CSS

Responsive Design, Media Queries, Grid System

Floats, Flexbox, CSS Grid

doc. Radek Burget, Ph.D.

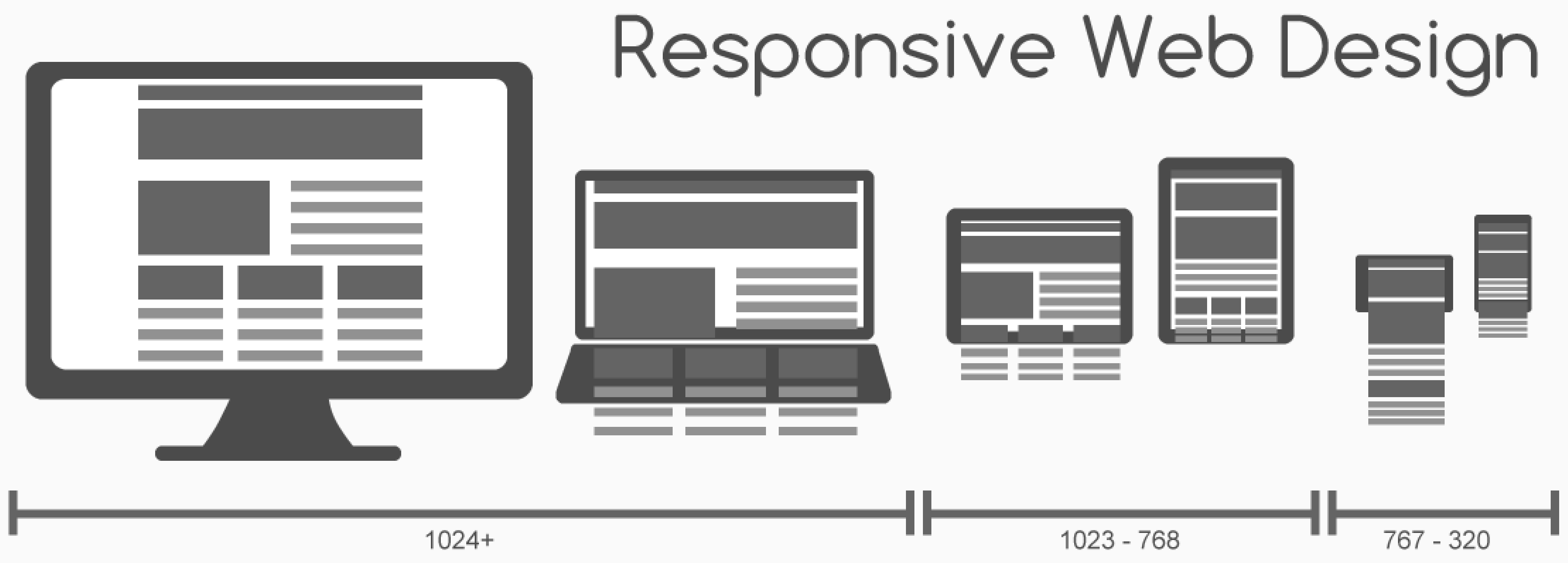
burgetr@vut.cz

Ing. Jiří Hynek, Ph.D.

hynek@vut.cz

Motivation

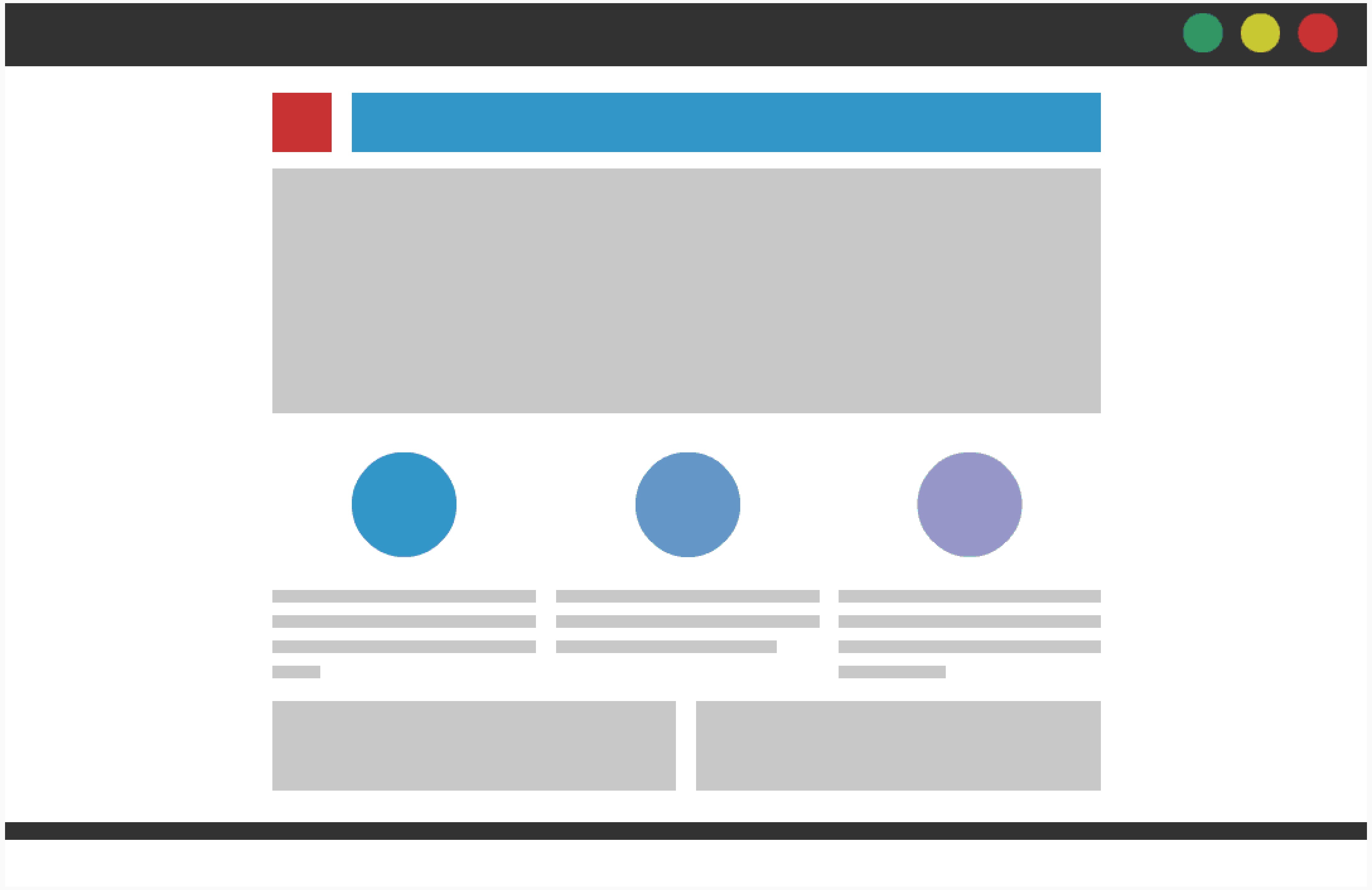
- adjust the document to devices with different resolutions
  - mobile devices, tablets, laptops, standard monitors, …
  - statistics…



