FIO

Fabric Infrastructure and Operations

CERN IT Department

CASTOR2: Design and Development of a Scalable Architecture for a Hierarchical Storage System at CERN

Olof Bärring, Rosa María García Rioja, **Giuseppe Lo Presti**, Maria Isabel Martin Serrano, Sébastien Ponce, Giulia Taurelli, Dennis Waldron

CHEP 2007, Victoria, September 4th, 2007

CERN - IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

CERN

Outline

- Overview
- Architecture
  - Main system's components
  - Workflow of a job
  - Features
- Software design choices
  - Code generation facility
- Conclusion

CERN - IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Overview

- CASTOR: a Hierarchical Mass Storage System
  - Transparently storing data on tape
  - Handling tape drives/robots
  - Handling a disk cache

- Latest developments (CASTOR2) motivated by LHC requirements:

	For LHC	Achieved (Aug 07)
Disk cache size (# of files on disk)	*O(*PB*)*	2 PB (7.8M)
Tape archive size	10 PB/yr	8 PB total
Aggregate I/O rate	*O(*10 GB/s*)*	7 GB/s

CERN - IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

CASTOR Client Functionality

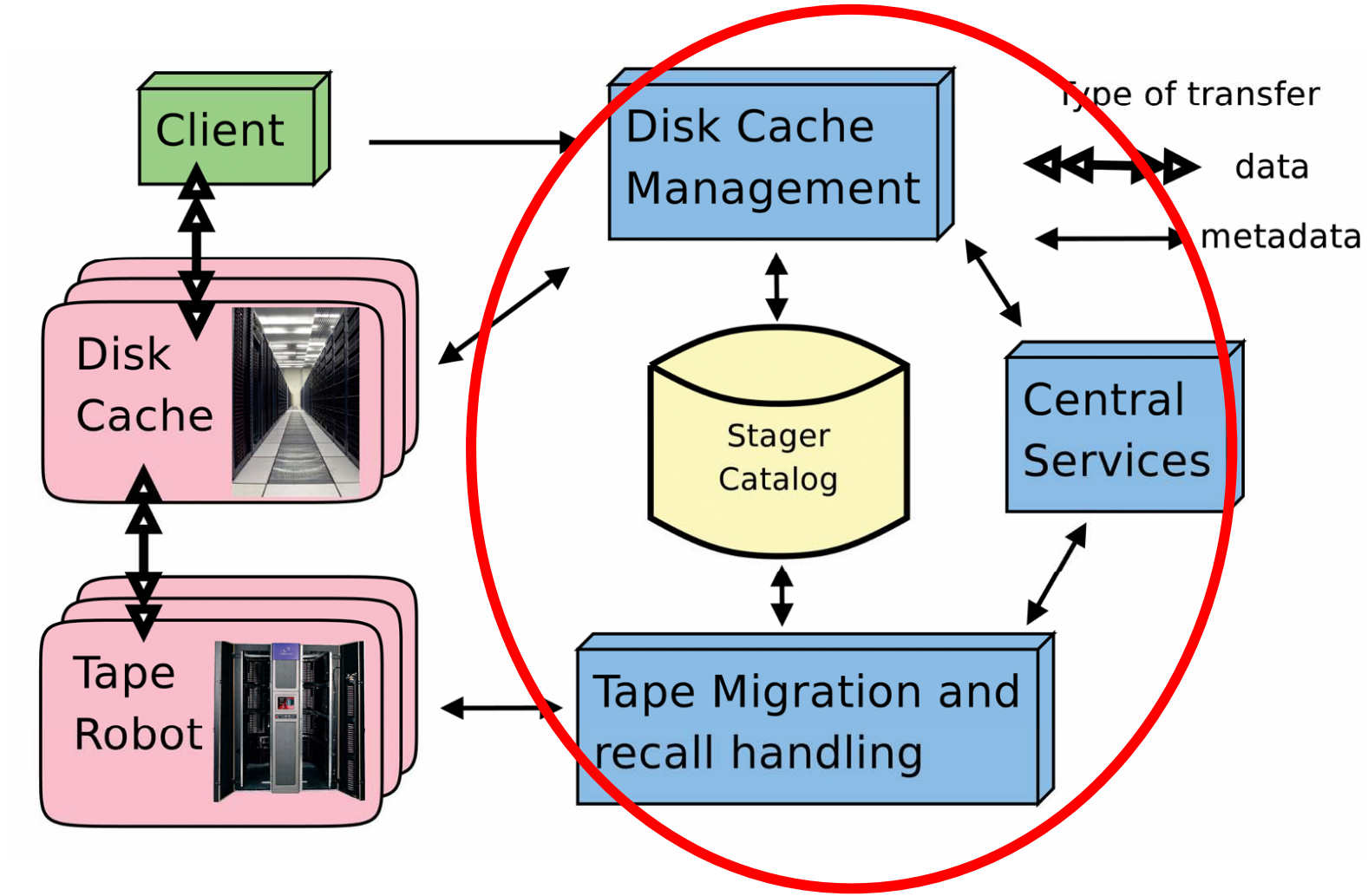
- Interfaces
  - File access
  - *Prestaging* (i.e. recall from tape)
  - Querying
- File access protocols
  - RFIO (POSIX API, native access)
  - ROOT
  - XROOT (provided by SLAC)
  - GridFTP v1, v2 coming
- Grid enabled
  - SRM interfaces v1.1 and v2.2 (provided by RAL)

