Common Package Specification (CPS) in Practice:

A Full Round Trip Implementation in Conan C++ Package Manager

DIEGO RODRIGUEZ-LOSADA GONZALEZ

JFrog

CONAN 2.0

C/C++ Package Manager

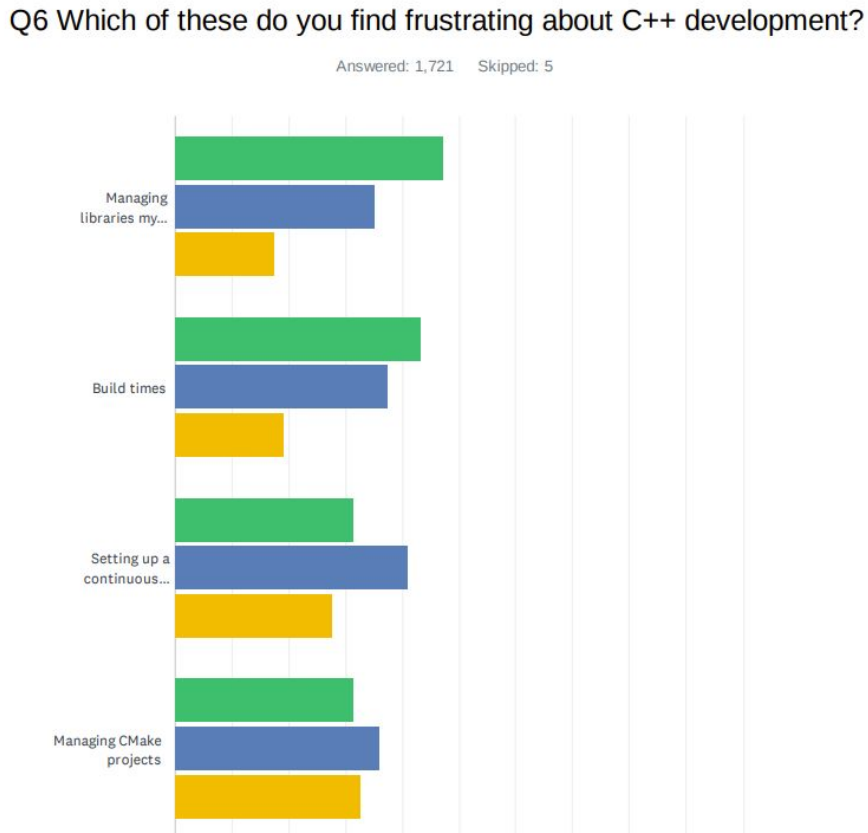
Outline

- **Introduction to Common Package Specification (CPS)**
- Creation of CPS files from existing Conan packages
- Loading CPS files generated by build systems
- Generating build system native files from CPS
- Location of CPS files
- Lessons learned and conclusions

Why a Common Package Specification (CPS)

C/C++ projects top 4 pains:

- Managing libraries
- Build times
- Setting CI
- Managing CMake projects

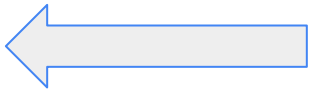


*ISO C++ survey 2024

Common Package Specification (CPS)

...

zlib.cps



ZLib

```
├──include
│       zconf.h
│       zlib.h
│
├──lib
│       zlib.lib
```

Existing solutions

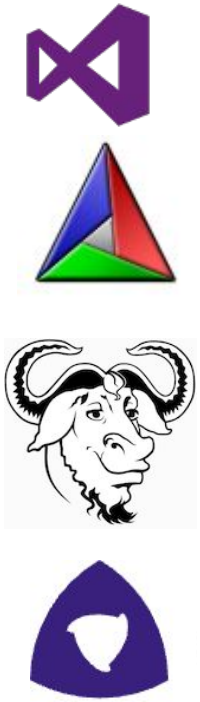
```
prefix=@CMAKE_INSTALL_PREFIX@
exec_prefix=@CMAKE_INSTALL_PREFIX@
libdir=@INSTALL_LIB_DIR@
sharedlibdir=@INSTALL_LIB_DIR@
includedir=@INSTALL_INC_DIR@

Name: zlib
Description: zlib compression library
Version: @VERSION@

Requires:
Libs: -L${libdir} -L${sharedlibdir} -lz
Cflags: -I${includedir}
```

```
set(_ZLIB_x86 "(x86)")
set(_ZLIB_SEARCH_NORMAL PATHS
"[HKEY_LOCAL_MACHINE\\SOFTWARE\\GnuWin32\\Zlib;InstallPath]"
list(APPEND _ZLIB_SEARCHES _ZLIB_SEARCH_NORMAL)

if(ZLIB_USE_STATIC_LIBS)
  set(ZLIB_NAMES zlibstatic zlibstat zlib z)
  set(ZLIB_NAMES_DEBUG zlibstaticd zlibstatd zlibd zd)
else()
  set(ZLIB_NAMES z zlib zdll zlib1 zlibstatic zlibwapi ..)
  set(ZLIB_NAMES_DEBUG zd zlibd zdlld zlibd1 zlib1d ..)
endif()

if(ZLIB_FOUND)
    set(ZLIB_INCLUDE_DIRS ${ZLIB_INCLUDE_DIR})

    if(NOT TARGET ZLIB::ZLIB)
      add_library(ZLIB::ZLIB UNKNOWN IMPORTED)
      set_target_properties(ZLIB::ZLIB PROPERTIES
                            INTERFACE_INCLUDE_DIRECTORIES
                           "${ZLIB_INCLUDE_DIRS}")
```

Searching for Convergence in C++ Package Management - Bret Brown & Daniel Ruoso - CppNow 2022

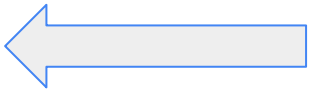
Case For a Standardized Package Description Format for External C++ Libraries by Luis Caro Campos - CppCon22

Interoperability



...

Read CPS

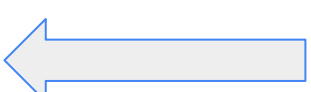


ZLib

```
├──include
│       zconf.h
│       zlib.h
│
├──lib
│       zlib.lib
```

zlib.cps

Creates CPS



...

Previous and ongoing work

- Other previous:
  - ISO C++: https://github.com/isocpp/pkg-fmt by Rene Rivera
  - Libman by Colby Pike (@vectorofbool)

C++ Ecosystem Evolution group

CPS: https://github.com/cps-org/cps

- Rooted in ISO paper 2018 p1313r0.html

Libraries: A First Step Toward Standard C++ Dependency Management - Bret Brown and Bill Hoffman- CppCon2023

A Common Package Specification: Getting Build Tools to Talk to Each Other - Diego Rodriguez-Losada - CppCon2023

Example CPS

zlib.cps

```
{
    "cps_version": "0.12.0",
    "name": "zlib",
    "version": "1.3.1",
    "configurations": ["release"],
    "default_components": ["zlib"],
    "components": {
        "zlib": {
            "type": "archive",
            "includes": ["@prefix@/include"],
            "location": "@prefix@/lib/libz.a"
        }
    }
}
```

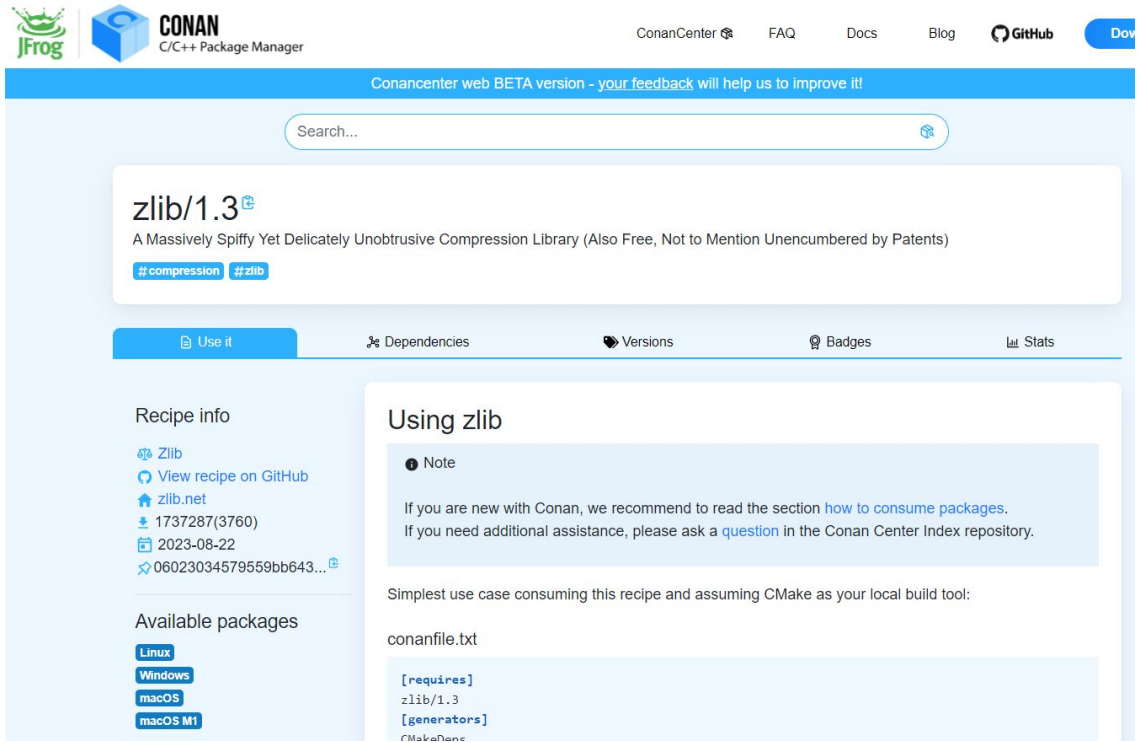
CPS files and where to find them

- Goals of the talk:
  - Provide implementation experience
  - Provide tools for testing
  - Discuss possible gaps and future work
  - Lessons learned
  - Push the collaboration
- Non goals:
  - Explain the specification

