# Key4HEP - The Common Turnkey Software Stack

Graeme A Stewart, CERN EP-SFT

Experiment Software Lifecycle I

- First ideas and inspiration...
  - Cool idea... would that work?
- Concepts
  - Very fast approximate methods, e.g. Delphes (smeared tracks, etc.)
- Design
  - Still need to be flexible to decide between alternatives
  - Ultimately need to pay a lot of attention to details for accurate performance evaluation
    - Accurate geometry, full simulation, realistic digitisation, ...

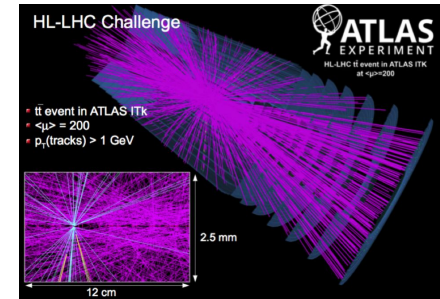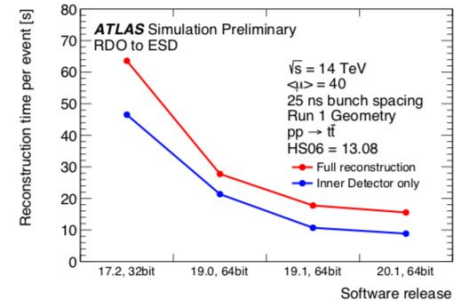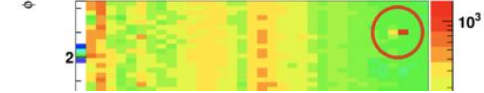Simulation of a H → μμ ee event in ATLAS
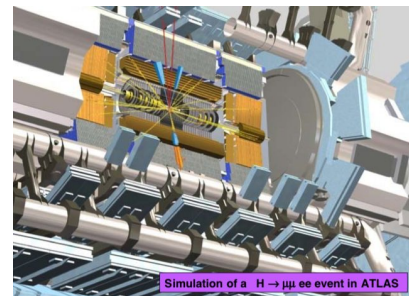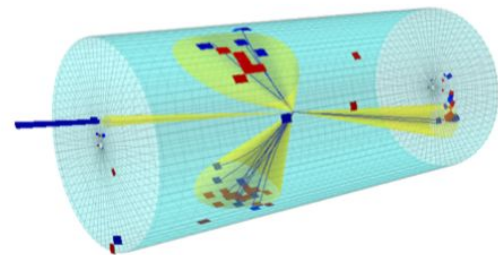
Experiment Software Lifecycle II

- Production
  - Dealing with the real world - calibration, alignments, dead and noisy elements
  - Learn about the detector, need stability but also continual improvements
- Upgrade
  - Design better sub-detectors for the next version
- Preservation
  - How can I make sure we can look at the data in the future?

For new experiments not everything needs to, or should be, solved up-front, but forgetting about the next step entirely will cause problems down the line (**technical debt!**)

Hit Map of clusters with E_clus> 2.5 GeV

ATLAS Simulation Preliminary RDO to ESD
√s = 14 TeV
<μ> = 40
25 ns bunch spacing
Run 1 Geometry
pp → tt̄
HS06 = 13.08
- Full reconstruction
- Inner Detector only

