Responsive Design (With Unity)

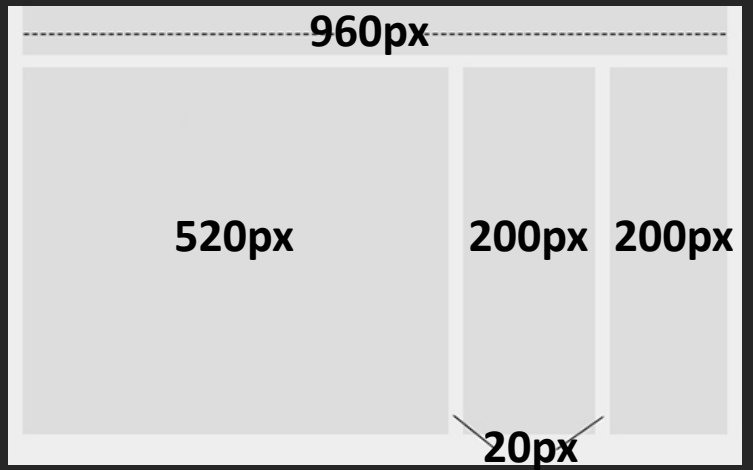
Presented by Lovelle Cardoso

Reflexion Health

Different approaches to layout design

- Fixed
- Fluid
- Adaptive
- Responsive

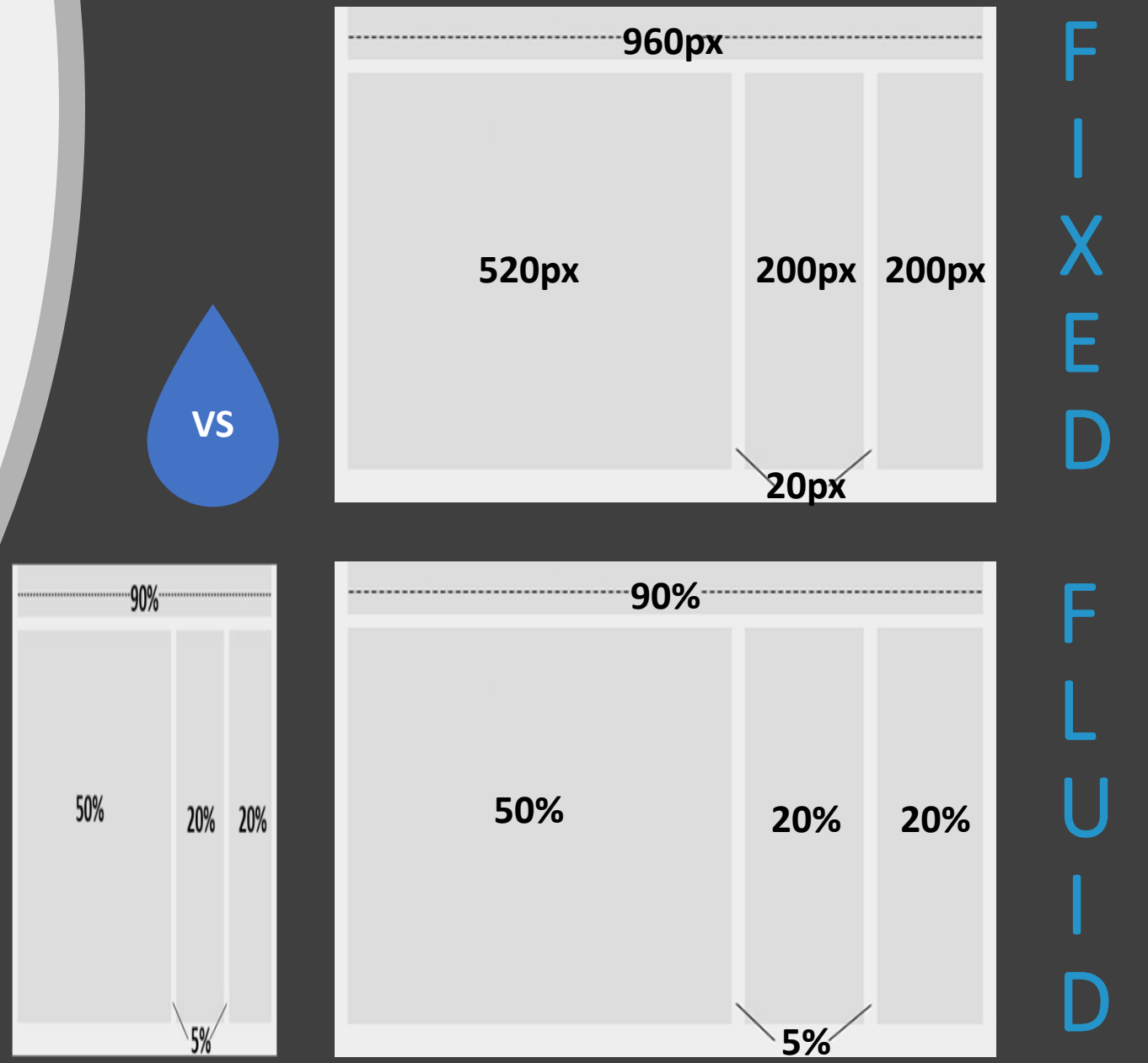
Fixed Design



- Elements have fixed width
- Works well when:
  - only deploying to 1 platform that is guaranteed to only have 1 screen size
- Causes problems when:
  - porting to other devices with different screen sizes and aspect ratios
- Complications when porting a fixed design:
  - Elements may be obscured, cut off, or overlap.
  - Elements may be difficult to see, requiring awkward zooming and scrolling
  - Elements may be difficult to click with fingers

Fluid Design

- Elements have proportional width
- Built using percentage measurements instead of pixel measurements
- Works well when:
  - deploying to platforms that all have similar screen sizes, aspect ratios, and orientation
- Causes problems when:
  - porting to other devices with vastly different screen sizes, aspect ratios, or orientation
