a *block formatting context*

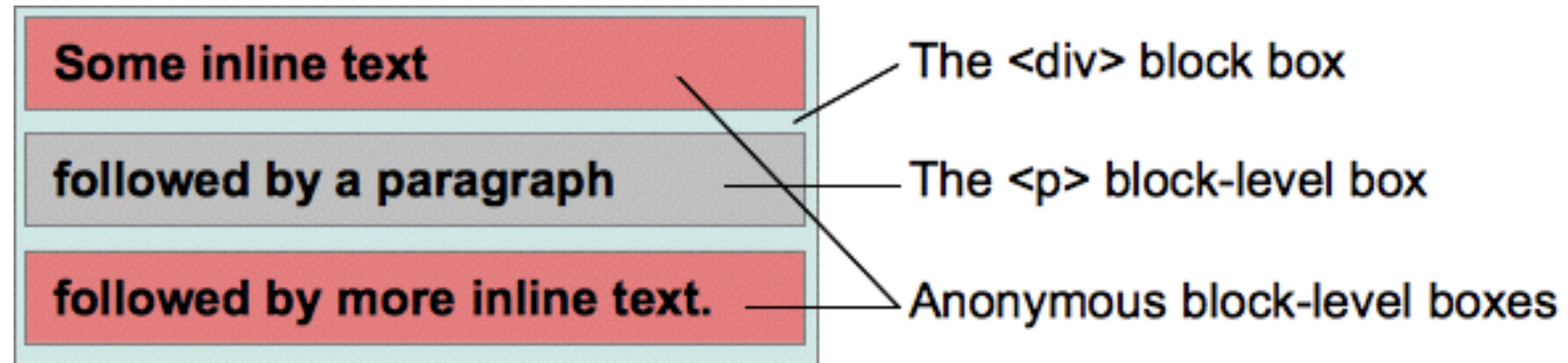
Block container boxes and block boxes

- Block-level box is also a *block container box* (except table boxes and replaced elements)
- A block container box either contains only block-level boxes or establishes an inline formatting context and thus contains only inline-level boxes
- Not all block container boxes are block-level boxes: non-replaced inline blocks and non-replaced table cells are block containers but not block-level boxes
- Block-level boxes that are also block containers are called block boxes

The three terms "block-level box,"
"block container box," and "block
box" are sometimes abbreviated as
"*block*" where unambiguous

Anonymous block boxes

Huh?



