My personal experience with ePIC's CI/CD workflow

Nathan Brei

Sept 19, 2025

ePIC Software Review Committee

Agenda

1. Context
2. Compatibility requirements
3. Code quality checks
4. Benchmarking and validation
5. Recommendations

Context

Context

- I'm a computer scientist, not a physicist
- I've been working on the JANA2 reconstruction framework since I joined JLab in 2019
- I've been the lead developer of JANA2 ever since David became too busy
- Much of my work has been driven by requirements and desiderata from ePIC
- However, I **rarely** contribute code directly to EICrecon
- Improving ePIC's overall developer experience requires us taking a systematic approach
- That is **not** what this is
- This is merely one **very opinionated** data point on an undulating surface

Compatibility requirements

Compatibility requirements

Idea

- EICrecon's Github CI builds against a very thorough test matrix. It is roughly like so:

$$\begin{aligned}\text{test\_matrix} = &\{\text{gcc, clang}\} \times \\ &\{\text{Debug, Release}\} \times \\ &\{(\text{None}), \text{USE\_ASAN}, \text{USE\_TSAN}, \text{USE\_UBSAN}\} \times \\ &\{\text{eic\_container:nightly}\} \\ \cup\ &(\text{clang}, \text{Release})\{\text{eic\_container:v25.01.1-stable, eic\_container:v25.05.0-stable}\}\end{aligned}$$

- Note that each `eic_container` version includes a single version of gcc, clang, JANA2, edm4eic, etc.
- The `eic_container` environment leverages Spack for reproducible package management

Compatibility requirements

Problems

- 6-month backwards compatibility requirement is a formidable obstacle to introducing new framework features
- Backwards compatibility requirement pushes dependency version logic into preprocessor directives and `constexpr if` blocks
- This also makes data model changes much harder!
- I don't see the benefit to supporting older environments, apart from nightly and the latest release.

```cpp
if constexpr (10000 * JVersion::major +
              100 * JVersion::minor +
              1 * JVersion::patch > 20403) {
    default_plugins.emplace_back("splitting");
}
```

```cpp
#if ((JANA_VERSION_MAJOR == 2) && (JANA_VERSION_MINOR >= 3))
|| (JANA_VERSION_MAJOR > 2)
#define JANA_NEW_CALLBACK_STYLE 1
#else
#define JANA_NEW_CALLBACK_STYLE 0
#endif

class JEventSourcePODIO : public JEventSource {
public:
  JEventSourcePODIO();
  ~JEventSourcePODIO() override;
  void Open() override;
  void Close() override;

#if JANA_NEW_CALLBACK_STYLE
  Result Emit(JEvent& event) override;
#else
  void GetEvent(std::shared_ptr<JEvent>) override;
#endif
```

Compatibility requirements: Effect on development workflow

- The previous examples kept the preprocessor logic in a limited area, but making a change across the entire codebase is basically impossible, e.g. removing the `app` argument from `JOmniFactoryGeneratorT`:
- The next major feature is external wiring, which by definition touches every single factory. For it to be mergeable, it effectively needs to have zero visible API changes.
- Features requested by ePIC have to be deeply tested outside of EICrecon before introduction. No co-evolution is possible.
- To compensate, I'm building a complete reconstruction chain inside the JANA examples

```cpp
app->Add(new JOmniFactoryGeneratorT<InclusiveKinematicsML_factory>(
    "InclusiveKinematicsML",
    {"InclusiveKinematicsElectron", "InclusiveKinematicsDA"},
    {"InclusiveKinematicsML"},
    app));

// Literally hundreds of these app->Add() calls across EICrecon
// A previous version of JANA2 made the extra `app` parameter
// unnecessary, but we can't remove it due to backwards compat
```

Code quality checks

Automated code quality checks: clang-tidy, etc

Idea

- Leverages heavy-duty tooling to enforce code style, correctness, and performance
  - Clang-tidy
  - `-Werror -Wall -Wextra -Wundef`
  - Asan, Tsan, UBsan
  - Profiling and code coverage

