

Event Data Model in the STAR experiment at RHIC

GRIGORY NIGMATKULOV

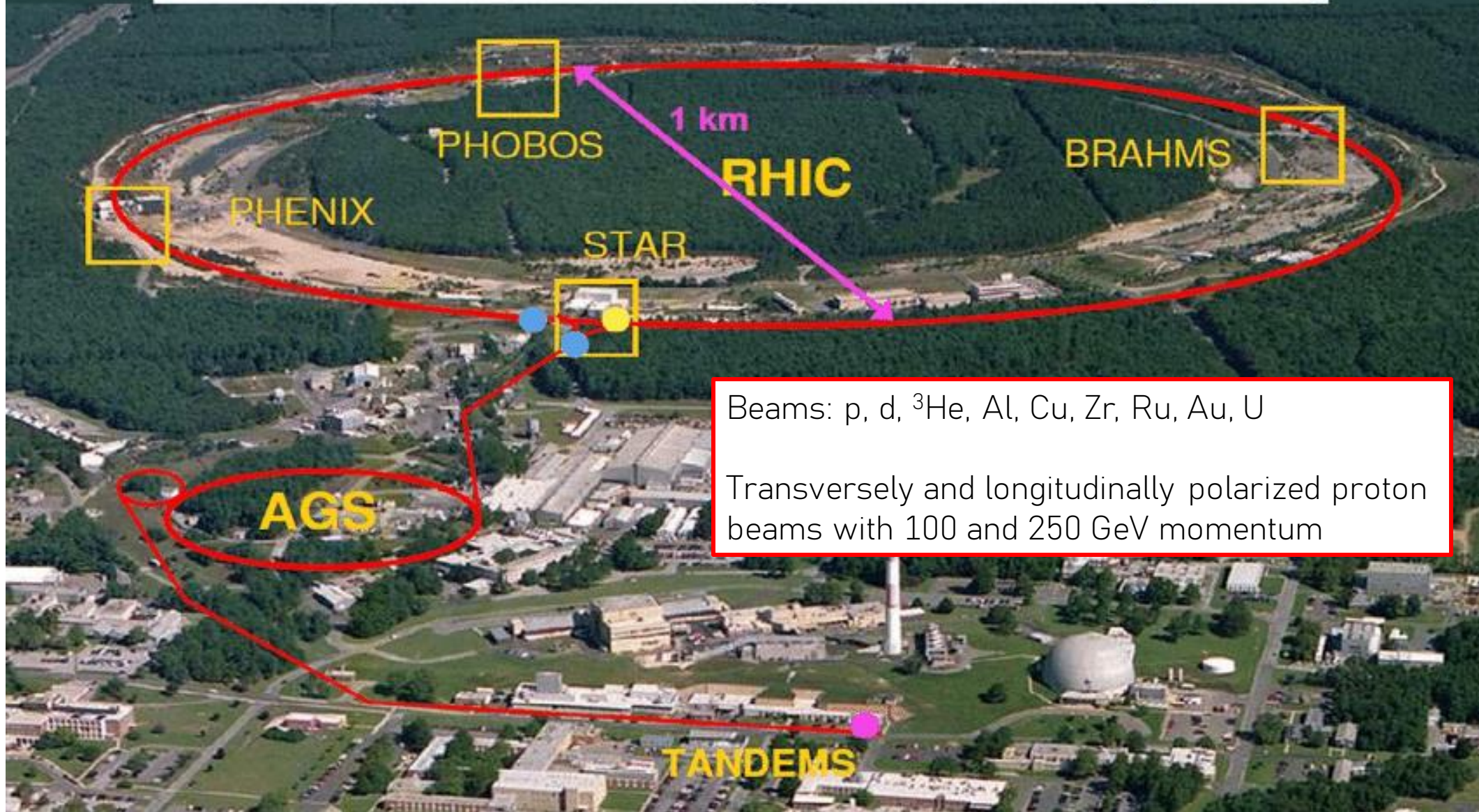
NATIONAL RESEARCH NUCLEAR UNIVERSITY MEPhI

SPD Physics & MC Meeting
Dec. 14, 2022

Brookhaven National Laboratory (BNL)

