Meta-Data experience in a modern collaboration

Solenoidal **T**racker **A**t **R**HIC (STAR)

Jérôme LAURET, S&C Leader

Outline

My mission: present “a bit of everything about Meta-Data” used in STAR … No room for technology details, …

- Defining Meta-Data & usage in STAR
  - General definition and classification
  - Structural Meta-Data
  - Bookkeeping and human level info
    - General run-time information
    - Calibration information
    - FileCatalog
    - Tags
- Last thoughts & remarks
- Conclusion

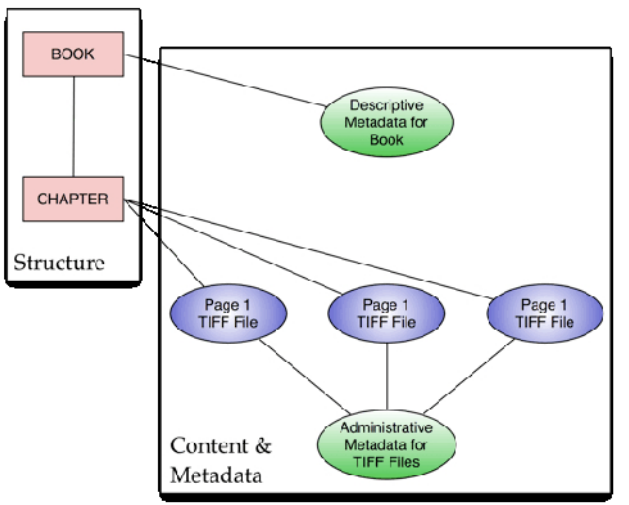
Defining Meta-Data & usage in STAR

General definition and classification

- Meta-Data: anything describing data
  - Funny thing is: index and structure of a DB is Meta-Data while the content can be its data but its usage be considered as Meta-Data by a higher level component …
  - Nearly all data could be Meta-Data for a higher level component
  - Already, the definition make your head spin …

- Classifications (many "theories", standards, …) & technologies
  - No intent to lose objectives: want to (a) use the damned thing at the end (b) be useful to select or supplement information in the data stream
  - Any other definition, fine with me (let us see if it is practical and works)
  - Our generic classification
    - Structural (object description)
    - Bookkeeping (human level – operational and guiding)

Structural Meta-Data



Structural Meta-Data

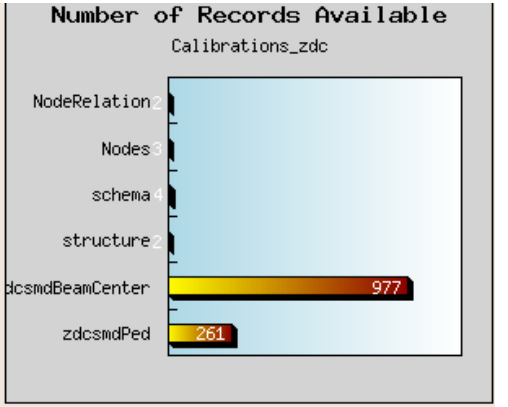
- Object description i.e. a description on how the information/objects are organized
- STAR workflow path to physics: nearly all data have structural Meta-Data
  - Exception: DAQ file do not have embedded schema evolution (old style “bank navigation” and conditional logic)
  - Otherwise STAR has taken a pragmatic approach from the start
    - **Schema or version evolution all the way** (data stream, database access, configuration access, …)
    - **An API layer handling the evolution** (in house or external, hidden or explicit)
- Several levels
  - Simple (text) Meta-Data: LoadBalancing, service (connection) information, … XML+XSD
  - Data streams: self-described structure (“Table” based or ROOT files) + handling of version or schema evolution
    - ROOT handles schema evolution for us
    - Table reading are version evolving
  - Database content: all DB based tables (calibrations) designed with version evolution in mind.
    - API layer handles reading and writing content
    - Object representation at user level – IO and storage behind the scene

Only recipe for productivity: users must remain agnostic …
Magic, incantations and structure handling happening behind the scene …

Structural Meta-Data

Database structure : Calibrations_zdc

Table Name	Last entryTime	Index Field(s)	Records
NodeRelation	2004-07-22 15:48:04	ParentID NodeID BranchID ConfigID	2
Nodes	2004-07-22 15:43:09	name versionKey	3
schema	2004-07-22 15:36:04	name ID	4
structure	2004-07-22 15:36:04	name ID	2
zdcsmdBeamCenter	2004-07-23 14:30:02	nodeID elementID beginTime flavor deactive	977
zdcsmdPed	2004-07-23 01:43:55	nodeID elementID beginTime flavor deactive	261

