Event selection for UPC in run 3

Sigurd Nese

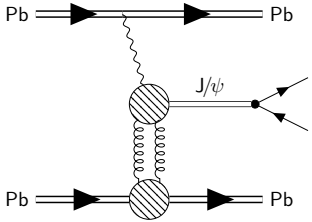
University of Oslo

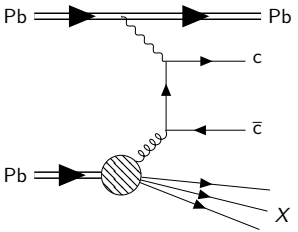
O2 Analysis Tutorial 5.0

November 13, 2025

In ultra-peripheral heavy-ion collisions we study photon-induced processes

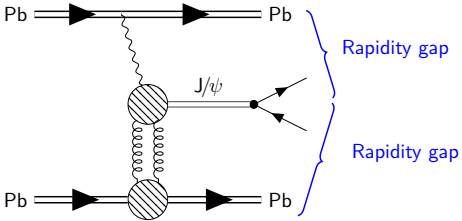
Examples of processes we want to study:

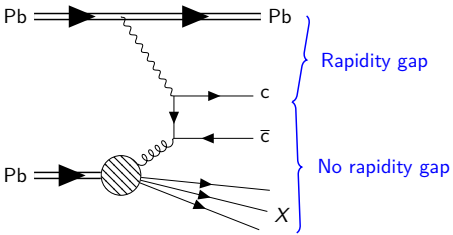




- Exclusive production of vector mesons
- Inelastic photonuclear processes

UPC events leave large gaps void of particle production (rapidity gaps) in the forward and backward directions

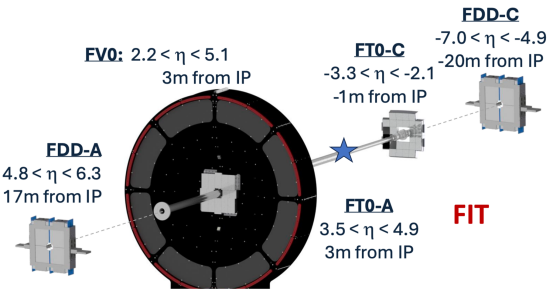




- Exclusive studies → Rapidity gaps on both sides (“Double Gap”)
- Inclusive studies → Rapidity gaps on one side (“Single Gap”)

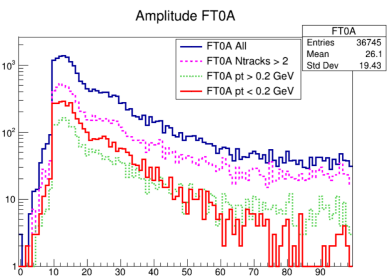
In run 3, we use the FIT detectors to find rapidity gaps

- FIT = Fast Interaction Trigger
  - Time resolution $\mathcal{O}(10 - 100$ ps)
  - Bunch spacing $\mathcal{O}(10$ ns)
- A-side: FT0A, FDDA, FV0A.
- C-side: FT0C, FDDC.
- Recent presentation about FIT in the ALICE Software and Computing Days: Slides by Andreas Molander

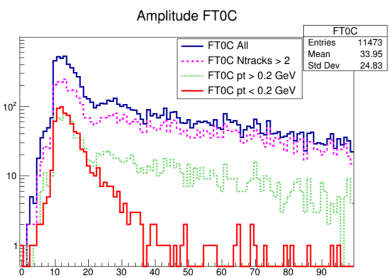


Using coherent ρ^0 production as a benchmark, we can look at the UPC separation power of the FIT detectors

(No activity on C side)



(No activity on A side)

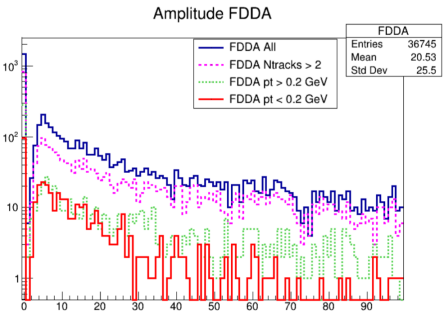


- Events with only 2 low p_T tracks dominated by coherent $\rho^0 \rightarrow$ FIT activity is due to electromagnetic pileup
- Require FT0A < 100, FT0C $< 50 \rightarrow$ reject true activity, keep coherent events

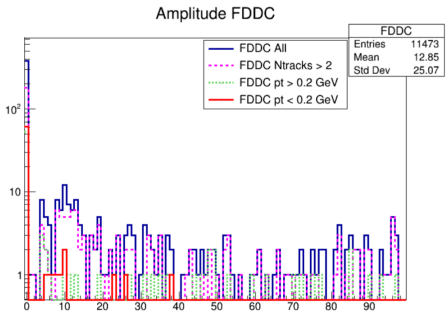
Plots from analysis note by Sasha Bylinkin and Simone Ragoni

