Reliable online data-replication in LHCb

Daniel Sonnick

CERN

March 23, 2009

Outline

Motivation

- The LHCb detector registers events with a rate of 40 MHz. From these 40 MHZ 2 KHz are selected and stored on a local storage in raw-data files

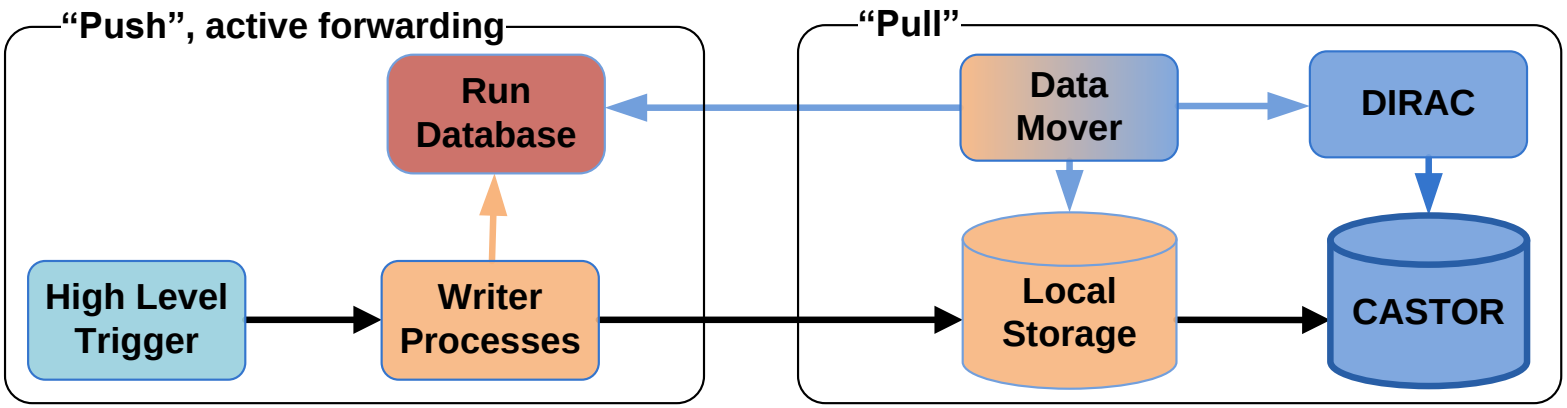
Motivation

- The LHCb detector registers events with a rate of 40 MHz. From these 40 MHZ 2 KHz are selected and stored on a local storage in raw-data files
- The Tasks:
  - Copy the data to a long term storage system in a reliable manner

Motivation

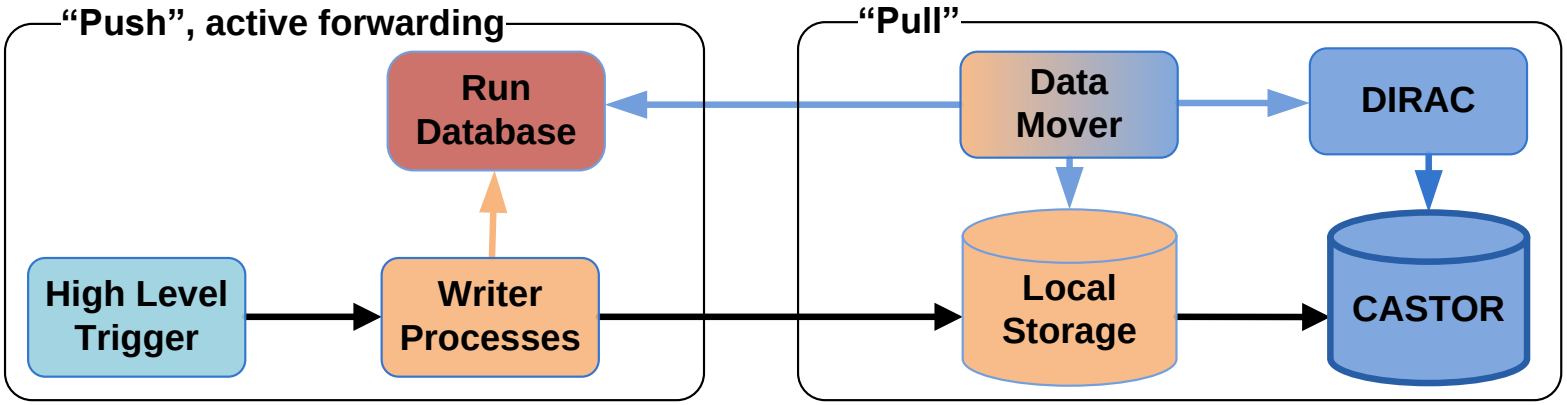
- The LHCb detector registers events with a rate of 40 MHz. From these 40 MHZ 2 KHz are selected and stored on a local storage in raw-data files
- The Tasks:
  - Copy the data to a long term storage system in a reliable manner
  - Prepare reconstruction and reprocessing by registering the data in the offline databases

Introduction



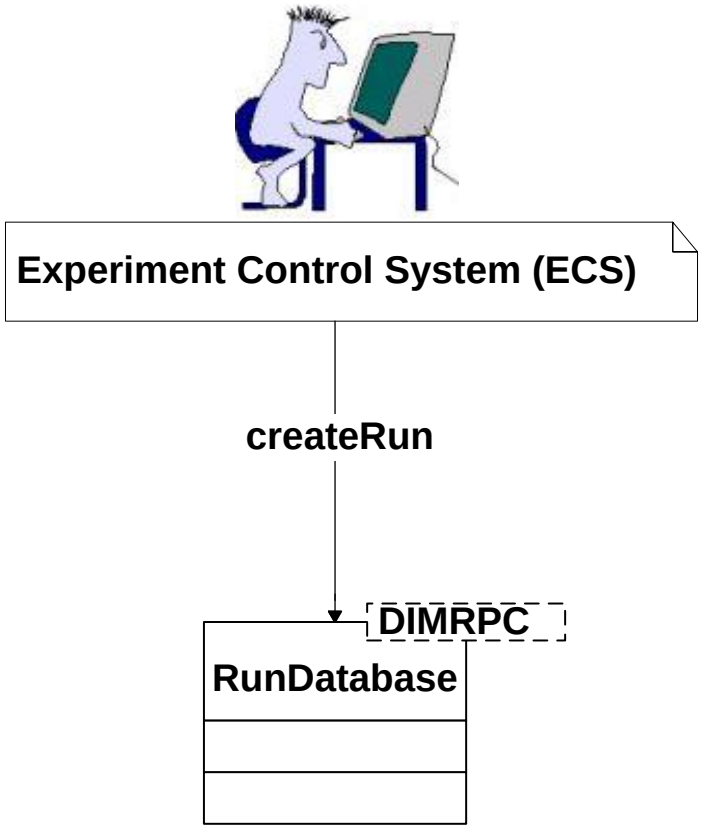
- From the High Level Trigger (HLT) Farm to local storage
  - The HLT sends events to speed optimized file writing processes
  - File identifiers are retrieved from the Run Database
  - Events are pooled together and are written into files on a local storage system