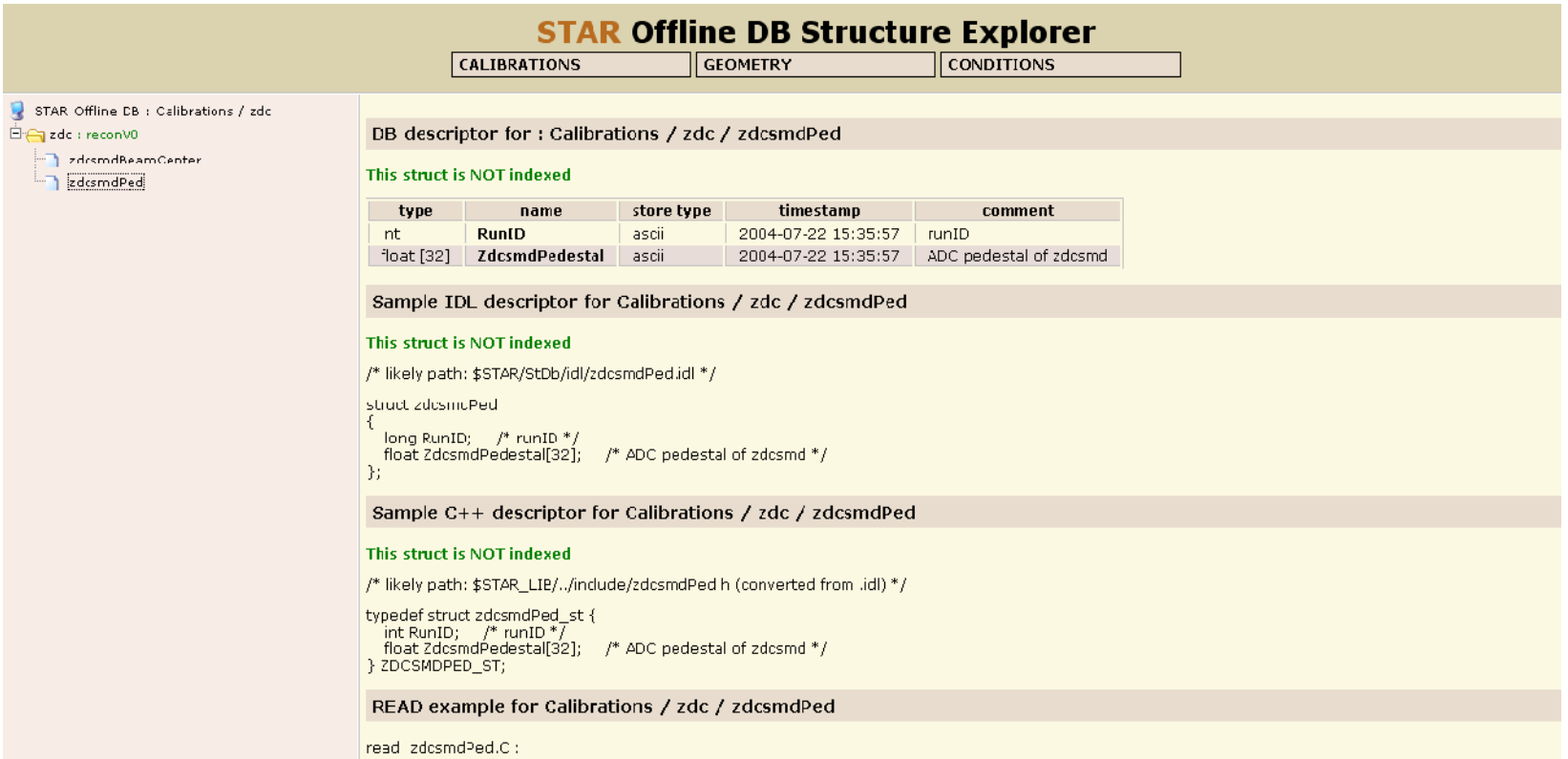


In this example, table schema has changed … fields will be used by the API to handle version evolution

- structure holds the names of all Objects
- schema holds the names of all elements associated with an Object, their order, and when fields appeared (at which schema version)

IT WORKS!

Structural Meta-Data



If users have doubts, a “structure explorer” will (a) decode the object names, fields, types and (b) generate code for reading and writing the object to the “DB”

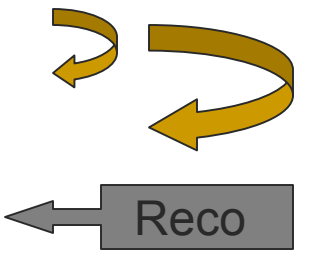
Bookkeeping Meta-Data

(operational or guiding)

