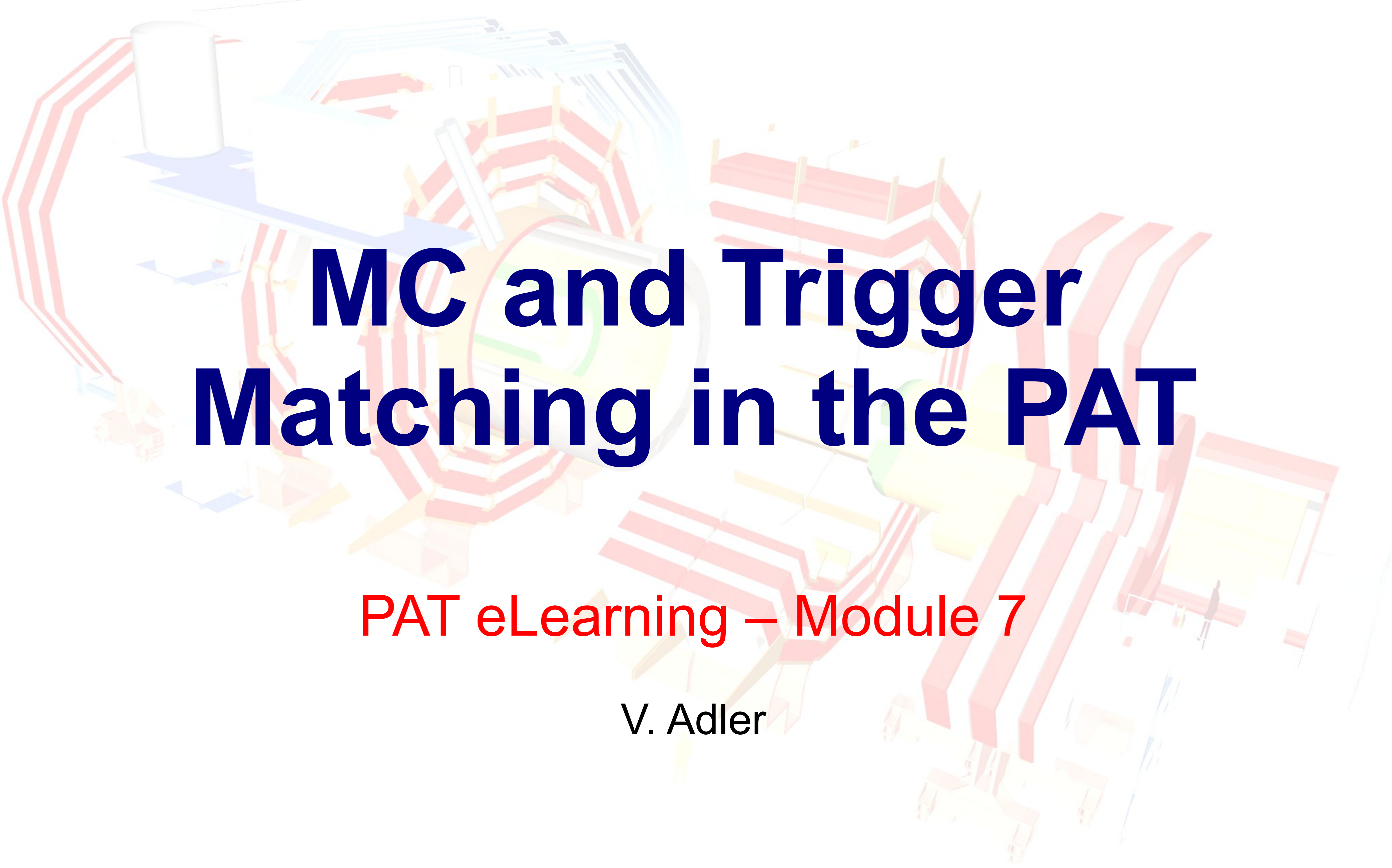
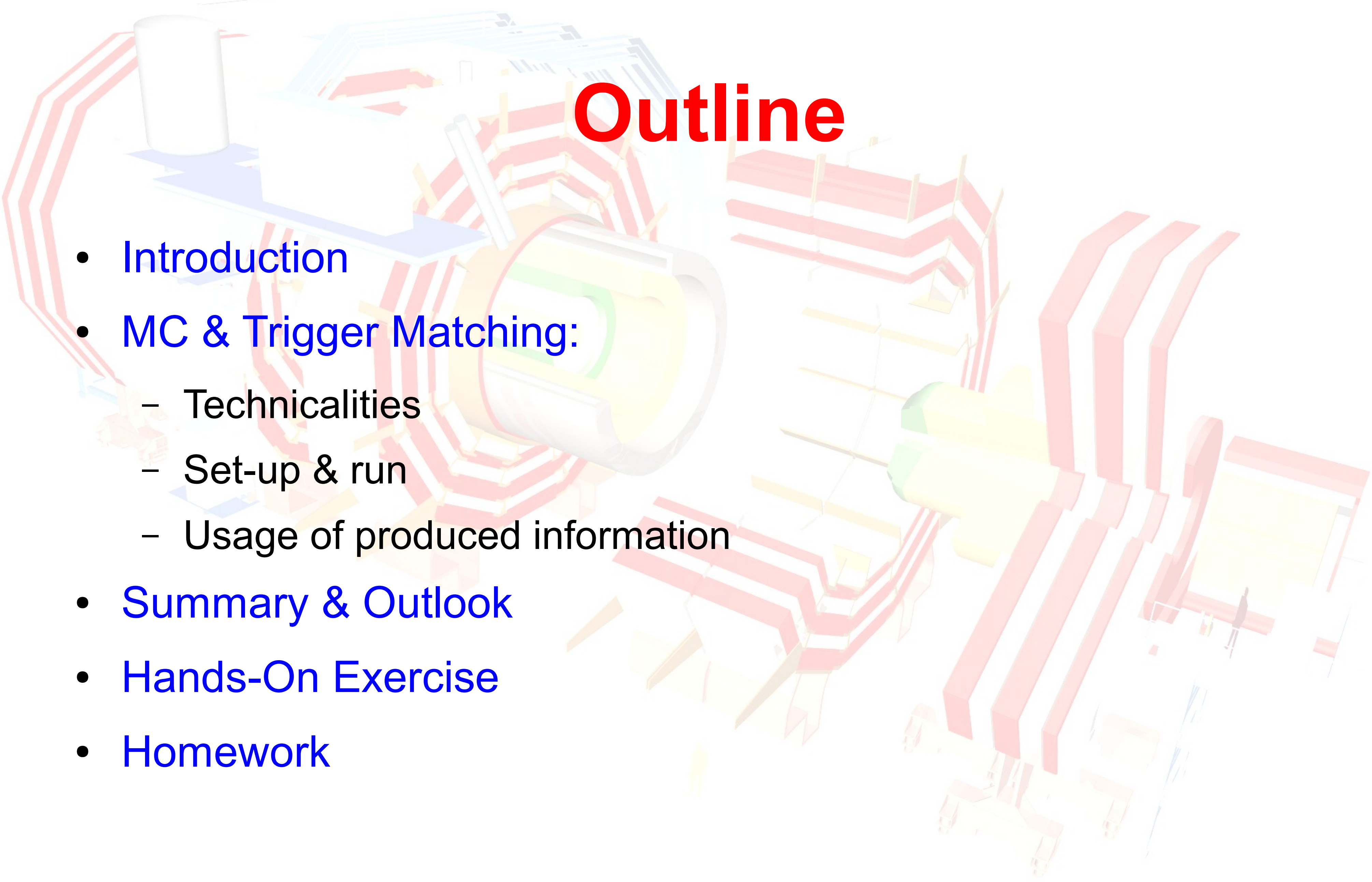


