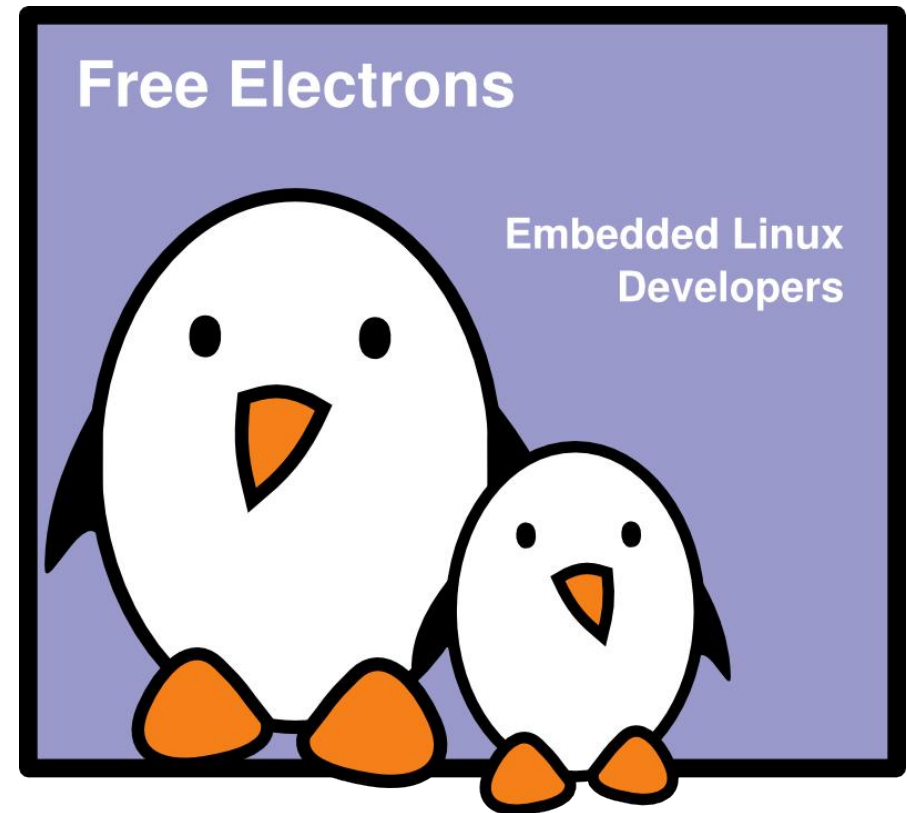




## Embedded Linux system development

Thomas Petazzoni  
Michael Opdenacker  
**Free Electrons**



© Copyright 2004-2009, Free Electrons.

Creative Commons BY-SA 3.0 license

Latest update: Feb 21, 2011,

Document sources, updates and translations:

<http://free-electrons.com/docs/sysdev>

Corrections, suggestions, contributions and translations are welcome!



# Contents

- ▶ Using open-source components
- ▶ Tools for the target device
  - ▶ Networking
  - ▶ System utilities
  - ▶ Language interpreters
  - ▶ Audio, video and multimedia
  - ▶ Graphical toolkits
  - ▶ Databases
  - ▶ Web browsers
- ▶ System building
- ▶ Emulators
- ▶ Commercial toolsets and distributions



## Leveraging open-source components in an Embedded Linux system



# Third party libraries and applications

- ▶ One of the advantages of embedded Linux is the wide range of third-party libraries and applications that one can leverage in its product
  - ▶ They are freely available, freely distributable, and thanks to their open-source nature, they can be analyzed and modified according to the needs of the project
- ▶ However, efficiently re-using these components is not always easy. One must:
  - ▶ Find these components
  - ▶ Choose the most appropriate ones
  - ▶ Cross-compile them
  - ▶ Integrate them in the embedded system and with the other applications



# Find existing components

- ▶ Freshmeat, a website referencing most open-source projects  
<http://www.freshmeat.net>
- ▶ Free Software Directory  
<http://directory.fsf.org>
- ▶ Look at other embedded Linux products, and see what their components are
- ▶ Look at the list of software packaged by embedded Linux build systems
  - ▶ These are typically chosen for their suitability to embedded systems
- ▶ Ask the community or Google
- ▶ This presentation will also feature a list of components for common needs



# Choosing components

- ▶ Not all free software components are necessarily good to re-use. One must pay attention to:
  - ▶ **Vitality** of the developer and user communities. This vitality ensures long-term maintenance of the component, and relatively good support. It can be measured by looking at the mailing-list traffic and the version control system activity.
  - ▶ **Quality** of the component. Typically, if a component is already available through embedded build systems, and has a dynamic user community, it probably means that the quality is relatively good.
  - ▶ **License**. The license of the component must match your licensing constraints. For example, GPL libraries cannot be used in proprietary applications.
  - ▶ **Technical requirements**. Of course, the component must match your technical requirements. But don't forget that you can improve the existing components if a feature is missing !



# Licenses (1)

- ▶ All software that are under a free software license give four freedoms to all users
  - ▶ Freedom to use
  - ▶ Freedom to study
  - ▶ Freedom to copy
  - ▶ Freedom to modify and distribute modified copies
- ▶ See <http://www.gnu.org/philosophy/free-sw.html> for a definition of Free Software
- ▶ Open Source software, as per the definition of the Open Source Initiative, are technically similar to Free Software in terms of freedoms
- ▶ See <http://www.opensource.org/docs/osd> for the definition of Open Source Software



# Licenses (2)

- ▶ Free Software licenses fall in two main categories
  - ▶ The copyleft licenses
  - ▶ The non-copyleft licenses
- ▶ The concept of « copyleft » is to ask for reciprocity in the freedoms given to an user.
- ▶ The result is that when you receive a software under a copyleft free software license and distribute modified versions of it, you must do so under the same license
  - ▶ Same freedoms to the new users
  - ▶ It's an incentive to contribute back your changes instead of keeping them secret
- ▶ Non-copyleft licenses have no such requirements, and modified versions can be kept proprietary, but they still require attribution



- ▶ GNU General Public License
- ▶ Covers ~55% of the free software projects
  - ▶ Including the Linux kernel, Busybox and many applications
- ▶ Is a copyleft license
  - ▶ Requires derivative works to be released under the same license
  - ▶ Programs linked with a library released under the GPL must also be released under the GPL
- ▶ Some programs covered by version 2 (Linux kernel, Busybox and others)
- ▶ More and more programs covered by version 3, released in 2007
  - ▶ Major change for the embedded market: the requirement that the user must be able to **run** the modified versions on the device, if the device is a « consumer » device



# GPL: redistribution

- ▶ No obligation when the software is not distributed
  - ▶ You can keep your modifications secret until the product delivery
- ▶ It is then authorized to distribute binary versions, if one of the following conditions is met:
  - ▶ Convey the binary with a copy of the source on a physical medium
  - ▶ Convey the binary with a written offer valid for 3 years that indicates how to fetch the source code
  - ▶ Convey the binary with the network address of a location where the source code can be found
  - ▶ See section 6. of the GPL license
- ▶ In all cases, the attribution and the license must be preserved
  - ▶ See section 4. and 5.



# LGPL

- ▶ GNU Lesser General Public License
- ▶ Covers ~10% of the free software projects
- ▶ A copyleft license
  - ▶ Modified versions must be released under the same license
  - ▶ But, programs linked against a library under the LGPL do not need to be released under the LGPL and can be kept proprietary
  - ▶ However, the user must keep the ability to update the library independently from the program, so dynamic linking must be used
- ▶ Used instead of the GPL for most of the libraries, including the C libraries
  - ▶ Some exceptions: MySQL, or Qt <= 4.4
- ▶ Also available in two versions, v2 and v3



# Licensing: examples

- ▶ You make modifications to the Linux kernel (to add drivers or adapt to your board), to Busybox, U-Boot or other GPL software
  - ▶ You must release the modified versions under the same license, and be ready to distribute the source code to your customers
- ▶ You make modifications to the C library or any other LGPL library
  - ▶ You must release the modified versions under the same license
- ▶ You create an application that relies on LGPL libraries
  - ▶ You can keep your application proprietary, but you must link dynamically with the LGPL libraries
- ▶ You make modifications to a non-copyleft licensed software
  - ▶ You can keep your modifications proprietary, but you must still credit the authors



# Non-copyleft licenses

- ▶ A large family of non-copyleft licenses that are relatively similar in their requirements
- ▶ A few examples
  - ▶ Apache license (~ 4%)
  - ▶ BSD license (~ 6%)
  - ▶ MIT license (~ 4%)
  - ▶ X11 license
  - ▶ Artistic license (~9 %)



# BSD license

```
Copyright (c) <year>, <copyright holder>  
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or  
without modification, are permitted provided that the following  
conditions are met:
```

```
* Redistributions of source code must retain the above  
copyright notice, this list of conditions and the following  
disclaimer.
```

```
* Redistributions in binary form must reproduce the above  
copyright notice, this list of conditions and the following  
disclaimer in the documentation and/or other materials provided  
with the distribution.
```

```
* Neither the name of the <organization> nor the names of its  
contributors may be used to endorse or promote products derived  
from this software without specific prior written permission.
```

```
[...]
```



# Is this free software?

- ▶ Most of the free software projects are covered by ~10 well-known licenses, so it is fairly easy for the majority of project to get a good understanding of the license
- ▶ Otherwise, read the license text
- ▶ Check Free Software Foundation's opinion  
<http://www.fsf.org/licensing/licenses/>
- ▶ Check Open Source Initiative's opinion  
<http://www.opensource.org/licenses>



# Respect free software licenses

- ▶ Free Software is not public domain software, the distributors have obligations due to the licenses
  - ▶ **Before** using a free software component, make sure the license matches your project constraints
  - ▶ Make sure to keep a complete list of the free software packages you use, the original version you used and to keep your modifications and adaptations well-separated from the original version
  - ▶ Conform to the license requirements **before** shipping the product to the customers
- ▶ Free Software licenses have been enforced successfully in courts
  - ▶ GPL-violations.org, <http://www.gpl-violations.org>
  - ▶ Software Freedom Law Center, <http://www.softwarefreedom.org/>
- ▶ Ask your legal department !



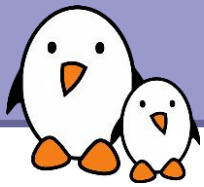
# Keeping changes separate (1)

- ▶ When integrating existing open-source components in your project, it is sometimes needed to make modifications to them
  - ▶ Better integration, reduced footprint, bug fixes, new features, etc.
- ▶ Instead of mixing these changes, it is much better to keep them separate from the original component version
  - ▶ If the component needs to be upgraded, easier to know what modifications were made to the component
  - ▶ If support from the community is requested, important to know how different the component we're using is from the upstream version
  - ▶ Makes contributing the changes back to the community possible
- ▶ It is even better to keep the various changes made on a given component separate
  - ▶ Easier to review and to update to newer versions



# Keeping changes separate (2)

- ▶ The simplest solution is to use Quilt
  - ▶ Quilt is a tool that allows to maintain a stack of patches over source code
  - ▶ Makes it easy to add, remove modifications from a patch, to add and remove patches from stack and to update them
  - ▶ The stack of patches can be integrated into your version control system
  - ▶ <https://savannah.nongnu.org/projects/quilt/>
- ▶ Another solution is to use a version control system
  - ▶ Import the original component version into your version control system
  - ▶ Maintain your changes in a separate branch



## Tools for the target device Networking



# ssh server and client: dropbear

<http://matt.ucc.asn.au/dropbear/dropbear.html>

- ▶ Very small memory footprint ssh server for embedded systems
- ▶ Satisfies most needs. Both client and server!
- ▶ Size: 110 KB, statically compiled with `uClibc` on `i386`.  
(`OpenSSH` client and server: approx 1200 KB,  
dynamically compiled with `glibc` on `i386`)
- ▶ Useful to:
  - ▶ Get a remote console on the target device
  - ▶ Copy files to and from the target device (`scp` or `rsync -e ssh`).