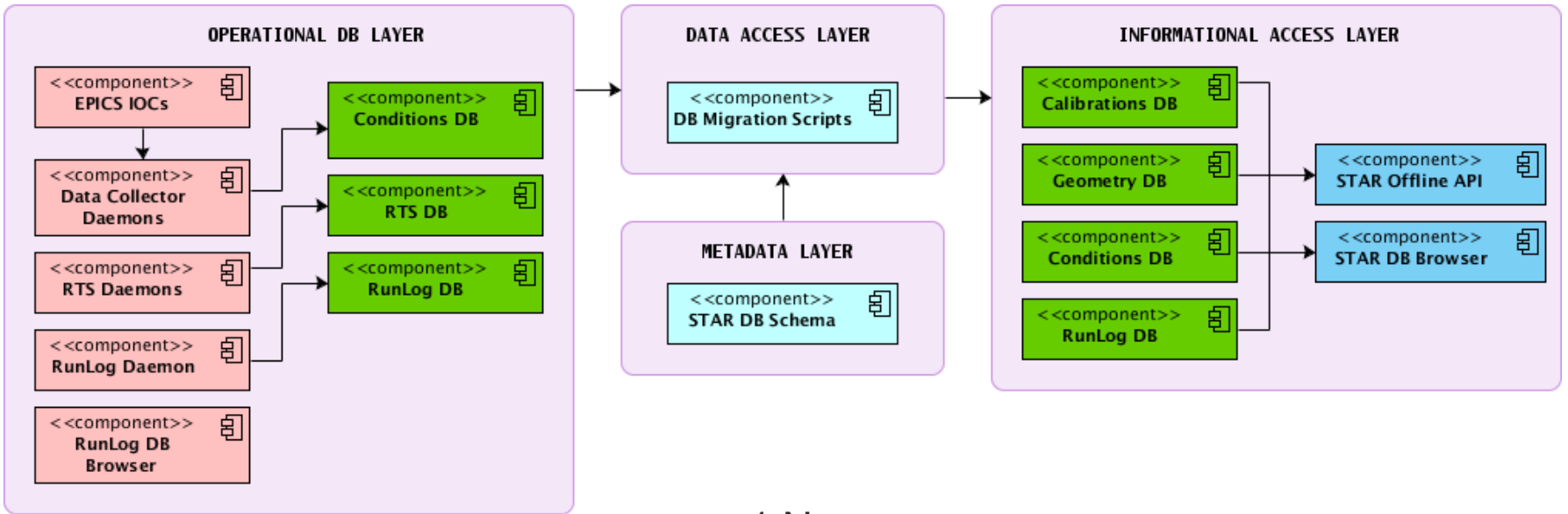
Bookkeeping and human level info

- The traps:
  - Too little information and data cannot be reconstructed, datasets cannot be located, analysis lacks performance (lack of selectors)
  - Too much information and M-D becomes as large as data

- Several kind in STAR
  - **General run-time information** **(operational)**
  - **Calibration data** **(operational)**
  - **FileCatalog data** **(guiding)**
  - **Tags** **(guiding)**
  - **Others: ShiftInfo, …**



General flow …



1:N

Reduction, aggregation, synthesis, transformation

Single timestamp

Multiple timestamps + …
Unified API

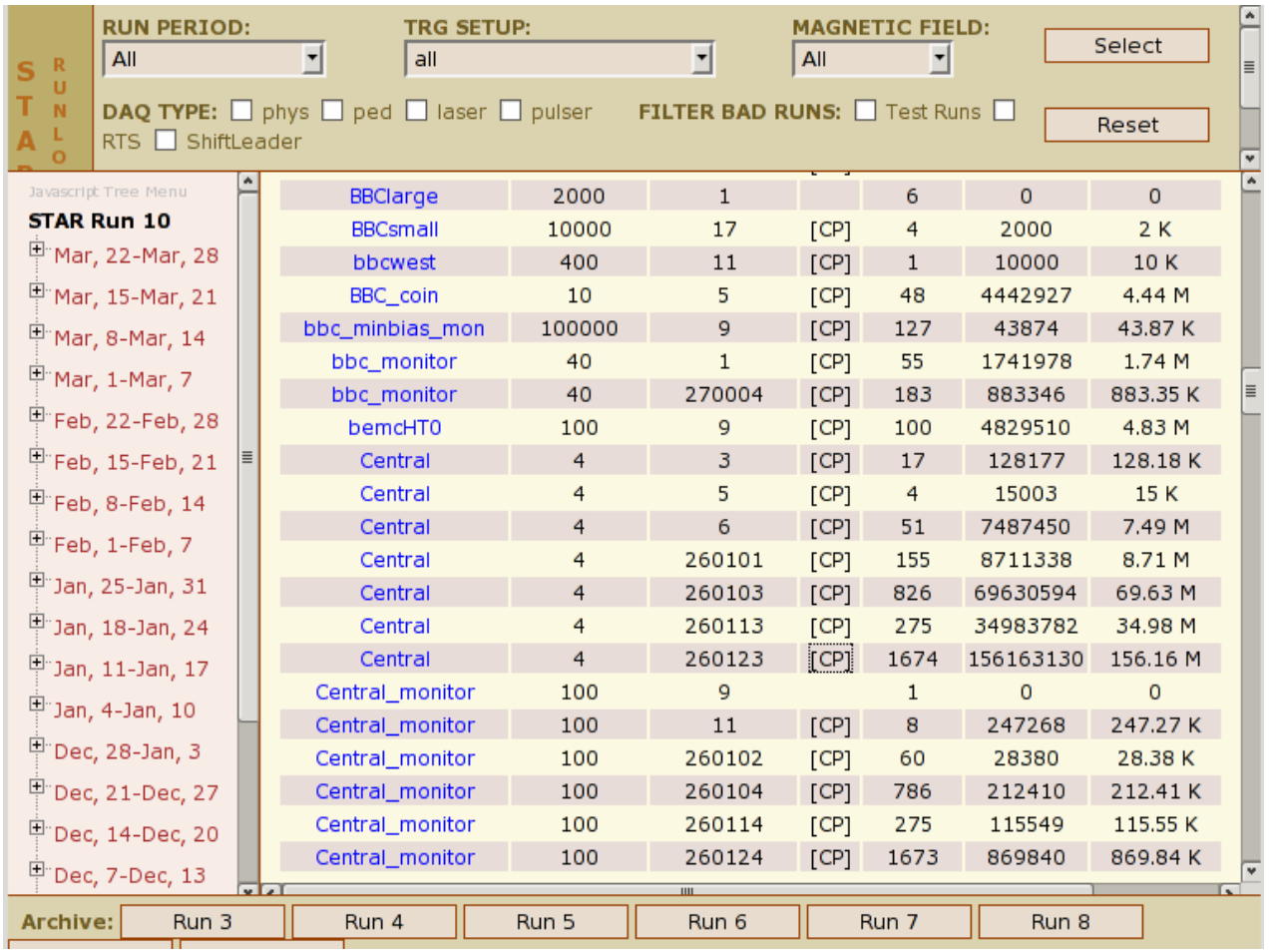
Bookkeeping and human level info

(General run-time information)

