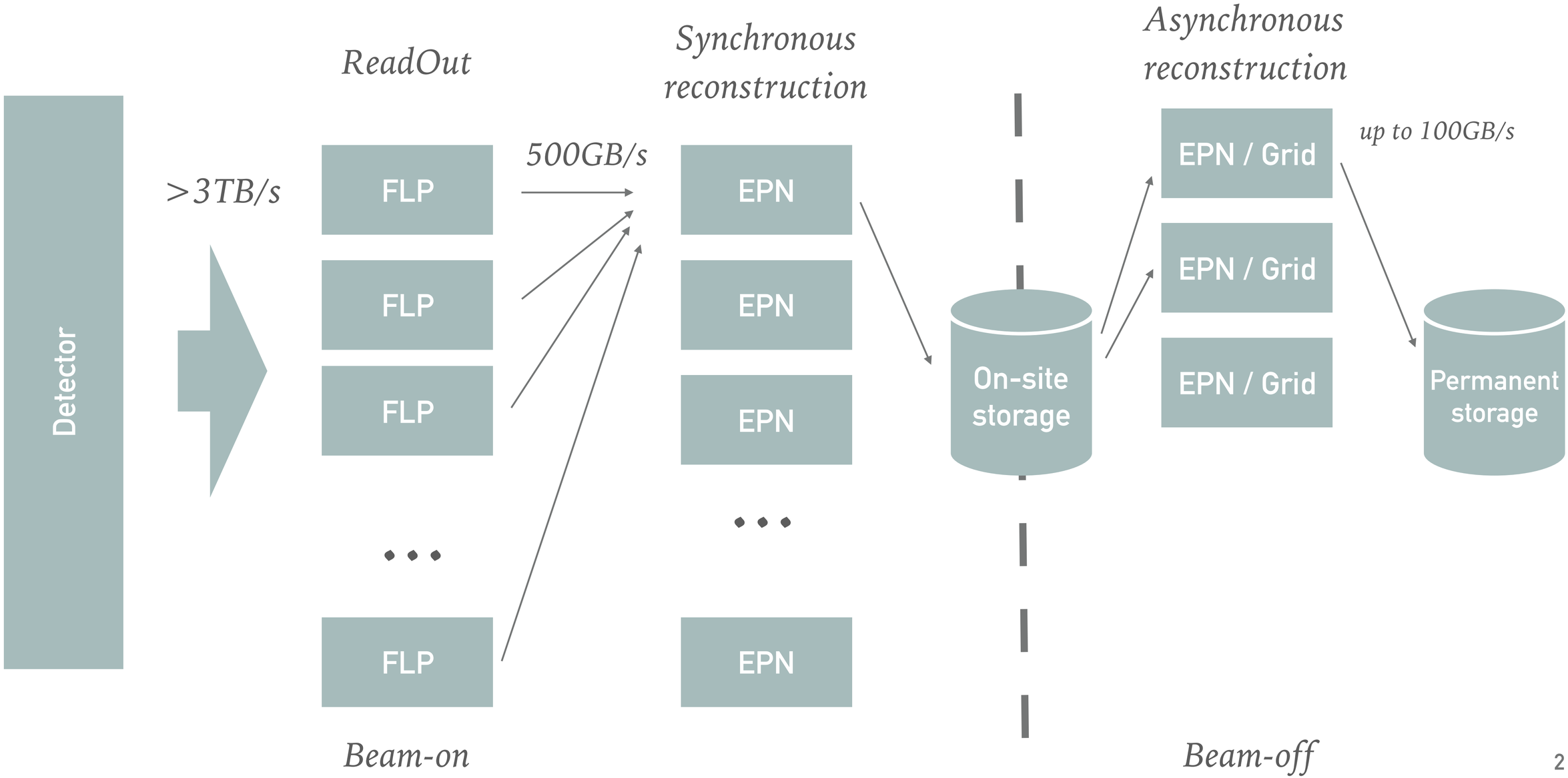
# ALICE I/O IN RUN 3

*Giulio Eulisse*

# ALICE IN RUN 3: O2



# ALICE IN RUN 3: O2

This computing architecture will be implemented in terms of message passing entities called "devices".