fixed × fluid × adaptive × **responsive** layout

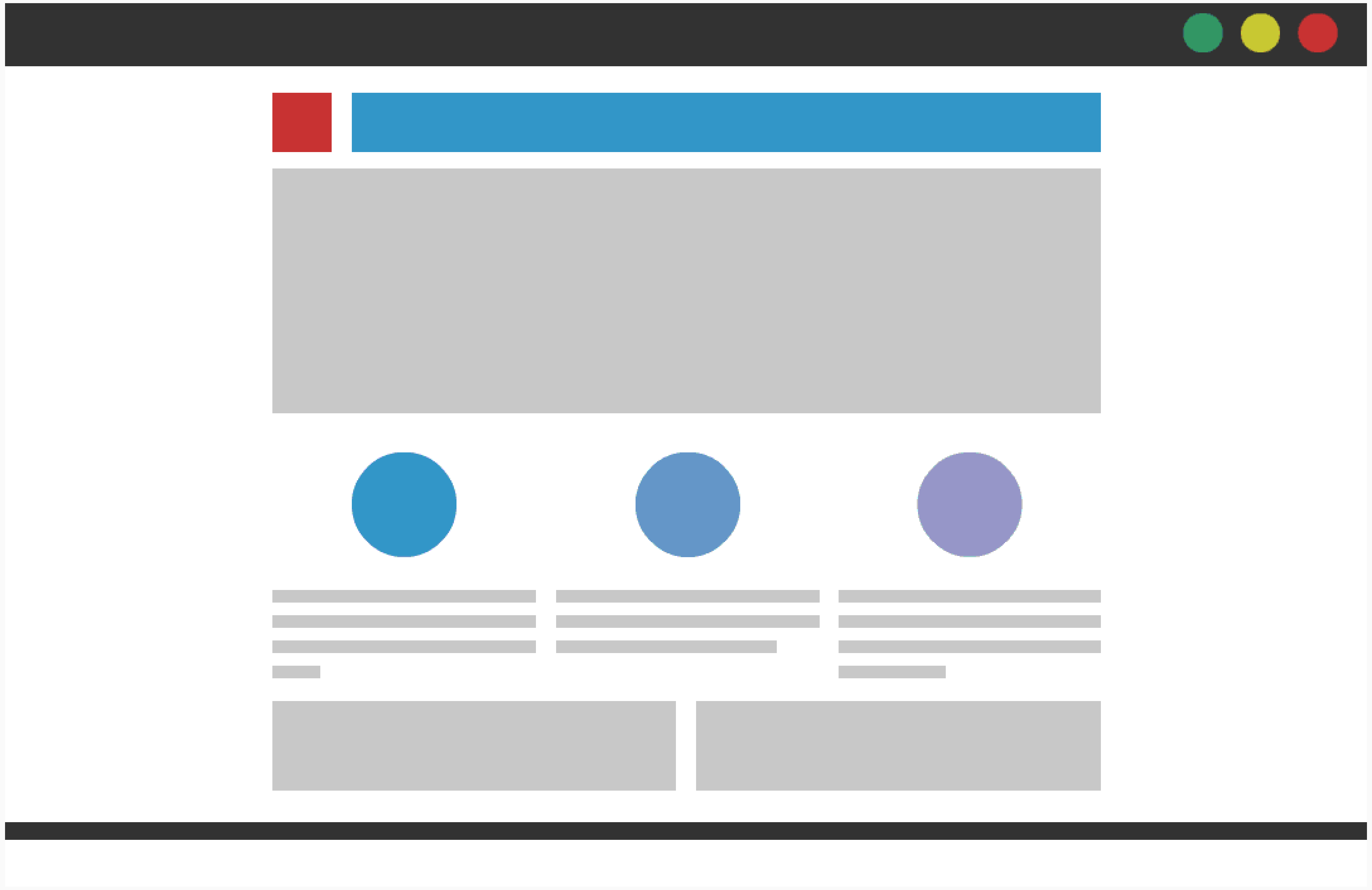
Fluid layout

- width expressed in percentages



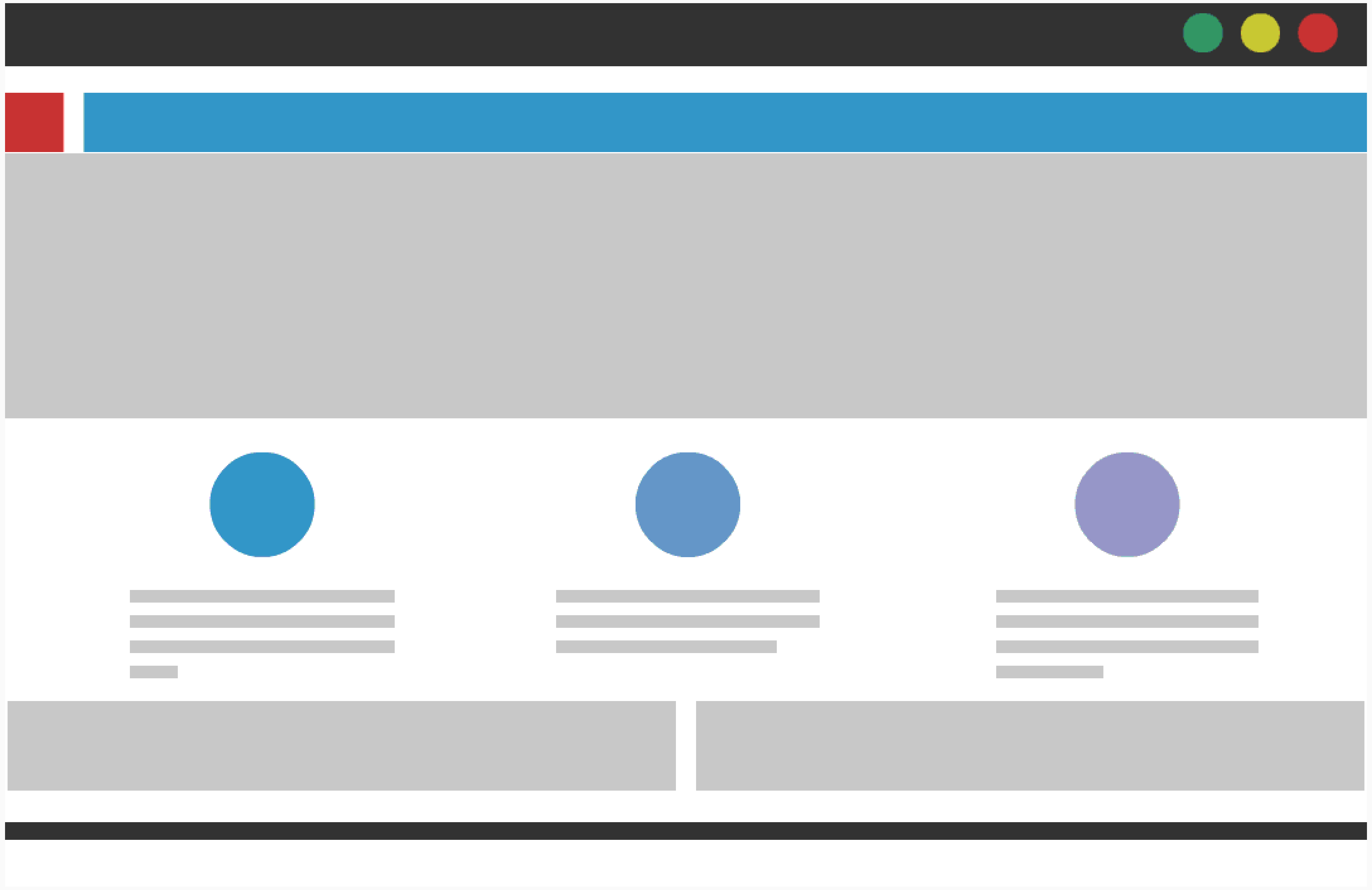
Adaptive layout

- contains so-called breakpoints



