

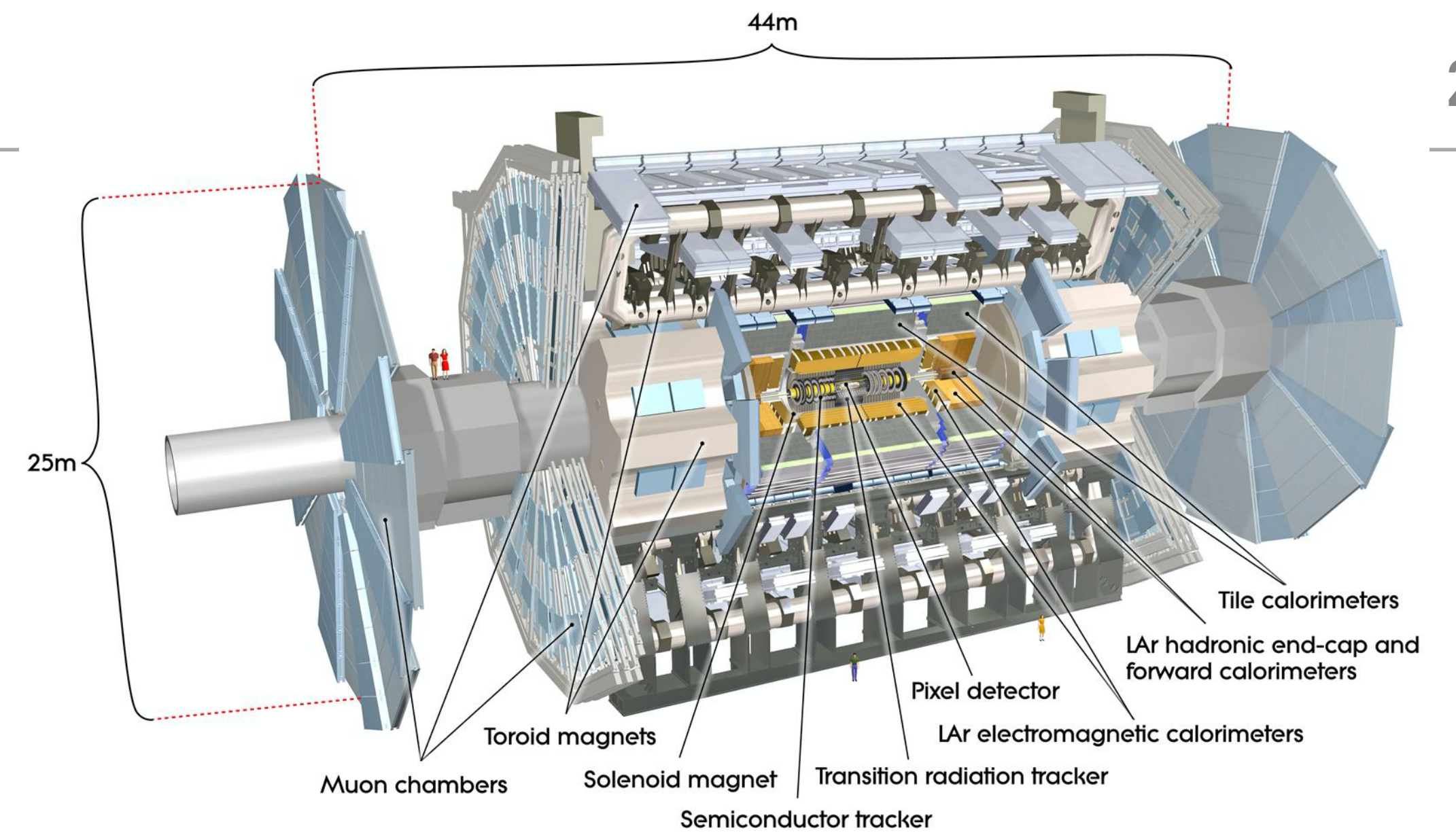
EDWARD MOYSE

SOFTWARE SUSTAINABILITY :

ATLAS

INTRODUCTION TO ATLAS

- ▶ Large international collaboration
 - ▶ ~ 2'900 Scientific Authors
 - ▶ ~ 1'200 Students
- ▶ Not unique to us (!), but poses some key challenges:
 - ▶ Complexity of managing large distributed teams of coders
 - ▶ We often have to convince collaborators to volunteer to help, because we have limited sticks and carrots
 - ▶ Much easier to get help with flashy new project, than maintaining an old piece of code



▶ [Athena](#) is ATLAS's event processing framework

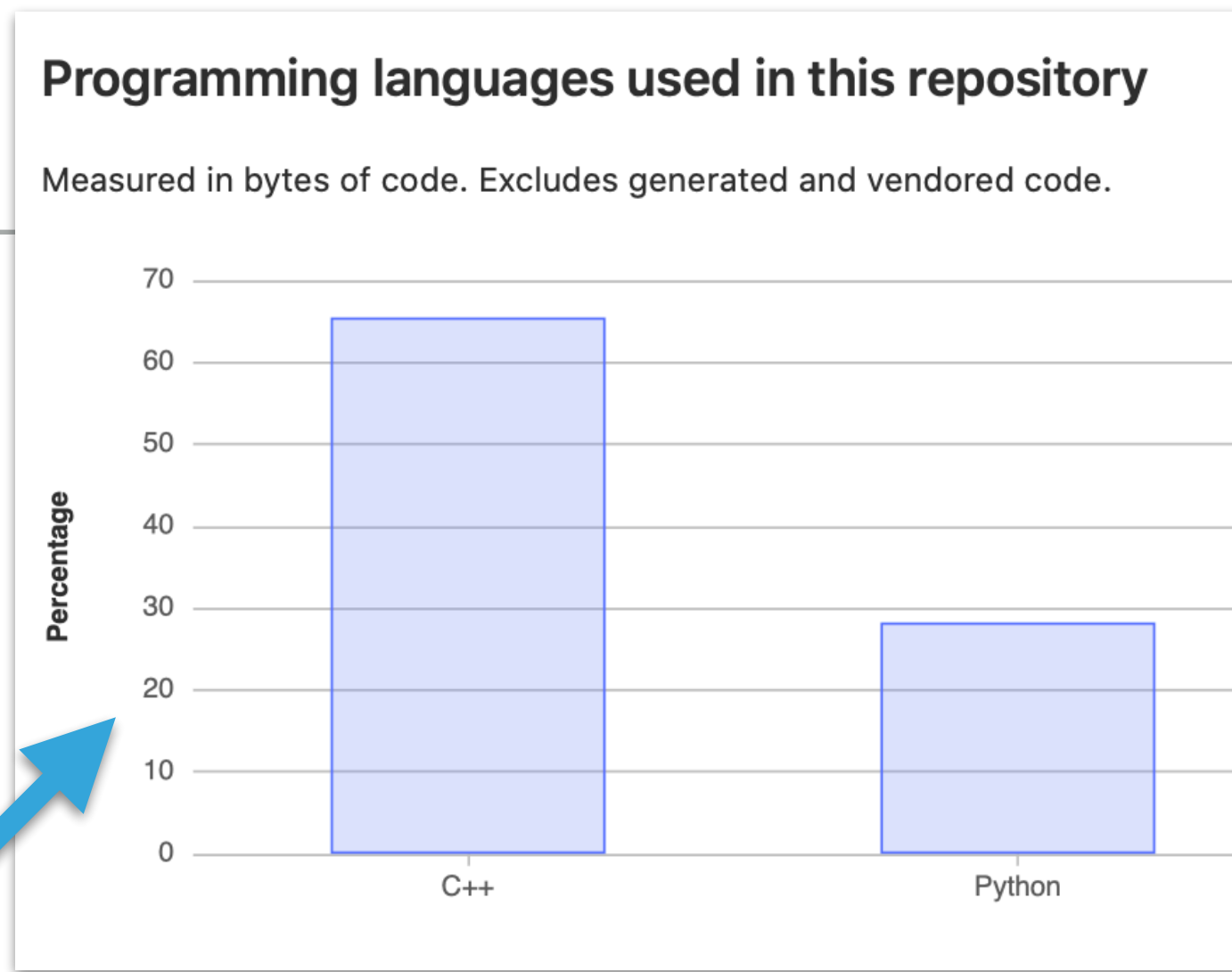
▶ >1 million lines of python and ~4 million lines of C++

