

# HTML & CSS

---

*Layout laid out*

# HTML

# CSS

A Venn diagram consisting of two overlapping circles. The left circle is filled with a light red color and contains the word "content". The right circle is filled with a light blue color and contains the word "style". The overlapping area in the center is filled with a light purple color and contains the word "layout". Above the circles, the word "HTML" is centered above the left circle, and the word "CSS" is centered above the right circle.

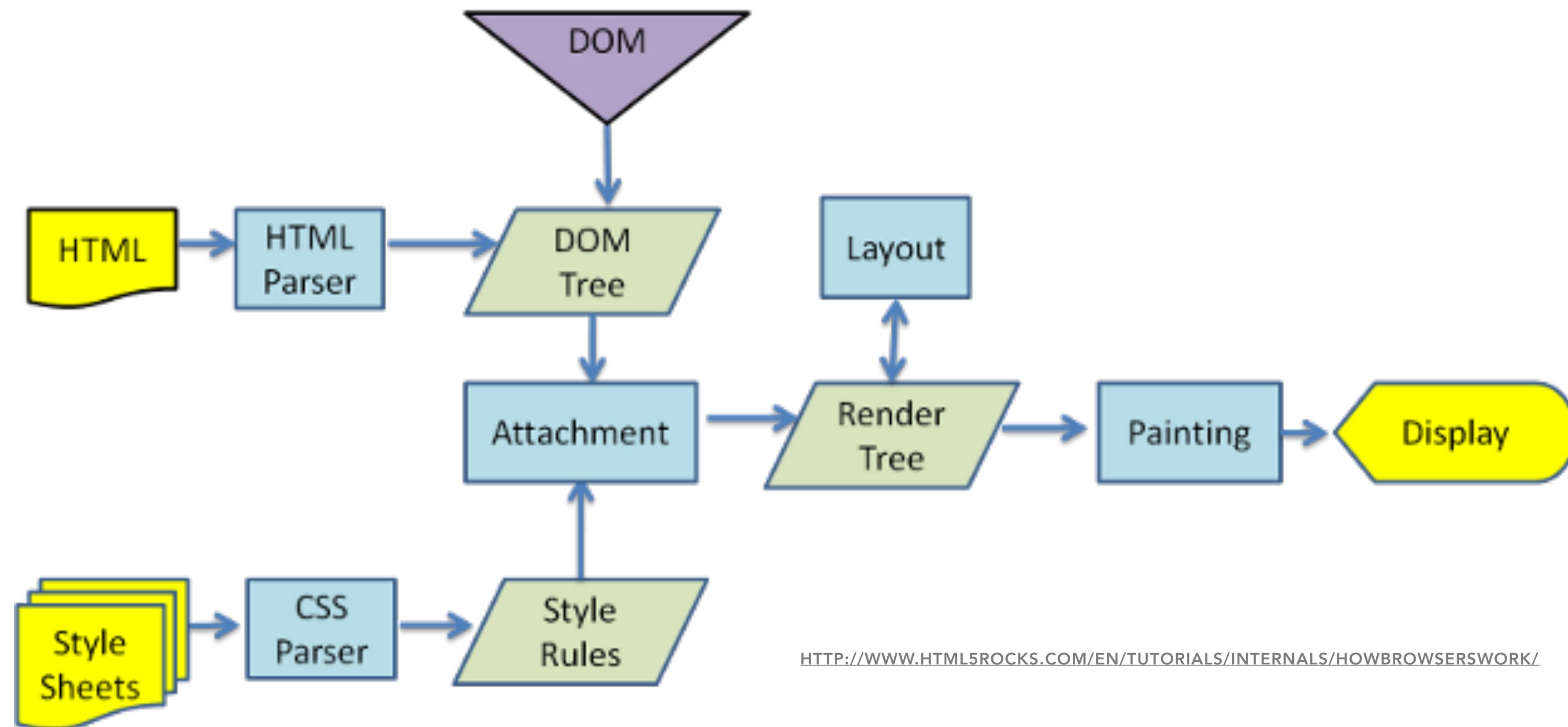
content

layout

style



# BROWSER RENDERING



# WHY IS CSS IMPORTANT?

# CSS IS IMPORTANT

- **The web needs to look nice.**
- **It's the only game in town.**
- **Be a triple threat.**

# WITH CSS

Workshop  
**Shoestring**

Overview Edit Comments Tracking Pairs Topics Materials Videos

**Overview & Objectives**

Many companies use a CSS framework for development speed & convenience. Popular frameworks are carefully designed, compatible across many browsers, and rich in features. However, there is a contingent of developers who believe that frameworks like Bootstrap are too aggressive or opinionated in what they provide, and that it's better to either build your own framework or write ad-hoc styles for each project.

In this workshop, we're going to try to recreate the look of a certain [Bootstrap Template](#) without actually using Bootstrap. To accomplish this, we'll have to create our own CSS framework — a subset of Bootstrap which we'll affectionately call "Shoestring". Shoestring will have three key components:

- Typography
- Grids
- Forms
- You are also encouraged to implement another major Bootstrap component, the navbar.

Along the way we'll learn about building modern semantic CSS using tools like Sass (a high-quality CSS extension language).

< Next >

1. Introduction  
Pre-reading  
[Overview & Objectives](#)

2. Setup  
Get the Repo  
Start the App

3. Sass and Grids  
Intro to Sass  
Using Sass in Our App  
Grid Systems  
Build It Already

4. Responsive Setup  
What Is Responsive Layout?  
Feature Branches  
Update the View

5. Responsive Exercise  
Write the Responsive Branch  
Pull Requests

# WITHOUT CSS

Workshop  
**Shoestring**

Overview Edit Comments Tracking Pairs Topics Materials Videos

**Overview & Objectives**

Many companies use a CSS framework for development speed & convenience. Popular frameworks are carefully designed, compatible across many browsers, and rich in features. However, there is a contingent of developers who believe that frameworks like Bootstrap are too aggressive or opinionated in what they provide, and that it's better to either build your own framework or write ad-hoc styles for each project.

In this workshop, we're going to try to recreate the look of a certain [Bootstrap Template](#) without actually using Bootstrap. To accomplish this, we'll have to create our own CSS framework — a subset of Bootstrap which we'll affectionately call "Shoestring". Shoestring will have three key components:

- Typography
- Grids
- Forms
- You are also encouraged to implement another major Bootstrap component, the navbar.

Along the way we'll learn about building modern semantic CSS using tools like Sass (a high-quality CSS extension language).

[Edit](#)  
Select Cohort

- 1510FE
- 1511
- 1511JS
- 1511JS-MID
- 1601FE
- 1601
- 1601F
- 1601GH

< Next

1. Introduction  
Pre-reading  
[Overview & Objectives](#)

# TERMS

```
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# TERMS

```
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# TERMS

rule

```
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# TERMS

```
article li > a:hover {  
  border: 1px solid red;  
  font-style: italic;  
}
```

# TERMS

```
article li > a:hover {  
  declaration [ border: 1px solid red;  
               font-style: italic;  
               }  
             
```

# TERMS

```
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# TERMS

selector

```
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# TERMS

```
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# TERMS

## combinator

```
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# TERMS

```
article li > a:hover {  
  border: 1px solid red;  
  font-style: italic;  
}
```

# TERMS

property

```
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# TERMS

```
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# TERMS

The diagram illustrates a CSS rule structure. At the top, the word "value" is written in bold black font, with a vertical line pointing down to the "a:hover" pseudo-class selector in the CSS code below. The "a:hover" selector is highlighted with a light gray rectangular box. The CSS code is as follows:

```
article li > a:hover {  
  border: 1px solid red;  
  font-style: italic;  
}
```

# TERMS

```
article li > a:hover{  
    border: 1px solid red;  
    font-style: italic;  
}
```

# TERMS

```
keyword  
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# RULE EXAMPLE

```
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# RULE EXAMPLE

apply these styles →

```
article li > a:hover {  
    border: 1px solid red;  
    font-style: italic;  
}
```

# RULE EXAMPLE

apply these styles →

```
article li > a:hover  
border: 1px solid red;  
font-style: italic;
```

to any elements matching this selector

# RULE EXAMPLE

apply these styles →

```
article li > a:hover  
border: 1px solid red;  
font-style: italic;
```

to any elements matching this selector

even for any future changes

# RULE EXAMPLE

apply these styles →

```
article li > a:hover  
border: 1px solid red;  
font-style: italic;
```

to any elements matching **this** selector

even for any future changes ***declarative!***

# SELECTORS

# SELECTORS

**tag**

# SELECTORS

**tag**

**class**

# SELECTORS

**tag**

**class**

**id**

# SELECTORS

**tag**

**class**

**id**

**attribute**

# SELECTORS

**tag**

**class**

**id**

**attribute**

**pseudo-element**

# SELECTORS

**tag**

**class**

**id**

**attribute**

**pseudo-element**

**pseudo-class**

# SELECTORS

**tag**

**class**

**id**

**attribute**

**pseudo-element**

**pseudo-class**

\*

# SELECTORS

<b>tag</b>	<code>input</code>
<b>class</b>	<code>.btn</code>
<b>id</b>	<code>#upload</code>
<b>attribute</b>	<code>[type="file"]</code>
<b>pseudo-element</b>	<code>::after</code>
<b>pseudo-class</b>	<code>:hover</code>
*	*

# COMBINATORS

# COMBINATORS

**descendant**

**child**

**next sibling**

**later sibling**

# COMBINATORS

**descendant**      (whitespace)  
**child**  
**next sibling**  
**later sibling**

# COMBINATORS

**descendant**      (whitespace)

**child**      >

**next sibling**

**later sibling**

# COMBINATORS

**descendant**      (whitespace)

**child**      >

**next sibling**      +

**later sibling**

# COMBINATORS

**descendant**      (whitespace)

**child**      >

**next sibling**      +

**later sibling**      ~

# BEWARE!

tag.class

tag .class

tag,.class

tag+.class

# BEWARE!

tag.class

element with BOTH tag AND .class

tag .class

tag,.class

tag+.class

# BEWARE!

`tag.class` element with BOTH `tag` AND `.class`

`tag .class` element with `.class` whose ANCESTOR matches `tag`

`tag,.class`

`tag+.class`

# BEWARE!

`tag.class` element with BOTH `tag` AND `.class`

`tag .class` element with `.class` whose ANCESTOR matches `tag`

`tag,.class` element with EITHER `tag` OR `.class`

`tag+.class`

# BEWARE!

`tag.class` element with BOTH `tag` AND `.class`

`tag .class` element with `.class` whose ANCESTOR matches `tag`

`tag,.class` element with EITHER `tag` OR `.class`

`tag+.class` element with `.class` whose left SIBLING matches `tag`

# What happens when rules conflict?



# CASCADING STYLE SHEETS

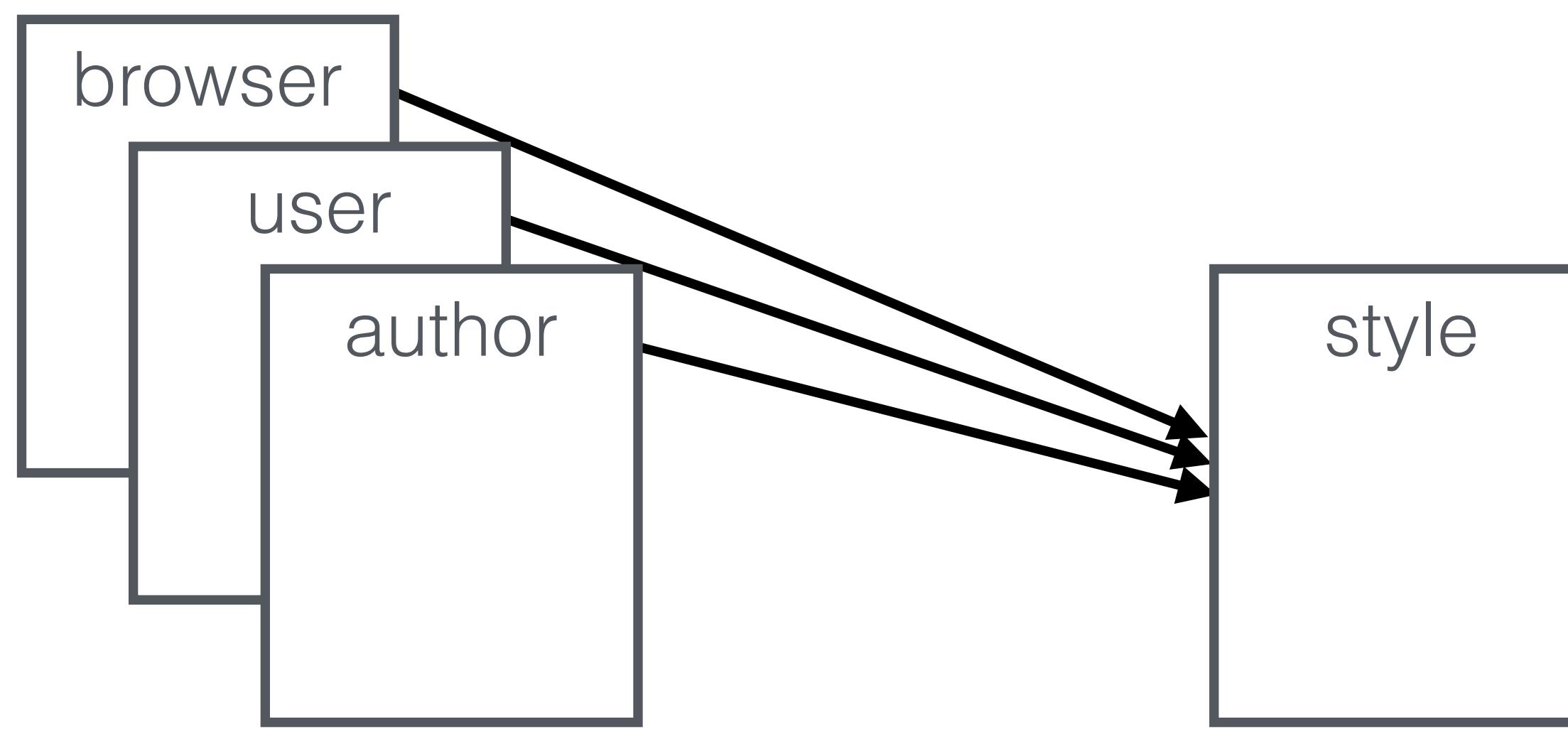
# CASCADING STYLE SHEETS

# CASCADING

In ~1994... CSS had one feature that distinguished it from all the [competing style languages]: it took into account that on the Web the style of a document couldn't be designed by either the author or the reader on their own, but that their wishes had to be combined, or "cascaded," in some way.

CASCADING STYLE SHEETS, DESIGNING FOR THE WEB, BY HÅKON WIUM LIE AND BERT BOS (1999) - CHAPTER 20

# CASCADING



# What happens when rules conflict?



```
div {  
  background: red;  
}
```



```
<div id="thing"></div>
```



```
#thing {  
  background: blue;  
}
```



```
<div id="thing"></div>
```

```
div {  
  background: red;  
}
```



```
#thing {  
  background: blue;  
}
```



```
div {  
  background: red;  
}
```



```
<div class="foo"></div>
```



```
.foo {  
  background: green;  
}
```

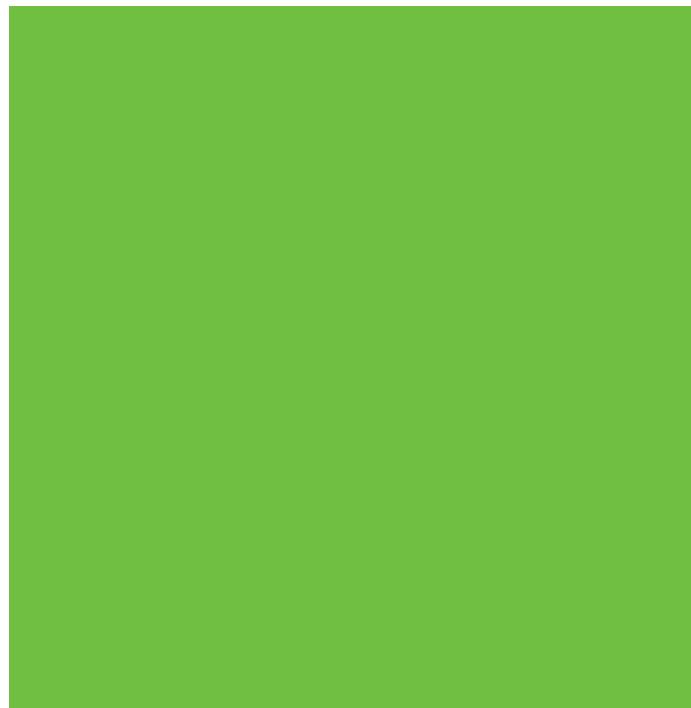


```
<div class="foo"></div>
```

```
div {  
  background: red;  
}
```

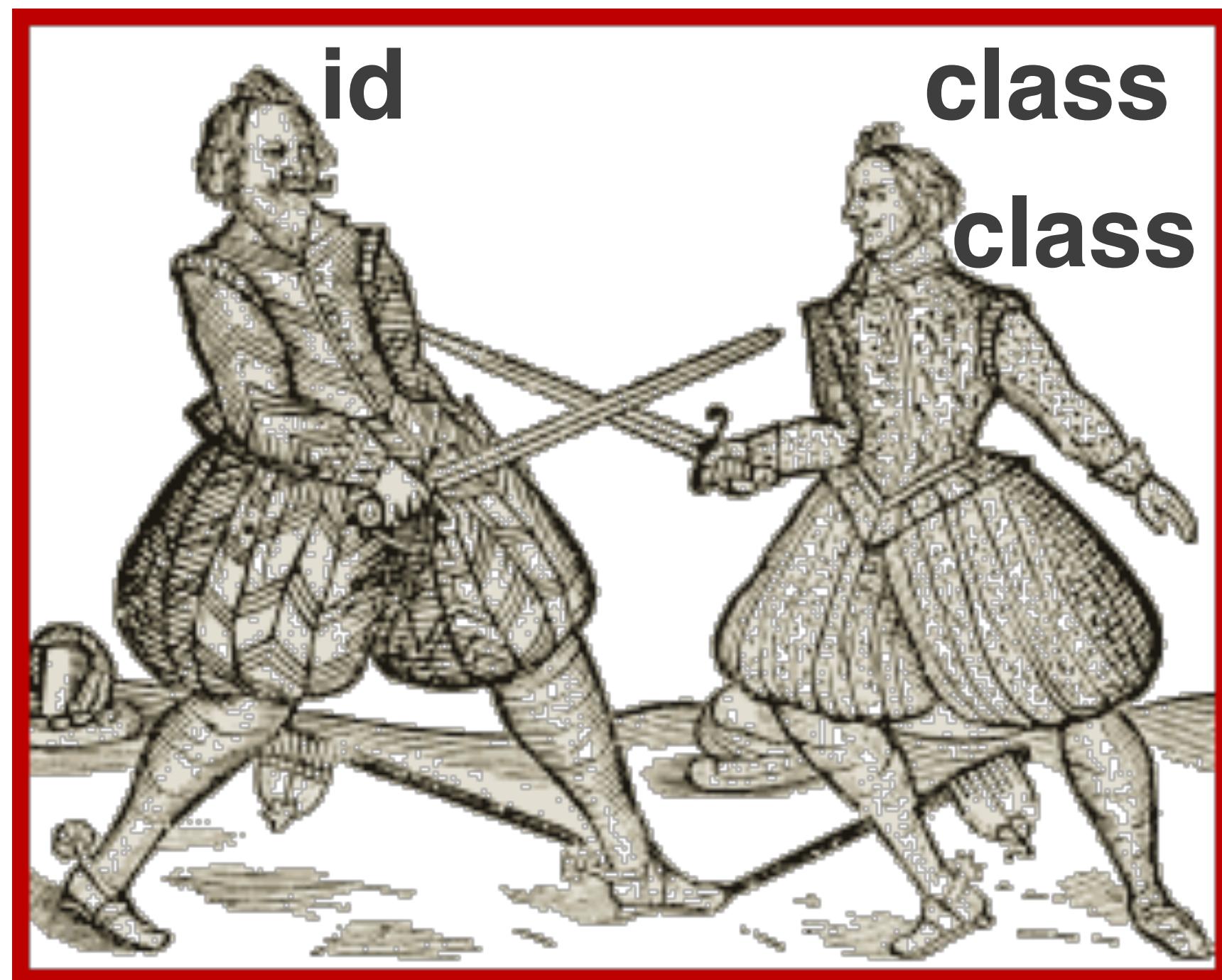
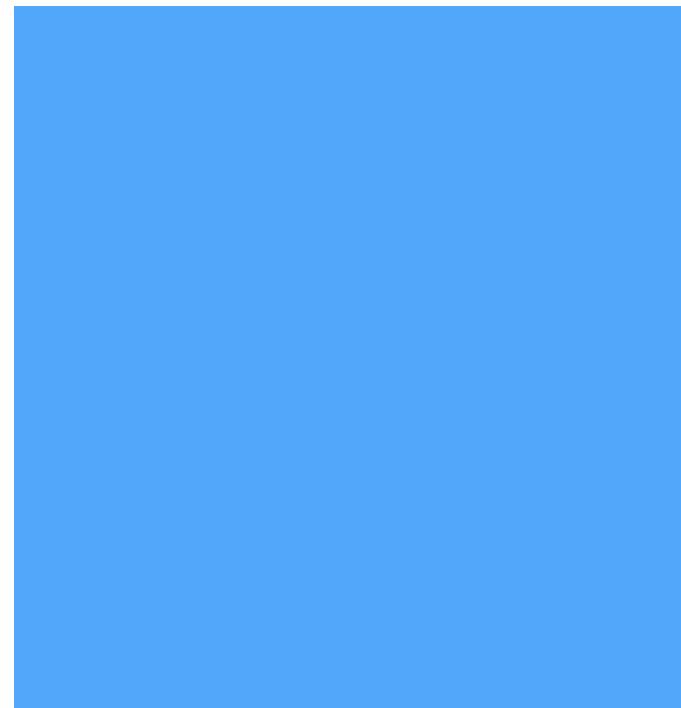


```
.foo {  
  background: green;  
}
```



```
<div id="thing" class="foo bar"></div>
```

```
#thing {  
  background: blue;  
}
```

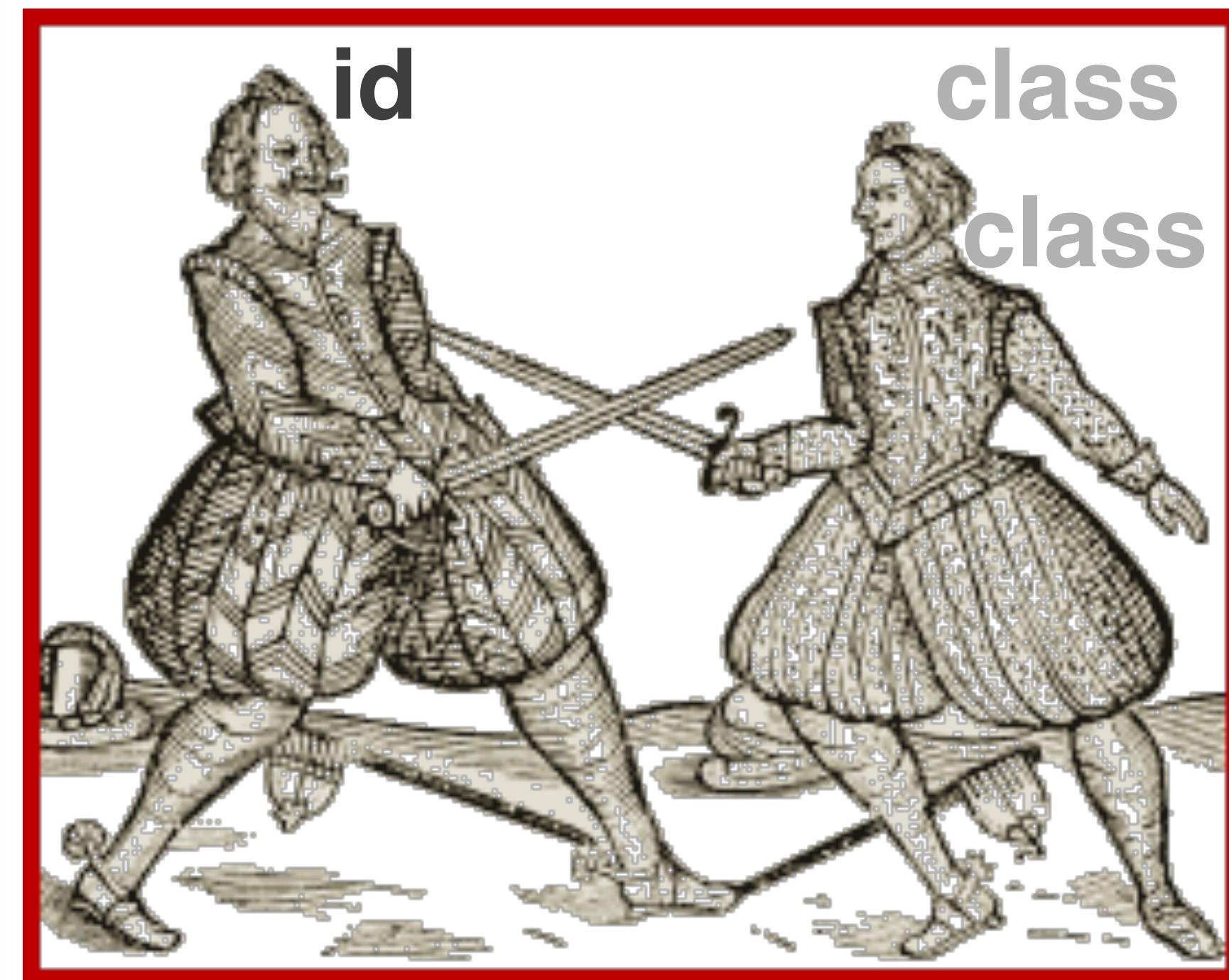
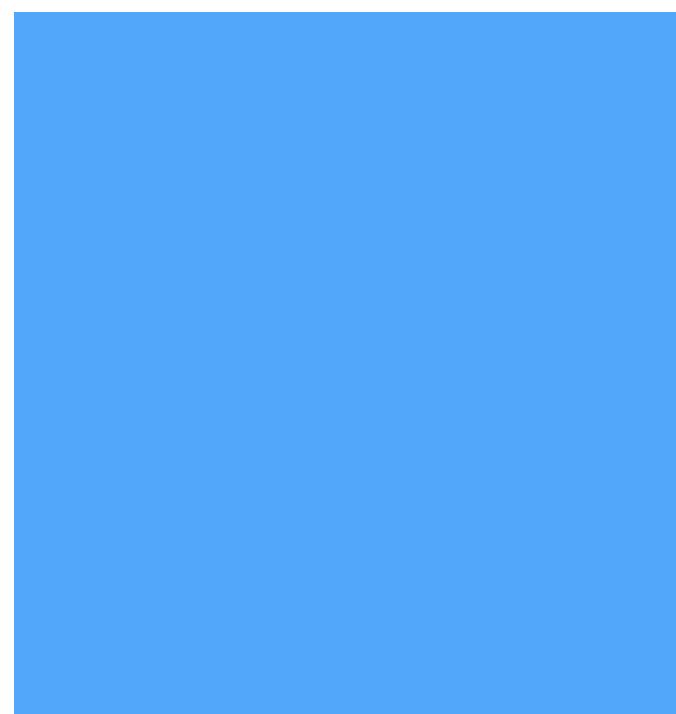