~~Problems~~

- Clang-tidy is configured quite sensibly
- Previously had problems with bugprone-easily-swappable-parameters in one of my interfaces
- Have incorporated some of these in JANA2's CI, plan on incorporating more

```plain
HeaderFilterRegex: ''
Checks: '
  bugprone-*,
  concurrency-*,
  cppcoreguidelines-*,
  modernize-*,
  portability-*,
  readability-*,
  -bugprone-easily-swappable-parameters,
  -bugprone-macro-parentheses,
  -bugprone-narrowing-conversions,
  -bugprone-unchecked-optional-access,
  -modernize-use-trailing-return-type,
  -modernize-avoid-c-arrays,
  -modernize-use-nodiscard,
  -readability-function-cognitive-complexity,
  -readability-identifier-length,
  -readability-magic-numbers,
  -readability-math-missing-parentheses,
  -cppcoreguidelines-avoid-c-arrays,
  -cppcoreguidelines-avoid-magic-numbers,
  -cppcoreguidelines-narrowing-conversions,
  -cppcoreguidelines-owning-memory
'
FormatStyle: file
```

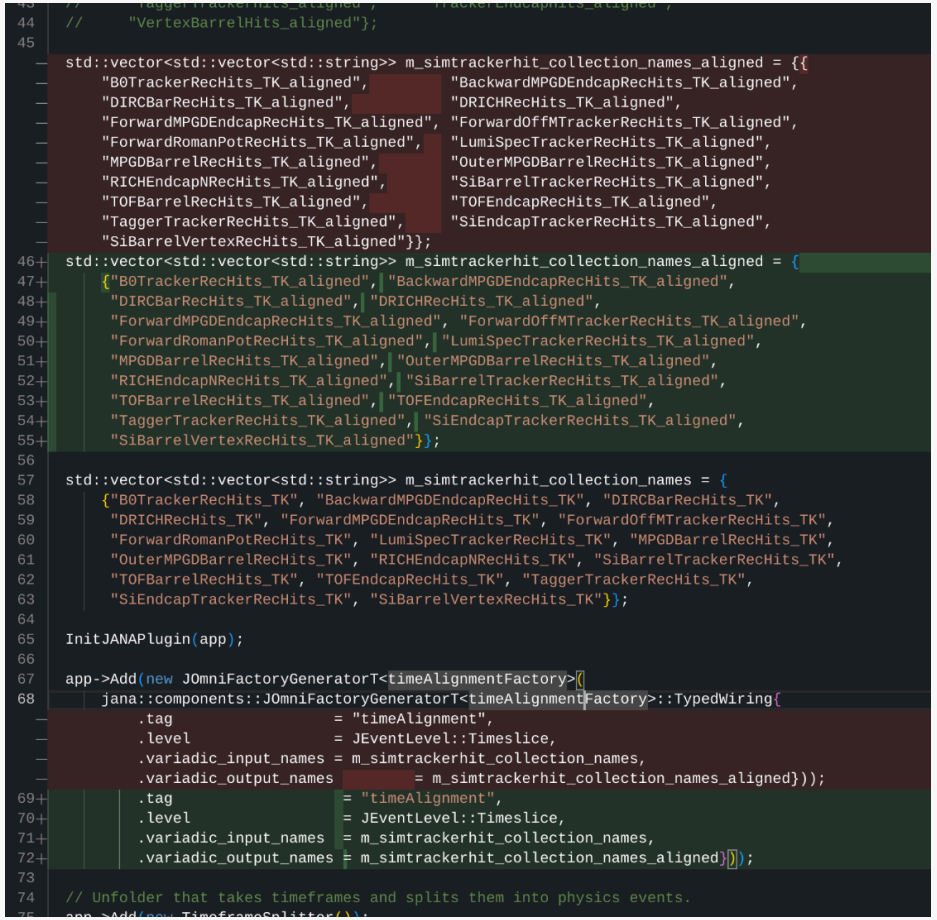
Automated code quality checks: clang-format

Idea

- Ends inconsistent and misleading formatting (e.g. mixed tabs and spaces) for once and for all

Problems

- The clang-format settings mangle the diffs
  - Lists of strings (we have a lot of these) wrap instead of having one-per-line
  - Assignments get lined up with adjacent lines
  - Can be disabled via `// clang-format off`
- The automated commits create merge conflicts
  - Diff blocks become much larger than necessary, often the entire file
  - This happens on **every** push to the PR
- The clang-format settings change frequently



Automated code quality checks: include-what-you-use

Idea

- C++'s `#include` mechanism is unhygenic
- Build times were getting too slow
  - Partially due to a large "datamodel glue" header that was being (transitively) imported everywhere: #628
  - Future versions of JANA2 won't need datamodel glue at all #446
  - Current build time is ~5 mins (depending on compiler flags)

