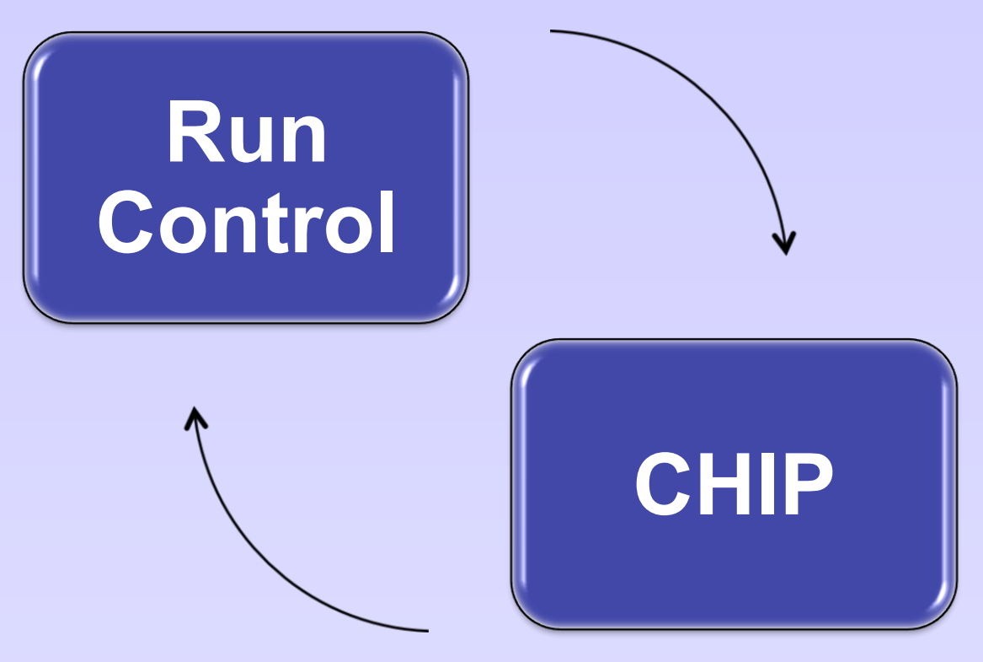
Intelligent operations of the data acquisition system of the ATLAS experiment at the LHC

G. Anders, G. Avolio, G. Lehmann Miotto, L. Magnoni

CERN, Geneva, Switzerland

1. Introduction



The **Trigger and Data Acquisition**[1] (TDAQ) system of the **ATLAS**[2] detector at the Large Hadron Collider (LHC) at CERN is composed of a large number of distributed hardware and software components (about 2000 machines and more than 15000 concurrent processes at the end of LHC's Run I) which in a coordinated manner provide the data-taking functionality of the overall system.

The **Run Control** (RC) and the **Central Hint and Information Processor** (CHIP) are key components of the **Online Software** framework that encompasses the software to configure, control and monitor the TDAQ system.

