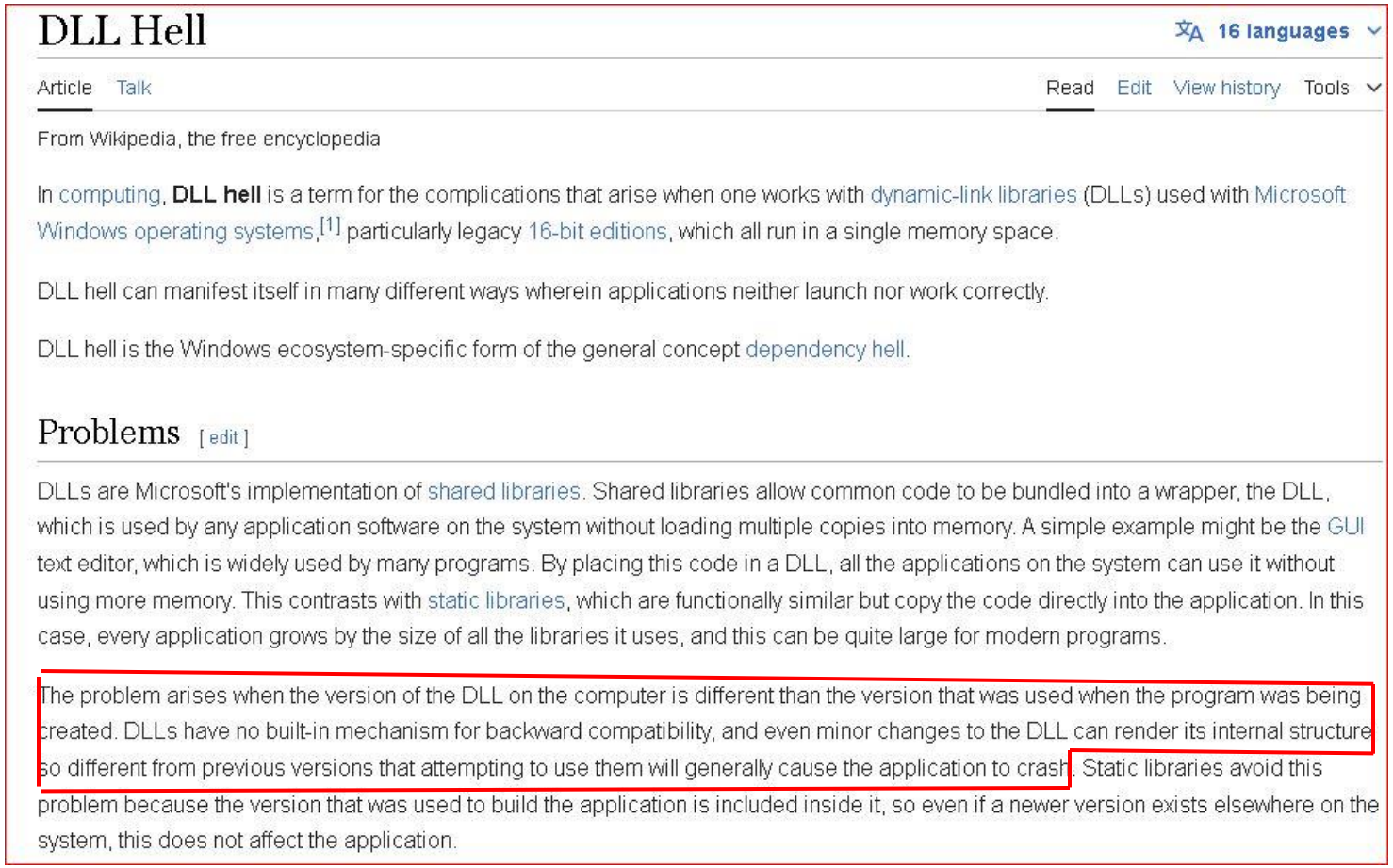
How are phase2 machines setup and why?

What is DLL hell?



DLL Hell

16 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

In computing, **DLL hell** is a term for the complications that arise when one works with dynamic-link libraries (DLLs) used with Microsoft Windows operating systems,[1] particularly legacy 16-bit editions, which all run in a single memory space.

DLL hell can manifest itself in many different ways wherein applications neither launch nor work correctly.

DLL hell is the Windows ecosystem-specific form of the general concept dependency hell.

Problems [edit]

DLLs are Microsoft's implementation of shared libraries. Shared libraries allow common code to be bundled into a wrapper, the DLL, which is used by any application software on the system without loading multiple copies into memory. A simple example might be the GUI text editor, which is widely used by many programs. By placing this code in a DLL, all the applications on the system can use it without using more memory. This contrasts with static libraries, which are functionally similar but copy the code directly into the application. In this case, every application grows by the size of all the libraries it uses, and this can be quite large for modern programs.

The problem arises when the version of the DLL on the computer is different than the version that was used when the program was being created. DLLs have no built-in mechanism for backward compatibility, and even minor changes to the DLL can render its internal structure so different from previous versions that attempting to use them will generally cause the application to crash. Static libraries avoid this problem because the version that was used to build the application is included inside it, so even if a newer version exists elsewhere on the system, this does not affect the application.

