Styles in Qt and KDE: A new approach

Eduardo Madeira Fleury
July 3rd, 2010

openBossa
Software in a creative way

Who are we?

Nokia research institute in Brazil – INdT

openBossa – FOSS stream at INdT

Collaborating with Qt Software / KDE

Qt Kinetic Qt Webkit

AnchorLayout Plasma Netbook

QtQuick Components (QML)

Styling

Background

Qt 4 on the road since 2005

QWidgets widely used but aging

New technologies QGraphicsView and QtQuick

Should we port old widgets to newer canvas?

Avoid code duplication

Was time for the styling system to evolve

Agenda

About Styles

What can be improved

A new approach

Interaction with QtQuick

Conclusion

ABOUT STYLES

What are Styles

Classes that handle the painting of widgets

Provides separation between widget logic and painting

Allow the same widgets to have different looks (KDE Styles, Gnome, Mac, Win, etc)

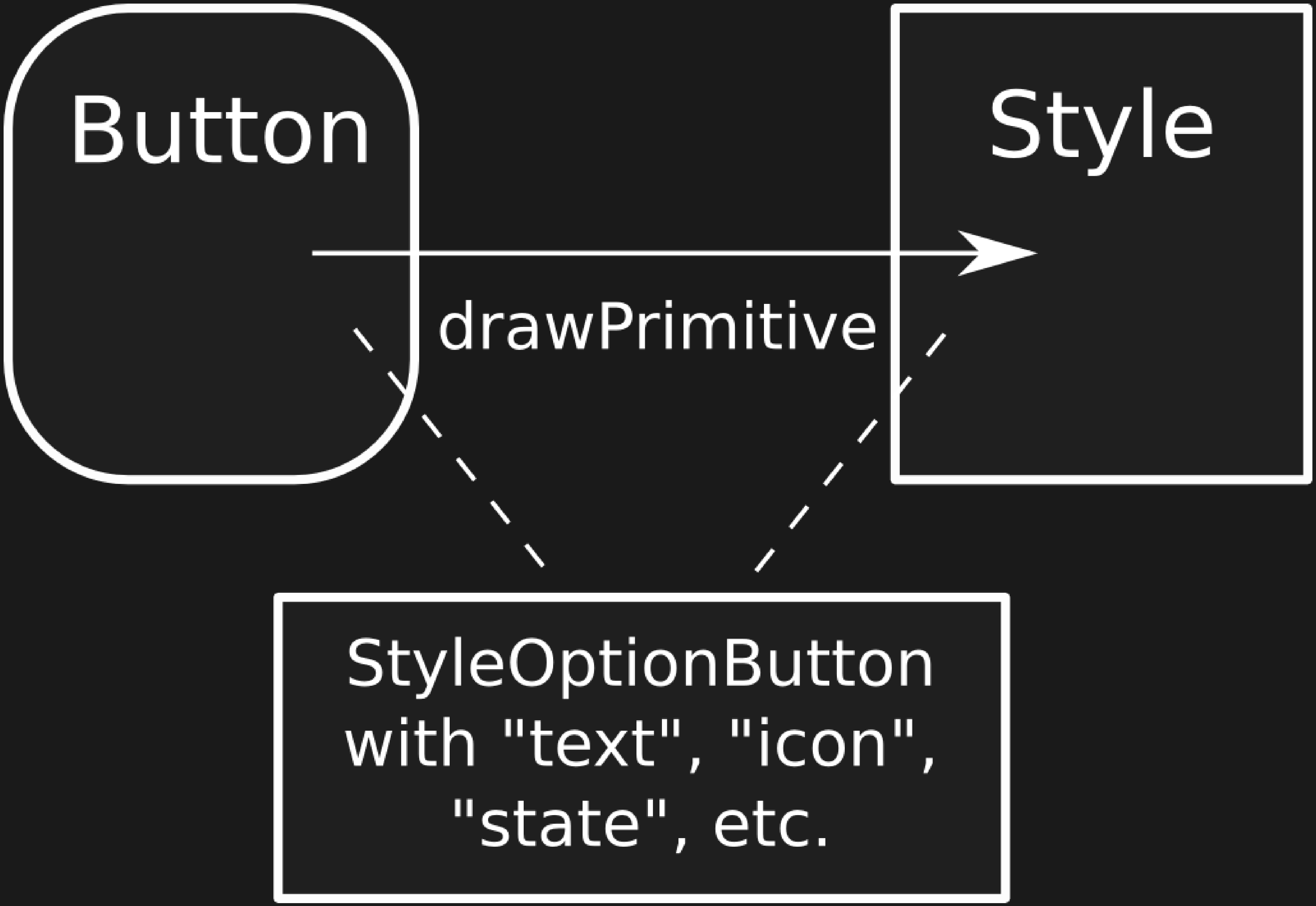
Styles in Qt and KDE

QStyle subclasses

Able to draw pieces of different widgets

Widgets delegate their painting to classes that implement the QStyle interface

Styles in Qt and KDE



Button

Style

drawPrimitive

StyleOptionButton with "text", "icon", "state", etc.

WHAT CAN BE IMPROVED

What can be improved

Procedural painting

(bottleneck when targetting high FPS animations)

Ability to customize **look and feel**

(flexibility to also reach mobile touch interfaces)