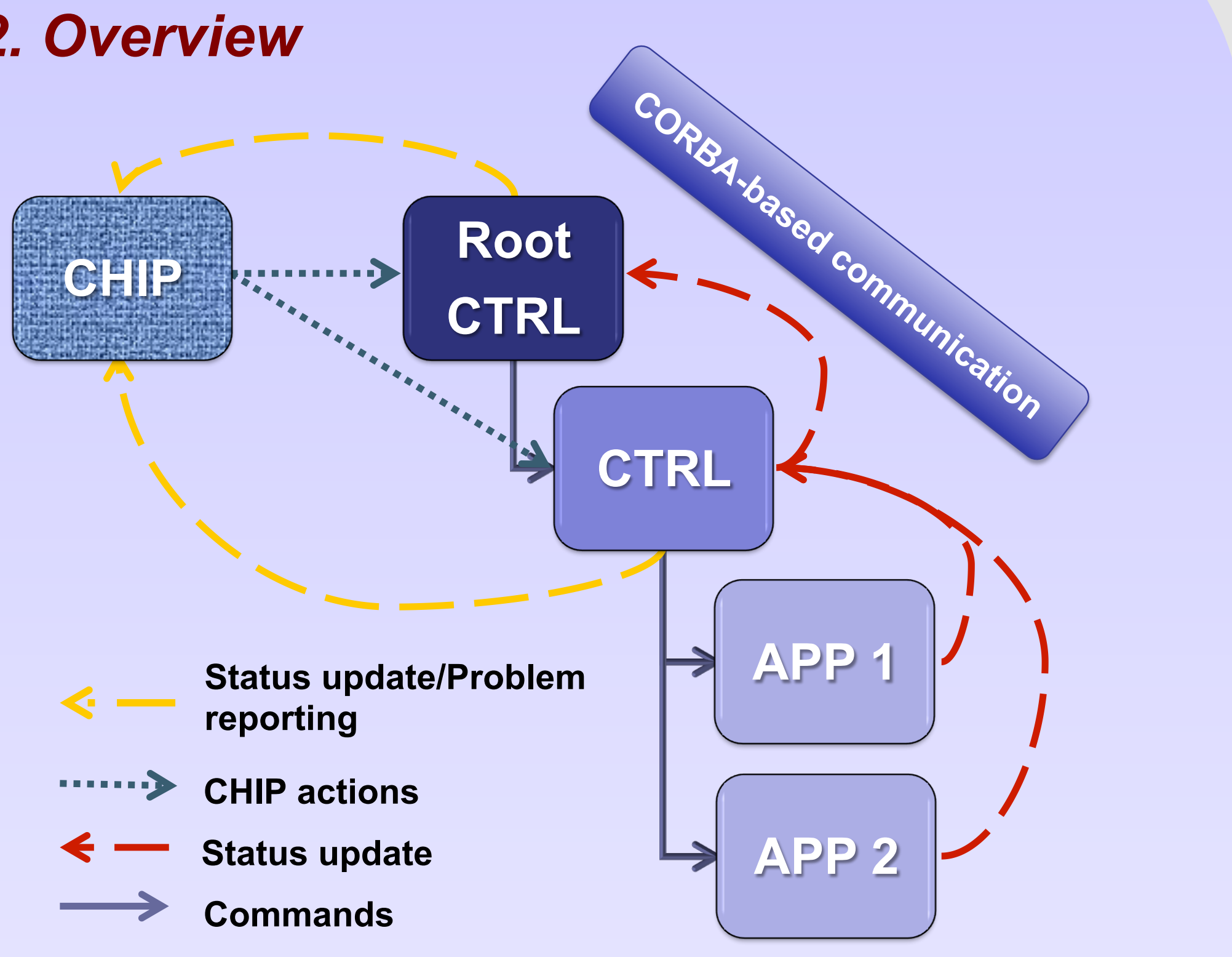
The RC system steers the data acquisition by starting and stopping processes and by carrying all data-taking elements through well-defined states in a coherent way. Given the size and complexity of the TDAQ system, errors and failures are bound to happen and must be dealt with. The data acquisition system has to **recover from these errors promptly and effectively**, possibly without the need to stop data taking operations. During LHC Run 1, the detection and handling of problems was based on an embedded rule-based forward-chaining expert system (CLIPS), which was deeply integrated with the RC system. Even though the system performed well, it had major disadvantages: new rules could not be tested without reproducing the error conditions in the real TDAQ environment and monitoring of system resources used by specific rules was not possible. This made the development and debugging of new rules difficult. Additionally, the expert system was lacking the natural support for detections of temporal patterns.

During the **LHC Long Shutdown 1** (LS1) the RC has been completely re-designed and re-implemented in order to address the new requirements mentioned beforehand. The new RC is assisted by the CHIP that can be truly considered as its "brain". CHIP is an **intelligent system** having a global view on the TDAQ system. It supervises the ATLAS data taking, takes operational decisions and handles abnormal conditions. Furthermore CHIP automates complex procedures and performs advanced recoveries.

2. Overview

Applications in the ATLAS TDAQ system are organized in a tree-like hierarchical structure (the **run control tree**), where each application is managed by a parent **Controller**. The topmost node of the tree is the **Root Controller**. Controller applications are responsible for keeping the system in a coherent state by starting and stopping their child applications and by sending them the proper commands needed to reach a state suitable for data-taking.

