Kinematic Vertexing

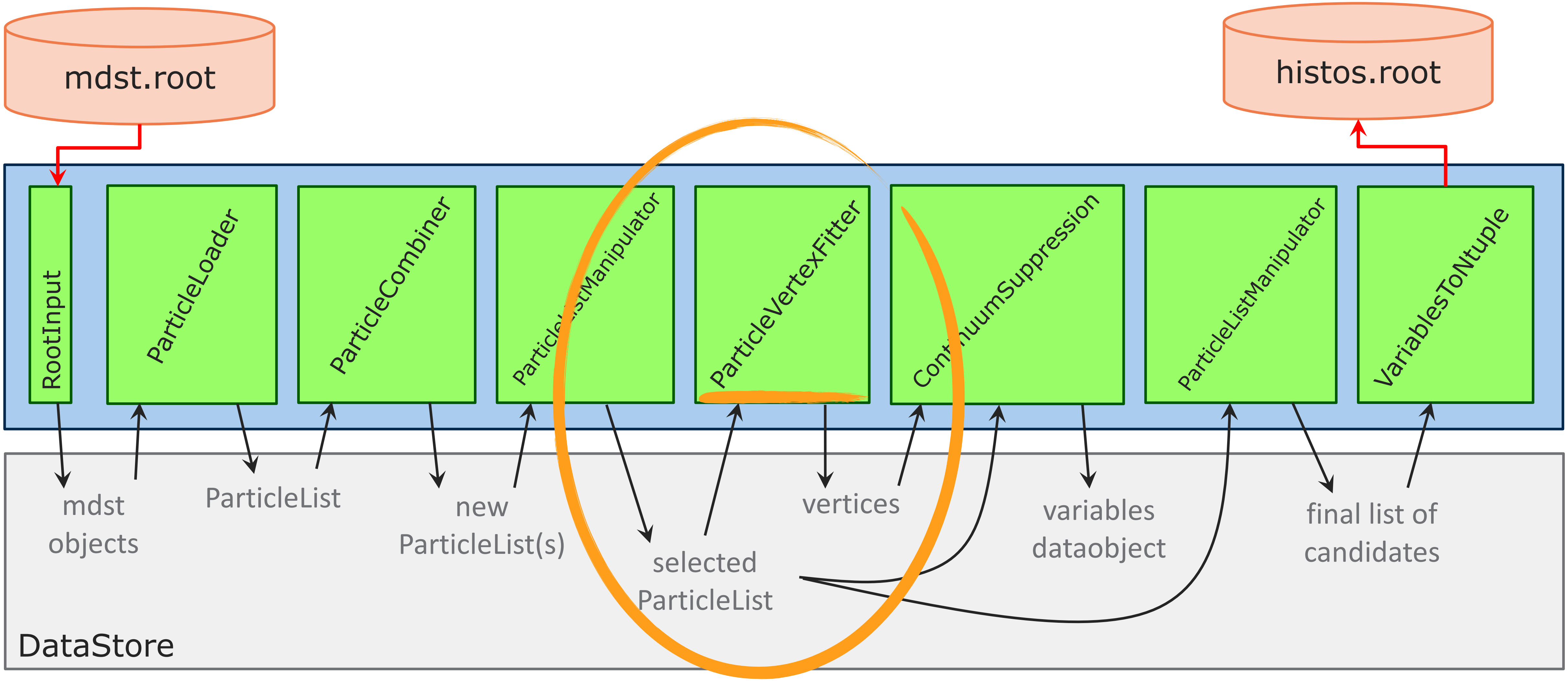
5th Starter Kit Workshop

Francesco Tenchini
31st January, 2020

A longer and more advanced version of this talk is available at
https://indico.belle2.org/event/493/contributions/4545/ for those interested.

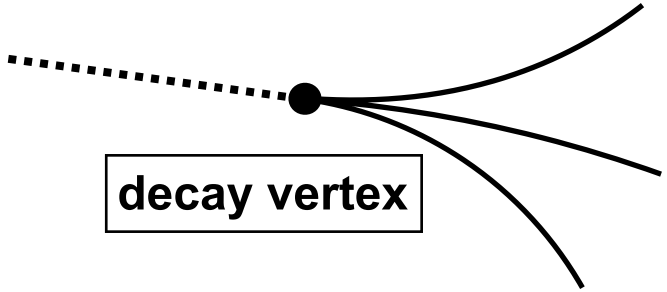
HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

Previously on this Starter Kit...



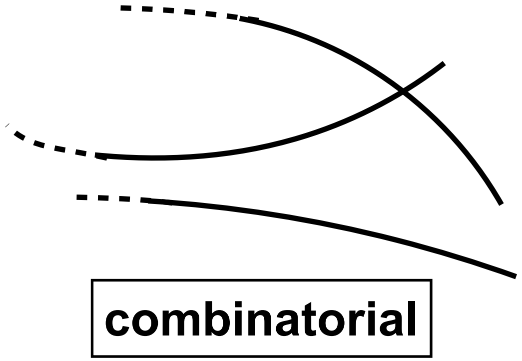
What is Vertexing?

