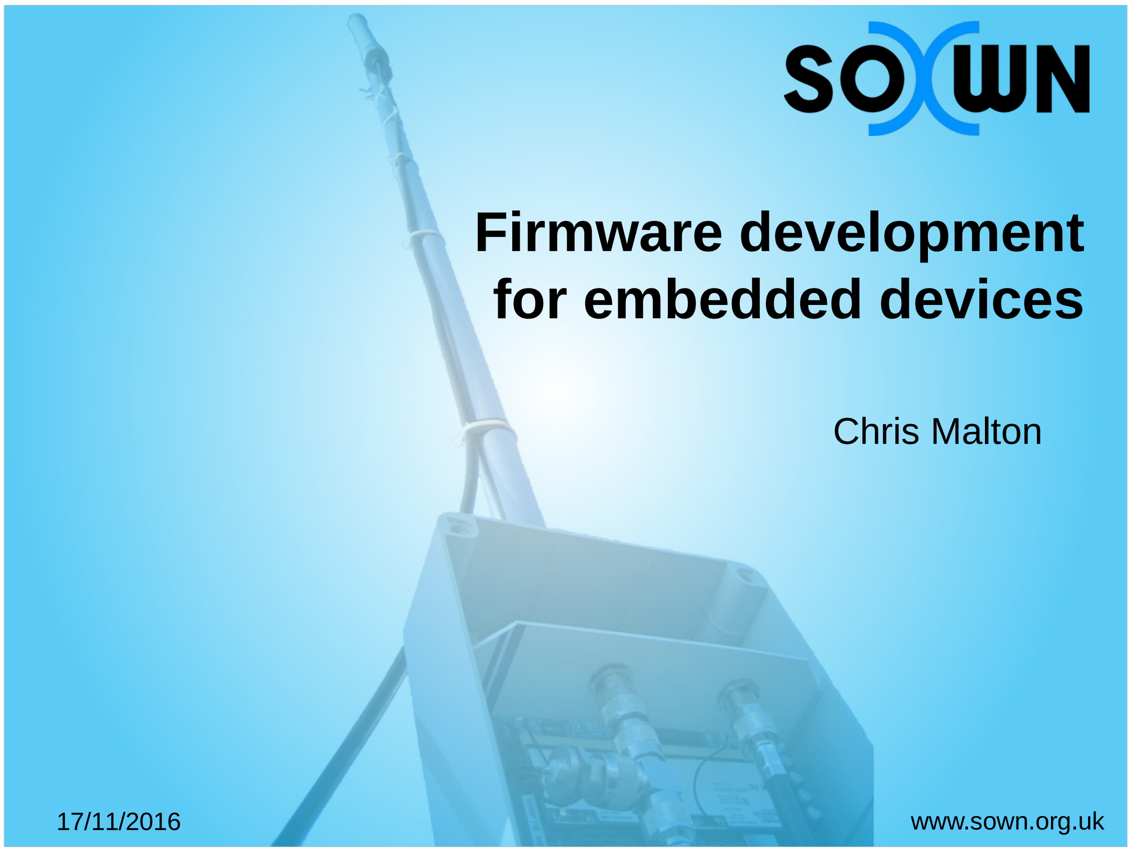


SOWN

Firmware development for embedded devices

Chris Malton

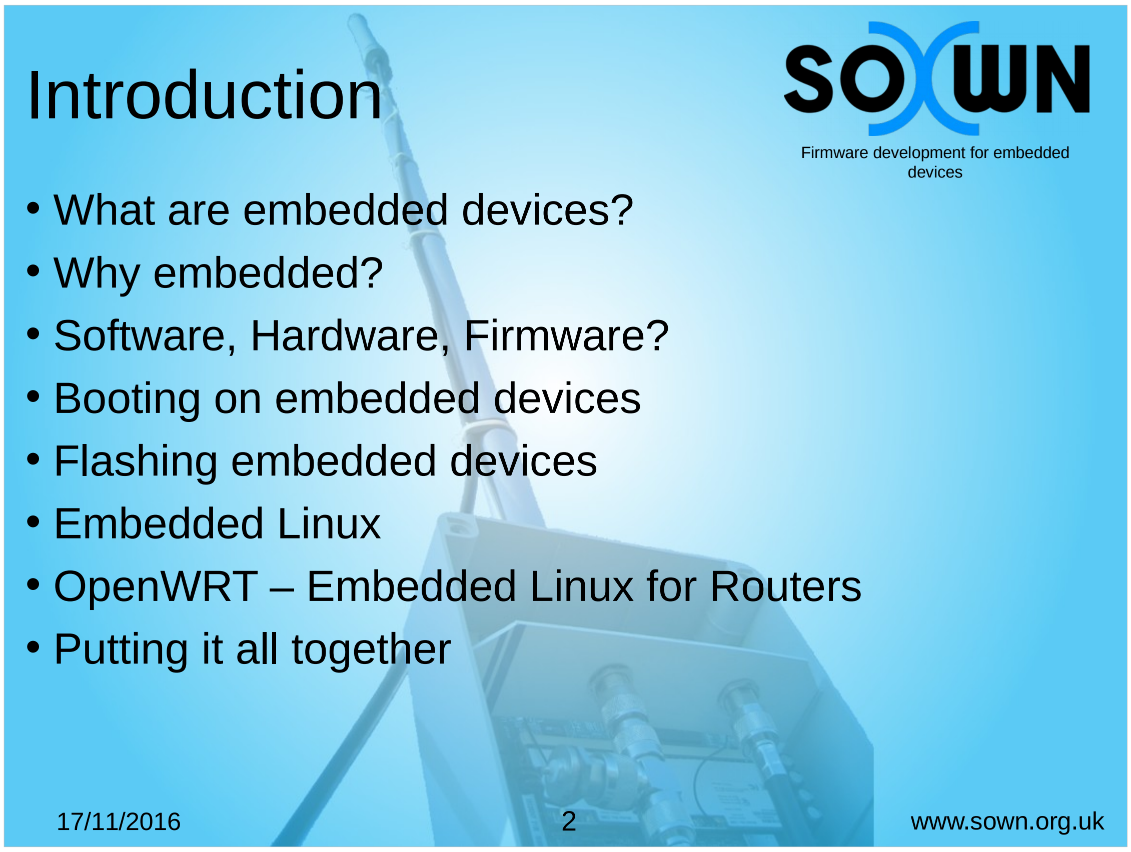
17/11/2016

www.sown.org.uk

About me

- Been with SOWN since 2009!
- Network engineer at Vostron
- Now work for swlines and Realtime Trains
    - Includes embedded hardware development

Node firmware management for the latest generation of SOWN nodes.