Using coherent ρ^0 production as a benchmark, we can look at the UPC separation power of the FIT detectors

(No activity on C side)



(No activity on A side)

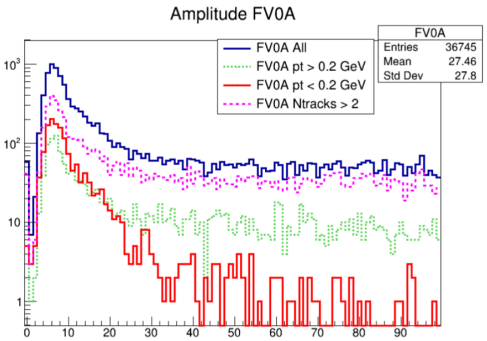


- FDD does not have as good discrimination power for UPCs

Plots from analysis note by Sasha Bylinkin and Simone Ragoni

Using coherent ρ^0 production as a benchmark, we can look at the UPC separation power of the FIT detectors

(No activity on C side)

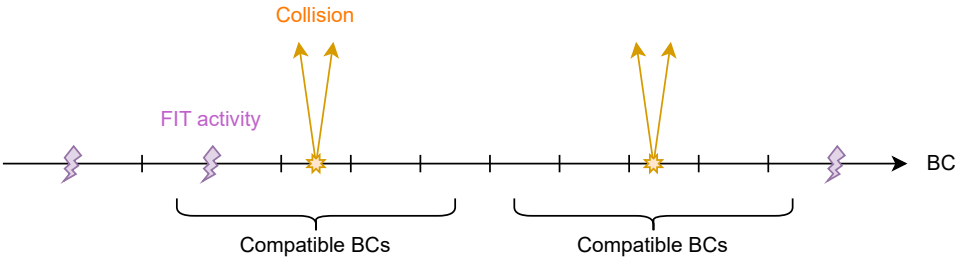


- FV0 also works well for UPC selection

Plots from analysis note by Sasha Bylinkin and Simone Ragoni

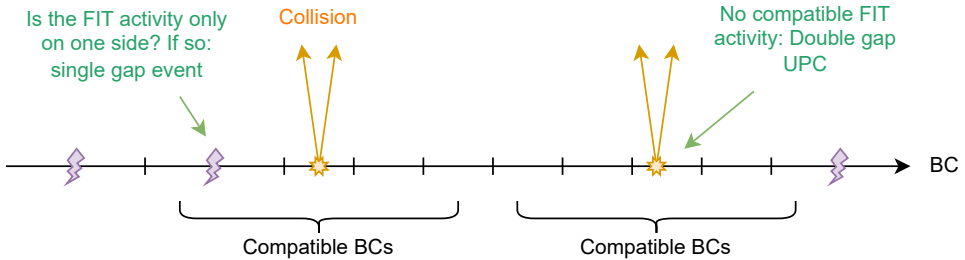
We look at a range of bunch crossings to determine whether an event was a UPC

- A collision is defined by tracks forming a primary vertex
  - Collision time resolution depends on track time precision
  - TPC+ITS tracks $\rightarrow$ 200 ns precision $\rightarrow$ several BCs



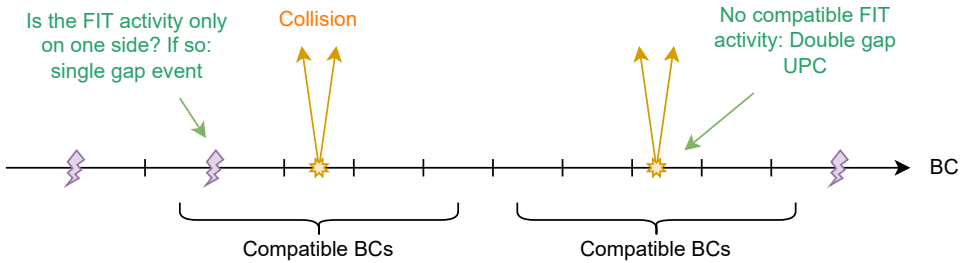
We look at a range of bunch crossings to determine whether an event was a UPC

- A collision is defined by tracks forming a primary vertex
  - Collision time resolution depends on track time precision
  - TPC+ITS tracks $\rightarrow$ 200 ns precision $\rightarrow$ several BCs