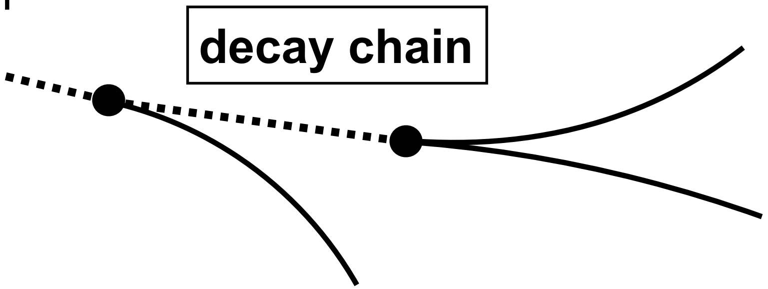
- Combine particle measurements under the assumption that they originate from a common point (or a set of points) → incorporate additional information in your measurement.
- **Inputs:** track helix, energy deposits, associated measurement covariances
- **Outputs:** vertex position, 4-momentum, covariance matrix



When is it useful?

- Improving decay vertex position resolution
- Lifetime measurements
- Mass measurements
- (Combinatorial) Background rejection
- ...

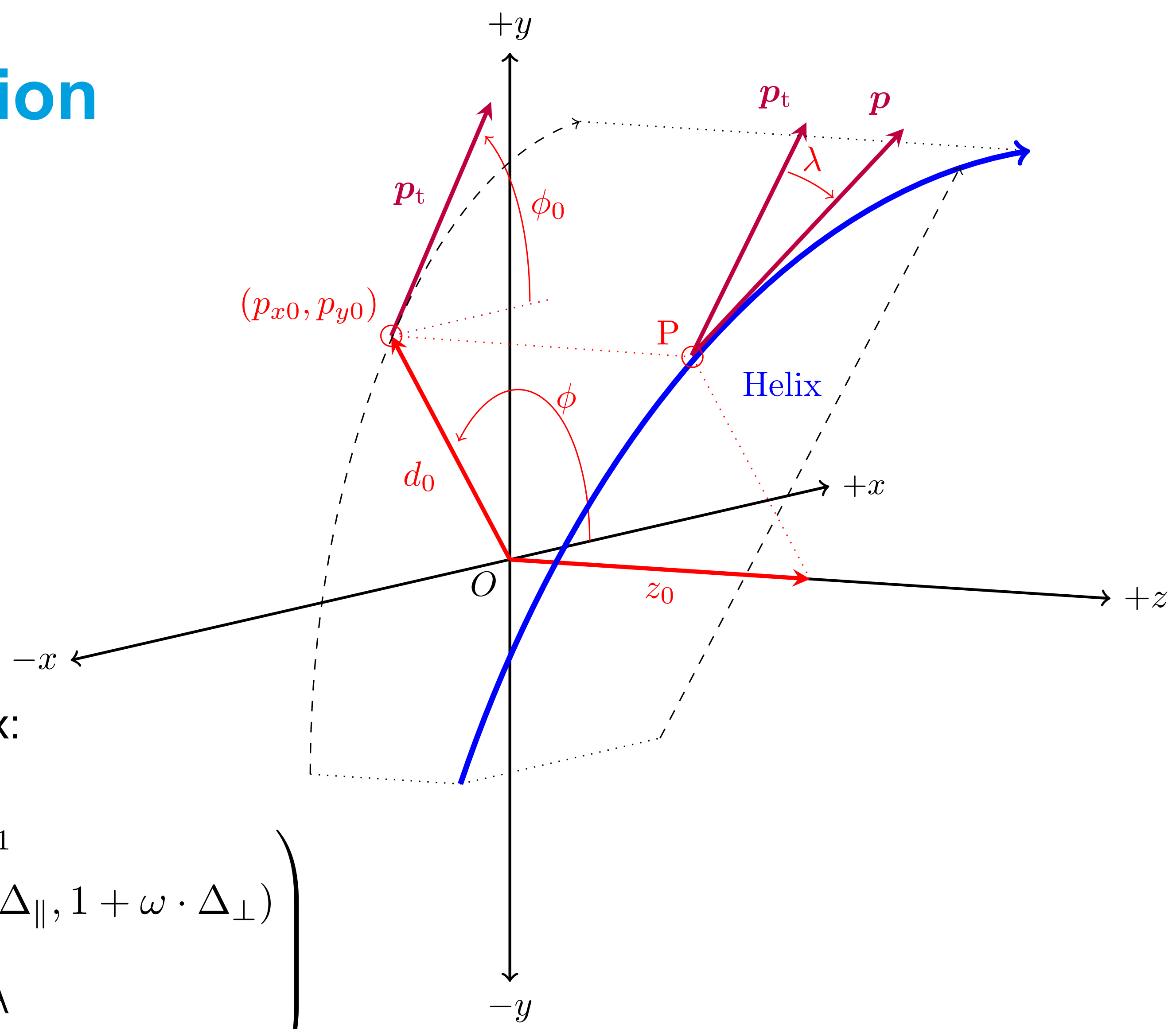




Example: Track Parametrisation

- Track equations are linear for a free body:

$$\boldsymbol{h}_{track}(\boldsymbol{x}) = \begin{pmatrix} x_0 \\ y_0 \\ t_x \\ t_y \\ p \end{pmatrix} = \begin{pmatrix} x - zp_x/p_z \\ y - zp_y/p_z \\ p_x/p_z \\ p_y/p_z \\ p \end{pmatrix}$$



