Layout

- Box model
- Widget sizes
- Measure and Layout Steps
- Implementations: fixed, centred, wrap, fill

(User Interface) Layout

- In general English:
  - The way in which the parts of something are arranged
  - The way in which text or pictures are set out on a page
- In user interface architecture:
  - The visual arrangement of widgets in a user interface

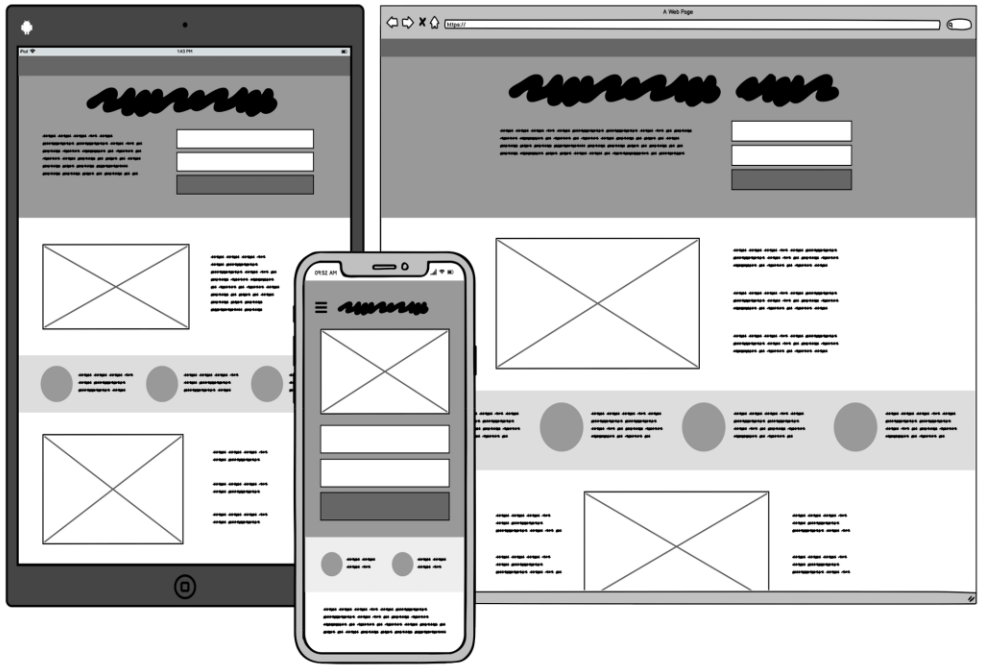
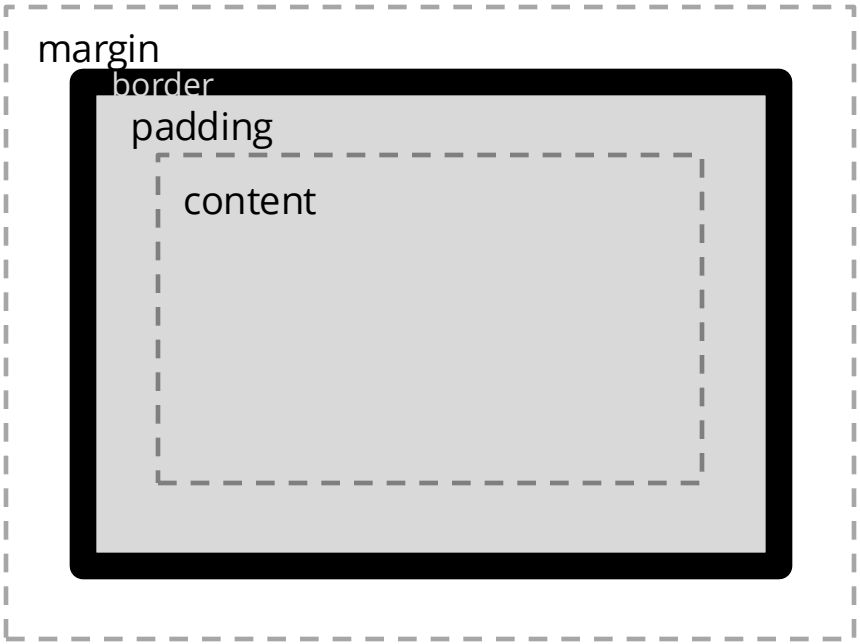