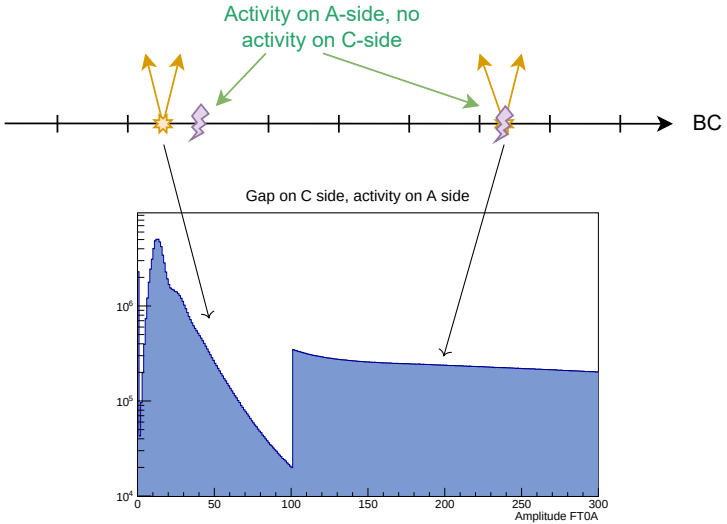
We look at a range of bunch crossings to determine whether an event was a UPC

- A collision is defined by tracks forming a primary vertex
  - Collision time resolution depends on track time precision
  - TPC+ITS tracks $\rightarrow$ 200 ns precision $\rightarrow$ several BCs

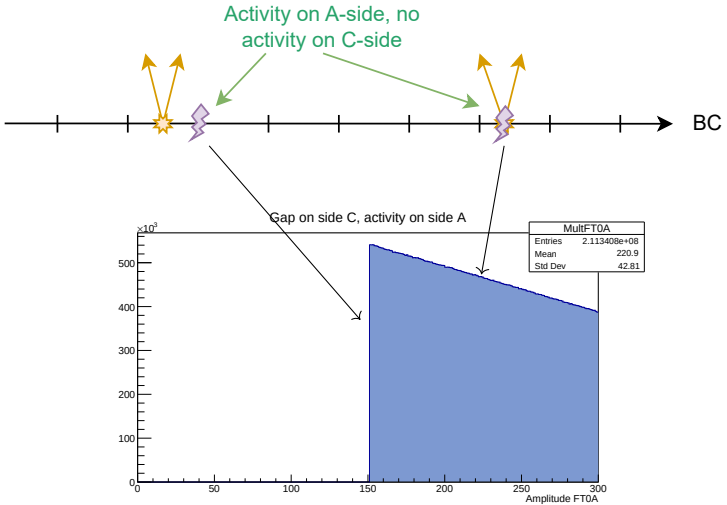


- Selection can be done on BCs directly $\rightarrow$ useful for analyses of forward tracks (no collision vertex)
  - Check out `PWGUD/TableProducer/UPCCandidateProducer`

We must take care to save the FIT info from the relevant BC for further studies



We must take care to save the FIT info from the relevant BC for further studies



Rapidity gap event selection algorithm

We need an algorithm which

- processes collisions in the raw heavy-ion data,
- finds the range of compatible BCs based on the collision time resolution,
- looks through that range of BCs and checks whether...
  - the FIT A side activity is below threshold,
  - the FIT C side activity is below threshold,
- ...and classifies the event as no gap, single gap A, single gap C, or double gap,
- saves the BC which had activity above the threshold (or the most active BC if event is double gap)

Such an algorithm is implemented in `PWGUD/Core/SGSelector.h`, as the class `SGSelector`

SGSelector in practice: Tutorials/PWGUD/UDTutorial_08.cxx

- Set up dependencies and the tables we want to process

```
#include <Framework/AnalysisDataModel.h>
#include <Framework/AnalysisTask.h>
#include <Framework/runDataProcessing.h>

#include "PWGUD/Core/SGSelector.h"

using namespace o2::framework;

// EvSels table contains connection between collision and BC
using MyEvents = o2::soa::Join<o2::aod::Collisions, o2::aod::EvSels>;
// BcSels table contains connection between BC and FIT info.
// Run3MatchedToBCSparse table contains index into ZDC table.
using MyBCs = o2::soa::Join<o2::aod::BCs, o2::aod::BcSels, o2::aod::Run3MatchedToBCSparse>;
```

SGSelector in practice: Tutorials/PWGUD/UDTutorial_08.cxx

- Declare task, histograms and instances of helper classes

```
struct UDTutorial08 {

  // Histogram setup
  OutputObj<TH1I> hSelectionResult{TH1I("hSelectionResult", "hSelectionResult", 5, -0.5, 4.5)};
  OutputObj<TH1I> hSelectionResultAfterCut{TH1I("hSelectionResultAfterCut",
                                                "hSelectionResultAfterCut", 5, -0.5, 4.5)};
  OutputObj<TH1F> hZNAEnergy{TH1F("hZNAEnergy", "hZNAEnergy", 200, 0, 20)};
  OutputObj<TH1F> hZNAEnergyAfterCut{TH1F("hZNAEnergyAfterCut", "hZNAEnergyAfterCut", 200, 0, 20)};
  OutputObj<TH1F> hZNCEnergy{TH1F("hZNCEnergy", "hZNCEnergy", 200, 0, 20)};
  OutputObj<TH1F> hZNCEnergyAfterCut{TH1F("hZNCEnergyAfterCut", "hZNCEnergyAfterCut", 200, 0, 20)};
  // Create instance of the selector class which runs the gap selection algorithm
  SGSelector sgSelector;
  // Create instance of cut holder class to contain the user defined cuts
  SGCutParHolder sgCuts = SGCutParHolder();
```

