Software Survey and Plans for A Repository for Common Analysis Algorithms

CLAS Collaboration Meeting

Jefferson lab

November 2023

Christopher Dilks

Jefferson Lab

U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

Software Survey

Many thanks to the participants!

Mission Statement:
The primary goal of this survey is to design a repository of common methods shared among physics analyses, such as fiducial cuts and enhanced PID criteria. This repository will aim to provide simple access to common techniques, and to preserve them under version control.

We are conducting this survey for the full CLAS Collaboration so that the design of this repository is focused on the user needs and interests. After evaluating feedback from this survey, we will focus the prototype design on Run Group A, and if well-received, continue to support other Run Groups.

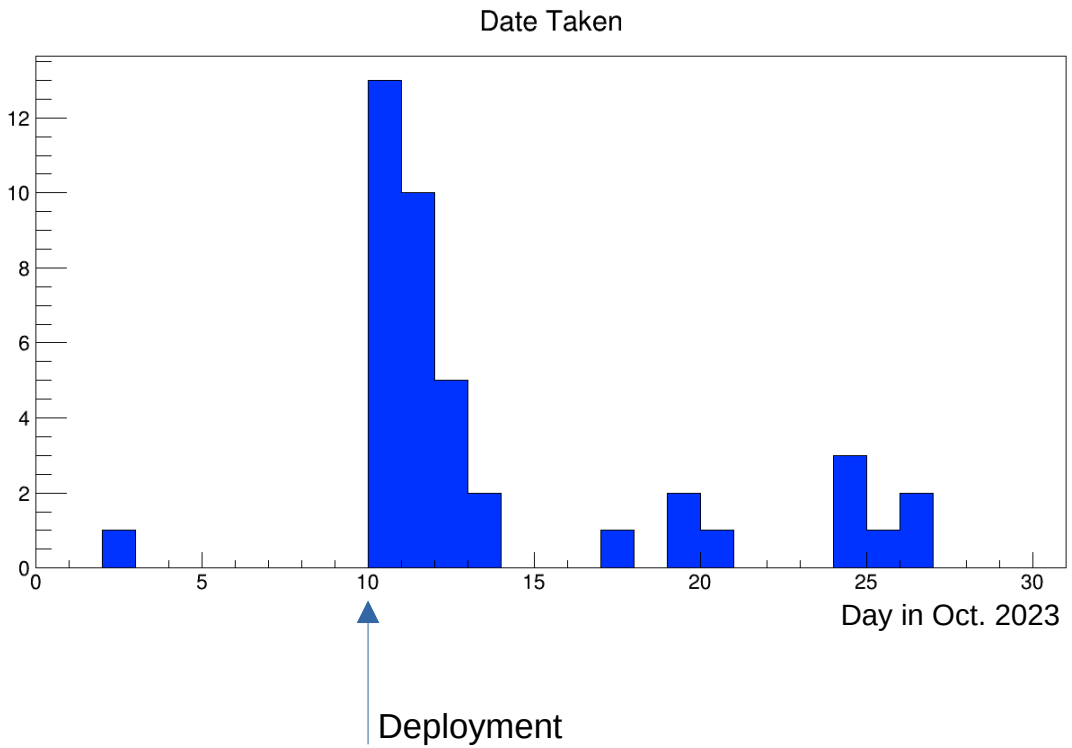
Software Survey

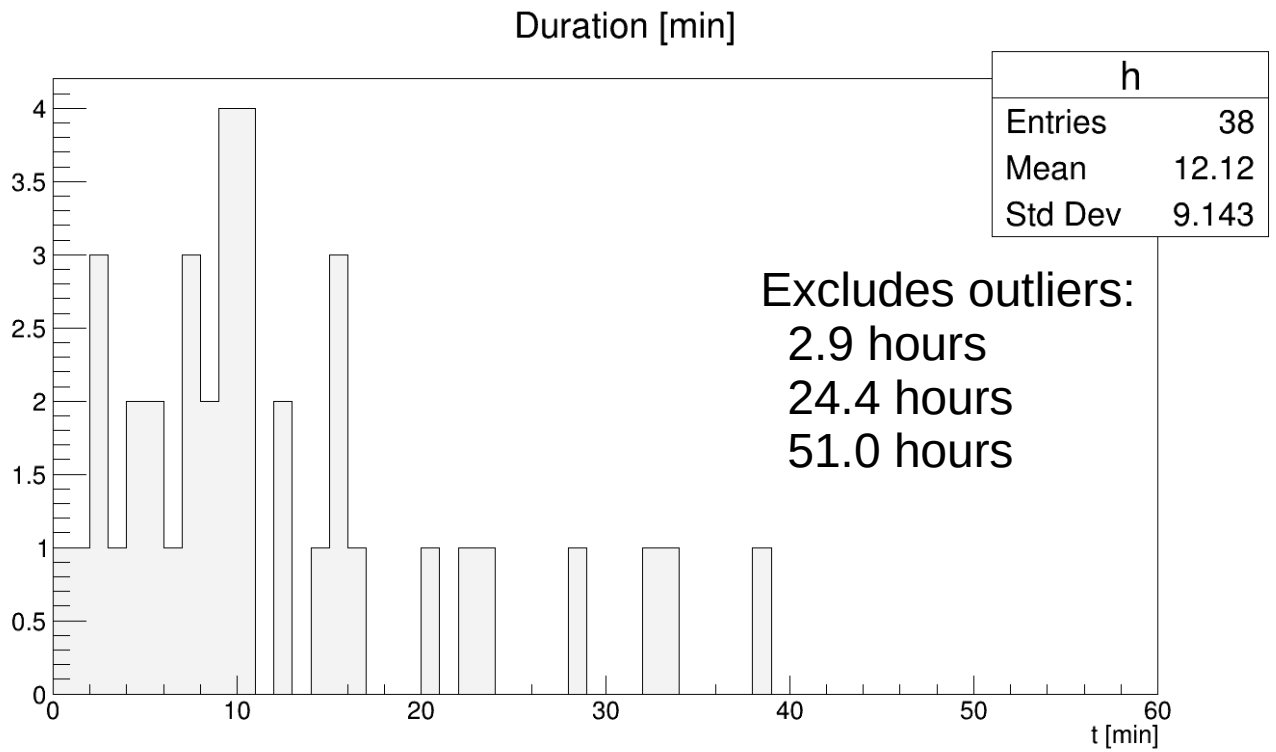
Many thanks to the participants!