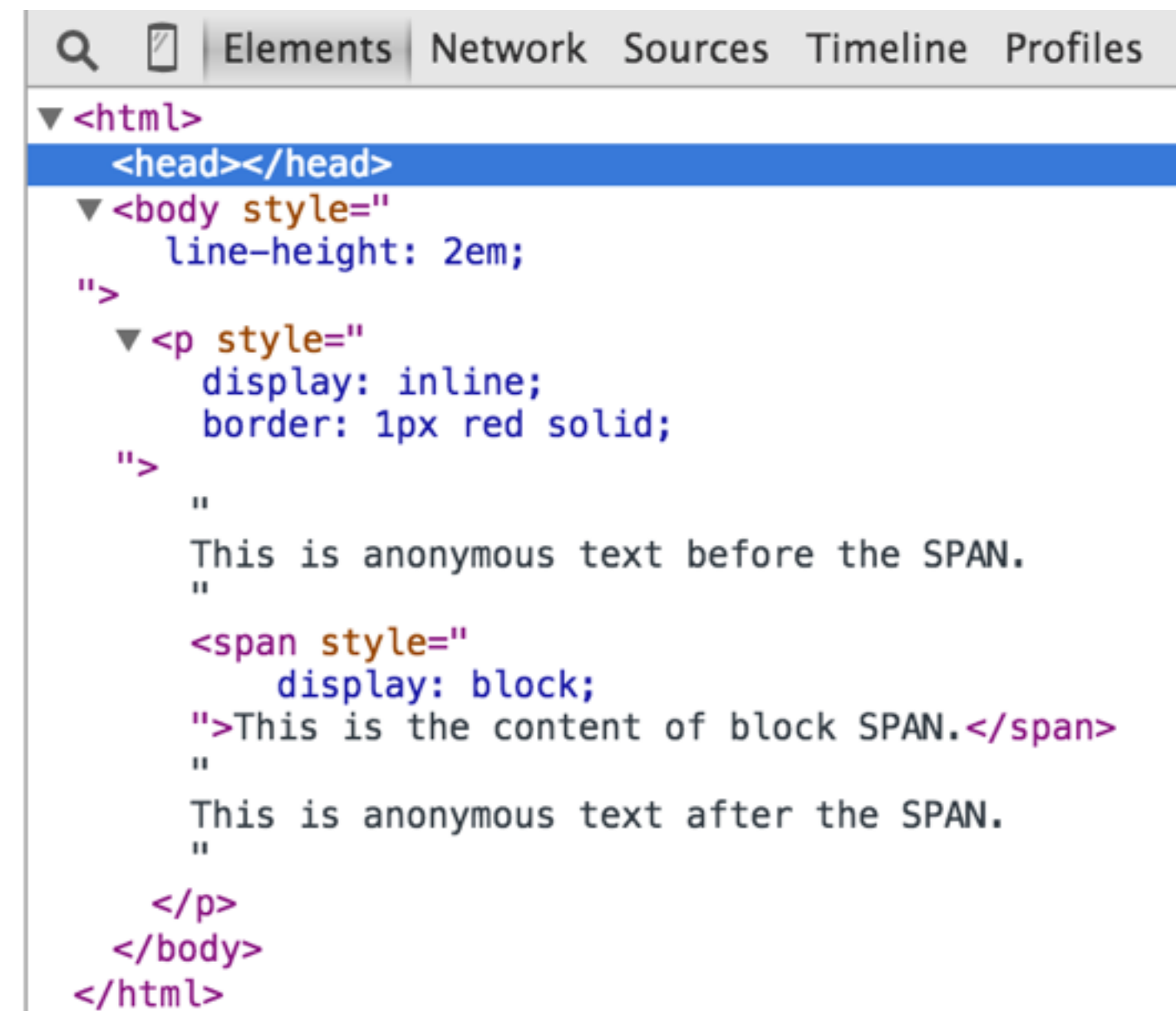
Anonymous block boxes

- The properties of anonymous boxes are inherited from the enclosing non-anonymous box
- Non-inherited properties have their initial value
- Anonymous block boxes are ignored when resolving percentage values that would refer to it: the closest non-anonymous ancestor box is used instead