- **General run-time information**: M-D accompanying real physics data taken during the Run: operator actions, internal states of various DAQ/RTS/SC components, detector states (health, parameters, throughput etc ...)
- Granularity: daqEventTag (event level), daqFileTag (file metadata), daqRunTag (run metadata), daqtrgSumCnt (triggers metadata per year as many configurations)
- Content examples and details:
  - **Single timestamp (when the information was acquired/recorded) – no change with time**
  - **daqEventTag** database (RTS) : run ID, file sequence, event number, token, size, time, trigger word, trigger command, DAQ related command, detector bits (on/off), l3 flags, additional trigger bits, dsm bits;
  - **daqFileTag** database (RTS) : run ID, begin/end event, number of events, file sequence, file, storage type (hpss/local);
  - **daqRunTag** database (RTS) : run ID, start/stop time, run type, total number of events
  - **daqsumTrgCnts** database (RTS) : run ID, trigger ID, number of events, event builder, average size;
- Size & Problems: EventTag
  - Grew to 100 GB per year by 2006, started to reach 100 GB per few weeks
  - Event based information dropped as out of balance and rarely consulted

Bookkeeping and human level info

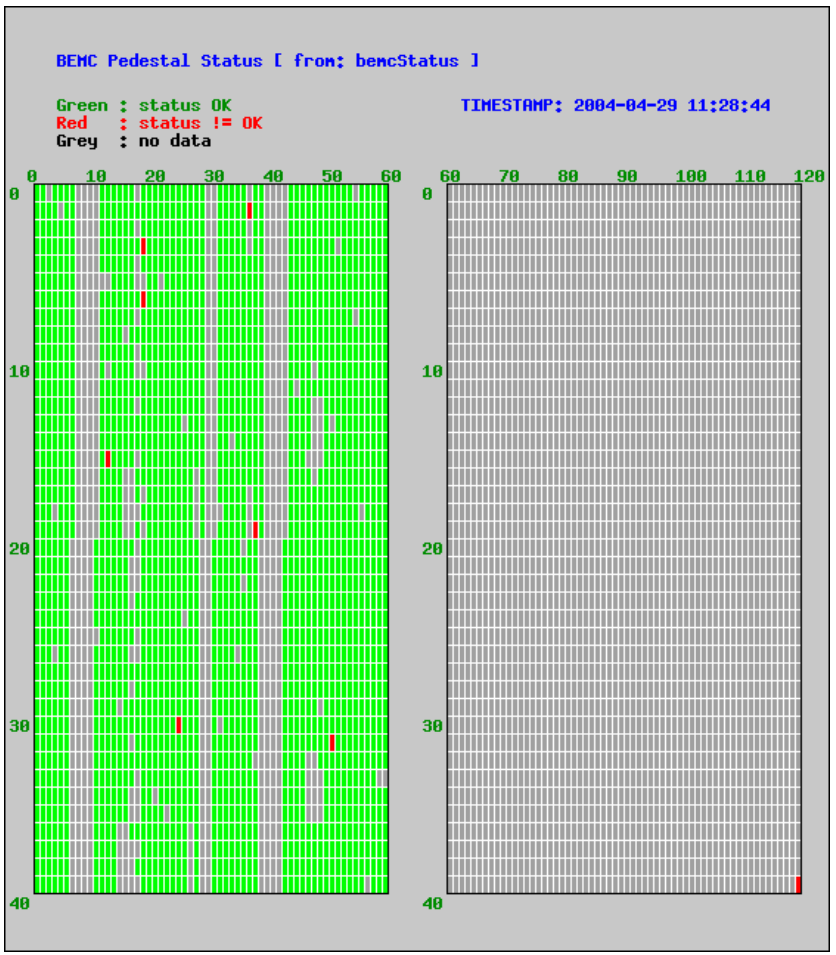
(General run-time information)



Typical bookkeeping # events per trigger word

Bookkeeping and human level info

(General run-time information)

