The Run Control System and the Central Hint and Information Processor of the Data Acquisition System of the ATLAS Experiment at the LHC

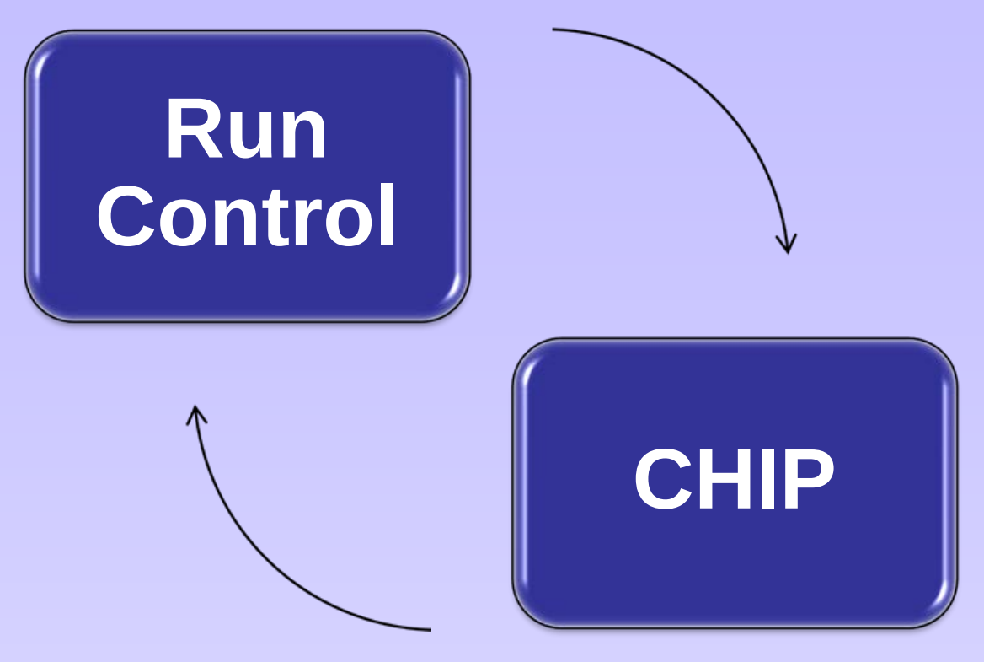
G. Anders, G. Avolio, G. Lehmann Miotto, L. Magnoni

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 3000 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**[3] framework that encompasses the software to configure, control and monitor the TDAQ system.