- More complicated with a magnetic field - 5D helix:

$$\boldsymbol{h}_\mathrm{track}(\boldsymbol{x}) = \begin{pmatrix} d_0 \\ \phi_0 \\ \omega \\ z_0 \\ \tan\lambda \end{pmatrix} = \begin{pmatrix} A(1+U)^{-1} \\ \mathrm{atan2}(p_y, p_x) - \mathrm{atan2}(\omega \cdot \Delta_\parallel, 1 + \omega \cdot \Delta_\perp) \\ a \cdot q/p_t \\ z + l \cdot \tan\lambda \\ p_z/p_t \end{pmatrix}$$

Fitting the Measurements

- We want to minimise a chi square, or in other words:

$$\chi^2_{global} = (\vec{m} - \vec{h})\, V^{-1} (\vec{m} - \vec{h})^T \longrightarrow \frac{\partial \chi^2_{global}}{\partial \vec{x}_{params}} = 0$$

- This is the "fit" part. I'll gloss over the algebra in this talk.
- If your measurement is nonlinear (e.g. tracks in a B field) finding the solution can be challenging
  - Computationally we solve this by **linearisation** and **iteration.**
- Either way, the analytical solution requires matrix inversion
  - If there are many parameters, this is slow.
  - Partially solved by techniques such as Kalman filtering.

track

$$\begin{pmatrix} A(1+U)^{-1} \\ \mathrm{atan2}(p_y, p_x) - \mathrm{atan2}(\omega \cdot \Delta_{\parallel}, 1 + \omega \cdot \Delta_{\perp}) \\ a \cdot q/p_t \\ z + l \cdot \tan\lambda \\ p_z/p_t \end{pmatrix}$$

- Practical takeaway message: **vertexing has a time cost**. Trim your sample before fitting.
- **`basf2:`** after the fit, your minimised χ^2 is converted to a p-value and saved as the **`chiProb`** variable.

V0?

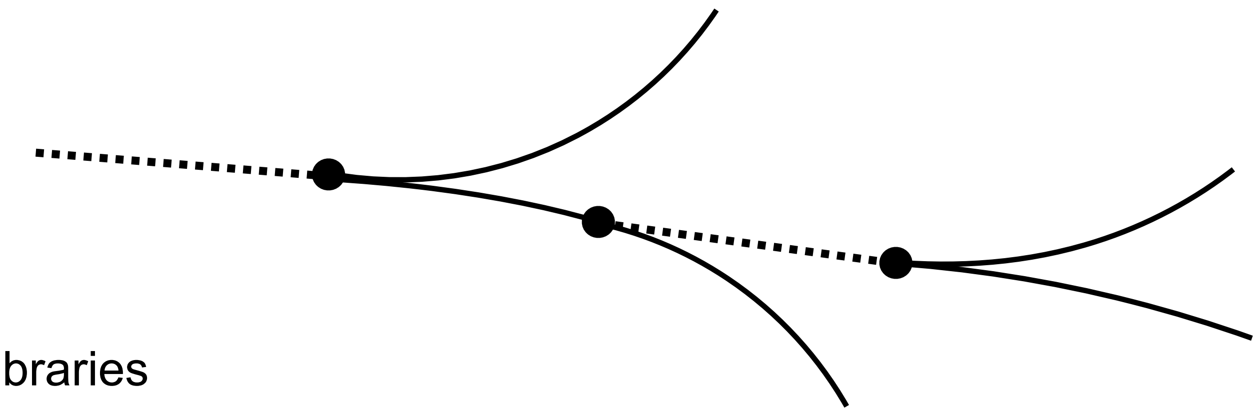
- Analyst is provided with track objects (5D helices).
- Information on the individual track hits is not available. Track fit can't be repeated at analysis level.
- Tracks are reconstructed assuming they originate from the IP. If they don't (displaced vertex), the energy loss due to material effects is overestimated → a bias is introduced
- V0Finder locates and fits opposite charged pairs to produce V0 (Ks, Λ, γ) candidates at tracking level.
- You can then load them in your analysis with a syntax such as:

```
ma.fillParticleList('K_S0:V0 -> pi+ pi-', cut=<yourcut>, path=<yourpath>)
```

- Compared to building the Ks (or Λ) at analysis level, these have lower efficiency but higher quality.
- Standard lists are available through convenience functions (`stdKshorts, stdLambdas`) merging the two approaches. See the documentation for more details.

