HEP Analysis Ecosystem Workshop Report

August 4 2017

Contents

Introduction

Over the past 20 years the HEP community has developed and gravitated around an analysis ecosystem centered on ROOT. ROOT and its ecosystem both dominate HEP analysis and impact the full event processing chain, providing foundation libraries, I/O services etc. that have prevalence in the field.
The analysis tools landscape is however evolving in ways that can have a durable impact on the analysis ecosystem and a strong influence on the analysis and core software landscape a decade from now, a timescale currently in an intensive planning round with the HEP Software Foundation (HSF) Community White Paper process. Data intensive analysis is growing in importance in other sciences and in the wider world. Powerful tools and new development initiatives, both within our field and in the wider open source community, have emerged. Creative developers have an ever more powerful open source toolkit available and are applying it towards innovations that leverage both open source and the ROOT ecosystem. ROOT itself is approaching a major re-engineering with ROOT 7, leveraging the powerful evolution of C++, with a major overhaul in its interfaces as seen both by its users and by satellite tools in the ROOT ecosystem.

With these considerations the HSF organized an open HEP software community workshop in Amsterdam May 22-24 2017 to examine the analysis ecosystem, currently and in the future with a 5-10 year view. The workshop was organized as something of a retreat aimed at an inclusive building of consensus (wherever possible) among developers, users, projects and their supporters in the HEP analysis ecosystem, and to increase communication, coherence, and awareness within the community. Where complete consensus was not possible we have documented the disagreements.

The workshop chairs were Pere Mato (CERN), Liz Sexton-Kennedy (FNAL), and Torre Wenaus (BNL). The local organization was led by Peter Elmer (Princeton) and Jeff Templon (Nikhef). The chairs and participants gratefully thank Peter, Jeff and Nikhef for arranging an excellent venue and a very enjoyable, smoothly executed workshop amid pleasant Spring weather on Amsterdam's harbour.

The success of the workshop was due in largest part to the broad participation by experts in the field, including analysis experts in the experiments, developers of analysis workflows and frameworks, and analysis tool developers. The workshop had 55 registered attendees plus additional remote participants.

The workshop had four topical sessions spread over 2.5 days covering

1) **Today's analysis ecosystem and toolset.** Surveying the present analysis tool ecosystem, ROOT as the ecosystem hub, and the analysis models of the experiments.
2) **The analysis ecosystem vision.** Personal visions from a diverse group of experienced individuals for how the analysis ecosystem will develop in coming years.
3) **Concrete technologies.** Talks on present and future foundation technologies, data formats, languages, analysis as a service, analytics and machine learning, and technologies as applied in experiments.
4) **Missing pieces.** A synthesis of what has come before, what's been missed, what are the weak spots in the ecosystem, where are there opportunities for new work and new projects, what is the priority work.

These were followed by a closing session to draw conclusions and consider next steps. After the workshop's conclusion the chairs, session conveners, scribes and interested others stayed on for a few hours to begin summarizing the sessions and overall conclusions as a first step to writing the report. We departed the workshop with about 25 pages of notes in hand, with this report following a few weeks later.

This report includes a narrative summary of the workshop session by session, drawing out the highlights and themes; the prominent overall themes and important points that emerged in the workshop; developer feedback, particularly to the ROOT roadmap; near term priorities; R&D and demonstrator prospects; and action items. The report will be a primary contributor to the 'data analysis and interpretation' chapter of the HSF community white paper, the status and process for which were summarized in the workshop.

Session summary

The conveners for each session were asked to ensure that designated scribes (most often the conveners themselves) take notes during the session in the live notes document. Note taking was completely open however, with all encouraged to contribute. Scribes and conveners later participated in the report draft writing session that followed the conclusion of the workshop. Version 1 of this report contains the largely unmodified session summaries written in the hours and days after the workshop. It was felt that a more condensed and coherent distillation of the summaries was needed for the final report, hence the version here.

Session 1: Today's analysis ecosystem landscape and toolset

Session conveners: Attila Krasznahorkay, Paul Laycock, Andrea Rizzi
Speakers: Pere Mato, Jakob Blomer, Graeme Stewart, Gianluca Cerminara, Philippe Canal, Eduardo Rodrigues, Martin Ritter, Steven Schramm, Andrew Gilbert

This session surveyed ROOT's current capabilities and development plans; I/O issues including ROOT I/O performance comparisons with other technologies; experiment approaches to data analysis and its evolution; machine learning for analysis; and how the very final steps of the analyses are performed (statistical analysis, sharing of results between experiments, etc.).

Taking the starting point of analysis to be where centralised production of datasets ends, current analysis models typically involve data reduction steps to produce intermediate files that can fit on local resources or laptops. Data reduction is achieved either via event selections or reduction and harmonisation of per-event content or both. A frequent observation during the workshop is that these steps often involve semantics that can be expressed in a declarative programming style, now in use by several experiments and projects (ATLAS, Belle II, ROOT TDataFrame, …). Another noteworthy observation is the ability of LHCb to produce physics-ready data from the output of the (second stage) high-level trigger, requiring the calibration and alignment of the detector in real time in situ with the trigger system. This is a key enabler of a simpler analysis model that allows simple stripping of data and very efficient data reduction.

Late stage analysis is generally I/O bound, and decompression is a substantial part of CPU usage. Highly optimized I/O is a necessity. In terms of formats there is a clear consensus: while there are many industry standard formats, some with important roles, the most important format for our community now and in the future is ROOT I/O. Designed specifically for HEP, it remains best suited to our needs. Understanding exactly where I/O performance is lost is a topic needing study: how much is lost in decompression, what are the performance costs as the data format becomes more complex. Since optimising I/O performance is very important, and yet perceived to be difficult to get right, could we reduce the phase space of I/O tuning parameters? We could benefit from a systematic study looking at variations of ROOT-based storage (e.g. vectors or vectors of simple types) and see where performance is lost, thus giving us a way to easily perform a cost-benefit analysis in designing a storage format.

In terms of languages, amid the many languages in use there is consensus on the special roles of some. Performance is key in HEP, and certainly when we talk about reconstruction and performance-critical code, we will continue to look to C++, because the community has invested a lot in this language. It is not the only high performance language, but it is and (in the estimation of the workshop) will remain the dominant one. However, there was a very strong message from the community that for analysis code, python is highly desirable especially because of the fast development cycle and in general people can achieve the same results with fewer lines of code. Python should be treated as a first class language. Performance-critical code should be offloaded to libraries, probably written in C++, but the analysis code can be python. Python is also the language of choice in the wider scientific data analytics community. Sustaining PyROOT in particular is a key concern of the community. There is an argument that we lose C++ experience in the community by encouraging python but the overall view was that freeing up analysts time by enabling faster analysis turnaround is the best way to free up time for important work outside of analysis. An important point for language choices is that we need to ensure that senior people also need to be included, requiring adequate education and documentation. End users should be exposed to simple interfaces, which should reduce the complexity of analysis code.

For analysis code quality, this has been usually checked at the level of physics, i.e. data vs MC comparisons, with the code itself largely lacking any peer review. The widespread adoption of git is helping to cast a brighter light on the code and its quality, as merge request reviews, automatic testing and continuous integration with Jenkins et al are becoming ubiquitous.

Non-event data is an area with little commonality between experiments, a consequence of its roots in the detectors for conditions and calibrations. However it is highly desirable to use common infrastructure software given the shared facility infrastructure. Both the ATLAS and CMS speakers pointed out that analysis calibrations like b-tagging or muon identification efficiencies are underserved by the analysis software systems used in the experiments as these systems do not reuse the infrastructure available in the full framework applications. Improved software infrastructure sharing is being discussed in the Conditions DB WG, where the

system uses REST (Representational State Transfer) style interfaces to communicate between client and server. ATLAS and CMS are taking this approach, and Belle II intends to use this for conditions delivery to analysts. For other aspects of non-event data (luminosity, book-keeping), the problem needs to be better defined before commonalities and common solutions can be discussed.

There was some discussion on whether we're approaching concurrency in the right way, and how much we should expose the analysts to. The use of declarative interfaces may allow a hiding of the concurrency complexity. Many analysis workflows are still I/O bound, but fitting is a good example of where utilising GPUs can lead to much faster turnaround times (c.f. LHCb). The ROOT team plans to invest a significant amount of work in this area.

