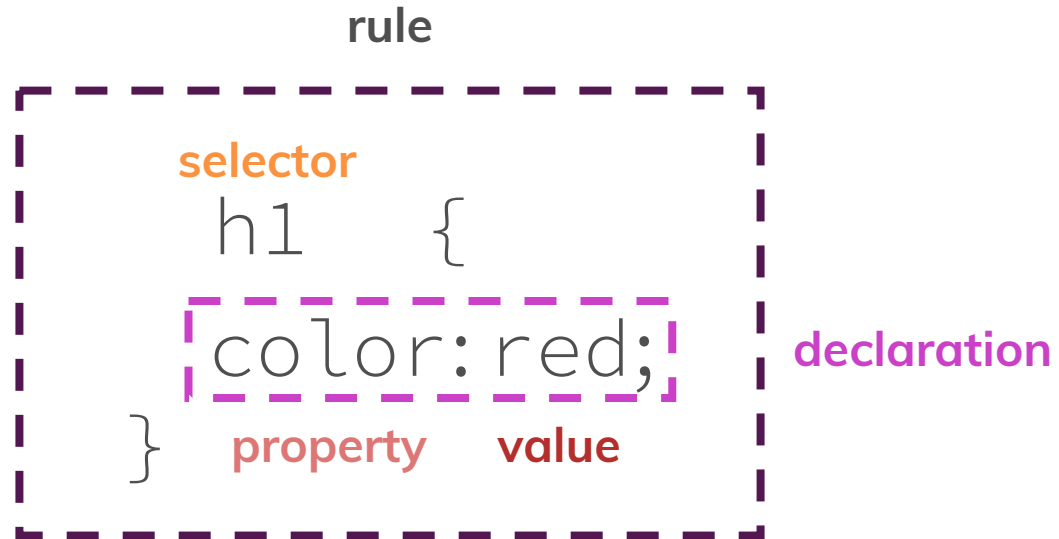


Understanding the CSS Syntax



More about Selectors



Elements

➡ Set equal style for these elements

```
<h1>Our header</h1>
<p>The Blog Post</p>
<div>More Info</div>
```

```
h1 {
  color: red;
}
```

Classes

➡ Set equal style for elements within the same class

```
<h1 class="blog-post">
Our header</h1>
<p class="blog-post">
The blog post</p>
<div class="blog-post">
More info</div>
```

```
.blog-post {
  color: red;
}
```

Universal

```
<h1>Our header</h1>
<p class="blog-post">The
blog post</p>
```

```
* {
  color: red;
}
```

Rarely use this one!

More about Selectors



IDs

➡ Set style to one specific element

```
<h1 id="main-title">Our  
header</h1>
```



```
#main-title {  
  color: red;  
}
```

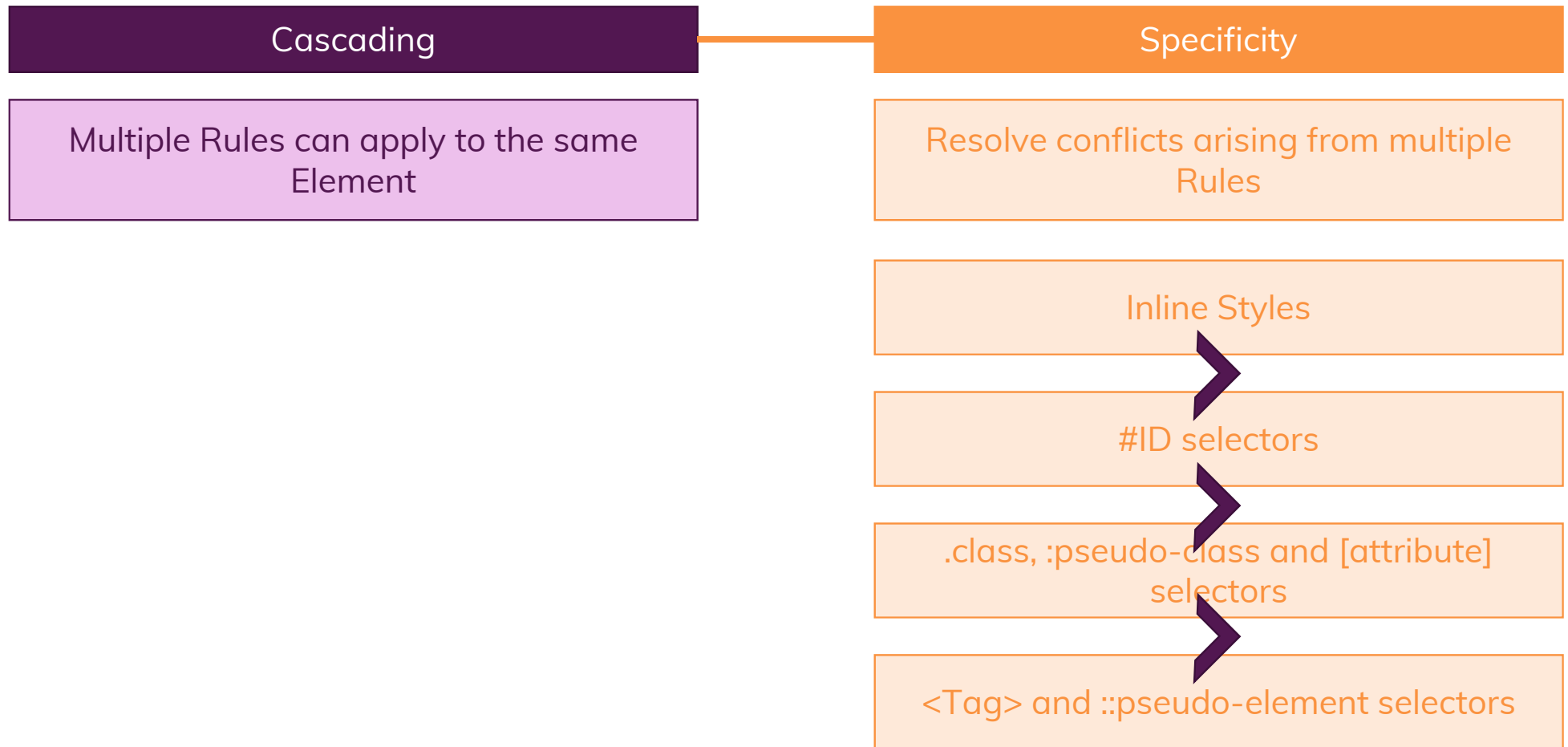
Attributes

➡ Set equal styles to all elements with attribute(s)

```
<button disabled>  
  Click  
</button>
```

```
[disabled] {  
  color: red;  
}
```

Cascading Style Sheets & Specificity



Cascading Style Sheets & Specificity

Cascading

Multiple Rules can apply to the same Element

Specificity

Resolve conflicts arising from multiple Rules

Selector Hierarchy

Directly applied Styles win over Inheritance

```
p {...} > div {...} for  
<div><p>Hi!</p></div>
```

More specific Selector wins over less specific one

```
p.some-class {...} > p {...} for  
<p class="some-class">Hi!</p>
```

Inheritance

```
div {  
  color: red;  
}
```



```
p {  
  color: green;  
}
```



Parent styles are inherited by child elements if not overwritten!



```
<div>  
  <div>  
    <h1>Inherited!</h1>  
  </div>  
  <p>Overwritten</p>  
  <div>Inherited!</div>  
  <article>  
    <p>Overwritten</p>  
  </article>  
  <p>Overwritten</p>  
</div>
```



Understanding Combinators

 Adjacent Sibling

```
div + p {  
}
```

 General Sibling

```
div ~ p {  
}
```

 Child

```
div > p {  
}
```

 Descendant

```
div p {  
}
```



Combinators – Adjacent Sibling

+ Adjacent Sibling

```
h2 + p {  
  color: red;  
}
```



```
<div>  
  <h2>Not applied</h2>  
  <p>CSS applied</p>  
  <h2>Not applied</h2>  
  <h3>Not applied</h3>  
  <p>Not applied</p>  
  <h2>Not applied</h2>  
  <p>CSS applied</p>  
</div>
```



Elements share the same parent



Second element comes **immediately** after first element



Combinators – General Sibling

General Sibling

```
h2 ~ p {  
  color: red;  
}
```



```
<div>  
  <h2>Not applied</h2>  
  <p>CSS applied</p>  
  <h2>Not applied</h2>  
  <h3>Not applied</h3>  
  <p>CSS applied</p>  
</div>
```



Element share the same parent



Second element comes after first element



Combinators – Child

> Child

```
div > p {  
  color: red;  
}
```



```
<div>  
  <div>Not applied</div>  
  <p>CSS applied</p>  
  <div>Not applied</div>  
  <article>  
    <p>Not applied</p>  
  </article>  
  <p>CSS applied</p>  
</div>
```



Second element is a direct child of first element



Combinators – Descendant

 Descendant

```
div p {  
  color: red;  
}
```



Second element is a descendant of the first element



```
<div>  
  <div>Not applied</div>  
  <p>CSS applied</p>  
  <div>Not applied</div>  
  <article>  
    <p>CSS applied</p>  
  </article>  
  <p>CSS applied</p>  
</div>
```



Value Types

Values are tightly coupled to specific property!

Pre-defined Options	Colors	Length, Sizes & Numbers	Functions
<code>display: block;</code>	<code>background: red;</code>	<code>height: 100px;</code>	<code>background: url(...);</code>
<code>overflow: auto;</code>	<code>color: #fa923f;</code>	<code>width: 20%;</code>	<code>transform: scale(...);</code>
	<code>color: #ccc;</code>	<code>order: 1;</code>	

Possible Values can be found in CSS References
(e.g. MDN)!

Summary

CSS works with Rules

```
h1 {  
  color: red;  
}  
p {  
  color: red;  
}
```

Different Types of Selectors

```
h1 {...}  
.some-class {...}  
[disabled] {...}  
#some-id {...}  
* {...}
```

Selectors with Combinators

```
div + p {  
  color: red;  
}  
div ~ p {  
  color: red;  
}  
div > p {  
  color: red;  
}  
div p {  
  color: red;  
}
```

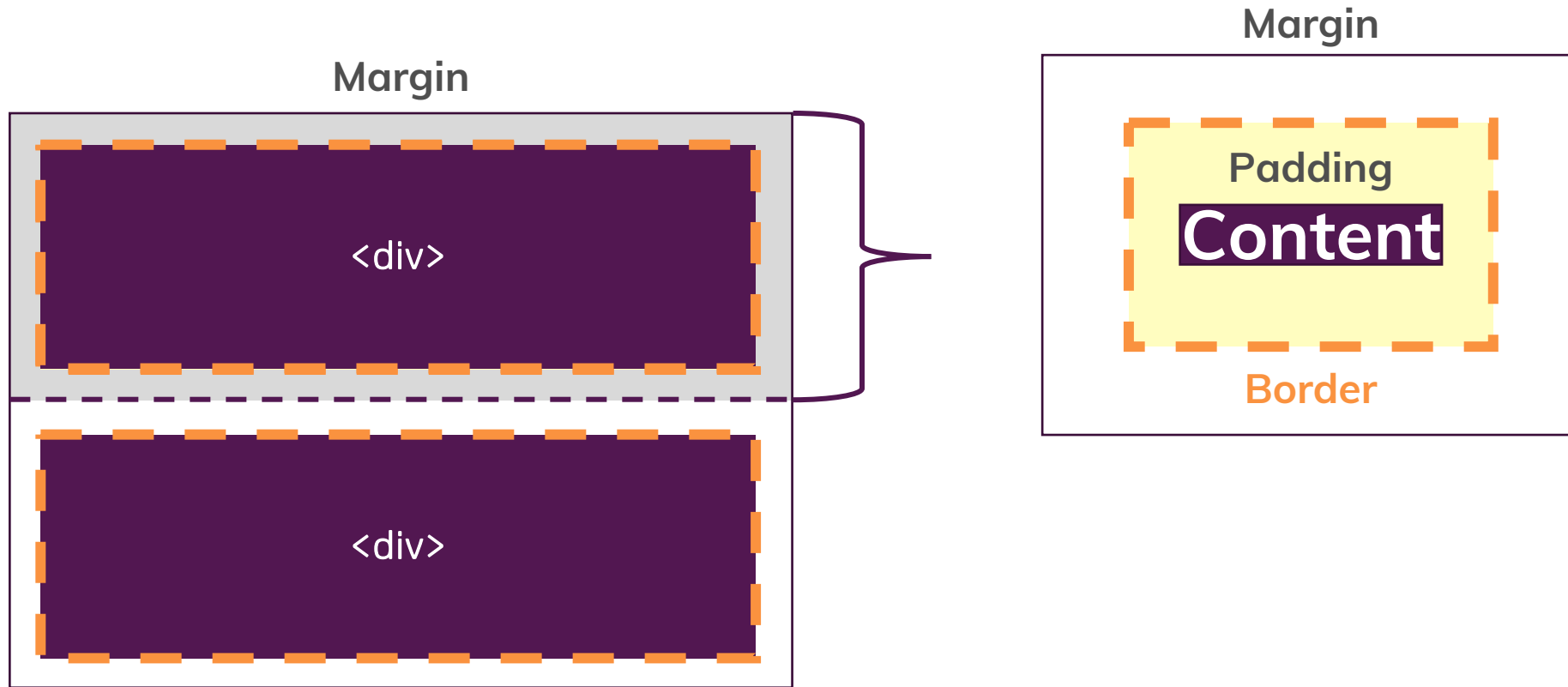
Properties & Values

- Long list of available Properties and Values
- Check MDN or comparable References
- Different Type of Values, depending on Properties