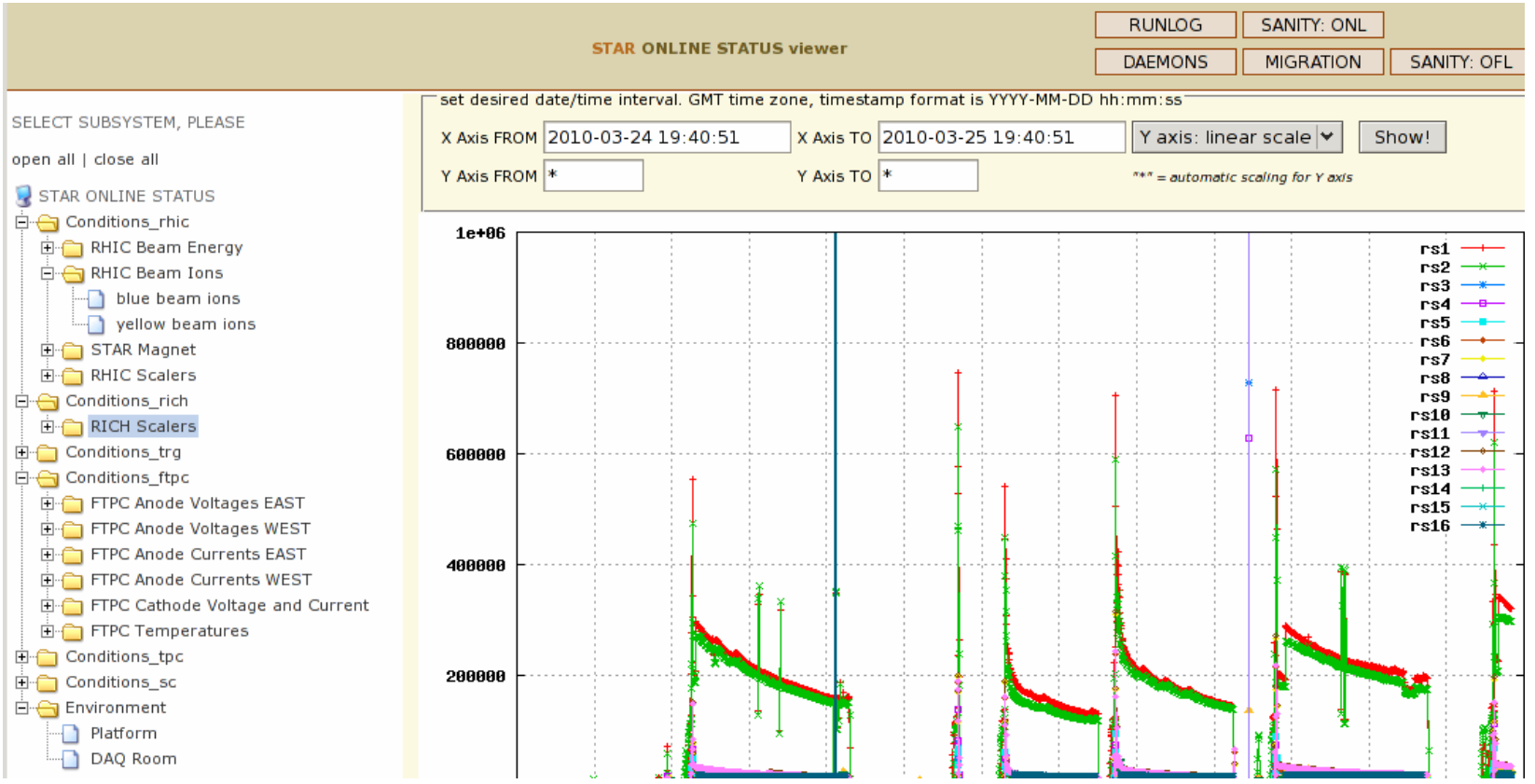


Typical monitoring
Calorimeter status (1/0)

Bookkeeping and human level info
(calibration information)

- **Calibration data**: M-D applied to physics data taken during the Run.
- Granularity: Detector *Conditions* data is collected every 1-5 minutes, resulting *Calibration* (derived) would follow this timeline, *Geometry* seldom granularity (a few)
- Content examples and details:
  - A two timestamps layer allowing historical preservation of all entries
    - **beginTime** defines a validity range for the entry with respects of a collision event time. Given an event time, the first begingTime < eventTime will be considered
    - **entryTime** allows for refining calibrations. Given a moment in the year at which production is made, only values entered in the DB at times < entryTime will be considered
    - RULE OF THUMB: **ONLY insert, NO UPDATE for older values**
    - Consequence: **Given data production FULLY reproducible at all times**
  - A "flavor" dimension – allows separation by "realm" such as simulation or real-data, … or test. API fully aware of flavors
  - At higher logic, **hierarchical**
    - TPC → DriftVelocity → values (object). Object would contain east and west values for two methods.
    - API ask for the TPCDriftVelocity "object"
- Sizes & problems:
  - Granularity results in ~15 GB raw data – (reduction) → 0.5 - 1 GB processed data per Run(!), ~20 GB in total for Runs 1-10 (offline) with one outliner
    - Problem in Run 5 & 6: size for the SSD alone is 10 GB for both years – ill-defined table
  - Burden not on user end but on DB admin to think of his storage model (split object, row repetition suppression, …)

Bookkeeping and human level info
(calibration information)



Bookkeeping and human level info
(calibration information)

entryTime	nodeID	elementID	beginTime	flavor	schemaID	deactive	barometricPressure
2010-06-10 11:01:06	41	0	2010-06-10 10:35:00	ofl	1	0	1007.12945557
2010-06-10 10:41:07	41	0	2010-06-10 10:25:00	ofl	1	0	1007.10498047
2010-06-10 10:41:07	41	0	2010-06-10 10:05:00	ofl	1	0	1007.11749268
2010-06-10 10:41:07	41	0	2010-06-10 09:45:00	ofl	1	0	1006.88397217
2010-06-10 10:01:07	41						
2010-06-10 09:41:07	41						
2010-06-10 09:41:07	41						
2010-06-10 09:41:07	41						
2010-06-10 09:01:07	41						
2010-06-10 08:41:07	41						
2010-06-10 08:41:07	41						
2010-06-10 08:41:07	41						
2010-06-10 08:01:07	41						
2010-06-10 07:41:06	41						
2010-06-10 07:41:06	41						

Field Name	Type	Flags
dataID	int (11)	not_null multiple_key auto_increment
entryTime	timestamp (19)	not_null multiple_key unsigned zerofill binary timestamp
nodeID	int (11)	not_null primary_key
elementID	int (6)	not_null primary_key
beginTime	datetime (19)	not_null primary_key binary
flavor	string (32)	not_null primary_key
schemaID	int (11)	not_null
deactive	int (10)	not_null primary_key unsigned
barometricPressure	real (16)	
inputTPCGasPressure	real (16)	
nitrogenPressure	real (16)	
gasPressureDiff	real (16)	
inputGasTemperature	real (16)	
outputGasTemperature	real (16)	
flowRateArgon1	real (16)	
flowRateArgon2	real (16)	
flowRateMethane	real (16)	
percentMethaneIn	real (16)	
ppmOxygenIn	real (16)	
flowRateExhaust	real (16)	
percentMethaneOut	real (16)	
ppmWaterOut	real (16)	
ppmOxygenOut	real (16)	
flowRateRecirculation	real (16)	