- Complications when porting a fluid design:
  - Elements may be squished too much or stretched too far, resulting in an awkward or unusable layout



ADAPTIVE

A different design layout is created for each target platform

RESPONSIVE

One design layout is created with rules that allow it to adapt to each target platform

Adaptive Design

Responsive Design

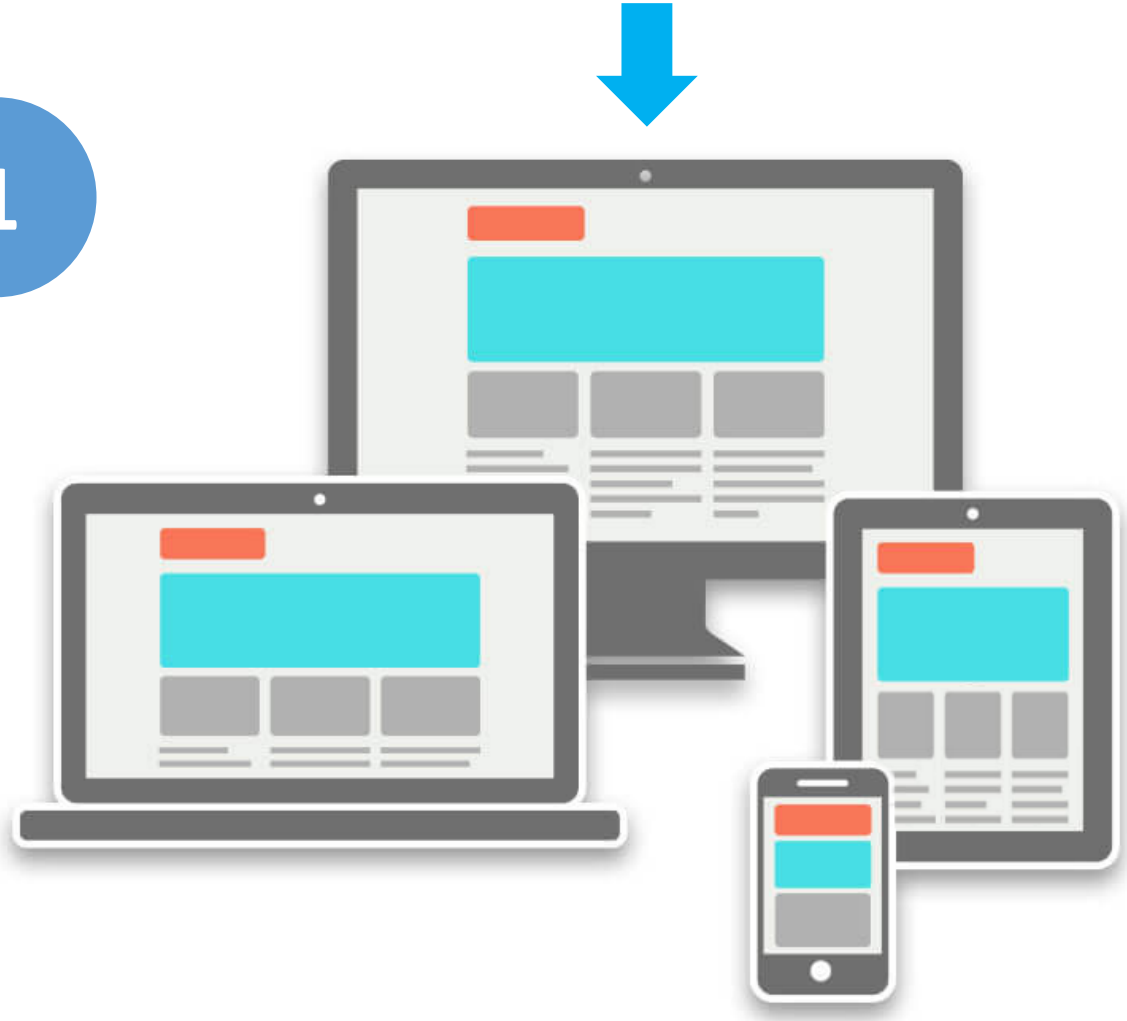
- A different design layout is created for each target platform
- Each layout is designed to be optimal for its specific target platform
- Works well when:
  - Deploying to a reasonably small number of different platforms
  - The variation of screen size within each platform type is small
- Causes problems when:
  - Deploying to a large number of different platforms
  - The variation of screen size within each platform is large
- Complications when porting an adaptive design:
  - Mockup & spec bloat: (takes more time and effort to design and maintain a different layout for every platform)