```
.foo.bar {  
  background: green;  
}
```

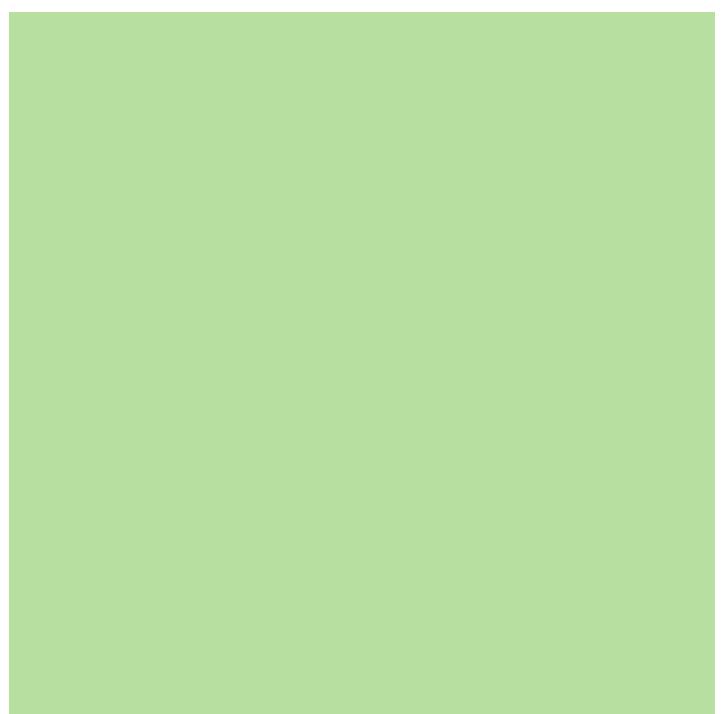


```
<div id="thing" class="foo bar"></div>
```

```
#thing {  
  background: blue;  
}
```

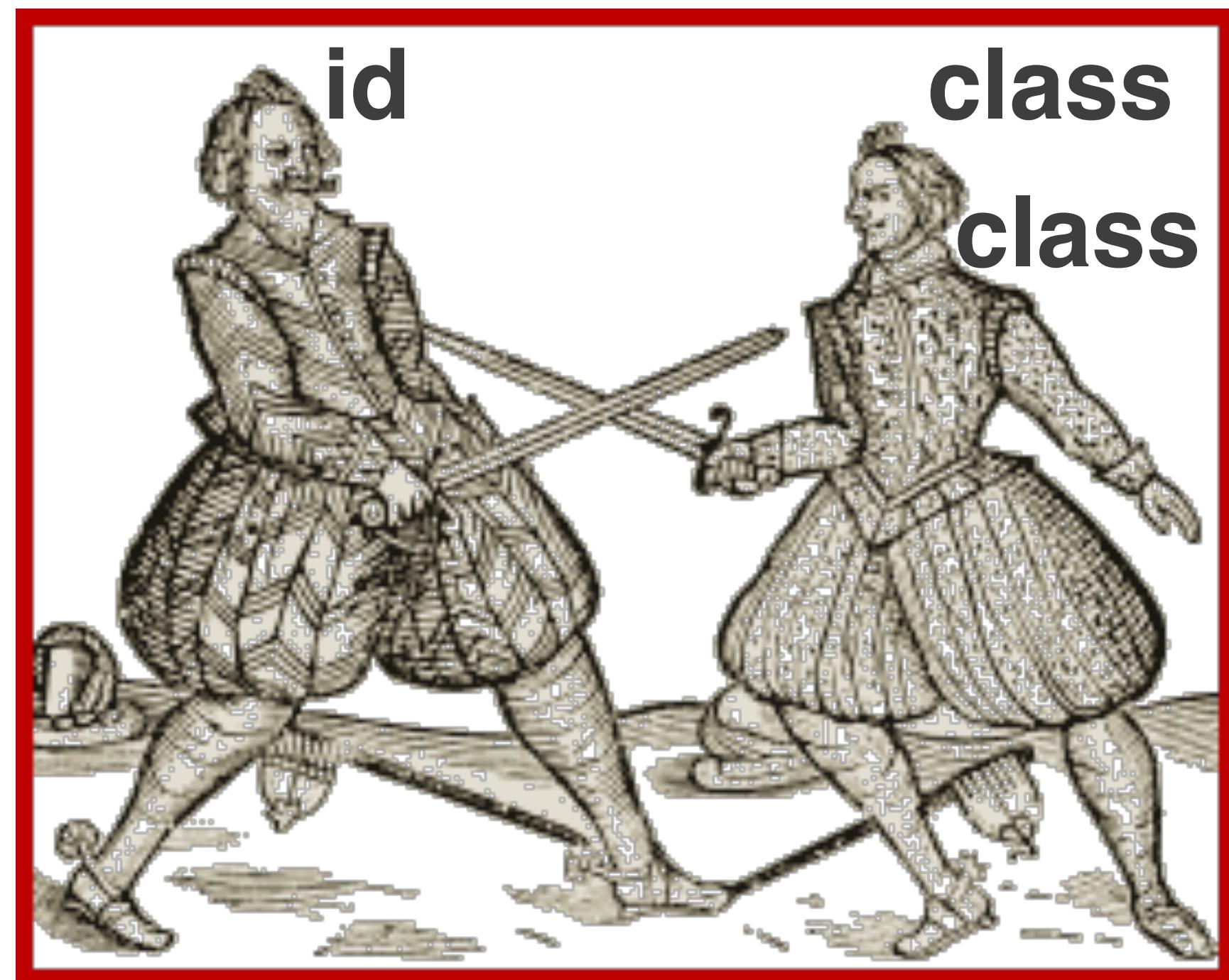


```
.foo.bar {  
  background: green;  
}
```



```
<div class="outer">  
  <div id="thing" class="foo" style="background:orange;"></div>  
</div>
```

```
#thing {  
  background: blue;  
}
```



```
.outer .foo {  
  background: green;  
}
```



```
<div class="outer">  
  <div id="thing" class="foo" style="background:orange;"></div>  
</div>
```

```
#thing {  
  background: blue;  
}
```

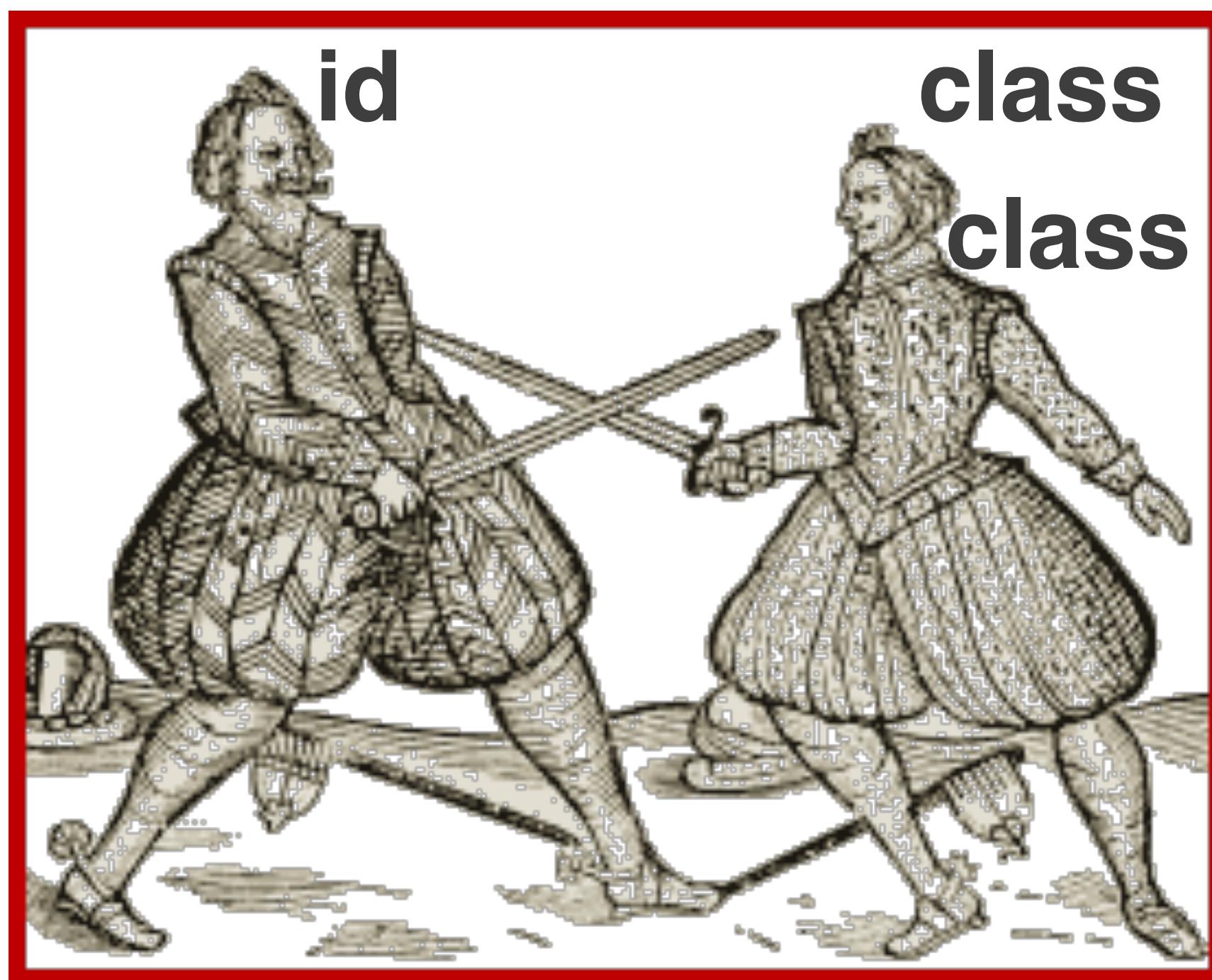


```
.outer .foo {  
  background: green;  
}
```



```
<div class="outer">  
  <div id="thing" class="foo" style="background:orange;"></div>  
</div>
```

```
div {  
  background: red !important;  
}
```

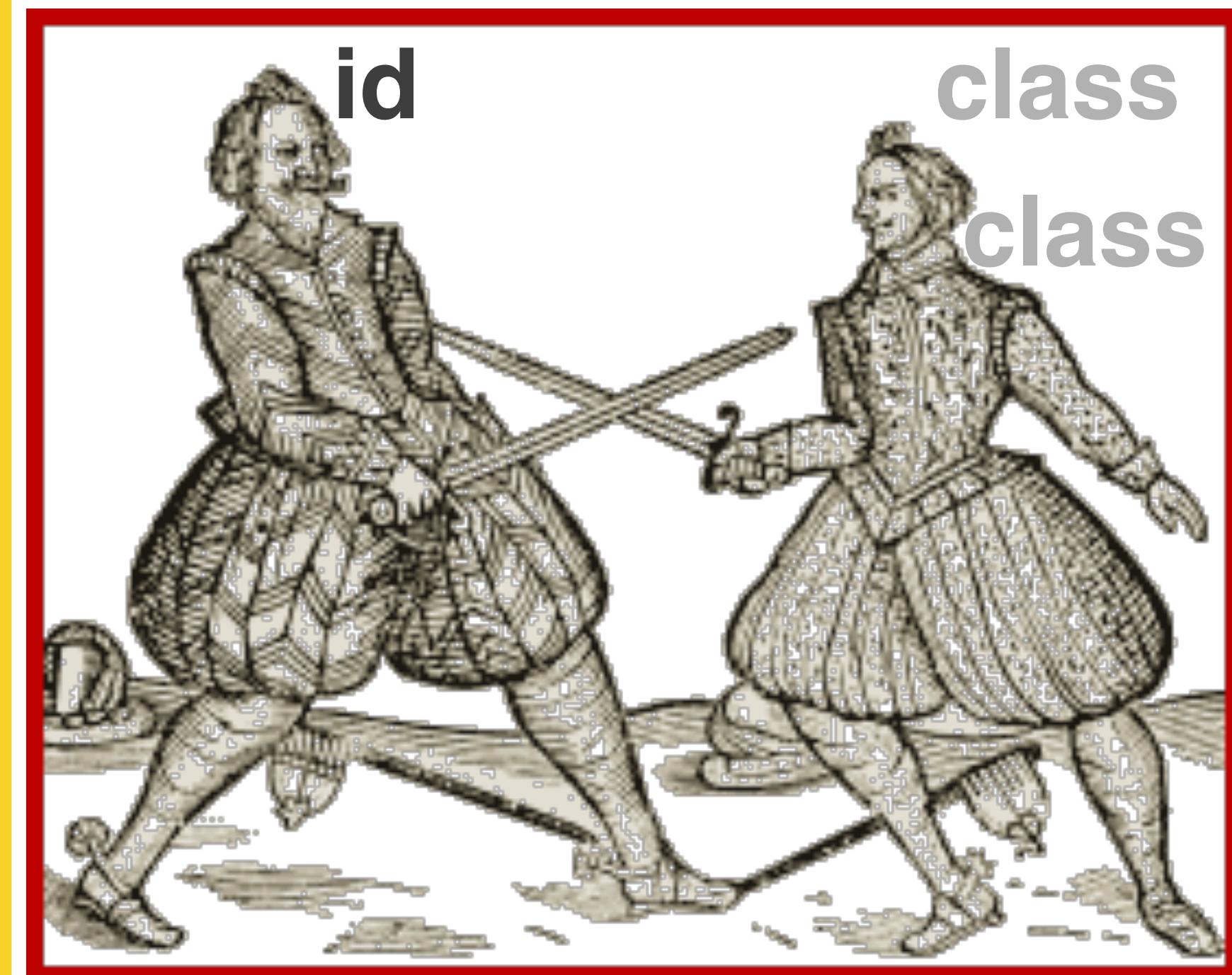


```
.outer .foo {  
  background: green;  
}
```



```
<div class="outer">  
  <div id="thing" class="foo" style="background:orange;"></div>  
</div>
```

```
div {  
  background: red !important;  
}
```

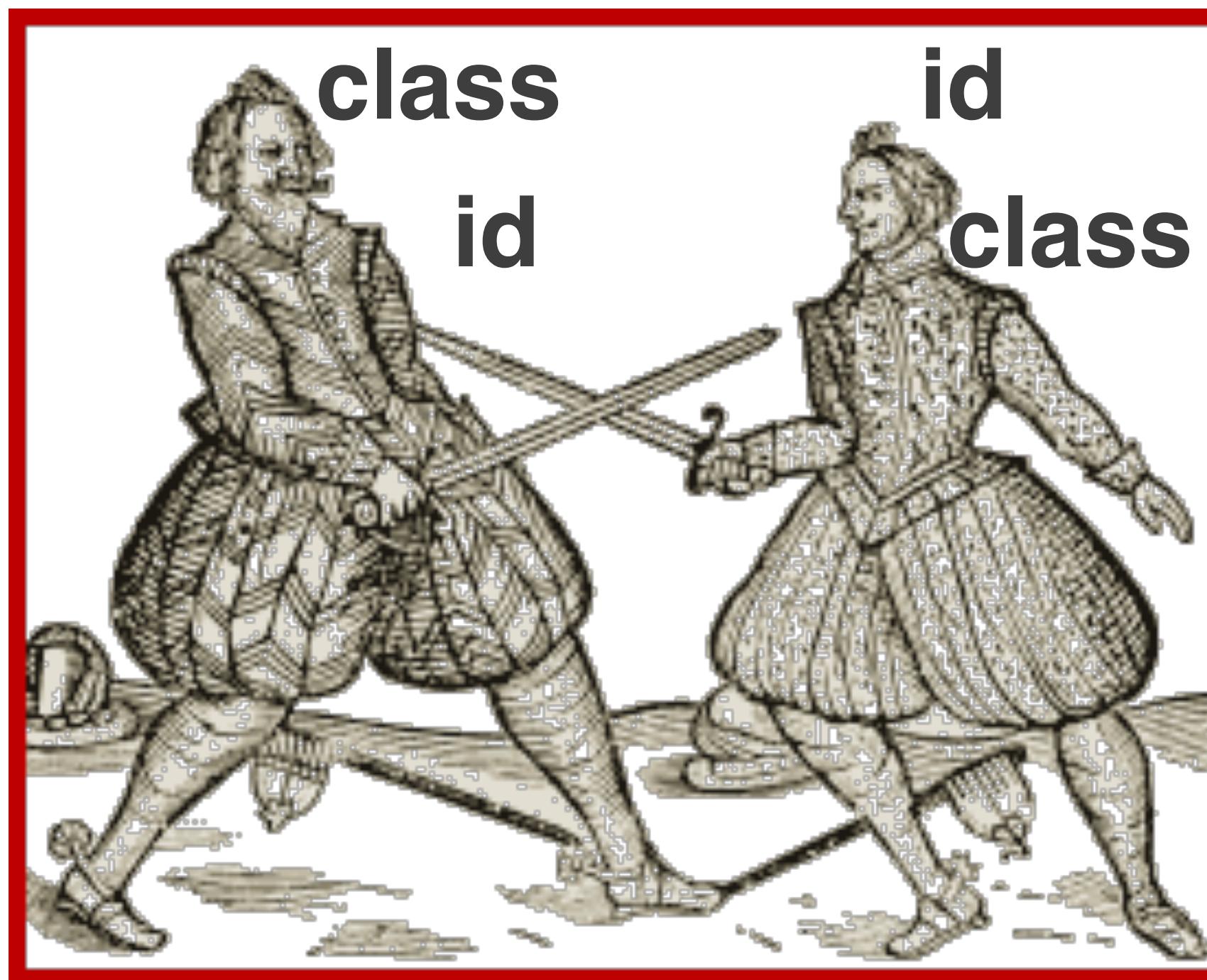
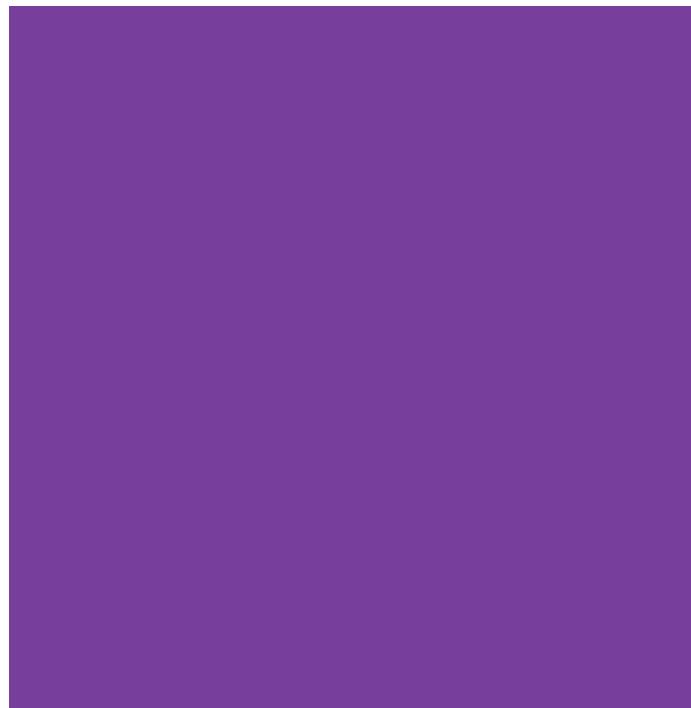


```
.outer .foo {  
  background: green;  
}
```



```
<div id="profile" class="outer">  
  <div id="thing" class="foo"></div>  
</div>
```

```
.outer #thing {  
  background: purple;  
}
```



```
#profile .foo {  
  background: brown;  
}
```



**depends on which rule gets defined *last***

```
<div id="profile" class="outer">  
  <div id="thing" class="foo"></div>  
</div>
```