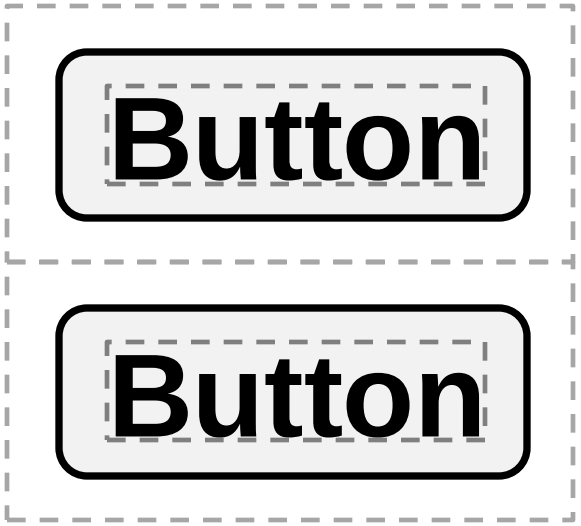
image from https://balsamiq.com/learn/articles/what-are-wireframes/

Box Model

- UI elements typically use a hierarchy of dimensions for size
- For example, the CSS Box Model has 4 dimensions:
  - **margin**: "outside" space away from other elements
  - **border**: thickness of stroke outlining element
  - **padding**: "inside" space between border and content
  - **content**: the actual content of the element

the margin is not part of "rendered" element

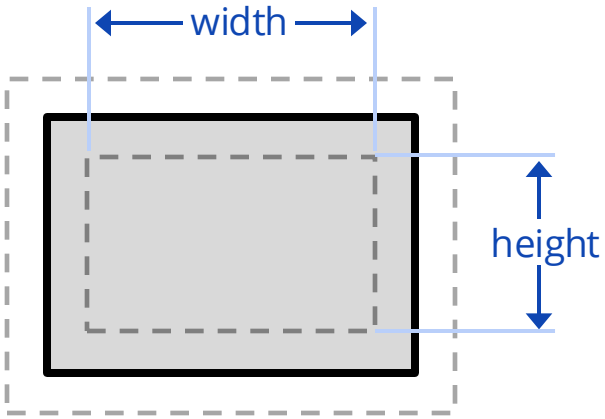




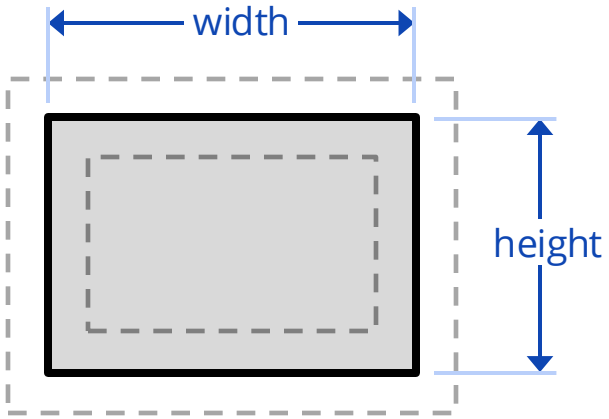
https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/The_box_model

Box Model “width” and “height”

- How does element width and height work with the box model?
- Standard CSS Box Model
  - width and height defined by the *content size*
  - actual rendered size is content size plus padding and border
- Alternative CSS Box Model
  - width and height define the *rendered element size*
  - content size is actual rendered size minus padding and border



