

WTF is BuildContext?

So how exactly a widget lands on a screen?

Widget

“Describes the configuration for an Element.”

```
@protected  
@factory  
Element createElement();
```

StatelessWidget

“A widget that does not require mutable state”

```
@override  
StatelessElement createElement() => StatelessElement(this);
```

StatefulWidget

“A widget that has mutable state”

```
@override  
StatefulElement createElement() => StatefulElement(this);
```

Widgets are immutable



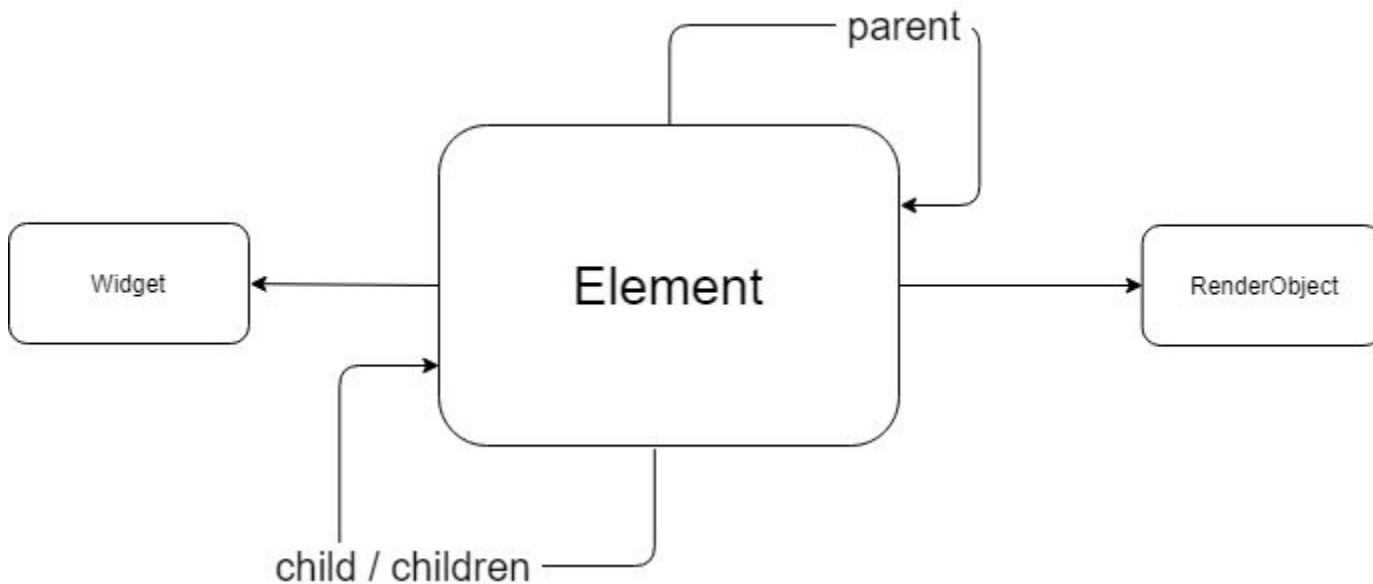
**Widgets can't hold
any reference to the
tree**



**Same Widget can be
used in multiple
subtrees**

Element

“An instantiation of a Widget at a particular location in the tree.”



Element

Element always has a RenderObject at or below its location.

```
RenderObject? get renderObject {  
  RenderObject? result;  
  void visit(Element element) {  
    assert(result == null); // this verifies that there's only one child  
    if (element is RenderObjectElement)  
      result = element.renderObject;  
    else  
      element.visitChildren(visit);  
  }  
  visit(this);  
  return result;  
}
```

**Element could either
be:
ComponentElement
RenderObjectElement**

StatelessElement

```
@override  
Widget build() => widget.build(this);
```

StatefulElement

```
@override  
Widget build() => state.build(this);
```

RenderObject

```
@protected  
void performLayout();
```

```
void paint(PaintingContext context, Offset offset) { }
```

RenderObject

```
@override
void performLayout() {
  final BoxConstraints constraints = this.constraints;
  _resolve();
  assert(_resolvedPadding != null);
  if (child == null) {
    size = constraints.constrain(Size(
      _resolvedPadding!.left + _resolvedPadding!.right,
      _resolvedPadding!.top + _resolvedPadding!.bottom,
    ));
    return;
  }
  final BoxConstraints innerConstraints = constraints.deflate(_resolvedPadding!);
  child!.layout(innerConstraints, parentUsesSize: true);
  final BoxParentData childParentData = child!.parentData! as BoxParentData;
  childParentData.offset = Offset(_resolvedPadding!.left, _resolvedPadding!.top);
  size = constraints.constrain(Size(
    _resolvedPadding!.left + child!.size.width + _resolvedPadding!.right,
    _resolvedPadding!.top + child!.size.height + _resolvedPadding!.bottom,
  ));
}
```