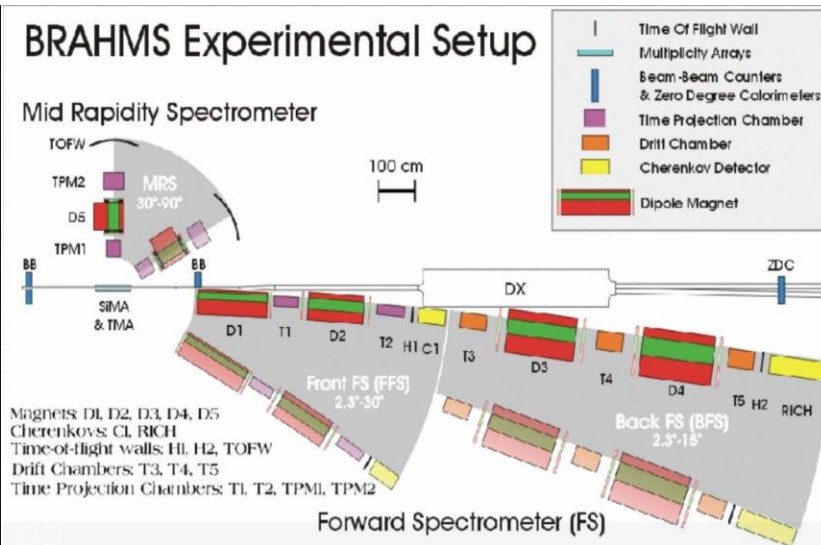


The Relativistic Heavy Ion Collider (RHIC)



Beams: p, d, ^3He , Al, Cu, Zr, Ru, Au, U
Transversely and longitudinally polarized proton beams with 100 and 250 GeV momentum

Experiments at RHIC



Years of operations:
 BRAHMS: 2000-2006
 PHOBOS: 2000-2006
 PHENIX: 2000-2017
 STAR: 2000-present

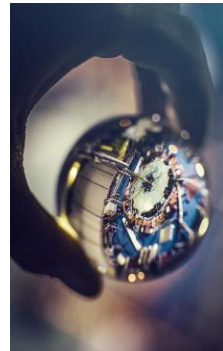
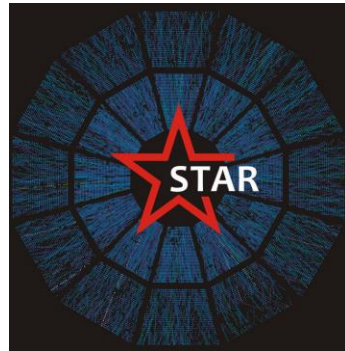
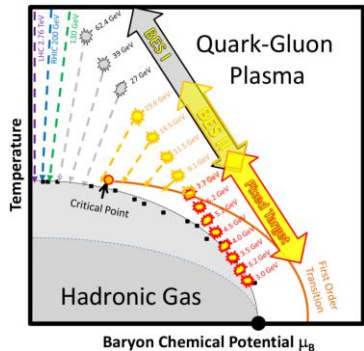
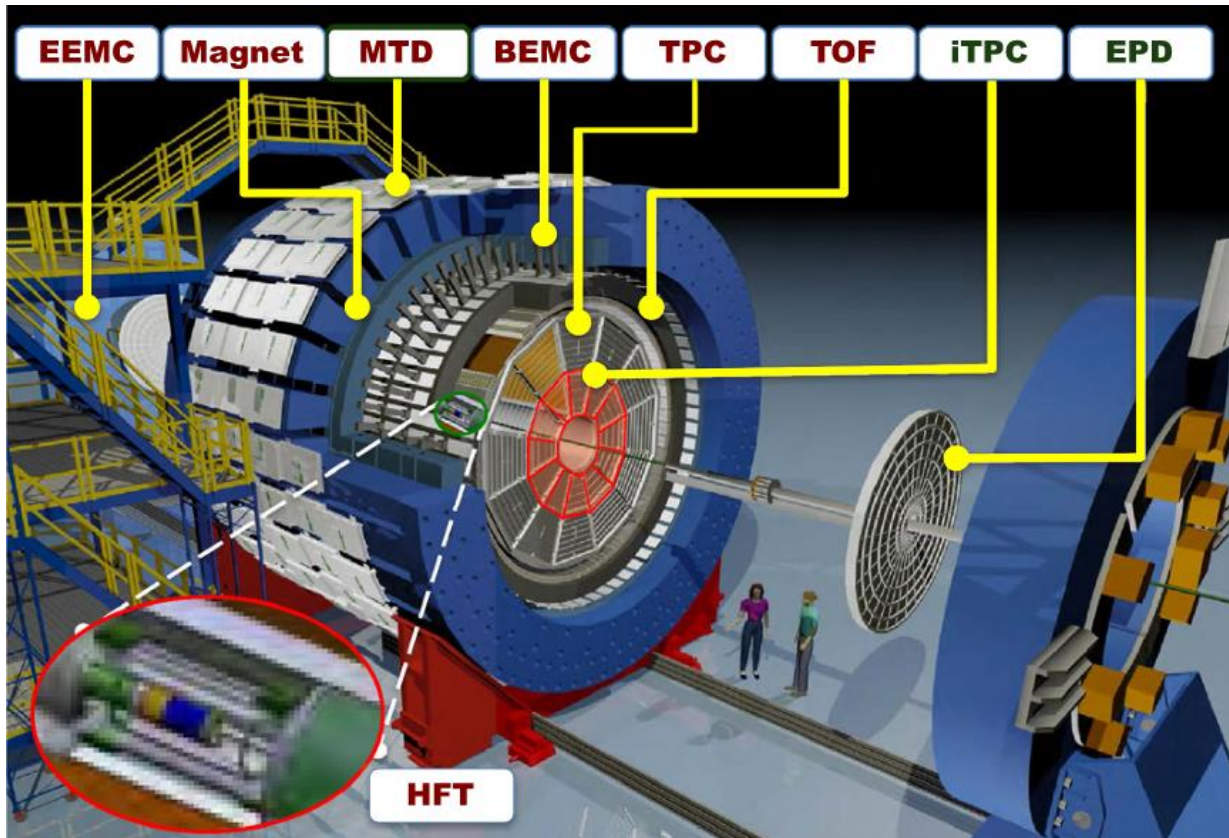


