Styling Qt Using Style Sheets

TROLLTECH®

About me

- Me
  - Girish Ramakrishnan
  - Software Developer + Release manager
- Qt Development
  - Qt Style Sheet architect
  - MinGW platform boss
  - Auto completion framework
  - Desktop integration
  - Part of widget team (message box, label)
- FOSS
  - KDocker (kdocker.sourceforge.net)
  - GUC (guc.mozdev.org)

History

- QStyle
  - Excellent flexible API. Used by Qt itself
  - “Compile, test, compile test” cycle
  - QStyle completely shuts off graphics designers

Introducing Qt Style Sheets

- Simple customization like setting colors and backgrounds
- Style Guarantees
- Complete customization of widget look
- Introduced in Qt 4.2 for form widgets

What are Style Sheets?

- “Style strings”
  - Similar to CSS in concept and syntax
  - Adapted to the world of widgets
- Interactive UI development
  - Designer friendly. Compile free

Style Sheet Syntax

- A Style Rule

```
QPushButton { color: red; }
```

QPushButton – Selector
{ color: red } - Declaration
color – attribute/property

Style Sheet Syntax

- A Style Sheet is a set of rules

```
QPushButton, QCheckBox {
    background-color: magenta;
}

QRadioButton, QCheckBox {
    spacing: 8px;
    color: gray;
}
```

Selectors

- Selectors decide whether a rule applies
  - *Type* QPushButton { color: red }
  - *Class* .QCheckBox { background-color: pink; }
  - *ID* #foodMenuView { alternating-color: gray; }
  - *Descendant* QDialog QRadioButton { spacing: 10px; }
  - *Attribute* QLabel[text=”falafel”] { font-size: 14pt; }
- Whole range of CSS2.1 selectors supported

