



FH Salzburg

# **VO Web-Technologien**

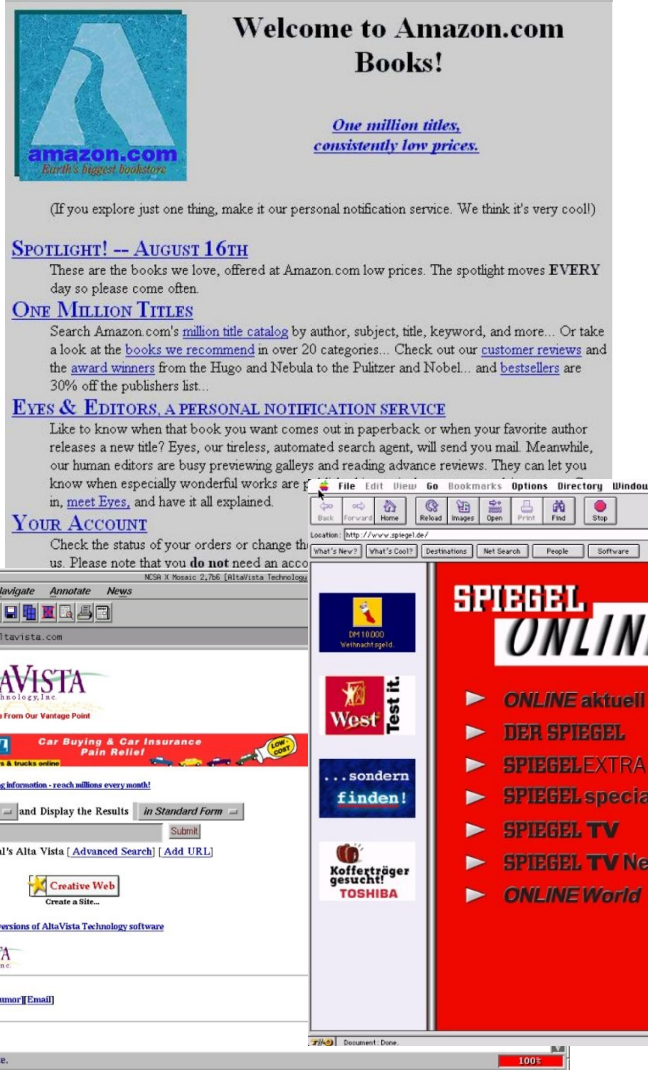
**Einheit 4, Oliver Jung**

Technik  
Gesundheit  
Medien

# Webseiten

Klassisches Design  
mit einer fixen in  
Pixel definierten  
Breite => fixe  
Auflösung, fixer  
Browser und/oder  
fixes Betriebssystem

Fast schon  
ausgestorben...



# Webseiten – Entwicklung „damals“



**Best Viewed in  
Internet Explorer @ 1024**

# Browser Plugins



# Webseiten

Klassisches  
Design mit einer  
fixen in Pixel  
definierten Breite  
=> fixe Auflösung,  
fixer Browser  
und/oder fixes  
Betriebssystem



# Neue Ansätze



Der klassische Ansatz ist **nicht** mehr praktikabel

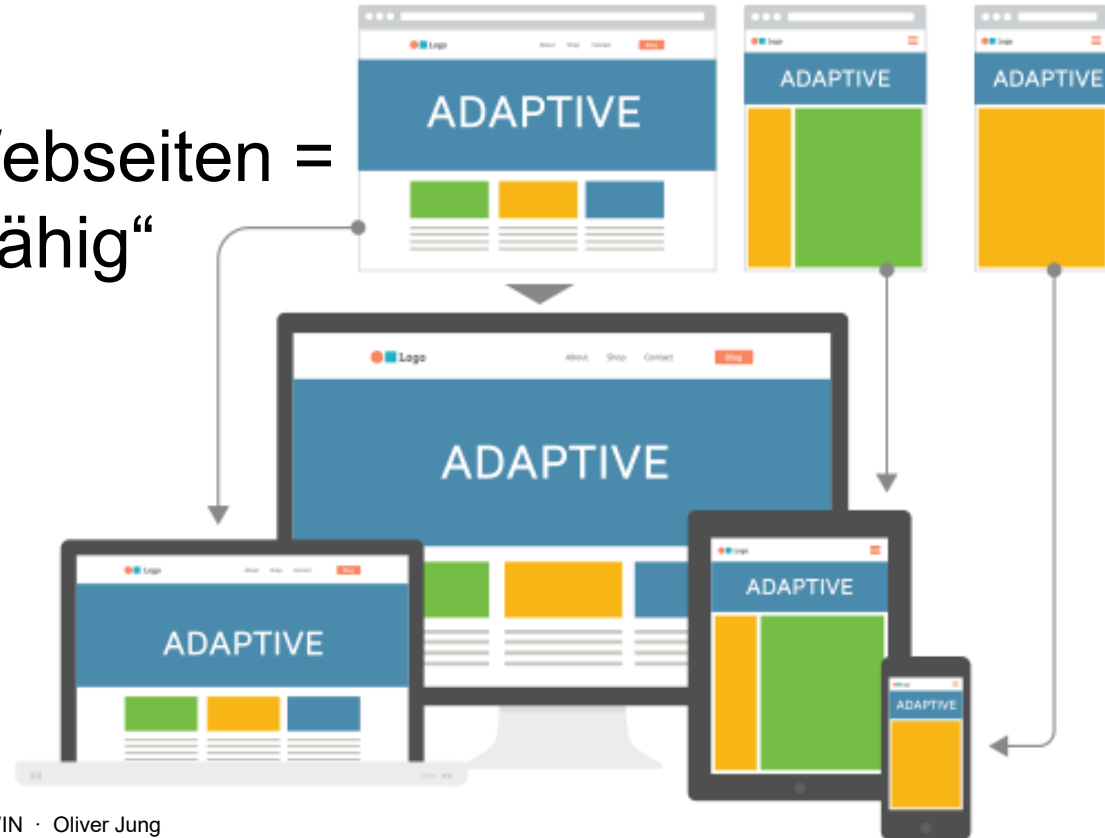
## Neue Wege:

- Adaptive Design
- Responsive Design

# Webseiten Typen



„**Adaptive**“ Webseiten =  
„anpassungsfähig“



# „Adaptive“ Webseiten



- Ein „Adaptive“ Layout oder „Adaptive Web Design“ **AWD** beschreibt ein für bestimmte Displaygrößen optimiertes Web-Layout.
- Nicht für alle Displaygrößen
- => „exakte“ Viewports wie: Desktop, Tablet, Smartphone
- Breitendefinition mit Media Queries
- Oft mit serverseitiger „Automatic Client Detection“



# Webseiten Typen



„**Responsive**“ Webseiten =  
„reaktionsfähig“



# „Responsive“ Webseiten



Ein „Responsive“ Layout oder „Responsive Web Design“ beschreibt eine Lösung um **alle erdenklichen Geräte** zu optimieren. Praktisch stark auf Displaygrößen bezogen, sollten **auch andere Geräteeigenschaften** im Design berücksichtigt werden.

- Design Elemente sollten entsprechend dem Gerät verwendet bzw. dargestellt werden (z.B. Handy: statt pop-up Dialog, eine Sidebar)
- Design Features werden je nach Geräteunterstützung eingeblendet

# RWD vs AWD



Oben Responsive; Unten Adaptive

Animation: <https://css-tricks.com/the-difference-between-responsive-and-adaptive-design>

# „Adaptive“ Webseiten - Vorteile



- Es kann gut mit **klassischen Mockups, Wireframe** und Skizzen gearbeitet werden, da feste Abmessungen existieren
- Ladezeiten können optimiert werden.
- Inhalte müssen nur **für klar definierte Abmessungen optimiert** werden, aber nicht vollkommen flexibel sein
- Viel **gestalterischer Freiraum**, da mit einem starren Raster gearbeitet wird
- Technisch **recht unkompliziert** umzusetzen
- **Zeitsparendere** Umsetzung pro Layout



# „Adaptive“ Webseiten - Nachteile



- Es wird nur für **bestimmte** Viewports / bestimmte Geräte optimiert
- **Häufige Fehldarstellungen** auf abweichenden Endgeräten
- **Aufwändige Zielgruppenanalyse** um die relevanten Viewports zu bestimmen
- **Häufig mehr CSS-Code als notwendig**
- Evtl. nicht SEO freundlich



# „Responsive“ Webseiten - Vorteile



- „**Jede**“ **Displaygröße** wird optimal berücksichtigt
- Es wird **kein Platz verschenkt**
- Die **Information** steht im **Vordergrund**
- **Zukünftige** mobile Endgeräte werden automatisch mit abgedeckt
- SEO freundlicher