Introduction



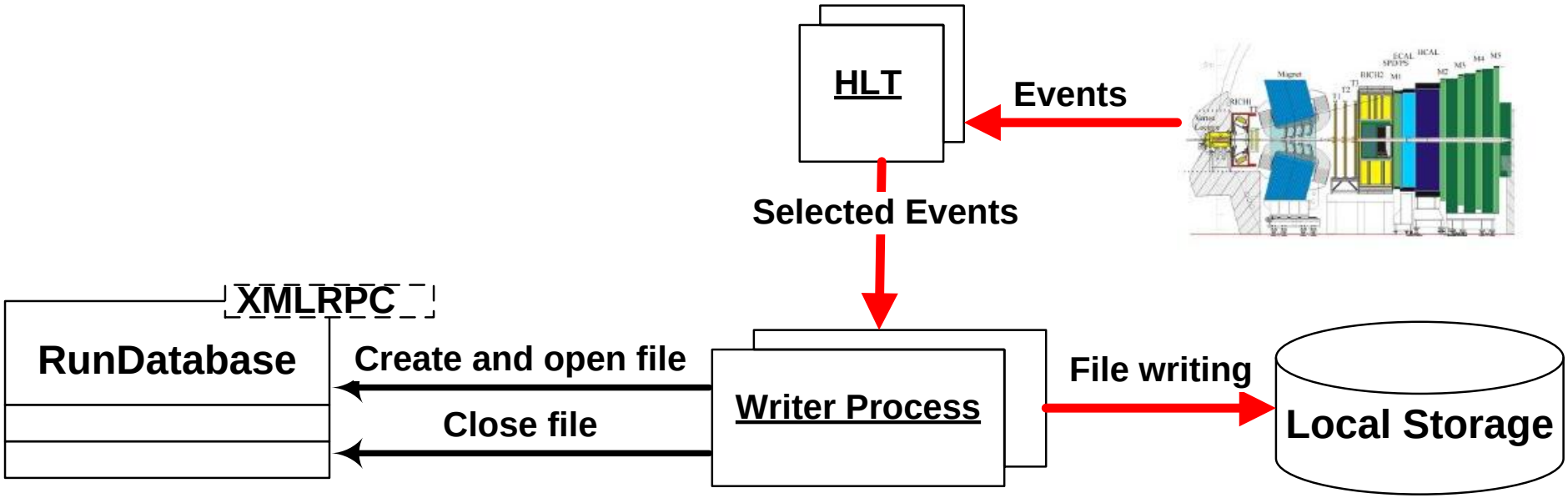
- From the High Level Trigger (HLT) Farm to local storage
  - The HLT sends events to speed optimized file writing processes
  - File identifiers are retrieved from the Run Database
  - Events are pooled together and are written into files on a local storage system
- From the local storage system to long term storage
  - The Data Mover is polling the Run Database for new files
  - Using LHCb's Grid middleware tool DIRAC these files are then transfered to CASTOR (Cern Advanced Storage Manager) which is CERN's facility for long term storage

Storage System: Run Creation



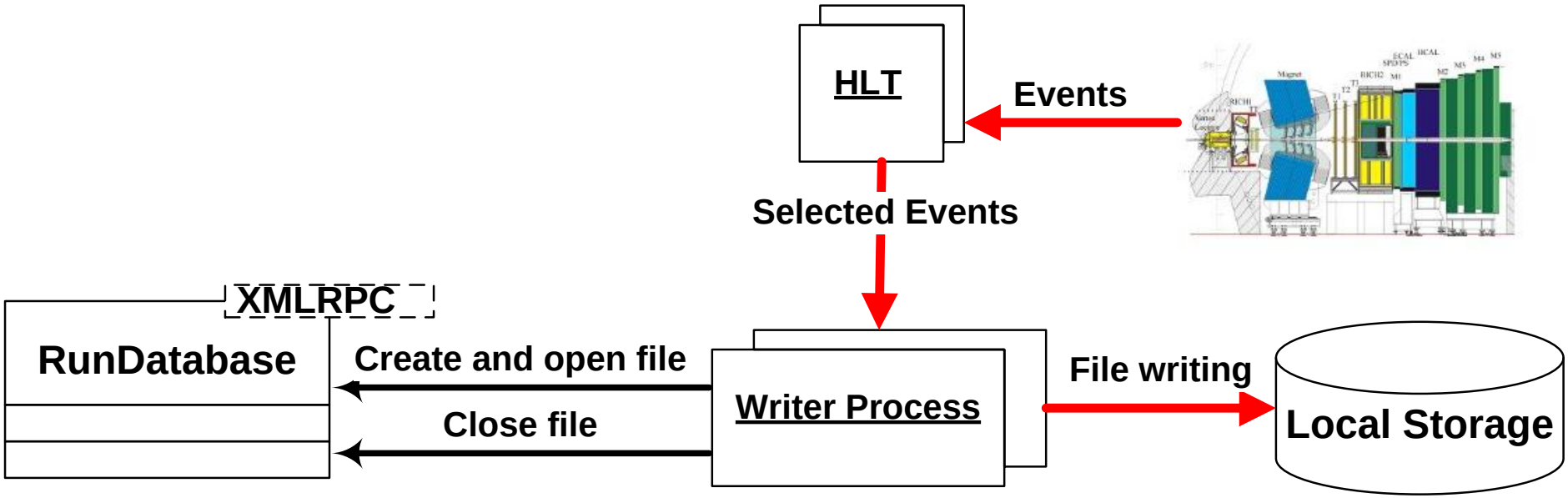
- Runs (data taking sessions) are created via DIMRPC by the Experiment Control System (ECS)
- After creating the run in the Run Database the run number is returned

Storage System: File Writing



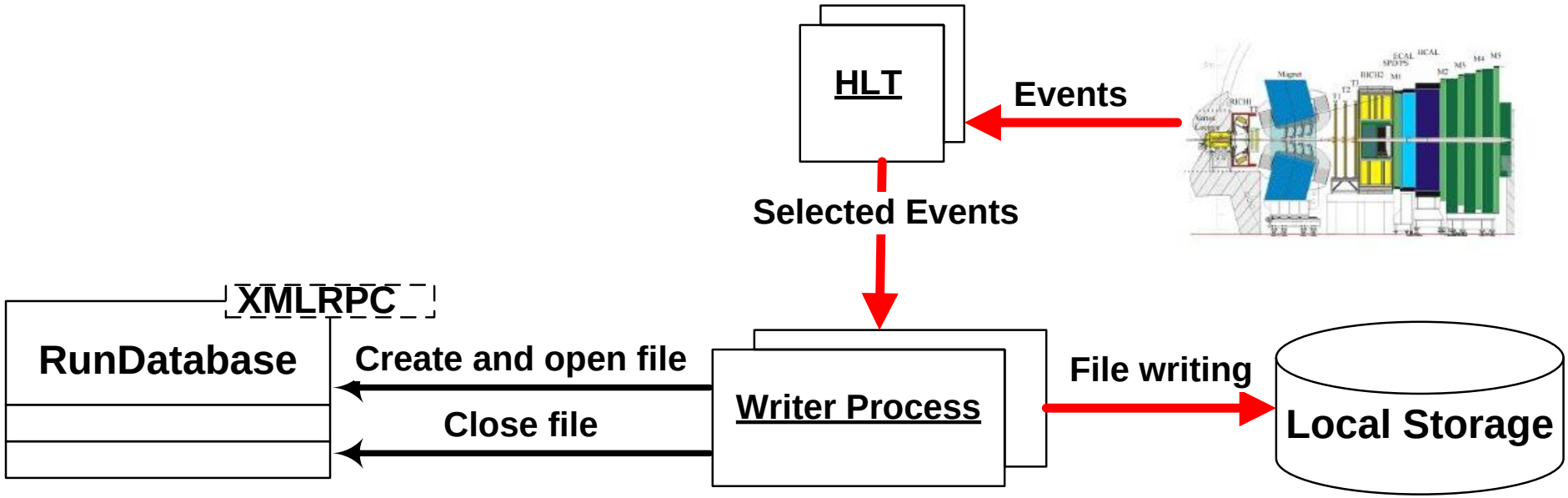
- The Writer processes are receives events by the High Level Trigger (HLT) Farm

