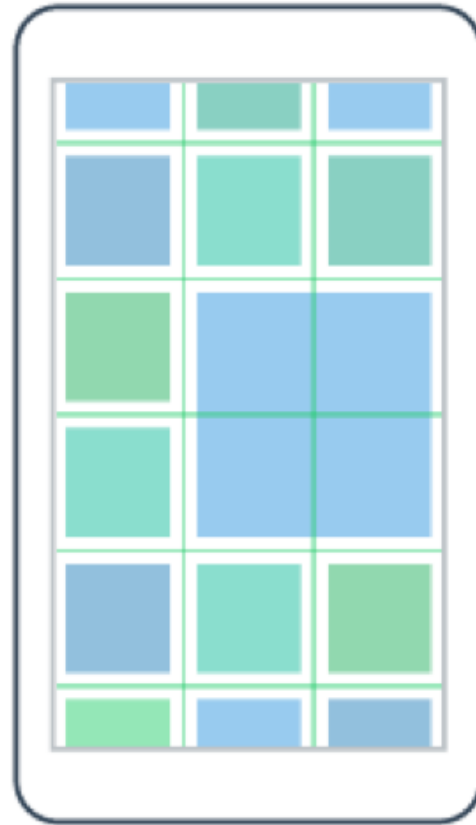


# 31. Organizar eventos usando Grid

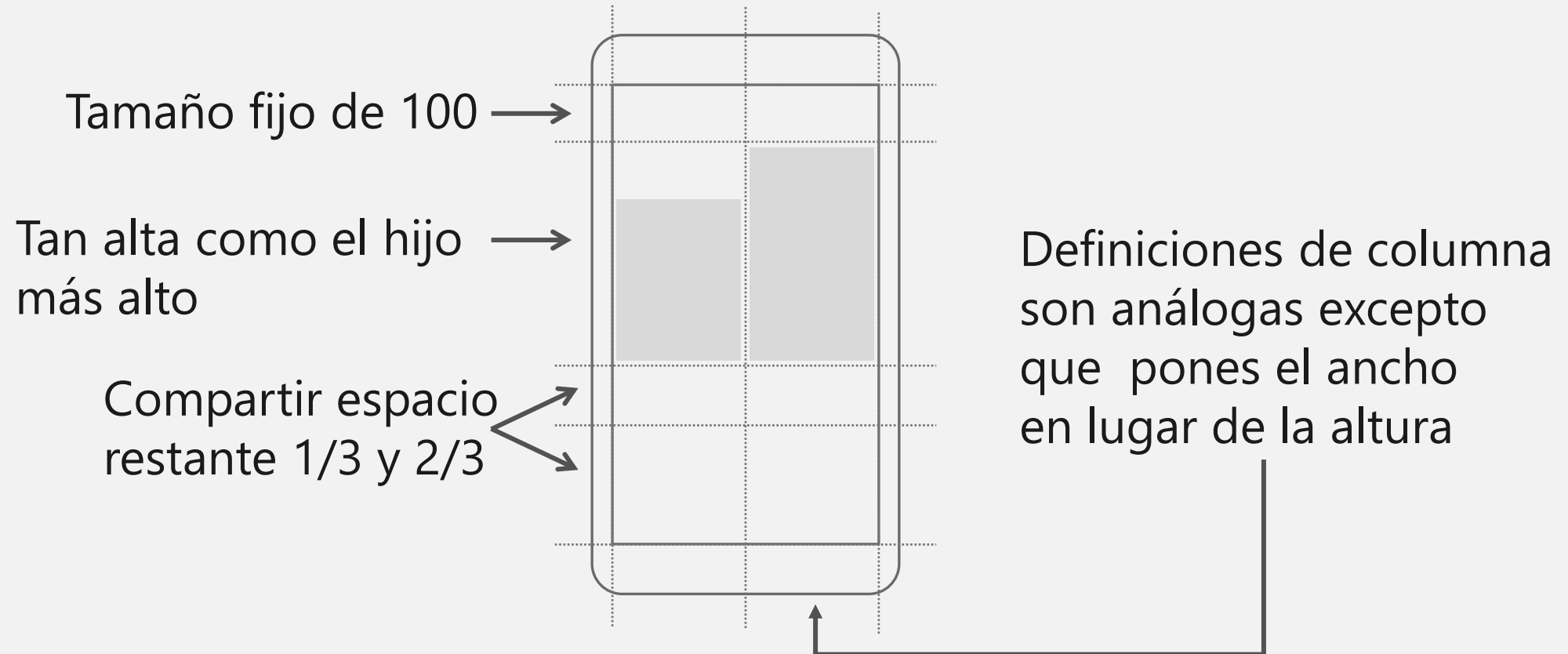
# ¿Qué es el Grid?