Standard CSS Box Model



Alternative CSS Box Model

https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/The_box_model

css-box

Change box model with box-sizing CSS attribute

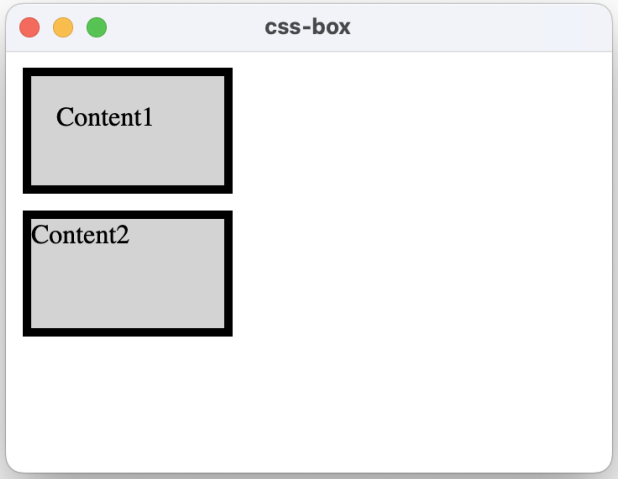
In CSS, can set top, right, bottom, left dimensions


```css
.box {
  box-sizing: border-box;
  width: 125px;
  height: 75px;
  margin: 10px;
  padding: 15px;
  border: 5px solid black;
  background: lightgrey;
}
```

Use alternate box model

Note CSS can set different margin, border, and padding for each side

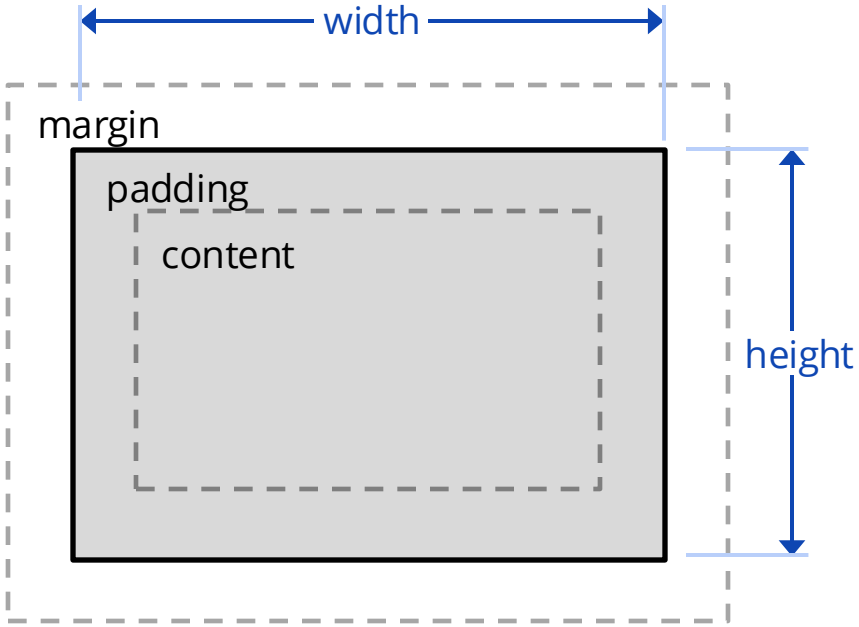
```html
<div class="box">Content1</div>
<div class="box">Content2</div>
```