Storage System: File Writing



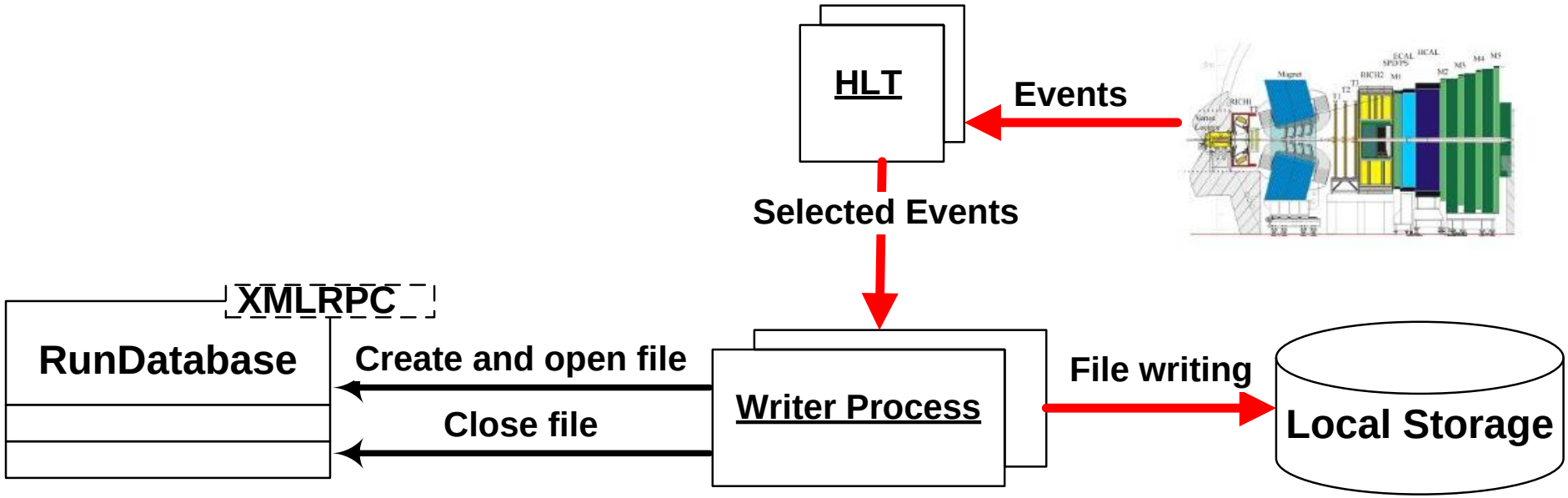
- The Writer processes are receives events by the High Level Trigger (HLT) Farm
- Events a pooled and then written into files

Storage System: File Writing



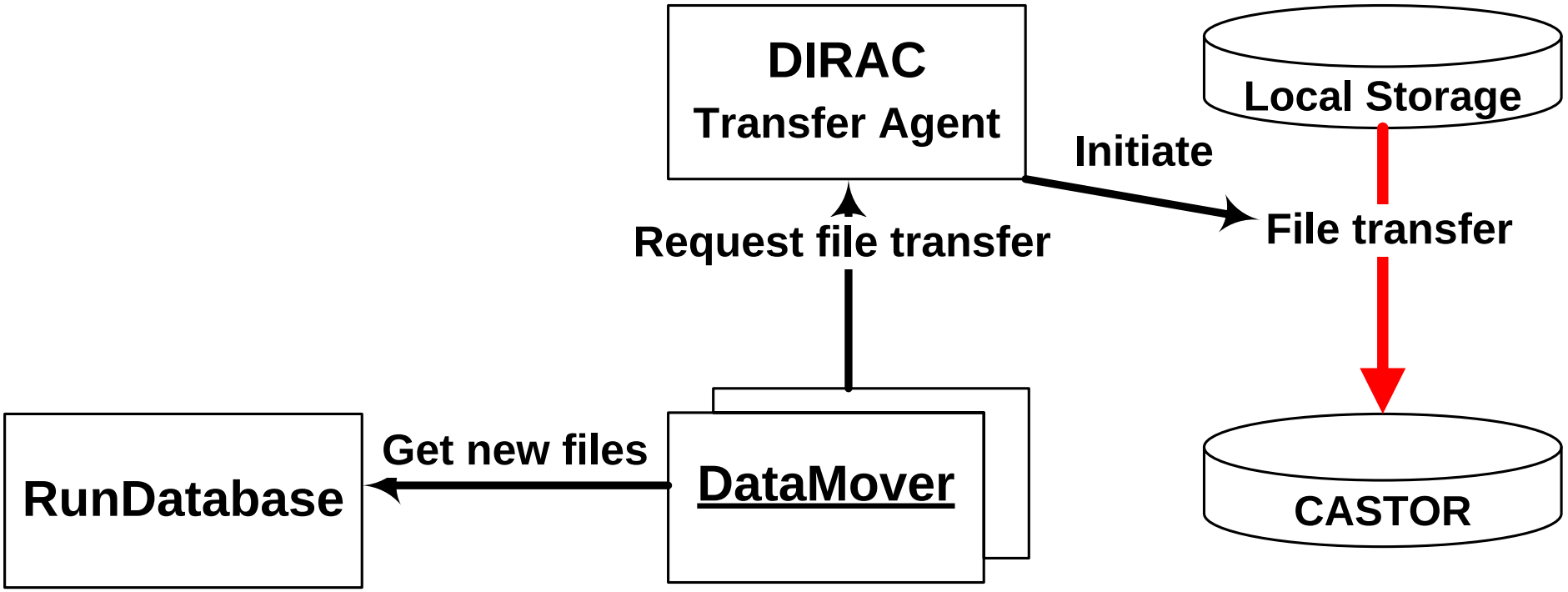
- The Writer processes are receives events by the High Level Trigger (HLT) Farm
- Events a pooled and then written into files
- File names are retrieved from the Run Database via XMLRPC

