


synectique
Inventive Analysis

Inria

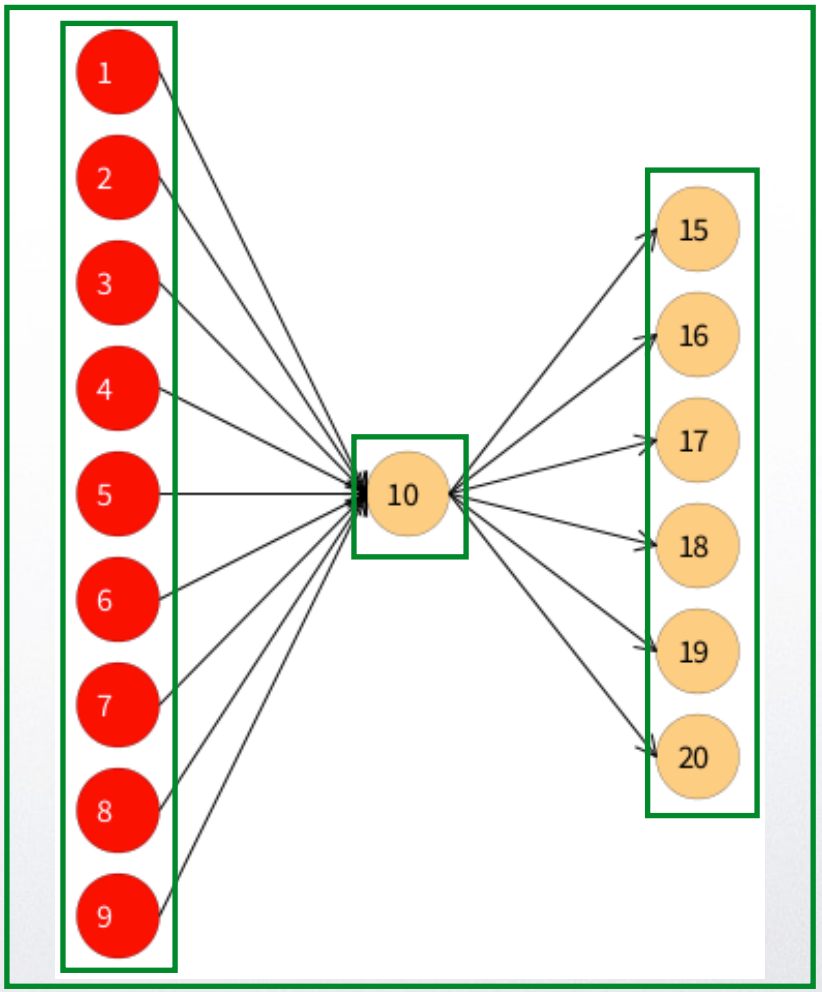
Telescope:

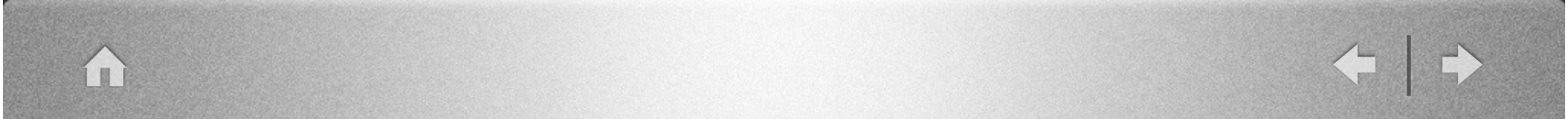
A High-Level Model to Build Dynamic Visualizations

Guillaume Larchevêque, Usman Bhatti,
Nicolas Anquetil, Stéphane Ducasse


A visualization

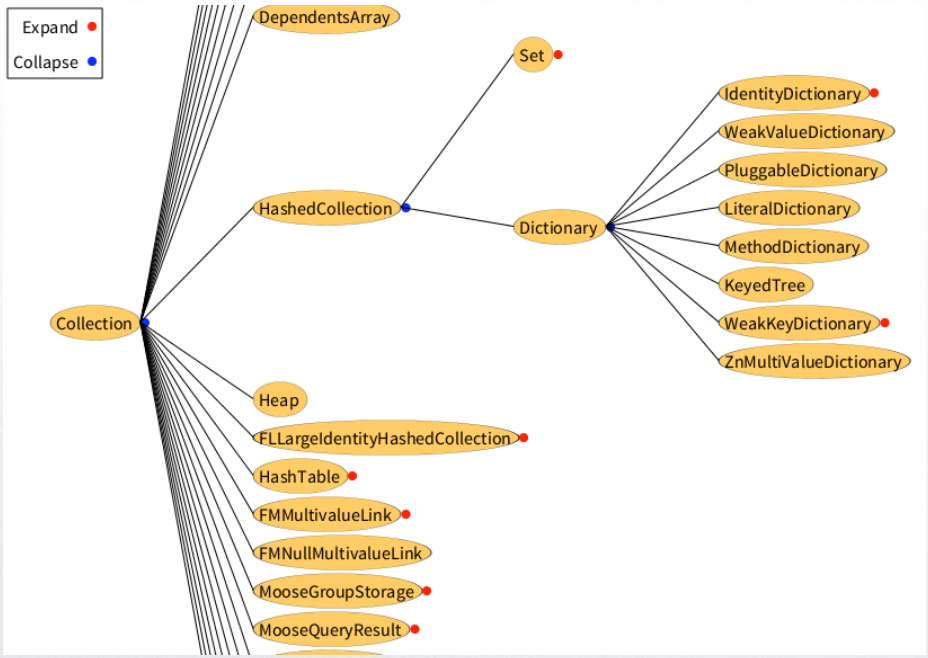
- groups
- layout
- nodes
- connections
- style
- interactions
- legend

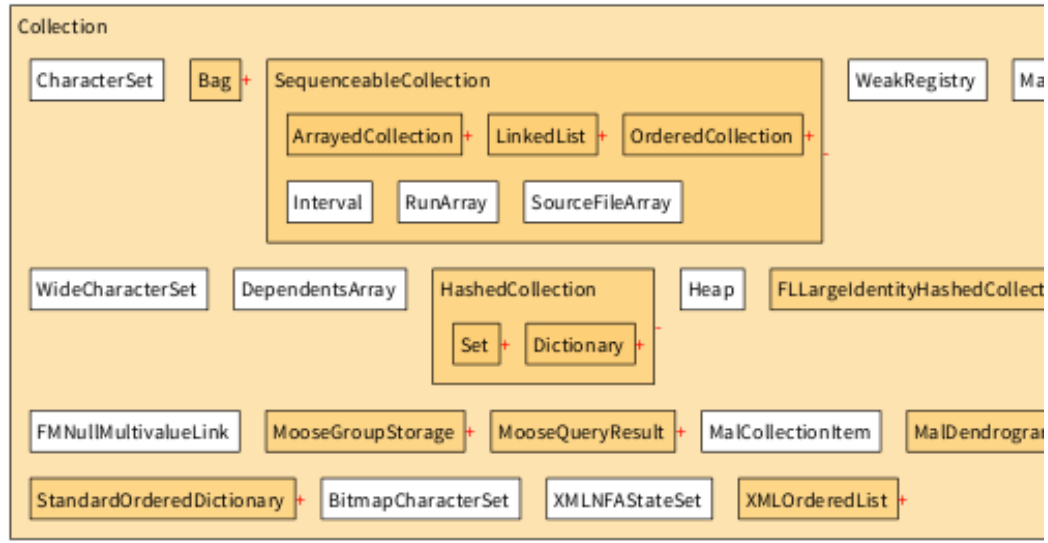




Telescope

- A visualization framework
- Built on top of Roassal






Why Telescope?

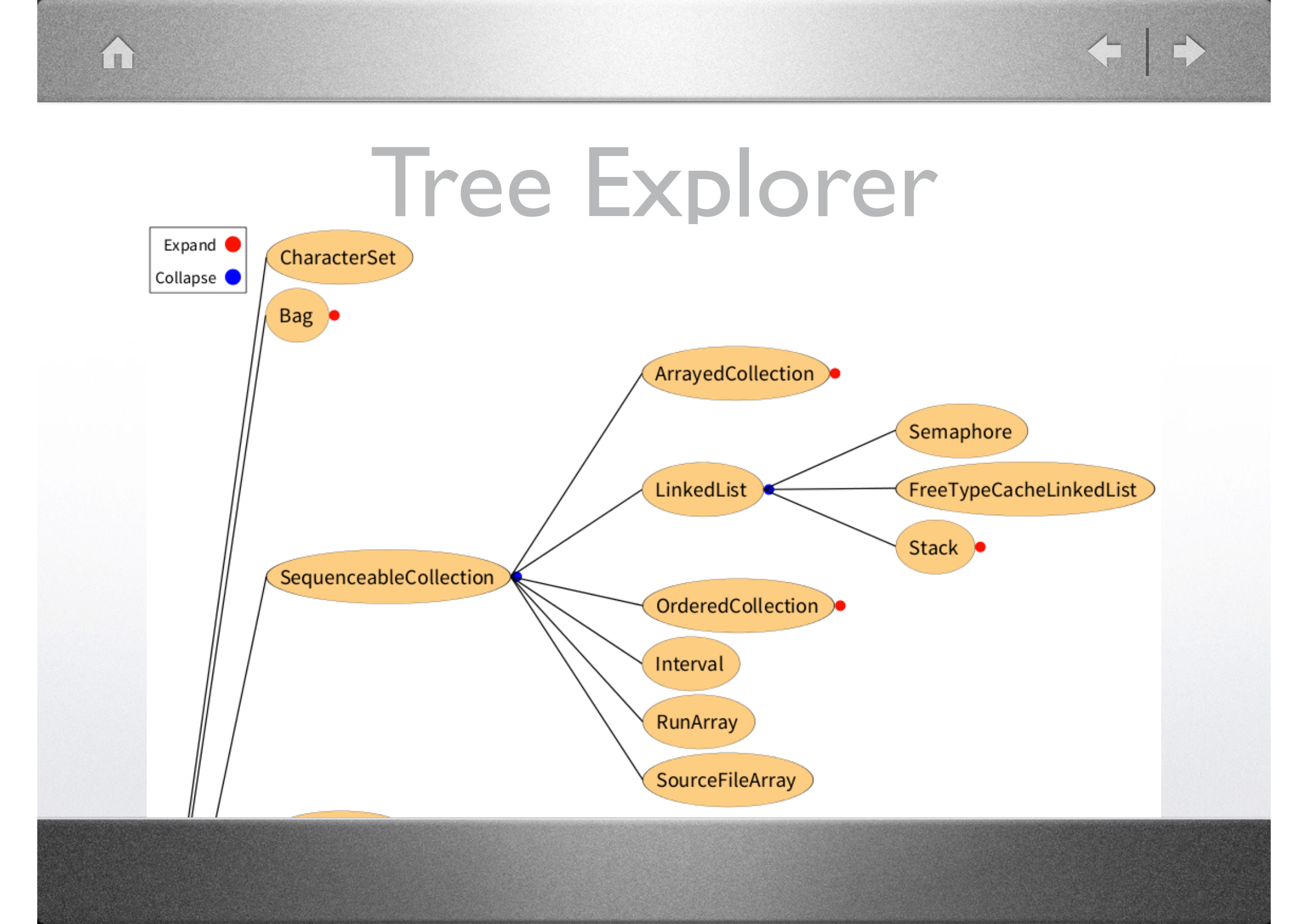
- Roassal is cool but requires skills
- Capitalize experts knowledge
- Provide high-level mechanisms
- Capture the visualization domain
- Need a stable API