- Properties set on elements that cause anonymous block boxes to be generated still *apply* to the boxes and content of that element

This is anonymous text before the SPAN.

This is the content of block SPAN.

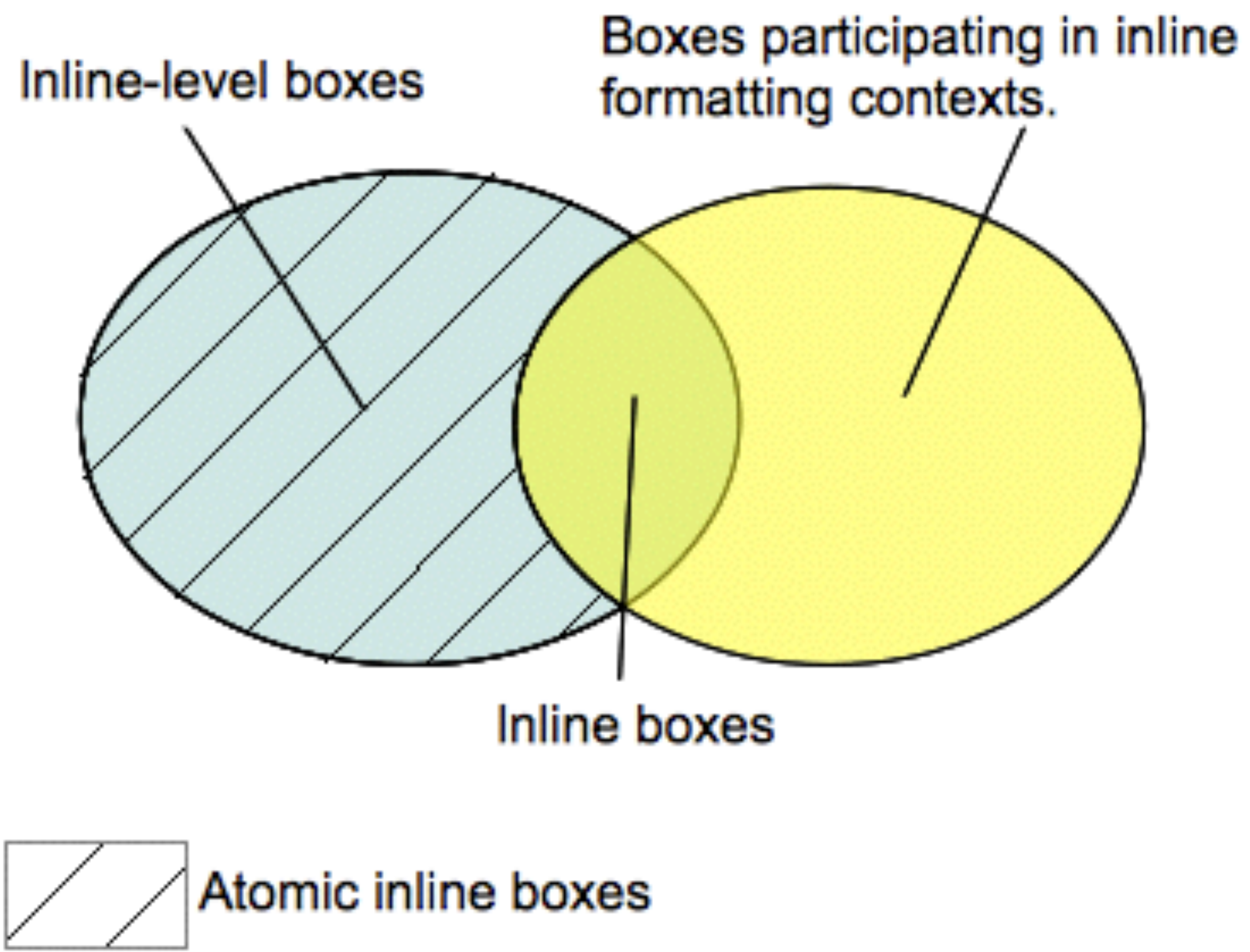
This is anonymous text after the SPAN.