GRIGORY NIGMATKULOV, EDM IN STAR

Solenoidal Tracker At RHIC (STAR)



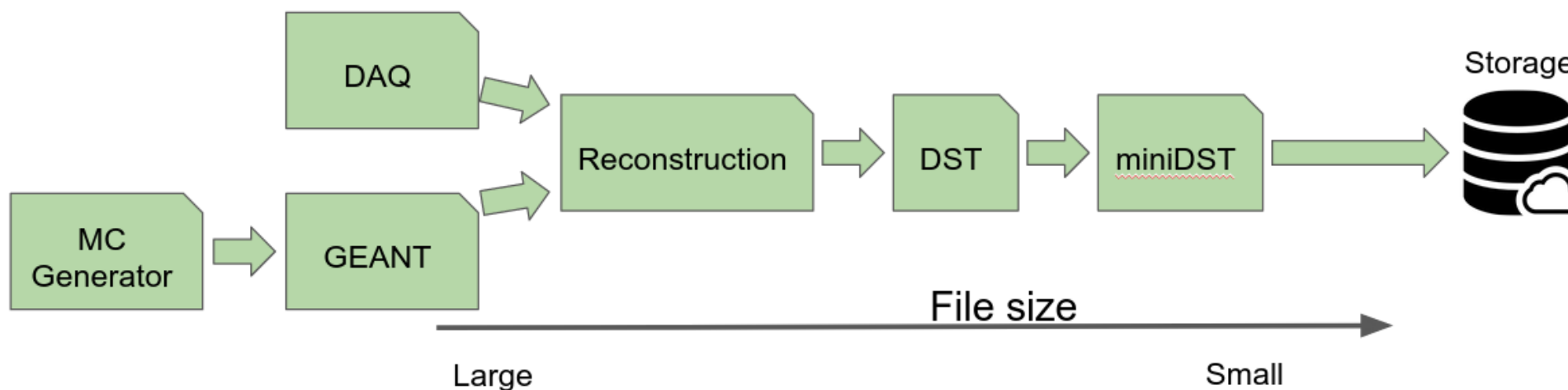
STAR is composed of 71 institutions from 14 countries, with a total of 740 collaborators.