Newbie users

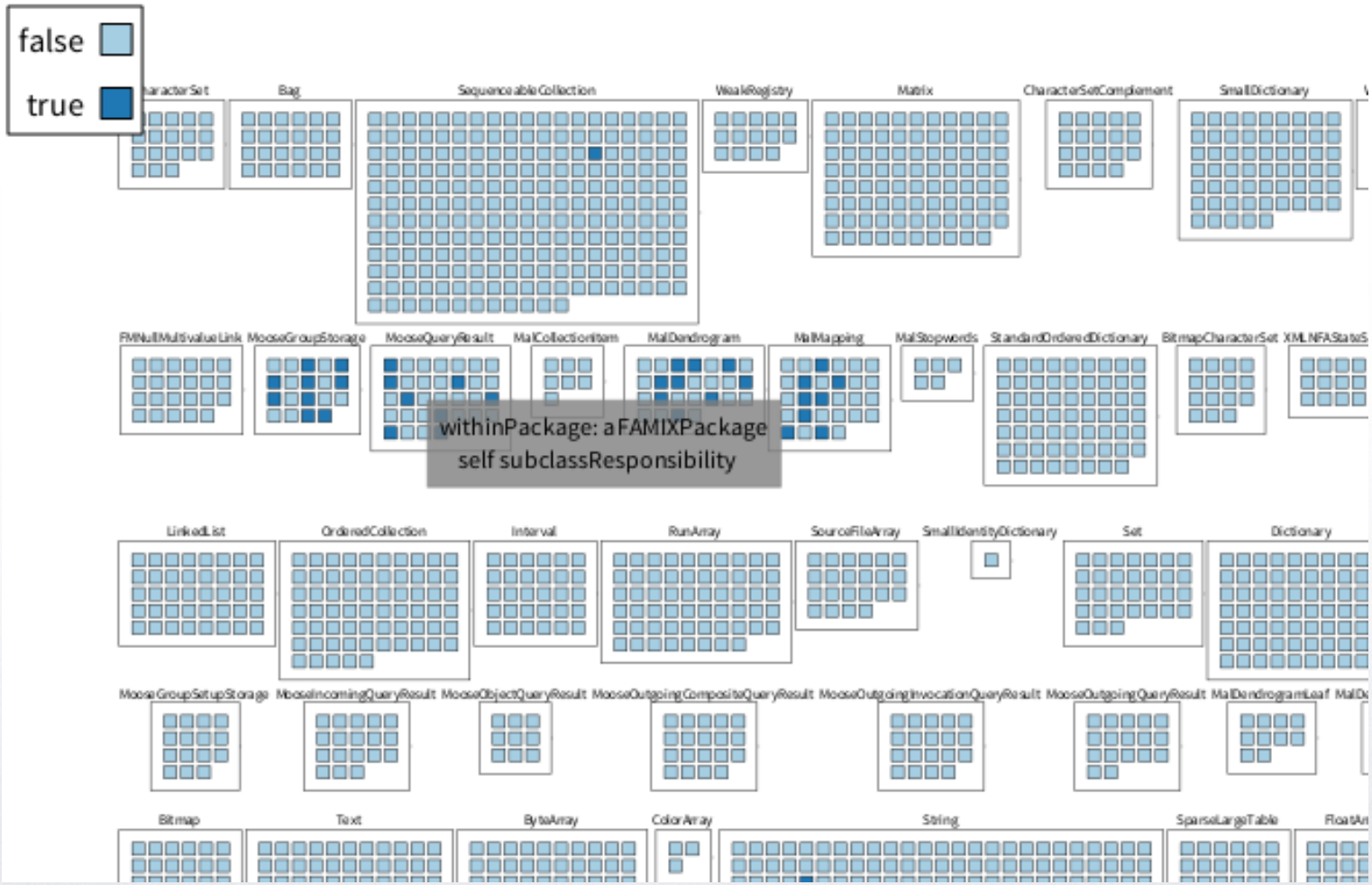
- No time to invest
- Want to visualize data
- Will reuse pre-defined visualizations
- Eventually customize for their needs


Demo

#exampleTreeExplorerCollectionHierarchy
#exampleDistributionMapAbstractMethodsCollection

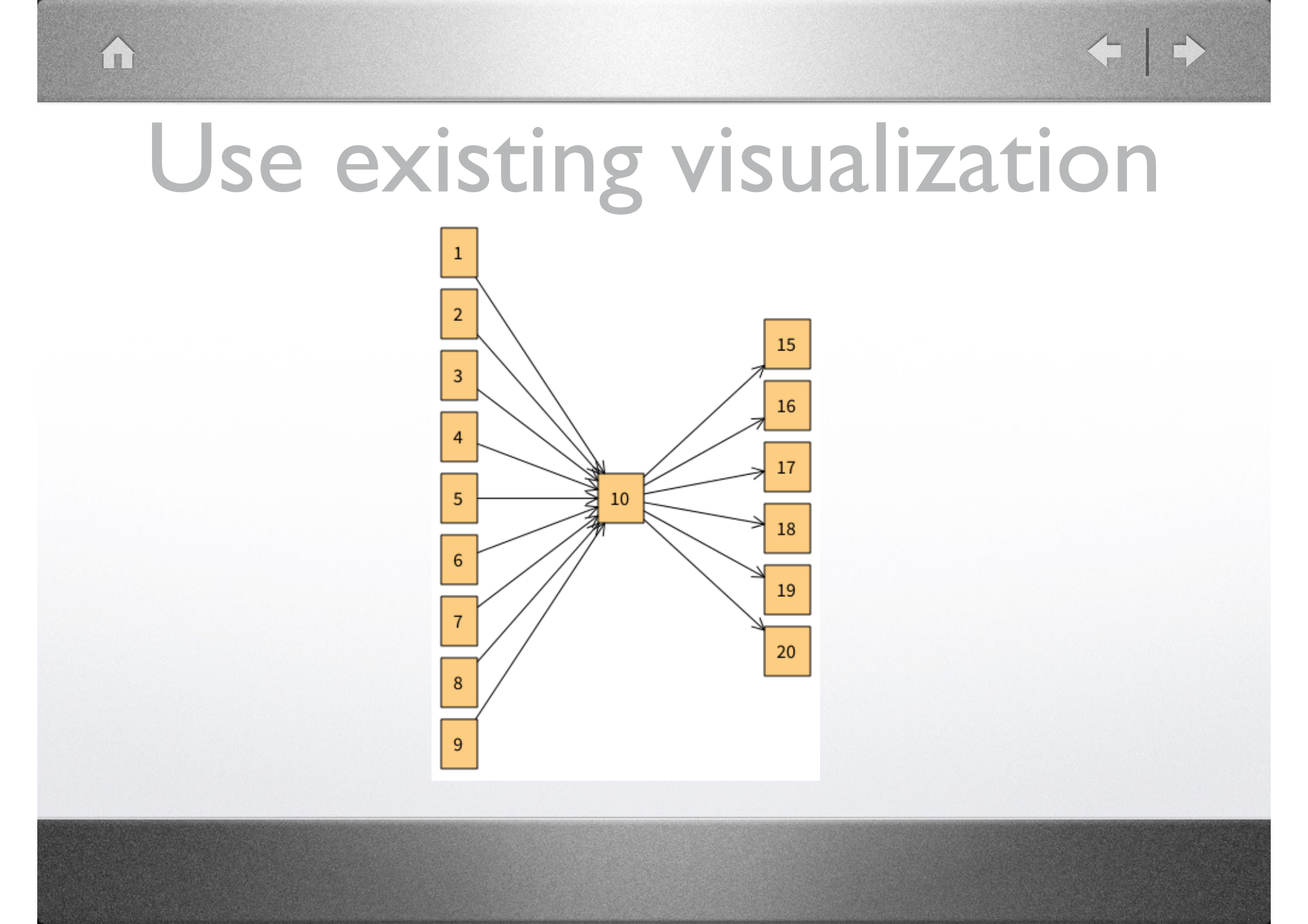

Tree Explorer
Expand
Collapse
CharacterSet
Bag
ArrayedCollection
Semaphore
LinkedList
FreeTypeCacheLinkedList
Stack
SequenceableCollection
OrderedCollection
Interval
RunArray
SourceFileArray

Distribution Map




Use existing visualization

```
| butterfly |
butterfly := TLButterfly new
    mainEntity: 10;
    leftEntities: [ :n | 1 to: n - 1 ];
    rightEntities: (15 to: 20).
butterfly open.
```



Use existing visualization

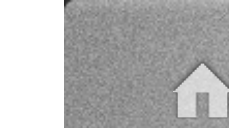

Customize visualization

```
|
butterfly
    mainEntity:
    leftEntities: [
    rightEntities: (
butterfly styleSheet shape: TLEllipse; width: 40.
butterfly styleSheet > #redBackground
    backgroundColor: Color red;
    textColor: Color white.
butterfly > #left addStyle: #redBackground.
butterfly
```

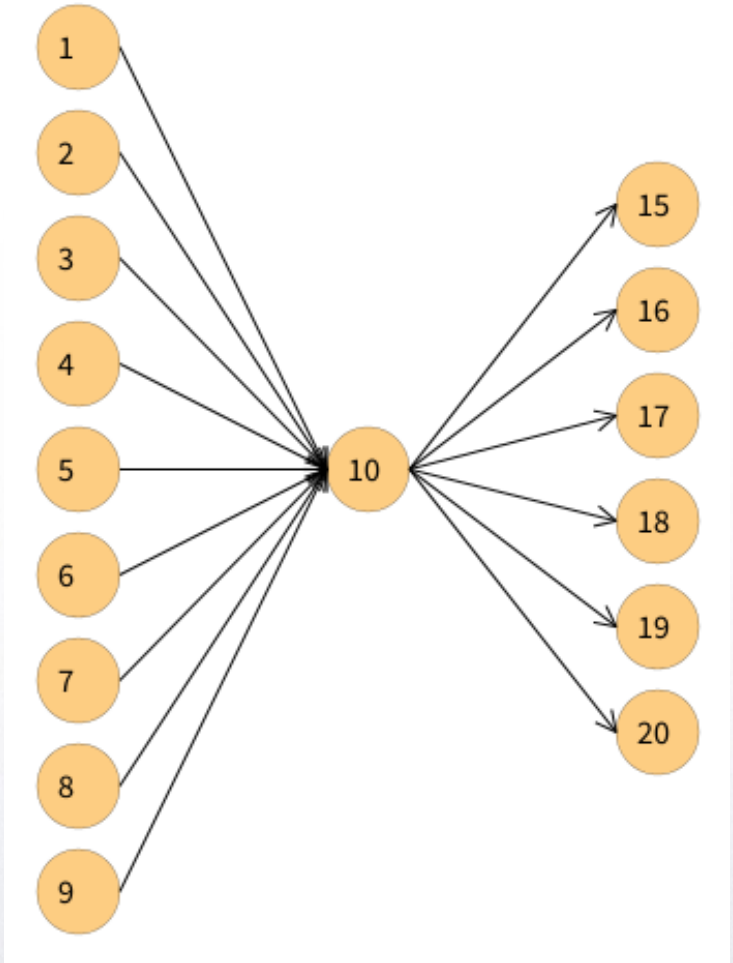

Customize visualization

```
|
butterfly
    mainEntity:
    leftEntities: [
    rightEntities: (
butterfly styleSheet shape: TLEllipse; width: 40.
butterfly styleSheet > #redBackground
    backgroundColor: Color red;
    textColor: Color white.
butterfly > #left addStyle: #redBackground.
butterfly
```

Change default style of visualization




Customize visualization




Customize visualization

```
|
butterfly
    mainEntity:
    leftEntities: [
    rightEntities: (
butterfly styleSheet shape: TLEllipse; width: 40.
butterfly styleSheet > #redBackground
    backgroundColor: Color red;
    textColor: Color white.
butterfly > #left addStyle: #redBackground.
butterfly
```