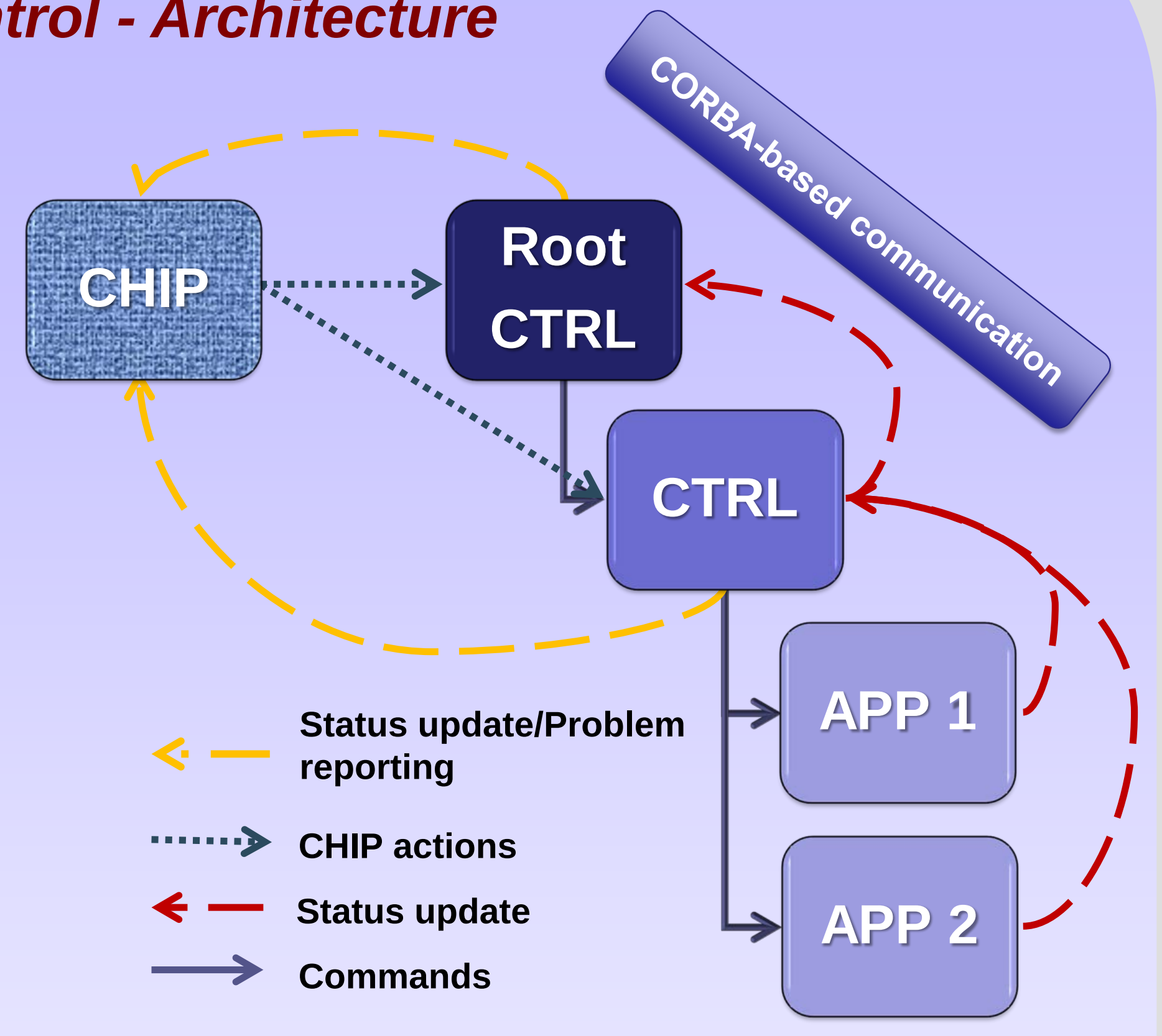
The RC system steers the data acquisition by starting and stopping processes and by carrying all data-taking elements through well-defined states in a coherent way. During the **LHC Long Shutdown 1** (LS1) the RC has been completely re-designed and re-implemented in order to better fulfill the new requirements which emerged during the LHC Run 1 and were not foreseen during the initial design phase.

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. That's why the RC is assisted by the CHIP that can be considered as its "brain". CHIP supervises the ATLAS data taking, takes operational decisions and handles abnormal conditions. It automates procedures and performs advanced recoveries.

2. The Run Control - Architecture

Applications in the ATLAS TDAQ systems 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 to keep 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.

