

# L4: CSS Responsive Design

**Web Engineering**  
188.951 2VU SS20

**Jürgen Cito**

# L4: CSS Responsive Design

- Media Queries
- Responsive and Adaptive Images and Fonts
- Flexible Box Layouts (Flexbox)
- Grid Layouts

# Learning Goals

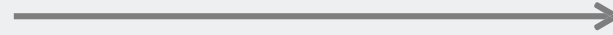
- Differentiate between different options to achieve responsive layouts
- Understand how images and fonts can be made responsive
- Properly use media queries for responsive design
- Understand the role of Flexbox and Grid layouts

# Web Layout Approaches

Responsive Design is a way of implementing web layouts based on current standards, HTML5 and CSS3.

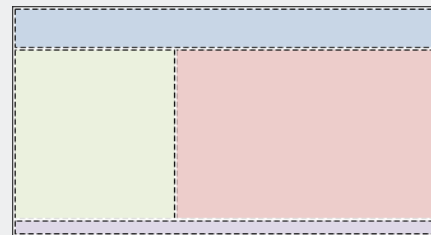


Interface A

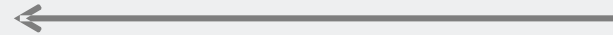


Interface B

Graceful degradation

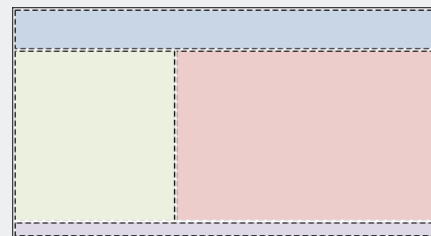


Interface A

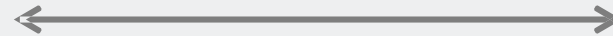


Interface B

Mobile-first /  
Progressive enhancement



Interface A



Interface B

Responsive design

# Responsive Design

Let **content fill the container** and define min/max constraints

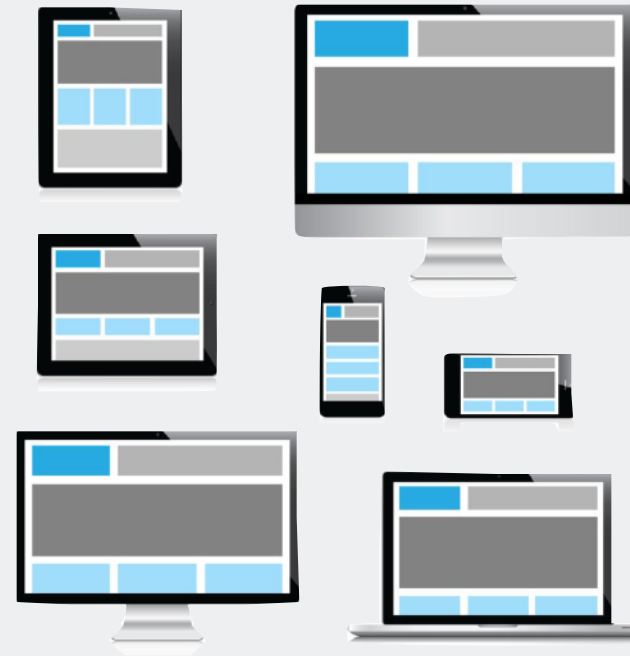
Use **relative units** to specify position and size of text and media

## Techniques

- Media Queries
- Fluid, Grid-based Layout
- Responsive Images
- Font Scaling
- ...

Required Reading:

<http://alistapart.com/article/responsive-web-design/>



# Media Queries

- Previously only media types (screen, print, braille, handheld ...)
- @media rule
- Additional features
  - color
  - aspect-ratio
  - max-width
  - orientation
  - resolution
  - scan
  - ...
- Build complex queries using logical operators (not, and, only)

```
@media only screen and (max-width: 500px) { ... }  
@media tv and (min-width: 700px) and (orientation: landscape) { ... }  
<!-- comma acts as 'or' -->  
@media (min-width: 700px), handheld and (orientation: landscape) { ... }
```

