Realization of Inventory Databases and Object-Relational Mapping for the Common Information Model

Øystein S. Haaland

Department of Physics and Technology, University of Bergen.

November 8, 2011

Systems and Virtualization Management: Standards and the Cloud

Outline

Introduction

ALICE detector system

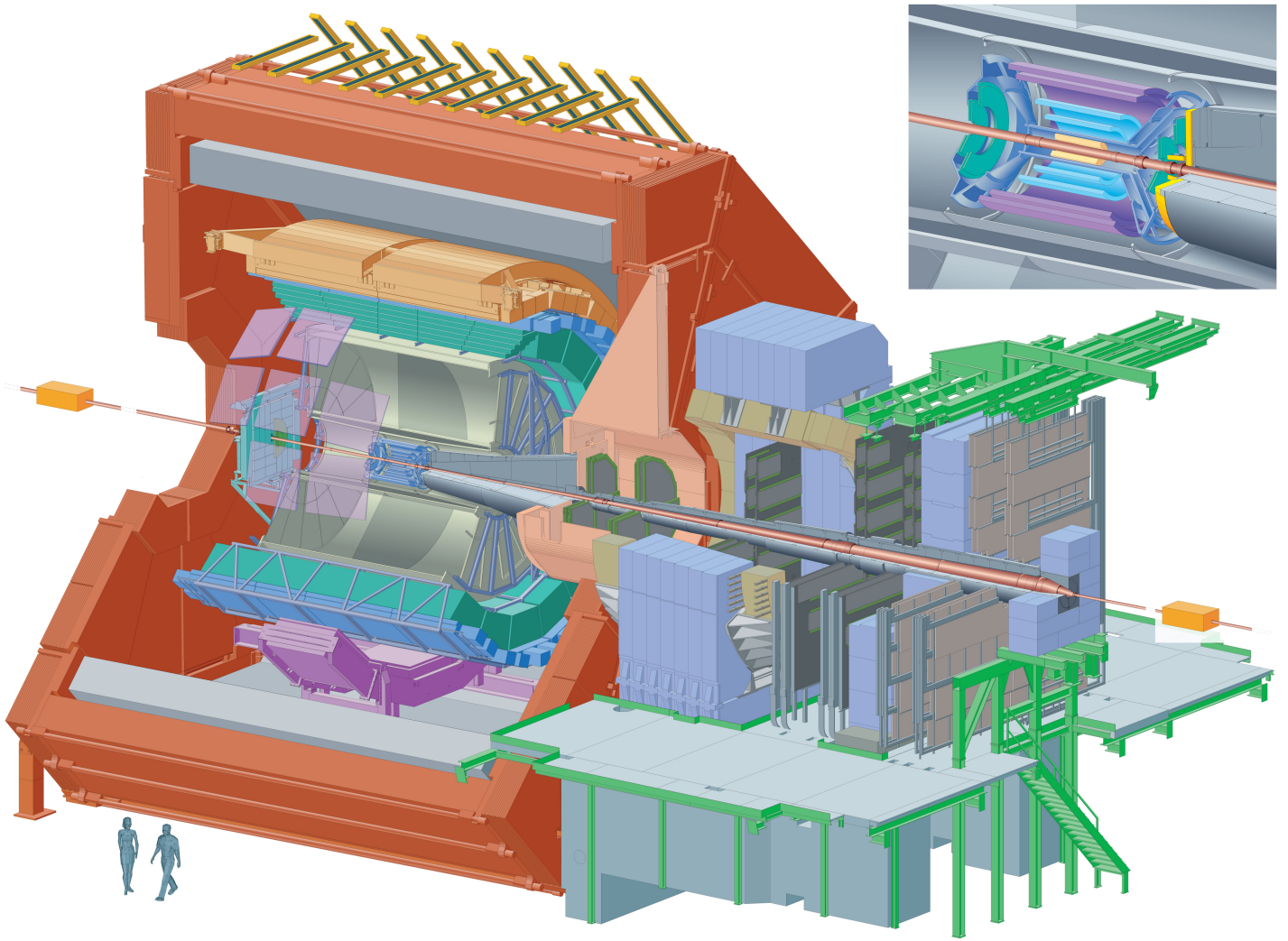

Figure: Illustration of the ALICE detector system from the ALICE web pages.

