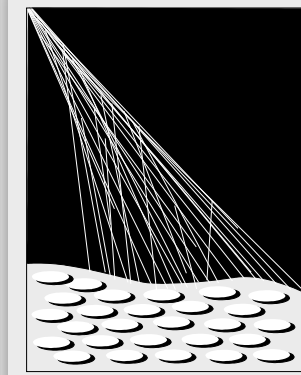




PIERRE
AUGER
OBSERVATORY

Detector and Event

Lukas Nellen

ICN-UNAM

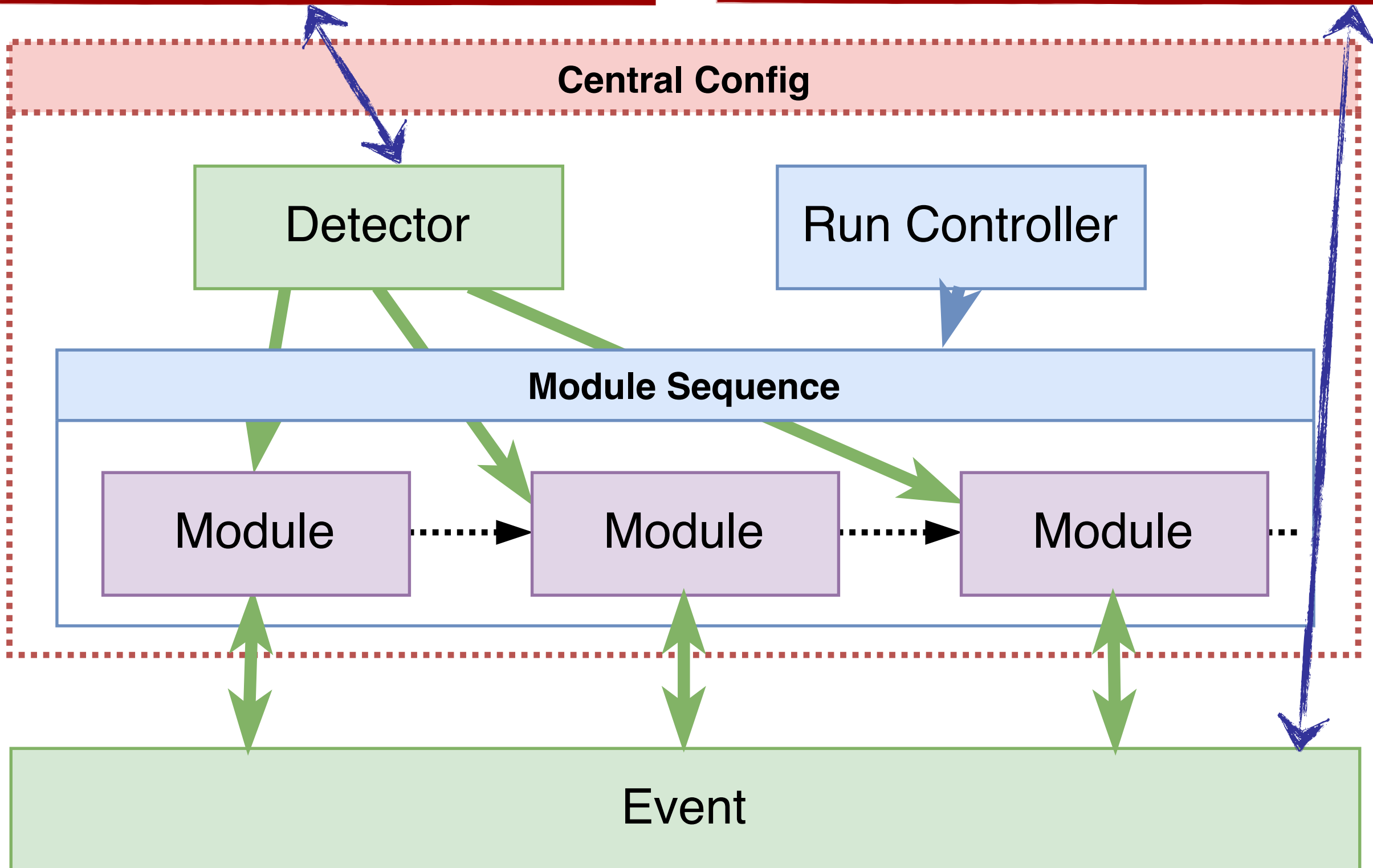
lukas@nucleares.unam.mx

PAPIIT IN114924

Recall: building blocks

- Read only
- Slowly changing data

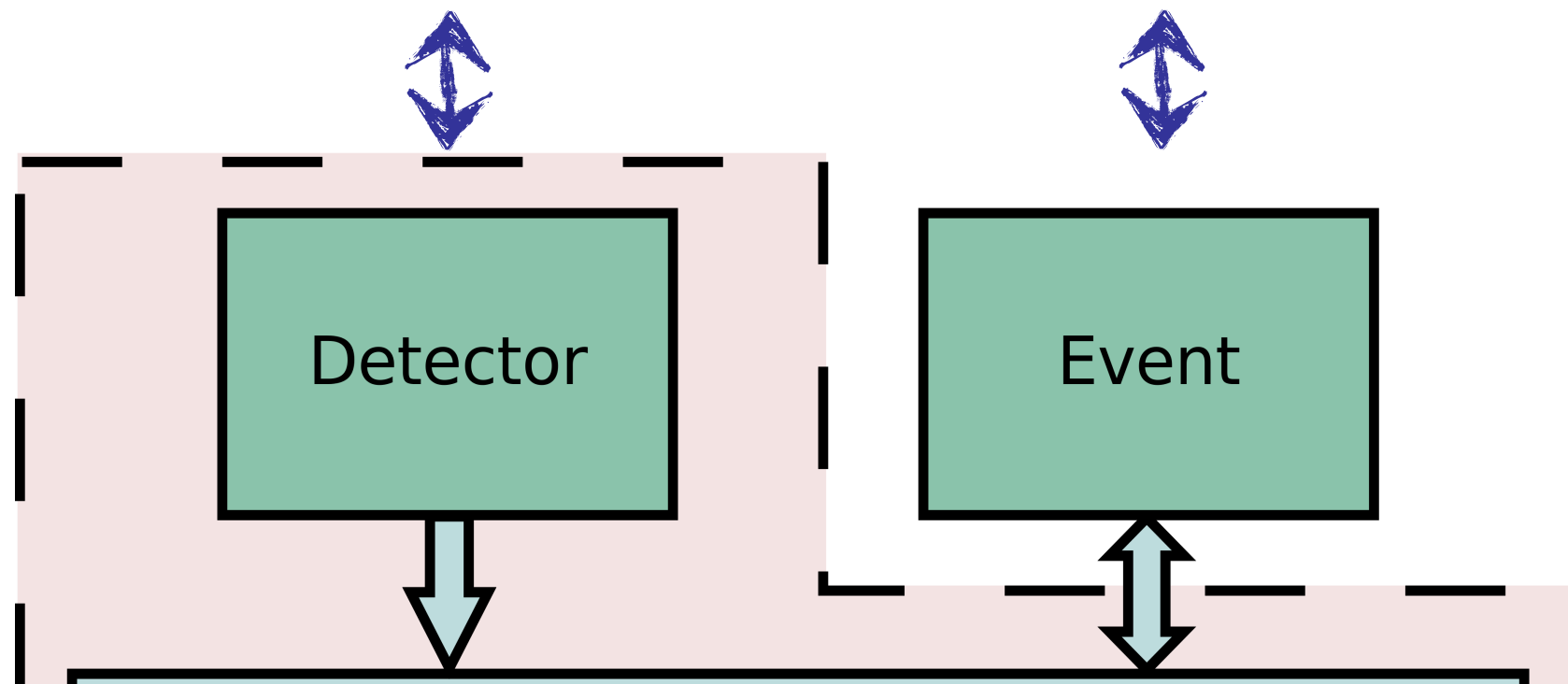
- Read/Write
- Fast changing data



Recall: building blocks

- Information from
 - Monitoring
 - Calibration
 - Survey
- Provided in
 - XML files
 - databases

- Created by
 - DAQ
 - Simulation
- Filled by processing



The Event

- The backbone data structure
 - Accumulate information during processing
 - Information exchanged by modules
- Rigid structure
 - Error checking at compile time
 - named enumerations
 - Fixed: modules know what can be in the event
 - Plug-and-play replacement of modules more likely to work
- Flexible white-board structures require conventions
 - Automated checking is harder

The naming of components

- Parallel names in detector and event
 - `sdet::Station` vs `sevt::Station`
 - Namespaces to avoid conflicts
- Regular naming of sim/raw/rec data
 - Raw data directly accessible
 - `sevt::PMT::GetFADCTrace(...)`
 - Others:
 - `MyComponet.GetXYZData()`
 - `sevt::PMT::GetSimData()`
 - `sevt::PMT::GetRecData()`
 - `sevt::PMT::GetCalibData()`
- Different interface protocols for sub-components

Protocols for sub-components

● Always present

- `Parent::GetChild()`
 - `Event::GetHeader()`

● Optional, e.g., simulated data

- `Parent::GetChild()`
 - Throw exception if not existent
- `Parent::HasChild()`
- `Parent::MakeChild()`

● Collection of sub-components

- `Parent::GetChild(someId)`
- `Parent::HasChild(someId)`
- `Parent::MakeChild(someId)`
- Iterator: `Parent::ChildrenBegin()` / `Parent::ChildrenEnd()`

Example: from Framework/FEvent/Eye.h

```
/// Trigger data for this eye
fevt::EyeTriggerData& GetTriggerData();
const fevt::EyeTriggerData& GetTriggerData() const;
void MakeTriggerData();
bool HasTriggerData() const { return fTriggerData; }

/// Reconstructed data for this eye
fevt::EyeRecData& GetRecData();
const fevt::EyeRecData& GetRecData() const;
void MakeRecData();
bool HasRecData() const { return fRecData; }

/// Number of telescopes that participated in the event
unsigned int GetNActiveTelescopes() const { return fTelescopes.size(); }

/// Retrieve Telescope by Id, throw exception if not existent
Telescope& GetTelescope(const unsigned int telescopeId,
                       const ComponentSelector::Status status = ComponentSelector::eHasData);
const Telescope& GetTelescope(const unsigned int telescopeId,
                              const ComponentSelector::Status status = ComponentSelector::eHasData) const;
/// Make Telescope telescopeId
void MakeTelescope(const unsigned int telescopeId,
                  const ComponentSelector::Status status = ComponentSelector::eHasData);
/// Check if the telescope is in the event
bool HasTelescope(const unsigned int telescopeId,
                  const ComponentSelector::Status status = ComponentSelector::eHasData) const;

