Hall A Analysis Software & Computing Update

Ole Hansen

Jefferson Lab

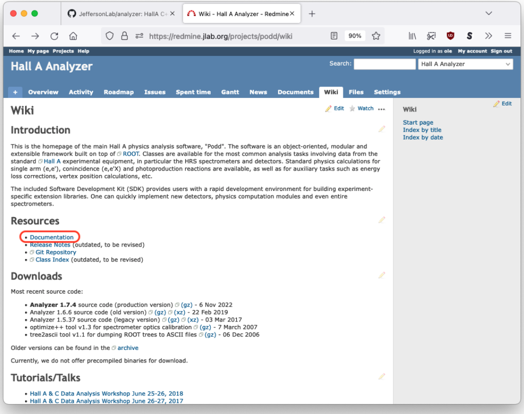
Hall A Winter Collaboration Meeting
January 26, 2023

Core Software: Podd Event Processing Framework

- C++ class library built on top of ROOT. Steering via ROOT interpreter.
- Developed in-house. Standard choice for Hall A reconstruction & analysis since 2003.
- Shared development with Hall C since 2012 ("hcana").
- Documentation & bug tracker in Redmine. Sources on GitHub.
- Strengths
  - Highly modular to accommodate frequently changing experimental setups.
  - Intuitively conceptualizes analysis in terms of physical apparatuses (spectrometers, detectors) and physics calculations (kinematics, energy loss corrections, etc.)
  - Light-weight: minimal dependencies, small memory footprint.
  - Output & cuts run-time configurable via text files. Flat text file database.
- Limitations
  - Currently still single-threaded.
  - Designed for one-pass analysis: EVIO raw data → n-tuple-like ROOT trees + histograms
- Requirements
  - Linux or macOS
  - ROOT 6
  - CMake 3. C++11 compiler. (ROOT 6.26+ requires C++14.)

Podd Documentation & Source Code

JLab Redmine


JeffersonLab/analyzer: HallA C+
Wiki - Hall A Analyzer - Redmine
https://redmine.jlab.org/projects/podd/wiki
90%
Home My page Projects Help
Logged in as ole My account Sign out
Hall A Analyzer
Search:
Hall A Analyzer
+ Overview Activity Roadmap Issues Spent time Gantt News Documents Wiki Files Settings
Edit Watch
Wiki
Start page
Index by title
Index by date

Wiki

Introduction

This is the homepage of the main Hall A physics analysis software, "Podd". The software is an object-oriented, modular and extensible framework built on top of ROOT. Classes are available for the most common analysis tasks involving data from the standard Hall A experimental equipment, in particular the HRS spectrometers and detectors. Standard physics calculations for single arm (e,e'), conincidence (e,e'X) and photoproduction reactions are available, as well as for auxiliary tasks such as energy loss corrections, vertex position calculations, etc.

The included Software Development Kit (SDK) provides users with a rapid development environment for building experiment-specific extension libraries. One can quickly implement new detectors, physics computation modules and even entire spectrometers.

Resources

- Documentation
- Release Notes (outdated, to be revised)
- Git Repository
- Class Index (outdated, to be revised)

Downloads

Most recent source code:

- **Analyzer 1.7.4** source code (production version) (gz) - 6 Nov 2022
- Analyzer 1.6.6 source code (old version) (gz) (xz) - 22 Feb 2019
- Analyzer 1.5.37 source code (legacy version) (gz) (xz) - 03 Mar 2017
- optimize++ tool v1.3 for spectrometer optics calibration (gz) - 7 March 2007
- tree2ascii tool v1.1 for dumping ROOT trees to ASCII files (gz) - 06 Dec 2006

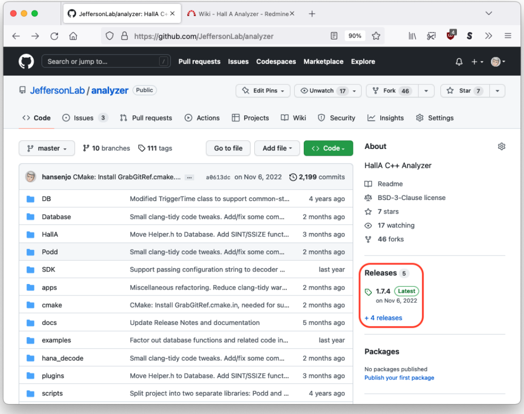
Older versions can be found in the archive

Currently, we do not offer precompiled binaries for download.

Tutorials/Talks

- Hall A & C Data Analysis Workshop June 25-26, 2018
- Hall A & C Data Analysis Workshop June 26-27, 2017

GitHub


JeffersonLab/analyzer: HallA C+
Wiki - Hall A Analyzer - Redmine
https://github.com/JeffersonLab/analyzer
90%
Search or jump to...
Pull requests Issues Codespaces Marketplace Explore
JeffersonLab / analyzer Public
Edit Pins
Unwatch 17
Fork 46
Star 7
Code Issues 3 Pull requests Actions Projects Wiki Security Insights Settings
master
10 branches
111 tags
Go to file
Add file
Code

hansenjo CMake: Install GrabGitRef.cmake.... a0613dc on Nov 6, 2022 2,199 commits

DB	Modified TriggerTime class to support common-st...	4 years ago
Database	Small clang-tidy code tweaks. Add/fix some com...	2 months ago
HallA	Move Helper.h to Database. Add SINT/SSIZE funct...	3 months ago
Podd	Small clang-tidy code tweaks. Add/fix some com...	2 months ago
SDK	Support passing configuration string to decoder ...	last year
apps	Miscellaneous refactoring. Reduce clang-tidy war...	2 months ago
cmake	CMake: Install GrabGitRef.cmake.in, needed for su...	2 months ago
docs	Update Release Notes and documentation	5 months ago
examples	Factor out database functions and related code in...	last year
hana_decode	Small clang-tidy code tweaks. Add/fix some com...	2 months ago
plugins	Move Helper.h to Database. Add SINT/SSIZE funct...	3 months ago
scripts	Split project into two separate libraries: Podd and ...	4 years ago

About

HallA C++ Analyzer

- Readme
- BSD-3-Clause license
- 7 stars
- 17 watching
- 46 forks

Releases 5

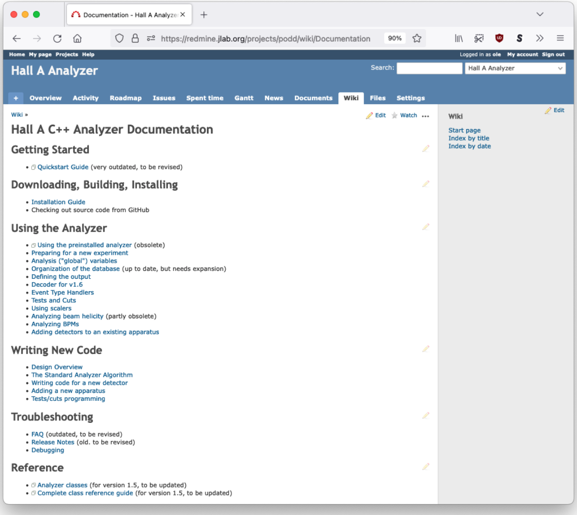
1.7.4 Latest
on Nov 6, 2022

+ 4 releases

Packages

No packages published
Publish your first package

Podd Documentation



Hall A C++ Analyzer Documentation

Getting Started

- Quickstart Guide (very outdated, to be revised)

Downloading, Building, Installing

- Installation Guide
- Checking out source code from GitHub

Using the Analyzer

- Using the preinstalled analyzer (obsolete)
- Preparing for a new experiment
- Analysis ("global") variables
- Organization of the database (up to date, but needs expansion)
- Defining the output
- Decoder for v1.6
- Event Type Handlers
- Tests and Cuts
- Using scalers
- Analyzing beam helicity (partly obsolete)
- Analyzing BPMs
- Adding detectors to an existing apparatus

Writing New Code

- Design Overview
- The Standard Analyzer Algorithm
- Writing code for a new detector
- Adding a new apparatus
- Tests/cuts programming

Troubleshooting

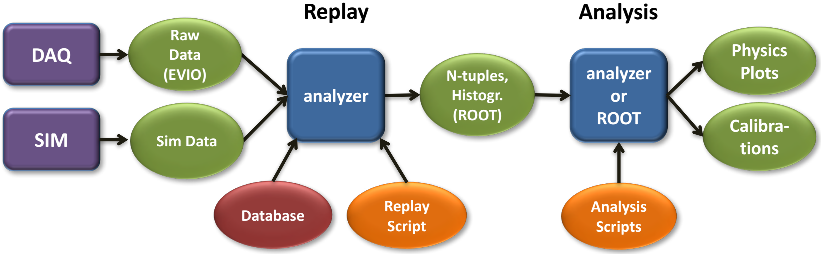
- FAQ (outdated, to be revised)
- Release Notes (old. to be revised)
- Debugging

Reference

- Analyzer classes (for version 1.5, to be updated)
- Complete class reference guide (for version 1.5, to be updated)

- Some sections outdated/obsolete.
- Newer features not yet documented.
- A User's Guide and a formal publication would be nice.

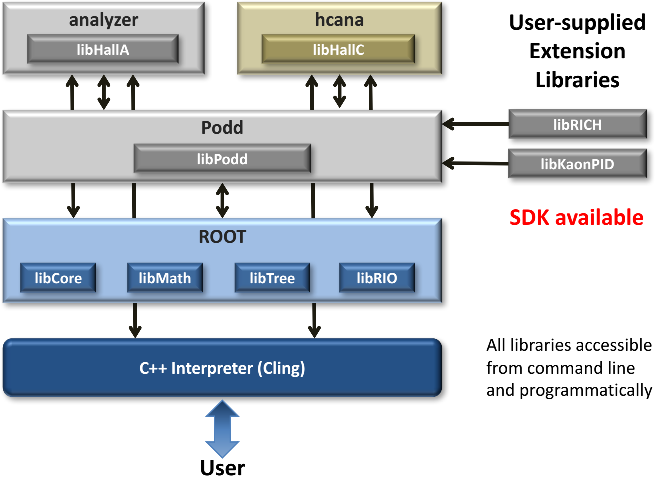
Reconstruction & Analysis Workflow



1. **Reconstruction (Replay)**
   - Runs in ROOT interpreter (`analyzer` prompt)
   - Calls mostly Podd functions & classes
   - Scripts set up by experiment experts or advanced users
   - After setup, runs in mass replay on the farm
2. **Analysis**
   - Also runs in ROOT interpreter (`analyzer` prompt)
   - Calls mostly ROOT functions and classes (but may need Podd classes)
   - Done by everyone on the experiment
   - Calibration and final physics usually done here

Podd Modular Architecture

- User interface: ROOT prompt (C++ interpreter)
- All loaded libraries (ROOT, Podd, etc.) accessible from command prompt for scripting
- Extension libraries for experiment-specific code can be loaded dynamically
  - Software Development Kit (SDK) to get started
- Entire SBS software package implemented as such an extension



User-supplied Extension Libraries

SDK available

All libraries accessible from command line and programmatically

Hall A (and C) application software area

farm/ifarm (works in Counting House, too)

```
$ module use /group/halla/modulefiles
$ module avail
-------- /group/halla/modulefiles --------
analyzer/1.7.0          evio/5.3(default)       group.apps              python/3.11.0           root/6.26.08(default)
analyzer/1.7.4(default) evio/5.3_gcc48          hcana/0.96              root/6.22.06            sbs-offline/20211206
analyzer/1.7.4_dbg      gcc/12.2.0              panguin/20211124        root/6.26.06
...
$ module load analyzer
$ analyzer --version
Podd 1.7.4 git@Release-174-0-ga0613dca 6 Nov 2022
Built for CentOS-7 using gcc-12.2.0, ROOT 6.26/08
```

Counting House (local installation, faster, safer)

```
$ module use /adaqfs/apps/modulefiles
$ module load analyzer
$ analyzer --version
Podd 1.7.4 git@Release-174-0-ga0613dca 6 Nov 2022
Built for CentOS-7 using gcc-4.8.5, ROOT 6.24/06
```

The SDK is located in `$ANALYZER_SDK`

Podd Status & Roadmap

- Current release: 1.7.4 (6 Nov 2022)
  - Base software for SBS experiments and current Hall C `hcana`.
  - Source-level backwards compatible (mostly). Suitable for replaying older data as well.
  - Many speed improvements, CODA 3 support, etc. (see Release Notes)
- Additions in 2022 (1.7.1 – 1.7.4)
  - CODA 3 trigger supervisor bank decoder. Gives access to trigger bits etc.
  - Decoder for DAQ configuration events (event types 137/138)
  - `MultiFileRun` class. Supports transparent input from multiple run segments and event streams.
- The Next Generation: 2.0 ("real soon now", hopefully this summer)
  - Multithreading!
  - Will benefit SBS and Hall C, primarily for online replay
  - Requires `C++17` (e.g. `gcc 9+`, available on `ifarm`)
  - Existing code will need minor modifications

MultiFileRun Demonstration

multi_run_test.C (simplified)

```
#include "MultiFileRun.h"

auto run = make_unique<Podd::MultiFileRun>("e1209016_1455.evio.?.*");
run->SetDataRequired(THaRunBase::kDate);
auto st = run->Init();
st = run->Open();
run->Print();
cout << "CODA version " << run->GetDataVersion() << endl;
for( int i = 1; i <= 100; ++i ) {
    st = run->ReadEvent();
    cout << i                           // Event counter
         << " "  << run->GetStream()    // Stream index
         << "."  << run->GetSegment()   // File segment
         << ": " << evbuf[0] << " ";    // Event length
    ...
}
st = run->Close();
```

MultiFileRun Demo Output

MultiFileRun Output

```
$ ls
e1209016_1455.evio.0.0  e1209016_1455.evio.1.0
e1209016_1455.evio.2.0
$ analyzer -l -b -q multi_run_test.C
Processing multi_run_test.C...
MultiFileRun: 3 files, 3 streams
Prestart at 1
DAQ info at 2
...
DAQ info at 12
Prestart at 14
Prestart at 17
File name (wildcards):   e1209016_1455.evio.?.*
Stream 0:
e1209016_1455.evio.0.0
Stream 1:
e1209016_1455.evio.1.0
Stream 2:
e1209016_1455.evio.2.0
CODA version 3
...
```

Output (cont.)

```
Count, stream.segment, length, tag, physics event, comment
1  0.0:      4 ffd1         Prestart
2  0.0: 816880  137         DAQ info
3  0.0: 804495  137         DAQ info
4  0.0: 540177  137         DAQ info
5  0.0:      4 ffd2         Go
6  0.0:    224  137         DAQ info
7  0.0:   2598  137         DAQ info
8  0.0:   3095  137         DAQ info
9  0.0:   1750  137         DAQ info
10 0.0:   1916  137         DAQ info
11 0.0:   3078  137         DAQ info
12 0.0:   1996  137         DAQ info
13 0.0:  54249 ff70    1    Physics
14 1.0:      4 ffd1         Prestart
15 1.0:      4 ffd2         Go
16 1.0:  20084 ff70    2    Physics
17 2.0:      4 ffd1         Prestart
18 2.0:      4 ffd2         Go
19 2.0:  17687 ff70    3    Physics
20 0.0:  17520 ff70    4    Physics
21 1.0:  19286 ff70    5    Physics
22 2.0:  19857 ff70    6    Physics
...
```

Podd 2.0

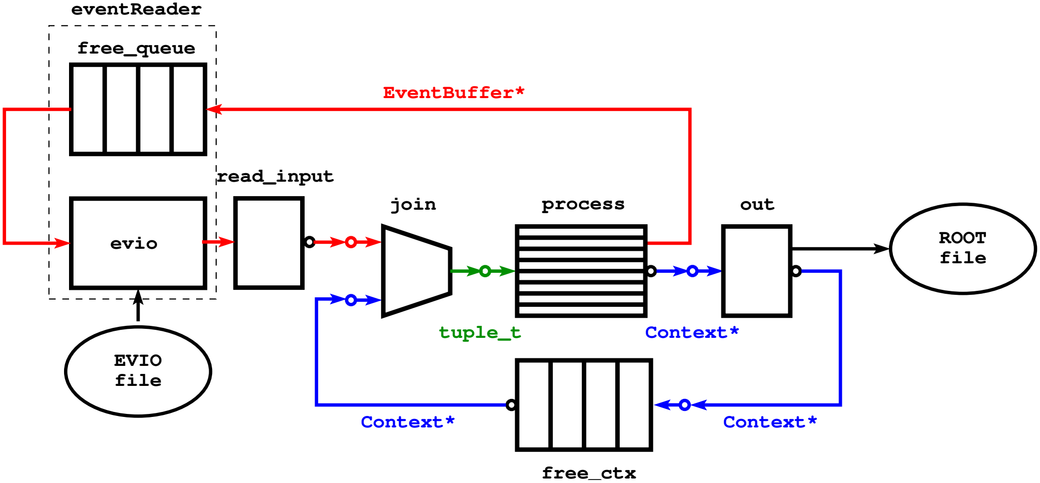
- Event-level parallelization/multithreading
  - Especially important for online replay
  - Reduced memory footprint compared to multiple individual jobs
  - Requires thread safe user code (→ only const or protected globals, statics)
- I/O improvements
  - Output system upgrade (full set of data types, object variables) — largely complete
  - EVIO 6 input format support (HIPO-like raw data files) — once EVIO 6 stable
  - Possible alternative (non-ROOT) output file formats
  - Goal: Make output easily usable with Python and Julia tools (*e.g.* `uproot`, `UnROOT`)

Goal: Multithreading & output data typing ready for SBS-GEp and Hall C NPS run

Parallel Podd Prototype

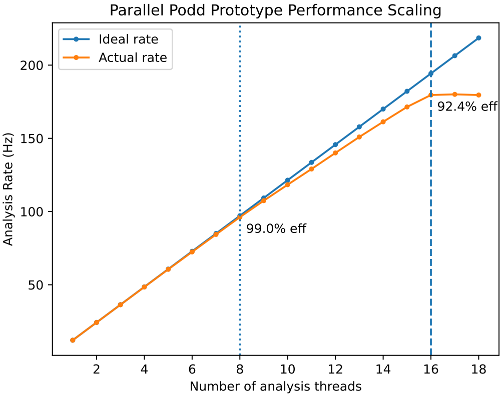
- `https://github.com/hansenjo/parallel`
- Mimics main components of Podd (*e.g.* decoder, analysis variables, output)
- A few example "detectors" included whose processing is intended to burn CPU cycles
- Uses oneAPI TBB library (formerly Intel Thread Building Blocks)
- Three processing modes:
  1. Unordered — event numbers may not be consecutive in output
  2. Ordered — consecutive event numbers guaranteed
  3. Barriers — events guaranteed to stay between special barrier events (*e.g.* scalers)
- Output serialized for now (hard to avoid because of ROOT) — potential bottleneck

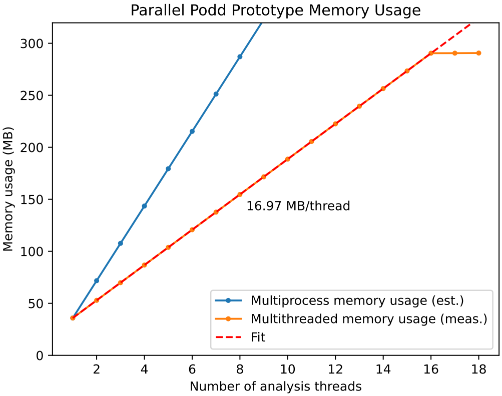
TBB-based Parallel Podd — Unordered Mode Flow Graph



TBB-based Parallel Podd Performance Scaling Benchmark

- Unordered mode. (Other modes are naturally less performant.)
- Processing rate and real memory usage (resident set size) as function of number of analysis threads.
- Test system: Intel i7-10700K (8C/16T) @ 3.80 GHz, 32 GB RAM, macOS 11, idle.
- 16 MB per-thread event buffer size for illustration purposes.

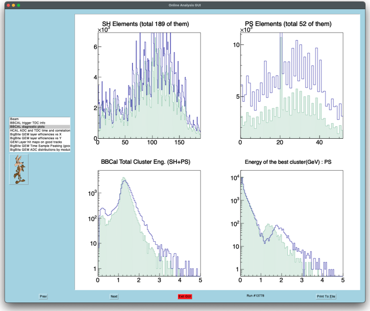




Panguin (Online GUI)

Panguin 2.5 Command Line Options

```
$ panguin --version
Panguin version 2.5 (23-Oct-2022)
$ panguin --help
panguin: configurable ROOT data visualization tool
Usage: panguin [OPTIONS]

Options:
-h,--help                   Print this help message and exit
-f,--config-file <file name> [default.cfg] Job configuration file
-r,--run <run number>       Run number
-R,--root-file <file name>  ROOT file to process
-G,--goldenroot-file <file name> Reference ROOT file
-P,-b,--batch               No GUI. Save plots to summary file(s)
-E,--plot-format <fmt>      Plot format (pdf, png, jpg ...)
-C,--config-dir <path>      Search path for configuration files & macros
                            (":"-separated)
--root-dir <path>           ROOT files search path (":"-separated)
-O,--plots-dir <dir>        Output directory for summary plots
-I,--images                 Save individual plots as images (implies -P)
-F,--image-format <fmt>     Image file format (png, jpg ...)
-H,--images-dir <dir>       Output directory for individual images
                            (default: plots-dir)
-v,--verbosity <level>      Set verbosity level (>=0)
-V,--version                Display program version information and exit
```


Online Analysis GUI
Beam
BBCAL trigger TDC info
BBCAL diagnostic plots
HCAL ADC and TDC time and correlation
BigBite GEM layer efficiencies vs X
BigBite GEM layer efficiencies vs Y
GEM Layer hit maps on good tracks
BigBite GEM Time Sample Peaking (goo
BigBite GEM ADC distributions by modul
SH Elements (total 189 of them)
PS Elements (total 52 of them)
BBCal Total Cluster Eng. (SH+PS)
Energy of the best cluster(GeV) : PS
Prev
Next
Exit GUI
Run #13778
Print To File

- New command line options for easier scripting
- Configuration files support include
- File names and directory paths expand environment variables and placeholders:

  $ROOTFILES/$EXPERIMENT_%R.root

  summaryPlots_%R_page%P_%C.%E
- See README.md for full documentation

Hall A Online Computing

- Previous: Online replay on 2014-vintage aonlX systems (128 threads)
- Due to system failure, have been down to 96 threads since September 2022.
- New server with additional 128 threads/512 GB RAM being configured (Rocky Linux 9), ready shortly

- This should meet online computing requirements through the MOLLER experimental run (2028/29).

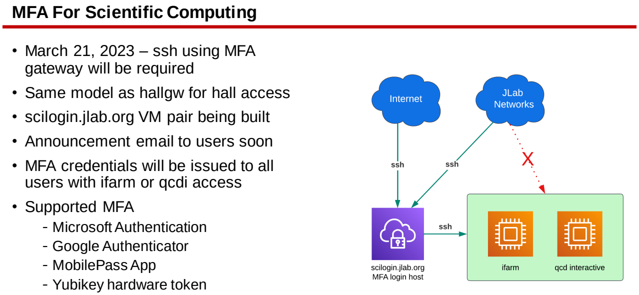
```
[a-onl@aonl5 ~]# cat /etc/redhat-release
Rocky Linux release 9.1 (Blue Onyx)

[a-onl@aonl5 ~]# lscpu|head
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Address sizes:                   48 bits physical, 48 bits virtual
Byte Order:                      Little Endian
CPU(s):                          128
On-line CPU(s) list:             0-127
Vendor ID:                       AuthenticAMD
Model name:                      AMD EPYC 7543 32-Core Processor
Thread(s) per core:              2
Core(s) per socket:              32
Socket(s):                       2
Stepping:                        1
Frequency boost:                 enabled
CPU max MHz:                     2800.0000
CPU min MHz:                     1500.0000
BogoMIPS:                        5589.69

[a-onl@aonl5 ~]# free -h
       total     used     free   shared  buff/cache   available
Mem:   502Gi    7.6Gi    494Gi   156Mi        3.6Gi       495Gi
Swap   63Gi        0B    63Gi
```

Scientific Computing Resources

- Farm/ifarm still on CentOS 7.9. Rocky 8 (RHEL 8 clone) to be rolled out this year. Increasing containerization (Apptainer/Singularity) decreases importance of host OS.
- Farm batch system running new Slurm and `swif2` job scheduler. See the Farm Users Guide.
- Farm resources
  - Disk: Lustre: 4.1 PB, Work: 1.4 PB.
  - CPU: 13192 cores / 26384 threads ≈ 160 Skylake (2018) M-core-hours/year
  - New farm23 nodes being installed (AMD "Milan"). Will raise capacity to 240 Skylake M-core-hours/year
  - 6 nodes with Nvidia TitanRTX and/or A100 GPUs dedicated for ML ("gpu" partition)
- Mass storage system (as of Jan 2023)
  - Throughput ≈ 10 GB/s (24 LTO-8 drives, uncompressed, theoretical)
  - ≈ 150 PB capacity (LTO-8, uncompressed), ≈ 97.6 PB used (22.9 production, 28.1 raw, 27.6 rawdup).
  - Significant capacity headroom (more frames, LTO-9) with current silo, up to ≈ 325 PB.

Mandatory MFA Authentication Coming to Scientific Computing

- Starting **March 21, 2023**, ifarm/lqcd hosts will require login through a multi-factor authentication gateway, as with the hall computers.
- No rationale given
- Available gateways
  - `scilogin.jlab.org`
  - `acclogin.jlab.org`
  - `hallgw.jlab.org`
- See Knowledge Base article with suggestions for convenient SSH configuration



MFA For Scientific Computing

- March 21, 2023 – ssh using MFA gateway will be required
- Same model as hallgw for hall access
- scilogin.jlab.org VM pair being built
- Announcement email to users soon
- MFA credentials will be issued to all users with ifarm or qcdi access
- Supported MFA
  - Microsoft Authentication
  - Google Authenticator
  - MobilePass App
  - Yubikey hardware token

(Slide from Bryan Hess)

Outlook

- Podd expected to be used throughout the SBS program and for MOLLER counting mode measurements.
- Similarly, upcoming Hall C experiments (NPS etc.) will use Podd/hcana for the foreseeable future.
- Significant modernization work (multithreading etc.) ongoing.
- MOLLER integrating mode experiments will use existing “japan” (parity analyzer) software
- As we have many new collaborators, we may organize an analysis workshop later this year, likely together with Hall C.