XLVIII PANDA Collaboration Meeting (GSI)

New Event Filtering Concept for PandaRoot / FairRoot

Martin J. Galuska, Katja Kleeberg, J. Sören Lange, Wolfgang Kühn
Justus Liebig Universität Gießen

This work was supported in part by BMBF (06GI9107I), HGS-HIRe for FAIR and the LOEWE-Zentrum HICforFAIR.

Motivation for an Event Filter in PandaRoot / FairRoot

- **Problem**: Imagine you want to...
  - ... generate events using **some event generator** (or a combination of multiple event generators).
  - … run the **full PandaRoot simulation** only for a **specific** subset of these **events.**
  - … **time is short.**
- **Example**:
  - Background studies for specific signal channel.
  - Specific signal events extraction (from existing event file).
- **Solution**: **Event filter**
  - You take a **look at** the **particles produced by the event generator(s)** **BEFORE** they are **transport**ed through the detector model and BEFORE digitization, tracking, PID, …
  - **If** the generated **event** is **interesting** for you, you **run** the **full PandaRoot simulation** and run your analysis on this event.
  - **If not**, you **discard** the entire **event** and **rerun** your **event generator(s)**.

Desirable Event Filter Features

- Filter events produced by (a combination of) **arbitrary event generators** based on arbitrary combinations of numerous criteria.
  - See next slide for a list of criteria.
- Combine multiple filters with NOT, AND, OR.
- Define veto filters.
- Accept only desirable events, reject all other events.
- Count how many events were generated in order to reach the user-defined number of accepted events.
- Avoid infinite loops.
  - If no acceptable event can be found (in a user-defined amount of tries), accept the latest ”random” event and warn the user.
- If events are read from a file which does not contain enough suitable events, reduce number of events to be simulated.