Define a new style #redBackground

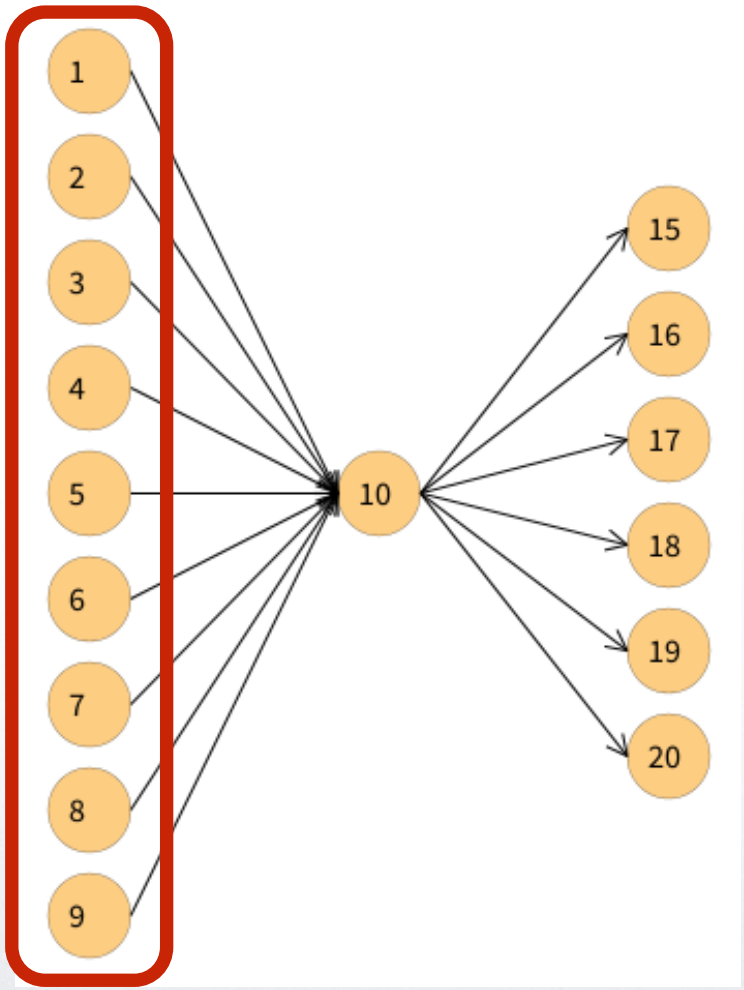

Customize visualization

```
|
butterfly
    mainEntity:
    leftEntities: [
    rightEntities: (
butterfly styleSheet shape: TLEllipse; width: 40.
butterfly styleSheet > #redBackground
    backgroundColor: Color red;
    textColor: Color white.
butterfly > #left addStyle: #redBackground.
butterfly
```

Apply this style on a group

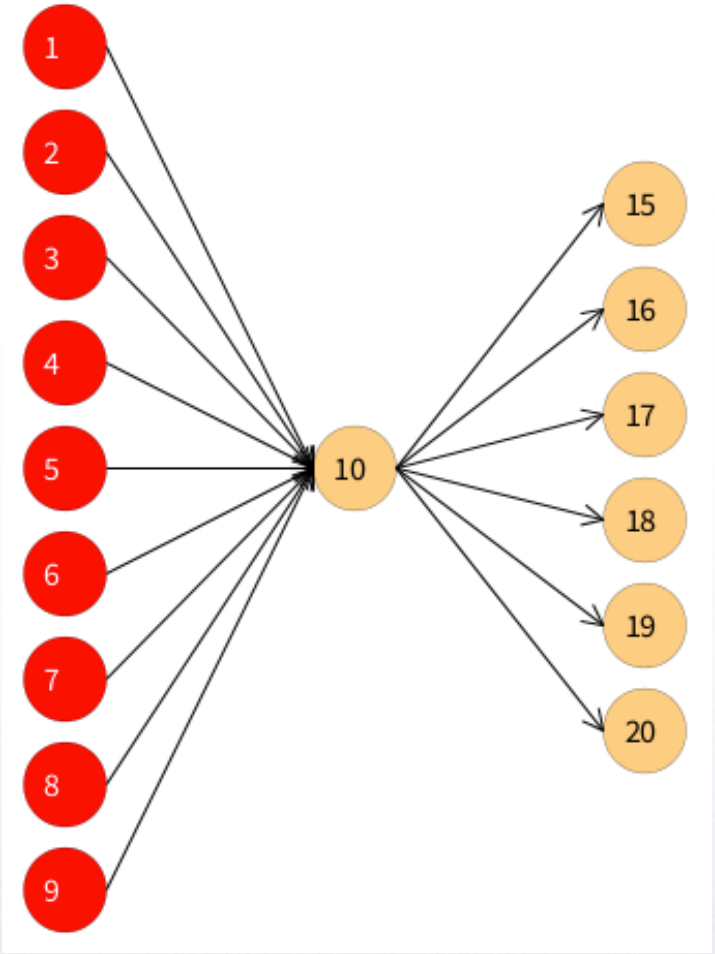
Customize visualization

#left group



Customize visualization

Customisation is efficient because Telescope modelize a visualization logic



Customized Butterfly

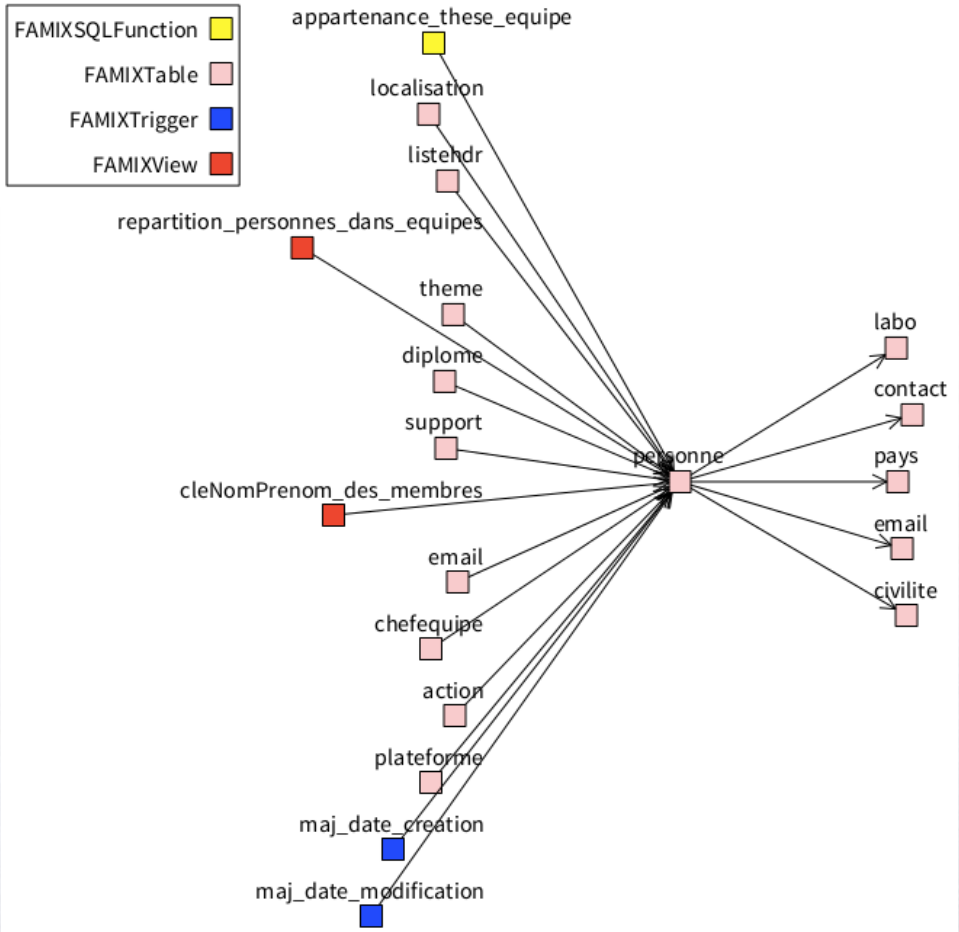
```
visu  := TLButterfly new

mainEntity: (model entities detect: [ :e | e
isTable and: [ e name = 'personne' ] ]);

leftEntities: [ :table | ((table queryAll: #in)
atScope: FAMIXSQLFunction),((table
queryAll: #in) atScope: FAMIXTable),(table
triggers )] ;

rightEntities: [ :table | ((table queryAll: #out
atScope: FAMIXTable) reject: #isNil) ].

visu styleSheet nodeLabelPosition: #top;
backgroundColor: [ :entity | entity isView
ifTrue: [ Color red ] ifFalse: [ entity isTrigger
ifTrue: [Color blue] ifFalse: [ entity
isSQLFunction ifTrue: [ Color yellow ] ifFalse:
[ entity isTable ifTrue: [Color lightRed] ifFalse:
[ Color white ] ] ] ] ].
```




Expert developers

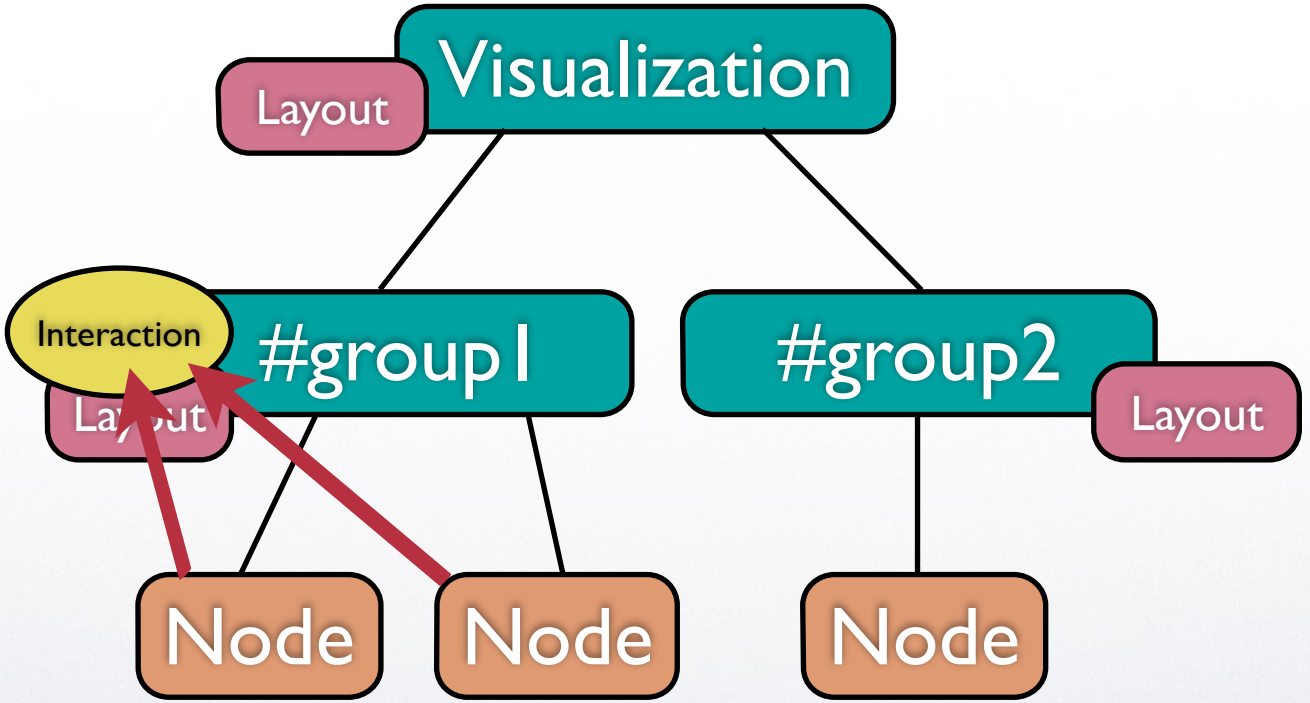
- Create new visualizations
- Have to know the model
- No need to know (or care) anything about drawing
- Can reuse pre-defined actions
- High level abstractions