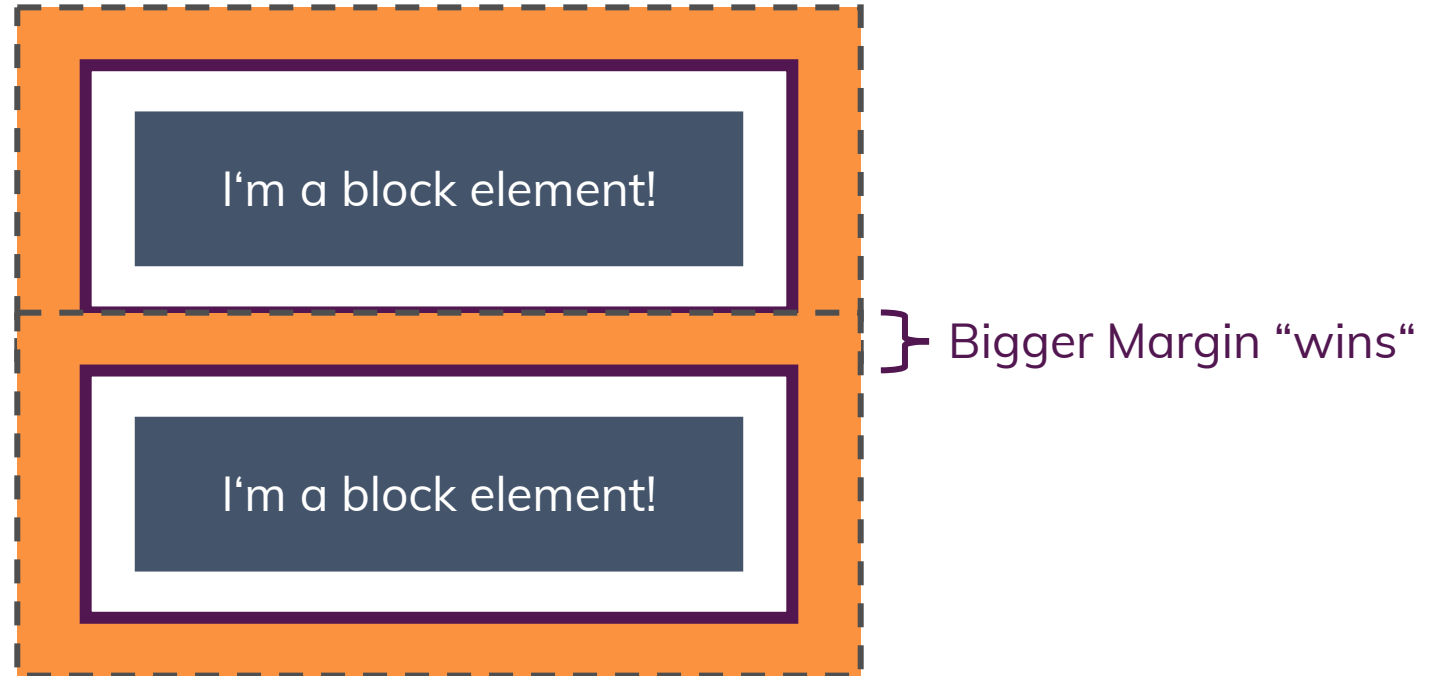
Inheritance & Specificity

- Parent styles are generally inherited
- Multiple rules can apply to one element
- Specificity resolves “multiple rules” conflicts
- Inheritance defaults can be changed

The Box Model



Margin Collapsing



In General: Use either `margin-top` or `margin-bottom`

Shorthand Properties



Combine values of multiple properties in a single property (the shorthand property)



Separate properties

```
border-width: 2px  
border-style: dashed | solid  
border-color: orange
```

```
margin-top: 5px  
margin-right: 10px  
margin-bottom: 5px  
margin-left: 10px
```



Shorthand property

```
border: 2px dashed orange
```

```
margin: 5px 10px 5px 10px;
```

top right bottom left

```
margin: 5px 10px;
```

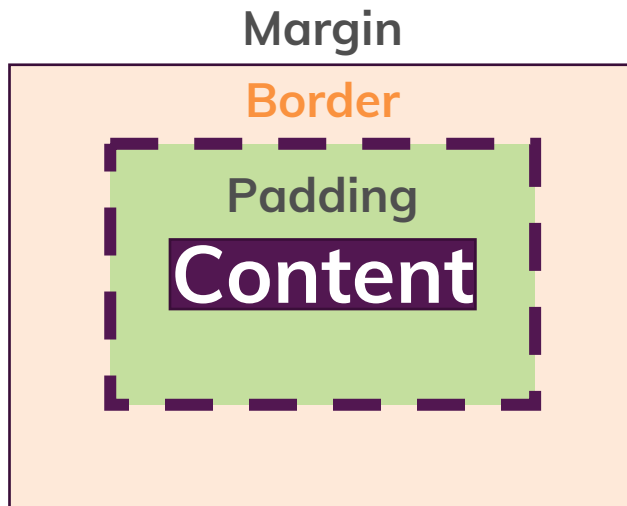
top & bottom

left & right

```
margin: 10px;
```


Summary

The Box Model



Styling Width & Height

- px or % (or other units)
- % refers to container
- width and height
- box-sizing can be content-box (default) or border-box

The “display” Property

- Control behavior (block vs inline) of elements
- Mix behavior via inline-block
- Remove elements via none

Pseudo Classes & Elements

:hover
:active
:first-of-type

::after
::first-letter

Summary

CSS Class Selectors

- You can apply more than one class to an element
- You can chain selectors (e.g. `a.active`, `.priority.highlighted`)
- Class selectors are the most-used type of CSS selectors

`!important`

- Important: Don't use `!important` in 99% of cases

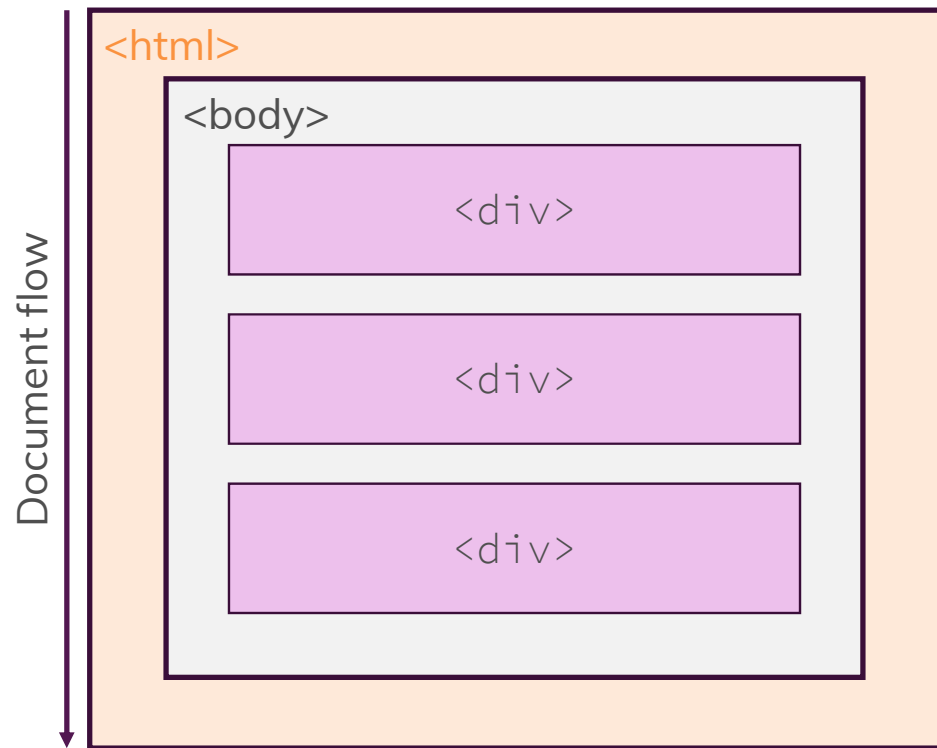
Pseudo Selectors & `:not`

- You use the same pseudo-selectors in most cases (`:hover`, `:active`)
- Explore your possibilities to solve edge cases with ease
- Use `:not` with caution but when needed to exclude certain elements

Positioning

How to change the position of elements

Positioning Elements

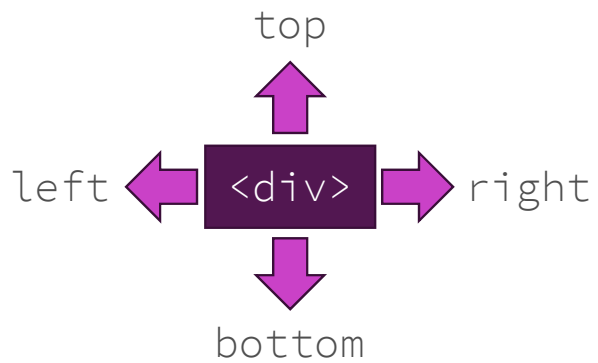


position:
static
absolute
relative
fixed
sticky

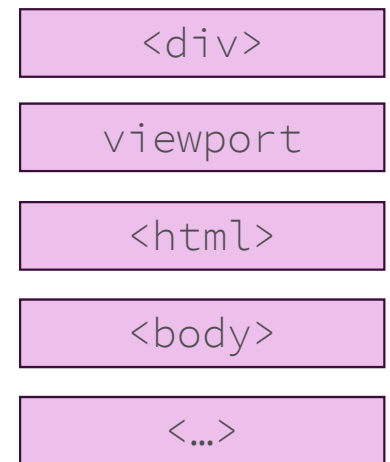
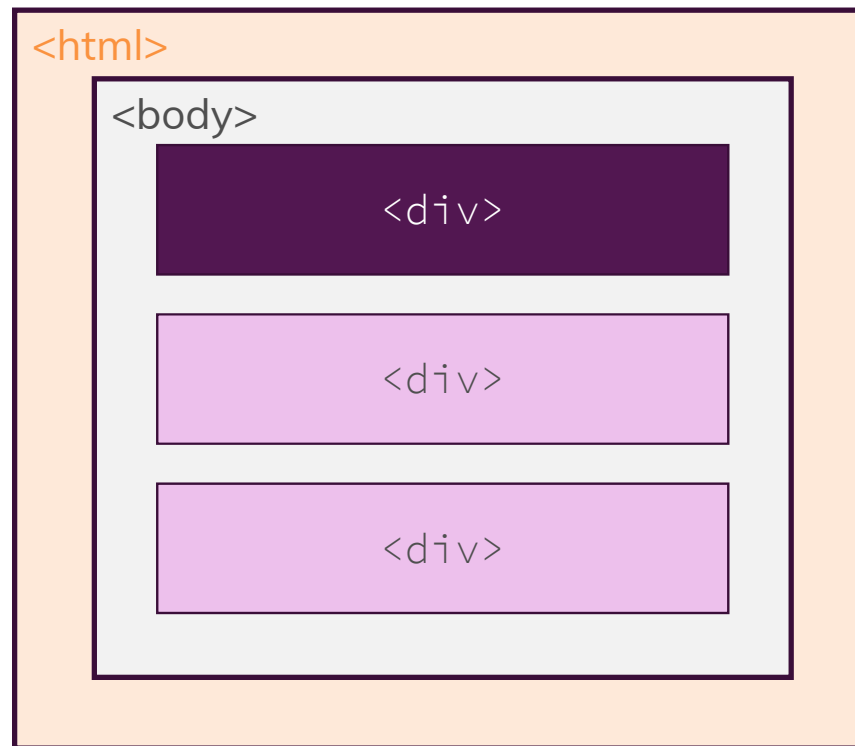


Default

Changing the Position



Document Flow



Positioning Context

Summary

The “position” Property

static (default)
fixed
absolute
relative
sticky

Positioning Context

- Defines the anchor point for your position change
- The viewport for `fixed`
- Another element for `absolute`
- The element itself for `relative`
- The viewport and another element for `sticky`

The “Document Flow”

- The default positioning behaviour of html elements
- Can be changed with `position`
- Elements can remain in the document flow or be excluded from it

Stacking Context

- Created when applying `fixed/ sticky` or `absolute/ relative` in combination with `z-index`
- Defines stacking behaviour of child elements

Moving Elements

- `top`
- `bottom`
- `left`
- `right`

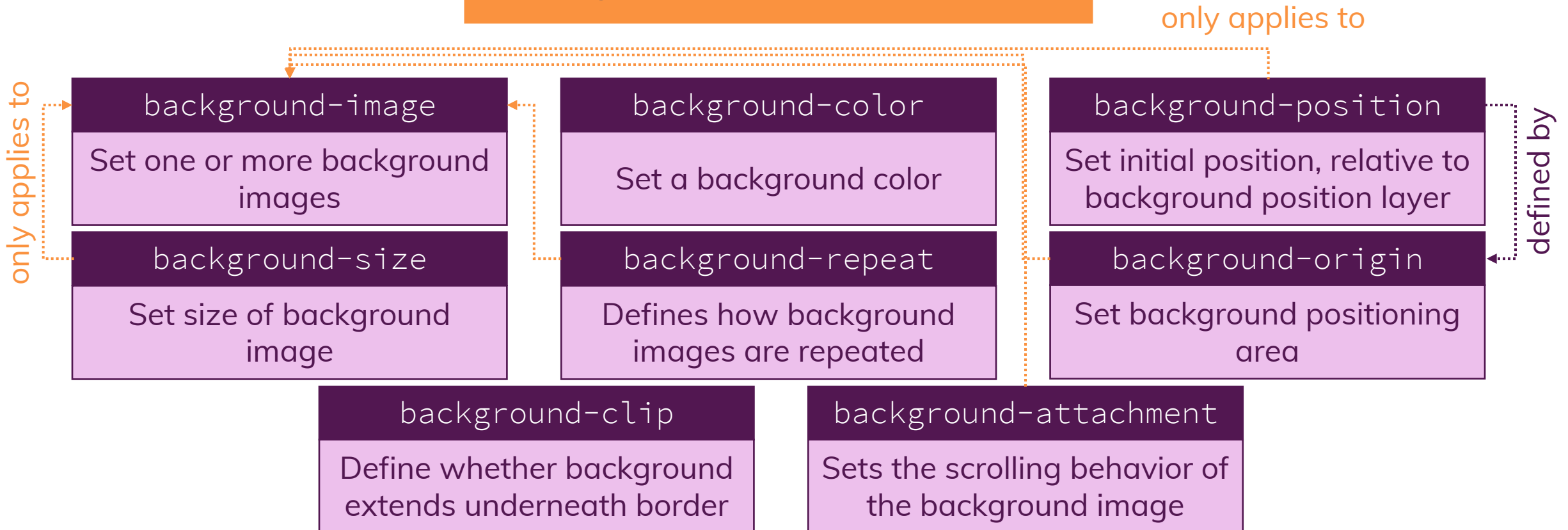
Z-Index

- Changes the default element positioning along the z-axis
- `auto (0)` as default value
- Changes only possible when `position` is applied
- Larger value = element is positioned on top of other elements

