The grid system

Sections in this chapter:

4-1. The grid system

Responsive and mobile-first

Bootstrap 3 comes with a **mobile first responsive grid system**, providing us an easy way to approach responsive web design.

Bootstrap grid system lets us take distance from plenty of **handwritten media queries**, with its **ready-made classes** for us to apply instead. *4-1-2*

The grid system is based on a **12 column** layout. *4-1-3*

It's basically a set of elements with some of the **ready-made** classes applied.

We create responsive, nicely lined up system, in form of rows and column where we house our content.

The code for a simple grid layout could look like this: *4-1-4*

```
<div class="container">
  <div class="row">
    <div class="col-xs-6">hello</div>
    <div class="col-xs-6">world</div>
  </div>
  <div class="row">
    <div class="col-xs-3">this</div>
    <div class="col-xs-9">is pretty cool!</div>
  </div>
</div>
```

There's some underlying style applied on the example that is not visible in the code snippet.

Bootstrap classes could also be applied on semantic elements for example: *4-1-5*

```
<main class="container">
    <section class="row">
        <article class="col-xs-6">
            <header>Content in article with col-xs-6</header>
        </article>
        ...
    </section>
</main>
```

Picking it apart, we see that we have: *4-1-6*

- **container**: element with the `container` (or `container-fluid`) class wrapping the actual grid.
- **rows**: element with the `row` class, holding a set of columns.
- **columns**: elements defined with one or more of the media query range classes.

4-2. Container

Our grid will be placed inside a **container**, which is an element with `.container` or `.container-fluid`. *4-2-1*

```
<div class="container">
    ... and here the grid will go!
</div>
```

The containers have the **means to center your sites content**, which comes with this CSS: *4-2-2*

```
.container, .container-fluid {
  padding-right: 15px;
  padding-left: 15px;
  margin-right: auto;
  margin-left: auto;
}
```

`container vs container-fluid` *4-2-3*

This is a div with the **container** class.

This is a div with the **container-fluid** class.

`.container`

4-2-4

Sets a fixed width for each screen size (xs, sm, md, lg).

The width comes from these lines of CSS: *4-2-5*

```
@media (min-width: 568px) {
  .container {
    width: 550px;
  }
}
@media (min-width: 992px) {
  .container {
    width: 970px;
  }
}
@media (min-width: 1200px) {
  .container {
    width: 1170px;
  }
}
```

`.container-fluid`

4-2-6

Takes the full available width of each screen size.

When we have our page within a `.container`, content might seem to "jump" when resizing the window. *4-2-7*

This is because the container is set to a specific width for each device size. *4-2-8*

Reality here is that a general user opens a website on a device and that screen is what it is, the user would not see this transition effect. *4-2-9*

4-3. Rows

Rows are **horizontal groups of columns**. *4-3-1*

The row ensure that your columns are **lined up properly**.

The more rows we have the longer the page gets from top to bottom. *4-3-2*

```
<div class="container">
    <div class="row">
        ...
    </div>
</div>
```

It is up to you to set the height of these rows, or the containing columns though.

The following shows a container with one row in it: *4-3-3*

This is a div with the container class

This is a div within container with the row class

The **row** takes up the **full width of the container**. *4-3-4*

Q But didn't the containers have padding applied? *4-3-5*

A Correct! We could see the following applied on `.container` and `.container-fluid`: *4-3-6*

```
.container, .container-fluid {
  padding-right: 15px;
  padding-left: 15px;
  margin-right: auto;
  margin-left: auto;
}
```

But the `.row` in turn **subtract these paddings by adding negative margins**: *4-3-7*

```
.row {
  margin-right: -15px;
  margin-left: -15px;
}
```

This makes it hard for the eye to see that the row actually is placed inside the container. *4-3-8*

We can demonstrate to see that the row is located inside of the container, by resetting the negative margins: *4-3-9*

This is a div with the container class

This is a div with the row class

4-4. Columns

Within each row we have a set of columns, and only columns should be **immediate children** of a row. *4-4-1*

```
<div class="container">
    <div class="row">
        <div class="col-sm-3">
            This is a column
        </div>
        <div class="col-sm-9">
            This is another column
        </div>
    <div>
</div>
```

Column classes indicate the **number of columns** you would like a column to use out of the **possible 12 per row**. *4-4-2*

We have **12 possible per row**:

4-4-3

1/12	1/12	1/12	1/12	1/12	1/12	1/12	1/12	1/12	1/12	1/12	1/12

We can divide them as we wish:

1/12	1/12	2/12	3/12	5/12

If you want **three equal-width columns** in a row, you can use `.col-xs-4` for example:

4-4-4

4/12	4/12	4/12

Under the hood

4-4-5

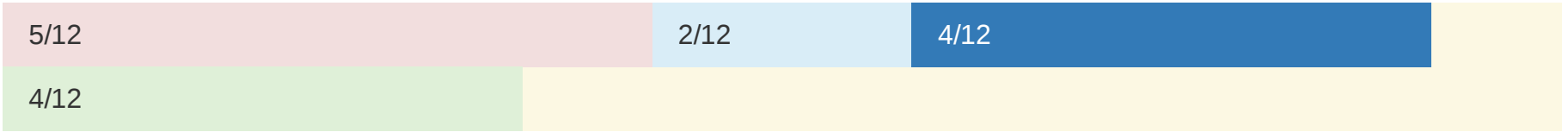
The grid system columns is implemented using the **CSS float-property** within different **media queries**.

The *behind the scenes* CSS could look something like: 4-4-6

```
@media (min-width: 992px) {
    .col-sm-* {
        float: left;
    }
    .col-sm-6 {
        width: 50%;
    }
    .col-sm-3 {
        width: 25%;
    }
}
```

If you define **more than 12 columns** in a row, thus **exceeding 100 percent**, Bootstrap will stack them. 4-4-7

This means that the overflowing columns will **wrap onto a new line**, but keep the given width. 4-4-8



5 + 2 + 4 + 4 = 15, which is why the last column will end up on a new line.

Q So, what does sm mean? 4-4-9

A Well, turns out we have different **break points** that target different devices and screen sizes. 4-4-10

There is one tier for each media query range like: 4-4-11

- **Extra small** devices ~ phones >= 480px
- **Small** devices ~ tablets >= 768px
- **Medium** devices ~ square desktops >= 992px
- **Large** devices ~ widescreen desktops >= 1200px

Every one of these four tiers have a particular class to address it: 4-4-12

- `.col-xs-*` ~ Extra small devices
- `.col-sm-*` ~ Small devices
- `.col-md-*` ~ Medium devices
- `.col-lg-*` ~ Large devices

The asterisks will be replaced by a .digit <= 12.

Mobile first 4-4-13

Bootstrap approaches **mobile first**, meaning default it starts on **extra small** devices to scale up.

Rules for the smallest screen will be used first.

The **extra small break point** rule will apply for **all devices** if nothing else if given. 4-4-14

All devices include extra small screens, small screens, medium screens and large screens.

The following example will result with two columns with a 50 percent width each on all devices. *4-4-15*

```
<div class="col-xs-6">
    ...
</div>
<div class="col-xs-6">
    ...
</div>
```

