web programming

html & css


oli


rec

agenda

1. HTML & CSS basics
2. CSS selectors
3. box model
4. values & units
5. layout
6. responsive design

1. HTML & CSS basics

what is HTML?

- **H**yper**T**ext **M**arkup **L**anguage
- it's a **markup language**
- it's **NOT a programming language**

“HTML is the code you use to **structure** your web page and its **content**.”

HTML document

```
<!DOCTYPE html>
<html>

   <head>
      <meta charset="utf-8">
      <title>...</title>
   </head>

   <body>
      ...
   </body>

</html>
```

HTML document

```
<!DOCTYPE html>
<html>

   <head>
      <meta charset="utf-8">
      <title>...</title>
   </head>

   <body>
      ...
   </body>

</html>
```

HTML document

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8">
    <title>...</title>
  </head>

  <body>
    ...
  </body>

</html>
```

HTML document

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8">
    <title>...</title>
  </head>

  <body>
    ...
  </body>

</html>
```

metadata
doesn't appear in the viewport of the browser

HTML document

```
<!DOCTYPE html>
<html>

   <head>
      <meta charset="utf-8">
      <title>...</title>
   </head>

   <body>
      ...
   </body>

</html>
```

content
renders in the viewport of the browser

HTML element

- **building block** of an HTML document
- **encloses** content to make it appear a certain way, e.g. a paragraph of text
  - can have attributes
  - can be self-closing
  - can contain other elements

HTML element – anatomy

self-closing elements

```
<img src="path/to/image" alt="My first image" />
```

nested elements

```
<p>My <strong>first</strong> paragraph.</p>
```

some HTML elements

Headings (six levels, `<h1>-<h6>`)	`<h1>My first heading</h1>`
Paragraph	`<p>My first paragraph.</p>`
Line break	`My first line ` `break!`
Image	`<img src="path/to/image" />`
Link	`<a href="google.com">My first link</a>`
Strong	`<strong>bold words!</strong>`

reference: https://developer.mozilla.org/en-US/docs/Web/HTML/Element

exercise

Web Programming

Our Team
Matías Olivera
Rodrigo Espinosa

Course Info

solution

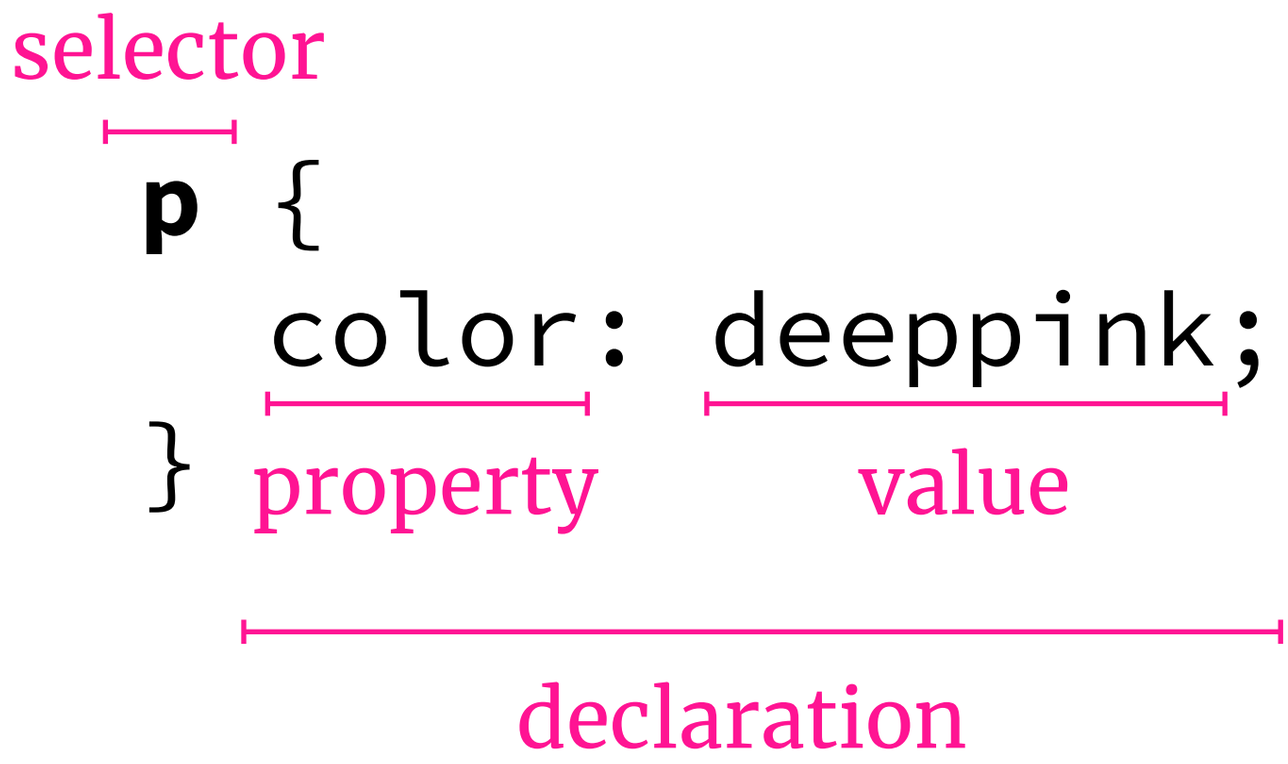
```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>Web Programming</title>
</head>

<body>
    <h1>Web Programming</h1>
    <p>
        <strong>Our Team</strong><br />
        Matías Olivera<br />
        Rodrigo Espinosa
    </p>
    <a href="https://github.com/ucudal-wp/web-programming">
        Course Info
    </a>
</body>

</html>
```

what is CSS?

- **C**ascading **S**tyle **S**heets
- it's **NOT a programming language**
- it's **NOT a markup language** either
- it's a **style sheet language**

“CSS is the code you use to define the **appearance** and **layout** of your webpage.”

CSS ruleset – anatomy



multiple elements

```
h1,
p {
   color: deeppink;
}
```

reference stylesheet

```
<!DOCTYPE html>
<html>

   <head>
      ...
      <link rel="stylesheet" href="filename.css">
   </head>

   <body>
      ...
   </body>

</html>
```

some CSS properties

Text color	`color: deeppink;`
Background color	`background-color: aquamarine;`
Text alignment	`text-align: center;`
Border	`border: 3px solid deeppink;`

reference: https://developer.mozilla.org/en-US/docs/Web/CSS/Reference

CSS colors

- everything in HTML can have colors
- different ways of representing them:
  - keyword, e.g. `aquamarine`
  - hexadecimal, e.g. `#7fffd4`
  - RGB functional, e.g. `rgb(127, 255, 212)`
    `rgba(0, 0, 0, .5)`
  - HSL functional, e.g. `hsl(160deg, 100%, 75%)`

pick your team's color!

goo.gl/SLf5jJ

exercise

Web Programming

Aquamarine Team
Matías Olivera
Rodrigo Espinosa

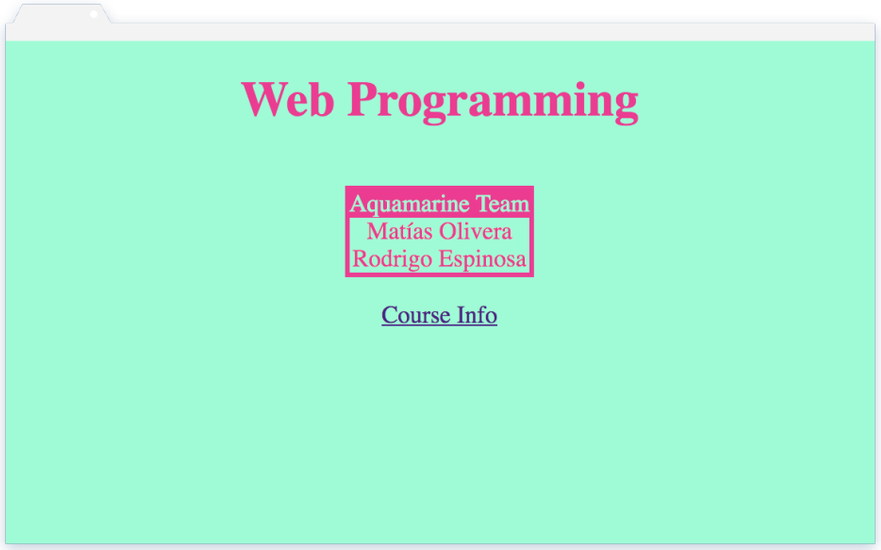
Course Info

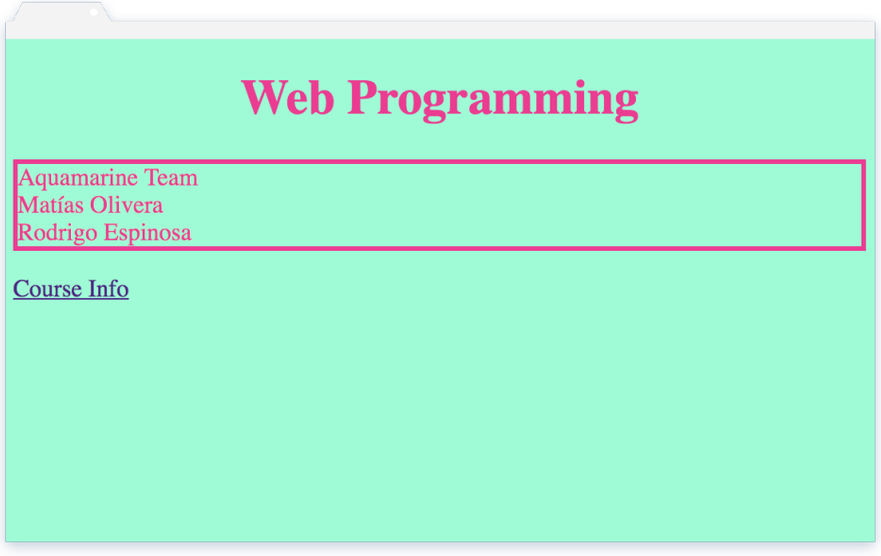
solution? 🤔

```
body {
  background-color: aquamarine;
  color: deeppink;
}

h1 {
  text-align: center;
}

p {
  border: 3px solid deeppink;
}

a {
  text-align: center;
}
```

solution? nope 😔

```
body {
  background-color: aquamarine;
  color: deeppink;
}

h1 {
  text-align: center;
}

p {
  border: 3px solid deeppink;
}

a {
  text-align: center;
}
```





weirdnesses

- the box is not centered!
- the box is too wide!
- why is the link not centered!?
- how do I highlight the title of the box?
- you know what? I'm done! 😡

wait!

block vs. inline 😏

- HTML elements are usually either "block-level" or "inline-level"
  - **block**: `<p>`, `<h1>`, …
  - **inline**: `<a>`, `<strong>`, …
    - inline-block: `<img />`

key differences

block	inline
begins on a new line	can start anywhere in a line
takes up the full width available	takes up as much width as necessary
flows top to bottom	flows left to right
can set width and height	**cannot** set width nor height
can contain inline and block elements	**cannot** contain block elements

inline-block

- combines features from block and inline elements:
  - can start anywhere in a line (inline)
  - takes as much width as necessary (inline)
  - can set width and height (block)
  - can contain block elements (block)

CSS display

- every HTML element has a default display value depending on the type of element
- it can be changed with CSS using the `display` property
- possible values: `block`, `inline`, `inline-block`, and others

`div` and `span`

- generic HTML elements
- `div`:
  - block-level
  - often used as a container for other HTML elements
- `span`:
  - inline-level
  - often used as a container for some text

solution

```html
<head>
    ...
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <h1>Web Programming</h1>
    <p>
        <span>Aquamarine Team</span><br />
        Matías Olivera<br />
        Rodrigo Espinosa
    </p>
    <a href="...">Course Info</a>
</body>
```


```css
body {
  background-color: aquamarine;
  color: deeppink;
  text-align: center;
}

p {
  border: 3px solid deeppink;
  display: inline-block;
}

span {
  background-color: deeppink;
  color: aquamarine;
}

a {
  display: block;
}
```

2. CSS selectors

`class` and `id`

- **global attributes** – can be used on all HTML elements
- **class**:
    - used on one or more elements
    - identifies a **group** of elements
    - multiple classes per element, e.g. `class="foo bar"`
- **id**:
    - used on **exactly one** element per page
    - identifies a **unique** element