The Background Property

```
background: red;
```

background is a shorthand property



Summary

The “background” Property

background-image
background-color
background-position
background-size
background-origin
background-clip
background-repeat
background-attachment

Gradients

- Linear and radial gradients
- Linear gradients: Direction + color stops
- Radial gradients: shape, size, position and color stops

The “background” Shorthand

- Watch out for background-position and background-size (center/cover)
- As all shorthands: Overwrites other properties

Multiple Backgrounds

- You can stack background images (only one solid color which has to be at the bottom)
- Using transparency can create cool effects

Filters

- Easily add visual effects to boxes
- Affect all content

 vs background-image

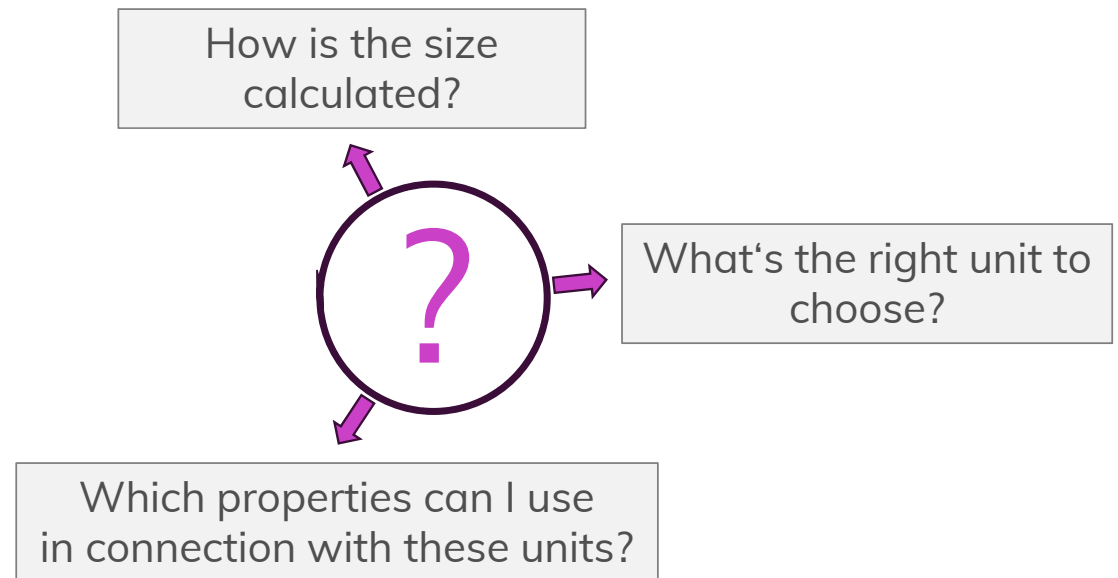
- is better for accessibility but way more difficult to style
- background-image can be sized and positioned easier

Dimensions, Sizes & Units

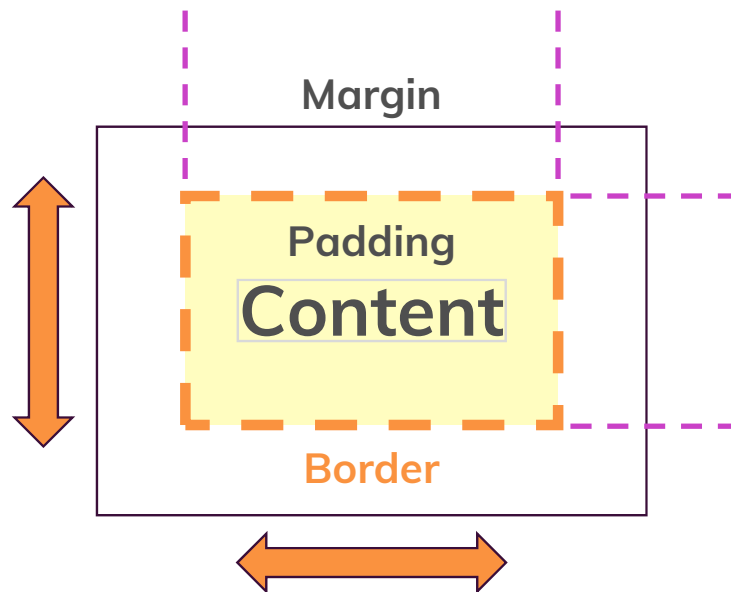
Because there is more than "px"

Pixels, Percentages & More

Units	
pixels	px
percentages	%
root em	rem
em	em
viewport height	vh
viewport width	vw



Where Units Matter



Which properties can I use?

font-size

padding

border

margin

width

height

top

bottom

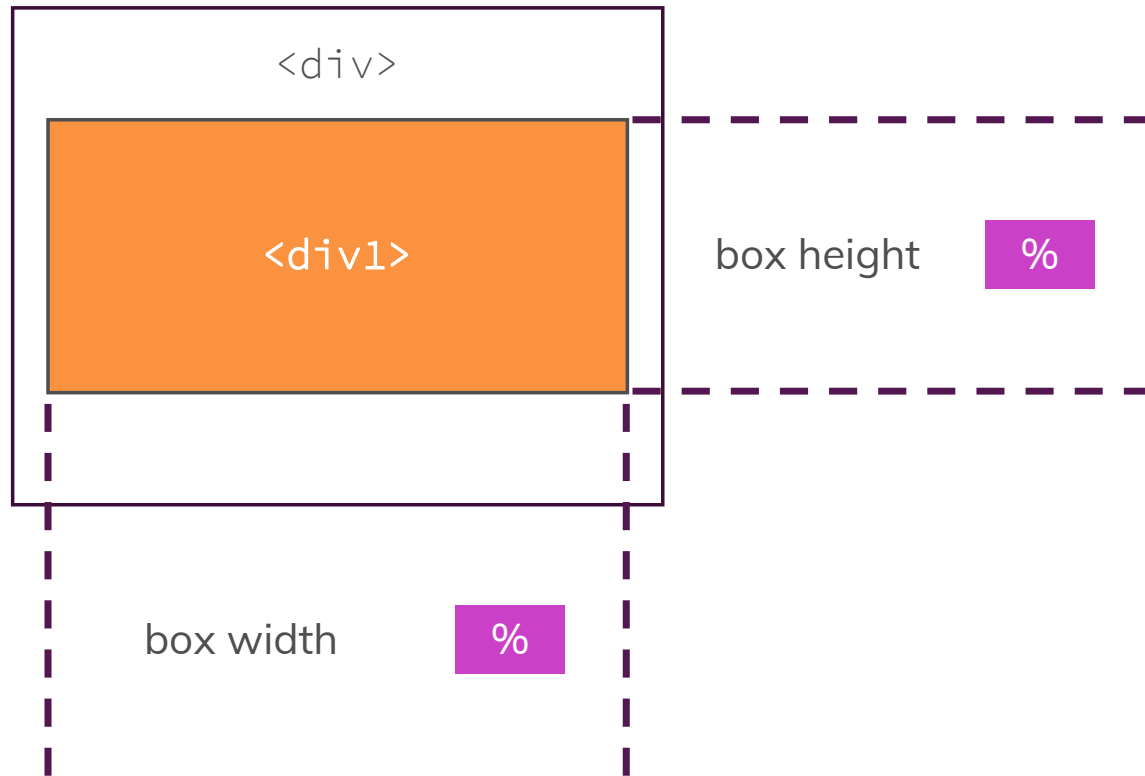
left

right

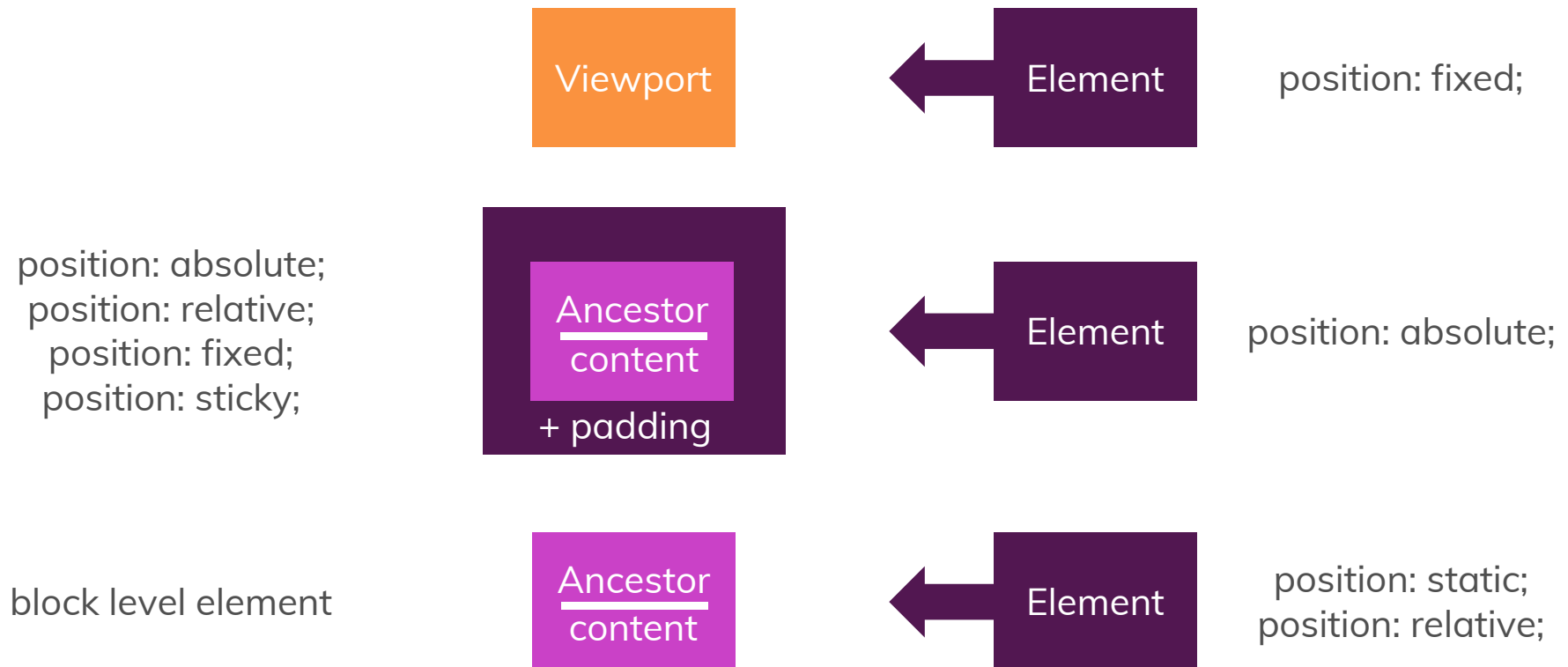
How is the Size Calculated?

Absolute Lengths	Viewport Lengths	Font-Relative Lengths
Mostly ignore user settings	Adjust to current viewport	Adjust to default font size
px	vh	rem
cm	vw	em
mm	vmin	...
...	vmax	
	%	
	Special Case	

How is the Box Size for % Units Calculated?



3 Rules to Remember



Which Unit Should I Choose?

Property			“Recommended” Unit			
font-size (root element)			%		-	
font-size			rem (em => Children only)			
padding	border	margin	rem	px	rem	
width		height	%	vw	%	vh
top		bottom	%		%	
left		right	%		%	

Summary

Units	The Containing Block	100% Height
pixels (px) percentages (%) rem & em viewport (vh % vw) auto	<ul style="list-style-type: none">• The reference point when applying % units to an element• Depends on the position property applied to this element• Can be the closest ancestor or the viewport	<ul style="list-style-type: none">• The element itself and the ancestors use position static/relative => 100% height is not working• Adding 100% height to all ancestors fixes this issue• Position fixed/absolute or using viewport units (vw or vh) as alternatives
Min/Max-Width	Em & Rem	
<ul style="list-style-type: none">• Always use these in combination with the width property• Set width to a relative value (e.g. %) and the min/max value to px to limit the element size• Also available for height	<ul style="list-style-type: none">• Sizes always depend on the font-size of the root element (rem) or the element itself (em)• Not restricted to font-size	

Responsive Design

Let's make our page look awesome on all devices

Which Tools do we Have?