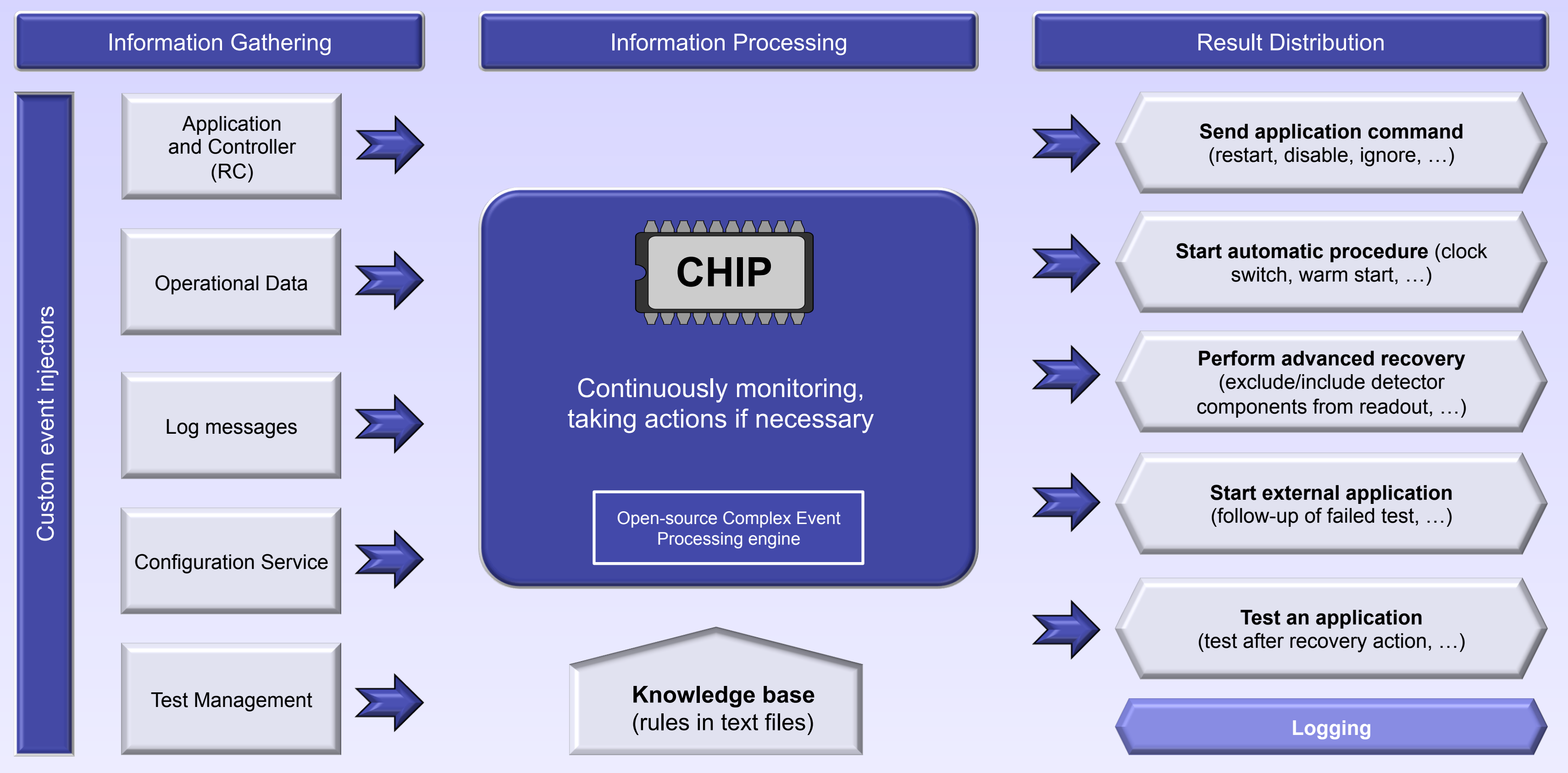
Operations across the run control tree are synchronized using **Finite State Machine** (FSM) principles. FSM transitions are usually initiated by the human operator via a graphical user interface: commands are sent directly to the Root Controller and then automatically propagated throughout the tree by intermediate controllers. Once an application completes the execution of a command (or changes its internal status by any reason) it notifies the parent controller which in this way can evaluate when a coherent state is reached.



Moreover, controller applications are the **RC elements interacting with CHIP**. Controllers inform CHIP about any change in their own status or in the status of their controlled children. CHIP, in its turn, is able to detect any anomaly in the system analyzing the status of all the applications and can notify the controllers about actions to be taken in order to resolve the issue. Examples of actions are setting a simple error flag or restarting/ignoring offending applications.