Grid coloca a sus hijos en celdas formadas por filas y columnas



# Grid Rows/Columns

Se especifica la forma de la cuadrícula definiendo cada fila y columna individualmente



# Row/Column Definitions

Hay clases dedicadas que definen una fila o una columna

Especifica  
la altura de la  
fila

```
public sealed class RowDefinition : ...  
{  
    ...  
    public GridLength Height { get; set; }  
}
```


Especifica  
el ancho de  
la columna

```
public sealed class ColumnDefinition : ...  
{  
    ...  
    public GridLength Width { get; set; }  
}
```

# ¿Qué es GridLength?

GridLength encapsula dos cosas: unidad y valor

```
public struct GridLength
{
    ...
    public GridUnitType GridUnitType { get; }
    public double Value { get; }
}
```



Las unidades pueden ser: Absolute, Auto, Star

# Absolute GridLength

Absolute GridLength especifica una altura de fila fija o un ancho de columna

```
var row = new RowDefinition() { Height = new GridLength(100) };
```

```
<RowDefinition Height="100" />
```



El valor está en  
unidades  
Independientes de  
la plataforma

# Auto GridLength

Auto GridLength permite que la altura de la fila o el ancho de la columna se adapten, automáticamente se convierte en el tamaño del niño más grande

```
var row = new RowDefinition() {Height = new GridLength(1, GridUnitType.Auto)};
```

```
<RowDefinition Height="Auto" />
```

↑  
El valor es irrelevante para Auto,  
es típico usar 1 como el valor  
al crear en código


# Star GridLength

Star GridLength comparte el espacio disponible proporcionalmente entre todas las filas/columnas que utilizan tamaño de estrella

```
var row = new RowDefinition() { Height = new GridLength(2.5, GridUnitType.Star) };
```

```
<RowDefinition Height="2.5*" />
```

Nota: "1\*" y "\*" son equivalentes en XAML.





# Colecciones Grid Row/Column

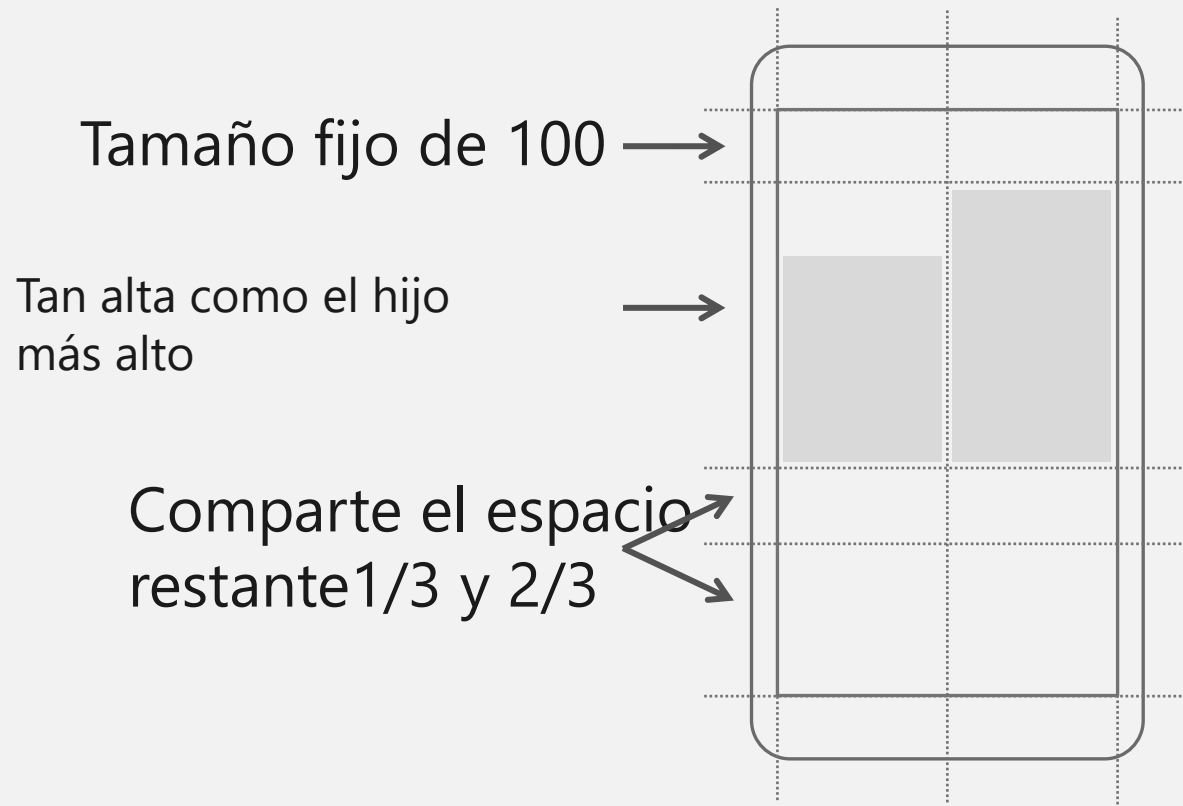
Grid contiene colecciones para las definiciones de filas y columnas

```
public partial class Grid : Layout<View>
{
    ...
    public ColumnDefinitionCollection ColumnDefinitions { get; set; }
    public RowDefinitionCollection RowDefinitions { get; set; }
}
```