# „Responsive“ Webseiten - Nachteile



- Mockups, Wireframes und Skizzen stoßen an ihre Grenzen. Häufig muss mit **Prototypen** gearbeitet werden um Kunden das Verhalten der Website zu zeigen
- **Komplexer** in
  - der Gestaltung
  - der technischen Umsetzung
  - der Anpassung der Seiteninhalte
- **Zeitintensivere** Umsetzung



# Breakpoints - Umbruch



- ... ist der Punkt, an dem das Design für die Größe des Darstellungsfeld umspringt
  - Major breakpoints
  - Minor breakpoints
- wird durch eine @media-Regel (Media Query) beschrieben:

```
.column { width: 90%; margin: 1em auto; }  
.column h2 { font-size: 1.2em; color: firebrick; }
```

```
@media only screen and (min-device-width: 40em) {  
    .column { width: 48%; float: left; }  
}
```



# Beispiel Bootstrap Grid



	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Containerbreite (max.)	None (auto)	540px	720px	960px	1140px

Bei angenommen 16px  
Basisschriftgröße

## Andere Beispiel- Breakpoints



```
/* Small screens: Define mobile styles */
```

```
@media only screen { }
```

```
/* max-width 640px, mobile-only styles */
```

```
@media only screen and (max-width: 40em) { }
```

```
/* Medium screens: min-width 641px, medium screens */
```

```
@media only screen and (min-width: 40.063em) { }
```

```
/* min-width 641px and max-width 1024px */
```

```
@media only screen and (min-width: 40.063em) and (max-width: 64em) { }
```

```
/* Large screens: min-width 1025px, large screens */
```

```
@media only screen and (min-width: 64.063em) { }
```

```
/* min-width 1025px and max-width 1440px */
```

```
@media only screen and (min-width: 64.063em) and (max-width: 90em) { }
```

```
/* XLarge screens: min-width 1441px, xlarge screens */
```

```
@media only screen and (min-width: 90.063em) { }
```

```
/* min-width 1441px and max-width 1920px */
```

```
@media only screen and (min-width: 90.063em) and (max-width: 120em) { }
```

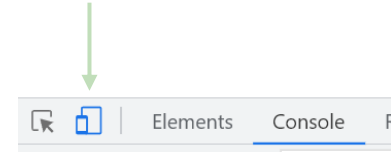
```
/* XXL large screens: min-width 1921px, xxlarge screens */
```

```
@media only screen and (min-width: 120.063em) { }
```

# Breakpoints – Umbruch, aber wie?



1. Webseite z.B. mit Developer Toolbar inspizieren
2. Z.B. bei Google Chrome die Responsive Ansicht wählen
3. Seitengröße verändern, von klein (Mobil) bis groß (Desktop)
4. Beim Verändern der Größe auf Probleme achten (Welche?)



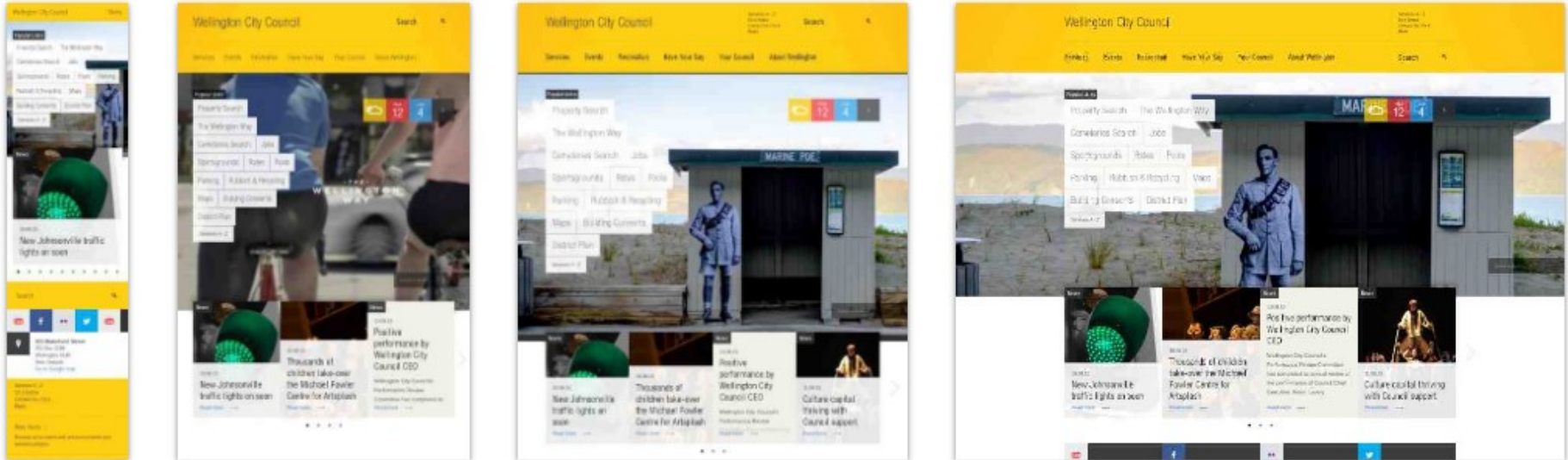
# Breakpoints – Umbruch, aber wie?



Auf welche Veränderungen achten

- Entstehen **Fehler** im Design?
- Sind alle **wichtigen Element** vorhanden? Müssen **zusätzliche** Elemente hinzugefügt werden oder **Funktionalitäten verbessert** werden?
- Müssen Element in **ihrer Form verändert** werden. z.B.:
  - Von: Burgermenü => Zu: Menü immer sichtbar
  - Von: Slider => Zu: Cards
- Stimmen die **Bildgrößen/-auflösungen** bzw. sind Bilder **unscharf**?
- Stimmen die **Abstände** und **Größenverhältnisse**?
- Hat Text in den Zeilen die **passende** Länge?  
(Daumenregel: **45-90 Zeichen pro Zeile**)

# Breakpoints - Beispiel



# Pixel ist nicht gleich Pixel



- **Hardware-Pixel (physische Pixel)** sind Pixel, die tatsächlich dem des Gerätes entsprechen.
- **Device Independent Pixel (DIP)** sind eine Maßeinheit, die Pixel mit einem realen Abstand in Beziehung setzt und physische Pixel zusammenfasst.
- **CSS-Pixel (Software-Pixel)** sind eine Maßeinheit, die zur Positionierung von Elementen auf dem Bildschirm verwendet werden. Es wird der **Viewport** als Referenz verwendet

Für das Design relevant, weil die Pixel in Bezug auf den Viewport gesetzt werden. Was ist mit Geräten mit hoher Pixeldichte?

## Device Pixel Ratio (DPR)



Das DPR ist das Verhältnis zwischen dem DIP und den tatsächlichen physischen Pixel auf dem Bildschirm.

$$DPR = \text{Hardware-Pixel} / DIP$$

$$\textit{iPhone 12 Pro DPR} = 2532 / 844 = \textbf{DPR-3}$$

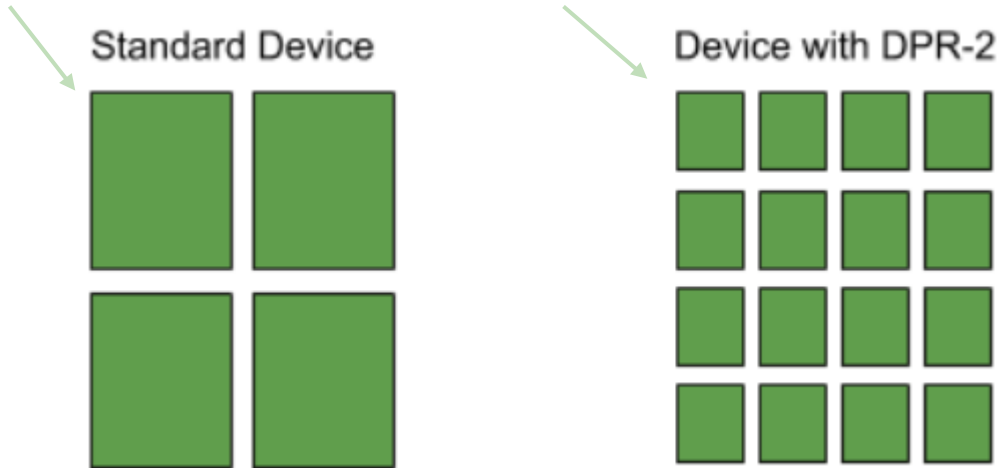
$$\textit{Samsung Galaxy S20 Ultra} = 3200 / 915 \sim \textbf{DPR-3,5}$$

$$\textit{Surface Pro 7} = 2736 / 1368 = \textbf{DPR-2}$$

# Beispiel DPR-2



- 1x1 Pixel wird mit DPR-2 auf 2x2 = 4px gerendert
- 2x2 Pixel werden auf 4x4 = 16px gezeichnet





