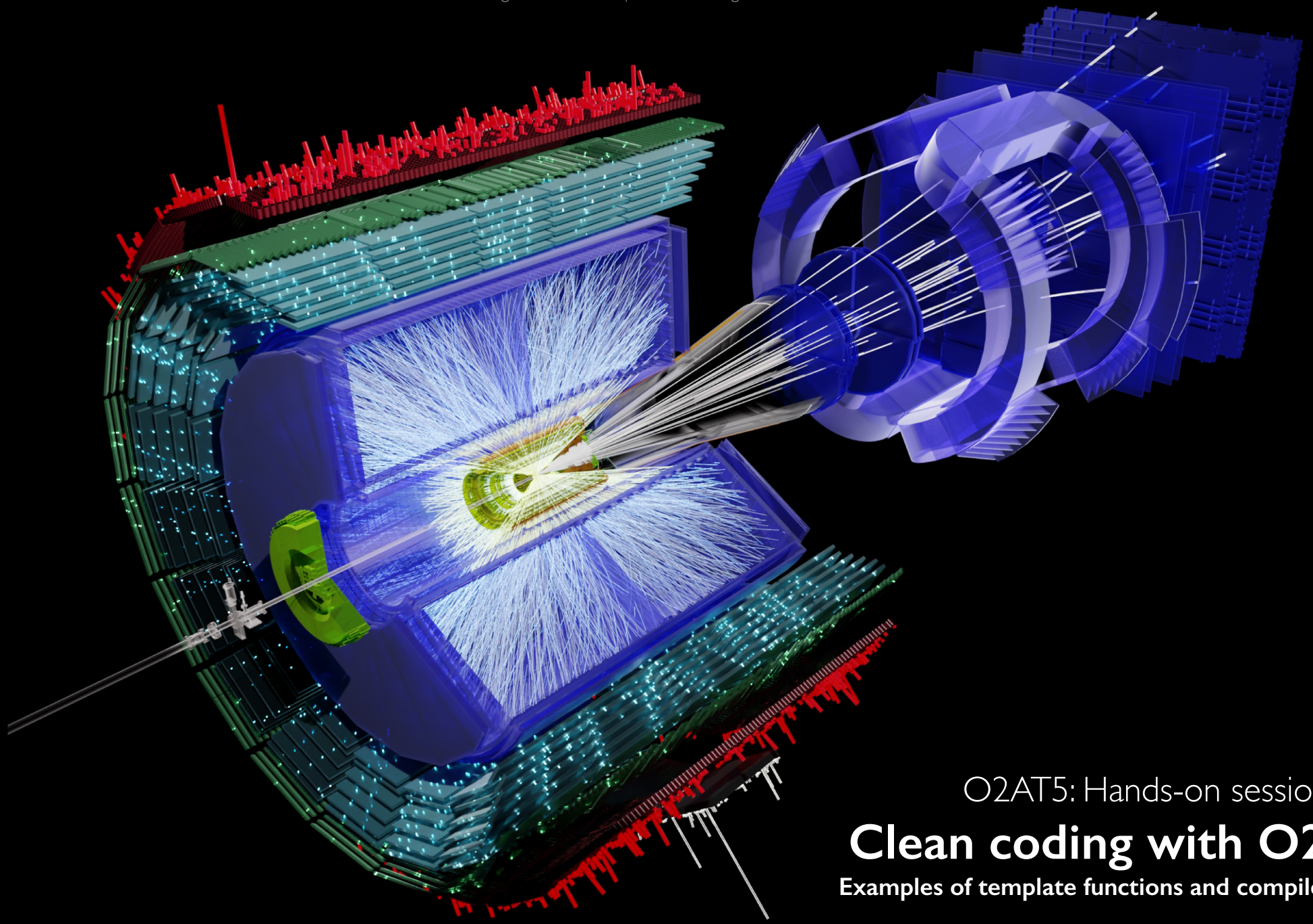
Image reference: https://zenodo.org/records/13732384