↑  
Agrega elementos a estas colecciones  
para crear las filas/columnas

# Ejemplo Grid

Es común mezclar diferentes configuraciones de GridLength en la misma cuadrícula



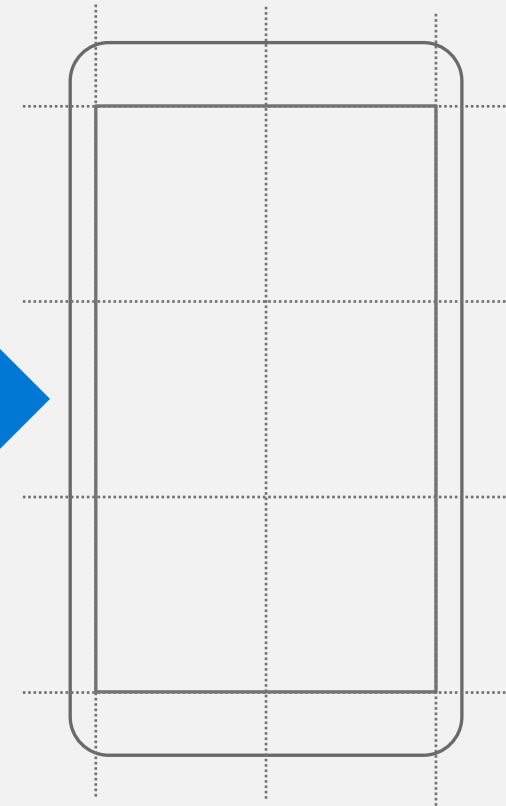
```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="100" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="1*" />
    <RowDefinition Height="2*" />
  </Grid.RowDefinitions>
  ...
</Grid>
```

# Tamaño por defecto

El valor por defecto de Rows y columns es "1\*"

