Run Qt on Linux embedded systems using Yocto

Marco Cavallini, KOAN

http://koansoftware.com

Follow us on LinkedIn

Rights to copy

ftp://ftp.koansoftware.com/public/talks/QtDay-2019/QtDay2019-Koan.pdf

Abstract

This talk will give you all the information to run Qt applications on a real Linux embedded system using Yocto Project.

Will be discussed how to prepare the target system with all the needed Qt5 libraries, setup the ARM cross-compiler and finally run your beautiful Qt application on it.

The purpose of this talk is to show how to use Qt5 with Yocto Project on a real embedded hardware like Raspberry PI3 or similar like iMX6.

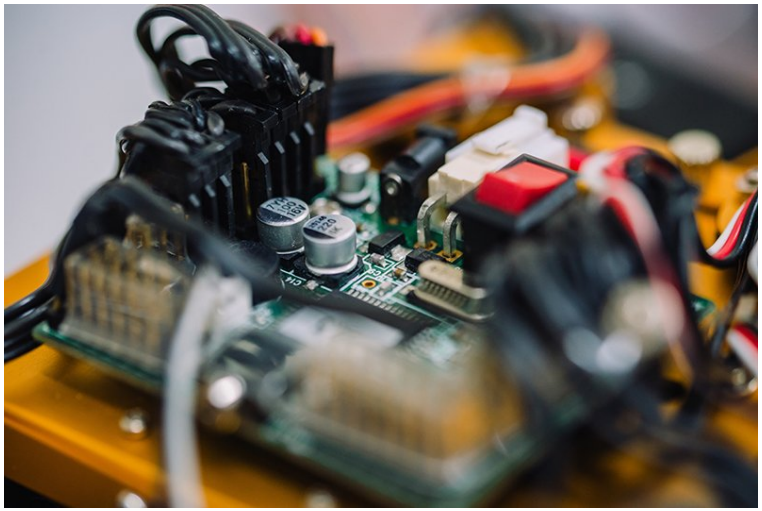
Agenda

- What is an embedded system
- Differences between a normal distro and embedded
- How to use Qt on embedded systems
- How to use Yocto Project
- Layer meta-qt5
- Yocto customization for Qt5
- Adding a custom layer to customize Qt5
- Deploy and debug on embedded system
- Demo on real hardware

Embedded systems

What is an embedded system

The term "embedded system" generically identifies electronic microprocessor systems designed specifically for a given application, often with an ad hoc hardware platform.



Embedded system software

- **Bare metal**
  - installed directly on hardware rather than within the host operating system

- **Operating system**
  - Android
  - Linux
  - Windows Embedded
  - VxWorks
  - FreeRTOS
  - Zephyr
  - Etc...

Linux distribution

Linux distribution

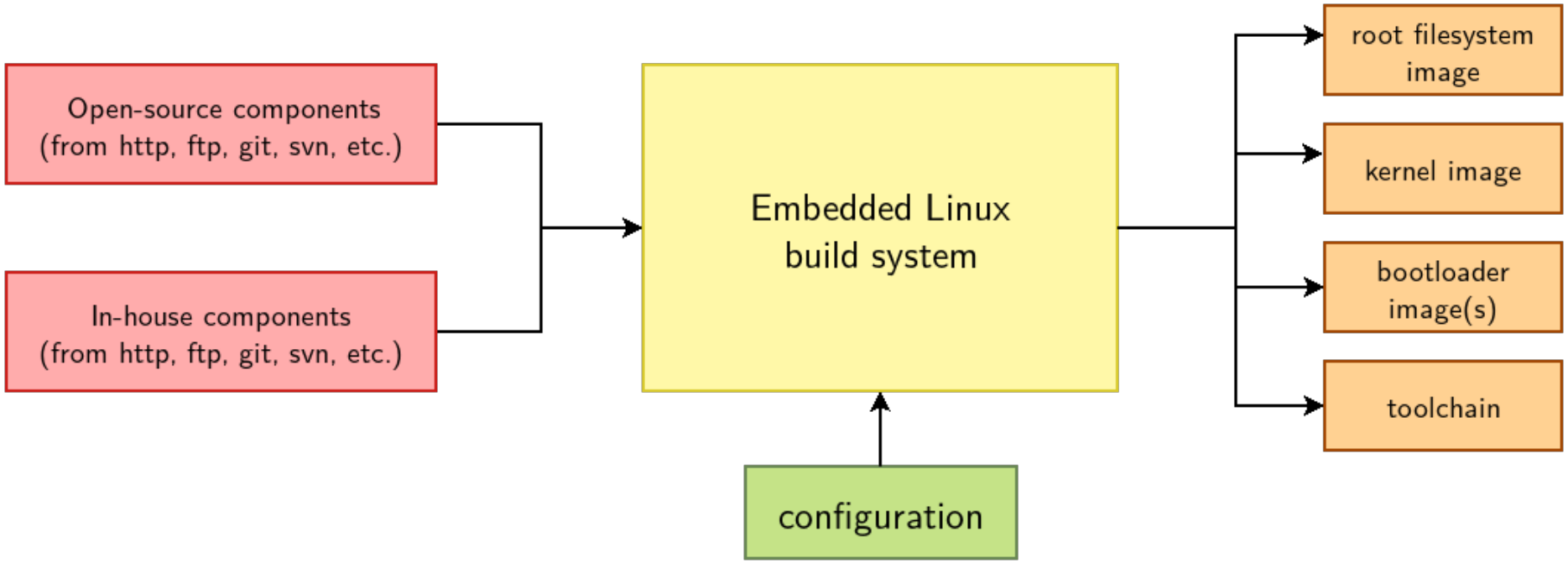
- **Linux distribution** (often abbreviated as **distro**)
  - operating system made from a software collection, which is based upon the Linux kernel and, often, a package management system.

- **Typical Linux distros** (not embedded)
  - Debian
  - Ubuntu and (*)ubuntu
  - Red Hat
  - Fedora
  - ArchLinux
  - Etc...

System integration possibilities

	Pros	Cons
Building everything manually	Full flexibility Learning experience	Dependency hell Need to understand a lot of details Version compatibility Lack of reproducibility
Binary distribution Debian, Ubuntu, Fedora, etc.	Easy to create and extend	Hard to customize Hard to optimize (boot time, size) Hard to rebuild the full system from source Large system Uses native compilation (slow) No well-defined mechanism to generate an image Lots of mandatory dependencies Not available for all architectures
Build systems Yocto, Buildroot, PTXdist, etc.	Nearly full flexibility Built from source: customization and optimization are easy Fully reproducible Uses cross-compilation Have embedded specific packages not necessarily in desktop distros Make more features optional	Not as easy as a binary distribution Build time

Build system workflow



Qt on embedded system

Get Qt

With Qt, you can reach all your target platforms
– desktop & embedded –

Qt is available under a **dual-licensing model**
– you choose what's right for your needs.