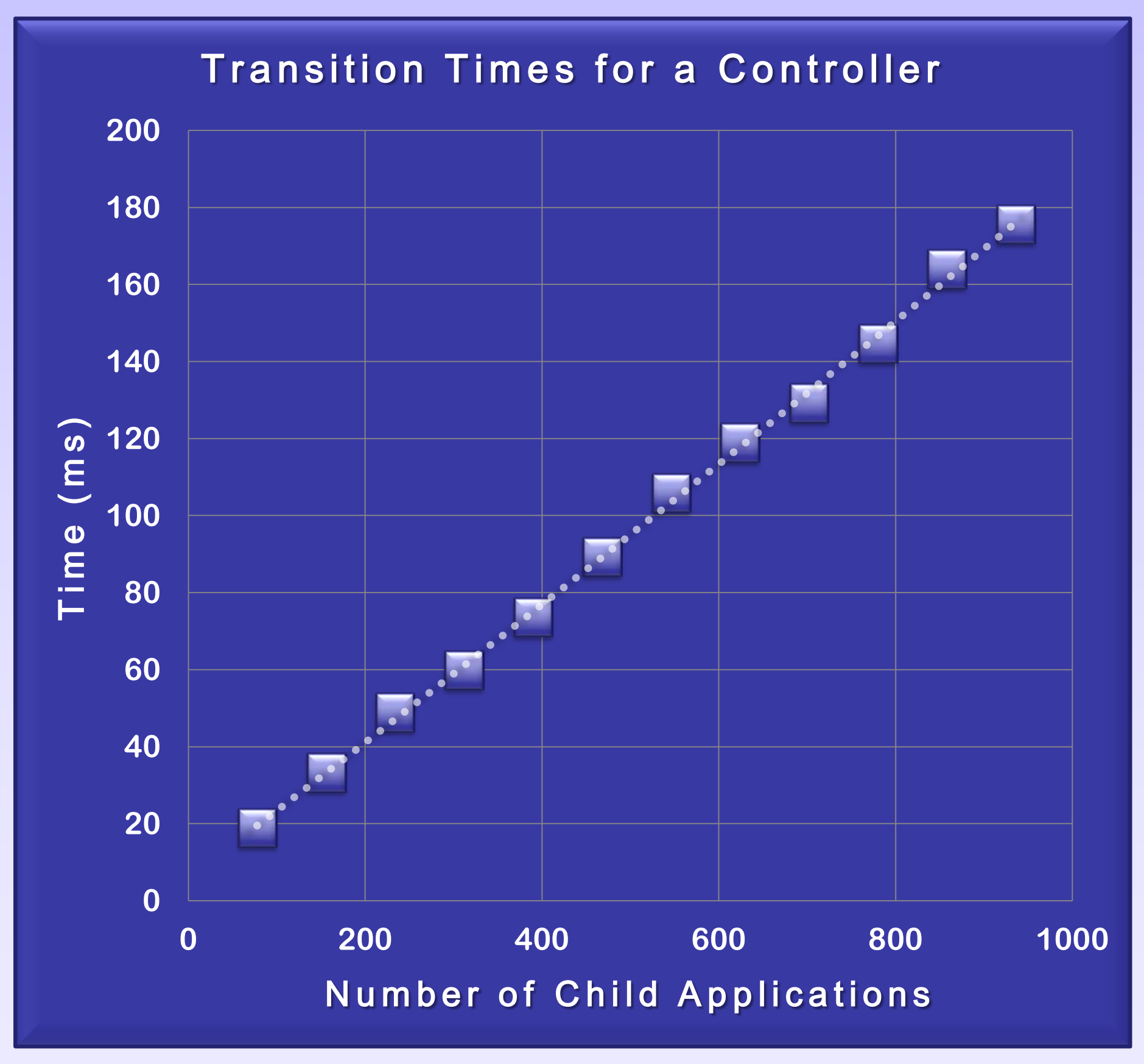
Operations across the run control tree are synchronized using **Finite State Machine** (FSM) principles. FSM transitions are usually initiated by the human operator via a graphical user interface: commands are sent directly to the Root Controller and then automatically propagated throughout the tree 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. It is also possible for controllers to directly report problems to CHIP in very well defined scenarios.

3. The Run Control - Controller Performances

From a performance point of view it is important **to keep low the overhead** introduced by the RC system in dispatching commands and receiving their acknowledgments. In order to evaluate such an overhead, the time needed by a controller application to fully perform an FSM state transition is measured as a function of the number of child applications.



The plot shows the time needed by a controller to perform an FSM state transition as a function of the number of child applications (evenly distributed on a rack of 39 nodes). Child applications are configured to have a zero burning time during state transitions (*i.e.,* they just receive commands from the parent controller and notify it when the command execution starts or completes).