# Viewport festlegen

- Der sichtbare Bereich der Useransicht
  - ⇒ Viewport (Browser Fenster Inhalt)
- Browser skaliert „runter“ um eine Standard Webseite einzupassen
  - ⇒ Unterschiedlich für verschiedene Geräte
  - ⇒ Media Queries alleine sind **nicht ausreichend!**



Quelle: [https://www.w3schools.com/css/css\\_rwd\\_viewport.asp](https://www.w3schools.com/css/css_rwd_viewport.asp)

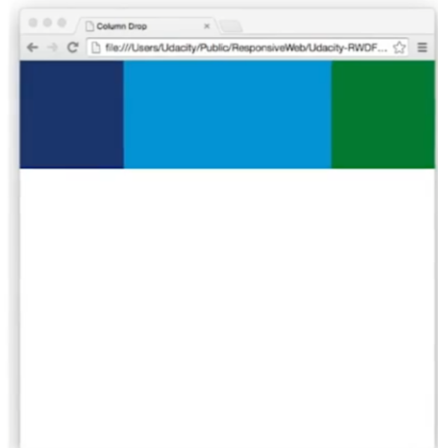
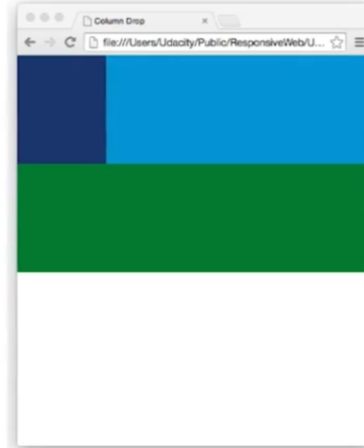
```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

# Design Strategien und Patterns



- Responsive Patterns
  - Column Drop
  - Mostly fluid
  - Layout Shifter
  - Off Canvas
- Mobile First
- Card Layout
- Konkrete Beispiel Layouts (bottom-up)

# Responsive Patterns – Column Drop

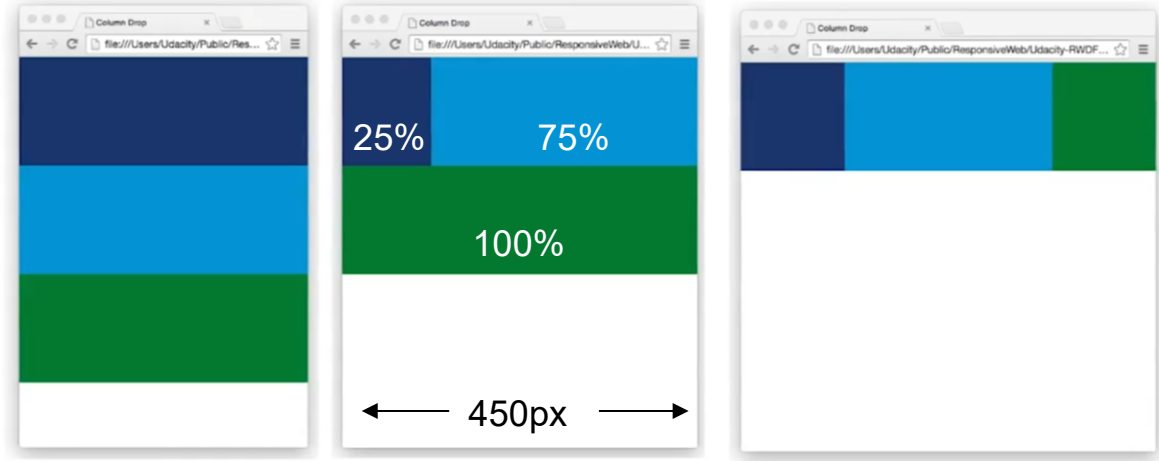


```
<div class="container">  
  <div class="box dark_blue"></div>  
  <div class="box light_blue"></div>  
  <div class="box green"></div>  
</div>
```

```
.container {  
  display: flex;  
  flex-wrap: wrap;  
}
```

```
.box {  
  width: 100%;  
  height: 200px;  
}
```

# Responsive Patterns – Column Drop



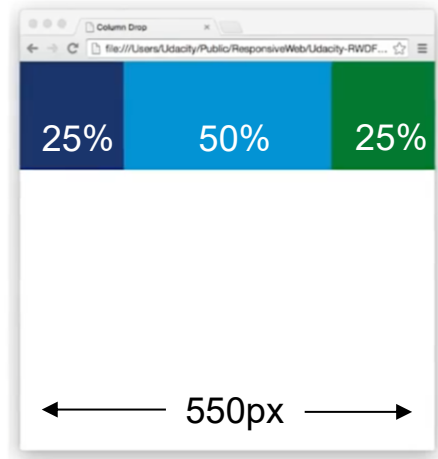
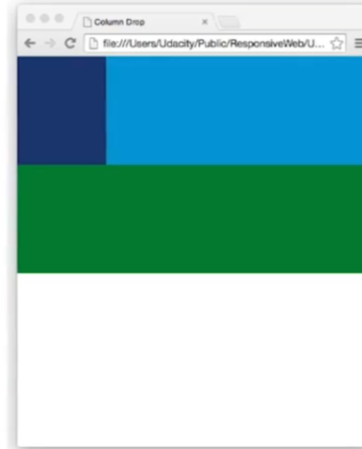
```
<div class="container">
  <div class="box dark_blue"></div>
  <div class="box light_blue"></div>
  <div class="box green"></div>
</div>
```

```
.container {
  display: flex;
  flex-wrap: wrap;
}
```

```
.box {
  width: 100%;
  height: 200px;
}
```

```
@media screen and (min-width: 450px) {
  .dark_blue {
    width: 25%;
  }
  .light_blue {
    width: 75%;
  }
}
```

# Responsive Patterns – Column Drop



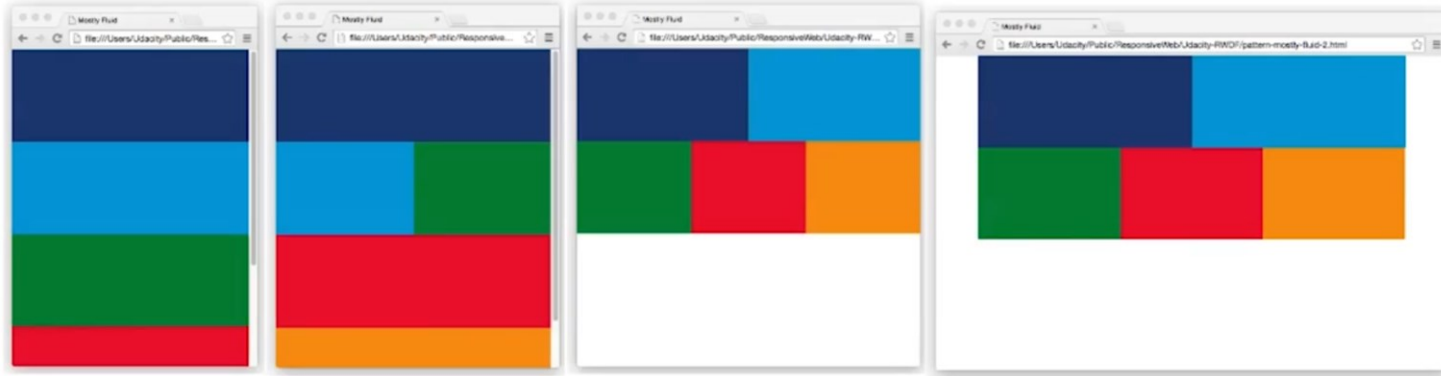
```
<div class="container">
  <div class="box dark_blue"></div>
  <div class="box light_blue"></div>
  <div class="box green"></div>
</div>
```