Demo in order to explain construction and logic

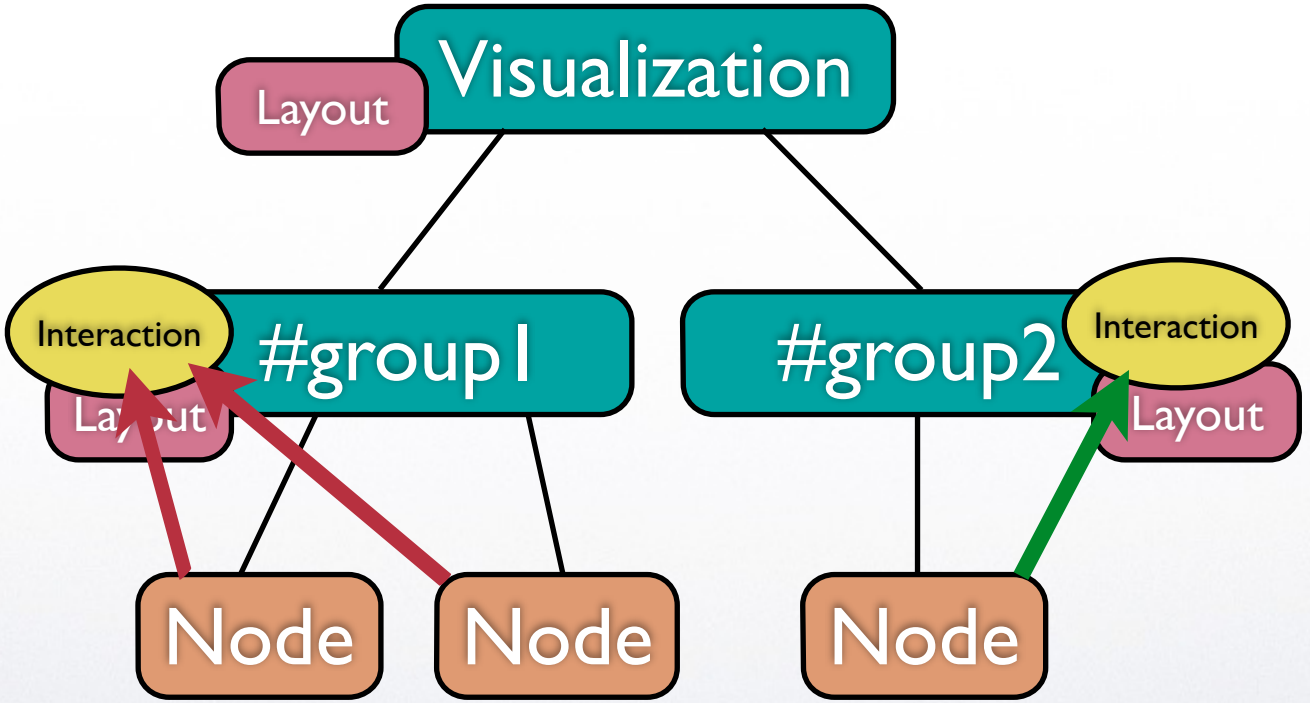
#exampleMovingNodeToAnotherGroup


Logic of the visualisation

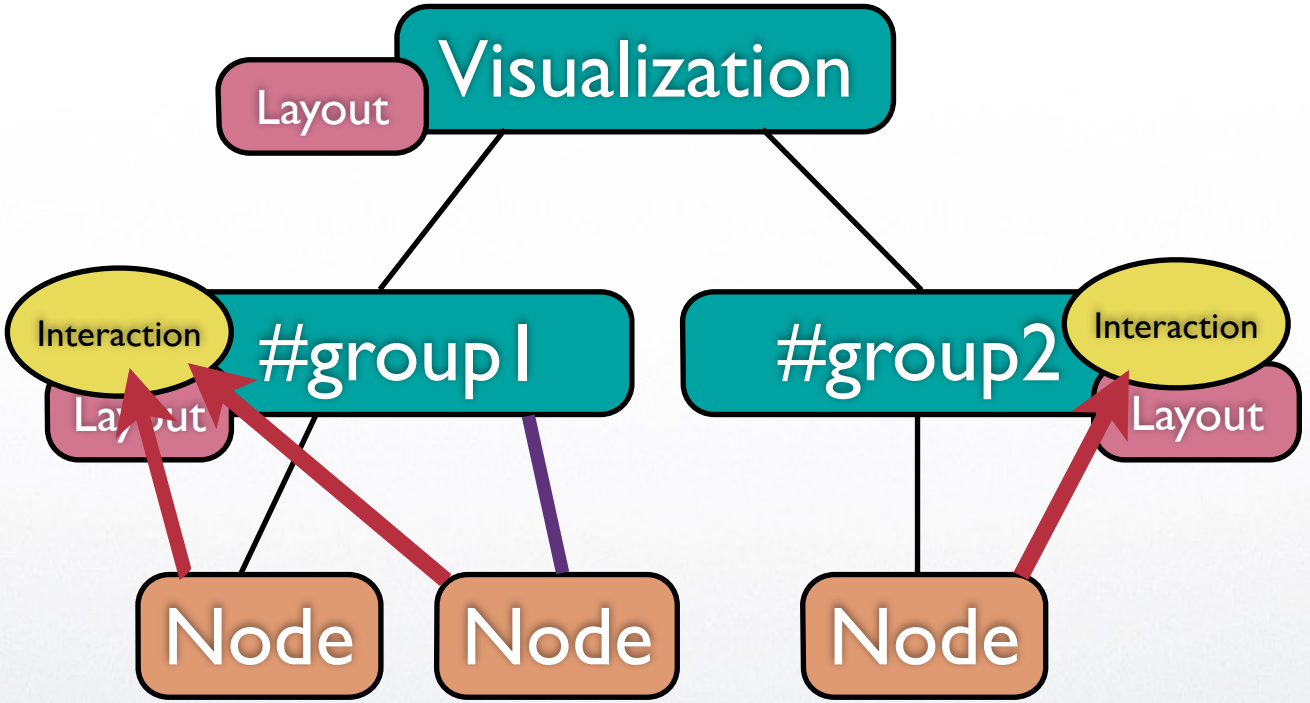
Visualization Model

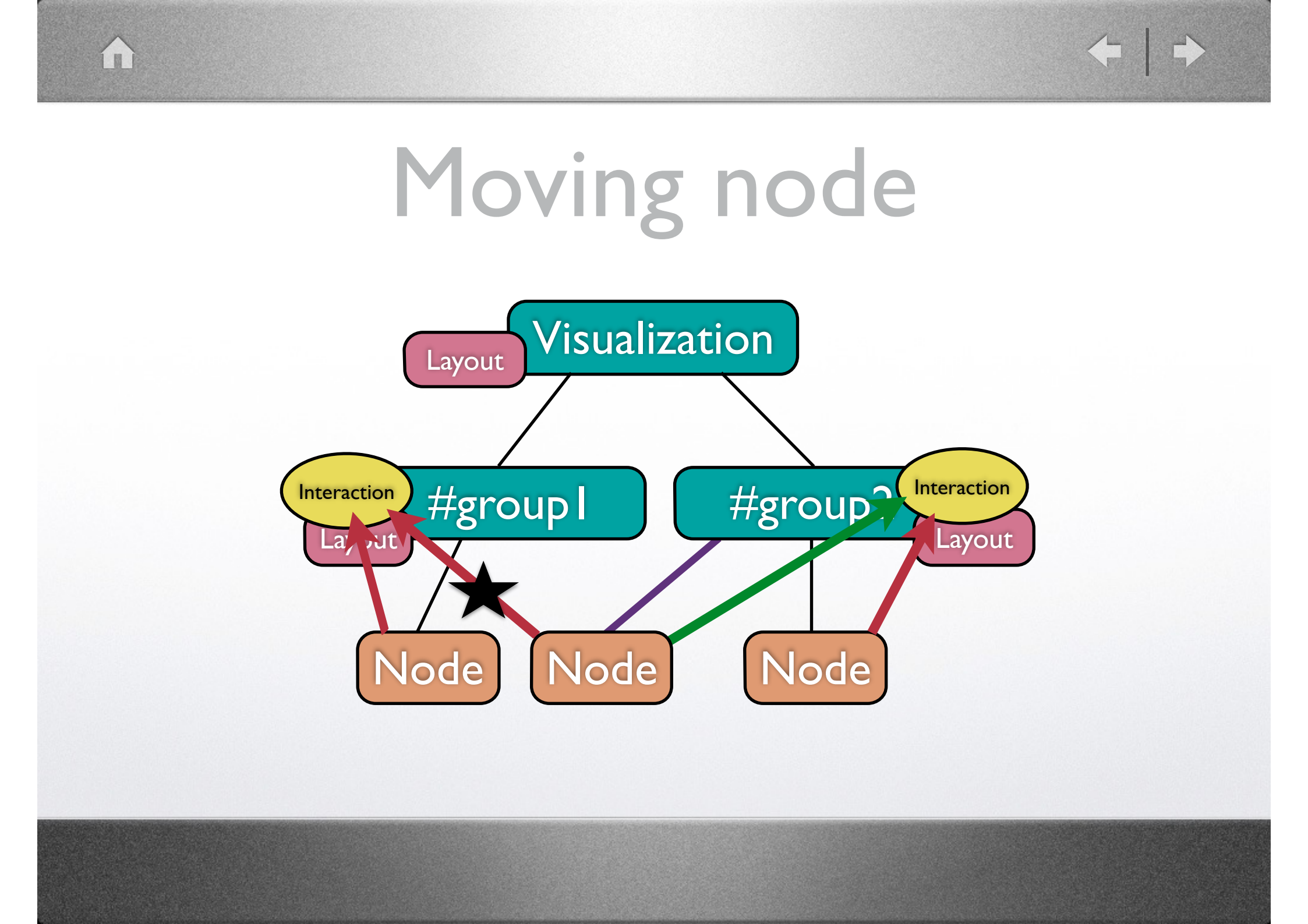


Add new interaction

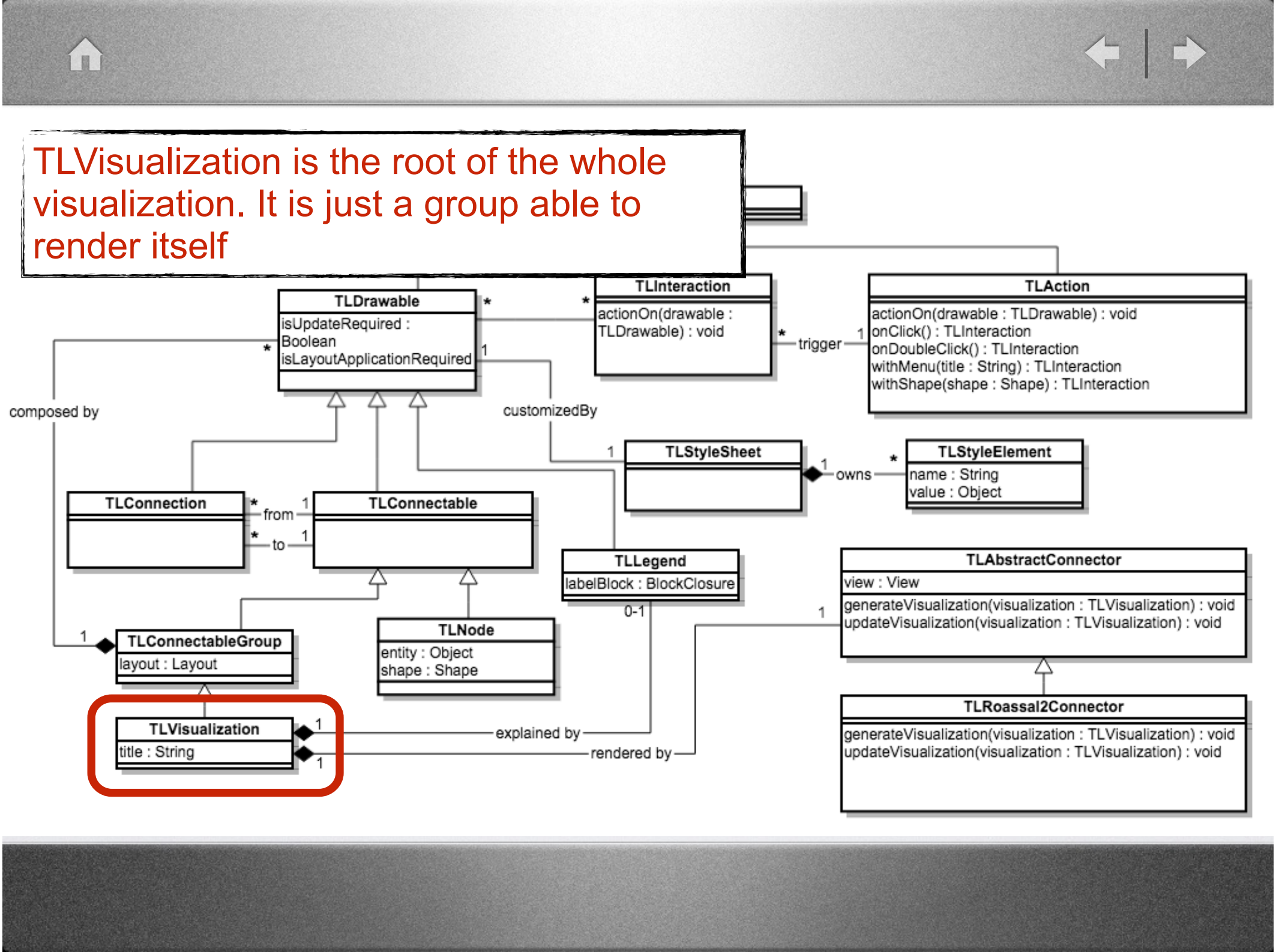


Moving node


Visualization
Layout
Interaction
#group1
#group2
Interaction
Layout
Node
Node
Node

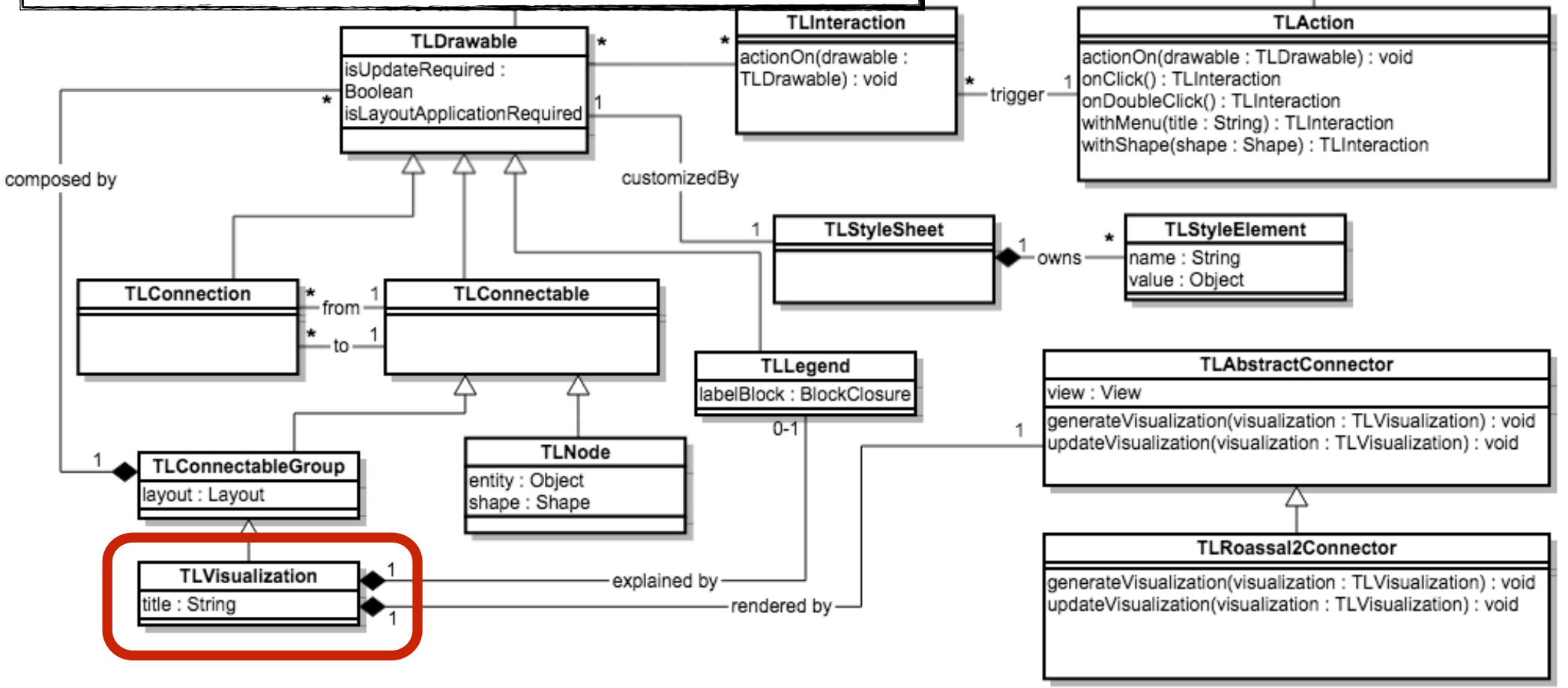

Moving node
Visualization
Layout
Interaction
#group 1
#group 2
Interaction
Layout
Node
Node
Node

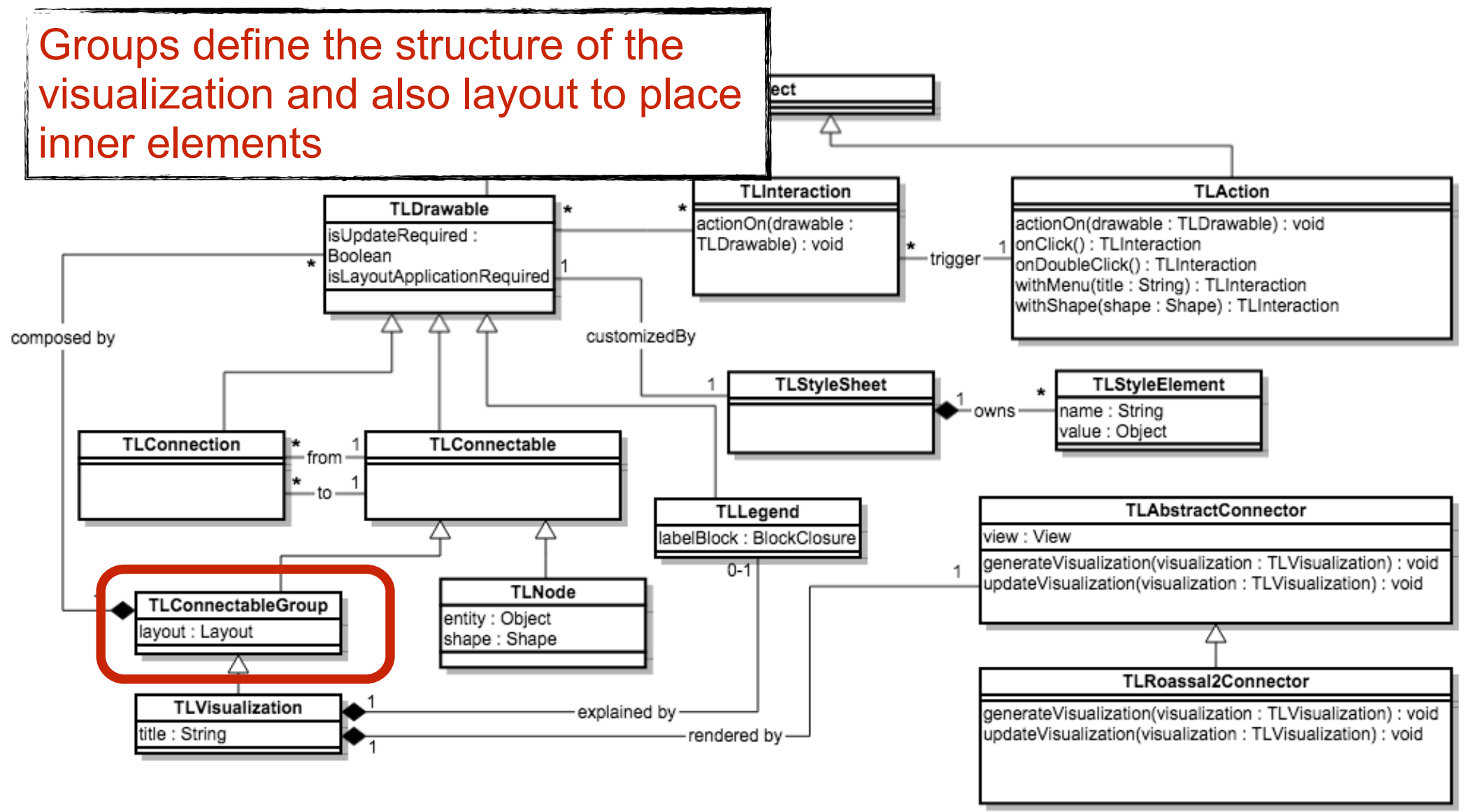

Creating a visualization


TLVisualization is the root of the whole visualization. It is just a group able to render itself
TLDrawable
isUpdateRequired : Boolean
isLayoutApplicationRequired
TLInteraction
actionOn(drawable : TLDrawable) : void
TLAction
actionOn(drawable : TLDrawable) : void
onClick() : TLInteraction
onDoubleClick() : TLInteraction
withMenu(title : String) : TLInteraction
withShape(shape : Shape) : TLInteraction
trigger
composed by
customizedBy
TLStyleSheet
owns
TLStyleElement
name : String
value : Object
TLConnection
from
to
TLConnectable
TLLegend
labelBlock : BlockClosure
0-1
TLAbstractConnector
view : View
generateVisualization(visualization : TLVisualization) : void
updateVisualization(visualization : TLVisualization) : void
TLConnectableGroup
layout : Layout
TLNode
entity : Object
shape : Shape
TLVisualization
title : String
explained by
rendered by
TLRoassal2Connector
generateVisualization(visualization : TLVisualization) : void
updateVisualization(visualization : TLVisualization) : void

TLVisualization is the root of the whole visualization. It is just a group able to render itself

```
visu := TLVisualization new.
visu open
```