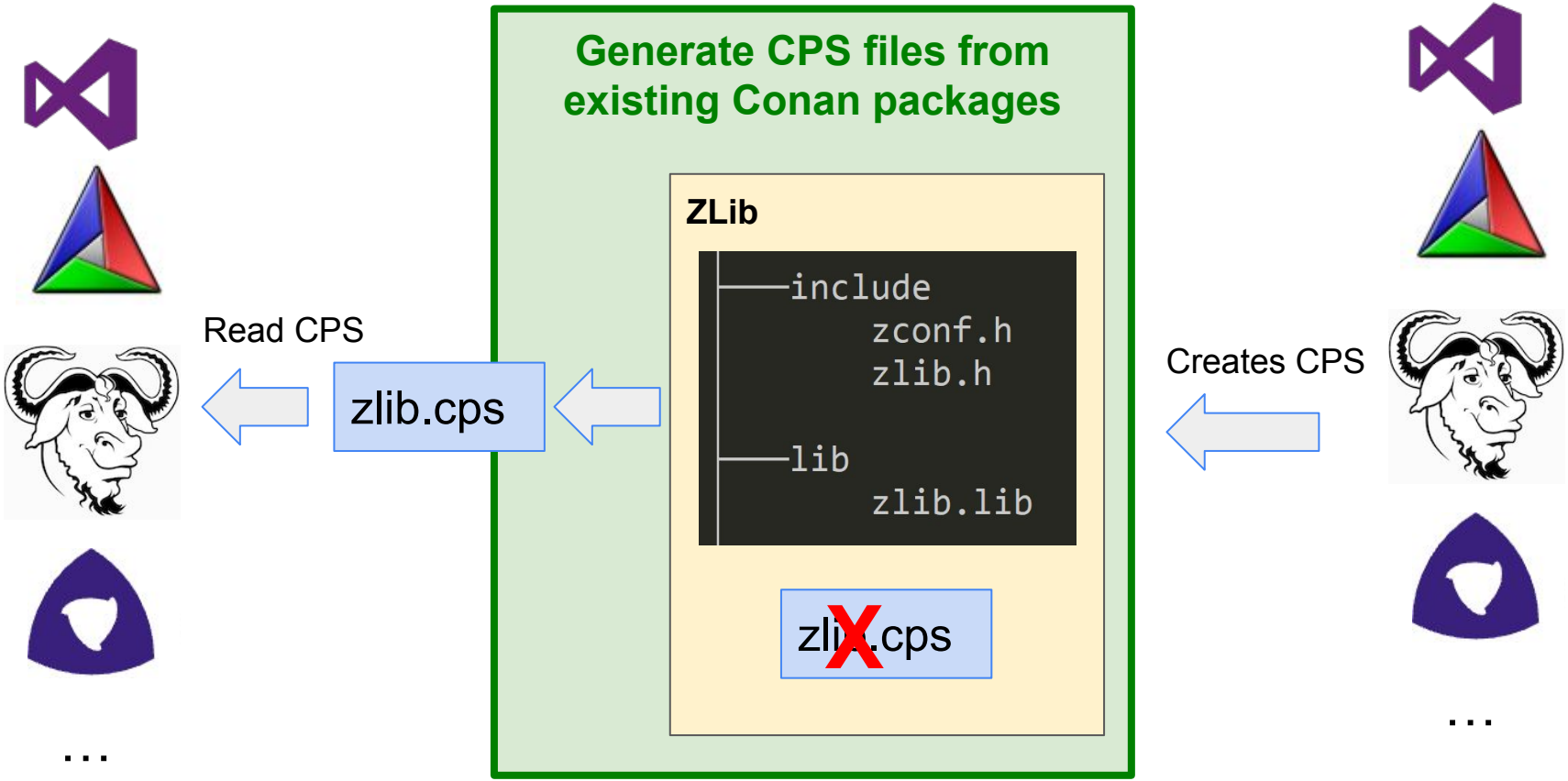
Outline

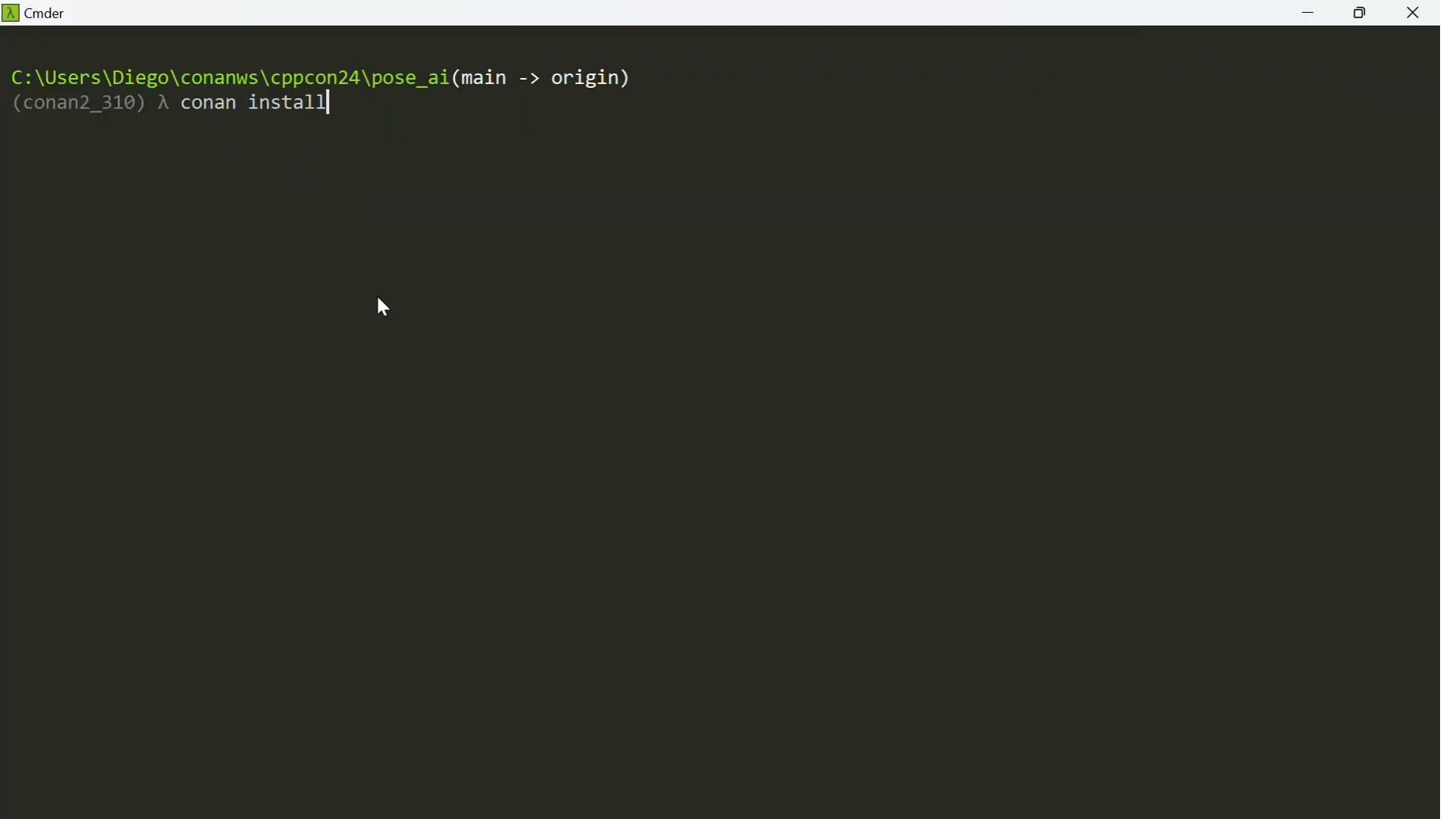
- Introduction to Common Package Specification (CPS)
- **Creation of CPS files from existing Conan packages**
- Loading CPS files generated by build systems
- Generating build system native files from CPS
- Location of CPS files
- Lessons learned and conclusions