/// iterator pointing to first available tel of given status
TelescopeIterator TelescopesBegin(const ComponentSelector::Status status)
{ return TelescopeIterator(ComponentSelector(status), AllTelescopesBegin(), AllTelescopesEnd()); }
/// first available tel of status eHasData (DEPRECATED)
TelescopeIterator TelescopesBegin();
```

Example: from Framework/FEvent/Eye.h

```
/// Trigger data for this eye
fevt::EyeTriggerData& GetTriggerData();
const fevt::EyeTriggerData& GetTriggerData() const;
void MakeTriggerData();
bool HasTriggerData() const { return fTriggerData; }
```

```
/// Reconstructed data for this eye
fevt::EyeRecData& GetRecData();
const fevt::EyeRecData& GetRecData() const;
void MakeRecData();
bool HasRecData() const { return fRecData; }
```

Get/Make/Has

```
/// Number of telescopes that participated in the event
unsigned int GetNActiveTelescopes() const { return fTelescopes.size(); }

/// Retrieve Telescope by Id, throw exception if not existent
Telescope& GetTelescope(const unsigned int telescopeId,
                        const ComponentSelector::Status status = ComponentSelector::eHasData);
const Telescope& GetTelescope(const unsigned int telescopeId,
                              const ComponentSelector::Status status = ComponentSelector::eHasData) const;

/// Make Telescope telescopeId
void MakeTelescope(const unsigned int telescopeId,
                  const ComponentSelector::Status status = ComponentSelector::eHasData);

/// Check if the telescope is in the event
bool HasTelescope(const unsigned int telescopeId,
                  const ComponentSelector::Status status = ComponentSelector::eHasData) const;

/// iterator pointing to first available tel of given status
TelescopeIterator TelescopesBegin(const ComponentSelector::Status status)
{ return TelescopeIterator(ComponentSelector(status), AllTelescopesBegin(), AllTelescopesEnd()); }
/// first available tel of status eHasData (DEPRECATED)
TelescopeIterator TelescopesBegin();
```

Example: from Framework/FEvent/Eye.h

```
/// Trigger data for this eye
fevt::EyeTriggerData& GetTriggerData();
const fevt::EyeTriggerData& GetTriggerData() const;
void MakeTriggerData();
bool HasTriggerData() const { return fTriggerData; }
```

```
/// Reconstructed data for this eye
fevt::EyeRecData& GetRecData();
const fevt::EyeRecData& GetRecData() const;
void MakeRecData();
bool HasRecData() const { return fRecData; }
```

```
/// Number of telescopes that participated in the event
unsigned int GetNActiveTelescopes() const { return fTelescopes.size(); }
```

```
/// Retrieve Telescope by Id, throw exception if not existent
Telescope& GetTelescope(const unsigned int telescopeId,
                       const ComponentSelector::Status status = ComponentSelector::eHasData);
const Telescope& GetTelescope(const unsigned int telescopeId,
                              const ComponentSelector::Status status = ComponentSelector::eHasData) const;
```