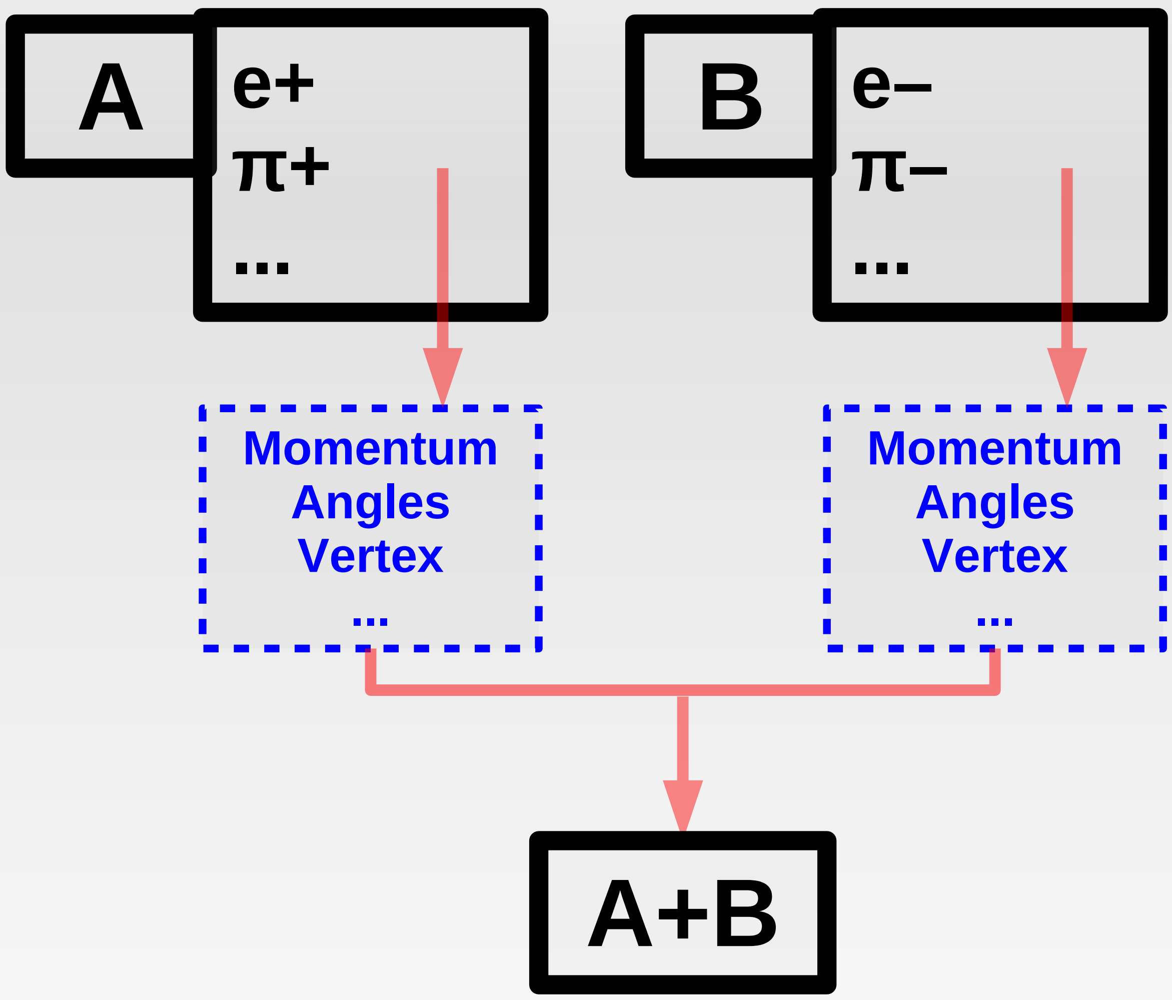
Event Filter Criteria

- On **arbitrary particle combinations**
  - **Invariant masses**
- On **individual particles** or **arbitrary particle combinations**
  - **Momentum** (total / transversal / z) [in lab system]
  - **Angles** (theta, phi) [in lab system]
  - **Vertex** (z / rho / r)
  - **PDG code** / **Charge** (neutral / + / – / charged)
  - Angles + momenta in arbitrary **center of mass system**

