Evolution of the STAR Framework OO Model for Multi-Core Era

V.Fine, J.Lauret,V.Perevoztchikov
for the STAR collaboration

Office of Science
U.S. DEPARTMENT OF ENERGY

BROOKHAVEN NATIONAL LABORATORY

History

The Solenoidal Tracker At RHIC (STAR) is a large acceptance collider detector which started data taking at Brookhaven National Laboratory in summer 2000.

At the time STAR developed a modular ROOT package-based Object-Oriented framework for simulation, reconstruction and analysis in offline production, interactive physics analysis, and online monitoring.

With the era of multi-core CPUs, software parallelism is becoming both affordable as well as a practical need.

Especially interesting is to re-evaluate the adaptability of the high energy and nuclear physics sophisticated, but time-consuming, event reconstruction applications to the reality of the multi-threaded environment.

Is it acceptable?

- Discussions within STAR collaboration indicated that any appealing solution means *no changes* of codes should be needed by STAR scientist / changes are not acceptable

- The usage of the parallel architectures is needed and, if it can be done *transparently* for the existing end-user codes, would bring immediate benefits to experimental frameworks

STAR framework OO model

STAR framework is designed to support the chronologically chained components, which can themselves be composite sub-chains, with components (*"makers"*) managing named "*datasets*" they have created and are responsible for.

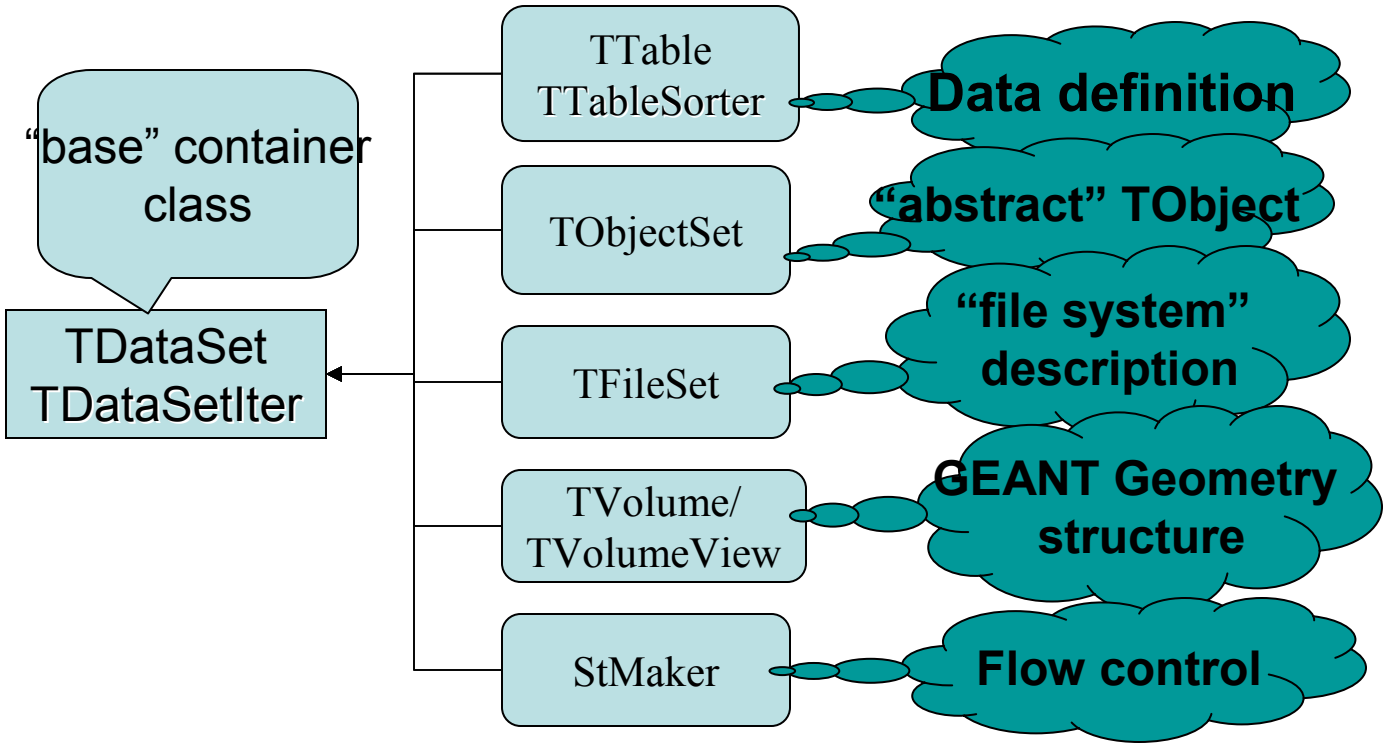
Makers and data sets inherit from the TDataSet class which supports their organization into hierarchical structures for management.

