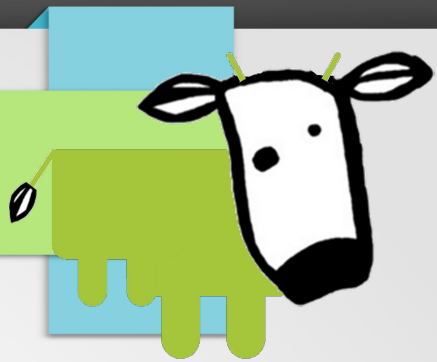
Gentoo-Bionic

We can Rebuild him. Better. Stronger. Faster.



Christopher Friedt

BlackBerry™

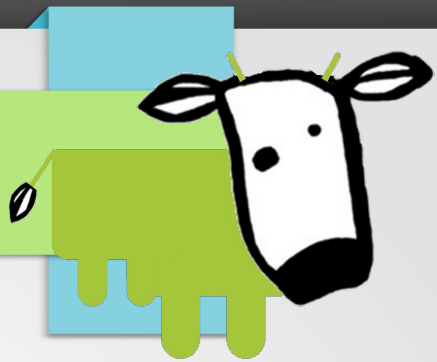
Embedded Linux Conference, 2013

San Francisco, CA

chrisfriedt@gmail.com

Gentoo-Bionic

We can Rebuild him. Better. Stronger. Faster.



http://code.google.com/p/gentoo-bionic

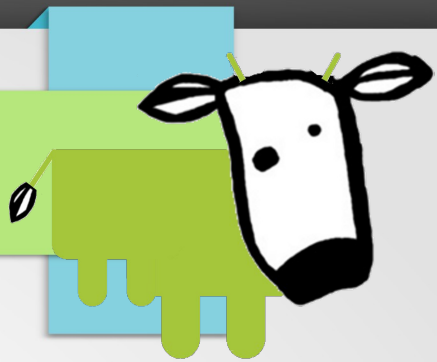
http://gentoo-bionic.blogspot.com

http://gitorious.org/gentoo-bionic/gentoo-bionic

http://www.facebook.com/GentooBionic

https://plus.google.com/113359270067626599390

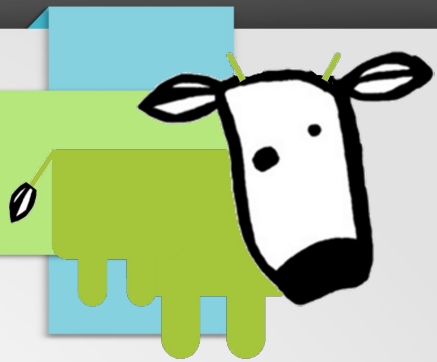
Why? Where? When? What?



RATIONALE

- **BACKGROUND**
- **LICENSING**
- **MAINTAINABILITY / COMPLEXITY**
- **MINIMAL BUT EXTENSIBLE**
- **SCOPE**
- **THINK INSIDE THE BOX**
- **SHORT TERM GOALS**

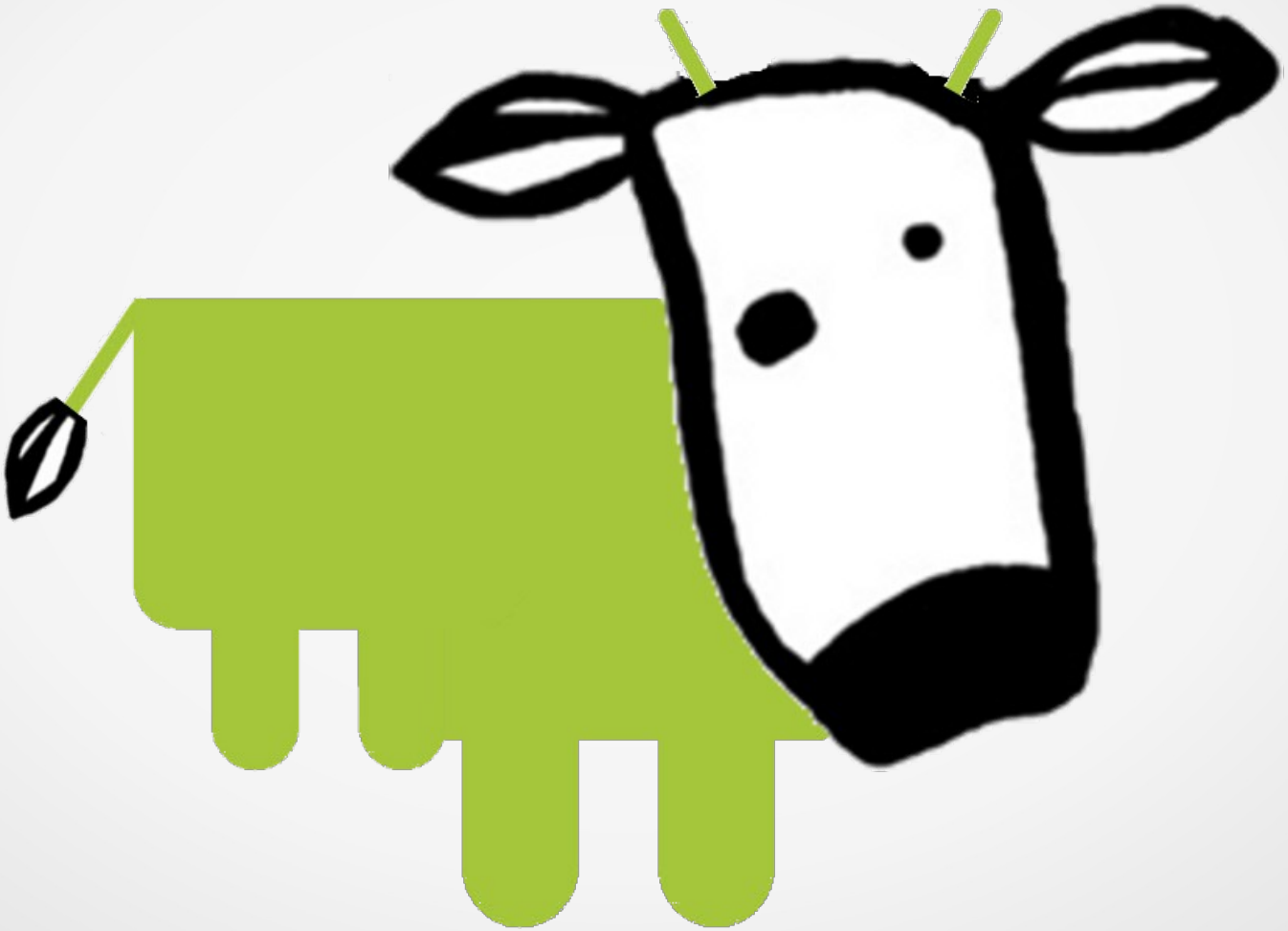
How?



