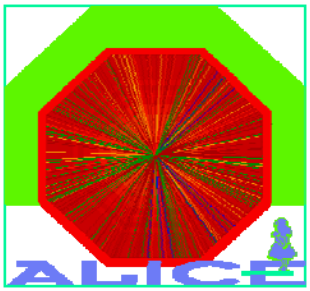


AMORE

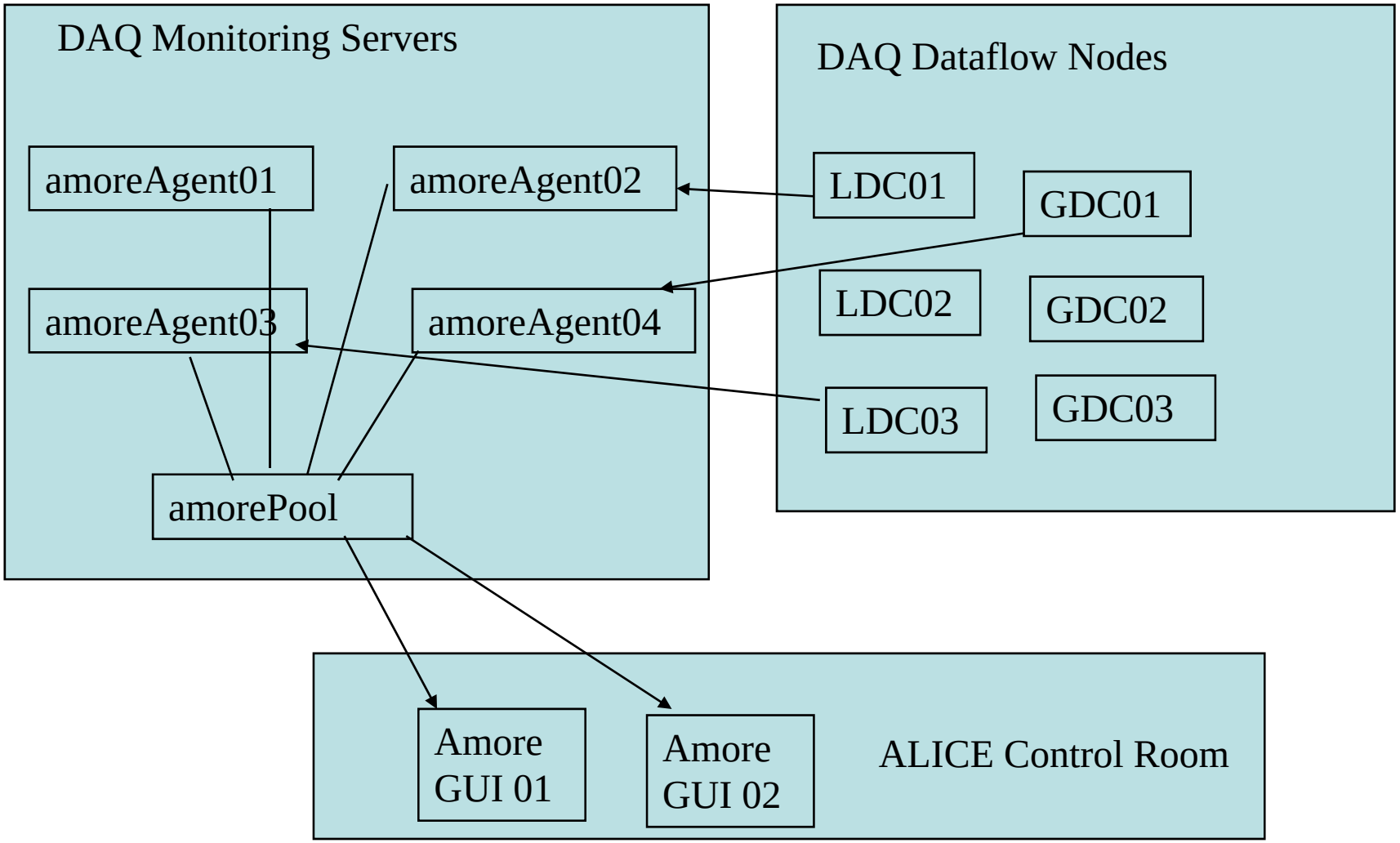
Quality Assurance Integration

Filimon Roukoutakis

CERN Physics Department
ALICE DAQ

ALICE Offline Week
CERN, 11/10/2007

The big picture

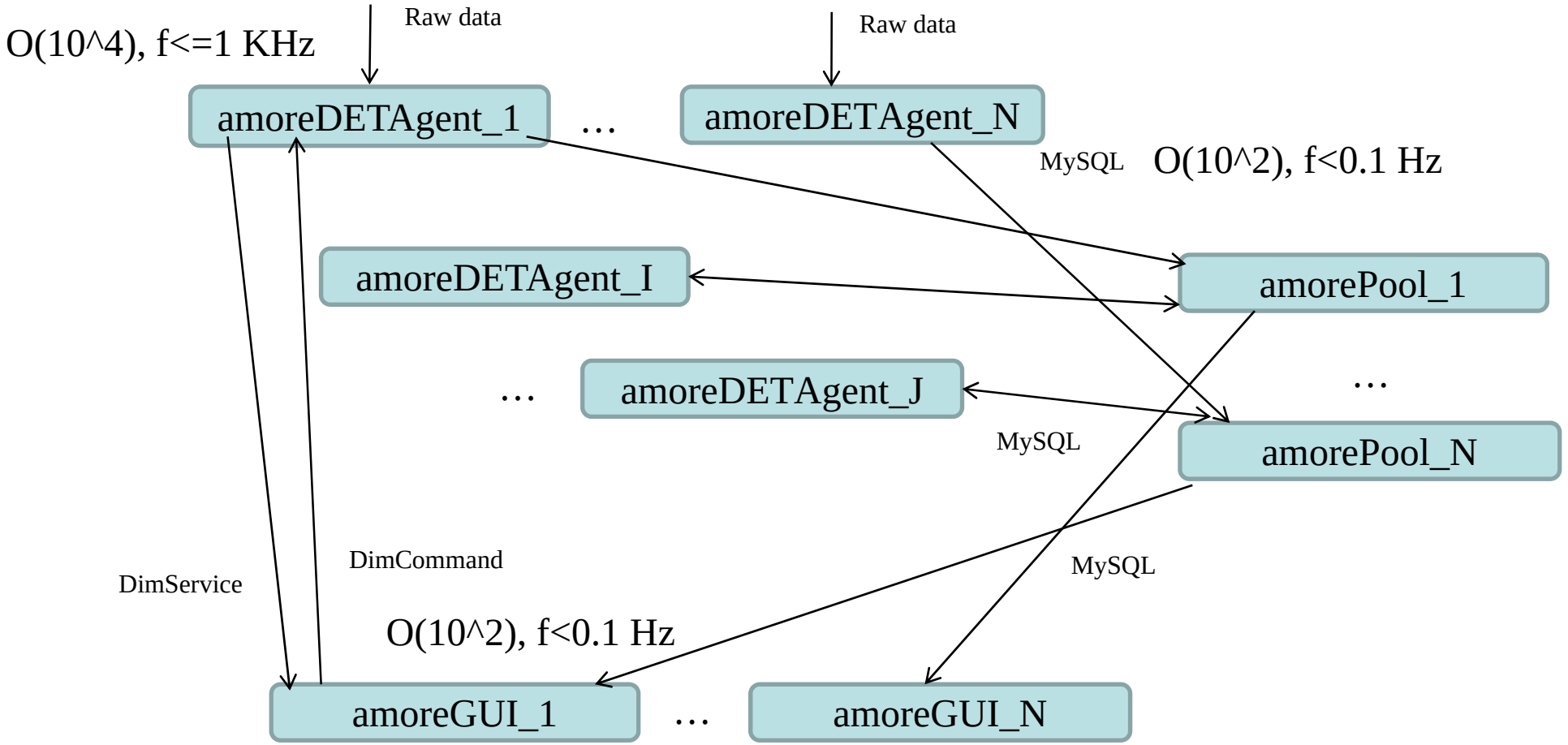


Fundamental design requirements

- Follow the publish-subscribe paradigm to allow scalable many-to-many connection between monitoring information producers and consumers. Introduce intermediate pools of monitoring data for full decoupling, thus use a classic 3-tier design
- No dependency on user code, which will be stored in the offline CVS. This is done through usage of C++/ROOT reflection

  abc_ptr* myptr=gROOT->GetClass("derived_class_name")->New()
- Use the same IPC/control systems as our DAQ framework, namely DIM and SMI++
- Use MySQL for data pools and configuration
- It should have a "lovely" name, like all the DAQ software (DATE, MOOD, AFFAIR). We called it AMORE (=Automatic MOnitoRing Environment)