TDataSet also centralizes almost all system tasks:

- data set navigation,
- I/O, database access,
- inter-component communication.

OO model of the STAR simulation / reconstruction chain (ancient version)

TDataSet object ::= the "named" collection of TDataSet objects



The C++ classes with the prefix "T" are available from the "table" ROOT package

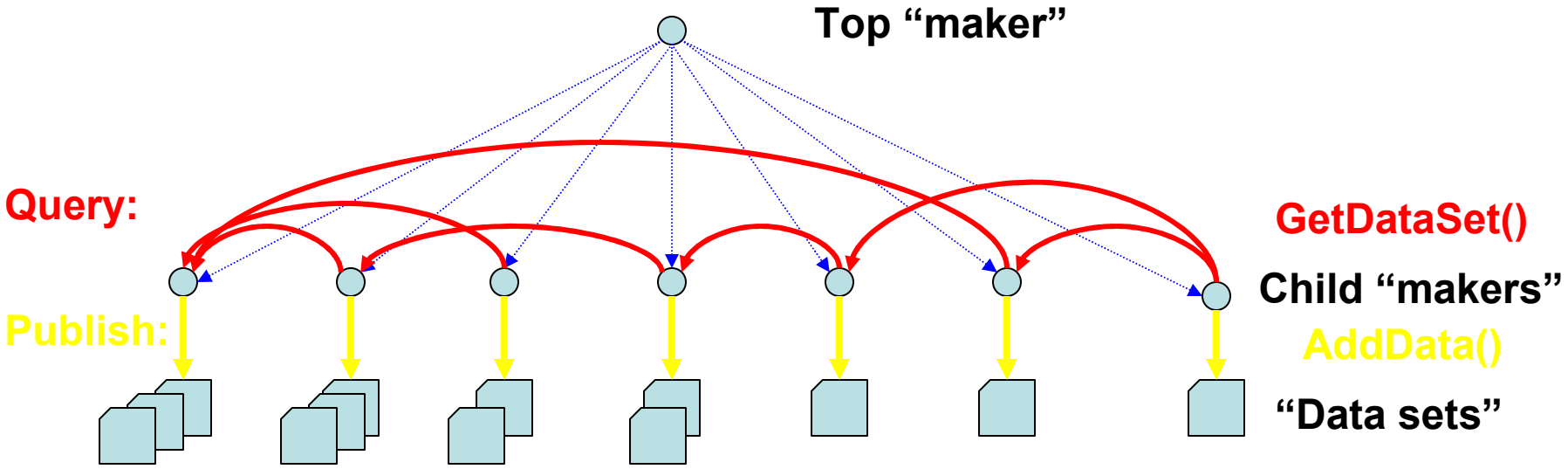
STAR Framework Features

The main features of the STAR framework design:

- its “makers” and “data sets” share the common object model
- they are derived from one and the same base TDataSet class
- the entire STAR reconstruction chain is a single instance of the StChain class (subclass of TDataSet)
- This includes both “maker” objects as well as “data sets”.

STAR "Maker" communication within the reconstruction chain

At its basic principle, modules do **not** communicate with each other directly and act as **consumers and providers of data structures**. They use the framework via the "query" represented by method StMaker::GetDataSet / "publish" represented by method StMaker::AddData() API to query the presence of the input data and publish the output results the modules produce.



