ROOT Package Manager

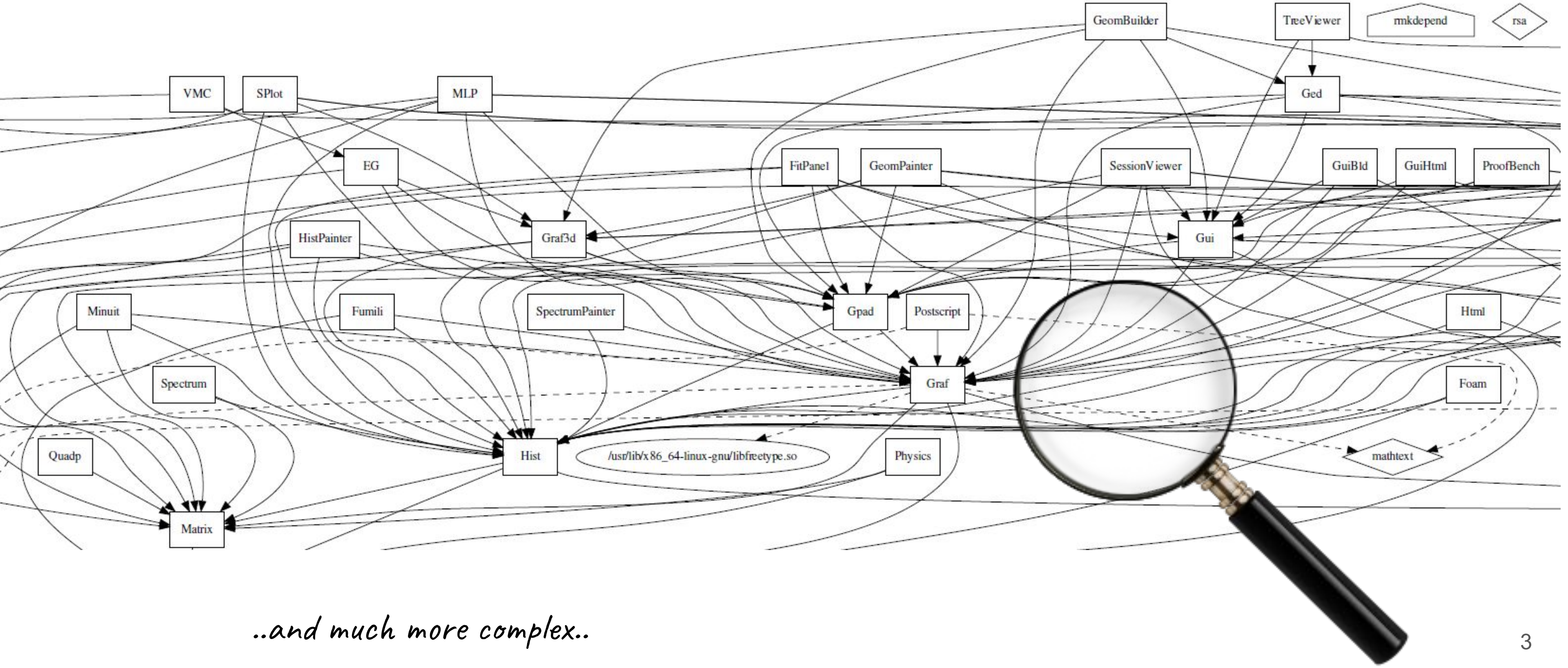
Oksana Shadura (University of Nebraska Lincoln)
Brian Paul Bockelman (University of Nebraska Lincoln)
Vassil Vassilev (Princeton University)

Goal

Introducing modules/packages to the ROOT follows the design principle:
do not pay for what you don't use.

Practical use case: instead of downloading more then 1GB of full ROOT sources or pre configured ROOT binaries, you can decide to start with minimal set ~50 Mb and expand with any customization you want.