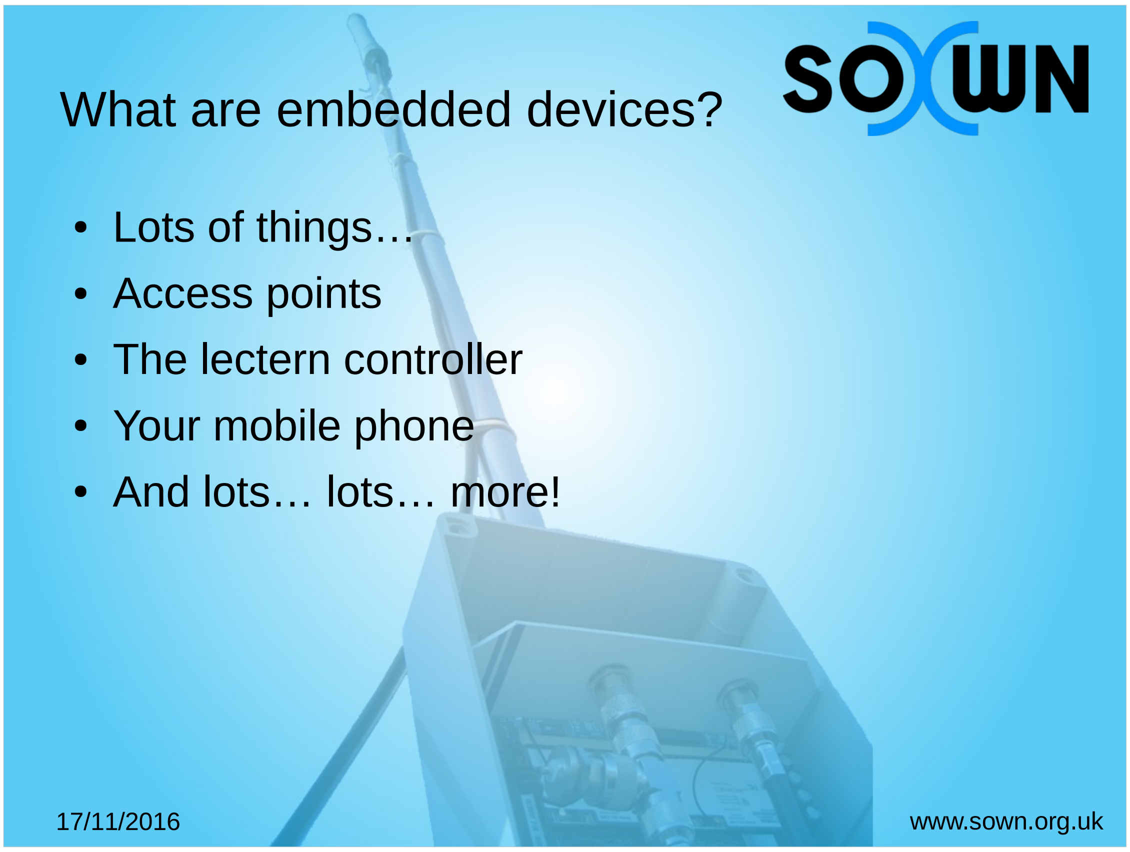
Introduction

Firmware development for embedded devices

- What are embedded devices?
- Why embedded?
- Software, Hardware, Firmware?
- Booting on embedded devices
- Flashing embedded devices
- Embedded Linux
- OpenWRT – Embedded Linux for Routers
- Putting it all together

17/11/2016 2 www.sown.org.uk

Lot of content to cover
Not a lot of time
Quite text-heavy presentation.
We'll start with details of what embedded devices are, then move on to define the differences between software hardware and firmware.
Booting on embedded devices is a complex process and we'll skip over much of the details. With apologies in advance to those who think I've over-simplified it!
Once we've looked at how you boot on an embedded device, we'll look at how you get firmware onto it to start with.
Then we'll consider what Embedded Linux is, and consider a specific type of embedded linux. Then we'll put it all together.



What are embedded devices?

- Lots of things…
- Access points
- The lectern controller
- Your mobile phone
- And lots… lots… more!

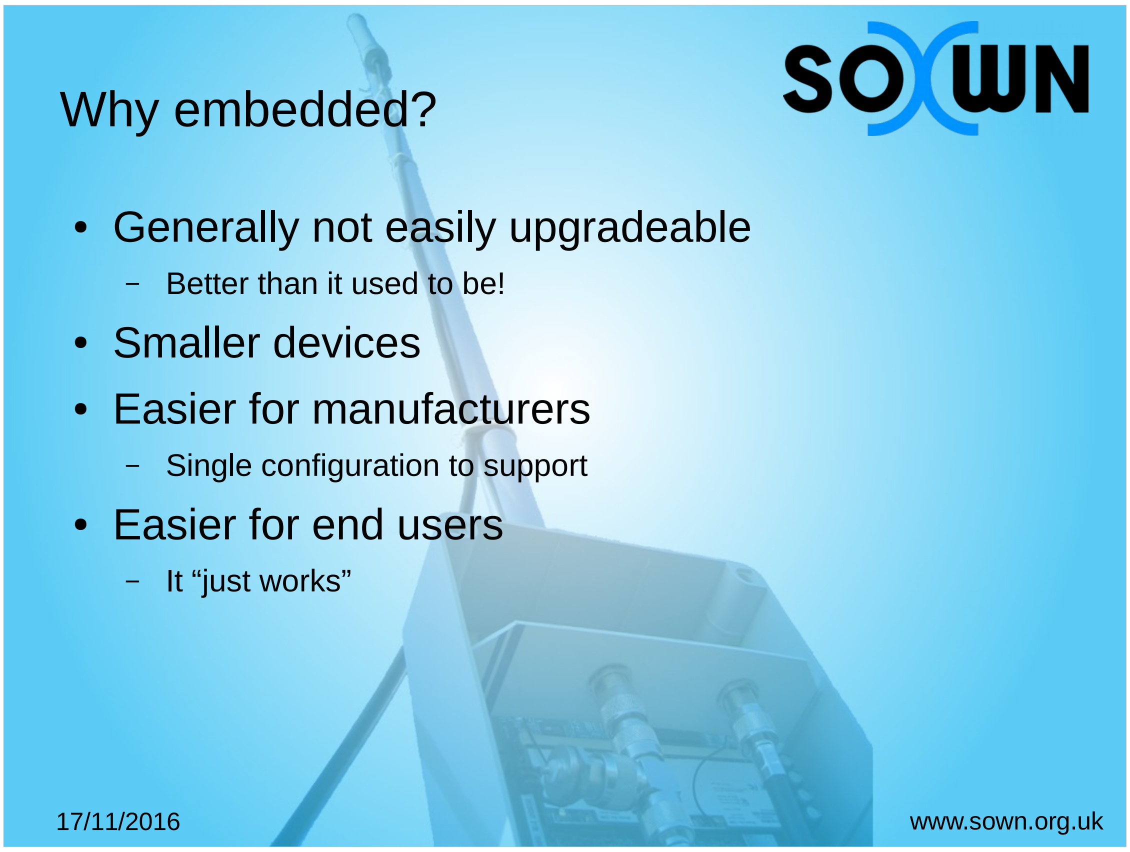
17/11/2016 www.sown.org.uk

There are lots and lots and lots of embedded devices. LOTS!

That includes…. Access points – like the one above me, the lectern controller that's controlling the projector. Even your mobile phone…

There's literally hundreds of them everywhere. For those of you with amateur radios, your radios are embedded devices.

Embedded normally refers to something running some low-level code and not being frequently updated.



Why embedded?

- Generally not easily upgradeable
  - Better than it used to be!
- Smaller devices
- Easier for manufacturers
  - Single configuration to support
- Easier for end users
  - It “just works”