Viewport

Adjust site to device viewport

No design changes



Media Queries

Change design depending on size

Design changes defined by us

Summary

Responsive Design

- Required to ensure that our website looks beautiful on all devices

The Viewport Metatag

- Should be added to your HTML files to adjust the viewport to device size
- Converts „hardware pixels“ into „software pixels“ and therefore takes into account the actual device width

Media Queries

- Allow us to change properties and therefore the entire design depending on device widths/heights
- Added to the CSS code with @media

Summary

Styling Inputs

- Input elements tend to have many browser default styles
- Use pseudo-selectors (`:focus`) to provide good user feedback
- `outline` vs `border`

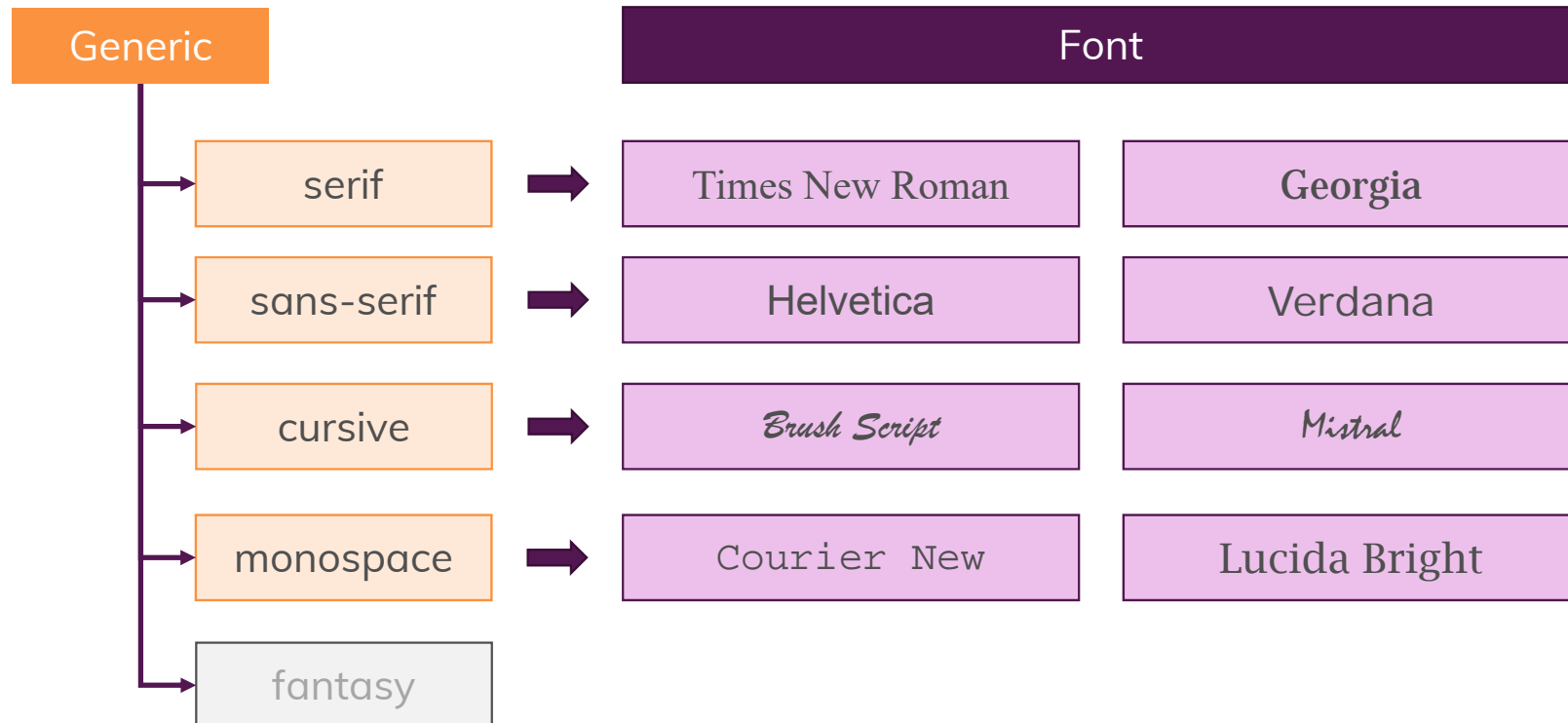
Validation Feedback

- `:valid` and `:invalid` pseudo selectors
- Manual validation feedback via class addition (e.g. `invalid`)

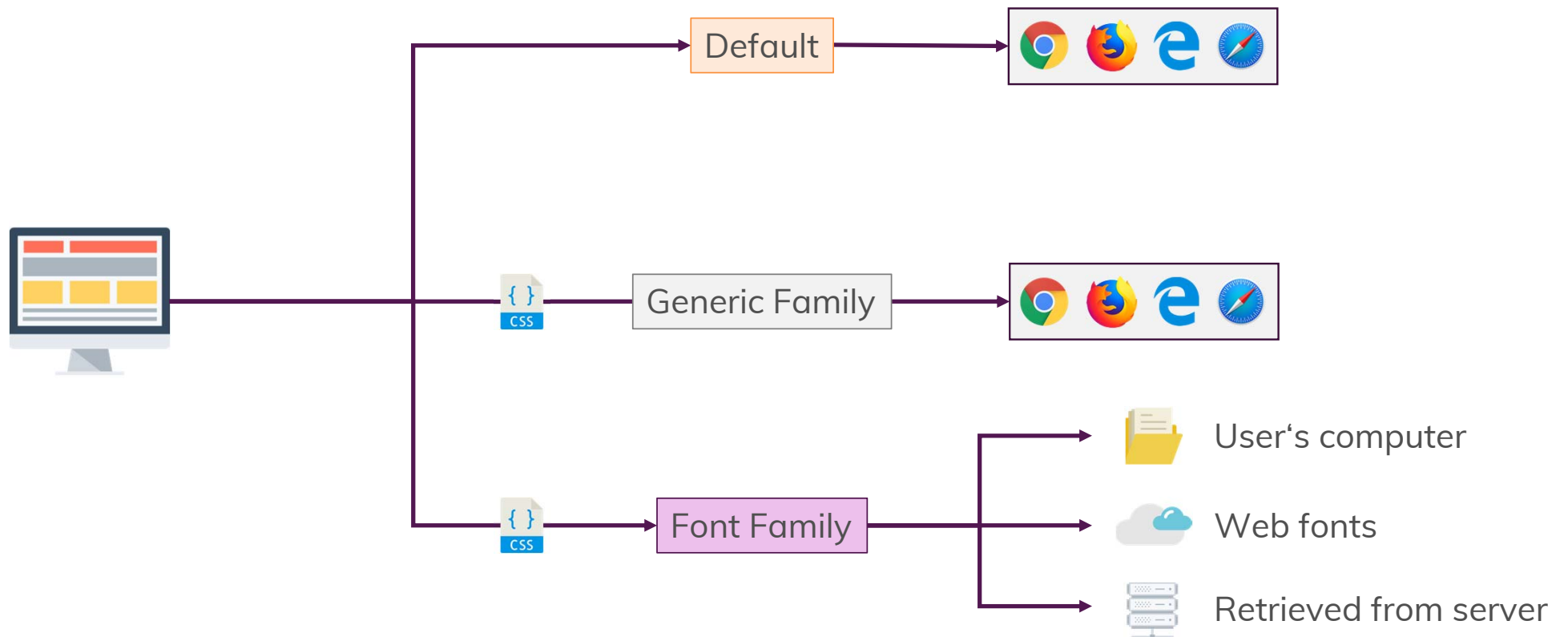
Working with Text and Fonts

How we can make our information look beautiful

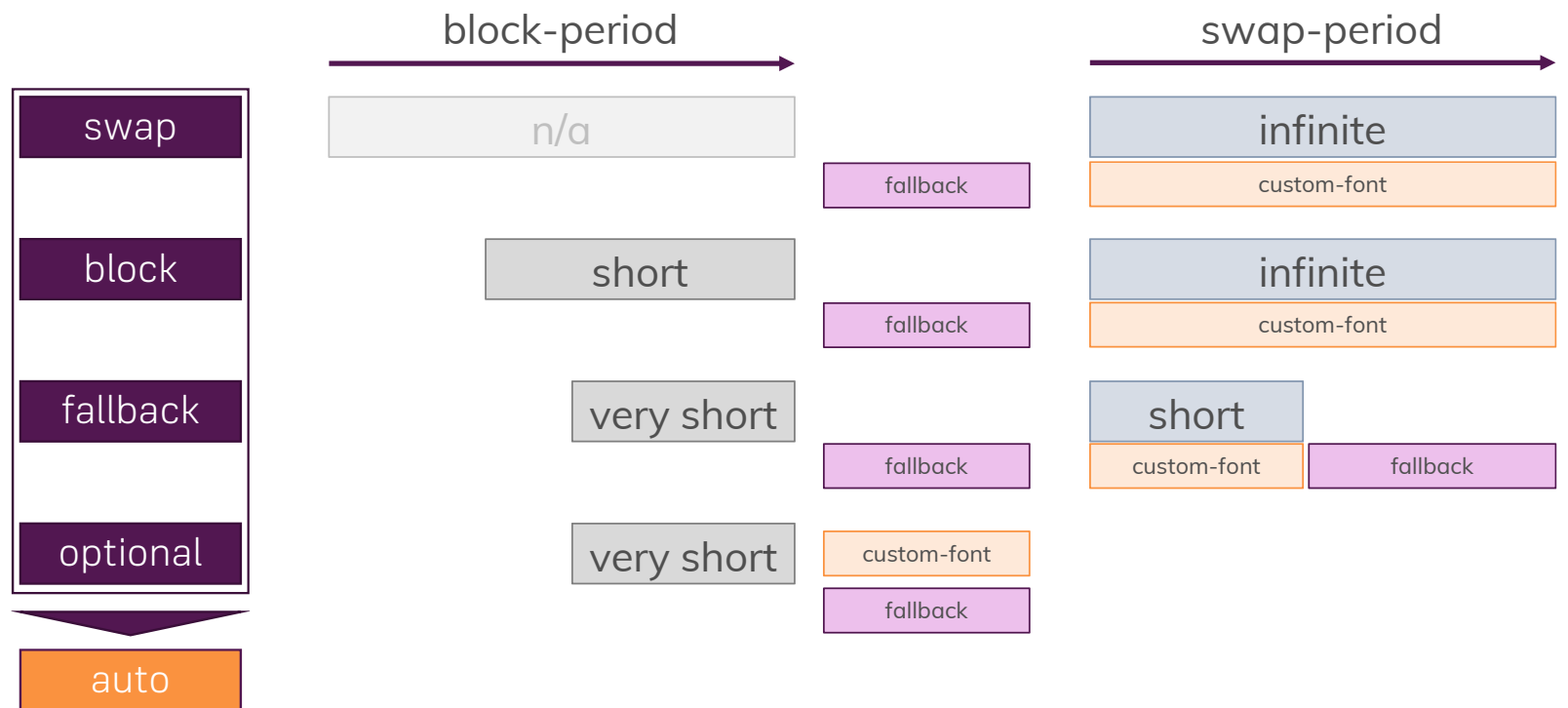
Generic Families & Font Families



What will be displayed?



font-display:



Summary

Generic & Font Families

- Generic families as fallback in case font family is not available
- Define exact font by using a specific font family

“font-display”

- Define the font family loading behaviour to ensure fonts are immediately visible for the user
- Available values mainly differentiate in block-period and swap-period

Importing Font Families

- Font families must be available to be displayed correctly on the browser
- Locally installed font families vs. embedded font-families with @font-face
- Import font families from Google Fonts

The “font” Shorthand

- Apply font family according to available systems fonts
- Shorthand for multiple font properties
- font-size & font-family are obligatory

The “font” Properties

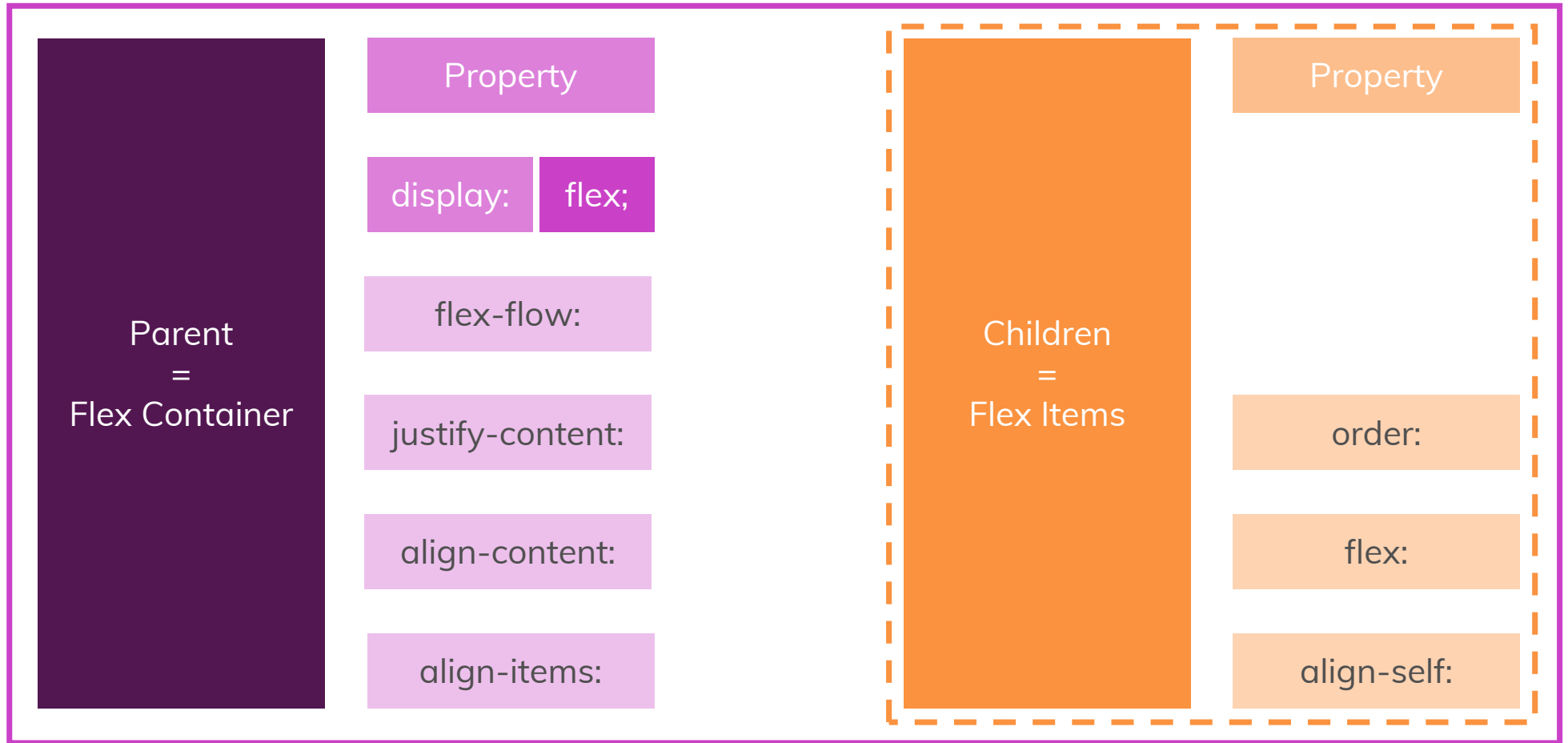
font-size
font-style
font-weight
font-stretch
font-variant