DST for Reconstructed Data



The data formats may vary from an experiment to experiment, but a general (oversimplified) scheme of the experimental data flow:



Nuclear Instruments and Methods in Physics Research A 389 (1997) 81–86

 ELSEVIER

NUCLEAR INSTRUMENTS & METHODS IN PHYSICS RESEARCH
Section A

ROOT – An object oriented data analysis framework

Rene Brun^{a,*}, Fons Rademakers^b

^aCERN, Geneva, Switzerland
^bNIKHEF & Hewlett-Packard, Geneva, Switzerland

5. The ROOT Trees

For many years, the data flow model in HEP has been:
Raw Data Tapes → Data Summary Tapes → Mini/Micro DSTs

First Day's STAR Event Data Model (EDM)



- STAR was planned to perform at 5 Hz rates (current trigger rates ~ 5-10 kHz)
- Suppose to store everything within TTree (STAR's event data format (**StEvent**))
- Persistence and ROOT:

All StEvent [classes inherit from StObject which itself inherits from TObject](#). During the build of StEvent all classes run through rootcint. This adds the following features:

1. All StEvent classes can be used on the root4star command line.
2. Almost all StEvent classes are persistent capable, i.e. they can be stored in ROOT files.

As usual each coin has two sides. The disadvantage of this is that we cannot use some features of the ANSI/ISO C++ and from the Standard C++ Library as:

- type bool
- Templates
- STL containers and algorithms
- Namespaces

This however applies for the header files only. Source files are not processed via rootcint and therefore all the stuff mentioned above can be used. And indeed in the implementation of various StEvent classes we make heavily use of the STL

Retrieving Data from StEvent



- StEvent is set up and filled in a “maker” with the name StEventManager. ([Modern ROOT switched from the Maker scheme to Task](#)).
- This maker reads DST tables stored in memory and does all the things to make StEvent nice and useful. In principle all you have to do is to make sure that StEventManager is in the chain and called at the right place and at the right time

The StEvent Model



- StEvent opens the door to all the info there is on the DST. To get there, one needs to navigate through the tree. Only few objects, mostly container and collections, can be reached directly from the StEvent objects. Here's a list of some important objects which are directly stored in StEvent and let you climb further down the tree:
 1. Collection of software monitors
 2. TPC hit collection
 3. FTPC hit collection
 4. SVT hit collection
 5. List of all track nodes
 6. List of the detector info for each track
 7. Primary vertices (mostly only one)
 8. List of all V0 vertex candidates
 9. List of all Xi vertex candidate
 10. List of all kink vertex candidates
 11. Level-0 trigger

Software Monitors

- The STAR DST contains a bunch of tables called software monitors. During the reconstruction of the various detectors lots of statistics and summary information is generated which is not necessarily of importance for the physics of the event but tells you a lot on how the reconstruction programs performed. These are mostly quantities which cannot be derived from other objects in StEvent and would be lost otherwise. In a sense they monitor the reconstruction details. That's where the name 'software monitor' comes from