Storage System: File Writing



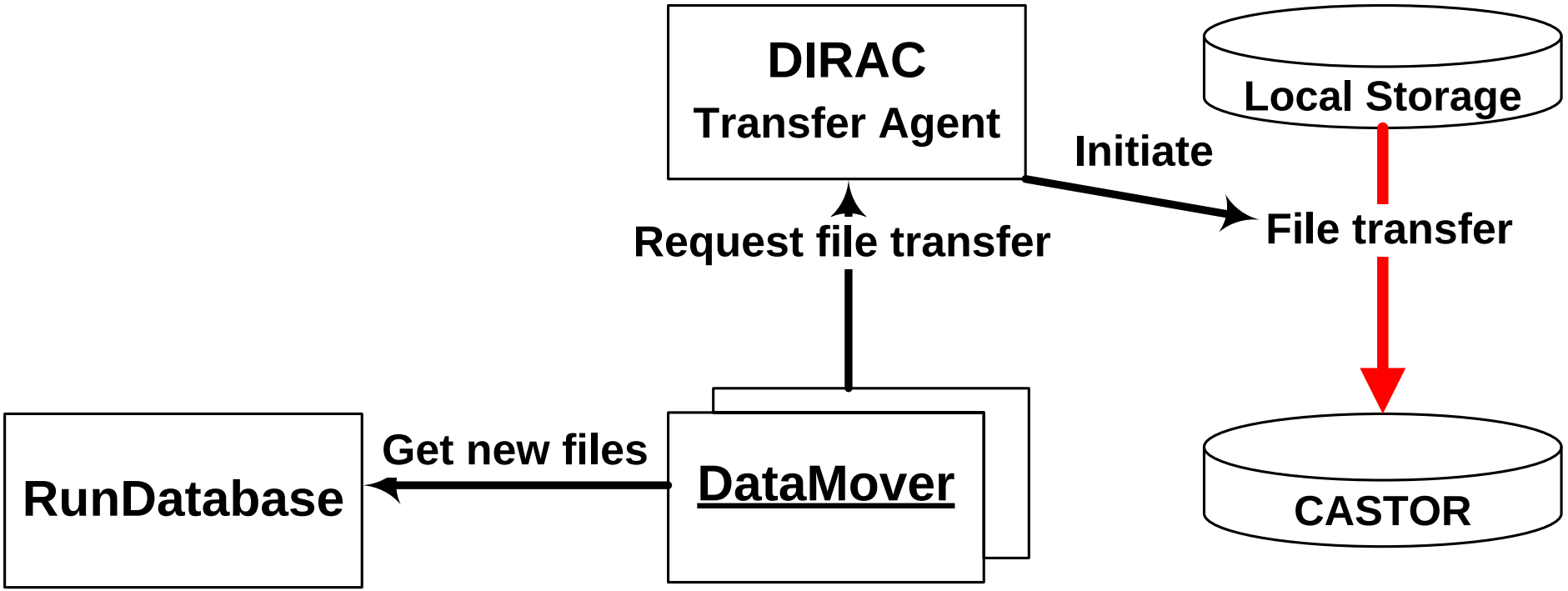
- The Writer processes are receives events by the High Level Trigger (HLT) Farm
- Events a pooled and then written into files
- File names are retrieved from the Run Database via XMLRPC
- When a file is closed, meta data is send via XMLRPC (size, events, etc.)

Storage System: File Transfer



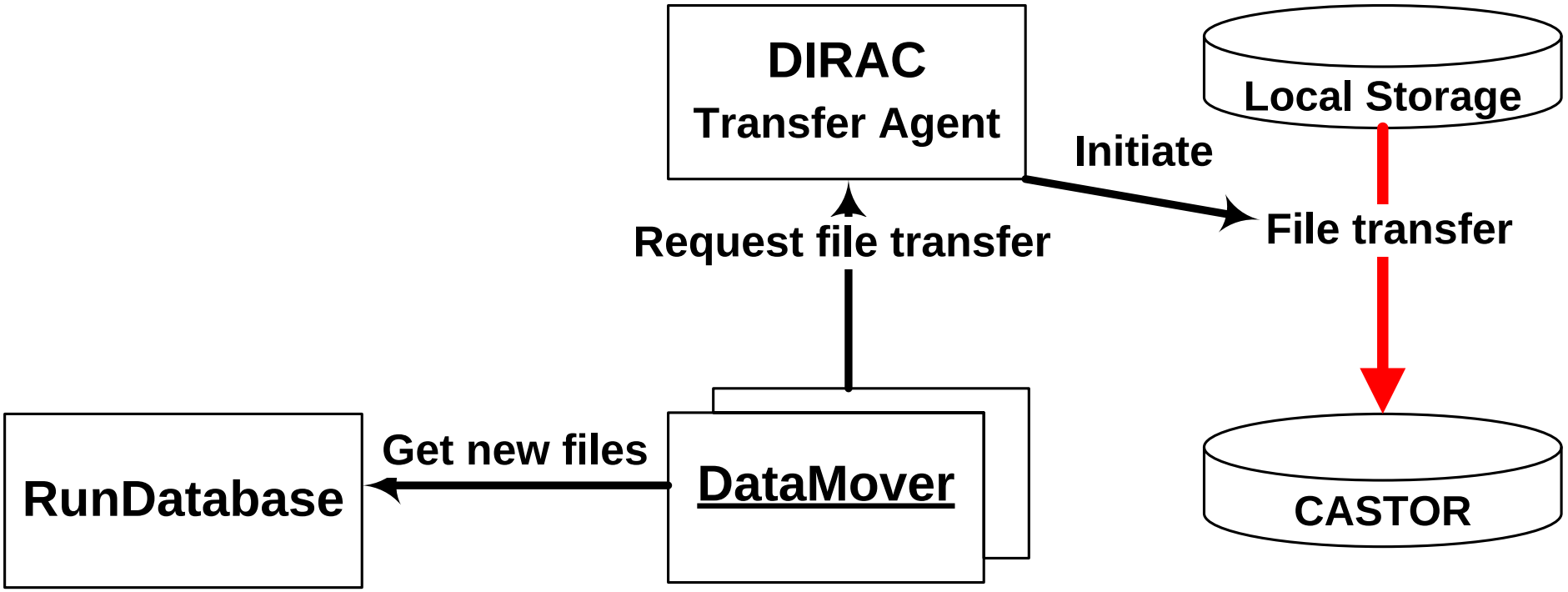
- The Data Mover uses DIRAC, CERN's Grid middleware, to create transfer jobs