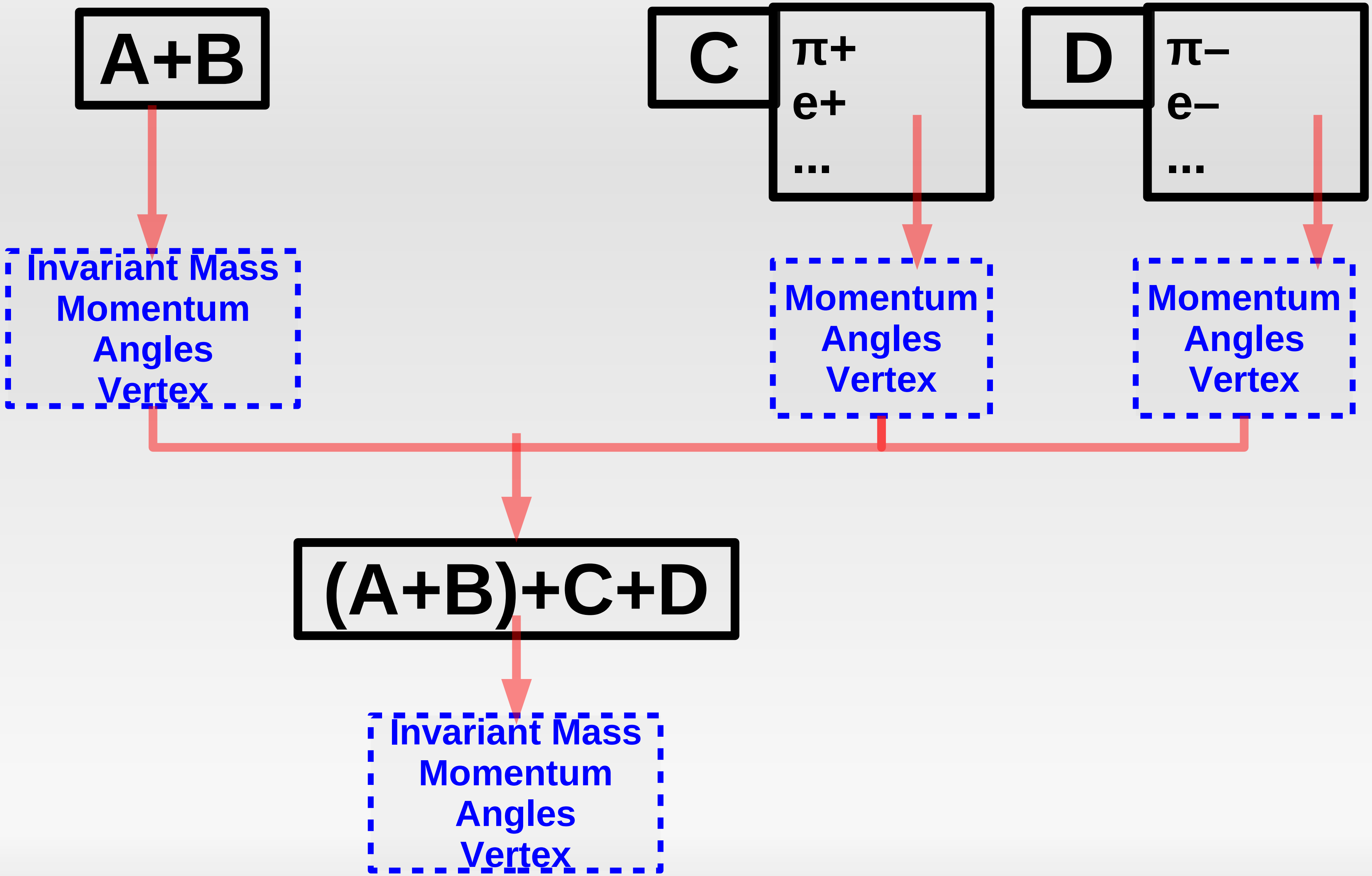
Option A: Pre-Implemented Event Filters

- **Idea**
  - We pre-implement every possible filter a user might want.
  - The user configures the filter via the sim macro.
- **How it works**
  - Filters act on **particle lists**.
  - Filters need to be ”**piped**” and combined with **AND**, **OR**, **NOT**.
  - At the end of a ”filter chain” the **number of entries** in each list will be checked against a **minimum** and a **maximum** value.
- Problems
  - Configuring the filters becomes very complicated.
  - Users need to learn a new ”meta-programming language”.

Example: ψ(2S) → J/ψ (e+ e–) π+ π–



Example: ψ(2S) → J/ψ (e+ e−) π+ π−



Option A: Conclusion

- Problem with predefined filters:
  - Predefined event filters will never have all desirable features.
  - An event filter can be just as complex as an analysis.
    - Invariant masses of multiple particle combinations, …
- How we do analysis:
  - We don't try to precode every possible analysis.
  - Instead we give users the tools to write their own analysis easily.
  - The same tools should be available for writing user-defined event filter(s).
- What users need to do to implement their own event filter:
  - User-defined filters need to be derived from a virtual class FairEvtFilter and implement a Bool_t acceptEvent() method.
  - Users can use the existing analysis code around RhoCandidate for writing their own event filter.
  - Users add their filter(s) to FairPrimaryGenerator from the sim macro.

Option B: User-Implemented Event Filters

- We implement a **framework for event filters**.
- The **users implement their own filters** (as derived classes).
  - Using pre-existing already tested code.
  - Using pre-existing knowledge.
  - The custom filters can be added from the sim macro.

Event Filter Framework

- User adds a combination of event filters and a selection of event generators to FairPrimaryGenerator from sim macro.
- FairPrimaryGenerator calls the event generator(s).
- The event generator(s) put(s) the event as TCA of TParticle onto the PndStack.
- FairPrimaryGenerator gets the TParticle from the stack, converts them to RhoCandidate and passes them to all active event filters.
- FairPrimaryGenerator calls acceptEvent() of all active event filters.
- The event filters look at the RhoCandidate and decide whether the event is interesting (return kTRUE) or not (return kFALSE).
  - We (probably) only need a new FillList() method.
  - Existing analysis code can be used for event filtering.
- FairPrimaryGenerator does the necessary logical connections between all active filters. FairPrimaryGenerator either accepts the event or resets the stack and calls the event generators again.

Option C: Combining the Advantages of A and B

- We provide a framework for users to implement their own event filters with tools they are familiar with from analysis.
  - We provide **AND**, **OR** and **NOT** for combining filters.
  - We don't ”pipe” different filters.
- We provide a **fairly flexible event filter** that **satisfies most wishes** and **works out of the box**.
  - Does everything for single particles
  - Does not handle combinations of arbitrary particle lists.