ROOT dependency map



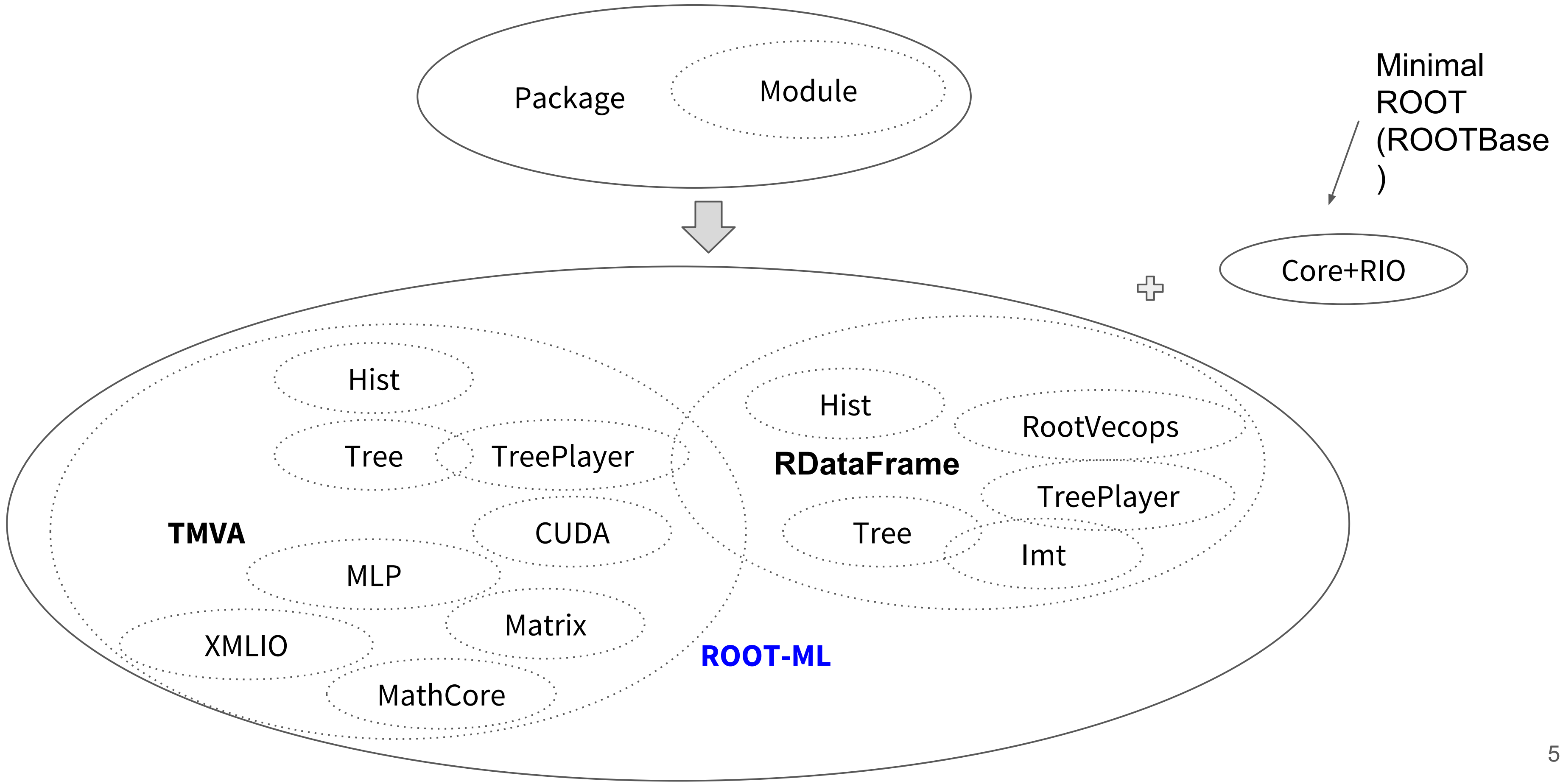
..and much more complex..

Motivation for ROOT Package Manager (PM)

- **Better layering:** introducing a module layer, **that allow ROOT to scale even further.**
- **Better boundaries:** by making the boundaries and relationships more explicit through modules - we can better define “minimal ROOT,” **that enables ROOT users to interact with the wider data science ecosystem.**
- **Sharing between users:** package management provide a mechanism for ROOT users to socialize and and reuse projects built in the context of ROOT, **that allow ROOT be more flexible and open for new customers.**

These items allow ROOT to continue successfully to serve as a HEP community nexus.