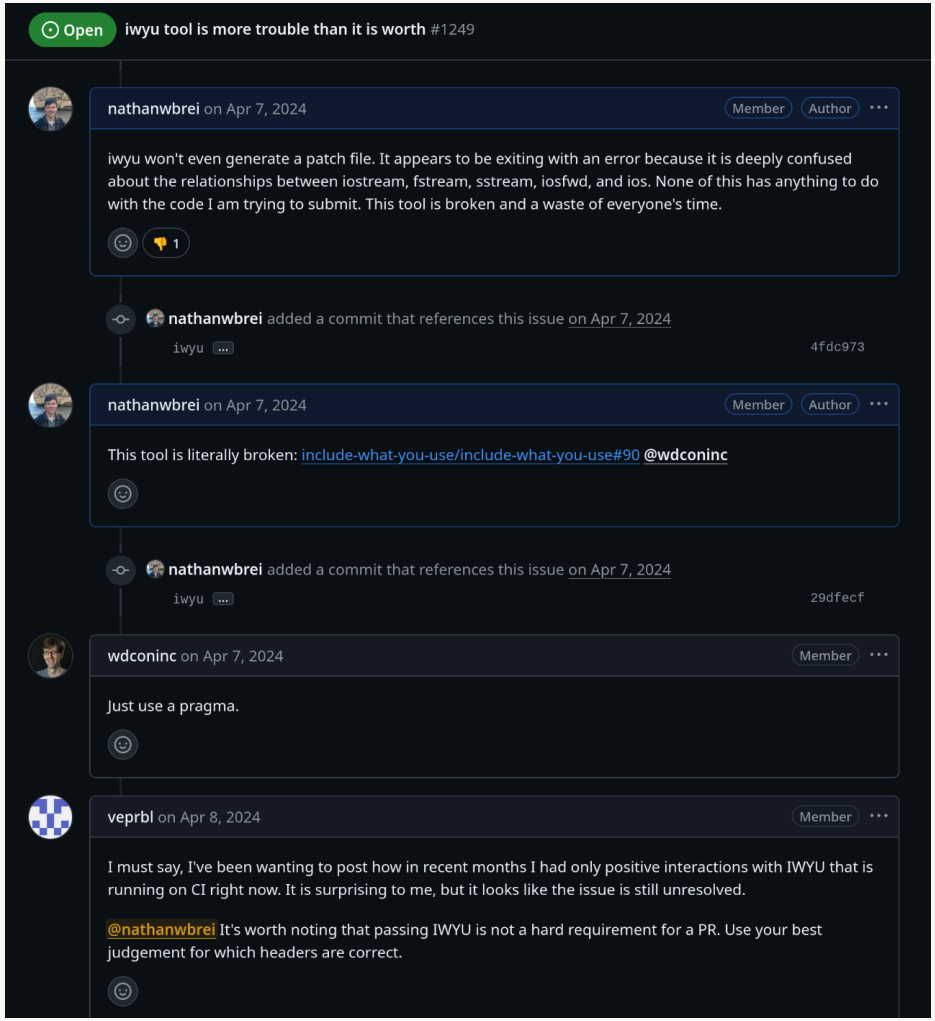
Problems, part 1

- The iwyu tool produces a patch file, rather than automatically committing its changes
- The user must navigate to the "Artifacts" tab, download the patch file, apply the patch file, commit, push, and wait
- The iwyu tool gets caught in cycles, where it rejects its own patch and generates a new one
- It's complaints are not limited to the user's own changes
  - Upgrading a dependency can cause iwyu to rearrange includes throughout the entire codebase

Automated code quality checks: include-what-you-use

Problems, part 2

- iwyu struggles to understand forward declarations...
  - **including 'iosfwd' in the standard library**
  - The iwyu developers aren't concerned: #144
  - iwyu breaks every now and then for ePIC due to changing compiler/stdlib versions: #1984
- The solution is, once again, pragmas
  - There is a rich and sophisticated family of pragmas available (link)
  - Sometimes all you need is `// IWYU pragma: keep`
  - Otherwise, you'll need deep knowledge of both C++ semantics and your dependencies' internals, and lots of trial and error