Dynamic
Link
Libraries
is the windows equivalent of shared libraries (.so) in linux

Comments on DLL hell -1

- Why the problem appeared on the 90s?
  - It was the first time that people got access to lots of precompiled software
- Why linux had not this problem?
  - Software in linux (unix) was delivered in source code form. If you always compile all the code in your machine you are essentially not having a problem.
- Why linux might be having this problem?
  - Because we use precompiled software packages for all of our dependencies
- How to avoid the problem?
  - Use only a single linux distribution and software from the official repositories of your distribution
  - Use a predefined self-consistent collection of all the software packages that you might need

Comments on DLL hell -2

- How is LHC offline software avoiding this problem?
  - LHC experiments have configuration steps that impose the correct paths for finding libraries
  - They are using a central repository for all their software that is needed before the experiment specific software

How are LHC era computers being setup?
-before each experiment sets up its own software

- Use only the agreed upon distribution
  - Currently Alma9
- Make sure everyone has installed the same packages from the baseline installation
  - by installing HEP_OSlibs metapackage

What is HEP_Oslibs?

HEP_OSlibs meta-package

This is the main repository and documentation page for HEP_OSlibs.

HEP_OSlibs is a meta-package that captures the Linux operating system (O/S) build- and run-time dependencies of the software of the four LHC experiments.

HEP_OSlibs is a **pure meta-package that contains no software.** Installing it simply pulls in the packages it depends on, as well as any other packages on which these in turn depend.

```
11  Requires: attr(%{__isa})
12  Requires: autoconf
13  Requires: automake
14  Requires: bzip2(%{__isa})
15  Requires: bzip2-devel(%{__isa})
16  #Requires: ccache(%{__isa}) # EPEL
17  Requires: cmake(%{__isa})
18  Requires: cyrus-sasl-devel(%{__isa})
19  Requires: elfutils-debuginfod-client(%{__isa})
20  Requires: expat-devel(%{__isa})
21  Requires: file(%{__isa})
22  Requires: gcc(%{__isa})
23  Requires: gcc-c++(%{__isa})
24  Requires: gcc-gfortran(%{__isa})
25  Requires: gdb(%{__isa})
26  Requires: gdbm-devel(%{__isa})
27  Requires: git(%{__isa})
28  Requires: glibc-devel(%{__isa})
29  Requires: gmp-devel(%{__isa})
30  Requires: jq(%{__isa})
```

```
114 [Requestor: Marco Clemencic (LHCb)]
115 Add libzstd-devel as requested by LHCb (#15).
116 Require 66 packages on x86_64 and aarch64 (62 x86-64/aarch-64, 4 noarch).
117
118 * Mon Mar 11 2024 Andrea Valassi 9.1.1-6 (x86_64 and aarch64)
119 [Requestor: Maarten Litmaath (WLCG)]
120 Add openldap-compat, which was reported by ATLAS as a missing dependency of GFAL (#14).
121 Require 65 packages on x86_64 and aarch64 (61 x86-64/aarch-64, 4 noarch).
122
123 * Mon Mar 11 2024 Andrea Valassi 9.1.1-4 (x86_64 and aarch64)
124 [Requestor: Attila Krasznahorkay (ATLAS)]
125 Add expat-devel, which is needed in the ATLAS builds of Geant4 (#12 and SPI-2396).
126 Require 64 packages on x86_64 and aarch64 (60 x86-64/aarch-64, 4 noarch).
127
128 * Mon Mar 11 2024 Andrea Valassi 9.1.1-3 (x86_64 and aarch64)
129 [Requestor: Andre Sailer (SPI)]
130 Add elfutils-debuginfod-client, which is needed to build R in the LCG stack (#11 and INC3609512)
131 Require 63 packages on x86_64 and aarch64 (59 x86-64/aarch-64, 4 noarch).
132
```

How are LHC era computers being setup?
Before each experiment sets up its own software

- Use only the agreed upon distribution
  - Currently Alma9
- Make sure everyone has installed the same packages from the baseline installation
  - by installing HEP_OSlibs metapackage
- Add a self-consistent repository of all external packages needed by LHC experiments, over the network
  - Add the cvmfs service to your system
  - Load the LCG release that you like

What is an LCG release?

Welcome to the LCG Releases provided by the SPI team in EP-SFT at CERN.

In the CVMFS repository `/cvmfs/sft.cern.ch` you can find a software stack containing over 450 external packages as well as HEP specific tools and generators. There are usually two releases per year as well as development builds every night.

The releases start with the prefix `LCG_` followed by the major version of the release, e.g. `101`. A major release implies major version changes in all packages of the software stack. For patches, we append lowercase letters like `a`, `b`, ... to the name of the release.

acts	None	None	26.0.0	
acts_core	Simulation	None	0.10.05	
agile	Generator	C++	1.5.0	• Sacrifice
AIDA	Math	C++	3.2.1	• Lorenzo
aiohttp	None	None	3.9.5	
aiosignal	None	None	1.2.0	
aiostream	None	None	0.4.5	
alabaster	None	None	0.7.12	
alembic	None	None	1.13.3	
alpaka	Other	C++	0.9.0	
alpgen	Generator	Fortran	2.1.4	• Michelar
altair	None	None	5.2.0	

x86_64-el9-clang16-dbg

x86_64-el9-clang16-opt

x86_64-el9-gcc11-opt

x86_64-el9-gcc12-dbg

x86_64-el9-gcc12-opt

x86_64-el9-gcc13-dbg

x86_64-el9-gcc13-opt

x86_64-el9-gcc14fp-opt

x86_64-el9-gcc14-opt

Q&A

- Q1:Online machines do have more low level needs
  - Kernel modules to map VME memory space to pc memory
  - Kernel modules to arrange the memory regions according to needs
  - Special drivers for networking
  - ……
- A1: hopefully you know more than what this presentation talks about
- Q2:I have dependencies on software that is not part of the repository
- A2: you do have to compile from scratch but the real question is why you depend on software that no one else on LHC experiments depends upon.