BOOTSTRAP

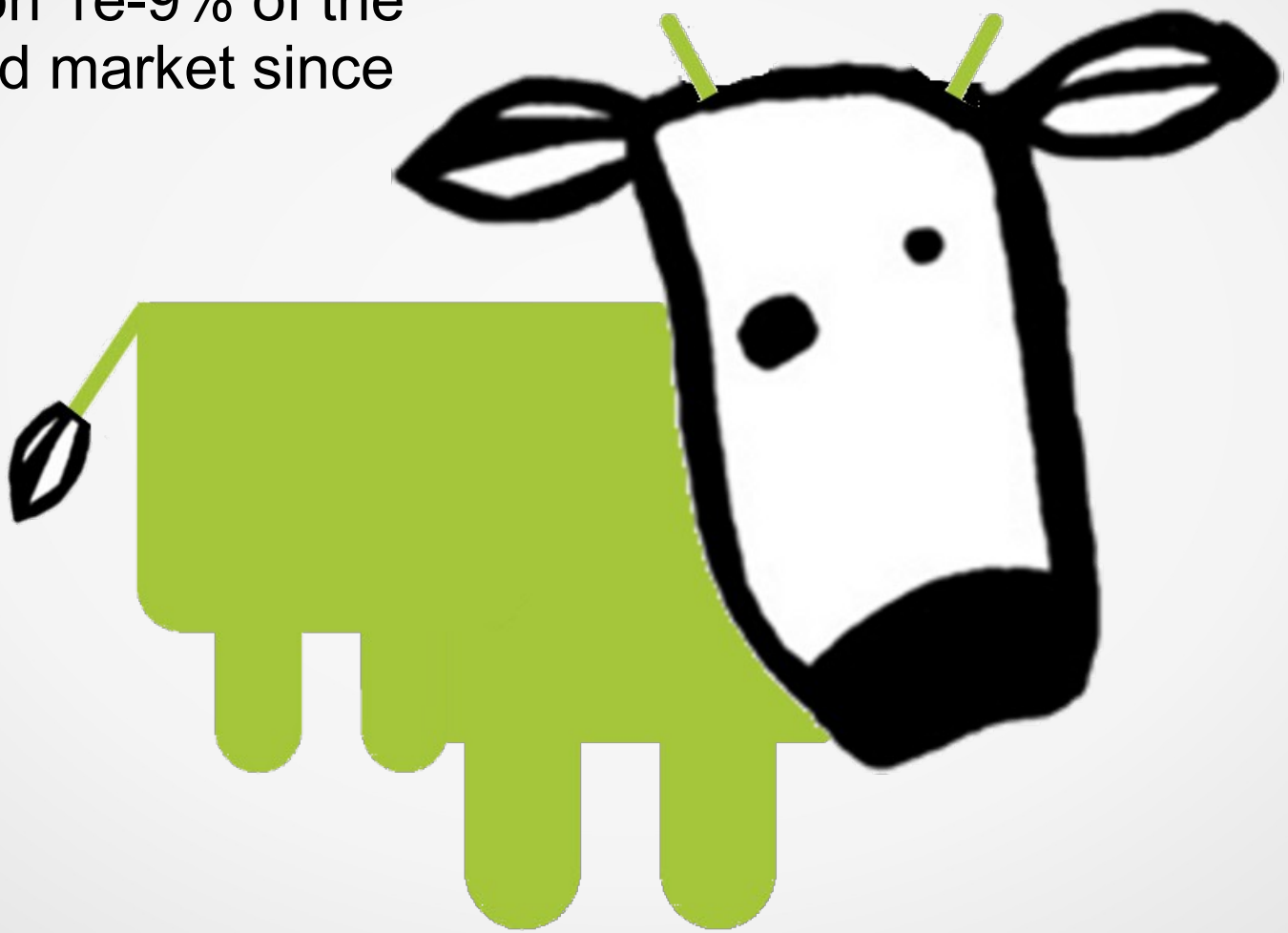
- **LINARO GCC PATCH**
- **REMOVE ANDROID CRUFT**
- **ADD A NEW ELIBC IN GENTOO**
- **./CONFIGURE; MAKE SENSE**
- **CROSSDEV WAY OR THE HIGHWAY**
- **EMERGE WORLD**
- **WHAT NEXT?**

Bionic Larry...

