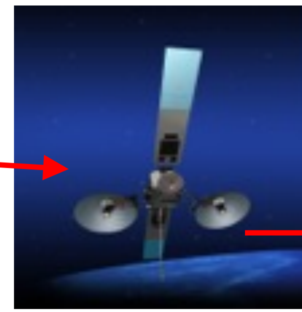
The ACQt data format

AMS-02 data flow

- AMS-02 records ~ 15.000.000.000 cosmic-ray events per year

TDRS
~ 10 MBit/s



NASA White Sands

Permanent storage @ CERN
~ 40 TB raw data / year

Data reconstruction @ JSC
~ 350 TB data / year

AMS-02 computing overview

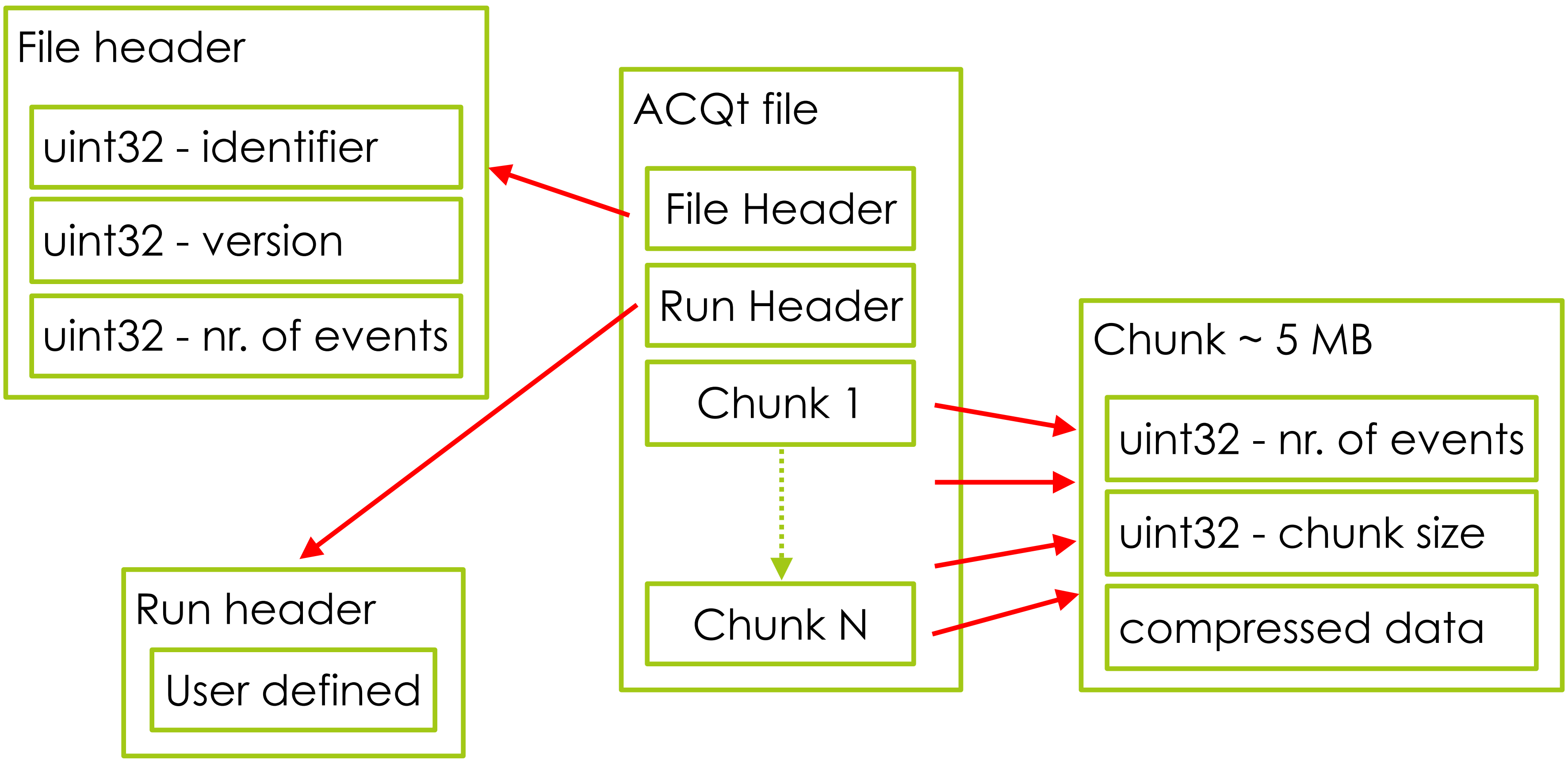
- Raw data stored at CERN contains the measurements of all „active channels“ in the detector
- The raw data is reconstructed using the AMS offline software and high-level reconstruction algorithms are applied (track finding, etc.)
- The reconstructed data is stored using the standard file format in high-energy physics: ROOT trees.
- 20 minutes AMS data == one run, spanning at least one physical file (~ 8 - 12 GB)
- Each run contains N events, where an event contains all necessary information for the physics analysis

Replacing ROOT trees?