- Commercial
- Open Source *(We will cover only this one)*

https://www.qt.io/download

Open Source Rights & Obligations

Qt Usage under (L)GPL v3 license

- Must provide a relinking mechanism for Qt libraries
- Must provide a license copy & explicitly acknowledge Qt usage
- Must make a Qt source code copy available for customers
- Qt source code modifications aren't proprietary
- Must make "open" consumer devices
- For Digital Rights Management please see Qt FAQs
- Special consideration should be taken when attempting to enforce software patents

Additional features and Tools

Qt has it's own cross-platform IDE d UIs once and deploying them across multiple operating systems.
Some features are limited to GPL

(Here we don’t care about Workstation side features and tools)

Embedded tooling & solutions

Qt has **ready-made** solutions that speed up your device creation.

Their setup and management may be difficult.

- Direct on-device debugging
- One-click deployment
- Qt sources for DIY embedded software stack

  We are going to learn how to use them…

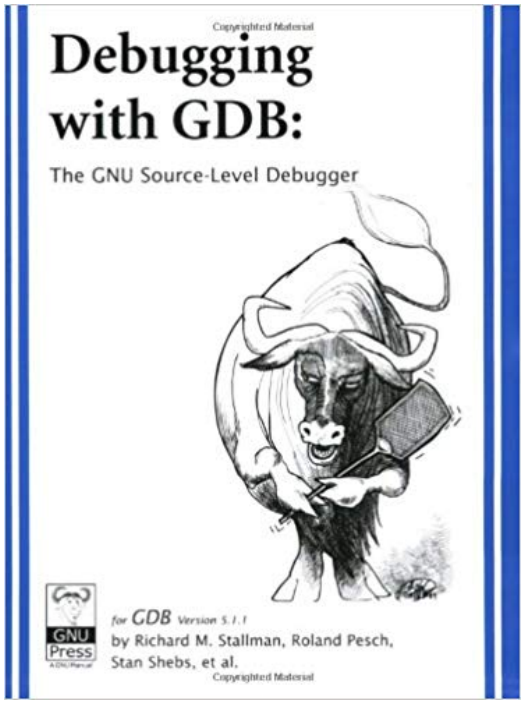
- Reference software stack (Yocto based **Boot2Qt**) & SDK *
- Qt Configuration tool (Qt Lite) *
- Qt Virtual Keyboard (license GPL only) *

  **(Not covered here)*

Remote debugging

Direct on-device debugging

- Available through the Yocto Project toolchain
- Cross debugging

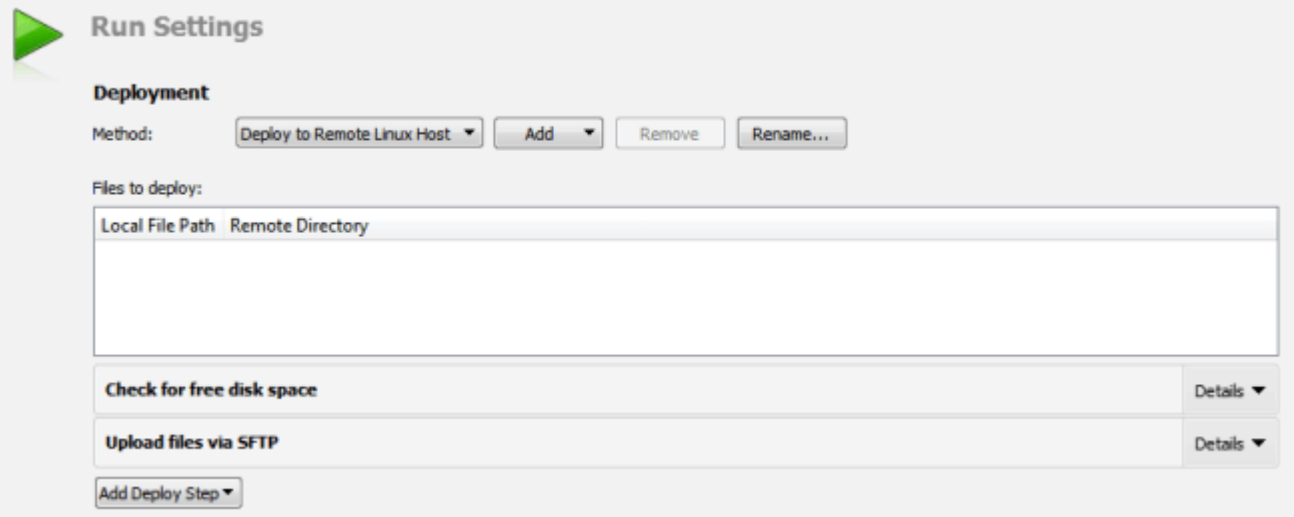


Target deployment (1/2)

One-click deployment on embedded target

- Edit the qmake **INSTALLS** variable in the **project .pro**

```
target.path = /home/root
INSTALLS += target
```



Target deployment (2/2)

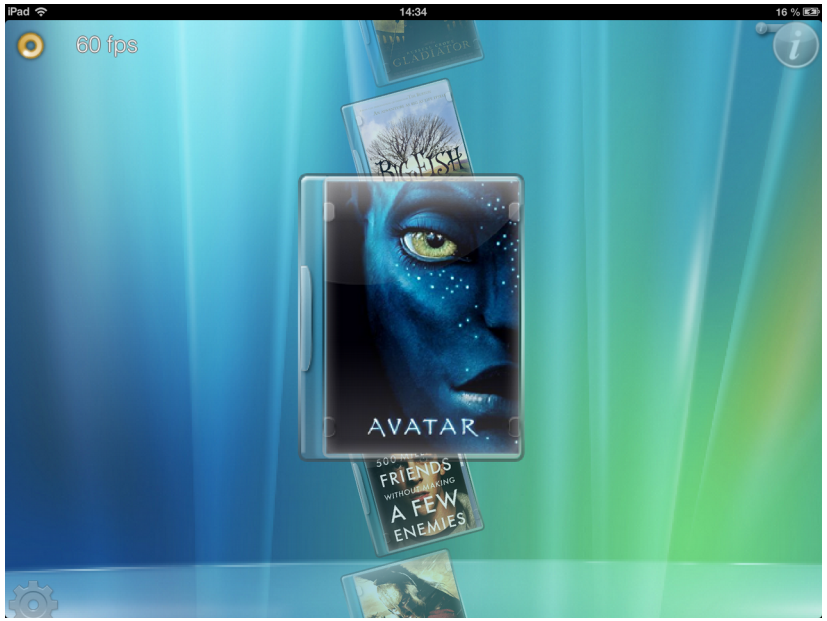
One-click deployment on embedded target

- By default, Qt Creator copies the application files to the device by using the SSH file transfer protocol (SFTP)
- Available through the Yocto Project configuration
- SSH virtual package : **dropbear** vs. **openssh**

Yocto standard software stack

Qt sources for ‘DIY’ embedded software stack

- Available for the Yocto Project
- Layer : **meta-qt5**



https://github.com/meta-qt5/meta-qt5

Boot2Qt software stack

Reference software stack (Yocto based **Boot2Qt**) & SDK

https://doc.qt.io/QtForDeviceCreation/qtb2-index.html

- Boot to Qt (b2qt) is the reference distro used in Qt for Device Creation.
- It combines **Poky**, **meta-qt5** and various BSP meta layers to provide an integrated solution for building device images and toolchains with the latest Qt version.