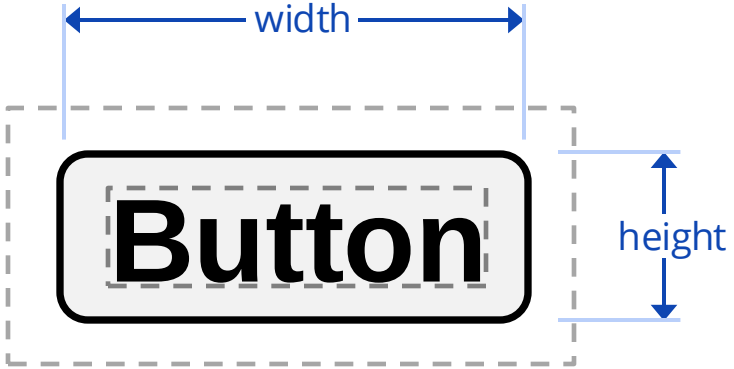


https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/The_box_model

SimpleKit Box Model

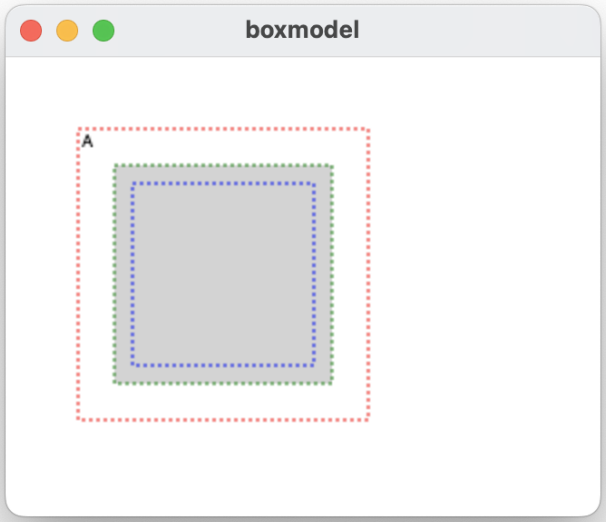
- Uses only 3 dimensions:
  - margin, padding, content
  - top, right, bottom, left cannot be set separately
- Uses CSS alternate box model for width and size:
  - width and height define the *size of the rendered element*





boxmodel

- Implemented in SKElement
- Box model properties to set
  - margin
  - padding
  - width, height (can be undefined)
- box model calculated sizes
  - contentBox, paddingBox, marginBox
- debug toggles boxModel visualization
  - `drawBoxModel`
- What if width is less than 2 * padding?



Widget Sizes

- **Content Size** is the size of what's "inside" the widget
  - doesn't include padding or margin
  - doesn't consider a user-specified width or height
- **Intrinsic Size** is the "normal" widget size
  - includes padding and margin
  - uses content size *or* user-specified width or height
- **Layout Size** is how much space the widget occupies in the layout
  - includes padding and margin
  - can be intrinsic size, but ...
    some layout methods shrink or expand the widget

SimpleKit box model implementation

- SKContainer (in widget/element.ts)
  - `measure` calculates intrinsic size
  - `layout` sets element layout size
- SKButton (in widget/button.ts)
  - `updateContentSize` measures size of rendered text for button
  - Called when font or text changes

Two Pass Layout

1. **Measure**
   - Calculate intrinsic sizes of children
   - Calculate intrinsic size of children in layout
   - Update parent intrinsic size
2. **Layout**
   - Assign children position and size in layout
   - May override child position
   - May ignore child intrinsic size and set different size

Simplekit recursive layout root

- In imperative-mode.ts

```
function layoutRoot() {
  if (uiTreeRoot && gc) {
    // 1. calculate ”intrinsic size" of all widgets
    uiTreeRoot.measure();
    // 2. set position and size of all widgets
    uiTreeRoot.layout(gc.canvas.width, gc.canvas.height);
  }
}
```

Strategy Design Pattern for Layouts

- Factor out layout algorithm into object

```
interface LayoutMethod {
  measure: (elements: SKElement[]) => Size;
  layout: (
    width: number, height: number,
    elements: SKElement[]
  ) => Size;
}
```

width and height type

- Layout method property in SKContainer

```
protected _layoutMethod: LayoutMethod;
set layoutMethod(method: LayoutMethod | "default") {
  this._layoutMethod = (method !== "default”) ?
    method : new Layout.FixedLayout();
}
```

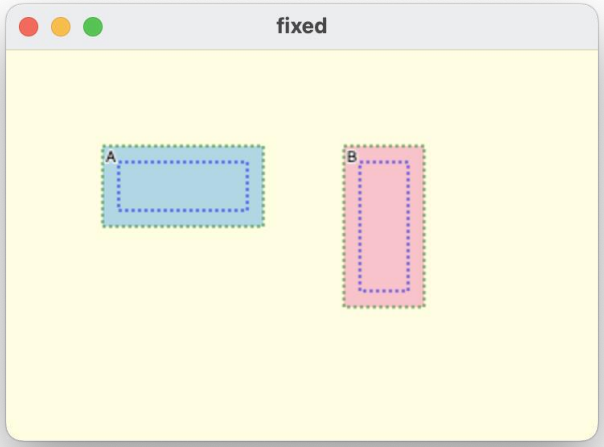
fixed

- Layout allows elements to set their x, y, width, height
  - simplest layout, essentially a "null" layout algorithm

```
// set layout method
root.layoutMethod = new Layout.FixedLayout();
```