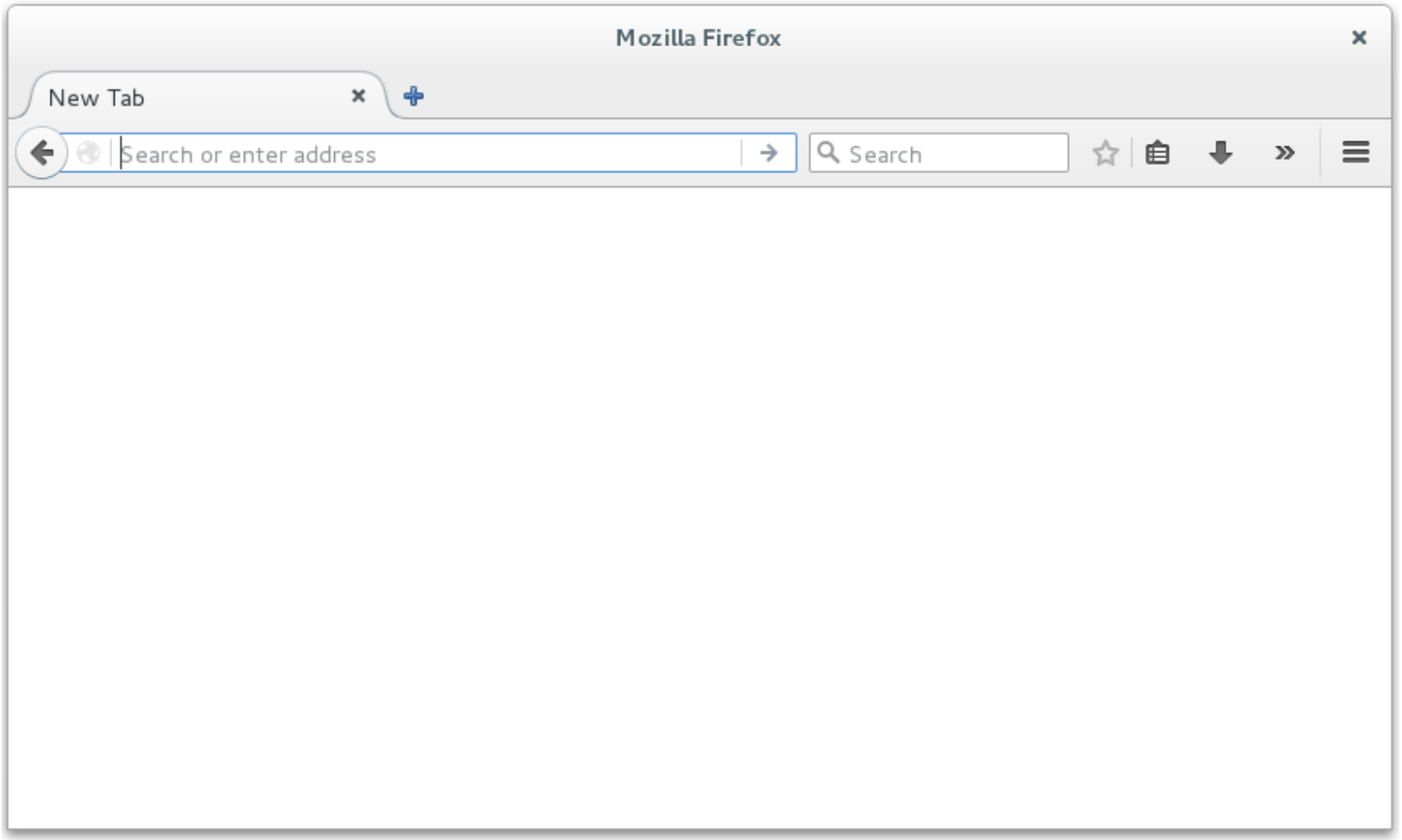
In this example, both columns will be **full width** in both **xs** and **sm** screens, since Bootstrap goes from the smallest screen up. *4-4-16*

```
<div class="col-md-6">
    ...
</div>
<div class="col-md-6">
    ...
</div>
```

In the other devices, **md** and **lg**, the columns will take up 50% each of the width.

With that said, every device will follow the **width of the smallest screen** until there is something else given. *4-4-17*

4-5. Viewport meta tag



The area of a web page that is visible for the user, is called **viewport**.

The viewport **varies with device**, meaning it will be smaller on mobile phones compared to computer screens. *4-5-2*

Before the use of mobile phones and tablets, it was common for web pages to use **fixed size designed only for computer screens**. *4-5-3*

The `viewport meta tag` lets web designers take **control over the viewport**. *4-5-4*

It is a way to give the browser instructions on how to control dimensions and scaling of the page.

In our case, for the responsiveness to work properly, we want to add the **viewport meta tag** as following: *4-5-5*

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

This ensures **proper rendering** and **touch zooming** on your site. *4-5-6*

`width=device-width` sets the width to follow the screen-width of the device. *4-5-7*

`initial-scale=1.0` sets what zoom level of the page when loaded by the browser.

Some browsers has this covered by default, **Firefox** for example. *4-5-8*

But for browsers such as **Chrome**, we need to set this ourselves. *4-5-9*

A good practice is to always include the `viewport meta tag`.

Disable zooming

We can also **disable zooming capabilities** on mobile devices by extending the meta tag content attribute.

```
<meta name="viewport" content="width=device-width, initial-scale=1,
                               maximum-scale=1, user-scalable=no">
```

Use caution! It is not recommended everywhere.