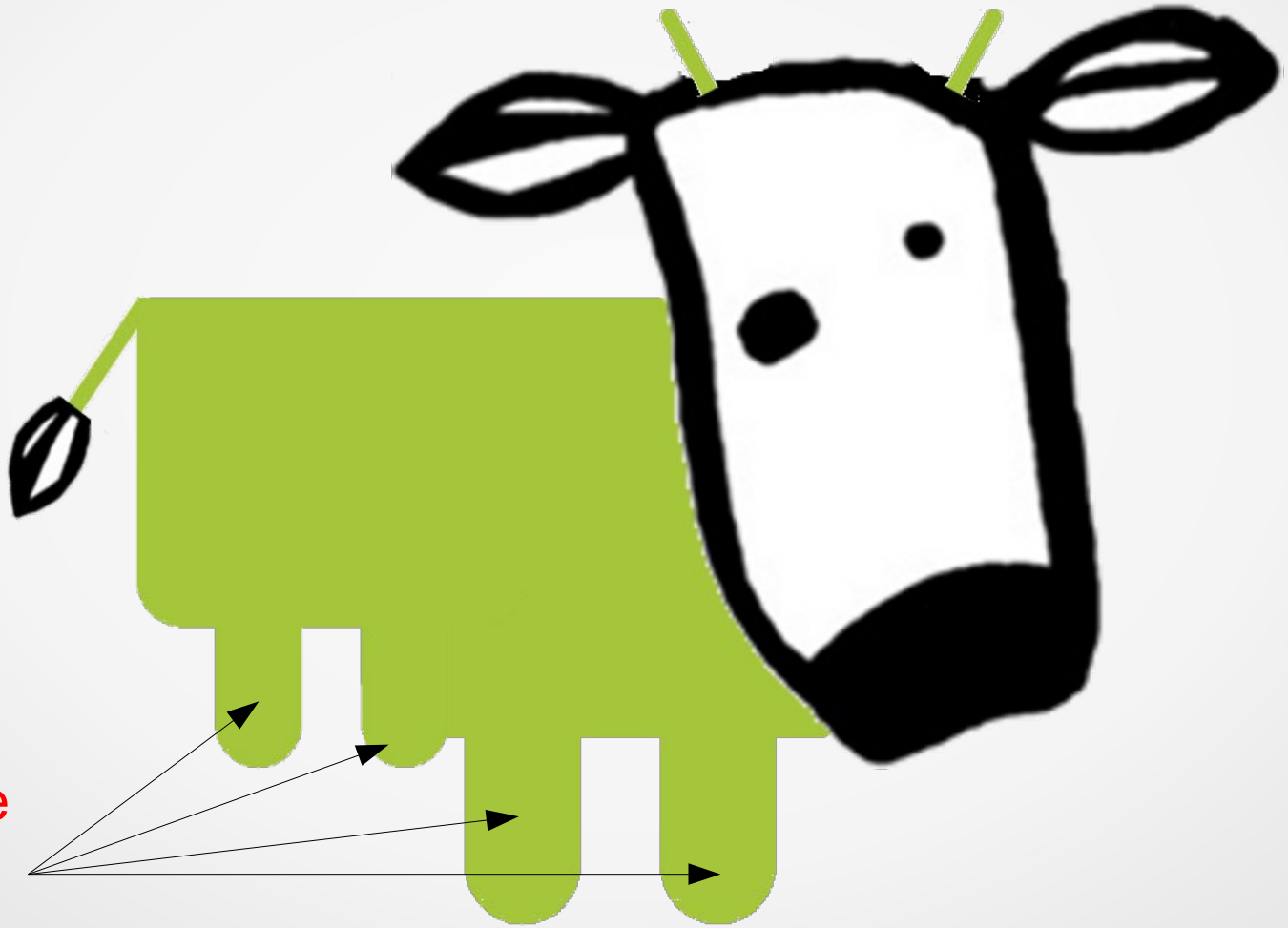


Bionic Larry...

Grazing on 1e-9% of the embedded market since 2010!

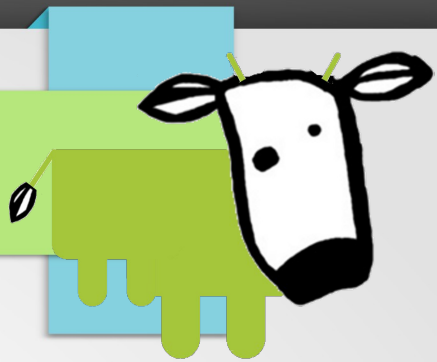


Bionic Larry... do **not** try an milk him



These are
<u>not</u>
udders

Why? Where? When? What?

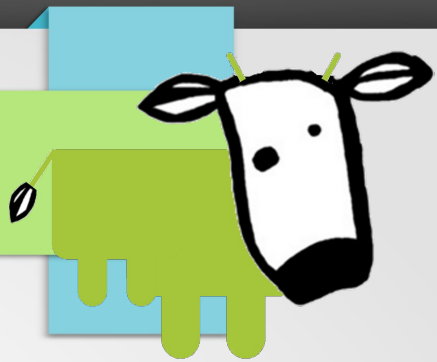


RATIONALE - HISTORY

- Originally, I wanted to do something to help Google while they were in the middle of the Oracle / Java legal dispute
- I actually wrote Google with my ideas...
- they interviewed me for a couple of positions...
- but otherwise didn't care :-(

  ... talk to me after the presentation for some tidbits

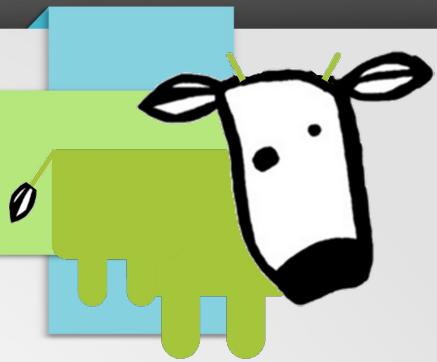
Why? Where? When? What?



RATIONALE - LICENSING

- **In spite** of the GPLv3 exclusion of “system libraries” from the linking clauses, companies are still terrified to incorporate GPL software into their embedded products
  - afraid of being forced to open their codebase
    - poorly written / insecure code vetted by 3rd parties
    - leaking intellectual property

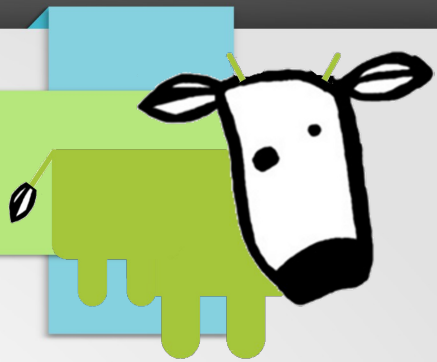
Why? Where? When? What?



RATIONALE - LICENSING

- How does a company retain IP in a predominantly open-source / GPL universe?
  - static / shared linking constitute derived works in many opinions
  - most shared library code is not explicitly LGPL
  - zero to practically zero static libraries are LGPL

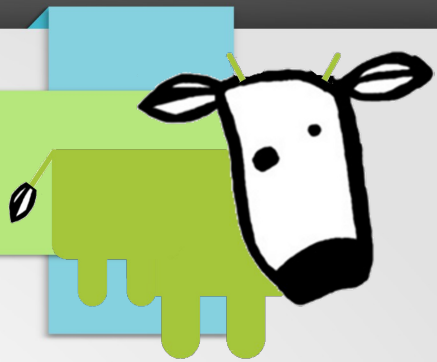
Why? Where? When? What?



RATIONALE - LICENSING

- **lease** embedded devices to customers and charge for usage / data
  - no change of ownership / no source sharing req
  - limited revenue model
  - questionable circumvention of software license

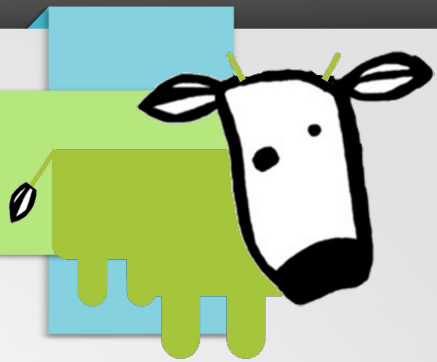
Why? Where? When? What?



RATIONALE - LICENSING

- Take chances with FLOSS licenses or reinvent the wheel
  - potentially a lot of extra implementation work
  - limited domain expertise
  - possible license that could change over time
  - Could require fork and back-porting new patches
  - delaying the inevitable?