17/11/2016 www.sown.org.uk

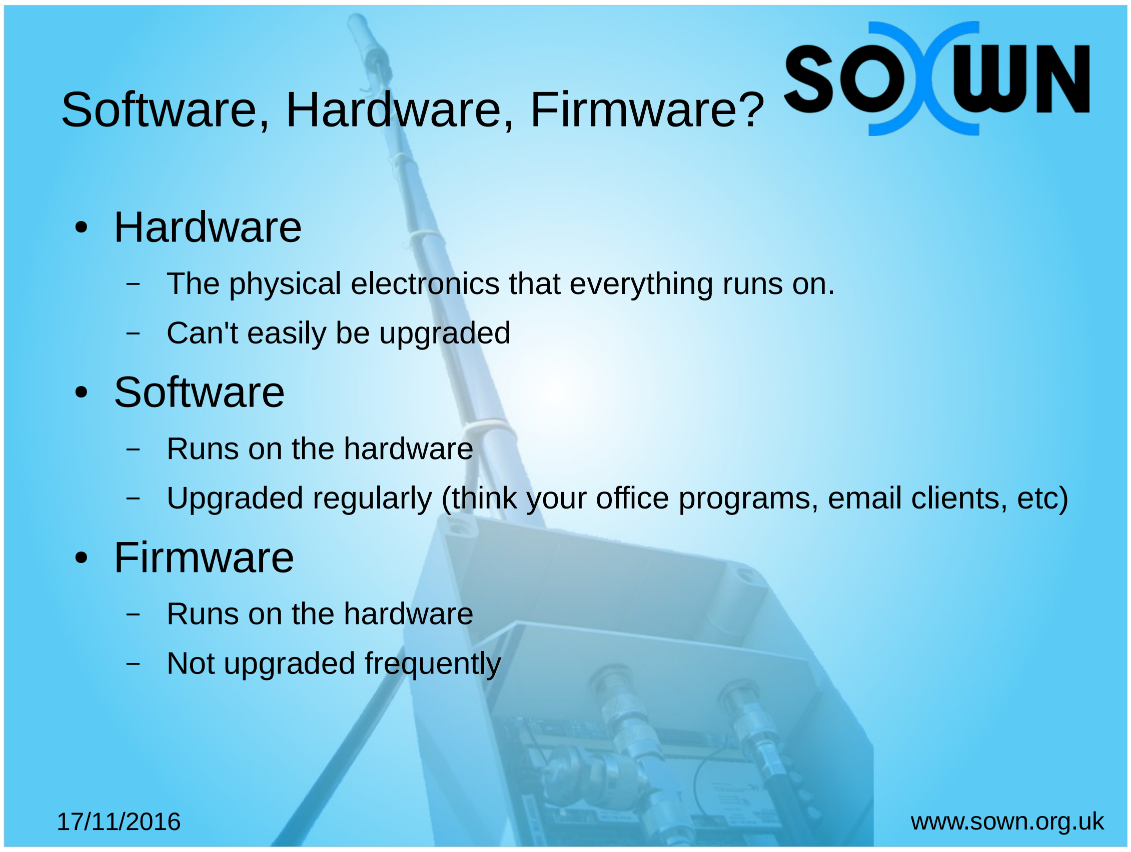
Embedded devices are often generally fairly static and can't be readily updated. This is a great idea, but sometimes you just need to fix things.

These days, things have got better and it's now much, much easier to change things. This isn't necessarily a good thing from a security perspective.

It allows manufacturers to make smaller devices as they can use chips designed for specific tasks rather than lots of individual chips for each feature.

This leads to an easier support arrangement for the manufacturer because there's only one configuration to support.

Embedded devices have to “just work” for the end user because they normally aren't user servicable.



Software, Hardware, Firmware?

- Hardware
  - The physical electronics that everything runs on.
  - Can't easily be upgraded
- Software
  - Runs on the hardware
  - Upgraded regularly (think your office programs, email clients, etc)
- Firmware
  - Runs on the hardware
  - Not upgraded frequently

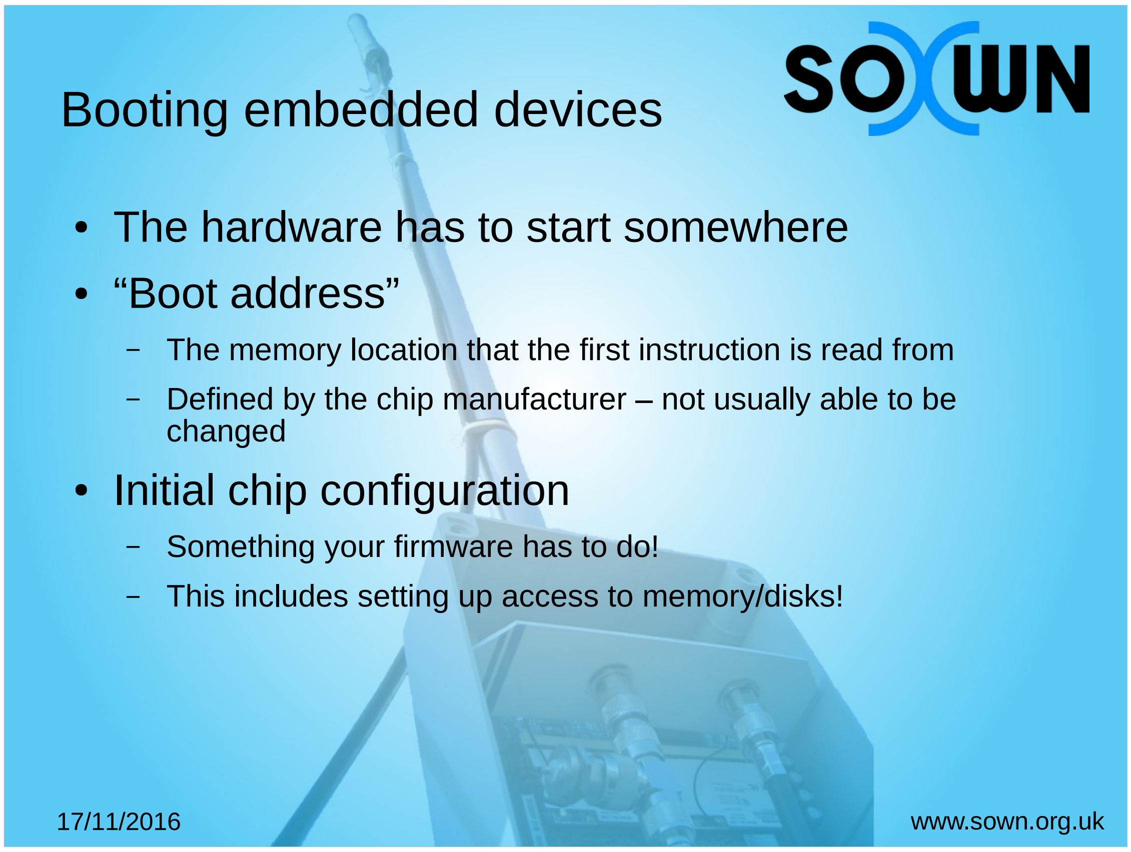
17/11/2016 www.sown.org.uk

Hardware is the physical devices that are running code. They are manufactured to specifications and usually cannot be modified after production. They have to made right first time.

Software, on the other hand, is the exact opposite. It can be modified regularly and runs on hardware. Therefore, as long as software supports self-upgrade you can upgrade all the software that runs on a bit of hardware.

Firmware sits somewhere between the two. It runs on hardware, but isn't normally upgraded regularly. If you think of a standard PC, the BIOS could be regarded as firmware. It is essentially the first bit of code that gets loaded and is responsible for loading the rest of the software.

PC Boot sequence: Firmware (BIOS) → Software (Boot loader) → Software (Windows/Linux) – Macs are a little bit different!



Booting embedded devices

- The hardware has to start somewhere
- “Boot address”
  - The memory location that the first instruction is read from
  - Defined by the chip manufacturer – not usually able to be changed
- Initial chip configuration
  - Something your firmware has to do!
  - This includes setting up access to memory/disks!