Computing model



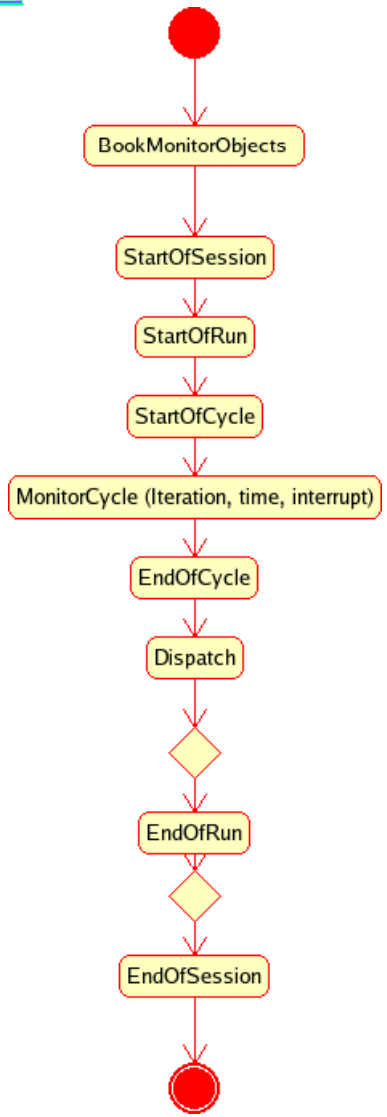
Status

- First released mid-July
- Current version 0.2.1, 4 versions released
- Can partially replace MOOD (Full physics analysis possible but no interactivity on the agent by design)
- 1 detector implementation (TOF), consult Roberto Preghenella's talk "First experience with AMORE", ALICE week Oct'07, DAQ meeting
- Fully standalone framework with the ability to use AliRoot

Publishers and subscribers

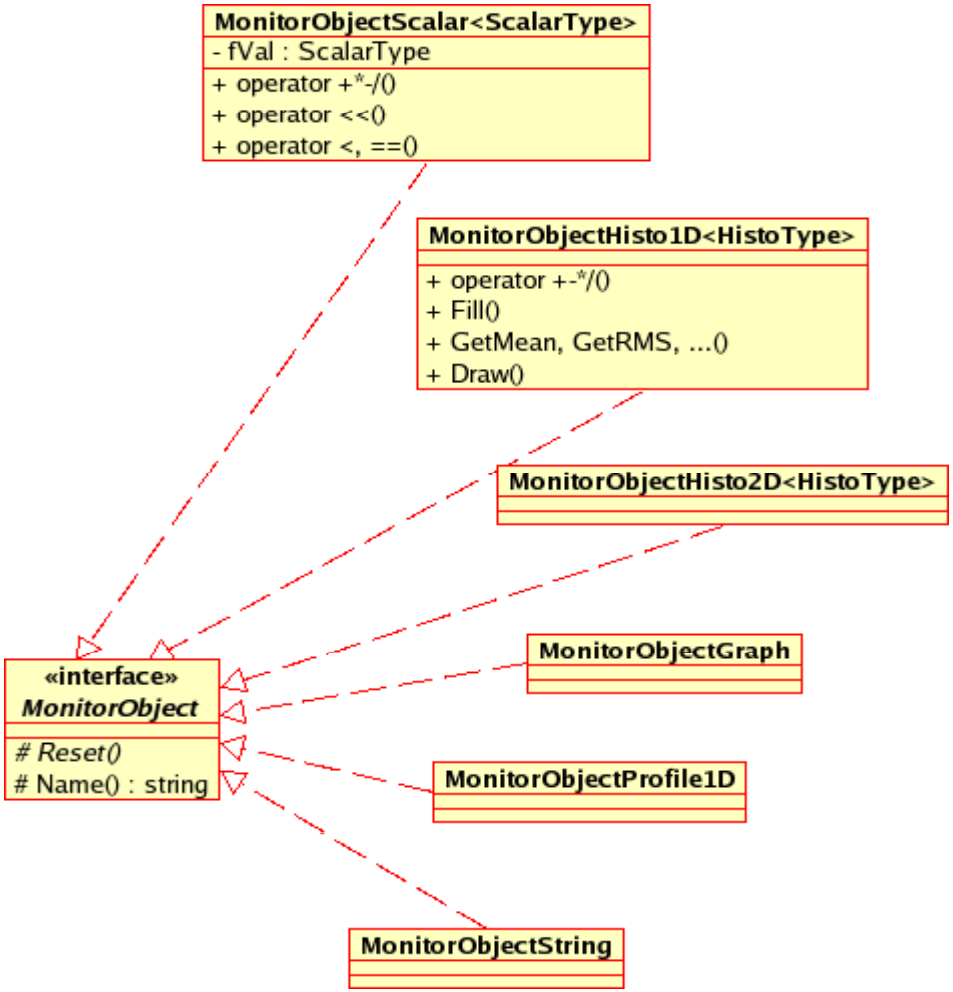
- amoreAgent is an FSM running a single publisher and (optionally) many subscribers
- amoreGUI runs many subscribers
- Each subscriber is associated to a single publisher
- Publisher and subscribers execute user code in classes derived from special ABCs that define the interface to be called by the FSM
- Arbitrary number of publisher and subscriber “modules” with user code can exist in user-created libraries and can be loaded at runtime by the framework