Problems, part 3

- iwyu creates problems for me as a framework developer
- Template methods are an important part of my interface, but they create problems with circular includes
- To fix these, I split the problematic headers into
  - "Foward headers" containing the class and template declarations
  - "Full headers", which import the forward header and also provide the template definitions
- iwyu insists on my users providing both!
  - Breaks the separation between my interface and internals
  - Forces me to annotate JANA2 code with iwyu pragmas

```C++
#include <DD4hep/DetElement.h>
#include <DD4hep/Detector.h>
#include <Evaluator/DD4hepUnits.h>
#include <JANA/JApplication.h>
#include <JANA/JApplicationFwd.h>
#include <JANA/Utils/JTypeInfo.h>

#include <functional>
#include <map>
#include <memory>
#include <string>
#include <vector>

#include "algorithms/digi/PhotoMultiplierHitDigiConfig.h"
#include "algorithms/pid/IrtCherenkovParticleIDConfig.h"
#include "algorithms/pid/MergeParticleIDConfig.h"
#include "algorithms/tracking/TrackPropagationConfig.h"
#include "extensions/jana/JOmniFactoryGeneratorT.h"
#include "factories/digi/PhotoMultiplierHitDigi_factory.h"
#include "factories/pid/IrtCherenkovParticleID_factory.h"
#include "factories/pid/MergeCherenkovParticleID_factory.h"
#include "factories/pid/MergeTrack_factory.h"
#include "factories/pid/RichTrack_factory.h"

#include "services/geometry/richgeo/ActsGeo.h"
#include "services/geometry/richgeo/RichGeo.h"
#include "services/geometry/richgeo/RichGeo_service.h"
```

Automated code quality checks: Effect on workflow

- I know of no way to run the CI test suites locally automatically
- Instead, I write my code against my own unit and integration tests
- Once satisfied, I download an npsim test data file from a recent CI job, and run it against EICrecon locally, until I observe no crashes
- Then I push my code and create a pull request
- Almost every single time I push, the CI adds an additional commit changing around whitespace and includes, which clutter my history and introduce merge conflicts
- PRs take several days in the best case, and up to a year in the worst case, to be merged
- As an developer who prefers rebasing and stacking small PRs, I'm instead forced to split the automated commits, squash them into my clean commits, cherry pick my clean commits on top of main, and then force-push my branch.
- Developers less comfortable with git will repeatedly merge main back into their branch and accumulate a long messy history