SGSelector in practice: Tutorials/PWGUD/UDTutorial_08.cxx

- In init function, set the event selection parameter values

```
void init(o2::framework::InitContext&)
{
  // Configure the gap selection criteria. Rest of the values are kept default
  sgCuts.SetNDtcoll(1);          // Time range to consider, in units of collision time resolution
  sgCuts.SetMinNBCs(2);          // Minimum number of BCs to check
  sgCuts.SetNTracks(2, 100);     // Reject collisions with < 2 tracks and > 100 tracks
  sgCuts.SetMaxFITtime(34);      // Reject collisions with FIT time > 34 ns in a compatible BC
  sgCuts.SetFITAmpLimits({-1,    // Don't use the FV0A for selection
                          150,   // Require FT0A amplitude to be below 150 in all
                                 // compatible BCs to classify as gap
                          50,    // Require FT0C amplitude to be below 50 in all
                                 // compatible BCs to classify as gap
                          -1,    // Don't use the FDDA for selection
                          -1}); // Don't use the FDDC for selection
}
```

SGSelector in practice: Tutorials/PWGUD/UDTutorial_08.cxx

- Define the process function, which processes collision by collision

```
void process(MyEvents::iterator const& collision, // Process collision by collision
             MyBCs const& bcs,                    // We will check a range of bunch crossings
             o2::aod::FT0s const&,                // Must subscribe to the FIT tables for
             o2::aod::FDDs const&,                // the SGSelector to access them
             o2::aod::FV0As const&,
             o2::aod::Zdcs const&) // Want to plot ZDC energies, so we need to
                                   // subscribe to the ZDC table
{
  // Find the bunch crossing assigned to this collision
  auto bc = collision.foundBC_as<MyBCs>();
  // Find the range of bunch crossings compatible with this collision
  auto bcRange = udhelpers::compatibleBCs(collision, sgCuts.NDtcoll(), bcs, sgCuts.minNBCs());
  // Determine whether this event is single gap (A or C), double gap, or no gap
  auto selectorResult = sgSelector.IsSelected(sgCuts, collision, bcRange, bc);
  auto newbc = *(selectorResult.bc);

  // --- Process the event here: Apply cuts, save to derived tables, fill histograms... ---
```

SGSelector in practice: Tutorials/PWGUD/UDTutorial_08.cxx

- Define the process function, which processes collision by collision

```
void process(MyF                                                    sion
             MyF                                                    unch crossings
             o2.                                                    tables for
             o2.                                                    hem
             o2.
             o2.

{
  // Find the bu
  auto bc = collision.foundBC_as<MyBCs>();
  // Find the range of bunch crossings compatible with this collision
  auto bcRange = udhelpers::compatibleBCs(collision, sgCuts.NDtcoll(), bcs, sgCuts.minNBCs());
  // Determine whether this event is single gap (A or C), double gap, or no gap
  auto selectorResult = sgSelector.IsSelected(sgCuts, collision, bcRange, bc);
  auto newbc = *(selectorResult.bc);

  // --- Process the event here: Apply cuts, save to derived tables, fill histograms... ---
```

Sidenote:

The SGSelector::IsSelected method returns a struct with two members:

```
template <typename BC>
struct SelectionResult {
  int value;    // The original integer return value
  const BC* bc; // Pointer to the BC object
};
```

SGSelector in practice: Tutorials/PWGUD/UDTutorial_08.cxx

- Fill the histograms we want *before* making any event cuts

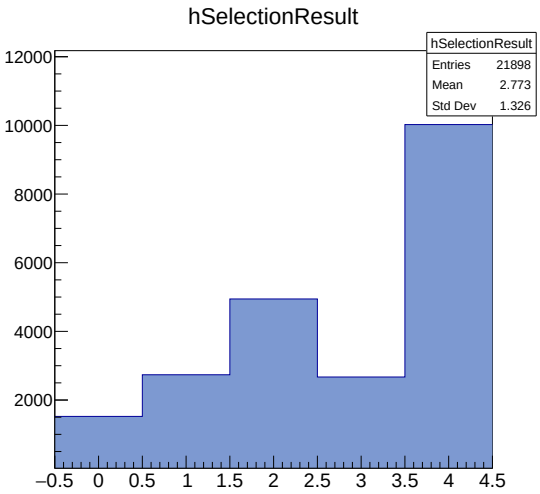
```
// --- Process the event here: Apply cuts, save to derived tables, fill histograms... ---

// Plot the outcome of the gap selection algorithm
hSelectionResult->Fill(selectorResult.value);
// Plot ZDC energies
if (newbc.has_zdc()) {
  auto zdc = newbc.zdc();
  hZNAEnergy->Fill(zdc.energyCommonZNA());
  hZNCEnergy->Fill(zdc.energyCommonZNC());
} else {
  hZNAEnergy->Fill(-999);
  hZNCEnergy->Fill(-999);
}
```