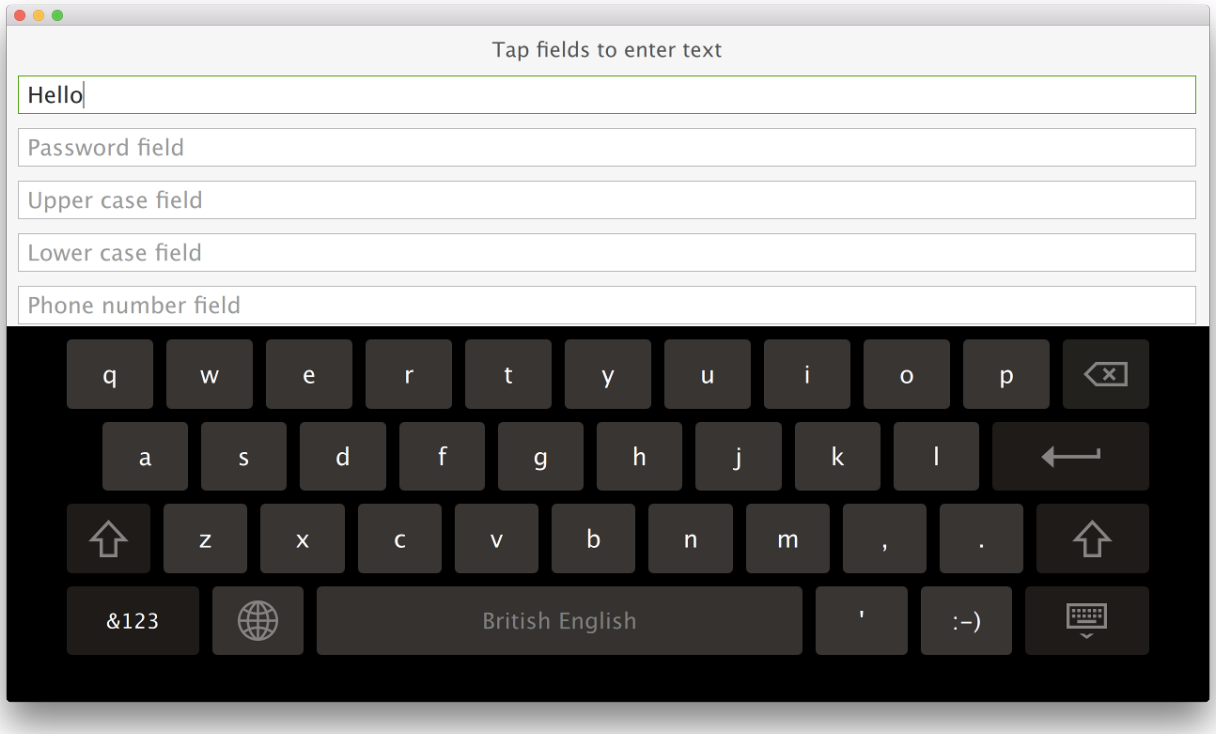
- Available for the Yocto Project under commercial license
- Layer : **meta-boot2qt**

https://code.qt.io/cgit/yocto/meta-boot2qt.git/

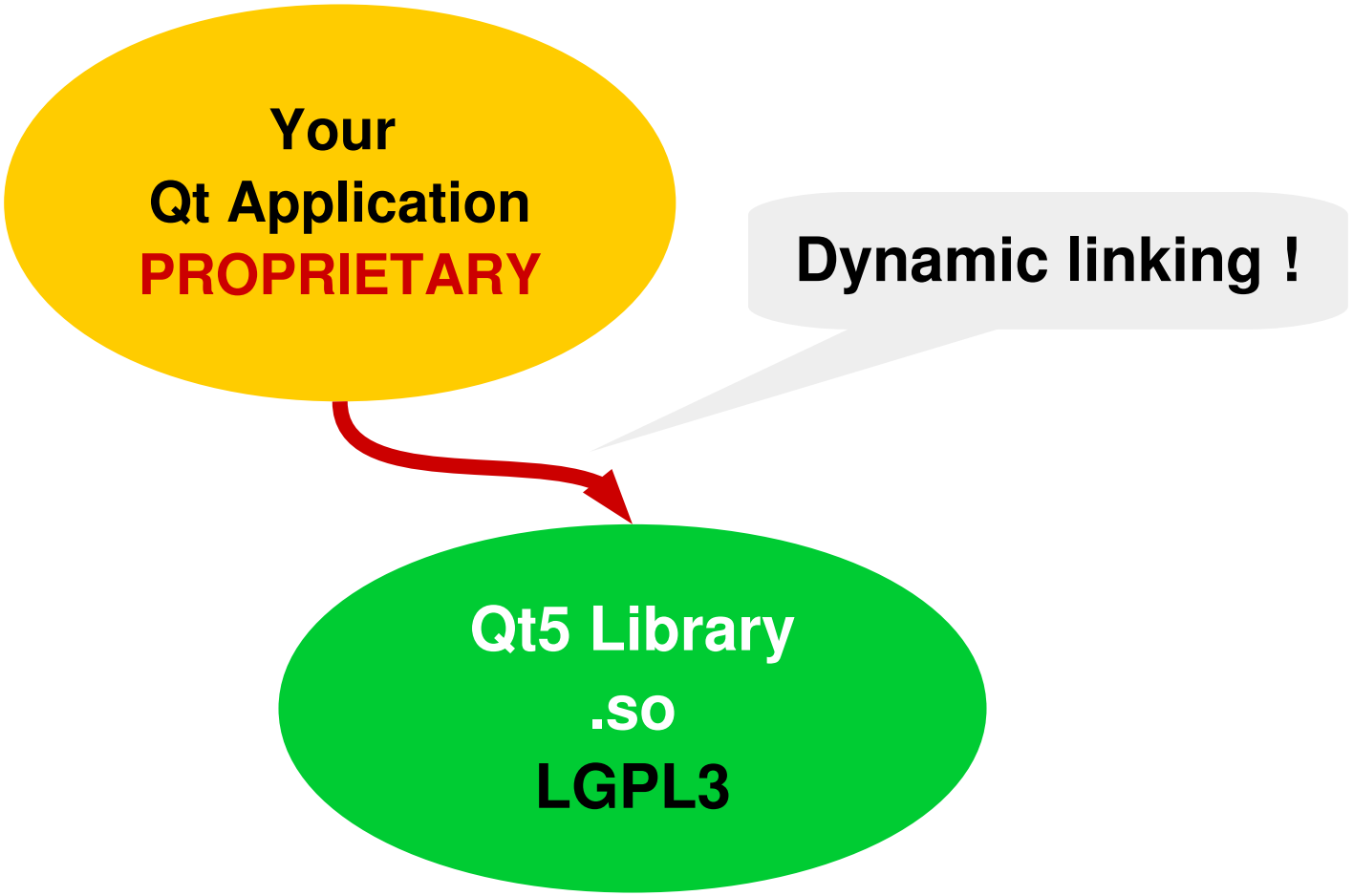
QtVirtualKeyboard

Qt Virtual Keyboard (license GPL only)

```
int main(int argc, char *argv[])
{
    qputenv("QT_IM_MODULE", QByteArray("qtvirtualkeyboard"));
```



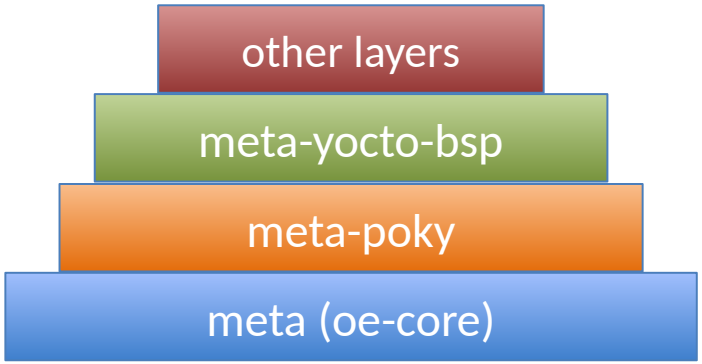
How to use LGPL libraries



How to use Yocto project

What is Yocto Project ?