ROOT package & component (module)



ROOT PM manifest

The definition of package assumes a contract for code organization in order to simplify the build and deploy steps. The contract defines **a *manifest* file** and particular organization of each module.

Usage scenarios and benefits of manifest files

1. **I am a ROOT subsystem user or developer (e.g. io)**. The manifest file is generated by the info in the build system.
2. **I am a third party developer (PhD student)** who has 5 files and does not know anything about build systems and alike -- I 'just' describe in a human form what my package does and what ROOT components it depends on.
3. **I am experiment librarian and I know what exactly I need** -- writing manifest file or some other configuration to tell ROOT what packages I need is the ideal scenario. The other scenario could be to describe a pre-built package.

Easier with manifests!

Connect the PM to ROOT's runtime

- This is where CMake falls short as it does not have any support for steps happening after build/install time
- **PM allows bootstrapping minimal ROOT and installing packages automatically on demand**
  - It provides a basic interpreter functionality, which will allow to call:

```
[] #include "TMVA/DataLoader.h"

[] error: TMVA/Dataloader.h not found.

note: TMVA/Dataloader.h is part of TMVA package, do you wish to install it?[Y/n]

[] auto dataloader = new TMVA::DataLoader(‘test’);// works without quitting ROOT.
```

ROOT Package manager flow *(example - external package https://github.com/hep-cce/TMPIFile)*

Project code ⇨ Package manifest file

We download from git:

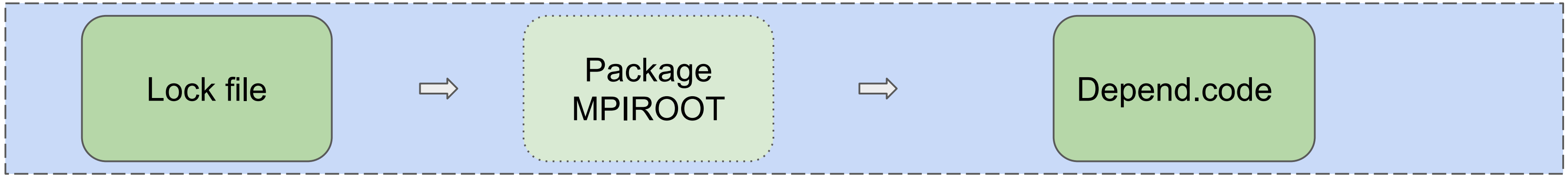
- *src/*
- *include/*
- *NO CMakeList.txt with ROOT_STANDARD_LIBRARY_PACKAGE()*
- *Makefile*
- *Setup script*

```
package:
  name: MPIROOT
  targets:
     target:
        name: MPIROOT
  products:
     package:
        name: MPIROOT
     module:
        name: TMPIFile
           …
```

```
module:
  name: "TMPIFile"
   packageurl:

"https://github.com/hep-cce/TMPIFile.git
"
   Path:
     X
   publicheaders: *.h
   sources: *.cxx
   tests: test_tmpi.C
   targets: TMPIFile
   deps: mpich mpi mpicxx mpl opa
```

Package manager flow



Generating a **dependency graph (DAG)** and resolving dependencies via various strategies), listed in the project's manifest *(for example how to install them)*

Generating and deploying MPIROOT.zip:

- *inc/*
- *libTMPIFile.so*
- *License file*
- *Manifest.yml*
- *TMPIFile_rdict.pcm*
- *TMPIFile.rootmap*

- All of the source code from lock file, **arranged on disk in a such way that the compiler/interpreter can use it** as intended, **but isolated** to be avoid mutation.
- **Modules/packages can be installed in any location**, even outside of the install path of ROOT, all you need is to have ROOTbase and root-get installed in system.