- Implement a repository of common methods (“algorithms”) shared among physics analyses, such as fiducial cuts and enhanced PID criteria.
  - Provide simple access to common techniques
  - Algorithm preservation
- User-centered design → the software survey
  - Additional questions of general interest to the CLAS Software Group were included

Responses

- 41 respondents
- Average time to take the survey: 12 minutes
- **More responses are still welcome!**



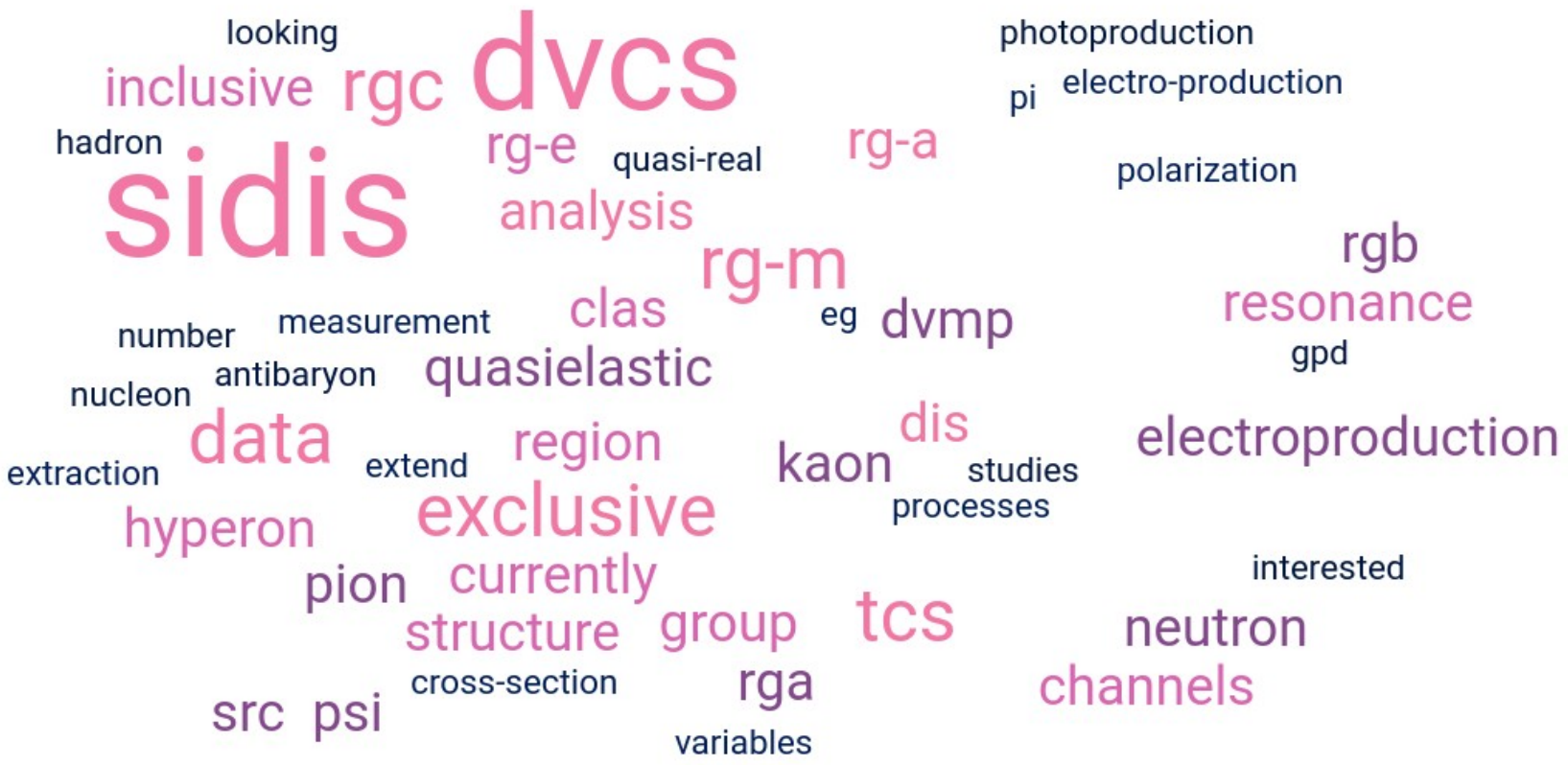


Jefferson Lab

Survey Results

What physics analyses are you working on?

- Nothing surprising here…
- Broad range!
- More details than what the word cloud shows
  - MesonX
  - KY
  - N*
  - J/psi
  - CLAS6
  - and more



Jefferson Lab

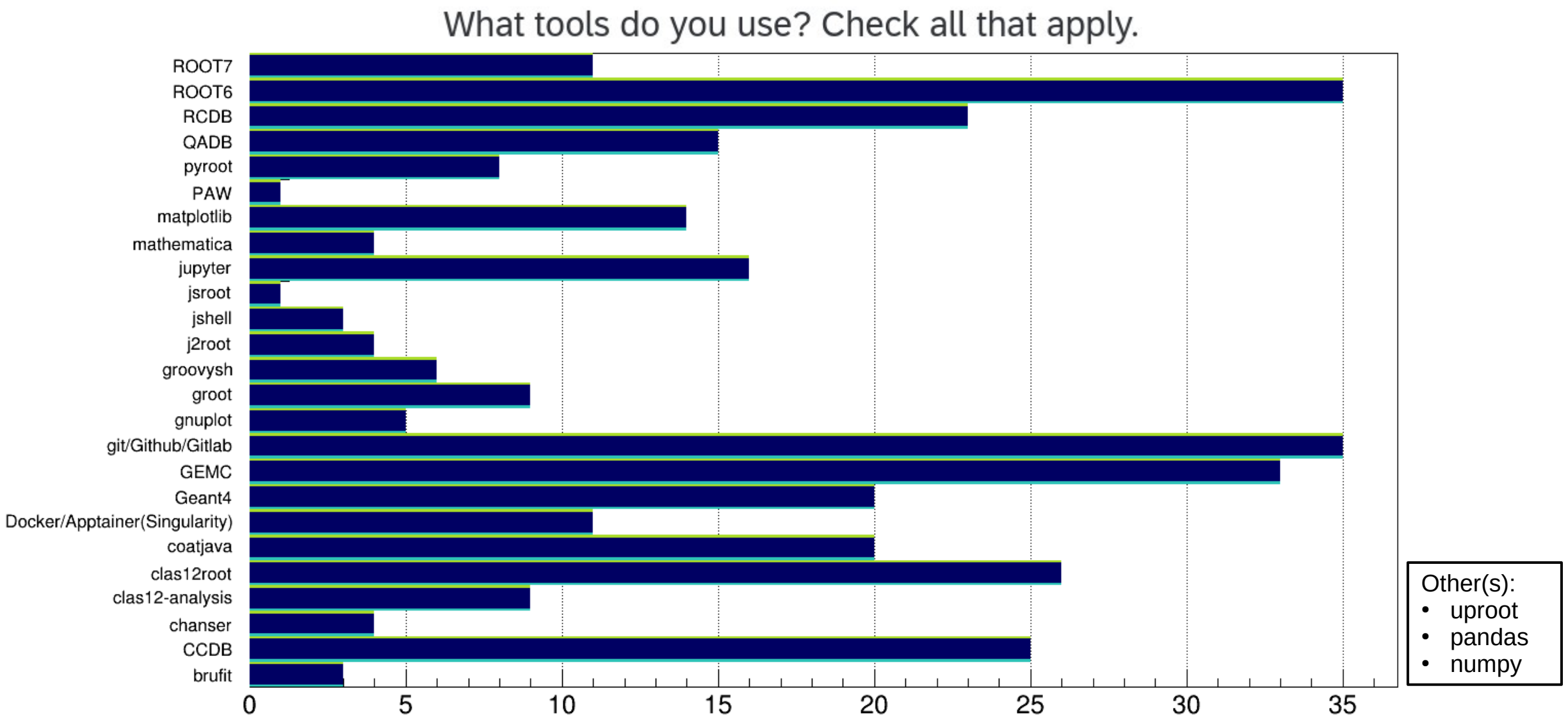
Survey Results

Are you working on anything else that uses CLAS12 software or data? If so, please list:

(one response per line)

```
A bit of Machine Learning
BeamSpot
Calibration: alignment, beam-offset, etc.
calibration and tuning of the FTTRK detector
Calibrations of CND, DC alignment, beam offset calibration, reconstruction
COATJAVA
GMC
I converted the CLAS6 EG2 BOS files into HIPO files and analyze with coatjava.
I sometimes like looking at existing CLAS12 data to try to plan future proposals.
I use CLAS12 software for calibrating the HTCC.
Machine learning simulations.
momentum corrections, PID refinements
n/a
No
no. but I do use clas results (cross-sections and the like) in some of my other projects.
Preparing training data for AI applications (eg level 3 trigger)
QADB
RICH monitoring
simulations (GEMC)
```

Survey Results



Jefferson Lab

Survey Results

What tools do you use? Check all that apply.