# Benefits of a web server interface

- Many network enabled devices can just have a network interface
- ▶ Examples: modems / routers, IP cameras, printers...
  - ▶ No need to develop drivers and applications for computers connected to the device. No need to support multiple operating systems!
  - ▶ Just need to develop static or dynamic HTML pages (possibly with powerful client-side JavaScript).  
Easy way of providing access to device information and parameters.
  - ▶ Reduced hardware costs (no LCD, very little storage space needed)



# thttpd

Tiny/Turbo/Throttling **HTTP**  
server

<http://acme.com/software/thttpd/>

- ▶ Simple  
Implements the **HTTP/1.1**  
minimum (or just a little more)  
Simple to configure and run.
- ▶ Small  
Executable size: 88K (version  
**2.25b**), **Apache 2.0.52**: 264K  
Very low memory consumption:  
does not fork and very careful  
about memory consumption.

- ▶ Portable  
Compiles cleanly on most  
Unix-like operating systems
- ▶ Fast  
About as fast as full-featured  
servers. Much faster on very  
high loads (because reduces  
the server load for the same  
amount of work)
- ▶ Secure  
Designed to protect the  
webserver machine from  
attacks.



# Other web servers

- ▶ BusyBox http server: <http://busybox.net>

Tiny: only adds 9 K to BusyBox 1.5 (dynamically linked with [glibc](#) on [i386](#), with all features enabled.)! Sufficient features for many devices with a web interface, including CGI, http authentication and script support (like PHP).

License: GPL



- ▶ Boa: <http://www.boa.org/>

Designed to be simple, fast and secure.

Unlike [thttpd](#), no particular care for memory or disk footprint though.

Embedded systems: pretty popular, though not targeted by developers.



- ▶ [lighttpd](#): <http://lighttpd.net>

Low footprint server good at managing high loads.

May be useful in embedded systems too.





# Network utilities (1)

- ▶ **avahi** is an implementation of Multicast DNS Service Discovery, that allows programs to publish and discover services on a local network
- ▶ **bind**, a DNS server
- ▶ **iptables**, the userspace tools associated to the Linux firewall, Netfilter
- ▶ **iw and wireless tools**, the userspace tools associated to Wireless devices
- ▶ **net-snmp**, implementation of the SNMP protocol
- ▶ **openntpd**, implementation of the Network Time Protocol, for clock synchronization
- ▶ **openssl**, a toolkit for SSL and TLS connections



# Network utilities (2)

- ▶ **pppd**, implementation of the Point to Point Protocol, used for dialup connections
- ▶ **samba**, implements the SMB and CIFS protocols, used by Windows to share files and printers
- ▶ **coherence**, a UPnP/DLNA implementation
- ▶ **vsftpd**, **proftpd**, FTP servers



## Tools for the target device System utilities



# System utilities

- ▶ **dbus**, an inter-application object-oriented communication bus
- ▶ **gpsd**, a daemon to interpret and share GPS data
- ▶ **hal**, the Hardware Abstraction Layer suite. A daemon that receives hardware notifications, maintains a database of available hardware devices and offers a D-Bus interface
- ▶ **libraw1394**, raw access to Firewire devices
- ▶ **libusb**, a userspace library for accessing USB devices without writing an in-kernel driver
- ▶ Utilities for kernel subsystems: **i2c-tools** for I2C, **input-tools** for input, **mtd-utils** for MTD devices, **usbutils** for USB devices



Tools for the target device  
Language interpreters



# Language interpreters

- ▶ Interpreters for the most common scripting languages are available. Useful for
  - ▶ Application development
  - ▶ Web services development
  - ▶ Scripting
- ▶ Languages supported
  - ▶ Lua
  - ▶ Python
  - ▶ Perl
  - ▶ Ruby
  - ▶ TCL
  - ▶ PHP



Tools for the target device  
Audio, video and multimedia



# Audio, video and multimedia

- ▶ **gstreamer**, a multimedia framework
  - ▶ Allows to decode/encode a wide variety of formats
  - ▶ Supports hardware encoders and decoders through plugins
- ▶ **alsa-lib**, the userspace tools associated to the ALSA sound kernel subsystem
- ▶ Encoding and decoding libraries such as flac, libogg, libtheora, libvorbis, libmad, libsndfile, speex, etc.



## Tools for the target device Graphical toolkits



# Graphical toolkits

## « Low-level » solutions and layers



# DirectFB (1)



- ▶ Low-level graphical library
  - ▶ Lines, rectangles, triangles drawing and filling
  - ▶ Blitting, flipping
  - ▶ Text drawing
  - ▶ Windows and transparency
  - ▶ Image loading and video display
- ▶ But also handles input event handling: mouse, keyboard, joystick, touchscreen, etc.
- ▶ Provides accelerated graphic operations on various hardware, more can be added in an easy way
- ▶ Integrated windowing infrastructure

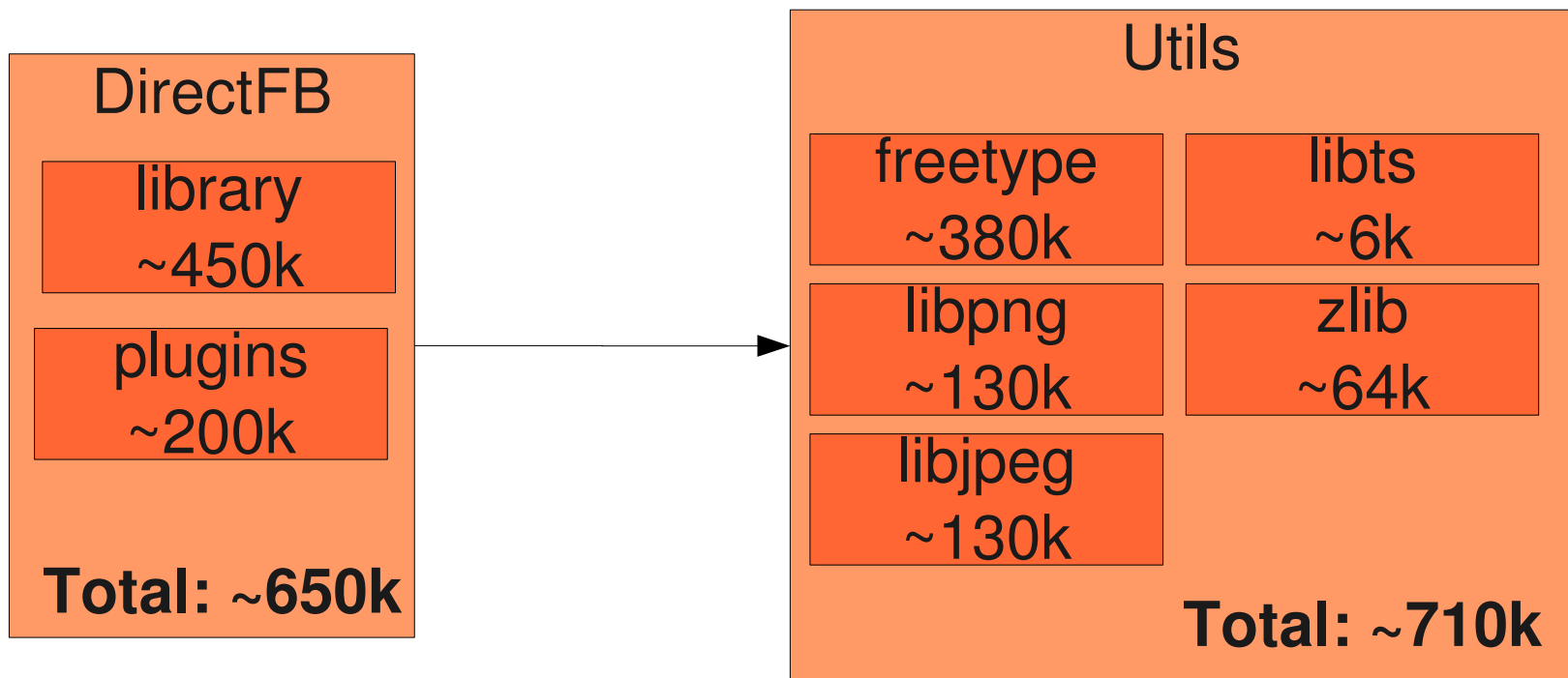


# DirectFB (2)

- ▶ Single-application by default, but multiple applications can share the framebuffer thanks to « fusion »
- ▶ Development and community: very active
- ▶ License: LGPL 2.1
- ▶ <http://www.directfb.org>