3. CHIP - Architecture

CHIP is an application which gathers information from various sources and employs an open-source **Complex Event Processing (CEP)** engine in order to aggregate, correlate and analyze this information. Besides using RC information, CHIP accesses **the logging service, the operational data service** and **the configuration service**. Furthermore it interacts with the so-called **Test Management service**. Amongst others this allows for detecting DAQ problems originating from hardware or network failures. The knowledge base consists of a set of rules loaded from text files.


Information Gathering
Information Processing
Result Distribution
Custom event injectors
Application and Controller (RC)
Operational Data
Log messages
Configuration Service
Test Management
CHIP
Continuously monitoring, taking actions if necessary
Open-source Complex Event Processing engine
Knowledge base (rules in text files)
Send application command (restart, disable, ignore, …)
Start automatic procedure (clock switch, warm start, …)
Perform advanced recovery (exclude/include detector components from readout, …)
Start external application (follow-up of failed test, …)
Test an application (test after recovery action, …)
Logging

5. CHIP – Why using a CEP engine?

A CEP engine was chosen as the core of CHIP, because it allows for:

1. Efficient handling of very high information update rates
- peak rate typically several tens of thousands of events per second

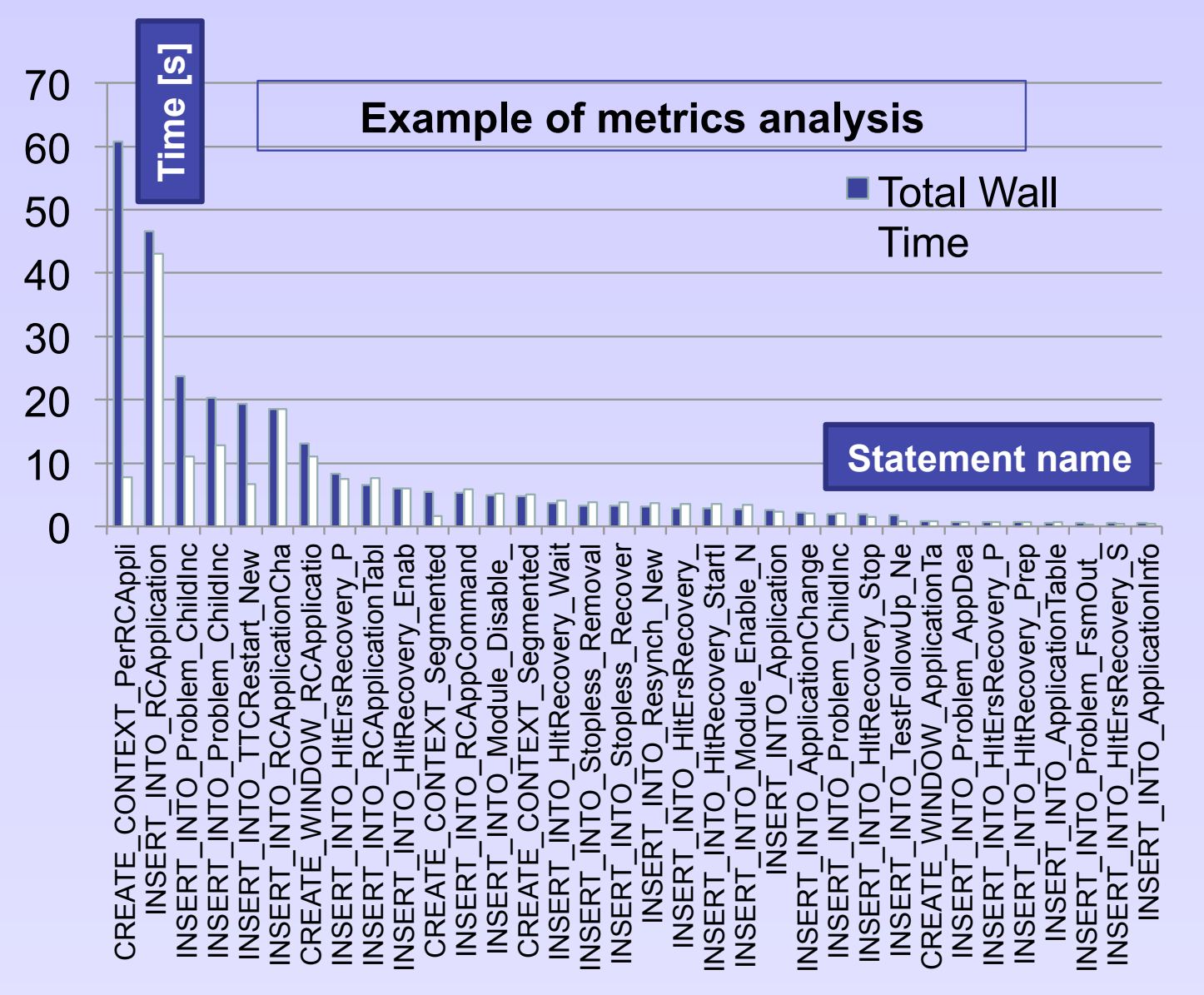
2. Rule testing
- correct logic of new rules can be verified by artificial injection of events in a unit test