Benchmarking and validation

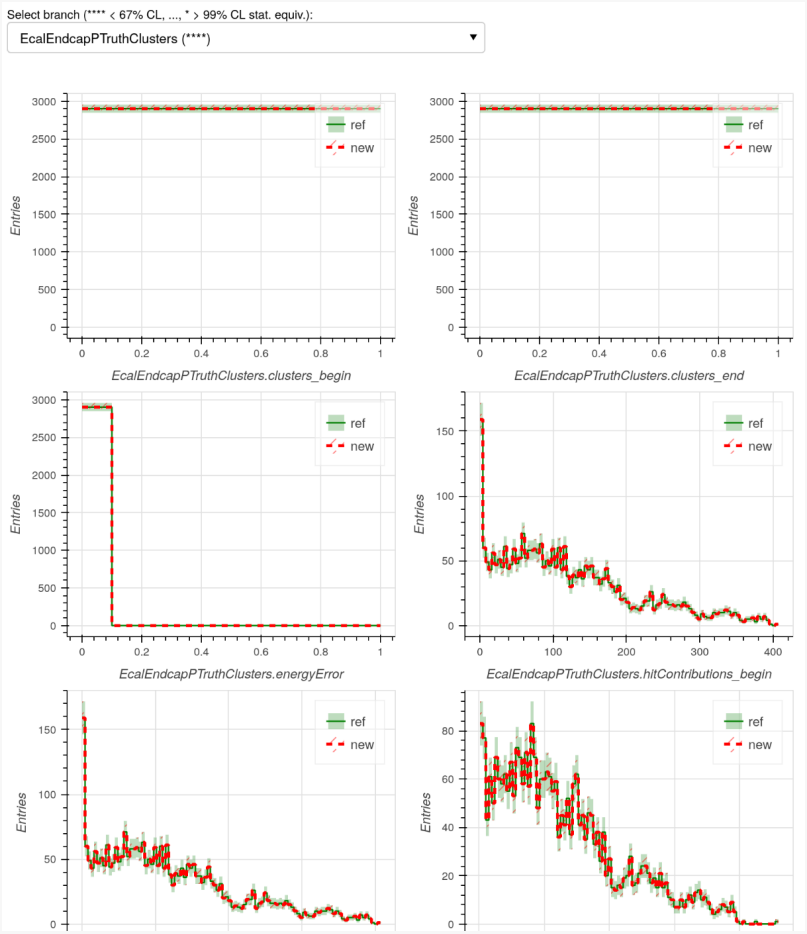
Benchmarking and validation

Idea

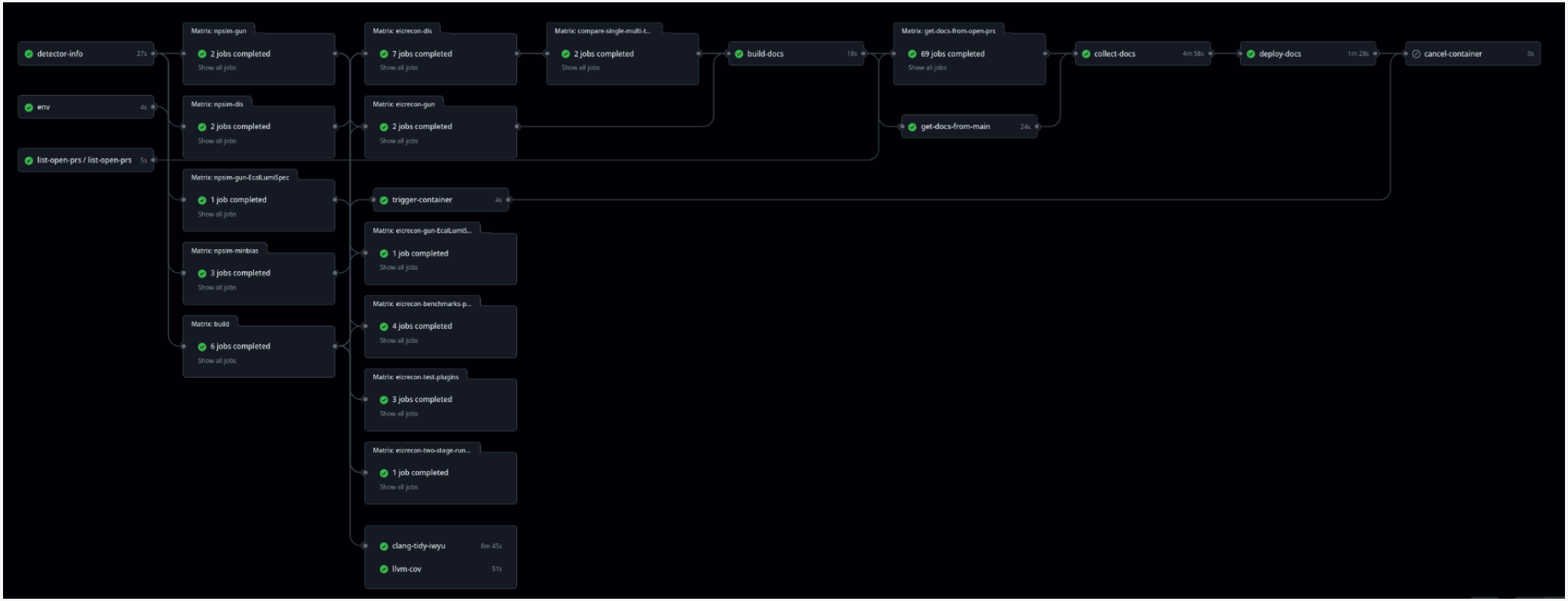
- Physics benchmarks are run for every PR, and returns a diff of each histogram. A CI job will fail if the diff is above some threshold.
- I love this. This is amazing!