```
.outer #thing {  
  background: purple;  
}
```



```
#profile .foo {  
  background: brown;  
}
```



# DECLARATION SPECIFICITY

>

>

>

>

>

# DECLARATION SPECIFICITY

important

>

>

>

>

>

# DECLARATION SPECIFICITY



# DECLARATION SPECIFICITY



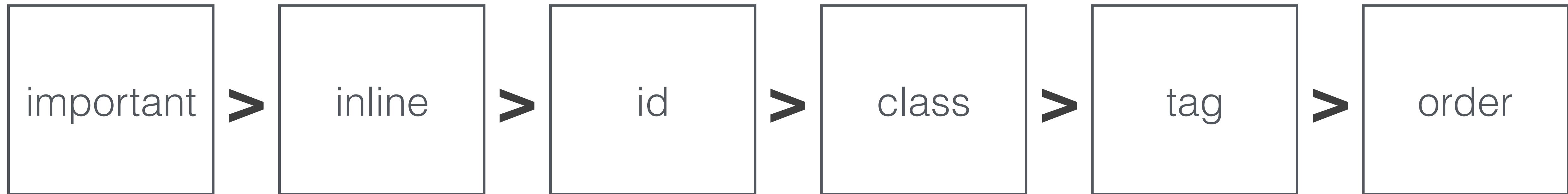
# DECLARATION SPECIFICITY



# DECLARATION SPECIFICITY



# DECLARATION SPECIFICITY



# DECLARATION SPECIFICITY

includes attribute  
and pseudo-class

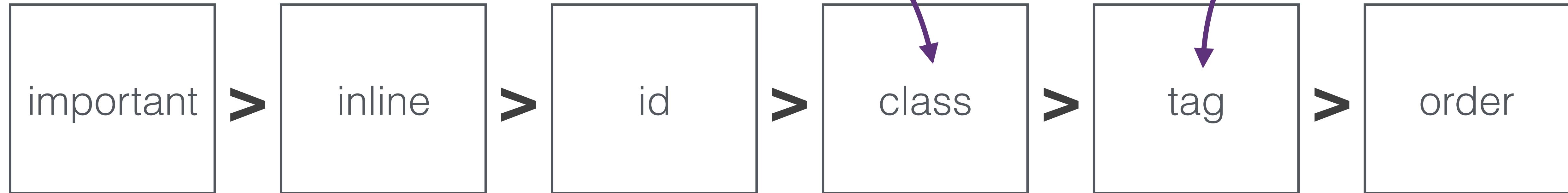
includes pseudo-element



# DECLARATION SPECIFICITY

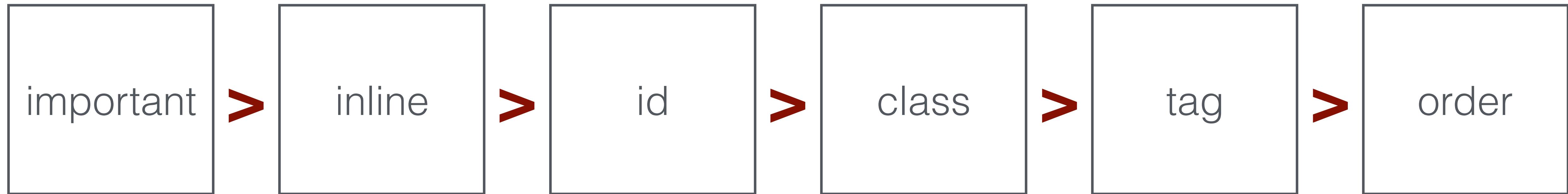
includes attribute  
and pseudo-class

includes pseudo-element

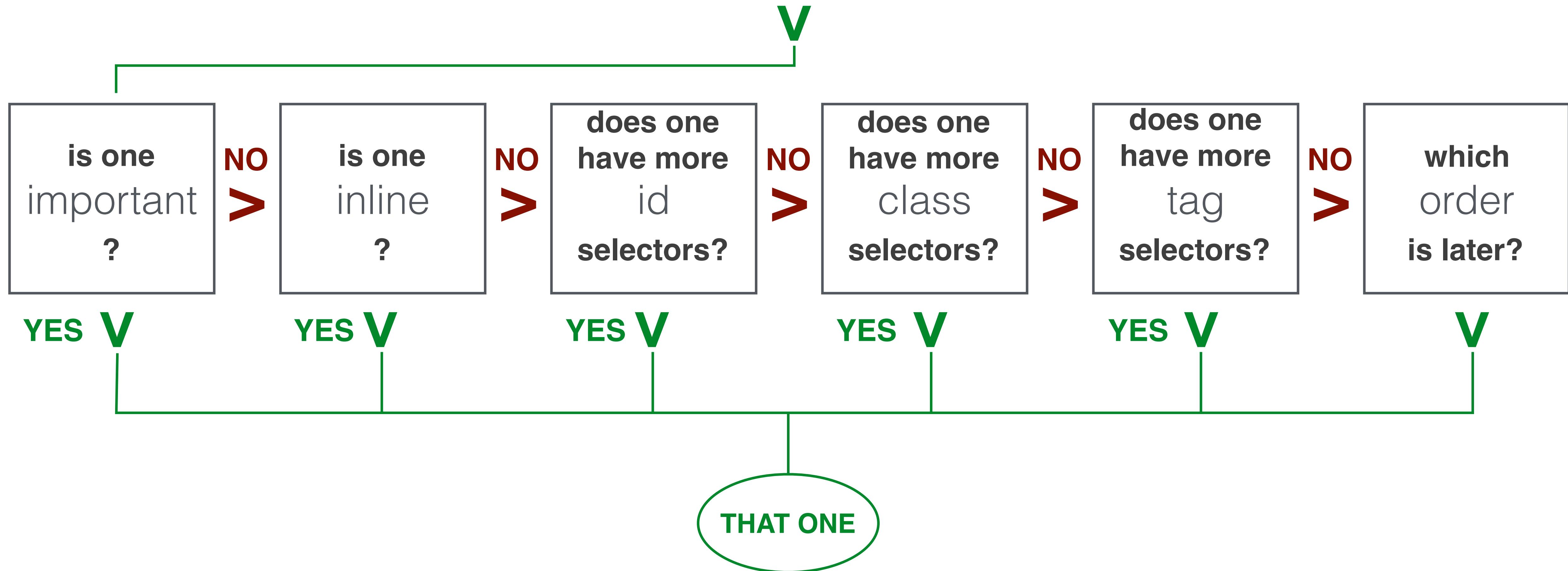


Combinators don't affect specificity!

# COMPARING TWO RULES



# COMPARING TWO RULES



# BROWSER COMPATIBILITY

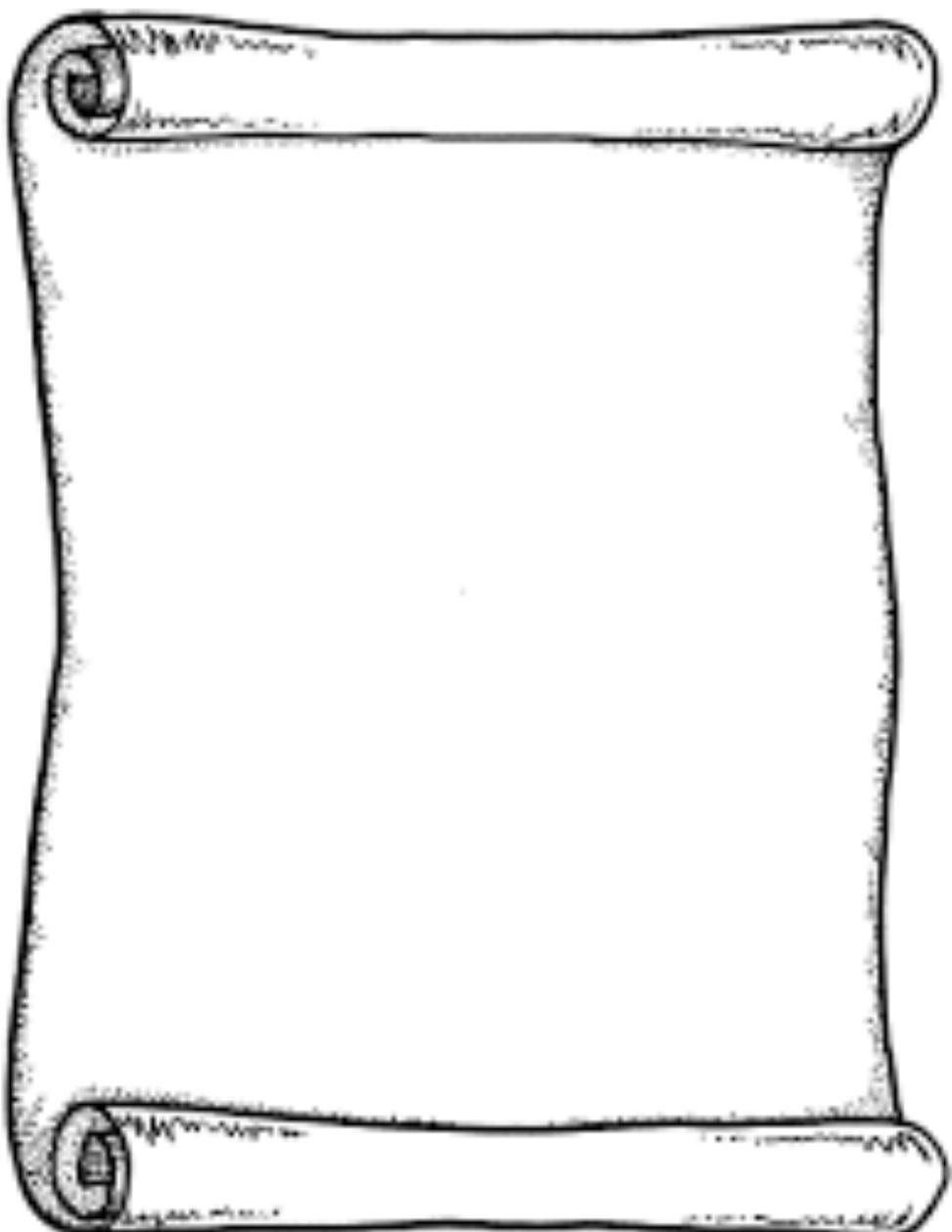
# BROWSER COMPATIBILITY

- CSS “resets”

# BROWSER COMPATIBILITY

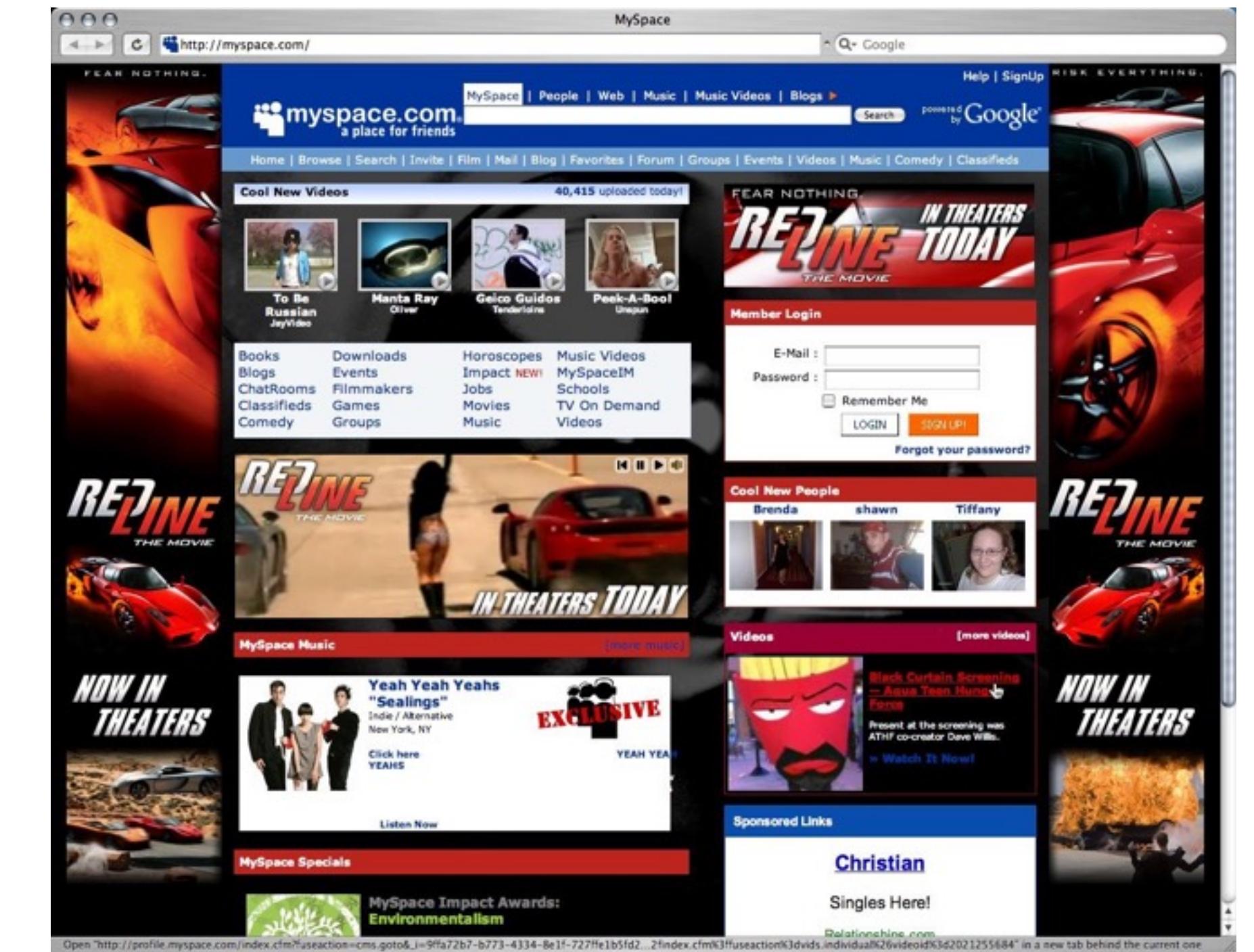
- CSS “resets”
- CSS normalization

# LAYOUT



<HTTP://WWW.CLIPARTBEST.COM/CLIPART-914055P6T>

[HTTP://ETC.USF.EDU/CLIPART/44800/44880/44880\\_GUTEN\\_PRESS\\_LG.GIF](HTTP://ETC.USF.EDU/CLIPART/44800/44880/44880_GUTEN_PRESS_LG.GIF)



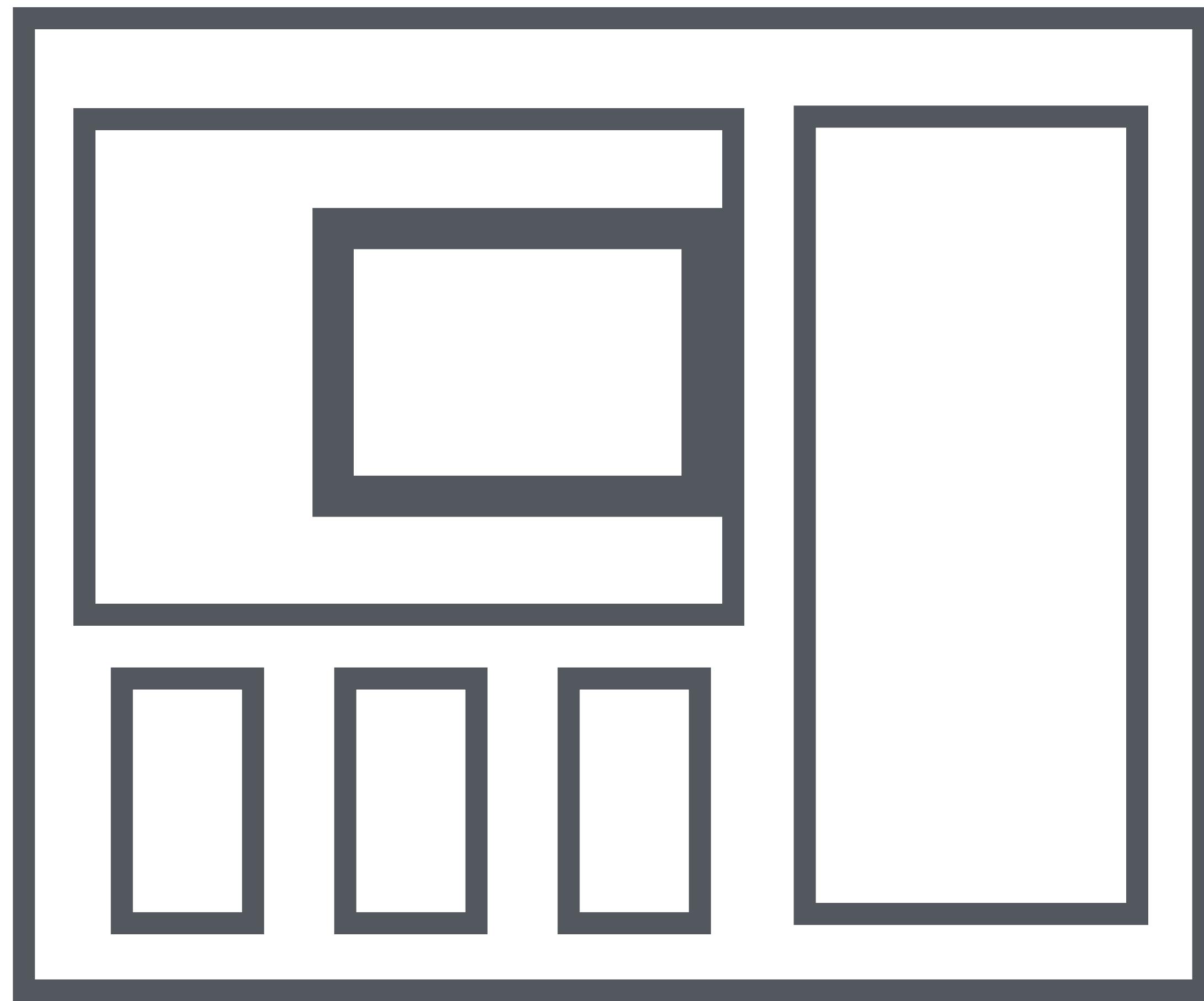
[HTTP://WWW.UIE.COM/IMAGES/BLOG/MYSPACE-HOME PAGE-04-13-2007.JPG](HTTP://WWW.UIE.COM/IMAGES/BLOG/MYSSPACE-HOME PAGE-04-13-2007.JPG)

# THE BOX MODEL

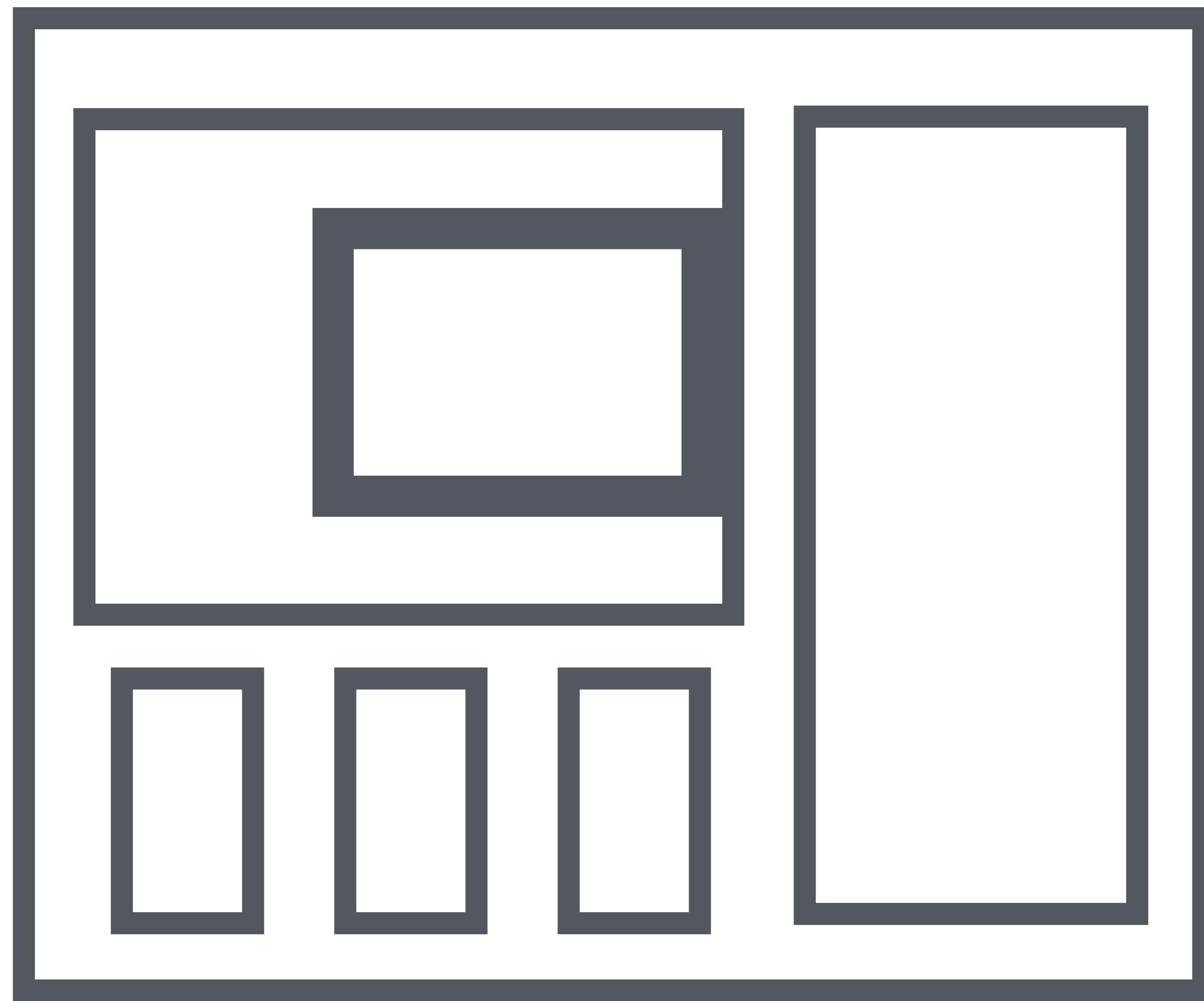
*“Everything on the DOM is a rectangle”*

JOE ALVES

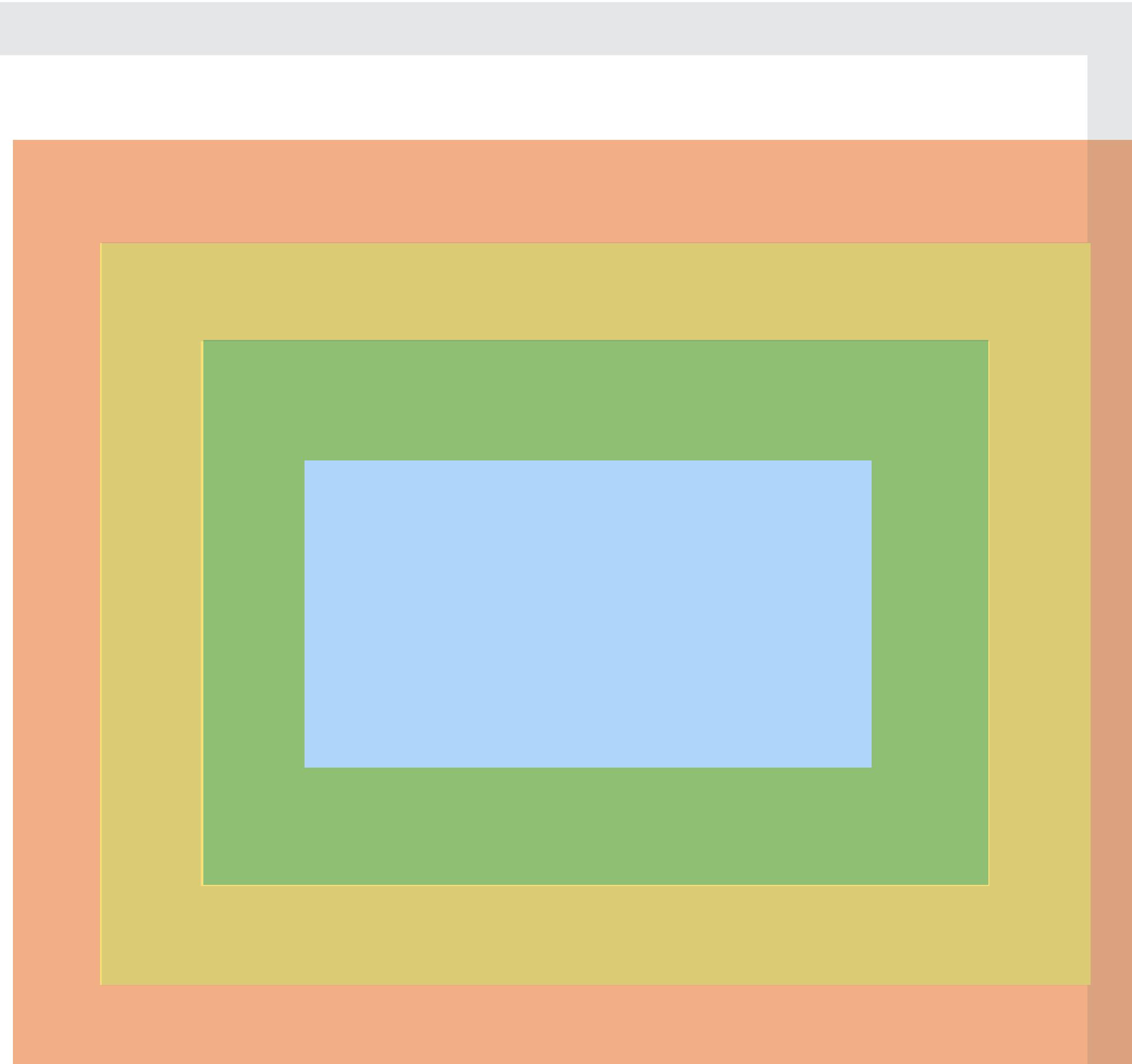
# BOX MODEL



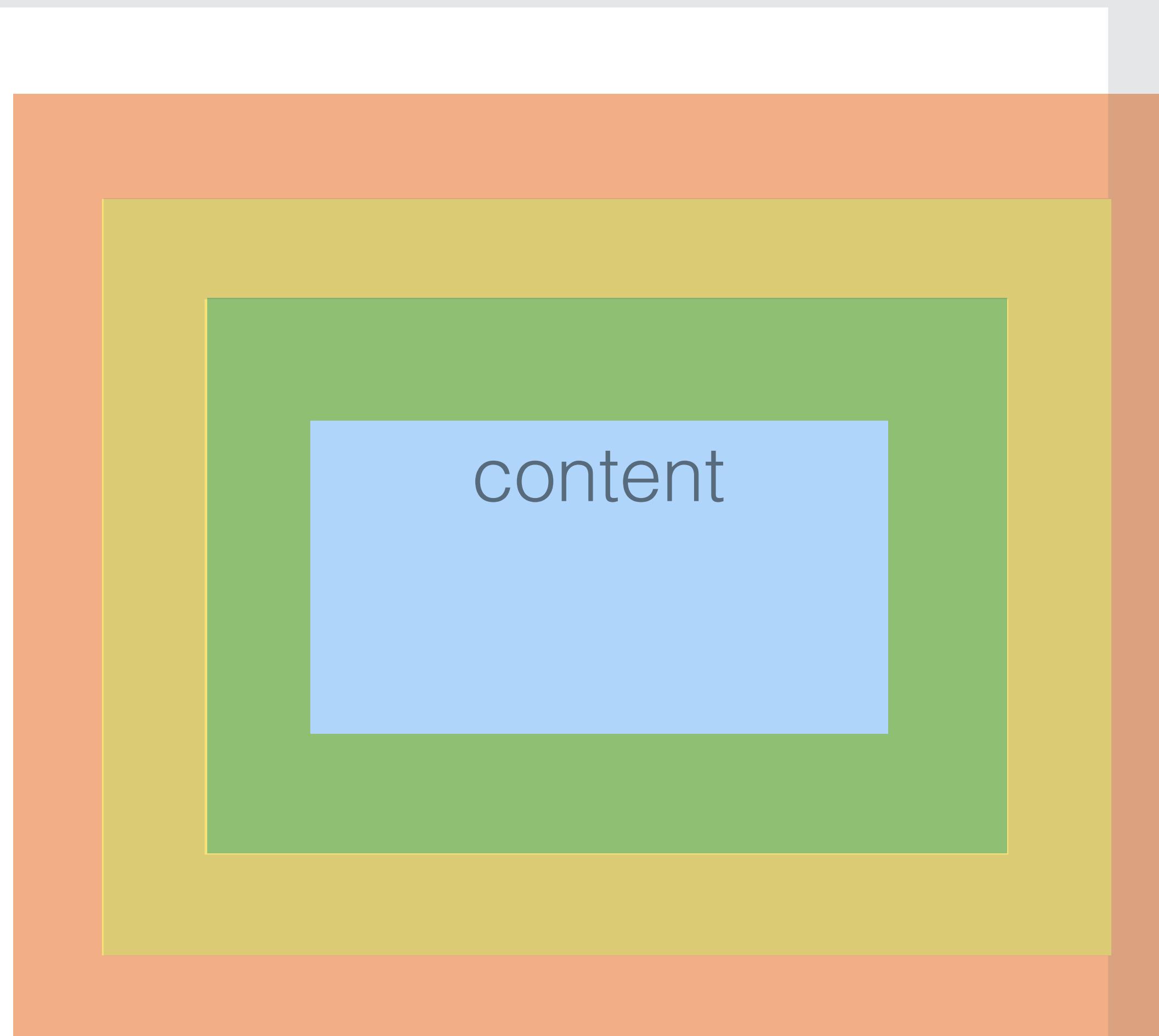
# BOX MODEL



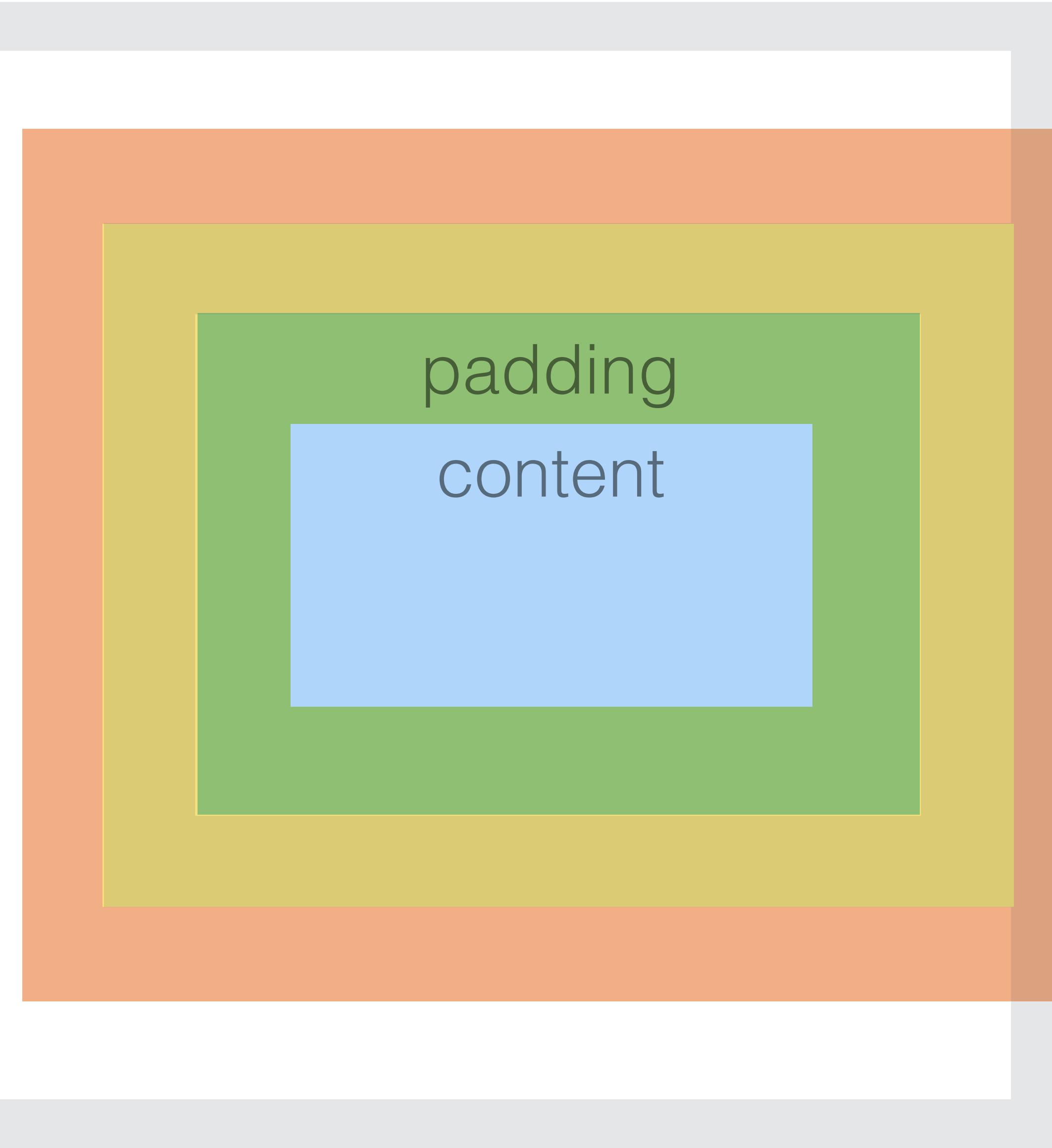
# BOX MODEL



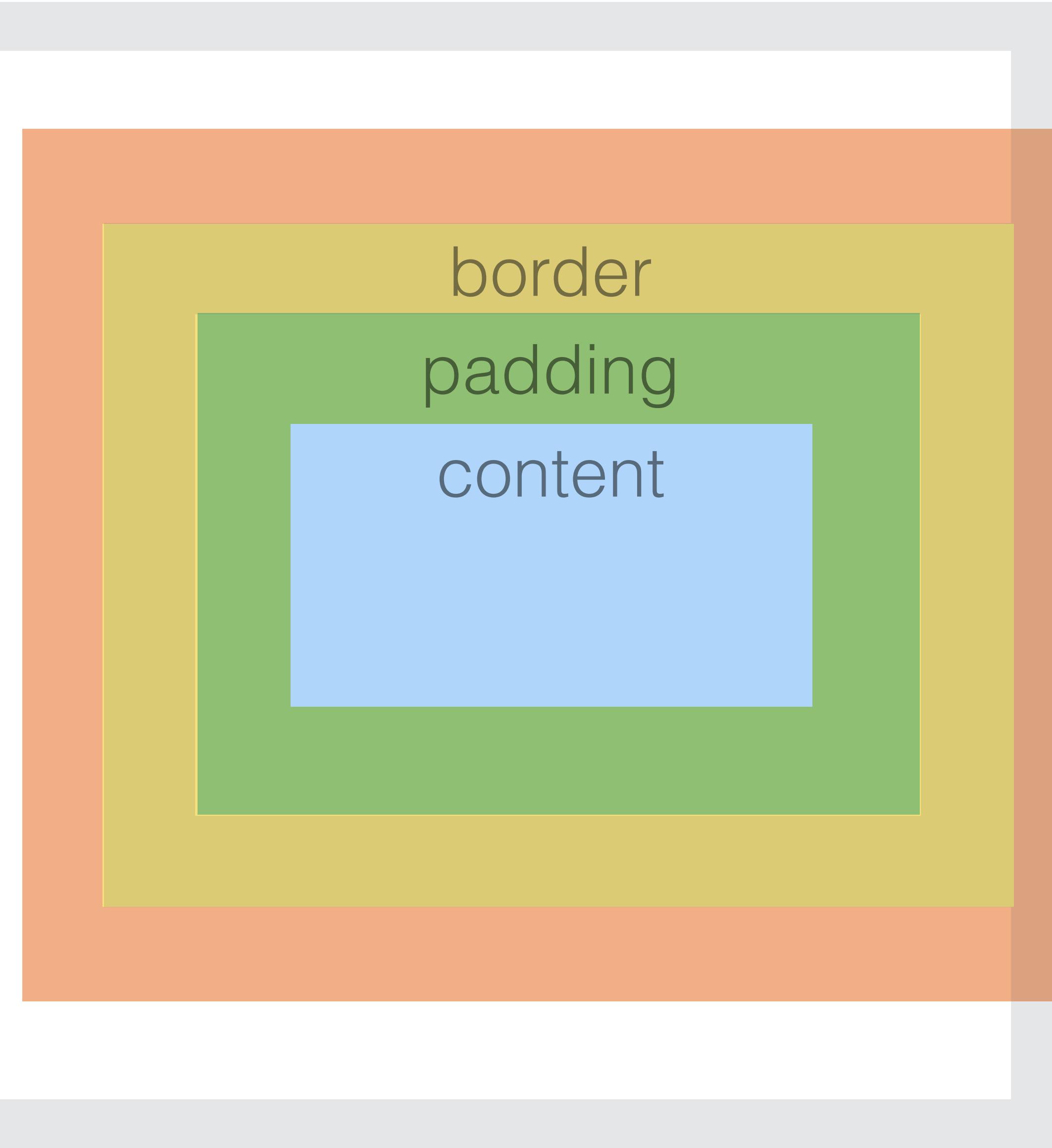
# BOX MODEL



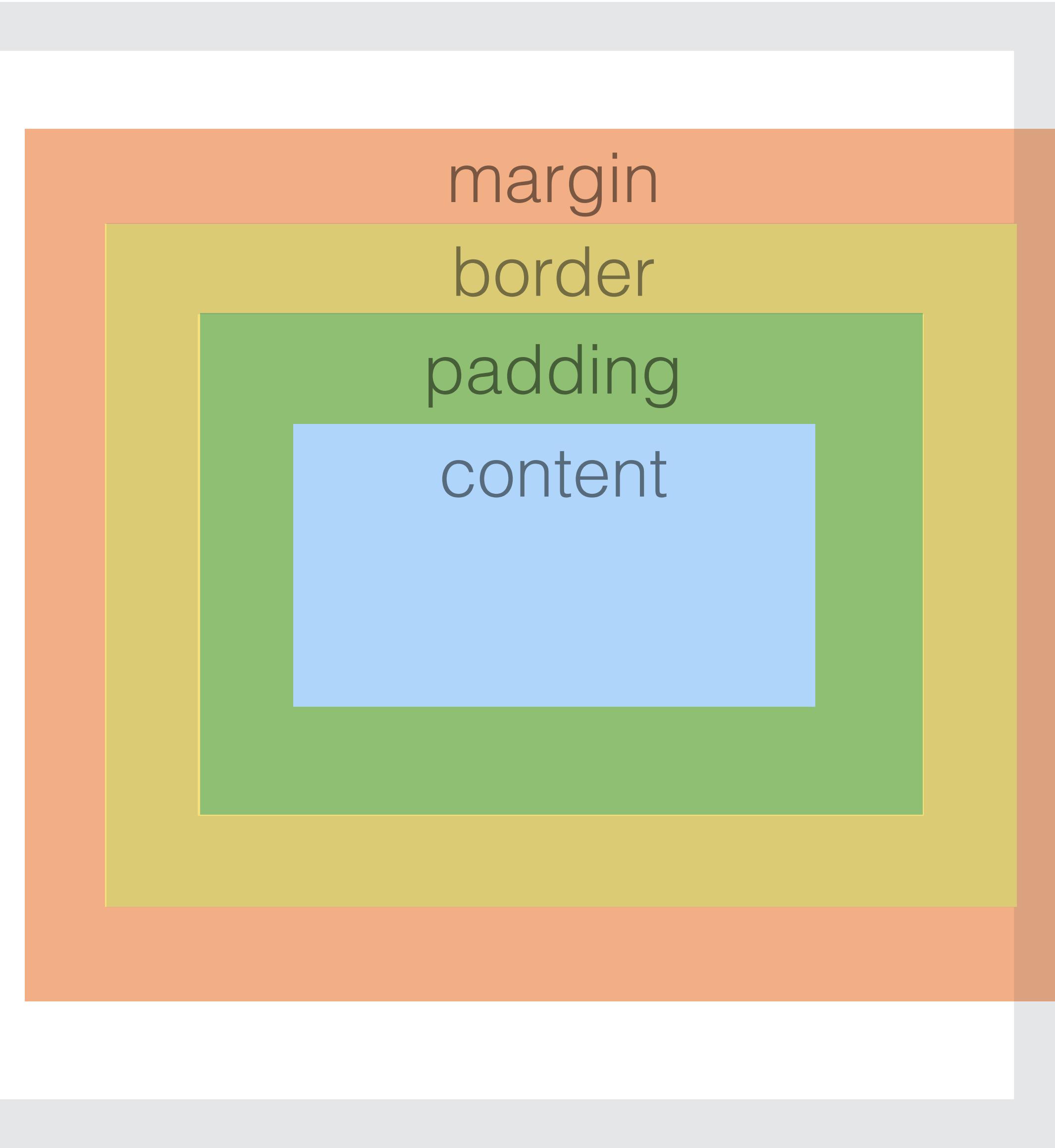
# BOX MODEL



# BOX MODEL



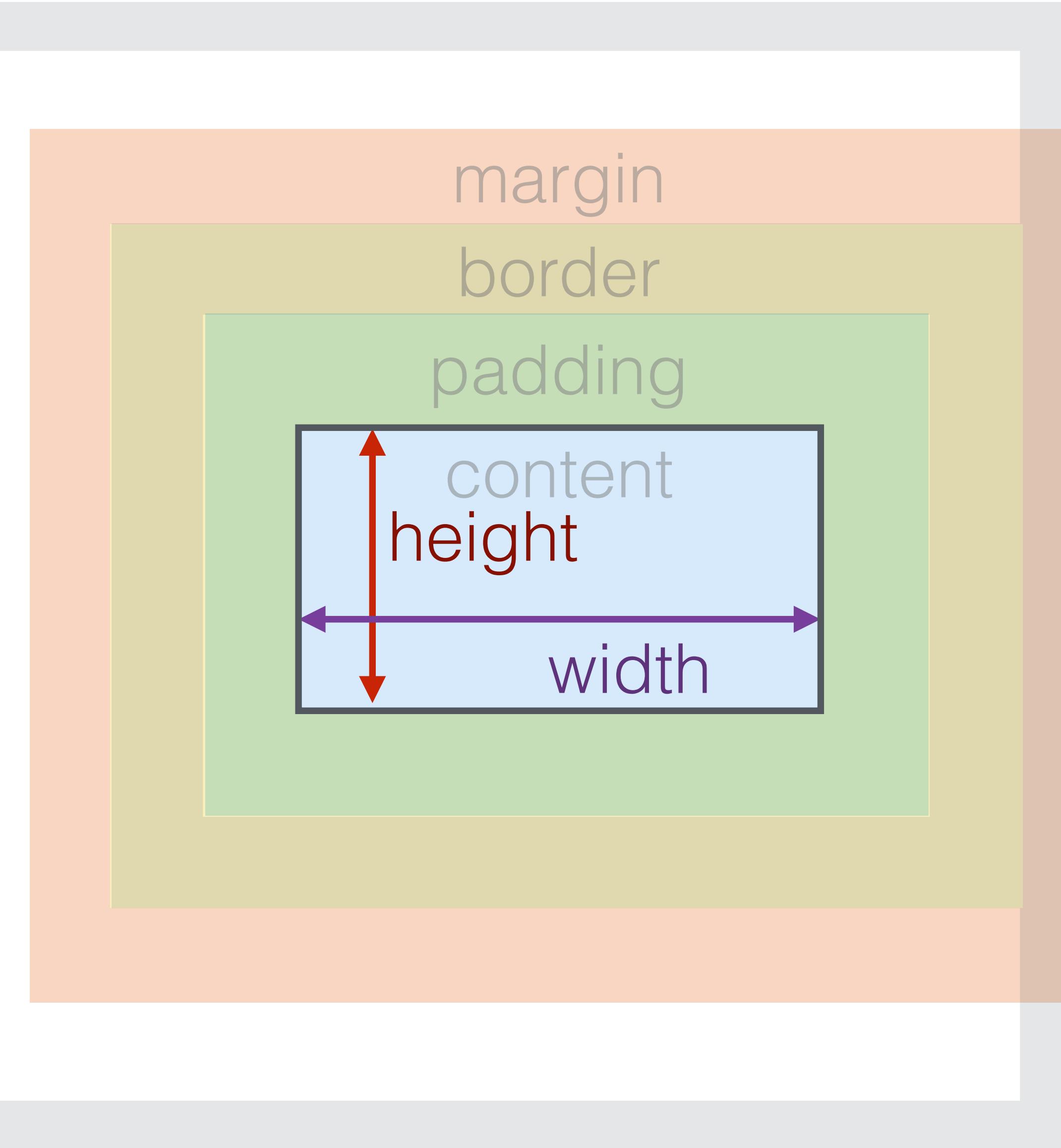
# BOX MODEL



# BOX MODEL

**content box**

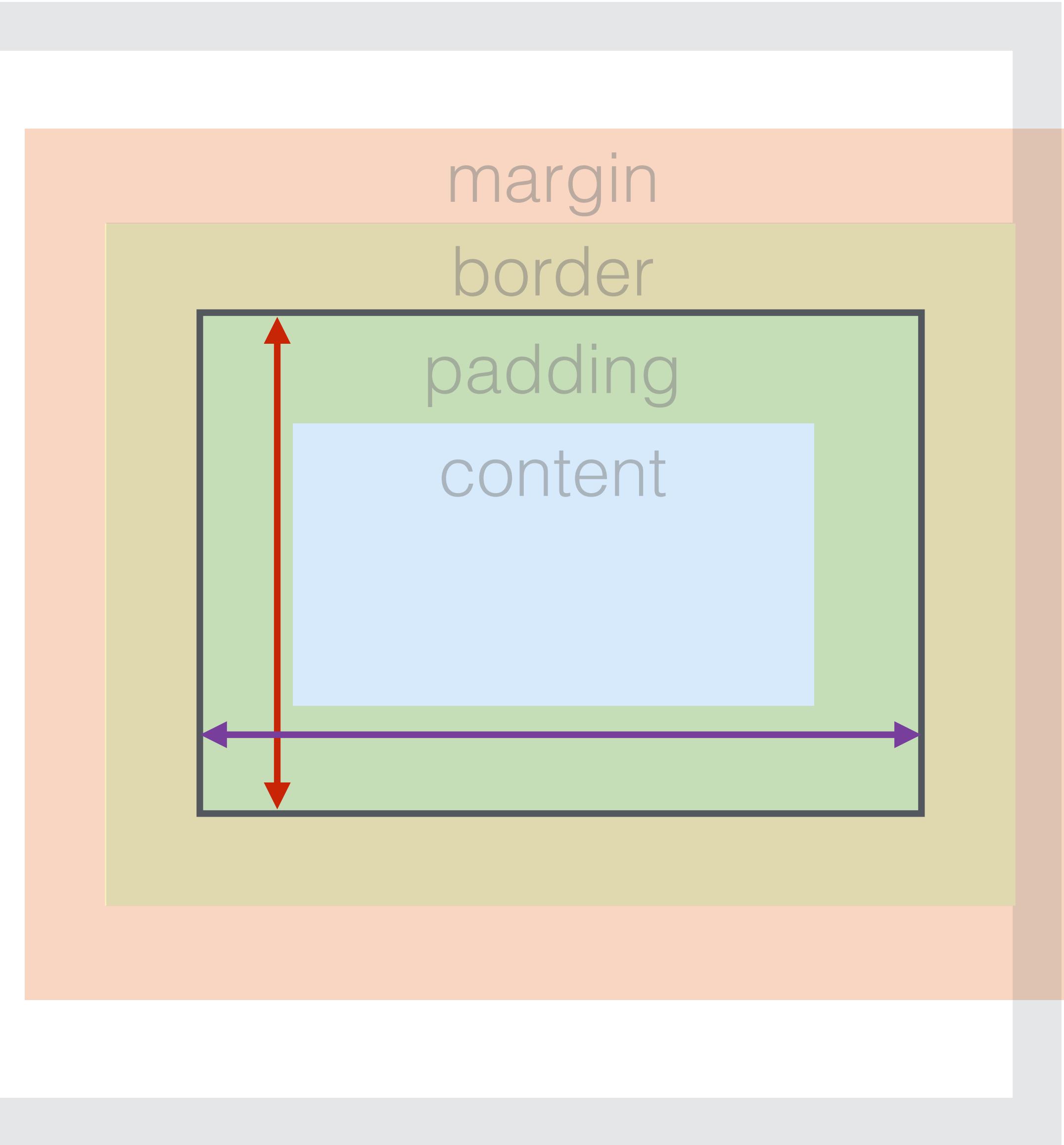
```
{  
  box-sizing: content-box;  
}
```



# BOX MODEL

**padding box**

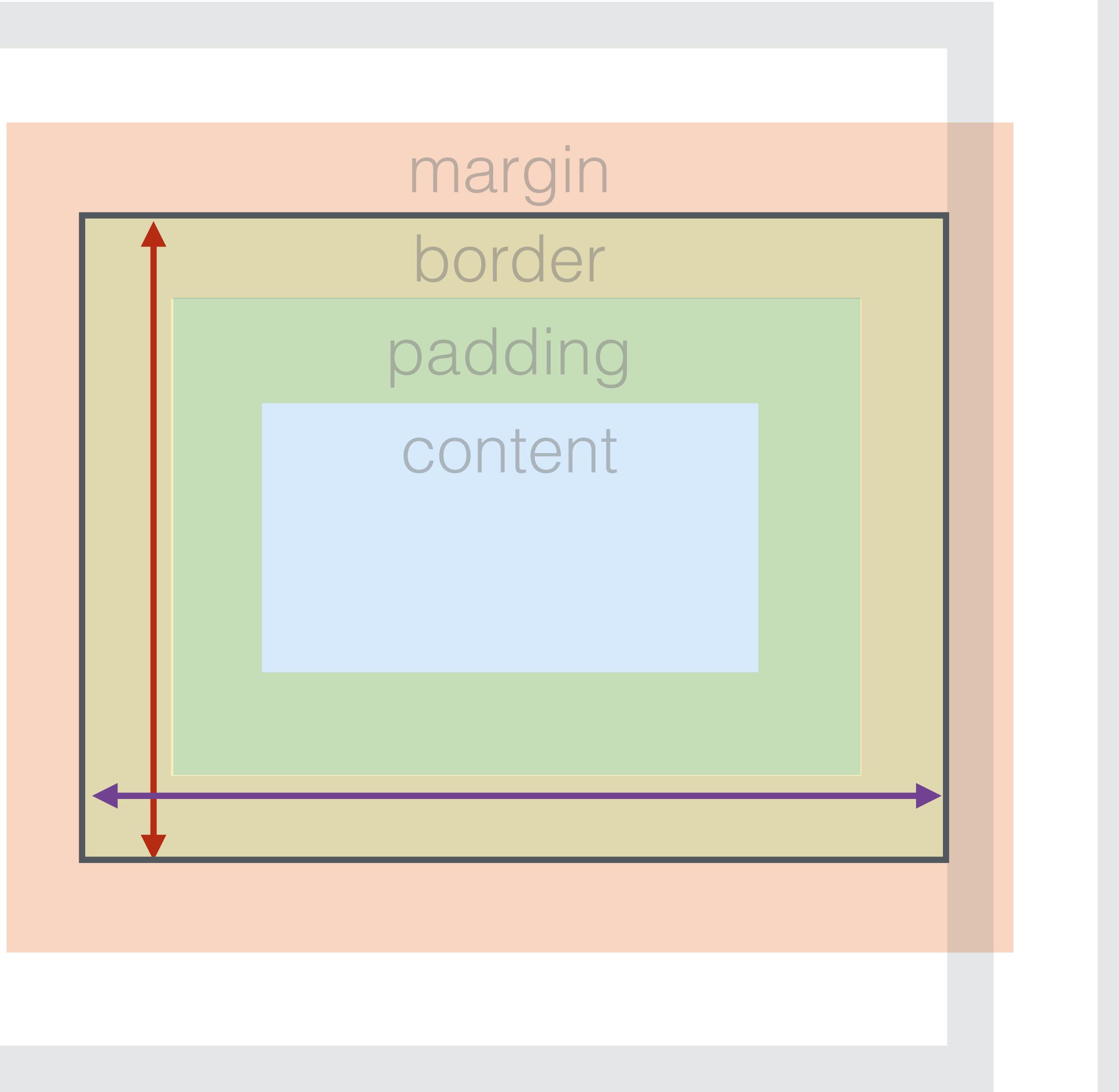
```
{  
  box-sizing: padding-box;  
}
```



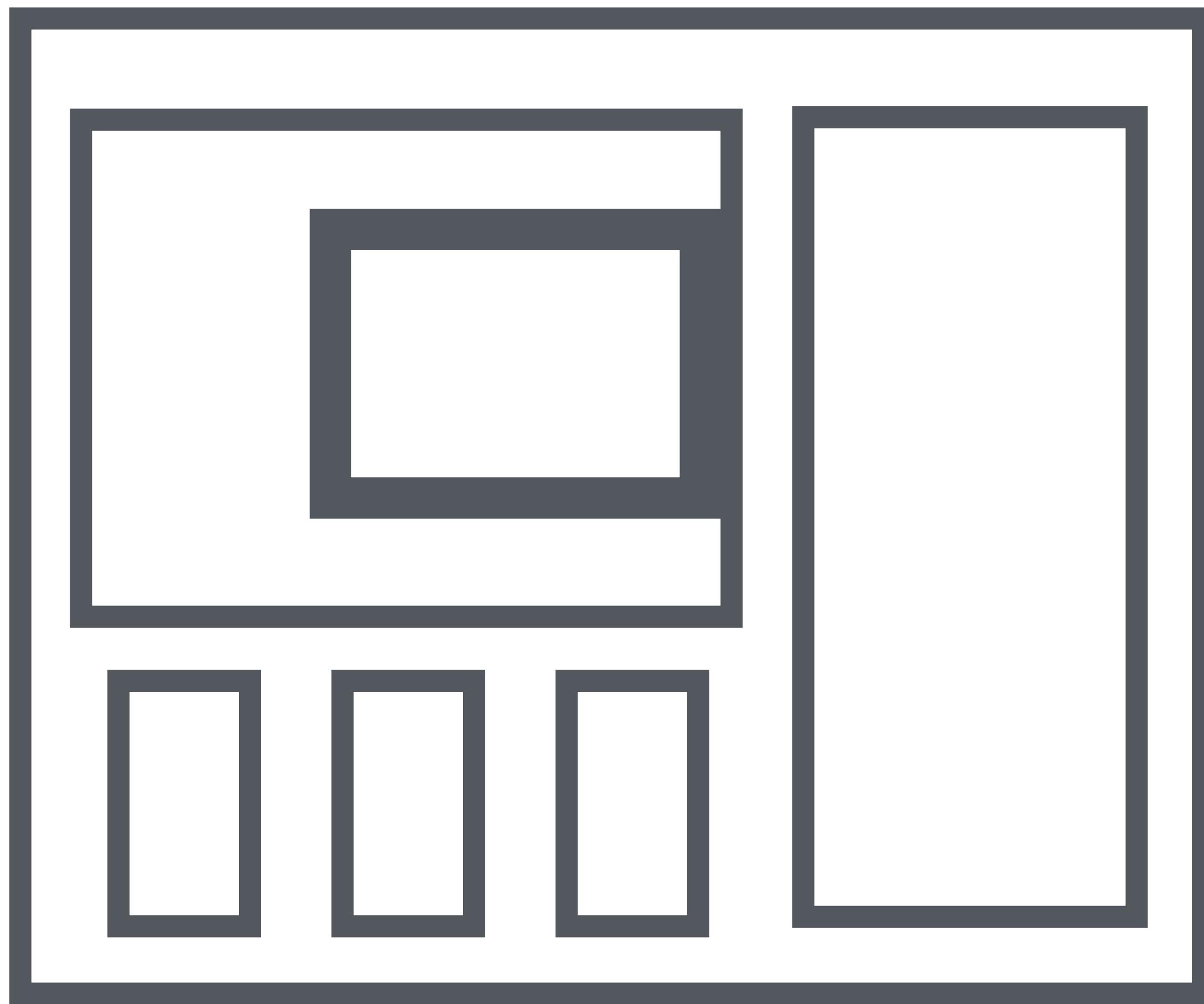
# BOX MODEL

**border box**

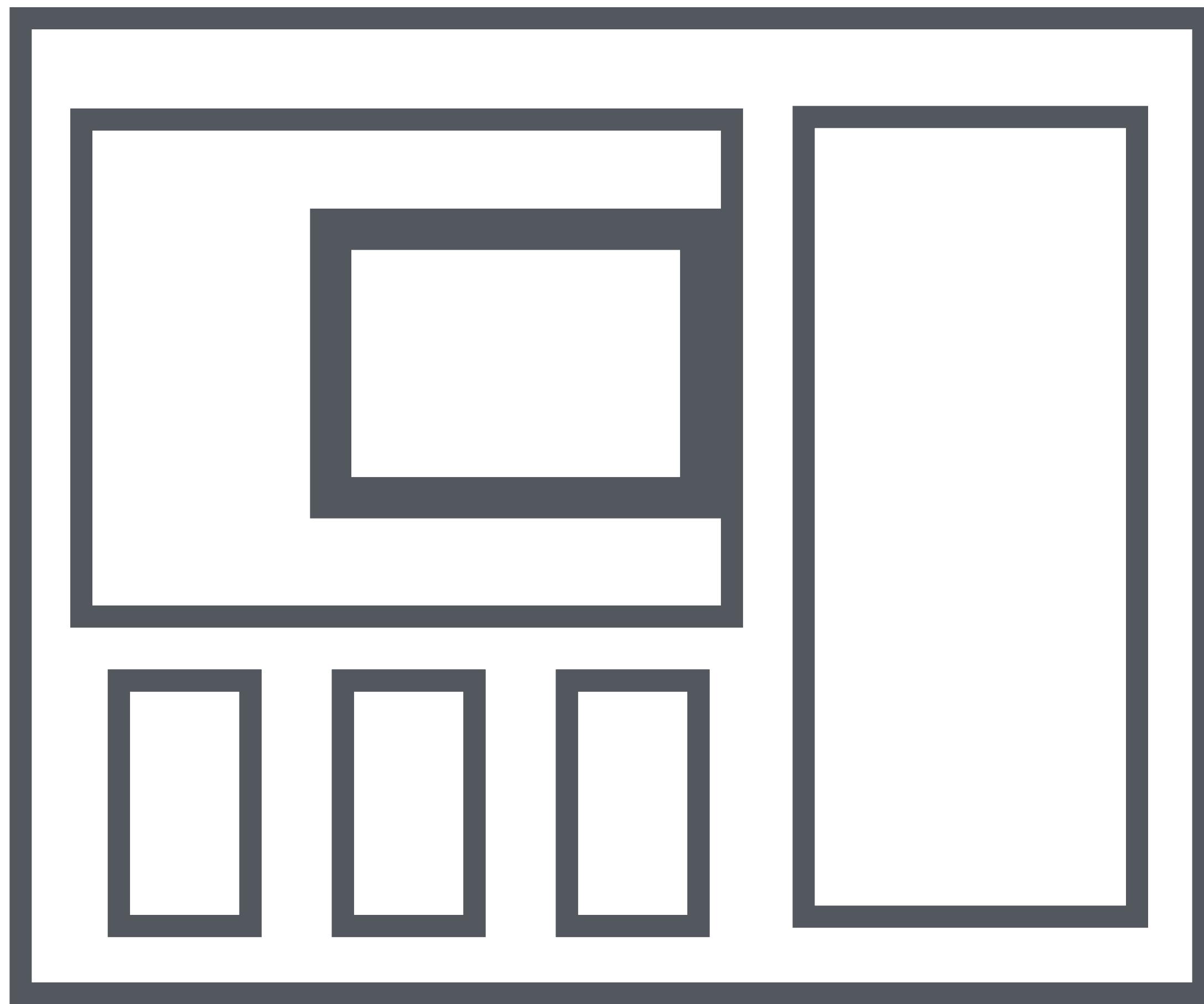
