Modern CSS Layouting

1. A little bit of the history
2. Box sizing approach
3. CSS flex-box approach
4. CSS grid approach

A little bit of the history

Old layouting methods

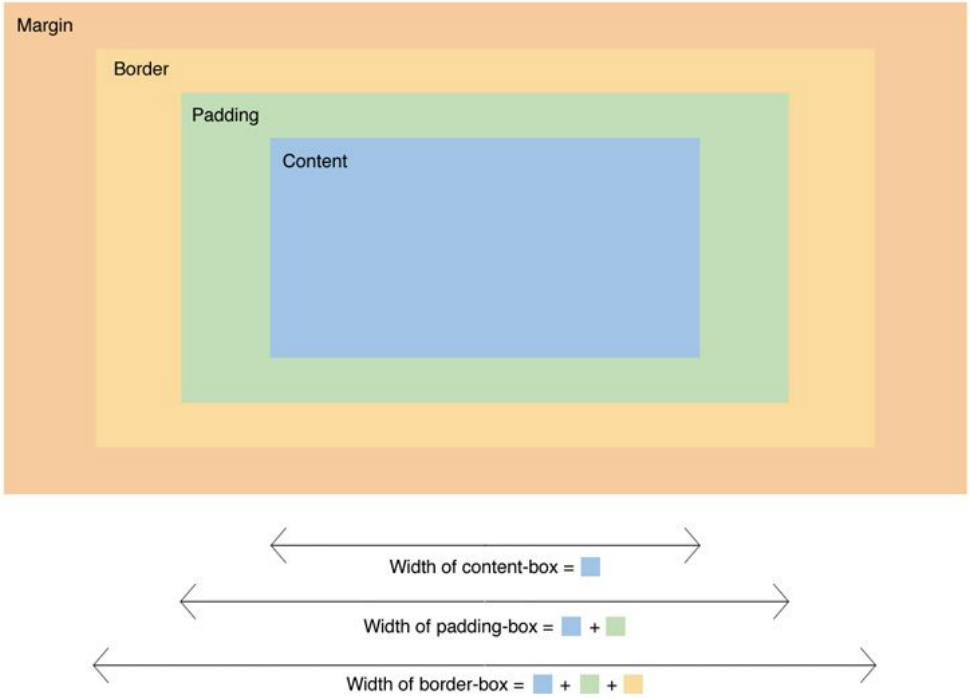
All ‘past’ techniques, like

- floats
- inline-block
- **display: table - the closest to perfection**
- absolute, relative positionning

resulted in MUCH of hussle (time = money).