Each maker can query the data published by the other makers.

3 “transparent” steps towards multi-core era

By

- Complementing the TDataSet-base framework with the ability to start several modules in parallel
- synchronizing the global data access between the "consumer" modules and "producer“

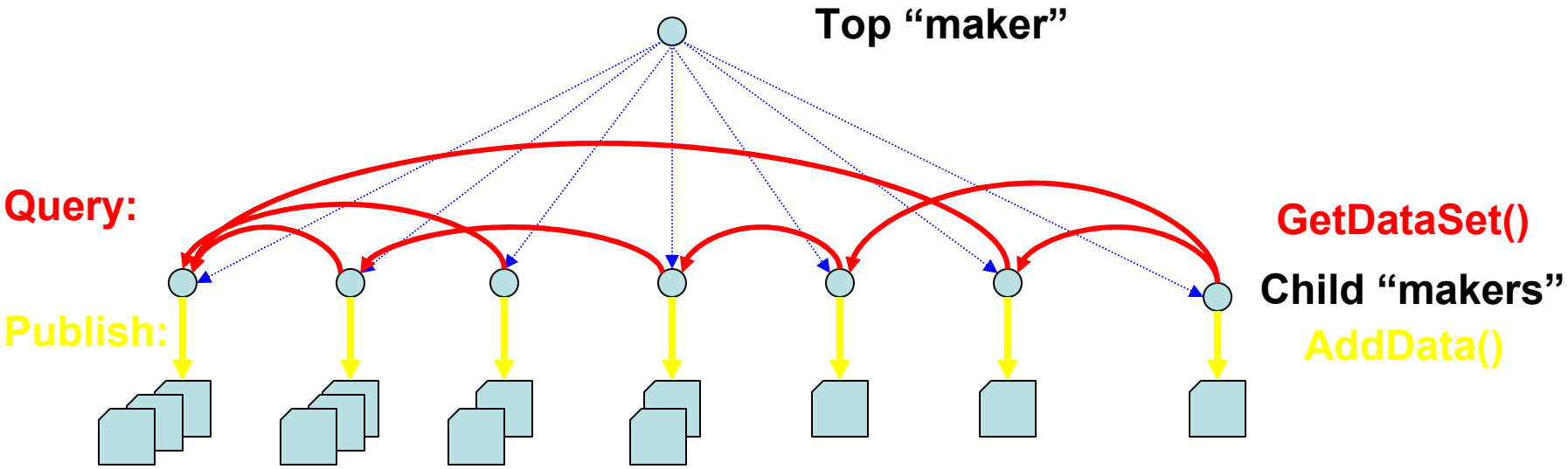
one can “transparently” enhance the existent packages to leverage the multi-core hardware capabilities.

Evolution: main directions

We are concentrating our effort to implement and test the crucial components such as "transparent"