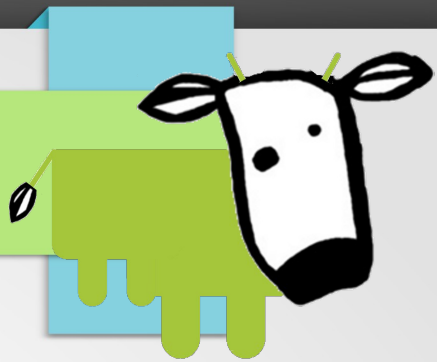
Why? Where? When? What?



RATIONALE - LICENSING

- base design around **newlib**
  - BSD licenced libc
  - retain userspace IP, linking to newlib
  - distribute source for the Linux kernel
  - optimized? ... not really

Why? Where? When? What?

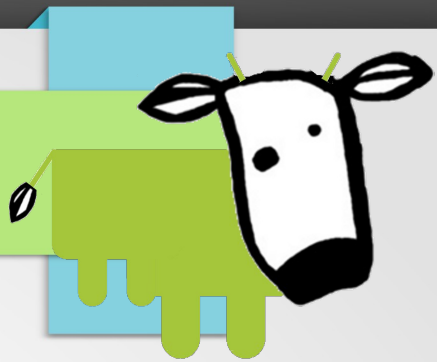


RATIONALE - LICENSING

- I gathered this was fairly common industry opinion after consulting for various companies in industrial radio, embedded imaging, shipping / receiving, automated asset management, etc

BUT...

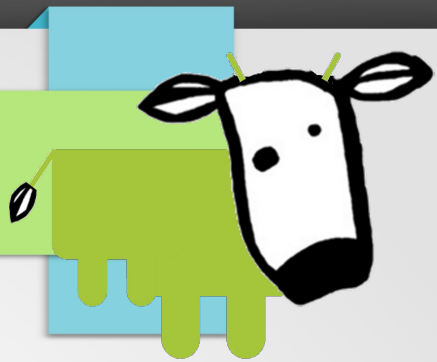
Why? Where? When? What?



RATIONALE - LICENSING

- the Linux kernel is awesome
  - the GPLv2 license is working well for it
  - no need to “fix” what isn't broken

Why? Where? When? What?

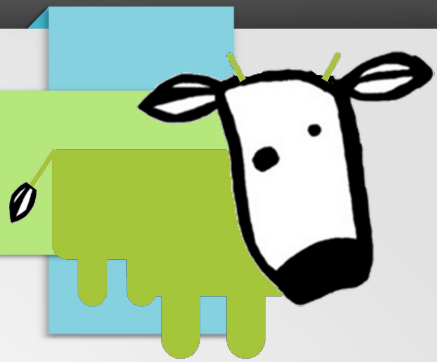


RATIONALE - LICENSING

- There ***might*** even be some kernel-envy in the rest of the embedded world!
- Linux supports more arch's, chips, platforms, boards than any other OS kernel on the planet!
- It's easy to get ported, and Linux is what clients want driving their embedded platforms

LUCKILY...

Why? Where? When? What?

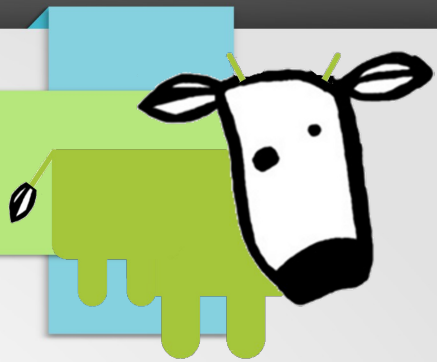


RATIONALE - LICENSING

- The Linux kernelspace / userspace interface is **BINARY**
- This is what allows arbitrarily licensed userspace software to run on top of the GPLv2 Linux kernel
- Kernel interaction is not “linking” (at runtime or compile-time), it's setting up arguments on the stack and jumping!
- Corollary: libc builds own syscalls using **NUMBERS**

SOOOO...

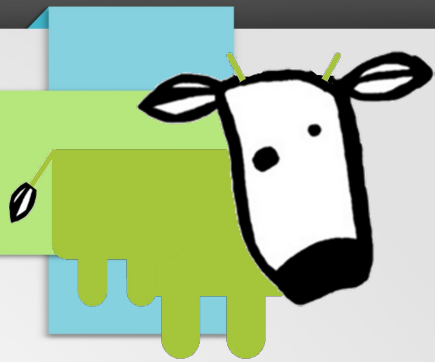
Why? Where? When? What?



RATIONALE - LICENSING

- Use a BSD-licensed C library and other system libraries on top of the Linux kernel

Why? Where? When? What?

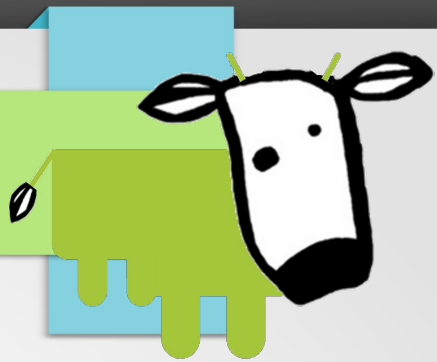


RATIONALE - LICENSING

- Use a BSD-licensed C library and other system libraries on top of the Linux kernel

  » **USE BIONIC**

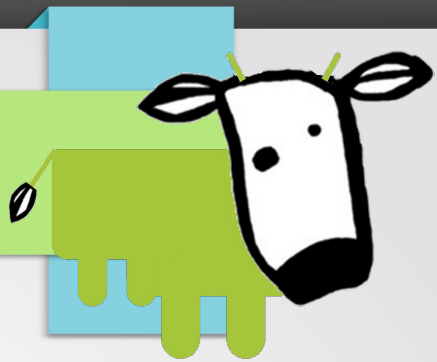
Why? Where? When? What?



RATIONALE – MAINTAINABILITY / COMPLEXITY

- already hacked the same Bionic C runtime for a couple of different clients a couple of different times
  - Just Worked™
  - was very slim, but optimized where it counts
  - it didn't take a lot of effort
  - (to me the effort part was important!)

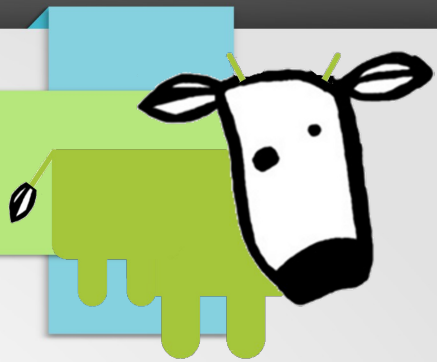
Why? Where? When? What?



RATIONALE – MAINTAINABILITY / COMPLEXITY

- The Bionic C library is - fairly well organized..
  - is documented / commented where merited
  - is kept simple (intentionally!)
  - has no cryptic autotools or (many) scripts for building
  - is easily extensible
  - compiles really quickly!