Storage System: File Transfer



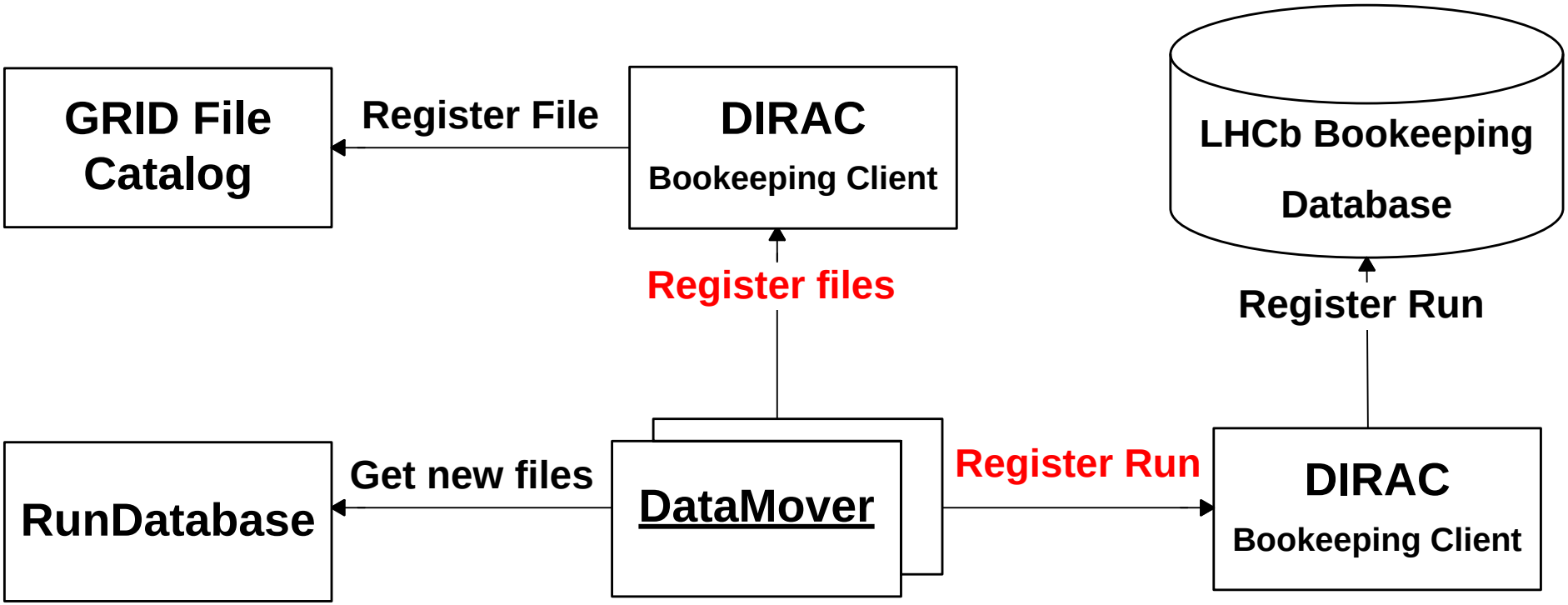
- The Data Mover uses DIRAC, CERN's Grid middleware, to create transfer jobs
- After the transfer the DIRAC transfer agents reply via XMLRPC

Storage System: File Transfer



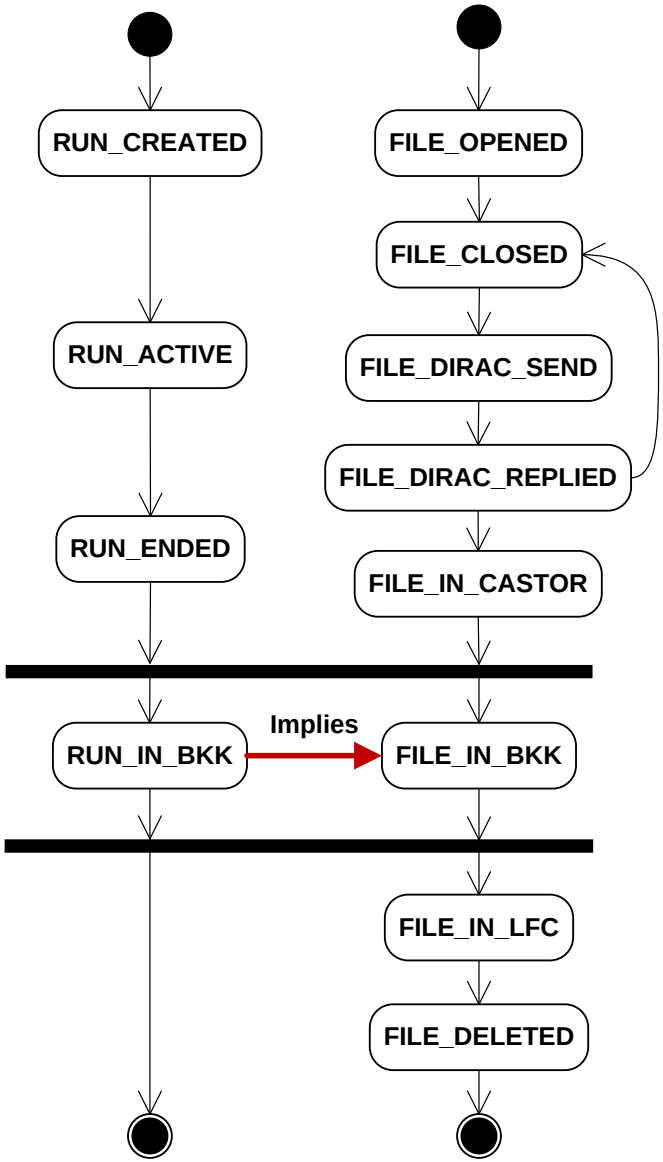
- The Data Mover uses DIRAC, CERN's Grid middleware, to create transfer jobs
- After the transfer the DIRAC transfer agents reply via XMLRPC
- The transfer target is CASTOR (=CERN Advanced Storage Manager) which is CERN's facility for long term storage

Storage System: Bookkeeping and Grid File Catalog



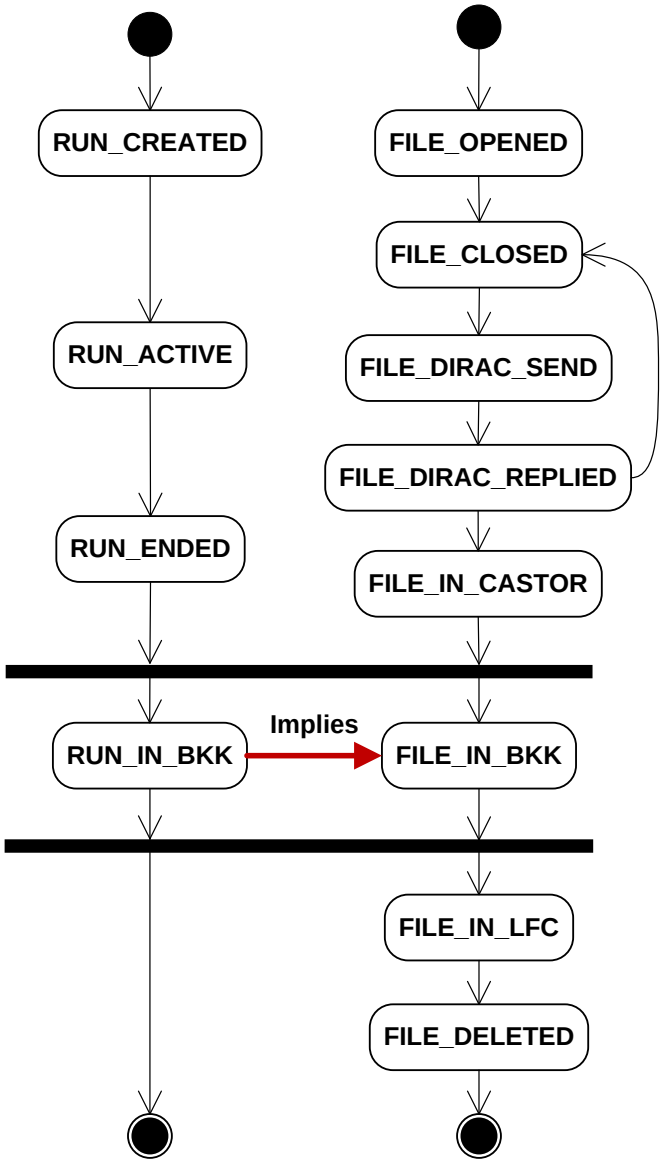
- The Data Mover uses the DIRAC Bookkeeping Client to...
  - ...register runs in LHCb's Bookkeeping database
  - ...register files in the Grid file catalog