# DirectFB: size and dependencies

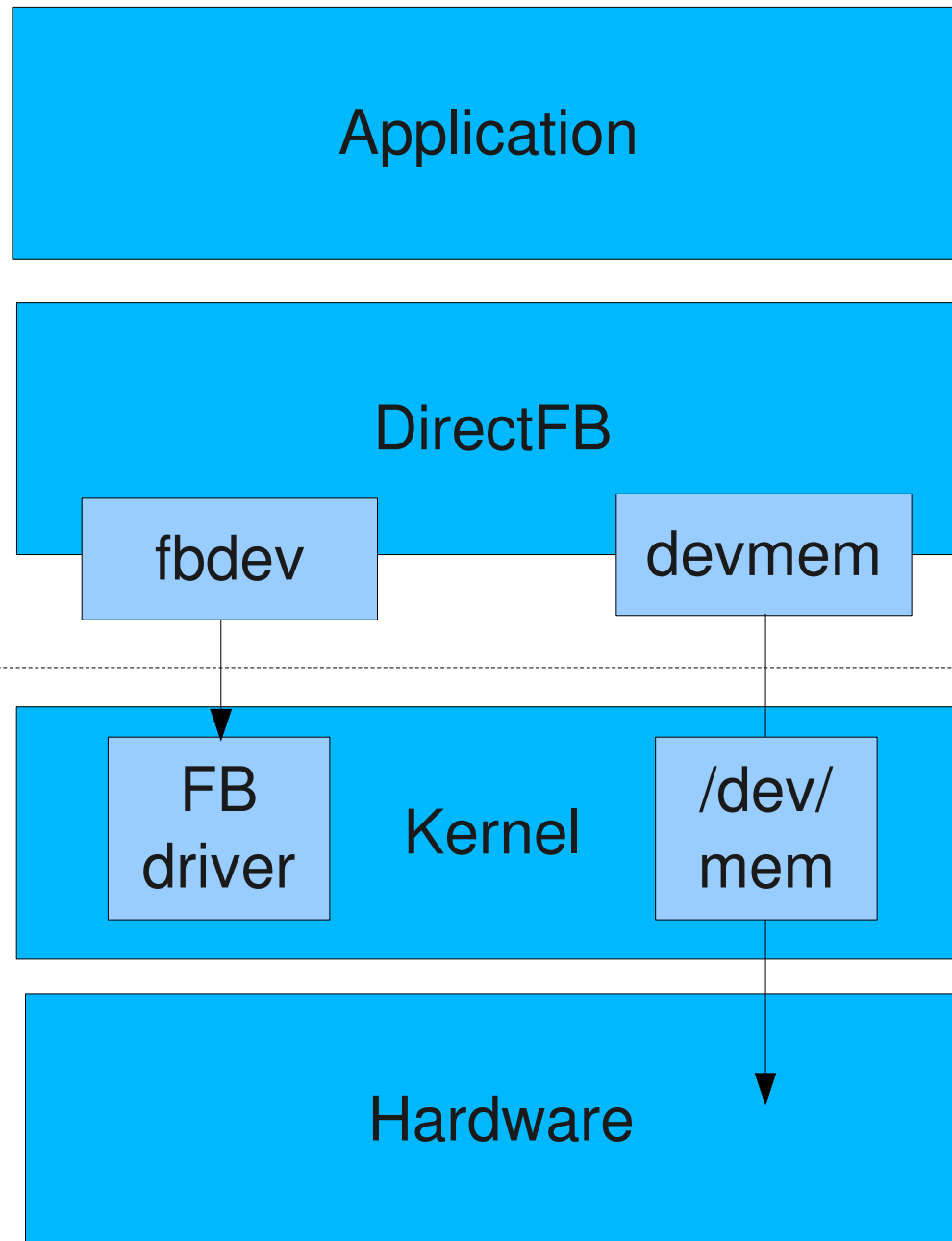


**Total: ~1.4m**

Some of these dependencies are optional. This is a typical setup.



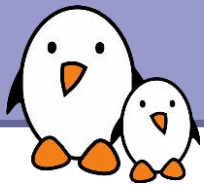
# DirectFB: architecture





# DirectFB: usage (1)

- ▶ Multimedia applications
  - ▶ For example the Disko framework, for set-top box related applications
- ▶ « Simple » graphical applications
  - ▶ Industrial control
  - ▶ Device control with limited number of widgets
- ▶ Visualization applications
- ▶ As a lower layer for higher-level graphical libraries



# DirectFB: usage (2)





# SDL

- ▶ A library originally designed for game development
- ▶ In addition to graphic display, also provides input event management, sound, CD-ROM audio, threads, timers, etc.
- ▶ Can work on top of X11, the framebuffer or DirectFB
  - ▶ And DirectFB can work on top of SDL as well :-)
- ▶ The API is roughly the same level as the one of DirectFB
- ▶ Developed in C, C API, many bindings available
- ▶ Actively maintained
- ▶ DirectFB is probably more common in embedded systems, while SDL is more common for small desktop games
- ▶ License: LGPL
- ▶ <http://www.libsdl.org/>





# SDL screenshots

```
QEMU
Partition check:
hda:
ne.c:v1.10 9/23/94 Donald Becker (becker@scyld.com)
Last modified Nov 1, 2000 by Paul Gortmaker
NE*000 ethercard probe at 0x300: 52 54 00 12 34 56
eth0: NE2000 found at 0x300, using IRQ 9.
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 2048 bind 2048)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 72k freed
EXT2-fs warning: mounting unchecked fs, running e2fsck is recommended

Linux version 2.4.20 (bellard@voyager) (gcc version 2.95.2 20000220 (Debian GNU/
Linux)) #45 Sat Aug 9 02:26:16 CEST 2003

This is the Red Hat rescue disk. Most of the basic system commands are
in /bin.

Type exit to halt the system.
#
```

QEMU: CPU and system emulator



Pig: a demo arcade game. 7000 lines.



# X.org - KDrive

- ▶ Stand-alone simplified version of the X server, for embedded systems
  - ▶ Formerly know as Tiny-X
  - ▶ Kdrive is integrated in the official X.org server
- ▶ Works on top of the Linux frame buffer, thanks to the Xfbdev variant of the server
- ▶ Real X server
  - ▶ Fully supports the X11 protocol: drawing, input event handling, etc.
  - ▶ Allows to use any existing X11 application or library
- ▶ Actively developed and maintained
- ▶ X11 license
- ▶ <http://www.x.org>





# Kdrive: size and dependencies

## X server

Xfbdev  
~1.2m

## Fonts

from a few kb  
to several mb

## X libraries

libxcb  
~300k

libXfont  
~380k

liblbxutil  
~156k

Misc libs  
~770k

libX11  
~920k

**Total: 2.5m**

## X toolkit (optional)

libXaw6,7,8  
~900k

libXt  
~330k

## Utils

dbus  
lib: ~200k  
bin: ~350k

libsfsfs  
~27k

libpng  
~130k

expat  
~120k

zlib  
~64k

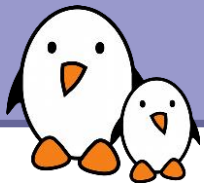
freetype  
~380k

pixman  
~130k

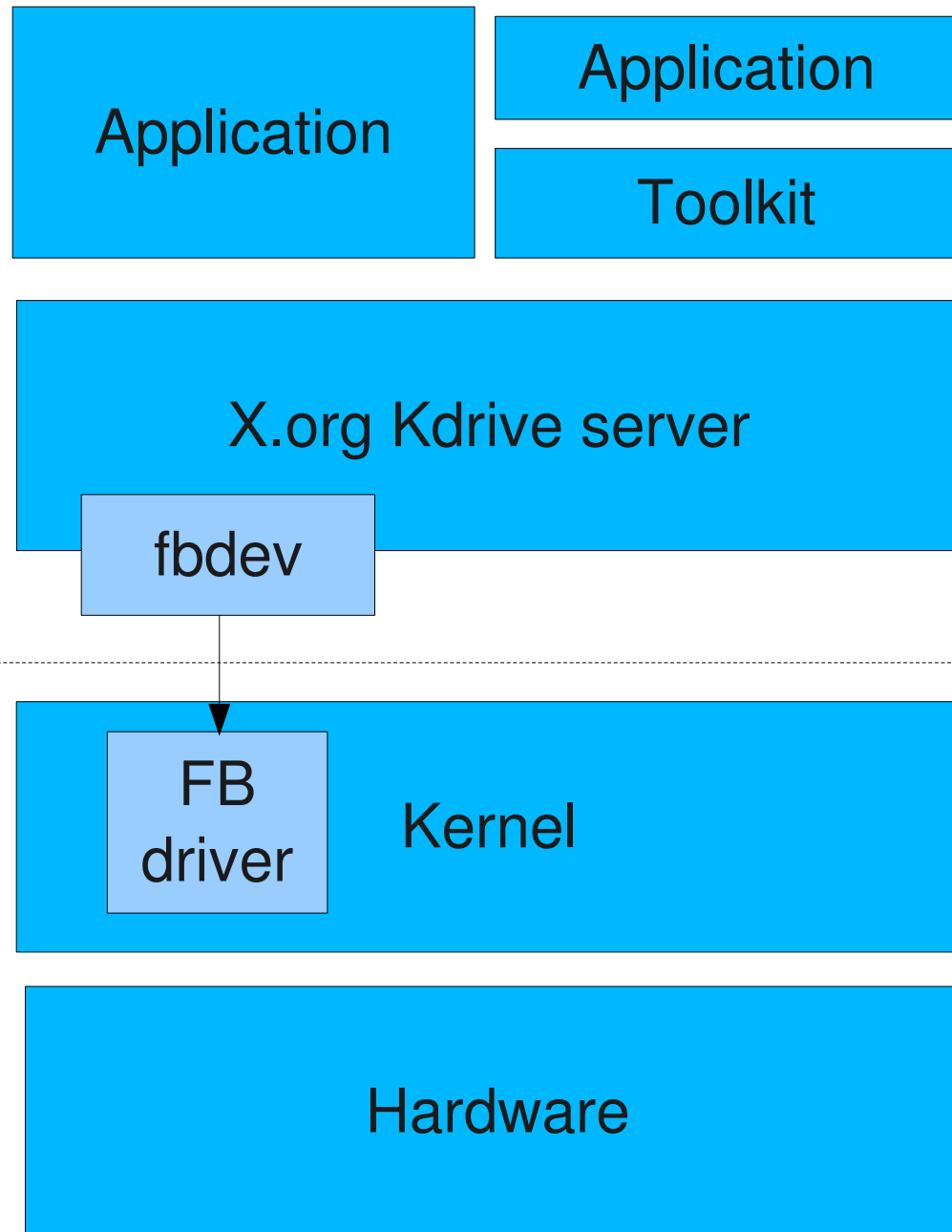
fontconfig  
~165k

**Total: 1.5m**

**Total, without X  
toolkit: 5.4m**



# Kdrive: architecture





# Kdrive: usage

- ▶ Can be directly programmed using Xlib / XCB
  - ▶ Low-level graphic library
  - ▶ Probably doesn't make sense since DirectFB is a more lightweight solution for an API of roughly the same level (no widgets)
- ▶ Or, usually used with a toolkit on top of it
  - ▶ Gtk
  - ▶ Qt
  - ▶ Fltk
  - ▶ WxEmbedded



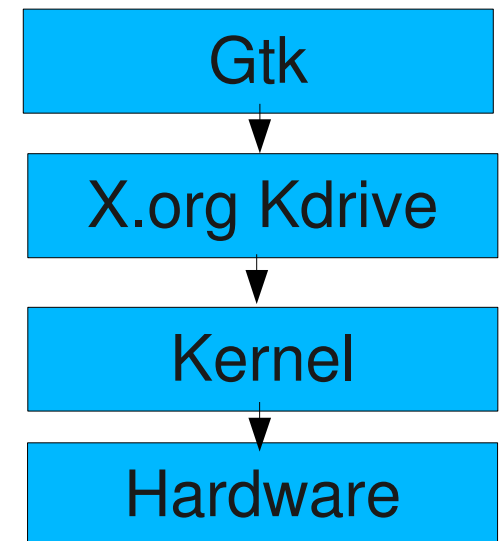
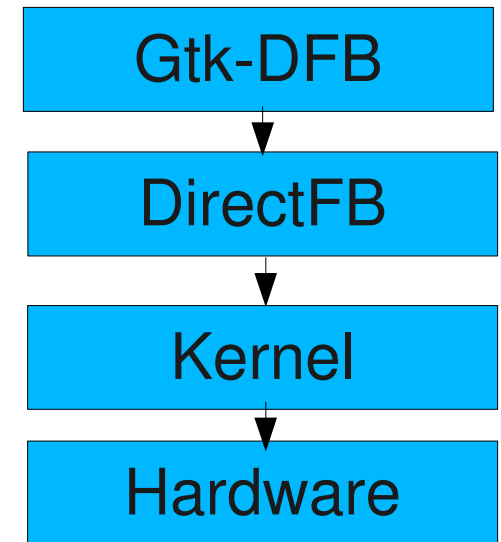
# Graphical toolkits