ROOT PM and its relation with ROOT C++ modules

- ROOT C++ modules & runtime C++ modules [ROOT is using Clang C++ modules]
  - Clang C++ modules is precompiled headers that optimize header parsing
    - Clang can load on-demand code from modules
    - It is similar to ROOT PCH
  - Due we build interpreter, we are optimizing header parsing at runtime and we call it runtime C++ modules
- **ROOT runtime C++ modules will solve problem that ROOT PCH can be only single in the system**
  - **it is important part for ROOT PM design!**
- In the same time while using C++ modules for PM, we will try to help to solve a global problem of distribution C++ modules
- Status: ROOT is working on enabling C++ modules and runtime C++ modules for 6.16 release
  - For more details about ROOT C++ modules **please check talk of Yuka Takahashi (Princeton University)**

Progress

- Ongoing work on *changes to build system of ROOT***;**
  - *SearchSoftware.cmake* is trying to discover all components based on defined set of default ROOT components / or extra components that they depend on;
    - User has less control if custom package was actually enabled, because of complex requirements (ROOT builtins and SearchSoftware.cmake);
      - **Goal: to move from ROOT component-based schema to package-enabled and introduce dynamic "package-component map" in ROOT**
  - Update the way **how external packages are been discovered;**
    - If user is requesting package, and for some reasons you can't reach requirements for package, ROOT build will be failed;
  - Release a ROOT-patched LLVM+Clang: it will give a possibility to build ROOT 30 % faster;
- Ongoing work on *a delivery of first version of root-get;*
- Ongoing work on *integration of root-get in ROOT runtime;*

We plan to land some changes for 6.16.xx or 6.18.xx, so you will be able to test PM!

Planned features

- *root-get* should be able to interact with OS package managers and install missing dependencies;
- *root-get* will be able to find released minimal part of ROOT (interpreter+Core+RIO) or build it by itself;
- *root-get* could be able to collect and generate statistics on ROOT components popularity (similar to https://cocoapods.org/pods/protobuf-c);

Conclusions

- We described how could look like **package management ecosystem for ROOT**
- Also during work was defined additions to CMake build system for generating packages from a ROOT build
- *All ideas was adopted in a preliminary prototype (root-get) that can download and install packages.* A prototype could be connected to ROOT runtime and serve as a runtime dependency management tool [work in progress]

Thank you for your attention!

Backup slides

Main motivation for application in HEP and beyond

Need of an improved ROOT granularity

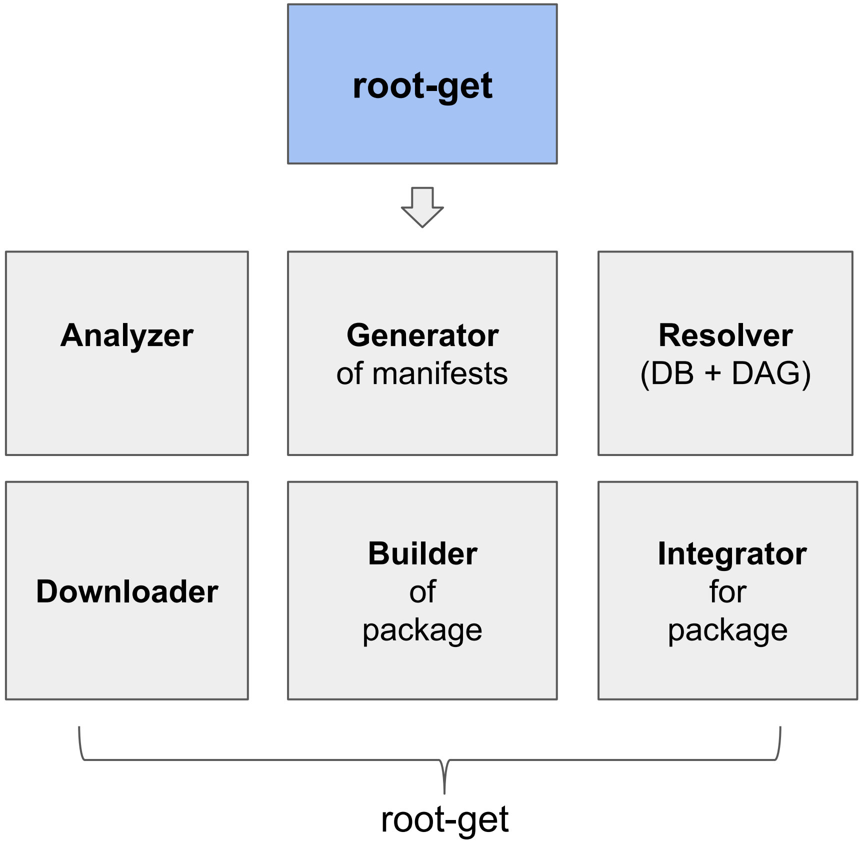
(LHCB presentation for EP-SFT Librarian and Integrators workshop: https://indico.cern.ch/event/720948/contributions/2968723/))