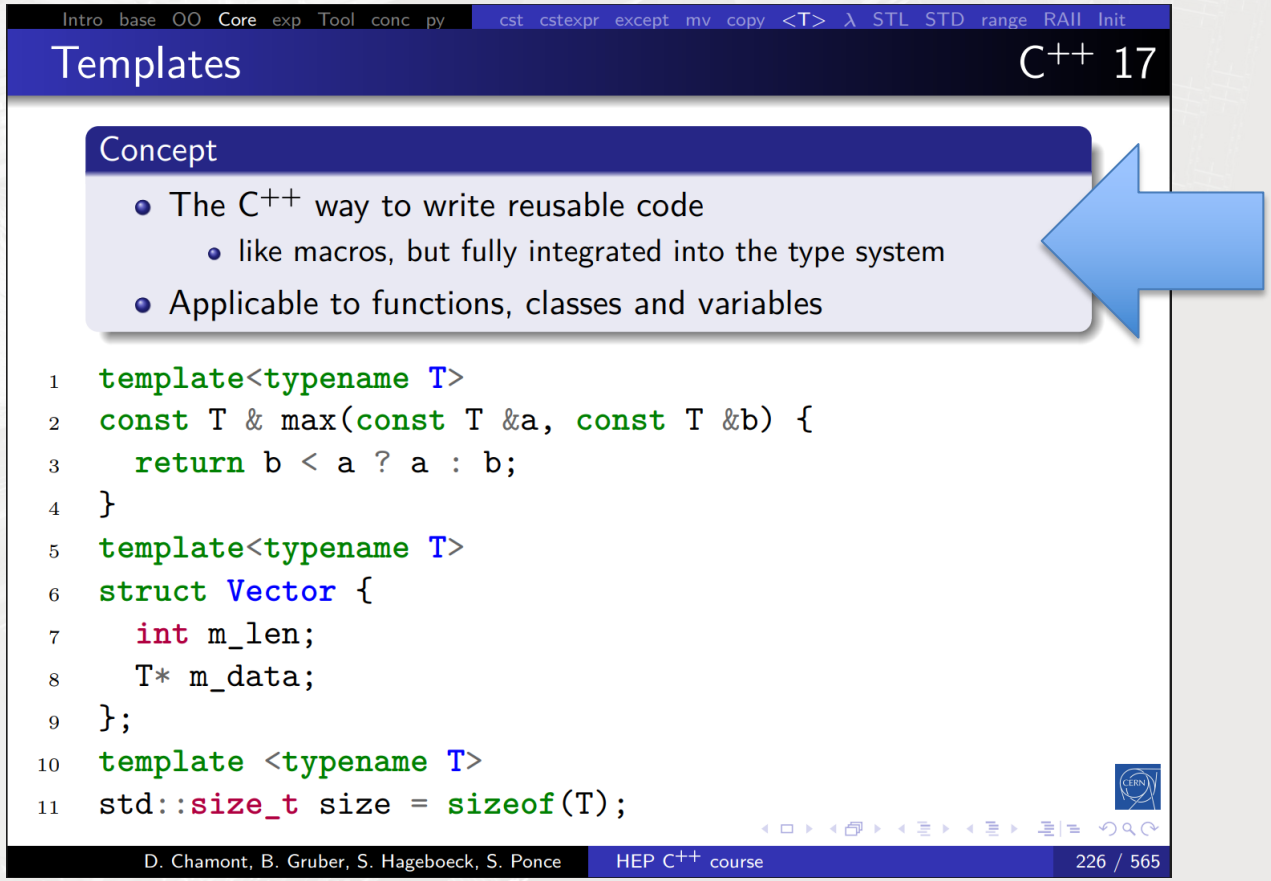


O2AT5: Hands-on session IV

Clean coding with O2Physics

Examples of template functions and compile time directives

Further resources on modern C++ coding:
13th HEP C++ Course and Hands-on Training - Advanced C++



- If you'd like to learn C++ from the basics to templates and beyond, please take a look!

Data for this exercise

- Let's use the same data you downloaded for the first and second hands-on exercises!
- These correspond to real data (dropbox, cernbox) and simulation (dropbox, cernbox) for Pb-Pb 2023

- We'll basically take the task you had at the end of the second hands-on and clear it up a little such that it is more flexible and does not involve any code documentation