The screenshot shows the 'Elements' tab of a web browser's developer tools. The HTML structure is as follows:

```
<html>
  <head></head>
  <body style="line-height: 2em;">
    <p style="display: inline; border: 1px red solid;">
      "
      This is anonymous text before the SPAN.
      "
      <span style="display: block;">This is the content of block SPAN.</span>
      "
      This is anonymous text after the SPAN.
      "
    </p>
  </body>
</html>
```

Inline-level elements
and inline boxes



Inline-level elements and inline boxes

- Inline-level elements are those elements of the source document that do not form new blocks of content
- The content is distributed in lines
- The following values of the 'display' property make an element inline-level: 'inline', 'inline-table', and 'inline-block'
- Inline-level elements generate inline-level boxes, which are boxes that participate in an inline formatting context
- An inline box is one that is both inline-level and whose contents participate in its containing inline formatting context

Anonymous inline boxes

Kill me plz..

Anonymous inline boxes

- Any text that is directly contained inside a block container element (*not inside an inline element*) must be treated as an anonymous inline element
- Anonymous inline boxes inherit inheritable properties from their block parent box
- Non-inherited properties have their initial value
- White space content that would subsequently be collapsed away according to the 'white-space' property does not generate any anonymous inline boxes
- *There are more types of anonymous boxes that arise when formatting tables :)*

Positioning schemes

Box may be laid out according to three positioning schemes

- **Normal flow.** In CSS 2.1, normal flow includes block formatting of *block-level* boxes, *inline* formatting of inline-level boxes, and *relative and sticky* positioning of block-level and inline-level boxes
- **Floats.** In the float model, a box is first laid out according to the normal flow, then taken out of the flow and shifted to the left or right as far as possible. Content may flow along the side of a float
- **Absolute positioning.** In the absolute positioning model, a box is removed from the normal flow entirely (it has no impact on later siblings) and assigned a position with respect to a containing block

- An element is called out of flow if it is floated, absolutely positioned, or is the root element.
- An element is called in-flow if it is not out-of-flow.

Position property

static | relative | absolute | fixed | inherit

Examples:

```
position: static;  
position: relative;  
position: absolute;  
position: fixed;  
position: sticky;  
position: inherit;
```

Relative positioning

- The box's position is calculated according to the normal flow
- Then the box is offset relative to its normal position
- When a box B is relatively positioned, the position of the following box is calculated as though B *were not offset*.