Vertexing Tools at Belle II

- KFit:
  - Basic kinematic fitter
  - Inherited from Belle
- RAVE:
  - Progressive single vertex fit
  - External package from CMS vertexing libraries
  - Deprecated for analysis use
- TreeFitter:
  - Progressive fit of the decay chain



KFit

- Single vertex fit inherited from Belle.
- Non-iterative: pure matrix inversion.
- Called through:

old (struck through):

```
vertex.vertexKFit(list_name, conf_level, decay_string='', constraint='', path=None)
vertex.vertexKFitDaughtersUpdate(list_name, conf_level, constraint='', path=None)
vertex.massKFit(list_name, conf_level, decay_string='', path=None)
vertex.massKFitDaughtersUpdate(list_name, conf_level, decay_string='', path=None)
vertex.massVertexKFit(list_name, conf_level, decay_string='', path=None)
vertex.massVertexKFitDaughtersUpdate(list_name, conf_level, decay_string='', path=None)
```

new:

```
vertex.KFit(list_name, conf_level, fit_type='vertex', constraint='', daughtersUpdate=False, decay_string='', path=None)
```

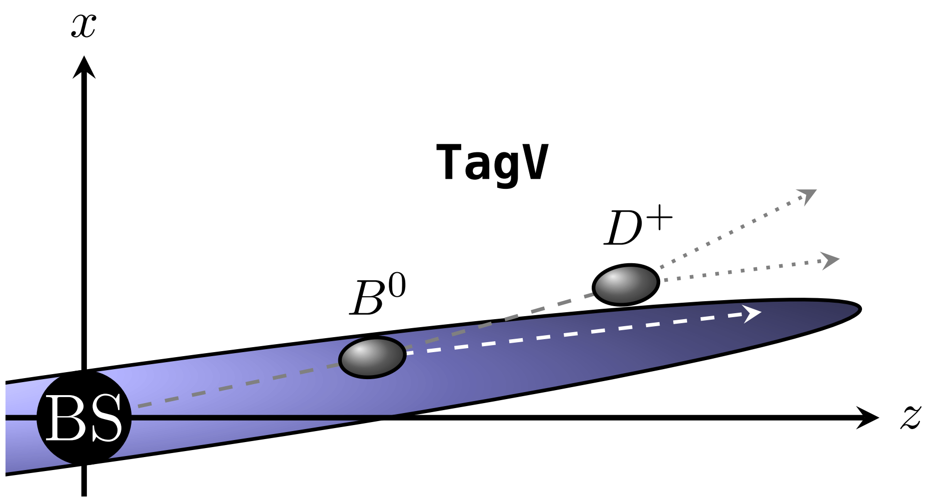
- Can perform mass constrained fits
- Can perform IP constrained fits (more on this later).
- Need to fit a single vertex? KFit is (usually) fine.

RAVE

Warning

RAVE is deprecated since it is not maintained. Whilst we will not remove RAVE, it is not recommended for analysis use, other than benchmarking or legacy studies. Instead, we recommend Tree Fitter (`vertex.treeFit`) or `vertex.KFit` .

- External package from CMS vertexing libraries
- Progressive single vertex fit (Kalman)
  - Actually *slower* than KFit, due to API overhead.
- Still required for some key applications:
  - **V0Finder** through Genfit integration in tracking.
  - TagV using adaptive vertex fitting.
    - If you want to do TDCVP you probably need this, but I won't cover it here.



TreeFitter

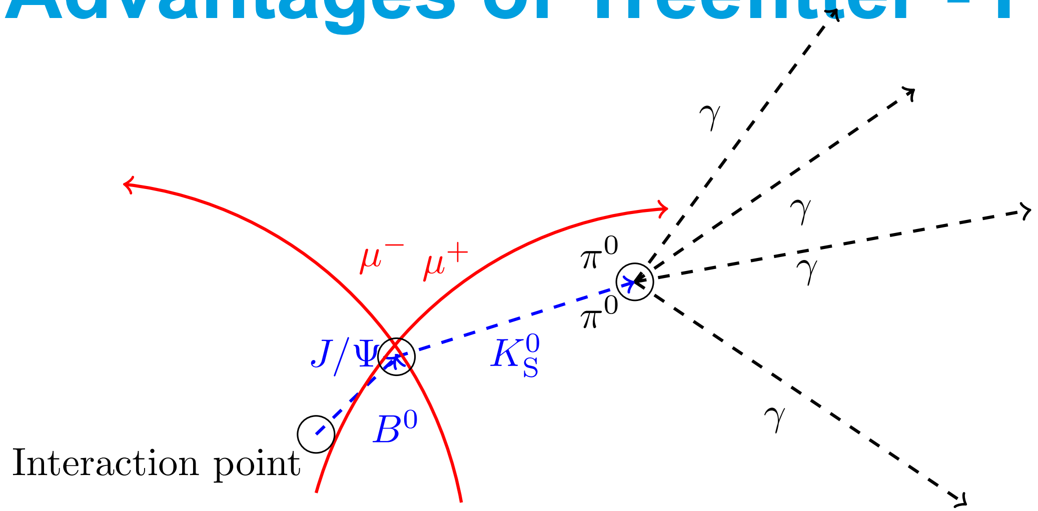
- Kalman based filter for a whole particle decay chain ("tree").
- Automatically builds the tree structure based on provided logic (hypothesis-based).
  - Can even be a sum of different decay channels - the fitter will know from particle relations.

```
import basf2 as b2
import modularAnalysis as ma

main = b2.create_path()

ma.fillParticleList('K-:all', 'some cut', path=main)
ma.fillParticleList('pi-:all', 'some cut', path=main)
ma.reconstructDecay('D0:kpi -> K-:all pi+:all', 'some cut', path=main)
ma.reconstructDecay('D*+:D0pi -> D0:kpi pi+:all', 'some cut', path=main)

ma.treeFit(
    list_name='D*+:D0pi',
    conf_level=-1,
    massConstraint=['D0'],
    ipConstraint=True,
    updateAllDaughters=True,
    path=main,
)
```