Machine learning in HEP was a major discussion point. The point was made several times that better integration of external machine learning toolkits with ROOT is highly desirable, in particular to the deep learning toolkits. This was evidenced most clearly in a dedicated talk (Steven) where the results of a survey were shown. It was strongly felt that directly supporting the exchange of data formats of external toolkits, building data bridges, is a better approach than incorporating these tools as TMVA-as-interface-to-all. Detailed discussions on how to do this were in other sessions, in particular how we feed data efficiently into Keras et al and transform data formats from one type to another (see Jim's talk on Tuesday).

Not addressed directly in the sessions but a useful summarizing input to the workshop was the HEP analysis ecosystem tool set summary document describing tools used today and tools being investigated.

Session 2: The analysis ecosystem vision

Session conveners: Brian Bockelman, Axel Naumann, Gordon Watts
Speakers: Fons Rademakers, Max Baak, Eduardo Rodrigues, Gordon Watts, Gerhard Raven, Pere Mato Vila, Axel Naumann

This session presented and discussed the personal visions from a diverse group of experienced individuals for how analysis models and the analysis ecosystem might develop in coming years.

Thanks to Moore's Law, a top-of-the-line CPU from 1995 (an Intel 486 running at 66MHz) would no longer have sufficient performance to run a smartwatch. Unsurprisingly, the software approaches currently used in HEP analysis - the fundamentals are still familiar to a physicist from 1995 - do not tightly align with today's hardware platforms. The years of Denard scaling - increases in clock rate - are nearly a decade in the past; modern CPUs achieve speed improvements through increased use of thread-level parallelism and increasing the instructions retired per clock cycle (often through improved usage of memory hierarchies and awareness of latency/prefetching). It appears these approaches will be important between now and the HL-LHC era. Fons Rademakers presented that in addition to these techniques we can expect hardware to improve through the use of accelerators (such as discrete GPUs or in-chip FPGAs) and hybrid CPUs providing specialized services such as hardware compression/decompression or random number generation. A further prospect is new fast layers in the storage hierarchy. For example, Intel's 3D X-Point technology appears to be faster and lower-latency than SSDs, yet cheaper than DRAM. Utilizing these new hardware approaches is likely as big a challenge as multithreaded programming: we must develop software programming models allowing users to effectively use the features (implicitly or explicitly) without becoming deep experts.

Recently, a range of industries have developed data and analysis problems that meet or exceed the size of LHC's. This was presented by Max Baak. More importantly, the ecosystem *investment* in "Big Data" wildly overpower the software investments from HEP. A theme throughout the session was the LHC community should strive to improve interaction with external toolkits (both in terms of *utilization* and giving back to the community) while focusing on simplifying and improving the core functionalities (particularly, within ROOT) where our analysis use case is unique. Concern was expressed that too much re-implementation was done in the past and we should work carefully to avoid doing the same when interacting with the (large) machine learning ecosystem.

From Axel Naumann's perspective as a ROOT core developer, ROOT should focus on being a foundational layer -- particularly IO -- and "community watering hole," but also work to improve interoperability with external toolkits instead of reimplementing features. A concrete suggestion was to improve ROOT's presence in the Python packaging ecosystem. Python is best-suited to allow the field to leverage significant existing code (>50M lines of C++), skillset of the current workforce, and integrate the wider scientific community's tools, particularly in the area of machine learning. This should not be done at the cost of performance of C++ or walling off individuals from experimenting to integrate ROOT with other higher-level languages.

Another mechanism for improving HEP analysis is to augment our current programming styles (often utilizing C++ or Python in a procedural manner) with higher-level approaches. Declarative- or functional-like models are a particularly advantageous way HEP analysts could focus on describing "what they want to do" as opposed to "how to do it." Analogously to how programming in C++ abstracts away implementation features compared to programming in assembler, it appears that these high-level approaches will allow the underlying implementations more freedom in utilizing the available computing resources. This would provide one mechanism to avoid having every individual learn the new skills necessary to utilize the hardware available during the HL-LHC era. The field already is seeing the initial R&D (individual analysis frameworks or projects like FemtoCode) and experimental (ROOT's TDataFrame, as presented by Pere Mato) forays into this area. It appeared clear to the workshop that this was an exciting area that could benefit from additional R&D.

As advocated by Gordon Watts, the HEP field can invest in improved techniques for software engineering via continuous integration, allowing a greater focus on fast reproducibility and provenance of analysis artifacts. How analysts interact with the computing system was also important to Eduardo Rodrigues, who emphasized the need for individuals to bridge cleanly from their local working environment to the distributed computing system. The current system, often focusing on a batch-system-like interface, may eventually give way to more query-focused or web-based "notebook" systems such as Jupyter.

A vision for query-based analysis

One proposed vision for analysis that was put forward by several people (Neubauer, Pivarski, *et. al.*) was for a query-based analysis where the user provides only essential information (event indices, description of information to access and code to operate on those indexed events and event information) and is given directly the "end" products -- summary objects from which physics inference can be, such as histograms, ntuples, tables, networks, etc. -- without intermediate data (e.g. skimmed data files). This is in contrast to the model common to these experiments which is to undertake a campaign of *data reduction*, producing a sequence of data files resident on disk in simpler formats (ntuples, typically) before arriving at the output analysis product. An often stated aim of the data reduction steps is to arrive at a dataset that "can fit on one's laptop", but one wonders whether some future infrastructure might make such comforts of having small, local ntuples files unnecessary. While the event processing rates and latency to make histograms that one gets on the laptop

might be difficult to match when the data is remote, it seems conceivable given future technology evolution, to approach local data performance with elements such as (a) nearby data caching and (b) scheduling of analysis workflow to give the time for the system to prepare the data for low-latency access of the data. Currently, there are couple of projects working to address certain elements of this model, include Femtocode from DIANA-HEP (see talk) and the Event Streaming Service from ATLAS.

The model has the advantage of *simplicity of interface*. Users provide only what is needed and get back the results they want. A related advantage is the *democratization of physics analysis*. As one gets further down the typical data reduction chain, the user (or small group of users) needs to figure out how to provision and manage the storage required to accommodate the intermediate data which in many cases is accessed with small ($<10^{-4}$) or zero duty cycle. For small groups, executing such a data reduction campaign might exceed their available resources, effectively "pricing them out" of contributing strongly to analyses. We heard from at least one user that this is an issue for their small group. Removing requirements on storing the intermediate data helps "democratize" data analysis and streamlines the overall analysis workflow. It would also allow such resources to be allocated for other purposes. (Transient intermediate data to e.g. provide low-latency access to indexed events may be created, and may or may not be exposed to the user.)

Session 3: Concrete technologies

Session conveners: Benedikt Hegner, David Lange
Speakers: Jim Pivarski, Sebastien Binet, Enric Saavedra, Chris Tunnell, Augusto Alvez Jr, Dirk Duellmann, Sascha Caron, Anton Poluektov

The concrete technologies session included presentations on emerging analysis tools and technologies, data analysis outside of HEP, and technologies that could become useful for HEP. The session included interesting discussions about data formats, languages, and user interfaces for analysis.

The theme of the importance of Python as a first class analysis language returned. Presentations included the popularity of Python across data science and programming (Binet), as well as at CERN (Tejedor), and how a small experiment can use Python as its end-to-end implementation. Python was also the language used for implementation of both Femtocode tool (Pivarski), some of the IT analytics (Duellmann) and Tensorflow analysis (Poluektov).

Provisioning and deploying analysis software on cloud and local resources was identified as a problem for which containers is a promising solution. Docker was mentioned in presentations (Tejedor) and discussions (on Tensorflow) as a way to generalize and expand facilities.

Data and tool interoperability was a theme. Data conversion tools between ROOT and other data science formats will enable use of new tools for analysis and will facilitate promoting ROOT outside of HEP. Related discussions included dedicated data formats for local/institute cluster analysis for use in special tools (or processors) and for analysis using low latency access to input data (e.g., memory only). Analogy with Proof as a motivation for scaling demonstrations for tools as they get developed.

Another recurring theme was functional / declarative style programming, decoupling problem description from implementation. A number of presentations included examples (Pivarski, Duellmann)

Towards modularizing ROOT, the question arose is MINUIT2 a good example of a piece of ROOT that would easily become an independent tool. An anecdote from an industry data scientist in attendance was that MINUIT2 was the one reason he had to continue to use ROOT in his work outside the field. “Other available minimization tools do not propagate errors properly.”