ConanCenter



Demo: Generate CPS files

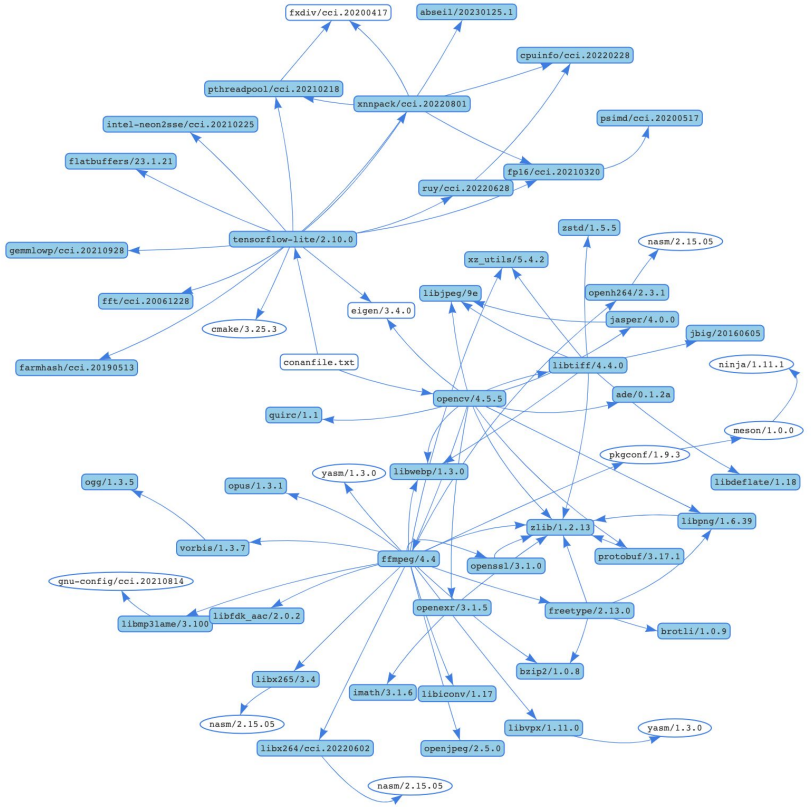



Cmder
C:\Users\Diego\conanws\cppcon24\pose_ai(main -> origin)
(conan2_310) λ conan install

Demo: Generate CPS files

cppcon24/pose_ai

```
$ conan install . -g CPSDeps
```



CPSDeps

```
from conan.cps import CPS

class CPSDeps:

    def generate(self):
        ...
        mapping = {}
        for dep in self._conanfile.dependencies.host.values():
            cps_in_package = os.path.join(dep.package_folder, f"{dep.ref.name}.cps")
            if os.path.exists(cps_in_package):
                mapping[dep.ref.name] = cps_in_package
                continue

            cps = CPS.from_conan(dep)
            output_file = cps.save(folder)
            mapping[dep.ref.name] = output_file
```

CPS Python library

```python
from conan.cps import CPS

class CPSComponentType(Enum):
    DYLIB = "dylib"
    ARCHIVE = "archive"
    INTERFACE = "interface"
    EXE = "exe"
    JAR = "jar"


class CPSComponent:
    """ Each component within a CPS file """

class CPS:
    """ represents the CPS file for 1 package """
    @staticmethod
    def from_conan(dep):
        """ Computes CPS from conanfile.py info """
```

Conan “conanfile.py” recipes

conanfile.py

```
class ZLibConan(ConanFile):
    name = "zlib"
    version = "1.2.13"
    url = "https://github.com/conan-io/conan-center-index"
    homepage = "https://zlib.net"
    license = "Zlib"
    description = "A Massively Spiffy Yet Delicately Unobtrusive …"
    settings = "os", "compiler", "build_type", "arch"

    def build(self):
        cmake = CMake(self)
        cmake.configure()
        cmake.build()

    def package(self):
        cmake = CMake(self)
        cmake.install()

    def package_info(self):
        self.cpp_info.libs = ["zlib"]  # not real!!
```

Conan “conanfile.py” recipes

conanfile.py

```
class ZLibConan(ConanFile):
    …

    def build(self):
        ...

    def package(self):
        ...

    def package_info(self):
        self.cpp_info.set_property("cmake_find_mode", "both")
        self.cpp_info.set_property("cmake_file_name", "ZLIB")
        self.cpp_info.set_property("cmake_target_name", "ZLIB::ZLIB")
        # self.cpp_info.includedirs = ["include"] # default
        # self.cpp_info.libdirs = ["lib"] # default
        if self.settings.os == "Windows" and not self._is_mingw:
            libname = "zdll" if self.options.shared else "zlib"
        else:
            libname = "z"
        self.cpp_info.libs = [libname]
```

zlib.cps (Windows/msvc)

```
|--include
|    zconf.h
|    zlib.h
|
|--lib
|    zlib.lib
|
|--licenses
     LICENSE
```

```
{
  "cps_version": "0.12.0",
  "name": "zlib",
  "version": "1.3.1",
  "license": "Zlib",
  "description": "A Massively Spiffy Yet Delicately Unobtrusive …",
  "website": "https://zlib.net",
  "configurations": ["release"],
  "default_components": ["zlib"],
  "components": {
    "zlib": {
      "type": "archive",
      "includes": ["C:/Users/Diego/.conan2/p/zlib6f797a4dd16fb/p/include"],
      "location": "C:/Users/Diego/.conan2/p/zlib6f797a4dd16fb/p/lib/zlib.lib"
    }
  }
}
```

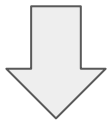
zlib.cps (Linux/gcc)

```
|--include
|    zconf.h
|    zlib.h
|
|--lib
|    libz.a
|
|--licenses
     LICENSE
```

```
{
  "cps_version": "0.12.0",
  "name": "zlib",
  "version": "1.3.1",
  "license": "Zlib",
  "description": "A Massively Spiffy Yet Delicately Unobtrusive …",
  "website": "https://zlib.net",
  "configurations": ["release"],
  "default_components": ["zlib"],
  "components": {
    "zlib": {
      "type": "archive",
      "includes": ["C:/Users/Diego/.conan2/p/zlibcdd5270f9ab9f/p/include"],
      "location": "C:/Users/Diego/.conan2/p/zlibcdd5270f9ab9f/p/lib/libz.a"
    }
  }
}
```

Filling the gaps

```
name = "zlib"
version = "1.3.1"

def package_info(self):
    if self.settings.os == "Windows" and not self._is_mingw:
        libname = "zdll" if self.options.shared else "zlib"
    else:
        libname = "z"
    self.cpp_info.libs = [libname]
```

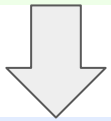


conan.cps.deduce_full_lib() ~
CMake find_library()

```
"components": {
  "zlib": {
    "type": "archive",
    "location": "C:/Users/Diego/.conan2/p/zlib6f797a4dd16fb/p/lib/zlib.lib"
   }
}
```

Explicitly modeling the gaps

```
name = "zlib"
version = "1.3.1"

def package_info(self):
    self.cpp_info.libs = {
        "mylib": {
            "location": "my_custom_location/mylib.a",
            "type": "static-library"
        }
    }
```

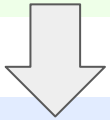


Direct mapping

```
"components": {
  "zlib": {
    "type": "archive",
    "location": "my_custom_location/mylib.a"
   }
}
```

libiconv

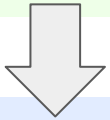
```python
def package_info(self):
    self.cpp_info.libs = ["iconv", "charset"]
```



```
  "name": "libiconv",
  "version": "1.17",
  "configurations": ["release"],
  "default_components": ["iconv", "charset"],
  "components": {
    "iconv": {
      "type": "archive",
      "includes": ["C:/Users/Diego/.conan2/p/libicb4e788959979c/p/include"]
      "location": "C:/Users/Diego/.conan2/p/libicb4e788959979c/p/lib/iconv.lib"
    },
    "charset": {
      "type": "archive",
      "includes": ["C:/Users/Diego/.conan2/p/libicb4e788959979c/p/include"]
      "location": "C:/Users/Diego/.conan2/p/libicb4e788959979c/p/lib/charset.lib"
    }
  }
```

libiconv

```
def package_info(self):
    self.cpp_info.libs = ["iconv", "charset"]
```