Outline

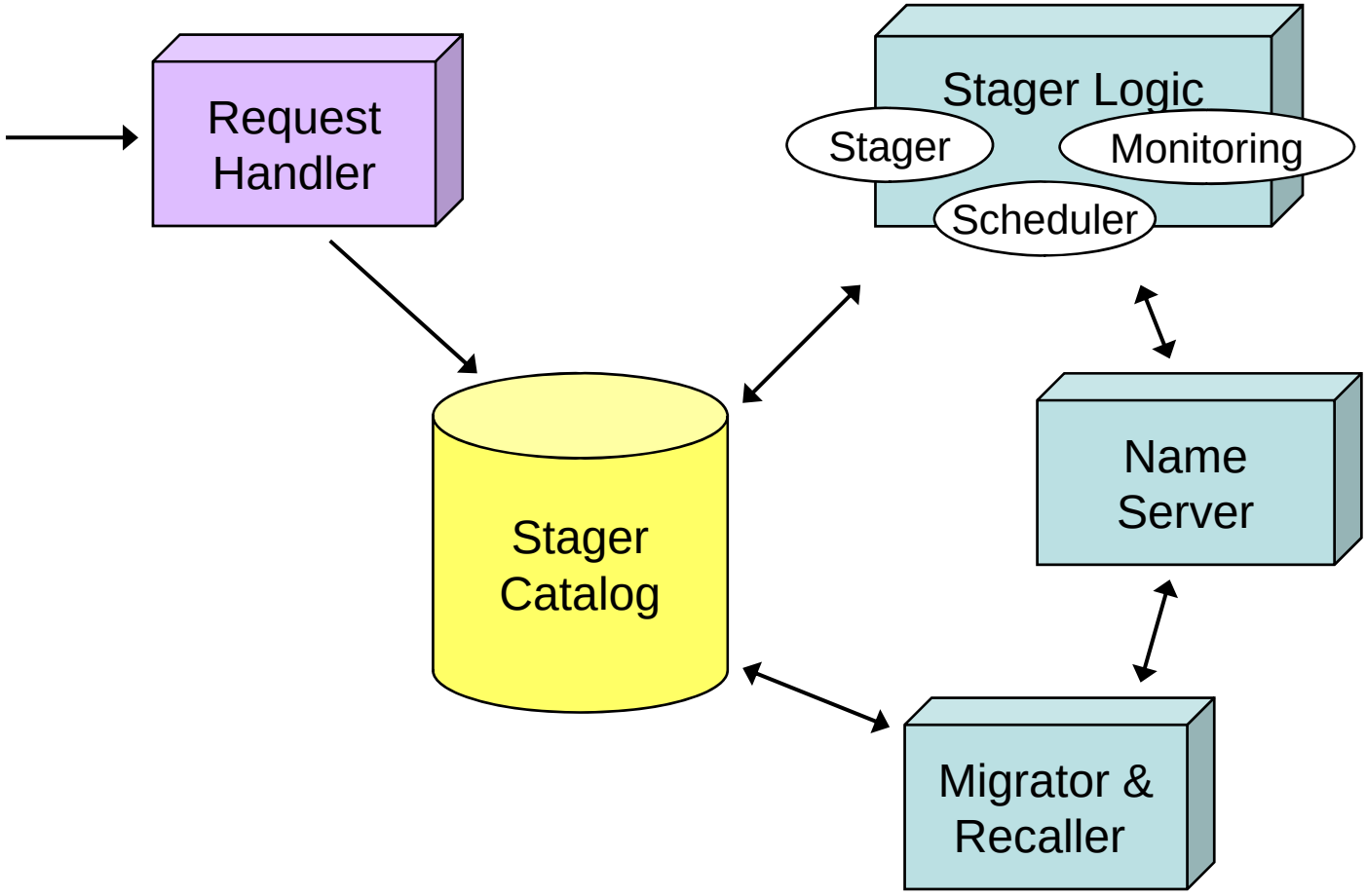
- Overview
- Architecture
  - Main system's components
  - Workflow of a job
  - Features
- Software design process
  - Code generation facility
- Conclusion

