Global Muon Trigger in CMSSW

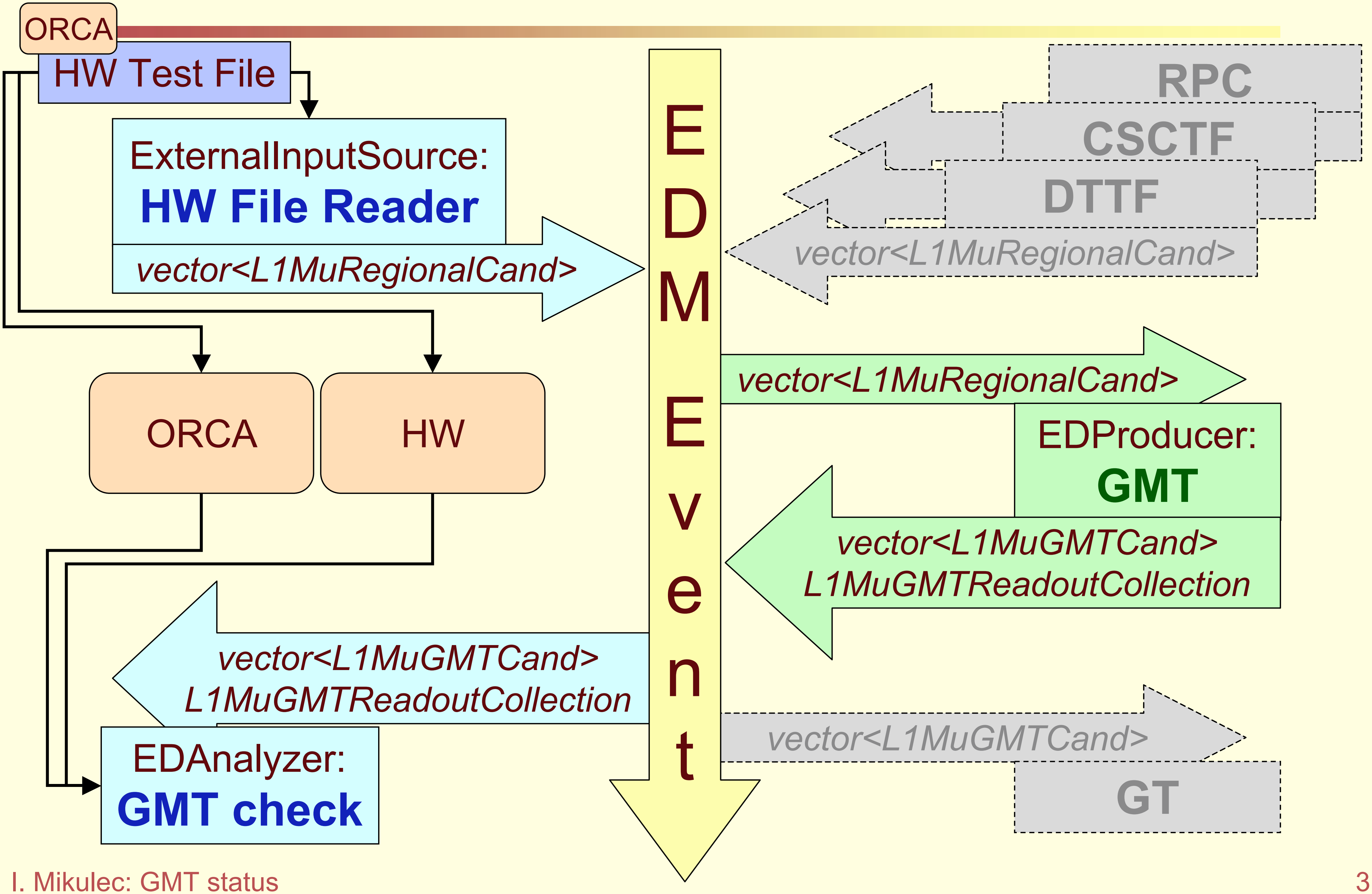
Ivan Mikulec
HEPHY Vienna

CMS Online Selection meeting
22 June 2006

Status of GMT emulator in CMSSW

- A fully working standalone GMT emulator was committed to CMSSW. It is using interface classes - ready to be run in the full framework.
- The GMT emulator code resides in L1Trigger/GlobalMuonTrigger. It contains also a gmt.cfi file with the default GMT configuration and some .cfg files to run and test in the standalone mode with sample data.
- The interface classes reside in: DataFormats/L1GlobalMuonTrigger. These have been tagged put into nightly builds and scheduled for the 0_8_0 prerelease - can be referenced by other systems.

Scheme of GMT emulator in CMSSW



ORCA

HW Test File

ExternalInputSource:
HW File Reader

vector<L1MuRegionalCand>

EDM Event

RPC

CSCTF

DTTF

vector<L1MuRegionalCand>

vector<L1MuRegionalCand>

EDProducer:
GMT

vector<L1MuGMTCand>
L1MuGMTReadoutCollection

ORCA

HW

vector<L1MuGMTCand>
L1MuGMTReadoutCollection

EDAnalyzer:
GMT check

vector<L1MuGMTCand>

GT

Functions

- Functions of the GMT code in CMSSW:
  - Simulate GMT response (in the MC framework)
  - Emulate GMT response (in the HLT fw or standalone)
  - Generate GMT LUTs
- Functions (at present) of the GMT Data Formats
  - Emulate inter-module communication (TFs-GMT-GT)
  - Provide SW representation of the GMT DAQ data (data stored in the bit-coded format)
  - Provide access to individual bit fields (phi, eta, pt - integer)
  - Provide access to physical representation of the bit fields (needs trigger scales - now part of data formats)

Trigger scales issue

➔ Definitely trigger scales will have to reside in the database because they are:

- needed by online (TS)
- needed by CMSSW
- they might change in time (need validity intervals).

➔ Database is accessed in CMSSW through the Event Setup. There are two possibilities:

- Data formats provide access to physical representation as it is now but how will the database access be provided (no a priori pointer to the Event Setup)?
- Separate physical and HW representations as proposed by Werner, create an extra EDProducer (has access to Event Setup) and make physical representation persistent (for the HLT and user)

GMT Emulator
Output Data Formats
(present status)

L1MuGMTReadoutCollection

class **L1MuGMTReadoutCollection** contains GMT readout records (RR) for the triggered and surrounding BXs.

Methods:

```
L1MuGMTReadoutRecord const& getRecord() const;
```

get the GMT RR for the triggered BX.

```
L1MuGMTReadoutRecord const& getRecord(int bx) const;
```

get the GMT RR for a given BX.

```
vector<L1MuGMTReadoutRecord>  getRecords() const;
```

get all GMT RRs.

L1MuGMTReadoutRecord

class **L1MuGMTReadoutRecord** contains full DAQ record of GMT for a given BX. This includes full info inputs, output and intermediate results.

Methods:

```
int getBxCounter() const;
vector<L1MuGMTExtendedCand> getGMTCands() const;
vector<L1MuGMTExtendedCand> getGMTBrlCands() const;
vector<L1MuGMTExtendedCand> getGMTFwdCands() const;
vector<L1MuRegionalCand> getDTBXCands() const;
vector<L1MuRegionalCand> getCSCCands() const;
vector<L1MuRegionalCand> getBrlRPCCands() const;
vector<L1MuRegionalCand> getFwdRPCCands() const;
unsigned getMIPbit(int eta, int phi) const;
unsigned getQuietbit(int eta, int phi) const;
```

L1MuGMTExtendedCand

class **L1MuGMTExtendedCand** derives from the **L1MuGMTCand**. In addition it gives access to the sort rank and the origin of a GMT muon candidate.

Methods:

unsigned int `rank()` **const;** - get rank

unsigned `getDTCSCIndex()` **const;** - get DT/CSC muon index

unsigned `getRPCIndex()` **const;** - get RPC muon index

bool `isFwd()` **const;** - forward=true, barrel=false

bool `isRPC()` **const;** - unmatched RPC=true

L1MuGMTCand

class **L1MuGMTCand** contains the actual information about the GMT muon candidates as needed and used by the GT and (for now) provides access to the physical quantities.

Main methods:

int `bx()` **const;** - get the bx number

unsigned int `phi()` **const;** - get bit code of phi

float `phiValue()` **const;** - get phi in radians

unsigned int `eta()` **const;** - get bit code of eta

float `etaValue()` **const;** - get real eta value

unsigned int `pt()` **const;** - get bit code of pt

float `ptValue()` **const;** - get pt in GeV

unsigned int `quality()` **const;** - get quality code

int `charge()` **const;** - get charge

bool `isol()` **const;** - get the isolation bit

bool `mip()` **const;** - get the mip bit