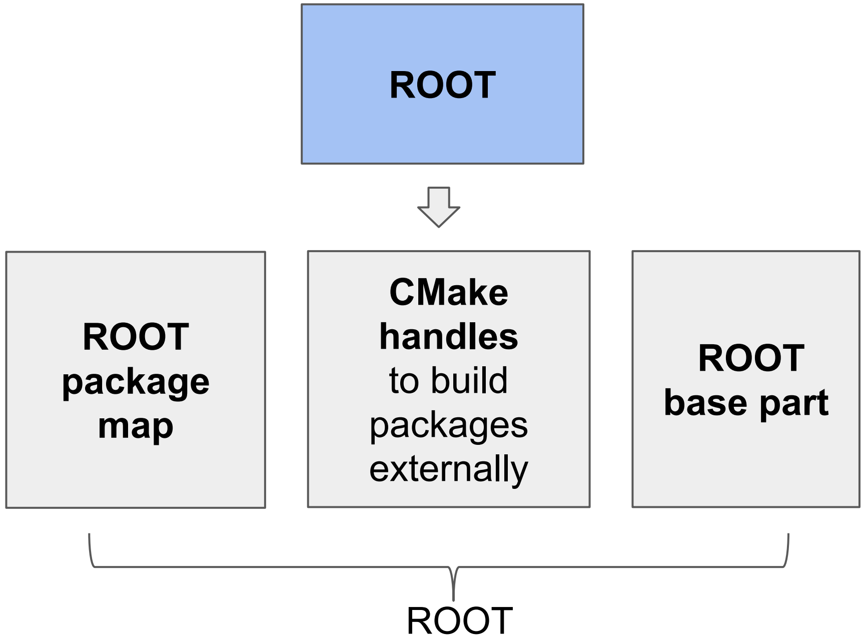
CMSSW: ROOT is one of core dependences, any changes are causing to rebuild almost whole stack of packages (one pch, but many components)

Easy versioning of dependencies (ROOT externals and builtins)