position: relative

Absolute positioning

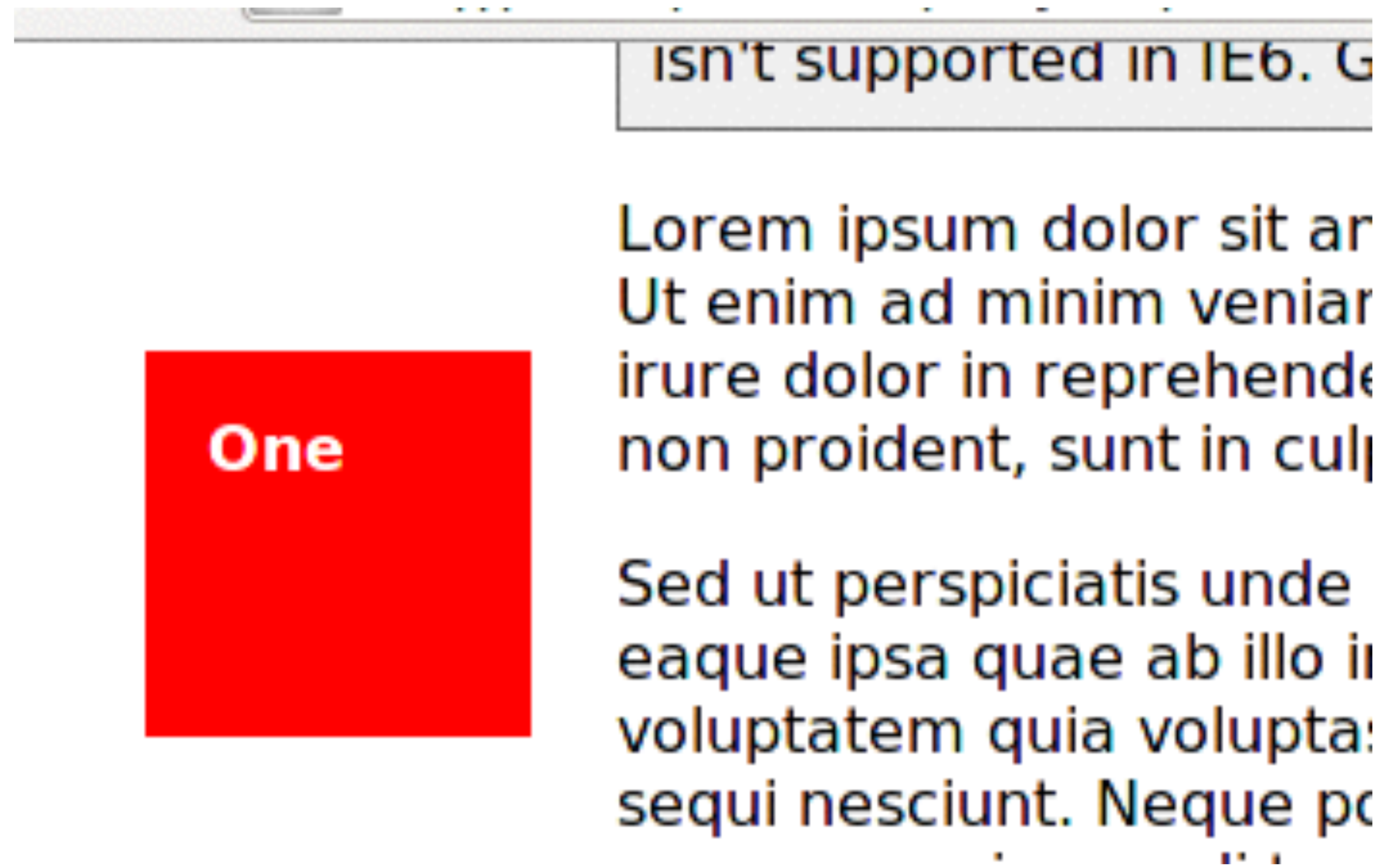
- The box's position is specified with the 'top', 'right', 'bottom', and 'left' properties
- These properties specify offsets with respect to the box's containing block
- Absolutely positioned boxes are taken out of the normal flow
- They have no impact on the layout of later siblings



position: absolute

Fixed positioning

- The box's position is calculated according to the 'absolute' model, but in addition, *the box is fixed with respect to some reference*
- In the case of handheld, projection, screen, tty, and tv media types, the box is fixed with respect to the *viewport* and *does not move* when scrolled
- In the case of the print media type, the box is rendered on every page, and is fixed with respect to the page box, *even if the page is seen through a viewport*



position: fixed

Sticky positioning

- A stickily positioned box is positioned similarly to a relatively positioned box until it crosses a specified threshold, at which point it is treated as fixed positioned
- The offset is computed with reference to the nearest ancestor with a scrolling box, or the viewport if no ancestor has a scrolling box
- A stickily positioned box keeps its normal flow size, including line breaks and the space originally reserved for it
- A stickily positioned box establishes a new containing block for absolutely positioned descendants, just as relative positioning does

Normal flow

- Boxes in the normal flow belong to a formatting context, which may be block or inline, but not both simultaneously.
- Block-level boxes participate in a block formatting context.
- Inline-level boxes participate in an inline formatting context.