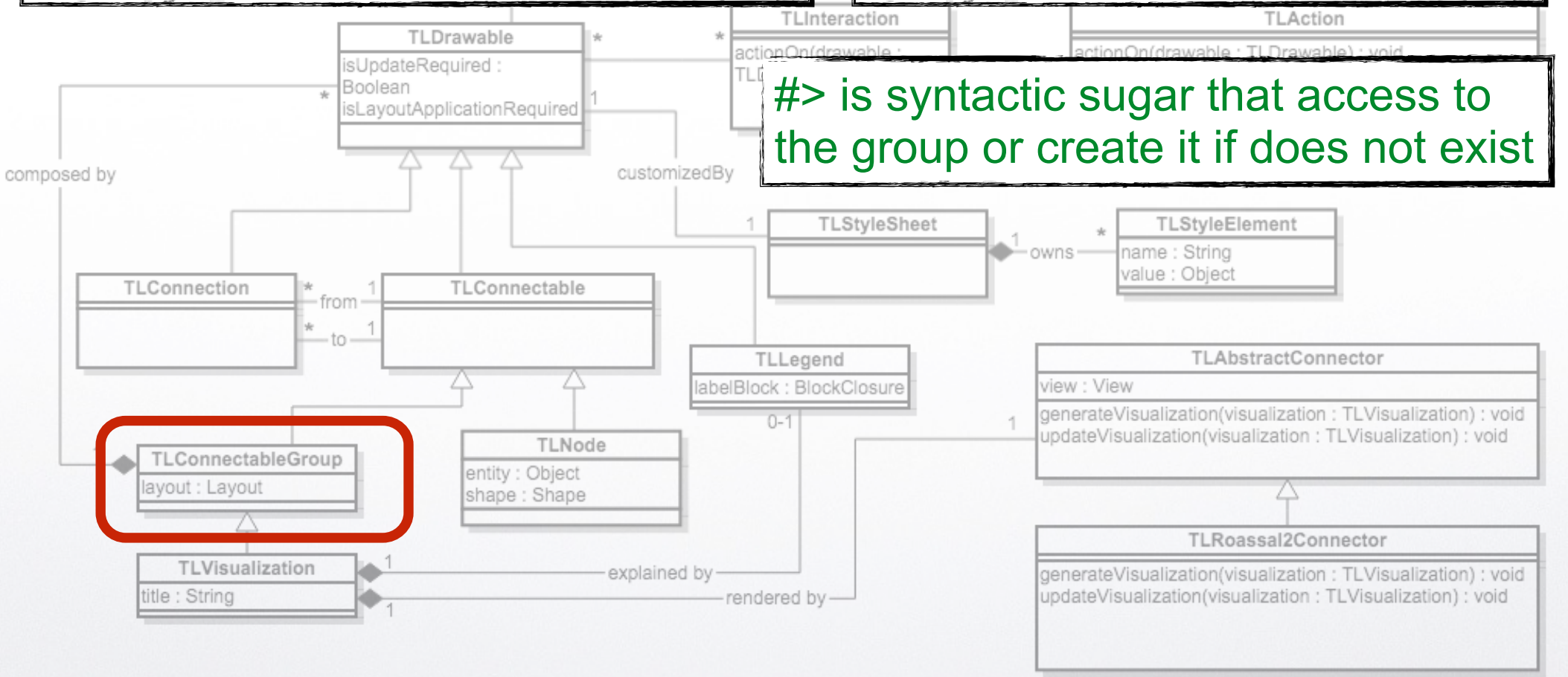



Groups define the structure of the visualization and also layout to place inner elements
ect
TLDrawable
isUpdateRequired : Boolean
isLayoutApplicationRequired
TLInteraction
actionOn(drawable : TLDrawable) : void
TLAction
actionOn(drawable : TLDrawable) : void
onClick() : TLInteraction
onDoubleClick() : TLInteraction
withMenu(title : String) : TLInteraction
withShape(shape : Shape) : TLInteraction
trigger
composed by
customizedBy
TLStyleSheet
owns
TLStyleElement
name : String
value : Object
TLConnection
from
to
TLConnectable
TLLegend
labelBlock : BlockClosure
0-1
TLAbstractConnector
view : View
generateVisualization(visualization : TLVisualization) : void
updateVisualization(visualization : TLVisualization) : void
TLConnectableGroup
layout : Layout
TLNode
entity : Object
shape : Shape
TLVisualization
title : String
explained by
rendered by
TLRoassal2Connector
generateVisualization(visualization : TLVisualization) : void
updateVisualization(visualization : TLVisualization) : void

Groups define the structure of the visualization and also layout to place inner elements

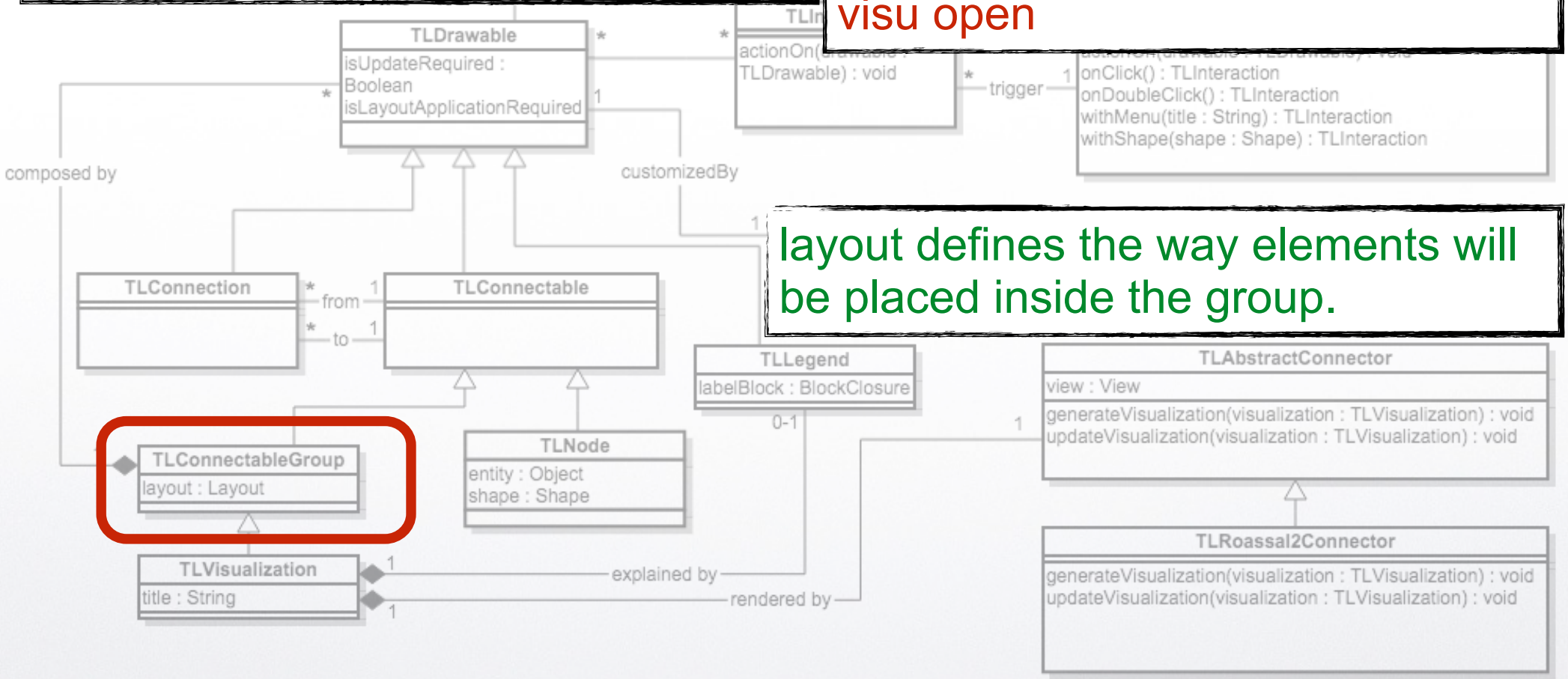
```
visu := TLVisualization new.
visualization > #group1.
visu open
```

#> is syntactic sugar that access to the group or create it if does not exist



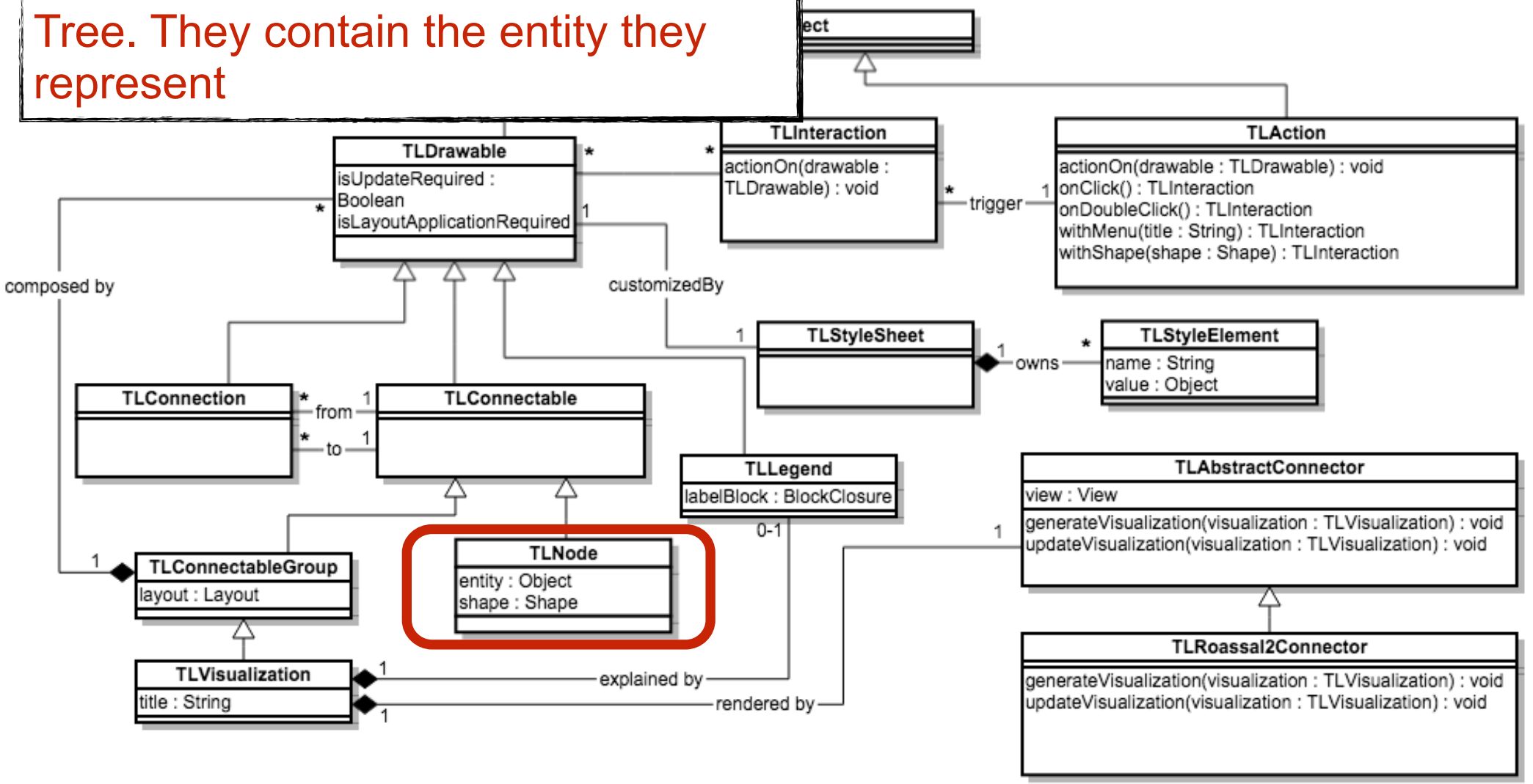
Groups define the structure of the visualization and also layout to place inner elements

```
visu := TLVisualization new.
visualization > #group1
    layout: RTVerticalLineLayout
visu open
```