CSS selectors

- defines the elements to which a CSS ruleset applies
- three main selectors:
  - **type** (element): selects all the elements of the given type, e.g. **`p`**
  - **class**: selects all the elements that have the given `class` attribute, e.g. **`.foo`**
  - **id**: selects an element based on the value of its `id` attribute, e.g. **`#foo`**

other selectors (combinators)

- **descendant**: selects elements that are descendants of the first element, e.g. **`div span`**
- **child**: selects elements that are direct children of the first element, e.g. **`ul > li`**
- **general sibling**: selects elements that follow the first element and share the same parent, e.g. **`p ~ span`**
- **adjacent sibling**: selects elements that directly follow the first element and share the same parent, e.g. **`h2 + p`**

summary

type (element)	`p`	all `<p>` elements
class	`.foo`	all elements with `class="foo"`
id	`#foo`	the element with `id="foo"`
descendant	`div span`	all `<span>` elements that are inside a `<div>`
child	`ul > li`	all `<li>` elements that are nested directly inside a `<ul>`
general sibling	`p ~ span`	all `<span>` elements that follow a `<p>`
adjacent sibling	`h2 + p`	all `<p>` elements that directly follow an `<h2>`

styles collision

- when styles collide, the most specific selector wins:
  - `div span { color: deeppink; }`
    `span { color: aquamarine; }`
- if there's a tie, the later selector wins:
  - `span { color: deeppink; }`
    `span { color: aquamarine; }`

styles collision

- when styles collide, the most specific selector wins:
  - ```
    div span { color: deeppink; }
    span { color: aquamarine; }
    ```
- if there's a tie, the later selector wins:
  - ```
    span { color: deeppink; }
    span { color: aquamarine; }
    ```

styles collision

- when styles collide, the most specific selector wins:
  - `div span { color: deeppink; }`
    `span { color: aquamarine; }`
- if there's a tie, the later selector wins:
  - `span { color: deeppink; }`
    `span { color: aquamarine; }`

specificity

type < class < id

inheritance

- styles are inherited from parent to child:
  - `body { text-align: center; }`
- but **not all properties are inherited**, e.g. `display`
- defined in the spec, check MDN

why isn't <a> colored? 🤔

```
body {
   color: deeppink;
}
```

Web Programming

Aquamarine Team
Matías Olivera
Rodrigo Espinosa

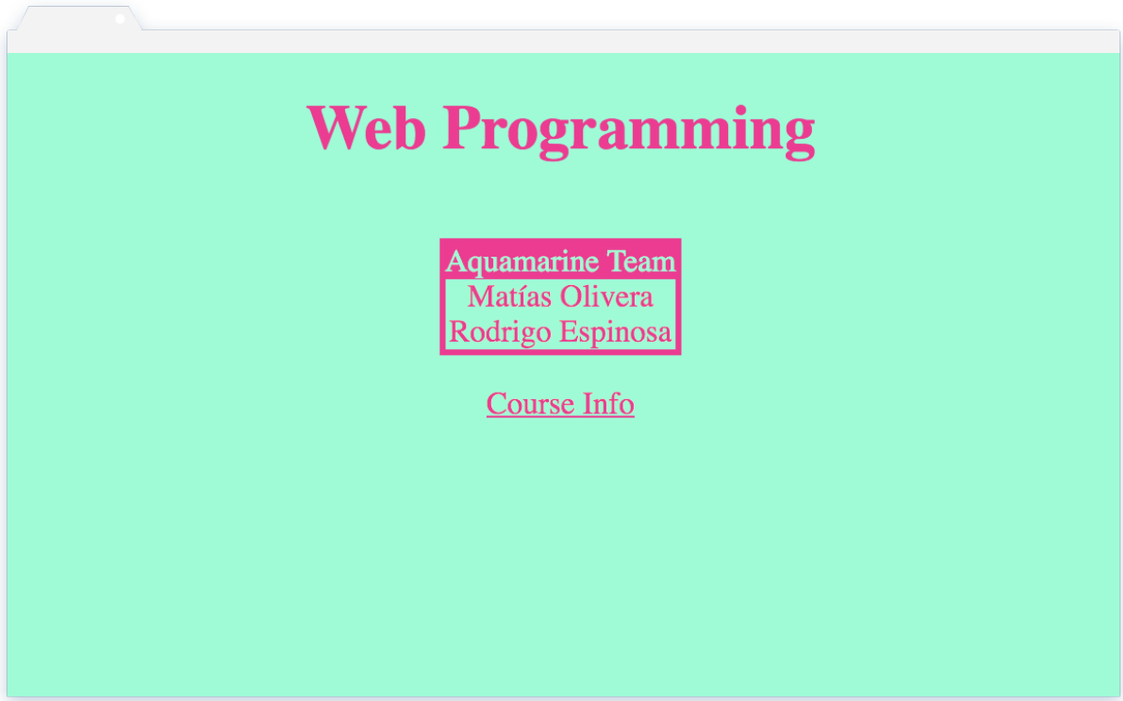
Course Info

user-agent stylesheets

- owned by the browser
- gives a default style to any document:
  - `<a>` colors, among other defaults, follow the W3C recommendation

overriding default styles

```
a {
   color: deeppink;
}
```

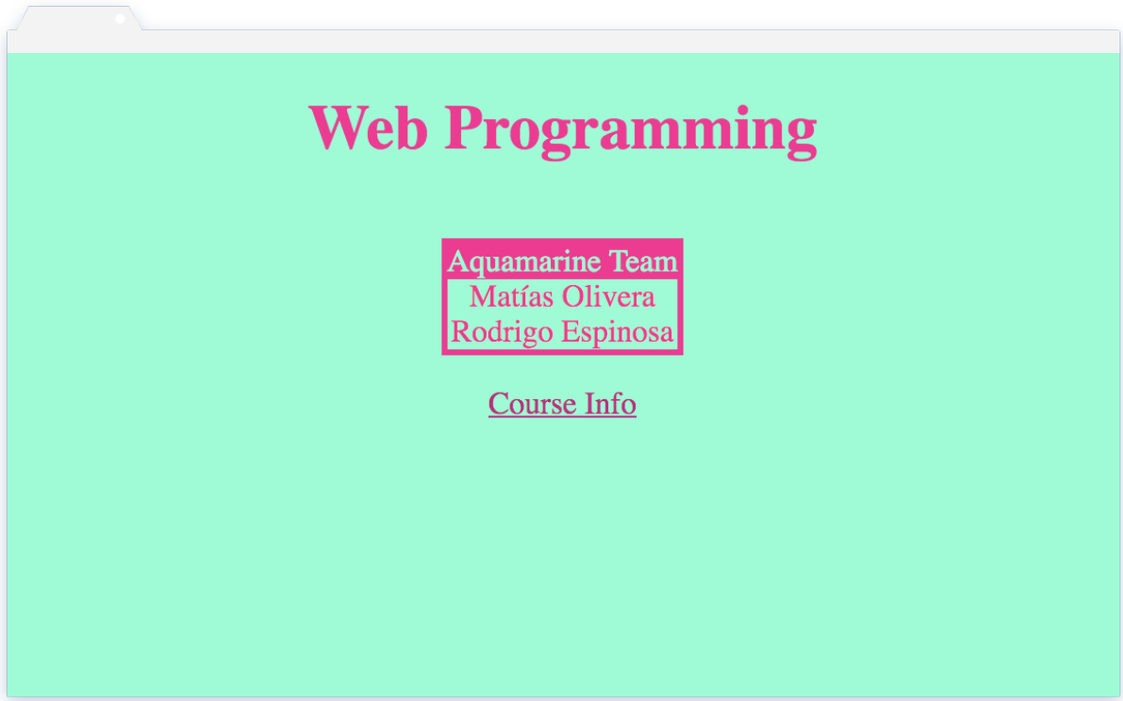


pseudo-classes

- allows selectors to act based on element's state:
  - **`a:`**`visited` matches all <**`a`**> elements that have been visited by the user
- possible: `:hover`, `:active`, and others

visited link

```
a:visited {
  color: mediumvioletred;
}
```

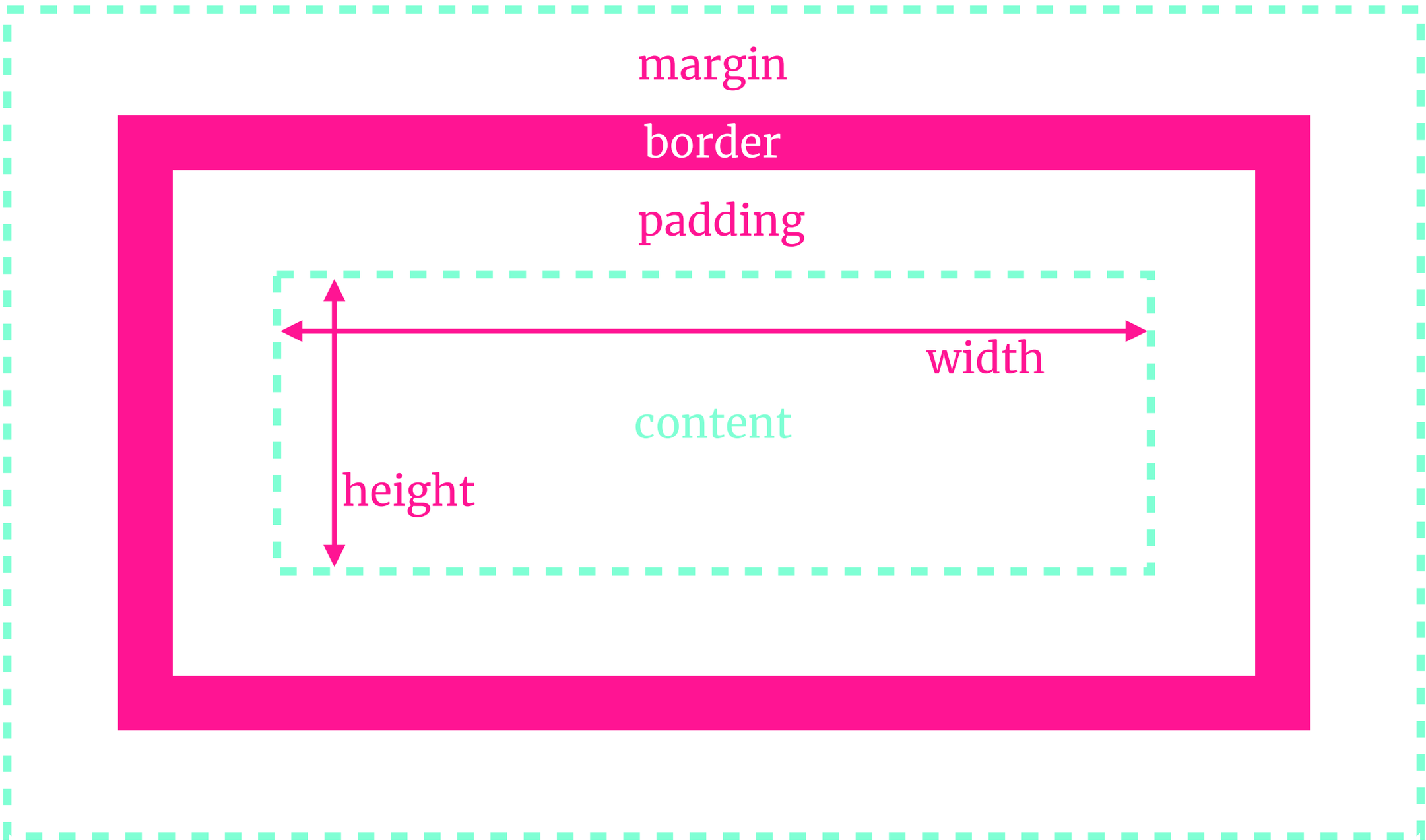


3. box model

CSS box model

- each element is represented as a box with four layers (like an onion):
  - content
  - padding
  - border
  - margin

box properties



`width` and `height`

- set the size of the **context box**
- includes text content and other nested boxes

- other properties: `min-width`, `max-width`, `min-height`, and `max-height`

`padding`

- space between the content and the border
- includes text content and other nested boxes
- properties: `padding` (shorthand), `padding-top`, `padding-right`, `padding-bottom`, and `padding-left`

`border`

- sits between the padding and the margin
- invisible by default (size of zero)
- properties:
  - `border` (shorthand), `border-top`, `border-right`, `border-bottom`, `border-left`
  - `border-width`, `border-style`, and `border-color`

`margin`

- space between the border and other elements

- properties: `margin` (shorthand), `margin-top`, `margin-right`, `margin-bottom`, and `margin-left`