- **parallelization,**
- **synchronization,**
- **registration**

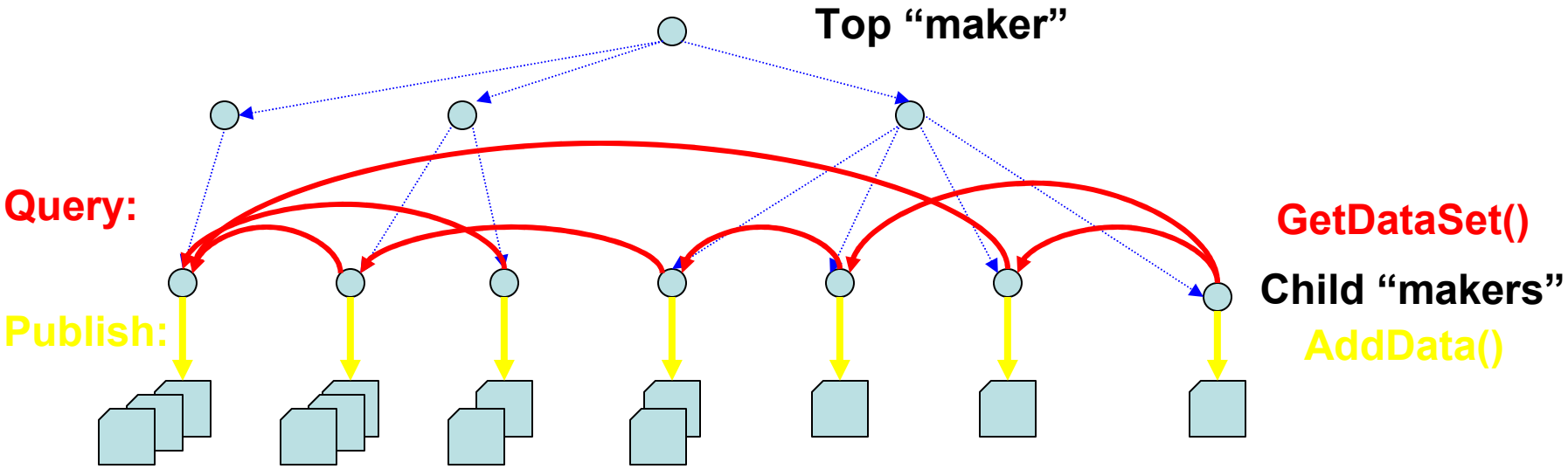
Evolution: main directions



Each maker can query the data published by the other makers.

GetDataSet() and AddData() are “base” methods. Normally, within STAR framework, no end-user subclass is to re-implement it.

Evolution: main directions



Each maker can query the data published by the other makers.

Change the hierarchical level does not change the ability to query the data and doesn’t require to change any low-level implementations either.

“Transparent” parallelization and synchronization

It is done by

- Re-implementing the base class methods used to handle main initialization, executing, end and garbage collection
  - like Init(), Make(), Finish() and Clear() in STAR’s StMaker

  to allow generating or using as many threads as many “makers” are present on the next level of hierarchy instance of the StChain class instead of the calling the methods in loop

- It is enough to add the extra thread related private data-members to the base class. One needs of a data-member for the thread ID and make methods “atomic”.