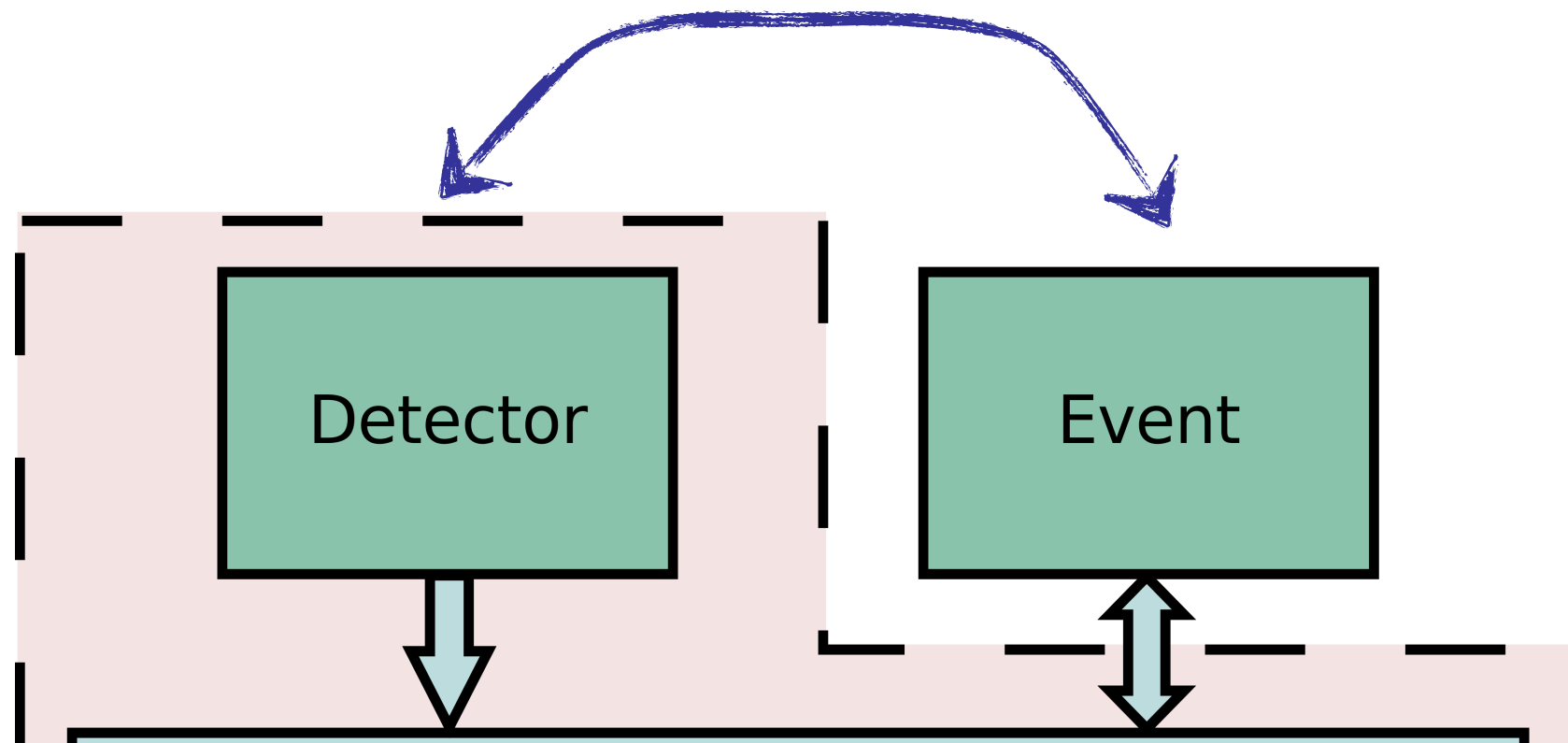
```
/// Make Telescope telescopeId
void MakeTelescope(const unsigned int telescopeId,
                  const ComponentSelector::Status status = ComponentSelector::eHasData);
/// Check if the telescope is in the event
bool HasTelescope(const unsigned int telescopeId,
                  const ComponentSelector::Status status = ComponentSelector::eHasData) const;
```

```
/// iterator pointing to first available tel of given status
TelescopeIterator TelescopesBegin(const ComponentSelector::Status status)
{ return TelescopeIterator(ComponentSelector(status), AllTelescopesBegin(), AllTelescopesEnd()); }
/// first available tel of status eHasData (DEPRECATED)
TelescopeIterator TelescopesBegin();
```

Get/Make/Has/iterator

Recall: building blocks

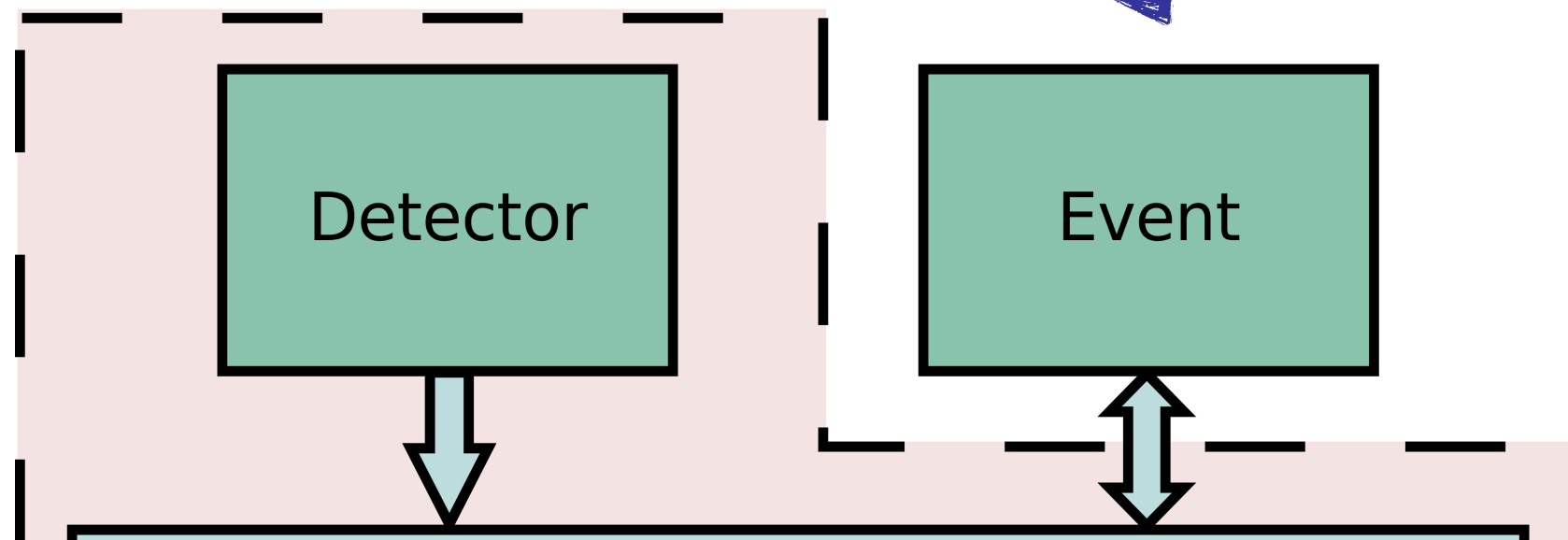
- Parallel hierarchies
- Have to be synchronized
 - Detector has to be configured for the event(s) to be processed
- Correct time
- Correct data sources
- Automatic detector update when reading event