CERN - IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

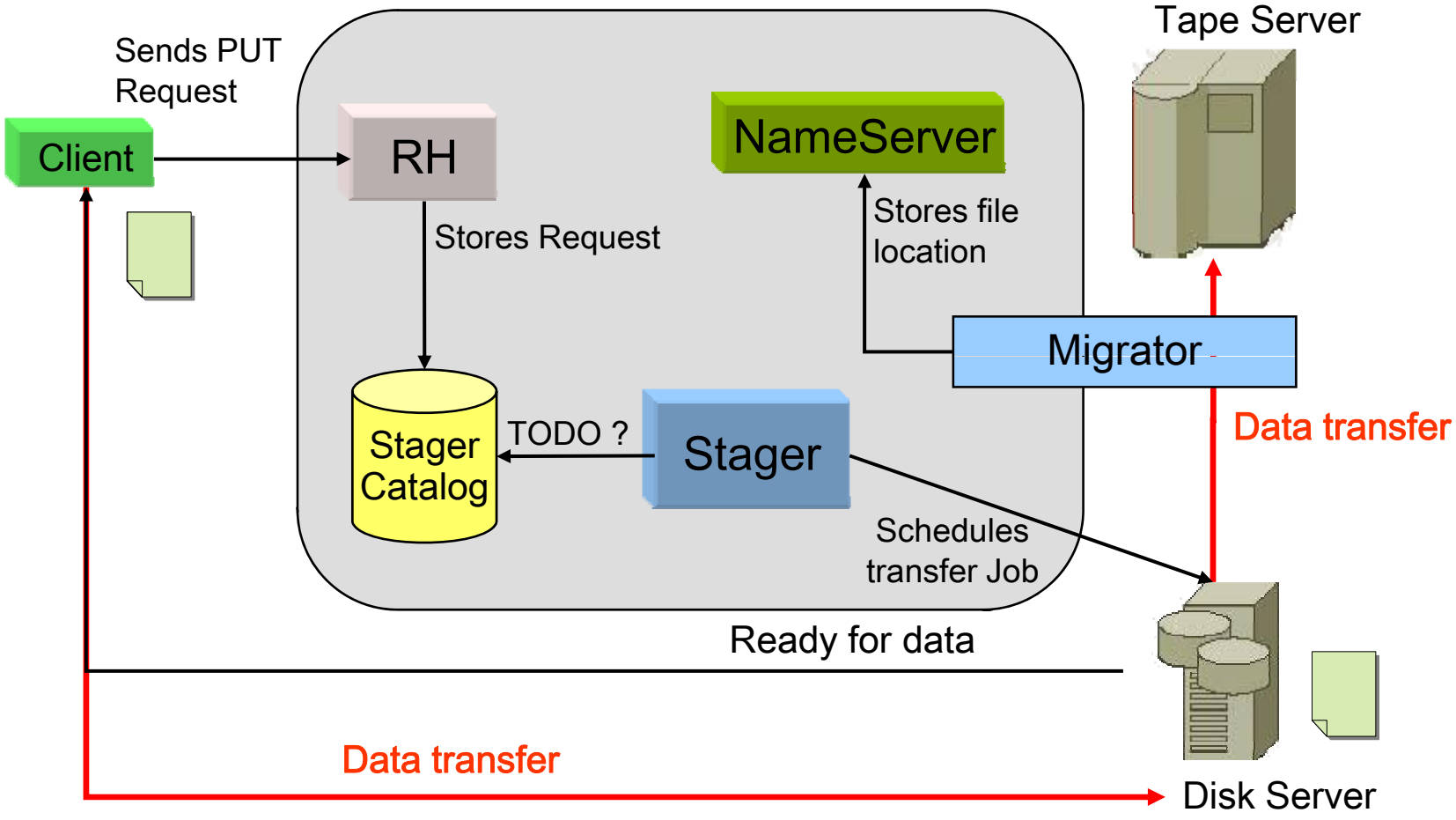
Architecture Overview



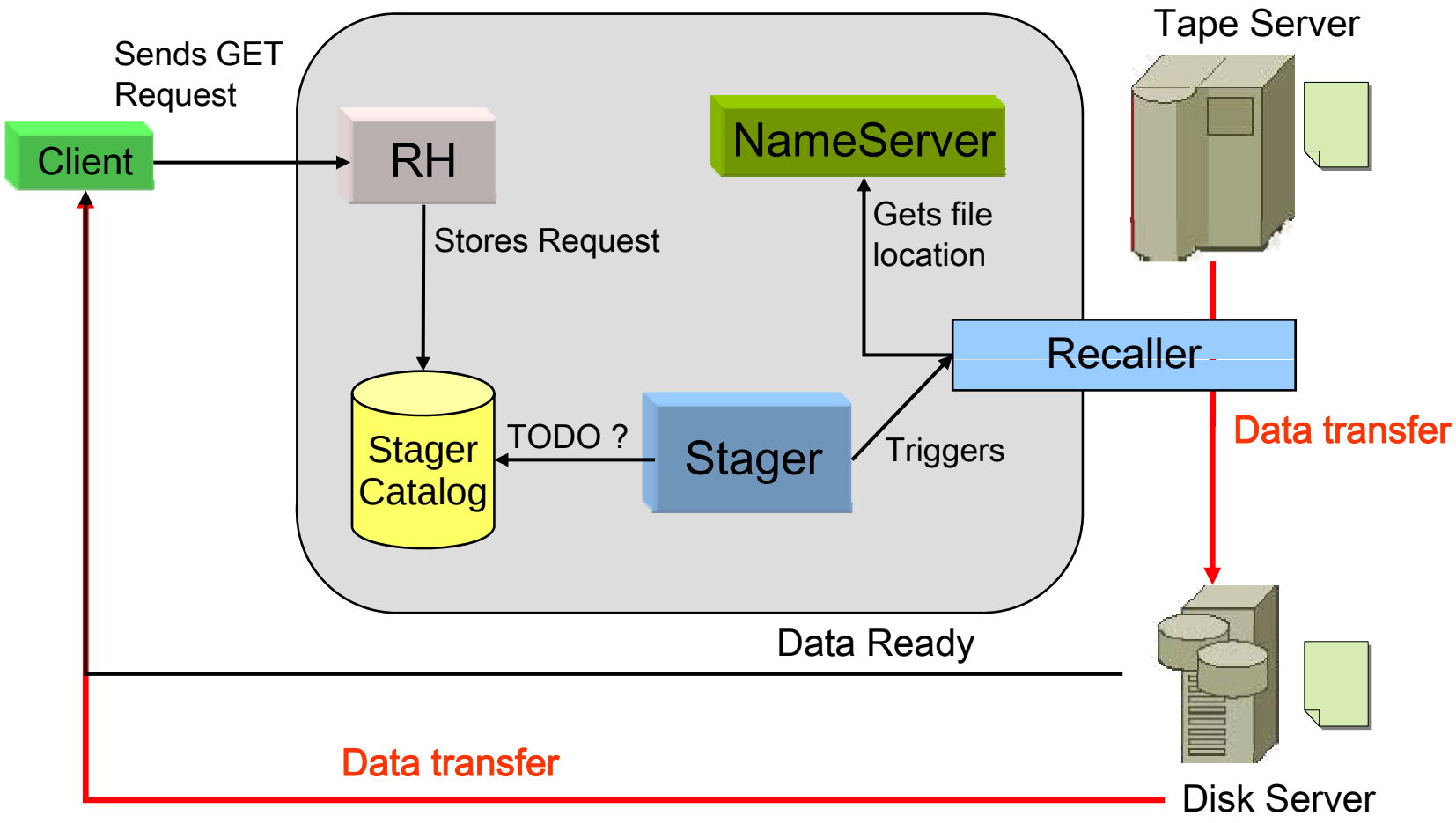
Server-side Detailed View



Workflow of a *Put* Request



Workflow of a *Get* Request



Architectural Features

- **Scheduled** access to storage resources
  - Achieve predictable performance
- Pluggable framework
  - Request scheduling delegated to a pluggable third party scheduler, e.g. Maui or LSF
  - **Externalized policies** governing resource matchmaking
- Disk server autonomy and automation
  - In charge of local resources: file system selection and execution of garbage collection
  - Resiliency to hardware failures
  - Streamlined deployment

Why Database Centric?

- Reliability
  - Components can be replicated on different nodes
  - Atomicity, state persistency and backups handled by the DB
- Scalability

  - Components can be replicated on different nodes
