# GNU Mes – Reduced Binary Seed bootstrap

janneke@gnu.org

FOSDEM'19

2019-02-02

# Outline

# Reduced Binary Seed bootstrap: GNU Mes

## GNU Mes

- A Scheme interpreter written in ~5,000LOC of simple C.
- A C compiler written in Scheme.
- Built on eval/apply, the Maxwell Equations of Software.

# What is a Compiler?

## A compiler takes source code and produces executable object code

```
$ gcc hello.c -o hello
$ ./hello => "Hello, Mes!"
$ gcc hello.c -S -o hello.s
```

## hello.c

```
void
main ()
{
  puts ("Hello, Mes!");
}
```

## hello.s

```
main: push   %ebp
      mov    %esp,%ebp
      push   _string_0
      call   puts
      add    $4,%esp
      leave
      ret
_string_0: .string "Hello, Mes!"
```

# What is a Compiler?

## A compiler takes source code and produces executable object code

```
$ gcc -c hello.s -o hello.o
$ gcc hello.o -o hello
$ ./hello => "Hello, Mes!"
```

## hello.o object code

```
5589e5689b020001e8dc00000083c404c9c3
```

## hello executable code

```
7f454c460101...5589e5689b020001e8dc00000083c404c9c3
```

# What is a Binary Seed?

1. A binary (program) that was not build from source.
2. A binary (program) that was injected from a previous generation.
   - Think: Binutils, GCC, Glibc, Go, Haskel, Java, Perl, Rust, ...

# What is a Bootstrap Seed?

## In Guix 0.16 ~250MB

- "bootstrap-binaries": bash, binutils, bzip2, coreutils, gawk, gcc, glibc, grep, gzip, patch, sed, tar, and xz.


## In Debian ~ 450MB

- "debootstrap" + "build-essential": adduser, apt, base-files, base-passwd, bash, binutils, bsdutils, bzip2, coreutils, cpp, cpp-6, dash, debconf, debian-archive-keyring, debianutils, diffutils, dpkg, dpkg-dev, e2fslibs, e2fsprogs, findutils, g++, g++-6, gcc, gcc-6, gcc-6-base, gpgv, grep, gzip...

# What is a Bootstrap?

## Impossible task: pull yourself up on your boot straps

## Software: to create your first: kernel, shell, C compiler, ...


source + ?? =

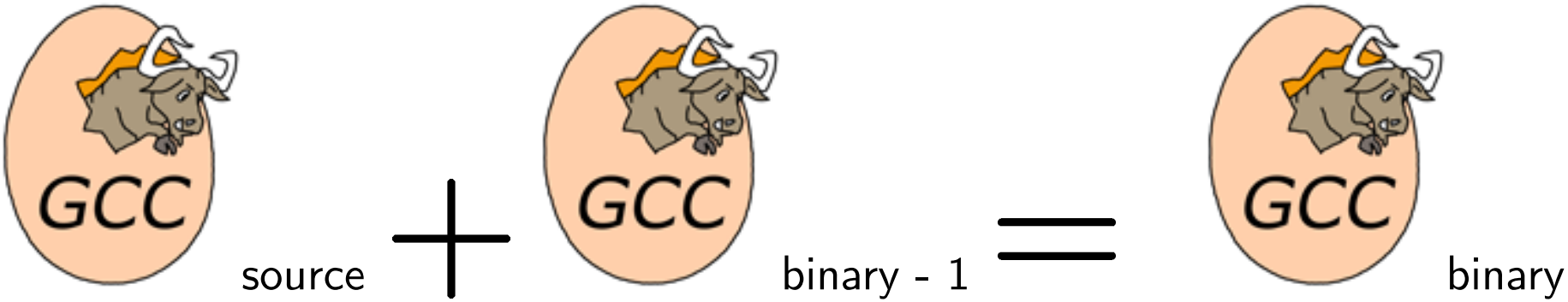

binary

# How to Bootstrap: An Old Recipe...

*Recipe for yoghurt: Add yoghurt to milk – Anonymous*

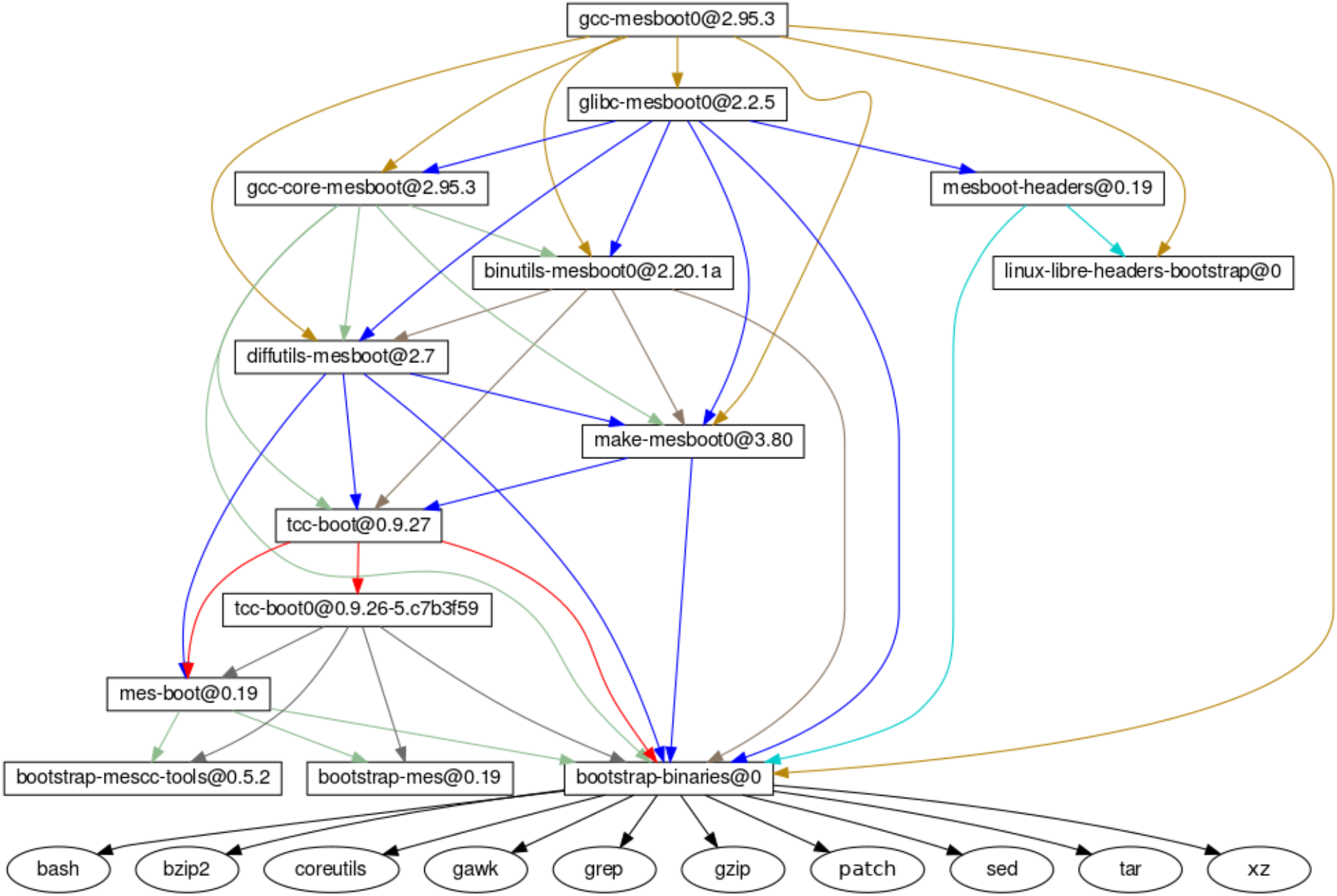
# How to Bootstrap: Create your first GCC

