

Flutter intro / Layouts 2

widget in a widget in a widget in a-

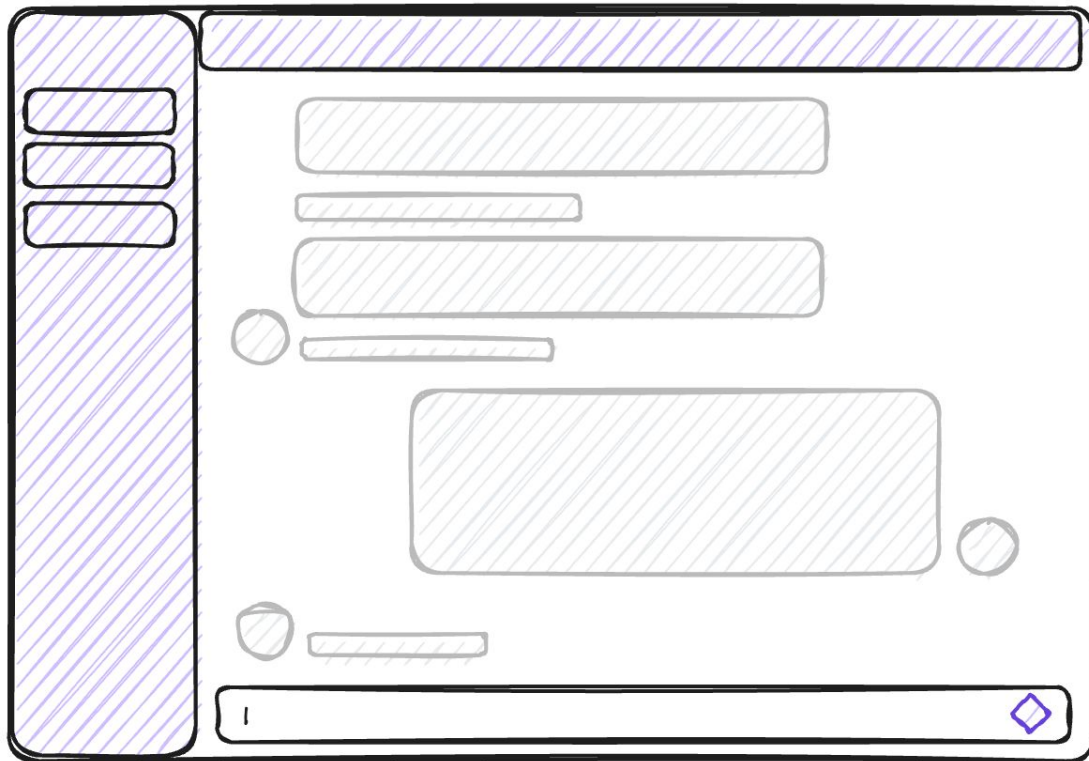


Flex - simple linear layout

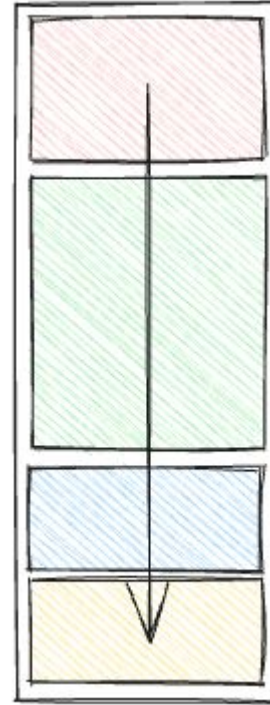
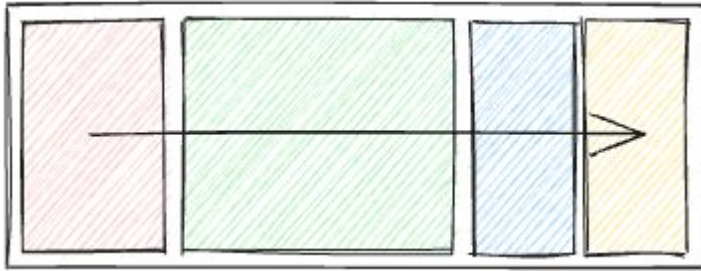
Flex - examples



Flex - examples



Flex - one-dimensional layouts

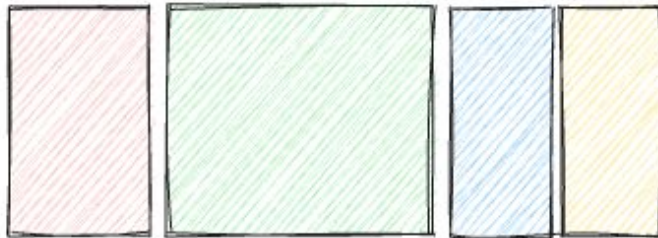


Flex - components

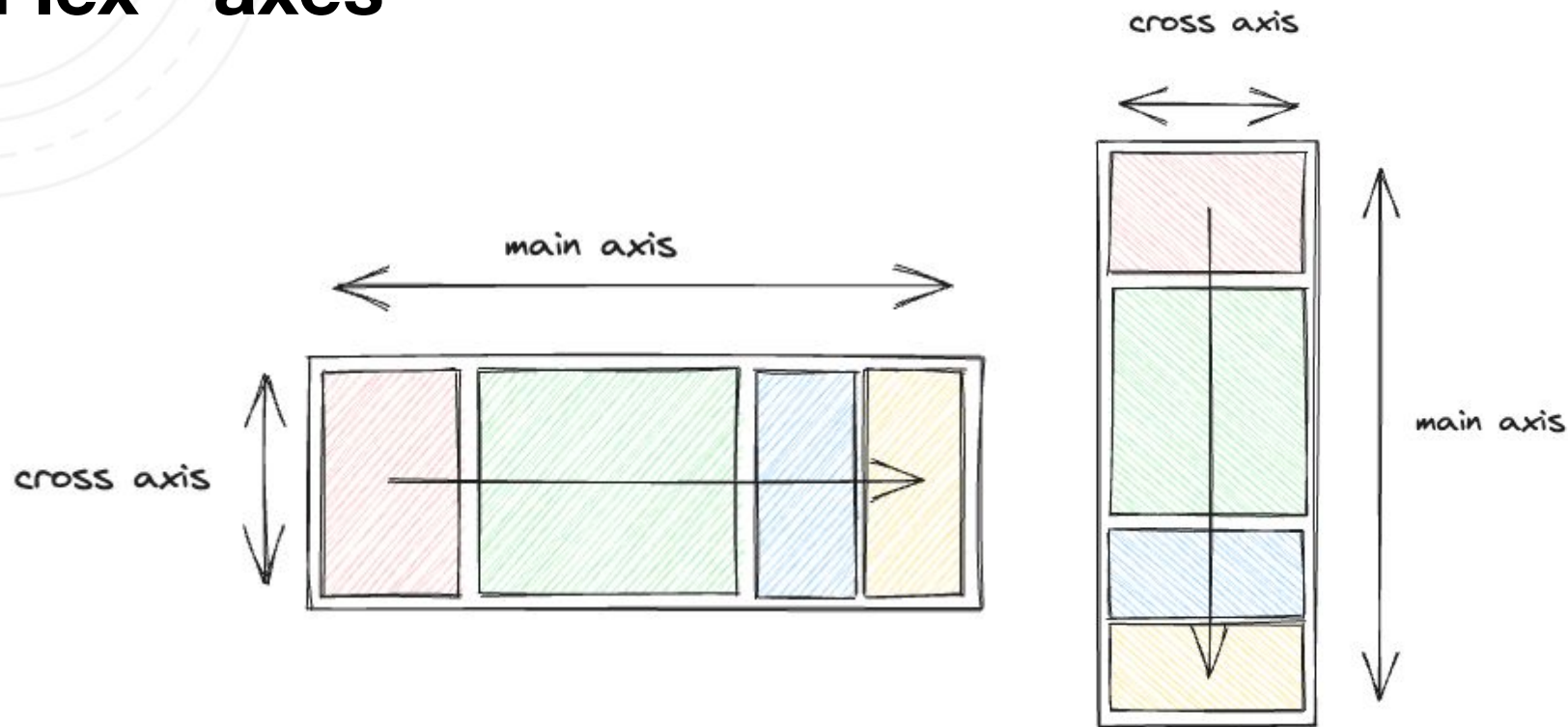
container



children



Flex - axes

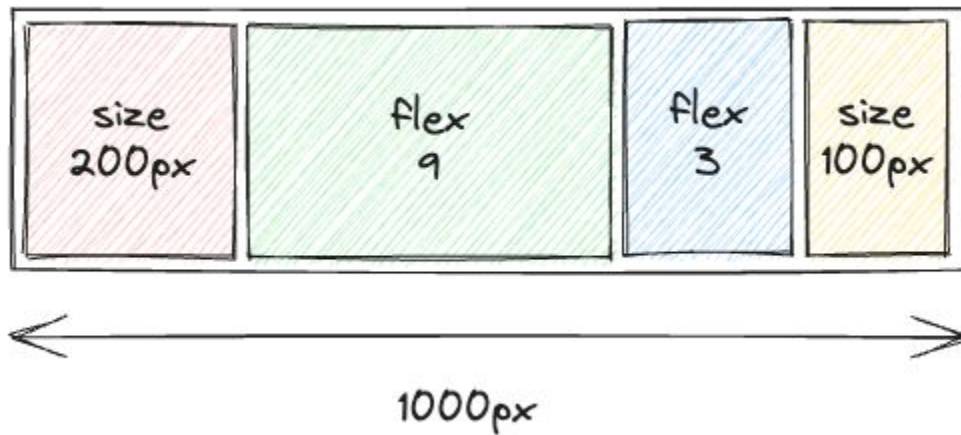


Flex - child item types

fixed size
size: X px

flexible
flex factor: N

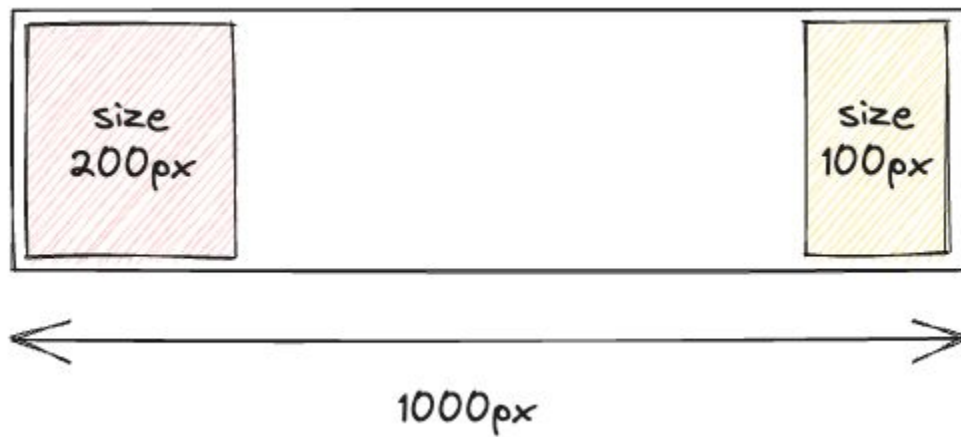
Flex - algorithm



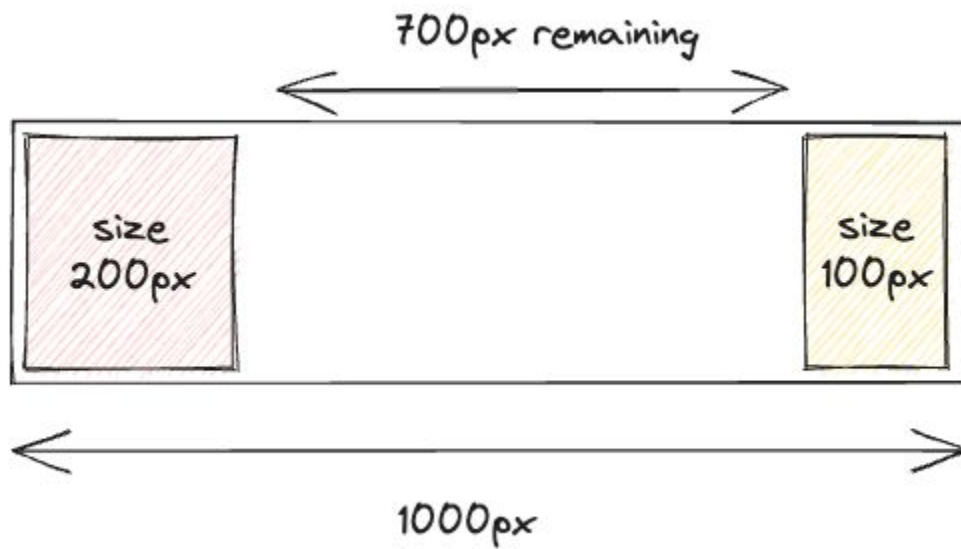
Reminder

**Constraints go down,
sizes go up**

Flex - algorithm



Flex - algorithm



Flex - algorithm

total flex: $9 + 3 = 12$

$$\frac{9}{12} \times 700\text{px} = 525\text{px}$$

$$\frac{3}{12} \times 700\text{px} = 175\text{px}$$



1000px

Flex - code

```

    SizedBox(
      width: 1000,
      height: 200,
      child: Row(
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: [
          // the red box
          Container(width: 200, color: Colors.red),
          // the green box
          Flexible(flex: 9, child: Container(width: double.infinity, color: Colors.green)),
          // the blue box
          Flexible(flex: 3, child: Container(width: double.infinity, color: Colors.blue)),
          // the yellow box
          Container(width: 100, color: Colors.yellow),
        ],
      ),
    );
```