Key ingredients:

- **Standalone processes** for deployment flexibility.
- **Message passing** as a parallelism paradigm
- **Shared memory** backend for reduced memory usage and improved performance.
- **Simplified, zero-copy** data model for performance.
- **Dataflow framework** built on top to simplify life of the end user.

# TIMEFRAME

*Data quantum will not be the event, but the "Timeframe".*

- *~23ms worth of data taking in continuous readout. Equivalent to 1000 collisions. Atomic unit.*
- *~10GB after readout. Vast majority in TPC clusters.*
- *Compressed to ~2GB after synchronous reconstruction, mostly thanks to track-model-compression.*
- *Flat format with tables and indices. Optimised for TPC reconstruction on the GPU. Direct result of the synchronous reconstruction and directly usable (no-deserialisation) for asynchronous reconstruction.*

# O2 DATA MODEL

*A timeframe is a collection of* `(header, payload)` *pairs. Headers defines the type of data. Different header types can be stacked to store extra metadata (mimicking a Type hierarchy structure). Both header and payloads should be usable in a* ***message passing*** *environment.*


*Different payloads might have* ***different serialisation strategies****. E.g.:*

- *TPC clusters / tracks:* ***flat POD data with relative indexes****, well suitable for GPU processing.*
- *QA histograms:* ***serialised ROOT*** *histograms.*
- *AOD:* ***some columnar data format****. Multiple solutions being investigated.*

# ANALYSIS MODEL: RUN 2

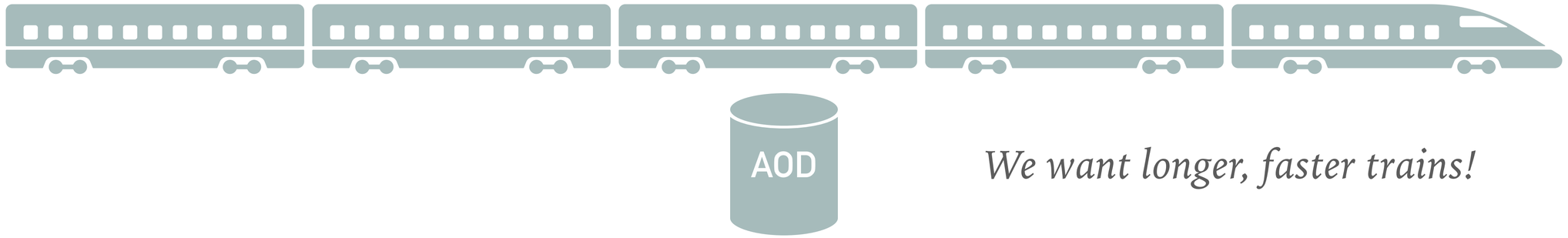
In order to offset the costs of reading data, ALICE has as strong tradition of organised analysis (i.e. trains):

- Users provide "wagons", organised in "trains". Trains run on the Grid.
- Data is read only once per train, wagons get applied to it.
- Data is kept in a generic C++ object store, backed by ROOT, as you know.
- Slow sites / site issues is what dominates performance.