```
.container {
  display: flex;
  flex-wrap: wrap;
}
```

```
.box {
  width: 100%;
  height: 200px;
}
```

```
@media screen and (min-width: 550px) {
  .dark_blue, .green {
    width: 25%;
  }
  .light_blue {
    width: 50%;
  }
}
```

# Responsive Patterns – Mostly Fluid

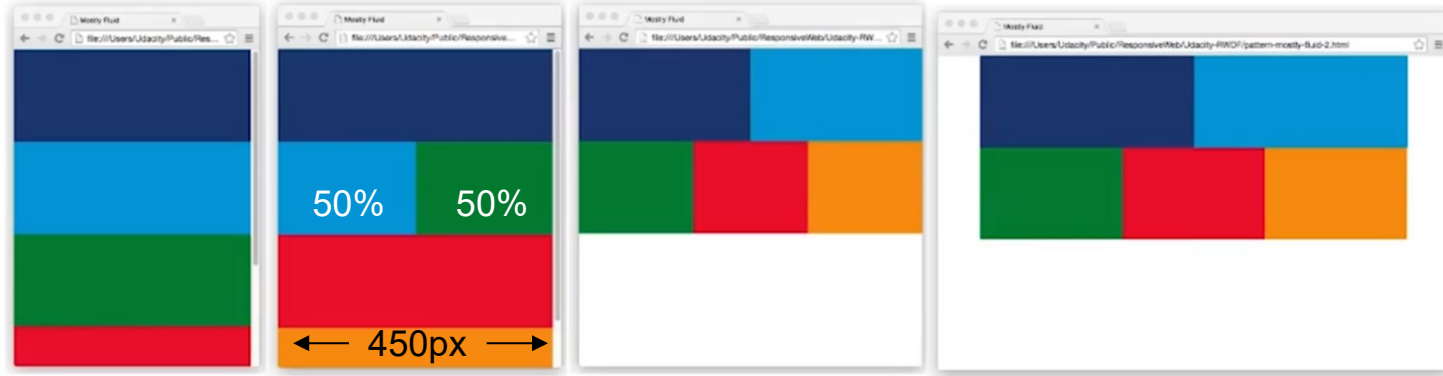


```
<div class="container">
  <div class="box dark_blue"></div>
  <div class="box light_blue"></div>
  <div class="box green"></div>
  <div class="box red"></div>
  <div class="box orange"></div>
</div>
```

```
.container {
  display: flex;
  flex-wrap: wrap;
}

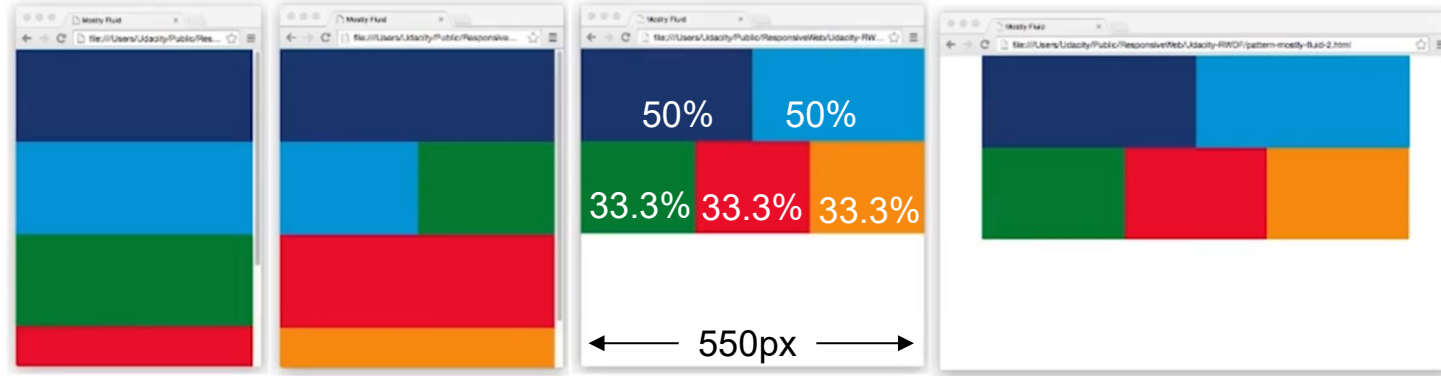
.box {
  width: 100%;
  height: 200px;
}
```

# Responsive Patterns – Mostly Fluid



```
@media screen and (min-width: 450px) {  
  .light_blue, .green {  
    width: 50%;  
  }  
}
```

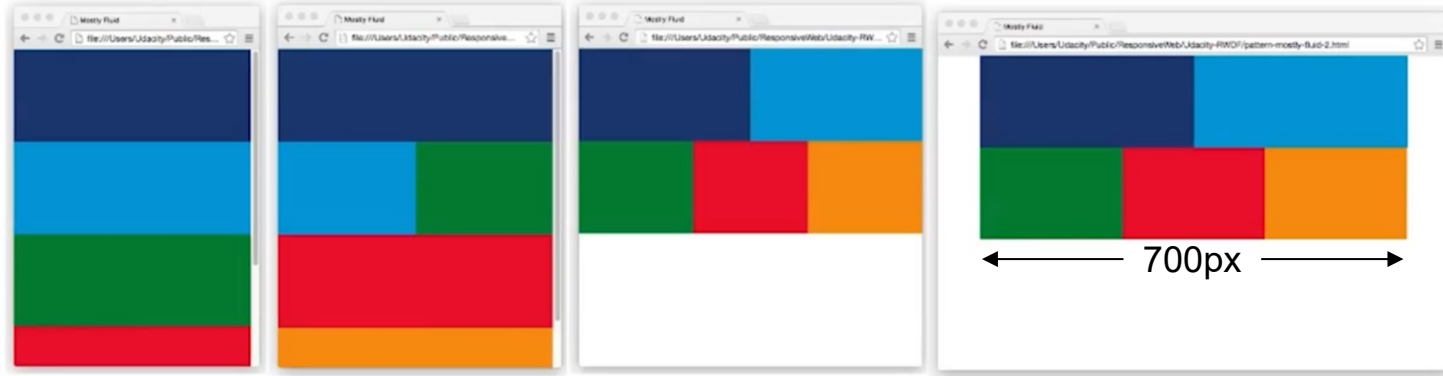
# Responsive Patterns – Mostly Fluid



```
@media screen and (min-width: 550px) {  
  .dark_blue, .light_blue {  
    width: 50%;  
  }  
  .green, .red, .orange {  
    width: 33.333333%;  
  }  
}
```

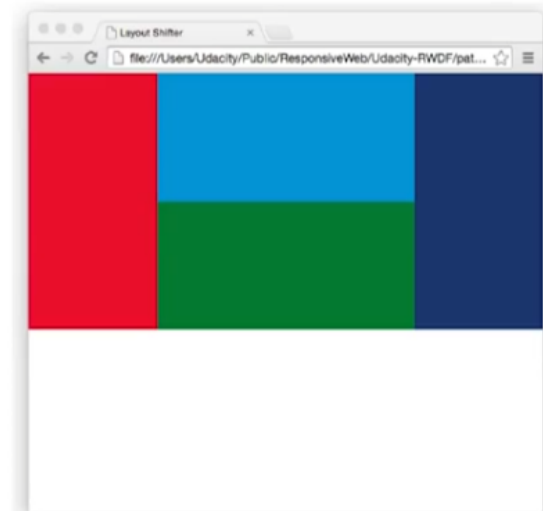
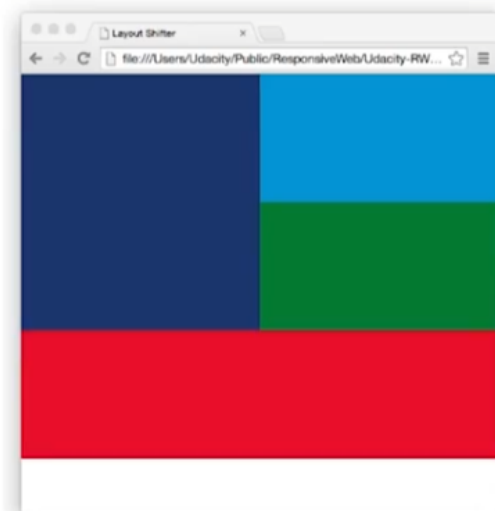


# Responsive Patterns – Mostly Fluid



```
@media screen and (min-width: 700px) {  
  .container {  
    width: 700px;  
    margin-left: auto;  
    margin-right: auto;  
  }  
}
```

# Responsive Design – Layout Shifter



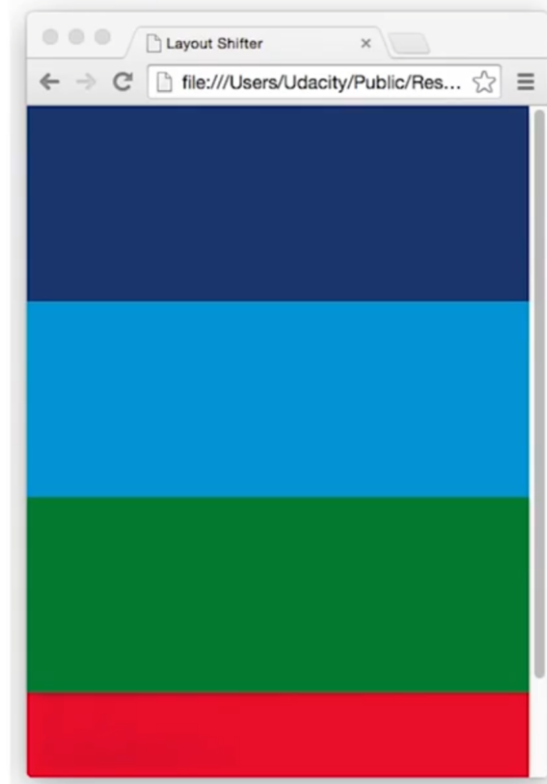
# Responsive Design – Layout Shifter