The starting point: the task to calculate efficiencies

```
struct myExampleTask {
  // Histogram registry: an object to hold your histograms
  HistogramRegistry histos{"histos", {},
OutputObjHandlingPolicy::AnalysisObject};

  void processReco(aod::Collision const& collision, myFiltered const& tracks,
aod::McParticles const&)
  {
     // Huge block of code
  }
  PROCESS_SWITCH(myExampleTask, processReco, "process reconstructed
information", true);


  void processSim(aod::McCollision const& mcCollision,
soa::SmallGroups<soa::Join<aod::McCollisionLabels, aod::Collisions>> const&
collisions, aod::McParticles const& mcParticles, myFiltered const& tracks)
  {
     // Huge block of code
  }
  PROCESS_SWITCH(myExampleTask, processSim, "process pure simulation
information", true);
};
```

- Starting point: the previous code!
- The good old:

myExampleTask.cxx
+
Changes (link)

- A common problem: this task can actually only run on MC, because a subscription to McParticles is present!

- What if we would like to use the same task in real data?

The starting point: the task to calculate efficiencies

```
struct myExampleTask {
  // Boilerplate stuff goes here

  void processRecoInData(aod::Collision const& collision, [tracks])
  {
     // Huge block of code
  }
  PROCESS_SWITCH(myExampleTask, processRecoInData, "process reco in data", true);

  void processRecoInSim(aod::Collision const& collision, [tracks], [McParticles])

  {
     // Huge block of code
  }
  PROCESS_SWITCH(myExampleTask, processRecoInSim, "process reco in MC", false);


  void processSim(aod::McCollision const& mcCollision,
soa::SmallGroups<soa::Join<aod::McCollisionLabels, aod::Collisions>> const&
collisions, aod::McParticles const& mcParticles, myFiltered const& tracks)
  {
     // Huge block of code
  }
  PROCESS_SWITCH(myExampleTask, processSim, "process pure sim", false);
};
```

- Starting point: the previous code!
- The good old:

myExampleTask.cxx
+
Changes (link)

- A common problem: this task can actually only run on MC, because a subscription to McParticles is present!

- What if we would like to use the same task in real data? Yet another process function…

The starting point: the task to calculate efficiencies

```
struct myExampleTask {
  // Boilerplate stuff goes here

  void processRecoInData(aod::Collision const& collision, [tracks])
  {
     // Huge block of code
  }
  PROCESS_SWITCH(myExampleTask, processR                    , true);

  void processRecoInSim(aod::Collision                      ticles])

  {
     // Huge block of code
  }
  PROCESS_SWITCH(myExampleTask, processRecoInSim, "process reco in MC", false);


  void processSim(aod::McCollision const& mcCollision,
soa::SmallGroups<soa::Join<aod::McCollisionLabels, aod::Collisions>> const&
collisions, aod::McParticles const& mcParticles, myFiltered const& tracks)
  {
     // Huge block of code
  }
  PROCESS_SWITCH(myExampleTask, processSim, "process pure sim", false);
};
```

These huge blocks of code will have common parts! Dangerous and inconvenient

- Starting point: the previous code!
- The good old:

myExampleTask.cxx
+
Changes (link)

- A common problem: this task can actually only run on MC, because a subscription to McParticles is present!

- What if we would like to use the same task in real data? Yet another process function…

This can be handled conveniently in modern C++ with template functions a series of tricks!

What do we want to achieve?

```
struct myExampleTask {
  // Boilerplate stuff goes here

  void processStuff(arguments){
      // do data stuff here

     // do MC stuff here but only if it's MC
  }

  void processRecoInData(aod::Collision const& collision, [tracks])
  {
       processStuff(collision, tracks); // simple data tracks
  }
  PROCESS_SWITCH(myExampleTask, processRecoInData, "process reco in data", true);

  void processRecoInSim(aod::Collision const& collision, [tracks], [McParticles])
  {
       processStuff(collision, tracks); // these tracks will be labeled!
  }
  PROCESS_SWITCH(myExampleTask, processRecoInSim, "process reco in MC", false);
};
```