Session 4: Missing pieces

Session conveners: Peter Elmer, Graeme Stewart, Fons Rademakers
Speakers: Graeme Stewart

This session was intended to synthesize what has come before, return to interesting questions and discussion points accumulated during the workshop, address what has been missed in the coverage, what are the weak spots in the ecosystem, where are there opportunities for new work and new projects, what is the priority work to do, etc. In the end it took the form of a productive discussion around slides prepared by Graeme Stewart postulating a “typical” HEP analysis in 2027. Key elements of the 2027 analysis captured themes arising in the workshop:

- Easy access to bookkeeping information and other metadata. Common support for this across experiment boundaries.
- A declarative language in use to describe tasks.
- Analysis scaling easily and transparently to different resources such as clouds, grids and special resources like HPCs, based on a robust workflow engine capable of automatically recovering the problematic last 1% of a task.
- Easy integration with other tools, both in persistent formats and transient representations.

Other themes exposed or reinforced in the discussion included

- Python as a first class language
- We’ll still be using C++ as our standard performance language in 2027
- RooFit support is essential
- The model of successive stages of data reduction, finally analyzing a compact dataset with quick real time iteration, is still the model
- Data bridges between languages and formats. Reflection is an enabler.
- Physical files will probably still exist, but the model should be flexible enough to accommodate more radical ideas (object stores, query language interfaces)

Conclusions and outcomes

The workshop wrapped up with a session dedicated to conclusions, outcomes, actions and next steps, continuing the synthesis begun in the missing pieces session and extending into the later report writing session. Pere Mato, Liz Sexton-Kennedy, and Torre Wenaus convened. Specific questions to be addressed included

- What is the priority work going forward? Where do we need new effort the most?
- Where are the gaps and opportunities?
- Identify R&D opportunities, demonstrators, prospective/planned funding proposals, new/extended projects
- Strengthening and growing the analysis ecosystem
- What are the big risks and challenges, how do we mitigate and prepare for them

Common themes and important points

Here we draw together the prominent themes and important points from all the sessions, roughly categorized by topical area.

Development

Ascendancy of python. Should be a first class language. Needs full support. For the ROOT project, restoring support for PyROOT is a top priority.

- identifying personpower for this, even to pick up pyROOT support, is very difficult. The ROOT project is seeking new collaborators and partners.
- (XENON1T talk mentioned Numba, which allowed them to use Python in their trigger. (External tools.))
- Python interfaces to modern machine learning tools are prevalent.
- Offload performance-critical code to C++ libraries.

C++ will persist. We'll still be using C++ in 2027. While it's not the only high performance language, its performance will continue to drive its use as our 'standard' high performance language which supports abstractions. There is also a big legacy effect with about 50M C++ lines of code in HEP (an estimate from this workshop). If another language were going to threaten to topple C++'s role as the HEP workhorse language on a 10 year timescale, it would be visible now. While there are new high performance languages garnering some interest in the community, such as Go and Swift, they do not appear to be gaining much momentum in our community thus far.

Modularity is ever more important, including clean packaging and easy installation of components via package managers that manage dependencies and explicit version requirements. **Containers** are an ascendant technology providing the ability to assemble a working stack or toolkit and isolate it from the underlying OS, thus helping a lot with reusability, reproducibility and portability.

Decouple what you want from how you get it. Well defined APIs. Also important to save low-level provenance. Even if the low-level where and how is decided automatically and the data analyst isn't interested, it should be preserved so that the impact of bugs can be determined retroactively.

Event throughput is the definitive metric for measuring performance of workflows (but latency is more relevant for queries or end-stage ntuple iterations), file formats (except that file size is more relevant for long-term storage).

Tools and approaches

ROOT continues as a foundational layer, and a connector with other technologies. Its **reflection system** is a key to creating bridges to other tools and moving data between, should be regarded in itself as a foundational enabling technology.

More generally, HEP needs to **better leverage external tools** and **better promote its own tools** outside of HEP.

- Bridges, connectors, converters between ROOT and other tools and between ROOT I/O and other formats. Will enable use of new tools for analysis and another way to promote ROOT outside of HEP.
- ML tools from outside the field should be better bridged
- Many fields with increasing data volumes might benefit from our ROOT based I/O and analysis tools. Especially since our tools will be available and maintained for many years to come, which very likely is not the case for many currently fashionable tools.

ROOT I/O will persist as the key highly optimized data format and persistency technology for our I/O needs, but within an ecosystem in which others are important also. Accordingly, creating **data bridges** between formats, languages and tools is important for gaining full access to non-HEP tools.

On fast storage, ROOT I/O performance is **dominated by deserialization**. Compression default may be reconsidered.

Important for the end analysis is the I/O performance, **analyze where performance is lost** in current end analysis workflows. We could benefit from a systematic study looking at variations of ROOT-based storage (e.g. vectors or vectors of simple types) and see where performance is lost, thus giving us a way to easily perform a cost-benefit analysis in designing a storage format.

ROOT could use more help. If ROOT is to remain central and critical, how do we reflect that in effort, priorities, development plans.

Developments beyond PROOF. The PROOFlite (multi-process on a single node) functionality has been replaced by TProcessExecutor. There are ongoing 'demonstrators' to validate the idea that a Spark+PyROOT could replace PROOF.

Notebooks interfaces have already demonstrated their value for tutorials and exercises in training sessions and facilitating reproducibility. It is not yet clear how to use them for a real analysis unless they are connect to large computing resources (e.g. Spack clusters) to allow scaling. To be followed for the future.

Avoid too much re-implementation is a lesson from the past and we should work carefully to avoid doing the same when interacting with the (large) machine learning ecosystem.

Automation of workflows and the use of automated pipelines are increasingly important and prevalent, often leveraging open source software such as continuous integration tools.

Metadata handling (e.g. lumi). Provide a way “on the ROOT level” to easily find non-event-data associated with a given event**.** Every experiment has its own implementation of this, but some common core code and APIs could very much benefit everyone.

Analysis models

Most probably analysis in 2027 will be an **evolution** from present models. At least at present, a need for revolution isn’t apparent.

Our standard analysis model of **successive stages of data reduction**, finally analyzing a compact dataset with quick real time iteration, continues to be the baseline model. Both CMS and ATLAS use a series of

processing steps to reduce the large input datasets down to the size suitable for laptop scale analysis. That said, there is not a consensus on where the line sits between managed production-like analysis processing and individual analysis. This differs by experiment based on their driving needs and where the experiment is in its life cycle.

Beyond the conservative baseline analysis model, a declarative approach is a distinct model to explore for R&D. Separate “what” (isolated electrons etc) from how: simpler analysis specification, better performance due to optimizing backend.

Functional, declarative programming was a prominent theme. *Declarative* avoids over-specifying the order of operations. *Functional* avoids over-specifying sequential/parallel. This was part of a more general theme of splitting the “what I want” from the “how it’s produced,” which avoids over-specifying incidental details in general.

Local vs remote analysis: late stage analysis will neither be entirely local nor entirely remote. We will need to support both, e.g. support syncing from remote services like notebook based analysis-as-a-service into a local laptop environment to support ‘airplane mode’.

Analysis should scale easily from the laptop to cloud/grid resources to special resources, via robust workflow engines that automatically/transparently recover the slow tail of distributed computing failures.

Memory resident analysis for late stage analysis, for example an O(10TB) memory pool on a cluster as shared analysis facility, came up as an important approach for the future if technologies such as Xpoint mature and become cost competitive.

Facilities and services

Sharing infrastructure will be more and more important, particularly in the context of **analysis as a service** systems. This includes sharing specialized facilities and services such as GPU farms, TPU farms, clouds, memory-intensive systems and web based services. Mechanisms to share them elastically support easy, efficient use. **CVMFS** is the backbone tool for deploying software; a complementary approach gaining a lot of interest is **containers** for deployment on the Grid, Cloud, HPC and local resources.

Provenance and preservation

Code is Data. Caching code for reproducibility/preservation, efficiency with multiple queries, translating code to separate the what from the how. (But software is a special kind of data that can be a creative, executable tool that operates on data. See https://doi.org/10.7287/peerj.preprints.2630v1 for another view.)

Reproducibility is becoming something we have to take seriously. We may start seeing papers audited for reproducibility by journals and/or funding agencies, as is happening in other fields.

ROOT roadmap feedback

One objective of the workshop was to provide developer-oriented feedback to the ROOT team on their development roadmap. A general theme was that ROOT should focus on key ingredients: I/O, Math Libraries,

Histograms, Graph (e.g. event display and graphics objects representing particle physics results), Foundation and glue. Focus on pieces that are HEP domain specific and fundamentally better than the alternatives and deprecate the rest. Provide efficient bridges to the analysis ecosystem beyond ROOT.