letter-spacing
white-space
line-height
text-decoration
text-shadow

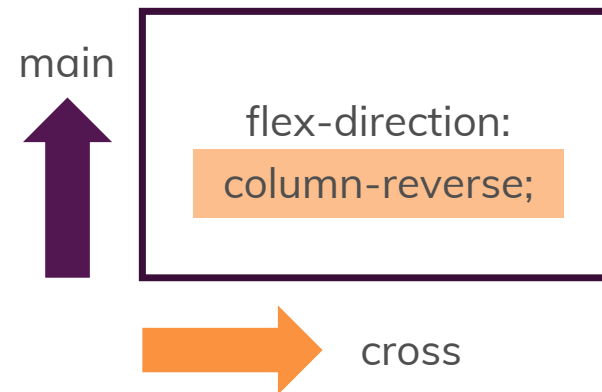
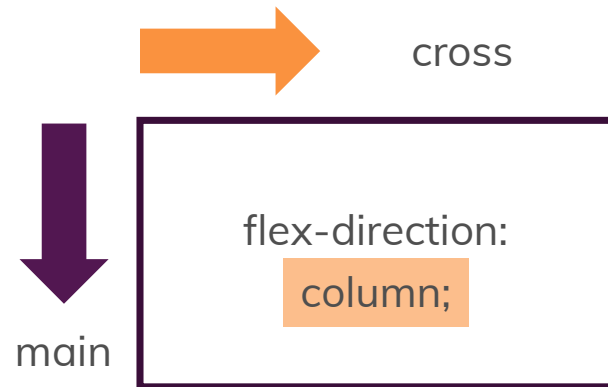
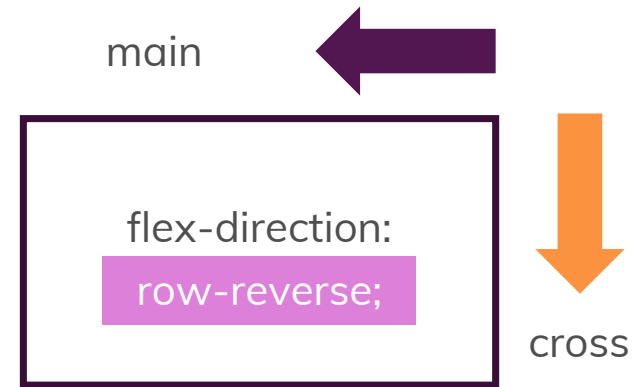
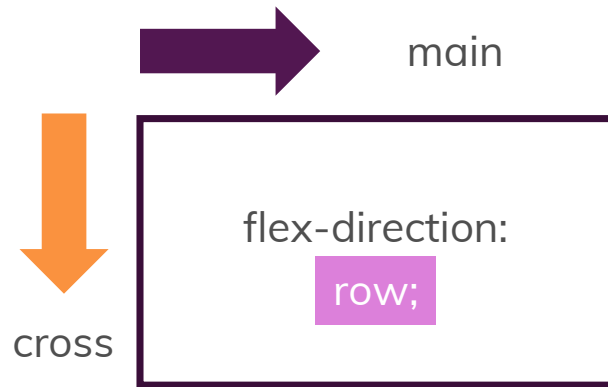
Working with Flexbox

The modern way to change the way
our elements are displayed

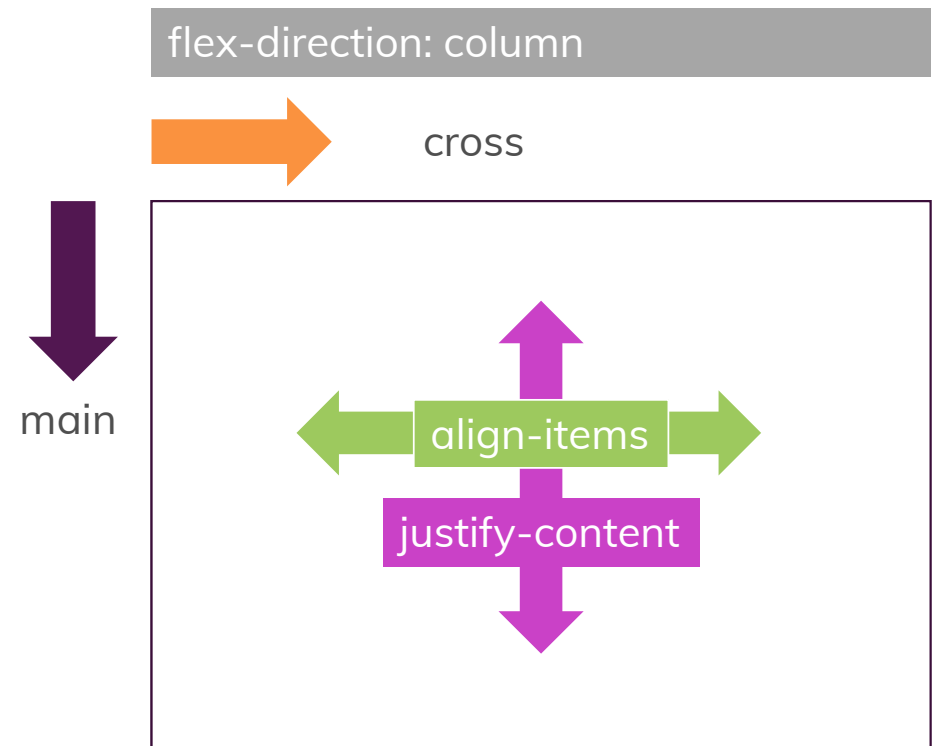
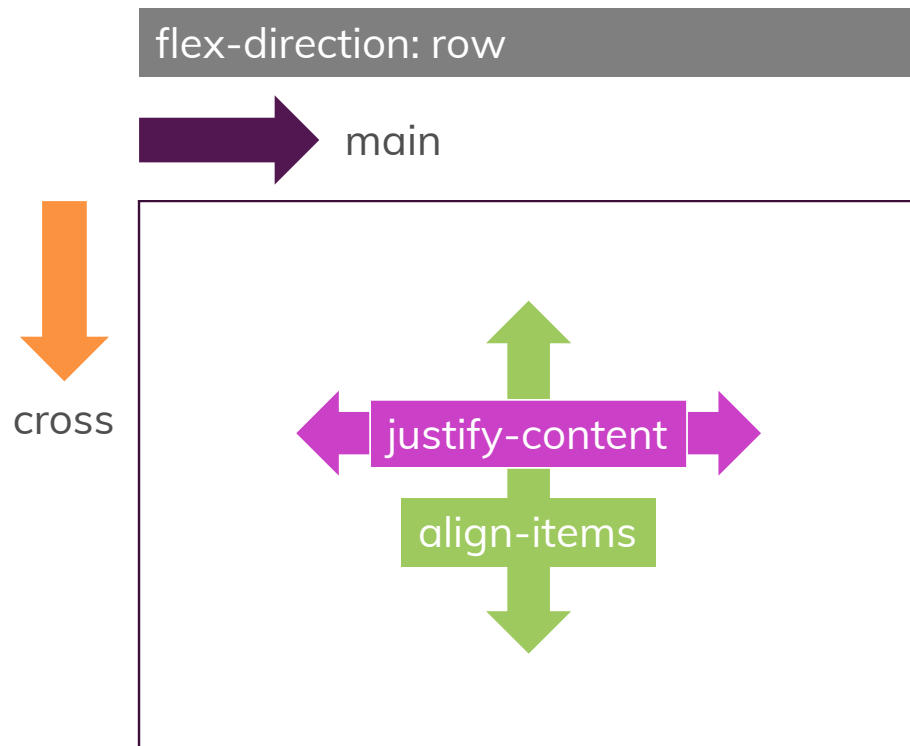
Understanding Flexbox



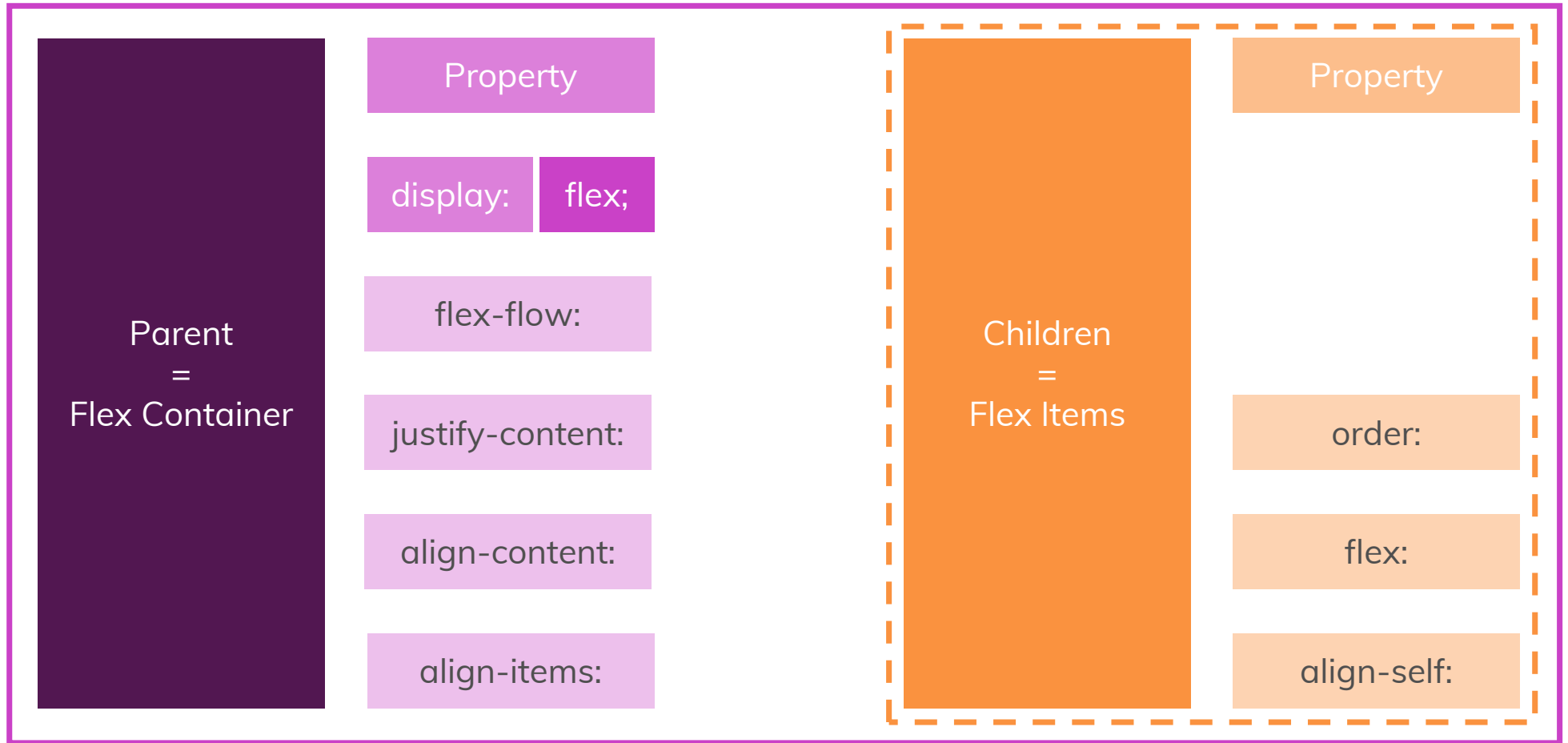
Main Axis vs. Cross Axis



Align Items and Justify Content



Understanding Flexbox



Summary

Flexbox	Flex Container	Flex Container - Properties
<ul style="list-style-type: none">• Changes the way elements are displayed on a website• Flexbox consists of the Flex-Container and Flex-Items	<ul style="list-style-type: none">• Adding <code>display: flex</code> to an element will turn it into a Flex-Container	<code>display: (inline-)flex</code> <code>flex-direction</code> <code>flex-wrap</code> <code>flex-flow</code> (shorthand) <code>align-items</code> <code>justify-content</code> <code>align-content</code>
Main Axis vs Cross Axis	Flex Items	Flex Items - Properties
<ul style="list-style-type: none">• <code>flex-direction</code> defines main axis• Properties refer to main or cross axis• Behaviour of Flex-Items changes depending on <code>flex-direction</code>	<ul style="list-style-type: none">• All elements/children of the Flex-Container will become Flex-Items• Behaviour can be changed by properties applied to the Flex-Container and applied to individual Flex-Items	<code>order</code> <code>align-self</code> <code>flex-grow</code> <code>flex-shrink</code> <code>flex-basis</code> <code>flex</code> (shorthand)

Grid Templates



```
.grid-container {  
  height: 140px;  
  display: grid;  
  grid-template-columns: [col-1-start] 150px [col-1-end] 180px auto;  
  grid-template-rows: auto [row-2-start] 100px [row-2-end];  
}
```

Turn <div> into a grid

Assign name to line

Assign width/height

150px

180px

auto

<unnamed line>

100px

row-2-start

auto

row-2-end

col-1-start

col-1-end

<unnamed line>

<unnamed line>



From a Grid Cell Perspective



```
.cell-1 {  
  grid-column-start: col-1-start;  
  grid-column-end: col-1-end;  
  grid-row-start: 1;  
  grid-row-end: 2;  
}
```

Shorthand: `grid-column: col-1-start / col-1-end`

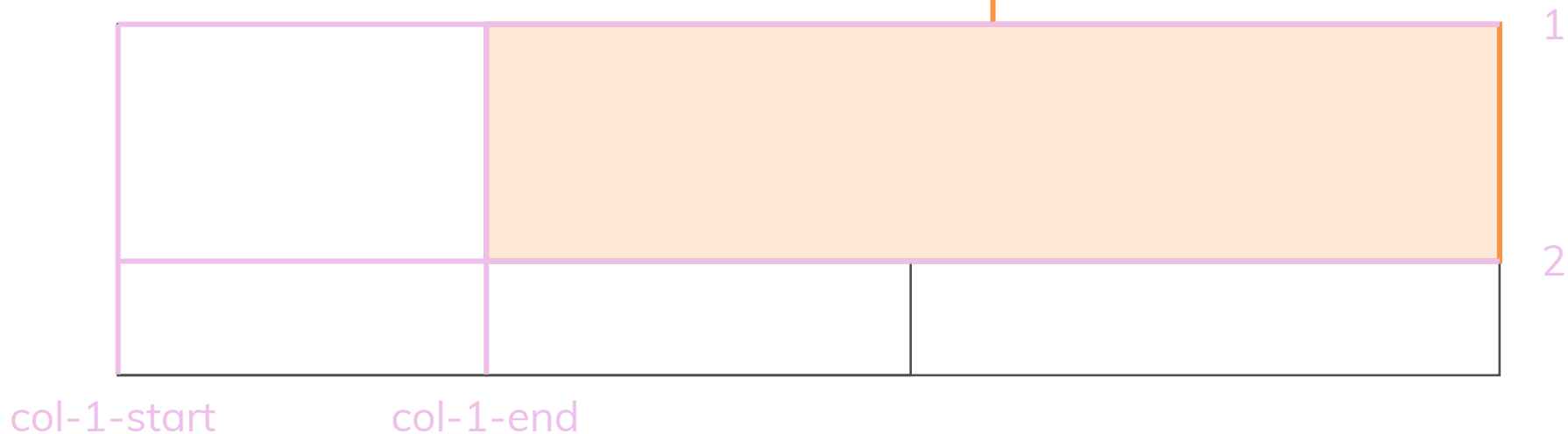
Refer to line names (if set) or line numbers