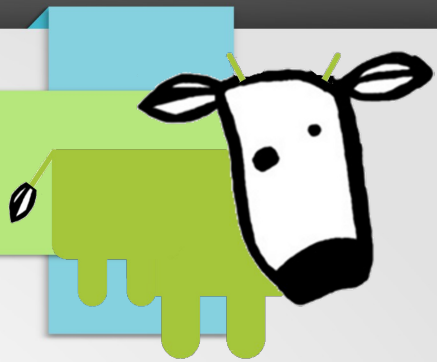
Why? Where? When? What?



RATIONALE – MAINTAINABILITY / COMPLEXITY

- Why reinvent the wheel every time?
  - Bionic is BSD licensed, and there was no client-specific IP in it, that I had added.
  - Allow others to benefit from it's usage
  - And ***contribute back***
- No need to reinvent distro's, package managers, etc

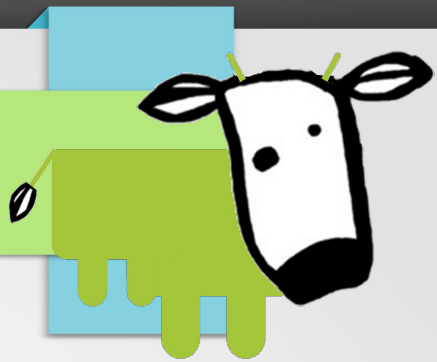
Why? Where? When? What?



RATIONALE – MAINTAINABILITY / COMPLEXITY

- I was familiar with Gentoo .ebuild syntax
- Same code could easily be built & packaged for
  - Ångström / OpenEmbedded / OpenWRT (.ipk)
  - Debian / Ubuntu (.deb)
  - Redhat (.rpm)

Why? Where? When? What?

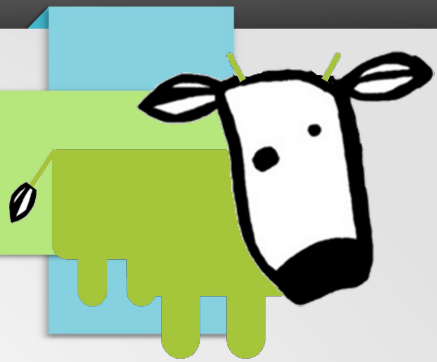


RATIONALE - MINIMAL BUT EXTENSIBLE

- Bionic is small (e.g. for libc.so)

C Library	Size (bytes)
glibc	1209672
uClibc	327023
bionic	290912

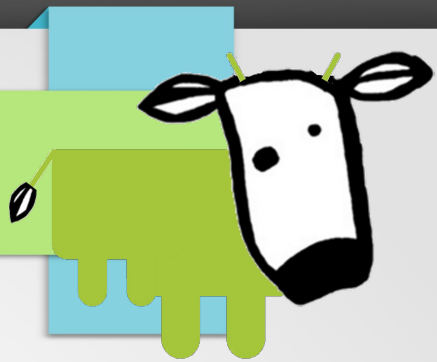
Why? Where? When? What?



RATIONALE - MINIMAL BUT EXTENSIBLE

Adding syscalls?

Why? Where? When? What?

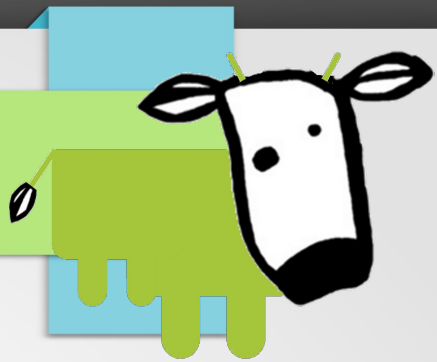


RATIONALE - MINIMAL BUT EXTENSIBLE

- Adding syscalls? - **SUPER EASY!** Just add them to libc/SYSCALLS.TXT!
- an assembly wrapper is created automagically by

  libc/tools/gensyscalls.py
- e.g. int pivot_root(const char *, const char *) 117,118,117
- [return type] [syscall name]([parameters]) [arm,x86,mips]

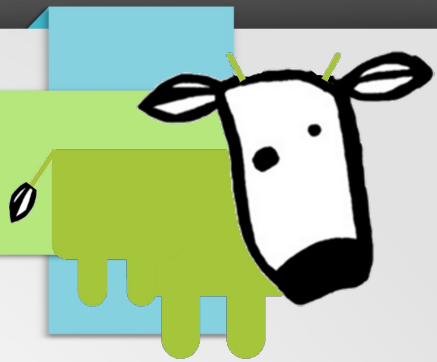
Why? Where? When? What?



RATIONALE - MINIMAL BUT EXTENSIBLE

Added syscalls:

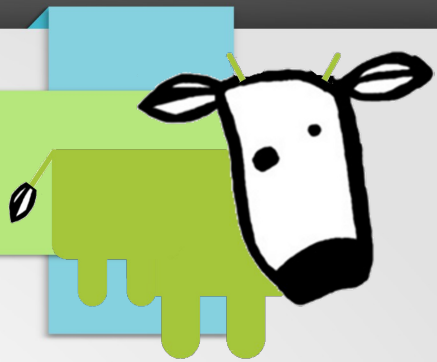
Why? Where? When? What?



RATIONALE - MINIMAL BUT EXTENSIBLE

- Missing an ioctl / syscall / struct declaration?

Why? Where? When? What?

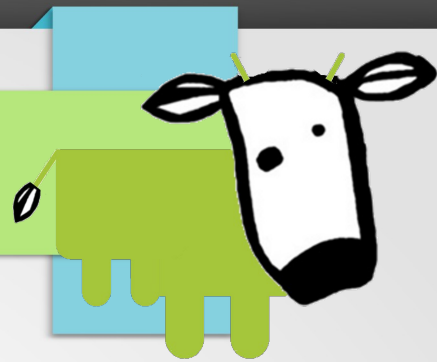


RATIONALE - MINIMAL BUT EXTENSIBLE

- Missing an ioctl / syscall / struct declaration?
- **ALSO SUPER EASY!**
- preprocess the raw header information with