# MC and Trigger Matching in the PAT

## PAT eLearning – Module 7

V. Adler

# Outline



- Introduction
- MC & Trigger Matching:
  - Technicalities
  - Set-up & run
  - Usage of produced information
- Summary & Outlook
- Hands-On Exercise
- Homework

# What is "matching"?

- "Matching" means the association of objects based on their similarity in spatial coordinates and/or kinematics.
- Discrete object properties can restrict possible matches, like e.g.:
  - object IDs/types
  - charges
- Goal is to find representations of the **same** object in **different** collections.

# What is matching good for?

- MC matching:
  - validate reconstruction
  - check object/event selections in analysis
- Trigger matching
  - compare trigger and full reconstruction
  - check object/event selection in analysis

# How is matching generally performed in the PAT?

- Currently, any matching is split into steps over both PAT layers:
  - **layer 0:**
    - produce object collection (with base class `reco::Candidate`) to match to, if necessary (***`EDProducer`*** modules)
    - perform the match (***`EDFilter`*** modules) and store it in an `edm::Association`
  - **layer 1:**
    - store the matching objects in the PAT objects (***`EDProducer`*** modules):
      - either as reference to original object, which also has to be stored then
      - or by "embedding" as data member

# How is the matching implemented?

- One central class template is used: `reco::PhysObjectMatcher`
- Particular matcher modules are concrete instances of this template.
- Such an instance specifies:
  - input collection types to be matched
  - a (pre-)selector
  - matching definition
  - ranking
- Several instances used for matching in PAT.

# MC matching

- MC matching associates generator level objects to PAT objects.
- Generator level objects are:
  - generator particles
  - generator level jets
- Most generator object collections are in AOD.
  - exceptions:
    - jets from taus
    - jets form other than the default jet reonstruction algorithm
- The number of collections is small, so a comprehensive default can be provided by the PAT.