InheritedWidget

```
T? dependOnInheritedWidgetOfExactType<T extends InheritedWidget>(
    {Object? aspect}
)
```

InheritedWidget - Theme example

```
static ThemeData of(BuildContext context) {  
  final _InheritedTheme? inheritedTheme = context.dependOnInheritedWidgetOfExactType<InheritedTheme>();  
  final MaterialLocalizations? localizations = Localizations.of<MaterialLocalizations>(context, MaterialLocalizations);  
  final ScriptCategory category = localizations?.scriptCategory ?? ScriptCategory.latin;  
  final ThemeData theme = inheritedTheme?.theme.data ?? _kFallbackTheme;  
  return ThemeData.localize(theme, theme.typography.geometryThemeFor(category));  
}
```


InheritedWidget - actually it's not like that

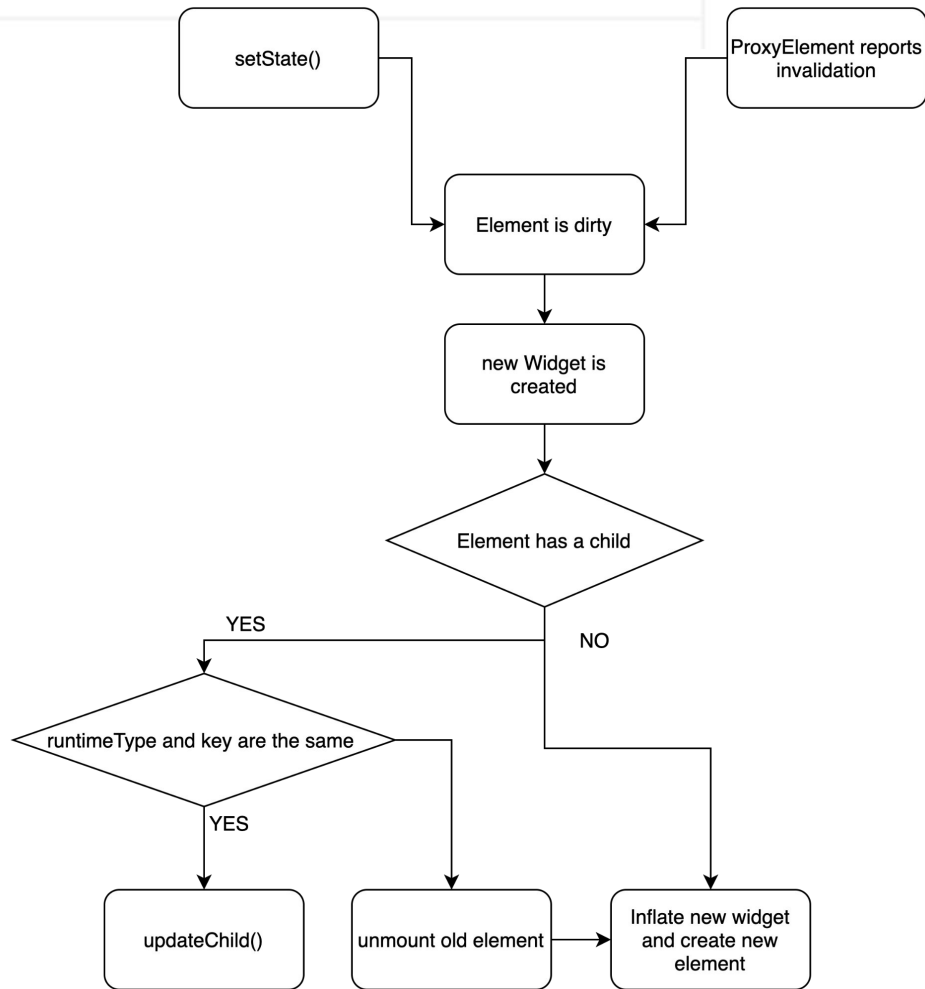
```
@override
T? dependOnInheritedWidgetOfExactType<T extends InheritedWidget>({Object? aspect}) {
  assert(_debugCheckStateIsActiveForAncestorLookup());
  final InheritedElement? ancestor = _inheritedWidgets == null ? null : _inheritedWidgets![T];
  if (ancestor != null) {
    return dependOnInheritedElement(ancestor, aspect: aspect) as T;
  }
  _hadUnsatisfiedDependencies = true;
  return null;
}
```