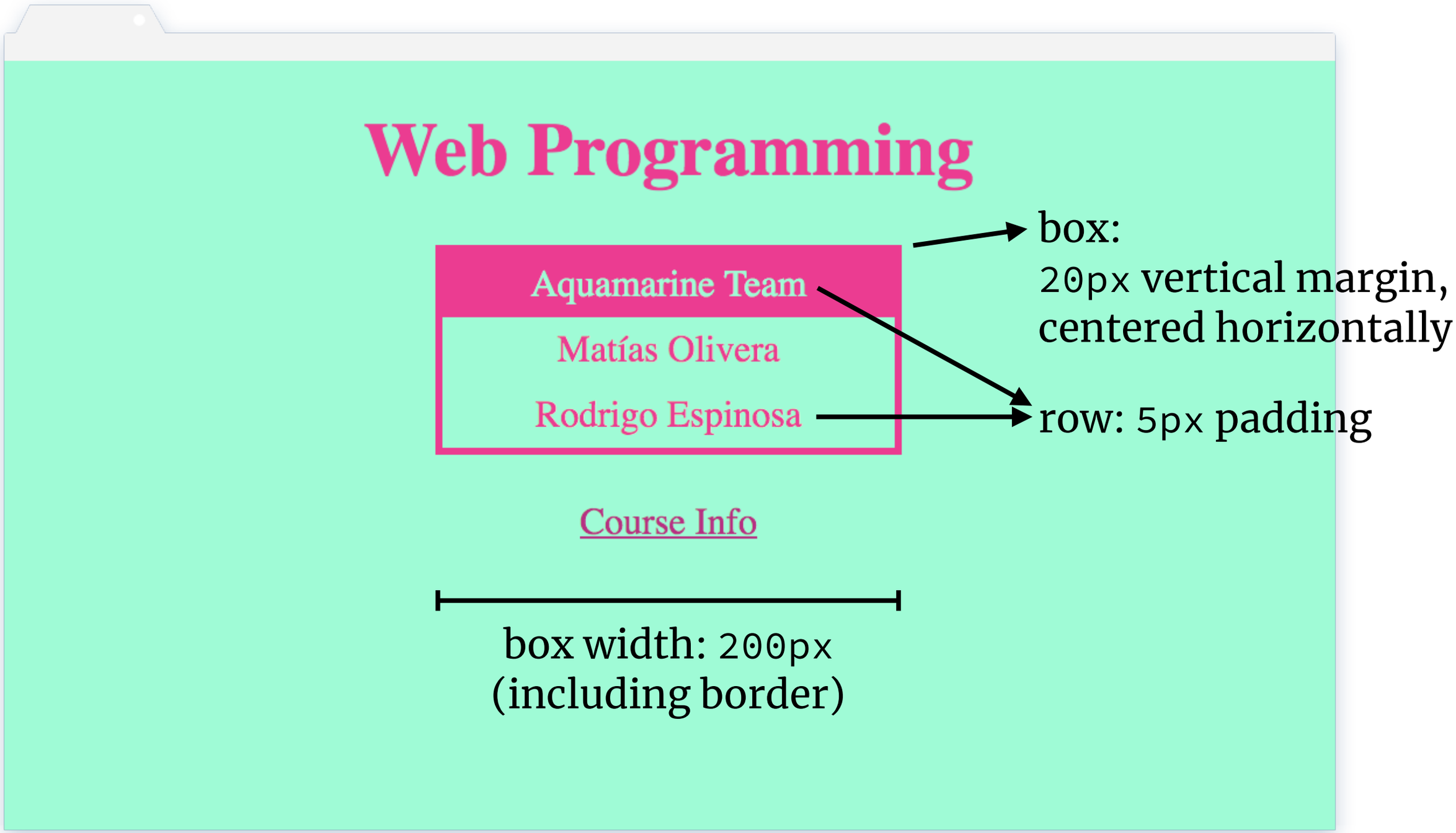
margin collapsing

- top and bottom margins of adjacent block elements are often combined (collapsed)
- size is the largest of the individual margins

auto margins

- set right and left margins to `auto` to center element
- only for block-level elements

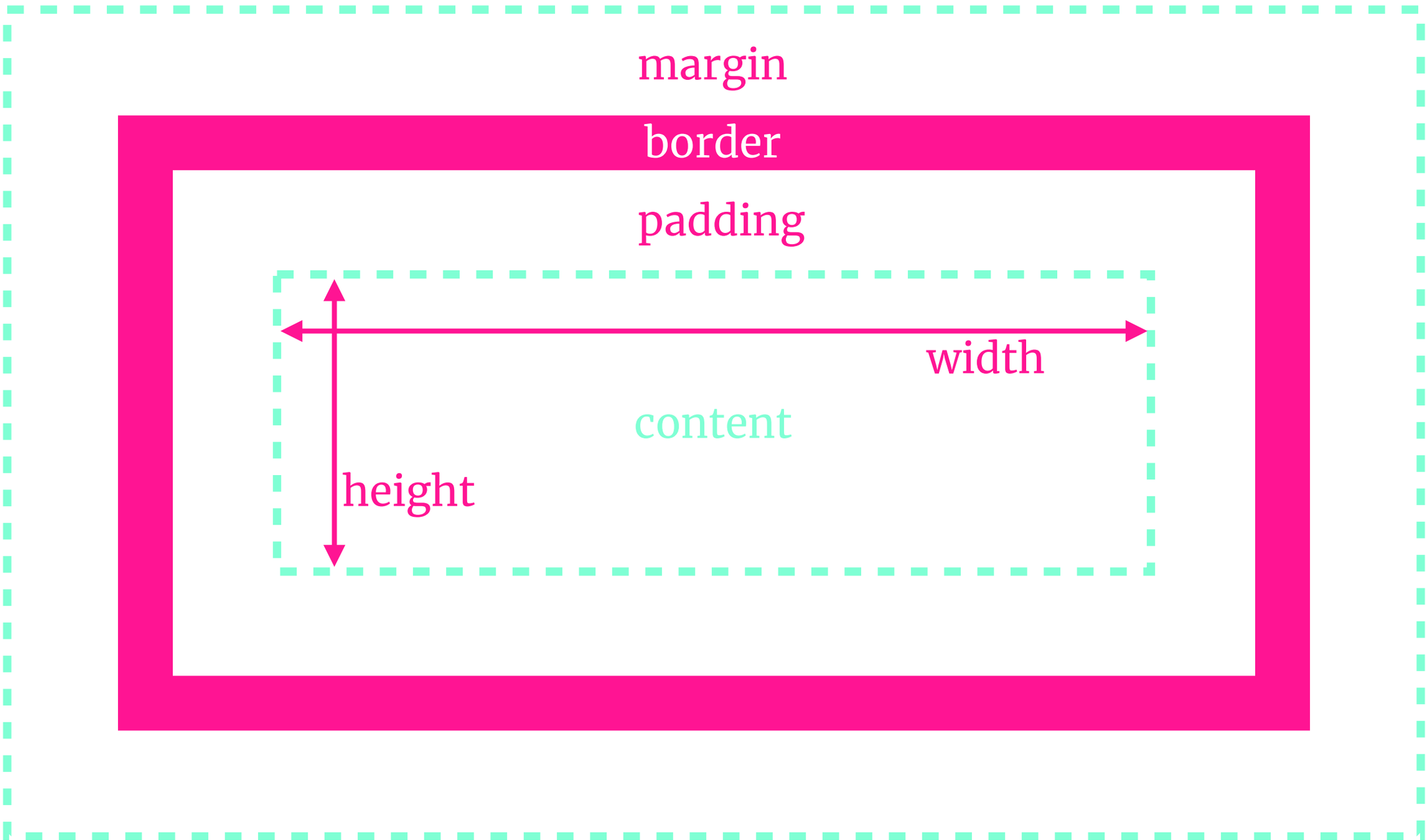
exercise


Web Programming
Aquamarine Team
Matías Olivera
Rodrigo Espinosa
Course Info
box:
20px vertical margin,
centered horizontally
row: 5px padding
box width: 200px
(including border)

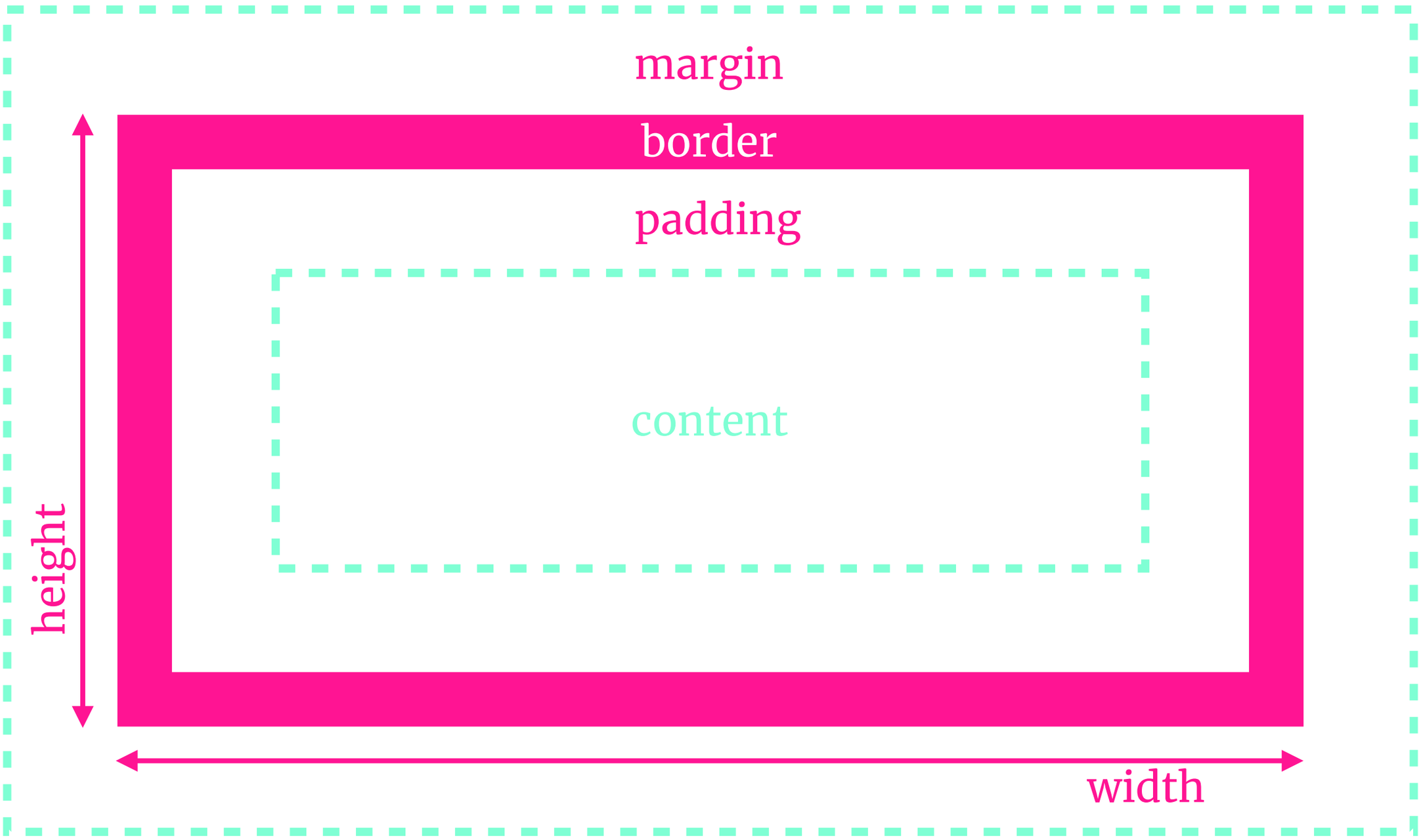
box sizing 😏

- allows to change how width and height of an element is calculated
- by default: `content-box`
  - include content
  - does not include padding, border, or margin
- another possible value: `border-box`

box sizing: content-box



box-sizing: border-box



solution

```html
<body>
    <h1>Web Programming</h1>
    <div class="box">
        <div class="row team-name">Aquamarine Team</div>
        <div class="row">Matías Olivera</div>
        <div class="row">Rodrigo Espinosa</div>
    </div>
    <a href="...">Course Info</a>
</body>
```


```css
body {
  background-color: aquamarine;
  color: deeppink;
  text-align: center;
}

a {
  color: deeppink;
}

a:visited {
  color: mediumvioletred;
}

.box {
  border: 3px solid deeppink;
  box-sizing: border-box;
  margin: 20px auto;
  width: 200px;
}

.row {
  padding: 5px;
}

.team-name {
  background-color: deeppink;
  color: aquamarine;
}
```

4. values & units

CSS values & units

- some property values rely on units for specifying the exact value they represent:
    - *do you want your box to be 30 pixels wide, or 30 centimeters?*

- different characteristics: **length**, position, color, etc.

length and size

- used **all the time**
- **absolute units** (always the same size):
    - px
    - `mm`, `cm`, `in`, `pt`, `pc` (you won't probably use any of these very often)
- **relative units** (relative to another length property):
    - percentage (*)
    - `vw`, `vh`
    - `em`, `rem`
    - `ex`, `ch`

(*): not exactly a unit but, *yaknow*!