# How is the MC matching set up?

- Individual configuration files for different object types in `PhysicsTools/PatAlgos/python/mcMatchLayer0/`
- Sequence definitions in file `mcMatchSequences_cff.py` in the same directory
- PAT objects producer modules in `PhysicsTools/PatAlgos/python/producersLayer1/`

# How is the MC matching set up for generator particles?

- Used modules are the EDProducers **`MCMatcher`** or **`MCMatcherByPt`** with the following configurables:
  - ***`InputTag src`***:
    PAT object collection label (**`reco::CandidateView`**)
  - ***`InputTag matched`***:
    MC object collection label (**`reco::GenParticleCollection`**)
  - ***`vint32 mcPdgId`***:
    PDG Ids of pyrticle types to match
  - ***`vint32 mcStatus`***:
    PYTHIA style status code (1: stable, 2: shower, 3: hard scattering)
  - ***`bool checkCharge`***:
    only equally charged objects matched, if set to ***True***

# How is the MC matching set up for generator particles?

- `double maxDeltaR`:
  maximum distance in η-φ space to apply match
- `double maxDPtRel`:
  maximum difference in rel. pt to apply match
- `bool resolveAmbiguities`:
  only one PAT object is matched, if set to ***True***
- `bool resolveByMatchQuality`:
  if **`resolveAmbiguities=`*****True***, choose best match rather then first

# How is the MC matching set up for generator particles?

- Example configuration for electrons:

```
electronMatch = cms.EDFilter("MCMatcher",
    src     = cms.InputTag("allLayer0Electrons"),
    matched = cms.InputTag("genParticles"),
    mcPdgId     = cms.vint32(11),
    checkCharge = cms.bool(True),
    mcStatus    = cms.vint32(1),
    maxDeltaR = cms.double(0.5),
    maxDPtRel = cms.double(0.5),
    resolveAmbiguities    = cms.bool(True),
    resolveByMatchQuality = cms.bool(False)
)
```

# How is the MC matching set up for generator level jets?

- Used module is the `EDProducer` `GenJetMatcher` with the following configurables differently used compared to `MCMatcher`:
  - *`InputTag`* `matched`:
    the label of the MC jet collection (`reco::GenJetCollection`)
  - *`vint32`* `mcPdgId`:
    no meaning here, remains empty
  - *`vint32`* `mcStatus`:
    no meaning here, remains empty
  - *`bool`* `checkCharge`:
    no meaning here, remains empty

# How is the MC matching set up for generator level jets?

- Example configuration for jets:

```
jetGenJetMatch = cms.EDFilter("GenJetMatcher",
    src         = cms.InputTag("allLayer0Jets"),
    matched     = cms.InputTag("iterativeCone5GenJets"),
    mcPdgId        = cms.vint32(),       # n/a
    mcStatus       = cms.vint32(),       # n/a
    checkCharge = cms.bool(False),    # n/a
    maxDeltaR = cms.double(0.4),
    maxDPtRel = cms.double(3.0),
    resolveAmbiguities    = cms.bool(True),
    resolveByMatchQuality = cms.bool(False)
)
```

# How is the MC matching included into the PAT workflow?

- Sequences of MC matches are defined in `PhysicsTool/PatAlgos/python/mcMatchLayer0/mcMatchSequences_cff.py`
- This is imported into `PhysicsTool/PatAlgos/python/patLayer0_cff.py`
- All matches are scheduled there after the PAT layer 0 cleaners in order to provide all needed input collections, e.g.:

```
[...]
from PhysicsTools.PatAlgos.mcMatchLayer0.mcMatchSequences_cff import *
patLayer0_withoutTrigMatch = cms.Sequence(
        patBeforeLevel0Reco *
        patLayer0Cleaners   *
        patHighLevelReco    *
        patMCTruth              # MC matching sequence
)
```

# How is the MC matching included into the PAT workflow?

- Special treatment of MC jets from taus:
  - not in AOD
  - need inclusion of reconstruction before matching, e.g. in PhysicsTool/PatAlgos/python/mcMatchLayer0/mcMatchSequences_cff.py

```
from PhysicsTools.JetMCAlgos.TauGenJets_cfi import tauGenJets
patMCTruth_Tau = cms.Sequence (
     [...]
     tauGenJets      *    # produces MC jets from taus
     tauGenJetMatch       # takes 'tauGenJets' as parameter matched
)
```

# How are matched MC objects added to PAT objects?

- MC matches are stored in two different ways:
  - by "embedding":
    - adds the objects to data member collections of the `pat::PATObject`
    - possible for all types
  - by saving an `edm::Ref`:
    - possible only for MC particles, not jets or the MET
    - default in that case
- The addition of the MC matches is configured in the particular PAT objects (leptons, jets MET) producers in `PhysicsTools/PatAlgos/python/producersLayer1/`

# How are matched MC objects added to PAT objects?

- The particular configuration depends on the PAT object type:
  - photons, electrons, muons, taus, jets:
    - ***bool addGenMatch***:
      general switch to add MC particle match
    - ***bool embedGenMatch***:
      switch for embedding
    - ***InputTag genParticleMatch***:
      input product label, specified by the MC particle matching module