▶ Largest & most active repository in CERN GitLab (by far)

▶ We also have many smaller repositories

▶ I don't know the total, and am not sure it's possible to find out

▶ (Will come back to this!)



atlas Group ID: 4114 | Leave group

ATLAS Software main group (with few super experts who have global rights)

Subgroups and projects Shared projects Archived projects Search by name

Subgroup/Project	Stars
Diversity and Inclusion	Owner
EventDisplayServer	★ 0
athmcproduction	★ 0
GeoModelPlugins	★ 1
athena	★ 107
atlantis	★ 3
athena-archive	★ 0
CIJenkinsConfig	★ 1
atlas-rpm-install	★ 0
athenaprivate1	★ 114

- ▶ Will now go through a few topics I think are relevant to sustainability, covering what I think works and what doesn't
- ▶ Includes
 - ▶ Standard tools
 - ▶ Open-sourcing our code
 - ▶ Social coding
 - ▶ Validation and testing, static analysis
 - ▶ Documentation and training, minimising expertise loss
- ▶ (Large overlap in topics in some cases, so there will be some repetition)

- ▶ **Initially, ATLAS used many home-grown tools (or exclusive to HEP)**
 - ▶ Examples: [CMT](#) build system, TagCollector to manage releases, CLHEP etc etc
 - ▶ Some advantage of handwritten solutions:
 - ▶ We get a tool that (hopefully!) perfectly fits our use case
 - ▶ **Some problems:**
 - ▶ Ongoing maintenance load (which take resources from elsewhere)
 - ▶ Dedicated training required
- ▶ **Since then, made many efforts to move to industry standards:**
 - ▶ e.g. Git + CMake (2016/2017), Eigen (2014) etc
 - ▶ Some advantages of industry standards:
 - ▶ Generally better functionality (have real experts writing the code, rather than physicists with some fraction of their time)
 - ▶ Many fantastic tutorials online beginners can use (and means they learn transferrable skills)
 - ▶ **Some problems**
 - ▶ Some older developers struggled to move e.g. to git, and in some cases, migration was a lot of work

- ▶ **Initially, ATLAS used many home-grown tools (or exclusive to HEP)**

- ▶ Examples: [CMT](#) build system, TagCollector to manage releases, CLHEP etc etc

- ▶ Some advantage of handwritten solutions:

- ▶ We get a tool that (hopefully!) perfectly fits our use case

- ▶ **Some problems:**

- ▶ Ongoing maintenance load (which take resources from elsewhere)

- ▶ Dedicated training

REDUCING THE AMOUNT OF UNNECESSARY CODE WE NEED TO MAINTAIN IS A HUGE STEP TOWARDS SOFTWARE SUSTAINABILITY

- ▶ **Since then, made many eff**

- ▶ e.g. Git + CMake (2016/2017), Eigen (2014) etc

- ▶ Some advantages of industry standards:

- ▶ Generally better functionality (have real experts writing the code, rather than physicists with some fraction of their time)