```
{  
  box-sizing: border-box;  
}
```



# BOX MODEL

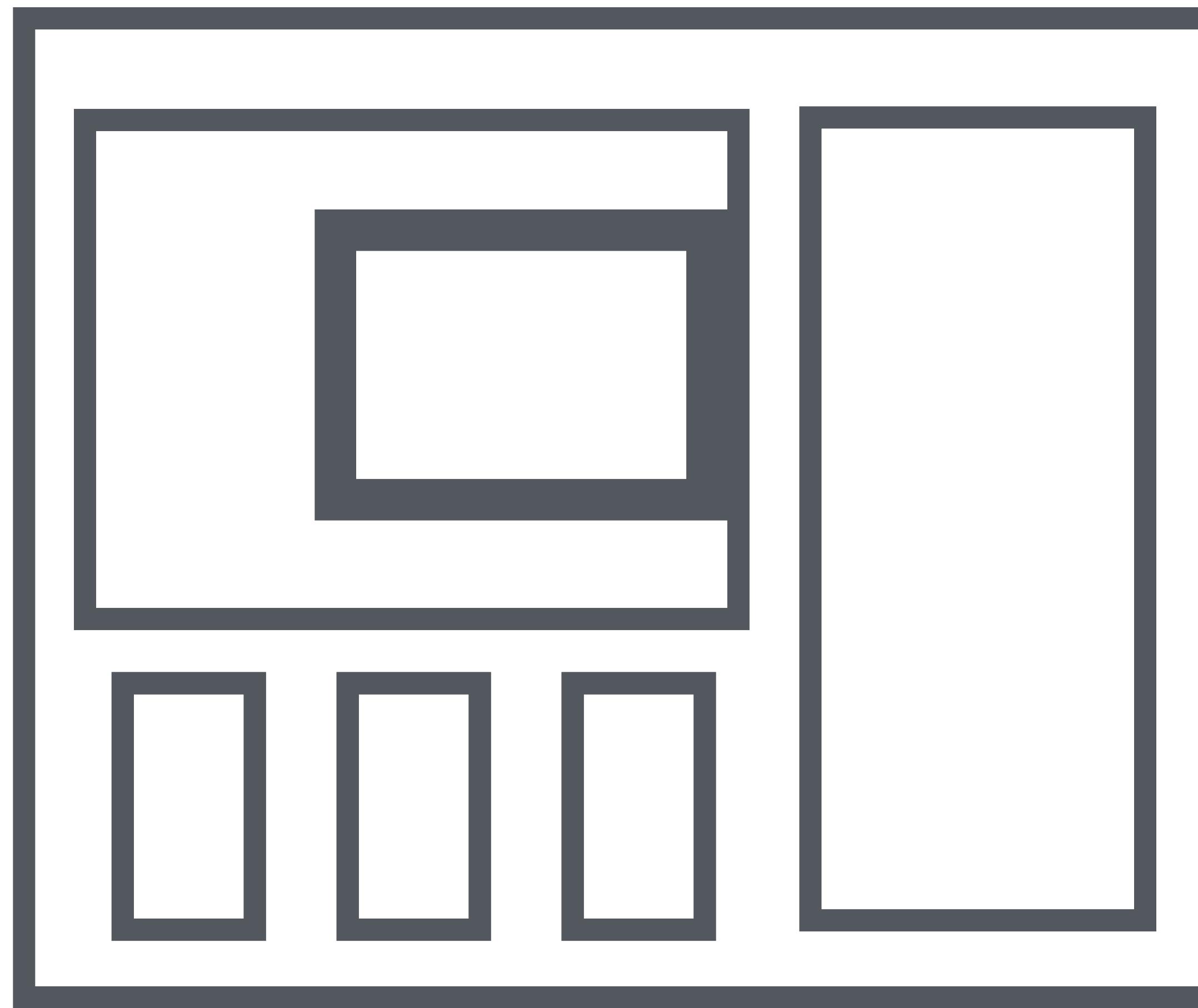


# BOX MODEL



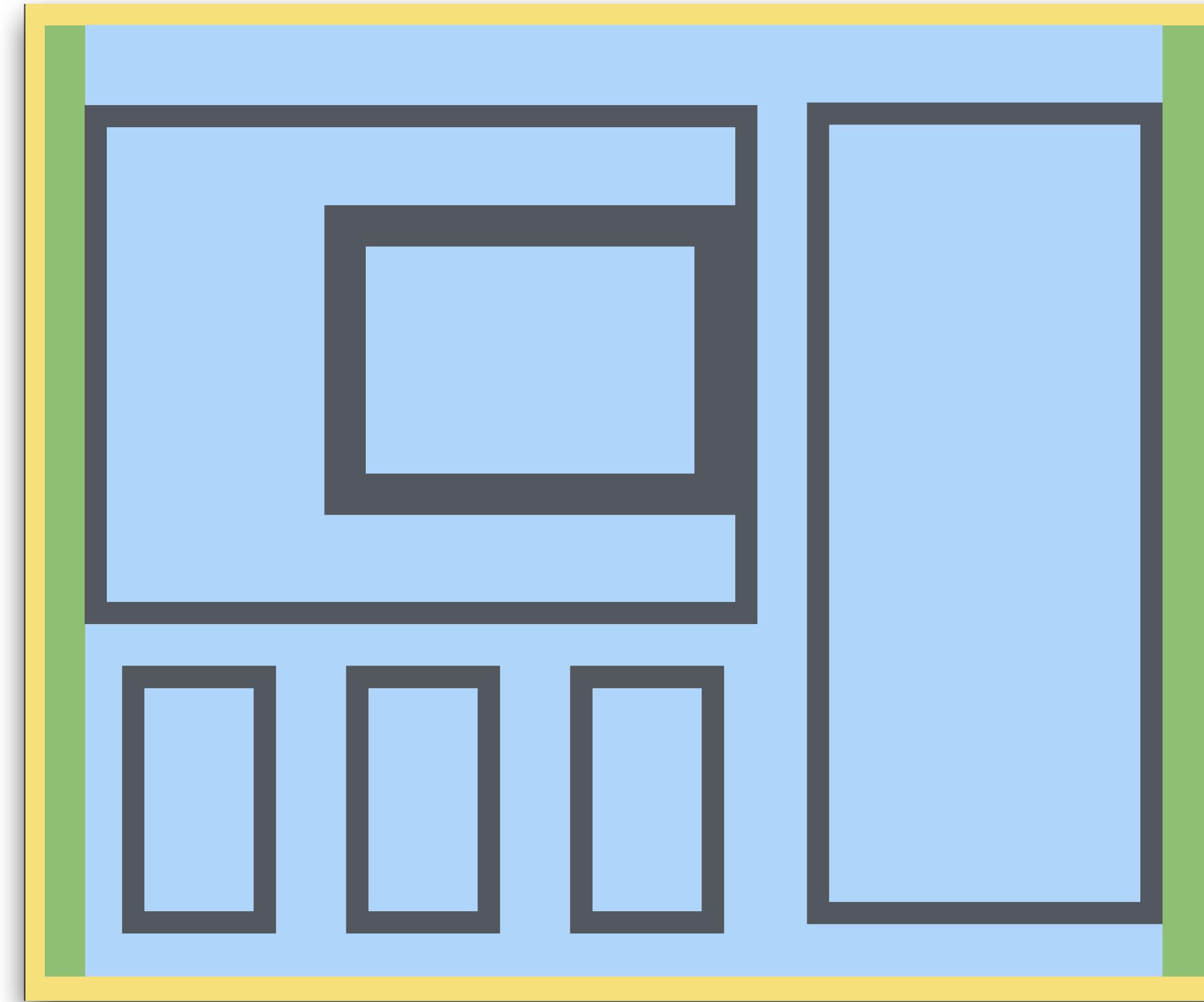
# BOX MODEL

**fractal!**



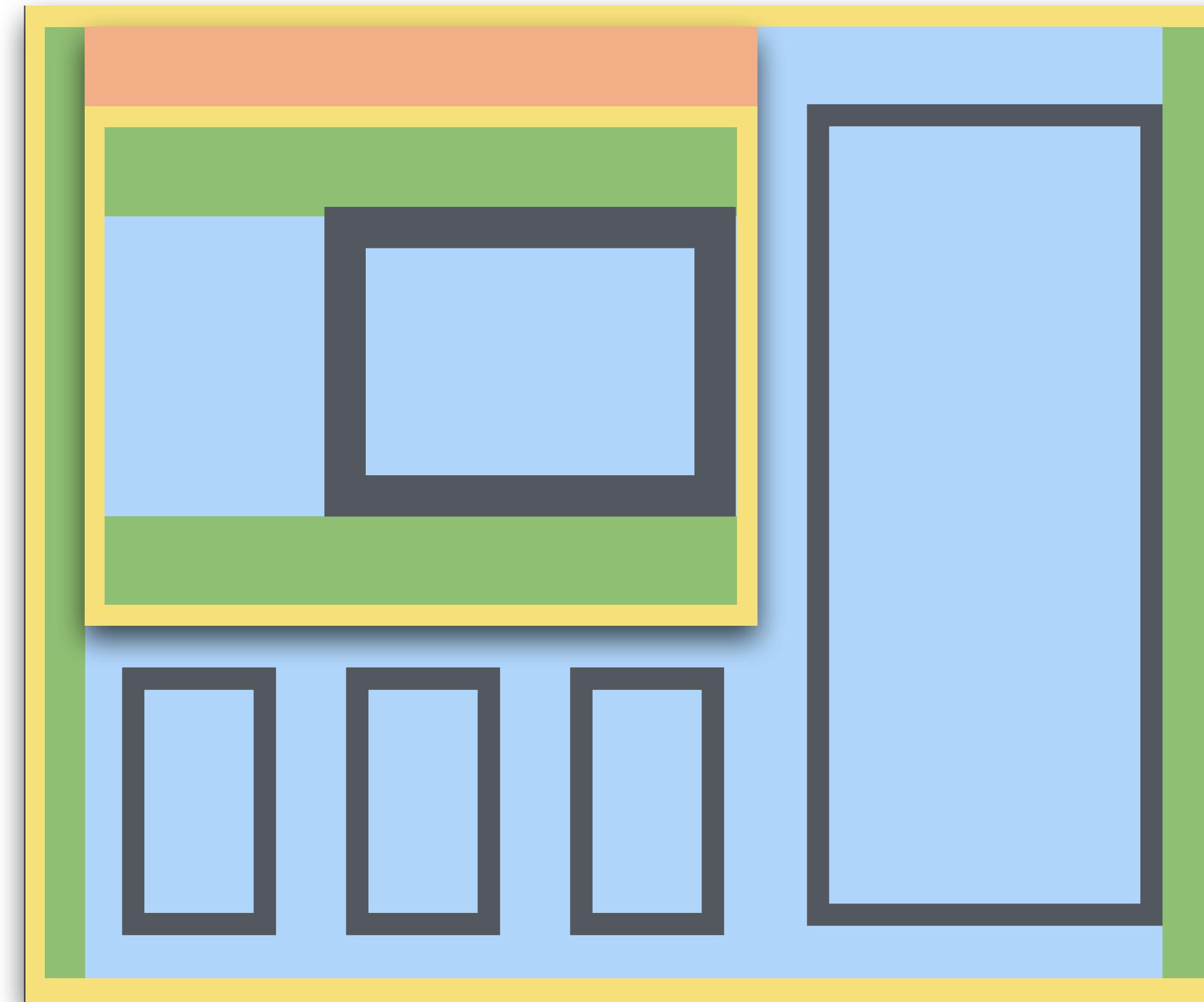
# BOX MODEL

**fractal!**



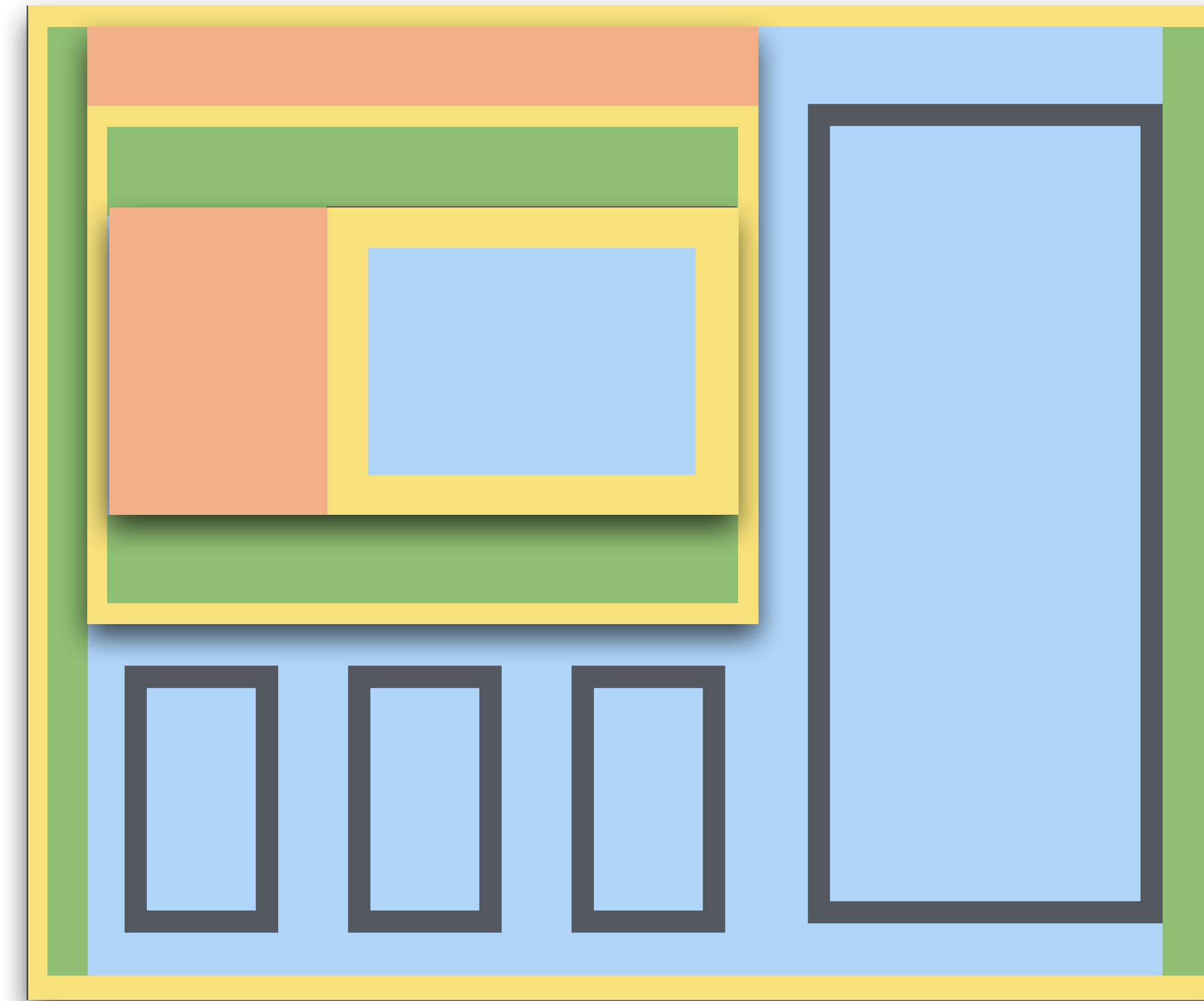
# BOX MODEL

**fractal!**



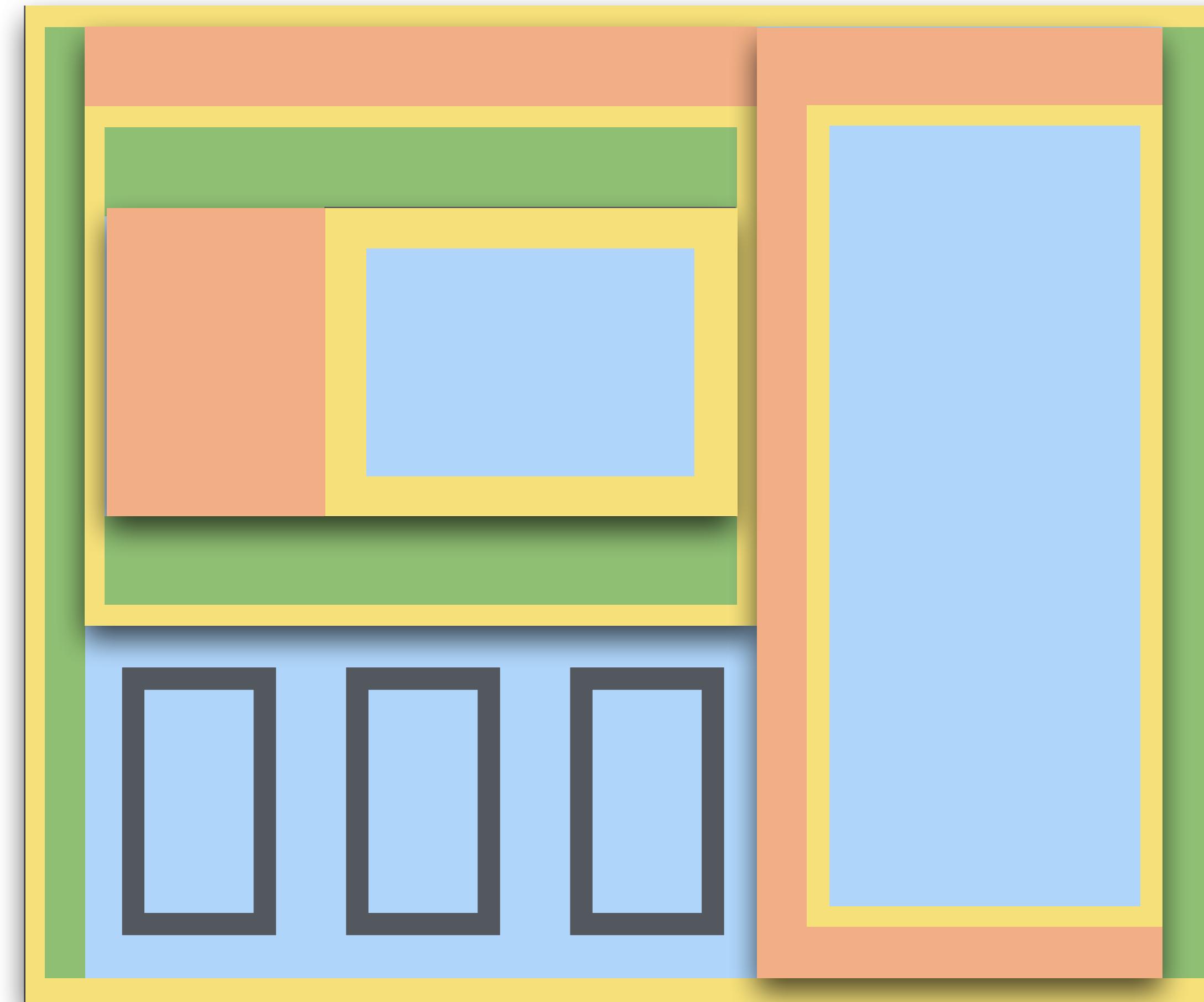
# BOX MODEL

**fractal!**



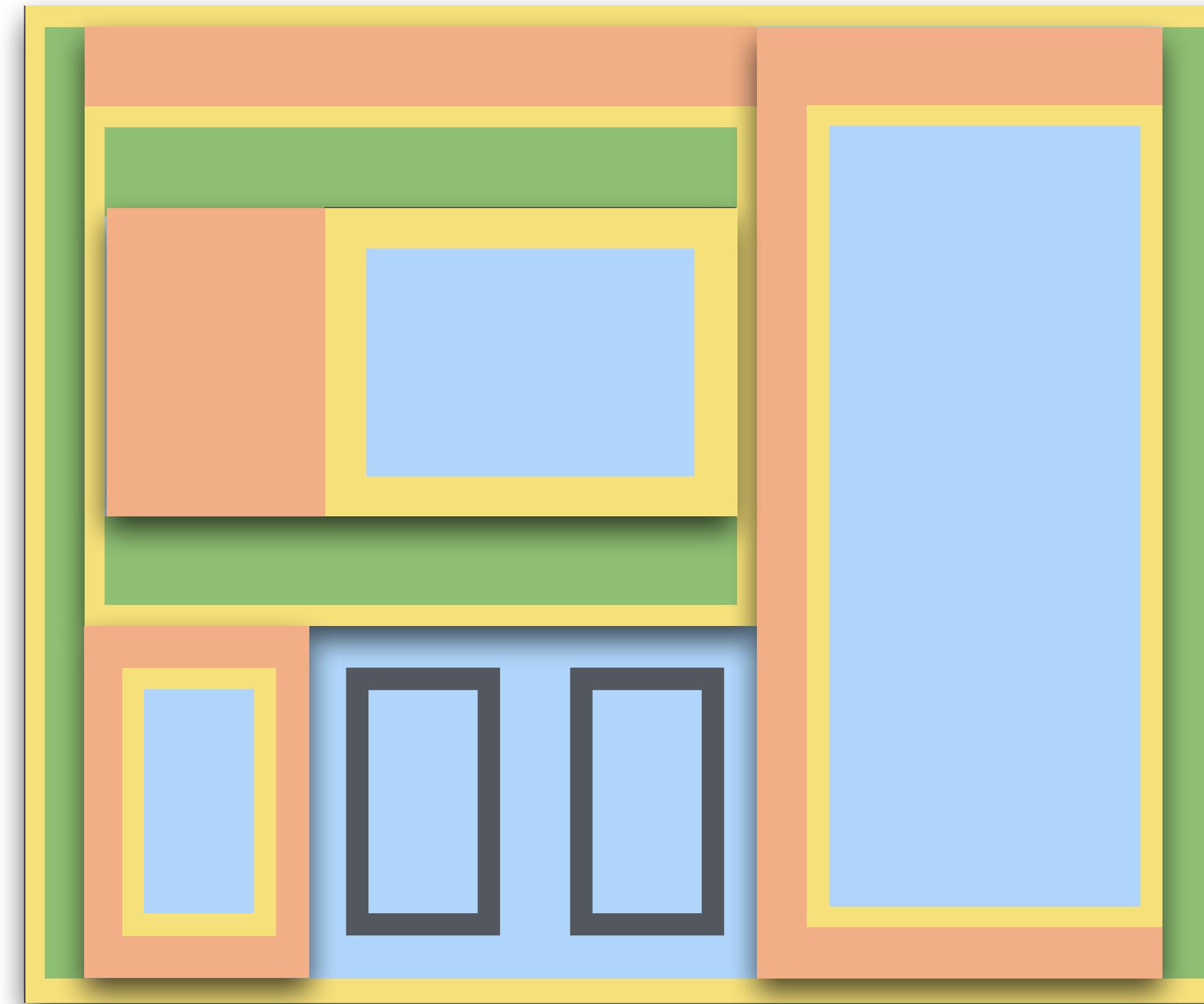
# BOX MODEL

**fractal!**



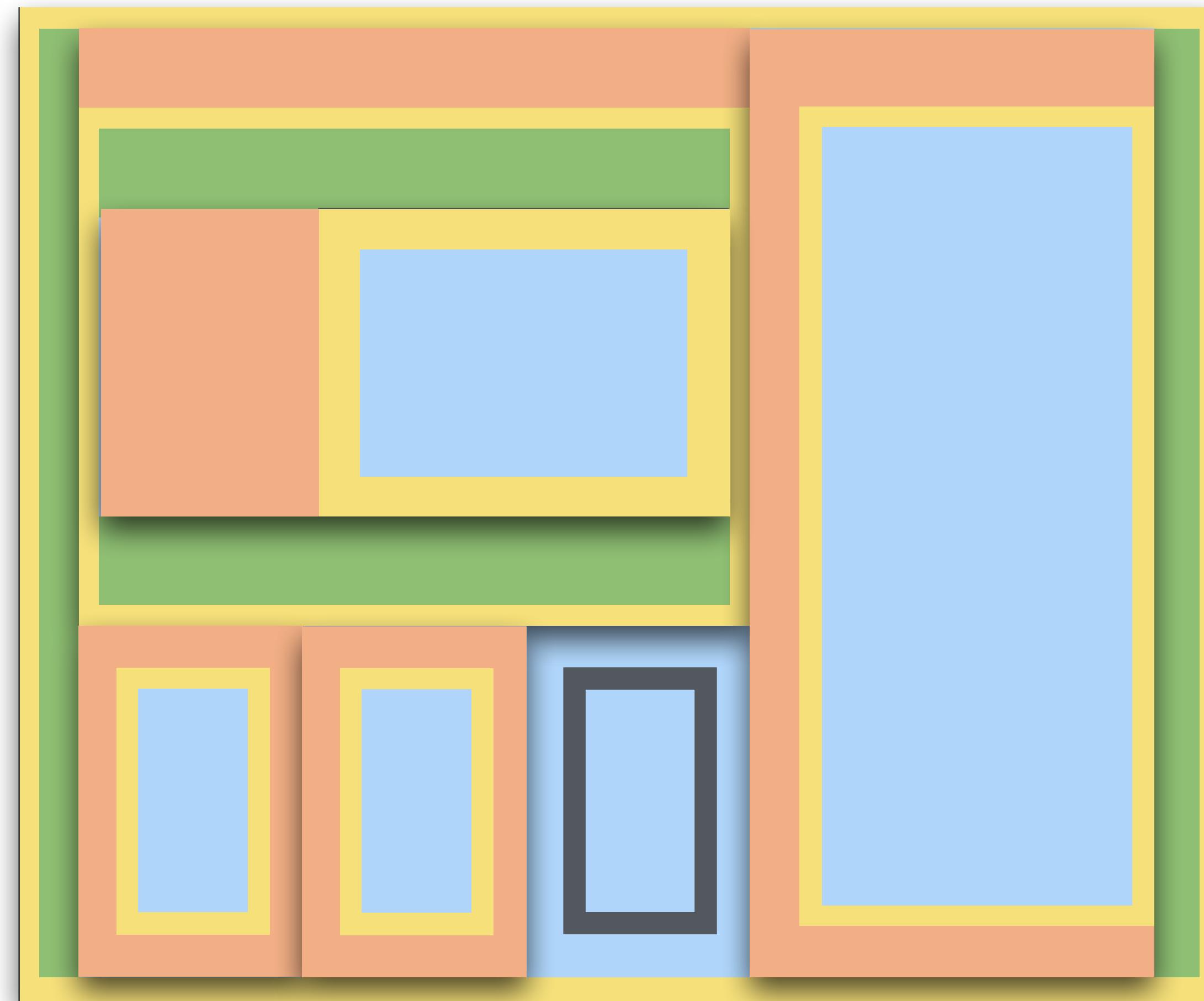
# BOX MODEL

**fractal!**



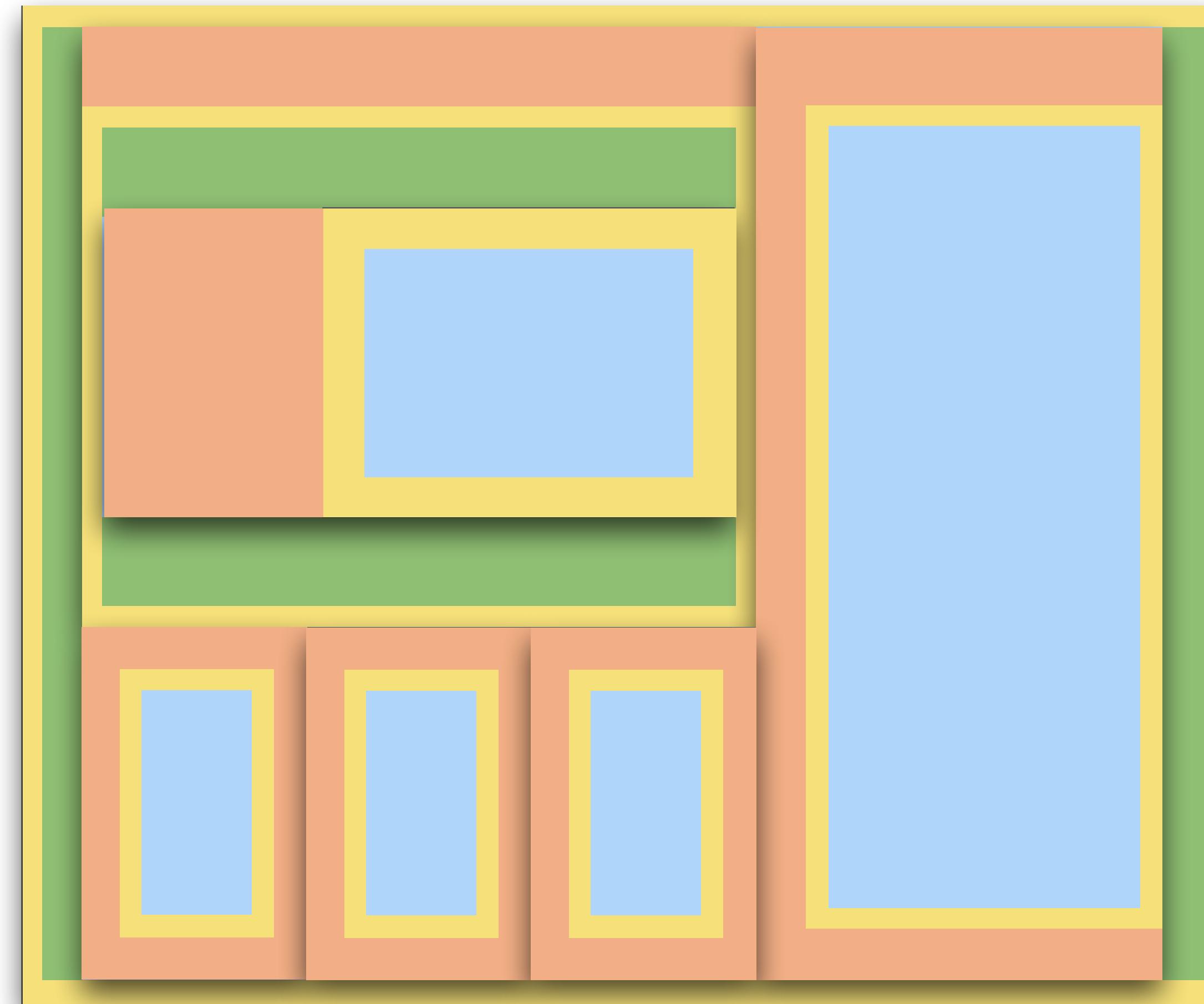
# BOX MODEL

**fractal!**



# BOX MODEL

**fractal!**



# BLOCK

vs

# INLINE

- **<div>**
- **<h1>, <h2>, etc.**
- **<p>**
- **<form>**
- **<header>, <footer>,  
<main>, <section>, <nav>**
- **<a>**
- **<span>**
- **<strong>, <em>**

# BLOCK

vs

# INLINE

vs

# INLINE-BLOCK

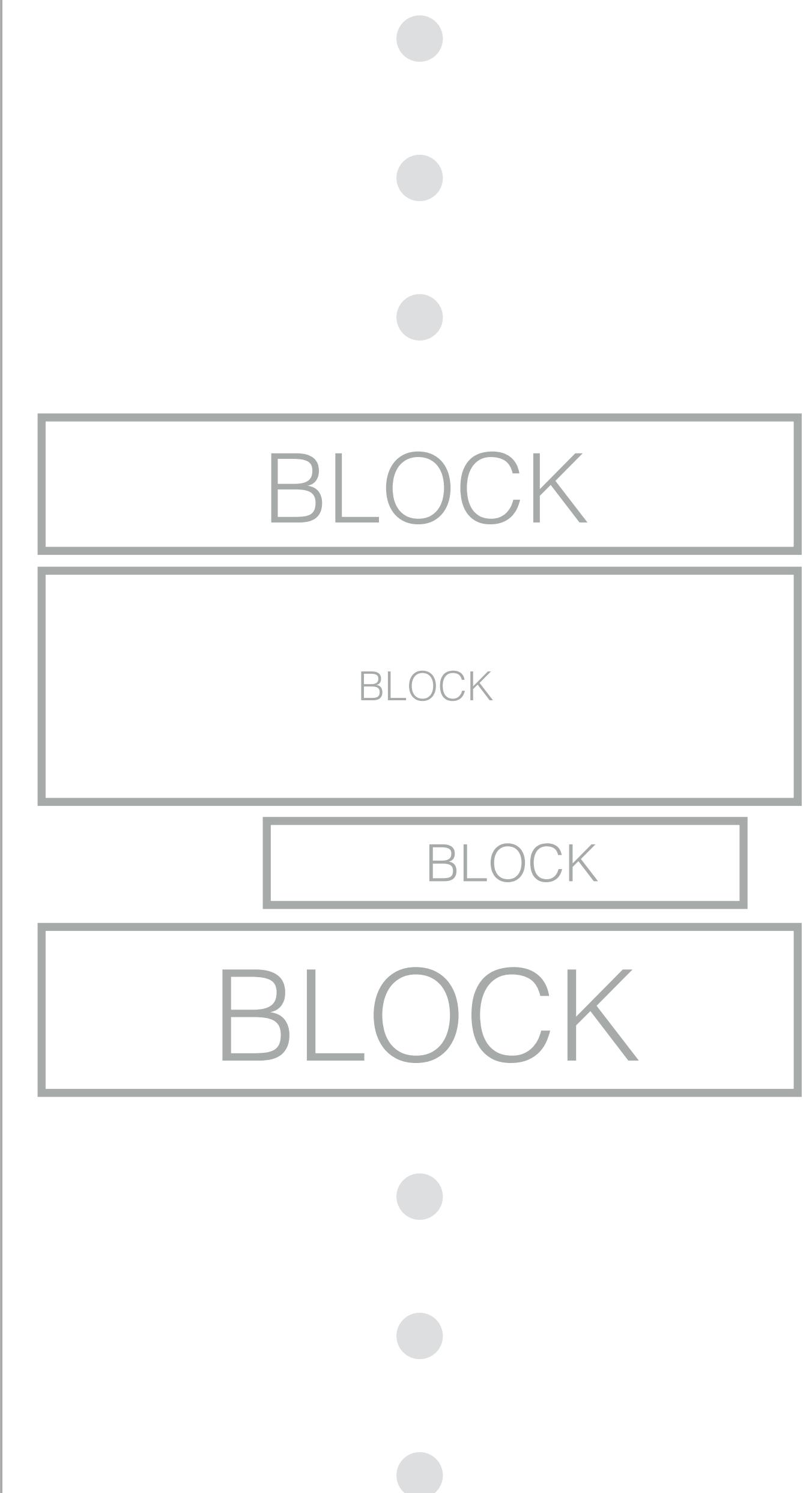
- **<div>**
- **<h1>, <h2>, etc.**
- **<p>**
- **<form>**
- **<header>, <footer>,  
<main>, <section>, <nav>**

- **<a>**
- **<span>**
- **<strong>, <em>**

- **<img>**
- **<input>**
- **<textarea>**

# BLOCK-LEVEL

- By default will try to **clear** their own line
- Default width is 100% of parent
- Default height will expand to fit all children
- Can have margins on all sides
- Can set height and width



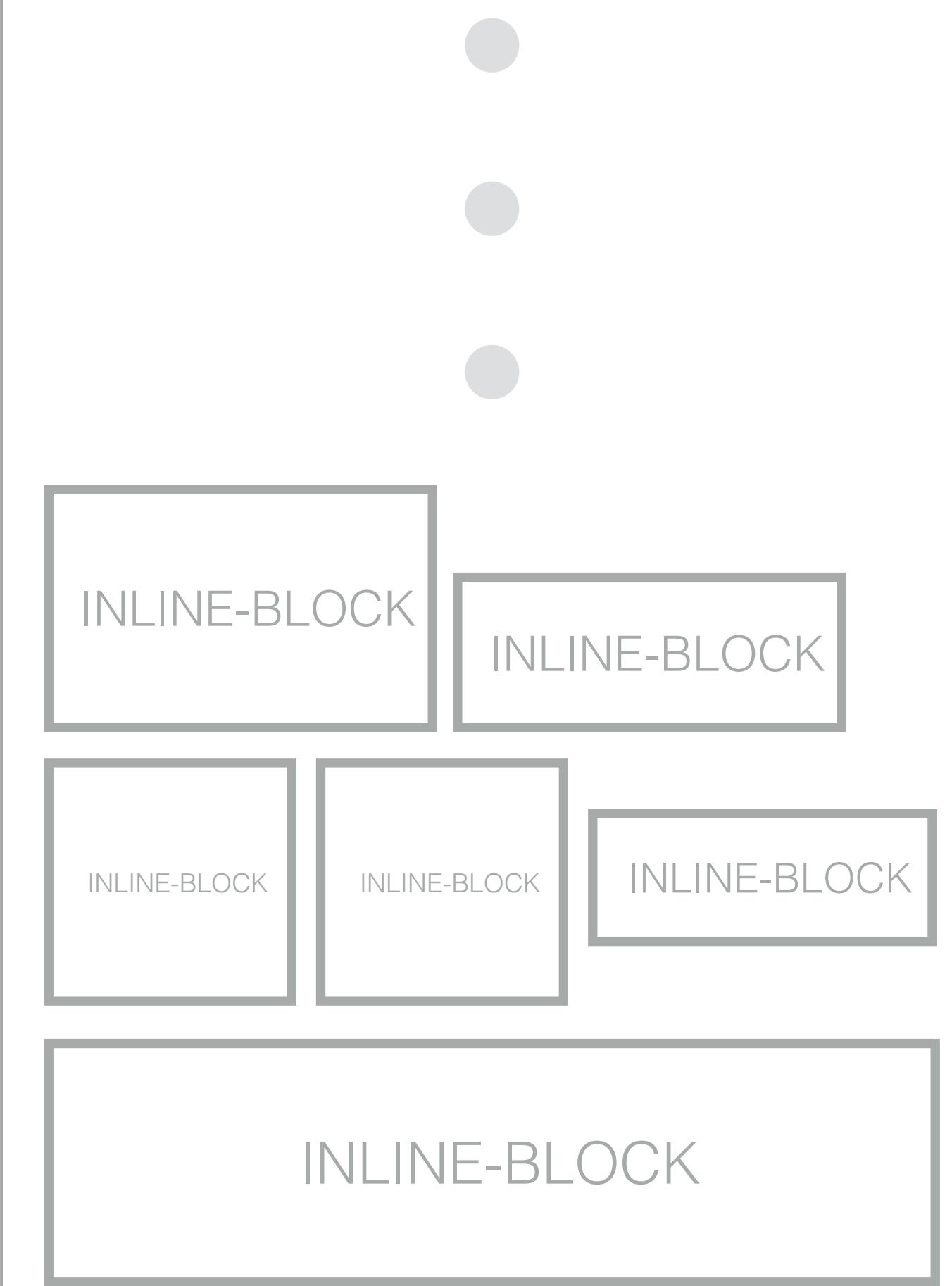
# INLINE-LEVEL

- Flows with content (does **not** clear line)
- Ignores top and bottom margins
- Height and width fit content
- Cannot set height or width



# INLINE-BLOCK

- Flows with content (does **not** clear line)
- Default width will expand to fit all children
- Default height will expand to fit all children
- Can have margins on all sides
- Can set height and width



# UNITS

# UNITS

- px

# UNITS

- px**
- %**

# UNITS

- px**
- %**
- em**

# UNITS

- px**
- %**
- em**
- rem**

# UNITS

- px**
- %**
- em**
- rem**
- vw**

# UNITS

- px**
- %**
- em**
- rem**
- vw**
- vh**

# UNITS

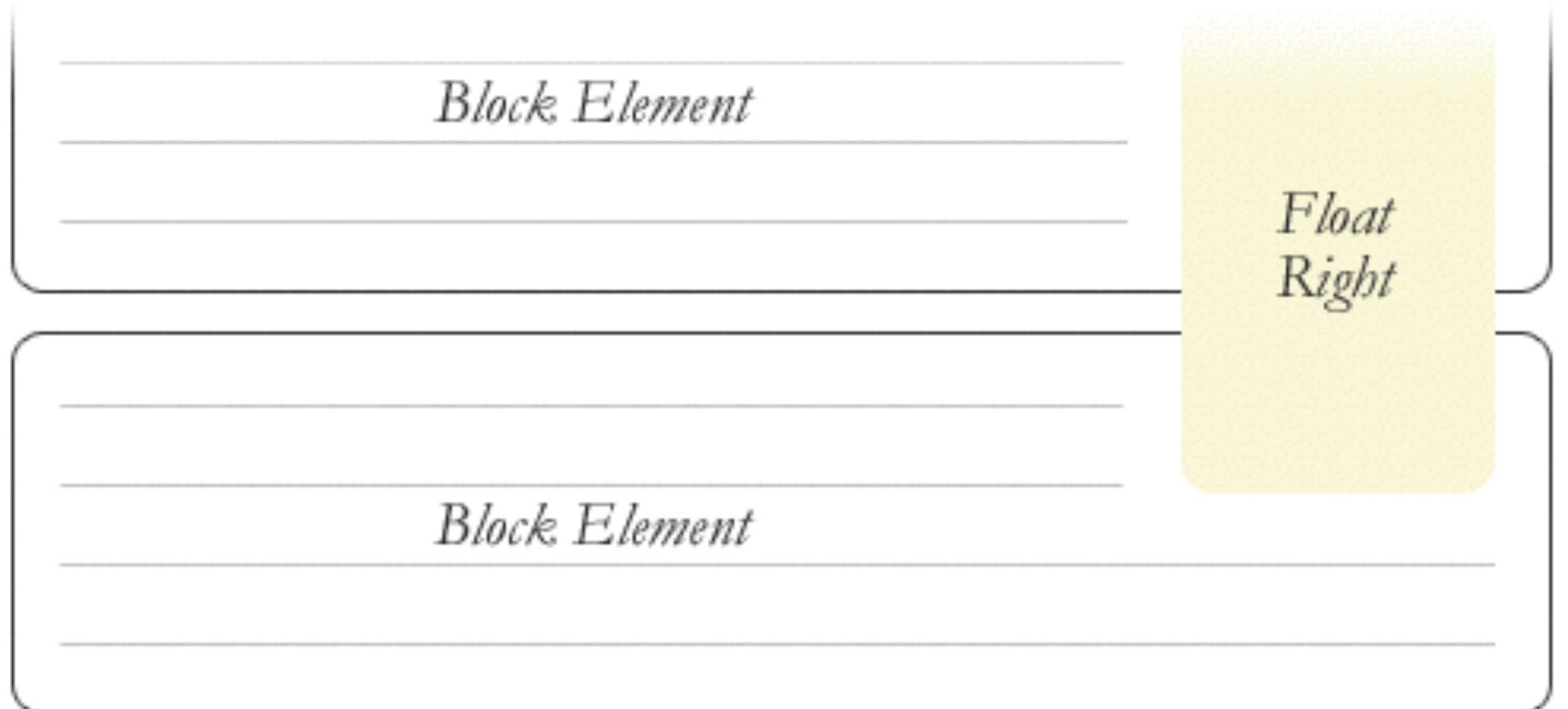
- **px**
- **%**
- **em**
- **rem**
- **vw**
- **vh**
- **...others**

# FLOAT

*“I understand floats.”*

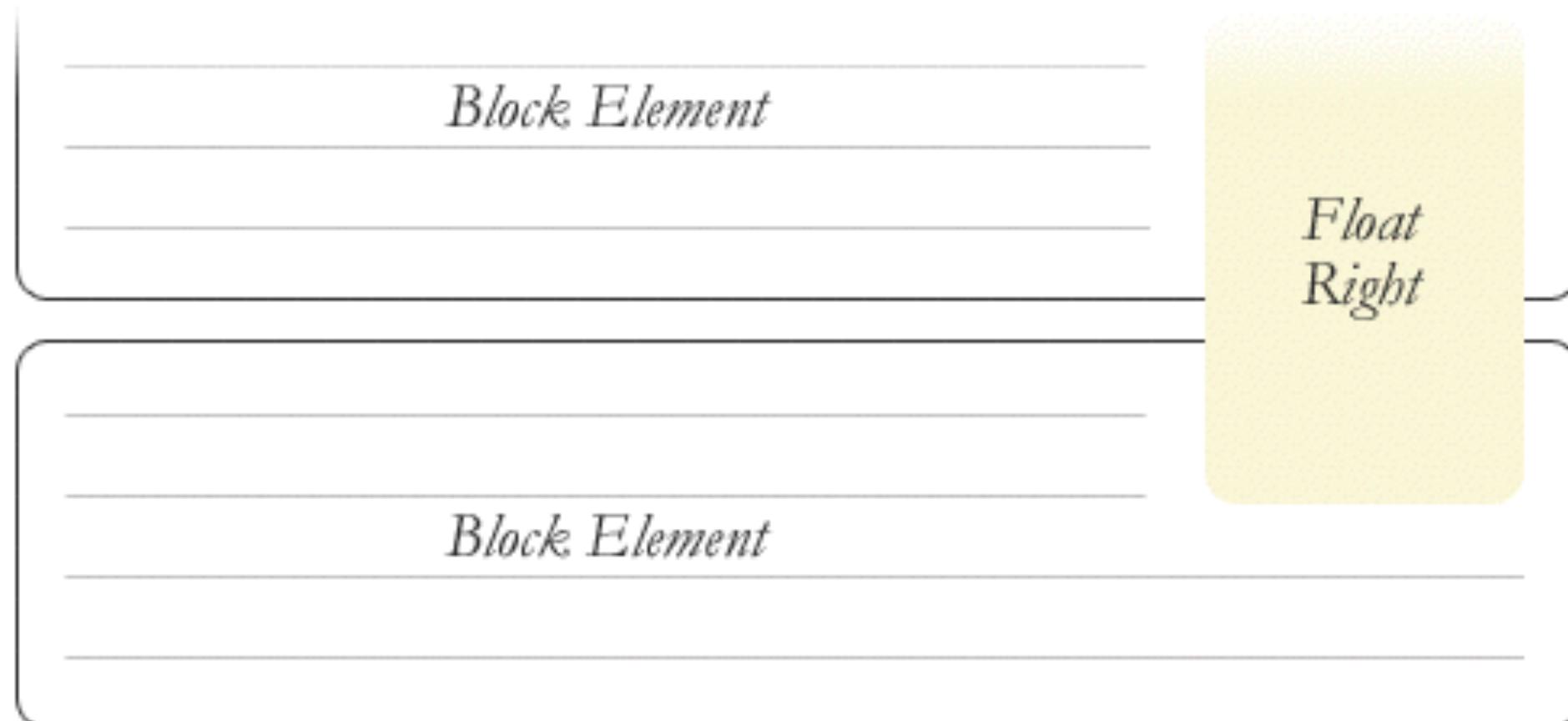
-LIAR MCPANTSONFIRE