“Displays are roughly all the same size so use static fonts.”

–CSS1 (1996)

percentage

- allows to create boxes whose size will always be a percentage of their container's size

- `width` and `height` are defined relative to their parent element

vw and vh

- allow to define size in terms of the viewport

- use `vw` and `vh` to set `width` and `height` to a **percentage of the viewport's width or height**, respectively

- `1vw` = 1/100th of the viewport's width

- `1vh` = 1/100th of the viewport's height

`em` and `rem`

- allow to define size in terms of value of `font-size`
  - `em`s are the most common relative unit you will use
  - `1em` = current element's `font-size`
  - `1rem` = default base `font-size` (*)

(*): usually 16px

ex and ch

- allow to define size in terms of specific characters of the current font
- not as commonly used or well-supported as ems
- 1ex = lower case x's height
- 1ch = number 0's width

in summary

- size your HTML elements relative to the font or viewport size
- layout stays correct and consistent
- **huge value!**

unitless values

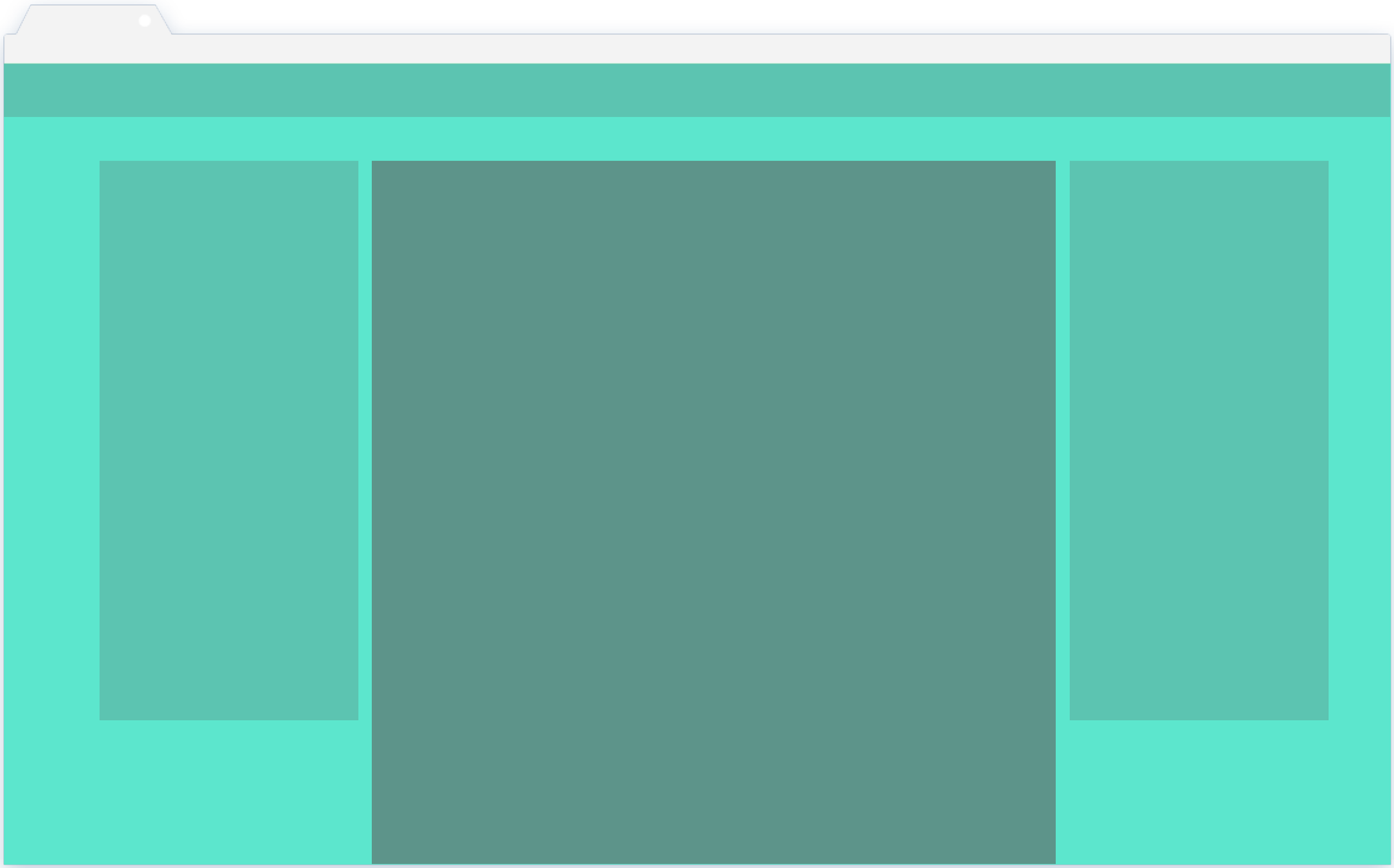
- sometimes it's perfectly fine to use unitless values:
  - unitless zero, e.g. `margin: 0;`
  - unitless `line-height`: acts as a multiplying factor, e.g. `line-height: 1.5;`

5. layout

sectioning elements

- describe the **structure** and **outline** of a web page in a standard way
- help people and machines understand the page
- e.g.: `article`, `aside`, `footer`, `header`, `nav`, `section`
- **prefer these elements to `<div>`s when it makes sense!**

<header> <section> <aside>

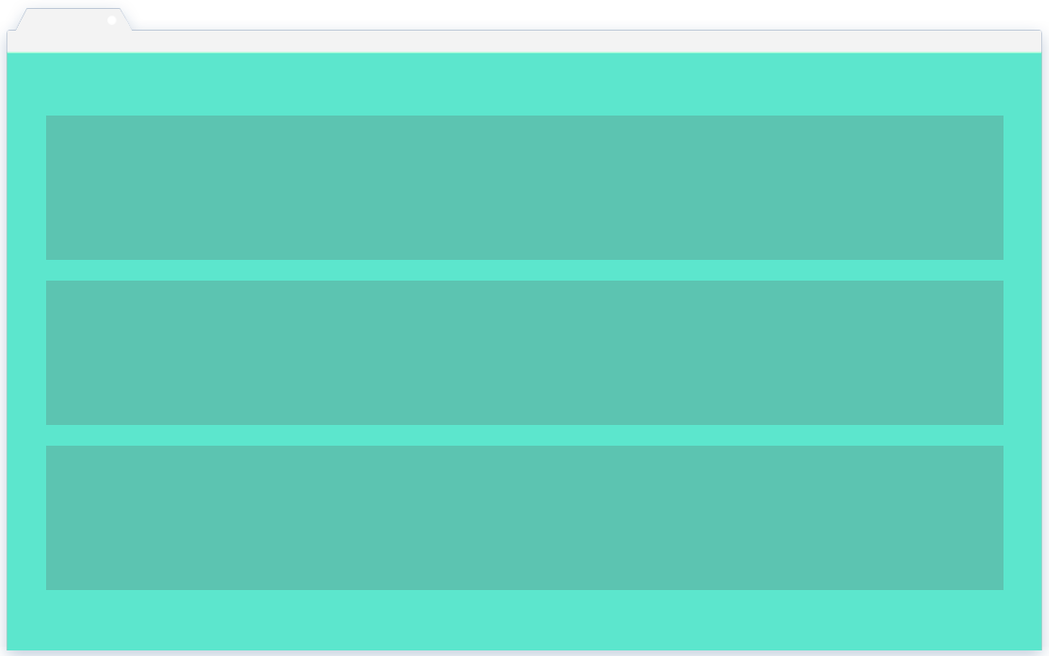


<header> <section> <aside>

<header>
<section>
<aside>

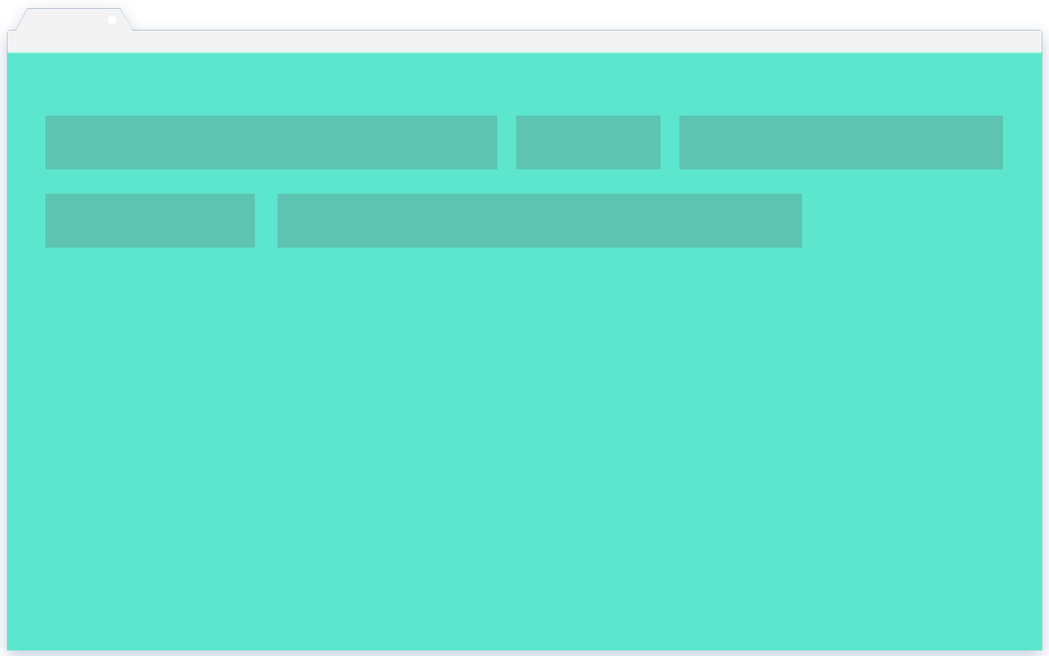
exercise

layout so far...



block layout

laying out large
sections of the page



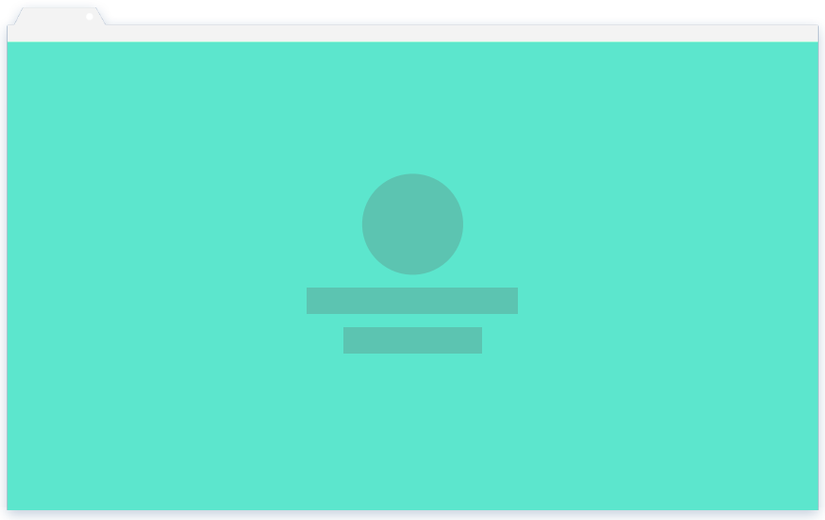
inline layout

laying out text
and other content
within a section

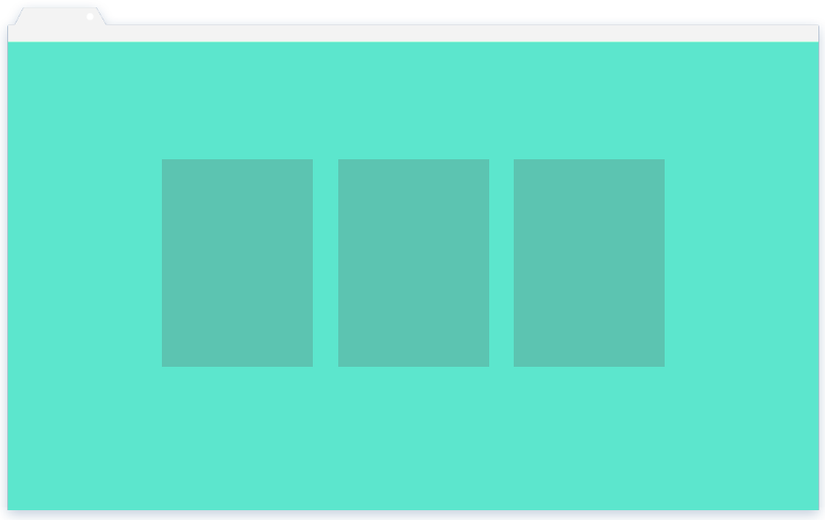
flexbox

- layout model
- more efficient way to **lay out**, **align** and **distribute space** among items in a container
- one-dimensional