Box sizing approach

Introducing box-sizing



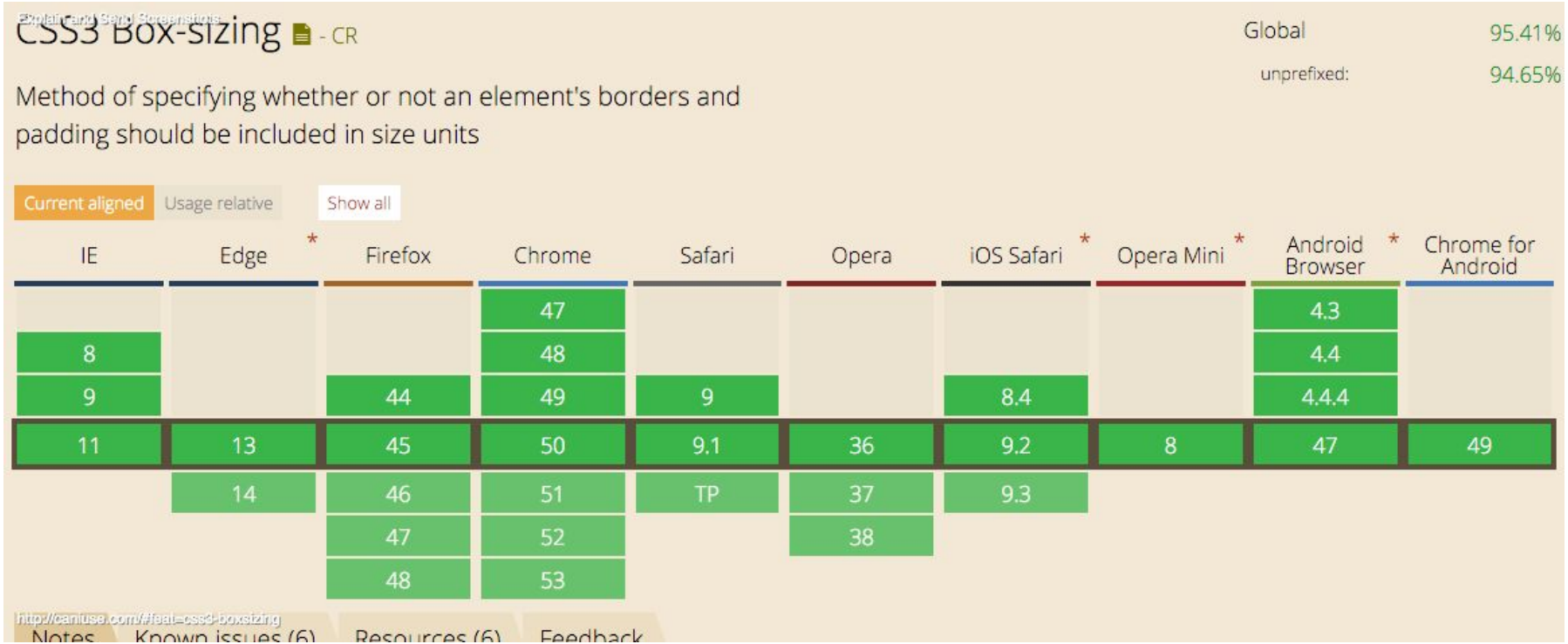
Twitter’s Bootstrap grid

```
<div class="row">
  <div class="col-md-4">.col-md-1</div>
  <div class="col-md-4">.col-md-1</div>
  <div class="col-md-4">.col-md-1</div>
</div>
<div class="row">
  <div class="col-md-8">.col-md-8</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-5">.col-md-4</div>
  <div class="col-md-5">.col-md-4</div>
  <div class="col-md-2">.col-md-4</div>
</div>
```

- non semantic code
- messy markup code
- markup overhead
- no cell’s corelation

We can use SASS @extend to make it more semantic - but still it’s not what we’re looking for