- Could we make the processing such that 'process' functions define the interface to O2Physics but serve only as an entry point to a 'processStuff' function that adapts depending on complex compile-time conditions?
- For instance, in the second process function the tracks will contain a "mcParticleId", but not in the first. How can we deal with this and detect, inside processStuff, if the MC part is to be done or not?

```
using myCompleteTracksMC = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA, aod::McTrackLabels>;
using myCompleteTracks = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA>;
using myFilteredTracksMC = soa::Filtered<myCompleteTracksMC>;
using myFilteredTracks = soa::Filtered<myCompleteTracks>;

template <typename TCollision, typename TTracks>
void processStuff(TCollision const& collision, TTracks const& tracks) {
  histos.fill(HIST("eventCounter"), 0.5f);
  for (const auto& track : tracks) {
    if( track.tpcNClsCrossedRows() < minTPCCrossedRows ) continue; //badly tracked
    histos.fill(HIST("etaHistogram"), track.eta());
    histos.fill(HIST("ptHistogram"), track.pt());

    // this part only done if dealing with MC
    if constexpr (requires { track.has_mcParticle(); }) { // does the getter exist?
      if(track.has_mcParticle()){ // is the return 'true'? -> N.B. different question!
        auto mcParticle = track.mcParticle();
        histos.fill(HIST("ptResolution"), track.pt(), track.pt() - mcParticle.pt());
        if(mcParticle.isPhysicalPrimary() && fabs(mcParticle.y())<0.5){
          if(abs(mcParticle.pdgCode())==kPiPlus) histos.fill(HIST("ptHistogramPion"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kKPlus) histos.fill(HIST("ptHistogramKaon"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kProton) histos.fill(HIST("ptHistogramProton"), mcParticle.pt());
        }
      }
    }
  }
}

void processRecoInData(aod::Collision const& collision, myFilteredTracks const& tracks)
{
  processStuff(collision, tracks);
}
PROCESS_SWITCH(myExampleTask, processRecoInData, "process reco in real data", true);

void processRecoInSim(aod::Collision const& collision, myFilteredTracksMC const& tracks, aod::McParticles
const&)
{
  processStuff(collision, tracks);
}
PROCESS_SWITCH(myExampleTask, processRecoInSim, "process reco in mc data", false);
```

data+mc

MC only

Yes, we can: using template functions and compile time directives

There are plenty of elements in this solution: let's break it down…

ALICE

```
using myCompleteTracksMC = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA, aod::McTrackLabels>;
using myCompleteTracks = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA>;
using myFilteredTracksMC = soa::Filtered<myCompleteTracksMC>;
using myFilteredTracks = soa::Filtered<myCompleteTracks>;

template <typename TCollision, typename TTracks>
void processStuff(TCollision const& collision, TTracks const& tracks) {
  histos.fill(HIST("eventCounter"), 0.5f);
  for (const auto& track : tracks) {
    if( track.tpcNClsCrossedRows() < minTPCCrossedRows ) continue; //badly tracked
    histos.fill(HIST("etaHistogram"), track.eta());
    histos.fill(HIST("ptHistogram"), track.pt());

    // this part only done if dealing with MC
    if constexpr (requires { track.has_mcParticle(); }) { // does the getter exist?
      if(track.has_mcParticle()){ // is the return 'true'? -> N.B. different question!
        auto mcParticle = track.mcParticle();
        histos.fill(HIST("ptResolution"), track.pt(), track.pt() - mcParticle.pt());
        if(mcParticle.isPhysicalPrimary() && fabs(mcParticle.y())<0.5){
          if(abs(mcParticle.pdgCode())==kPiPlus) histos.fill(HIST("ptHistogramPion"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kKPlus) histos.fill(HIST("ptHistogramKaon"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kProton) histos.fill(HIST("ptHistogramProton"), mcParticle.pt());
        }
      }
    }
  }
}

void processRecoInData(aod::Collision const& collision, myFilteredTracks const& tracks)
{
  processStuff(collision, tracks);
}
PROCESS_SWITCH(myExampleTask, processRecoInData, "process reco in real data", true);

void processRecoInSim(aod::Collision const& collision, myFilteredTracksMC const& tracks, aod::McParticles
const&)
{
  processStuff(collision, tracks);
}
PROCESS_SWITCH(myExampleTask, processRecoInSim, "process reco in mc data", false);
```

Yes, we can: using template functions and compile time directives

Function with template arguments: in each context in which the function is used, the compiler checks arguments and specializes the processStuff function