Recall: building blocks

- Parallel hierarchies
- Have to be synchronized
 - Detector has to be configured for the event(s) to be processed
- Correct time
- Correct data sources
- Automatic detector update when reading event

`det::Detector::Update(timestamp)`



Detector description

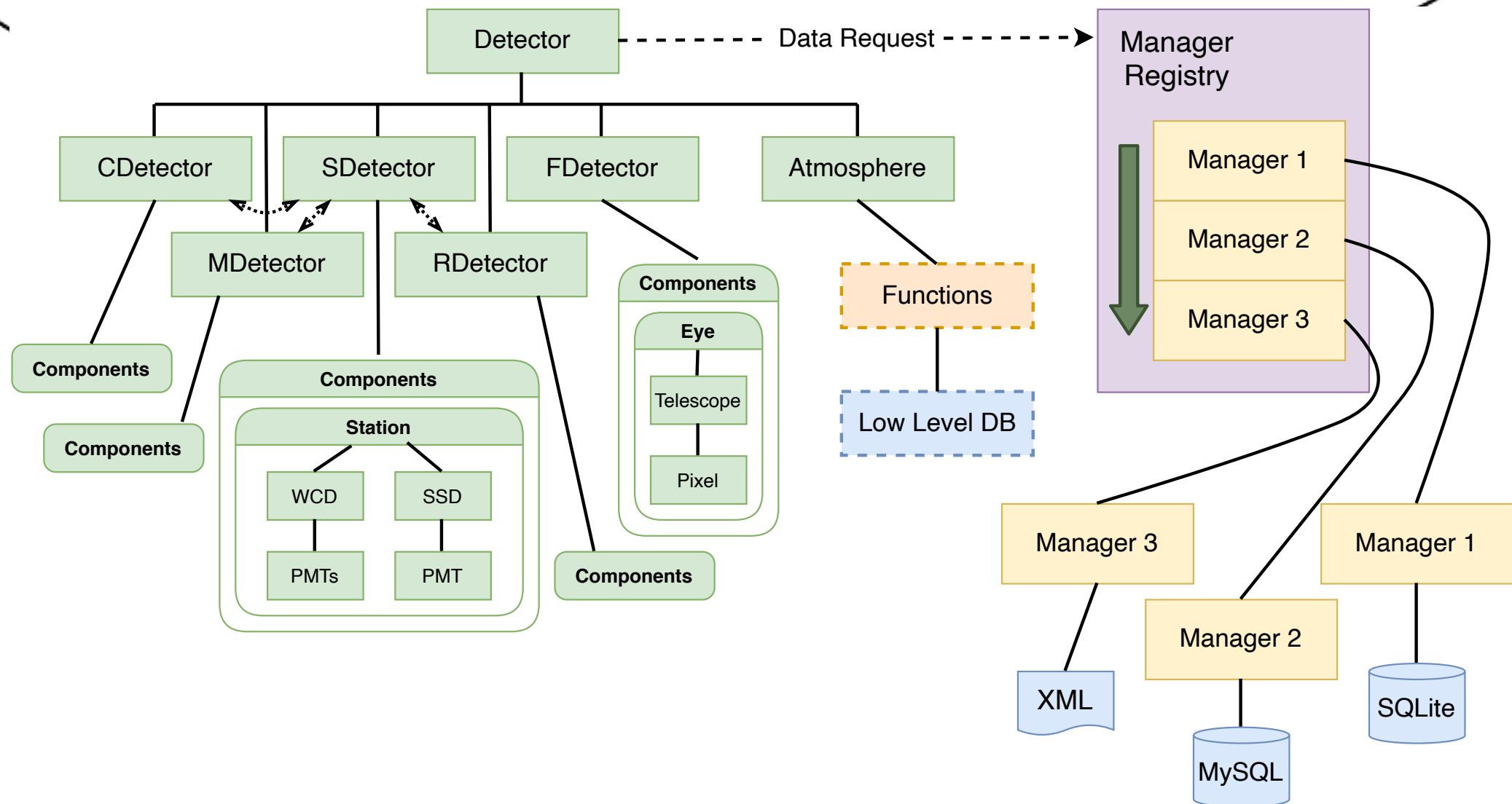
- Provide a **single, simple-to-use interface** to all information on detector properties and performance
- Access to data in **different formats**
 - MySQL & SQLite databases
 - XML files
- **Consolidate detector information** in one place
 - Discourage storing detector parameters in private configuration files or hardwiring numbers (error prone)
 - It's much easier to compare modules that (theoretically) do the same thing if they are using the same data source !
- Reasonably **efficient** : load and cache detector data as required