HL-LHC Challenge
- tt̄ event in ATLAS ITk
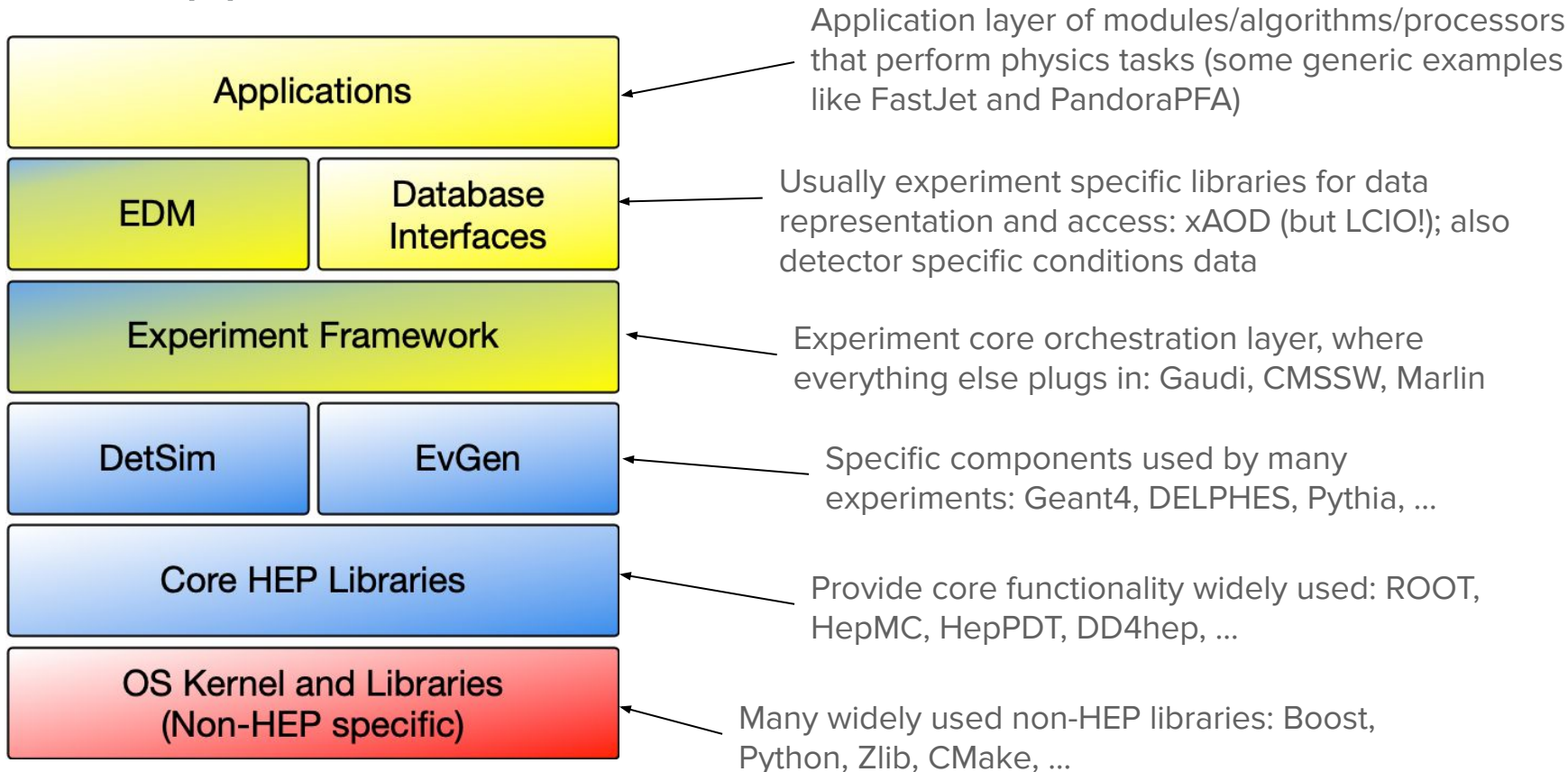- <μ> = 200
- p_t(tracks) > 1 GeV
2.5 mm
12 cm

```
TOFGN=TOFG*1.E+9
WRITE(CHMAIL,1000)ITRA,ISTAK,NTMULT,(NAPART(I),I=1,5),TOFGN
CALL GMAIL(0,0)
WRITE(CHMAIL,1100)
CALL GMAIL(0,0)
IEVOLD=IEVENT
NTMOLD=NTMULT
```
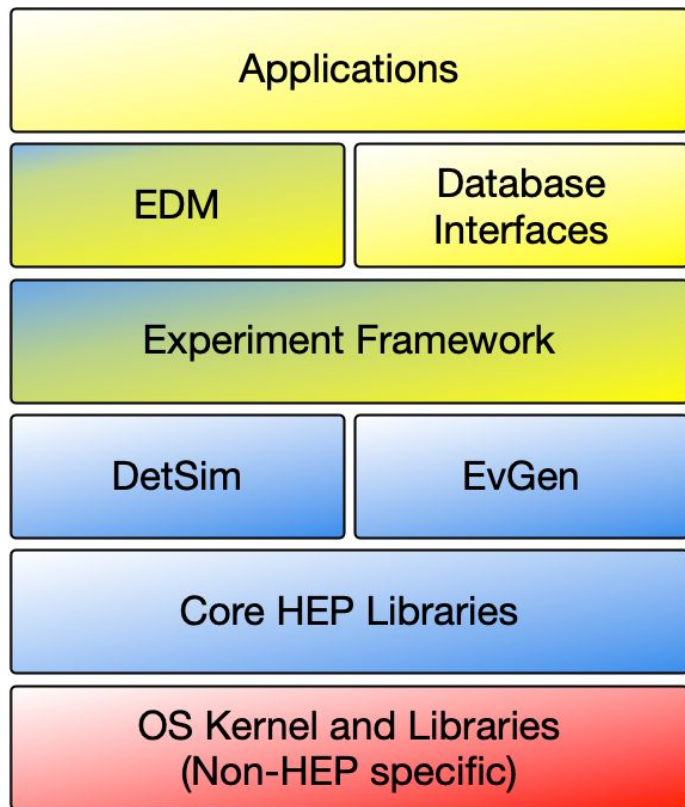
Snippet from CERNLIB