Data life cycle



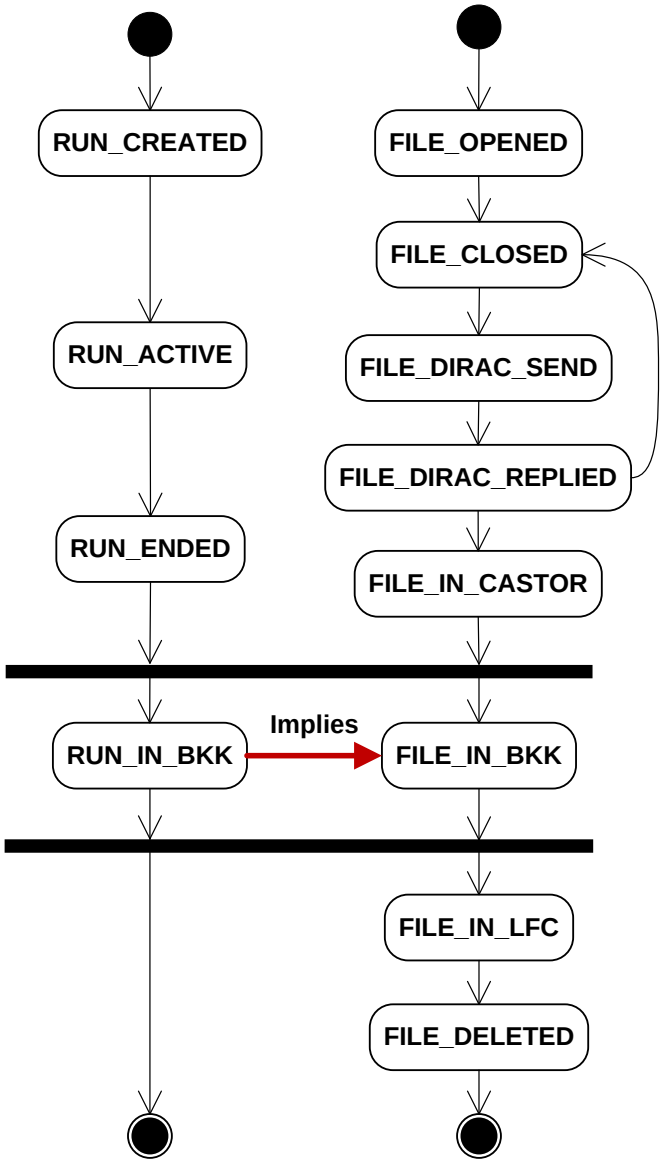
- Run states
  - ECS: RUN_CREATED, RUN_ACTIVE, RUN_ENDED

Data life cycle



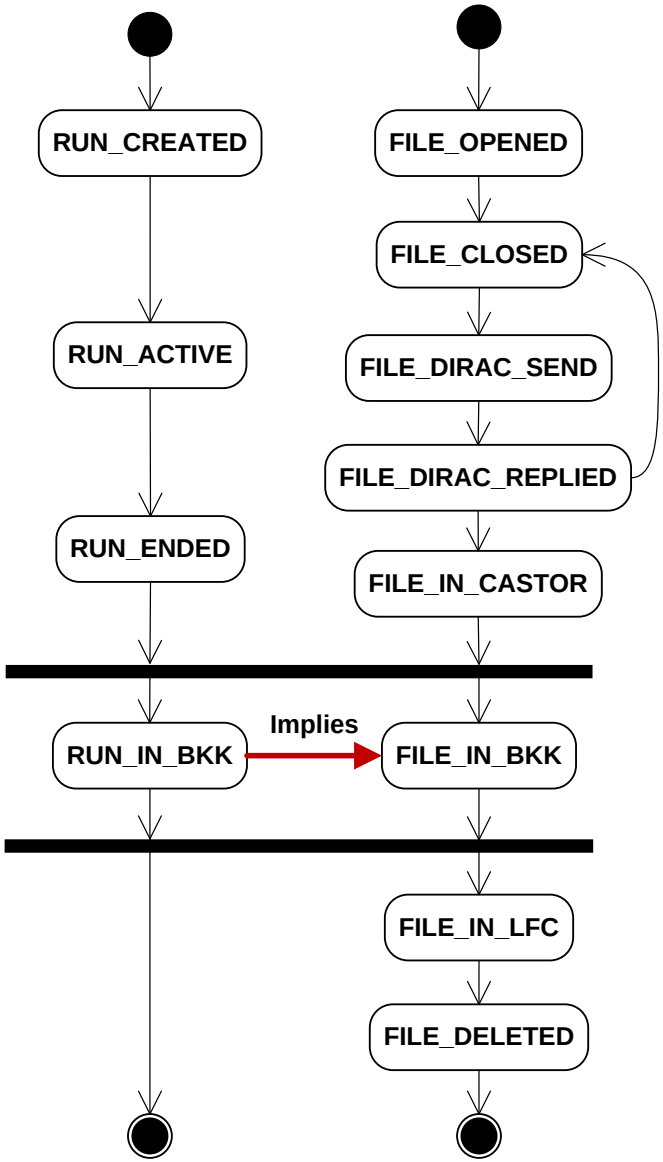
- Run states
  - ECS: RUN_CREATED, RUN_ACTIVE, RUN_ENDED
  - Data Mover: RUN_IN_BKK (Bookkeeping)

Data life cycle



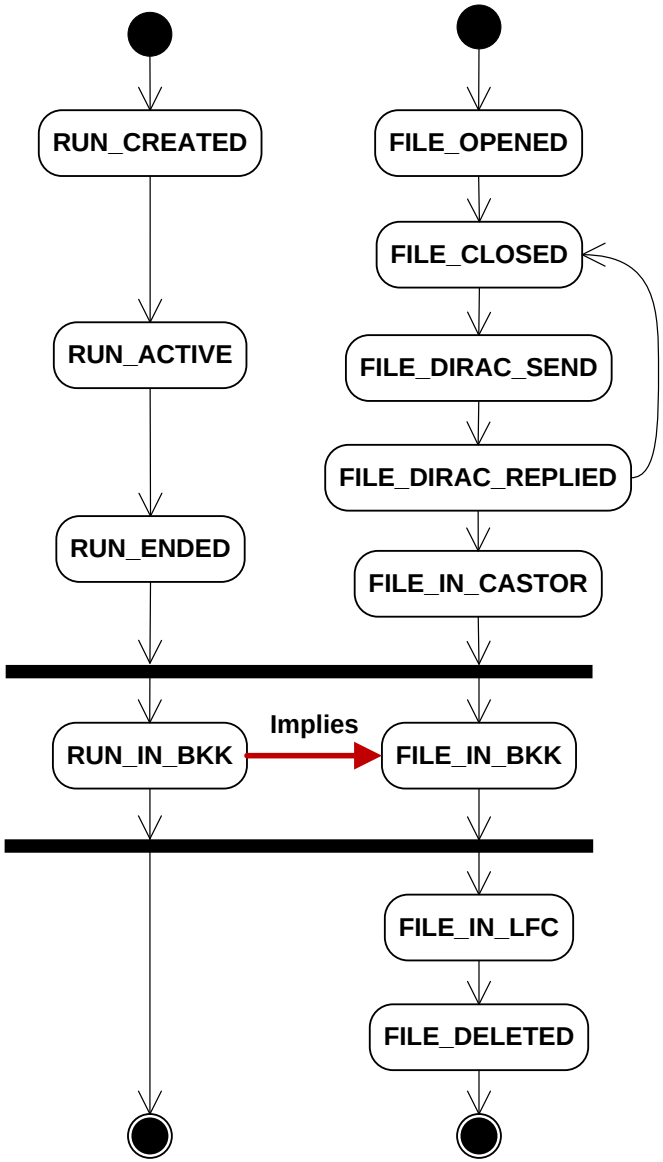
- Run states
  - ECS: RUN_CREATED, RUN_ACTIVE, RUN_ENDED
  - Data Mover: RUN_IN_BKK (Bookkeeping)
- File states
  - Writer Processes: FILE_OPENED, FILE_CLOSED