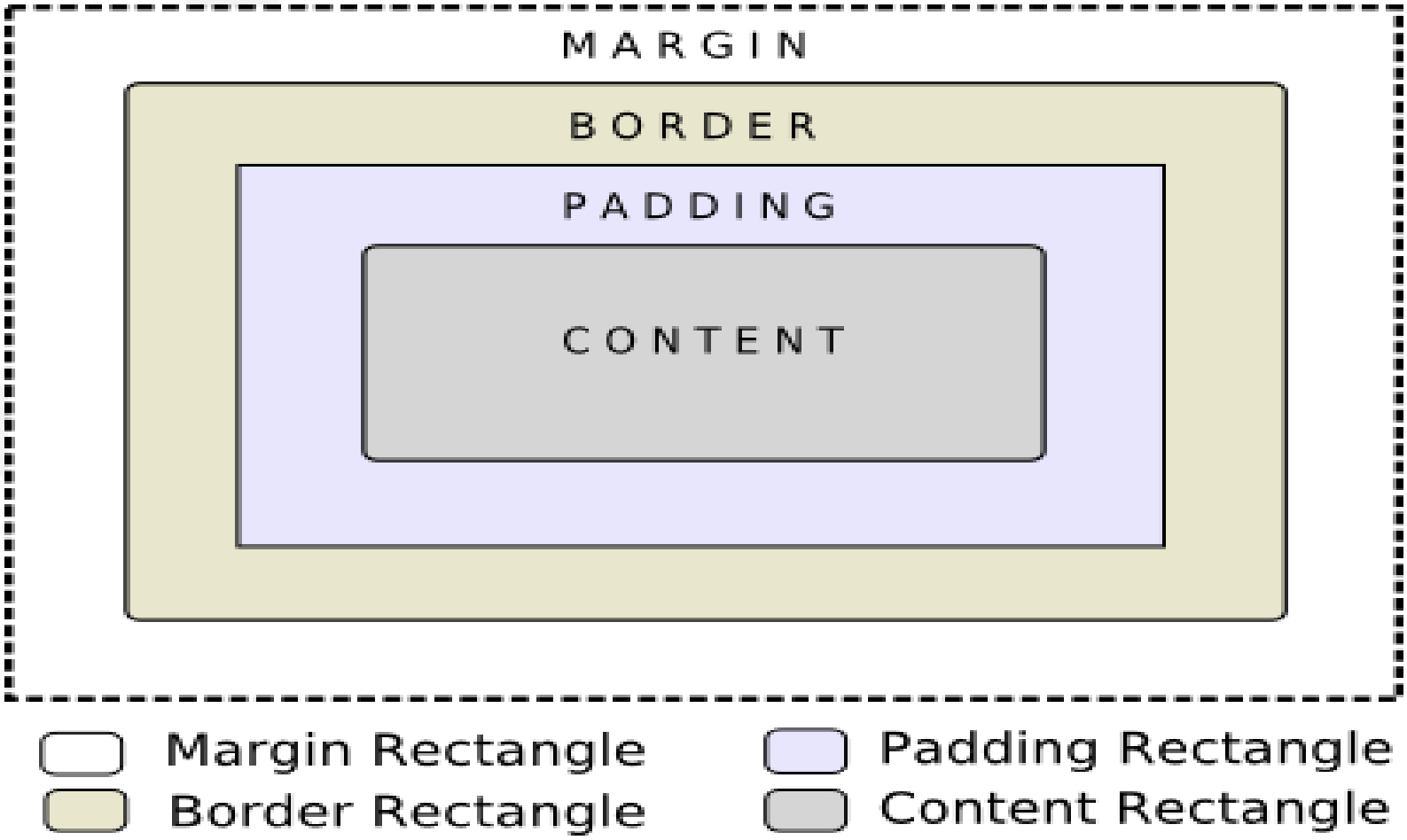
Pseudo States

- User interface states

```
QPushButton:hover { color: red }
QCheckBox:pressed:checked { color:
  pink }
```

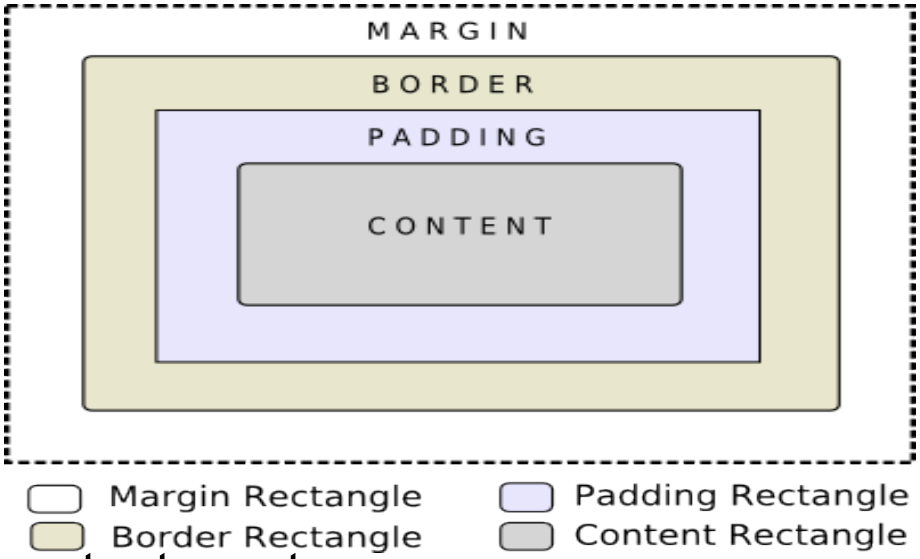
  - :checked , :disabled , :enabled, :focus, :hover , :indeterminate , :off :on , :pressed, :unchecked (around 36)
  - Use “!” for negation
    - `QPushButton:!hover { color: white; }`

The Box Model



Qt Style Sheet Box Model Attributes

- margin
- border
  - border-width
  - border-style
  - border-color
- padding
- min-width, min-height
  - width and height refer to contents rect
- width, height not supported