- Design of ROOT I/O dates back to 1990s, without Parallel I/O, large scale clusters in mind
- Analysis with OpenMP / MPI cumbersome and inefficient. Not officially supported. No MPI parallel I/O possible
- Wishlist for a new high-energy physics data format
  - Inherently scalable, from laptop to large-scale cluster
  - Fine-grained control over data layout on disk
  - Constant memory usage while processing file
  - Both Serial I/O and Parallel I/O should be possible
  - Backwards compatible (just like ROOT trees)

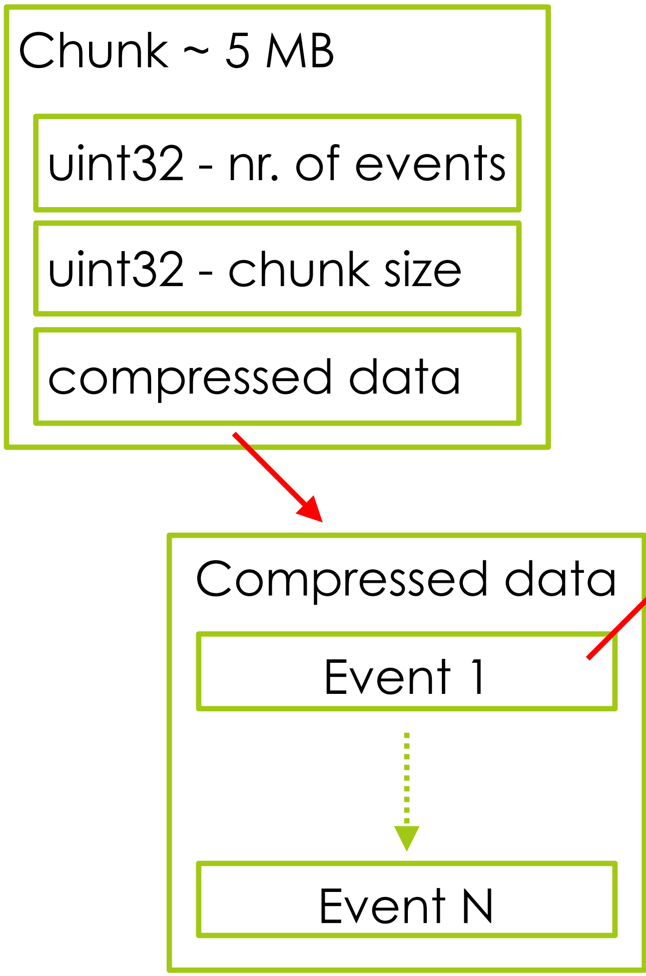
ACQt files

- ACQt files, developed since 2012 in Aachen to overcome the limitations of the ROOT trees used for the AMS physics analysis



ACQt chunks

- Each ACQt chunk contains N events as zlib compressed data



Event.idl:

```
[Description]
Namespace       = AC
Class           = Event
Members         = \
  AC::EventHeader    fEventHeader
  ParticlesVector    fParticles
  AC::Trigger        fTrigger
 …
  AC::TOF            fTOF
  AC::ACC            fACC
  AC::TRD            fTRD
 …
```

ACQt IDL files

- All classes contained in the `AC::Event` are described with their own IDL file
- A helper utility generates C++ code covering the class member initialization, declaration as well as serialization/deserialization
- No need to deal with byte-order, alignment, padding manually. No need to remember boiler-plate code to serialize/deserialize data. ACQt auto-generates all necessary C++ code.
- Easy to maintain and understand by non C++ professionals. All it takes to change the data format is:
  - Alter IDL file
  - Add/remove accessor from C++ class
  - Recompile