Flex - code

```

    SizedBox(
      width: 200,
      height: 1000,
      child: Column(
        children: [
          // the red box
          Container(width: 200, color: Colors.red),
          // the green box
          Flexible(flex: 9, child: Container(width: double.infinity, color: Colors.green)),
          // the blue box
          Flexible(flex: 3, child: Container(width: double.infinity, color: Colors.blue)),
          // the yellow box
          Container(width: 100, color: Colors.yellow),
        ],
      ),
    );
```

Flex - code



```
// This:
```

```
Row(children: [childA, childB])
```

```
// is the same as this:
```

```
Flex(direction: Axis.horizontal, children: [childA, childB])
```


Flex - code



```
// This:
```

```
Column(children: [childA, childB])
```

```
// is the same as this:
```

```
Flex(direction: Axis.vertical, children: [childA, childB])
```

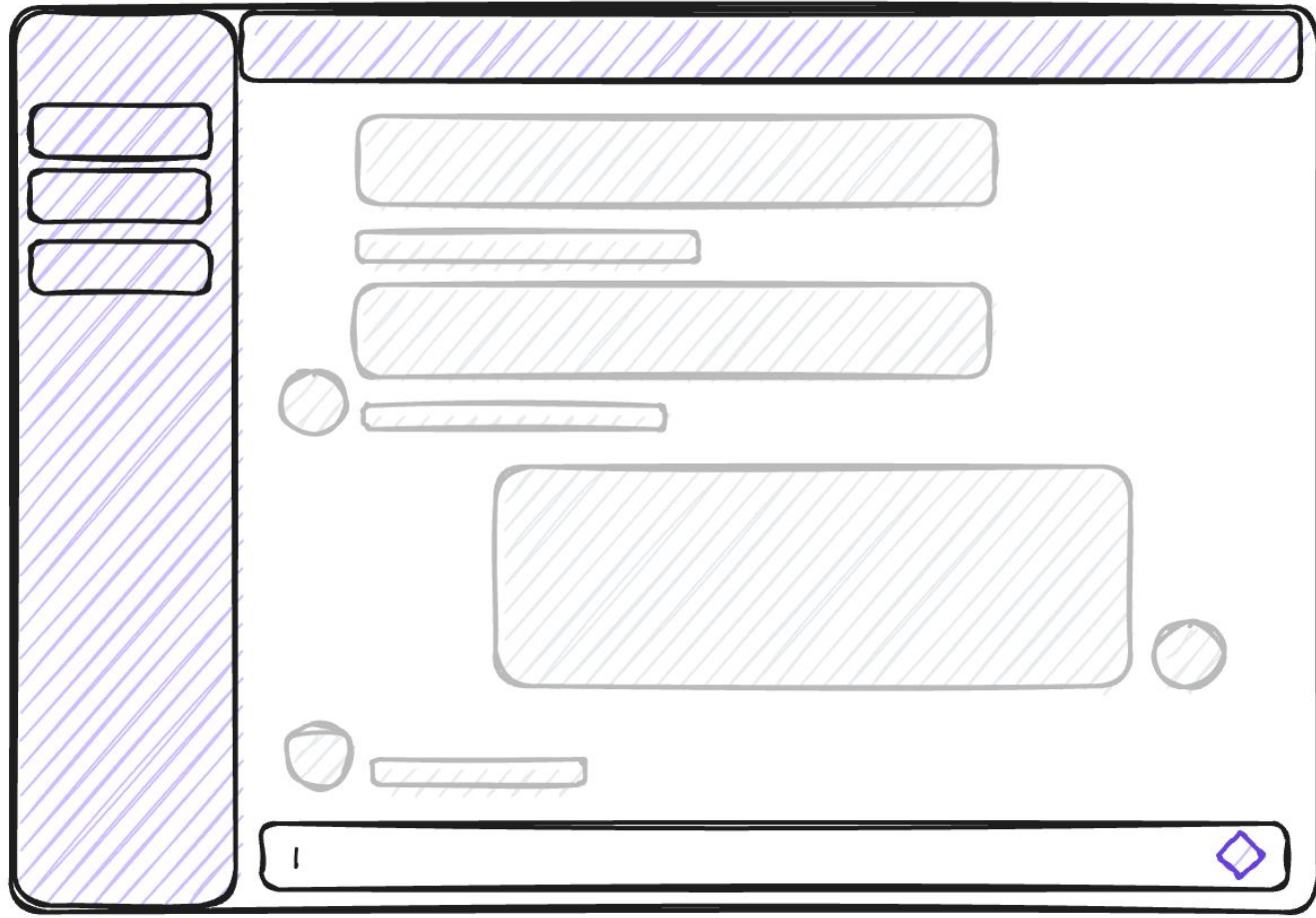
Flex - widget overview

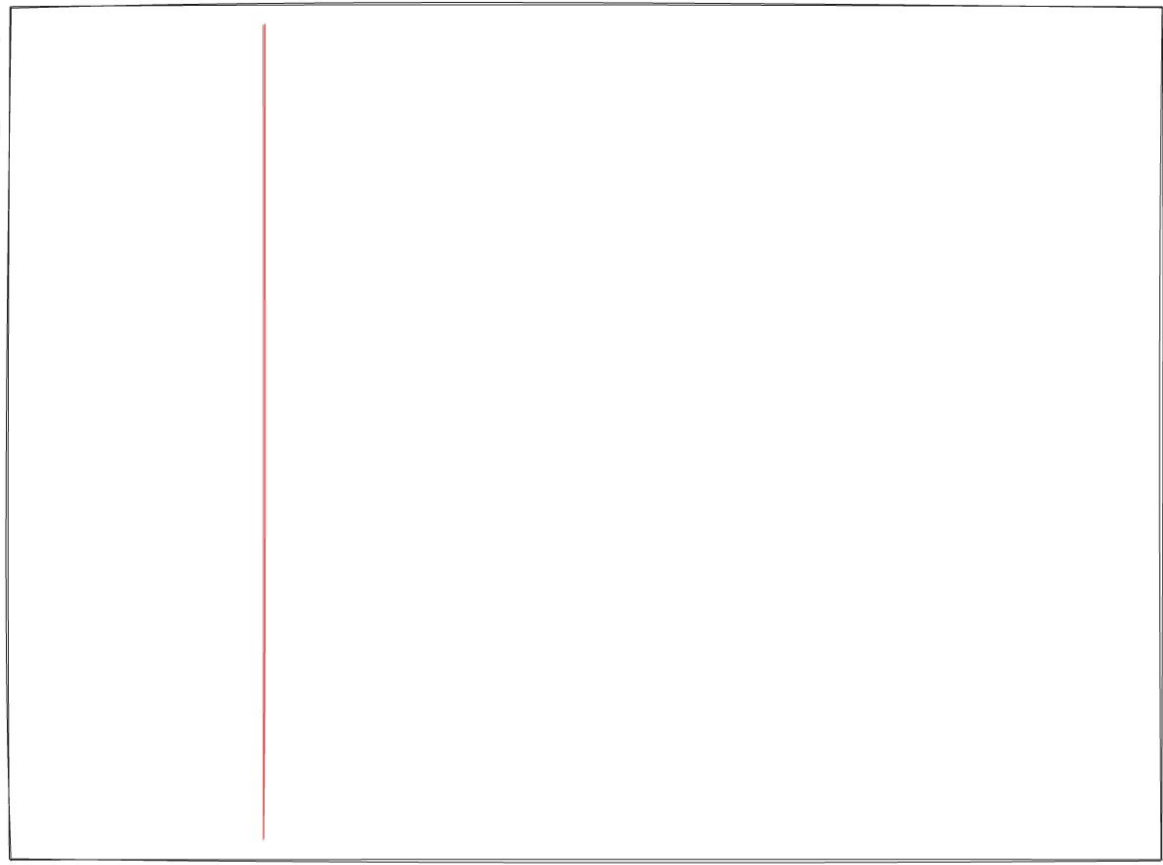


```
Flex(direction, children) // generalized flex container widget  
Row(children) // shorthand for horizontal Flex  
Column(children) // shorthand for vertical Flex
```

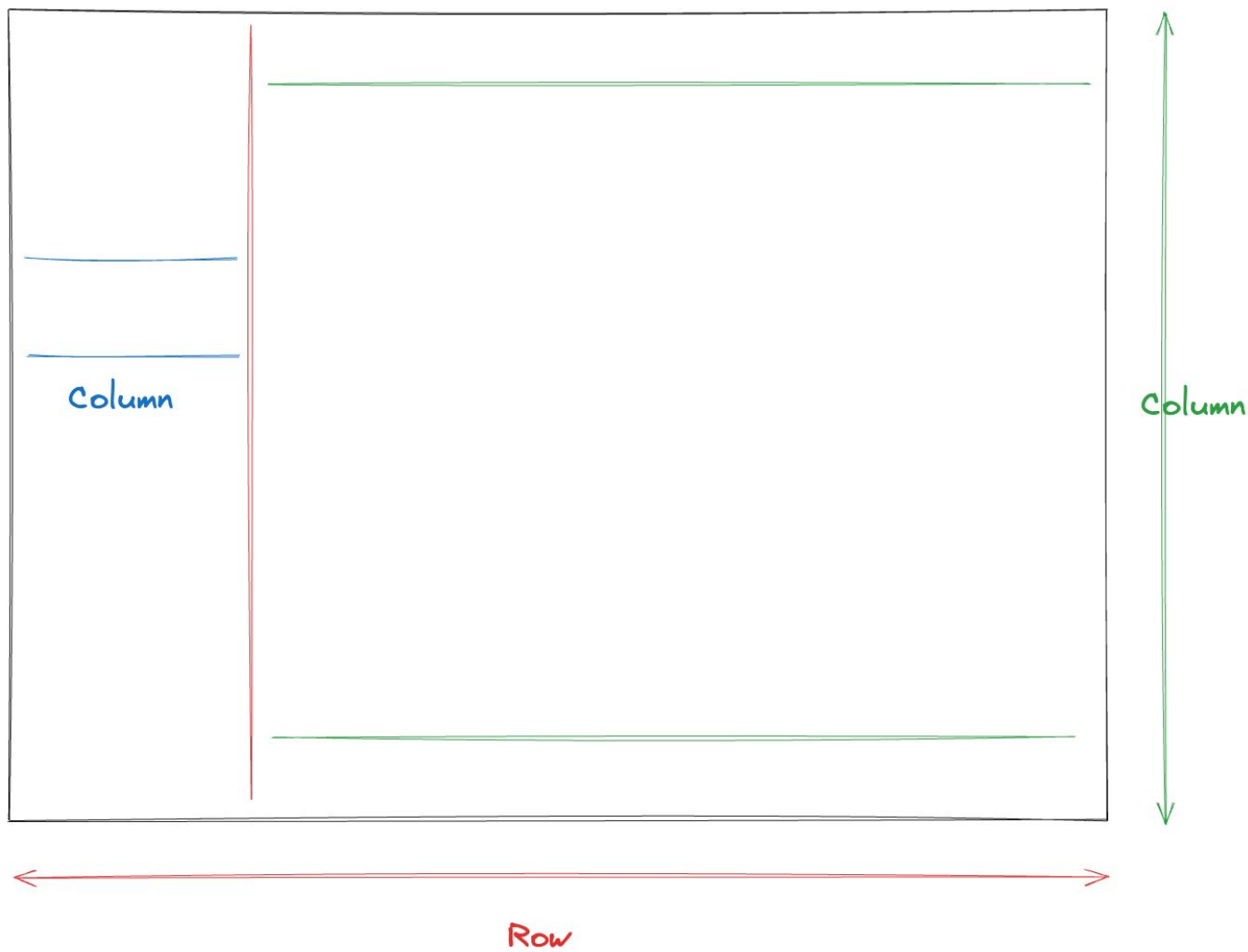
```
Flexible(flex, fit, child) // flexible child of a flex container. Child size customizable with `fit`  
Expanded(flex, child) // same as Flexible but forces its child to fill available space (expand)
```