Responsive layout

- combination of fluid and adaptive layout

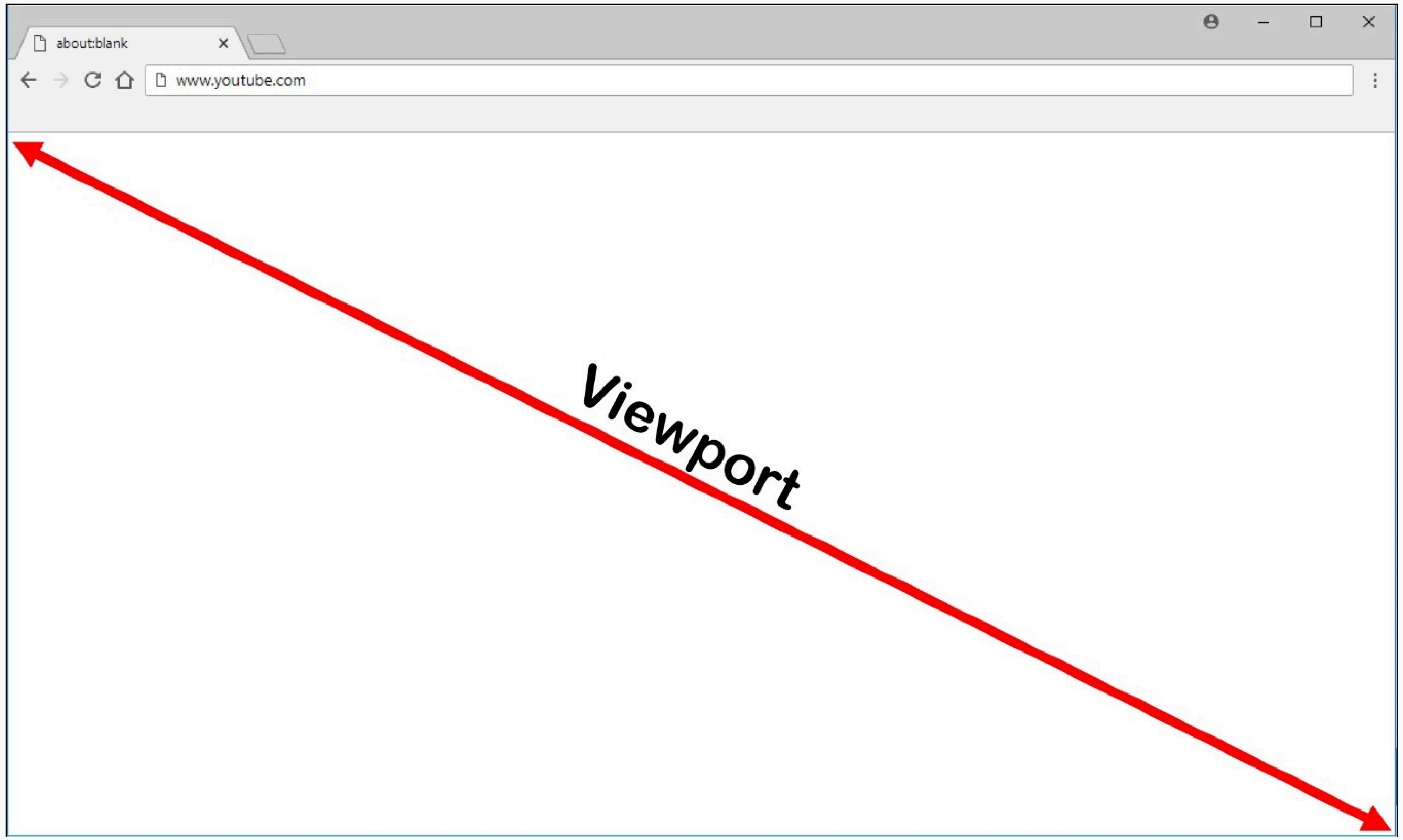


Introduction, Terms

Viewport, Media Queries, Breakpoints, Grid System

Viewport