SGSelector in practice: Tutorials/PWGUD/UDTutorial_08.cxx

- Result of running on 3.8 GB AO2D file from LHC23_PbPb_pass5 [1]
- `SelectionResult.value` is an int between 0 and 4, corresponding to the enum `o2::aod::sgselector::TrueGap`

```
namespace o2::aod::sgselector
{
enum TrueGap : int {
  NoGap = -1,
  SingleGapA = 0,
  SingleGapC = 1,
  DoubleGap = 2,
  NoUpc = 3,
  TrkOutOfRange = 4,
  BadDoubleGap = 5
};
} // namespace o2::aod::sgselector
```



[1] Go to e. g. /alice/data/2023/LHC23zzh/544123/apass5/ and navigate into one of the subdirectories to find a similar data file

SGSelector in practice: Tutorials/PWGUD/UDTutorial_08.cxx

- Apply an event selection, and fill the histograms we want for the events which passed

```
// Apply a selection -- as an example, keep only events classified as single gap on C side
if (selectorResult.value != o2::aod::sgselector::SingleGapC) {
  return;
}
hSelectionResultAfterCut->Fill(selectorResult.value);
if (newbc.has_zdc()) {
  auto zdc = newbc.zdc();
  hZNAEnergyAfterCut->Fill(zdc.energyCommonZNA());
  hZNCEnergyAfterCut->Fill(zdc.energyCommonZNC());
} else {
  hZNAEnergyAfterCut->Fill(-999);
  hZNCEnergyAfterCut->Fill(-999);
}
```

SGSelector in practice: Tutorials/PWGUD/UDTutorial_08.cxx

- ...and finish the code by defining the workflow

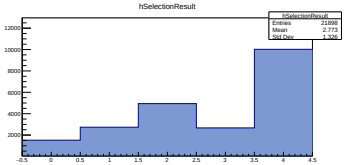
```
  ...
  }
};

WorkflowSpec defineDataProcessing(ConfigContext const& cfgc)
{
  return WorkflowSpec{
    adaptAnalysisTask<UDTutorial08>(cfgc, TaskName{"udtutorial08"})};
}
```

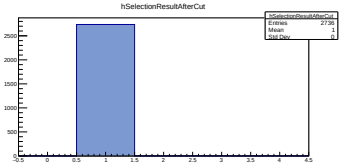
- Run this task by executing:

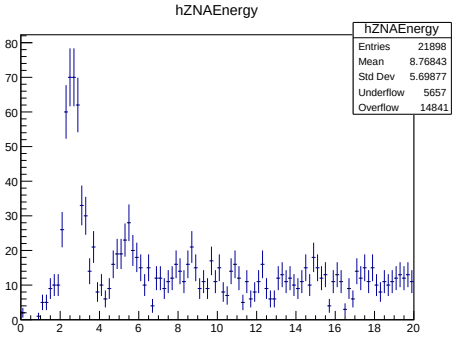
```
o2-analysis-ud-udtutorial-08 -b | o2-analysis-event-selection-service -b
--aod-file /path/to/your/AO2D.root
```

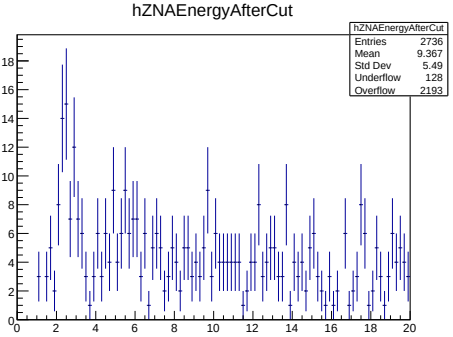
As a result of the event selection, the ZDC energies are biased as expected