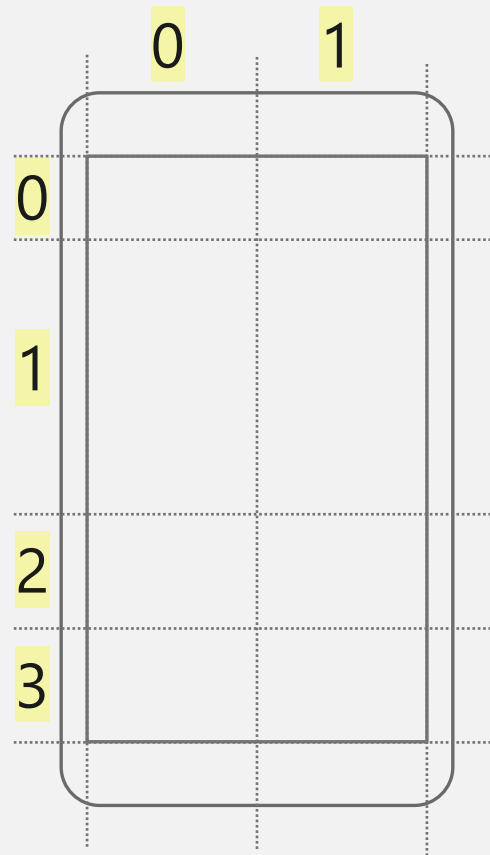
```
<Grid>  
  <Grid.RowDefinitions>  
    <RowDefinition />  
    <RowDefinition />  
    <RowDefinition />  
  </Grid.RowDefinitions>  
  
  <Grid.ColumnDefinitions>  
    <ColumnDefinition />  
    <ColumnDefinition />  
  </Grid.ColumnDefinitions>  
  ...  
</Grid>
```

Crear un grid  
uniforme de 3x2



# Numeración de Row/Column

La numeración de filas/columnas comienza en 0



# Propiedades de posicionamiento del Grid

Grid define cuatro propiedades adjuntas que se utilizan para posicionar a los hijos.

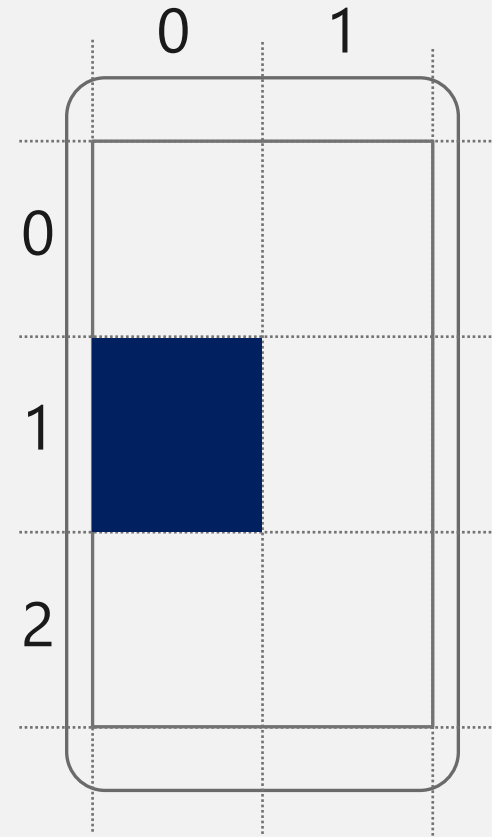
ATTACHED PROPERTY	VALUE
Column	An integer that represents the Column in which the item will appear.
ColumnSpan	An integer that represents the number of Columns that the item will span.
Row	An integer that represents the row in which the item will appear.
RowSpan	An integer that represents the number of rows that the item will span.

# Cell Specification

Aplicar las propiedades adjuntas de Row y Column a cada hijo

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <BoxView Grid.Row="1" Grid.Column="0"
    BackgroundColor="Navy" />
</Grid>
```

Establece  
Fila/columna



# Span

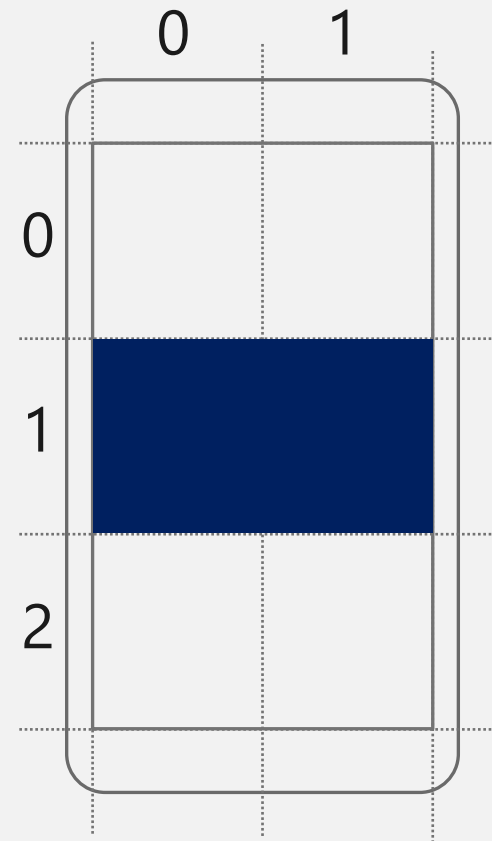
Aplique RowSpan y ColumnSpan a cada niño según sea necesario.

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>

  <BoxView Grid.Row="1" Grid.Column="0"
    Grid.ColumnSpan="2"
    BackgroundColor="Navy" />

</Grid>
```