Detector description

Data in any format accessed through one **user interface**

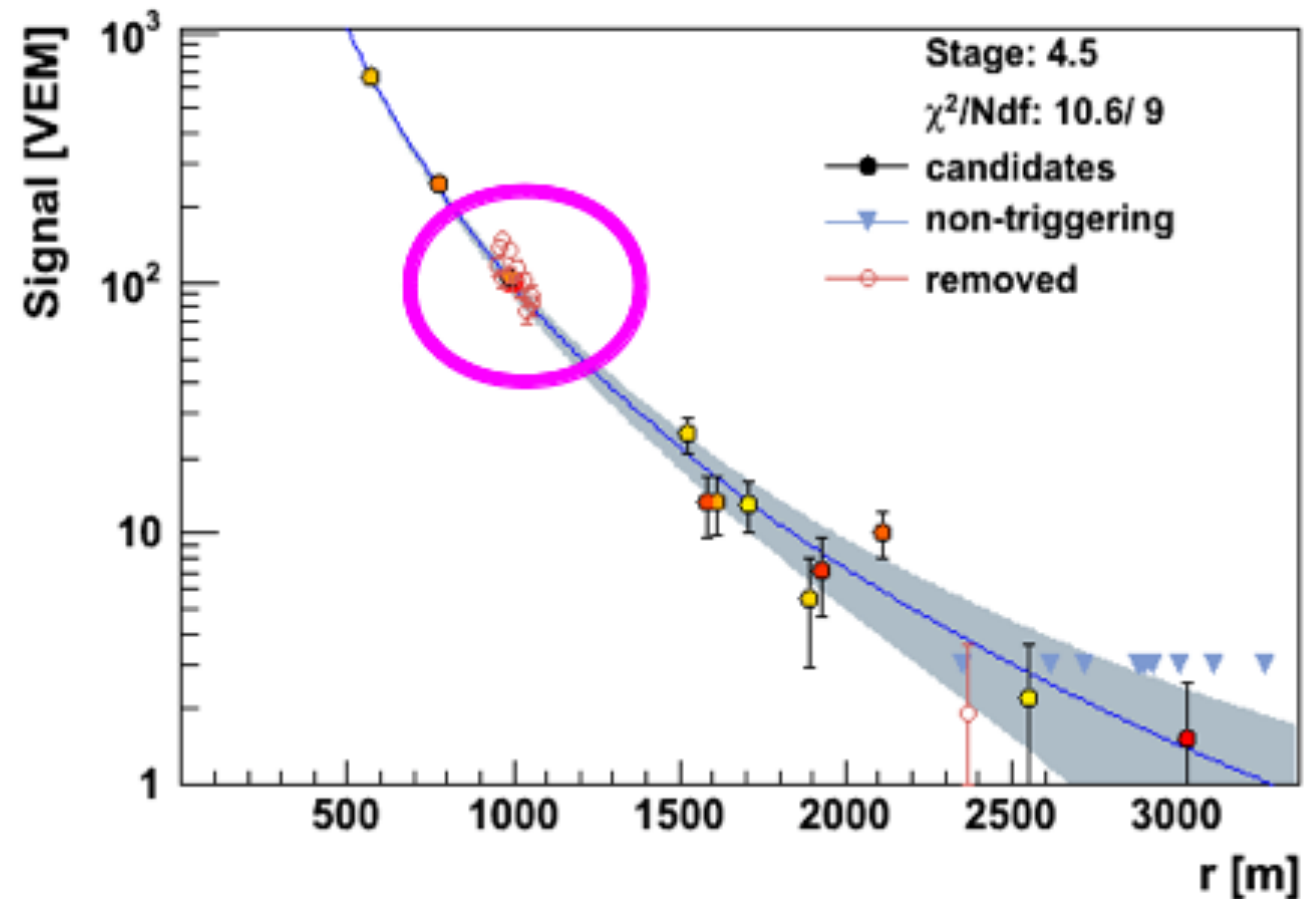
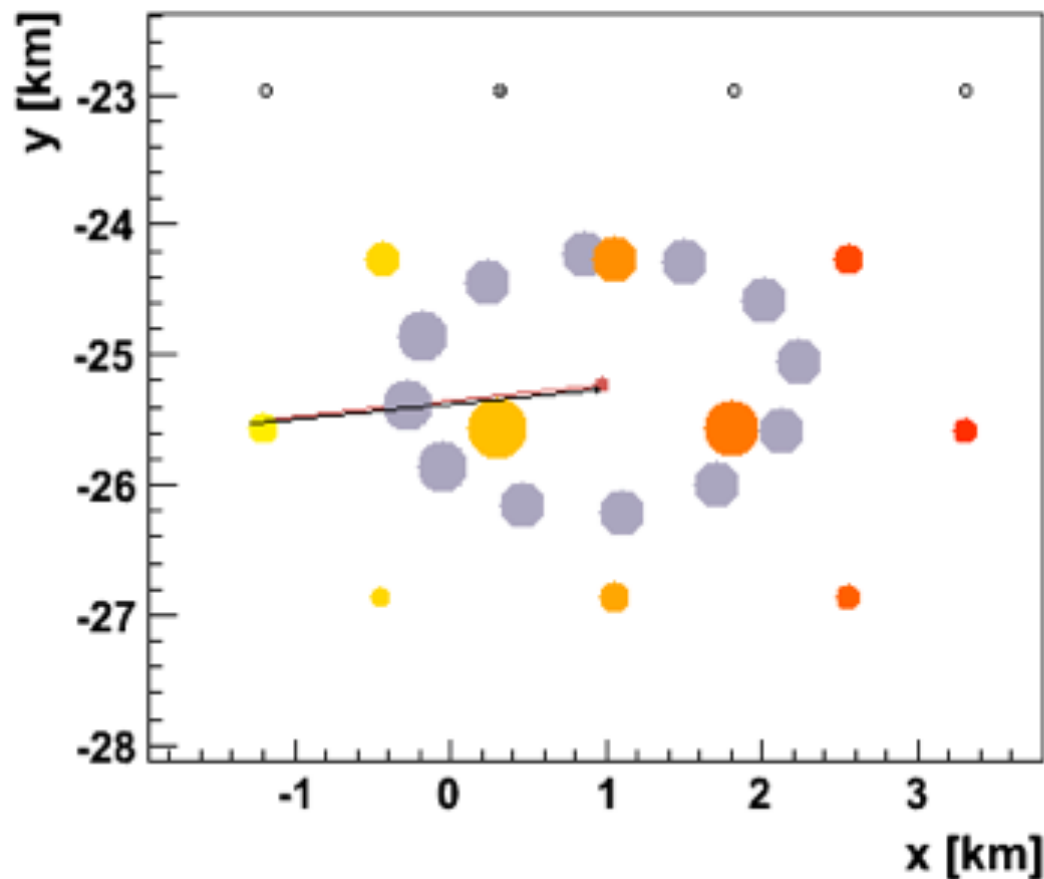
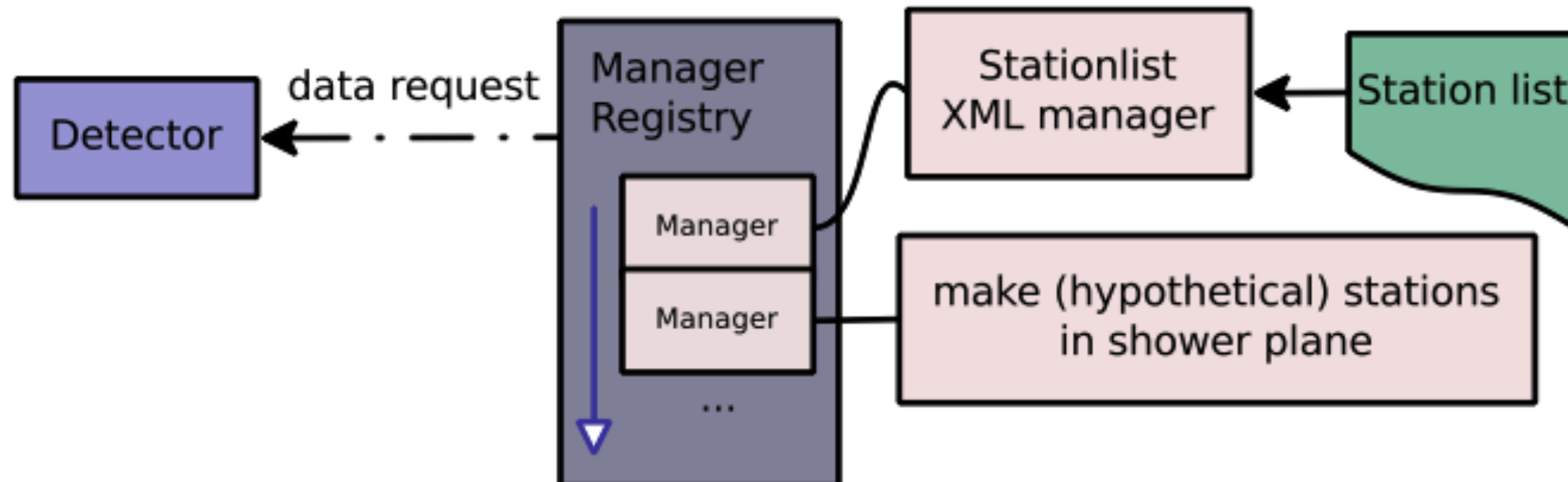
det::Detector::GetInstance()

Configurable **manager registry** specifies which managers are to be used for a given application.