Background properties

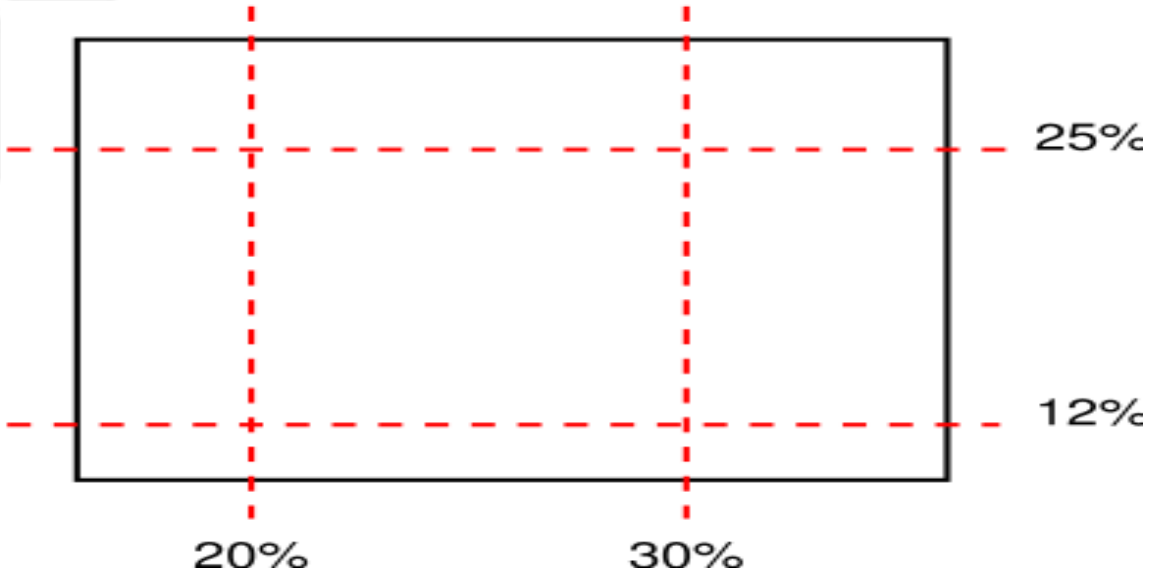
- background
  - background-image
  - background-repeat
  - background-position
  - background-clip, background-origin
  - background-attachment
- setting background does not scale!
  - Why?

Scalable style sheet

- Use gradients
- border-radius to provide rounded corners

Border Image

- Borrowed idea from CSS3 to make borders scalable
- Image cut into 9 parts

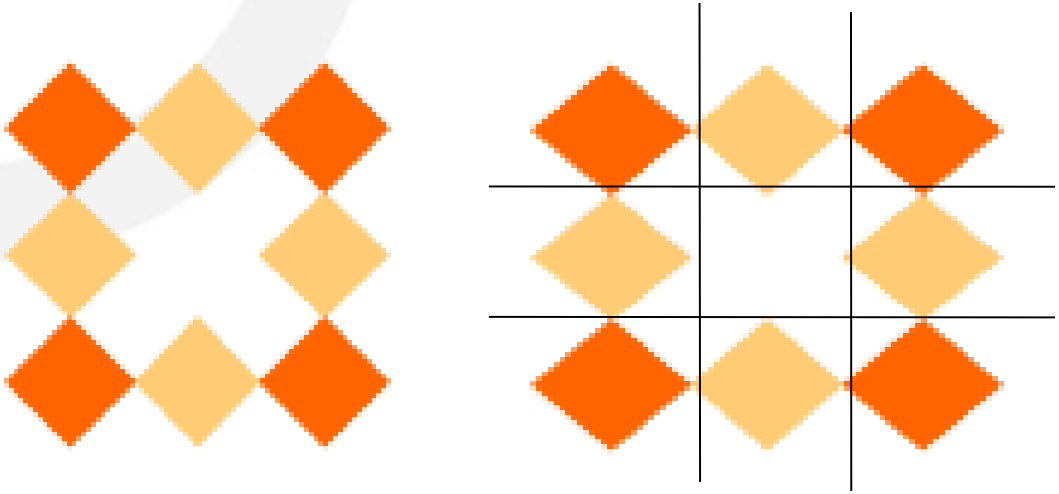


Border Image

- border-image: url(x.png) 3 3 3 3 tile stretch
  - border-image: url top right bottom left hstretch vstretch