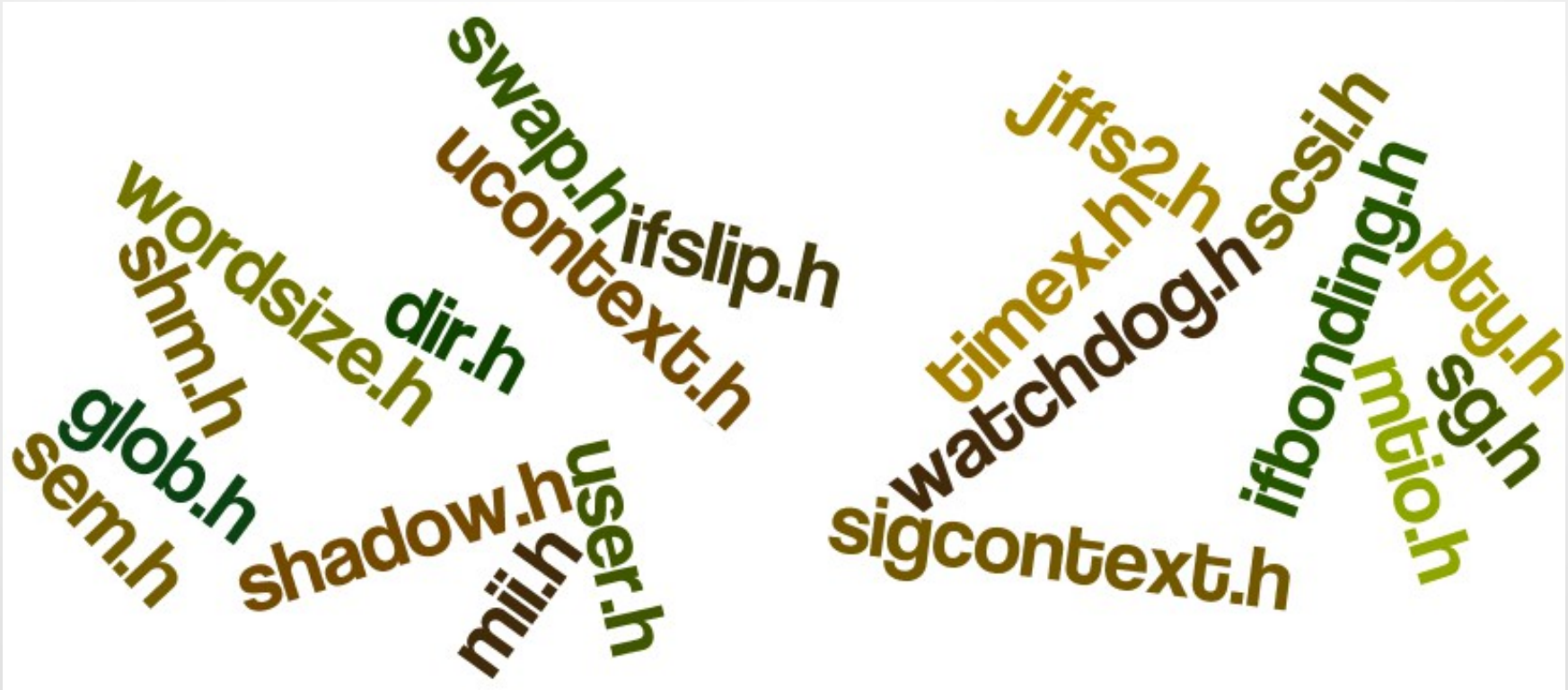
  libc/tools/clean_header.py
- just remember: no inline functions, no macros, no comments!

Why? Where? When? What?

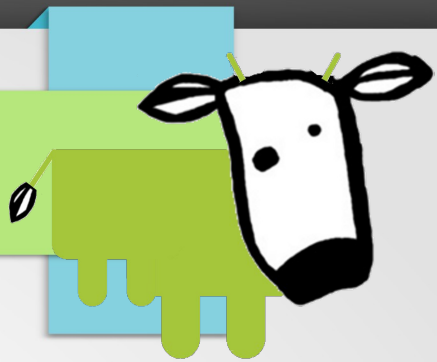


RATIONALE - MINIMAL BUT EXTENSIBLE

Added headers



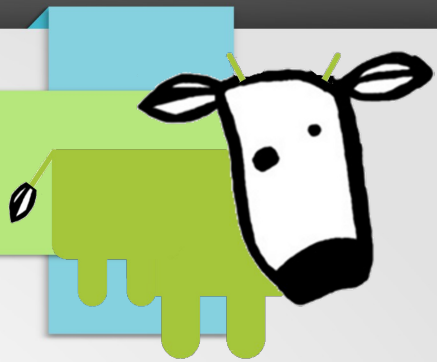
Why? Where? When? What?



RATIONALE - MINIMAL BUT EXTENSIBLE

- Adding libc functions?

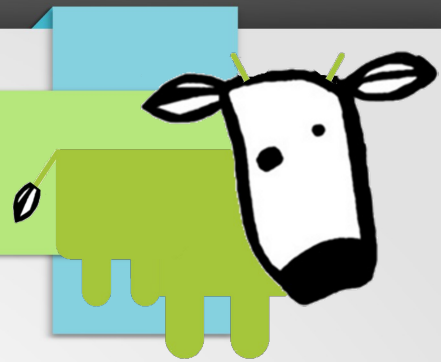
Why? Where? When? What?



RATIONALE - MINIMAL BUT EXTENSIBLE

- Adding libc functions? - **SUPER EASY!**
  - create a test rig outside of libc
  - compile your test rig
  - test your libc function
  - when testing done, add to ${FILESDIR} as a patch!
  - files/${PV}/NNNN-yay-i-implemented-a-libcfunc.patch

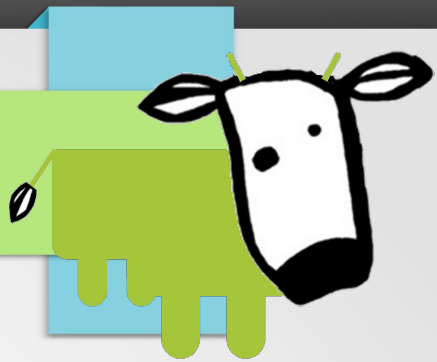
Why? Where? When? What?



RATIONALE - MINIMAL BUT EXTENSIBLE

Added libc functions

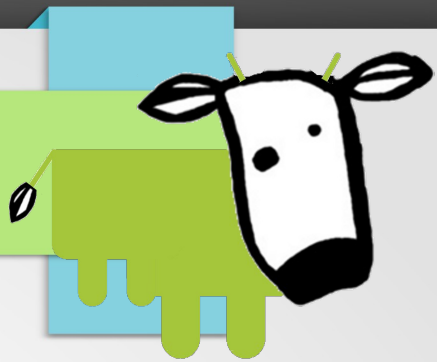
Why? Where? When? What?