- **fixed.ts** in simplekit/layout
  - warn if element is outside bounds
  - returns Size of bounds used

- Demos
  - try changing padding and margin

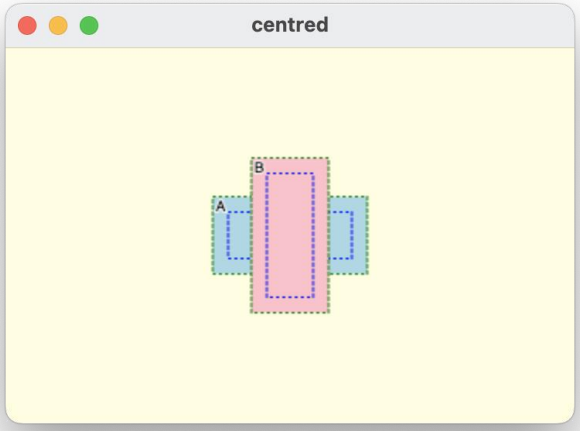


centre

- Centres all elements in centre of container
  - stacked back to front

```
// set layout method
root.layoutMethod = new Layout.CentredLayout();
```

- **centred.ts** in simplekit/layout
  - centering calculation using layout bounds
  - sets new x and y position for each element
  - warns if element is outside container bounds
  - supports fillWidth and fillHeight
  - returns Size of bounds used

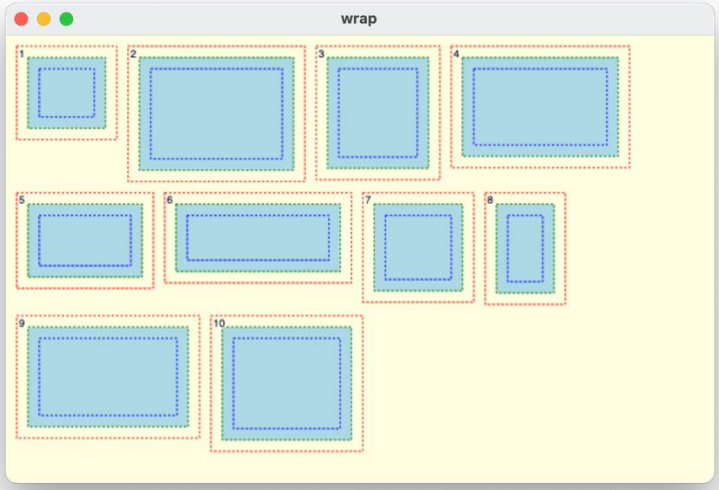


wrap

- Places elements in rows, wrapping to next row as needed
  - "gap" property in constructor

```
// set layout method
root.layoutMethod = new Layout.WrapRowLayout({ gap: 10 });
```

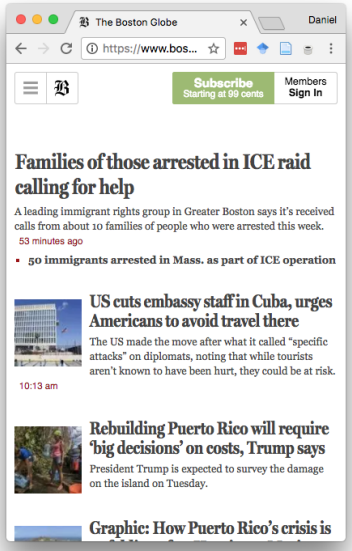
- **wrapRow.ts** in simplekit/layout
  - measure calculation is simple (is there an alternative?)
  - warns if elements wider than bounds
  - condition to wrap to next line
  - tallest element determines row height
  - warns if vertical overflow