Data life cycle



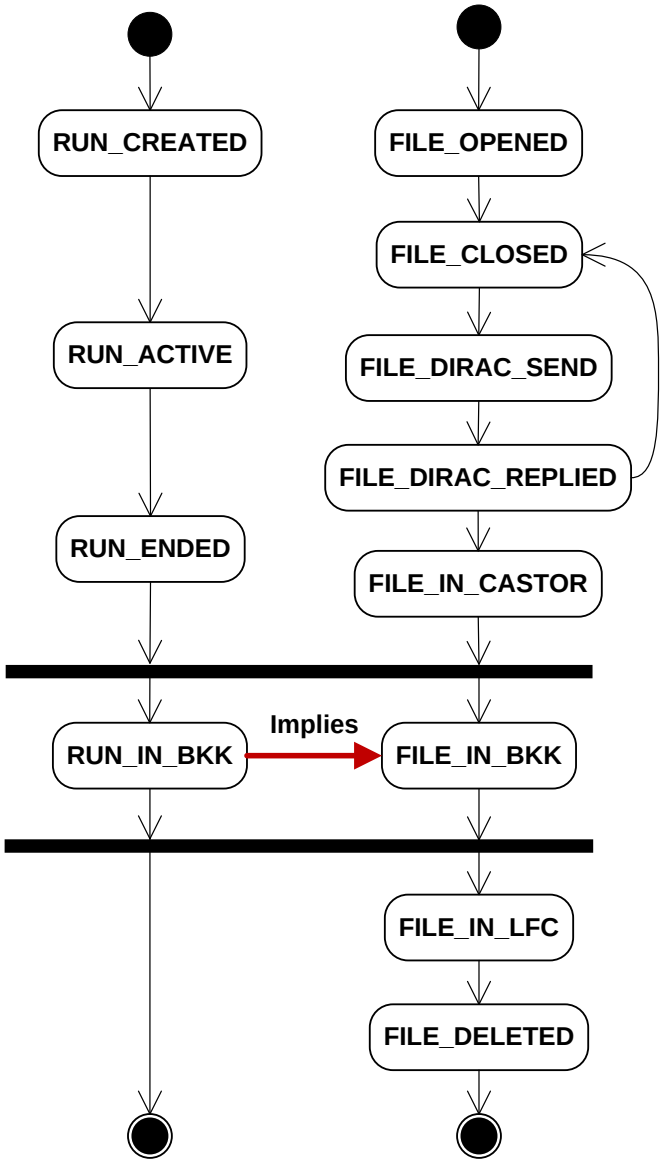
- Run states
  - ECS: RUN_CREATED, RUN_ACTIVE, RUN_ENDED
  - Data Mover: RUN_IN_BKK (Bookkeeping)
- File states
  - Writer Processes: FILE_OPENED, FILE_CLOSED
  - Data Mover:
    - ★ File Transfer: FILE_DIRAC_SEND, FILE_DIRAC_REPLIED, FILE_IN_CASTOR

Data life cycle



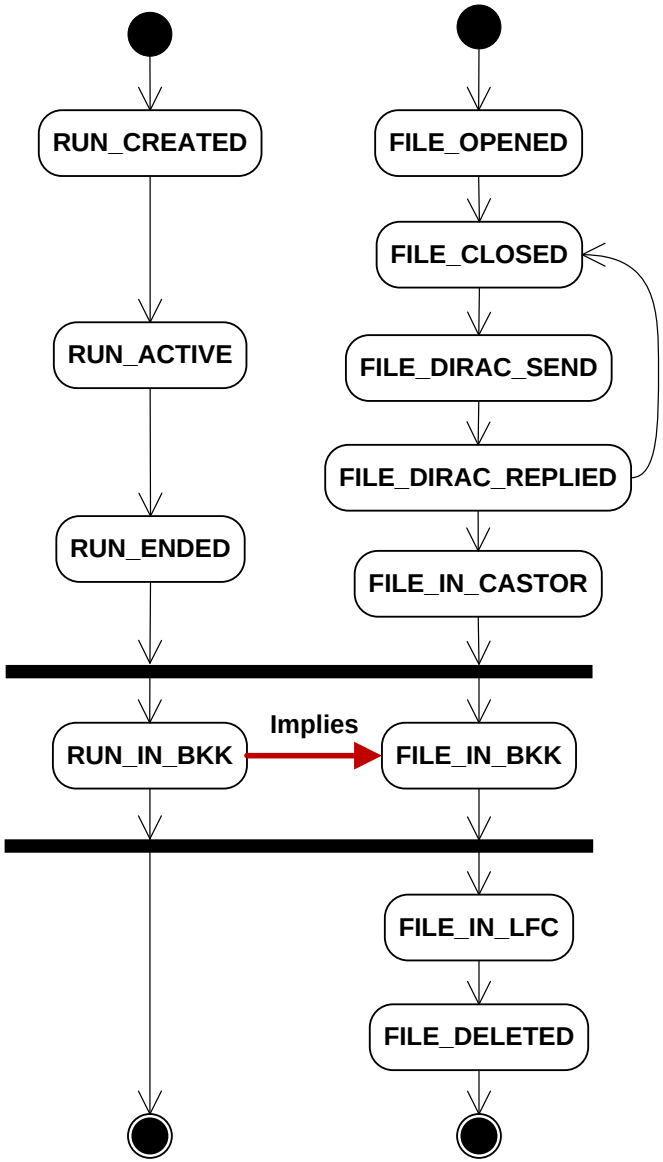
- Run states
  - ECS: RUN_CREATED, RUN_ACTIVE, RUN_ENDED
  - Data Mover: RUN_IN_BKK (Bookkeeping)
- File states
  - Writer Processes: FILE_OPENED, FILE_CLOSED
  - Data Mover:
    - ★ File Transfer: FILE_DIRAC_SEND, FILE_DIRAC_REPLIED, FILE_IN_CASTOR
    - ★ When the run is entered into the Bookkeeping, this includes the files as well