Calibration as M-D tpcGas example table

In a DB sense, this part handles the table M-D

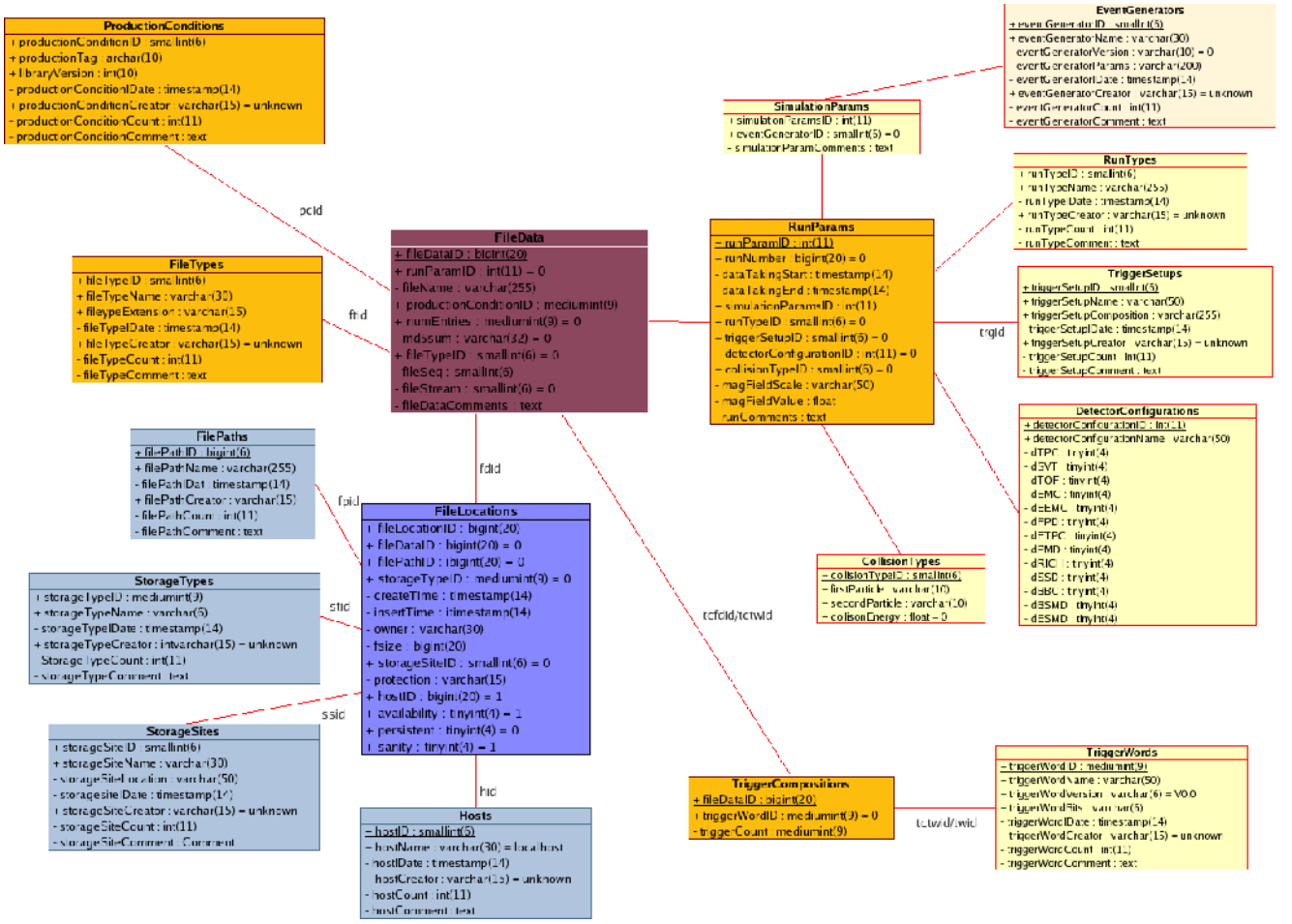
This would be return as an object

Bookkeeping and human level info
(File catalog)