- **Collection of tools and methods enabling**
  - Rapid evaluation of embedded Linux on many popular off-the-shelf boards
  - Easy customization of distribution characteristics
- **Supports x86, ARM, MIPS, PowerPC**
- **Based on technology from the OpenEmbedded Project**
- **Layer architecture allows for easy re-use of code**



Yocto Project overview

- **YP builds packages - then uses these packages to build bootable images**
- Supports use of popular package formats including:
  - rpm, deb, ipk
- Releases on a 6-month cadence
- Latest (stable) kernel, toolchain and packages, documentation
- App Development Tools including Eclipse plugin, SDK, Toaster

Yocto release versions

Name	Revision	Poky	Release Date
Bernard	1.0	5.0	Apr 5, 2011
Edison	1.1	6.0	Oct 17, 2011
Denzil	1.2	7.0	Apr 30, 2012
Danny	1.3	8.0	Oct 24, 2012
Dylan	1.4	9.0	Apr 26, 2013
Dora	1.5	10.0	Oct 19, 2013
Daisy	1.6	11.0	Apr 24, 2014
Dizzy	1.7	12.0	Oct 31, 2014
Fido	1.8	13.0	Apr 22, 2015
Jethro	2.0	14.0	Oct 31, 2015
Krogoth	2.1	15.0	Apr 29, 2016
Morty	2.2	16.0	Oct 28, 2016

Name	Revision	Poky	Release Date
Pyro	2.3	17.0	Apr, 2017
Rocko	2.4	18.0	Oct, 2017
Sumo	2.5	19.0	Apr, 2018
Thud	**2.6**	**20.0**	**Oct, 2018**
Warrior	2.7	21.0	Apr, 2019
Zeus	2.8	22.0	Oct, 2019
			Apr, 2020
			Oct, 2020
			...

The build engine: bitbake

- **bitbake**
  - Powerful and flexible build engine (Python)
  - Reads metadata
  - Determines dependencies
  - Schedules tasks

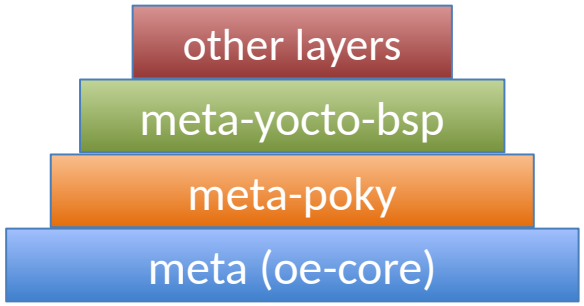
Metadata – a structured collection of "recipes" which tell BitBake what to build, organized in layers

What is Poky

- **Poky is a reference distribution**
- **Poky has its own git repo**

```
git clone git://git.yoctoproject.org/poky
```

- **Primary Poky layers**
  - oe-core (poky/meta)
  - meta-poky (poky/meta-poky)
  - meta-yocto-bsp



- **Poky is the starting point for building things with the Yocto Project**

BSP layers

- **A BSP Layer allows to support a specific hardware**
- **List of available layers**

  http://git.yoctoproject.org - http://layers.openembedded.org

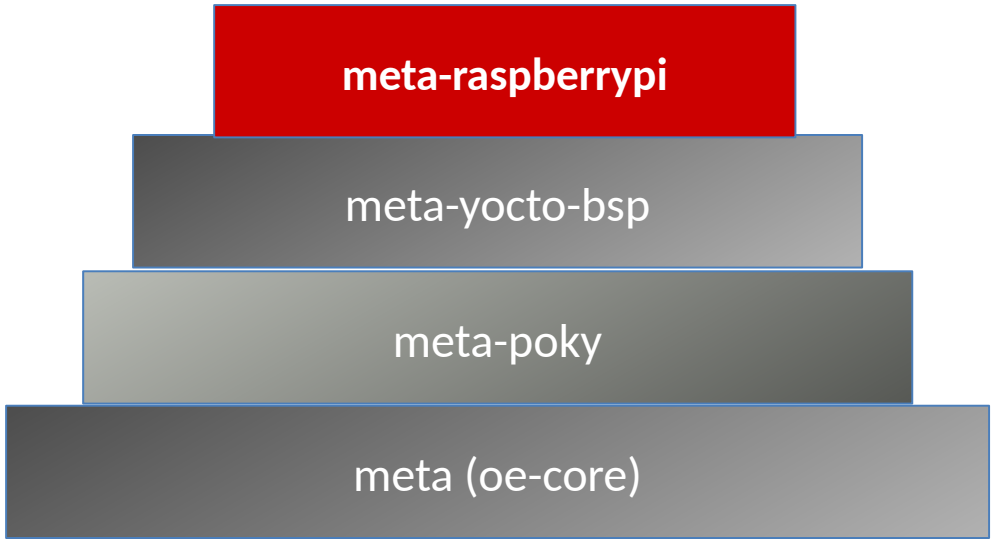
-Yocto Metadata Layers

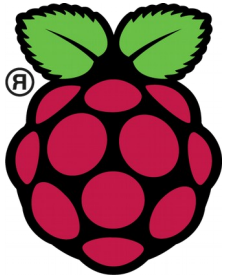
intel-iot-refkit	Iot Reference OS Kit for Intel
meta-alexa-demo	Layer containing recipes to add an Alexa client to an image
meta-amd	Layer containing AMD hardware support metadata
meta-anaconda	Layer to provide Anaconda installer
meta-axxia	Layer for LSI's family of Axxia mobile & enterprise communication processors
meta-cgl	Enable Carrier Grade Linux compliance through a reference Linux distribution
meta-cloud-services	Provides the packages and images for Cloud compute, control and storage nodes (O...
meta-dpdk	Data Plane Developer Kit (dpdk.org)
meta-external-toolchain	OE/Yocto external toolchain support layer
meta-freescale	Layer containing NXP hardware support metadata
meta-gplv2	GPLv2 versions of software where upstream has moved to GPLv3 licenses
meta-intel	Layer containing Intel hardware support metadata
meta-intel-clear-containers	Layer enabling Intel Clear Containers
meta-intel-contrib	Intel hardware support extras
meta-intel-iot-middleware	Shared middleware recipes for Intel IoT platforms
meta-intel-qat	Intel Quick Assist Technology Layer
meta-ivi	Collection of software related to In-Vehicle Infotainment systems
meta-java	Layer containing recipes for OpenJDK and other open source Java- related compone...
meta-maker	Layer supporting applications and tools for Makers in OE
meta-mentor	Layer containing Mentor Graphics support metadata
meta-mingw	Layer for mingw based SDKs
meta-mingw-contrib	Mingw based SDKs extras
meta-mono	Metadata layer to build the Mono runtime
meta-oic	Layer containing recipes for building the Open Interconnect Consortium Iotivity ...
meta-qcom	Layer containing Qualcomm hardware support metadata
meta-qt3	Qt3 layer for supporting LSB Testing and Compliance
meta-qt4	Qt4 layer for supporting LSB Testing and Compliance
meta-raspberrypi	Hardware specific BSP overlay for the RaspberryPi device