# ANALYSIS MODEL: RUN 3

**Solid foundations:** *the idea of organised analysis will remain. Improve on the implementation.*

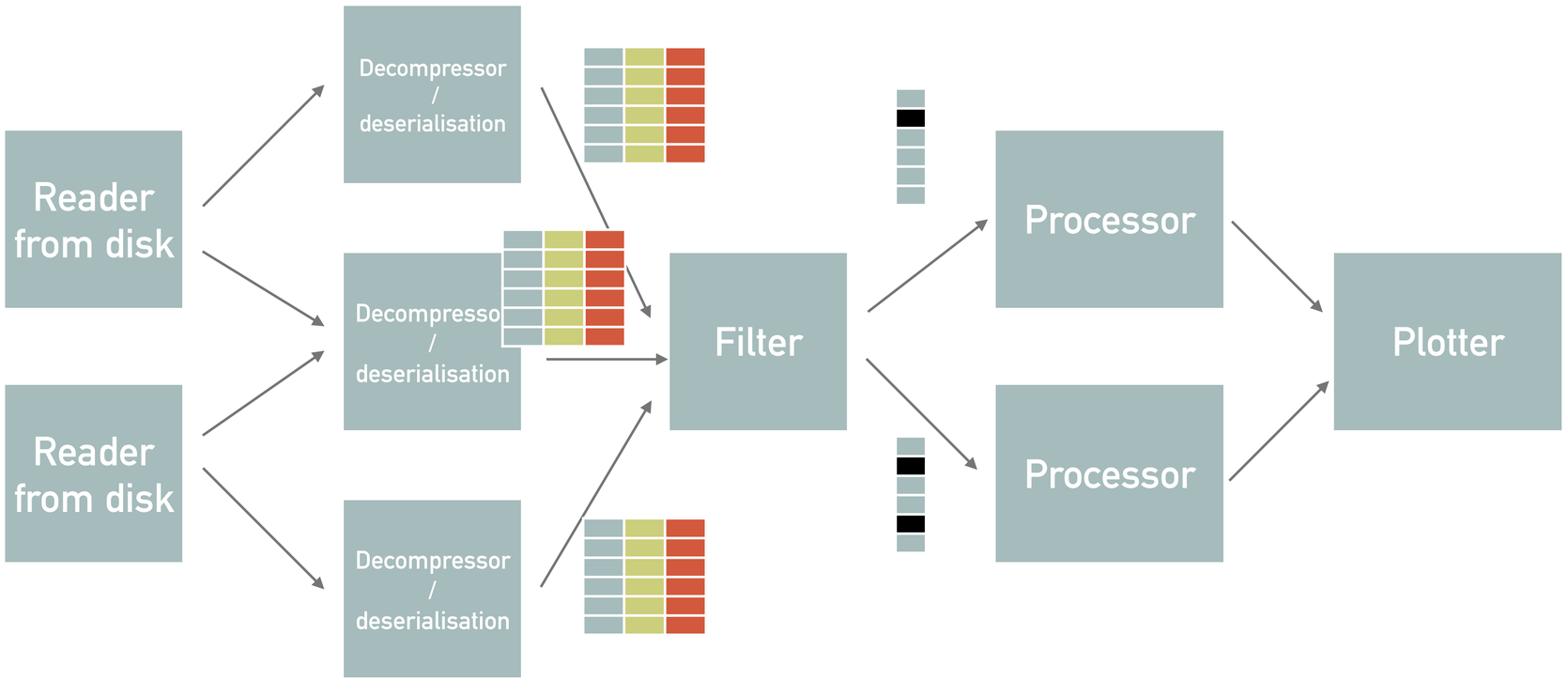
- ***x100** more collisions compared to present setup*
- *Do analysis on fewer, highly performant, **Analysis Facilities** (e.g. @GSI).*
- ***Streamline data model**, reducing generality and features set to improved speed.*
- *Explore different **compression strategies** (e.g. LZ4, Zstd, custom compression code)*
- ***Recompute** quantities on the fly rather than storing them. CPU cycles are cheap.*
- *Goal is to have each Analysis Facility go through 5PB of AODs every 12 hours (~100GB/s).*