Traditional recipe: like yoghurt




. . . and done!

# Reduced Binary Seed: Remove Binutils, GCC, and GLIBC



# GNU Mes: Reduced Binary Seed bootstrap

## GNU Mes v0.19 (dec 2018)

- mes.c: small Scheme interpreter written in a simple C subset
  - 5000LOC
  - mostly Guile-compatible
- mescc.scm: A C compiler written in mes-compatible Guile Scheme
  - Nyacc C99 parser
- Mes C Library
  - libc.c: small C library for mes.c (25 functions, 1000LOC)
  - libc+tcc.c (80 functions, 3000LOC)
  - libc+gnu.c: bootstrap support libraries (160 functions, 6000LOC)

## Reduced Binary Seed bootstrap (unreleased: @core-updates)

- Bootstrap GNU without binutils, GCC, or C Library
- Halves the size of the trusted set of binaries
  - Debian: 450MB, Guix: 250MB, RBSb-Guix: 130MB

# Reduced Binary Seed bootstrap: Why?

## Safety/Security

- Ken Thompson's "Reflections on trusting trust" attack.

## Moral duty

- James Comey: ought to take responsibility for safety and security.

## We like source

- Everything in Guix is built from source, except the bootstrap binaries.

## Tradition

- This is how we used to do it.

## Pragmatism

- Support new hardware architecture

# Reduced Binary Seed bootstrap: Why?

## Legality

- Is it even legal to distribute a GCC binary in DisneyWorld?

## Inspiration

- Stage0's hex0 Monitor/Assembler.

# Bruce Schneier: "Trusting trust" attack gotten easier

*It's interesting: the "trusting trust" attack has actually gotten easier over time, because compilers have gotten increasingly complex, giving attackers more places to hide their attacks.*

*Here's how you can use a simpler compiler – that you can trust more – to act as a watchdog on the more sophisticated and more complex compiler. – Bruce Schneier, 2006*

# Peter Herdman: "Trusting trust" more fitting today

*Reflecting on 'Reflections on Trusting Trust'*

*Enterprises appear to be overlooking or bypassing robust software assurance processes and procedures*

*Thompsons essay is probably more fitting today than it was when it was written.*

*The moral of this article is that you still cannot trust any software. – Peter Herdman, 2014*

# David A. Wheeler: Address subverted bootstrap code

*Bootstrappable builds focuses on minimizing the amount of bootstrap binaries. They're not just interested in the direct "bootstrap" code to boot a computer, but also what is necessary to generate the direct bootstrap code.*

*The problem bootstrappable builds is trying to address is a real one, namely, they are worried about subverted bootstrap code. – David A. Wheeler, 2016*

# James Comey: Ought to take responsibility

> *I put a piece of tape [..] over the camera [of my personal laptop ..] so that people who don't have authority don't look at you. I think that's a good thing. I think people ought to take responsibility for their own safety and security. – US FBI director James Comey, 2016*

That probably also applies to dowloading binaries from the internet and running them; paraphrasing

> *The FBI thinks that we ought to bootstrap our computers from source.*

# Ludovic Courtès: Reduce seeds to bare minimum

*These big chunks of binary code are practically non-auditable which breaks the source to binary transparency that we get in the rest of the package dependency graph.*

*Every unauditable binary leaves us vulnerable to compiler backdoors as described by Ken Thompson in the 1984 paper Reflections on Trusting Trust.*

*Thus, our goal is to reduce the set of bootstrap binaries to the bare minimum. – Ludovic Courtès (GNU Guix documentation, December 2017)*

# Is it legal to distribute a GCC binary in DisneyWorld?

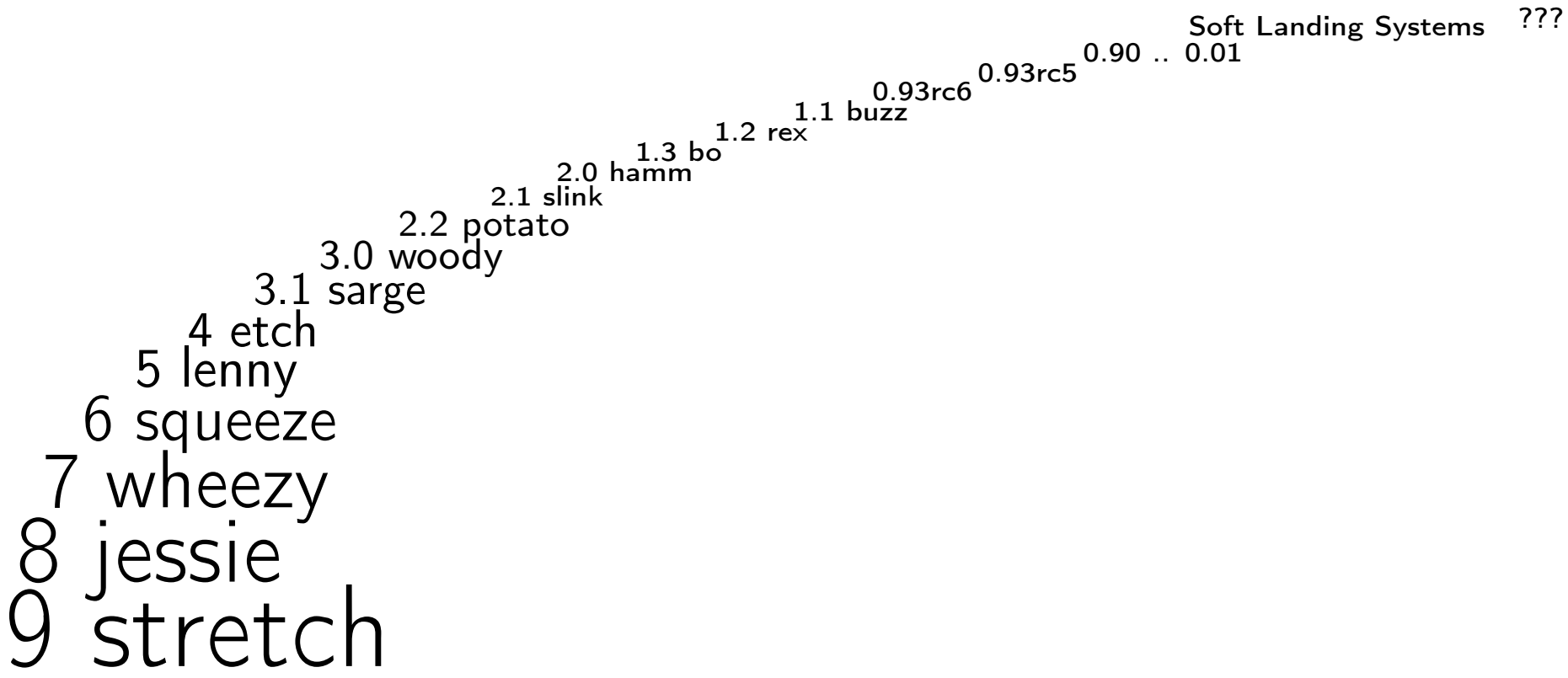
## Only if you distribute, or give access to the 'Corresponding Source'