The following points emerged:

1) PyROOT is vitally important and must be properly supported. A long-term maintainer needs to be identified. PyROOT could be upgraded to build just on top of CLING, this should be in the plan, since PyROOT is also interesting for the non-HEP community ("a world-class, unique, fantastic binding" was one comment). Direct interfaces of C++ objects to NumPy are also needed avoiding any unneeded memory copies.
2) RooFit continues to be essential
3) Minuit is also important, albeit below the level of PyROOT and RooFit. Is MINUIT2 a good example of a piece of ROOT that would easily become an independent tool?
4) Recipes for common package managers (spack, conda, homebrew, etc), slim easy install. Modularization could enable wide range of recipes (late installation of pythia, hdfs, xrootd bindings)
5) Support for other persistency backends
6) Efficient conversions from transient TTrees into other library formats; NumPy would be the obvious first target

Actions and R&D prospects

The following is a list of prospective actions and R&D topics suggested by the common themes and important points developed in the meeting. Rough estimate of the timescale in years is given in brackets.

- Establish PyROOT support and identify other needed measures to promote python as a first class language. [1-3]
- Modularization and plugin support plan and roadmap for ROOT et al [1-3]
- Inventory the needed bridges, connectors and converters, identify which are missing or need further work, develop workplans for them [1]
    - Support for efficient conversion from ROOT Tree format to NumPy C structs (first good test of transient representation bridge)
- Interfacing external ML tools: inventory the needs and develop workplans [1]
- ROOT I/O performance analysis and tuning [1-3]
    - Can we systematically understand where performance hot spots are? Analyze where performance is lost in current end analysis workflows.
    - Can we usefully define several 'optimization points' that are tunes for particular usage modes (reco output typically write once read rarely, optimized for compression; final analysis optimized for turnaround time; streamed data optimized for over-the-wire efficiency)
    - Can we get fast I/O on simple data models? How important are complex data models for the community?
    - On fast storage, ROOT I/O performance is dominated by deserialization. Revisit compression defaults.
- "Helping ROOT help us". Within the experiments, examine how internal ROOT related work and software can be constructively aligned with ROOT itself and the work of the ROOT team. Identify ways to expand the ROOT self-support community from within the experiments. Help the ROOT team tilt the balance between user support vs. development, more towards development. [1-3]

- Development roadmap for next-generation analysis services beyond PROOF. Assess the relevance of AaaS and notebook for the analysis community, including syncing from remote services into a local laptop environment to support 'airplane mode'. [1-3]
- Study eventual need, requirements and chance of adoption of workflow automation -- generalization & integration of workflow management systems as general analysis services. Utilizing robust workflow engines to automatically/transparently recover the last 1%. Several implementations exist; why have they not seen large-scale adoption? [3-5]
- Metadata handling: Common format/toolset for non-event data? Provide a way "on the ROOT level" to manage and find non-event data associated with a given event. Establish something as common as TTree for event data? [3]
- In the area of software life cycle and sustainability, ecosystem components become dependencies. We need to broaden support for effort to maintain the ecosystem. Establish a community building model for building and sustaining the ecosystem. [1]
- R&D on how new hardware technologies could affect the analysis tool chain [1-5]

Links and references

- Workshop links
  - Indico timetable
  - Live notes
  - Previous, frozen Draft 1 from June 7
- Preparatory materials
  - Googledoc used to develop the agenda
  - Googledoc used to plan the workshop
  - Google Form for survey of currently used analysis tools
- Workshop reports
  - CERN EP/SFT group trip reports
    - Today's ecosystem, Gerardo Ganis
    - Vision, Axel Naumann
    - Missing pieces and conclusions, Pere Mato
  - Workshop report to ATLAS, Attila Krasznahorkay

Appendix 1: Workshop organization

We received comments that the workshop was well planned and organized and we should document what we did. An outline…

- Developed the workshop idea as a proposal to the HSF open forum (about 5 months before the workshop), evolving the idea in a googledoc
- Developed the plan for the workshop through ~weekly discussion in the open HSF meeting
- Wrote a detailed agenda outline (about 2 months before the workshop) and nominated several expert conveners for each session
- Put out an open solicitation for a venue "close to CERN" to the community
- Delegated the lead (but not exclusive) role in proposing talks, identifying speakers and assembling the session agendas to the session conveners

- Charged the conveners with identifying scribe(s) -- themselves by default -- to take real time notes in a live googledoc
- Built a report drafting period into the last day of the workshop.

In general, plan early, plan openly and in some detail, enlist experts and engage them in the planning and preparation, and attract a great venue.

Appendix 2: Detailed session notes

Here follow the session notes compiled primarily live during the sessions by the conveners and others.

Today's analysis ecosystem landscape and toolset (Monday)

Session conveners: Attila Krasznahorkay, Paul Laycock, Andrea Rizzi

Pere Mato - ROOT

Pere Mato Vila mentions community contributions to ROOT. For comparison of see AstroPy's
https://github.com/astropy/astropy/graphs/contributors ~200 contributors, ~600 publl requests, 2011-2017
https://github.com/root-project/root/graphs/contributors 123 contributors, ~4000 pull requests, 2000-2017
Would be interesting to look at activity beyond the core ROOT team.
User support estimated to consume about 50% of core team efforts.

Discussion of Modularization. ROOT -> BOOT. Interface with popular package managers like Spack, Conda etc.

Discussion about new C++ interfaces & functional chains (TDataFrame).
Perhaps we should think more about API Specification?

- Common in areas like deep learning: eg. Keras (https://blog.keras.introducing-keras-2.html) *Keras is best understood as an API specification, not as a specific codebase. In fact, going forwards there will be two separate implementations of the Keras spec: the internal TensorFlow one, available as tf.keras, written in pure TensorFlow and deeply compatible with all TensorFlow functionality, and the external multi-backend one supporting both Theano and TensorFlow (and likely even more backends in the future).*
- This is also relevant for the JavaScript discussion.
- See also Jupyter. It is reference implementation for the messaging protocol between kernel and UI. But other implementations, like hydrogen https://atom.io/packages/hydrogen

Need to worry about possible performance issues with more abstraction (c.f. C++ libraries).

Danilo: try to hide as much concurrency as possible from analysts, but cannot get to ultimate performance that way.

Do modules help with removing deprecated components? Possibly not as an end user doesn't really care if a critical component for them is a module. This is a difficult issue - needs to be part of the landscape.

Pere notes 2

ROOT's Place and its Capabilities in the HEP Analysis Ecosystem

- ROOT is at the centre of the analysis eco-system
- Many contributors over the years, to many different components. Git helps a lot with this.
- Core containers are not recommended anymore. (std:: containers should be used instead)
- 2D graphics is a very large amount of code.
- SQL code: who is using it?
- R binding: which direction is this binding?
- Additional effort in Python bindings could definitely be added.
- Much effort going into JSROOT these days.
- PyROOT usage on MacOS with its latest built-in protections?
- ROOT 6 packages for Ubuntu?
- How many clients use ${ROOT_USE_FILE}?
- Master repository is in GitHub now.
- How do core developers modify the code? Through pull-requests?

Peter: How much of the effort is user support, and how much into "core software"?
Maybe 50% of user support? (Or even more than that.)

What could be dropped from support?
To be discussed...

- How will multi-threaded/multi-process code look like with TTreadExecutor/TProcessExecutor?
- How many heavy-lifting reconstructor code will pick up SIMD types from ROOT?
- Prioritise what to parallelise first?

Liz: How many lines of code we would get rid of with JS? Hard to tell.
Code is not strictly in specific modules.

ML "inside" ROOT? Or rather just export data?
Need efficient way of providing ROOT data to the (python) tools. How to compare different tools?

Extending ROOT with VecCore? SIMD capabilities?

Kyle: New interfaces should be rather API definitions than about actual code. New ML code (Keras) is working that way. For instance missing clear specifications of ROOT I/O at the moment.

How to deprecate interfaces/modules?

Jakob Blomer - HEP File Formats

Suggestion that bcolz is interesting: https://github.com/Blosc/bcolz http://bcolz.blosc.org/en/latest/
Use case here is towards the end of the analysis chain (1.5GB of data, fits on a laptop). Also need to care about 100s of TB on grid to get here.
Ease of use is an important criterion.