```
<div class="container">
  <div class="box dark_blue"></div>
  <div class="container" id="container2">
    <div class="box light_blue"></div>
    <div class="box green"></div>
  </div>
  <div class="box red"></div>
</div>
```

```
.container {
  width: 100%;
  display: flex;
  flex-wrap: wrap;
}
```

```
.box {
  width: 100%;
}
```



# Responsive Design – Layout Shifter



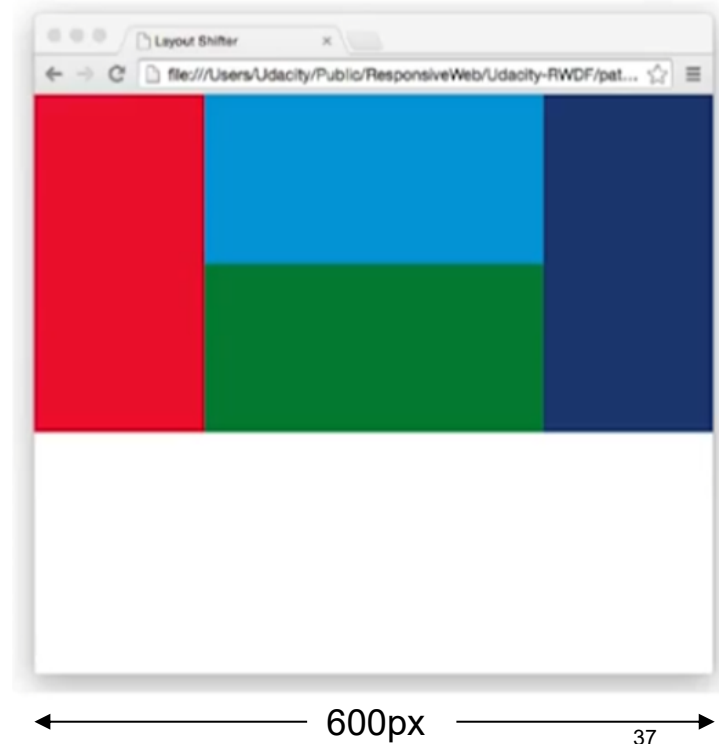
```
@media screen and (min-width: 500px) {  
    .dark_blue {  
        width: 50%;  
    }  
  
    #container2 {  
        width: 50%;  
    }  
}
```



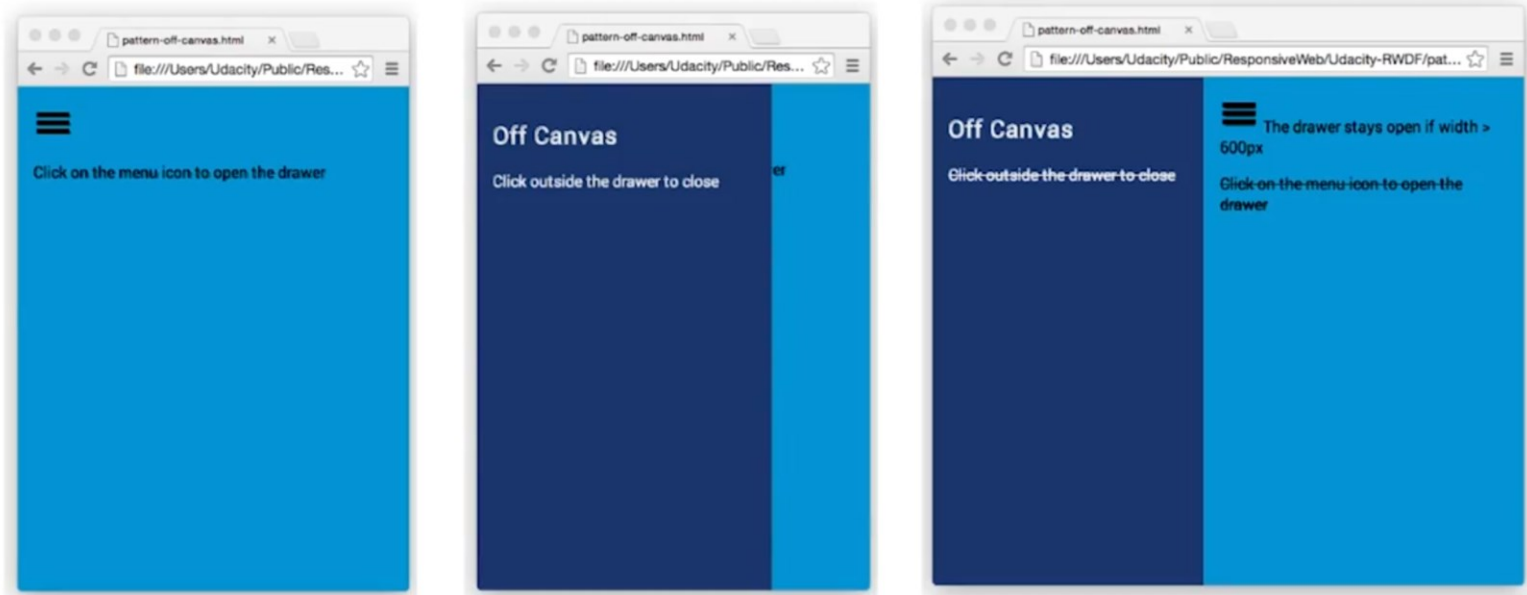
# Responsive Design – Layout Shifter



```
@media screen and (min-width: 600px) {  
  .red {  
    width: 25%;  
    order: -1;  
  }  
  
  .dark_blue {  
    width: 25%;  
    order: 1;  
  }  
}
```



# Responsive Design – Off Canvas



Hilfestellung: <https://codepen.io/chriscoyier/pen/DGgGGO>

# Mobile First Strategie durchgängiges Layout generieren



1. Man baut das Design von Grund auf und startet mit der am meisten eingeschränkten Ansicht – den mobilen Geräten bzw. dem kleinsten Viewport.
  2. Es ist notwendig den Inhalt auf das Wichtigste zu reduzieren.
  3. Je mehr Relevanz ein Inhalt hat, umso weiter oben in der Seite sollte dieser stehen.
  4. Oft sind „nur“ mehr Blöcke untereinander angeordnet.
  5. Wenn nun der kleinste Viewport korrekt designend ist, dann kommt der nächst größere dran – also nach Mobiler Ansicht, z.B. eine Tablet Ansicht, usw.
  6. Danach weiter bis man bei dem größten Viewport angekommen ist, also z.B. XXLarge Desktop
- ⇒ nutzen “cascading” Eigenschaft von CSS

# Beispiele für RWD



- Bilder
  - Größe steuern
  - als Hintergrund
  - mit einfachem Textumbruch
- Card Design
- Menu
- Gesamt Layouts und Spaltendesign

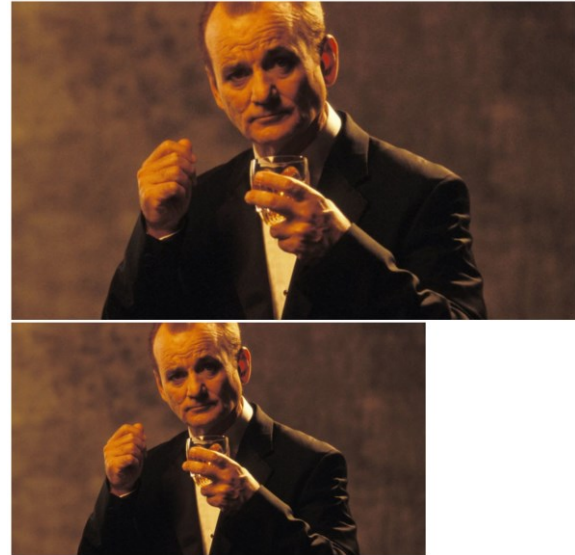


# Responsive Images



```
img {  
  width: 100%;  
  height: auto;  
}
```

```
img {  
  max-width: 100%;  
  height: auto;  
}
```



# Responsive Images - Hintergrund



```
div {  
  height: 400px;  
  background-image: url('img_flowers.jpg');  
  background-repeat: no-repeat;  
  border: 1px solid red;  
  background-size: ...;  
}
```

`background-size: contain;`



`background-size: 100% 100%;`



`background-size: cover;`



# Artikel mit Bild



```
img {  
  float: left;  
  width: 150px;  
  display: inline-block;  
  margin-right: 10px;  
  margin-bottom: 5px;  
}
```