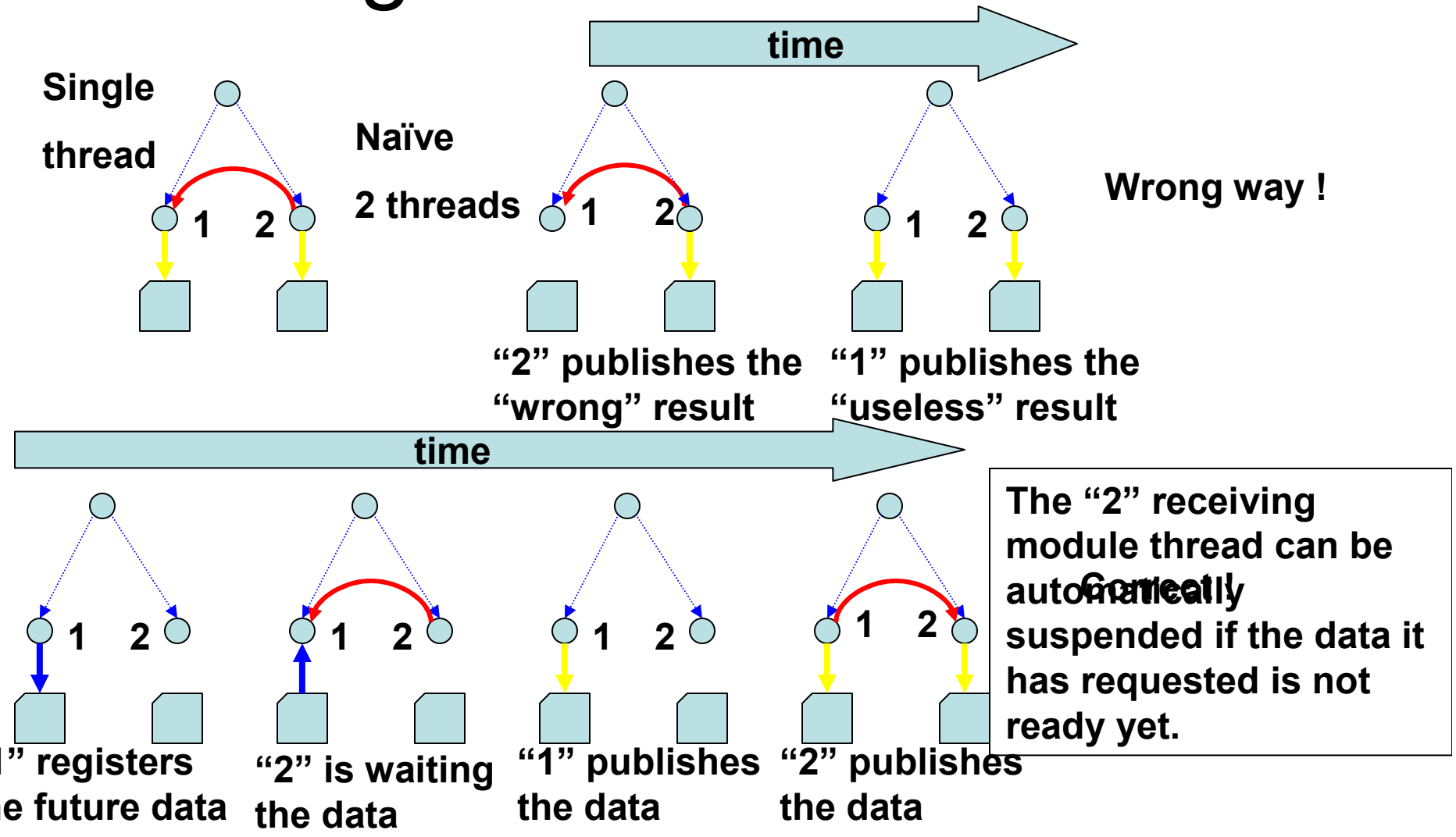
“Transparency” can be achieved without change of the class public interface