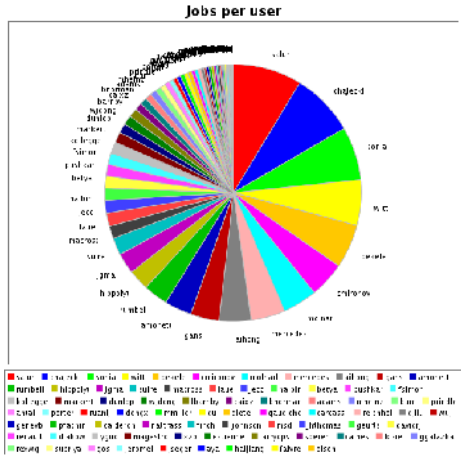
- **File Catalog:** A lose term including "some" of the real data M-D, file M-D, replica information. Usage is to define datasets along: global M-D → Files or Datasets → accessible replicas – used for placing, locating, accessing datasets & ensure consistency (MD5), minimal quanta replication, etc … (data management)
- Granularity: contains all files produced by STAR (online, offline, simulated or real data).
- Content examples and details:
  - Syntax is based on "give me this info considering those constraints"
    - Give me all files available at BNL for year 10 data, production P10ih and the sample passing "AuAu200 production"
    - Give me all possible trigger setup used in the year2010 run
    - Give me all event generator and version ever used in STAR as well as the total number of events generated by each of them
  - Nearly all user analysis start with a query to STAR's FileCatalog
  - API shield users entirely from field association – context based
    - FC->set_context("name~physics||laser","trigger=AuAu200_minbias");
  - FileCatalog contains technical M-D in "dictionaries" (there are standardized tables of modest size 100th) and more complex relational tables (for example, list of triggers), queries are cached + FileCatalog has two main/core tables (File and Replica a.k.a. FileData and FileLocations)
    - Values in dictionaries are set at Tier0 but available everywhere
    - Each "site" responsible for updating its replica information (multi-master approach)
- Sizes and problems:
  - 18 M files, 40 M replicas, 11 GB
  - Problems: none fundamental so far
    - Selections on partial string slow [hiding sometimes make user think the impossible is possible from the start]
    - User tend to "wish" for event level M-D in it …

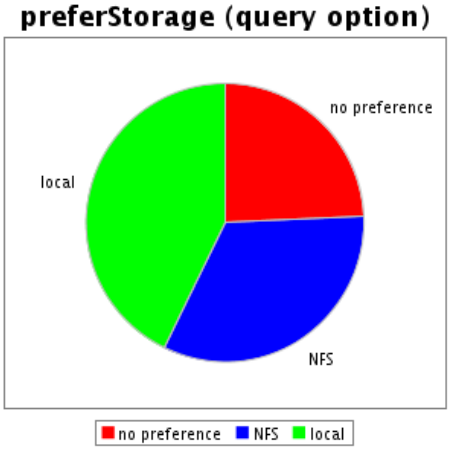
Bookkeeping and human level info

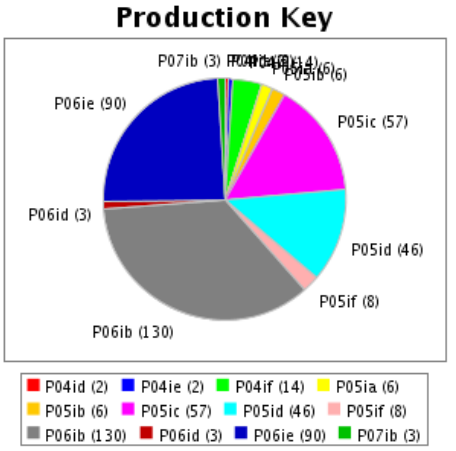
(File catalog)

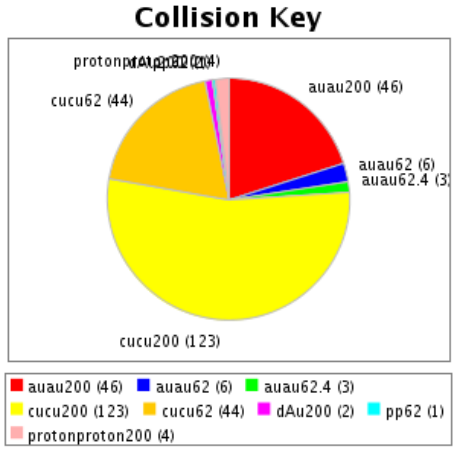


Bookkeeping and human level info
(File catalog)