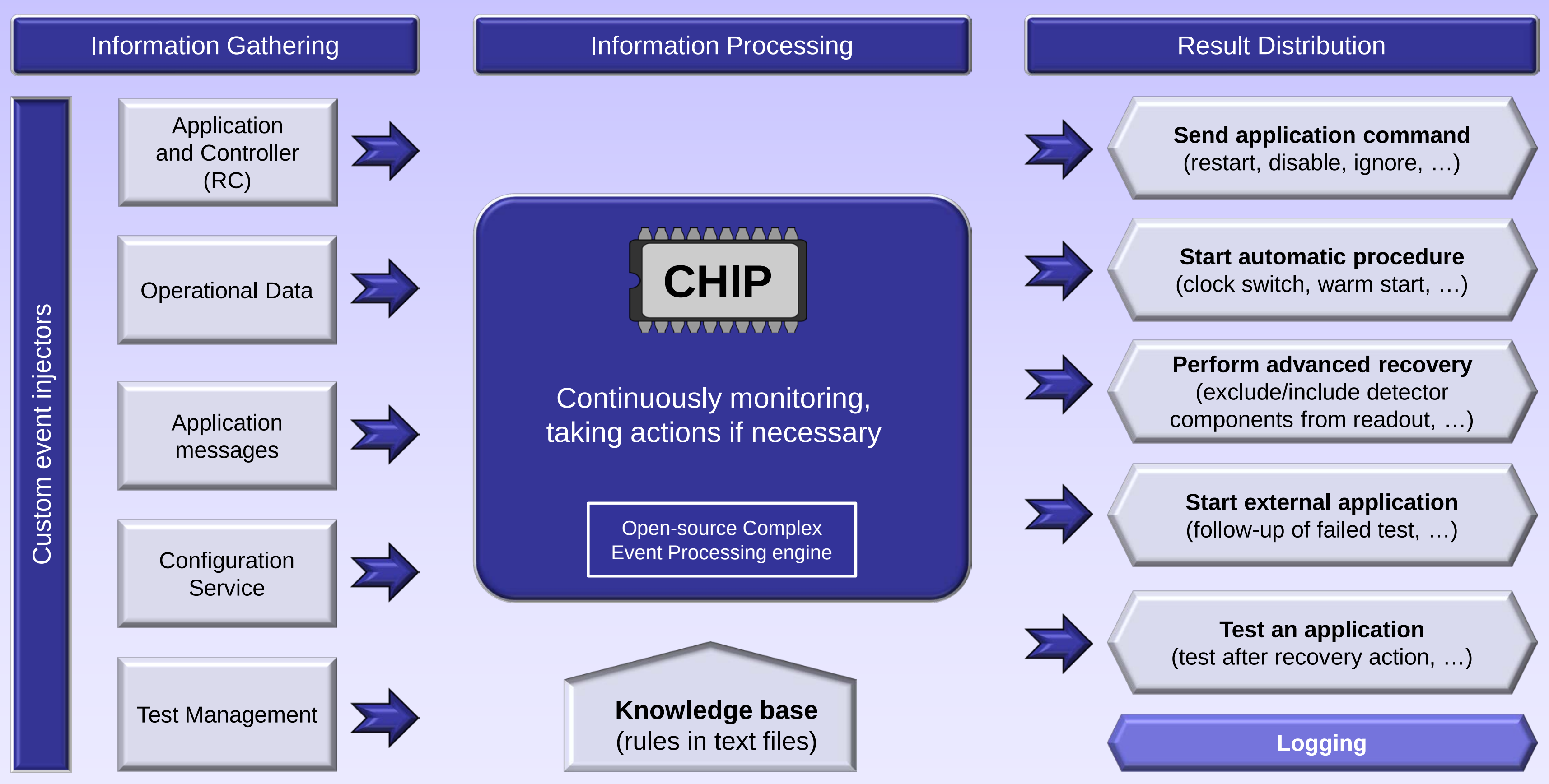
With **936 child applications** the time needed to perform a state transition is less than **180 ms**. Taking into account that transition actions performed by real-life applications during physics runs take tens of seconds and that during LHC Run 2 a single controller will supervise O(100) children, the controller's performance is considered fully satisfactory.

The observed linear scaling is somehow expected given the high number of child applications with respect to the available HW concurrency.

Tests have been executed on nodes equipped with two Intel Xeon E5645 CPUs, 24 GB of RAM and Gb link connection.

4. CHIP - Architecture

The **CHIP** is an application which gathers information from various sources and employs an open-source **Complex Event Processing engine** in order to aggregate, correlate and analyze this information. Furthermore it has the possibility to interact with the so-called Test Management service which allows it to make informed decisions based on the outcome of the test results.