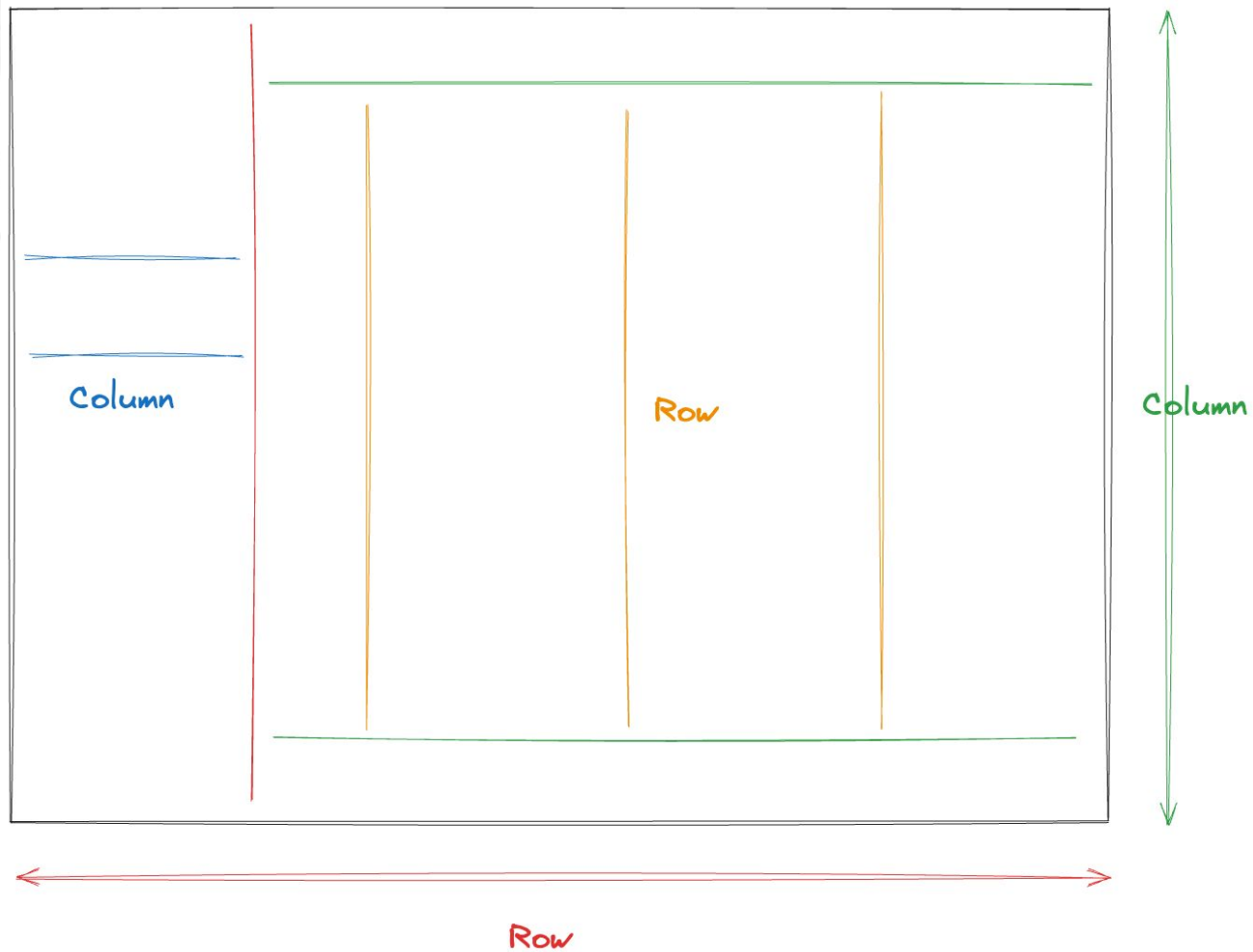
Break complex UI down





Row







**Problem: none of this
is scrollable**



Lists & friends

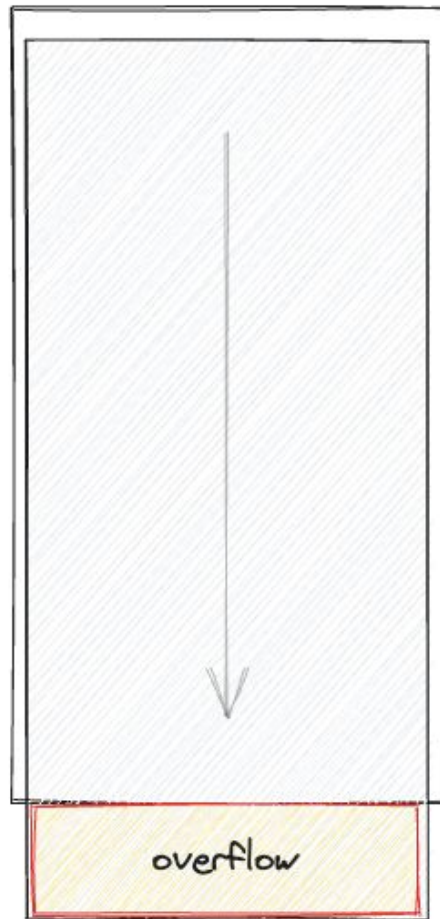
SingleChildScrollView

does not fit :(no scroll :/ overflow error 😱



```
Column(  
  children: [  
    // quite a lot of children  
  ],  
)
```

phone screen



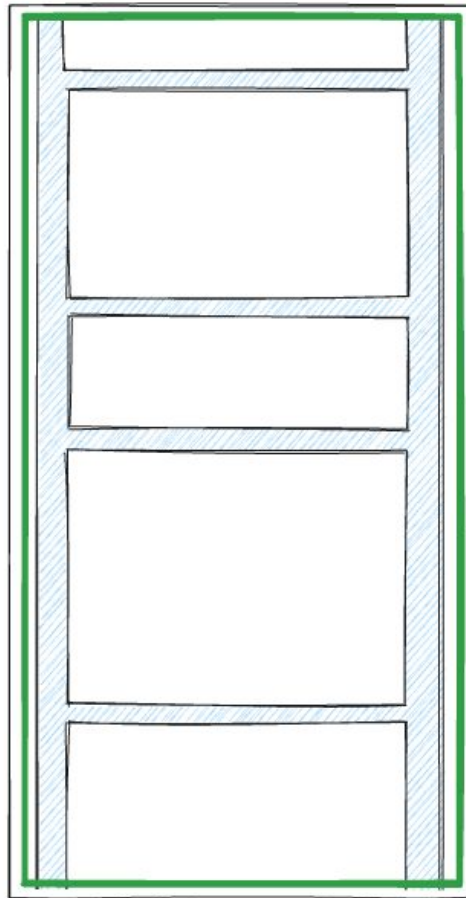
SingleChildScrollView

yes scroll 🕶️

still does not fit but in a good way :)

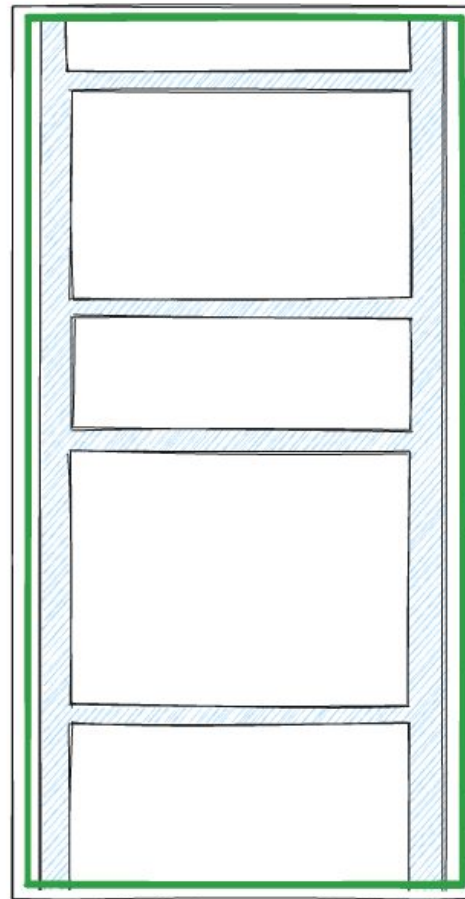
no overflow error :hackerman:

```
SingleChildScrollView(  
  child: Column(  
    children: [  
      // quite a lot of children  
    ],  
  ),  
)
```



ListView

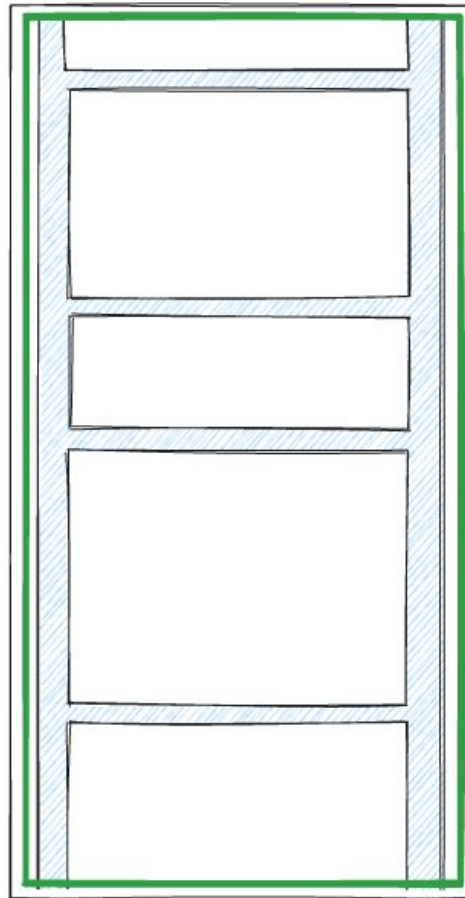
- better performance in long list setups
- provides quality of life helpers for working with lists



50 constructors of ListView

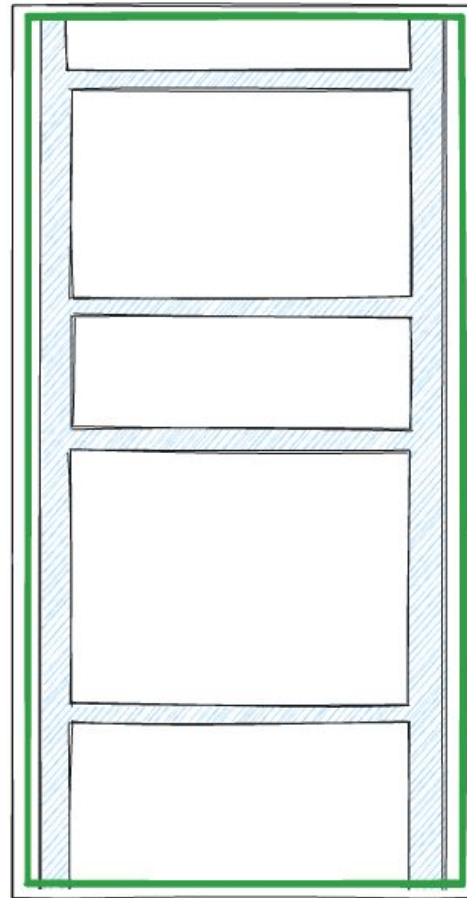


```
// just like SingleChildScrollView + Column
ListView(
  children: [
    // ...
  ]
)
```



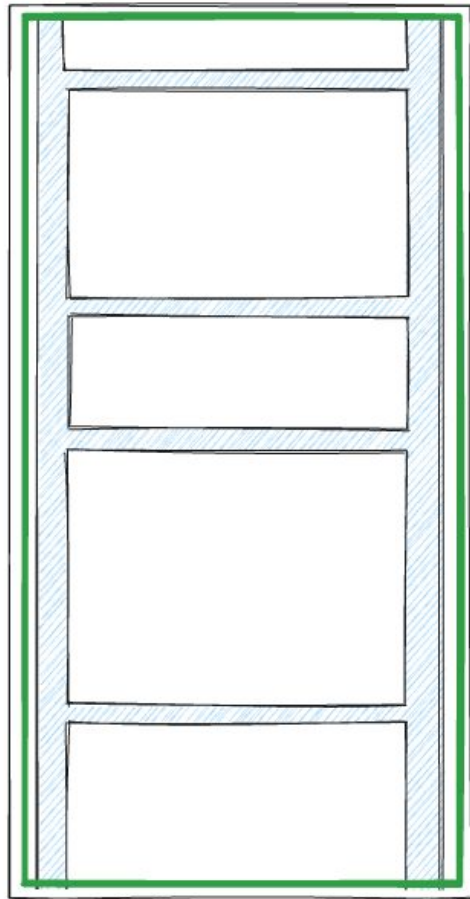
50 constructors of ListView

```
// The performant one
ListView.builder(
  itemBuilder: (context, index) => MyListItem(
    data: items[index],
  ),
  itemCount: items.length,
)
```



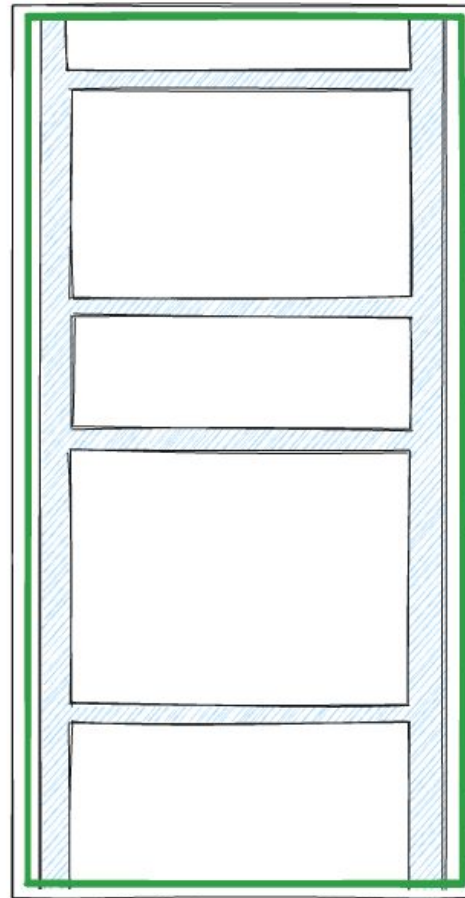
50 constructors of ListView

```
// The neat one with separators
ListView.separated(
  itemBuilder: (context, index) => MyChild(
    data: items[index],
  ),
  separatorBuilder: (context, index) =>
    const SizedBox(height: 8),
  itemCount: items.length,
)
```



50 constructors of ListView

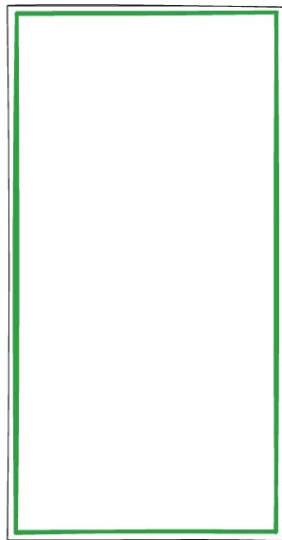
```
// The low-level one. Go crazy  
ListView.custom(  
    childrenDelegate: /* ... */,  
)
```



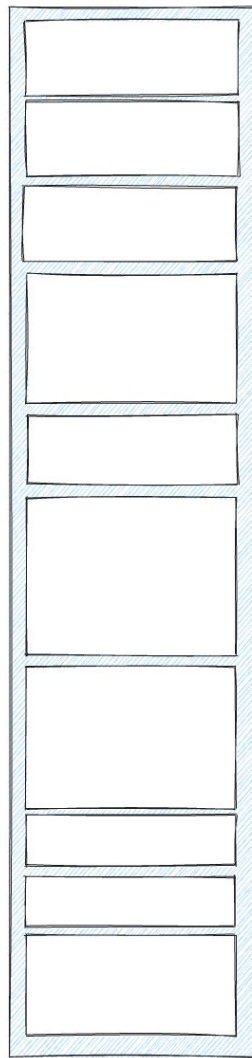


Performance

Virtualization

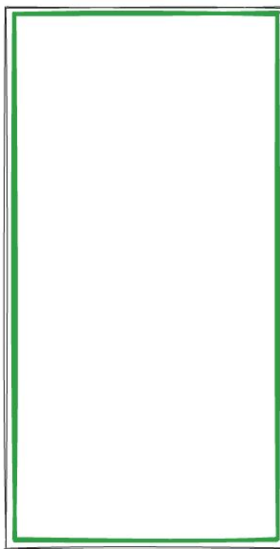


Viewport

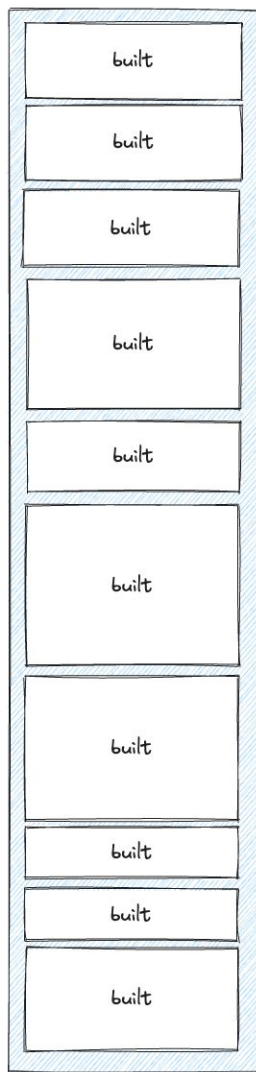


Content

Virtualization



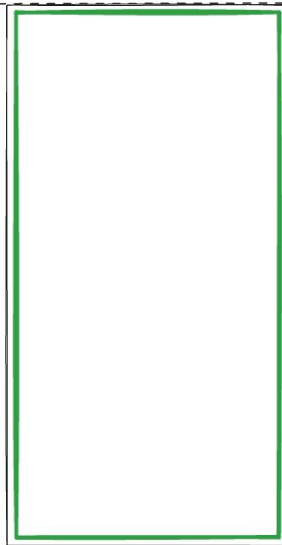
SingleChildScrollView



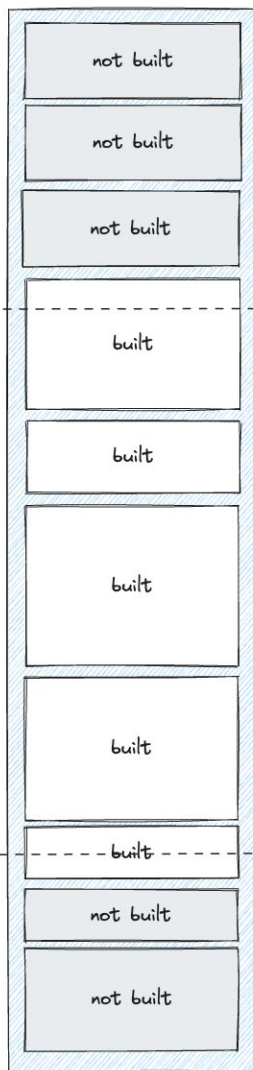
Column

Virtualization

ListView.builder

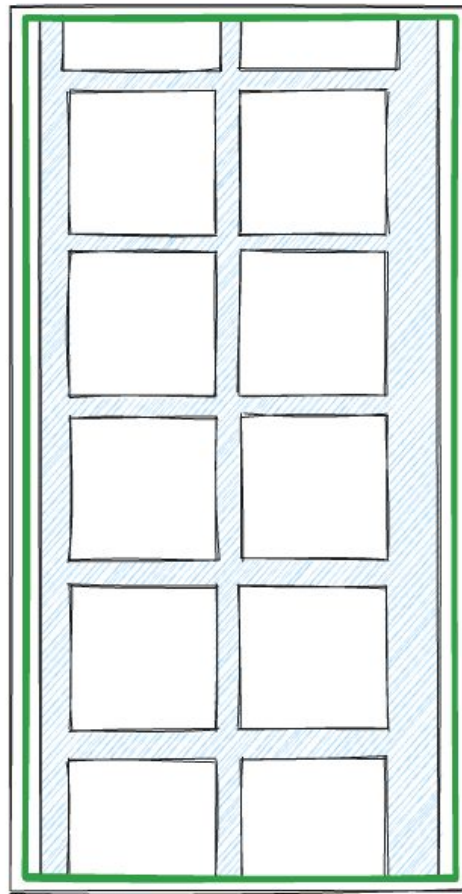


children built with itemBuilder



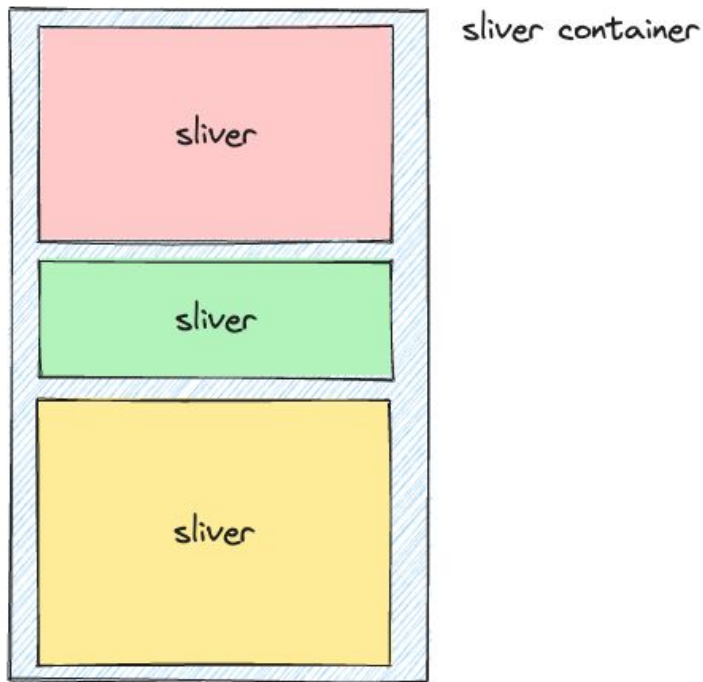
GridView

Same story as with ListView but you get a nice, responsive, even grid



Slivers

 Sliver - part of a scrollable view



Sliver-related widgets

- SliverList (ListView but as a sliver)
- SliverGrid (GridView sliver)
- CustomScrollView (container for slivers)
- SliverToBoxAdapter (embed a box as a sliver)
- Notable mention: [sliver tools | Flutter Package \(pub.dev\)](#)

Slivers vs boxes

(Render)Sliver ← incompatible! → (Render)Box

CustomScrollView

```

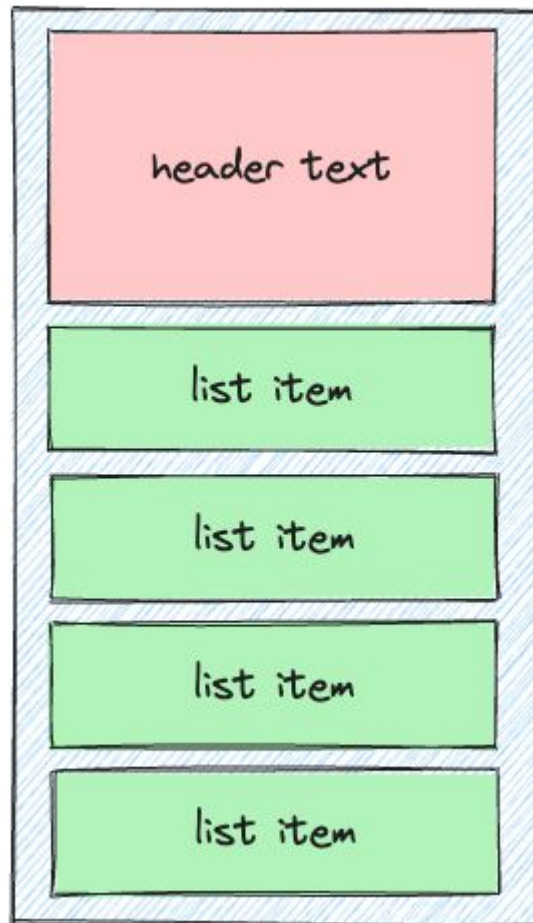
// This
ListView(/* ... */)

// is the same as this
CustomScrollView(
  slivers: [
    SliverList(/* ... */),
  ],
)

```

CustomScrollView

How do you go about this?

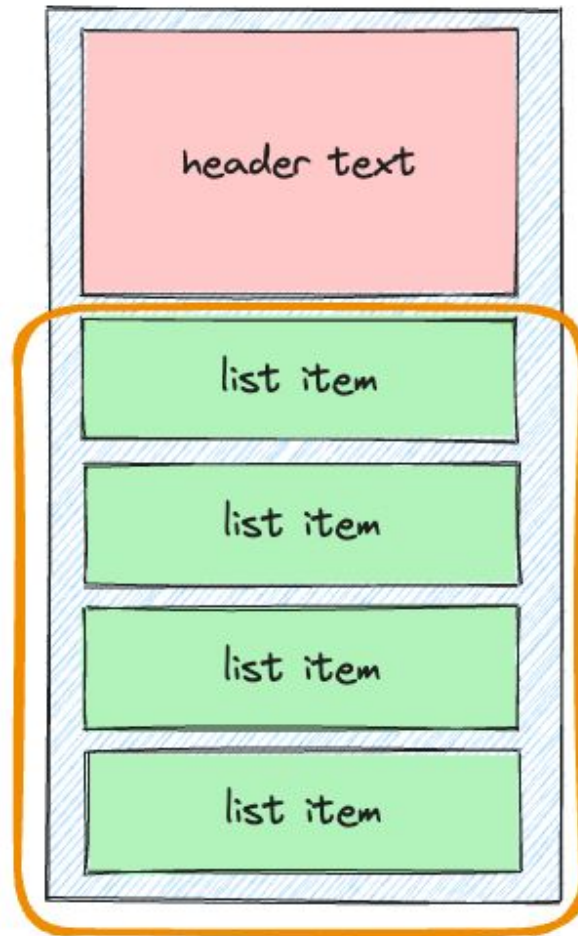


CustomScrollView

How do you go about this?



```
Column(  
  children: [  
    Text('header text...') ,  
    const SizedBox(height: 8),  
    // uh-oh: the header sticks to the top  
    // of the screen  
    ListView(  
      children: [  
        listItem1,  
        listItem2,  
        listItem3,  
      ],  
    ),  
  ],  
)
```

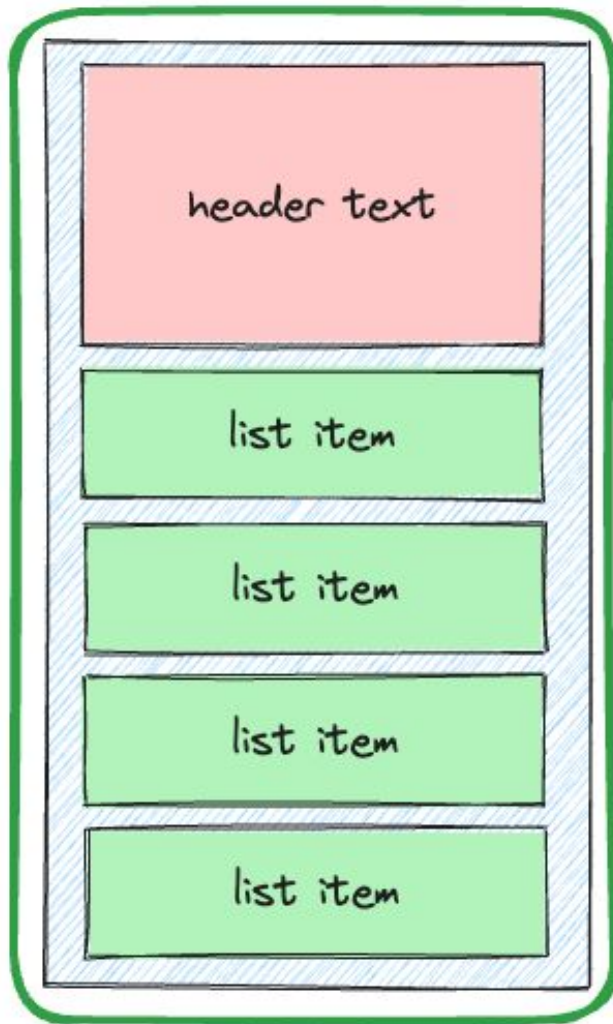


only this scrolls

CustomScrollView

How do you go about this?

```
CustomScrollView(  
  slivers: [  
    SliverToBoxAdapter(  
      child: Text('header text...'),  
    ),  
    SliverToBoxAdapter(  
      child: const SizedBox(height: 8),  
    ),  
    SliverList(  
      // the simplest delegate  
      delegate: SliverChildListDelegate([  
        listItem1,  
        listItem2,  
        listItem3,  
      ]),  
    ),  
  ],  
)
```



everything scrolls

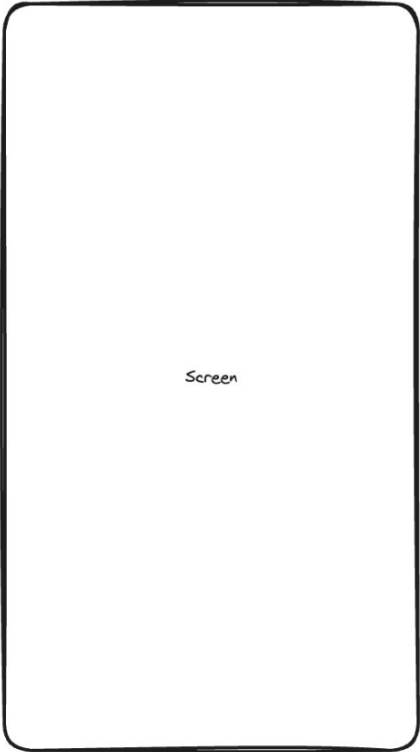


Other useful widgets

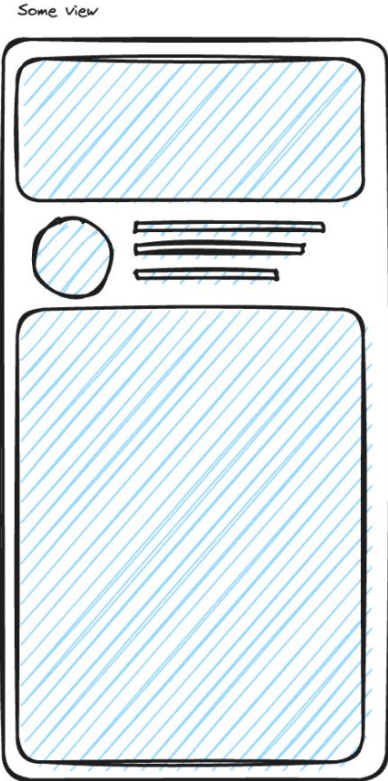


Stack

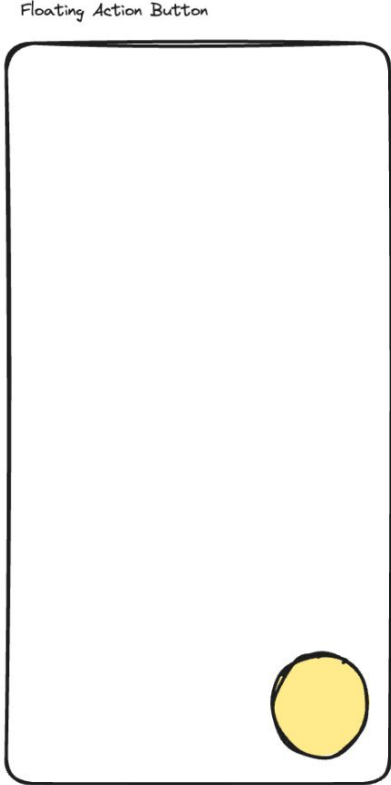
Stack



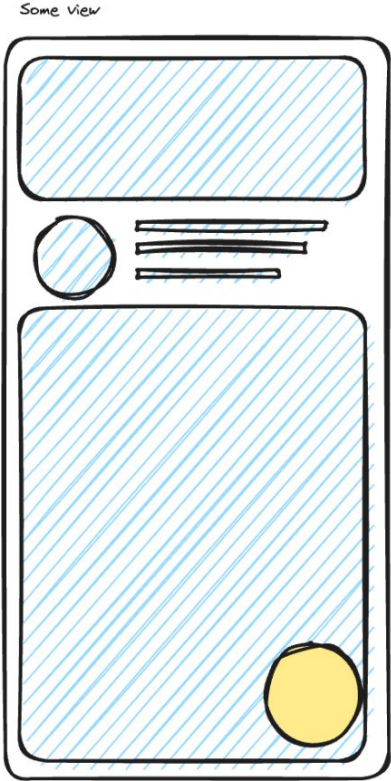
+

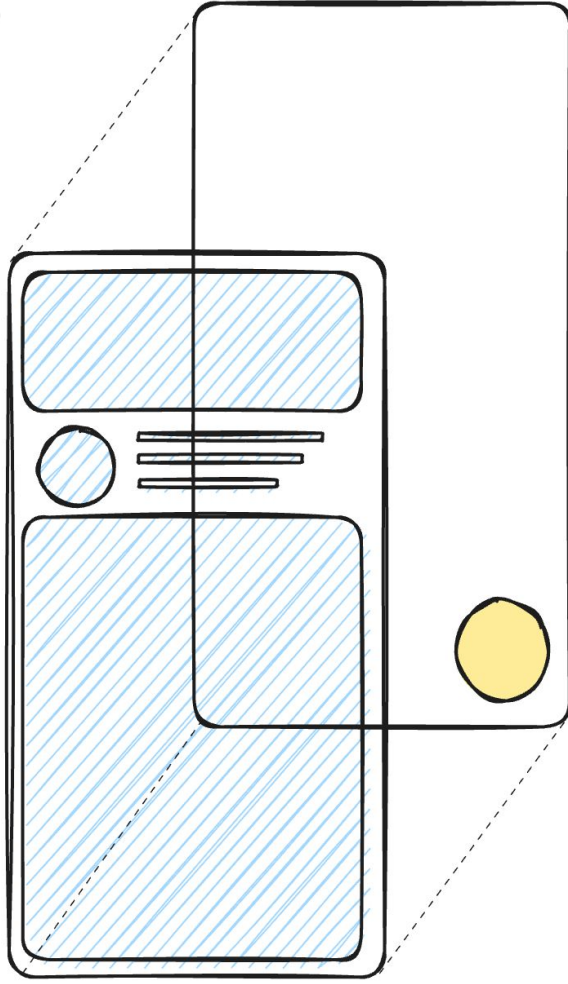


+



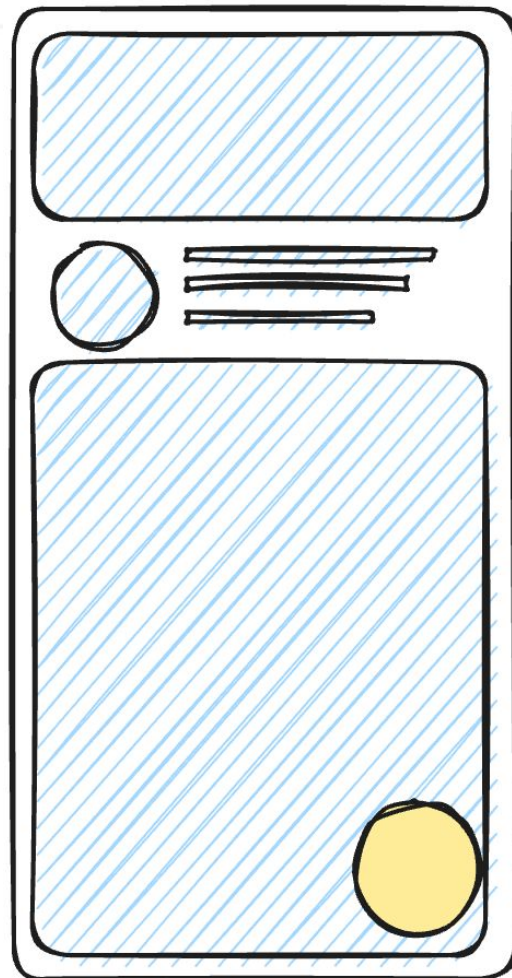
=





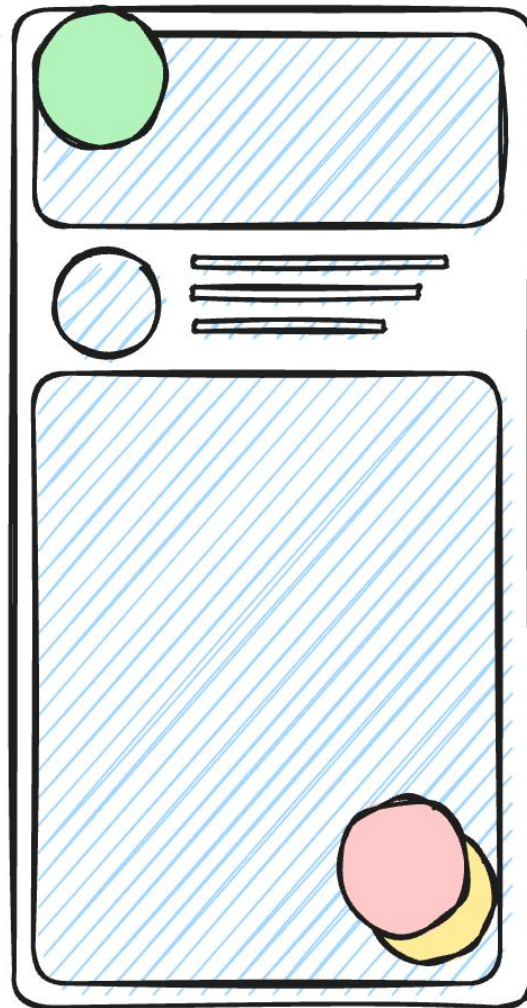
Stack + Positioned

```
Stack(
  children: [
    SomeView(), // the blue one
    // This widget positions content relatively to
    // Stack's edges
    Positioned(
      bottom: 32,
      right: 32,
      child: FloatingActionButton(), // the yellow one
    ),
  ],
)
```



Stack + Positioned

```
Stack(  
  children: [  
    SomeView(), // the blue one  
    // This widget positions content relatively to  
    // Stack's edges  
    Positioned(  
      bottom: 32,  
      right: 32,  
      child: FloatingActionButton(), // the yellow one  
    ),  
    // You can go crazy  
    Positioned(  
      bottom: 48,  
      right: 48,  
      child: FloatingActionButton(), // the red one  
    ),  
    Positioned(  
      top: 0,  
      left: 24,  
      child: FloatingActionButton(), // the green one  
    ),  
  ],  
)
```





GestureDetector

GestureDetector

Handle taps & other gestures

[Taps, drags, and other gestures | Flutter](#)

[GestureDetector class - widgets library - Dart API \(flutter.dev\)](#)

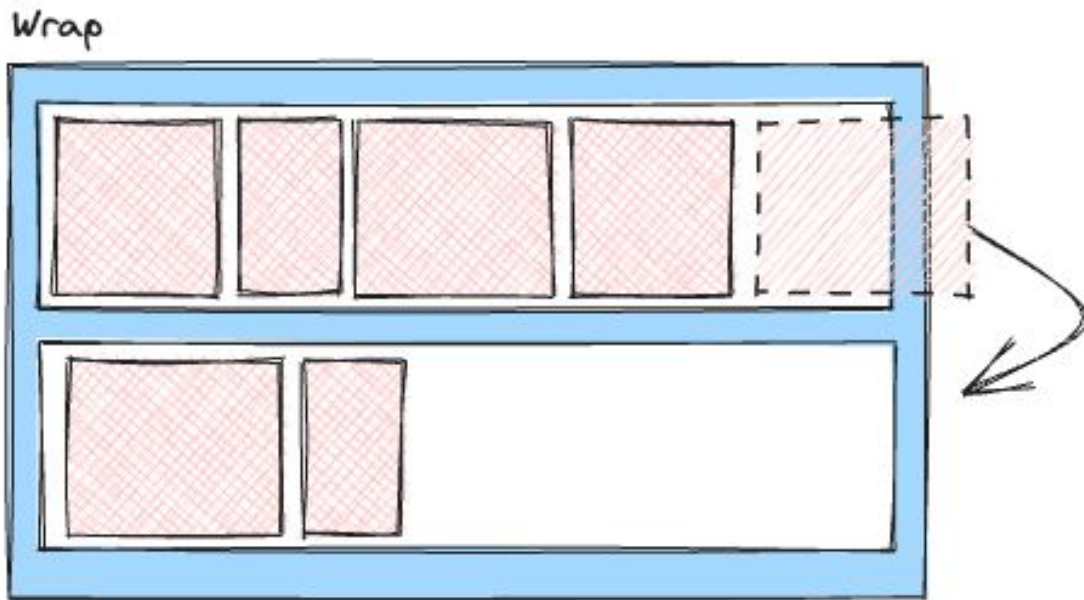
```
GestureDetector(  
  onTap: () { /* do stuff */ },  
  child: /* the thing you want to make interactive */  
)
```



Wrap

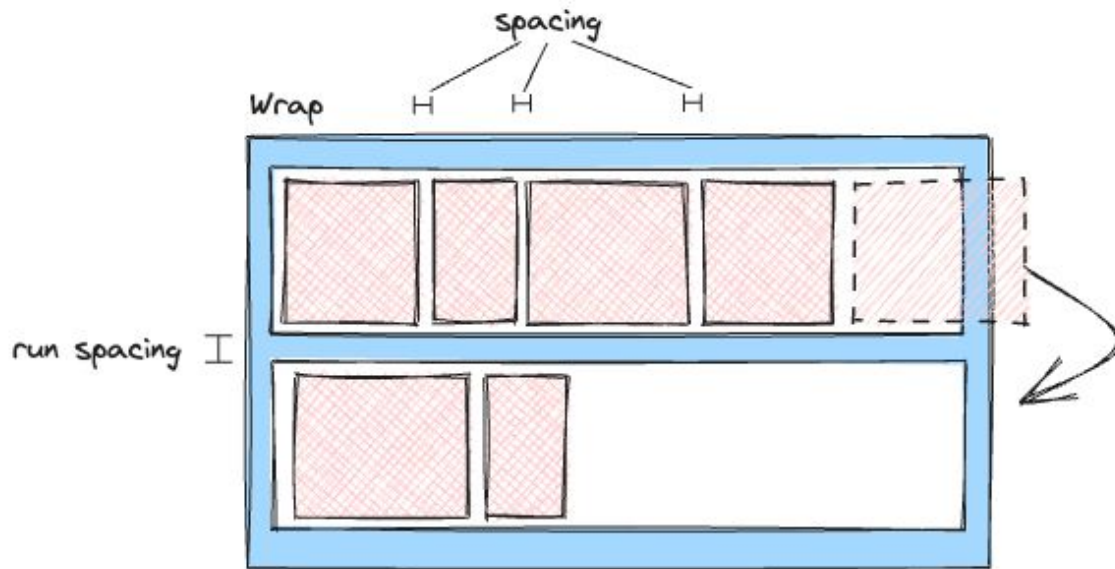
Wrap

It's a Flex/Row/Column that wraps to the next line/column



Wrap

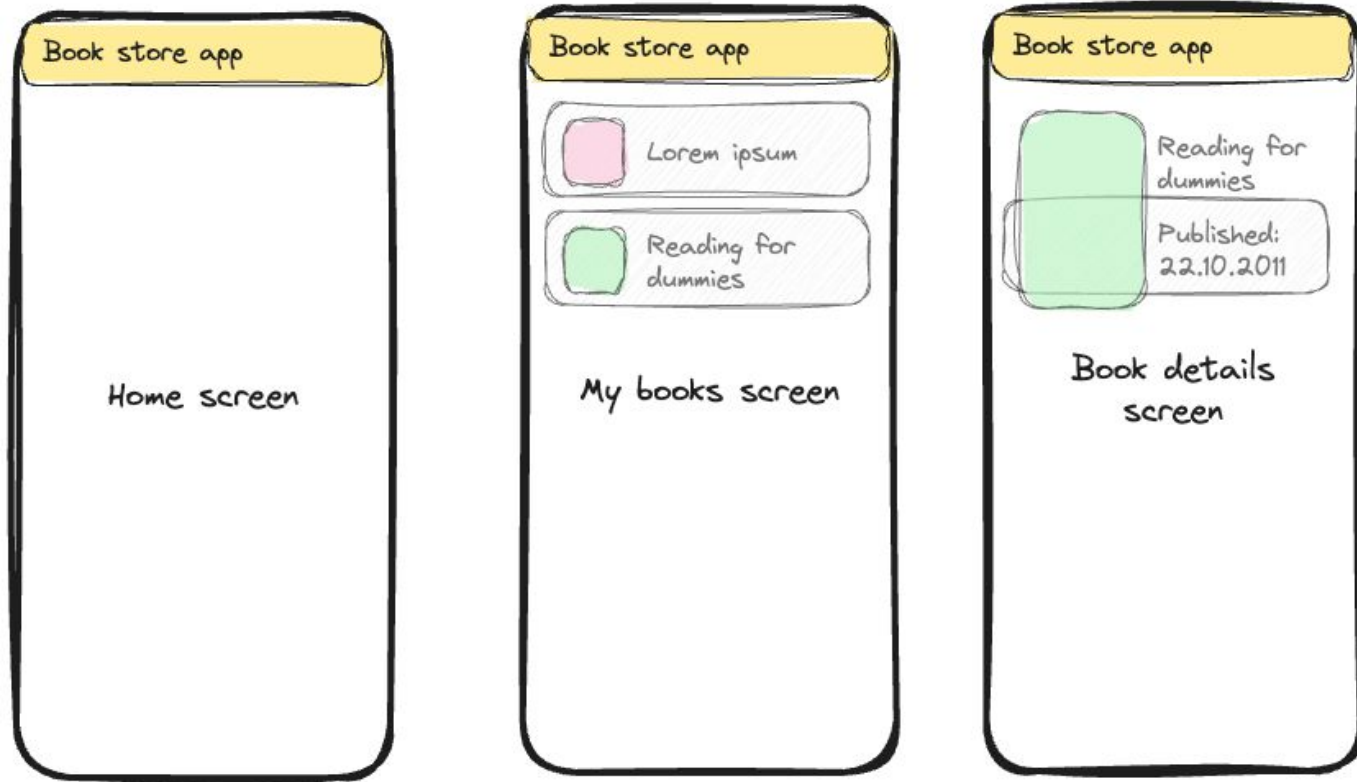
It's a Flex/Row/Column that wraps to the next line/column