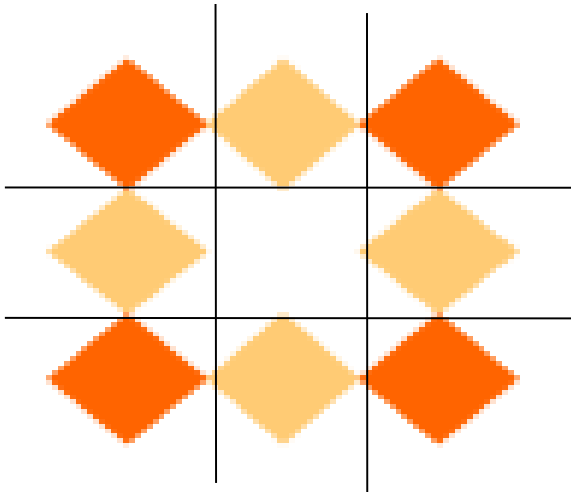
Border Image

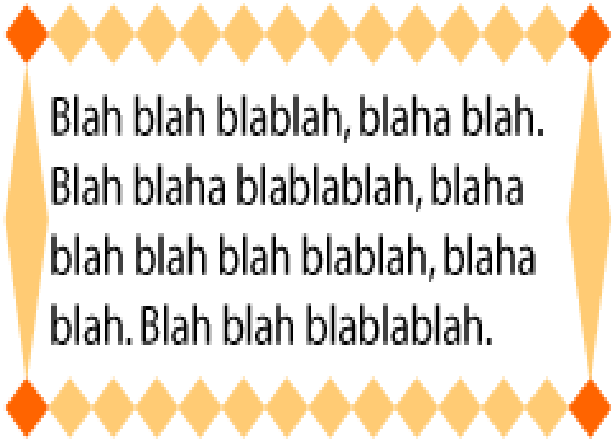
- border-image: url(x.png) 3 3 3 3 tile stretch
  - border-image: url top right bottom left hstretch vstretch



Border Image

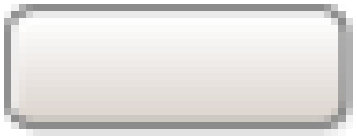
- border-image: url(x.png) 3 3 3 3 tile stretch
  - border-image: url top right bottom left hstretch vstretch