Information Gathering
Information Processing
Result Distribution
Custom event injectors
Application and Controller (RC)
Operational Data
Application 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 – Rule Testing and Performance Profiling

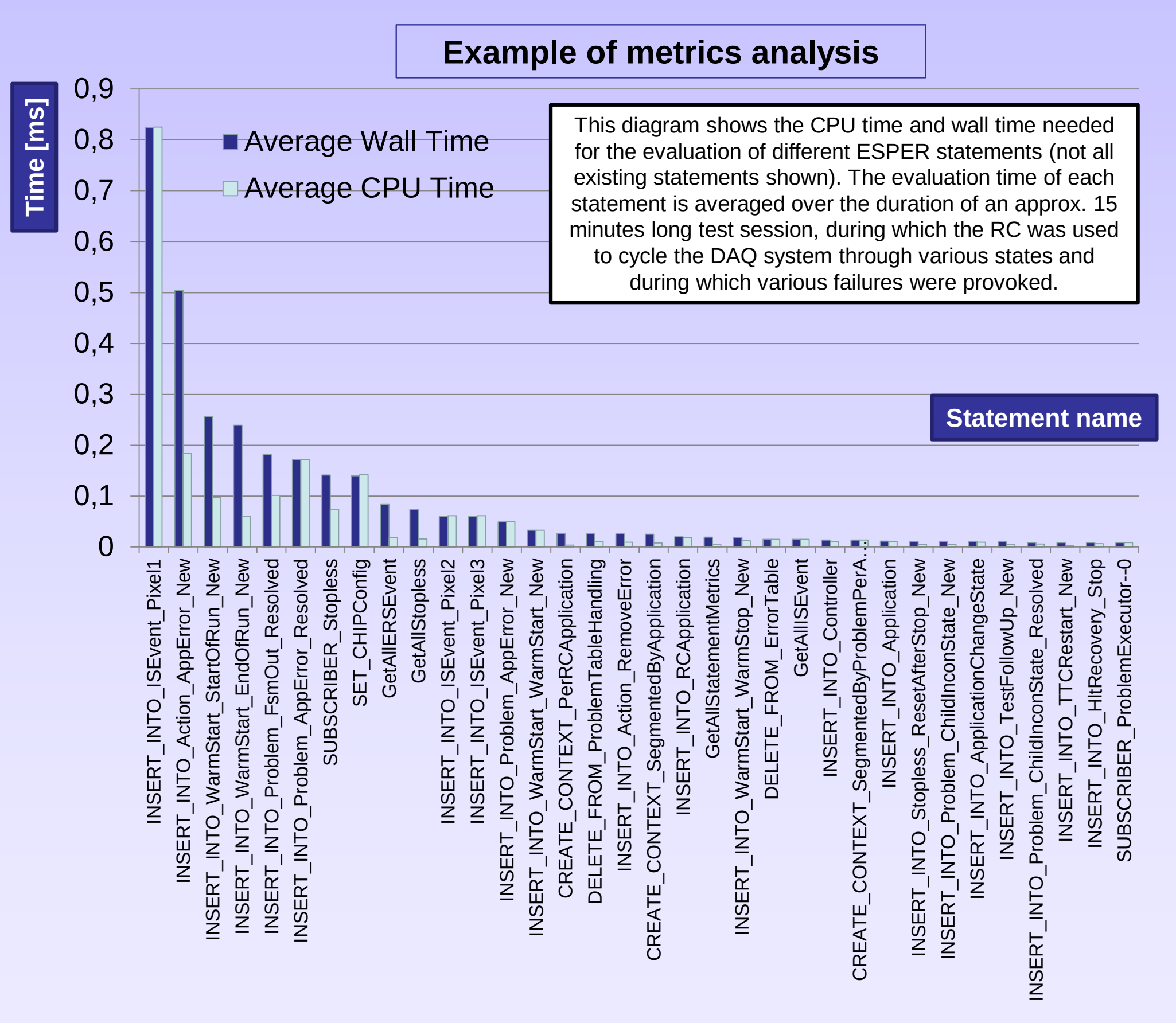
At its core, CHIP employs the CEP engine ESPER[4] which **has advanced built-in testing and monitoring** support. The knowledge base is given by a set of rules (ESPER statements). This setup allows for:

1. Rule testing
- Correct logic of new rules can be tested by artificial injection of events in a unit test

2. Metrics analysis
- Monitor CPU usage of individual rules
- CPU intensive rules can be revised

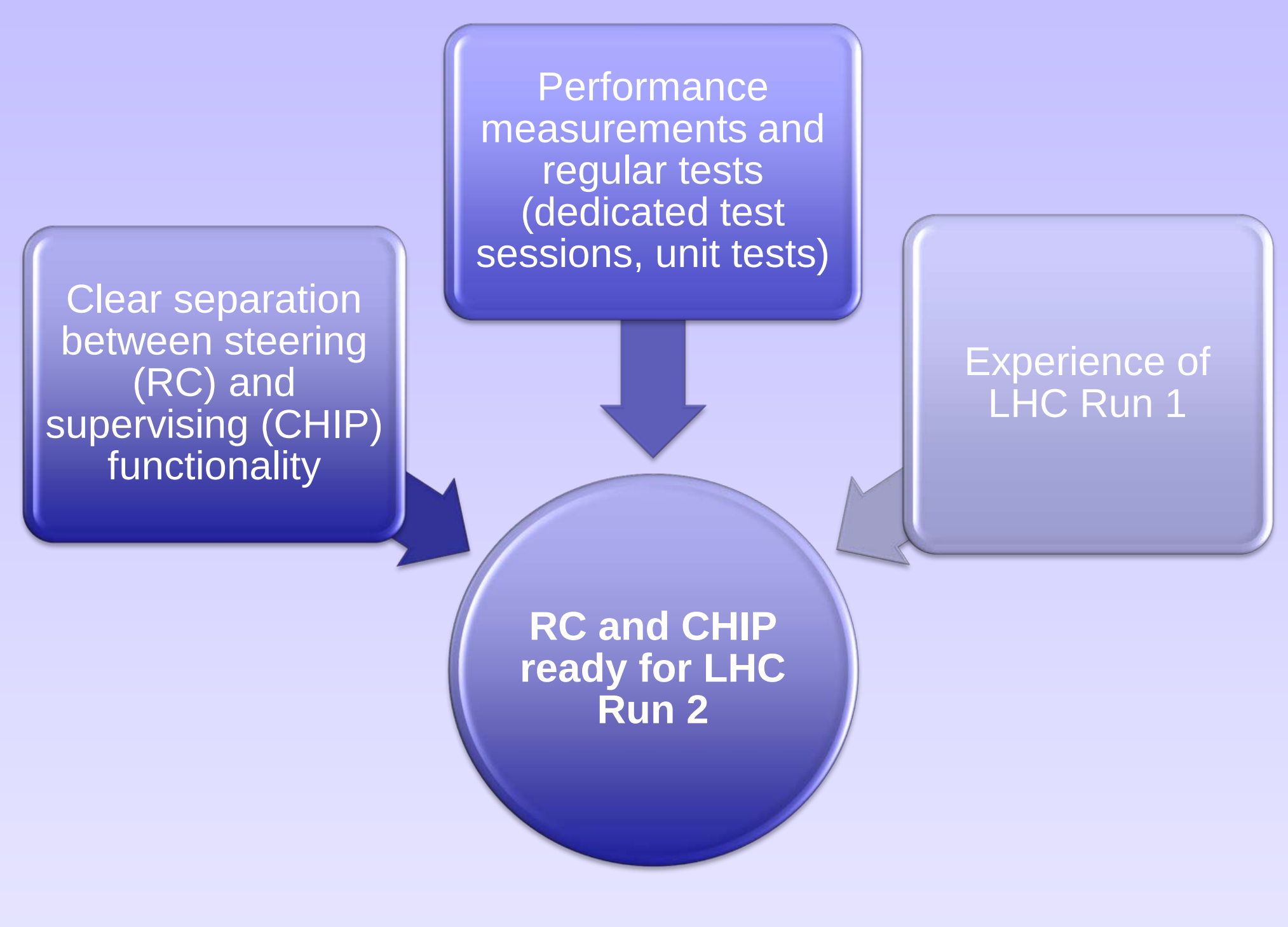
3. Configurable threading model

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



This diagram shows the CPU time and wall time needed for the evaluation of different ESPER statements (not all existing statements shown). The evaluation time of each statement is averaged over the duration of an approx. 15 minutes long test session, during which the RC was used to cycle the DAQ system through various states and during which various failures were provoked.

6. Conclusions


Performance measurements and regular tests (dedicated test sessions, unit tests)
Clear separation between steering (RC) and supervising (CHIP) functionality
Experience of LHC Run 1
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. Lehmann G., Soloviev I., *Configuration & control of the ATLAS trigger and data acquisition*, in TIPP09
4. EsperTECH, http://esper.codehaus.org/ *(May 2014)*