ALICE

```
using myCompleteTracksMC = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA, aod::McTrackLabels>;
using myCompleteTracks = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA>;
using myFilteredTracksMC = soa::Filtered<myCompleteTracksMC>;
using myFilteredTracks = soa::Filtered<myCompleteTracks>;

template <typename TCollision, typename TTracks>
void processStuff(TCollision const& collision, TTracks const& tracks) {
  histos.fill(HIST("eventCounter"), 0.5f);
  for (const auto& track : tracks) {
    if( track.tpcNClsCrossedRows() < minTPCCrossedRows ) continue; //badly tracked
    histos.fill(HIST("etaHistogram"), track.eta());
    histos.fill(HIST("ptHistogram"), track.pt());

    // this part only done if dealing with MC
    if constexpr (requires { track.has_mcParticle(); }) { // does the getter exist?
      if(track.has_mcParticle()){ // is the return 'true'? -> N.B. different question!
        auto mcParticle = track.mcParticle();
        histos.fill(HIST("ptResolution"), track.pt(), track.pt() - mcParticle.pt());
        if(mcParticle.isPhysicalPrimary() && fabs(mcParticle.y())<0.5){
          if(abs(mcParticle.pdgCode())==kPiPlus) histos.fill(HIST("ptHistogramPion"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kKPlus) histos.fill(HIST("ptHistogramKaon"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kProton) histos.fill(HIST("ptHistogramProton"), mcParticle.pt());
        }
      }
    }
  }
}

void processRecoInData(aod::Collision const& collision, myFilteredTracks const& tracks)
{
  processStuff(collision, tracks);
}
PROCESS_SWITCH(myExampleTask, processRecoInData, "process reco in real data", true);

void processRecoInSim(aod::Collision const& collision, myFilteredTracksMC const& tracks, aod
const&)
{
  processStuff(collision, tracks);
}
PROCESS_SWITCH(myExampleTask, processRecoInSim, "process reco in mc data", false);
```

Yes, we can: using template functions and compile time directives

Function with template arguments: in each context in which the function is used, the compiler checks arguments and specializes the processStuff function

The two calls are different! In one case, I have access to mcParticle label getters, but in the other one I do not!

ALICE

```
using myCompleteTracksMC = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA, aod::McTrackLabels>;
using myCompleteTracks = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA>;
using myFilteredTracksMC = soa::Filtered<myCompleteTracksMC>;
using myFilteredTracks = soa::Filtered<myCompleteTracks>;

template <typename TCollision, typename TTracks>
void processStuff(TCollision const& collision, TTracks const& tracks) {
  histos.fill(HIST("eventCounter"), 0.5f);
  for (const auto& track : tracks) {
    if( track.tpcNClsCrossedRows() < minTPCCrossedRows ) continue; //badly tracked
    histos.fill(HIST("etaHistogram"), track.eta());
    histos.fill(HIST("ptHistogram"), track.pt());

    // this part only done if dealing with MC
    if constexpr (requires { track.has_mcParticle(); }) { // does the getter exist?
      if(track.has_mcParticle()){ // is the return 'true'? -> N.B. different question!
        auto mcParticle = track.mcParticle();
        histos.fill(HIST("ptResolution"), track.pt(), track.pt() - mcParticle.pt());
        if(mcParticle.isPhysicalPrimary() && fabs(mcParticle.y())<0.5){
          if(abs(mcParticle.pdgCode())==kPiPlus) histos.fill(HIST("ptHistogramPion"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kKPlus) histos.fill(HIST("ptHistogramKaon"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kProton) histos.fill(HIST("ptHistogramProton"), mcParticle.pt());
        }
      }
    }
  }
}

void processRecoInData(aod::Collision const& collision, myFilteredTracks const& tracks)
{
  processStuff(collision, tracks);
}
PROCESS_SWITCH(myExampleTask, processRecoInData, "process reco in real data", true);

void processRecoInSim(aod::Collision const& collision, myFilteredTracksMC const& tracks, aod
const&)
{
  processStuff(collision, tracks);
}
PROCESS_SWITCH(myExampleTask, processRecoInSim, "process reco in mc data", false);
```

Yes, we can: using template functions and compile time directives

Function with template arguments: in each context in which the function is used, the compiler checks arguments and specializes the processStuff function

The two calls are different! In one case, I have access to mcParticle label getters, but in the other one I do not!