PublisherModule



- We follow the “Template Method” OO Design Pattern. User code overriding this ABC resides in a dynamic library loaded at runtime
- Arbitrary number of modules can coexist in the detector library, loadable by name automagically by ROOT reflection ☺
- An agent can exchange the whole module with another one at runtime with time resolution of a MonitorCycle. (Infrastructure exists, API on client side missing to do it and probably to risky to do anyway!)

MonitorObjects



- Template based design, static safe, one cannot e.g. Fill(x, y, z) a 1-D histogram.
- All fundamental C++ types and relevant ROOT types are supported.
- Attributes control the reset/update behavior, done automatically by the framework.
- Factories are responsible for the creation/registration/access/destruction of MonitorObjects
- MonitorObjectArray (of fundamental C++ types) under evaluation
- MonitorObjectTObj also supported for transport of arbitrary objects to/from amorePools (Will be used by QA)

Data Quality Assesment

- The idea is to reuse MonitorObjects as dq assesment “results” combined with references to other MonitorObjects if needed
- Data Quality checks are friends or members of the MonitorObject type they make sense to be applied on. Examples are: Value within range, chi^2, Kolmogorov,…
- Based on the “value/status” of such a MonitorObject, a process that subscribes to it can take required action, like printing messages, changing screen colors, notifying the ECS,…
- This is not relevant to QA execution, which will have its own assessment mechanisms

Visualization

- gROOT->LetThereBeLight()
- Arbitrary number of prebuilt visual layouts, loaded by name any time during a run on AMORE GUI.
- Unlike any other existing DQM framework, in AMORE the C++ code IS the visual layout configuration (no need for elaborate XML/txt configuration) through the magic of Reflection. We will still be able load configuration values from an external file/db if this is needed by detectors.
- A “browser” of the available MonitorObjects should be eventually in place

Current development activities

- According to design specifications, a subscribing process must be able to collect data from multiple agents. Implemented, under testing
- Implement selective subscription mechanism and second-level agents that can act as subscribers. Under design finalization.
- DA to AMORE interface: Prototype implemented, need to discuss a few housekeeping/bootstrapping issues within DAQ. Essentially a cut down version of AMORE allowing dispatching/retrieval of discrete, single TObjects at-a-time to/from amorePools.

Offline QA execution in AMORE

- Will use part of AMORE capabilities
- QA framework has been refactored to be compatible with AMORE FSM
- QA Objects can export a TList of histograms that is accessible by AMORE for publication
- Detector experts need to implement an AMORE “Publisher” module in such a way that there is 1-1 correspondence between the AMORE and the QA flow of execution. An example will be available soon.
- A UI module must be also provided in order to visualize the histograms at runtime.
- These modules will be part of AliRoot as separate targets ala DA code
- A tagged version of AliRoot must be deployed on the monitoring servers, could change once a week and will be common for all detectors (shared libraries are used). -> Detector code is production quality, no possibility to use HEAD version.