From a Grid Cell Perspective



```
.cell-2 {  
  grid-column-start: col-1-end;  
  grid-column-end: 4;  
  grid-row-start: 1;  
  grid-row-end: 2;  
}
```

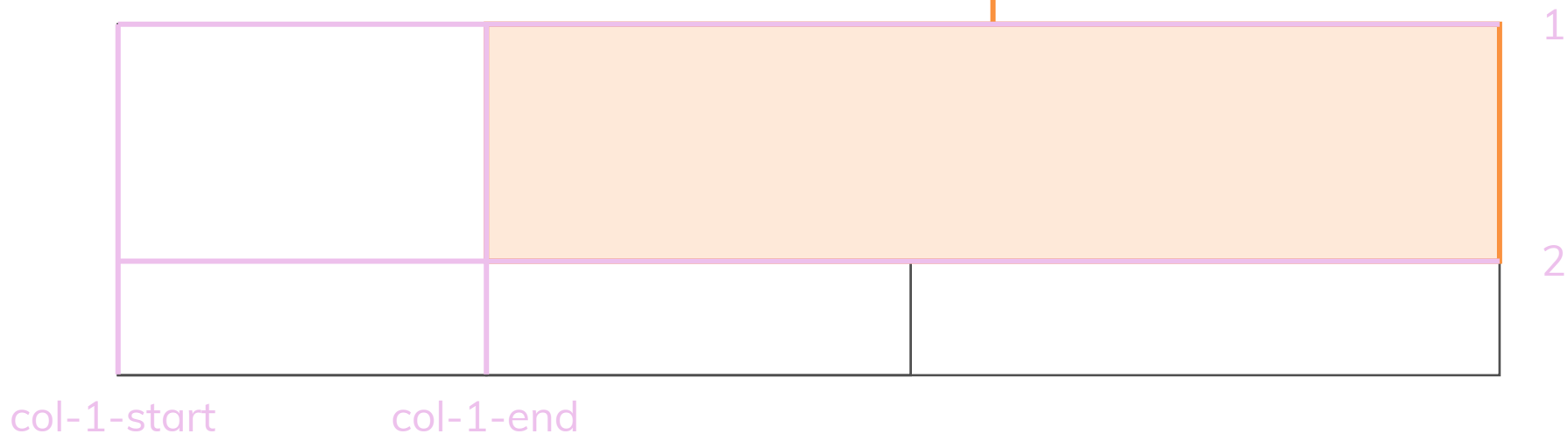


An Alternative Way



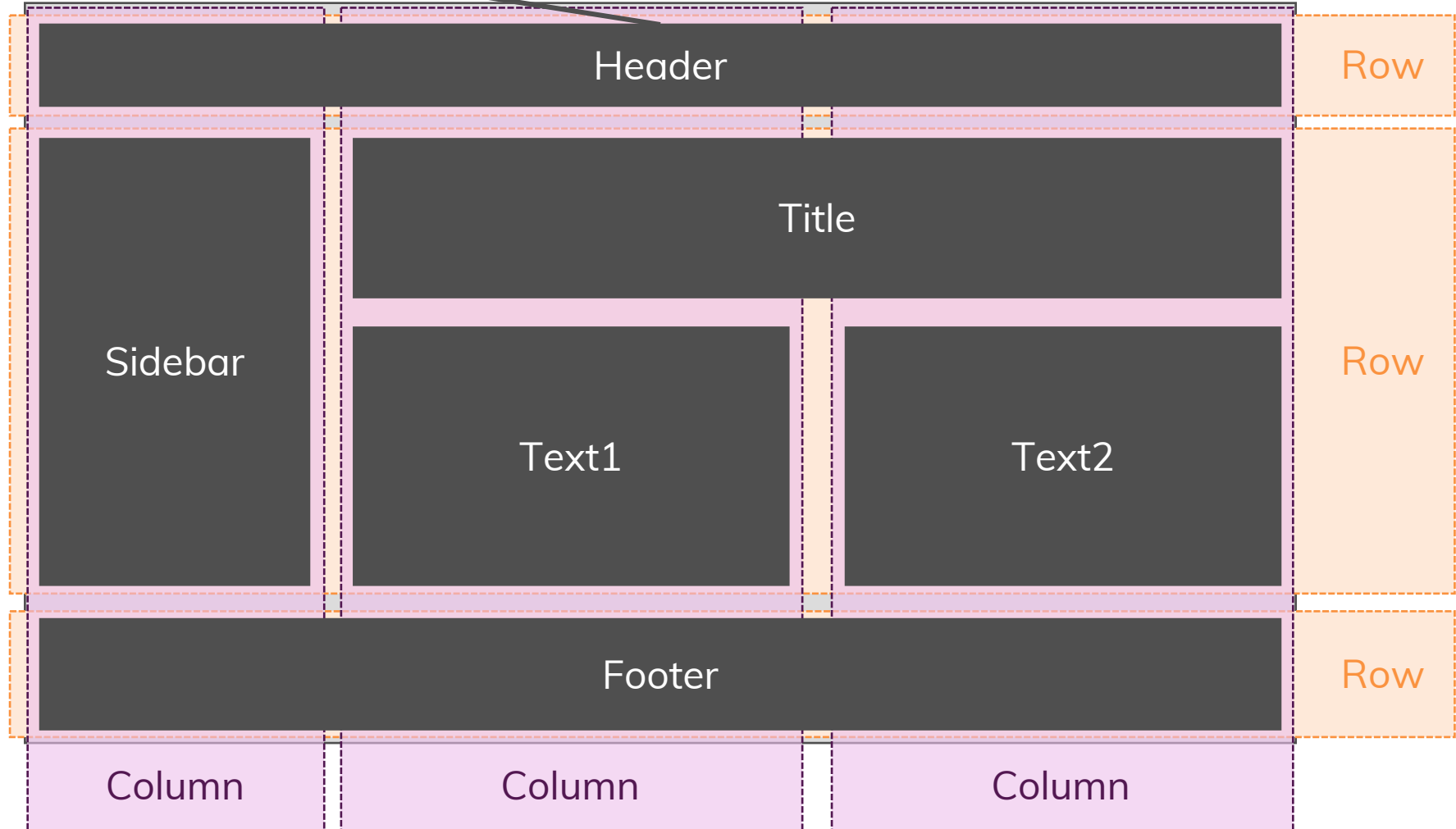
```
.cell-2 {  
  grid-column-start: col-1-end;  
  grid-column-end: span 2;  
  grid-row-start: 1;  
  grid-row-end: 2;  
}
```

By the Way: Overlapping is allowed, control
stacking via z-index



Grid Areas

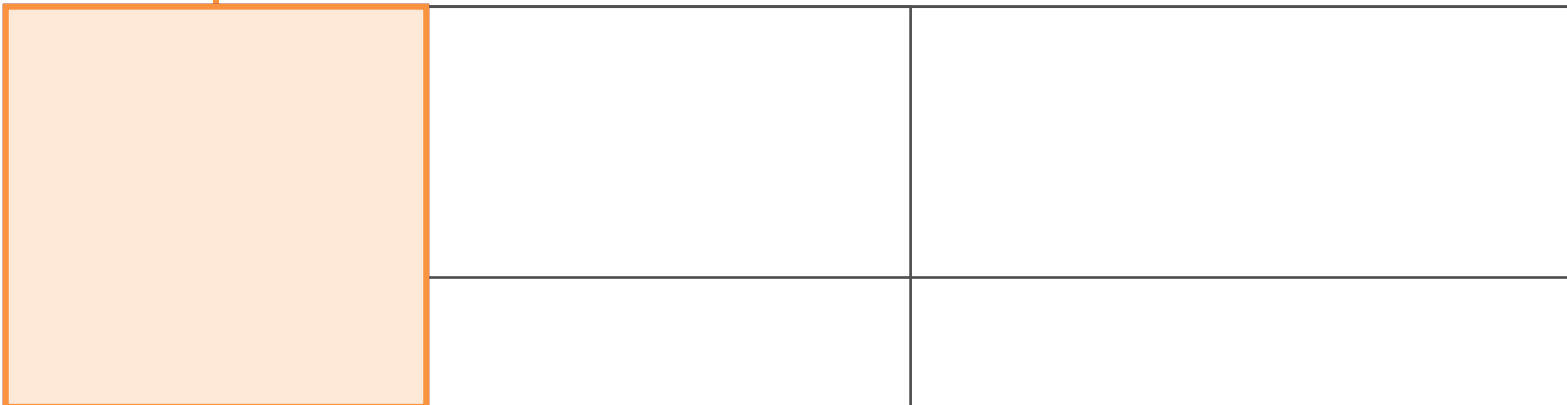
Header Area



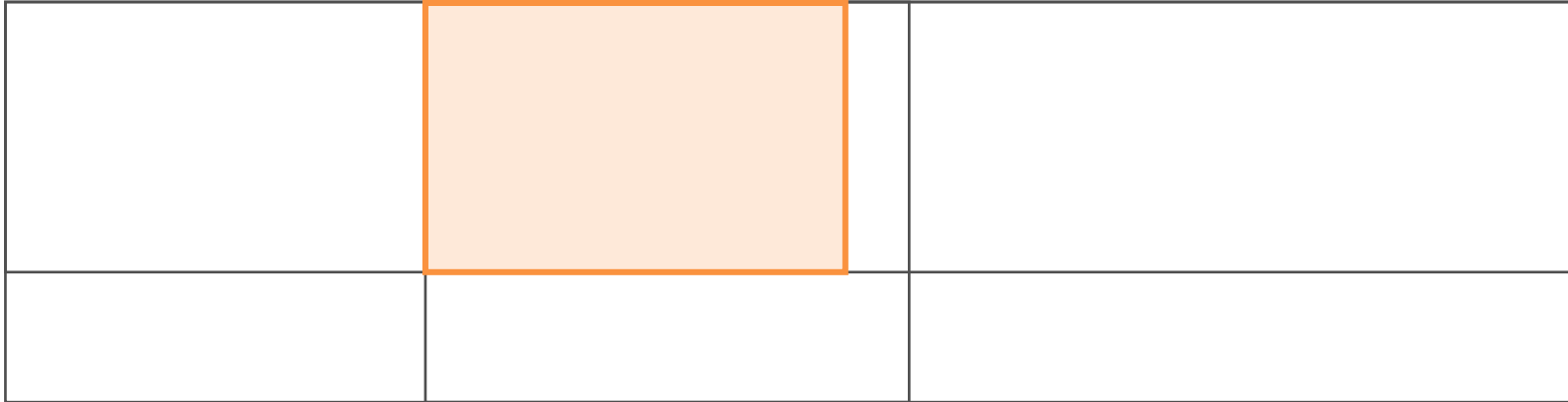
From a Grid Cell Perspective



```
.sidebar {  
  grid-area: sidebar;  
}
```

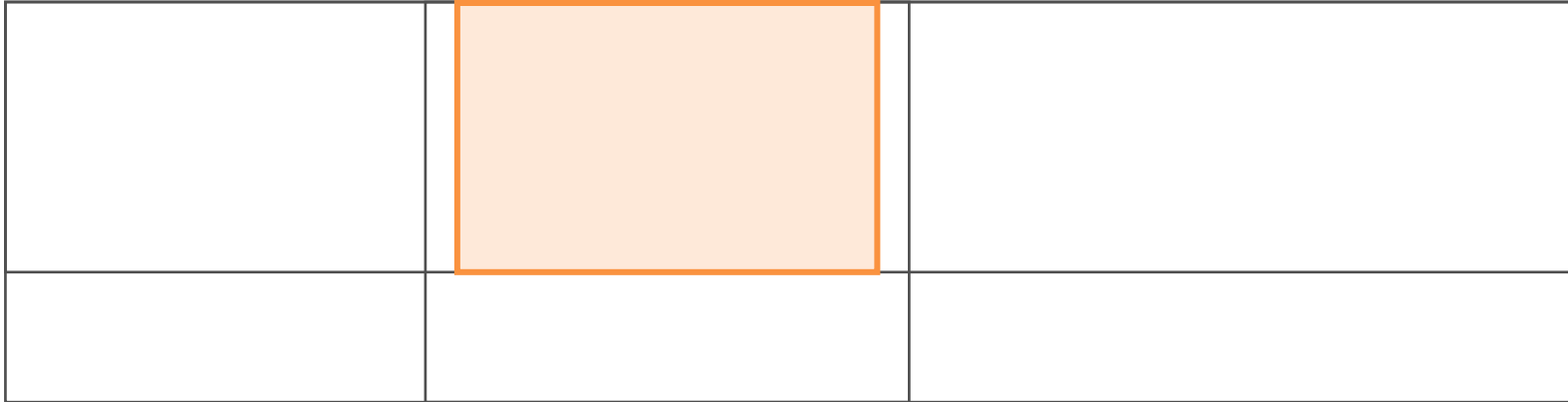


Grid Alignment – Horizontal Start



```
.grid-container {  
  justify-items: start;  
}
```