3. Metrics analysis
- monitor CPU usage of individual rules
- CPU intensive rules can be revised

4. Configuration of threading model

5. Natural support for temporal correlations
- e.g. for detecting frequently failing applications

6. Sophisticated anomaly detection
- CHIP is prepared for sophisticated anomaly detection, since the CEP engine is well-suited for complex correlations of the various information providers.



CHIP has rules for about **20 different kind of recoveries**
- some of which are very generic (e.g. for crashed applications)
- others are sophisticated (e.g. recovering failing high level trigger applications)

Additionally, **CHIP automates about 10 procedures**, e.g.:
- raise of high voltage of detector components when collisions are imminent and beam conditions are safe
- switch of main ATLAS clock from internal to LHC one depending on current detector and beam conditions

Required resources
- CHIP runs on a single dedicated multi-core machine (16 cores, 24 GB memory) and typically uses about 1GB of memory. The CPU utilization follows the incoming event rate and typically varies between <1% and 300% (i.e. the equivalent of 3 CPU cores out of 16)
- The responsiveness of CHIP was verified in tests during which the information update peak rate was at least twice as high as expected for LHC Run 2

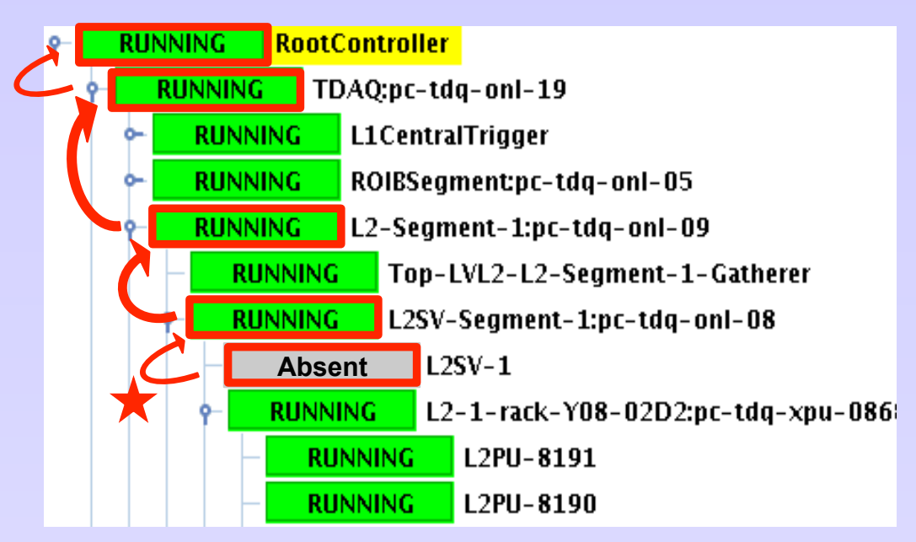
4. CHIP – Rule Examples

Example for simple rule (crashed application)

For demonstration purposes the shown rules were simplified with respect to the ones in the production system.