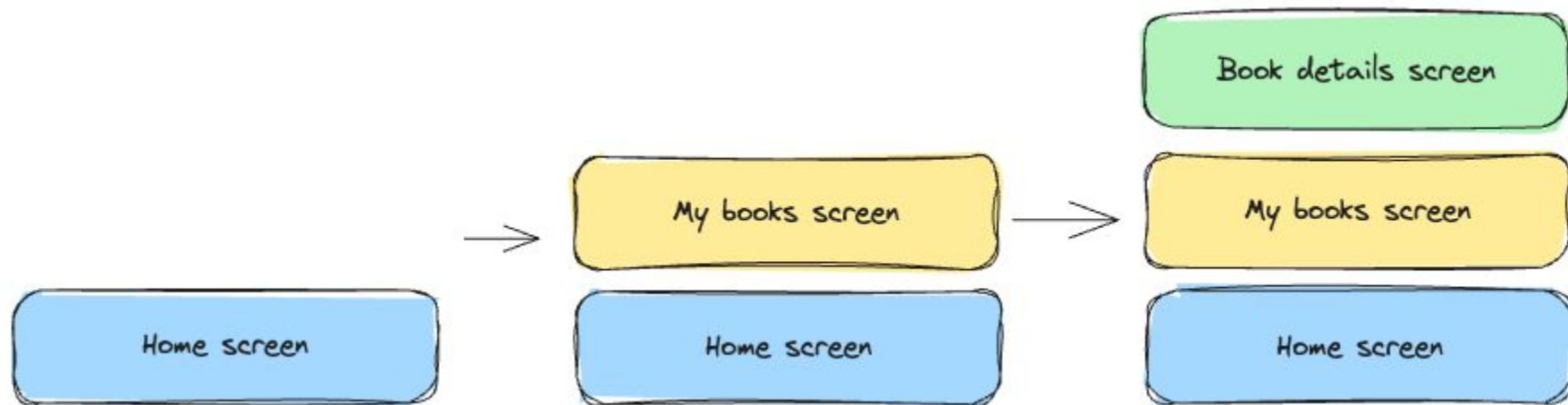
Navigation

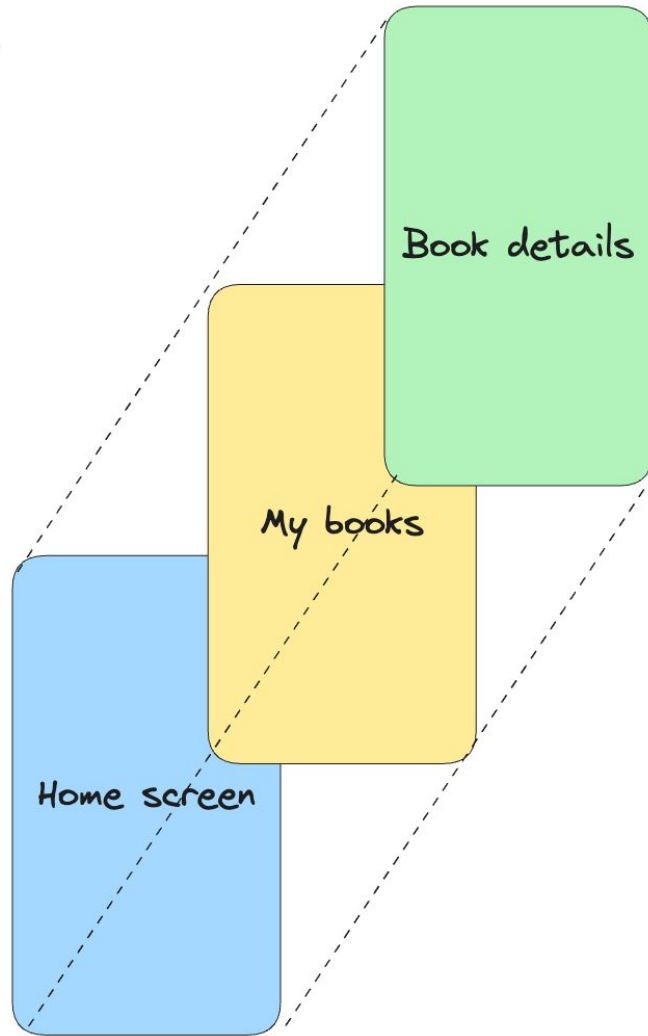
Navigation principles



Navigation principles

Screens stack on top of each other



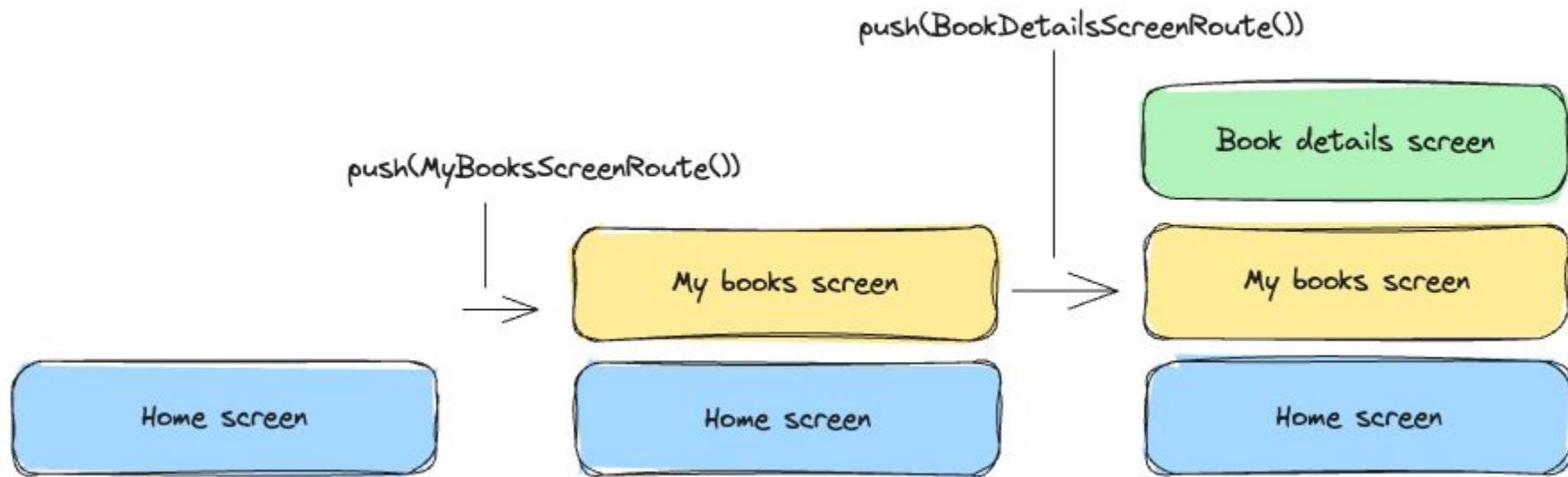




Imperative vs declarative

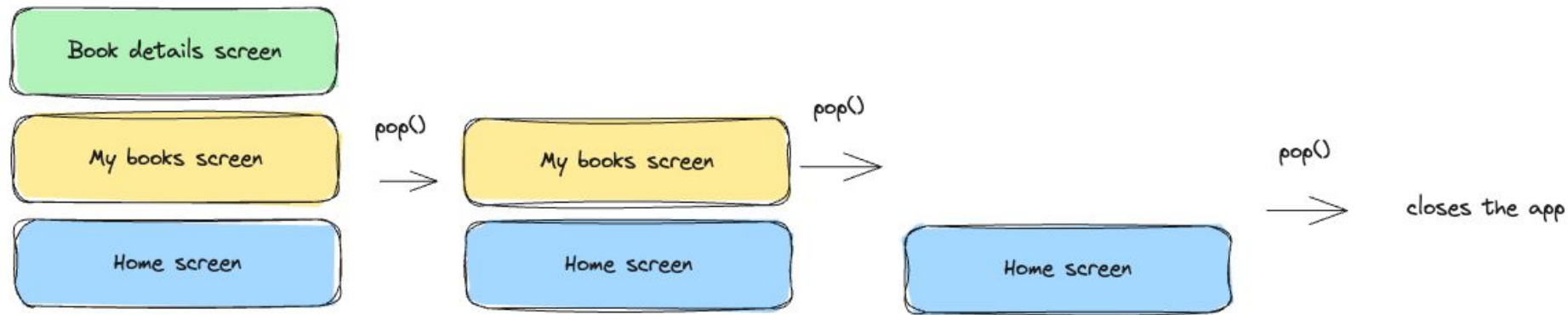
Imperative navigation

1. Start with a home screen
2. Push and pop screens as needed



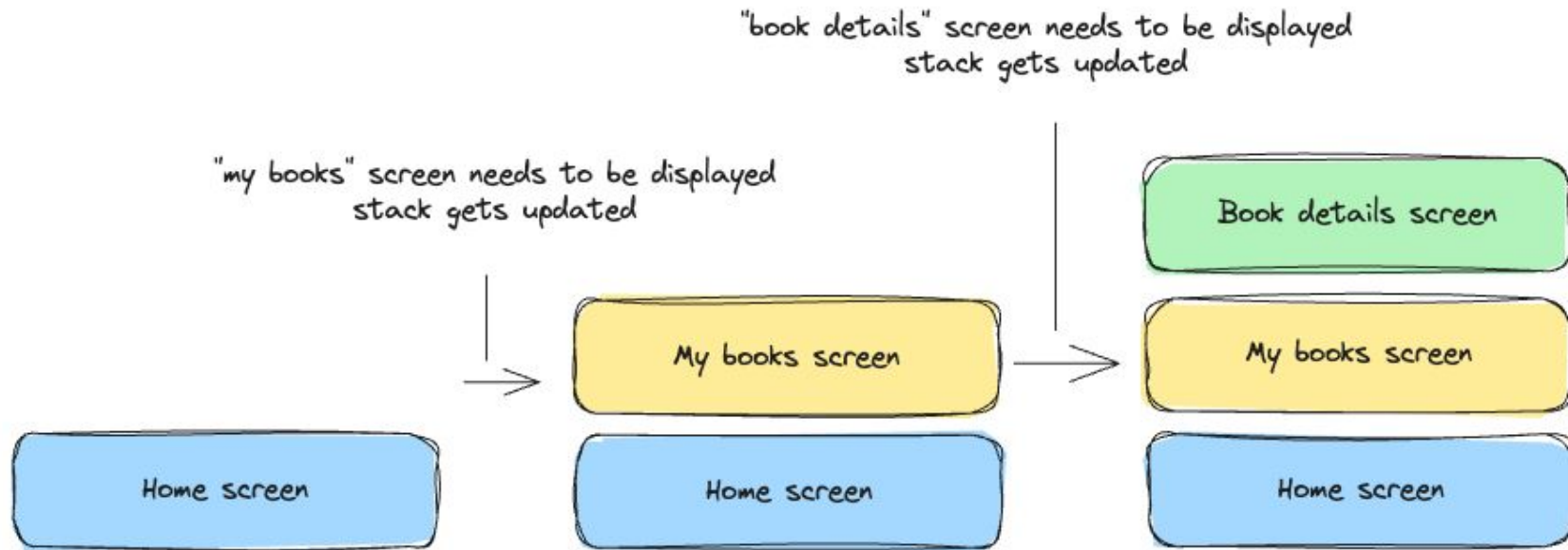
Imperative navigation

1. Start with a home screen
2. Push and pop screens as needed



Declarative navigation

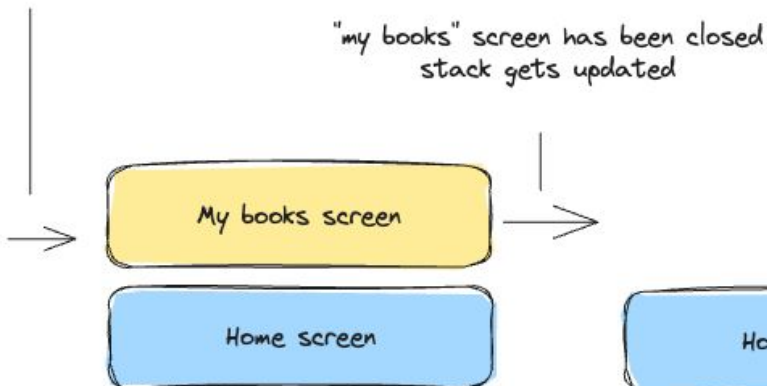
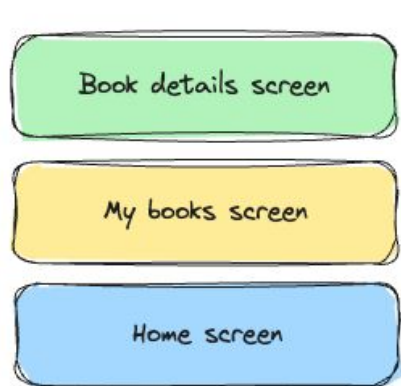
Decide the entire stack of screens whenever something changes



Declarative navigation

Decide the entire stack of screens whenever something changes

"book details" screen has been closed
stack gets updated



"my books" screen has been closed
stack gets updated



back gesture was used on home screen
stack gets updated (the app gets closed)



The difference

Imperative: “push this screen on top, now”

- decides how it's done, when it's done
- describes step by step how to go from state A to B
- think instructions

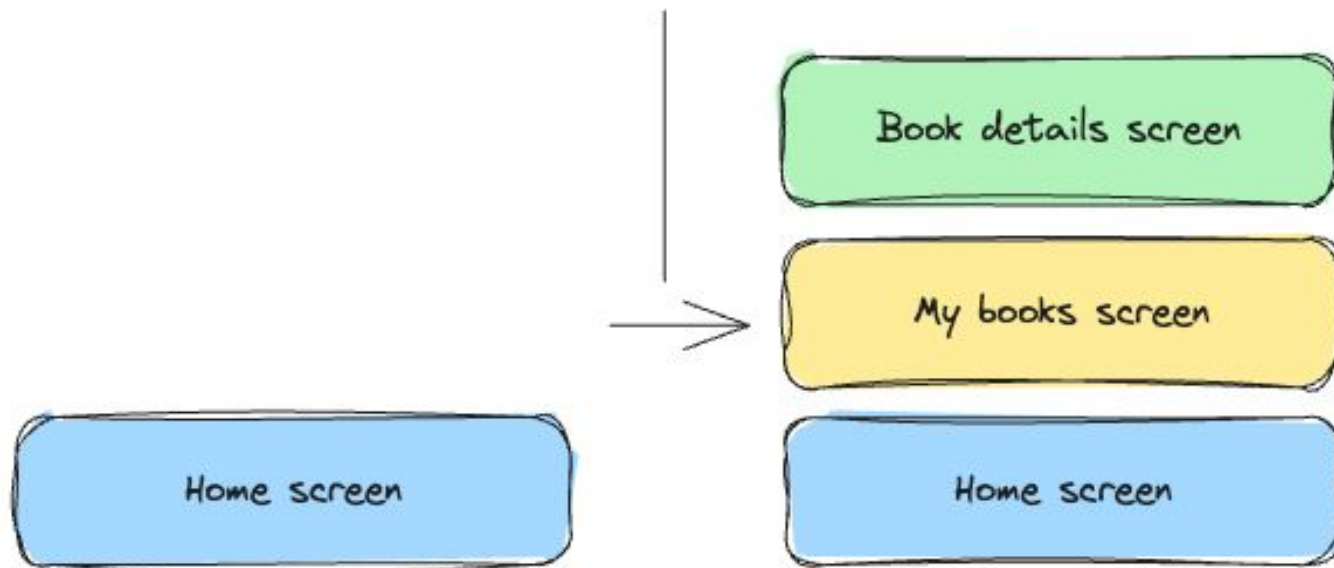
Declarative: “I need this screen to show up”

- only declares the desired result
- describes state A, describes state B
- think configuration

The difference

The imperative way

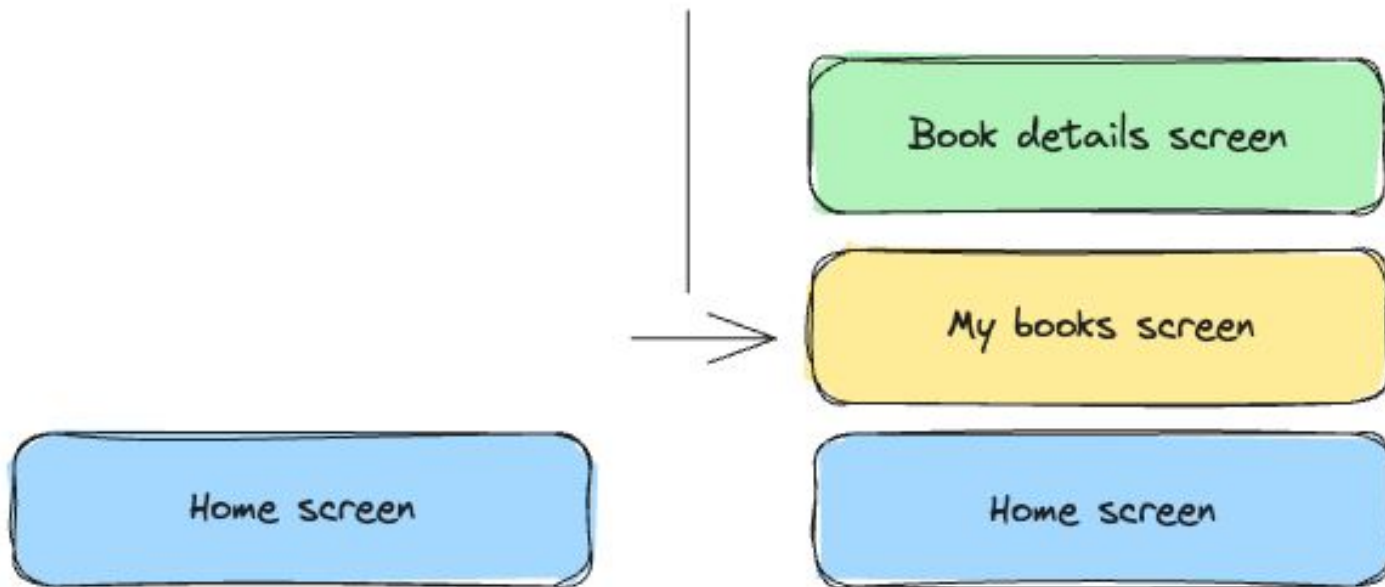
push the "my books" screen
and then push the "book details" screen



The difference

The declarative way

I need the "book details" screen to show up



Tradeoffs

Imperative navigation

- you have to specify each action
- widgets contain a lot of unneeded logic
- it's very plug-and-play
- gets messy at scale
- there might (will) be issues when designing complex processes like auth, multi-screen forms

Declarative navigation

- you only say what you need
- widgets can be dumb(er)
- needs a lot of boilerplate
- easier to manage at scale
- good luck setting it up without prior experience



The navigation APIs

Point of confusion

There are multiple navigation APIs:

- Old imperative navigation
- “New” declarative navigation, Router 2.0
- [go_router | Flutter Package \(pub.dev\)](#) (formerly 3rd party package)
- More options available (e.g. [auto_route | Flutter Package \(pub.dev\)](#))

The builtins

Here's a good in-depth article on the built in options if you're interested

[Learning Flutter's new navigation and routing system | by John Ryan | Flutter | Medium](#)

The Navigator

Navigator

Route stack:

Top route

Another route

...