3

HEP Application Software

Applications — Application layer of modules/algorithms/processors that perform physics tasks (some generic examples like FastJet and PandoraPFA)

EDM / Database Interfaces — Usually experiment specific libraries for data representation and access: xAOD (but LCIO!); also detector specific conditions data

Experiment Framework — Experiment core orchestration layer, where everything else plugs in: Gaudi, CMSSW, Marlin

DetSim / EvGen — Specific components used by many experiments: Geant4, DELPHES, Pythia, ...

Core HEP Libraries — Provide core functionality widely used: ROOT, HepMC, HepPDT, DD4hep, ...

OS Kernel and Libraries (Non-HEP specific) — Many widely used non-HEP libraries: Boost, Python, Zlib, CMake, ...

Most General → Most Specific

Building HEP Applications

Applications

EDM	Database Interfaces

Experiment Framework

DetSim	EvGen

Core HEP Libraries

OS Kernel and Libraries (Non-HEP specific)

- Each piece of software does not live in isolation
- There is an ecosystem of interacting pieces
- Compatibility between the elements doesn't usually come for free
  - Common standards do help a lot
- Building a consistent set of software for an experiment is a task in itself
  - We are in the Packaging Group, so we know this 👍

Key4HEP Motivation

- Future detector studies critically rely on well-maintained software stacks to model detector concepts and to understand a detector's limitations and physics reach
- We have a scattered landscape of specific software tools on the one hand and integrated frameworks tailored for a specific experiment on the other hand
- Aim at a low-maintenance common stack for FCC, ILC/CLIC, CEPC with ready to use "plug-ins" to develop detector concepts
- Identified as an important project in the CERN EP R&D initiative
- Reached consensus among all communities for future colliders to develop a common turnkey software stack at recent Future Collider Software Workshop

Key4HEP Goals

- Put together a stack of the software packages covering the different domains
  - most commonly used, avoiding as much as possible functionality overlaps
- The turnkey stack connects and extends the individual packages to enable a complete data processing framework
  - converting a set of disconnected packages into a 'turnkey' system
- and should be
  - easy to use: for librarians, developers, users
  - easy to set up
  - easy to deploy (CVMFS and containers)
  - easy to extend
  - full of functionality
- Plenty of examples for simulation and reconstruction of detectors