# How are matched MC objects added to PAT objects?

- taus, jets only:
  - ***bool* addGenJetMatch**:
    general switch to add MC jet match
  - ***InputTag* genJetMatch**:
    input product label, specified by the MC jet matching module
- MET only (no matching is performed):
  - ***bool* addGenMET**:
    general switch to add generator MET
  - ***InputTag* genMETSource**:
    input product label of the generator MET

# How are matched MC objects added to PAT objects?

- Example configuration for electrons:

```
addGenMatch         = cms.bool(True),
embedGenMatch       = cms.bool(False),
genParticleMatch = cms.InputTag("electronMatch")
```

- Example configuration for MET:

```
addGenMET     = cms.bool(True),
genMETSource = cms.InputTag("genMet")
```

# How are matched MC objects added to PAT objects?

- Example configuration for jets:

```
addGenPartonMatch   = cms.bool(True),
embedGenPartonMatch = cms.bool(False),
genPartonMatch      = cms.InputTag("jetPartonMatch"),
addGenJetMatch    = cms.bool(True),
genJetMatch       = cms.InputTag("jetGenJetMatch")
```

# How are MC matches added to the event content?

- Configurations are in
  `PhysicsTools/PatAlgos/python/patLayer?_EventContent_cff.py`
- PAT layer 0:
  - MC objects:

```
'keep *_genParticles_*_*',
'keep *_iterativeCone5GenJets_*_*',
'keep *_tauGenJets_*_*',
'keep *_genMet_*_*',
```

  - MC matches:

```
'keep recoGenParticlesedmAssociation_*_*_*',
'keep recoGenJetsedmAssociation_*_*_*'
```

- PAT layer 1:
  - only, if MC particles are stored by reference:

```
'keep recoGenParticles_genParticles_*_*'
```

# How are MC matches used in analysis?

- The base interface is provided by `pat::PATObject`:
  - `reco::GenParticleRef genParticleRef(size_t idx=0) const;`
    get MC particle reference, index is optional
  - `reco::GenParticleRef genParticleById(int pdgId, int status) const;`
    get MC particle reference for specific PDG ID and PYTHON status
  - `const reco::GenParticle * genParticle(size_t idx=0) const;`
    get C++ pointer to MC particle
  - `size_t genParticlesSize() const;`
    number of matches
  - `std::vector<reco::GenParticleRef> genParticleRefs() const;`
    vector of references to all matches
  - further methods ("setters")

# How are MC matches used in analysis?

- Further functionalities are added by concrete PAT objects classes:
  - `pat::Lepton`
  - `pat::Photon`
  - `pat::Tau`
  - `pat::Jet`
  - `pat::MET`
- The interfaces to access information stored in the MC objects themselves are found in the classes:
  - `reco::GenParticle`
  - `reco::GenJet`
  - `reco::GenMET`

# Trigger Matching

- Trigger matching associates trigger objects to PAT objects.
- Trigger objects:
  - are physics objects reconstructed (quickly) at trigger level
  - are saved
    - only for run trigger filter modules
    - only, if they pass the filter requirements
- Trigger information is available in AOD, **but**:
  - trigger objects are **not** stored on basis of `reco::Candidate`
  - some informations are **not easy to access**, especially filter/objects-path associations
- Due to the large number of possible matches
  - the PAT provides only a small default set of trigger matches (serving as examples)
  - most probably, a desired matching has to be newly configured

# Which trigger objects can or should be matched?

- The most common question to the trigger matching is:
  **Which PAT objects let the events pass a given trigger path?**
- Examples of trigger matches can be e.g.:
  - trigger electrons to PAT electrons (s. question above)
  - trigger photons to PAT electrons
  - trigger electrons to PAT jets (fake electron triggers?)
  - trigger MET to PAT muons (fake MET triggers?)
  - trigger muons to PAT photons (who knows...?)

# How is necessary information on HLT paths and filters found?

- To answer the question, one first needs to know,
  **which filter ran in the path of interest**
  to access the correct collection.
- In **`CMSSW_2_2_3`**, tools to access this information:
  - are available
  - are not yet used in the PAT
  - are limited to L3
- To make use of these tools, a **`cmsRun`** job over one event needs to be run:
  - use the following configuration:

# How is necessary information on HLT paths and filters found?

```
import FWCore.ParameterSet.Config as cms
process = cms.Process( "HLTPROV" )
process.source = cms.Source("PoolSource",
    fileNames = cms.untracked.vstring([input file])
)
process.maxEvents = cms.untracked.PSet(input = cms.untracked.int32(1))
process.load( "HLTrigger.HLTcore.hltEventAnalyzerAOD_cfi" )
process.hltEventAnalyzerAOD.triggerName = cms.string( '@' )
process.load( "HLTrigger.HLTcore.triggerSummaryAnalyzerAOD_cfi" )

process.p = cms.Path(
    process.hltEventAnalyzerAOD       +
    process.triggerSummaryAnalyzerAOD
)
```