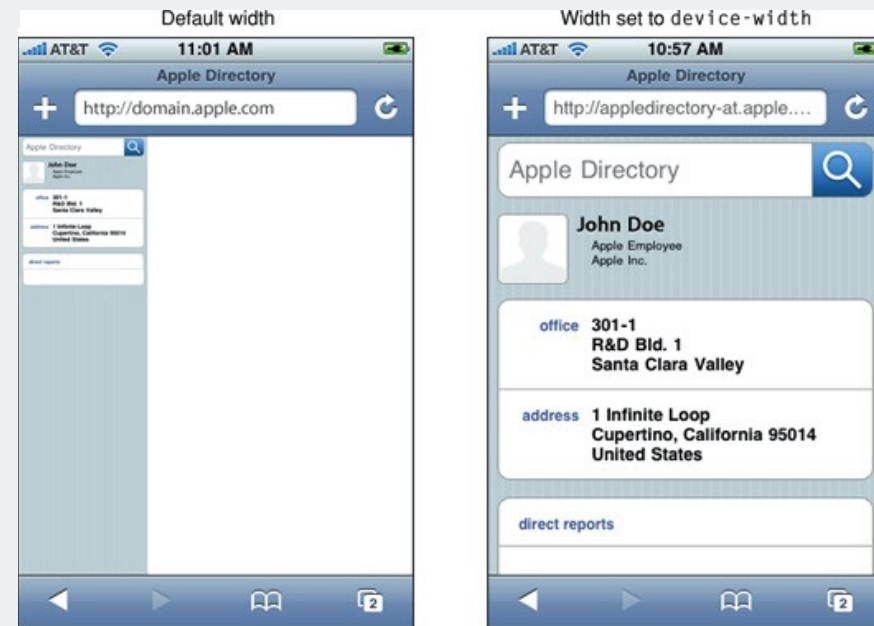
# Viewport

## The virtual “window”

### Viewport

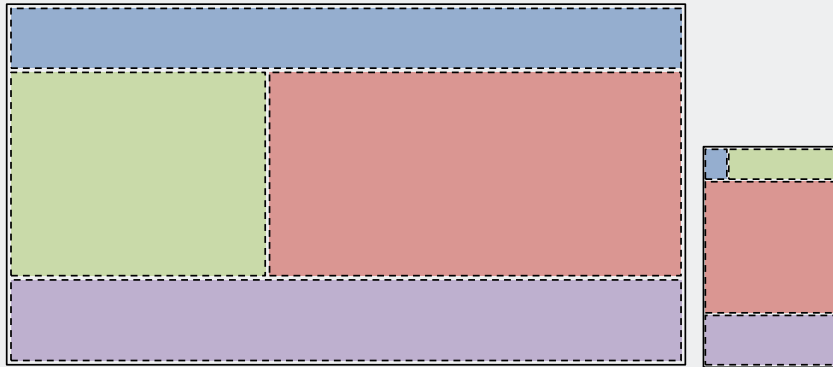
- Visible area of webpage
  - Size content to viewport
  - Avoid horizontal scrolling
  - Avoid necessity to zoom
- 
- Control viewport in HTML5
    - Through meta-element
  - Consider mobile
    - Sometimes narrower/wider
    - Prevent default shrinking

```
<meta name="viewport"
      content="width=device-width,
              initial-scale=1.0,
              maximum-scale=3,
              minimum-scale=0.5" />
```



<https://developer.apple.com/library/safari/documentation/AppleApplications/Reference/SafariWebContent/UsingtheViewport/>

# Media Queries



```
#header {  
  height: 200px;  
}  
#navigation {  
  width: 300px;  
}  
#content {  
  width: 900px;  
}  
#footer {  
  height: 200px;  
}
```

```
@media screen and (max-width: 768px) {  
  #header {  
    width: 80px;  
    height: 120px;  
    display: inline;  
  }  
  #navigation {  
    width: 560px;  
    height: 120px;  
  }  
  #content {  
    width: 640px;  
    height: 760px;  
  }  
  #footer {  
    height: 80px;  
  }  
}  
@media screen and (max-width: 1024px) {  
  ...  
}
```



# Media Queries and Fluid Layouts

Use of CSS3 media queries for defining breakpoints and style switches

```
@media screen and (max-width: 680px) { ... }
```

However...