- the visible part of the document, generally depends on the browser window size



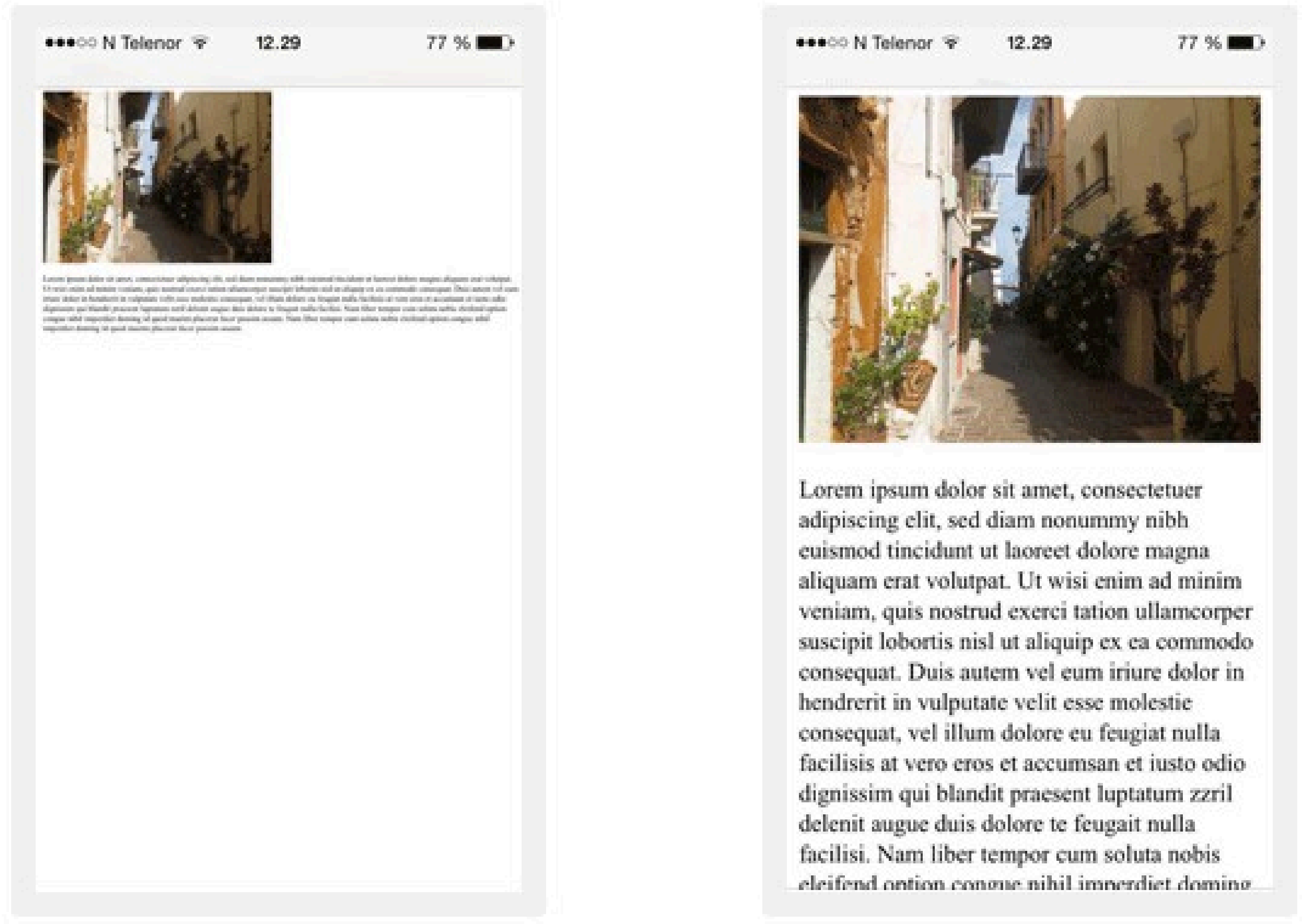
- we are particularly interested in the **available width** (horizontal scrolling is not user-friendly)