  - Daemons are stateless
    - Memory footprint independent of load
  - Limited by DB
    - No risk from the space point of view
    - CPU is the limit. A lot of tuning done in collaboration with CERN database experts

CERN - IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Outline

- Overview
- Architecture
  - Main system's components
  - Workflow of a job
  - Features
- Software design choices
  - Code generation facility
- Conclusion

Design Choices

- High level requirements translate into the following main design choices
  - Service-oriented framework
  - Integration with UML Design methodology
  - *Autogenerated* database access layer
- A software framework is in place, which provides all the needed facilities

CERN - IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Service-oriented Framework

- Usage of the *Factory* Design Pattern
- Notion of *Services* and *Converters*
  - A *Service* provides an internal functionality (e.g., connection to the database)
  - A *Converter* translates the content of an object to a different representation (e.g., from memory to database)
- Central place where services are loaded and instantiated
  - Dynamic loading of the needed services via shared libraries
  - Allows for dynamic configuration

```
castor::IService *svc =
   Services::service("StreamCnvSvc", castor::SVC_STREAMCNV);
```

Integration with UML

- Use of UML design methodology
  - *Sequence and Activity* diagrams to describe the workflow and the information flow
  - *State* diagrams to describe the status lifecycle and evolution of all the stateful entities
    - e.g. a request, a disk copy, a tape copy…
  - *Class* diagrams to design in details the software components
  - *Class* diagrams to *also* describe the database model: all entities must be made persistent
    - special case of an **E-R** model mapped to a set of classes