## « High-level » solutions



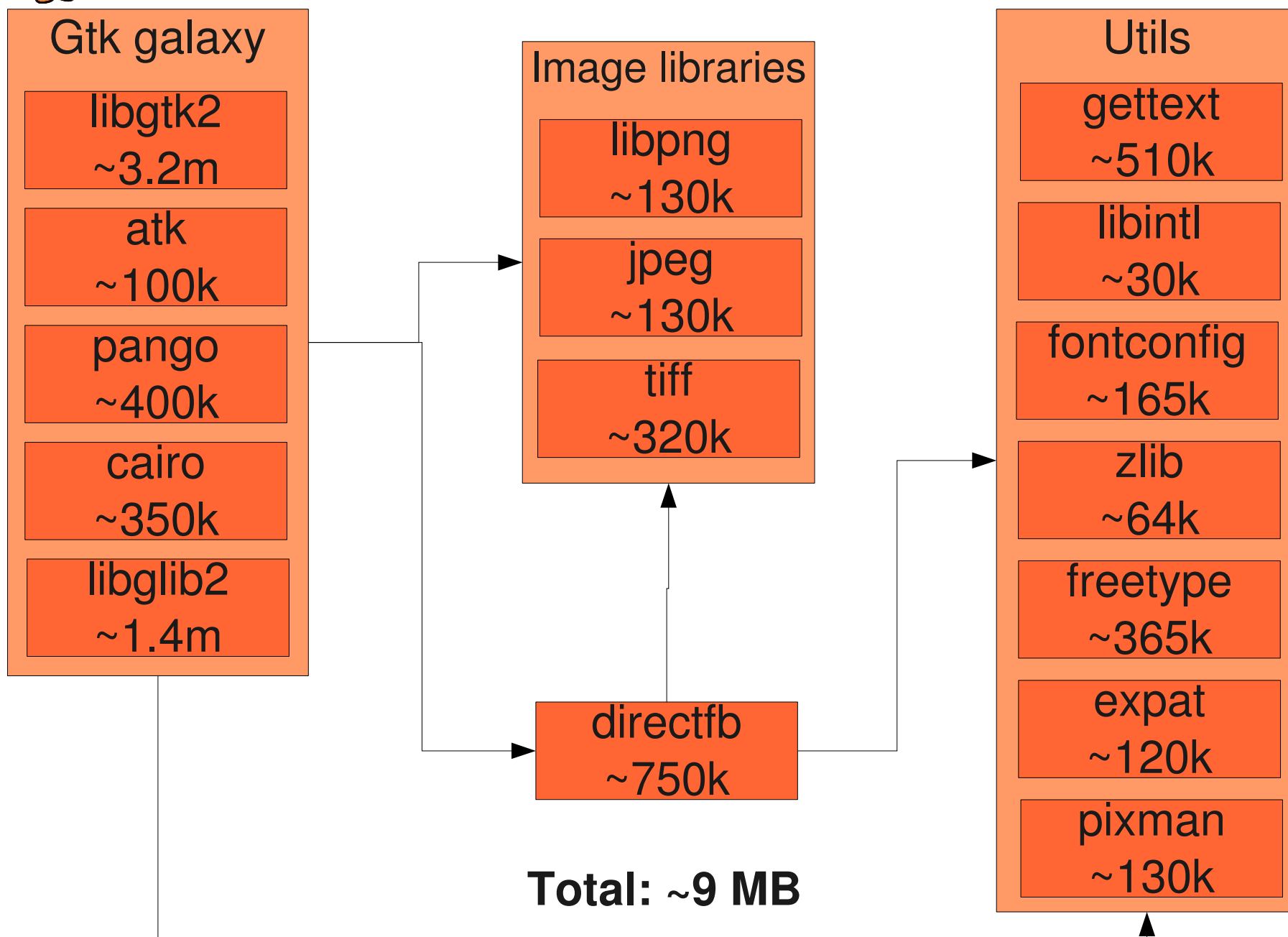
# Gtk

- ▶ The famous toolkit, providing widget-based high-level APIs to develop graphical applications
- ▶ Standard API in C, but bindings exist for various languages: C++, Python, etc.
- ▶ Two GDK back-ends
  - ▶ The classical Xorg back-end
  - ▶ The DirectFB back-end, which removes the need for an Xorg server
- ▶ No windowing system, a lightweight window manager needed to run several applications. Possible solution: Matchbox.
- ▶ License: LGPL
- ▶ <http://www.gtk.org>





# Gtk-DFB: dependencies and size





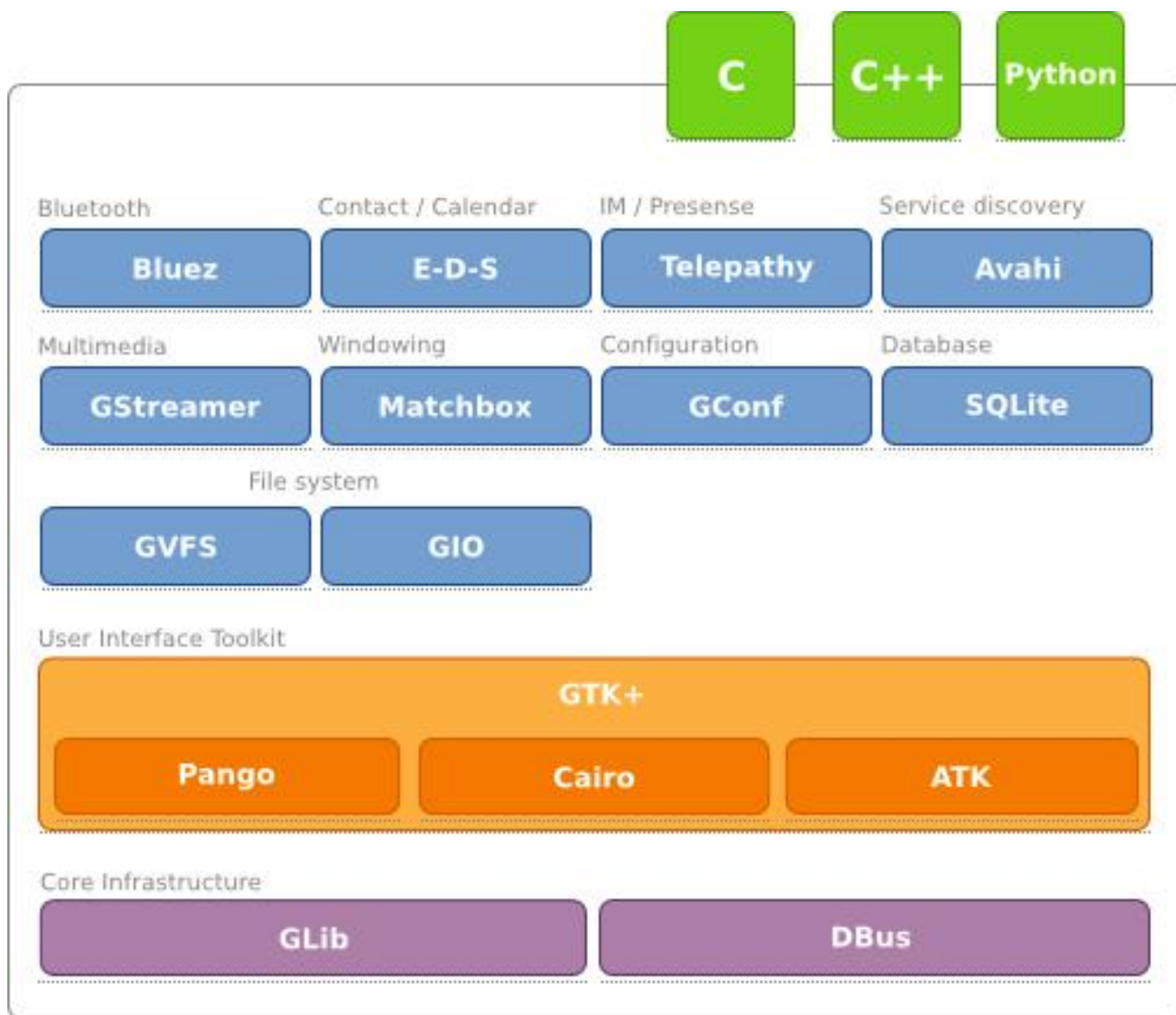
# Gtk stack components

- ▶ **Glib**, core infrastructure
  - ▶ Object-oriented infrastructure GObject
  - ▶ Event loop, threads, asynchronous queues, plug-ins, memory allocation, I/O channels, string utilities, timers, date and time, internationalization, simple XML parser, regular expressions
  - ▶ Data types: memory slices and chunks, linked lists, arrays, trees, hash tables, etc.
- ▶ **Pango**, internationalization of text handling
- ▶ **ATK**, accessibility toolkit
- ▶ **Cairo**, vector graphics library
- ▶ **Gtk+**, the widget library itself
- ▶ *The Gtk stack is a complete framework to develop applications*



# GNOME Mobile

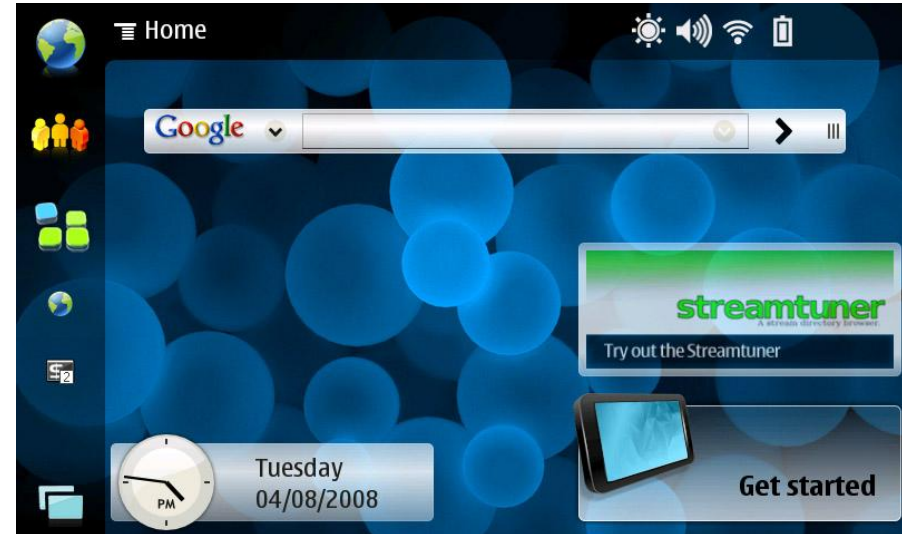
- ▶ The Gtk stack is part of the larger GNOME Mobile platform





# Gtk examples

OM 2007.2 platform on OpenMoko phone



Maemo platform on Nokia Internet tablets



Interface of Vernier data acquisition and visualization systems



# Qt (1)

- ▶ The other famous toolkit, providing widget-based high-level APIs to develop graphical applications
  - ▶ « Qt for Embedded Linux », formerly known as Qtopia Core, is the version of Qt that runs on top of a frame buffer, on embedded devices. It includes a windowing system
  - ▶ « Qt Extended », formerly known as Qtopia, extends « Qt for Embedded Linux » with useful components on embedded devices: communication, contents, application-specific and user experience components.
- ▶ Implemented in C++
  - ▶ the C++ library is required on the target system
  - ▶ standard API in C++, but bindings are also available for other languages