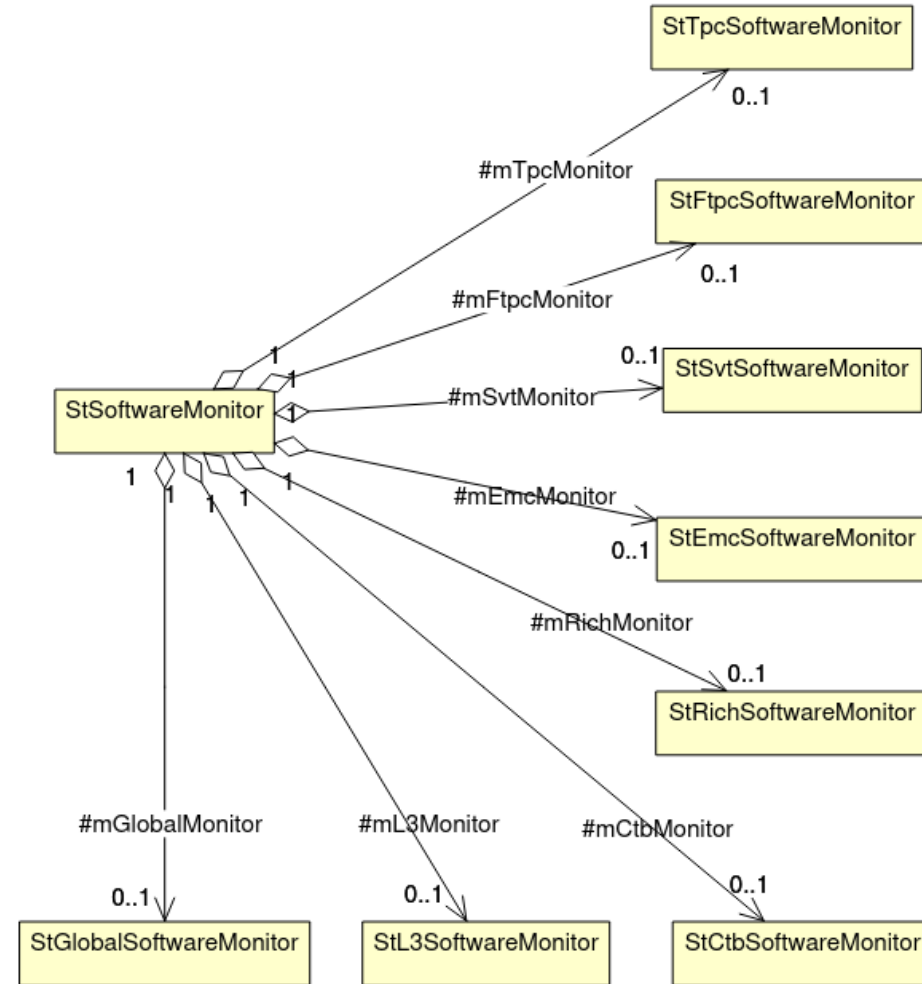
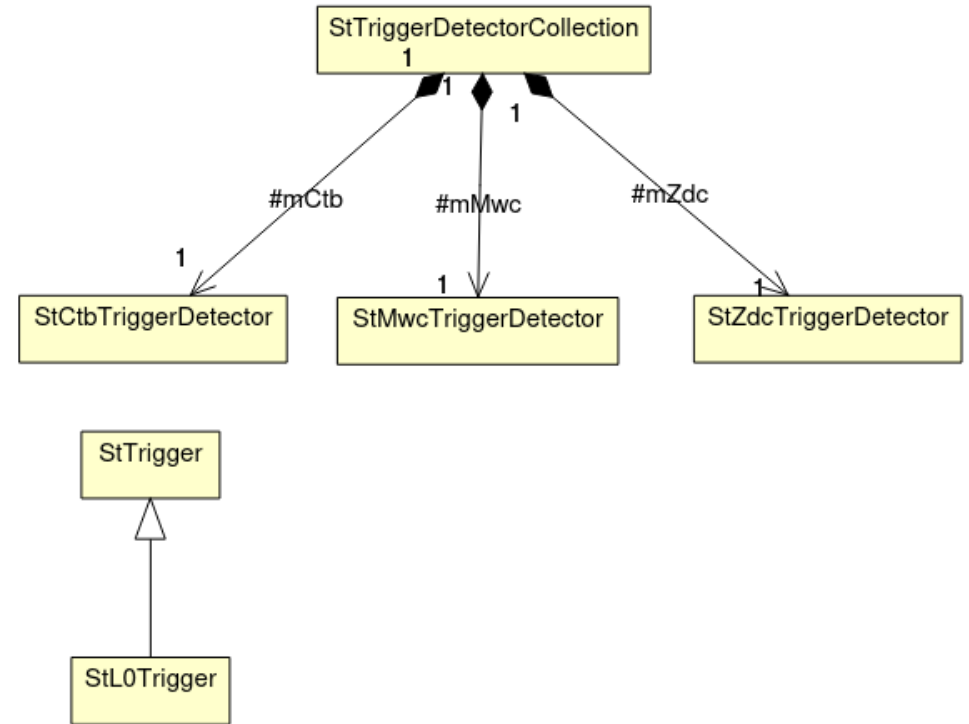


Figure 3.2: Class diagrams for the software monitors.