- ▶ Many fantastic tutorials online beginners can use

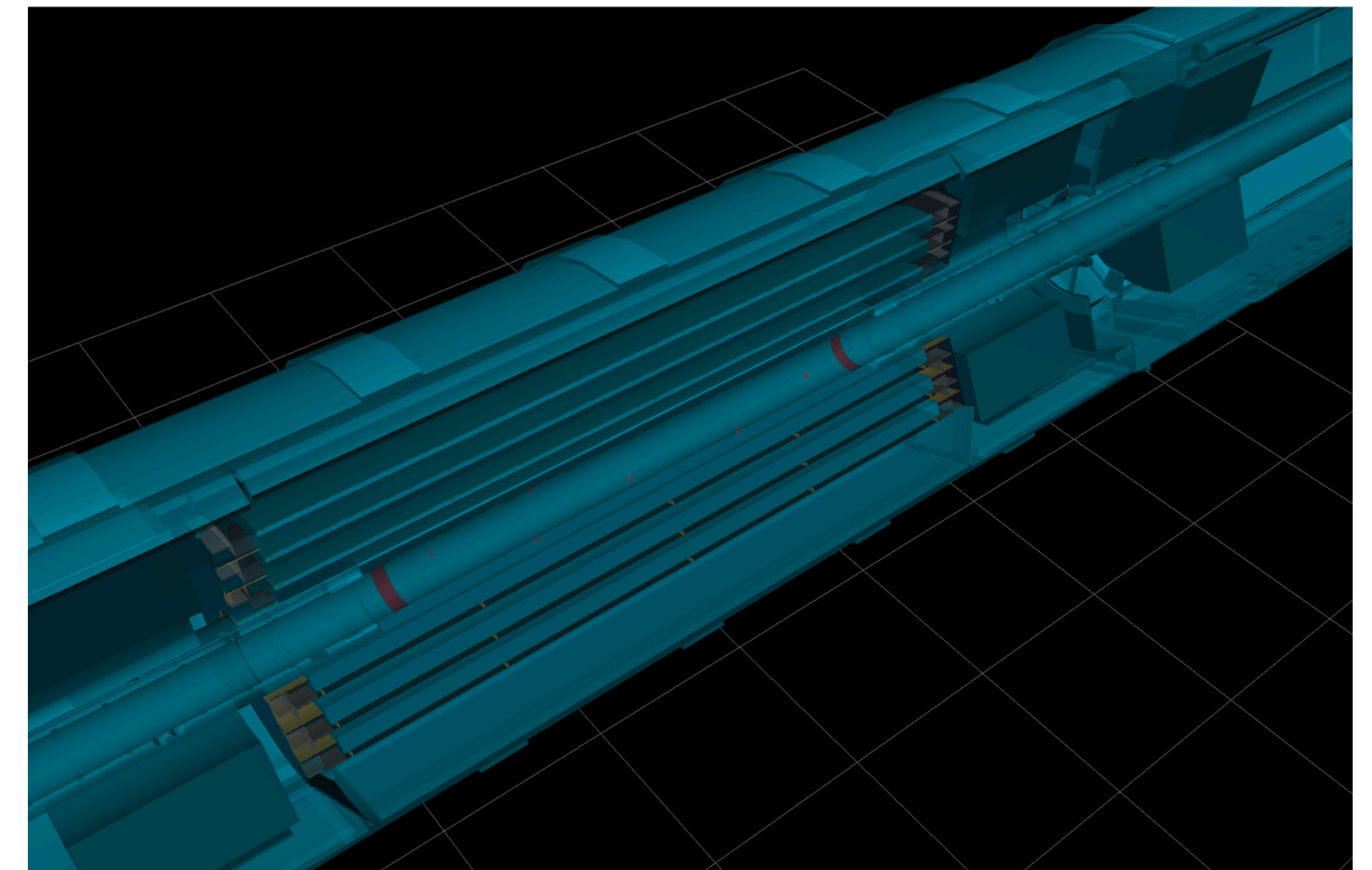
- ▶ Learn transferable skills

- ▶ **Some problems**

- ▶ ... in some cases, migration was a lot of work

OPEN SOURCE

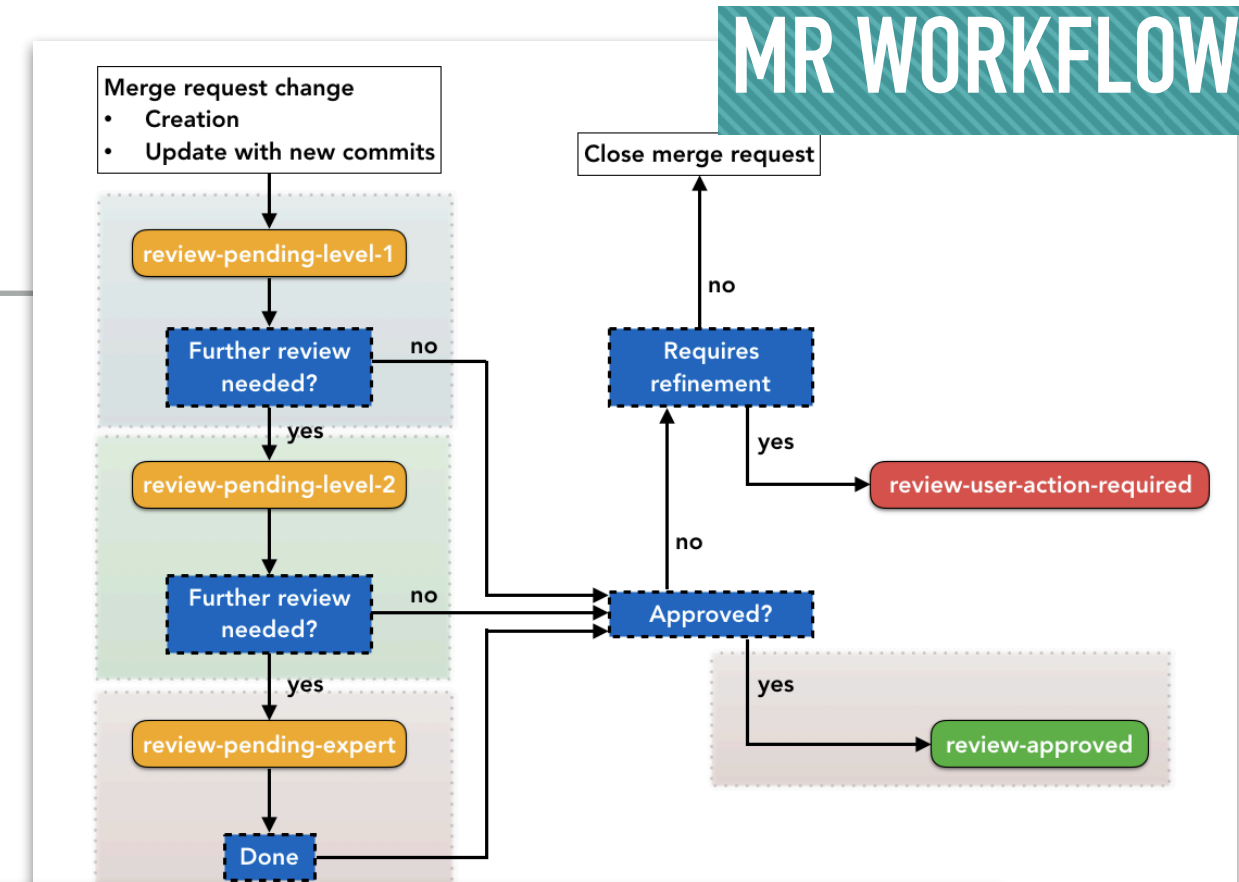
- ▶ Athena was open-sourced at the end of run-2, under an Apache 2.0 licence
 - ▶ ATLAS is committed to opening all of its software (some exceptions e.g. analysis)
 - ▶ Our experience has been very positive - much easier to share with interested outsiders
 - ▶ **One issue:** CERN lightweight account is not enough to contribute to GitLab.
- ▶ Examples of other open-source projects originating from ATLAS
 - ▶ [Rucio](#) - data management
 - ▶ [GeoModel](#) - geometry description language and tools
 - ▶ [ACTS](#) - experiment agnostic tracking software
 - ▶ [Phoenix](#) - experiment agnostic event display
- ▶ **Open sourcing software allows us to share effort with other experiments, and facilitates help from e.g. industry**
 - ▶ (It is also the right thing to do, IMO, with publicly funded projects)