Criteria in Predefined Event Filter

Already implemented / Ongoing / Planned / Missing

- Single particle properties
  - Momentum (total / transversal / z) [in lab system / arb. CMS]
  - Angles (theta, phi) [in lab system / arb. CMS)
  - Vertex (z / rho / r)
  - (List of) PDG code(s) / Charge (neutral / charged / + / –)
  - Geometry + momenta in center of mass system for arbitrary 4 vectors
- Filter on particle combinations
  - Invariant masses (of arbitrary combinations)
  - + same criteria as for single particles

Event Filter Strategy for PandaRoot and Status

Already implemented
Ongoing
Planned

- Add event filtering capabilities to FairPrimaryGenerator.
  - And / or / not for connecting multiple filters.
- Implement new virtual filter class for FairRoot.
- Implement general use filter for single particle properties.
  - Vertices (z / rho / r).
  - Momentum (total / transversal / z) [in lab system].
  - Geometry (theta, phi) [in lab system].
  - List of PDG code(s) / Charge (neutral / charged / + / –).
- Provide a framework so that users can implement their own sophisticated event filters easily using the familiar tools from analysis.
- Handle cases in which not enough events can be produced (e.g. events are read from a file).

Example: Require Min. 1 Λ and Min. 1 $\bar{\Lambda}$ of Certain Momenta

- In sim macro:

```
// THE SAME AS ALWAYS

FairPrimaryGenerator* primGen = new FairPrimaryGenerator();
fRun->SetGenerator(primGen);

// here you put the generator that you want to use, let's say DpmDirect
PndDpmDirect *Dpm= new PndDpmDirect(mom,1);
primGen->AddGenerator(Dpm);

// The standard behaviour is the same as before (i.e. no event filtering)

// now add some filters on multiplicities of charged/neutral particles
// or pdg codes within certain momentum ranges

// NEXT SLIDE
```

Example: Require Min. 1 Λ and Min. 1 $\bar{\Lambda}$ of Certain Momenta

- In sim macro:

```
... // THE SAME AS ALWAYS

// now add filter on multiplicities of lambda and anti-lambda

FairEvtFilterOnCounts* min1Ld_1aLd = new FairEvtFilterOnCounts();

min1Ld_1aLd->AndMinPdgCodes(1, 3122); // request min. 1 lambda

min1Ld_1aLd->AndMinPdgCodes(1, -3122); // request min. 1 anti-lambda

min1Ld_1aLd->SetPtRange(0.5, 1.0); // with a p_T within [0.5, 1.0] GeV/c

min1Ld_1aLd->SetPRange(1.5, 3.0); // and with a p within [1.5, 3.0] GeV/c

primGen->AndFilter(min1Ld_1aLd);
```

More examples will be made available soon.

Setting Parameters and Getting Info

- You can tell FairPrimaryGenerator how often it should try to find a suitable event before giving up:
  ```
  primGen->SetFilterMaxTries(99999);
  ```
- and the FairPrimaryGenerator can tell you at the end of the simulation macro how many events the generator simulated (in total) to get the number of filtered events that you wanted...
  ```
    cout << primGen->GetNumberOfSimulatedEvents() << " events were simulated by the generators.\n";
  ```
- … as well as how many events reached the limit of tries without success:
  ```
    // should be 0 if filter is applicable and SetFilterMaxTries is not too low
    cout << primGen->GetNumberOfFilterFailedEvents() << " unsuccessful attempts to find an event that suits your filters.\n\n";
  ```
- Note: The number of generator runs and the number of failed filterings will also be written into the root output file.

On the Filter Usage: A Word of Caution

- The event filter is a versatile, but dumb tool.
- As a potential user remember that:
- The event filter **only looks at particles** produced by the event generator(s).
  - It does not know about any non-generator created particles (because it analyses the event BEFORE the transport engine is run)!
  - It does not take material effects / interaction with detector / tracking efficiencies / momentum resolutions / PID / etc. into account.
- **Be careful** when you **extract physical meaning** from an analysis of filtered events!
  - In case of the DPM generator, the user should analyse ALL events from DPM with sufficient statistics and based on the results, (s)he can decide that mainly certain event topologies contribute (after cuts) to the background events for the analysed channel.
  - Once such specific events were identified, the filter can be useful in saving simulation time and computer ressources.

Summary and Outlook

- A new concept for event filtering in PandaRoot was suggested.
- The code will be usable within a couple of...
  - … **days** (reduced functionality for single particle properties).
  - … **weeks** (full functionality for single particles).
  - at some point (for user-defined easy to implement filters).