RATIONALE - SCOPE

- Is Gentoo-Bionic Gentoo-specific?
  - **NO!**
  - Gentoo was used as the initial vehicle for compiling the Bionic C library and toolchain
  - Bionic / toolchain could be built for any Linux distro
  - build system changes for said distros would be minimal, once autoconf integration is done

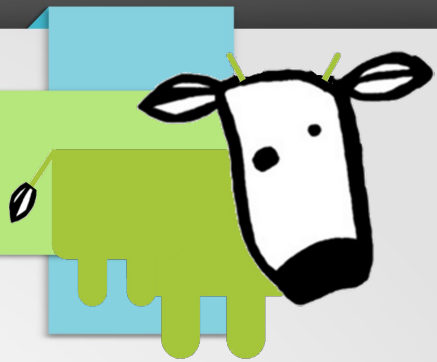
Why? Where? When? What?



RATIONALE - SCOPE

- Also not limited to usage on existing distributions
- Rather, it should serve as a starting point, inspiring new and different distributions

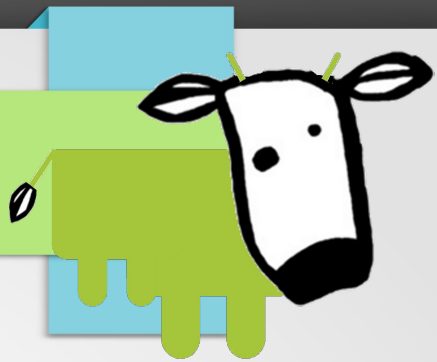
Why? Where? When? What?



RATIONALE - SCOPE

- Gentoo just has a *really* great cross-compiler infrastructure and build system (Portage)
  - chost=armv7a-neon-linux-bioniceabi
  - chost=i686-pc-linux-bionic
  - crossdev --target ${chost}
  - ${chost}-emerge bash
    - all (runtime / build) dependencies included

Why? Where? When? What?

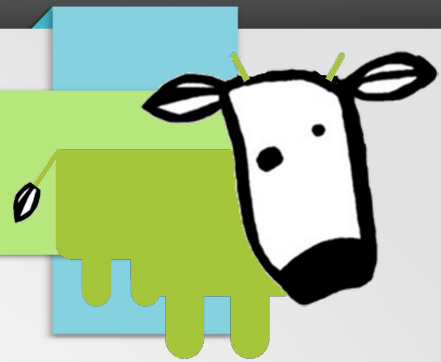


RATIONALE - THINK INSIDE THE BOX

- Recently switched to OS X from Linux for my workstation

  (queue Booing from crowd)
- The UI (partially) did it for me, but I also liked **not** feeling the need to **fix** things!
- I liked the minimalistic “feel” to the libc, and how things basically always Just Worked™

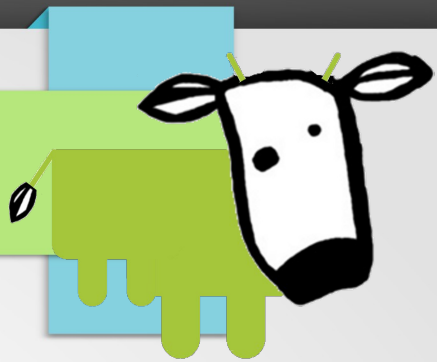
Why? Where? When? What?



RATIONALE - THINK INSIDE THE BOX

- The graphics stack intrigued me
- Liked the idea of using some (certain) proprietary software packages
- Why can't we have one (or many) “proprietary” Linux variants?
- ... but Mac OS X kind of sucks under the hood
- supported & default FS (global lock?)

Why? Where? When? What?

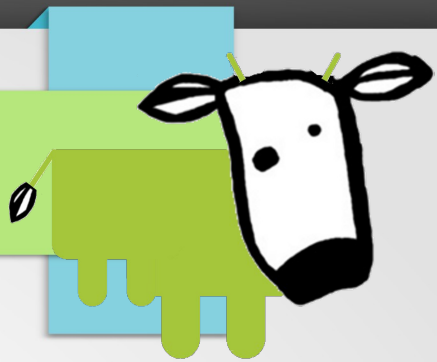


RATIONALE - (SHORT TERM) GOALS

- Layman(8) overlay
- Upstream (basic) inclusion in Portage
- Downloadable (tiny) VM images
  - qemu, VMWare, VirtualBox
  - arm (qemu), x86 for VMWare / VirtualBox

Why? Where? When? What?

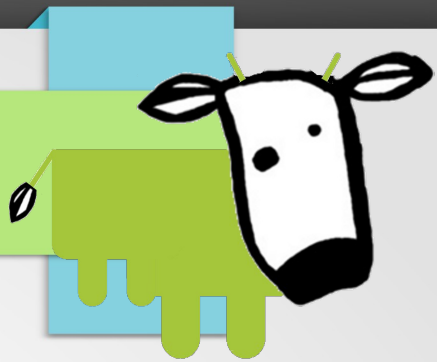
How?



BOOTSTRAP - LINARO GCC PATCH

- Alexandre Sack's gcc-4.6 patch
  - default linker specs with -mandroid
  - crt*.o for linking
  - /system/bin/linker
  - toolchain was no longer tied to android.com

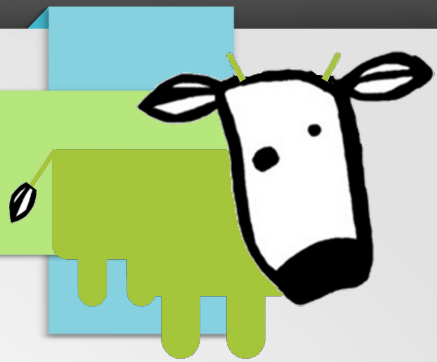
How?



BOOTSTRAP - LINARO GCC PATCH

- why use the /system prefix?
- why use /system/bin/linker?
- not just arm!
- preserve that for -mandroid
- remove the Android cruft for -mbionic
- try to behave like a normal toolchain!

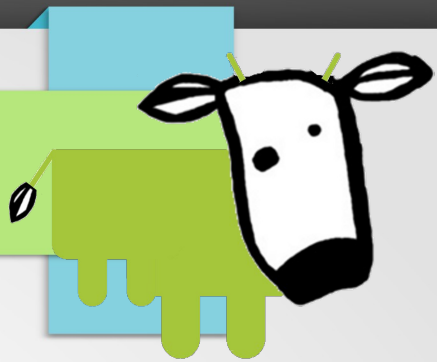
How?



BOOTSTRAP - REMOVE ANDROID CRUFT

- for Bionic to behave like a normal libc
  - /etc/passwd, /etc/group, /etc/resolv.conf, ...
- Android went through system properties and hard-coded UID's and GID's

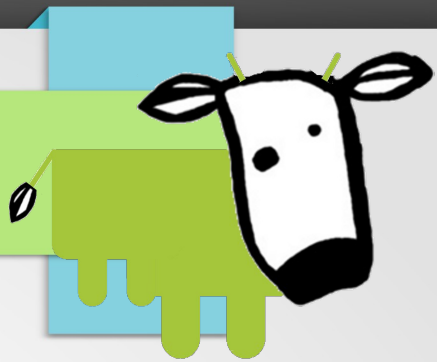
How?