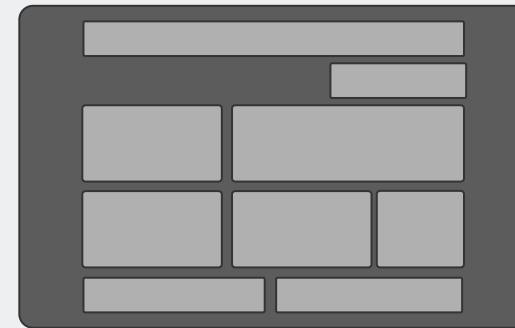
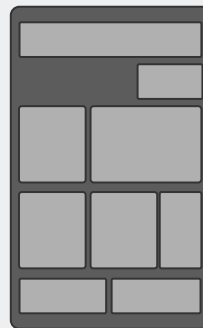
- No linear progression between fix-sized designs
- Snaps at break points
- Horizontal scrolling may be required in-between
- Doesn't allow styling for future or unknown devices



Fluid Layout: Deal with the "grey area" between breakpoints

- Use relative values for sizes to resize
- Consider maximum width for content

```
#info { width: 50%; }  
#container { max-width: 960px; }
```



# Fluid Images

- Scale images like content using relative values
- Problems
  - Browser needs to scale
  - Large download

```
img {  
  width: 50%;  
  max-width: 100%;  
}
```



# Responsive and Adaptive Images

- Detect visitor screen size, resolution, and pixel density
- Fetch respective image, size or version (additional download!)
- Use JavaScript, CSS media queries, and/or HTML5 markup

```
<picture>
  <source src="pic-mobile.jpg" media="(max-width: 720px)" />
  <source src="pic-tablet.jpg" media="(max-width: 1280px)" />
  <source src="pic-desktop.jpg" />
   <!-- User Agent not supporting picture element -->
</picture>
```



# Scaling Fonts

## Scaling Fonts

- Use media queries and relative values for your fonts
- Font scales according to viewport and stays readable

```
@media screen and (min-width: 481px) and (max-width: 768px) {  
  p { font-size: 0.9em; }  
}  
@media screen and (max-width: 480px) {  
  p { font-size: 0.7em; }  
}
```

## What's wrong with pixels?

- Dependent on screen resolution
- Resolution increase → font size decrease
- Many modern mobile devices have high-density screens → difficult to read

# CSS Layout Modes

## Layouts

- Block, Inline, Table, Positioned, Grid, Flexible, ...
- Not all CSS properties apply to all modes
- Support for layouts still vary
  - Check with: <http://caniuse.com>

```
display: <mode>;
```

```
<p>Lorem ipsum dolor sit  
amet, consectetur  
adipiscing elit...</p>
```

```
-webkit-column-count: 3;  
-moz-column-count: 3;  
column-count: 3;
```

Lorem ipsum dolor sit amet, consectetur 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 dui dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis

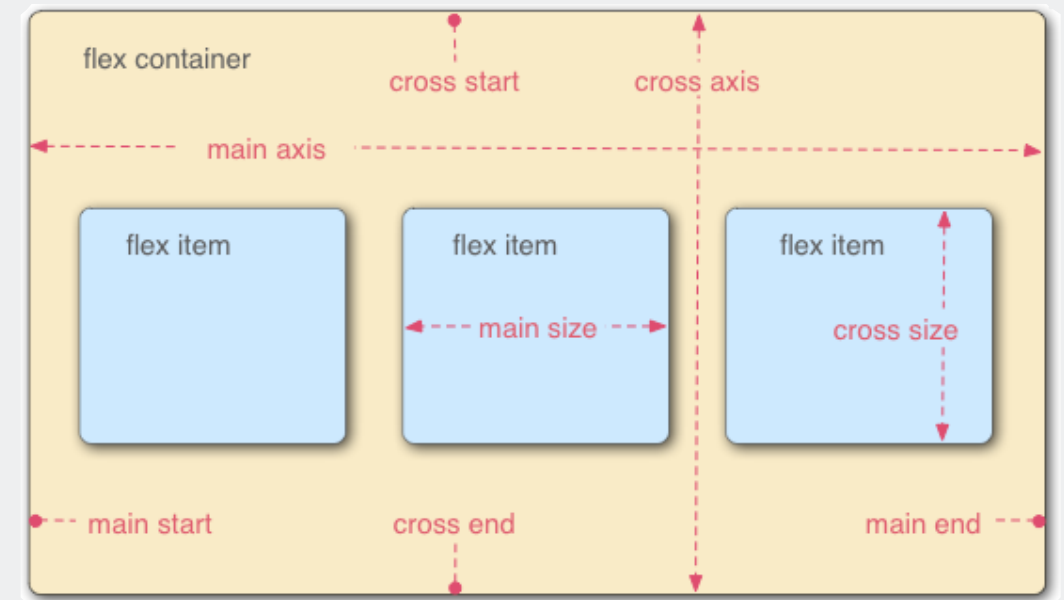
eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius.