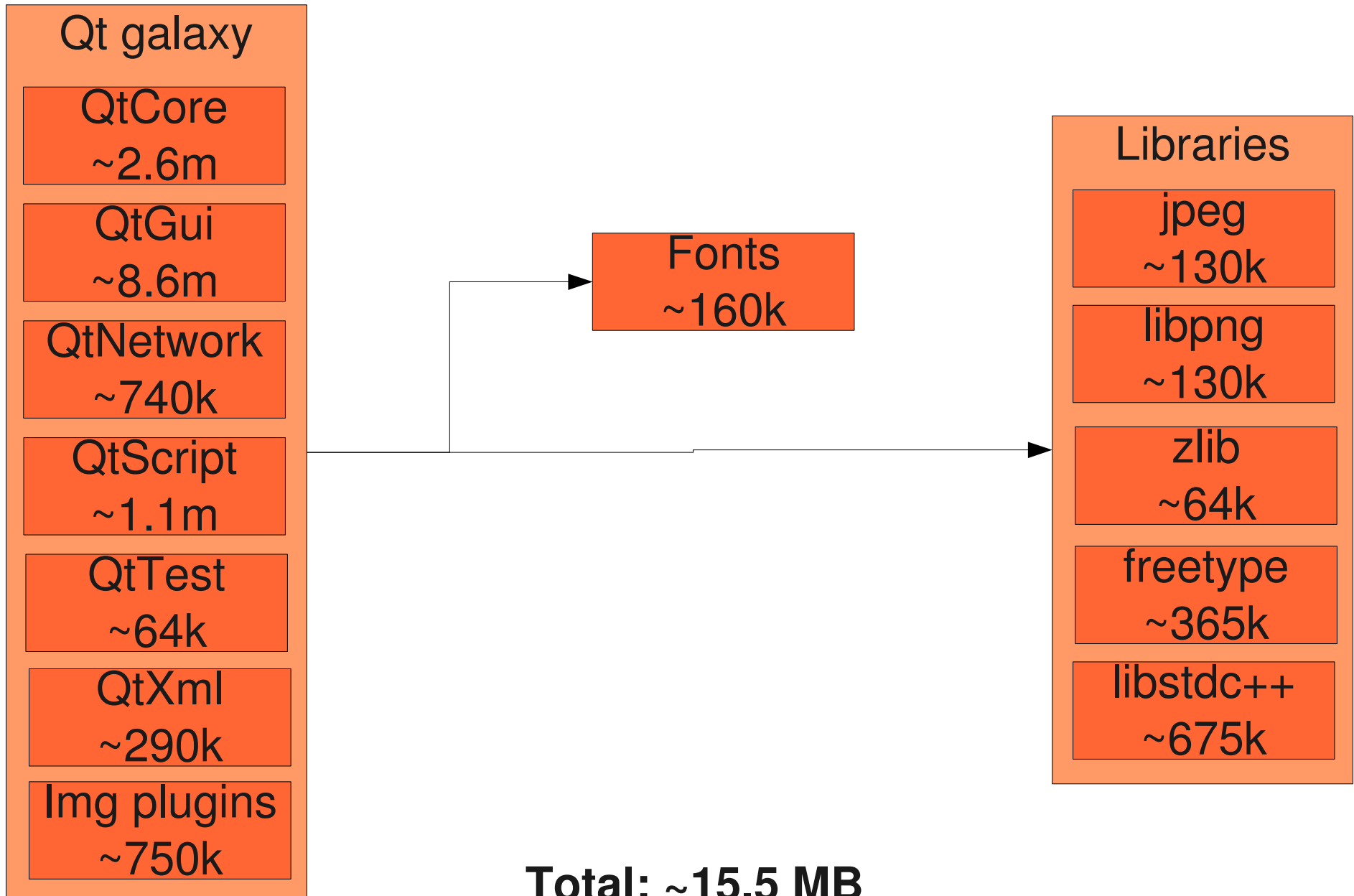
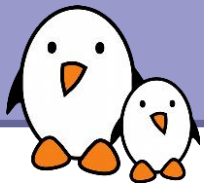
<http://www.qtsoftware.com/products/platform/qt-for-embedded-linux>

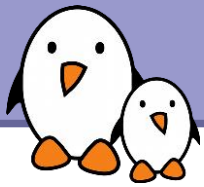


## Qt (2)

- ▶ Works either on top of
  - ▶ Framebuffer
  - ▶ X11
  - ▶ DirectFB backend integrated in version 4.4, which allows to take advantage of the acceleration provided by DirectFB drivers
- ▶ Qt is more than just a graphical toolkit, it also offers a complete development framework: data structures, threads, network, databases, XML, etc.
- ▶ Qt Embedded has an integrated windowing system, allowing several applications to share the same screen
- ▶ Very well documented
- ▶ Since version 4.5, available under the LGPL, allowing proprietary applications

# Qt: size and dependencies





# Qt's usage



Qt on the OpenMoko phone



Qt on the Dash Express navigation system



Qt on the Netflix Player by Roku



# Other less used solutions

- ▶ Enlightenment foundation libraries

- ▶ Very powerful, but complicated to use due to the lack of documentation

- ▶ <http://www.enlightenment.org/p.php?p=about/efl>

- ▶ FLTK

- ▶ Lightweight C++ toolkit. Version 2.x works only on top of X.org. Used by relatively few applications.

- ▶ <http://www.fltk.org>

- ▶ WxEmbedded

- ▶ The embedded version of WxWindows. Works on top of X.org and DirectFB

- ▶ <http://www.wxwidgets.org/docs/embedded.htm>



## Tools for the target device Databases



# Lightweight database - SQLite

<http://www.sqlite.org>

- ▶ SQLite is a small C library that implements a self-contained, embeddable, lightweight, zero-configuration SQL database engine
- ▶ The database engine of choice for embedded Linux systems
  - ▶ Can be used as a normal library
  - ▶ Can be directly embedded into a application, even a proprietary one since SQLite is released in the public domain



## Tools for the target device Web browsers



# Fast and tiny web browser: Dillo

<http://www.dillo.org/>

- ▶ Very fast, lightweight web browser written in C/C++, with a FLTK2 interface  
The Dillo binary fits in 940 KB on x86 (doesn't include the FLTK2 shared libraries)
- ▶ License: GPL
- ▶ Supports many standard features: cookies, images, tables, CSS...  
Extensible through plugins (e.g. ftp)
- ▶ Fits well on small screens
- ▶ Still missing: frames, javascript
- ▶ A good solution if your product just needs to display its own HTML pages. Not sufficient as a general purpose web browser.





# Links

<http://links.twibright.com/>

- ▶ Portable browser with many features: frames, tables, SSL, Javascript...
- ▶ Runs either in text mode or with several graphical back-ends: **X**, **DirectFB**, **SDL**, **SVGAlib**.
- ▶ Size: 4 MB (**i386**, **glibc**) + shared libraries (**libpng**, **libssl**...)





# Full featured browser: Mozilla Firefox

<http://www.mozilla.org/products/firefox/>

- Lightweight and fast browser based on **Mozilla**
- Full featured: **CSS**, **SSL**, **Javascript**, tabbed browsing, pop-up blocking..., but very easy to configure.
- Takes around 40 MB of RAM with 8 tabs open.  
Need 25 MB of storage space (**Sharp Zaurus**)
- Designed to be cross-platform. Already used in embedded systems with sufficient screen resolution (web pads, high-end PDAs)
- Great for consumers appliances. Looks familiar to consumers: the default theme recalls **IE**.



# WebKit

<http://webkit.org/>

- ▶ Web browser engine.  
Application framework that can be used to develop web browsers.
- ▶ License: portions in LGPL and others in BSD.  
Proprietary applications allowed.
- ▶ Used everywhere (MacOS X, iPhone, Google Android and Chrome...)  
Many applications (browsers, e-mail clients...) are already using WebKit:  
<http://trac.webkit.org/projects/webkit/wiki/Applications%20using%20WebKit>
- ▶ Multiple graphical back-ends: Qt4, GTK...
- ▶ Lightweight web-browsers : Midori (GTK), Arora (Qt)
- ▶ You could use it to create your custom browser.





# Embedded Linux system development

Example of components used in real devices



# Industrial applications

- ▶ In many industrial applications, the system is only responsible for monitoring and control a device
- ▶ Such a system is usually relatively simple in terms of components
  - ▶ Kernel
  - ▶ BusyBox
  - ▶ C library
  - ▶ Applications relying directly on the C library, sometimes using the real-time capabilities of the Linux kernel
  - ▶ Sometimes a Web server for remote control, or another server implementing a custom protocol



# Digital Photo Frame: requirements

- ▶ Example taken from a conference of Matt Porter, Embedded Alley at ELC 2008
- ▶ Hardware: ARM SoC with DSP, audio, 800x600 LCD, MMC/SD, NAND, buttons, speakers
- ▶ The photo frame must be able to
  - ▶ Display to the LCD
  - ▶ Detect SD card insertion, notify applications of the insertion so that applications can build a catalog of the pictures on the SD card
  - ▶ Modern 3D GUI with nice transitions
  - ▶ Navigation through buttons
  - ▶ Support audio playback (MP3, playlists, ID3 tag)
  - ▶ JPEG resize and rotation



# Digital Photo Frame: components (1)

- ▶ Base system
  - ▶ Components present in virtually all embedded Linux systems
  - ▶ The U-Boot bootloader
  - ▶ Linux Kernel
    - ▶ Drivers for SD/MMC, framebuffer, sound, input devices
  - ▶ Busybox
  - ▶ Build system, in this case was OpenEmbedded
  - ▶ Components: **u-boot, linux, busybox**



# Digital Photo Frame: components (2)

- ▶ Event handling to detect SD card insertion
  - ▶ udev, that receives events from the kernel, creates device nodes, and sends events to HAL
  - ▶ HAL, which maintains a database of available devices and provides a D-Bus API
  - ▶ D-Bus to connect HAL with the application. The application subscribes to HAL event through D-Bus and gets notified when they are triggered
  - ▶ Components: **udev, hal, dbus**



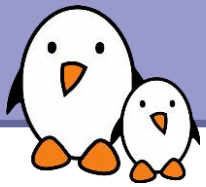
# Digital Photo Frame: components (3)

- ▶ JPEG display
  - ▶ **libjpeg** to decode the pictures
  - ▶ **jpegtran** to resize and rotate them
  - ▶ FIM (Fbi Improved) for dithering
- ▶ MP3 support
  - ▶ **libmad** for playing
  - ▶ **libid3** for ID3 tags reading
  - ▶ **libm3u** to support playlists
  - ▶ Used vendor-provided components to leverage the DSP to play MP3



# Digital Photo Frame: components (4)

- ▶ 3D interface
  - ▶ Vincent, an open-source implementation of OpenGL ES
  - ▶ Clutter, higher-level API to develop 3D applications
- ▶ Application itself
  - ▶ Manages media events
  - ▶ Uses the JPEG libraries to decode and render pictures
  - ▶ Receives Linux input events from buttons and draws OpenGL-based UI developed with Clutter
  - ▶ Manage a user-defined configuration
  - ▶ Play the music with the MP3-related libraries
  - ▶ Display photo slideshow



## System building



# System building: goal and solutions

- ▶ Goal
  - ▶ Integrate all the software components, both third-party and in-house, into a working root filesystem
  - ▶ It involves the download, extraction, configuration, compilation and installation of all components, and possibly fixing issues and adapting configuration files
- ▶ Several solutions
  - ▶ Manually
  - ▶ System building tools
  - ▶ Distributions or ready-made filesystems



# System building: manually

- ▶ Manually building a target system involves downloading, configuring, compiling and installing all the components of the system.
- ▶ All the libraries and dependencies must be configured, compiled and installed in the right order.
- ▶ Sometimes, the build system used by libraries or applications is not very cross-compile friendly, so some adaptations are necessary.
- ▶ There is no infrastructure to reproduce the build from scratch, which might cause problems if one component needs to be changed, if somebody else takes over the project, etc.



