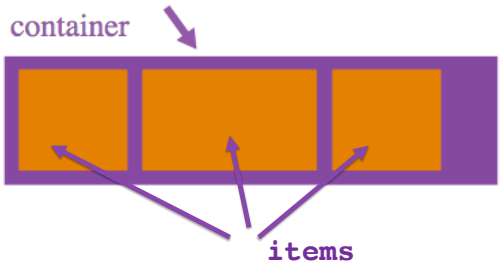
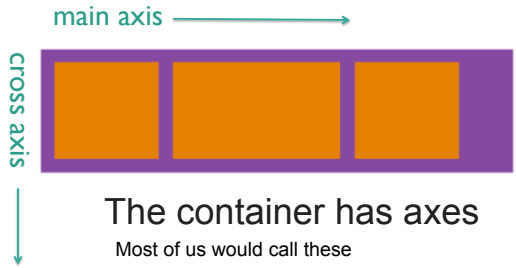
# Layouts with Flexbox


flex involves a container and contents






## The container has axes

Most of us would call these

- horizontal
- vertical

## The container lays out the items

```
container: {
 flexDirection: 'column-reverse',
 flexDirection: 'column',
 flexDirection: 'row',
 flexDirection: 'row-reverse',
}
```

## flex items have sizes

They have a "favorite" size

flexBasis

They know how to shrink from that and how to grow from that

flexShrink
flexGrow

## flexBasis is that favorite size

| When flexDirection: 'row' | When flexDirection: 'column' |
|---|---|
| • flexBasis == width<br>• width == width<br>• height == height | • flexBasis == height<br>• width == width<br>• height == height |

## Sizes of the items

- flexBasis = the item's "favorite" size.
  - in px
- If the viewport is increased from flex-basis ...
- flexGrow = by how much it grows
  - unitless - a relative number
- if the viewport is decreased from flex-basis ...
- flexShrink= by how much it shrinks
  - unitless - a relative number

## To have fixed widths/heights, set flexBasis on each item

```
.anItem {
  flexBasis: 150;
}
```

**width and height will still work as well**

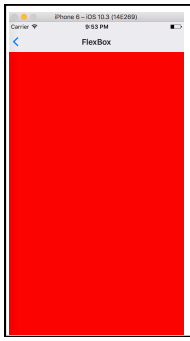
```
item1:{
  flexGrow: 3;
}
item2:{
  flexGrow: 1;
}
item3:{
  flexGrow: 2;
}
```

To have relative widths/heights set flexGrow/ flexShrink on each item

**These numbers are unitless and relative to one another**

## A single number is a shorthand

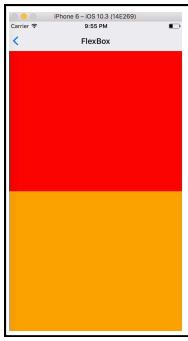
If you provide a flex property to elements in the same container, and that has a number, then those numbers are compared and allocated the free space available.



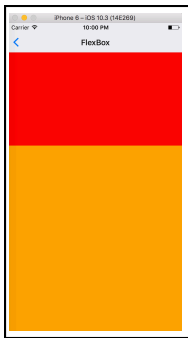

```
const styles = {
 container: {flex: 1,},
 box1: {flex: 1,
  backgroundColor: 'red',},


}
<View style={styles.container}>
 <View style={styles.box1} />

</View>
```



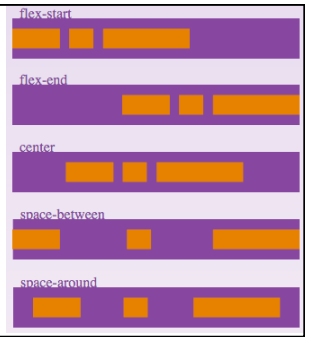

```
const styles = {
 container: {flex: 1,},
 box1: {flex: 1,
  backgroundColor: 'red',},
 box2: {flex: 1,
  backgroundColor: 'orange',},
}
<View style={styles.container}>
 <View style={styles.box1} />
 <View style={styles.box2} />
</View>
```




```
const styles = {
 container: {flex: 1,},
 box1: {flex: 1,
  backgroundColor: 'red',},
 box2: {flex: 2,
  backgroundColor: 'orange',},
}
<View style={styles.container}>
 <View style={styles.box1} />
 <View style={styles.box2} />
</View>
```

```
container: {
 justifyContent: '_____',
}
```

justifyContent controls distribution of children along the main axis



## tl;dr

- React Native layouts are handled with a framework that mimics some features of CSS flex layouts
- The learning curve for flexbox is high
- To put things side-by-side, you put them in a container with a flexDirection of 'row'
- Then they will share the space in that row based on
  - flexBasis - their preferred size
  - flexGrow - If there's extra space, how do they share it?
  - flexShrink - Not enough space? How do they donate it?
- "flex: <number>" is a shortcut for flexBasis, flexGrow, and flexShrink