The processStuff function checks if I have the getter functionality at compile time

```
using myCompleteTracksMC = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA, aod::McTrackLabels>;
using myCompleteTracks = soa::Join<aod::Tracks, aod::TracksExtra, aod::TracksDCA>;
using myFilteredTracksMC = soa::Filtered<myCompleteTracksMC>;
using myFilteredTracks = soa::Filtered<myCompleteTracks>;

template <typename TCollision, typename TTracks>
void processStuff(TCollision const& collision, TTracks const& tracks) {
  histos.fill(HIST("eventCounter"), 0.5f);
  for (const auto& track : tracks) {
    if( track.tpcNClsCrossedRows() < minTPCCrossedRows ) continue; //badly tracked
    histos.fill(HIST("etaHistogram"), track.eta());
    histos.fill(HIST("ptHistogram"), track.pt());

    // this part only done if dealing with MC
    if constexpr (requires { track.has_mcParticle(); }) { // does the getter exist?
      if(track.has_mcParticle()){ // is the return 'true'? -> N.B. different question!
        auto mcParticle = track.mcParticle();
        histos.fill(HIST("ptResolution"), track.pt(), track.pt() - mcParticle.pt());
        if(mcParticle.isPhysicalPrimary() && fabs(mcParticle.y())<0.5){
          if(abs(mcParticle.pdgCode())==kPiPlus) histos.fill(HIST("ptHistogramPion"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kKPlus) histos.fill(HIST("ptHistogramKaon"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kProton) histos.fill(HIST("ptHistogramProton"), mcParticle.pt());
        }
      }
    }
  }
}
```

Yes, we can: using template functions and compile time directives

Asking if the getter exists ≠ getter returns true

Final effects:

- The compiled executable will technically contain two versions of 'processStuff', but **you will need to edit only one block of code!**
- 'if constexpr' is checked at compile time: executes faster than runtime checks because the decision is already done
- Multiple other tricks exist for compile-time decisionmaking

Function with template arguments: in each context in which the function is used, the compiler checks arguments and specializes the processStuff function

The two calls are different! In one case, I have access to mcParticle label getters, but in the other one I do not!

The processStuff function checks if I have the getter functionality at compile time

Compile-time arguments: example

```
template <bool fillResolution = true, typename TCollision, typename TTracks>
void processStuff(TCollision const& collision, TTracks const& tracks) {
  histos.fill(HIST("eventCounter"), 0.5f);
  for (const auto& track : tracks) {
    if( track.tpcNClsCrossedRows() < minTPCCrossedRows ) continue; //badly tracked
    histos.fill(HIST("etaHistogram"), track.eta());
    histos.fill(HIST("ptHistogram"), track.pt());


    // this part only done if dealing with MC
    if constexpr (requires { track.has_mcParticle(); }) { // does the getter exist?
      if(track.has_mcParticle()){ // is the return 'true'? -> N.B. different question!
        auto mcParticle = track.mcParticle();
        if constexpr (fillResolution){ // compile-time check
          histos.fill(HIST("ptResolution"), track.pt(), track.pt() - mcParticle.pt());
        }
        if(mcParticle.isPhysicalPrimary() && fabs(mcParticle.y())<0.5){
          if(abs(mcParticle.pdgCode())==kPiPlus) histos.fill(HIST("ptHistogramPion"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kKPlus) histos.fill(HIST("ptHistogramKaon"), mcParticle.pt());
          if(abs(mcParticle.pdgCode())==kProton) histos.fill(HIST("ptHistogramProton"), mcParticle.pt());
        }
      }
    }
  }
}
```

- Another option is to define arguments such as a boolean for doing something specific
- In this case, one can now invoke processStuff<true>(collision, tracks)
- This might run faster (conditional checked at compile time and NOT at runtime)
- Drawback: the compiler with create two copies of the compiled function

The solution to this exercise can be found here

Backup

Surgical compilation: ninja to the rescue

- **If you have a compiled O2Physics installation** and just want to recompile one subdirectory or even one particular executable, this can be done using a tool called “ninja” – it will help you be faster!

- Go to your build directory using:

Bear in mind: different path needed for any non-standard installation path

```
cd ~/alice/sw/BUILD/O2Physics-latest/O2Physics
```

- Once in that directory, load the build environment doing: (important: you should not have called alienv before!)

```
direnv allow
```

- You can then rebuild only one specific subdirectory of O2Physics:

```
ninja <directory>/install
```

```
# Example: ninja PWGCF/Tasks/install
```

- You can even rebuild only one single executable:

```
ninja stage/bin/<target>
```

```
# Example: ninja stage/bin/o2-analysis-cf-correlations
```