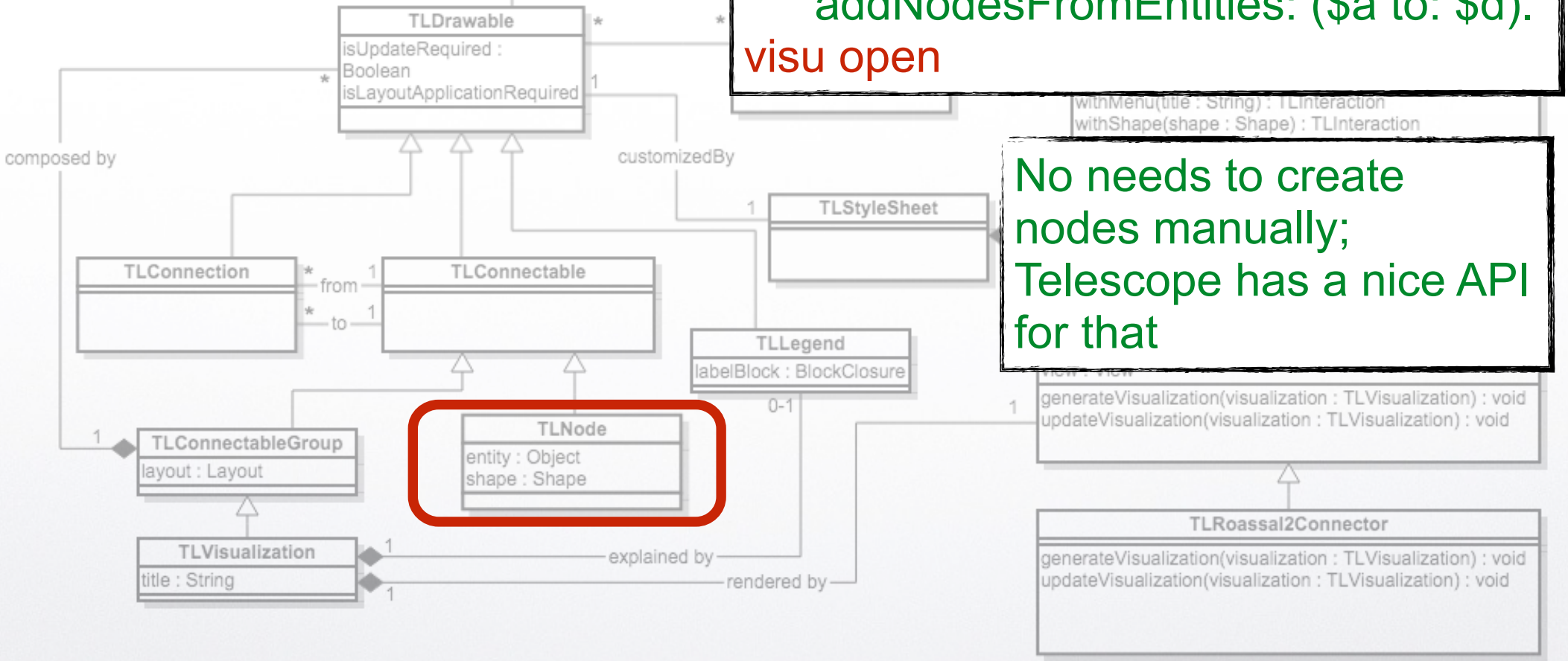
layout defines the way elements will be placed inside the group.

Nodes are leaves of the visualization Tree. They contain the entity they represent



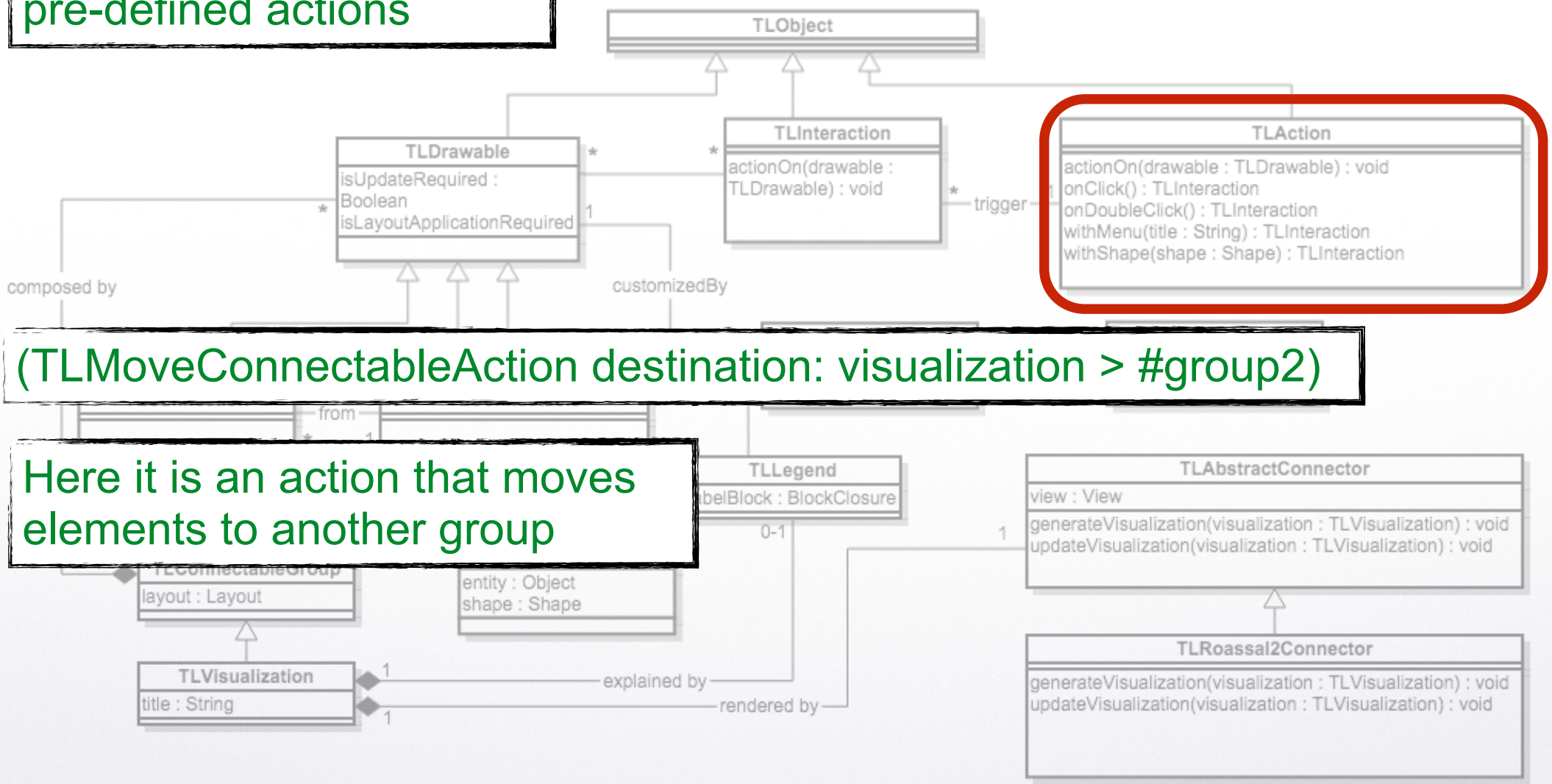
Nodes are leaves of the visualization Tree. They contain the entity they represent

```
visu := TLVisualization new.
visualization > #group1
    layout: RTVerticalLineLayout;
    addNodesFromEntities: ($a to: $d).
visu open
```



No needs to create nodes manually; Telescope has a nice API for that

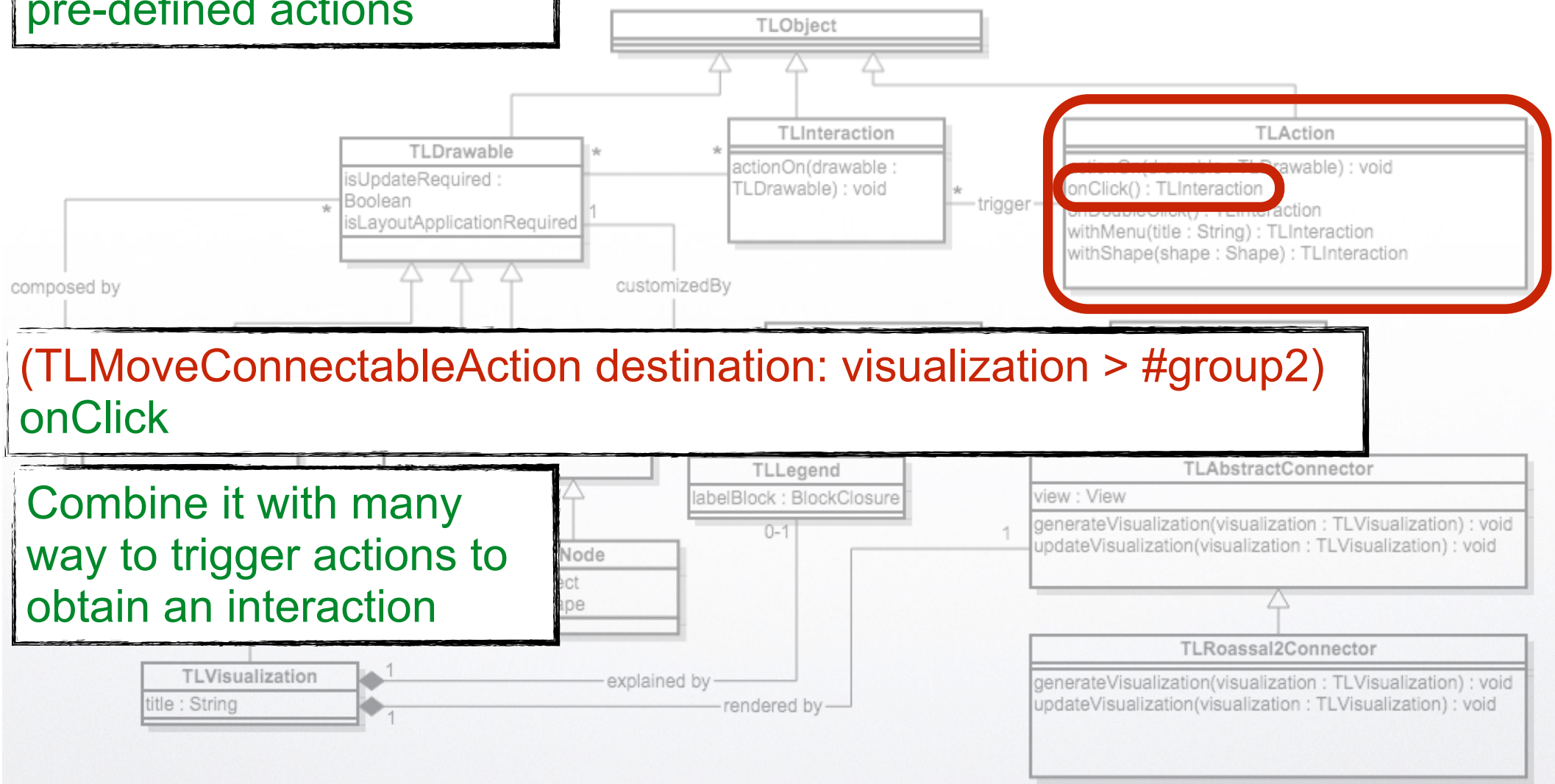
Telescope offers many pre-defined actions



(TLMoveConnectableAction destination: visualization > #group2)