*We want longer, faster trains!*

# THE FAIRMQ BIT: A POSSIBLE TOPOLOGY

*Current bottleneck*




*… can be scaled out by adding extra devices (assuming multiple cores)…*

# REQUIREMENTS FOR THE AOD FORMAT

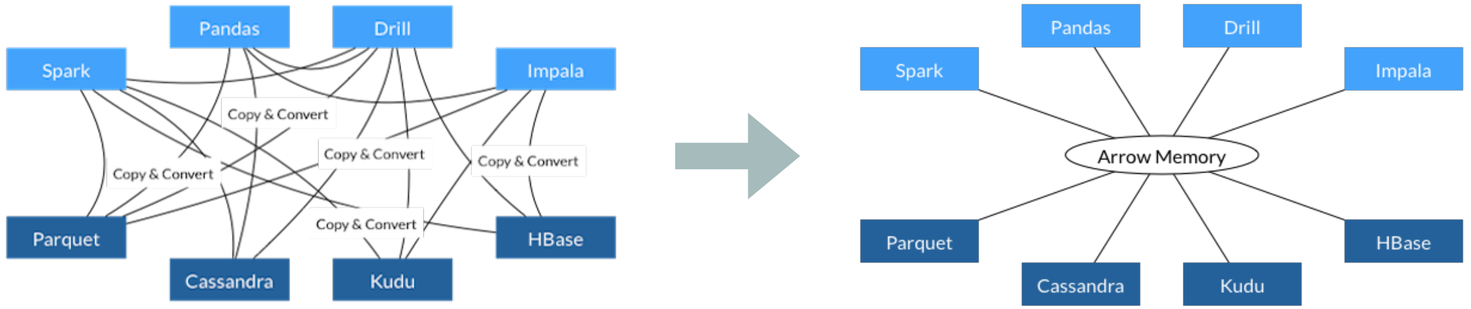
AOD's data format will have to play well with AliceO2 message passing, shared memory backed, distributed nature.

- **Zero-{Copy,Serialisation,Adjustments}**: *we want to be able to reuse data between processes.*
- **Growable**: *ability to extend columns on the fly.*
- **Prunable:** *ability to drop columns on the fly.*
- **Skimmable:** *ability to select only certain rows.*

*Strategy: we are willing to lose some degree of generality for performance.*

# A POSSIBLE SOLUTION FOR AOD: APACHE ARROW

*"Cross-language development platform for in-memory columnar data."*




**Well established.** *Top-Level Apache project backed by key developers of a number of opensource projects:* ***Calcite****, Cassandra, Drill, Hadoop, HBase, Ibis, Impala, Kudu,* ***Pandas****,* ***Parquet****, Phoenix,* ***Spark****, and Storm.*

**Very active.** *119 contributors, https://github.com/apache/arrow*

**O2 design friendly.** *message passing / shared memory friendly. Support for zero-copy slicing, filtering.*

# APACHE ARROW: A FEW TECHNICAL DETAILS

***In memory column oriented storage.*** *Full description https://arrow.apache.org/docs/memory_layout.html. Data is organized in Tables. Tables are made of Columns. Columns are (<metadata>, Array). An Array is backed by one or multiple Buffers.*

***Nullable fields.*** *An extra bitmap can optionally be provided to tell if a given slot in a column is occupied.*

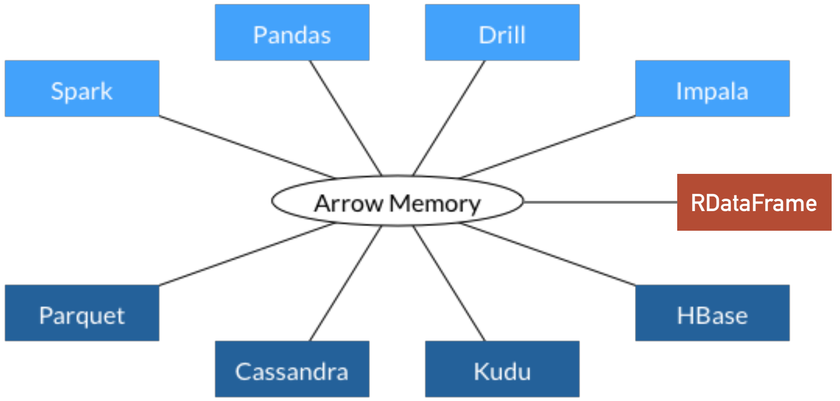
***Nested types.*** *Usual basic types (int, float, ..). It's also possible (via the usual record shredding presented in Google's Dremel paper) to support nested types. E.g. a String is a List<Char>.*