ACQt versioning support

- Explicit version control for each variable in each class

ECALShower.idl:

```
[Description]
Namespace      = AC
Class          = ECALShower
Members        = \
 UInt_t                 fStatus
 UShort_t               fNumberOfHits
 [590-Float_t           fDepositedEnergy]
 [594+Float_t           fEnergyAt1CMRatio]
 Float_t                fEnergyAt3CMRatio
 [591_593Float_t        fOldEnergyScale]
 ...
```

Removed in ACQt 5.9.0 (→ fDepositedEnergy)

Added in ACQt 5.9.4 (→ fEnergyAt1CMRatio)

Added in ACQt 5.9.1 Removed in ACQt 5.9.3 (→ fOldEnergyScale)

Fixed-precision data support

- A special type **`AC::Round`** indicates fixed precision data

```
AC::Round fVar(LowBound | HighBound | Transformer | Bytes)
```

- Examples: (from **`AC::TOFBeta`**)

  Mapping numeric range to 16 bit:

```
AC::Round fInverseBetaUncertainty(-10.0 | 10.0 | 0 | 2)
```

  Mapping numeric range with transformation to 16 bit:

```
AC::Round fBeta( -3.0 |  3.0 | Tanh3_Transformer | 2)
AC::Round fChi2(1e-10 | 1e10 |   Log_Transformer | 2)
```

- Philosophy: Every bit counts.
  Strip down all variables to their intrinsic precision.
  Don’t waste disk space by serializing numerical noise, e.g. using a 64 bit double to store a temperature, which has steps in 16 bit

Enforcing constant memory usage

- All ACQt classes use custom vector-like data-structures, almost providing source compatibility with std::vector from STL
- Usage: **`WTF::Vector<AnyType, InlineCapacity>`**

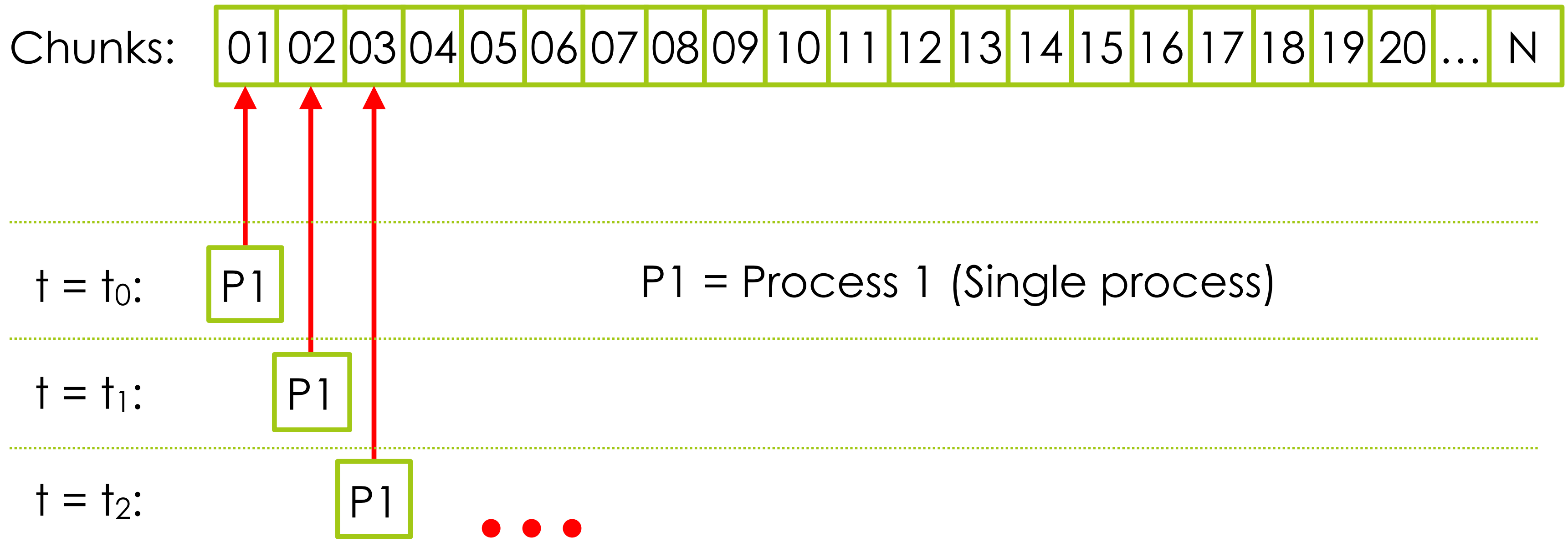
  e.g. **`AC::ECAL`** contains
  `typedef WTF::Vector<AC::ECALShower, 10> ShowersVector;`
- The underlying buffer of the vector is allocated on the stack, reserving **`InlineCapacity`** objects of type **`AnyType`**
- When exceeding the inline capacity, memory is dynamically allocated on the heap as fallback (slow case)
- Optimal usage never exceeds the inline capacity
- Built-in benchmarking capabilities to detect exceeds