```
<body>
  <div>...</div>
  <div style="float:right;">...</div>
  <div>...</div>
</body>
```

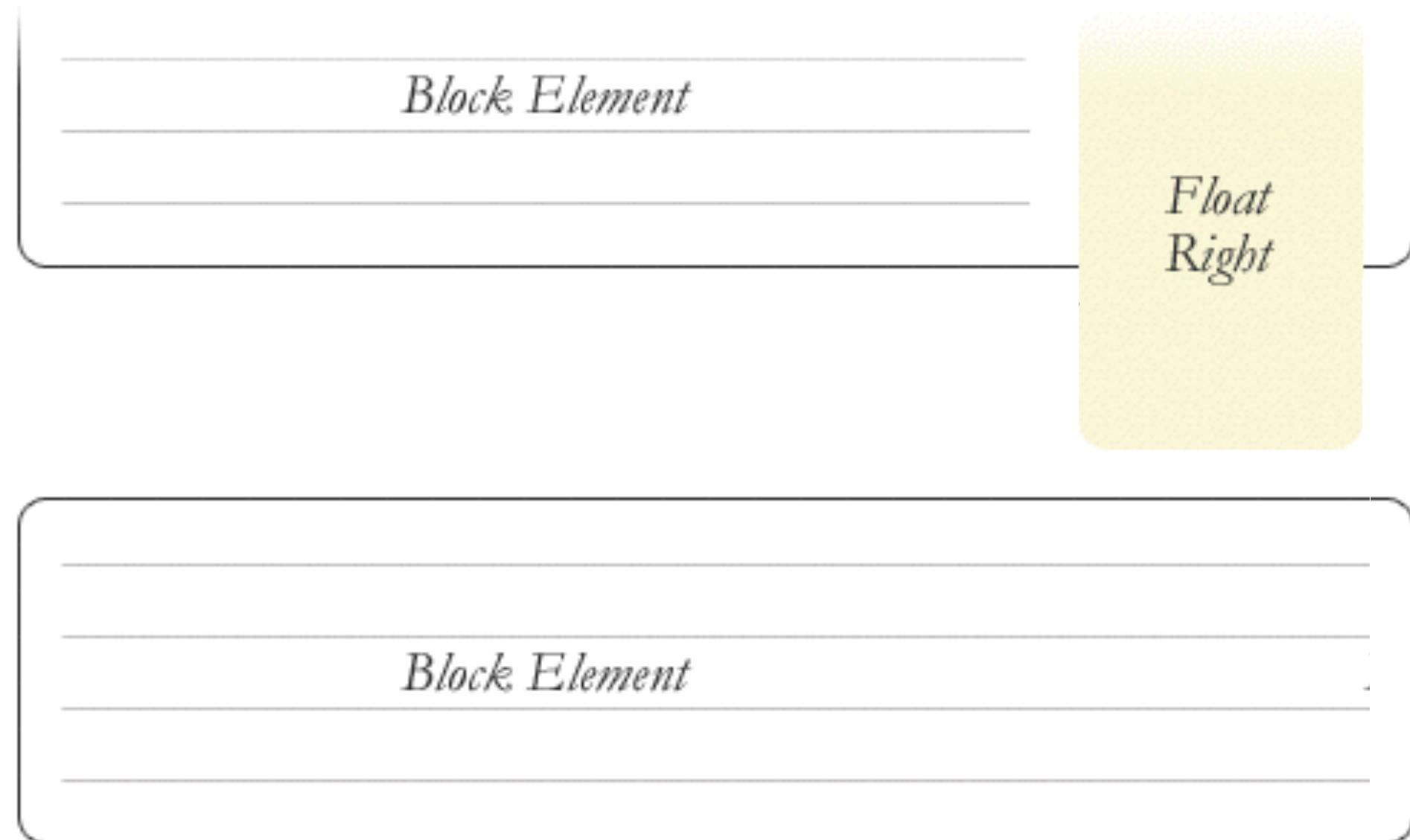


[HTTPS://CSS-TRICKS.COM/ALL-ABOUT-FLOATS/](https://css-tricks.com/all-about-floats/)