Box-sizing support - 95.41%

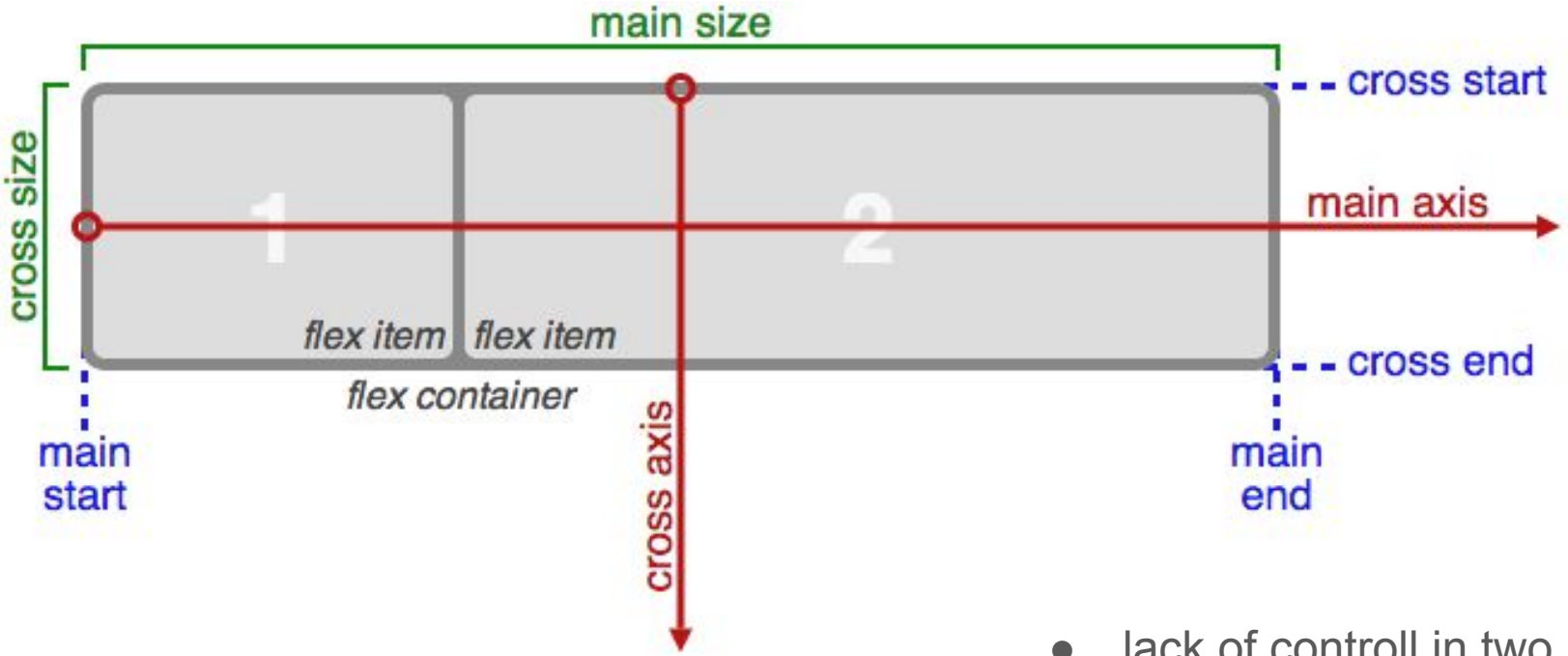


CSS flex-box approach

Flexbox abstracts

- Flexbox container - container for all flexbox items
- Flexbox items - all flexing items within the container (only first children apply)
- Main axis - axis with which flexing items will be placed along (it can be horizontal or vertical - along with the css flex-direction property)
- Main start, Main end - starting and ending point of the main axis
- Main size - flex items size in line with the main axis
- Cross axis - perpendicular axis to the main axis (opposite to the main axis direction)
- Cross start, Cross end - starting and ending point of the cross axis
- Cross size - flex items size in line with the cross axis

Flexbox abstracts



- lack of controll in two dimensions

Flexbox layout example - HTML

```
<div class "wrapper">
 <header class "header">Header</header>
 <article class "main">
   <p>Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo </p>
 </article>
 <aside class "aside aside-1">Aside  </aside>
 <aside class "aside aside-2">Aside  </aside>
 <footer class "footer">Footer</footer>
</div>
```

Flexbox layout example - CSS

```
.wrapper {
  display: flex;
  flex-flow: row wrap;
}

/* We tell all items to be 100% width */
.header, .main, .nav, .aside, .footer {
  flex: 1 100%;
}

/* Medium screens */
@media all and (min-width: 600px) {
  /* We tell both sidebars to share a row */
  .aside { flex: 1 auto; }
}
```

```
/* Large screens */
@media all and (min-width: 800px) {
  /* We invert order of first sidebar and main
   * And tell the main element to take twice as much width
as the other two sidebars
   */
  .main { flex: 2 0px; }

  .aside-1 { order: 1; }
  .main    { order: 2; }
  .aside-2 { order: 3; }
  .footer  { order: 4; }
}
```