LHC/CERN

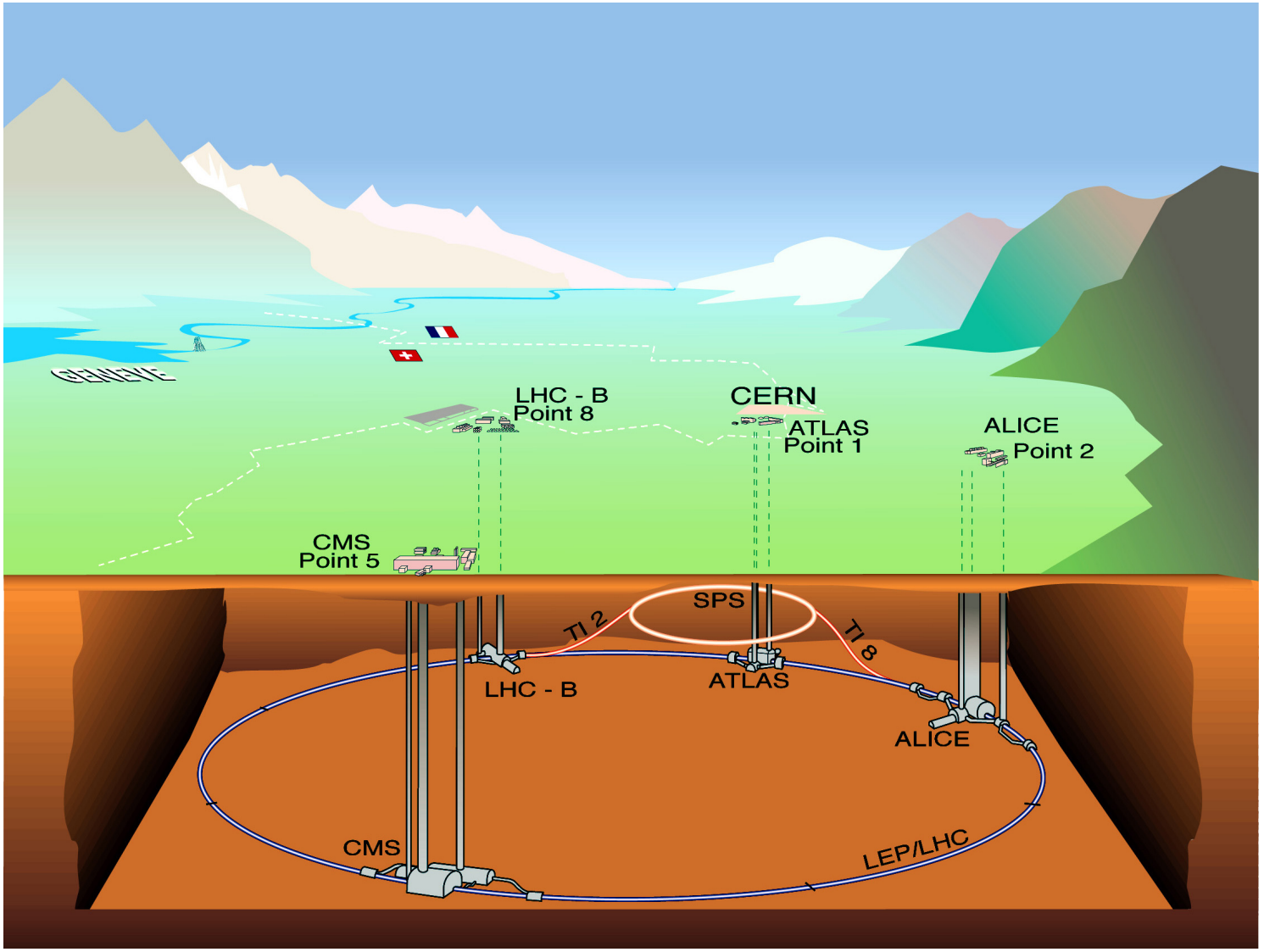

Figure: Illustration of LHC from CERN document server.