Block formatting context

- Floats, absolutely positioned elements, block containers that are not block boxes, and block boxes with 'overflow' other than 'visible' establish new block formatting contexts for their contents
- Boxes are laid out one after the other, vertically, beginning at the top of a containing block
- The vertical distance between two sibling boxes is determined by the 'margin' properties
- Vertical margins between adjacent block-level boxes in a block formatting context *collapse*

Inline formatting contexts

- Boxes are laid out horizontally, one after the other, beginning at the top of a containing block
- Horizontal margins, borders, and padding are respected between these boxes
- The boxes may be aligned vertically in different ways
- The rectangular area that contains the boxes that form a line is called a line box

Line box

- The width of a line box is determined by a containing block and the presence of floats
- The height of a line box is determined by the rules of line height calculations.
- A line box is always *tall enough* for all of the boxes it contains
- When several inline-level boxes cannot fit horizontally within a single line box, they are distributed among two or more vertically-stacked line boxes

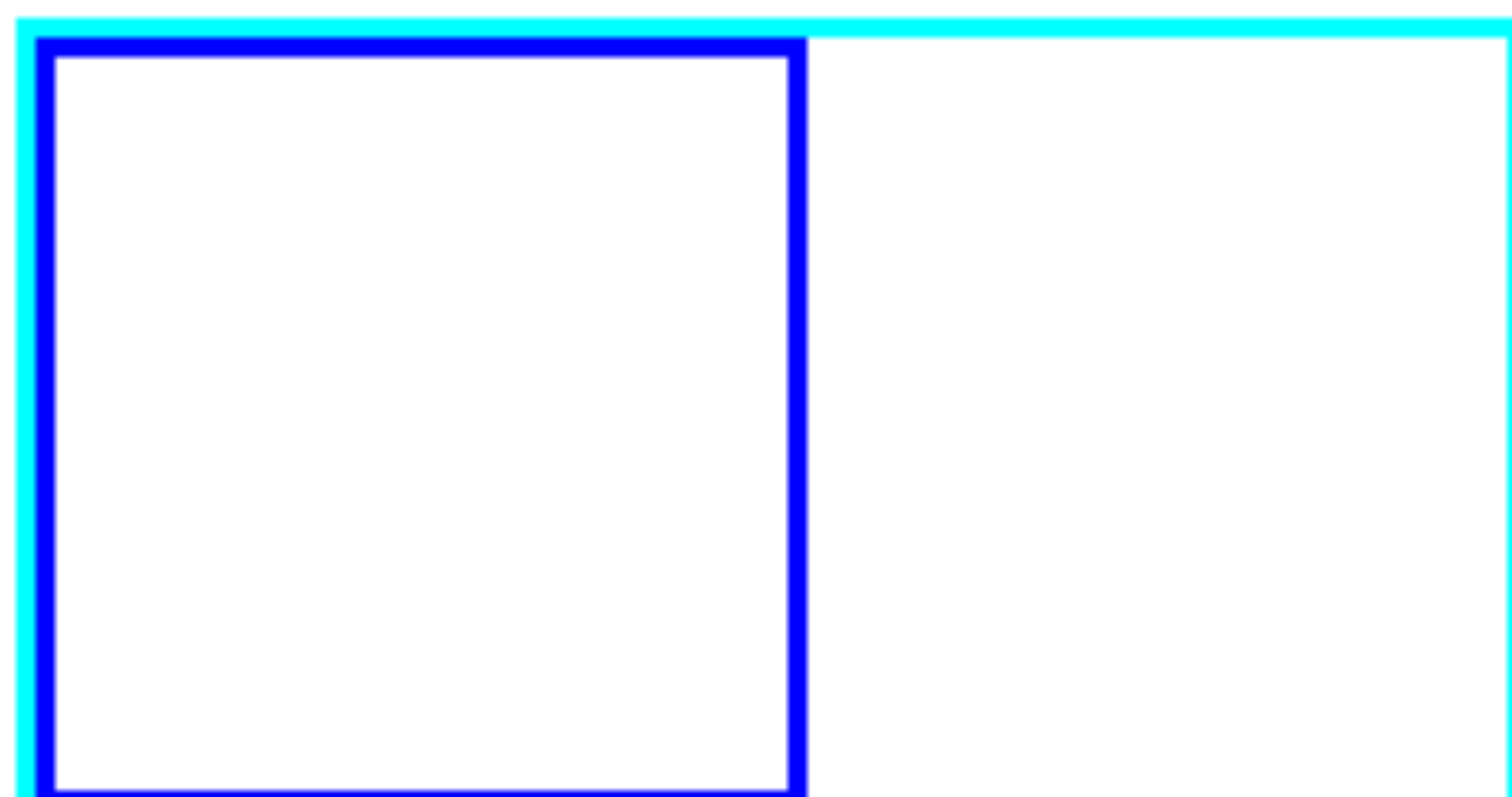
Line box

- When the total width of the inline-level boxes on a line is less than the width of the line box containing them, their horizontal distribution within the line box is determined by the 'text-align' property
- When an inline box exceeds the width of a line box, it is split into several boxes and these boxes are distributed across several line boxes
- If an inline box cannot be split then the inline box overflows the line box
- When an inline box is split, margins, borders, and padding have no visual effect where the split occurs