Viewport: Mobile Devices

- mobile device browsers have a viewport width larger than their resolution
  - an issue for responsive design

- it is necessary to adjust the HTML document header:

```html
<head>
  <meta name="viewport"
    content="width=device-width,
         initial-scale=1">
</head>
```



Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming

Without the viewport meta tag **With the viewport meta tag**

Design Process

1. design the layout (visual element arrangement) of the document
   starting with mobile devices: (approach *mobile first*)
   - the document usually consists of a single column made up of blocks whose width is relative (usually 100% – full width, possibly with some padding/margin)
   - avoid large fixed-width elements

2. design the document layout **for higher resolutions:**

- dynamically rearrange blocks into multiple columns (floats, flex, grid, …) at higher resolutions
- **CSS3 Media Queries**

Media Queries

- use the **`@media`** rule for conditional style definitions
- CSS3 Media Queries (W3C recommendation)

```
@media not|only mediatype and (expressions) {
  /* CSS rules */
}
```

- **`not`**: negates the entire rule
- **`only`**: older browsers will ignore the construction
- **`mediatype`**: type of media/device
  - `screen`, `print`, `speech`, `all`, …
- **`expressions`**: conditions for the rule (screen size, …)

Media Queries: Conditions and Breakpoints

- **`min-width`**, **`max-width`**, `orientation`, other expressions…
- Media Queries allow limiting (so-called **breakpoints**) the application of rules to specific screen resolutions:

```css
/* extra small screen rules */

@media only screen and (min-width: 576px) {
  /* small screen rules */
}

@media only screen and (min-width: 768px) {
  /* medium screen rules */
}

@media only screen and (min-width: 992px) {
  /* large screen rules */
}

/* etc… */
```