***No (generic) polymorphism.*** *The type in an array can be nested, but there is no polymorphisms available (can be faked via nullable fields & unions).*

***Suitable for ALICE analysis needs?***

# APACHE ARROW: INTEGRATION WITH ROOT

*The main concern here is of course "how do I use this from ROOT"?*




**RDataFrame:** *see talk by Danilo. Arrow fits naturally as a "RDataSource".*

**Show me the code!** *https://github.com/root-project/root/pull/1712*

**Bonus:** *ROOT gets seamless integration with many OpenSource projects which you can mention to impress your friends and that make your CV look good to head-hunters.*

# APACHE ARROW: HOPES

**Query optimisation:** *by writing an analysis in form of a filter - emit - reduce query, the system has the ability to optimise it's execution. "All vertices with Vertex.z > 1" should run on "All vertices with Vertex.z > 0".*

**Remove / minimise serialisation, scale decompression.**

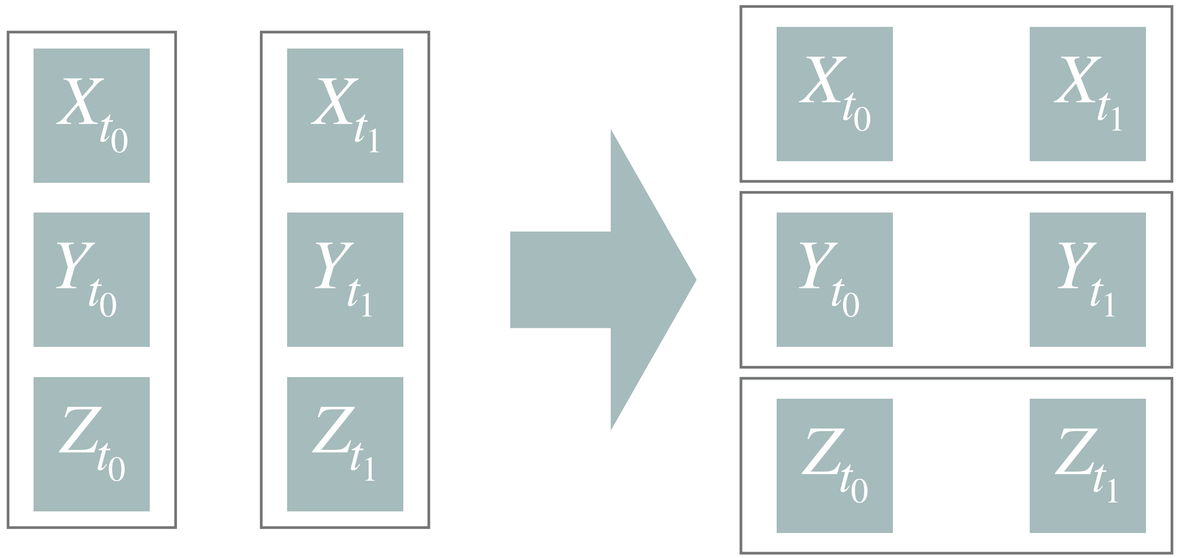
**Allow vectorised processing.** *Not a big deal at the moment (bottleneck is deserialisation). Might become relevant in the future?*

**Homogeneity with the rest of the system:** *analysis wagons become yet another device / data processor in the O2 framework.*

**Integration with other system:** *using established OpenSource components add complications, but potentially simplifies long term sustainability and integration with other tools.*

# MORE IDEAS: CONDITIONS STORAGE

- Currently using ROOT files served from CVMFS.
- Given the fact that condition objects evolve over time, it is probably worth investigating compression of different **TKey**s across different files.




- This is what git does if you substitute TKey -> blob, TFile -> commit. Why not storing TKeys as git blobs (or mimick that internally in CVMFS)?