Establece  
el  
span

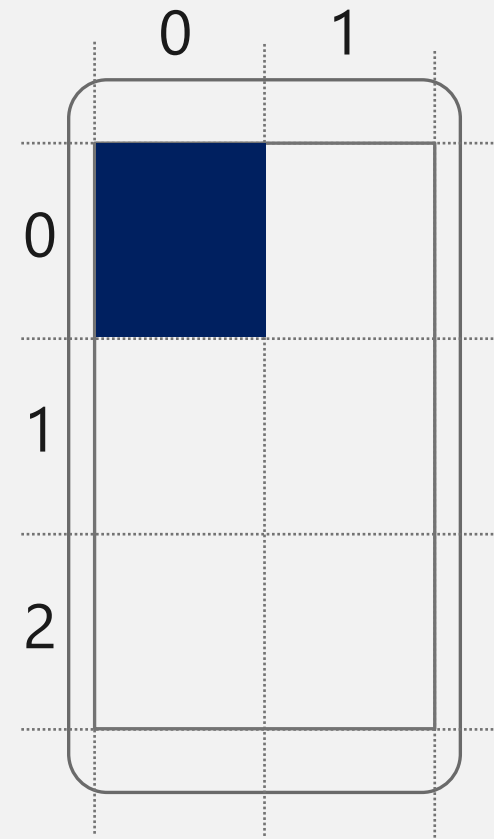
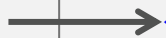


# Celdas y Span, valores por defecto

Las ubicaciones de las celdas están predeterminadas en 0 y Span están predeterminados en 1