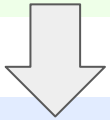


```
"name": "libiconv",
"version": "1.17",
"configurations": ["release"],
"default_components": ["iconv", "charset"],
"components": {
  "_libiconv": {
    "type": "interface",
    "includes": ["C:/Users/Diego/.conan2/p/libicb4e788959979c/p/include"]
  },
  "iconv": {
    "type": "archive",
    "requires": [":_libiconv"],
    "location": "C:/Users/Diego/.conan2/p/libicb4e788959979c/p/lib/iconv.lib"
  },
  "charset": {
    "type": "archive",
    "requires": [":_libiconv"],
    "location": "C:/Users/Diego/.conan2/p/libicb4e788959979c/p/lib/charset.lib"
  }
}
```

private

libiconv

```
def package_info(self):
    self.cpp_info.libs = ["iconv", "charset"]
```

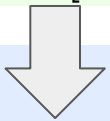


```
"name": "libiconv",
"version": "1.17",
"configurations": ["release"],
"default_components": ["iconv", "charset"],
"components": {
  "_libiconv": {
    "type": "interface",
    "includes": ["C:/Users/Diego/.conan2/p/libicb4e788959979c/p/include"]
  },
  "iconv": {
    "type": "archive",
    "requires": [":_libiconv", ":charset"],
    "location": "C:/Users/Diego/.conan2/p/libicb4e788959979c/p/lib/iconv.lib"
  },
  "charset": {
    "type": "archive",
    "requires": [":_libiconv"],
    "location": "C:/Users/Diego/.conan2/p/libicb4e788959979c/p/lib/charset.lib"
  }
}
```

Likely missing if requiring only iconv component

libiconv (solution)

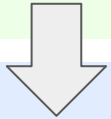
```python
def package_info(self):
    self.cpp_info.components["charset"].libs = ["charset"]
    self.cpp_info.components["iconv"].libs = ["iconv"]
    self.cpp_info.components["iconv"].requires = ["charset"]
```




```
  "name": "libiconv",
  "version": "1.17",
  "configurations": ["release"],
  "default_components": ["iconv", "charset"],
  "components": {
    "_libiconv": {
      "type": "interface",
      "includes": ["C:/Users/Diego/.conan2/p/libicb4e788959979c/p/include"]
    },
    "iconv": {
      "type": "archive",
      "requires": [":_libiconv", ":charset"],
      "location": "C:/Users/Diego/.conan2/p/libicb4e788959979c/p/lib/iconv.lib"
    },
    "charset": {
      "type": "archive",
      "requires": [":_libiconv"],
      "location": "C:/Users/Diego/.conan2/p/libicb4e788959979c/p/lib/charset.lib"
    }
  }
```

flac

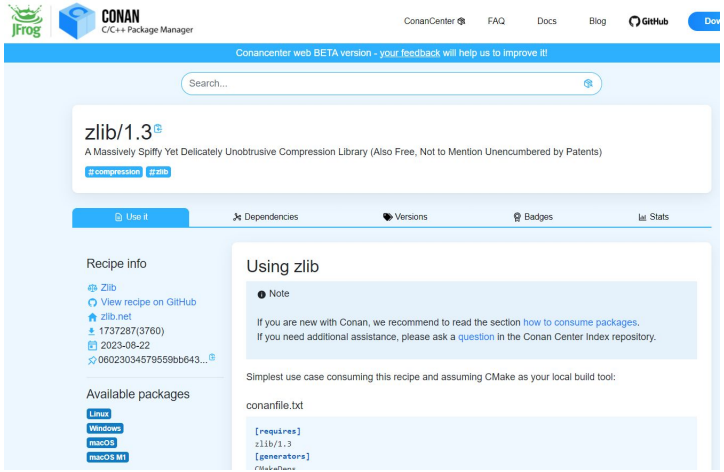
```python
def package_info(self):
    self.cpp_info.components["libflac"].libs = ["FLAC"]
    self.cpp_info.components["libflac"].requires = ["ogg::ogg"]

    self.cpp_info.components["libflac++"].libs = ["FLAC++"]
    self.cpp_info.components["libflac++"].requires = ["libflac"]
    if not self.options.shared:
        self.cpp_info.components["libflac"].defines = ["FLAC__NO_DLL"]
        if self.settings.os in ["Linux", "FreeBSD"]:
            self.cpp_info.components["libflac"].system_libs += ["m"]
```

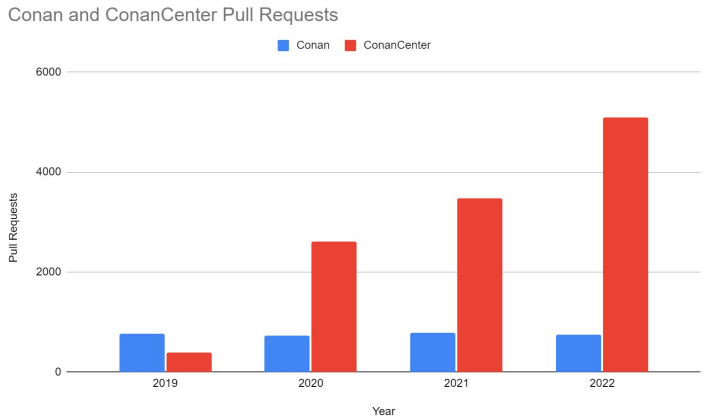


```json
"components": {
    "libflac": {
      "type": "archive",
      "requires": ["ogg:ogg"],
      "includes": ["/home/diegor/.conan2/p/flac14b90fabbd015/p/include"],
      "definitions": ["FLAC__NO_DLL"],
      "location": "/home/diegor/.conan2/p/flac14b90fabbd015/p/lib/libFLAC.a",
      "link_libraries": ["m"]
    },
    "libflac++": {
      "type": "archive",
      "requires": [":libflac"],
      "includes": ["/home/diegor/.conan2/p/flac14b90fabbd015/p/include"],
      "location": "/home/diegor/.conan2/p/flac14b90fabbd015/p/lib/libFLAC++.a"
    }
  }
```

Public and available CPS

- 1500 recipes x 3 versions x 100 binaries = **500K packages**
- Almost **6000 PRs** in **2023**
- **3,1 million** packages download/month = **16Tb/month**
- Also proprietary packages
- CMake, Meson, Autotools, etc, not only CMake

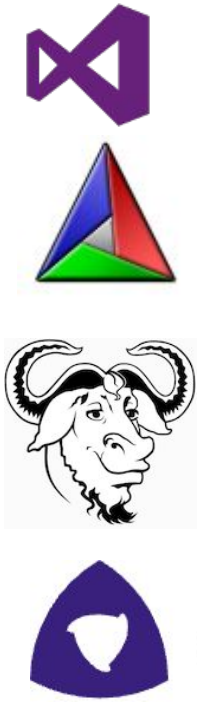




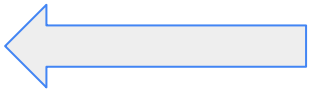
Outline

- Introduction to Common Package Specification (CPS)
- Creation of CPS files from existing Conan packages
- **Loading CPS files generated by build systems**
- Generating build system native files from CPS
- Location of CPS files
- Lessons learned and conclusions