```
<body>
  <div>...</div>
  <div style="float:right;">...</div>
  <div>...</div>
</body>
```



```
<body>
  <div>...</div>
  <div style="float:right;">...</div>
  <div style="clear:right;">...</div>
</body>
```



[HTTPS://CSS-TRICKS.COM/ALL-ABOUT-FLOATS/](https://css-tricks.com/all-about-floats/)

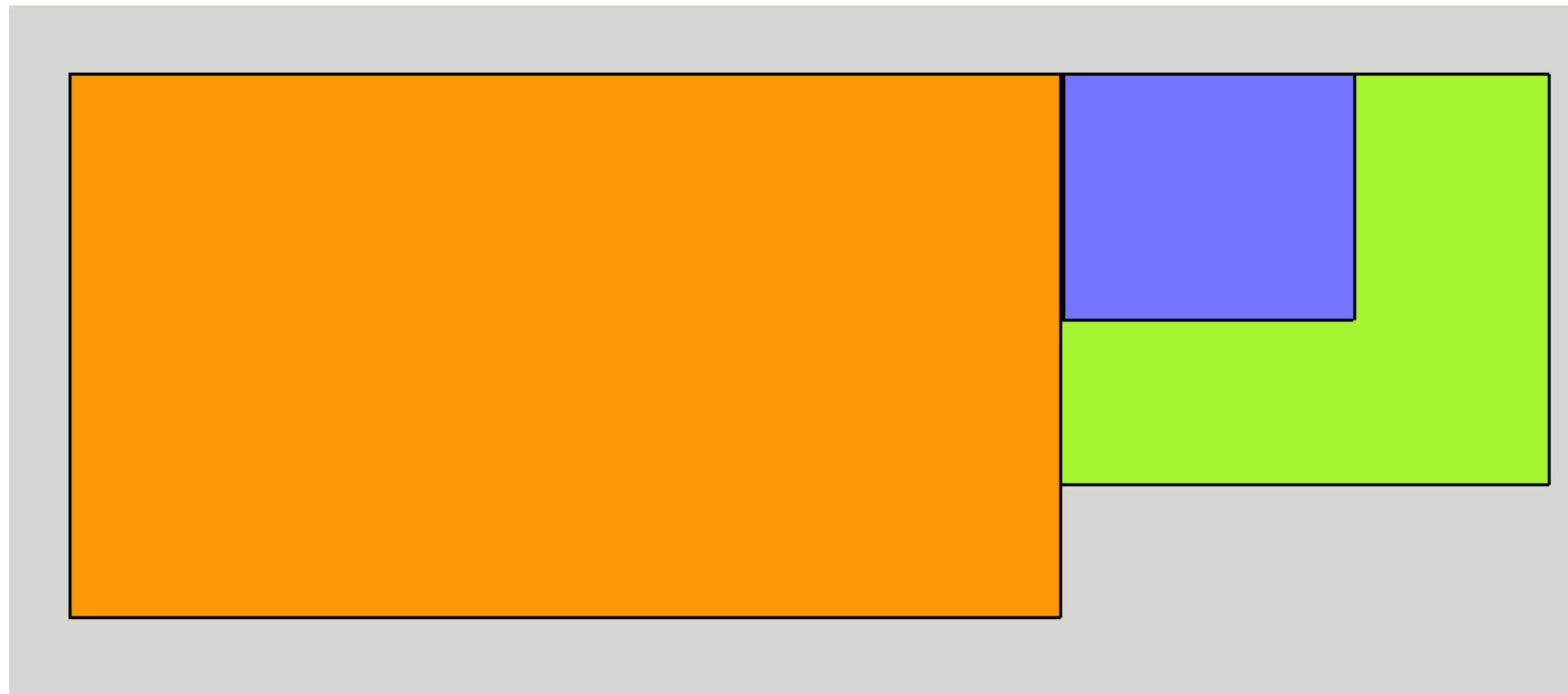
# GRIDS VIA FLOAT

- Floats were originally intended for *wrapping text*
- Floats can be used to implement a grid/column layout
- ..but if so you must use clearfix to avoid parent collapse