EXERCISE

Bootstrap basic grid system

4-6. Nesting grids

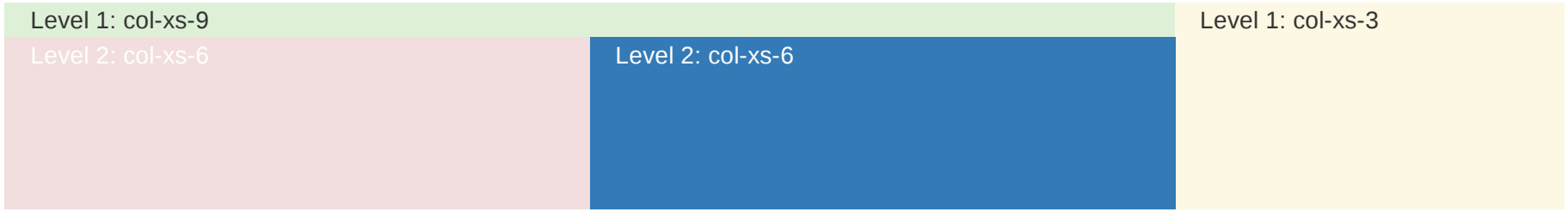
With Bootstrap responsive grids, we can **nest** one grid in another. *4-6-1*

This is done by adding a new row with a set of columns inside of an already existing column. *4-6-2*

```
<div class="row">
  <div class="col-xs-9">
    ...
    <div class="row">
      <div class="col-xs-6">
       ...
      </div>
      <div class="col-xs-6">
       ...
      </div>
    </div>
  </div>
  <div class="col-xs-3">
    ...
  </div>
</div>
```

Could look like this for example:

4-6-3



Exercise

4-6-4

Bootstrap nesting grid

4-7. Column offset

We can move columns to the right using `.col-sm-offset-*`. *4-7-1*

We can use this for every **media query range** so it could might as well be `.col-lg-offset-*`. *4-7-2*

We move the columns somewhere along the 12 column range, without necessarily using all 12 per row. *4-7-3*

The code behind: *4-7-4*

```
<div class="container-fluid">
    <div class="row">
      <div class="col-xs-4">.col-sm-4</div>
      <div class="col-xs-4 col-xs-offset-4">
        .col-xs-4 .col-xs-offset-4
        </div>
    </div>
    <div class="row">
      <div class="col-xs-6 col-xs-offset-3">
        .col-xs-6 .col-xs-offset-3
      </div>
    </div>
</div>
```

As we know, Bootstrap is **mobile first**, so offset for **extra small** screen will impact all the other screen sizes. *4-7-5*

Offset for medium screen will cover medium and large screen, and so on.

If we have a given offset for **extra small** screen, we can override that offset on other screen with `.col-*-offset-0`. *4-7-6*

If we have `.col-xs-offset-3` on a column we could override the offset on a medium screen using `.col-md-offset-0`. *4-7-7*

Exercise *4-7-8*

Bootstrap offset

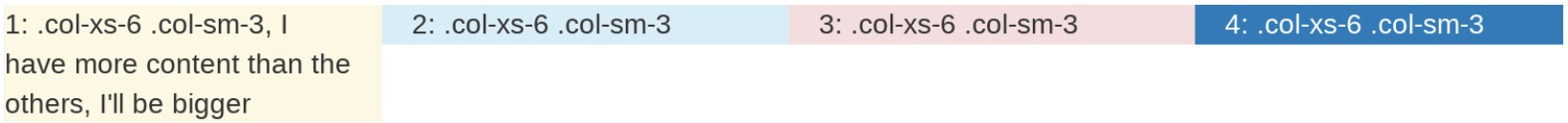
4-8. Column resets

Scenario: We have a row with four columns with different column size depending on device and the first column has more content. 4-8-1

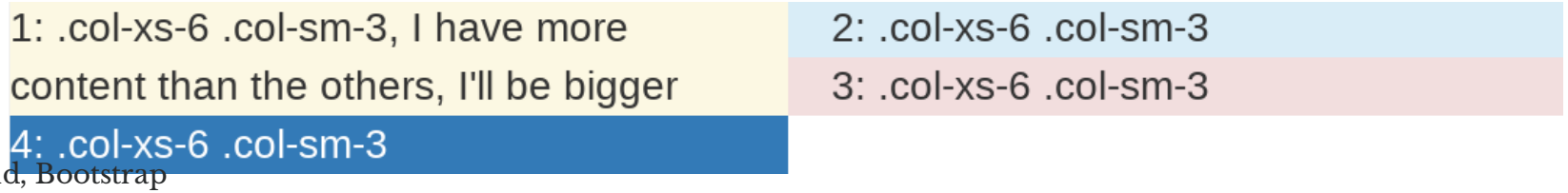
```
<div class="row">
  <div class="col-xs-6 col-sm-3">
    1: .col-xs-6 .col-sm-3,
    I have more content than the others,
    I'll be bigger
  </div>
  <div class="col-xs-6 col-sm-3">2: .col-xs-6 .col-sm-3</div>
  <div class="col-xs-6 col-sm-3">3: .col-xs-6 .col-sm-3</div>
  <div class="col-xs-6 col-sm-3">4: .col-xs-6 .col-sm-3</div>
</div>
```

It gives us the following... 4-8-2

In sm view:



In xs view:



We can see that in the view for **extra small device** the columns are not lined up correctly. *4-8-3*

Column *1* and *2* should be on the same row, while the other two should be on another separate row.