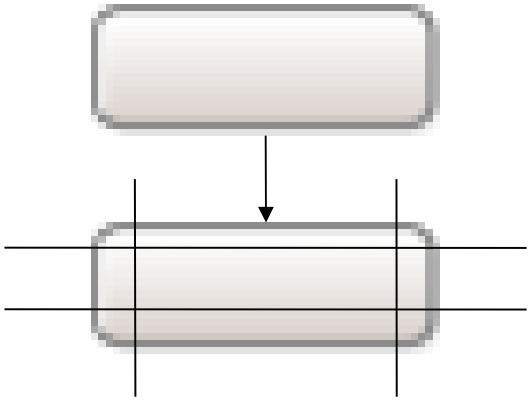
Border Image

- `border-image: url(btn.png) 5 5 5 5 stretch stretch;`



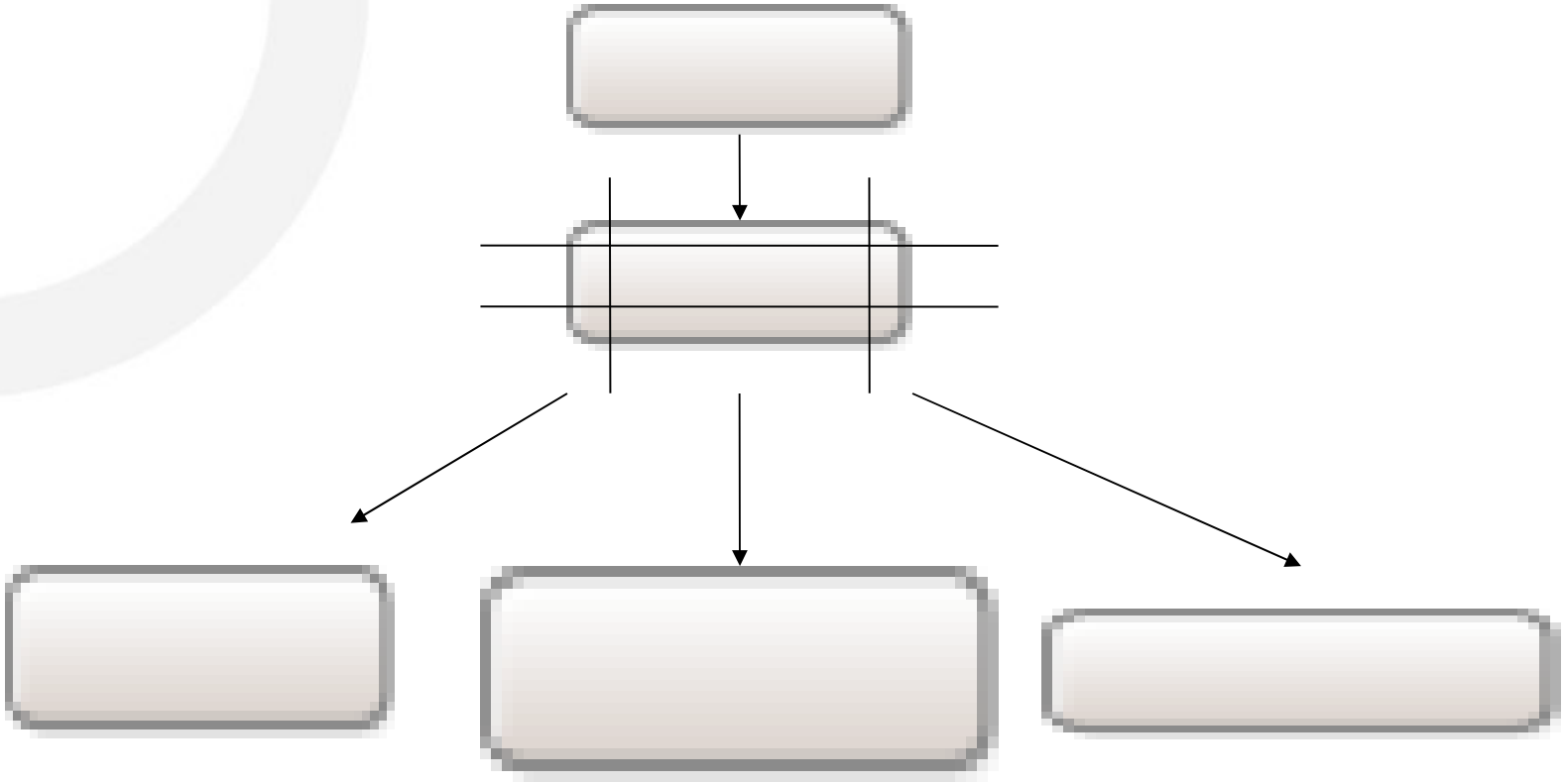
Border Image

- `border-image: url(btn.png) 5 5 5 5 stretch stretch;`



Border Image

- `border-image: url(btn.png) 5 5 5 5 stretch stretch;`



Rendering Model

- So what happens if you set the background, border and border-radius?

Rendering Model

- Set clip for entire rendering operation
  - border-radius
- Draw the background
  - First fill with background-color
  - Then draw the background-image
- Draw the border
  - Draw the border-image or the border
- Draw overlay image
  - image

Setting the style sheet

- QWidget::setStyleSheet()
- QApplication::setStyleSheet()

- What happens when rules conflict?

Sub controls

- Sub controls are “Parts” of complex widget
  - drop-down indicator of combo box
  - menu-indicator of push button
  - buttons of a spin box
- Follow CSS3 Pseudo Element syntax but has little to do conceptually
  - QPushButton::menu-indicator { image:url(indicator.png) }

Sub controls

- Sub controls are styled the exact same way as normal elements
  - Box model
  - *border-image* property
  - *image* property
  - Pseudo states

```
QPushButton::menu-indicator:hover {
  border: 2px solid red;
 image: url(indicator_hover.png)
}
```

Sub controls (Geometry)