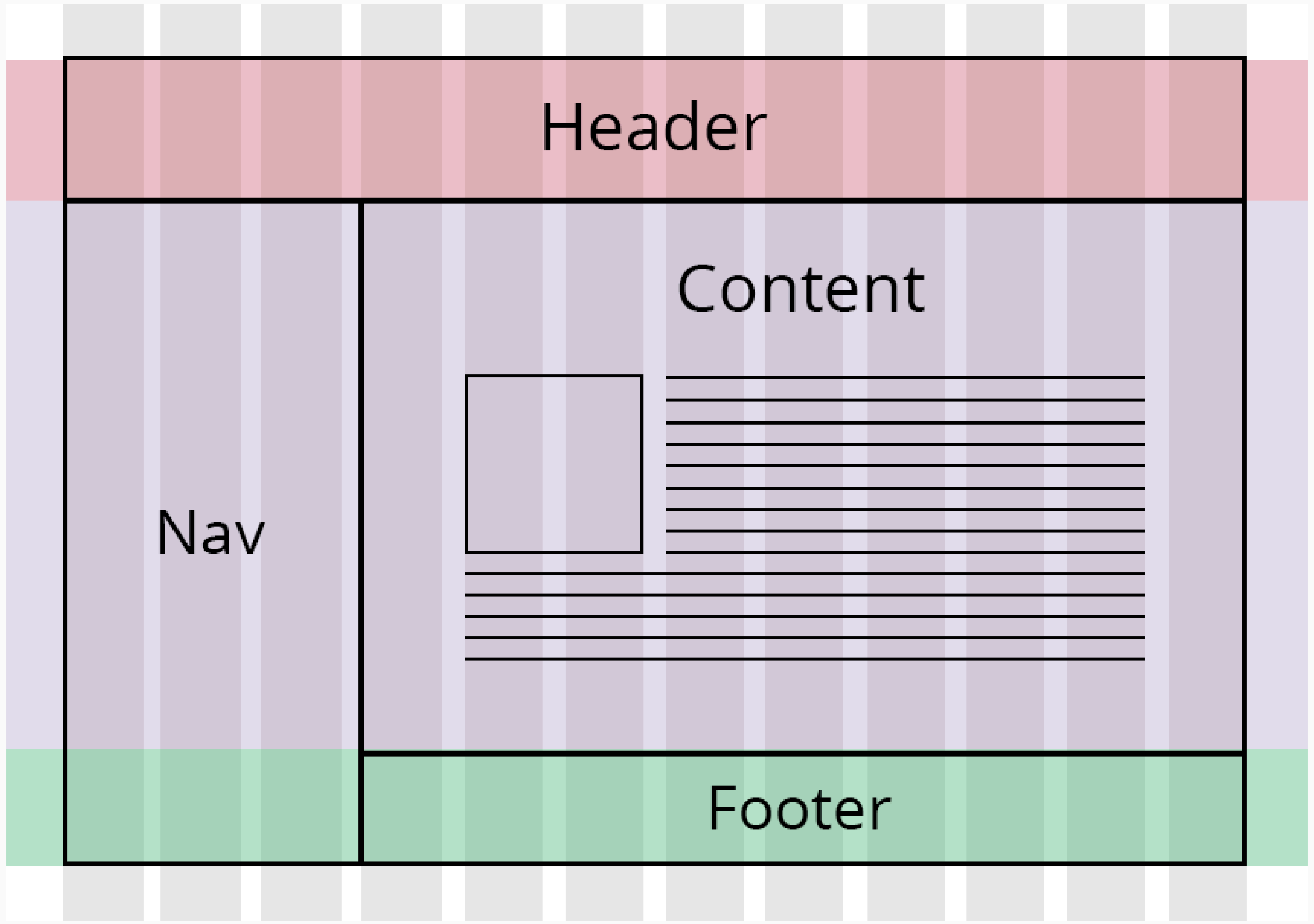
Bootstrap Breakpoints

Breakpoint	Class Prefix	Width
Extra small	none	<576px
Small	`sm`	≥576px
Medium	`md`	≥768px
Large	`lg`	≥992px
Extra large	`xl`	≥1200px
Extra extra large	`xxl`	≥1400px

Grid System

- a technique for organizing elements into a grid consisting of ***n* columns**
  - often **12** – easily divisible, but generally any number

- easier design
- easier implementation
- better organization of elements on the page
- content is perceived better by users



Grid System: Implementation

- it is necessary to address the problem of horizontal positioning of blocks:
  - **floats** (used, for example, in Bootstrap 3)
  - **Flexbox** (used, for example, in Bootstrap 4)
  - **CSS Grid** – an advanced method for creating layouts

- it is **recommended to combine** approaches depending on the specific problem
  - positioning elements in a single row vs. into a grid, etc.

Grid System: Floats

1. Rows

- blocks will be positioned into columns on rows – class **.row**

- necessary to address **issues with floating elements:**
  - clear: both – breaking floating (float) blocks after each row:

```
.row::after {
    content: "";
    clear: both;
}
```

  - overflow: hidden – some columns in a row may have a smaller height than the resulting row height (columns of the next row could overlap the previous row):

```
.row {
    overflow: hidden;
}
```

2. Columns

- elements will be organized into columns within a row:
- necessary to define classes for columns of different widths:

```
[class*="col-"] { float: left; }

.col-1 { width: 25%; } /* 1/4 */
.col-2 { width: 50%; } /* 2/4 */
.col-3 { width: 75%; } /* 3/4 */
.col-4 { width: 100%; } /* 4/4 */
```

  - the **`float`** property causes elements to float and align side by side
  - the **`width`** property determines how much space a column will relatively occupy in a row
  - the number of columns can be chosen to best suit the specific problem
  - a 12-column layout is often used (easily divisible)

2. Columns: HTML

- classes will be assigned to elements in the HTML document based on the desired layout:

```html
<div id="content" class="row">
  <div id="sidebar" class="col-1">
    <!-- 1/4 -->
  </div>
  <div id="article" class="col-3">
    <!-- 3/4 -->
  </div>
</div>
```

```
     sidebar
1/4
```

Example

3. Breakpoints

- use **Media Queries** for breakpoints for different browser window (viewport) widths
  - e.g., Bootstrap breakpoints, …

```
/* Small devices – default display */
[class*="col-"] { float: left; width: 100%; }

/* Medium devices (tablet) */
@media only screen and (min-width: 768px) {
  .col-md-1 { width: 25%; }
  .col-md-2 { width: 50%; }
  …
}

/* Large devices (PC) */
@media only screen and (min-width: 1200px) {
  .col-lg-1 { width: 25%; }
  .col-md-2 { width: 50%; }
  …
}
```

3. Breakpoints: HTML

- classes will be assigned to elements in the HTML document based on the desired layout for different width groups:

```html
<div id="content" class="row">
  <div id="sidebar" class="col-md-2 col-lg-1">
  </div>
  <div id="article" class="col-md-2 col-lg-3">
  </div>
</div>
```

sidebar

Example

4. Border and Padding

- often, we want to set some **`padding`** or **`border`** for blocks positioned into columns

```
border: 0
padding: 0
```

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
```

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
```

Actual box width = `margin` + `border` + `padding` + `width`

Example

Summary

- a way to create a Grid system when Flexbox and Grid were not available

- **disadvantages**:
  - need to set `overflow` and `clear` properties
  - column height may not fill the row height
  - floating elements are generally better suited for text wrapping, not layout creation
  - gaps between columns must be handled using `padding`