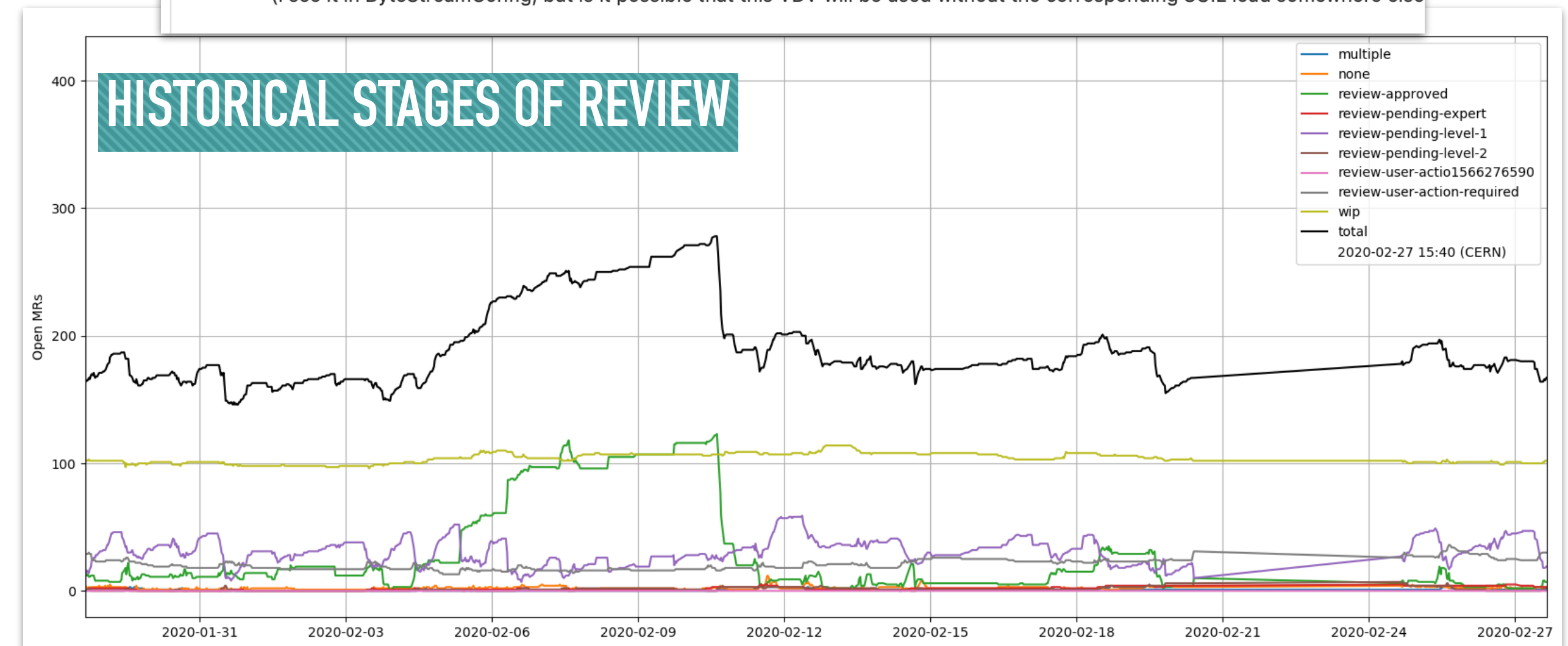
GeoModelClash

SOCIAL CODING / MERGE REVIEWS

- ▶ Within ATLAS, we make extensive use of gitlab (and github) & make extensive use of social coding feature, in particular: Merge(Pull) Reviews
- ▶ For Athena in particular this is very organised:
 - ▶ Two levels of shifters, working morning and afternoon
 - ▶ Review approximately 40 MRs per day (14k in 19 months)
 - ▶ Check for :
 - ▶ CI failures (see next slide)
 - ▶ Known gotcha (e.g. memory leaks)
 - ▶ Good code documentation
 - ▶ Following ATLAS [coding conventions](#) (writing “good” code, but also trying for some level of conformity)
 - ▶ Comments added inline to code - can trigger many rounds of updates
- ▶ IMO this is one of the most important improvements towards sustainability we've made
 - ▶ **Continuous code review**
 - ▶ (Though balance between moving to latest and greatest feature, and rapidly fixing important bugs)



The screenshot shows a code editor with a Python file: `Trigger/TriggerCommon/TriggerMenuMT/python/HLTMenuConfig/Muon/generateMuon.py`. The code includes a function `MuFastViewDataVerifier`. A comment from Benjamin Michael Wynne (@bwynne) is visible, discussing the loading of `EventInfo` and its availability at the whole-event level. The review interface shows the user's profile, role (Developer), and a timestamp (1 day ago).



- ▶ On every MR, we run continuous integration
 - ▶ Label MRs with software domain, so correct experts are notified
 - ▶ Compile code
 - ▶ Run unit tests (ctests) for affected packages (or all, if developer sets relevant gitlab label)
 - ▶ Runs some simple jobs to test Athena
 - ▶ Optionally: check to see if physics objects are changed
- ▶ Currently runs on Jenkins, but investigating moving to Gitlab CI.
- ▶ **Vitally important** to ensuring sustainability of our code
 - ▶ It is incredibly rare to lose a nightly because of a coding mistake

The screenshot displays a GitLab merge request interface. At the top, a user 'Atlas Nightlybuild' is noted as adding a watcher. Below, 'ATLAS Robot' reports that the merge request affects two packages: 'MuonSpectrometer/MuonCnv/MuonByteStreamCnvTest' and 'MuonSpectrometer/MuonDigitization/sTGC_Digitization'. It also lists several watchers. A section shows labels added by ATLAS Robot: '21.9', 'Digitization', 'MuonSpectrometer', 'review-pending-level-1', 'review-approved', and 'sweep:from 21.3'. The main CI result is 'SUCCESS' with a green checkmark and hash '93c2834d'. A table shows test results for 'Athena' across various categories, all with green checkmarks. At the bottom, there are links for 'this CI monitor view', 'Athena: number of compilation errors 0, warnings 0', and 'For experts only: Jenkins output [CI-MERGE-REQUEST 44479]'. On the right sidebar, there are sections for '0 Assignees', 'Milestone None', 'Time tracking', 'Labels' (with the same labels as in the main content), 'Lock merge request Unlocked', '8 participants', 'Notifications' (disabled), and 'Reference: atlas/athena!34977'.

	Athena
externals	✓
cmake	✓
make	✓
required tests	✓
optional tests	✓

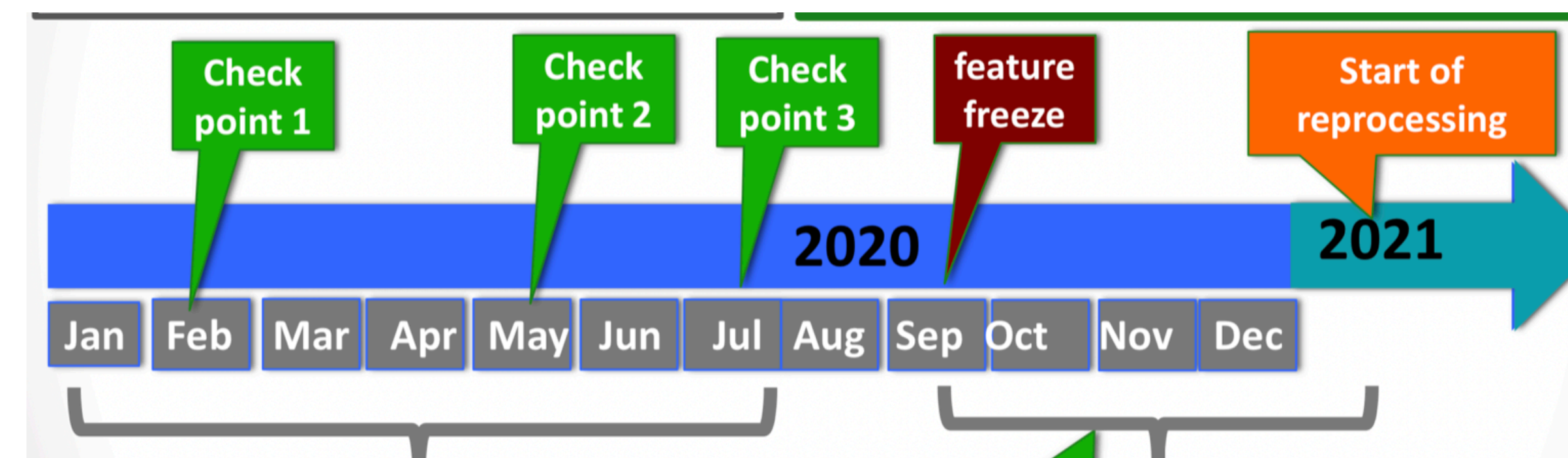
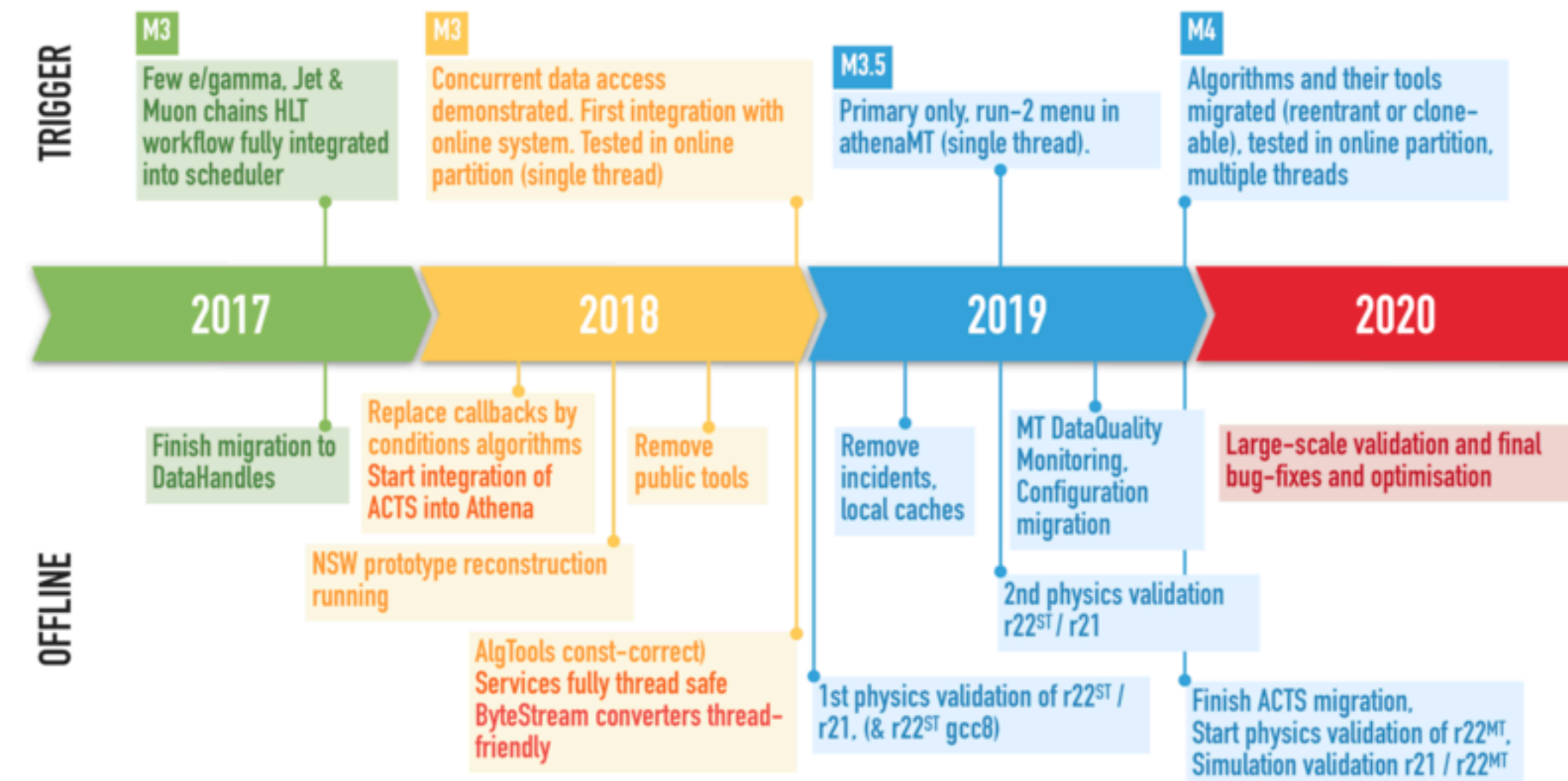
NIGHTLIES