- We (Katja and I) will provide:
  - The event filter extension of FairPrimaryGenerator.
  - An event filter for single particle properties.
  - A framework for user-defined event filters.
  - Tutorials on event filter usage and implementation.

Thank You!

- We hope the features will be useful for background analysis and many more applications in the future and can be adopted into the **trunk/base/sim** code.
- The preliminary implementation can be found here:
  https://subversion.gsi.de/trac/fairroot/browser/pandaroot/development/mgaluska/eventFilter

  (Read the howto.txt for installation notes.)
- Discussion and status updates are here:
  https://forum.gsi.de/index.php?t=msg&th=4135&goto=15933

BACKUP SLIDES

Example Usage of an Event Filter on Single Particle Properties

- Imagine that we are only interested in events that have:
  - at least 2 e- OR pi-
    - with a p_T within [1.0, 3.0] GeV/c
    - and with a p within [1.5, 3.0] GeV/c
  - at most 4 particles
    - with a p_z within [0.5, 4.0] GeV/c
  - at least 4 and at most 6 pos. Charged particles
    - with theta within $[20.0, 90.0]^0$
    - and phi within $[10.0, 80.0]^0$
  - at least 3 and at most 10 gamma
    - With E>2 GeV

Example Usage Code

- In sim macro:

```
FairPrimaryGenerator* primGen = new FairPrimaryGenerator();
fRun->SetGenerator(primGen);

// here you put the generator that you want to use, let's say DpmDirect
PndDpmDirect *Dpm= new PndDpmDirect(mom,1);
primGen->AddGenerator(Dpm);

// The standard behaviour is the same as before (i.e. no event filtering)

// now add some filters on multiplicities of charged/neutral particles or pdg codes
within certain momentum regions

// NEXT SLIDE
```

Example Usage Code

- In sim macro:

…

// now add some filters on multiplicities of charged/neutral particles or pdg codes within certain momentum regions

```
FairEvtFilterOnCounts* min2eM_piP = new FairEvtFilterOnCounts();
min2_eM_piP->AndMinPdgCodes(2, 11, -211); // request at least 2 e- OR pi-
min2_eM_piP->SetPtRange(1.0, 3.0); // with a pT within [1.0, 3.0] GeV/c
min2_eM_piP->SetPRange(1.5, 3.0); // and with a p within [1.5, 3.0] GeV/c
primGen->AndFilter(min2_eM_piP);
```

```
FairEvtFilterOnCounts* max4Particles = new FairEvtFilterOnCounts();
max4_eP_piP->AndMaxAllParticles( 4 ); // request at most 4 particles
max4_eP_piP->SetPzRange(0.5, 4.0); // with a pz within [0.5, 4.0] GeV/c
primGen->AndFilter(max4_eP_piP);
```

```
FairEvtFilterOnCounts()* min4_max6_P = new FairEvtFilterOnCounts();
min4_max6_P->AndMinMaxCharge(4,6,'+'); // request at least 4 and at most 6 pos. Charged particles
min4_max6_P->SetThetaRange(20.0, 90.0); // with theta within [20.0, 90.0]°
min4_max6_P->SetPhiRange(10.0,80.0); // and phi within [10.0, 80.0]°
primGen->AndFilter(min4_max6_P);
```

```
FairEvtFilterOnCounts()* min3_max10_gamma = new FairEvtFilterOnCounts();
min3_max10_gamma->AndMinMaxPdgCodes( 3, 10, 22 ); // request at least 3 and at most 10 gamma
min3_max10_gamma->SetPRange(2.0, 9999.0); // and with a p within [2.0, 9999.0] GeV/c
primGen->AndFilter(min3_max10_gamma);
```

Reminder:

- Imagine that we are only interested in events that have:
  - at least 2 e- OR pi-
    - with a p_T within [1.0, 3.0] GeV/c
    - and with a p within [1.5, 3.0] GeV/c
  - at most 4 particles
    - with a p_z within [0.5, 4.0] GeV/c
  - at least 4 and at most 6 pos. Charged particles
    - with theta within $[20.0, 90.0]^0$
    - and phi within $[10.0, 80.0]^0$
  - at least 3 and at most 10 gamma
    - With E>2 GeV