Sample of AMORE-QA Interface

```
void PublisherQA::BookMonitorObjects() {

 fqadmList=fqadm->Init(AliQualAss::kRAWS, 0, 0);
 TListIter* lIt=(TListIter*)fqadmList->MakeIterator();
 TNamed* obj;
 while((obj=(TNamed*)lIt->Next())) Publish(obj, obj->GetName());

}

void PublisherQA::StartOfCycle() {

 fqadm->StartOfCycle(AliQualAss::kRAWS);

}

void PublisherQA::EndOfCycle() {

 fqadm->EndOfCycle(AliQualAss::kRAWS);

}

void PublisherQA::MonitorEvent(amore::core::Event& event) {

 fqadm->Exec(AliQualAss::kRAWS, event.GetAliRawReader());

}
```

Partially un-dressed ☺ rehearsal with AMORE during cosmics

- Following ATLAS/CMS experience on several combined cosmic runs, <= 2x10^4 histograms should be expected to be available -> Deploy < 10 agents

- The goal will be to test the framework with the most advanced detectors at that time and in addition the DA-AMORE interface, especially if more detectors requests come for this.

- Plan of work for each detector (Contact person):
  - Implement your QA according to offline requirements (Yves)
  - Implement the AMORE interface to the QA (Filimon)
  - Deploy on a basic test system (ala DAs) (Filimon)
  - Deploy at Pt2 (Filimon)

High priorities

- An agent control mechanism through SMI++ Implement 3 basic functions (SOR, EOR, Reset) to test the concept. Only afterwards integrate with ECS…
- A MonitorObject “browser”. If an agent has several thousands MonitorObjects, dynamic selection of the viewed histograms is needed. This requires a working implementation of the full subscritpion mechanism (previous slides). It will reuse AliEVE components…
- Archival of results at EOR. We can have a semi-trivial implementation by exporting ROOT files to FXS before other needs are identified. Related to first point.
- Retrieval of DCS information will be done through DCS FXS in form of ROOT files that will be available to an amoreAgent at SOR.

Conclusions

- Released since >2 months
- Stable functionality, API is probably well defined, still some work to bring the promised futures to the users (but these will magically appear with every new version ☺)
- AMORE is currently MOOD on steroids. amoreAgent can easily handle O(10^3) MonitorObjects and the MonitorObject handling is already well defined.
- Several things need to be in place for cosmic run but the needs are better defined now. Further detector requirements will be honored after cosmics but discussion can be ongoing.
- QA integration into AMORE is rather trivial. It is now top priority that QA code is functional so it can be used by AMORE.

BACKUP

RDBMS as amorePool

- Pool implementation is the most critical one for several reasons
- Most problems are common to the ones RDBMS try to solve -> Try to use a solution written by experts to do our job -> Use MySQL
- Each amoreAgent has its own table in a database to store the published data. When a subscriber “subscribes’ to this agent, essentially accesses this table to retrieve monitoring data
- MySQL is also used for the configuration of the system and agent bootstrapping
- The pool implementation is isolated behind an abstract “dispatching” interface. A custom created pool can almost transparently replace the MySQL implementation.

Deployment

$AMORE -\ (=/opt/amore)

bin -\

The AMORE batch and interactive executables and the setup/configuration shell scripts

include -\

amore -\

*.h (The AMORE API)

lib -\

The AMORE libraries, static and dynamic versions)

- Distributed as a binary RPM package for SLC4
- Requirements: ROOT >= 5.16, DATE 6.x (DAQ software framework), MySQL (DATE compatible version)

$AMORE_SITE -\ (=/opt/amoreSite)

lib -\

libAmoreDETCommon

libAmoreDETPublisher

libAmoreDETSubsciber

libAmoreDETUI

- DET=3 alphanumeric digit official DET code
- A template DET tarball contains example code and makefiles to create the libraries above and package them in RPM These libraries readily link against ROOT/AliROOT by default (SINGLE version existing on DAQ network). They use the AMORE API in $AMORE/include/amore
- No need for a central repository for DET code in DAQ CVS (The offline may however pursue this if desirable).
- All common stuff that is to be used by both servers and clients should be in Common lib. Do not assume that all the libraries exist in every system. All code lives under amore::DET::xxx namespaces, xxx=publisher, subscriber, ui, common

MOOD replacement (To be scheduled)

- A single process application like MOOD, essentially fusing amoreAgent+amore and bypassing the need of amorePool.
- It will load concurrently the PublisherModule and UIModule desired, so full AMORE code reuse is guaranteed.
- All the advantages of AMORE, namely revised core functionality and no DET code dependency of the source distribution. Could be just another binary target in the AMORE distribution, namely a “MOOD” distributed within AMORE.
- Timeline: Prototype early 2008 ???