# Layout Modes - Flexbox

```
display: flex;
```

Flexbox enables aligning and distributing elements within a container

- Expands items to fill available free space
- Shrinks items to prevent overflow
- Flex container: Contains flex items
- Flex items: Define properties on how the element should align and flow within the container



W3C: <https://www.w3.org/TR/css3-flexbox/>

Image credit: [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Flexible\\_Box\\_Layout/Using\\_CSS\\_flexible\\_boxes](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Using_CSS_flexible_boxes)

# Layout Modes - Flexbox

- Container Properties
  - flex-direction, flex-wrap, flex-flow, justify-content, align-items, align-content
- Item Properties
  - order, flex-grow, flex-shrink, flex-basis, align-self



# “Holy Grail Layout” with Flexbox

## Holy Grail Layout

- Header, Footer
- Fluid content, fixed sides
- >2 equal height columns
- Content before remaining columns
- “order” property only for **visual** ordering



## Flexible Box Layout

```
main { display: flex; }
main > article { order: 2; min-width: 12em; flex: 1; }
main > nav     { order: 1; width: 200px; }
main > aside   { order: 3; width: 200px; }
```

```
@media all and (max-width: 600px) {
  main { flex-flow: column; }
  main > article, main > nav, main > aside {
    order: 0; width: auto;
  }
}
```

```
<!DOCTYPE html>
...
<header>...</header>
<main>
  <article>...</article>
  <nav>...</nav>
  <aside>...</aside>
</main>
<footer>...</footer>
```





# Layout Modes - Grid

```
display: grid;
```

Grid layout enables control of sizing and positioning of boxes within a grid system