```
<div>
  <div style="float:left; background:orange; width:70%;"></div>
  <div style="background:green;">
    <span style="background:blue;">X</span>
  </div>
</div>
```

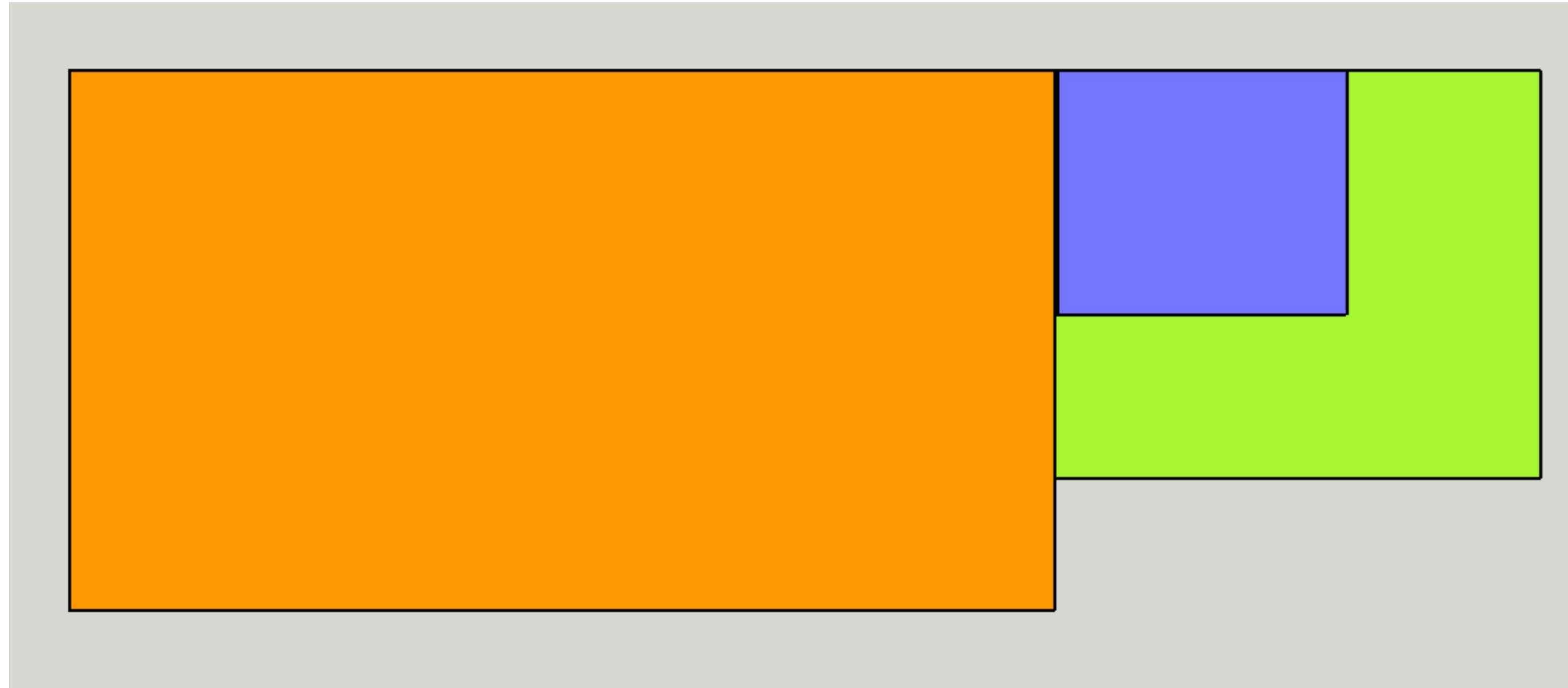
```
<div>
  <div style="float:left; background:orange; width:70%;"></div>
  <div style="background:green;">
    <span style="background:blue;">x</span>
  </div>
</div>
```

renders as...

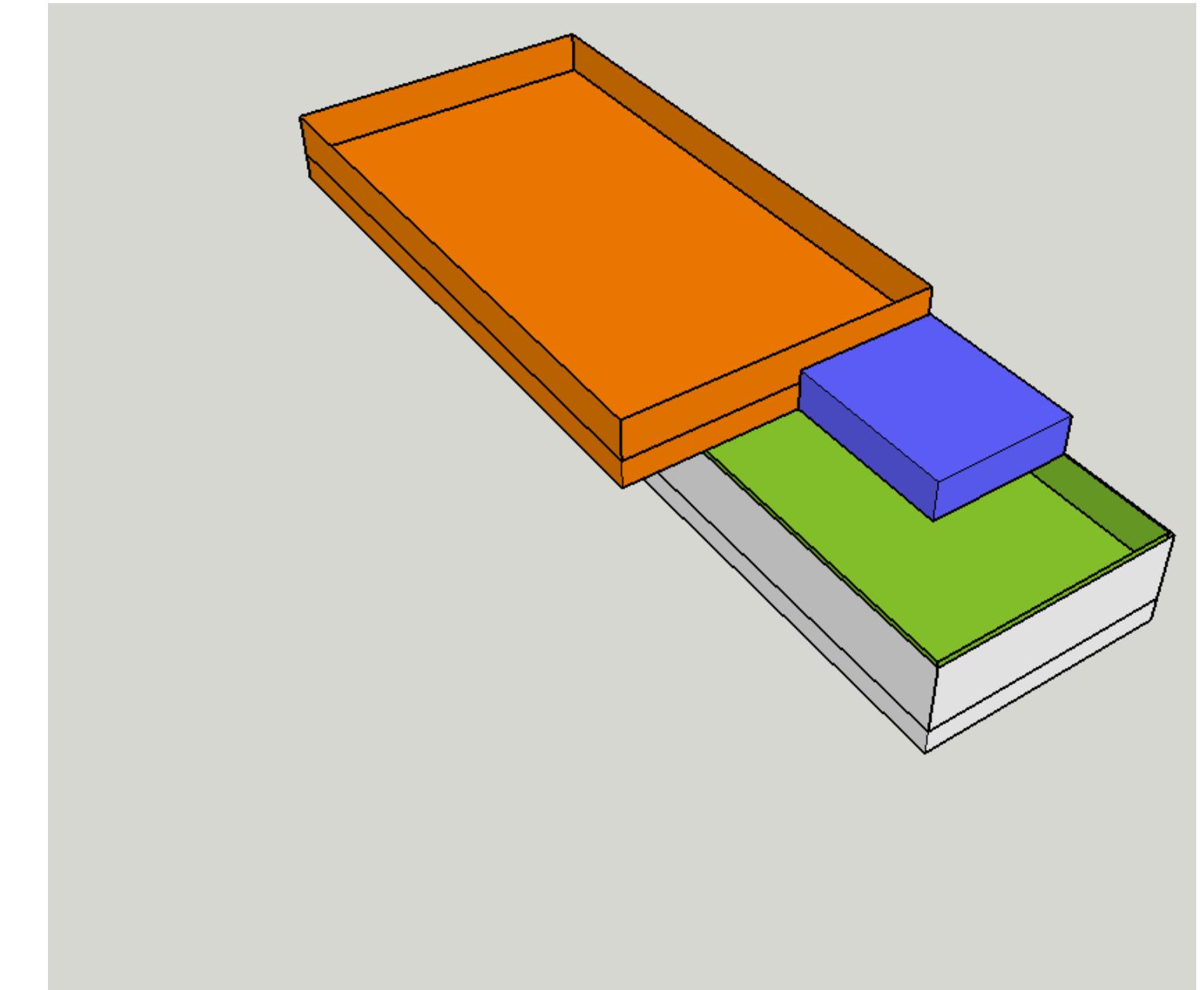


```
<div>
  <div style="float:left; background:orange; width:70%;"></div>
  <div style="background:green;">
    <span style="background:blue;">x</span>
  </div>
</div>
```

renders as...



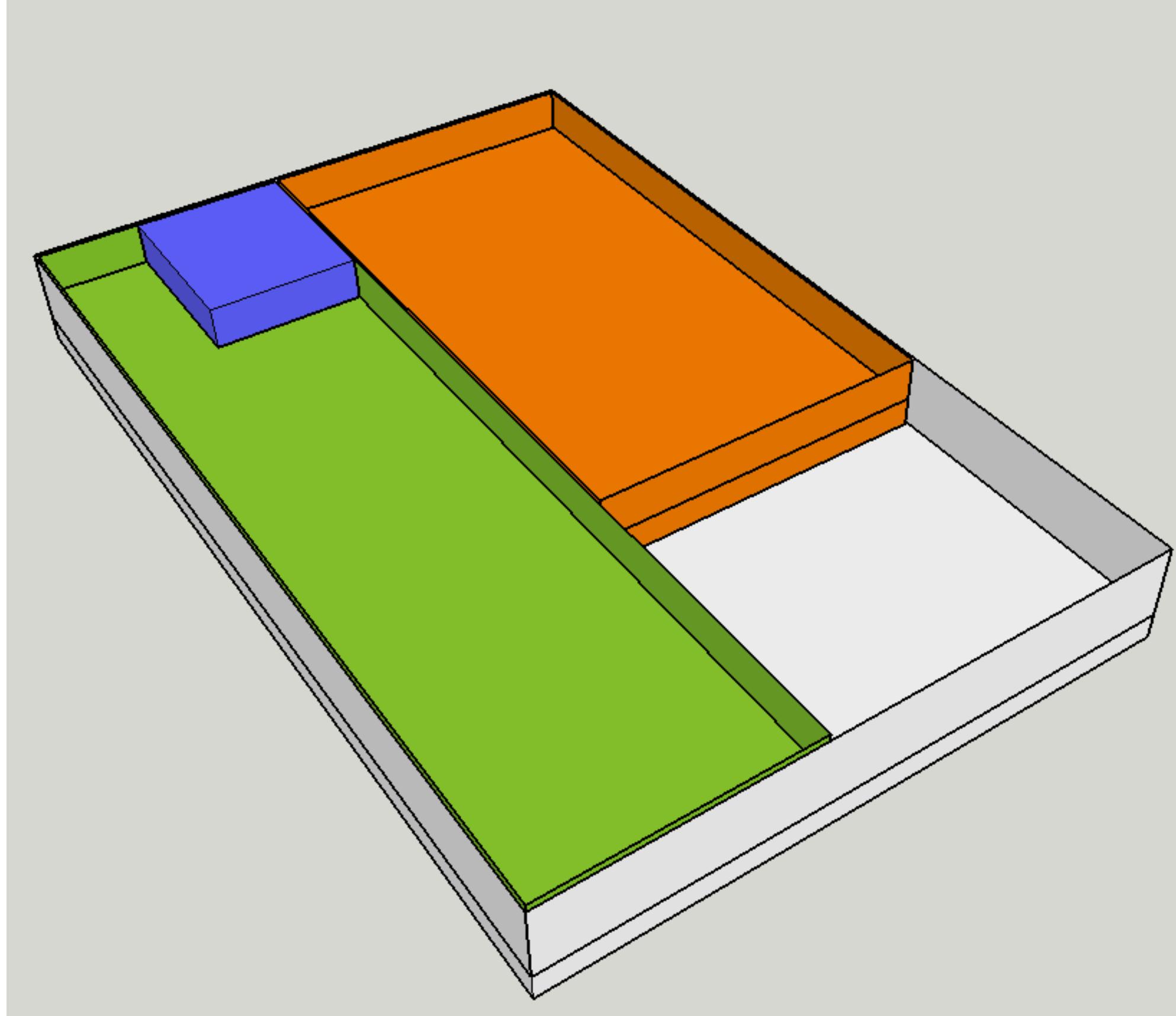
“different” perspective...



# **VISUAL METAPHOR**

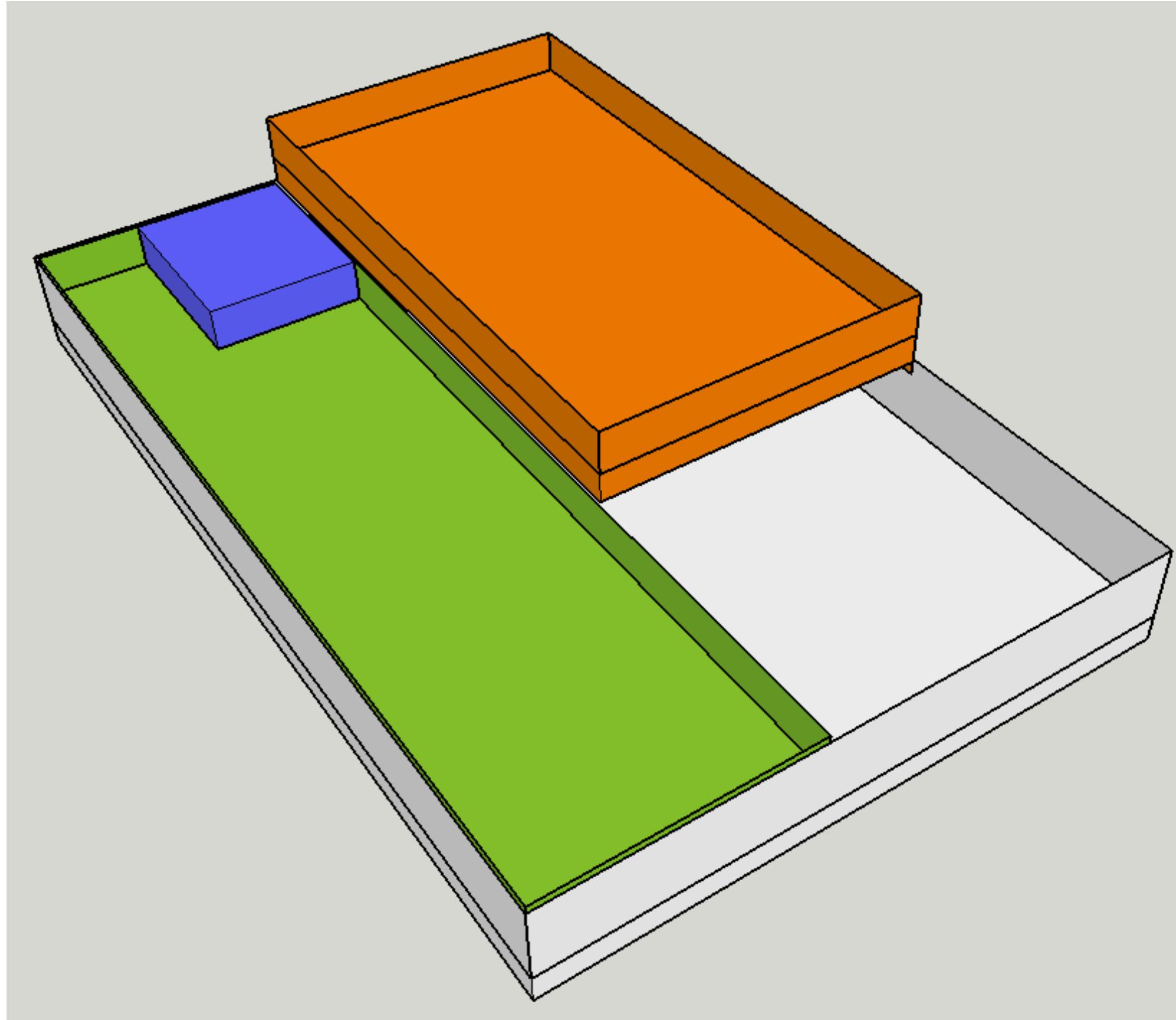
## **“STEP BY STEP”**

```
<div>
  <div style="float:left; background:orange; width:70%;"></div>
  <div style="background:green;">
    <span style="background:blue;">x</span>
  </div>
</div>
```



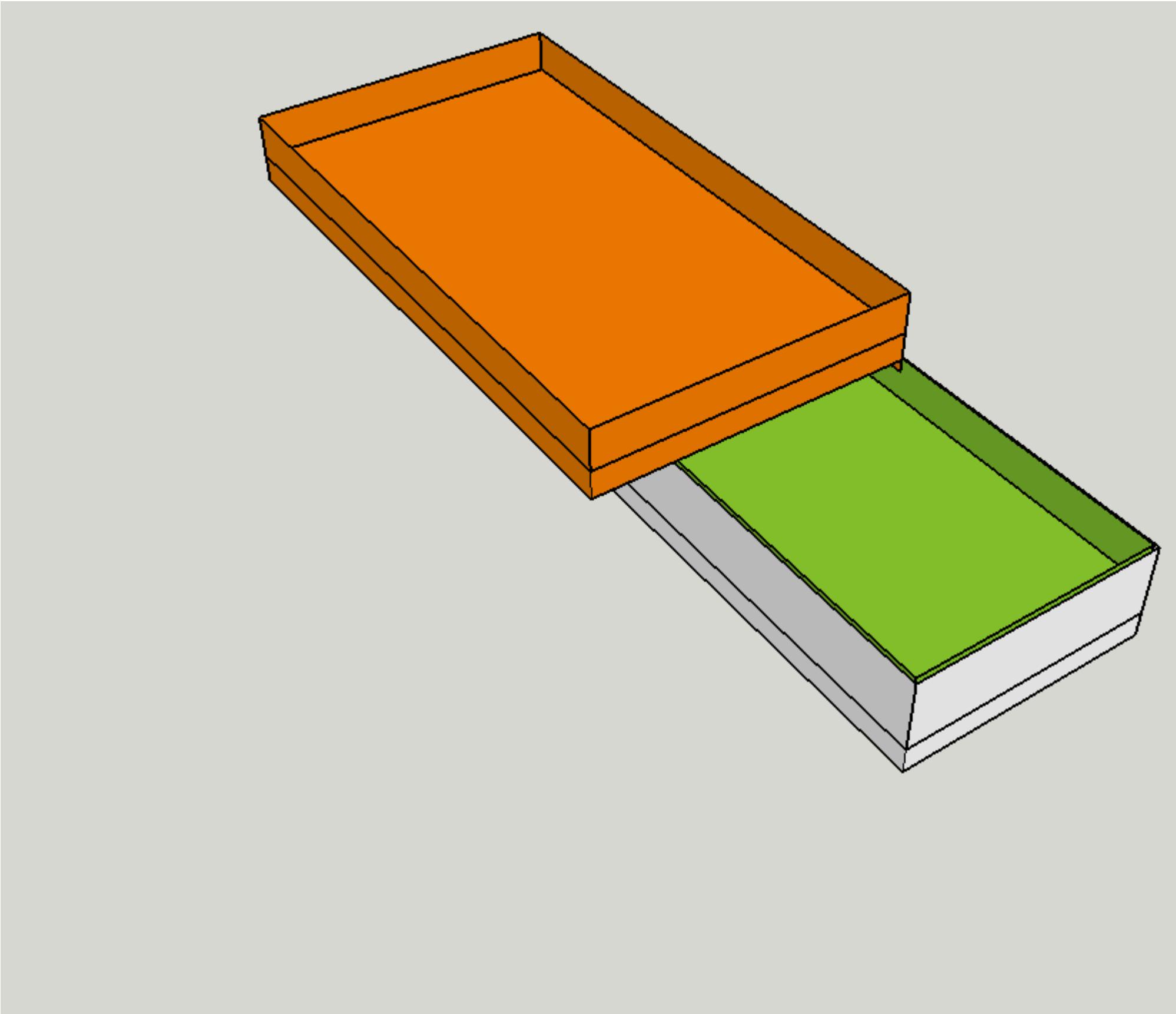
```
<div>
  <div style="float:left; background:orange; width:70%;"></div>
  <div style="background:green;">
    <span style="background:blue;">x</span>
  </div>
</div>
```

**orange floats and moves to top left**



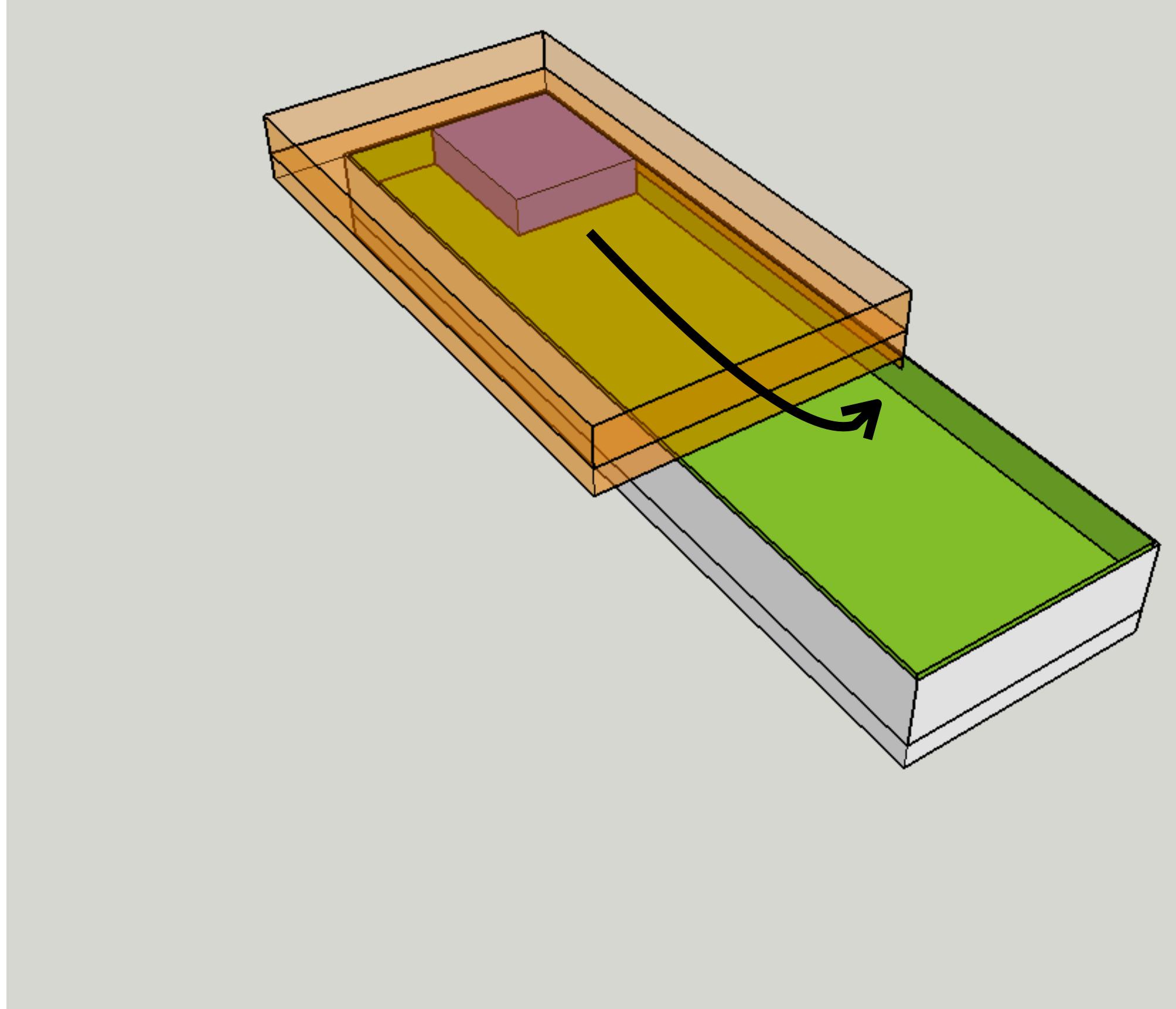
```
<div>
  <div style="float:left; background:orange; width:70%;"></div>
  <div style="background:green;">
    <span style="background:blue;">x</span>
  </div>
</div>
```

parent collapses



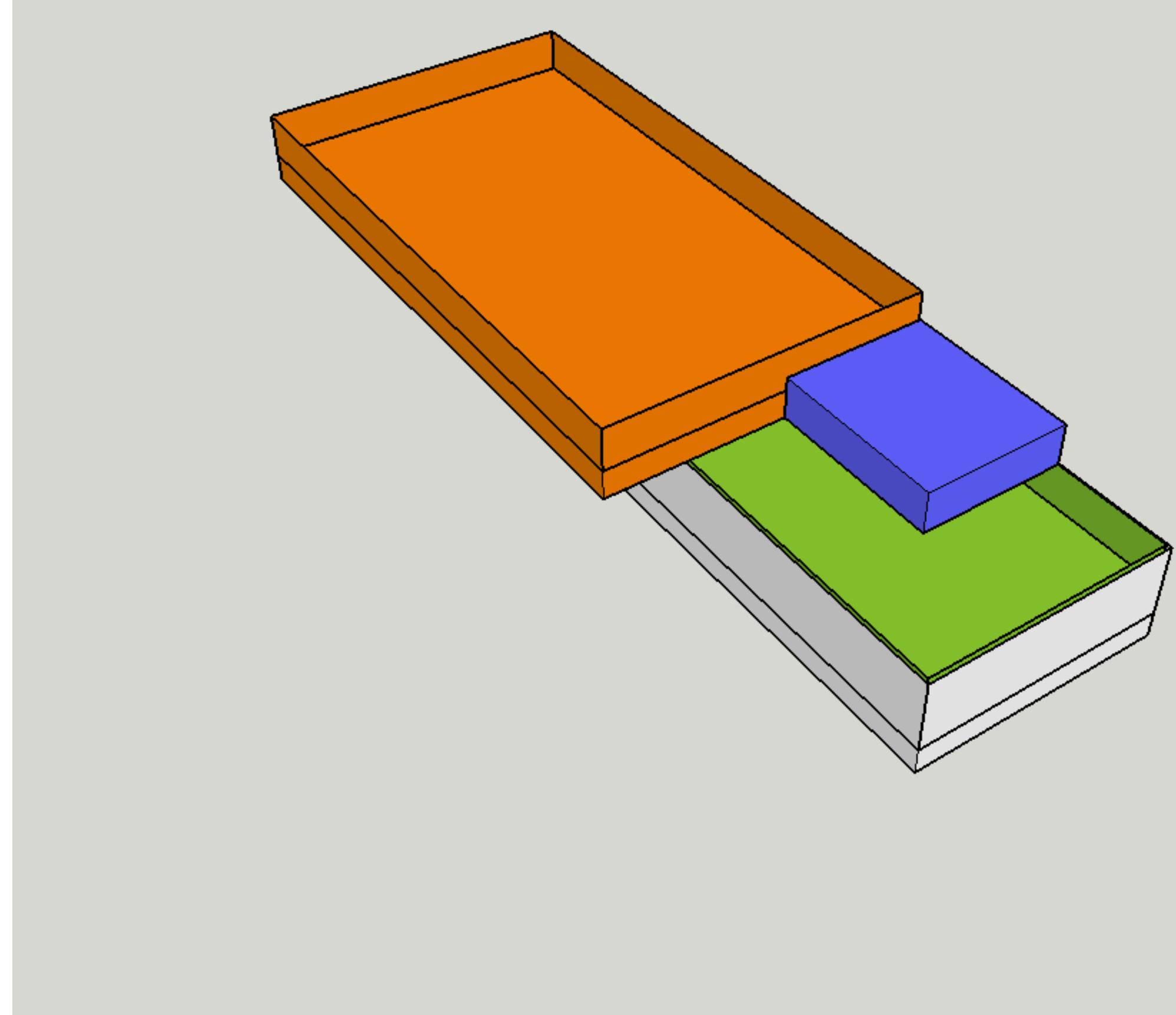
```
<div>
  <div style="float:left; background:orange; width:70%;"></div>
  <div style="background:green;">
    <span style="background:blue;">x</span>
  </div>
</div>
```

inline elements are “buoyant”

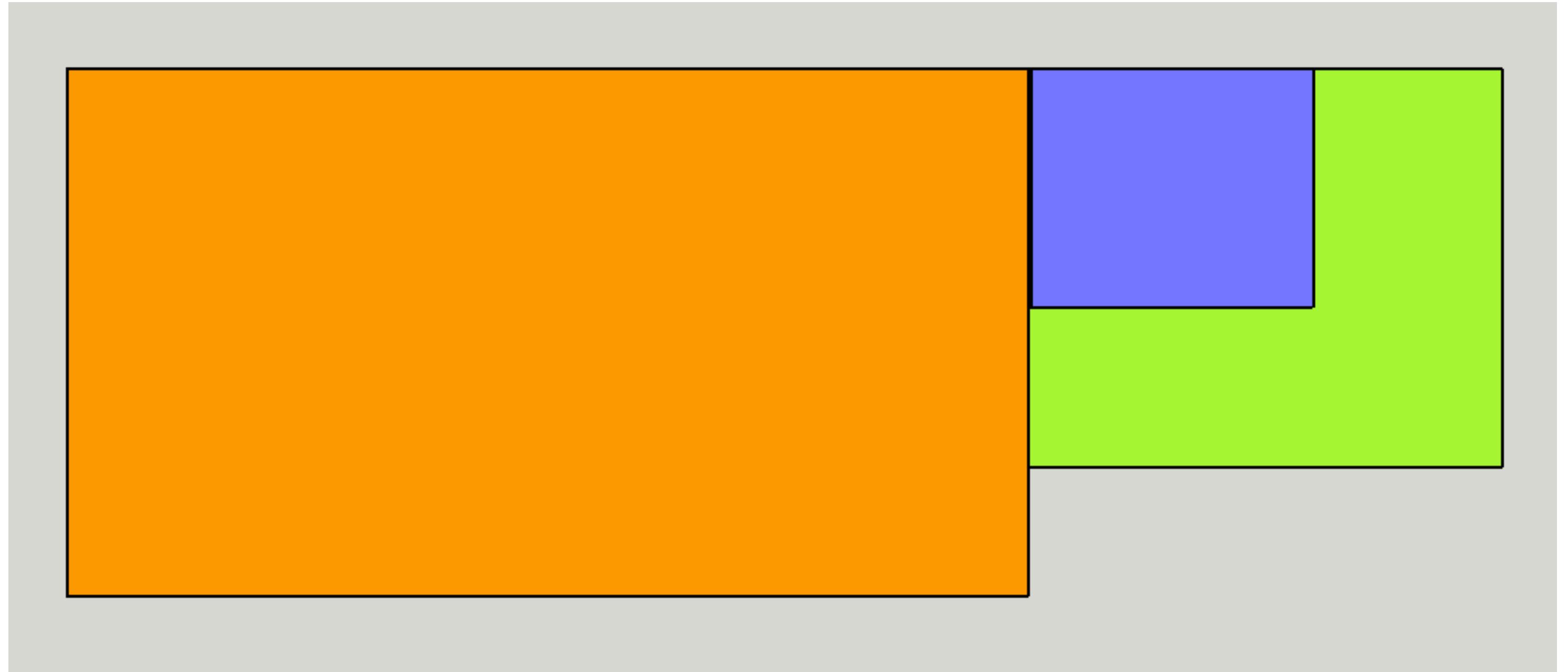
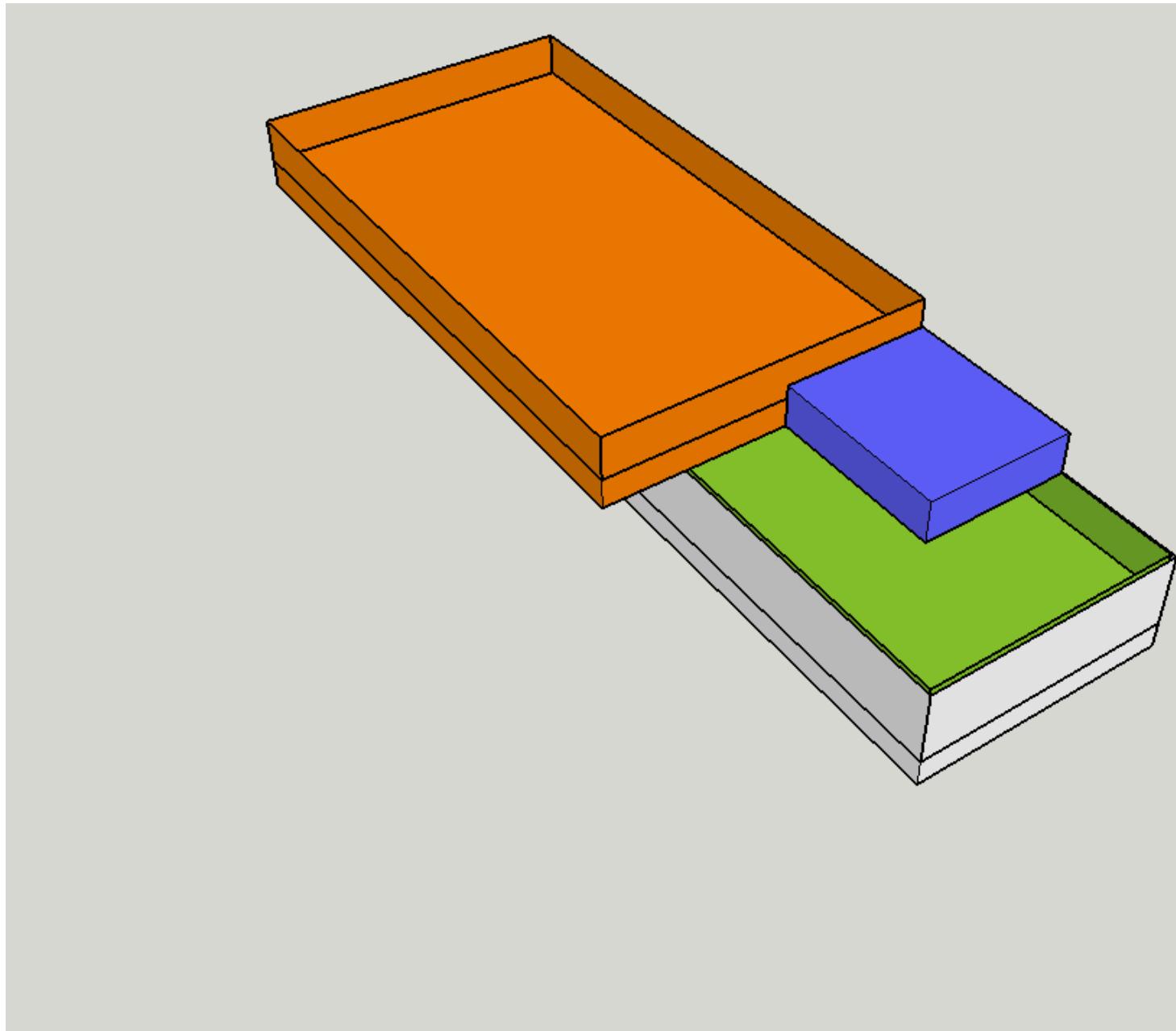