```
<article>  
  <h2>The Planet</h2>  
    
  <p>Lorem ipsum ...</p>  
  <p>Lorem ipsum ...</p>  
</article>
```

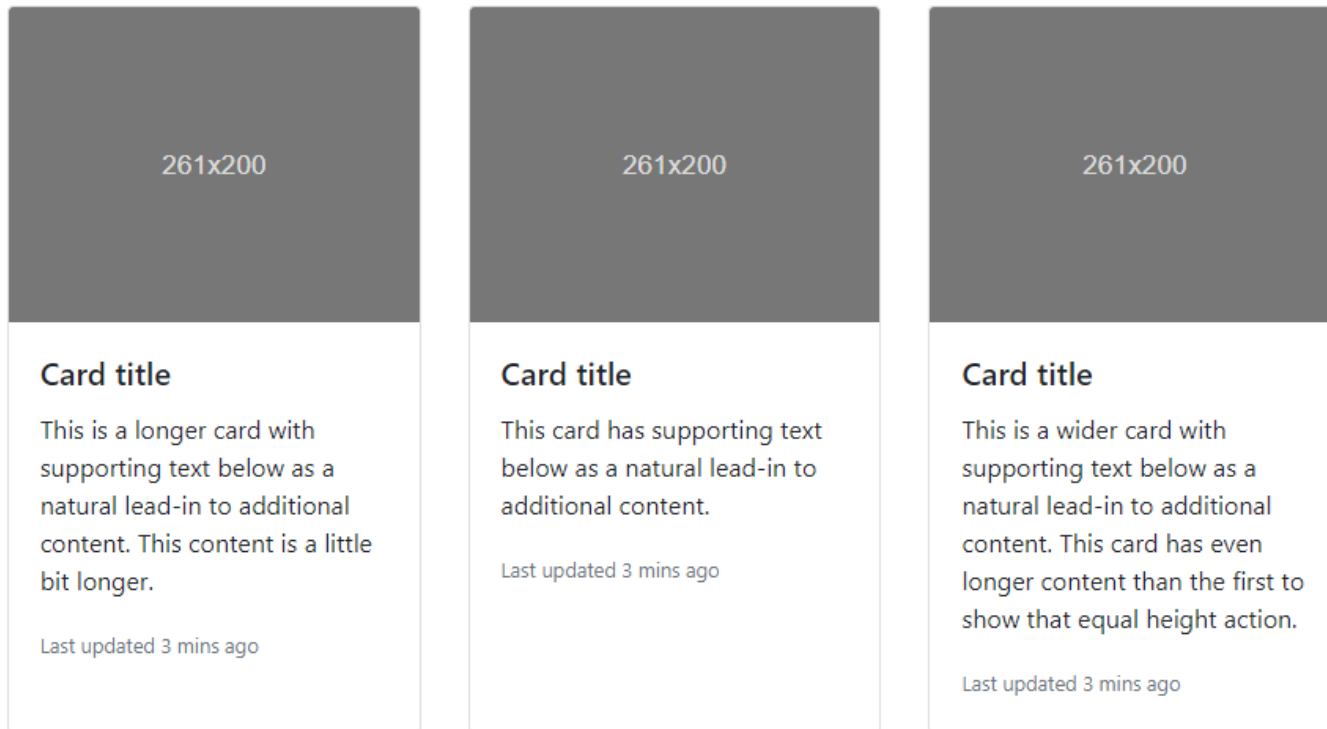
## The Planet



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

# Card Design



# Cards



## Wie sehen Cards aus

- Hervorgehobene Bilder oder Medieninhalt
- Titel
- Autor
- Zusammenfassung
- Datum
- Kategorie
- Social Share Links
- „Read More“ Button

## Was sind Cards

- Für sich abgeschlossen
- unabhängig
- individuell

# Cards Vorteile



- Ist „automatisch“ responsive
- Simpel und übersichtlich
- Leicht zu organisieren – man kann sich auf Details fokussieren ohne das Gesamtlayout zu beeinflussen
- Einfach zu lesen
- Gut geeignet für Medien-Inhalte (Bilder, Videos)
- Kann je nach Darstellungsposition bzw. –ort einfach „resized“ werden



# Wann sollten Cards nicht eingesetzt werden

- als reiner Layout-Mechanismus
- für sehr bereite Layouts
- „zu viel“ Inhalt
- eher weniger für medienarme Inhalte
- für Elemente die viele „Desktop“-Features beinhalten

# Cards Beispiel

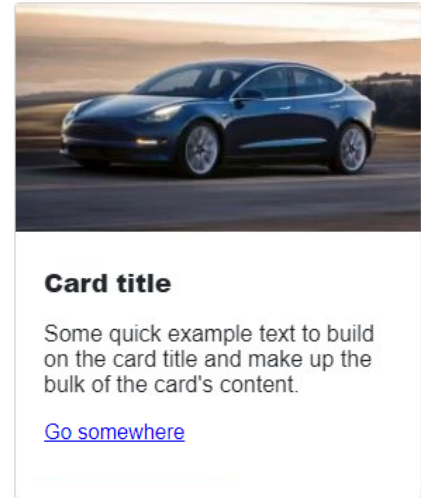
```
<div class="card">
  
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">Some ...</p>
    <a href="#">Go ...</a>
  </div>
</div>
```

```
.card {
  border: 1px solid lightgray;
  margin-top: 10px;
  margin-right: 10px;
  border-radius: 0.25rem;
  font-family: Arial;
  width: 18rem;
}

.card img {
  width: 100%;
  display: block;
}

.card h5 {
  font-weight: 800;
  font-size: 1.1rem;
  margin: 0;
}

.card-body {
  color: #212529;
  text-align: left;
  padding: 20px;
}
```





# Menü



Lorem Nobis Nostrum Quia Magni

```
<nav class="navigation">
  <ul class="menu">
    <li><a href="#">Lorem</a></li>
    <li><a href="#">Nobis</a></li>
    <li><a href="#">Nostrum</a></li>
    <li><a href="#">Quia</a></li>
    <li><a href="#">Magni</a></li>
  </ul><!-- .menu -->
</nav><!-- .navigation -->
```

```
.navigation {
  display: inline-block;
}

/* Listeneigenschaften ändern */
.menu,
.sub-menu {
  margin: 0;
  padding: 0;
}
.navigation .menu {
  list-style: none;
  display: flex;
}
```

# Menü

Lorem Nobis Nostrum Quia Magni

```
<nav class="navigation">
  <ul class="menu">
    <li><a href="#">Lorem</a></li>
    <li><a href="#">Nobis</a></li>
    <li><a href="#">Nostrum</a></li>
    <li><a href="#">Quia</a></li>
    <li><a href="#">Magni</a></li>
  </ul><!-- .menu -->
</nav><!-- .navigation -->
```

```
/* Menueinträge umformen */
.navigation ul.menu li {
  margin: 0;
  background-color: #ddd;
}
.navigation ul.menu a
{
  /*
   * einem Link wird die Farbe
   * nicht von Haus aus vererbt
   */
  color: inherit;
  text-decoration: none;
  display: inline-block;
  padding: 0.6rem;
}
.navigation ul.menu li:hover {
  color: #ddd;
  background-color: #222;
}
```



# Menü



```
<nav class="navigation">
  <ul class="menu vertical">
    <li><a href="#">Lorem</a></li>
    <li><a href="#">Nobis</a></li>
    <li><a href="#">Nostrum</a></li>
    <li><a href="#">Quia</a></li>
    <li><a href="#">Magni</a></li>
  </ul><!-- .menu -->
</nav><!-- .navigation -->
```

```
.navigation .menu.vertical {
  width: 5rem;
  display: inline-block;
}
```

Lorem  
Nobis  
Nostrum  
Quia  
Magni

# Burger – Menü

```
<nav>
  <div class="burger">
    <a>=Menu=</a>
  </div>
  <ul class="menu">
    <li><a href="#home">Home</a></li>
    <li><a href="#news">News</a></li>
    <li><a href="#contact">Contact</a></li>
    <li><a href="#about">About</a></li>
  </ul>
</nav>
```

=Menu=

[Home](#) [News](#) [Contact](#) [About](#)

[Home](#)

[News](#)

[Contact](#)

[About](#)

```
ul.menu {
  list-style: none;
  margin: 0;
  padding: 0;
}
```

```
@media screen and (max-width:583px) {
  .burger {
    user-select: none;
  }
  ul.menu {
    display: none;
  }
  nav > .menu:hover,
  nav > .burger:active ~ .menu,
  nav > .burger:hover ~ .menu {
    display: block;
  }
}
```



```
@media screen and (min-width:584px) {
  nav > .burger {
    display: none;
  }