Data life cycle



- Run states
  - ECS: RUN_CREATED, RUN_ACTIVE, RUN_ENDED
  - Data Mover: RUN_IN_BKK (Bookkeeping)
- File states
  - Writer Processes: FILE_OPENED, FILE_CLOSED
  - Data Mover:
    - ★ File Transfer: FILE_DIRAC_SEND, FILE_DIRAC_REPLIED, FILE_IN_CASTOR
    - ★ When the run is entered into the Bookkeeping, this includes the files as well
    - ★ Grid file catalog registration: FILE_IN_LFC

Data life cycle



- Run states
  - ECS: RUN_CREATED, RUN_ACTIVE, RUN_ENDED
  - Data Mover: RUN_IN_BKK (Bookkeeping)
- File states
  - Writer Processes: FILE_OPENED, FILE_CLOSED
  - Data Mover:
    - ★ File Transfer: FILE_DIRAC_SEND, FILE_DIRAC_REPLIED, FILE_IN_CASTOR
    - ★ When the run is entered into the Bookkeeping, this includes the files as well
    - ★ Grid file catalog registration: FILE_IN_LFC
    - ★ File Deletion: FILE_DELETED

Reliable Copying - Design of the Data Mover

- Implementation in Python
  - Fast development cycle (compared to C++)
  - Bindings to all other LHCb systems exist (DIRAC, DIM, XMLRPC)

Reliable Copying - Design of the Data Mover

- Implementation in Python
  - Fast development cycle (compared to C++)
  - Bindings to all other LHCb systems exist (DIRAC, DIM, XMLRPC)
- Data Mover has one class with several methods
- Each method runs in its own instance of a Data Mover
  - Complete separation of the tasks
  - Improved performance by parallelism
  - Easy control of many processes

Reliable Copying - Design of the Data Mover

- Implementation in Python
  - Fast development cycle (compared to C++)
  - Bindings to all other LHCb systems exist (DIRAC, DIM, XMLRPC)
- Data Mover has one class with several methods
- Each method runs in its own instance of a Data Mover
  - Complete separation of the tasks
  - Improved performance by parallelism
  - Easy control of many processes
- Use of an abstraction layer to access the database

Interfaces to the Run Database Abstraction Layer

- DIMRPC (= Distributed Information Management System)
  - Used because the Experiment Control System uses it
  - DIM is a interprocess communication layer
  - DIM makes the network structure transparent

Interfaces to the Run Database Abstraction Layer

- DIMRPC (= Distributed Information Management System)
  - Used because the Experiment Control System uses it
  - DIM is a interprocess communication layer
  - DIM makes the network structure transparent
- XMLRPC (= XML Remote Procedure Call)
  - Used for the Communication with DIRAC and the writer processes
  - XMLRPC libraries exist for all modern programming languages
  - Is based on XML as message structure and HTTP for transport
  - Can be easily traced with a network sniffer (like Wireshark)

Interfaces to the Run Database Abstraction Layer

- DIMRPC (= Distributed Information Management System)
  - Used because the Experiment Control System uses it
  - DIM is a interprocess communication layer
  - DIM makes the network structure transparent
- XMLRPC (= XML Remote Procedure Call)
  - Used for the Communication with DIRAC and the writer processes
  - XMLRPC libraries exist for all modern programming languages
  - Is based on XML as message structure and HTTP for transport
  - Can be easily traced with a network sniffer (like Wireshark)
- Introspection
  - Used to find all public methods
  - Automatically done on every startup of the XMLRPC- and DIMRPC-Servers
  - Servers are always providing the latest version of the Run Database interface
  - No maintenance in the servers necessary

Implementation of the Run Database Abstraction Layer

- Written in Python
- Use of SQLAlchemy[1] , which is a Python SQL Toolkit
  - Table and Column definitions are given within normal Python code
  - Ability to sync table definitions with the database
  - No SQL queries: SELECTS are build with SQLAlchemy constructs

[1] sqlalchemy.org

Reliability

- Needs for reliability
  - All components of the system are still under development
  - Runs are sometimes stopped "unfriendly" (process just killed, no clean up)
  - Files are sometimes not closed (and so the final meta is missing)
  - When a writer process dies some meta date is missing
  - Before the files in the local storage system can be transfered and registered, all meta data has to exist

Reliability: Realization

- Stability
  - Methods of the RunDatabase are fault tolerant
  - Heavy use of exceptions and exception handling
  - The RunDatabase and the DataMover processes shall not crash on any circumstances

Reliability: Realization

- Stability
  - Methods of the RunDatabase are fault tolerant
  - Heavy use of exceptions and exception handling
  - The RunDatabase and the DataMover processes shall not crash on any circumstances
- Automatic check and repair methods
  - Most frequently used: Automatic file closing
  - Also ending of runs which have been started weeks ago
  - Most of the meta data can be found later (file size, file paths)
  - Some meta data can be recalculated later (e.g. number of events in the file by reprocessing it, this is in preparation)

Reliability: Realization

- Stability
  - Methods of the RunDatabase are fault tolerant
  - Heavy use of exceptions and exception handling
  - The RunDatabase and the DataMover processes shall not crash on any circumstances
- Automatic check and repair methods
  - Most frequently used: Automatic file closing
  - Also ending of runs which have been started weeks ago
  - Most of the meta data can be found later (file size, file paths)
  - Some meta data can be recalculated later (e.g. number of events in the file by reprocessing it, this is in preparation)
- Tools (command line, web interface) to check entries in the Run Database

Reliability: Realization

- Stability
  - Methods of the RunDatabase are fault tolerant
  - Heavy use of exceptions and exception handling
  - The RunDatabase and the DataMover processes shall not crash on any circumstances
- Automatic check and repair methods
  - Most frequently used: Automatic file closing
  - Also ending of runs which have been started weeks ago
  - Most of the meta data can be found later (file size, file paths)
  - Some meta data can be recalculated later (e.g. number of events in the file by reprocessing it, this is in preparation)
- Tools (command line, web interface) to check entries in the Run Database
- Retries and cross-checks
  - Failed tasks will be retried (need for retries e.g. when a network was down or a server was not running)
  - Before the transition to the next task an intensive check is done

Additional logging into a database

- Standard logging is done via the Linux syslog daemon
- for each run or file and each state there is one entry
  - Contains date of last action
  - In error cases it gives a description of the error
- There exists a command line tool accessible for administrators and users to retrieve these entries.
- Very fast track of the history
- Can be much faster scanned than log files which are probably moved by log rotation

Numbers of Data handled

- $\approx 40K$ Runs

Numbers of Data handled

- $\approx 40K$ Runs
- $\approx 100K$ Files

Numbers of Data handled

- $\approx 40K$ Runs
- $\approx 100K$ Files
- $\approx 2.7Bn$ Events

Numbers of Data handled

- $\approx 40K$ Runs
- $\approx 100K$ Files
- $\approx 2.7Bn$ Events
- $\approx 140\,TiB$ Total data, $\approx 98\,TiB$ which has been moved to long term storage

Conclusion

Main Aspects for reliable online data-replication:

- Use of grid middleware for all tasks (DIRAC)
- Simple design, small components, dedicated for a single task
- Fault tolerance
- Retries
- Automatic check and repair routines
- Detailed logging, tools for dealing with the logs