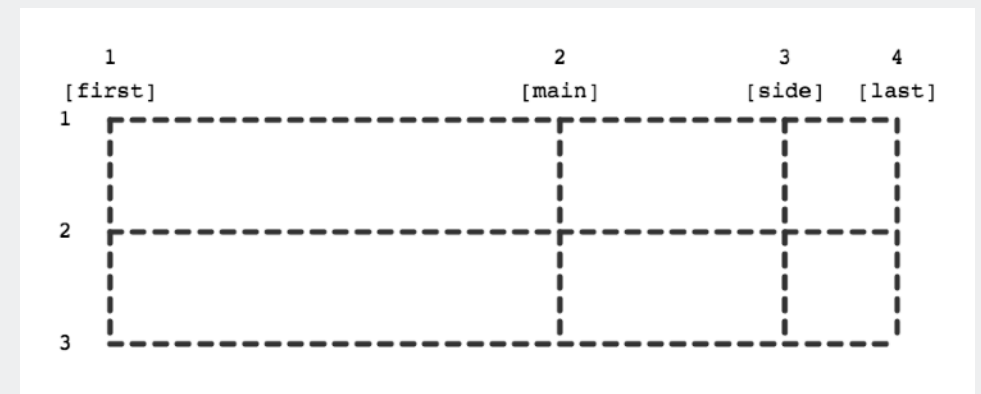
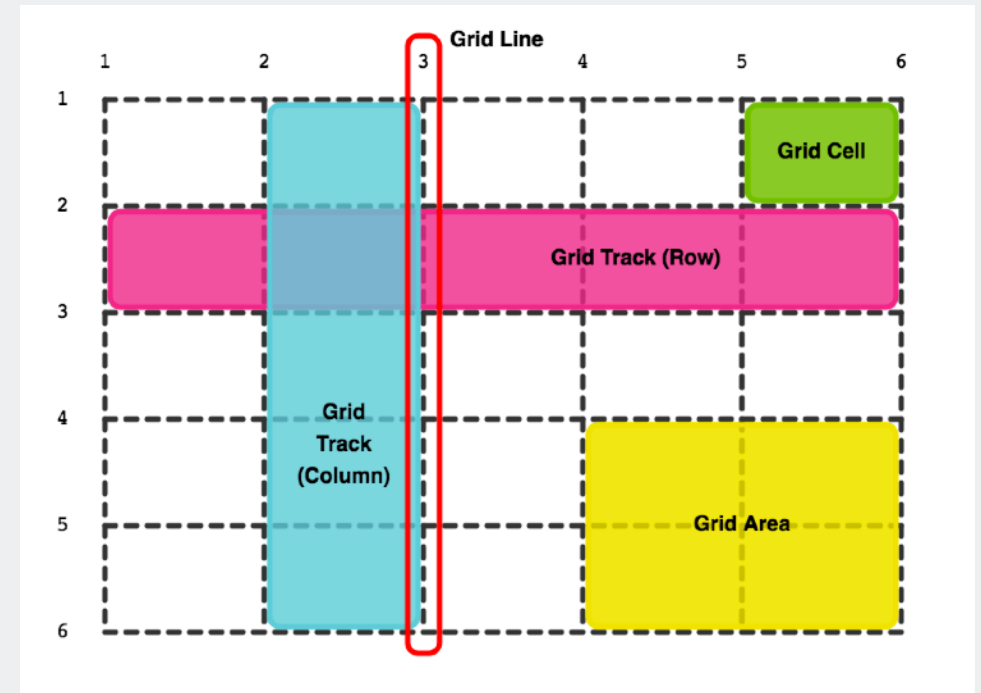
- Grid Line: Horizontal and vertical dividing line within a grid
- Grid Track: Space between two adjacent grid lines — columns or rows of the grid
- Grid Cell: Single unit of the grid
- Grid Area: Adjacent grid cells that form a rectangle

Example: Defining tracks in a grid (2 rows and 3 columns)

```
display: grid;  
grid-template-rows: 100px 100px;  
grid-template-columns: 400px 200px 100px;
```

Name individual lines to reference later

```
grid-template-columns: [first] 400px  
[main] 200px [side] 100px [last];
```



# Layout Modes - Grid

## Placement of child elements in the grid

- `grid-row`, `grid-column`
- Element in one particular grid cell
- Element spanning a grid area