Trigger and Trigger Detectors



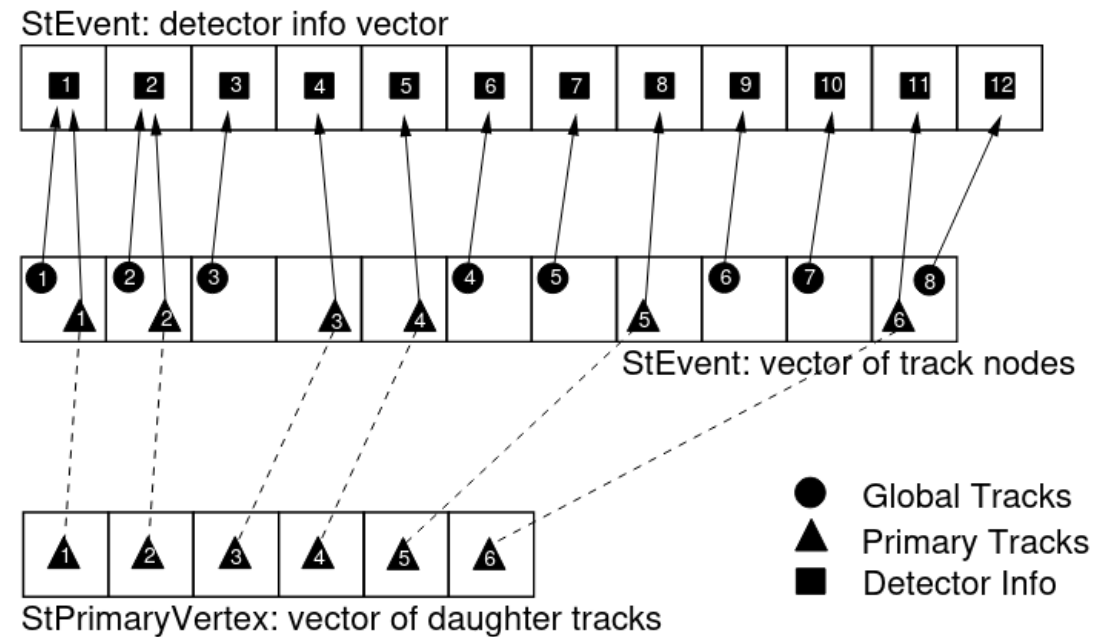
- The **trigger** is put together from data recorded by a bunch of trigger detectors combined in some logic. So far STAR deals with 4 trigger levels numbered 0 – 3. Currently only level-0 (L0) is implemented. Others will follow. All trigger classes inherit from a common base class StTrigger.
- The **trigger detectors** are those detectors which data is used in the trigger (which doesn't mean that the data isn't useful for other things as well). There's a couple of them: the Central Trigger Barrel (CTB), the Zero Degree Calorimeter (ADC), the Vertex Position Detector (VPD), and the Multiwire Proportional Chamber (MWC). This means we need a collection to hold them together and indeed this is what StTriggerDetectorCollection is all about.