your decay reconstruction

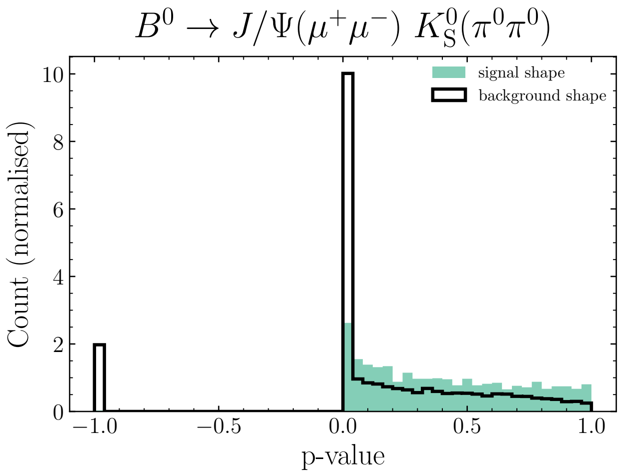
- conf_level: minimum confidence level, -1 = not converged
- massConstraint: takes names or pdg codes
- ipConstraint: beam constraint
- updateAllDaughters: update daughter kinematics

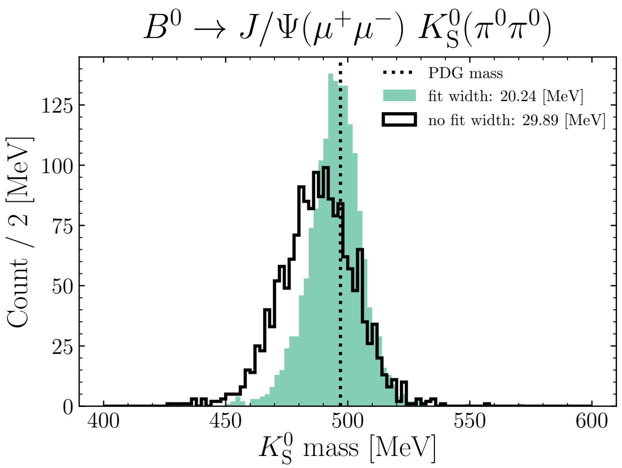
Advantages of Treefitter - Fitting π^0



- Consider the decay $B^0 \to J/\psi$ Ks, Ks$\to \pi^0\pi^0$

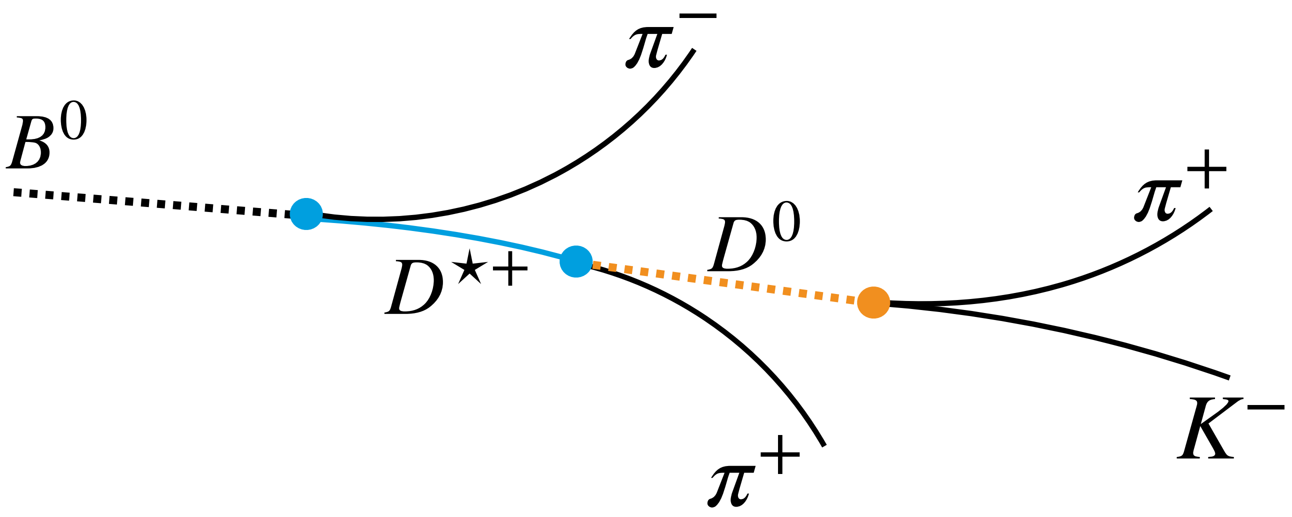
- Since photons are assumed to come from (0,0,0), this will introduce a bias on the Ks mass.
- KFit can refit ONE neutral pion to the fitted vertex position, but no more, and only if tracks are present.
    - TreeFitter can handle the fit provided the π^0 is mass constrained.