- One design layout is created with rules that allow it to adapt to each target platform
- Rules define "breakpoints" that determine when the layout should change to a more appropriate layout
- Designing a layout responsively allows a design to look correct, navigate easily, and function well across potentially any platform or screen size.
- It keeps mockup & spec bloat to a minimum since there should only be one design for all platforms, rather than a different design for each.

Designing Responsively

How to create responsive mockups

1



- The primary layout is the one that is most appropriate for the application's most frequently used platform.
- The layout will generally fall into 2 categories: horizontal or vertical
  - When working on a mobile-first application, the primary layout is typically a vertical layout
  - When working on a tv-first, desktop-first, laptop-first, or tablet-first application, the primary layout is typically a horizontal layout
  - NOTE: horizontal and vertical does NOT refer to the orientation of the screen, but rather the orientation of the elements within the screen.
- Create a high-fidelity mockup and a wireframe mockup for the primary layout

How to create a responsive mockup?

Step 1: Primary Layout

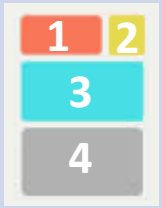
2

- The secondary layout is the opposite of the primary layout
  - When working on a mobile-first application, the secondary layout is typically a horizontal layout
  - When working on a tv-first, desktop-first, laptop-first, or tablet-first application, the secondary layout is typically a vertical layout
- Create a wireframe mockup for the secondary layout

How to create a responsive mockup?

Step 2: Secondary Layout

Example Mockups:

Primary High-Fidelity	Primary Wireframe	Secondary Wireframe
		

- Common Responsive Design Strategies:
  - Horizontal and Vertical switching
  - Collapsing and expansion
  - Encapsulation
  - Padding reduction
  - Pillarboxing (aka Max layout width)

(A secondary high-fidelity mockup can be created as well. But usually it will not be necessary.)