entries

Release	Job time stamp	git clone	Extern. build	CMake config	Build time	Comp. errors (w/warn)	Test time	CTest errors (w/warn)	ART LOCAL	ART GRID	CVMFS (on server)	CVMFS (on client)
2020-03-04T2134	2020/03/05 04:37	☑	☑	☑	2020/03/05 04:37	0 (0)	2020/03/05 05:52	2 (2)	145,28	43,162,84,35	2020/03/05 06:30 ☑	2020/03/05 06:41
2020-03-03T2133	2020/03/04 04:34	☑	☑	☑	2020/03/04 04:34	0 (0)	2020/03/04 05:46	3 (3)	147,26	2,196,84,41	2020/03/04 06:21 ☑	2020/03/04 06:31
2020-03-02T2133	2020/03/03 04:29	☑	☑	☑	2020/03/03 04:29	0 (0)	2020/03/03 05:44	1 (1)	151,28	0,197,86,34	2020/03/03 06:24 ☑	2020/03/03 06:32
2020-03-01T2140	2020/03/02 04:32	☑	☑	☑	2020/03/02 04:32	0 (0)	2020/03/02 05:41	0 (0)	151,28	0,197,84,39	2020/03/02 06:13 ☑	2020/03/02 06:21
2020-02-29T2133	2020/03/01 04:32	☑	☑	☑	2020/03/01 04:32	0 (0)	2020/03/01 05:46	0 (0)	151,28	0,195,83,41	2020/03/01 06:20 ☑	2020/03/01 06:31
2020-02-28T2133	2020/02/29 04:31	☑	☑	☑	2020/02/29 04:31	0 (0)	2020/02/29 05:42	0 (0)	151,28	0,179,100,38	2020/02/29 06:28 ☑	2020/02/29 06:31
2020-02-27T2133	2020/02/28 04:35	☑	☑	☑	2020/02/28 04:35	0 (0)	2020/02/28 05:44	2 (2)	145,34	0,183,94,40	2020/02/28 06:18 ☑	2020/02/28 06:31
2020-02-26T2201	2020/02/27 05:06	☑	☑	☑	2020/02/27 05:06	0 (0)	2020/02/27 06:12	55 (55)	150,29	0,173,69,73	2020/02/27 06:49 ☑	2020/02/27 07:01
2020-02-25T2133	2020/02/26 04:43	☑	☑	☑	2020/02/26 04:43	0 (0)	2020/02/26 05:49	2 (2)	139,40	0,191,92,32	2020/02/26 07:05 ☑	2020/02/26 07:11
2020-02-24T2133	2020/02/25 04:37	☑	☑	☑	2020/02/25 04:37	0 (2)	2020/02/25 05:42	2 (2)	144,35	0,188,93,34	2020/02/25 06:15 ☑	2020/02/25 06:21
2020-02-23T2132	2020/02/24 04:39	☑	☑	☑	2020/02/24 04:39	0 (2)	2020/02/24 05:45	2 (2)	129,42	N/A	2020/02/24 06:21 ☑	2020/02/24 06:32

- ▶ We currently build ~20 branches per night
- ▶ On these we:
 - ▶ run unit tests (as with CI)
 - ▶ local longer tests, and
 - ▶ grid-based large statistics test (from which plots can be made and compared with references)
- ▶ (Non-unit tests are controlled by [ART](#), unit tests run by [ctest](#))
- ▶ Longer tests check for regressions, subtle bugs missed by CI
- ▶ Aside: **Compilers & heterogeneous platforms**
 - ▶ Our workhorse right now is gcc8 but we also compile nightly with clang8 (and developers run local builds with more exotic choices)
 - ▶ More compilers & platforms = more chances to find bugs in older code

- ▶ So, as demonstrated we build our software every night, and run a battery of tests on it
- ▶ We also run larger validation campaigns
 - ▶ Typically ~1 million events
 - ▶ Primarily to measure physics performance
 - ▶ Also to find (very) rare bugs, and measure technical performance