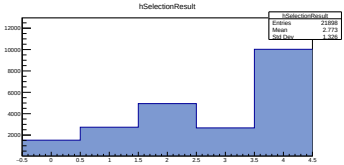
Event selection



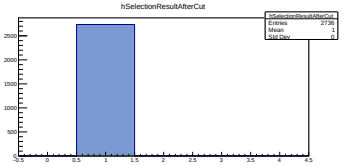


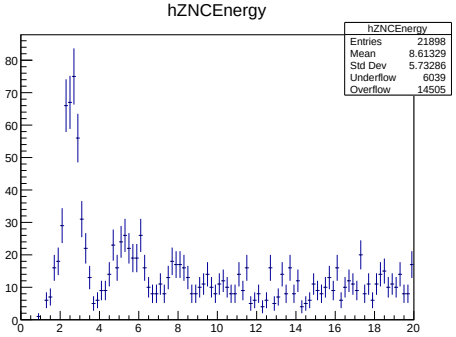


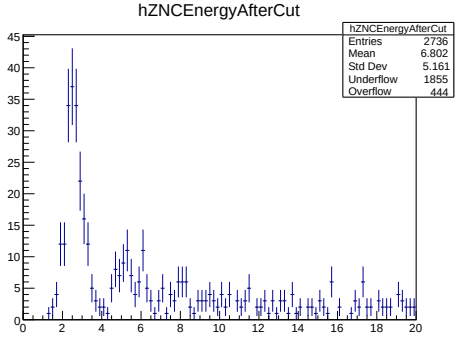
As a result of the event selection, the ZDC energies are biased as expected



Event selection

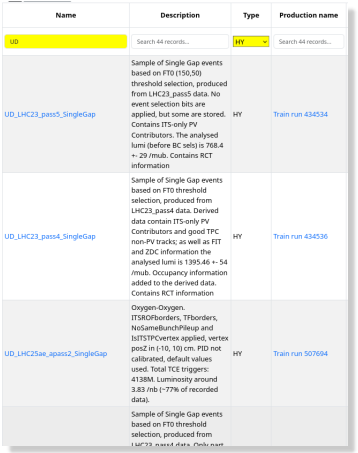






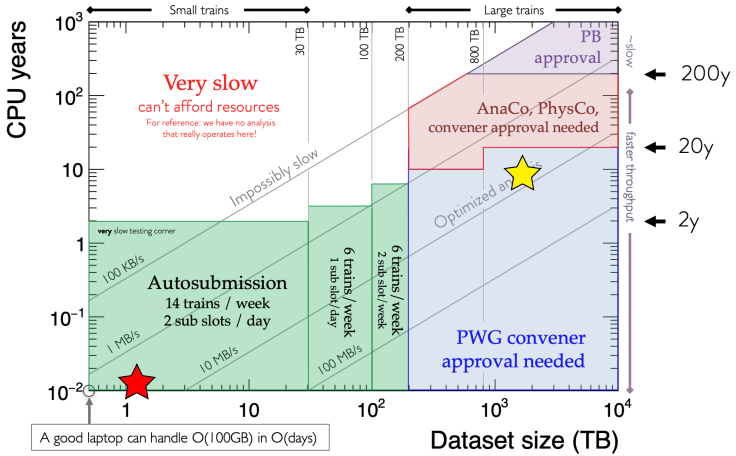
Tasks for UPC event selection are already implemented in the UD code framework

- Using the `SGSelector` class, we were able to select single-gap and double-gap events with < 100 lines of code
- In practice, run already existing tasks, such as `PWGUD/TableProducer/SGCandProducer.cxx`
  - Selects UPC events using `SGSelector` and stores the necessary data in *UD derived tables*
  - Events are kept if they are classified as `SingleGapA`, `SingleGapC` or `DoubleGap`
  - Allows analysis on much smaller data files containing only the interesting events
  - Saves tracks associated to the collision if wanted by user
- *Actually* in practice, this has already been done!
  - E. g. "UD_LHC23_pass5_SingleGap"
  - Reduction factor 1311: 1.6 PB $\rightarrow$ 1.2 TB
- For muons, check out e. g. "UD_LHC23_PbPb_pass5_muon"



Name	Description	Type	Production name
UD	Search 44 records...	HY	Search 44 records...
UD_LHC23_pass5_SingleGap	Sample of Single Gap events based on FT0 (150,50) threshold selection, produced from LHC23_pass5 data. No event selection bits are applied, but some are stored. Contains ITS-only PV Contributors. The analysed lumi (before BC sels) is 768.4 +- 29 /mub. Contains RCT information	HY	Train run 434534
UD_LHC23_pass4_SingleGap	Sample of Single Gap events based on FT0 threshold selection, produced from LHC23_pass4 data. Derived data contain ITS-only PV Contributors and good TPC non-PV tracks; as well as FIT and ZDC information the analysed lumi is 1395.46 +- 54 /mub. Occupancy information added to the derived data. Contains RCT information	HY	Train run 434536
UD_LHC25ae_apass2_SingleGap	Oxygen-Oxygen. ITSROFborders, TFborders, NoSameBunchPileup and IsITSTPCvertex applied, vertex pos2 in (-10, 10) cm. PID not calibrated, default values used. Total TCE triggers: 4138M. Luminosity around 3.83 /nb (~77% of recorded data).	HY	Train run 507694
	Sample of Single Gap events based on FT0 threshold selection, produced from		

One of the goals of initial event selection is to create managable dataset sizes which we can process frequently



SGCandProducer

Your analysis task

After initial selection, more refined event cuts can be done using the `trueGap` function