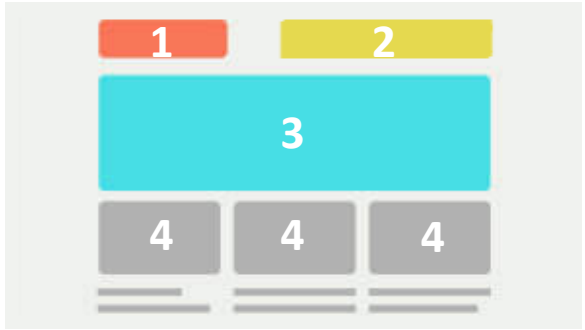
Other benefits of wireframe mockups

- Wireframe mockups are useful when a screen is used in many different contexts with various different configurations
- Wireframe mockups are robust to specific design adjustments and application wide changes
- Wireframe mockups are much more appropriate to refer to when writing official requirements for an application
- Wireframe mockups make it easier for general testing to focus only on the most important aspects of a design and not get bogged down in the details.

Example Layout Specifications:

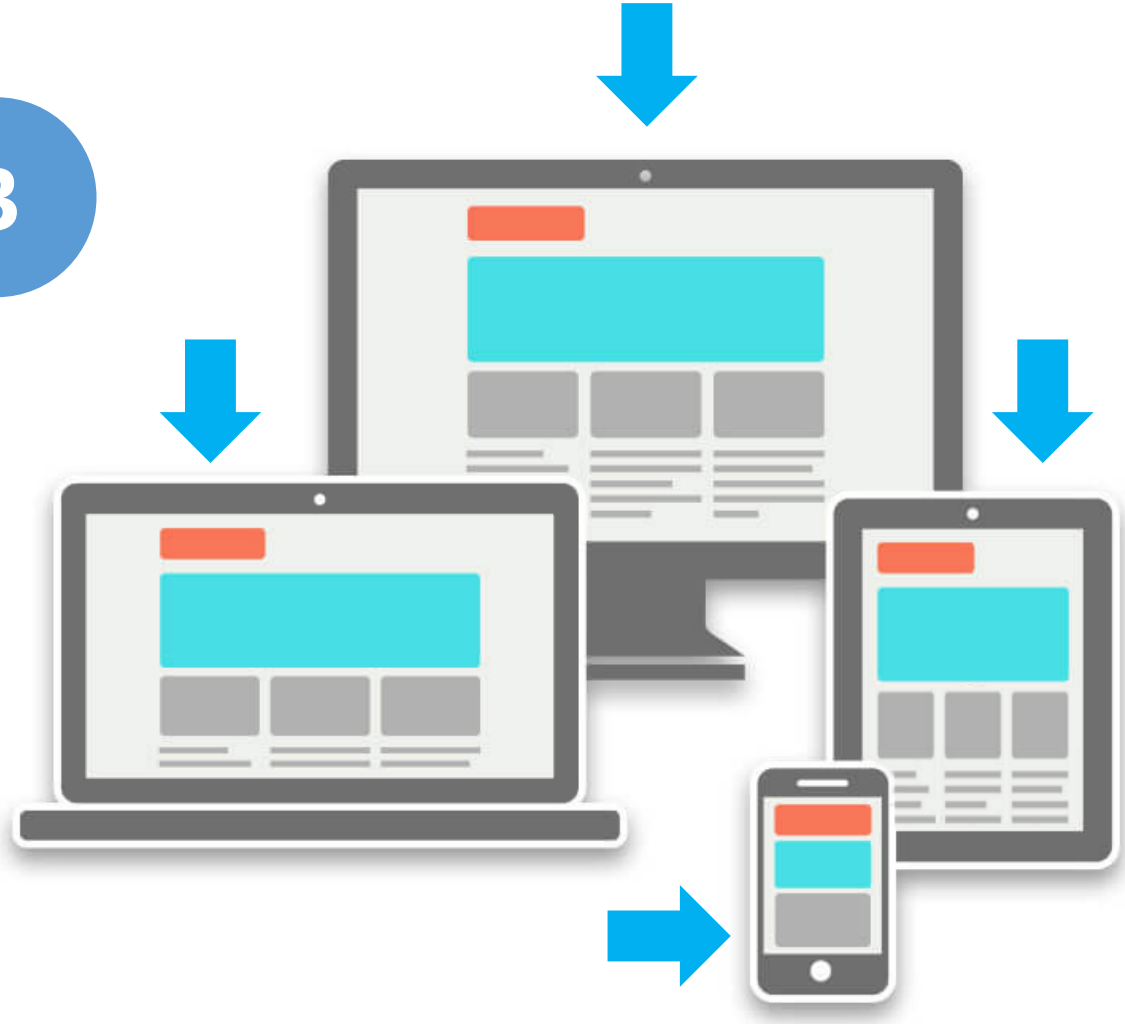
Content	Primary Layout	Secondary Layout
1. Logo • Logo Image 2. Navigation Menu • Option Buttons/Texts (primary) • Menu Icon (secondary) 3. Main Content • Title Text • Subtitle Text • Body Text • Image 4. Secondary Content • Button/Image • Caption Text	1. Logo • Left anchored • Width = 100px 2. Navigation Menu • Right anchored • Width = fit all options • Horizontal layout 3. Main Content • Width = 100% 4. Secondary Content • Width = 1/options each • Horizontal layout	1. Logo • Left anchored • Width = 100px 2. Navigation Menu • Right anchored • Width = 40px 3. Main Content • Width = 100% 4. Secondary Content • Width = 100% • Vertical layout

