1. Introduction
We will present our results and challenges with C++ modules in ROOT. ROOT was extended with experimental support for using C++ modules during runtime, with aims to reduce its memory usage and improve its correctness.

2. C++ Modules in a Nutshell
- Header information is stored in precompiled PCM files.
- No more header parsing during ROOT’s runtime.

   In C/C++, the interface of a library is accessed by including the appropriate header files:

   ```
   #include <stdio.h>
   ```

   These textual includes cause well-known problems:
   - **Compile-time scalability:** Include is copying the contents to the includer’s code, so the parser has to reparse the same common header files multiple times, which is expensive.
   - **Fragility:** Textual includes are influenced by previously defined macros. For example, macro PI is #defined in Rcpp library. Including this library while using local variable PI will end up in a redefinition error.

   C++ modules are a mechanism to boost compilation time by precompiling headers into PCM files, where AST information can be lazily loaded.

3. From C++ Modules to Runtime C++ Modules
   - PCM file (E.g. Core.pcm) corresponds to a library. Including this library while using local variable PI will end up in a redefinition error.

   ```
   #include <stdio.h>
   ```

   - C++ modules are able to reduce compilation times. However, the compilation scalability issues in C/C++ becomes runtime issues for an interpretative environment which ROOT provides. They span from slow prompt to slow IO.

4. Advantages over the Status Quo

   PCH files are precompiled header files and work similar to C++ modules. The advantage of modules over PCH is that they can be used by experiments. Experiments are still using textual includes as PCH only covers ROOT. PCH cannot be exported to experiments because of various technical limitations.

5. Results
5.1 Performance

   ![Figure 3: CPU Time required to run selected tutorials. The first column is displaying ROOT’s time to start into an empty shell.](image)

   ![Figure 4: Residential memory used to run tutorials.](image)

5.2 Correctness

   ![Figure 5: Visualization of ROOT interpreter core.](image)

   Fig.3 and Fig.4 are the performance results we receive from modules, compared to textual headers. The results are coming from synthetic benchmarks close to the experiment software stacks and in particular CMSWW.

6. Implementation

   - **Adopt by experiments and other ROOT users**
   - **Stabilize modules behavior and tests**
   - **Improve performance of loading modules**
   - **Resolve symbols and cases like those are now correctly handled.**

7. Roadmap

   - **Compile ROOT with C++ modules**
   - **Compile CMSSW with C++ modules**
   - **Use runtime C++ modules in ROOT**
   - **Use runtime C++ modules in experiments**

8. Conclusion

   Here we briefly introduced our experimental runtime C++ modules support in ROOT and how it will affect experiments’ software stacks. Modules are not yet competent compared to PCH, but are more flexible and have clear advantage over textual includes.

9. Future work

   Runtime modules are still an experimental feature. Our ultimate goal is to make it default in ROOT and in experiments:
   - **Stabilize modules behavior and tests**
   - **Adoption by experiments and other ROOT users**
   - **Improve performance of loading modules**

Further Information

- https://clang.llvm.org/docs/Modules.html
- https://root.cern.ch/
- https://root.cern.ch/clang