*The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. – GNU GPL version 3*

## Let's assume GCC was built using GCC-1

- Is GCC-1 a 'System Library? or
- Is GCC-1 a 'general-purpose tool'? or
- Was GCC-1 a 'generally available free program'?

# Journey to the source?




*As time goes on we will expire the binary packages for old releases. Currently we have binaries for squeeze, lenny, etch, sarge, woody, potato, slink, hamm and bo available, and only source code for the other releases. – www.debian.org/distrib/archive*

# Inspiration: Stage0's 500 byte hex0 Monitor

```
## ELF Header
7F 45 4C 46         ## e_ident[EI_MAG0-3] ELF's magic number
02                  ## e_ident[EI_CLASS] Indicating 64 bit
01                  ## e_ident[EI_DATA] Indicating little endian
...

## ascii other
48 c7 c0 ff ff ff ff # mov $0xffffffffffffffff,%rax
c3                   # retq

## start
49 c7 c7 ff ff ff ff # mov $0xffffffffffffffff,%r15
49 c7 c6 00 00 00 00 # mov $0x0,%r14

## Loop
48 c7 c2 01 00 00 00 # mov $0x1,%rdx
48 c7 c6 99 01 60 00 # mov $0x600199,%rsi
```

# How: Remove Yoghurt-software!

# How: Remove Yoghurt-software!

# Is GNU GCC Yoghurt-software?


$\simeq$

?

# From boot-strap to boot-strip

=>

!

# Is TCC Yoghurt-software?

$\simeq$

?

# Is Mes+MesCC Yoghurt-software?

$\simeq$

?

# MesCC: Compile to M1

```
mescc -S scaffold/hello.c -o hello.M1
mescc scaffold/hello.c    -o a.out
```

## hello.c

```
void
main ()
{
  puts ("Hello, Mes!");
}
```

## hello.M1

```
:main
  push___%ebp
  mov____%esp,%ebp
  sub____$i32,%esp %0x1054
  push___$i32 &_string_0
  call32 %puts
  add____$i8,%esp !0x4
  leave
  ret
```

# MesCC [M1-macro]: Assemble to hex2

```
mescc -c scaffold/hello.c -o hello.hex2
```

## hello.hex2

```
:main
55
89e5
81ec
54100000
68
&_string_0
e8
%puts
83c4
04
c9
c3
```

# MesCC [hex2-linker]: Link to ELF

## M1-Macros

```
DEFINE push__%ebp          55
DEFINE mov___%esp,%ebp     89e5
DEFINE sub___$i32,%esp     81ec
DEFINE push__$i32          68
```

## ...continued

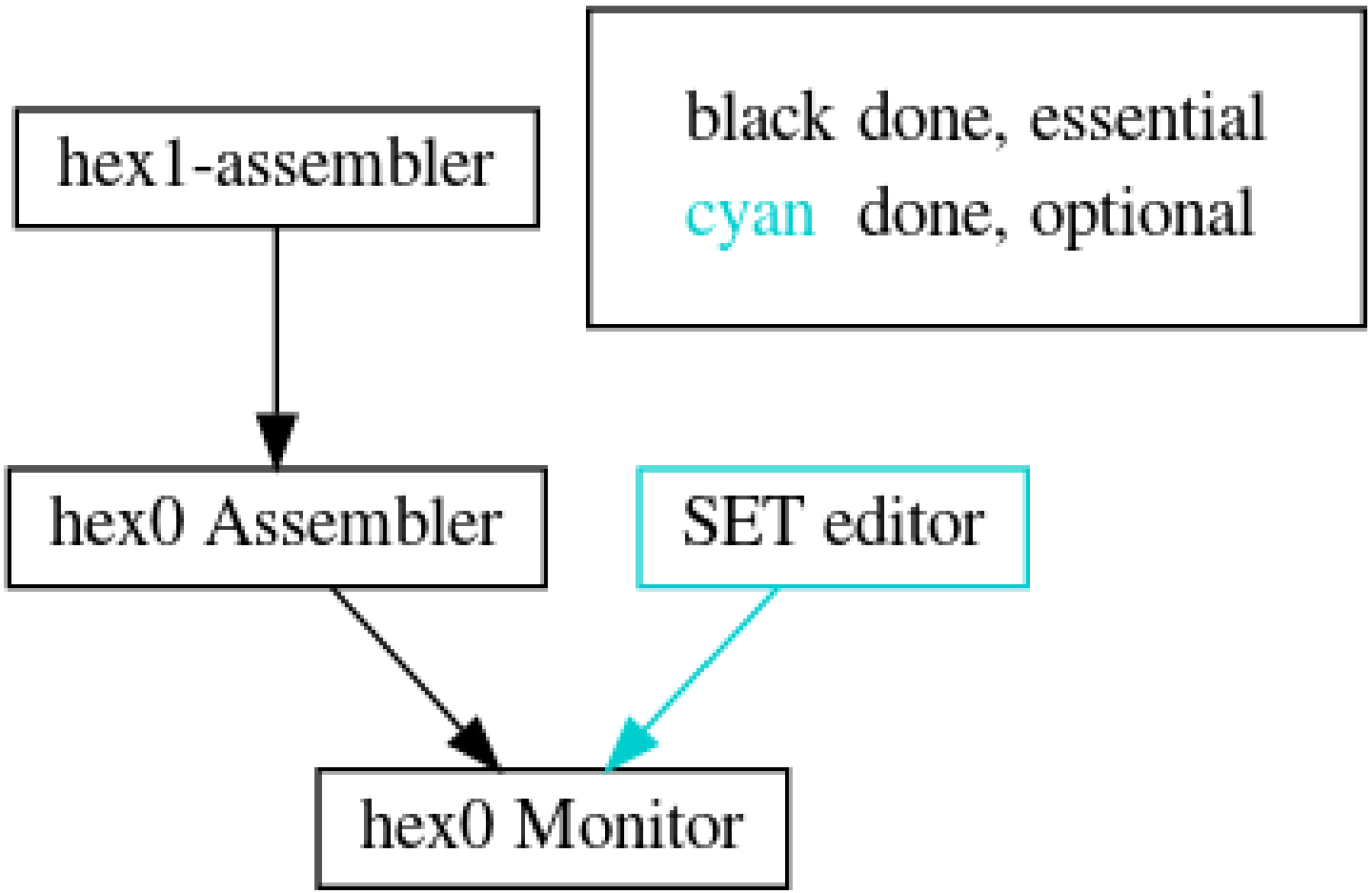
```
DEFINE call32              e8
DEFINE sub___$i32,%esp     81ec
DEFINE leave               c9
DEFINE ret                 c3
```