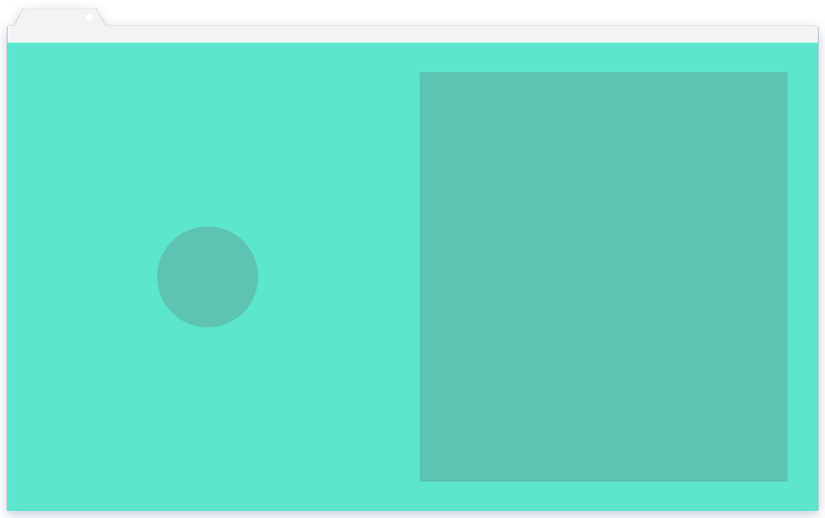
more layout (*)



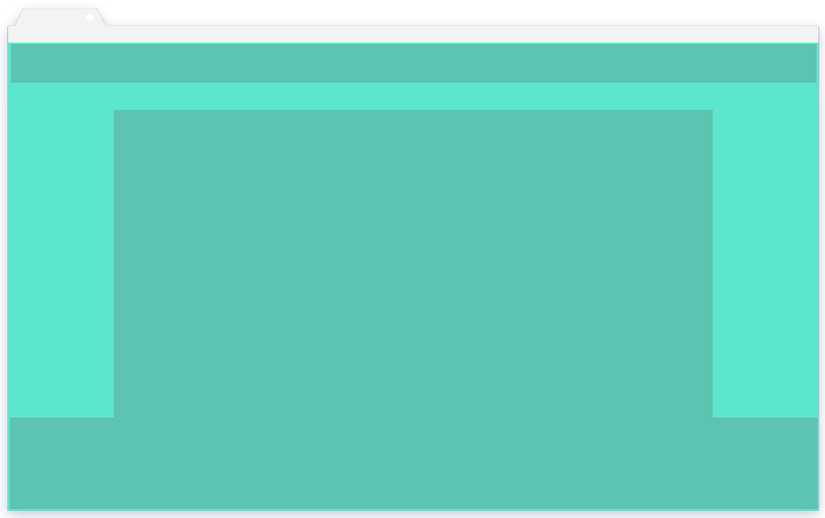
centering



cards



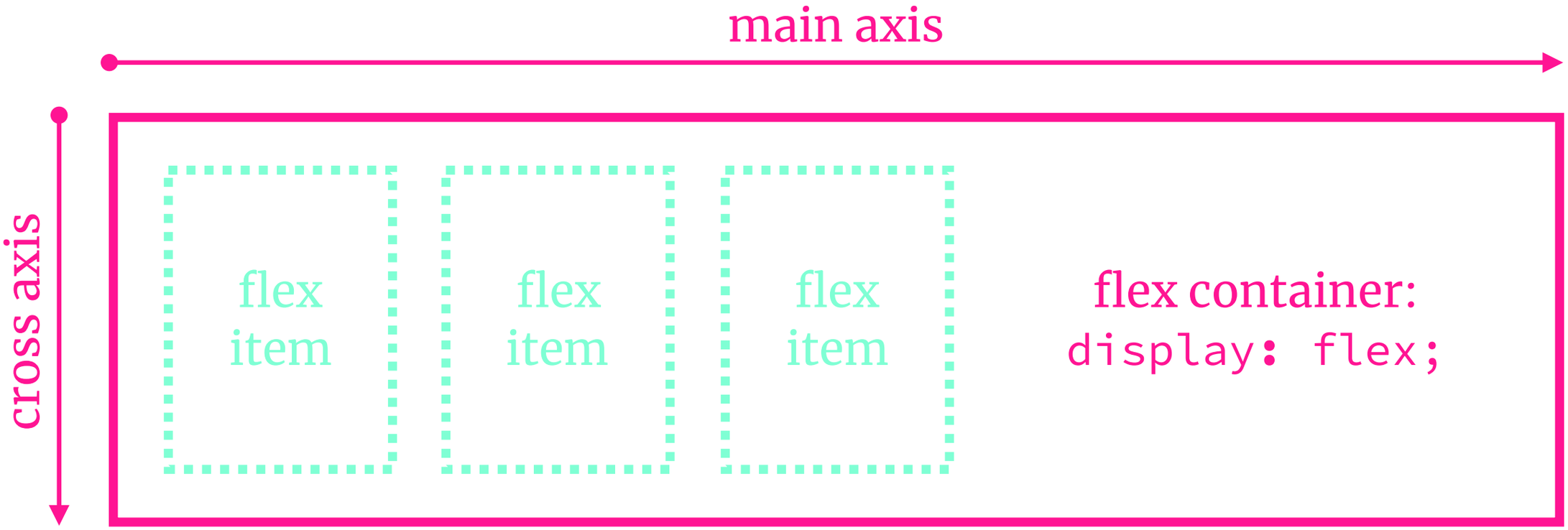
split screen



sticky footer

(*): easy with flexbox, otherwise complicated!

flexbox properties



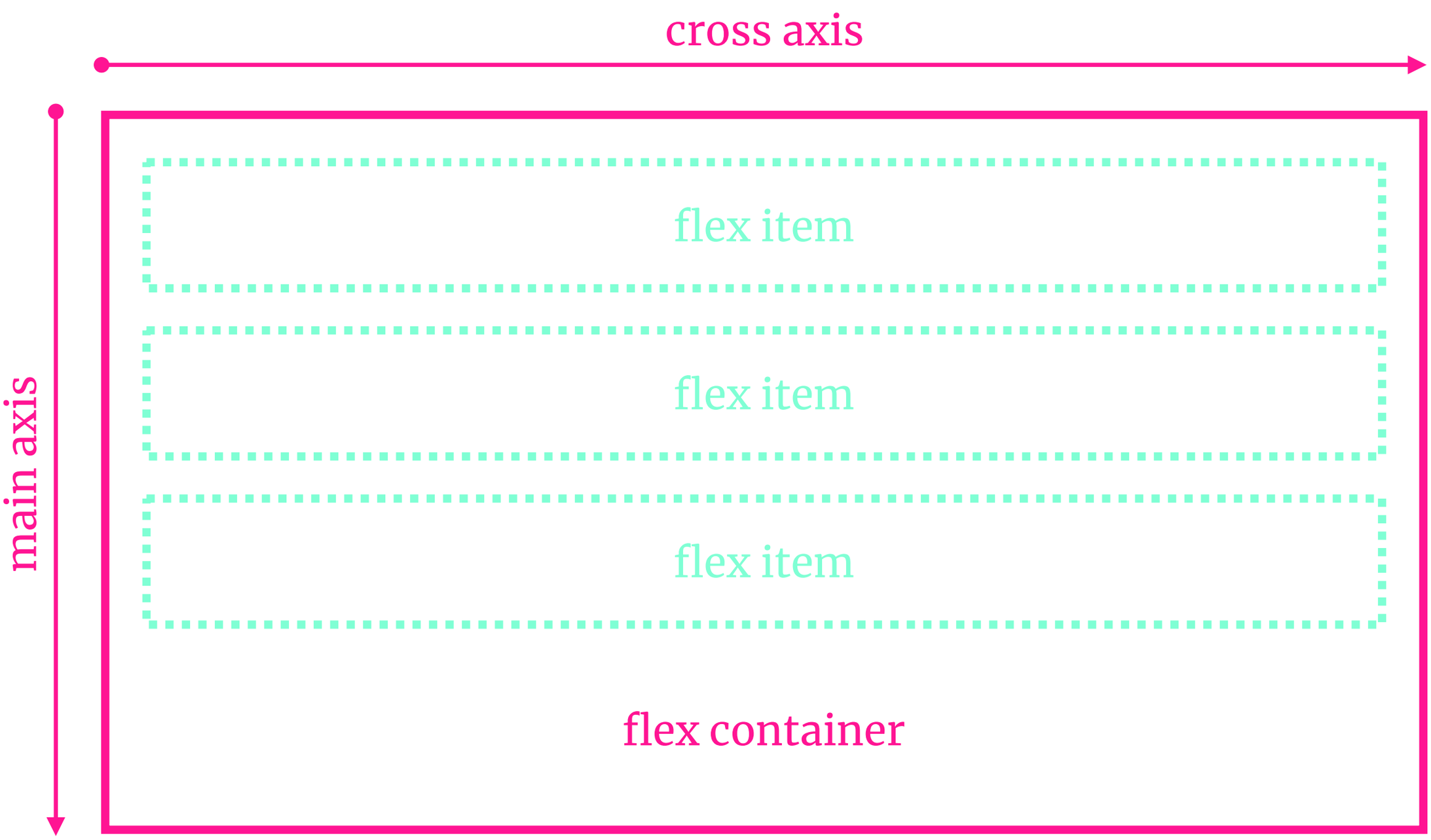
flex container

- `display: flex;` OR `display: inline-flex;`
- by default:
  - items display in a row
  - items **start from start edge** of **main axis**
  - items **stretch to fill** the size of **cross dimension**
  - items **do not stretch** on the **main dimension**, but **can shrink**

`flex-direction`

- specifies **how flex items are placed** in the flex container
- defines main axis and direction (normal or reversed)
- by default: `row`
- other values: `row-reverse`, `column`, `column-reverse`