Parametrising the Decay Tree

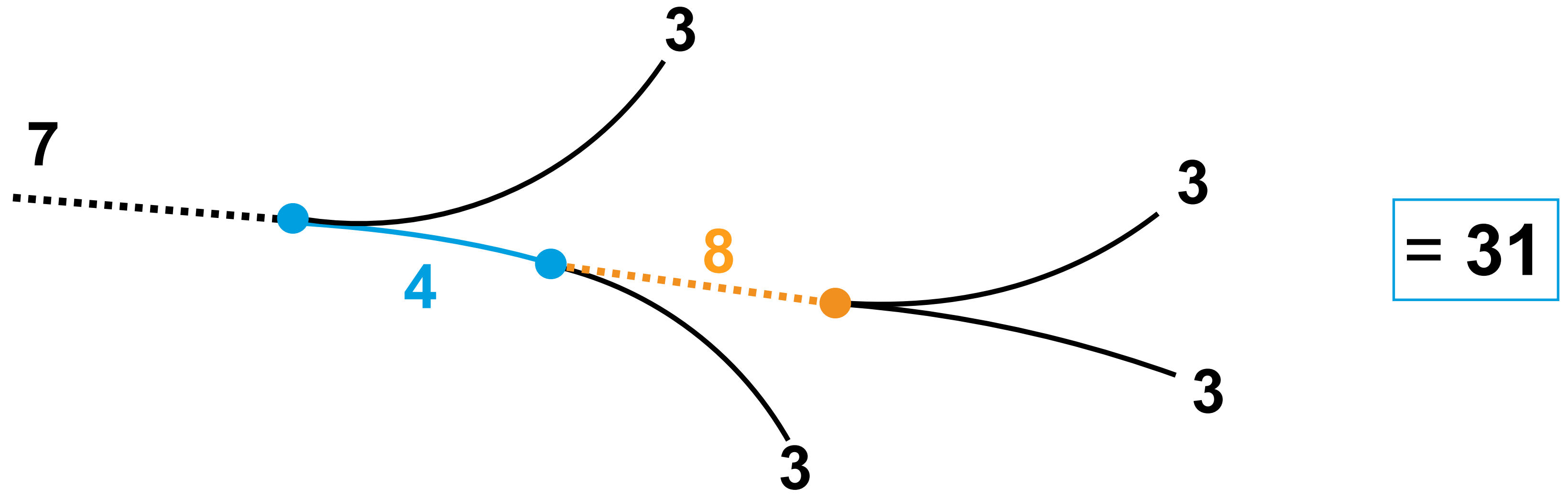
- There are several ways. This is one of them.
- For each particle, assign a 3-momentum {px,py,pz}
  - If the particle is a final state, use the nominal mass. Otherwise, assign an energy {E}.
- If the particle is short lived ("resonance") do nothing more.
- If it's long lived ("composite") assign a decay vertex {x,y,z} and a flight length {θ}.
- If it's the head of the decay, assign a decay vertex {x,y,z}.



Mini-Exercise: Can you count the parameters of this decay?

Parametrising the Decay Tree

- There are several ways. This is one of them.
- For each particle, assign a 3-momentum {px,py,pz}
  - If the particle is a final state, use the nominal mass. Otherwise, assign an energy {E}.
- If the particle is short lived ("resonance") do nothing more.
- If it's long lived ("composite") assign a decay vertex {x,y,z} and a flight length {θ}.
- If it's the head of the decay, assign a decay vertex {x,y,z}.



Mass Constraint

$$r^{\alpha}(\boldsymbol{x}) = m^2_{\mathrm{PDG}} - E^2 + |\boldsymbol{p}|^2 - \ \ldots$$

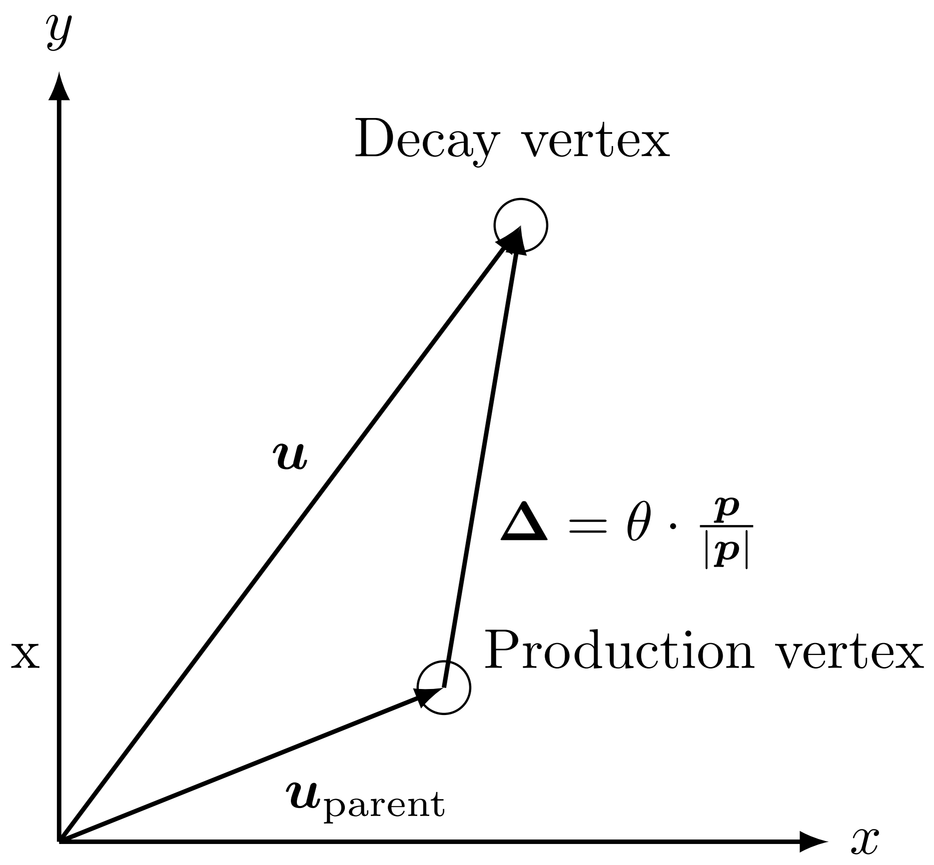
- Some things to note:
  - Fixing the mass to the nominal value only makes sense if the particle has a narrow mass peak. If the width is measurable, then a mass constraint will cause a bias.
  - If you fit on the mass, your mass will be a delta, you can't use it anymore...