Demo



Read CPS



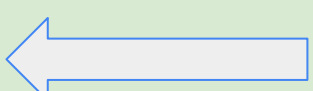
From existing CPS to Conan

ZLib

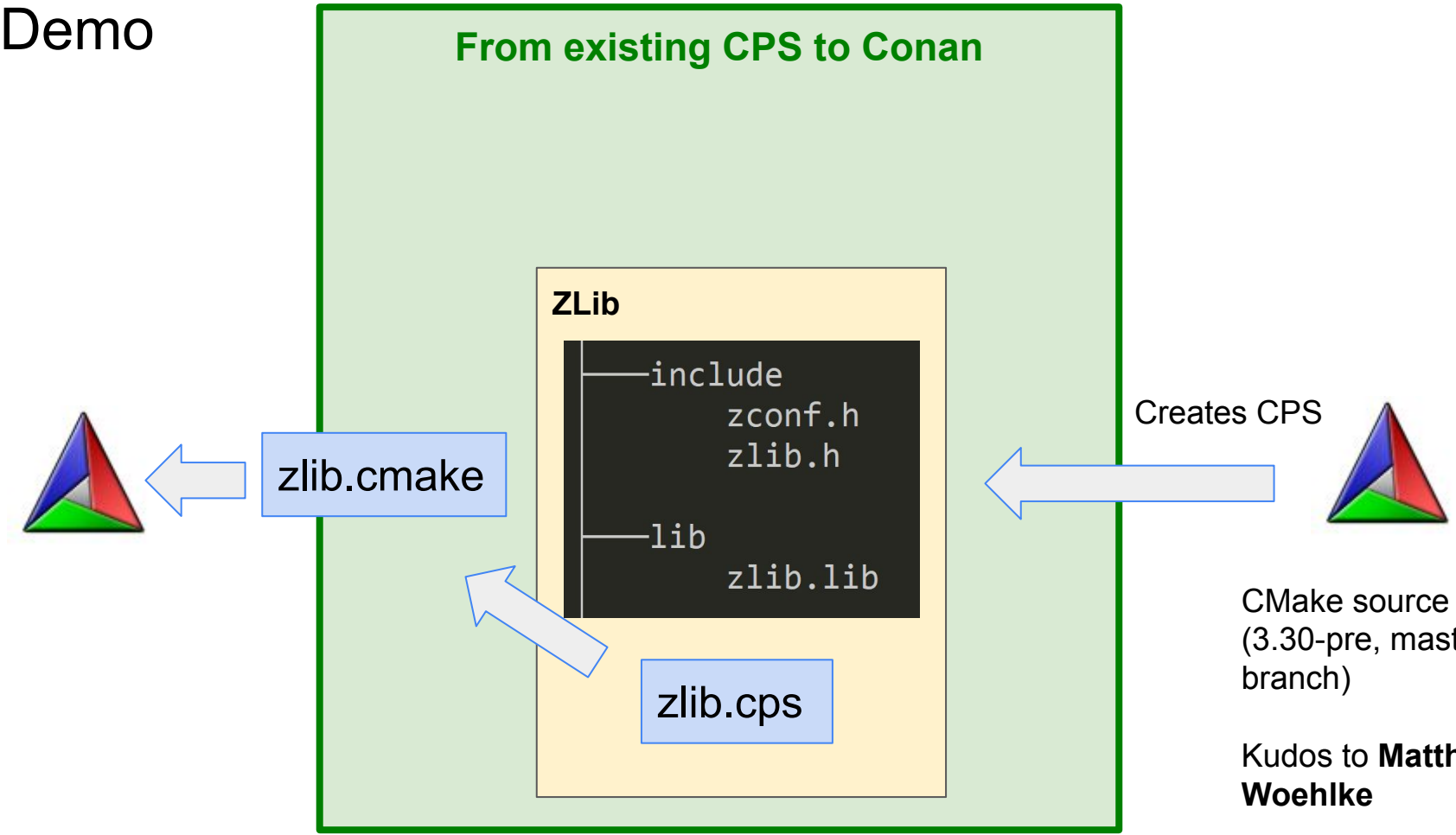
```
├──include
│       zconf.h
│       zlib.h
│
├──lib
│       zlib.lib
```

zlib.cps

Creates CPS



Demo



CMake source (3.30-pre, master branch)

Kudos to **Matthew Woehlke**

CMake CPS generation

```
cmake_minimum_required(VERSION 3.15)
project(mypkg CXX)

set(CMAKE_EXPERIMENTAL_EXPORT_PACKAGE_INFO "b80be207-778e-46ba-8080-b23bba22639e")

add_library(mypkg src/mypkg.cpp)
target_include_directories(mypkg PUBLIC
            $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
            $<INSTALL_INTERFACE:include>)

set_target_properties(mypkg PROPERTIES PUBLIC_HEADER "include/mypkg.h")
install(TARGETS mypkg EXPORT mypkg)

install(PACKAGE_INFO mypkg EXPORT mypkg)
```

Generated CPS files

mypkg.cps

```
{
  "components":
  {
    "mypkg":
    {
      "includes": ["@prefix@/include"],
      "type": "archive"
    }
  },
  "cps_path": "@prefix@/cps",
  "cps_version": "0.12.0",
  "name": "mypkg"
}
```

mypkg@release.cps

```
{
  "components":
  {
    "mypkg":
    {
      "link_languages": ["cpp"],
      "location": "@prefix@/lib/mypkg.lib"
    }
  },
  "configuration": "Release",
  "name": "mypkg"
}
```

CPS to Conan

```
    def build(self):
        cmake = CMake(self)
        cmake.configure()
        cmake.build()  # This should generate a zlib.cps

    def package(self):
        cmake = CMake(self)
        cmake.install()  # OR this should generate a zlib.cps?

    def package_info(self):
        from conan.cps import CPS
        self.cpp_info = CPS.load("cps/mypkg.cps").to_conan()
```

CPS library (cps -> conan)

```python
from conan.cps import CPS

class CPSComponentType(Enum):
    DYLIB = "dylib"
    ARCHIVE = "archive"
    INTERFACE = "interface"
    EXE = "exe"
    JAR = "jar"


class CPSComponent:
    """ Each component within a CPS file """

class CPS:
    """ represents the CPS file for 1 package """
    def to_conan():
        """ Returns a Conan cpp_info object from current CPS """
```

CPS generation by build systems (build + install)

```
def build(self):
    cmake = CMake(self)
    cmake.configure()
    cmake.build()  # This should generate a zlib.cps

def package(self):
    cmake = CMake(self)
    cmake.install()  # This should copy/adapt or generate a zlib.cps

def package_info(self):
    from conan.cps import CPS
    self.cpp_info = CPS.load("cps/mypkg.cps").to_conan()
```

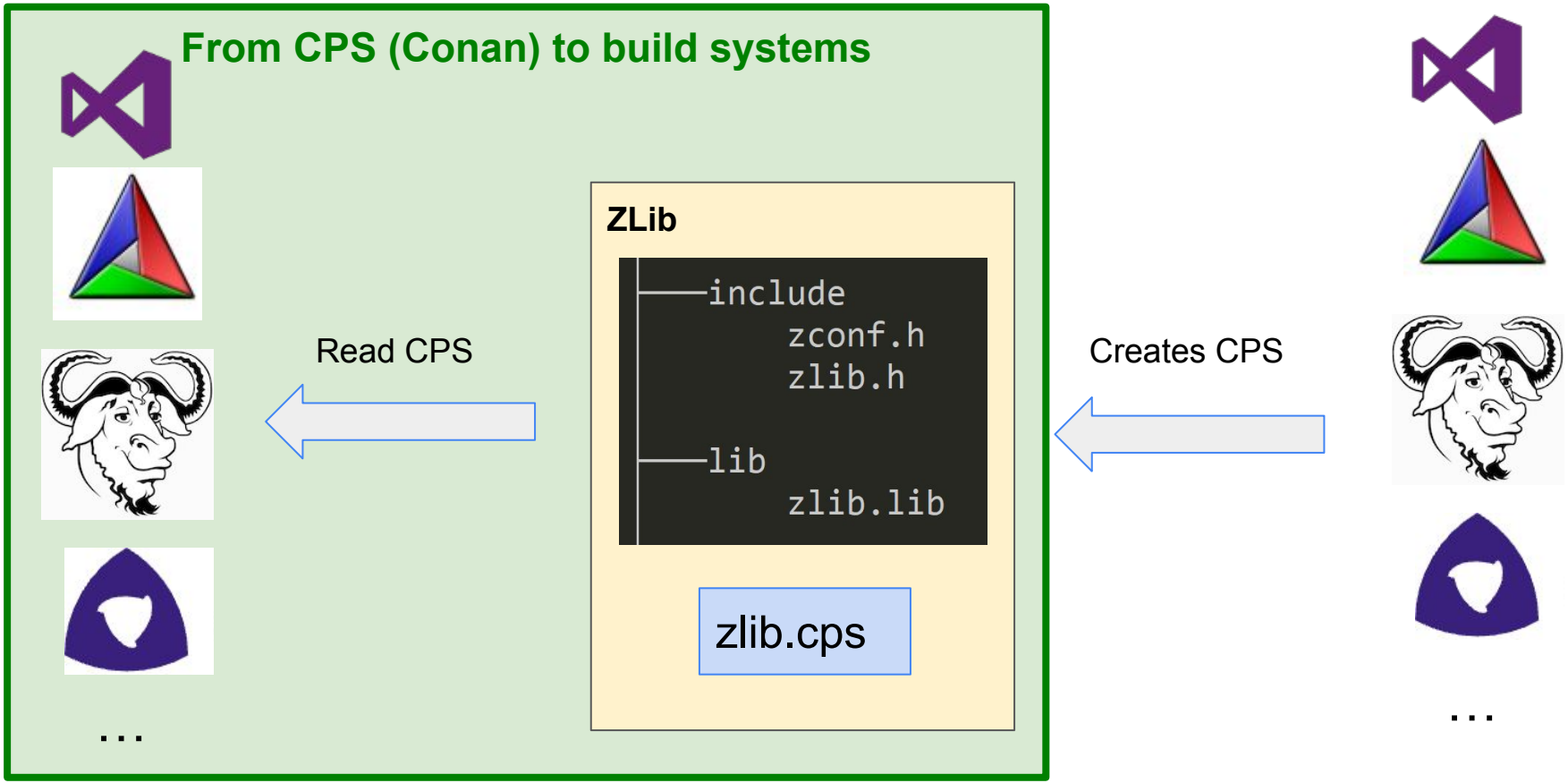
“Editable/non installed usage”

- Debugging
- Incremental build
- Meta-projects with subprojects

Outline

- Introduction to Common Package Specification (CPS)
- Creation of CPS files from existing Conan packages
- Loading CPS files generated by build systems
- **Generating build system native files from CPS**
- Location of CPS files
- Lessons learned and conclusions

Demo


From CPS (Conan) to build systems
Read CPS
ZLib
include
zconf.h
zlib.h
lib
zlib.lib
zlib.cps
Creates CPS
...
...

