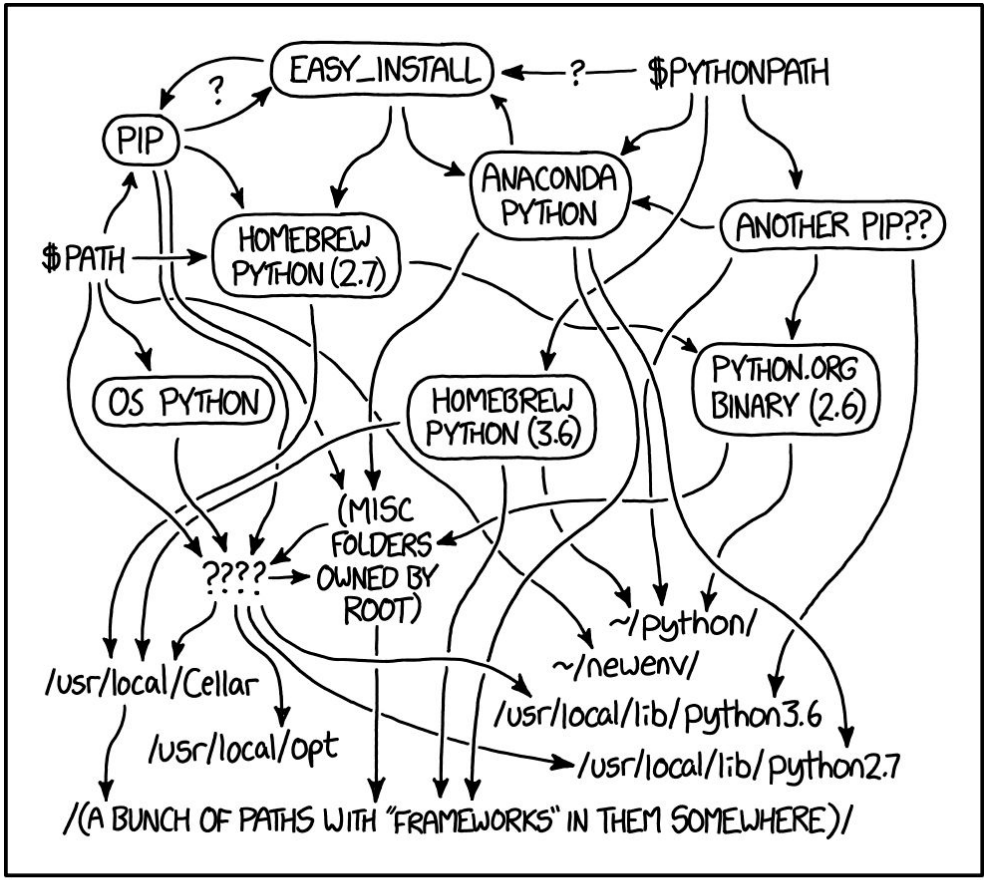
Robust Linux Binaries

G. Amadio



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

https://xkcd.com/1987

Overview

- Why **LD_LIBRARY_PATH** has drawbacks
- How dynamic library loading works
- Alternatives to **LD_LIBRARY_PATH**
- Building a solid base for HEP software stacks

Why LD_LIBRARY_PATH has drawbacks

- **LD_LIBRARY_PATH** is commonly used to add directories to the linker's search path for software installed in non-standard paths. It's what everyone is usually doing
- If added paths contain many installed packages, they may override system libraries
- Common problem: source a setup script from CVMFS, then try to use system software, like **vim** or **git** (e.g. https://sft.its.cern.ch/jira/browse/SPI-1083)

```
$ ssh lxplus7
$ lsb_release -d
Description:    CentOS Linux release 7.5.1804 (Core)
$ source /cvmfs/sft.cern.ch/lcg/views/LCG_latest/x86_64-centos7-gcc7-opt/setup.sh
$ ldd /usr/bin/git
      linux-vdso.so.1 => (0x00007ffdb55b3000)
      libpcre.so.1 => /cvmfs/sft.cern.ch/lcg/views/LCG_latest/x86_64-centos7-gcc7-opt/lib/libpcre.so.1
      libz.so.1 => /cvmfs/sft.cern.ch/lcg/views/LCG_latest/x86_64-centos7-gcc7-opt/lib/libz.so.1
      libpthread.so.0 => /lib64/libpthread.so.0
      libc.so.6 => /lib64/libc.so.6
      /lib64/ld-linux-x86-64.so.2
```

Why LD_LIBRARY_PATH has drawbacks (2)

```
$ ssh lxplus7
$ source /cvmfs/sft.cern.ch/lcg/views/LCG_latest/x86_64-centos7-gcc7-opt/setup.sh
$ source /cvmfs/projects.cern.ch/intelsw/psxe/linux/18-all-setup.sh
$ echo $LD_LIBRARY_PATH
/cvmfs/projects.cern.ch/intelsw/psxe/linux/x86_64/2018/itac/2018.2.020/intel64/slib:/cvmfs/projects.cern.ch/intelsw/psxe
/linux/x86_64/2018/compilers_and_libraries_2018.2.199/linux/compiler/lib/intel64:/cvmfs/projects.cern.ch/intelsw/psxe/li
nux/x86_64/2018/compilers_and_libraries_2018.2.199/linux/compiler/lib/intel64_lin:/cvmfs/projects.cern.ch/intelsw/psxe/l
inux/x86_64/2018/compilers_and_libraries_2018.2.199/linux/mpi/intel64/lib:/cvmfs/projects.cern.ch/intelsw/psxe/linux/x86
_64/2018/compilers_and_libraries_2018.2.199/linux/mpi/mic/lib:/cvmfs/projects.cern.ch/intelsw/psxe/linux/x86_64/2018/com
pilers_and_libraries_2018.2.199/linux/ipp/lib/intel64:/cvmfs/projects.cern.ch/intelsw/psxe/linux/x86_64/2018/compilers_a
nd_libraries_2018.2.199/linux/compiler/lib/intel64_lin:/cvmfs/projects.cern.ch/intelsw/psxe/linux/x86_64/2018/compilers_
and_libraries_2018.2.199/linux/mkl/lib/intel64_lin:/cvmfs/projects.cern.ch/intelsw/psxe/linux/x86_64/2018/compilers_and_
libraries_2018.2.199/linux/tbb/lib/intel64/gcc4.7:/cvmfs/projects.cern.ch/intelsw/psxe/linux/x86_64/2018/compilers_and_l
ibraries_2018.2.199/linux/tbb/lib/intel64/gcc4.7:/cvmfs/projects.cern.ch/intelsw/psxe/linux/x86_64/2018/debugger_2018/ig
a/lib:/cvmfs/projects.cern.ch/intelsw/psxe/linux/x86_64/2018/debugger_2018/libipt/intel64/lib:/cvmfs/projects.cern.ch/in
telsw/psxe/linux/x86_64/2018/compilers_and_libraries_2018.2.199/linux/daal/lib/intel64_lin:/cvmfs/projects.cern.ch/intel
sw/psxe/linux/x86_64/2018/compilers_and_libraries_2018.2.199/linux/daal/../tbb/lib/intel64_lin/gcc4.4:/cvmfs/sft.cern.ch
/lcg/views/LCG_latest/x86_64-centos7-gcc7-opt/lib/python2.7/site-packages/tensorflow:/cvmfs/sft.cern.ch/lcg/views/LCG_la
test/x86_64-centos7-gcc7-opt/lib/python2.7/site-packages/tensorflow/contrib/tensor_forest:/cvmfs/sft.cern.ch/lcg/views/L
CG_latest/x86_64-centos7-gcc7-opt/lib/python2.7/site-packages/tensorflow/python/framework:/cvmfs/sft.cern.ch/lcg/release
s/java/8u91-ae32f/x86_64-centos7-gcc7-opt/jre/lib/amd64:/cvmfs/sft.cern.ch/lcg/views/LCG_latest/x86_64-centos7-gcc7-opt/
lib64:/cvmfs/sft.cern.ch/lcg/views/LCG_latest/x86_64-centos7-gcc7-opt/lib:/cvmfs/sft.cern.ch/lcg/contrib/gcc/7binutils/x
86_64-centos7/lib64:/cvmfs/sft.cern.ch/lcg/contrib/binutils/2.28/x86_64-centos7/lib
```

How to ensure consistency between all libraries?

Executable and Linking Format (ELF)

```
$ readelf -h $(which bash)

ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x41d3e1
  Start of program headers:          64 (bytes into file)
  Start of section headers:          962688 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         29
  Section header string table index: 28
```

ELF header
program header
.text section
.data section
.bss section
.dynamic
.rel*
.symtab
.debug
section header table

bash

https://binvis.io

Executable and Linking Format (ELF)

```
$ readelf -d $(which bash)

Dynamic section at offset 0xdde08 contains 26 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libtinfo.so.5]
 0x0000000000000001 (NEEDED)             Shared library: [libdl.so.2]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0x41ae98
 0x000000000000000d (FINI)               0x4a6cd4
 0x0000000000000019 (INIT_ARRAY)         0x6dddf0
 0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x6dddf8
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
 0x000000006ffffef5 (GNU_HASH)           0x400298
 0x0000000000000005 (STRTAB)             0x4104a0
 0x0000000000000006 (SYMTAB)             0x403960
 0x000000000000000a (STRSZ)              33816 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000015 (DEBUG)              0x0
 0x0000000000000003 (PLTGOT)             0x6de000
 0x0000000000000002 (PLTRELSZ)           4992 (bytes)
 0x0000000000000014 (PLTREL)             RELA
 0x0000000000000017 (JMPREL)             0x419b18
 0x0000000000000007 (RELA)               0x419a58
 0x0000000000000008 (RELASZ)             192 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x000000006ffffffe (VERNEED)            0x4199a8
 0x000000006fffffff (VERNEEDNUM)         2
 0x000000006ffffff0 (VERSYM)             0x4188b8
 0x0000000000000000 (NULL)               0x0
```

ELF header
program header
.text section
.data section
.bss section
.dynamic
.rel*
.symtab
.debug
section header table

bash

https://binvis.io

How does the linker load dynamic libraries?

If a shared object dependency does not contain a slash, then it is searched for in the following order:

- Using the **DT_RPATH** attribute if present *and* ***DT_RUNPATH*** *attribute does not exist*
- Using the environment variable **LD_LIBRARY_PATH**
- Using the **DT_RUNPATH** attribute if present (only for dependencies in **DT_NEEDED**)
- From the cache file **/etc/ld.so.cache**
- In the default paths **/lib**, then **/usr/lib** (**/lib64**, then **/usr/lib64** in 64-bit architectures).

Alternatives to LD_LIBRARY_PATH

- Compile packages injecting **DT_RPATH** and/or **DT_RUNPATH**
- Keeps user environment clean
- Each library looks first in its embedded paths, then the system
- What about relocation?
  - Use **patchelf** or **chrpath** command line tools to change the **RPATH**
  - Configure with final install prefix and/or re-link during installation
  - Use a staged installation for packaging

```
$ readelf -d $(which root.exe)
Dynamic section at offset 0x1da0 contains 31 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libRint.so.6.14]
 0x0000000000000001 (NEEDED)             Shared library: [libCore.so.6.14]
 0x0000000000000001 (NEEDED)             Shared library: [libstdc++.so.6]
 0x0000000000000001 (NEEDED)             Shared library: [libgcc_s.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000001d (RUNPATH)            Library runpath: [/usr/lib64/root/6.14/lib64]
 0x000000000000000c (INIT)               0x998
 0x000000000000000d (FINI)               0xc74
 0x0000000000000019 (INIT_ARRAY)         0x201d80
 0x000000000000001b (INIT_ARRAYSZ)       16 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x201d90
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
 0x000000006ffffef5 (GNU_HASH)           0x278
 0x0000000000000005 (STRTAB)             0x560
 0x0000000000000006 (SYMTAB)             0x308
 0x000000000000000a (STRSZ)              525 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000015 (DEBUG)              0x0
 0x0000000000000003 (PLTGOT)             0x202000
 0x0000000000000002 (PLTRELSZ)           144 (bytes)
 0x0000000000000014 (PLTREL)             RELA
 0x0000000000000017 (JMPREL)             0x908
 0x0000000000000007 (RELA)               0x800
 0x0000000000000008 (RELASZ)             264 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x000000006ffffffb (FLAGS_1)            Flags: PIE
 0x000000006ffffffe (VERNEED)            0x7a0
 0x000000006fffffff (VERNEEDNUM)         3
 0x000000006ffffff0 (VERSYM)             0x76e
 0x000000006ffffff9 (RELACOUNT)          4
 0x0000000000000000 (NULL)               0x0
```

```
$ which root.exe
/usr/lib64/root/6.14/bin/root.exe

$ ldd $(which root.exe)
      linux-vdso.so.1 (0x00007ffffaef8000)
      libRint.so.6.14 => /usr/lib64/root/6.14/lib64/libRint.so.6.14 (0x00007ff1ab6e5000)
      libCore.so.6.14 => /usr/lib64/root/6.14/lib64/libCore.so.6.14 (0x00007ff1ab082000)
      libstdc++.so.6 => /usr/lib/gcc/x86_64-pc-linux-gnu/8.1.0/libstdc++.so.6 (0x00007ff1aac7b000)
      libgcc_s.so.1 => /usr/lib/gcc/x86_64-pc-linux-gnu/8.1.0/libgcc_s.so.1 (0x00007ff1aaa63000)
      libc.so.6 => /lib64/libc.so.6 (0x00007ff1aa697000)
      /lib64/ld-linux-x86-64.so.2 (0x00007ff1abb11000)
      libpcre.so.1 => /lib64/libpcre.so.1 (0x00007ff1aa425000)
      liblzma.so.5 => /lib64/liblzma.so.5 (0x00007ff1aa1ff000)
      libxxhash.so.0 => /usr/lib64/libxxhash.so.0 (0x00007ff1a9ffc000)
      liblz4.so.1 => /usr/lib64/liblz4.so.1 (0x00007ff1a9dde000)
      libz.so.1 => /lib64/libz.so.1 (0x00007ff1a9bc7000)
      libdl.so.2 => /lib64/libdl.so.2 (0x00007ff1a99c3000)
      libpthread.so.0 => /lib64/libpthread.so.0 (0x00007ff1a97a3000)
      libm.so.6 => /lib64/libm.so.6 (0x00007ff1a9417000)

$ ldd /usr/lib64/root/6.12/bin/root.exe | grep root
      libRint.so.6.12 => /usr/lib64/root/6.12/lib64/libRint.so.6.12 (0x00007f0d8de7a000)
      libCore.so.6.12 => /usr/lib64/root/6.12/lib64/libCore.so.6.12 (0x00007f0d8d83f000)
```

```
$ cat Makefile

CC=gcc
CXX=g++

ROOTSYS=$(shell root-config --prefix)
ROOTFLAGS=$(shell root-config --cflags)
ROOTLDFLAGS=-Wl,-rpath=$(ROOTSYS)/lib
ROOTLIBS=$(ROOTLDFLAGS) $(shell root-config --libs)

root-ls: root-ls.cc
        $(CXX) $(CXXFLAGS) $(ROOTFLAGS) -o $@ $^ \
               $(ROOTLIBS) $(LDFLAGS)

$ ls
Makefile root-ls.cc

$ make root-ls
```

```
$ ldd root-ls | grep root
        libCore.so.6.14 => /usr/lib64/root/6.14/lib/libCore.so.6.14
        libImt.so.6.14 => /usr/lib64/root/6.14/lib/libImt.so.6.14
        libRIO.so.6.14 => /usr/lib64/root/6.14/lib/libRIO.so.6.14
        libNet.so.6.14 => /usr/lib64/root/6.14/lib/libNet.so.6.14
        libHist.so.6.14 => /usr/lib64/root/6.14/lib/libHist.so.6.14
        libGraf.so.6.14 => /usr/lib64/root/6.14/lib/libGraf.so.6.14
        libGraf3d.so.6.14 => /usr/lib64/root/6.14/lib/libGraf3d.so.6.14
        libGpad.so.6.14 => /usr/lib64/root/6.14/lib/libGpad.so.6.14
        libROOTDataFrame.so.6.14 =>
           /usr/lib64/root/6.14/lib/libROOTDataFrame.so.6.14
        libROOTVecOps.so.6.14 =>
           /usr/lib64/root/6.14/lib/libROOTVecOps.so.6.14
        libTree.so.6.14 => /usr/lib64/root/6.14/lib/libTree.so.6.14
        libTreePlayer.so.6.14 ⇒
           /usr/lib64/root/6.14/lib/libTreePlayer.so.6.14
        libRint.so.6.14 => /usr/lib64/root/6.14/lib/libRint.so.6.14
        libPostscript.so.6.14 =>
           /usr/lib64/root/6.14/lib/libPostscript.so.6.14
        libMatrix.so.6.14 => /usr/lib64/root/6.14/lib/libMatrix.so.6.14
        libPhysics.so.6.14 =>
           /usr/lib64/root/6.14/lib/libPhysics.so.6.14
        libMathCore.so.6.14 =>
           /usr/lib64/root/6.14/lib/libMathCore.so.6.14
        libThread.so.6.14 => /usr/lib64/root/6.14/lib/libThread.so.6.14
        libMultiProc.so.6.14 =>
          /usr/lib64/root/6.14/lib/libMultiProc.so.6.14
```

Relative RPATHs

- Link with relative **RPATH** with **${ORIGIN}** and **${LIB}**
- **${ORIGIN}** is the directory where the library/binary is
- **${LIB}** is either **lib** or **lib64**
- Link binaries with **${ORIGIN}/../${LIB}** in **RPATH**

Building a Solid Base for HEP

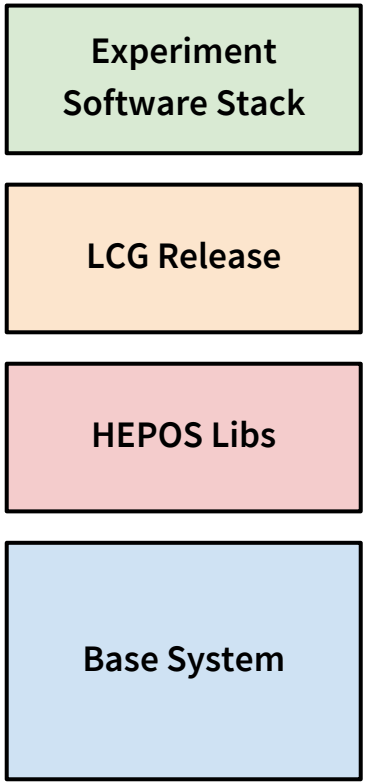
Platform Combinatorics

Currently, the LCG stack relies on libraries installed on the base system and on the HEPOS libs meta-package.

Since each Linux distribution uses its own version of libc, binutils, GCC, and includes a different set of system packages, the HEPOS libs meta-package contains different packages for different distros.

For that reason, LCG must also be tuned to provide what's missing in these base layers.

Building everything from the ground up would make it possible to reduce platform combinatorics.




```
$ cd /cvmfs/sft.cern.ch/lcg/views
$ ls LCG_latest
x86_64-centos7-clang60-opt
x86_64-centos7-gcc62-dbg
x86_64-centos7-gcc62-opt
x86_64-centos7-gcc7-dbg
x86_64-centos7-gcc7-opt
x86_64-slc6-clang60-opt
x86_64-slc6-gcc62-dbg
x86_64-slc6-gcc62-opt
x86_64-slc6-gcc7-dbg
x86_64-slc6-gcc7-opt
x86_64-ubuntu1604-gcc54-dbg
x86_64-ubuntu1604-gcc54-opt
```

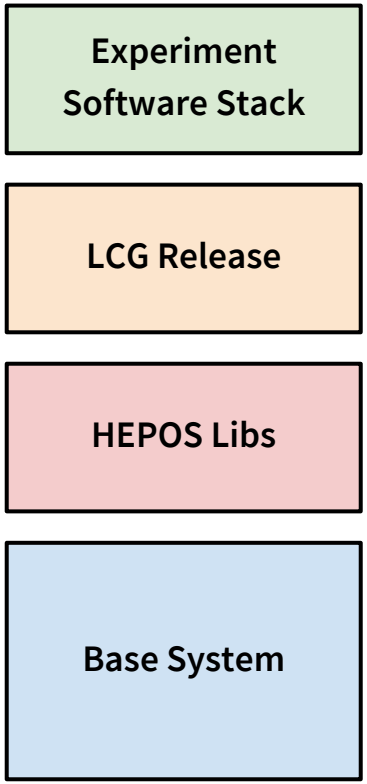
Platform Combinatorics

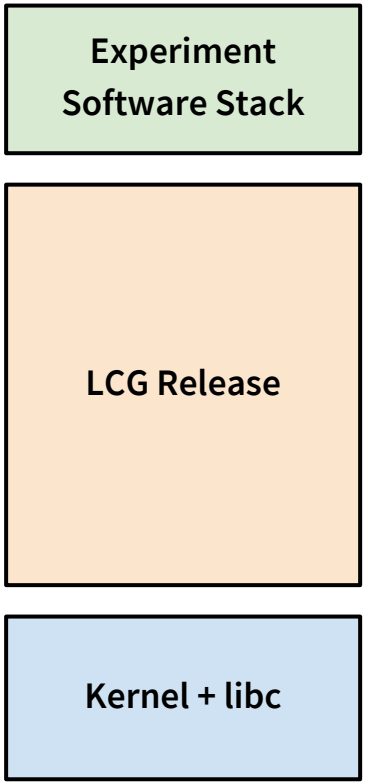
Currently, the LCG stack relies on libraries installed on the base system and on the HEPOS libs meta-package.

Since each Linux distribution uses its own version of libc, binutils, GCC, and includes a different set of system packages, the HEPOS libs meta-package contains different packages for different distros.

For that reason, LCG must also be tuned to provide what's missing in these base layers.

Building everything from the ground up would make it possible to reduce platform combinatorics.






```
$ ls LCG_latest
x86_64-linux
x86_64-macos-10.12
x86_64-macos-10.13
```

Including HEPOS libs and other packages picked up from the system into the LCG release would allow to reduce the number of platforms, since the Linux kernel promises to never break userspace APIs.

Unfortunately, for macOS (Darwin + Apple libc), breakage is common between releases, and some distros have really old glibc.

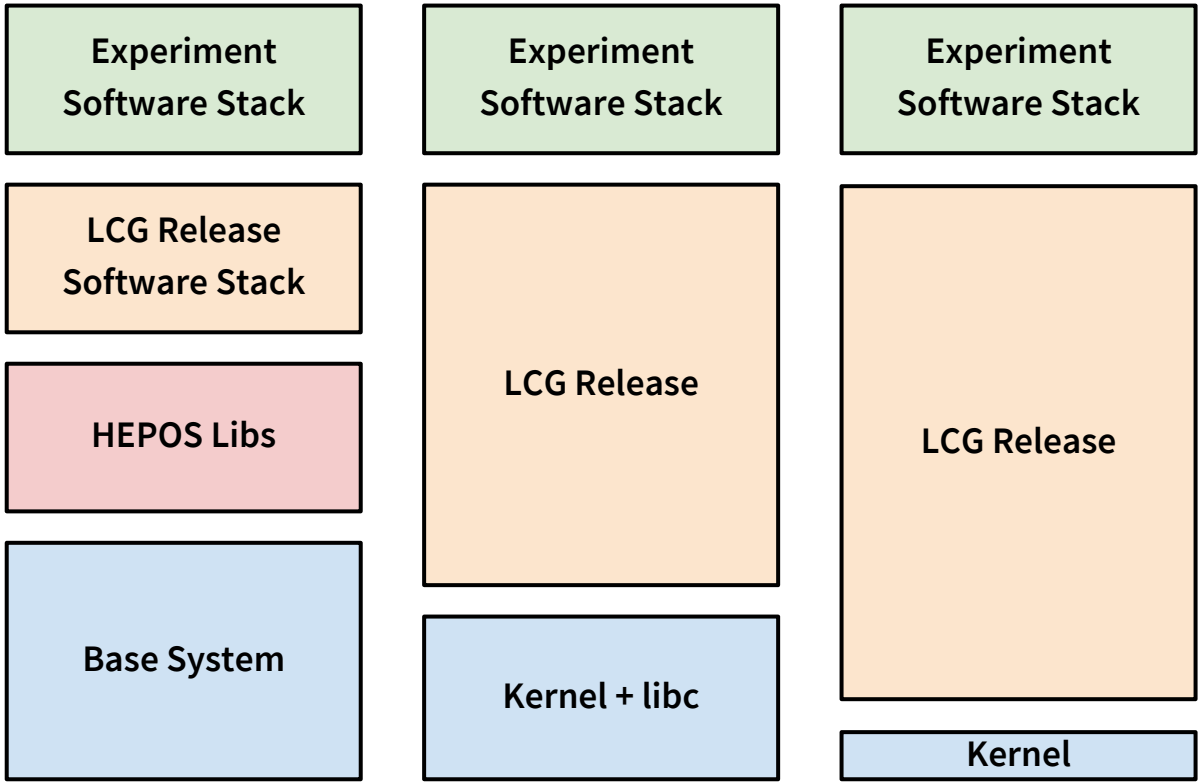
Platform Combinatorics

Currently, the LCG stack relies on libraries installed on the base system and on the HEPOS libs meta-package.

Since each Linux distribution uses its own version of libc, binutils, GCC, and includes a different set of system packages, the HEPOS libs meta-package contains different packages for different distros.

For that reason, LCG must also be tuned to provide what's missing in these base layers.

Building everything from the ground up would make it possible to reduce platform combinatorics.



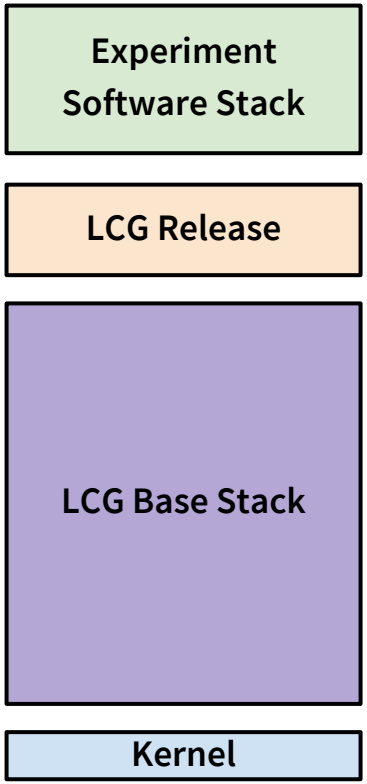
Alternatives to LD_LIBRARY_PATH (2)

- Add paths to `/etc/ld.so.cache`
- Problem: requires root access, modifies original system
- Solution: install everything into a non-standard prefix
- Requires patching glibc, binutils, gcc
- Bigger problem? No! Can rely on work already done
- Gentoo Prefix — https://wiki.gentoo.org/wiki/Project:Prefix
- Single stack works on any Linux distribution

```
$ /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/startprefix
Entering Gentoo Prefix /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux
$ which root.exe
/cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/usr/lib64/root/6.14/bin/root.exe
$ ldd $(which root.exe)
      linux-vdso.so.1
      libX11.so.6 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/usr/lib64/libX11.so.6
      libXext.so.6 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/usr/lib64/libXext.so.6
      libXpm.so.4 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/usr/lib64/libXpm.so.4
      libstdc++.so.6 =>
         /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/usr/x86_64-pc-linux-gnu/lib/gcc/libstdc++.so.6
      libgcc_s.so.1 =>
         /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/usr/x86_64-pc-linux-gnu/lib/gcc/libgcc_s.so.1
      libpthread.so.0 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/lib64/libpthread.so.0
      libc.so.6 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/lib64/libc.so.6
      libxcb.so.1 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/usr/lib64/libxcb.so.1
      libdl.so.2 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/lib64/libdl.so.2
      libm.so.6 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/lib64/libm.so.6
      libXau.so.6 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/usr/lib64/libXau.so.6
      libXdmcp.so.6 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/usr/lib64/libXdmcp.so.6
      libbsd.so.0 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/usr/lib64/libbsd.so.0
      librt.so.1 => /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/lib64/librt.so.1
      /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/lib64/ld-linux-x86-64.so.2
```

A Solid Base for LCG Releases with Gentoo Prefix

- Replace “system” + HEPOS libs with a Gentoo Prefix stack
- Build LCG releases on top of it, same way as currently done
- Cut down on number of stacks (1 Linux + 1 Mac + …)
- Can absorb a lot more software into base stack,
  let LCG focus on HEP software
- Overcome limitation of 10 year old operating systems
- For more information, please see presentations below
  - Providing an LTS distro with Gentoo Prefix, FOSDEM 2015
  - Unix? Windows? Gentoo! Native Portability to the max!, FOSDEM 2018



Try it out on your machine, only CVMFS needed

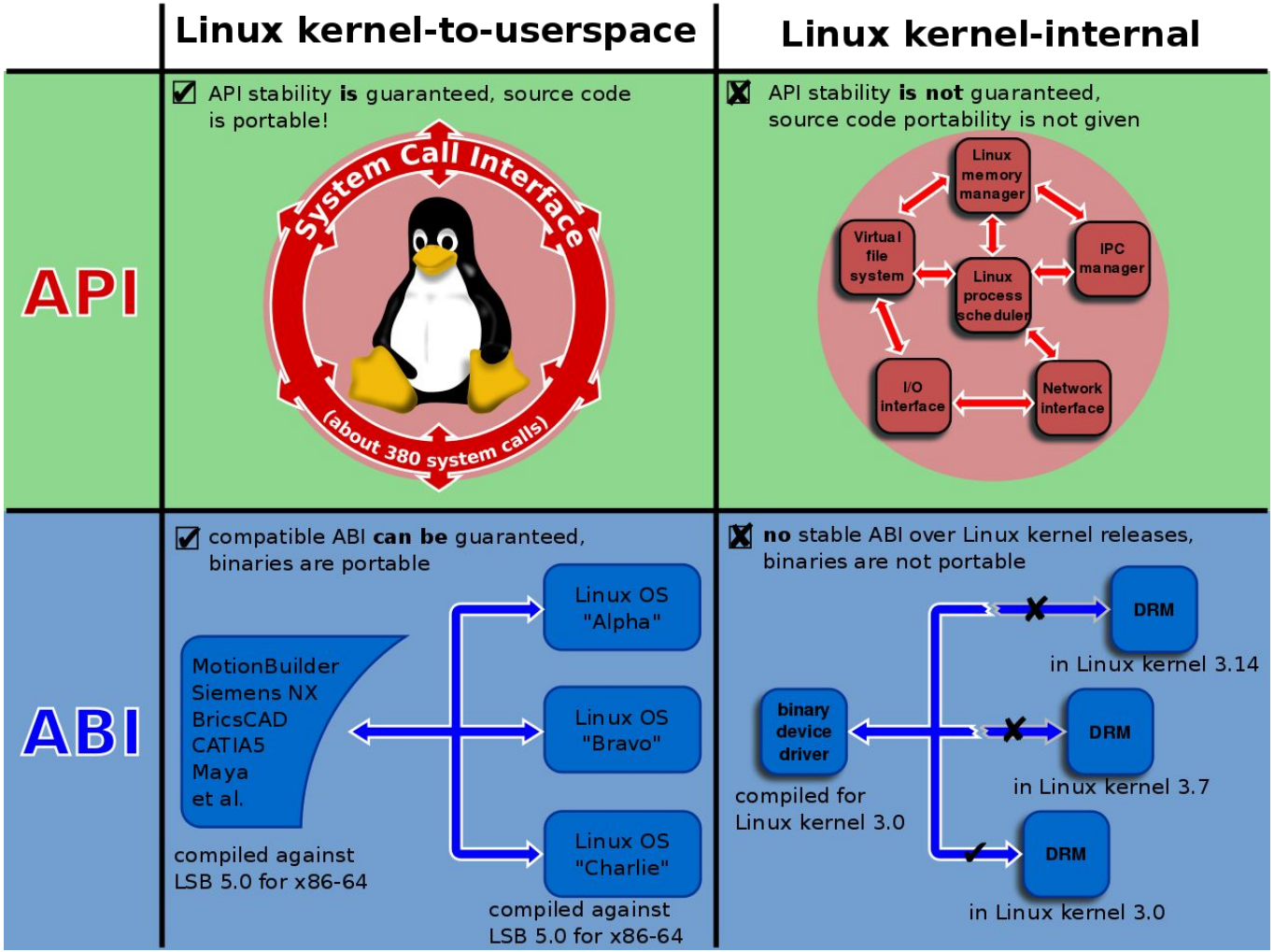
```
No setup needed! Just run software by typing full path

Start ROOT 6.14/00 directly from the prefix
$ /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/usr/lib64/root/6.14/bin/root

Optionally, can start a shell to preferentially use software from prefix

Gentoo prefix for Linux (any distribution) via CVMFS
$ /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/startprefix

Gentoo prefix for MacOS via CVMFS (older, less packages)
$ /cvmfs/sft.cern.ch/lcg/contrib/gentoo/macos/startprefix

Starting a busybox Docker container (on a machine with CVMFS installed)
$ docker run -it -v /cvmfs:/cvmfs busybox \
    /cvmfs/sft.cern.ch/lcg/contrib/gentoo/linux/bin/bash -l
```

Backup Slides

How does the linker load dynamic libraries?

From the manual page of ld.so:
If a shared object dependency does not contain a slash, then it is searched for in the following order:

- Using the directories specified in the **DT_RPATH** dynamic section attribute of the binary if present and **DT_RUNPATH** attribute does not exist. Use of **DT_RPATH** is deprecated.
- Using the environment variable **LD_LIBRARY_PATH**, unless the executable is being run in secure-execution mode (see below), in which case this variable is ignored.
- Using the directories specified in the **DT_RUNPATH** dynamic section attribute of the binary if present. Such directories are searched only to find those objects required by **DT_NEEDED** (direct dependencies) entries and do not apply to those objects' children, which must themselves have their own **DT_RUNPATH** entries. This is unlike **DT_RPATH**, which is applied to searches for all children in the dependency tree.
- From the cache file **/etc/ld.so.cache**, which contains a compiled list of candidate shared objects previously found in the augmented library path. If, however, the binary was linked with the -z nodeflib linker option, shared objects in the default paths are skipped. Shared objects installed in hardware capability directories (see below) are preferred to other shared objects.
- In the default path **/lib**, and then **/usr/lib**. (On some 64-bit architectures, the default paths for 64-bit shared objects are **/lib64**, and then **/usr/lib64**.) If the binary was linked with the -z nodeflib linker option, this step is skipped.


Linux kernel-to-userspace
Linux kernel-internal
API
☑ API stability is guaranteed, source code is portable!
System Call Interface
(about 380 system calls)
☒ API stability is not guaranteed, source code portability is not given
Linux memory manager
Virtual file system
IPC manager
Linux process scheduler
I/O interface
Network interface
ABI
☑ compatible ABI can be guaranteed, binaries are portable
MotionBuilder
Siemens NX
BricsCAD
CATIA5
Maya
et al.
compiled against LSB 5.0 for x86-64
Linux OS "Alpha"
Linux OS "Bravo"
Linux OS "Charlie"
compiled against LSB 5.0 for x86-64
☒ no stable ABI over Linux kernel releases, binaries are not portable
binary device driver
compiled for Linux kernel 3.0
DRM
in Linux kernel 3.14
DRM
in Linux kernel 3.7
DRM
in Linux kernel 3.0

$

$