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <BoxView BackgroundColor="Navy" />
</Grid>
```

Celda  
(0,0)



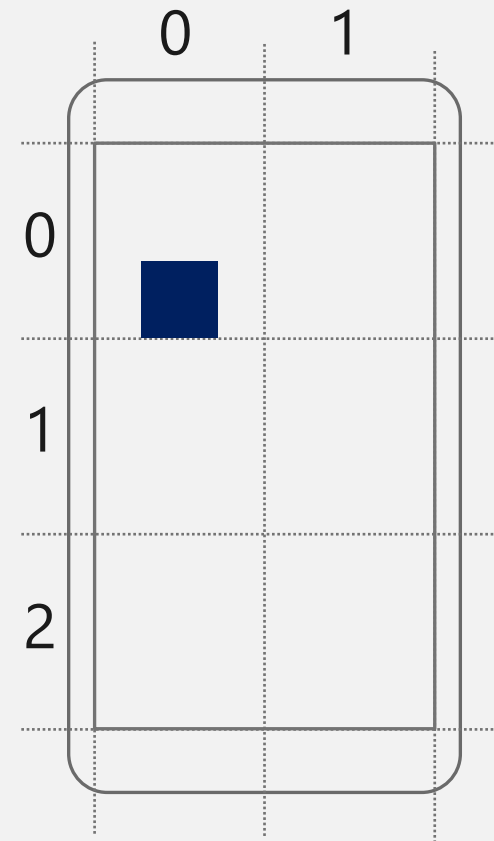


# Opciones de Layout

Las opciones de diseño horizontal y vertical de una vista controlan el tamaño y la posición dentro de su celda en la cuadrícula (el valor predeterminado es Fill)

Posición  
dentro de  
la celda

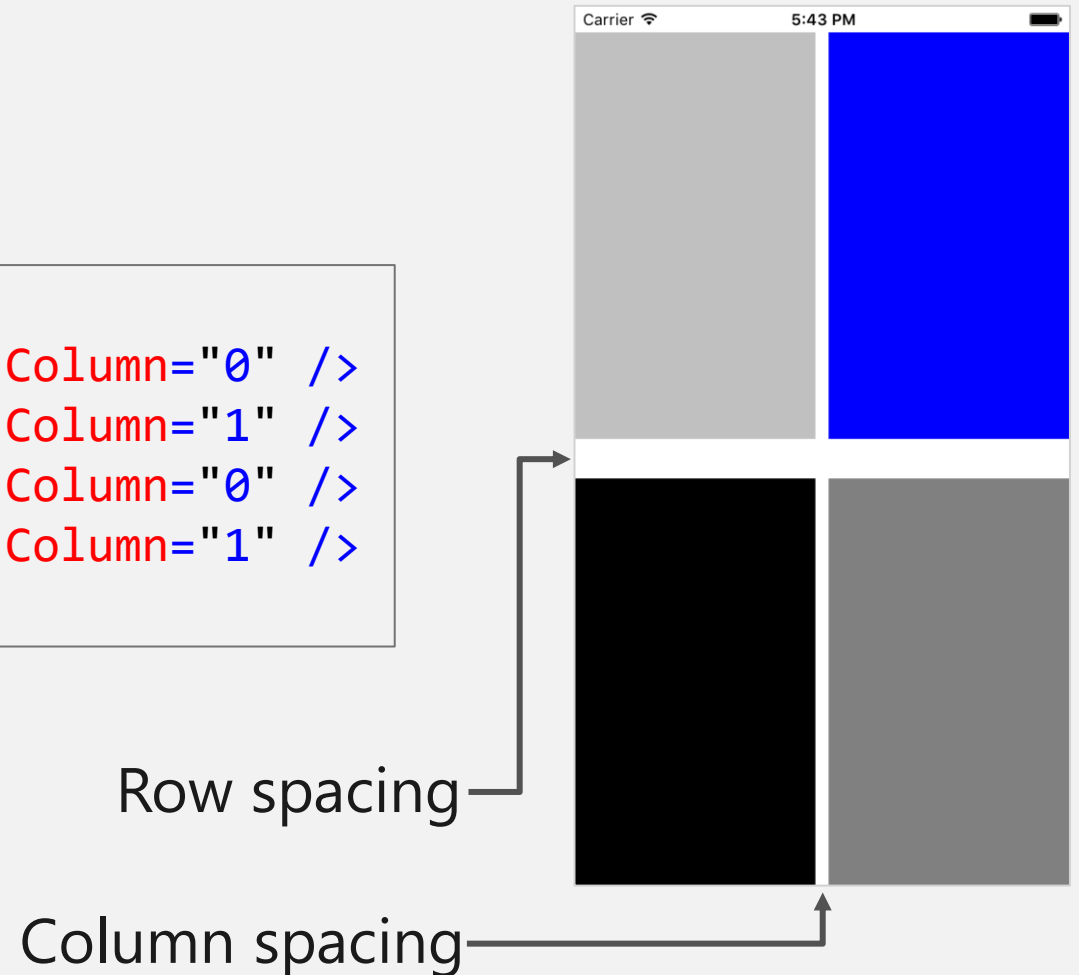
```
<Grid>
  ...
  <BoxView HorizontalOptions="Center"
            VerticalOptions="End"
            BackgroundColor="Navy"
            WidthRequest="50"
            HeightRequest="50" />
</Grid>
```



# Grid Child Spacing

Las propiedades RowSpacing y ColumnSpacing de Grid separan a los elementos secundarios (ambos tienen un valor predeterminado de 6)