- Solution: **Flexbox**

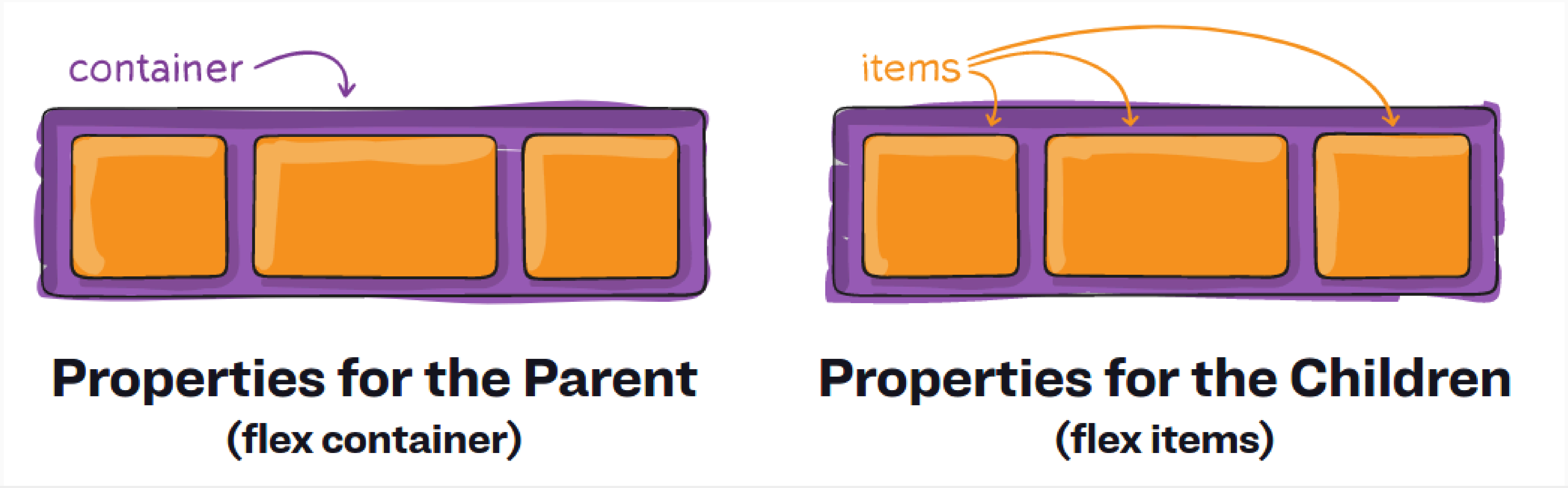
Grid System: Flexbox

Flexbox

- a tool for arranging elements in one dimension (row/column)
  - covered in lecture CSS – Page Layout

- the layout consists of:

1. **containers** (*flex container*)
2. **container items** (*flex items*)



1. Flex Container: Row

- a parent element that contains other elements we want to position within the container

```
.row {
  display: flex;

  flex-direction: row; /* setting the main axis - row orientation
                    not necessary to set, default value */

  flex-wrap: wrap; /* items will wrap if they overflow */

  align-items: stretch; /* alignment of items on the cross axis
                     row items will occupy the full height of the row */
}
```

2. Flex Container Items: Columns

```
[class*="col-"] {
  flex-basis: 100%; /* base size */
  flex-grow: 1; /* width growth will be even for all columns */
  flex-shrink: 1; /* width reduction will be even for all columns */
}

@media only screen and (min-width: 768px) {
  .col-sm-1 { flex-basis: 25%; }
  .col-sm-2 { flex-basis: 50%; }
  .col-sm-3 { flex-basis: 75%; }
  .col-sm-4 { flex-basis: 100%; }
}

/* etc… */
```

All Together

body

- About the Course
- Schedule
- Lectures
- Exercises

Example

Summary

- especially useful when positioning elements within one dimension (e.g., in a row)
  - compared to floats, it offers additional properties (e.g., `align-items`, `gap` …)

- responsiveness can be achieved in alternative ways:
  - using the `wrap` property combined with `min-width` – items will wrap when the page width decreases
  - defining `flex-grow` and `flex-shrink` properties for each element individually + combining with Media Queries
  - …