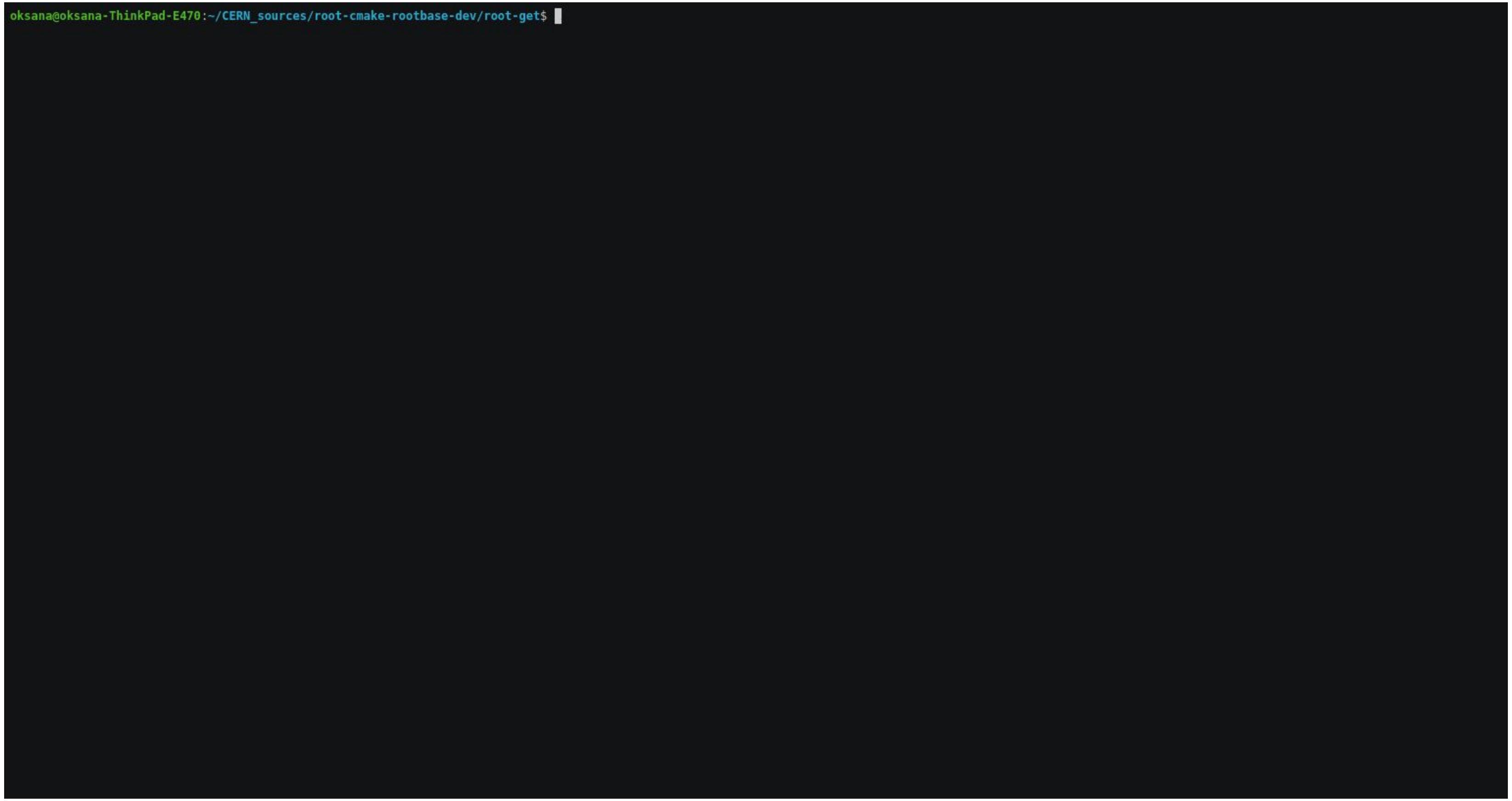
ROOT Packages intend to introduce a different flow of design & development of software around ROOT's ecosystem

ROOT package manager: ingredients





Demo



```
oksana@oksana-ThinkPad-E470:~/CERN_sources/root-cmake-rootbase-dev/root-get$
```

https://asciinema.org/a/jUkLgJjxpUZISFrh9Cv9dkeUz

Project Dependency Manager & “compiler, phase zero”

- System for managing the source code dependencies of a *single project* in a particular language. That means specifying, retrieving, updating, arranging on disk, and removing sets of dependent source code. PDMs reproducible output is a self-contained source tree that acts as the input to a compiler or interpreter. => “compiler, phase zero.”

*“compiler, phase zero’ idea is similar to JIT’ing. In addition to putting the code on disk, the PDM typically needs to override the interpreter’s code loading mechanism in order to resolve includes correctly (PDM is producing a filesystem layout for itself to processed.)”

[1] https://www.tutorialspoint.com/compiler_design/compiler_design_phases_of_compiler.htm

ROOT external map and CMake handles

- How to divide a ROOT into packages and modules?
  - We introduced ROOTComponentMap.cmake to help us with this task (statically and dynamically)
- We will provide a possibility to plug your own map of packages and modules!

custom_map.yml:

ML:

MathCore, MathMore, PyRoot, TMVA, CustomDiplomaOfPHDStudentLib

cmake ../ -DML=ON* or *root-get -i ML

(only what will be needed in addition - it is a manifest file of CustomDiplomaOfPHDStudentLib)