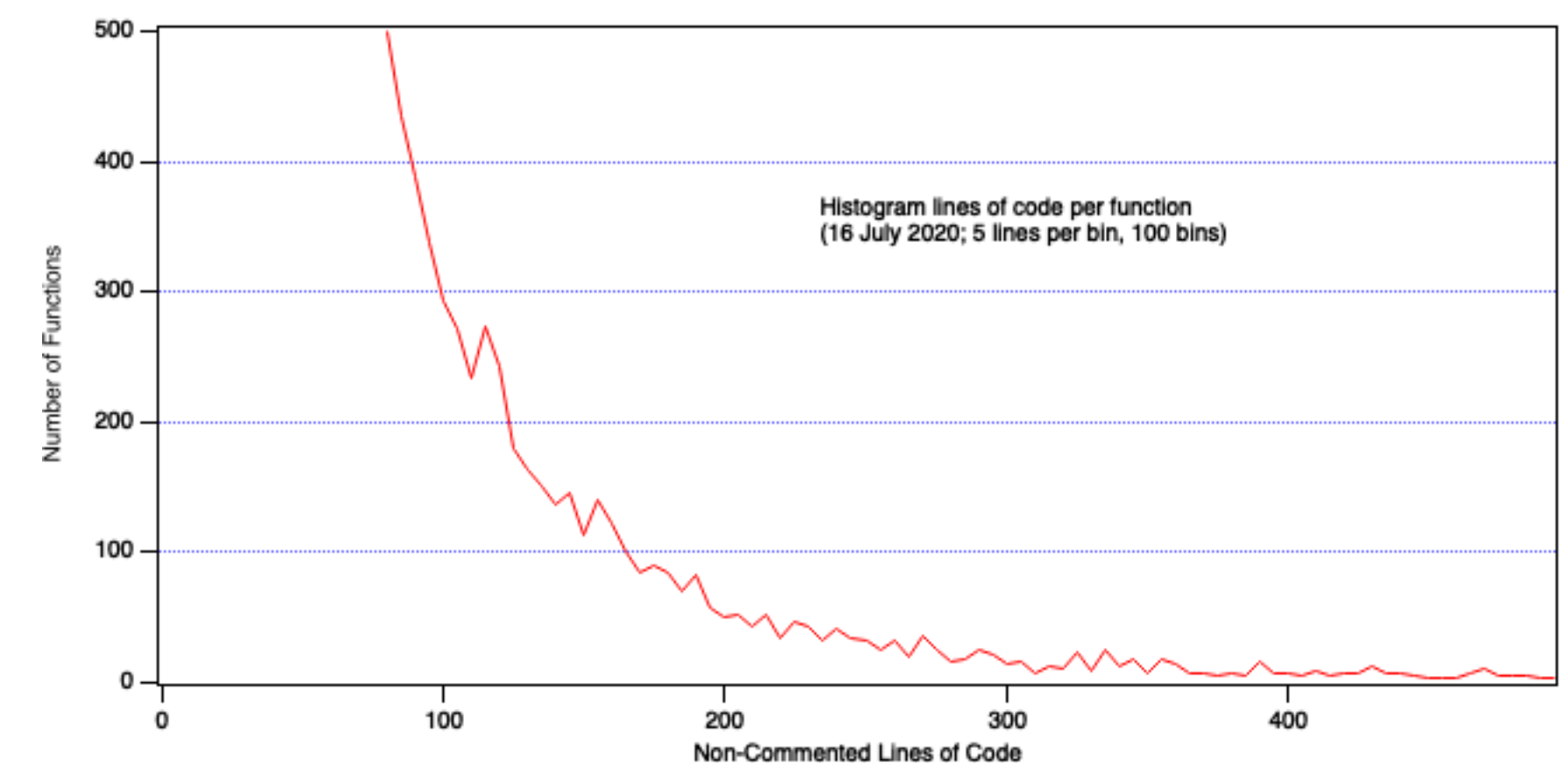
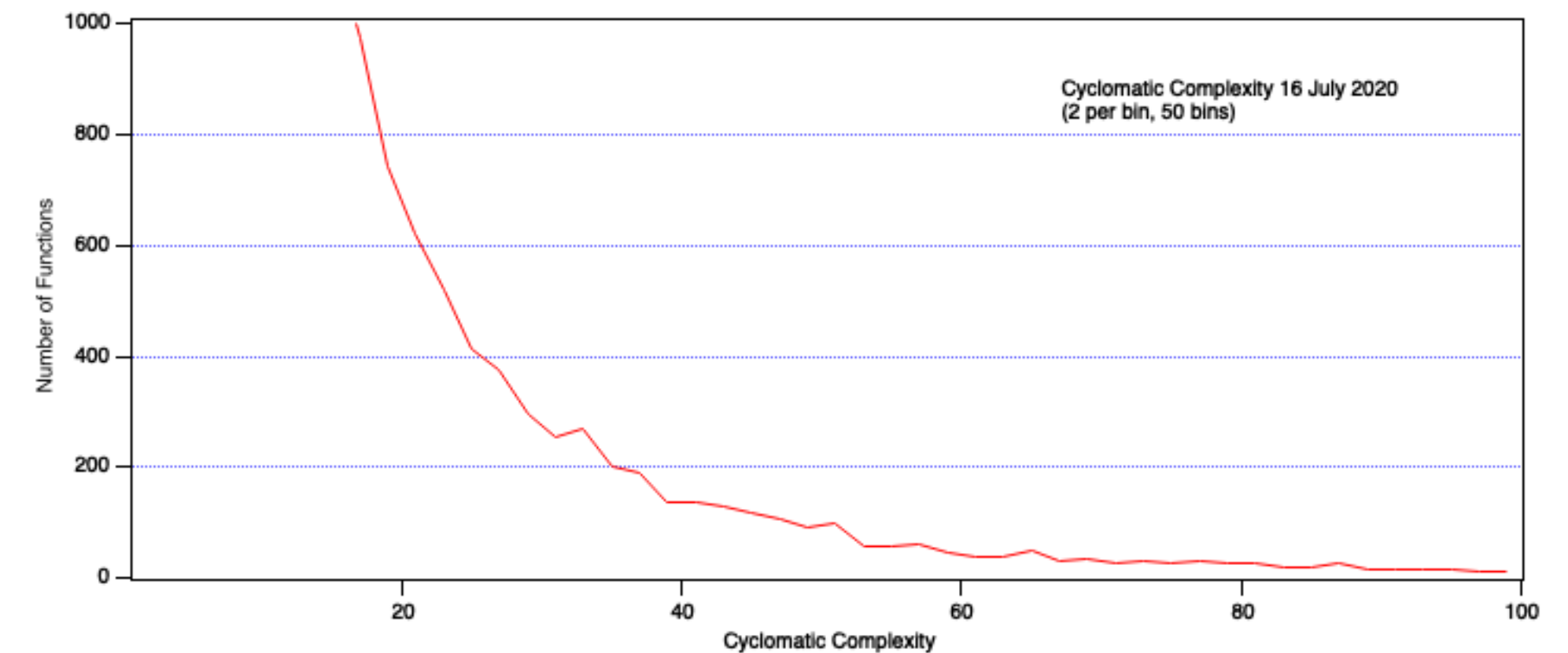


- ▶ For large codebase in particular, running tools to look for problems is very important
 - ▶ Not feasible to do complete code review of our software
- ▶ [Cppcheck](#)
 - ▶ Very static analysis useful tool
 - ▶ >100 MRs to Athena handling cppcheck warnings
 - ▶ Some false positives
- ▶ [Coverity](#)
 - ▶ Used to use this. Struggled to get it working recently.
 - ▶ Very slow (scans entire repository), complex licensing, requires dedicated server ... but probably better than cppcheck
- ▶ [Lizard](#)
 - ▶ Cyclomatic Complexity Analyzer
 - ▶ Potentially gives some interesting clues to 'hotspots'