Grid Alignment - Horizontal Center

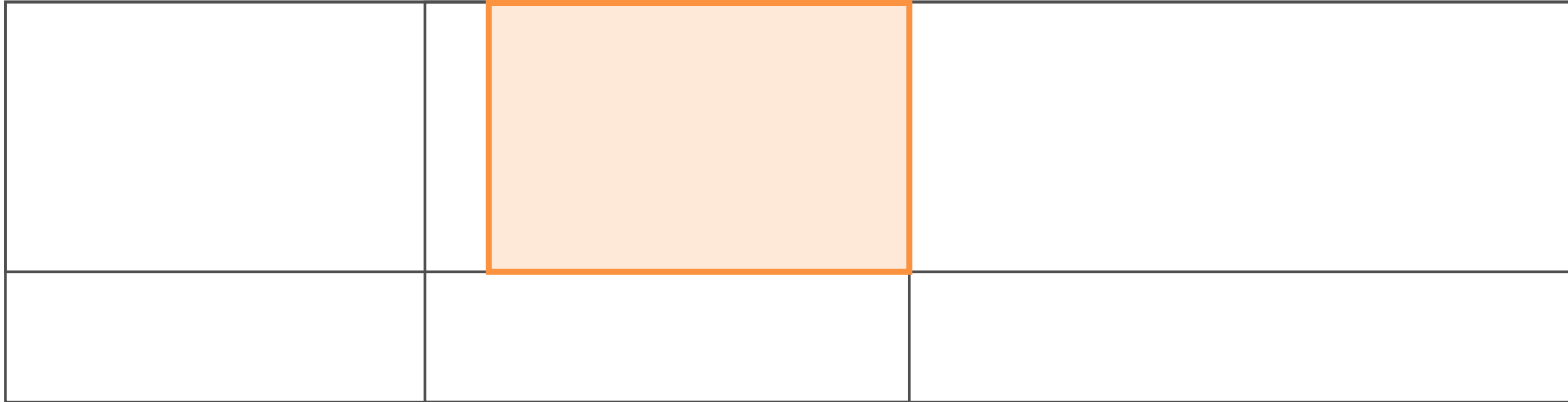


{ }

CSS

```
.grid-container {  
  justify-items: center;  
}
```

Grid Alignment - Horizontal End

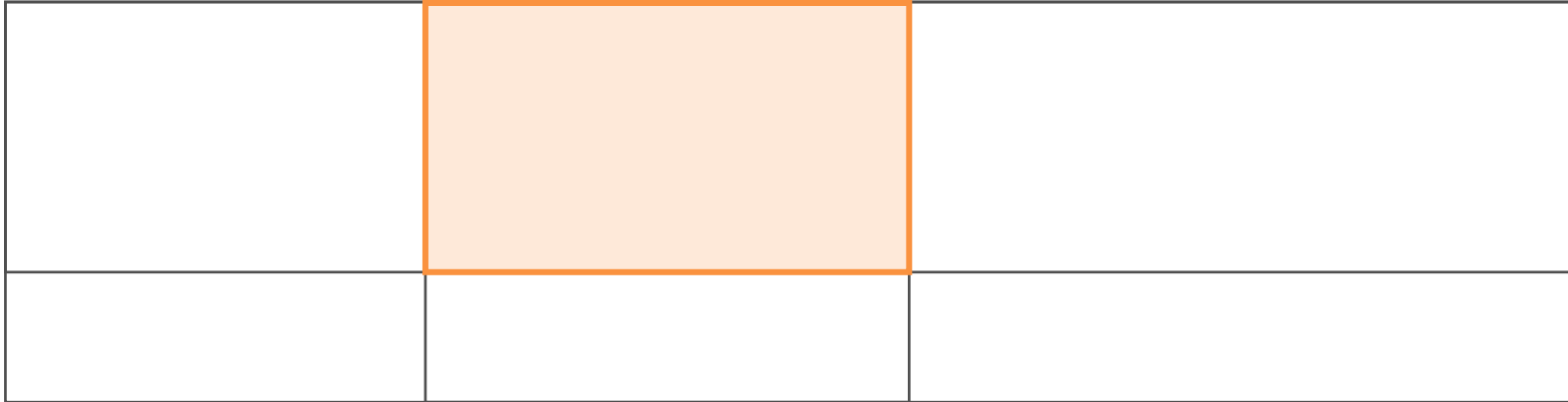


{ }

CSS

```
.grid-container {  
  justify-items: end;  
}
```


Grid Alignment - Horizontal Stretch



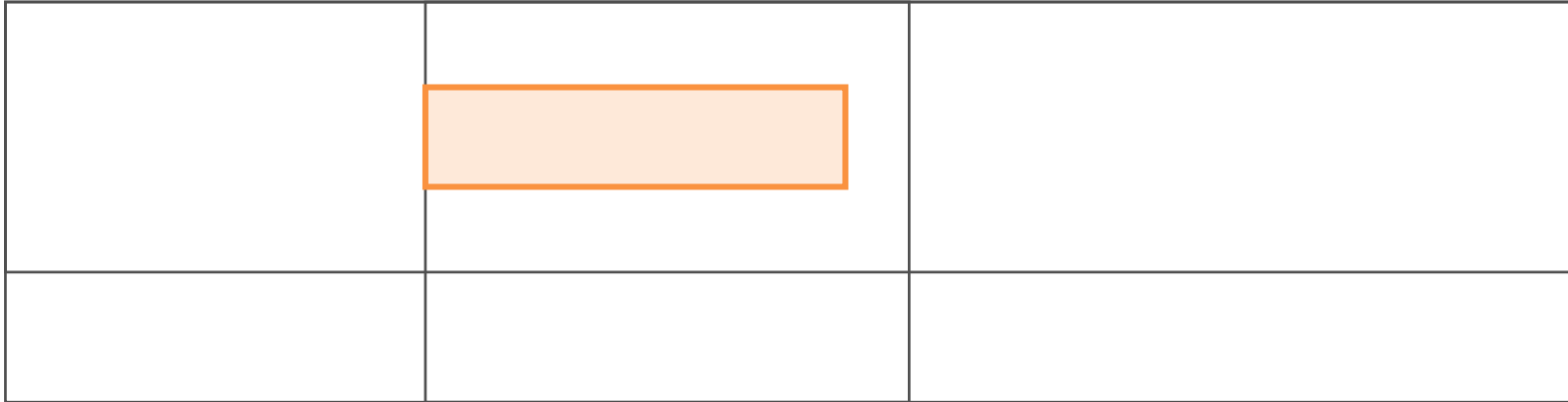
```
.grid-container {  
  justify-items: stretch;  
}
```

Grid Alignment – Vertical Start



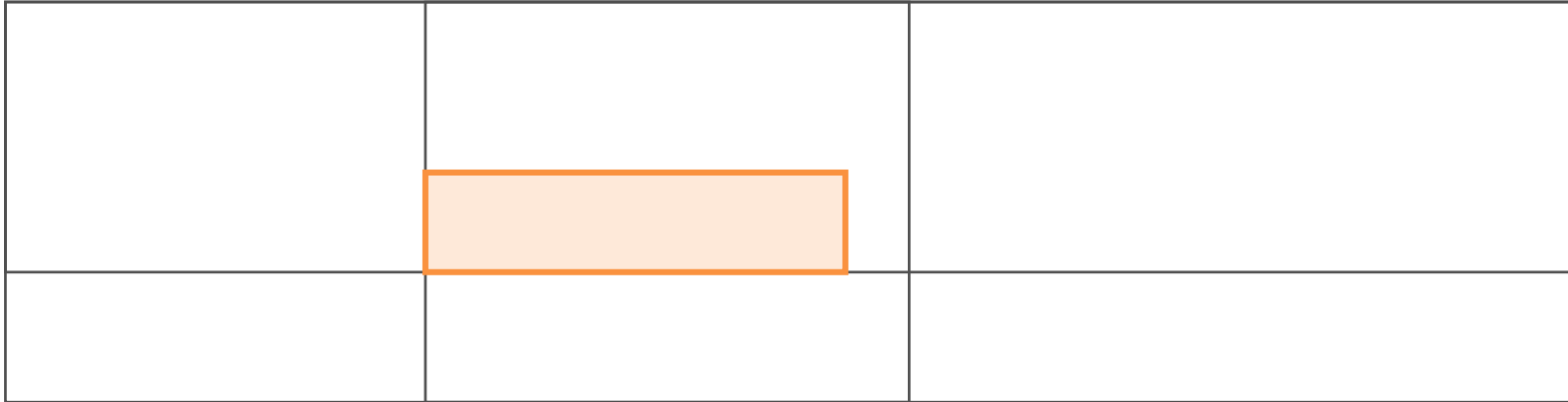
```
.grid-container {  
  align-items: start;  
}
```

Grid Alignment - Vertical Center



```
.grid-container {  
  align-items: center;  
}
```

Grid Alignment - Vertical End

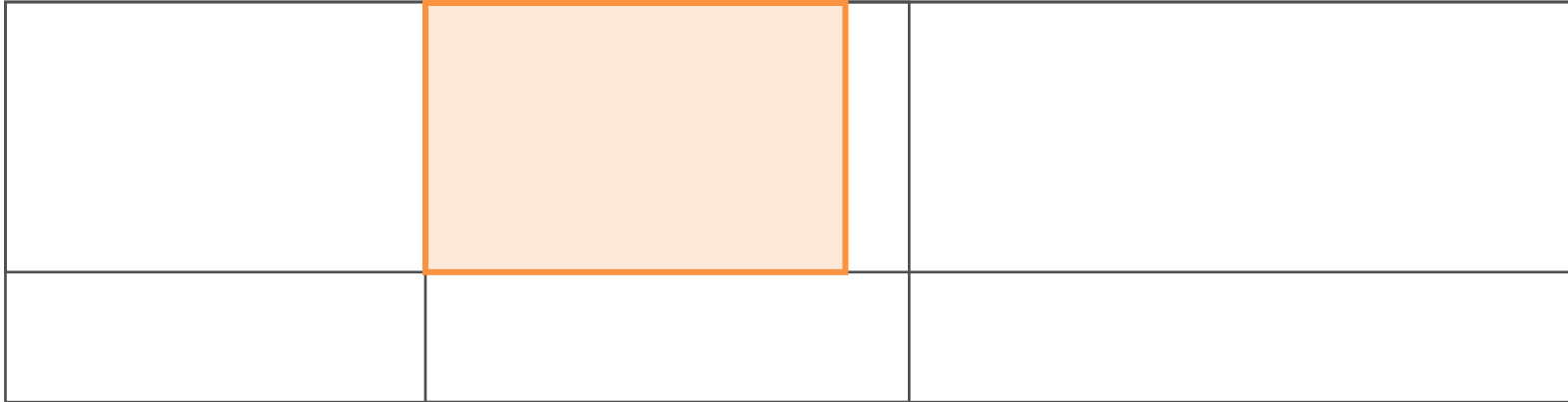


{ }

CSS

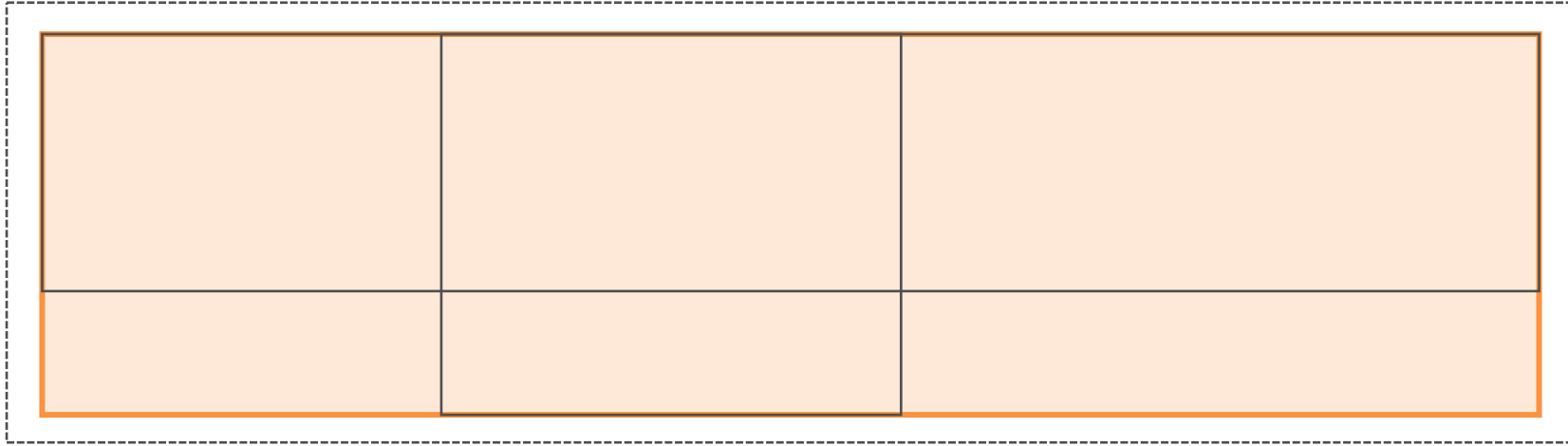
```
.grid-container {  
  align-items: end;  
}
```

Grid Alignment - Vertical Stretch



```
.grid-container {  
  align-items: stretch;  
}
```

Grid Alignment – Align Grid Itself

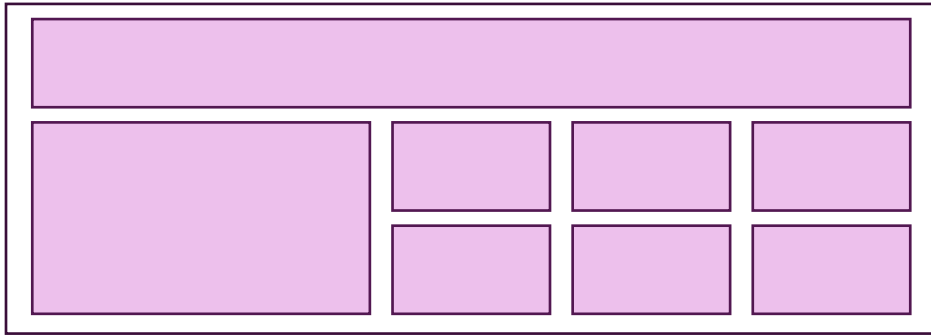


```
.grid-container {  
  justify-content: start | end | center | stretch | space-around | space-between | space-evenly;  
  align-content: start | end | center | stretch | space-around | space-between | space-evenly;  
}
```