Build systems: CPS to CMake

zlib.cps

```
{
    "cps_version": "0.12.0",
    "name": "zlib",
    "version": "1.3.1",
    "configurations": ["release"],
    "default_components": ["zlib"],
    "components": {
        "zlib": {
            "type": "archive",
            "includes": ["@prefix@/include"],
            "location": "@prefix@/lib/zlib.lib"
        }
    }
}
```

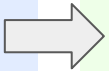
CMakeLists.txt

```
...

find_package(ZLIB REQUIRED)

...

target_link_libraries(... ZLIB::ZLIB)
```

Build systems: CPS to CMake

zlib.cps

```
"components": {
   "zlib": {
      "type": "archive",
      "includes": ["@prefix@/include"],
      "location": "@prefix@/lib/zlib.lib"
   }
}
```



ZLIBConfig.cmake

```
set(zlib_PKG_FOLDER "<full-path>")

add_library(zlib IMPORTED)

set_property(TARGET ZLIB::ZLIB PROPERTY
               INTERFACE_LINK_LIBRARIES
             zlib.lib APPEND)
set_property(TARGET ZLIB::ZLIB PROPERTY
             INTERFACE_INCLUDE_DIRECTORIES
             "${zlib_PKG_FOLDER}/include"
               APPEND)
```

Extra information not modeled yet by CPS

zlib.cps

```
"components": {
      "zlib": {
          "type": "archive",
          "includes": ["@prefix@/include"],
          "location": "@prefix@/lib/zlib.lib"
      }
  }
```

CMakeLists.txt

```
find_package(ZLIB REQUIRED)

target_link_libraries(... ZLIB::ZLIB)
```

conanfile.py

```
    def package_info(self):
        from conan.cps.cps import CPS
        self.cpp_info = CPS.load("zlib.cps").to_conan()
        self.cpp_info.set_property("cmake_file_name", "ZLIB")
        self.cpp_info.set_property("cmake_target_name", "ZLIB::ZLIB")
```

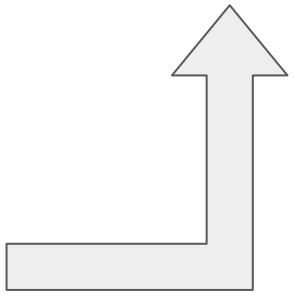
Build systems: CPS to MSBuild

```
<ConanzlibRootFolder>path</ConanzlibRootFolder>

<ClCompile>
   <AdditionalIncludeDirectories>$(ConanzlibRootFolder)/include
   </AdditionalIncludeDirectories>
</ClCompile>
<Link>
  <AdditionalLibraryDirectories>$(ConanzlibRootFolder)/lib
  </AdditionalLibraryDirectories>
  <AdditionalDependencies>zlib.lib</AdditionalDependencies>
</Link>
```

zlibs.props

zlib.cps

```
"components": {
   "zlib": {
      "type": "archive",
      "includes": ["@prefix@/include"],
      "location": "@prefix@/lib/zlib.lib"
   }
}
```



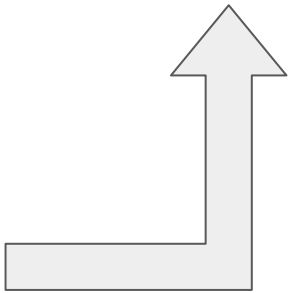
Build systems: CPS to Meson (PkgConfig)

```
prefix=C:/Users/Diego/.conan2/p/zlib6f797a4dd16fb/p
libdir=${prefix}/lib
includedir=${prefix}/include
bindir=${prefix}/bin

Name: zlib
Description: Conan package: zlib
Version: 1.3.1
Libs: -L"${libdir}" -lzlib
Cflags: -I"${includedir}"
```

zlib.cps

zlibs.pc

```
"components": {
   "zlib": {
      "type": "archive",
      "includes": ["@prefix@/include"],
      "location": "@prefix@/lib/zlib.lib"
   }
}
```

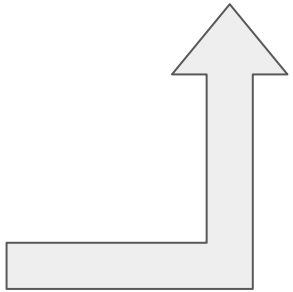


Build systems: CPS to Makefile

```
export CPPFLAGS="$CPPFLAGS -I/home/diegor/.conan2/p/opensa8f76cb308528/p/include
                           -I/home/diegor/.conan2/p/zlib2f8af40ed7773/p/include"
export LIBS="$LIBS -lssl -lcrypto -lz -ldl -lpthread -lrt"
export LDFLAGS="$LDFLAGS -L/home/diegor/.conan2/p/opensa8f76cb308528/p/lib
                         -L/home/diegor/.conan2/p/zlib2f8af40ed7773/p/lib"
export CXXFLAGS="$CXXFLAGS"
export CFLAGS="$CFLAGS"
```

zlib.cps

```
"components": {
   "zlib": {
      "type": "archive",
      "includes": ["@prefix@/include"],
      "location": "@prefix@/lib/zlib.lib"
   }
}
```



autotoolsdeps.sh

Demo

```
|--include
|    openssl/sha.h
|--lib
|    crypto.lib
|-openssl.cps
```

```
|--include
|    zlib.h
|--lib
|    zlib.lib
|-zlib.cps
```

```cpp
#include <openssl/sha.h>

int main(){
    unsigned char sha256_digest[SHA256_DIGEST_LENGTH];
    char sha256_string[SHA256_DIGEST_LENGTH*2+1] = {0};
    char string[] = "happy";

    SHA256((unsigned char*)&string, strlen(string),
           (unsigned char*)&sha256_digest);
    // print sha256_digest into sha256_string as
    // readable str
    std::cout << "******* Hello world ***********\n";
    std::cout << "sha256: " << sha256_string << "\n";
}
```

.props → myapp.exe

xxx-config.cmake
toolchain.cmake → myapp.exe

Env-vars scripts → myapp

meson-toolchain.ini
*.pc → myapp.exe

ZLib

Windows/VS

```
|zlib.cps
|--include
|    zconf.h
|    zlib.h
|--lib
|    zlib.lib
|--licenses
     LICENSE
```

Linux/gcc

```
|zlib.cps
|--include
|    zconf.h
|    zlib.h
|--lib
|    libz.a
|--licenses
     LICENSE
```

zlib.cps

```
{
    "cps_version": "0.12.0",
    "name": "zlib",
    "version": "1.3.1",
    "configurations": ["release"],
    "default_components": ["zlib"],
    "components": {
        "zlib": {
            "type": "archive",
            "includes": ["@prefix@/include"],
            "location": "@prefix@/lib/{{libname}}"
        }
    }
}
```

OpenSSL

Windows/VS

```
|openssl.cps
|--include
|    openssl
|       sha.h
|       …
|--lib
|    libcrypto.lib
|    libssl.lib
|--licenses
     LICENSE
```

openssl.cps

```
{
  "name": "openssl",
  "version": "3.2.2",
  …
  "requires": {"zlib": null},
  "configurations": ["release"],
  "default_components": ["crypto","ssl"],
  "components": {
     "crypto": {
        "type": "archive",
        "requires": ["zlib:zlib"],
        "includes": ["@prefix@/include"],
        "location": "@prefix@/lib/libcrypto.lib",
        "link_libraries": ["crypt32","ws2_32","advapi32","user32","bcrypt"]
     },
     "ssl": {
        "type": "archive",
        "requires": [":crypto"],
        "includes": ["@prefix@/include"],
        "location": "@prefix@/lib/libssl.lib"
      }
   }
}
```

OpenSSL

Linux/gcc

```
|openssl.cps
|--include
|    openssl
|       sha.h
|       …
|--lib
|    libcrypto.a
|    libssl.a
|--licenses
     LICENSE
```

openssl.cps

```
{
    "name": "openssl",
    "version": "3.2.2",
    "requires": {"zlib": null},
    "configurations": ["release"],
    "default_components": ["crypto","ssl"],
    "components": {
        "crypto": {
            "type": "archive",
            "requires": ["zlib:zlib"],
            "includes": ["@prefix@/include"],
            "location": "@prefix@/lib/libcrypto.a",
            "link_libraries": ["dl","rt","pthread"]
        },
        "ssl": {
            "type": "archive",
            "requires": [":crypto"],
            "includes": ["@prefix@/include"],
            "location": "@prefix@/lib/libssl.a",
            "link_libraries": ["dl","pthread"]
        }
    }
}
```

CPS to Conan

conanfile.py (openssl)

```
def package_info(self):
   from conan.cps.cps import CPS
   self.cpp_info = CPS.load("openssl.cps").to_conan()

   self.cpp_info.set_property("cmake_file_name", "OpenSSL")
   self.cpp_info.components["crypto"].set_property("cmake_target_name", "OpenSSL::Crypto")
   self.cpp_info.components["ssl"].set_property("cmake_target_name", "OpenSSL::SSL")
```

conanfile.py (zlib)

```
def package_info(self):
   from conan.cps.cps import CPS
   self.cpp_info = CPS.load("zlib.cps").to_conan()
   self.cpp_info.set_property("cmake_file_name", "ZLIB")
   self.cpp_info.set_property("cmake_target_name", "ZLIB::ZLIB")
```

Demo!