BSP layer for Raspberry PI

- **Example of a BSP Layer to support Raspberry PI**

```
git clone git://git.yoctoproject.org/meta-raspberrypi
```

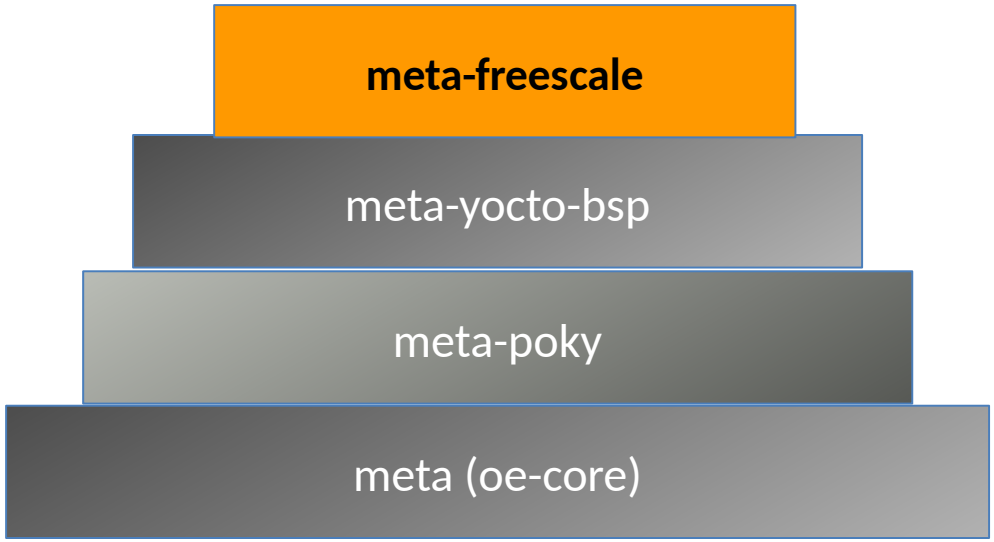




BSP layer for Freescale/NXP

- **Example of a BSP Layer to support Freescale eval boards**

```
git clone git://git.yoctoproject.org/meta-freescale
```



Case study (1/3)

- Case study of a custom BSP layer
- Managed using repo
- Example of a BSP Layer to support Freescale eval boards

https://github.com/koansoftware/koan-toradex-bsp-repo

```
mkdir yocto-koan-toradex
cd yocto-koan-toradex

repo init -u \
https://github.com/koansoftware/koan-toradex-bsp-repo

repo sync
```

Case study (2/3)

```
<?xml version="1.0" encoding="UTF-8"?>

<!--  koan-toradex-bsp-repo -->

<manifest>

  <default sync-j="4" revision="rocko"/>

  <remote fetch="https://git.yoctoproject.org/git" name="yocto"/>
  <remote fetch="https://github.com/koansoftware" name="koan"/>
  <remote fetch="https://github.com/openembedded" name="oe"/>
  <remote fetch="https://github.com/meta-qt5" name="qt5"/>

  <remote fetch="http://github.com/Freescale" name="githf"/>
  <remote fetch="http://git.toradex.com" name="tdx"/>

  <project name="meta-freescale.git" path="sources/meta-freescale" remote="githf" revision="rocko"/>
  <project name="meta-freescale-3rdparty.git" path="sources/meta-freescale-3rdparty" remote="githf" revision="rocko"/>
  <project name="meta-toradex-nxp.git" path="sources/meta-toradex-nxp" remote="tdx" revision="rocko"/>
  <project name="meta-toradex-bsp-common.git" path="sources/meta-toradex-bsp-common" remote="tdx" revision="rocko"/>

  <project remote="yocto" revision="rocko" name="poky" path="sources/poky"/>
  <project remote="oe" revision="rocko" name="meta-openembedded" path="sources/meta-openembedded"/>
  <project remote="qt5" revision="rocko" name="meta-qt5" path="sources/meta-qt5"/>

  <project remote="koan" revision="master" name="koan-toradex-bsp-base" path="sources/base">
        <linkfile dest="README.md" src="README.md"/>
        <linkfile dest="setup-environment" src="setup-environment"/>
  </project>
</manifest>
```

Case study (3/3)

- **Project directory tree**

```
~/yocto-koan-toradex$ tree -L 2
.
├── README.md -> sources/base/README.md
├── setup-environment -> sources/base/setup-environment
└── sources
    ├── base
    ├── meta-freescale
    ├── meta-freescale-3rdparty
    ├── meta-openembedded
    ├── meta-qt5
    ├── meta-toradex-bsp-common
    ├── meta-toradex-nxp
    └── poky
```

- *We will see on the next slide what's the content of **meta-qt5**...*
- *... and later how to customize a qt image ...*

Layer meta-qt5

meta-qt5

- **Get the layer supporting Qt5**

```
git clone git://github.com/meta-qt5/meta-qt5.git
```

- **Layer dependency**
  - This layer depends on **meta-openembedded**

- When building stuff like **qtdeclarative**, **qtquick**, **qtwebkit**, make sure that you have required **PACKAGECONFIG** options enabled in **qtbase** build
- See **qtbase** recipe for details

meta-qt5

- **Using meta-qt5**

- You need to include the **meta-qt5** layer into your Yocto build environment.
- Editing the **conf/bblayers.conf** file and include path for meta-qt5.
- Then you can include any Qt module into your image or toolchain.