# System building: manually (2)

- ▶ Manual system building is not recommended for production projects
- ▶ However, using automated tools often requires the developer to dig into specific issues
- ▶ Having a basic understanding of how a system can be built manually is therefore very useful to fix issues encountered with automated tools
  - ▶ We will first study manual system building, and during a practical lab, create a system using this method
  - ▶ Then, we will study the automated tools available, and use one of them during a lab



# System foundations

- ▶ A basic root file system needs at least
  - ▶ A traditional directory hierarchy, with `/bin`, `/etc`, `/lib`, `/root`, `/usr/bin`, `/usr/lib`, `/usr/share`, `/usr/sbin`, `/var`, `/sbin`
  - ▶ A set of basic utilities, providing at least the init program, a shell and other traditional Unix command line tools. This is usually provided by Busybox
  - ▶ The C library and the related libraries (thread, math, etc.) installed in `/lib`
  - ▶ A few configuration files, such as `/etc/inittab`, and initialization scripts in `/etc/init.d`
- ▶ On top of this foundation common to most embedded Linux system, we can add third-party or in-house components



# Target and build spaces

- ▶ The system foundation, Busybox and C library, are the core of the target root filesystem
- ▶ However, when building other components, one must distinguish two directories
  - ▶ The « target » space, which contains the target root filesystem, everything that is needed for **execution** of the application
  - ▶ The « build » space, which will contain a lot more files than the «target» space, since it is used to keep everything needed to **compile** libraries and applications. So we must keep the headers, documentation, and other configuration files



# Build systems

- ▶ Each open-source component comes with a mechanism to configure, compile and install it
  - ▶ A basic simple Makefile
    - ▶ Need to read the Makefile to understand how it works and how to tweak it for cross-compilation
  - ▶ A build system based on the Autotools
    - ▶ As this is the most common build system, we will study it in details
  - ▶ CMake, <http://www.cmake.org/>
    - ▶ Newer and simpler than the autotools. Used by large projects such as KDE or Second Life
  - ▶ Scons, <http://www.scons.org/>
  - ▶ Waf, <http://code.google.com/p/waf/>
  - ▶ Other manual build systems

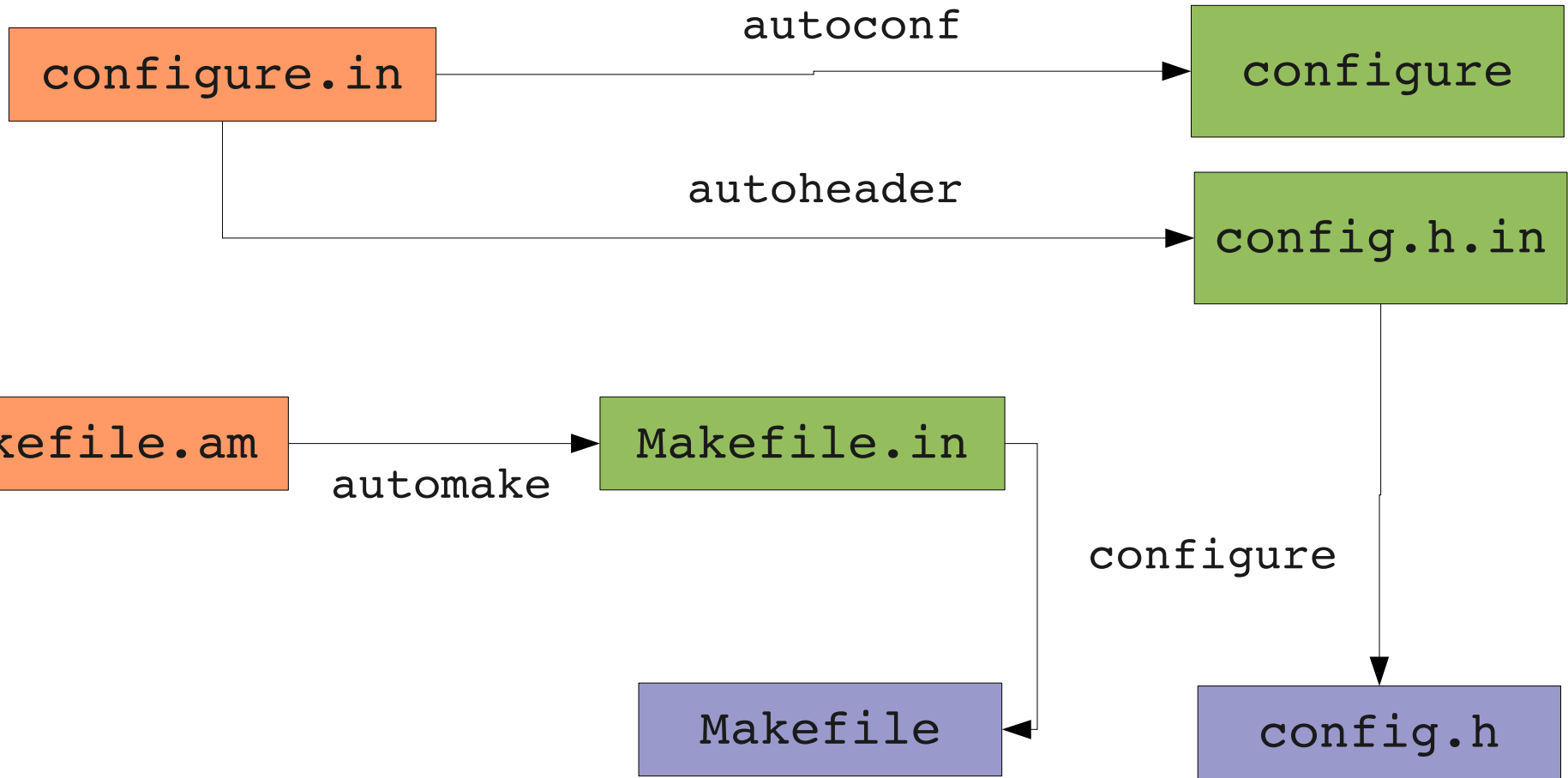


# Autotools and friends

- ▶ A family of tools, which associated together form a complete and extensible build system
  - ▶ **autoconf** is used to handle the configuration of the software package
  - ▶ **automake** is used to generate the Makefiles needed to build the software package
  - ▶ **pkgconfig** is used to ease compilation against already installed shared libraries
  - ▶ **libtool** is used to handle the generation of shared libraries in a system-independent way
- ▶ Most of these tools are old and relatively complicated to use, but they are used by a majority of free software packages today. One must have a basic understanding of what they do and how they work.



# automake / autoconf / autoheader



Written by the developer



Generated by the developer using the autotools



Generated by the user by running the configure script



# automake / autoconf

- ▶ Files written by the developer
  - ▶ `configure.in` describes the configuration options and the checks done at configure time
  - ▶ `Makefile.am` describes how the software should be built
- ▶ The `configure` script and the `Makefile.in` files are generated by `autoconf` and `automake` respectively.
  - ▶ They should never be modified directly
  - ▶ They are usually shipped pre-generated in the software package, because there are several versions of `autoconf` and `automake`, and they are not completely compatible
- ▶ The `Makefile` files are generated at configure time, before compiling
  - ▶ They are never shipped in the software package.



# Configuring and compiling (1)

- ▶ The traditional steps to configure and compile an autotools based package are :  
`./configure`  
`make`  
`make install`
- ▶ These steps work well for native compilation. For cross-compilation, things are a little bit more complicated.
  - ▶ At least some of the environment variables `AR`, `AS`, `LD`, `NM`, `CC`, `GCC`, `CPP`, `CXX`, `STRIP`, `OBJCOPY` must be defined to point to the proper cross-compilation tools. The host tuple is also by default used as prefix.
  - ▶ The `--host` argument must be passed to the `configure` script. The `--build` argument is automatically detected, and `--target` is only for tools generating code.
  - ▶ It is recommended to pass the `--prefix` argument. It defines from which location the software will run in the target environment. Usually, `/usr` is fine.



## Configuring and compiling (2)

- ▶ If one simply runs `make install`, the software will be installed in the directory passed as `--prefix`. For cross-compiling, one must pass the `DESTDIR` argument to specify where the software must be installed.
- ▶ Making the distinction between the prefix (as passed with `--prefix` at configure time) and the destination directory (as passed with `DESTDIR` at installation time) is very important.
- ▶ Example :

```
export PATH=/usr/local/arm-linux/bin:$PATH
export CC=arm-linux-gcc
export STRIP=arm-linux-strip
./configure --host=arm-linux
make
make DESTDIR=/home/<user>/work/rootfs install
```



# Installation

- ▶ The autotools based software packages provide both a `install` and `install-strip` make targets, used to install the software, either stripped or unstripped.
- ▶ For applications, the software is usually installed in `<prefix>/bin`, with configuration files in `<prefix>/etc` and data in `<prefix>/share/<application>/`.
- ▶ The case of libraries is a little more complicated:
  - ▶ In `<prefix>/lib`, the library itself (a `.so.<version>`), a few symbolic links, and the libtool description file (a `.la` file)
  - ▶ The pkgconfig description file in `<prefix>/lib/pkgconfig`
  - ▶ Include files in `<prefix>/include/`
  - ▶ Sometimes a `<libname>-config` program in `<prefix>/bin`
  - ▶ Documentation in `<prefix>/share/man` or `<prefix>/share/doc/`



# Installation (2)

## Contents of /usr after installation of zlib and libpng

./lib			Libtool description file
./lib/libpng12.la	←		Static version of the library
./lib/libpng.la		-> libpng12.la	
./lib/libpng12.a	←		Dynamic version of the library
./lib/libpng.a		-> libpng12.a	
./lib/libpng.so.3.32.0	←		
./lib/libpng12.so.0.32.0			
./lib/libpng12.so.0		-> libpng12.so.0.32.0	
./lib/libpng12.so		-> libpng12.so.0.32.0	
./lib/libpng.so		-> libpng12.so	
./lib/libpng.so.3		-> libpng.so.3.32.0	
./lib/pkgconfig			Pkgconfig description file
./lib/pkgconfig/libpng.pc		-> libpng12.pc	
./lib/pkgconfig/libpng12.pc	←		Zlib dynamic library
./lib/libz.so.1.2.3	←		
./lib/libz.so		-> libz.so.1.2.3	
./lib/libz.so.1		-> libz.so.1.2.3	



# Installation in the build and target spaces

- ▶ From all these files, everything except documentation is necessary to build an application that relies on libpng.
  - ▶ These files will go into the «build space»
- ▶ However, only the library binary in `<prefix>/lib` and some symbolic links are needed to execute the application on the target.
  - ▶ Only these files will go in the «target space»
- ▶ The build space must be kept in order to build other applications or recompile existing applications.