NEW APPROACH

Requirements

Fit current and future canvasses

Provide an alternative to procedural paiting

Empower designers to implement their ideas

Solution

Primitives-Graph

Populating of Widgets

Property Binding

Event-handling primitives

Respect to public API

Primitives-Graph

Create small building blocks, or primitives

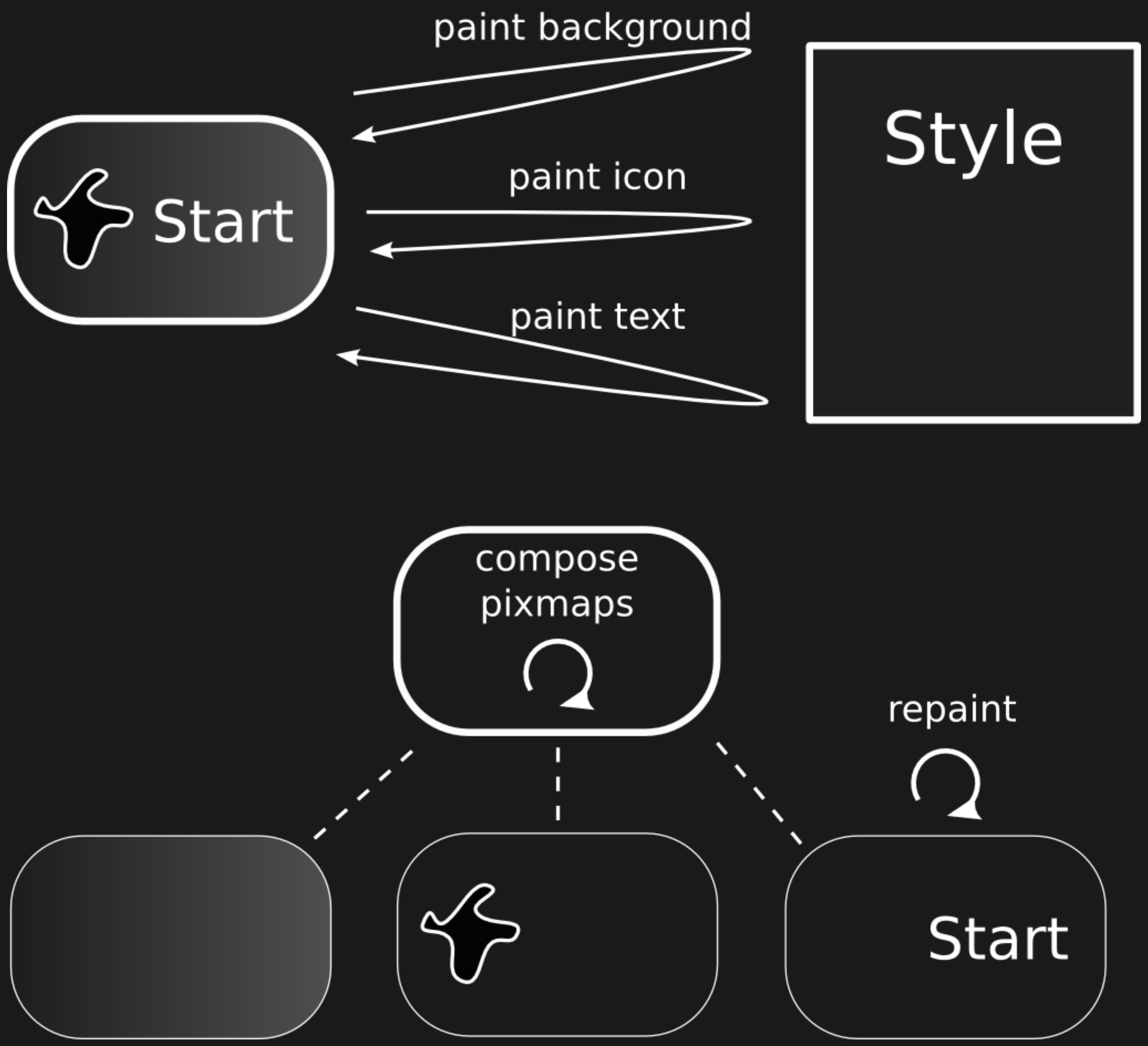
Primitives do the painting

Widgets are represented as a set of primitives

Widgets do no painting

Primitives-Graph

Cost of changing the text of a button



Populating Widgets

Current widgets call Style to paint well defined parts (background, text, etc)