- By default, **meta-qt5** enables only a minimal set of features, thus you may need to customize it…

Yocto customization for Qt5

Qt Platform Abstraction

- The **Qt Platform Abstraction** (**QPA**) is the platform abstraction layer for Qt5

https://doc.qt.io/qt-5.12/embedded-linux.html

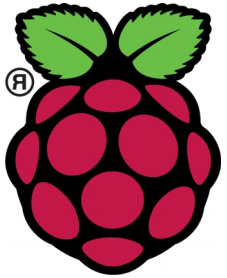
https://doc.qt.io/qt-5.12/qpa.html

*It replaced Qt for Embedded Linux and the platform ports from Qt4 that was using its own window system (**QWS**) implementation*

QPA plugins

- Since the Qt 5.0 release there are multiple platform plugins that are potentially usable on Embedded Linux systems:
  - **eglfs** - Uses the OpenGL ES in fullscreen mode. It has no concept of a window manager.
  - **linuxfb** - Uses the linux frame buffer in fullscreen mode. It has no concept of a window manager.
  - **directfb** - Uses the linux frame buffer with OpenGL ES via the directfb layer. Integrates into the directfb windowing.
  - **wayland** - provides a wayland platform plugin that allows Qt application to connect to a wayland compositor. Wayland is intended as a simpler replacement for X.
  - Other platforms are: xcb, offscreen, minimal

QPA - eglfs



- For iMX6 and RPI3
- enable **eglfs**, add in **local.conf**

```
DISTRO_FEATURES_remove = "X11 wayland"
```

- On the target machine enable Qt to run eglfs platform
  (in case editing /etc/profile)

```
export QT_QPA_PLATFORM=eglfs
```

- or when you run an application; you need to add

```
-platform eglfs
```

```
helloworld -platform eglfs
```

QPA - linuxfb

- For Atmel/Microchip At91SAMA5
- enable **linuxfb**, add in **local.conf**


```
DISTRO_FEATURES_remove = "opengl X11 wayland"
```

- On the target machine enable Qt to run linuxfb

```
export QT_QPA_PLATFORM=linuxfb:fb=/dev/fb0
```

- or when you run an application; you need to add

```
-platform linuxfb
```

```
helloworld -platform linuxfb
```

Adding a custom layer to customize Qt5

Create your layer (1/2)

- **Let's add our custom layer**

```
~/yocto-koan-toradex$ tree -L 2
.
├── README.md -> sources/base/README.md
├── setup-environment -> sources/base/setup-environment
└── sources
    ├── base
    ├── meta-freescale
    ├── meta-freescale-3rdparty
    ├── meta-openembedded
    ├── meta-qt5
    ├── meta-koan
    ├── meta-toradex-bsp-common
    ├── meta-toradex-nxp
    └── poky
```

Create your layer (2/2)

- **Create a custom layer**

```
bitbake-layers create-layer \
                    ~/yocto-koan-toradex/meta-koan
```

- **Add the layer to bblayers.conf**

```
bitbake-layers add-layer \
                    ~/yocto-koan-toradex/meta-koan
```

PACKAGECONFIG (1/3)

- Enable **OpenGL ES2** support

- Create a **qtbase_%.bbappend** file
- Then add the PACKAGECONFIG option as follows:

```
PACKAGECONFIG_append = " gles2"
```

- Very likely already included by your BSP layer

PACKAGECONFIG (2/3)

- Enable **SQLite** support
- Again, in a **qtbase_%.bbappend** file add the PACKAGECONFIG option as follows:

```
PACKAGECONFIG_append = " sql-sqlite"
```

PACKAGECONFIG (3/3)

- Other PACKAGECONFIG cases

```
PACKAGECONFIG[eglfs] = "-eglfs,-no-eglfs,drm"
PACKAGECONFIG[gl] = "-opengl desktop, ,
                                  virtual/libgl"
PACKAGECONFIG[gles2] = "-opengl es2, ,
                       virtual/libgles2 virtual/egl"

PACKAGECONFIG[tslib] = "-tslib,-no-tslib,tslib"
```

- PACKAGECONFIG syntax

```
PACKAGECONFIG[f1] = "--with-f1,--without-f1,
                           build-deps-f1,rt-deps-f1"
```

QtSerial

- Add **QtSerial** support

```
IMAGE_INSTALL_append = " qtserialport"
```

QtVirtualKeyboard

- Add **Qt Virtual Keyboard** support

```
IMAGE_INSTALL_append = " qtvirtualkeyboard"
```

Typical issues

- Imports for a QtQuick application

```
import QtQuick 2.4
import QtQuick.Controls 2.3
```

- A typical issue when running the QtQuick application

```
qrc:/MainWindowForm.ui.qml:2:1: module
"QtQuick.Controls" version 2.3 is not installed
```

- Means a missing components on the taget system

```
IMAGE_INSTALL_append = " qtquickcontrols \
   qtquickcontrols-qmlplugins \
   qtquickcontrols2 \
   qtquickcontrols2-qmlplugins”
```

Add your image

- Let's add a new Qt custom image

```
# koan-qt5-image proof of concept with Qt

DESCRIPTION = "A Qt5 minimal image by Koan"
LICENSE = "MIT"
inherit core-image

require recipes-core/images/core-image-minimal.bb

IMAGE_INSTALL_append = "
 packagegroup-core-ssh-openssh openssh-sftp-server \
 qtbase \
 qtquick1 qtdeclarative \
 qtquickcontrols qtquickcontrols2 \
 qtgraphicaleffects qtimageformats qtmultimedia \
 qtserialport"
```

Qt5 application development

Cross compiler

- Yocto can create a re-distributable cross-compiler

```
bitbake meta-toolchain
```

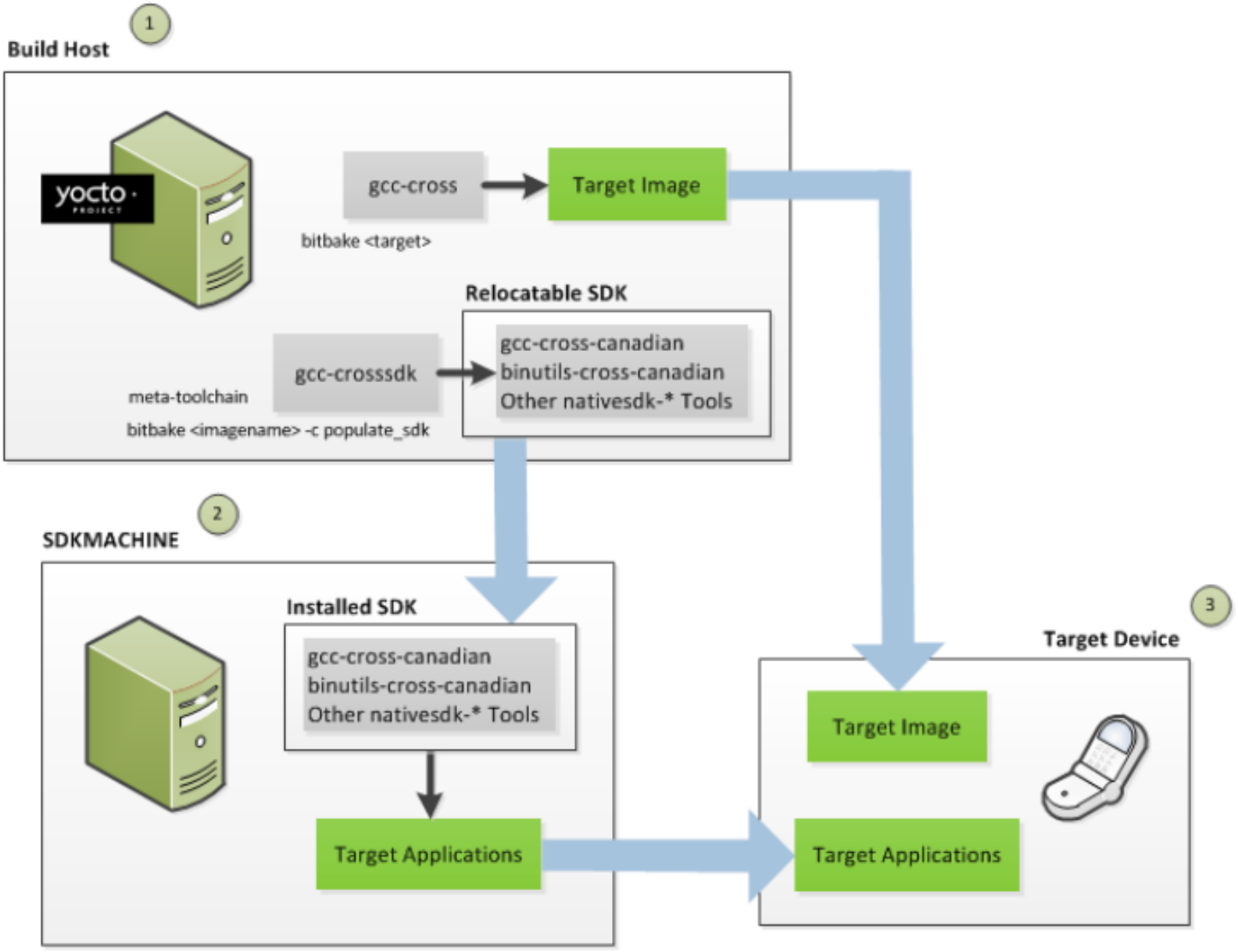
- Or a complete SDK for your target

```
bitbake -c populate_sdk <image-name>
```