To solve this we can add a new div between those columns that we know should appear on separate rows. *4-8-4*

The block should have two classes applied: *4-8-5*

```
<!-- Add the extra clearfix for only the required viewport -->
<div class="clearfix visible-xs-block"></div>
```

`.clearfix` adds the necessary style for the correct line up: *4-8-6*

```
.clearfix:before, .clearfix:after {
    display: table;
    content: " ";
}

.clearfix:after {
    clear: both;
}
```

`.visible-xs-block` will show the block on extra small screens only. *4-8-7*

```
@media (max-width: 767px) {
    .visible-xs-block {
        display: block !important;
    }
}
```

This will be discussed in a later chapter.

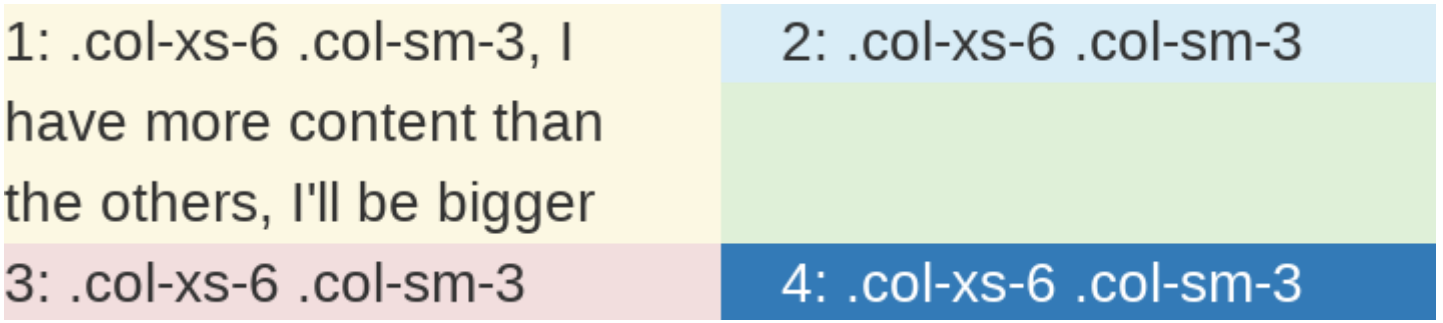
Adding this to our code we now have: 4-8-8

```
<div class="row">
  <div class="col-xs-6 col-sm-3">
    1: .col-xs-6 .col-sm-3,
    I have more content than the others,
    I'll be bigger
  </div>
  <div class="col-xs-6 col-sm-3">2: .col-xs-6 .col-sm-3</div>

  <div class="clearfix visible-xs-block"></div>

  <div class="col-xs-6 col-sm-3">3: .col-xs-6 .col-sm-3</div>
  <div class="col-xs-6 col-sm-3">4: .col-xs-6 .col-sm-3</div>
</div>
```

It will look the same in the *sm* view, but in *xs*: 4-8-9



It fills out the space to keep our rows and columns correctly lined up.

Exercise 4-8-10

Bootstrap resets

4-9. Push and pull columns

We can change the order of columns by using **push** or **pull**. *4-9-1*

```
<div class="row">
    <div class="col-sm-9 col-sm-push-3">
        1st column, but pushed
    </div>
    <div class="col-sm-3 col-sm-pull-9">
        2nd column, but pulled
    </div>
</div>
```

2nd column, but pulled | 1st column, but pushed

In this example the columns will be in the correct order on extra small screens.

On small screens and larger the columns will switch place.

To push or pull we are using the `col-sm-push-*` and `col-sm-pull-*`. *4-9-2*

These can be applied with every **media query range**, so it could also be `col-lg-push-*` for example.

Commonly one column gets pushed as much as the size of the other column, and the same thing with the pull option. *4-9-3*

As shown in the previous example, the first column gets **pushed 3** and the other one gets **pulled 9**, therefor they switch place. *4-9-4*

Exercise *4-9-5*

Bootstrap push and pull