- Layout specifications do not have to be explicitly written up by designers. They can simply be inferred using the mockup as a reference
  - A designer can correct these inferences if necessary
- Defining universal rules in a separate style guide, rather than repeating them across different screen layout specifications, allows for these specifications to be short and to-the-point
  - Example style guide rules:
    - "The main content of all screens should have:"
      - "at least 80px of padding"
      - "a max width of 2000px"



Reflexion Health

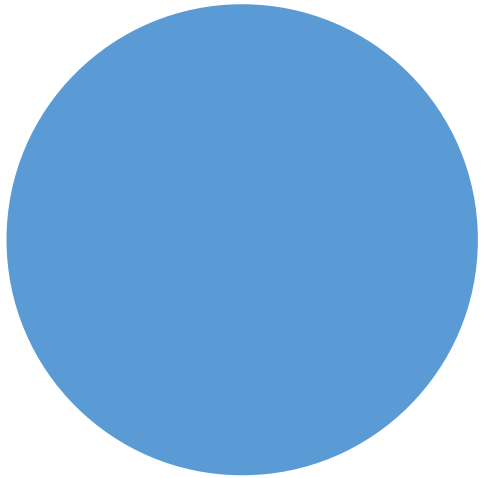
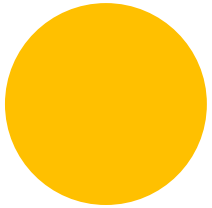
3



- After designing a primary and secondary layout, define breakpoints that will cause shifts from one layout to another
- (Good) Define one universal explicit breakpoint
  - e.g. if the screen width is smaller than 1000 pixels, switch from horizontal to vertical layout
- (Better) Define several explicit breakpoints for various screen elements
  - e.g. if the menu width is smaller than 400 pixels, collapse the menu
- (Best) Define automatic breakpoints for various screen elements
  - e.g. if there isn't enough space to display the menu options horizontally, collapse the menu

How to create a responsive mockup?