Managers provide flexibility

SD sim example : “Dense” array of stations at 1000m



Examples

```
// Get detector instance. Detector is a singleton, hence GetInstance().  
// Note the detector is logically const. You don't change the Detector,  
// (but the managers do)  
const det::Detector& detector = det::Detector::GetInstance();
```

```
// Get a handle to the station. Note const references. C'tors are private  
// to prevent accidental copying and orphaned components  
const sdet::Station& station = detector.GetSDetector().GetStation(157);
```

```
// reads from an XML detector simulation parameters file  
TabulatedFunction w = station.GetWaterAbsorptionLength();
```

```
// reads from XML station list, which lists stations in UTM coordinates  
// UI converts from UTM and loads position into a Point object.  
Point pos = station.GetPosition();
```

```
// reads position and creates a local coordinate system  
CoordinateSystemPtr stationCS = station.GetLocalCoordinateSystem();
```

```
// reads calib vs. wavelength from FD calibration database  
Pixel& p = detector.GetFDetector().GetEye(LosLeones).GetPixel(100);  
TabulatedFunction cal = p.GetEndToEndCalibration();
```

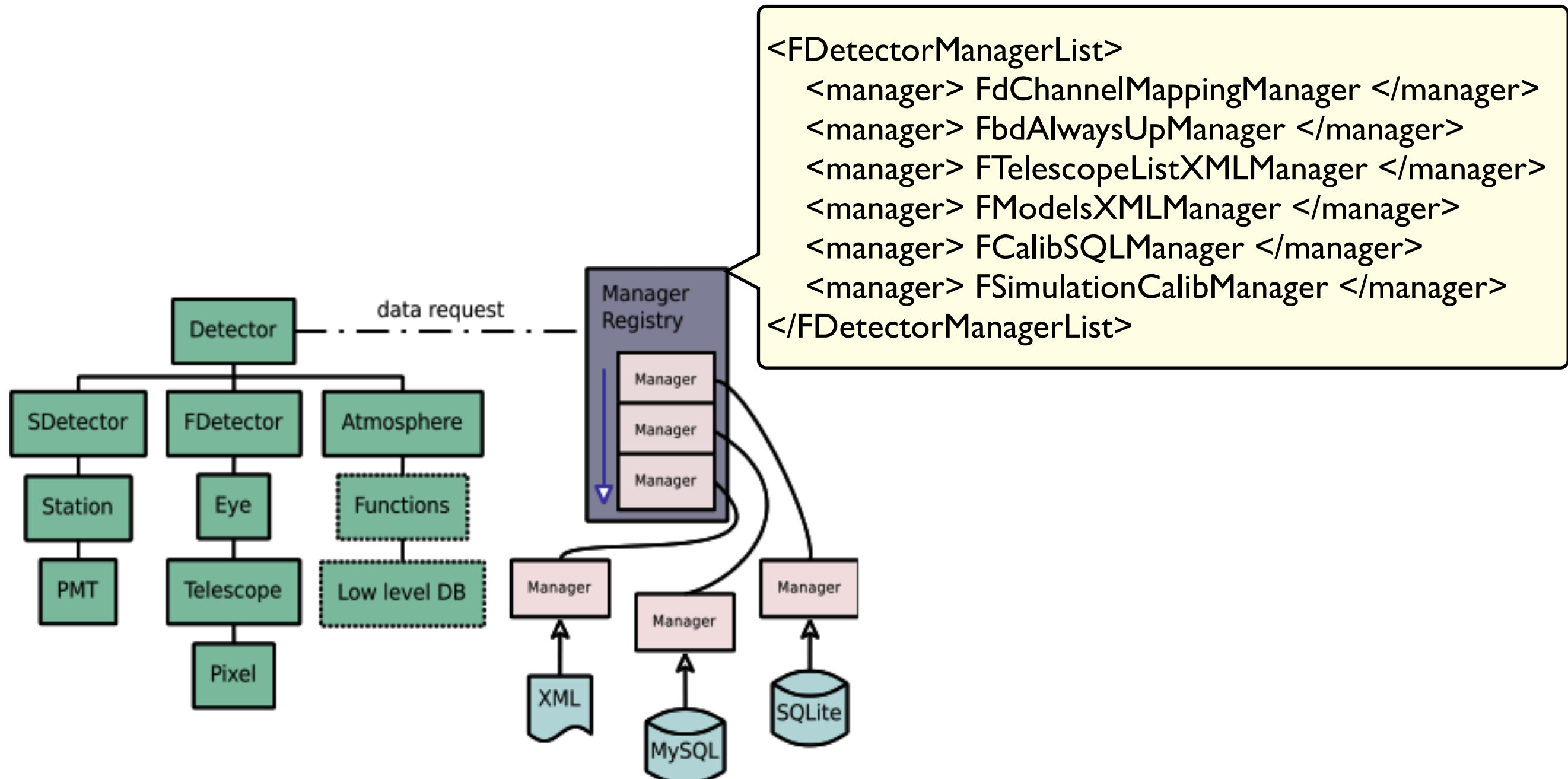
Shortcuts

- It is possible to look up detector components from the corresponding event component.