Interoperability I

- Level 0 - Common Data Formats
  - Allows interoperability between different programs, even running on different hardware
  - E.g., HepMC event records, LCIO, GDML, ALFA messages
- Level 1 - Callable Interfaces
  - Basic calling interfaces defined by the programming language
    - Cross language calls are, of course, possible
  - Can be dependent on the compiler and language version (C++ in particular)
  - Details are important
    - how to handle errors and exceptions, is it thread safe, are objects const, dependent libraries and runtime setup
  - E.g., FastJet, Eigen, Boost

Interoperability II

- Level 2 - Introspection Capabilities
  - Software elements to facilitate the interaction of objects in a generic manner such as Dictionaries and Scripting interfaces
  - Example: PyROOT, which is a Python extension module that allows the user to interact with any ROOT (C++) class from the Python interpreter
- Level 3 - Component Model
  - Software components of a "common framework" offers maximum re-use
  - 'standard' way to configure its parameters, to log and report errors, manage object lifetime and ownership rules, 'standard' plug-in management, etc.
  - Unfortunately, no single Framework has been generally adopted

The right interoperability point between packages varies, but fixing it correctly eases life a lot for other developers and users

HSF Project Template

- To enable interoperability and long term maintainability
  - Build system
    - Share build results, but also knowledge of how to build
    - HSF Packaging working group studying various options
  - Testing
  - Licensing and Copyright
  - Documentation
    - If it is not documented, does it exist?
- Ensure as much uniformity as reasonable
  - https://github.com/HSF/tools

Common HSF Tools

This is a collection of tools used within the context of the HEP Software Foundation (HSF).

create_project

The tool create_project creates a template CMake project. The created project contains the standard use patterns for small CMake projects, plus support for Doxygen and CPack. Further documentation is provided within the created package itself inside the README.md.

```
1    # - Define the minimum CMake version
2    # HSF recommends 3.3 to support C/C++ compile features for C/C++11 across all
3    # platforms
4    cmake_minimum_required(VERSION 3.3)
5    # - Call project() to setup system
6    # From CMake 3, we can set the project version easily in one go
7    project(prmon VERSION 1.1.1)
8
9    #--- Define basic build settings --------------------------------------------
10   # - Use GNU-style hierarchy for installing build products
11   include(GNUInstallDirs)
12
13   # Add some build option variables, for static builds and profiling
14   if(NOT BUILD_STATIC)
15     set(BUILD_STATIC OFF)
16   endif()
17   set(BUILD_STATIC "${BUILD_STATIC}"
18     CACHE BOOL "Build a static version of the prmon binary" FORCE)
19
20   if(NOT PROFILE_GPROF)
21     set(PROFILE_GPROF OFF)
22   endif()
```

EDM4HEP

- To achieve the highest levels of interoperability components should directly talk the same language
  - For HEP this is the *Event Data Model*, EDM
- The experience of the ILC/CLIC community in sharing an EDM has been a very positive one
  - Defining it may not be easy
  - But once achieved it pays off handsomely
- As an outcome of the Future Collider Software Workshop a small working group is being setup (ILC/CLIC/FCC/CECP) to discuss this
  - Our baseline is the highly successful LCIO from the linear collider community
- First meeting yesterday with presentations on LCIO and FCC-EDM
  - Aiming for first early release in 3-6 months