Performing AMS-02 analysis

- Equipped with the ACQt files and a software framework we can now perform analysis fully parallelized.
- From each AMS ROOT file (8 - 12 GB) reduced ACQt files are written containing only a small fraction of the original data (~ 400 MB)
- Any number of ACQt files can be merged into large Multi-ACQt files, spanning up to several TB per file if needed.
- The Multi-ACQt files are intended for the actual physics analysis, which typically requires lots of runs over the same data set
- Multi-ACQt files combine the benefits of a dense data format with the necessities for large-scale parallel I/O (large files!)

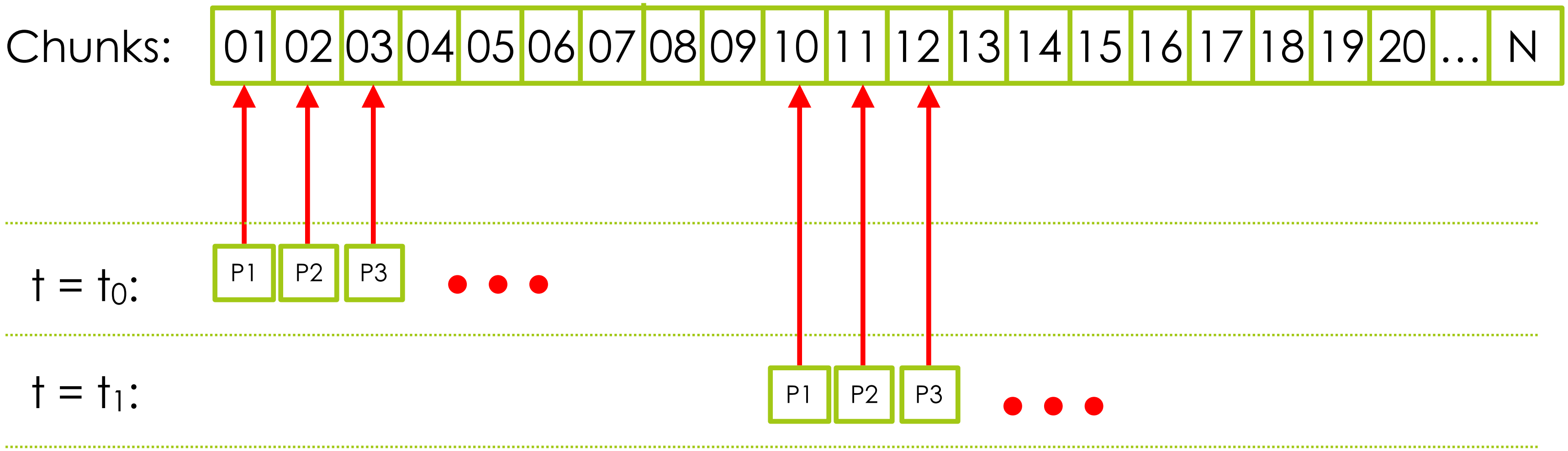
Serial I/O mode

- One process analyzes one chunk after the other, sequentially looping over the events in each chunk
- Important analysis mode if time-order of event matters e.g. calibration of an AMS-02 sub-detector, etc.



Parallel I/O mode

- Launch N MPI processes analyzing N chunks in parallel
- Benefit from MPI Parallel I/O - coordinates access to the physical file with the underlying file system
- Inherently scalable from two processes up to thousands
- Only limited by the size of the ACQt files (contained chunks)



Summary

- A dense file format was designed specifically to cope with the difficulties in data processing and scalable solutions like MPI Parallel I/O.
- We can use the JUROPA cluster very efficiently now, compared to our initial serial ROOT tree processing
- Recently developed Multi-ACQt files solve the issue of having lots of small files (< 1 GB), resulting in performance depredations on both Lustre and GPFS
- Hybrid jobs (OpenMP + MPI) to allow to exploit multicore environments for the actual data analysis, besides the I/O, are under development.

Thanks for your attention!