- Keeping the cuts used in `SGSelector::IsSelected` relatively open allows further cuts to be made on the derived dataset
  - Benefit from huge reduction factor, while keeping some flexibility in analysis
- `SGSelector::trueGap` exists for this purpose

```
template <typename CC>
int trueGap(CC const& collision, const float fv0, const float ft0a,
            const float ft0c, const float zdc_cut)
```

  - Expects to find UD derived data columns in the passed `collision`
- Single gap events can be reclassified as `NoGap` if they don't meet the stricter criteria
- Double gap events can be reclassified as `SingleGapA`, `SingleGapC` or `NoGap` if they don't meet the stricter criteria
- ZDC cut allows e. g. separation into neutron classes 0n0n, 0nXn, Xn0n, XnXn

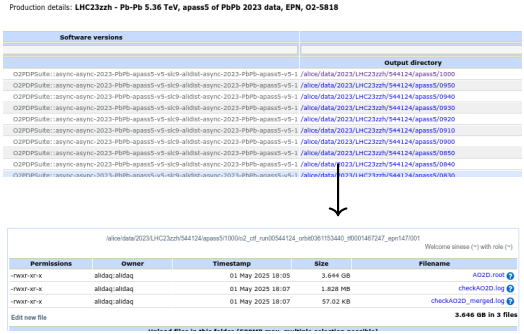
After initial selection, more refined event cuts can be done using the `trueGap` function

- `trueGap` does not remove events – just gives you an integer to use as you wish in your analysis task
  - Can call it multiple times for the same event – e. g. for making several neutron classes

```
namespace o2::aod::sgselector
{
enum TrueGap : int {
  NoGap = -1,          // no gap due to change of threshold(s) in any of FV0, FT0A, ZNA, FT0C, ZNC
  SingleGapA = 0,      // initially single gap at A side event
  SingleGapC = 1,      // initially single gap at C side event
  DoubleGap = 2,       // initially double gap event
  NoUpc = 3,           // initially no UPC event with default thresholds (FT0A=150, FT0C=50)
  TrkOutOfRange = 4,   // to many tracks (>100 default)
  BadDoubleGap = 5     // unknows status of double gap check with changed thresholds
};
} // namespace o2::aod::sgselector
```

Optional exercise: Run and modify the SGSelector example task

- To run UDTutorial_08 on your laptop, you need a raw data file
- For example, use one from LHC23zzh apass5
- Put UDTutorial_08.cxx in O2Physics/Tutorials/PWGUD
- Add the following to O2Physics/Tutorials/PWGUD/CMakeLists.txt and compile:




```
o2physics_add_dpl_workflow(udtutorial-08
  SOURCES UDTutorial_08.cxx
  PUBLIC_LINK_LIBRARIES O2::Framework O2Physics::AnalysisCore O2Physics::SGCutParHolder
  COMPONENT_NAME Analysis)
```

- Run the task using

```
o2-analysis-ud-udtutorial-08 -b | o2-analysis-event-selection-service -b --aod-file /path/to/your/AO2D.root
```

Backup

Definition of trueGap function (O2Physics/PWGUD/Core/SGSelector.h)

```
template <typename CC>
int trueGap(CC const& collision, const float fv0, const float ft0a, const float ft0c, const float zdc_cut)
{
  const float fit_cut[3] = {fv0, ft0a, ft0c};
  int gap = collision.gapSide();
  int true_gap = gap;
  // float FV0A, FT0A, FT0C, ZNA, ZNC;
  const float FV0A = collision.totalFV0AmplitudeA();
  const float FT0A = collision.totalFT0AmplitudeA();
  const float FT0C = collision.totalFT0AmplitudeC();
  const float ZNA = collision.energyCommonZNA();
  const float ZNC = collision.energyCommonZNC();
  if (gap == o2::aod::sgselector::SingleGapA) { // gap == 0
    if (FV0A > fit_cut[0] || FT0A > fit_cut[1] || ZNA > zdc_cut)
      true_gap = o2::aod::sgselector::NoGap;           // -1
  } else if (gap == o2::aod::sgselector::SingleGapC) { // gap == 1
    if (FT0C > fit_cut[2] || ZNC > zdc_cut)
      true_gap = o2::aod::sgselector::NoGap;          // -1
  } else if (gap == o2::aod::sgselector::DoubleGap) { // gap == 2
    if ((FV0A > fit_cut[0] || FT0A > fit_cut[1] || ZNA > zdc_cut) && (FT0C > fit_cut[2] || ZNC > zdc_cut)) {
      true_gap = o2::aod::sgselector::NoGap; // -1
    } else if ((FV0A > fit_cut[0] || FT0A > fit_cut[1] || ZNA > zdc_cut) && (FT0C <= fit_cut[2] && ZNC <= zdc_cut)) {
      true_gap = o2::aod::sgselector::SingleGapC; // 1
    } else if ((FV0A <= fit_cut[0] && FT0A <= fit_cut[1] && ZNA <= zdc_cut) && (FT0C > fit_cut[2] || ZNC > zdc_cut)) {
      true_gap = o2::aod::sgselector::SingleGapA; // 0
    } else if (FV0A <= fit_cut[0] && FT0A <= fit_cut[1] && ZNA <= zdc_cut && FT0C <= fit_cut[2] && ZNC <= zdc_cut) {
      true_gap = o2::aod::sgselector::DoubleGap; // 2
    } else {
      LOGF(info, "Something wrong with DG");
      true_gap = o2::aod::sgselector::BadDoubleGap; // 5
    }
  }
  return true_gap;
}
```