Bottom route



```
Navigator() // there is always one at the top of the app
MaterialApp() // secretly contains a Navigator
CupertinoApp() // this guy does as well
```

```
// access the navigator from a widget
final navigator = Navigator.of(context);
```


The Navigator

Navigator

Route stack:

Top route

Another route

...

Bottom route

```
// imperative navigation - push
Navigator.of(context).push(
  MaterialPageRoute(
    builder: (context) => const MyBooksScreen(),
  ),
);

// imperative navigation - pop
Navigator.of(context).pop();

// imperative navigation - passing data to the pushed screen
final bookId = '123456';
Navigator.of(context).push(
  MaterialPageRoute(
    builder: (context) => BookDetailsScreen(bookId: bookId),
  ),
);

// imperative navigation - getting data from the pushed screen
final image = await Navigator.of(context).push(
  MaterialPageRoute(
    builder: (context) => const ImagePickerScreen(),
  ),
);

// inside the image picker screen:
final selectedImage = /* ... somehow user selected an image */
// argument to `pop` is the awaited return value from `push`
Navigator.of(context).pop(selectedImage);
```

The Navigator

Navigator

Route stack:

Top route

Another route

...

Bottom route

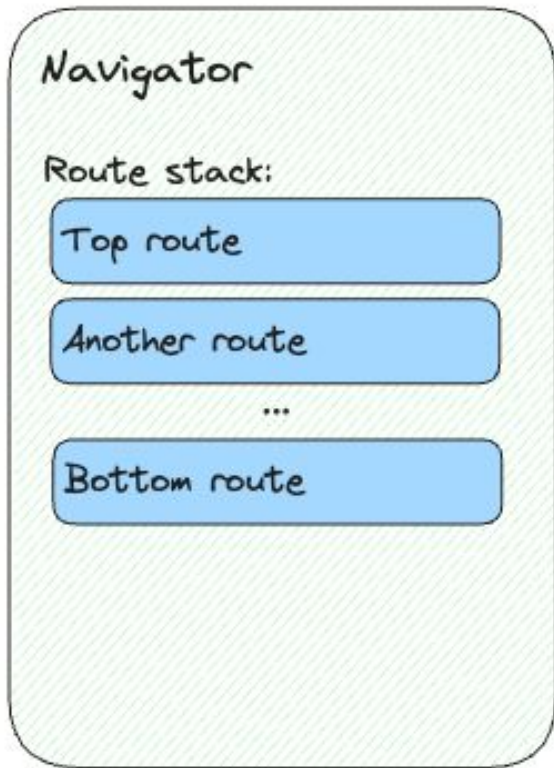
```
// But wait there's more
```

```
Navigator.of(context).push();  
Navigator.of(context).pushReplacement();  
Navigator.of(context).pushAndRemoveUntil();
```

```
Navigator.of(context).pop();  
Navigator.of(context).popUntil();
```

```
Navigator.of(context).replace();  
Navigator.of(context).removeRoute();
```

The Route



- is an entry on Navigator's stack
- describes a screen, a popup, a drawer etc.
- knows how to display itself
- can have an animated transition
- is very abstracted out → use `MaterialPageRoute`

The Route

Navigator

Route stack:

Top route

Another route

...

Bottom route

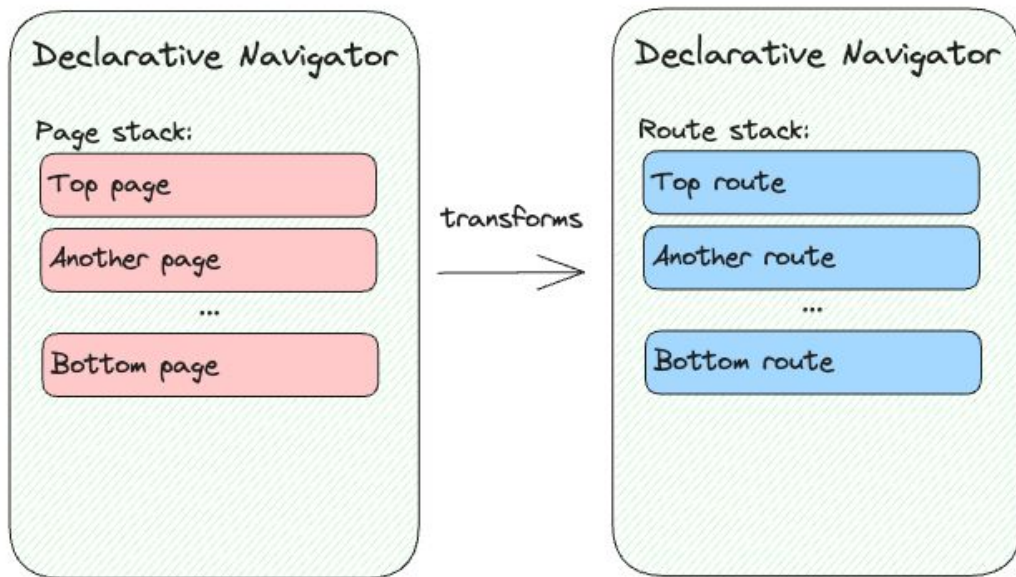
MaterialPageRoute:

- has sensible defaults (material-based)
- only needs one parameter: widget to display (the builder param)

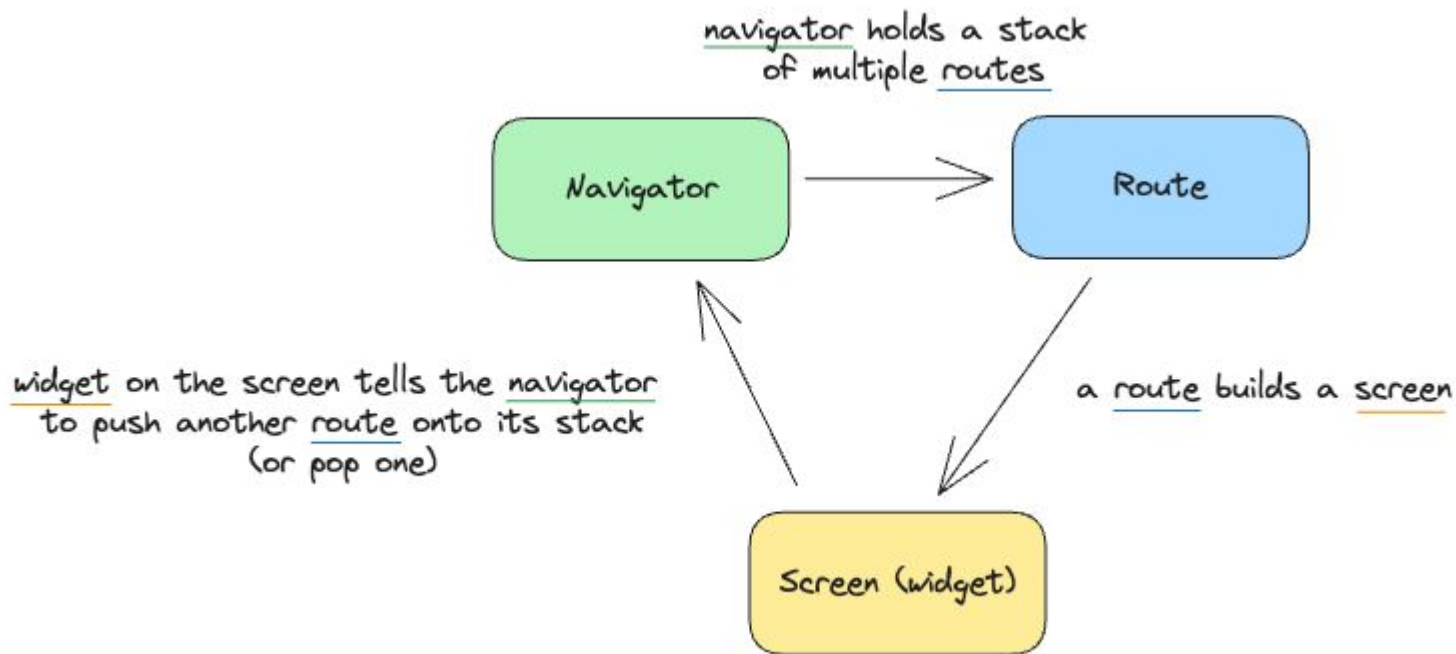
```
MaterialPageRoute(  
  builder: (context) => MyScreen(),  
);  
  
class MyScreen extends StatelessWidget {  
  // ...  
}  
  
Navigator.of(context).push(  
  MaterialPageRoute(  
    builder: (context) => MyScreen(),  
  ),  
);
```

The Page

Page – declarative Router 2.0 only. Don't bother unless using it.
Route had a builder for widgets, Page has a builder for routes



And how they fit together



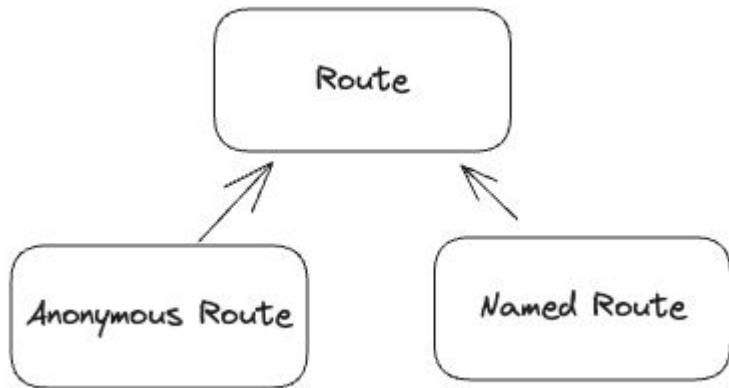
Declarative navigation

This is a complex and low-level subject.

Let's not go into that. Read the article if you're interested.

[Learning Flutter's new navigation and routing system | by John Ryan | Flutter | Medium](#)

Route types



Named routes

```
// Definition at the root of the app
MaterialApp(
  routes: {
    '/': (context) =>
      const HomeScreen(),
    '/my-books': (context) =>
      const MyBooksScreen(),
    '/book-details': (context) =>
      const BookDetailsScreen(),
  }
)
```

```
// Usage - in any widget
Navigator.of(context).pushNamed('/my-books')
```

Named routes

⚠ caution: it's cumbersome to read parameters when building named routes (e.g. the book details screen might need a book id to know which book to display)

```
// Definition at the root of the app
MaterialApp(
  routes: {
    '/': (context) =>
      const HomeScreen(),
    '/my-books': (context) =>
      const MyBooksScreen(),
    '/book-details': (context) =>
      const BookDetailsScreen(),
  }
)
```

go_router (package)

- much more streamlined than using raw Navigator
- declare a hierarchy of routes at the top of the app

```
final router = GoRouter(  
  routes: [  
    GoRoute(  
      path: '/',  
      builder: (context, state) => const HomeScreen(),  
      routes: [  
        // Nesting a route allows to pop it  
        GoRoute(  
          path: 'my-books',  
          builder: (context, state) => const MyBooksScreen(),  
          routes: [  
            GoRoute(  
              path: 'book-details/:bookId',  
              builder: (context, state) => BookDetailsScreen(  
                bookId: state.pathParameters['bookId'],  
              ),  
            ],  
          ),  
        ],  
      ),  
    ],  
  ),  
);
```

go_router



```
// navigate to route (declarative)
// builds a stack of three screens: home, my books and book details
context.go('/my-books/book-details/reading-for-dummies')

// parameter 'bookId' is parsed from path and the following screen is built:
BookDetailsScreen(bookId: 'reading-for-dummies')

// imperative still works: push and pop
context.push('/my-books')
context.pop()
```



Extras

Discover more widgets

- [Flutter Widget of the Week - YouTube](#)
- [Widget catalog | Flutter](#)
- The documentation is rather expansive. Visit the API reference and readmes for in-depth explanations of specific concepts, e.g.
 - [ListView class - widgets library - Dart API \(flutter.dev\)](#)
 - [Scrollable class - widgets library - Dart API \(flutter.dev\)](#)
 - [go_router | Flutter Package \(pub.dev\)](#)
 - [Flex class - widgets library - Dart API \(flutter.dev\)](#)

Guide to flexbox

This is CSS but Flutter's flex model follows the CSS one quite closely.

This is a handy cheat sheet:

[A Complete Guide to Flexbox | CSS-Tricks - CSS-Tricks](#)