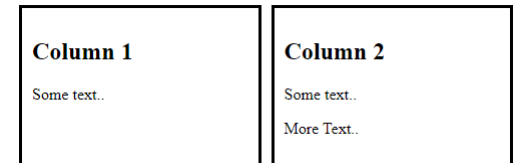
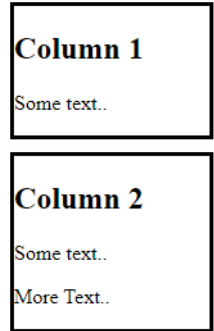
  nav > .menu {
    display: flex;
    justify-content: space-around;
  }
}
```

# Spalten/Column Layout mit Flexbox



```
<div class="row">
  <div class="column">
    <h2>Column 1</h2>
    <p>Some text..</p>
  </div>
  <div class="column">
    <h2>Column 2</h2>
    <p>Some text..</p>
    <p>More text..</p>
  </div>
</div>
```

```
* {
  box-sizing: border-box;
}
.column {
  border-style: solid;
  flex: 100%;
}
.row {
  display: flex;
  flex-wrap: wrap;
  gap: 10px;
}
@media screen and (min-width: 480px) {
  .column {
    flex: none;
    flex-grow: 1;
  }
}
```



# Grid Layout using CSS Grid



```
<div class="container">
  <header>Header</header>
  <nav>Navigation</nav>
  <main>Main area</main>
  <footer>Footer</footer>
</div>
```

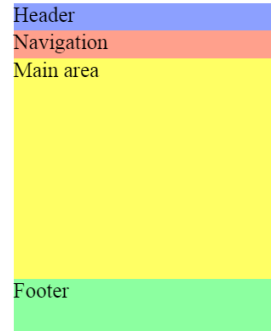
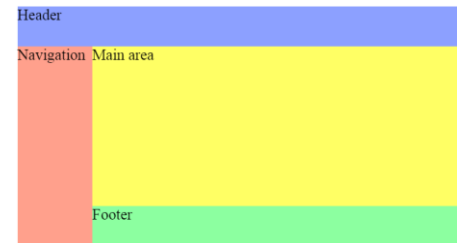
```
.container {
  display: grid;
  grid-template-areas:
    "head"
    "nav"
    "main"
    "foot";

  grid-template-rows: 1fr 1fr 8fr 1fr;
  grid-template-columns: 1fr;
}
```

```
.container > header {
  grid-area: head;
  background-color: #8ca0ff;
}
.container > nav {
  grid-area: nav;
  background-color: #ffa08c;
}
.container > main {
  grid-area: main;
  background-color: #ffff64;
}
.container > footer {
  grid-area: foot;
  background-color: #8cffa0;
}
```

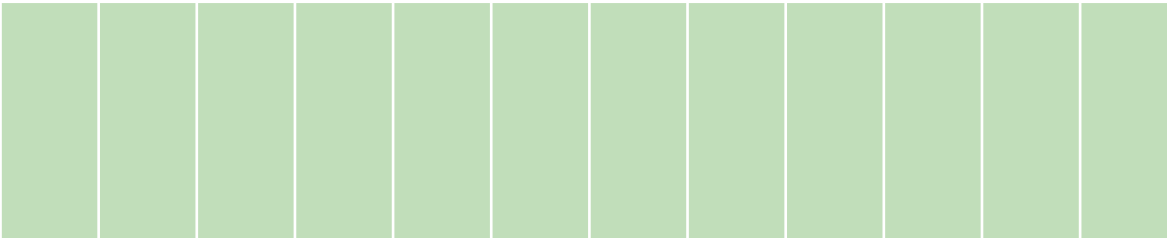
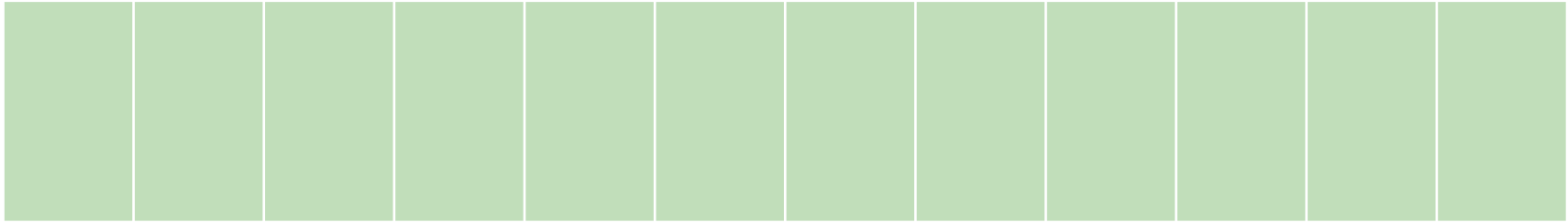
```
@media screen and (min-width: 30.06em) {
  .container {
    grid-template-areas:
      "head head"
      "head head"
      "nav main"
      "nav foot";

    grid-template-columns: 1fr 5fr;
    grid-template-rows: 1fr 1fr 10fr 1fr;
  }
}
```

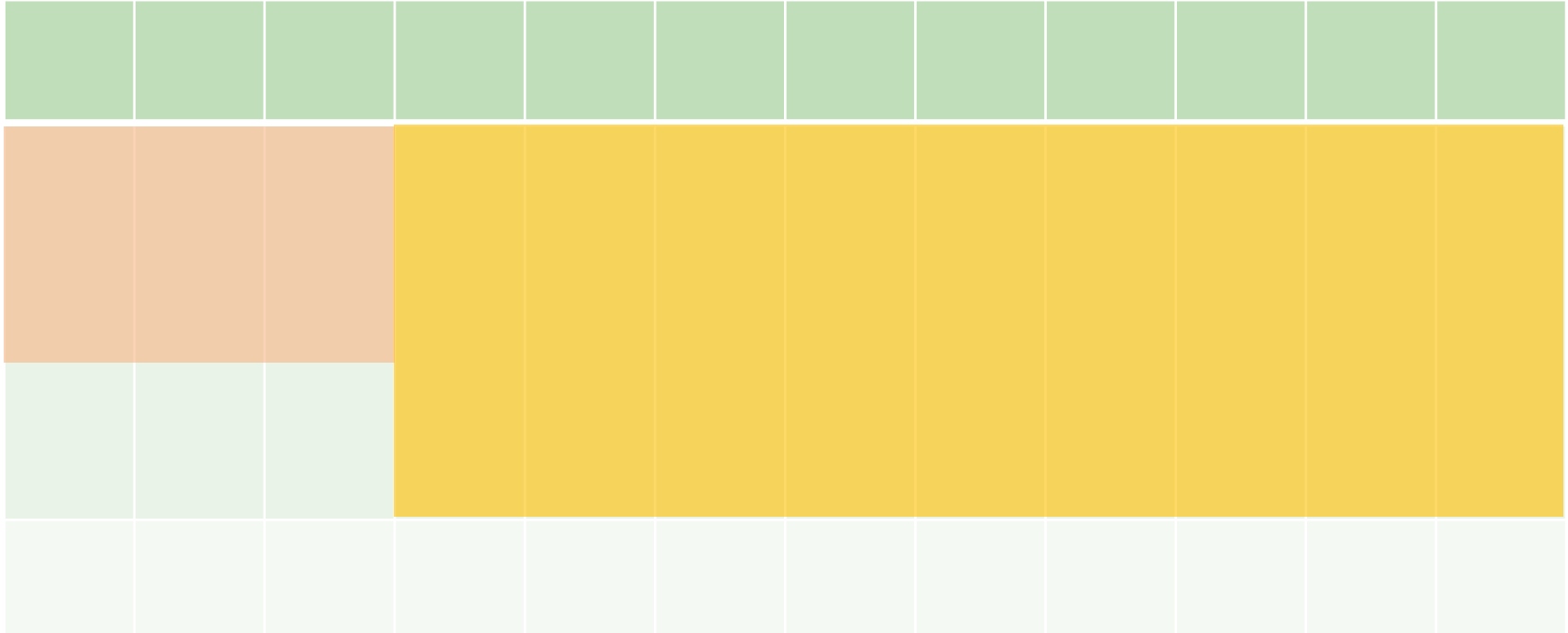


Hilfestellung: <https://developer.mozilla.org/en-US/docs/Web/CSS/grid-template>

# Spalten/Column Layout – 12 Spalten



# Spalten/Column Layout – Teiler von 12





# CSS Frameworks



CSS ist einfach aufgebaut und dadurch auch nutzbar ohne tiefergehende Programmierkenntnisse

→ dadurch weniger mächtig als „echte“ Programmiersprachen

- DRY-Prinzip (*Don't repeat yourself* – wiederhole dich nicht) kann nicht umgesetzt werden, da CSS verschiedene Features fehlen, die die meisten Programmiersprachen bieten
- So müssen z.B. gleiche Eigenschaften für verschiedene Selektoren stets wiederholt werden
  - CSS-Dateien werden groß und unübersichtlich

Lösung: **CSS-Frameworks** und **CSS-Präprozessoren**

# CSS Frameworks



## ■ Module

- Grid → Einfachere Umsetzung von Responsive Designs
- Icons und Fonts
- Formulare, Tooltips, Buttons
- UI-Elemente: Accordions, Tabs, etc.

## ■ Nutzen oft CSS-Präprozessoren

## ■ Beispiele:

- Bootstrap
- Pure
- Foundation
- ...

Semantic UI	e 9' G + O 6' u + 12' +	LESS SASS	☑ ☑ ☑	MIT
Bootstrap	e 8' G + O + u + 0 +	LESS SASS	☑ ☑ ☑	MIT
Foundation	e 9' G + O + u + 0 +	LESS SASS	☑ ☑ ☑	MIT
UIKit	e 9' G + O + u + 0 +	LESS SASS	☑ ☑ ☑	MIT
960 Grid System	e 7' G + O + u + 0 +	LESS SASS	☑ ☑ ☑	GPL & MIT
Skeleton	e 7' G + O + u 3' 0 +	LESS SASS	☑ ☑ ☑	MIT
99lime HTML KickStart	e 8' G + O + u + 0 +	LESS SASS	☑ ☑ ☑	MIT
Kube	e 8' G + O + u + 0 +	LESS SASS	☑ ☑ ☑	Open Source
Less Framework	e 7' G + O + u + 0 +	LESS SASS	☑ ☑ ☑	MIT
Flaminwork	e 7' G + O + u + 0 +	LESS SASS	☑ ☑ ☑	Open Source
G5 Framework	e 7' G + O + u + 0 +	LESS SASS	☑ ☑ ☑	Open Source
Easy Framework	e 7' G + O + u + 0 +	LESS SASS	☑ ☑ ☑	CC
Blueprint	e 7' G + O + u + 0 +	LESS SASS	☑ ☑ ☑	MIT

<http://usablica.github.io/front-end-frameworks/compare.html?v=2.0>

# Bootstrap

Entwickelt von Twitter

- seit 2011 Open Source (MIT-Lizenz)
- sehr populär und weit verbreitet
- CSS und JavaScript
  - in Kombination mit jQuery
- Module
  - Responsive Grids
  - Icon Library (Glyphicons)
  - <http://getbootstrap.com/components/>
- Support für nahezu alle modernen Browser
- Bootlint zur HTML-Prüfung



## Using the framework



### Starter template

Nothing but the basics: compiled CSS and JavaScript along with a container.



### Bootstrap theme

Load the optional Bootstrap theme for a visually enhanced experience.



### Grids

Multiple examples of grid layouts with all four tiers, nesting, and more.



### Jumbotron

Build around the jumbotron with a navbar and some basic grid columns.



### Narrow jumbotron

Build a more custom page by narrowing the default container and jumbotron.

## Navbars in action



### Navbar

Super basic template that includes the navbar along with some additional content.



### Static top navbar

Super basic template with a static top navbar along with some additional content.



### Fixed navbar

Super basic template with a fixed top navbar along with some additional content.

## Custom components



# Pure CSS



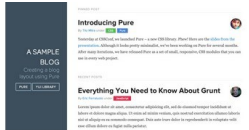
## Open Source (BSD-Lizenz)

- leichtgewichtig (Basis nur 1 Kilobyte)
- reines CSS optimiert für verschiedene Anzeigegrößen (Smartphone, Tablet, PC)

## ■ Responsive Design Module

- Base
- Grids
- Formulare
- Buttons
- Tabellen
- Menüs

## ■ Support für nahezu alle modernen Browser

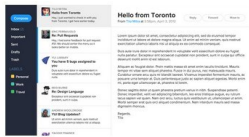


**Blog**

A layout example that shows off a blog page with a list of posts.

Base | Grids | Buttons | Menus

[View](#) [Download](#)

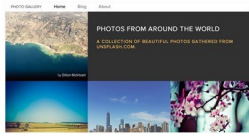


**Email**

A layout example that shows off a responsive email layout.

Base | Grids | Buttons | Menus

[View](#) [Download](#)

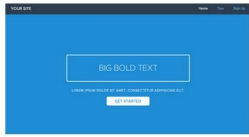


**Photo Gallery**

A layout example that shows off a responsive photo gallery.

Base | Grids | Forms | Buttons | Menus

[View](#) [Download](#)



**Landing Page**

A layout example that shows off a responsive product landing page.

Base | Grids | Buttons | Menus | Forms

[View](#) [Download](#)

# CSS Präprozessoren



CSS-Frameworks nutzen häufig **CSS-Präprozessoren**

- CSS-Präprozessoren
  - übersetzen (transpilieren) eigene Sprach-Syntax zu validem CSS
  - erweitern CSS um **Variablen**, **Funktionen** und **Mixins**
- Bekannte Projekte: **Less** und **Sass**

# Sass



- **Sass: Syntactically Awesome Stylesheets**
- Ursprünglich in Ruby (→ Woche 5) entwickelt
  - Syntax stark an Ruby angelehnt, keine Klammern oder Semikolons nötig, strikte Einrückung erforderlich
- Sass Version 3.0 führte neue SCSS-Syntax ein
  - Syntax neu als „CSS mit Sass-Erweiterungen“ entwickelt
  - Komplette **abwärtskompatibel mit CSS**
    - jede CSS-Datei ist eine gültige SCSS-Datei
- Als **LibSass** nach C/C++ portiert
  - Wrapper für viele verschiedene Programmiersprachen: Java, JavaScript, .Net, PHP, Python, ...
  - Sass kann so nahtlos in verschiedenste Projekte integriert werden

# Sass – Variablen



- **Variablen** ermöglichen zentrale Definition und Wiederverwendung häufig benutzter Werte, z.B.: Farben, Abstände, Rahmenstile, ...
  - ➔ müssen bei Änderung nur an einer Stelle angepasst werden
- **\$-Symbol** definiert Variablen

SCSS:

```
$primary-color: #333;  
  
h1 {  
  color: $primary-color;  
}  
  
a {  
  color: $primary-color;  
  text-decoration: none;  
}
```

Generiertes CSS:

```
body {  
  color: #333;  
}  
  
a {  
  color: #333;  
  text-decoration: none;  
}
```

# Sass – Erweiterungen



- Vererbung von Eigenschaften → bessere Lesbarkeit des Codes

SCSS:

```
.message {  
  border: 1px solid #ccc;  
  padding: 10px;  
  color: #333;  
}  
.success {  
  @extend .message;  
  border-color: green;  
}  
.error {  
  @extend .message;  
  border-color: red;  
}
```

Generiertes CSS:

```
.message, .success, .error {  
  border: 1px solid #ccc;  
  padding: 10px;  
  color: #333;  
}  
.success {  
  border-color: green;  
}  
.error {  
  border-color: red;  
}
```



# Sass – Mixins



Wiederverwendbarkeit von Deklarationen

- Werte können als Parameter an Mixin übergeben werden
- Nützlich z.B. für Hersteller-Präfixe experimenteller CSS-Eigenschaften

SCSS:

```
@mixin border-radius($radius) {  
  -webkit-border-radius: $radius;  
  -moz-border-radius: $radius;  
  -ms-border-radius: $radius;  
  border-radius: $radius;  
}  
div {  
  @include border-radius(10px);  
}  
.footer {  
  @include border-radius(5px);  
  margin-top: 10px;  
}
```

Generiertes CSS:

```
div {  
  -webkit-border-radius: 10px;  
  -moz-border-radius: 10px;  
  -ms-border-radius: 10px;  
  border-radius: 10px;  
}  
.footer {  
  -webkit-border-radius: 5px;  
  -moz-border-radius: 5px;  
  -ms-border-radius: 5px;  
  border-radius: 5px;  
  margin-top: 10px;  
}
```

# Bemerkungen und Zusammenfassung



- CSS-Präprozessoren erlauben deutlich lesbarere und einfacher wartbarere Style-Definitionen
- Insbesondere bei großen Web-Projekten mit dutzenden CSS-Dateien und tausenden Deklarationen macht sich bessere Übersicht und Einsparung von Code deutlicher sichtbar, als in gezeigten kleinen Beispielen
- (Leichter) Nachteil: Transpiler bringen zusätzlichen Aufwand bei Entwicklung und Veröffentlichung von Webseiten mit
  - Aber: Viele Web-Frameworks bieten bereits eingebaute Unterstützung für CSS-Präprozessoren
- Zweiter genannter Vertreter – Less – bietet recht ähnliche Features wie vorgestellte Sass



**FH Salzburg**

# **VO Web-Technologien**

**Einheit 4, Oliver Jung**