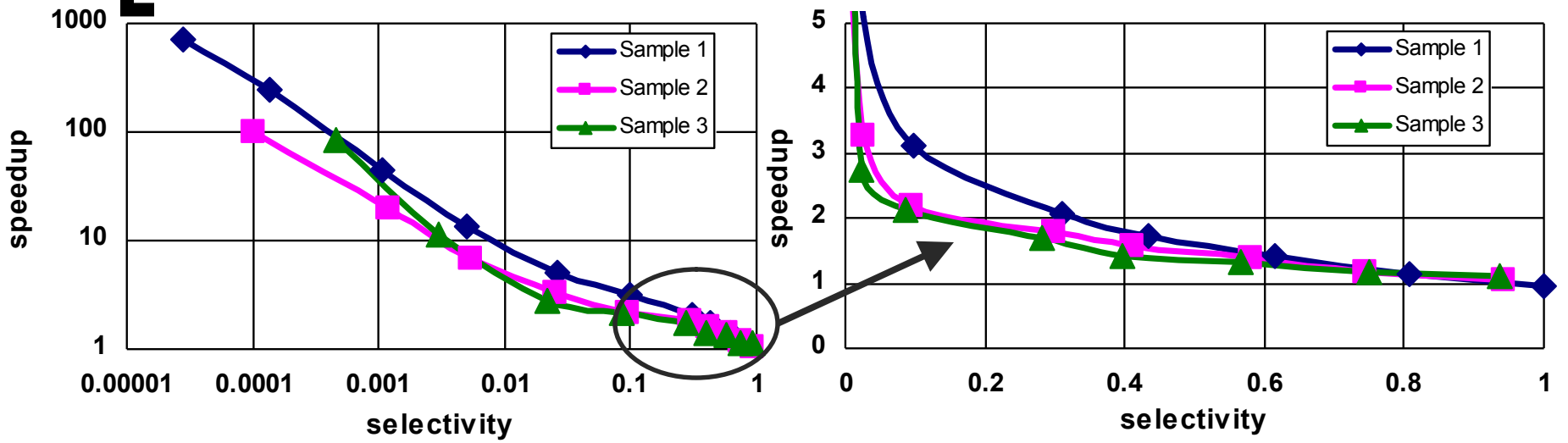


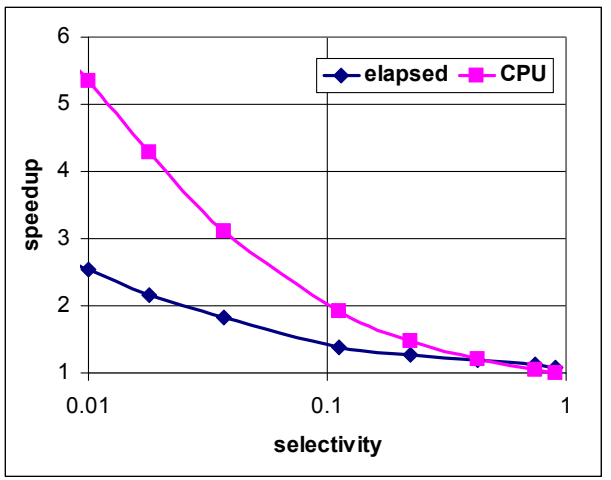


Bookkeeping and human level info (Tags)

- **Tags**: anything associated to event level information, may include detailed run information (by event) or data production information
- Granularity: event
  - Possibly huge (billion of events in Year 10 for STAR) implying selection AND storage could be a challenge
- Content examples and details:
  - **RunInfo** : M-D describing data taking and data production process for each run: run ID, beam parameters (composition, intensity, lifetime, polarization, fill ID, etc), STAR trigger detector rates, magnetic field, production version, time etc…
  - **EventInfo:** M-D describing each event: event ID, runID (for index search in case of merging), trigger Mask for the event, …
  - **EventSummary** : M-D describing physics event. Information includes: number of tracks, number of good tracks, number of good primary tracks, number of positive/negative tracks, number of vertices, vertex types, mean pt, mean pt2, etc…
- Problems and sizes:
  - This M-D could be considered the data stream (in for STAR) – care is needed on what becomes external to the data (file, set) and what remains internal
  - Format known as “TAG file” (STAR internal) used with ~15 parameters used per file for fast forward of events – no aggregation of data (1/20th per file still a lot)
  - BitMap index techniques with 15 parameters or so a great success for a full run aggregation
  - Anything else remains internal (analysis user select)

Bookkeeping and human level info
(Tags)





The infrastructure related to this STAR (+SDM) developed & tried technology is not maintained.
Main problems:

- new data production implies re-generation of tags
- adding a parameter ↔ merging (delay)
- biggest: user approach “the more the merrier” is a problem (size)

More on Meta-Data?

Last thoughts & remarks

1. Defining M-D and usage in STAR – previous slides

2. Internally stored versus externally stored
   - If M-D (tag) can be stored in the data stream itself, do it (self-consistent)
     - Event information: runNumber, time data was taken, trigger, runType, …
     - No reason to rely on an event ID ↔ external M-D association scheme (over-kill as analysis will likely need it at each event anyhow)
     - The more granular, more likely its place (whole of M-D) is internal
   - Location choices?
     - If M-D will change with time (a) internal may not the way (case of calibration data) and (b) reproducibility of data production MUST be ensured
     - If M-D is internal, it does NOT prevent it from being external. FC may contain internal information for bookkeeping and rapid dataset selections (runNumber, trigger setup, …)
   - Operational choice?
     - If M-D is external and/or centralized, no workflow is self-sufficient
     - Ex:
       - Cloud data production from STAR + isolated resources + canned did not allow communication with external DB. Full DB 20 GB large not suitable for a VM
       - "DB snapshot" (< 0.5 GB) for Cloud – portable in a VM - Outcome: 12 Billion Pythia events generated over 400,000 CPU hours

Conclusions

- Many kind of Meta-Data in STAR
  - Structural and bookkeeping (human level operational or guiding information)
  - Version evolution and strong yet flexible API design important from the start: users should not know + but users should be helped (schema browser, code generator)
  - 3 APIs in STAR: Generic API + FileCatalog + Tags - tuned for usage
  - Pitfalls: Balance need to be achieved
    - Guiding Meta-Data could be large if not under control. General Run-time information, calibration, ……, Meta-Data and FileCatalog, event level information (tags)
- Features and approach
  - Provide all tools to users from day 1 – shield them from details & provide version evolution + Flexibility & convenience
    - STAR API is 10 years old – has served all the way and still working smoothly
    - STAR API allowed switching from Full DB to "DB snapshot"
    - WebService plug-and-play in operation as we speak …
  - Provide tools: interfaces to browse, represent (graph), code generate for read/write, browser to inspect schema
  - Physics reproducibility requires multi-layers timestamps, flavors, …
- Q – should it be external or internal?
  - Internal to first order ↔ self-consistency (don't drop it)
  - In STAR, external event based (tags) have showed to be hardly maintainable (size & dynamic)
  - Could be multiple-sources combined (probably best at first)
    - Tags showed not to be practical may years through the program …
    - DaqEventTag (also a form of tags) survived 7 years of running then dropped
  - External M-D has some impact on distributed computing processing
    - Cloud usage in STAR with Virtualization "self-canned" approach especially …
    - Many services need locality …