Most respondents use ROOT

Those who do not use:

(one response per line)

```
groot,matplotlib,clas12-analysis,Other(s)
matplotlib,jupyter,Geant4,GEMC,clas12root,coatjava,CCDB,RCDB,QADB,git/Github/Gitlab,Other(s)
PAW,GEMC,CCDB,RCDB,QADB,git/Github/Gitlab,Docker/Apptainer(Singularity)
```

Other(s):
- uproot
- pandas
- numpy

Some respondents use ROOT v7

Most of which also use ROOT v6

```
ROOT6,ROOT7,Geant4,GEMC,clas12root
ROOT6,ROOT7,GEMC,clas12root,coatjava,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,ROOT7,gnuplot,Geant4,GEMC,clas12root,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,ROOT7,gnuplot,groovysh,Geant4,GEMC,clas12root,coatjava,CCDB,RCDB,git/Github/Gitlab
ROOT6,ROOT7,groot,mathematica,groovysh,GEMC,coatjava,clas12-analysis,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,ROOT7,jsroot,clas12root,coatjava,j2root,clas12-analysis,CCDB,git/Github/Gitlab
ROOT6,ROOT7,jupyter,Geant4,GEMC,clas12root,brufit,chanser,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,ROOT7,matplotlib,jupyter,GEMC,clas12root,coatjava,CCDB,RCDB,git/Github/Gitlab
ROOT6,ROOT7,pyroot,matplotlib,jupyter,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT7,GEMC,git/Github/Gitlab
ROOT7,pyroot,jupyter,Geant4,GEMC,RCDB,git/Github/Gitlab
```

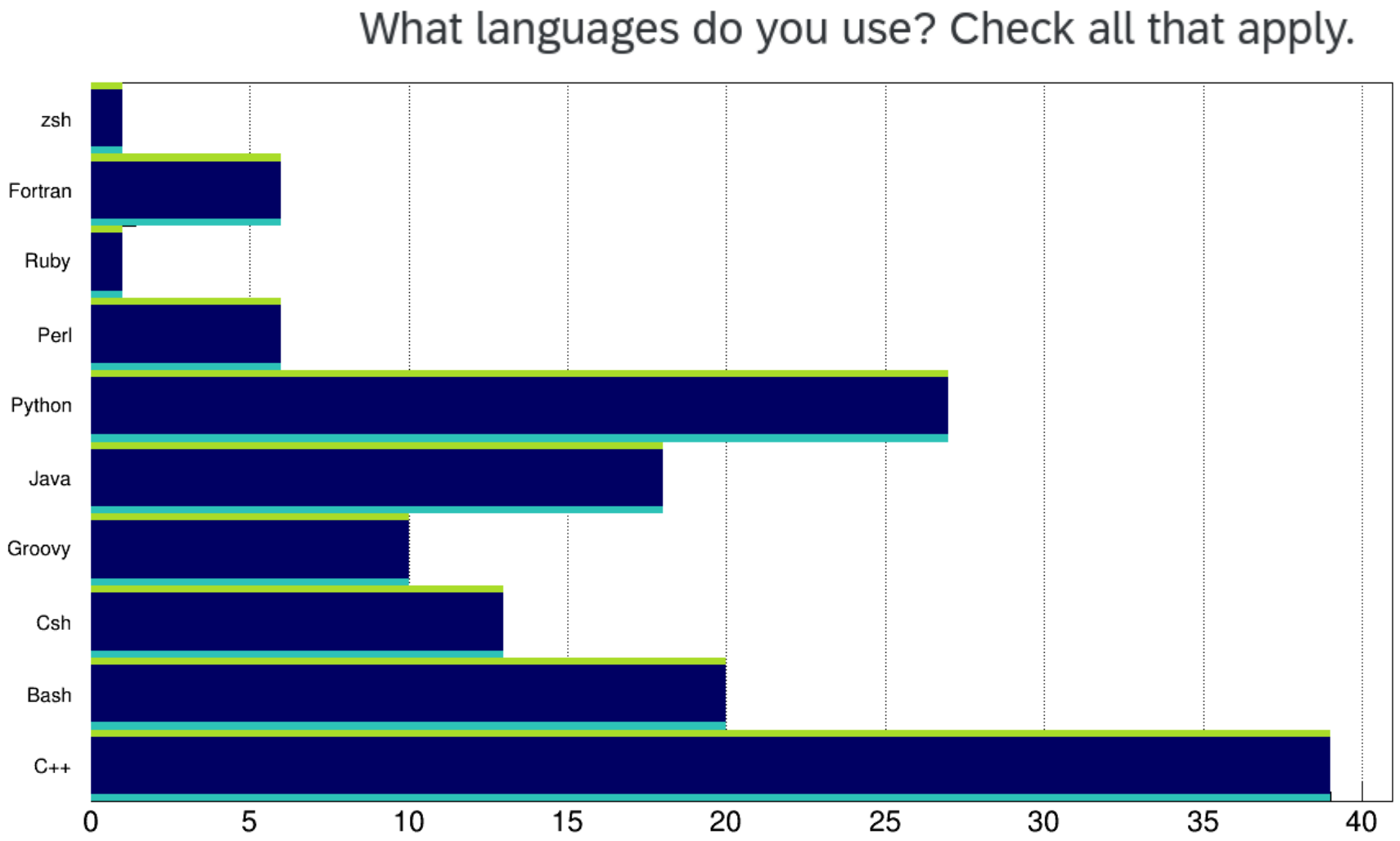
Survey Results

(all responses, one per line)

What tools do you use? Check all that apply.

groot,matplotlib,clas12-analysis,Other(s)
matplotlib,jupyter,Geant4,GEMC,clas12root,coatjava,CCDB,RCDB,QADB,git/Github/Gitlab,Other(s)
PAW,GEMC,CCDB,RCDB,QADB,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT6,clas12root
ROOT6,clas12root,git/Github/Gitlab
ROOT6,clas12root,j2root,clas12-analysis,QADB
ROOT6,Geant4,GEMC,clas12root,coatjava,clas12-analysis,CCDB,RCDB,git/Github/Gitlab
ROOT6,Geant4,GEMC,coatjava,CCDB,RCDB,git/Github/Gitlab
ROOT6,GEMC,clas12root,coatjava,CCDB,RCDB
ROOT6,GEMC,clas12root,coatjava,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,GEMC,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT6,gnuplot,jupyter,Geant4,clas12root,git/Github/Gitlab
ROOT6,groot,GEMC,clas12root,brufit,chanser,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,groot,GEMC,clas12root,coatjava,clas12-analysis,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,groot,groovysh,jshell,Geant4,GEMC,coatjava,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT6,groot,jupyter,groovysh,Geant4,GEMC,coatjava,CCDB,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT6,jupyter,Geant4,GEMC,clas12root,chanser,CCDB,RCDB,QADB,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT6,jupyter,Geant4,GEMC,clas12root,clas12-analysis,CCDB,RCDB,git/Github/Gitlab
ROOT6,matplotlib,Geant4,GEMC,clas12root,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,matplotlib,gnuplot,Geant4,GEMC,clas12root,git/Github/Gitlab
ROOT6,matplotlib,jupyter,GEMC,clas12root,coatjava,CCDB,RCDB,QADB,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT6,matplotlib,jupyter,GEMC,clas12root,coatjava,clas12-analysis,git/Github/Gitlab
ROOT6,matplotlib,jupyter,jshell,Geant4,GEMC,coatjava,CCDB,RCDB,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT6,pyroot,gnuplot,jshell,Geant4,GEMC,coatjava,CCDB,RCDB,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT6,pyroot,groot,matplotlib,mathematica,groovysh,Geant4,GEMC,coatjava,j2root,brufit,QADB,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT6,pyroot,groot,matplotlib,mathematica,jupyter,Geant4,GEMC,clas12root,coatjava,chanser,clas12-analysis,CCDB,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT6,pyroot,groot,matplotlib,mathematica,jupyter,groovysh,Geant4,GEMC,coatjava,j2root,CCDB,RCDB,git/Github/Gitlab
ROOT6,pyroot,matplotlib,jupyter,GEMC,clas12root,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,pyroot,matplotlib,jupyter,GEMC,clas12root,git/Github/Gitlab
ROOT6,ROOT7,Geant4,GEMC,clas12root
ROOT6,ROOT7,GEMC,clas12root,coatjava,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,ROOT7,gnuplot,Geant4,GEMC,clas12root,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,ROOT7,gnuplot,groovysh,Geant4,GEMC,clas12root,coatjava,CCDB,RCDB,git/Github/Gitlab
ROOT6,ROOT7,groot,mathematica,groovysh,GEMC,coatjava,clas12-analysis,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,ROOT7,jsroot,clas12root,coatjava,j2root,clas12-analysis,CCDB,git/Github/Gitlab
ROOT6,ROOT7,jupyter,Geant4,GEMC,clas12root,brufit,chanser,CCDB,RCDB,QADB,git/Github/Gitlab
ROOT6,ROOT7,matplotlib,jupyter,GEMC,clas12root,coatjava,CCDB,RCDB,git/Github/Gitlab
ROOT6,ROOT7,pyroot,matplotlib,jupyter,git/Github/Gitlab,Docker/Apptainer(Singularity)
ROOT7,GEMC,git/Github/Gitlab
ROOT7,pyroot,jupyter,Geant4,GEMC,RCDB,git/Github/Gitlab

Survey Results



Wide range
C++ dominant, followed by Python, Bash, Java

We did not ask the application of these languages (e.g., it's unlikely one does CPU-intensive analysis in Bash)

Survey Results

What languages do you use? Check all that apply.

Java users also use:

```
C++,Java
C++,Java,Groovy,Bash
C++,Java,Groovy,Bash,Csh
C++,Java,Groovy,Python
C++,Java,Groovy,Python,Bash,Csh,Perl
C++,Java,Groovy,Python,Bash,Csh,Perl
C++,Java,Groovy,Python,Bash,Ruby
C++,Java,Groovy,Python,Fortran,Bash,Csh
C++,Java,Groovy,Python,Fortran,Bash,Csh
C++,Java,Python
C++,Java,Python
C++,Java,Python,Bash
C++,Java,Python,Bash
C++,Java,Python,Bash,Csh
C++,Java,Python,Csh
C++,Java,Python,Fortran,Perl
C++,Java,Python,Perl
Java,Groovy,Fortran
```

Python users also use:

```
C++,Java,Groovy,Python
C++,Java,Groovy,Python,Bash,Csh,Perl
C++,Java,Groovy,Python,Bash,Csh,Perl
C++,Java,Groovy,Python,Bash,Ruby
C++,Java,Groovy,Python,Fortran,Bash,Csh
C++,Java,Groovy,Python,Fortran,Bash,Csh
C++,Java,Python
C++,Java,Python
C++,Java,Python,Bash
C++,Java,Python,Bash
C++,Java,Python,Bash,Csh
C++,Java,Python,Csh
C++,Java,Python,Fortran,Perl
C++,Java,Python,Perl
C++,Python
C++,Python
C++,Python
C++,Python,Bash
C++,Python,Bash
C++,Python,Bash
C++,Python,Bash
C++,Python,Bash
C++,Python,Bash,Csh
C++,Python,Bash,Csh,Perl,zsh
C++,Python,Csh
C++,Python,Fortran,Bash,Csh
C++,Python,Fortran,Csh,Perl
```

Only 1 respondent does not use C++: they use:

```
Java,Groovy,Fortran
```

Fortran users also use:

```
C++,Java,Groovy,Python,Fortran,Bash,Csh
C++,Java,Groovy,Python,Fortran,Bash,Csh
C++,Java,Python,Fortran,Perl
C++,Python,Fortran,Bash,Csh
C++,Python,Fortran,Csh,Perl
Java,Groovy,Fortran
```

A common analysis algorithm repository must support ***everyone***

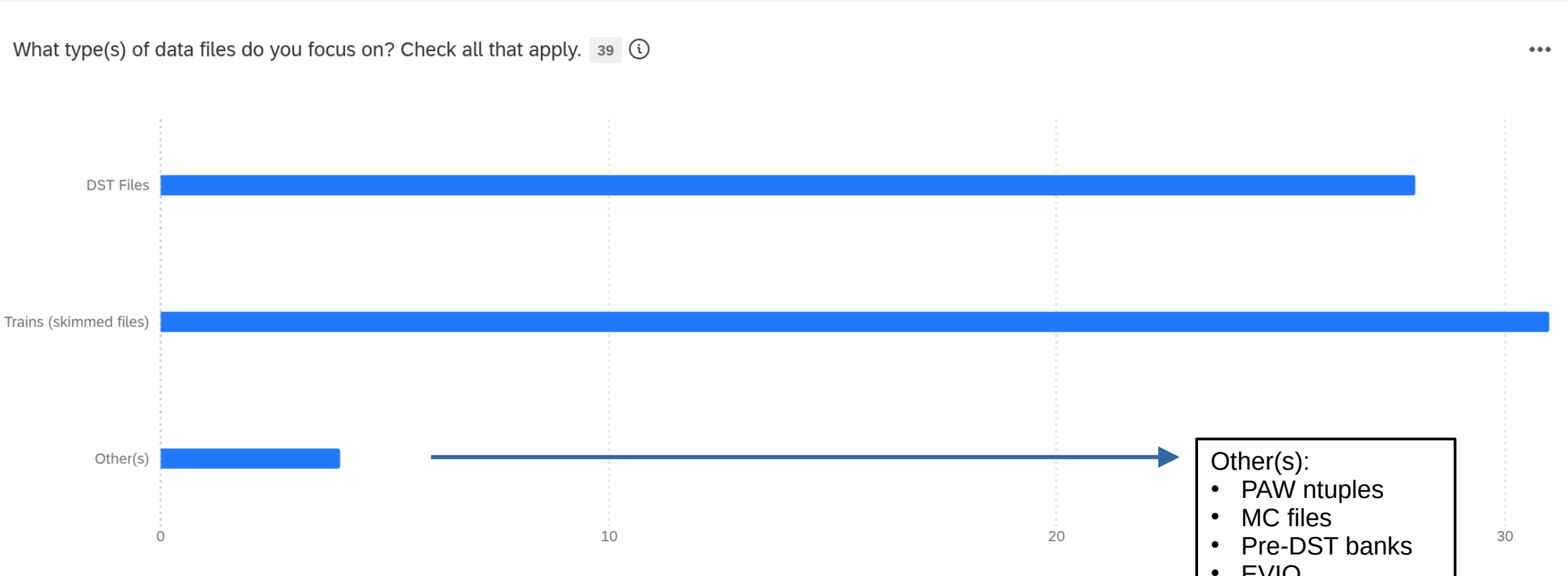
Survey Results

(all responses, one per line)

What languages do you use? Check all that apply.

C++
C++
C++
C++
C++
C++,Bash
C++,Bash
C++,Csh
C++,Groovy
C++,Java
C++,Java,Groovy,Bash
C++,Java,Groovy,Bash,Csh
C++,Java,Groovy,Python
C++,Java,Groovy,Python,Bash,Csh,Perl
C++,Java,Groovy,Python,Bash,Csh,Perl
C++,Java,Groovy,Python,Bash,Ruby
C++,Java,Groovy,Python,Fortran,Bash,Csh
C++,Java,Groovy,Python,Fortran,Bash,Csh
C++,Java,Python
C++,Java,Python
C++,Java,Python,Bash
C++,Java,Python,Bash
C++,Java,Python,Bash,Csh
C++,Java,Python,Csh
C++,Java,Python,Fortran,Perl
C++,Java,Python,Perl
C++,Python
C++,Python
C++,Python
C++,Python,Bash
C++,Python,Bash
C++,Python,Bash
C++,Python,Bash
C++,Python,Bash
C++,Python,Bash,Csh
C++,Python,Bash,Csh,Perl,zsh
C++,Python,Csh
C++,Python,Fortran,Bash,Csh
C++,Python,Fortran,Csh,Perl
Java,Groovy,Fortran

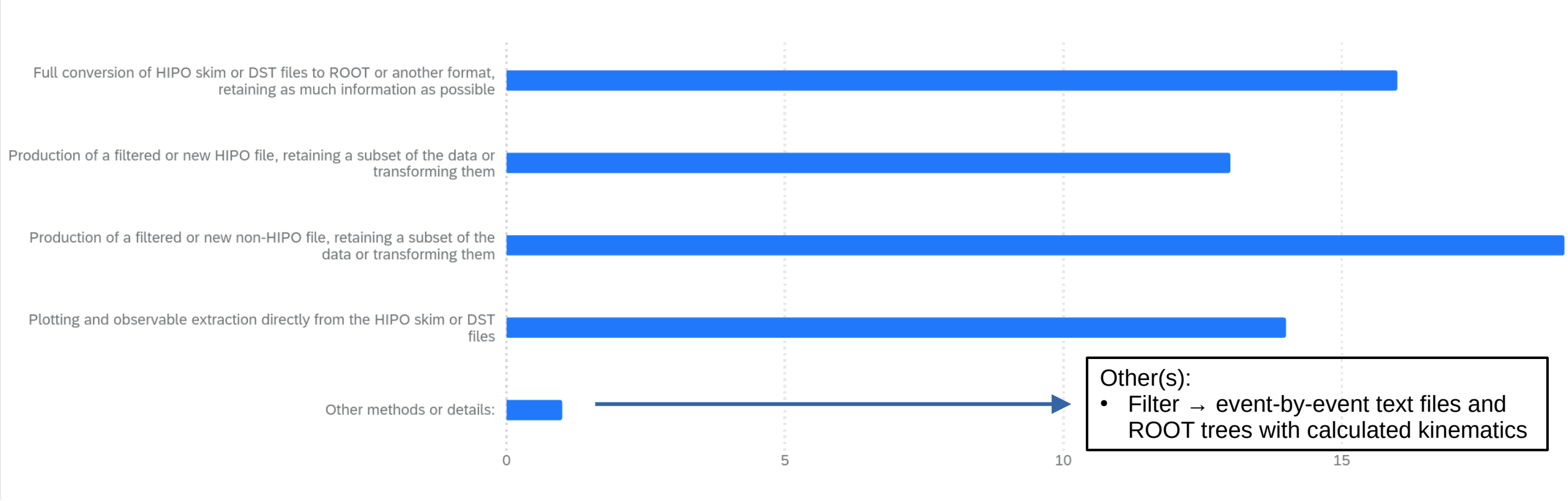
Survey Results



Other(s):
- PAW ntuples
- MC files
- Pre-DST banks
- EVIO

Survey Results

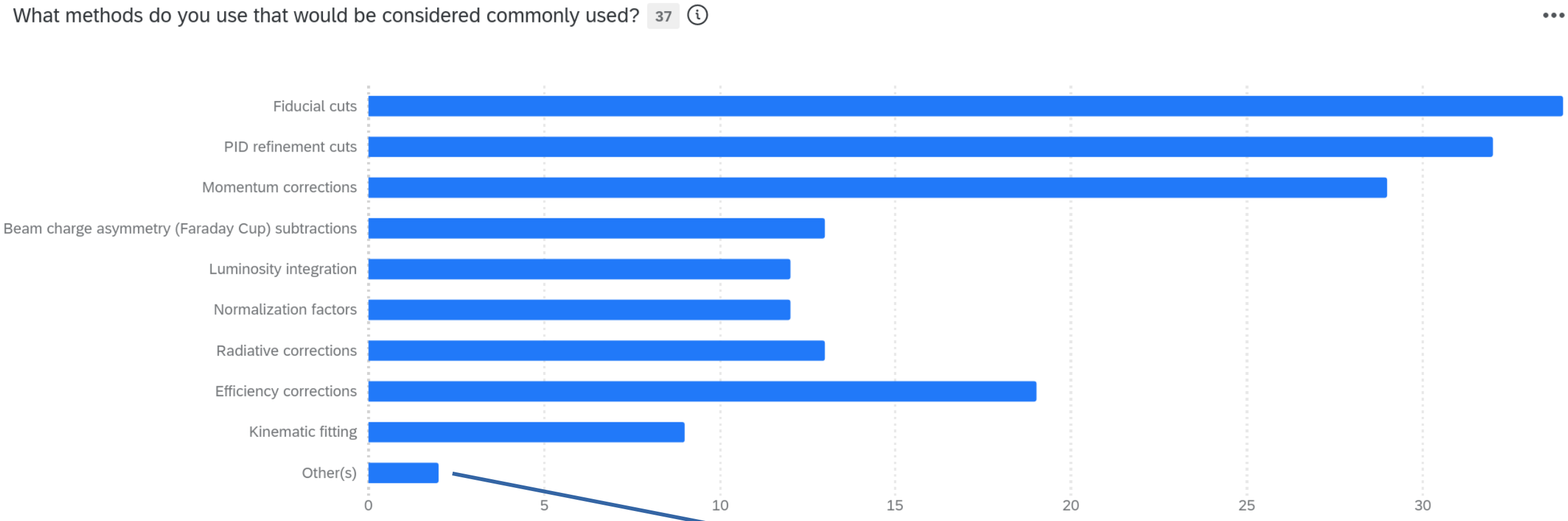
Do you use these data files directly, or do you convert them to another format? If you convert them, what information is retained: the full information, or a filtered or transformed subset? Check all that apply.



Other(s):
- Filter → event-by-event text files and ROOT trees with calculated kinematics

Jefferson Lab

Survey Results



Other(s):
- Momentum Smearing for MC/Data matching

Jefferson Lab

Survey Results

Do you maintain any commonly used methods? If so, please describe, and include what language(s) are used and/or where the code can be found.

Summary of responses

- RG-A methods maintained in Chanser
- Fiducial Cuts: in C++, Python, Groovy, Java, (and Fortran?)
- PID refinements
- BANDsoft tools
- RG-M tools
- CLAS6
- QA

DRY:
Don’t Repeat Yourself!

Survey Results

If you separate your analysis into "stages", briefly name the stages

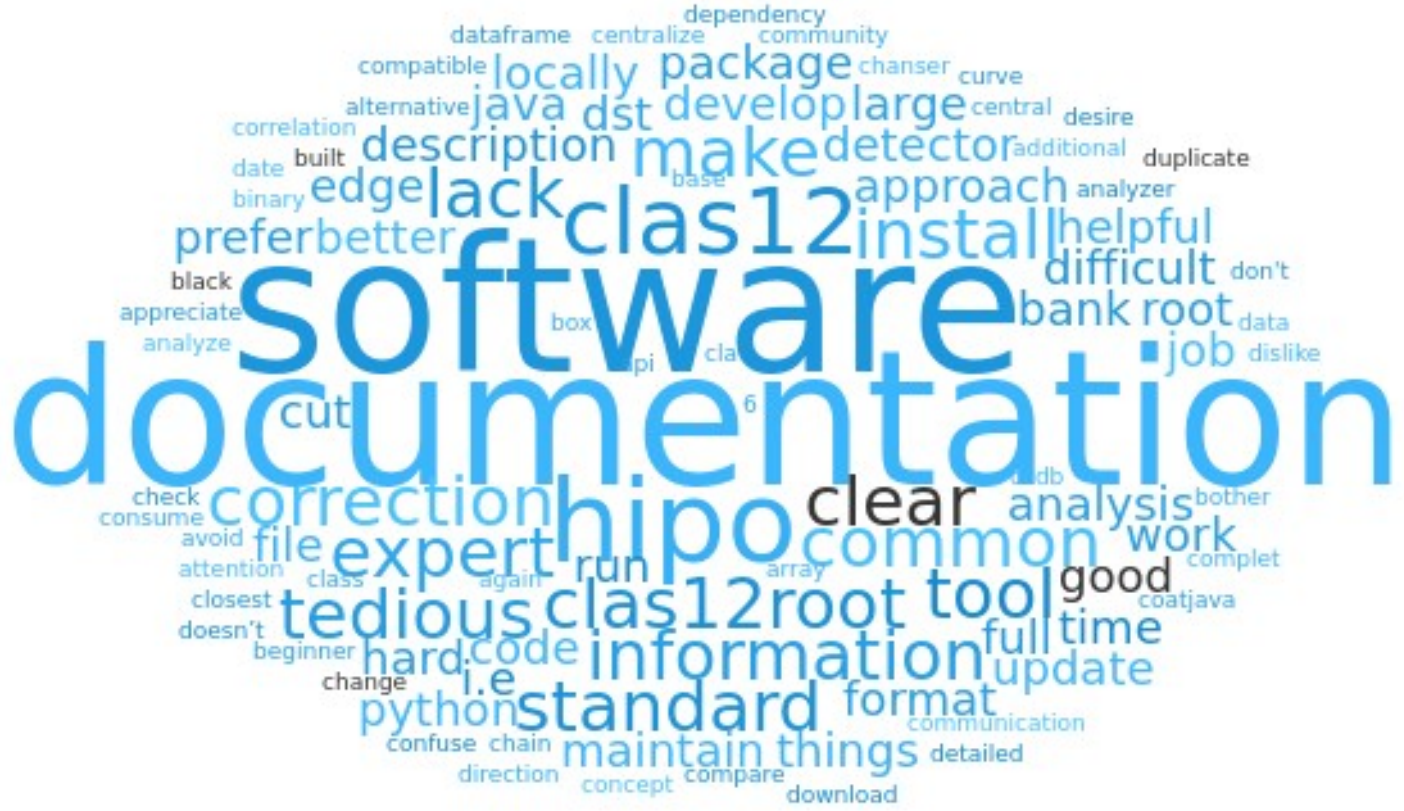
Summary of responses

- Select by PID
  - Combinatorics, topology selection
- Filtering
  - Criteria
  - Corrections
- Repeat for MC
- Handle BG
- Extract Observable

- Often iterative

Survey Results

What do you dislike about CLAS12 software? Is there anything you have found to be tedious or annoying? Any particular stage(s) from the previous question? Is there anything you would like to see changed?



Jefferson Lab

Survey Results

What do you dislike about CLAS12 software? Is there anything you have found to be tedious or annoying? Any particular stage(s) from the previous question? Is there anything you would like to see changed?

Summary of responses

- **Documentation outdated, not clear, not centralized, need common examples – no clear entry point**
- Better API to read HIPO
- Corrections built-in to trains
- Python + data frames preferred – lack of data frame support in HIPO
- Handling ragged edge arrays (awkward arrays)
- Hard to install locally, not flexible – prefer to run small tasks and testing on local machine
- clas12root – steep learning curve – but without it, it would be “tedious” to study HIPO files
- Prefer ROOT over CLAS12 Java software
- Great compared to CLAS at 6 GeV
- Lack of unified software with procedures (PID, corrections); closest is Chanser
- Lack of communication between run groups and analyzers about tools – wheel reinvention
- Simulation work is tedious – changes require expert involvement
- Software is becoming “black box” and not educating students
- Too large variety of repeated tools
- Prefer more info in HIPO rather then in databases (CCDB, RCDB)

Survey Results

Do you use any machine learning techniques? If so, what language(s) do you use, and what types of architectures, models, and software libraries do you use?

Summary of responses

- ROOT TMVA
- Convolutional NNs
- TensorFlow
- Keras
- PyTorch
- scikit-learn

- C++
- Python

Survey Results

Do you run any simulations? If so, do you use OSG, ifarm, or any other resources?

Summary of responses

- Most respondents do
- Most use OSG and/or ifarm
  - “OSG is working nicely. Congratulations.”
- Small simulations on local resources

Survey Results

What are your thoughts on the idea presented in the above mission statement? How would you like to interact with such a repository? What features would you like?

And some critical thoughts:

- Difficult to create one-size-fits-all methods
- Channel / observable / run period / analysis dependence is difficult
- Do not be opaque, black box
  - Stifle innovation
  - Does not educate students
  - May overlook a major issue in the code
- Do not force a framework, should be flexible
- Preference to do things themselves

Mostly positive feedback!