High Level Trigger

What is triggering?

- Tell data acquisition systems when something interesting happens so that it can be read-out.
- Done in several levels: lower levels in front-end electronics, while high level or event filtering typically is done in software on high-performance compute clusters.

Motivation

- ALICE can produce as much as ~25 GB/s, but bandwidth to storage is limited to ~1.25 GB/s.
- Much of the data is not new physics and it can therefore be discarded.

How is it done?

- Selection of interesting events: Event filtering or triggering.
- Selection of interesting parts of events: Region of Interest.
- Compression of events: Reconstruction and lossless compression.

ALICE HLT cluster

Online system

- Online system, not a batch farm. Cannot rerun when something goes wrong.
- Any faults have direct impact on the physics results.

Process placement

- Hierarchical and distributed processing.
- Process placement important aspect.

Challenge

- Must have reliable and stable operation.
- But using commodity hardware + complex software configuration => have to build in resilience for failure.

Cluster

Figure: Picture of cluster.

Usefulness of Inventory Databases

Inventory database

- To know which part to order when something breaks.
- To keep track of systems when reorganizing.
- To have relevant and up to date information available when debugging.

Former solution

- Typical LAMP stack with a manually maintained database.
- Too much work to keep information up-to-date with the available man-power.

Advantages of automatic updating

- Automatic process placement of the distributed physics application.
- Implementing fault-tolerance with fail-over procedures.
- Increased consistency in inventory data. No room for human mistakes.
- Reduced cost as the cluster administrator is relieved from a tedious and laborious task.
- System management and monitoring.

Requirements

Functional

- Provide up to date information about the state of the cluster for the physics applicaton.
- System management and monitoring applications need inventory information.

Non-functional

- Support the programming languages that are in use in HLT: Python, Java, C++.
- Scalability. HLT was designed for a maximum of around 1000 nodes. Currently around 200.
- Reliability. Need to ensure availablility of inventory information through use of redundancy or clustering.

Choosing an approach

Options

- Use existing CIM implementations (sblim, openpegasus, OpenWBEM).
- Use own model with ORM.
- Use CIM model with own implementatino in ORM.

- Needed support for multiple programming languages.
- Wanted tight integration with existing software.
- Needed only information gathering, not complete instrumentation.
- RDBS in the back-end would make it easier to implement high-availability and scalability.

Object-relational mapping

- Object-relational mapping is a programming technique for persisting data from object-oriented programming languages to relational databases.
- Software packages that implements this conversion in a generic way, so that it can be used for any object hierarchy, are called object relational mappers.

Examples

- Hibernate for Java was one of the first of such mappers and is maybe also the most well-known.
- Sqlalchemy for python is an ORM, that includes a declarative way of defining objects, further simplifying the process of mapping objects to databases.

Advantages of ORM

- Relational Database Systems: transactionality and enhanced data integrity.
- Object-Oriented programming: Reuse and Modularity.

Model hierarchies and level of mapping

Two options:

- Mapping at the level of CIM meta schema: flexible, but no intuitive mapping between CIM classes and programming language classes. Lack of constraints.
- Mapping at the level of CIM schema: less flexible, but more natural mapping between CIM classes and OOP classes.

Decision: map at level of CIM schema, counter less flexibility by automatic generation. Makes it as flexible as mapping at meta level.

Applying ORM to an existing model

The typical mappings one have to consider when applying an ORM to a model, are:

- Model to relational database schema.
- Model to class definitions in target programming language.
- The actual ORM mapping between the relational database schema and the class definitions in the programming language.