“Transparent” registration

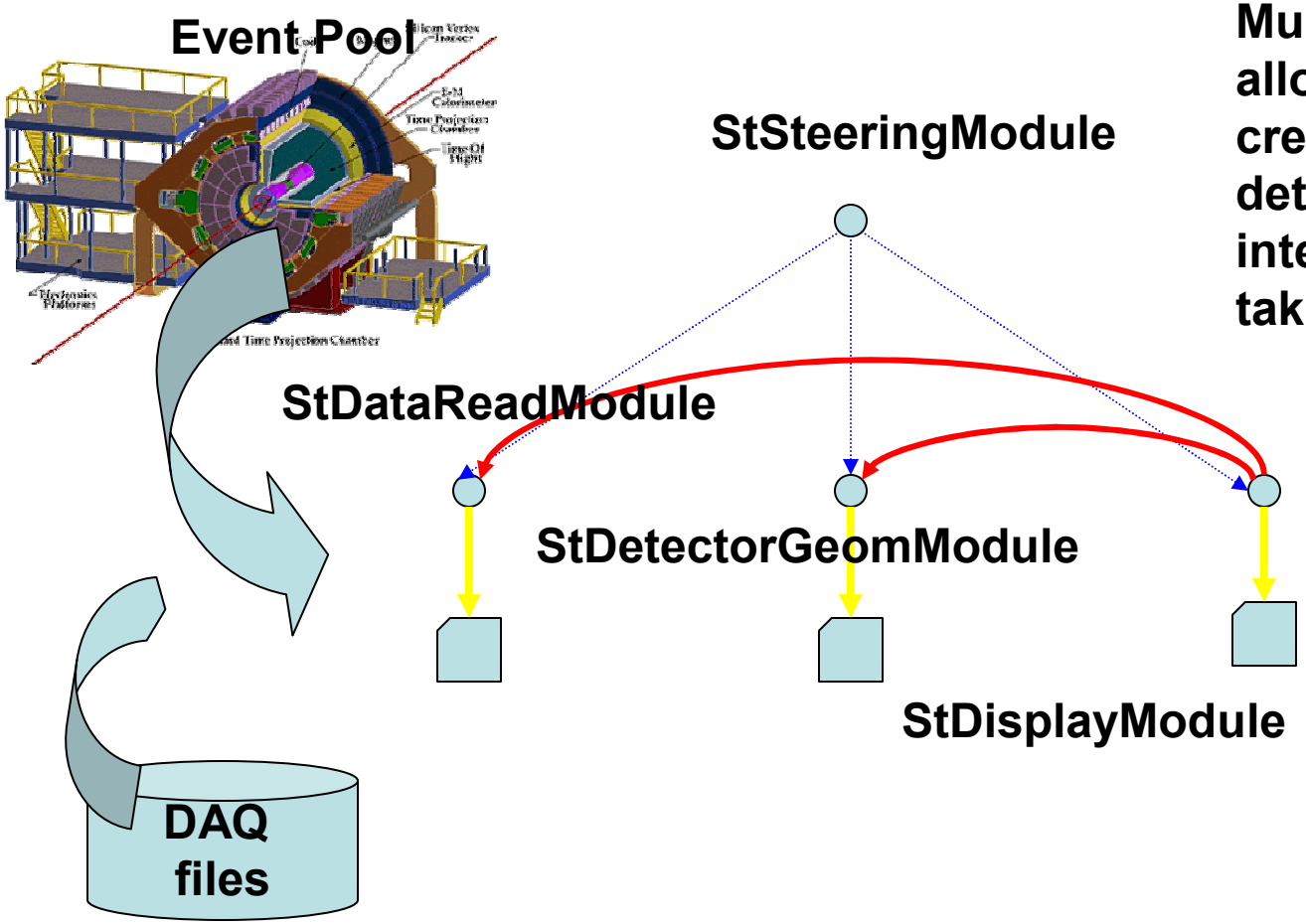
- BUT, we realize that the "**query**"/"**publish**" paradigm is not sufficient to run the multi-threaded application effectively.

- It should be complemented with an API to “**register**” the module output to notify the framework members about an output dataset “to be produced soon”.

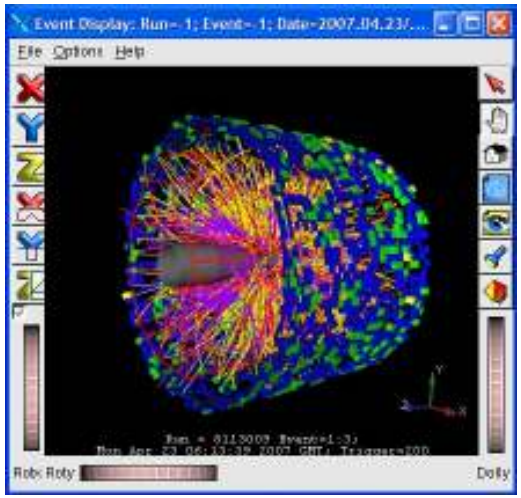
“registration” use case



Practical step: “Event Display”



Multi-threaded version allowed the STAR shift crew investigate the detector parameters interactively without data taking interruption.



Status

- The analysis showed that the existing design of STAR single-threaded framework can evolve transparently to meet the Multi-Core Era requirement with no revolution.
  - It is clear one should anticipate many problems ahead. We are concentrating our effort to implement and test the crucial components such as “transparent” **parallelization, synchronization, registration.**
- Our approach relying on TDataSets could be made general for ROOT users
  - Changes will be added to ROOT SVN
  - (one only need to use it ☺)
- The first version of the “parallel” chain was used to enhance the STAR Online Event Display.
  - It has been deployed during the Run8 and proved the approach can be implemented on the large scale with the reasonable amount of manpower.
  - The enhanced version of our beta-application example will be used during Run9
  - STAR will then be ready to expand its production or analysis framework and move ahead in the multi-core era
    - Benefits need to re-evaluated