Detect application problem

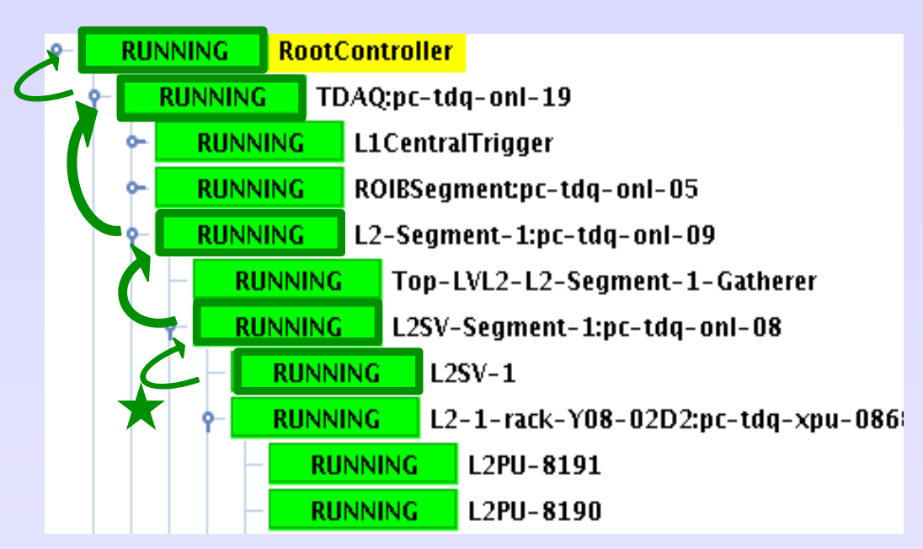
```
on RCApplication(name != 'RootController',
   application.status = STATUS.ABSENT,
   application.membership = MEMBERSHIP._IN)
   as application
insert into Problem(controller, application, type, state)
select application.controller,
   application.name,
   Problem$TYPE.APPLICATION_DEAD,
   Problem$STATUS._NEW;
```


RUNNING RootController
RUNNING TDAQ:pc-tdq-onl-19
RUNNING L1CentralTrigger
RUNNING ROIBSegment:pc-tdq-onl-05
RUNNING L2-Segment-1:pc-tdq-onl-09
RUNNING Top-LVL2-L2-Segment-1-Gatherer
RUNNING L2SV-Segment-1:pc-tdq-onl-08
Absent L2SV-1
RUNNING L2-1-rack-Y08-02D2:pc-tdq-xpu-086
RUNNING L2PU-8191
RUNNING L2PU-8190

CHIP detects when an application crashes and reacts according to the application specific configuration. Here the parent controller is put into error state. Another rule is responsible for propagating the error flag up the Run Control hierarchy.

Reset application problem

```
insert into Problem(controller, application, type, state)
select application.controller,
      application.name,
      Problem$TYPE.APPLICATION_DEAD,
      Problem$STATUS.RESOLVED
from pattern [ every (error=Problem(
      type=Problem$TYPE.APPLICATION_DEAD,
      state=Problem$STATUS.HANDLING))
      -> application = RCApplication(
      name=error.application,
      status = STATUS.UP,
      membership=MEMBERSHIP._IN)];
```