Graeme Stewart - ATLAS Analysis

I (JT) didn't raise this as a question due to time … from a PhD student / local institute point of view, something that needs solving is a standard, supported, working model for analysis code integration at the last-mile stage.

What we see happening is that students have a mix of ATLAS/ROOT code, Python, and Python 3rd party machine learning (or other) tools. The integration of these takes a lot of time, from the outside (I am not ATLAS) it appears to go like this: use of ATLAS software ties one into CVMFS/ATLAS; in order to have this work, one has to use the ROOT version in the ATLAS/CVMFS repository (which some people aren't happy with) … using this stuff as a library from python tends to require Python to effectively come from ATLAS/CVMFS (same gcc versions) … then the students try to install 3rd party python stuff that is not included in ATLAS CVMFS and get horribly stuck due to them not knowing where to find step-by-step documentation on how to do this. (See slide 12 of Eduardo's talk, python enthusiasts often have their own installation).

- DxAOD production time dominated by file writing/merging
- lxplus is a reference point. Very much needed.

Gianluca Cerminara - CMS Analysis

Converting out of miniAOD to other formats is ad hoc.
microAOD intended to come in a few flavours, depending on the analysis (but not 100s).

- Can MET be recomputed from miniAOD? (Apparently yes...)
- CP analyses? Tracking efficiency calculated from miniAOD?
- When not using FWLITE, how do analysis codes read miniAOD?
- Batch processing more important than in ATLAS.
- What sort of code would run on microAODs? (ATLAS D3PDReader lesson...)

Philippe Canal - Intensity Frontier Experiments

- Wishes:
  - Tighter integration with python
  - Data formats to connect to open-source tools
  - Not concerned about performance
- Documentation:
  - See also forward-looking documentaiton in LSST: https://community.lsst.org/t/introducing-lsst-the-docs-for-continuous-documentation-delivery/781?u=jsick

Need to ensure that languages and features are used appropriately, not for the sake of it.
Data formats came up again and the lack of a standard one on the HEP side (maybe if we had one other tools would develop the means to read it).

Last slide (about parallel with Fortran-to-C++ transition and mentoring / senior people): this is essentially an aspect of sustainability, which we need to address in the CWP.

- 1-2 order of magnitude less collaborators than in LHC
- Which parts of CMSSW is in Art?
- NOvA analysis ntuples directly out of reconstruction?
- FIFE: Grid middleware (Built on HTCondor, etc.)
- Very important to teach senior people about the new technologies that we may switch to.
- Fortran->C++ switch similar to current C++->ML one?

Eduardo Rodrigues - LHCb Analysis

Alignment is very stable and very fast to redo if needed, so can be done ‘real time’ (~10 minutes).
Comment on slide 12 about Python enthusiasts with own installation mirrors my (JT) comment on Graeme’s talk.
µDSTs made in a stripping process, there are many different types (each line only requires a few 100 lines of python).
Most analysts use standard toolkits to convert DST or µDST into ntuples.
“We simply cannot in the future analyse data the way we do it today !”
Analysis preservation is being developed in an experiment agnostic way.

- Calibration done in ~10 minutes
- There are analyses done directly on the HLT results. For instance PID calibration.
- Few hundred microDST coming out of stripping.
- GPUs are already used for analysis. (pyCUDA!)
- How much analysis happens in the microDST making? How much analysis happens during the ntuple making?
- Private MC production?

Martin Ritter - Belle II Analysis

Jupyter kernels can be submitted to other machines.
Expect physics groups to contribute more and more tools.

- CentOS 7?
- DESY pays for Atlassian tools.
- Condition data handling very similar to the ATLAS/CMS Run3 effort. Any common development there?
- Much of the analysis code part of the software release. Part of central validation.
- Liz: Number of people writing analysis tools? 10-15 Hopefully with data taking will see new people joining the effort.
- C++ user code in grid running is not solved yet.
- Jupyter tasks running on a cluster/cloud? Or only on a local machine?
- Size of final ntuples?
- How many analyses covered by the presented model?
- Docker used in analysis preservation to manage the user analysis part.
- Physics groups not "fully formed" yet. Organisation will become more complicated later on.

Steven Schramm - Machine Learning

Presenting survey on ML, mostly LHC.
BDT most common but summing various NN is same amount, people moving away from TMVA for modern NN
Comment: No feedback from theory community
Peak usage for final analysis and reconstruction (such as ID, tagging etc..)
Many (really many) toolkits on the market.
Main complaints for TMVA: no GPU, too slow, my preferred ML technique/feature not there, lack of docs
Comment: for writing a CV TMVA is not cool (as in “TensorFlow” skills are better on a person CV)
People from survey want to “kill TMVA” and prefer interfaces to external tools

The main push to move away from TMVA is deep learning (looking at survey correlations)
Collect solution and tools developed to adapt external tools to HEP (many solution exists privately)
GPU mostly used for NN/DL training
Discussion: how do we measure GPU to CPU improvements… not too clear but people claim “much better than a factor 2”
Not clear how much DNN is about hype and how much is real performance, but for sure real performance effects in HEP are seen.
Comment: we are still at the beginning, we may still have to find the best way to use DNN in HEP
Question/Comment: are GSOC students on TMVA consolidation devel and/or on external tools adapters
Comment: which external tools should we invest on? All of them? How easy is to switch? *Answer:* better focus on few
Discussion: Provide interface to Keras (because keras is already a wrapper, at least for Theano and TensorFlow) or go to Theano and TensorFlow directly… or have yet another layer with TMVA
HEP community should contribute and help with features we need (e.g. negative weights) that could anyhow me interesting for IT community too.
ML tools are new every few years, hard to keep TMVA updated. Converting from one external tool to another is also supported by the DL community (for DL existing tools).
Interfaces of external tools are “nicer” than TMVA.
TMVA not being adopted by starting experiments (belle II)

- Most people use BDTs.
- TMVA still dominates, but not as much as a few years ago.
- Significant speedup with GPUs is a widely accepted experience.
- Much discussion about TMVA's role...

Andrew Gilbert - Final steps of analysis

Last step: extract the various type of measurement from HEP analysis, often combining experiments.
RooFit used as a framework at LHC experiments.
RooFit very widely used for defining and testing models. Used together with MINUIT most of the time.
RooStat built on top of RooFit and ROOT: implement commonly used stat techniques (profile LL, FC etc..)
RooWorkspace: preserve links between vars and functions (not just a dump “by value”)
High level tools exists to build the models with O(10) channels x O(10) processes x O(100) systematic variations
Examples: “HistFitter” originally from Atlas SUSY, “combine” originally from CMS Higgs
Not always possible to combine at likelihood level.
Several way to talk to the outside: opendata (event by event), rivet / unfolded distributions (TUnfold,RooUnfold), HEPData, publish simplified likelihood.
Comment: additional approach to go the other way (model to detector, aka recast)
Recast offering facilities to use experiment MC directly for unfolding
Comment: similar issues on Neutrino experiments (model for neutrino beams)

Comment: “Rarely done: publish the full likelihood model (i.e. the RooWorkspace)” (slide 15) - this is probably what our national funding agency would ask you to do … they are asking for a “replication package” that future colleagues could download and would include all relevant stuff allowing to reproduce the analysis. Here is the NWO definition for the replication package, for inspiration:

“Replication package - the full set of data, metadata including the persistent identifier and provenance information, documentation, possibly a description of the required software, hardware and tools, as well as a reference to where the log books, lab journals, research protocols et cetera can be consulted. This package of information is archived together with the data. It is the set of details and information needed to be able to reuse and replicate the data.“

This is written in a data-publication specific way, but the same will hold for the analyses.

Round table on Data Analysis

Mark: Can we think about providing events in such a way that we don’t need intermediate formats? Random access on huge datasets is hard. Can it be scheduled?
Philippe: bottleneck is not i/o, but it deserialisation (too complex data model?)
Attila: ATLAS data model is a lot simpler now: vector<vector<float>> is as complicated as it gets
Frank: miniAOD analysis in CMS runs at 5Hz, which is ludicrously slow

ROOT team lacks effort to support pyROOT at the moment. Python libraries can give performance (numpy) and concurrency. C++ will exist for the frameworks and ultimate performance.

For non-event data need to define the problem exactly.

GPUs - there are many different problems in the analysis domain. Only in some are GPUs a solution.