```
<div>
  <div style="float:left; background:orange; width:70%;"></div>
  <div style="background:green;">
    <span style="background:blue;">x</span>
  </div>
</div>
```

inline element wraps floated one

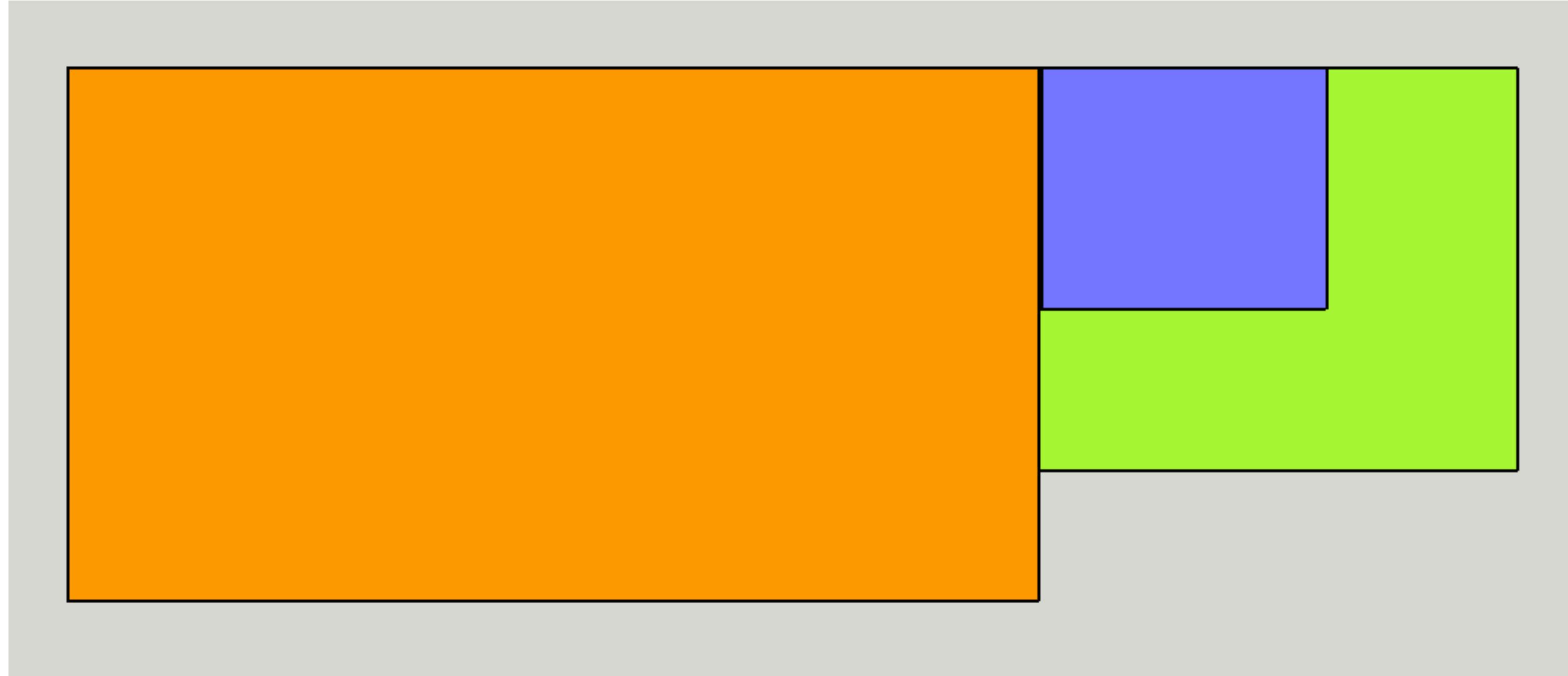
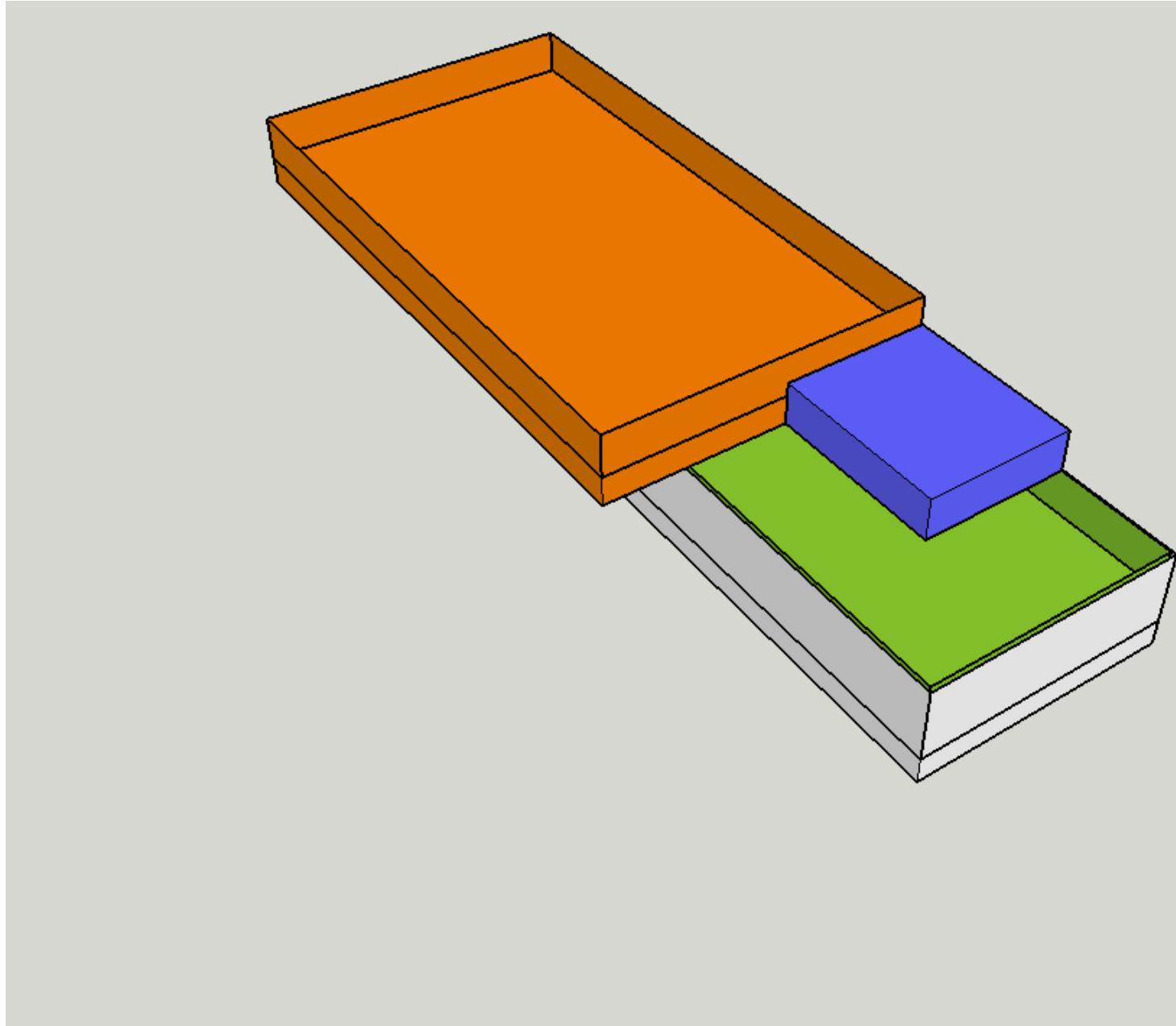
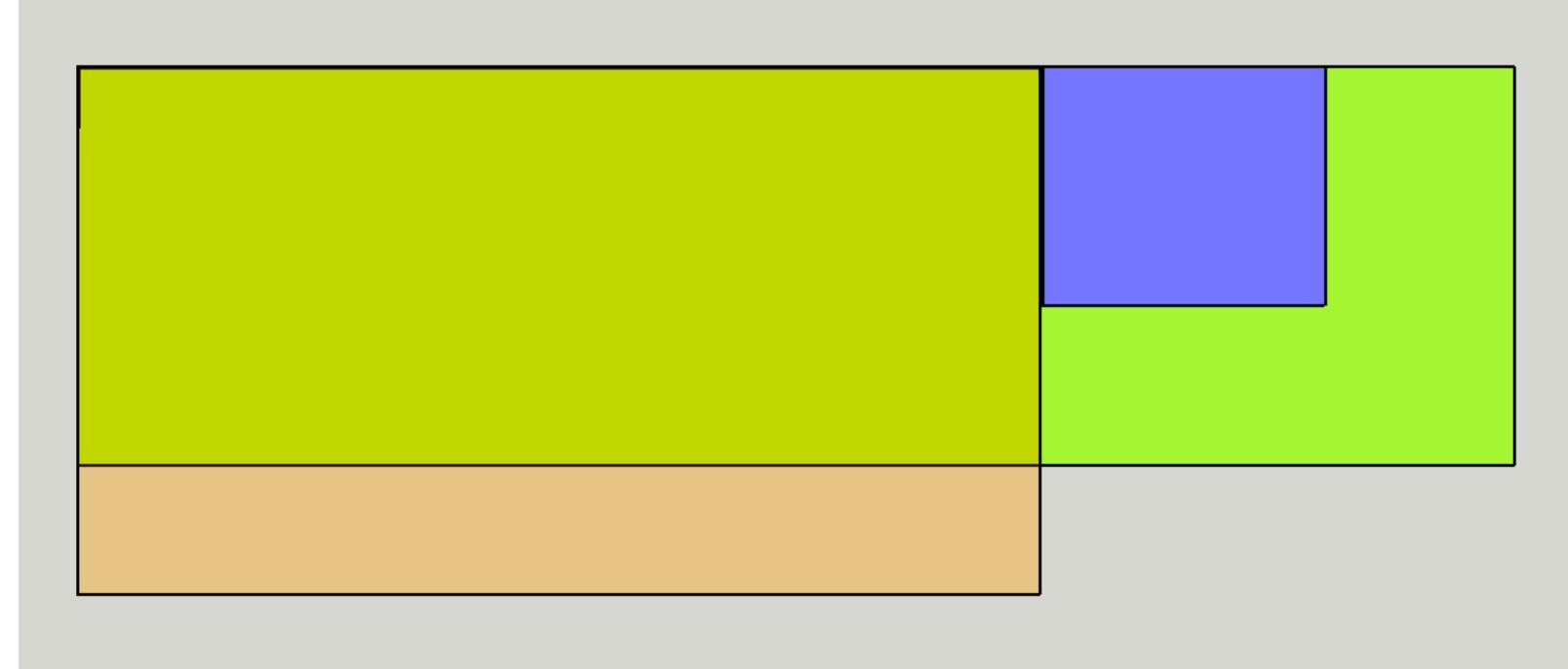


```
<div>
<div style="float:left; background:orange; width:70%;"></div>
<div style="background:green;">
  <span style="background:blue;">x</span>
</div>
</div>
```



```
<div>  
  <div style="float:left; background:orange; width:70%;"></div>  
  <div style="background:green;">  
    <span style="background:blue;">x</span>  
  </div>  
</div>
```

green div is behind orange div



# GRIDS VIA FLOAT

- Floats were originally intended for *wrapping text*
- Floats can be used to implement a grid/column layout
- ..but if so you must use clearfix to avoid parent collapse

# CLEARFIX

```
.clearfix::before, .clearfix::after {  
  clear: both;  
  display: block;  
  content: "";  
}
```





**S C S S**

**S C S S**  
**A A T H**  
**S S Y E**  
**S C L E**  
**Y A E T**  
D S  
I  
N  
G

S Y N T A C T I C A L L Y  
A W E S O M E  
S T Y L E  
S H E E T S

# SCSS

- ◎ “nesting”
- ◎ “variables”
- ◎ “loops”
- ◎ “functions”
- ◎ “modules”

# HOW DOES SCSS WORK?

# **SCSS COMPILES TO CSS**

# NESTING

**SCSS**

```
article {  
  border: 1px solid red;  
  li {  
    background: gray;  
  }  
}
```

**CSS**

# NESTING

## SCSS

```
article {  
  border: 1px solid red;  
  li {  
    background: gray;  
  }  
}
```

## CSS

```
article {  
  border: 1px solid red;  
}  
article li {  
  background: gray;  
}
```

# VARIABLES

## SCSS

```
$deep-red: #990000;  
a {  
    color: $deep-red;  
}  
.warning {  
    border-color: $deep-red;  
}
```

## CSS

# VARIABLES

## SCSS

```
$deep-red: #990000;  
a {  
  color: $deep-red;  
}  
.warning {  
  border-color: $deep-red;  
}
```

## CSS

```
a {  
  color: #990000;  
}  
.warning {  
  border-color: #990000;  
}
```

# LOOPS

## SCSS

```
@for $i from 1 through 3 {  
  h#${$i} {  
    font-size: $i * 10px;  
  }  
}
```

## CSS

# LOOPS

## SCSS

```
@for $i from 1 through 3 {  
  h#${$i} {  
    font-size: $i * 10px;  
  }  
}
```

## CSS

```
h1 {  
  font-size: 10px;  
}  
h2 {  
  font-size: 20px;  
}  
h3 {  
  font-size: 30px;  
}
```

# MIXINS

## SCSS

```
@mixin border-radius ($r) {  
  -webkit-border-radius: $r;  
  -moz-border-radius: $r;  
  border-radius: $r;  
}  
  
.thing {  
  @include border-radius(10px);  
}
```

## CSS

# MIXINS

## SCSS

```
@mixin border-radius ($r) {  
  -webkit-border-radius: $r;  
  -moz-border-radius: $r;  
  border-radius: $r;  
}  
  
.thing {  
  @include border-radius(10px);  
}
```

## CSS

```
.thing {  
  -webkit-border-radius: 10px;  
  -moz-border-radius: 10px;  
  border-radius: 10px;  
}
```

# MODULES

**SCSS**

```
/* pulls in normalize.scss */  
@import 'normalize';
```

**CSS**

# MODULES

## SCSS

```
/* pulls in normalize.scss */
@import 'normalize';
```

## CSS

```
/* ... */
/**
 * 1. Set default font family to sans-serif
 * 2. Prevent iOS text size adjust after orientation change, without disabling
 *    user zoom.
 */
html {
  font-family: sans-serif;
  /* 1 */
  -ms-text-size-adjust: 100%;
  /* 2 */
  -webkit-text-size-adjust: 100%;
  /* 2 */
}
/**
 * Remove default margin.
 */
body {
  margin: 0;
}
=====
Links
=====
*/
/*
 * Remove the gray background color from active links in IE 10.
 */
a {
  background: transparent;
}
/* ... */
```

# RESPONSIVE



# RESPONSIVE





# RESPONSIVE DESIGN



# RESPONSIVE DESIGN

- Website is fully functional for all screen sizes, resolutions and orientations



# RESPONSIVE DESIGN

- **Website is fully functional for all screen sizes, resolutions and orientations**
- **Born out of necessity (see previous slide)**



# RESPONSIVE DESIGN

- Website is fully functional for all screen sizes, resolutions and orientations
- Born out of necessity (see previous slide)
- Developers and designers should cater to the user's environment, not the other way around





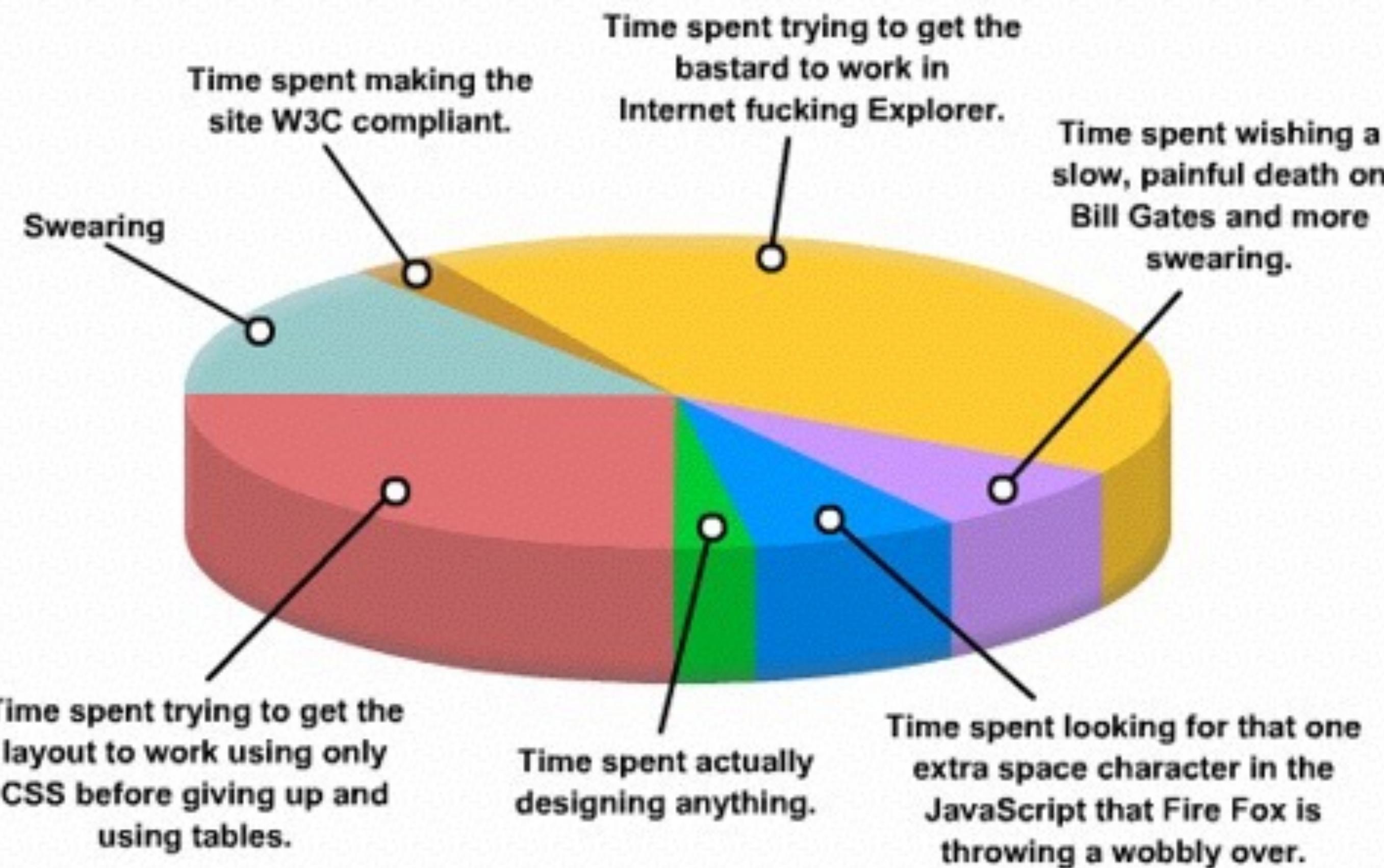
# RESPONSIVE DESIGN

- Website is fully functional for all screen sizes, resolutions and orientations
- Born out of necessity (see previous slide)
- Developers and designers should cater to the user's environment, not the other way around





# TIME BREAKDOWN OF MODERN WEB DESIGN



©2006 Alan "IE users must DIE" Foreman [poisonedminds.com](http://poisonedminds.com)



# EXAMPLES

- <http://hicksdesign.co.uk/>
- <https://medium.com/art-and-life/715c7c4d068d>

# TWO STEPS TO RESPONSIVE



# STEP 1: ADD META TAG

- **Most mobile browsers scale HTML pages to a wide viewport width so it fits on the screen**
- **You can use the viewport meta tag to reset this**
- **The viewport tag below tells browser to use the device width as the viewport width and disable the initial scale**

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```



# STEP 2: ADD MEDIA QUERIES

The CSS3 media queries let us set CSS rules that only apply at specific screen sizes

```
/* EXAMPLE */  
/* for 980px or less */  
@media screen and (max-width: 980px) {  
    #pagewrap {  
        width: 94%;  
    }  
    #content {  
        width: 65%;  
    }  
    #sidebar {  
        width: 30%;  
    }  
}
```



# HOW TO USE MEDIA QUERIES

- Set default styles for your page **OUTSIDE** of the media queries
- Progressively target smaller page sizes and let styles cascade
- Make things flexible and fluid
- Customary to add your responsive media-query tags to a separate CSS file

# EXAMPLE MEDIA QUERIES



# MIN-WIDTH

```
/* This rule will apply if viewing area is >900px */
@media screen and (min-width: 900px) {
  .class {
    background: #666;
  }
}
```



# MULTIPLE MEDIA QUERIES

```
/* The following code will apply if the viewing area is  
between 600px and 900px */  
@media screen and (min-width: 600px) and (max-width: 900px) {  
  .class {  
    background: #333;  
  }  
}
```



# DEVICE WIDTH

**max-device-width** means the **actual resolution of the device**  
**max-width** means the **viewing area resolution**

```
/* The following code will apply if the max-
device-width is 480px (eg. iPhone display). */
@media screen and (max-device-width: 480px) {
  .class {
    background: #000;
  }
}
```



# FOR TABLETS

**You can use a media query to detect orientation (portrait or landscape)**

```
<link media="all and (orientation:portrait)" href="portrait.css" rel="stylesheet">
<link media="all and (orientation:landscape)" href="landscape.css" rel="stylesheet">
```



# SUMMARY

- **CSS role**
- **terminology**
- **specificity**
- **layout & float**
- **box model**
- **SCSS**
- **responsive CSS**



# SO MUCH MORE...

- **consistent centering (guide)**
- **flexbox (guide, game)**
- **animations (intro)**
- **z-index (explanation)**
- **margin collapsing (docs)**
- **attribute selection operators (docs)**
- **patterns (intro)**
- **sprites (intro)**

# QUESTIONS?