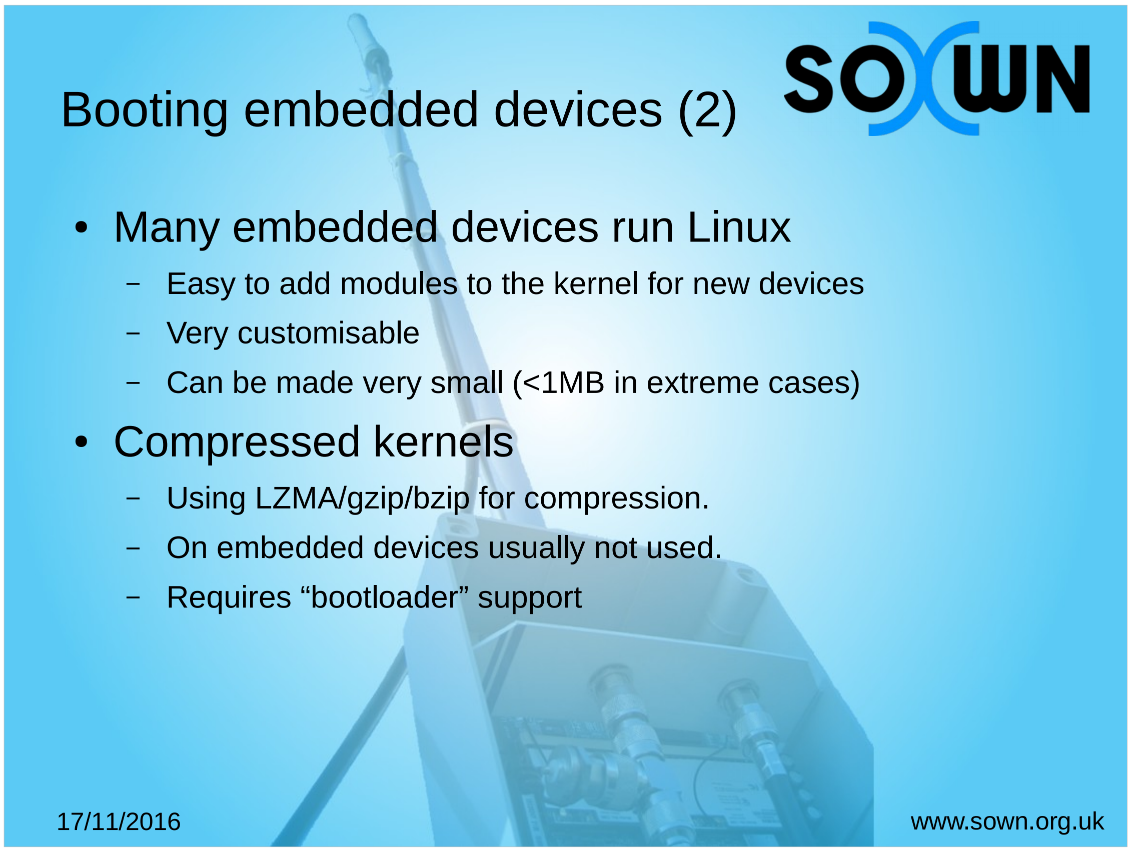
17/11/2016 www.sown.org.uk

Every bit of hardware has to start running code from somewhere. This is called the Boot Address. This is normally a small space of non-volatile memory.

On a desktop PC, this is usually the processor microcode, which reads the BIOS into memory and starts executing it.

On embedded devices, they normally just boot straight into code from the built-in electronically erasable programmable read only memory (EEPROM).

The first thing you need to do in this microcode or EEPROM-stored program is initialise hardware, timing parameters, access to other memory and disks (if present).



Booting embedded devices (2)

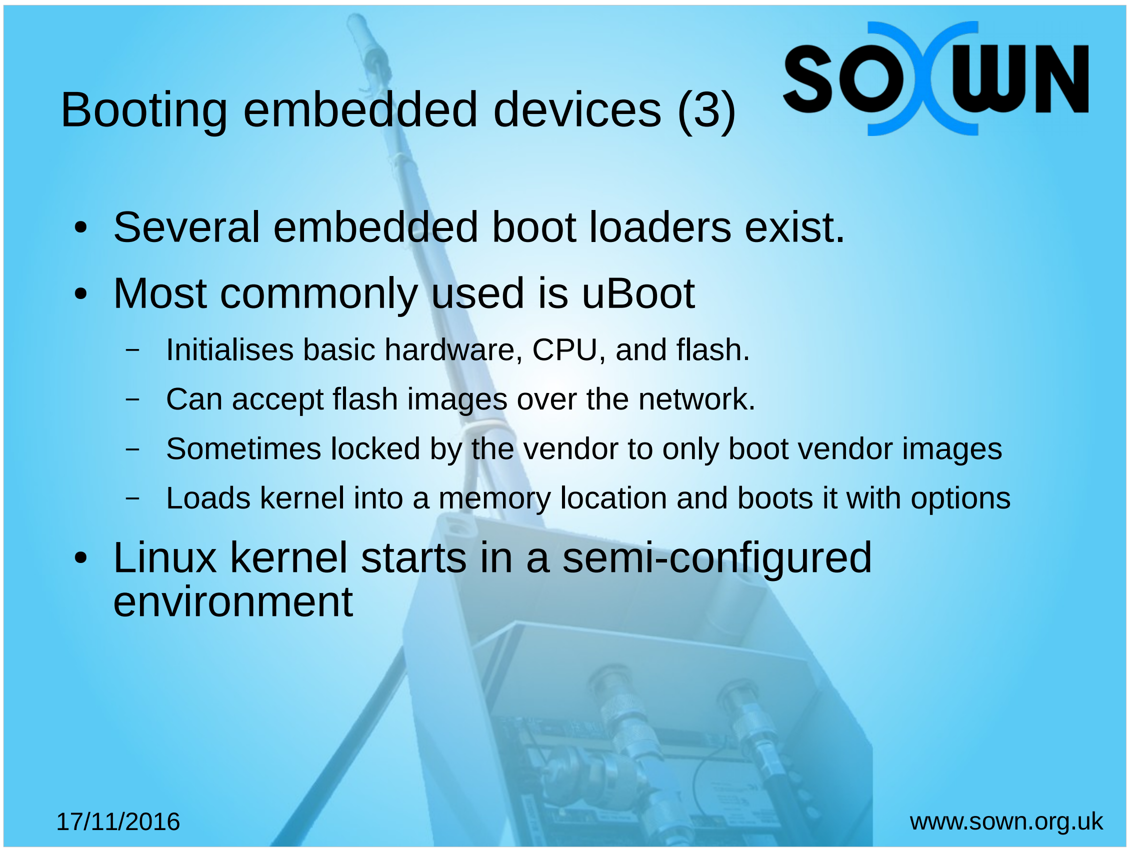
- Many embedded devices run Linux
  - Easy to add modules to the kernel for new devices
  - Very customisable
  - Can be made very small (<1MB in extreme cases)
- Compressed kernels
  - Using LZMA/gzip/bzip for compression.
  - On embedded devices usually not used.
  - Requires “bootloader” support

17/11/2016 www.sown.org.uk

Somewhere in the middle of it all, there are a whole collection of (normally) ARM-based devices. These are system-on-chip machines (think like the Raspberry Pi). Many of these support Linux as an operating system.

Linux, being a module-based operating system, can have a very small kernel, with only the necessary drivers to load the rest of the modules. In extreme cases, this can be as small as under 1MB, although typically most Linux kernels are 1-2MB.

Compressing a kernel is often a good way to save space – however, needs support from whatever launches the kernel (normally a bootloader). This can reduce the kernel by as much as 50% in some cases, but normally this is not used on embedded devices as compression and decompression are slow and complex routines to implement and are normally software driven.



Booting embedded devices (3)