Flexbox example - result

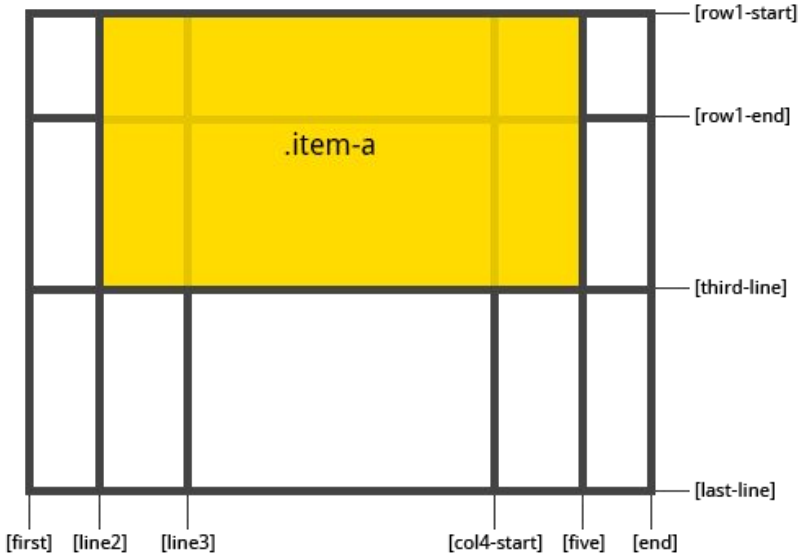
Flexbox support - 94.1%

CSS grid approach

Grid abstracts

- Grid container - container for all the grid items (has to have css display: grid property set)
- Grid item - grid cells (only first children apply)
- Grid line - horizontal or vertical line separating (it occur between both, columns and rows)
- Grid track - horizontal or vertical line built of cells and limited by two adjacent grid lines
- Grid cell - a single grid unit
- Grid area - area built of grid cells (separeted by two horizontal and two vertical grid lines)

Grid abstracts




```
.container{
  grid-template-columns: 40px 50px auto 50px 40px;
  grid-template-rows: 25% 100px auto;
}
```

```
.item-a{
  grid-column-start: 2;
  grid-column-end: five;
  grid-row-start: row1-start
  grid-row-end: 3
}
```

Grid layout example - HTML

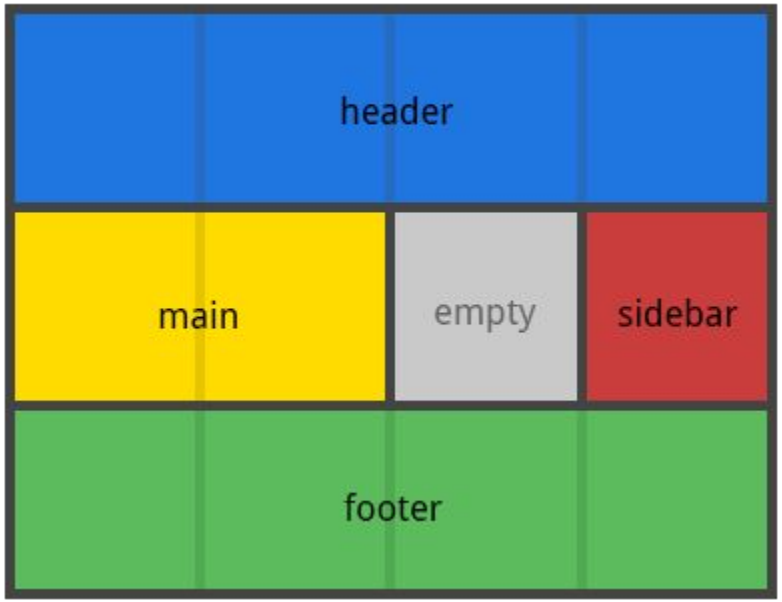
```
<div class "container">
 <header class "item-a">Header</header>
 <main class "item-b">Main</main>
 <aside class "item-c">Sidebar</aside>
 <footer class "item-d">Footer</footer>
</div>
```

Grid layout example - CSS

```css
.item-a{
  grid-area: header;
}
.item-b{
  grid-area: main;
}
.item-c{
  grid-area: sidebar;
}
.item-d{
  grid-area: footer;
}
```

```css
.container{
  grid-template-columns: 50px 50px 50px 50px;
  grid-template-rows: auto;
  grid-template-areas: "header header header header"
                       "main main . sidebar"
                       "footer footer footer footer"
}
```