- Size
  - width, height
  - width and height of the element's *content*

Rendering Model

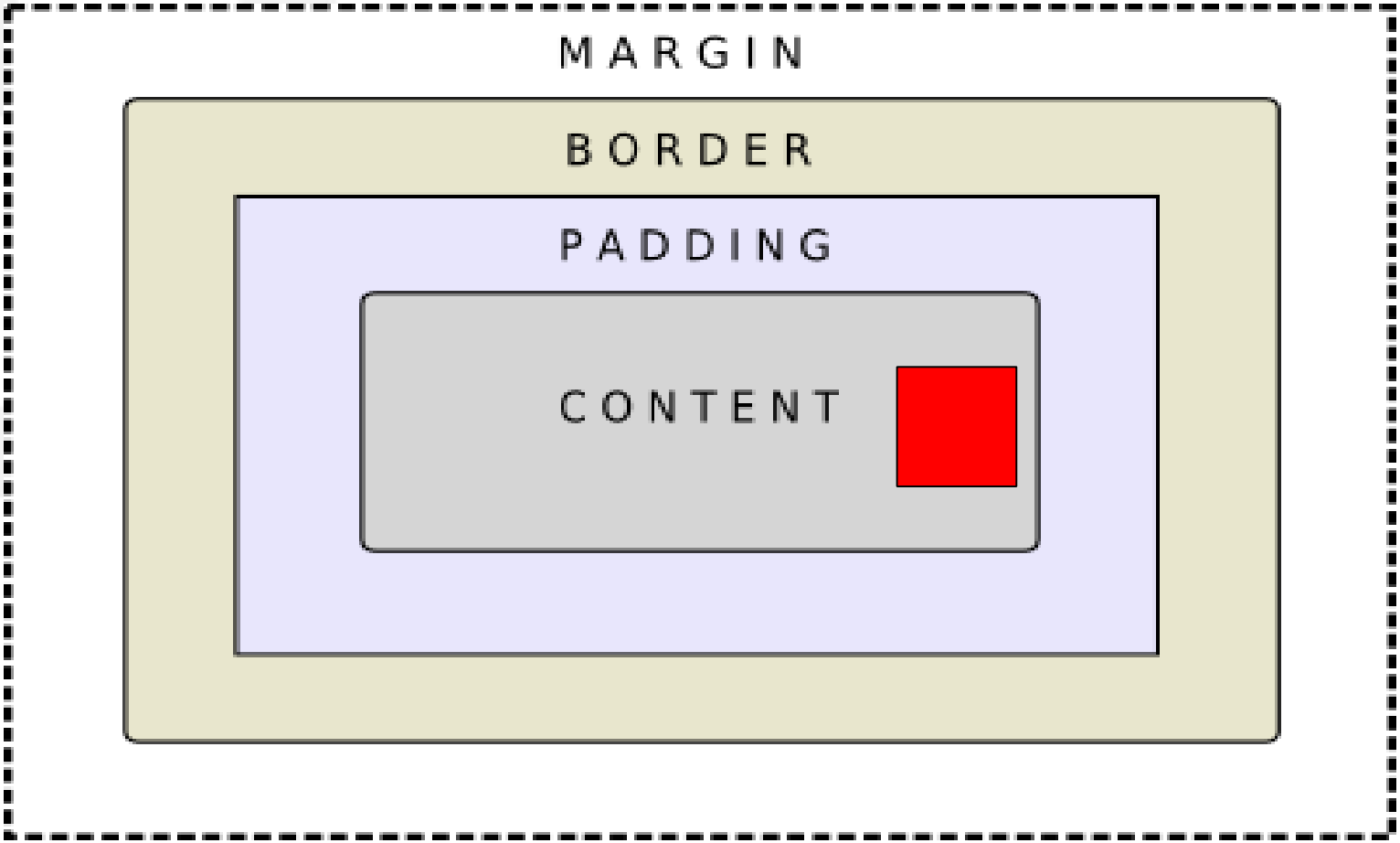
- Sub controls are rendered hierarchically
- Rendering
  - QComboBox { }
  - QComboBox::drop-down { }
  - QComboBox::down-arrow { }
- Subcontrols always have a parent and are positioned w.r.t parent
  - The drop-down is positioned w.r.t QComboBox
  - The down-arrow is positioned w.r.t drop-down

Sub control positioning

- subcontrol-origin
  - The origin rect in the parent's box model
- subcontrol-position
  - The alignment within the above rect

```
QPushButton::menu-indicator {
    subcontrol-origin: content;
    subcontrol-position: right center;
    image: url(menu.png);
}
```