- Or even an SDK for Qt5

```
bitbake meta-toolchain-qt5
```

Cross-Development Toolchain



Cross compiler

- Install it on any linux distribution

```
$ cd $HOME/poky/build/tmp/deploy/sdk

$ ./poky-glibc-x86_64-meta-toolchain-
cortexa8hf-vfp-neon-toolchain-2.4.sh
```

- Once installed you can use it setting the build environment

```
$ source /opt/poky/2.4/environment-setup-
cortexa8hf-vfp-neon-poky-linux-gnueabi
```

Cross-compile, Deploy and debug on embedded system

Cross compilation with QtCreator

- Prepare the cross compilation environment for **QtCreator**

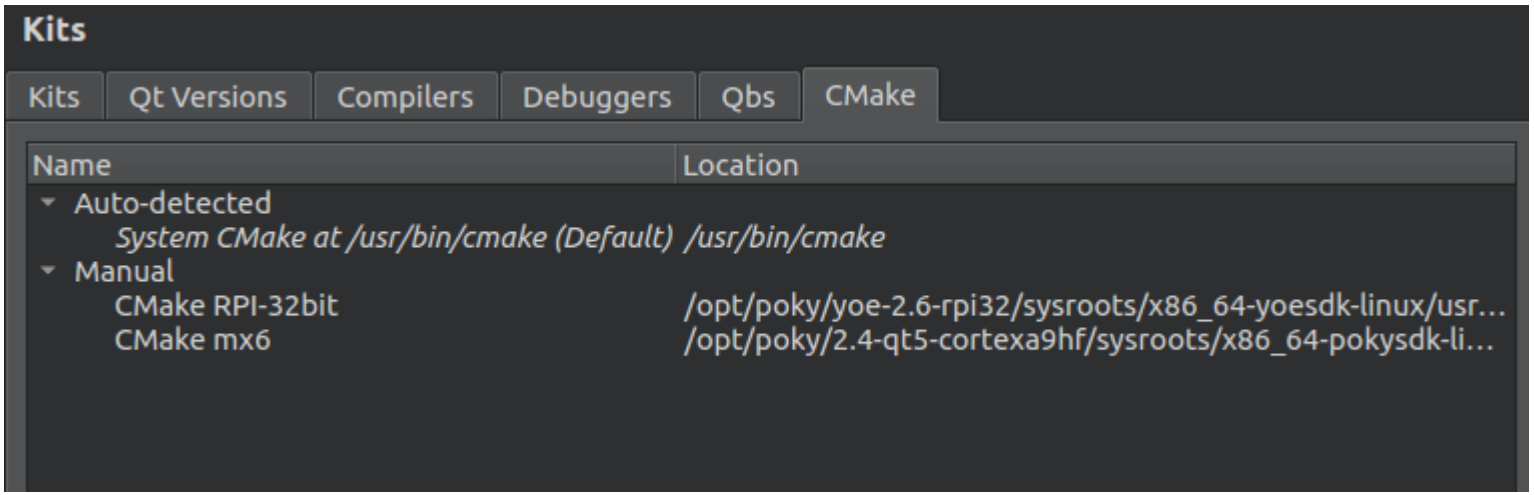
```
cd ~/Qt/Tools/QtCreator/bin
cp qtcreator.sh qtcreator-arm.sh
```

- Edit **qtcreator-arm.sh**

```
#! /bin/bash
. /opt/poky/2.4-qt5-cortexa9hf/environment-
setup-cortexa9hf-neon-poky-linux-gnueabi
```