Problems

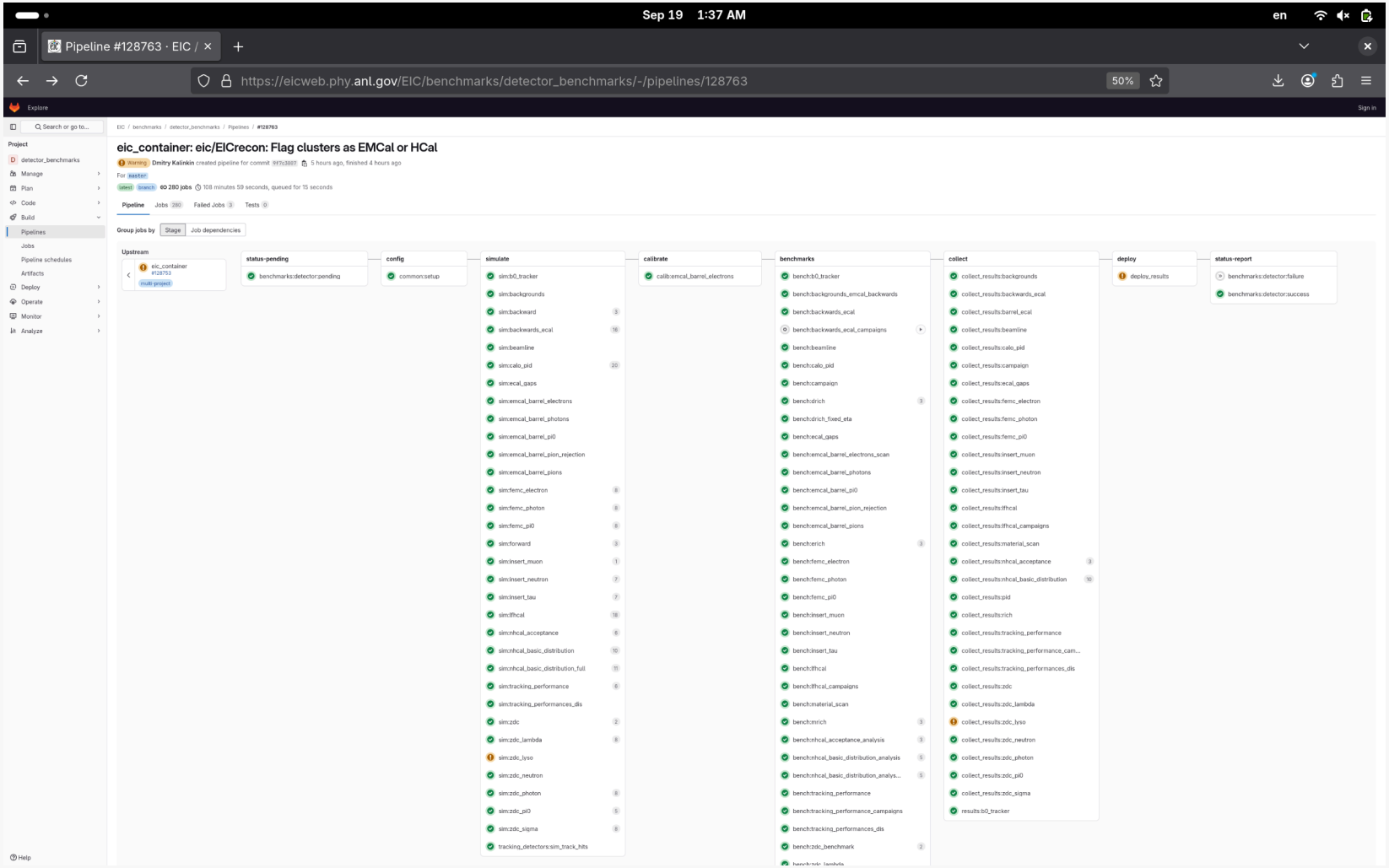
- (Most) validation does **not** happen on https://github.com/eic/EICrecon, but rather https://eicweb.phy.anl.gov/containers/eic_container
- Does validation still use Juggler?
- Are benchmarks deterministic? What happens when a code change which cannot change a benchmark result still changes a benchmark result?



The public GitHub CI pipeline



The ANL GitLab CI pipeline



Pipeline latency

Problems

- Running the full CI pipeline takes a long time even if everything passes (at least an hour)
- Many of these CI jobs appear to be redundant or bizarre
  - Running sim jobs when none of the sim code has changed
  - Merging GitHub Pages documentation for all open PRs (~60)
- I've **never** had a PR pass the full pipeline without at least a day's work
  - My workflow: Remember to pull my code to pick up any automatic whitespace fixes, make my fix, push it, set a timer for 15 minutes, go work on something else, repeat
  - Each `build` job terminates its sibling `build` jobs on failure, forcing more iterations
  - Oftentimes something breaks where I have no knowledge how (or the permissions) to fix it
    - Example: Segfault in the `#pragma once` analyzer
    - Example: Missing CI tokens
    - Example: Nonsensical errors in benchmark output
  - I almost always end up asking for help from D.K. or W.D.

Recommendations

Recommendations

Low-hanging fruit

- Get rid of iwyu completely
- Use sensible defaults with clang-format and don't change them
- Get rid of the overengineered documentation merging
- Any CI failures that can be fixed via a simple pragma need to advertise this fact loudly in their error logs, and link to the corresponding documentation

Medium-hanging fruit

- Create repository for sim data rather than always regenerating (and other testing data too!)
- Create a separate "release" CI pipeline which runs clang-format, etc, getting it off of the critical path for pull requests completely
- Reconsider backwards compatibility requirement
- CI `build` jobs shouldn't kill the others when one fails

High-hanging fruit

- Add a mechanism for running the full GitHub CI pipeline locally

Thank you for listening!