New approach: Styles get empty widgets and populate them with primitives

Property Binding

Painting depends on widget status

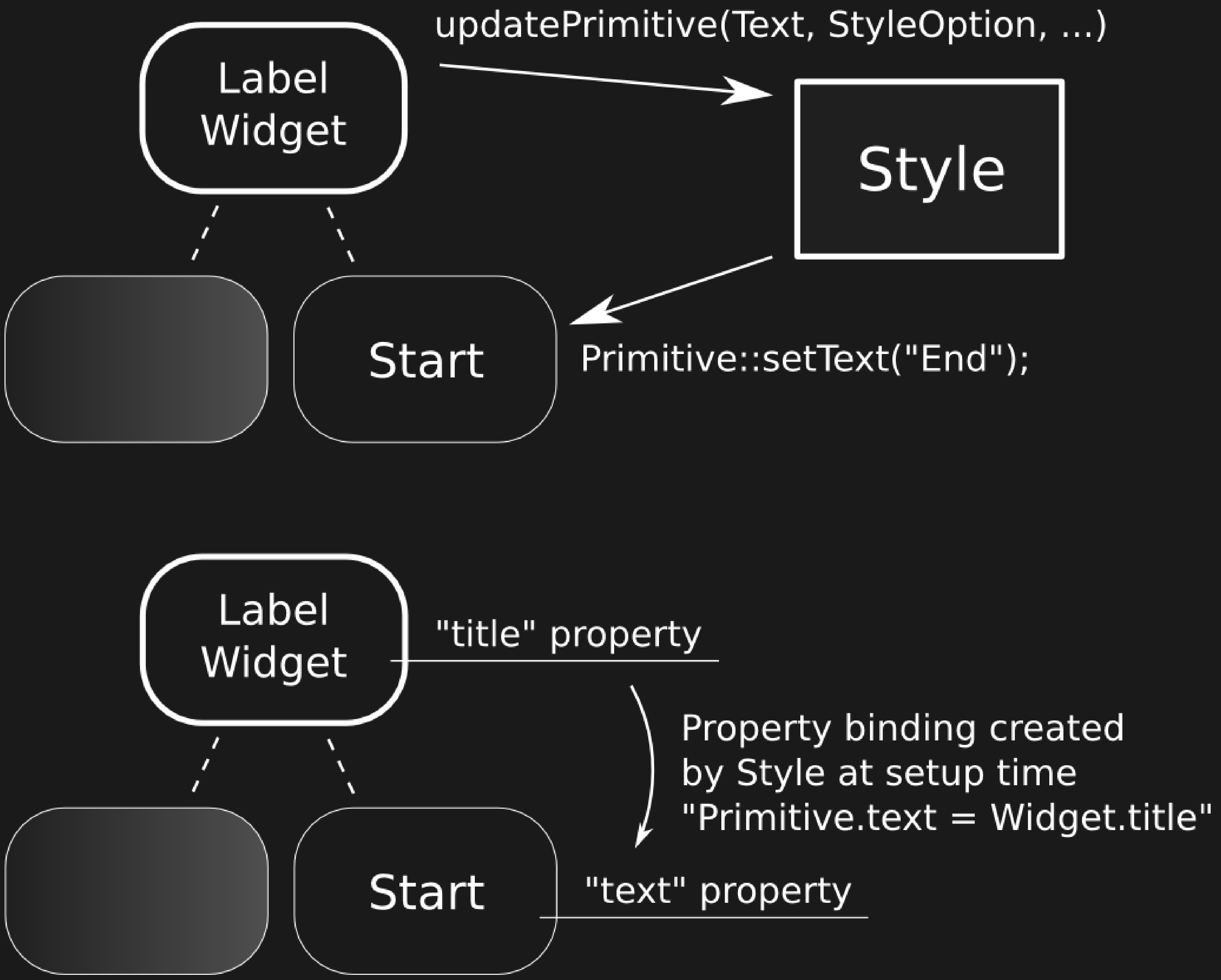
Communication between widget and style

Former solution was to pass data structures

New approach: binding widgets and primitive properties together

Property Binding

Data flow between widget and primitive



Event-handling Primitives

Current QWidgets do all event-handling

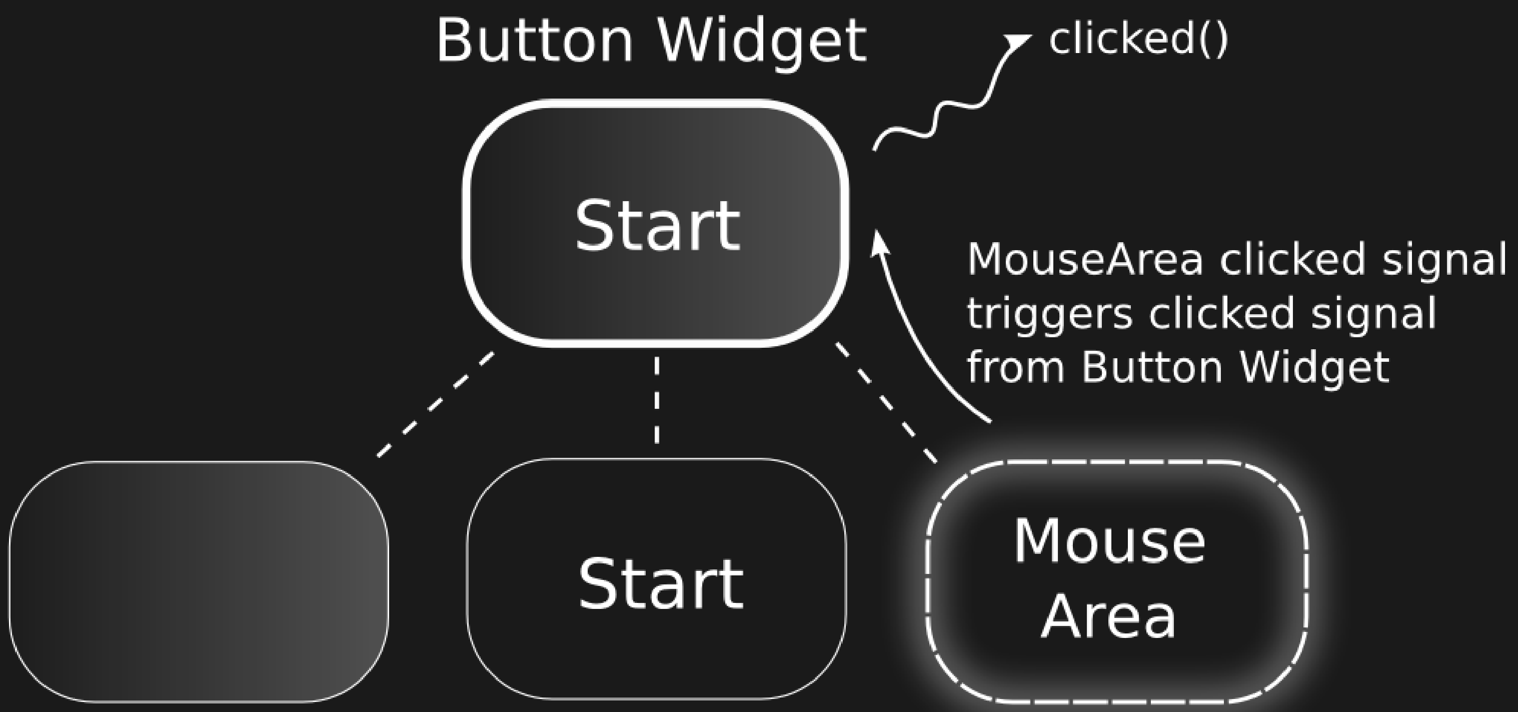
Do not allow for customization

New approach: add Event-handling primitives