Floats

- A float is a box that is shifted to the left or right on the current line
- Content may flow along its side
- A floated box is shifted to the left or right until its outer edge touches the containing block edge or the outer edge of another float
- If there is a line box, the outer top of the floated box is aligned with the top of the current line box
- If there is not enough horizontal room for the float, it is shifted downward until either it fits or there are no more floats present

- Since a float is **not in the flow**, non-positioned block boxes created before and after the float box flow vertically **as if the float did not exist**. However, the current and subsequent line boxes created **next to the float are shortened** as necessary to make room for the margin box of the float.
- If a shortened line box is too small to contain any content, then the line box is shifted downward



Supercalifragilisticexpialidocious

- The contents of floats are stacked as if floats generated **new stacking contexts**, except that any positioned elements and elements that actually create new stacking contexts take part in the float's parent stacking context.
- A float can **overlap** other boxes in the normal flow (e.g., when a normal flow box next to a float has negative margins). When this happens, floats are rendered in front of non-positioned in-flow blocks, but behind in-flow inlines

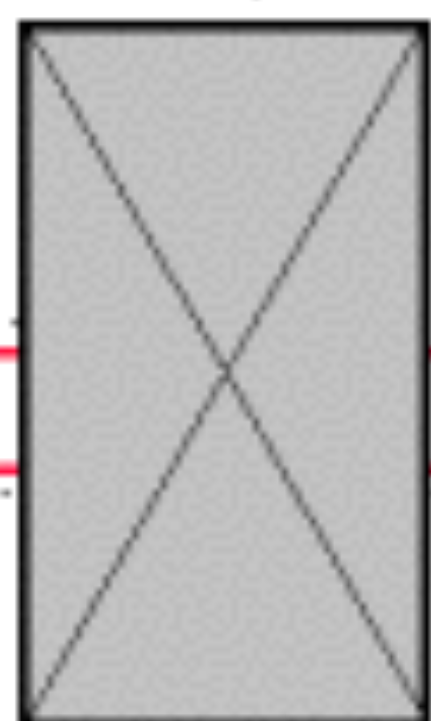
image margin

paragraph margin

paragraph border

paragraph padding

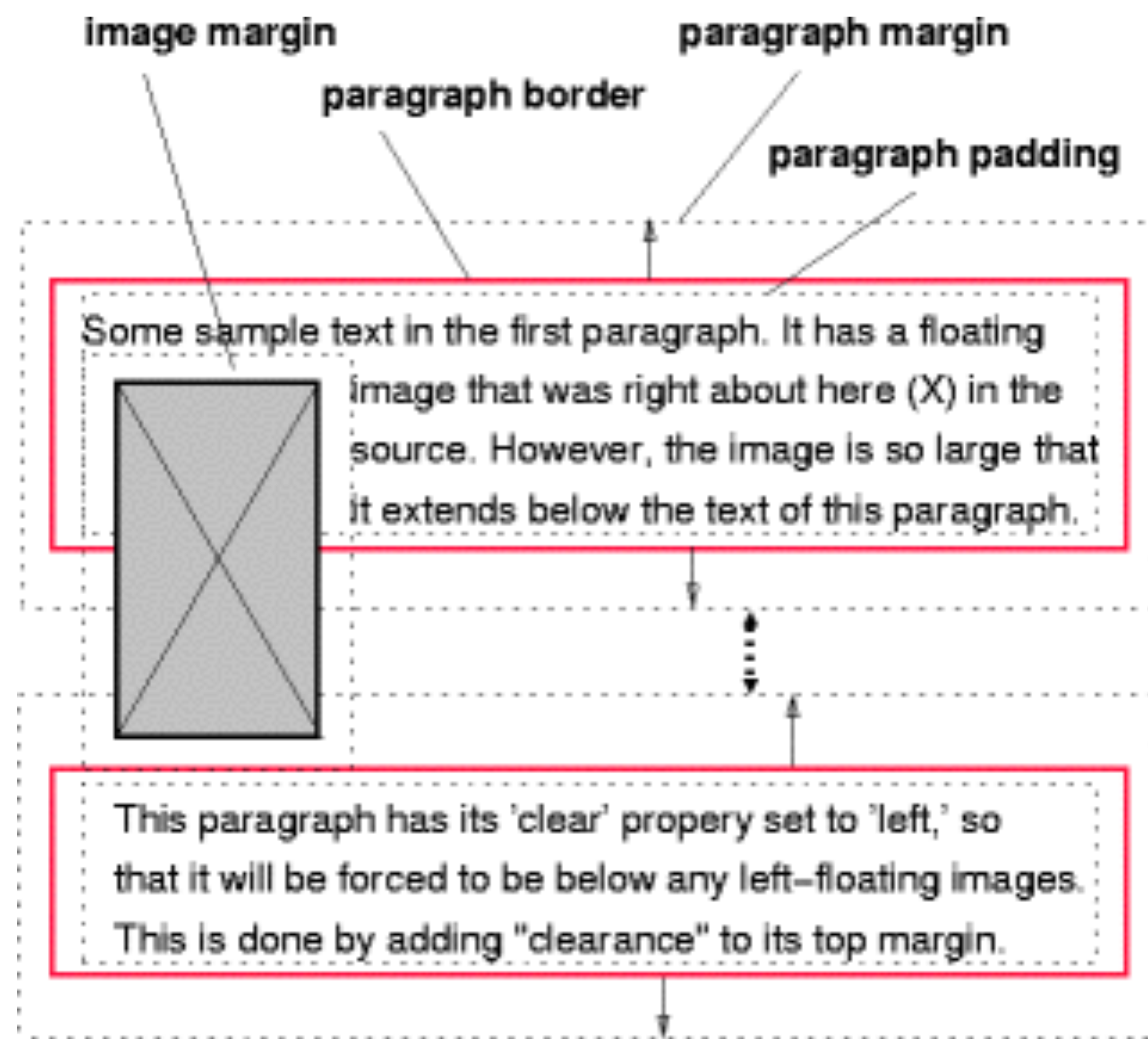
Some sample text in the first paragraph. It has a floating image that was right about here (X) in the source. However, the image is so large that it extends below the text of this paragraph.



The second paragraph is therefore also affected. Any inline boxes in it are "pushed aside," as they are forbidden from coming inside the area delimited by the floating image's margins. Note that the paragraph boxes are still rectangular, but their borders and backgrounds are "clipped" or interrupted by the floating image.

Controlling flow next to floats: the 'clear' property

- This property indicates which sides of an element's box(es) may not be adjacent to an earlier floating box
- The 'clear' property does not consider floats inside the element itself or in other block formatting contexts
- Values other than 'none' potentially introduce clearance



Layered presentation

DIV #1

position: relative;
z-index: 5;

DIV #4

position: relative;
z-index: 6;

DIV #3

position: absolute;
z-index: 4;