Grid layout example - result

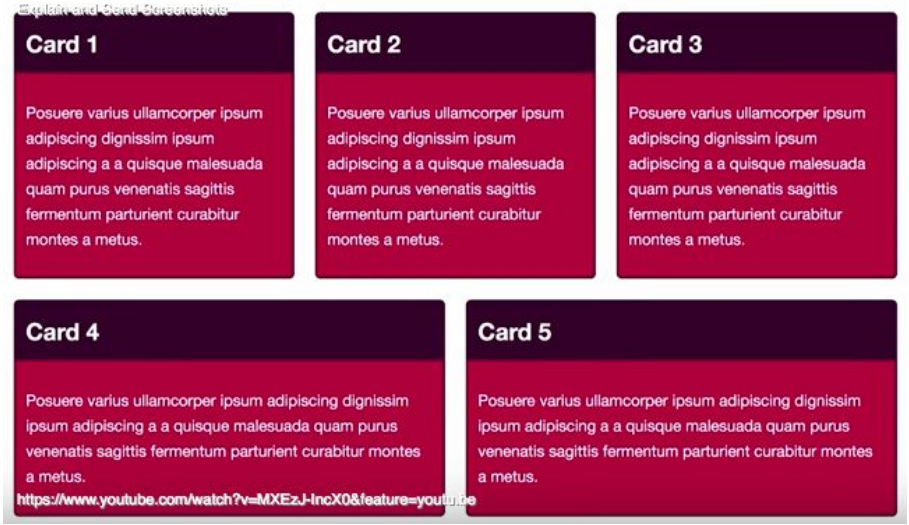


CSS grid support - 8.1%

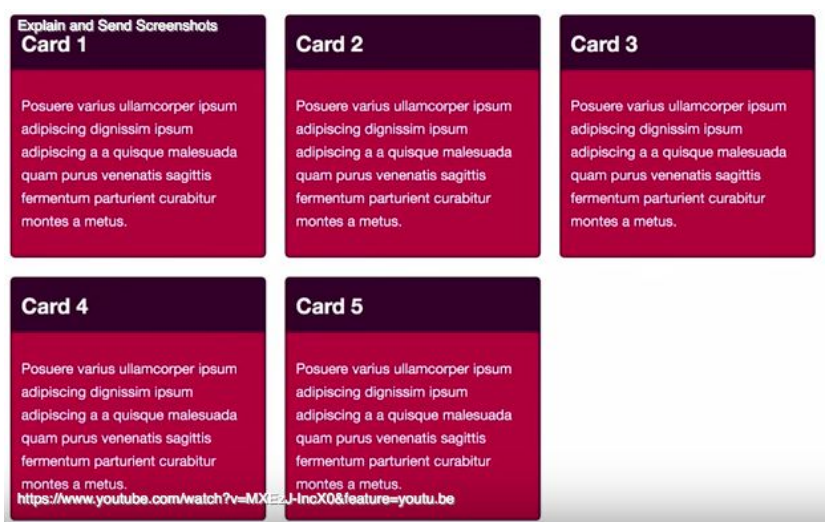
Bottom line

CSS grid vs Flexbox

Flexbox, Grid



CSS Grid ONLY - **winner**



CSS grid vs box-sizing

CSS Grid - **winner**

```
<div class "container">
 <header class "item-a">Header</header>
 <main class "item-b">Main</main>
 <aside class "item-c">Sidebar</aside>
 <footer class "item-d">Footer</footer>
</div>
```

box-sizing

```
<div class="row">
  <div class="col-md-12">header</div>
</div>
<div class="row">
  <div class="col-md-6">.col-md-6</div>
  <div class="col-md-3"></div>
  <div class="col-md-3">sidebar</div>
</div>
<div class="row">
  <div class="col-md-12">footer</div>
</div>
```

Conclusion

It’s worth to **wait** a little while **for the CSS grids**, as they should be shipped with the new versions of the browsers really soon. The flex-box compatibility coverage might be tempting, but as we’ve highlighted, it’s purpose is not to be used for the layouting. I’d stick with the classic **Bootstrap’s grid** technique **in the meantime**, while waiting for the silver bullet to come.

Thank you :)