# How is necessary information on HLT paths and filters found?

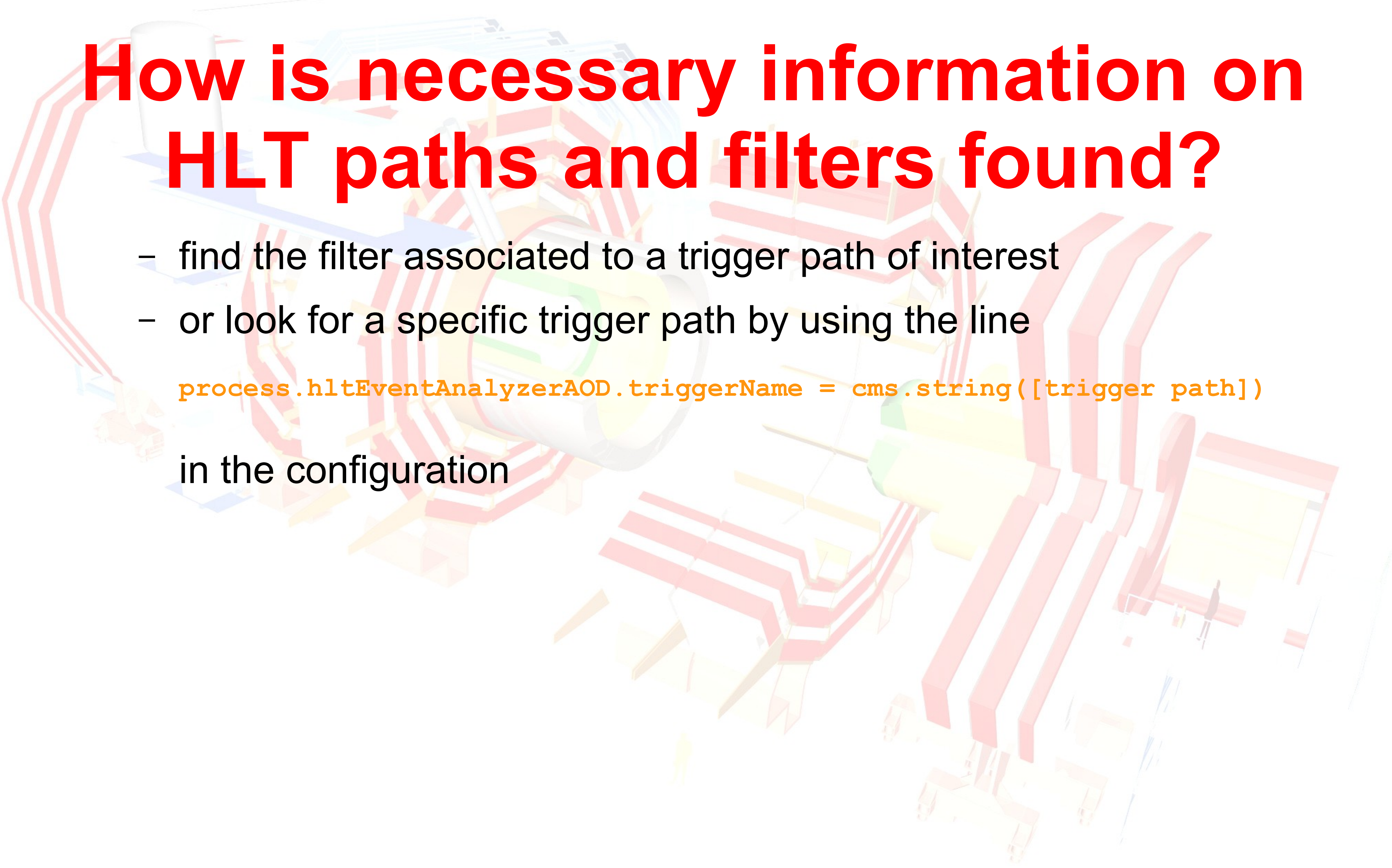
- run it through a '**grep**'-pipe:

```
cmsRun myHltAna_cfg.py | grep -B 3 "'L3' filter in slot"
```

- inspect the output, that looks like e.g.

```
[...]
--
HLTEventAnalyzerAOD::analyzeTrigger: path HLT_LooseIsoEle15_LW_L1R [47]
 Trigger path status: WasRun=1 Accept=0 Error =0
 Last active module - label/type:
hltL1NonIsoHLTLooseIsoSingleElectronLWEt15TrackIsolFilter/
HLTElectronTrackIsolFilterRegional [64 out of 0-65 on this path]
 'L3' filter in slot 64 - label/type
hltL1NonIsoHLTLooseIsoSingleElectronLWEt15TrackIsolFilter/
HLTElectronTrackIsolFilterRegional
--
[...]
```

# How is necessary information on HLT paths and filters found?



- find the filter associated to a trigger path of interest
- or look for a specific trigger path by using the line

```
process.hltEventAnalyzerAOD.triggerName = cms.string([trigger path])
```

in the configuration

# How is the trigger object production set up?

- The configuration file is `PhysicsTools/PatAlgos/python/triggerLayer0/patTrigProducer_cfi.py`
- Newly created configurations can be appended to the existing examples.
- Used module is the **`EDProducer`** **`PATTrigProducer`** with the following configurable parameters:
  - ***`InputTag`*** **`triggerEvent`**:
    source of trigger information (**`trigger::TriggerEvent`**)
  - ***`InputTag`*** **`filterName`**:
    actual collection label within the **`trigger::TriggerEvent`** as found before