DIV #6

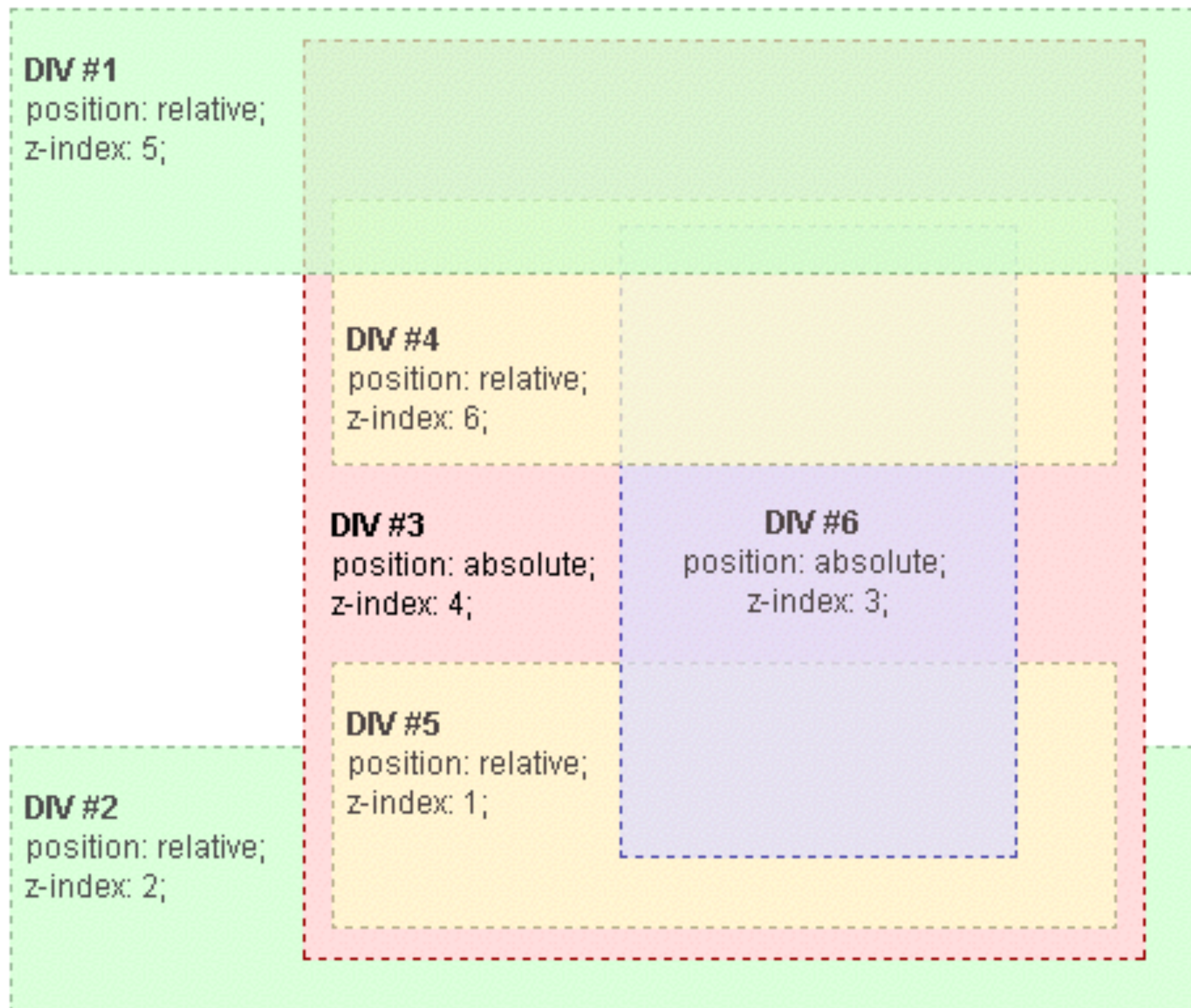
position: absolute;
z-index: 3;

DIV #5

position: relative;
z-index: 1;

DIV #2

position: relative;
z-index: 2;



- In CSS 2.1, each box has a position in three dimensions. In addition to their horizontal and vertical positions, boxes lie along a "z-axis" and are formatted one on top of the other.
- Z-axis positions are particularly relevant when boxes overlap visually.
- The order in which the rendering tree is painted onto the canvas is described in terms of *stacking contexts*

Stacking contexts

- Stacking contexts can contain further stacking contexts.
- A stacking context is atomic from the point of view of its parent stacking context
- Each box belongs to one stacking context
- Each positioned box in a given stacking context has an integer stack level
- It is position on the z-axis *relative other stack levels* within the same stacking context
- Boxes with the same stack level in a stacking context are stacked back-to-front according to document tree order

- The root element forms the root stacking context
- Other stacking contexts are generated by any positioned element (including relatively positioned elements) having a computed value of 'z-index' other than 'auto'
- Stacking contexts are not necessarily related to containing blocks
- Other properties may introduce stacking contexts, for example 'opacity'

Paint order

- the background and borders of the element forming the stacking context.
- the child stacking contexts with negative stack levels (most negative first).
- the in-flow, non-inline-level, non-positioned descendants.
- the non-positioned floats.
- the in-flow, inline-level, non-positioned descendants, including inline tables and inline blocks.
- the child stacking contexts with stack level 0 and the positioned descendants with stack level 0.
- the child stacking contexts with positive stack levels (least positive first).

New stacking contexts

- the root element (HTML),
- positioned (absolutely or relatively) with a z-index value other than "auto",
- a flex item with a z-index value other than "auto",
- elements with an opacity value less than 1. (See the specification for opacity),
- elements with a transform value other than "none",

- elements with a mix-blend-mode value other than "normal",
- elements with isolation set to "isolate",
- on mobile WebKit and Chrome 22+, position: fixed always creates a new stacking context, even when z-index is "auto"
- specifying any attribute above in will-change even you don't write themselves directly

- <http://www.w3.org/TR/CSS21/visuren.html>
- <http://www.w3.org/TR/CSS21/visudet.html#containing-block-details>
- https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Visual_formatting_model
- <https://developer.mozilla.org/en-US/docs/Web/CSS/float>
- https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Understanding_z_index/The_stacking_context
- <https://css-tricks.com/all-about-floats/>

QA

Thank you