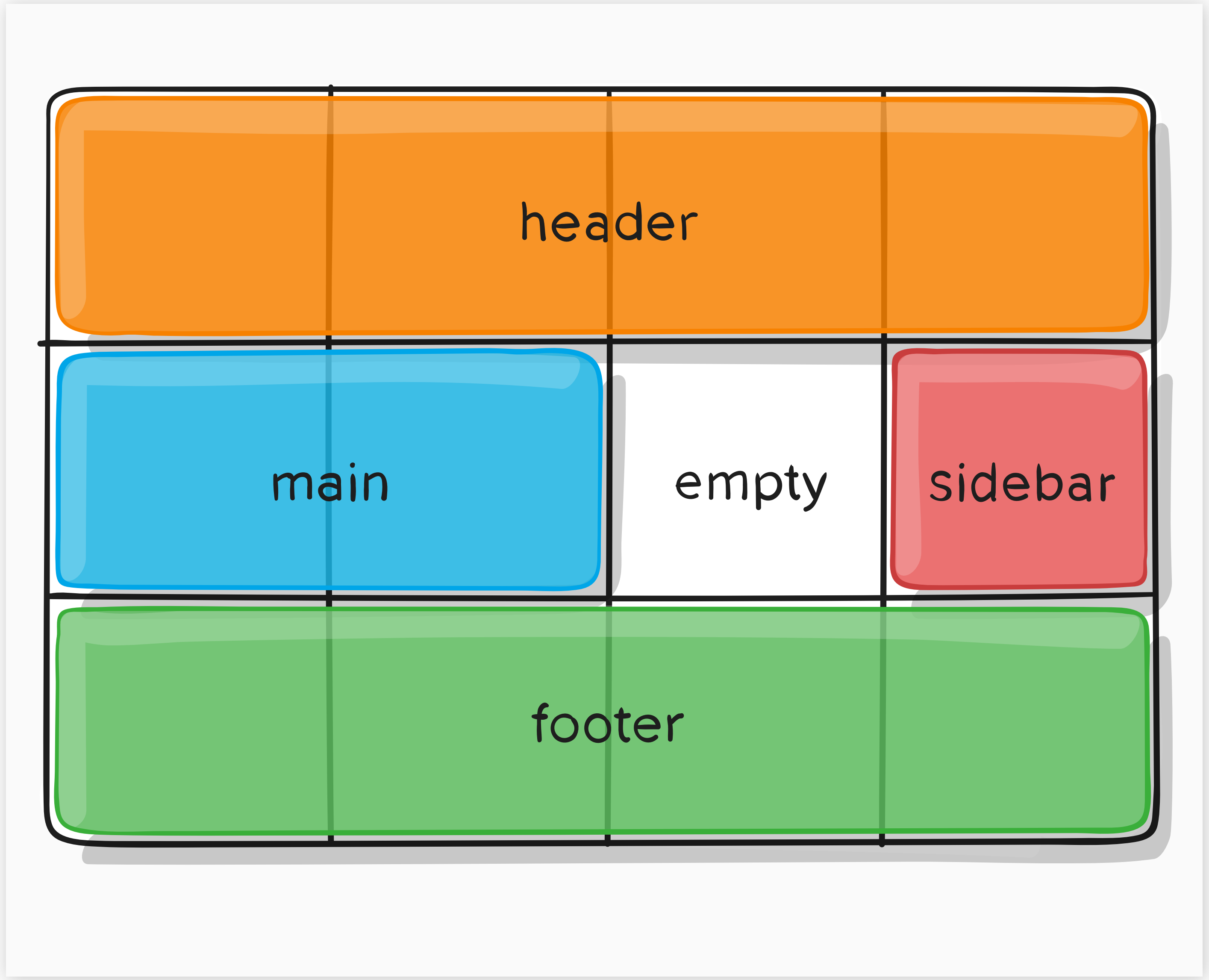
- **disadvantages**
  - still necessary to wrap row elements in a separate container
  - the Grid system uses utility classes in HTML (`row`, `col-`)
  - solution: **CSS Grid**

Grid System: CSS Grid

CSS Grid

- a tool for arranging elements in two dimensions (rows + columns)
  - covered in lecture CSS – Page Layout
- the most advanced tool for arranging elements in CSS so far
- allows easy definition of a Grid system without modifying the HTML document

- the layout consists of:
  1. **container** (*grid container*)
     - contains a grid (*grid*) made up of cells (*cells*)
     - groups of cells form tracks or areas (*tracks*, *areas*), which map to container items (*grid items*)
  2. **container items** (*grid items*)
     - elements (direct children) of the container



1. Grid Container: Layout

- the parent element that forms the grid
- we choose a layout strategy using **`grid-template-areas`**

```css
.container {
  display: grid;

  grid-template-columns: repeat(4, 1fr); /* 4-column layout */

  grid-template-areas: "header header header header"
              "sidebar content content content";
              /* custom naming of areas - creating the layout itself */
}
```

header

header

Example

2. Grid Container Items

```
header {
  grid-area: header;
}

nav {
  grid-area: sidebar;
}

article {
  grid-area: content;
}
```

```
<div class="container">
  <header>header</header>
  <nav>nav</nav>
  <article>article</article>
</div>
```

```
header

header
```

Example

3. Responsive Grid

```
.container {
  display: grid;
  grid-template-columns: 1fr;
  grid-template-areas: "header"
              "sidebar"
              "content";
}
@media only screen and (min-width: 768px) {
  .container {
  grid-template-columns: repeat(2, 1fr);
  grid-template-areas: "header header"
              "sidebar content";
  }
}
@media only screen and (min-width: 1200px) {
  .container {
  grid-template-columns: repeat(4, 1fr);
  grid-template-areas: "header header header header"
              "sidebar content content content";
  }
}
```

3. Responsive Grid

body

- About the Course
- Schedule
- Lectures

Example

Responsive Grid Without Media Queries

```css
.container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr) minmax(300px, 1fr));
  gap: .5rem;
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur tempus

Example

Summary

- CSS Grid provides advanced options for arranging elements
  - compared to Flexbox, it allows 2D arrangement (rows + columns) – complete layout definition
  - no need to define a container for each row
  - no need to use utility classes in the HTML document (`row`, `col-`)

Summary

links

Key Information

- **Responsive layout** = fluid + adaptive layout
- **Viewport** – visible part of the document (browser window)
- **Grid system** – technique for organizing elements into a grid consisting of n columns (12)
- **Mobile first** approach – design the layout for small widths first, then progressively for larger ones
- **Media Queries**
- **breakpoints**
- **floats** vs. **Flexbox** vs. **CSS Grid**
  - these approaches are not mutually exclusive; it is often appropriate to combine them depending on the specific situation

- **it is important to try everything out practically**
- tutorial:
  - https://www.w3schools.com/css/css_rwd_intro.asp

To be continued…

CSS3, frameworks