## a.out

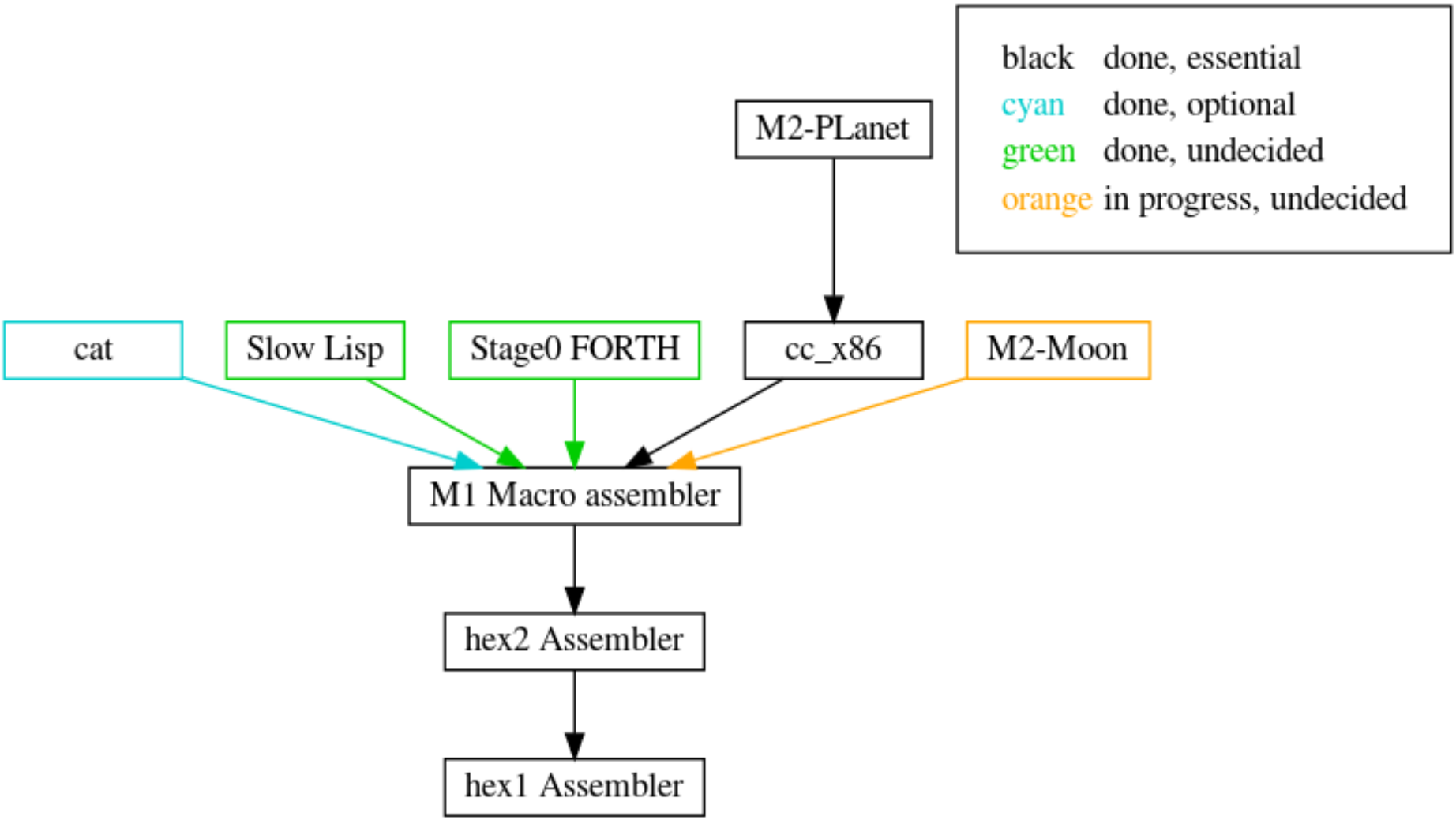
```
0100026d <main>:
 100026d: 55                    push   %ebp
 100026e: 89 e5                 mov    %esp,%ebp
 1000270: 81 ec 54 10 00 00     sub    $0x1054,%esp
 1000276: 68 9b 02 00 01        push   $0x100029b
 100027b: e8 dc 00 00 00        call   100035c <puts>
 1000280: 83 c4 04              add    $0x4,%esp
 1000283: c9                    leave
 1000284: c3                    ret
```

# Future: Aim for the Stars: Full Source Bootstrap

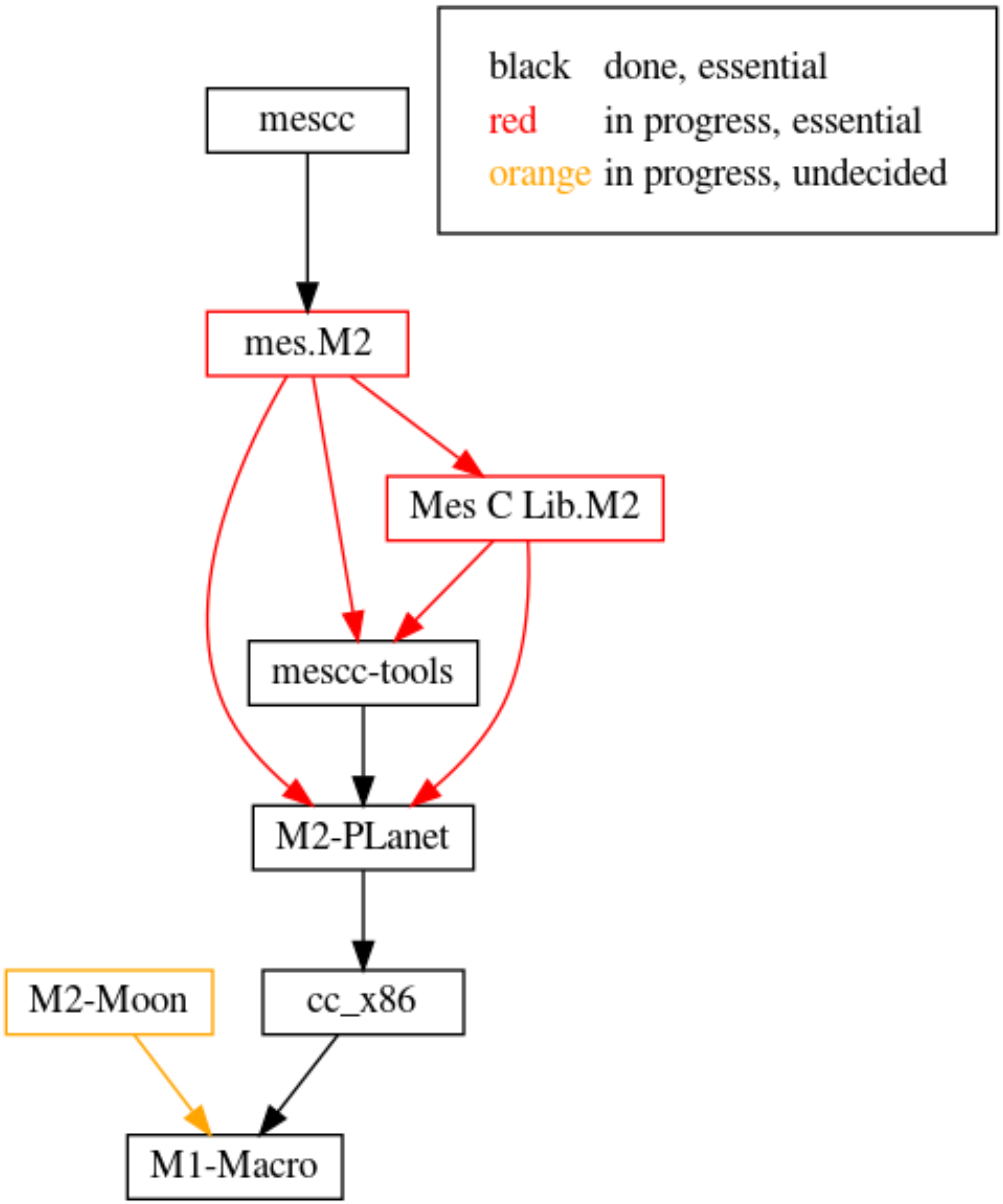
# Aim for the Stars: Stage 0



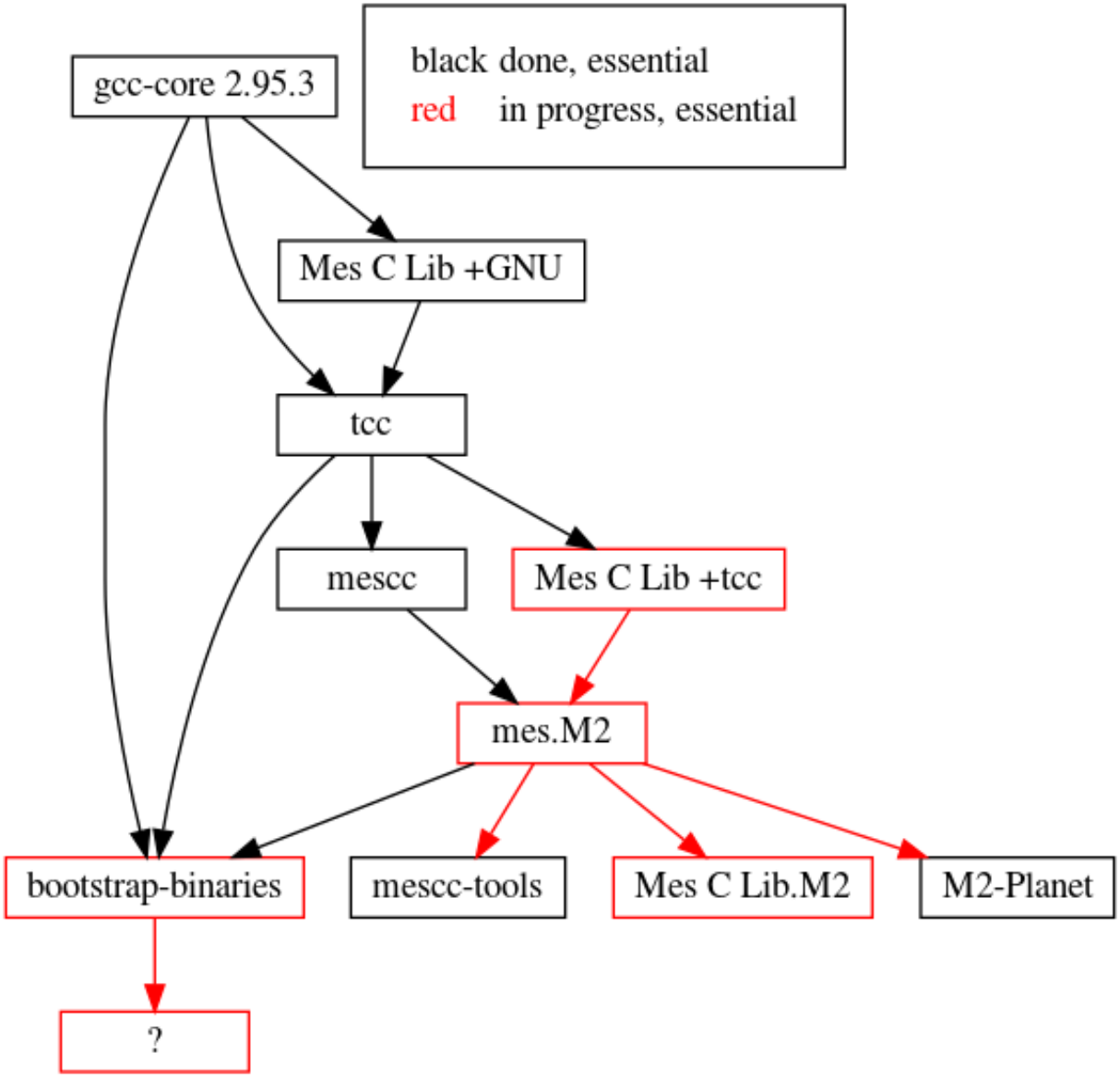
# Aim for the Stars: Stage 1



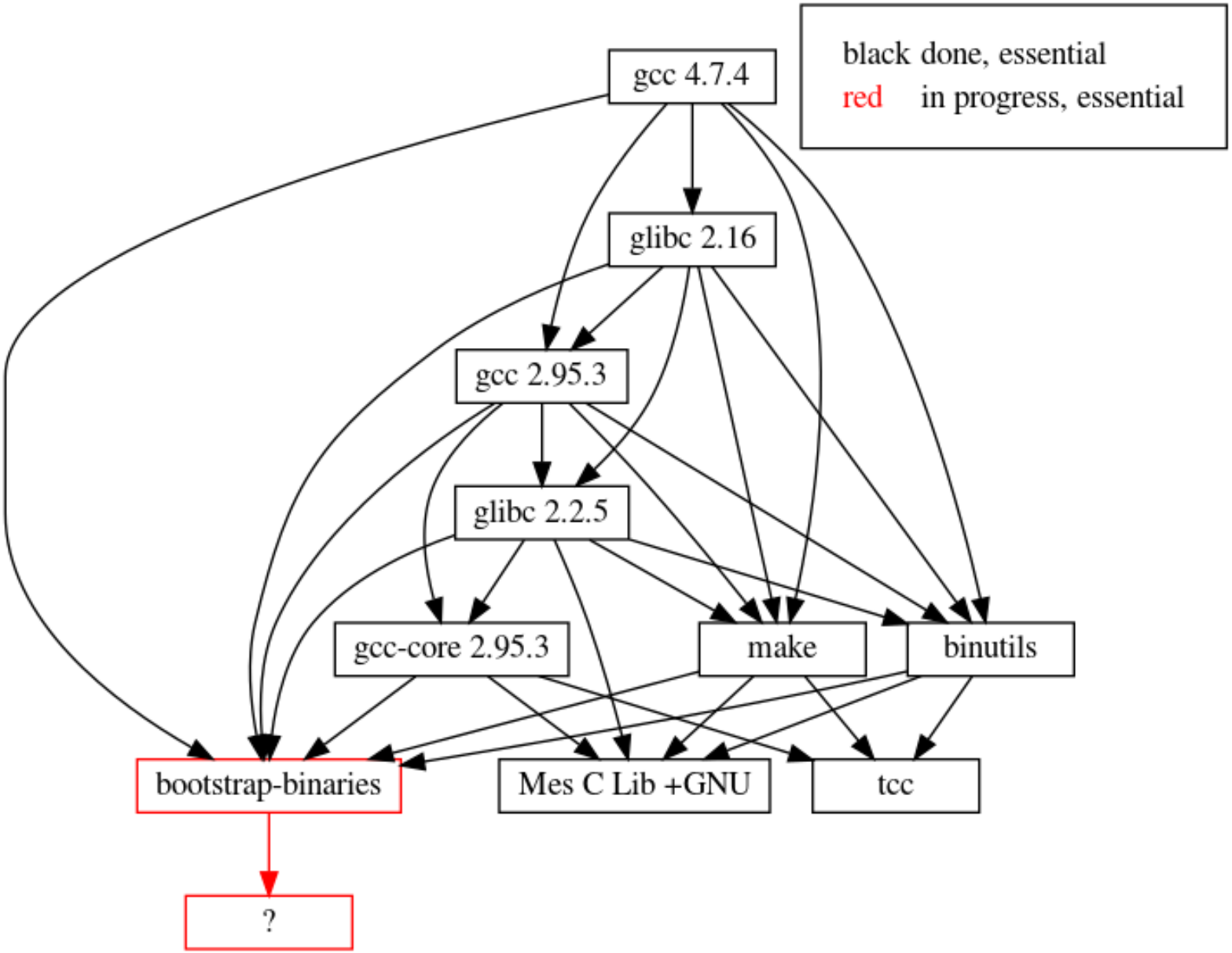
# Aim for the Stars: Stage 2



# Aim for the Stars: Stage mes



# Aim for the Stars: Stage mesboot



# Aim for the Stars: Further reductions

## In progress

- Gash: Scheme-only Bootstrap (Guix @wip-bootstrap)
  - Bootstrap Guix from only `mescc-tools`, `mes`, `gash`, `guile`.
- Mes v0.20: Mes C Lib support for awk, bash, sed, tar.
- Bootstrap Mes.M2 using M2-Planet.
- A Reduced Binary Seed bootstrap for Nix.
- Skip gcc-2.95.3 stage, build gcc-4.x directly?

## Later

- Inspire the GCC developers to write their own bootstrap story.
- Remove `bootstrap-mescc-tools`, `bootstrap-mes`.
- Fully replace `bootstrap-guile` with `bootstrap-mes`.
- Other Architectures (ARM).
- Non-functional distributions (Debian, a *BSD?).

# Aim for the Stars: Gash

## Recent merger between historical Gash and Geesh

- Gash: experimental PEG parser for Bash
  - focus on converting shell to Guile
  - shelly Guile scripting and interactive use
- Geesh: LALR parser for POSIX sh
  - focus on sh compliance and bootstrap

## Current focus

- Scheme-only bootstrap
- 0.1 release

## Features

- Bootstraps Bash 4.4: configure script, make shell snippets
- awk lexer parser basename cat chmod cmp compress cp cut diff dirname expr find grep ln ls mkdir mv printf reboot rm rmdir sed reader sleep sort tar test testb touch tr uname uniq wc which

# Thanks

## Thanks

- John McCarthy
- Eelco Dolstra
- Ludovic Courtès
- Matt Wette
- Jeremiah Orians
- Rutger van Beusekom

## Thanks everyone else

- LISP-1.5
- GNU/Linux
- Nix
- Debian
- Reproducible builds
- Guix

## Connect

- irc freenode.net #bootstrappable #guix
- mail guix-devel@gnu.org, bug-mes@gnu.org
- git https://git.savannah.gnu.org/git/mes.git
- web bootstrappable.org

# LISP as the Maxwell's Equations of Software

*That was the big revelation to me when I [..] finally understood that the half page of code on the bottom of page 13 of the Lisp 1.5 manual was Lisp in itself. These were "Maxwell's Equations of Software!" – Alan Kay*

# LISP-1.5 John McCarthy: page 13

```
apply[fn;x;a] =
      [atom[fn] → [eq[fn;CAR] → caar[x];
                   eq[fn;CDR] → cdar[x];
                   eq[fn;CONS] → cons[car[x];cadr[x]];
                   eq[fn;ATOM] → atom[car[x]];
                   eq[fn;EQ] → eq[car[x];cadr[x]];
                   T → apply[eval[fn;a];x;a]];
       eq[car[fn];LAMBDA] → eval[caddr[fn];pairlis[cadr[fn];x;a]];
       eq[car[fn];LABEL] → apply[caddr[fn];x;cons[cons[cadr[fn];
                                                  caddr[fn]];a]]]

eval[e;a] = [atom[e] → cdr[assoc[e;a]];
      atom[car[e]] →
                [eq[car[e],QUOTE] → cadr[e];
                eq[car[e];COND] → evcon[cdr[e];a];
                T → apply[car[e];evlis[cdr[e];a];a]];
      T → apply[car[e];evlis[cdr[e];a];a]]
```

# LISP-1.5 in Guile Scheme: APPLY

```
(define (apply fn x a)
  (cond
   ((atom fn)
    (cond
     ((eq fn CAR)  (caar x))
     ((eq fn CDR)  (cdar x))
     ((eq fn CONS) (cons (car x) (cadr x)))
     ((eq fn ATOM) (atom (car x)))
     ((eq fn EQ)   (eq (car x) (cadr x)))
     (#t           (apply (eval fn a) x a))))
   ((eq (car fn) LAMBDA)
                 (eval (caddr fn) (pairlis (cadr fn) x a)))
   ((eq (car fn) LABEL)
                 (apply (caddr fn) x (cons (cons (cadr fn)
                                                 (caddr fn))
                                           a)))))
```

# LISP-1.5 in Guile Scheme: EVAL

```
(define (eval e a)
  (cond
   ((atom e) (cdr (assoc e a)))
   ((atom (car e))
    (cond ((eq (car e) QUOTE) (cadr e))
          ((eq (car e) COND)  (evcon (cdr e) a))
          (#t                 (apply (car e)
                                     (evlis (cdr e) a) a))))
   (#t       (apply (car e) (evlis (cdr e) a) a))))
```

# LISP-1.5 in Scheme: ASSOC, PAIRLIS, EVCON, EVLIS

```
(define (assoc x a)
  (cond ((eq (caar a) x) (car a))
        (#t (assoc x (cdr a)))))

(define (pairlis x y a)
  (cond ((null x) a)
        (#t (cons (cons (car x) (car y))
                  (pairlis (cdr x) (cdr y) a)))))

(define (evcon c a)
  (cond ((eval (caar c) a) (eval (cadar c) a))
        (#t (evcon (cdr c) a))))

(define (evlis m a)
  (cond ((null m) NIL)
        (#t (cons (eval (car m) a) (evlis (cdr m) a)))))
```

# History – 1984 Four Software Freedoms: GNU GPL

## 1984 Four Software Freedoms: GNU GPL

- The freedom to
  - 0 run the program as you wish, for any purpose
  - 1 study how the program works, and change it if you wish
  - 2 redistribute copies so you can help your neighbor
  - 3 share copies of your modified versions with others

– *Richard M. Stallman*

# History – 1990s Reproducible GNU Tools @Cygnus

- 1984 Four Software Freedoms: GNU GPL

## 1990s Reproducible GNU Tools @Cygnus

*We made the GNU tools that we were shipping and supporting – and all of our test cases compiled by them – reproducible. That includes gcc, gdb, gas, binutils, gnu make, and a few other things. – John Gilmore*

# History – 2006 Nix: Purely Functional Software Deployment

- 1984 Four Software Freedoms: GNU GPL
- 1990s Reproducible GNU Tools @Cygnus

## 2006 Nix: Purely Functional Software Deployment

- functional package management
- isolated builds
- Nix (and GNU Guix) are designed for reproducibility

*Installation of a component can lead to the failure of previously installed components; a component might require other components that are not present; and it is difficult to undo deployment actions.*

*This thesis describes a better approach based on a purely functional deployment model, implemented in a deployment system called Nix. – Eelco Dolstra*

# History – 2007 debian-devel: Reproducibility

- 1984 Four Software Freedoms: GNU GPL
- 1990s Reproducible GNU Tools @Cygnus
- 2006 Nix: Functional package management

## 2007 debian-devel: Reproducibility

*I think it would be really cool if the Debian policy required that packages could be rebuild bit-identical from source.*

*At the moment, it is impossible to independly verify the integricity of binary packages. – Martin Uecker*

# History – 2012 GNU Guix: user atonomy and safety

- 1984 Four Software Freedoms: GNU GPL
- 1990s Reproducible GNU Tools @Cygnus
- 2006 Nix: Functional package management
- 2007 debian-devel: Reproducibility

## 2012 GNU Guix: user atonomy and safety

- Reproducible builds: a means to an end
- User autonomy and safety

*We view "reproducible builds" as a technical means to an end: that of guaranteeing user autonomy and safety. – Ludovic Courtès*

# History – 2013 DebConf13: reproducible-builds.org

- 1984 Four Software Freedoms: GNU GPL
- 1990s Reproducible GNU Tools @Cygnus
- 2006 Nix: Functional package management
- 2007 debian-devel: Reproducibility
- 2012 GNU Guix: user atonomy and safety

## 2013 DebConf13: reproducible-builds.org

- Lunar organizes reproducible-builds.org

*A build is reproducible if given the same source code, build environment and build instructions, any party can recreate bit-by-bit identical copies of all specified artifacts. – reproducible-builds.org*

# History – 2016 R-B Summit II: bootstrappable.org

- 1984 Four Software Freedoms: GNU GPL
- 1990s Reproducible GNU Tools @Cygnus
- 2006 Nix: Functional package management
- 2007 debian-devel: Reproducibility
- 2012 GNU Guix: user atonomy and safety
- 2013 DebConf13: reproducible-builds.org

## 2016 R-B Summit II: bootstrappable.org

- reproducible builds summit II
- session: Writing a statement about what it means to do bootstrappable compilers II
- host: Ludovic Courtès
- result: Following up on the first session focusing on this effort, the group drafted a first version of the bootstrappable.org website.

# History – 2016 Initial release of Stage0 and Mes

- 1984 Four Software Freedoms: GNU GPL
- 1990s Reproducible GNU Tools @Cygnus
- 2006 Nix: Functional package management
- 2007 debian-devel: Reproducibility
- 2012 GNU Guix: user atonomy and safety
- 2013 DebConf13: reproducible-builds.org
- 2016 R-B Summit II: bootstrappable.org

## 2016 Initial release of Stage0 and Mes

Release of Stage0 and Mes

# History – 2018 Reduced Binary Seed bootstrap

- 1984 Four Software Freedoms: GNU GPL
- 1990s Reproducible GNU Tools @Cygnus
- 2006 Nix: Functional package management
- 2007 debian-devel: Reproducibility
- 2012 GNU Guix: user atonomy and safety
- 2013 DebConf13: reproducible-builds.org
- 2016 R-B Summit II: bootstrappable.org
- 2016 Initial release of Stage0 and Mes

## 2018 Reduced Binary Seed bootstrap

This talk!

# Timeline 2016

## October 23: 0.1 [not announced]

- `let-syntax`, `match`
- compile main.c in 2s (was 1'20")
- add REPL

## November 21: 0.2 [not announced]

- psyntax integration, `syntax-case`, `load`

## December 12: on bootstrapping: first Mes 0.3 released

- Garbage Collector/Jam Scraper

## December 25: Mes 0.4 released

- run Nyacc, PEG, reduced core

# Timeline 2017

## April 27: Mes 0.5 released

- mutual self-hosting
  - mes.c runs mescc.scm
  - mescc.scm compiles mes.c

## May 14: Mes 0.6 released

- MesCC runs on unpatched Nyacc
- MesCC compiles 33/55 of tinycc/tests/test2

## June 3: Mes 0.7 released

- Mes C Library headers and stubs support working on compiling tcc.c

## June 25: Mes 0.8 released

- MesCC compiles to stage0's hex2 format

# Timeline 2017-2

## July 26: Mes 0.9 released

- MesCC compiles mes-tcc, to a mostly segfaulting executable

## September 10: Mes 0.10 released

- mes-tcc can compile a working trivial C program "int main () {return 42;}"

## November 18: Mes 0.11 released

- MesCC: test suite with 69 tests
- less-heavily patched mes-tcc passes 41/69 MesCC C tests

# Timeline 2018

## April 8: Mes 0.12 released

- performance work: MesCC compiles mes-tcc in ~2h30' (was: ~1day)

## April 28: Mes 0.13 released

- MesCC builds functional mes-tcc
- Patches offered to tcc community, rejected

## May 24: Mes 0.14 released

- MesCC builds functional, only slightly patched mes-tcc

## June 12: Mes 0.15 released

- Experimental Guix integration
- Mes C Library supports building binutils-2.14, gcc-2.95.3, glibc-2.2.5.

# Timeline 2018-2

## June 26: Mes 0.16 released

- Fix ELF header bug: all Mes binaries segfault on Linux 4.17
- Guix integration: build gcc-4.1.0

## August 10: GNU Mes 0.17 released

- Mes is an official GNU package
- Guix integration: build gcc-4.7.4

## Ocotber 7: GNU Mes 0.18 released

- Guix integration: Reduced Binary Seed bootstrap (cheat using Guile)
- Introduce embarrasing bug: MesCC only runs on Guile

## December 16: GNU Mes 0.19 released

- Compile mes-tcc in ~8' (was: ~1h30).
- Guix integration: Remove MesCC-on-Guile shortcut

# Metrics: Mes since Fosdem'17

- 14 releases: 0.5..0.19
- 1174 commits

# Metrics: simplifying-tcc patches

```
 135 0001-bootstrappable-Outline-elf-unions.patch
  41 0002-bootstrappable-Outline-CValue_str.patch
  44 0003-bootstrappable-Outline-enum-TCCState_pflag.patch
 162 0005-bootstrappable-Heterogeneous-initializer-list.patch
  51 0006-bootstrappable-Simple-initializer-lists.patch
 496 0007-bootstrappable-Heterogeneous-switch-case.patch
  26 0008-bootstrappable-(foo--)->bar.baz.patch
  47 0010-bootstrappable-foo (bar (), baz ()).patch
 176 0011-bootstrappable-foo ()->bar.patch
  39 0012-bootstrappable-char foo[][].patch
  94 0013-bootstrappable-Multi-line-strings.patch
 187 0014-bootstrappable-sizeof-type.patch
  36 0015-bootstrappable-str-r-chr-str-0.patch
  30 0016-bootstrappable-uint16_t-in-struct-on-heap.patch
  64 0017-bootstrappable-constant-pointer-arithmetic.patch
1704 total
```

# Metrics: remaining tcc patches

```
 35 0001-bootstrappable-Work-around-Nyacc-0.80.42-bug.patch
 47 0002-bootstrappable-HAVE_LONG_LONG.patch
 47 0003-bootstrappable-HAVE_BITFIELD.patch
 94 0004-bootstrappable-HAVE_FLOAT.patch
 27 0005-bootstrappable-Skip-tidy_section_headers.patch
 26 0006-bootstrappable-Handle-libtcc1.a.patch
 30 0007-bootstrappable-uint16_t-in-struct-on-heap.patch
193 0008-bootstrappable-add-tcc.h-include-guards-to-include-l.p
 33 0009-bootstrappable-Work-around-MesCC-bug.patch
 26 0010-bootstrappable-Force-static-link.patch
558 total
```

# Metrics: GNU patches

```
 26 tcc-boot-0.9.27.patch
157 binutils-boot-2.20.1a.patch
137 gcc-boot-2.95.3.patch
251 glibc-boot-2.2.5.patch
 68 gcc-boot-4.7.4.patch
352 glibc-boot-2.16.0.patch
991 total
```

# Metrics: GNU Guix patches

```
gnu: Use i686-linux bootstrap binaries on x86_64-linux.
bootstrap: Merge mes-minimal into mes-minimal-stripped.
doc: Update mesboot graph without bootstrap-guile.
bootstrap: Do not fake, use Mes instead of Guile.
bootstrap: bootstrap-mes: Update.
bootstrap: mes-minimal-stripped: Do not strip bin.
bootstrap: Switch to official bootstrap urls.
bootstrap: mes-boot: Use mes-boot0 version.
doc: Update for bootstrap-mescc-tools change.
bootstrap: Force i686-linux for bootstrap-tarballs.
bootstrap: Update %bootstrap-tarballs.
bootstrap: Replace %mescc-tools-seed with %bootstrap-mescc-tool
bootstrap: Update %bootstrap-mes.
bootstrap: Add %bootstrap-mescc-tools.
bootstrap: Add %mes-minimal.
bootstrap: Add mescc-tools-static, mescc-tools-static-tarball.
```