Allow for Styles to customize widget behaviour

Event-handling Primitives

Event handling primitive used in Button Widget



Respect to public API

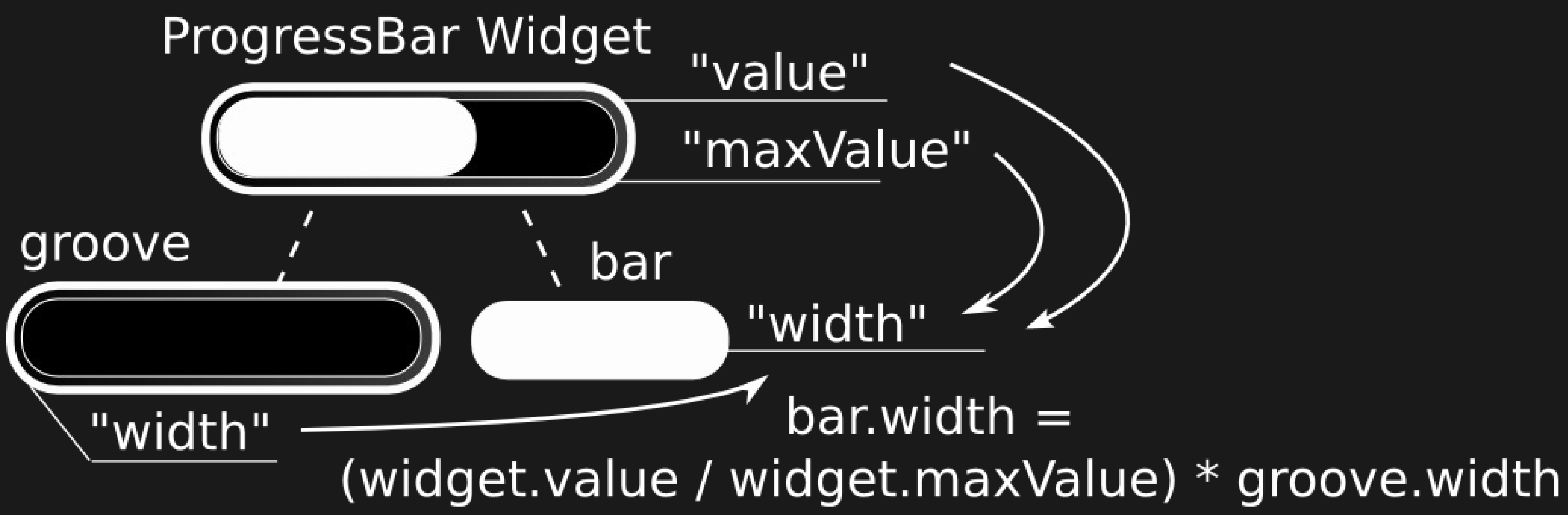
If look and behaviour can be changed, what is left to define a Widget?

New approach: The public API of a widget needs to be respected in a consistent way

API is exported as properties that are bound to the primitives

Respect to public API

Data binding in ProgressBar Widget
made of two primitives



ProgressBar Widget

"value"

"maxValue"

groove

bar

"width"

"width"

bar.width =
(widget.value / widget.maxValue) * groove.width

QtQuick INTEGRATION

Interaction with QtQuick

QtQuick is a tool to describe Graphic User Interfaces in a declarative way

(in opposition to the standard way of imperativelly creating widgets)

Allows for faster creation of fluid GUIs

Shortens the gap between designers and developers

Interaction with QtQuick

Should my interface use custom QtQuick components or standard, native looking, widgets ?

Custom sexy looks or native familiar experience?

Differentiate between what is the UI core and what is support.