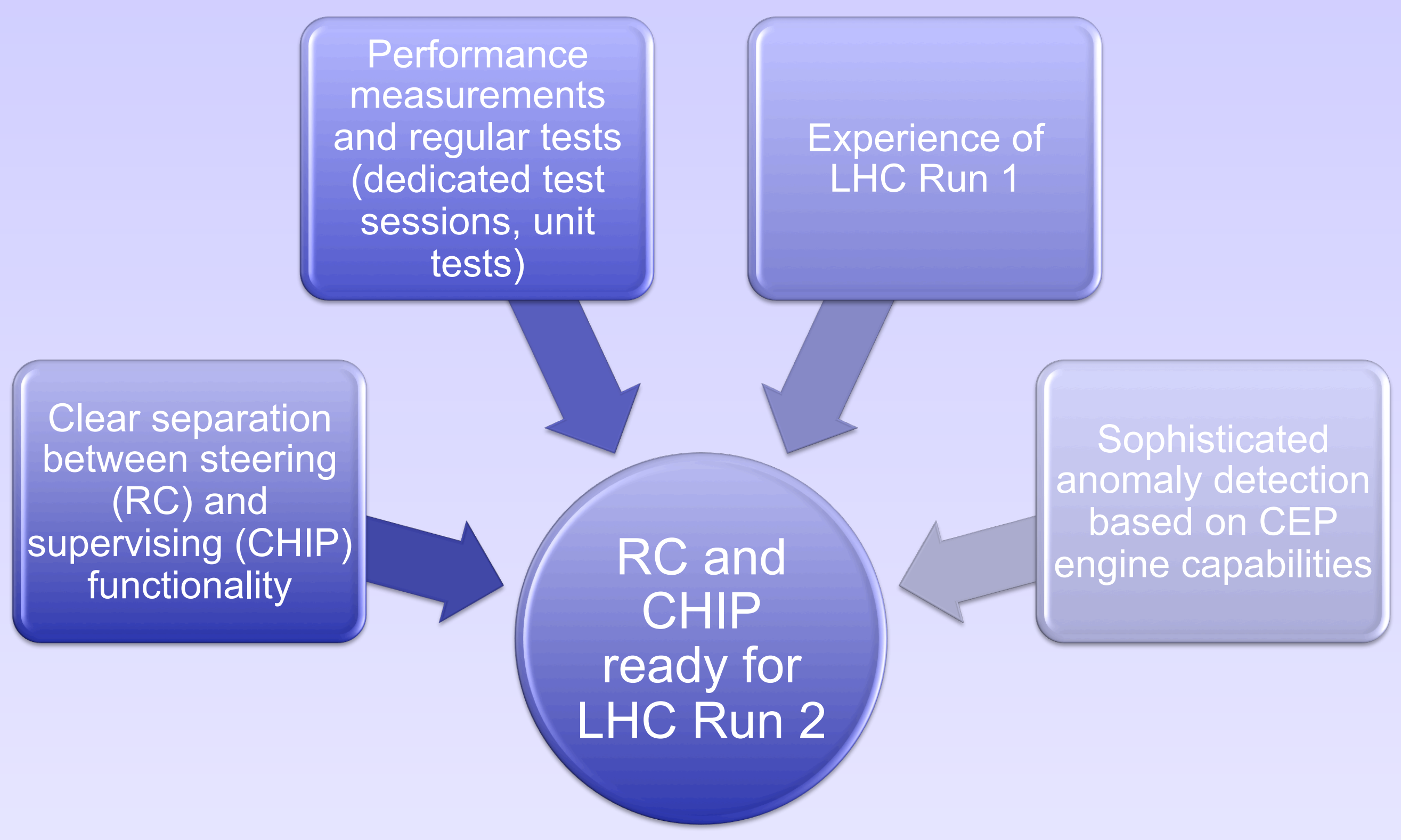

RUNNING RootController
RUNNING TDAQ:pc-tdq-onl-19
RUNNING L1CentralTrigger
RUNNING ROIBSegment:pc-tdq-onl-05
RUNNING L2-Segment-1:pc-tdq-onl-09
RUNNING Top-LVL2-L2-Segment-1-Gatherer
RUNNING L2SV-Segment-1:pc-tdq-onl-08
RUNNING L2SV-1
RUNNING L2-1-rack-Y08-02D2:pc-tdq-xpu-086
RUNNING L2PU-8191
RUNNING L2PU-8190

When the crashed application is up again, either due to automatic recovery or manual intervention, CHIP resets the internal problem state of the application. This triggers the clearance of the error flag of the parent controller, and subsequently of all other affected controllers in the RC hierarchy.

Example for sophisticated recovery (stopless removal)

In case the data taking is blocked due to a faulty sub-detector component, CHIP can dynamically disable the corresponding read-out links **without the need for stopping the data taking session**. When the sub-detector issue has been resolved, CHIP can **re-enable** the corresponding read-out components and integrate them back. The advantages of this procedure are two-fold:
- the recorded data may still be suited for physics analyses, depending on the disabled sub-detector channels
- the down-time of the DAQ system caused by this procedure is much smaller than the down-time caused by a stop and restart of the complete data taking session. **Thus less integrated luminosity is lost for physics analyses.**

6. Conclusions


Performance measurements and regular tests (dedicated test sessions, unit tests)
Experience of LHC Run 1
Clear separation between steering (RC) and supervising (CHIP) functionality
Sophisticated anomaly detection based on CEP engine capabilities
RC and CHIP ready for LHC Run 2

References
1. The ATLAS Collaboration, 2002, *ATLAS high-level trigger, data-acquisition and controls: Technical Design Report*
2. The ATLAS Collaboration, 2008, *The ATLAS Experiment at the CERN Large Hadron Collider*, J. Instrum. 3
3. EsperTECH, http://esper.codehaus.org/ *(August 2014)*