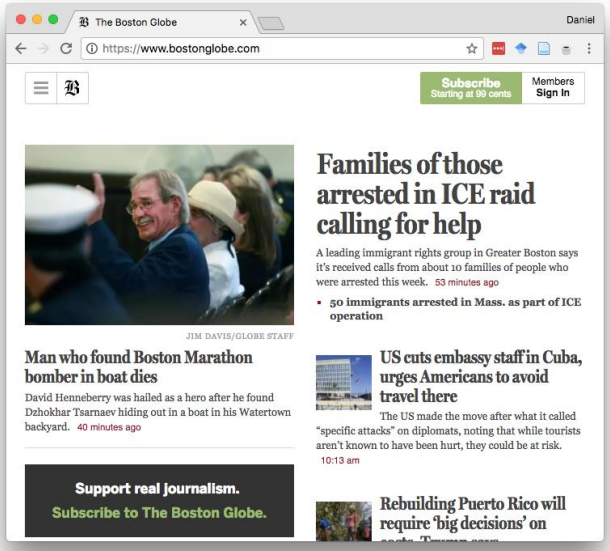


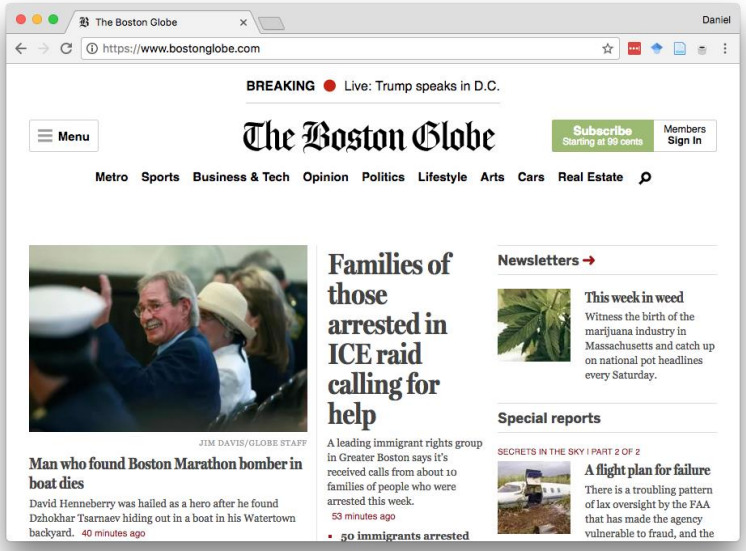
Responsive Layout

Dynamically reposition, resize, hide content in response to:

- Change in screen resolution (e.g. different computers or devices)
- Resizing the application window (e.g. user adjustments)





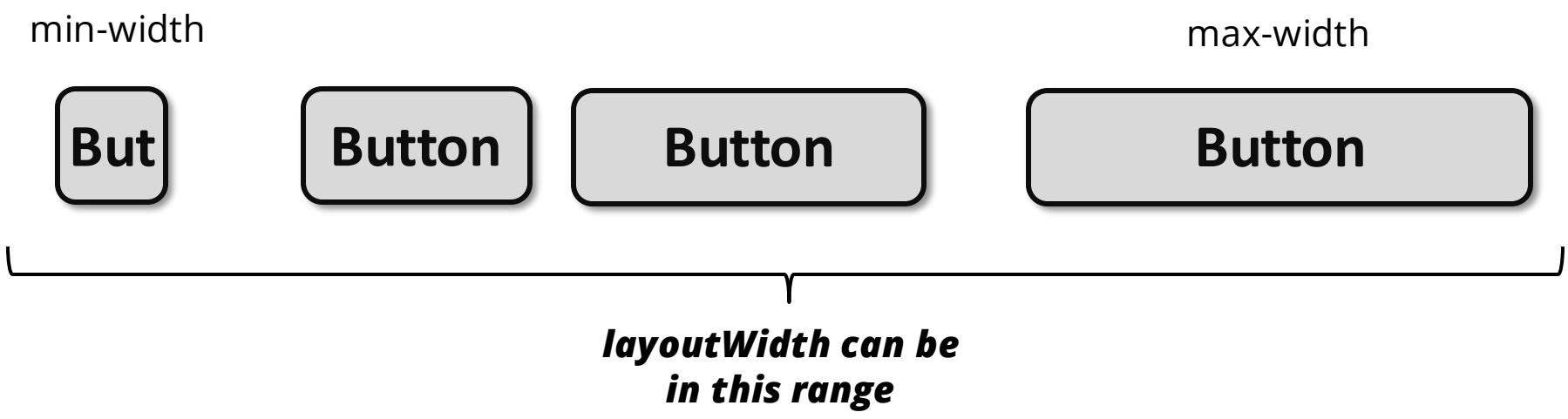


Widget Basis Sizes

- Specifying width or height is referred to as a “basis size”
- Some UI Toolkit support multiple basis sizes as "hints” for layout
  - Usually, a minimum and a maximum
  - e.g. CSS attributes:

```
min-width <= width <= max-width
```