CSS Grid vs Flexbox

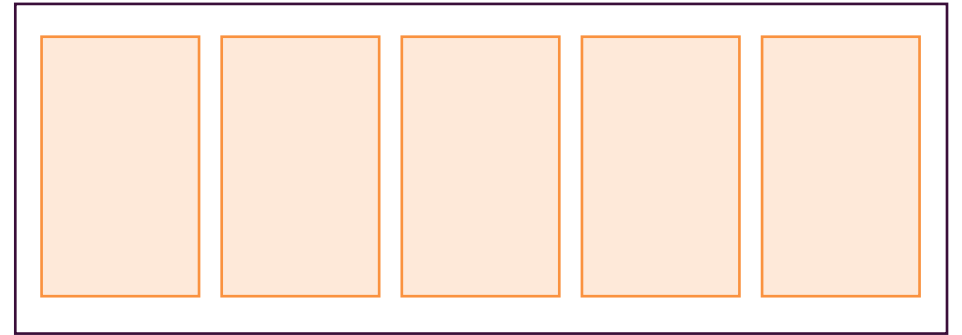
CSS Grid

Two-dimensional Positioning



CSS Flexbox

One-dimensional Positioning



Summary

Creating a Grid

- `display: grid` creates a grid where child elements are automatically placed in rows
- This default can be overwritten with `grid-auto-flow` (and then also `grid-auto-rows` or `grid-auto-columns`)
- Use `grid-gap` to add gaps between columns and rows

Defining the Grid Structure

- You define columns and/ or rows explicitly via `grid-template-columns/ grid-template-rows`
- Use `repeat(times, size)` to create multiple columns or rows with ease
- Use `auto-fill/ auto-fit` to derive the number of columns automatically
- Use `minmax` for dynamic sizing

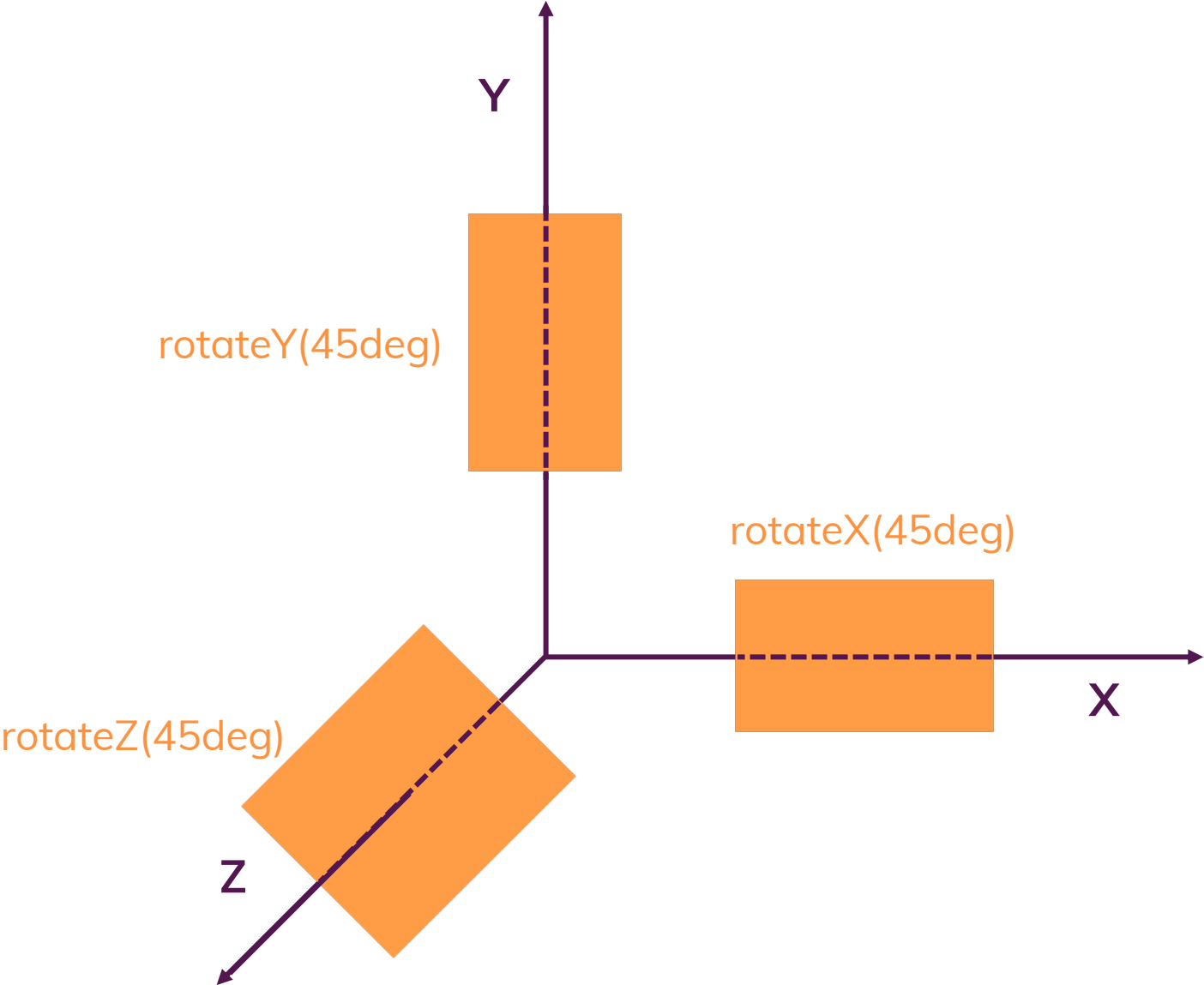
Placing Elements

- Position elements in the grid via `grid-row` and/ or `grid-column`
- Use `span X` to span an element over multiple columns or rows
- Use line numbers, line names or named areas

Aligning Elements

- Align grid items via `justify-items` (X-axis) and `align-items` (Y-axis)
- Align the entire grid content via `justify-content` (X-axis) and `align-content` (Y-axis)

Rotate



Summary

The “transform” Property

- Allows you to `translate()`, `scale()`, `rotate()` and `skew()` elements
- 3D transformations are possible via the Z-axis
- `transform-origin` and `transform-style` for customization

Perspective

- `perspective` allows you to define the perspective of the viewer
- `perspective-origin` allows you to manipulate the origin of the viewer

Summary

Keyframes

- Define all animation steps on your own: Via `from` and `to` or `%` values
- Animate as many properties as you want
- Animate different properties in each keyframe step
- Timing function interpolates transition between keyframes

The “animation” Property

- Define which keyframe set should be played
- Set a duration and delay (if wanted)
- Define how many iterations should be played and if the animation should alternate or not
- Set the `animation-fill-mode` to decide whether the properties of the last keyframe should be kept
- Listen to animation events via JavaScript

CSS Variables



```
.element-1 {  
  color: #fa923f;  
}  
  
.element-2 {  
  color: #fa923f;  
}  
  
.element-3 {  
  color: #fa923f;  
}
```

CSS Variables



```
:root {  
  --my-color: #fa923f;  
}  
  
.element-1 {  
  color: var(--my-color);  
}  
  
.element-2 {  
  color: var(--my-color);  
}  
  
.element-3 {  
  color: var(--my-color, #fa923f);  
}
```

Vendor Prefixes



Browsers implement new Features Differently and at different Speed



```
.container {  
  display: -webkit-box;  
  display: -ms-flexbox;  
  display: -webkit-flex;  
  display: flex;  
}
```

Support Queries

Some Features just aren't implemented (yet) in some Browsers



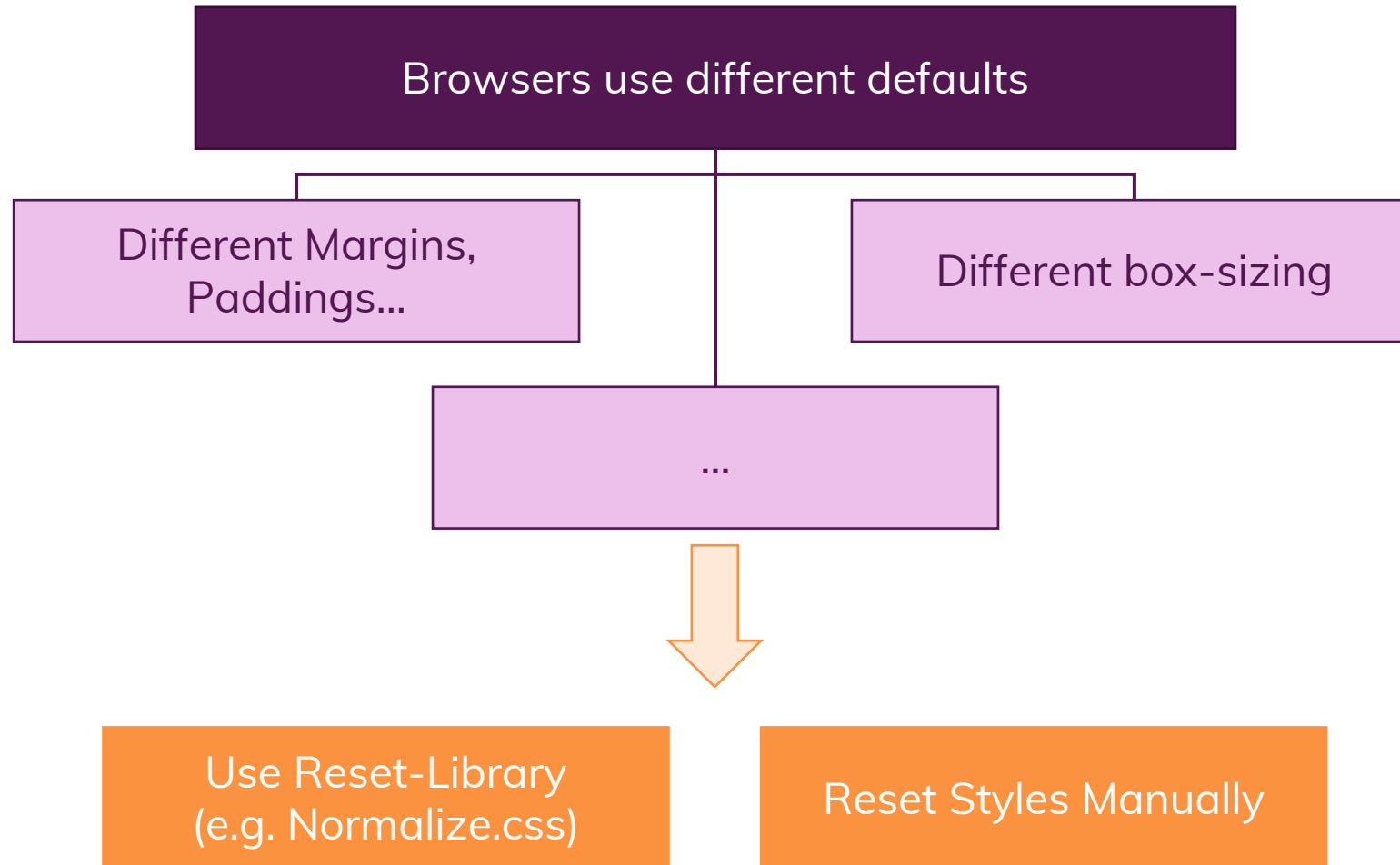
```
@supports (display: grid) {  
  .container {  
    display: grid;  
  }  
}
```

Polyfills

A **Polyfill** is a **JavaScript Package** which **enables certain CSS Features** in Browsers which would not support it otherwise.

Remember: Polyfills come at a cost! The JavaScript has to be loaded and parsed!

Eliminate Cross-Browser Inconsistencies



Choosing Class Names Correctly

Do

Use kebab-case

Because CSS is case-insensitive

Name by feature

For example `.page-title`

Don't

Use snakeCase

Because CSS is case-insensitive

Name by style

`.title-blue`

Block Element Modifier (BEM)

A uniform and consistent way of naming your CSS classes

.	BLOCK	--	ELEMENT	--	MODIFIER
---	-------	----	---------	----	----------

Example

.	menu-main	--	item	--	size-big
---	-----------	----	------	----	----------

Example

.	button	--		--	success
---	--------	----	--	----	---------

“Vanilla CSS” vs CSS Frameworks

Vanilla CSS



Write all your styles and layouts on your own

Component Frameworks



Foundation
Start here, build everywhere.



Bootstrap 4

Choose from a rich suite of pre-styled components & utility features/ classes

Utility Frameworks



Tailwind CSS

Build your own styles and layouts with the help of utility features and classes

“Vanilla CSS” vs CSS Frameworks

Vanilla CSS	Component Frameworks	Utility Frameworks
Full Control	Rapid Development	Faster Development
No unnecessary Code	Follow Best Practices	Follow Best Practices
Name Classes as you like	No Need to be an Expert	No Expert Knowledge Needed
Build everything from Scratch	No or Little Control	Little Control
Danger of “bad code”	Unnecessary Overhead Code	Unnecessary Overhead Code
	“All Websites Look the Same”	

Summary

CSS Variables

- `--your-name: 1rem;`
- Define values once, use them multiple times
- Only supported in modern browsers

Naming CSS Classes

- Use kebab-case (e.g. `page-title`) and name classes by feature not by style (e.g. `title-blue`)
- Avoid class name collisions, for example by using BEM class names

Cross-Browser Support

- Browser implement new features differently and with different speed
- Use vendor-prefixes to use cutting-edge features AND support older browsers (partly)
- `@supports` allows you to check for feature-support before using a property
- Polyfills can enable some CSS features which wouldn't work otherwise
- Consider normalizing CSS defaults across browsers

Vanilla CSS vs Frameworks

- Writing all styles from scratch gives you full control but comes with more work and responsibility
- Component frameworks (e.g. Bootstrap 4) allow you to build web pages rapidly but with less control
- Utility frameworks can be a good compromise