Step 3: Breakpoints

Building Responsively

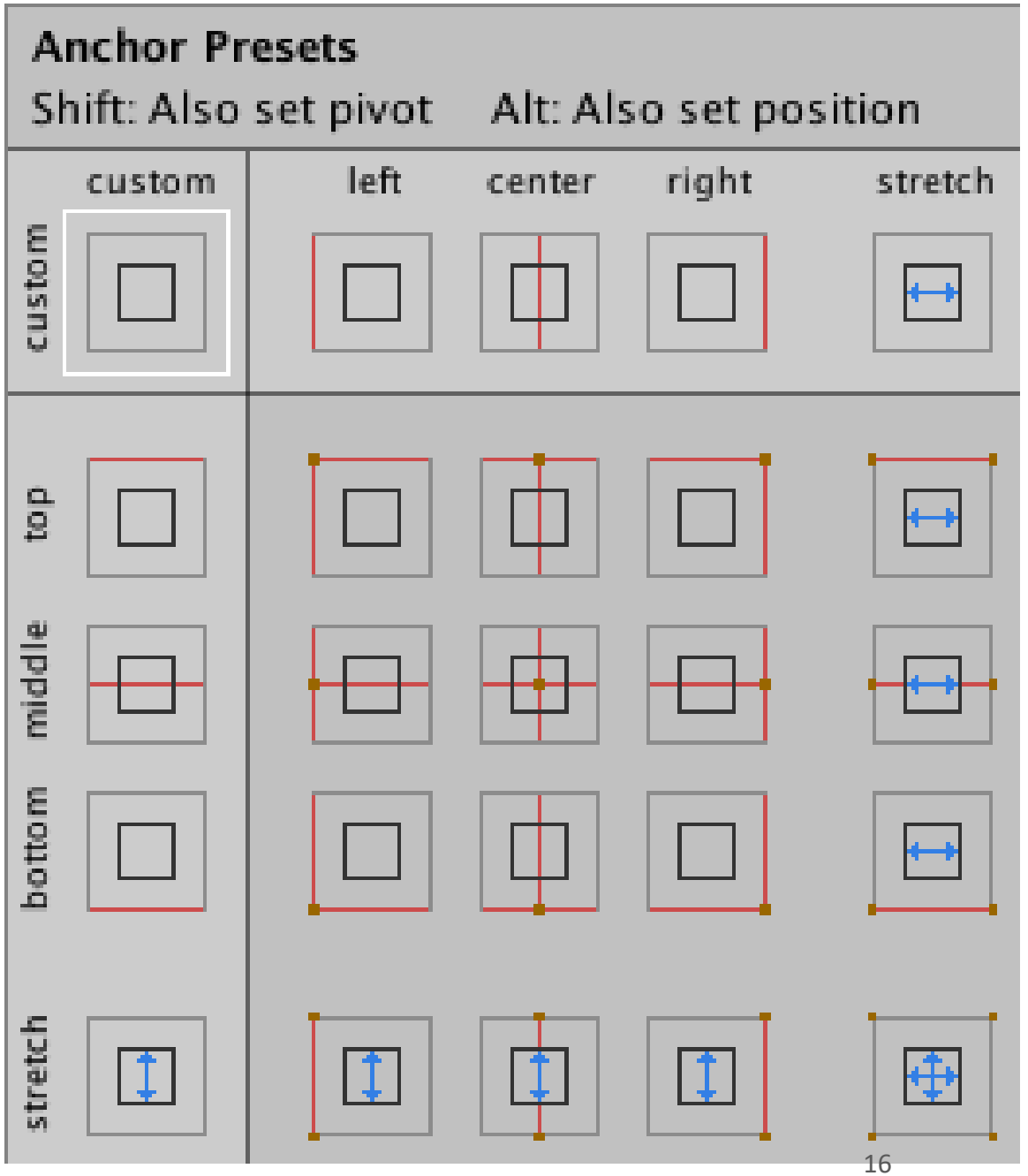
How to build responsive layouts in unity