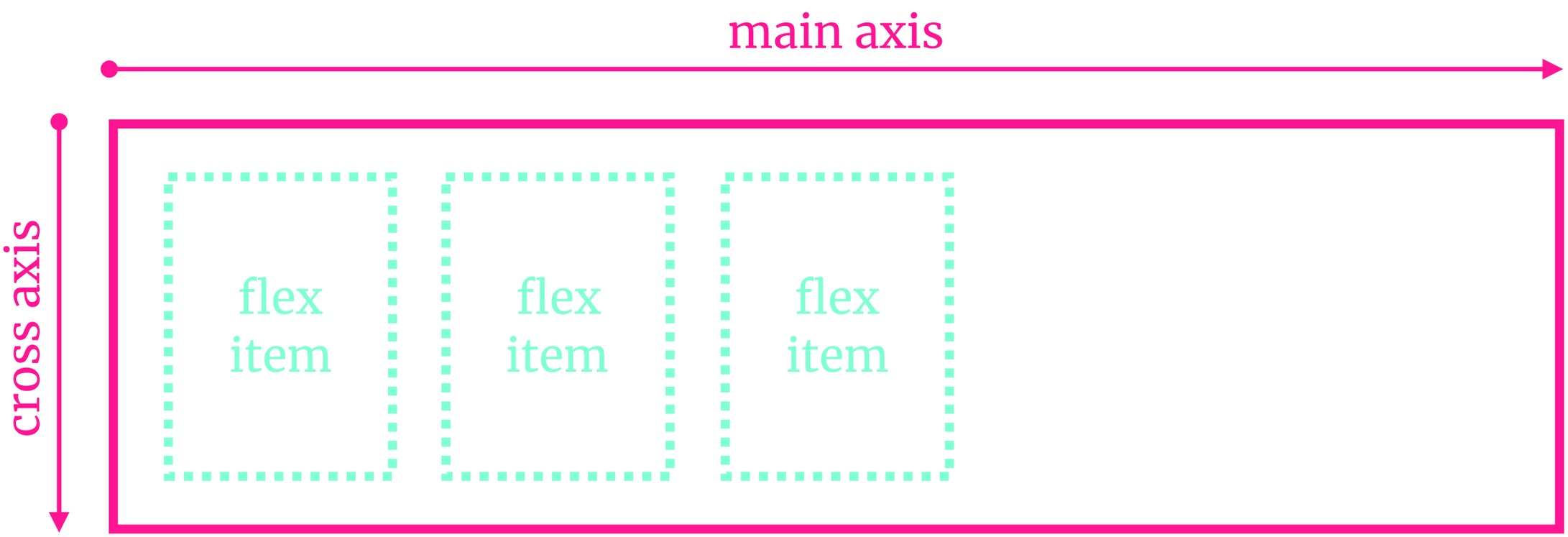
flex-direction: column



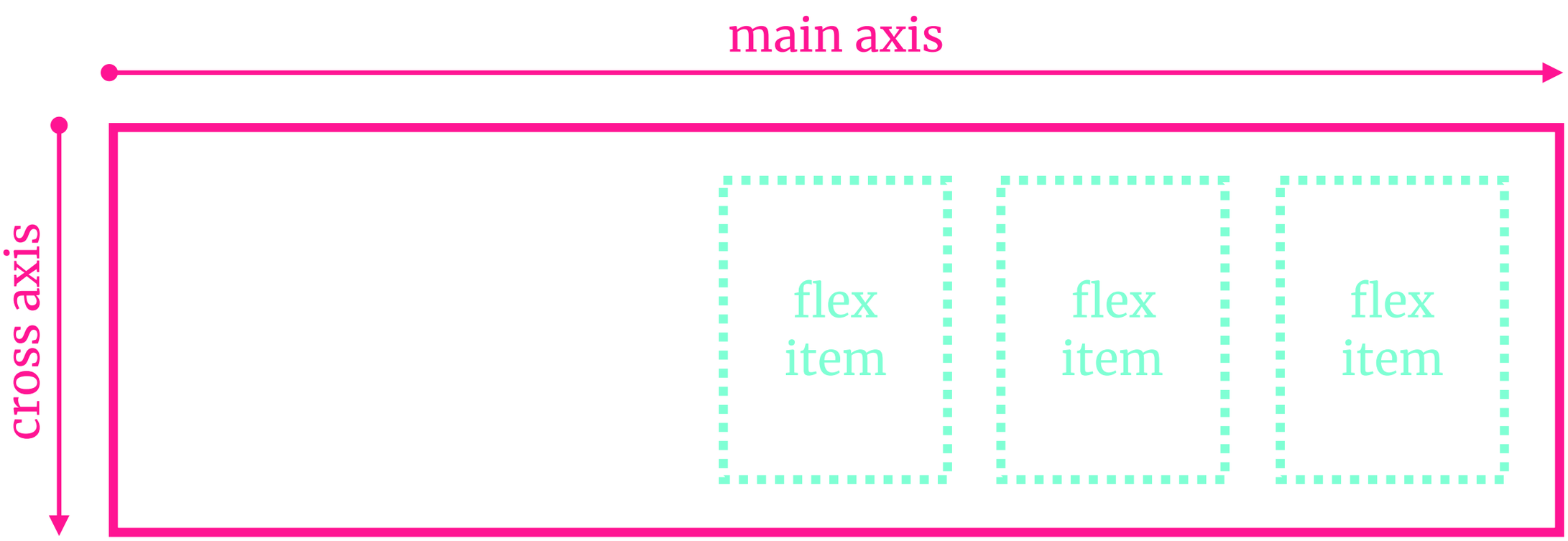
`justify-content`

- specifies **where flex items are placed** with reference to the main axis
- by default: `flex-start`
- other values: `flex-end`, `center`, `space-between`, `space-around`, and others

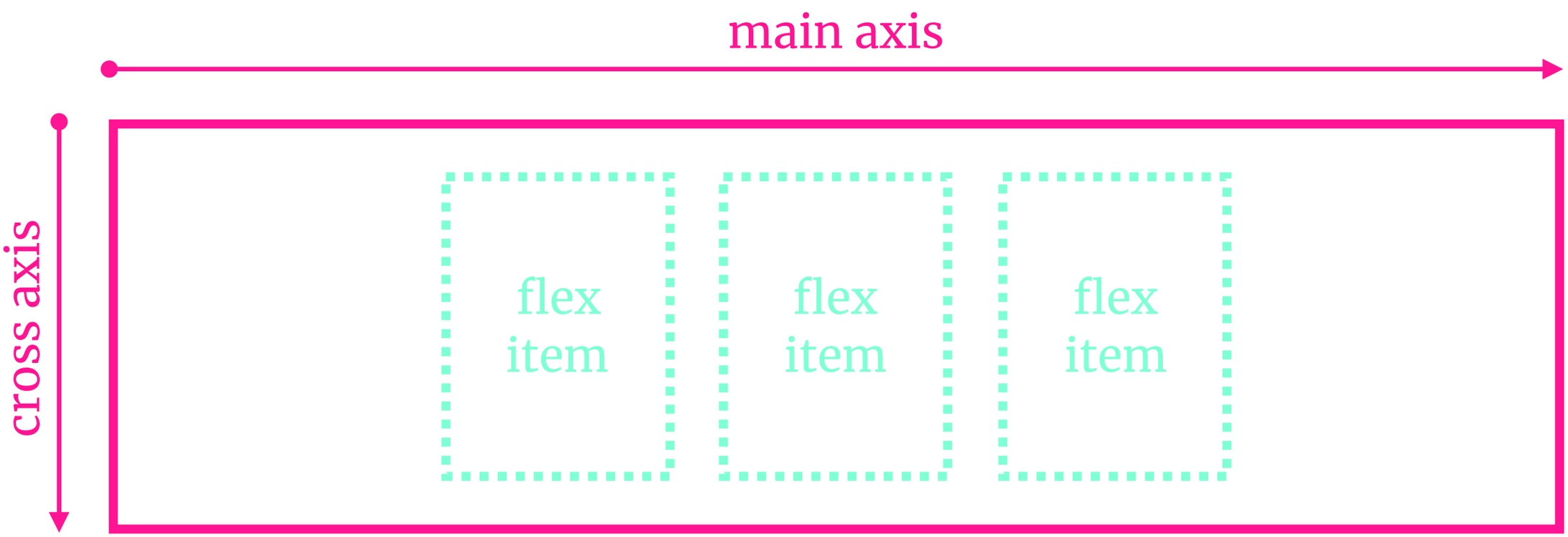
```
justify-content: flex-start;
```



```
justify-content: flex-end;
```

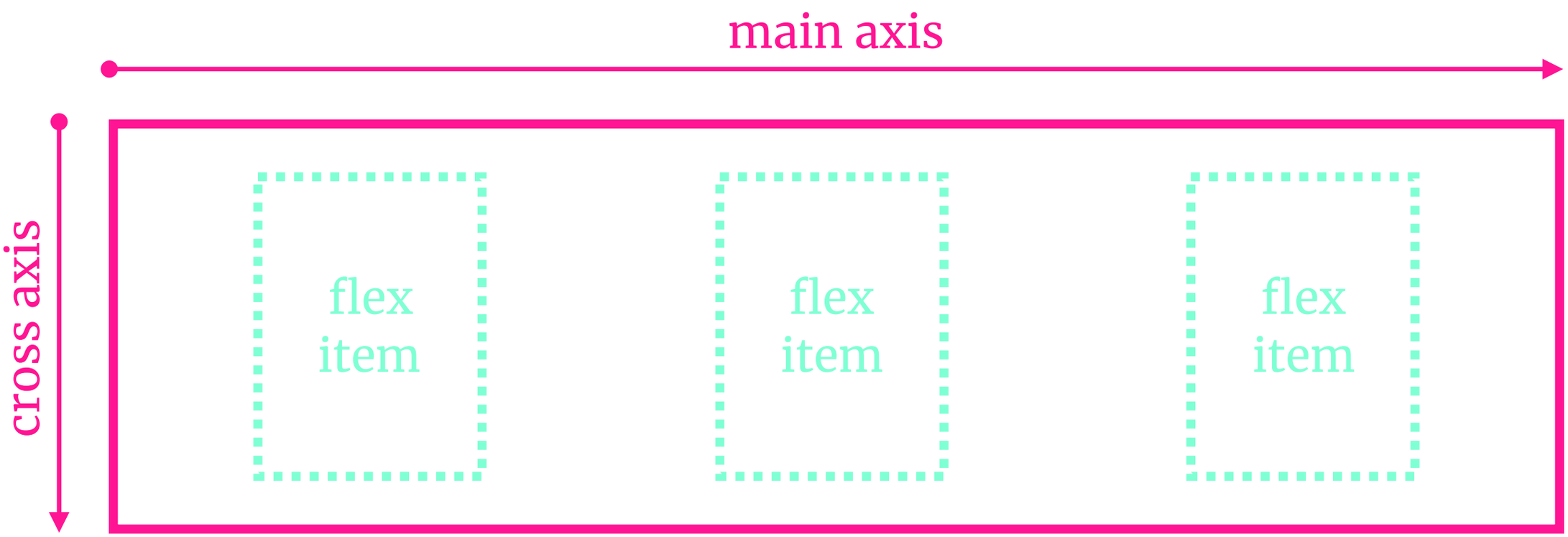


```
justify-content: center;
```



```
justify-content: space-between;
```