Insisting on TMVA is a dead end for sure - that attitude will die. Seems like consensus that bridges to other tools are needed.

Kyle: On the fly feedback for MC generation can be a complete game changer.

Functional programing: style that suits our problems? Doesn’t fit well with ROOT’s procedural interfaces. TDataFrame going in this direction. Can our community express the problem that way? C++, Python can easily have side effects (usefully!).

The analysis ecosystem vision (Tuesday morning)

Session conveners: Brian Bockelman, Axel Naumann, Gordon Watts
Session speakers: Fons Rademakers, Max Baak, Eduardo Rodrigues, Gordon Watts, Gerhard Raven, Pere Mato Vila, Axel Naumann

Fons Rademakers - Analysis hardware and computing facilities in 5-7 years

(Note - order of talk was swapped with the other)

- CERN OpenLab - engaging external companies to get started on the next generation of hardware/software from partner companies. Typically looking ahead 3-4 years; OpenLab around since 2001.
- What was around in 1995 when ROOT started?
    - Hardware: $4,400 got you a 486 66Mhz / 8MBRAM / 320MB HDD. In 1996, HP PCs running Linux 1.0 on Pentium Pro. Wrote some of the device drivers!

    - Software: Linux 1.0, WinNT 3.51, Win95. Amazon / yahoo / ebay created. Java released. C++ 10 years old - but not standardized and no STL.
    - HEP hardware: Pentium Pro, end of MPP era. Future was to be Intel Itanium.
    - HEP software: Windows everywhere (Linux is a toy but not serious), Global OO Objectivity database, commercial software for math, 2D/3D gfx (s/w for LHC era surely too complex to be developed by physicists). Open source - fears about quality and sustainability. C++ or Eiffel - surely an OO language.
- NVRAM: 3D XPoint greatly decreases the latencies again.
    - Will likely disrupt how we approach OS disk buffers. DMA / mmap vs. POSIX.
    - Reduces need for caching or read-ahead.
    - Will move bottleneck to compute part; language becomes relevant
- Code needs optimization for tomorrow's CPUs. Example: 700 line kernel from brain cell growth, 320x speedup.
    - Normal story: fix memory allocations, AoS to SoA, OpenMP, Cilk++ scatter/gather.
    - Lots of expert-level stuff.
    - Top ten European and one US American. Cannot look for keywords for a hire; smart motivated people do it, no matter what they learned.
    - Whenever feasible, these optimizations should be done by frameworks and experts, not by users.
- Hybrid CPUs are coming out - FPGA inside the Xeon processor. How can we take advantage of it?
- Things to consider: Electron WebGUI, WebGL; Swift, interpreted and compiled (but still missing C++ binding)
- Go out into the world and explain what ROOT etc are good at
- Questions from audience:
    - Graeme: Xpoint == DRAM price, thus very expensive for mass storage

Fons notes 2

- OpenLab - looks 3 to 4 years ahead to see how it would work in our environment (for 16 years)
- 1995 - looking into very long instruction word and pipe-lining instructions. So, nothing has changed.
- 3D XPoint NVRAM (Intel, Micron)
    - Almost DDRAM speed as a hard disk (e.g. a RAMDISK that is like a HD in volatility)
    - Fix/Improve the latency problem. You don't wait for I/O - how do you change your solutions?
    - Reliability is much improved.
    - SATA bus first, and RAM slots later (OS Support Required!)
    - 1000x lower latency than SSD (NAND), 10x denser than DRAM, 1000x durability of NAND, Cost of DRAM (so 1TB is ~$4K)
        - 4GB DRAM currently $16, so 1TB ~ 4k$ (https://epsnews.com/2016/11/22/will-dram-prices-continue-rise-2017/) (thx)
        - c.f. HDD prices of 2c per GB (x200 cheaper) - this won't replace the disk in most laptops for a while
    - Changes: how you do I/O. All our efficiency optimizations have to change (e.g. how you layout branches, etc.). Programming languages are a bottleneck (e.g. Java, etc.) -> more cores as you can now feed them.
- Many-Core Co-Processors (and code Modernization)
    - For certain problems (e.g. Brain Cell Simulation) 320x speedup relative to KNC single cpu - so this is a test of doing it in parallel (by just a French guy)
    - Code improvements to take advantage:

        - Custom memory allocator for small arrays
        - Array of Structures -> Structure of Arrays
        - OpenMP
        - Compiler Intel - does scatter/gather
        - Missed the bull about European vs American.
- Hybrid CPUs
    - Can put so many cores on, but more cores than you can feed for most people
    - Add custom co-processors (compression, random numbers, encryption, etc.).
    - What would HEP like to add?
- Software
    - New software technologies (Typescript, electron, WebGL - new ROOT 2#/3D backend)
    - New languages should be integrated
- Soapbox
    - ROOT I/O - keep testing to see what is faster
        - Analyses still I/O bound
        - So slow language is not yet a problem
        - But this will change as soon as things like XPoint show up
    - Vectorise a parallelise and use GPU's

Max Baak - Consultant's view on HEP analysis software

Max is the *Chief Data Scientist* at KPMG, and a former particle physicist who leads a team of 20 and works with clients to setup and deploy big data analysis.

- (By an ex-physicist turned data scientist)
- Max Baak "Chief Data Scientist" @ KPMG. Leads a team of ~20 folks that works with clients (big companies) on setting up a data / analytics strategy, provide insights, and initial project deploy/prototyping.
    - Example project: customer store card use. Modeling to predict when a card can be "written off" and how much money to transfer between stores.
    - Credit card fraud detection: Determine which cards are most likely to be stolen considering where they are used; determine likely location of a compromise.
    - WiFi tracking: tracking where a phone is going - used to follow people walking throughout a store. A lot of overlap with tracker detecting!
- Typical datasets: no filtering on data, relational data. Often structureless. Security/privacy is a concern. Quantized variables as opposed to continuous.
- Typical tools: lots of the "usual suspects": language tools (python, numpy, pandas, R), frameworks (spark, hadoop, hive, zeppelin) and infrastructure (java, docker). personal favorites: pandas, spark, docker.
- When migrating to industry, was pleasantly surprised by how much is "out there" in the Open Source community.
    - Everything is out there - HEP should be using it!
- Questionnaire/observations: SAS/R is the most popular data analysis framework, Python is the most popular language for data science / machine learning. Used ROOT only sparingly (most folks are unfamiliar with it).
    - Would be great if impact of HEP's software contribution to the outside industry "would make the same impact as reputation of the field". Not claiming there are *no* contributions but we could have much more!

- Retrospective on ROOT: law of the handicap of a head start (initial head start in a given area may result in a handicap in the long term).
    - If you could start from scratch, would there still be a need for ROOT?
- HEP has an obligation to actively use and contribute to existing open-source tooling.
    - Use existing open source where you can and contribute where it needs to be improved. Similar to the “make Google work for you” approach.
- Tips for migration of ROOT:
    - Could become a serious competitor of R.
    - Split up and make a light-weight version of ROOT. Make it easy to install.
    - Replacements: persistence -> protobuf/avro/capn’ proto. PROOF>spark.
    - **Actively support and test with homebrew and conda / anaconda!**
    - Make python support as important as C++.
    - Actively support spin-off activities: sparkroot, protobuf, anaconda installation, rootpy.
        - I.e., there is a ROOT in anaconda but it is a PhD student project and may die when the student finishes.
        - Absorb them to make ROOT future proof.
- Questions from audience:
    - Dirk D.: What is the biggest “missing thing” from ROOT? Category based analysis.
    - Brian B: What tools in training are we missing / how do you approach education? Work with local data analysts and teach them on the KPMG toolkit.
    - Federico: If you were king of ROOT tomorrow, what would you do? (1) Increase focus on / integration with Python, (2) replace IO backend, (3) Absorb spin-off activities.
    - Attila / Pete: Do we feel like ROOT has the mandate to do distribution or push ROOT beyond community? I.e., Ubuntu has a very old version - how can we do outreach beyond the community?
    - Pete: We don’t actively go out to data science conferences, into other communities. We need
    - Jakob: Why the interest in the other persistence layer (why protobuf vs ROOT IO)? Interoperability with other tools and languages. Bigger / better support ecosystem.
    - Liz: How long does your data last? Does industry have the same data longevity requirements as HEP? Claim that industry tools can allow this - but they’re still a young team.