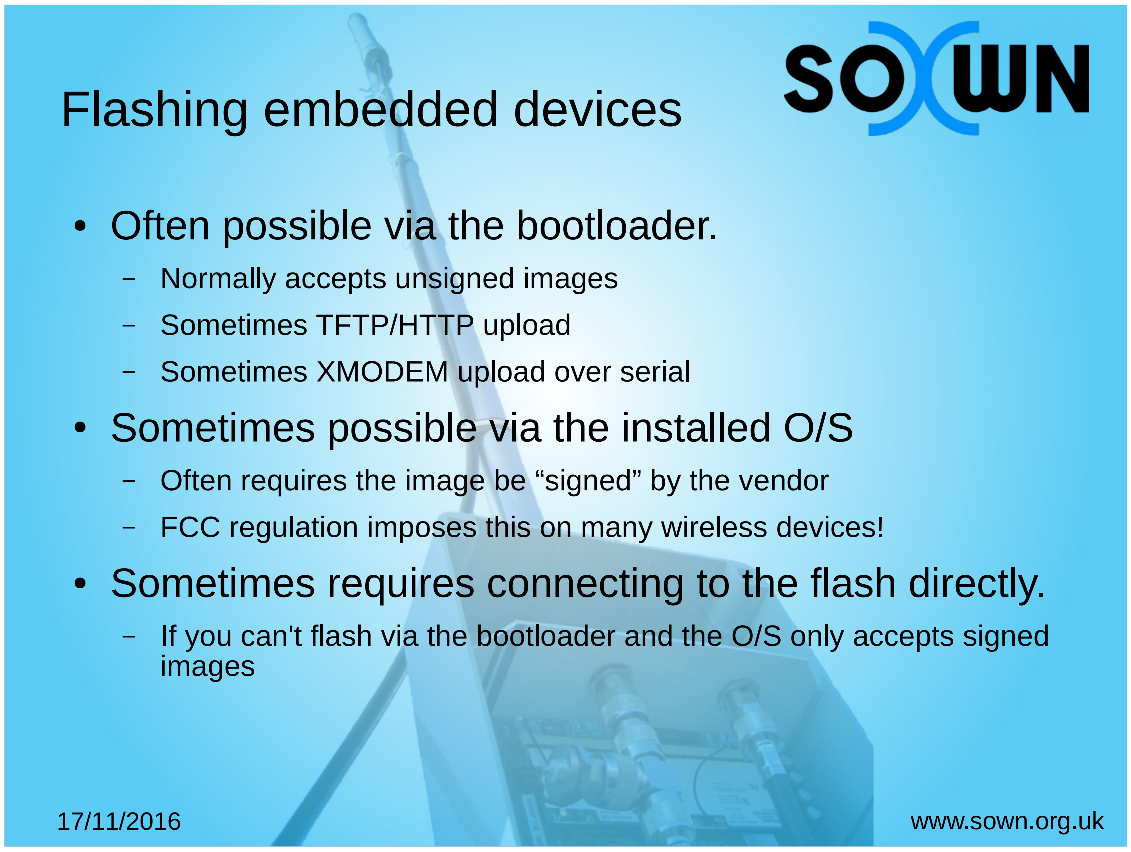
- Several embedded boot loaders exist.
- Most commonly used is uBoot
  - Initialises basic hardware, CPU, and flash.
  - Can accept flash images over the network.
  - Sometimes locked by the vendor to only boot vendor images
  - Loads kernel into a memory location and boots it with options
- Linux kernel starts in a semi-configured environment

17/11/2016 www.sown.org.uk

Most embedded devices start the Linux kernel in a partially configured environment. This is normally configured by a boot-loader – the most common one used on embedded devices is uBoot – pronounced you-boot.

It's a very small boot-loader that initialises basic hardware operations, usually allows reflashing the device. It's also open source, which means that hardware vendors can change things….. and they do.

In response to recent FCC rulings, some vendors have required digitally signed images to restrict customised firmware being flashed onto devices.



Flashing embedded devices

- Often possible via the bootloader.
  - Normally accepts unsigned images
  - Sometimes TFTP/HTTP upload
  - Sometimes XMODEM upload over serial
- Sometimes possible via the installed O/S
  - Often requires the image be "signed" by the vendor
  - FCC regulation imposes this on many wireless devices!
- Sometimes requires connecting to the flash directly.
  - If you can't flash via the bootloader and the O/S only accepts signed images

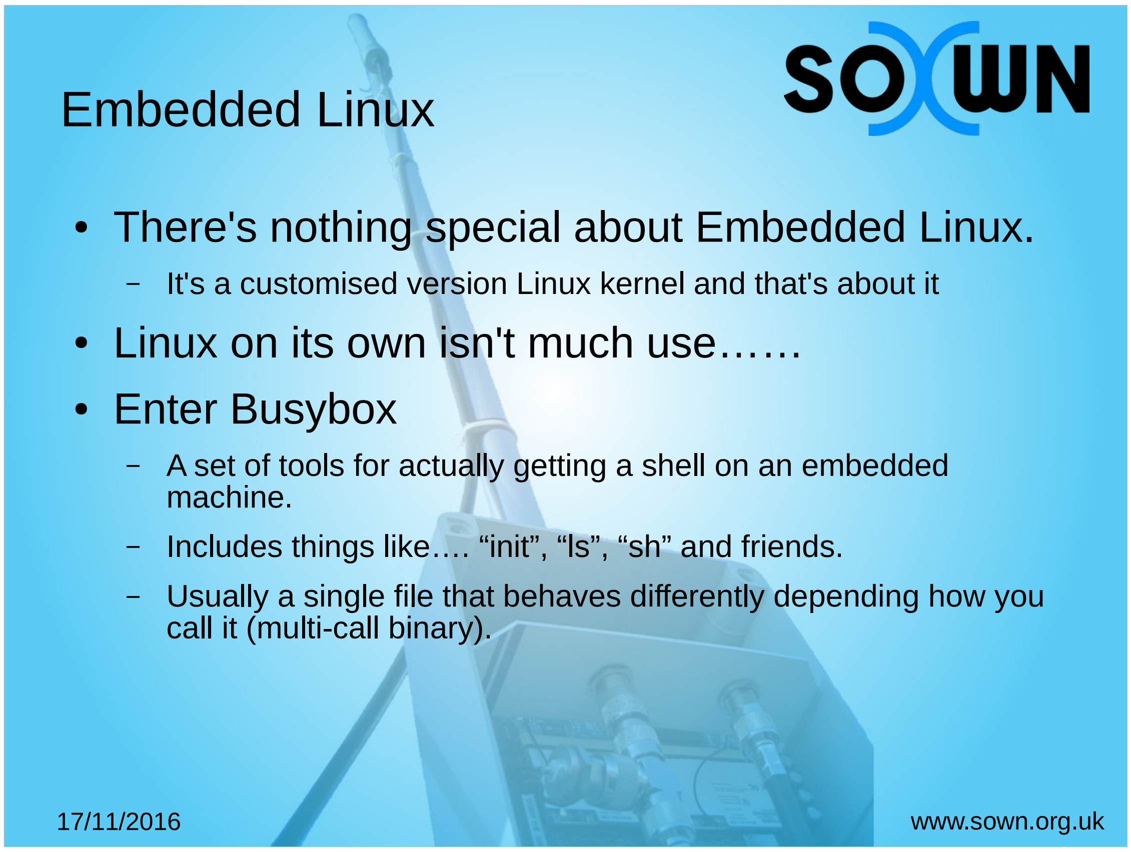
17/11/2016 www.sown.org.uk

Flashing embedded devices requires support on the device for rewriting the flash memory chip. Most devices now support this and very few do not.

The bootloader is the most common way of accepting new images. It often accepts unsigned images, but not on all hardware. Flashing methods vary. Some will copy from network, others require more elaborate flashing schemes.

The running O/S can also write to the flash. In the case of some devices, this doesn't overwrite the running image, only the startup image so changes are only reflected after a reboot.

In rare cases, you can't flash a device without directly connecting to the flash itself. This sometimes is via the software debugging port, but the most common method using the JTAG protocol and port. This is quite an involved process.



Embedded Linux

SOWN

- There's nothing special about Embedded Linux.
  - It's a customised version Linux kernel and that's about it
- Linux on its own isn't much use……
- Enter Busybox
  - A set of tools for actually getting a shell on an embedded machine.
  - Includes things like…. "init", "ls", "sh" and friends.
  - Usually a single file that behaves differently depending how you call it (multi-call binary).

17/11/2016 www.sown.org.uk

Embedded Linux isn't actually anything special. It's a standard Linux image. The Linux kernel won't actually give you anything useful. You need a full distribution. In reality, the "easiest" way to do this is using the "busybox" multi-call binary. It behaves differently depending how you invoke the program.