Recap

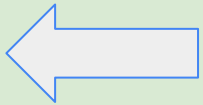
From CPS (Conan) to build systems

zlib.props
openssl.props

ZLIBConfig.cmake
OpenSSLConfig.cmake

conanbuild.sh
(CXXFLAGS, etc)

zlib.pc
openssl.pc



```
|--include
|    zlib.h
|--lib
|    zlib.lib
|-zlib.cps
```

```
|--include
|    openssl/sha.h
|--lib
|    crypto.lib
|-openssl.cps
```

Why didn’t you use the CMake generated CPS files?

CMakeLists.patch

```
--- a/CMakeLists.txt
+++ b/CMakeLists.txt
@@ -5,6 +5,8 @@ project(zlib C)

 set(VERSION "1.3.1")

+set(CMAKE_EXPERIMENTAL_EXPORT_PACKAGE_INFO "b80be207-778e-46ba-8080-b23bba22639e")
+
 option(ZLIB_BUILD_EXAMPLES "Enable Zlib Examples" ON)

 if(NOT SKIP_INSTALL_LIBRARIES AND NOT SKIP_INSTALL_ALL )
-    install(TARGETS zlib
+    install(TARGETS zlib EXPORT zlib

 endif()
+install(PACKAGE_INFO zlib EXPORT zlib)
```

Why didn’t you use the CMake generated CPS files?

zlib.cps

```
{
  "components" :
  {
    "zlib" :
    {
      "type" : "archive"
    }
  },
  "cps_path" : "@prefix@/cps",
  "cps_version" : "0.12.0",
  "name" : "zlib"
}
```

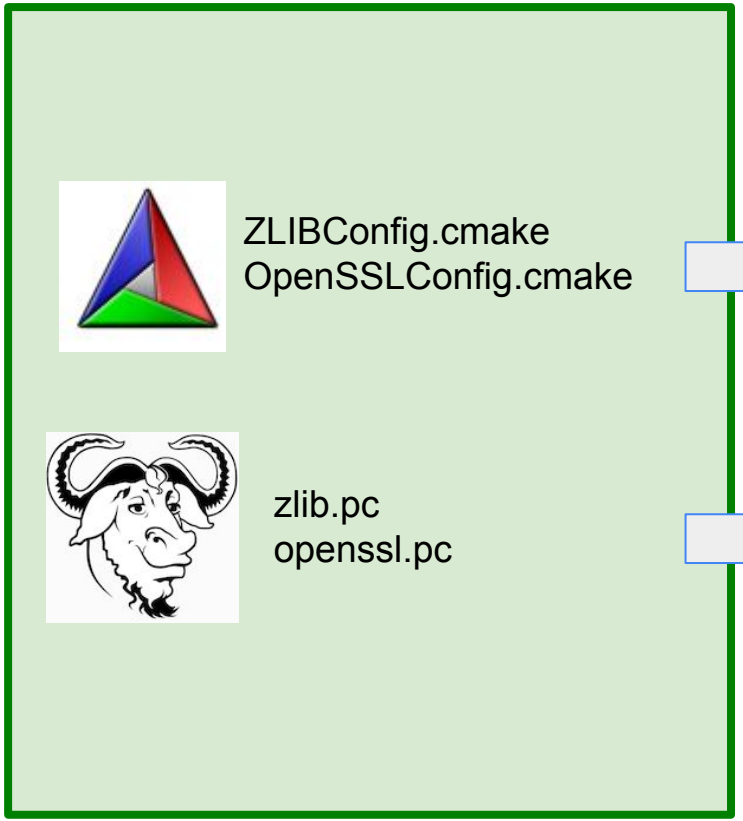
zlib@release.cps

```
{
  "components": {
    "zlib": {
      "link_languages": [
        "c"
      ],
      "location": "@prefix@/lib/zlib.lib"
    }
  },
  "configuration": "Release",
  "name": "zlib"
}
```

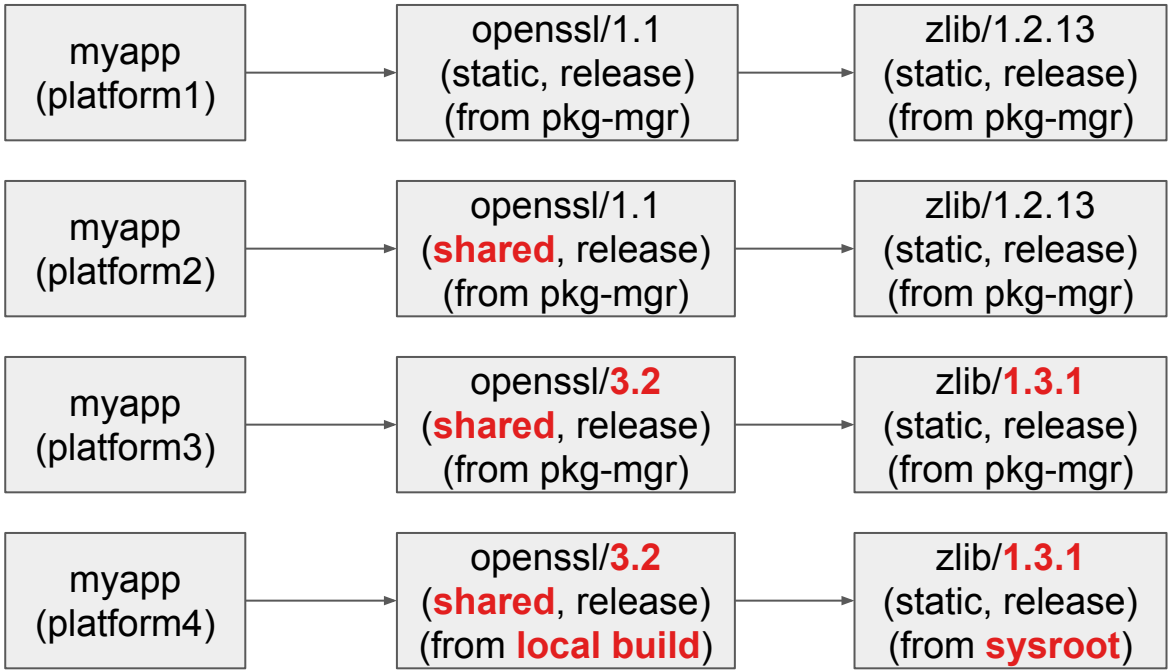
Outline

- Introduction to Common Package Specification (CPS)
- Creation of CPS files from existing Conan packages
- Loading CPS files generated by build systems
- Generating build system native files from CPS
- **Location of CPS files**
- Lessons learned and conclusions

Location of dependencies files



Problem will increase with versions and configurations



Problem will increase with versions and configurations

myapp
(platform1)

myapp
(platform2)

myapp
(platform3)

myapp
(platform4)

```
cmake_minimum_required(VERSION 3.15)
project(myapp CXX)

find_package(openssl REQUIRED)

add_executable(myapp src/main.cpp)

target_link_libraries(myapp PRIVATE
                      openssl::openssl)
```