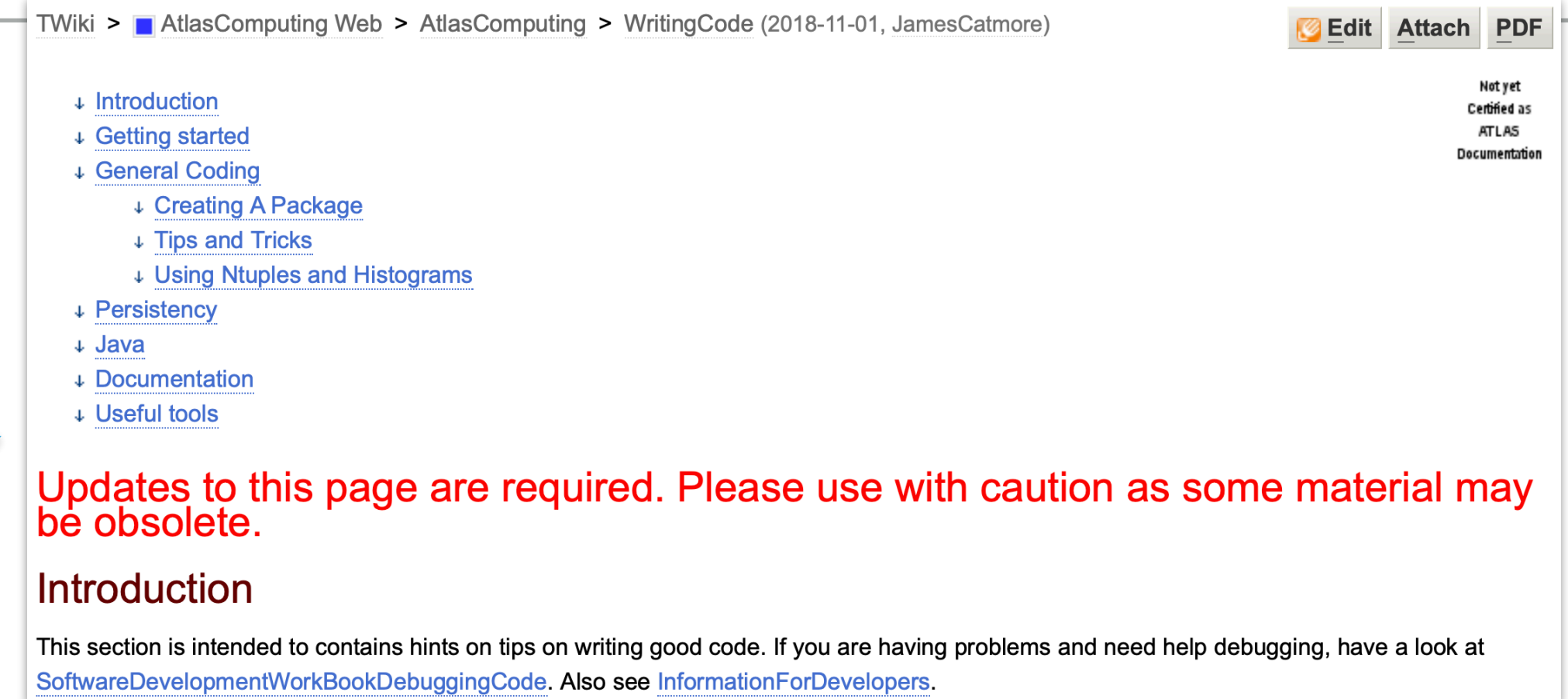
InDetVKalVxInJetTool+TrkVKalVrtFitter: Fix cppcheck warnings.

Overview **3** Commits **2** Pipelines **1** Changes **10**

- Prefer preincrement to postincrement for user iterators.
- Pass large structures by const reference rather than by value.
- The MAT << 0,0,... notation confuses cppcheck. But setZero is anyway clearer, so just use that.
- Fix use of uninitialized variable.



- ▶ ATLAS has historically documented software using Twiki
 - ▶ Search is awful, it decays fast, etc etc
 - ▶ Lots is restricted to atlas users
- ▶ We now have some documentation, [atlassoftwaredocs](#), maintained by experts
 - ▶ Public, searchable by google, cleaner interface
 - ▶ Problem is it is a lot of work for over-burdened experts



Twiki > AtlasComputing Web > AtlasComputing > WritingCode (2018-11-01, JamesCatmore)

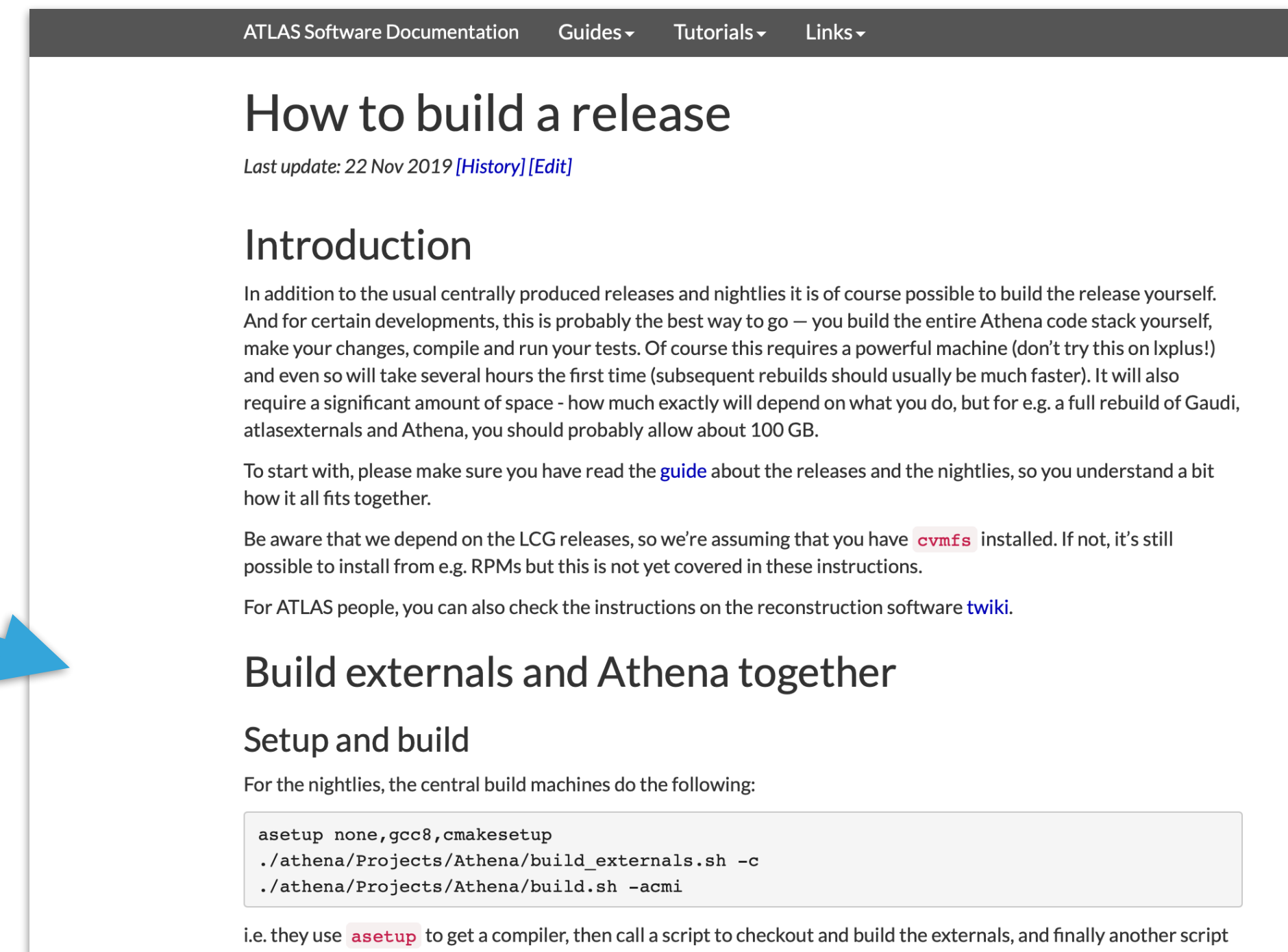
Not yet Certified as ATLAS Documentation

- ↓ Introduction
- ↓ Getting started
- ↓ General Coding
 - ↓ Creating A Package
 - ↓ Tips and Tricks
 - ↓ Using Ntuples and Histograms
- ↓ Persistency
- ↓ Java
- ↓ Documentation
- ↓ Useful tools

Updates to this page are required. Please use with caution as some material may be obsolete.