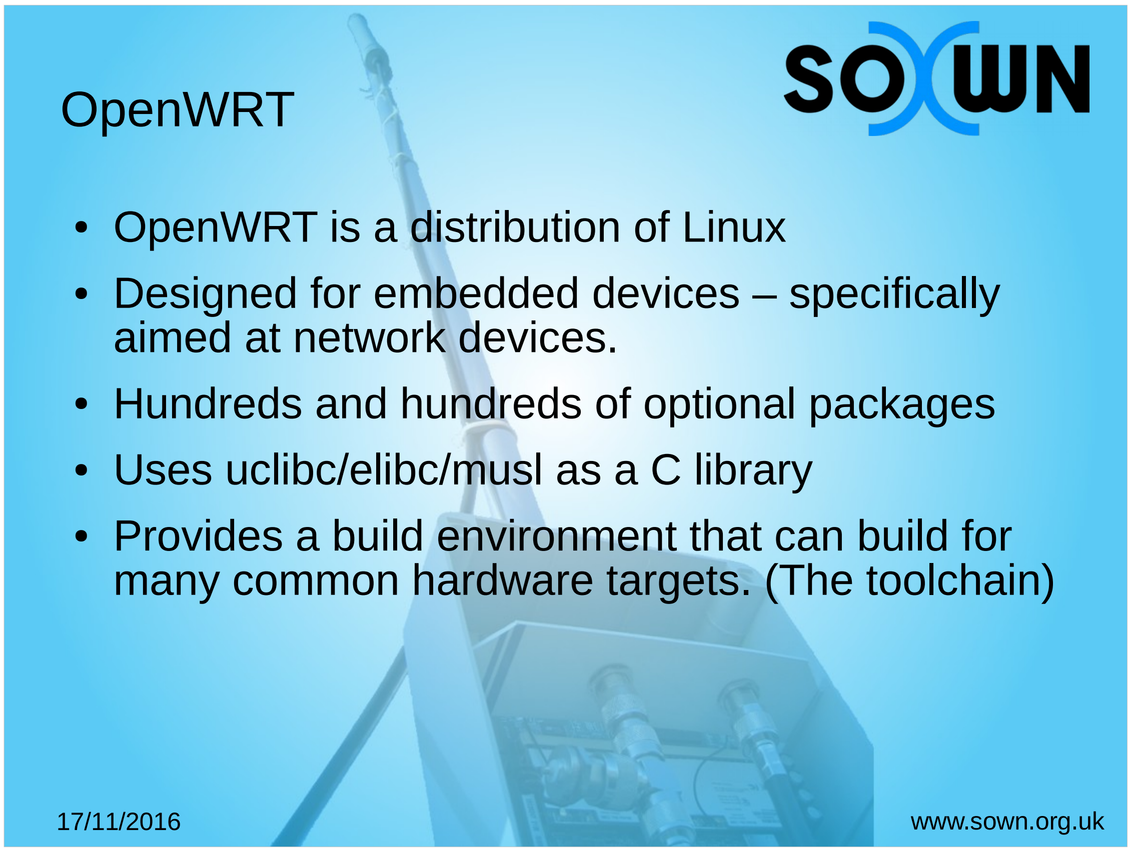
Busybox gives a bare minimum set of system utilities, such as cp, wget and module loading tools. Fundamentally it includes "init" a system binary that runs other processes.

It's designed to give enough functionality to do basic operations without doing anything more.

Embedded Linux

SOWN

- We need to be able to run programs….
  - Like DHCP servers, DNS servers, wireless authentication supplicants (wpa-supplicant)
- “libc” - The standard C library
  - Too big for many embedded linux devices.
  - 1.8MB on a standard Linux installation
  - Some devices only have 4MB of flash!
- Enter uClibc
  - Standard C Library replacement for embedded devices
  - Others exist – musl, elibc, etc.



OpenWRT

- OpenWRT is a distribution of Linux
- Designed for embedded devices – specifically aimed at network devices.
- Hundreds and hundreds of optional packages
- Uses uclibc/elibc/musl as a C library
- Provides a build environment that can build for many common hardware targets. (The toolchain)

17/11/2016

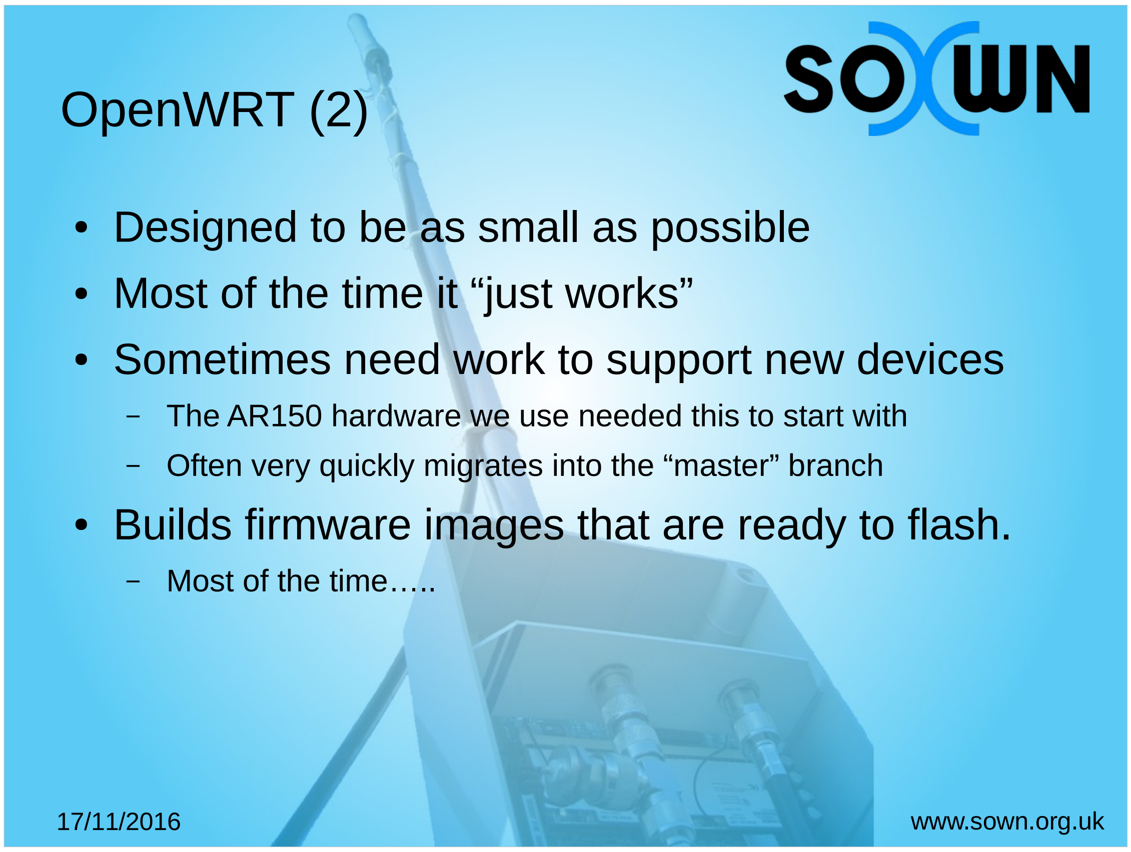
www.sown.org.uk

OpenWRT is a linux distribution built on top of busybox and a stripped down Linux image. It's aimed at network devices, which is why we're interested in it. Other distributions exist.

It includes a whole raft of additional packages that are available right the way up to full window managers and graphical environments.

It also has several choices of embedded C library. These libraries are stripped down versions of the standard C library. “Stripped down” in so much as they try and implement the largest possible feature set in the smallest possible space.

The build environment for OpenWRT is a well established environment for building firmware images for embedded systems. It is used on many network devices.



OpenWRT (2)

- Designed to be as small as possible
- Most of the time it “just works”
- Sometimes need work to support new devices
  - The AR150 hardware we use needed this to start with
  - Often very quickly migrates into the “master” branch
- Builds firmware images that are ready to flash.
  - Most of the time…..

17/11/2016 www.sown.org.uk

OpenWRT is designed to be as small as possible while delivering as much functionality as possible.