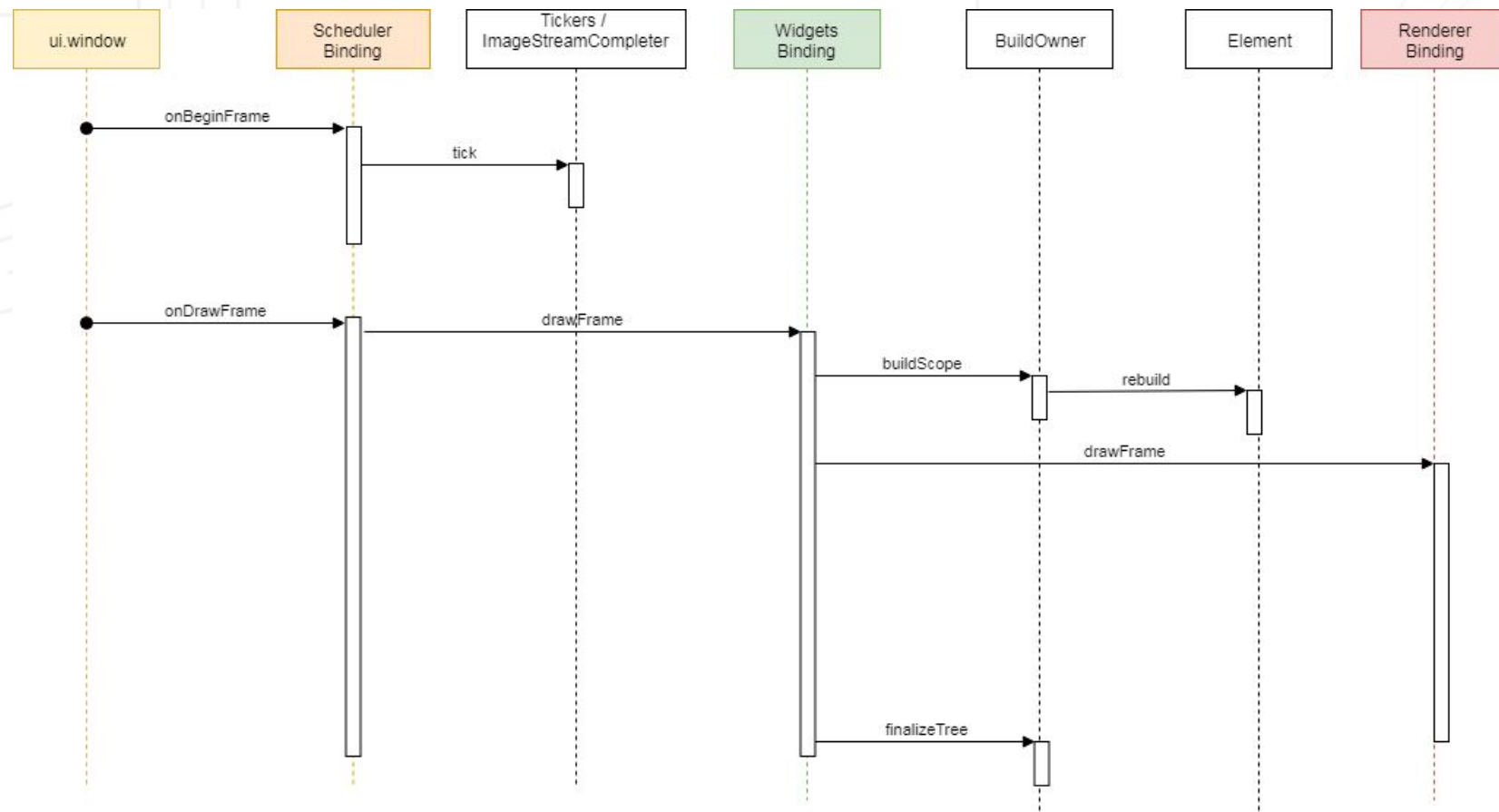
Provider

```
class MyWidget extends StatelessWidget {  
  const MyWidget({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Provider<int>.value(  
      value: 5,  
      child: Column(  
        children: [  
          Text('number is ${Provider.of<int>(context)}'),  
          Text('number is ${context.watch<int>()}'),  
        ],  
      ), // Column  
    ); // Provider.value  
  }  
}
```

So how it works?

```
@protected  
void setState(VoidCallback fn) {  
  final Object? result = fn() as dynamic;  
  _element!.markNeedsBuild();  
}
```





BuildContext is just an Element



Thanks