Styleable widgets in QtQuick

Easy of use of QML layouting

Platform consistency with native-looking widgets

Export C++ widgets that use the Style interface behind the scenes

Primarily for the support part of QML interfaces

QtQuick as a widget styling tool

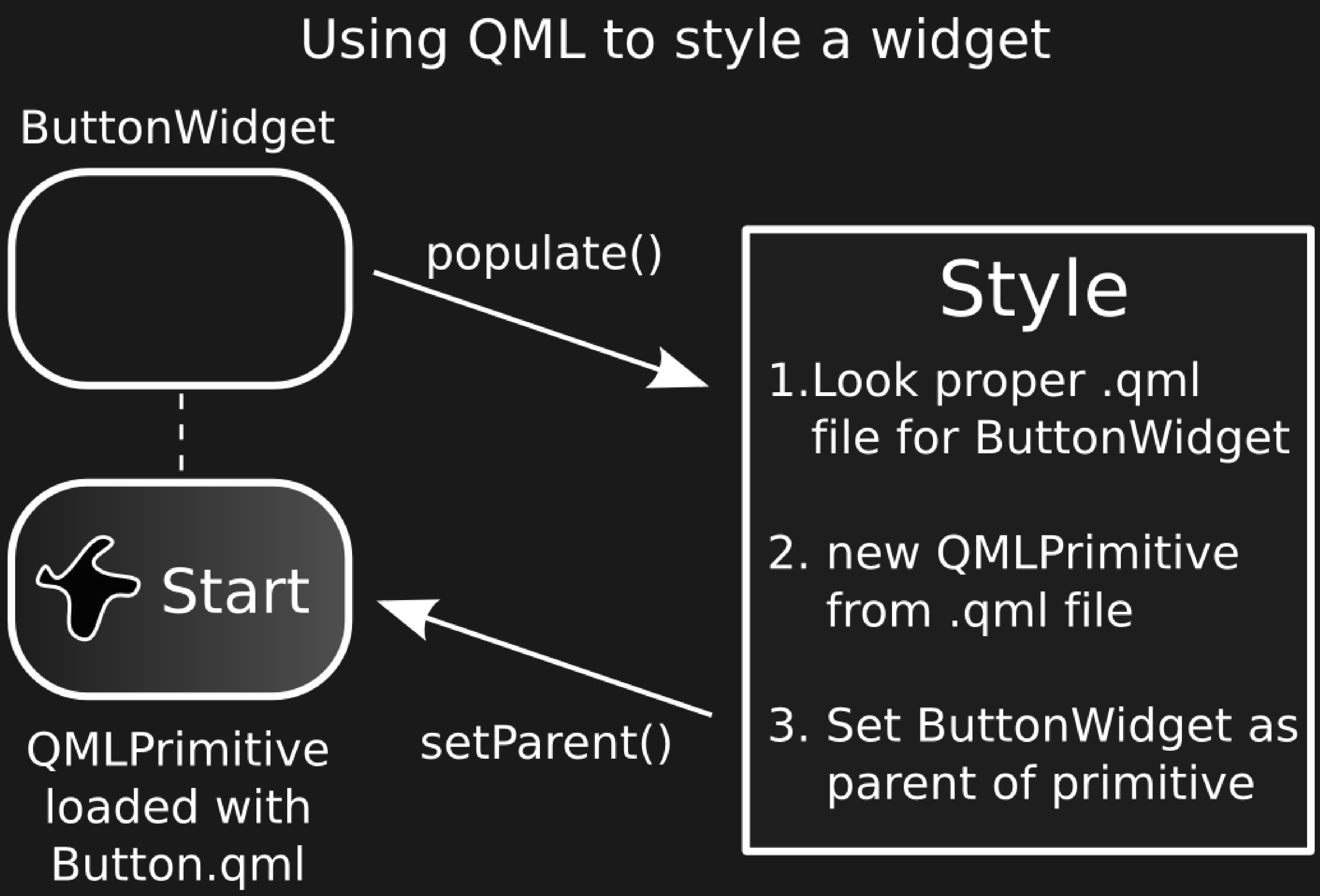
Current workflow requires C++ developers to implement styles as directed by designers

Some designers are able to use QML themselves

What if designers could change the looks of all existing KDE applications by using QML only?

KDE app developers do not need to leave C++

QtQuick as a widget styling tool



Using QML to style a widget

ButtonWidget

populate()

Style

1. Look proper .qml file for ButtonWidget
2. new QMLPrimitive from .qml file
3. Set ButtonWidget as parent of primitive

setParent()

Start

QMLPrimitive loaded with Button.qml

Conclusion

Solution relies on QObject properties and data binding, concepts similar to QML

Does not rely on specific canvas implementation

Tries to use few high level concepts

Not much C++ magic or machinary

On going work

Still in Proof of Concept stage

Can be implemented upstream in Qt or in KDE

Has been discussed in plasma-devel

More info

Check Akademy 2010 technical papers

fleury @ #qt-labs

gitorious.org/qt-components

http://eduardofleury.com

Thanks to friends at openBossa and Qt DF