# Let's find the libraries

- ▶ When compiling an application or a library that relies on other libraries, the build process by default looks in `/usr/lib` for libraries and `/usr/include` for headers.
- ▶ The first thing to do is to set the `CFLAGS` and `LDFLAGS` environment variables:  

```
export CFLAGS=-I/my/build/space/usr/include/  
export LDFLAGS=-L/my/build/space/usr/lib
```
- ▶ The `libtool` files (`.la` files) must be modified because they include the absolute paths of the libraries:
  - `libdir='/usr/lib'`
  - + `libdir='/my/build/space/usr/lib'`
- ▶ The `PKG_CONFIG_PATH` environment variable must be set to the location of the `.pc` files and the `PKG_CONFIG_SYSROOT_DIR` variable must be set to the build space directory.



# pkg-config

- ▶ `pkg-config` is a tool that allows to query a small database to get information on how to compile programs that depend on libraries
- ▶ The database is made of `.pc` files, installed by default in `<prefix>/lib/pkgconfig/`.
- ▶ `pkg-config` is used by the configure scripts to get the library configurations
- ▶ It can also be used manually to compile an application:  

```
arm-linux-gcc -o test test.c $(pkg-config --libs --cflags thelib)
```
- ▶ By default, `pkg-config` looks in `/usr/lib/pkgconfig` for the `*.pc` files, and assumes that the paths in these files are correct.
- ▶ `PKG_CONFIG_PATH` allows to set another location for the `*.pc` files and `PKG_CONFIG_SYSROOT_DIR` to prepend a prefix to the paths mentioned in the `.pc` files.



# Practical lab – Manual cross-compiling

- ▶ Manually cross-compiling applications and libraries
- ▶ Learning about common techniques and issues.





# System building tools: principle

- ▶ Different tools are available to automate the process of building a target system, including the kernel, and sometimes the toolchain.
- ▶ They automatically download, configure, compile and install all the components in the right order, sometimes after applying patches to fix cross-compiling issues.
- ▶ They already contain a large number of packages, that should fit your main requirements, and are easily extensible.
- ▶ The build becomes reproducible, which allows to easily change the configuration of some components, upgrade them, fix bugs, etc.



# Available system building tools

Large choice of tools

- ▶ **Buildroot**, developed by the community  
<http://www.buildroot.net>
- ▶ **PTXdist**, developed by Pengutronix  
[http://www.pengutronix.de/software/ptxdist/index\\_en.html](http://www.pengutronix.de/software/ptxdist/index_en.html)
- ▶ **OpenWRT**, originally a fork of Buildroot for wireless routers, not a more generic project  
<http://www.openwrt.org>
- ▶ **LTIB**, developed mainly by Freescale. Good support for Freescale boards, but small community  
<http://www.bitshrine.org/>
- ▶ **OpenEmbedded**, more flexible but also far more complicated  
<http://www.openembedded.org>
- ▶ Vendor specific tools (silicon vendor or embedded Linux vendor)



# Buildroot (1)

- ▶ Allows to build a toolchain, a root filesystem image with many applications and libraries, a bootloader and a kernel image
  - ▶ Or any combination of the previous items
- ▶ Supports building uClibc toolchains only, but can use external uClibc or glibc toolchains
- ▶ Over 500+ applications or libraries integrated, from basic utilities to more elaborate software stacks: X.org, Gstreamer, Qt, Gtk, Webkit, etc.
- ▶ Good for small to medium embedded systems, with a fixed set of features
  - ▶ No support for generating packages (.deb or .ipk)
  - ▶ Needs complete rebuild for most configuration changes.
- ▶ Active community, releases published every 3 months.



# Buildroot (2)

- ▶ Configuration takes place through a *\*config* interface similar to the kernel  
`make menuconfig`
- ▶ Allows to define
  - ▶ Architecture and specific CPU
  - ▶ Toolchain configuration
  - ▶ Set of applications and libraries to integrate
  - ▶ Filesystem images to generate
  - ▶ Kernel and bootloader configuration
- ▶ Build by just running  
`make`

```
/home/thomas/local/buildroot/.config - buildroot v2010.11-git Configuration

Buildroot Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> selects a feature,
while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for
Help, </> for Search. Legend: [*] feature is selected [ ] feature

Target Architecture (arm) --->
Target Architecture Variant (arm926t) --->
Target ABI (EABI) --->
Build options --->
Toolchain --->
System configuration --->
Package Selection for the target --->
Target filesystem options --->
Bootloaders --->
Kernel --->

---
Load an Alternate Configuration File
Save an Alternate Configuration File

<Select> <Exit> <Help>
```



# Buildroot: adding a new package (1)

- ▶ A package allows to integrate a user application or library to Buildroot
- ▶ Each package has its own directory (such as `package/gqview`). This directory contains:
  - ▶ A `Config.in` file (mandatory), describing the configuration options for the package. At least one is needed to enable the package. This file must be sourced from `package/Config.in`
  - ▶ A `gqview.mk` file (mandatory), describing how the package is built.
  - ▶ Patches (optional). Each file of the form `gqview-*.patch` will be applied as a patch.



# Buildroot: adding a new package (2)

- ▶ For a simple package with a single configuration option to enable/disable it, the `Config.in` file looks like:

```
config BR2_PACKAGE_GQVIEW
    bool "gqview"
    depends on BR2_PACKAGE_LIBGTK2
    help
        GQview is an image viewer for Unix operating systems

        http://prdownloads.sourceforge.net/gqview
```

- ▶ It must be sourced from `package/Config.in`:

```
source "package/gqview/Config.in"
```



# Buildroot: adding new package (3)

- ▶ Create the `gqview.mk` file to describe the build steps

```
GQVIEW_VERSION = 2.1.5
GQVIEW_SOURCE = gqview-$(GQVIEW_VERSION).tar.gz
GQVIEW_SITE = http://prdownloads.sourceforge.net/gqview
GQVIEW_AUTORECONF = NO
GQVIEW_INSTALL_STAGING = NO
GQVIEW_INSTALL_TARGET = YES
GQVIEW_DEPENDENCIES = host-pkg-config libgtk2

$(eval $(call AUTOTARGETS,package,gqview))
```

- ▶ The last argument of the `AUTOTARGETS` macro, the prefix of all variables must be identical to the suffix of the main configuration option `BR2_PACKAGE_GQVIEW`
- ▶ The `AUTOTARGETS` infrastructure knows how to build autotools packages. A more generic `GENTARGETS` infrastructure is available for packages not using the autotools as their build system.



# OpenEmbedded

- ▶ The most versatile and powerful embedded Linux build system
  - ▶ A collection of recipes (`.bb` files)
  - ▶ A tool that processes the recipes: `bitbake`
- ▶ Integrates 2000+ application and libraries, is highly configurable, can generate packages (`.ipk`) to make the system customizable, supports multiple versions/variants of the same package, no need for full rebuild when the configuration is changed.
- ▶ Configuration takes place by editing various configuration files
- ▶ Good for larger embedded Linux systems, or people looking for more configurability and extensibility
- ▶ Drawbacks: no stable releases, very steep learning curve, very long first build.



# Distributions (1)

Debian GNU/Linux, <http://www.debian.org>



- ▶ Available on ARM, MIPS and PowerPC architectures
- ▶ Provides a ready-to-use filesystem with all the software you need.
- ▶ Huge flexibility thanks to the package management system, but only works only systems with enough storage size (> 300 MB) and RAM (> 64 MB).
- ▶ Software is compiled natively by default.
- ▶ You can build your own root filesystem images on x86 by using the [debootstrap](#) command.
- ▶ [Emdebian](#) is a project to make Debian better for embedded systems: leverage Debian package descriptions, but reduces dependencies, smaller configuration, removes documentation, supports uClibc... See <http://emdebian.org>.





# Distributions (2)



## Ubuntu GNU/Linux

- ▶ Based on Debian, same benefits
- ▶ New release every 6 months, supported for 18 months or even 3 years.
- ▶ Supported on ARM, but only on Cortex A8 and beyond. Supplies Thumb2 binaries. Neon not supported.
- ▶ Good solution for mobile multimedia devices.

## Others

- ▶ Fedora also has support for ARM, but not actively maintained.

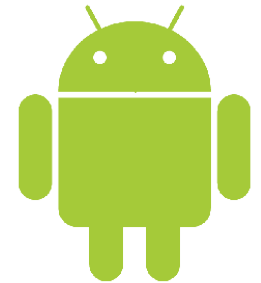


# Embedded distributions

Distributions designed for specific types of devices

- ▶ **Meego:** <http://meego.com/>  
Distribution targeting phones, media players, netbooks, TVs and In Vehicle Infotainment. Backed by Intel and Nokia.
- ▶ **Android:** <http://www.android.com/>  
Google's distribution for phones and tablet PCs. Except the Linux kernel, very different userspace than other Linux distributions. Very successful, lots of applications available (many proprietary).
- ▶ **Ångström:** <http://www.angstrom-distribution.org/>  
Targets PDAs and webpads (**Siemens Simpad...**)  
Binaries available for `arm` little endian.

MeeGo™





# Practical lab – Buildroot

- ▶ Rebuild the same system, this time with Buildroot.
- ▶ See how easier it gets!
- ▶ Adding your own DirectFB based application.





## GNU / Linux workstation Emulators



<http://fabrice.bellard.free.fr/qemu/>

Fast processor emulator  
using a portable dynamic translator.



## Full system emulation

- ▶ Emulates the processor and various peripherals  
Supported: `x86`, `x86_64`, `ppc`, `arm`, `sparc`, `mips`, `m68k`
- ▶ To know which machine types are supported:  
`qemu-system-arm -M ?`
- ▶ `i386`, `x86_64` system emulation: now close to native speeds  
thanks to the `kqemu` kernel module (now GPL v2!).



# Other emulators

- ▶ ARM platform

- ▶ **SkyEye**: <http://skyeeye.sourceforge.net>

- Emulates several **ARM** platforms (**AT91**, **Xscale**...) and can boot several operating systems (**Linux**, **uClinux**, and others)

- ▶ **Softgun**: <http://softgun.sourceforge.net>

- Virtual **ARM** system with many virtual on-board peripherals. Boots **Linux**.

- ▶ **SWARM** - Software **ARM** - **arm7** emulator

- <http://www.cl.cam.ac.uk/~mwd24/phd/swarm.html>

- Can run **uClinux**

- ▶ **ColdFire** emulator

- <http://www.slicer.ca/coldfire/>

- Can boot **uClinux**



GNU / Linux workstation  
Commercial toolsets



# Commercial toolsets

Caution: *commercial* doesn't mean *proprietary*!

- ▶ Vendors play fair with the GPL and do make their source code available to their users, and most of the time, to the community.
  - ▶ As long as they distribute the sources to their users, the GPL doesn't require vendors to share their sources with any third party.
- ▶ No issue with all the GPL sources developed by or with the community.
- ▶ Graphical toolkits developed by the vendors look proprietary. Their licenses are not advertised on their websites! You have to be a customer to know or get a free preview kit to know.



# Commercial toolset strengths

- ▶ Technical advantages
  - ▶ Well tested and supported kernel and tool versions
  - ▶ Including early patches not supported by the mainstream kernel yet
- ▶ Complete development tool sets: kernels, toolchains, utilities, binaries for impressive lists of target platforms
- ▶ Integrated utilities for automatic kernel image, initrd and filesystem generation.
- ▶ Graphical developments tools
- ▶ Development tools available on multiple platforms: **GNU / Linux**, **Solaris**, **Windows**...
- ▶ Support services
  - ▶ Useful if you don't have your own support resources
  - ▶ Long term support commitment, even for versions considered as obsolete by the community, but not by your users!