Tracks



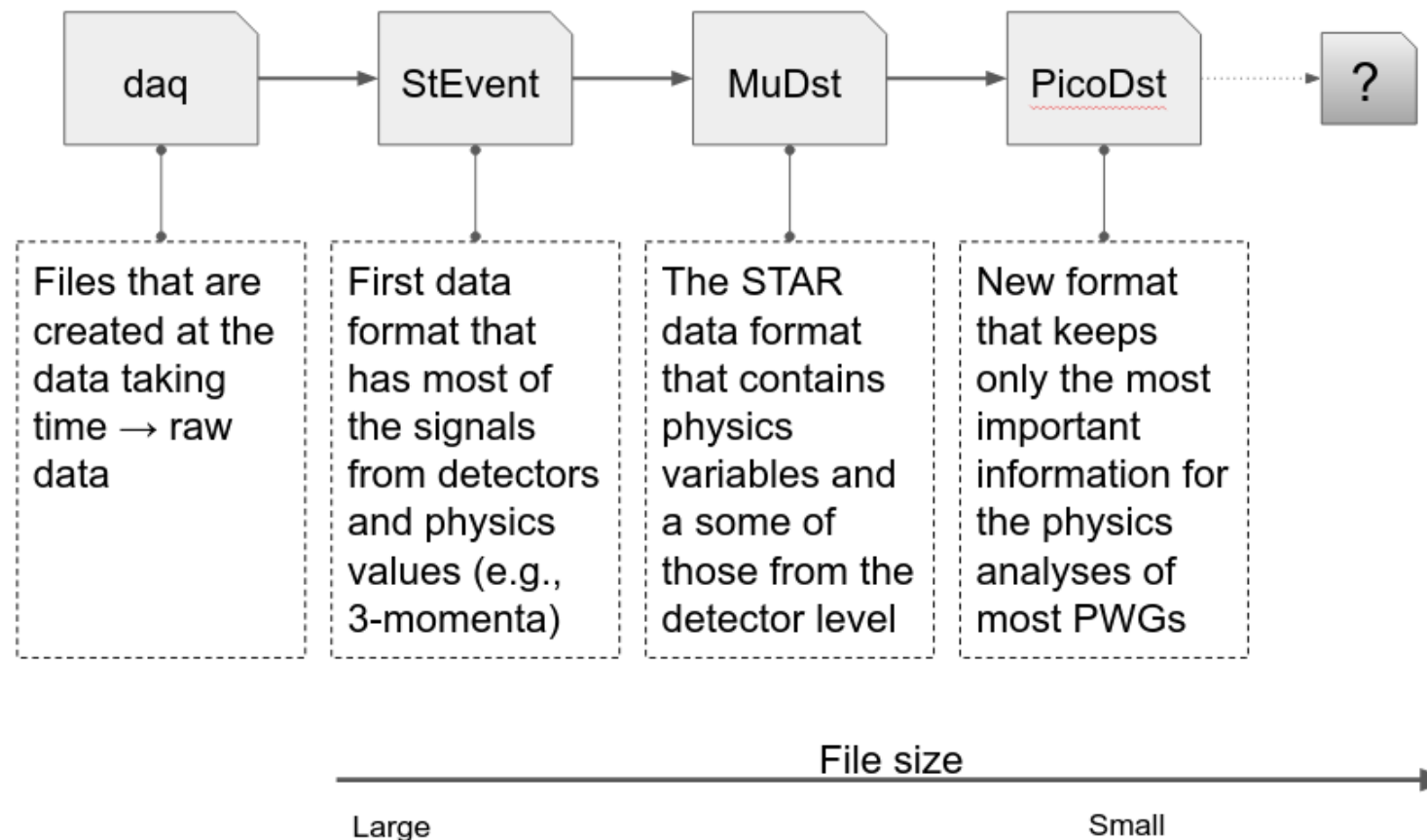
- There are two kinds of tracks (global and primary) of which each might get possibly fitted with different models or algorithms such as creating a whole bunch of tracks. But we have to keep in mind that all come originally from the same seed formed early in the reconstruction chain.
- A track node holds all tracks which originate from the same seed. Every track knows about the node it belongs to and thus allows to navigate from one track in the node to the other. Each node contains 1–n tracks
- All detector specific information (essentially the list of hits) is contained in this class. A track can well live without them since all the reconstruction is already done.



StPicoDst (since 2018)



STAR ☆ Data production in STAR



Modern Requirements



STAR ☆ The format requirements

1. As small as possible
 - a. Small file size
 - b. Only vital variables

2. Independent of STAR software
 - a. Must work on any computer farm or laptop with vanilla ROOT
 - b. Only simple (native) data types (char, short, float)
 - c. Should be easily compiled with Makefile or CMake

3. Easy to produce
 - a. Produce from BFC (daq→StEvent→μDST→picoDst)
 - b. Produce from μDST→picoDst (useful for existing data sets)

What is Currently In?



What does picoDst contain?

[StPicoDst](#) - main container class that allows to access data

[StPicoDstReader](#) - allows to load and read picoDst files or list of those

[Event](#) - event info

[Track](#) - track info

[BTofPidTraits](#) - barrel TOF-matched tracks

[BEmcPidTraits](#) - BEMC-matched tracks

[MtdPidTraits](#) - MTD-matched tracks

[ETofPidTraits](#) - endcap TOF-matched tracks

[BTofHit](#) - barrel TOF hit

[ETofHit](#) - endcap TOF hit

[BbcHit](#) - BBC hit

[EpdHit](#) - EPD hit

[SystemOfUnits](#) - units for conversion

[PhysicalConstants](#) - physical constants and conversions

[BTowHit](#) - BEMC tower info

[BEmcSmdE\(P\)Hit](#) - BEMC SMD hit

[MtdHit](#) - MTD hit

[FmsHit](#) - FMS hit

[TrackCovMatrix](#) - track covariance matrix

[EmcTrigger](#) - BEMC trigger info

[MtdTrigger](#) - MTD trigger info

[\(Physical\)Helix](#) - helix parameterization

[MessMgr](#) - class to handle messaging

[Makefile](#) - makefile to compile StPicoDst classes to the lib that one can use to run on a local machine (laptop, university cluster)

[picodst_env.sh](#) - bash script to set the picoDst environment variable to compile and run codes in a standalone mode