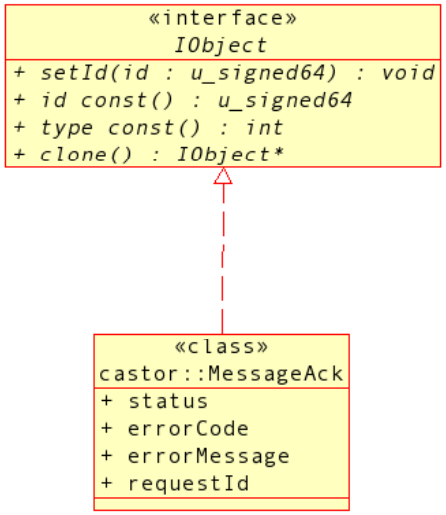
The Code Generation Facility

- Goals
  - Automate some facilities (object printing, C++ introspection)
  - Automate streaming and DB access, providing an homogeneous abstraction layer for **any** entity
  - Ease maintenance
- Input
  - A standard UML Class Diagram
- Tools
  - *Umbrello UML Modeller* (Open Source, www.umbrello.org)
  - Custom extensions maintained by the Castor team
    - Standard UML tools don't usually provide much more than just empty C++ or Java class ***declarations***

CERN - IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Autogenerated Implementation

C++ code




```
class MessageAck : public virtual IObject {
…
private:
    /// Status of the request
    bool m_status;
    /// Code of the error if status shows one
    int m_errorCode;
```

*.hpp

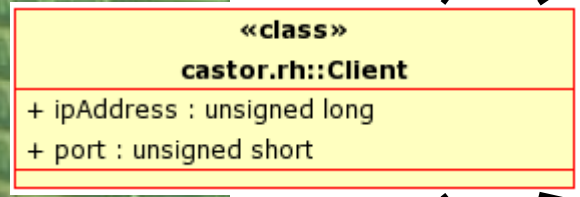
```
void castor::MessageAck::print(…) const {
  …
  stream << indent << "status : "
         << m_status << std::endl;
  stream << indent << "errorCode : "
         << m_errorCode << std::endl;
```

*.cpp

- Generated skeletons also contain basic facilities (e.g., printing)
- A C interface is provided as well
- SQL code is generated as in DB design tools

CERN - IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Autogenerated *Converters*



Streaming code

Stream*Cnv.hpp/cpp

```
virtual void createRep(IAddress*, IObject*);
```

```
void … StreamMessageAckCnv::createRep(…) {
  …
  ad->stream() << obj->type();
  ad->stream() << obj->ipAddress();
  ad->stream() << obj->port();
  ad->stream() << obj->id();
```

Db*Cnv.hpp/cpp

```
virtual void createRep(IAddress*, IObject*);
```

```
void … StreamMessageAckCnv::createRep(…) {
  …
  const std::string insertStmStr =
"INSERT INTO Client (ipAddress, port, id)…";
  …
 insertStmt = createStatement(insertStmStr);
  …
 insertStmt->executeUpdate();
```

… and DB code

CERN - IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Example

- Reading a request from a socket and storing it in the db

```
castor::IObject* obj = sock->readObject();
…
castor::BaseAddress ad;
ad.setCnvSvcName("DbCnvSvc");
…
svcs()->createRep(&ad, obj);
```

Core of the Request Handler code

Outline

- Overview
- Architecture
  - Main system's components
  - Workflow of a job
  - Features
- Software design choices
  - Code generation facility
- Conclusion

Conclusion

- CASTOR 2 software architecture proved to be able to scale up to LHC requirements
  - CERN Tier-0, Tier-1 sites
  - **7 GB/s** sustained over a week, **7.8M** disk resident files, PB-size disk cache, **10K** concurrent transfers
- Mature infrastructure, focus on tuning
  - Several parameters affect overall performance
  - Ongoing work to sustain target throughput of *O(***10 GB/s***)*
- ***Questions?***

www.cern.ch/castor

CERN - IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it