Definition of compatibleBCs function (O2Physics/PWGUD/Core/UDHelpers.h)

```cpp
template <typename C, typename T>
T compatibleBCs(C const& collision, int ndt, T const& bcs, int nMinBCs = 7)
{
  LOGF(debug, "Collision time / resolution [ns]: %f / %f", collision.collisionTime(), collision.collisionTimeRes());

  // return if collisions has no associated BC
  if (!collision.has_foundBC() || ndt < 0) {
    return T{{bcs.asArrowTable()->Slice(0, 0)}, static_cast<uint64_t>(0)};
  }

  // get associated BC
  auto bcIter = collision.template foundBC_as<T>();

  // due to the filling scheme the most probable BC may not be the one estimated from the collision time
  uint64_t mostProbableBC = bcIter.globalBC();
  uint64_t meanBC = mostProbableBC + std::lround(collision.collisionTime() / o2::constants::lhc::LHCBunchSpacingNS);

  // enforce minimum number for deltaBC
  int deltaBC = std::ceil(collision.collisionTimeRes() / o2::constants::lhc::LHCBunchSpacingNS * ndt);
  if (deltaBC < nMinBCs) {
    deltaBC = nMinBCs;
  }
  LOGF(debug, "BC %d,  deltaBC %d", bcIter.globalIndex(), deltaBC);

  return compatibleBCs(bcIter, meanBC, deltaBC, bcs);
}
```

Definition of compatibleBCs function (O2Physics/PWGUD/Core/UDHelpers.h)

```cpp
template <typename B, typename T>
T compatibleBCs(B const& bc, uint64_t const& meanBC, int const& deltaBC, T const& bcs)
{
  // get BCs iterator
  auto bcIter = bcs.iteratorAt(bc.globalIndex());
  // range of BCs to consider
  uint64_t minBC = static_cast<uint64_t>(deltaBC) < meanBC ? meanBC - static_cast<uint64_t>(deltaBC) : 0;
  uint64_t maxBC = meanBC + static_cast<uint64_t>(deltaBC);
  LOGF(debug, "  minBC %d maxBC %d bcIterator %d (%d) #BCs %d", minBC, maxBC, bcIter.globalBC(), bcIter.globalIndex(), bcs.size());
  // check [min,max]BC to overlap with [bcs.iteratorAt(0),bcs.size() - 1]
  if (maxBC < bcs.iteratorAt(0).globalBC() || minBC > bcs.iteratorAt(bcs.size() - 1).globalBC()) {
    LOGF(debug, "<compatibleBCs> No overlap of [%d, %d] and [%d, %d]", minBC, maxBC, bcs.iteratorAt(0).globalBC(), bcs.iteratorAt(bcs.size() - 1).globalBC());
    return T{{bcs.asArrowTable()->Slice(0, 0)}, static_cast<uint64_t>(0)};
  }
  // find slice of BCs table with BC in [minBC, maxBC]
  int64_t minBCId = bcIter.globalIndex();
  int64_t maxBCId = bcIter.globalIndex();
  // lower limit
  if (bcIter.globalBC() < minBC) {
    while (bcIter != bcs.end() && bcIter.globalBC() < minBC) {
      ++bcIter;
      minBCId = bcIter.globalIndex();
    }
  } else {
    while (bcIter.globalIndex() > 0 && bcIter.globalBC() >= minBC) {
      minBCId = bcIter.globalIndex();
      --bcIter;
    }
  }
  // upper limit limit
  if (bcIter.globalBC() < maxBC) {
    while (bcIter != bcs.end() && bcIter.globalBC() <= maxBC) {
      maxBCId = bcIter.globalIndex();
      ++bcIter;
    }
  } else {
    while (bcIter.globalIndex() > 0 && bcIter.globalBC() > maxBC) {
      --bcIter;
      maxBCId = bcIter.globalIndex();
    }
  }
  // create bc slice
  T bcslice{{bcs.asArrowTable()->Slice(minBCId, maxBCId - minBCId + 1)}, static_cast<uint64_t>(minBCId)};
  bcs.copyIndexBindings(bcslice);
  LOGF(debug, "  size of slice %d", bcslice.size());
  return bcslice;
}
```

UD derived data model