# How is the trigger object production set up?

- Example configuration for trigger path #47:

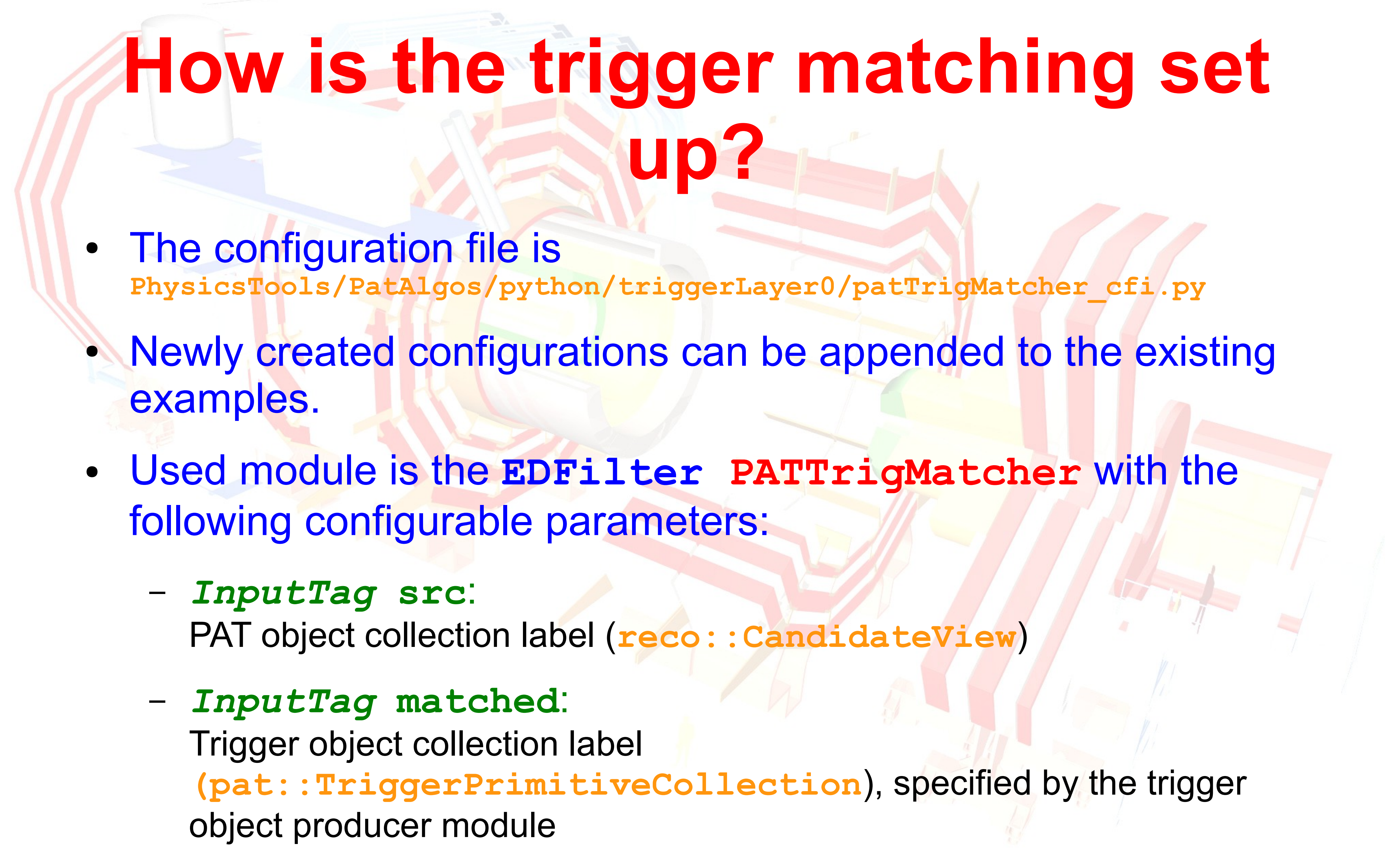
```
myTrigObjects = cms.EDProducer("PATTrigProducer",
    triggerEvent = cms.InputTag("hltTriggerSummaryAOD","","HLT"),
    filterName = cms.InputTag(
        "hltL1NonIsoHLTLooseIsoSingleElectronLWEt15TrackIsolFilter",
        "",
        "HLT")     # process name required!
)
```

# How are trigger objects stored in the PAT?

- The PAT uses the data format `pat::TriggerPrimitive` to store trigger objects:
  - Sorry for the confusing naming!
  - base class is `reco::Candidate`
  - additional data members:
    - *`std::string`* `filterName_`:
      holds name of filter module the object was used in
    - *`int`* `triggerObjectType_`:
      ID according to *`enum`* `trigger::TriggerObjectType`
  - differently used data member of reco::Candidate:
    - *`int`* `pdgId_`:
      trigger object ID slightly different fom PDG IDs