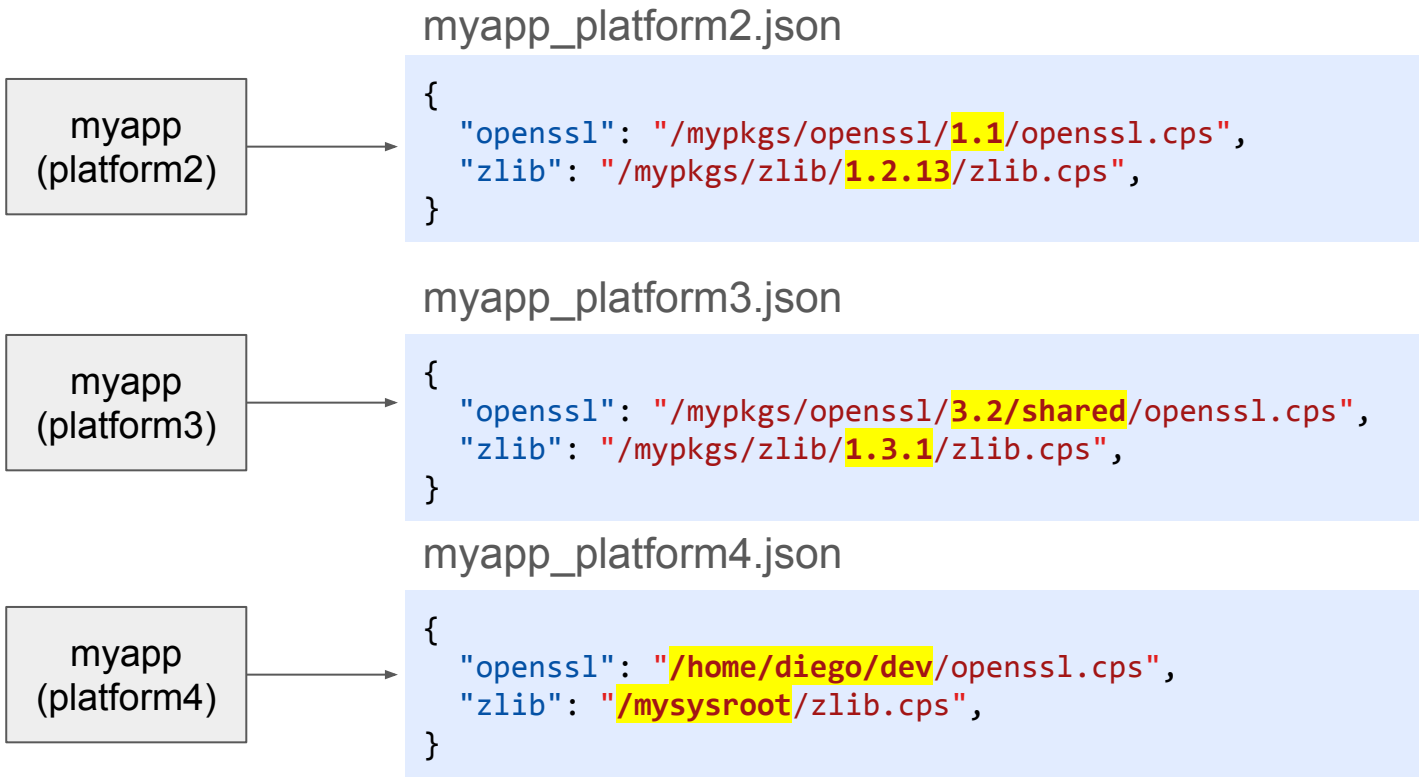
Proposal: CPS location files

cpsmap-msvc-194-x86_64-17-release.json

```
{
  "tensorflow-lite": "/path/to/cps/tensorflow-lite.cps",
  "abseil": "/path/to/cps/abseil.cps",
  "farmhash": "/path/to/cps/farmhash.cps",
  "fft": "/path/to/cps/fft.cps",
  "flatbuffers": "/path/to/cps/flatbuffers.cps",
  "gemmlowp": "/path/to/cps/gemmlowp.cps",
  "ruy": "/path/to/cps/ruy.cps",
  "xnnpack": "/path/to/cps/xnnpack.cps",
  "cpuinfo": "/path/to/cps/cpuinfo.cps",
  "pthreadpool": "/path/to/cps/pthreadpool.cps",
  "opencv": "/path/to/cps/opencv.cps",
  "protobuf": "/path/to/cps/protobuf.cps",
  "ade": "/path/to/cps/ade.cps",
  ...
}
```

Already generated by CPSDeps

CPS location files: versions, locations



CPS location files: requires conflicts resolution

openssl.cps

```
{
  "name": "openssl",
  "version": "3.2.2",
  …
  "components": {
    "crypto": {
      "type": "archive",
      "requires": ["zlib:zlib"],
      "includes": ["@prefix@/include"],
    },
  }
}
```

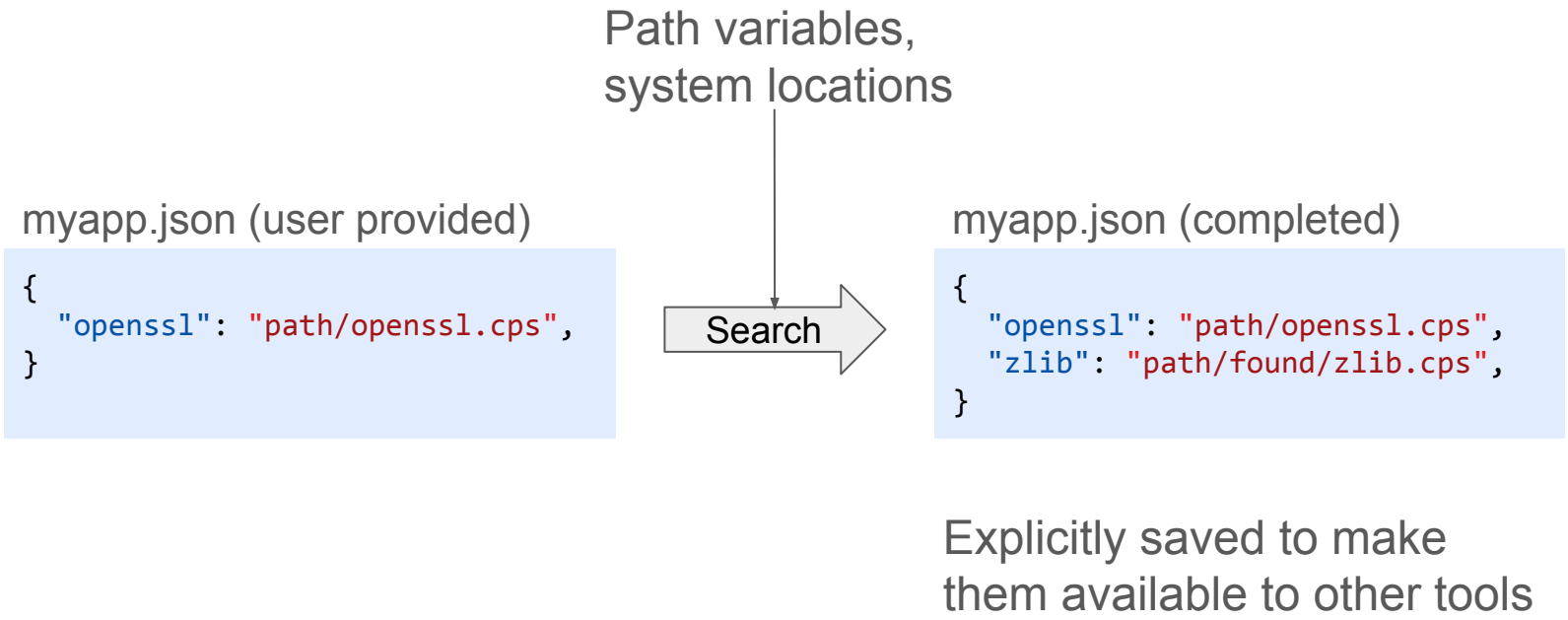
mypkg.cps

```
{
  "name": "mypkg",
  "version": "1.0",
  …
  "components": {
     "crypto": {
        "type": "archive",
        "requires": ["myzlib:myzlib"],
        "includes": ["@prefix@/include"],
     },
  }
}
```

myapp.json

```
{
  "openssl": "/mypkgs/openssl/openssl.cps",
  "mypkg": "/mypkgs/mypkg/mypkg.cps",
  "zlib": "/mypkgs/zlib/zlib.cps",
  "myzlib": "/mypkgs/zlib/zlib.cps",
}
```

Works well with search policies



Outline

- Introduction to Common Package Specification (CPS)
- Creation of CPS files from existing Conan packages
- Loading CPS files generated by build systems
- Generating build system native files from CPS
- Location of CPS files
- **Lessons learned and conclusions**

Lessons

- Talk is cheap, show me the code
- Spec good, but a bit scarce/rough:
  - Need full real examples (not clear about “requires” syntax)
  - Need best practices, like private components
  - Need merge zlib.cps + zlib@release.cps examples
- Generation of CPS from existing CMakeLists.txt will be challenging
- Proposal: CPS generation also at configure/build time
- Usability/utility to existing projects will be important for adoption
  - find_package(ZLIB) instead of load_package(zlib.cps)
  - Complementary information for existing code (like CMake special target names) like target_link_libraries(myapp ZLIB::ZLIB)

Conclusions

- Pushed the CPS effort:
  - Very large public corpus/benchmark of CPS files for many thousands of existing packages (via CPSDeps) Also for private packages
  - Initial testing and feedback about the CPS generation of CMake
  - Proved and demoed CPS usage (openssl + zlib) in 4 different build systems
  - Discussed location of CPS files, lessons learned and possible gaps
- Future work:
  - Keep validating with CPS community
  - Contribute more to the spec and to other tools
  - Implement executables model in in CPS (first in Conan)
  - Start iterating and closing the loop with other tool providers, like CMake
  - Evolve CPS library, make it independent if valuable

Thanks!

Find us at the Conan booth!

https://conan.io

https://github.com/conan-io/conan

https://cps-org.github.io/cps

#ecosystem_evolution
(CppLang slack)