BOOTSTRAP - ADD A NEW LIBC

- Portage changes:
  - portage/profiles/desc/elibc.desc
  - portage/profiles/embedded/bionic/*

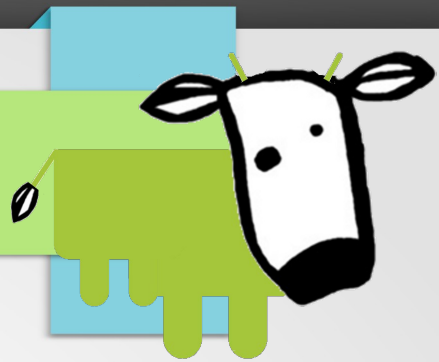
How?



BOOTSTRAP - ./CONFIGURE; MAKE SENSE

- gnuconfig changes:
  - config.sub
  - config.guess

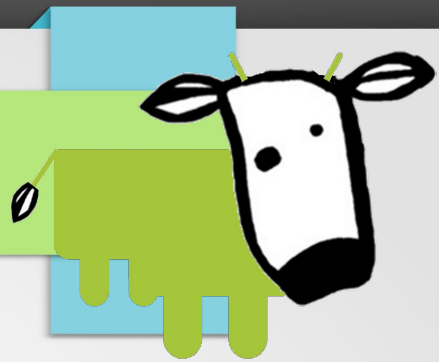
How?



BOOTSTRAP - CROSSDEV WAY OR THE HIGHWAY

- crossdev changes:
  - LPKG=bionic; KPKG=bionic-kernel-headers
  - include/site/*bionic* (basically a copy of *uclibc*)
  - above files necessary for autoconf functionality
- crosscompile_opts_headers-only
- nocxx / cxx

How?



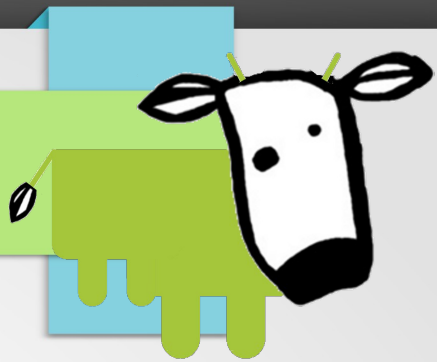
BOOTSTRAP - EMERGE WORLD!

- Although there is/are a/many rigid specifications of what must be in a libc.. thanks to GNU, there is a monotonically increasing list of “expected” features as well.
- emerge busybox, emerge bash, emerge jamvm...
- Keep testing, finding, and reporting bugs, adding features as required

How?

- **BOOTSTRAP - EMERGE WORLD!**
- **Compiled packages (so far)**

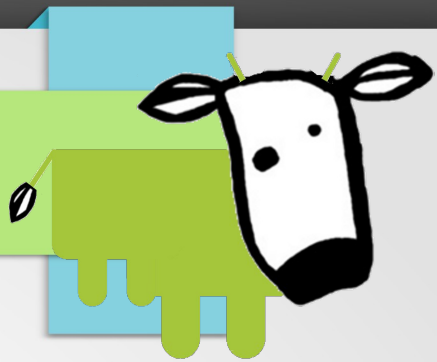
How?



BOOTSTRAP - WHAT NEXT?

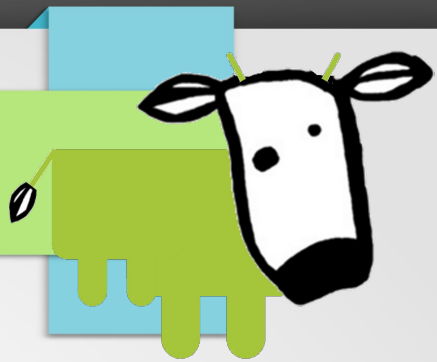
- HELP WANTED!
  - pthread_cancel
  - glibc-like ld.so behaviour
  - optional locale
  - self-hosting gcc (clang?)
  - [func]_r (thread-safe versions of functions)
  - more crypt algos

How?



BOOTSTRAP - WHAT NEXT?

- Beyond bootstrap
  - Talk to me after the presentation
  - I could go on... seriously!

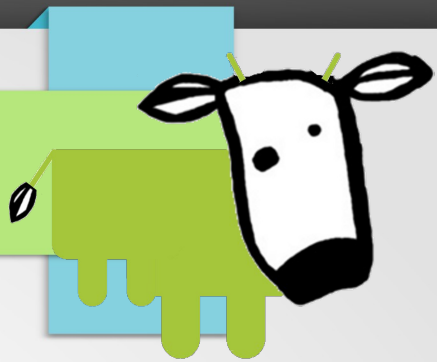


mäk – A SHORTER / FASTER MAK

- separately installable variant of the Android build system
  - non-C language support to be included via extension
- like Automake's .am files, mäk's .mk files are declarative
- export MAK_ROOT=/usr/share/mak
- ./configure; make -jN; make -jN install
- **non-recursive** replacement for Automake = **FAST**

Gentoo-Bionic

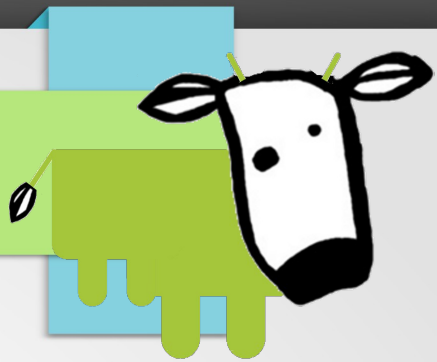
We can Rebuild him. Better. Stronger. Faster.



DEMOS

Gentoo-Bionic

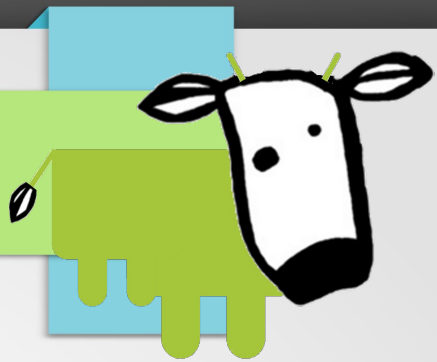
We can Rebuild him. Better. Stronger. Faster.



Q&A

Gentoo-Bionic

We can Rebuild him. Better. Stronger. Faster.



THANKS!