# How are trigger objects stored in the PAT?

- constructors:
  - take Lorentz vector (obligatory) and mentioned data member initializers (mandatory)
- methods:
  - only simple setters and getters for mentioned data members
  - inherited functionality of `reco::Candidate`
- s. class definition in `DataFormats/PatCandidates/interface/TriggerPrimitive.h` for details



# How is the trigger matching set up?

- The configuration file is `PhysicsTools/PatAlgos/python/triggerLayer0/patTrigMatcher_cfi.py`
- Newly created configurations can be appended to the existing examples.
- Used module is the **`EDFilter`** **`PATTrigMatcher`** with the following configurable parameters:
  - ***`InputTag`*** **`src`**:
    PAT object collection label (`reco::CandidateView`)
  - ***`InputTag`*** **`matched`**:
    Trigger object collection label `(pat::TriggerPrimitiveCollection`), specified by the trigger object producer module

# How is the trigger matching set up?

- `double maxDeltaR`:
  maximum distance in η-φ space to apply match
- `double maxDPtRel`:
  maximum difference in rel. pt to apply match
- `bool resolveAmbiguities`:
  only one PAT object is matched, if set to ***True***
- `bool resolveByMatchQuality`:
  if **`resolveAmbiguities=`*****True***, choose best match rather then first

- To guarantee the correct order of execution, a sequence to embrace producer and matcher is recommended.
  - producer configurations are imported to the matcher configuration file by default.

# How is the trigger matching set up?

- Example configuration for electrons and the example trigger:

```
from PhysicsTools.PatAlgos.triggerLayer0.patTrigProducer_cfi import *
[...]
myTrigMatches = cms.EDFilter("PATTrigMatcher",
    src      = cms.InputTag("allLayer0Electrons"),
    matched = cms.InputTag("myTrigObjects"),        # producer label
    maxDPtRel = cms.double(0.5),
    maxDeltaR = cms.double(0.5),
    resolveAmbiguities    = cms.bool(True),
    resolveByMatchQuality = cms.bool(False),
)
myTrigMatchSequence = cms.Sequence(
    myTrigObjects *
    myTrigMatches
)
```

# How is the trigger matching included into the PAT workflow?

- Sequences of trigger producers/matchers are defined in `PhysicsTool/PatAlgos/python/triggerLayer0/trigMatchSequences_cff.py`
- This is imported into `PhysicsTool/PatAlgos/python/patLayer0_cff.py`
- All producer/matcher sequences are scheduled there after the PAT layer 0 cleaners in order to provide the needed input collections, e.g.:

```
[...]
from PhysicsTools.PatAlgos.triggerLayer0.trigMatchSequences_cff import *
patLayer0 = cms.Sequence(
        patLayer0_withoutTrigMatch *
        patTrigMatch        * # default trigger prod./match sequence
        myTrigMatchSequence
)
```

# How are matched trigger objects added to PAT objects?

- Trigger matches are stored by "embedding".
- The addition of the trigger matches is configured in the particular PAT objects (leptons, jets MET) producers in `PhysicsTools/PatAlgos/python/producersLayer1/`
- The two configurable parameters in each producer module are:
  - *`bool`* `addTrigMatch`:
    general switch to add trigger object match
  - *`VInputTag`* `trigPrimMatch`:
    input product labels, specified by the trigger particle matching modules

# How are matched trigger objects added to PAT objects?

- Example configuration for the electron example:

```
addTrigMatch = cms.bool(True),
trigPrimMatch = cms.VInputTag
    cms.InputTag("electronTrigMatchHLT1ElectronRelaxed"),
    cms.InputTag("electronTrigMatchCandHLT1ElectronStartup"),
    cms.InputTag("myTrigMatches")
)
```

- It is recommended to maintain these configurables centrally as described in the SWGuidePATMatching.

# How are trigger matches added to the event content?

- Configurations are in
  PhysicsTools/PatAlgos/python/patLayer?_EventContent_cff.py
- PAT layer 0:
  - trigger objects:
    'keep patTriggerPrimitivesOwned_*_*_*'
  - trigger matches:
    'keep patTriggerPrimitivesOwnededmAssociation_*_*_*'
- PAT layer 1:
  - not needed due to "embedding"

# How are trigger matches used in analysis?

- The interface is provided by `pat::PATObject`, which has a data member `std::vector<TriggerPrimitive> triggerMatches_`:
  - `const std::vector<TriggerPrimitive> & triggerMatches() const`:
    C++ reference to the data member
  - `const std::vector<TriggerPrimitive> triggerMatchesByFilter(const std::string & aFilt) const`:
    newly created vector of trigger objects from one particular filter module (and so, most likely, one particular trigger path)
  - further methods (setters)