Introduction

This section is intended to contains hints on tips on writing good code. If you are having problems and need help debugging, have a look at [SoftwareDevelopmentWorkBookDebuggingCode](#). Also see [InformationForDevelopers](#).



ATLAS Software Documentation Guides Tutorials Links

How to build a release

Last update: 22 Nov 2019 [History] [Edit]

Introduction

In addition to the usual centrally produced releases and nightlies it is of course possible to build the release yourself. And for certain developments, this is probably the best way to go – you build the entire Athena code stack yourself, make your changes, compile and run your tests. Of course this requires a powerful machine (don't try this on lxplus!) and even so will take several hours the first time (subsequent rebuilds should usually be much faster). It will also require a significant amount of space - how much exactly will depend on what you do, but for e.g. a full rebuild of Gaudi, atlasexternals and Athena, you should probably allow about 100 GB.

To start with, please make sure you have read the [guide](#) about the releases and the nightlies, so you understand a bit how it all fits together.

Be aware that we depend on the LCG releases, so we're assuming that you have `cvmfs` installed. If not, it's still possible to install from e.g. RPMs but this is not yet covered in these instructions.

For ATLAS people, you can also check the instructions on the reconstruction software [twiki](#).

Build externals and Athena together

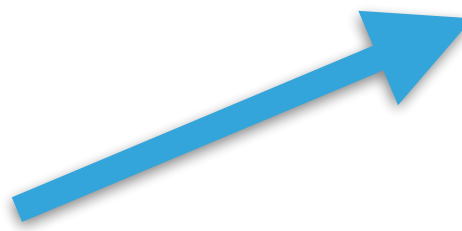
Setup and build

For the nightlies, the central build machines do the following:

```
asetup none, gcc8, cmake
./athena/Projects/Athena/build_externals.sh -c
./athena/Projects/Athena/build.sh -acmi
```

i.e. they use `asetup` to get a compiler, then call a script to checkout and build the externals, and finally another script

- ▶ Every person joining ATLAS is encouraged to go to week long induction
 - ▶ Happen multiple times a year
 - ▶ Software training is part of this
 - ▶ Primarily aimed at analysts, not software developers ...
 - ▶ ... but they do learn e.g. CMake and git!
- ▶ Have some infrequent software specific training
 - ▶ Merge request shifter training
 - ▶ ATLAS software developer training
 - ▶ Most recently [GeoModelXML](#)
- ▶ Have some very complete [Coding guidelines](#)



TUESDAY, 14 JULY

15:00 → 19:00 **Session 2: Software Essentials**

ATLAS CMake Docker GitLab CI/CD Git & Version Control

15:00 Q&A Session (30m)

Join Zoom Meeting
https://cern.zoom.us/j/98004111920?pwd=dZLWUzNm54ZWJlOVNlJGVzZFNvcrUT09

Meeting ID: 980 0411 1920
Password: 423005

Speakers and tutors answer questions on the prepared material.

- ATLAS CMake
- Git, GitLab & Version Control
- Docker
- GitLab Continuous Integration

Add Questions

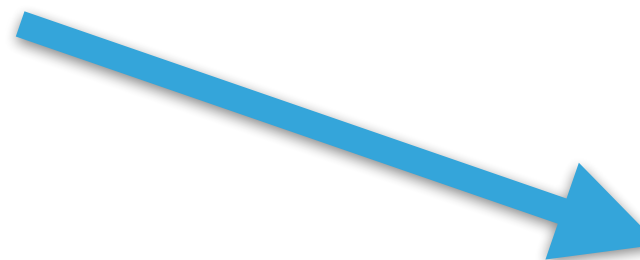
15:30 Break (15m)

15:45 Hands-on: Software Essentials (1h 15m)

1: Git for Analysis 2: Introduction to C... 3: GitLab Continuo... Advanced/Optiona... AST Day 2 - Softwa...
Optional: Docker

17:00 Break (30m)

17:30 Hands-on: Software Essentials (1h 30m)



ATLAS Software Development Tutorial

23-27 September 2019
CERN
Europe/Zurich timezone

Search...

Overview
Timetable
Contribution List
My Conference
My Contributions
Registration
Participant List
Videoconference Rooms

Timetable

Mon 23/09 Tue 24/09 Wed 25/09 Thu 26/09 Fri 27/09 All days

Print PDF Full screen Detailed view Filter
Session legend

Day 1

14:00 **Introduction** Edward Moyse
6/2-024 - BE Auditorium Meyrin, CERN 14:00 - 14:20

Introduction to Concepts in Multi-Threading Vakho Tsulaia
6/2-024 - BE Auditorium Meyrin, CERN 14:20 - 14:35

Introduction to Gaudi / Athena Vakho Tsulaia

15:00 6/2-024 - BE Auditorium Meyrin, CERN 14:35 - 15:15

Coffee
6/2-024 - BE Auditorium Meyrin, CERN 15:15 - 15:30

- ▶ **Big problem for us:**

- ▶ Without retaining people who understand the software, maintaining (and training the next generation of developers, is very hard)
- ▶ Unfortunately for many, focusing on SW development is perceived to damage their career prospects
- ▶ And indeed, we lost some key people because they could not get a job(!)

- ▶ **We have tried to combat this with**

- ▶ Grants for SW development - paying people to become experts
- ▶ Institutional commitments - i.e. institute takes responsibility for a core task)

- ▶ **Mixed results** - the core problem is (IMO) funding agencies

- ▶ Some countries are MUCH worse than others

▶ **(Dark) code rot**

- ▶ We don't (typically) review untouched code - but it might still run in production

▶ **Fractured codebase**

- ▶ Used to be that almost all production code was in Athena
- ▶ Nowadays we have many, many repositories (truthfully, I am not sure how many), not all of which are in gitlab
- ▶ Some are not visible even to the SW coordinators
- ▶ Best we can do is to try to educate people about best practices and concentrate on Athena

▶ **Documentation & Training**

- ▶ Could always do better

▶ **Rapid turnover of personnel**

- ▶ In some areas we have long term experts
- ▶ In others, code is written by young physicists who move on
- ▶ No surprise which makes code easier to sustain (though of course, longer term passing on knowledge is important too)

▶ **Size of codebase:**

- ▶ Makes migrations e.g. Move to MT, Python 2 to Python 3 etc painful
- ▶ (however forces us to review code!)

- ▶ ATLAS's experience with software sustainability:
 - ▶ A hard problem given our large codebase, distributed users (without clear hierarchy), and high turnover of experts
- ▶ What works:
 - ▶ Use industry-standard tools (don't re-invent the wheel)
 - ▶ Open source
 - ▶ Social coding: merge reviews
 - ▶ Documentation, training and coding guidelines
- ▶ Where we could do better
 - ▶ See previous slide!

BACKUP

- ▶ Track memory and CPU for each night
- ▶ Comparisons between nightlies