Sub control positioning



Positioning Modes

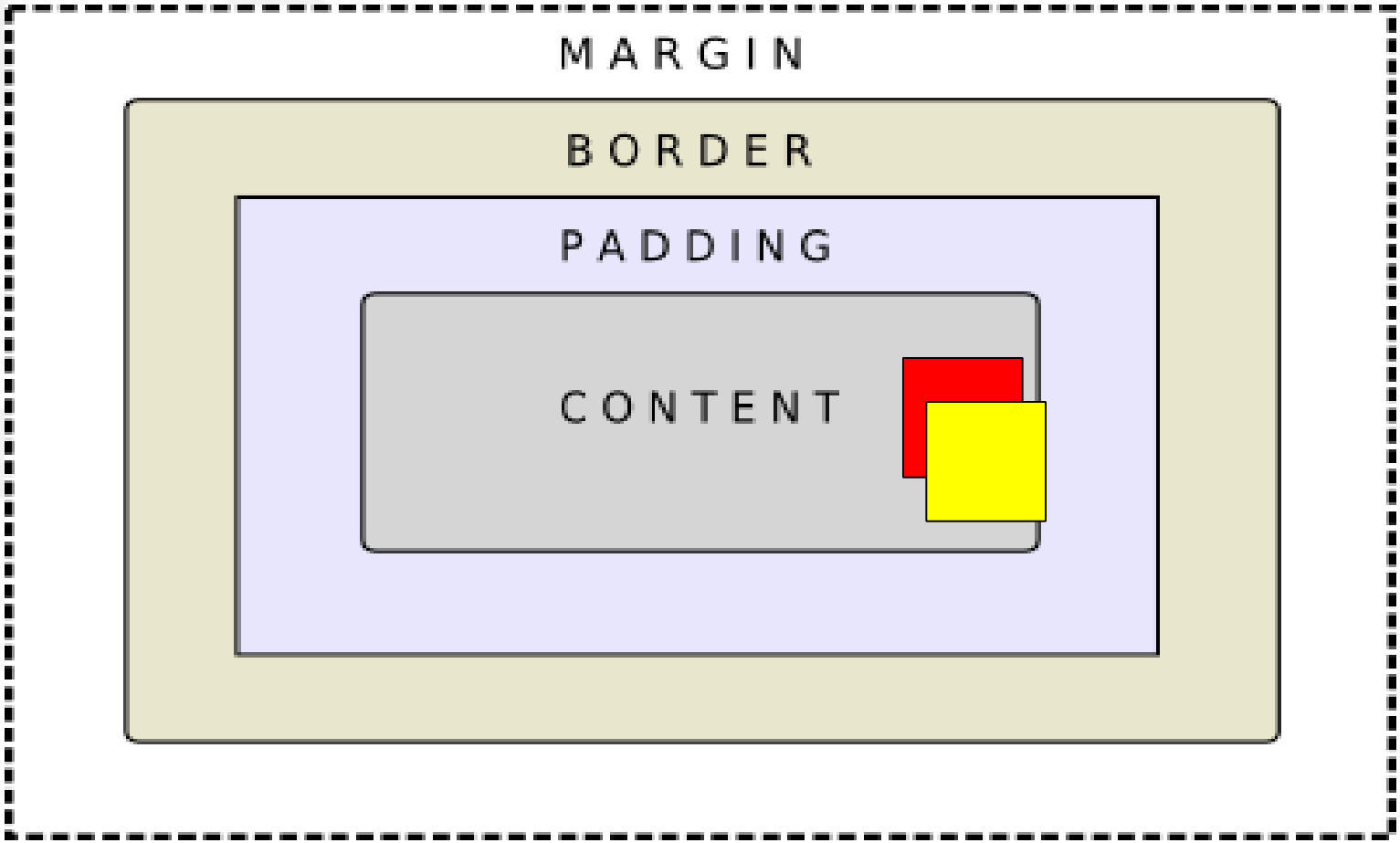
- Fine tune position using position modes
  - Relative Positioning
  - Absolute Positioning

Relative Positioning

- Use top, left, right, bottom to move a sub control from its current position

```
QPushButton::menu-indicator {
    subcontrol-origin: content;
    subcontrol-position: right center;
    image: url(menu.png);
    position: relative;
    top: 5px; left: 4px;
}
```