Mapping details

Tree aspects to consider for ORM:

- Mapping of structure.
  - Classes and properties: Normally a straight-forward mapping from CIM model.
  - Datatypes: relatively easy to find corresponding data types in both programming languages and relational databases.
  - Inheritance: This is where it gets harder. Has to decide for common type of mapping.
- Mapping of constraints.
  - Qualifiers: for instance maximum values for integers or lenght of strings.
- Mapping of behavior.
  - Methods: This we do not consider as it is not needed in our environment.

Major tasks

The effort of designing the inventory solution can be divided in two major tasks:

1. Creating a programming language representation of a CIM model instance as well as the corresponding object-relational mapping.
2. Automatically keeping the data in the CIM model instance up to date.

Two implementations

Two implementations have been developed:

- For supporting more than one programming language: python and java.
- To be able in the future to test interoperability between solutions.

Mapper generation tools 1/2

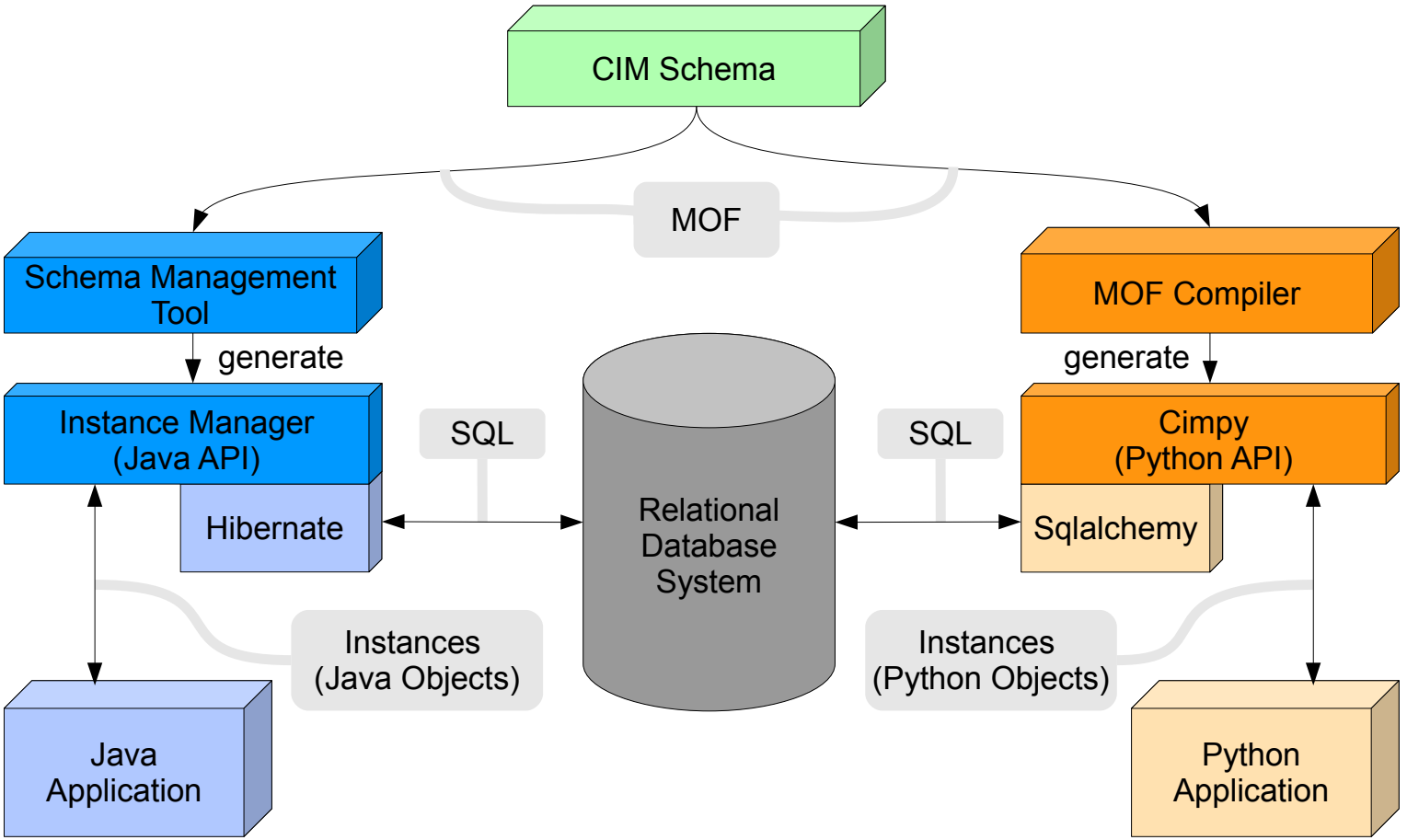

Figure: Overview of the tools developed.

Mapper generation tools 2/2

Mapper generation tools

- For Python, the mapper generation tool is implemented as a simple back-end to the SBLIM MOF compiler that outputs ORM-enabled python code.
- For Java this is a flexible framework with a lot of functionality (Chainreaction). For more information, see the references of the paper.

Inventory gathering systems

- The python inventory solution is a stand-alone inventory gathering system for compute clusters.
- The Java solution is a module that is integrated in SysMES, where the framework takes care of the client/server