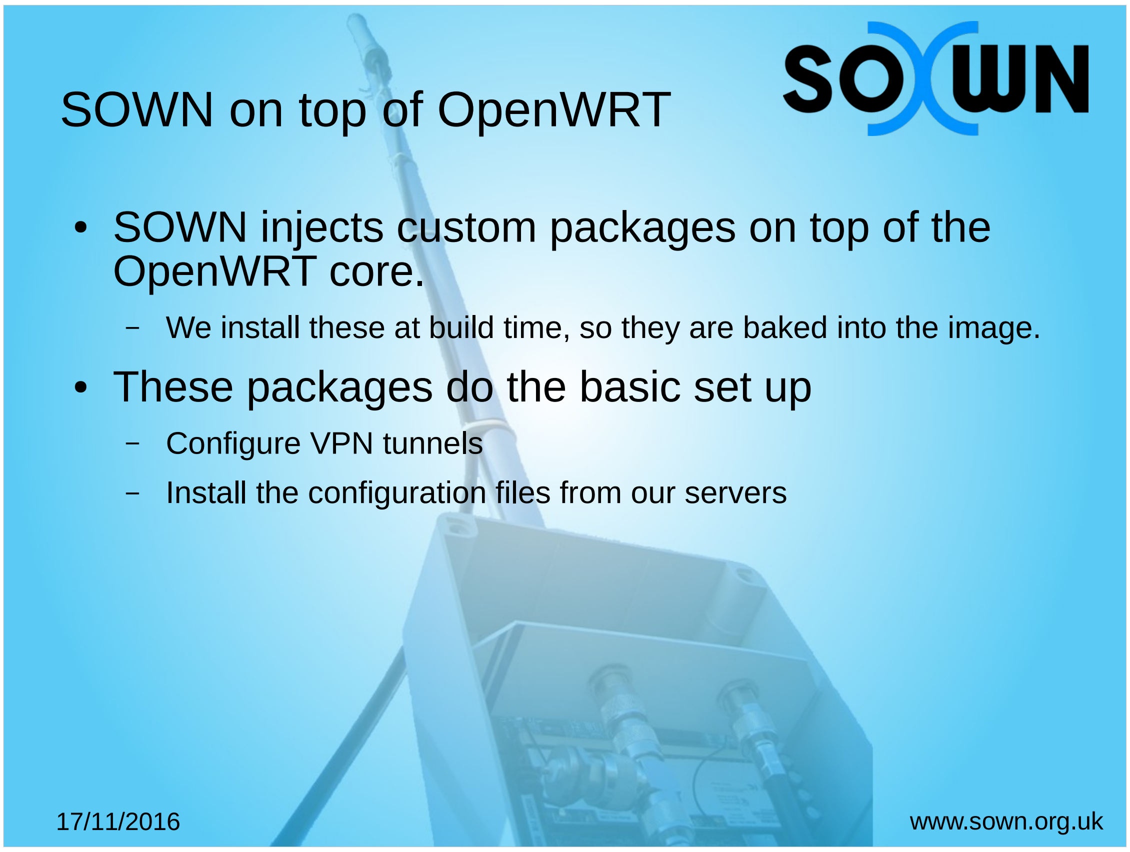
Typically fits in <8MB of flash memory.

Often the build environments just work – but it's not always the case.

Our nodes required a custom build of firmware to run on the new hardware we were working on. We had to merge extra features from patches on the OpenWRT mailing list, which then made it into master very quickly.

We're now trying to support a new hardware platform, and that needs work.

The build environment “just work” for the vast majority of situations. But they don't always. Some images require some custom tweaking to get things to work reliably. This can involve spending a long time adjusting kernel options.



SOWN on top of OpenWRT

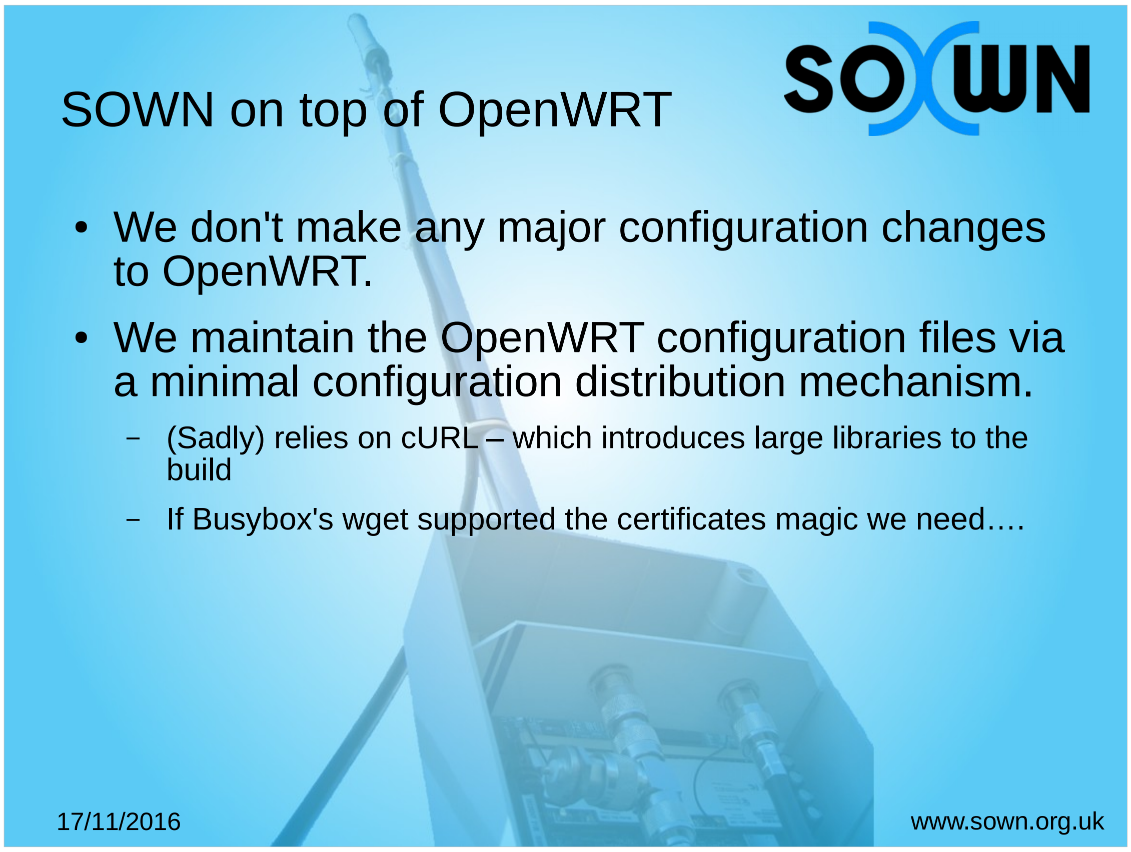
- SOWN injects custom packages on top of the OpenWRT core.
  - We install these at build time, so they are baked into the image.
- These packages do the basic set up
  - Configure VPN tunnels
  - Install the configuration files from our servers

17/11/2016 www.sown.org.uk

SOWN builds on the flexibility of OpenWRT and injects custom packages at build time on top of the OpenWRT core.

We install as little as possible in terms of configuration at build time and then install configuration after the tunnels are all configured and running.

The basic packages retrieve configuration from our servers, and then configure the remaining parts of the build.