Here it is an action that moves elements to another group

Telescope offers many pre-defined actions

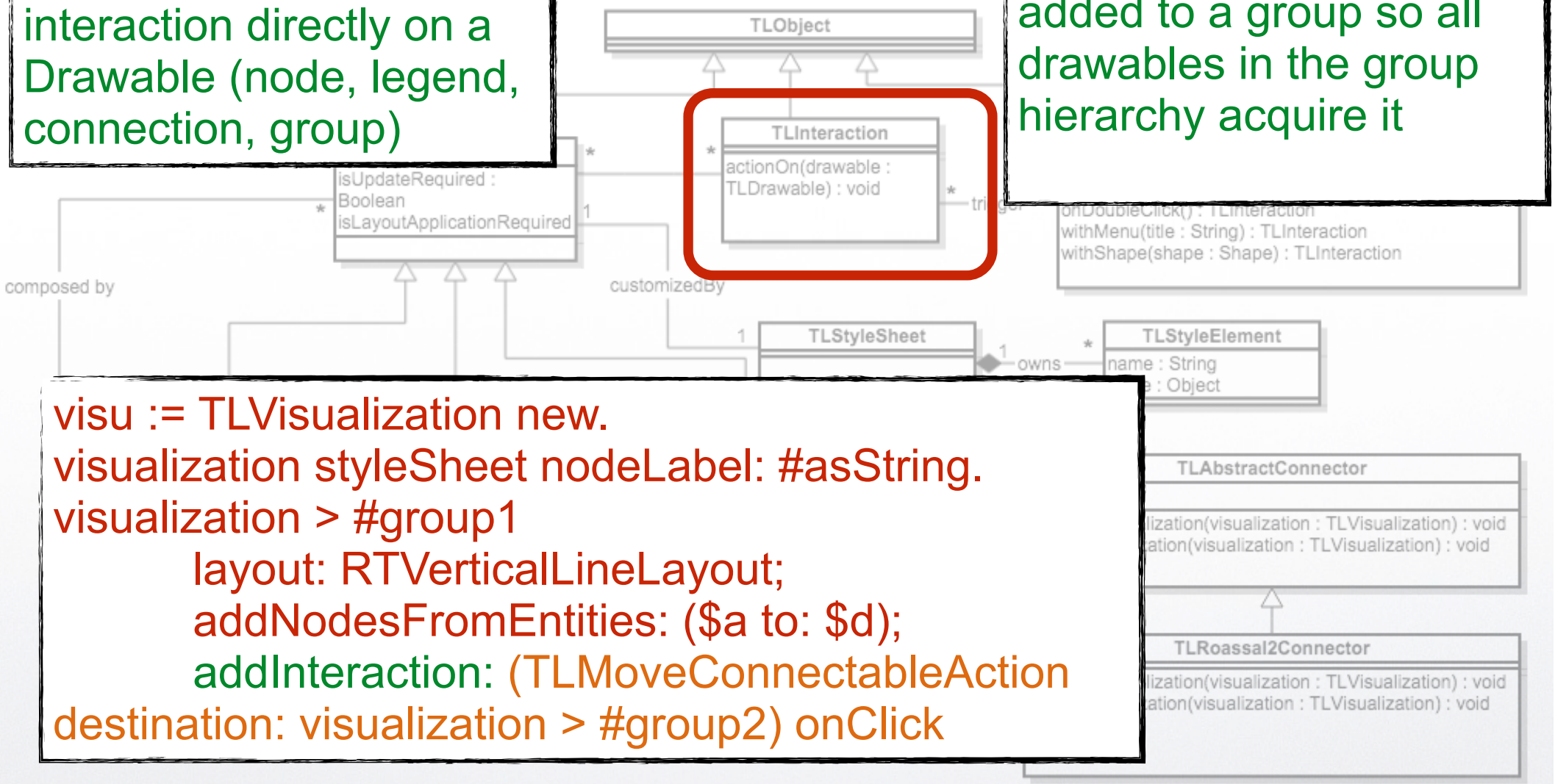


(TLMoveConnectableAction destination: visualization > #group2) onClick

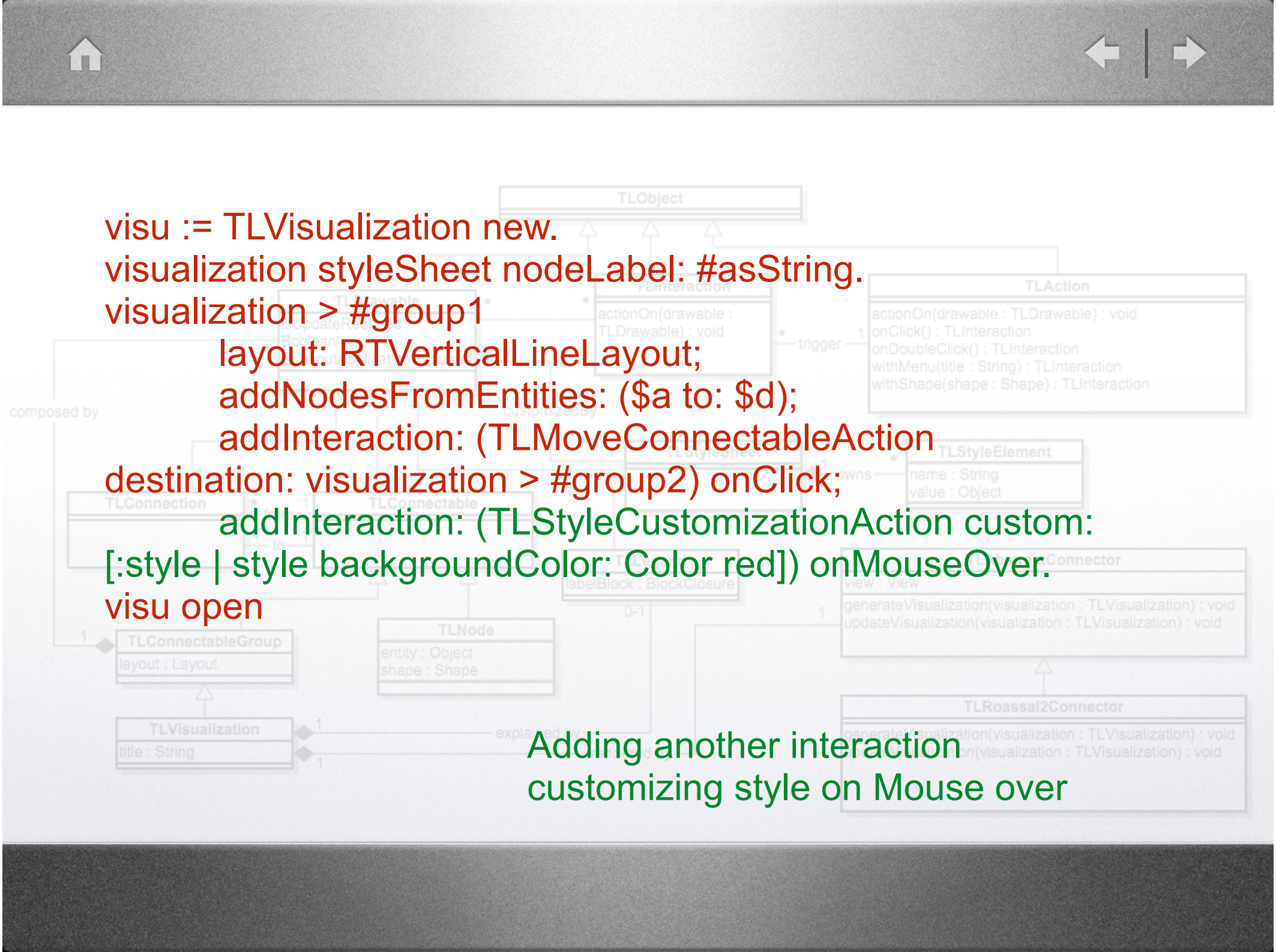
Combine it with many way to trigger actions to obtain an interaction

Now you can use this interaction directly on a Drawable (node, legend, connection, group)

Here the interaction is added to a group so all drawables in the group hierarchy acquire it

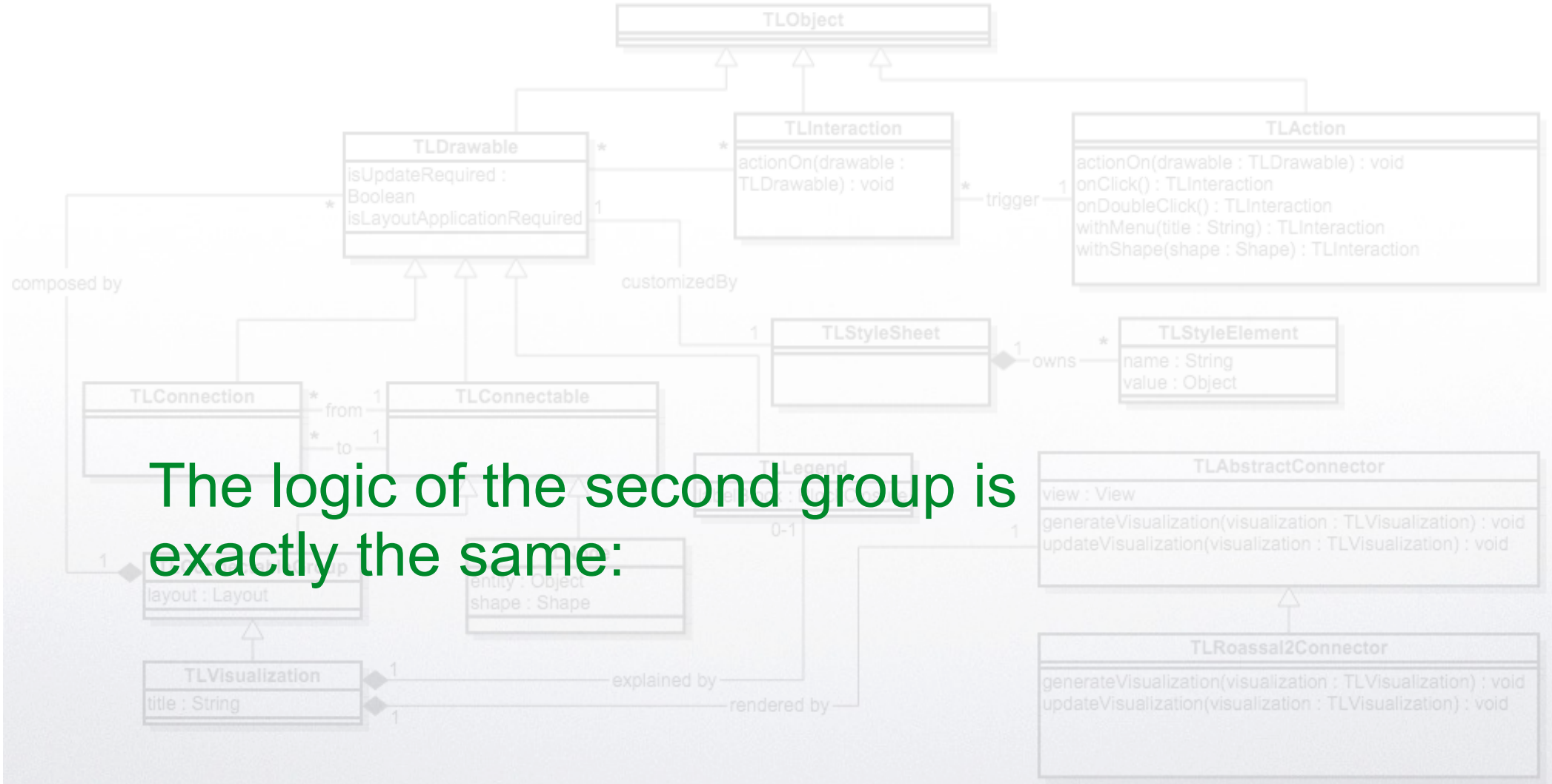



```
visu := TLVisualization new.
visualization styleSheet nodeLabel: #asString.
visualization > #group1
        layout: RTVerticalLineLayout;
        addNodesFromEntities: ($a to: $d);
        addInteraction: (TLMoveConnectableAction
destination: visualization > #group2) onClick
```



```
visu := TLVisualization new.
visualization styleSheet nodeLabel: #asString.
visualization > #group1
        layout: RTVerticalLineLayout;
        addNodesFromEntities: ($a to: $d);
        addInteraction: (TLMoveConnectableAction
destination: visualization > #group2) onClick;
        addInteraction: (TLStyleCustomizationAction custom:
[:style | style backgroundColor: Color red]) onMouseOver.
visu open
```

Adding another interaction
customizing style on Mouse over



The logic of the second group is exactly the same:


```
visu := TLVisualization new.
visualization styleSheet nodeLabel: #asString.
visualization > #group1
        layout: RTVerticalLineLayout;
        addNodesFromEntities: ($a to: $d);
        addInteraction: (TLMoveConnectableAction
destination: visualization > #group2) onClick;
        addInteraction: (TLStyleCustomizationAction custom:
[:style | style backgroundColor: Color red]) onMouseOver.
visualization > #group2
        layout: RTVerticalLineLayout;
        addNodesFromEntities: ($e to: $h);
        addInteraction: (TLMoveConnectableAction
destination: visualization > #group1) onClick;
        addInteraction: (TLStyleCustomizationAction custom:
[:style | style backgroundColor: Color blue]) onMouseOver.
visu open
```


Conclusion

- Telescope is a model to represent visualization
- Telescope is built on top of Roassal and let it the rendering.

- If you get any question please ask guillaume.larcheveque@synectique.eu