Max notes 2

Max Baak presented his view on HEP analysis software as a former physics now in working as Big Data consultant. Max is the Chief Data Scientist at KPMG, who leads a team of 20 and works with clients to setup and deploy big data analysis. Three example of projects tackled by KPMG:

- Example project: customer store card use. Modeling to predict when a card can be “written off” and how much money to transfer between stores.
- Credit card fraud detection: Determine which cards are most likely to be stolen considering where they are used; determine likely location of a compromise.
- WiFi tracking: tracking where a phone is going - used to follow people walking throughout a store. A lot of overlap with tracker detecting!

The typical datasets they are working is ‘raw’ data without filtering some relation in nature, some structureless, one major concern is security and privacy of the data. Most of the variable are quantize (as opposed to continuous). The tools they are typically using includes the “usual suspects”: language tools (python, numpy, pandas, R), frameworks (spark, hadoop, hive, zeppelin) and infrastructure (java, docker). His personal

favorites are pandas, spark, docker. When migrating to industry, he was pleasantly surprised by how much is "out there" in the Open Source community and he recommended that HEP takes much more advantages of that. He also recommended that rather than developing our own that ends up not being widely used that we make a bigger impact by contributing (more than we already do) back to widely used project. This should contribution should "make the same impact as the reputation of the field".
At the moment ROOT is somewhat victim of the law of the handicap of a head start (initial head start in a given area may result in a handicap in the long term). However he feels that ROOT is in a unique position and could even mount a serious challenge to R.

He recommended that ROOT increase significantly its focus on Python in particular and interoperability in general. Python should be 'as important' as C++.
He also recommended to make ROOT and lighter weight to install, likely by splitting it up and actively support package manager like home-brew or anaconda. There are several spin-off activities that deserves to be actively supported, for example sparkroot, protobuf converter, anaconda installation, rootpy. ROOT should consider to absorb them to make itself future proof. For example the anaconda support for developed by a PhD student and may die when the student finishes. He also wishes for more integration of ROOT with other ecosystems. His vision for this integration called for replacing the I/O backend by (multiple or one of) industry standard protobuf, capn' proto, avro. He did not have a good answer to the question on the longevity of those file formats. The other significant missing piece in ROOT is the ability to do category based analysis.

One additional comment from the audience was the realization that HEP does not actively go out to data science conferences, into other communities and should.

Eduardo Rodrigues - My Vision

- Worst enemies of analysts.
    - Lack of knowledge, inertia
    - Inappropriate or not timely available software working environment.
    - Inappropriate analysis software ecosystem.
    - Complexity of analysis.
- Evolution of the analysis ecosystem: starting to adopt python, but was very late to machine learning.
- Current LHCb tools are not performant enough to survive the next 3-5 years.
- Very important to improve python support - "expose ROOT in Python"
- Toolset vs Toolkit -- move more to a toolset model.
- SWAN-style frameworks are useful introduction. Focus is on things that "just work".
    - How does this interact with the distributed computing system? Cannot scale to full analysis
- Everware - allow running of code / code sharing.
- Large ecosystem: tools are there, need to focus on interoperability.
- Sustainability - need a sustainable underlying ecosystem, particularly as a bridge between projects. DIANA/HEP is an example of this approach.
- SciKit-HEP: modeled after astropy. Core packages with a strong affiliated ecosystem.
- LHCb Turbo Analysis is the way to go, also for other experiments
- Questions:
    - Mike S.: As the data rates go up, the analysis tooling may really change. For example, if you need to put the tools in the trigger itself (as LHCb will in Run3), your approach may completely change.

Eduardo notes 2

- GRID - our interaction with tools is not that great
- Build/run tools
    - What we currently have (cmake, etc.) not fast enough
- External vs Internal
    - We should not try to compete with dedicated tools.
    - Central to experiments
- ROOT: Python should be first citizen (Python3 too)
- Working environment
    - Getting something up and running - almost impossible
    - SWAN/Juptyr - good for little studies, etc. But not analysis. But it is a nice way to have something that “just works”
- Everware
    - Tool to edit and run others code, even if complex
    - http://everware.xyz
    - Built in python, jupyter, docker
- Current Ecosystem
    - Summary: Tools are mostly there, but our interoperability story is not there
- Sustainability
    - Too many single person projects
    - DIANA/HEP will help out with this
- Scikit-hep
    - http://scikit-hep.org
    - Expand typical toolkit of HEP folks
    - Common definitions and APIs to eash cross-talk
    - Community
    - (not LHC centric)

Gordon Watts - My Vision

- Background: This presentation describes work done for a few papers (so it actually exists). One explicit goal was to be able to work even with low duty cycle (i.e., professor actually gets his hands dirty!).
- Complex analysis (at the physics level): 1-2TB of input data - flat ntuples - from the grid goes into the framework. 340 branches, 500 plots as output. Written in C#
- Focus on declaring what he wants to do, as opposed to encoding *how* to do it.
    - Instead of event loop, a function chain.
    - Can utilized custom functions for highly detailed procedural work.
- Backend pieces: declaration (physics) -> translator (run planning) -> TSelector code (C++) -> Executor (actual running). User sees the declaration, but can change backend (both local executor and PROOF cluster was attempted).
- Your analysis in one file:
    - Analysis is a mess of scripts and procedures. Can we clean this up?
    - One way to clean this up is a Makefile -- but you have to write/maintain this.
    - Wrote a dependency analyzer that looks at code and figure out the dependency graph. Write code, not dependency relationships.

        - Possibly less-efficient *but* reduces/removes the need for shell scripts. Decrease in human time is worth the possible increase in resource usage.
    - Includes running / fetching grid data.
- CI server for analysis.
    - Push to repo triggers run of analysis.
    - Output artifacts: training, log files, ROOT files with plot. Can get history of analysis evolution - and artifacts.
- Strongly typed information allows better editor support (refactor, type analysis, etc)
    - C# compiler can output AST, allowing simple manipulation. Examine the AST to determine the branches needed - delay final compilation until we actually see the files.
    - Auto-translation of AoS -> SoA, for example.
    - Can use query written as a key into a cache. Analyzer can help decide what parts of execution to skip, allowing significant speedup to "human" timescales (minutes, not hours).
    - Can combine/rewrite multiple steps together.
    - Need to integrate rapid prototyping / dev and analysis production in the same system.
- Using CI system improve reproducibility. Embeds EXIF data into JPEG as a key to trace back to ATLAS production system.
    - Capture selection expression, input files, code git hash etc
    - Can completely re-run the analysis based on the JPEG itself.

Gerhard Raven - My Vision

- Background: LEP, BaBAR most recently came from LHCb - did some complex analyses, led on the trigger.
- Claim: you can take a LEP-era physicist and they would be familiar with LHC analysis today. All is recognizable.
    - Did Moore's Law allow us to be lazy or conservative?
- Evolving code is all about taking away control. . As you evolve from structured -> procedural -> object-oriented -> functional -> declarative, various pieces of control are taken way (remove modularization, then mutable state, then control flow, etc.). Can help think about higher-level abstractions (for example, gives compiler room to optimize).
- Imperative programming - which HEP loves - enables illusion of control. Alternative we should invest in is functional / declarative.
- We need to work to better separate into frontend (physics/what) and backend (how).
    - Better allows backend developers evolve freely.
    - Risk: furthers the split between analysis and computing knowledge/people. (black box risk)
- We need to better establish provenance/vocabulary. People make their own ntuples because they can mentally understand the provenance of "pT".
    - Git-like version tracking and allows dependency tracking.
    - The LHCb trigger is using a blockchain to watch for updates/transactions to the trigger config (novel!). Builts a "trust" system.
        - Aside (Brian): Every job that uses CMSSW does this, but about 100% chance no analyst knows/cares it exists
    - Follow things when making ntuples - at the leaf level
        - 90% of the ntuple is the same on every re-make… so why not just make the new leaves?
- Why don't we use AOD/xAOD/uDST

    - We alway use a flat ntuple
    - Will automated pipelines, etc., fix this problem?
- Funding agencies will demand reproducibility
- Outside world - take advantage of all the work there
    - Contribute back to projects for stuff internal to us so they grow in the community and are supported, etc.
    - What if we pick the wrong one? Own it!
- Automation is the future
- “People can not contribute because computing nowadays is too complex” -> actually, it may be due to things that have built up over many years and/or inertia. Need to tackle existing code: rewrite, simplify.
- Can only convince by showing the complete analysis chain, must be simply better (faster, precise etc)