# Wind River

Wind River Linux:

<http://www.windriver.com/products/linux/>

**WIND RIVER**

- ▶ New market leader, recently acquired by Intel
- ▶ A lot of embedded and real-time experience from **VxWorks**.
- ▶ Now say they integrate, test and support Linux as rigorously as they do with **VxWorks**.  
Linux development supported with their Workbench integrated development environment, already used for **VxWorks**.
- ▶ Support standard and recent Linux kernel sources, including real-time preempt patches (Linux 2.6.27). Also offer hard real-time Linux (**Real Time Core**: formerly **RTLinux**).



# Montavista



<http://www.mvista.com/>

The second market leader

- Employs some of the most active kernel hackers, in particular on the [arm](#) platform.
- Kernel development eventually shared with the community. kernel. Many drivers merged in mainstream Linux.
- Graphical development tools are proprietary.



<http://timesys.com>

- Similar toolset offering as other vendors. Great flexibility available to their LinuxLink™ subscribers
- Community friendly: they share very interesting and generic technical whitepapers and articles. They also employ key community hackers (Thomas Gleixner, Rob Landley...).
- Free Software BSPs (Board Support Packages) available.
- **Linux** soft and hard real-time OS product.
- Development tools seem to be proprietary.



# Sysgo - Koan Software

<http://sysgo.com>



- ▶ ELinOS development toolset, in particular based on Eclipse and the Linux Trace Toolkit.
- ▶ Includes FreeToolBox, a freely downloadable compiling and rootfs creating toolchain.
- ▶ Supports i386, arm and ppc.
- ▶ Hard real-time support with their own microkernel (PikeOS), an approach similar to RTAI.

<http://koansoftware.com>



- Makers of KaeilOS ([http://koansoftware.com/kaeilos/index\\_en.htm](http://koansoftware.com/kaeilos/index_en.htm)), a GPL embedded Linux distribution for industrial applications.
- KaeilOS supports i386 and popular arm platforms. Other platforms supported upon request.
- Includes several graphical toolkits and supports hard real-time (RTAI, Xenomai, preemption patches).
- Unfortunately, KaeilOS is GPL but not available for public download.



# Denx Software Engineering



<http://denx.de>

- ▶ Created by Wolfgang Denk, the author of the U-Boot bootloader.
- ▶ Create and support the Embedded Linux Development Kit (ELDK), a complete and well documented development environment.
- ▶ This kit is not only Free Software, it can be downloaded freely by anyone.
- ▶ A great community member and contributor!



# Commercial toolsets - Summary

- ▶ Major vendors: **MontaVista**, **Wind River**, **TimeSys**  
Involved in Linux development.
- ▶ Smaller vendors: **Koan**, **Sysgo**, **Denx...**  
Trying to differentiate their products.
- ▶ Community based companies: **Denx**, **CodeSourcery**  
Contribute to community tools.  
Mainly offer support and development.



# Commercial or community solutions?

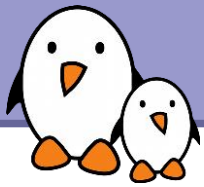
## Commercial distributions and toolsets

- ▶ Best if you don't have your own support resources and have a sufficient budget
- ▶ Really help focusing on your real job: making an embedded device.
- ▶ You can even subcontract driver development to the vendor

## Community distributions and tools

- ▶ Best if you are on a tight budget
- ▶ Best if you are willing to build your own embedded Linux expertise and train your own support resources.

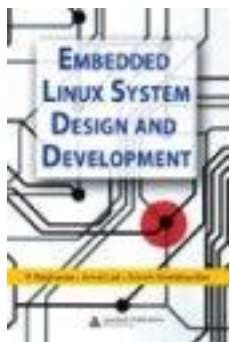
In any case, your products are based on Free Software!



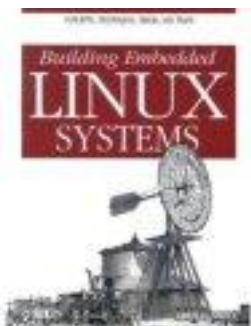
## References



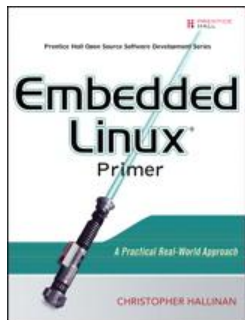
# Useful reading (1)



- ▶ Embedded Linux System Design and Development ★ ★  
P. Raghavan, A. Lad, S. Neelakandan, Auerbach, Dec. 2005.  
<http://free-electrons.com/redirect/elsdd-book.html>  
Useful book covering most aspects of embedded Linux system development (kernel and tools).



- ▶ Building Embedded Linux Systems, O'Reilly ★ ★ ★  
By Karim Yaghmour, Jon Masters, Gilad Ben-Yossef and Philippe Gerum, and others (including Michael Opdenacker), August 2008  
<http://oreilly.com/catalog/9780596529680/>



- ▶ Embedded Linux Primer, Prentice Hall ★ ★ ★  
By Christopher Hallinan, September 2006  
Covers a very wide range of interesting topics.



# Useful reading (2)

▶ <http://www.denx.de/wiki/DULG/Manual>



Lots of useful command examples, generic help and advice for embedded Linux systems.

See <http://www.linuxdevices.com/articles/AT2969812114.html> for more books on Linux for embedded systems.



# Useful web sites

LinuxDevices.com: <http://linuxdevices.com>

- ▶ Weekly newsletter with news and announcements about embedded devices running Linux.
- ▶ Articles, whitepapers, and Linux embedded devices catalog.
- ▶ An excellent site to follow industry news!





# International conferences

Useful conferences featuring embedded Linux and kernel topics

- ▶ Embedded Linux Conference: <http://embeddedlinuxconference.com/>  
Organized by the CE Linux Forum: California  
(San Francisco, April), in Europe (October-November).  
Very interesting kernel and userspace topics for embedded systems developers. Presentation slides freely available
- ▶ Linux Plumbers  
<http://linuxplumbersconf.org>  
Conference on the low-level plumbing of Linux: kernel, audio, power management, device management, multimedia, etc.
- ▶ Fosdem: <http://fosdem.org> (Brussels, February)  
For developers. Presentations about system development.
- ▶ Don't miss our free conference videos on  
[http://free-electrons.com/community/videos/conferences/!](http://free-electrons.com/community/videos/conferences/)



# Related documents

**Free Electrons**  
Embedded Freedom

HOME DEVELOPMENT SERVICES TRAINING DOCS COMMUNITY COMPANY BLOG

**Recent blog posts**

- ELC Europe in Grenoble
- Free Electrons at ELC
- Linux kernel 2.6.29 - New features for embedded users
- The Buildroot project begins a new life
- FOSDEM 2009 videos
- USB-Ethernet device for Linux
- Program for Embedded Linux Conference 2009 announced
- Public session changes
- Real hardware in our training sessions
- Call for presentations for the LSM embedded track

**Docs**

Most of the below documents are presentations used in our [training sessions](#), or in technical conferences.

**License**

All our documents are available under the terms of the [Creative Commons Attribution-ShareAlike 3.0 license](#). This essentially means that you are free to download, distribute and even modify them, provided you mention us as the original authors and that you share these documents under the same conditions.

**Linux kernel**

- [Embedded Linux kernel and driver development](#)
- [New features in Linux 2.6](#) (since 2.6.10)
- [Kernel initialization](#)
- [Porting Linux to new hardware](#)
- [Power management in Linux](#)
- [Linux PCI drivers](#)
- [Block device drivers](#)
- [Linux USB drivers](#)
- [DMA](#)

**Architecture specific documents**

- [ARM Linux specifics](#)
- [Linux on TI OMAP processors](#)

**Embedded Linux system development**

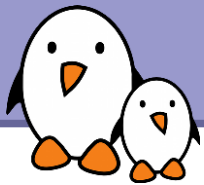
- [Embedded Linux system development](#)
- [Real time in embedded Linux systems](#)
- [Block filesystems](#)
- [Flash filesystems](#)
- [Free software development tools](#)
- [The U-boot bootloader](#)
- [The GRUB bootloader](#)
- [The blob bootloader](#)
- [Hotplugging with udev](#)
- [Introduction to uClinux](#)
- [Java in embedded Linux](#)
- [Embedded Linux optimizations](#)
- [Audio in embedded Linux systems](#)
- [Multimedia in embedded Linux systems](#)
- [Embedded Linux From Scratch... in 40 minutes!](#)
- [Building embedded Linux systems with Buildroot](#)
- [Developing embedded distributions with OpenEmbedded](#)
- [The Scratchbox development environment](#)

**Miscellaneous**

- [Introduction to the Unix command line](#)
- [SSH](#)
- [Linux virtualization solutions \(with an embedded perspective\)](#)
- [Advantages of Free Software and Open Source in embedded systems](#)
- [Introduction to GNU/Linux and Free Software](#)

All our technical presentations on <http://free-electrons.com/docs>

- ▶ Linux kernel
- ▶ Device drivers
- ▶ Architecture specifics
- ▶ Embedded Linux system development



# How to help

You can help us to improve and maintain this document...

- ▶ By sending corrections, suggestions, contributions and translations
- ▶ By asking your organization to order development, consulting and training services performed by the authors of these documents (see <http://free-electrons.com/>).
- ▶ By sharing this document with your friends, colleagues and with the local Free Software community.
- ▶ By adding links on your website to our on-line materials, to increase their visibility in search engine results.

## Linux kernel

- Linux device drivers
- Board support code
- Mainstreaming kernel code
- Kernel debugging

## Embedded Linux Training

***All materials released with a free license!***

- Unix and GNU/Linux basics
- Linux kernel and drivers development
- Real-time Linux, uClinux
- Development and profiling tools
- Lightweight tools for embedded systems
- Root filesystem creation
- Audio and multimedia
- System optimization

# Free Electrons

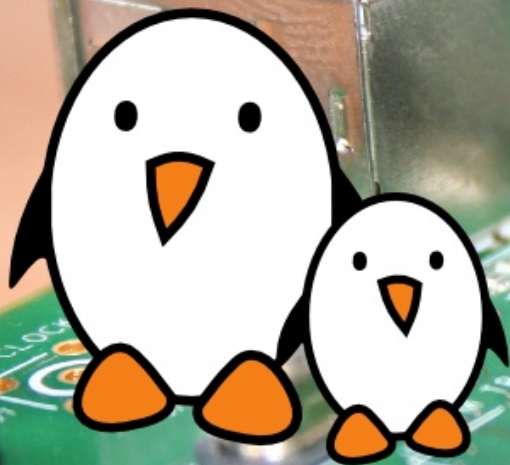
## Our services

### Custom Development

- System integration
- Embedded Linux demos and prototypes
- System optimization
- Application and interface development

### Consulting and technical support

- Help in decision making
- System architecture
- System design and performance review
- Development tool and application support
- Investigating issues and fixing tool bugs



**Free Electrons**  
Embedded Linux Experts

<http://free-electrons.com>