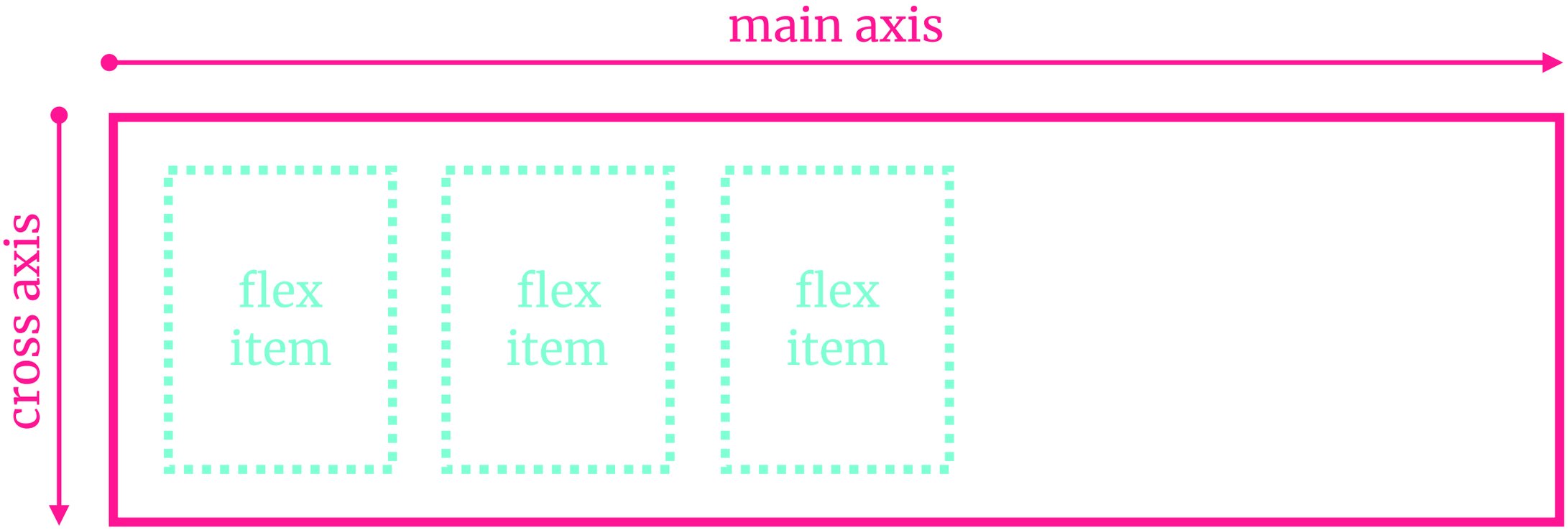
```
justify-content: space-around;
```



`align-items`

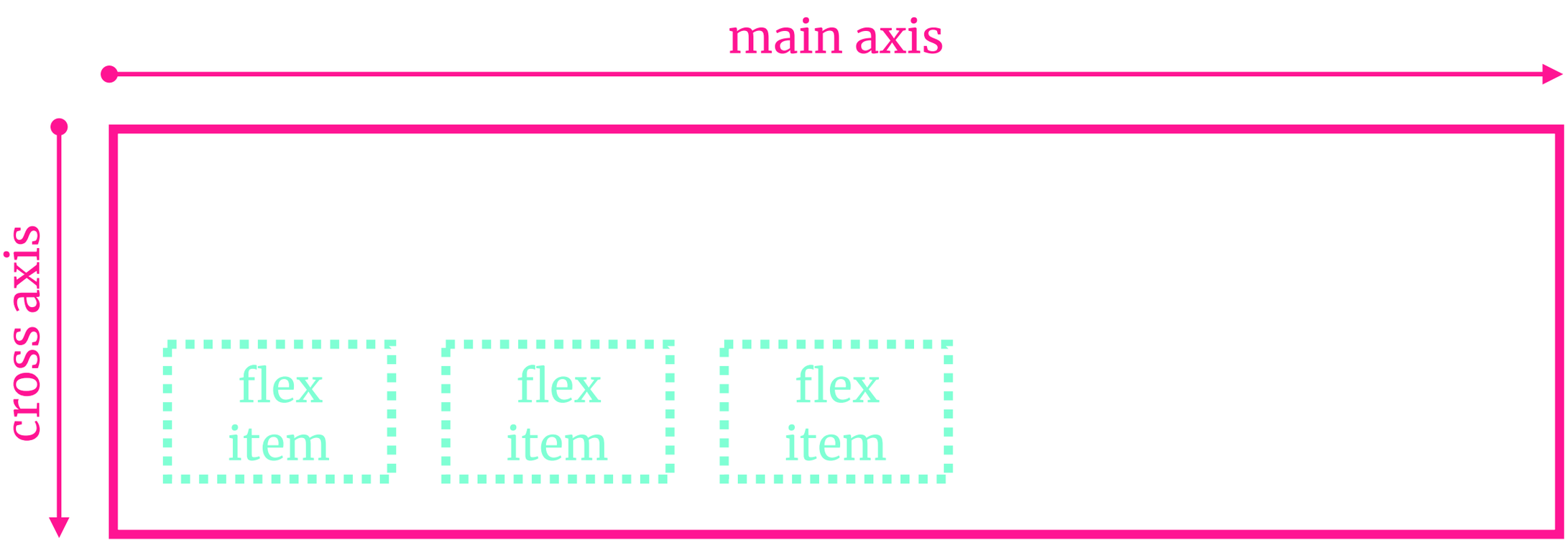
- specifies **where flex items are placed** with reference to the cross axis
- by default: `stretch`
- other values: flex-start, `flex-end`, `center`, and others

align-items: stretch;

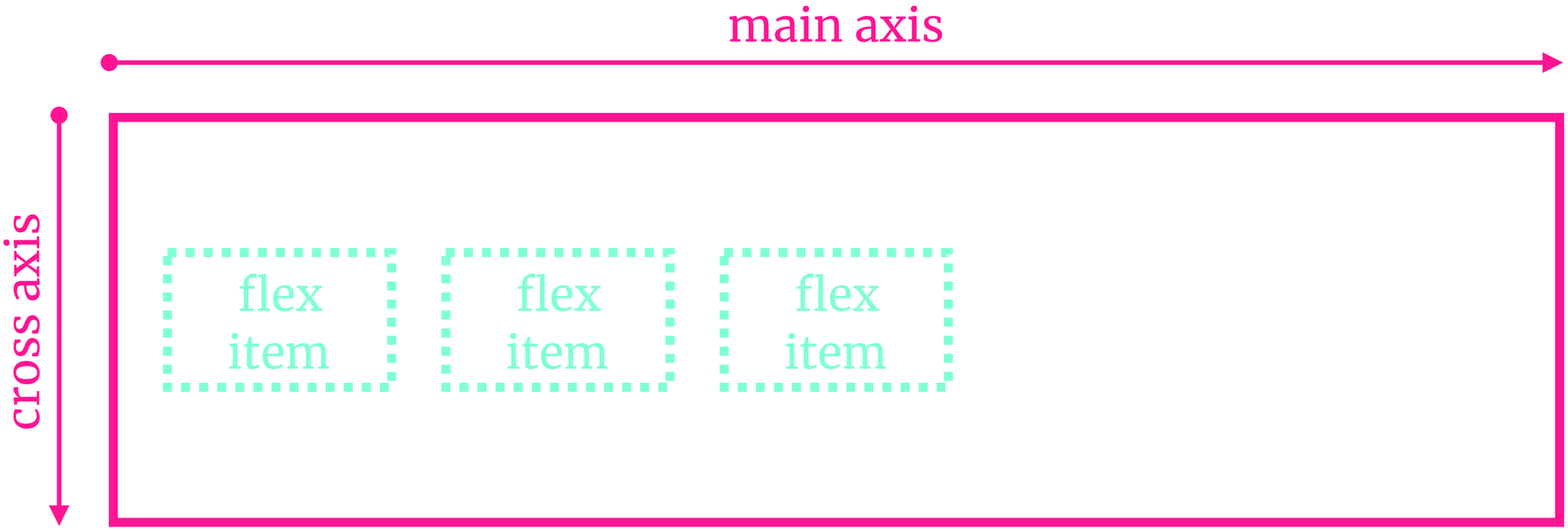


```
align-items: flex-start;
```

```
align-items: flex-end;
```



```
align-items: center;
```



flex item

- direct children of a container with `display: flex;`
- follow **new set of rules** (not block nor inline)
- item's main size (with relation to main axis):
  - content size, or
  - explicitly set `width` or `height` property, or
  - explicitly set `flex-basis` property

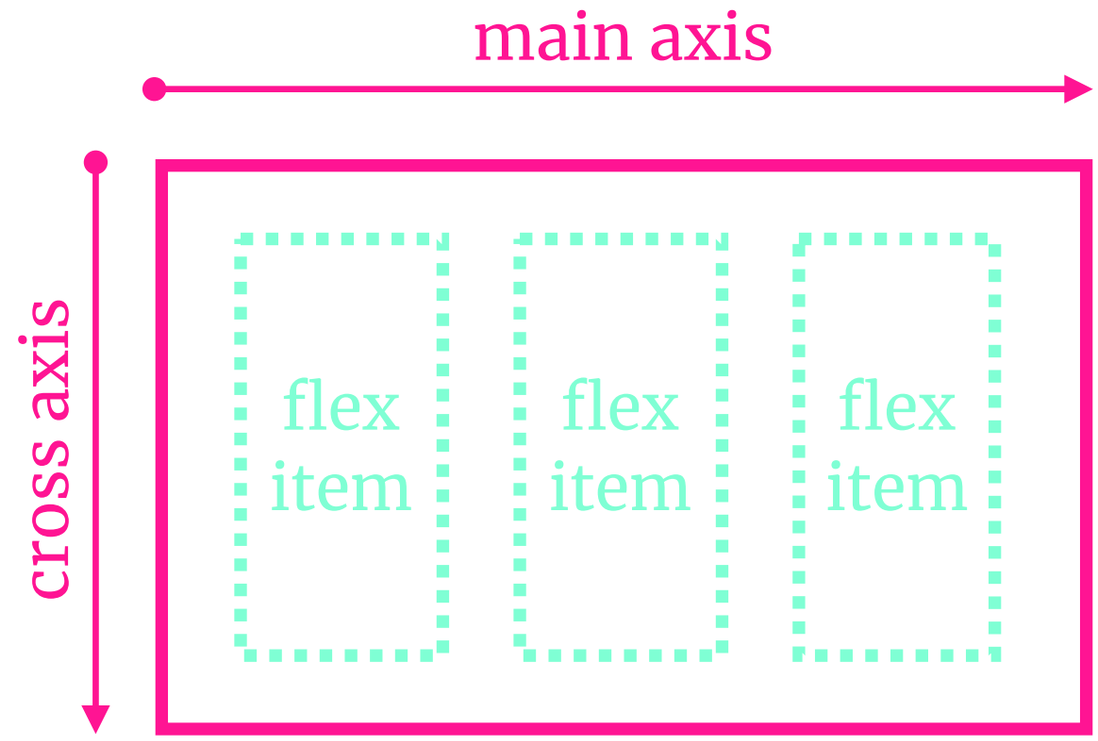
flex-basis

- specifies initial size of the item, before any additional space is distributed according to the flex factors:
  - initial **width** of the item in case of **row** layout
  - initial **height** of the item in case of **column** layout

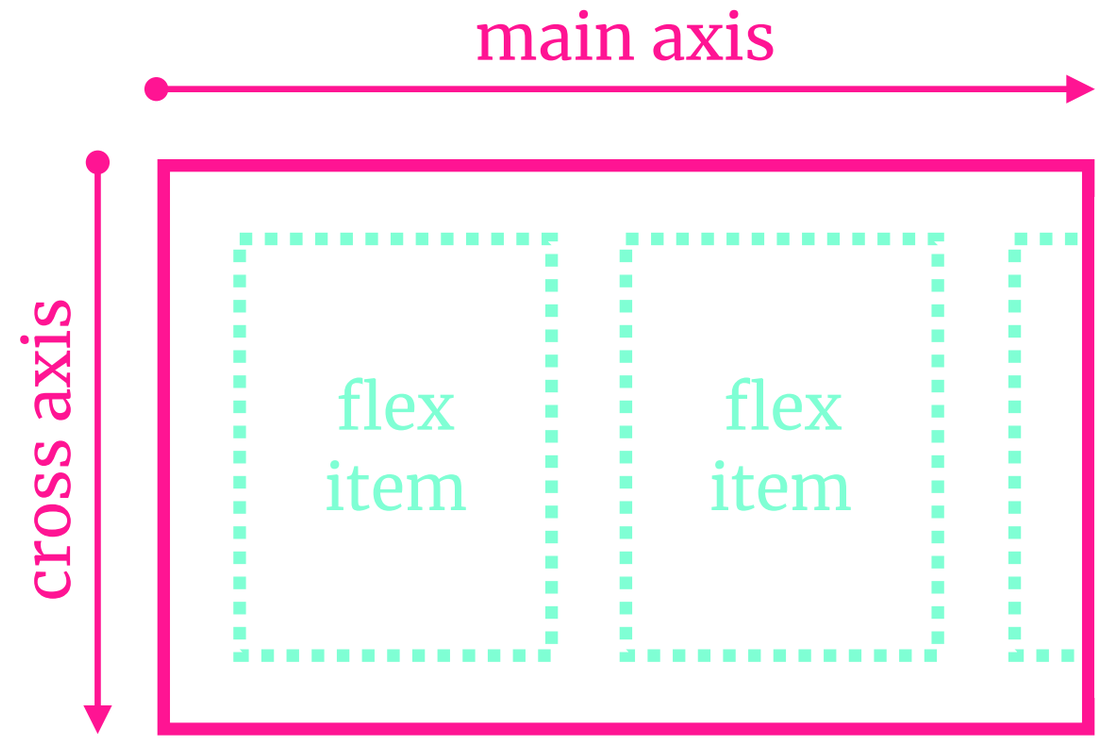
`flex-shrink`

- reduces main size of the item so it fits in the available space
- factor, from `0` to `1`
- by default: `0` (do not shrink)
- value of `1`: shrinks as small as it can

```
flex-shrink: 1;
```