Unity Layout

How to define layouts in unity

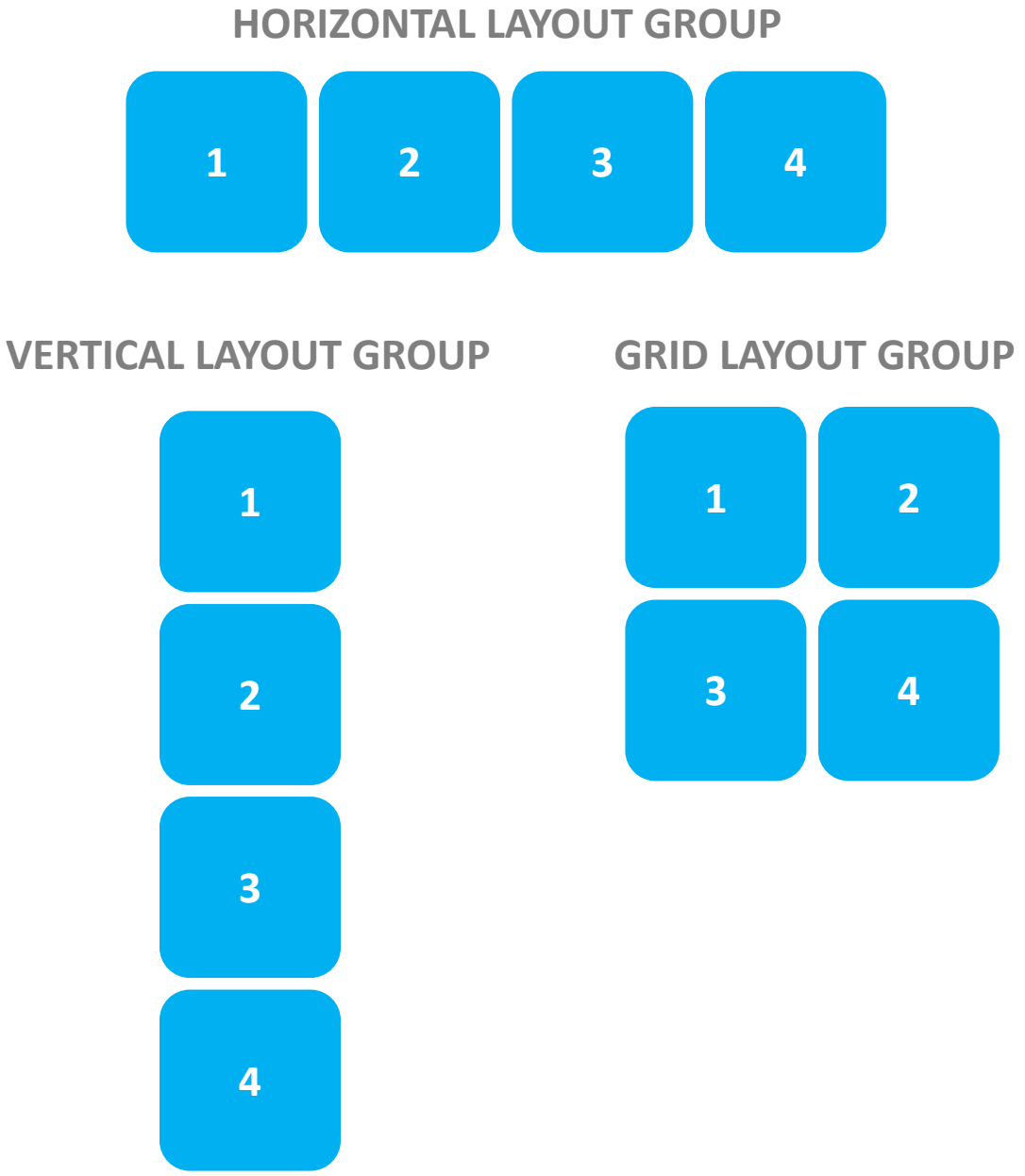
Basic Layout: Anchoring

https://docs.unity3d.com/Manual/UIBasicLayout.html



Auto Layout: Layout Groups

https://docs.unity3d.com/Manual/UIAutoLayout.html

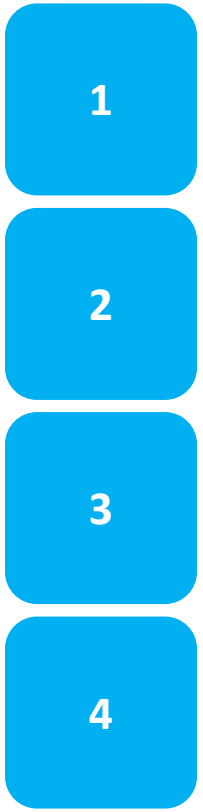


Auto Layout: Responsive & Adaptive Layout Groups

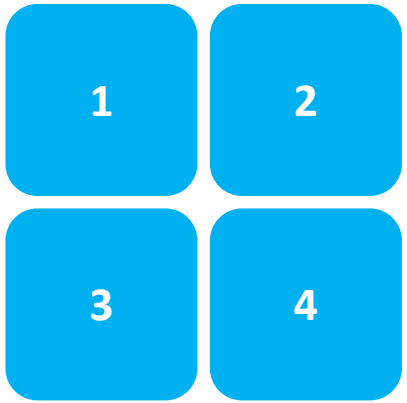
- Common Responsive Design Strategies:
  - (Responsive) Horizontal and Vertical switching
  - (Responsive) Encapsulation
  - (Adaptive) Collapsing and expansion
  - (Adaptive) Padding reduction
  - Pillarboxing (aka Max layout width)