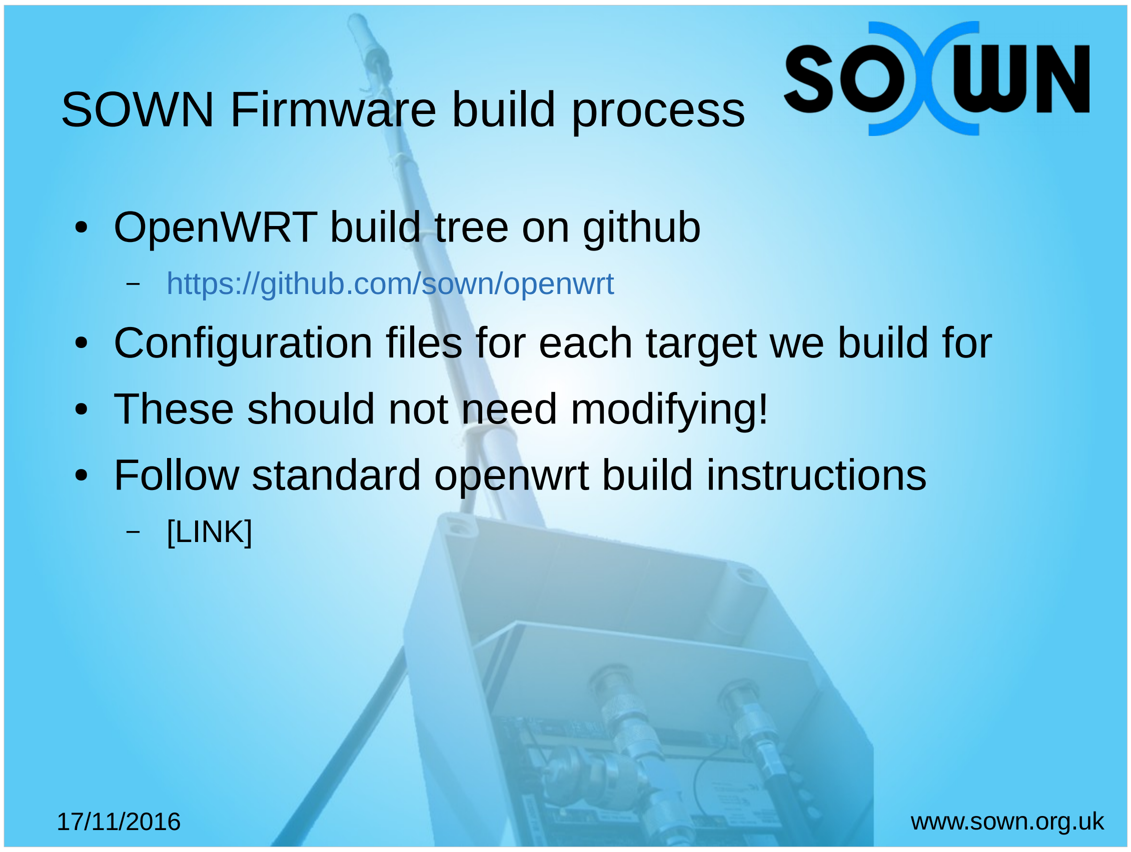
SOWN on top of OpenWRT

- We don't make any major configuration changes to OpenWRT.
- We maintain the OpenWRT configuration files via a minimal configuration distribution mechanism.
  - (Sadly) relies on cURL – which introduces large libraries to the build
  - If Busybox's wget supported the certificates magic we need….

17/11/2016 www.sown.org.uk

We don't do any major configuration changes to OpenWRT – we do overwrite some of their configuration files using our own configuration management tool. This configuration tool is actually based on cURL due to the way the client-side certificates work for authentication which brings in a large file dependency (libcurl). It's not practical to run Puppet/Chef/configuration-manager-du-jour on a node.

In overwriting some of these files (like /etc/config/network) we actually break a few things. We're still working on a solution for the new MT300 based nodes, because they don't use a config like anything we've used before.



SOWN Firmware build process

SOWN

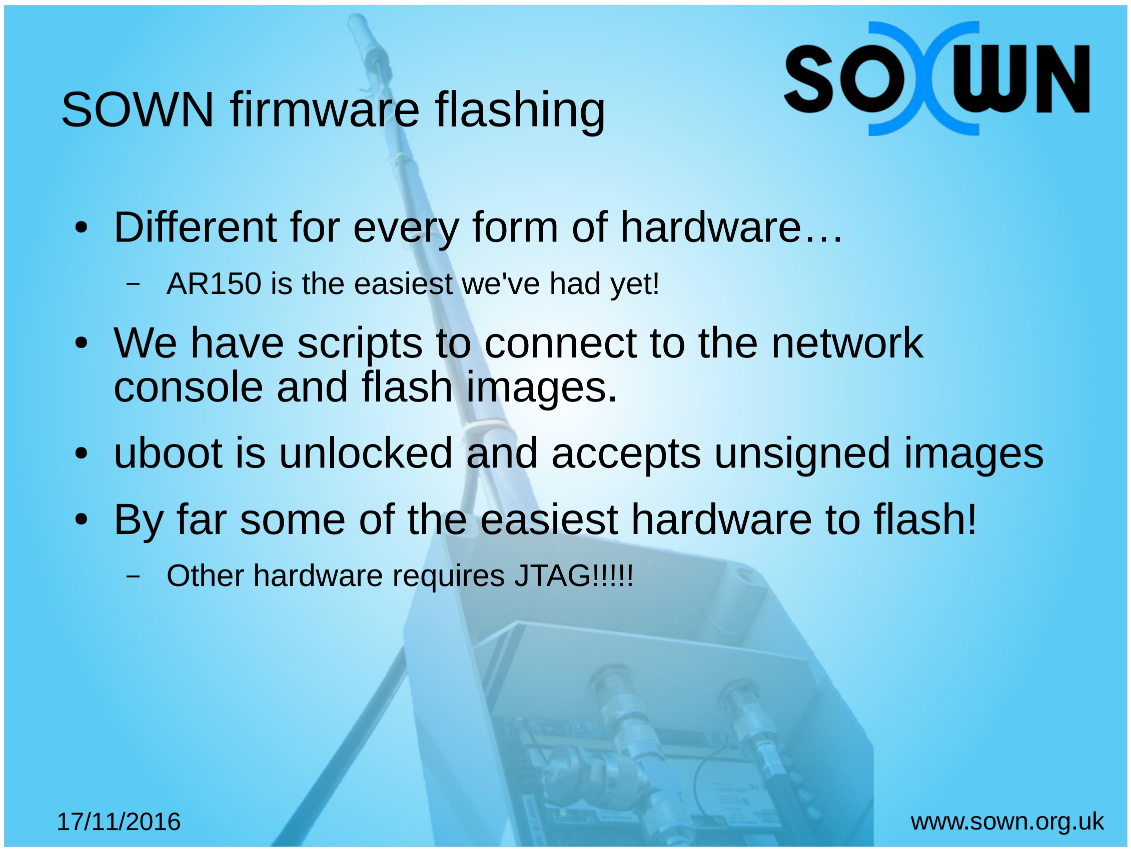
- OpenWRT build tree on github
  - https://github.com/sown/openwrt
- Configuration files for each target we build for
- These should not need modifying!
- Follow standard openwrt build instructions
  - [LINK]

17/11/2016 www.sown.org.uk

The SOWN firmware for the latest nodes follows standard OpenWRT build process. There's actually nothing special, bar a couple of packages being selected. We put the configuration files for the build tree in the repository, so you can copy them to .config as per standard OpenWRT and just build.

The main essence of the OpenWRT build sequence is simply make menuconfig/oldconfig then simply make.

The build process takes about 3-4 hours from a “cold” environment, ~1 hour from a “warm” environment, and minutes in a “hot” environment.



SOWN firmware flashing

- Different for every form of hardware…
  - AR150 is the easiest we've had yet!
- We have scripts to connect to the network console and flash images.
- uboot is unlocked and accepts unsigned images
- By far some of the easiest hardware to flash!
  - Other hardware requires JTAG!!!!!

17/11/2016 www.sown.org.uk

Flashing the AR150 nodes is one of the easiest tasks we've had yet. Apart from the console, which relies on some UDP port 6666 hackery, with the right sequence of button presses we get an unlocked uboot instance that we can flash images over a web interface!!!!

Our older nodes sometimes required JTAG, but only in rare cases. All SOWN hardware has been network-flashable for many years.

SOWN Firmware Flashing

- Demo