```
<Grid RowSpacing="30" ColumnSpacing="10">  
  <BoxView Color="Silver" Grid.Row="0" Grid.Column="0" />  
  <BoxView Color="Blue" Grid.Row="0" Grid.Column="1" />  
  <BoxView Color="Black" Grid.Row="1" Grid.Column="0" />  
  <BoxView Color="Gray" Grid.Row="1" Grid.Column="1" />  
</Grid>
```



Propiedades adjuntas

# ¿Qué es una propiedad adjunta?

Una propiedad adjunta es una propiedad que se define en una clase pero se establece en objetos de otros tipos

**Button** no cuentan con propiedades **Row/Column** →



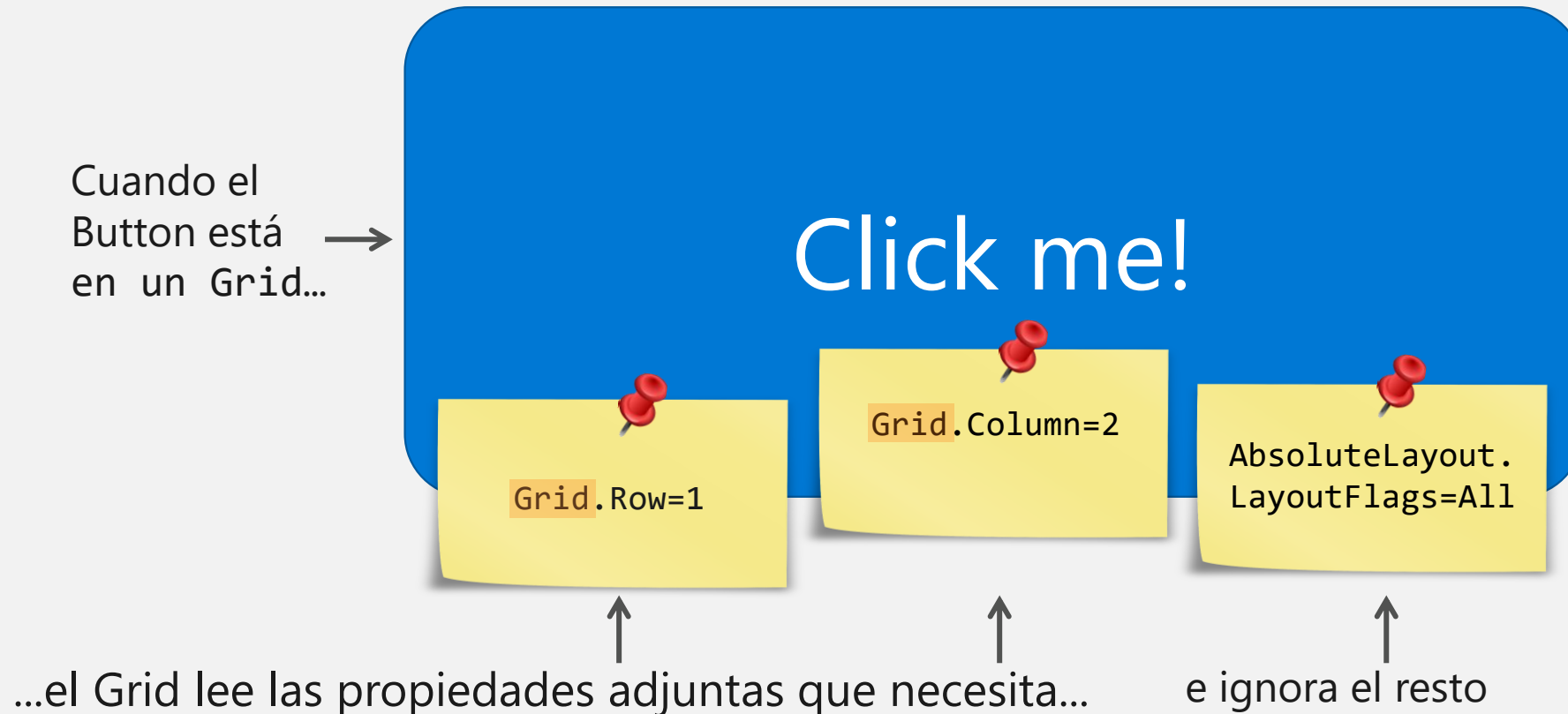
Se definen en **Grid** y unido a los objetos de otros tipos según sea necesario →

`Grid.Row=1`

`Grid.Column=2`

# ¿Quién puede usar una propiedad adjunta?

Por lo general, un Layout buscará propiedades adjuntas en sus elementos secundarios.



# Aplicar una propiedad adjunta

En XAML, use el nombre de la clase propietaria y el nombre de la propiedad adjunta (sin el sufijo Property)

Adjunta la  
configuración  
de fila y columna  
al Button

→ `<Button Grid.Row="1" Grid.Column="2" ... />`

```
public partial class Grid : Layout<View>
{
    ...
    public static readonly BindableProperty RowProperty = BindableProperty.CreateAttached(...);

    public static int GetRow(BindableObject bindable) { ... }
    public static void SetRow(BindableObject bindable, int value) { ... }
}
```