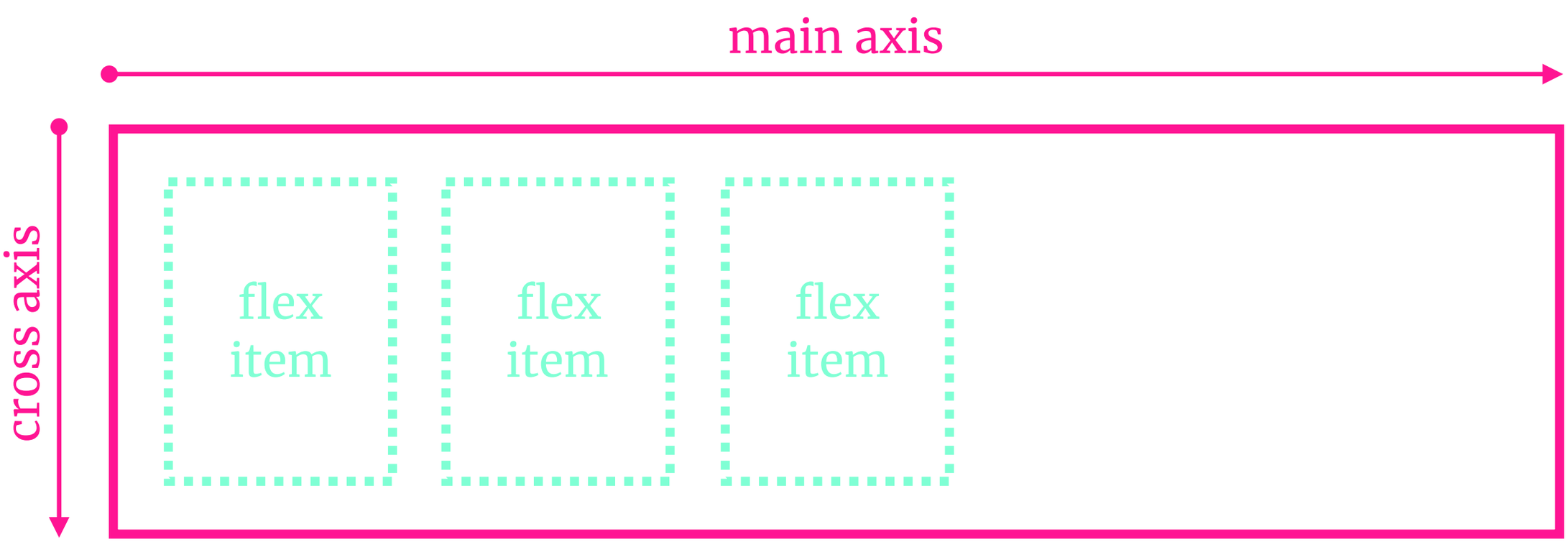
flex-shrink: 0;



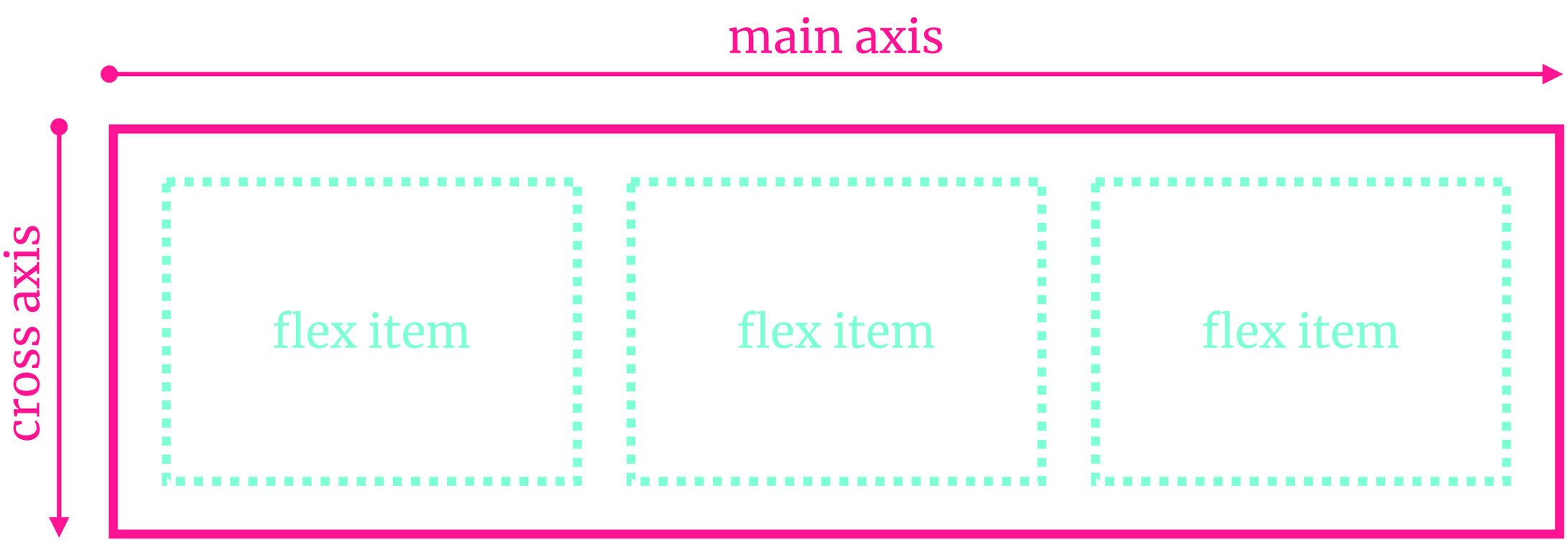
`flex-grow`

- increases main size of the item so it fills the available space
- proportion
- by default: `0` (do not grow)
- value of `1`: grows as large as it can

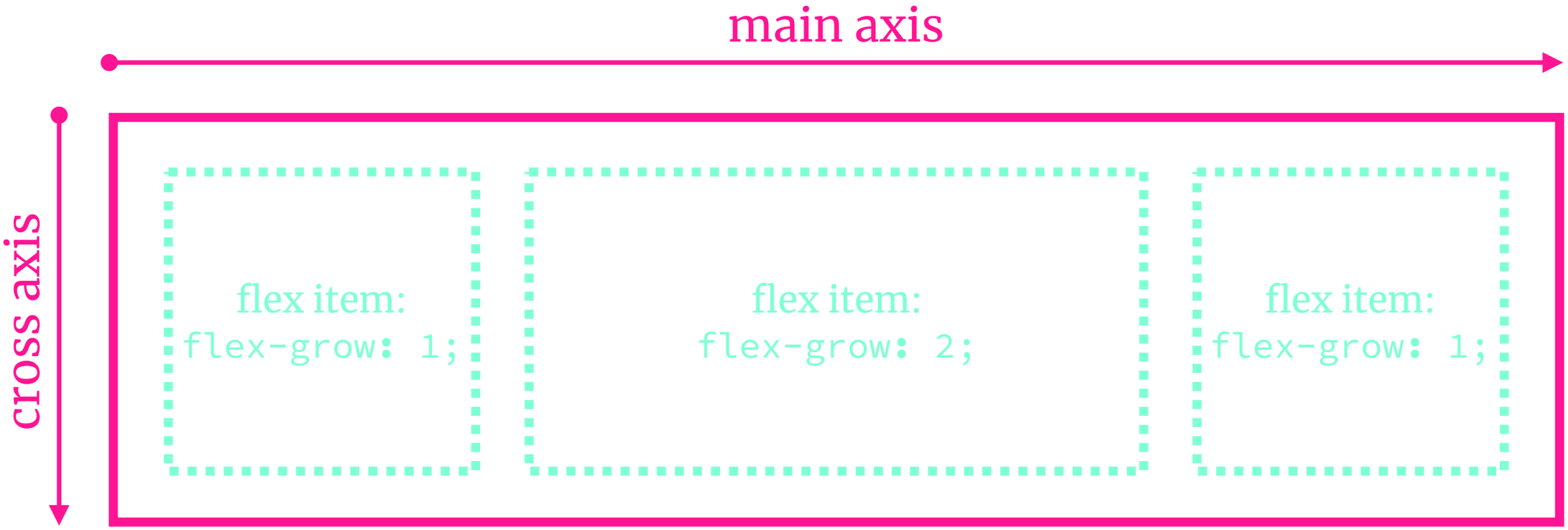
flex-grow: 0;



flex-grow: 1;



other distributions



`flex`

- shorthand property for `flex-grow`, `flex-shrink` and `flex-basis`, in that order (*)
- **recommended**, as it sets the other values intelligently
- by default: `0 1 auto`

(*): `flex-shrink` and `flex-basis` are optional

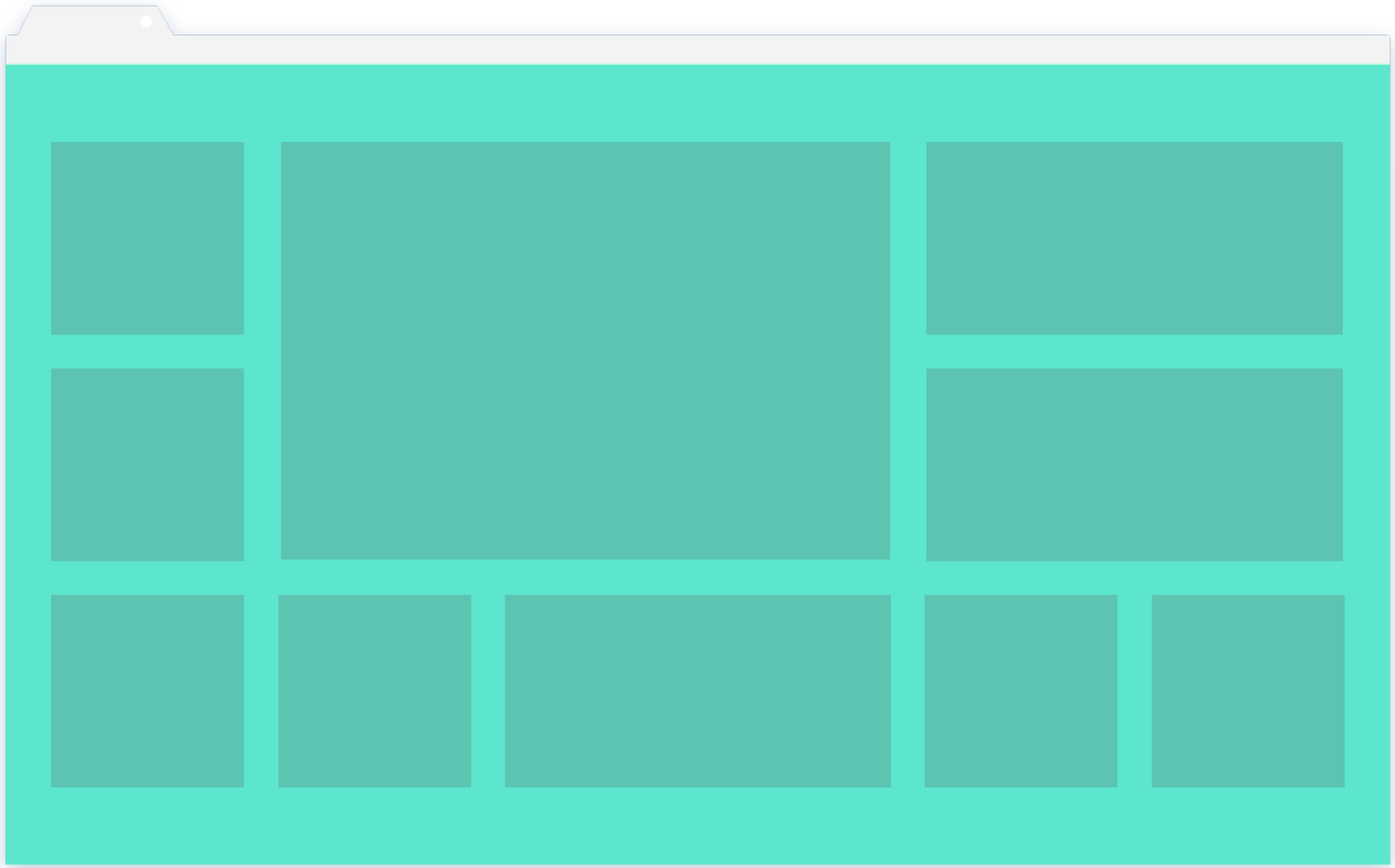
flexbox guide

goo.gl/99ws45

grid

- another layout model
- **most powerful layout system** available to date
- two-dimensional (in contrast to flexbox)

more layout (*)



(*): easy with grid, otherwise complicated!

grid guide

goo.gl/EyuLRc

solution

```
<body>
  <header></header>

  <div class="content">
    <aside class="trends"></aside>

    <section class="tweets"></section>

    <aside class="suggestions"></aside>
  </div>
</body>
```

```
.content {
  display: flex;
  justify-content: center;
}

.trends,
.suggestions {
  flex: 1;
}

.tweets {
  flex: 2;
  margin: 0 1em;
}
```

6. responsive design

responsive

- change the layout & content based on the **screen size**
- change graphics & images based on the **pixel density** of the device
- using **media queries**

media queries

- conditionally apply styles to your **CSS**
- can detect a vary of **features**, such us:
  - width, height, aspect-ratio, resolution, and others
  - device type: `screen`, `print`, `speech`
- accepts **logical operators** (`and`, `not`, `only`)

targeting features

```
@media (max-width: 1245px) {
   /* styles */
}
```

```
@media only screen
and (min-device-width : 768px)
and (max-device-width : 1024px)  {
   /* styles */
}
```

exercise

make your Twitter-like layout responsive!

solution

```
@media (max-width: 1245px) {
  .content {
    flex-direction: column;
  }

  .tweets {
    margin: 0;
    order: 1;
  }

  .trends: {
    order: 2;
  }

  .suggestions {
    order: 3;
  }
}
```

thanks!