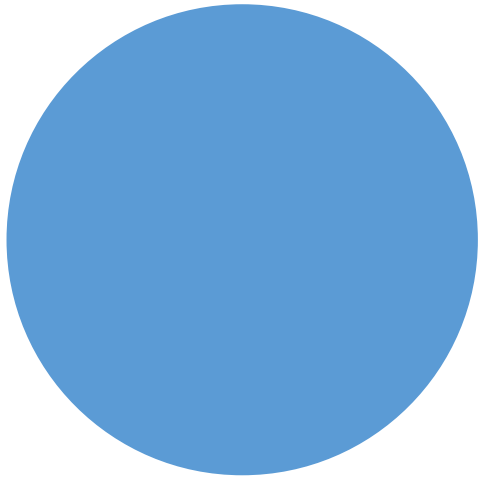
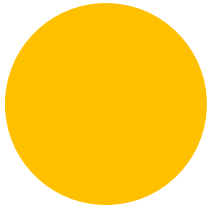
RESPONSIVE LIST LAYOUT GROUP (HORIZONTAL)

RESPONSIVE LIST LAYOUT GROUP (VERTICAL)



RESONSIVE GRID LAYOUT GROUP



Multi-Platform Control Schemes & Navigation Strategies

How to build layouts and design control schemes that work well on different platforms

Unity UI Elements

How to define interactable elements in unity

UI Element Types

- Visual
  - Text
  - Image
- Interactable
  - Button
  - Toggle
  - Toggle Group
  - Slider
  - Dropdown
  - Input Field
  - Scroll Rect
  - Scroll Bars

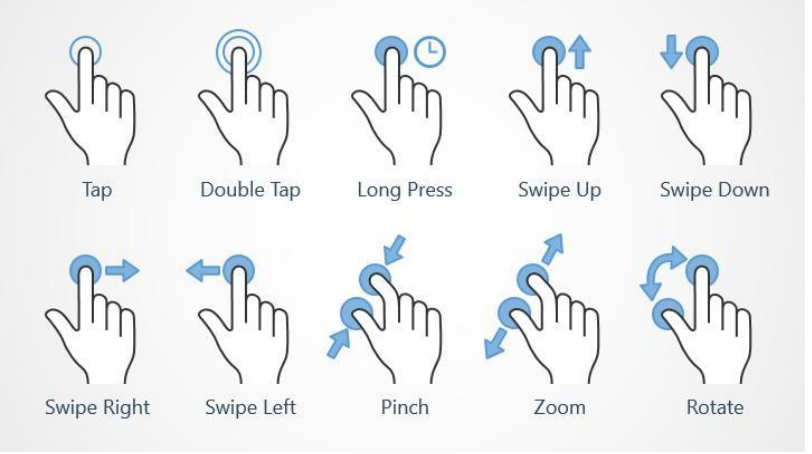
Unity Navigation

How to define navigation in unity

Navigation Controls

	Mouse	Touch	Remote / Controller
Click	Y	Y	Y
Hover	Y	N	Y (Hover = Selection)
Context-click	Y	N	N
Drag/Drop	Y	Y	N
Scroll wheel	Y	N	N
Multi-touch gestures	N	Y	N

Navigation Strategies



- Mouse
  - Most control options, so easiest to design for
  - If the application will be ported to other systems, consider how to adapt to limited control options.
- Touch
  - No easy access to contextual popups or menus triggered by hovers or context-clicks. So contextual info or actions will have to be remapped to different touch gestures or delivered in a different way
    - e.g. long press for hover, double tap for context-click, etc
  - Can leverage single and multi touch gestures to make certain actions easier or feel more natural
    - e.g. Swipe up/down to scroll, Swipe left/right to delete, Pinch to zoom, Spin to rotate, etc
- Remote/controller
  - Minimize the amount of remote/controller button presses and movement as much as possible
  - The path from one button to another is usually defined using automatic pathing algorithms, so buttons should be organized and placed logically on screen to minimize the amount of selectables between elements that are commonly selected one after another
- (Start building for most constrained controls -> then build for least constrained controls)

Scroll Navigation

- Scroll modes:
  - Reveal
  - Align (Top), Align (Middle), Align (Bottom), Align (Left), Align (Center), Align (Right)
- Scroll Limits:
  - Unrestricted, Elastic, Clamped
- Scroll Indication:
  - Scroll bars (interactable scrollbars not recommended for remote or touch control schemes)
  - Scroll direction buttons
  - Scroll alignment
- Scroll controllers:
  - Dragging
  - Scroll wheel
  - Remote directional buttons
  - Scroll buttons