Relative Positioning

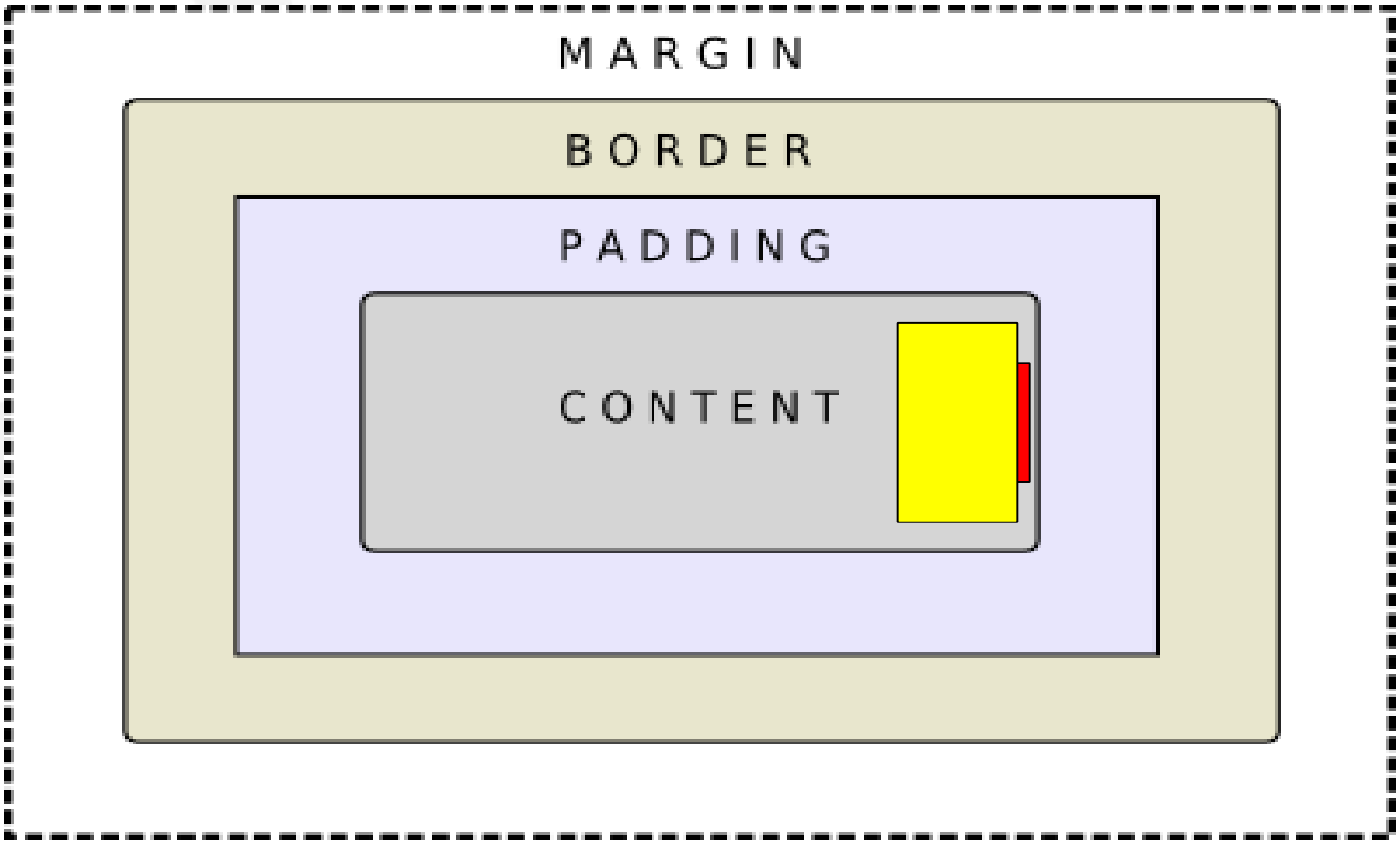


Absolute Positioning

- Absolute Positioning
  - Use top, left, right, bottom to define absolute positions within the origin rectangle

```
QPushButton::menu-indicator {
    subcontrol-origin: content;
    subcontrol-position: right center;
    image: url(menu.png);
    position: absolute;
    width: 20px;
    top: 5px; right: 2px; bottom: 5px;
}
```

Absolute Positioning



Advanced features

- You can set any Q_PROPERTY using style sheets
  - QlistView { qproperty-alternatingRowColors: true; }
- Dynamic properties can be used in the selector
  - QObject::setProperty can be used to create properties dynamically
- Can customize various style hints

  "activate-on-singleclick", "button-layout", "gridline-color", "lineedit-password-character", "messagebox-text-interaction-flags", "opacity", "show-decoration-selected"

Negative Margins

- Used to make the element grow outward

Qt 4.3's well kept secret

- Demo

Future direction

- Customizable icons and icon size
  - Already in 4.4 snapshots!
- Make all widgets styleable
  - QDockWidget, QMdiSubWindow already in snapshots!
- Support Mac
- Support for RTL
- Support custom QStyle subclasses
- QstyleItemDelegate
  - Already in snapshots!

Qt Style Sheets and KDE

- Style Sheets work currently with built-int Qt styles
  - By following some guidelines, we make it work with any Qstyle
  - Making it work with KStyle imples that all KDE applications are automagically customizable with style sheets

More information

- Style Sheet Example
  - examples/widgets/stylesheet
- Style Sheet Documentation
  - http://doc.trolltech.com/4.3/stylesheet.html
- Labs
  - http://labs.trolltech.com/blogs/
  -
- Email
  - Girish Ramakrishnan (ramakrishnan.girish@trolltech.com)
  - Jens Bache Wiig (jbache@trolltech.com)