SimpleKit only lets you specify a width, or let the layout or widget choose one for you with fillWidth

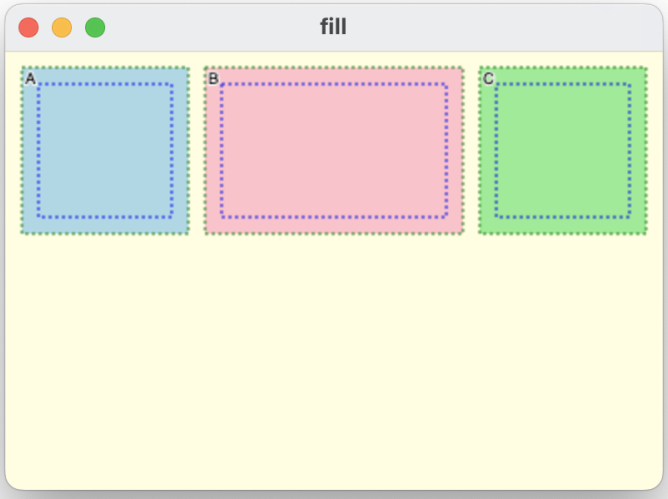


fill

- Lays out elements in a row, they can grow or shrink to fit space
  - "gap" property in constructor

```
// set layout method
root.layoutMethod = new Layout.FillRowLayout({ gap: 10 });
```

- **fillRow.ts** in simplekit/layout
  - measure calculation uses intrinsic width of all elements + gaps
  - element fillWidth to find proportion to fill
  - fillWidth property on SKElement
    (sets proportional change)
  - fillHeight option
  - calculate bounds



Layout Invalidation

- Every time widget size or layout-related property changes, at least some of the UI must be laid out again
- There could be multiple changes each frame of the run-loop
  - e.g. in response to events, model updates, etc.
  - Best practice is to run layout at most once per run-loop frame

Simplekit running layout only when necessary

- In imperative-mode.ts, flag to run layout process next frame

```
let layoutRequested = false;
// widgets call this to trigger layout next frame
function invalidateLayout() {
  layoutRequested = true;
}
```

- In runloop(), if flag is set, then run layout

```
if (uiTreeRoot && layoutRequested) {
  layoutRoot();
  layoutRequested = false;
}
```

todo

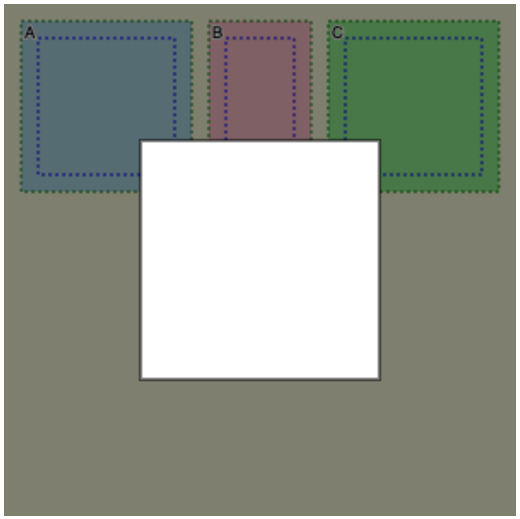
- More complex example of building UI with layouts
- Introduces componentization of layout parts into “views”
- Note lots of nesting of SKContainers
- Uses fake data to ”mockup” the app
- Using Settings.debug and Settings.debugLayout to troubleshoot

Exercise

EXERCISE

- Create an *overlay layout*
  - Often used for a modal UI panel
- Use the **fill layout demo** app layer that the overlay covers
- Use a semi-transparent fill for the "overlay" layer background
- The overlay foreground is a panel with white fill and black border
  - It should be centred with 80px space on all sides
- Show/hide the overlay with a keyDown

Showing overlay:



Not showing overlay: