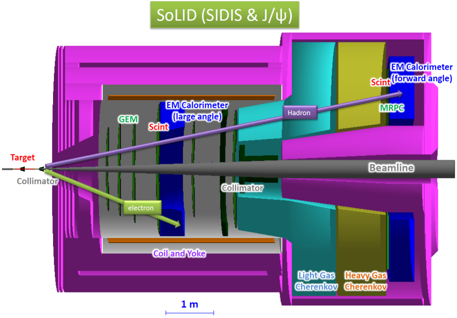
# Software design ideas for SoLID

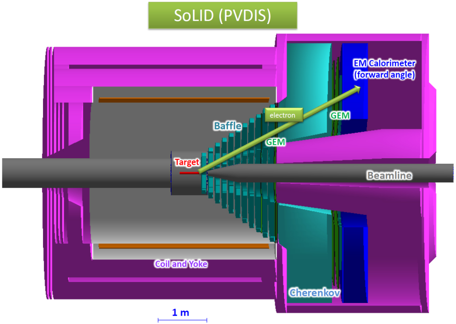
Ole Hansen

Jefferson Lab

EIC Software Meeting
Jefferson Lab
September 25, 2015

# The SoLID Experiments @ JLab




- Hall A, 11 GeV polarized beam, fixed targets ($^3\vec{\text{He}}$, $\vec{\text{NH}}_3$, $D_2$, $H_2$).
- GEM trackers (approx. 165k channels)

| Experiment | Event size (kB) | Trigger rate (kHz) | Data rate (MB/s) | Raw data (PB) |
|---|---|---|---|---|
| SIDIS | 3 | 100 | 300 | 5.6 |
| PVDIS | 50 | 20 | 1,000 $\xrightarrow{\text{HLT}}$ 300 | 7.0 |
| *cf.* GlueX | 15 | 200 | 3,000 $\xrightarrow{\text{HLT}}$ 300 | 3.2/yr |

# Choosing A Computing Model

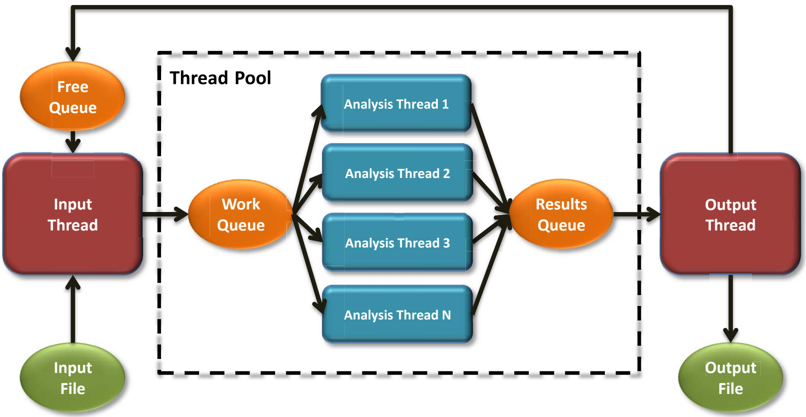
3 minute run $\rightarrow$ 18M SIDIS events, 50 GB raw data
Assume 20 ms/event $\rightarrow$ to keep up with data taking, need 2000 cores

- **Single-threaded:** no framework support for parallelism
  - 2000 runs in parallel $\rightarrow$ 100 TB disk space for input
  - $\approx$ 100 hours turn-around time per run
  - Problems: unrealistic cost & turnaround time
- **Multi-process:** parallelism through the job scheduler
  - *E.g.* 32 single-threaded jobs working on different event ranges of one run
  - 62.5 runs in parallel $\rightarrow$ 3 TB disk space for input, 3 hours/run
  - Potential problems: I/O bottlenecks (disk head thrashing), limited scalability, complexity outsourced to job scheduler
- **Multi-threaded:** event-level parallelism through modern CPU architecture
  - Similar to multi-process, but reduced random disk access & memory footprint
  - Problems: scalability limited by cores/node, code complexity
- **Distributed:** event-level parallelism through built-in scheduler
  - 1 run in real time, 0.05 TB disk space for input.
  - Virtually unlimited scalability
  - Potential problems: even more code complexity, network bottlenecks

# My Take On the Computing Model Choice

- A multi-threaded design offers
  - best performance in terms of I/O and memory use
  - reasonable compromise in terms of complexity
  - sufficient scalability for SoLID needs
- A distributed system can be built on top of a multi-threaded implementation

# Possible Multi-Threaded Architecture

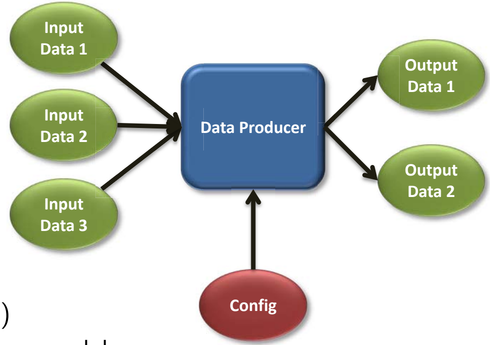



- Thread Pool with three thread-safe queues
- Queues hold working sets: event object, analysis chain & modules
- Option to sync event stream at certain events (*e.g.* scaler events, run boundaries)
- Option to preserve strict event ordering (at a performance penalty)

# Some General Considerations

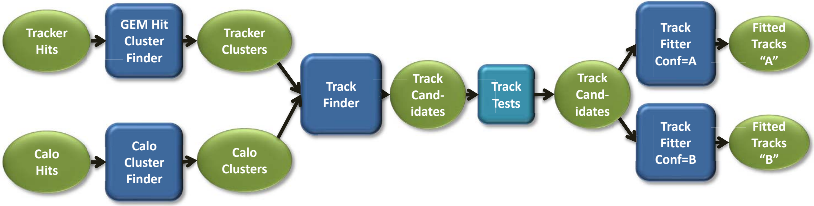
- Maximize consistency: Framework should support all of simulation, digitization, reconstruction and physics analysis
- Must support multi-pass processing: output $\longrightarrow$ input for next pass
- Support multiple analysis chains per job
- DST file format not very important, but interactive analysis must be possible with ROOT
- DSTs should contain extensive metadata: database parameters from previous stages (geometry etc.), data provenance, etc.

# Generic Data Model




- Data producers (algorithms)
  - Ideally, single algorithm per module
  - Run-time configurable
  - Must be reusable without recompilation → multiple instances allowed, differing in configuration
- Data objects (results)
  - transient or persistent
  - separate from producers
  - may reference other data objects
  - should hold metadata about their origin

# Analysis Chains




- Modules communicate only via data objects
- Module relationships partly configurable at run time (select input from one of multiple instances of a data object)
- Support condition testing modules. Select subset of results and/or skip further processing if certain tests fail or succeed.
- May have multiple chains per job

# Simulation Support

- MC truth info available in data objects
- Digitized data objects contain references to truth info (hits, tracks, particles) that generated them
- Support for embedding hits from MC tracks in real data for efficiency calculations → job of event source module

# Conclusions

- SoLID computing challenges are similar to those of CLAS12 and GlueX: 5–7 PB of data per physics topic, requiring massively parallel processing
- Currently evaluating available HEP/NP frameworks
- Ideally, would like to avoid reinventing the wheel and adopt an existing one
- Joint effort with EIC development would be beneficial if sufficient overlap