Inventory gathering system

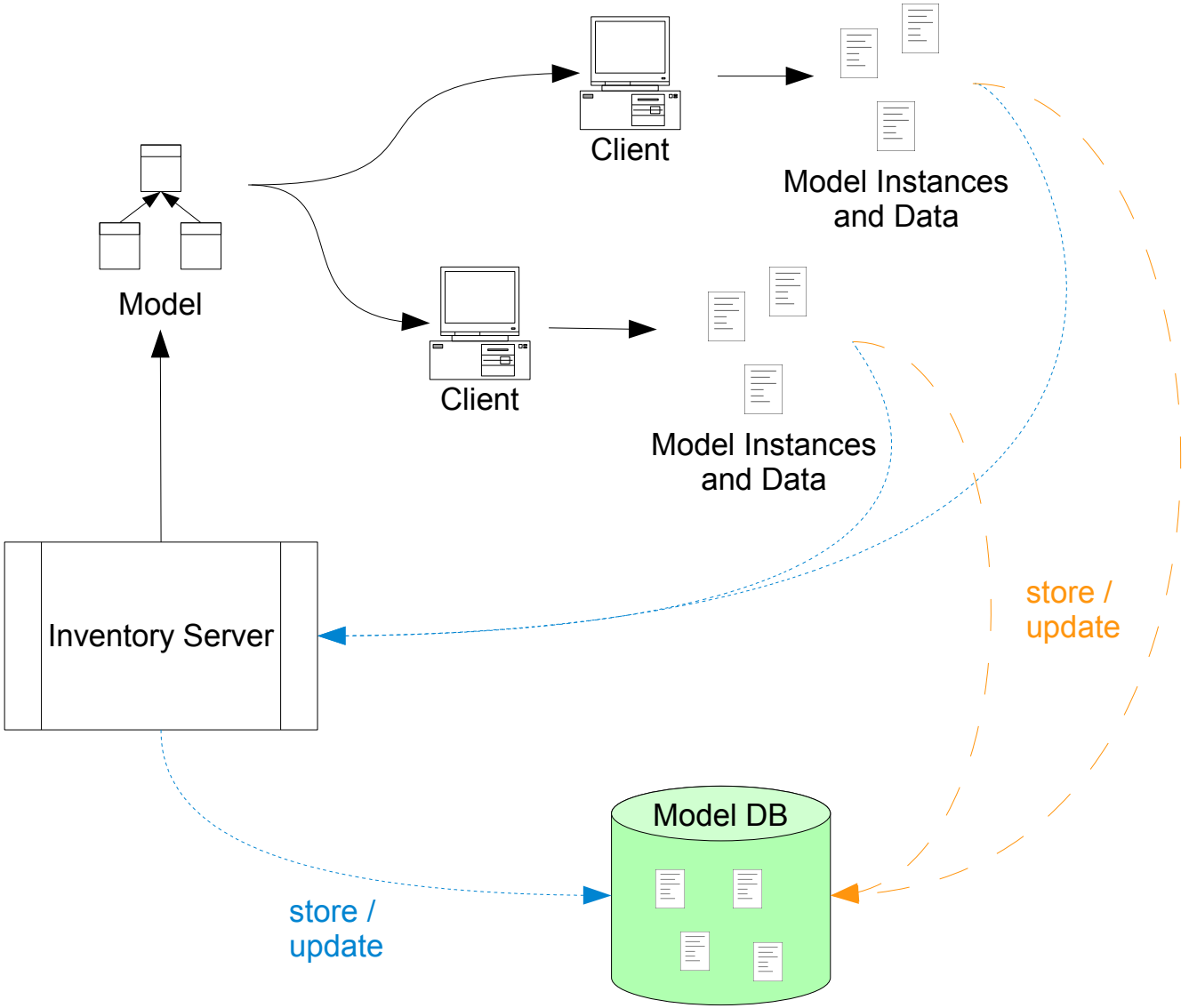

Figure: The two inventory gathering systems.