- Analyses will need to test and adapt
- Website interaction desired
- Easier than searching through analysis notes
- One language
- Peer review
- Why not apply corrections during reconstruction?
- Compatibility with C++/ROOT/Chanser/etc.
- Run period dependence
- Ability to customize
- Executable on ifarm
- Up-to-date documentation
- Examples
- Easy for new users
- Kinematic calculations (e.g. particle → z, phi, etc.)
- Polarization from closest Moeller measurement
- C++ / Java / Python

Repository Design

Survey Results

Do you maintain any commonly used methods? If so, please describe, and include what language(s) are used and/or where the code can be found.

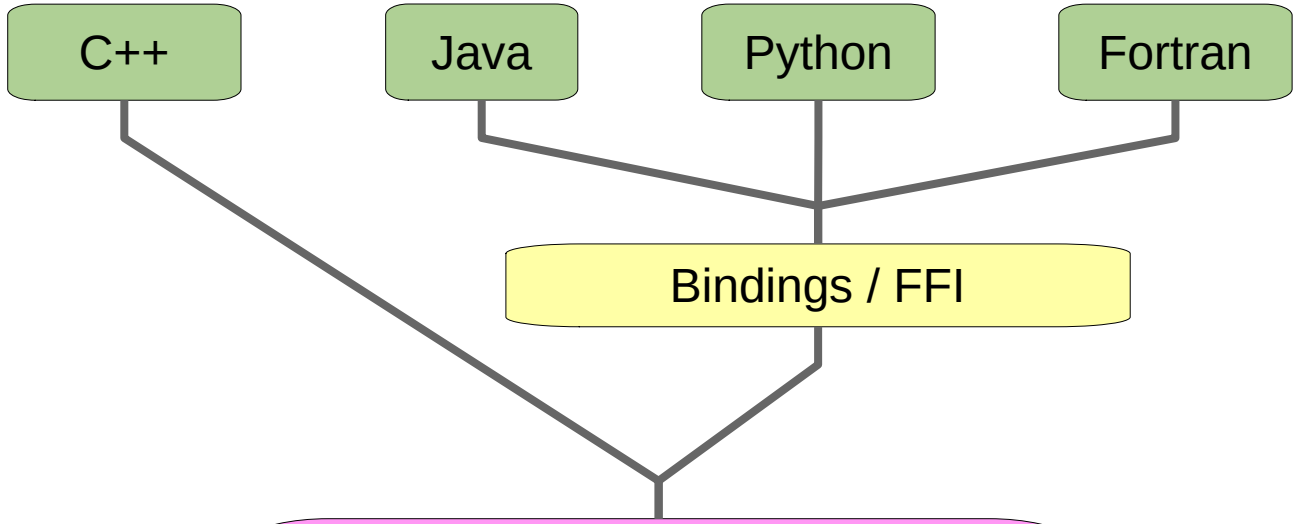
- RGM methods
- Ports of Fiducial cuts from C++ to:
  - Python
  - Groovy
  - Java
  - Fortran
- Common RGA methods in Chanser

- **Issue**: ports and code duplication
  - DRY: Don't Repeat Yourself!
  - If the C++ fiducial cuts are updated, who updates the ports?
  - Are the ports cross checked?
  - Automated testing?

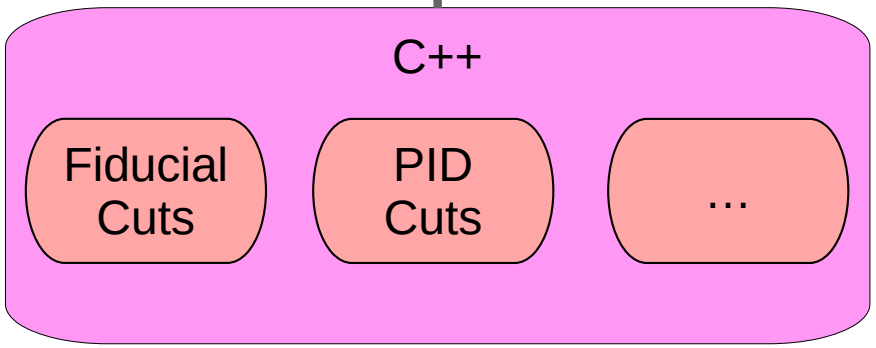
- **Chanser**
  - Includes RGA common methods
    - Fiducial cuts
    - PID refinements
    - Vertex cuts
    - (maybe more)
  - Dependent on ROOT and clas12root (?)
  - C++
- Our goal for the common repository differs:
  - Modularity: stay lightweight and as framework-independent as possible

Dominant Language Model

API Level:
For the Users

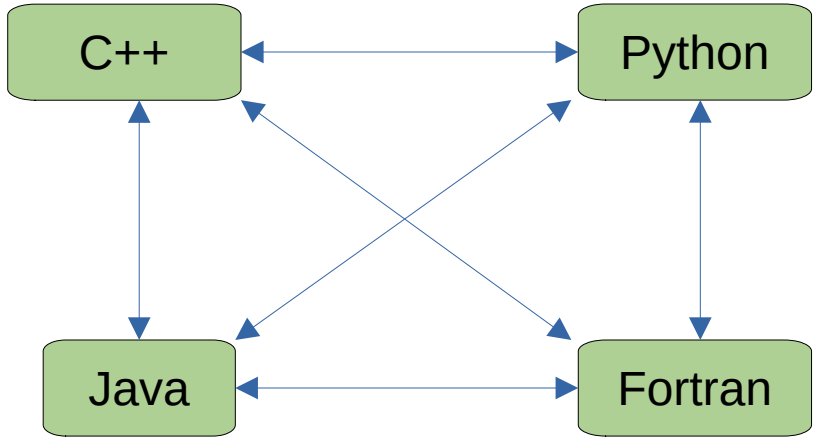


Criteria “Algorithm” level:
The Code



- Require all criteria (algorithms) to be in one “dominant” language, e.g., C++
- Consistent and maintainable
- If an algorithm is not in the dominant language, either:
  - Port it to the dominant language
  - Write a wrapper algorithm in the dominant language
- Use bindings / foreign function interfacing to expose API in other languages

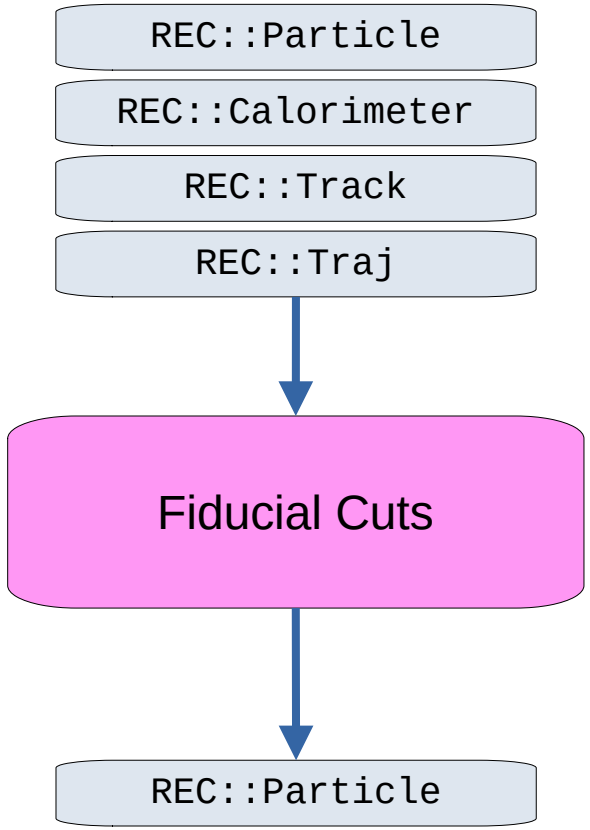
Alternative: Free Model



- Allow algorithms to be in any language
  - No need to port or wrap any existing algorithms / criteria
- Need bidirectional bindings between all of them
  - 4 languages → 8 bindings
- Hard to implement
- Hard to maintain
- **Prefer Dominant Language Model**