# Summary

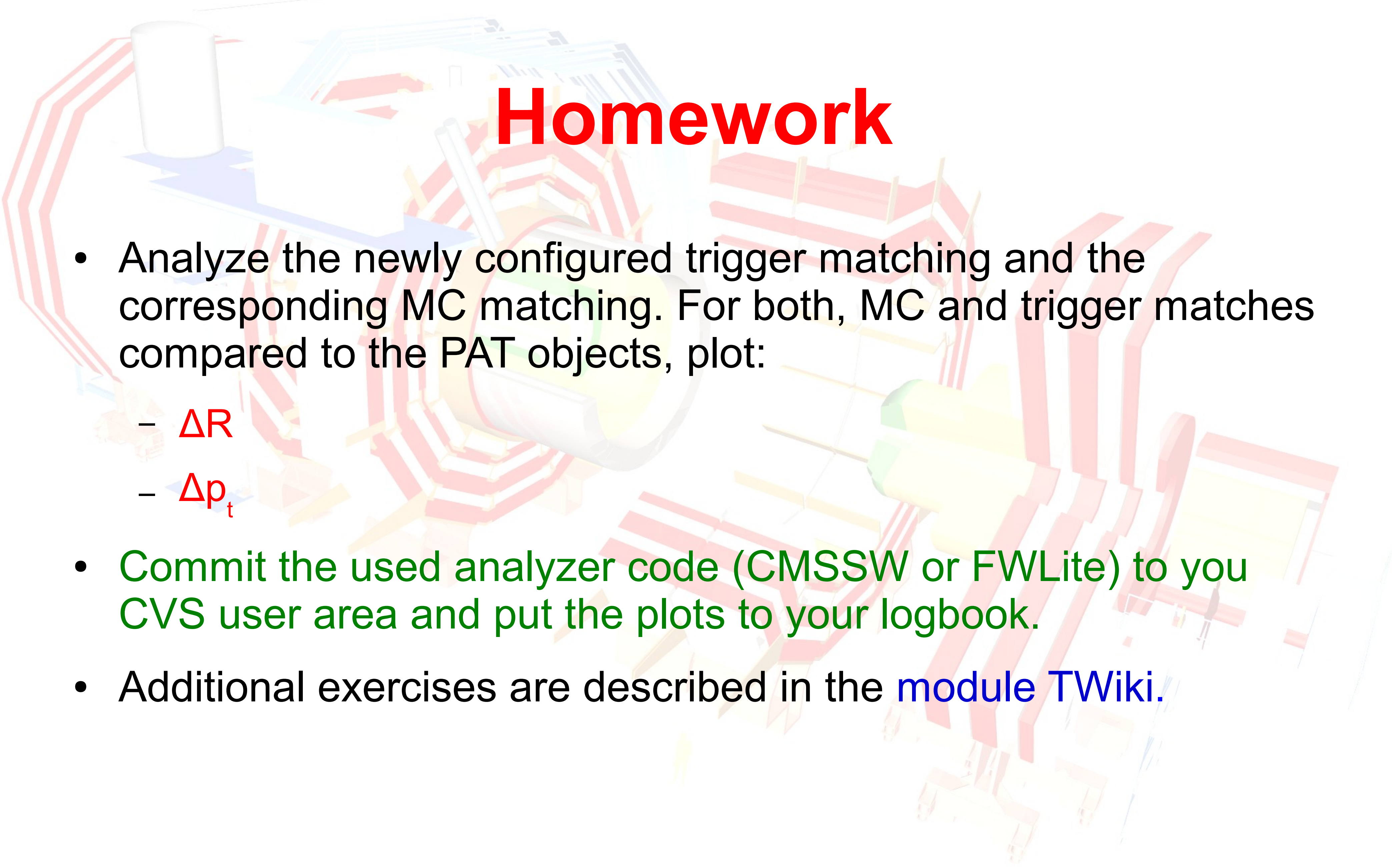
- This PAT eLearning module introduced the matching of MC and trigger objects to PAT objects.
- Both tasks allow to
  - evaluate certain factors of data reconstruction
    (e.g. correspondance between trigger and full reconstruction)
  - test object and event selections in an analysis
- Additional information can be found in the already mentioned SWGuidePATMatching.

# Outlook

- So far, the trigger matching is the only trigger information available directly in the PAT.
- New tools have been provided by trigger experts to access the complicated data structure of HLT information in AOD.
- Comprehensive PAT trigger information is currently being implemented  based on these tools:
  - data formats and producers in place for CMSSW_2_2_X
  - matching not yet transferred
  - not yet documented

# Hands-on exercise

- Reproduce the examples given in the SWGuidePATMatching.
- Configure and run a trigger matching for a trigger relevant for your *own* analysis.



# Homework

- Analyze the newly configured trigger matching and the corresponding MC matching. For both, MC and trigger matches compared to the PAT objects, plot:
  - ΔR
  - $\Delta p_t$
- Commit the used analyzer code (CMSSW or FWLite) to you CVS user area and put the plots to your logbook.
- Additional exercises are described in the module TWiki.