Data access 1/2

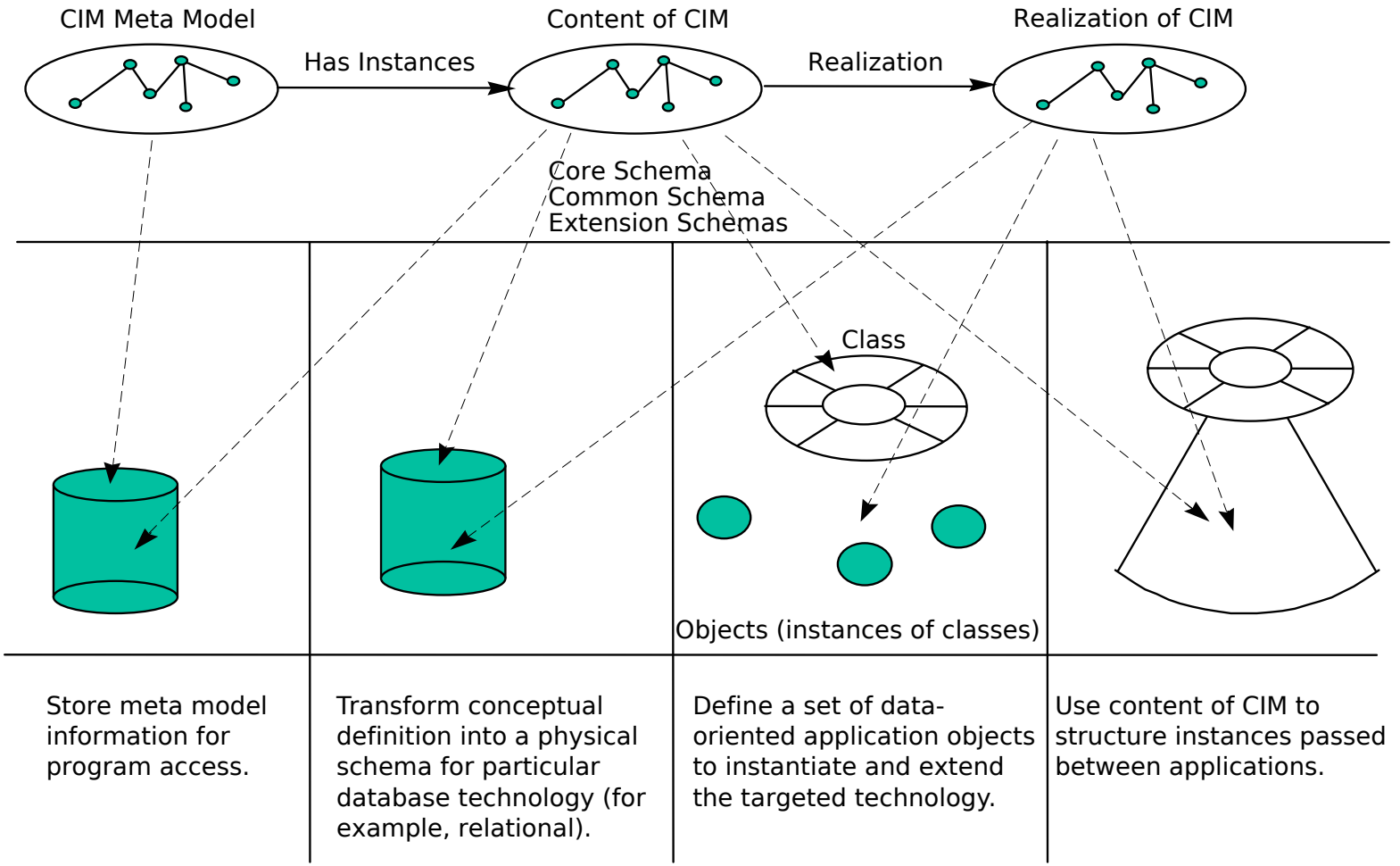

Figure: Four ways to use the CIM. Figure taken from "Common Information Model (CIM) Infrastructure" - DSP0004.

Data access 2/2

- Exchange of inventory information at both SQL- and ORM-level.
- Provide XML-RPC (or similar) bindings created by introspection of the generated CIM instance to extend compatibility to languages without ORM.
- The client/server transport of the inventory gathering solution is a forth option for data access. The ways of access inventory information is summarized in the table in the next slide:

Summary of access methods

Table: Comparison of the different access methods to the inventory data.

Access method	Data operations	Interoperability	OO-like client side
ORM	CRUD and complex queries	Limited to languages with ORM solutions	Yes
Inventory Gathering System	CRUD	Depends on implementa-tion	hwdiscover: Yes SySMES: No
XML-RPC	CRUD	All common languages	No
SQL	CRUD and complex queries	All common languages	No
Web service	CRUD	All common languages	No

Scalability and reliability

- Typically with ORMs, one can choose which RDBS should be used for persisting objects.
- This means that high-availability and scalability can be achieved by careful selection and configuration of database solution, mostly independently of the rest of the system.

Python implementation

Two differences

- Used together with the PerspectiveBroker of the twisted framework, to implement a "Remote Persistent Object".
- Uses declarative module of sqlalchemy -> only needs to provide the class definitions and the declarative mapper does the rest.

Future

- Make use of inotify for updating information.
- Extend the inventory gathering system with resource management capabilities.

Conclusion

- Normal relational databases can be used for storing an instance of the CIM model.
- Transactionality is handled by the database/ORM layer, don't need to think about this in code.
  - Makes it easier to code concurrent, distributed applications (with a central data storage).
- Can use features already present in relational database products. Most important for us:
  - High-reliability: i.e. can choose database with clustering support
  - Scalability: can go from convenient unittesting in sqlite to mysq/postgresql/oracle for deployment.
- Native and intuitive language bindings due to automatic creation of ORM-enabled code.

Results

- Only the functionality of the python implementation has been tested on the production cluster.
  - Works as expected. Information is gathered and the databasse is filled.
  - No performance or interoperability tests have been done so far on the production cluster.
- Limited performance testing has been done on smaller setups.
  - sqlite in memory is twice as fast as on disk.
- The java implementation has been tested in a test cluster in Heidelberg.

Outlook

The tasks for the near future will be extend the infrastructure that can be used to collaboratively work on increasing the interoperability of the CIM+ORM frameworks through unification of the mapping and its configuration.

- Facilities that can be used to automate conformance testing, interoperability testing, unisttesting, package building and more.
- If the goal of supporting many programming languages working on the same database at the same time (given generators are provided) is reached, it will dramatically improve the interoperability on the data persistence layer.