Pere Mato - ROOT Vision I

- TDataFrame - functional approach to looking at the the data, much like above, but deep inside ROOT
    - Generates futures
    - Can coalesce various queries
    - Correctly interleaves code as long as they come from common sources
- Implementation
    - TTreeReader based (so pretty efficient)
    - Uses “strings” to define new columns
    - Python linkage - strings as cuts/code make it easily move between worlds
    - Ability to write out the data format
    - Can you add new extensions, etc., without hacking the source?
    - Aiming for ROOT 6.10

Axel Naumann - ROOT Vision II

- ROOT should be simple and efficient
- Efficient
    - Productivity is the figure of merit
    - Solution oriented coding approach
- Debugging:
    - Can get to the point that all errors are compile time errors
    - ROOT - no need, because it is tested (well, working on it)
- C++
    - HEP is around 50MLOC right now
    - Gives speed, etc.
    - Needs education, etc.
- Python
    - Always part
    - Not there for the “hot” number crunching
    - PyROOT - we should put it on top of cling, but we don’t have it.
    - Someone has to fix this.
- Other language bindings
    - Many provided by the community

    - JavaScript - will become integral into ROOT
    - C++ will do the heavy lifting (I/O, calculation, etc.)
    - Data pipes into other languages
        - TDataFrame style interfacing
        - PyROOT should map branches to numpy arrays
- Simplicity
    - Simple install, use (no setup, docs), a good place to ask and answer back.
    - Interfaces of ROOT were defined 15 years ago
    - Modern libraries - should be able to use them inside ROOT - what do we need to do to get there?
- Vision
    - Simplifying, Core improvements (I/O, math/hist, graphics, foundation)
    - Concurrency under the hood
    - Glue options/no re-implementing
    - C++ and PyROOT as the core.
- ROOT is everywhere
    - The whole community uses it
    - How do we make it part of the future and the future part of it?
- Questions
    - Read into root all formats - also write to other persistent formats from ROOT
    - There seems to be consensus that putting effort into making a direct connection from pyROOT to cling would be a great way to modularize this part of the ROOT ecosystem and there would be more interest in the outside world for ROOT if this were done.

Concrete technologies (Tuesday afternoon)

Session conveners: Benedikt Hegner, David Lange

Jim Pivarski - Data formats and conversion tools

Many paths to convert from one format to another. Many formats trying to position themselves as the interoperable format, ROOT could play to its strengths by having more persistency options for ingesting and exporting.

ROOT converters are a clear entrance for ecosystem portability

Sebastien Binet - Survey of languages for analysis

Can HSF github repo help track interesting tools?
Keep communication at least a layer above the file format (code reuse)

Enric Saavedra - Swan

Python dominant language used.
Q. Can users open an xterm in their browser connected to SWAN? Not available at the moment, could be integrated if there is demand.
Q. How can ATLAS analysis use notebooks - can you start a notebook locally on your laptop? Yes, but currently you need to install extra components. Reported to be not too hard though. Axel: there are limitations to handling plots in browsers though.

Q. For non CERN users? Docker compose being used to develop a solution.
Q. Wackamole service
Q. request for go

Chris Tunnell - Xenon1T

Numba as a mechanism for JITing tight loops in python
DAQ runs C++ to Mongo - trigger runs the jitted python for event selection

Antonio Alves - Hydra

Templated lib (header only) for parallel computations (OpenMP, CUDA, TBB)
Interest in integration with matrix element method analysis users
Invasive integration required for using?

Jim Pivarski - Femtocode

Reduction without intermediate steps. Data set available on RSS/SSD or other quick to access storage
Large cluster supporting many simultaneous analyses needed (like TB of memory?)

- shared query-based data access against a large in-memory cache (10TB)
- columnar flat tables is the approach used by all the buzzy tools out there. Efficient, extensible, can add new columns, new versions of particular columns from other files.
- operate on all events at once in GPU friendly way. Or for CPUs, roll up into loops for more efficient use of memory bandwidth.
- language in strings to apply a stronger type system than any language offers. If it compiles, it should work; if it fails, it's not your fault.

Dirk Duellmann - IT analytics

Difference with raw data processing (json formatting CPU bound) and binary formatted (pre-processed as data arrives at cluster). Since CERN using EOS machines for cluster, I/O capacity not easily measured

- Spark is a very good programming model for batch like processing on a Hadoop cluster
- standard map/reduce in java not convenient, takes a lot of code to do something simple, Spark preferable
- towards the end of analysis chain, doing low latency active analysis, total data volume is rather small, makes other tools attractive
    - have been using R with all the data in memory. Direct vector ops on table in memory.
    - Analysis app as pure functional code. Good for trivial multithreading.

Anton Poluektov - Analysis with tensorflow and coprocessors

Double vs single precision - minuit (at least for numerical derivative calculations) needs double
Remember tradeoff with people time vs computational savings
Minuit integration (or functionality) of interest
Edward(?) layers on top of Tensorflow - and looks like this (potentially)
Push HEP changes upstream or at least to github repo for sharing
MINUIT seems like a tool very generally useful outwith HEP

Missing pieces (Wednesday morning)

Session conveners: Peter Elmer, Graeme Stewart, Fons Rademakers

This session was intended to synthesize what has come before, return to interesting questions and discussion points accumulated during the workshop, address what has been missed in the coverage, what are the weak spots in the ecosystem, where are there opportunities for new work and new projects, what is the priority work to do, etc. It was somewhat compressed by overflow from the previous session. In the end it took the form of a productive discussion around slides prepared by Graeme Stewart postulating a “typical” HEP analysis in 2027.

- MFL - my favorite language (not invented yet)
- haven’t discussed so much the early stages of the analysis chain. Good data discovery tools and event metadata access needed. Solid interface with bookkeeping.
- didn’t come to consensus where the line is between managed production system analysis processing and individual analysis
- do agree that the analysis starting point is production system outputs
- solve easy access to web APIs through the CERN SSO. Web APIs a very convenient means of accessing things like data discovery and metadata tools. **Comment**: this is what the AARC project is supposed to solve (command line use of grid-authenticated web services), it is useful to take this into account at the start (and to remind the AARC people about this).
    - in general: make catalogs and metadata services easily accessible globally via convenient APIs to web services

- work with declarative languages. Description of the analysis, selection, NN training is not dependent on the (heterogeneous) back end machinery.
- could there be mileage in describing bookkeeping and other metadata in a more homogeneous experiment independent way
    - common effort for conditions, metadata in analysis?
    - TTree very effective for event data, can we have an analogue for non-event data
    - can we define an API with an experiment neutral abstraction, or is it impossible? (for conditions data it is happening, there’s a CWP group addressing it)
    - same infrastructure, different schema for different experiments. What is added value over generic DB?
        - for time dependent condition: interval of validity (IOV). But this is trivial in SQL.
    - bottom line conclusion: probably not (outside of conditions DB)?
- be sure you don’t build in an expectation of working with data as a local physical instance (but don’t exclude it). Support transparent streaming. Plan to accommodate WAN latency.
- think what underlying infrastructure declarative analysis implies and plan to have it in place. Build upwards, not downwards.
- we need to support both laptop mode and a remote cluster fronted by some notebook like interface.
    - 10 years from now the bandwidth to remote services will not be a problem? No agreement on this.
    - remote/local will need exploration but don’t build in assumption of either. Plan for both.

    - support caching locally well. Sync what you want from the remote services before getting on the plane.
- general theme: decouple what you want from how you get it.
- general approach: when we discover better options, better tools, add them. Some physicists will continue with the old, some will try the new. They'll vote with their feet.
- 'objectivity lesson': view ideas and technical components in the context of the larger system and plan to test them at scale in the larger, full systems (user, interface, software, data, hardware/facility). No technical component should be credible based only on its tit-for-tat merits in its specific niche in the system.
- Spark model is intermediate between 'small local' and 'big remote'
- Sharing infrastructure will be more and more important. Including sharing specialized facilities, services, GPU farms, TPU farms etc. Mechanisms to share them elastically.
- move aggressively in clean modular packaging, simple tool based installation ala brew fink npm etc.
    - as HSF has been establishing the infrastructure to do in its packaging working group
- haven't talked much here about the container revolution. Helps greatly with config issues and environment preservation. Do containers bring their own technical problems to address?
- have to think about software life cycle and sustainability. Ecosystem components become dependencies.
- how do we focus effort on our priorities? Need sources of effort (besides CERN).
- need a section on community building to sustain an ecosystem, what is the model. OSG has one, is that a model for us?