Data Communication

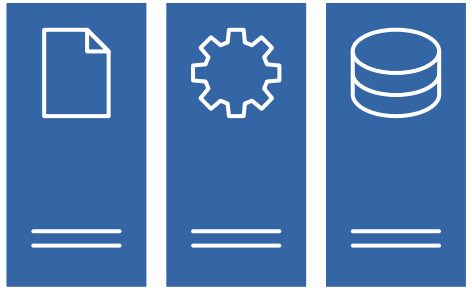
- Need a standard of communication of information
  - Users ↔ Algorithms
  - Algorithms ↔ Other Algorithms
  - For full generality, algorithm I/O should be banks
- HIPO data unit: HIPO bank
  - Need bidirectional converters from the analysis “user” language to the dominant language (C++)
  - Exploring ideas of “language independent banks” or data structures
- Alternative: structs with specific information
  - Pros:
    - better compatibility with clas12root caching
    - simpler to implement and use
  - Cons
    - the user has to fill the struct, correctly
    - output is not consistently handled: boolean vs. correction factor vs …



- with only particles which pass the fiducial cuts
- maybe different name, but same structure

Services

- The algorithms will all have some basic common needs: “service singletons”
  - **Logging system**
    - Log-level control
    - Silence for production, verbose for debugging
    - Errors always print
  - **Unit system**
    - Define what is “1” in each system
    - For example, in Geant4: 1 = mm = MeV = ns
  - **Algorithm configuration**
    - For example: fiducial cut levels (loose, medium, tight)
    - Configuration file model
      - Default config file: the defaults for *all* algorithms
      - Handle run-period dependent configuration
      - Users may override any part (or all) of it with custom config files



Testing

- Needed to maintain stability
- Some Options for automated testing:
  - Unit tests, requiring high coverage
  - clas12-validation: automated testing of full chain
    - event generation → simulation → reconstruction → analysis
    - no analysis step yet
    - https://github.com/JeffersonLab/clas12-validation
- Need also cross checks / peer review of algorithms

Documentation

- We have analysis notes
- The algorithm itself, although maybe hard to read, is effectively self-documenting
  - Comment your code!
  - Version control → algorithm is preserved
- Documentation of common repository usage is a separate issue
  - API documentation
  - Examples

Containerization

Multi-lingual support → difficult to setup (compile) for users!
Too many dependencies!

- Provide a Docker image with all dependencies + the common criteria repository, compiled and ready to use
  - Analysis code would run in containers, either locally or on clusters (*ifarm,* OSG)
- Customization:
  - Straightforward to replace software with no dependents
  - Replacing upstream software may require recompilation of dependent software
  - Adopt upstream package manager (e.g., Spack)
- Continuous Deployment: most recent version
  - Combined with a package manager makes replacing any piece of software an automated process
- Maintenance: everyone gets the same bugs

APPTAINER

docker

docker hub

Base image Layer

- Underlying Linux distribution
- Package updates
- Typical common software, e.g., vim, emacs
- Python, C++, Java, Groovy, Fortran

Maintained by JLab

Common Physics Software Layer

- ROOT, PAW
- Geant4

Maintained by JLab

CLAS Software Layer

- Clas12root
- Chanser
- Brufit
- … … …
- Common criteria repository

Maintained by CLAS

See Brad's Talk

Outlook and Plans

- Focus prototype design on:
  - Run Group A
  - Fiducial Cuts
  - PID Refinements
- Need maintainers of common methods
  - …Eventually… after the design and prototyping phase
- Anyone want to help test and design?
  - Service work opportunity?

… and fill out the survey if you haven't!

backup

Pseudocode Prototyping

```cpp
// ANALYSIS PSEUDOCODE: C++ version
// -------------------------------

#include <CommonAnalysisCriteria.h>

// initialize
auto Criteria = CommonAnalysisCriteria();

// event loop pattern
for(event : events) {
  bankFiducialResult = Criteria.FiducialCuts(event.getBank("bank1"), event.getBank("bank2"));
}

// data frame pattern (assuming the elements of the frame are banks)
auto dataFrameFiducial = dataFrame.Define(
    "bankFiducialResult",
    [] (bank1, bank2) { return Criteria->FiducialCuts(bank1, bank2); },
    {"bank1", "bank2"}
    );
```

- Banks are in the analysis code's language
- CommonAnalysisCriteria is
  - In C++: the main class
  - In Python: the main class, wrapping the C++ algorithms (needs some thought how to design…)

```python
# ANALYSIS CODE: python version
# -----------------------------

import CommonAnalysisCriteria

# initialize
criteria = CommonAnalysisCriteria.CommonAnalysisCriteria()

# event loop pattern
for event in events:
    bankFiducialResult = criteria.FiducialCuts(event.getBank("bank1"), event.getBank("bank2"))
```

Pseudocode Prototyping

```
// API CODE: C++ version
// ---------------------
outputBankType FiducialCuts(inputBank1Type bank1, inputBankType bank2) {
  // convert input C++ banks to JSON
  auto json1 = BankToJSON(bank1);
  auto json2 = BankToJSON(bank2);
  // call algorithm
  auto jsonOut = FiducialCutsAlgorithm->Process(json1, json2);
  // convert output back to a C++ bank
  return JSONToBank(jsonOut);
}
```

- The API code will handle the conversion from the analysis code banks to language-independent banks, and call the appropriate underlying algorithm
- These API methods could be auto-generated
- Assumes JSON is the “language independent bank” (needs some thought and testing)

```
# API CODE: python version
# ------------------------
def FiducialCuts(bank1, bank2):
    # convert input python banks to JSON
    json1 = BankToJSON(bank1)
    json2 = BankToJSON(bank2)
    # call algorithm; e.g., a SWIG wrapper of the underlying C++ algorithm
    jsonOut = FidicualCutsAlgorithm.Process(json1, json2)
    # convert output back to a python bank
    return JSONtoBank(jsonOutput)
```

Pseudocode Prototyping

```
// THE ALGORITHM
// -------------
class FiducialCutsAlgorithm {
  public:

    FiducialCutsAlgorithm() { /* initialize services */ }

    // run before any events
    void Init(std::string configFile="") {
      // configuration (if specified an override)
      // initialize anything that needs it
    }

    // run on every event
    outputJsonType Process(inputJson1Type json1, inputJson2Type json2) {
      // the fiducial cuts algorithm
    }

    // run at the end of all events
    void End() {
      // cleanup
    }
}
```

- The algorithm itself follows the typical 3-methods pattern:
  - Init
  - Process
  - End
- A main CommonAnalysisCriteria can handle
  - Service initialization
  - Algorithm configuration
  - Cleanup at the end