Geometric Constraint (Flight Length)

- Not really a constraint, but rather a way the TreeFitter accounts for correlations between flight parameters.
- Applied automatically: $0 = u_{parent} + \Delta - u$
- We have a very good handle on momentum uncertainties:
  - Measure on the momentum projection

$$r(x) = u_{parent} + \theta \cdot \frac{p}{|p|} - u$$

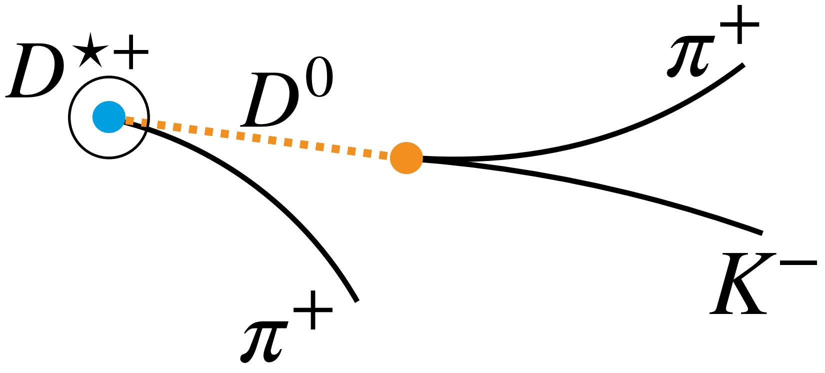
- 1 new parameter {θ} but 3 equations: **-2 degrees of freedom!**



- After the fit, basf2 fills `ExtraInfo("decayLength")` and `ExtraInfo("decayLengthErr")`
  - ... or you can use the variables `flightDistance` and `flightDistanceErr`

Beam Spot Constraint

- This means different things in different fitters.
- General concept: use the beam spot information to constrain the vertex. But how?



- **KFit** has 3 versions:
  - **ipconstraint -** constrain the vertex to the beam spot.
  - **iptube -** as above, but only on the x-y plane.
  - **pointing -** constrain the momentum vector to pass through the beam spot.
- **TreeFitter** handles it by creating a new "Origin" particle to act as the new head of the decay.
  - If the old head was short lived, this is equivalent to **ipconstraint**.
  - If it was long lived, it's more akin to **pointing** (and removes 2 degrees of freedom because of the flight length constraint)

Can I Fit This? Degrees of Freedom

- When in doubt, count the degrees of freedom: NDF = N(equations) - N(parameters)
- If NDF>0 you can fit, otherwise you need to add a constraint (mass, beamspot, ...)

Mini-Exercise: Can you fit these decays?

- $\pi^0(\gamma\gamma)$?
- $D^0 \to K\pi\pi^0(\gamma\gamma)$?
- $D^0 \to K_s(\pi\pi)\pi^0(\gamma\gamma)$?

	parameters	equations	net
track	{px, py, pz}	5 (helix)	2
neutral	{px, py, pz}	3 (energy + cluster position)	0
resonance	{E, px, py, pz}	4 (energy conservation)	0
long lived	{E, px, py, pz} +{x,y,z,θ}	4 (energy conservation) +3 (flight)	-1
head	{E, px, py, pz} +{x,y,z}	4 (energy conservation)	-3
mass		1	1
beamspot		0 or 2 (flight)	0/2

Can I Fit This? Degrees of Freedom

- When in doubt, count the degrees of freedom: NDF = N(equations) - N(parameters)
- If NDF>0 you can fit, otherwise you need to add a constraint (mass, beamspot, ...)

Mini-Exercise: Can you fit these decays?