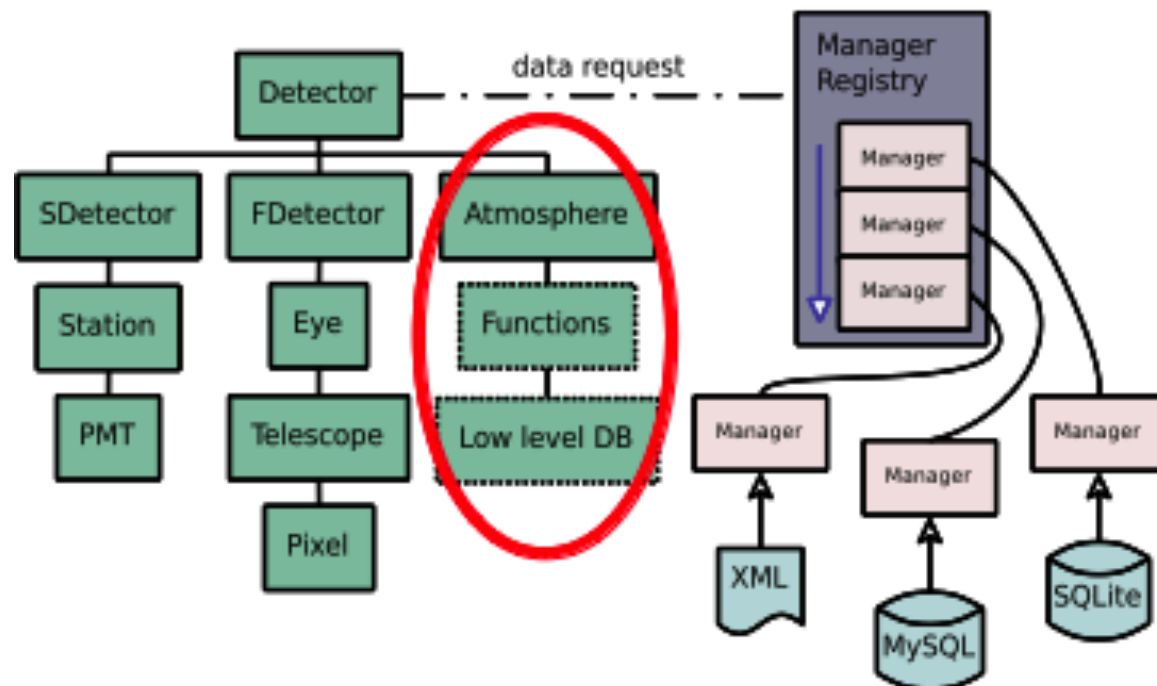
```
for (sevt::SEvent::ConstStationIterator sIt = sevent.StationsBegin();  
    sIt != sevent.StationsEnd(); sIt++) {  
  
    const sdet::Station& detStation =  
        det::Detector::GetInstance().GetSDetector().GetStation(*sIt);  
}
```

Setting up the managers

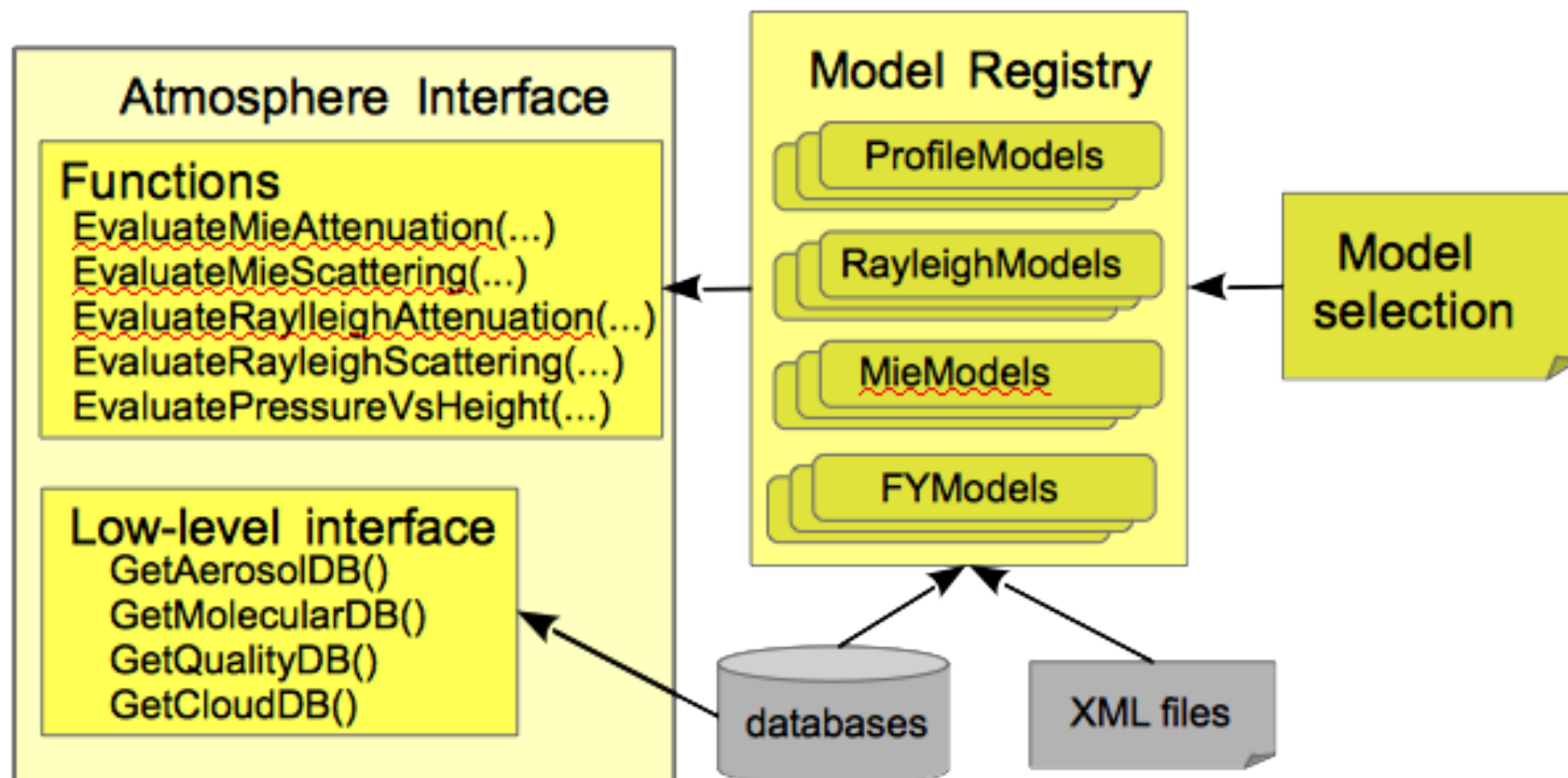
- Managers are registered in a ManagerRegistry
- An XML config file is used to select which managers to use
- Order of queries follows order of managers