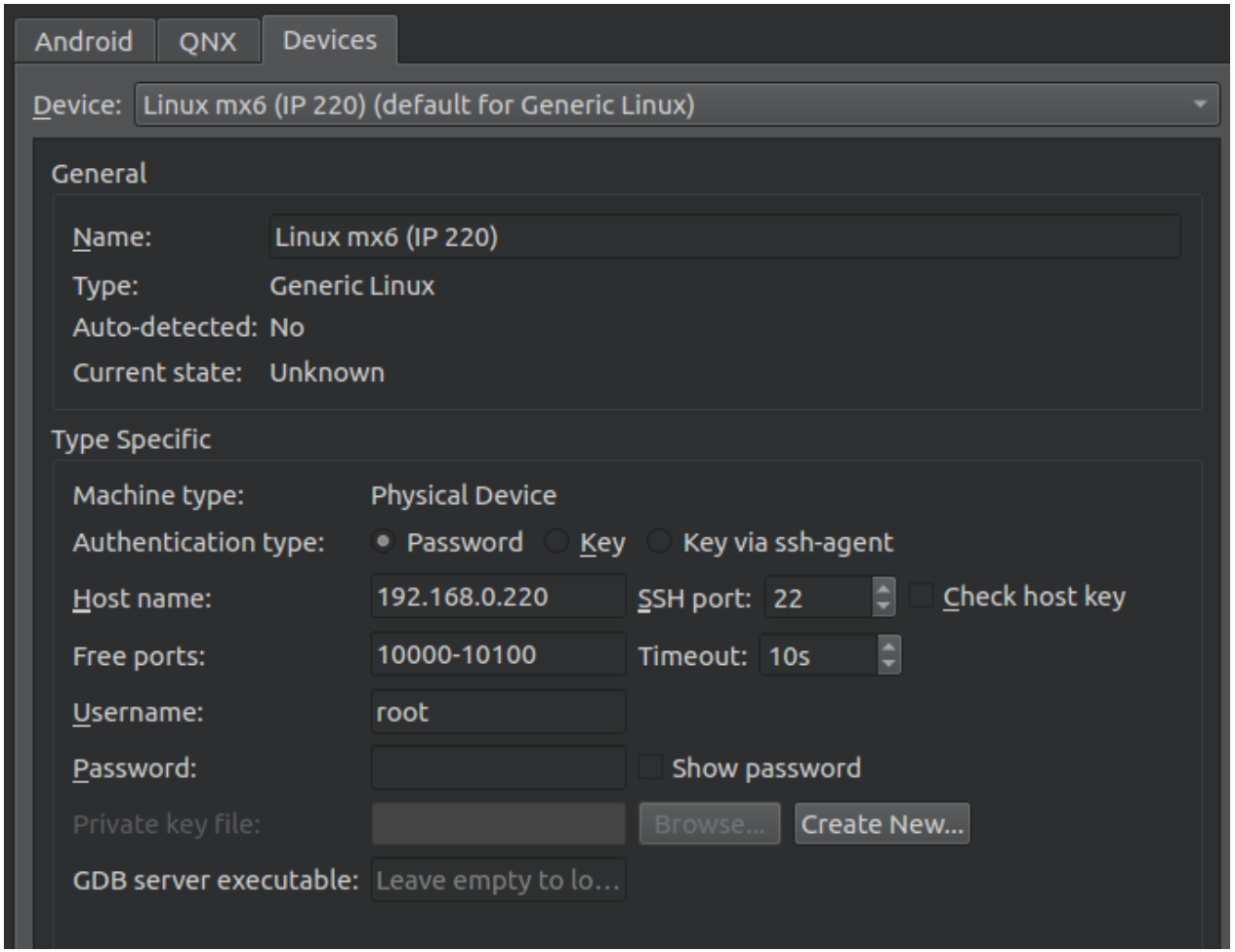
QtCreator for embedded

- Setup a KIT for your embedded board
- Kits | Qt Version | Compilers | Qbs | CMake



Application deployment (1/2)

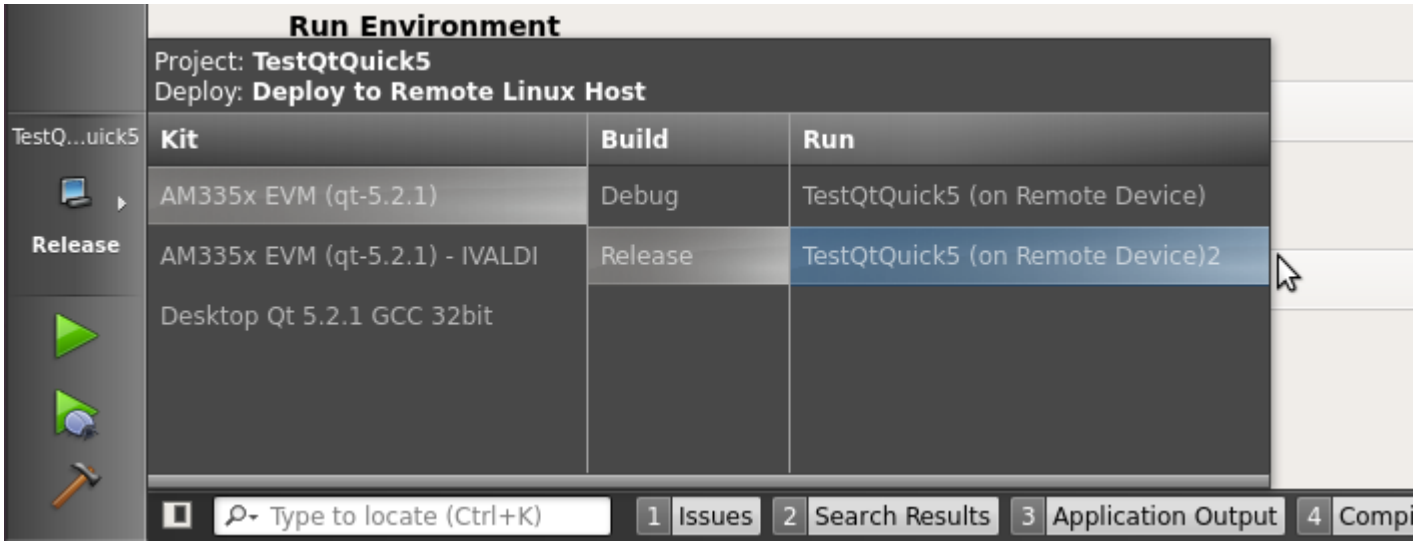
- Setup the Generic Linux device in QtCreator



Application deployment (2/2)

- QtCreator can deploy only if there is **openssh** on the target. *(**DO NOT** use dropbear !)*

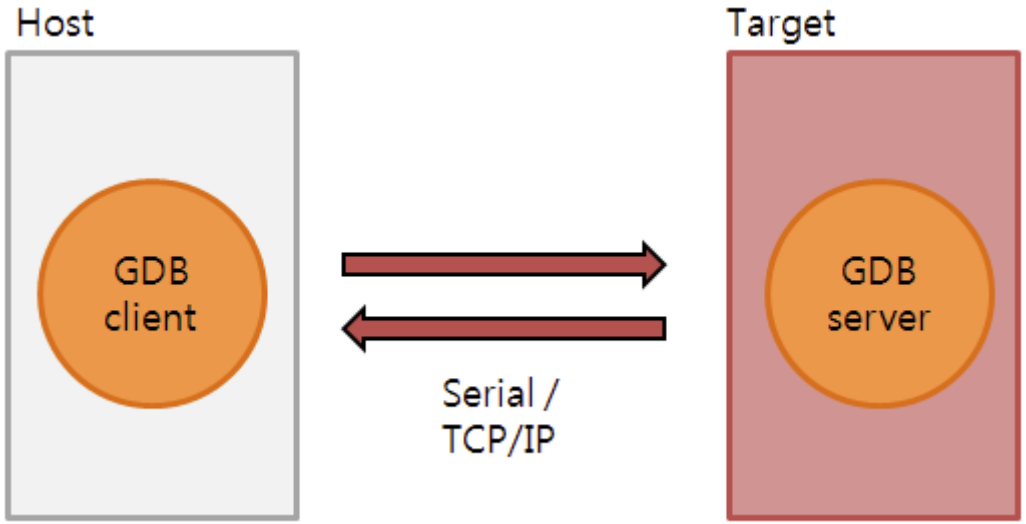
```
EXTRA_IMAGE_FEATURES_append = " ssh-server-openssh"
```



Embedded debug

- Add **remote** debug support to the image

```
EXTRA_IMAGE_FEATURES_append = " debug-tweaks"
```



Demo on real hardware


59 fps
Cinematic Experience
Qt
Rendering
Particles
Sprites
SAMSUNG
NVIDIA
Host PC
USB
(Console)
Ethernet
USB-SS
USB
VCM HDMI
VCM
DP
MXM DP
#1
#2
Audio
Line Out
Line In
USB
USB


Angle: -3
Speed left: 0
Speed right: 0
Distance: 80
Voltage: 1191
Toradex
Embedded. Computing.
antmicro
Qt


Angle: -3
Speed left: 0
Speed right: 0
Distance: 88
Voltage: 1191
Toradex
Embedded. Computing.
antmicro
Qt

Thank you!

http://yoctoproject.org

Questions?