- $\pi^0(\gamma\gamma)$?
  - [-3+0+0] = **-3** → **NO, not even with a mass constraint**
- $D^0 \to K\pi\pi^0(\gamma\gamma)$?
  - [-3+2*2+0+2*0] = **1** → **YES**
- $D^0 \to K_s(\pi\pi)\pi^0(\gamma\gamma)$?
  - [-3-1+2*2+0+2*0] = **0** → **YES, if mass constrained**

	parameters	equations	net
track	{px, py, pz}	5 (helix)	2
neutral	{px, py, pz}	3 (energy + cluster position)	0
resonance	{E, px, py, pz}	4 (energy conservation)	0
long lived	{E, px, py, pz} +{x,y,z,θ}	4 (energy conservation) +3 (flight)	-1
head	{E, px, py, pz} +{x,y,z}	4 (energy conservation)	-3
mass		1	1
beamspot		0 or 2 (flight)	0/2

Recommendations and Conclusions

- Remember that **garbage in is garbage out**, and may slow down the execution and/or cause fit failures.
  - Do some preselection before fitting.
- Be careful when comparing p-values of different fitters (or different TreeFitt-ed channels); they might have different distributions. Make sure you're not comparing apples to oranges before cutting over it.
- You might come across a **`vertex.fitVertex`** function. This is not recommended for use, although I occasionally see it being copy-pasted around. Please use **`vertex.KFit`** or **`vertex.treeFit`**.
  - I'd really like to get rid of it someday...
- Don't use TreeFitter in release-03 (**bug alert!**). But you shouldn't be using old releases anyways.

- If you have questions, please ask.
- ... or visit questions.belle2.org .

Thank you for your attention!

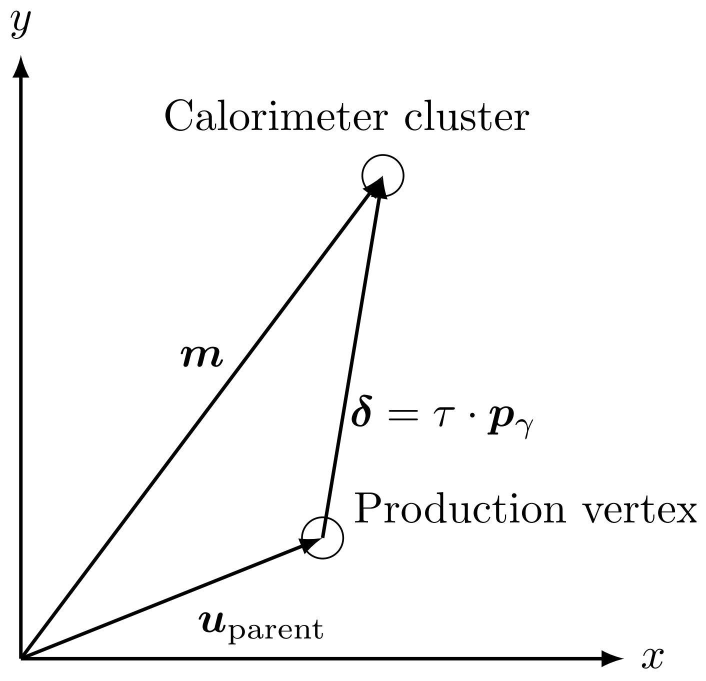
Backup

Neutral Final States

- We already discussed the charged track:

$$\boldsymbol{h}_{\text{track}}(\boldsymbol{x}) = \begin{pmatrix} d_0 \\ \phi_0 \\ \omega \\ z_0 \\ \tan\lambda \end{pmatrix} = \begin{pmatrix} A(1+U)^{-1} \\ \text{atan2}(p_y, p_x) - \text{atan2}(\omega \cdot \Delta_{\parallel}, 1 + \omega \cdot \Delta_{\perp}) \\ a \cdot q/p_t \\ z + l \cdot \tan\lambda \\ p_z/p_t \end{pmatrix}$$

- For neutral clusters, we have:



$$0 = u_{parent} + \delta - m$$

$$\boldsymbol{h}_{\text{photon}}(\boldsymbol{x}) = \begin{pmatrix} u_x + \tau \cdot p_x \\ u_y + \tau \cdot p_y \\ u_z + \tau \cdot p_z \\ \sqrt{p_x^2 + p_y^2 + p_z^2} \end{pmatrix} \quad \text{and} \quad \boldsymbol{m}_{\text{photon}}(\boldsymbol{x}) = \begin{pmatrix} m_x \\ m_y \\ m_z \\ E_m \end{pmatrix}$$

Or KLong, or... neutron?

$$\boldsymbol{r}'^{\alpha}_{\text{photon}}(\boldsymbol{x}) = \begin{pmatrix} (m_i - u_i) - (m_k - u_k)\frac{p_i}{p_k} \\ (m_j - u_j) - (m_k - u_k)\frac{p_j}{p_k} \\ E_m - \sqrt{p_i^2 + p_j^2 + p_k^2} \end{pmatrix} + \boldsymbol{H}^{\alpha-1} \cdot (\boldsymbol{x}^{\alpha}_{k-1} - \boldsymbol{x}^{\alpha-1})$$

3 equations

Obtained by factoring out the highest momentum component.

Flight Length: Geometric Constraint (2)

- But what about charged, long lived particles?
- Particles such as $\Sigma+$ can travel for several cm and are affected by the magnetic field.
- Unfortunately at the moment **all composites are assumed to fly straight**. (Jira ticket BII-3893)
- If you're interested in working on channels with long lived charged composites, the feature will have to be developed. Please comment on the ticket or send me an e-mail.