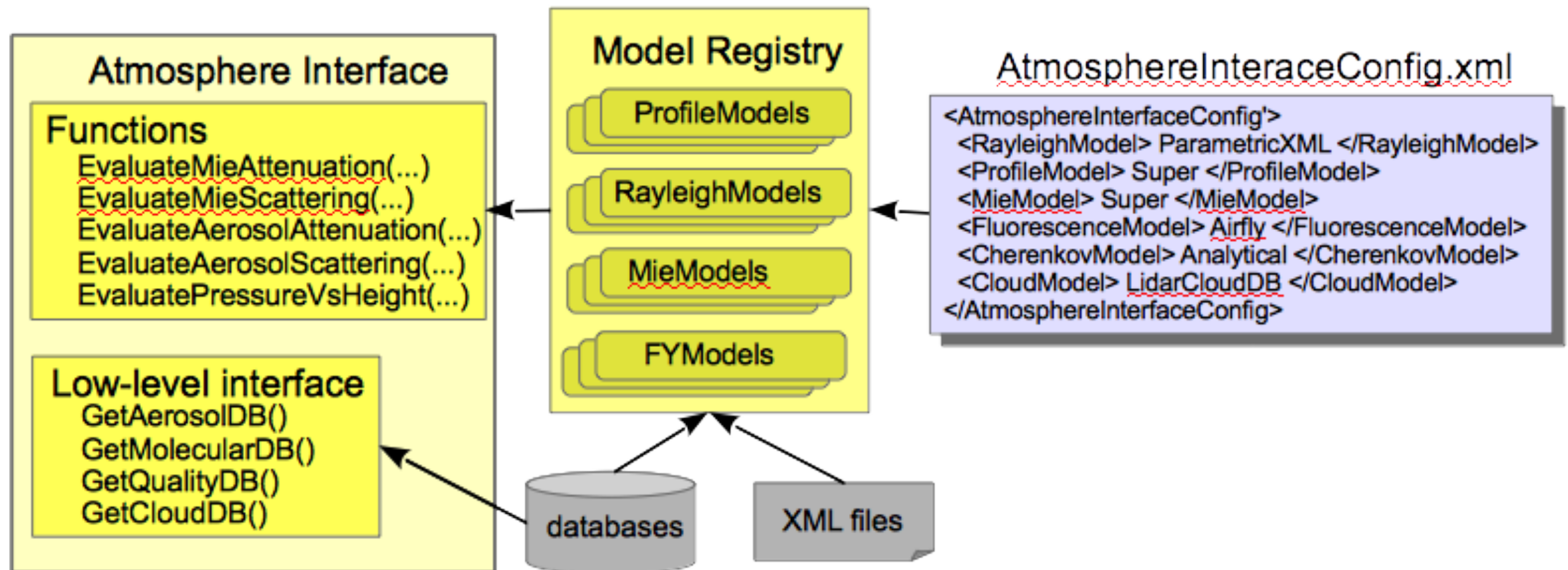
Access to Atmosphere data



Two access layers :
Low-level DB access via managers
and High-level functions via “models”
which interpret low-level data



High-level atmosphere functions



```

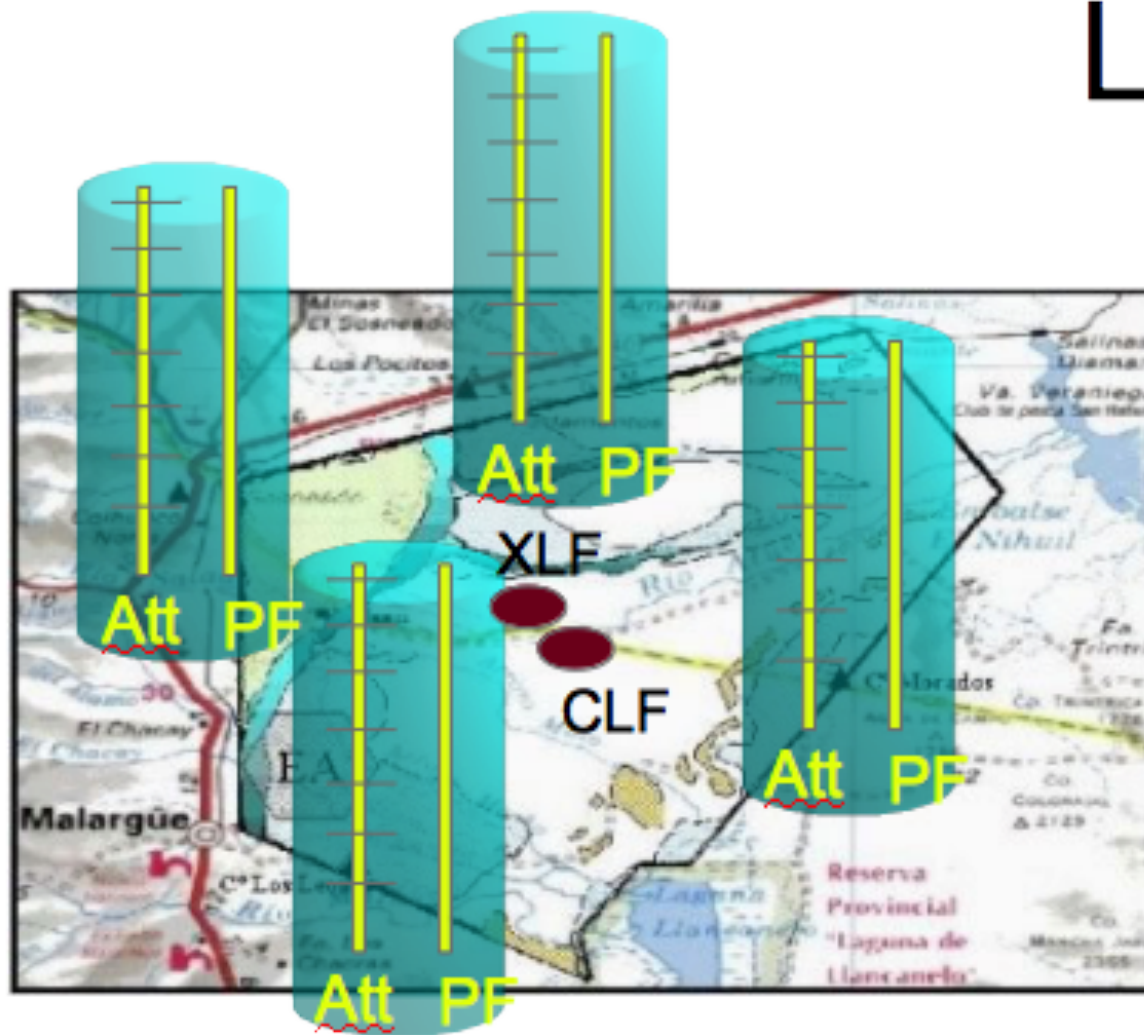
CoordinateSystemPtr CS(Detector::GetInstance().GetFDetector().
                      GetEye("LosLeones").GetLocalCoordinateSystem());
Point p1(0., 0., 0., CS);
Point p2(0., 0., 9000*m, CS);
vector<double> lambda = boost::assign::list_of(300*nm)(350*nm)(400*nm);

AttenuationResult att = atmosphere.EvaluateMieAttenuation(p1, p2, lambda);
double transmission = att.GetTransmissionFactor().Y(320*nm);

```

basically, a TabulatedFunction

Low-level aerosol DB access example



- Array divided into aerosol “zones”
- **Attenuation** and **phase function** stored in slices in altitude for each zone

```
const AerosolDB& aDB = Detector::GetInstance().GetAtmosphere().GetAerosolDB();
```

```
// iterate over zones
```

```
for (aDB::Zoneliterator zlt = aDB.ZonesBegin() ; zlt != aDB.ZonesEnd() ; ++zlt){
```

```
    // iterate over attenuation length slices in the zone
```

```
    for (AerosolZone::AttSliceliterator attlt = zlt->AttSlicesBegin() ;
```

```
        attlt != AttSlicesEnd() ; ++attlt) {
```

```
        double attenuationLength = attlt->GetAttLength();
```

```
    }
```

```
}
```