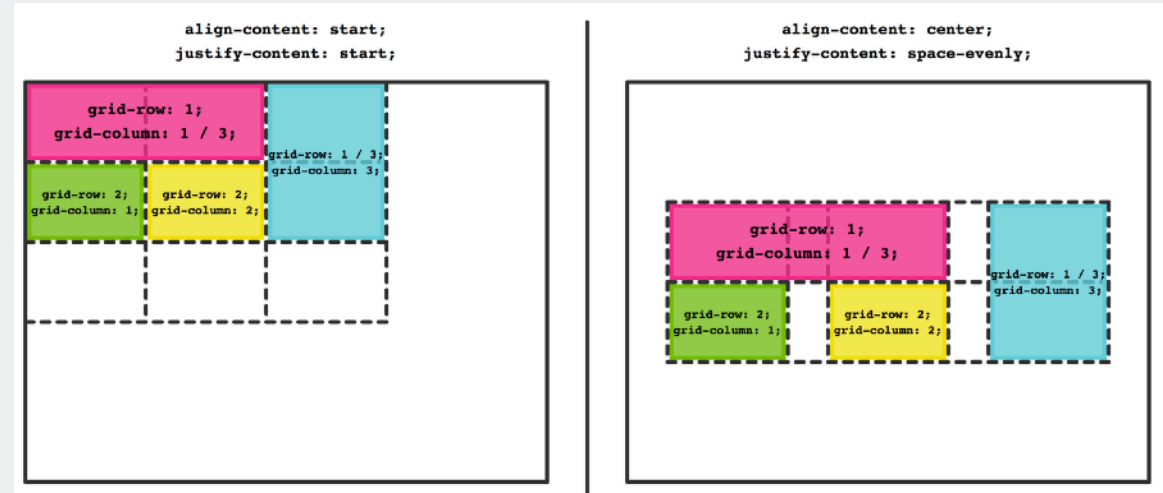
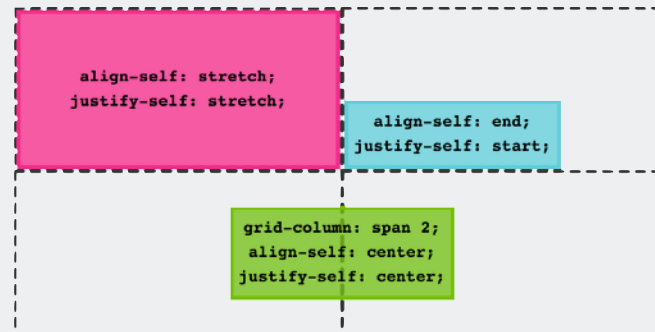
```
grid-row: 3;
```

```
grid-row: 2 / 5;  
grid-column: 3 / span 2;
```



## Horizontal and vertical **alignment** support

- Content distribution — align within the container
  - `justify-content` — align horizontally
  - `align-content` — align vertically
- Aligning elements within grid cell
  - `justify-self`
  - `align-self`



# Layout Modes - Grid

## Fractional Unit “fr”

- One part of the available space
- Can be mixed with other units when defining grid areas
- Examples of 4 column grid layout
  - All 4 columns each taking the same amount of space

```
grid-template-columns: 1fr 1fr 1fr 1fr;
```

- 3rd column has fixed size and 4th column is relative to container size

```
grid-template-columns: 1fr 1fr 20px 20%;
```

- Multiple fractions are the sum of single fractions  
(in this example, the 4th column is the same size as column 1 and 2)

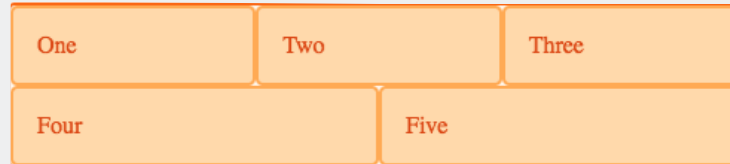
```
grid-template-columns: 1fr 1fr 20px 2fr;
```

# Layout Modes - Flexbox vs Grid

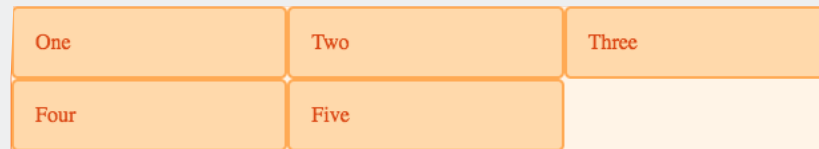
## One-dimensional vs Two-dimensional layout

- Flexbox (1D) - Content first
  - Enable existing content to align and adapt
  - Rules decided by children
- Grid (2D) - Layout first
  - Specific rigid layout in mind children are placed in
  - Declaratively control specifically where elements end up — Rules decided by parent

```
<div class="wrapper">  
  <div>One</div>  
  <div>Two</div>  
  <div>Three</div>  
  <div>Four</div>  
  <div>Five</div>  
</div>
```



```
.wrapper {  
  width: 500px;  
  display: flex;  
  flex-wrap: wrap;  
}  
  
.wrapper > div {  
  flex: 1 1 150px;  
}
```



```
.wrapper {  
  display: grid;  
  grid-template-columns: repeat(3, 1fr);  
}
```

Layout modes are not defined for entire page but for a specific container. Flexbox and Grid are not mutually exclusive within a page. Mix and match as you see fit