Implementing the EDM

- Original HEP C++ Event Data Models were heavily inspired by the Object Oriented paradigm
  - Deep levels of inheritance
  - Access to data through various indirections
  - Scattered objects in memory
- In practice access to the data in this way can be very slow
  - LHC experiments needed to optimise this a lot during Run 2
- Challenges in this area are only growing
  - As well as increasing memory latency, we look more and more towards accelerated computing devices
- Use PODIO EDM generator
  - Data model described at a high level in YAML
  - Code is then generated for target languages and different persistency backends
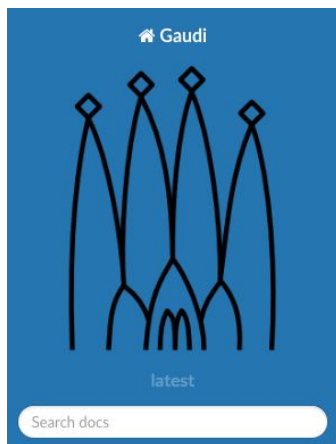 - Insulates users from implementation details and allows for common optimisation
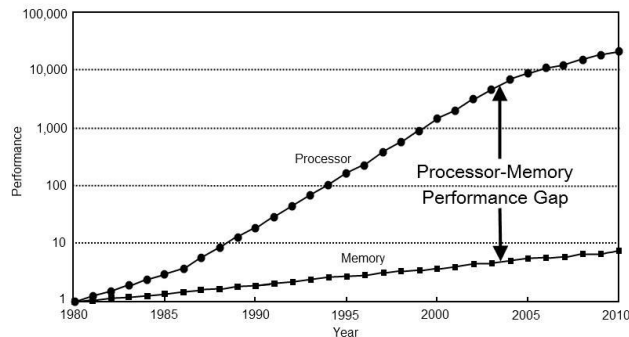
Experiment Framework

- Data processing frameworks are the skeleton on which HEP applications are built
- To get software to 'click' in the best way possible it's a huge advantage to share a software framework
  - In HEP we have traditionally not done so well here - many frameworks and a lot of duplication of effort
- Marlin was used by the LC community and was very successful as a common project
  - Unfortunately very far behind in conversion to modern concurrency needs
  - It does have a lot of well liked features (e.g. configuration)
- Gaudi is another shared framework, used by LHCb and ATLAS, as well as smaller experiments
  - Supports concurrency and has all hooks needed for data taking
- We will base Key4HEP on Gaudi and contribute to development where needed
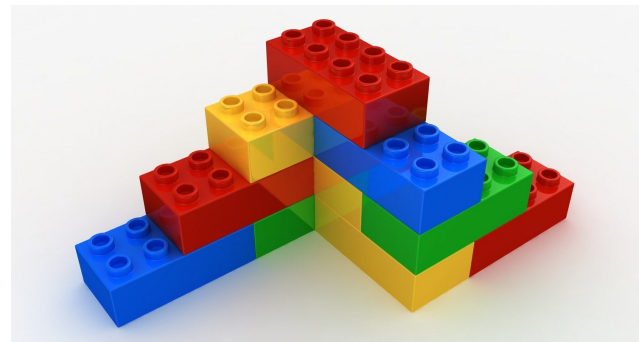
Gaudi Modernisation Project

- A lot of the very hard things are done
- Improved documentation coming
- CMake build system being rewritten this summer

Practical Progress

- To help integrate ILCsoft algorithms into Key4HEP a Marlin➜Gaudi wrapper has been developed (André Sailer)
    - https://github.com/andresailer/GMP
    - Prototype for now, but proves that framework transition is quite practical to achieve
- As part of the AIDA2020 project LCIO has been re-implemented using PODIO (Frank Gaede)
    - This demonstrates, as expected, that PODIO can implement a generic data model like LCIO
    - Some interface changes, e.g. shift from pointer to member semantics
        - These can probably be temporarily masked

Spack Building

- Summer student project (Hobbs Willett) working with Javier and Graeme
- Currently building a minimal stack with ROOT and all dependencies
  - Checked binary cache function
  - Working on using module files to distribute runtime environment setup
  - Checked relocation of packages
- Further Plans
  - Move to using EOS/Web based binary cache
  - Distribute to CVMFS for testing

Next Steps and Conclusions

- General agreement on moving to a common HEP software stack from future experiments
  - A lot to be learned from ILC/CLIC experiments in how to do this and the benefits
- Move build infrastructure to Spack
  - Progressing on this now
- Started on EDM4HEP
  - Working group now active
- Provide common and popular HEP package sets
- Gaudi framework to bind elements together
- Shims to form a plugin system of software that really works coherently
  - This is the critical point to make this a *turnkey* stack